KB

VOLUME ONE

The Machine

from *How Computers Work*

KedByte Technologies Private Limited

KEDBYTE

TECHNOLOGIES PRIVATE LIMITED

HOW COMPUTERS WORK

From a Grain of Sand to Artificial Intelligence

VOLUME ONE

The Machine

Rock pulled from the ground, electricity, silicon, transistors, logic, memory, numbers, screens, sound, the processor, storage, and the long road that produced all of it.

WRITTEN BY

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

Parts A, B and C • Chapters 1 to 14 • First Edition

HOW COMPUTERS WORK

Volume One — The Machine

Parts A, B and C, chapters 1 to 14

Author	Shikhar Singh
Designation	Founder & Chairman, KedByte Technologies Private Limited
Published by	KedByte Technologies Private Limited
Imprint	KedByte Press
Edition	First Edition
Volume	1 of 5
Complete work	Nine parts, fifty-three chapters

About this volume. This is Volume 1 of five. It contains Parts A, B and C of *How Computers Work*, chapters 1 to 14. Chapter numbers are those of the complete work, so a cross-reference to any chapter means the same chapter whether you are reading a single volume or the complete edition. References to chapters outside this volume point to the other volumes in the set.

Accuracy and currency. This book describes technology that changes. Specifications, prices, version numbers and industry figures were checked at the time of writing and are dated in the text wherever they are liable to change. Where experts disagree, the book says so rather than choosing a side.

Disclaimer. The information in this book is provided on an "as is" basis, for education. Neither the author nor KedByte Technologies Private Limited shall have any liability to any person or entity with respect to any loss or damage caused, or alleged to have been caused, directly or indirectly by the information contained in this book.

BEFORE YOU START

How to read this book

Every idea in this book is explained twice. First in plain language, with an everyday comparison and a worked example, so that a beginner can follow it. Then again in the correct technical vocabulary, with real units, real specifications and the exact figures an engineer would quote. The two halves are labelled, so you can read the plain pass end to end and then come back for the engineering.

Block	What it is for
PLAIN in simple words	The idea for a complete beginner. No jargon at all.
PLAIN a picture in your head	One everyday comparison, and then a line telling you exactly where that comparison stops being true.
PLAIN a worked example	Concrete numbers, worked step by step.
PLAIN what is really happening inside	The mechanism, still in easy words, but with nothing hand-waved.
TECHNICAL the engineer's version	Correct professional vocabulary, exact units, real specifications, real figures, and the commands that observe it.
WORDS remember these	Each term twice: the plain meaning, then the technical meaning.

1. Almost everything is written as short numbered lines, one idea per line, so nothing hides inside a paragraph.
2. *The honest version* marks the places where a simple explanation is not strictly true. The accurate model follows immediately.
3. The book says whether a thing is a **standard**, a **convention**, or one product's **implementation detail**.
4. Anything that goes stale is dated. Check it again before relying on it.
5. Every chapter ends with *Common wrong ideas* and a twenty-line summary. Short of time, read those two first.

The five volumes

1. **The Machine** — chapters 1 to 14
2. **Software** — chapters 15 to 22
3. **Networks** — chapters 23 to 34

4. **Building and Shipping Software** — chapters 35 to 44
5. **Intelligence, Games and Reference** — chapters 45 to 53

Contents

KB

PART A

The Physical World

Rock, electricity, silicon, transistors, logic and memory. Everything a computer is made of, before any software exists.

1

CHAPTER

KB

The Whole Machine in One Look

1.0 What this chapter gives you

1. You will be able to say what a computer actually is, with nothing magical left in it.
2. You will name the five parts every computer has ever had, and find them in a 1945 machine and in a watch.
3. You will explain the one idea that made computers general-purpose, and name the 1945 document that wrote it down.
4. You will draw the layer cake from rock to artificial intelligence, and say what each layer hides.
5. You will trace one key press to one letter on screen, through about thirty steps.
6. You will feel the real speed gaps inside a machine, because we will stretch nanoseconds into human time.
7. You will stop being confused by KB versus KiB, by MB/s versus Mb/s, and by a 1 TB drive that reports 931 GB.
8. You will know where hardware stops and software starts, and why that border keeps moving.
9. You will have a map of all 53 chapters and a method for reading them.

1.1 What a computer actually is

1.1.1 PLAIN – in simple words

1. A computer is a machine that moves and changes patterns of on and off.
2. “On” and “off” mean a wire carries a higher or a lower voltage. Voltage is electrical push.
3. We write on as 1 and off as 0. One 1 or 0 is a bit.
4. The machine does not know whether a pattern is a photo, a song or a bank balance.
5. Meaning is given by us, not by the metal.

1.1.2 PLAIN – a picture in your head

1. Picture a long corridor with a million light switches on the wall.
2. Now imagine a rule-follower walking that corridor at enormous speed.
3. He carries one card: “if switch 7 is up and switch 8 is up, put switch 9 up”.
4. He never thinks and never decides. Run him fast enough and the pattern becomes a spreadsheet, a call, a game.
5. Where this comparison breaks: there is no walker. The rules are in the wiring, and huge numbers of switches settle at the same instant.

1.1.3 PLAIN – a worked example

1. Take four switches: S3, S2, S1, S0. We agree S3 is worth 8, S2 is worth 4, S1 is worth 2, S0 is worth 1.

```
S3 S2 S1 S0   value
 0  0  0  0     0
```

```
0  0  1  1     3
0  1  0  1     5
1  0  0  1     9
1  1  1  1    15
```

2. Now we want 5 plus 3. One group is set to 0101, another to 0011, and both feed a small adding circuit.
3. That circuit has no memory and no cleverness. Its output settles to 1000, which is 8.
4. Nobody told the circuit this was arithmetic. Its wiring forced the result.

1.1.4 PLAIN - what is really happening inside

1. Inside a chip, a bit is a voltage held on a tiny wire by a silicon switch called a transistor.
2. A transistor is a tap: a small voltage on one leg controls current between the other two.
3. A few transistors make a gate, a circuit applying one logical rule. Millions of gates make an adder or a memory cell.
4. Voltages take time to settle, because every wire has capacitance, the tendency to store charge and resist sudden change.
5. So the machine has a clock, a square wave ticking billions of times a second. On each tick, settled results are captured.
6. The honest version: a big 2026 chip is not one clock. It has many clock domains at different rates, and some blocks are deliberately clockless. One global heartbeat is a teaching simplification.

1.1.5 TECHNICAL - the engineer's version

1. Mainstream digital logic is CMOS, complementary metal-oxide-semiconductor, using paired n-channel and p-channel field-effect transistors.
2. A logic level is a voltage band, not a point. Above VIH is 1, below VIL is 0, and the gap between them is forbidden in steady state.

Rail	Typical voltage	Where it appears
CPU core, 2026	0.6 to 1.2 V	logic on the die
DDR5 VDD	1.1 V	main memory
LVCMOS 3.3 V, VIH	2.0 V minimum	input read as 1
LVCMOS 3.3 V, VIL	0.8 V maximum	input read as 0

3. Claude Shannon's 1937 master's thesis at MIT showed that relay switching circuits implement Boolean algebra. That is the formal bridge from wires to logic.
4. Alan Turing's 1936 paper "On Computable Numbers" defined what a general computing machine can compute, before any such machine existed.
5. Observation tools: `lscpu` and `dmidecode` report clocks and memory type; `turbostat` and `powermetrics` report frequency and power live.

1.1.6 WORDS - remember these

1. Bit — one on-or-off — a binary digit, value 0 or 1, the smallest unit of information.
2. Voltage — electrical push — potential difference in volts, the physical carrier of a logic level.
3. Transistor — an electrical tap — a semiconductor switch, in CMOS a field-effect transistor.
4. Clock — the machine's heartbeat — a periodic square wave defining when state elements sample inputs.

1.2 The five parts of every computer ever built

1.2.1 PLAIN – in simple words

1. Every computer, from 1945 to the one on your wrist, has the same five parts.
2. Input: how the outside world gets patterns into the machine.
3. Memory: fast temporary space holding what the machine is working on right now.
4. Storage: slow permanent space that keeps things when power is off.
5. Processing: the part that changes patterns by rules, called the processor or CPU.
6. Output: how patterns get back out where you can sense them.
7. There is no sixth part. Memory forgets when power stops; storage remembers. Confusing those two causes more beginner errors than anything else.

1.2.2 PLAIN – a picture in your head

1. Think of a kitchen. Shopping bags coming through the door are input.
2. The countertop is memory: small, close to hand, cleared at the end of the night.
3. The fridge and cupboard are storage: bigger, slower to reach, still full tomorrow.
4. The cook is processing, the only part that changes anything. The plate carried to the table is output.
5. Where this comparison breaks: a cook holds an idea in his head while walking to the fridge. A processor cannot. It has a few tiny slots called registers, and everything else must come back through memory.

1.2.3 PLAIN – a worked example

1. The five parts in a 1945 machine and in an ordinary desktop.

Part	ENIAC, 1945	Desktop, 2026
Input	punched cards, switches	keyboard, mouse
Memory	20 accumulators	16 to 64 GB of RAM
Storage	punched cards, paper	1 to 4 TB NVMe SSD
Processing	tube arithmetic units	multi-core CPU
Output	punched cards, lamps	monitor, speakers

2. The same five parts in a phone and in a smartwatch.

Part	Phone	Smartwatch
Input	touch, mics, cameras	touch, crown, sensors
Memory	6 to 16 GB RAM	small, not published
Storage	128 GB to 1 TB flash	64 GB flash
Processing	CPU, GPU, NPU on a die	low-power SiP
Output	OLED screen, speaker	OLED screen, buzzer

1.2.4 PLAIN – what is really happening inside

1. The processor never works directly on storage. Storage is far too slow and is not wired for it.
2. So the life of any data is: storage to memory, memory to processor, processor to memory, memory to storage.
3. Output runs in reverse: the processor writes numbers into a region of memory, and a device reads that region and makes light or sound.
4. The processor never touches a pixel. It writes numbers. Something else turns numbers into light.

1.2.5 TECHNICAL - the engineer's version

1. ENIAC was built at the Moore School of Electrical Engineering, University of Pennsylvania, by J. Presper Eckert and John Mauchly. It first ran productively on 10 December 1945 and was announced publicly on 14 February 1946.
2. ENIAC used about 18,000 vacuum tubes, weighed more than 30 short tons, drew about 150 kW, and did roughly 5,000 additions per second. It was programmed by plugboard wiring, which could take days.
3. The Apple Watch Series 10, announced September 2024, uses the S10 SiP with a 64-bit dual-core processor, 64 GB of storage, and a 416 by 496 pixel display in the 46 mm size.
4. Observation tools: `lsblk` and `diskutil list` show storage; `free -h` and `vm_stat` show memory; `lspci` and `lsusb` show what sits on the buses.

1.2.6 WORDS - remember these

1. Input — how things get in — any device converting an external event into machine-readable data.
2. Memory — the countertop — volatile working store, today DRAM, addressed byte by byte.
3. Storage — the cupboard — non-volatile store, today NAND flash or magnetic disk, addressed in blocks.
4. Processing — the cook — the CPU and co-processors that fetch, decode and execute instructions.
5. Output — how things get out — any device turning data into a signal a person or machine can sense.
6. Bus — a bundle of shared wires — a defined interface carrying address, data and control signals.

1.3 The one big idea: instructions and data are the same stuff

1.3.1 PLAIN - in simple words

1. The list of steps a machine should follow is itself just a pattern of bits.
2. So it can live in the same memory, in the same kind of slots, as the numbers being worked on.
3. Before this idea the steps lived in the wiring. To change the job you rewired the machine.
4. After it, you load a different pattern into memory. That takes a moment, not a week. This is the stored-program idea.
5. The same idea is why a bad pattern arriving from outside can sometimes be run as instructions.
6. One idea gives you both general-purpose computing and computer viruses. They are the same coin.

1.3.2 PLAIN - a picture in your head

1. Imagine a kitchen where recipe cards sit in the same drawer as shopping lists, on identical cards in the same handwriting.
2. A card is a recipe only because the cook happened to pick it up and follow it.
3. Where this comparison breaks: a human cook would notice that a shopping list is not a recipe and stop.
4. A processor does not notice. It will read your holiday photo as instructions and execute it, producing noise or a crash.

1.3.3 PLAIN - a worked example

1. Consider one byte with the pattern 01001000. In base 16 that is 48. In decimal it is 72.

```
byte 01001000 (hex 48, decimal 72)

read as a number      -> 72
read as ASCII text    -> the letter H
read as x86 machine   -> part of a REX prefix
read as a pixel value -> a mid-grey level
read as an audio byte -> one sample, fairly loud
```

2. The bits never change. Only the interpretation changes.
3. The program counter is the register holding "where I am reading instructions from".

4. If it points at that spot, the processor will try to execute those bytes as an instruction, and usually crash within microseconds.
5. Nothing in the memory chip distinguishes the two cases. The only difference is which register pointed at the address.

1.3.4 PLAIN - what is really happening inside

1. The processor keeps a program counter holding the address of the next instruction.
2. Each cycle it reads the bits there, works out what they mean, does it, and advances the counter. That loop is fetch, decode, execute.
3. Because instructions are ordinary bytes, an attacker who can write memory and redirect the counter runs code of their choosing.
4. So operating systems mark memory pages writable or executable and try never to allow both at once.
5. The honest version: "never both at once" is the goal, not the reality. Just-in-time compilers must briefly have both, and the whole field of exploit mitigation lives in that gap.

1.3.5 TECHNICAL - the engineer's version

1. The canonical statement is the "First Draft of a Report on the EDVAC", circulated 30 June 1945 under John von Neumann's name alone.
2. It drew on work by Eckert and Mauchly, and credit for it is still disputed by historians.
3. The structure it describes, one memory holding both instructions and data reached over one path, is the **von Neumann architecture**.

Machine	First stored program	People
ENIAC, converted	April 1948	Clippinger, von Neumann
Manchester Baby	21 June 1948	Kilburn, Williams
EDSAC, Cambridge	6 May 1949	Maurice Wilkes
EDVAC	1951 in service	Eckert, Mauchly

4. The hardware defence against executing data is the no-execute page bit: AMD shipped NX from 2003, Intel calls it XD, Arm calls it XN. The operating system policy built on it is W xor X.
5. Observation tools: `objdump -d` disassembles bytes as instructions; `xxd` shows the same bytes as hex and text together; `/proc/<pid>/maps` shows which regions are executable.

1.3.6 WORDS - remember these

1. Stored program — the steps live in memory like any other data — instructions encoded in the same address space as operands.
2. Program counter — a bookmark — the register holding the address of the next instruction.
3. von Neumann architecture — one memory for everything — a single address space and data path for instructions and data.
4. Just-in-time compilation — building instructions while running — generating native code at run time and jumping into it.
5. NX bit — a "do not run this" mark — a page-table attribute preventing instruction fetch from a page.

1.4 The layer cake

1.4.1 PLAIN - in simple words

1. At the bottom is rock: sand, which is silicon dioxide, purified into silicon crystal. Above it is electricity, charge pushed along wires.
2. Above electricity are transistors, tiny switches made in the silicon. Above them are gates, small groups of switches applying one rule each.

3. Above gates is the processor, millions of gates arranged to fetch and obey instructions, and above that is machine code, the bit patterns it obeys.
4. Above machine code is the operating system, which shares the machine among other programs, and then the programs you use.
5. Above the programs come networks, then the cloud, which is other people's machines rented by the hour, then artificial intelligence services.
6. Hiding the layer below is called abstraction. Its promise is that you can use one layer without knowing the one under it.
7. The promise is never fully kept. Details leak upward, and most hard bugs live in a leak.

1.4.2 PLAIN - a picture in your head

1. Think of a tall building where each floor was built by a different team.
2. The team on floor six is told only that the floor holds their weight and there is a lift.
3. But on a windy day floor six sways, and on a hot day the lift is slow. Nobody promised that. Lower floors are leaking through.
4. Where this comparison breaks: floors are physically separate; computer layers are not.
5. There is no wall between the operating system and the processor. The separation is an agreement in documents, enforced only partly by hardware.

1.4.3 PLAIN - a worked example

```
AI services        <- text in, text out
cloud              <- machines by the hour
networks           <- messages to elsewhere
programs           <- what you actually use
operating system   <- sharing, files, safety
machine code       <- what the CPU obeys
CPU                <- fetch, decode, execute
logic gates        <- one rule per circuit
transistors        <- one switch each
electricity        <- charge that moves
silicon / rock     <- purified sand
```

1. Now the honest part: what each layer hides, and what leaks through anyway.

Layer	Hides	Leaks anyway
Transistors	atoms, physics	heat, wear, errors
Gates	voltages, timing	propagation delay
CPU	gate wiring	cache misses, branches
Machine code	circuits	which CPU family is needed
Operating system	hardware detail	file locks, page faults
Programs	the OS	memory limits, crashes
Networks	distance	latency, packet loss
Cloud	the building	region choice, outages
AI services	the model	token limits, wrong answers

2. When a web page in India loads slowly, the network layer is leaking the speed of light through five layers of abstraction into a person's experience.

1.4.4 PLAIN - what is really happening inside

1. Each boundary between layers is a contract listing what you may ask for and what you get back.
2. A contract that says what happens and never how is a clean abstraction. A leak happens when the "how" shows through timing, limits or failure.

3. Timing leaks are the worst, because the contract is still honoured. You get the right answer, just late. A cache miss is exactly that.
4. The honest version: layers are not a neat pile. A graphics driver reaches almost straight to hardware.
5. A hypervisor sits under an operating system that believes it is on bare metal. The cake is a teaching device; the real thing is a tangle.

1.4.5 TECHNICAL - the engineer's version

1. The processor-software contract is the instruction set architecture, ISA: x86-64 from Intel and AMD, Arm A-profile at Armv9, and RISC-V, an open specification maintained by RISC-V International since 2015.
2. Below the ISA sits microarchitecture: pipelines, caches, branch predictors, out-of-order execution. It is invisible in the contract and visible on the stopwatch.
3. Joel Spolsky named the general phenomenon in his 2002 essay "The Law of Leaky Abstractions": all non-trivial abstractions, to some degree, are leaky.
4. The most expensive proof arrived in January 2018 with Spectre and Meltdown, published by teams including Google Project Zero and Graz University of Technology.
5. Speculative execution, purely a microarchitectural optimization, leaked memory across the security boundary the ISA had promised.
6. To keep the three kinds of claim apart: x86-64 instruction encoding and POSIX system call names are **standards**; the layer diagram above is a **convention**; the L3 cache size on one chip is an **implementation detail**.
7. Observation tools: `perf stat` counts cache misses and branch mispredictions; `strace` and `dtruss` log system calls; `mtr` shows where network latency accumulates.

1.4.6 WORDS - remember these

1. Abstraction — hiding the mess below — an interface exposing behaviour while concealing implementation.
2. Leaky abstraction — the mess showing through — an interface whose implementation is observable via timing, limits or failure modes.
3. Instruction set architecture — the deal between chip and software — the specified instructions, registers and memory model.
4. Protocol — an agreed way to talk — a specification of message formats and exchange rules between systems.

1.5 A journey in one page: you press the letter A

1.5.1 PLAIN - in simple words

1. You press A. The letter appears. It feels instant and it feels like one event.
2. It is not one event. It is about thirty handovers across nine layers.
3. A physical push becomes an electrical contact, becomes a number, becomes a message, becomes a shape, becomes light.
4. This section is the trailer. Chapter 21 does the whole journey properly, with real timings and real code paths.

1.5.2 PLAIN - a picture in your head

1. Think of a relay race with thirty runners, each carrying a baton a short distance.
2. Runner one carries "a key moved". Runner ten carries "key number 4 went down". Runner twenty carries "the character A". Runner thirty carries "these pixels are white".
3. Where this comparison breaks: several runners are actually queued behind other work.
4. And some handovers are not a pass at all. They are a note left in a shared box that the next runner checks periodically.

5. That periodic checking, called polling, is where a surprising amount of the delay comes from.

1.5.3 PLAIN – a worked example

1. You move your finger. Layer: physics.
2. The key cap pushes a switch and two metal contacts touch. Layer: mechanics.
3. The contacts bounce, opening and closing several times in a few milliseconds. Layer: physics.
4. A small processor in the keyboard ignores the bounce by waiting. This is debouncing. Layer: firmware.
5. That processor scans a grid of rows and columns thousands of times a second to find which key is down. Layer: firmware.
6. It works out a position number for the key, called a scan code. Layer: firmware.
7. It builds a small report listing which keys are held. Layer: USB HID protocol.
8. The report waits until the host asks for it, on a fixed schedule. Layer: USB bus.
9. The host controller chip receives the report as electrical pulses. Layer: hardware.
10. That chip writes the bytes into main memory itself, without the CPU. This is direct memory access. Layer: hardware.
11. The chip raises an interrupt, a signal meaning “stop and look”. Layer: hardware.
12. The CPU finishes its current instruction, saves its place, and jumps to a fixed address. Layer: CPU.
13. That address holds the interrupt handler inside the operating system kernel. Layer: kernel.
14. The USB driver reads the report and turns the scan code into a kernel key code. Layer: driver.
15. The input subsystem timestamps the event and puts it in a queue. Layer: kernel.
16. The window system wakes, reads the queue, and asks which window has focus. Layer: display server.
17. The keyboard layout is applied. The same key is A on a US layout and Q on a French AZERTY layout. Layer: keymap.
18. Modifier state is applied: shift gives capital A, control gives a control character. Layer: keymap.
19. The event is delivered to the focused application as a message. Layer: operating system.
20. The application’s event loop, a loop that waits for messages forever, picks it up. Layer: program.
21. The text widget inserts the character into its internal buffer. Layer: program.
22. The character is stored as a number: Unicode code point 65, encoded in UTF-8 as the single byte 0x41. Layer: text encoding.
23. The program marks the text area as needing redraw. Layer: program.
24. The text shaping engine picks the font, finds the glyph for A, and works out its position. Layer: text stack.
25. The rasterizer turns the glyph outline, a set of curves, into a grid of coverage values. Layer: font engine.
26. Those values become pixel colours blended with the background. This is anti-aliasing. Layer: graphics.
27. The pixels are written into a block of memory called a framebuffer, often by the GPU. Layer: GPU.
28. The compositor combines every window’s framebuffer into one final image. Layer: display server.
29. The display controller reads that image out line by line and sends it over the cable. Layer: display hardware.
30. The panel’s driver chips set the voltage on each subpixel. Light changes. You see A. Layer: physics.

1.5.4 PLAIN – what is really happening inside

1. Time budget matters more than step count. Most of those thirty steps cost far less than a microsecond.
2. The expensive parts are the waits, not the work.
3. Waiting for the keyboard to be polled: up to 8 milliseconds on an old 125 Hz USB device, about 1 millisecond on a 1000 Hz gaming keyboard.
4. So the honest total, finger to photon, is usually 20 to 60 milliseconds, of which perhaps 50 to 200 microseconds is CPU work.
5. That is a general truth worth carrying: the machine is mostly waiting, and speeding up the work rarely helps as much as removing a wait.

1.5.5 TECHNICAL - the engineer's version

1. Keyboards speak USB HID, Human Interface Device, defined in the USB HID class specification version 1.11 from 2001. A boot-protocol keyboard report is 8 bytes: modifier byte, reserved byte, six key codes.
2. Scan code, key code, keysym and character are four distinct things, and confusing them is a classic bug: a hardware position, a kernel-normalized position, a layout-resolved symbol, and a Unicode value.

Stage	Typical time
Debounce in keyboard	1 to 5 ms
USB poll wait	1 to 8 ms
Kernel and driver	under 100 us
App and compositor	1 to 10 ms
Wait for vsync	8 to 17 ms
Panel response	1 to 20 ms

3. Observation tools: `evtest` and `libinput debug-events` show raw input events; `xev` shows keysyms; `showkey -s` prints scan codes.

1.5.6 WORDS - remember these

1. Debounce — ignoring the rattle of a switch — filtering multiple contact transitions from one mechanical actuation.
2. Polling — checking regularly instead of being told — the host querying a device on a fixed interval.
3. Interrupt — a tap on the shoulder — a hardware signal making the CPU suspend execution and run a handler.
4. Direct memory access — the device writes to memory itself — a controller moving data to RAM without CPU involvement.
5. Framebuffer — the picture waiting to be shown — a memory region holding pixel values for one display image.

1.6 What "fast" and "big" mean here

1.6.1 PLAIN - in simple words

1. Inside a computer there is no single speed. There is a ladder, and the rungs are enormously far apart.
2. The processor is fast. Everything it talks to is slower, and each step outward is slower again.
3. So we do one trick. We pretend one nanosecond, a billionth of a second, lasts one whole second.
4. On that scale a coffee break becomes a lifetime, and the shape of the ladder becomes unforgettable.

1.6.2 PLAIN - a picture in your head

1. You are at a desk and you need one fact. If it is already in your head, one second.
2. On the desk, four seconds. On the shelf behind you, half a minute.
3. In the filing room down the corridor, nearly two minutes. Ordered from a warehouse, the day after tomorrow.
4. In a paper archive across the country, four months. Written to someone abroad and waiting for a reply, six years.
5. Where this comparison breaks: a person waiting six years does nothing else. A processor does not wait; it switches to other work and comes back.

1.6.3 PLAIN – a worked example

1. Here is the ladder, with real figures and the same figures stretched so 1 nanosecond becomes 1 second.

Step	Real time	If 1 ns were 1 second
CPU cycle at 4 GHz	0.25 ns	a quarter second
L1 cache hit	about 1 ns	1 second
L2 cache hit	about 4 ns	4 seconds
L3 cache hit	about 30 ns	half a minute
Main memory, RAM	80 to 100 ns	1.5 minutes
NVMe SSD read	50 to 100 us	14 to 28 hours
Hard disk seek	8 to 10 ms	3 to 4 months
India to US, return	about 200 ms	6.3 years

```
cache      -> seconds
memory     -> a couple of minutes
flash SSD  -> a day
hard disk  -> a season
the world  -> most of a degree course
```

2. So one extra round trip to another country can waste more time than millions of memory accesses.

1.6.4 PLAIN – what is really happening inside

1. Two limits build this ladder. The first is distance: signals cannot outrun light.
2. In copper or fibre they travel slower still, about two thirds of light speed.
3. In one nanosecond light covers about 30 centimetres in vacuum and about 20 centimetres in glass fibre.
4. The second limit is mechanism. A cache is a small grid of transistors that answers immediately.
5. Main memory is a huge grid of tiny capacitors that must be selected, sensed, amplified and refreshed, which costs tens of nanoseconds however much you spend.
6. A hard disk must move an arm and wait for a platter to spin, which is a mechanical event.
7. The honest version: every number in that table is an average hiding a wide spread. An SSD read under heavy write load can be ten times its idle figure.
8. Treat latency as a distribution with a long tail, never as a single constant.

1.6.5 TECHNICAL – the engineer's version

1. Quoting a latency ladder is a **convention**, not a specification. Peter Norvig published an early version in his 2001 essay "Teach Yourself Programming in Ten Years".
2. Jeff Dean's version, circulated in Google talks from about 2009, made the practice standard.
3. Cache latencies are specified in cycles, not nanoseconds, because they scale with clock: L1 data 4 to 5 cycles, L2 12 to 20 cycles, L3 30 to 60 cycles.
4. L3 varies most, since it depends on core count and on-die topology. A large server part with a mesh interconnect can exceed 40 nanoseconds; a small desktop part can be near 10.
5. NVMe SSDs are quoted as 4 KiB random read latency: consumer TLC drives land near 50 to 100 microseconds, while Intel Optane, discontinued in 2022, reached about 10.
6. Round-trip time across an ocean has a hard floor set by fibre. Mumbai to northern Virginia is roughly 12,900 km great-circle, so about 25,800 km return.
7. At 200,000 km/s that is about 129 milliseconds before a single router touches the packet.
8. Observation tools: `perf stat -e cache-misses`, `lmbench` with its `lat_mem_rd` test, `fio` with `--iodepth=1` for true storage latency, and `ping` or `mtr` for network round trip.

1.6.6 WORDS - remember these

1. Latency — how long before you get an answer — the delay between issuing a request and its first result.
2. Cache — a small fast copy kept close — an SRAM store holding recently or likely used lines from a slower level.
3. Round-trip time — go and come back — the time for a packet to reach a host and its reply to return, in milliseconds.
4. Tail latency — the slow few — high percentile response times such as p99, which dominate what users feel.

1.7 Units done properly

1.7.1 PLAIN - in simple words

1. A bit is one on-or-off. A byte is eight bits stuck together.
2. Capital B means bytes. Lowercase b means bits. That one letter changes the number by eight.
3. Sellers of storage use round decimal thousands, so one kilobyte is 1,000 bytes.
4. Operating systems traditionally use binary thousands, so one unit is 1,024 bytes.
5. Both are defensible and neither is lying, but they disagree by a margin that grows with size. That is the whole reason a 1 TB drive appears as 931 GB.
6. Hertz means times per second. A 3 GHz processor ticks three billion times each second, but more ticks does not automatically mean more work done.

1.7.2 PLAIN - a picture in your head

1. Think of buying rice. The shop sells a bag holding 1,000 grams. Your kitchen jar holds 1,024 grams.
2. You buy a thousand bags. The shop says 1,000 bags. Your jars say you filled 976.6 jars.
3. Where this comparison breaks: rice pours freely, and the difference there is a pure counting choice.
4. With computers the 1,024 is not arbitrary. Memory addressing is binary, so memory really does come in powers of two.
5. Storage has no such constraint, which is exactly why the two industries drifted apart.

1.7.3 PLAIN - a worked example

```
1 TB drive as sold:
  1,000,000,000,000 bytes
/ 1024 ->     976,562,500 KiB
/ 1024 ->         953,674 MiB
/ 1024 ->           931.32 GiB
Windows prints "931 GB". Nothing is missing.
```

```
100 Mbps line:
  100,000,000 bits per second
/ 8     = 12,500,000 bytes per second
        = 12.5 MB/s on paper
x 0.949   Ethernet + IP + TCP overhead
        = about 11.9 MB/s of real file
```

1. Those two sums explain most unit confusion. A 100 Mbps line downloading a 1.2 GB file takes about 100 seconds, not 12 and not 1,200.
2. Clock speed works the same way. A 3 GHz processor has a cycle time of 1 divided by 3,000,000,000, which is 0.333 nanoseconds.

1.7.4 PLAIN - what is really happening inside

1. Memory chips are addressed with binary numbers, so 30 address lines give exactly 2 to the power 30 locations, which is 1,073,741,824.

2. Not every byte on a network carries your file. Each packet carries headers and the link adds framing, so useful throughput is always below line rate.
3. The honest version: a byte has not always been eight bits. Early machines used 6, 7 and 9-bit bytes, which is why standards documents say **octet** when they mean exactly eight. Eight became universal by convention through the 1970s and only later entered standards.

1.7.5 TECHNICAL - the engineer's version

1. Decimal prefixes are SI, maintained by the BIPM: k is 10^3, M is 10^6, G is 10^9, T is 10^12.
2. Binary prefixes are IEC: Ki is 2^10, Mi is 2^20, Gi is 2^30, Ti is 2^40. They were published as IEC 60027-2 Amendment 2 in January 1999 and adopted as IEEE 1541-2002, which the IEEE Standards Association raised to a full-use standard on 19 March 2005.

Unit	Bytes	Difference
1 kB / 1 KiB	1,000 / 1,024	2.4 percent
1 MB / 1 MiB	1e6 / 1,048,576	4.9 percent
1 GB / 1 GiB	1e9 / 1.0737e9	7.4 percent
1 TB / 1 TiB	1e12 / 1.0995e12	10.0 percent

3. Practice differs by product, and this is an **implementation detail** with a date: macOS has reported decimal GB since Mac OS X 10.6 Snow Leopard in 2009, while Windows File Explorer still shows binary values labelled GB as of 2026. On Linux `ls -h` is binary and `ls --si` is decimal.
4. Ethernet efficiency at a full 1500-byte MTU with IPv4 and TCP: 1460 bytes of payload inside 1538 bytes on the wire, counting the 14-byte header, 4-byte FCS, 8-byte preamble and 12-byte interframe gap. That is 94.9 percent, before any retransmission.
5. Observation tools: `lsblk -b` shows exact byte counts; `df -B1` avoids rounding; `iperf3` measures true throughput; `dd` with `oflag=direct` measures raw device rate.

1.7.6 WORDS - remember these

1. Byte — eight bits — a group of 8 bits, symbol B; standards say octet when precision matters.
2. Kibibyte — the binary thousand — 2^10 = 1,024 bytes, symbol KiB, per IEC 60027-2.
3. Kilobyte — the decimal thousand — 10^3 = 1,000 bytes, symbol kB, per SI.
4. Mbps — millions of bits per second — a line rate; divide by 8 for bytes, then subtract protocol overhead.
5. Hertz — times per second — cycles per second, symbol Hz; cycle time is the reciprocal.
6. IPC — how much work per tick — instructions retired per clock cycle, the other half of performance.

1.8 Hardware, software and firmware

1.8.1 PLAIN - in simple words

1. Hardware is the part you could drop on your foot: metal, silicon, plastic, wires.
2. Software is the pattern of bits telling the hardware what to do. It weighs nothing.
3. Firmware is software that lives inside a piece of hardware and ships with it, so most people never see it.
4. The boundary sounds obvious and is not. It moves every few years, in both directions.
5. Jobs move into hardware when they are stable and popular, and out of hardware when they must change often.

1.8.2 PLAIN - a picture in your head

1. Think of a road. The tarmac is hardware. The traffic rules are software.
2. Firmware is the painted lines and fixed signs: part of the road, really instructions, repainted occasionally.

3. Where this comparison breaks: firmware can be replaced remotely and instantly, and a bad replacement can make the road unusable.
4. That failure has a name, bricking: a failed firmware update leaving a device unable to start at all.

1.8.3 PLAIN - a worked example

1. Modems were once a box of dedicated signal chips. From the late 1990s the winmodem moved that work onto the main processor, because it was cheaper.
2. Encryption moved the other way: AES was ordinary software until Intel added AES-NI instructions in 2010.

Job	Was	Became
Modem signal processing	hardware chip	host software
Video decode	software	fixed-function block
AES encryption	software	CPU instruction, 2010
Neural network math	software	NPU, from 2017

1.8.4 PLAIN - what is really happening inside

1. Even the processor's own instruction list is not pure hardware on every design.
2. On x86, complicated instructions are broken into simpler internal steps by a small program stored on the chip. That program is microcode.
3. Microcode can be updated, which is how makers fix processor bugs without recalling chips.
4. Going the other way, a field-programmable gate array is a chip whose wiring you rewrite: you write something like a program and get real circuits.
5. The honest version: "hardware is fixed, software is changeable" is comfortable and wrong. The real question is how fast a thing can change and who is allowed to change it.

1.8.5 TECHNICAL - the engineer's version

1. UEFI replaced the legacy PC BIOS, which dates from the IBM PC of 1981. The UEFI specification is maintained by the UEFI Forum and reached version 2.10 in 2022.
2. Microcode updates load at boot from firmware or from the operating system. On Linux they arrive as `intel-ucode` or `amd-ucode` packages and appear in `dmesg`.
3. FPGA designs are written in Verilog or VHDL and compiled to a bitstream. An ASIC is the same design frozen into silicon, cheaper per unit and impossible to change.
4. Observation tools: `fwupdmgr get-devices` lists updatable firmware on Linux; `system_profiler SPHardwareDataType` on macOS shows boot ROM version; `nvme id-ctrl` reports SSD firmware revision.

1.8.6 WORDS - remember these

1. Firmware — software shipped inside a device — code in non-volatile memory on a peripheral or platform, updated rarely.
2. Microcode — the processor's inner script — on-die translation of architectural instructions into internal micro-operations.
3. FPGA — a chip you can rewire — a field-programmable gate array configured by a bitstream into arbitrary logic.
4. Bricking — an update that kills the device — a failed firmware write leaving hardware unable to boot or recover.

1.9 How to read this book

1.9.1 PLAIN - in simple words

1. This book is written to be read twice, and the second reading is built into the structure.
2. Every section has six blocks: four marked PLAIN, one marked TECHNICAL, one word list.
3. On your first pass read only the PLAIN blocks, straight through, Chapter 1 to Chapter 53.
4. On your second pass go back for the TECHNICAL blocks. Now you have somewhere to put the numbers.
5. Do not read the technical half first. It is correct, it will not stick, and you will wrongly conclude you are bad at this.

1.9.2 PLAIN - a picture in your head

1. Think of learning a city. First you walk the main roads and learn how the districts sit next to each other.
2. The PLAIN blocks are the main roads. The TECHNICAL blocks are the house numbers.
3. Where this comparison breaks: a city stays still, and some technical detail here has a shelf life.
4. When you see a year attached to a fact, that is deliberate. It tells you when to check whether it is still true.

1.9.3 PLAIN - a worked example

1. Here is the whole book, one line per chapter.
2. PART A, THE PHYSICAL WORLD, chapters 1 to 6.
3. Chapter 1, The whole machine in one look.
4. Chapter 2, Electricity. Charge, voltage, current.
5. Chapter 3, From sand to silicon. Making a chip.
6. Chapter 4, The transistor. The switch under everything.
7. Chapter 5, Logic gates and arithmetic.
8. Chapter 6, How a machine remembers.
9. PART B, TURNING BITS INTO MEANING, chapters 7 to 9.
10. Chapter 7, Numbers and meaning. Binary, hex, text.
11. Chapter 8, Pixels and screens.
12. Chapter 9, Sound, magnets, speakers and calls.
13. PART C, THE MACHINE, chapters 10 to 14.
14. Chapter 10, Inside the CPU.
15. Chapter 11, The memory hierarchy.
16. Chapter 12, Storage. Disks, flash, wear.
17. Chapter 13, Buses, ports, drivers and firmware.
18. Chapter 14, A history of computing machines.
19. PART D, SOFTWARE, chapters 15 to 20.
20. Chapter 15, The birth of software.
21. Chapter 16, How a compiler works.
22. Chapter 17, Programming languages.
23. Chapter 18, Operating systems.
24. Chapter 19, The terminal and the shell.
25. Chapter 20, Files, extensions and the .exe.
26. PART E, SEEING AND SHOWING, chapters 21 to 22.
27. Chapter 21, From key press to pixel, in full.
28. Chapter 22, Graphics and the GPU.
29. PART F, NETWORKS, chapters 23 to 34.
30. Chapter 23, Networking from zero.
31. Chapter 24, IP, private vs public, NAT and CGNAT.
32. Chapter 25, DNS. Names into addresses.

33. Chapter 26, Packets and routing.
34. Chapter 27, Traceroute decoded, hop by hop.
35. Chapter 28, TCP and why a timeout is evidence.
36. Chapter 29, The diagnostic method.
37. Chapter 30, How the internet was built.
38. Chapter 31, Wireless. Wi-Fi, mobile data, radio.
39. Chapter 32, TLS and HTTPS.
40. Chapter 33, The OSI and TCP/IP models.
41. Chapter 34, Firewalls, VPNs, CDNs, IPv6 and localhost.
42. PART G, BUILDING AND SHIPPING, chapters 35 to 44.
43. Chapter 35, Web and app development.
44. Chapter 36, APIs. Machines calling machines.
45. Chapter 37, Git with no network.
46. Chapter 38, What a commit actually is.
47. Chapter 39, The index, push, merge and rebase.
48. Chapter 40, Your own git server, and what GitHub adds.
49. Chapter 41, CI/CD and what a run is.
50. Chapter 42, SSH, keys, tokens and scopes.
51. Chapter 43, Cloud, the Bezos mandate, AWS and Azure.
52. Chapter 44, Security. Threats and honest limits.
53. PART H, GAMES AND MACHINE INTELLIGENCE, chapters 45 to 51.
54. Chapter 45, Games. Loops, frames, input latency.
55. Chapter 46, Machine learning from scratch.
56. Chapter 47, Parameters, tokens and embeddings.
57. Chapter 48, The transformer.
58. Chapter 49, How a model is actually made.
59. Chapter 50, Benchmarks, evaluation, versions and dates.
60. Chapter 51, Serving, RAG, agents and limits.
61. PART I, REFERENCE, chapters 52 to 53.
62. Chapter 52, The engineer's reference.
63. Chapter 53, Master timeline and glossary.

1.9.4 PLAIN - what is really happening inside

1. One more thing about Parts F and G, the networking part and the building-and-shipping part.
2. They do not use invented examples. They use one real fault and one real work session, from the reader this book was written for.
3. A real home broadband connection in India, on macOS, that reached some things and silently could not reach others.
4. A real traceroute crossing a home router, a provider's private core, then a large company's backbone through Delhi, Mumbai and Pune, and then answering nothing.
5. A real set of git problems in the same session: pushes that failed halfway, a token missing a permission, a rebase that changed a commit's identity.
6. In particular you will learn to separate what an observation proves from what it merely suggests. That distinction is the whole of diagnosis.

1.9.5 TECHNICAL - the engineer's version

1. Where this book touches artificial intelligence, in Part H, it separates three kinds of statement, and you should demand the same separation elsewhere.
2. **Established fact**: the transformer architecture was published in the 2017 paper "Attention Is All You Need" by Vaswani and colleagues at Google. That is checkable.
3. **Active research**: how much of a model's ability comes from scale versus from data quality is still argued, with published results on both sides.
4. **Marketing claim**: any statement that a model reasons, understands, or is near human level is a claim about words with no agreed test behind it. A benchmark score is evidence about the benchmark first and about the world second.
5. Dates are attached to volatile facts deliberately. Anything about a current product, model or price should be re-checked; this chapter was written in 2026.

1.9.6 WORDS - remember these

1. PLAIN block — the everyday half — beginner-level explanation with no jargon, safe to read in isolation.
2. TECHNICAL block — the engineer's half — precise terms, units, specifications and observation commands.
3. Standard — written down and agreed — behaviour defined in a published specification such as an RFC or an IEC document.
4. Convention — everyone just does it — widespread practice with no governing specification.
5. Implementation detail — true of this one product — behaviour of a specific version that may change without notice.

1.98 Common wrong ideas

1. Wrong: a computer understands what it is doing. Right: it combines bits by fixed rules, and all meaning is assigned by people outside the machine.
2. Wrong: memory and storage are two words for one thing. Right: memory is fast and forgets without power; storage is slow and remembers.
3. Wrong: more gigahertz always means a faster computer. Right: work done is instructions per cycle times frequency, and a stalled processor retires almost nothing at any clock speed.
4. Wrong: a 1 TB drive showing 931 GB is missing space. Right: a trillion divided by 1024 three times is 931.32, and the operating system is labelling binary units with a decimal name.
5. Wrong: a 100 Mbps line downloads at 100 MB/s. Right: divide by 8 for bytes and subtract protocol overhead, giving about 11.9 MB/s.
6. Wrong: hardware is fixed and software is changeable. Right: firmware and microcode are updatable and reconfigurable chips exist, so the real question is how fast a thing can change and who may change it.
7. Wrong: the delay you feel when typing is the computer thinking. Right: most of it is waiting for a polling interval and a screen refresh, not computation.
8. Wrong: a slow reply from a distant server means bad engineering. Right: a round trip from India to the United States has a physical floor near 130 milliseconds, because light in fibre is not instantaneous.

1.99 Chapter summary in 20 lines

1. A computer stores, moves and combines patterns of on and off, and does nothing else.
2. A bit is one on-or-off, and meaning is assigned by people, never by the metal.
3. Every computer has five parts: input, memory, storage, processing, output.
4. ENIAC in 1945, a desktop, a phone and a watch differ in size, not in those roles.
5. Memory is the countertop and forgets without power. Storage is the cupboard and remembers.
6. The one big idea is that instructions are data, held in the same memory as the numbers.

7. It was written down in the First Draft of a Report on the EDVAC, 30 June 1945.
8. That made machines general-purpose and made malicious code possible, at the same stroke.
9. The Manchester Baby ran the first stored program on 21 June 1948, and EDSAC followed on 6 May 1949.
10. The machine is built in layers, from silicon up to artificial intelligence services.
11. Each layer hides the mess below it, and that hiding is called abstraction.
12. No abstraction is watertight. Timing, limits and failures leak upward.
13. One key press becoming one letter takes about thirty handovers across nine layers.
14. Almost all of that delay is waiting for a poll and a screen refresh, not computing.
15. Speeds form a ladder: 1 ns for L1, 100 ns for RAM, 100 us for an SSD, 10 ms for a disk seek, 200 ms across the world.
16. Stretched so one nanosecond is one second, that ladder reads: a second, a minute and a half, a day, a season, six years.
17. A byte is eight bits. Capital B means bytes and lowercase b means bits.
18. Decimal kB is 1,000 and binary KiB is 1,024, so a 1 TB drive reports 931 GB.
19. Hardware, software and firmware differ by how easily a thing changes, not by material.
20. Read every PLAIN block first, end to end, then return for the TECHNICAL blocks.

2

CHAPTER

KB

Electricity — The Thing That Does All The Work

2.0 What this chapter gives you

1. You will be able to say what electricity is, in terms of atoms and the particles inside them.
2. You will be able to explain voltage, current and resistance, and do sums with them.
3. You will be able to work out how much heat a chip makes, and say where that heat comes from.
4. You will be able to trace energy from a wall socket to one transistor inside a processor.
5. You will be able to say what a resistor, capacitor, inductor and diode do, and read real part values.
6. You will be able to explain why computers went digital, using noise margin, not a slogan.
7. You will be able to say what "3.5 GHz" physically means, and why light speed limits chip size.
8. You will be able to explain how one bit is held as charge, and why it can be lost.

2.1 What electricity really is

2.1.1 PLAIN - in simple words

1. Everything is made of atoms. An atom has a heavy centre, the nucleus, with light electrons around it.
2. The nucleus holds protons and neutrons. Each particle has a property called **charge**, a label for how it pushes and pulls on others.
3. A proton is positive, an electron is negative, and a neutron has none, so it is called neutral.
4. Same charges push apart, opposite charges pull together. A normal atom has as many electrons as protons, so it seems to have no charge.
5. In a metal such as copper, the outermost electron of each atom is held weakly and can wander between atoms.
6. Those wandering electrons are electricity. A battery pushes them one way, pulling electrons in at one terminal and shoving them out at the other.

2.1.2 PLAIN - a picture in your head

1. Picture a pipe, already completely full of water, joined end to end in a ring, with a pump in it.
2. Because the pipe is already full, water moves at the far end almost the instant you start the pump.
3. The pump is the battery, its pressure is the voltage, the moving water is the current, and a narrow section is a resistor.
4. Put a water wheel in the ring and the flow turns it. That is a load. Not one drop is destroyed.

Where this comparison breaks

1. In a real wire the electrons crawl at about a tenth of a millimetre per second, far slower than water.
2. Water carries energy inside the pipe. In a circuit most energy travels in the field around the wire, not in the copper.

3. The honest version: the pipe picture models pressure and flow only. Use it for sums, never to decide what is physically true.

2.1.3 PLAIN - a worked example

1. Take a copper wire one square millimetre in cross section carrying one ampere. Copper has about 8.5 x 10^28 free electrons per cubic metre.

```
Drift velocity
v = I / (n * A * e)
  = 1 / (8.5e28 * 1e-6 * 1.602e-19)
  = 0.0000735 m/s = 0.0735 mm per second
  = about 26 centimetres per hour
```

2. So one electron takes over an hour to crawl the length of a desk lamp cable.
3. Yet the lamp lights instantly, because the push travels through the already-full wire near the speed of light.

2.1.4 PLAIN - what is really happening inside

1. A battery's chemistry separates charge, leaving one terminal short of electrons and one with a surplus.
2. That separation creates an electric field, an invisible push filling the space around the terminals.
3. Connect a wire and the field spreads along it near light speed, so every free electron feels the push almost at once.
4. As the electrons drift they bump into vibrating copper atoms, handing over energy that appears as heat.
5. The electrons are not consumed; the same number leaves as enters. What is consumed is energy, delivered by the field to wherever the bumping happens.

2.1.5 TECHNICAL - the engineer's version

1. Charge is quantized in units of the elementary charge e, fixed exactly at 1.602176634 x 10^-19 C by the 2019 SI redefinition.
2. The ampere is now defined from that fixed e, rather than from a force between wires as it was from 1948 to 2019.
3. In a metal the valence and conduction bands overlap, so carriers just above the Fermi level respond to any applied field.
4. Copper gives about one conduction electron per atom, so n is roughly 8.5 x 10^28 per cubic metre, and drift velocity is $v = J / (n\,e)$.
5. Signal propagation is a field phenomenon, travelling at c divided by the square root of the effective relative permittivity.
6. Poynting's theorem, published by John Henry Poynting in 1884, places the energy flux outside the conductor, not inside it.
7. History: von Kleist and van Musschenbroek made the Leyden jar in 1745 and 1746, J. J. Thomson identified the electron in 1897, and Ernest Rutherford proposed the nuclear atom in 1911.

2.1.6 WORDS - remember these

1. Atom — the smallest normal piece of a material — a nucleus of protons and neutrons with bound electrons.
2. Charge — the property that makes things push or pull electrically — quantized in units of e, measured in coulombs.
3. Electron — a tiny negative particle that moves between atoms in a metal — the mobile charge carrier in a conductor.
4. Drift velocity — the slow crawl of the electron crowd — $v = J / (n\,e)$, typically under a millimetre per second.

5. Electric field — the invisible push that moves charges — force per unit charge, in volts per metre.

2.2 Voltage, current and resistance

2.2.1 PLAIN - in simple words

1. **Voltage** is how hard the electricity is pushed, measured in volts. It is always a difference between two points.
2. **Current** is how much charge flows past a point each second, measured in amperes.
3. **Resistance** is how strongly a material fights the flow, measured in ohms.
4. One rule ties all three together: voltage equals current times resistance, written V = I R. Know any two and you can find the third.
5. **Power** is how fast energy is delivered. It is voltage times current, P = V I, measured in watts.
6. Nearly all the power that goes into a chip comes out as heat, which is why chips need fans and metal blocks.

2.2.2 PLAIN - a picture in your head

1. Back to the water ring. Pressure is voltage, litres per second is current, a narrow section is resistance.
2. More pressure through the same pipe gives more flow, so more voltage across the same resistance gives more current.
3. Narrow the pipe further and less flows. More resistance means less current.
4. The pump's effort depends on both. High pressure with no flow does no work, and neither does flow with no pressure.

Where this comparison breaks

1. Water resistance changes messily with speed. A good resistor keeps almost the same ohms at any current.
2. The honest version: Ohm's law is not a law of nature like gravity. It describes materials that happen to behave in a straight line. Diodes, lamps and transistors do not.

2.2.3 PLAIN - a worked example

```
Sum 1. A 5 V supply across a 220 ohm resistor.
  I = V / R = 5 / 220 = 0.0227 A = 22.7 mA

Sum 2. A wire of 0.05 ohms carrying 2 A.
  V = I * R = 2 * 0.05 = 0.1 V lost along it

Sum 3. A part dropping 1.8 V at 15 mA.
  R = V / I = 1.8 / 0.015 = 120 ohms

Power. A core at 1.0 V burning 125 W.
  I = P / V = 125 / 1.0 = 125 A
```

1. That last figure is real. A modern processor pulls over a hundred amps at about one volt.
2. One volt is a gentle push, but a hundred amps is the current of a small welding machine.

2.2.4 PLAIN - what is really happening inside

1. Voltage is energy per unit charge. One volt means one joule given to every coulomb that passes.
2. Current is a counting rate. One ampere is about 6.24 million million million electrons per second.
3. Resistance comes from electrons colliding with warm, vibrating atoms, and each collision turns motion into more vibration, which is heat.
4. So heat is not a side effect bolted on. It is energy the electrons lose when they cannot travel freely.

5. In a chip most heat is not wire resistance. It is the charging and discharging of tiny capacitances inside every transistor.
6. Heat is the enemy because hot silicon leaks more current, which makes more heat, which makes more leaking. That loop can run away.

2.2.5 TECHNICAL - the engineer's version

1. Ohm's law comes from Georg Simon Ohm, published in 1827 in Die galvanische Kette, mathematisch bearbeitet.
2. Power has three equivalent forms: P = V I, P = I squared times R, and P = V squared divided by R.
3. Conductor loss is I squared R, so doubling current quadruples loss. That is why power is transmitted at high voltage and low current.
4. Dynamic switching power in CMOS is about P = alpha C V squared f, with alpha the activity factor and C the switched capacitance.
5. The V squared term is why lowering supply voltage is the strongest power lever, and why core voltages fell from 5 V to under 1 V.
6. Static power is leakage, from subthreshold conduction and gate-oxide tunnelling, and it rises sharply with temperature.
7. Thermal Design Power is a cooling specification, not a measured maximum. Real peaks routinely exceed it.

Processor	Base power	Max turbo power
Intel Core Ultra 9 285K	125 W	250 W
AMD Ryzen 9 9950X	170 W TDP	230 W PPT
Thin laptop chip	15 to 28 W	45 to 65 W
Phone application chip	2 to 5 W	8 to 12 W

8. These are published figures as of 2025. Tom's Hardware measured a 285K peak near 325 W under Prime95 with AVX, above its 250 W rating.
9. Observe power live with `sensors` and `turbostat` on Linux, `powermetrics` on macOS, or the RAPL counters under `/sys/class/powercap`.

2.2.6 WORDS - remember these

1. Volt — how hard electricity is pushed — joules per coulomb of potential difference.
2. Ampere — how much charge flows per second — one coulomb per second, defined from fixed e since 2019.
3. Ohm — how much a part fights the flow — one volt of drop per ampere of current.
4. Watt — how fast energy is used — one joule per second, equal to V times I.
5. TDP — the heat a cooler must remove — Thermal Design Power, a cooling target, not a measured peak.
6. Leakage — current that escapes when nothing is switching — subthreshold and tunnelling current, strongly temperature dependent.

2.3 A circuit: the loop that makes it work

2.3.1 PLAIN - in simple words

1. A circuit needs four things: a source, a path out, something that does work, and a path back.
2. The source is a battery, adapter or generator, the paths are conductors, and the thing that does work is the load.
3. It must be a complete loop, because charge cannot pile up at the end of a wire and stop.
4. A **short circuit** is a return path that skips the load. With nothing to limit it, current becomes enormous and things melt.

5. **Ground** is the point we agree to call zero volts, and every other voltage is measured against it.
6. In **DC** the push is steady and one way. In **AC** it swaps direction many times a second.

2.3.2 PLAIN – a picture in your head

1. Think of a one-way ring road with a roundabout where a machine pushes cars along.
2. Cars leave, drive through a town delivering goods, and come back to the roundabout.
3. Close the road beyond the town and no car can complete the trip, so none leave at all. That is an open circuit.
4. Build a slip road that skips the town and every car takes it, delivering nothing. That is a short circuit.

Where this comparison breaks

1. Cars choose routes. Charge does not choose; it splits between all available paths in inverse proportion to their resistance.
2. The honest version: ground is not one thing. Circuit common, chassis ground and protective earth are three different ideas sharing one word and one symbol.

2.3.3 PLAIN – a worked example

1. Indian mains supplies about 230 V AC at 50 Hz, so the push reverses direction 100 times a second. Here is the journey to one transistor gate.

```
Wall 230 V AC, 50 Hz
   |
   v  bridge of 4 diodes: all half-cycles made positive
Pulsing DC, peaks near 325 V
   |
   v  large capacitor: fills the dips between peaks
Rough DC bus, about 324 V
   |
   v  transistor chops it at 60 kHz to 150 kHz
High-frequency square wave
   |
   v  small transformer: steps down and isolates
Low-voltage AC
   |
   v  fast diodes, capacitors, feedback control
Clean 20 V DC out of the brick
   |
   v  buck converters on the motherboard
12 V rail -> 5 V, 3.3 V, 1.8 V, about 1.0 V core
   |
   v
One transistor gate inside the CPU
```

2. A transformer shrinks as frequency rises. At 50 Hz it is a heavy iron block; at 100 kHz it fits on a fingertip.
3. That is why modern chargers are light. Old adapters worked directly at 50 Hz and were heavy.

2.3.4 PLAIN – what is really happening inside

1. Four diodes act as one-way gates, so both halves of the AC wave come out the same way up.
2. The big capacitor stores charge on the peaks and gives it back in the gaps, smoothing the bumps.
3. A transistor switches that high DC on and off fast, making a square wave the transformer can use.
4. The transformer has two coils. Changing current in the first makes a changing field, which makes a voltage in the second.

5. The turns ratio sets the voltage ratio, and the coils never touch electrically. That isolation keeps mains voltage away from your hands.
6. A feedback circuit adjusts how long the switch stays on, holding the output steady as the load changes.

2.3.5 TECHNICAL - the engineer's version

1. Kirchhoff's current law, from Gustav Kirchhoff in 1845, says currents into any node sum to zero. That is why the loop is mandatory.
2. Indian mains is nominally 230 V RMS at 50 Hz per IS 12360 and IEC 60038; North America uses 120 V RMS at 60 Hz.
3. RMS is the equivalent heating value. Peak is RMS times root two, so 230 V RMS peaks near 325 V, and a bridge leaves a bus near 324 V.
4. The ATX specification defines desktop rails. Legacy ATX 2.x supplies these, plus -12 V for legacy serial ports.

Rail	Nominal	Tolerance	Typical use
+12 V	12.0 V	plus or minus 5%	CPU VRM, GPU, fans
+5 V	5.0 V	plus or minus 5%	SATA, USB, logic
+3.3 V	3.3 V	plus or minus 5%	DIMM support, M.2
+5 VSB	5.0 V	plus or minus 5%	standby, wake on LAN

5. Intel's ATX12VO, published around 2019 and revised to 2.0 in February 2022, keeps only 12 V and 12 V standby. The board then makes 5 V and 3.3 V locally.
6. ATX 3.0, February 2022, introduced the 12VHPWR connector rated to 600 W. ATX 3.1, 2024, replaced it with the 12V-2x6 variant after connector failures.

USB level	Voltage	Current	Power
USB 2.0 high power	5 V	500 mA	2.5 W
USB 3.x	5 V	900 mA	4.5 W
USB Type-C 3 A	5 V	3 A	15 W
USB PD SPR max	20 V	5 A	100 W
USB PD 3.1 EPR max	48 V	5 A	240 W

7. Power Delivery fixed voltages in the Standard Power Range are 5 V, 9 V, 15 V and 20 V.
8. PD 3.1, announced in May 2021, added 28 V, 36 V and 48 V, giving 140 W, 180 W and 240 W.
9. PD 3.0 also defines Programmable Power Supply, a continuously adjustable output from about 3.3 V to 21 V in 20 mV steps.

2.3.6 WORDS - remember these

1. Circuit — a complete loop for charge to travel round — a closed network obeying Kirchhoff's current and voltage laws.
2. Load — the part that does the useful work — the element converting electrical energy into heat, light, motion or computation.
3. Short circuit — an unwanted low-resistance path skipping the load — a fault path of near-zero resistance giving very high current.
4. Ground — the point everyone agrees to call zero volts — the reference node, distinct from chassis and protective earth.
5. AC and DC — a push that swaps direction, against a steady one-way push — alternating current at 50 Hz in India, against direct current.

6. RMS — the equivalent steady value of a wobbling one — root mean square; peak is RMS times root two for a sine.
7. Buck converter — a circuit that steps voltage down efficiently — a switching regulator using an inductor and a switch.

2.4 Conductors, insulators and semiconductors

2.4.1 PLAIN - in simple words

1. A **conductor** lets electricity pass easily: copper, silver, aluminium, gold. An **insulator** blocks it: plastic, rubber, glass, dry air.
2. A **semiconductor** sits between them. Silicon is the famous one, and on its own it barely conducts.
3. Copper conducts because each atom releases one outer electron, and those electrons roam free through the metal.
4. Plastic does not conduct because every electron is locked into a bond between two atoms, with nothing free to move.
5. Silicon locks its electrons too, but weakly, so a little heat or light can free a few of them.
6. The magic is that we can change how well silicon conducts, on purpose, by mixing in traces of other elements. That is why computers are silicon.

2.4.2 PLAIN - a picture in your head

1. Imagine a car park. A conductor is one only half full, so any car can move at once.
2. An insulator is packed solid, bumper to bumper, with a high wall and the next empty level far above. Nothing moves.
3. A semiconductor is the same packed park, but the empty level is only a short ramp up, so a nudge of warmth or light lets a car reach it.
4. A neighbour shuffles into the space it left, so the space appears to move the other way. Engineers call that moving space a hole.

Where this comparison breaks

1. Real electrons are not objects in slots. They are quantum states shared across the whole crystal.
2. The honest version: band gap is the true idea. The levels are energy bands and the wall height is the gap in electron-volts. The picture gets the behaviour right and the physics only roughly right.

2.4.3 PLAIN - a worked example

1. Resistivity says how strongly a material fights current for a standard sample size. The range spans over twenty powers of ten.

Material	Resistivity, ohm-metre	Band gap, eV
Silver	1.59 x 10^-8	none, metal
Copper	1.68 x 10^-8	none, metal
Aluminium	2.65 x 10^-8	none, metal
Pure silicon	about 2.3 x 10^3	1.12
Germanium	about 0.46	0.66
Glass	10^11 to 10^15	wide
PTFE plastic	above 10^23	very wide

2. Copper conducts about 10^11 times better than pure silicon, and silicon about 10^20 times better than PTFE.

3. Now the trick. Add one phosphorus atom per ten million silicon atoms and the resistivity falls by a factor of thousands.
4. That is **doping**, and it is the step that turns sand into a switch.

2.4.4 PLAIN - what is really happening inside

1. Silicon has four outer electrons and bonds with four neighbours, so every electron is paired and none is free.
2. Phosphorus has five outer electrons, so four bonds form and one electron is left loosely held. That is **n-type** silicon.
3. Boron has three, so one bond is incomplete, leaving a vacancy that behaves like a positive carrier. That is **p-type** silicon.
4. Put n-type and p-type next to each other and the junction passes current one way and blocks the other.
5. Stack those junctions in the right pattern and you get a transistor: a switch with no moving parts, controlled by a voltage, repeated billions of times in a chip.

2.4.5 TECHNICAL - the engineer's version

1. Conduction follows band structure. Metals have a partly filled conduction band; insulators and semiconductors have a filled valence band and a forbidden gap.
2. Silicon has an indirect band gap of 1.12 eV at 300 K. Germanium is 0.66 eV, gallium arsenide 1.42 eV, gallium nitride 3.4 eV, diamond about 5.5 eV.
3. Silicon dioxide, the gate insulator, has a gap near 9 eV, which is why it works as a barrier only a few atomic layers thick.
4. Intrinsic carrier concentration in silicon at 300 K is about 1.0 x 10^10 per cubic centimetre, against 5 x 10^22 atoms per cubic centimetre.
5. Doping runs from about 10^15 per cubic centimetre in lightly doped wells to above 10^20 at source and drain contacts.
6. Copper resistivity has a temperature coefficient near 0.00393 per degree Celsius, so a trace is about 40% more resistive at 100 C than at 0 C.
7. Semiconductors move the opposite way: resistance falls as temperature rises, because thermal energy frees more carriers.
8. Aluminium was the standard on-chip interconnect until IBM shipped copper interconnect in 1997, cutting wire resistance by roughly 40%.

2.4.6 WORDS - remember these

1. Conductor — a material electricity passes through easily — partly filled conduction band, high carrier mobility.
2. Insulator — a material electricity cannot pass through — a wide forbidden band gap, typically above 4 eV.
3. Semiconductor — a material in between, whose conducting can be controlled — a moderate band gap, roughly 0.5 to 3.5 eV.
4. Band gap — the energy step an electron must climb to move — the forbidden range between valence and conduction bands, in eV.
5. Doping — adding traces of another element to change conductivity — controlled introduction of donor or acceptor impurities.
6. n-type and p-type — silicon with spare electrons, against silicon with missing ones — donor-doped and acceptor-doped material.
7. Hole — the moving gap where an electron is missing — a quasiparticle carrying positive charge in the valence band.

2.5 The resistor

2.5.1 PLAIN - in simple words

1. A resistor is a small part whose only job is to fight current by a fixed, known amount.
2. Inside is a ceramic rod coated with a thin film of carbon or metal, with a spiral groove cut into the film.
3. A longer, thinner spiral means a longer electrical path, so more ohms. That is how the value is set.
4. Resistors turn the energy they take into heat, so each has a power rating that must not be exceeded.
5. Two very common uses are the **pull-up** and the **pull-down**, which stop an input wire from having no defined voltage.
6. A wire connected to nothing is **floating**, and a floating input on a chip is a genuine bug, not a harmless idle state.
7. Another everyday use is limiting the current into an LED, which would otherwise destroy itself instantly.

2.5.2 PLAIN - a picture in your head

1. A pull-up resistor is like a weak spring holding a door closed.
2. Anyone can push the door open easily, but if nobody touches it the spring decides where it sits.
3. Without the spring the door drifts, banging half open in any draught. That is a floating input.
4. The spring is deliberately weak so a real signal can override it without wasting effort.

Where this comparison breaks

1. A doorway caps how many people pass. A resistor does not cap current at all.
2. The honest version: a resistor develops a voltage proportional to whatever current passes through it. Force enough current through and it burns.

2.5.3 PLAIN - a worked example

1. Light a red LED from a 5 V supply at about 15 mA. A red LED drops about 2.0 V, and that drop barely changes with current.

```
Voltage left for the resistor
  = 5.0 - 2.0 = 3.0 V

R = V / I = 3.0 / 0.015 = 200 ohms
Nearest standard E12 value: 220 ohms

Actual current  I = 3.0 / 220 = 13.6 mA
Heat in resistor
  P = I * I * R = 0.01364 * 0.01364 * 220
    = 0.041 W

A 0.25 W part is comfortable. Even an 0603 surface
mount part rated 0.1 W is fine.
```

2. Leave the resistor out and the LED tries to pass everything the supply can give, and fails within seconds.

2.5.4 PLAIN - what is really happening inside

1. In the resistive film, electrons collide with the disordered atoms of the alloy, turning electrical energy into heat.
2. Because electrons lose energy along the way, the potential falls steadily from one end to the other. That fall is the voltage drop.
3. A pull-up connects an input pin to the supply, so a small current holds the pin high when nothing else drives it.

4. When a switch pulls the pin to ground it wins easily, because the pull-up is weak. A 10 kilohm pull-up on 3.3 V wastes only 0.33 mA.
5. Floating is dangerous because a MOS gate is an insulator, giving an input resistance above a million million ohms, so any stray charge drags the pin anywhere.
6. Worse, a pin resting halfway turns on both transistors of the input stage at once, drawing extra current and adding heat.

2.5.5 TECHNICAL - the engineer's version

1. Types include carbon film, metal film, metal oxide, thick film on alumina for surface mount, wirewound for power, and foil for precision.
2. Values follow the IEC 60063 preferred series. E12 gives twelve values per decade at 10% spacing, E24 gives 24, and E96 gives 96 for 1% parts.
3. The E12 sequence is 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82, and decade multiples of those.
4. Through-hole colour bands are read left to right, with the tolerance band set off by a wider gap.

Colour	Digit	Multiplier	Tolerance
Black	0	x1	none
Brown	1	x10	1%
Red	2	x100	2%
Orange	3	x1 k	none
Yellow	4	x10 k	none
Green	5	x100 k	0.5%
Blue	6	x1 M	0.25%
Violet	7	x10 M	0.1%
Grey	8	x100 M	0.05%
White	9	x1 G	none
Gold	none	x0.1	5%
Silver	none	x0.01	10%

5. Worked reading: red, red, brown, gold is 2, 2, times 10, at 5 percent, giving 220 ohms plus or minus 11 ohms.
6. Surface mount sizes and typical ratings: 0402 at 0.063 W, 0603 at 0.1 W, 0805 at 0.125 W, 1206 at 0.25 W, sized in hundredths of an inch.
7. Common pull-up values are 10 kilohm general purpose, and 4.7 or 2.2 kilohm for I2C depending on bus capacitance and speed.
8. The I2C specification, from Philips in 1982 and now maintained by NXP as UM10204, caps bus capacitance at 400 pF and constrains rise time per mode.
9. Microcontroller internal pull-ups are typically 20 to 50 kilohms and are too weak for I2C. A floating CMOS input shows a slow wandering trace on a scope, not a flat line.

2.5.6 WORDS - remember these

1. Resistor — a part that fights current by a known amount — a two-terminal component with specified ohmic value and power rating.
2. Pull-up — a weak connection to the supply that holds a line high — a resistor from a node to VDD defining the idle logic level.
3. Pull-down — a weak connection to ground that holds a line low — a resistor from a node to VSS defining the idle logic level.
4. Floating — a wire connected to nothing, with undefined voltage — a high-impedance node with no DC path, open to noise coupling.

5. E12 series — the standard twelve resistor values per decade — an IEC 60063 preferred number series at 10% spacing.
6. Current limiting resistor — the resistor that stops an LED burning out — a series element sized as supply minus Vf, divided by target If.

2.6 The capacitor

2.6.1 PLAIN - in simple words

1. A **capacitor** is two sheets of metal held very close, with a thin insulator between them so they never touch.
2. Apply a voltage and electrons pile onto one sheet while the same number are pulled off the other. No charge crosses the gap.
3. The sheets simply become oppositely charged, and the capacitor now holds energy.
4. How much charge it holds per volt is its **capacitance**, in farads. A farad is huge, so real parts are millionths or million-millionths.
5. Charging is not instant. Fed through a resistor, it fills quickly at first and then more slowly, and the time depends on resistance times capacitance. That product is the **time constant**.
6. One bit of your computer's main memory is literally a capacitor holding charge, and it leaks, so it must be topped up.

2.6.2 PLAIN - a picture in your head

1. Think of a rubber sheet stretched tightly across a pipe, sealing it completely.
2. Water cannot get past, but pushing against one side stretches the sheet, which pushes water out of the other side.
3. Push harder and it stretches more. That stretch is stored energy, exactly like charge on the plates.
4. Push and release repeatedly and water sloshes back and forth. That is why a capacitor passes AC and blocks DC.

Where this comparison breaks

1. A rubber sheet has a breaking point. A capacitor has a voltage limit too, but past it the insulator punches through and the part usually shorts.
2. The honest version: a capacitor does not pass AC. No charge ever crosses the insulator. Current flows in one wire and out the other because the charge on each plate is changing.

2.6.3 PLAIN - a worked example

1. Take a 10 kilohm resistor feeding a 100 nanofarad capacitor from a 3.3 V supply.

```
tau = R * C = 10000 * 0.0000001
    = 0.001 s = 1 millisecond
```

2. The voltage climbs along a curve, reaching about 63.2% after one time constant, and never quite arriving.

Time	Multiples of tau	Percent charged	Volts
1 ms	1	63.2%	2.09
2 ms	2	86.5%	2.85
3 ms	3	95.0%	3.14
5 ms	5	99.3%	3.28

3. Engineers treat five time constants as fully charged, because the last 0.7% is below normal measurement noise.
4. This exact sum sets how long a reset pin stays low at power-on, how long a button debounce lasts, and how fast an edge can rise.

2.6.4 PLAIN - what is really happening inside

1. When a chip switches millions of transistors at once, it demands a burst of current within a few billionths of a second.
2. The supply is centimetres away, and the copper path to it behaves like a small coil that cannot change current that fast.
3. Without help the voltage at the chip would dip every time it worked hard, and it would misread its own signals.
4. A **decoupling capacitor** sitting millimetres from the power pins supplies that burst locally, then refills from the main supply.
5. Large ones hold plenty but respond slowly; small ones hold little but respond fast, so designers fit several values in parallel.
6. In main memory one bit is one capacitor plus one transistor, and charge leaks away within tens of milliseconds.
7. So the controller reads every row and writes it straight back, forever. That is **refresh**, and it never stops while power is on.

2.6.5 TECHNICAL - the engineer's version

1. Capacitance is C = Q / V in farads. For parallel plates, C equals epsilon-zero times epsilon-r times area, divided by separation.
2. Charging through a resistor gives $v(t) = V$ times one minus e to the power of minus t over RC, and discharging drops to 36.8% after one tau.
3. Energy stored is one half C V squared, so a 470 microfarad bulk capacitor at 12 V holds 0.034 joules.
4. Real capacitors have equivalent series resistance and inductance, creating a self-resonant frequency above which they behave as inductors.
5. A typical 0402 100 nF ceramic self-resonates around 30 to 50 MHz, which is why power networks use several decades of value in parallel.
6. Dielectric class matters. C0G or NP0 is stable and low loss; X7R is denser but loses 30% to 60% of its capacitance under DC bias near its rating.
7. Robert Dennard invented the one-transistor, one-capacitor DRAM cell at IBM in 1966; United States patent 3,387,286 was granted in 1968.
8. Modern DRAM storage capacitance is roughly 6 to 10 femtofarads, held in a deep trench or tall pillar to gain area without floor space.
9. DDR4 requires every row refreshed within 64 ms, with an average interval tREFI of 7.8 microseconds at or below 85 C, halved above it.
10. DDR5 tightens the window to 32 ms at normal temperature and 16 ms in the extended range, with on-die thermal sensing to request faster refresh.
11. Refresh costs real bandwidth; on large DDR4 modules the overhead can exceed 5% of available cycles. Read module timings on Linux with `decode-dimms`.

2.6.6 WORDS - remember these

1. Capacitor — two plates with a gap, storing charge — a component storing energy in an electric field, C = Q / V.
2. Farad — the unit of charge stored per volt — one coulomb per volt; practical parts are pF, nF and uF.
3. Time constant — how long a capacitor takes to fill — tau = R C; 63.2% after one tau, 99.3% after five.
4. Decoupling capacitor — a local energy store beside a chip — a capacitor supplying transient current and lowering power network impedance.

5. Dielectric — the insulating layer between the plates — the material whose permittivity sets capacitance and whose breakdown sets voltage rating.
6. ESR — the hidden resistance inside a capacitor — equivalent series resistance, which dominates ripple loss and heating.
7. Refresh — reading and rewriting memory so it does not forget — periodic restoration of DRAM cell charge within the retention window.

2.7 The inductor and the coil

2.7.1 PLAIN - in simple words

1. Whenever current flows through a wire, an invisible **magnetic field** wraps around it in circles.
2. One straight wire makes a weak field. Wind it into a coil and every turn adds its field to the others.
3. A coil carrying current therefore behaves like a magnet. That is an **inductor**.
4. The second half of the rule matters more: a magnetic field that is changing pushes electrons in any nearby wire.
5. Current makes a field, and a changing field makes current. Those two sentences explain motors, generators, transformers and speakers.
6. An inductor resists changes in current, passing a steady current freely but fighting any sudden rise or fall. That makes it the opposite of a capacitor.
7. Coils are all over a computer: the blocks near the processor socket, the coil in every speaker, the aerial in every wireless chip.

2.7.2 PLAIN - a picture in your head

1. Think of a heavy flywheel on a shaft. To get it spinning you must push for a while.
2. It refuses to speed up instantly however hard you shove, and once spinning it fights hard if you try to stop it.
3. Current in an inductor behaves exactly like that speed. It builds slowly and resists being stopped.
4. Cut the current to a coil abruptly and it produces a large voltage spike, which is why any circuit switching a relay or motor includes a diode.

Where this comparison breaks

1. A flywheel stores energy in motion. An inductor stores it in the magnetic field around the coil, not in the copper.
2. The honest version: nothing is spinning. The inertia is the field's resistance to change, described by Faraday's law and Lenz's law, acting at the speed of light.

2.7.3 PLAIN - a worked example

1. A motherboard must make about 1.0 V for the processor core from the 12 V rail, at over 100 amps.
2. It cannot use a resistor, because 11 V times 100 A would waste 1100 watts as heat. It uses a **buck converter** instead.

```
Vout is about D * Vin, where D is the duty cycle

To get 1.0 V from 12 V:
  D = 1.0 / 12 = 0.083 = 8.3%

At 1 MHz switching, one period is 1000 ns, so the
switch is on 83 ns and off 917 ns, a million times
every second.

Energy in one choke, L = 0.15 uH at I = 30 A:
```

```
E = 0.5 * L * I * I
  = 0.5 * 0.00000015 * 900 = 67.5 microjoules
```

3. Because 100 A is too much for one switch and one coil, the design splits into phases, often 8, 12 or 16.
4. The phases take turns, evenly spaced in time, so their ripple largely cancels. Those evenly spaced blocks around the socket are exactly this.

2.7.4 PLAIN - what is really happening inside

1. When the switch turns on, 12 V appears across the inductor and current ramps up, building the magnetic field.
2. Current cannot jump, so it rises in a straight ramp while the output capacitor collects the charge.
3. When the switch turns off, the field collapses and drives the current onward through a second transistor to ground.
4. So current into the load is continuous, even though the switch is only connected 8.3% of the time.
5. A transformer uses the same physics differently: two coils share one field, and the voltage ratio equals the turns ratio.
6. In a loudspeaker a coil sits in a magnet's gap and is glued to a cone.
7. Current in the coil makes a force, so the cone moves. The speaker chapter builds on this.

2.7.5 TECHNICAL - the engineer's version

1. Hans Christian Orsted found in 1820 that a current deflects a compass needle, and Andre-Marie Ampere formalized the force between conductors that same year.
2. Michael Faraday demonstrated electromagnetic induction in 1831 at the Royal Institution, showing a changing field induces an electromotive force.
3. Heinrich Lenz stated in 1834 that the induced current opposes the change producing it, which is why Faraday's law carries a minus sign.
4. The defining relation is v = L di/dt, with L in henries, named after Joseph Henry, who found induction independently around 1831.
5. Energy stored is one half L I squared. Real inductors also have DC resistance, core loss, and a saturation current.

Application	Typical inductance	Typical current
CPU VRM choke	0.1 to 1 uH	25 to 60 A
Buck for 5 V or 3.3 V	1 to 10 uH	1 to 5 A
Speaker voice coil	0.1 to 1 mH	0.1 to 3 A
Wireless charging coil	6 to 24 uH	under 2 A

6. Multiphase controllers interleave phases 360 degrees divided by N apart, cancelling much of the input and output ripple current.
7. Switching frequencies run 300 kHz to 1 MHz for desktop CPU VRMs, and 1 to 3 MHz for point-of-load converters.
8. Coils appear in antennas as matching elements, in relays and motors as the actuator, and formerly in disk drives as inductive read heads.
9. Inductive heads gave way to giant magnetoresistive heads, shipped by IBM in 1997, from work by Albert Fert and Peter Grunberg that won the 2007 Nobel Prize.
10. Qi wireless charging, from the Wireless Power Consortium, operates at 87 to 205 kHz in the baseline power profile.

2.7.6 WORDS - remember these

1. Magnetic field — the invisible pull wrapping around a current — the vector field B, in tesla, produced by moving charge.
2. Inductor — a coil that resists changes in current — a component with defined inductance, following v = L di/dt.
3. Henry — the unit of inductance — one volt induced per ampere per second of current change.
4. Induction — a changing field creating a voltage in a nearby wire — Faraday's law, opposed in direction per Lenz's law.
5. Transformer — two coils sharing a field to change voltage — a coupled pair with output set by turns ratio, giving galvanic isolation.
6. VRM — the circuit making the processor's low voltage — voltage regulator module, a multiphase synchronous buck converter.
7. Saturation — the point where a coil's core gives up — the current above which permeability collapses and inductance falls sharply.

2.8 The diode

2.8.1 PLAIN - in simple words

1. A **diode** lets current pass one way and blocks it the other. It is made by joining p-type silicon to n-type silicon.
2. Push current the right way and it flows, but the diode keeps about 0.7 volts as a toll, called the **forward voltage**.
3. Push the other way and almost nothing flows, until the voltage is so high that the diode breaks and usually dies.
4. Four diodes in a diamond form a **rectifier bridge**, turning alternating current into current that always flows one way.
5. An **LED** is a diode made from materials that give off light when current passes.
6. A **photodiode** is the same idea backwards. Light falling on the junction makes a small current, so it works as a light sensor.

2.8.2 PLAIN - a picture in your head

1. Picture a swing door hinged to open only one way, with a weak spring holding it shut.
2. From the correct side you can push through, but you must lean on it a little. That lean is the forward voltage.
3. From the wrong side the door will not budge, and shoving hard enough breaks the hinges, which is exactly reverse breakdown.
4. Once open, extra people flow through very easily, which is why a diode cannot limit current by itself.

Where this comparison breaks

1. A door is open or shut. A diode conducts smoothly and exponentially: a trickle at 0.5 V, plenty at 0.7 V, far too much at 0.8 V.
2. The honest version: 0.7 V is not a fixed property. It is where a small silicon diode reaches a few milliamps at room temperature, and it shifts by about minus 2 mV per degree Celsius as it warms.

2.8.3 PLAIN - a worked example

1. Feed 230 V AC into a bridge of four silicon rectifiers with a smoothing capacitor.

```
Peak = RMS * square root of 2 = 230 * 1.414 = 325 V

Two diodes are in the path at any moment, each
```

```
dropping about 0.7 V:
     325 - 1.4 = about 324 V DC
```

2. This is why the inside of a mains adapter is dangerous even after unplugging. The bulk capacitor can hold over 300 V for many seconds.
3. Now LEDs. Forward voltage depends on colour, because colour is set by the energy step each electron falls through.

LED colour	Forward voltage at 20 mA
Infrared	1.2 to 1.6 V
Red	1.8 to 2.2 V
Yellow or amber	2.0 to 2.2 V
Green	2.8 to 3.5 V
Blue	3.0 to 3.6 V
White	2.8 to 3.6 V

4. Redder light means less energy per photon and a lower forward voltage. Blue and white need about 3 V, so one 1.5 V cell cannot light a white LED.

2.8.4 PLAIN - what is really happening inside

1. Where p-type and n-type meet, spare electrons from the n side drift across and fill vacancies on the p side.
2. That leaves a thin zone with no free carriers, called the depletion region, with a built-in voltage across it.
3. Connect positive to the p side and you push against that built-in voltage. Overcome it and carriers flood across.
4. Connect positive to the n side and the empty zone widens instead, so nothing can cross and the diode blocks.
5. In an LED, an electron crossing and dropping into a vacancy loses energy that leaves as a photon, and the band gap sets its colour exactly.
6. In a photodiode the reverse happens. An arriving photon knocks an electron free and the built-in field sweeps it out as current.

2.8.5 TECHNICAL - the engineer's version

1. The current-voltage relation is the Shockley diode equation, with thermal voltage about 25.9 mV at 300 K, giving an exponential curve.
2. Forward voltage by technology: silicon p-n 0.6 to 0.7 V, germanium 0.25 to 0.35 V, Schottky 0.2 to 0.45 V, silicon carbide about 0.8 V.
3. Schottky diodes use a metal-semiconductor junction and have almost no reverse recovery time, which is why switching supplies use them.
4. Zener diodes run deliberately in reverse breakdown, at standard values from 2.4 V to 200 V, as simple references and clamps.

Part	Type	Rating
1N4148	small signal	100 V, 200 mA
1N4007	rectifier	1000 V, 1 A
1N5819	Schottky	40 V, 1 A
BAT54	surface mount Schottky	30 V, 200 mA

5. Ratings to check on any datasheet: peak inverse voltage, average forward current, forward voltage at the operating current, and reverse recovery time.
6. Oleg Losev reported light from silicon carbide junctions in 1927, but the effect was not developed at the time.
7. Nick Holonyak Jr. made the first practical visible-spectrum LED, a red one, at General Electric in 1962.
8. Shuji Nakamura produced high-brightness blue LEDs at Nichia in 1993, making white LEDs possible. He shared the 2014 Nobel Prize in Physics with Isamu Akasaki and Hiroshi Amano.
9. Photodiodes usually run reverse biased in photoconductive mode, giving faster response and a current linear in optical power.
10. Every CMOS input pin has ESD protection diodes to VDD and VSS, which is why driving a signal into an unpowered chip can back-feed its supply rail.

2.8.6 WORDS - remember these

1. Diode — a one-way valve for current — a p-n junction device conducting only under forward bias.
2. Forward voltage — the toll for letting current through — Vf, about 0.7 V for silicon, set by the Shockley equation and temperature.
3. Reverse breakdown — where a diode gives up and conducts backwards — avalanche or Zener breakdown beyond the peak inverse voltage.
4. Rectifier — a circuit turning AC into one-way current — a half-wave or full-wave bridge arrangement of diodes.
5. LED — a diode that gives out light — a direct-band-gap junction emitting photons on carrier recombination.
6. Photodiode — a diode turning light into current — a reverse-biased junction producing photocurrent proportional to optical power.
7. Schottky diode — a fast diode with a low toll — a metal-semiconductor junction with low Vf and negligible reverse recovery.

2.9 Signals: analogue and digital

2.9.1 PLAIN - in simple words

1. A **signal** is any physical thing that changes over time in order to carry information: a voltage on a wire, sound in air, light in a fibre.
2. An **analogue** signal carries meaning in its exact value. Half a volt more is a genuinely different message.
3. A **digital** signal carries meaning only in which side of a line the value falls on, so 2.9 V and 3.2 V mean exactly the same thing.
4. Every wire picks up unwanted wobble from nearby wires, motors and radio. That wobble is **noise**.
5. Noise damages an analogue signal a little at every stage, and the damage adds up and can never be removed.
6. A digital signal survives, because each stage rebuilds a clean, full-strength copy, as long as noise never pushes a value across the line.
7. The gap between the worst a sender may produce and the least a receiver demands is the **noise margin**.

2.9.2 PLAIN - a picture in your head

1. Imagine passing a message down a line of a hundred people in a noisy room.
2. In the analogue version each person whispers the exact loudness they heard, so small errors creep in at every step and the end is mush.
3. In the digital version everyone agrees on only two messages, a nod or a shake, and each person makes a fresh, full-strength one.
4. Person one hundred receives exactly what person one sent, even though every step was imperfect.

Where this comparison breaks

1. If the room gets loud enough that a nod is mistaken for a shake, the error becomes permanent and is copied faithfully forever.
2. The honest version: digital is not immune to noise. Below the threshold errors vanish completely; above it they become total. That cliff edge is why systems add error-detecting codes on top.

2.9.3 PLAIN – a worked example

1. A 3.3 V chip drives another. The sender promises a one will be at least 2.4 V, and the receiver reads anything above 2.0 V as a one.

```
High noise margin = VOH min - VIH min
                  = 2.4 - 2.0 = 0.4 V

Low noise margin  = VIL max - VOL max
                  = 0.8 - 0.4 = 0.4 V
```

2. So any noise smaller than 0.4 V cannot corrupt a bit. Drop the system to 1.2 V logic and the cushion falls to roughly 0.15 V.
3. The same 0.2 V of noise that was harmless at 3.3 V now flips bits. Nothing about the noise changed, only the cushion.
4. That is the central trade of the last thirty years: lower voltage saves a great deal of power and costs you margin.

2.9.4 PLAIN – what is really happening inside

1. A digital output is a pair of transistors, one connecting the wire to the supply and one to ground, with only one on at a time.
2. When the output flips, the wire's own capacitance must be charged or drained, and that takes time. That time is the **rise time**.
3. Rise time is measured from 10% to 90% of the swing, because the very start and end of the curve are slow and vague.
4. During the rise the voltage passes through the forbidden middle zone where the receiver's answer is undefined, which is safe only because the clock says when to read.
5. A faster edge spends less time undefined, but carries higher frequency content that reflects, radiates and rings, so designers sometimes slow edges on purpose.

2.9.5 TECHNICAL – the engineer's version

1. The four defining parameters on every logic datasheet are VOH minimum, VOL maximum, VIH minimum and VIL maximum.
2. Noise margin high is VOH min minus VIH min. Noise margin low is VIL max minus VOL max.

Family	Supply	VIL max	VIH min
TTL 74xx	5.0 V	0.8 V	2.0 V
CMOS 74HC	5.0 V	1.35 V	3.15 V
LVTTL, LVCMOS	3.3 V	0.8 V	2.0 V
LVCMOS	2.5 V	0.7 V	1.7 V
LVCMOS	1.8 V	0.63 V	1.17 V

Family	VOL max	VOH min	Margins, low and high
TTL 74xx	0.4 V	2.4 V	0.4 V and 0.4 V
CMOS 74HC	0.1 V	4.4 V	1.25 V and 1.25 V

Family	VOL max	VOH min	Margins, low and high
LVTTL 3.3 V	0.4 V	2.4 V	0.4 V and 0.4 V
LVCMOS 1.8 V	0.45 V	1.35 V	0.18 V and 0.18 V

3. These are typical family values from standard datasheets and the JEDEC JESD8 series. Always confirm against the exact part and its test conditions.
4. TTL, transistor-transistor logic, came from the Texas Instruments 7400 series introduced in 1964 and dominated for two decades.
5. TTL thresholds are asymmetric because bipolar inputs sink current when low. CMOS inputs draw almost none, so their thresholds sit near half the supply.
6. Supply voltages fell in steps as features shrank: 5 V through the 1980s, 3.3 V from the mid 1990s, then 2.5 V, 1.8 V, 1.5 V and 1.2 V.
7. Processor core voltage today runs roughly 0.7 V to 1.35 V, adjusted dynamically thousands of times a second by the power management unit.
8. DDR4 uses a 1.2 V supply, DDR5 uses 1.1 V, and LPDDR5 uses 1.05 V with 0.5 V on the data lines.
9. Rise time relates to bandwidth by the approximation: bandwidth in gigahertz equals 0.35 divided by rise time in nanoseconds.
10. Modern high-speed links drop single-ended thresholds for differential pairs. LVDS swings about 350 mV around a common mode near 1.2 V, and the receiver measures only the difference, rejecting common-mode noise.

2.9.6 WORDS - remember these

1. Signal — something that changes over time to carry information — a time-varying physical quantity conveying data.
2. Analogue — meaning carried by the exact value — a continuous-amplitude representation where every level is distinct information.
3. Digital — meaning carried by which side of a line — a discrete representation quantized to defined logic levels.
4. Noise — unwanted wobble picked up along the way — random or coupled interference added to a signal.
5. Noise margin — the cushion before noise breaks a bit — VOH min minus VIH min, and VIL max minus VOL max.
6. Rise time — how long an edge takes to go up — the 10% to 90% transition time, which sets the bandwidth required.
7. Differential pair — two wires carrying opposite copies of one signal — a balanced scheme rejecting common-mode noise.

2.10 Clocks: how a computer keeps time

2.10.1 PLAIN - in simple words

1. A computer needs a steady heartbeat so every part agrees when a step begins and ends. That is the **clock**.
2. The clock is a square wave: a voltage going high, low, high, low, forever, at an exactly even rate.
3. The steadiness comes from a small piece of quartz crystal, usually sealed in a tiny metal can on the board.
4. Quartz has an unusual property. Squeeze it and it makes a small voltage; apply a voltage and it physically bends. That is the **piezoelectric effect**.
5. A quartz bar of a given size wants to vibrate at one particular rate, the way a tuning fork wants to sound one note.

6. Put it in an amplifier that feeds its own output back into it and it settles at exactly that rate and stays there.
7. The crystal is slow compared with a processor, so a circuit called a multiplier turns a low rate into a very high one.

2.10.2 PLAIN - a picture in your head

1. Think of a pendulum in a grandfather clock. Its swing rate is set by its length, not by how hard you push it.
2. A tiny nudge each swing keeps it going, and the swings stay even for years. The nudge does not set the rate; the pendulum does.
3. A quartz crystal is the same idea, but the pendulum is a sliver of stone the size of a grain of rice, swinging thirty-two thousand times a second.

Where this comparison breaks

1. A pendulum drifts with gravity and temperature. A crystal drifts with its cut, its temperature and its age.
2. The honest version: a crystal is not perfect. A typical watch crystal is specified at plus or minus 20 parts per million, which is about 1.7 seconds a day, or roughly ten minutes a year.

2.10.3 PLAIN - a worked example

1. Nearly every real-time clock chip and quartz wristwatch uses exactly 32,768 hertz, which is two multiplied by itself fifteen times.

```
32768 -> 16384 -> 8192 -> 4096 -> 2048 -> 1024
  -> 512 -> 256 -> 128 -> 64 -> 32 -> 16
  -> 8 -> 4 -> 2 -> 1 Hz

Fifteen halvings. A chain of 15 flip-flops does it
with no arithmetic, no adder and almost no power.
```

2. It is also a good compromise. A lower frequency needs a physically larger crystal, and a higher one burns more power in the divider.

```
Base clock (BCLK) = 100 MHz
Multiplier        = 35
Core clock        = 100 * 35 = 3.5 GHz

One clock period at 3.5 GHz:
  T = 1 / 3.5e9 = 0.2857 ns = 285.7 picoseconds
```

3. So 3.5 GHz physically means the core's heartbeat completes 3,500,000,000 full cycles a second, each lasting 286 trillionths of a second.

2.10.4 PLAIN - what is really happening inside

1. In one clock period a signal can only travel so far, because nothing beats light and copper is slower than that.
2. Light in empty space covers about 30 centimetres in one nanosecond. On a circuit board a signal covers roughly half that.

Medium	Speed	In 1 ns	In one 3.5 GHz tick
Vacuum, light	30 cm/ns	30 cm	8.6 cm
Coax at 66%	20 cm/ns	20 cm	5.7 cm

Medium	Speed	In 1 ns	In one 3.5 GHz tick
FR-4 microstrip	17.5 cm/ns	17.5 cm	5.0 cm
FR-4 stripline	15.0 cm/ns	15.0 cm	4.3 cm

3. The honest version: the often-repeated figure of about 8.5 centimetres per tick at 3.5 GHz is the vacuum figure. In copper on a board it is roughly half that, and inside a chip it is worse again.
4. That is a hard physical ceiling. A processor die is one to three centimetres across, so a signal cannot cross a large die and settle within one cycle.
5. This is why big chips are divided into regions, why data takes several cycles to cross a die, and why making a chip bigger does not make it faster.
6. The clock itself must reach every corner at almost the same moment, so designers build a balanced tree of buffers with equal path lengths.
7. Any remaining difference in arrival time is clock skew, and it eats directly into the time available for real work.

2.10.5 TECHNICAL - the engineer's version

1. Jacques and Pierre Curie discovered piezoelectricity in 1880, working with quartz, tourmaline and Rochelle salt.
2. Warren Marrison and J. W. Horton built the first quartz clock at Bell Telephone Laboratories in 1927.
3. Watch crystals use a tuning-fork cut in a small cylindrical can, commonly specified at 32.768 kHz plus or minus 20 ppm at 25 C.
4. Frequency drifts with temperature along a parabola for tuning-fork cuts, peaking near 25 C. TCXO and OCXO parts add compensation or an oven, reaching 0.01 ppm.
5. The oscillator is typically a Pierce circuit: one inverting amplifier, the crystal, two load capacitors and a feedback resistor.
6. A phase-locked loop multiplies the reference, using a phase detector, a loop filter, a voltage-controlled oscillator and a feedback divider.
7. A 100 MHz base clock has been the convention on Intel and AMD desktop platforms since around 2011, replacing earlier 133 MHz and 200 MHz references.
8. That is a convention, not a written standard, and board makers may shift it slightly.
9. Advertised frequency is several numbers, not one: a base clock, a single-core boost and an all-core boost, all limited by temperature and current.
10. Clock jitter, the cycle-to-cycle variation, matters as much as average frequency for serial links. PCI Express 5.0 at 32 GT/s has a unit interval of 31.25 ps.
11. On-die clock distribution uses H-trees or grids with skew budgets of tens of picoseconds, and can consume 20% to 30% of dynamic power.

```
lscpu | grep -i mhz
grep MHz /proc/cpuinfo
sudo dmidecode -t processor
cpupower frequency-info
```

2.10.6 WORDS - remember these

1. Clock — the steady heartbeat every part follows — a periodic square wave defining the timing reference for synchronous logic.
2. Piezoelectric effect — squeeze makes voltage, voltage makes bend — reversible electromechanical coupling, found by the Curies in 1880.
3. Crystal — the quartz sliver that sets the rate — a mechanical resonator of very high Q used as a frequency reference.
4. 32.768 kHz — the standard watch frequency — two to the power of fifteen hertz, divisible to exactly 1 Hz by fifteen flip-flops.

5. PLL — the circuit that multiplies a slow clock into a fast one — phase-locked loop with phase detector, loop filter, VCO and divider.
6. Base clock — the low reference frequency on the board — BCLK, conventionally 100 MHz on modern desktop platforms.
7. Clock skew — the small difference in when the beat arrives — the spread in clock edge arrival times across a die, in picoseconds.

2.11 How one bit is physically stored and moved

2.11.1 PLAIN - in simple words

1. A bit is not a thing you can hold. It is an agreement about what a physical state means.
2. On a wire in transit, a bit is a voltage: above an agreed level means one, below another agreed level means zero.
3. In main memory, a bit is an amount of electric charge sitting on a tiny capacitor.
4. In a solid-state drive, a bit is charge trapped inside an insulated island, where it stays for years without power.
5. On a spinning hard disk, a bit is the direction a microscopic patch of magnetic material points.
6. In every case the physical thing is continuous and messy, and the digital reading is made by comparing it against a threshold.
7. The cushion between what a sender guarantees and what a receiver requires is the **noise margin**, and it keeps the reading correct.

2.11.2 PLAIN - a picture in your head

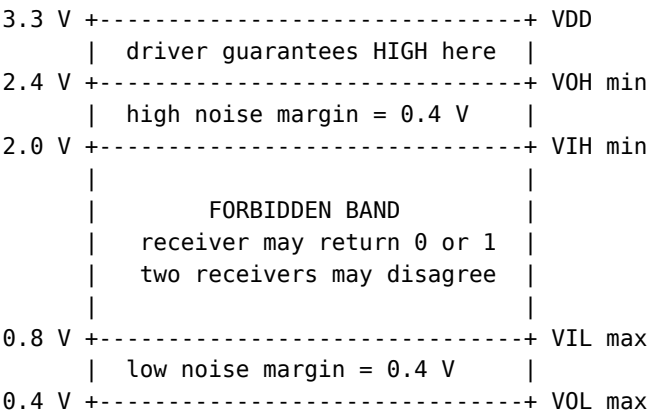
1. Picture a water tank with a line painted on the side, and a rule: above the line means yes, below means no.
2. Anyone filling it is told to fill well above the line, and anyone reading it answers yes for anything above the line.
3. The gap between fill-to-here and read-as-yes-from-here is the safety cushion, and ripples smaller than that cushion do not matter.
4. Shrink the tank to save water and the cushion shrinks with it, while the ripples stay the same size.

Where this comparison breaks

1. A tank has one line. Real logic has two, one for each direction, with a forbidden band between them.
2. The honest version: in that forbidden band a real receiver does not report an error. It returns a one or a zero effectively at random, and two receivers on the same wire may disagree.

2.11.3 PLAIN - a worked example

1. Here is a 3.3 V logic signal drawn to scale in volts.



```
      |  driver guarantees LOW here   |
0.0 V +-------------------------------+ GND
```

2. Now one bit of DRAM, using real numbers.

```
Cell capacitance  C = 10 femtofarads = 1e-14 F
Cell voltage      V = 1.1 V

Charge stored     Q = C * V = 1.1e-14 coulombs
Electrons         N = Q / 1.602e-19
                    = about 69,000 electrons
```

3. Sixty-nine thousand electrons is the entire physical existence of one bit of memory, and it leaks away within tens of milliseconds.

2.11.4 PLAIN - what is really happening inside

1. To write a DRAM bit, the controller raises the word line, which switches on the cell's single transistor.
2. That connects the tiny capacitor to a long wire called the bit line, and the capacitor charges or discharges to match.
3. Lower the word line, the transistor turns off, and the charge is sealed in. The bit is stored.
4. To read it, the bit line is first set exactly halfway between one and zero, then the word line is raised.
5. The cell's small charge nudges the bit line up or down by only tens of millivolts, and a sense amplifier detects which way it moved.
6. Reading destroys the stored charge, so the sense amplifier immediately writes the value back. Every read is also a rewrite.
7. Nothing here is truly digital. It is careful analogue engineering, with the decision made at the end, on purpose, at a chosen moment.

2.11.5 TECHNICAL - the engineer's version

1. Storage mechanisms and their physical carriers, with real figures.

Technology	Physical carrier	Typical retention
SRAM cell	6 cross-coupled MOSFETs	while powered
DRAM cell	6 to 10 fF capacitor	32 to 64 ms
NAND flash	trapped gate charge	about 1 to 10 years
Hard disk	magnetic domain	decades

2. DRAM sense amplifier input is on the order of 50 to 150 mV of bit line swing, against a precharge reference of half the supply.
3. NAND flash stores several bits per cell using more threshold levels: SLC 2 levels, MLC 4, TLC 8, QLC 16, and PLC 32 in research.
4. Each extra level halves the voltage window per state, which is exactly why QLC has lower endurance and needs stronger error correction.
5. Modern NAND relies on LDPC error-correcting codes in the controller, because the raw bit error rate is far too high to use directly.
6. Row hammer, published by Yoongu Kim and colleagues in 2014, showed that repeatedly activating one DRAM row flips bits in neighbouring rows.
7. That is a direct consequence of everything in this chapter: bits are charge, charge leaks, and neighbours are very close together.
8. Mitigations include Target Row Refresh, higher refresh rates and on-die ECC in DDR5. Research since 2020 has defeated several of them, so this is active research, not a solved problem.

9. Cosmic-ray neutrons and alpha particles also flip DRAM bits. Server memory uses ECC, usually single-error-correcting and double-error-detecting.
10. Inspect memory error events on Linux with `edac-util -v`, or with `rasdaemon` and `ras-mc-ctl --summary`.

2.11.6 WORDS - remember these

1. Bit — the smallest piece of information, a yes or a no — a binary digit, physically encoded as a distinguishable state.
2. Threshold — the level that decides one from zero — VIH and VIL, the guaranteed input decision points.
3. Sense amplifier — the circuit that reads a very faint memory bit — a differential amplifier resolving tens of millivolts of bit line swing.
4. Word line — the wire selecting one row of memory cells — the gate control line activating access transistors across a row.
5. Bit line — the wire carrying the value in or out — the shared column line joining cells to sense amplifiers.
6. ECC — extra bits that catch and fix memory errors — error-correcting code, commonly SECDED for DRAM and LDPC for NAND.
7. Row hammer — poking memory until neighbours change — a disturbance fault from repeated row activation, first published in 2014.

2.98 Common wrong ideas

1. Wrong: Electrons travel through wires at the speed of light. Right: Individual electrons crawl, usually under a tenth of a millimetre per second. The field that pushes them travels near light speed, and that is what makes the lamp light instantly.
2. Wrong: The bulb uses up the current, so less comes back than went in. Right: Exactly the same current returns, because charge is conserved. What the bulb uses is energy, which the charge carries and gives up as heat and light.
3. Wrong: Higher voltage always means more power. Right: Power is voltage times current. A 10,000 V static spark carries microjoules and is harmless, while a 12 V car battery at 400 A can melt a spanner.
4. Wrong: Static electricity is a different kind of electricity from current in a wire. Right: It is the same electrons and the same charge. The only difference is that static charge sits still on an insulator instead of flowing through a conductor.
5. Wrong: An input pin with nothing connected reads zero. Right: It floats. Its voltage is undefined and wanders with nearby noise, so it can read one, zero, or both within a microsecond, and it makes the chip draw extra current.
6. Wrong: Ground is a real place where electricity is disposed of. Right: Ground is whichever node we choose to call zero volts. A phone on battery has a ground that touches nothing in the earth at all.
7. Wrong: A capacitor lets direct current pass through it. Right: No charge ever crosses the insulating gap. Current appears to flow because charge accumulates on one plate and leaves the other.
8. Wrong: Digital circuits are immune to noise. Right: They tolerate noise up to the noise margin and then fail completely. Below the margin errors vanish; above it bits flip, which is why error-correcting codes exist.
9. Wrong: Ohm's law applies to everything electrical. Right: It applies to ohmic materials over a limited range. Diodes are exponential, lamps change resistance as they heat, and transistors are controlled devices, not resistors.
10. Wrong: Chips get hot because of poor design, and better engineering could remove the heat. Right: Computing physically consists of charging and discharging capacitance. The energy has to go somewhere, and it goes into heat. It can be reduced, never abolished.

2.99 Chapter summary in 20 lines

1. Atoms have a nucleus of protons and neutrons surrounded by electrons, and charge is the property that makes them push and pull.
2. In a metal one outer electron per atom is loosely held, and that crowd of free electrons is what carries current.
3. Voltage is energy per unit charge, current is charge per second, and resistance is how strongly a material fights the flow.
4. Ohm's law, from Georg Ohm in 1827, ties them together as V equals I times R, and holds only for ohmic materials.
5. Power is V times I, also I squared R, and almost all power delivered to a chip leaves it again as heat.
6. Heat comes from electrons colliding with the lattice, and inside chips mainly from charging capacitance billions of times a second.
7. A circuit must be a closed loop because charge cannot pile up at the end of a wire, and ground is simply the node chosen as zero volts.
8. Indian mains is 230 V RMS at 50 Hz, peaking near 325 V, and an adapter rectifies, smooths, chops, transforms and regulates it down.
9. ATX supplies 12 V, 5 V, 3.3 V and 5 V standby, while USB Power Delivery 3.1 has reached 48 V and 240 W since 2021.
10. Conductors have free carriers, insulators have a wide band gap, and semiconductors sit between, with silicon at 1.12 eV.
11. Doping silicon with phosphorus or boron makes n-type or p-type material, and the junction between them is the basis of every chip.
12. A resistor sets a known voltage drop for a given current, provides pull-ups and pull-downs, and limits LED current.
13. A floating input is a genuine bug, because a MOS gate has near-infinite input resistance and follows any stray noise.
14. A capacitor stores charge on two plates, charges along an exponential curve with time constant RC, and decouples power locally.
15. One DRAM bit is about 69,000 electrons on a few femtofarads, which leaks and must be refreshed within 32 ms on DDR5.
16. An inductor stores energy in a magnetic field and resists changes in current, which is what makes buck converters and VRMs possible.
17. A diode conducts one way only, costing about 0.7 V in silicon; LEDs emit photons on recombination and photodiodes do the reverse.
18. Digital beat analogue because every stage regenerates a clean level, so errors stop accumulating while noise stays inside the margin.
19. A quartz crystal, using the piezoelectric effect found by the Curie brothers in 1880, gives the stable reference a PLL multiplies to gigahertz.
20. At 3.5 GHz one clock period is 286 picoseconds, in which a board signal travels four to five centimetres, a hard limit on chip size.

3

CHAPTER

KB

From Sand to Silicon — Rock, Furnace, Wafer, Fab

3.0 What this chapter gives you

1. You will be able to say where the silicon in a chip comes from, naming the rock and the furnace.
2. You will be able to explain why 99 percent pure is useless here.
3. You will be able to describe how a single crystal is grown, and why the whole ingot must be one unbroken lattice.
4. You will be able to walk from a 265 kilogram crystal to a 775 micrometer polished wafer, naming every cut.
5. You will be able to explain photolithography, and why shorter wavelength light gives smaller features.
6. You will be able to explain how extreme ultraviolet light is made from a falling tin droplet.
7. You will be able to explain doping, deposition, the copper wiring stack, testing, dicing and packaging.
8. You will be able to say honestly what "3 nm" means in 2026.
9. You will be able to explain why leading-edge chip making sits in a handful of buildings on Earth.

3.1 Where silicon comes from

3.1.1 PLAIN – in simple words

1. Silicon is a chemical element, a basic building block of matter, like iron or oxygen.
2. About 28 percent of the mass of the Earth's rocky shell is silicon. Only oxygen is more common.
3. But you will never dig up a lump of pure silicon anywhere on Earth.
4. Every silicon atom in the ground is already bound to oxygen, tightly.
5. Silicon plus oxygen makes silica. Silica plus metals makes the silicate minerals, which is most rock.
6. So the job is not to find silicon. The job is to tear the oxygen off it.
7. The purest common form of silica is the mineral quartz. That is what we mine.

3.1.2 PLAIN – a picture in your head

1. Think of silicon as a person already holding hands with two oxygen partners, and refusing to let go.
2. Rock is that crowd, frozen solid. No silicon atom stands alone.
3. Quartz is the tidiest part of the crowd. Beach sand is a messier part, with shell and iron mixed through.
4. Where this comparison breaks: atoms do not want anything. The real fact is that the silicon-oxygen bond is strong, near 800 kilojoules per mole.

3.1.3 PLAIN – a worked example

1. Take a kilogram of average continental crust and sort it by mass.

Element	Share of crust by mass
Oxygen	about 46 percent
Silicon	about 27.7 percent
Aluminum	about 8 percent
Iron	about 5 percent

2. So that kilogram holds roughly 277 grams of silicon, and none of it is loose.
3. Typical beach sand runs 90 to 99 percent silica. Chip-grade quartzite runs above 99.5 percent.
4. That gap of a few percent is why we mine specific rock instead of a beach.

3.1.4 PLAIN - what is really happening inside

1. Silica is silicon dioxide: one silicon atom joined to two oxygen atoms, in an endless solid network.
2. In quartz that network is a regular crystal with little else mixed in.
3. In granite and clay the same units combine with sodium, potassium, aluminum, iron and magnesium. Those are silicates.
4. Iron is the enemy. Iron atoms inside silicon later act as traps that ruin the electrical behaviour we want.
5. So mining is really sorting. We look for rock where nature already sorted for us.
6. The ore must also be lumpy, because the furnace needs gas to flow up through the charge. Powder would choke it.

3.1.5 TECHNICAL - the engineer's version

1. Silicon, atomic number 14, is the second most abundant crustal element at about 27.7 percent by mass, after oxygen.
2. It does not occur natively, only as silicon dioxide and as the silicate mineral families.
3. Feedstock is lump quartz or quartzite, sized roughly 10 to 100 millimetres, specified above 99 percent silicon dioxide.
4. Limits are set on iron, aluminum, calcium, titanium, boron and phosphorus.
5. Boron and phosphorus matter most. They are dopants, and they resist later chemical removal.
6. Producing regions include Norway, Spain, Turkey, Brazil, China, Russia and the United States.

Material	What it is	Typical purity
Beach sand	Mixed sediment	90 to 99 pct silica
Silica sand	Washed and graded	98 to 99.5 pct
Quartzite lump ore	Metamorphic rock	above 99.5 pct
Spruce Pine quartz	Ultra-high purity	above 99.99 pct

7. The honest version: Spruce Pine quartz, in North Carolina, is famous, but it is not usually the feedstock that becomes the chip.
8. It is mainly melted into the fused-quartz crucibles used later in crystal growth. Do not repeat the claim that chips start as Spruce Pine sand.

3.1.6 WORDS - remember these

1. Silica — silicon joined to oxygen — silicon dioxide, SiO2, the network solid forming quartz.
2. Silicate — silica with metals mixed in — a mineral family built on silicon-oxygen tetrahedra.
3. Quartz — the clean crystal form of silica — crystalline SiO2, the preferred carbothermic feedstock.
4. Quartzite — hard rock made mostly of quartz — metamorphosed sandstone supplied as graded lump ore.

3.2 First refining: the submerged-arc furnace

3.2.1 PLAIN - in simple words

1. We have crushed quartz. Now we pull the oxygen off, using carbon and a very large amount of heat.
2. Carbon takes oxygen readily. Hot enough, it strips oxygen from silicon and leaves as carbon monoxide gas.
3. The machine is a submerged-arc furnace: a huge lined bowl with thick carbon rods pushed into the charge.
4. Electricity arcs through the material and heats it to around 1,900 degrees Celsius, hotter than lava.
5. Molten silicon collects at the bottom and is drained through a tap hole.
6. What comes out is metallurgical-grade silicon, about 98 to 99.5 percent pure.
7. That sounds excellent. For a chip it is hopelessly dirty.

3.2.2 PLAIN - a picture in your head

1. Picture a stone pot several metres across, filled with gravel, coal and wood chips.
2. Three carbon rods, each as thick as a tree trunk, are pushed down into the mixture.
3. Where the current jumps it makes an arc, and that arc sits buried inside the charge. That is "submerged arc".
4. Where this comparison breaks: nothing is cooking. This is a chemical reduction reactor, with reactions stacked in temperature zones.

3.2.3 PLAIN - a worked example

1. Here is the overall reaction, written the simple way.

```
SiO2   +   2 C    ->    Si     +   2 CO
quartz     carbon       silicon    carbon monoxide
```

2. Silicon dioxide is near 60 grams per mole; silicon is near 28. So 60 kilograms of quartz gives at most 28 kilograms of silicon.
3. Real furnaces recover roughly 80 to 90 percent. Some silicon escapes as silicon monoxide gas.
4. Energy cost is roughly 11 to 13 kilowatt hours per kilogram of product.
5. A large furnace runs at 20 to 40 megawatts, continuously, for years. It is never switched off if that can be avoided.

3.2.4 PLAIN - what is really happening inside

1. The charge is quartz plus carbon sources: coal, charcoal, petroleum coke and wood chips.
2. Wood chips are added for a physical reason. They keep the charge porous so gas can escape.
3. In the hot lower zone, around the arc, silicon dioxide is reduced to liquid silicon.
4. In the cooler upper zone, escaping silicon monoxide reacts with carbon to form silicon carbide, which sinks and reacts further.
5. That recycling is why the process works. Without it, far too much silicon would leave as gas.
6. The honest version: the tidy equation is a summary, not a description. It is a coupled set of reactions across a temperature gradient.
7. Impurities stay with the metal. Iron, aluminum and calcium dissolve in the liquid silicon.

3.2.5 TECHNICAL - the engineer's version

1. This is carbothermic reduction of silicon dioxide in a three-phase submerged-arc furnace, with Soderberg or prebaked graphite electrodes.
2. Arc temperature is around 1,900 to 2,000 degrees Celsius. Silicon melts at 1,414 degrees Celsius, so the product is liquid.
3. The dominant intermediate is silicon carbide.
4. Output is metallurgical-grade silicon, written MG-Si, at 98 to 99.5 percent.

5. Ladle refining with oxygen and slag additions removes some aluminum and calcium. It does not remove boron or phosphorus.

Grade	Silicon purity	Main use
Ferrosilicon	alloy, not pure	Steel making
MG-Si	98 to 99.5 pct	Alloys, silicones
UMG-Si	99.9 to 99.999 pct	Some solar cells
Polysilicon EG	99.9999999 pct up	Chips

6. World output is millions of tonnes per year. Only a small share reaches semiconductor purity.
7. Silica fume, the dust caught from the offgas, is sold as a concrete additive. It is a revenue stream, not waste.

3.2.6 WORDS - remember these

1. Carbothermic reduction — using carbon and heat to strip oxygen — carbon as the reducing agent at high temperature.
2. Submerged-arc furnace — a furnace whose arc is buried in the material — a three-phase electrode furnace.
3. Metallurgical-grade silicon — first-pass silicon metal — MG-Si at 98 to 99.5 percent, the input to chemical purification.
4. Silica fume — fine dust caught from the exhaust — amorphous silicon dioxide sold as a concrete admixture.

3.3 Getting to nine nines

3.3.1 PLAIN - in simple words

1. Silicon at 99 percent has one foreign atom in every hundred.
2. For a chip we need roughly one foreign atom in every billion, and for some impurities far fewer.
3. You cannot get there by melting and skimming. You need chemistry.
4. The trick is to turn solid silicon into a liquid chemical, clean it, then turn it back into solid silicon.
5. The chemical is trichlorosilane. It boils just under 32 degrees Celsius, so we can distil it.
6. Then we pipe the cleaned vapour over hot silicon rods, and silicon grows back onto them, atom by atom.
7. The result is polysilicon, at nine nines purity or better: 99.9999999 percent.

3.3.2 PLAIN - a picture in your head

1. Imagine a bucket of mixed salt and sand, and you want pure salt.
2. You dissolve what you can, filter out what will not dissolve, then boil the water away. The salt comes back clean.
3. Making trichlorosilane is the dissolving step. Distillation is the filtering step. Growing rods is the boiling-dry step.
4. Where this comparison breaks: salt water separates in one pass. Nine nines needs columns in series, some tens of metres tall, plus days of deposition.

3.3.3 PLAIN - a worked example

1. Purity numbers are hard to feel, so convert them into countable things.

Purity	Foreign atoms	In everyday terms
99 pct (2N)	1 in 100	1 bad brick in a wall
99.9999 pct (6N)	1 in 1 million	1 person in a city
99.9999999 pct (9N)	1 in 1 billion	1 second in 32 years
11N	1 in 100 billion	1 card in a stack

2. That stack of 100 billion cards would stand about 15,000 kilometres tall, with exactly one wrong card in it.
3. Now the humbling part. A cubic centimetre of silicon holds about 5 times 10 to the power 22 atoms.
4. So even at nine nines, that cubic centimetre still holds around 50 thousand billion foreign atoms.
5. Nine nines is not "no impurities". It is "impurities below the level where they change the behaviour we care about".

3.3.4 PLAIN - what is really happening inside

1. Step one. Ground metallurgical silicon meets hydrogen chloride gas at around 300 degrees Celsius in a fluidized bed.
2. Out comes trichlorosilane. Impurities come along as their own chlorides, with their own boiling points.
3. Step two. The mixture goes through distillation columns. Boron and phosphorus compounds are hardest to strip and set the final purity.
4. Step three. The cleaned liquid is vaporized, mixed with hydrogen, and fed into a sealed bell-jar reactor.
5. Inside stand thin silicon filaments heated to about 1,100 degrees Celsius. The gas breaks down there, silicon sticks, hydrogen chloride leaves.
6. Over two to four days the filaments grow into grey rods 15 to 20 centimetres across, which are broken into chunks.
7. This is polysilicon: extremely pure, but made of many small crystals pointing in random directions. We fix that next.

3.3.5 TECHNICAL - the engineer's version

1. The dominant route is the Siemens process, developed by Siemens in Germany in the late 1950s and still the workhorse in 2026.
2. Hydrochlorination runs at roughly 300 to 350 degrees Celsius. Trichlorosilane boils at 31.8 degrees Celsius, making distillation practical.
3. Deposition is chemical vapour deposition onto filaments at about 1,100 to 1,150 degrees Celsius inside a water-cooled bell jar.
4. Conversion per pass is low, near 20 percent, so unreacted gas and silicon tetrachloride are recycled.

Grade	Purity	Typical use
Solar, multicrystalline	7N to 8N	Standard PV cells
Solar, monocrystalline	9N to 10N	Higher-eff PV
Electronic grade	10N to 11N	Integrated circuits

5. The main alternative is the fluidized bed reactor route, fed with monosilane, giving granular polysilicon at roughly one tenth of the heating electricity.
6. The honest version: "eleven nines" is a bulk metallic-impurity figure, not a statement about every element.
7. Dopants such as boron and phosphorus are specified separately, in parts per billion atomic or as resistivity. Carbon and oxygen are separate again.
8. Metrology includes glow discharge mass spectrometry, inductively coupled plasma mass spectrometry, and photoluminescence.

3.3.6 WORDS - remember these

1. Trichlorosilane — the liquid we clean instead of the metal — SiHCl3, boiling point 31.8 degrees Celsius.
2. Siemens process — growing pure silicon back onto hot rods — CVD from trichlorosilane and hydrogen at about 1,100 degrees Celsius.
3. Polysilicon — very pure silicon made of many small crystals — electronic-grade polycrystalline silicon, the feedstock for crystal growth.
4. Nine nines — 99.9999999 percent pure — 9N, one foreign atom per billion by the stated metric.

3.4 Growing a single crystal

3.4.1 PLAIN - in simple words

1. Our polysilicon chunks are pure, but their internal arrangement is a mess.
2. They are many small crystals at random angles. Where two meet there is a seam called a grain boundary.
3. Electrons behave badly at those seams. A chip needs one repeating pattern across the whole piece, with no seams.
4. So we melt the polysilicon completely and grow it back as one single crystal.
5. We dip a small perfect seed crystal into the melt and slowly pull it upward while turning it.
6. Liquid silicon freezes onto the seed and copies its arrangement exactly.
7. Keep pulling for a day or more and you get a cylinder of one crystal, called a boule or ingot.
8. A 300 millimetre boule is about 2 metres long and weighs about 265 kilograms.

3.4.2 PLAIN - a picture in your head

1. Think of pulling hot toffee slowly out of a pan so it stretches into a smooth rod instead of breaking.
2. Now add one rule: it can only harden by copying the pattern already above it. The seed is that template.
3. Pull too fast and the pattern breaks. Pull too slow and the rod grows fat. Spinning keeps the heat even all round.
4. Where this comparison breaks: toffee just cools. Each silicon atom must find one site in a repeating three-dimensional lattice. And the crucible slowly dissolves into the melt.

3.4.3 PLAIN - a worked example

1. Here is a full run for a 300 millimetre crystal.

```
Charge polysilicon into crucible   about 250 to 450 kg
Melt down under argon              melts at 1414 deg C
Dip the seed crystal               seed a few mm across
Neck: pull thin and fast           neck about 3 mm wide
Grow the crown, widen out          out to 300 mm across
Grow the body                      about 2 m of cylinder
Taper the tail and lift clear      whole run 1 to 3 days
```

2. Pull rate through the body is around 0.5 to 1.5 millimetres per minute.
3. Crucible and crystal rotate in opposite directions, a few turns per minute each.
4. The neck stage looks wasteful but is essential. Pulling thin and fast forces line defects out to the surface, where they vanish.
5. That trick came from William Dash at General Electric around 1958. It is why modern crystals can be dislocation-free.

3.4.4 PLAIN - what is really happening inside

1. The polysilicon sits in a fused-quartz crucible inside a graphite support, under argon so no air gets in.
2. Heaters bring everything above 1,414 degrees Celsius, the melting point of silicon.
3. A measured amount of boron or phosphorus goes into the melt now. This is the first doping step.

4. As the seed rises, liquid at the interface freezes, and each atom locks into the position dictated by the seed.
5. The seed's lattice direction therefore sets the direction of the whole boule. That is crystal orientation.
6. Impurities prefer to stay liquid, so the crystal purifies itself as it grows. That is segregation, and it leaves the tail dirtier than the head.
7. The quartz crucible slowly dissolves, so oxygen enters the melt and ends up in the crystal. This is unavoidable here.

3.4.5 TECHNICAL - the engineer's version

1. The method is the Czochralski process, named for Jan Czochralski, the Polish chemist who described the pulling technique in 1916.
2. It was adapted to semiconductors at Bell Telephone Laboratories by Gordon Teal and John Little around 1950.
3. Orientation is given by Miller indices. The two common wafer orientations are

(100) and (111).

4. (100) dominates for modern CMOS logic, because the silicon to silicon-dioxide interface has lower trapped charge on that plane.
5. Resistivity is set by melt doping: boron gives p-type; phosphorus, arsenic or antimony give n-type.
6. Interstitial oxygen lands near 10 to 20 parts per million atomic. It gives internal gettering, which traps metals away from the device region.

Diameter	Typical body length	Typical mass
150 mm	about 1 to 1.5 m	about 30 to 50 kg
200 mm	about 1.5 to 2 m	about 100 kg
300 mm	about 2 m	about 265 kg

7. The alternative is float-zone growth. A radio-frequency coil melts a narrow band passed along a rod, with no crucible touching the melt.
8. Float-zone silicon has far lower oxygen and much higher resistivity, so it suits power devices, radiation detectors and some radio-frequency parts.
9. It is limited to about 200 millimetres and costs more, so almost all logic and memory wafers are Czochralski.
10. The honest version: a Czochralski crystal is dislocation-free, not defect free. Point defects, vacancy clusters and dissolved oxygen remain.

3.4.6 WORDS - remember these

1. Single crystal — one unbroken repeating pattern of atoms — a monocrystalline solid with no grain boundaries.
2. Grain boundary — the seam where two crystals meet — a planar defect that scatters carriers and traps charge.
3. Boule — the big grown cylinder — the as-grown single-crystal ingot before machining.
4. Czochralski — the dip-and-pull growth method — crucible-based melt pulling, the dominant industrial route.
5. Float zone — crucible-free growth by a moving molten band — FZ silicon, low oxygen, high resistivity, limited diameter.

3.5 From boule to wafer

3.5.1 PLAIN - in simple words

1. We have a two-metre crystal cylinder. Chips are built on thin flat slices of it.
2. First the cone-shaped ends are cut off and thrown back into the melt pot.
3. Then the cylinder is ground on the outside to exactly the right diameter all the way along.
4. A small notch is ground into the side. It tells every later machine which way the crystal pattern faces.
5. Then the cylinder is sliced, using one very long wire coated with cutting grit.
6. Sawing leaves slices rough and damaged, so they are ground flat, etched in acid, and polished.
7. The result is a wafer: a disc about three quarters of a millimetre thick, with a mirror surface.

3.5.2 PLAIN - a picture in your head

1. Think of a very long, expensive salami sliced into perfect discs by a machine that never wobbles.
2. Except the slicer is not a blade. It is one wire, hundreds of kilometres long, wound across grooved rollers.
3. The wire runs at speed carrying hard particles, cutting the whole ingot into hundreds of slices in one pass.
4. Where this comparison breaks: salami only needs to look right. A wafer must be flat to a fraction of a wavelength of light, or the printing step goes out of focus.

3.5.3 PLAIN - a worked example

1. Follow one 300 millimetre boule through the shop.

```
Boule 2 m, 265 kg
  -> crop the ends       lose about 10 to 20 pct
  -> grind to 300.0 mm   lose a few mm of radius
  -> grind the notch     one small notch on the edge
  -> wire saw            about 900 to 1200 wafers
  -> lap and edge-round  remove the saw damage
  -> acid etch           remove more damage
  -> CMP one side        mirror finish
  -> clean and pack      sealed cassettes
```

2. Slicing loses material as kerf, the width of the cut. Diamond wire kerf is roughly 100 to 150 micrometers.
3. A finished 300 millimetre wafer is 775 micrometers thick, plus or minus about 25. That is standardized, not a guess.
4. So each wafer plus kerf uses close to 0.9 millimetres of boule, giving around one thousand wafers from 2 metres.
5. A blank polished 300 millimetre prime wafer sells for very roughly 100 to 150 US dollars in volume.

3.5.4 PLAIN - what is really happening inside

1. Cropping removes the crown and tail, where diameter is wrong and impurity levels are off.
2. The notch is ground after an X-ray measurement finds the true crystal direction.
3. Older small wafers used a straight flat on the edge. Modern 200 and 300 millimetre wafers use a notch, which wastes less area.
4. In wire sawing the wire does not cut. The abrasive does. Sawing leaves a damaged layer tens of micrometers deep, full of microcracks.
5. Lapping presses wafers between plates with slurry to make both faces parallel and flat. Edge rounding stops chipping and particle shedding.
6. Acid etching removes the crushed layer. Chemical-mechanical polishing then finishes one face with a soft pad and an alkaline silica slurry.
7. Final cleaning ends with drying in filtered air. From here the wafer never touches ordinary air again.

3.5.5 TECHNICAL - the engineer's version

1. Wafer geometry is set by SEMI standards: diameter, thickness, flatness, edge profile and notch dimensions.

Diameter	Thickness	Area	Notes
150 mm (6 in)	675 um	177 sq cm	Legacy, power, MEMS
200 mm (8 in)	725 um	314 sq cm	Analog, auto, RF
300 mm (12 in)	775 um	707 sq cm	All leading-edge
450 mm (18 in)	925 um	1590 sq cm	Never commercialized

2. Thickness rises with diameter for stiffness. A thin 300 millimetre wafer would sag and crack under its own weight.
3. Going from 200 to 300 millimetres multiplies usable area by about 2.25 for less than 2.25 times the cost. That is the case for larger wafers.
4. The 450 millimetre transition stalled. The Global 450 Consortium, based in Albany, New York, effectively collapsed around 2016 and 2017.
5. Reported reasons: huge tool redevelopment cost, EUV absorbing the industry's capital, and reluctance to fund a change helping mainly the largest players.
6. As of 2026 there is no credible 450 millimetre roadmap.
7. Processed wafer prices at leading nodes, reported by analysts in early 2026. These are negotiated and vary by customer.

Node	Approx price per wafer
28 nm	about 3,000 USD
7 nm	about 9,500 USD
5 nm class	about 18,500 USD
2 nm class	about 30,000 USD

8. The honest version: the raw silicon disc is a rounding error. At the 2 nanometre class it is well under 1 percent of the finished wafer price.
9. When people say "sand is cheap, so chips should be cheap", this table is the answer. The material is cheap. The processing is not.

3.5.6 WORDS - remember these

1. Kerf — the material lost to the cut — the saw slot width, roughly 100 to 150 micrometers with diamond wire.
2. Notch — the alignment mark on the wafer edge — a SEMI-specified notch replacing the older primary flat.
3. CMP — polishing with chemistry and rubbing together — chemical-mechanical planarization, using a pad and reactive slurry.
4. Prime wafer — the best grade sold — a polished wafer meeting full flatness, particle and defect specification.

3.6 The cleanroom

3.6.1 PLAIN - in simple words

1. The features on a modern chip are far smaller than a speck of dust.
2. One particle in the wrong place can short two wires or block a printing step, and kill that chip.

3. So chip factories are built inside rooms far cleaner than an operating theatre.
4. Air is pushed down through ceiling filters, constantly, and pulled out through the floor.
5. The dirtiest thing in the room is the human. People wear sealed suits, called bunny suits, that keep skin flakes and breath in.
6. The building must be still, because even small vibrations blur the printing step.
7. The rinse water is purer than drinking water by an enormous margin.
8. All of this is why a leading-edge fab costs billions before it makes a single chip.

3.6.2 PLAIN - a picture in your head

1. Picture an operating theatre, made a hundred times cleaner, inside a concrete box the size of several football pitches.
2. The whole ceiling is a filter. Air falls straight down like an invisible waterfall, sweeping particles to the floor.
3. Wafers do not ride through the open room. They travel in sealed pods on overhead rails and dock onto machines.
4. Where this comparison breaks: a theatre worries about living germs. A fab does not care whether a particle is alive. It cares about size.

3.6.3 PLAIN - a worked example

1. Suppose one chip is 1 square centimetre, and the process leaves 0.1 killer defects per square centimetre.
2. A simple model gives a good fraction of e to the power of minus 0.1, about 0.90. So 90 percent survive.
3. Now make the chip ten times bigger, 10 square centimetres, like a large graphics processor.
4. The same defect density gives e to the power of minus 1, about 0.37. Only 37 percent survive.
5. Same factory, same wafer. Bigger chips are punished brutally, which drives the move to chiplets in section 3.10.

3.6.4 PLAIN - what is really happening inside

1. Cleanliness is measured by counting particles in a fixed volume of air, at a fixed particle size.
2. The cleanest areas allow about ten particles of 0.1 micrometer or larger in a whole cubic metre of air.
3. To hold that, room air is replaced hundreds of times per hour, needing enormous fan power running permanently.
4. Not all of the fab is equally clean. The tightest classes apply where wafers are exposed, especially at the lithography tools.
5. Water is purified until it barely conducts, then deoxygenated and filtered again just before use. A big fab uses millions of litres a day.
6. Vibration matters because the printing tool must hold alignment to a few nanometres while a heavy stage moves fast.
7. So the lithography bay sits on an isolated slab with active dampers, and the fab is not built next to a railway line.

3.6.5 TECHNICAL - the engineer's version

1. Air cleanliness is classified by ISO 14644-1. The older United States standard, Federal Standard 209E, was withdrawn in 2001.

ISO class	Max particles per cu m at 0.1 um	Air changes per hour
ISO 1	10	500 to 750
ISO 2	100	500 to 750
ISO 3	1,000	500 to 750
ISO 5	100,000	250 to 300

2. ISO 3 corresponds roughly to the old Class 1, and ISO 5 to the old Class 100.
3. Modern practice is a mini-environment strategy. The bay runs at ISO 5 or 6 while the wafer sees ISO 1 inside a sealed pod and inside the tool.
4. Wafers travel in a Front Opening Unified Pod, a SEMI-standard carrier, moved by an overhead hoist transport system.
5. Ultrapure water is specified at 18.2 megaohm-centimetre at 25 degrees Celsius, the theoretical maximum for pure water.
6. Molecular contamination is controlled too. Ammonia poisons chemically amplified photoresist, so lithography bays use chemical filtration.
7. Vibration is specified against generic vibration criteria curves. Advanced lithography demands VC-E or better, near 3 micrometres per second RMS.
8. TSMC stated in 2025 and 2026 that a 2 nanometre fab module of about 20,000 wafer starts per month costs roughly 25 to 35 billion US dollars.
9. TSMC guided 2026 capital spending to roughly 60 to 64 billion US dollars. Most of that is tools; a single EUV scanner can exceed 200 million dollars.
10. Standard, convention or implementation detail: ISO 14644-1 is a standard, the bunny suit is a convention, and each bay's class is an implementation detail of that fab.

3.6.6 WORDS - remember these

1. Cleanroom — a room with almost no dust — a controlled environment classified by ISO 14644-1 particle limits.
2. Bunny suit — the full-body coverall — a cleanroom garment containing human-generated particles and fibres.
3. FOUP — the sealed wafer box — Front Opening Unified Pod, the standard 300 millimetre carrier.
4. Ultrapure water — water with nothing else in it — UPW at 18.2 megaohm-centimetre at 25 degrees Celsius.
5. Defect density — how many killer flaws per area — D0 in defects per square centimetre, the main input to yield models.

3.7 Photolithography, the heart of it

3.7.1 PLAIN - in simple words

1. A chip is a stack of patterned layers. Photolithography puts the pattern on each layer.
2. It is printing with light, and it is the most important step in the factory.
3. First we coat the wafer with a liquid that hardens into a thin film and reacts to light. That is photoresist.
4. We spin the wafer very fast so the liquid spreads into an even layer, much thinner than a hair.
5. Then we shine light through a patterned plate, called a mask or reticle, onto the resist.
6. Where light lands, the resist changes chemically. A developer washes away either the lit parts or the unlit parts.
7. We etch through the holes left behind, then strip the remaining resist off.
8. That is one layer. A modern chip repeats this cycle, with different masks, many dozens of times.

3.7.2 PLAIN - a picture in your head

1. Think of spray painting a wall through a cardboard stencil. The mask is the stencil; the resist records where the spray hit.
2. Now change two things. The stencil is four times bigger than the pattern you want, and a lens shrinks the image on the way down.
3. And the stencil covers only one small rectangle. The machine prints it, steps sideways, prints again, and repeats across the wafer.
4. That is why the tool is a stepper, or a scanner when it sweeps rather than flashes.

5. Where this comparison breaks: paint stays where it lands. Photoresist never becomes part of the chip. It is destroyed and removed on every layer.

3.7.3 PLAIN - a worked example

1. Here is one complete layer cycle, in order.

```
1  clean and prime      make surface accept resist
2  spin coat resist     3000 rpm, film 30 to 200 nm
3  soft bake            drive off solvent
4  align to previous    match marks already on wafer
5  expose               light through the reticle
6  post-exposure bake   drive the resist chemistry
7  develop              wash away the soluble parts
8  inspect and measure  check width and overlay
9  etch                 cut into the layer below
10 strip resist         remove the mask and clean
```

2. Now the print-and-repeat step. A standard exposure field is 26 by 33 millimetres, which is 858 square millimetres.
3. A 300 millimetre wafer is about 70,700 square millimetres, so the machine prints roughly 80 fields to cover one wafer.
4. If each field holds four dies of about 200 square millimetres, one wafer carries about 320 dies for that layer.
5. Then it does the whole thing again for the next layer. And again.

3.7.4 PLAIN - what is really happening inside

1. The resist is not a simple dye. In modern deep ultraviolet resists, light creates a small amount of acid inside the film.
2. The bake after exposure lets that acid move and trigger a chain reaction, changing the solubility of the polymer around it.
3. That amplification is why modest light flips a whole film. It is also why stray ammonia in the air ruins the pattern.
4. Alignment is separate and just as hard. Each new layer must line up with the layers already there, to a few nanometres.
5. The machine reads marks already on the wafer and adjusts stage position, rotation and scale before printing. The error left over is overlay.
6. Etching transfers the pattern downward. Wet etching uses liquids and eats sideways as well as down.
7. Dry etching uses a plasma, an electrically excited gas, and cuts almost straight down. That is why small features need it.

3.7.5 TECHNICAL - the engineer's version

1. Resolution follows the Rayleigh criterion.

```
CD  = k1 * wavelength / NA
DOF = k2 * wavelength / (NA * NA)

CD  = smallest printable half-pitch
NA  = numerical aperture of the lens
k1  = process factor, hard physical floor at 0.25
DOF = depth of focus, the focus error allowed
```

2. Three levers exist: shorter wavelength, higher numerical aperture, lower k1.
3. Single exposure cannot go below k1 of 0.25. That physical limit is the reason multi-patterning exists.
4. Note the second formula. Numerical aperture costs depth of focus quadratically, which is why wafer flatness is specified so tightly.

5. Reticles are typically 4x reduction, so 60 nanometres on the mask prints at 15 nanometres on the wafer.
6. Optical proximity correction distorts the mask shapes so the printed result is correct. Sub-resolution assist features help their neighbours print.
7. The honest version: the mask is not a picture of the chip. It is the computed input that yields the chip after diffraction.
8. History: the first commercial wafer stepper was the GCA DSW4800, introduced in 1978. Step-and-scan replaced steppers at advanced nodes in the 1990s.
9. A leading-node mask set is commonly quoted at 10 to 30 million US dollars. That is the main barrier to low-volume advanced chips.

Term	Meaning	Typical value
Field size	One printed rectangle	26 by 33 mm
Reduction	Mask to wafer scale	4x
Overlay	Layer to layer error	1 to 3 nm
Throughput	Wafers per hour	100 to 220

3.7.6 WORDS - remember these

1. Photoresist — light-sensitive coating — a polymer film whose solubility changes on exposure, via a photoacid generator.
2. Reticle — the patterned plate — a quartz plate with an absorber pattern, imaged at 4x reduction onto the wafer.
3. Stepper and scanner — print-and-repeat machines — step-and-repeat or step-and-scan exposure tools working field by field.
4. Numerical aperture — how widely the lens gathers light — NA, appearing in the Rayleigh resolution and depth of focus equations.
5. Overlay — how well layers line up — registration error between a printed layer and the layers beneath it.
6. OPC — deliberately distorting the mask — optical proximity correction, a computational step applied before mask writing.

3.8 The light

3.8.1 PLAIN - in simple words

1. The smallest thing you can print depends on the wavelength of your light. Shorter waves draw finer lines.
2. Chip making has therefore been a long march toward shorter wavelengths.
3. It started with mercury lamps giving ultraviolet light at 436 and then 365 nanometres.
4. Then came gas lasers at 248 nanometres, then 193 nanometres. These are deep ultraviolet.
5. Progress stalled at 193, so engineers filled the gap between lens and wafer with water.
6. Water bends light more than air, which shrinks the effective wavelength in the gap. That is immersion lithography.
7. When that ran out, they printed one layer more than once with offset patterns. That is multi-patterning.
8. Finally came extreme ultraviolet at 13.5 nanometres, fourteen times shorter than 193.

3.8.2 PLAIN - a picture in your head

1. Imagine drawing with a marker pen. A thick pen cannot draw thin lines, no matter how steady your hand.
2. Wavelength is the thickness of the pen tip. To draw finer, you need a finer tip.

3. Immersion is like drawing underwater, where the ink spreads less. You get a finer line from the same pen.
4. Multi-patterning is drawing every second line, letting it dry, then going back for the ones in between.
5. Extreme ultraviolet is not a finer pen. It is a different tool entirely, which no glass lens can focus.
6. Where this comparison breaks: a pen deposits ink. Light deposits nothing. The limit comes from diffraction, how waves spread past an edge.

3.8.3 PLAIN - a worked example

1. Put real numbers into the Rayleigh formula from section 3.7: half-pitch equals k1 times wavelength divided by numerical aperture.
2. Dry 193 nanometre tool, numerical aperture 0.93, k1 of 0.30: about 62 nanometres.
3. Immersion 193 nanometre tool in water, numerical aperture 1.35, since water has refractive index near 1.44 here: about 43 nanometres.
4. EUV at 13.5 nanometres, numerical aperture 0.33: about 12 nanometres. High numerical aperture EUV at 0.55: about 7 nanometres.
5. Notice the jump. Moving to EUV bought more resolution in one step than the previous twenty years of optical improvement.

3.8.4 PLAIN - what is really happening inside

1. Here is how EUV light is actually made. It is not a lamp and not a normal laser.

```
tin droplet generator
   |  drops of molten tin, about 25 um across
   |  released about 50,000 times per second
   v
[ pre-pulse laser ]  -> flattens the drop into a disc
   |
   v
[ main pulse laser ] -> vaporizes it into hot plasma
   |
   v
plasma over 200,000 degrees emits 13.5 nm light
   |
   v
collector mirror -> multilayer mirrors -> reticle
   |
   v
more mirrors -> wafer
```

2. A carbon dioxide laser of over ten kilowatts average power fires twice at each falling droplet.
3. The first shot squashes the sphere flat. The second turns it into plasma, an extremely hot ionized gas radiating at 13.5 nanometres.
4. Everything absorbs this light. Air absorbs it, glass absorbs it. So there are no lenses, and the path must be in vacuum.
5. Instead the machine uses mirrors of about forty to fifty alternating pairs of molybdenum and silicon, each layer a few nanometres thick.
6. Even those reflect only around 70 percent. After ten bounces most of the light is gone, which is why source power is such a struggle.
7. The reticle is a mirror too, not a transparent plate. That is a fundamental break from all earlier lithography.
8. Tin debris coats the collector and shortens its life, so hydrogen gas is flowed through to clean it.

3.8.5 TECHNICAL - the engineer's version

Generation	Wavelength	Source	Leading use
g-line	436 nm	Mercury lamp	1980s
i-line	365 nm	Mercury lamp	late 1980s to 90s
KrF DUV	248 nm	Krypton fluoride	mid 1990s to 2000s
ArF DUV	193 nm	Argon fluoride	2001 onward
ArF immersion	193 nm in water	ArF plus water	2007 onward
EUV	13.5 nm	Tin plasma	2019 onward

1. The 157 nanometre fluorine laser generation was researched hard and abandoned around 2003, because 193 immersion proved cheaper and better.
2. Immersion entered volume production around 2007, with water enabling numerical apertures up to 1.35.
3. EUV entered high-volume manufacturing in 2019, first at Samsung and TSMC for 7 nanometre class layers.
4. Low numerical aperture tools such as the ASML TWINSCAN NXE:3800E have numerical aperture 0.33 and are reported at 180 to 220 million US dollars.
5. High numerical aperture EUV, the TWINSCAN EXE:5200 family, has numerical aperture 0.55 and is reported at about 380 million US dollars per tool.
6. Anamorphic means the reduction differs by axis, 4x one way and 8x the other. The field halves to about 26 by 16.5 millimetres, so large dies are stitched.
7. ASML of Veldhoven, the Netherlands, is the only supplier of EUV scanners.
8. Zeiss SMT supplies the optics and TRUMPF the carbon dioxide laser, both of Germany. The tin source traces to Cymer, now part of ASML.
9. Where experts disagree: whether High-NA EUV beats low-NA plus double patterning on cost per layer. Intel has pushed High-NA; some analysts disagree. As of 2026 this is unsettled.
10. Established fact: EUV works and is in volume production. Active research: higher source power and dry resists. Marketing claim: any statement that a named node "requires" High-NA today.

3.8.6 WORDS - remember these

1. Deep ultraviolet — the 248 and 193 nanometre light — DUV from krypton fluoride and argon fluoride excimer lasers.
2. Immersion lithography — printing through water — a water film between final lens and wafer, raising NA above 1.
3. Multi-patterning — printing one layer in several passes — LELE, SADP and SAQP decomposition of a dense layer.
4. Extreme ultraviolet — 13.5 nanometre light — EUV, in the soft X-ray region, needing vacuum and reflective multilayer optics.
5. Laser-produced plasma — light made by vaporizing tin — LPP source, a dual carbon dioxide laser pulse at 50 kilohertz.
6. High-NA — the next EUV generation — numerical aperture 0.55 anamorphic optics with a halved exposure field.

3.9 The other steps

3.9.1 PLAIN - in simple words

1. Printing patterns is half the story. We also add and change material.
2. Doping means adding a tiny controlled amount of a foreign element to change how the silicon conducts.

3. The main method is ion implantation. We turn the dopant into charged atoms, accelerate them, and fire them into the wafer.
4. That smashes the crystal, so afterwards we heat the wafer briefly. The lattice repairs and the dopants settle into place.
5. Oxidation means growing a glass layer by heating silicon in oxygen. The wafer rusts, in a useful and controlled way.
6. Then three ways of laying material on top: chemical deposition, physical deposition, and one-atomic-layer-at-a-time deposition.
7. Finally the transistors are wired together, in a stack of copper layers above them, fifteen or more.
8. A modern chip goes through many hundreds of process steps and spends about three months inside the fab.

3.9.2 PLAIN - a picture in your head

1. Think of building a city on a plain, where you may only work from above and must finish each floor before the next.
2. Doping is changing the soil chemistry in specific plots so buildings there behave differently.
3. Chemical vapour deposition is a fine mist that reacts and settles everywhere, even inside narrow trenches.
4. Physical vapour deposition is shot-blasting from one direction. It coats what it can see and struggles down deep holes.
5. Atomic layer deposition is placing one tile at a time across the whole city. Very slow, very exact.
6. The metal stack is the road network built above the buildings, with vertical lifts between levels.
7. Where this comparison breaks: a city is built once. Wafer layers are built, partly cut away and rebuilt, and nothing buried is reachable again.

3.9.3 PLAIN - a worked example

1. Here is the damascene method, which is how the copper wiring is made.

```
Start: a flat insulating layer over the transistors

1 etch a trench and a via hole into the insulator
2 line the hole with a thin barrier film (TaN/Ta)
3 sputter a thin copper seed layer
4 electroplate copper until it overflows the trench
5 polish the whole surface flat with CMP
6 the only copper left is inside the trench

Result: a wire buried in glass, top surface perfectly flat
```

2. Notice what did not happen. We never etched copper. Copper is hard to etch cleanly, so we cut the shape first and fill it.
3. That is damascene, named after the inlay metalwork of Damascus. Doing trench and vertical connection in one fill is dual damascene.
4. Repeat fifteen or more times, with wires getting wider and thicker as you go up.
5. Bottom layers carry signals inside one circuit block. Top layers are thick and carry power across the chip.

3.9.4 PLAIN - what is really happening inside

1. Ion implantation: a gas of boron, phosphorus or arsenic is ionized, sorted by mass with a magnet, then accelerated.
2. Energies run from under one thousand electron volts for shallow layers up to millions for deep wells.
3. Dose is counted as ions per square centimetre, typically ten to the twelve through ten to the sixteen, and is very repeatable.

4. Annealing: the wafer is heated in seconds, or milliseconds, to around 1,000 degrees Celsius, to repair damage and activate dopants.
5. It must be brief, because heat also makes dopants diffuse and smear the tiny features we just made.
6. Oxidation: heating silicon in oxygen or steam at 800 to 1,200 degrees Celsius grows silicon dioxide, consuming some silicon.
7. Chemical vapour deposition: gases decompose on the hot wafer. Physical vapour deposition: an argon plasma knocks atoms off a target onto the wafer.
8. Atomic layer deposition: two gases are fed alternately, never together. Each pulse adds at most one atomic layer, so thickness is counted in cycles.

3.9.5 TECHNICAL - the engineer's version

Method	How it works	Typical use
Thermal oxidation	Grows from substrate	Gate oxide, isolation
LPCVD and PECVD	Gas reacts on surface	Nitride, oxide, poly
PVD sputtering	Ions knock off target	Barriers, seed, pads
ALD	One layer per cycle	High-k, liners, spacers

1. The gate insulator was silicon dioxide for about forty years. Intel replaced it with hafnium-based high-k plus metal gates at 45 nanometres in 2007.
2. The honest version: since 2007 "gate oxide" is largely historical. The layer is a hafnium oxide film laid by atomic layer deposition.
3. Interconnect switched from aluminum to copper. IBM announced copper interconnect in 1997, using dual damascene with CMP.
4. Copper needs a barrier, historically tantalum nitride and tantalum, because copper diffuses into silicon and poisons devices.
5. Dielectrics moved from silicon dioxide, near 3.9, to carbon-doped low-k films near 2.5 to 3.0, to cut wire capacitance.
6. Advanced logic in 2026 uses 15 to 20 metal levels, from roughly 20 to 30 nanometre pitch at the bottom to micrometres at the top.
7. Backside power delivery is the current change. Intel calls its version PowerVia, shipped in 18A; TSMC's equivalent arrives with A16.
8. Step counts: advanced logic is commonly described as more than 1,000 process steps with roughly 80 to 100 mask layers. Exact counts are confidential, so treat these as approximate.
9. Cycle time: a leading-edge logic wafer spends about 3 months in the fab, often quoted as 12 to 16 weeks. With packaging and test, four to five months.

3.9.6 WORDS - remember these

1. Doping — adding a trace element to change conduction — introducing acceptors or donors to set carrier type and concentration.
2. Ion implantation — firing dopant atoms into the wafer — mass-analyzed ion beam implantation, dose in ions per square centimetre.
3. Annealing — a short hot step to repair and settle — rapid thermal, spike, flash or laser anneal for damage repair and activation.
4. ALD — one atomic layer per cycle — atomic layer deposition, self-limiting surface reactions giving exact conformal thickness.
5. Damascene — cut the shape then fill it — trench-first copper metallization finished by CMP, in single and dual forms.
6. High-k — a better gate insulator than glass — a high permittivity dielectric, usually hafnium based, in use since 2007.

3.10 Testing, dicing, packaging

3.10.1 PLAIN - in simple words

1. At the end of the fab, the wafer holds hundreds of finished chips, still joined in one disc. Some are broken.
2. So we test every one, while they are still attached.
3. A machine lowers a head with hundreds of fine needles onto each chip, sends in signals, and checks the answers.
4. Bad chips are marked for scrap. Good chips are sorted by how well they performed, which is called binning.
5. The same design, from the same wafer, can sell as an expensive fast part or a cheaper slow part.
6. Then the wafer is cut into individual chips. Each is glued into a package, connected electrically, and covered.
7. The package is not just protection. It carries power in, carries heat out, and fans tiny pads out to board-sized contacts.

3.10.2 PLAIN - a picture in your head

1. Think of biscuits baked as one sheet, which must be checked, graded, cut and boxed.
2. Testing is tasting each one before cutting. Cheaper than boxing a bad one and finding out later.
3. Binning is grading: perfect ones in the premium box, uneven ones in the value box, from the same recipe.
4. Modern high-end products do not use one big biscuit. They use several smaller ones side by side, connected together.
5. Where this comparison breaks: biscuits need not line up to a micrometer, and none needs 1,000 watts of heat carried off a postage stamp.

3.10.3 PLAIN - a worked example

1. Take a 300 millimetre wafer, about 70,700 square millimetres, and a die of 100 square millimetres.
2. A circle cannot be tiled perfectly by rectangles, so you get about 600 full dies, not 707.
3. Apply defects at 0.1 per square centimetre and about 90 percent are good, so about 540 good dies.
4. If the processed wafer cost 18,500 US dollars, that is about 34 US dollars per good die.

Die area	Dies per wafer	Yield	Cost per good die
50 sq mm	about 1,240	95 pct	about 16 USD
100 sq mm	about 600	90 pct	about 34 USD
200 sq mm	about 290	82 pct	about 78 USD
600 sq mm	about 88	55 pct	about 380 USD

5. Doubling die area more than doubles cost per working chip. You lose twice over: fewer dies and worse yield.
6. That table, and nothing else, explains why the industry moved to chiplets.

3.10.4 PLAIN - what is really happening inside

1. Wafer probe, or wafer sort, is the first electrical test. A probe card with hundreds of needles contacts the pads, often at several voltages and temperatures.
2. Dicing separates the dies. A diamond blade saw is traditional. Laser stealth dicing makes a weak plane inside the silicon, then the wafer is stretched and splits.
3. Die attach fixes the die down, face up on a pad or face down onto a substrate.
4. Wire bonding joins the die pads to the package with fine gold or copper wires, welded by heat and ultrasound. Cheap and mature.

5. Flip-chip turns the die upside down and connects through solder bumps across the whole face, allowing many more connections and shorter power paths.
6. Above the die goes a lid or heat spreader, joined by a thermal interface material: a paste, a pad, or soldered metal.
7. Underneath sits the substrate, a small multilayer board fanning fine connections out to larger solder balls or pins.
8. Chiplets change the picture. Several dies sit on a shared carrier and behave as one product.

3.10.5 TECHNICAL - the engineer's version

1. Yield is modelled from defect density. Two standard forms are used.

```
Poisson: Y = exp(-A * D0)
Murphy:  Y = ((1 - exp(-A * D0)) / (A * D0)) ^ 2

A  = die area in square centimetres
D0 = defect density per square centimetre
```

2. Murphy's model is usually closer to reality, because defects cluster rather than spreading evenly.
3. Mature high-volume nodes run D0 near 0.05 to 0.1 per square centimetre. A new node starts far worse and improves over quarters.
4. Reported N2 test-chip yields in early 2026 were about 70 to 80 percent for TSMC, with Intel and Samsung lower. These are press reports, not disclosure.
5. Binning sorts by maximum stable frequency, leakage, and functional core count. Fusing off a defective core is normal and deliberate.
6. The reticle limit sets the largest single die at about 26 by 33 millimetres, 858 square millimetres, at low numerical aperture EUV.
7. 2.5D packaging places multiple dies on a silicon interposer. TSMC's platform is CoWoS, introduced in 2012; Intel's bridge alternative is EMIB.
8. 3D packaging stacks dies with through-silicon vias. Examples are Intel Foveros, announced 2018, and TSMC SoIC.
9. High Bandwidth Memory is a JEDEC standard: stacked DRAM joined by through-silicon vias, placed beside the logic die. HBM4 was standardized in 2025.

Product	Year	Transistors
NVIDIA H100	2022	80 billion
NVIDIA B200	2024	208 billion, 2 dies
Cerebras WSE-3	2024	4 trillion
NVIDIA Rubin	announced 2026	336 billion, 4 tiles

10. The known good die problem is central. Stacking three good dies with one bad one wastes all four, so pre-assembly test coverage must be very high.
11. Advanced packaging capacity, not wafer capacity, has been the binding constraint on artificial intelligence accelerator supply since 2023.

3.10.6 WORDS - remember these

1. Wafer probe — testing chips before cutting — wafer sort using a probe card and automated test equipment.
2. Yield — the share of chips that work — good die over gross die, modelled from area and defect density.
3. Binning — sorting identical chips into product grades — speed, power and core-count sorting of one die design.

4. Flip-chip — die mounted face down on bumps — controlled collapse chip connection, giving area-array input and output.
5. Interposer — a carrier that wires dies together — a silicon or organic substrate with fine-pitch routing, the basis of 2.5D.
6. Chiplet — a small die that is part of a bigger product — a partitioned die assembled with others into one package.
7. HBM — stacked memory beside the processor — High Bandwidth Memory, a JEDEC standard using through-silicon-via DRAM stacks.

3.11 What "3 nm" actually means now

3.11.1 PLAIN - in simple words

1. Chip generations have names like 7 nanometre, 5 nanometre, 3 nanometre, 2 nanometre.
2. Most people assume the number is the size of something on the chip. It is not, and has not been for over a decade.
3. Today it is a generation label meaning "our next, better process", and each company picks its own numbers.
4. A silicon atom is about 0.2 nanometres across. Nothing useful can be three atoms wide and still work as a transistor.
5. On a real 3 nanometre process, the spacing between neighbouring gates is about 45 nanometres, fifteen times the name.
6. What did change, really and physically, is the shape of the transistor.
7. It went from flat, to a standing fin gripped on three sides, to stacked ribbons with the gate wrapped all the way around.
8. Those shape changes are real engineering. The numbers on the box are marketing.

3.11.2 PLAIN - a picture in your head

1. Think of a garden hose with a hand squeezing it. The hand is the gate; the water is the current.
2. Flat transistors are a hose on the ground with one hand pressing from the top. It works, but water leaks past.
3. A FinFET stands the hose on edge so the hand grips three sides. Much better control, much less leak.
4. Gate-all-around wraps the hand fully around the hose, with several thin hoses stacked above one another.
5. That is why the industry moved: at small sizes a flat gate simply stopped being able to turn the current off.
6. Where this comparison breaks: no water flows. Current is carried by electrons or holes, and "off" means suppressing quantum tunnelling and thermal leakage.

3.11.3 PLAIN - a worked example

1. Here is the growth in transistor count, with real parts and real years.

Chip	Year	Transistors
Intel 4004	1971	about 2,300
Intel 8086	1978	29,000
Intel 80386	1985	275,000
Intel Pentium	1993	3.1 million
Pentium 4	2000	42 million
Apple M4	2024	28 billion
NVIDIA B200	2024	208 billion

Chip	Year	Transistors
Cerebras WSE-3	2024	4 trillion

2. From 2,300 to 208 billion is a factor of about 90 million, in 53 years.
3. The honest version on the 4004: Intel has long quoted 2,300, and that is the number in most books. Careful die counts give about 2,250.
4. A second honesty note: Apple publishes no transistor count for every part. None was published for the A19 Pro of 2025, so the table uses the M4.
5. The Cerebras WSE-3 is a special case. It is not diced at all. It is one square cut from a wafer, about 46,225 square millimetres, with 900,000 cores.

3.11.4 PLAIN - what is really happening inside

1. Originally the node name did mean something. In the 1970s and 1980s it tracked the gate length.
2. Then it drifted to meaning half the spacing between the first metal wires.
3. Then, from roughly the 22 and 20 nanometre generations, it stopped tracking anything measurable.
4. Companies chose different numbering, so an Intel node and a TSMC node with the same name were not the same thing.
5. Intel renamed its own nodes in 2021 to line up with the competition, which is itself an admission that the numbers are labels.
6. Look instead at transistor density, in millions of transistors per square millimetre, plus the gate and metal pitches.
7. Even density is arguable, because it depends on which cell library you count and what mix of logic you assume.

3.11.5 TECHNICAL - the engineer's version

Node	Gate pitch	Density	Structure
TSMC N3	about 45 nm	about 197 MTr/sq mm	FinFET
Samsung 3GAE	about 40 nm	about 150 MTr/sq mm	Nanosheet GAA
TSMC N2	not disclosed	about 15 pct over N3E	Nanosheet GAA
Intel 18A	not disclosed	not disclosed	RibbonFET GAA

1. Planar bulk MOSFET dominated from the 1960s to about 2011.
2. The FinFET concept was demonstrated as DELTA by Hisamoto and colleagues at Hitachi in 1989.
3. It was then developed and named FinFET at the University of California, Berkeley around 1999, by a team including Chenming Hu, Tsu-Jae King Liu and Jeffrey Bokor.
4. Intel put it into volume production first, as Tri-Gate at 22 nanometres, announced 2011 and shipping 2012.
5. Gate-all-around nanosheet reached mass production first at Samsung, with 3GAE in 2022.
6. TSMC moved to nanosheet at N2, starting mass production in late 2025 at Fab 20 in Hsinchu and ramping through 2026.
7. Reported N2 wafer price is about 30,000 US dollars, against roughly 20,000 to 25,000 for 3 nanometre class.
8. Intel 18A, with RibbonFET gate-all-around plus PowerVia backside power, entered high-volume manufacturing in 2025 with Panther Lake.
9. Where experts disagree: how to compare nodes fairly. Density metrics weighted across cell types are contested, and there is no neutral referee.
10. Established fact: nanosheet is in production in 2026. Active research: complementary FET, stacking n-type and p-type vertically, and two-dimensional channel materials. Marketing claim: node names as measurements.

11. The honest version: if someone says a 3 nanometre chip has 3 nanometre features, they are repeating a press release. Ask for gate pitch and density.

3.11.6 WORDS - remember these

1. Node — a process generation — a named technology offering, no longer tied to any physical dimension.
2. Gate length — how far current travels under the gate — the physical channel length, now decoupled from the node name.
3. Contacted poly pitch — spacing between transistor gates — CPP, a real comparable dimension, about 45 nanometres at TSMC N3.
4. FinFET — a transistor standing on edge — a fin channel with the gate on three sides, in production from 2011.
5. Gate-all-around — the gate wrapped fully around — GAA nanosheet or ribbon channels, in production from 2022.

3.12 Who makes chips and why it matters

3.12.1 PLAIN - in simple words

1. Designing a chip and making a chip used to be the same business. Not any more.
2. Most chip companies do not own a factory. They design and pay someone else to build. That is the fabless model, and the builder is a foundry.
3. The largest foundry by far is TSMC, in Taiwan. It builds for companies that compete with each other and sells no branded chips.
4. Samsung, in South Korea, is both a foundry and a chip company of its own.
5. Intel, in the United States, historically built only its own chips, and is now trying to be a foundry for others too.
6. There is a second layer of licensing. ARM, in the United Kingdom, makes no chips. It designs instruction sets and cores, and licenses them.
7. RISC-V is a newer alternative, where the instruction set itself is open and free to use.
8. The important fact is concentration. Very few buildings on Earth can make the most advanced chips.

3.12.2 PLAIN - a picture in your head

1. Think of book publishing. An author writes, a printing plant prints, and a typeface company licenses the letterforms.
2. Fabless chip companies are the authors. Foundries are the printing plants, owning the expensive machines and printing for whoever pays.
3. ARM is the typeface company. Nobody buys a typeface alone, but almost every book uses one, and the fee is small per copy and huge in total.
4. RISC-V is a typeface released free for anyone to use and modify.
5. Where this comparison breaks: there are thousands of printing plants. There are fewer than five organizations that can print at the leading edge, and one supplier of the key machine.

3.12.3 PLAIN - a worked example

1. Trace an Apple phone chip from idea to shipment.
2. Apple designs it in California, using an ARM instruction set under an architecture licence, then sends layout files to TSMC.
3. TSMC makes masks, then runs wafers in Taiwan on ASML machines from the Netherlands, with optics from Germany.
4. The blank wafers most likely came from Japan, where Shin-Etsu and SUMCO dominate 300 millimetre supply.
5. Finished wafers are tested, diced and packaged, often in Taiwan, then assembled into phones elsewhere in Asia.

6. Not one country in that chain could complete it alone. That is the point.

3.12.4 PLAIN - what is really happening inside

1. The reason for the split is cost. A leading-edge fab now costs 25 to 35 billion US dollars and must be upgraded every few years.
2. Only a company running many customers' products can keep such a fab full enough to pay for itself.
3. So designers stopped buying fabs, and the surviving fab owners got larger and fewer.
4. This is not a conspiracy. It follows from rising fixed costs in a business that needs volume to cover them.
5. The same logic applies at ASML, where the research cost of EUV was so large that only one company saw it through.
6. Concentration then becomes a security problem, because one earthquake or export restriction can affect the whole world's supply.
7. Governments responded with subsidy programmes, including the United States CHIPS and Science Act of 2022 and the European Chips Act of 2023.

3.12.5 TECHNICAL - the engineer's version

Company	Role	Base
TSMC	Pure-play foundry	Taiwan
Samsung Foundry	Foundry plus own chips	South Korea
Intel Foundry	IDM plus foundry	United States
ASML	EUV and DUV scanners	Netherlands

1. TSMC was founded in 1987 by Morris Chang and created the pure-play foundry model. Before that, building for others was a side business.
2. Early fabless companies include Xilinx, founded 1984, Qualcomm, founded 1985, and NVIDIA, founded 1993.
3. ARM was founded in 1990 as Advanced RISC Machines, a joint venture of Acorn Computers, Apple and VLSI Technology.
4. ARM licences come in two tiers: core licences, where you use ARM's design, and architecture licences, where you build your own compatible core. Apple holds the latter.
5. RISC-V began at the University of California, Berkeley in 2010. RISC-V International moved its legal home to Switzerland in 2020.
6. Important distinction: RISC-V being open does not make any particular RISC-V chip open. The instruction set is open; implementations may be proprietary.
7. Market share: TSMC held roughly 67 to 71 percent of the global foundry market in early 2026, depending on the analyst. Its leading-edge share is higher.
8. In materials, Shin-Etsu Chemical and SUMCO, both Japanese, supply most of the world's 300 millimetre polished wafers.
9. TSMC's Arizona site began 4 nanometre class production in 2024. The company has announced about 265 billion US dollars of United States investment.
10. Established fact: leading-edge capacity is concentrated in Taiwan and South Korea. Active development: Arizona, Japan and Dresden fabs ramping. Marketing claim: that any subsidy programme has already ended that concentration.
11. To see part of this chain yourself: on Linux, `lscpu` reports vendor and microarchitecture; on macOS, `sysctl -n machdep.cpu.brand_string` names the part.

3.12.6 WORDS - remember these

1. Fabless — designs chips, owns no factory — a company outsourcing all wafer manufacturing to a foundry.

2. Foundry — a factory that builds other people's designs — a contract wafer manufacturer, pure-play if it sells no branded chips.
3. IDM — a company that designs and builds — integrated device manufacturer, the older model, as at Intel and Samsung.
4. Tape-out — sending the finished design to be made — release of final layout data to mask making.
5. RISC-V — an open, free instruction set — an open standard ISA governed by RISC-V International.

3.98 Common wrong ideas

1. Wrong: chips are made from ordinary beach sand. Right: they are made from selected lump quartz or quartzite above 99 percent silica, chosen for low iron content.
2. Wrong: silicon is mined as a metal. Right: silicon never occurs pure. It is always bound to oxygen and must be reduced out with carbon at about 1,900 degrees Celsius.
3. Wrong: 99 percent pure is good enough. Right: that is one foreign atom in a hundred. Chips need nine to eleven nines, roughly one per billion or better.
4. Wrong: a wafer is just a slice of purified silicon. Right: it must be one single crystal with no grain boundaries, grown from a seed, with specified orientation and resistivity.
5. Wrong: the mask is a stencil shaped like the circuit. Right: it is a 4x enlarged plate of computed shapes, distorted by optical proximity correction so the printed result is correct.
6. Wrong: EUV is just a brighter ultraviolet lamp. Right: it is 13.5 nanometre soft X-ray light made by vaporizing tin droplets, absorbed by air and glass, so it needs vacuum and mirrors.
7. Wrong: a 3 nanometre chip has 3 nanometre features. Right: the node name is a marketing label. Gate pitch on a 3 nanometre class process is around 45 nanometres.
8. Wrong: sand is cheap, so chips should be cheap. Right: the blank wafer is well under 1 percent of a processed leading-edge wafer. The value is in more than a thousand process steps.
9. Wrong: a faster and a slower processor from one family are different designs. Right: they are frequently the same die, binned by measured performance, sometimes with defective parts fused off.
10. Wrong: making the chip is hard and packaging is trivial. Right: since 2023, advanced packaging capacity rather than wafer capacity has limited artificial intelligence accelerators.

3.99 Chapter summary in 20 lines

1. Silicon is about 27.7 percent of the Earth's crust by mass, second only to oxygen, and never occurs pure.
2. It is locked in silica and silicate minerals, and the cleanest common source is the mineral quartz.
3. Chip feedstock is lump quartz or quartzite above 99 percent silica, not beach sand, for both purity and particle size.
4. A submerged-arc furnace at about 1,900 degrees Celsius reduces quartz with carbon, giving metallurgical-grade silicon at 98 to 99.5 percent.
5. That silicon becomes trichlorosilane, is distilled, and is deposited back onto hot filaments by the Siemens process at about 1,100 degrees Celsius.
6. The result is polysilicon at nine to eleven nines purity, roughly one foreign atom per billion or fewer.
7. Polysilicon is melted and regrown as one single crystal by the Czochralski method, using a seed, a thin neck, and slow rotating pulling.
8. A 300 millimetre boule is about 2 metres long and weighs about 265 kilograms, with a defined orientation and dopant level.
9. Float-zone growth gives purer, higher-resistivity silicon without a crucible, but is limited in diameter and used for power and detector devices.
10. The boule is cropped, ground, notched, wire sawn, lapped, etched and polished into 775 micrometer mirror wafers.
11. 300 millimetres is the industry limit. The 450 millimetre transition stalled when its consortium collapsed around 2016 and 2017.

12. Fabs run at ISO 14644-1 cleanliness with hundreds of air changes per hour, ultrapure water at 18.2 megaohm-centimetre, and vibration isolation.
13. A 2 nanometre class fab module costs roughly 25 to 35 billion US dollars, most of it tools rather than building.
14. Photolithography prints one 26 by 33 millimetre field at a time through a 4x reticle, then steps across the wafer and repeats.
15. Resolution follows the Rayleigh criterion, so shorter wavelength and higher numerical aperture give smaller features, at the cost of depth of focus.
16. Light went from 436 and 365 nanometre mercury lamps, to 248 and 193 nanometre lasers, to 193 nanometre immersion, to 13.5 nanometre EUV.
17. EUV comes from tin droplets struck twice by a carbon dioxide laser, 50,000 times a second, focused by molybdenum-silicon mirrors in vacuum, and only ASML builds the machine.
18. Doping, annealing, oxidation, CVD, PVD, ALD and 15 or more copper damascene metal layers add up to over 1,000 process steps and about three months in the fab.
19. Wafers are probe tested, binned, diced and packaged, and defect density punishes large dies so hard that the industry moved to chiplets, interposers and stacked memory.
20. Node names are marketing labels, transistors went planar to FinFET to gate-all-around, and the ability to build at the leading edge sits in very few hands.

4

CHAPTER

KB

The Transistor — The Switch That Built The World

4.0 What this chapter gives you

1. You will be able to say what a transistor is, in one sentence, without lying.
2. You will understand doping: how a few foreign atoms turn silicon from a poor conductor into a controllable one.
3. You will understand what a "hole" is, and why it behaves like a positive particle.
4. You will be able to explain a PN junction and a diode from first principles.
5. You will know the real history of the 1947 invention, including the argument about credit.
6. You will understand the MOSFET properly: gate, oxide, threshold, inversion, and its two operating regions.
7. You will understand CMOS, and be able to calculate the power a chip burns.
8. You will be able to state Moore's Law correctly and explain why it was always an economic claim, not a promise about speed.

4.1 Before the transistor: relays and glowing glass

4.1.1 PLAIN – in simple words

1. A computer needs one thing above all: a switch that another wire can flip.
2. The first such switch was the **relay**: an electromagnet that pulls a metal arm onto a contact.
3. The next switch was the **vacuum tube**: a glass bulb with the air pumped out and a hot wire inside.
4. Put a wire mesh in between, and a small voltage on that mesh throttles the flow. A switch with no moving parts.
5. Tubes are about a thousand times faster than relays. They are also hot, fragile, power-hungry, and they die.

4.1.2 PLAIN – a picture in your head

1. A relay is a drawbridge with a motor. A small signal starts the motor, the bridge lowers, and heavy traffic crosses.
2. A vacuum tube is a hose with a valve. The water is already under pressure; your hand on the valve barely works at all.
3. The tube's valve is a voltage on a mesh. Turning it costs almost nothing, but it controls a large current.
4. Where this comparison breaks: a drawbridge and a valve must physically move. In a tube only electrons move, which is why it is so much faster.
5. And no plumbing picture covers the heater. The tube burns power all the time just to keep electrons boiling off, whether or not it is doing anything.

4.1.3 PLAIN - a worked example

1. Konrad Zuse built the Z3 in Berlin from telephone relays and presented it to an audience of scientists on 12 May 1941.
2. It used about 2,600 relays, roughly 1,400 of them for memory, and ran at about 5 to 10 cycles per second.
3. ENIAC, built from tubes at the University of Pennsylvania, was formally dedicated on 15 February 1946.
4. Its machine cycle was 200 microseconds and it did about 5,000 additions per second.

Machine	Year	Switch	Additions per second
Zuse Z3	1941	Relay	About 1
ENIAC	1946	Vacuum tube	About 5,000

5. The Z3 itself was destroyed on 21 December 1943 in an Allied bombing raid on Berlin.

4.1.4 PLAIN - what is really happening inside

1. The **cathode** is metal heated by a filament to roughly 1,000 degrees Celsius. Hot metal throws electrons off its surface.
2. The **anode**, or plate, sits opposite at a positive voltage, often 100 to 300 volts, so electrons stream across the empty space. That is the current.
3. The **grid**, a fine mesh, sits in the path. A small negative voltage on it pushes electrons back, thinning the stream.
4. Because the grid is negative, almost no current flows into it. Large plate current, tiny grid current. That is amplification.
5. Now the costs. Every heater runs constantly, so ENIAC drew about 150 kilowatts, much of it just keeping filaments hot.
6. Filaments burn out, because that is physically what they are. And the machine cannot be used the instant you switch it on: cathodes need tens of seconds to reach temperature. That is warm-up time.

4.1.5 TECHNICAL - the engineer's version

1. John Ambrose Fleming patented the thermionic diode in 1904. Lee de Forest added the control grid in 1906, producing the triode he called the Audion.
2. ENIAC used ordinary octal-base radio tubes: 6SN7 flip-flops in the decimal accumulators, with 6L7, 6SJ7, 6SA7 and 6AC7 types in logic roles.

ENIAC figure	Value
Vacuum tubes	About 17,468
Crystal diodes	7,200
Resistors	70,000
Capacitors	10,000
Soldered joints	About 5,000,000
Power draw	150 kW
Weight	About 30 short tons

3. The tube count is quoted differently by different sources. 17,468 is the usual figure for the machine as built. Penn and several standard references round it to 18,000, and the count changed over its working life.
4. Early on, several tubes failed almost every day and ENIAC was unusable roughly half the time. High-reliability tubes arrived in 1948, after which failures dropped to about one every two days.

5. The longest failure-free run was 116 hours, close to five days, in 1954. ENIAC was switched off for the last time on 2 October 1955.
6. The honest version: failures were reduced partly by never switching the machine off. Thermal cycling, not steady running, kills filaments.
7. A 6SN7 heater draws 6.3 V at 0.6 A, about 3.8 W per tube. Multiplied by roughly 17,000 tubes that is about 65 kW of pure heater load, a large share of the 150 kW total.

4.1.6 WORDS - remember these

1. Relay — an electromagnet that flips a metal switch — electromechanical single-pole element with millisecond operate time.
2. Vacuum tube — a glass bulb with a hot wire controlling electron flow — thermionic valve using emission into an evacuated envelope.
3. Triode — a tube with cathode, mesh and plate — three-electrode thermionic device with a control grid.
4. Thermionic emission — hot metal throwing off electrons — thermally activated escape over the material's work function barrier.
5. Warm-up time — the wait before a tube machine works — the time for indirectly heated cathodes to reach emission temperature, 10 to 60 seconds.

4.2 Doping, properly

4.2.1 PLAIN - in simple words

1. Pure silicon is a poor conductor: not an insulator, not a metal, something annoying in between.
2. Each silicon atom has four outer electrons, and in a crystal every one is locked into a shared bond with a neighbour.
3. So we cheat. We add a very small number of foreign atoms. This is called **doping**.
4. Phosphorus and arsenic have five outer electrons. Four join the bonds; the fifth wanders off nearly free. Now there are spare electrons.
5. Silicon doped this way is **N-type**, N for negative, because the loose carriers are negative electrons.
6. Boron has three outer electrons. It fills three bonds and leaves the fourth with a gap in it.
7. That gap is a **hole**, and silicon doped this way is **P-type**.

4.2.2 PLAIN - a picture in your head

1. Picture a cinema where every seat is taken. Nobody can move. That is pure silicon.
2. Make one person stand in the aisle. That is N-type: a free electron that can walk anywhere.
3. Now instead take one person out of a seat in the middle, leaving it empty. That is P-type.
4. Nobody is standing, but the empty seat moves: the person to its left slides over, so the gap travels across the room while each person shifts one place. The empty seat is the hole.
5. It is not a thing. It is an absence. But film only the gap and you would swear a real object was moving, in the opposite direction to the people.
6. Also, holes are genuinely harder to push. In silicon a hole moves at roughly a third the speed of an electron under the same push. Hold on to that; it matters enormously later.

4.2.3 PLAIN - a worked example

1. Silicon has about 5 times 10 to the 22 atoms per cubic centimetre.
2. Now dope it. A typical light doping is 1 times 10 to the 16 dopant atoms per cubic centimetre.
3. Ratio: 1e16 divided by 5e22 is 2 times 10 to the minus 7, which is one dopant atom per five million silicon atoms.
4. But free carriers rise from 1e10 to 1e16. A million times more.

```
Pure silicon: 5e22 atoms, ~1e10 free electrons -> poor conductor
Add 1e16 P:   5e22 atoms, ~1e16 free electrons -> useful N-type
```

```
Dopant to silicon ratio : 1 in 5,000,000
Change in free carriers : about 1,000,000 times
```

5. One foreign atom in five million changes conductivity by a factor of about a million. The knob is absurdly sensitive.

4.2.4 PLAIN - what is really happening inside

1. A phosphorus atom takes a silicon lattice site. It is a similar size, so the crystal barely notices, and four of its five outer electrons pair with the four silicon neighbours.
2. The fifth is only weakly bound, needing about 0.045 electron-volts to break free. Room temperature heat energy is about 0.026 electron-volts, and given how many attempts happen per second that frees essentially all of them.
3. The freed electron leaves behind a phosphorus atom with one more proton than electron: a fixed positive ion.
4. Boron is the mirror image. It fills three bonds and leaves one incomplete. An electron from a nearby bond hops in to complete it, so the gap moves, and boron becomes a fixed negative ion.
5. So the picture is mobile carriers wandering, and charged ions frozen in place. Section 4.3 depends entirely on that distinction.
6. The honest version: a hole is not a particle. It is the collective behaviour of a nearly full band of electrons, which physics lets us describe exactly as one positive particle with its own effective mass.

4.2.5 TECHNICAL - the engineer's version

1. Silicon is a group 14 element with an indirect bandgap of 1.12 eV at 300 K and an atomic density of 5.0 times 10 to the 22 per cubic centimetre.
2. Intrinsic carrier concentration ni at 300 K is about 1.0 times 10 to the 10 per cubic centimetre. Older texts quote 1.45e10; the lower value is the modern accepted figure.
3. The mass action law holds in equilibrium: n times p equals ni squared. So N-type doping of 1e16 gives n about 1e16 and p about 1e4 per cubic centimetre.
4. Mobility at 300 K in lightly doped silicon: electrons about 1,400 cm squared per volt-second, holes about 450. A ratio near 3 to 1.

Region	Doping (per cm3)	Role
Lightly doped body	1e15 to 1e16	Substrate
Channel or well	1e17 to 1e18	Sets threshold
Source and drain	1e20 to 1e21	Low resistance

5. Measurement tools: a four-point probe gives sheet resistance in ohms per square; secondary ion mass spectrometry (SIMS) gives the dopant depth profile.
6. Note the convention: N-type and P-type name the majority carrier, not a net charge. Both materials are electrically neutral in bulk.

4.2.6 WORDS - remember these

1. Doping — adding a tiny amount of another element — controlled introduction of substitutional impurities to set carrier concentration.
2. N-type — silicon with spare electrons — donor-doped semiconductor with electrons as majority carrier.
3. P-type — silicon with gaps in its bonds — acceptor-doped semiconductor with holes as majority carrier.
4. Hole — a missing electron that acts positive — an empty valence band state, treated as a quasiparticle of charge plus q.
5. Intrinsic carrier concentration — how conductive pure silicon is — ni, about 1e10 per cubic centimetre at 300 K.

4.3 The PN junction and the diode

4.3.1 PLAIN - in simple words

1. Take N-type silicon, with spare electrons, and P-type silicon, with holes, and join them as one crystal with the doping changing partway through.
2. Electrons on the N side are crowded and the P side has almost none, so they spill across. Holes spill the other way.
3. That leaves a thin strip near the boundary with no free carriers at all, called the **depletion region**.
4. But the fixed ions remain, positive on the N side and negative on the P side, so a voltage now exists across that strip. Nobody applied it. It built itself, and it is a barrier to further spilling.
5. Push current one way and the barrier shrinks, so current flows easily. Push the other way and the barrier grows, so almost nothing flows.
6. A device that passes current one way and blocks the other is a **diode**.

4.3.2 PLAIN - a picture in your head

1. Picture two adjoining rooms at a party with an open doorway. The left room is packed, the right nearly empty.
2. People spill through the doorway, because crowds spread out. That is diffusion. But every person who crosses must hand over their coat at the door, and the coats pile up until crossing stops.
3. The coat pile is the depletion region. It builds itself out of what already crossed, and it is what stops any more crossing.
4. A doorman who pushes people toward the doorway compresses the pile and crossing resumes. That is forward bias.
5. Where this comparison breaks: coats are a passive obstruction, but the depletion region is an electric field, acting instantly and at a distance.

4.3.3 PLAIN - a worked example

1. Take a 1N4148 small-signal silicon diode. Connect its P side (the anode) to plus and its N side (the cathode) to minus. That is forward bias.

Forward voltage	Current	Change
0.60 V	1 mA	baseline
0.66 V	10 mA	10 times
0.72 V	100 mA	100 times
0.78 V	1 A	1000 times

2. Look at the pattern. Every extra 60 millivolts multiplies the current by ten. That is not a straight line, it is exponential.
3. Now reverse it. At minus 5 volts you might measure 10 nanoamps. At minus 50 volts, still nanoamps.
4. Forward current at 0.7 V is about 10 milliamps. Reverse current is about 10 nanoamps. A ratio of a million to one.

4.3.4 PLAIN - what is really happening inside

1. At the moment of joining, electrons diffuse into the P side and holes diffuse into the N side.
2. Every electron that leaves the N side leaves a fixed positive donor ion behind, and every hole that leaves the P side leaves a fixed negative acceptor ion. Neither ion can follow.
3. So a wall of positive fixed charge builds on the N side of the boundary and a wall of negative fixed charge on the P side.
4. Separated charge makes an electric field that pushes electrons back toward the N side, exactly opposing the diffusion that created it. Balance is reached when the two cancel, and nothing net flows.

5. The voltage across the region is the **built-in potential**, typically 0.6 to 0.8 volts in silicon. You cannot measure it with a voltmeter: the two metal-to-silicon contacts produce equal and opposite offsets.
6. Apply an external voltage with the P side positive and the depletion region narrows, the barrier drops, and carriers flood across.
7. Apply the opposite and the region widens, the barrier grows, and majority carriers cannot get over it at all.
8. The honest version: current is not simply blocked one way. Diffusion and drift currents always flow in both directions. Diode current is the small difference between two large, nearly equal opposing flows.

4.3.5 TECHNICAL - the engineer's version

1. The Shockley diode equation: I = Is times (exp(V divided by (n times VT)) minus 1).
2. VT is the thermal voltage, kT divided by q, equal to 25.85 mV at 300 K. Is is the reverse saturation current, typically 1e-12 to 1e-15 A for small silicon diodes, roughly doubling every 10 K. n is the ideality factor, between 1 and 2.
3. The decade-per-60-mV rule follows directly: 1 times 25.85 mV times 2.303 equals 59.5 mV.
4. Built-in potential is Vbi = VT times ln(Na times Nd divided by ni squared). With Na and Nd both 1e17 and ni equal to 1e10, Vbi = 0.02585 times 32.2, which is 0.83 V.

Diode type	Forward drop	Typical use
Silicon PN	0.6 to 0.7 V	General rectifier
Schottky	0.2 to 0.45 V	Fast switching
Germanium PN	0.25 to 0.3 V	Legacy detectors
Red LED	About 1.8 V	Indicator
Blue LED	2.8 to 3.4 V	Lighting, display

5. Reverse breakdown has two mechanisms: Zener tunnelling below about 5 V with a negative temperature coefficient, and avalanche multiplication above about 6 V with a positive one. Near 5.6 V they cancel, which is why 5.6 V Zener diodes were once used as temperature-stable references.
6. A Schottky diode is a metal-semiconductor junction with no stored minority charge, so it has essentially no reverse recovery. That is why it dominates high-frequency switching supplies.
7. To observe the curve: a curve tracer or a source measure unit sweep. In simulation, a SPICE DC sweep with the .DC directive plots it from the model parameters IS, N, RS and BV.

4.3.6 WORDS - remember these

1. PN junction — where N-type meets P-type — a metallurgical junction with a self-forming space-charge region.
2. Depletion region — the empty strip at the boundary — space-charge region containing only fixed ionized dopants.
3. Built-in potential — the voltage that appears by itself — Vbi, set by doping levels and ni, about 0.7 V in silicon.
4. Forward bias — pushing the easy way — P side positive, barrier lowered, exponential injection current.
5. Reverse bias — pushing the blocked way — N side positive, barrier raised, only saturation current flows.

4.4 The first transistor, and the argument about it

4.4.1 PLAIN - in simple words

1. Bell Telephone Laboratories had a business problem in the 1940s. Long distance calls needed amplifiers, amplifiers meant vacuum tubes, and tubes kept dying.

2. On 16 December 1947, two of their people made one work.
3. Their device was a small slab of very pure germanium with two gold contacts pressed onto its surface, extremely close together.
4. A signal on one contact controlled a larger current through the other, amplifying by up to about a hundred times.
5. On 23 December 1947 they demonstrated it to management. Their group leader, William Shockley, called it a magnificent Christmas present.
6. Bell Labs kept it quiet for six months, then announced it at a New York press conference on 30 June 1948.

4.4.2 PLAIN – a picture in your head

1. Imagine controlling a river using only the shape of the riverbed just under the surface, without touching the water.
2. That was the field-effect idea, tried since the 1920s. It kept failing and nobody knew why.
3. Bardeen worked out why in 1947. The semiconductor surface was covered in trapped electrons acting like a shield.
4. So Bardeen and Brattain stopped pushing from outside and went in through the surface directly, with two metal points almost touching.
5. Where this comparison breaks: the point-contact device is not really a field-effect transistor at all, and exactly how it worked was argued about for years afterwards.

4.4.3 PLAIN – a worked example

1. Brattain wrapped gold foil around the point of a small plastic wedge.
2. He cut the foil at the tip with a razor blade, leaving two gold edges about 50 micrometres apart. That is roughly half the width of a human hair.
3. He pressed the wedge onto a slab of high-purity germanium with a spring. One gold edge was the emitter, the other the collector, and the germanium slab itself was the base.
4. They measured a power gain. That is the test that matters; voltage gain alone can be had from a transformer.

Date	Event
16 Dec 1947	First working device
23 Dec 1947	Demonstrated to management
26 Feb 1948	Patent priority date
30 Jun 1948	Public announcement
Jan 1948	Junction transistor conceived
1951	Junction transistor working
1956	Nobel Prize in Physics

4.4.4 PLAIN – what is really happening inside

1. Shockley was the group leader who had pushed the field-effect programme for years, and it had failed repeatedly. The device that finally worked was built by two of his subordinates, using Bardeen's surface-state insight, in a direction Shockley had not chosen.
2. Then Bell Labs lawyers examined the patent and found Shockley's own field-effect writing was uncomfortably close to patents Julius Edgar Lilienfeld had filed in the 1920s.
3. To keep the patent safe they left Shockley's name off it. The application "Three-electrode circuit element utilizing semiconductive materials", with a priority date of 26 February 1948, named Bardeen and Brattain.
4. Shockley was furious. Over the New Year he worked alone, largely in a Chicago hotel room, and in January 1948 conceived a better device.

5. Instead of two metal points scratching a surface, he proposed a sandwich of three doped layers grown into one crystal: the **junction transistor**. His own application carries a priority date of 26 June 1948.
6. It was more robust and manufacturable, and it made the point-contact device obsolete. Gordon Teal and Morgan Sparks at Bell Labs produced working grown-junction transistors in 1951. Bardeen left for the University of Illinois that year.
7. In 1956 all three shared the Nobel Prize in Physics, "for their researches on semiconductors and their discovery of the transistor effect."
8. Where experts disagree: some historians treat Shockley as the essential figure, since the junction transistor is the ancestor of everything since. Others treat him as a manager who claimed a result he did not produce. The safest reading is that invention and improvement both mattered, and came from different people.

4.4.5 TECHNICAL - the engineer's version

1. The 1947 device was a point-contact transistor on N-type germanium with two gold point contacts about 50 micrometres apart, amplifying the input by up to about 100 times.
2. Germanium, not silicon. Germanium has a 0.66 eV bandgap and much higher carrier mobility, and in 1947 it could be purified far better than silicon.
3. Gordon Teal at Texas Instruments announced the first commercial silicon transistor in 1954. Silicon's wider 1.12 eV bandgap gives far better high-temperature behaviour.
4. The Regency TR-1, announced on 18 October 1954, was the first commercial transistor radio. It used four TI germanium transistors and sold for $49.95.
5. Shockley left Bell Labs in 1955 and founded Shockley Semiconductor Laboratory in Mountain View, California, in 1956.
6. In 1957 eight of his staff resigned over his management and founded Fairchild Semiconductor. Fairchild's descendants include Intel and AMD. That is the direct reason Silicon Valley is where it is.
7. John Bardeen won a second Nobel Prize in Physics in 1972, with Leon Cooper and John Robert Schrieffer, for the BCS theory of superconductivity. He remains the only person to have won the physics prize twice.

4.4.6 WORDS - remember these

1. Point-contact transistor — two metal points on a germanium slab — the 1947 device, with emitter and collector point contacts on a base.
2. Surface states — trapped charge that shields the inside — states at a semiconductor surface that pin the Fermi level and screen applied fields.
3. Junction transistor — a sandwich of three doped layers — Shockley's 1948 bipolar structure, ancestor of the BJT.
4. Power gain — output power larger than input power — the test that separates an amplifier from a transformer.
5. Germanium — the first material that worked — group 14 semiconductor, 0.66 eV bandgap, superseded by silicon for thermal reasons.

4.5 The bipolar junction transistor

4.5.1 PLAIN - in simple words

1. Take three doped layers in a row: N, then P, then N. That is an **NPN** transistor. Reverse every layer and you have a **PNP**.
2. The base is deliberately made very thin and only lightly doped. That is the whole secret.
3. Feed a small current into the base and a much larger current flows from collector to emitter, typically 100 to 300 times larger.
4. There is a catch, and it is the catch that killed this device for computing.

5. Base current means power burned. Millions of these would burn power constantly, even doing nothing.

4.5.2 PLAIN – a picture in your head

1. Picture a wide fast river with a narrow island in the middle.
2. Water hitting the island drains away down a small side channel. That side channel is the base current.
3. Because the island is so narrow, almost all the water sweeps past it and reaches the far bank. Maybe two hundred litres arrive for every one that drains.
4. But the side channel must keep draining. Block it and the whole flow stops.
5. And the small side flow is never optional. It is a permanent cost, and section 4.7 is about a device that removed it.

4.5.3 PLAIN – a worked example

1. Take a 2N3904, one of the most common small NPN transistors ever made. Its current gain, beta, is roughly 100 to 300 at 10 milliamps.
2. Say we want to switch a 100 milliamp load. Use the worst-case beta of 100, never the typical value, so base current needed = 100 mA divided by 100 = 1 mA.
3. In practice you overdrive by 5 times to force it hard on, so use 5 mA.
4. With a 5 V control signal, the base-emitter junction drops about 0.7 V, so the resistor sees 4.3 V, and R = 4.3 V divided by 0.005 A = 860 ohms. Use the standard value 820 ohms.

```
5V ---[820 ohm]--- base
                    |
100 mA load ---> collector
                    |
                emitter --- ground

Base power  : 5 mA x 0.7 V   = 3.5 mW
Switch loss : 100 mA x 0.2 V = 20 mW
Total burnt : about 23.5 mW, continuously
```

5. Holding this one transistor on costs about 23.5 milliwatts forever, doing nothing.
6. A billion of them would burn 23.5 megawatts. That is a power station, and that number is why bipolar logic was never going to build a modern processor.

4.5.4 PLAIN – what is really happening inside

1. In an NPN the emitter is heavily doped, the base thin and lightly doped, and the collector moderately doped and physically larger.
2. Forward bias lowers the emitter-base barrier, so electrons flood from the N emitter into the P base, where they are minority carriers and ought to recombine.
3. But the base is thinner than a micrometre, so most cross it before meeting a hole, and at the far edge the reverse-biased base-collector field sweeps them straight into the collector.
4. So perhaps 99.5 percent arrive at the collector and 0.5 percent recombine in the base and must be replaced by base current.
5. Beta is just the ratio of those two numbers. Lose 0.5 percent and beta is about 200. Gain comes from geometry and doping, not magic.
6. **Cut-off** is when base-emitter voltage is below about 0.6 V. Nothing is injected and the device is off.
7. The honest version: calling the BJT current-controlled is a convention, not a law. The physics is exponentially controlled by base-emitter voltage. But that exponential is so steep and so temperature-sensitive that designing with base current is far more reliable, so everyone does.

4.5.5 TECHNICAL - the engineer's version

1. The Ebers-Moll model describes the BJT. In forward active mode IC = Is times exp(VBE divided by VT).
2. Common-emitter gain beta equals IC over IB. Common-base gain alpha equals IC over IE. They relate as beta = alpha divided by (1 minus alpha).

Parameter	2N3904 typical	Meaning
hFE (beta)	100 to 300	Current gain
VCE(sat)	0.2 V	On-state drop
VBE(on)	0.65 to 0.75 V	Turn-on voltage
VCEO max	40 V	Breakdown limit
fT	About 300 MHz	Gain bandwidth

3. Bipolar logic families ran RTL, then DTL, then TTL. The 7400 series launched by Texas Instruments in 1964 and dominated digital design for twenty years.
4. Standard TTL dissipated roughly 10 mW per gate at rest; the 74LS low-power Schottky family reduced this to about 2 mW. Emitter-coupled logic was faster still at 25 mW or more per gate, and Cray supercomputers cooled it with liquid.
5. Bipolar devices still dominate analogue and radio work: low noise, high transconductance per unit current, excellent matching. They lost digital, not everything. Silicon-germanium heterojunction bipolar transistors reach fT above 300 GHz.
6. SPICE models BJTs with the Gummel-Poon model, which extends Ebers-Moll to cover high injection and base-width modulation.

4.5.6 WORDS - remember these

1. BJT — a sandwich transistor controlled by current — bipolar junction transistor, using both electrons and holes as carriers.
2. Emitter — the layer that supplies carriers — heavily doped terminal that injects minority carriers into the base.
3. Base — the thin middle layer that controls things — lightly doped region, thinner than the minority carrier diffusion length.
4. Collector — the layer that catches carriers — reverse-biased terminal that sweeps arriving minority carriers out.
5. Beta or hFE — how many times the current is multiplied — common-emitter DC current gain, IC over IB, typically 20 to 500.

4.6 The MOSFET, the transistor inside your computer

4.6.1 PLAIN - in simple words

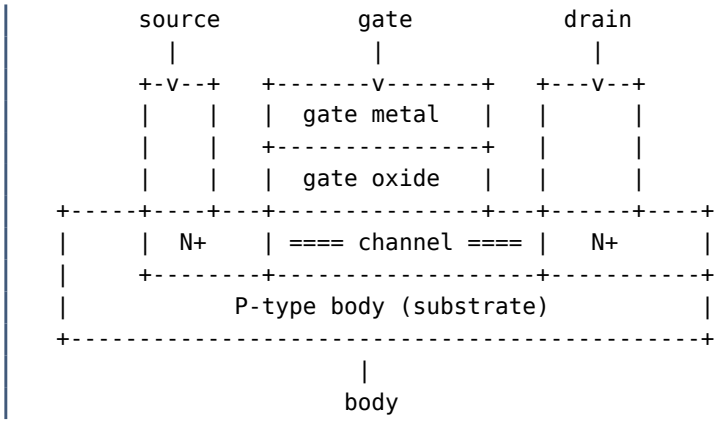
1. Almost every transistor in the device you are reading this on is a MOSFET.
2. It has three parts you must know. The **source**, where carriers come from. The **drain**, where they go. The **gate**, which decides whether they go.
3. Because the gate is insulated, no current flows into it. It is a capacitor plate, not a wire into the device.
4. Put a positive voltage on the gate of an N-channel MOSFET and its electric field reaches through the insulator into the silicon below.
5. That field drags electrons up to the surface, forming a thin conducting layer bridging source to drain. That layer is the **channel**.
6. Here is the point that changes everything. Because the gate is insulated, holding the switch on costs no continuing current. You charge the gate once and it stays.

4.6.2 PLAIN - a picture in your head

1. Imagine a dry canal between two full reservoirs. Nothing can cross.
2. Lower the plate and it does not push water. Instead it pulls groundwater up from below, filling the canal bed.
3. That is the gate, and that is why MOSFET logic can sit still without burning power.
4. Where this comparison breaks: lowering a real plate costs energy each time because of gravity, and so does switching a real gate, because its capacitance must be charged and discharged. Holding is free; changing is not.
5. Also, the insulator is not perfect. At modern thicknesses a few electrons tunnel straight through. The plate leaks, slightly.

4.6.3 PLAIN - a worked example

1. Here is a planar N-channel MOSFET, sliced through the middle.



2. The channel is the dashed strip. It does not exist until the gate says so.
3. Now real numbers. Threshold voltage, the gate voltage at which the channel appears: 0.4 V. Supply: 1.0 V. Channel length L: 20 nanometres. Channel width W: 100 nanometres.
4. Below 0.4 V on the gate the device is off, passing only leakage.
5. At 1.0 V on the gate the overdrive is 1.0 minus 0.4, which is 0.6 V, and the device is fully on.
6. Double the width to 200 nanometres and on-current doubles, because a wider channel is a wider pipe.
7. Halve the length to 10 nanometres and on-current doubles too, because a shorter pipe has less resistance.
8. That is why engineers speak of the W over L ratio. It is the shape knob for strength.

4.6.4 PLAIN - what is really happening inside

1. Start with the gate at zero volts, on an N-channel device built in P-type silicon. The body is full of holes and the two N+ islands are full of electrons.
2. Source to body is one PN junction and body to drain is another, facing the other way. Two back-to-back diodes cannot conduct either direction, so the device is firmly off.
3. Raise the gate a little. The field pushes holes downward, away from the surface, which is now depleted: no holes, no electrons. Still off, but the barrier is thinning.
4. Raise the gate more and the field starts pulling electrons up to the surface. Electrons in P-type silicon are minority carriers and very rare, but the field only needs a thin layer.
5. At some gate voltage the electron count at the surface equals the hole count in the bulk. That point is the **threshold voltage**.
6. Push past it and electrons outnumber holes at the surface. The surface has stopped behaving as P-type and started behaving as N-type.
7. That flip is called **inversion**, and it is the heart of the device. The material did not change. Its carrier population did.

8. Now source, channel and drain are all N-type. One continuous path. Current flows, and what happens next depends on the drain voltage.
9. If drain voltage is small, the channel is roughly even from end to end and the device behaves as a resistor whose value the gate sets. That is the **linear** or triode region.
10. Raise drain voltage and the channel is squeezed at the drain end, because the gate-to-channel voltage there is smaller. When its thickness there reaches zero, the device has reached **pinch-off**.
11. Past that, raising drain voltage barely raises current. The device behaves as a current source set by the gate. That is the **saturation** region.
12. Careful, this word is a trap. In a BJT, saturation means fully on with the lowest voltage drop. In a MOSFET it means pinched off at constant current. Same word, opposite feeling.
13. A **PMOS** device is the mirror image: P+ source and drain in an N-type well, turned on by a negative gate voltage relative to the source, which inverts the surface to P-type.
14. The honest version: "the channel disappears at pinch-off" is not quite true. A thin high-field region forms at the drain end and carriers are swept across it. Current does not stop, it stops increasing.

4.6.5 TECHNICAL - the engineer's version

1. Julius Edgar Lilienfeld filed the first field-effect patents in the mid 1920s and Oskar Heil filed a British patent in 1934. Neither produced a working device.
2. Shockley's 1945 field-effect attempts failed. Bardeen's 1947 surface state theory explained why: interface traps pinned the Fermi level and screened the applied field.
3. The solution was a clean thermally grown silicon dioxide interface. Mohamed Atalla and Dawon Kahng at Bell Labs built the first working MOSFET in 1959 and reported it in 1960.
4. Linear region, long-channel model: ID = mu times Cox times (W over L) times ((VGS minus Vth) times VDS minus VDS squared over 2), for VDS below VGS minus Vth.
5. Saturation region, long-channel model: ID = one half times mu times Cox times (W over L) times (VGS minus Vth) squared, for VDS at or above VGS minus Vth. Cox is gate oxide capacitance per unit area, the oxide permittivity divided by oxide thickness tox.
6. The square law is a long-channel result. Modern short-channel devices are velocity saturated, so drain current is close to linear in (VGS minus Vth), not quadratic. Textbooks teaching only the square law are teaching a device that stopped being manufactured decades ago.
7. Below threshold the device is not off. Subthreshold current falls exponentially, at a rate called **subthreshold swing** S, in millivolts per decade.
8. S has a hard floor of ln(10) times kT over q, which is 59.6 mV per decade at 300 K. Real planar devices reach 70 to 100; FinFETs reach about 65 to 70.
9. That floor is thermodynamic, not an engineering limit. It is the single biggest reason supply voltage stopped scaling, and therefore the reason for the power wall in section 4.10.
10. Intel reported reaching 1.2 nm of silicon dioxide, about five atomic layers, on its 65 nm process. Below that, direct tunnelling leakage became intolerable.

Quantity	Typical modern value
Threshold voltage	0.2 to 0.5 V
Supply voltage	0.65 to 1.1 V
Gate oxide EOT	0.8 to 1.0 nm
Subthreshold swing	65 to 80 mV/decade
Gate leakage	Picoamps per device

4.6.6 WORDS - remember these

1. MOSFET — a switch controlled by a voltage on an insulated plate — metal oxide semiconductor field-effect transistor, a unipolar device.
2. Source and drain — where carriers come from and go to — heavily doped regions of opposite type to the body.
3. Gate — the insulated control plate — electrode capacitively coupled to the channel, drawing only tunnelling leakage.
4. Gate oxide — the thin insulator under the gate — dielectric layer, silicon dioxide historically, hafnium-based high-k since 2007.
5. Threshold voltage — the gate voltage where it switches on — Vth, the point of strong inversion at the semiconductor surface.
6. Channel — the conducting bridge the gate creates — inversion layer of minority carriers at the oxide interface.
7. NMOS and PMOS — the electron version and the hole version — N-channel and P-channel devices, on with positive and negative gate overdrive.

4.7 CMOS: the pairing that made low power possible

4.7.1 PLAIN - in simple words

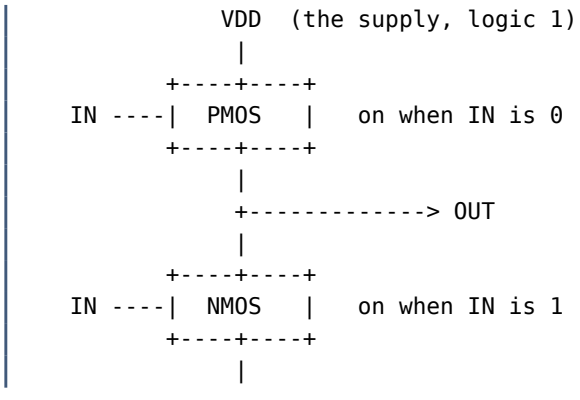
1. One MOSFET alone still wastes power. Pull a signal low through a resistor and current keeps flowing through that resistor.
2. Put a PMOS on top connected to the supply, and an NMOS below connected to ground. Join their gates and join their drains.
3. Feed in a 0. The PMOS turns on, the NMOS turns off, and the output is pulled up to the supply, a 1.
4. In both steady states exactly one transistor is on, so there is never a path from supply to ground. No path means no current.
5. **CMOS** stands for Complementary Metal Oxide Semiconductor. Complementary means the two types are used as a matched opposing pair.

4.7.2 PLAIN - a picture in your head

1. Think of a sink with a tap above and a plug below, joined by one lever.
2. Push the lever one way and the tap opens while the plug closes, so the basin fills.
3. The mechanism guarantees tap and plug are never open together, so water never runs straight down the drain.
4. Where this comparison breaks: during the flick of the lever both really are slightly open for an instant. That overlap is called crowbar or short-circuit current, and it is a real cost in fast circuits.

4.7.3 PLAIN - a worked example

1. Here is the CMOS inverter as a circuit.



```
              GND  (ground, logic 0)

   IN = 0 -> PMOS on, NMOS off -> OUT pulled to VDD = 1
   IN = 1 -> PMOS off, NMOS on -> OUT pulled to GND = 0
```

2. Now calculate the power a real chip burns. The formula is P = alpha times C times V squared times f.
3. C is the capacitance being charged, V the supply voltage, f the clock frequency, and alpha the activity factor, the fraction of nodes that change on a given tick.
4. Take a plausible processor core cluster: one billion switching nodes, 0.5 femtofarads each, 1.0 V supply, 3 GHz clock, activity factor 0.05.
5. Capacitance switched per cycle = 0.05 times 1e9 times 0.5e-15 farads = 2.5 times 10 to the minus 8 farads, that is 25 nanofarads.
6. Energy per cycle = C times V squared = 25 nF times 1.0 squared = 25 nanojoules.
7. Power = 25 nanojoules times 3 billion cycles per second = 75 watts.
8. Now change one number. Raise the supply to 1.2 V and power scales with V squared: 75 times 1.44 = 108 watts, for zero extra speed.

4.7.4 PLAIN - what is really happening inside

1. When the input flips from 1 to 0, the PMOS turns on and connects the output wire to the supply.
2. That output wire, plus the gates of everything it drives, forms a capacitance that must be charged from 0 volts up to the supply voltage.
3. Charging a capacitor C to voltage V stores one half C V squared and burns the same amount as heat in the transistor doing the charging. On the way back, the NMOS dumps the stored half to ground and that is lost too.
4. So a full 0 to 1 to 0 cycle costs C times V squared in total. That is where the formula comes from. Nothing at all is spent while the output sits still, which is the property bipolar logic could never offer.
5. Two other losses exist. First, crowbar loss: for the moment when the input is halfway, both transistors conduct and a spike passes through.
6. Second, and worse, leakage. Every off transistor passes a small current, and every gate leaks a tunnelling current through its oxide.
7. In the 1990s leakage was negligible. By the 90 nanometre generation, around 2003 to 2005, it had become a large share of total chip power.

4.7.5 TECHNICAL - the engineer's version

1. CMOS was invented by Frank Wanlass and Chih-Tang Sah at Fairchild in 1963. Their ISSCC paper described it as "nanowatt logic".
2. US patent 3,356,858 was filed 18 June 1963 and issued 5 December 1967. RCA shipped the CD4000 logic family in 1968.
3. Dynamic power: P_dyn = alpha times C_L times VDD squared times f. Static power: P_static = VDD times I_leak, summed over every device. Short-circuit power is typically 5 to 15 percent of dynamic power.
4. Leakage components are subthreshold conduction, gate oxide tunnelling, junction band-to-band tunnelling, and gate-induced drain leakage (GIDL).
5. Subthreshold leakage rises exponentially as threshold voltage falls, at roughly one decade per S millivolts, with S about 70 mV per decade.

Era	Node	Supply	Leakage share
1990	800 nm	5.0 V	Under 1 percent
2000	180 nm	1.8 V	A few percent
2004	90 nm	1.2 V	20 to 40 percent

Era	Node	Supply	Leakage share
2010	32 nm HKMG	1.0 V	Reduced again
2024	3 nm GAA	0.7 V	Managed by design

6. Mitigations in current use: high-k metal gate from Intel's 45 nm in 2007, multi-threshold cell libraries, power gating with sleep transistors, clock gating, dynamic voltage and frequency scaling, and FinFET or nanosheet geometry for better electrostatic control.
7. Why clocks stopped climbing: Intel's Pentium 4 Prescott reached 3.8 GHz in November 2004 and stopped there.
8. In May 2004 Intel cancelled the Tejas and Jayhawk projects, which had aimed at much higher clock rates, and redirected to dual-core designs. The first dual-core desktop part, the Pentium D, shipped in May 2005.
9. The reason is arithmetic. Raising frequency requires raising voltage to keep timing closed, and power then rises with roughly the cube of frequency.
10. Two cores at 3 GHz do more total work than one core at 4.5 GHz for less power, provided the software can use them. That proviso is the entire subject of parallel programming.
11. Top desktop parts in 2025 boost to roughly 5.7 to 6.0 GHz, about 1.5 times the 2004 peak across twenty years. In the twenty years before 2004, clock rates rose more than a thousand times.

```
# read package energy counter, microjoules, Intel RAPL
cat /sys/class/powercap/intel-rapl:0/energy_uj

# per-core frequency, temperature and package watts
sudo turbostat --interval 1
```

4.7.6 WORDS - remember these

1. CMOS — using an N and a P transistor as an opposing pair — complementary MOS logic with no static path from supply to ground.
2. Inverter — the gate that flips 0 to 1 — series PMOS pull-up and NMOS pull-down sharing gate and drain nodes.
3. Dynamic power — the cost of changing — alpha C V squared f, paid per transition.
4. Static power — the cost of merely existing — VDD times total leakage current, paid continuously.
5. Activity factor — how often a wire actually changes — alpha, typically 0.01 to 0.2 in real designs.

4.8 The same device, two jobs

4.8.1 PLAIN - in simple words

1. A transistor does not know whether it is in a computer or a radio. It is the same device doing the same thing.
2. Digital circuits use only the two extremes: fully off and fully on.
3. In between is a region where a small input change makes a large output change. Analogue circuits live there.
4. Digital design crosses that middle region as fast as possible, because it wastes power and its output is undefined.
5. Analogue design treats the same region as the whole point, because that is where gain comes from.

4.8.2 PLAIN - a picture in your head

1. Think of a light switch versus a dimmer knob.
2. A light switch has two positions and you flip past the middle quickly. That is digital.
3. A dimmer sits deliberately in the middle, where a small turn makes a noticeable change. That is analogue.

4. Where this comparison breaks: a mechanical dimmer wastes heat in proportion to how much it dims, and so does a transistor in its middle region, because it is dropping voltage while passing current.
5. That is why a class A audio amplifier is warm with no music playing, and why digital chips never sit in the middle if they can help it.

4.8.3 PLAIN – a worked example

1. The **transfer curve** plots output voltage against input voltage for one CMOS inverter. Read it left to right.

Input	Output	Region
0.0 V	1.0 V	Flat, logic 1 out
0.40 V	0.95 V	Starting to move
0.50 V	0.50 V	The cliff, high gain
0.60 V	0.05 V	Nearly done
1.0 V	0.0 V	Flat, logic 0 out

2. At 0 V in, the NMOS is off and the PMOS fully on, so the output sits at the full supply and the curve is flat. At about 0.5 V, the switching point, it plunges almost vertically to near 0 V, then flattens again.
3. So the curve is flat, then a cliff, then flat again.
4. Digital design wants the two flat ends. Flat means a sloppy input still gives a clean output. That property is noise margin, and it is why digital signals can be copied endlessly without decay.
5. Analogue design wants the cliff. Its slope is the voltage gain, often 20 to 50 for a simple CMOS inverter.
6. Bias an inverter at its switching point with a feedback resistor and you have an amplifier, not a logic gate. This is a real technique, used in crystal oscillator circuits.

4.8.4 PLAIN – what is really happening inside

1. In the flat regions one transistor is in its linear region acting as a small resistance and the other is fully cut off.
2. Almost the full supply appears across the off device and almost none across the on device, so current is essentially zero and so is power.
3. In the cliff region both devices are in saturation, acting as current sources. Two opposing current sources fight, and the output swings wildly for a small input nudge.
4. That is where gain comes from. It is not amplification of energy. The energy comes from the supply, and the transistor is a valve deciding how much of it reaches the output.
5. In the cliff region there is a real path from supply to ground, so current flows and heat is produced. Digital circuits cross that region in tens of picoseconds; analogue circuits sit in it and pay for it in heat.
6. The honest version: a transistor never amplifies power in the sense of creating it. It modulates a large power flow using a small one. Gain is a ratio of signals, never a violation of energy conservation.

4.8.5 TECHNICAL – the engineer's version

1. Small-signal voltage gain of a CMOS inverter at its switching threshold is minus (gm_n plus gm_p) times (ro_n in parallel with ro_p).
2. Intrinsic gain gm times ro for a single modern short-channel device is only about 10 to 30, and it has fallen as nodes shrank. That is why analogue design got harder, not easier, with scaling.
3. Noise margins: NMH = VOH minus VIH, and NML = VIL minus VOL. For standard CMOS at 5 V, VIH is 3.5 V, VIL is 1.5 V, VOH is 4.9 V and VOL is 0.1 V, giving margins near 1.4 V each way.
4. Radio front ends still commonly use bipolar or silicon-germanium devices for low-noise amplifiers, because their transconductance per unit current and their noise behaviour beat CMOS at the same power.

5. Where experts disagree: whether analogue functions should share the same advanced node as the digital logic. Integration saves packaging and interconnect; separate older nodes give better analogue devices and lower cost. Chiplet packaging has shifted this argument, not settled it.

4.8.6 WORDS - remember these

1. Transfer curve — the plot of output against input — DC voltage transfer characteristic, flat at both ends with a steep transition.
2. Noise margin — how much interference a signal can take — the voltage gap between guaranteed output levels and required input levels.
3. Gain — how much bigger the output change is — the slope of the transfer curve at the operating point.
4. Bias — where you park the device on the curve — the DC operating point set before any signal is applied.
5. Class A — an amplifier that conducts all the time — conduction over the full 360 degrees of the input cycle, at most 50 percent efficient.

4.9 The changing shape of the transistor

4.9.1 PLAIN - in simple words

1. For about forty years transistors were flat. The channel lay along the top surface of the wafer with the gate above it. That is a **planar** transistor.
2. As transistors shrank, the flat design began to fail in a specific way. The gate could only touch the channel from one side, the top, while the drain, sitting right next to a very short channel, started to influence it too.
3. So the gate lost the argument. It could no longer switch the channel fully off, and leakage rose sharply.
4. The fix was to change the shape: stand the channel up as a thin vertical fin and wrap the gate around three of its sides. That is a **FinFET**.
5. The next step wraps the gate around all four sides, making the channel a stack of thin horizontal ribbons with gate material threaded between them.
6. That is **gate-all-around**, also called nanosheet, and it is the current leading-edge shape.

4.9.2 PLAIN - a picture in your head

1. Imagine stopping water in a pipe by pressing with one hand from above.
2. Now use both hands and a thumb underneath. Three points of grip, much better control.
3. Now imagine a ring clamp closing all the way around. Total control from every direction.
4. Where this comparison breaks: the gate never squeezes anything. It only produces an electric field. But "surrounding it more completely gives more control" is exactly right.

4.9.3 PLAIN - a worked example

1. Here is the timeline of the shape change, with companies and years.

Shape	First volume use	Who
Planar bulk	1960s to 2011	Everyone
SOI planar	From 1998	IBM, AMD
FinFET (Tri-Gate)	2012	Intel 22 nm
FinFET	2015	TSMC, Samsung
GAA nanosheet	2022	Samsung 3 nm
GAA nanosheet	2025	TSMC N2, Intel 18A

2. Intel announced its 22 nanometre Tri-Gate process on 4 May 2011 and shipped Ivy Bridge processors using it in April 2012. That was the first high-volume FinFET product.

3. Samsung began production with 3 nanometre gate-all-around, marketed as MBCFET, in June 2022.
4. TSMC's N2 process entered volume production in the fourth quarter of 2025. It is TSMC's first gate-all-around node.

4.9.4 PLAIN - what is really happening inside

1. The problem the shape solves has a name: **short-channel effects**.
2. When the channel is long, the gate alone sets the barrier between source and drain. When it is very short, the drain's own depletion region reaches close to the source.
3. Now raising the drain voltage lowers the source barrier by itself, with no help from the gate. That is drain-induced barrier lowering, or DIBL.
4. Wrapping the gate around more of the channel restores its authority, because the gate's field now dominates from several directions at once.
5. The results are all bad: threshold voltage drops as the channel shortens, the subthreshold slope degrades so turn-off is less sharp, and off current rises.
6. Silicon-on-insulator, or SOI, attacks a related problem differently. A buried layer of oxide sits under the whole device, cutting off deep leakage paths through the substrate, reducing junction capacitance, and removing the latch-up failure mode entirely.
7. The gate insulator changed too, for a hard physical reason. Silicon dioxide was thinned generation after generation to keep the gate's grip strong, and Intel reached 1.2 nanometres of it, about five atomic layers, at 65 nanometres.
8. Five atoms is not a film any more. Electrons tunnel straight through it and leakage explodes.
9. The escape was a material with a higher dielectric constant, called **high-k**, which can be physically thicker while giving the same electrical coupling. Thicker means far less tunnelling.
10. Hafnium-based oxides have a dielectric constant around 20 to 25 against 3.9 for silicon dioxide, which buys back several atomic layers of thickness.
11. High-k films do not work with polysilicon gates, so metal gates had to return at the same time. Hence the paired name, high-k metal gate.

4.9.5 TECHNICAL - the engineer's version

1. Intel introduced high-k metal gate at 45 nanometres in 2007 with the Penryn family. Gordon Moore called it the biggest change to transistor technology since the polysilicon gate in the late 1960s.
2. Intel's published figures for that change: gate oxide leakage reduced more than 10 times, source-to-drain leakage more than 5 times, and drive current improved by more than 20 percent.
3. Equivalent oxide thickness (EOT) is the standard figure of merit: the thickness of silicon dioxide that would give the same capacitance per area. Modern EOT is around 0.8 to 1.0 nanometres, at a physical thickness of 2 to 3 nanometres.
4. FinFET geometry parameters are fin height, fin width (typically 5 to 8 nm) and fin pitch (roughly 24 to 30 nm at recent nodes).
5. FinFET width is quantized. You cannot ask for 1.5 fins, so device strength comes in integer multiples of one fin. That constrains circuit design in a way planar devices never did. Nanosheet removes the quantization, because sheet width is drawn continuously.
6. FD-SOI, fully depleted silicon-on-insulator, is a live alternative at 22 nanometres and below, offered by GlobalFoundries as 22FDX. It allows back-bias tuning of threshold voltage at runtime, which FinFET cannot do.
7. Where experts disagree: whether FD-SOI or FinFET suits low-power and radio products at moderate nodes. FinFET won the leading edge outright; FD-SOI retains real advocates for internet-of-things and automotive parts.
8. Beyond nanosheet, the announced direction is CFET, stacking an NMOS device directly on a PMOS device. As of 2026 that is active research and pathfinding, not production.

4.9.6 WORDS - remember these

1. Planar transistor — the old flat design — channel in the wafer surface with a gate above it on one side only.
2. FinFET — a channel on edge with the gate around three sides — tri-gate device with width quantized in integer fins.
3. Gate-all-around — gate wrapped completely around the channel — nanosheet or nanowire device, marketed as MBCFET by Samsung and RibbonFET by Intel.
4. Short-channel effects — the gate losing control as things shrink — DIBL, threshold roll-off, punch-through, subthreshold slope degradation.
5. SOI — a buried insulating layer under the devices — silicon-on-insulator, cutting substrate leakage, junction capacitance and latch-up.

4.10 Scale, Moore's Law, and the power wall

4.10.1 PLAIN - in simple words

1. A silicon atom is about 0.22 nanometres across, so a gate insulator one nanometre thick is four or five atoms.
2. In 1971 the Intel 4004 held 2,250 transistors. In 2024 the Apple M4 held about 28 billion. That is a factor of over twelve million in fifty-three years.
3. **Moore's Law** is the name for that trend. It is not a law of physics. It is an observation about industry that turned into a plan.
4. It is also not about speed, despite what almost everyone says.
5. It has been slowing since around 2015. Doubling now takes closer to three years than two, and each step costs far more than the last.

4.10.2 PLAIN - a picture in your head

1. Imagine a printing press that gets twice as good every two years, so a page holds twice as many words. For decades each new press also printed faster and used no more electricity. Better in every direction, free.
2. Then something changed. The presses kept holding more words but stopped getting faster, and started running hot.
3. That is exactly what happened to processors around 2005. Cores multiplied because clock speed could not rise.
4. Where this comparison breaks: printing more pages in parallel always helps, but computing in parallel only helps if the task can be split. Many cannot. That limit is called Amdahl's Law and it is not going away.

4.10.3 PLAIN - a worked example

1. Here is the transistor count table, with real parts and real years.

Chip	Year	Transistors
Intel 4004	1971	2,250
Intel 8086	1978	29,000
Intel 80386	1985	275,000
Intel Pentium	1993	3.1 million
Pentium 4	2000	42 million
Core 2 Duo	2006	291 million
Apple A7	2013	About 1 billion
Apple M1	2020	16 billion

Chip	Year	Transistors
Apple M4	2024	28 billion
Nvidia GB200	2024	208 billion
Cerebras WSE-3	2024	4 trillion

2. Now count switching events. Take the Apple M4: 28 billion transistors, a 4 GHz clock, and an activity factor of 5 percent.
3. 28e9 times 4e9 = 1.12 times 10 to the 20. Times 0.05 gives 5.6 times 10 to the 18 switching events every second.
4. That is 5.6 billion billion switches per second, in a chip you can cover with a thumb, drawing a few tens of watts.
5. The industry as a whole made about 250 times 10 to the 18 transistors in 2014, according to an IEEE Spectrum estimate. That is roughly 8 trillion transistors manufactured every second of that year.

4.10.4 PLAIN - what is really happening inside

1. Gordon Moore, then at Fairchild, wrote an article for Electronics magazine published on 19 April 1965, titled “Cramming more components onto integrated circuits”.
2. His actual observation: the complexity for minimum component costs had increased at a rate of roughly a factor of two per year, and he expected that to continue for at least ten more years.
3. Read that carefully. “Complexity for minimum component costs” means the number of components on the chip that is cheapest per component.
4. In 1975, at the IEEE International Electron Devices Meeting, Moore revised it: doubling yearly until about 1980, then slowing to about every two years.
5. The “every 18 months” version everyone quotes is not Moore’s. David House, an Intel executive, said in 1975 that more transistors plus faster ones would double performance about every 18 months.
6. Now the second law, the one that actually stopped. Robert Dennard and colleagues at IBM published a 1974 paper on designing ion-implanted MOSFETs with very small physical dimensions.
7. Their scaling rules: shrink every dimension by a factor k, shrink the supply voltage by k, and raise the doping by k.
8. The consequences are beautiful. Area falls by k squared, delay falls by k, power per device falls by k squared, and power per unit area stays constant.
9. Constant power density is the magic line. It means you can double the transistor count, run them faster, and the chip does not get hotter.
10. For roughly thirty years that held, and engineers came to treat it as normal. It was not normal. It was a gift.
11. It ended because supply voltage could not keep falling. Supply voltage must stay well above threshold voltage, and threshold voltage cannot fall without subthreshold leakage rising exponentially.
12. Section 4.6 gave the reason: the 59.6 millivolt per decade thermal floor. That is physics, at any temperature above absolute zero. So voltage stalled near 1 volt around 2005, having fallen from 5 volts.
13. Transistors kept shrinking, but the V squared term stopped shrinking.
14. Power density therefore began to climb. That is the **power wall**.
15. The industry’s answer was to stop raising clock speed, add cores, and leave large parts of the chip powered down at any moment. That last practice has a name: dark silicon.

4.10.5 TECHNICAL - the engineer’s version

1. Dennard constant-field scaling, with factor k greater than 1: dimensions divided by k, voltage divided by k, doping times k, current divided by k, capacitance divided by k, delay divided by k, power per device divided by k squared, power density unchanged.
2. Post-Dennard reality since roughly 2005: dimensions still divided by k, but voltage nearly constant, so power density rises by roughly k squared.

3. Node names are marketing, not measurements. This is a convention, not a standard. No physical feature on a "3 nm" process measures 3 nanometres.
4. The meaningful density metrics are contacted poly pitch (CPP), metal 1 pitch and cell height in track count. Leading nodes in 2025 sit around 45 to 50 nm CPP and 20 to 24 nm minimum metal pitch. Transistor density is quoted around 200 to 300 million per square millimetre.

Cause of slowdown	Effect
Atomic dimensions	Variability, few dopants
EUV tool and mask cost	Cost per wafer rises
Fab capital cost	Above 20 billion dollars
Design and verification	NRE cost per chip rises
Interconnect resistance	Wires now limit delay

5. Interconnect is the underrated one. Copper resistivity rises sharply as wire cross-sections approach the electron mean free path, so wire delay has scaled far worse than transistor delay.
6. Where experts disagree, sharply: whether cost per transistor still falls at the leading edge. Foundries state that it does. Several analysts argue it has been roughly flat since the 28 nanometre node. Nvidia's chief executive said publicly in 2022 that Moore's Law is dead. The honest answer is that it depends on volume, yield and which costs you count.
7. Established fact: transistor counts per chip are still rising. Active research: CFET stacking, two-dimensional channel materials such as molybdenum disulphide, and backside power delivery. Marketing claim: any specific node name, and any single number offered as "the density".
8. Tools to observe scale in practice: `lscpu` and `/proc/cpuinfo` on Linux for core and cache topology, and `hwloc-ls` for the physical layout.

```
# core count, cache sizes and topology
lscpu

# hierarchical view of packages, cores and caches
hwloc-ls
```

4.10.6 WORDS - remember these

1. Moore's Law — chips roughly double in transistor count every two years — an economic observation about components at minimum cost per component.
2. Dennard scaling — shrinking used to make chips faster and no hotter — constant-field scaling giving constant power density per unit area.
3. Power wall — the point where you cannot cool it any faster — the limit reached when power density stopped being constant under scaling.
4. Dark silicon — parts of the chip that must stay off — the fraction of a die that cannot be powered at once within the thermal budget.
5. Process node — the name of a manufacturing generation — a marketing label since roughly 2011, not a measurement of any feature.

4.11 What a transistor costs

4.11.1 PLAIN - in simple words

1. The most important number about the transistor is not its size or its speed. It is its price.
2. In the mid 1950s one transistor cost a few dollars. You counted them individually and designed circuits to use as few as possible.
3. Today a single dollar buys hundreds of millions of them.
4. That is a fall of roughly a billion times, and it is the reason every other chapter of this book is possible.

5. Caches, branch predictors, graphics units, encryption engines and neural accelerators all exist because transistors became too cheap to ration.

4.11.2 PLAIN - a picture in your head

1. Imagine paper cost as much as gold. You would write only what mattered, on both sides, in tiny letters.
2. Now imagine paper becomes free. You print drafts, take notes you never read, and keep three copies of everything.
3. Where this comparison breaks: transistors are not free, only individually cheap. A leading-edge chip design still costs hundreds of millions of dollars to bring to market.
4. The cost simply moved. It moved off the individual component and onto the design, the masks and the factory.

4.11.3 PLAIN - a worked example

1. Here are real price points across seventy years, as quoted at the time and not adjusted for inflation.

Year	Price per transistor	Note
1954	A few dollars	Early TI devices
1960	About 1 dollar	Small-quantity discrete
1964	About 4 dollars	Noyce, low volume
1971	Under 1 cent	Intel 4004 era
2024	A few billionths	Leading-edge logic

2. Work the modern number. Take a chip with 28 billion transistors whose finished die costs, say, 80 dollars to make.
3. 80 dollars divided by 28 billion is about 2.9 times 10 to the minus 9 dollars per transistor, roughly three billionths of a dollar.
4. Compared with about 1 dollar in 1960, that is a ratio of about 350 million to one.
5. In 1954 the Regency TR-1 pocket radio, announced on 18 October 1954, sold for $49.95 and contained four transistors.

4.11.4 PLAIN - what is really happening inside

1. The price collapse comes from one structural fact: cost scales with wafer area, not with transistor count.
2. Processing a wafer costs roughly the same whether the features on it are large or small. The steps are the same steps.
3. So halving the linear dimension fits four times as many transistors on the same wafer for nearly the same processing cost. That is the true engine of Moore's Law, and why Moore framed his observation in terms of cost per component.
4. Yield modifies this. A wafer has a certain number of defects, and a bigger die catches more of them, so bigger dies yield worse and not linearly. That is why the industry moved to chiplets: several small high-yielding dies packaged together.
5. The relationship has weakened in the last decade, because leading-edge wafers now cost far more, thanks to extreme ultraviolet lithography tools and masks.
6. So density still improves, but cost per transistor improves much less than it used to, and at some nodes it may not improve at all.
7. The honest version: "transistors keep getting cheaper" was true without qualification until roughly 2012. Since then it depends on the node, the volume and who is doing the accounting.

4.11.5 TECHNICAL - the engineer's version

1. Die cost is approximately wafer cost divided by (gross die per wafer times yield).
2. Yield follows a negative binomial model: Y = (1 + (D0 times A) divided by alpha) raised to the power minus alpha, where D0 is defect density per square centimetre, A is die area and alpha is a clustering factor.
3. Worked figures: at D0 of 0.1 defects per square centimetre and alpha of 3, a 100 square millimetre die yields about 90 percent, while a 600 square millimetre die yields about 42 percent. That yield curve, not physics, is why reticle-limit dies cost what they do.

Cost item	Rough scale, 2025
Leading-edge 300 mm wafer	18,000 to 30,000 USD
EUV mask set	Tens of millions USD
Leading-node design NRE	300 to 700 million USD
New leading-edge fab	20 billion USD and up

4. These figures are approximate and vary by customer, volume and contract. Foundry pricing is confidential, so all public numbers are estimates.
5. Non-recurring engineering cost is now the dominant barrier to entry. That is why only a handful of companies design at the leading node and everything else stays on mature 28 nanometre and 16 nanometre processes.
6. The consequence for a software engineer: compute is cheap and engineer-hours are expensive, so the industry trades transistors for productivity. That trade is why abstraction layers keep multiplying.
7. Established fact: cost per transistor fell by orders of magnitude from 1960 to about 2012. Active debate: whether it still falls at the leading edge. Marketing claim: any single cost-per-transistor number offered without stating node, volume and yield assumptions.

4.11.6 WORDS - remember these

1. Cost per transistor — what one switch costs you — die cost divided by transistor count, the metric Moore's 1965 paper was actually about.
2. Yield — the fraction of chips that work — the proportion of die on a wafer passing test, modelled against defect density and die area.
3. Defect density — how dirty the process is — D0, defects per square centimetre, around 0.1 or below for a mature leading process.
4. NRE — the one-off cost before you make any chips — non-recurring engineering cost, covering design, verification and mask sets.
5. Chiplet — several small dies packaged as one — a partitioning strategy trading packaging complexity for much better yield.

4.98 Common wrong ideas

1. Wrong: a transistor is a tiny mechanical switch. Right: nothing moves. A voltage or current changes how many charge carriers are available in a region, and that changes its conductivity.
2. Wrong: N-type silicon is negatively charged and P-type is positively charged. Right: both are electrically neutral. The letters name the majority carrier, not a net charge.
3. Wrong: a hole is a real positive particle. Right: it is a missing electron in a bond. The physics lets us treat it exactly as a positive particle, which is why the model is used, but nothing positive is moving.
4. Wrong: the gate of a MOSFET draws current to control the channel. Right: the gate is insulated and draws only a tiny tunnelling leakage. That is the whole reason CMOS logic can idle cheaply.
5. Wrong: saturation means the same thing in a BJT and a MOSFET. Right: BJT saturation means fully on with a low voltage drop. MOSFET saturation means pinched off with a nearly constant current.

6. Wrong: CMOS uses no power when idle. Right: it uses very little dynamic power when idle, but leakage is paid continuously, and since about the 90 nanometre generation it has been a major share of total power.
7. Wrong: Moore's Law says computers get twice as fast every 18 months. Right: Moore's 1965 statement was about components per chip at minimum cost, revised in 1975 to a two-year doubling. The 18-month performance version came from David House at Intel.
8. Wrong: clock speeds stopped rising because we ran out of transistor speed. Right: they stopped because power density stopped being constant when Dennard scaling ended, so faster clocks needed more voltage and produced unmanageable heat.
9. Wrong: a "3 nanometre" process has 3 nanometre transistors. Right: node names have been marketing labels since roughly 2011. No feature measures the node name. Use contacted poly pitch if you want a real number.
10. Wrong: Shockley invented the transistor. Right: Bardeen and Brattain built the first working device in December 1947 and are named on its patent. Shockley invented the later junction transistor. All three shared the 1956 Nobel Prize.

4.99 Chapter summary in 20 lines

1. A computer needs an electrically controlled switch, and the transistor is the best one ever found.
2. Relays worked but had to physically move, so they were slow. Zuse's Z3 of 1941 used about 2,600 of them at a few hertz.
3. Vacuum tubes were about a thousand times faster, but needed hot filaments, burned enormous power and failed constantly. ENIAC used about 17,468 of them and drew 150 kilowatts.
4. Pure silicon barely conducts, because every outer electron is locked into a bond.
5. Adding phosphorus or arsenic, with five outer electrons, gives spare electrons and makes N-type silicon.
6. Adding boron, with three outer electrons, leaves gaps called holes and makes P-type silicon. A hole is an absence that moves and behaves as a positive carrier.
7. One dopant atom per five million silicon atoms changes conductivity about a million-fold. That sensitivity is why purity matters so much.
8. Joining N and P material makes a depletion region and a built-in voltage of about 0.7 volts. That is a diode: easy one way, blocked the other.
9. Bardeen and Brattain made the first working transistor on 16 December 1947 at Bell Labs, announced on 30 June 1948, using germanium and two gold contacts.
10. Shockley, left off that patent, conceived the junction transistor in January 1948. It was working by 1951. All three shared the 1956 Nobel Prize.
11. The bipolar junction transistor uses a thin base to let a small base current control a collector current 100 to 300 times larger.
12. It burns power continuously, because base current can never stop while the device is on. That cost ruled it out for large digital chips.
13. The MOSFET, demonstrated by Atalla and Kahng at Bell Labs in 1959 and 1960, controls a channel through an insulator using an electric field alone.
14. Its gate draws essentially no current, so holding it on is free. Above the threshold voltage the surface inverts and a channel forms.
15. CMOS, invented by Wanlass and Sah at Fairchild in 1963, pairs an NMOS with a PMOS so exactly one is on in each steady state, leaving no path from supply to ground.
16. Dynamic power is alpha C V squared f. A billion nodes at 0.5 femtofarads, 1 volt, 3 gigahertz and 5 percent activity gives about 75 watts.
17. The same device is a switch at the flat ends of its transfer curve and an amplifier on the steep cliff between them.
18. Shapes changed because the gate lost control of short channels: planar until 2011, FinFET from Intel's

22 nanometre process in 2012, and gate-all-around nanosheet from Samsung in 2022 and TSMC and Intel in 2025.

19. Moore's 1965 observation was about components per chip at minimum cost, revised to two years in 1975. Dennard scaling ended around 2005, power density began rising, and cores multiplied instead of clock speeds.
20. The real story is price: from roughly one dollar per transistor in 1960 to a few billionths of a dollar today. Everything else in this book rests on that.

5

CHAPTER

KB

Logic Gates and Arithmetic — How Switches Learn to Add

5.0 What this chapter gives you

1. You will be able to explain how a rule about true and false became a rule about electricity, and who made that jump and when.
2. You will be able to draw a CMOS inverter, a NAND gate and a NOR gate from real transistors, and count how many transistors each one needs.
3. You will know all seven basic gates by truth table and by plain meaning, and be able to say what each one is for.
4. You will be able to build every gate out of NAND gates only, and say why chip factories care about that.
5. You will be able to take a written specification, turn it into a truth table, and shrink it with a Karnaugh map.
6. You will be able to build a half adder, a full adder and an 8-bit adder, and trace a carry moving through it bit by bit.
7. You will be able to explain how the same adder does subtraction, and how the hardware notices that a result went wrong.
8. You will be able to describe a multiplexer, decoder, comparator, barrel shifter and priority encoder, and say where each sits in a CPU.
9. You will be able to compute the maximum clock speed of a circuit from the delays of its gates.
10. You will be able to explain, from the hardware upward, why 0.1 plus 0.2 does not equal 0.3 on almost every computer you will ever touch.

5.1 Boolean algebra: true and false as 1 and 0

5.1.1 PLAIN - in simple words

1. Ordinary algebra works with numbers. You add them, multiply them, and follow rules.
2. Boolean algebra works with only two values: true and false.
3. We write true as 1 and false as 0. That is all the values there are.
4. It has three basic operations: AND, OR and NOT.
5. AND is true when both of its two inputs are true.
6. OR is true when at least one of its two inputs is true.
7. NOT flips a value. It turns true into false and false into true.
8. That tiny system is enough to describe every calculation a computer does.
9. A man named George Boole invented it in the 1840s and 1850s.
10. He was doing philosophy and mathematics. He was not thinking about machines, because there were no electronic machines yet.
11. Almost a hundred years later, a student noticed that electrical switches obey exactly those same rules.

12. That student was Claude Shannon, and his observation is why computers work the way they do.

5.1.2 PLAIN - a picture in your head

1. Think of a torch (a flashlight) with two switches wired one after the other along the same wire.
2. Current can only reach the bulb if switch one is closed AND switch two is closed.
3. One switch open anywhere in the line, and the bulb stays dark.
4. That wiring is an AND gate. Two switches in a line.
5. Now rewire the two switches side by side, each with its own path to the bulb.
6. Now the bulb lights if switch one is closed OR switch two is closed, or both.
7. That wiring is an OR gate. Two switches side by side.
8. So "in a line" means AND, and "side by side" means OR. Nothing more complicated than that is needed to start.
9. Where this comparison breaks: real gates do not pass the input current through to the output.
10. A real gate reads its inputs and then uses its own power supply to drive a fresh output. It is a decision, not a pipe.
11. That difference matters, because it is why you can chain a million gates together without the signal fading away.

5.1.3 PLAIN - a worked example

1. Take a door that should unlock only when a valid card is presented and the time is inside working hours.
2. Call the card signal C. C is 1 when a valid card is seen.
3. Call the time signal T. T is 1 when the clock is between 09:00 and 18:00.
4. The unlock rule is U = C AND T.
5. Here is every case.

C (card)	T (in hours)	U (unlock)
0	0	0
0	1	0
1	0	0
1	1	1

6. Now add a fire alarm. If the alarm F is 1, the door must open no matter what.
7. The new rule is U = (C AND T) OR F.
8. If F is 1, the OR makes U 1 whatever the left side says.
9. If F is 0, the OR passes the left side through unchanged.
10. You have just written a real access-control specification in Boolean algebra, and it can be built directly out of two gates.

5.1.4 PLAIN - what is really happening inside

1. Inside a chip there is no "true" and no "false". There is only voltage.
2. A wire is held at one of two voltages. In a typical modern core, about 0.75 volts or about 0 volts.
3. The high voltage is agreed to mean 1. The low voltage is agreed to mean 0.
4. That agreement is a choice made by the designers, not a law of nature.
5. There is a band of voltages in the middle that means nothing. Circuits are designed to pass through it as fast as possible and never rest there.
6. A gate is a small circuit that senses the voltages on its input wires.
7. It then connects its output wire either up to the power rail or down to ground, depending on what it senses.

8. Because the gate drives the output from its own power supply, the output is a clean full-strength 1 or 0.
9. That is called restoring the signal, and it is what lets you build circuits thousands of gates deep.
10. The honest version: the two voltages are not exact. A real 1 might be 0.71 volts on one wire and 0.78 volts on another.
11. The circuit only has to keep every 1 above a guaranteed minimum and every 0 below a guaranteed maximum. Those two numbers are in the datasheet.

5.1.5 TECHNICAL - the engineer's version

1. George Boole published "The Mathematical Analysis of Logic" in 1847 and "An Investigation of the Laws of Thought" in 1854.
2. Boole's system was a two-valued algebra over the set $\{0, 1\}$ with the operations we now call conjunction, disjunction and complement.
3. Augustus De Morgan, a contemporary and correspondent of Boole, published "Formal Logic" in 1847, containing the two duality laws that carry his name.
4. Claude Elwood Shannon submitted his master's thesis at the Massachusetts Institute of Technology on 10 August 1937, titled "A Symbolic Analysis of Relay and Switching Circuits".
5. A revised, abridged version appeared in the Transactions of the American Institute of Electrical Engineers, volume 57, in 1938, pages 713 to 723.
6. The paper won the Alfred Noble Prize in 1939. That prize is named after an American engineer, not after Alfred Nobel of the Nobel Prizes.
7. In 1985 the psychologist Howard Gardner described it as "possibly the most important, and also the most famous, master's thesis of the century".
8. Shannon's contribution was the mapping: a relay contact in series is a Boolean product, a relay contact in parallel is a Boolean sum, and a normally-closed contact is a complement.
9. That mapping turned circuit design from trial and error into algebra. You can now simplify a circuit by simplifying an expression.
10. Modern logic levels are defined per family. Some real figures:

Family	Supply	Minimum input high
74LS TTL	5.0 V	2.0 V
74HC CMOS	5.0 V	3.5 V
74LVC CMOS	3.3 V	2.0 V
LVCMOS18	1.8 V	1.17 V

11. These are standards written in vendor datasheets and in JEDEC documents, not conventions. A part that violates them is defective.
12. Positive logic means high voltage equals 1, and it is a near-universal convention. Negative logic, where low means 1, still appears on reset and chip-select pins, marked with an overbar or a leading letter n.

5.1.6 WORDS - remember these

1. Boolean algebra — maths with only true and false — a two-element algebraic structure over $\{0, 1\}$ with AND, OR and NOT.
2. Bit — one true-or-false value — one binary digit, the smallest unit of information.
3. Logic level — which voltage counts as 1 — a specified voltage band with guaranteed input and output thresholds.
4. Gate — a small circuit that makes one decision — a combinational element implementing one Boolean function.
5. Truth table — a list of every input case and its answer — the complete functional specification of a combinational circuit.

6. Positive logic — high voltage means 1 — the near-universal signalling convention in current CMOS families.

5.2 Building gates from real MOSFETs

5.2.1 PLAIN - in simple words

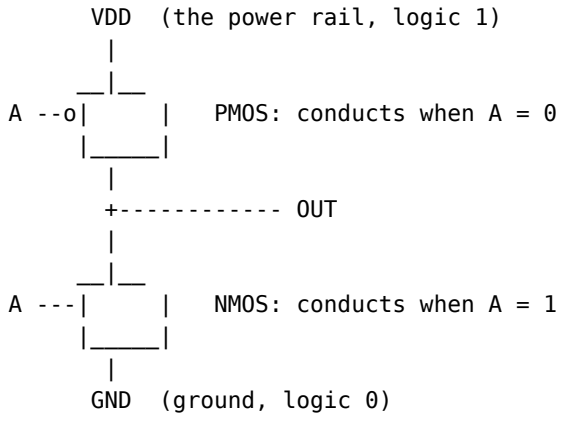
1. A transistor is a switch with no moving parts, controlled by a voltage on a third wire called the gate.
2. Modern chips use two kinds of transistor together. The style is called CMOS.
3. An NMOS transistor conducts when its gate is high. High turns it on.
4. A PMOS transistor conducts when its gate is low. Low turns it on. It is the opposite.
5. Every CMOS gate has two halves. A group of PMOS transistors on top, joined to the power supply.
6. And a group of NMOS transistors underneath, joined to ground.
7. The output wire sits between the two halves.
8. The design rule is simple. Exactly one half must be conducting at any time.
9. If the top half conducts, the output is pulled up to the power supply and reads as 1.
10. If the bottom half conducts, the output is pulled down to ground and reads as 0.
11. If both halves ever conducted at once, current would pour straight from power to ground and the chip would waste energy and heat up.

5.2.2 PLAIN - a picture in your head

1. Imagine a bucket with a tap above it and a plug hole below it.
2. The tap is the PMOS half. The plug hole is the NMOS half.
3. Open the tap and close the plug: the bucket fills. That is output 1.
4. Close the tap and open the plug: the bucket empties. That is output 0.
5. The whole trick of CMOS is that the two are wired to be opposites. Whenever one opens, the other closes.
6. So the bucket is always either full or empty, and never both filling and draining at once.
7. Where this comparison breaks: water takes real time to fill a bucket, and so does charge, but the amount of charge involved is tiny.
8. The other break is that a real gate does pass a very small current while it is switching over, in the moment when both halves are briefly part-open.
9. That brief overlap is called short-circuit current, and it is a genuine part of a chip's power budget, though usually a small part.

5.2.3 PLAIN - a worked example

1. Here is the simplest CMOS gate. It has one input and one output, and it flips the value. It is the NOT gate, also called an inverter.



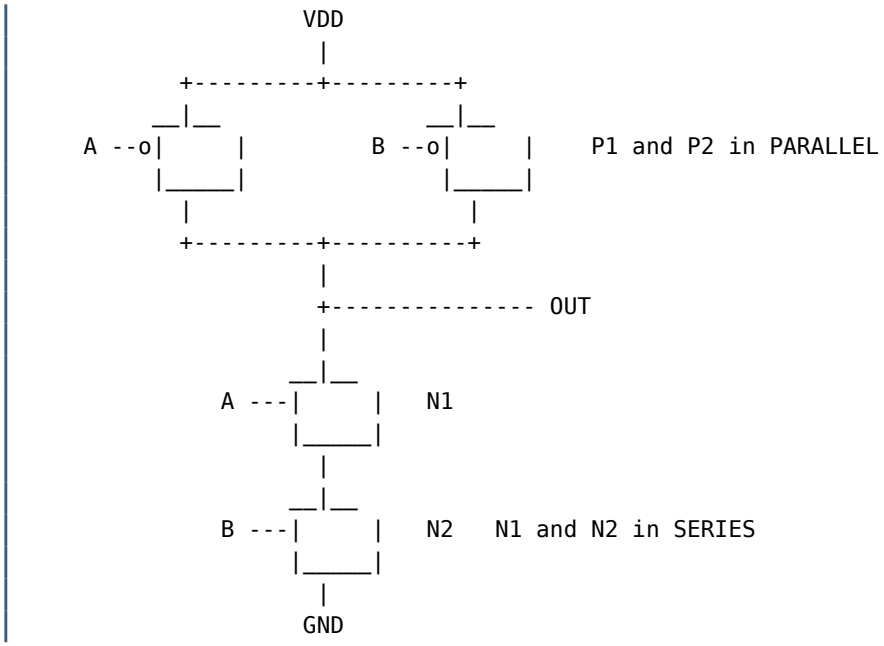
2. The small circle on the PMOS gate is the standard way to mark "this one is on when the input is low".

3. Case A = 0. The PMOS is on, because its gate is low. The NMOS is off, because its gate is low. The output is connected up to VDD. OUT = 1.
4. Case A = 1. The PMOS is off, because its gate is high. The NMOS is on. The output is connected down to GND. OUT = 0.
5. Two transistors. That is the entire NOT gate.
6. Note what the table shows: the output is always the opposite of the input, which is exactly the definition of NOT.

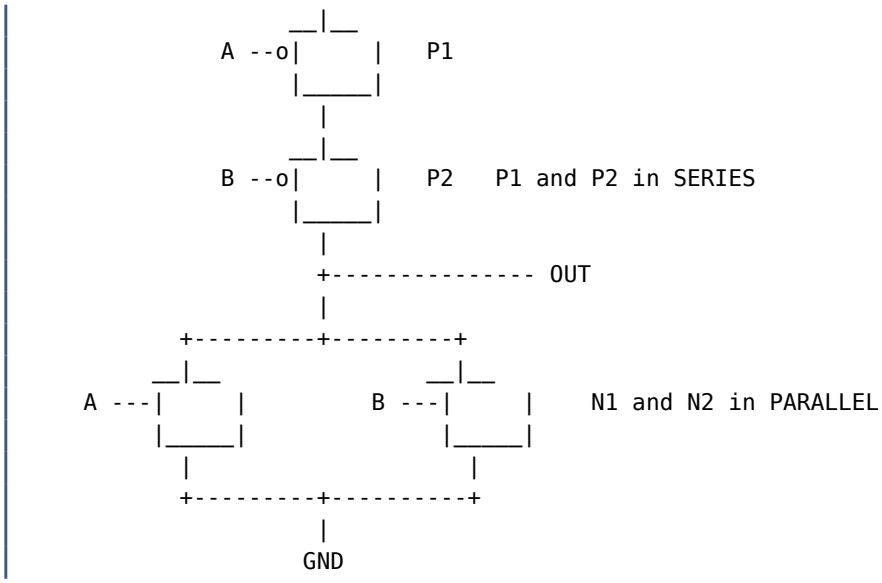
A	PMOS	NMOS	OUT
0	on	off	1
1	off	on	0

5.2.4 PLAIN - what is really happening inside

1. Now the NAND gate. NAND means NOT-AND. Its output is 0 only when both inputs are 1.
2. The rule for building any CMOS gate is a rule of shapes. For AND-like behaviour, put transistors in series. For OR-like behaviour, put them in parallel.
3. The NMOS half of a NAND has the two transistors in series, so the output can only be pulled down when both inputs are high.
4. The PMOS half has the two transistors in parallel, so the output is pulled up if either input is low.



5. Case A=0, B=0. Both PMOS are on. Both NMOS are off. Output pulled up. OUT = 1.
6. Case A=0, B=1. P1 is on, P2 is off, so the parallel pull-up still conducts. N1 is off, so the series pull-down is broken. OUT = 1.
7. Case A=1, B=0. P1 is off, P2 is on, so the pull-up still conducts. N2 is off, so the pull-down is broken. OUT = 1.
8. Case A=1, B=1. Both PMOS are off. Both NMOS are on, so the series chain is complete. Output pulled down. OUT = 0.
9. Four transistors. That is the entire NAND gate.
10. The NOR gate is the same idea with the two halves swapped over. PMOS in series on top, NMOS in parallel underneath.



11. Case A=0, B=0. Both PMOS on, series chain complete, pull-up wins. OUT = 1.
12. Case A=0, B=1. P2 is off, so the series pull-up is broken. N2 is on, so the parallel pull-down conducts. OUT = 0.
13. Case A=1, B=0. P1 is off, pull-up broken. N1 is on, pull-down conducts. OUT = 0.
14. Case A=1, B=1. Both PMOS off. Both NMOS on. OUT = 0.
15. Four transistors again.
16. Notice something important. Both of these natural, cheap, four-transistor gates produce an inverted output. CMOS is naturally inverting.
17. To get a plain AND you must build a NAND and then add an inverter, which costs two more transistors. AND is six transistors, not four.

5.2.5 TECHNICAL - the engineer's version

1. MOSFET stands for metal-oxide-semiconductor field-effect transistor. The gate terminal is insulated from the channel by a thin oxide layer.
2. An NMOS device conducts when the gate-to-source voltage exceeds its threshold voltage, typically around 0.2 to 0.4 volts in modern low-voltage processes.
3. CMOS as a circuit style was invented in 1963 by Frank Wanlass and Chih-Tang Sah at the Fairchild Semiconductor research laboratory.
4. They presented "Nanowatt Logic Using Field-Effect Metal-Oxide Semiconductor Triodes" at the International Solid-State Circuits Conference on 20 February 1963.
5. Wanlass received United States patent 3,356,858, filed 18 June 1963 and issued 5 December 1967.
6. The headline property of CMOS is that static power is near zero. Current flows mainly during switching, to charge and discharge load capacitance.
7. The dynamic power of a CMOS node is approximately P = a x C x V x V x f, where a is the activity factor, C the switched capacitance, V the supply voltage and f the frequency.
8. Because power scales with the square of the voltage, supply voltages fell from 5 volts in the 1980s to roughly 0.7 to 0.9 volts in current logic.
9. The honest version: static power is no longer negligible. At small geometries, sub-threshold leakage and gate-oxide tunnelling leakage can be a large share of total power, which is why unused blocks are power-gated.
10. Transistor counts for static complementary CMOS gates:

Gate	Transistors	Note
NOT (inverter)	2	1 PMOS, 1 NMOS
2-input NAND	4	PMOS parallel
2-input NOR	4	PMOS series
2-input AND	6	NAND plus inverter

11. A 2-input OR is likewise 6 transistors: a NOR plus an inverter.
12. A static CMOS 2-input XOR is commonly drawn with 12 transistors. A transmission-gate XOR needs 6, and various pass-transistor designs need 4. The exact count is an implementation detail of the standard-cell library, not a property of XOR.
13. In a real chip you do not draw transistors. You instantiate cells from a standard-cell library such as NAND2_X1 or INVX4, where the trailing number is the drive strength.
14. Tools that let you observe this: Yosys and OpenLane for open synthesis flows, ngspice for transistor-level simulation, and Magic or KLayout for viewing layout.

5.2.6 WORDS - remember these

1. MOSFET — a voltage-controlled switch — a field-effect transistor with an insulated gate over a channel.
2. NMOS — the type that turns on with a high gate — an n-channel enhancement MOSFET conducting when Vgs is above threshold.
3. PMOS — the type that turns on with a low gate — a p-channel enhancement MOSFET conducting when Vgs is below its negative threshold.
4. CMOS — using both types together — complementary metal-oxide-semiconductor logic with a pull-up network and a complementary pull-down network.
5. Pull-up network — the half that connects the output to power — the PMOS network conducting for input patterns that produce a 1.
6. Pull-down network — the half that connects the output to ground — the NMOS network conducting for input patterns that produce a 0.
7. Standard cell — a ready-made gate layout — a characterized, pre-laid-out logic cell in a foundry library with published timing and power data.

5.3 The seven gates

5.3.1 PLAIN - in simple words

1. There are seven gates you must know. Three basic ones and four built from them.
2. AND: output 1 only when both inputs are 1. Plain meaning: “both of them”.
3. OR: output 1 when at least one input is 1. Plain meaning: “either one, or both”.
4. NOT: one input, output is the opposite. Plain meaning: “the reverse”.
5. NAND: AND with the answer flipped. Plain meaning: “not both of them”.
6. NOR: OR with the answer flipped. Plain meaning: “neither of them”.
7. XOR: output 1 when the two inputs differ. Plain meaning: “exactly one of them”.
8. XNOR: XOR with the answer flipped. Plain meaning: “the two are the same”.
9. That is the whole vocabulary. Every digital circuit ever built is these seven, repeated.

5.3.2 PLAIN - a picture in your head

1. Picture two people voting on whether to go out.
2. AND is a strict rule: we go only if both say yes.
3. OR is a relaxed rule: we go if anybody says yes.
4. NAND is a veto rule reversed: we go unless both say yes.
5. NOR is a very strict rule: we go only if nobody says yes.

6. XOR is a disagreement detector: it lights up exactly when the two people disagree.
7. XNOR is an agreement detector: it lights up exactly when they agree.
8. Where this comparison breaks: people can abstain, hesitate, or change their mind halfway. A gate input is always exactly 0 or exactly 1 at the moment the answer is read.
9. Real wires do spend a few picoseconds in between while changing. Circuits are designed so nobody reads the answer during that window. Chapter 6 covers how that is arranged with a clock.

5.3.3 PLAIN - a worked example

1. Here are all seven truth tables. Each has at most two inputs.

AND — "both of them":

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR — "either one, or both":

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT — "the reverse":

A	NOT A
0	1
1	0

NAND — "not both of them":

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR — "neither of them":

A	B	A NOR B
0	0	1
0	1	0
1	0	0

A	B	A NOR B
1	1	0

XOR — "exactly one of them":

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR — "the two are the same":

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

2. Read down the last column of NAND and compare with AND. Every row is flipped. That is all "N" in front of a name means.
3. Read down XOR. It is 1 in exactly the two rows where A and B are different.
4. XOR is the single most useful gate in arithmetic, because adding 1 and 1 in binary gives 0 with a carry, and XOR gives exactly that 0.

5.3.4 PLAIN - what is really happening inside

1. Nothing about a gate knows the word "AND". The gate is a shape of transistors, and the truth table is what that shape does.
2. A gate does not compute in steps. It settles. You change the inputs, and after a short delay the output arrives at the right value.
3. That delay is called propagation delay, and it is the single most important number about a gate.
4. Gates with more inputs are slower, because more transistors sit in series in one of the two halves.
5. That is why a 4-input NAND is slower than a 2-input NAND, and why libraries usually stop at three or four inputs and build wider functions as trees.
6. XOR is not a natural CMOS shape. There is no simple series and parallel arrangement that produces "the inputs differ".
7. So XOR is built from other gates or from special transistor tricks, and it is always more expensive and slower than NAND or NOR.
8. This has a real consequence you will meet in section 5.6: an adder is mostly XOR gates, and that is a large part of why addition is not free.

5.3.5 TECHNICAL - the engineer's version

1. Standard symbols come from two competing sets: the distinctive shapes of ANSI/IEEE 91-1984, and the rectangular symbols of IEC 60617-12.
2. In practice, distinctive shapes dominate American and textbook usage, and rectangles dominate European formal drawings. This is a convention split, not a correctness issue.
3. In hardware description languages the operators are written as symbols. Verilog uses & for AND, | for OR, ~ for NOT and ^ for XOR.

```
wire y_and  =  a & b;
wire y_or   =  a | b;
wire y_not  = ~a;
wire y_nand = ~(a & b);
wire y_nor  = ~(a | b);
wire y_xor  =  a ^ b;
wire y_xnor = ~(a ^ b);
```

4. Discrete parts in the 7400 family, first shipped by Texas Instruments as the military 5400 series in October 1964 and the commercial plastic 7400 series in the third quarter of 1966:

Part	Function	Gates per package
7400	quad 2-input NAND	4
7402	quad 2-input NOR	4
7404	hex inverter	6
7486	quad 2-input XOR	4

5. The 7408 is quad 2-input AND and the 7432 is quad 2-input OR. It is not an accident that the very first part in the family, the 7400, was a NAND.
6. Typical transistor counts in a static CMOS standard-cell library:

Cell	Transistors	Relative delay
INV	2	1.0
NAND2	4	about 1.4
NOR2	4	about 1.8
XOR2	8 to 12	about 2.5

7. The relative delay figures are indicative for a typical library at equal drive strength, not a specification. Exact values come from the foundry characterization data, usually in Liberty format files with a .lib extension.
8. NOR2 is slower than NAND2 at equal area, because a NOR's series transistors are PMOS. Hole mobility is two to three times lower than electron mobility, so those PMOS must be made wider.
9. Fan-in is the number of inputs to a gate. Fan-out is the number of gate inputs a single output drives. Both increase delay, fan-out roughly linearly.
10. The standard unit for comparing logic speed across process nodes is the fan-out-of-four inverter delay, written FO4. It is the delay of an inverter driving four copies of itself.

5.3.6 WORDS - remember these

1. XOR — exactly one of them is 1 — the two-input parity function, also called modulo-2 addition.
2. XNOR — the two are the same — the equivalence function, the complement of XOR.
3. NAND — not both of them — the complement of conjunction, and a functionally complete operator.
4. NOR — neither of them — the complement of disjunction, also functionally complete.
5. Propagation delay — how long a gate takes to answer — the time from an input crossing 50 percent of the supply to the output crossing 50 percent.
6. Fan-in — how many inputs a gate has — the count of gate inputs, which raises delay through series transistor stacking.
7. Fan-out — how many inputs one output feeds — the capacitive load on a driver, which raises delay roughly in proportion.
8. FO4 — a fair speed yardstick — the delay of an inverter loaded by four identical inverters, used to compare process technologies.

5.4 Universality: why NAND alone is enough

5.4.1 PLAIN - in simple words

1. Here is a surprising fact. You do not need seven kinds of gate. You need one.
2. If you have an unlimited supply of NAND gates and wire, you can build every other gate.
3. And if you can build every other gate, you can build every digital circuit that has ever existed.
4. A gate that can do this on its own is called universal, or functionally complete.
5. NAND is universal. NOR is universal too, on its own, in exactly the same way.
6. AND is not universal. Neither is OR. Neither is XOR. None of them can produce a NOT.
7. The reason is simple: AND, OR and XOR of two 0s all give 0. They can never turn a 0 into a 1 on their own.
8. NAND can. NAND of two 0s gives 1. That ability to invert is the key.

5.4.2 PLAIN - a picture in your head

1. Think of a language with only one word, where the word changes meaning depending on how you repeat and arrange it.
2. That sounds useless, but it is exactly what NAND is. One operation, rearranged, becomes all the others.
3. A closer everyday comparison is Lego bricks. There are many shapes, but you could build almost any model from one single brick shape, given enough of them.
4. It would be less elegant and use more bricks, but it would work, and you would only need one bin of parts.
5. Where this comparison breaks: with Lego, using one brick shape is a handicap. With NAND, it is often an advantage.
6. NAND is the cheapest and fastest gate CMOS can make, so building things out of NAND is not a sacrifice. It is frequently the best option.

5.4.3 PLAIN - a worked example

1. NOT from one NAND. Tie both inputs together.

```
A ---+
     |---[ NAND ]--- OUT = NOT A
A ---+
```

2. Check it. If A = 0, then NAND(0,0) = 1. If A = 1, then NAND(1,1) = 0. That is NOT.
3. AND from two NANDs. NAND then invert.

```
A ---+
     +--> NAND1 --- X ---+
B ---+                   +--> NAND2 ---> OUT = A AND B
                         +
Both inputs of NAND2 are tied to X, so NAND2 is an inverter.
```

4. Written plainly: X = A NAND B, then OUT = X NAND X. Two gates.
5. OR from three NANDs. Invert both inputs first, then NAND them.

```
A ---+--> NAND1 --- P ---+
A ---+                   |
                         +--> NAND3 ---> OUT = A OR B
B ---+--> NAND2 --- Q ---+
B ---+

NAND1 and NAND2 are inverters, so P = NOT A and Q = NOT B.
```

6. Written plainly: P = NOT A, Q = NOT B, OUT = P NAND Q. Three gates.

7. Why does that give OR? Because NAND(NOT A, NOT B) is NOT(NOT A AND NOT B), and "not (neither)" means "at least one". That is OR.
8. XOR from four NANDs. This is the classic arrangement.

```
X = A NAND B
Y = A NAND X
Z = B NAND X
OUT = Y NAND Z          and this equals A XOR B

A ---+--> NAND1 ---+--> NAND2 --- Y ---+
     |      ^      |       ^           |
B ---+------+      |       A           +--> NAND4 --> OUT
     |             |                   |
     +-------------+--> NAND3 --- Z ---+
                           ^
                           B
```

9. Step by step with A = 1, B = 0.
10. NAND1 = NAND(1, 0) = 1.
11. NAND2 = NAND(A, NAND1) = NAND(1, 1) = 0.
12. NAND3 = NAND(B, NAND1) = NAND(0, 1) = 1.
13. NAND4 = NAND(0, 1) = 1. And XOR(1, 0) = 1. Correct.
14. Now A = 1, B = 1. NAND1 = 0. NAND2 = NAND(1, 0) = 1. NAND3 = NAND(1, 0) = 1. NAND4 = NAND(1, 1) = 0. And XOR(1, 1) = 0. Correct.

Gate built	NAND gates needed
NOT	1
AND	2
OR	3
XOR	4

15. NOR from NAND takes 4 as well: build OR with 3, then invert with 1.

5.4.4 PLAIN - what is really happening inside

1. The deep reason NAND works is that it combines two abilities in one gate.
2. It can combine two signals, and it can invert. Any gate that can do both can build everything.
3. A formal way to see it: every Boolean function can be written as a sum of products, meaning ORs of ANDs of inputs and inverted inputs.
4. If you can make AND, OR and NOT, you can write any sum of products. And NAND makes all three.
5. So NAND builds every possible function of any number of inputs. There are no exceptions and no special cases.
6. NOR is universal for the mirror-image reason: it can combine and it can invert, so it builds product of sums instead.
7. The honest version: universality is a statement about what is possible, not about what is sensible.
8. Building a 64-bit multiplier out of nothing but 2-input NAND gates is possible and would be enormous and slow. Real designs use whatever cell in the library is cheapest for the job.

5.4.5 TECHNICAL - the engineer's version

1. Functional completeness of the Sheffer stroke was shown by Henry M. Sheffer in his 1913 paper in the Transactions of the American Mathematical Society. The Sheffer stroke is NAND.
2. Charles Sanders Peirce had found the same result earlier, in unpublished work dated around 1880. The NOR operator is still called Peirce's arrow.

3. NAND is preferred over NOR in CMOS for a physical reason. In a NAND, the series stack is NMOS and the parallel network is PMOS.
4. Electron mobility in silicon is roughly 1400 square centimetres per volt-second, while hole mobility is roughly 450. The ratio is about 3 in bulk, and about 2 to 2.5 for carriers in a MOSFET inversion layer.
5. Because PMOS devices carry less current per unit width, a series PMOS stack in a NOR must be made two to three times wider than the equivalent NMOS stack in a NAND to reach the same speed.
6. Wider transistors mean more area, more input capacitance and more power. So NOR2 costs more than NAND2 for the same performance.
7. Consequence: standard-cell libraries are NAND-heavy, synthesis tools map preferentially onto NAND and inverter, and NAND2_X1 is typically the most instantiated cell on a die.
8. NAND flash memory is named for the same series-stack idea, with memory cells in series along a bit line. It is a separate use of the word from NAND logic and the two should not be confused.
9. In FPGA design the picture changes completely. An FPGA has no gates. It has lookup tables, typically 6-input LUTs in current Xilinx and AMD parts and in Intel and Altera adaptive logic modules.
10. A 6-input LUT is a 64-bit memory addressed by the six inputs. It implements any function of six variables at identical cost, so gate-count reasoning does not apply. This is an implementation detail of FPGAs, not of logic.

5.4.6 WORDS - remember these

1. Universal gate — one gate that can build all the rest — a functionally complete operator whose closure is all Boolean functions.
2. Functional completeness — nothing else is needed — the property that every Boolean function is expressible using only the given operator set.
3. Sheffer stroke — another name for NAND — the binary connective written as a vertical bar, proved complete by Sheffer in 1913.
4. Peirce arrow — another name for NOR — the dual complete connective, attributed to Charles Sanders Peirce.
5. LUT — a small table that fakes any gate — a lookup table in an FPGA, a configuration memory addressed by the input signals.
6. Technology mapping — turning a design into real cells — the synthesis step that covers a Boolean network with cells from a target library.

5.5 Writing and simplifying logic

5.5.1 PLAIN - in simple words

1. A truth table says what a circuit must do. An expression says how to build it. You need to move between the two.
2. Going from a table to an expression is mechanical. Look at every row where the output is 1.
3. For each such row, write an AND of all the inputs, with a NOT on any input that is 0 in that row.
4. Then OR all those AND terms together. You now have a working circuit.
5. That form is called sum of products, because OR behaves like a sum and AND behaves like a product.
6. The expression you get this way is correct but usually wasteful. It has one AND term for every 1 in the table.
7. Simplifying means finding a smaller expression that gives exactly the same table. Fewer gates, less area, less power, less delay.
8. There are two ways to simplify. By algebra, using rules. Or by drawing a picture called a Karnaugh map and spotting groups by eye.

5.5.2 PLAIN - a picture in your head

1. Think of writing driving directions. The literal version lists every single junction: "at junction 1 go straight, at junction 2 go straight, at junction 3 turn left".
2. The simplified version says: "go straight until the church, then turn left".
3. Both get you to the same place. The second is shorter because it noticed a run of identical steps and merged them.
4. Simplifying logic is exactly that. You look for groups of rows that agree on most inputs, and merge them into one shorter rule.
5. Where this comparison breaks: with directions there is one obvious way to merge. With logic there can be several different smallest answers, all equally good.
6. Also, the smallest expression is not always the fastest circuit. A shallower circuit with more gates can beat a deeper circuit with fewer.

5.5.3 PLAIN - a worked example

1. Here are the core identities. A prime mark means NOT.

Name	Rule
Identity	A + 0 = A, A . 1 = A
Null	A + 1 = 1, A . 0 = 0
Idempotent	A + A = A, A . A = A
Complement	A + A′ = 1, A . A′ = 0
Involution	(A′)′ = A
Absorption	A + A.B = A
Distributive	A.(B + C) = A.B + A.C
Consensus	A.B + A′.C + B.C = A.B + A′.C

2. In these lines a dot means AND and a plus means OR. That is the standard written shorthand.
3. Now De Morgan's two laws, proved by writing out every case.
4. First law: NOT (A AND B) equals (NOT A) OR (NOT B).

A	B	(A.B)′	A′ + B′
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

5. The last two columns match on every row. The law holds. That is a complete proof, because there are only four cases and we checked all four.
6. Second law: NOT (A OR B) equals (NOT A) AND (NOT B).

A	B	(A+B)′	A′ . B′
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

7. Again both columns match on every row. The law holds.
8. In plain words: "not both" is the same as "either one is missing", and "neither" is the same as "this one is missing and that one is missing".

5.5.4 PLAIN - what is really happening inside

1. Now a complete Karnaugh map for four variables, from the table to the answer.
2. The specification: four sensor bits A, B, C and D arrive, with A as the most significant bit. Output F must be 1 for these input values, and 0 otherwise.
3. This function is invented for teaching. The method is the real thing.

Value	A B C D	F
0	0 0 0 0	1
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	0
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	1
11	1 0 1 1	1
12	1 1 0 0	0
13	1 1 0 1	0
14	1 1 1 0	1
15	1 1 1 1	1

4. Step 1. Write the raw sum of products straight from the ten rows with a 1. That is ten AND terms of four inputs each. It works and it is horrible.
5. Step 2. Draw the map. The rows are AB, the columns are CD, and both are listed in the order 00, 01, 11, 10.
6. That strange order is Gray code. Only one bit changes between neighbours, so squares that touch differ in exactly one variable.

```
            CD=00   CD=01   CD=11   CD=10
AB=00   |   1   |   1   |   1   |   1   |   (m0  m1  m3  m2)
AB=01   |   1   |   1   |   0   |   0   |   (m4  m5  m7  m6)
AB=11   |   0   |   0   |   1   |   1   |   (m12 m13 m15 m14)
AB=10   |   0   |   0   |   1   |   1   |   (m8  m9  m11 m10)
```

7. Step 3. Find the biggest rectangles of 1s. Sizes must be 1, 2, 4, 8 or 16, and rectangles may wrap around the edges.
8. Group one: the whole top row, four squares, m0 m1 m3 m2. Across that row A is always 0 and B is always 0, while C and D take every value. So the term is A'B'.
9. Group two: the left two columns of the top two rows, m0 m1 m4 m5. Here A is always 0 and C is always 0. The term is A'C'.
10. Group three: the right two columns of the bottom two rows, which is m15, m14, m11 and m10. Across those four squares A is always 1 and C is always 1. The term is AC.
11. Step 4. Check the cover. A'B' covers 0, 1, 2, 3. A'C' covers 0, 1, 4, 5. AC covers 10, 11, 14, 15. Together that is all ten 1s and nothing else.
12. Step 5. Write the answer.

```
F = A'B' + A'C' + AC
```

13. Ten four-input AND terms became three two-input AND terms. That is three AND gates, one OR gate and two inverters, instead of dozens of gates.
14. Sanity check value 6, which is A=0 B=1 C=1 D=0. A'B' is 0 because B is 1. A'C' is 0 because C is 1. AC is 0 because A is 0. So F = 0, matching the table.
15. Sanity check value 11, which is A=1 B=0 C=1 D=1. A'B' is 0 and A'C' is 0, both because A is 1. AC is 1. So F = 1, matching the table.

5.5.5 TECHNICAL - the engineer's version

1. The canonical sum of products form is also called the minterm expansion or disjunctive normal form. The dual is the maxterm expansion, or conjunctive normal form.
2. A minterm is a product term containing every variable exactly once. Minterm m5 of four variables is A'BC'D, because 5 is binary 0101.
3. The standard notation for the function above is F(A,B,C,D) = sum of m(0, 1, 2, 3, 4, 5, 10, 11, 14, 15).
4. Maurice Karnaugh published "The Map Method for Synthesis of Combinational Logic Circuits" in the Transactions of the American Institute of Electrical Engineers in November 1953.
5. It refined Edward W. Veitch's 1952 Veitch chart, which was itself a rediscovery of Allan Marquand's logical diagram of 1881.
6. Karnaugh maps are practical up to four variables, awkward at five and six, and useless beyond that. Beyond four variables, use an algorithm.
7. The Quine-McCluskey algorithm, published by Willard Quine in 1952 and extended by Edward McCluskey in 1956, gives a guaranteed minimal cover but has exponential worst-case cost.
8. Production tools use Espresso, developed at the University of California, Berkeley in the early 1980s. Espresso is a heuristic minimizer: it is fast and very good, but it does not promise the absolute minimum.
9. A prime implicant is a product term that cannot be enlarged further without covering a 0. An essential prime implicant is one that uniquely covers at least one minterm.
10. In the worked map, all three terms are essential prime implicants, so the minimal cover is unique. That is not always the case.
11. Don't-care conditions, written X or d, are input combinations that cannot occur. They may be read as 1 or 0, whichever makes groups larger. Binary-coded decimal decoders are the classic case: 10 to 15 never occur.
12. De Morgan's laws generalize to any number of variables and are the formal basis of bubble pushing, the visual technique of moving inversion circles across a gate symbol while swapping AND for OR.
13. Modern practice: nobody minimizes by hand for production. You write the behaviour in Verilog, SystemVerilog or VHDL and a synthesis tool such as Synopsys Design Compiler, Cadence Genus or the open-source Yosys performs minimization and technology mapping.
14. You still do it by hand to read a datasheet, to debug a synthesis result, and to pass an interview.

5.5.6 WORDS - remember these

1. Sum of products — ORs of ANDs — disjunctive normal form, a two-level AND-OR realization.
2. Minterm — one row of the table as a term — a product containing every variable in true or complemented form.
3. Karnaugh map — a grid that makes groups visible — a Gray-coded truth table arrangement placing logically adjacent minterms physically adjacent.
4. Gray code — an order where one bit changes at a time — a reflected binary code with unit Hamming distance between neighbours.
5. Prime implicant — a group that cannot grow — a product term implying the function that is not contained in any larger such term.
6. Don't care — a case that cannot happen — an unspecified output used to enlarge groups during minimization.

7. De Morgan's laws — not both means either is missing — the dual identities $(A.B)' = A' + B'$ and $(A+B)' = A'.B'$.

5.6 Adding

5.6.1 PLAIN - in simple words

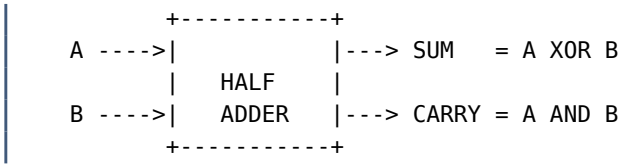
1. Binary addition has only four cases for a single column: 0+0, 0+1, 1+0 and 1+1.
2. The first three give 0, 1 and 1. The fourth gives 0 with a 1 carried into the next column.
3. So each column produces two outputs: a sum bit and a carry bit.
4. The sum bit is 1 when exactly one input is 1. That is XOR.
5. The carry bit is 1 when both inputs are 1. That is AND.
6. A circuit with two inputs producing those two outputs is called a half adder.
7. It is called half because it cannot accept a carry coming in from the column to its right.
8. Add that third input and you have a full adder. A full adder takes three bits in and produces a sum bit and a carry bit out.
9. Chain eight full adders together, feeding each carry-out into the next carry-in, and you can add two 8-bit numbers.

5.6.2 PLAIN - a picture in your head

1. Think of a line of eight people, each holding two coins and passing notes to the left.
2. Each person adds their two coins plus any note passed to them from the right.
3. If the total is 2 or 3, they pass a note left saying "carry one", and keep the remainder.
4. The person on the far right starts with no incoming note.
5. Nobody can finish until the person on their right has finished, because they are waiting for the note.
6. So the total time is not the time for one person. It is the time for the note to travel all the way from the right end to the left end.
7. That waiting chain is the single reason addition is not instant, and it is what carry-lookahead exists to fix.
8. Where this comparison breaks: the people do not really take turns. All eight full adders are working continuously and all the time, on whatever values currently sit on their wires.
9. The values just happen to be wrong until the carry has finished travelling. The circuit does not know it is wrong. It simply settles.

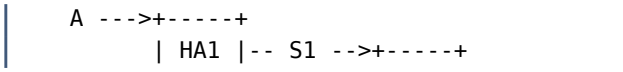
5.6.3 PLAIN - a worked example

1. The half adder.



A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2. The full adder, built from two half adders and one OR gate.



```
   B --->+-----+         | HA2 |---> SUM
         |     |         |     |
         C1    Cin -->+-----+
         |              |
         |              C2
         |              |
         +---> [ OR ] <--+
                 |
                 v
                Cout
```

3. The first half adder adds A and B. The second adds that partial sum to the incoming carry.
4. A carry can be produced by either half adder, but never by both, so a plain OR is enough to combine them.
5. Here is the full adder truth table, split so no table is wider than four columns.

A	B	Cin	SUM
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

6. In equations: SUM = A XOR B XOR Cin, and Cout = (A AND B) OR (Cin AND (A XOR B)).
7. Read the Cout column again: it is 1 whenever two or three of the inputs are
    1. Cout is a majority vote of three bits.
8. Now the 8-bit ripple-carry adder. Eight full adders in a line.

```
Cin=0 -> [FA0] -c1-> [FA1] -c2-> [FA2] -c3-> [FA3] -c4->
      -> [FA4] -c5-> [FA5] -c6-> [FA6] -c7-> [FA7] -> Cout

Each FAn also takes bit n of A and bit n of B, and
produces bit n of the sum.
```

9. Worked sum. Add 109 and 46, which are 0110 1101 and 0010 1110 in binary.
10. Bit 0 is the rightmost. Here is the full carry trace.

```
bit  A  B  Cin   SUM  Cout
 0   1  0   0     1     0
```

```
1   0  1   0     1     0
2   1  1   0     0     1
3   1  1   1     1     1
4   0  0   1     1     0
5   1  1   0     0     1
6   1  0   1     0     1
7   0  0   1     1     0
```

11. Reading the SUM column from bit 7 down to bit 0 gives 1001 1011, which is 155. And 109 plus 46 is 155. Correct.
12. Watch bit 2 and bit 3. Bit 2 generates a carry, which bit 3 receives, and bit 3 passes another one on. That is a carry rippling.
13. The worst case for rippling is 0111 1111 plus 0000 0001, which is 127 plus
    1. The carry born at bit 0 must travel through every one of the eight stages before the answer is right.

5.6.4 PLAIN - what is really happening inside

1. Every full adder in the chain starts computing the moment its inputs change. None of them waits for permission.
2. But seven of the eight are computing with a carry-in that has not settled yet, so seven of them are producing a wrong answer at first.
3. As the correct carry arrives at each stage in turn, that stage recomputes and its output may flip. The flipping travels left like a wave.
4. Those intermediate wrong values are called glitches. They are real voltage changes, they burn real power, and they are harmless as long as nothing reads the output too early.
5. The time to settle is roughly the delay of one stage's carry path multiplied by the number of stages.
6. Double the width from 8 bits to 16 bits and you roughly double the delay. That is a terrible scaling law for a 64-bit machine.
7. The fix is to stop waiting. Notice that a stage does not need the actual carry to know how it will behave.
8. A stage generates a carry if both its bits are 1, no matter what comes in. Call that G.
9. A stage propagates a carry if exactly one of its bits is 1, meaning it will pass on whatever comes in. Call that P.
10. G and P can be computed for all 64 bits simultaneously, because they depend only on A and B and not on any carry.
11. Then the carry into any position is a single wide formula built from the G and P values below it, computed in a fixed small number of gate levels.
12. That is carry-lookahead. It trades a lot of extra gates for a delay that grows like the logarithm of the width instead of linearly.

5.6.5 TECHNICAL - the engineer's version

1. The generate and propagate signals are $G(i) = A(i)\ .\ B(i)$ and $P(i) = A(i)$ XOR $B(i)$. Some texts use $P(i) = A(i) + B(i)$, which is equivalent for carry purposes but not for the sum.
2. The recurrence is $C(i+1) = G(i) + P(i)\ .\ C(i)$.
3. Expanding for four bits gives the classic flat lookahead equation:

```
C1 = G0 + P0.C0
C2 = G1 + P1.G0 + P1.P0.C0
C3 = G2 + P2.G1 + P2.P1.G0 + P2.P1.P0.C0
C4 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.G0 + P3.P2.P1.P0.C0
```

4. Every one of those is two gate levels after P and G are available, so a 4-bit block produces its carry-out in about three levels rather than eight.

5. Flat lookahead does not scale past about four bits, because the AND terms grow in both count and fan-in. Real designs build a tree of 4-bit blocks with block-level group generate and group propagate signals.
6. Gerald B. Rosenberger of IBM filed for a binary carry-lookahead adder in 1957 and received United States patent 2,966,305 in 1960. Konrad Zuse is believed to have used carry anticipation in the Z1 in the 1930s.
7. Weinberger and Smith published "A One-Microsecond Adder Using One-Megacycle Circuitry" in the IRE Transactions on Electronic Computers in 1956, which is the standard reference for the lookahead equations above.
8. Delay comparison for a 16-bit adder, counting gate levels:

Adder type	Gate levels	Area cost
Ripple carry	about 31	lowest
4-bit block lookahead	about 8	moderate
Kogge-Stone prefix	about 9	highest

9. Prefix adders treat carry generation as a parallel prefix problem. Kogge-Stone, from Peter Kogge and Harold Stone in 1973, has minimum depth and maximum wiring. Brent-Kung, from 1982, trades depth for far less wiring.
10. Sklansky and Han-Carlson sit between those two. Experts disagree on the best default: some favour Kogge-Stone for depth, others argue its wiring load makes Han-Carlson faster in practice at modern process nodes.
11. The 7483 and 74283 are 4-bit full adders with internal fast carry. The 74182 is a lookahead carry generator designed to sit above four 74181 ALU slices.
12. In Verilog you simply write the plus sign and let the synthesis tool choose the architecture, guided by your timing constraint:

```
module add8 (input  [7:0] a, b,
             input        cin,
             output [7:0] sum,
             output       cout);
  assign {cout, sum} = a + b + cin;
endmodule
```

5.6.6 WORDS - remember these

1. Half adder — adds two bits, no carry in — a circuit producing S = A XOR B and C = A AND B.
2. Full adder — adds three bits — a circuit producing sum and carry from A, B and a carry-in.
3. Ripple carry — each stage waits for the last — a linear-delay adder where carry-out chains into the next carry-in.
4. Carry generate — this column makes a carry regardless — G = A AND B.
5. Carry propagate — this column passes a carry through — P = A XOR B.
6. Carry-lookahead — work out all carries at once — a logarithmic-depth carry network computed from G and P without waiting.
7. Glitch — a wrong value on the way to the right one — a transient logic hazard caused by unequal path delays.

5.7 Subtracting without a subtractor

5.7.1 PLAIN - in simple words

1. A computer does not contain a subtractor. It contains an adder, and it cheats.
2. To compute A minus B, it computes A plus the negative of B.

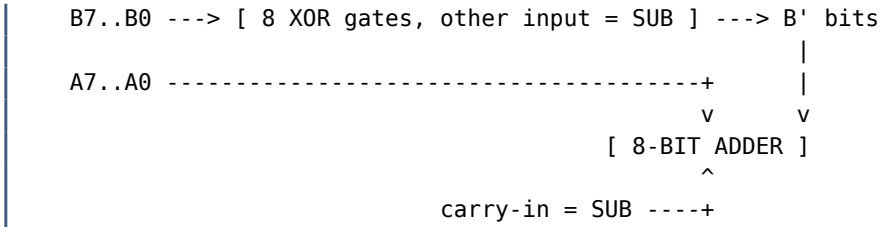
3. So the only question is how to represent a negative number in bits.
4. The scheme every modern machine uses is called two's complement.
5. The rule to negate a number is: flip every bit, then add 1.
6. Take 5 in 8 bits: 0000 0101. Flip every bit: 1111 1010. Add 1: 1111 1011. That is minus 5.
7. To check it, add 5 and minus 5. 0000 0101 plus 1111 1011 is 1 0000 0000. The ninth bit falls off the end and you are left with zero. Correct.
8. The top bit acts as a sign flag. If it is 1, the number is negative.
9. In 8 bits the range is minus 128 to plus 127. There is one more negative value than positive, because zero uses up one of the non-negative slots.

5.7.2 PLAIN - a picture in your head

1. Think of a car odometer with only three digits, so it can show 000 to 999.
2. Wind it back one from 000 and it shows 999. So on that dial, 999 behaves exactly like minus one.
3. Add 5 to 999 and you get 1004, but the leading 1 cannot be shown, so the dial reads 004. And 5 minus 1 is 4. It worked.
4. Two's complement is that same wrap-around idea with two digits per position instead of ten.
5. The bits that fall off the left end are simply discarded, and the arithmetic still comes out right.
6. Where this comparison breaks: on an odometer, everything above 500 is not automatically treated as negative. It is just a big number.
7. In two's complement, the machine decides which half is negative by the top bit, and it is the instruction you choose that decides whether the same bits mean 200 or minus 56.

5.7.3 PLAIN - a worked example

1. Compute 45 minus 67 in 8 bits.
2. 45 is 0010 1101. 67 is 0100 0011.
3. Flip every bit of 67: 1011 1100.
4. Feed that to the adder together with 45, and set the carry-in to 1. The carry-in supplies the "add one" for free, with no extra gate.
5. 0010 1101 plus 1011 1100 plus 1 gives 1110 1010, with a carry-out of 0.
6. Read 1110 1010 as a signed value. The top bit is 1, so it is negative.
7. To read it, flip and add one: 0001 0101 plus 1 is 0001 0110, which is 22. So the answer is minus 22.
8. And 45 minus 67 is minus 22. Correct.
9. Here is the whole trick in one diagram. One control wire, called SUB, does everything.



10. When SUB is 0, XOR with 0 leaves every B bit unchanged and the carry-in is
    0. The circuit computes A plus B.
11. When SUB is 1, XOR with 1 flips every B bit and the carry-in is 1. The circuit computes A minus B.
12. Cost of adding subtraction to an adder: eight XOR gates and one wire. That is the whole subtractor.

5.7.4 PLAIN - what is really happening inside

1. Two's complement works because of arithmetic modulo 256 for 8 bits, or modulo 2 to the power 64 for 64 bits.
2. Flipping every bit of B gives 255 minus B, because the all-ones pattern is 255 and flipping is the same as subtracting from all ones.

3. Adding 1 gives 256 minus B.
4. So A plus the flipped B plus 1 equals A plus 256 minus B.
5. The 256 is exactly one more than the largest 8-bit value, so it appears only in the ninth bit position, which the hardware drops.
6. What is left on the wires is A minus B. The mathematics is exact, not an approximation.
7. Now, the hardware must also spot when the answer does not fit. There are two separate failure signals, and they are not the same thing.
8. Carry-out means the result was too big for the width, treating the numbers as unsigned. It is the ninth bit that fell off.
9. Overflow means the result was too big for the width, treating the numbers as signed. It is detected differently.
10. The signed overflow rule is short: overflow happens when the carry into the top bit differs from the carry out of the top bit.
11. In one XOR gate: V = C(n) XOR C(n-1).
12. A second way to say the same thing: overflow happens if two numbers of the same sign are added and the answer has the opposite sign.
13. Adding a positive and a negative number can never overflow, because the answer is always between the two inputs.

5.7.5 TECHNICAL - the engineer's version

1. Two's complement is the representation mandated by essentially every current general-purpose instruction set: x86-64, ARM AArch64, RISC-V and POWER.
2. Since the C23 standard, published in 2024, signed integers in C are required to use two's complement. Before that, C permitted sign-magnitude and one's complement too, though no mainstream compiler used them.
3. The asymmetry is real and it bites. In 8 bits, negating minus 128 gives back minus 128, because plus 128 does not exist. The same holds at every width.
4. In C, computing 0 minus INT_MIN, or INT_MIN divided by minus 1, is undefined behaviour. On x86-64 the IDIV instruction raises a divide-error exception, the same one as division by zero.
5. Worked signed overflow example: 100 plus 50 in 8 bits. 0110 0100 plus 0011 0010 gives 1001 0110.
6. As unsigned that is 150 and there was no carry-out, so the unsigned carry flag is 0.
7. As signed it is minus 106, which is wrong. The carry into bit 7 was 1 and the carry out of bit 7 was 0, so V = 1 XOR 0 = 1. Overflow is flagged.
8. Flag conventions differ, and this is a genuine trap when porting code.

Machine	After SUB, carry means
x86 / x86-64	1 means a borrow happened
ARM AArch64	1 means no borrow
RISC-V	no flags at all

9. This is a convention, not a standard. On ARM the C flag is the literal carry out of A plus NOT B plus 1. On x86 it is inverted before being stored.
10. RISC-V has no condition-code register by design. Comparison is done with instructions such as SLT and SLTU that write a 0 or 1 into a normal register, and branches compare two registers directly.
11. Tools to observe this: in GDB, "info registers eflags" on x86-64 shows CF, ZF, SF and OF. In LLDB, "register read cpsr" shows N, Z, C and V on ARM.
12. Compilers exploit signed overflow being undefined behaviour in C and C++ to optimize loops. Build with -fwrapv to force wrapping, or use -fsanitize=signed-integer-overflow to trap it at runtime.

5.7.6 WORDS - remember these

1. Two's complement — flip the bits and add one — the radix complement representation where the top bit has weight minus 2 to the power n-1.
2. Sign bit — the top bit says positive or negative — bit n-1 of a two's complement value.
3. Carry flag — a bit fell off the end, unsigned — the carry-out of the most significant adder stage, inverted after subtraction on x86.
4. Overflow flag — the signed answer is wrong — set when the carry into the sign bit differs from the carry out of it.
5. Modular arithmetic — numbers wrapping round a dial — arithmetic modulo 2 to the power n, which is what fixed-width binary hardware performs.
6. Undefined behaviour — the language makes no promise — a construct for which the C and C++ standards impose no requirements, permitting any result.

5.8 More building blocks

5.8.1 PLAIN - in simple words

1. Beyond gates and adders there is a small set of standard parts. Every CPU is made mostly of copies of these.
2. A multiplexer is a chooser. It has many data inputs, a few select inputs, and one output. The select value picks which input reaches the output.
3. A demultiplexer is the reverse. One data input, many outputs, and the select value picks which output receives it.
4. A decoder turns a small binary number into one hot wire. Three input bits become eight output wires, exactly one of which is 1.
5. An encoder is the reverse of a decoder. One hot wire in, a binary number out.
6. A priority encoder is an encoder that copes when several inputs are 1 at once. It reports the highest-numbered one and ignores the rest.
7. A comparator takes two numbers and says whether they are equal, or which one is bigger.
8. A barrel shifter moves all the bits of a word left or right by any amount, in one step, with no looping.
9. A parity generator counts whether the number of 1 bits is odd or even. It is one XOR tree.

5.8.2 PLAIN - a picture in your head

1. Think of a railway station. A multiplexer is the points where many tracks merge into one platform, and a lever chooses which train comes through.
2. A demultiplexer is the points at the far end, where one track fans out to many platforms.
3. A decoder is the departure board lighting up exactly one platform number.
4. A priority encoder is the station controller when three trains all signal at once: the express goes first and the others wait.
5. A barrel shifter is the whole train being moved sideways onto a parallel track, all carriages at once, rather than shunted one carriage at a time.
6. Where this comparison breaks: trains physically travel and take time proportional to distance. In a multiplexer no data moves through the unused paths at all. The unselected inputs are simply blocked.
7. Also, a real multiplexer does not queue. If two inputs change, both are read continuously. Only one is passed on.

5.8.3 PLAIN - a worked example

1. A worked 3-to-8 decoder. Three inputs A2, A1, A0 and eight outputs Y0 to Y7.
2. The rule is: Y(n) is 1 when the input bits spell the number n in binary, and 0 otherwise. Exactly one output is ever 1. That is called one-hot.

A2 A1 A0	Value	Output that is 1
0 0 0	0	Y0
0 0 1	1	Y1
0 1 0	2	Y2
0 1 1	3	Y3
1 0 0	4	Y4
1 0 1	5	Y5
1 1 0	6	Y6
1 1 1	7	Y7

3. Each output is a single 3-input AND gate over the inputs, with inverters where a 0 is required.

```
Y0 = A2' . A1' . A0'
Y1 = A2' . A1' . A0
Y2 = A2' . A1  . A0'
Y3 = A2' . A1  . A0
Y4 = A2  . A1' . A0'
Y5 = A2  . A1' . A0
Y6 = A2  . A1  . A0'
Y7 = A2  . A1  . A0
```

4. Cost: eight 3-input AND gates plus three inverters. Delay: two gate levels.
5. Check A2=1, A1=0, A0=1. Y5 needs A2 true, A1 false, A0 true. All hold, so Y5 = 1. Every other line has at least one term false, so all others are 0.
6. Now a 4-to-1 multiplexer with select bits S1 and S0.

```
OUT = S1'.S0'.D0 + S1'.S0.D1 + S1.S0'.D2 + S1.S0.D3
```

7. Notice the four product terms are exactly the four decoder outputs. A multiplexer is a decoder whose one-hot lines gate the data through.
8. A 2-to-1 multiplexer is the smallest useful case: OUT = S′.A + S.B. It is the atom of all control logic in a CPU.

5.8.4 PLAIN - what is really happening inside

1. Where each part sits in a real CPU.
2. Multiplexers sit everywhere. In front of every register write port, choosing which value gets written.
3. In the forwarding paths of a pipeline, choosing whether an operand comes from the register file or straight from a later stage.
4. And at the program counter, choosing between the next sequential address and a branch target.
5. Decoders sit in the instruction decoder, turning an opcode field into control lines, and in the memory system, turning an address into one row select line inside a RAM array.
6. Demultiplexers sit on write paths, steering one result to one of many destination registers.
7. Priority encoders sit in the interrupt controller, choosing which of many simultaneous interrupt requests to serve first.
8. They also sit in a floating-point unit, finding the position of the highest set bit so a result can be normalized.
9. Comparators sit in branch units, in cache tag matching, and in address range checks in a memory protection unit.
10. A barrel shifter sits inside the ALU, so that shift and rotate instructions take one cycle instead of one cycle per bit position.
11. Parity generators sit on memory buses and on cache arrays, producing one extra check bit per byte or per word.

12. The honest version: a modern core has no part labelled "priority encoder". It has a synthesized cloud of gates produced from a description of that behaviour. The names are how engineers think, not how silicon is drawn.

5.8.5 TECHNICAL - the engineer's version

1. Sizes and costs of the standard blocks:

Block	Inputs to outputs	Delay levels
2-to-1 mux	2 data, 1 select	2
3-to-8 decoder	3 to 8	2
8-to-3 priority encoder	8 to 3 plus valid	3 to 4
32-bit barrel shifter	32 plus 5 select	5

2. A barrel shifter is built as a chain of 2-to-1 multiplexer stages, one per bit of the shift amount. A 32-bit shifter needs five stages, shifting by 16, 8, 4, 2 and 1.
3. That gives logarithmic depth: shifting a 64-bit word by any amount takes six multiplexer stages, not 63 steps.
4. A funnel shifter is a barrel shifter fed by two concatenated words. It implements rotate, and on x86 the SHLD and SHRD double-precision shift instructions.
5. Classic discrete parts, all from the 7400 family:

Part	Function
74138	3-to-8 decoder / demultiplexer
74151	8-to-1 multiplexer
74148	8-to-3 priority encoder
7485	4-bit magnitude comparator

6. The 74280 is a 9-bit odd and even parity generator and checker. Parity over n bits is an XOR tree of depth log base 2 of n, so 64-bit parity is six levels deep.
7. x86-64 has a POPCNT instruction that counts set bits, and a parity flag in EFLAGS covering only the low 8 bits of a result. That 8-bit limit is a hangover from the 8086 of 1978.
8. An equality comparator is an XNOR per bit followed by a wide AND. A magnitude comparator is usually a subtraction whose sign and zero flags are read, which is why CMP on x86 is SUB with the result discarded.
9. One-hot encoding is standard for finite state machines on FPGAs, because flip-flops are plentiful and the next-state logic becomes shallow. On ASICs, binary encoding is usually preferred to save flip-flops. This is a technology-dependent trade-off, not a rule.

5.8.6 WORDS - remember these

1. Multiplexer — a chooser — a combinational selector routing one of 2 to the power n data inputs to a single output under n select bits.
2. Demultiplexer — a router — the inverse of a multiplexer, steering one input to one of many outputs.
3. Decoder — small number in, one hot wire out — an n-to-2-to-the-n one-hot converter.
4. Priority encoder — reports the most important active input — an encoder that resolves multiple simultaneous requests by fixed rank, usually with a valid output.
5. Barrel shifter — shifts any distance in one step — a logarithmic-depth multiplexer network performing arbitrary shifts and rotates.
6. One-hot — exactly one wire is 1 — an encoding using one signal per state or value.
7. Parity — odd or even count of 1 bits — the XOR reduction of a word, used as a single-bit error detection code.

5.9 The ALU

5.9.1 PLAIN - in simple words

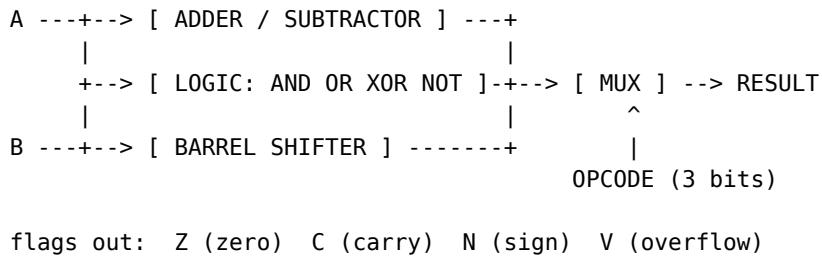
1. An ALU is the arithmetic and logic unit. It is the part of a CPU that actually computes things.
2. It is simpler than people expect. You build every operation you want, all in parallel, all at once.
3. Then you put a multiplexer at the end and use a control code to pick which answer to keep.
4. So the adder is always adding. The AND unit is always ANDing. The shifter is always shifting. Only one result is chosen.
5. That sounds wasteful, and it is. But it is fast, because no operation waits for any other to finish.
6. The control code that selects the operation comes from the instruction being executed. It is called the ALU control, or the opcode.
7. Besides the result, the ALU also produces a few single-bit facts about that result. Those are called flags or condition codes.
8. The four classic flags are zero, carry, sign and overflow.

5.9.2 PLAIN - a picture in your head

1. Think of a kitchen where four cooks each prepare a different dish from the same two ingredients, all at the same time.
2. When the order ticket arrives it does not say “start cooking”. It says “serve dish number three”.
3. All four dishes were already made. Three are thrown away.
4. Serving is instant because the work was done speculatively, in parallel.
5. Where this comparison breaks: throwing away three meals costs real food, and in an ALU the unused operations cost real energy every cycle.
6. That is why modern designs add clock gating and operand isolation, which hold the inputs of unused units steady so those gates never switch and never burn power.
7. So the picture is right about the timing and slightly wrong about the cost.

5.9.3 PLAIN - a worked example

1. Here is the shape of a small ALU.



2. A control table for an 8-operation ALU with a 3-bit opcode.

Opcode	Name	Result
000	ADD	A + B
001	SUB	A − B
010	AND	A AND B
011	OR	A OR B
100	XOR	A XOR B
101	NOT	NOT A
110	SHL	A shifted left by B
111	SHR	A shifted right by B

3. Note that opcode bit 0 doubles as the SUB control wire from section 5.7, feeding the eight XOR gates and the adder carry-in. One bit, two jobs.
4. Now the four flags, which are computed from the result and from the adder.

Flag	Set when
Z (zero)	every bit of the result is 0
C (carry)	carry out of the top adder stage
N (sign)	the top bit of the result is 1
V (overflow)	carry into top differs from out

5. Z is a wide NOR gate over all result bits. For 8 bits it is one NOR of eight inputs, or a tree of smaller NORs.
6. N is a wire. It is literally bit 7 of the result, with no gate at all.
7. C comes straight from the adder's carry-out.
8. V is one XOR gate: $V = C(8)\ \text{XOR}\ C(7)$.
9. Worked case. Opcode 001, A = 45, B = 67, as in section 5.7. Result is 1110 1010.
10. Z is 0, because the result is not all zeros. N is 1, because bit 7 is 1. C is 0, meaning a borrow occurred on x86 convention. V is 0, because minus 22 fits fine in 8 bits.

5.9.4 PLAIN - what is really happening inside

1. Not every instruction updates every flag, and the rules are precise. Getting them wrong is a classic source of bugs.
2. On x86-64, arithmetic instructions write the full set. Logic instructions write some and force others to zero.
3. Move instructions write none at all, which is why you can load a value between a compare and a conditional jump without destroying the comparison.

x86-64 instruction	Flags it writes
ADD, SUB, CMP, NEG	CF ZF SF OF AF PF
AND, OR, XOR, TEST	ZF SF PF; CF and OF forced 0
INC, DEC	ZF SF OF AF PF; CF untouched
MOV, LEA, PUSH, POP	none

4. INC and DEC deliberately leave the carry flag alone so that they can be used inside a multi-word addition loop without destroying the running carry.
5. SHL and SHR put the last bit shifted out into the carry flag, and set OF only when the shift count is exactly 1.
6. A shift by zero on x86 leaves all flags completely unchanged, which is a genuine special case in the specification and a well-known trap.
7. The order of events in one ALU operation: operands arrive on the A and B buses, every unit computes, the multiplexer selects, the flags are derived from that result, and the destination register captures it at the next edge.
8. Nothing is sequenced inside that. It is one settling of combinational logic.

5.9.5 TECHNICAL - the engineer's version

1. The 74181 was introduced by Texas Instruments in February 1970 and was the first complete ALU on a single chip.
2. It is a 4-bit slice performing 16 arithmetic and 16 logic functions, selected by four function lines S0 to S3, a mode line M and a carry-in Cn.

3. Standard versions completed an operation in 22 nanoseconds. The 74S181 did it in 11 nanoseconds and the 74F181 in 7 nanoseconds.
4. It was used in the Data General Nova 1200, the Digital Equipment PDP-11 series, the Xerox Alto and the VAX-11/780.
5. Four 74181 slices plus one 74182 lookahead carry generator gave a 16-bit ALU. Sixteen 74S181 slices plus five 74S182 parts gave 64 bits.
6. Modern cores have several ALUs. Intel Skylake, released in August 2015, has four integer execution ports, p0, p1, p5 and p6, each able to retire a simple ALU operation every cycle.
7. That is why the reciprocal throughput of ADD on Skylake is 0.25 cycles: four independent adds can issue per cycle even though each takes one cycle of latency.
8. Condition-code architectures differ sharply. x86-64 and AArch64 have a flags register. RISC-V has none. MIPS has none for integers.
9. Flags are expensive in an out-of-order core, because the flags register is a single hot dependency that every arithmetic instruction writes. Designs cope by renaming the flags register and by splitting it into independently renamed groups.
10. The x86 partial-flag problem is real. INC writes some flags and leaves CF, so a later instruction reading all flags may need to merge two sources. Intel added a merging micro-operation for it, and compilers prefer ADD 1.
11. To observe flags: in GDB use “info registers eflags”; on AArch64 read the NZCV field of CPSR or PSTATE. In a disassembly, look for SETcc, CMOVcc and Jcc instructions, which are the consumers.

5.9.6 WORDS - remember these

1. ALU — the part that computes — the arithmetic and logic unit, a combinational block selecting among parallel functional results.
2. Opcode — the code that says which operation — the instruction field driving the ALU function select multiplexer.
3. Flag — a one-bit fact about the result — a condition code recording zero, carry, sign, overflow or parity.
4. Zero flag — the answer was all zeros — the NOR reduction of the result bus.
5. Condition code register — where the flags live — EFLAGS or RFLAGS on x86, NZCV in PSTATE on AArch64.
6. Operand isolation — stop unused units switching — holding inputs of unselected functional units constant to save dynamic power.
7. Bit slice — one chip handling four bits — a design style where identical narrow ALU chips are cascaded to build a wider machine.

5.10 Multiply and divide in hardware

5.10.1 PLAIN - in simple words

1. There is no such thing as a multiply gate. Multiplication has to be built out of adding and shifting.
2. The school method works in binary and is much easier there, because every digit of the multiplier is either 0 or 1.
3. For each 1 bit in the multiplier, you add a shifted copy of the multiplicand. For each 0 bit, you add nothing.
4. So multiplying two 32-bit numbers means adding up to 32 shifted copies.
5. Doing that one step at a time takes 32 clock cycles. That is the slow way, called shift-and-add.
6. The fast way is to build all 32 shifted copies at once, in wires, and then add them all together with a tree of adders.
7. Division is worse. You cannot know a quotient digit until you have tested it.
8. So division is inherently a guess-and-check loop, and that is why it is the slowest common instruction on every processor.

5.10.2 PLAIN - a picture in your head

1. Multiplying by hand on paper is writing several rows and then adding the column totals.
2. A hardware array multiplier is that same sheet of paper, made of wire, with every row present at once.
3. Adding the rows is the slow part. Doing it one row at a time is like adding a long column top to bottom.
4. A Wallace tree instead pairs rows off in a knockout tournament: 32 rows become 22, then 15, then 10, and so on down to 2.
5. Because the tournament halves the field each round, the number of rounds grows like the logarithm of the number of rows.
6. Where this comparison breaks: in a knockout tournament the losers are eliminated. In a Wallace tree nothing is lost. Three rows are merged into two rows with the same total, using carry-save adders.
7. Nothing is thrown away. The value is preserved exactly at every level.

5.10.3 PLAIN - a worked example

1. Multiply 13 by 11 in binary. 13 is 1101 and 11 is 1011.

```
        1 1 0 1        (13)
 x      1 0 1 1        (11)
 -----------------
        1 1 0 1        bit 0 of 11 is 1, shift 0
      1 1 0 1          bit 1 of 11 is 1, shift 1
    0 0 0 0            bit 2 of 11 is 0, shift 2
  1 1 0 1              bit 3 of 11 is 1, shift 3
 -----------------
 1 0 0 0 1 1 1 1       = 143
```

2. And 13 times 11 is 143. Correct.
3. Note the product of two 4-bit numbers needs 8 bits. In general n bits times n bits needs 2n bits, which is why x86 MUL writes its result across two registers.
4. Restoring division of 13 by 3, using 4-bit values. The rule each step is: shift the remainder left, bring down the next bit, try subtracting the divisor.
5. If the subtraction goes negative, undo it and write a 0 quotient bit. If it does not, keep it and write a 1.

```
step  rem  bring  trial rem-3   keep?  quotient bit
 1     0     1      1-3 = -2      no        0
 2     1     1      3-3 =  0     yes        1
 3     0     0      0-3 = -3      no        0
 4     0     1      1-3 = -2      no        0
```

6. Quotient 0100 is 4, remainder 1. And 13 divided by 3 is 4 remainder 1. Correct.
7. The word restoring means undoing the subtraction whenever it went negative. That undo is a whole extra add, which is pure waste.
8. Non-restoring division avoids the undo by allowing negative partial remainders and correcting at the end. It costs one add-or-subtract per bit instead of up to two.

5.10.4 PLAIN - what is really happening inside

1. Booth's algorithm speeds up multiplication by noticing that a long run of 1s can be handled with one subtraction and one addition.
2. Multiplying by 0111 1111, which is 127, is the same as multiplying by 128 and then subtracting 1 copy. Two operations instead of seven.
3. Modified Booth encoding, also called radix-4 Booth, looks at three multiplier bits at a time and emits one partial product per two bits.
4. That halves the number of rows to add before the tree even starts. It is used in essentially every production multiplier.

5. The adder tree uses carry-save adders. A carry-save adder is just a full adder with its carry not connected onward, so it takes three numbers in and produces two numbers out, with no carry rippling at all.
6. Only the very last step, turning the final two numbers into one, needs a real carry-propagating adder. That is where a fast prefix adder is used.
7. Division has no such trick, because the quotient bits are not known in advance. Each digit depends on the remainder produced by the previous digit.
8. SRT division, named after Sweeney, Robertson and Tocher, speeds this up by guessing several quotient bits at a time from a small lookup table and tolerating a slightly wrong guess.
9. That lookup table is where Intel's famous 1994 bug lived.

5.10.5 TECHNICAL - the engineer's version

1. Andrew Donald Booth devised his signed multiplication technique in 1950 at Birkbeck College, London, while working on crystallography. It appeared as "A Signed Binary Multiplication Technique" in the Quarterly Journal of Mechanics and Applied Mathematics, volume 4, part 2.
2. Chris Wallace published "A suggestion for a fast multiplier" in the IEEE Transactions on Electronic Computers in February 1964. A Wallace tree has depth of order log n and places multiplication in complexity class NC1.
3. Luigi Dadda proposed the Dadda multiplier in 1965. It reduces later and uses fewer full adders than a Wallace tree for the same delay, at the cost of a slightly wider final adder.
4. An n by n array multiplier has delay of order n and area of order n squared. A Wallace or Dadda tree has delay of order log n and similar area.
5. Measured Intel Skylake figures, from the uops.info instruction database. Latency is in clock cycles; reciprocal throughput is cycles between independent issues, so smaller is better.

Instruction	Latency	Recip. throughput
ADD r64, r64	1	0.25
IMUL r64, r64	3	1.00
DIV r32	25 to 28	6.00
DIV r64	35 to 90	about 21

6. Floating-point division on the same core: DIVSS, single precision, latency up to 11 with reciprocal throughput 3.00; DIVSD, double precision, latency up to 13 with reciprocal throughput 4.00.
7. The legacy x87 FDIV on Skylake is approximately 14 to 16 cycles, per Agner Fog's instruction tables. Treat that as approximate; it is not listed in the same form by uops.info. Compilers targeting x86-64 emit DIVSD instead.
8. Integer division latency is data-dependent, which is why DIV r64 is a range and not a number. The hardware exits early when the operands are small. ADD and IMUL are fixed-latency.
9. Division has improved sharply in recent parts:

Microarchitecture	DIV r64 latency	Recip. tput
Skylake, 2015	35 to 90	about 21
Meteor Lake-P, 2023	14	10.0
AMD Zen 4, 2022	9 to 17	7.00
AMD Zen 5, 2024	10 to 18	7.00

10. This is why compilers replace division by a constant with a multiply by a magic reciprocal and a shift. Look for that pattern in any optimized build: a division by 10 becomes an IMUL and an SHR.

11. The Pentium FDIV bug: Thomas R. Nicely of Lynchburg College noticed inconsistencies on 13 June 1994, confirmed them on 19 October 1994 and reported them to Intel on 24 October 1994.
12. The cause was five cells missing from an SRT quotient-digit lookup table that should have held 1,066 populated entries. Intel announced full replacement on 20 December 1994, and took a pre-tax charge of 475 million US dollars.
13. The lesson engineers took from it: division hardware is intricate, and verifying it is now done with formal methods rather than test vectors alone.

5.10.6 WORDS - remember these

1. Shift-and-add — the school method in binary — iterative multiplication adding a shifted multiplicand per set multiplier bit.
2. Partial product — one shifted row before adding — the term contributed by one multiplier bit or Booth group.
3. Booth encoding — handle runs of 1s in one step — a signed-digit recoding of the multiplier that halves the partial product count at radix 4.
4. Carry-save adder — three numbers in, two out — a full adder array with carries kept as a separate vector rather than propagated.
5. Wallace tree — a knockout tournament of adders — a logarithmic-depth partial product reduction network.
6. Restoring division — undo the subtraction if it went negative — a one-bit per cycle division algorithm with a correction step.
7. SRT division — guess several quotient bits from a table — the radix-4 or higher division method named after Sweeney, Robertson and Tocher.

5.11 Propagation delay and the critical path

5.11.1 PLAIN - in simple words

1. A gate does not answer instantly. It takes a small, fixed amount of time to change its output after its inputs change.
2. That time is called propagation delay, or gate delay.
3. If you wire three gates in a line, the delays add up. Three gates in series take three gate delays.
4. A circuit has many paths from its inputs to its outputs. Each path has its own total delay.
5. The longest of all those paths is called the critical path.
6. The whole circuit is not finished until the critical path is finished. The fast paths finished long ago and are just waiting.
7. So the critical path sets the maximum clock speed. Nothing else does.
8. To make a circuit faster you must shorten its longest path. Speeding up anything else changes nothing at all.

5.11.2 PLAIN - a picture in your head

1. Think of a group hike where everyone must reach the hut before dinner is served.
2. Dinner starts when the last walker arrives, not when the first does.
3. Making the fastest walker faster does nothing. Dinner is still at the same time.
4. Only helping the slowest walker moves dinner earlier.
5. That slowest walker is the critical path, and finding them is the whole job of a timing analysis tool.
6. Once you speed that walker up, someone else becomes the slowest, and you repeat.
7. Where this comparison breaks: walkers are independent, but circuit paths share gates. Enlarging a gate to speed up one path slows another, because a bigger gate is a bigger load on whatever feeds it.
8. That coupling is why timing closure is an iterative struggle and not a single calculation.

5.11.3 PLAIN - a worked example

1. We will compute the maximum clock frequency of the 8-bit ripple-carry adder from section 5.6, using assumed but realistic delay figures.
2. Assumptions, which stand in for a slow discrete logic family:

Element	Delay
XOR gate	3.0 ns
AND gate	2.0 ns
OR gate	2.0 ns
Register clock-to-output	1.0 ns

3. The register setup time is 0.5 ns. Clock skew is assumed to be zero.
4. Recall the full adder equations: SUM = A XOR B XOR Cin, and Cout = (A AND B) OR (Cin AND (A XOR B)).
5. Delay from carry-in to carry-out of one stage: AND then OR, so 2.0 plus 2.0 equals 4.0 ns.
6. Delay from A or B to carry-out of the first stage: XOR then AND then OR, so 3.0 plus 2.0 plus 2.0 equals 7.0 ns.
7. Delay from carry-in to that stage's sum bit: one XOR, so 3.0 ns.
8. Now trace the critical path across all eight bits.

```
A0/B0 -> C1                    7.0 ns
C1 -> C2 -> ... -> C8          7 hops x 4.0 = 28.0 ns
total A0/B0 -> Cout            35.0 ns

alternative ending:
A0/B0 -> C7                    7.0 + 6 x 4.0 = 31.0 ns
C7 -> S7 (one XOR)             3.0 ns
total A0/B0 -> S7              34.0 ns
```

9. The critical path is the longer of the two, 35.0 ns.
10. Now the clock period. It must fit three things: the register's own delay, the logic, and the setup time of the capturing register.

```
T_min = clock-to-output + critical path + setup
T_min = 1.0 + 35.0 + 0.5 = 36.5 ns

f_max = 1 / 36.5 ns = 27.4 MHz
```

11. So this adder cannot be clocked faster than about 27 MHz, no matter how good the rest of the design is.
12. Now replace it with two 4-bit carry-lookahead blocks. Generating P and G takes one XOR, 3.0 ns. Each block's carry-out is then one AND plus one OR, 4.0 ns.
13. Carry out of the first block arrives at 3.0 plus 4.0 equals 7.0 ns. Carry into bit 7 arrives 4.0 ns later, at 11.0 ns. The final sum XOR adds 3.0 ns, giving 14.0 ns. Allow 15.0 ns with margin.

```
T_min = 1.0 + 15.0 + 0.5 = 16.5 ns
f_max = 1 / 16.5 ns = 60.6 MHz
```

14. The same arithmetic, more than twice as fast, purely by shortening the longest path.

5.11.4 PLAIN - what is really happening inside

1. Setup time is how long before the clock edge the data must already be stable at the register input.
2. Hold time is how long after the clock edge the data must stay stable.

3. Setup violations are fixed by slowing the clock down. Hold violations are not. A hold violation means a path is too fast, and slowing the clock does not help at all.
4. Hold violations are fixed by inserting delay, usually pairs of buffers, into the offending path.
5. If either is violated, the register can enter a state that is neither 0 nor 1 for an unpredictable time. That is called metastability.
6. Chapter 6 covers clocks, registers, setup, hold and metastability properly. Here you only need the one idea that the longest path sets the speed.
7. One more honesty point. Gate delay is not a single number. It depends on temperature, on supply voltage, and on random manufacturing variation.
8. So every real timing figure comes as a set of corners: slow-slow at high temperature and low voltage, fast-fast at low temperature and high voltage, and typical in between.
9. A design must pass setup checks in the slow corner and hold checks in the fast corner. Both, always.

5.11.5 TECHNICAL - the engineer's version

1. Propagation delay is measured between 50 percent supply crossings, from input transition to output transition. Rise and fall delays differ and are listed separately as tPLH and tPHL.
2. Real 2-input NAND propagation delays across the 7400 family, showing sixty years of progress:

Family	Introduced	Typical NAND delay
7400 standard TTL	1964	about 10 ns
74LS00	1971	about 9 ns
74HC00	1983	about 7 ns at 5 V
74LVC00	1993	about 3.5 ns at 3.3 V

3. The 74HC00 datasheet limit is 15 ns maximum at 6 volts with a 50 picofarad load. Datasheet maxima are guaranteed numbers; typicals are not.
4. Inside a modern chip, gate delays are in picoseconds, not nanoseconds. A fan-out-of-four inverter delay is roughly 5 to 15 picoseconds at leading process nodes, which is why cores run at several gigahertz.
5. The setup-time equation used by every static timing analysis tool is:

```
T_clk >= t_cq + t_logic_max + t_setup + t_skew_uncertainty
```

6. And the hold-time check, which does not involve the clock period at all:

```
t_cq + t_logic_min >= t_hold + t_skew
```

7. Static timing analysis, or STA, checks every path in a design against these two inequalities without simulating any vectors. Tools: Synopsys PrimeTime, Cadence Tempus, and the open-source OpenSTA.
8. Slack is the margin on a path: required arrival time minus actual arrival time. Negative slack is a violation. Worst negative slack, WNS, is the headline number a designer watches.
9. Logical effort, the design method published by Ivan Sutherland, Bob Sproull and David Harris in their 1999 book "Logical Effort", gives a systematic way to size gates along a path for minimum delay.
10. Process, voltage and temperature corners are usually enumerated as SS, TT and FF for slow, typical and fast silicon, combined with voltage and temperature extremes. A modern signoff may check dozens of corner and mode combinations.

5.11.6 WORDS - remember these

1. Propagation delay — how long a gate takes to answer — the 50 percent to 50 percent input-to-output transition time.
2. Critical path — the slowest route through the logic — the timing path with the largest delay, which bounds the clock period.

3. Slack — how much margin a path has — required arrival time minus actual arrival time, negative meaning a violation.
4. Setup time — data must be ready before the edge — the minimum stable time at a register input preceding the active clock edge.
5. Hold time — data must stay put after the edge — the minimum stable time following the active clock edge.
6. Static timing analysis — checking speed without simulating — exhaustive path-based timing verification against setup and hold constraints.
7. PVT corner — the worst case combination — a specified process, voltage and temperature condition at which timing must be met.

5.12 Why floating point maths is not exact

5.12.1 PLAIN - in simple words

1. Type 0.1 plus 0.2 into almost any programming language and you get 0.30000000000000004, not 0.3.
2. This is not a bug in the language. It is not a bug in the processor. It is the direct consequence of building numbers out of a fixed number of bits.
3. In base ten, one third cannot be written exactly. It is 0.3333 forever, and you must stop somewhere.
4. In base two, one tenth cannot be written exactly either. It is 0.0001100110011 with 0011 repeating forever.
5. The hardware has room for 53 significant bits. It stores the closest value it can and throws the rest away.
6. So the value stored for 0.1 is not 0.1. It is very slightly more than 0.1.
7. The same is true for 0.2. Add the two slightly wrong values and the errors do not cancel. They add up.
8. The result is slightly more than 0.3, and it is a different bit pattern from the closest value to 0.3. So the comparison fails.
9. Numbers like 0.5, 0.25 and 0.75 are exact, because they are sums of powers of two. Only fractions whose denominator is a power of two are exact.

5.12.2 PLAIN - a picture in your head

1. Imagine a ruler marked only in halves, quarters, eighths, sixteenths and so on, down to a very fine division.
2. Every mark on that ruler is a number the computer can hold exactly.
3. Now try to measure one tenth of the ruler's length. There is no mark there. There never will be, however fine you make the divisions.
4. So you pick the nearest mark. That is what rounding to 53 bits does.
5. Measure a tenth twice and add the two nearest marks. The two small errors both pointed the same way, so the total is a little further off.
6. Where this comparison breaks: the marks on a real ruler are evenly spaced. Floating-point marks are not.
7. Near zero the marks are extremely close together. Near very large values they are far apart. The spacing doubles every time the exponent increases by one.
8. That is what floating means. The point moves so that you always get about 16 significant decimal digits, whether the number is tiny or huge.

5.12.3 PLAIN - a worked example

1. Here are the exact values a 64-bit double actually holds. These are not approximations of the stored values. They are the stored values, written out in full decimal.

```
stored for 0.1:
  0.1000000000000000055511151231257827021181583404541015625
```

```
stored for 0.2:
  0.200000000000000011102230246251565404236316680908203125

exact sum of those two:
  0.300000000000000044408920985006261616945266723632812 5

nearest double to 0.3:
  0.299999999999999988897769753748434595763683319091796875
```

2. The sum lands on a different mark from the one 0.3 lands on. That is why the equality test is false.
3. Printed with the shortest string that round-trips, that sum shows as 0.30000000000000004. The extra digits are not noise. They are the true value.

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 == 0.3
False
>>> from decimal import Decimal
>>> Decimal(0.1)
Decimal('0.1000000000000000055511151231257827021181583404541015625')
```

4. Now the same idea in hardware terms. A double has one sign bit, 11 exponent bits and 52 stored fraction bits.
5. The 53rd significant bit is implied. For normal numbers there is always a leading 1 that does not need storing, which buys one free bit of precision.
6. 0.1 in binary is 1.100110011001100… times 2 to the power minus 4, with 1001 repeating.
7. The hardware keeps 53 bits of that and rounds the rest. Rounding up here, which is why the stored value is slightly too big.

5.12.4 PLAIN - what is really happening inside

1. Watch what the adder actually does when it is handed the two stored values.
2. Step 1, align. The two numbers have different exponents, minus 4 and minus 3. The smaller one is shifted right by one bit so the binary points line up.
3. That shift can push bits off the right-hand end straight away. They are not lost yet, because the hardware keeps a few extra bits for exactly this reason.
4. Step 2, add. The two 53-bit fractions go into an ordinary integer adder, the same kind built in section 5.6. There is nothing special about it.
5. Step 3, normalize. Shift the result so the leading bit is a 1 again, and adjust the exponent to match. That shift uses a barrel shifter and a priority encoder to find the leading 1.
6. Step 4, round. The result is now wider than 53 bits, so it must be cut down. The default rule is round to nearest, ties to even.
7. That rounding step is where the error is created. It is a deliberate, specified, deterministic operation, not an accident.
8. The three extra bits the hardware carries through the alignment are called guard, round and sticky. The sticky bit records whether anything non-zero fell off the end.
9. Without those three bits, results would be measurably worse. With them, the result is provably the correctly rounded value of the exact sum.
10. The honest version: the hardware is not sloppy. Every basic floating-point operation is required to give the exact result rounded once. The error you see is the minimum possible error for the format.

5.12.5 TECHNICAL - the engineer's version

1. The format is IEEE 754 binary64, universally called double. It was first standardized as IEEE 754-1985, revised as IEEE 754-2008 and again as IEEE 754-2019.

Field	Bits	Meaning
Sign	1	0 positive, 1 negative
Exponent	11	biased by 1023
Fraction	52	plus 1 implied bit

2. Precision is 53 bits, which is about 15.95 decimal digits. Machine epsilon for binary64 is 2 to the power minus 52, roughly 2.22 times 10 to the power minus 16.
3. The bit patterns for the worked example, as 64-bit hexadecimal:

```
0.1             = 3FB999999999999A
0.2             = 3FC999999999999A
0.3             = 3FD3333333333333
0.1 + 0.2       = 3FD3333333333334
```

4. The two results differ in the final hexadecimal digit only. That is a difference of one unit in the last place, written 1 ulp.
5. IEEE 754 requires that add, subtract, multiply, divide and square root be correctly rounded: the result must equal the exact mathematical result rounded once to the destination format.
6. Five rounding modes are defined. The default is roundTiesToEven. The others are roundTiesToAway, roundTowardPositive, roundTowardNegative and roundTowardZero.
7. Correct rounding is not required for transcendental functions such as sine and exponential. Those are library code, and different libraries give slightly different answers. That is permitted by the standard.
8. Floating-point addition is commutative but not associative. Changing the grouping changes the result, which is why compilers may not reorder floating-point sums unless you pass -ffast-math and accept the consequences.
9. Practical rules for engineers: never test floating-point values with equality for computed results. Compare against a tolerance, or scale to integers, or use a decimal type.
10. For money, use integers of the smallest unit, or a decimal library. Python's decimal module, Java's BigDecimal and the SQL NUMERIC type all exist for this reason.
11. Fused multiply-add, or FMA, computes a times b plus c with a single rounding at the end instead of two. It was added to x86 as FMA3 with Intel Haswell in 2013 and is standard on AArch64.
12. Tools to observe this: printf with "%.20f" in C, Python's decimal.Decimal constructor applied to a float, and the online notion of shortest round-trip printing implemented by repr in Python 3 and by Double.toString in Java.

5.12.6 WORDS - remember these

1. Floating point — the point moves to keep precision — a sign, exponent and significand representation of the form s times m times 2 to the power e.
2. Significand — the digits of the number — the 53-bit mantissa of binary64, with one bit implied for normal values.
3. Rounding — cutting to the bits you have — mapping an exact result to the nearest representable value under a specified rule.
4. ULP — one step on the ruler — unit in the last place, the gap between adjacent representable numbers at a given magnitude.
5. Machine epsilon — the smallest relative step — 2 to the power minus 52 for binary64, about 2.22 times 10 to the power minus 16.
6. Guard, round and sticky — the three spare bits — extra precision retained during alignment so the final rounding is correct.
7. IEEE 754 — the rule book — the standard defining binary and decimal floating-point formats, operations and rounding, current edition 2019.

5.98 Common wrong ideas

1. Wrong: a gate passes the input current through to the output. Right: a gate reads its inputs and drives a fresh output from its own power supply, which is why signals do not fade along a chain.
2. Wrong: AND and OR are the basic gates. Right: in CMOS, NAND and NOR are the cheap four-transistor primitives, and AND and OR are each one of those plus an inverter, costing six transistors.
3. Wrong: XOR is just another basic gate like AND. Right: XOR has no simple series and parallel transistor form, so it costs 8 to 12 transistors and is noticeably slower, which matters because adders are mostly XOR.
4. Wrong: a computer contains a subtractor circuit. Right: it contains an adder, plus one XOR gate per bit and one control wire, and subtraction is addition of the two's complement.
5. Wrong: the carry flag and the overflow flag mean the same thing. Right: carry means the unsigned result did not fit, overflow means the signed result did not fit, and either can be set without the other.
6. Wrong: making any gate faster makes the circuit faster. Right: only the critical path matters, so speeding up a gate that is not on the longest path changes the maximum clock frequency by exactly nothing.
7. Wrong: a Karnaugh map gives the fastest circuit. Right: it gives a small two-level expression, but a shallower circuit with more gates is often faster, and synthesis tools optimize for timing, not gate count.
8. Wrong: 0.1 plus 0.2 not equalling 0.3 is a rounding bug in the language. Right: it is the specified, correct behaviour of IEEE 754 binary64, because one tenth has no exact binary representation at any finite width.
9. Wrong: division is slow because it is complicated to write. Right: it is slow because each quotient digit depends on the remainder from the previous one, so it cannot be fully parallelized the way multiplication can.
10. Wrong: universality means NAND is the best gate for every job. Right: universality says NAND is sufficient, not optimal, and real designs pick whichever library cell is cheapest for each specific function.

5.99 Chapter summary in 20 lines

1. Boolean algebra has two values, 1 and 0, and three operations, AND, OR and NOT, published by George Boole in 1847 and 1854.
2. Claude Shannon's MIT master's thesis, submitted on 10 August 1937, showed that relay switching circuits obey exactly those rules.
3. Inside a chip, 1 and 0 are two voltage bands, and the assignment of high to 1 is a convention called positive logic.
4. A CMOS gate has a PMOS pull-up network and a complementary NMOS pull-down network, and exactly one of them conducts at a time.
5. An inverter costs 2 transistors, a 2-input NAND or NOR costs 4, and an AND or OR costs 6 because it is a NAND or NOR plus an inverter.
6. Series transistors give AND-like behaviour and parallel transistors give OR-like behaviour, and CMOS gates are naturally inverting.
7. The seven gates are AND, OR, NOT, NAND, NOR, XOR and XNOR; XOR means exactly one, XNOR means the same, NAND means not both, NOR means neither.
8. NAND alone is functionally complete, needing 1 gate for NOT, 2 for AND, 3 for OR and 4 for XOR; NOR alone is complete too.
9. Fabs favour NAND because its series stack is NMOS, and electron mobility is two to three times higher than hole mobility.
10. Any truth table becomes a sum of products, and a Karnaugh map shrinks it by grouping adjacent 1s in Gray-code order.
11. De Morgan's two laws, provable in four rows each, say that not-both equals either-missing and neither equals both-missing.

12. A half adder is one XOR and one AND; a full adder is two half adders and an OR, giving sum and a majority-vote carry.
13. An 8-bit ripple-carry adder is eight full adders in a chain, and its speed is set by the carry travelling from bit 0 to bit 7.
14. Carry-lookahead computes generate and propagate for every bit at once, turning linear carry delay into logarithmic delay.
15. Subtraction is addition of the two's complement: invert every bit of B and set the adder carry-in to 1, costing eight XOR gates.
16. Carry means the unsigned answer overflowed; signed overflow is the XOR of the carry into and out of the top bit.
17. Multiplexers, decoders, priority encoders, comparators, barrel shifters and parity trees are the standard blocks a CPU is assembled from.
18. An ALU computes every operation in parallel and picks one with a multiplexer driven by the opcode, then derives the Z, C, N and V flags.
19. Multiplication is shift-and-add accelerated by Booth encoding and a Wallace tree; division is a guess-and-check loop and is the slowest common instruction, at 35 to 90 cycles for DIV r64 on Intel Skylake.
20. The critical path sets the clock, and rounding a 53-bit significand is why 0.1 plus 0.2 gives 0.30000000000000004 on every IEEE 754 machine.

6

CHAPTER

KB

How a Machine Remembers — Latches, Clocks and Memory Cells

6.0 What this chapter gives you

1. You will explain why a plain logic gate cannot remember anything.
2. You will draw an SR latch and say what all four input pairs do.
3. You will state the difference between a latch and a flip-flop in one line.
4. You will know setup time and hold time in picoseconds, and why breaking them causes metastability.
5. You will say why metastability can be made rare but never removed.
6. You will explain a clock tree, clock skew, jitter and clock gating.
7. You will build registers, shift registers and counters from flip-flops.
8. You will write a state machine as a table and a diagram, and see that a CPU control unit is one of these.
9. You will know how SRAM, DRAM and flash cells hold a bit, and why they cost such different amounts.
10. You will work out how many address lines a given capacity needs.

6.1 The problem: a gate forgets the moment its input changes

6.1.1 PLAIN - in simple words

1. A logic gate looks at its inputs and drives an output, all the time.
2. Change an input and the output changes a moment later. Always.
3. So a gate has no past. It knows only what is on its wires right now.
4. A machine of gates alone therefore cannot remember.
5. The fix is short to say: wire the output back into the input.
6. Now the circuit tells itself what it was. That loop is **feedback**.
7. A good loop has two comfortable resting positions, one for 0 and one for 1. We call that **bi-stable**.

6.1.2 PLAIN - a picture in your head

1. Think of a light switch on a wall.
2. Your finger is the input. The switch position is the output.
3. Push it up, take your finger away, and it stays up.
4. A spring inside pushes it further whichever way it has started to move.
5. That spring is feedback. It will not sit halfway. Two positions only.

Where this comparison breaks:

1. A wall switch holds its position with no power. Our circuit needs power every second or it forgets.
2. A real switch can balance on the edge for a moment. So can a real memory circuit, and that matters a great deal (section 6.4).

6.1.3 PLAIN - a worked example

1. Take two NOT gates, each giving the opposite of its input.
2. Wire the first into the second, and the second back into the first.

```
      +--------------------------+
      |                          |
      v                          |
A --[ NOT ]--> B --[ NOT ]--> A -+

If A = 0 then B = 1 then A = 0.   Consistent. It holds.
If A = 1 then B = 0 then A = 1.   Consistent. It holds.
```

3. Suppose A is 0. The first gate makes B equal 1, and the second makes A equal 0 again. Nothing moves.
4. Suppose A is 1. Then B is 0, which makes A equal 1. Again nothing moves.
5. Both answers are self-consistent, so that is one stored bit.
6. The bad news: no wire is left to say which bit to store. Section 6.2 fixes that.

6.1.4 PLAIN - what is really happening inside

1. Each NOT gate is a pair of transistors acting as a fast switch pair.
2. Input low connects the output to the supply. Input high connects it to ground.
3. In the loop, one gate always pulls up while the other pulls down.
4. A tiny voltage wobble from a nearby wire is amplified and pushed back round the loop, so the state survives noise.
5. There is a third position, in the middle, where both gates are half on.
6. In theory the loop can balance there. In practice it falls off in well under a nanosecond, because noise always exists.
7. The honest version: while settling, the loop is not digital at all. It is an analogue amplifier with its output tied to its input.

6.1.5 TECHNICAL - the engineer's version

1. Two cross-coupled inverters form a bistable with two stable DC operating points and one metastable point between them.
2. Plotting one inverter's voltage transfer curve against the other's mirrored curve gives the **butterfly plot**, whose curves cross three times.
3. The largest square fitting inside a lobe is the **static noise margin**. For a 6T SRAM cell in a modern process, read noise margin is typically 100 to 200 mV at nominal supply, and collapses at low voltage.
4. Stability needs small-signal loop gain above 1 at the crossing point.
5. Recovery from the metastable point is exponential, with a time constant tau set by the loop gain-bandwidth product. Section 6.4 uses this.
6. The first electronic bistable was the **Eccles-Jordan trigger circuit**, built by William Eccles and Frank Jordan in 1918 from two vacuum triodes, published in the Radio Review in 1919, British patent 148,582.

6.1.6 WORDS - remember these

1. Feedback — sending an output back to an input — a closed signal loop whose gain can exceed unity.
2. Bi-stable — two comfortable resting states — a circuit with two stable DC operating points.
3. Metastable point — the wobbly middle — the unstable equilibrium between the two stable points.
4. Noise margin — how much interference a bit survives — static noise margin in millivolts, from the butterfly plot.
5. Volatile — forgets without power — state held only by current from the supply rail.

6.2 The SR latch from cross-coupled NOR gates

6.2.1 PLAIN - in simple words

1. Our two-inverter loop could hold a bit but we could not write one.
2. A NOR gate outputs 1 only when all inputs are 0. Any input at 1 forces the output to 0.
3. Take two NOR gates and feed each one's output into the other's input.
4. The spare inputs are our controls: S for set, R for reset.
5. The outputs are Q and Q-bar, normally opposites.
6. Raise S and Q becomes 1. Raise R and Q becomes 0.
7. Put both back to 0 and the latch keeps what it had. That is remembering.
8. Raise both at once and it misbehaves. That case is called forbidden.

6.2.2 PLAIN - a picture in your head

1. Two people, each holding the other's arm down. Whoever pushes down wins.
2. S is a referee who taps the first to make them stand. R taps the second.
3. With neither referee touching anyone, the pair stays as it was.
4. If both tap at once, both try to stand and nobody holds anybody down. The arrangement is broken while that lasts.
5. If both let go at the same instant, both lunge, and luck decides.

Where this comparison breaks:

1. People are slow and roughly equal. Gates differ by picoseconds, so the race is settled by manufacturing variation, not fairness.
2. People cannot stand half up for a measurable time. Gates can, and we call that metastability.

6.2.3 PLAIN - a worked example

1. All four cases of the NOR-based SR latch.

S	R	Q next	Name
0	0	no change	Hold
0	1	0	Reset
1	0	1	Set
1	1	both outputs 0	Forbidden

2. Case S=1, R=0. S forces one gate's output to 0. The other then sees two zeroes and outputs 1. Q is 1, Q-bar is 0.
3. Case S=0, R=1 is the mirror image. Q becomes 0.
4. Case S=0, R=0 from Q=1. Each gate sees the other's output holding it, so nothing moves.
5. Case S=1, R=1. Both gates have an input at 1, so both outputs go to 0. Q and Q-bar are now both 0, contradicting their names.
6. Drop both to 0 together and both gates try to output 1, then each sees the other's 1 and tries to go back. The result is unpredictable.

6.2.4 PLAIN - what is really happening inside

1. Hold works because each output quietly holds the other gate's input.
2. Set works because a 1 on S overpowers the feedback. A NOR gate with any input at 1 must output 0, whatever the feedback says.
3. The other gate then flips, reinforcing the first. A write is a short fight that the input wins, after which the loop locks in the winner.

4. The forbidden state is not magic. The outputs are simply no longer opposites, so anything reading Q-bar as "not Q" is being lied to.
5. The danger is leaving it. Both gates start identical and race, and the loser is decided by picoseconds.
6. The NAND version flips everything. A NAND outputs 0 only when all inputs are 1, so the controls are active low, written S-bar and R-bar.
7. For the NAND latch, S-bar=0 sets, R-bar=0 resets, both at 1 holds, and both at 0 is forbidden, driving both outputs to 1.

6.2.5 TECHNICAL - the engineer's version

1. The NOR form is active-high on S and R. The NAND form is active-low.
2. The NAND form dominates in practice because a CMOS NAND is smaller and faster than a NOR of equal drive. That is a CMOS fact, not a logic rule.
3. Four NAND gates give the gated SR latch, and one more inverter gives the D latch of section 6.3.
4. The forbidden combination is precisely a **non-complementary output condition**, and the failure on release is a **race condition** decided by device mismatch and edge arrival order.
5. If the release edges are more than one loop delay apart, the last released loses, deterministically. Closer than that, the latch can go metastable.
6. The data sheet for the TTL part SN74LS279, a quad latch with active-low inputs, states it plainly: both inputs low is not a valid state.
7. In Verilog an `always @*` block with an incomplete `if` and no `else` infers a latch by accident. Synthesis warns about it, and the warning is almost always a real bug.

6.2.6 WORDS - remember these

1. SR latch — the simplest two-gate memory — a bistable set-reset element from cross-coupled NOR or NAND gates.
2. Set — store a 1 — drive Q high and let feedback hold it.
3. Hold — keep what is there — both controls inactive, feedback loop closed.
4. Forbidden state — the input pair you must not use — the non-complementary output condition, S=R=1 in the NOR form.
5. Race condition — two signals fighting to arrive first — an outcome decided by arrival time, not by logic values.
6. Active-low — does its job at 0 — asserted at logic low, written with a bar or a leading n, as in nRESET.

6.3 The gated D latch

6.3.1 PLAIN - in simple words

1. The SR latch has two data inputs when we want one, and accepts changes at any moment.
2. The gated D latch fixes both. One data input, D. One control, enable.
3. When enable is 1, the latch copies D straight through to Q, following every move D makes.
4. When enable is 0, the latch stops looking and holds what it had.
5. Enable is a window. Open, data flows. Closed, the last value is trapped.
6. There is no forbidden input, because S and R now come from one D and can never both be 1.
7. The awkward part is the open window. While enabled the latch is see-through. That is **transparency**.

6.3.2 PLAIN - a picture in your head

1. A photocopier with a page sliding underneath it.
2. Enable is the shutter. While open, the copy keeps being redrawn as the page moves.
3. Close the shutter and the copy freezes at whatever was underneath then.
4. For a clean copy the page must be still while the shutter is open and for a moment after it closes.

Where this comparison breaks:

1. A smeared photocopy is obviously wrong. A latch that catches a value mid-change gives an ordinary-looking 0 or 1 that happens to be the wrong one. Nothing looks broken.

6.3.3 PLAIN - a worked example

1. A gated D latch over time. Time runs down the table.

Time	EN	D	Q after
t1	0	1	0, held
t2	1	1	1
t3	1	0	0
t4	1	1	1
t5	0	0	1, held
t6	0	1	1, held

2. At t1 enable is 0, so D is ignored and the old 0 stays.
3. From t2 to t4 enable is 1, and three changes of D give three changes of Q.
4. At t5 enable falls while D is 0, but Q keeps the value it had at the instant enable fell, which was 1.
5. Rows t2 to t4 are the problem. Q moved three times inside one enable pulse.

6.3.4 PLAIN - what is really happening inside

1. Build it from the NAND SR latch, which sets on S-bar=0 and resets on R-bar=0.
2. Put two steering gates in front. Feed D into one and NOT D into the other, and feed enable into both.
3. Enable at 0 makes both steering gates output 1, which is the NAND latch's hold command. The latch is sealed.
4. Enable at 1 sends exactly one steering gate to 0, chosen by D. So D=1 sets and D=0 resets.
5. D and NOT D can never both be 1, so the forbidden case is unreachable. That is the real reason the D latch is safe.
6. Chain two transparent latches on the same enable and data can run through both in one pulse. Your two-stage pipeline silently becomes one stage.
7. The honest version: people say a latch "stores on the falling edge". It does not. It stores continuously while enabled and merely stops at that edge.

6.3.5 TECHNICAL - the engineer's version

1. The gated D latch is **level-sensitive**. Q is a function of D throughout the active level of the enable.
2. Its timing has setup and hold measured against the closing edge, plus D to Q delay while transparent and enable to Q delay at the opening edge.
3. In CMOS it is usually not gates but a transmission-gate latch: one pass gate into the storage node, one feedback inverter, one feedback pass gate.
4. That costs roughly 8 to 12 transistors against about 20 to 24 for an edge-triggered flip-flop, so latches are smaller and faster.
5. Latch-based pipelines allow **time borrowing**: a slow path can run past the nominal boundary into the next stage's transparent window. Flip-flop pipelines cannot.
6. The cost is much harder static timing analysis, because tools must model transparency, borrowing limits and level-sensitive loops.
7. Standard cell libraries name these cells with D and L, as in DLH and DLL, meaning transparent when the enable is high or low.
8. Where experts disagree: whether latch-based design is worth it. Custom CPU teams say yes, for the area and the borrowing. Most ASIC and FPGA teams say no, because verification cost outweighs the gain.

6.3.6 WORDS - remember these

1. D latch — one data wire, one open-or-shut wire — a level-sensitive bistable whose output follows D while enabled.
2. Enable — the open-or-shut control — the level-sensitive gating signal, named EN, G or CLK.
3. Transparent — see-through — a live combinational path from D to Q during the active level.
4. Level-triggered — cares about the level, not the change — output is a function of D throughout the enable window.
5. Time borrowing — slow work spilling into the next stage — using transparency to absorb path delay past the nominal boundary.

6.4 The D flip-flop: edge-triggered memory

6.4.1 PLAIN - in simple words

1. A latch is see-through while enabled, which is dangerous in a big machine.
2. We want something that takes one sample at one instant and then stops looking, however long the control stays high.
3. That is the **D flip-flop**, and it is edge-triggered.
4. Edge-triggered means it acts on the change of the clock, not on its level.
5. A rising-edge flip-flop copies D to Q the moment the clock goes 0 to 1, then ignores D until the next rising edge.
6. A flip-flop takes a photograph. A latch runs a video.
7. This makes it safe to wire one flip-flop's output into the next one's input, which is how every pipeline in every processor is built.
8. The price is that D must be steady just before the edge and just after it. Those are setup time and hold time.
9. Break them and the flip-flop can hesitate. That is metastability, and it can never be fully removed.

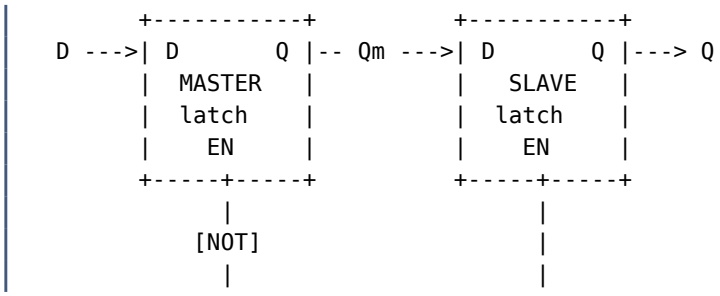
6.4.2 PLAIN - a picture in your head

1. A submarine airlock with two doors and one rule: both are never open.
2. Water enters the outer room while the outer door is open, but goes no further, because the inner door is shut.
3. The outer door shuts, the inner opens, and what is in the middle moves inward. Nothing new can follow it.
4. That is exactly a master-slave flip-flop. The master fills while the clock is low, the slave accepts while the clock is high.

Where this comparison breaks:

1. Airlock doors take a second. A flip-flop's doors close in tens of picoseconds, and any overlap is what makes hold violations possible.
2. Water cannot be half in and half out. A voltage can.

6.4.3 PLAIN - a worked example



```
CLK --------+----------------------+

CLK = 0 : master transparent, slave holding
CLK = 1 : master holding,     slave transparent
```

1. Clock at 0. The master is transparent so Qm follows D. The slave is shut, so Q does not move.
2. Clock rises. The master seals, trapping the last value of D. The slave opens and passes that trapped value to Q.
3. Change D while the clock is still 1. The master is shut, so nothing happens.
4. Clock falls. The slave seals, keeping Q, and the master opens again to track the new D for the next edge.
5. So Q changes exactly once per cycle, at the rising edge.
6. That is the race edge-triggering solves. With plain latches a value could travel through three stages in one enable pulse, skipping two cycles of work. With flip-flops one edge moves data exactly one stage.

6.4.4 PLAIN - what is really happening inside

1. Setup time is how long D must already be steady before the edge arrives.
2. Hold time is how long D must stay steady after it.
3. Together they define a small forbidden window in which D may not change.
4. The reason is physical. The master is closing a switch and handing over to a feedback loop, and that handover needs a firm push one way.
5. If D moves as the switch closes, the loop starts near the middle, where neither 0 nor 1 is winning.
6. It still recovers, because the middle is unstable, but recovery takes an unpredictable time.
7. Occasionally the next stage samples a half-formed voltage, and two different stages can then read the same wire as 0 and as 1.
8. The chance drops exponentially with the extra settling time you allow, but it never reaches zero.
9. The fix is a **synchronizer**: two or three flip-flops in a row on the same clock, whose only job is to buy the first one extra clock periods.

6.4.5 TECHNICAL - the engineer's version

1. A positive-edge-triggered D flip-flop samples D in an aperture defined by setup time t-su and hold time t-h, and drives Q after clock-to-output delay t-cq.

Part or node	t-su	t-h	t-cq
DM74LS74A, TTL	20 ns	0 ns	25 to 30 ns
Typical 7 nm cell	15 to 30 ps	5 to 20 ps	20 to 40 ps

2. The 74LS74 figures are from the Fairchild and Texas Instruments data sheets at 5 V, 25 degrees Celsius and a 15 pF load. The 7 nm figures are typical orders of magnitude, since vendor libraries are confidential.
3. Static timing analysis checks two inequalities on every path. Setup: t-cq + t-logic + t-skew <= T-clk - t-su. Hold: t-cq + t-logic >= t-h + t-skew.
4. Setup failures are fixed by slowing the clock. Hold failures are not, and must be fixed by adding delay in silicon.
5. Metastability resolution is exponential in the settling time available:

```
MTBF = exp(t_r / tau) / (T0 * f_clk * f_data)

t_r    = settling time available after the sampling edge
tau    = resolution time constant of the flip-flop
T0     = effective width of the metastable aperture
f_clk  = sampling clock frequency
f_data = rate at which the asynchronous input changes
```

6. Worked example at a deliberately pessimistic slow corner, with tau = 50 ps, T0 = 1 ps, f_clk = 1 GHz and f_data = 10 MHz:

Stages	Settling time	MTBF
1 flip-flop	400 ps	about 0.3 s
2 flip-flops	1400 ps	about 4.6 years
3 flip-flops	2400 ps	about 2.2 billion yr

7. One extra flip-flop moved the answer eight orders of magnitude, because the exponential dominates everything else.
8. Two-flop synchronizers are the convention for single-bit control signals crossing clock domains. Three are used at very high frequency or in safety-critical logic.
9. A synchronizer handles one bit. Multi-bit values crossing domains need Gray coding, a handshake or an asynchronous FIFO, because independently synchronized bits can resolve on different cycles.
10. Thomas Chaney and Charles Molnar documented this in their 1973 IEEE Transactions on Computers paper on anomalous synchronizer behaviour, and Leslie Lamport gave the impossibility argument in his 1984 note on Buridan's principle.
11. Established fact: no purely digital circuit can eliminate metastability. Marketing claim: any product described as metastability-proof. What such products mean is a very long MTBF.
12. Tools that observe this: `report_timing` in Synopsys PrimeTime or Cadence Tempus, and clock-domain-crossing checkers such as SpyGlass CDC.

6.4.6 WORDS - remember these

1. Flip-flop — memory sampling once per clock edge — an edge-triggered bistable, usually master-slave.
2. Setup time — data ready before the edge — minimum stable interval before the clock edge, t-su.
3. Hold time — data steady after the edge — minimum stable interval after the clock edge, t-h.
4. Metastability — hesitating between 0 and 1 — an output resting near the unstable equilibrium for an unbounded time.
5. MTBF — average time between rare failures — mean time between failures, exponential in available settling time.
6. Synchronizer — flip-flops that buy settling time — a chain of same-clock registers at a clock domain crossing.

6.5 The clock: why the whole chip marches in step

6.5.1 PLAIN - in simple words

1. A clock is a wire carrying a signal that goes 0, 1, 0, 1 at a steady rate.
2. Drawn on paper it looks like a row of blocks, so we call it a square wave.
3. The moment it goes from 0 to 1 is the **rising edge**, when most flip-flops act.
4. Every flip-flop shares the same clock, so on each edge the whole machine takes one step together.
5. Between edges the gates that compute work out the answers for the next step.
6. The period must be long enough for the slowest of those. That is what limits speed.
7. One billion ticks per second is 1 GHz, so each tick lasts one nanosecond.

6.5.2 PLAIN - a picture in your head

1. A large rowing boat with many rowers and one drum.
2. The drum beats and every rower pulls once. Nobody watches anyone else.
3. The beat cannot be faster than the slowest rower's stroke.
4. If the sound reaches the back later than the front, the margin shrinks. That difference is **clock skew**.

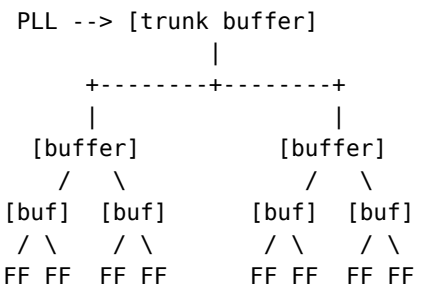
5. If the drummer is slightly irregular, that is **jitter**.
6. If some rowers rest, stop sending them the beat. That is **clock gating**.

Where this comparison breaks:

1. Rowers hear one drum. On a chip the clock must be copied through thousands of buffers to reach millions of flip-flops, and every copy adds delay and its own irregularity.

6.5.3 PLAIN – a worked example

1. A 3 GHz clock has a period of 0.333 nanoseconds, which is 333 picoseconds.
2. Say clock-to-Q is 30 ps and the next flip-flop needs 25 ps of setup. That leaves 278 ps for all the logic.
3. Allow 20 ps of skew and 10 ps of jitter and the logic has 248 ps.
4. At roughly 15 ps per gate stage, that is about 16 gate delays per cycle.
5. Roughly 15 to 20 gate delays per cycle has been near constant in processor design for two decades, whatever the frequency.



6. The aim of that tree is that every path from trunk to leaf is the same length, so all flip-flops see the edge at the same time.

6.5.4 PLAIN – what is really happening inside

1. The clock starts as a wobbling sine wave from a quartz crystal, typically 24 or 25 MHz on a modern board.
2. A phase-locked loop multiplies it up to the frequency the chip needs and squares it off.
3. That is driven into the trunk and copied outwards by buffers, roughly doubling at each level.
4. Every buffer costs delay and power. The clock network alone commonly burns 20 to 40 per cent of a high-performance chip's dynamic power.
5. Clock gating puts an AND gate in front of a branch and switches off the beat to a block with nothing to do, saving flip-flop and buffer power together.
6. Skew is not always bad. Deliberately delaying one leaf, called **useful skew**, lends time from one pipeline stage to a slower one.
7. The honest version: a real clock edge is not vertical. It rises over tens of picoseconds, and every flip-flop decides somewhere on that slope.

6.5.5 TECHNICAL – the engineer's version

1. Period T is 1 divided by frequency: 1 GHz gives 1000 ps, 3 GHz gives 333 ps, 5 GHz gives 200 ps.
2. Desktop parts now pass 6 GHz. The Intel Core i9-14900KS, launched in March 2024, was specified at 6.2 GHz maximum turbo.
3. Skew is the arrival-time difference between two flip-flops. Common sign-off targets keep global skew under about 5 per cent of the period.
4. Jitter splits into period, cycle-to-cycle and long-term jitter. A few picoseconds RMS is typical for an on-chip PLL.
5. Skew helps or hurts by direction. Positive skew from launch to capture relaxes setup and tightens hold, which is why hold violations often appear only after clock tree synthesis.

6. Distribution styles are the balanced H-tree, the buffered fishbone and the full clock mesh. Grids give the lowest skew and cost the most power, and IBM POWER designs used them.
7. Clock gating is inserted as integrated clock-gating cells, usually inferred by synthesis from an enabled register description. **Power gating** is different: it removes the supply, and loses state without retention cells.
8. Asynchronous design drops the global clock for local handshakes. It works and stays a niche. The Amulet processors at the University of Manchester in the 1990s were asynchronous implementations of the ARM architecture.
9. Tools that observe this: `report_clock_timing -type skew` in PrimeTime, and `cpupower frequency-info` on Linux for the current core clock.

6.5.6 WORDS - remember these

1. Clock — the steady tick everything follows — a periodic square wave whose active edge triggers all synchronous elements.
2. Rising edge — the moment the tick goes up — the low-to-high transition, the sampling instant for a positive-edge flip-flop.
3. Clock skew — the tick arriving at different times — the spatial difference in clock arrival between two sequential elements.
4. Jitter — the tick being slightly irregular — temporal variation of edge position, in RMS or peak-to-peak picoseconds.
5. Clock tree — the branching wiring that copies the clock — the buffered distribution network from source to every leaf register.
6. Clock gating — switching the tick off to idle parts — inserting an enable into the clock path to save dynamic power.

6.6 Registers, shift registers and counters

6.6.1 PLAIN - in simple words

1. One flip-flop holds one bit. For a byte, use eight side by side on the same clock. That group is a **register**.
2. A 32-bit register is 32 flip-flops. A CPU's small set of named working registers is the **register file**.
3. Pass each flip-flop's output to its neighbour instead and you get a **shift register**, whose whole row slides one place each edge.
4. A shift register turns one wire into a row of bits, one bit per tick. That is how a slow single-wire link becomes a parallel byte.
5. A **counter** is a register that adds one to itself on every clock edge.
6. Counters measure time, address memory, count events and divide clocks.

6.6.2 PLAIN - a picture in your head

1. A row of eight boxes on a belt, each holding a black or white marble.
2. A register is the belt stopping and all eight boxes being refilled at once from eight chutes above.
3. A shift register is the belt moving one box right each tick, a new marble dropping in at the left and the rightmost falling off.
4. Feed one marble per tick, wait eight ticks, and a full byte is lined up. That is serial-to-parallel conversion.
5. A counter is a row of odometer wheels, the rightmost turning every tick and nudging its neighbour when it passes nine.

Where this comparison breaks:

1. On an odometer the nudge travels wheel by wheel, taking time. That is a **ripple counter**, the slow kind.
2. A good digital counter computes every digit in parallel and changes them all on one edge. That is a **synchronous counter**.

6.6.3 PLAIN - a worked example

1. A 4-bit synchronous up counter, one row per clock edge.

Tick	Q3 Q2 Q1 Q0	Value
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	10
11	1 0 1 1	11
12	1 1 0 0	12
13	1 1 0 1	13
14	1 1 1 0	14
15	1 1 1 1	15
16	0 0 0 0	0, wrapped

2. Q0 changes every tick, so its frequency is half the clock. Q1 changes every second tick, so its frequency is a quarter.
3. A 4-bit counter is therefore also a divide-by-2, 4, 8 and 16 circuit, free.
4. The rule for each bit: bit n flips when all the bits below it are 1.
5. At tick 16 it rolls over. Four bits count only to 15, and wrapping is not an error, it is arithmetic modulo 16.

6.6.4 PLAIN - what is really happening inside

1. In a synchronous counter every flip-flop shares the clock, with a little logic in front computing its next value.
2. For bit 0 that logic is "the opposite of what I am". For bit 1 it is "flip me if bit 0 is 1", and so on up the chain.
3. All of those run in parallel during the clock period, and at the edge all bits update together.
4. In a ripple counter only bit 0 sees the real clock. Bit 1 is clocked by bit 0's output, bit 2 by bit 1's, and so on.
5. That is cheaper in gates, but the change travels along the chain, so the top bit updates several gate delays after the bottom.
6. Going from 7 to 8, a 4-bit ripple counter can briefly show 6, then 4, then 0. Anything sampling it then reads a lie.
7. A shift register is simpler still. Each flip-flop's D input is the previous flip-flop's Q output, with no logic in between.

6.6.5 TECHNICAL - the engineer's version

1. A register is an n-bit array of edge-triggered flip-flops sharing a clock, and usually a reset and a load enable.
2. A **register file** is a small multi-ported array. A 64-bit RISC core has 32 architectural integer registers, and a 4-wide machine may need 8 read and 4 write ports, which is why register files use custom SRAM-like cells rather than standard flip-flops.

3. Shift registers are classed by interface: SISO, SIPO, PISO and PIPO, for serial or parallel in and out.
4. Real parts: the 74HC595 is an 8-bit serial-in, parallel-out shift register with an output latch, widely used to drive LED arrays from three microcontroller pins. The 74HC165 is its parallel-in, serial-out partner.
5. Serial-to-parallel conversion by shift register is how SPI, JTAG boundary scan and many display drivers work.
6. A **linear feedback shift register** feeds an exclusive-OR of chosen taps back to the input. A maximal-length n-bit LFSR cycles through all 2 to the power n, minus 1, non-zero states, and is used for test patterns and for scrambling in Ethernet and PCI Express.
7. Ripple counters cost the fewest gates but glitch and are limited by the chain delay. Synchronous counters cost more logic, never glitch at the outputs, and are limited only by one stage.
8. A synchronous counter in Verilog is four lines:

```
always @(posedge clk) begin
  if (rst)      count <= 4'd0;
  else if (en)  count <= count + 4'd1;
end
```

9. Tools that observe this: `perf stat` on Linux reads hardware performance counters, which are wide synchronous counters inside the CPU, and `rdtsc` on x86 reads the time-stamp counter.

6.6.6 WORDS - remember these

1. Register — a row of flip-flops holding a number — an n-bit array of edge-triggered elements on a common clock.
2. Register file — the CPU's small set of named boxes — a multi-ported array of architectural registers with dedicated ports.
3. Shift register — bits sliding one place per tick — a chain of flip-flops with Q of one feeding D of the next.
4. Ripple counter — digits nudging each other along — an asynchronous counter where each stage clocks the next.
5. Synchronous counter — all digits changing together — next-state logic on a shared clock, so all bits settle on one edge.
6. LFSR — a shift register with a feedback trick that looks random — a linear feedback shift register giving a pseudo-random sequence.

6.7 Finite state machines

6.7.1 PLAIN - in simple words

1. A machine with memory can be in different situations. Each is a **state**.
2. A **finite state machine** has a fixed list of states and rules for moving between them.
3. A rule says: in this state, seeing this input, move to that state on the next clock tick.
4. It also produces outputs, which are the things it makes happen.
5. Three pieces build one: a register holding the state, logic computing the next state, and logic computing the outputs.
6. If outputs depend only on the state, it is a **Moore machine**.
7. If they depend on the state and the current inputs, it is a **Mealy machine**. Moore outputs are steadier, Mealy machines need fewer states.

6.7.2 PLAIN - a picture in your head

1. A board game where you move one token between numbered squares.
2. The token's square is the state, and there is only one token.
3. The painted arrows are the transitions, each with a condition written on it.
4. The clock is the drum from section 6.5. Between beats you decide, on the beat you move.

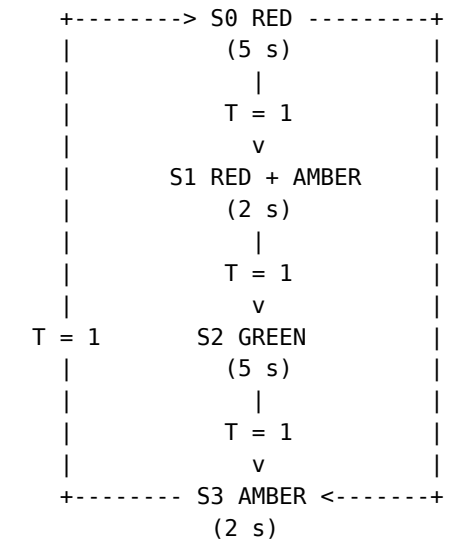
5. Squares with an instruction printed on them, like “turn the green lamp on”, give Moore outputs. Arrows with an instruction, like “ring the bell as you pass”, give Mealy outputs.

Where this comparison breaks:

1. In a board game you can pause and think. A real state machine must have its next state fully computed before the next clock edge, every time.

6.7.3 PLAIN – a worked example

1. A traffic light for one direction, with one input T meaning the countdown timer has finished.



2. The state table, with the lamps written R, A and G.

State	Input T	Next state	Lamps on
S0 RED	0	S0	R
S0 RED	1	S1	R
S1 RED+AMBER	0	S1	R, A
S1 RED+AMBER	1	S2	R, A
S2 GREEN	0	S2	G
S2 GREEN	1	S3	G
S3 AMBER	0	S3	A
S3 AMBER	1	S0	A

3. This is a Moore machine. In the last column the lamps depend only on the state, never on T.
4. Four states need two bits, so the state register is two flip-flops, with S0 = 00, S1 = 01, S2 = 10 and S3 = 11.
5. To make it Mealy, add an output firing only during a move, such as a pulse reloading the timer. That pulse depends on the state and on T being 1.

```
always @(posedge clk)
  if (rst) state <= S0;
  else case (state)
    S0: state <= T ? S1 : S0;
    S1: state <= T ? S2 : S1;
    S2: state <= T ? S3 : S2;
    S3: state <= T ? S0 : S3;
  endcase
```

6.7.4 PLAIN - what is really happening inside

1. On each clock edge the state register captures whatever the next-state logic worked out during the previous period.
2. The next-state logic is pure gates, seeing the current state bits and the inputs and producing the next state bits.
3. The output logic is pure gates too, and in a Moore machine it sees only the state bits.
4. So the loop is: state register drives logic, logic drives state register, clock decides when it advances. Nothing else.
5. A Moore output cannot react to an input in the same cycle. That one-tick delay is the price of its stability.
6. Now the big claim. A CPU's control unit is exactly this structure and nothing more.
7. Its state includes where it is in the instruction cycle. Its inputs include decoded instruction bits, flags and interrupt lines. Its outputs are the control wires that open registers, pick an ALU operation and start a memory access.
8. The only difference from our traffic light is size: four states here, hundreds across dozens of units in an out-of-order core.
9. The honest version: a modern CPU is not one giant state machine. It is many communicating state machines, plus large regular structures such as caches and queues that nobody describes state by state.

6.7.5 TECHNICAL - the engineer's version

1. Formally an FSM is a state set, an input alphabet, an output alphabet, a transition function, an output function and an initial state.
2. Any Mealy machine converts to a Moore machine, usually at the cost of more states.
3. Encoding matters in silicon. Binary uses the fewest flip-flops, 3 bits for 8 states. One-hot uses 8, but gives the simplest next-state logic. Gray uses 3 and changes only one bit per step.
4. One-hot is the convention in FPGA design, where flip-flops are plentiful and wide logic is expensive. Binary is more common in ASICs, where flip-flop area is the cost.
5. Gray encoding is used where state bits cross a clock domain, because only one bit changes per transition, so only one bit can be sampled ambiguously. That is standard inside asynchronous FIFOs.
6. Illegal states must be handled. Safety-related designs use a full `default` branch forcing a return to a known state, so a single-event upset cannot wedge the machine.
7. Maurice Wilkes proposed **microprogramming** in 1951, holding the control unit's transitions in a small read-only memory rather than in fixed gates. IBM used it heavily in System/360 from 1964, which is how one architecture ran on very different hardware.
8. Modern x86 processors still contain microcode, which is why microcode updates can change processor behaviour after manufacture. That is how the 2018 Spectre and Meltdown mitigations reached shipped chips.
9. Tools that observe this: `yosys -p 'fsm_detect; fsm_map'` reports inferred state machines, and commercial synthesis lists the encodings it chose.

6.7.6 WORDS - remember these

1. State — the situation the machine is in — one value held in the state register, from a finite set.
2. Transition — the move from one situation to another — the next-state function evaluated at a clock edge.
3. Moore machine — outputs depend on where you are — output is a function of the current state alone.
4. Mealy machine — outputs depend on where you are and what you see — output is a function of state and inputs.
5. One-hot — one flip-flop per state, only one set — an encoding of Hamming weight 1, favoured in FPGAs.
6. Microcode — the control unit's own tiny program — a stored microprogram in ROM or writable control store sequencing control signals.

6.8 SRAM: the six-transistor cell

6.8.1 PLAIN - in simple words

1. **SRAM** means static random access memory. Static means it holds its value while power is on, with no help.
2. One cell is the two-inverter loop of section 6.1 plus two switches that let us reach inside.
3. Each inverter is two transistors, and the two switches make six. Hence the name, the six-transistor cell.
4. Both switches are opened by one wire, the **word line**.
5. Outside are two wires, the **bit lines**, one carrying the value and one its opposite.
6. To read, open the word line and let the cell tip the bit lines slightly apart. An amplifier notices which way.
7. To write, drive the bit lines hard and open the word line. The drivers overpower the loop and flip it.
8. SRAM is fast because the answer is already sitting there, and expensive because six transistors per bit is a lot of silicon.

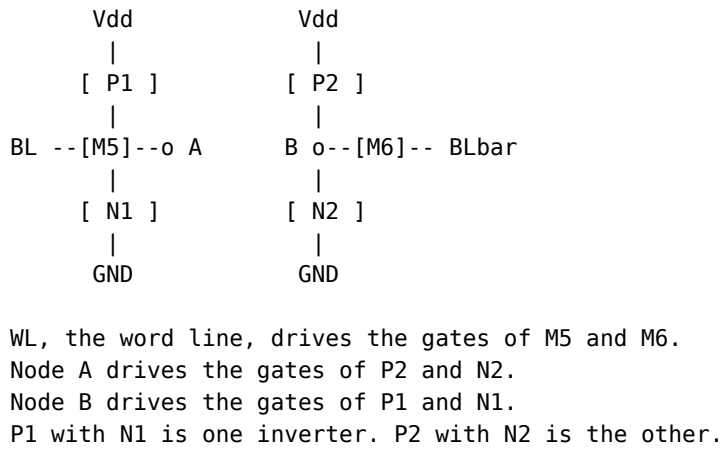
6.8.2 PLAIN - a picture in your head

1. A see-saw with a child sitting firmly on one end.
2. Reading is feeling which end is down, without changing it.
3. Writing is an adult pushing the high end down until it settles the other way.
4. It never needs topping up. While gravity works, it stays.
5. But a see-saw takes a lot of space, and only a few fit in the playground.

Where this comparison breaks:

1. Gravity is free. An SRAM cell's gravity is a small current from the supply running for ever, and across a large cache that leakage is a real part of a chip's idle power.

6.8.3 PLAIN - a worked example



1. To store a 1, node A is high and node B is low. P1 holds A up, N2 holds B down, and each keeps the other in place.
2. To read, first charge both bit lines to the supply and let go. That is precharging.
3. Raise the word line. B is low, so the cell pulls BLbar down through M6 and N2, while BL stays high.
4. After a short time the bit lines differ by about 50 to 100 millivolts, and a sense amplifier drives that difference out at full logic level.
5. To write a 0, drive BL to ground and BLbar to the supply, then raise the word line. The external drivers beat the cell and node A is dragged down.
6. Sizing is the whole art. M5 and M6 must be weak enough not to disturb the cell on a read, yet the write drivers must still beat P1 and P2.

6.8.4 PLAIN - what is really happening inside

1. A cache is a rectangular array of millions of these cells, sharing word lines along rows and bit lines down columns.
2. Only one word line is raised at a time, which selects one row.
3. Every column in that row dumps its value onto its own bit line pair at once, so a whole row is read in parallel, and column select logic then picks the bits you asked for.
4. SRAM is fast because the sequence is short: decode, raise a word line, let the bit lines separate a little, amplify, output.
5. There is no waiting for a capacitor to fill and no restoring afterwards, because reading did not destroy anything.
6. The honest version: reading does disturb the cell slightly. The read current lifts the low node, and in a weak cell or at low supply that can flip the bit. This is read upset, and it is why SRAM fails at low voltage before ordinary logic does.

6.8.5 TECHNICAL - the engineer's version

1. The standard cell is 6T. An 8T cell adds a separate read port to remove read disturb, and 10T cells are used at very low voltage.
2. Published high-density 6T bitcell areas:

Process	Bitcell area	Volume from
TSMC N5	0.021 sq um	2020
TSMC N3E	0.021 sq um	Q4 2023
TSMC N2	0.0175 sq um	H2 2025

3. Sources quote TSMC N5 as either 0.021 or 0.0199 square micrometres depending on the cell variant. The fact to keep is that SRAM barely shrank from 5 nm to 3 nm and shrank again at 2 nm.
4. Cost estimate, since SRAM is never sold by the gigabyte. A gigabyte is about 8.6 thousand million bits, so at 0.021 square micrometres per bit that is roughly 180 square millimetres of bitcell, or about 300 once decoders, sense amplifiers and error correction are added.
5. A 300 mm wafer holds about 70 700 square millimetres, and leading-edge wafer prices are publicly estimated at 16 000 to 20 000 US dollars. That puts SRAM near 100 to 200 US dollars per gigabyte, as an order of magnitude only.
6. Typical latencies in a current core, roughly 1 ns, 4 ns and 12 ns at 4 GHz:

Level	Technology	Typical latency
L1 data cache	SRAM	4 to 5 cycles
L2 cache	SRAM	12 to 20 cycles
L3 cache	SRAM	40 to 60 cycles

7. The Intel 3101, released in 1969, was Intel's first product: a 64-bit bipolar SRAM. The Intel 1101 of the same year was a 256-bit MOS SRAM.
8. SRAM leakage is dominated by subthreshold and gate leakage and scales badly, so designs use multiple threshold voltages and array power gating.
9. Tools that observe this: `lscpu` and `getconf -a | grep CACHE` print cache sizes, and `perf stat -e cache-misses` counts misses against them.

6.8.6 WORDS - remember these

1. SRAM — memory that holds itself up while powered — static RAM built from cross-coupled inverter cells.
2. 6T cell — the six-transistor bit — two inverters plus two access transistors sharing one word line.
3. Word line — the wire that selects a row — the row line driving the gates of the access transistors.
4. Bit line — the wire carrying the value in and out — one of a complementary pair, precharged before each read.
5. Sense amplifier — hears a whisper and shouts it — a differential amplifier resolving a small bit line split into full logic.
6. Read upset — a read accidentally changing the bit — a cell flip caused by read current lifting the storage node.

6.9 DRAM: one transistor and one capacitor

6.9.1 PLAIN - in simple words

1. **DRAM** means dynamic random access memory. Dynamic means it does not hold itself up and must be topped up constantly.
2. One cell is a single transistor acting as a switch, plus a tiny capacitor, which is a bucket for charge.
3. Charge in the bucket means 1. An empty bucket means 0.
4. Two components instead of six is why DRAM is so much cheaper and denser than SRAM. That is the entire reason it exists.
5. The bucket leaks, and left alone it empties in a fraction of a second.
6. So the chip walks through every row, reads it and writes it straight back. That is **refresh**, and it never stops while power is on.
7. Worse, reading a cell empties the bucket, so every read is automatically followed by a write-back of what was found.

6.9.2 PLAIN - a picture in your head

1. A wall of thousands of small buckets, each with a slow drip.
2. In under a tenth of a second a full bucket becomes ambiguous.
3. A worker walks the wall topping up every bucket that still looks full, and must get round before the first goes bad. That deadline is the refresh interval.
4. Checking a bucket means tipping it into a measuring tray, so you must then pour it back.
5. The tray starts half full, so a bucket tips it slightly up or down. That tiny difference is what the sense amplifier measures.

Where this comparison breaks:

1. Water can be at any level. A cell that has half leaked is not half a bit, it is a bit the sense amplifier will read wrongly. No partial credit.

6.9.3 PLAIN - a worked example

1. A modern DRAM cell capacitor is around 6 to 10 femtofarads and the supply is about 1.1 volts.
2. Charge equals capacitance times voltage, so 10 femtofarads times 1.1 volts is 11 femtocoulombs.
3. Dividing by the charge on one electron gives roughly 69 000 electrons for a stored 1. That is the whole physical difference between 1 and 0 in your main memory.
4. Now the refresh arithmetic for DDR4. The standard refresh window is 64 milliseconds up to 85 degrees Celsius.
5. A DDR4 device needs 8192 refresh commands to cover everything, so one command must go out on average every 64 divided by 8192 milliseconds, which is 7.8 microseconds. That figure is tREFI.

6. Above 85 degrees the window halves to 32 milliseconds and tREFI halves to 3.9 microseconds.
7. DDR5 tightened this. Its normal window is 32 milliseconds up to 85 degrees, giving a tREFI of 3.9 microseconds, and 16 milliseconds from 85 to 95.
8. So the familiar answer of 64 milliseconds is right for DDR through DDR4 and out of date for DDR5. Both figures are in the JEDEC standards.

6.9.4 PLAIN - what is really happening inside

1. Cells sit in a grid. A row is selected by a word line, a column is read out on a bit line.
2. Precharge. Every bit line is driven to exactly halfway between supply and ground, then left floating.
3. Activate. The chosen word line rises, connecting every cell in that row to its bit line.
4. The cell's tiny charge tips the much larger bit line by tens of millivolts, and the cell is now drained. That is the destructive read.
5. Sense. The sense amplifier is a cross-coupled pair, exactly like section 6.1, which latches onto whichever side is higher and drives the pair fully apart.
6. That same act refills the cell through the still-open word line, so the write-back is automatic and free.
7. The whole row now sits in the sense amplifiers, together called the **row buffer**, typically 1 or 2 kilobytes.
8. Column select. Only now do we pick the bytes we asked for and send them off the chip.
9. Another request in the same row skips straight to that last step. That is a **row buffer hit**, and it is much faster. A different row means precharging and activating again.
10. Because the address arrives in two halves, the control signals are called **RAS**, row address strobe, and **CAS**, column address strobe. The names are from the 1970s and the signals still exist.

6.9.5 TECHNICAL - the engineer's version

1. Robert Dennard invented the one-transistor, one-capacitor cell at IBM's Thomas J. Watson Research Center in 1966, filed in 1967, and was granted United States patent 3,387,286 in 1968.
2. The Intel 1103, launched in October 1970, was a 1 kilobit PMOS DRAM and the first commercially successful one. It ended magnetic core memory.
3. Address multiplexing latches the row address on the falling edge of RAS and the column address on the falling edge of CAS. This halved the pin count, and is why the terms survive in modern timing parameters.
4. The four numbers on any memory module label:

Symbol	Meaning	Typical DDR5
CL	CAS to data out	40 to 46 cycles
tRCD	activate to CAS	39 cycles
tRP	precharge time	39 cycles
tRAS	row active minimum	76 to 84 cycles

5. A row buffer hit costs CL alone. A row conflict costs tRP plus tRCD plus CL. At DDR5-6400 those are about 12.5 ns and about 37 ns, before queueing.
6. Above the cell: cells form subarrays, subarrays form **banks**, banks form **bank groups**, chips accessed together form a **rank**, and a controller port is a **channel**. DDR5 splits each DIMM into two 32-bit sub-channels.
7. Banks let the controller keep several rows open and overlap activation with data transfer, which is where most real bandwidth comes from.
8. The DDR family, with JEDEC publication years:

Generation	Standard year	Voltage	Data rate MT/s
SDR SDRAM	1993	3.3 V	66 to 133
DDR, JESD79	2000	2.5 V	200 to 400
DDR2, JESD79-2	2003	1.8 V	400 to 1066
DDR3, JESD79-3	2007	1.5 V	800 to 2133
DDR4, JESD79-4	2012	1.2 V	1600 to 3200
DDR5, JESD79-5	2020	1.1 V	3200 to 8800

9. DDR4 parts reached the market in 2014, two years after the standard. DDR5 was published on 14 July 2020, and JEDEC raised its ceiling to 8800 MT/s in an update published in April 2024 that also added anti-rowhammer features.
10. DDR6 is in development. Reporting through 2025 and 2026 points to a 2027 timeframe and an 8800 to 17 600 MT/s range. No standard is published yet, so treat that as expectation, not fact.
11. Now the point of the section. DDR-400 at CL3 in 2000 gave a CAS latency of 15 ns and 3.2 GB/s per 64-bit module. DDR5-6400 at CL40 gives 12.5 ns and 51.2 GB/s. Bandwidth rose sixteenfold, latency improved about twenty per cent, in twenty-five years.
12. The reason is physics, not laziness. Bandwidth is bought with more banks, wider buses and faster signalling. Latency is set by charging a long, heavy bit line through one small transistor, and that has not changed.
13. DDR5 added a same-bank refresh command, REFsb, beside the all-bank REFab. For a 16 Gb device Micron quotes tRFC of 295 ns for REFab and 130 ns for REFsb, and reports 6 to 9 per cent throughput gain from using REFsb.
14. Every DDR5 chip carries on-die ECC, 8 parity bits over 128 data bits. That covers internal cell errors and is not the same as module-level ECC.
15. Tools that observe this: `dmidecode -t memory` prints module type, speed and rank, and Intel MLC measures achieved bandwidth and idle latency.

6.9.6 WORDS - remember these

1. DRAM — memory that must be topped up — dynamic RAM using a one-transistor, one-capacitor cell.
2. Refresh — the constant topping up — periodic read and restore of every row inside the retention window, paced by tREFI.
3. Destructive read — looking at it empties it — charge sharing from cell to bit line, requiring an automatic restore.
4. Row buffer — the row currently held open — the sense amplifier array holding one activated row, typically 1 to 2 kilobytes.
5. RAS and CAS — the two halves of an address — row address strobe and column address strobe, from multiplexed addressing.
6. Bank — an independently openable section — a subarray group with its own row buffer, allowing overlapped access.
7. Rank — a set of chips answering together — devices on a module selected by one chip select and sharing the data bus.
8. Channel — one road between controller and memory — an independent memory controller port with its own command and data buses.

6.10 Non-volatile cells: from mask ROM to flash

6.10.1 PLAIN - in simple words

1. Everything so far forgets when power goes. Now we want cells that remember without it.
2. **Mask ROM** has the bits built in during manufacture by the shape of one layer. It can never be changed.

3. **PROM** leaves the factory blank with a tiny fuse at every bit. You blow selected fuses with current, once, and a blown fuse cannot be repaired.
4. **EPROM** stores charge instead of blowing metal. Its package has a quartz window: shine ultraviolet light through it for about twenty minutes and every bit returns to blank.
5. **EEPROM** does the same erase with voltage rather than light, one byte at a time.
6. **Flash** is EEPROM that erases in large blocks instead of by byte. That one restriction made it far smaller and far cheaper.
7. All three of the last ones store a bit as electrons trapped on a small island of conductor with no wires attached to it.

6.10.2 PLAIN - a picture in your head

1. A hollow in the top of a hill, ringed by a high ridge, with no path in.
2. Marbles in the hollow are the trapped electrons, and present or absent is the stored bit.
3. Nothing takes them out, because the ridge is too high. They sit for years.
4. To put marbles in you either fire them hard enough to fly over the ridge, which is hot-electron injection, or tilt the hill steeply so they slip across a thin part, which is tunnelling.
5. To read you notice that the hill's shape is slightly different when the pile is there.
6. Every firing chips rock off the ridge, and after enough thousands of times marbles leak out on their own. That is wear.

Where this comparison breaks:

1. Marbles are in or out. Electrons are quantum objects and tunnelling is a probability, not a rolling motion. The ridge is an energy barrier of roughly 3.2 electron volts between silicon and silicon dioxide.

6.10.3 PLAIN - a worked example

```
     control gate
=========================
     oxide layer
-------------------------
     floating gate          <- charge lives here
-------------------------
     tunnel oxide, 8 to 10 nm thick
=========================
 source |  channel  | drain
=========================
          substrate
```

1. A floating-gate transistor is an ordinary transistor with a second gate buried in the insulator, connected to nothing.
2. With no electrons on it, a moderate voltage on the control gate switches the transistor on. Call that erased.
3. Trap electrons there and their negative charge repels the control gate's influence, so the same voltage no longer switches it on. Call that programmed.
4. Reading is applying a fixed voltage and seeing whether current flows. The threshold has shifted, typically by two to four volts in older parts.
5. Control the charge finely and you get several distinct levels, and so more than one bit per cell.

Cell type	Bits per cell	Levels	Typical endurance
SLC	1	2	50k to 100k cycles
MLC	2	4	about 10k cycles
TLC	3	8	1k to 3k cycles

Cell type	Bits per cell	Levels	Typical endurance
QLC	4	16	a few hundred

6.10.4 PLAIN - what is really happening inside

1. NOR flash programs by **channel hot-electron injection**: a large voltage across the channel accelerates electrons until some jump the barrier.
2. That is fast per byte but draws heavy current, so it cannot be done to many cells at once.
3. NAND flash uses **Fowler-Nordheim tunnelling** for both writing and erasing. A high field across a very thin oxide lets electrons tunnel through.
4. Tunnelling draws almost no current, so thousands of cells can be programmed together. That is exactly why NAND is used for bulk storage and NOR is not.
5. The 15 to 20 volts needed are made on the chip by charge pumps, from an ordinary 3.3 volt supply.
6. Erase pulls electrons back off, and only a whole block at a time, typically several megabytes.
7. That block-erase rule causes nearly everything odd about SSDs. You cannot overwrite in place, so you write elsewhere, remember where, and collect the garbage later.
8. Retention comes from the barrier height, so it is always quoted with a temperature, because heat helps electrons escape.
9. The honest version: most flash sold today is not floating-gate at all. Modern 3D NAND mostly uses **charge trap** cells, where electrons sit in a layer of silicon nitride full of defects rather than on a conductive island.

6.10.5 TECHNICAL - the engineer's version

1. The floating-gate MOSFET was invented by Dawon Kahng and Simon Sze at Bell Labs in 1967.
2. Dov Frohman at Intel turned it into a product. The Intel 1702 of 1971 was the first EPROM, holding 2048 bits as 256 words of 8 bits, erased by ultraviolet light through a quartz window.
3. That erase uses light at 253.7 nm, typically for 20 to 30 minutes. Ordinary glass blocks that wavelength, which is why the window had to be quartz.
4. EEPROM based on Fowler-Nordheim tunnelling was patented by Siemens in 1974. Intel's 2816, a 16 kilobit EEPROM developed by a team including George Perlegos, arrived in 1978.
5. Fujio Masuoka invented flash at Toshiba in 1980. NOR flash was presented in 1984 and NAND flash at the IEEE International Electron Devices Meeting in San Francisco in 1987. Intel shipped the first commercial NOR flash in 1988.
6. NOR gives random byte-level read and is used for boot code executed in place. NAND is read and written a page at a time and erased a block at a time.
7. Since Samsung's V-NAND in 2013, NAND has been built vertically. Layer counts passed 200 in 2022 and current products sit in the 200 to 300 plus range. Stacking, not lithographic shrinking, is now the density driver.
8. Retention is standardized. JEDEC JESD218 requires a client SSD to hold data 1 year at 30 degrees Celsius and an enterprise SSD 3 months at 40, at the end of rated endurance. Fresh cells do far better.
9. Embedded MRAM has replaced embedded flash at 22 nm and below at TSMC, Samsung and GlobalFoundries, because flash cell physics does not scale below about 28 nm.
10. Tools that observe this: `smartctl -a /dev/nvme0` and `nvme smart-log` report NAND wear as percentage used, and `flashrom` reads and writes NOR devices.

6.10.6 WORDS - remember these

1. Mask ROM — bits built in at the factory — contents fixed by a photomask layer during fabrication.
2. PROM — write once, never again — one-time programmable memory using fusible links or antifuses.
3. EPROM — erasable by ultraviolet light — floating-gate memory in a package with a quartz window, erased in bulk.

4. EEPROM — erasable by voltage, byte by byte — electrically erasable floating-gate memory using tunnelling.
5. Flash — EEPROM that erases in big blocks — block-erasable floating-gate or charge-trap memory, NOR or NAND organized.
6. Floating gate — an island of conductor holding electrons — an isolated gate in the oxide that shifts the transistor threshold voltage.
7. Tunnelling — electrons crossing a barrier they should not — Fowler-Nordheim conduction through thin oxide under a high field.
8. Endurance — how many rewrites it survives — rated program and erase cycles before the oxide degrades out of specification.

6.11 Memory as a grid: addresses, word lines and bit lines

6.11.1 PLAIN - in simple words

1. A memory chip is a rectangle of cells laid out in rows and columns.
2. Horizontal wires run across the rows. Each is a word line, and raising one selects that whole row.
3. Vertical wires run down the columns. Each is a bit line, carrying one cell's value in or out.
4. An **address** is a number saying which cell or group of cells you want.
5. The circuit that turns an address into exactly one raised word line is the **address decoder**.
6. A decoder with n inputs selects 2 to the power n rows, because n binary digits make that many patterns.
7. So with n address lines you reach 2 to the power n locations, and to reach L locations you need n equal to the base-2 logarithm of L, rounded up.

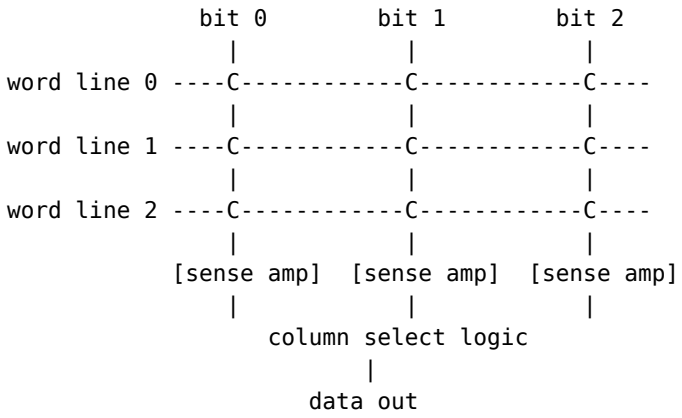
6.11.2 PLAIN - a picture in your head

1. A block of post office boxes arranged in a grid on a wall.
2. Every box has a number, and that number is the address.
3. A clerk takes the number, walks to one row, then along it to one box.
4. The decoder is the clerk. Number in, one box selected, nothing else.
5. The clerk does not read the letter. Reading is a separate job done by the sense amplifiers on the bit lines.

Where this comparison breaks:

1. A clerk walks, so distant boxes take longer. A decoder is a tree of gates and takes the same time for any address. That is exactly what "random access" means: every location costs the same.

6.11.3 PLAIN - a worked example



1. Forward. How many locations can 16 address lines reach? Two to the power 16 is 65 536.
2. Backward. How many lines does a 1 gibibyte byte-addressable memory need? One gibibyte is 2 to the power 30 bytes, so 30 lines.

Capacity, byte-addressed	Address lines	Locations
1 KiB	10	1 024
64 KiB	16	65 536
4 MiB	22	4 194 304
1 GiB	30	1 073 741 824
16 GiB	34	17 179 869 184

3. Every doubling of capacity costs exactly one more address line.
4. A historical note this explains. The original IBM PC used an Intel 8088 with 20 address lines, reaching 2 to the power 20 bytes, which is 1 MiB. That is where the famous 640 KiB usable limit came from, once the top 384 KiB was reserved.

6.11.4 PLAIN - what is really happening inside

1. A decoder with 20 inputs and a million outputs, built directly, would be absurd. Real designs never do that.
2. First trick, split the address: some bits pick a row, the rest pick a column within the row.
3. A one megabit array becomes 1024 rows by 1024 columns, needing one 10-to-1024 row decoder and one 10-to-1024 column selector.
4. That is also why arrays are roughly square. A square minimizes total wire length, and wire length is delay.
5. Second trick, predecoding: decode in two stages, for instance two 5-to-32 decoders whose outputs are combined in pairs.
6. Third trick, used in DRAM: send row bits and column bits down the same pins at different times. That is the RAS and CAS scheme of section 6.9, halving the address pins.
7. Real chips also carry spare rows and columns. If test finds a bad one, fuses redirect its address to a spare, so almost every memory chip you have used contains repaired defects.

6.11.5 TECHNICAL - the engineer's version

1. An n-to-2-to-the-n decoder asserts exactly one output per input code. Built flat it needs 2 to the power n gates of n inputs each, so it is always built hierarchically, with predecoding feeding NAND word line drivers.
2. Array organization is quoted as rows by columns by bits per word, plus a column multiplexer ratio. A 4:1 column mux means four bit line pairs share one sense amplifier, trading speed for area.
3. Sub-arrays, called mats, keep word lines and bit lines short. A large cache is dozens of mats with hierarchical decoding. The academic tool CACTI models exactly this trade-off.
4. Cache address terminology is the same grid indexed cleverly: low bits are the byte offset in a line, middle bits are the set index driving the decoder, high bits are the tag that is compared.
5. Redundancy repair uses laser fuses or electrical antifuses set at wafer test. DDR4 and DDR5 also support post-package repair, retiring a failing row in the field.
6. The formulas worth memorizing:

```
locations      = 2 ** address_lines
address_lines  = ceil(log2(locations))
total_bits     = locations * word_width
```

7. Tools that observe this: `lsmem` and `/proc/iomem` show the physical address layout on Linux, and `cpuid` reports the physical address width a processor implements, commonly 46 to 52 bits rather than the full 64.

6.11.6 WORDS - remember these

1. Address — the number naming a location — a binary code decoded to select one row and one column of an array.
2. Address decoder — turns a number into one selected row — an n-to-2-to-the-n selector, built with predecoding.
3. Word line — the horizontal select wire — the row line driven high to connect a row of cells to their bit lines.
4. Bit line — the vertical data wire — the column line carrying charge or current between a cell and a sense amplifier.
5. Random access — every location costs the same — access latency independent of the address, unlike tape or a spinning disk.
6. Redundancy repair — spare rows replacing broken ones — fuse or antifuse remapping of failing addresses to redundant elements.

6.12 Volatile, non-volatile, and memory versus storage

6.12.1 PLAIN - in simple words

1. **Volatile** means the contents vanish when power goes. SRAM and DRAM are volatile.
2. **Non-volatile** means they survive. Flash, ROM and disks are non-volatile.
3. That is a property of the cell, and it is not the same as the difference between memory and storage, though people mix the two up constantly.
4. **Memory** is what the processor reaches directly, one byte at a time, with ordinary load and store instructions.
5. **Storage** is what it reaches by asking a device to move a block, through operating system calls like read and write.
6. The real dividing line is how the processor talks to it, not whether it forgets.
7. Memory is measured in nanoseconds. Storage is microseconds for an SSD and milliseconds for a spinning disk.
8. The line is genuinely blurring, because some devices are non-volatile yet sit on the memory bus and are addressed a byte at a time.

6.12.2 PLAIN - a picture in your head

1. Memory is the desk you work at. Storage is the filing cabinet in the corner.
2. Anything on the desk you touch instantly, without asking anyone.
3. Anything in the cabinet you must fetch, and you fetch a whole folder, not one line of one page.
4. At the end of the day the cleaners clear the desk and leave the cabinet.
5. Persistent memory is a desk the cleaners have been told to leave alone. Still a desk, with desk speed and desk habits, but it survives the night.

Where this comparison breaks:

1. A desk that survives the night sounds purely good. It creates a hard new problem: if power fails halfway through an update you are left with a half-updated permanent structure, and most programs are not written for that.

6.12.3 PLAIN - a worked example

1. Watch what happens when you save a file.
2. Your editor holds the text in DRAM: fast, byte-addressable, gone if power drops.
3. You press save, and the operating system copies the bytes into a page cache, still in DRAM.
4. It then issues a block write to the SSD, whose controller collects it into a page and eventually programs it into NAND cells.

5. Only after that program completes is the data non-volatile.
6. Lose power between steps 3 and 5 and the file is lost, even though the save appeared to succeed. That is why `fsync` exists and why databases care about it so much.
7. With persistent memory on the memory bus the sequence is shorter: a store instruction plus a cache line flush and a fence makes the data durable, with no block layer at all.

6.12.4 PLAIN - what is really happening inside

1. Byte-addressable means the address bus can name any single byte, and load and store instructions work on it directly.
2. Block-addressable means the smallest unit that can move is a sector or page, historically 512 bytes and now usually 4096.
3. That difference is not a detail. It is why file systems, page caches, drivers and interrupt handling exist at all.
4. Non-volatile memory on the memory bus removes that stack. You map the device into your address space and use pointers.
5. The catch is the CPU caches. A store lands in a cache, not in the persistent device, and until it is flushed out it is still volatile.
6. So persistent memory programming needs explicit instructions to push cache lines out and to order those pushes.
7. The honest version: both “persistent memory is just fast storage” and “it is just memory that survives” are wrong. It is a third thing, with the addressing of memory and the durability rules of storage.

6.12.5 TECHNICAL - the engineer’s version

1. The three-way comparison, with figures for August 2026:

Property	SRAM	DRAM	NAND flash
Devices per bit	6 transistors	1 tr + 1 cap	1 cell, 1 to 4 bits
Read latency	1 to 10 ns	45 to 90 ns	30 to 100 us
Cost per GB	100 to 200 USD, est	16 to 18 USD	about 0.08 USD
Volatility	volatile	volatile, refreshed	non-volatile

2. Read the cost row carefully. SRAM is never sold by the gigabyte, so that figure is an estimate from silicon area and public wafer price estimates, as worked out in section 6.8.
3. The DRAM figure must be dated. TrendForce reported conventional DRAM contract prices up 90 to 95 per cent in the first quarter of 2026. Retail DDR5 in August 2026 was 16 to 18 US dollars per gigabyte, against 2 to 4 before.
4. The NAND figure is a drive-level price, around 76 US dollars per terabyte for consumer Gen 4 NVMe in mid 2026, including controller, cache and packaging. Raw NAND is cheaper.
5. So the ratios matter more than the absolute numbers. SRAM to DRAM is roughly ten to one, and DRAM to NAND roughly two hundred to one. Those ratios have been broadly stable for two decades, and they are why the memory hierarchy has the shape it does.
6. Persistent memory history, told honestly. Intel and Micron announced 3D XPoint in July 2015. Intel shipped it as Optane DC Persistent Memory in DIMM form from April 2019 and announced the wind-down of the Optane business in July

2022. The product is discontinued. The idea is not.

7. NVDIMM-N remains available: ordinary DRAM plus flash plus a supercapacitor, copying DRAM to flash on power loss. It is byte-addressable at DRAM speed and persistent across power cuts.
8. CXL, Compute Express Link, attaches memory over a PCI Express physical layer. CXL 2.0 arrived in 2020, 3.0 in 2022 and 3.1 in late 2023. Expanders add capacity at 150 to 250 ns rather than a local DIMM’s 80 ns.

9. Persistent memory programming uses the x86 `CLWB` and `CLFLUSHOPT` instructions with `SFENCE`, and on Linux the DAX mode of ext4 and xfs with the `ndctl` and `daxctl` tools.
10. Where experts disagree: whether byte-addressable persistence will become mainstream. One camp points to Optane's commercial failure. The other points to CXL, to embedded MRAM shipping in volume, and to the argument that the software problem, not the device, was Optane's real barrier.

6.12.6 WORDS - remember these

1. Volatile — forgets without power — state maintained only while the supply rail is present.
2. Non-volatile — remembers without power — state held by trapped charge, magnetization or physical structure.
3. Memory — what the processor reaches directly — byte-addressable storage on the load-store path, latency in nanoseconds.
4. Storage — what the processor asks a device for — block-addressable media behind a controller and a driver stack.
5. Persistent memory — non-volatile but used like memory — byte-addressable non-volatile media mapped into the address space.
6. CXL — a newer way to hang memory off a processor — Compute Express Link, a cache-coherent protocol layered on PCI Express.

6.98 Common wrong ideas

1. Wrong: a latch and a flip-flop are the same thing. Right: a latch is transparent while its enable is active, a flip-flop samples only at an edge.
2. Wrong: a flip-flop stores whatever is on D while the clock is high. Right: it stores what was on D during the setup window before the edge.
3. Wrong: metastability is a bug a good engineer avoids. Right: it is a physical certainty at any asynchronous boundary, and the only defence is buying settling time to make it astronomically rare.
4. Wrong: two flip-flops in a synchronizer guarantee correctness. Right: they give a long MTBF for one bit. Multi-bit crossings still need Gray coding, a handshake or a FIFO.
5. Wrong: a faster clock always means a faster machine. Right: useful work per cycle is set by logic depth, and roughly 15 to 20 gate delays per cycle has been near constant for two decades.
6. Wrong: DRAM refresh is like topping up a battery. Right: refresh is a full read and rewrite of a row through the same sense amplifiers as a normal read, because reading DRAM destroys the stored charge.
7. Wrong: DRAM has got much faster over the years. Right: bandwidth rose about sixteenfold from DDR-400 to DDR5-6400 while CAS latency improved only from about 15 ns to about 12.5 ns.
8. Wrong: RAM is memory and an SSD is storage because one forgets and one does not. Right: the line is byte-addressable load and store versus block transfer. Persistent memory is non-volatile and is still memory.
9. Wrong: flash stores bits in tiny capacitors like DRAM. Right: flash shifts a transistor's threshold voltage using charge trapped on an isolated gate or in a nitride layer, and needs no power to hold it.
10. Wrong: SRAM is faster because it sits closer to the CPU. Right: it is faster because the value is already held by a driven inverter pair, so there is no charge sharing, sensing or restore step.

6.99 Chapter summary in 20 lines

1. A logic gate has no past. Its output depends only on its inputs right now.
2. Wiring an output back to an input creates feedback, and feedback creates two stable resting states. That is one stored bit.
3. Two cross-coupled inverters are the ancestor of every memory cell, back to the Eccles-Jordan trigger circuit of 1918.
4. Two cross-coupled NOR gates give the SR latch, with set, reset, hold and one forbidden combination.

5. Generating S and R from a single D input removes the forbidden case and gives the gated D latch, transparent while enabled.
6. Transparency is a hazard, because data can race through several stages inside one enable pulse.
7. Two latches on opposite clock phases give a master-slave D flip-flop, which samples once per edge and cannot be raced through.
8. A flip-flop needs D steady for a setup time before the edge and a hold time after it, from tens of nanoseconds in 1970s TTL to tens of picoseconds today.
9. Violating that window can leave the output hovering between 0 and 1. That is metastability, and it can be made rare but never removed.
10. MTBF rises exponentially with settling time, which is why one extra flip-flop in a synchronizer moves the answer by many orders of magnitude.
11. The clock is a square wave whose rising edge makes the whole chip take one step together, distributed through a buffered tree.
12. Clock skew, jitter and the tree's own power are the costs of marching in step, and clock gating is the first way to cut that power.
13. Flip-flops side by side make a register, chained make a shift register, and with next-value logic make a counter.
14. Synchronous counters update every bit on one edge. Ripple counters are cheaper and briefly show values nobody intended.
15. A state register plus next-state logic plus output logic is a finite state machine, Moore if outputs depend on state alone, Mealy if on inputs too.
16. A CPU control unit is exactly that, only larger, and since Wilkes proposed microprogramming in 1951 much of it has lived in microcode.
17. SRAM holds a bit in six transistors as a live inverter loop, so it is fastest and dearest per bit and is used for registers and caches.
18. DRAM holds a bit as about 69 000 electrons on a tiny capacitor. It is dense and cheap, it leaks, it must be refreshed within 64 ms for DDR4 or 32 ms for DDR5, and reading it destroys it.
19. Flash holds a bit as charge trapped on an isolated gate, placed and removed by tunnelling or hot-electron injection, surviving years without power at the cost of limited rewrite endurance.
20. Every memory is a grid of word lines and bit lines behind a decoder, n address lines reach 2 to the power n locations, and memory differs from storage by byte-addressing, not by volatility.

KB

PART B

Turning Bits Into Meaning

How switches become numbers, numbers become text,
pictures on a screen and sound in the air.

CHAPTER

7

KB

Numbers and Meaning — Binary, Text and Unicode

7.0 What this chapter gives you

1. You will be able to explain why computers count in twos, and convert any number to binary and back by two different methods.
2. You will be able to read hexadecimal on sight, and say where you meet it in real work: memory addresses, colour codes, MAC addresses, error codes.
3. You will be able to say what a byte is, why "octet" exists, and what "64-bit CPU" means about registers and address space.
4. You will be able to explain two's complement, give the exact range of any signed integer size, and name three real disasters caused by overflow.
5. You will be able to encode a decimal fraction into IEEE 754 float32 bits by hand, decode a pattern back, and say why 0.1 plus 0.2 is not 0.3.
6. You will be able to trace text encoding from Morse code through ASCII to Unicode, and write the UTF-8 bytes for an emoji by hand.
7. You will be able to explain endianness, parity, checksums, CRC32 and ECC memory, and say what each one can and cannot catch.
8. You will be able to build a Huffman tree, explain LZ77, and state Shannon entropy in one formula.
9. You will understand the deepest idea here: a pattern of bits has no meaning until something decides how to read it.

7.1 Why base 2

7.1.1 PLAIN – in simple words

1. A computer is made of switches. A switch is either on or off. There is no "half on" a machine can read back reliably.
2. So a computer can store only two marks. Call them 0 and 1.
3. To store bigger numbers, line up many switches in a row.
4. Two switches give four patterns: 00, 01, 10, 11. Three give eight. Each new switch doubles the count.
5. Counting with only two marks is called **binary**, or base 2.
6. Our normal counting uses ten marks, 0 to 9. That is base 10, or decimal, and the only reason for ten is that people have ten fingers.
7. A machine has two voltage levels, so it uses two marks.

7.1.2 PLAIN - a picture in your head

1. Picture a row of eight light switches on a wall.
2. Each has a price tag above it. From the right: 1, 2, 4, 8, 16, 32, 64, 128. Each tag is double the one to its right.
3. To show a number, flip up the switches whose tags add to it. For 5, flip up 4 and 1. That is 00000101. For 156, flip up 128, 16, 8 and 4.
4. Every number from 0 to 255 has exactly one pattern, and no number has two.
5. Reading the wall is just adding the tags of the switches that are up.

Where this comparison breaks

1. Real memory does not hold a switch position. It holds a small charge or a magnetic direction that fades and must be refreshed or rewritten.
2. Real switches are also not exactly on or off. They sit above or below a voltage threshold, and the circuit rounds them to a clean 0 or 1.

7.1.3 PLAIN - a worked example

Counting from 0 to 32 in binary, five bits then six.

Dec	Binary	Dec	Binary
0	00000	17	10001
1	00001	18	10010
2	00010	19	10011
3	00011	20	10100
4	00100	21	10101
5	00101	22	10110
6	00110	23	10111
7	00111	24	11000
8	01000	25	11001
9	01001	26	11010
10	01010	27	11011
11	01011	28	11100
12	01100	29	11101
13	01101	30	11110
14	01110	31	11111
15	01111	32	100000
16	10000		

1. Notice 32 needs a sixth bit. Five bits stop at 31.
2. N bits hold 2 to the power N patterns, from 0 to (2 to the N) minus 1.

The place values in one byte:

Bit position	Place value	Power of 2
7 (leftmost)	128	2^7
6	64	2^6
5	32	2^5
4	16	2^4
3	8	2^3
2	4	2^2

Bit position	Place value	Power of 2
1	2	2^1
0 (rightmost)	1	2^0

Method one, decimal to binary by repeated division. Convert 156.

```
156 / 2 = 78  remainder 0   <- lowest bit
 78 / 2 = 39  remainder 0
 39 / 2 = 19  remainder 1
 19 / 2 =  9  remainder 1
  9 / 2 =  4  remainder 1
  4 / 2 =  2  remainder 0
  2 / 2 =  1  remainder 0
  1 / 2 =  0  remainder 1   <- highest bit

Read the remainders bottom to top: 1 0 0 1 1 1 0 0
156 decimal = 10011100 binary
```

Method two, decimal to binary by subtraction. Same number, 156.

```
Largest place value not bigger than 156 is 128. Use it.
  156 - 128 =  28    bit 7 = 1
Is 64 <= 28? No.    bit 6 = 0
Is 32 <= 28? No.    bit 5 = 0
Is 16 <= 28? Yes.  28 - 16 = 12   bit 4 = 1
Is  8 <= 12? Yes.  12 -  8 =  4   bit 3 = 1
Is  4 <=  4? Yes.   4 -  4 =  0   bit 2 = 1
Is  2 <=  0? No.    bit 1 = 0
Is  1 <=  0? No.    bit 0 = 0

Result: 10011100
```

Binary back to decimal is easier still. Take 10011100 and add the place values of the bits that are 1: 128 + 16 + 8 + 4 = 156.

1. Both directions agree, as they must. Division scales better to large numbers; subtraction is easier to do in your head for one byte.

7.1.4 PLAIN - what is really happening inside

1. Inside a chip a 1 is a wire held near the supply voltage and a 0 is a wire held near ground. Modern logic cores run at roughly 0.7 to 1.2 volts, so the whole gap is under one volt.
2. The circuit never measures the exact voltage. It compares it with a threshold and snaps the answer to a clean 0 or 1.
3. That snapping is why binary survives noise. A wire at 0.9 volts and one at 1.05 volts are both read as 1, and both come out identical.
4. A base-10 circuit would need ten bands in the same one volt gap, about 0.1 volts each, and noise would ruin it.
5. So base 2 is not a preference. It is the widest safety margin one wire can give you.
6. The honest version: some devices do store more than two levels. NAND flash stores 3 or 4 bits per cell using 8 or 16 charge levels. It works because a memory cell is read slowly and carefully, not billions of times a second.

7.1.5 TECHNICAL - the engineer's version

1. Binary is a positional numeral system with radix 2. The digit at position i carries weight 2^i, counted from 0 at the least significant bit. An n-bit unsigned field represents integers in [0, 2^n - 1] inclusive.
2. Repeated division produces the radix-2 expansion least significant digit first. The subtraction method is greedy expansion against descending powers of 2; it is correct because binary place values are super-increasing, each power exceeding the sum of all lower powers.
3. Gottfried Wilhelm Leibniz described binary arithmetic in a 1703 paper for the Paris Academy, long before any machine used it.
4. Claude Shannon's 1937 MIT master's thesis, "A Symbolic Analysis of Relay and Switching Circuits", first tied Boolean algebra to switching hardware.
5. Noise margin is the real engineering reason for radix 2. With two levels the margin is roughly half the supply rail; with ten levels it is one eighteenth, before any process variation is counted.
6. Multi-level storage exists where read latency may be large: NAND flash SLC (1 bit per cell), MLC (2), TLC (3), QLC (4).

Bits	Patterns	Max unsigned value
4	16	15
8	256	255
10	1024	1023
16	65536	65535
32	4294967296	4294967295

7. Observe it with `python3 -c "print(bin(156))"` or `echo 'obase=2; 156' | bc`.

7.1.6 WORDS - remember these

1. Bit — one on-or-off mark — a binary digit, the smallest unit of information, one element of the set {0, 1}.
2. Binary — counting with only 0 and 1 — base 2 positional notation with place weights 2^i.
3. Base or radix — how many different digits a system uses — the number of distinct symbols per position in a positional numeral system.
4. Place value — the worth of a digit's slot — the weight r^i applied to the digit at position i in radix r.
5. Most significant bit — the leftmost, biggest-value bit — the bit with the highest positional weight, often written MSB.
6. Least significant bit — the rightmost, worth 1 — the bit of weight 2^0, often written LSB.

7.2 Hexadecimal and octal

7.2.1 PLAIN - in simple words

1. Binary is correct but painful to read. 11010110 is easy to mistype.
2. Engineers group the bits in fours and give each group one short name.
3. Four bits have sixteen patterns, so we need sixteen digit names: 0 to 9 and then A, B, C, D, E, F. A means ten, F means fifteen.
4. This is **hexadecimal**, usually shortened to hex. So 11010110 becomes 1101 and 0110, which is D and 6, written 0xD6.
5. The `0x` in front is just a label saying "the rest is hex".
6. Hex is not a different kind of number. It is a shorter way of writing the same bits.
7. There is also **octal**, which groups bits in threes and uses digits 0 to 7. It is mostly historical, but it survives in file permissions on Linux and macOS.

7.2.2 PLAIN - a picture in your head

1. Think of a long phone number with no spaces: 919876543210. Hard to read, easy to lose your place, easy to copy wrong.
2. Now group it: 91 98765 43210. Same number, far easier to handle.
3. Hex does exactly this to bits, but the grouping is fixed at four and each group gets a single character instead of a space.
4. One hex digit is always exactly four bits. Two hex digits are always exactly one byte. That rule never bends.

Where this comparison breaks

1. Phone number grouping is a display habit and can vary by country.
2. Hex grouping is arithmetic. Because 16 is 2 to the fourth power, each hex digit maps to four bits with no carrying between groups. Decimal grouping has no such property, which is why decimal to binary needs real division.

7.2.3 PLAIN - a worked example

The complete conversion table. Learn this and hex becomes automatic.

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Worked conversion, hex to binary to decimal. Take 0xB4.

```
0xB4  ->  B    4
      -> 1011 0100
      -> 10110100

Now add the place values:
  128 + 0 + 32 + 16 + 0 + 4 + 0 + 0 = 180

0xB4 = 10110100 binary = 180 decimal
```

Worked conversion, decimal to hex. Take 3000.

```
3000 / 16 = 187 remainder  8  -> digit 8
 187 / 16 =  11 remainder 11  -> digit B
  11 / 16 =   0 remainder 11  -> digit B
```

```
Read bottom to top: 0xBB8
Check: 11 x 256 + 11 x 16 + 8 = 2816 + 176 + 8 = 3000
```

Octal, three bits per digit, for Unix permissions.

```
chmod 755 script.sh

7 = 111 = read, write, execute   (owner)
5 = 101 = read, no write, execute (group)
5 = 101 = read, no write, execute (everyone else)
```

1. The three bits are, in order, read (4), write (2), execute (1).
2. So 6 is read plus write, 4 is read only, 0 is nothing.

7.2.4 PLAIN - what is really happening inside

1. The machine never stores hex. Memory holds bits and nothing else. Hex exists only where a human reads or types a value.
2. A debugger, a hex editor and an error message all convert bits to hex just before printing, and hex back to bits just after you type.
3. Hex won over decimal for this job because of alignment. Byte, nibble and word boundaries all fall on hex digit boundaries.
4. A 32-bit value is always exactly 8 hex digits and a 64-bit value always exactly 16, so you can count the digits and know the size. In decimal the same value takes 1 to 10 digits, so the width is invisible.
5. Octal survives from machines whose word sizes divided by three, such as the 12-bit PDP-8 and the 36-bit PDP-10, where four bits per digit did not fit.

7.2.5 TECHNICAL - the engineer's version

1. Hexadecimal is radix 16, octal is radix 8. Both are power-of-two radices, so conversion to and from binary is pure regrouping, never division.
2. A 4-bit group is a **nibble** (also spelled nybble). One hex digit is one nibble. Two nibbles form one octet.
3. Notation is language-specific: `0x1F` in C, C++, Java, Python, Go, Rust and JavaScript; `1Fh` or `$1F` in assemblers; `16#1F#` in Ada; `#x1F` in Lisp. Octal is `0o755` in Python 3 and Rust, and a bare leading `0` in C, which is a well-known source of bugs.
4. Where you meet hex in real work:

Context	Example value	What it is
Memory address	0x7ffee3b04a20	Pointer on x86-64
Colour	#FF8800	R=255 G=136 B=0
MAC address	00:1A:2B:3C:4D:5E	48-bit NIC address
Windows error	0x80070005	HRESULT, access denied

5. File magic numbers, the first bytes that identify a format:

Format	First bytes (hex)	As text
PNG	89 50 4E 47 0D 0A 1A 0A	.PNG....
PDF	25 50 44 46	%PDF
ELF binary	7F 45 4C 46	.ELF
ZIP / JAR	50 4B 03 04	PK..
Java class	CA FE BA BE	(none)

6. JPEG files start FF D8 FF. GIF files start 47 49 46 38, which reads "GIF8". The `file` command on Unix works mainly from a database of these patterns, held in `/usr/share/file/magic` or similar.
7. In a MAC address the first three octets are the Organizationally Unique Identifier assigned to a vendor by the IEEE, and the last three are chosen by that vendor.
8. Tools that show hex: `xxd`, `hexdump -C`, `od -t x1`, `objdump -d`, and the `x/16xb` command inside GDB.

7.2.6 WORDS - remember these

1. Hexadecimal — base 16, digits 0 to F — radix-16 positional notation, four bits per digit, standard for displaying binary data.
2. Nibble — half a byte — a 4-bit group, exactly one hexadecimal digit.
3. Octal — base 8, digits 0 to 7 — radix-8 notation, three bits per digit, used for Unix permission modes and legacy word sizes.
4. Magic number — the fingerprint at the start of a file — a fixed byte sequence at a known offset used to identify a file format.
5. Prefix 0x — a label meaning "hex follows" — a lexical marker for hexadecimal integer literals in C-family languages.

7.3 Bits, bytes and words

7.3.1 PLAIN - in simple words

1. One bit is one 0 or 1, and almost useless alone. Eight bits make one **byte**, which holds 256 different values.
2. A byte is the smallest chunk memory hands out. You cannot ask for "bit number 5" directly; you ask for the byte and pick the bit yourself.
3. A byte is enough for one English letter, or a number from 0 to 255.
4. A **word** is how many bits the processor works on at once. Old processors used 8-bit words, then 16, then 32. Today it is 64.
5. "A 64-bit computer" means the processor handles 64 bits at a time and can point at a huge amount of memory.
6. Networking people say **octet** instead of byte, because "byte" has not always meant exactly eight bits.

7.3.2 PLAIN - a picture in your head

1. Think of a warehouse of identical boxes on numbered shelves. Each box is a byte and each shelf number is a memory address.
2. The forklift is the processor, and its fork width is the word size. A 64-bit forklift picks up eight boxes in one trip; a 32-bit one picks up four.
3. The address space is how many shelf labels the building can print. With 32-bit labels you get about 4.29 billion; with 64-bit labels, 18.4 billion billion.

Where this comparison breaks

1. A real forklift wastes no effort carrying fewer boxes. A real 64-bit CPU moving one byte still uses a full 64-bit path, and pointers get bigger, which uses more cache. Wider is not free.
2. Real memory addresses are not all backed by RAM. Large regions are mapped to devices, or to nothing at all.

7.3.3 PLAIN - a worked example

The bits inside one byte, with their weights.

```
   bit 7   6   5   4   3   2   1   0
       |   |   |   |   |   |   |   |
value 128  64  32  16   8   4   2   1
```

```
Byte  1   0   1   1   0   1   1   0   = 0xB6
      128     32  16      4   2       = 182
```

1. The largest value in one byte is 11111111, which is 255, or 0xFF.
2. Two bytes side by side make 16 bits, holding 0 to 65535.
3. Four bytes make 32 bits, holding 0 to 4,294,967,295.

Why a 32-bit machine stopped near 4 GB:

```
2^32 = 4,294,967,296 distinct addresses
Each address names one byte
So the total space named = 4,294,967,296 bytes
                         = 4 GiB exactly
```

1. That 4 GiB is not only RAM. The graphics card aperture, the BIOS ROM and other device windows are carved out of the same range.
2. That is why 32-bit Windows machines with 4 GB installed typically reported about 3.2 to 3.5 GB usable. The rest of the addresses were spoken for.

7.3.4 PLAIN - what is really happening inside

1. Inside the processor are small named storage slots called registers. On a 64-bit CPU each general register is 64 bits wide, and so is the adder, so one add instruction handles 64 bits in one step.
2. Separately, the address the CPU puts on the memory path decides how much memory it can name. These two widths are usually equal but need not be: some 8-bit chips had 16-bit addresses, some 32-bit chips 36-bit ones.
3. A pointer is a plain integer holding an address, so on a 64-bit system it occupies 8 bytes. Moving to 64-bit doubles every pointer, and programs full of pointers use noticeably more memory afterwards.
4. The honest version: no shipping x86-64 chip wires all 64 address bits. Current designs use 48, giving 256 TiB; newer server parts support 57, giving 128 PiB. The unused top bits must copy the highest used bit, a rule called canonical form.

7.3.5 TECHNICAL - the engineer's version

1. The term **byte** was coined by Werner Buchholz at IBM in July 1956 during the Stretch project, which shipped as the IBM 7030. The spelling was deliberately changed from "bite" so it could not be misread as "bit".
2. A byte was not always 8 bits. The CDC 6600 used 60-bit words with 6-bit characters. The PDP-10 used 36-bit words with programmable byte sizes of 6, 7, 8 or 9 bits. Some early machines used 9-bit bytes.
3. Because of that history, IETF and ITU specifications say **octet** when they mean exactly 8 bits. RFC documents use octet throughout for this reason.
4. The 8-bit byte became universal with the IBM System/360, announced 7 April 1964, which fixed 8-bit bytes and 32-bit words.
5. Word size across the generations:

Word	Example chip	Year
8-bit	Intel 8080	1974
16-bit	Intel 8086	1978
32-bit	Intel 80386	1985
64-bit	DEC Alpha 21064	1992
64-bit x86	AMD Opteron	2003

6. Address space arithmetic:

Address bits	Bytes addressable	Common name
16	65,536	64 KiB
32	4,294,967,296	4 GiB
48	281,474,976,710,656	256 TiB
64	1.8446744 x 10^19	16 EiB

7. Physical Address Extension (PAE), introduced with the Intel Pentium Pro in 1995, widened physical addresses to 36 bits, allowing 64 GiB of RAM on a 32-bit machine. Each individual process was still limited to a 4 GiB virtual address space, so it helped servers running many processes and did little for a single large application.
8. x86-64 canonical addressing sign-extends bit 47 into bits 48 to 63. Intel 5-level paging, shipped in Ice Lake server parts from 2019 and supported in Linux since kernel 4.14, extends this to 57 bits.
9. Observe with `getconf LONG_BIT`, `uname -m`, `lscpu` (look for "Address sizes"), and `cat /proc/cpuinfo | grep "address sizes"` on Linux.

7.3.6 WORDS - remember these

1. Byte — eight bits — the smallest individually addressable unit of memory on virtually all modern architectures.
2. Octet — a guaranteed eight bits — the term used in IETF and ITU standards because "byte" was historically machine-dependent.
3. Word — the chunk size a CPU works on naturally — the native register and datapath width of an architecture.
4. Register — a slot inside the processor — the fastest storage in the machine, named rather than addressed, typically 64 bits wide today.
5. Address space — the set of memory locations that can be named — the range of addresses expressible in the architecture's pointer width.
6. Canonical address — a valid pointer shape on x86-64 — an address whose bits 63 to 48 are all copies of bit 47.

7.4 Negative numbers

7.4.1 PLAIN - in simple words

1. A byte holds a pattern of 0s and 1s. Nothing in it says "minus", so we must agree on a rule reserving some patterns for negatives.
2. The first idea was to use the leftmost bit as a minus sign. That is **sign-magnitude**. It gives two zeros, plus zero and minus zero, and needs special rules for adding mixed signs.
3. The second idea was to flip every bit to negate. That is **one's complement**. It also gives two zeros and needs an extra carry step.
4. The third idea won everything: flip every bit, then add 1. That is **two's complement**. There is exactly one zero, and normal addition just works, because the circuit never needs to know the signs.
5. Every processor you will use today stores signed integers this way.
6. The cost is a lopsided range. In one byte you get -128 to +127, one more negative than positive.

7.4.2 PLAIN - a picture in your head

1. Picture a clock face with 256 marks instead of 12. Going past the top wraps to the start, so 255 plus 1 is 0.
2. Now agree that marks past halfway are read as negatives. The mark we would call 255 we call -1; 254 becomes -2.
3. Under that reading, "add -1" and "add 255" are the same movement. The hand lands in the same place either way.

4. That is the whole trick: subtracting is adding, if you go far enough forward around the dial. The circuit never thinks about signs. It steps around and throws away whatever falls off the top.

Where this comparison breaks

1. A clock hand moving past 12 is harmless. A number wrapping past its top is often a serious bug, because the program keeps running with a value that is wildly wrong rather than stopping.
2. Also, a clock has no "unsigned versus signed" reading. In a computer, the same bits are read one way or the other depending on the declared type, and mixing the two in one expression causes real defects.

7.4.3 PLAIN – a worked example

Making -5 in eight bits, by all three methods.

```
Start with +5:            00000101

SIGN-MAGNITUDE
  set the top bit:        10000101   (reads as "minus five")
  problem: 10000000 is "minus zero", a second zero

ONE'S COMPLEMENT
  flip every bit:         11111010
  problem: 11111111 is "minus zero", a second zero

TWO'S COMPLEMENT
  flip every bit:         11111010
  add one:                11111011   <- this is -5
  as an unsigned byte that pattern is 251
```

Now check that plain addition works with no special cases.

```
    00000101      (+5)
  + 11111011      (-5)
  ----------
  1 00000000

The 1 falls off the left end and is discarded.
What remains is 00000000, which is zero. Correct.
```

Wraparound at the top of the range.

```
    01111111      (+127, the largest signed byte)
  + 00000001      (+1)
  ----------
    10000000      which as signed is -128, not +128
```

1. Adding 1 to the biggest positive number gives the most negative number.
2. This is **overflow**. The answer did not fit, so it wrapped.

The exact ranges you should know by heart:

Bits	Unsigned max	Signed min	Signed max
8	255	-128	127
16	65,535	-32,768	32,767
32	4,294,967,295	-2,147,483,648	2,147,483,647
64	1.8446744e19	-9.2233720e18	9,223,372,036,854,775,807

1. The 64-bit unsigned maximum written out is 18,446,744,073,709,551,615.

2. The 64-bit signed minimum written out is -9,223,372,036,854,775,808.

7.4.4 PLAIN - what is really happening inside

1. The adder inside a CPU adds two n-bit patterns and produces an n-bit result plus a carry-out bit, the same way every time.
2. To compute A minus B, the CPU negates B and adds. Many designs feed the inverted B into the adder with carry-in set to 1, performing "flip and add one" in a single pass.
3. So one adder does both jobs. That is the real reason two's complement won: fewer transistors.
4. The CPU sets status flags after each arithmetic instruction, and there are two separate overflow signals:
    1. The carry flag says the result did not fit as an unsigned number.
    2. The overflow flag says it did not fit as a signed number.
5. The processor does not know which reading you intended. It sets both and leaves the program to check. Most programs never check, and the wrong value continues silently.
6. To test whether a pattern is negative, look at the top bit alone. To read the magnitude of a negative, apply the same operation again: flip and add one to 11111011 gives 00000101, which is 5.

7.4.5 TECHNICAL - the engineer's version

1. Two's complement represents value v in n bits as the residue v mod 2^n. Interpretation: the MSB carries weight -2^(n-1), all other bits carry the usual positive weights.
2. Range is [-2^(n-1), 2^(n-1) - 1]. The asymmetry is unavoidable: 2^n patterns cannot be split evenly around zero if zero takes one pattern.
3. Consequence to remember: negating the minimum value overflows. In 32-bit arithmetic, `-INT_MIN` is `INT_MIN`, and `abs(-2147483648)` returns -2147483648. This is a genuine source of security bugs.
4. In C and C++, signed integer overflow is **undefined behaviour**. Unsigned overflow is defined and wraps modulo 2^n. Compilers exploit the undefined case for optimization, so an overflowing signed loop can be transformed in ways that surprise you. In Java, signed overflow is defined to wrap. In Rust, it panics in debug builds and wraps in release builds by default.
5. Historical use: the EDSAC (1949) used two's complement. The IBM System/360

(1964) used it for binary integers and made it the mainstream choice. One's complement persisted in the CDC 6600 (1964) and UNIVAC 1100 series; sign-magnitude persisted in the IBM 7090 series.

Three real overflow failures.

6. **Year 2038.** Unix time is seconds since 00:00:00 UTC on 1 January 1970, traditionally stored in a signed 32-bit `time_t`. The last representable second is 2,147,483,647, which is 03:14:07 UTC on 19 January 2038. The next increment wraps to -2,147,483,648, which reads as 20:45:52 UTC on 13 December 1901. The fix is a 64-bit `time_t`; Linux has supported 64-bit `time_t` on 32-bit architectures since kernel 5.6 (2020). The bug already bites early: AOLserver crashed in May 2006 on a timeout one billion seconds in the future, and a Microsoft Exchange antimalware update failed in January 2022 when a version number was read as a Unix timestamp.
7. **YouTube, 2 December 2014.** The view counter for the video "Gangnam Style" reached 2,147,483,647, the maximum signed 32-bit integer. Google's public statement said the company "never thought a video would be watched in numbers greater than a 32-bit integer". The counter was moved to 64-bit, which raises the ceiling to 9,223,372,036,854,775,807. Note the honest caveat: press coverage described the site as "breaking", but what the public saw was largely a display Easter egg, and reports differ on exactly when the underlying storage was widened.
8. **Ariane 5 Flight 501, 4 June 1996.** About 37 seconds after main engine ignition, both Inertial Reference Systems failed within milliseconds of each other. The cause was a conversion of the horizontal bias value, BH, from a 64-bit floating point number to a 16-bit signed integer. The value exceeded 32,767 and raised an Operand Error that was not handled. The code was Ariane 4 alignment software kept running after liftoff "for commonality reasons" although it served no purpose on Ariane 5. The launcher broke

up and was destroyed about 39 seconds after ignition. The inquiry board was chaired by Professor Jacques-Louis Lions and reported on 19 July 1996. The financial loss is widely reported as roughly 370 million US dollars; the inquiry report itself states no figure, so treat that number as approximate.
9. Detection tools: `-fsanitize=signed-integer-overflow` and `-fsanitize=unsigned-integer-overflow` in Clang and GCC, `__builtin_add_overflow` in GCC and Clang, `checked_add` in Rust, `Math.addExact` in Java.

7.4.6 WORDS - remember these

1. Two's complement — flip the bits and add one to negate — the standard signed integer encoding where the MSB carries weight -2^(n-1).
2. One's complement — flip the bits to negate — an older encoding with two representations of zero and end-around carry addition.
3. Sign-magnitude — a separate sign bit plus the plain value — an encoding with two zeros, still used for the sign bit in IEEE 754 floats.
4. Overflow — the answer was too big to fit — the condition where an arithmetic result falls outside the representable range of the type.
5. Wraparound — the number rolls over the top back to the bottom — modular reduction of the result by 2^n, defined for unsigned types in C.
6. Unix time — seconds counted since the start of 1970 — POSIX `time_t`, seconds since the epoch 1970-01-01T00:00:00Z, ignoring leap seconds.

7.5 Fractions: fixed point and floating point

7.5.1 PLAIN - in simple words

1. Bits count whole things. To store 3.75 you need an agreement about where the point goes.
2. The simple way is **fixed point**: decide once that the last two digits are after the point, and never change it. Storing money in paise or cents is fixed point. You store 1075 and everyone knows it means 10.75.
3. Fixed point is exact and fast, but the range is narrow. One setting must cover both tiny and huge values, and it cannot.
4. The other way is **floating point**: store the digits, and store separately where the point sits. That is scientific notation, like 6.02 times 10 to the 23. Two small numbers describe one enormous one.
5. Floating point gives a gigantic range at a price. Most values are stored slightly wrong, rounded to the nearest one the format can hold.
6. The rounding is tiny but it accumulates, and it is why 0.1 plus 0.2 does not come out as exactly 0.3.

7.5.2 PLAIN - a picture in your head

1. Imagine a metre ruler marked every millimetre. That is fixed point: every measurement is a whole number of millimetres, every gap the same size. But you cannot measure a bacterium and you cannot measure a road.
2. Now imagine an instrument that reports "3 point something, times ten to the power something".
3. Near 1 it can tell 1.001 from 1.002. Near a million it can only tell 1,000,000 from 1,000,001,000. The steps grow with the number.
4. That is floating point. Precision is relative, not absolute. You get about the same number of significant digits at every scale.

Where this comparison breaks

1. A ruler has no equivalent of infinity, of "not a number", or of a negative zero. Floating point has all three, and they behave in specific ways.
2. And a ruler's marks are evenly spaced everywhere. Floating point steps are even only within one power of two, and double at every power boundary.

7.5.3 PLAIN - a worked example

The layout of a 32-bit float, called binary32 or float32.

```
bit 31   bits 30..23        bits 22..0
+-----+---------------+---------------------------+
|  S  |   exponent    |         fraction          |
+-----+---------------+---------------------------+
  1 bit    8 bits              23 bits

value = (-1)^S  x  1.fraction  x  2^(exponent - 127)
```

1. S is the sign. 0 means positive, 1 means negative.
2. The exponent field stores the real exponent plus 127. That offset is called the **bias**, and it lets the field stay an unsigned number.
3. The fraction stores only the digits after the point. The leading 1 is not stored, because a normalized binary number always starts with 1. That free bit is called the **hidden bit** or implicit leading one.

Encoding 0.15625 step by step.

```
Step 1: convert 0.15625 to binary by repeated doubling.
  0.15625 x 2 = 0.3125  -> digit 0
  0.3125  x 2 = 0.625   -> digit 0
  0.625   x 2 = 1.25    -> digit 1, keep 0.25
  0.25    x 2 = 0.5     -> digit 0
  0.5     x 2 = 1.0     -> digit 1, keep 0.0, stop

  0.15625 = 0.00101 in binary

Step 2: normalize to 1.something
  0.00101 = 1.01 x 2^-3

Step 3: bias the exponent
  -3 + 127 = 124 = 01111100

Step 4: take the fraction bits after the leading 1
  01 followed by 21 zeros
  = 01000000000000000000000

Step 5: assemble
  S=0  exp=01111100  frac=01000000000000000000000

  0 01111100 01000000000000000000000
  = 0011 1110 0010 0000 0000 0000 0000 0000
  = 0x3E200000
```

Encoding -6.25 step by step.

```
Step 1: 6 = 110 in binary. 0.25 = 0.01 in binary.
  6.25 = 110.01

Step 2: normalize
  110.01 = 1.1001 x 2^2

Step 3: bias
  2 + 127 = 129 = 10000001

Step 4: fraction after the leading 1
  1001 followed by 19 zeros
  = 10010000000000000000000
```

```
Step 5: sign is 1 because the number is negative
  1 10000001 10010000000000000000000
  = 1100 0000 1100 1000 0000 0000 0000 0000
  = 0xC0C80000
```

Decoding 0x41200000 back to a number.

```
0x41200000 = 0100 0001 0010 0000 0000 0000 0000 0000

  sign     = 0                      -> positive
  exponent = 10000010 = 130         -> 130 - 127 = 3
  fraction = 01000000000000000000000
           = 0.25 (only the 2^-2 bit is set)

  value = +1.25 x 2^3 = 1.25 x 8 = 10.0
```

1. All three results can be confirmed in one line of Python: `import struct; struct.pack('>f', 0.15625).hex()` prints `3e200000`.

7.5.4 PLAIN - what is really happening inside

1. When the exponent field is neither all zeros nor all ones, the value is **normal** and the hidden leading 1 applies. That is the ordinary case.
2. When the exponent field is all zeros, the hidden bit is 0 instead of 1 and the exponent is treated as -126. These are **subnormal** or denormal numbers, and they fill the tiny gap between the smallest normal number and zero. Without them, the gap around zero would be larger than the gap between the smallest two normal numbers, which breaks the rule that "a minus b is zero only if a equals b".
3. When the exponent field is all zeros and the fraction is also all zeros, the value is zero. The sign bit still applies, so there is a positive zero and a **negative zero**. They compare as equal, but 1 divided by +0 gives positive infinity and 1 divided by -0 gives negative infinity.
4. When the exponent field is all ones and the fraction is all zeros, the value is **infinity**, positive or negative by the sign bit.
5. When the exponent field is all ones and the fraction is not zero, the value is **NaN**, meaning not a number. It is produced by 0 divided by 0, by infinity minus infinity, and by the square root of a negative number.
6. NaN has a strange and useful property: it is not equal to anything, including itself. `x != x` is a valid test for NaN, and many libraries implement `isnan` exactly that way.
7. Now the famous problem. In binary, 0.1 is a repeating fraction, exactly the way 1/3 is repeating in decimal. It is 0.0001100110011001100... forever.
8. The format has to stop somewhere, so it stores the nearest value it can. The stored value is very slightly too big.
9. 0.2 is likewise stored slightly too big. Their sum, rounded again, lands on a value slightly above the nearest stored value of 0.3.
10. So the comparison fails, not because the machine is broken, but because it is doing exactly what a finite binary format must do.

7.5.5 TECHNICAL - the engineer's version

1. IEEE 754 was approved in 1985. William Kahan of UC Berkeley led the effort; the initial proposal was drafted in 1978 by Kahan, Jerome Coonen and Harold Stone. Kahan received the ACM Turing Award in 1989. The Intel 8087 coprocessor shipped in 1980 implementing the draft, five years before approval. The standard was revised as IEEE 754-2008 and IEEE 754-2019, and adopted internationally as ISO/IEC 60559.
2. Format parameters:

Format	S / E / M bits	Bias	Approx decimal digits
binary16	1 / 5 / 10	15	3.3
bfloat16	1 / 8 / 7	127	2.4
binary32	1 / 8 / 23	127	7.2
binary64	1 / 11 / 52	1023	15.9

3. Ranges and step sizes:

Format	Largest finite	Machine epsilon
binary16	65,504	2^-10 = 9.77e-4
bfloat16	3.39e38	2^-7 = 7.81e-3
binary32	3.4028235e38	2^-23 = 1.1921e-7
binary64	1.7976931e308	2^-52 = 2.2204e-16

4. **Machine epsilon** is the distance from 1.0 to the next representable value above it. It equals 2^-p where p is the number of stored fraction bits. It is the correct constant to use in a relative tolerance test, not an arbitrary value like 0.000001.
5. The exact stored values behind the famous example, in binary64:

```
0.1 stores as
  0.1000000000000000055511151231257827021181583404541015625
0.2 stores as
  0.200000000000000011102230246251565404236316680908203125
their sum rounds to
  0.300000000000000044408920985006261616945266723632812 5
but the nearest double to 0.3 is
  0.29999999999999998889776975374843459576368331909179 6875

so (0.1 + 0.2) == 0.3 is false, and 0.1 + 0.2 prints as
0.30000000000000004
```

6. Rounding modes defined by the standard: round to nearest with ties to even (the default), round toward zero, round toward positive infinity, round toward negative infinity, and round to nearest with ties away from zero. Ties-to-even avoids the small upward bias that ties-away introduces.
7. Money must not use binary floating point. Reasons: 0.01 is not exactly representable, sums are order-dependent, and financial rules require specific decimal rounding. Use one of:
   1. Integers in minor units, such as paise or cents, with explicit scaling.
   2. A decimal type: `decimal.Decimal` in Python, `BigDecimal` in Java, `System.Decimal` in .NET, `NUMERIC` or `DECIMAL` in SQL.
   3. The IEEE 754-2008 decimal formats decimal32, decimal64 and decimal128, which are hardware-supported on IBM POWER and z/Architecture.
8. **bfloat16** was developed at Google Brain and used in Tensor Processing Units from TPU v2 onward. It is binary32 with the low 16 fraction bits removed, so conversion to and from float32 is a truncate or a zero-fill. Established fact: it keeps float32's exponent range, so it does not underflow on the very small gradient values that appear in deep networks, which is where binary16 struggles and needs loss scaling. Hardware support is now broad, including Intel Cooper Lake (2020), Armv8.6-A, and NVIDIA GPUs from the Ampere generation (2020).
9. Active research rather than settled practice: 8-bit formats. The E4M3 and E5M2 shapes proposed jointly by NVIDIA, Arm and Intel in 2022, and now carried in the Open Compute Project microscaling specifications, are in production use for inference and increasingly for training, but best practice is still moving. Treat any specific claim about FP8 training quality as version-dependent and dated.

10. Inspect bit patterns with `struct.pack` in Python, `printf "%a"` in C (which prints hexadecimal floating point), `frexp` and `ldexp` in the C library, or `numpy.float32(x).view(numpy.uint32)`.

7.5.6 WORDS - remember these

1. Fixed point — the point never moves — an integer with an implicit constant scale factor, usually a power of two or ten.
2. Floating point — the point moves with the exponent — a sign, an exponent and a significand encoding a value as significand times base^exponent.
3. Mantissa or significand — the digits of the number — the fractional field plus the implicit leading bit for normal values.
4. Bias — a fixed offset added to the exponent — the constant 2^(k-1) - 1 that lets a k-bit exponent field stay unsigned.
5. Subnormal — very small numbers below the normal range — values with a zero exponent field and no implicit leading one, giving gradual underflow.
6. NaN — a result that is not a number — an exponent field of all ones with a non-zero significand, unequal to every value including itself.
7. Machine epsilon — the smallest step above 1.0 — 2^-p for p stored fraction bits; 1.19e-7 for binary32 and 2.22e-16 for binary64.

7.6 Text before Unicode

7.6.1 PLAIN - in simple words

1. A computer stores numbers. To store letters, someone must publish a table saying which number means which letter.
2. That table is called a **character encoding**. It is only an agreement.
3. The first such agreements were for telegraphs, not computers.
4. Morse code gave each letter a pattern of short and long signals, with the common letters kept short. E is one dot.
5. Baudot code gave every letter exactly five on-off marks, so machines could handle it without a human ear.
6. IBM built its own table, EBCDIC, for its big machines.
7. Then in 1963 a committee published **ASCII**, which used 7 bits and 128 codes, and it slowly won.
8. ASCII covered English and nothing else. There was no space for accents, no Devanagari, no Chinese, no Arabic.
9. So every country invented its own extension using the spare eighth bit. Those extensions were called **code pages**.
10. A file did not say which code page it used. Open it with the wrong one and you got garbage. That mess is what Unicode was created to end.

7.6.2 PLAIN - a picture in your head

1. Imagine a phone keypad where each button number stands for a letter, and two friends each keep their own written key.
2. If both keys are identical, messages arrive fine.
3. If one friend swaps a few entries, most of the message still reads, but some words come out as nonsense.
4. Now imagine a hundred friends with a hundred slightly different keys, and no way to write on the envelope which key you used.
5. That is exactly the code page era. Text arrived as bare numbers with no label saying how to read them.

Where this comparison breaks

1. In the keypad picture, both sides know a key exists. In the real problem, most software silently assumed its own local default and never asked.
2. And guessing was often close enough to look right, which was worse than failing loudly, because the corruption was saved back to disk.

7.6.3 PLAIN - a worked example

The clever parts of the ASCII layout.

Character	Decimal	Hex	Binary (7 bit)
space	32	0x20	0100000
'0'	48	0x30	0110000
'9'	57	0x39	0111001
'A'	65	0x41	1000001
'Z'	90	0x5A	1011010
'a'	97	0x61	1100001
'z'	122	0x7A	1111010

1. The digits sit at 0x30 to 0x39, so the low four bits are the digit value. To turn the character '7' into the number 7, mask with 0x0F.
2. 'A' is 65 and 'a' is 97. The difference is 32, which is a single bit, bit 5.

```
'A' = 1000001
'a' = 1100001
       ^
       this one bit is the whole difference

To lowercase:  c OR  0x20
To uppercase:  c AND 0xDF
To flip case:  c XOR 0x20
```

3. That is why case conversion on ASCII is one instruction, not a lookup.
4. Control characters occupy 0 to 31. They line up with letters too. The Ctrl key clears bits 6 and 5, so Ctrl-I is 9 (tab), Ctrl-M is 13 (carriage return), Ctrl-J is 10 (line feed), Ctrl-C is 3 (end of text).
5. DEL is 127, all seven bits set. On paper tape you deleted a character by punching every hole, which no other code could be mistaken for.
6. Uppercase comes before lowercase, and digits before both, so sorting by byte value roughly matches alphabetical order for plain English.

7.6.4 PLAIN - what is really happening inside

1. A text file is a sequence of bytes. There is no letter anywhere in it.
2. When you open the file, a program picks a decoding table, looks up each byte, and asks a font for a shape to draw.
3. Three separate things are involved and people confuse them constantly:
    1. The **code**, meaning which number stands for which character.
    2. The **encoding**, meaning how that number is packed into bytes.
    3. The **font**, meaning what the shape looks like on screen.
4. For 7-bit ASCII the code and the encoding are almost the same thing, because every character fits in one byte with a spare top bit.
5. That spare top bit is where the trouble started. It doubled the table from 128 to 256 entries, and nobody agreed on the second half.
6. Russian text encoded in KOI8-R and read as Windows-1252 produces Latin letters and punctuation. Nothing crashes. The bytes are all valid. Only the meaning is lost.

7. The honest version: some code pages were also **stateful**, using shift codes to switch between banks of characters. Baudot did this with LTRS and FIGS, and some Asian encodings did it with escape sequences. In a stateful encoding you cannot decode a byte without knowing the state, so you cannot start reading in the middle of a file.

7.6.5 TECHNICAL - the engineer's version

1. Morse code was developed by Samuel Morse and Alfred Vail through the late 1830s; the famous line "What hath God wrought" was sent on 24 May 1844. It is a variable-length code with frequency-weighted symbols, an early instance of the idea Huffman later formalized. It is not binary: it has dot, dash, and three distinct gap lengths.
2. Baudot code was devised by Emile Baudot around 1870 and patented in 1874. Five bits give 32 combinations, extended by LTRS and FIGS shift codes. Donald Murray revised it in 1901; the result became International Telegraph Alphabet No. 2, standardized by the CCITT in 1930. The unit "baud" is named after Baudot.
3. EBCDIC (Extended Binary Coded Decimal Interchange Code) was created by IBM for the System/360, announced 7 April 1964. It is 8-bit and derived from punched-card BCD. Its letters are not contiguous: A to I occupy 0xC1 to 0xC9, J to R occupy 0xD1 to 0xD9, S to Z occupy 0xE2 to 0xE9. Code that tests `c >= 'A' && c <= 'Z'` is wrong on EBCDIC.
4. ASCII was published as ASA X3.4-1963 on 17 June 1963 by the American Standards Association. Work began at the first meeting of the X3.2 subcommittee on 6 October 1960. Bob Bemer was a leading contributor. Revisions followed as USAS X3.4-1967 and ANSI X3.4-1986. The international equivalents are ECMA-6 and ISO/IEC 646.
5. Common 8-bit code pages:

Encoding	Year	Covers
IBM CP437	1981	IBM PC, box drawing
ISO 8859-1	1987	Western Europe
Windows-1252	1990s	Latin-1 plus 0x80-0x9F
KOI8-R	RFC 1489, 1993	Russian Cyrillic
Shift-JIS	1982	Japanese
ISCII (IS 13194)	1991	Indian scripts

6. ISO 8859-15 arrived in 1999 mainly to add the euro sign, which ISO 8859-1 had no room for. That single missing character forced a new standard, which tells you how badly the fixed 256-slot model had run out.
7. Windows-1252 is not ISO 8859-1, although it is often labelled as such. It places printable characters in the C1 control range 0x80 to 0x9F, including the curly quotes and the em dash. Mislabelled Windows-1252 content is the single most common source of stray question marks on the web.
8. Tools: `iconv -f WINDOWS-1252 -t UTF-8`, `file -i`, `chardet` and `charset-normalizer` in Python, `enca`.

7.6.6 WORDS - remember these

1. Character encoding — the table linking numbers to letters — a mapping from a character repertoire to byte sequences.
2. ASCII — the 1963 English 7-bit table — ASA X3.4-1963, 128 code positions, 33 control characters and 95 printable characters.
3. Code page — a national extension of the top 128 byte values — a vendor or national single-byte character set covering 0x80 to 0xFF.
4. Control character — a code that means an action, not a shape — ASCII 0x00 to 0x1F plus 0x7F, used for framing, flow control and terminals.

5. EBCDIC — IBM's rival table — an 8-bit punched-card-derived encoding used on IBM mainframes, with non-contiguous alphabetic ranges.
6. Stateful encoding — you must know what mode you are in — an encoding whose byte meanings depend on preceding shift or escape sequences.

7.7 Unicode and UTF-8

7.7.1 PLAIN - in simple words

1. Unicode's idea is simple: give every character in every writing system its own permanent number, once, for everyone.
2. That number is called a **code point**. It is written as U+ then hex, for example U+0041 for the letter A.
3. Unicode assigns the numbers. It does not say how to store them in bytes. That is a separate decision.
4. There are three main storage schemes: UTF-8, UTF-16 and UTF-32.
5. **UTF-8** uses one to four bytes per character. English text takes one byte per letter, exactly like ASCII.
6. That backward compatibility is why UTF-8 took over the world. Every old ASCII file was already a valid UTF-8 file, unchanged.
7. **UTF-16** uses two bytes for most characters and four for the rest.
8. **UTF-32** uses four bytes for everything, always.
9. Once you use these, the old question "which code page is this file?" disappears. There is one table for the whole planet.

7.7.2 PLAIN - a picture in your head

1. Think of a global postal system where every building on Earth gets one permanent number, and nobody ever reuses a number.
2. Unicode is that numbering scheme. U+0915 always means the Devanagari letter ka, in every country, forever.
3. Now think of how you write that number on an envelope. You could always use a fixed seven-digit box, wasting space on short numbers.
4. Or you could use a variable scheme where short numbers get short writing and long numbers get more digits, with a marker telling the reader how many digits to expect.
5. That second scheme is UTF-8. The first few bits of the first byte announce the length of the whole character.

Where this comparison breaks

1. Postal numbers name one thing each. A visible character on screen is often several code points glued together, and the glue is itself a code point.
2. And postal numbers are never reordered. Unicode has to handle text that runs right-to-left, and combining marks that attach to the character before them, so visual order and stored order differ.

7.7.3 PLAIN - a worked example

The UTF-8 rules in one table.

Code point range	Bytes	Bit template
U+0000 to U+007F	1	0xxxxxxx
U+0080 to U+07FF	2	110xxxxx 10xxxxxx
U+0800 to U+FFFF	3	1110xxxx 10xxxxxx 10xxxxxx
U+10000 to U+10FFFF	4	11110xxx 10xxxxxx x2 more

1. The full 4-byte template is `11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`, which carries $3 + 6 + 6 + 6 = 21$ bits.

Character one: the English letter A, U+0041.

```
0x41 = 65, which is under 128, so one byte.
Result: 41
```

Character two: the Devanagari letter ka, U+0915.

```
0x915 in binary, padded to 16 bits:
  0000 1001 0001 0101

U+0915 is in the 3-byte range, template 1110xxxx 10xxxxxx 10xxxxxx
That carries 4 + 6 + 6 = 16 bits, so use all 16.

Split 0000100100010101 as  0000 | 100100 | 010101

byte 1 = 1110 0000 = 0xE0
byte 2 = 10 100100 = 0xA4
byte 3 = 10 010101 = 0x95

Result: E0 A4 95
```

Character three: the grinning face emoji, U+1F600.

```
0x1F600 in binary, padded to 21 bits:
  0 0001 1111 0110 0000 0000
= 000011111011000000000

U+1F600 is in the 4-byte range, carrying 3 + 6 + 6 + 6 = 21 bits.

Split as  000 | 011111 | 011000 | 000000

byte 1 = 11110 000 = 0xF0
byte 2 = 10 011111 = 0x9F
byte 3 = 10 011000 = 0x98
byte 4 = 10 000000 = 0x80

Result: F0 9F 98 80
```

2. Confirm any of these with `python3 -c "print('A'.encode('utf-8').hex())"`.

7.7.4 PLAIN - what is really happening inside

1. Look again at the templates. Every continuation byte starts with the bits 10. No leading byte ever starts with 10.
2. So if you drop into the middle of a UTF-8 stream, you can find a character boundary by walking backwards until you hit a byte that does not start with 10. You never scan back more than three bytes.
3. That property is called **self-synchronising**, and it is the reason UTF-8 beat the alternatives. A corrupted or truncated stream loses one character, not the rest of the file.
4. Second property: a byte below 0x80 is always a real ASCII character and never part of a multi-byte sequence.
5. That means old code searching for the byte 0x2F (a slash) or 0x00 (a string terminator) still works correctly on UTF-8 text without changes. This is what "file system safe" meant in the original name.
6. Third property: sorting UTF-8 byte strings gives the same order as sorting by code point. That is not true of UTF-16.
7. UTF-16 handles characters above U+FFFF with a pair of 16-bit units called a **surrogate pair**. Two thousand and forty-eight code points, U+D800 to U+DFFF, are permanently reserved for this and are never characters.

8. The arithmetic: subtract 0x10000 from the code point, leaving 20 bits. The top 10 bits go into 0xD800 plus that value, the bottom 10 into 0xDC00 plus that value.
9. For U+1F600: 0x1F600 minus 0x10000 is 0xF600. The high half is 0xD800 plus 0x3D, which is 0xD83D. The low half is 0xDC00 plus 0x200, which is 0xDE00.
10. This is why a JavaScript or Java string reports a length of 2 for a single emoji. Those languages count 16-bit units, not characters.

7.7.5 TECHNICAL - the engineer's version

1. Unicode 1.0 was published in October 1991, following Joe Becker's 1988 draft "Unicode 88". The Unicode Consortium was incorporated in January 1991. Unicode 2.0, in July 1996, broke the original 16-bit assumption and added surrogates. Unicode 17.0 was released on 9 September 2025 and contains 159,801 assigned characters.
2. The code space is U+0000 to U+10FFFF: 1,114,112 code points in 17 planes of 65,536 each. Plane 0 is the Basic Multilingual Plane. Plane 1 is the Supplementary Multilingual Plane, holding emoji and historic scripts. Plane 2 holds CJK extensions. Planes 15 and 16 are private use.
3. The ceiling of U+10FFFF is not a natural limit. It exists because that is the largest value UTF-16 surrogate pairs can express. UTF-8's own structure could have reached U+1FFFFF or, in the original 1992 design, six bytes and 31 bits. RFC 3629 cut it back to four bytes to match UTF-16.
4. UTF-8 was designed by Ken Thompson and Rob Pike in September 1992. Pike's account states it was worked out "on a placemat in a New Jersey diner", the scheme was described to the X/Open committee on 8 September 1992, and the whole of Plan 9 was converted within days. The original name was FSS-UTF, for File System Safe UCS Transformation Format. Pike and Thompson presented it at the USENIX Winter 1993 conference in San Diego.
5. The current specification is RFC 3629, published November 2003 by Francois Yergeau, which is also STD 63. It obsoletes RFC 2279 (1998), which obsoleted RFC 2044 (1996).
6. RFC 3629 forbids **overlong encodings**: a code point must use the shortest template that fits. Accepting overlongs was a real security hole, exploited against Microsoft IIS in 2001 to smuggle a directory-traversal slash past a filter that checked for 0x2F only.
7. Encoding sizes for one character:

Character	UTF-8	UTF-16BE	UTF-32BE
A (U+0041)	41	00 41	00 00 00 41
e-acute (U+00E9)	C3 A9	00 E9	00 00 00 E9
ka (U+0915)	E0 A4 95	09 15	00 00 09 15
emoji (U+1F600)	F0 9F 98 80	D8 3D DE 00	00 01 F6 00

8. Byte order marks. U+FEFF at the start of a stream signals encoding and byte order: FE FF for UTF-16BE, FF FE for UTF-16LE, 00 00 FE FF for UTF-32BE, FF FE 00 00 for UTF-32LE. In UTF-8 it appears as EF BB BF. The Unicode standard permits but does not recommend a UTF-8 BOM; it breaks shell scripts, PHP output, and many CSV parsers. Note the ambiguity: FF FE alone is UTF-16LE, but FF FE 00 00 is UTF-32LE, so a decoder must look further.
9. Combining characters and normalization, defined in Unicode Standard Annex #15. The character e-acute has two spellings:

```
precomposed:  U+00E9                    UTF-8: C3 A9
decomposed:   U+0065 U+0301             UTF-8: 65 CC 81

They render identically. They are not equal as byte strings
and not equal under a naive == comparison.
```

10. The four normalization forms are NFC (canonical composition), NFD (canonical decomposition), NFKC and NFKD (compatibility forms, which also fold the ligature fi to "fi" and the circled digit one to "1"). NFC is the web default; the W3C Character Model recommends it. Compatibility forms lose information and must never be used for round-tripping.
11. macOS HFS+ historically stored filenames in a variant of NFD, while Linux stores whatever bytes it is given. That mismatch broke real Git repositories containing accented or Indic filenames; `git config core.precomposeunicode true` exists for exactly this reason.
12. String length has at least four correct answers. For the four-person family emoji, built from four people joined by three zero-width joiners (U+200D):

Unit	Count
UTF-8 bytes	25
UTF-16 code units	11
Code points	7
Grapheme clusters	1

13. Grapheme cluster rules live in Unicode Standard Annex #29 and change between versions. Unicode 15.1, released September 2023, added rule GB9c so that Indic conjunct sequences such as the Devanagari cluster in "namaste" group correctly. So even "how many characters is this" depends on which Unicode version your library implements. Python's `len` counts code points, Go's `len` counts bytes, Java's `String.length` counts UTF-16 units, Swift's `count` counts grapheme clusters. Four languages, four different answers, all defensible.
14. Tools: `unicode` and `uniname` on Linux, `python3 -c "import unicodedata; print(unicodedata.name(ch))"`, `iconv`, `hexdump -C`, and the ICU library.

7.7.6 WORDS - remember these

1. Code point — the permanent number for a character — an integer in U+0000 to U+10FFFF assigned by the Unicode Standard.
2. UTF-8 — one to four bytes per character, ASCII-compatible — the variable width encoding defined by RFC 3629 and STD 63.
3. Surrogate pair — two halves standing for one big character — a high surrogate in U+D800 to U+DBFF followed by a low surrogate in U+DC00 to U+DFFF, encoding a code point above U+FFFF in UTF-16.
4. BOM — an invisible marker at the file start — U+FEFF used to signal encoding and byte order.
5. Mojibake — text turned to garbage by the wrong table — the result of decoding bytes with an encoding other than the one used to produce them.
6. Grapheme cluster — what a person calls one character — a sequence of code points treated as one unit by the rules in Unicode Annex #29.
7. Normalization — putting equal-looking text into one standard spelling — the NFC, NFD, NFKC and NFKD transformations of Unicode Annex #15.

7.8 Endianness

7.8.1 PLAIN - in simple words

1. A number bigger than one byte must be split across several bytes.
2. There are two sensible orders to write them in, and both are in use.
3. **Big-endian** writes the biggest part first, the way we write numbers.
4. **Little-endian** writes the smallest part first, backwards from how we read.
5. Neither is better. They are two conventions that both work.
6. The trouble comes when a big-endian machine sends bytes to a little-endian machine and nobody converts. The number arrives scrambled.

7. So network protocols pick one order and everyone must obey it. They picked big-endian, and it is called network byte order.
8. Nearly every computer you own runs little-endian internally, so it converts on the way out and on the way in.

7.8.2 PLAIN - a picture in your head

1. Think of writing the date. Some people write 13/08/2026, day first. Others write 2026-08-13, year first.
2. Both write the same date. Both are unambiguous once you know the rule.
3. Give a form filled in one way to a reader expecting the other, and 08/13 becomes an impossible 13th month, or worse, a plausible wrong date.
4. Endianness is the same problem at the byte level, and computers do not notice that a date is impossible. They just use the wrong number.

Where this comparison breaks

1. Date formats are ambiguous forever. Byte order is fixed by the machine and by the protocol specification, so it is always knowable.
2. Also, little-endian is not simply "backwards". It has a real advantage: the low byte sits at the lowest address, so reading a 32-bit value as an 8-bit value needs no address adjustment.

7.8.3 PLAIN - a worked example

Store the 32-bit value 0x12345678 at address 0x1000.

```
BIG-ENDIAN (most significant byte at the lowest address)

  address: 0x1000 0x1001 0x1002 0x1003
  byte:      12     34     56     78
             ^ biggest part first

LITTLE-ENDIAN (least significant byte at the lowest address)

  address: 0x1000 0x1001 0x1002 0x1003
  byte:      78     56     34     12
             ^ smallest part first
```

1. The number is identical in both. Only the layout in memory differs.
2. A real dump from a little-endian machine looks like this:

```
$ python3 -c "import sys; sys.stdout.buffer.write(
    (0x12345678).to_bytes(4,'little'))" | hexdump -C

00000000  78 56 34 12                     |xV4.|
```

3. If you send those four bytes to a big-endian reader without conversion, it reads 0x78563412, which is 2,018,915,346 instead of 305,419,896.

7.8.4 PLAIN - what is really happening inside

1. The CPU does not store a number "the wrong way round". It stores it the only way its memory interface is wired.
2. Registers have no endianness. A 64-bit register is just 64 bits. Endianness only exists when a multi-byte value touches byte-addressed memory, a file, or a wire.
3. When a program writes a 32-bit integer, the store instruction splits it and places the bytes in the architecture's fixed order.
4. When it reads it back on the same machine, the load reverses that split perfectly. So within one machine, endianness is invisible.

5. It becomes visible the moment bytes cross a boundary: a network socket, a file shared between machines, or a memory-mapped device.
6. That is why protocol code is full of conversion calls. On a little-endian host `htonl` swaps the bytes. On a big-endian host the same function does nothing at all and compiles away.
7. This is also why casting a byte buffer straight into a struct pointer is a portability bug, even though it usually works on the machine you tested on.

7.8.5 TECHNICAL - the engineer's version

1. The terms come from Danny Cohen's memo "On Holy Wars and a Plea for Peace", Internet Experiment Note 137, dated 1 April 1980, later published in IEEE Computer in October 1981. Cohen borrowed Big-Endians and Little-Endians from Jonathan Swift's Gulliver's Travels of 1726, where the dispute is over which end of a boiled egg to open.
2. Architecture byte orders:

Architecture	Byte order
x86, x86-64	Little-endian
ARM (in practice)	Little-endian
RISC-V	Little-endian
SPARC, m68k, z/Arch	Big-endian

3. ARM, PowerPC, MIPS and RISC-V are bi-endian, meaning the mode is selectable. In practice ARM Linux, Android, iOS and macOS all run little-endian, and Linux on POWER moved to little-endian with the ppc64le port around POWER8 in 2014.
4. Network byte order is big-endian, fixed by the Internet Protocol specification, RFC 791 (September 1981), and restated in RFC 1700. The conversion functions are `htons`, `htonl`, `ntohs`, `ntohl` in `<arpa/inet.h>`, plus `htobe32` and friends in `<endian.h>` on Linux.
5. File formats also pick a side. PNG, JPEG, Java class files and most internet formats are big-endian. ZIP, GIF, BMP and RIFF or WAV are little-endian. TIFF is explicitly both: the file starts with "II" for Intel little-endian or "MM" for Motorola big-endian.
6. There is also **middle-endian**, seen on the PDP-11, which stored 32-bit values as two 16-bit words in big-endian order with each word little-endian inside. It is why "PDP-endian" is a term.
7. Bit order within a byte is a separate question, handled by the hardware serializer rather than by software.
8. Observe with `lscpu | grep -i "byte order"`, `sysctl hw.byteorder` on macOS, `python3 -c "import sys; print(sys.byteorder)"`, and the C macro `__BYTE_ORDER__` in GCC and Clang.

7.8.6 WORDS - remember these

1. Endianness — which end of a number goes first in memory — the byte ordering convention for multi-byte scalar values.
2. Big-endian — biggest byte at the lowest address — most significant byte first, the order used by network protocols.
3. Little-endian — smallest byte at the lowest address — least significant byte first, used by x86-64, ARM and RISC-V.
4. Network byte order — the order the internet insists on — big-endian, fixed by RFC 791 for all IP header fields.
5. Byte swap — reversing the order — the `bswap` instruction on x86 or the `REV` instruction on ARM, one cycle in hardware.

7.9 Checking that data is intact

7.9.1 PLAIN - in simple words

1. Bits get flipped. Cables pick up noise, memory cells lose charge, disks develop weak spots, cosmic rays strike chips.
2. A flipped bit is silent. Nothing complains. The data is simply wrong.
3. So we send extra information alongside the data, computed from the data itself, and check that it still matches.
4. The simplest is a **parity bit**: one extra bit making the number of 1s even. If it comes out odd, something changed.
5. A **checksum** is slightly better: add all the bytes and send the total.
6. A **CRC** is much stronger. It uses division rather than addition, and it catches whole runs of damaged bits.
7. None of these stop a deliberate attacker, who can simply recompute the check value. For that you need a **cryptographic hash**, designed so that nobody can find a second message with the same value.
8. **ECC memory** goes further still. It stores enough extra bits to spot a single flipped bit and repair it on the fly.

7.9.2 PLAIN - a picture in your head

1. A shop counts a delivery of boxes and writes the total on the paperwork.
2. If one box goes missing, the count disagrees and the error is caught. That is a checksum.
3. If one box goes missing and someone adds a different box, the count still matches. The check passed and the delivery is still wrong.
4. Now suppose the paperwork records not just the count but the weight, the volume, and a code derived from the serial numbers in order.
5. Now swapping one box for another almost certainly disagrees somewhere. That is roughly what a CRC does.
6. And suppose there is enough redundant paperwork to work out exactly which box is missing and reorder it automatically. That is ECC.

Where this comparison breaks

1. A shop clerk can think and investigate. A CRC only ever says “matches” or “does not match”. It never says which bit changed or how to fix it.
2. And the paperwork analogy suggests deliberate fraud is hard. With a CRC it is trivially easy, because the formula is public and reversible.

7.9.3 PLAIN - a worked example

Parity on one byte.

```
Data byte:  1 0 1 1 0 1 0 0     number of 1s = 4, which is even
Even parity bit = 0   ->  sent as  10110100 0

Now flip one bit in transit:
Received:   1 0 1 1 0 1 1 0  0   number of 1s = 5, which is odd
The parity bit says it should be even -> error detected

Now flip two bits:
Received:   1 0 1 1 1 1 1 0  0   number of 1s = 6, still even
Parity says fine. The error is missed.
```

1. Parity detects any odd number of flipped bits and misses every even number.
2. It can never repair anything, because it does not know which bit moved.

A real CRC32 value you can reproduce:

```
$ python3 -c "import zlib; print(hex(zlib.crc32(b'123456789')))"
0xcbf43926
```

3. That value, 0xCBF43926 for the nine ASCII digits "123456789", is the standard published check value for CRC-32. Any correct implementation produces it. It is the first thing to test when you write one.

7.9.4 PLAIN - what is really happening inside

1. A CRC treats the whole message as one enormous binary number.
2. It divides that number by a fixed constant and keeps only the remainder. The remainder is the CRC value.
3. The division is not ordinary division. It uses XOR instead of subtraction, so there are no carries and no borrows.
4. In practice you do it by shifting the message through a register, and every time a 1 falls off the top you XOR the register with the constant.
5. The constant is called the **polynomial**, because mathematicians describe the same operation as polynomial division over the field with two elements. The name is historical; you can implement it knowing only XOR and shift.
6. The strength comes from that division. Adding bytes lets errors cancel out. Division spreads every input bit's influence across the whole remainder.
7. ECC memory works differently. It computes several overlapping parity bits, each covering a different subset of the data bits.
8. When a bit flips, the pattern of which parity checks fail points directly at the position of the flipped bit. Flip it back and the data is repaired.
9. This is a Hamming code. Adding one more overall parity bit lets it also detect, without correcting, a double flip. That combination is called SECDED, single error correct, double error detect.

7.9.5 TECHNICAL - the engineer's version

1. Parity is used in UART serial framing, written as 8N1 for eight data bits, no parity, one stop bit, or 7E1 for seven data bits, even parity, one stop bit. Hamming distance 2: it detects one error, corrects none.
2. The Internet checksum, specified in RFC 1071, is the 16-bit one's complement of the one's complement sum of 16-bit words. It protects the IPv4 header, TCP and UDP. It is weak: it cannot detect a reordering of 16-bit words, and it misses many pairs of compensating errors.
3. The CRC idea was published by W. Wesley Peterson in 1961. CRC-32 as used in Ethernet (IEEE 802.3), PNG, gzip and ZIP uses the polynomial 0x04C11DB7 in normal notation, or 0xEDB88320 reflected, with an initial value of all ones and a final XOR of all ones.
4. Guaranteed properties of a 32-bit CRC with a well-chosen polynomial:
    1. Detects all single-bit and all double-bit errors within its bound.
    2. Detects any odd number of bit errors, if the polynomial includes the factor $(x + 1)$.
    3. Detects all burst errors up to 32 bits long.
    4. Misses longer bursts with probability 2^-32, about one in 4.3 billion.
5. Common CRC variants:

Name	Width	Polynomial	Used in
CRC-32	32	0x04C11DB7	Ethernet, PNG, zip
CRC-32C	32	0x1EDC6F41	iSCSI, ext4, Btrfs
CRC-16-CCITT	16	0x1021	XMODEM, Bluetooth
CRC-8	8	0x07	SMBus, 1-Wire

6. CRC-32C has a dedicated x86 instruction, `crc32`, added with SSE4.2 in the Intel Nehalem generation (2008), which is why modern filesystems chose it.
7. Cryptographic hashes solve a different problem: collision resistance against an adversary. Named here and covered properly in the security chapter: MD5 (1992, collisions demonstrated in 2004), SHA-1 (1995, collision demonstrated by Google and CWI Amsterdam in February 2017, the SHAttered attack), the SHA-2 family including SHA-256 (2001), SHA-3 based on Keccak (standardized 2015), and BLAKE3 (2020). A CRC is not a hash and must never be used where tampering is possible.
8. ECC DRAM adds 8 check bits per 64 data bits, so a registered ECC DIMM is 72 bits wide rather than 64. Standard SECDED corrects one bit per 64-bit word and detects two. Chipkill, called Advanced ECC or SDDC by different vendors, arranges the code so that a whole failed DRAM device is correctable. The underlying theory is Richard Hamming's 1950 paper "Error Detecting and Error Correcting Codes" in the Bell System Technical Journal.
9. DDR5, specified in JEDEC JESD79-5 published July 2020, mandates on-die ECC inside the DRAM chip. Be careful: on-die ECC protects the array, not the bus or the connector. It is not a substitute for full ECC, and marketing material sometimes blurs the two.
10. Real failure rates, from "DRAM Errors in the Wild: A Large-Scale Field Study" by Schroeder, Pinheiro and Weber, published at SIGMETRICS 2009, covering Google's fleet from January 2006 to June 2008:

Measure	Value
DIMMs with a correctable error	8.2% per year
Correctable errors per DIMM	about 4,000 per year
DIMMs with an uncorrectable fault	0.22% per year
Machines with any error	about one third per year

11. That study overturned the belief that memory errors were rare and caused mainly by cosmic rays. Errors correlated strongly with specific DIMMs, pointing at hard faults rather than random strikes.
12. Observe ECC events with `edac-util -v`, `ras-mc-ctl --summary`, `dmesg | grep -i edac`, or `mcelog` on Linux.

7.9.6 WORDS - remember these

1. Parity bit — one extra bit making the count of 1s even or odd — a distance-2 code detecting any odd number of bit errors.
2. Checksum — add everything up and compare — a weak integrity value, often a one's complement sum as in RFC 1071.
3. CRC — a remainder from carry-less division — a cyclic redundancy check, the remainder of the message polynomial divided by a generator polynomial over GF(2).
4. ECC memory — memory that repairs single flipped bits — DRAM with SECDED Hamming coding, 8 check bits per 64 data bits.
5. SECDED — fix one error, spot two — single error correct, double error detect, the standard server DRAM protection level.
6. Collision resistance — nobody can forge a match — the property that distinguishes a cryptographic hash from a checksum or CRC.

7.10 Compression

7.10.1 PLAIN - in simple words

1. Real data repeats itself. English text is full of "the". Photographs have large areas of near-identical sky. Compression finds the repetition and writes it down once instead of many times.
2. It also exploits unevenness. In English 'e' is far more common than 'z', so giving both the same eight bits is wasteful.

3. Short codes for common things and long codes for rare things shrink the total. That is the second big idea.
4. **Lossless** compression returns the exact original bytes. ZIP, PNG and gzip are lossless.
5. **Lossy** compression throws away detail people are unlikely to notice and cannot give the original back. JPEG and MP3 are lossy.
6. Never use lossy compression on text, code or a bank record. One changed character can change the meaning completely.
7. And no scheme shrinks everything. Compressing an already compressed file usually makes it very slightly larger.

7.10.2 PLAIN - a picture in your head

1. Think of packing a suitcase. Folding clothes flat removes trapped air. The clothes are unchanged; you removed only the wasted space. That is lossless.
2. Now think of leaving three of your five shirts at home. The suitcase is far lighter, but those shirts are gone. That is lossy.
3. Now think of shorthand. A stenographer writes one squiggle for a whole common word and spells out the rare ones. That is entropy coding.
4. And think of a note saying "same as page 4, lines 2 to 9". That is dictionary compression: pointing back instead of repeating.

Where this comparison breaks

1. A suitcase can always be packed a bit tighter with effort. Data has a hard mathematical floor, the entropy, below which no lossless scheme can go.
2. And unfolding a suitcase is easy either way. Decompression cost varies a great deal between schemes, which is often the deciding factor.

7.10.3 PLAIN - a worked example

Run-length encoding. Replace each run with a count and a value.

```
Input  (53 characters):
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWB

Output (15 characters):
12W1B12W3B24W1B

53 down to 15, a saving of about 72 percent.
```

1. But run-length encoding on text with no runs makes it twice as long, since every single character becomes a count plus a character.

Huffman coding, worked completely on the word "abracadabra".

```
Symbol counts: a=5  b=2  r=2  c=1  d=1     total 11 symbols

Step 1: join the two smallest, c(1) and d(1)      -> node cd = 2
Step 2: smallest are now b(2), r(2), cd(2).
        join b(2) and r(2)                        -> node br = 4
Step 3: join cd(2) and br(4)                      -> node X = 6
Step 4: join a(5) and X(6)                        -> root  = 11
```

The finished tree, with 0 for left and 1 for right:

```
                [11]
               /    \
          0 /        \ 1
```

```
          a(5)         [6]
                      /   \
                 0 /     \ 1
              cd[2]        br[4]
              /   \        /   \
           0 /     \ 1  0 /     \ 1
          c(1)     d(1) b(2)     r(2)
```

The code table read off the tree:

Symbol	Count	Code	Bits
a	5	0	1
c	1	100	3
d	1	101	3
b	2	110	3
r	2	111	3

Encoding the whole word:

```
 a   b    r    a   c    a   d    a   b    r    a
 0  110  111   0  100   0  101   0  110  111   0

= 01101110100010101101110      (23 bits)

Plain ASCII would need 11 x 8 = 88 bits.
A fixed 3-bit code for 5 symbols would need 11 x 3 = 33 bits.
Huffman needs 23 bits.
```

2. No code is a prefix of any other code, so the decoder never needs separators. It walks down the tree, emits a symbol at each leaf, and returns to the root.

7.10.4 PLAIN - what is really happening inside

1. Huffman coding builds the tree from the bottom by always joining the two least frequent items. Rare symbols therefore end up deepest, with the longest codes. This is provably optimal for whole-bit codes.
2. LZ77 works on a completely different principle. It keeps a window of the recently seen bytes and, whenever the upcoming text has appeared before, it emits a back-reference instead of the text.
3. A back-reference is a pair: how far back to look, and how many bytes to copy. For example, “go back 3 bytes and copy 9”.
4. The length may exceed the distance. Copying is done one byte at a time, so “back 3, copy 9” on “abc” produces “abcabcabc”. That is how LZ77 also does run-length encoding for free.
5. **DEFLATE** is simply both together. First LZ77 replaces repeats with back-references. Then Huffman coding compresses the resulting stream of literals, lengths and distances.
6. That two-stage design is why gzip, ZIP and PNG all behave alike: they are the same algorithm with different containers around it.
7. The mathematical floor is **entropy**. For a source where symbol x has probability $p(x)$, the entropy in bits per symbol is:

```
H = - sum over all x of  p(x) x log2 p(x)
```

8. In words: it is the average number of bits genuinely needed per symbol. No lossless method can average fewer than H bits per symbol over the long run.
9. For “abracadabra” the entropy is 2.04 bits per symbol, so 22.4 bits for the whole word. Huffman achieved 23 bits. It came within one bit of the theoretical floor, which is exactly the guarantee Huffman coding offers.

7.10.5 TECHNICAL - the engineer's version

1. David A. Huffman published "A Method for the Construction of Minimum-Redundancy Codes" in the Proceedings of the IRE in September 1952, written as a term paper for Robert Fano's information theory class at MIT. Huffman coding is optimal among prefix codes with integer code lengths; it is never worse than H + 1 bits per symbol.
2. Arithmetic coding and its modern relative asymmetric numeral systems, or ANS, published by Jarek Duda from 2009, break the whole-bit restriction and reach the entropy bound more closely. ANS is what makes Zstandard fast.
3. Jacob Ziv and Abraham Lempel published "A Universal Algorithm for Sequential Data Compression" in IEEE Transactions on Information Theory in May 1977, giving LZ77. Their 1978 follow-up gave LZ78, from which LZW derives.
4. DEFLATE was created by Phil Katz for PKZIP 2.0 in 1993 and specified by L. Peter Deutsch in RFC 1951, May 1996. The related containers are RFC 1950 for zlib and RFC 1952 for gzip. DEFLATE uses a 32 KiB sliding window and match lengths of 3 to 258 bytes.
5. Claude Shannon defined entropy in "A Mathematical Theory of Communication", Bell System Technical Journal, July and October 1948. The source coding theorem in that paper is the statement that H is a hard lower bound.
6. Modern general-purpose compressors:

Name	Year	Basis
DEFLATE	1993	LZ77 plus Huffman
bzip2	1996	Burrows-Wheeler plus Huffman
LZMA / xz	2001	LZ77 plus range coding
Brotli	2015	LZ77, Huffman, static dict
Zstandard	2016	LZ77 plus ANS

7. Brotli is specified in RFC 7932 (2016) and includes a built-in 120 KiB dictionary of common web strings. Zstandard, created by Yann Collet at Facebook, is specified in RFC 8878 (February 2021) and is now the default in Linux kernel modules, Btrfs and many package managers.
8. The counting argument for why universal compression is impossible: there are 2^n distinct inputs of length n and only 2^n - 1 possible outputs shorter than n bits in total across all shorter lengths. A lossless injective map cannot fit them all. Therefore any scheme that shrinks some inputs must expand others. DEFLATE's stored block mode caps the expansion at about 5 bytes per 65,535-byte block.
9. Lossless versus lossy, with real formats: lossless are PNG, FLAC, ALAC, gzip, ZIP, TIFF with LZW; lossy are JPEG, WebP in lossy mode, MP3, AAC, Opus, H.264, H.265, AV1. Image and audio codecs are covered in Chapters 8 and 9; the compression machinery underneath them is what you have just read.
10. Measure with `gzip -9 -c file | wc -c`, `zstd -19`, `xz -9e`, and `ent` for a quick entropy estimate of a file.

7.10.6 WORDS - remember these

1. Lossless — you get the exact bytes back — a reversible encoding, as in DEFLATE, PNG and FLAC.
2. Lossy — some detail is discarded forever — an irreversible encoding tuned to human perception, as in JPEG and MP3.
3. Run-length encoding — write "twelve whites" instead of twelve whites — a code replacing each maximal run with a count and a value.
4. Huffman coding — short codes for common symbols — an optimal prefix code built bottom-up by repeatedly merging the two least frequent nodes.
5. Prefix code — no code is the start of another — a uniquely decodable code requiring no separators between symbols.

6. LZ77 — point back to text you already sent — sliding-window dictionary compression emitting (distance, length) back-references.
7. Entropy — the true information content — H = -sum p(x) log2 p(x) bits per symbol, the lower bound on lossless compression.

7.11 How a number becomes a thing

7.11.1 PLAIN - in simple words

1. Here is the deepest idea in this chapter, and it is short: a pattern of bits means nothing at all by itself.
2. The same 32 bits can be a number, a fraction, four letters, a colour or a machine instruction.
3. Nothing in memory says which. There is no tag, no label, no type stored alongside the data.
4. The meaning comes entirely from what the program decides to do with it. If the program treats those bits as a number, they are a number. If it treats them as text, they are text.
5. That is why file formats begin with magic numbers, why network protocols have header fields, and why programming languages have types. Each is a way of writing down, somewhere, what the bits are supposed to mean.

7.11.2 PLAIN - a picture in your head

1. Think of the digits 0812. On a door they are a house number. On a form they are a date. On a receipt they are a price. On a lock they are a code.
2. The digits never change. Where you find them, and what you were expecting, decides what they mean.
3. Now remove all context. Someone hands you a slip of paper with 0812 and walks away. You genuinely cannot know.
4. Raw memory is that slip of paper. Every byte, every time.

Where this comparison breaks

1. A person seeing 0812 on a receipt can notice the context is wrong and ask.
2. A program cannot. It applies whatever interpretation it was written with, silently, and produces a confident wrong answer. That failure mode is called **type confusion** and it is a common security vulnerability class.

7.11.3 PLAIN - a worked example

One pattern, five readings. The bits are 0x41424344.

```
The 32 bits:
  0100 0001 0100 0010 0100 0011 0100 0100

Read as a 32-bit unsigned integer      -> 1,094,861,636
Read as a 32-bit signed integer        -> 1,094,861,636
Read as an IEEE 754 float32            -> 12.141422
Read as four ASCII bytes               -> A B C D
Read as R,G,B,A colour bytes           -> 65, 66, 67, 68
Read as two 16-bit big-endian integers -> 16706 and 17220
```

1. The float reading works out as sign 0, exponent 0x82 which is 130, giving 2^3, and a fraction of 0.5176778, so 1.5176778 times 8 is 12.141422.
2. The colour reading gives a very dark, slightly blue grey. In web notation the first three bytes are the colour #414243.
3. If the same four bytes are read on a little-endian machine as one 32-bit integer loaded from memory in the other order, the value becomes 0x44434241, which is 1,145,258,561. A sixth reading, from byte order alone.

You can reproduce all of it:

```
import struct
p = 0x41424344
b = struct.pack('>I', p)
print(struct.unpack('>I', b)[0])   # 1094861636
print(struct.unpack('>f', b)[0])   # 12.141422271728516
print(b.decode('ascii'))           # ABCD
print(list(b))                     # [65, 66, 67, 68]
```

7.11.4 PLAIN - what is really happening inside

1. Memory is a flat array of bytes with addresses. It carries no type information whatsoever.
2. The type lives in the compiled code, not in the data. When the compiler saw `int x`, it emitted an integer load instruction for every use of x.
3. If the same address is later read through a float pointer, the CPU issues a floating point load, and the floating point unit interprets the exponent and fraction fields of whatever is there.
4. Neither instruction checks anything. Both succeed. Only one is meaningful.
5. A **cast** in C changes nothing in memory. It changes which instruction the compiler emits.
6. A C `union` deliberately overlaps two types at one address, which is the controlled version of the same trick.
7. This is also why the same bytes can be executed. Jump to an address and the CPU fetches those bytes into the instruction decoder instead of a register. Attacks that redirect execution into data exploit exactly this.
8. The honest version: some systems do tag data in hardware. The Burroughs B5000 and Lisp machines tagged words; CHERI, developed at Cambridge and Arm and shipped in the Arm Morello prototype in 2022, tags pointers with capability metadata. These are real but exceptional. Mainstream hardware stores untagged bits.

7.11.5 TECHNICAL - the engineer's version

1. Type is a static property of a program, not a runtime property of memory, on all mainstream hardware. C's object model calls the intended reading the **effective type** of the storage.
2. Reading an object through a pointer of an incompatible type violates the strict aliasing rule in C and C++ and is undefined behaviour. The legal way to reinterpret bits is `memcpy`, or `std::bit_cast` in C++20, both of which compile to nothing on real targets.
3. `-fno-strict-aliasing` disables the compiler's use of that rule. The Linux kernel builds with it, because too much existing code depends on aliasing.
4. Type confusion is CWE-843 and appears regularly in browser and interpreter vulnerabilities, where an object is allocated as one class and later accessed as another.
5. Because meaning is external, every serious data format carries it explicitly. Examples already met in this chapter: file magic numbers, the TIFF "II" or "MM" tag, HTTP `Content-Type` headers, and Unicode byte order marks.
6. The same principle explains why the encoding of a text file cannot be determined with certainty. Charset detectors such as `charset-normalizer` are statistical guesses, not measurements. Any file of bytes below 0x80 is simultaneously valid ASCII, valid UTF-8, valid Latin-1 and valid Windows-1252.
7. Inspect the same bytes several ways with `xxd`, `od -t x1 -t c -t d4`, `objdump -D -b binary -m i386:x86-64`, and `numpy.ndarray.view`.

7.11.6 WORDS - remember these

1. Interpretation — the decision about what bits mean — the effective type applied to a region of storage by the code that accesses it.
2. Cast — telling the compiler to read bits differently — a conversion that may reinterpret or may genuinely convert, depending on the language.

3. Type confusion — using data as the wrong kind of thing — CWE-843, accessing an object through an incompatible type, a common exploit class.
4. Magic number — a format's fingerprint — a fixed byte signature at a known offset used to identify content when no external label exists.
5. Strict aliasing — the rule that objects of different types do not overlap — a C and C++ assumption that permits optimization and makes type punning through pointers undefined.

7.98 Common wrong ideas

1. Wrong: computers use binary because it is more efficient. Right: they use it because a switch has two reliable states. Binary is the widest noise margin you can get from one wire, not the densest coding.
2. Wrong: hexadecimal is a different kind of number that computers understand. Right: hex is only a way of writing binary for humans. The machine stores bits; hex appears when something prints them.
3. Wrong: a byte has always been 8 bits. Right: the CDC 6600 used 6-bit characters and the PDP-10 used programmable 6, 7, 8 or 9-bit bytes. That is precisely why standards say "octet".
4. Wrong: a 64-bit CPU can address 16 exbibytes of memory. Right: the architecture defines 64-bit pointers, but shipping x86-64 chips implement 48 address bits, giving 256 TiB, or 57 bits with 5-level paging.
5. Wrong: floating point numbers are approximate because computers are imprecise. Right: every float32 and float64 value is an exact binary number. The error comes from rounding a decimal input to the nearest representable binary value, which is a property of the format, not a hardware defect.
6. Wrong: use a small tolerance like 0.000001 to compare floats. Right: use a relative tolerance based on machine epsilon, 1.19e-7 for float32 and 2.22e-16 for float64, scaled to the magnitude being compared.
7. Wrong: UTF-8 is a 1-byte encoding and Unicode is a 2-byte encoding. Right: Unicode is a numbering scheme with no byte width at all. UTF-8 uses 1 to 4 bytes and UTF-16 uses 2 or 4.
8. Wrong: a Unicode character is one code point. Right: what a reader calls one character is a grapheme cluster, which may be many code points. The family emoji is 7 code points and 25 UTF-8 bytes.
9. Wrong: a CRC or a checksum proves the data was not tampered with. Right: it detects accidental corruption only. Anyone can alter data and recompute a CRC. Tamper resistance needs a cryptographic hash or a MAC.
10. Wrong: compression can shrink any file if the algorithm is clever enough. Right: by a simple counting argument, any lossless scheme that shrinks some inputs must expand others. Shannon entropy sets a hard floor.

7.99 Chapter summary in 20 lines

1. Computers use base 2 because a wire is reliably above or below a threshold, and nothing in between can be read back safely.
2. N bits hold 2^N patterns, giving unsigned values from 0 to 2^N minus 1.
3. Convert decimal to binary by repeated division by 2, or by greedy subtraction of descending powers of 2. Both give the same answer.
4. Hexadecimal is binary written four bits at a time. Two hex digits are always exactly one byte, which is why engineers read memory in hex.
5. A byte is 8 bits today, but was not always, which is why standards say octet. A word is the CPU's natural width, now 64 bits.
6. A 32-bit address space names 4,294,967,296 bytes, exactly 4 GiB, which is the wall 32-bit systems hit.
7. Two's complement won because flipping the bits and adding one makes subtraction into addition, so one adder circuit does both jobs.
8. Signed 32-bit tops out at 2,147,483,647. That number caused the Year 2038 problem and the YouTube view counter overflow of 2 December 2014.
9. Ariane 5 Flight 501 was destroyed on 4 June 1996 because a 64-bit float was converted into a 16-bit signed integer that could not hold it.

10. IEEE 754, approved in 1985 under William Kahan, stores a sign bit, a biased exponent and a fraction with an implicit leading one.
11. In float32 the bias is 127, so 0.15625 encodes as 0x3E200000 and -6.25 encodes as 0xC0C80000.
12. All-ones and all-zeros exponents are reserved, giving zero, negative zero, subnormals, infinities and NaN, which is unequal to itself.
13. 0.1 plus 0.2 is not 0.3 because 0.1 is a repeating binary fraction, so both inputs and the sum are rounded to different nearby values. Money must use integer minor units or a decimal type.
14. ASCII, published as ASA X3.4-1963 on 17 June 1963, used 7 bits, put digits at 0x30 so masking gives their value, and separated ‘A’ from ‘a’ by one bit, bit 5.
15. The spare eighth bit produced the code page era, where the same bytes meant different characters in different countries, with no label in the file.
16. Unicode gives every character one permanent code point from U+0000 to U+10FFFF, across 17 planes; version 17.0 of 9 September 2025 holds 159,801 characters.
17. UTF-8, designed by Ken Thompson and Rob Pike in September 1992 and now specified by RFC 3629, is 1 to 4 bytes, ASCII-compatible and self-synchronising, which is why it won.
18. Endianness decides which byte of a multi-byte value sits at the lowest address; network byte order is big-endian, while x86-64 and ARM are little-endian.
19. Parity detects odd numbers of flipped bits, CRC32 catches all bursts up to 32 bits, ECC memory repairs one bit per 64 and detects two, and none of them resist a deliberate attacker.
20. Compression works because data repeats and is uneven; Huffman gives short codes to common symbols, LZ77 points back at earlier text, DEFLATE is both, and Shannon entropy is the floor none of them can pass.

8

Pixels, Screens and How You See an Image

8.0 What this chapter gives you

1. You will be able to say what light is, and why your eye sees a slice of it.
2. You will be able to explain why three colours can fake millions.
3. You will be able to say what a pixel is, and why it is not one dot.
4. You will be able to work out the PPI of any screen from two numbers.
5. You will be able to say when extra pixels stop helping.
6. You will be able to explain how an LCD blocks light and an OLED makes it.
7. You will be able to describe a framebuffer, its stride and double buffering.
8. You will be able to do the bandwidth sum for 4K at 60 Hz and 120 Hz.
9. You will be able to explain tearing, vsync and adaptive sync.
10. You will be able to explain how a finger becomes two numbers.

Drawing the picture, by the graphics chip, is Chapter 22. This chapter runs from light, through the panel, to the pixels in memory and out to your eye.

8.1 Light and the eye

8.1.1 PLAIN – in simple words

1. Light travels as a wave, and the distance from one crest to the next is its wavelength.
2. Your eye reacts to about 380 to 700 nanometres, a nanometre being a billionth of a metre. Short waves look violet, long waves red.
3. At the back of the eye sits the retina. Rods work in dim light and report no colour; cones report colour.
4. You have only three kinds of cone, so every colour you see is your brain comparing three numbers.

8.1.2 PLAIN – a picture in your head

1. Imagine three people watching one lamp through tinted glasses, one bluish, one greenish, one reddish.
2. None can name a colour. Each shouts one number: how bright it looks.
3. Those three numbers are the whole message your brain gets.
4. So a different lamp giving the same three numbers cannot be told apart.

Where this comparison breaks: the glasses overlap heavily, since green and red cones peak only about 30 nanometres apart, and the retina turns the three numbers into difference signals before they leave the eye.

8.1.3 PLAIN – a worked example

1. A pure beam at 580 nanometres looks yellow. Red at 630 plus green at 530 on one spot can look identical, though physically they differ.
2. Below are illustrative cone responses scaled to 100, not measurements.

Light source	S cone	M cone	L cone
Pure 580 nm	0	60	85
630 nm + 530 nm	0	60	85

3. Two lights matching this way are **metamers**. Not an illusion: it follows from having only three sensors.

8.1.4 PLAIN - what is really happening inside

1. A photon strikes a cone, a molecule changes shape, and the cell's voltage changes. That is the signal.
2. Whether a photon is caught depends on wavelength, and each cone type has its own catching curve.
3. Once caught, the wavelength is forgotten. The cell only says "I caught one". This is univariance.
4. The retina combines the three into light-dark, red-green and blue-yellow channels.
5. About 6 million cones feed about 1 million nerve fibres per eye, so heavy compression happens inside the eye.

8.1.5 TECHNICAL - the engineer's version

1. The CIE, the international body for light measurement, quotes 380 nm to 780 nm; useful sensitivity is roughly 400 nm to 700 nm.
2. The honest version: there is no hard edge. Sensitivity falls smoothly, and at high intensity people detect out to about 310 nm and 1100 nm. Any stated boundary is a convention.

Cone	Name	Peak (nm)	Share of cones
S	Short wave	about 420	about 2%
M	Medium	about 530	roughly 30%
L	Long wave	about 560	roughly 65%

3. The L to M ratio ranges from about 1.1:1 to 16:1 with no difference in colour naming. Rods number 90 to 120 million per retina and cones 6 to 7 million; sources differ, so treat these as approximate.
4. The CIE 1931 standard observer defines colour matching over 2 degrees, the 1964 supplementary observer over 10. The CIE 1924 photopic curve peaks at 555 nm and the scotopic curve at 507 nm, a shift called the Purkinje effect.
5. Trichromatic theory came from Thomas Young in 1802 and Hermann von Helmholtz in the 1850s; cone pigments were measured directly only in the 1960s.
6. Tools: a spectroradiometer measures the full spectral power distribution, while a colorimeter such as an X-Rite i1Display measures only tristimulus values, which is why cheap meters get fooled by new panel types.

8.1.6 WORDS - remember these

1. Wavelength — how long one wave is — crest-to-crest distance in nanometres.
2. Retina — the light-sensing sheet in the eye — neural tissue holding the photoreceptors.
3. Rod — the dim-light cell — scotopic photoreceptor, peak 507 nm, no colour.
4. Cone — the colour cell — photopic photoreceptor in S, M and L types.
5. Metamerism — different lights looking identical — distinct spectra, equal tristimulus values.
6. Tristimulus — three numbers for a colour — CIE X, Y, Z from the standard observer.

8.2 What a pixel actually is

8.2.1 PLAIN - in simple words

1. A pixel is the smallest part of a picture the screen controls separately. The word is short for picture element.
2. It is not one dot but a group of separate lights, called sub-pixels, usually one red, one green and one blue.
3. Yellow means red on, green on, blue off. White is all on, black is all off.
4. Your eye cannot separate them, so it blends them into one colour. A pixel is a control unit, not an object you could pick up.

8.2.2 PLAIN - a picture in your head

1. Think of a stadium crowd holding coloured cards to make a giant picture.
2. On the pitch you see individual people with red, green and blue cards.
3. From the far end the people vanish and one smooth image appears. Only your ability to separate the parts changed.
4. Each block of three neighbours is one pixel.

Where this comparison breaks: cards are up or down, but sub-pixels have hundreds of brightness levels, and in some layouts a pixel borrows a sub-pixel from its neighbour.

8.2.3 PLAIN - a worked example

1. Look closely at white on a monitor, through a drop of water acting as a lens. You see repeating vertical stripes.

```
one pixel      one pixel      one pixel
+---+---+---+  +---+---+---+  +---+---+---+
| R | G | B |  | R | G | B |  | R | G | B |
+---+---+---+  +---+---+---+  +---+---+---+
 RGB stripe: 3 sub-pixels per pixel, side by side
```

2. A 1920 by 1080 monitor has 2,073,600 pixels, so 6,220,800 driven sub-pixels.
3. On a 24-inch 1080p panel one pixel is about 0.277 mm wide, so one sub-pixel is about 0.092 mm. Thinner than a hair.

8.2.4 PLAIN - what is really happening inside

1. Each sub-pixel has its own switch, and drivers set a voltage deciding how much light passes or is emitted.
2. The three land on the same tiny patch of retina, and your cones add up all light in that patch without reporting where it came from.
3. Software can exploit this. Lighting only the red sub-pixel puts a sliver of edge one third of a pixel to the left.
4. That is sub-pixel anti-aliasing. Anti-aliasing means smoothing a jagged staircase edge with in-between brightness levels.
5. It triples horizontal text accuracy at the cost of faint colour fringes.

8.2.5 TECHNICAL - the engineer's version

Layout	Sub-pixels per pixel	Typical use
RGB stripe	3 full	Monitors, TVs
RGBW	4, W adds brightness	LG WOLED TVs
PenTile RGBG	2 per pixel, shared	AMOLED phones
QD-OLED	3 in a triangle	Samsung panels

1. PenTile came from Clairvoyante, founded by Candice Brown Elliott; Samsung bought the intellectual property in 2008 and formed Nouvoyance.
2. PenTile RGBG uses one red and one blue per two greens, exploiting peak luminance sensitivity near green, so a "2560 by 1440" PenTile panel has fewer than 2560 x 1440 x 3 sub-pixels.
3. The honest version: makers count PenTile panels by pixel address, not sub-pixel count. Both are defensible; marketing picks the larger.
4. Sub-pixel rendering was announced by Microsoft as ClearType at COMDEX in 1998. It needs the physical stripe order, so it fails on rotated monitors and on PenTile.
5. Apple removed it in macOS 10.14 Mojave, September 2018, arguing Retina panels no longer need it. Users on non-Retina external monitors disagreed, and that disagreement is still live.
6. Fill factor is the pixel area that actually emits or transmits; the rest is transistors, wiring and black matrix. LCDs are typically 50% to 70%.
7. Tools: a 200x USB microscope resolves sub-pixels, `xrandr --verbose` reports geometry, and fontconfig's `rgba` setting selects the stripe order.

8.2.6 WORDS - remember these

1. Pixel — the smallest controllable picture part — one addressable sample in the framebuffer and panel matrix.
2. Sub-pixel — one coloured light inside a pixel — an individually driven cell with its own transistor.
3. RGB stripe — red, green, blue side by side — the conventional vertical geometry of desktop LCDs.
4. PenTile — a layout that shares sub-pixels — RGBG or RGBW with under three sub-pixels per address.
5. Anti-aliasing — smoothing jagged edges — filtering to reduce spatial aliasing.
6. Fill factor — how much of a pixel lights up — emissive area over total pixel area.

8.3 Resolution and PPI

8.3.1 PLAIN - in simple words

1. Resolution is a count. "1920 by 1080" means 1920 columns and 1080 rows, so 2,073,600 pixels. It says nothing about sharpness.
2. PPI is a density: pixels per inch. It says how tightly packed they are.
3. A phone and a television can share a resolution and look nothing alike, one packing it into 6 inches and the other over 55.
4. What decides sharpness is neither alone. It is how big one pixel looks from where you sit.

8.3.2 PLAIN - a picture in your head

1. Think of a mosaic of square tiles. Resolution is the number of tiles; density is how small each one is.
2. Nose against the wall, you see every tile edge. Walk back and the edges disappear.
3. That distance depends on tile size, not on how many tiles the mural has.
4. A billboard of huge tiles looks perfect from the road and terrible from arm's length. Nothing about it changed.

Where this comparison breaks: tiles have grout lines and pixels usually do not, and your eye blurs slightly from its own lens imperfections, hiding the grid before your neurons get involved.

8.3.3 PLAIN - a worked example

1. Take the diagonal in pixels, then divide by the diagonal in inches.

```
diagonal_px = sqrt(width_px^2 + height_px^2)
PPI         = diagonal_px / diagonal_inches
24 in 1080p : sqrt(1920^2+1080^2)=2202.9 ; /24 = 91.8 PPI
27 in 1440p : sqrt(2560^2+1440^2)=2937.2 ; /27 = 108.8 PPI
6.8 in phone: sqrt(3120^2+1440^2)=3436.3 ; /6.8= 505.3 PPI
```

2. Samsung publishes 505 PPI for the Galaxy S24 Ultra, which matches.

Screen	Resolution	Size	PPI
Office monitor	1920 x 1080	24 in	91.8
4K monitor	3840 x 2160	27 in	163.2
4K television	3840 x 2160	55 in	80.1
Galaxy S24 Ultra	3120 x 1440	6.8 in	505.3

3. The television has four times the pixels of the office monitor and a lower density.

8.3.4 PLAIN – what is really happening inside

1. Your eye measures angle, not size.
2. A good eye just separates two lines about one arcminute apart. An arcminute is one sixtieth of a degree, the definition behind 20/20 vision.
3. So the question is whether one pixel covers less than one arcminute of your view.
4. Shortcut: the PPI you need is about 3438 divided by viewing distance in inches, so 287 PPI at 12 inches, 143 at 24 inches, 34 at 100 inches.
5. This is why a 505 PPI phone and an 80 PPI television both look sharp. Each beats the requirement at its own distance.

8.3.5 TECHNICAL – the engineer's version

1. Angular density is measured in pixels per degree, PPD. The classical 1 arcminute limit corresponds to 60 PPD.

```
PPD = PPI * distance_inches * tan(1 degree)
    = PPI * distance_inches * 0.017455
```

Display	PPI	Distance	PPD
24 in 1080p	91.8	24 in	38.5
27 in 1440p	108.8	24 in	45.6
27 in 4K	163.2	24 in	68.4
Phone at 505 PPI	505.3	12 in	105.8

2. Apple introduced the term Retina with the iPhone 4 on 7 June 2010: 960 by 640 on 3.5 inches, 326 PPI, claiming 300 PPI at 10 to 12 inches beats the eye's limit.
3. Raymond Soneira of DisplayMate argued the limit is nearer 0.6 arcminute, implying about 477 PPI at 12 inches. Others defended 1 arcminute. Experts genuinely disagree, and acuity varies between people.
4. Ashraf, Chapiro and Mantiuk published in Nature Communications on 27 October 2025 a measured foveal achromatic limit of 94 PPD, individuals reaching 120 PPD, with chromatic limits lower at 89 PPD red-green and 53 PPD yellow-violet. So 60 PPD is a floor, not a ceiling.
5. PPI is display pixel density; DPI is printer ink dot density. Printer dots are binary, so a 1200 DPI inkjet needs many dots per halftone cell and delivers perhaps 200 to 300 PPI of real detail.
6. The honest version: software calls display scaling "DPI" anyway. Windows does, and CSS fixes one inch at 96 CSS pixels. That is a convention, not a measurement.
7. Tools: `xdpyinfo | grep resolution` on X11, `system_profiler SPDisplaysDataType` on macOS, and `edid-decode` for the panel's physical size.

8.3.6 WORDS - remember these

1. Resolution — how many pixels — horizontal and vertical sample counts of a raster.
2. PPI — how tightly packed the pixels are — pixels per linear inch of diagonal.
3. PPD — how big a pixel looks from where you sit — pixels per degree of visual angle.
4. Arcminute — one sixtieth of a degree — the unit of visual acuity, the 20/20 reference.
5. DPI — printer dot density — dots per inch of binary ink, not the same as PPI.
6. EDID — the panel's identity card — Extended Display Identification Data, giving size and timings.

8.4 Colour

8.4.1 PLAIN - in simple words

1. A screen stores each pixel as three numbers: red, green, blue. Most use 8 bits each, giving 256 levels from 0 for off to 255 for full.
2. 256 x 256 x 256 = 16,777,216 combinations, the famous 16.7 million colours.
3. Better screens use 10 bits each, about 1.07 billion combinations. More levels does not mean more colourful; it means smoother steps.
4. A colour space is a written agreement on exactly which red, green and blue you meant. Without it the same numbers look different everywhere.

8.4.2 PLAIN - a picture in your head

1. Think of three paint taps, red, green and blue, each with 256 notches.
2. But "red" only helps if we both own the same tin of red paint.
3. A colour space is the label on the tin. sRGB is the small tin everybody has, DCI-P3 is bigger and more vivid, Rec.2020 is huge and no consumer screen fully owns it.
4. Sending sRGB numbers to a P3 screen without saying so is like following a recipe written for another tin. Everything comes out too strong.

Where this comparison breaks: paint mixes by subtracting light and screens by adding it. The notches are also not evenly spaced in brightness, which is the next idea.

8.4.3 PLAIN - a worked example

1. Mid grey is written 128, 128, 128. You would expect half of white's brightness. It emits about 21%.
2. The reason is gamma: light out is roughly (value / 255) to the power 2.2, and 0.502 to the power 2.2 = 0.216.
3. This is deliberate. Your eye sees differences in dark tones far better than bright ones, so gamma spends the 256 codes where you can see them.

```
3840 x 2160 pixels           = 8,294,400 pixels
8,294,400 x 3 bytes (24 bpp) = 24,883,200 B = 23.7 MiB
at 60 frames per second      = 1.49 GB per second
at 4 bytes per pixel         = 33,177,600 B = 31.6 MiB
```

4. One second of raw 4K at 60 Hz is about 1.5 gigabytes. That is why every video file you have opened is compressed.

8.4.4 PLAIN - what is really happening inside

1. Each number becomes a voltage on one sub-pixel through a lookup table, because the panel's own response is not a straight line either.
2. Gamma appears twice: encoding gamma when the image is stored, decoding gamma at the display. They are meant to cancel.
3. White point sets what 255, 255, 255 looks like, stated as colour temperature in kelvin. Lower is warmer and oranger, higher is bluer.

4. If white is wrong, every colour is wrong, because everything is measured relative to white.
5. HDR changes the deal: instead of "255 is as bright as this screen goes", a code means an absolute number of candelas per square metre.

8.4.5 TECHNICAL - the engineer's version

Space	Standard	Year	Note
sRGB	IEC 61966-2-1	1999	HP and MS, 1996
Rec.709	ITU-R BT.709	1990	Same primaries
Adobe RGB	Adobe de facto	1998	Wider green
DCI-P3	SMPTE RP 431-2	2011	Cinema projection
Rec.2020	ITU-R BT.2020	2012	UHD, very wide

1. sRGB primaries in CIE 1931 xy: red (0.6400, 0.3300), green (0.3000, 0.6000), blue (0.1500, 0.0600), white D65 at (0.3127, 0.3290), whose nominal correlated colour temperature is 6504 K.
2. The sRGB transfer function is piecewise, not a pure power law:

```
if C_srgb <= 0.04045:  C_lin = C_srgb / 12.92
else:                  C_lin = ((C_srgb + 0.055) / 1.055) ^ 2.4
```

3. The exponent is 2.4, but the linear toe near black makes the whole curve approximate 2.2. "sRGB gamma is 2.2" is a useful lie, not the specification.
4. Bit depths give 16,777,216 codes at 8-bit and 1,073,741,824 at 10-bit. Many consumer "10-bit" panels are 8-bit plus frame rate control, which dithers over time.
5. HDR curves: PQ, standardized as SMPTE ST 2084 in 2014, is absolute and defined to 10,000 cd/m2. HLG, from ARIB STD-B67 and ITU-R BT.2100, is relative and backward compatible with SDR broadcast.
6. VESA DisplayHDR tiers state peak luminance: 400, 600, 1000, 1400, plus True Black 400, 500 and 600. Marketing claim to distrust: an "HDR" badge with no tier number promises only that the panel accepts an HDR signal.
7. Tools: ArgyllCMS `dispcal` and `colprof` build ICC profiles and `dispwin` loads them; ColorSync Utility inspects them on macOS.

8.4.6 WORDS - remember these

1. Colour space — which red you meant — defined primaries, white point and transfer function.
2. Gamma — the bend between stored number and light — a nonlinear transfer function, nominally 2.2.
3. White point — the agreed shade of white — a chromaticity coordinate, usually D65 at 6504 K.
4. Bit depth — how many steps per colour — bits per component, 8, 10 or 12.
5. Gamut — the colours a screen can hit — the triangle enclosed by the display primaries.
6. PQ — the HDR curve with absolute brightness — the Perceptual Quantizer of SMPTE ST 2084.

8.5 How a display makes light, part 1: LCD

8.5.1 PLAIN - in simple words

1. An LCD does not make light. It blocks light. Behind the screen is a flat lamp that is always on: the backlight.
2. Liquid crystal flows like a liquid but its molecules line up like a crystal.
3. A voltage twists or untwists them, so light either passes or is stopped, and a coloured filter then makes it red, green or blue.
4. Because the lamp is always on, LCD black is a lamp behind a nearly closed shutter. Some light always leaks.

8.5.2 PLAIN – a picture in your head

1. Picture a window with two polarizing sunglass lenses, one rotated 90 degrees from the other. Together they block almost everything.
2. Now put a twisting layer between them that can rotate light by 90 degrees.
3. Twist on: light passes the first lens, is rotated, and lines up with the second. Bright.
4. Twist off: light passes the first unchanged and is stopped by the second. Dark. That twisting layer is the liquid crystal.

Where this comparison breaks: crossed polarizers reject far more than sunglasses, and the twist is not on or off but varies smoothly, which is how you get 256 grey levels instead of two.

8.5.3 PLAIN – a worked example

1. Here is the stack, from lamp to eye.

```
backlight (LED bar or sheet)
    |
diffuser + brightness films
    |
rear polarizer  (passes vertical waves only)
    |
glass + TFT layer (one transistor per sub-pixel)
    |
liquid crystal  (twist set by voltage)
    |
colour filter   (R, G or B dye)
    |
front polarizer (passes horizontal waves only)
    |
    v  your eye
```

2. Blocking is never perfect. A good IPS panel leaks about 1 part in 1000, a contrast ratio near 1000:1, so if white is 300 candelas per square metre, black is about 0.3, not zero.
3. In a dark room that leak looks dark grey. It is the biggest weakness of LCD.

8.5.4 PLAIN – what is really happening inside

1. Every sub-pixel has its own thin-film transistor, a TFT, and a small storage capacitor.
2. The panel is driven one row at a time: a gate line switches on that row's transistors while source drivers put the right voltage on every column.
3. Each capacitor holds its voltage until that row comes round again, one frame later. This is active matrix addressing.
4. The molecules take time to rotate, and that settling time is the panel's response time.
5. Drive voltage is reversed in polarity every frame, because constant DC would chemically damage the liquid crystal.
6. Local dimming attacks black level from behind, splitting the backlight into independently dimmed zones.

8.5.5 TECHNICAL – the engineer's version

1. Liquid crystals were discovered by the Austrian botanist Friedrich Reinitzer in 1888 in cholesteryl benzoate; Otto Lehmann named the phase.
2. George Heilmeier, Louis Zanoni and Lucian Barton at RCA Laboratories demonstrated dynamic scattering in 1964, and RCA announced the first liquid crystal displays in 1968.
3. The twisted nematic effect was patented by Wolfgang Helfrich and Martin Schadt at Hoffmann-La Roche, filed 4 December 1970 and published in Applied Physics Letters on 15 February 1971. James Fergason filed independently in the United States and the dispute ended in shared royalties.

Type	Contrast	Response	Viewing angle
TN	700-1000:1	fastest	poor, 170/160
IPS	1000-1500:1	medium	178/178
VA	2000-5000:1	slowest	178/178

4. TechSpot's aggregate measurements across many models give average contrast of about 872:1 for TN, 1037:1 for IPS and 2898:1 for VA.
5. TN is twisted nematic, IPS in-plane switching, VA vertical alignment. IPS rotates crystals in the panel plane, so off-axis colour shift is small. VA has black crush, IPS has off-angle glow, TN has vertical gamma shift.
6. Mini-LED shrinks the backlight LEDs so hundreds to thousands of dimming zones fit; high-end 2025 and 2026 monitors quote 1,000 to 5,000 zones.
7. The limit is blooming. With 2,000 zones on a 3840 by 2160 panel each zone covers about 4,000 pixels, so a white cursor on black lights a halo.
8. Tools: `ddcutil detect` and `ddcutil getvcp 10` read and set backlight level over DDC/CI.

8.5.6 WORDS - remember these

1. Backlight — the lamp behind the picture — edge-lit or direct-lit LED array with diffuser films.
2. Polarizer — a filter passing light waving one way — a film selecting one linear polarization state.
3. Liquid crystal — a material that twists light under voltage — a nematic phase, field-controlled.
4. TFT — the tiny switch at each sub-pixel — thin-film transistor in amorphous silicon, LTPS or IGZO.
5. Contrast ratio — how much brighter white is than black — white luminance over black luminance.
6. Local dimming — dimming the lamp behind dark parts — zone-wise backlight modulation.

8.6 How a display makes light, part 2: OLED

8.6.1 PLAIN - in simple words

1. An OLED has no backlight. Every sub-pixel is its own tiny lamp of organic, meaning carbon-based, material that glows when current flows.
2. For black, the sub-pixel switches off and emits nothing. That is true black, so contrast is effectively unlimited.
3. Switching a light on beats rotating a molecule, so OLED reacts far faster than LCD.
4. The cost is wear. Areas showing the same bright thing for years dim faster than the rest, and that uneven wear is burn-in.

8.6.2 PLAIN - a picture in your head

1. An LCD is a stage lit by one floodlight, each actor holding a shutter.
2. An OLED is a stage where every actor carries their own torch, and switches it off to be invisible; on the LCD stage the floodlight still spills.
3. But every torch has a battery that runs down with use.
4. An actor lit in the same spot every night grows dimmer, and you see their outline even in dark scenes.

Where this comparison breaks: the sub-pixel does not run out like a battery. It degrades chemically, and blue degrades fastest, so worn areas shift colour as well as dimming.

8.6.3 PLAIN - a worked example

1. Show full-screen black in a dark room. The LCD shows a faint grey rectangle; the OLED is indistinguishable from the wall.
2. Contrast ratio: LCD 300 / 0.3 = 1000:1. OLED 300 / 0.0005 = 600,000:1, and instruments often just report infinity.

3. Now show a small white square at full brightness. The OLED pushes it to 1000 candelas per square metre or more.
4. Make the square fill the screen and brightness drops sharply, often to a quarter. That is the automatic brightness limiter protecting the panel from heat and current. LCDs do not do this.

8.6.4 PLAIN - what is really happening inside

1. Each OLED sub-pixel is a sandwich: a cathode, several thin organic layers and a transparent anode.
2. Current injects electrons from one side and holes, meaning missing electrons, from the other; they meet in the emissive layer and release a photon whose colour depends on that layer's chemistry.
3. Brightness follows current, so OLED circuits control current, not voltage. That is a real design difference from LCD.
4. Lowering current shifts colour slightly, so many phones instead flash the panel on and off very fast and vary how long it stays on. That is PWM, pulse width modulation.
5. Averaged over time your eye reads that flicker as lower brightness. Some people are sensitive to it and report headaches.
6. Panels also track use per sub-pixel, quietly boost worn ones, and run pixel refresh cycles when idle.

8.6.5 TECHNICAL - the engineer's version

1. The efficient thin-film OLED was demonstrated by Ching Wan Tang and Steven Van Slyke at Eastman Kodak, published as "Organic Electroluminescent Diodes" in Applied Physics Letters volume 51, pages 913 to 915, in 1987. Both entered the National Inventors Hall of Fame in 2018.

Variant	How colour is made	Typical use
RGB OLED	Separate R, G, B	Phones, tablets
WOLED	White stack + filters	LG TVs
QD-OLED	Blue stack + QD layer	Samsung panels
Tandem	Stacked emitter decks	2024+ tablets

2. QD-OLED uses a blue emitter with quantum dots, nanocrystals that convert blue photons to pure red or green by size, avoiding WOLED's filter losses.
3. Tandem OLED stacks two or more emissive units in series, roughly doubling brightness at the same current density; Apple shipped it in the iPad Pro in May 2024.
4. Micro-LED replaces organics with inorganic LEDs, giving OLED blacks without burn-in. As of 2026 it is confined to very large modular walls and small wearables, because transferring millions of dice remains unsolved at cost. Treat consumer micro-LED as active engineering, not a shipping product.
5. PWM frequency is an implementation detail: older AMOLED phones used 240 Hz or 480 Hz, while 2024 and 2025 models advertise 1440 Hz, 2160 Hz and 3840 Hz. Whether PWM causes harm is contested: flicker below about 100 Hz has established effects, but evidence above 1000 Hz is weak and mostly self-reported.

Property	LCD (IPS/VA)	OLED
Black level	0.1-0.3 cd/m2	0.0000 cd/m2
Contrast	1000-5000:1	effectively inf.
Response	1-8 ms	0.03-0.2 ms
Full white	holds brightness	limiter kicks in
Burn-in risk	none	real, cumulative
Viewing angle	good to poor	excellent

6. Tools: a phone camera at 960 fps is the cheapest PWM detector; an oscilloscope with a photodiode is the correct one.

8.6.6 WORDS - remember these

1. OLED — a screen where each dot makes its own light — organic light emitting diode, current-driven.
2. Burn-in — a permanent ghost of an old image — differential luminance degradation, worst in blue.
3. PWM — flashing fast to look dimmer — pulse width modulation of drive current.
4. Quantum dot — a nanocrystal that converts colour — emission wavelength set by particle diameter.
5. ABL — the brightness limiter — automatic brightness limiter, cutting output as lit area grows.
6. Tandem OLED — two emitter stacks in one pixel — series units sharing a charge generation layer.

8.7 The older ways

8.7.1 PLAIN - in simple words

1. Before flat screens there was the cathode ray tube, or CRT: a glass vacuum bottle with a gun firing electrons at a phosphor coating that glows.
2. Magnets bend the beam so it sweeps in lines, left to right, top to bottom.
3. Each spot fades fast, so the picture is redrawn many times a second. Colour tubes use three guns and three phosphors.
4. Plasma screens later used tiny gas cells that glow when excited, and e-ink moves coloured particles and leaves them, holding an image with no power.

8.7.2 PLAIN - a picture in your head

1. Imagine writing on a wall with a torch whose trail fades in a fraction of a second.
2. To keep a picture visible you must run the torch over every line again and again, faster than the glow fades.
3. One sweep across is a scanline; switching off to return to the left edge is horizontal blanking.
4. Switching off to return to the top is vertical blanking, and complete sweeps per second is the refresh rate.

Where this comparison breaks: flat panels have no beam and nothing fades. They keep the same words and blanking gaps only because every cable, timing standard and driver was written around the sweeping beam.

8.7.3 PLAIN - a worked example

1. Take VGA: 640 by 480 at 60 Hz. The real signal is 800 by 525, the extra 160 columns and 45 rows being blanking.
2. 800 x 525 x 60 = 25,200,000, and the published pixel clock is 25.175 MHz.

Field	Active	Total	Blanking
Horizontal (px)	640	800	160
Vertical (lines)	480	525	45

3. Modern links still carry blanking. Even DisplayPort to an OLED sends those empty periods, though reduced blanking timings shrink them.

8.7.4 PLAIN - what is really happening inside

1. In a colour CRT a metal sheet full of holes, the shadow mask, sits behind the phosphor so the red gun can only reach red phosphor. It absorbs most of the beam energy, which is why CRTs ran hot.
2. In a plasma panel each cell holds a noble gas that voltage ionizes into plasma, emitting ultraviolet light.
3. That ultraviolet strikes phosphor inside the cell, which emits visible red, green or blue. Plasma is an array of tiny fluorescent tubes.

4. Plasma cells are on or off, so grey comes from switching them many times per frame, which some viewers saw as flicker.
5. In e-ink each capsule holds white particles with one charge and black ones with the opposite, in clear fluid. Voltage pulls one colour to the top, and removing it leaves them held by friction.
6. That is bistability, and it is why an e-reader holds a page on a flat battery.

8.7.5 TECHNICAL - the engineer's version

1. The cathode ray tube was built by Karl Ferdinand Braun at the University of Strasbourg in 1897, and was long called the Braun tube.
2. The plasma display panel was invented in 1964 by Donald Bitzer, H. Gene Slottow and Robert Willson at the University of Illinois, for the PLATO education system.
3. E Ink Corporation was spun out of the MIT Media Lab in 1997 from work by Joseph Jacobson with Barrett Comiskey and J. D. Albert; the founding paper, "An electrophoretic ink for all-printed reflective electronic displays", appeared in Nature in 1998.
4. IBM introduced VGA in 1987 with the PS/2 line, fixing 640 by 480 at 60 Hz on a 25.175 MHz pixel clock as a floor that still appears in boot screens.
5. Vocabulary inherited from CRTs and still in every specification: scanline, horizontal and vertical blanking interval, front porch, back porch, sync pulse, interlacing, overscan.
6. Overscan is the clearest fossil: CRT televisions drew beyond the visible glass, so many televisions still crop about 2.5% of an HDMI input in 2026 unless you find the 1:1 pixel mapping setting.
7. Plasma manufacturing ended in the 2010s: Panasonic stopped in 2014, LG and Samsung in 2014 and 2015.
8. Tools: `cvt 1920 1080 60` generates timings including blanking, and `xrandr --verbose` prints the active modeline.

8.7.6 WORDS - remember these

1. CRT — the old glass tube screen — deflected electron beam exciting phosphor.
2. Scanline — one horizontal sweep — one raster row, still the unit of display timing.
3. Blanking — the gap when nothing is drawn — retrace intervals, now carrying control data.
4. Phosphor — the powder that glows — a luminescent coating with a decay time.
5. Bistable — holds its image without power — a display whose two states are both stable.
6. Overscan — cropping the picture edges — legacy CRT television margin, worth switching off.

8.8 The framebuffer

8.8.1 PLAIN - in simple words

1. Before a picture can be shown it must exist as numbers in memory. That block of memory is the framebuffer.
2. It holds one complete picture, row by row from the top-left corner, each pixel taking a fixed number of bytes, usually four.
3. So finding any pixel is arithmetic, not searching.
4. One thing writes into this block and another reads out of it, at different speeds and without agreeing, which causes most of the trouble that follows.

8.8.2 PLAIN - a picture in your head

1. Think of a long shelf of numbered boxes. The first 1920 are the top row, the next 1920 the second row, and so on.
2. To find column 100 of row 50, count 50 full rows then 100 more boxes.
3. Now suppose the shelf is built in sections of 64 boxes and each row must start at a section boundary, leaving a few boxes empty at the end.

4. Those wasted boxes are padding, and the distance from one row's start to the next is the stride.

Where this comparison breaks: real memory is not one flat shelf. It has caches, pages and interleaved banks, and the padding exists to line those up, not for tidiness.

8.8.3 PLAIN - a worked example

1. A tiny image: 5 pixels wide, 3 tall, 4 bytes per pixel. A row of real data is 20 bytes, but rows must start on 32-byte boundaries, so the stride is 32 with 12 bytes of padding.

```
addr    contents
0x1000  px(0,0) px(1,0) px(2,0) px(3,0) px(4,0)   20 bytes
0x1014  pad pad pad                               12 bytes
0x1020  px(0,1) px(1,1) px(2,1) px(3,1) px(4,1)
0x1034  pad pad pad
0x1040  px(0,2) px(1,2) px(2,2) px(3,2) px(4,2)
0x1054  pad pad pad
total: 3 rows x 32 bytes = 96 bytes
```

2. The address formula, and one 32-bit pixel in memory:

```
addr = base + (y * stride) + (x * bytes_per_pixel);
/* pixel (3,1): 0x1000 + (1*32) + (3*4) = 0x102C */

/* orange, R=255 G=128 B=0 A=255, little-endian BGRA */
/* bytes in memory:  00   80   FF   FF               */
/*                    B    G    R    A               */
/* read as a word:  0xFFFF8000                       */
```

3. The byte order in memory is the reverse of the number you would print, a permanent source of bugs.

8.8.4 PLAIN - what is really happening inside

1. The display controller reads the framebuffer continuously at the rate the panel needs. At 60 Hz it starts a full read every 16.67 milliseconds whether new content exists or not.
2. If your program writes into the buffer while the controller reads it, the controller sends a mixture of old and new.
3. On screen that is a horizontal line with one frame above and another below. That is tearing.
4. The fix is two buffers: the controller reads A while the program writes B, then they swap. That is double buffering.
5. The swap must happen during vertical blanking, the gap between frames, or you tear anyway.
6. Triple buffering adds a third so the program can start another frame instead of stalling, at the cost of memory and about one frame of delay.

8.8.5 TECHNICAL - the engineer's version

Resolution	One buffer	Triple buffered
1920 x 1080	7.91 MiB	23.7 MiB
2560 x 1440	14.06 MiB	42.2 MiB
3840 x 2160	31.64 MiB	94.9 MiB
7680 x 4320	126.6 MiB	379.7 MiB

1. Common pixel formats: XRGB8888 and ARGB8888 at 32 bits, RGB565 at 16 bits, XRGB2101010 for 10-bit HDR, and NV12 for 4:2:0 video planes.
2. Stride, also called pitch, is in bytes and is always at least width times bytes per pixel. Alignment of 64 or 256 bytes is common; some GPUs demand far more.

3. Modern GPUs rarely store the framebuffer in simple row order. They use tiled or swizzled layouts so a 2D block of pixels lands in one cache line; linear layout is what gets exported for scanout.
4. The Linux DRM/KMS model exposes this directly: a `drm_framebuffer` wraps GEM buffer objects, and `drmModePageFlip` requests a swap at the next vertical blank, returning a `DRM_EVENT_FLIP_COMPLETE` event.
5. Legacy interfaces survive: `/dev/fb0` with `FBIOGET_VSCREENINFO`, and on many boards a linear framebuffer set up by UEFI GOP before any driver loads.
6. Tearing without vsync is not a bug in the strict sense. It is the defined behaviour of an unsynchronized page flip; calling it a bug is a convention of user expectation.
7. Tools: `modetest` lists planes, CRTCs and connectors; `wayland-info` reports buffer formats; `sudo fbset -i` prints stride and pixel format.

8.8.6 WORDS - remember these

1. Framebuffer — memory holding one full picture — a pixel buffer scanned out by the display engine.
2. Stride — bytes from one row start to the next — the row pitch including alignment padding.
3. Double buffering — draw in one, show the other — front and back buffers swapped by a page flip.
4. Page flip — the swap itself — changing the scanout base address at vertical blank.
5. Tearing — two frames visible at once — scanout crossing a mid-frame buffer change.
6. Vertical blank — the gap between frames — the safe window for atomic updates.

8.9 Getting the picture out

8.9.1 PLAIN - in simple words

1. The display controller reads the framebuffer and turns it into a signal, in the same order the old CRT beam moved. That is scanout.
2. The signal goes down a cable carrying not a picture but a stream of numbers plus timing marks saying "new line here" and "new frame here".
3. How much data must flow depends on pixel count, bits per pixel and refreshes per second. Multiply them for the bandwidth.
4. If that is bigger than the cable can carry, you lower one of the three or squeeze the data.

8.9.2 PLAIN - a picture in your head

1. Think of a conveyor belt feeding a machine that must never stop.
2. The belt must deliver at a fixed rate; too slow and the machine stalls, because there is no store to draw on.
3. Every part is one pixel, and belt speed is bandwidth.
4. A wider belt is more lanes in the cable, a faster belt is a higher rate per lane, and packing parts tighter is compression.

Where this comparison breaks: the link has a small buffer at the receiving end, and it does stop and restart during blanking and link training. The "never stops" rule is about display timing, not literally the wire.

8.9.3 PLAIN - a worked example

1. Work out 4K at 60 Hz with 8 bits per colour, then at 120 Hz with 10 bits.

```
4K60, 8-bit:
  3840 x 2160          = 8,294,400 pixels
  x 60 refreshes       = 497,664,000 pixels/s
  x 24 bits per pixel  = 11.94 Gbit/s

4K120, 10-bit:
```

```
8,294,400 x 120      = 995,328,000 pixels/s
x 30 bits per pixel  = 29.86 Gbit/s
```

2. Those are active pixels only. Blanking adds about 5% with reduced blanking and about 20% with old broadcast timings, so 4K at 60 Hz needs a 594 MHz pixel clock, giving 14.26 Gbit/s on the wire.
3. HDMI 2.0 has 18 Gbit/s raw, but 8b/10b encoding leaves 14.4 Gbit/s of payload. 14.26 fits, barely. Add 10-bit colour and it does not, which is why early HDR televisions dropped to 4:2:0 chroma.

8.9.4 PLAIN - what is really happening inside

1. The controller keeps a small FIFO buffer topped up from memory; if memory is too busy the FIFO empties and you see a glitch line, an underrun.
2. Pixels are serialized: parallel bytes become a fast stream of ones and zeros on a few differential wire pairs.
3. The encoding scheme adds extra bits so the receiver can recover the clock and detect errors, which is why raw and useful speed differ.
4. Before any picture flows, the source reads the display's EDID capability block, then trains the link by trying speeds until one works reliably.
5. The cable carries pixel lanes, a clock, an auxiliary control channel, hot-plug detect and power, all at once.
6. If the picture will not fit, DSC is switched on: Display Stream Compression squeezes each line with a fixed small delay and a fixed output size, so timing stays predictable.

8.9.5 TECHNICAL - the engineer's version

Interface	Raw Gbit/s	Payload	Year
HDMI 2.0	18	14.4	2013
HDMI 2.1	48	42.67	2017
HDMI 2.2	96	85.3	2025
DP 1.4 HBR3	32.4	25.92	2016
DP 2.1 UHBR20	80	77.37	2022

1. Encoding efficiency explains those pairs. HDMI 2.0 TMDS uses 8b/10b at 80%; HDMI 2.1 and 2.2 use FRL with 16b/18b at 88.9%; DisplayPort to 1.4 uses 8b/10b at 80%, and DisplayPort 2.x uses 128b/132b at about 96.9%.
2. HDMI 1.0 was released on 9 December 2002 by a founder group including Hitachi, Matsushita, Philips, Silicon Image, Sony, Thomson and Toshiba. DVI 1.0 came from the Digital Display Working Group in April 1999, and DisplayPort 1.0 was approved by VESA in May 2006.
3. The HDMI Forum released version 2.2 in June 2025 with a new Ultra96 cable category. Important caveat: the higher rates are optional, so an HDMI 2.2 label does not guarantee 96 Gbit/s.
4. DisplayPort 2.1 was announced on 17 October 2022, and DP 2.1b followed with the DP80LL low-loss cable category, announced alongside HDMI 2.2 in January 2025.
5. eDP, embedded DisplayPort, is the internal laptop link; version 1.5 was published by VESA in October 2021. Its distinguishing feature is Panel Self Refresh, which lets the source stop sending frames for a static image.
6. DSC is VESA Display Stream Compression, current version 1.2a: line-based, fixed-rate and low-latency, typically 8 to 12 bits per pixel against a 24 or 30 bit source, so roughly 2:1 to 3:1. VESA calls it visually lossless, verified under ISO/IEC 29170 testing; it is not mathematically lossless, which matters for medical and reference work.
7. The journey from memory to eye:

```
framebuffer in RAM
     |  scanout read, row by row
```

```
display controller (CRTC) -> colour pipeline, gamma LUT
     |
optional DSC encoder
     |
serializer -> 4 differential lane pairs + AUX channel
     |
cable (HDMI / DisplayPort / eDP)
     |
timing controller (TCON) in the panel
     |
source and gate drivers -> sub-pixel voltages
     |
light -> your retina
```

8. Tools: `xrandr --listmonitors`, `drm_info` and `get-edid | parse-edid` on Linux; `dmesg | grep -i drm` reveals link training failures.

8.9.6 WORDS - remember these

1. Scanout — reading the picture out in order — the display engine's fixed rate raster read.
2. Bandwidth — data per second on the link — bits per second, raw or payload after encoding.
3. TMDS and FRL — the two HDMI signalling schemes — Transition Minimized Differential Signalling and Fixed Rate Link.
4. DSC — squeezing the picture to fit the cable — VESA Display Stream Compression, fixed-rate.
5. Link training — the handshake before the picture — negotiating lane count, rate and equalization.
6. TCON — the chip inside the panel — timing controller driving the row and column drivers.

8.10 Refresh rate, frame rate and smoothness

8.10.1 PLAIN - in simple words

1. Refresh rate is how many times a second the screen redraws itself, in hertz. Frame rate is how many new pictures a second your software makes.
2. These are two different numbers and are usually not equal.
3. A 60 Hz screen redraws 60 times a second even if a game is stuck at 17 FPS. It just shows the same picture again.
4. When the two do not line up you get a torn image or uneven motion, and adaptive sync fixes it by letting the screen wait for the software.

8.10.2 PLAIN - a picture in your head

1. A photographer takes one picture every second, on the second, no matter what. A dancer moves through poses at her own irregular pace.
2. If she changes pose halfway through a shot, the photo catches half of each. That is tearing.
3. If she holds some poses for two shots and others for one, the series looks jerky even though every photo is sharp. That is stutter.
4. Adaptive sync is the photographer agreeing to shoot whenever she settles.

Where this comparison breaks: real adaptive sync works only inside a stated range, such as 48 to 144 Hz, and outside it the display repeats frames instead.

8.10.3 PLAIN - a worked example

1. A 60 Hz screen has a frame period of 1000 / 60 = 16.67 milliseconds; at 120 Hz it is 8.33 ms, at 240 Hz 4.17 ms.
2. Suppose a game takes 20 ms per frame, 50 FPS, with vsync on at 60 Hz. It misses the deadline, waits for the next one at 33.3 ms, and the effective rate collapses to 30 FPS.

3. A 20% overrun caused a 40% drop. That cliff is why vsync feels bad near a game's limit.

Refresh	Frame period	Missed deadline
60 Hz	16.67 ms	falls to 30 FPS
120 Hz	8.33 ms	falls to 60 FPS
144 Hz	6.94 ms	falls to 72 FPS
240 Hz	4.17 ms	falls to 120 FPS

4. With adaptive sync the screen simply refreshes at 50 Hz to match, and there is no cliff.

8.10.4 PLAIN - what is really happening inside

1. Adaptive sync works by stretching the vertical blanking interval, the gap between frames. The display holds the last line and waits.
2. A panel can only wait so long before its pixels drift, so there is a minimum rate; below it, drivers send the same frame two or three times.
3. Response time is a panel property: how long a pixel takes to change colour.
4. Input lag is a system property: the whole time from your action to a change on screen, across device, operating system, engine, queue, cable and panel.
5. Motion blur on a sample-and-hold display is not slow pixels. It is your eye tracking a moving object while that object sits still on screen for the whole frame period.
6. That is why black frame insertion and backlight strobing reduce blur: they shorten how long each frame is visible, at the cost of brightness.

8.10.5 TECHNICAL - the engineer's version

1. Nvidia announced G-Sync on 18 October 2013, using a proprietary FPGA module inside the monitor in place of the normal scaler; early modules used an Altera Arria V GX FPGA with 768 MB of DDR3L.
2. VESA added Adaptive-Sync as an optional part of DisplayPort 1.2a in May 2014, and AMD's FreeSync branding built on it with monitors shipping from 2015. Nvidia began certifying such monitors as "G-Sync Compatible" at CES in January 2019. HDMI added its own variable refresh rate in HDMI 2.1 in 2017, which is the path consoles use.
3. VESA introduced AdaptiveSync Display certification in 2022, with tested tiers such as AdaptiveSync Display 144, plus MediaSync Display for judder-free video.
4. Response time reporting is a mess. "1 ms" is usually grey-to-grey with maximum overdrive, measured between the 10% and 90% points, often with visible overshoot. GtG, MPRT and black-to-white numbers are not comparable.

Stage	Typical time
USB mouse poll, 1 kHz	1 ms
Render + present queue	5-30 ms
Scanout of full frame	frame period
LCD pixel transition	1-8 ms
OLED pixel transition	under 0.2 ms

5. Persistence blur is roughly the frame period times motion speed, a rule of thumb popularized by Blur Busters: an object moving 1000 pixels per second on a 60 Hz sample-and-hold display smears about 16.7 pixels wide.
6. So 60 Hz to 120 Hz is not mainly "more frames". It halves persistence blur and halves the worst-case wait for a new frame, both visible in ordinary scrolling.
7. Tools: `vblank_mode=0` and `__GL_SYNC_TO_VBLANK=0` disable vsync on Linux; `presentmon` on Windows measures real present intervals and latency.

8.10.6 WORDS - remember these

1. Refresh rate — how often the screen redraws — the vertical scan frequency in hertz.
2. Frame rate — how many pictures the software makes — presents per second.
3. Vsync — waiting for the screen before swapping — syncing page flips to the vertical blank.
4. Adaptive sync — the screen waits for the software — variable refresh by extending blanking.
5. Input lag — total delay from action to picture — end-to-end latency across the whole chain.
6. Persistence — how long a frame stays lit — hold time of a sample-and-hold display.

8.11 Touchscreens

8.11.1 PLAIN - in simple words

1. A touchscreen is a separate sensor on or inside the display. Resistive screens feel a press; capacitive screens feel a conductive object.
2. A resistive screen is two flexible conductive sheets with a tiny gap: press hard enough and they touch, completing a circuit at that spot.
3. Resistive works with gloves and pens but needs pressure and reliably senses only one point.
4. A capacitive screen has an invisible grid of transparent wires that your finger disturbs, needing no pressure and sensing many fingers at once.
5. A dedicated chip scans the grid many times a second and hands the operating system a list of contact points as coordinates.

8.11.2 PLAIN - a picture in your head

1. Picture a chessboard with a wire under every row and over every column.
2. At each crossing the wires are close but not touching, and a small amount of charge passes between them.
3. Put a finger near one crossing and some of that charge takes a shortcut through your body, so less arrives.
4. The controller pulses each row in turn, measures every column, and builds a grid of numbers. Wherever the numbers dip, there is a finger.

Where this comparison breaks: your finger is not a hole that swallows charge. It is a conductor tied to a large body acting as a weak ground, and the dip spreads over several crossings, which is what allows sub-cell accuracy.

8.11.3 PLAIN - a worked example

1. A small 6 by 4 sensor grid reports these values, where higher means more disturbance.

```
        c0   c1   c2   c3   c4   c5
r0       0    0    0    0    0    0
r1       0    2   14   22    6    0
r2       0    3   28   45   11    0
r3       0    1   10   16    4    0
```

2. The peak is row 2, column 3, value 45, but whole-cell accuracy is far too coarse, so the controller interpolates across row 2: $(2 \times 28 + 3 \times 45 + 4 \times 11) / (28 + 45 + 11) = 235 / 84 = 2.80$.
3. If the panel is 1080 pixels wide over 6 columns, each column is 180 pixels, so $x = 2.80 \times 180 + 90 = 594$ pixels. That number, a matching y and a contact identifier are what the operating system receives.

8.11.4 PLAIN - what is really happening inside

1. The touch controller runs its own scan loop, independent of the display and usually faster, because tracking a fast swipe needs more samples than showing it does.

2. Each scan gives a raw grid, and the controller subtracts a stored baseline to remove the effect of the case, temperature and water film.
3. It finds peaks, interpolates centres, and matches each to a contact from the previous frame, so a finger keeps its identifier while it moves.
4. It rejects blobs that are too large, which is how a phone ignores your palm and your cheek during a call.
5. Coordinates go to the main processor over I2C or SPI, with a separate interrupt line saying "new data ready".
6. The operating system's input stack turns that stream into gestures: tap, long press, drag, pinch, fling.

8.11.5 TECHNICAL - the engineer's version

1. The capacitive touchscreen was described by E. A. Johnson at the Royal Radar Establishment in Malvern, England, in 1965, for air traffic control. Resistive touch came from Samuel Hurst, whose Elographics was founded in

1971.

2. Two capacitive modes exist. Self capacitance measures each electrode against ground, is very sensitive, but ghosts with more than two fingers. Mutual capacitance measures each row-column intersection and is the basis of true multi-touch.
3. The transparent conductor is traditionally indium tin oxide, ITO; metal mesh and silver nanowire are used on larger and flexible panels because ITO's sheet resistance scales badly.

Parameter	Typical value
Sensor pitch	4 to 5 mm
Scan/report rate	120 to 480 Hz
Capacitance change	around 1 picofarad
Host bus	I2C or SPI plus IRQ
Reported contacts	5 to 10 simultaneous

4. High report rates are a measurable latency win: 240 Hz or 480 Hz sampling shortens the sensing step to 4.2 ms or 2.1 ms, though the rest of the pipeline usually dominates.
5. On Linux, multi-touch reaches userspace through evdev as type B events: ABS_MT_SLOT, ABS_MT_TRACKING_ID, ABS_MT_POSITION_X and ABS_MT_POSITION_Y. Windows uses HID digitizer usage pages; Android uses the same evdev events.
6. In-cell and on-cell integration put the touch electrodes inside or directly on the display stack instead of a separate glass layer, saving thickness and an air gap. In-cell is standard on flagship phones.
7. Tools: `evtest` on Linux prints raw touch events live; `getevent -lt` does the same on Android over adb.

8.11.6 WORDS - remember these

1. Resistive touch — two sheets pressed together — a voltage-divider scheme needing deflection.
2. Capacitive touch — sensing a conductive finger — measuring capacitance change on an electrode array.
3. Mutual capacitance — measuring each crossing — the row-column method enabling multi-touch.
4. Baseline — the no-touch reference reading — a slowly adapting per-node value.
5. Report rate — how often touch positions are sent — contact reporting frequency, not panel refresh.
6. In-cell — touch sensing built into the panel — electrodes integrated in the display stack.

8.12 Scaling and sharpness

8.12.1 PLAIN - in simple words

1. Every flat panel has exactly one real pixel grid, its native resolution. Unlike a CRT it cannot change that grid, because it is physical.
2. Send it a picture of a different size and something must stretch or shrink it. That is scaling, and it always guesses.
3. One output pixel usually falls between input pixels, so its colour is invented from the neighbours, which is why non-native looks soft.
4. Doubling exactly is special: each input pixel becomes a clean 2 by 2 block with no guessing, which is why 1080p looks clean on a 4K panel and poor on a 1440p one.

8.12.2 PLAIN - a picture in your head

1. Think of copying a drawing from squared paper onto paper with differently sized squares.
2. If the new squares are exactly half the size, each old square becomes four new ones. Perfect copy, no decisions.
3. If they are 1.33 times smaller, most old squares straddle two new ones and you must decide how to split the colour.
4. Pick the nearest and you get hard, uneven edges; blend the neighbours and you get smooth but blurry edges. Neither is wrong, because the information was never there.

Where this comparison breaks: real scalers work on gamma-encoded values, so naive averaging darkens the result. Correct scaling converts to linear light first, which many cheap scalers skip.

8.12.3 PLAIN - a worked example

1. Scale one row of 4 pixels up to 8.

```
input  index:   0    1    2    3
input  value:  10   20   30   40

nearest  : 10  10   20   20   30   30   40  40
bilinear : 10 12.5 17.5 22.5 27.5 32.5 37.5 40
```

2. Nearest keeps original values exactly but shows visible steps; bilinear gives a smooth ramp where no output pixel except the ends matches any original value.
3. Bicubic uses four neighbours and a smoother curve, sharper than bilinear at the cost of slight overshoot, a faint bright halo beside hard edges.

Method	Neighbours used	Look
Nearest	1	blocky, exact
Bilinear	2 per axis	soft, no halo
Bicubic	4 per axis	sharp, mild halo
Lanczos	6 or 8 per axis	sharpest, ringing

8.12.4 PLAIN - what is really happening inside

1. Scaling can happen in three places, and it matters which.
2. The application can render lower and scale up itself, choosing a good method; the graphics chip can scale during output, fast and usually bilinear; the monitor's own scaler is often worst and adds lag.
3. Integer scaling is the case where the ratio is a whole number, so each input pixel becomes an exact block and nothing is invented.
4. High-DPI operating system scaling is a different problem: not how to stretch an image, but how big to draw the interface in the first place.

5. The right approach is to tell applications "one logical point is now two physical pixels" and let them draw text and vectors at full detail.
6. The fallback, for applications that do not understand this, is to let them draw small and stretch the window. That is what a blurry old application on a modern laptop looks like.

8.12.5 TECHNICAL - the engineer's version

Platform	Mechanism	Typical factors
macOS	Backing scale	1x, 2x
Windows	Per-monitor DPI	100% to 350%
GNOME	Wayland fractional	100% to 300%
Android	Density buckets	mdpi to xxxhdpi

1. Apple's approach at non-integer settings is to render the desktop at 2x into an offscreen buffer and downscale to the panel. It looks correct and costs memory bandwidth; that is an implementation detail of macOS, not a general rule.
2. Wayland fractional scaling was standardized with the `wp_fractional_scale_v1` protocol, merged in 2022, letting a client render at the exact fractional scale instead of the next integer and down.
3. Android density buckets are nominal: mdpi is 160 dpi and defines 1 dp equal to 1 px, hdpi 240, xhdpi 320, xxhdpi 480, xxxhdpi 640. Devices are assigned to the nearest bucket, so dp is not a physical unit.
4. CSS fixes the reference pixel at 1/96 inch, and `devicePixelRatio` reports the ratio to physical pixels. That specification is why a "1px" border can be 3 physical pixels on a phone.
5. Scaling should be done in linear light. Scaling gamma-encoded sRGB values directly darkens fine detail, worst on high-contrast text, and many hardware scalers do it wrong for speed.
6. Reconstruction is moving past fixed filters. Temporal upscalers such as Nvidia DLSS, AMD FSR and Intel XeSS combine previous frames plus motion vectors, and learned variants use neural networks. Established fact: they beat bilinear at equal cost. Active research: removing ghosting and temporal instability. Marketing claim: output indistinguishable from native. Chapter 22 covers this properly.
7. Tools: `xrandr --output DP-1 --scale 1.5x1.5` applies GPU scaling on X11; on Windows, `SetProcessDpiAwarenessContext` declares whether an application handles scaling itself.

8.12.6 WORDS - remember these

1. Native resolution — the panel's real pixel grid — the fixed physical addressable matrix.
2. Scaling — resizing a picture to fit — resampling to a different sample grid.
3. Nearest neighbour — copy the closest pixel — zero-order hold resampling, exact but aliased.
4. Bilinear and bicubic — blend nearby pixels — interpolation over 2 or 4 taps per axis.
5. Integer scaling — exact whole-number blocks — resampling at 2:1 or 3:1 with no interpolation.
6. devicePixelRatio — real pixels per logical one — CSS reference pixels against device pixels.

8.98 Common wrong ideas

1. Wrong: more pixels always looks better. Right: beyond about 60 pixels per degree at your actual viewing distance, extra pixels cost power, bandwidth and frame time while adding very little you can see. The 2025 Nature Communications figure of 94 PPD raises that ceiling but does not remove it.
2. Wrong: 4K on a phone matters. Right: at 12 inches you need about 287 PPI to pass the 1 arcminute test, and a 6.8-inch flagship is already over 500 PPI, so 4K at that size mostly costs battery.
3. Wrong: OLED and LED are different backlights. Right: an "LED TV" is an LCD with an LED backlight, so it has one, while an OLED has no backlight at all because every sub-pixel is its own light source. The names are marketing and they are actively misleading.

4. Wrong: a bigger screen has better quality. Right: size and quality are independent, and making a panel bigger at the same resolution lowers PPI. What matters is resolution, density, contrast, colour accuracy and viewing distance together.
5. Wrong: a pixel is a single coloured dot. Right: it is a control unit of two, three or four separately driven sub-pixels, and on PenTile layouts some are shared with the neighbouring pixel.
6. Wrong: 1 ms response time means no lag. Right: response time is only the pixel transition, while input lag also covers polling, rendering, queueing, scanout and panel processing, usually ten to fifty times larger.
7. Wrong: a higher refresh rate only matters for games. Right: persistence blur scales with frame period, so 120 Hz makes ordinary text scrolling measurably clearer whatever the application is.
8. Wrong: HDR means brighter. Right: HDR means code values carry absolute luminance under a defined transfer function. A panel that accepts HDR but peaks at 350 candelas per square metre often looks worse than in standard mode.
9. Wrong: DSC is lossy so it degrades my picture. Right: it is lossy mathematically and visually lossless under ISO/IEC 29170 testing, a real trade-off for reference work and a non-issue for desktop and gaming use.
10. Wrong: running below native resolution is like having a smaller monitor. Right: it forces a resampling step that invents pixel values, which is why text goes soft. Only exact integer ratios avoid it.

8.99 Chapter summary in 20 lines

1. Light is a wave, and your eye reacts to roughly 380 to 700 nanometres of it, with no hard boundary at either end.
2. Three cone types peaking near 420, 530 and 560 nanometres reduce every spectrum to three numbers.
3. Because only three numbers survive, different spectra can look identical. That is metamerism, and it makes RGB screens possible.
4. A pixel is not a dot. It is a group of separately driven sub-pixels, most often a red, green and blue vertical stripe.
5. Resolution is a count of pixels; PPI is a density, the diagonal pixel count divided by the diagonal in inches.
6. Sharpness is decided by angular density: about 60 pixels per degree meets the 1 arcminute limit, and the PPI needed is about 3438 divided by viewing distance in inches.
7. A 24-inch 1080p panel is 91.8 PPI, a 27-inch 1440p is 108.8 PPI and a 6.8-inch phone is 505 PPI, and all can look correct at their own distances.
8. Colour is three numbers per pixel: 8 bits each gives 16.7 million combinations, 10 bits each about 1.07 billion.
9. A colour space such as sRGB, DCI-P3 or Rec.2020 states which exact red, green, blue and white those numbers mean.
10. Gamma exists because perception is not linear: mid grey code 128 emits about 21% of white, and that is deliberate.
11. One uncompressed 4K frame at 24 bits per pixel is 23.7 MiB, so 60 per second is about 1.49 GB.
12. An LCD blocks light from a permanent backlight, so its black is never truly black, and TN, VA and IPS trade speed, contrast and viewing angle.
13. An OLED emits its own light per sub-pixel, giving true black and very fast transitions, at the cost of burn-in and brightness limiting.
14. CRTs gave us the vocabulary every modern display standard still uses: scanline, blanking, refresh rate and overscan.
15. E-ink is bistable: once the pigment particles move, they hold position with no power at all.
16. A framebuffer stores pixels row by row, and the stride is the byte distance between row starts, including alignment padding.

17. Double buffering exists because scanout never stops; swapping during the vertical blank is what prevents tearing.
18. Bandwidth is pixels times bits times refresh rate: 4K at 60 Hz 8-bit is 11.94 Gbit/s and 4K at 120 Hz 10-bit is 29.86 Gbit/s, which is why HDMI 2.1, DisplayPort 2.1 and DSC exist.
19. Refresh rate and frame rate are separate numbers; vsync couples them harshly, while adaptive sync lets the panel wait for the frame.
20. Scaling always invents pixel values except at exact integer ratios, which is why non-native resolutions look soft and 1080p is clean on a 4K panel.

9

CHAPTER

KB

Sound, Magnets, Speakers and Phone Calls

9.0 What this chapter gives you

1. You will be able to say what sound physically is, with numbers.
2. You will be able to draw a speaker and name every part in it.
3. You will be able to explain why a magnet and a coil of wire move air.
4. You will be able to explain a microphone as a speaker run backwards.
5. You will be able to turn a wave into numbers and back, and say why 44,100 samples per second was chosen.
6. You will be able to calculate the size of a song from first principles.
7. You will be able to say what a codec throws away and why you do not hear it.
8. You will be able to trace a phone call from 1876 copper to a modern internet call, and say what changed at each step.
9. You will be able to explain why voice tolerates lost data but hates delay, and why file transfer is the exact opposite.
10. You will be able to explain noise cancelling without any magic.

9.1 What sound actually is

9.1.1 PLAIN – in simple words

1. Sound is air being pushed and pulled.
2. When something vibrates, it squashes the air next to it, and that squashed patch pushes the next patch, and so on outward.
3. A squashed patch is a **compression**. A stretched patch, with less air in it, is a **rarefaction**.
4. So sound is a travelling pattern of "more pressure, less pressure".
5. Nothing travels from the drum to your ear except the pattern. The air particles jiggle back and forth in place.
6. Cycles per second is the **frequency**, in hertz (Hz). It sets the pitch.
7. How hard the squeeze is, is the **amplitude**. It sets the loudness.
8. A young ear hears roughly 20 Hz to 20,000 Hz. The top falls with age.
9. Sound moves at about 343 metres per second, which is slow. That is why you see lightning long before you hear thunder.

9.1.2 PLAIN – a picture in your head

1. Picture a line of people packed shoulder to shoulder in a corridor.
2. Push the first person. They bump the next, who bumps the next.
3. A bump travels the whole corridor, but nobody walked anywhere. Each person leaned forward and came back.
4. Push rhythmically twice a second and bumps leave twice a second: 2 Hz.

5. Push harder for amplitude, faster for frequency. The gap between bumps is the **wavelength**.

Where this comparison breaks: real air particles are not in a line and are not touching. They already fly about randomly at hundreds of metres per second. Sound is a small organized pressure change riding on that permanent chaos. And the corridor carries the wave one way, while real sound spreads in a sphere.

9.1.3 PLAIN - a worked example

1. Take the note A above middle C, 440 Hz by convention.
2. Wavelength = speed divided by frequency. 343 / 440 = 0.78 m.
3. At the bottom of hearing, 343 / 20 = 17.15 m, longer than a bus.
4. At the top, 343 / 20,000 = 0.017 m, about 17 mm.
5. Across your hearing range, wave sizes differ by a factor of 1,000.
6. That single fact explains most of loudspeaker design. Hold on to it.

Frequency	Wavelength in air	About the size of
20 Hz	17.15 m	A five-storey building
100 Hz	3.43 m	A small room
1,000 Hz	34.3 cm	A forearm
10,000 Hz	3.43 cm	A thumb
20,000 Hz	1.72 cm	A fingernail

9.1.4 PLAIN - what is really happening inside

1. Loudness is measured in decibels (dB), on a scale that is odd on purpose.
2. The ear responds to ratios, not to absolute energy. Whisper to shout is about a million times in pressure, and taking the logarithm turns that into 120.
3. Every extra 6 dB doubles the pressure. Every extra 10 dB multiplies power by ten, and sounds roughly twice as loud.
4. Zero decibels is not silence. It is a chosen reference near the quietest sound a good young ear can detect.
5. Now the ear. The outer ear funnels pressure down the canal to the eardrum, and three tiny bones pass that movement on and strengthen it.
6. The last bone pushes fluid inside a spiral tube, the cochlea.
7. Inside is a strip that is stiff at one end and floppy at the other. High notes shake the stiff end, low notes the floppy end, so the cochlea sorts frequency by physical shape alone.
8. Hair cells on that strip bend as it moves and fire a nerve signal. Pressure has become nerve pulses.

9.1.5 TECHNICAL - the engineer's version

1. Sound in air is a longitudinal wave: particle motion is parallel to travel, unlike light, which is transverse.
2. Speed in dry air: c = 331.3 + 0.606 T m/s, T in degrees Celsius, so 343.4 m/s at 20 C. It depends on temperature, not pressure.
3. Sound Pressure Level: SPL = 20 log10 (p / p0), with p0 = 20 micropascals RMS. That reference is a standard, set near threshold at 1 kHz. Intensity level uses 10 log10 (I / I0) with I0 = 1e-12 W/m^2: 10 for power, 20 for pressure.
4. Characteristic acoustic impedance of air at 20 C is about 413 Pa*s/m.
5. Equal-loudness contours, standardized in ISO 226 (2003 revision), peak near 3 to 4 kHz, matching the quarter-wave resonance of the ear canal. Age-related high-frequency loss is called presbycusis.
6. The cochlea is about 35 mm long uncoiled, with roughly 3,500 inner hair cells as sensors and about 12,000 outer hair cells acting as an active amplifier. The frequency-to-place mapping is called tonotopy.

Sound	SPL	Pressure (RMS)
Threshold of hearing	0 dB	20 uPa
Normal speech at 1 m	60 dB	20 mPa
Busy road	80 dB	200 mPa
Rock concert, front	110 dB	6.3 Pa
Threshold of pain	130 dB	63 Pa

The honest version: "20 Hz to 20 kHz" is a convention, not a wall. Sensitivity fades gradually at both ends. Below about 20 Hz you feel pressure rather than hear a pitch, and the upper figure describes young ears in a laboratory.

9.1.6 WORDS - remember these

1. Compression — a squashed patch of air — pressure above ambient.
2. Rarefaction — a stretched patch of air — pressure below ambient.
3. Frequency — how fast it wobbles — cycles per second, unit hertz.
4. Amplitude — how big the wobble is — peak deviation from ambient pressure.
5. Wavelength — the length of one wobble — lambda = c / f, in metres.
6. Decibel — a loudness step — 20 log10 of a pressure ratio against 20 uPa.
7. Cochlea — the spiral in your ear — the organ performing a mechanical frequency analysis on the basilar membrane.

9.2 The speaker, in full

9.2.1 PLAIN - in simple words

1. A speaker turns an electrical wiggle into an air wiggle, using four things.
2. A **permanent magnet**, a lump of magnetized metal that always pulls.
3. A **voice coil**, thin wire wound into a tube, sitting in the magnet's gap.
4. A **cone** of stiff paper or plastic, glued to that coil.
5. A **suspension** of rubber and cloth, holding it centred so it can only move in and out.
6. Send electricity through the coil and the coil becomes a magnet itself.
7. Now there are two magnets. They push apart or pull together depending on which way the current flows, so reversing the current reverses the force.
8. The coil is glued to the cone, so the cone moves out, then in, squashing the air in front of it and then stretching it. That is a pressure wave.
9. If the signal wiggles 440 times a second, so does the cone, and you hear a 440 Hz note.
10. The speaker knows nothing about music. It copies the shape of an electrical signal into the shape of a pressure wave.

9.2.2 PLAIN - a picture in your head

1. Picture standing in a swimming pool holding a large flat tray.
2. Push it forward and you shove a wall of water away. Pull it back and water rushes in to fill the space.
3. Do that rhythmically and you make waves. Faster gives closer waves, further gives taller ones.
4. The tray is the cone, the water is the air, your arms are coil and magnet.
5. A big slow wave needs a big tray and a long movement, while tiny fast ripples want a small light paddle. That is why a speaker box has a big driver for bass and a small one for treble.

Where this comparison breaks: water is nearly incompressible, while the air wave exists precisely because air can be squashed. And your arms overpower the water, whereas a real speaker turns only about 0.5 to 2 percent of its electrical power into sound. The rest becomes heat in the coil.

9.2.3 PLAIN - a worked example

1. A bookshelf speaker is marked 8 ohms and 87 dB.
2. The 87 dB means: put in 1 watt, stand 1 metre away, measure 87 dB.
3. One watt into 8 ohms needs 2.83 volts, because power = volts squared divided by resistance, and 2.83 squared is 8.
4. Since +10 dB needs ten times the power, 10 W gives 97 dB and 100 W gives 107 dB.
5. So going from 87 to 107 dB needs one hundred times the amplifier power.
6. Swap in a 90 dB speaker and it reaches 97 dB on 5 W instead of 10 W. A sensitive speaker buys you more than a big amplifier does.

```
power for a target level, 8 ohm speaker
  sensitivity 87 dB at 1 W / 1 m, target 100 dB at 1 m

  difference   = 100 - 87 = 13 dB
  power ratio  = 10 ^ (13 / 10) = 19.95
  power needed = 19.95 W
  voltage      = sqrt(19.95 * 8) = 12.6 V RMS
  current      = 12.6 / 8 = 1.58 A RMS
```

9.2.4 PLAIN - what is really happening inside

1. Name the parts from the back forwards. The **magnet** is a ring or slug of ferrite or neodymium.
2. Steel shaped into a **top plate** and a **pole piece** squeezes its field into a narrow ring-shaped **gap**, where the field points straight outward.
3. The **voice coil** is wound on a stiff tube, the **former**, and sits there.
4. The **spider**, a corrugated cloth disc, keeps the coil centred, and the **surround**, a rubber ring, holds the cone edge. Together they are the suspension and act as a return spring.
5. The **cone** is glued to the former, a **dust cap** covers the middle, and the assembly bolts to a **basket**.
6. Physics law one, the Lorentz force: a wire carrying current in a magnetic field feels a sideways push, equal to field times current times wire length.
7. Because the field points outward all round the gap, and the wire runs round the gap, that push is straight forwards or backwards. Nothing is wasted.
8. Double the current and you double the force, so the cone faithfully copies the signal.
9. Physics law two, Faraday's law: a wire moving in a magnetic field generates a voltage. A moving cone therefore opposes its own drive, called back-EMF, and that is why a speaker also works, badly, as a microphone.
10. Loudness depends on the volume of air moved: cone area times travel. For the same loudness, dropping an octave needs four times the travel.
11. A 20 cm cone moving 5 mm shifts plenty of air but is too heavy to reverse 15,000 times a second. A 25 mm dome is light enough but shifts almost no air, so it makes no bass.
12. Hence two or three drivers, with a **crossover** splitting the signal.
13. Finally the box. When the cone pushes forward it pulls backward on the air behind it, producing an exactly opposite wave.
14. At high frequencies the wave is short and the cone body blocks the path. At low frequencies the wave is metres long, wraps round, and cancels.
15. A naked bass driver held in your hand makes almost no bass. The box exists to stop the back wave reaching the front.

```
cross-section of one driver (side view)

        surround
           |
   +-------v---------------------+
    \                           /   <- cone
```

```
      \      dust cap          /
       \       ___            /
        \     |   |          /
         +====|===|==========+       <- spider
              | | |
              | | |                  <- voice coil on former
         +----| | |----+
         | N  |_|_|  N |            <- gap, field points
         |   [pole ]   |               outward all round
         |   [piece]   |
         +-------------+            <- permanent magnet
```

9.2.5 TECHNICAL - the engineer's version

1. Force on the coil: F = B * l * I, with B the gap flux density in tesla, l the conductor length in the gap in metres, I the current in amperes. The product Bl, in tesla metres, is the driver's motor strength. Back-EMF is V = Bl * u, with u the cone velocity in metres per second.
2. Nominal impedance of 4, 6, 8 or 16 ohms is a **convention**, not a measured constant. An "8 ohm" driver measures about 5.5 to 6.5 ohms DC (Re) and swings from roughly 4 ohms to over 40 ohms across the band. The peak sits at free-air resonance Fs, where the coil moves fastest and generates the most back-EMF.
3. Sensitivity is stated as dB SPL at 1 m for 2.83 V RMS, which is 1 W only into 8 ohms. Into 4 ohms that is 2 W, so 4 ohm figures flatter the driver by 3 dB.
4. Typical values: 84 to 88 dB for a sealed bookshelf, 90 to 96 dB for a floorstander, 100 to 110 dB for a horn-loaded professional cabinet. Electroacoustic efficiency is 0.5 to 2 percent for a direct radiator.
5. Low-frequency behaviour is described by the Thiele/Small parameters, developed by Neville Thiele in 1961 and extended by Richard Small in the early 1970s: Fs, Qts, Vas, Sd, Xmax, Re, Le and Mms. Volume displacement Vd = Sd * Xmax, and for constant SPL required displacement rises as 1/f^2.
6. Crossovers are LC filters, passive after the amplifier or active before it, with slopes of 6, 12, 18 or 24 dB per octave. Fourth-order Linkwitz-Riley is the common modern choice, and a two-way typically crosses at 1.8 to 3 kHz.
7. Enclosures: sealed (acoustic suspension, 12 dB per octave rolloff, best transient behaviour), bass reflex (a Helmholtz resonator formed by port and box, 24 dB per octave, about 3 dB more output near tuning), and transmission line.
8. Why a phone cannot do bass, with numbers: Sd about 1 cm^2, Xmax under 0.3 mm, back volume under 1 cm^3, so Vd is around 0.03 cm^3. Producing 80 dB SPL at 50 Hz at half a metre needs tens of cm^3 of displacement. The phone is short by three orders of magnitude. It is a volume-of-air problem, so no amount of processing fixes it.
9. Other driver types:
    1. **Piezoelectric**: a ceramic disc that bends under voltage. No magnet or coil, cheap, high impedance, poor bass. Buzzers and alarms.
    2. **Balanced armature**: a magnetized reed balanced between two poles, driven by a coil, coupled by a rod to a tiny diaphragm. Efficient and small, standard in hearing aids since the 1940s and in in-ear monitors. Narrow bandwidth each, so in-ears stack two to twelve with crossovers.
    3. **Planar magnetic**: a thin film with a printed conductor between magnet arrays, so force spreads over the whole diaphragm and it flexes less.
    4. **Electrostatic**: a charged film between perforated stators driven at high voltage, needing a bias of one to six kilovolts. Very low moving mass and distortion, poor bass without a large panel.

Driver type	Moving part	Typical use
Dynamic (coil)	Cone plus coil	Almost everything
Balanced armature	Reed plus rod	In-ear monitors

Driver type	Moving part	Typical use
Piezo	Ceramic disc	Buzzers, alarms
Planar magnetic	Printed film	Studio headphones
Electrostatic	Charged film	High-end panels

The honest version: "the amplifier drives the speaker" is half the story. The speaker drives back, through back-EMF and its reactive impedance. A real amplifier must absorb returning energy, which is why output impedance must be low and why cable resistance measurably shifts response with some loads.

9.2.6 WORDS - remember these

1. Voice coil — the wire that gets pushed — the conductor in the magnetic gap, described by Re, Le and Bl.
2. Bl — motor strength — flux density times conductor length, in tesla metres.
3. Excursion — how far the cone travels — Xmax, peak linear displacement in mm.
4. Sensitivity — how loud for a given input — dB SPL at 1 m for 2.83 V RMS.
5. Crossover — the circuit splitting bass from treble — a filter network with a stated order and slope in dB per octave.
6. Bass reflex — a box with a hole — a Helmholtz resonator extending output.
7. Back-EMF — the speaker generating as it moves — the motional voltage Bl*u.
8. Thiele/Small parameters — the driver's spec sheet — the lumped-parameter model of a driver near and below resonance.

9.3 The microphone as a speaker in reverse

9.3.1 PLAIN - in simple words

1. A speaker turns electricity into movement, and movement into pressure. A microphone does the same three things in the opposite order.
2. Pressure arrives, pushes a thin sheet, and the sheet moves.
3. The moving sheet changes something electrical, and that change is the signal.
4. Way one: move a coil near a magnet. Moving wire in a field makes voltage. That is a **dynamic** microphone.
5. Way two: make the sheet one plate of a capacitor, which is two plates with a gap. Change the gap and the electrical behaviour changes. That is a **condenser** microphone.
6. Way three: build the whole thing on a chip the size of a grain of rice. That is a **MEMS** microphone, and it is the one in your phone.

9.3.2 PLAIN - a picture in your head

1. Think of a trampoline with a heavy rope ring tied underneath.
2. Somebody drops a ball. The trampoline dips and springs back, and the ring goes down and up with it.
3. Now imagine the ring passes through an invisible field that makes a small voltage whenever the ring moves.
4. Big bounce, big voltage. Fast wobble, fast wobbling voltage. The trampoline is the diaphragm and the ring is the coil: a dynamic microphone.

Where this comparison breaks: a real diaphragm moves nanometres for ordinary speech, the scale of a hundred atoms. And a trampoline keeps bouncing, whereas a good diaphragm is damped so it stops the instant the sound stops, or it would add its own ringing to everything.

9.3.3 PLAIN - a worked example

1. Take a condenser microphone: one fixed plate, one moving diaphragm.
2. Say the gap is 20 micrometres and the plate area is 3 cm^2. That is roughly 13 picofarads, a very small capacitor.
3. Put a fixed charge on it and never let the charge change.
4. Voltage across a capacitor = charge divided by capacitance.
5. Sound pushes the diaphragm in by 20 nanometres, so the gap shrinks by one part in a thousand and capacitance rises by about the same fraction.
6. Charge is fixed, so voltage falls by about one part in a thousand. Charged to 48 volts, that is a change of about 48 millivolts. That is the signal.
7. Notice the trick. We did not generate power from the sound. We used the sound to modulate a power supply that was already there.

9.3.4 PLAIN - what is really happening inside

1. The **dynamic** microphone is a small speaker used backwards: a light diaphragm with a coil glued to it, hanging in a magnet gap.
2. Sound moves the coil through the field and Faraday's law gives a voltage. It needs no power supply, because the sound itself does the work.
3. Because the sound must move the coil's mass, dynamics are less sensitive and less detailed high up, but rugged and happy with very loud sources.
4. The **ribbon** microphone replaces the coil with a corrugated strip of very thin aluminium which is itself the diaphragm. Very low mass, fragile, and naturally equally sensitive front and back.
5. The **condenser** needs an external voltage to charge its plates, plus an amplifier right beside the capsule, because the signal is tiny and the source impedance is enormous. That voltage is usually 48 volts sent down the same cable as the audio, called phantom power.
6. The **electret** is a condenser whose charge is permanently baked into a plastic film during manufacture, so no polarizing supply is needed. Electrets made microphones cheap enough for every telephone and laptop for forty years.
7. The **MEMS** microphone is that idea built by chip-making processes: a diaphragm a fraction of a millimetre across etched from silicon, with a perforated silicon backplate a few micrometres behind it.
8. Beside it sits a small chip that buffers the signal and often digitizes it immediately. The package is about 3 x 4 x 1 mm with a sound hole.
9. Because they are made photographically, thousands come out identical. That is what makes microphone arrays and beamforming possible in a phone.

9.3.5 TECHNICAL - the engineer's version

1. Dynamic output follows V = Bl * u, the same expression as speaker back-EMF. Sensitivity is typically 1 to 3 mV/Pa. The Shure SM58, introduced in 1966, is specified at 1.85 mV/Pa at 1 kHz into 150 ohms.
2. Condenser capsules are 10 to 60 pF. Capacitive reactance at 20 Hz for 20 pF is about 400 megohms, so the impedance converter, a FET or valve stage, must sit within millimetres of the capsule.
3. Phantom power is standardized as P48 in IEC 61938: 48 V nominal fed through two 6.81 kilohm resistors, one per leg of a balanced pair.
4. The electret was invented at Bell Labs by Gerhard Sessler and James West in 1962, and the foil-electret condenser became the most manufactured microphone type in history.
5. Polar pattern is a property of capsule venting, not of transducer type: omnidirectional if only the front is open, figure-of-eight if front and back are equally open, cardioid if the rear path is acoustically delayed.
6. Digital MEMS parts output PDM, a one-bit stream clocked at 1 to 3.072 MHz and decimated in the host codec, or I2S/TDM directly. PDM lets a microphone sit far from the processor on two wires without picking up noise.

7. Observation: `arecord -l` lists Linux capture devices, and `system_profiler SPAudioDataType` lists macOS inputs.

Parameter	Typical MEMS	Note
Sensitivity	-38 dBV/Pa	About 12.6 mV/Pa
Signal-to-noise ratio	64 to 70 dB(A)	Higher is better
Acoustic overload point	120 to 133 dB SPL	Where 10 % THD hits
Supply current	100 to 250 uA	Always-on capable

The honest version: "a microphone is a speaker in reverse" is exactly true for dynamic microphones and false for condenser and MEMS types. In a condenser the sound does not supply the signal energy; it modulates a bias already present. That is why a condenser needs power and a dynamic does not.

9.3.6 WORDS - remember these

1. Diaphragm — the thin sheet the sound pushes — the moving membrane.
2. Transducer — anything converting one energy to another — here acoustic to electrical, or the reverse.
3. Phantom power — voltage sent up the microphone cable — P48 per IEC 61938.
4. Electret — a permanently charged film — a dielectric with frozen-in polarization, removing the need for a bias supply.
5. MEMS — a microphone built like a chip — micro-electro-mechanical system with a silicon diaphragm and an integrated ASIC.
6. Acoustic overload point — where it distorts — the SPL at which total harmonic distortion reaches 10 percent.

9.4 From wave to numbers

9.4.1 PLAIN - in simple words

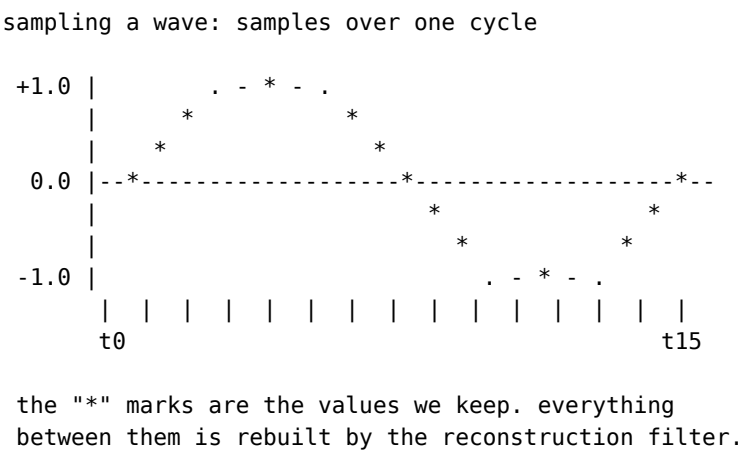
1. A computer cannot store a wiggle. It can only store numbers.
2. So we measure the wiggle over and over, very fast, and write the numbers down.
3. Each measurement is a **sample**: a snapshot of the voltage at one instant.
4. How many snapshots per second is the **sample rate**. A music CD takes 44,100 per second, of each of two channels.
5. How precisely each snapshot is written is the **bit depth**. A CD writes each one as a 16-bit number, one of 65,536 levels.
6. Measure often, measure precisely, write it down. That is the whole idea.
7. Playback means reading the numbers and rebuilding the voltage.

9.4.2 PLAIN - a picture in your head

1. Imagine a heart-rate monitor drawing a line on moving graph paper.
2. You cannot keep the paper. You may only write the height of the line at fixed moments: 42, 58, 71, 65, 44, 30, 41.
3. Later somebody plots your numbers as dots.
4. If the dots are close enough together, exactly one smooth curve fits them, and it is the original line.
5. If they are too far apart, many curves fit equally well, and the information is gone forever.

Where this comparison breaks: on paper you would join dots with straight lines and get a jagged shape. Real reconstruction does not. The mathematics says that if you sampled fast enough there is exactly one band-limited curve through those points, and a filter finds it. The output is a smooth wave, not a staircase.

9.4.3 PLAIN – a worked example



1. Now watch what goes wrong when we sample too slowly.
2. Suppose the real signal is 30,000 Hz but we take 44,100 samples a second.
3. The readings are indistinguishable from a 14,100 Hz signal, because 44,100 minus 30,000 is 14,100.
4. A tone you could barely hear reappears as a loud whistle in the middle of your hearing range. That is **aliasing**, and it cannot be removed afterwards, because both signals produced the same numbers.
5. So we put a filter before the converter that removes everything above half the sample rate. That is the **anti-aliasing filter**, and it is not optional.
6. You have seen aliasing in films: a wheel spinning forward appears to turn slowly backwards, because 24 frames a second cannot track the spokes.

9.4.4 PLAIN – what is really happening inside

1. There is a theorem behind all this, worth stating properly.
2. If a signal contains no frequency above F, sampling at any rate above 2F captures it perfectly, with nothing lost.
3. That is the Nyquist-Shannon sampling theorem. Harry Nyquist stated the underlying result at Bell Labs in 1928, and Claude Shannon published the modern form in 1949. "Perfectly" means perfectly, not approximately.
4. So to record up to 20,000 Hz we need more than 40,000 samples a second.
5. Why 44,100 and not 40,000? A practical accident. Before disks were big enough, early digital audio was stored on video tape using a PCM adaptor which packed samples into fake video lines.
6. 44,100 fits both video standards: 60 fields x 245 lines x 3 samples, and 50 fields x 294 lines x 3 samples, both give 44,100.
7. The extra 4,100 Hz gives the anti-aliasing filter room to roll off between 20 kHz and 22.05 kHz.
8. Now bit depth. Each sample is rounded to the nearest available level, and that **quantization error** appears as faint hiss.
9. Each extra bit halves the error, which is 6 dB quieter, so 16 bits gives about 96 dB of range and 24 bits about 144 dB.
10. One subtlety: for very quiet signals the error stops being random hiss and correlates with the music, sounding like gritty distortion.
11. The fix is **dither**: add a tiny amount of deliberate noise before rounding. It sounds wrong and it works, turning ugly distortion into benign hiss, and letting you hear signals quieter than one single level.

9.4.5 TECHNICAL – the engineer's version

1. Nyquist-Shannon: a signal band-limited to B hertz is completely determined by samples at $f_s > 2B$, and $f_s/2$ is the Nyquist frequency. Energy above $f_s/2$ folds back to $|f - f_s|$ and cannot be separated afterwards, so anti-alias filtering before the sample-and-hold is mandatory.

2. Signal-to-quantization-noise ratio for an ideal uniform quantizer with a full-scale sine: SQNR = 6.02 N + 1.76 dB. For N = 16 that is 98.09 dB. The commonly quoted 96 dB is 6.02 x 16 without the 1.76 term.
3. Dither is usually triangular PDF noise at 2 LSB peak-to-peak, fully decorrelating the error at the cost of 4.77 dB more noise. Noise-shaped dither moves that noise above 15 kHz where the ear is least sensitive.
4. Two converter architectures dominate.
    1. **Successive approximation (SAR)**: a binary search. The held sample is compared against half full scale, then a quarter, and so on, one bit per clock, so N bits take N comparisons and latency is one conversion.
    2. **Sigma-delta**: samples at 64 to 256 times the target rate with a coarse, often one-bit, quantizer inside a feedback loop that shapes quantization noise up out of the audio band, then decimates digitally. This trades speed for resolution and is why 24-bit converters are cheap. Nearly every audio ADC and DAC since the mid-1990s is sigma-delta.
5. Oversampling also lets the analogue anti-alias filter be gentle, because the first alias lands 64 times higher. The steep filtering happens in the digital decimator, where it can be made exact and phase-linear.

Sample rate	Nyquist limit	Where it is used
8 kHz	4 kHz	Telephone, G.711
16 kHz	8 kHz	Wideband voice, AMR-WB
44.1 kHz	22.05 kHz	CD, most music files
48 kHz	24 kHz	Video, broadcast, most OS
96 kHz	48 kHz	Studio production
192 kHz	96 kHz	Studio, largely marketing

Experts disagree here. A 2007 AES paper by E. Brad Meyer and David R. Moran found listeners could not distinguish high-resolution playback from the same signal passed through a 16-bit 44.1 kHz loop. A 2016 AES meta-analysis by Joshua Reiss pooled many studies and found a small but statistically significant ability to discriminate, especially with training. Any effect is small, and higher rates are most clearly useful during production, where repeated processing accumulates error.

9.4.6 WORDS - remember these

1. Sample — one measurement of the wave — one quantized amplitude at one instant.
2. Sample rate — how often we measure — samples per second, in hertz.
3. Bit depth — how precisely we measure — bits per sample, giving 2^N levels.
4. Nyquist frequency — the highest note recordable — half the sample rate.
5. Aliasing — a high note pretending to be a low one — spectral folding of content above fs/2 back into the baseband.
6. Dither — helpful noise added on purpose — low-level TPDF noise decorrelating quantization error from the signal.
7. Sigma-delta — the converter in everything — oversampling noise-shaping converter plus digital decimation filter.

9.5 From numbers back to wave

9.5.1 PLAIN - in simple words

1. Playback runs the recording chain backwards.
2. A chip reads the numbers in order, at exactly the right speed, and sets an output voltage to match each one. That is the **DAC**, the digital-to-analogue converter.
3. Its raw output is a series of steps, because it holds each value until the next arrives.

4. A filter smooths those steps back into the original wave. That is the **reconstruction filter**.
5. The smoothed wave is correct but far too weak to move a speaker.
6. So it goes into an **amplifier**, which uses the small wave as a template and builds a big copy, taking the energy from the power supply.

9.5.2 PLAIN - a picture in your head

1. Imagine a precise but tiny artist painting a perfect miniature.
2. It is correct in every detail but one centimetre across, so nobody at the back of the hall can see it.
3. A second worker copies it onto a wall, stroke for stroke, a thousand times bigger, using buckets from a store room.
4. The tiny artist is the DAC, the wall painter is the amplifier, the store room is the power supply.
5. The painter adds nothing new, and a mistake in the miniature becomes a huge mistake on the wall. The paint comes from the store room, not the miniature.

Where this comparison breaks: a real amplifier is not a passive copier. It is a feedback system constantly comparing its output against its input and correcting the difference far faster than the music changes. It also adds a little of its own noise and distortion.

9.5.3 PLAIN - a worked example

1. A laptop headphone socket puts out at most about 1 volt RMS, with a source impedance of perhaps 10 ohms.
2. Headphone A is a 32 ohm in-ear rated 105 dB per milliwatt. Power = volts squared divided by ohms, so at 0.8 V that is 0.64 / 32 = 20 milliwatts. It will be painfully loud.
3. Headphone B is a 250 ohm studio model rated 96 dB per milliwatt. At 1 V, power = 1 / 250 = 4 milliwatts, which is +6 dB over one milliwatt, giving about 102 dB. Loud enough, but with little headroom.
4. Now the source impedance problem. A 10 ohm output feeding a 32 ohm headphone is a divider, losing about a quarter of the voltage.
5. Worse, headphone impedance varies with frequency, so a high source impedance changes the tone. The design rule is to keep source impedance under one eighth of the load.
6. That is why 250 and 600 ohm headphones want a real amplifier, and why 16 ohm in-ears sound tonally different from a high-impedance output.

9.5.4 PLAIN - what is really happening inside

1. The DAC receives a stream of numbers plus a clock saying when each is due.
2. The clock matters enormously. If its timing wobbles, the wave is rebuilt at slightly wrong moments. That is **jitter**.
3. Most audio DACs oversample rather than building the exact voltage in one step.
4. A digital filter invents extra samples between the real ones, raising the internal rate by 8, 16, 64 or 128 times.
5. Then a very fast, very coarse converter produces an output at that high rate, with its errors pushed up into ultrasonic frequencies.
6. Then a gentle analogue filter removes the ultrasonic mess. It is the sigma-delta idea run in reverse.
7. The output is typically 1 to 2 volts RMS at a few milliamps: line level, meant for another circuit, not for a speaker.
8. The amplifier keeps that voltage shape but supplies amperes.
9. A class AB amplifier uses transistors that are always partly on, wasting power as heat, at roughly 50 to 70 percent efficiency.
10. A class D amplifier switches fully on and fully off hundreds of thousands of times a second, varying how long it stays on, and a filter averages that into the wanted wave. Efficiency reaches 85 to 95 percent.
11. Class D is why a phone, a soundbar and a car stereo can be loud without a heatsink the size of a brick.

9.5.5 TECHNICAL - the engineer's version

1. The raw DAC output is a zero-order hold, imposing a sinc amplitude droop of sin(x)/x with x = pi*f/fs, about -3.2 dB at 20 kHz with fs = 44.1 kHz. Oversampling DACs correct it in the digital interpolation filter.
2. Reconstruction is usually a linear-phase FIR filter at 8x to 128x oversampling followed by a second or third order analogue filter. The choice between linear phase (pre-ringing) and minimum phase (phase shift) is selectable on many current DAC chips and is a matter of taste.
3. Clock jitter budget: below roughly 100 picoseconds RMS for 16-bit performance at 20 kHz, and about 1 picosecond for 24-bit. That is why asynchronous USB audio, where the DAC owns the clock, replaced adaptive USB audio.
4. Headphone impedances in use: 16 to 32 ohms for in-ears and headsets, 32 to 80 ohms for consumer over-ears, 250 and 600 ohms in the Beyerdynamic DT770 and DT880 families, 300 ohms for the Sennheiser HD 600 and HD 650. The high-impedance variants were designed for professional equipment with high output voltage. Damping factor is load impedance divided by source impedance, and the "rule of eighth" convention says keep it above 8.
5. What a product sold as "a DAC" actually contains: a USB, S/PDIF or Bluetooth receiver, a clock, the converter silicon, an analogue output stage, a volume control, usually a headphone amplifier, and a power supply. The converter chip is often a few dollars of the bill of materials, and most audible difference between such products comes from the analogue output stage, the headphone amplifier and the volume control, not the chip named on the box.
6. Observation: `cat /proc/asound/card0/pcm0p/sub0/hw_params` on Linux shows the rate, format and buffer actually in use while audio plays. On macOS, Audio MIDI Setup shows the current device rate and bit depth.

Class	How it works	Efficiency
A	Always fully conducting	20 to 30 percent
AB	Mostly on, brief overlap	50 to 70 percent
D	Switching at 300 kHz plus	85 to 95 percent
G / H	AB with stepped rails	60 to 80 percent

9.5.6 WORDS - remember these

1. DAC — the chip turning numbers into voltage — digital-to-analogue converter, usually oversampling sigma-delta.
2. Zero-order hold — holding each value until the next — the staircase output causing sinc amplitude droop.
3. Reconstruction filter — the smoother — the low-pass filter recovering the unique band-limited waveform.
4. Jitter — timing wobble — deviation in sample clock edge timing, in picoseconds RMS.
5. Line level — a weak but correct signal — nominally 2 V RMS consumer or 1.228 V RMS (+4 dBu) professional.
6. Class D — the efficient switching amplifier — pulse-width modulated output stage with an LC output filter.
7. Damping factor — how firmly the amplifier grips the driver — load impedance divided by amplifier output impedance.

9.6 Storing audio

9.6.1 PLAIN – in simple words

1. The simplest way to store sound is to write every sample in order, with nothing clever done to it. That is **PCM**, pulse-code modulation.
2. A WAV file is mostly raw PCM with a small label saying how fast, how many bits and how many channels.
3. Raw PCM is honest and huge, so we compress it, in two different ways.
4. **Lossless** compression packs the data tighter and gives back every original number exactly. FLAC and ALAC do this, roughly halving the size.
5. **Lossy** compression throws information away permanently, choosing parts you are unlikely to notice. MP3, AAC and Opus do this, cutting by ten or more.
6. A **codec** is the method of coding and decoding. A **container** is the file format holding the coded data plus the title, artist and length.
7. The same codec can live in different containers, and one container can hold different codecs. That is the most confusing thing about audio files.

9.6.2 PLAIN – a picture in your head

1. Imagine posting a long letter. Raw PCM is writing every word out in full.
2. Lossless compression is shorthand. The reader expands it back to the same letter word for word. Nothing lost, thinner envelope.
3. Lossy compression is writing a summary. The reader gets the meaning and never knows which words you dropped.
4. And you cannot un-summarize. Converting a summary back into a full letter just produces a fatter summary.
5. That is why converting an MP3 to FLAC restores nothing. It stores the damaged version losslessly.

Where this comparison breaks: a written summary drops whole ideas, which you would notice. A good audio codec drops only sounds physically masked by louder sounds happening at the same moment, which your ear could not have detected. It is closer to not printing the parts of a photograph that are behind a wall.

9.6.3 PLAIN – a worked example

1. Step one: 44,100 samples per second.
2. Step two: 16 bits each, so 44,100 x 16 = 705,600 bits per second per channel.
3. Step three: stereo doubles it, giving 1,411,200 bits per second. That is the famous 1.411 Mbps.
4. Step four: divide by 8 for bytes, giving 176,400 bytes per second.
5. Step five: a three-minute song is 180 seconds, so 176,400 x 180 = 31,752,000 bytes, about 31.8 MB, shown as 30.3 MiB by a file manager.
6. The same song as a 128 kbps MP3 is 128,000 / 8 x 180 = 2,880,000 bytes, about 2.9 MB, roughly eleven times smaller.
7. As FLAC at a typical 60 percent, about 19 MB. As 64 kbps Opus, about 1.4 MB.

```
CD audio bitrate, from first principles

  44100 samples/s
  x    16 bits/sample        = 705600 bits/s per channel
  x     2 channels           = 1411200 bits/s = 1.4112 Mbit/s
  / 8                        = 176400 bytes/s
  x 180 s (a 3 minute song)  = 31752000 bytes
                             = 31.75 MB  (30.28 MiB)

  a full 74 minute CD:
  176400 x 4440 s            = 783216000 bytes = 783 MB
```

9.6.4 PLAIN – what is really happening inside

1. Lossless codecs work in two stages. First, prediction: guess the next sample from the previous few using a small polynomial or adaptive filter. Music is smooth, so the guess is usually close.
2. Second, store only the error between guess and truth. Those errors are small numbers clustered near zero, and they compress very well with a code that gives short bit patterns to common values.
3. Nothing is discarded, so decoding reproduces the original bit for bit.
4. Lossy codecs add a third idea: a model of your ear.
5. Fact one, the threshold of hearing: very quiet sounds are inaudible, and the level at which that happens depends strongly on frequency.
6. Fact two, simultaneous masking: a loud tone makes nearby quieter tones inaudible. A loud 1 kHz tone can hide a tone 40 dB quieter at 1.1 kHz.
7. Fact three, temporal masking: for a few milliseconds before, and up to about 200 milliseconds after a loud sound, quieter sounds near it are inaudible.
8. So the encoder splits the signal into frequency bands, works out how much noise each band can hide, and spends just enough bits per band to keep its own quantization noise under that hidden threshold.
9. The bits go where the ear is looking, and everywhere else gets almost nothing.
10. That is why 128 kbps MP3 is not "lower quality audio". It is full-range audio with its errors deliberately parked underneath louder sounds.
11. When the bitrate is too low the encoder runs out of bits, errors climb above the masking threshold, and you hear it: swishy cymbals, a metallic ring on voices, and a hard cut-off of high frequencies.

9.6.5 TECHNICAL – the engineer's version

1. WAV is a container built on the Microsoft and IBM RIFF chunk format from 1991: a `fmt` chunk declares rate, channels and bits, a `data` chunk holds interleaved PCM. AIFF is Apple's 1988 equivalent on the Electronic Arts IFF format, big-endian where WAV is little-endian.
2. MP3 is formally MPEG-1 Audio Layer III, from the Fraunhofer Institute for Integrated Circuits in Erlangen and the University of Erlangen-Nuremberg, with Karlheinz Brandenburg completing the underlying doctoral work in 1989. ISO/IEC 11172-3 reached committee draft in 1991, was finalized in 1992 and published in 1993. MPEG-2 Layer III, ISO/IEC 13818-3, added lower sample rates and was published in 1995.
3. The `.mp3` extension was chosen by the Fraunhofer team on 14 July 1995; before that the files used `.bit`. The last US patent covering MP3 expired on 16 April 2017.
4. MP3 supports 32, 44.1 and 48 kHz under MPEG-1 and 16, 22.05 and 24 kHz under MPEG-2, with 14 fixed bitrates from 32 to 320 kbit/s. The 128 kbit/s convention is an 11:1 ratio against 1,411.2 kbit/s CD audio. Internally it uses a 32-band polyphase filter followed by an MDCT of 18 or 6 lines, non-uniform quantization, Huffman coding, and a bit reservoir.
5. AAC was standardized as MPEG-2 Part 7, ISO/IEC 13818-7, in 1997, then as MPEG-4 Part 3, ISO/IEC 14496-3, in 1999, developed jointly by AT&T Labs, Dolby, Fraunhofer IIS and Sony, with Nokia joining as co-licensor in 2002. It uses a pure MDCT filter bank with 2048 or 256 sample windows.
6. Opus was standardized by the IETF as RFC 6716 on 10 September 2012, merging SILK, a linear-prediction speech coder from Skype, with CELT, a low-latency MDCT coder from Xiph.Org. It spans 6 to 510 kbit/s with frames of 2.5 to 60 ms and a default algorithmic delay of 26.5 ms, reducible to 5 ms. It is the default codec for WebRTC.
7. FLAC reached version 1.0 on 20 July 2001, written by Josh Coalson, and typically compresses to 50 to 70 percent of the original. In December 2024 it was formally specified as IETF RFC 9639. ALAC is Apple's equivalent, introduced in 2004 and open sourced in 2011.
8. Codec versus container, precisely: a codec defines a bitstream syntax and a decoding process, while a container defines how coded streams, metadata, timestamps and index tables are packed into a file. MP4 and M4A are the same container, the ISO base media file format, and may hold AAC or ALAC. Ogg and WebM commonly hold Opus. The `.mp3` case is unusual: a bare stream with no real container,

which is why its metadata is a bolt-on called ID3.

Format	Type	Typical bitrate	Best use
PCM in WAV	None	1,411 kbps	Editing, mastering
FLAC	Lossless	700 to 1,000 kbps	Archiving music
ALAC	Lossless	700 to 1,000 kbps	Apple archiving
MP3	Lossy	128 to 320 kbps	Legacy compatibility
AAC	Lossy	96 to 256 kbps	Streaming, video, iOS
Opus	Lossy	6 to 128 kbps	Calls, WebRTC, chat

```
# what does this file actually contain?
ffprobe -hide_banner song.m4a
# container = mov,mp4,m4a   codec = aac   48000 Hz   stereo

# convert losslessly, keeping every sample
ffmpeg -i track.wav -c:a flac -compression_level 8 track.flac

# encode speech efficiently
opusenc --bitrate 32 --speech voice.wav voice.opus
```

The honest version: "MP3 removes frequencies above 16 kHz" is a symptom, not the method. Low-bitrate encoders often apply a lowpass because high frequencies are expensive and least missed, but that is a bit-allocation decision by one encoder, not part of the standard. Implementation detail: LAME, the dominant open-source MP3 encoder, and Fraunhofer's own encoder make different choices at the same bitrate, and LAME's variable-bitrate presets beat fixed 128 kbps.

9.6.6 WORDS - remember these

1. PCM — the raw numbers — pulse-code modulation, uncompressed linear samples.
2. Codec — the coding method — the coder/decoder pair defining a bitstream.
3. Container — the box the data sits in — a file format multiplexing streams, metadata and timing.
4. Lossless — exactly what you put in — bit-exact reconstruction of the PCM.
5. Lossy — some detail gone for good — perceptually irrelevant information discarded permanently.
6. Masking — a loud sound hides a quiet one — simultaneous and temporal elevation of the hearing threshold.
7. MDCT — the maths inside every modern codec — modified discrete cosine transform with 50 percent overlapping windows.

9.7 How recorded sound was stored before computers

9.7.1 PLAIN - in simple words

1. Before computers, sound was stored as a physical shape or a magnetic pattern.
2. The phonograph stored it as a wiggly groove cut into a spinning cylinder.
3. The groove is literally the waveform. You can see the sound under a microscope.
4. Magnetic tape stored it as magnetized specks on a plastic ribbon.
5. The vinyl record used a groove again, on a flat pressed disc.
6. The compact disc stored numbers instead, as microscopic dents read by a laser.
7. Only the last is digital. The first three are direct physical copies of the wave.

9.7.2 PLAIN - a picture in your head

1. Picture drawing a wobbly line along a very long road with chalk.
2. Later, walk the same road dragging your finger along the chalk line.

3. Your finger is forced to wobble exactly as the line wobbles.
4. Connect the finger to a drum skin, and the skin wobbles, and the air wobbles, and you hear the original sound.
5. That is a record player. No code, no numbers, no cleverness. The groove is a drawing of the pressure wave, and the needle is pushed by it.

Where this comparison breaks: a record groove wobbles sideways, and a stereo groove wobbles both walls independently at 45 degrees. Records are also not cut flat: bass is reduced and treble boosted when cutting, then reversed on playback, so grooves stay narrow and surface noise falls. That curve is the RIAA equalization standard from 1954.

9.7.3 PLAIN - a worked example

1. Take a 33 and a third rpm LP, 12 inches across.
2. It turns once every 1.8 seconds, so the outer groove passes the needle at about 50 cm per second and the inner groove at only about 21 cm per second.
3. That is why the last track on a side often sounds slightly worse. Less groove length is available per second of music.
4. A side holds about 25 minutes, and the microgroove standard allows a stylus tip radius of 25 micrometres.
5. A CD instead runs at constant linear velocity, 1.2 to 1.4 m/s, so it spins faster in the middle and slower at the edge, and every part of the surface carries data at the same density.

9.7.4 PLAIN - what is really happening inside

1. Magnetic tape is plastic ribbon coated with billions of needle-shaped iron oxide particles.
2. Each particle is a tiny permanent magnet holding whichever direction you last pushed it into.
3. The record head is an electromagnet with a very narrow gap, and audio current through it makes a field at that gap.
4. Tape passes the gap, and particles leaving the field freeze in whatever magnetization the field last gave them: strongly aligned for a loud moment, weakly aligned for a quiet one.
5. On playback the moving magnetized tape passing another gap induces a voltage in a coil, by Faraday's law, the same law as the microphone.
6. Hold this for Chapter 12: a hard disk is the same idea with the tape replaced by a rigid spinning platter, a head flying nanometres above it, and the magnetization standing perpendicular to the surface rather than along it. The physics of "particles keep the direction you last pushed them" is unchanged.
7. The compact disc is different, because it stores numbers rather than a wave.
8. A spiral track of pits is pressed into plastic and coated with reflective aluminium. A laser shines up at it and a photodiode watches the reflection.
9. A pit is about a quarter wavelength deep, so light from a pit edge cancels light from the flat land beside it and the brightness drops sharply.
10. The honest version: a pit is not a 1 and a land is not a 0. Every transition from pit to land or land to pit is a 1, and no transition is a 0. That coding is called NRZI, and it exists because an edge is far more reliably detected than an absolute level.

9.7.5 TECHNICAL - the engineer's version

1. Edouard-Leon Scott de Martinville patented the phonautograph on 25 March 1857. It traced sound onto soot-blackened paper but could not play back. The oldest recovered recording from it, of "Au Clair de la Lune", dates from 9 April 1860 and was recovered digitally in 2008.
2. Thomas Edison announced the phonograph on 21 November 1877 and demonstrated it on 29 November 1877, recording on tinfoil wrapped round a cylinder.
3. Emile Berliner moved to flat discs, beginning commercial gramophone production in 1892 with five-inch records and moving to shellac in 1895.

4. Valdemar Poulsen demonstrated magnetic recording on steel wire at the end of the 1890s. Fritz Pfleumer patented magnetic tape in Germany in 1928, AEG built it into the Magnetophon with BASF tape in the mid-1930s, and American engineers, notably Jack Mullin, brought captured machines home after the Second World War, after which Ampex productionized them.
5. Columbia Records introduced the 33 and a third rpm microgroove LP at a press conference on 21 June 1948. Stereo LPs followed from 1957.
6. Compact Disc Digital Audio, the Red Book, was published by Philips and Sony in 1982 and adopted as IEC 60908 in 1987. The first commercial CD release was on 17 August 1982, and the first fifty titles went on sale in Japan on 1 October 1982.
7. EFM, devised by Kees Schouhamer Immink, maps each 8-bit byte to a 14-bit pattern chosen so runs of zeros are between 2 and 10 long, keeping pit lengths within what the optics resolve while leaving enough transitions for clock recovery. Three merging bits join the patterns.
8. CIRC is Sony's cross-interleaved Reed-Solomon code. Interleaving spreads consecutive samples across the disc so a scratch destroys one symbol from many codewords rather than many symbols from one, which is exactly what Reed-Solomon repairs. It fully corrects bursts of about 4,000 bits, roughly 2.5 mm of track.

CD specification	Value
Disc diameter	120 mm
Laser wavelength	780 nm, infrared
Track pitch	1.6 micrometres
Maximum playing time	74 min 33 s
Channel coding	EFM, eight-to-fourteen
Error correction	CIRC, Reed-Solomon

9.7.6 WORDS - remember these

1. Groove — the wiggly channel in a record — a mechanically modulated spiral carrying the waveform laterally.
2. Stylus — the needle — the tip tracing the groove, typically 25 um radius on microgroove records.
3. Coercivity — how hard a magnet is to flip — the reverse field needed to demagnetize a recorded particle.
4. Pit and land — the dents and the flat between them — features whose transitions encode 1 bits under NRZI.
5. EFM — how bytes become pit patterns — eight-to-fourteen modulation with a run-length constraint of 2 to 10.
6. CIRC — the CD's scratch repair — cross-interleaved Reed-Solomon coding.
7. RIAA curve — the record equalization standard — the 1954 pre-emphasis and playback de-emphasis specification.

9.8 Audio inside a computer

9.8.1 PLAIN - in simple words

1. Inside a computer, sound is a stream of numbers moving to a deadline.
2. A chip called the audio codec holds the converters, in both directions.
3. Your program writes numbers into a shared area of memory called a **buffer**, and the hardware reads from it at a fixed, unforgiving rate.
4. If your program is late even once, the hardware runs out of numbers and plays silence or repeats old data. That gap is heard as a click.
5. So the whole design problem of computer audio is: never be late, ever.

6. A bigger buffer gives more safety margin but more delay before you hear anything. That delay is **latency**.

9.8.2 PLAIN - a picture in your head

1. Think of a conveyor belt feeding a machine that must never stop.
2. You stand at one end putting parts on. The machine takes one every second, forever.
3. A long belt lets you wander off for a minute, but a part you place now is not used for a long time.
4. A short belt means what you place is used almost immediately, which is what you want when playing a guitar through the computer, but blink at the wrong moment and the machine starves and jams.
5. Length buys safety, shortness buys responsiveness, and you cannot have both.

Where this comparison breaks: the computer's belt is circular. It is a ring buffer, written at one point and read at another chasing it round. Nothing is added to an end, and if the reader catches the writer you get the click.

9.8.3 PLAIN - a worked example

1. Buffering latency = frames in the buffer divided by the sample rate.
2. At 48,000 samples a second, a 256-frame buffer is 256 / 48000 = 5.33 milliseconds.
3. But there are normally at least two such periods, one playing and one being filled, so real output latency is around 10.7 ms.
4. Add the input side for live monitoring and you are near 21 ms.
5. Sound travels 343 metres a second, so 21 ms is the delay of standing 7.2 metres from an amplifier. Musicians notice about 10 ms and hate 25 ms.
6. Drop to a 64-frame buffer and each period is 1.33 ms. Beautiful, and it will crackle on a busy machine.

Buffer size	At 48 kHz	Typical use
32 frames	0.67 ms	Live guitar, low latency
128 frames	2.67 ms	Studio recording
512 frames	10.7 ms	Mixing, general use
2048 frames	42.7 ms	Video playback, games

9.8.4 PLAIN - what is really happening inside

1. The codec chip contains ADCs, DACs, a mixer, headphone and speaker amplifiers, and often a microphone bias supply.
2. It talks to the processor over a small serial bus: I2S or TDM for the audio samples, I2C or SoundWire for control commands.
3. A DMA engine, which is hardware that moves memory without troubling the CPU, walks round the ring buffer handing samples to the codec on a timer.
4. When it finishes a period it raises an interrupt, the driver wakes the audio thread, and that thread must fill the next period before the engine arrives.
5. If it does not, that is an **underrun**, called an **xrun** in ALSA.
6. Why does an underrun click rather than just go quiet? Because the waveform jumps instantly to zero, and a vertical step contains energy at every frequency at once. Your ear hears a broadband snap.
7. Sample rate conversion is the other everyday complication: your file is 44.1 kHz and your device runs at 48 kHz, so somebody must resample.
8. The ratio is 160 to 147, so good resampling needs a long filter and costs CPU, while bad resampling adds audible aliasing.
9. Above the driver sits a mixer, because many applications want the one device at once. It resamples everything to a common rate, applies per-stream volume, sums, and hands a single stream to the hardware.

9.8.5 TECHNICAL - the engineer's version

1. Historical anchors: the Creative Labs Sound Blaster arrived in 1989 and set the PC convention for a decade. Intel's AC'97 specification came in 1997 and split the digital controller from the analogue codec. Intel High Definition Audio, codenamed Azalia, replaced it in 2004 and remains the PC standard, usually paired with a Realtek ALC-series codec.
2. WASAPI has two modes. Shared mode goes through the Windows audio engine, is resampled to a fixed device format and mixes with other applications. Exclusive mode hands the device to one application bit-perfectly. ASIO bypasses the Windows engine entirely, which is why it survives in professional work.
3. On Linux, ALSA is the kernel driver interface. PulseAudio, from 2004, added per-application volume and network transparency but had poor low-latency behaviour, while JACK served professional audio. PipeWire replaced both with one graph-based server, becoming the default in Fedora 34 in April 2021 and in Ubuntu 22.10 in October 2022.
4. Core Audio expresses everything as float32 at the device rate, and is the only one of the three major platforms where the low-latency path is also the ordinary path.
5. Rule of thumb for round-trip latency: buffer periods, plus converter group delay (sigma-delta decimation and interpolation filters add roughly 0.5 to 1.5 ms each way), plus any resampling delay. Manufacturers quote only the buffer figure, so measured round-trip is always worse than the control panel.

Stack	Platform	Note
Core Audio	macOS and iOS	Since Mac OS X 10.0, 2001
WASAPI	Windows	Since Vista, 2007
ASIO	Windows	Steinberg, 1997, pro use
ALSA	Linux kernel	Kernel driver layer
PipeWire	Linux userspace	Fedora 34 default, 2021

```
# Linux: what hardware exists, and what it is doing now
aplay -l
cat /proc/asound/card0/pcm0p/sub0/hw_params
pw-top                 # live quantum, rate and xrun counts

# macOS
system_profiler SPAudioDataType
```

9.8.6 WORDS - remember these

1. Buffer — the waiting area for samples — a ring buffer read by DMA on a fixed timer.
2. Latency — the delay before you hear it — frames divided by sample rate, times the number of periods, plus converter delay.
3. Underrun / xrun — the moment audio runs out — the DMA pointer overtaking the application's write pointer.
4. Codec chip — the audio front end on the board — integrated ADC, DAC, mixer and amplifiers on an I2S/I2C interface.
5. Resampling — changing the sample rate — polyphase interpolation and decimation, 147:160 between 44.1 and 48 kHz.
6. Exclusive mode — one application owns the device — bit-perfect path bypassing the system mixer, WASAPI exclusive or ASIO.

9.9 Phone calls, part 1 - the old way

9.9.1 PLAIN - in simple words

1. The original telephone is astonishingly simple.
2. A microphone at one end changes the electric current in a wire, following the sound.
3. That same current runs down the wire through a small speaker at the far end, which moves and makes the sound again.
4. No code, no numbers, no computer. The wire carries a shrinking and growing electrical copy of the pressure wave.
5. The pair of copper wires from your house to the exchange is the **local loop**.
6. To connect any phone to any other, you go through an **exchange**, a building whose whole job is joining wires together.
7. At first humans did the joining, plugging cords into sockets. Later machines did it, driven by the clicks your dial sent down the line.
8. The crucial point: while you talked, a continuous electrical path existed from your house to theirs, and nobody else could use it. That is **circuit switching**.

9.9.2 PLAIN - a picture in your head

1. Think of the old phone network as a railway that gives you a private track.
2. To make a call, the railway builds a track from your station to theirs, and it stays yours for the whole call, even during silences.
3. When you hang up, the track is dismantled and the pieces return to the pool.
4. This is wonderful for quality. Nothing can get in your way, so the delay never changes and the sound never breaks up.
5. It is terrible for efficiency. About half of a conversation is silence, and all that track sits idle.

Where this comparison breaks: after the 1960s the private track stopped being a continuous piece of copper. Your voice was digitized at the exchange and given a fixed repeating slot on a shared high-speed line. It behaves exactly like a private track, with guaranteed capacity and constant delay, but physically it shares one cable with dozens of other calls.

9.9.3 PLAIN - a worked example

1. Here is how a phone call became exactly 64,000 bits a second.
2. The telephone band is 300 Hz to 3,400 Hz, chosen to keep speech intelligible while using as little of the wire as possible.
3. To sample up to about 4,000 Hz you need at least 8,000 samples a second, so 8 kHz was chosen, and each sample is stored in 8 bits.
4. 8,000 x 8 = 64,000 bits per second. That is one voice channel, called a DS0.
5. Why is 8 bits acceptable when a CD needs 16? Because of **companding**.
6. Instead of 256 evenly spaced levels, the steps are fine near zero and coarse near the loudest values, following a logarithmic curve.
7. Quiet passages get fine resolution where you would notice the noise, and loud passages get coarse steps where loudness hides them.
8. An 8-bit companded sample carries roughly the useful range of a 12 to 13-bit linear sample.

```
one voice channel, from air to bits

 speech -> 300-3400 Hz filter -> sample at 8 kHz
        -> compand to 8 bits  -> 64000 bit/s (one DS0)

 T1 = 24 DS0 + 1 framing bit
    = 193 bits every 125 us = 1.544 Mbit/s

 E1 = 32 timeslots x 8 bits
```

```
    = 256 bits every 125 us = 2.048 Mbit/s
      (TS0 framing, TS16 signalling, 30 voice channels)
```

9.9.4 PLAIN – what is really happening inside

1. The original transmitter was the **carbon microphone**: a chamber of carbon granules behind a diaphragm.
2. Squeeze the granules and they touch better, so resistance falls. Release them and resistance rises.
3. A battery pushes current through the granules, so sound modulates a current far stronger than the sound itself.
4. That makes the carbon microphone an amplifier as well as a microphone, which is why telephones worked over kilometres of wire decades before the electronic amplifier existed. It is the single reason early telephony was commercially possible.
5. The exchange supplies power down your own line, which is why a landline kept working when the house lost electricity.
6. Automatic switching started with an undertaker's grudge. Almon Brown Strowger believed an operator was diverting calls to a rival and patented an automatic exchange in 1891.
7. His **step-by-step** switch used the dial pulses directly. The first digit's pulses ratcheted a contact arm up one of ten rows, and the second digit's pulses rotated it to one of ten contacts in that row. Ten by ten is a hundred lines, and switches were stacked in stages for longer numbers.
8. That is why old dials sent pulses, and why "dialling" is still the word.
9. Signalling, meaning the instructions about a call rather than the call itself, originally travelled in the same audio path as the voice, as tones.
10. That was elegant and disastrous: anyone able to play the right tone into a handset could instruct the network. Out-of-band signalling replaced it, so call control now travels on its own separate data network.

```
a 1960s long-distance call, end to end

 handset -> local loop (copper pair, -48 V from exchange)
         -> local exchange (switch)
         -> channel bank: filter, sample, compand -> DS0
         -> multiplexed into a T1/E1 with other calls
         -> trunk exchanges, path reserved end to end
         -> demultiplex -> DAC -> local loop -> handset

 the path is held for the duration of the call,
 whether anyone is speaking or not.
```

9.9.5 TECHNICAL – the engineer's version

1. Alexander Graham Bell was granted US patent 174,465 on 7 March 1876, for "Improvement in Telegraphy", covering transmission of vocal sounds by electrical undulations. The first intelligible sentence followed on 10 March

1876. Elisha Gray filed a caveat on 14 February 1876, and Antonio Meucci had filed caveat 3335 in 1871, which lapsed in 1874. The priority dispute has never fully died down.

2. Edison filed for a carbon transmitter on 27 April 1877; US patent 222,390 was granted in 1879.
3. Strowger's automatic exchange patent dates from 1891, and the first installation opened in La Porte, Indiana, on 3 November 1892 with about 75 subscribers. Crossbar switches displaced step-by-step from the late 1930s, and stored-program electronic switching from the mid-1960s.
4. Local loop electrical figures, which are conventions of the North American and most European networks rather than one global standard: -48 V DC central office battery, 20 to 50 mA loop current off-hook, ringing at about 90 V RMS at 20 Hz in North America and 25 Hz in much of Europe. Loops are typically 24 or 26 AWG copper, up to roughly 5 km without loading coils.

5. G.711, released by the CCITT in 1972, defines the 8 kHz, 8-bit, 64 kbit/s voice channel. A-law compands a 13-bit signed linear sample to 8 bits and is used in most of the world; mu-law compands 14 bits to 8 and is used in North America and Japan. Algorithmic delay is 0.125 ms with no look-ahead.
6. Multiplexing: the North American DS1 (T1) carries 24 DS0s plus one framing bit in a 193-bit frame every 125 microseconds, giving 1.544 Mbit/s. The CEPT E1 carries 32 timeslots of 8 bits in a 256-bit frame every 125 microseconds, giving 2.048 Mbit/s, with timeslot 0 for framing and timeslot 16 usually for signalling, leaving 30 voice channels.
7. SS7, Signalling System No. 7, grew out of Bell System work in the 1970s and was standardized by the ITU-T as the Q.700-series recommendations in 1980. It moved call control onto a separate packet network. Links run at 56 or 64 kbit/s, with high-speed links at 1.5 or 2.0 Mbit/s. ISUP sets up and tears down voice circuits, and SIGTRAN carries SS7 over IP using SCTP.
8. SS7 was designed for a closed club of national carriers and has essentially no authentication between operators. Since 2014, published research and reported incidents have demonstrated subscriber location tracking and SMS interception, which is the concrete reason SMS is now considered a weak second factor for authentication.

9.9.6 WORDS - remember these

1. Local loop — the pair of wires to your house — the subscriber line from premises to the serving exchange.
2. Circuit switching — a reserved path for the whole call — end-to-end resource allocation held for the call duration.
3. Companding — squeezing loud and stretching quiet — logarithmic quantization per ITU-T G.711 A-law or mu-law.
4. DS0 — one voice channel — 64 kbit/s, 8 kHz at 8 bits.
5. TDM — sharing one cable by taking turns — time-division multiplexing into fixed 125 microsecond frames.
6. SS7 — the network's own instruction channel — out-of-band common channel signalling, ITU-T Q.700 series.
7. Voice band — the slice of hearing a phone carries — 300 to 3,400 Hz.

9.10 Phone calls, part 2 - mobile and internet

9.10.1 PLAIN - in simple words

1. A mobile phone cannot send a wave down a wire, so it sends numbers by radio.
2. Radio time is scarce and expensive, so those numbers must be squeezed hard. A digital landline used 64,000 bits a second per call, and early mobile used about 13,000.
3. To get that small, mobile phones do not send the sound at all.
4. They send a description of how your throat and mouth were shaped, updated fifty times a second, and the far end builds a fresh voice from that recipe.
5. That is why mobile voice sounds correct but slightly synthetic, and why music down a phone sounds ruined. The recipe assumes a human voice.
6. Internet calls, such as WhatsApp or a work meeting, work differently again.
7. They chop the sound into small parcels of about twenty milliseconds, address each one, and throw it onto the ordinary internet.
8. Nobody reserves anything. The parcels take their chances with everyone else's video, downloads and games.
9. Most arrive. Some arrive late. Some never arrive at all.

9.10.2 PLAIN - a picture in your head

1. The old phone call was a private railway track, held for you alone.
2. An internet call is posting numbered postcards, one every twenty milliseconds, and hoping.
3. Postcards may arrive out of order, bunched up, or not at all.
4. So the receiver keeps a small tray. Postcards land in it, get sorted, and are read out at a perfectly steady rhythm from the tray rather than as they arrive. That tray is the **jitter buffer**.
5. If a postcard is missing when its turn comes, the receiver does not wait. It invents a plausible twenty milliseconds from what came before.
6. Asking for a re-send would be pointless. By the time the replacement arrived, that moment of the conversation would be long past.

Where this comparison breaks: postcards are independent, but audio frames are not. Modern codecs predict each frame from the last, so a lost frame degrades the next few as well. Opus reduces this by optionally embedding a low-quality copy of the previous frame inside each new one, a form of forward error correction.

9.10.3 PLAIN - a worked example

1. Cost out a plain internet call using G.711, the old 64 kbps telephone codec, over IPv4.
2. We send one packet every 20 milliseconds, so 50 packets a second.
3. 20 ms of G.711 is 8,000 samples a second times 0.02 s = 160 samples of one byte each, so 160 bytes of audio.
4. Now the addressing: RTP header 12 bytes, UDP header 8 bytes, IPv4 header 20 bytes. That is 40 bytes of overhead.
5. Total 200 bytes per packet, so 200 x 8 x 50 = 80,000 bits per second. A 64 kbps codec costs 80 kbps on the wire, and a quarter of the traffic is envelopes.
6. Repeat with Opus at 24 kbps and 20 ms frames: 60 bytes of audio plus the same 40 bytes of headers = 100 bytes, so 40 kbps on the wire.
7. The fixed 40 bytes dominates at low bitrates, which is why voice codecs settled on 20 ms frames. Shorter frames would multiply the header cost.

Codec	Payload bitrate	On the wire, IPv4
G.711 (64 kbps)	64.0 kbps	80.0 kbps
G.729 (8 kbps)	8.0 kbps	24.0 kbps
AMR-NB (12.2 kbps)	12.2 kbps	28.2 kbps
Opus (24 kbps)	24.0 kbps	40.0 kbps

9.10.4 PLAIN - what is really happening inside

1. A 2G GSM phone runs the GSM Full Rate codec: 13 kbit/s, working on frames of 160 samples at 8 kHz, which is 20 milliseconds.
2. It is a vocoder. It fits a mathematical model of the vocal tract to your speech, then transmits the model's settings and a description of the buzz or hiss driving it, rather than the waveform itself.
3. That is why cellular voice sounds thin: band-limited to about 3.4 kHz, and then re-synthesized rather than reproduced.
4. Setting up an internet call needs two separate things, and confusing them is the classic beginner error.
5. **Signalling** is the conversation about the call: who you want, are they there, which codecs can we both do, are they ringing, have they answered. SIP does this.
6. **Media** is the actual sound. RTP does this, over UDP, which never retransmits anything.
7. Why UDP and not TCP? Because TCP guarantees delivery by retransmitting, and waiting for a retransmission is worse than the loss it fixes.

8. So voice deliberately chooses a protocol that gives up on lost data, then repairs the hole locally.
9. **Packet loss concealment** extends the last pitch period and fades it over a few tens of milliseconds. One or two lost packets are genuinely hard to hear.
10. **Echo** is the other big problem. Your speaker plays my voice, your microphone picks it up, and I hear myself two hundred milliseconds later, which is far more disruptive than noise.
11. **Acoustic echo cancellation** works because the device already knows exactly what it sent to its own speaker. It learns a filter describing the room path from speaker to microphone, predicts the echo, and subtracts it.
12. That is why speakerphone in a bathroom is hard. The room path is long, changing and full of reflections, so the filter cannot keep up.

9.10.5 TECHNICAL - the engineer's version

1. AMR-NB frames are 160 samples at 8 kHz, 20 ms long, and the network can switch mode frame by frame to trade voice bits against error protection as radio conditions change.
2. AMR-WB samples at 16 kHz with a 50 to 7,000 Hz passband and was developed by Nokia and VoiceAge. It is what carriers market as HD Voice. EVS, Enhanced Voice Services, extends this to superwideband and fullband.
3. SIP is defined in RFC 3261 (2002), replacing RFC 2543 (1999). It is a text request/response protocol resembling HTTP, on port 5060 plain and 5061 over TLS. Core methods are INVITE, ACK, BYE, CANCEL, REGISTER and OPTIONS. Media parameters are negotiated by carrying SDP bodies in an offer/answer exchange.
4. RTP is defined in RFC 3550 (2003), replacing RFC 1889 (1996). Its 12-byte header carries a 7-bit payload type, a 16-bit sequence number for loss and reorder detection, a 32-bit media timestamp, and a 32-bit synchronization source identifier. RTCP reports loss, jitter and round-trip time to the sender.
5. Jitter buffers are adaptive: they measure arrival variation and resize themselves, typically holding 20 to 100 ms, trading added delay against the probability that a packet arrives too late to use.
6. ITU-T Recommendation G.114 gives the delay budget: under 150 ms one-way mouth-to-ear is acceptable for essentially all applications, 150 to 400 ms is usable but degraded, and over 400 ms is unacceptable for conversation. Line echo cancellers are specified in ITU-T G.168.
7. VoLTE carries voice as IP over an LTE bearer, controlled by IMS, the IP Multimedia Subsystem, rather than by a circuit-switched core. The first commercial launch was by MetroPCS in Dallas in August 2012. Its media plane uses AMR-NB as a minimum, AMR-WB for HD Voice, and EVS in newer deployments.
8. The key architectural difference from an over-the-top application: VoLTE media rides a dedicated bearer with a guaranteed bit rate and a strict quality-of-service class. A WhatsApp call is best-effort traffic on the same data bearer as everything else, is end-to-end encrypted by the application, uses Opus, and crosses the operator's network as ordinary internet data. Both are voice over IP. Only one has the network holding capacity open.
9. Voice tolerates loss and hates delay because it is an isochronous stream consumed in real time: a 20 ms hole is concealed, but a 400 ms delay breaks turn-taking, which humans manage on a scale of about 200 ms. File transfer is the exact opposite. It is verified in full at the end, so one wrong byte ruins it, while an extra thirty seconds costs nothing. That is why one runs on UDP with concealment and the other on TCP with retransmission.

Codec	Bitrate	Standardized
GSM Full Rate (RPE-LTP)	13 kbit/s	Early 1990s
GSM Half Rate	5.6 kbit/s	1990s
GSM Enhanced Full Rate	12.2 kbit/s	1990s
AMR-NB, 8 modes	4.75 to 12.2 kbit/s	3GPP, Oct 1999
AMR-WB / G.722.2	6.6 to 23.85 kbit/s	ITU-T, 2002

```
a SIP call setup, simplified

  caller                              callee
    |------ INVITE (with SDP offer) --->|
    |<----- 100 Trying -----------------|
    |<----- 180 Ringing ----------------|
    |<----- 200 OK (with SDP answer) ---|
    |------ ACK ----------------------->|
    |==== RTP audio over UDP, both ways =|
    |------ BYE ----------------------->|
    |<----- 200 OK ---------------------|
```

9.10.6 WORDS - remember these

1. Vocoder — a codec that rebuilds your voice rather than copying it — a parametric coder transmitting vocal tract model coefficients.
2. RTP — the protocol carrying the sound — Real-time Transport Protocol, RFC 3550, over UDP.
3. SIP — the protocol that rings the phone — Session Initiation Protocol, RFC 3261, with SDP offer/answer.
4. Jitter buffer — the tray that smooths arrival times — an adaptive de-jitter queue trading latency for late-packet loss.
5. PLC — filling in a lost moment — packet loss concealment by pitch-period extrapolation.
6. AEC — stopping the caller hearing themselves — acoustic echo cancellation by adaptive filtering of the known playback signal.
7. VoLTE — carrier calls over the data network — IMS-controlled voice on a dedicated LTE bearer with guaranteed bit rate.
8. Best effort — no promises about delivery — the default internet service model with no reserved capacity.

9.11 Noise cancelling, spatial audio and the modern tricks

9.11.1 PLAIN - in simple words

1. There are two ways to make the world quieter in headphones.
2. **Passive** means physically blocking sound: a tight seal, heavy cups, foam. It is just a wall.
3. **Active** means fighting sound with sound.
4. A microphone on the outside listens to the noise arriving, and a chip works out the exact opposite wave: where the noise pushes, the opposite pulls.
5. The speaker plays that opposite. Push and pull arrive together and cancel.
6. This works well for low rumbling noise such as an aircraft or a bus engine, and poorly for sharp high noise such as speech or a fork on a plate.
7. Separately, phones use several microphones at once to work out which direction a sound came from and keep only the direction you speak from. That is **beamforming**.
8. And headphones can fool you into hearing a sound as if it came from your left, above, or behind, by copying the way your own head would have changed it.

9.11.2 PLAIN - a picture in your head

1. Imagine two people pushing a swing.
2. If they push together at the right moments, the swing goes higher. That is adding two waves in phase.
3. If the second pushes exactly when the first pulls, with equal strength, the swing does not move at all. That is cancellation.
4. Noise cancelling is the second person, and the chip's whole job is getting the timing exactly right.
5. To push at exactly the wrong moment, you must know when that moment is before it happens. For a slow swing that is easy. For one wobbling twenty thousand times a second, the chip has microseconds to hear, calculate and play.

Where this comparison breaks: cancellation is only exact at one point in space. Move your head a few centimetres and the arrival times change, so what cancelled now partly adds. At low frequencies the wave is metres long, so a few centimetres hardly matter, which is the real reason active cancellation is a low-frequency technique.

9.11.3 PLAIN - a worked example

1. Take a 100 Hz engine rumble. Its wavelength is 3.43 metres.
2. To cancel it you must be right within a small fraction of that, say 10 cm, which is 3 percent of a wavelength, about 10 degrees of phase error.
3. 10 cm of air is 0.29 milliseconds, comfortable for a chip running millions of operations a second.
4. Now take a 5,000 Hz hiss. Its wavelength is 6.9 cm.
5. The same 10 cm of positional error is now more than a whole wavelength. Your cancellation arrives at some arbitrary phase and may make it louder.
6. That single calculation is why every noise cancelling headphone works well below about 1 kHz and relies on the ear cup seal above it.
7. Beamforming numbers: two phone microphones 14 cm apart. Sound from directly ahead reaches both at the same instant, while sound from the side arrives 0.14 / 343 = 408 microseconds later at the far microphone. Delay one microphone by 408 microseconds and add, and side sounds partly cancel while front sounds double.

9.11.4 PLAIN - what is really happening inside

1. Active noise cancellation comes in three arrangements.
2. **Feedforward**: a microphone outside the cup hears the noise before it reaches your ear, so there is time to compute. It cannot tell what actually arrived at the eardrum.
3. **Feedback**: a microphone inside the cup hears the result, including whatever the headphone did, and corrects the error. It has almost no time, so it works only at low frequencies.
4. **Hybrid**: both. Nearly all current premium headphones do this.
5. Now spatial audio. Your brain locates sounds with three main clues.
6. **Interaural time difference**: sound from your left reaches the left ear first, by up to about 660 microseconds.
7. **Interaural level difference**: your head shadows the far ear, mostly at high frequencies where the head is large compared with the wavelength.
8. **Spectral shaping**: the folds of your outer ear boost and notch particular frequencies depending on whether a sound comes from above, in front or behind. That is what resolves up from down, which timing alone cannot.
9. All three together, measured as a filter for every direction, are the **head-related transfer function**, or HRTF.
10. Headphone spatial audio applies that filter to a sound, so your brain receives the clues it would have received from a real source.
11. Head tracking makes it far more convincing, because when you turn your head the virtual source stays put, exactly as a real one would.

9.11.5 TECHNICAL - the engineer's version

1. Paul Lueg was granted US patent 2,043,416 in 1936 for cancelling sinusoidal tones in a duct by phase advance. Lawrence Fogel patented cockpit noise cancellation in the 1950s. Bose shipped an active noise cancelling aviation headset in 1989 and the consumer QuietComfort headphone in 2000.
2. Practical performance for current premium over-ear models: roughly 20 to 30 dB of active attenuation below 300 Hz, tapering to near zero by 1.5 to 2 kHz, on top of 10 to 25 dB of passive attenuation above 1 kHz. Manufacturers quote the sum and rarely publish the curve.
3. Feedforward controllers are typically fixed or slowly adaptive FIR filters. Feedback loops are analogue or very low-latency digital, and are limited by the Bode sensitivity integral, which guarantees that at-

tenuation at one frequency must be paid for with amplification at another. That is why poorly designed cancellation adds a faint hiss or a pressure sensation around 2 to 4 kHz.

4. Beamforming: delay-and-sum is the simplest form. Adaptive methods, notably minimum variance distortionless response, steer nulls at interfering sources instead. A two-microphone phone array gives modest directivity, while laptop and smart-speaker arrays of four to eight microphones do far better.
5. Channel layouts, as a standard: 2.0 stereo; 5.1, meaning left, centre, right, surround left, surround right and a low-frequency effects channel limited to about 120 Hz; 7.1; and object-based systems such as Dolby Atmos, introduced in 2012, which transmit audio objects with position metadata and render them to whatever speakers exist.
6. HRTF datasets used in research include the CIPIC database and the MIT KEMAR measurements. Rendering convolves source audio with the left and right HRTF impulse responses for the wanted direction, interpolating between measured points.
7. Active research, not settled fact: individualized HRTFs. Generic HRTFs give weaker elevation cues and more front-back confusion for listeners whose ear geometry differs from the measurement dummy. Approaches under investigation include photogrammetry from phone cameras, anthropometric regression and machine-learned personalization. Vendor claims of personalized spatial audio from a short ear scan are an engineering approximation, not a solved problem.

9.11.6 WORDS - remember these

1. Anti-phase — the exact opposite wave — a signal inverted by 180 degrees at the frequency of interest.
2. Feedforward ANC — the outside microphone — control derived from the disturbance before it reaches the ear.
3. Feedback ANC — the inside microphone — closed-loop correction of the residual error at the ear.
4. Beamforming — listening in one direction — spatial filtering across a microphone array by delay-and-sum or adaptive weights.
5. ITD and ILD — which ear hears it first and loudest — interaural time and level differences, the primary horizontal localization cues.
6. HRTF — how your own ears colour a sound by direction — head-related transfer function, the direction-dependent filter of head, torso and pinna.
7. Object audio — sounds with positions rather than channels — metadata-driven rendering to an arbitrary speaker layout.

9.98 Common wrong ideas

1. Wrong: higher bitrate always sounds better. Right: bitrate matters only until the codec has enough bits to hide its errors under the masking threshold. Beyond that, extra bits change nothing audible, and a good encoder at 96 kbps AAC can beat a poor encoder at 256 kbps MP3.
2. Wrong: more watts means louder. Right: loudness follows speaker sensitivity far more than amplifier power. Doubling power adds only 3 dB, while a speaker 6 dB more sensitive is as loud on a quarter of the power.
3. Wrong: lossless always sounds audibly better than lossy. Right: lossless is guaranteed bit-identical, which matters for archiving, editing and re-encoding. In blind listening at modern high bitrates, most listeners cannot reliably tell them apart. Keep lossless because it is future-proof, not because you can hear it.
4. Wrong: the microphone records the sound. Right: the microphone only converts pressure into a voltage. Nothing is recorded until something samples that voltage and stores numbers. In a condenser or MEMS microphone the sound does not even supply the signal energy; it modulates a bias already present.
5. Wrong: digital audio is a staircase, so it is less smooth than analogue. Right: the staircase is an intermediate artefact inside the converter. The reconstruction filter recovers the unique smooth band-limited wave passing through the samples, so there is no jaggedness in the output.

6. Wrong: MP3 works by cutting off high frequencies. Right: that is one bit-allocation decision some encoders make. The mechanism is shaping quantization noise to hide beneath louder sounds across the whole spectrum.
7. Wrong: noise cancelling headphones cancel all noise. Right: they work well below roughly 1 kHz, because cancellation must be accurate to a fraction of a wavelength. Higher frequencies are blocked by the physical seal instead.
8. Wrong: a WhatsApp call and a mobile phone call are the same thing on the same network. Right: both are voice over IP, but a carrier VoLTE call rides a dedicated bearer with guaranteed bit rate under IMS control, while an application call is best-effort internet traffic with no reserved capacity.
9. Wrong: a bigger speaker is always better. Right: a large cone can move enough air for bass but is too heavy to reproduce treble accurately, which is why real systems split the range between drivers sized for each job.
10. Wrong: 44.1 kHz means the system can handle only 44,100 different sounds. Right: it means 44,100 measurements per second, which by the sampling theorem perfectly captures every frequency below 22,050 Hz.

9.99 Chapter summary in 20 lines

1. Sound is a travelling pattern of air pressure: compressions and rarefactions.
2. Frequency in hertz sets pitch, amplitude sets loudness, and wavelength is the speed of sound, about 343 m/s at 20 C, divided by frequency.
3. Decibels are logarithmic because hearing responds to ratios: +6 dB doubles pressure and +10 dB multiplies power by ten.
4. A speaker is a coil in a magnet's gap glued to a cone. Current makes force by F = Bl*I, the cone moves, the air moves, and you hear it.
5. Faraday's law runs the same machine backwards, which is why a moving cone generates voltage and why a dynamic microphone needs no power.
6. Big cones move enough air for bass but are too heavy for treble, so systems split the band with crossovers, and enclosures stop the back wave cancelling the front.
7. A phone speaker cannot make bass because it cannot displace enough air, by about three orders of magnitude. That is physics, not software.
8. Condenser, electret and MEMS microphones modulate a bias voltage rather than generating the signal energy from the sound itself.
9. Sampling captures a wave perfectly if the rate exceeds twice the highest frequency present, which is the Nyquist-Shannon theorem.
10. Anti-aliasing filtering before the converter is mandatory, because folded frequencies can never be separated afterwards.
11. Bit depth sets dynamic range at about 6 dB per bit, so 16 bits gives roughly 96 to 98 dB, and dither turns ugly quantization distortion into benign hiss.
12. Almost every audio converter today is oversampling sigma-delta, in both directions.
13. CD audio is 44,100 samples a second, 16 bits, two channels, which is 1.4112 Mbit/s and about 31.75 MB for a three-minute song.
14. Lossless codecs such as FLAC predict and store the error, reaching 50 to 70 percent, while lossy codecs such as MP3, AAC and Opus discard what masking makes inaudible and reach a tenth or less.
15. A codec is a coding method and a container is the file that holds it, which is why MP4 can hold either AAC or ALAC and why extensions mislead.
16. Records store the waveform as a physical groove, tape stores it as aligned magnetic particles, and the CD stores numbers as pit-to-land transitions read by a 780 nm laser.
17. Computer audio is a race against a DMA deadline: buffer size sets latency, and missing the deadline puts a step in the waveform heard as a click.

18. The old telephone reserved a whole path per call and digitized voice as 8 kHz, 8-bit companded samples giving 64 kbit/s, multiplexed into 1.544 Mbit/s T1 or 2.048 Mbit/s E1 and controlled by SS7.
19. Mobile voice sends vocal tract parameters instead of a waveform at around 13 kbit/s, which is why it sounds thin, while internet calls send RTP over UDP with jitter buffers, loss concealment and echo cancellation.
20. Voice tolerates loss but hates delay, because a 20 ms hole can be concealed and a 400 ms delay destroys turn-taking, while file transfer is the opposite, which is why one uses UDP and the other TCP.

PART C

The Machine

The processor, the memory ladder, storage, the buses and drivers that join it all up, and how we got here.

10

CHAPTER

KB

Inside the CPU

10.0 What this chapter gives you

1. You will be able to name every part inside a processor and say what it does.
2. You will be able to walk one instruction from memory to result, step by step, and say which wire carries what at each step.
3. You will be able to read a line of x86-64 assembly and the same line in ARM64 assembly, and say what the machine does with it.
4. You will be able to explain what an instruction set architecture is, and why it is a contract rather than a design.
5. You will be able to explain pipelining with real timing tables, and say exactly why a very deep pipeline made the Pentium 4 slower, not faster.
6. You will be able to explain out-of-order execution, register renaming and branch prediction without any magic, with real accuracy figures.
7. You will be able to explain why processors stopped getting faster in megahertz around 2005 and started getting wider and more numerous instead.
8. You will be able to walk the MESI cache coherence protocol between two cores by hand, and calculate Amdahl's law for a real program.
9. You will be able to read a desktop processor spec sheet and a phone system-on-chip spec sheet and decode every single number on the page.
10. You will be able to say why gigahertz alone tells you almost nothing, and what number to look at instead.

10.1 What a CPU is: a box that repeats one loop forever

10.1.1 PLAIN - in simple words

1. A CPU is a small square of silicon that does one thing over and over.
2. The one thing is: get the next order, work out what it means, do it.
3. Then it does it again. And again. Billions of times each second.
4. Nothing else. There is no cleverness hidden anywhere. It is a loop.
5. The orders are called **instructions**. Each is just a number in memory.
6. An instruction is tiny. "Add these two numbers." "Copy this to there." "If that number is zero, jump somewhere else."
7. A program is a long list of these tiny orders, one after another.
8. Everything you have ever seen a computer do is millions of these orders running fast enough that the steps blur into one smooth thing.
9. Earlier chapters built the pieces: switches, gates, an adder, an arithmetic unit, and cells that remember a bit. This chapter wires them into a CPU.

10.1.2 PLAIN – a picture in your head

1. Picture a very fast, very obedient cook alone in a kitchen.
2. On the wall is a long shelf of recipe cards, numbered 1, 2, 3, and so on.
3. The cook has a small clip that holds one card at a time. That clip is the **instruction register**.
4. The cook has a counter on a string round the neck showing which card is next. That is the **program counter**.
5. The cook has a few small bowls on the bench, always within reach. Those are the **registers**.
6. There is a big cold store at the far end of the building. That is main memory. Walking there takes a long time.
7. Near the bench there is a small fridge holding whatever was fetched recently, so the cook does not have to walk. That is the **cache**.
8. The cook does exactly what the card says, moves the counter to the next number, and picks up the next card. Forever.

Where this comparison breaks: the cook understands the recipe. The CPU does not understand anything. The bit pattern of the instruction physically opens and closes switches, and the result falls out. There is no reading and no deciding, only voltage arriving at gates.

10.1.3 PLAIN – a worked example

1. Say memory holds the number 7 at address 64 and the number 5 at address 65.
2. We want their sum written to address 66.
3. The CPU does this in four orders, and each order is one trip round the loop.

Step	The order	What moves
1	Load address 64 into bowl 0	7 arrives in bowl 0
2	Load address 65 into bowl 1	5 arrives in bowl 1
3	Add bowl 1 into bowl 0	bowl 0 becomes 12
4	Store bowl 0 to address 66	12 arrives in memory

4. Notice that steps 1, 2 and 4 move data around and do no maths at all.
5. Only step 3 does arithmetic. That is normal. Most instructions in a real program are moving things, not calculating things.
6. Notice also that the CPU never added "7 plus 5". It added whatever happened to be in bowl 0 and bowl 1. It has no idea what those numbers mean.

10.1.4 PLAIN – what is really happening inside

1. Split the CPU into two halves: the part that carries numbers, and the part that decides what happens to them.
2. The carrying part is the **datapath**: registers, the arithmetic unit, and the wires between them.
3. The deciding part is the **control unit**. It reads the bit pattern of the instruction and switches on exactly the right set of wires.
4. Think of the control unit as a room full of light switches. An instruction pattern is a hand that flips a particular set of switches on.
5. Those control wires say things like: "let register 1 out onto bus A", "tell the arithmetic unit to add, not subtract", "write the answer back into register 0", "increase the counter by one".
6. Everything else is plumbing. Buses are shared bundles of wires that carry an address, or data, or control signals.
7. A **memory controller** sits at the edge of the chip and talks the exact electrical language that memory chips expect.

8. All of it moves in step with a clock, a square wave that ticks billions of times a second and says "now everyone move".

10.1.5 TECHNICAL - the engineer's version

1. The functional blocks of any general-purpose CPU core are as follows.
2. **Control unit**: decodes the instruction and drives the control signals. In a modern x86 core it is itself a small program in read-only memory, plus hardwired fast paths. See section 10.6 on microcode.
3. **ALU** (**arithmetic logic unit**): add, subtract, AND, OR, XOR, shift, compare. Built from the adders and multiplexers of Chapter 5.
4. **Register file**: a small multi-ported SRAM array. Multi-ported means several reads and writes can happen in the same clock cycle.
5. **Program counter** (**PC**), called RIP on x86-64 and PC on ARM64: holds the address of the next instruction.
6. **Instruction register** (**IR**): holds the instruction word currently being decoded. On out-of-order cores this is not one register but a queue.
7. **Caches**: L1 instruction, L1 data, L2 per core, L3 shared. Section 10.11.
8. **Buses and interconnect**: on-die, these are now networks, not buses. AMD uses Infinity Fabric, Arm uses CMN mesh interconnects.
9. **Memory controller**: generates DDR5 command sequences, refresh, training. On AMD desktop parts it lives on a separate I/O die, not on the core die.

Chip	Year	Transistors	Clock
Intel 4004	1971	2,300	740 kHz
Intel 8086	1978	29,000	5 MHz
Intel 80386	1985	275,000	12 MHz
Intel Pentium	1993	3.1 million	60 MHz
Core 2 Duo	2006	291 million	2.93 GHz

10. The Intel 4004 shipped on 15 November 1971. Federico Faggin led the design with Ted Hoff, Stan Mazor and Masatoshi Shima of Busicom.
11. An AMD Zen 4 core complex die holds 6.57 billion transistors in 70 mm2 on TSMC N5. The Zen 5 equivalent is roughly 8.3 billion in about 70.6 mm2 on TSMC N4P. Treat the Zen 5 figure as approximate.
12. To observe the blocks on Linux: `lscpu`, `lscpu -C` for caches, `cat /proc/cpuinfo`. On macOS: `sysctl -a | grep machdep.cpu`.

10.1.6 WORDS - remember these

1. Instruction — one tiny order — an encoded operation in the ISA.
2. Datapath — the pipes numbers flow through — registers, ALU, buses.
3. Control unit — the switchboard — logic that drives control signals from the decoded opcode.
4. Program counter — a bookmark — register holding the next instruction address, RIP on x86-64, PC on ARM64.
5. Instruction register — the clip holding the current card — latch holding the instruction word being decoded.
6. Register file — the small bowls on the bench — multi-ported SRAM array of architectural or physical registers.
7. Memory controller — the translator at the door — on-die block generating DDR command and timing sequences.

10.2 The stored-program idea

10.2.1 PLAIN - in simple words

1. Here is the single biggest idea in computing, and it is very simple.
2. Instructions are numbers. Data is numbers. Memory holds numbers.
3. So instructions and data can live in the same memory, side by side.
4. Nothing in memory has a label saying "this one is an order". It is only an order because the program counter pointed at it.
5. That is why one machine can run any program. You do not rewire it. You just put different numbers in memory.
6. Before this idea, changing a computer's job meant physically replugging it.
7. It also means a program can write numbers that are then run as orders. That is enormously useful and enormously dangerous, and both are covered below.

10.2.2 PLAIN - a picture in your head

1. Picture a huge wall of numbered pigeonholes, all identical.
2. Some hold shopping lists. Some hold instructions for the shop assistant.
3. Nothing on the outside of a pigeonhole tells you which is which.
4. The assistant simply starts at pigeonhole 100 and treats whatever is there as an instruction, because that is where they were told to start.
5. If someone slips a shopping list into pigeonhole 100, the assistant will try to obey the shopping list as if it were an order.
6. That mistake is the root of a whole class of security holes.

Where this comparison breaks: real hardware now does add labels. Modern memory pages carry permission bits saying "may be executed" or "may be written", and in most cases not both. The pigeonholes are still identical, but the shelf now has rules about which ones the assistant may read orders from.

10.2.3 PLAIN - a worked example

1. Take the four bytes `48 83 C0 01` sitting in memory.
2. Read as data, that is just four numbers: 72, 131, 192, 1.
3. Read as an x86-64 instruction, it is `add rax, 1`, meaning "add one to register RAX".
4. Both readings are correct. The bytes do not choose. The CPU chooses, by where the program counter points.
5. Now take the two bytes `EB FE`. As an x86-64 instruction that is `jmp -2`, a jump back to itself: an infinite loop in two bytes.
6. If a program accidentally jumps into a block of text, the CPU will happily try to execute the letters. `H` is 0x48, which starts many valid x86 instructions. That is why crashes from bad jumps look so random.

10.2.4 PLAIN - what is really happening inside

1. There are two ways to arrange memory, and they have names.
2. **Von Neumann**: one memory, holding both instructions and data, reached through one set of wires.
3. **Harvard**: two separate memories with separate wires, one for instructions and one for data.
4. Von Neumann is flexible. You can load a program as if it were data, then run it. That is exactly what an operating system does when it starts an app.
5. Von Neumann has a bottleneck. One set of wires must carry both the next instruction and the data it needs, so they take turns.
6. Harvard has no such traffic jam, because the two roads are separate.
7. Real CPUs use a mixture called **modified Harvard**. Deep down there is one memory, von Neumann style. But right next to the core there are two separate small caches: one for instructions, one for data.

8. So the core sees Harvard, with two roads, and the system sees von Neumann, with one memory. You get the speed of one and the flexibility of the other.

10.2.5 TECHNICAL - the engineer's version

1. John von Neumann's "First Draft of a Report on the EDVAC" circulated on 30 June 1945 and described a single addressable store for code and data. The design work was joint with J. Presper Eckert and John Mauchly, whose names were left off the draft, a dispute that is still argued about.
2. The Harvard Mark I, completed in 1944 under Howard Aiken at Harvard, read instructions from punched paper tape and data from separate counters. That physical separation is where the name comes from.
3. Modified Harvard in practice: separate L1 instruction and L1 data caches fed from a unified L2. On AMD Zen 5 that is 32 KB L1i and 48 KB L1d per core, backed by a unified 1 MB L2.
4. This split means self-modifying code needs explicit help. Writing bytes goes into L1d. The fetcher reads L1i. The two can disagree.
    1. On x86-64 the hardware keeps them coherent for you, but a serializing instruction is still required before the new bytes are guaranteed run.
    2. On ARM64 you must do it by hand: `DC CVAU` to clean the data cache to the point of unification, `IC IVAU` to invalidate instruction cache, then `DSB` and `ISB` barriers. Skipping this is a classic JIT bug.
5. The danger side. A stack buffer overflow writes past the end of an array and overwrites the saved return address, so the CPU returns into attacker bytes. The Morris worm of 2 November 1988 used exactly this against the Unix `fingerd` service.
6. The defence is the no-execute page permission bit. AMD shipped NX with Athlon 64 in 2003, Intel called it XD, and Microsoft enabled it as Data Execution Prevention in Windows XP Service Pack 2 in August 2004.
7. Attackers answered with return-oriented programming: chain together fragments of code that already exists and is already marked executable. No new code is written, so NX does not stop it.
8. The useful side. A just-in-time compiler writes machine code into a data buffer at run time, flips the page permission from writable to executable, and jumps into it. The JavaScript engines in every browser do this thousands of times while a page loads.

Term	Code and data memory	Seen in
Von Neumann	One shared	Program model of a PC
Harvard	Two separate	Small microcontrollers
Modified Harvard	Split caches, one store	Every mainstream CPU

9. Tools: `readelf -l` shows segment permissions, `R E` for read-execute and `RW` for read-write. On Linux `cat /proc/self/maps` shows the same live.

10.2.6 WORDS - remember these

1. Stored program — orders live in memory like any other number — code and data share one address space.
2. Von Neumann architecture — one memory for everything — unified instruction and data store, with a shared bus bottleneck.
3. Harvard architecture — two separate memories — separate instruction and data address spaces and buses.
4. Modified Harvard — one memory but two front doors — split L1i and L1d over a unified lower hierarchy.
5. Buffer overflow — writing past the end of a box — overwriting adjacent stack or heap state, classically the saved return address.

6. NX bit — a page marked "not runnable" — no-execute permission bit in the page table entry, XD on Intel.
7. JIT — writing new code while running — just-in-time compilation into a writable page later marked executable.

10.3 The fetch-decode-execute cycle

10.3.1 PLAIN - in simple words

1. The loop the CPU repeats forever has three parts and standard names.
2. **Fetch**: read the instruction at the address in the program counter.
3. **Decode**: work out what it is and where its inputs are.
4. **Execute**: do it, and write the answer somewhere.
5. Then increase the program counter and start again.
6. Some instructions add a fourth part, a memory access, and a fifth, writing the result back into a register. We will use the five-part version later.
7. If the instruction was a jump, the program counter is not increased by one. It is loaded with a new address. That is how loops and decisions work.
8. That is the entire behaviour of a processor. Everything after this in the chapter is a trick to run this loop faster.

10.3.2 PLAIN - a picture in your head

1. Picture a postal sorting clerk at a desk.
2. The clerk reads the number on a slip of paper: "go to shelf 100".
3. They walk to shelf 100 and bring back the envelope. That is fetch.
4. They open it and read what kind of job it is. That is decode.
5. They do the job at their desk. That is execute.
6. They cross out 100 and write 101 on the slip, then start again.
7. If the envelope said "next, go to shelf 240", they write 240 instead.

Where this comparison breaks: a real CPU does not finish one envelope before starting the next. It has several envelopes open at once, in different stages, and it often guesses which shelf comes next before it knows. Those are pipelining and branch prediction, sections 10.7 and 10.8.

10.3.3 PLAIN - a worked example

1. Let us invent a small machine so nothing is hidden. Call it KB-1.
2. It has four registers R0 to R3, 16-bit words, and 16-bit instructions.
3. An instruction splits into fields like this.

```
bit 15..12   11..10   9..8    7..0
+--------+--------+--------+----------+
| opcode |   rd   |   rs   | imm/addr |
+--------+--------+--------+----------+
```

4. Five opcodes are enough for our program.

Opcode	Name	Meaning
0001	LOAD rd,[a]	rd <- MEM[a]
0010	STORE rd,[a]	MEM[a] <- rd
0011	ADD rd,rs	rd <- rd + rs
0100	SUB rd,rs	rd <- rd - rs
0111	JZ addr	jump if last result 0

5. Memory starts with 7 at address 0x40, 5 at 0x41, and 0 at 0x42.
6. The program, with its exact machine words, is this.

```
addr  word    assembly
0x00  0x1040  LOAD  R0, [0x40]
0x01  0x1441  LOAD  R1, [0x41]
0x02  0x3100  ADD   R0, R1
0x03  0x2042  STORE R0, [0x42]
```

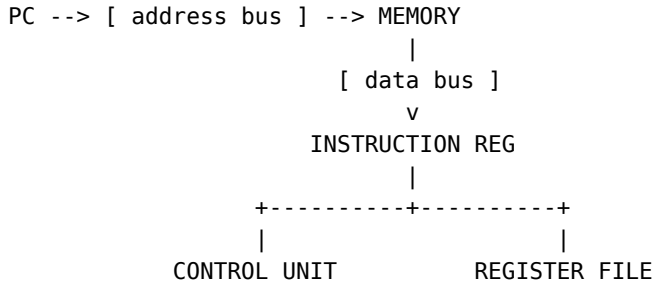
7. Check the encoding of 0x1441 by hand. In binary it is 0001 01 00 01000001. Opcode 0001 is LOAD, rd is 01 meaning R1, the low eight bits are 0x41. So: load from address 0x41 into R1.
8. Now run it and watch every value change.

After	PC	R0, R1	Mem[0x42]
start	0x00	0, 0	0
instr 1	0x01	7, 0	0
instr 2	0x02	7, 5	0
instr 3	0x03	12, 5	0
instr 4	0x04	12, 5	12

9. Three kinds of instruction did all the work: a load, an arithmetic operation, and a store. Almost all real code is these three kinds.

10.3.4 PLAIN - what is really happening inside

1. Now walk one cycle at the level of the wires. Take instruction 3, ADD.
2. Tick 1, fetch. The program counter holds 0x02. Its value is driven onto the address bus. The read line is raised.
3. Memory answers with the 16 bits 0x3100 on the data bus.
4. The write-enable line of the instruction register is raised, so on the next clock edge the instruction register latches 0x3100.
5. Tick 2, decode. The top four bits, 0011, go into the control unit. The control unit is a lookup that turns 0011 into a set of raised wires.
6. Bits 11 to 10 pick R0 as the destination and as source A. Bits 9 to 8 pick R1 as source B. Those go to the register file's address inputs.
7. The register file drives 7 onto bus A and 5 onto bus B. No clock needed: reading is combinational, the values just appear after a small delay.
8. Tick 3, execute. The control unit puts the code for "add" on the ALU's function input. The adder settles and 12 appears at its output.
9. The write-back line for R0 is raised. On the next clock edge, R0 latches 12.
10. In parallel, the program counter's own adder computes 0x02 plus 1, and the program counter latches 0x03 on the same edge.
11. Every one of these steps is switches opening and closing, as in Chapter 4. Nothing is interpreted. The bit pattern is the command.



```
                |                 |      |
          control wires          busA    busB
                |                 |      |
                +---------------> [   ALU   ]
                                    |
                                write back
```

10.3.5 TECHNICAL - the engineer's version

1. The same program in real x86-64, with RDI holding the base address.

```
8B 07        mov  eax, dword ptr [rdi]
03 47 04     add  eax, dword ptr [rdi+4]
89 47 08     mov  dword ptr [rdi+8], eax
```

2. Note the byte counts: two, three, three. x86-64 instructions are variable length, from 1 to 15 bytes, with 15 a hard architectural limit.
3. Note also that x86-64 can add straight from memory. One instruction does a load and an add. That is the classic CISC habit.
4. The same program in ARM64, with X0 holding the base address.

```
B9400001     ldr  w1, [x0]
B9400402     ldr  w2, [x0, #4]
0B020021     add  w1, w1, w2
B9000801     str  w1, [x0, #8]
```

5. Every ARM64 instruction is exactly four bytes. Always. That is the classic RISC habit, and it makes the decoder far simpler.
6. ARM64 cannot add from memory. Loads and stores are the only instructions that touch memory. This is called a load-store architecture.
7. Decode 0xB9400402 by hand. Bits 31 to 22 are 1011100101, the 32-bit LDR unsigned-offset form. Bits 21 to 10 are the scaled immediate, value 1, and the scale for a 32-bit load is 4, so the offset is 4 bytes. Bits 9 to 5 are Rn equals X0. Bits 4 to 0 are Rt equals W2.
8. On a modern core, none of this happens in three tidy ticks. The x86-64 `add eax, [rdi+4]` is split into two internal micro-operations, an address generation plus load, and an add. Section 10.6.
9. To watch real instructions: `objdump -d ./a.out` disassembles a binary, `gdb` with `layout asm` and `stepi` single-steps them, and `perf stat ./a.out` counts how many actually retired.

10.3.6 WORDS - remember these

1. Fetch — go and get the next order — read memory at the PC into the instruction register.
2. Decode — work out what it says — map opcode and operand fields onto control signals and register file ports.
3. Execute — do it — drive the ALU or address unit and produce a result.
4. Write-back — put the answer away — latch the result into the destination register on the clock edge.
5. Load-store architecture — only two instructions touch memory — arithmetic operates on registers only, as in ARM64, RISC-V and MIPS.
6. Combinational — settles by itself, no clock — logic whose output depends only on current inputs after a propagation delay.

10.4 Instruction set architecture

10.4.1 PLAIN - in simple words

1. An **instruction set architecture**, or ISA, is the list of orders a CPU understands, and exactly how each one is written as a number.

2. It also says how many registers there are, what they are called, how big a number is, and what happens when something goes wrong.
3. It is a promise, not a design. It says what the chip must appear to do. It says nothing about how the chip does it inside.
4. That promise is why a program compiled in 2003 still runs today. The inside of the chip changed completely. The promise did not.
5. Two chips can keep the same promise with wildly different insides. An Intel and an AMD desktop chip share almost nothing internally, yet run the same programs, because they honour the same ISA.
6. The three ISAs that matter today are x86-64, ARM64 and RISC-V.

10.4.2 PLAIN - a picture in your head

1. Think of an ISA as the menu in a restaurant, not the kitchen.
2. The menu says: item 14 is soup, item 22 is rice, item 30 is bread.
3. Any kitchen that produces the right dish for each number is a valid kitchen. One may use a wood fire, another an induction hob.
4. A customer who learned the menu twenty years ago can still order item 14 and get soup, even though the kitchen was rebuilt three times.
5. If a new kitchen decided item 14 now means salad, every old customer breaks. That is why ISAs almost never remove things.
6. That is also why x86-64 still carries instructions from 1978.

Where this comparison breaks: menus are short, and ISAs are not. A full x86-64 manual set from Intel runs to several thousand pages across its volumes, and nobody knows all of it. Also, some menu items are far slower to cook than others, and the menu does not tell you which.

10.4.3 PLAIN - a worked example

1. Every instruction is a number split into fields. Take RISC-V, which is the tidiest example, with a fixed 32-bit length.
2. The R-type format, used for register-to-register arithmetic, is this.

```
bits 31..25  24..20 19..15 14..12 11..7  6..0
+---------+------+------+------+------+-------+
| funct7  | rs2  | rs1  |funct3|  rd  |opcode |
+---------+------+------+------+------+-------+
```

3. For `add x5, x6, x7`: opcode 0110011, funct3 000, funct7 0000000, rs1 is x6, rs2 is x7, rd is x5. The whole word comes out as 0x007302B3.
4. Change funct7 to 0100000 and the same opcode becomes `sub`. One bit flip in a field turns add into subtract. That is what a field is for.
5. An **immediate** is a constant baked into the instruction itself, rather than read from a register. `addi x5, x6, 100` carries the 100 in 12 bits.
6. Now addressing modes: the different ways an instruction names where its data is. Here they are with worked values, using x86-64 notation.

Mode	Example	Where the data is
Immediate	mov rax, 42	In the instruction
Register	mov rax, rbx	In RBX
Absolute	mov rax, [0x600100]	At that fixed address
Reg indirect	mov rax, [rbx]	At the address in RBX
Base+disp	mov rax, [rbx+8]	At RBX plus 8
Indexed	mov rax, [rbx+rcx*4]	At RBX plus 4 times RCX

Mode	Example	Where the data is
PC-relative	lea rax, [rip+0x20]	0x20 past the next instr

7. Work one through. If RBX holds 0x1000 and RCX holds 3, then `mov rax, [rbx+rcx*4]` reads the 8 bytes at 0x1000 plus 12, which is 0x100C. That single instruction indexes an array of 4-byte elements.
8. That scale factor of 1, 2, 4 or 8 exists precisely because arrays of bytes, shorts, ints and longs are so common.

10.4.4 PLAIN - what is really happening inside

1. The decoder is a big lookup. It takes the instruction bits and produces control signals, plus register numbers, plus a constant.
2. With fixed-length instructions, the decoder knows where every field is before it has read anything. It can slice all fields in parallel.
3. With variable-length instructions it cannot. It must work out how long the first instruction is before it knows where the second begins.
4. On x86-64 the length depends on prefixes, on the opcode, on the ModRM byte, on the SIB byte and on the displacement size. Length decoding is a serial chain, and it is genuinely hard.
5. Intel and AMD solve it by brute force: try to decode at many byte offsets at once and throw away the wrong answers, plus cache the decoded results so the work is not repeated. Section 10.6.
6. That is a real, permanent cost of x86-64. It costs transistors and power that an ARM64 decoder simply does not spend.
7. It is also, in 2026, a survivable cost. It is a fixed overhead at the front of a core that is otherwise dominated by caches and execution units.

10.4.5 TECHNICAL - the engineer's version

1. The RISC idea began at IBM with John Cocke's 801 project, which ran from 1975 onward and was not published widely at the time.
2. David Patterson's group at Berkeley built RISC-I in 1981 and coined the term. John Hennessy's group at Stanford built MIPS from 1981.
3. Hennessy and Patterson shared the 2017 ACM Turing Award for this work.
4. The original argument: compilers do not use complex instructions, so spend the transistors on registers, pipelines and caches instead.
5. What is the real difference today. Honest answer: at the level of the execution core, almost none. Both x86-64 and ARM64 cores decode into internal micro-operations, rename registers, execute out of order and retire in order. The differences that survive are these.
    1. Decode complexity. Fixed 32-bit versus 1 to 15 bytes.
    2. Memory operands in arithmetic instructions, allowed on x86-64, not on ARM64.
    3. Memory ordering model, strong on x86-64, weak on ARM64. Section 10.10.
    4. Legacy baggage: x86-64 must still support 16-bit real mode at power-on.
6. Where experts disagree: some argue variable-length decode caps how wide an x86 front end can practically get, others point to Intel and AMD shipping 8-wide decode in 2024 and say the ceiling keeps moving. Both sides have a point and the argument is not settled.

Feature	x86-64	ARM64	RISC-V
Year	2000 spec	2011	2010 start
Designer	AMD	Arm Ltd	UC Berkeley
Licence	Cross-licence	Paid licence	Open, BSD terms
Registers	16 integer	31 integer	32 integer

Feature	x86-64	ARM64	RISC-V
Instr length	1 to 15 bytes	4 bytes fixed	4 or 2 bytes
Typical use	PC, server	Phone, Mac	Embedded, chips

7. Two footnotes to that table. RISC-V's embedded variant has 16 registers rather than 32, and its 2-byte instruction length comes from the optional C compressed extension, not the base set.
8. Dates to keep straight. AMD published the x86-64 specification in 2000 and shipped Opteron in April 2003. Intel shipped its compatible EM64T in the Nocona Xeon in June 2004. Arm announced ARMv8-A with AArch64 in October 2011. RISC-V began at Berkeley in 2010 under Krste Asanovic and David Patterson, its unprivileged ISA was ratified as version 20191213, and RISC-V International moved to Switzerland, with the rename completed in March 2020.
9. Tools: `gcc -march=native -Q --help=target` lists the ISA extensions the compiler will use, and on Linux the `flags` line in `/proc/cpuinfo` lists what the chip actually supports.

10.4.6 WORDS - remember these

1. ISA — the list of orders and how they are written — architectural contract defining instructions, registers, memory model and exceptions.
2. Opcode — the part that says what to do — the operation field of an instruction word.
3. Operand — the part that says what to do it to — register number, memory address or constant.
4. Immediate — a number written into the order itself — constant encoded in the instruction, not fetched.
5. Addressing mode — the recipe for finding the data — the rule turning instruction fields into an effective address.
6. RISC — few simple orders, all the same size — reduced instruction set computer, load-store, fixed length.
7. CISC — many orders, many sizes — complex instruction set computer with memory operands and variable length.
8. Microarchitecture — how one chip actually implements the promise — the internal design, invisible to software.

10.5 Registers in real CPUs

10.5.1 PLAIN - in simple words

1. A **register** is a tiny box inside the CPU that holds one number.
2. There are very few of them. A modern CPU has a few dozen that programs can name, not thousands.
3. They are the only storage the arithmetic unit can reach in the same tick of the clock. Everything else is further away.
4. Some registers are general: you can put anything in them.
5. Some have a fixed job. One always points at the next instruction. One always points at the top of the stack. One holds yes or no answers about the last calculation.
6. Almost all real work happens in registers. Data is pulled in from memory, chewed on in registers, and pushed back out.
7. A good compiler spends most of its effort deciding which values live in registers and which have to spill out to memory.

10.5.2 PLAIN - a picture in your head

1. Think of a carpenter at a bench.
2. The registers are the two or three tools actually in their hands.
3. The L1 cache is the tool tray at the edge of the bench, an arm's reach away.
4. Main memory is the tool cupboard on the other side of the workshop.

5. Storage is the hardware shop down the road.
6. Every step out costs time. Hands are instant. The tray takes a second. The cupboard takes a minute. The shop takes a day.
7. So the carpenter arranges work to keep what is needed in their hands. That is exactly what a compiler's register allocator does.

Where this comparison breaks: hands hold whole tools, and registers hold only fixed-size numbers, typically 64 bits. Also a carpenter can put a tool down anywhere, while a register value must be explicitly written somewhere or it is simply overwritten and lost.

10.5.3 PLAIN - a worked example

1. On x86-64 the general purpose registers have odd names, because they grew from 1978 by accident rather than by plan.
2. The 8086 of 1978 had eight 16-bit registers, and each name meant something.

Name	Stood for	Old special job
AX	Accumulator	Results of maths
BX	Base	A base address
CX	Counter	Loop counts
DX	Data	Second half of results
SI	Source index	Source of a copy
DI	Destination index	Target of a copy
BP	Base pointer	Bottom of a stack frame
SP	Stack pointer	Top of the stack

3. In 1985 the 80386 widened them to 32 bits and put an E in front: EAX, EBX, and so on. E stands for extended.
4. In 2003 x86-64 widened them to 64 bits and put an R in front: RAX, RBX, and so on. It also added eight plain-numbered ones, R8 to R15.
5. So RAX, EAX, AX, AH and AL are all the same physical register, seen through windows of 64, 32, 16, 8 and 8 bits.
6. ARM64 refused all of that. It has 31 registers with no story attached: X0 to X30 for the 64-bit view, W0 to W30 for the low 32 bits.
7. There is no X31. That slot means either the constant zero or the stack pointer, depending on the instruction. Reading the zero register always gives 0, and writing to it throws the value away.

10.5.4 PLAIN - what is really happening inside

1. Why are registers so fast. Four reasons, all physical.
2. They are physically next to the arithmetic unit, often microns away. Signal travel time is close to nothing.
3. They are addressed by a tiny number, three to five bits. That decode is a few gates, not a lookup in a table of tags like a cache.
4. They are built from fast, wide SRAM cells with many ports, so several registers can be read and written in the same cycle.
5. There are very few of them, so the array is small, so its wires are short, so it settles quickly.
6. The cost of all that is area and power per bit. A register file bit costs far more silicon than a cache bit, which is why you cannot have thousands.
7. There is one more thing hiding here. The names in the program are not the real boxes. A modern CPU has hundreds of real registers and shuffles which physical box currently holds RAX. That is register renaming, section 10.8.

10.5.5 TECHNICAL - the engineer's version

1. x86-64 architectural integer state: 16 general purpose registers of 64 bits, RIP as the instruction pointer, RFLAGS as the status register.
2. Writing a 32-bit sub-register zero-extends into the full 64 bits. Writing an 8-bit or 16-bit sub-register does not, and leaves the top bits alone. That asymmetry is a standard, written in the architecture manual, and it causes real partial-register stalls if you get it wrong.
3. RFLAGS bits worth knowing: CF is bit 0 carry, PF bit 2 parity, ZF bit 6 zero, SF bit 7 sign, IF bit 9 interrupt enable, DF bit 10 direction, OF bit 11 overflow.
4. ARM64 architectural integer state: X0 to X30, plus SP, plus PC which is not a general register and cannot be written directly. Condition flags live in PSTATE as N, Z, C, V.
5. By convention, not by hardware rule, ARM64 uses X30 as the link register holding the return address, and X29 as the frame pointer.
6. Calling conventions are conventions, set by an ABI document, not by the chip. The System V ABI used on Linux and macOS passes integer arguments in RDI, RSI, RDX, RCX, R8, R9 and returns in RAX. Microsoft's x64 ABI uses RCX, RDX, R8, R9. ARM64's AAPCS64 passes in X0 to X7 and returns in X0.
7. Physically, the architectural names map onto a much larger physical register file. AMD Zen 5 has a floating point physical register file of 384 entries, doubled from 192 in Zen 4, and an integer file of a couple of hundred entries.

Storage	Typical latency	Typical size
Register	Under 1 cycle	16 x 64 bits
L1 data cache	4 to 5 cycles	48 KB
L2 cache	14 cycles	1 MB
L3 cache	40 to 50 cycles	32 MB per die
DDR5 main memory	70 to 90 ns	32 GB

8. Those latency figures are for AMD Zen 5 on the Ryzen 9000 series, 2024. At 5.7 GHz one cycle is 0.175 nanoseconds, so an L1 hit is about 0.7 ns and a memory access is over 400 times slower.
9. Tools: `gdb` command `info registers` prints them live. On Linux, `perf stat -e ld_blocks.no_sr` and similar counters expose partial register and store-forwarding stalls on Intel parts.

10.5.6 WORDS - remember these

1. Register — a tiny box for one number inside the CPU — architecturally named fast storage, typically 64 bits wide.
2. General purpose register — one you can use for anything — an integer register with no fixed hardware role.
3. Stack pointer — marks the top of the working pile — RSP on x86-64, SP on ARM64.
4. Flags register — a strip of yes or no answers — RFLAGS on x86-64, the NZCV bits of PSTATE on ARM64.
5. Zero register — a tap that always gives zero — XZR and WZR on ARM64, no equivalent on x86-64.
6. Spill — running out of hands and putting something down — the compiler storing a value to the stack because registers ran out.
7. ABI — the agreement about who passes what where — application binary interface, a convention, not part of the ISA.

10.6 Microcode

10.6.1 PLAIN - in simple words

1. Some instructions are simple, and the hardware can do them directly.
2. Some are not. "Copy this whole block of memory to there" is one order for the programmer but hundreds of steps for the hardware.
3. So inside the CPU there is a second, smaller, more private instruction set.
4. A complicated outside instruction is turned into a short program of these small private steps. Those steps are called **micro-operations**.
5. The little program that produces them is the **microcode**.
6. This is why one x86 instruction can take one clock cycle or one hundred. Some are a single step, and some are a whole routine.
7. The important consequence is this: some of the behaviour of a CPU is stored, not wired. Stored things can be changed later.
8. That is how a firmware update can change how a chip behaves without anyone touching the silicon.

10.6.2 PLAIN - a picture in your head

1. Picture a restaurant kitchen again. The menu is the instruction set.
2. Some menu items are one action: "pour a glass of water".
3. Some are not: "make a three-course meal for six". The chef has a card file of standard sub-recipes and works through them in order.
4. The card file is the microcode. It sits in the kitchen, not on the menu.
5. Customers never see the cards. They only see the menu, which never changes.
6. If a sub-recipe turns out to be wrong, the owner can replace those cards overnight. The menu still says the same words the next morning.

Where this comparison breaks: the chef can improvise, and the microcode sequencer cannot. Also, most modern instructions skip the card file entirely and are handled by hardwired fast decoders, because going through microcode is slow. Microcode is the exception path, not the normal path.

10.6.3 PLAIN - a worked example

1. Take the x86-64 instruction `add [rax], rbx`, which means "add RBX to the number in memory at the address in RAX".
2. To a programmer that is one instruction, four bytes.
3. Inside, it becomes roughly three micro-operations.

```
1. load  t0 <- MEM[rax]        (read 8 bytes)
2. add   t0 <- t0 + rbx        (do the arithmetic)
3. store MEM[rax] <- t0        (write 8 bytes back)
```

4. `t0` is a temporary that has no name in the instruction set. Programs cannot see it. It exists only for the duration of the instruction.
5. Now take `rep movsb`, which copies a whole block of bytes. That is a microcoded loop. It may issue thousands of micro-operations from one instruction, and the exact number depends on the block length.
6. Compare with ARM64. The same job needs three separate instructions written out by the compiler, one load, one add, one store. The work is identical. The difference is who wrote the three steps down: the microcode, or the compiler.

10.6.4 PLAIN - what is really happening inside

1. The front end of an x86 core has two paths.
2. The fast path: several simple hardware decoders, each turning one instruction into one to four micro-operations directly.
3. The slow path: the microcode sequencer, a small read-only memory holding canned sequences, used for anything complicated.

4. Decoding x86 is expensive, so cores keep a **micro-op cache**. Once an instruction has been decoded, the resulting micro-operations are stored.
5. The next time round a loop, the front end reads finished micro-operations straight out of that cache and skips decoding entirely.
6. That saves a lot of power, and it also delivers more operations per cycle than the decoders can.
7. A microcode update is a signed blob of data. Firmware loads it into a small patch memory very early in boot. It is not stored on the chip permanently.
8. It disappears at every power-off and must be loaded again next boot, by the motherboard firmware or by the operating system.

10.6.5 TECHNICAL – the engineer's version

1. Maurice Wilkes proposed microprogramming in 1951, at the Manchester University Computer Inaugural Conference. EDSAC 2, running from 1958, was the first microprogrammed machine.
2. IBM made it famous with System/360 in 1964: one ISA implemented by many physically different machines, each with its own microcode. That is the clearest early proof that an ISA is a contract, not a design.
3. On Intel cores, simple decoders handle one-to-one and one-to-four cases. Anything longer goes to the MSROM, the microcode sequencer ROM.
4. Micro-op cache sizes, all implementation details rather than architecture.

Core	Year	Micro-op cache
Sandy Bridge	2011	1,536 micro-ops
Skylake	2015	1,536 micro-ops
Golden Cove	2021	About 4,000 micro-ops
AMD Zen 4	2022	About 6,750 ops
AMD Zen 5	2024	12 ops per cycle out

5. Micro-op fusion and macro-op fusion muddy instruction counting. A compare followed by a branch is commonly fused into one internal operation, so counting retired instructions is not the same as counting work.
6. Microcode update mechanics. On Intel, software writes the physical address of the update to model-specific register 0x79 and the load happens. On AMD the equivalent is MSR 0xC0010020. Updates are encrypted and signed. A CPU will reject an unsigned patch.
7. Limits, and this matters. Microcode cannot add execution units, cannot change cache sizes, and cannot rewrite the hardwired fast decoders. It patches sequences and it can flip configuration bits. The number of patch slots is finite.
8. Spectre and Meltdown were disclosed on 3 January 2018. CVE-2017-5753 is Spectre variant 1, bounds check bypass. CVE-2017-5715 is Spectre variant 2, branch target injection. CVE-2017-5754 is Meltdown.
9. Intel's microcode updates added entirely new architectural controls: IA32_SPEC_CTRL exposing IBRS and STIBP, and IA32_PRED_CMD exposing IBPB. A firmware update added new registers to a shipped ISA. That is unusual.
    1. IBRS restricts indirect branch prediction across privilege levels.
    2. IBPB flushes predictor state at a chosen barrier point.
    3. STIBP stops one hardware thread steering the other thread's predictor.
10. Google published retpoline on 4 January 2018, a purely software fix for variant 2 that replaces indirect jumps with a return trampoline the predictor cannot usefully steer.
11. Meltdown was not fixed by microcode at all. It was fixed in software by kernel page-table isolation, which unmaps kernel memory while user code runs, based on the 2017 KAISER work from TU Graz.

12. The rollout went badly. Intel's January 2018 microcode caused unexpected reboots, and Microsoft shipped an update on 29 January 2018 that disabled it. Measured performance drops of 2 to 14 percent were reported on eighth-generation Core platforms, depending heavily on workload.
13. Observe it: `grep microcode /proc/cpuinfo` on Linux gives the loaded revision, `dmesg | grep -i microcode` shows the load at boot, and `iucode_tool -l` lists available Intel updates.

The honest version: saying "the CPU runs microcode" suggests every instruction goes through an interpreter. It does not. On a modern x86 core the large majority of instructions never touch the microcode sequencer, and on ARM64 cores the sequencer is far smaller still. Microcode is the escape hatch for complicated and rare cases, plus a patching mechanism.

10.6.6 WORDS - remember these

1. Micro-operation — one small private step — the internal RISC-like operation an instruction is broken into, often written uop.
2. Microcode — the card file of stored sub-recipes — ROM sequences that expand complex instructions into micro-operations.
3. Micro-op cache — remembering the decode so you need not redo it — a cache of decoded micro-operations, also called the op cache or DSB.
4. Microcode update — an overnight replacement of some cards — a signed, volatile patch loaded by firmware at every boot.
5. IBRS, IBPB, STIBP — three new switches added by a patch — speculation controls introduced by 2018 microcode for Spectre variant 2.
6. KPTI — hiding the kernel's map from user code — kernel page-table isolation, the software fix for Meltdown.

10.7 Pipelining

10.7.1 PLAIN - in simple words

1. A simple CPU wastes almost all of itself, almost all of the time.
2. While it is fetching an instruction, the arithmetic unit sits idle. While it is doing arithmetic, the fetch hardware sits idle.
3. **Pipelining** fixes that. Break the work into stages, and keep every stage busy on a different instruction at the same time.
4. It does not make any one instruction faster. Each instruction still takes the same number of stages to get through.
5. It makes instructions come out more often. That is throughput, not latency.
6. If the pipeline has five stages and nothing goes wrong, you finish one instruction every clock cycle instead of one every five.
7. Every CPU made since the 1980s is pipelined. It is not optional.

10.7.2 PLAIN - a picture in your head

1. You have four loads of washing. Each load needs washing, drying, folding.
2. Each of the three jobs takes 30 minutes.
3. The naive way: wash, dry and fold load 1 completely, then start load 2. That is 90 minutes each, 360 minutes in total.
4. The pipelined way: as soon as load 1 leaves the washer, load 2 goes in.
5. Now the timeline looks like this, in 30-minute slots.

```
slot   1     2     3     4     5     6
wash  L1    L2    L3    L4
dry         L1    L2    L3    L4
fold              L1    L2    L3    L4
```

6. Total: 6 slots, 180 minutes, not 360. Twice as fast for four loads.
7. With 100 loads it approaches three times as fast, the number of stages.
8. Notice load 1 still took 90 minutes. Nothing got faster. Things just overlapped.

Where this comparison breaks: laundry loads are independent, and instructions are not. Load 2 might need the result of load 1. Worse, sometimes you do not know which load comes next until load 1 finishes. Those two problems are data hazards and control hazards, and they are the whole difficulty.

10.7.3 PLAIN - a worked example

1. The classic teaching pipeline has five stages, from the Berkeley and Stanford RISC designs of the early 1980s.

Stage	Short name	What it does
1	IF	Fetch instruction
2	ID	Decode, read registers
3	EX	Do arithmetic or address
4	MEM	Access data memory
5	WB	Write result to register

2. Now run five instructions and watch the pipeline fill and drain.

```
cycle    1   2   3   4   5   6   7   8   9
instr 1  IF  ID  EX  MEM WB
instr 2      IF  ID  EX  MEM WB
instr 3          IF  ID  EX  MEM WB
instr 4              IF  ID  EX  MEM WB
instr 5                  IF  ID  EX  MEM WB
```

3. Five instructions finish in 9 cycles, not 25. The first takes 5 cycles to appear, then one appears every cycle.
4. The formula for n instructions in a k-stage pipeline is k plus n minus 1 cycles, against n times k without pipelining.
5. For n equals 5 and k equals 5: 9 cycles against 25. Speedup 2.78.
6. For n equals 1,000 and k equals 5: 1,004 cycles against 5,000. Speedup 4.98, which is almost the full 5.
7. There is a second, bigger win. Each stage now does one fifth of the work, so each stage settles in one fifth of the time, so the clock can run roughly five times faster too. Pipelining is why clocks got fast at all.

10.7.4 PLAIN - what is really happening inside

1. Between every pair of stages there is a row of flip-flops, from Chapter 6, holding that stage's result until the next clock edge. These are pipeline registers, and they are the physical dividers.
2. Three things can go wrong. They are called hazards.
3. **Structural hazard**: two instructions want the same piece of hardware in the same cycle. For example, one instruction fetching while another reads data, when there is only one memory port. Fixed by splitting instruction and data caches, which is the modified Harvard idea from section 10.2.
4. **Data hazard**: an instruction needs a result that is not written yet.

```
add r1, r2, r3     ; result of r1 written in stage WB
sub r4, r1, r5     ; needs r1 in stage EX, two cycles earlier
```

5. The fix is **forwarding**, also called bypassing. Extra wires take the answer straight from the output of the arithmetic unit back into its own input, before it has been written anywhere.

6. Forwarding cures most data hazards but not all. If the producer is a load from memory, the value does not exist until the MEM stage, so the consumer must wait one cycle. That wasted cycle is a **stall**, and the empty slot pushed through the pipeline is a **bubble**.
7. **Control hazard**: a branch. Until the branch is resolved, the fetch unit does not know which instruction comes next.
8. In a five-stage pipeline you lose two or three cycles per branch if you just wait. Since roughly one instruction in five or six is a branch, that is a large loss, and it is why branch prediction exists.

10.7.5 TECHNICAL - the engineer's version

1. The IBM 7030, known as Stretch and delivered in 1961, was the first heavily pipelined machine, and it already fetched past unresolved branches.
2. Pipeline depth by generation, all measured in stages of the integer pipeline, which is the usual quoted figure.

Core	Year	Stages
Classic MIPS R2000	1985	5
Intel Pentium	1993	5
Pentium Pro	1995	12 to 14
Pentium 4 Willamette	2000	20
Pentium 4 Prescott	2004	31
Intel Core 2	2006	14

3. The cautionary tale is Intel NetBurst. Willamette shipped in November 2000 with a 20-stage pipeline. Prescott followed in February 2004 with 31 stages. The design goal was very high clock speed, and Intel spoke publicly about reaching 10 GHz.
4. It did not work, for three reasons that all reinforce each other.
    1. Deeper pipelines make every branch misprediction more expensive, because more work must be thrown away. Prescott's penalty was well over 30 cycles.
    2. Every extra stage adds pipeline registers, and those flip-flops burn power and add setup and hold overhead that eats the timing gain.
    3. Leakage current at 90 nm rose faster than expected, so the promised clock speeds never arrived. Intel stopped at 3.8 GHz in November 2004.
5. Intel cancelled the successors, Tejas and Jayhawk, in 2004. They were reported to target between 40 and 50 stages. Top NetBurst parts had a thermal design power of 115 W, which was extreme for the time.
6. Intel abandoned NetBurst and shipped the Core microarchitecture in July 2006, which went back to a 14-stage pipeline at lower clocks and was substantially faster in real work.
7. The lesson, stated properly: pipeline depth trades clock frequency against misprediction cost and power. There is an optimum, and it is well short of 30 stages for general-purpose code. Modern cores sit at roughly 14 to 20 stages from fetch to retire.
8. Measurement: `perf stat -e cycles,instructions,stalled-cycles-frontend, stalled-cycles-backend` on Linux separates front-end stalls, which are usually fetch and branch problems, from back-end stalls, which are usually memory and dependency problems.

10.7.6 WORDS - remember these

1. Pipelining — overlapping the stages of different instructions — splitting the datapath into stages separated by pipeline registers.
2. Throughput — how often results come out — instructions completed per unit time.
3. Latency — how long one thing takes end to end — cycles from issue to completion for a single instruction.

4. Hazard — a reason the overlap cannot happen — structural, data or control conflict preventing the next stage from proceeding.
5. Forwarding — passing the answer straight back — bypass paths routing an ALU result to a dependent instruction before write-back.
6. Stall — waiting — holding earlier stages while a dependency resolves.
7. Bubble — an empty slot moving down the pipe — an injected no-operation filling a stalled stage.
8. Pipeline depth — how many stages there are — the count of stages from fetch to retire, a trade-off with clock speed.

10.8 Going wider and smarter

10.8.1 PLAIN - in simple words

1. Pipelining gets you to about one instruction per cycle. Then what.
2. You build more of everything. Two adders, three adders, six adders. Fetch several instructions at once. Finish several at once.
3. That is **superscalar**: more than one instruction completed per cycle.
4. But instructions often wait for each other. If instruction 2 needs the answer from instruction 1, doubling the hardware helps nothing.
5. So the CPU looks further ahead. If instruction 5 is ready and instruction 2 is not, it runs instruction 5 first. That is **out-of-order execution**.
6. It still has to pretend everything happened in order, in case something goes wrong. So results are held back and released in the original order.
7. And when it reaches a branch and does not yet know the answer, it guesses, and carries on as if the guess were right. That is **speculation**.
8. If the guess was right, the work is kept. If wrong, it is all thrown away and the CPU restarts down the other road.
9. Modern CPUs guess correctly the great majority of the time, which is the only reason this works at all.

10.8.2 PLAIN - a picture in your head

1. Picture a busy kitchen with six cooks instead of one.
2. Orders arrive on a rail in strict order. A dispatcher reads them.
3. Order 2 needs a sauce that order 1 is still making, so the dispatcher skips it and gives order 5, which needs nothing, to a free cook.
4. Finished dishes go to a shelf, not to the customer. The shelf is served strictly in the original order, so the customers never notice the reshuffle.
5. That shelf is the **reorder buffer**.
6. Now the head waiter sees a customer approaching the till and guesses they will order the usual. The kitchen starts making it before they speak.
7. Nine times out of ten the waiter is right and the food is already there. One time in ten the food is binned.

Where this comparison breaks: binned food in a real kitchen is wasted money, and binned speculative work in a CPU is only wasted time and energy. But it is not perfectly invisible. The traces speculation leaves in the caches are exactly what Spectre reads, which is why guessing turned out to be a security problem in 2018.

10.8.3 PLAIN - a worked example

1. Look at why renaming is needed. Take this sequence.

```
1: mul  rax, rbx      ; slow, takes several cycles
2: mov  rcx, rax      ; needs the result of 1
```

```
3: mov  rax, 5        ; writes rax again, needs nothing
4: add  rdx, rax      ; needs the new rax
```

2. Instruction 3 does not depend on anything. It could run immediately.
3. But it writes RAX, and instruction 2 still needs the old RAX. If 3 ran early, it would destroy the value 2 is waiting for.
4. This is a false dependency. The two instructions do not share data, only a name. It is a collision of labels, not of information.
5. The fix: give them different physical boxes. Say RAX currently lives in physical register P17. Instruction 3 is told to write P42 instead, and from then on RAX means P42.
6. Now instruction 2 reads P17 and instruction 3 writes P42. They are independent, and both can run at once.
7. That relabelling is **register renaming**, and modern cores do it for every single instruction, every cycle.
8. Now count the win. Suppose the multiply takes 4 cycles. Without renaming the sequence takes about 7 cycles. With renaming, instructions 3 and 4 run in parallel with the multiply and the sequence takes about 5.

10.8.4 PLAIN - what is really happening inside

1. Here is the shape of a modern core's back end, in order.
2. Fetch and predict. The branch predictor supplies a stream of addresses; the fetcher pulls bytes from the L1 instruction cache.
3. Decode into micro-operations, or read them from the micro-op cache.
4. Rename: map each architectural register name onto a free physical register, using a register alias table.
5. Allocate: give each operation a slot in the reorder buffer, which keeps program order, and a slot in a scheduler.
6. Schedule and issue: **reservation stations**, or a unified scheduler, hold operations until every input is ready, then fire them at a free unit.
7. Execute: the arithmetic units, address generation units, load and store units, and vector units all work in parallel.
8. Write back the result into the physical register, and mark the reorder buffer slot as complete.
9. Retire, in strict program order. Only at retirement do results become architecturally real. If an exception happened, everything after it is discarded and never becomes visible.
10. That last rule is what makes exceptions **precise**: the software sees a clean, ordered machine even though the hardware ran a mess.

10.8.5 TECHNICAL - the engineer's version

1. History. The CDC 6600 of 1964, designed by Seymour Cray, had ten functional units and a scoreboard to track dependencies. Robert Tomasulo's 1967 algorithm for the IBM System/360 Model 91 added reservation stations and register renaming, and it is still the basis of every design today.
2. The Intel Pentium of March 1993 was the first superscalar x86, 2-wide with its U and V pipes. The Pentium Pro of November 1995 was the first out-of-order x86.
3. Modern structure sizes, all implementation details, not architecture.

Structure	Zen 5, 2024	Golden Cove, 2021
Decode width	8, as two of 4	6
Micro-op cache out	12 per cycle	8 per cycle
Reorder buffer	448 entries	512 entries
Integer ALUs	6	5

4. Branch prediction, in order of invention.

1. Static: always predict not taken, or the backward-taken and forward-not-taken heuristic, which suits loops. No storage needed.
2. One-bit: remember the last outcome per branch. A loop of N iterations mispredicts twice, at entry and exit.
3. Two-bit saturating counter: James E. Smith, 1981. Four states, from strongly not taken to strongly taken. It takes two wrong outcomes to change the prediction, so a loop mispredicts once, not twice.
4. Two-level adaptive: Tse-Yu Yeh and Yale Patt, 1991. Index a table of counters using a history register of recent branch outcomes.
5. gshare: Scott McFarling, 1993. Combine the branch address with the global history by XOR, which uses the table far better.
6. Perceptron: Daniel Jimenez and Calvin Lin, 2001. A simple neural predictor, able to use very long histories.
7. TAGE: Andre Seznec and Pierre Michaud, 2006. Several tagged tables indexed with geometrically increasing history lengths, so short and long correlations are both captured. TAGE-SC-L, with a statistical corrector and a loop predictor, won the Championship Branch Prediction contests and is the basis of what ships today.

5. Real accuracy, in three honest lines.
   1. On many workloads modern predictors get above 99 percent of conditional branches right.
   2. On hard integer code the figure is lower. The 2019 paper "Branch Prediction Is Not A Solved Problem" measured about 95 percent average accuracy for an 8 KB TAGE-SC-L across SPEC CPU 2017 integer benchmarks.
   3. That same paper argues the remaining misses are worth an 18.5 percent instructions-per-cycle opportunity at current core sizes, rising to 55.3 percent for a core four times wider.
6. Real cost of one miss. Measured on Intel Skylake: 16.5 cycles average when the correct path hits in the micro-op cache, 19 to 20 cycles when it does not. At 5 GHz that is roughly 3.3 to 4 nanoseconds. On a core that can retire 6 instructions per cycle, a single misprediction throws away on the order of 100 instruction slots.
7. Two-bit counter state machine, worth memorizing.

```
   taken ->      taken ->      taken ->
[00]--------->[01]--------->[10]--------->[11]
strong NT     weak NT       weak T        strong T
  <--------     <--------     <--------
  not taken     not taken     not taken
predict NT    predict NT    predict T     predict T
```

8. Measurement: `perf stat -e branches,branch-misses ./program` gives the real misprediction rate for your code. A rate above about 2 percent on hot code usually means a data-dependent branch that should be turned into branchless arithmetic or a conditional move.

10.8.6 WORDS - remember these

1. Superscalar — more than one instruction finished per cycle — multiple issue, with duplicated execution units.
2. Out-of-order execution — run whatever is ready — dynamic scheduling with in-order retirement.
3. Register renaming — giving the same name different boxes — mapping architectural registers to a larger physical register file.
4. Reorder buffer — the shelf that restores order — a circular buffer holding results until in-order retirement.
5. Reservation station — a waiting bay for an operation — a scheduler entry holding a micro-op until its operands are ready.
6. Speculation — acting on a guess — executing past an unresolved branch and discarding on a miss.

7. Branch predictor — the thing that guesses — hardware predicting direction and target, today TAGE-style.
8. Precise exception — software sees a tidy machine — the guarantee that on a fault, all earlier instructions completed and none later did.

10.9 SIMD and accelerators

10.9.1 PLAIN - in simple words

1. Very often a program does the same sum to a long list of numbers.
2. Brighten every pixel. Add every element of two arrays. Scale every audio sample.
3. Doing them one at a time wastes the machine, because the operation is identical each time. Only the data changes.
4. So CPUs got wide registers that hold several numbers side by side, and instructions that do the same sum to all of them at once.
5. This is **SIMD**: single instruction, multiple data.
6. A 256-bit register holds eight 32-bit numbers. One add instruction adds all eight pairs in one go. Eight times the work, one instruction.
7. That is not the same as having eight cores. It is one core doing one instruction that happens to be eight-wide.
8. Later, chips added units for specific jobs: encryption, and now matrix multiply for neural networks.

10.9.2 PLAIN - a picture in your head

1. A normal instruction is a single hole punch: one hole per press.
2. A SIMD instruction is a row of eight punches on one bar. One press, eight holes, all in line.
3. The catch is obvious. The paper must be lined up. If your eight numbers are scattered around memory, you spend more time gathering them into a row than you save by punching them together.
4. Also, the bar punches all eight or none. If you only want six of them, you need a mask that blocks two of the punches.

Where this comparison breaks: the punch bar is fixed at eight, but Arm's SVE deliberately does not fix the width. The same program runs on hardware with 128-bit or 512-bit vectors without recompiling. That is a genuinely different design idea, not just a bigger bar.

10.9.3 PLAIN - a worked example

1. Add two arrays of eight floating point numbers, using AVX on x86-64. RDI points to array a, RSI to array b, RDX to the result.

```
vmovups ymm0, [rdi]        ; load 8 floats from a
vmovups ymm1, [rsi]        ; load 8 floats from b
vaddps  ymm0, ymm0, ymm1   ; 8 adds in one instruction
vmovups [rdx], ymm0        ; store 8 results
```

2. Four instructions do 8 additions plus 24 element moves. The scalar version would need a loop of 8 iterations with about 4 instructions each.
3. Same idea in ARM64 NEON, which is 128 bits wide, so four floats at a time.

```
ld1  {v0.4s}, [x0]         ; load 4 floats
ld1  {v1.4s}, [x1]         ; load 4 floats
fadd v0.4s, v0.4s, v1.4s   ; 4 adds at once
st1  {v0.4s}, [x2]         ; store 4 results
```

4. Read `v0.4s` as "register v0 treated as 4 single-precision values". Change it to `v0.2d` and the same register is 2 doubles. Change it to `v0.16b` and it is 16 bytes. The bits do not move. Only the interpretation changes, which is the numbers-and-meaning idea from Chapter 7.
5. With AVX-512, ZMM registers are 512 bits, so the same loop handles 16 floats per instruction.
6. Real speedups are usually below the theoretical multiple. Memory bandwidth, alignment and loop overhead all take a cut. Two to six times is typical for an eight-wide operation on real code.

10.9.4 PLAIN - what is really happening inside

1. Physically, a SIMD unit is several copies of the same arithmetic circuit, fed from one wide register and controlled by one decoded instruction.
2. There is only one instruction fetch, one decode, one schedule. That saving is a large part of why SIMD is efficient in energy, not just in time.
3. Wide units are hot. Running 512-bit operations flat out draws so much current that some Intel chips reduced their clock frequency while doing it, a well-documented behaviour on Skylake-based Xeon parts from 2017.
4. Then there are fixed-function accelerators. Instead of a general unit, put an entire algorithm in silicon.
5. AES encryption is the classic case. One instruction performs one complete round of the cipher, something that takes dozens of ordinary instructions.
6. Now the same idea has arrived for neural networks. The core operation there is multiplying matrices, so chips now include matrix units and separate neural processing units on the same die.
7. Those units are not general. They do one shape of arithmetic extremely efficiently and are useless for anything else.

10.9.5 TECHNICAL - the engineer's version

1. The term SIMD comes from Michael Flynn's 1966 taxonomy of parallel machines.
2. The x86 vector history, with widths and dates.

Extension	Year	Width
MMX	1997	64 bits, integer only
SSE	1999	128 bits, single float
SSE2	2000	128 bits, double, integer
AVX	2011	256 bits
AVX2	2013	256 bits, integer, FMA
AVX-512	2016 to 2017	512 bits, 32 registers

3. MMX arrived with the Pentium MMX in January 1997 and reused the x87 floating point registers, so you could not mix the two.
4. SSE arrived with the Pentium III in February 1999 and finally gave separate XMM registers. AVX arrived with Sandy Bridge in January 2011 and introduced the three-operand VEX encoding, so the destination need not also be a source.
5. AVX-512 first shipped in Knights Landing in 2016 and in Xeon Skylake-SP during 2017. Consumer support has been inconsistent: present on Rocket Lake, disabled on Alder Lake, and supported on AMD Zen 4 from 2022 and Zen 5 from 2024, with Zen 5 desktop parts having a full 512-bit wide data path. Intel announced AVX10 in July 2023 to tidy this up.
6. Arm's line runs in parallel.
    1. Advanced SIMD, marketed as NEON, is part of ARMv7-A and mandatory in ARMv8-A, with 32 registers of 128 bits.
    2. SVE was announced at Hot Chips in August 2016. It is vector-length agnostic, permitting 128 to 2048 bits in 128-bit steps.

    3. Fujitsu's A64FX, used in the Fugaku supercomputer from 2020, implements 512-bit SVE. SVE2 arrived with Armv9-A, announced in March 2021.
7. AES-NI arrived with Intel Westmere in January 2010. Six instructions: AESENC, AESENCLAST, AESDEC, AESDECLAST, AESIMC and AESKEYGENASSIST. One AESENC performs a complete AES round. Beyond speed, the crucial property is that it is constant time and uses no lookup tables, which removes the cache-timing side channels that plagued table-based software AES. Armv8 has the equivalent AESE and AESD instructions.
8. Matrix and neural units on the same die.
    1. Intel Advanced Matrix Extensions, AMX, shipped in Sapphire Rapids Xeon in January 2023: eight tile registers of 1 KB each with a TMUL unit.
    2. Arm's Scalable Matrix Extension, SME, is part of Armv9 and appears in recent Apple silicon.
    3. Separate NPUs: Intel Meteor Lake in December 2023 was the first Core part with one; Lunar Lake in September 2024 raised it to 48 TOPS. Microsoft's Copilot+ PC badge requires at least 40 TOPS.
    4. In phones, the Snapdragon 8 Elite Gen 5, announced 24 September 2025, has a Hexagon NPU that Qualcomm states is 37 percent faster than the previous generation. Apple's A19 Pro, from September 2025, has a 16-core Neural Engine.
9. Separate what is what, as the honesty rules require.
    1. Established fact: the silicon exists, it multiplies matrices at low precision far more efficiently than the general CPU cores, and it is shipping in ordinary phones and laptops today.
    2. Active research: how to split a model across CPU, GPU and NPU, which number formats to use, and how to keep memory bandwidth from becoming the limit.
    3. Marketing claim: TOPS figures. A TOPS number depends entirely on the number format assumed, is usually a peak that no real workload reaches, and is not comparable between vendors. Treat it as a rough class, not a measurement.
10. Chapters 46 onwards return to this: what those matrix units actually compute, and why a neural network reduces to matrix multiply.
11. Tools: on Linux `cat /proc/cpuinfo` lists `avx2`, `avx512f`, `aes` and so on in the flags line. On macOS, `sysctl hw.optional` lists the Arm features present.

10.9.6 WORDS - remember these

1. SIMD — one order, many numbers — single instruction, multiple data, a lane parallel execution model.
2. Vector register — a wide box holding several numbers — XMM at 128, YMM at 256, ZMM at 512 bits on x86-64.
3. Lane — one slot in the row — one element position within a vector register.
4. Mask register — a stencil saying which lanes count — k0 to k7 on AVX-512, predicate registers on SVE.
5. Vector-length agnostic — the same code on any width — the SVE model where the hardware vector length is discovered at run time.
6. AES-NI — encryption built into the metal — x86 instructions performing AES rounds directly, constant time.
7. NPU — a chunk of silicon for neural maths — neural processing unit, a fixed-function matrix and activation engine.
8. TOPS — a marketing speed number — tera-operations per second, precision dependent and not comparable across vendors.

10.10 Many cores

10.10.1 PLAIN - in simple words

1. Until about 2005, each new chip simply ran at a higher clock speed.
2. Then that stopped. Clocks have barely moved since. A fast desktop chip in 2004 ran at 3.8 GHz, and a fast desktop chip in 2025 boosts to 5.7 GHz. That is not much in twenty years.

3. The reason is heat. Pushing the clock higher needs more voltage, and the power a chip burns rises with the square of the voltage. The heat became impossible to remove.
4. So instead of one faster CPU, chipmakers put several CPUs on one piece of silicon. Each one is a **core**.
5. Two cores can do two things at once, genuinely, in parallel.
6. That only helps if the software can be split into pieces that run at the same time. Many programs cannot be split much, and for those, more cores do very little.
7. Cores also have to agree about memory. If two cores both hold a copy of the same number and one changes it, the other must find out.
8. Keeping them in agreement is called **cache coherence**, and it is one of the hardest parts of building a modern chip.

10.10.2 PLAIN - a picture in your head

1. One cook working faster has a limit: hands only move so quickly, and past a point the kitchen catches fire.
2. So you hire eight cooks. Now eight dishes can be made at once.
3. But if the recipe says "wait for the stock to reduce for an hour", eight cooks do not make the stock reduce in less than an hour. That step is stubbornly serial.
4. Worse, the cooks share one fridge. If two cooks both take out the butter, and one of them salts it, the other is holding stale butter.
5. So the kitchen needs a rule: before you change something, shout, and everyone else throws their copy away.
6. Shouting takes time, and the more cooks there are, the more shouting.

Where this comparison breaks: cooks can talk to each other in any order, but cores must agree on a strict order of memory events, or programs break in ways that only show up once in a million runs. That is memory ordering, and it is much stricter and much more subtle than shouting.

10.10.3 PLAIN - a worked example

1. Amdahl's law, from Gene Amdahl in 1967, tells you the ceiling.
2. Say 95 percent of your program can run in parallel and 5 percent cannot.
3. The formula for speedup with n cores is 1 divided by ((1 minus p) plus p divided by n), where p is the parallel fraction.
4. With p equal to 0.95, work it out.

Cores	Calculation	Speedup
2	1 / (0.05 + 0.475)	1.90
4	1 / (0.05 + 0.2375)	3.48
8	1 / (0.05 + 0.11875)	5.93
16	1 / (0.05 + 0.059375)	9.14
64	1 / (0.05 + 0.0148)	15.4
Infinite	1 / 0.05	20.0

5. Read the last line again. Even with infinite cores, a program that is 5 percent serial can never go more than 20 times faster.
6. At 16 cores you already get 9.14 of a possible 20. Doubling to 32 gets you to 12.6. The returns collapse quickly.
7. If only 50 percent is parallel, the ceiling is 2 times, no matter what.
8. This single calculation explains why buying more cores often does nothing.

10.10.4 PLAIN - what is really happening inside

1. Take two cores, each with its own L1 cache, sharing memory. Both want the variable X at address A. Follow the standard four-state rule set, MESI.
2. Each cached line is in one of four states: **Modified** (I changed it and nobody else has it), **Exclusive** (I have the only copy and it is clean), **Shared** (others may have it too, clean), **Invalid** (I have nothing).

Step	Action	Core 0 / Core 1
0	Nothing cached	I / I
1	Core 0 reads A	E / I
2	Core 0 writes A	M / I
3	Core 1 reads A	S / S
4	Core 1 writes A	I / M
5	Core 0 reads A	S / S

3. Step by step. At step 1 core 0 misses, fetches the line, and since no other core has it, it takes it Exclusive.
4. At step 2 core 0 writes. Because it already had it Exclusive, no message is needed at all. It flips silently to Modified. Memory is now out of date.
5. At step 3 core 1 asks for the line. Core 0 sees the request, notices it holds a Modified copy, and supplies the fresh data. Both settle at Shared.
6. At step 4 core 1 wants to write. It must first broadcast a request for ownership. Core 0 drops its copy to Invalid. Core 1 becomes Modified.
7. At step 5 core 0 wants to read again, and the whole dance repeats the other way round.
8. Each of those transfers costs tens of cycles. Two cores fighting over one variable in a tight loop is one of the slowest things a modern CPU can do.

10.10.5 TECHNICAL - the engineer's version

1. The physics. Dynamic power is approximately P equals alpha times C times V squared times f. Alpha is switching activity, C capacitance, V supply voltage, f frequency.
2. Dennard scaling, from Robert Dennard's 1974 IBM paper, said that as transistors shrink you can lower V and C in step, so power per square millimetre stays constant while frequency rises. That held for thirty years.
3. It broke down around 2005 to 2006. Threshold voltage could not keep falling without leakage current exploding, so supply voltage stalled near 1 volt, so power density started rising with every shrink.
4. Intel cancelled its 4 GHz Pentium 4 in October 2004 and topped out at 3.8 GHz in November 2004. IBM's POWER4 in 2001 was the first mainstream dual-core chip; AMD's Athlon 64 X2 and Intel's Pentium D both arrived in May 2005. That is the exact moment the industry turned.
5. Simultaneous multithreading, stated honestly. SMT duplicates the architectural state, meaning registers, program counter and flags, so one core presents two logical processors to the operating system. It does not duplicate execution units, caches or TLBs. Those are shared.
    1. The gain comes from filling issue slots that one thread leaves empty while it waits for memory. Typical real throughput gain is 10 to 30 percent, not 100 percent.
    2. It can be negative. Two threads with large working sets thrash the shared L1 and L2 and both run slower than one would alone.
    3. Intel shipped Hyper-Threading on Xeon in February 2002 and on the 3.06 GHz Pentium 4 in November 2002.
    4. It has a security cost. Shared resources leak timing information, which produced attacks including PortSmash in 2018 and the MDS family in

    2019. OpenBSD disabled SMT by default in 2018.

5. Intel dropped SMT entirely on Arrow Lake in October 2024. The Core Ultra 9 285K has 24 cores and exactly 24 threads.

6. Cache coherence. MESI comes from Mark Papamarcos and Janak Patel, 1984, and is often called the Illinois protocol. Real chips extend it: MOESI adds Owned, used by AMD, and MESIF adds Forward, used by Intel. Large chips use a directory rather than broadcast snooping, because broadcast does not scale past a few dozen cores.
7. Memory ordering is a separate problem from coherence. Coherence is about one address. Ordering is about the visible order of accesses to different addresses.
    1. x86-64 uses Total Store Order. Loads are not reordered with loads, stores are not reordered with stores, but a store followed by a load to a different address may appear reordered because of the store buffer.
    2. ARM64 is weakly ordered. Almost any reordering is allowed unless you write a barrier.
    3. Barriers: MFENCE, LFENCE and SFENCE on x86-64; DMB, DSB and ISB on ARM64. In portable code, use the C11 or C++11 atomics with an explicit memory order rather than inline assembly.
    4. This is why lock-free code that works on an Intel laptop can fail on an Apple silicon Mac. The bug was always there. x86-64's stronger model was hiding it.
8. False sharing. Two threads write two different variables that happen to sit in the same 64-byte cache line. There is no logical sharing at all, but the line ping-pongs between cores at tens of cycles per bounce.

```
struct bad  { long a; long b; };            /* same line */
struct good {
    long a; char pad[56];                   /* 8 + 56 = 64 */
    long b;
};
```

9. In C++, `alignas(64)` or `std::hardware_destructive_interference_size` expresses this properly. On Linux, `perf c2c record` and `perf c2c report` find false sharing directly by address.
10. Amdahl's law was presented at the 1967 AFIPS Spring Joint Computer Conference. John Gustafson's 1988 rebuttal points out that in practice people grow the problem to fit the machine, and for a fixed time budget rather than a fixed problem size the scaling looks much better. Both are correct; they answer different questions.
11. Tools: `nproc` and `lscpu` show cores and threads, `taskset` pins a process to specific cores, and `numactl --hardware` shows memory locality on multi-socket machines.

10.10.6 WORDS - remember these

1. Core — one complete CPU on the die — an independent instruction stream with its own registers, L1 and L2.
2. Power wall — the point where heat stopped the clock rising — the end of frequency scaling around 2005 due to power density.
3. Dennard scaling — shrinking used to be free — constant power density under scaling, broken since roughly 2006.
4. SMT — one core pretending to be two — simultaneous multithreading, duplicated architectural state over shared execution resources.
5. Cache coherence — everyone agrees what a given address holds — MESI and its relatives, enforced by snooping or a directory.
6. Memory ordering — the agreed order of unrelated accesses — TSO on x86-64, weak on ARM64, controlled by barriers.
7. False sharing — accidental fighting over one line — unrelated variables in the same 64-byte cache line bouncing between cores.
8. Amdahl's law — the serial part sets the ceiling — speedup equals $1 / ((1 - p) + p/n)$.

10.11 Caches from the CPU side

10.11.1 PLAIN – in simple words

1. Main memory is desperately slow compared to the CPU. Fetching from it takes the time of hundreds of instructions.
2. So the CPU keeps small, fast copies of recently used memory close by. Those are **caches**.
3. There are usually three levels. L1 is tiny and instant, L2 is bigger and slower, L3 is much bigger and slower again.
4. Nothing is copied one byte at a time. Memory is moved in blocks of 64 bytes, called a **cache line**.
5. So if you read one byte, you get its 63 neighbours for free. That is why walking through an array in order is enormously faster than jumping about.
6. Caches work because of two habits real programs have. If you used something recently you will probably use it again soon, and if you used something you will probably use the thing next to it.
7. Those two habits have names: temporal locality and spatial locality. Almost all of computer performance rests on them.

10.11.2 PLAIN – a picture in your head

1. Picture a library with a huge basement and a small desk.
2. You cannot work in the basement. You fetch books up to the desk.
3. Fetching one book takes twenty minutes, so you never fetch one book. You fetch the whole shelf section it sits in. That section is the cache line.
4. Your desk holds four books. The table behind you holds fifty. The room next door holds a thousand. Those are L1, L2 and L3.
5. When your desk is full and you need a new book, you must put one back. Which one you choose is the replacement policy.
6. If you always ask for books that are near each other, you almost never go to the basement. If you ask for random books, you go every time.

Where this comparison breaks: you decide what to fetch, and the CPU mostly does not. The hardware guesses your next request and fetches it before you ask. That guessing is prefetching, and when it works you never see the basement at all.

10.11.3 PLAIN – a worked example

1. Real figures for AMD Zen 5, the Ryzen 9000 desktop series from 2024.

Level	Size	Latency	Time at 5.7 GHz
L1 instruction	32 KB per core	-	-
L1 data	48 KB per core	4 cycles	0.70 ns
L2	1 MB per core	14 cycles	2.46 ns
L3	32 MB per die	about 50 cycles	8.8 ns
DDR5-6000	32 GB	-	about 75 ns

The L1 and L2 cycle counts are AMD's published figures. The L3 figure is approximate: AMD quoted 50 cycles for Zen 4, and independent measurements of Zen 5 land close to it but vary with the part and the memory clock.

2. Read the last two rows together. Going to memory instead of L3 costs about nine times longer. Going to memory instead of L1 costs over a hundred times longer.
3. Now measure it yourself. Walk a large array in order, then in random order.

```
/* sequential: hardware prefetch works, ~1 to 2 ns per element */
for (i = 0; i < n; i++) sum += a[i];

/* random: every access is a cache miss and a TLB risk */
for (i = 0; i < n; i++) sum += a[idx[i]];   /* idx shuffled */
```

4. On a current desktop, with an array well over the L3 size, the sequential loop typically runs at roughly 1 to 2 nanoseconds per element and the random loop at roughly 70 to 120 nanoseconds per element. Those are approximate and machine dependent, but the ratio, somewhere around 50 to 100 times, is consistent everywhere.
5. Same instructions. Same number of additions. Same array. The only difference is the order of the addresses.
6. This is the single most important performance fact in the chapter.

10.11.4 PLAIN - what is really happening inside

1. When the core wants an address, the cache must answer "do I have it" in one or two cycles. It cannot search. It must go straight to one place.
2. So the address is cut into three fields.
    1. The **offset**: which byte inside the 64-byte line.
    2. The **index**: which set of the cache to look in.
    3. The **tag**: the rest of the address, stored alongside the data to prove which line this actually is.
3. **Direct-mapped** means one place per index. Fast and simple, but two hot addresses that share an index knock each other out forever.
4. **Fully associative** means a line may sit anywhere. No conflicts, but you must compare every tag, which is impossibly expensive at any size.
5. **Set-associative** is the compromise, and it is what everyone uses. Each index selects a small set of, say, 8 or 12 slots, and only those tags are compared. You get most of the conflict resistance for a small cost.
6. When a set is full, something must go. The replacement policy picks the victim, usually an approximation of least recently used.
7. On a write, two choices. **Write-through** sends every write onward immediately, which is simple but floods the bus. **Write-back** marks the line dirty and only writes it out when it is evicted. Every modern data cache is write-back.

10.11.5 TECHNICAL - the engineer's version

1. Work the address split by hand for Zen 5's L1 data cache: 48 KB, 12-way set associative, 64-byte lines.
    1. Lines total: 48 x 1024 / 64 = 768.
    2. Sets: 768 / 12 = 64.
    3. Offset bits: log2(64) = 6.
    4. Index bits: log2(64) = 6.
    5. Tag bits: everything above bit 11.
2. Take the address 0x12345678 and split it.

```
0x12345678 = 0001 0010 0011 0100 0101 0110 0111 1000
             |<--------- tag ------->|index | off |
tag   = 0x12345   (bits 31..12)
index = 011001b = set 25   (bits 11..6)
offset= 111000b = byte 56  (bits 5..0)
```

3. Now a direct-mapped counter-example: a 32 KB direct-mapped cache with 64- byte lines has 512 lines, so 9 index bits. Any two addresses exactly 32 KB apart share an index. A loop touching two arrays that happen to be 32 KB apart will miss on every single access. That is a conflict miss, and it is why associativity exists.

4. The three kinds of miss, called the three Cs after Mark Hill's 1987 thesis: compulsory (first ever touch), capacity (working set exceeds the cache) and conflict (fits, but the sets collide).
5. Replacement policies in shipping hardware: true LRU up to about 4 ways; tree-based pseudo-LRU beyond that; RRIP and its dynamic variant from a 2010 ISCA paper for last-level caches, which resist thrashing far better; pure random in some Arm L2 designs, chosen for its tiny cost.
6. Write policy detail: write-back with write-allocate is standard for data caches. A dirty bit per line tracks whether a write-back is needed. Stores to memory marked write-combining, typically video memory, bypass the cache through separate combining buffers.
7. Hardware prefetchers, per level and multiple per level. Next-line, stride detectors that spot a constant step, stream prefetchers that follow several sequences at once, and region-based prefetchers. They can be counter productive on pointer-chasing code, and they can be disabled through model-specific registers on both Intel and AMD parts.
8. Cache line size is 64 bytes on x86-64 and on most Arm cores. Apple's M series uses 128 bytes, which is a real portability trap for anyone padding structures by hand.
9. Inclusion policy is an implementation detail, not a standard. Intel's ring-based designs historically used an inclusive L3; AMD's L3 is a victim cache, holding only lines evicted from L2, which is why AMD's L3 is quoted per core complex die rather than per chip.
10. Tools: `lscpu -C` prints every level with size, ways and line size; `getconf LEVEL1_DCACHE_LINESIZE` prints the line size; `perf stat -e cache-references,cache-misses,LLC-load-misses` measures the real miss rate; `valgrind --tool=cachegrind` simulates it per line of source code.

10.11.6 WORDS - remember these

1. Cache line — the block memory moves in — 64 bytes on x86-64 and most Arm, 128 on Apple silicon.
2. Tag, index, offset — proof, shelf, and position on the shelf — the three fields an address is split into for lookup.
3. Set-associative — a few slots per index — N-way lookup comparing N tags in parallel.
4. Conflict miss — two hot addresses on the same shelf — a miss that would not happen in a fully associative cache of the same size.
5. Write-back — write later, when evicted — a dirty bit marks lines needing a write-out, as opposed to write-through.
6. Prefetching — fetching before you ask — hardware or software issuing loads ahead of demand based on detected patterns.
7. Locality — you reuse things, and their neighbours — temporal and spatial locality, the reason caches work at all.

10.12 Interrupts, exceptions and privilege

10.12.1 PLAIN - in simple words

1. The loop of fetch, decode, execute never stops on its own. Something must be able to break into it.
2. Two kinds of thing break in.
3. An **interrupt** comes from outside: a key was pressed, a network packet arrived, a timer expired. It has nothing to do with the current instruction.
4. An **exception** comes from inside: a divide by zero, an instruction the chip does not know, a memory address that is not mapped.
5. Either way the CPU stops, remembers exactly where it was, jumps to a fixed handler routine, runs it, and then goes back.
6. It also changes mode. Handlers run with more power than ordinary programs.
7. That mode change is the whole basis of operating system security. Your program cannot touch the disk directly. It must ask, and asking means deliberately triggering a controlled jump into the more powerful mode.

10.12.2 PLAIN - a picture in your head

1. Picture a receptionist working through a pile of forms.
2. The phone rings. They put a bookmark in the pile, note exactly where they were, answer the phone, deal with it, and return to the exact same form.
3. The phone number is not decided on the spot. There is a printed list on the wall: "for fire, call this room; for deliveries, call that room". That list is the interrupt vector table.
4. Some rooms need a key the receptionist does not have. To get in, they must go through a specific door with a guard, who checks them and hands over the key on the way in and takes it back on the way out.
5. That guarded door is a system call.

Where this comparison breaks: the receptionist chooses when to look up. A CPU can be interrupted between any two instructions, whether it wants to be or not, and the hardware, not the software, decides when.

10.12.3 PLAIN - a worked example

1. Print thirteen characters to the screen. On Linux, x86-64.

```
mov  rax, 1          ; 1 = the write system call
mov  rdi, 1          ; file 1 = standard output
lea  rsi, [msg]      ; address of the text
mov  rdx, 13         ; how many bytes
syscall              ; cross into the kernel
```

2. The same thing on Linux, ARM64.

```
mov  x8, #64         ; 64 = the write system call
mov  x0, #1          ; file 1 = standard output
adr  x1, msg         ; address of the text
mov  x2, #13         ; how many bytes
svc  #0              ; cross into the kernel
```

3. Notice the shape is identical. Put a number saying which service, put the arguments in agreed registers, execute one special instruction.
4. Notice the register names differ, the call numbers differ, and the instruction differs. All three are conventions of that operating system on that architecture, not laws of the machine.
5. The `syscall` and `svc` instructions do almost nothing themselves. They save the return address, raise the privilege level, and jump to one fixed address the kernel installed at boot. The kernel does the rest.

10.12.4 PLAIN - what is really happening inside

1. Between instructions, the core checks whether an interrupt is pending. If interrupts are enabled and one is pending, it acts.
2. It finishes the instructions already in flight and discards the speculative ones, so the machine has a clean, precise state.
3. It looks up the vector number in a table of handler addresses whose location is held in a special register.
4. It pushes enough state to come back: at minimum the return address and the flags, and on x86 also the stack and code segment selectors and, for some faults, an error code.
5. It raises the privilege level and jumps to the handler.
6. The handler immediately saves everything else it will use, because the hardware only saved the minimum.
7. When finished, a return-from-interrupt instruction restores the saved state and drops the privilege level in one atomic step.
8. All of this costs real time. A modern system call is on the order of tens to a few hundred nanoseconds, and the Meltdown mitigations of 2018 made it noticeably more expensive by forcing a page-table switch on every crossing.

10.12.5 TECHNICAL - the engineer's version

1. x86-64 uses the Interrupt Descriptor Table, located by the IDTR register. It has 256 entries. Vectors 0 to 31 are architecturally defined exceptions; 32 to 255 are available for devices and software.

Vector	Name	Cause
0	#DE	Divide error
6	#UD	Invalid opcode
13	#GP	General protection fault
14	#PF	Page fault

2. Exceptions are classified as faults, which re-run the instruction after the handler, traps, which continue after it, and aborts, which do not return. A page fault is a fault; that is exactly how demand paging works.
3. ARM64 uses a vector table located by VBAR_EL1, with 16 entries of 128 bytes each.
    1. The 16 entries are four exception types, which are synchronous, IRQ, FIQ and SError, crossed with four sources.
    2. The four sources are the current exception level using SP0, the current level using SPx, a lower level in AArch64, and a lower level in AArch32.
    3. State is saved into ELR_EL1 for the return address, SPSR_EL1 for the saved processor state, and ESR_EL1 for the syndrome describing why.
4. Privilege levels. x86-64 defines rings 0 to 3. In practice only ring 0, the kernel, and ring 3, user code, are used; ring 1 and 2 are historical. Virtualization adds VMX root mode below ring 0, informally called ring -1, and System Management Mode sits outside all of it.
5. ARM64 defines exception levels instead: EL0 for applications, EL1 for the kernel, EL2 for a hypervisor, EL3 for the secure monitor. Higher number means more privilege, the opposite direction to x86 rings.
6. SYSCALL on x86-64 was introduced by AMD and is deliberately minimal.
    1. It saves RIP into RCX and RFLAGS into R11, then masks RFLAGS using IA32_FMASK.
    2. It loads the code and stack selectors from IA32_STAR and jumps to the address held in IA32_LSTAR.
    3. It performs no memory reads and no descriptor table lookup. That is precisely why it is fast compared with the old `int 0x80` gate.
7. Because SYSCALL destroys RCX, the Linux x86-64 system call ABI passes its fourth argument in R10 rather than RCX. The number goes in RAX, arguments in RDI, RSI, RDX, R10, R8, R9, and the result comes back in RAX. On ARM64 the number goes in X8, arguments in X0 to X5, result in X0.
8. Kernel page-table isolation, the Meltdown fix, unmaps almost all kernel memory while user code runs, so each crossing writes CR3 and disturbs the TLB. On processors without PCID tagging the cost is severe. This is a concrete case where a security fix has a permanent, measurable performance price.
9. The vDSO exists to avoid the crossing entirely for cheap, common calls. Linux maps a small shared page into every process so that `clock_gettime` and similar can read a kernel-updated timestamp without a system call.
10. Tools: `strace ./program` traces every system call and its arguments, `cat /proc/interrupts` shows interrupt counts per CPU per device, and `perf trace` gives timing per call.

10.12.6 WORDS - remember these

1. Interrupt — something outside wants attention — an asynchronous event from a device, delivered between instructions.
2. Exception — the current instruction went wrong — a synchronous fault, trap or abort caused by execution itself.
3. Vector table — the printed list of who to call — IDT on x86-64, the table at VBAR_EL1 on ARM64.

4. Context switch — putting a bookmark in and picking up another book — saving and restoring register state to change what is running.
5. Privilege ring — how much the running code is allowed to do — rings 0 to 3 on x86-64, exception levels EL0 to EL3 on ARM64.
6. System call — knocking on the kernel's door on purpose — SYSCALL on x86-64, SVC on ARM64, with the number in a register.
7. Page fault — the address had no memory behind it — a fault that lets the kernel map a page and re-run the instruction.

10.13 Reading a real spec sheet

10.13.1 PLAIN - in simple words

1. A CPU page is a wall of numbers. Almost all of them are simple once you know what each one is measuring.
2. Cores means how many complete CPUs are on the chip. Threads means how many programs the operating system can schedule at once, which may be the same number or twice it.
3. Base clock is what the chip guarantees under a heavy load on all cores. Boost clock is the best it will ever do, briefly, on one core, when cool.
4. Cache numbers tell you how much fast memory sits between the cores and main memory, at each of three levels.
5. TDP is a cooling target in watts. It is not the power the chip draws.
6. Process node, such as "3 nm", is a marketing name for a manufacturing generation, not a measurement of anything on the chip.
7. Socket says which motherboard it physically fits.
8. Memory channels and PCIe lanes say how much data can get in and out.

10.13.2 PLAIN - a picture in your head

1. Compare it to reading a car's specification.
2. Cores are cylinders. Threads are how many drivers the dashboard pretends there are.
3. Base clock is the speed you can hold all day up a hill. Boost clock is what the speedometer reaches downhill with a tailwind, for thirty seconds.
4. Cache is the fuel already in the lines. TDP is the radiator's rating.
5. PCIe lanes are the number of doors for luggage.

Where this comparison breaks: a car's top speed is a property of the car, but a CPU's boost clock is a property of the car, the road, the weather and the driver together. Cooling, motherboard power delivery and even the individual silicon sample all change it.

10.13.3 PLAIN - a worked example

1. Take a current desktop chip: the AMD Ryzen 9 9950X3D, launched on 12 March 2025. Every number decoded.

Item	Value	What it means
Cores / threads	16 / 32	16 real cores, SMT on
Base clock	4.3 GHz	Guaranteed all-core floor
Boost clock	5.7 GHz	Best case, one core, cool
L1 cache	80 KB per core	32 KB code, 48 KB data
L2 cache	1 MB per core	16 MB in total
L3 cache	128 MB	96 MB stacked, 32 MB plain
TDP	170 W	Cooling design target

Item	Value	What it means
Socket	AM5	LGA 1718 mainboard fit
Memory	Dual channel DDR5	Two 64-bit paths
PCIe	5.0, 24 usable lanes	Graphics and storage

2. The 128 MB of L3 is not one pool. This chip has two eight-core dies. One has a 64 MB cache die stacked on top of it, giving that die 96 MB. The other has the normal 32 MB.
3. So threads on the wrong die do not get the big cache. That is why the operating system scheduler has to be told which die is which.
4. Now the phone side: the Qualcomm Snapdragon 8 Elite Gen 5, announced on 24 September 2025.

Item	Value	What it means
Prime cores	2 at up to 4.6 GHz	Third-generation Oryon
Perf cores	6 at up to 3.63 GHz	Same design, lower clock
Threads	8	No SMT, one per core
Process	3 nm class	TSMC generation name
GPU	Adreno at 1.2 GHz	Sliced architecture
NPU	Hexagon	Qualcomm claims +37%
Memory	LPDDR5X	On-package, no slots
Modem	Snapdragon X85	Up to 12.5 Gbps down

5. Notice what is missing from the phone sheet: no TDP, no socket, no memory channels, no PCIe lanes. A phone chip is soldered, its memory is stacked on the package, and its power budget is a thermal envelope of a few watts, not a published number.

10.13.4 PLAIN - what is really happening inside

1. TDP deserves a paragraph of its own, because it is widely misread.
2. TDP is the heat, in watts, that the cooler must be able to remove for the chip to hold its rated behaviour. It is a design target for the cooling system.
3. AMD's socket AM5 parts will draw up to 1.35 times the TDP in package power. So a 170 W part draws up to about 230 W.
4. Intel now publishes two numbers instead: base power and maximum turbo power. On the Core Ultra 9 285K those are 125 W and 250 W.
5. Boost clock is a ceiling, not a promise. It requires a light load, a good core, headroom in the power budget, and cool silicon. All-core sustained clocks under a heavy load are typically several hundred megahertz lower.
6. Cores are no longer identical. Intel's desktop parts mix performance cores and efficient cores with different clocks and different capabilities. Phone chips have done this since Arm's big.LITTLE arrangement in 2011.
7. "Cores plus threads" on a mixed chip is genuinely confusing. The Core Ultra 9 285K has 8 performance cores plus 16 efficient cores, which is 24 cores, and because SMT was removed it has exactly 24 threads.

10.13.5 TECHNICAL - the engineer's version

1. The two current desktop parts side by side, from vendor specifications.

Spec	9950X3D	Core Ultra 9 285K
Launched	March 2025	Q4 2024
Cores	16 uniform	8 P plus 16 E

Spec	9950X3D	Core Ultra 9 285K
Threads	32	24
Base clock	4.3 GHz	3.7 GHz P, 3.2 GHz E
Max turbo	5.7 GHz	5.7 GHz
L2 total	16 MB	40 MB
L3	128 MB	36 MB
Power	170 W TDP	125 W base, 250 W max
Socket	AM5	FCLGA1851
Memory	DDR5, 2 channels	DDR5-6400, 2 channels
PCIe	5.0, 24 usable	5.0 and 4.0, 24 lanes

2. Process nodes: the 9950X3D uses TSMC N4P for its core dies with a separate 6 nm class I/O die; the 285K uses a TSMC N3B compute tile in a multi-tile package. In both cases the number is a process generation name. Chapter 3 covered why no dimension on the chip actually measures 3 or 4 nanometres.
3. The mobile parts for comparison, from vendor material.

Spec	Snapdragon 8E Gen 5	Apple A19 Pro
Announced	24 Sept 2025	9 Sept 2025
CPU	2 at 4.6, 6 at 3.63	2 at 4.26, 4 at 2.6
Process	3 nm class	TSMC N3P
Memory	LPDDR5X on package	12 GB LPDDR5X
Bandwidth	not published	about 76.8 GB/s
NPU	Hexagon	16-core Neural Engine

4. Note that Apple publishes core counts and little else; Qualcomm publishes clocks; neither publishes sustained power. Comparing them properly needs measurement, not spec sheets.
5. Memory bandwidth is worth computing rather than reading. Dual-channel DDR5-6000 gives 2 channels times 8 bytes times 6000 million transfers per second, which is 96 GB/s peak. Real achieved bandwidth is typically 70 to 85 percent of that.
6. PCIe bandwidth likewise: PCIe 5.0 runs at 32 gigatransfers per second per lane, and with 128b/130b encoding that is about 3.94 GB/s per lane per direction, so a 16-lane graphics slot is about 63 GB/s each way.
7. Tools: `lscpu`, `lscpu -C`, `dmidecode -t processor` for socket and package data, `sudo lspci -vv` for actual negotiated PCIe link width and speed, and `turbostat` for live clock, power and temperature.

10.13.6 WORDS - remember these

1. Base clock — the speed it can always hold — guaranteed frequency at rated power with all cores loaded.
2. Boost clock — the best it ever does — opportunistic maximum for a light load with thermal and power headroom.
3. TDP — how much heat the cooler must remove — thermal design power, a cooling target, not consumption.
4. PPT, PL1, PL2 — the real power limits — package power tracking on AMD, base and turbo power limits on Intel.
5. Socket — the physical fit — LGA 1718 for AM5, FCLGA1851 for Arrow Lake.
6. Memory channel — one independent 64-bit path to DRAM — two channels is standard on desktop, more on workstation and server.
7. PCIe lane — one serial link pair for expansion — PCIe 5.0 gives about 3.94 GB/s per lane per direction.
8. System on chip — everything in one package — CPU, GPU, NPU, modem, memory controller and I/O on one piece of silicon.

10.14 How fast is a CPU really

10.14.1 PLAIN - in simple words

1. Speed is not gigahertz. Gigahertz is how many times per second the clock ticks, and that is only half the story.
2. The other half is how much work the chip gets done per tick.
3. A chip at 3 GHz that does two instructions per tick beats a chip at 4 GHz that does one.
4. Work per tick is called **instructions per cycle**, or IPC.
5. Time taken equals the number of instructions, times cycles per instruction, divided by the clock speed. Those three numbers are the whole of it.
6. A compiler that emits fewer instructions helps. A wider chip that runs more per cycle helps. A higher clock helps. All three multiply together.
7. So you cannot compare two different chips by clock speed alone, and you certainly cannot compare two different instruction sets that way.

10.14.2 PLAIN - a picture in your head

1. Think of a factory line. The clock is how often the conveyor advances.
2. IPC is how many items each advance actually delivers.
3. A line that advances four times a second but usually delivers nothing is worse than one that advances three times a second and always delivers two.
4. Turbo is the line running fast for a short burst while everything is still cool, then slowing down once the machines heat up.
5. Thermal throttling is the safety cut-out that slows the line before anything melts.

Where this comparison breaks: a factory delivers nothing when starved of parts, and so does a CPU, but the starvation is invisible. A core waiting on memory still ticks its clock and still reports full frequency. It just retires almost no instructions. That is why frequency graphs look healthy while programs run slowly.

10.14.3 PLAIN - a worked example

1. Take a program of exactly 1,000,000,000 instructions.
2. CPU A: 4.0 GHz, IPC 1.0. Time = 1e9 / (4e9 x 1.0) = 0.250 seconds.
3. CPU B: 3.0 GHz, IPC 1.8. Time = 1e9 / (3e9 x 1.8) = 0.185 seconds.
4. CPU B is 26 percent faster while being 25 percent slower in gigahertz.
5. This is not a made-up case. It is exactly what happened in 2006. The Pentium 4 670 ran at 3.8 GHz. The Core 2 Duo E6600 ran at 2.4 GHz and beat it comfortably per core, because its IPC was far higher.
6. Now add the third term. If a better compiler removes 20 percent of the instructions, CPU A takes 0.200 seconds without any hardware change.
7. Real IPC values worth carrying around: pointer-chasing code that misses cache constantly runs at 0.1 to 0.3; ordinary integer application code runs at roughly 1 to 2.5 on a modern wide core; tight vectorized numeric loops can exceed 4.

10.14.4 PLAIN - what is really happening inside

1. Turbo works like a budget. The chip watches its power draw, its current draw and its temperature, thousands of times a second.
2. While all three are under their limits, it raises the clock. When any one hits its limit, it lowers it.
3. So the same chip in a small case with a weak cooler is genuinely a slower chip than in a big case with a good one. The number on the box does not change; the achieved clock does.
4. Not all cores are equal even within one chip. Manufacturing variation means some cores tolerate higher clocks, and the firmware knows which, and sends single-threaded work there.
5. Thermal throttling is the last-resort mechanism. If temperature reaches the limit, typically around 95 to 100 degrees Celsius, the clock drops sharply until it recovers.

6. This is why a benchmark run for 10 seconds and a benchmark run for 10 minutes give different answers, and why laptop reviews measure both.

10.14.5 TECHNICAL - the engineer's version

1. The performance equation, often called the iron law of processor performance, is: CPU time = instruction count x cycles per instruction x clock period.
2. IPC is the reciprocal of CPI. Each of the three terms is owned by a different party: the compiler and program own instruction count, the microarchitecture owns CPI, and the process and power budget own the clock.
3. Measuring it, on Linux.

```
perf stat -e cycles,instructions,branch-misses ./program
# prints, among other lines:
#   1,842,113,904  cycles
#   3,120,884,551  instructions   #  1.69  insn per cycle
```

4. That "insn per cycle" figure counts retired instructions, meaning the ones that actually completed. Speculative work that was thrown away is not counted, which is the correct definition.
5. IPC is not comparable across instruction sets. One x86-64 instruction may do a load, an arithmetic operation and an addressing calculation, while the ARM64 equivalent needs two or three instructions. Comparing IPC between them measures encoding density, not speed.
6. IPC is also not comparable across programs. A memory-bound program has low IPC because it is waiting, and no amount of core width fixes it.
7. Benchmark categories, and what each is honest about.
    1. SPEC CPU 2017: real compiled applications, with separate integer and floating point suites and separate rate and speed variants. Rate measures throughput with many copies; speed measures one job's time. It is the closest thing to a standard, and results must be submitted with full disclosure of compiler flags.
    2. Geekbench 6 and Cinebench 2024: quick, widely quoted, and dominated by short bursts, so they favour high boost clocks over sustained ability.
    3. Microbenchmarks: measure one instruction's latency or throughput. Useful for engineers, useless as a purchase guide.
    4. Your own workload, timed: always the best answer if you can get it.
8. Turbo control is exposed as real, documented limits. Intel defines PL1, the long-term power limit, PL2, the short-term limit, and tau, the time constant over which the average is taken. AMD defines PPT for package power, TDC for sustained current and EDC for peak current, with Precision Boost 2 adjusting clocks continuously against all three plus temperature.
9. Where experts disagree: whether single-number benchmark scores are useful at all. One camp says a composite score is the only practical way for a buyer to compare; the other says composites hide the fact that different workloads rank chips in different orders. Both are right in their own context, and the safe habit is to look at the sub-scores.
10. Tools: `turbostat` on Intel and `zenmonitor` or `ryzen_monitor` on AMD show live per-core clock, package power and temperature. `s-tui` shows throttling under load. `stress-ng` generates the load.

10.14.6 WORDS - remember these

1. IPC — how much gets done per tick — instructions retired per cycle, measured by hardware performance counters.
2. CPI — the same thing upside down — cycles per instruction, the reciprocal of IPC.
3. Iron law — the only performance formula you need — time equals instruction count times CPI times clock period.
4. Retired instruction — one that really counted — an instruction that committed architectural state, excluding discarded speculation.

5. Turbo — a temporary speed rise — opportunistic frequency increase within power, current and thermal limits.
6. Thermal throttling — slowing down to survive — automatic clock reduction when a temperature limit is reached.
7. SPEC CPU 2017 — the serious benchmark — a suite of real applications with mandatory disclosure of build settings.

10.98 Common wrong ideas

1. Wrong: a CPU understands your program. Right: it matches bit patterns to control signals. Nothing is understood at any point.
2. Wrong: a higher clock speed means a faster chip. Right: time equals instruction count times cycles per instruction divided by clock speed, and the middle term differs by more than 2 times between designs.
3. Wrong: instructions and data are stored differently. Right: they are the same bytes in the same memory. Only permission bits and where the program counter points make the difference.
4. Wrong: more cores means proportionally more speed. Right: Amdahl's law caps you at 1 divided by the serial fraction, so 5 percent serial code can never go more than 20 times faster.
5. Wrong: hyper-threading doubles performance. Right: it duplicates architectural state only, and the typical real throughput gain is 10 to 30 percent, sometimes negative.
6. Wrong: RISC chips are simple and CISC chips are complex. Right: both decode into internal micro-operations and both run out of order. The real surviving differences are decode complexity, memory operands and the memory ordering model.
7. Wrong: a deeper pipeline is always faster. Right: the Pentium 4 went from 20 to 31 stages, was cancelled at 3.8 GHz, and was beaten by a 14-stage design in 2006.
8. Wrong: microcode is what runs your program. Right: most instructions never reach the microcode sequencer. It handles complicated cases and provides a patching mechanism.
9. Wrong: cache makes memory faster. Right: cache makes repeated and nearby access faster. Truly random access over a large array gets almost no benefit at all.
10. Wrong: TDP is how much power the chip uses. Right: TDP is a cooling design target. An AMD 170 W part draws up to about 230 W at the package.

10.99 Chapter summary in 20 lines

1. A CPU is one loop repeated forever: fetch, decode, execute, repeat.
2. Its parts are a control unit, an ALU, a register file, a program counter, an instruction register, caches, interconnect and a memory controller.
3. The stored-program idea puts instructions and data in the same memory, which is why one machine runs any program.
4. That same idea permits buffer overflows, self-modifying code and JIT compilation, and it is why the NX permission bit had to be invented.
5. Real CPUs are modified Harvard: one memory underneath, split L1 instruction and data caches on top.
6. An instruction is a number split into fields: opcode, operands, immediates. Addressing modes are the rules that turn those fields into an address.
7. An ISA is a contract, not a design. x86-64 dates from 2000, ARM64 from 2011, RISC-V began at Berkeley in 2010 and is open.
8. Registers are the fastest storage that exists because they are tiny, near, many-ported and addressed by three to five bits.
9. Microcode turns one complex instruction into several micro-operations, and because it is stored rather than wired, it can be patched at boot.

10. That patching mechanism carried the 2018 Spectre mitigations, which added new control registers to a shipped architecture by firmware update.
11. Pipelining overlaps stages so one instruction completes per cycle instead of one every five, and it is why clock speeds could rise at all.
12. Hazards are structural, data and control. Forwarding fixes most data hazards; branch prediction fixes most control hazards.
13. Very deep pipelines lose more on every mispredict than they gain in clock. The 31-stage Pentium 4 Prescott is the proof.
14. Superscalar, out-of-order execution, register renaming and the reorder buffer extract parallelism while still showing software a tidy, ordered machine.
15. Modern TAGE-style branch predictors are right the great majority of the time, and each miss still costs about 16 to 20 cycles.
16. SIMD does one instruction to many numbers at once, from MMX in 1997 to AVX-512 and Arm SVE. Fixed-function units such as AES-NI and NPUs go further and put whole algorithms in silicon.
17. Clocks stopped rising around 2005 when Dennard scaling ended, so chips went multicore, and Amdahl's law then set the ceiling.
18. Cache coherence keeps cores agreeing about one address; memory ordering and barriers govern the visible order across different addresses.
19. Caches work only because programs reuse data and touch neighbours. Sequential access can be 50 to 100 times faster than random access.
20. Judge a CPU by instruction count times cycles per instruction times clock period, measured on your own workload, not by the number on the box.

11

CHAPTER

KB

The Memory Hierarchy — Cache, RAM and Virtual Memory

11.0 What this chapter gives you

1. You will be able to draw the whole memory ladder, from registers down to tape, with real sizes, real waiting times and real prices.
2. You will be able to define temporal and spatial locality and show, with a measured number, why reading an array the wrong way is about nine times slower on the same machine.
3. You will be able to read a memory stick's label, say what DDR5-6000 CL30 means, and work out its true delay in nanoseconds by hand.
4. You will be able to trace a physical address into channel, rank, bank, row and column, and say why memory delay is a spread of values, not one value.
5. You will be able to translate a real 64-bit virtual address into a physical one, step by step, through four levels of page table.
6. You will be able to tell a minor page fault from a major one and from an invalid one, and give the cost of each in nanoseconds.
7. You will be able to read a real process memory map and name every region in it, including where a memory leak would show up.
8. You will be able to make three specific code changes that gave measured speed-ups of 9.0 times, 8.2 times and 3.5 times on the machine used to write this chapter.

11.1 Why a hierarchy exists at all

11.1.1 PLAIN - in simple words

1. A computer needs to remember things. Some things it needs right now, some in a moment, some next year.
2. If we could buy one kind of memory that was huge, instant and cheap, there would be no chapter here. We cannot.
3. Memory that answers in under a nanosecond (a billionth of a second) is very expensive per byte and takes a lot of chip space.
4. So engineers build a ladder. Tiny and instant at the top. Huge and slow at the bottom.
5. Almost everything in this chapter is one idea repeated: guess what will be needed soon, and move it up the ladder before it is asked for.

11.1.2 PLAIN - a picture in your head

1. Think of a student studying at a desk.
2. In your hand is one pen. That is a register. You can use it with no delay.
3. On the desk are three open books. That is cache. Reaching one takes a second.

4. On the shelf behind you are two hundred books. That is main memory. You must stand up and walk over.
5. In the university library are two million books. That is the disk. You must walk across campus and wait at a counter.
6. Nobody carries two million books to their desk. Nobody studies with an empty desk either. You keep the right small set close.

Where this comparison breaks: the student decides what to fetch. In a computer the hardware guesses, automatically, with no help from the program. Also, a book on the shelf is still one book. In a computer the same data can exist in four places at once, and keeping those copies agreeing is a real problem that Chapter 10 dealt with under the name cache coherence.

11.1.3 PLAIN - a worked example

1. Take one real machine: the virtual server used to measure every benchmark in this chapter. It runs an Intel Xeon at 2.1 GHz.
2. One clock tick at 2.1 GHz lasts about 0.48 nanoseconds.
3. We measured how long one random memory read takes, for different amounts of data being walked over. Each read had to finish before the next one started, so no overlapping could hide the delay.

Data being walked	Measured time per read
16 KB (fits in L1)	1.79 ns
256 KB (fits in L2)	5.74 ns
8 MB (fits in L3)	37.65 ns
128 MB (goes to RAM)	143.37 ns
512 MB (goes to RAM)	179.76 ns

4. So the same instruction, `load from memory`, took anywhere from 1.79 to 179.76 nanoseconds. A factor of 100, decided entirely by where the data was. ### 11.1.4 PLAIN - what is really happening inside
5. Two different physical things are being traded off. Cost per bit, and time to answer.
6. Fast memory uses a circuit called SRAM, which needs six transistors to hold one bit. It holds its value as long as power is on, and answers in one or two clock ticks.
7. Main memory uses DRAM, which needs one transistor and one tiny capacitor per bit. Far smaller, far cheaper per bit.
8. But the DRAM capacitor leaks. Every cell must be read and rewritten thousands of times a second, which is called refresh, and reading it is destructive, so it must be written back.
9. Spinning disks and tape store magnetic patterns, and something physical has to move before you get an answer. Movement is measured in milliseconds. ### 11.1.5 TECHNICAL - the engineer's version
10. The full ladder, with figures accurate for a mid-2026 desktop or small server. Prices are retail per gigabyte and are unusually high because of the 2025 to 2026 memory shortage.

Level	Typical size	Latency
Register	32 x 64 bits	0 cycles
L1 data cache	32 to 64 KB per core	4 to 5 cycles
L2 cache	1 to 3 MB per core	14 to 20 cycles
L3 cache	24 to 260 MB shared	40 to 90 cycles
DRAM (DDR5)	16 to 512 GB	70 to 120 ns
NVMe SSD	0.5 to 8 TB	40 to 100 us
Hard disk	4 to 30 TB	5 to 10 ms

Level	Typical size	Latency
LTO-10 tape	30 TB per cartridge	30 to 90 s

2. The same ladder priced. Figures dated August 2026, taken from public price trackers. Treat them as approximate and expect them to move.

Level	Price per GB	Note
SRAM in cache	not sold separately	about 100x DRAM area
DDR5 DIMM	about 16.20 USD	was about 3 USD in 2024
DDR4 DIMM	about 7.64 USD	end-of-life squeeze
Consumer NVMe SSD	about 0.09 USD	1 TB near 90 USD
Nearline hard disk	0.014 to 0.030 USD	up about 50% in 2026
LTO-10 tape media	0.003 to 0.010 USD	drive costs thousands

3. The 2025 to 2026 price situation is worth stating plainly, because it is recent and it distorts every rule of thumb. TrendForce reported DDR5 spot prices rising 307 percent between September and November 2025, driven by AI datacentre demand. A 32 GB DDR5 kit that sold near 95 USD before the shortage was 380 to 589 USD in August 2026.
4. Established fact: the shortage happened and prices roughly doubled to quadrupled. Active forecast, not fact: several analysts expect no meaningful relief before late 2027. Treat that as a prediction.
5. The classic scaling rule, sometimes called the memory wall, is that CPU speed improved far faster than DRAM latency for about thirty years. DRAM bandwidth improved greatly. DRAM latency barely improved at all. Section 11.3 gives the numbers that prove it.
6. Tools that show you the real ladder on your own machine: `lscpu` and `getconf -a` on Linux, `sysctl -a | grep cache` on macOS, and the free Intel Memory Latency Checker for detailed per-level figures.

```
smaller, faster, dearer per byte
      ^
      |   registers      < 1 ns      bytes
      |   L1 cache       ~ 1 ns      tens of KB
      |   L2 cache       ~ 5 ns      a few MB
      |   L3 cache       ~ 40 ns     tens of MB
      |   DRAM           ~ 90 ns     tens of GB
      |   NVMe SSD       ~ 60 us     terabytes
      |   hard disk      ~ 8 ms      tens of TB
      v   tape           ~ 60 s      petabytes
bigger, slower, cheaper per byte
```

11.1.6 WORDS - remember these

1. Memory hierarchy — the ladder of stores — a multi-level storage system trading capacity against access latency.
2. Latency — the waiting time before data arrives — time from request issue to first data return, measured in cycles or nanoseconds.
3. SRAM — fast memory that needs power — static random access memory, typically a six-transistor cell, no refresh needed.
4. DRAM — cheap main memory — dynamic random access memory, one transistor and one capacitor per cell, requires periodic refresh.
5. Memory wall — CPUs got fast, memory did not — the growing gap between processor cycle time and DRAM access latency.

11.2 Locality: why the whole idea works

11.2.1 PLAIN - in simple words

1. The ladder only helps if we can guess what a program will want next.
2. Luckily, programs are extremely predictable in two specific ways.
3. First: if a program used something a moment ago, it will very likely use it again soon. That is called **temporal locality**, meaning locality in time.
4. Second: if a program used something, it will very likely use the thing sitting right next to it. That is **spatial locality**, meaning locality in space.
5. Because of these two habits, a small fast store holding recently used data and its neighbours will satisfy most requests.
6. Locality is not a law. It is a very strong statistical habit of real code. Code that breaks it runs slowly, and the machine cannot help you.

11.2.2 PLAIN - a picture in your head

1. Picture a cook in a kitchen with a small counter next to the stove.
2. Temporal locality: the salt gets used every two minutes, so it stays on the counter all evening.
3. Spatial locality: when the cook fetches the tin of tomatoes from the store cupboard, they carry the whole tray it sits on, because the other tins on that tray will probably be needed too.
4. Carrying the whole tray is exactly what a computer does. It never fetches one byte from memory. It fetches a fixed-size block, normally 64 bytes, called a cache line.
5. If the cook only ever needs one tin from each of fifty different trays, carrying trays is pure waste, and the evening goes badly.

Where this comparison breaks: the cook knows the recipe in advance. The hardware does not know your program. It uses fixed rules, mainly "keep what was just used" and "fetch the neighbours". Modern chips also add a prefetcher that spots simple patterns such as "every 64 bytes forward", but it is a pattern detector, not a mind reader.

11.2.3 PLAIN - a worked example

1. Here is the demonstration everybody should do once. A square grid of numbers, added up two different ways.
2. In C, a two-dimensional array is stored row by row. The whole first row sits in memory, then the whole second row, and so on. That is called row-major order and it is part of the C language definition.
3. Both read exactly the same 16,777,216 numbers and produce the same answer.

```
double a[4096][4096];          /* 128 MB of doubles */

/* version 1: along the rows, memory order */
for (i = 0; i < 4096; i++)
    for (j = 0; j < 4096; j++)
        s1 += a[i][j];

/* version 2: down the columns, jumping 32 KB each step */
for (j = 0; j < 4096; j++)
    for (i = 0; i < 4096; i++)
        s2 += a[i][j];
```

4. Measured on the 2.1 GHz Xeon used for this chapter, compiled with `gcc -O2`, three runs each.

Order	Time	Ratio
Along rows	0.0182 to 0.0209 s	1.0x

Order	Time	Ratio
Down columns	0.1837 to 0.1889 s	9.0x to 10.4x

5. So the column version was about nine to ten times slower, on the same data, in the same program, with the same compiler flags.
6. Why. Each row is 4096 doubles, which is 32,768 bytes. Stepping down a column moves 32,768 bytes at a time.
7. A cache line is 64 bytes and holds 8 doubles. Walking a row uses all 8 of them before moving on, so one memory fetch serves 8 additions.
8. Walking a column uses 1 of the 8 and throws the rest away, so it needs 8 times as many fetches. It also touches a new 4 KB page every single step, which wrecks address translation as well. Section 11.6 explains that part. ### 11.2.4 PLAIN - what is really happening inside
9. When the CPU asks for an address, the cache checks whether the 64-byte block containing it is already held.
10. If it is there, that is a hit. If it is not, that is a miss, and the whole 64-byte block is fetched from the level below, and something already in the cache is thrown out to make room.
11. Row order: byte 0 misses, and the fetch brings bytes 0 to 63. The next seven reads all hit. One miss buys eight uses.
12. Column order: byte 0 misses and brings bytes 0 to 63, but the program's next read is 32,768 bytes away, so the other 63 bytes are never touched before being evicted. Every read is a miss.
13. On top of that, the hardware prefetcher watches the address stream. A forward walk with a small constant step is exactly the pattern it detects, so it starts fetching the next lines before they are asked for. Row order gets that free help. Column order, jumping 32 KB, usually does not. ### 11.2.5 TECHNICAL - the engineer's version
14. Denning formalized locality as the working set model in 1968, defining the working set $W(t, T)$ as the set of pages referenced in the last T units of virtual time. It remains the standard framework for replacement policy.
15. Effective access time with a two-level model: `T_eff = h * T_fast + (1 - h) * T_slow`, where h is the hit rate.
16. Put real figures in. With L1 at 1.79 ns, DRAM at 143 ns and a 95 percent hit rate: `0.95 * 1.79 + 0.05 * 143 = 8.85 ns`. At 99 percent it is 3.20 ns. At 99.9 percent it is 1.93 ns. The tail dominates, which is why hit rate is quoted to two decimal places.
17. Cache line size is 64 bytes on essentially all x86-64 and on Apple silicon the L1 line is 64 bytes with a 128-byte L2 line. Verify with `getconf LEVEL1_DCACHE_LINESIZE` on Linux or `sysctl hw.cachelinesize` on macOS.
18. Row-major storage for multidimensional arrays is a **standard**, fixed by the C and C++ language definitions. Column-major is the standard in Fortran, MATLAB, R and Julia. NumPy defaults to row-major but supports both through the `order` argument, so the same loop can be fast or slow depending on how the array was created.
19. The measured 9.0x to 10.4x ratio above is specific to this machine, this array size and this compiler. On a machine with a larger L3 and a smaller array the ratio shrinks toward 1. On a machine with slower DRAM it grows. Do not quote a single universal number.
20. Measure it rather than guess. On Linux:

```
perf stat -e cache-references,cache-misses,\
LLC-load-misses,dTLB-load-misses ./program
```

On macOS, use Instruments with the Counters template, or `xcrun xctrace`.

11.2.6 WORDS - remember these

1. Temporal locality — used once, used again soon — the probability that a referenced address is referenced again within a short interval.
2. Spatial locality — neighbours get used together — the probability that addresses near a referenced address are referenced soon.
3. Cache line — the block that always moves together — the minimum unit of transfer between cache levels, 64 bytes on x86-64 and Arm.
4. Working set — what a program is using just now — the set of pages referenced in a sliding time window, from Denning 1968.
5. Prefetcher — the hardware guesser — a unit that detects address stride patterns and issues loads before the demand request.

11.3 RAM modules in the real world

11.3.1 PLAIN - in simple words

1. Main memory comes on a green stick you push into a slot on the motherboard. The stick is called a **DIMM**, short for dual in-line memory module.
2. Laptops use a shorter version called a **SO-DIMM**, where SO means small outline.
3. A group of chips that answers as one unit is called a **rank**. A stick can have one rank or two, and a two-rank stick has chips on both sides, or stacked.
4. The motherboard has a small number of paths to memory, called **channels**. Two channels means two separate roads, not a wider road.
5. Filling both channels roughly doubles how much data per second you can move. It does not shorten the wait for a single item.
6. A label like DDR5-6000 CL30 tells you two things: how many transfers per second, and how many clock ticks you wait for the first one.

11.3.2 PLAIN - a picture in your head

1. Think of a warehouse with numbered aisles, shelves and boxes.
2. A channel is a whole loading dock with its own truck. Two channels means two docks working at the same time.
3. A rank is one full team of workers. Two ranks means two teams that share the dock and take turns, so while one team is putting a pallet away, the other can be fetching.
4. A bank is an aisle. Only one shelf per aisle can be pulled out at a time, but different aisles can be busy at once.
5. A row is one shelf. When a worker pulls a shelf out, it sits on the trolley, and taking more boxes off the same shelf is quick.
6. Adding a second dock doubles throughput. It does not make one single box arrive any sooner. That is the difference between bandwidth and latency, and it is the single most misunderstood thing about RAM.

Where this comparison breaks: shelves in a real warehouse do not forget their contents. DRAM rows do. Reading a row destroys it and it must be written back, and every row must be refreshed every 32 or 64 milliseconds or the data fades away. There is no warehouse equivalent of that.

11.3.3 PLAIN - a worked example

1. Take a common 2026 desktop kit: 2 sticks of 16 GB DDR5-6000, timings written as 30-36-36-76.
2. Those four numbers are, in order, CL, tRCD, tRP and tRAS, all counted in clock ticks.
3. DDR5-6000 means 6000 million transfers per second. DDR means double data rate: data moves on both the rise and the fall of the clock.

4. So the clock itself runs at 3000 MHz, which is 3000 million ticks a second.
5. CL30 means 30 ticks of waiting. 30 x 0.3333 = 10.0 nanoseconds.
6. The general formula, worth memorizing:

```
true latency in ns = CL * 2000 / data rate in MT/s
```

7. Now the shock. Apply the same formula across twenty-five years.

Module and timing	CAS in ns
PC133 SDRAM, CL3	22.5
DDR-400, CL3	15.0
DDR2-800, CL5	12.5
DDR3-1600, CL9	11.25
DDR4-3200, CL16	10.0
DDR5-6000, CL30	10.0
DDR5-8000, CL38	9.5

8. Transfer rate went up about 60 times. The wait for the first byte went from 22.5 ns to about 10 ns, roughly a factor of two, in twenty-five years.
9. Bandwidth for the kit: 6000 MT/s x 8 bytes per transfer = 48 GB/s per stick. Two sticks in two channels = 96 GB/s.

11.3.4 PLAIN - what is really happening inside

1. A DDR5 stick is 64 data wires wide, but DDR5 splits them into two independent halves of 32 wires each, called sub-channels. DDR4 did not do this.
2. A burst moves 16 transfers in a row on DDR5. On a 32-wire sub-channel that is 16 x 4 bytes = 64 bytes, which is exactly one cache line. The design was chosen to match.
3. Ranks matter because a rank cannot do two things at once. With two ranks the controller can be closing a row on one while reading from the other, which raises useful throughput by roughly 5 to 15 percent in practice.
4. **XMP** and **EXPO** are little profiles stored in a small chip on the stick. They tell the motherboard "you may run me at 6000 with these timings", which is faster than the guaranteed baseline the standard defines.
5. **ECC** memory adds extra chips holding check bits. If one bit flips, from a cosmic ray or a marginal cell, the controller repairs it and carries on and logs it. If two bits flip, it detects the error and halts rather than returning wrong data.

11.3.5 TECHNICAL - the engineer's version

1. DDR generations, with JEDEC standard numbers and publication years.

Generation	Standard, year	Rates and voltage
DDR	JESD79, 2000	200-400 MT/s, 2.5 V
DDR2	JESD79-2, 2003	400-1066 MT/s, 1.8 V
DDR3	JESD79-3, 2007	800-2133 MT/s, 1.5 V
DDR4	JESD79-4, 2012	1600-3200 MT/s, 1.2 V
DDR5	JESD79-5, 2020	3200-6400 MT/s, 1.1 V
DDR5 rev C	JESD79-5C, 2024	up to 8800 MT/s

2. JESD79-5C was published on 17 April 2024. Besides extending timings to 8800 MT/s it added PRAC, per-row activation counting, which counts activations at wordline granularity so the system can react to rowhammer-style attacks. It also deprecated PASR, partial array self refresh, on security grounds.
3. DDR5 structural changes against DDR4: two 32-bit sub-channels instead of one 64-bit channel, 32 banks in 8 bank groups instead of 16 banks in 4 groups, 16n prefetch and burst length 16 instead of 8n and BL8, on-die ECC as standard, and the voltage regulator moved onto the module as a PMIC fed from 5 V.
4. On-die ECC is not the same as module ECC. On-die ECC protects the internal DRAM array only. It does not protect the bus between module and CPU. A non-ECC DDR5 stick has on-die ECC and is still not an ECC module.
5. Module ECC is SECDED: single error correct, double error detect. It widens the module from 64 to 72 data bits, 8 check bits per 64 data bits. Linux reports corrected and uncorrected counts through EDAC, readable under `/sys/devices/system/edac/mc/` or with `edac-util -v`.
6. XMP is an Intel **convention**, not a JEDEC standard. XMP 3.0 arrived with DDR5 in 2021 and provides five profile slots, three vendor-written and two user-writable. EXPO, AMD Extended Profiles for Overclocking, was announced in 2022 alongside Socket AM5 and Ryzen 7000. Both write into the module's SPD, serial presence detect, chip. Running either is technically operating the part outside its JEDEC-guaranteed bin.
7. LPDDR is a separate family for phones and thin laptops, physically soldered rather than socketed, with lower voltage and aggressive power-down states. JEDEC published the first LPDDR6 standard on 10 July 2025, specifying 10,667 to 14,400 MT/s and a new arrangement of four 24-bit sub-channels, in place of DDR5's two 32-bit sub-channels, for lower latency and more concurrency.
8. HBM, high bandwidth memory, stacks DRAM dies vertically and connects them with through-silicon vias to a very wide, short bus. JEDEC released JESD270-4, the HBM4 standard, on 16 April 2025: a 2048-bit interface, up to 8 Gb/s per pin, up to 2 TB/s per stack, 32 channels with 2 pseudo-channels each, 4-high to 16-high stacks, and up to 64 GB per stack using 32 Gb dies.
9. CXL, Compute Express Link, adds a third option: memory on the far side of a PCIe-style link. CXL 1.0 and 2.0 built on PCIe 5.0 in 2019 and 2020, CXL 3.0 on PCIe 6.0 in 2022, with 3.1 in 2023 and 3.2 in 2024. Type 3 devices are memory expanders. They add capacity and cost roughly 150 to 250 ns extra latency, so they sit between DRAM and SSD on the ladder.
10. Inspect real modules: `sudo dmidecode -t memory` on Linux, `sudo decode-dimms` for raw SPD contents, and `system_profiler SPMemoryDataType` on macOS.

11.3.6 WORDS - remember these

1. DIMM — the memory stick — dual in-line memory module, a 64-bit-wide printed circuit board carrying DRAM devices.
2. Rank — one full team of chips — a set of DRAM devices sharing a chip-select and responding together to form the full data width.
3. Channel — one road to memory — an independent memory controller port with its own command and data bus.
4. CAS latency — ticks of waiting for the first byte — column address strobe delay in clock cycles, from column command to first data.
5. ECC — memory that repairs itself — error correcting code memory, normally SECDED using 8 check bits per 64 data bits.
6. HBM — memory stacked on the GPU package — high bandwidth memory, a 1024 or 2048-bit stacked DRAM connected by through-silicon vias.

11.4 The memory controller

11.4.1 PLAIN - in simple words

1. Something has to turn "read address 4,297,318,416" into the right electrical commands on the right wires. That thing is the **memory controller**.
2. Today it is inside the processor itself, on the same piece of silicon. That change alone removed a large chunk of memory delay.
3. The controller takes an address and splits it into pieces: which channel, which rank, which bank, which row, which column.
4. It also keeps a queue of pending requests and is allowed to serve them out of order, choosing whichever one it can answer fastest.
5. So memory latency is not a number. It is a spread of numbers.

11.4.2 PLAIN - a picture in your head

1. Back to the warehouse. A worker has one shelf pulled out onto the trolley at a time, per aisle.
2. If your box is on the shelf already out, you get it fast. That is a **row buffer hit**.
3. If the wrong shelf is out, they must push it back in before pulling yours. Slowest. That is a **row buffer conflict**.
4. The supervisor sees twenty orders at once and deliberately serves all the orders for the shelf already out, before switching. That is scheduling.
5. This is why the same order can take one, two or three times as long, depending purely on what came before it.

Where this comparison breaks: the supervisor cannot delay an order forever. Real controllers have fairness and starvation limits, and they must interrupt everything periodically for refresh, which has no warehouse equivalent. ### 11.4.3 PLAIN - a worked example

1. Take DDR5-6000 with timings 30-36-36-76. Convert to nanoseconds first.
2. CL 30 ticks = 10.0 ns. tRCD 36 ticks = 12.0 ns. tRP 36 ticks = 12.0 ns.
3. Now three cases for one read, at the DRAM chip itself.

Case	Sum of timings	Time
Row already open	CL	10.0 ns
No row open	tRCD + CL	22.0 ns
Wrong row open	tRP + tRCD + CL	34.0 ns

4. So the chip alone can answer in 10 ns or 34 ns for identical instructions.
5. Which is why the measured figure in section 11.1 was 143 ns and not 10 ns. Most of the delay is not in the DRAM chip at all. ### 11.4.4 PLAIN - what is really happening inside
6. The controller receives a physical address, for example 0x0001_62B8_D010, and chops the bits into fields. A rough example layout for a two-channel DDR5 system, from the top of the address downward: row bits, then bank group and bank bits, then rank, then channel, then column, then the byte offset inside the burst.
7. The exact chopping is an **implementation detail**, different on every CPU family, and often deliberately scrambled by XOR-ing bits together.
8. Why scrambled: if consecutive cache lines all landed in the same bank, a simple sequential walk would hit one bank over and over and get no parallelism. Interleaving spreads consecutive lines across channels and banks so they can be served at once.
9. Having chosen a bank, the controller issues ACTIVATE with a row address. That copies the whole row, typically 8 or 16 kilobits, into the sense amplifiers, which form the row buffer.

10. If it needs a different row in that bank, it must first issue PRECHARGE, which writes the buffer back into the cells and clears the sense amps.
11. The scheduler picks among queued requests. The classic policy is FR-FCFS, first-ready first-come-first-served: prefer any request that hits an open row, otherwise take the oldest.

11.4.5 TECHNICAL - the engineer's version

1. History. AMD integrated the memory controller onto the CPU die with the Athlon 64, launched September 2003. Intel followed with Nehalem in November 2008, retiring the front-side bus. Before that, every memory access crossed a separate northbridge chip.
2. Consequence: memory latency became a property of the CPU, and multi-socket machines became NUMA, non-uniform memory access, where local DRAM is faster than DRAM attached to another socket. Typical remote penalty is 1.4x to 2.2x on latency. Inspect with `numactl --hardware` and `lstopo`.
3. Key DDR timing parameters, and what each one blocks.

Symbol	Meaning	DDR5-6000 30-36-36-76
tCL	column to data	10.0 ns
tRCD	activate to column	12.0 ns
tRP	precharge to activate	12.0 ns
tRAS	activate to precharge	25.3 ns
tREFI	refresh interval	3.9 us typical
tRFC	refresh duration	195 to 410 ns

4. Row buffer hit rate on real workloads is commonly 20 to 60 percent for general code, and much higher for streaming loops. It is a first-order term in memory performance, and it is why address interleaving schemes are tuned so carefully.
5. Latency is a distribution. On a loaded server, measured DRAM read latency commonly runs 75 ns at the median and 300 ns or worse at the 99th percentile, because queueing delay grows sharply as utilization approaches the bandwidth limit. This follows ordinary queueing theory: as utilization goes to 1, queue length goes to infinity.
6. Tools: Intel Memory Latency Checker prints a full loaded-latency curve. `perf stat -e uncore_imc/cas_count_read/` gives DRAM traffic on Intel. `pcm-memory` from Intel PCM shows per-channel bandwidth live. On AMD, use `amd_uprof`.
7. Rowhammer is the security consequence of all this. Repeatedly activating one row disturbs charge in neighbours, and since the 2014 Kim et al. paper it has been a real attack. Mitigations include target row refresh, and from JESD79-5C in 2024, PRAC.

11.4.6 WORDS - remember these

1. Memory controller — the part that talks to the sticks — the integrated circuit block that translates addresses into DRAM commands and schedules them.
2. Row buffer — the shelf currently pulled out — the sense amplifier array holding one open DRAM row, typically 1 to 2 KB per device.
3. Row hit — asking for something already open — a column access to the currently activated row, costing only tCL.
4. Bank conflict — the wrong row is open — an access requiring precharge then activate before the column read, costing tRP + tRCD + tCL.
5. NUMA — some memory is further away — non-uniform memory access, where latency depends on which socket owns the DRAM.

11.5 The address space

11.5.1 PLAIN - in simple words

1. An address is just a number that names a location. Nothing more.
2. If addresses are 64 bits, there are about 18.4 quintillion of them.
3. Every program believes it has the whole range to itself, starting near zero, with nobody else in it. That belief is a carefully maintained lie.
4. The addresses a program uses are **virtual addresses**. They are private invented numbers.
5. The addresses on the actual wires to the memory sticks are **physical addresses**. There is exactly one set of those in the machine.
6. This is why one program cannot read another's data by guessing. Its numbers simply do not refer to the same places.

11.5.2 PLAIN - a picture in your head

1. Think of a large office building where every company has its own internal room numbering: Room 1, Room 2, Room 3.
2. Three companies all have a Room 1. They are three different physical rooms.
3. At every door there is a translator holding a small book: "for this company, Room 1 means B4-102".
4. If a company asks for Room 9 and the book has no entry, the translator refuses to open anything and reports it.
5. Two companies can share a meeting room by both having book entries pointing at the same physical room. That is shared memory.

Where this comparison breaks: the translation happens billions of times a second, in hardware, in about a nanosecond when cached. And the book is not one book but a tree of books, which is section 11.6. ### 11.5.3 PLAIN - a worked example

1. Run two copies of the same program at the same time. Print the address of a variable in each.
2. Either way, neither number is a real place in the memory sticks.
3. Here are the real virtual addresses printed by a small test program on the Linux machine used for this chapter.

```
big heap block  at 0x7f19b4bff010
small heap block at 0x55a23da282a0
```

4. The big block sits high, around 0x7f19..., because a large allocation is served by a fresh mapping in the middle of the address space.
5. The small block sits lower, around 0x55a2..., in the classic heap area just above the program's own code. ### 11.5.4 PLAIN - what is really happening inside
6. Nothing in the CPU has 64 real address wires going to memory. That would be both useless and expensive.
7. Current x86-64 processors implement 48 bits of virtual address, extended to 57 bits on newer server parts. The rest of the bits are required to be copies of the top implemented bit.
8. An address obeying that rule is called **canonical**. A non-canonical address causes a fault immediately, before any translation is attempted.
9. So the usable space splits into two lumps: one at the very bottom starting at zero, and one at the very top ending at all ones. The vast middle is a hole.
10. The machine used for this chapter reports 52 bits of physical address and 57 bits of virtual address, which you can read with `lscpu`. ### 11.5.5 TECHNICAL - the engineer's version
11. x86-64 canonical form is a **standard**, defined by the architecture. With 48-bit addressing, bits 48 to 63 must all equal bit 47. With LA57 enabled, bits 57 to 63 must all equal bit 56.

12. Address space sizes:

Mode	Virtual bits	Total space
x86-64 classic	48	256 TiB
x86-64 with LA57	57	128 PiB
Arm64 typical	48	256 TiB
Arm64 with LVA	52	4 PiB

3. Intel published the 5-level paging specification in 2016, submitting Linux patches on 8 December 2016. Support landed in Linux 4.14 and was enabled by default in Linux 5.5. Hardware support arrived with Ice Lake server parts, and AMD supports it on EPYC 9004 and 8004 and Threadripper PRO 7000 WX. It is switched on by bit 12 of CR4, named LA57.
4. Standard Linux x86-64 split with 4-level paging:

Region	Range
User space	0 to 0x00007fff_ffffffff
Non-canonical hole	0x0000800.. to 0xffff7ff..
Kernel space	0xffff8000_00000000 up

5. That gives 128 TiB to user space and 128 TiB to the kernel. With LA57 both become 64 PiB. A process only gets the larger space if it explicitly asks, by passing an mmap hint above 47 bits, so that old programs storing tag bits in the top of pointers do not break. That compatibility hack is a real and current design decision, not history.
6. Observe the layout of any Linux process with `cat /proc/PID/maps`. On macOS use `vmmap PID`. On Windows use VMMap from Sysinternals.

The honest version: "programs never see physical addresses" is a simplification. The kernel routinely works with them, drivers must hand real physical addresses to devices doing direct memory access, and many small embedded microcontrollers have no MMU at all, so every address is physical. The accurate statement is that user-space code on a general-purpose operating system with an MMU sees only virtual addresses.

11.5.6 WORDS - remember these

1. Address — a number naming a location — an index into an address space, 64 bits wide on modern general-purpose CPUs.
2. Virtual address — the private number a program uses — the address produced by the program, subject to MMU translation.
3. Physical address — the real number on the wires — the address presented to the memory controller after translation.
4. Canonical address — an address with a legal top half — one whose unused high bits are sign-extended copies of the highest implemented bit.
5. LA57 — the switch for 57-bit addressing — CR4 bit 12, enabling 5-level paging and a 128 PiB virtual space.

11.6 Virtual memory, explained slowly

11.6.1 PLAIN - in simple words

1. The machine cannot keep a note for every single byte saying where it really lives. That table would be bigger than the memory itself.
2. So memory is cut into fixed-size blocks called **pages**. The usual size is 4 kilobytes, which is 4096 bytes.
3. Now the note only has to say "virtual page 17 lives in physical frame 90210". One entry covers 4096 bytes.
4. The collection of those notes is the **page table**, and every process has its own.
5. The hardware unit that does the lookup is the **memory management unit**, or MMU. It sits between the core and the caches.
6. Looking up a table in memory on every access would be hopeless, so the MMU keeps a small very fast cache of recent translations, called the **translation lookaside buffer**, or TLB. ### 11.6.2 PLAIN - a picture in your head
7. Imagine a huge hotel with a million rooms and a guest list.
8. A flat guest list with one line per room would be a book of a million lines, which you would have to carry everywhere.
9. Instead the hotel uses a tree. One thin card lists the buildings. Each building has a card listing floors. Each floor has a card listing corridors. Each corridor has a card listing rooms.
10. To find a guest you read four cards. But you only need cards for parts of the hotel that actually have guests.
11. The receptionist remembers the last few guests they looked up and answers those instantly without touching any card. That is the TLB.

Where this comparison breaks: the tree is walked by hardware, in silicon, not by anyone reading. And a missing card does not mean "no such guest". It can mean "that guest is out, fetch them from storage", which is a page fault. ### 11.6.3 PLAIN - a worked example

1. This is a real translation, measured on the Linux machine used for this chapter by reading `/proc/self/pagemap` as root.
2. The virtual address was 0x7f1e1a9ff010. On x86-64 with 4-level paging, a 64-bit address is chopped like this:

```
bits 63..48  must copy bit 47   (canonical check)
bits 47..39  index into level 4 table  (PML4)
bits 38..30  index into level 3 table  (PDPT)
bits 29..21  index into level 2 table  (PD)
bits 20..12  index into level 1 table  (PT)
bits 11..0   byte offset inside the 4 KB page
```

3. Each index is 9 bits, so each table has 2^9 = 512 entries. Each entry is 8 bytes, so each table is exactly 512 x 8 = 4096 bytes: one page. That is not a coincidence, it is the design.
4. Chopping 0x7f1e1a9ff010 gives:

Field	Value
PML4 index	254
PDPT index	120
PD index	212
PT index	511
Byte offset	16

5. The walk: read entry 254 of the level 4 table to get the address of a level 3 table. Read entry 120 of that to get a level 2 table. Read entry 212 of that to get a level 1 table. Read entry 511 of that to get the frame number.
6. Physical address = frame number x 4096 + offset = 0x162B8D000 + 0x10 = 0x162B8D010.
7. Now look at the next two pages of the same allocation, which are next to each other in virtual space:

Virtual address	Frame number
0x7f1e1a9ff010	0x162B8D
0x7f1e1aa00010	0x1C18DD
0x7f1e1aa01010	0x13BA83

8. Three pages that are neighbours in the program's view live in three scattered, unrelated places in the actual memory chips. ### 11.6.4 PLAIN - what is really happening inside
9. A register in the CPU holds the physical address of the top-level table. On x86-64 it is called CR3. On Arm64 it is TTBR0 and TTBR1.
10. Switching from one process to another means, at heart, writing a new value into that register. Everything the old process could reach becomes unreachable in one instruction.
11. On every memory access the MMU first asks the TLB. If the translation is there, done, no table reading at all.
12. If it is not, the hardware itself walks the tree. On x86-64 that is done by a hardware page walker, with no software involved and no interrupt.
13. Each entry it reads is 64 bits and carries more than an address. It carries flag bits: present, writable, user-accessible, accessed, dirty, and no-execute.
14. If the present bit is 0, the walker gives up and raises a **page fault**, which is an exception handled by the operating system. Section 11.7 covers what the operating system does next.
15. On success the translation is written into the TLB, and a few clock ticks later the data arrives. ### 11.6.5 TECHNICAL - the engineer's version
16. Page size 4096 bytes is a **convention** so widespread it feels like a standard, but it is not universal. It came from the VAX-11/780 in 1978 and was carried into x86 in 1985 with the 80386. Apple silicon uses 16 KB pages on macOS and iOS. Confirm with `getconf PAGE_SIZE`, which prints 4096 on x86-64 Linux and 16384 on an Apple silicon Mac.
17. Historical note worth knowing: the first machine with what we now call virtual memory was the Atlas, built by a joint University of Manchester and Ferranti team led by Tom Kilburn, and inaugurated in December 1962. They called it the **one-level store**. It had a 16,000 word core store and a 96,000 word drum, a block size of 512 words, and a replacement policy called the learning program that estimated reuse cycles per page and evicted the one predicted to be needed furthest in the future.
18. Page table levels on x86-64:

Levels	Name	Virtual bits
4	PML4, PDPT, PD, PT	48
5	PML5 added on top	57

4. Page table entry layout on x86-64, 64 bits wide:

Bit	Name	Meaning
0	P	present

Bit	Name	Meaning
1	R/W	writable
2	U/S	user accessible
5	A	accessed
6	D	dirty
7	PS	page size, huge page
51..12	PFN	frame number
63	NX	no execute

5. TLB structure on recent parts. AMD Zen 4 has a 72-entry fully associative L1 data TLB and a 3072-entry L2 data TLB. Intel Golden Cove has a 96-entry L1 data TLB and a 2048-entry shared L2 TLB. Figures for Zen 5 were not fully published at the time of writing, so treat any specific number for it as uncertain.
6. TLB reach, meaning how much memory the TLB can cover at once, is the number that matters:

TLB and page size	Reach
64 entries x 4 KB	256 KB
2048 entries x 4 KB	8 MB
2048 entries x 2 MB	4 GB
2048 entries x 1 GB	2 TB

7. Measure TLB behaviour with `perf stat -e dTLB-load-misses,dtlb_load_misses.walk_active ./prog` on Linux. Correlate with `perf stat -e page-faults`.
8. Every virtual machine adds a second layer. The guest's page tables map guest virtual to guest physical, and a second set, called extended page tables on Intel and nested page tables on AMD, maps guest physical to host physical. A full nested walk can require up to 24 memory accesses on a 4-level by 4-level system, which is why the measured latencies in this chapter, taken inside a virtual machine, are higher than bare metal.

```
virtual address 0x7f1e1a9ff010
      |
      |  CR3 -> level 4 table
      +--> [254] --> level 3 table
                +--> [120] --> level 2 table
                          +--> [212] --> level 1 table
                                    +--> [511] --> frame 0x162B8D
                                                       |
      physical address = 0x162B8D000 + 0x010 ------+
                       = 0x162B8D010
```

11.6.6 WORDS - remember these

1. Page — a fixed block of a program's memory — the unit of virtual to physical mapping, normally 4 KB, or 16 KB on Apple silicon.
2. Frame — a fixed block of real memory — a physical page-sized region of DRAM, identified by a page frame number.
3. Page table — the map from pages to frames — a radix tree of 512-entry tables walked by hardware.
4. MMU — the translator — memory management unit, the hardware performing address translation and permission checks.
5. TLB — the translator's memory of recent answers — translation lookaside buffer, a small associative cache of page table entries.
6. CR3 — where the map starts — the x86-64 control register holding the physical address of the top-level page table; TTBR0 and TTBR1 on Arm64. ## 11.7 Paging and swapping

11.7.1 PLAIN - in simple words

1. A **page fault** is the hardware saying "I cannot translate this address, you deal with it". Control jumps into the operating system.
2. That is not an error. It is the normal way memory gets allocated, and there are three kinds, which you must be able to tell apart.
3. A **minor fault** means the data is already in RAM somewhere, and the kernel only has to write a page table entry. Fast: microseconds.
4. A **major fault** means the data must be read from disk first. Slow: tens to thousands of microseconds.
5. An **invalid fault** means the address is genuinely not yours. The program is killed, with a message like segmentation fault.
6. When RAM runs short, the kernel writes some pages out to disk and takes the RAM back. That is **swapping**.
7. If it has to keep bringing them straight back, the machine spends all its time moving pages and none doing work. That is **thrashing**, and it is why a full machine feels frozen rather than merely slow.

11.7.2 PLAIN - a picture in your head

1. Picture a small desk and a big filing cabinet behind you.
2. A document that is in the room but under a pile: you find it in a second. That is a minor fault.
3. A document in the cabinet: you stand, walk, open a drawer, search. That is a major fault, and it takes a thousand times longer.
4. Now imagine the desk holds four documents but the task needs five, and you cycle through all five over and over.
5. Every single step needs a trip to the cabinet, and you also have to file something away first to make room. You get nothing done.
6. Adding a bigger desk fixes it completely. Adding a faster cabinet barely helps.

Where this comparison breaks: the desk in a computer is also full of documents nobody asked for, kept just in case, which the kernel will silently throw away the instant they are needed. That is the page cache, and it is why "free memory" is such a misleading measurement.

11.7.3 PLAIN - a worked example

1. Here are real numbers measured for this chapter, on the same 2.1 GHz Linux virtual machine.
2. A program mapped 512 MB of anonymous memory and wrote one byte in each 4 KB page, then wrote one byte in each page again.

Pass	Total	Per page
First touch, faults	0.220 to 0.271 s	1677 to 2071 ns
Second touch, no faults	0.002 to 0.003 s	18 to 20 ns

3. So a minor page fault cost about 100 times as much as a plain memory write.
4. 131,155 - 83 = 131,072. And 512 MB / 4 KB = 131,072. One fault per page, exactly.
5. Second experiment, showing that a promise is not memory:

```
at start            VmSize =    2.6 MB   VmRSS =    1.4 MB
after malloc 4 GB   VmSize = 4098.6 MB   VmRSS =    1.7 MB
after touching 1 GB VmSize = 4098.6 MB   VmRSS = 1025.7 MB
after touching 4 GB VmSize = 4098.6 MB   VmRSS = 4097.7 MB
```

6. Asking for 4 GB moved resident memory by 0.3 MB. The memory appeared only when it was written to.

7. Now put a cost on a major fault. A minor fault is about 2 microseconds. A read from an NVMe SSD is about 60 microseconds. A read from a hard disk is about 8 milliseconds.
8. So one major fault to SSD costs roughly 30 minor faults. One major fault to a hard disk costs roughly 4,000 minor faults, or about 90,000 ordinary memory writes. ### 11.7.4 PLAIN - what is really happening inside
9. When you call `malloc` for a large block, the C library asks the kernel for a mapping. The kernel records "this range of addresses belongs to you" in a list of regions, and stops there.
10. Your first write to that region raises a page fault. The kernel finds a free physical frame, fills it with zeros for safety, writes the page table entry, and returns to the exact instruction that faulted, which now succeeds.
11. Reading a file works the same way. The kernel keeps recently used file contents in RAM in the **page cache**. If your fault can be satisfied from the page cache, it is minor. If not, real disk input starts and it is major.
12. When free memory gets low, a kernel thread scans and reclaims. It has two easy sources: clean file pages, which can simply be dropped because the copy on disk is identical, and clean anonymous pages, which are rare.
13. Dirty pages must be written first. File pages go back to their file. Anonymous pages have no file, so they go to **swap**: a dedicated partition or a file, called the page file on Windows.
14. Before writing anything out, modern systems try compressing instead. Squeezing three pages into one costs microseconds of CPU and saves milliseconds of disk. ### 11.7.5 TECHNICAL - the engineer's version
15. Fault taxonomy in the terms the tools use:

Kind	Cause	Typical cost
Minor	anon first touch, COW, cache hit	1 to 3 us
Major	needs disk or swap read	50 us to 10 ms
Invalid	no mapping or bad access	SIGSEGV

2. Copy-on-write faults are minor but not cheap. Measured on this machine, a process with 1 GB resident forked, and the child then wrote every page: about 16 microseconds per page, against about 2 microseconds for a plain first-touch fault. That gap is larger than on bare metal because nested paging in a virtual machine makes TLB invalidation expensive. Treat 16 microseconds as this machine's figure, not a universal one.
3. Overcommit is a **policy**, not a law. Linux exposes it through `/proc/sys/vm/overcommit_memory`: 0 is heuristic and the default, 1 always allows, 2 enforces a strict limit set by `overcommit_ratio`. With the default, `malloc` of more memory than you have usually succeeds.
4. Memory compression, with dates:

System	Feature	Since
macOS	Compressed Memory	10.9, 2013
Windows	Memory Compression	Windows 10, 2015
Linux	zswap	kernel 3.11, 2013
Linux	zram	mainline 3.14, 2014

5. macOS compressed memory was introduced in OS X 10.9 Mavericks, announced June 2013 and released October 2013, using the WKdm family of fast dictionary compressors. Typical ratio on real workloads is around 2 to 1. zswap and zram commonly use LZO or LZ4 and report similar ratios.

6. Reading memory pressure on macOS. Activity Monitor's Memory tab shows a Memory Pressure graph, which is driven by the kernel's own pressure state, not by free memory. Green means fine. Yellow means reclaim is working hard. Red means applications are about to be terminated. On the command line:

```
vm_stat 1                # page-level counters, 4096-byte units
memory_pressure          # prints the current pressure level
top -l 1 -s 0 -n 0       # PhysMem and compressor lines
footprint -p PID         # per-process accounting, macOS 11+
```

7. Reading memory pressure on Linux:

```
free -m                  # the -/+ buffers view, and available
vmstat 1                 # watch si and so: swap in and out per s
cat /proc/meminfo        # everything, in kB
cat /proc/pressure/memory # PSI: some/full stall percentages
sar -B 1                 # pgpgin, pgpgout, majflt per second
```

8. On `free -m`, the column to trust is **available**, not free. On the machine used here: total 8023 MB, used 773 MB, free 6573 MB, buff/cache 910 MB, available 7249 MB. Available exceeds free because most of buff/cache can be reclaimed instantly.
9. Pressure Stall Information, added in Linux 4.20 in December 2018, is the best single signal. `/proc/pressure/memory` gives the percentage of time tasks were stalled waiting on memory. A `full` average above a few percent means real trouble, long before free memory looks alarming. ### 11.7.6 WORDS - remember these
10. Page fault — the hardware asking the kernel for help — an exception raised when translation or permission check fails.
11. Minor fault — fixable without disk — a fault resolved by mapping an existing or newly zeroed frame.
12. Major fault — needs disk input — a fault requiring a read from a file or from swap before it can be satisfied.
13. Demand paging — nothing until you touch it — allocating physical frames lazily on first access rather than at request time.
14. Page cache — file contents kept in spare RAM — the kernel cache of file pages, reclaimable on demand.
15. Swap — RAM contents parked on disk — backing store for anonymous pages; called the page file on Windows.
16. Thrashing — all paging, no progress — a state where the working set exceeds RAM and nearly every access faults. ## 11.8 Huge pages, memory protection and isolation

11.8.1 PLAIN - in simple words

1. A 4 KB page is small. A program using 64 gigabytes needs 16 million translations.
2. The translator's fast memory, the TLB, holds only a couple of thousand. So big programs miss constantly.
3. The fix is bigger pages. On a normal PC you can have 2 megabyte pages and 1 gigabyte pages.
4. One 2 MB page replaces 512 small ones, so the same TLB now covers 512 times as much memory.
5. Separately, every page carries permission flags: may be read, may be written, may be executed as code.
6. A sane rule is that a page is never both writable and executable at once. That is called **W xor X**, written W^X.
7. Programs are also loaded at a different random address every run, so an attacker cannot know where anything is. That is **address space layout randomization**. ### 11.8.2 PLAIN - a picture in your head
8. Think of a warehouse index card system again. Small pages are like indexing every single box.
9. Huge pages are like indexing whole pallets. Far fewer cards, so the whole index fits in your pocket.

10. Permissions are like signs on a room: READ ONLY, NO ENTRY, STAFF ONLY.
11. W^X is the rule "a room may be a workshop or a library, never both". An attacker who sneaks a fake instruction into a data room finds it cannot be executed there.
12. Randomization is renumbering all the rooms every morning. A burglar with yesterday's map is lost.

Where this comparison breaks: the signs are checked by hardware on every single access, in parallel with the lookup, so they cost nothing. No human system can check every action for free. ### 11.8.3 PLAIN - a worked example

1. Measured on this machine: a random pointer chase over 512 MB, once with ordinary 4 KB pages and once with transparent huge pages of 2 MB.

Pages used	Time per access
4 KB pages	170.2 to 180.8 ns
2 MB huge pages	136.4 to 139.5 ns

2. The speed-up was 1.25 to 1.30 times. Not enormous, but free, on a workload the CPU was already handling as badly as possible.
3. The arithmetic behind it. A 2048-entry TLB with 4 KB pages covers 2048 x 4 KB = 8 MB. With 2 MB pages it covers 2048 x 2 MB = 4 GB.
4. Now the permission side. Read a real memory map. Note the last column of letters:

```
55a209398000-55a209399000 r--p   /tmp/mb/maps2
55a209399000-55a20939a000 r-xp   /tmp/mb/maps2
55a20939a000-55a20939b000 r--p   /tmp/mb/maps2
55a20939c000-55a20939d000 rw-p   /tmp/mb/maps2
55a23da28000-55a23da49000 rw-p   [heap]
7f19b8c28000-7f19b8db0000 r-xp   libc.so.6
7ffd3da2b000-7ffd3da4d000 rw-p   [stack]
```

5. Every single region is either `r-x` or `rw-`. Not one is `rwx`. That is W^X being enforced, visible in one command.

11.8.4 PLAIN - what is really happening inside

1. A huge page is not a special kind of memory. It is the page walk stopping early.
2. Normally the level 2 entry points at a level 1 table. If its page-size flag is set, the walker treats it as pointing directly at a 2 MB region and stops. One less level, one less table, one TLB entry for 2 MB.
3. There are two ways to get them. Reserved pools, where you set aside a fixed number at boot and programs must ask explicitly. And transparent huge pages, where the kernel promotes ordinary allocations automatically and quietly.
4. The no-execute flag is bit 63 of a page table entry. When set, any attempt to fetch an instruction from that page faults.
5. Isolation needs no extra machinery. Process A's page table has entries for A's frames. Process B's has entries for B's. There is no address A can name that reaches B's memory, because a name is only meaningful through A's own table. ### 11.8.5 TECHNICAL - the engineer's version
6. Page sizes by architecture:

Architecture	Base page	Large pages
x86-64	4 KB	2 MB, 1 GB
Arm64, 4 KB granule	4 KB	2 MB, 1 GB

Architecture	Base page	Large pages
Arm64, 16 KB granule	16 KB	32 MB
Arm64, 64 KB granule	64 KB	512 MB

2. Apple silicon Macs and iOS devices use the 16 KB granule. `getconf PAGE_SIZE` returns 16384 there and 4096 on x86-64 Linux. Android has been moving apps and libraries to 16 KB page alignment since 2024 for the same reason. Support for 1 GB pages on x86-64 is reported by the `pdpe1gb` CPU flag in `/proc/cpuinfo`.
3. Two mechanisms on Linux, and they behave very differently:

```
# transparent huge pages: automatic, best effort
cat /sys/kernel/mm/transparent_hugepage/enabled
# prints e.g.  always [madvise] never
madvise(p, len, MADV_HUGEPAGE);   # opt in from code

# hugetlbfs: reserved, guaranteed, never swapped
echo 512 > /proc/sys/vm/nr_hugepages
grep -i huge /proc/meminfo
mmap(..., MAP_HUGETLB | MAP_HUGE_2MB, ...);
```

4. Where huge pages help most: large in-memory databases, JVM heaps, HPC working sets, and virtual machine guest memory, where they cut nested page walk cost sharply. Where they hurt: memory-tight desktops, workloads with many small sparse mappings, and latency-sensitive services that suffer from the kernel's page-compaction pauses. Oracle, PostgreSQL and Redis documentation all recommend disabling transparent huge pages while using explicit hugetlb pages instead. That disagreement between "automatic is good" and "automatic causes latency spikes" is real and unresolved.
5. Protection history, with dates:

Feature	First shipped	Year
NX bit	AMD64, Athlon 64	2003
W^X default	OpenBSD 3.3	2003
DEP	Windows XP SP2	2004
ASLR design	PaX project	2001
ASLR default	OpenBSD 3.4	2003

6. Fuller ASLR timeline: PaX published the first design and implementation in July 2001. OpenBSD 3.4 in 2003 was the first mainstream operating system with it on by default. Linux enabled a weak form by default from kernel 2.6.12 in June 2005. Windows Vista added it for opted-in binaries, RTM November 2006. macOS randomized system libraries in 10.5 Leopard in October 2007, extended it to all applications in 10.7 Lion in July 2011, and added kernel randomization in 10.8 Mountain Lion in July 2012.
7. W^X is not free in practice. Just-in-time compilers must generate code and then run it. The modern answer is dual mapping: map the same physical pages twice, once writable and once executable, and never both at the same address. Apple silicon requires this and provides `pthread_jit_write_protect_np` to flip a thread between write mode and execute mode.
8. Meltdown, disclosed in January 2018, broke the assumption that the user-accessible flag alone was enough, because speculative execution leaked kernel data through cache timing. The fix, kernel page table isolation, gives the kernel a separate set of page tables so kernel memory is not mapped at all while user code runs. It costs 5 to 30 percent on syscall-heavy workloads, which is why PCID, process context identifiers, matter: they let the CPU keep TLB entries across the switch instead of flushing. ### 11.8.6 WORDS - remember these

9. Huge page — one big page instead of many small — a mapping created by terminating the page walk early, 2 MB or 1 GB on x86-64.
10. TLB reach — how much memory the translator covers — TLB entry count multiplied by page size.
11. THP — automatic huge pages — transparent huge pages, kernel-managed promotion of 4 KB mappings to 2 MB.
12. NX bit — this page is data, not code — bit 63 of a page table entry marking the region non-executable.
13. W^X — never writable and executable at once — a policy that no mapping carries both PROT_WRITE and PROT_EXEC.
14. ASLR — everything moves every run — address space layout randomization, the randomizing of load addresses to defeat fixed-address exploits. ## 11.9 How a program's memory is laid out

11.9.1 PLAIN - in simple words

1. A running program's address space is not one blob. It is a set of regions, each with a job and a permission.
2. **Text** is the machine code. Readable and executable, never writable.
3. **Data** holds variables that start with a value, for example a counter set to 5. It is copied from the file at load time.
4. **BSS** holds variables that start at zero. It takes no space in the file at all, only a note saying "reserve this many zero bytes".
5. **Heap** is memory you ask for while running, and give back when done.
6. **Stack** holds local variables and the trail of function calls. It grows and shrinks automatically as functions are entered and left.
7. **Mapped regions** are everything else: shared libraries, big allocations, files made to look like memory.

11.9.2 PLAIN - a picture in your head

1. Picture a long street. At the low-numbered end is a printing works that cannot be altered: the code.
2. Then a builder's yard that grows outward as you request more land: the heap.
3. At the high-numbered end is a stack of trays. Each function call puts a new tray on top; returning takes it off. It grows toward the yard.
4. If the trays pile up so high they reach the yard, something has gone badly wrong. That is a stack overflow.

Where this comparison breaks: modern systems leave an enormous unused gap between the stack and the heap, and put a deliberately unmapped guard page just below the stack, so the two cannot silently collide. The crash is arranged in advance rather than allowed to happen.

11.9.3 PLAIN - a worked example

1. Here is the real map of a tiny C program on the Linux machine used for this chapter, printed by reading `/proc/self/maps`. Columns are range, permissions and what it is.

```
55a209398000-55a209399000 r--p   maps2      read-only data
55a209399000-55a20939a000 r-xp   maps2      the code (text)
55a20939a000-55a20939b000 r--p   maps2      constants
55a20939c000-55a20939d000 rw-p   maps2      data and BSS
55a23da28000-55a23da49000 rw-p   [heap]     small allocations
7f19b4bff000-7f19b8c00000 rw-p   (anon)     the 64 MB malloc
7f19b8c00000-7f19b8c28000 r--p   libc.so.6  library headers
7f19b8c28000-7f19b8db0000 r-xp   libc.so.6  library code
7f19b8e03000-7f19b8e05000 rw-p   libc.so.6  library variables
7f19b8ef7000-7f19b8ef9000 r-xp   [vdso]     kernel fast calls
7ffd3da2b000-7ffd3da4d000 rw-p   [stack]    the stack
```

2. Read the permissions. Code is `r-x`. Data is `rw-`. Nothing is `rwx`.

3. The heap region is only 132 KB. The 64 MB request did not go there. It got its own mapping at 0x7f19b4bff000, because glibc sends large requests straight to the kernel instead of extending the heap.
4. The stack is 0x7ffd3da2b000 to 0x7ffd3da4d000, which is 136 KB right now. It can grow down to the limit, which `ulimit -s` reported as 8192 KB. ### 11.9.4 PLAIN - what is really happening inside
5. `malloc` is a library function, not a system call. Most of the time it never talks to the kernel.
6. It keeps a pool of memory it already owns and hands out pieces from it, remembering which pieces are free in linked lists called **free lists**.
7. `free` does not return memory to the operating system. It puts the piece back on a free list, and often merges it with neighbouring free pieces.
8. That is why a program's memory use rarely goes down after a big job finishes. The memory is free to the program, still owned from the kernel's point of view.
9. Repeated allocate-and-free of different sizes leaves the pool full of holes. Plenty of free bytes, none of them big enough. That is **fragmentation**.
10. A **use-after-free** is keeping a pointer to something you freed. The allocator may have handed those bytes to someone else, so you now read or write their data. It is one of the most common serious security bugs.
11. Garbage collection removes the need to call free. The runtime finds objects nothing points at any more and reclaims them automatically.

11.9.5 TECHNICAL - the engineer's version

1. Segment names as the linker uses them. In ELF: `.text`, `.rodata`, `.data`, `.bss`. Inspect with `size ./prog` and `readelf -S ./prog` on Linux, or `size -m` and `otool -l` on macOS.
2. glibc `malloc` is ptmalloc2, derived from Doug Lea's dlmalloc. Key behaviours, all **implementation details** and all tunable:

Mechanism	Default
mmap threshold	128 KB, grows to 32 MB
Arenas per process	up to 8 x cores
Per-thread cache	tcache, glibc 2.26, 2017
Bin classes	fast, tcache, small, large

3. Arenas exist to reduce lock contention: each thread is steered to its own arena so allocations do not serialize. Set `MALLOC_ARENA_MAX` to limit it, which is a common fix for containers showing surprising memory growth.
4. Alternative allocators, with origins: tcmalloc from Google in 2005, jemalloc written by Jason Evans for FreeBSD in 2006 and later used by Firefox and Facebook, and mimalloc from Microsoft Research in 2019. Swapping allocator by `LD_PRELOAD` alone often changes throughput by 5 to 30 percent on allocation-heavy servers.
5. Slab allocation, published by Jeff Bonwick for SunOS in a 1994 USENIX paper, caches whole pre-constructed objects of one type. Linux uses SLUB today. Inspect with `slabtop` or `cat /proc/slabinfo`. Physical pages themselves are handed out by a buddy allocator, visible in `/proc/buddyinfo`.
6. Fragmentation has two forms. **Internal** is the unused tail inside a block rounded up to a size class. **External** is free memory split into pieces too small to use. Size-class allocators trade a little internal fragmentation to nearly eliminate external fragmentation.
7. Detecting the classic bugs:

```
valgrind --leak-check=full ./prog        # leaks, invalid access
gcc -fsanitize=address,undefined ./p.c   # fast, needs rebuild
heaptrack ./prog                         # allocation profiles
```

```
leaks PID          # macOS built in
MallocStackLogging=1 ./prog   # macOS allocation traces
```

8. The honest version: "free never returns memory to the operating system" is not quite true. glibc will trim the top of the heap with `brk` when a large contiguous run at the end becomes free, and blocks that were served by `mmap` are unmapped on `free`. What is true is that blocks freed from the middle of the heap stay owned by the process.
9. Garbage collection changes the failure modes rather than removing them. Leaks become "still reachable but never used", which no collector can fix. Use-after-free largely disappears. In exchange you get pauses, extra memory headroom, and worse locality unless the collector compacts. Java's G1 became default in JDK 9 in 2017 and ZGC reached production readiness in JDK 15 in 2020, targeting sub-millisecond pauses; Go has run a concurrent collector with sub-millisecond pauses since Go 1.8 in 2017. Rust takes the third path: no collector, ownership checked at compile time.

11.9.6 WORDS - remember these

1. Text segment — the code — the read-only executable region loaded from the binary, mapped `r-x`.
2. BSS — the zero-filled variables — uninitialized static storage, reserved by the loader and absent from the file.
3. Heap — memory you ask for at run time — the region managed by the allocator through `brk` and `mmap`.
4. Stack — the trail of function calls — a downward-growing region holding frames, guarded by an unmapped page.
5. Free list — the allocator's list of spare pieces — linked structures of reclaimed blocks, usually bucketed by size class.
6. Fragmentation — free but unusable — memory split so that no single free region satisfies a request. ## 11.10 Shared memory and mapped files

11.10.1 PLAIN - in simple words

1. Two page tables can point at the same physical frame. That is all shared memory is.
2. A file can also be attached to a range of addresses. Reading those addresses reads the file. That is a **memory-mapped file**.
3. **Copy-on-write** is the clever middle case. Two processes share a page and both see it, but the page is marked read-only.
4. The moment either one writes, the hardware faults, the kernel makes a private copy for the writer, and the two go their separate ways.
5. That is why starting a copy of a process is cheap: nothing is copied until something is changed. ### 11.10.2 PLAIN - a picture in your head
6. Think of a reference book in a shared office.
7. Everyone's desk index says "the dictionary is in cabinet 4". One book, many pointers to it. That is a shared library.
8. Now a rule: you may read it, but if you want to scribble in it, you must first photocopy the page you want to change and scribble on your copy.
9. A memory-mapped file is the librarian agreeing to place any page of any book onto your desk the instant you look for it, without you filling in a request slip.

Where this comparison breaks: photocopying is per page, not per book, and the granularity is exactly 4 KB. Changing one byte copies 4096.

11.10.3 PLAIN - a worked example

1. Measured for this chapter. A process filled 1 GB of memory, then forked.

```
parent RSS after filling 1 GB : 1050060 KB
child: fork returned after      15.07 ms
child RSS immediately after fork: 1049404 KB
child: after writing all 1 GB   : copy cost 4202 ms
                                = 16.0 us per 4 KB page
```

2. Forking a 1 GB process took 15 milliseconds, not the seconds a real 1 GB copy would need. Almost nothing was copied.
3. The child's reported resident size was immediately about 1 GB, which looks like a copy but is not. Resident size counts shared pages in full, for both processes. This is the single biggest reason memory numbers confuse people, and section 11.11 deals with it.
4. Second measurement: the sharing of library code. `/proc/self/smaps_rollup` for a small program reported:

Counter	Value
Rss	1640 kB
Pss	397 kB
Shared_Clean	1496 kB
Private_Dirty	108 kB

5. Resident size 1640 kB, but proportional size only 397 kB, because 1496 kB of it is shared library code counted once per sharer. The program's genuinely private, modified memory is 108 kB.

11.10.4 PLAIN - what is really happening inside

1. `mmap` asks the kernel to attach a range of addresses to something: a file, or nothing at all, which is called an anonymous mapping.
2. Two flags decide the sharing behaviour. `MAP_SHARED` means writes go to the underlying file and are visible to everyone. `MAP_PRIVATE` means writes become your own copy-on-write copies and are never written back.
3. No pages are brought in at mmap time. The mapping is a note. Pages arrive on first touch, exactly as in section 11.7.
4. On fork the kernel copies the page tables, not the pages, and marks every writable private page read-only in both parent and child.
5. Any write now faults. The kernel checks that the page is a copy-on-write page, allocates a fresh frame, copies 4096 bytes, points the writer's entry at the copy and makes it writable again.
6. For a mapped file, a write marks the page dirty. A kernel writeback thread sends dirty pages to the file later, or at `msync`, or at unmap. ### 11.10.5 TECHNICAL - the engineer's version
7. The core call, with the flags that matter:

```
void *p = mmap(NULL, len,
               PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

/* file made to look like memory */
int fd = open("big.dat", O_RDWR);
void *f = mmap(NULL, len, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
msync(f, len, MS_SYNC);      /* force it to disk */
madvise(f, len, MADV_SEQUENTIAL | MADV_WILLNEED);
```

2. mmap appeared in the Berkeley and Sun Unix line in the 1980s and is now POSIX. The Windows equivalents are `CreateFileMapping` and `MapViewOfFile`.
3. When mapped files win: random access to a large file, many processes reading the same file, and avoiding the copy from page cache into a user buffer. When they lose: sequential streaming, where `read` with a large buffer is usually faster and has predictable memory behaviour; files that may be truncated underneath you, which turns into SIGBUS; and network filesystems, where a page fault can block on the network with no error path.
4. Databases disagree publicly about this. LMDB and older MongoDB storage engines were built on mmap. The 2022 paper "Are You Sure You Want to Use MMAP in Your Database Management System" from CMU argues against it, citing loss of control over eviction, transactional safety and page fault costs. PostgreSQL and InnoDB manage their own buffer pools instead. This is a genuine, live disagreement between competent engineers.
5. Copy-on-write is why `fork` is cheap and why `fork` in a large process is still not free: page table copying is proportional to the number of mapped pages, and every subsequent write costs a fault. Measured here: 15 ms to fork 1 GB, then 16 microseconds per page written.
6. This has a famous consequence for garbage-collected and reference-counted runtimes. CPython updates a reference count on every object touched, which writes to the object header, which triggers copy-on-write. A forked CPython worker therefore un-shares memory quickly. CPython gained `gc.freeze` in 3.7, released 2018, partly to reduce this. ### 11.10.6 WORDS - remember these
7. mmap — attach memory to a thing — the system call mapping a file or anonymous memory into an address space.
8. Anonymous mapping — memory backed by nothing — a mapping with no file, zero filled on first touch, backed by swap if needed.
9. Copy-on-write — share until someone writes — a policy of mapping pages read-only and duplicating them on the first write fault.
10. Memory-mapped file — a file that looks like an array — file contents made accessible through ordinary loads and stores via the page cache.
11. PSS — your fair share of shared memory — proportional set size, private pages plus each shared page divided by its number of sharers.

11.11 Measuring memory for real

11.11.1 PLAIN - in simple words

1. Ask "how much memory is this program using" and there is no single true answer. There are about six, and they all mean different things.
2. **Virtual size** is how much address space the program has claimed. It can be enormous and mean nothing.
3. **Resident size** is how much is actually in RAM right now. Closer to useful, but it counts shared things in full for every sharer.
4. **Private** is the part only this process has. This is the number that would actually be freed if you killed it.
5. Add up resident sizes for all processes and you will get far more than the machine has, because shared pages are counted many times. ### 11.11.2 PLAIN - a picture in your head
6. Five flatmates share one kitchen, and each has a bedroom.
7. Ask each "how much of the flat do you use" and each says "my bedroom plus the kitchen". Add the answers and you get more than the flat.
8. Virtual size is like counting the whole building because you have a key to the front door.
9. Proportional is bedroom plus one fifth of the kitchen. The five proportional answers do add up to the

flat exactly.
10. Private is just the bedroom: what is actually emptied when someone leaves.

Where this comparison breaks: the kitchen can vanish and reappear. Clean file pages are dropped and reloaded silently, so the numbers move even when nothing in the program changes. ### 11.11.3 PLAIN - a worked example

1. Real numbers from this chapter's test program.

Moment	VmSize	VmRSS
At start	2.6 MB	1.4 MB
After malloc 4 GB	4098.6 MB	1.7 MB
After touching 1 GB	4098.6 MB	1025.7 MB
After touching 4 GB	4098.6 MB	4097.7 MB

2. Virtual size jumped by 4096 MB the moment memory was requested. Resident size moved by 0.3 MB. Anyone monitoring virtual size would have raised a false alarm.
3. Now the sharing side, from `/proc/self/smaps_rollup`: Rss 1640 kB but Pss 397 kB and Private_Dirty 108 kB.
4. Rule of thumb. To answer "will this fit", use PSS or private dirty. To answer "is this leaking", watch private dirty over time. Ignore virtual size unless you are chasing address space exhaustion.

11.11.4 PLAIN - what is really happening inside

1. The kernel maintains, per process, a list of mappings and a count of resident pages.
2. Resident size is the count of pages currently present in physical memory, times the page size, whether shared or not.
3. To compute proportional size the kernel must, for every page, look up how many processes map it and divide. That is expensive, which is why `/proc/PID/smaps` is much slower to read than `/proc/PID/status`.
4. Dirty pages are the ones modified since being loaded. Private dirty memory is the only kind that must be written to swap before it can be reclaimed.
5. This is why a process can show 2 GB resident and yet freeing it releases only 200 MB.

11.11.5 TECHNICAL - the engineer's version

1. The counters and where they come from:

Counter	Linux source	Meaning
VSZ	VmSize in status	all mappings summed
RSS	VmRSS in status	resident, shared counted
PSS	smaps, smaps_rollup	shared divided by sharers
USS	Private_* in smaps	unique to this process

2. Commands on Linux:

```
ps -o pid,vsz,rss,comm -p PID
cat /proc/PID/status | grep -E 'VmSize|VmRSS|VmSwap'
cat /proc/PID/smaps_rollup      # Pss, Private_Dirty
pmap -X PID                     # per mapping detail
smem -k -s pss                  # sorted by PSS
systemd-cgtop / cat memory.current  # cgroup v2 truth
```

3. On containers and cgroup v2, `memory.current` in the cgroup directory is the number the kernel enforces limits against, and it includes page cache charged to the group. A container killed for exceeding its limit was often killed by cache, not by the application heap. `memory.stat` breaks it down, and `memory.high` throttles before `memory.max` kills.
4. Commands on macOS:

```
vm_stat            # free, active, wired, compressed pages
top -l 1 -s 0 -n 0     # PhysMem, compressor size, swap
footprint -p PID       # Apple's authoritative per-process figure
vmmap PID              # every region, dirty and swapped columns
leaks PID              # unreachable allocations
```

5. Why the totals never add up, stated precisely: the sum of RSS over all processes double counts every shared page once per sharer, and omits kernel allocations such as slab, page tables and network buffers, which belong to no process. The sum of PSS plus kernel usage does reconcile with `MemTotal - MemFree`, to within a few percent.
6. On the machine used here, `free -m` reported total 8023, used 773, free 6573, buff/cache 910, available 7249. `available` is an estimate produced by the kernel of how much a new workload could get without swapping. It is the only number in that output worth alerting on.

11.11.6 WORDS - remember these

1. VSZ — address space claimed — virtual set size, the sum of all mapping lengths, including untouched ones.
2. RSS — pages actually in RAM — resident set size, counting shared pages in full for every process.
3. PSS — your share of the shared parts — proportional set size, private plus shared divided by sharer count.
4. USS — what freeing it would return — unique set size, the private pages of one process.
5. Available memory — what a new program could get — the kernel's estimate of free plus reclaimable, reported by `free`.

11.12 Cache-friendly programming

11.12.1 PLAIN - in simple words

1. You cannot change the memory hierarchy. You can change how your data sits in it, and that is often worth more than any algorithmic cleverness.
2. Touch memory in the order it is stored. Neighbours, not jumps.
3. Keep the things you use together next to each other, and the things you do not use out of the way.
4. Work on a chunk small enough to stay in cache, finish with it, then move on.
5. Make sure two threads never keep writing to the same 64-byte block. ### 11.12.2 PLAIN - a picture in your head
6. Think of packing a suitcase for a trip.
7. Array of structs is packing one bag per day, each with a shirt, socks, a book, a towel and a laptop charger.
8. Struct of arrays is packing one bag of shirts, one of socks, one of books. Need shirts, carry one bag.
9. Padding is leaving deliberate empty space so two people never reach into the same bag at once and knock each other's hands.
10. Blocking is unpacking one bag at a time on a small table, rather than spreading all seven across the floor.

Where this comparison breaks: if you genuinely need every field of every item, array of structs is the better layout, because it fetches everything in one go. The right choice depends entirely on your access pattern, and there is no universally faster layout. ### 11.12.3 PLAIN - a worked example

1. Change one, loop order. Summing a 4096 by 4096 array of doubles, measured on the 2.1 GHz Xeon used throughout.

```
for (i...) for (j...) s += a[i][j];   /* 0.019 s */
for (j...) for (i...) s += a[i][j];   /* 0.187 s */
```

Result: 9.0 to 10.4 times faster by swapping two lines.

2. Change two, blocking. Multiplying two 1024 by 1024 matrices, plain triple loop against a version that works on 64 by 64 tiles so each tile stays in cache, both compiled with the vectorizer off so only the memory effect is being measured.

Version	Time
Plain i, j, k loops	4.622 s
Tiled, 64 x 64 blocks	0.563 s

Result: 8.21 times faster, same arithmetic, same result.

3. Change three, struct of arrays. Eight million particles of 40 bytes each, summing only the x field.

Layout	Bytes walked	Time
Array of structs	305 MB	0.0251 s
Struct of arrays	31 MB	0.0069 s

Result: 3.5 to 3.6 times faster, because 10 times less data crossed the memory bus.

4. Bonus change, field order. Two structs with identical fields in different order:

```
struct bad  { char a; double b; char c; int d; };  /* 24 bytes */
struct good { double b; int d; char a; char c; };  /* 16 bytes */
```

Sorting fields from largest to smallest cut the size by a third, purely by removing padding. A million of them is 8 MB saved.

11.12.4 PLAIN - what is really happening inside

1. Every read pulls a whole 64-byte line. Your effective bandwidth is (bytes you used) divided by (bytes fetched).
2. Row-order summing uses 64 of 64. Column-order uses 8 of 64. That is the factor of eight, before prefetching and vectorization make it worse.
3. In the particle test, the struct is 40 bytes and you wanted 4 of them. So each 64-byte line delivered about 6 useful bytes. The struct-of-arrays version delivered 64 useful bytes out of 64.
4. Tiling works because a naive matrix multiply walks a whole column of the second matrix for every output element, and that column does not stay in cache. Restricting the work to a 64 by 64 tile makes the three tiles involved fit together in cache, so each loaded value is reused 64 times. ### 11.12.5 TECHNICAL - the engineer's version
5. False sharing, measured for this chapter with two threads pinned to different cores, each doing 30 million atomic increments.

Layout	Time per increment	Penalty
Both counters in one line	27.8 to 35.2 ns	3.1x to 4.2x
Counters 128 bytes apart	6.6 to 11.5 ns	baseline

2. The honest version: with plain non-atomic increments instead of atomic ones, the same test on the same machine showed no measurable penalty, about 0.49 ns per increment either way. A core can hold the line and retire many ordinary stores per ownership transfer, so the ping-pong amortizes. Atomic read-modify-write operations must be globally visible one at a time, so ownership really does move per operation. False sharing is therefore severe for atomics, counters and locks, and often mild for plain stores. Folklore usually omits this distinction.
3. The fix, in three languages:

```
struct counter { _Alignas(64) long value; };
```

```
alignas(std::hardware_destructive_interference_size) long v;
```

```
#[repr(align(64))] struct Padded(AtomicU64);
```

4. Padding to 128 bytes rather than 64 is the safer choice on Intel parts, because the adjacent-line prefetcher can pull in the sibling line. Apple silicon uses 128-byte L2 lines. This is why C++17 provides `hardware_destructive_interference_size` rather than a fixed constant.
5. Alignment rules: a scalar of size N must sit at an address that is a multiple of N. `sizeof` a struct is rounded up to its strictest member alignment so arrays stay aligned. Order fields from widest to narrowest and the padding usually disappears. Check with `pahole ./prog` on Linux, which prints holes explicitly, or `clang -Xclang -fdump-record-layouts`.
6. A checklist that reliably finds wins, roughly in order of payoff:

Change	Typical gain here
Fix traversal order	9x
Block or tile the loop	8x
Array of structs to struct of arrays	3.5x
Pad shared counters	3x to 4x
Enable huge pages	1.3x
Reorder struct fields	size, not speed

7. Verify every change with counters, not intuition:

```
perf stat -e cycles,instructions,cache-misses,\
LLC-load-misses,dTLB-load-misses ./prog
perf c2c record ./prog && perf c2c report   # false sharing
valgrind --tool=cachegrind ./prog           # simulated, exact
```

11.12.6 WORDS - remember these

1. Array of structs — one bag per item — AoS, contiguous records with all fields of one object together.
2. Struct of arrays — one bag per field — SoA, parallel arrays with one field of all objects together.
3. Padding — deliberate empty bytes — filler inserted by the compiler so each field meets its alignment requirement.
4. Blocking — work on one tile at a time — loop tiling, restructuring loops so the active working set fits in a cache level.
5. False sharing — different variables, same line — performance loss when independent data shares one cache line across cores. ## 11.98 Common wrong ideas

6. Wrong: more RAM always makes a machine faster. Right: more RAM removes swapping. If you were not swapping, it changes nothing measurable. Going from 8 GB to 32 GB on a thrashing machine can feel like a new computer; going from 32 GB to 64 GB on a machine that never fills 20 GB changes nothing at all. Faster RAM helps a little; more RAM helps only if you were short.
7. Wrong: swap is bad and should be turned off. Right: swap lets the kernel evict genuinely cold anonymous pages and use that RAM for something useful. Without swap, cold pages are stuck in RAM forever and the only reclaim target left is the page cache, which often makes things worse. Gigabytes sitting in swap with `si` and `so` at zero in `vmstat` is healthy. Constant swap-in and swap-out at the same time is the problem, and that is thrashing, not swap.
8. Wrong: virtual memory is just the page file. Right: virtual memory is address translation. Every process gets its own map from private addresses to real frames. Paging to disk is one optional feature built on top. A machine with swap disabled still uses virtual memory for every instruction it executes.
9. Wrong: "free RAM is wasted RAM" is a myth invented by Linux fans. Right: it is literally how every modern kernel works, including Windows and macOS. Unused RAM is filled with cached file contents which are dropped instantly when a program needs the space. On the machine used here, `free -m` showed 6573 MB free but 7249 MB available, and the difference is exactly cache the kernel will give back for nothing.
10. Wrong: dual channel doubles memory speed. Right: it roughly doubles peak bandwidth, and does not reduce latency at all. A latency-bound program, for example one chasing pointers through a linked list, sees almost no benefit. A bandwidth-bound program, such as an integrated GPU or a large streaming copy, sees a lot.
11. Wrong: lower CAS latency always means faster memory. Right: CAS latency is in clock ticks, so it only means something alongside the clock. DDR4-3200 CL16 and DDR5-6000 CL30 both work out to 10.0 nanoseconds. Convert to nanoseconds with `CL x 2000 / rate` before comparing anything.
12. Wrong: if two processes have 2 GB resident each, they use 4 GB. Right: shared pages, mainly library code and copy-on-write pages after fork, are counted in full for every process. Use PSS, which divides shared pages between sharers and does add up correctly.
13. Wrong: huge pages always help. Right: they help when TLB misses dominate, which mainly means large random working sets. Measured here, transparent huge pages gave 1.3 times on a 512 MB random walk, and major database vendors recommend turning them off because the kernel's compaction work causes latency spikes.

11.99 Chapter summary in 20 lines

1. Fast memory is small and dear, big memory is slow and cheap, so machines build a ladder from registers down to tape.
2. Measured on one 2.1 GHz machine, one random read cost 1.79 ns from L1 and 179.76 ns from RAM: a hundredfold spread for the same instruction.
3. In August 2026, DDR5 cost about 16.20 USD per gigabyte, SSD about 0.09 USD and hard disk about 0.02 USD, after a shortage that roughly quadrupled DRAM prices from 2024 levels.
4. The ladder works only because of locality: things used recently get used again, and their neighbours get used too.
5. Reading a 4096 by 4096 array along rows rather than down columns was 9.0 to 10.4 times faster, with no change to the arithmetic.
6. Memory arrives on sticks called DIMMs, organized into channels, ranks, bank groups, banks, rows and columns.
7. Extra channels multiply bandwidth and do nothing for latency, which is the most misunderstood fact about RAM.
8. True latency in nanoseconds is CL times 2000 divided by the data rate, so DDR4-3200 CL16 and DDR5-6000 CL30 are both exactly 10.0 ns.

9. Transfer rates rose about sixtyfold in twenty-five years while first-byte latency only halved. That gap is the memory wall.
10. The memory controller now lives on the CPU die, splits addresses into channel, rank, bank, row and column, and reorders requests, so latency is a distribution rather than a number.
11. Programs use virtual addresses; only the memory controller sees physical ones; the MMU translates on every access.
12. Memory is divided into 4 KB pages, mapped by a four-level or five-level page table tree rooted in CR3, cached by the TLB.
13. A real translation measured here turned 0x7f1e1a9ff010 into physical 0x162B8D010 through indices 254, 120, 212 and 511.
14. Neighbouring virtual pages land in scattered physical frames, which is the whole point of the mechanism.
15. Page faults are normal: a minor fault cost about 2 microseconds here, against 20 nanoseconds for an ordinary write, and touching 512 MB produced exactly 131,072 of them.
16. Demand paging means asking for 4 GB moves resident memory by 0.3 MB; memory appears only when written to.
17. Huge pages of 2 MB extend TLB reach from 8 MB to 4 GB and gave a measured 1.3 times on a large random walk.
18. Isolation comes free from the page tables themselves, reinforced by W^X from 2003, ASLR from 2001, and kernel page table isolation after Meltdown in 2018.
19. Copy-on-write makes fork cheap: forking a 1 GB process took 15 ms, and copying happened only when the child wrote, at 16 microseconds per page.
20. Layout beats cleverness: loop order gave 9x, tiling gave 8.2x, struct of arrays gave 3.5x, and padding shared atomic counters gave 3x to 4x, all on the same machine, all without changing a single result.

12

CHAPTER

KB

Storage — Disks, SSDs, Flash and Filesystems

12.0 What this chapter gives you

1. You will be able to say exactly what makes storage different from memory, and why every computer needs both.
2. You will be able to draw a hard disk drive and name every moving part in it, and say how one bit is written and read back.
3. You will be able to compute how long a spinning disk takes to reach a random piece of data, from the drive's rotation speed alone.
4. You will be able to explain a flash memory cell, why it wears out, and why erasing works on big blocks but writing works on small pages.
5. You will be able to explain why an SSD contains its own small computer, and what that computer spends its time doing.
6. You will be able to say why a big file copy to a cheap SSD starts fast and then suddenly slows down.
7. You will be able to trace a byte from a program down through the filesystem, the interface and the drive, and name each standard on the way.
8. You will be able to say what happens on disk when you create, write, rename and delete a file, step by step.
9. You will be able to explain why a 4 GB video will not copy onto a FAT32 USB stick, and what to do about it.
10. You will be able to explain journaling, RAID, backups and secure erase well enough to make real decisions with real money.

12.1 Storage versus memory: the real difference

12.1.1 PLAIN - in simple words

1. Memory is where a computer keeps what it is thinking about right now.
2. Storage is where a computer keeps what it wants to still have tomorrow.
3. The big difference is what happens when the power goes off.
4. Memory forgets. Pull the plug and the contents are gone in under a second.
5. Storage remembers. Pull the plug, come back in a year, the file is still there.
6. A thing that keeps its contents without power is called **non-volatile**.
7. A thing that loses its contents without power is called **volatile**.
8. Memory is fast and expensive and small. Storage is slow and cheap and huge.
9. Both exist because no single technology is fast, cheap, huge and permanent all at once. Nobody has found one. Engineers pick two or three and stack the results in layers.

12.1.2 PLAIN - a picture in your head

1. Think of a person working at a desk in a library.
2. The papers spread on the desk are memory. They are within arm's reach and you can grab any of them instantly.
3. The shelves around the room are storage. Getting a book takes you a walk.
4. The desk is small. You cannot fit the library on it.
5. At the end of the day the cleaner clears the desk completely. That is power loss. Anything not filed on a shelf is gone.
6. So you work fast on the desk, and you file to the shelves anything you want tomorrow.
7. There is also a basement archive, far away, cheap, enormous, and slow to reach. That is tape.

Where this comparison breaks: the desk in a real computer is not merely small, it is thousands of times faster than the shelves, not two or three times. Also, a librarian never has to rewrite a whole shelf to change one book, but a flash chip has almost exactly that problem, which we meet in section 12.5.

12.1.3 PLAIN - a worked example

1. Suppose you edit a photo. The photo file lives on your SSD.
2. You double-click it. The computer copies the file from storage into memory.
3. Every brush stroke changes the copy in memory. The file on disk is untouched.
4. You press Save. Now the memory copy is written back down to storage.
5. If the power fails between step 3 and step 4, your strokes are gone and the old file is still there, unharmed.
6. That single sentence explains why software nags you to save.

12.1.4 PLAIN - what is really happening inside

1. Memory in a normal computer is DRAM. Each bit is a tiny capacitor, a bucket holding a little charge.
2. The bucket leaks. It has to be topped up thousands of times per second. That topping up is called **refresh**, and it needs power.
3. Cut the power and every bucket empties within milliseconds. That is why RAM is volatile: not because of a rule, but because of a leak.
4. Storage stores a bit as something physical that does not need topping up: the direction of a magnet on a disk, or a pocket of trapped electrons in flash.
5. Nothing has to actively hold that state. It just sits there.
6. The price of permanence is speed. Flipping a magnet or pushing electrons through an insulator takes far longer than charging a tiny capacitor.
7. So the whole machine is built as a ladder. Tiny and instant at the top, huge and slow at the bottom, with each level acting as a cache for the one below.

12.1.5 TECHNICAL - the engineer's version

1. The memory hierarchy spans roughly eight orders of magnitude in latency and six in price per byte. Figures below are typical for mid-2026 desktop and server parts.

Level	Typical size	Typical latency
CPU register	under 2 KB	under 1 ns
L1 data cache	32 to 128 KB per core	about 1 ns
L2 cache	0.5 to 4 MB per core	3 to 5 ns
L3 cache	16 to 128 MB shared	10 to 25 ns
DRAM (DDR5)	8 to 512 GB	60 to 100 ns
NVMe SSD	0.5 to 64 TB	20 to 80 us
SATA SSD	0.25 to 8 TB	about 100 us

Level	Typical size	Typical latency
Hard disk	1 to 44 TB	5 to 15 ms
LTO tape	18 to 40 TB per tape	30 to 90 s

2. Prices move fast and 2025 to 2026 was unusual. AI datacentre demand pulled NAND and DRAM capacity away from consumers and prices roughly doubled.

Medium	Price per TB, Aug 2026	Note
DDR5 DRAM	about 6,000 to 10,000 USD	approximate, volatile market
NVMe SSD, consumer	from about 105 USD	2 TB and 4 TB class
SATA SSD, consumer	from about 96 USD	4 TB class
Hard disk, 12 to 24 TB	about 21 to 32 USD	new and renewed stock
LTO tape media	roughly 6 to 10 USD	drive costs thousands

3. The honest version: "non-volatile" is not "eternal". NAND flash loses charge over years when unpowered, magnetic domains slowly relax, and tape binders hydrolyse. Non-volatile means "does not need power to hold state for a useful period", not "permanent".
4. Byte-addressable persistent memory sat between DRAM and SSD for a few years. Intel Optane, based on 3D XPoint, shipped from 2017 with roughly 300 ns latency. Intel wound the business down in 2022. The idea is not dead, but no mainstream product occupies that rung today.
5. Tools that show the ladder: `lscpu` and `getconf -a` for cache sizes, `free -h` and `vm_stat` for memory, `lsblk` and `diskutil list` for block devices, `smartctl -a` for drive internals, `fio` for measured latency.

12.1.6 WORDS - remember these

1. Volatile — forgets without power — state decays once supply is removed.
2. Non-volatile — remembers without power — retains state with supply removed.
3. DRAM — the fast forgetful memory — dynamic RAM, one capacitor and one transistor per bit, requiring periodic refresh.
4. Latency — the wait before anything arrives — time from request issue to first byte returned.
5. Throughput — how much arrives per second once it starts — sustained transfer rate, usually MB/s.
6. Memory hierarchy — the ladder of speeds — cache levels plus main memory plus backing store, each caching the level below.

12.2 The old ways: cards, tape, drums, cores and floppies

12.2.1 PLAIN - in simple words

1. Before disks, computers stored data as holes in card, as magnetized spots on ribbon, and as rings of metal threaded on wire.
2. A **punched card** is a stiff paper card. A hole means one thing, no hole means the other. A machine feels for the holes with brushes or light.
3. **Paper tape** is the same idea on a long roll instead of separate cards.
4. **Magnetic tape** is a long plastic ribbon coated with rust-like powder. You magnetize small patches of it to record bits.
5. A **magnetic drum** is a spinning metal cylinder with heads along its side.
6. **Core memory** is a mesh of wires with tiny magnetic rings at every crossing. Each ring stores one bit by which way round it is magnetized.
7. A **floppy disk** is a thin flexible plastic disc, coated like tape, spun inside a sleeve.
8. All of these are gone from daily life except tape, which is quietly bigger than ever.

12.2.2 PLAIN - a picture in your head

1. Picture a cassette tape and a vinyl record sitting next to each other.
2. The cassette must be wound forward to reach the middle of a song. To hear track eight you wait for tracks one to seven to go past.
3. The record lets you drop the needle straight onto track eight.
4. That is the whole difference between **sequential access** and **random access**, and it is the difference between tape and disk.
5. Tape is not slow because the ribbon moves slowly. Modern tape moves fast. Tape is slow to start because the right spot may be half a kilometre away along the ribbon.

Where this comparison breaks: a record needle can reach any groove in a second, while a disk head reaches any track in about ten milliseconds, a thousand times quicker. And nobody streams a record at 400 megabytes per second, which a modern tape drive does easily once it gets going.

12.2.3 PLAIN - a worked example

1. The standard IBM punched card, in its final form from 1928, has 80 columns.
2. One column holds one character. So one card holds 80 characters, that is 80 bytes.
3. A standard box holds 2,000 cards. That is 160,000 bytes, about 160 kB.
4. A single modern phone photo of 4 MB would need 25 boxes of cards.
5. Stacked, those 50,000 cards stand about nine metres tall and weigh roughly 130 kilograms.
6. Now the modern end. An LTO-9 cartridge holds 18 TB of raw data in a plastic box the size of a thick paperback.
7. That is 18,000,000 MB, or about 4.5 million phone photos, or 225 million punched cards.

12.2.4 PLAIN - what is really happening inside

1. On tape and on disk the recording trick is identical. A coil of wire carries a current, the current makes a magnetic field, and the field lines up the magnetic particles in the coating underneath.
2. Reverse the current and the particles line up the other way. Two directions, two symbols, one bit.
3. Nothing holds them there. They stay because the coating is chosen to be hard to demagnetize once set.
4. Core memory is the same physics with a different shape. Each ring can be magnetized clockwise or anticlockwise. Wires threaded through the ring set it and sense it.
5. Reading a core is destructive: to find out which way it was, you force it one way and see whether it flipped. If it flipped, it was the other way. The machine must then write the value back.
6. Every core in early machines was threaded by hand, by workers with needles and microscopes. Memory was literally woven.
7. A drum stores data on the outside of a spinning cylinder, with one fixed head per track. No head has to move, so the only wait is for the drum to come round. That made drums fast for their day and small in capacity.

12.2.5 TECHNICAL - the engineer's version

1. Gustav Tauschek patented magnetic drum memory in Austria in 1932. His drum held about 500,000 bits, roughly 62.5 kB.
2. Jay Forrester at MIT developed practical coincident-current magnetic core memory around 1949 to 1953; the Whirlwind I machine received core in 1953. An Wang and Way-Dong Woo filed related work in 1949.
3. The UNIVAC I UNISERVO of 1951 was the first tape drive on a commercial computer. Its tape was nickel-plated phosphor bronze, 128 characters per inch, about 12,800 characters per second.
4. IBM's 726 tape unit of 1952 moved the industry to plastic tape at 100 bits per inch and about 7,500 characters per second.

5. Floppy disk lineage: IBM 23FD, 8-inch and read-only, 1971, about 80 kB. IBM 33FD, 8-inch read/write, 1973, 242.9 kB. Shugart SA400, 5.25-inch, 1976, about 110 kB. Sony's 3.5-inch design from 1981 settled at 720 kB double-density and 1.44 MB high-density from 1987.

Format	Year	Capacity
8-inch, single density	1973	242.9 kB
5.25-inch DD	1978	360 kB
3.5-inch DD	1983	720 kB
3.5-inch HD	1987	1.44 MB
3.5-inch ED	1987	2.88 MB
Iomega Zip	1994	100 MB

6. Linear Tape-Open (LTO) is the surviving tape standard, an open format run by HP, IBM and Quantum. It is a genuine multi-vendor standard, not one company's product.

Generation	Year	Native capacity
LTO-5	2010	1.5 TB
LTO-6	2012	2.5 TB
LTO-7	2015	6 TB
LTO-8	2017	12 TB
LTO-9	2021	18 TB
LTO-10	2025	30 TB, later 40 TB

7. LTO-9 and LTO-10 both run at 400 MB/s native. Marketing figures of 45 TB and 75 TB assume 2.5 to 1 hardware compression, which real data rarely reaches. Compressed figures are a marketing claim; native figures are the standard.
8. LTO-10 was announced on 13 August 2025 at 30 TB. In November 2025 the program raised it to 40 TB using an aramid-based tape substrate, and simultaneously cut its long-range roadmap: LTO-11 to 70 TB, LTO-12 to 120 TB, LTO-13 to 210 TB, LTO-14 to 365 TB.
9. Tape survives because of cost, air gap and shelf life. Media runs roughly 6 to 10 USD per TB against 21 to 32 USD for disk, a cartridge on a shelf cannot be reached by ransomware, and vendors rate media for about 30 years. The catch is the drive, which costs several thousand US dollars, so tape only pays off above roughly 100 TB of archive.
10. Tools: `mt` and `mt-st` to control a tape device, `tar` and `LTFS` to write it. LTFS, standardized as ISO/IEC 20919, makes a tape look like a filesystem.

12.2.6 WORDS - remember these

1. Sequential access — you must go past everything in between — media where access time is proportional to distance along the medium.
2. Random access — you can jump straight there — access time roughly independent of the previous position.
3. Core memory — memory woven from magnetic rings — coincident-current ferrite-core store, destructive read with write-back.
4. LTO — the modern tape standard — Linear Tape-Open, open multi-vendor format, generation 10 at 30 to 40 TB native.
5. Air gap — the archive is physically unplugged — offline copy unreachable by any network attacker.
6. Native capacity — the honest number — raw bytes stored with no compression assumed.

12.3 The hard disk drive, in full

12.3.1 PLAIN - in simple words

1. A hard disk is a stack of smooth metal or glass discs that spin very fast.
2. Each disc is coated with a magnetic film. That film is where the bits live.
3. A small arm swings across the discs, like the arm of a record player.
4. On the tip of the arm sits a **head**, which both writes and reads.
5. To write, the head makes a tiny magnetic field that flips a patch of the coating to point one way or the other.
6. To read, the head senses those patches as they rush past underneath.
7. The head never touches the disc. It flies above it on a cushion of air that the spinning disc drags along with it.
8. The gap is a few nanometres. That is smaller than a virus, thousands of times thinner than a hair.
9. If the head touches the disc at full speed, it gouges the coating. That is a head crash, and the data in that path is gone forever.

12.3.2 PLAIN - a picture in your head

1. Imagine a jumbo jet flying at full cruising speed, but only one millimetre above the ground, counting blades of grass as it goes.
2. Now scale that picture down to the size of a coin, and you have a disk head.
3. The head is a small block, roughly the size of a grain of salt cut in four.
4. The disc under it moves at around 100 kilometres per hour at the outer edge.
5. The arm can swing from the middle to the edge and stop on a chosen track in about eight thousandths of a second.
6. The whole thing is sealed, because a single speck of dust is a boulder at that height.

Where this comparison breaks: an aircraft is held up by its own wings and its own engines, while a disk head is a passive glider held up entirely by air the disc drags along. Slow the disc and the head lands. And the head is not merely looking at the ground, it is also repainting it as it goes.

12.3.3 PLAIN - a worked example

1. Take a common 7,200 RPM desktop drive. RPM means revolutions per minute.
2. 7,200 revolutions per minute is 120 revolutions per second.
3. One revolution therefore takes 1 divided by 120, which is 8.33 milliseconds.
4. Suppose the data you want is on the track under the head already. On average you wait half a turn for it to come round: 4.17 milliseconds.
5. Now suppose it is on a different track. The arm must move. That takes about 8.5 milliseconds on average for a desktop drive.
6. Total wait: about 12.7 milliseconds before a single byte arrives.
7. In those 12.7 milliseconds a modern CPU at 4 GHz executes roughly 50 million clock cycles. The processor is waiting an eternity.

12.3.4 PLAIN - what is really happening inside

1. The magnetic film is made of grains. Each grain is a tiny permanent magnet with a preferred direction it likes to point.
2. A group of grains all pointing the same way is a **magnetic domain**. One recorded bit uses tens of grains, not one. Using many grains averages out the noise so the read-back is reliable.
3. Writing: current through a coil in the head makes a field at a narrow gap. The field is strong enough to force the grains under it to flip direction.
4. Reading: here is the part that surprises people. The head does not sense “north” or “south”. It senses **changes**.

5. As the disc moves, wherever the magnetization reverses, there is a small stray field. The read sensor detects those reversals.
6. So the drive does not store 1 as north and 0 as south. It stores a stream of reversals, and an encoding scheme turns reversal patterns into bits.
7. That is why a long run of identical bits is a problem: no reversals means nothing to sense, and the drive loses its place. Encodings deliberately guarantee frequent reversals.
8. The head assembly is moved by a **voice coil**: a coil of wire sitting in a strong permanent magnet, exactly like the motor in a loudspeaker. Push current through it and the arm swings. Reverse it and the arm swings back.
9. The disc stack is turned by a **spindle motor** running on a fluid bearing, which is quieter and more precise than balls.

12.3.5 TECHNICAL - the engineer's version

1. The IBM 350 Disk Storage Unit, part of the IBM 305 RAMAC, was announced on 14 September 1956. It held 5 million six-bit characters across fifty 24-inch discs, with an average access time of 600 ms. It weighed over a ton and rented for about 3,200 US dollars a month.
2. In 1956 that is roughly 3.75 MB in modern eight-bit bytes, in a cabinet the size of two large refrigerators, delivered by cargo aircraft and moved with forklifts.
3. Flying height fell from about 20 micrometres in 1956 to roughly 1 to 3 nanometres today. Head sliders shrank through "pico" (1.25 by 1.0 by 0.3 mm) to "femto" (0.85 by 0.7 by 0.23 mm) classes.
4. The read sensor changed three times. Inductive coils gave way to anisotropic magnetoresistance, then to **giant magnetoresistance** (GMR), discovered independently in 1988 by Albert Fert in France and Peter Grunberg in Germany. IBM shipped the first GMR-head drive, the Deskstar 16GP, in 1997. Fert and Grunberg shared the 2007 Nobel Prize in Physics for the discovery. Modern heads use tunnelling magnetoresistance, a further refinement.
5. GMR is a genuine case of a physics discovery reaching a shipping product in nine years, and it is why disk capacity grew so fast in the late 1990s.
6. Recording orientation changed too. **Longitudinal** recording laid magnets flat, in the plane of the disc. **Perpendicular magnetic recording** (PMR) stands them on end, packing them tighter and resisting thermal decay. Toshiba shipped the first PMR drive in 2005 and Seagate followed in 2006. All modern drives are perpendicular.
7. **Shingled magnetic recording** (SMR) overlaps write tracks like roof tiles, because a write head is wider than a read head. It buys about 20 percent capacity at a severe cost: rewriting one track means rewriting the whole band. Drive-managed SMR hides this and can stall for seconds under sustained random writes. Western Digital shipping undisclosed SMR in Red NAS drives caused a public dispute in 2020, and the industry now labels it.
8. **Heat-assisted magnetic recording** (HAMR) puts a laser in the head. It heats a spot to roughly 400 to 450 degrees Celsius for about a nanosecond, which briefly makes a very stable, very small grain writable. Seagate's Mozaic 3+ platform shipped from 2024, and Mozaic 4+ began shipping 44 TB drives with ten platters at over 4 TB each in March 2026. Western Digital pursued microwave-assisted recording (MAMR) instead.

Parameter	1956 IBM 350	2026 HAMR drive
Capacity	about 3.75 MB	44 TB
Platters	50	10
Platter diameter	24 inch	3.5 inch
Access time	600 ms	about 8 ms

9. Drives above about 8 TB are usually sealed and filled with helium, first shipped by HGST as the Ultrastar He6 in 2013. Helium is one seventh the density of air, so there is less turbulence and less drag,

which allows more platters in the same height and cuts power.
10. Areal density is the headline metric, in bits per square inch. Production drives are now in the region of 1.5 to 2 terabits per square inch, and HAMR is the path beyond.
11. Tools: `smartctl -a /dev/sda` reads SMART attributes including reallocated sector count and head flying-height sensors on some models.

12.3.6 WORDS - remember these

1. Platter — one of the spinning discs — rigid aluminium or glass substrate with a sputtered magnetic film.
2. Head — the part that reads and writes — the slider carrying a write coil and a magnetoresistive read sensor.
3. Air bearing — the cushion the head flies on — the thin viscous film of gas dragged by the rotating platter, a few nanometres thick.
4. Magnetic domain — a patch of magnets all pointing the same way — a region of aligned grains representing part of one recorded symbol.
5. Voice coil actuator — the loudspeaker motor that swings the arm — rotary VCM under closed-loop servo control.
6. GMR — the effect that made big disks possible — giant magnetoresistance, Fert and Grunberg 1988, Nobel Prize 2007.
7. HAMR — heat the spot before writing it — heat-assisted magnetic recording, using a near-field laser to lower coercivity momentarily.
8. SMR — tracks overlapped like roof tiles — shingled magnetic recording, requiring band rewrites, sold as drive-managed or host-managed.

12.4 HDD geometry and performance

12.4.1 PLAIN - in simple words

1. The surface of a disc is divided into rings called **tracks**.
2. Each track is chopped into short arcs called **sectors**. A sector is the smallest chunk the drive will read or write.
3. A sector used to hold 512 bytes. Modern drives use 4,096 bytes.
4. There are several platters stacked up, each with a surface on top and one underneath, each with its own head.
5. All the heads move together on one arm. So track 500 on every surface is reachable without moving the arm. That set of tracks is a **cylinder**.
6. To reach data the drive does three things: move the arm to the right track, wait for the right sector to come round, then transfer the data.
7. Moving the arm is **seek time**. Waiting for rotation is **rotational latency**. Only the third part actually moves data.
8. If your next request is right next to the last one, you skip the first two steps. That is why reading a file in order is fast.
9. If your next request is somewhere random, you pay both waits every time. That is why random access on a hard disk is brutally slow.

12.4.2 PLAIN - a picture in your head

1. Picture a very large car park laid out in concentric rings, with numbered bays around each ring.
2. You are on a moped in the middle. Someone radios you a bay number.
3. First you ride out to the right ring. That is the seek.
4. But the car park is on a turntable, spinning. You must wait for your bay to come round to you. That is the rotational latency.

5. Then you read the number plate as it passes. That is the transfer.
6. If the next twenty bays you need are the next twenty in the same ring, you just sit still and read them all as they go by. Wonderful.
7. If each next bay is in a random ring, you spend all day riding and waiting and almost no time reading.

Where this comparison breaks: a real drive also reads ahead, caches, and reorders queued requests to reduce total travel, so a clever drive is better than a moped rider taking orders one at a time. And the drive's own controller hides a lot of this, so the outside world never sees the real geometry.

12.4.3 PLAIN - a worked example

1. Rotational latency is pure arithmetic. Average latency is the time for half a revolution.
2. Formula: average rotational latency in milliseconds equals 30,000 divided by the RPM.
3. At 5,400 RPM: 30,000 divided by 5,400 equals 5.56 ms.
4. At 7,200 RPM: 30,000 divided by 7,200 equals 4.17 ms.
5. At 10,000 RPM: 3.00 ms. At 15,000 RPM: 2.00 ms.
6. Now add a typical average seek. Take a 7,200 RPM drive with an 8.5 ms seek.
7. Average access time is 8.5 plus 4.17, about 12.7 ms.
8. Operations per second is 1 divided by 0.0127, about 79.
9. So a 7,200 RPM drive does roughly 80 random operations per second. That number has barely changed since 1995.

```
One random read on a 7200 RPM drive
seek 8.5 ms          rotate 4.17 ms      transfer 0.04 ms
|=================| |==========|         |
0                 8.5         12.7      12.74 ms
                                         ^ 4 KB actually moved here
99.7 percent of the time is waiting, not reading.
```

12.4.4 PLAIN - what is really happening inside

1. Old drives were addressed by **cylinder, head, sector**: tell the drive which ring, which surface, which arc. This is CHS addressing.
2. That worked while software could know the real geometry. It stopped working the moment drives started lying about their shape.
3. Drives had to lie because of a physical fact: outer tracks are longer than inner tracks. A fixed number of sectors per track wastes the outside.
4. So drives use **zoned recording**: more sectors per track near the edge, fewer near the middle. Geometry is no longer uniform, so CHS is fiction.
5. The fix is **logical block addressing**, LBA. The drive presents a simple numbered list of sectors, 0, 1, 2, and upward. The drive alone knows where block 4,000,000 physically is.
6. Zoned recording has a visible side effect: the outer tracks pass under the head faster, so they transfer more bytes per second. A drive is genuinely faster at the start of its address range than at the end, often by half.
7. Sectors are not just data. Each carries a preamble to synchronize timing, a servo pattern telling the head where it is, the data itself, and an error correcting code.
8. Moving from 512-byte to 4,096-byte sectors saves all that overhead per byte and allows a stronger error code. That is **Advanced Format**.

12.4.5 TECHNICAL - the engineer's version

1. CHS addressing in the PC BIOS interrupt 13h interface allowed 1,024 cylinders, 256 heads and 63 sectors, capping capacity at 8.4 GB with 512-byte sectors. Various translation hacks pushed it further before LBA replaced it outright.

2. LBA-28 in ATA capped out at 2^28 sectors of 512 bytes, that is 128 GiB, usually quoted as 137 GB. LBA-48, introduced in ATA-6 in 2003, raises the cap to 2^48 sectors, about 128 PiB.
3. Advanced Format was defined by IDEMA and the industry transitioned from around January 2011. Two variants exist: 512e presents 512-byte logical sectors over 4,096-byte physical ones, and 4Kn presents 4,096 bytes both ways.
4. On a 512e drive, a misaligned 4 KB write triggers a read-modify-write of the physical sector, roughly halving small random write performance. Aligning partitions to a 1 MiB boundary, now the default in every modern partitioner, avoids this.
5. Sustained transfer rate depends on linear density and rotation. A modern 7,200 RPM 20 TB drive sustains roughly 260 to 290 MB/s on outer tracks and about 120 to 140 MB/s on inner tracks. Seagate quotes about 300 MB/s for the 44 TB Mozaic 4+ drives.

Drive class	Rot. latency	Typical random IOPS
5,400 RPM laptop / NAS	5.56 ms	50 to 75
7,200 RPM desktop	4.17 ms	75 to 100
10,000 RPM enterprise	3.00 ms	125 to 150
15,000 RPM enterprise	2.00 ms	175 to 210

6. Native Command Queuing, part of SATA, lets a drive hold up to 32 outstanding commands and service them in an efficient order. It can raise random IOPS by 30 to 50 percent at high queue depth by reducing total head travel.
7. The gap between sequential and random is not subtle. A 20 TB drive doing 1 MB sequential reads delivers roughly 260 MB/s. The same drive doing 4 KB random reads at queue depth 1 delivers about 80 times 4 KB, that is roughly 0.32 MB/s. The ratio is about 800 to 1.
8. Measure it yourself:

```
# Sequential read throughput, 1 MB blocks
fio --name=seq --rw=read --bs=1M --size=4G \
    --filename=/dev/sdb --direct=1

# Random read IOPS, 4 KB blocks, one at a time
fio --name=rand --rw=randread --bs=4k --size=4G \
    --iodepth=1 --filename=/dev/sdb --direct=1
```

9. Inspect logical and physical sector size with `lsblk -o NAME,PHY-SEC,LOG-SEC` on Linux, or `fsutil fsinfo sectorinfo C:` on Windows.

12.4.6 WORDS - remember these

1. Track — one ring on the disc surface — a single concentric data path.
2. Sector — the smallest readable chunk — a physical block, 512 or 4,096 bytes plus servo, sync and ECC overhead.
3. Cylinder — the same track on every surface — the set of tracks reachable without moving the actuator.
4. Seek time — the wait while the arm moves — actuator settling time to a target track, quoted as an average over random seeks.
5. Rotational latency — the wait for the sector to come round — 30,000 divided by RPM in milliseconds, on average.
6. LBA — one long numbered list of blocks — logical block addressing, hiding real geometry behind a linear address space.
7. Advanced Format — 4K sectors instead of 512 — 512e or 4Kn physical sector formats defined by IDEMA.
8. IOPS — operations per second — input/output operations per second, always quoted with block size and queue depth or it is meaningless.

12.5 Flash memory, in full

12.5.1 PLAIN - in simple words

1. Flash memory stores a bit by trapping electrons inside an insulator, where they cannot get out on their own.
2. A cell is a transistor with an extra hidden pocket built into it.
3. Put electrons in the pocket and the transistor becomes harder to switch on.
4. Leave the pocket empty and it switches on easily.
5. To read a cell you apply a test voltage and see whether it switches on. That tells you whether the pocket is full.
6. To write, you shove electrons through the insulator into the pocket using a high voltage.
7. To erase, you shove them all back out again, also with a high voltage.
8. The insulator is what makes it non-volatile. Electrons stay put with no power. The insulator is also what wears out, because forcing charge through it damages it a little every single time.
9. That is the whole of flash memory. Everything else is engineering around those two sentences.

12.5.2 PLAIN - a picture in your head

1. Think of a water tank with no tap and no drain, sealed on all sides.
2. Normally water cannot get in or out. That is your data sitting safe.
3. To fill it you use enormous pressure to force water straight through the wall of the tank. To empty it you do the same in reverse.
4. It works, but every time you do it the wall gets a little more damaged.
5. After enough fills and empties the wall starts to seep. Now the tank slowly loses water on its own, and a tank you filled last year no longer reads full.
6. That is exactly why flash memory has a limited number of writes, and why an old worn drive also forgets faster when left unplugged.
7. Now the reading part. You do not open the tank to check it. You judge the level from outside by how much it resists a push.
8. If the tank holds only "empty" or "full", that judgement is easy. If you try to distinguish sixteen different water levels, it gets very hard, and small leaks matter enormously.

Where this comparison breaks: electrons are not water and do not slosh. The "pressure" is a quantum effect called tunnelling, where an electron simply appears on the far side of a thin barrier. And the damage is not a hole in a wall, it is charge getting stuck inside the insulating oxide itself.

12.5.3 PLAIN - a worked example

1. Cells are cheaper if you store more than one bit each, by using more charge levels. Here is what that costs you.

Type	Bits per cell	Voltage levels
SLC	1	2
MLC	2	4
TLC	3	8
QLC	4	16
PLC	5	32

2. Going from SLC to QLC gives you four times the capacity from the same silicon. That is why every consumer drive is TLC or QLC.
3. But the voltage window is fixed. With 16 levels in the same window, each level is a narrow band, and a small charge leak pushes a cell into the neighbouring band and corrupts the value.

4. So endurance and speed collapse as bits per cell rise.

Type	P/E cycles	Relative write speed
SLC	50,000 to 100,000	fastest
MLC	3,000 to 10,000	about half SLC
TLC	1,000 to 3,000	about a third
QLC	100 to 1,000	about a fifth
PLC	under 100, projected	not yet in products

5. PLC is active research, not a shipping product as of 2026. Treat any PLC capacity claim as a roadmap, not a specification.

12.5.4 PLAIN - what is really happening inside

1. Cells are wired into a grid. Rows are called **word lines**, columns are called **bit lines**.
2. In **NOR flash** every cell hangs directly off a bit line, in parallel. You can read any single byte instantly, but each cell needs its own contact, so cells are large and the chip is expensive per byte.
3. In **NAND flash** the cells are wired in long series strings, like old Christmas lights. Far fewer contacts are needed, so cells pack much tighter.
4. The price of that is that you cannot touch one cell alone. To read one cell you must switch every other cell in its string fully on so current can pass through them.
5. So NAND works on whole rows at a time. A row of cells read together is a **page**. Pages are grouped into a **block**. Blocks are grouped into **planes**, and planes into a **die**.
6. Now the crucial asymmetry, the fact that shapes every SSD ever built:
    1. You can read one page.
    2. You can write one page, but only if it is currently erased.
    3. You cannot erase one page. You can only erase a whole block.
7. Writing pushes cells one direction only. Erasing is the only way back, and erasing is coarse.
8. So changing one byte in a page is impossible in place. The drive must write the new version of that page somewhere else entirely, and remember where.
9. That single restriction is why an SSD needs the complicated software we meet in section 12.6.
10. **3D NAND** is the other big idea. Instead of shrinking cells sideways, which stopped working around 2013, manufacturers turned the strings on end and stacked layers vertically, then drilled holes through the stack.
11. This let them go back to a relaxed, reliable cell size and gain density by height instead. It is the single reason flash kept getting cheaper.

12.5.5 TECHNICAL - the engineer's version

1. The floating-gate MOSFET was invented by Dawon Kahng and Simon Min Sze at Bell Labs in 1967. It is an ordinary MOS transistor with an electrically isolated polysilicon gate buried in the oxide between control gate and channel.
2. Fujio Masuoka at Toshiba invented flash memory around 1980. NOR flash was presented in 1984 and Intel shipped the first commercial NOR part in 1988. NAND flash was presented at the IEEE International Electron Devices Meeting in San Francisco in 1987. The name came from a colleague who thought the block erase resembled a camera flash.
3. Charge is moved by Fowler-Nordheim tunnelling for erase and either tunnelling or channel hot-electron injection for program, at internally generated voltages of roughly 15 to 20 V produced by on-chip charge pumps.
4. Trapped charge shifts the transistor threshold voltage. Reading is a sequence of sense operations at different word-line reference voltages; a TLC read needs seven reference points to separate eight states.

5. **Charge trap flash** replaced the conducting floating gate with an insulating silicon nitride layer that traps charge in discrete defect sites. Because the storage layer is not conductive, a single defect does not drain the whole cell, which is what makes tall 3D stacks practical. Samsung V-NAND has used charge trap from the start; Intel and Micron used a floating-gate 3D design for several generations before moving across.
6. Modern 3D NAND geometry, typical of 2024 to 2026 TLC parts:

Unit	Typical size	Operation time
Page	16 KiB	read 50 to 90 us
Page program	16 KiB	500 to 2,500 us
Block	1,000 to 2,000 pages	erase 2 to 10 ms
Die (LUN)	512 Gbit to 2 Tbit	4 to 6 planes

7. Note the ratio: erasing a block costs roughly one hundred times a page read, and it destroys everything in that block. This is the reason garbage collection exists.
8. Layer counts as of 2026: SK hynix is in mainstream production at 321 layers, Samsung's V9 generation is in the high 200s, Micron's G9 is around 276, and Kioxia and SanDisk's BiCS8 is 218. Samsung announced a 400-layer part for 2026 and demonstrated a 900-layer research die in May 2026 by bonding two 450-layer stacks. Stacks above roughly 128 layers are almost always built as two or more bonded decks, a technique called string stacking.
9. NOR flash did not die. It survives wherever code must be executed directly from the chip with no controller in the way: BIOS and UEFI firmware, boot ROMs, microcontrollers and automotive parts. NOR gives byte-level random read and execute-in-place; NAND gives density and cost per bit. NAND won storage, NOR kept firmware.
10. Inspect real flash geometry on Linux with `nvme id-ctrl /dev/nvme0` and `nvme id-ns /dev/nvme0n1`, which report namespace sizes and, on drives that support it, optimal write size and NPWG or NPWA alignment hints.

12.5.6 WORDS - remember these

1. Floating gate — the hidden pocket that holds electrons — an isolated polysilicon gate whose stored charge shifts the transistor threshold.
2. Charge trap — the same job done by an insulator — a silicon nitride layer trapping charge in defect sites, standard in 3D NAND.
3. Tunnelling — electrons crossing a barrier they should not — Fowler-Nordheim tunnelling through the tunnel oxide under a high field.
4. Page — the smallest thing you can read or write — typically 16 KiB, plus spare area for error correction.
5. Block — the smallest thing you can erase — one to two thousand pages, several megabytes.
6. SLC, MLC, TLC, QLC — how many bits crammed into one cell — 1, 2, 3 and 4 bits, using 2, 4, 8 and 16 threshold levels.
7. P/E cycle — one wear event — a complete program and erase of a block, the unit in which flash lifetime is counted.
8. 3D NAND — stacking cells upward instead of shrinking them — vertical strings through a multi-layer stack, now over 300 layers.

12.6 The SSD controller and the flash translation layer

12.6.1 PLAIN - in simple words

1. An SSD is not just flash chips in a box. It contains a small computer whose only job is to manage those chips.

2. That computer has its own processor cores, its own firmware and often its own memory.
3. It exists because of the asymmetry in the last section: the outside world wants to overwrite any block at any time, and flash cannot do that.
4. So the controller lies, politely and consistently. It tells your operating system "here is a simple numbered list of blocks you can overwrite freely".
5. Underneath, it never overwrites anything. It writes each new version to a fresh place and updates a private map.
6. That map is the **flash translation layer**, or FTL. It converts the address your computer asked for into the physical page where the data really is.
7. The controller must also spread wear evenly, clean up rubbish in the background, and hide all of it from you.
8. Almost everything odd about SSD behaviour comes from that hidden work.

12.6.2 PLAIN - a picture in your head

1. Picture a large notebook where you are not allowed to use an eraser on one line. You can only tear out and rewrite a whole page at a time.
2. So when you want to change one line, you write the new version on the next blank line instead, and you keep an index at the front saying which line is the current one.
3. The old line is now rubbish, but it is still sitting there taking space.
4. Eventually the notebook fills with a mixture of current lines and rubbish lines scattered across every page.
5. To get space back you pick a page that is mostly rubbish, copy its few surviving lines onto a fresh page, update the index, and tear out the old page.
6. That tidying is **garbage collection**. It costs real work that nobody asked for, and it is why an SSD writes more than you told it to.
7. And you deliberately spread your writing across all pages rather than always using the front ones, so no page wears out first. That is **wear levelling**.

Where this comparison breaks: a real controller does this tidying while you are also writing, competing for the same chips, which is why performance can dip under sustained load. And the index itself must be kept safely, because losing it loses the whole drive, not one page.

12.6.3 PLAIN - a worked example

1. Here is **write amplification** with real arithmetic.
2. Take a block of 256 pages. 192 pages hold live data, 64 hold stale data.
3. You want to reclaim that block. First you must copy the 192 live pages elsewhere, then erase the block.
4. So to free 64 pages of space, the drive performed 192 page writes.
5. Now suppose your application writes 64 pages of new data into that freed space. Total flash writes: 192 copied plus 64 new equals 256.
6. Host writes: 64 pages. Flash writes: 256 pages.
7. Write amplification factor equals 256 divided by 64, which is 4.
8. Your drive is wearing out four times faster than your data volume suggests.
9. Now repeat the calculation on a drive that is only half full, where a typical block has 64 live pages and 192 stale.
10. Copy 64 live pages, free 192, then write 192 new. Flash writes 64 plus 192 equals 256, host writes 192, amplification is 1.33.
11. That is the whole explanation of why a full SSD is slow and short-lived and an empty one is fast. Free space is not idle. It is working capital.

12.6.4 PLAIN - what is really happening inside

1. The FTL keeps a table. Logical block number in, physical page location out.
2. Every write allocates a fresh page, writes there, and updates one table entry. Nothing is ever changed in place.
3. **Wear levelling** comes in two kinds. Dynamic levelling spreads new writes over blocks that are already free. Static levelling also moves data that has sat untouched for a long time, so that cold blocks take their share of wear. Without static levelling, a drive holding a large read-only archive would wear out the small remaining area very fast.
4. **Over-provisioning** is hidden spare capacity the controller keeps for itself. It never appears in your capacity figure. More spare space means garbage collection always has somewhere easy to work, which lowers write amplification.
5. **TRIM** solves a problem you would not guess at. When you delete a file, the filesystem just marks its blocks free in its own records. It does not tell the drive.
6. So the drive still believes every one of those pages holds live data, and dutifully copies them during garbage collection forever.
7. TRIM is a command that says "these logical blocks no longer contain anything you need to preserve". Now the controller can drop them, and the pages become free for nothing.
8. Without TRIM, an SSD gradually behaves as if it were completely full, even when the filesystem shows it as empty.
9. **SLC caching** is why big copies slow down. TLC and QLC cells can be operated in one-bit mode, which is much faster. Controllers keep part of the flash in that mode as a fast landing zone.
10. Incoming writes land in the fast zone at full speed. Later, when the drive is idle, they are folded down into dense multi-bit storage.
11. If you write more than the cache holds, the cache runs out mid-copy and writes must go straight to slow storage, while folding also competes for the same chips. Speed drops sharply and visibly.
12. The FTL table is far too large to search through, so the controller keeps it in fast memory. Either a dedicated DRAM chip on the drive, or a slice of your computer's own memory borrowed over the interface.

12.6.5 TECHNICAL - the engineer's version

1. Typical controllers are multi-core embedded designs. Phison E26 and E28, Silicon Motion SM2508, and Samsung's in-house controllers use two to five ARM Cortex-R class cores plus hardware engines for ECC, encryption and DMA.
2. Error correction is not optional. Modern TLC and QLC require LDPC (low-density parity-check) decoding with soft-decision reads. Raw bit error rates in the 10^-3 range are corrected down to below 10^-15.
3. Mapping granularity drives the DRAM requirement. A page-mapped FTL with 4 KiB granularity and a 4-byte entry needs about 1 GB of DRAM per 1 TB of NAND. That is the origin of the industry rule of thumb.
4. **Host Memory Buffer** (HMB) is an NVMe feature that lets a DRAM-less drive borrow host RAM, typically 32 to 64 MB. It holds only a hot fraction of the map, so DRAM-less drives are competitive on light desktop work and clearly worse on sustained random writes.
5. Over-provisioning has two sources. First, the decimal versus binary gap: a drive sold as 1 TB (10^12 bytes) built from 1 TiB (1.0995 times 10^12 bytes) of NAND has 7.37 percent spare for free. Enterprise parts add more deliberately, which is why capacities read 960 GB, 1,920 GB or 3,840 GB instead of round powers.

Class	Usable / raw	Over-provision
Consumer 1 TB	1,000 / 1,099 GB	about 7 percent

Class	Usable / raw	Over-provision
Enterprise 960 GB	960 / 1,099 GB	about 13 percent
Enterprise 800 GB	800 / 1,099 GB	about 27 percent

6. TRIM is the ATA `DATA SET MANAGEMENT` command with the deallocate bit; the SCSI equivalent is `UNMAP`, and NVMe uses `Dataset Management` with the deallocate attribute. Windows 7 (2009) was the first mainstream OS to issue it, Linux gained support in kernel 2.6.33 (2010), and macOS enabled it for Apple SSDs from 10.6.8 (2011).
7. Two ways to issue it on Linux: continuous with the `discard` mount option, or batched on a timer, which is now the preferred default.

```
# See whether the device reports discard support
lsblk -D            # DISC-GRAN and DISC-MAX columns

# Run a batched trim now, and report bytes released
sudo fstrim -av

# Check the systemd timer that normally does this weekly
systemctl status fstrim.timer
```

8. SLC cache sizing is an implementation detail that varies per model. A typical 2 TB consumer QLC drive might sustain 5,000 MB/s for the first 150 to 250 GB and then fall to 100 to 450 MB/s. TLC drives usually fall to 900 to 1,700 MB/s instead, which is far less painful. Reviewers call this the post-cache or steady-state write speed, and it is the number that matters for video work and large backups.
9. **Power-loss protection** in enterprise drives is a bank of tantalum or ceramic capacitors sized to flush the volatile write buffer and the mapping table to NAND after supply is cut. Consumer drives almost never have it. Without PLP, a drive may honestly acknowledge a write that is still in volatile buffer, which is why databases insist on explicit cache flushes.
10. Observe the drive's own accounting with `nvme smart-log /dev/nvme0`, which reports `data_units_written`, `percentage_used` and `unsafe_shutdowns`. On drives exposing vendor logs, `smartctl -a` may also show host writes versus NAND writes, from which you can compute a real write amplification factor.

12.6.6 WORDS - remember these

1. FTL — the private map from your addresses to real chips — flash translation layer, maintaining logical-to-physical mapping and block state.
2. Wear levelling — spread the damage evenly — dynamic and static algorithms equalizing program/erase counts across blocks.
3. Garbage collection — tidying up to reclaim erased blocks — relocating valid pages out of victim blocks before erasing them.
4. Write amplification — the drive writes more than you asked — WAF, the ratio of NAND writes to host writes.
5. Over-provisioning — hidden spare room — reserved capacity invisible to the host, lowering WAF and raising sustained performance.
6. TRIM — telling the drive a block is now rubbish — the deallocate hint, ATA DATA SET MANAGEMENT, SCSI UNMAP, NVMe Dataset Management.
7. SLC cache — a fast landing strip made of slow flash — a region operated in pseudo-SLC mode, later folded into TLC or QLC.
8. HMB — the drive borrows your computer's memory — Host Memory Buffer, an NVMe feature used by DRAM-less controllers.
9. Power-loss protection — capacitors that finish the job — onboard energy reserve flushing volatile buffers on supply loss.

12.7 SSD endurance and failure

12.7.1 PLAIN - in simple words

1. Every flash block can only be erased and rewritten a limited number of times.
2. When a block's insulator is too damaged to hold charge reliably, the controller retires it and uses a spare.
3. When the spares run out, the drive stops accepting writes.
4. Manufacturers publish this as **terabytes written**, or TBW: how much data you may write over the warranty period.
5. Reading is nearly free, but not entirely. Reading a page slightly disturbs its neighbours, so after a great many reads a block must be refreshed.
6. A flash cell also leaks charge slowly. An unpowered SSD gradually forgets, over months to years, and forgets faster if it is worn or hot.
7. Hard drives usually die gradually and noisily. SSDs usually die suddenly and silently.
8. That difference matters more for your backup plan than any speed figure in this chapter.

12.7.2 PLAIN - a picture in your head

1. Think of a hard drive as an old car with 300,000 kilometres on it. It squeaks, it leaks, warning lights come on, one cylinder misfires. You get weeks of notice that something is wrong.
2. Think of an SSD as a light bulb. It works perfectly, perfectly, perfectly, and then it does not work at all, with no warning of any kind.
3. Most hard drive failures leave the platters intact and let a recovery lab read the data off with a new head assembly.
4. Most SSD failures are controller failures. The flash chips are fine and completely unreadable, because only that controller knew the map.

Where this comparison breaks: plenty of hard drives also die instantly, from a seized motor or a snapped head, and plenty of SSDs fail gracefully by dropping into a read-only mode that lets you copy everything off. These are tendencies, not laws.

12.7.3 PLAIN - a worked example

1. Is a consumer SSD's endurance rating actually a problem for you? Let us find out with arithmetic.
2. A 2 TB Samsung 990 Pro is rated at 1,200 TBW over a five-year warranty.
3. 1,200 TB divided by 5 years is 240 TB per year.
4. 240 TB divided by 365 days is about 658 GB per day, every single day, for five years.
5. Typical desktop use is 10 to 40 GB per day. At 30 GB per day you would need 1,200,000 divided by 30, that is 40,000 days, about 109 years.
6. So for ordinary use, endurance is not the thing that will kill your drive. Controller failure, firmware bugs and power events are far more likely.
7. Now the other case. A database server writing 2 TB per day reaches 1,200 TBW in 600 days, under two years. This is exactly why enterprise drives are rated in drive writes per day instead.
8. Sanity check the rating from first principles. TBW is roughly capacity times P/E cycles divided by write amplification. 2 TB times 1,500 cycles divided by 2.5 gives 1,200 TB. The published figure is consistent with TLC flash of around 1,500 cycles at a write amplification of about 2.5.

12.7.4 PLAIN - what is really happening inside

1. Wear is not uniform damage across a chip. It is charge getting trapped inside the tunnel oxide, which shifts and blurs the voltage levels.
2. As levels blur, the error correction has to work harder. Reads take longer because the controller must retry at different reference voltages.
3. A worn drive therefore gets slower before it fails, especially on reads of old data.

4. **Read disturb**: reading one page requires putting a raised voltage on every other word line in the string. That nudges charge into neighbouring cells. After roughly one hundred thousand to a million reads of one block, the controller must copy the block elsewhere and erase it.
5. **Retention**: a powered drive can refresh weak blocks in the background. An unpowered drive cannot. The clock is running on every cell.
6. Retention halves roughly for every 10 degrees Celsius of extra storage temperature, and falls sharply once the drive is near its rated cycle count.
7. So a brand new SSD in a cool drawer will hold data for many years. A heavily worn SSD in a hot loft may not survive one summer.
8. If you use SSDs for cold archives, power them on and read them through once or twice a year. Or use disks or tape, which do not have this failure mode.

12.7.5 TECHNICAL - the engineer's version

1. JEDEC standard JESD218 defines the endurance and retention test method. Client SSDs must retain data for 1 year at 30 degrees Celsius after reaching rated endurance; enterprise SSDs must retain for 3 months at 40 degrees. Both are end-of-life figures, not new-drive figures.
2. Client drives are rated in TBW; enterprise drives in DWPD, drive writes per day, over a stated warranty term. Conversion: TBW equals DWPD times capacity in TB times 365 times years.
3. A 1 DWPD read-intensive 3.84 TB drive over 5 years is 1 times 3.84 times 365 times 5, that is 7,008 TBW. A 3 DWPD mixed-use part of the same size is 21,024 TBW.
4. SMART attributes worth watching. On SATA SSDs: 5 Reallocated Sector Count, 177 Wear Leveling Count, 231 SSD Life Left, 241 Total LBAs Written. On NVMe: `percentage_used`, `available_spare`, `available_spare_threshold`, `media_errors`, `unsafe_shutdowns`. `percentage_used` can legitimately exceed 100, which means the drive has passed its rated endurance, not that it failed.
5. Backblaze's published quarterly drive statistics report hard drive annualized failure rates generally in the range of about 1.3 to 1.6 percent across their fleet, with SSD boot drives comparable at similar ages. The often-repeated claim that SSDs are simply more reliable than HDDs is not well supported by that data; the failure modes differ more than the rates do.
6. Google and Facebook field studies published in 2016 found that SSD uncorrectable error rates correlate more strongly with age and with the number of blocks already retired than with raw bytes written, and that early life has a distinct higher-error period. Experts disagree on how far those findings, drawn from older MLC fleets, transfer to modern TLC and QLC parts.

Metric	HDD 7,200 RPM	SATA SSD	NVMe Gen5 SSD
Read latency	5 to 15 ms	about 100 us	20 to 80 us
4K random read IOPS	75 to 100	90k to 100k	1.0M to 2.5M
Sequential read	150 to 300 MB/s	500 to 560 MB/s	12 to 15 GB/s
Price per TB, 2026	21 to 32 USD	from 96 USD	from 105 USD

7. Failure mode summary. HDD: gradual, dominated by media defects, reallocated sectors and head or motor wear, usually preceded by SMART warnings and often partially recoverable. SSD: bimodal, either slow wear-out ending in a read-only state, or abrupt controller or firmware failure with total loss and essentially no consumer recovery path.
8. Firmware bugs are a real and repeated cause of sudden SSD loss. Several vendors have shipped defects that brick drives at a specific power-on-hours count, notably a family of HPE SAS SSDs in 2019 and 2020 that failed at 32,768 and again at 40,000 hours. Check for firmware updates on drives you depend on.

12.7.6 WORDS - remember these

1. TBW — how much you may write before the warranty ends — terabytes written, the client endurance rating.

2. DWPD — how many times you may rewrite the whole drive daily — drive writes per day over a stated warranty term.
3. Read disturb — reading damages the neighbours a little — cumulative disturbance requiring periodic block refresh.
4. Retention — how long it remembers unplugged — data retention time, specified by JESD218 at end of rated life and temperature dependent.
5. Available spare — how many replacement blocks are left — the NVMe SMART field that actually predicts end of life.
6. Bimodal failure — either slow decline or instant death — the two distinct SSD failure populations, wear-out and controller fault.

12.8 How storage connects: interfaces and buses

12.8.1 PLAIN - in simple words

1. A drive is useless until it is wired to the computer. That wiring is an **interface**: a cable, a connector and a language.
2. The language matters as much as the cable. It sets how many requests can be in flight at once.
3. Old drives were slow, so the language only allowed one request at a time. Nobody minded, because the disk was the bottleneck anyway.
4. Then SSDs arrived and could serve thousands of requests at once, and the polite one-at-a-time language became the thing holding everything back.
5. So the industry wrote a new language, **NVMe**, designed for a device with no moving parts, and connected drives straight to the processor's own fast bus.
6. That bus is **PCI Express**, usually shortened to PCIe. It is the same bus a graphics card uses.
7. Everything else on this page is a variation on those two ideas: how many wires, and how good the language is.

12.8.2 PLAIN - a picture in your head

1. Picture a warehouse counter with one clerk who takes one order, fetches it, comes back, and only then takes the next order. That is the old ATA and SATA model with a spinning disk behind it.
2. Now put a thousand robots in the warehouse. The clerk is still taking one order at a time, so 999 robots stand idle.
3. NVMe replaces the clerk with thousands of order slots that customers fill in themselves, and the robots pick work out of the slots as they become free.
4. Each processor core gets its own order slot, so cores never queue behind each other.

Where this comparison breaks: real NVMe queues live in the host's memory and the drive fetches from them by direct memory access, so there is no clerk at all. And the drive can complete work out of order, which the picture of a queue does not capture.

12.8.3 PLAIN - a worked example

1. A PCIe **lane** is one pair of wires each way. Drives usually use four lanes, written x4.
2. Each generation roughly doubles the speed per lane.

Generation	Year	Per lane	x4 total
PCIe 3.0	2010	0.985 GB/s	3.9 GB/s
PCIe 4.0	2017	1.97 GB/s	7.9 GB/s
PCIe 5.0	2019	3.94 GB/s	15.8 GB/s
PCIe 6.0	2022	7.56 GB/s	30.2 GB/s

3. Compare that with SATA 3.0 at 6 gigabits per second. After encoding overhead that is about 550 to 560 MB/s in practice.
4. So a PCIe 5.0 x4 slot carries roughly 28 times what a SATA cable carries.
5. Yet a fast NVMe drive reaching 14 GB/s is only about 90 percent of what its Gen5 x4 link allows. The drive, not the bus, is now the limit again.

12.8.4 PLAIN - what is really happening inside

1. The oldest common interface was **IDE**, later renamed **PATA**: a wide 40-pin ribbon cable carrying 16 bits side by side, with two drives per cable configured as master and slave by jumper.
2. Parallel wiring failed as speeds rose, because 16 signals never arrive at exactly the same instant. That mismatch is called skew.
3. **SATA** solved it by sending one bit stream very fast down a single differential pair each way, with one drive per cable and no jumpers.
4. **SAS** is the enterprise cousin: same cabling idea, richer command set from SCSI, two ports per drive so two servers can reach it, and expanders so one controller can address thousands of drives.
5. NVMe threw away the disk-shaped assumptions entirely. There is no cable protocol to translate, no single command queue, and no controller chip sitting between drive and CPU.
6. Physically, an NVMe drive is usually an **M.2** stick: 22 mm wide, most often 80 mm long, hence the name 2280.
7. The notch cut into the M.2 edge connector is the **key**, and it prevents you fitting a card the socket cannot drive. An M-key socket carries PCIe x4; a B-key socket carries at most PCIe x2 or SATA.
8. In phones and cheap tablets the flash is soldered down and uses **eMMC** or **UFS** instead, which are simpler and lower powered.

12.8.5 TECHNICAL - the engineer's version

1. IDE was created by Western Digital with Compaq in 1986 and standardized as ATA-1 in 1994. Its fastest mode, Ultra DMA 133, reached 133 MB/s and required an 80-conductor cable with ground wires interleaved to control crosstalk.
2. SATA revision 1.0 arrived in 2003 at 1.5 Gbit/s, 2.0 in 2004 at 3 Gbit/s, and 3.0 in 2009 at 6 Gbit/s. All three use 8b/10b encoding, so 6 Gbit/s yields 600 MB/s of raw payload and about 550 MB/s after protocol overhead. SATA has had no speed increase since 2009; it is a finished standard.
3. AHCI, the standard SATA host controller interface, provides one command queue of at most 32 commands, and each command costs several uncached register accesses. NVMe provides up to 65,535 I/O queues of up to 65,536 commands each, with doorbell registers written at most twice per command.
4. NVMe 1.0 was published on 1 March 2011; the current revision is NVMe 2.3, published in August 2025. The specification family split into base, command set and transport documents at NVMe 2.0 in 2021.
5. PCIe encoding changed twice. Generations 1 and 2 used 8b/10b, giving a 20 percent tax. Generations 3 to 5 use 128b/130b, about 1.5 percent. PCIe 6.0 moved to PAM4 signalling with fixed-size FLIT framing and forward error correction. PCI-SIG released the PCIe 7.0 specification in 2025 at 128 GT/s per lane, roughly 15.1 GB/s per lane; products are not yet shipping.
6. M.2 lengths in use are 2230, 2242, 2260, 2280 and 22110, the last two digits or three being the length in millimetres. Handheld consoles and thin laptops often take only 2230.
7. Larger form factors: U.2, defined by SFF-8639, puts NVMe in a 2.5-inch hot-swappable drive. U.3, defined by SFF-TA-1001, accepts SAS, SATA and NVMe in one bay. Datacentres increasingly use EDSFF rulers such as E1.S and E3.S, which cool better and pack more capacity per rack unit.

Interface	Raw rate	Practical payload
SATA 3.0	6 Gbit/s	about 550 MB/s
SAS-4	22.5 Gbit/s	about 2.2 GB/s
USB 3.2 Gen 2	10 Gbit/s	about 1.0 GB/s

Interface	Raw rate	Practical payload
USB4 / Thunderbolt 4	40 Gbit/s	3.0 to 3.8 GB/s

8. Thunderbolt 5 and USB4 version 2 raise the link to 80 Gbit/s symmetric, with Thunderbolt 5 able to run 120 Gbit/s in one direction for displays. External NVMe enclosures should use the UASP protocol, not the older BOT mass-storage protocol, or they lose command queuing and most of their speed.
9. Mobile storage: eMMC 5.1 uses an 8-bit parallel bus and reaches roughly 250 to 400 MB/s. UFS uses a serial MIPI M-PHY link with full command queuing. UFS 4.0, published in 2022, reaches about 4,200 MB/s sequential read, and UFS 4.1 followed in 2025. eMMC is effectively obsolete in flagship phones and survives in low-cost devices and embedded boards.
10. Useful commands: `lspci -vv` to see link speed and width negotiated, `nvme list` and `nvme id-ctrl`, `smartctl -a`, and on macOS `system_profiler SPNVMeDataType`.

```
# What PCIe generation and width did the SSD actually get?
sudo lspci -vv -s $(lspci | grep -i nvme | cut -d' ' -f1) \
  | grep -E 'LnkCap|LnkSta'
# LnkCap says what it can do, LnkSta says what it negotiated.
```

12.8.6 WORDS - remember these

1. PATA / IDE — the old wide ribbon cable — parallel ATA, 16-bit, two devices per channel, up to 133 MB/s.
2. SATA — one fast serial pair per direction — serial ATA, 6 Gbit/s since 2009, driven through AHCI with a 32-command queue.
3. SAS — the enterprise serial interface — Serial Attached SCSI, dual ported, expander addressable, 22.5 Gbit/s at SAS-4.
4. NVMe — the language written for flash — NVM Express over PCIe, up to 65,535 queues of 65,536 commands.
5. PCIe lane — one wire pair each way — a differential lane, doubling in rate roughly every generation.
6. M.2 — the small stick shaped drive — 22 mm wide card, keyed to advertise which signals its socket carries.
7. UFS — the phone's fast storage — Universal Flash Storage, serial and queued, about 4,200 MB/s at UFS 4.0.

12.9 What a filesystem is

12.9.1 PLAIN - in simple words

1. A drive does not know what a file is. It only offers a very long numbered list of fixed-size blocks.
2. Humans want named things, in folders, that can grow and shrink.
3. A **filesystem** is the software that turns one into the other.
4. It has to answer four questions at all times: which blocks are free, which blocks belong to which file, what each file is called, and what else we know about it.
5. The information about a file, as opposed to the file's contents, is called **metadata**: size, owner, permissions, dates.
6. A folder is not a special kind of object. It is just a file whose contents are a list of names and pointers.
7. In the Unix family, the record holding a file's metadata and the addresses of its data blocks is called an **inode**.
8. The surprise for most people: the file's name is not in the inode. The name lives in the folder. The inode does not know what it is called.

12.9.2 PLAIN - a picture in your head

1. Think of a hotel. The rooms are numbered blocks on the disk.
2. The inode is the guest's registration card. It records who they are, when they arrived, and which rooms they occupy.
3. The directory is the list at reception mapping the name "Patel" to registration card number 4,192.
4. Two names can point at the same card. That is a **hard link**: one guest, two names at reception, one bill.
5. Deleting a name only crosses it off the reception list. The card is only destroyed when the last name pointing to it is gone.
6. That is exactly how Unix deletion works, and why the system call is called `unlink` rather than `delete`.

Where this comparison breaks: a hotel guest knows their own name, but an inode genuinely does not contain its filename. And renaming a real guest requires nothing but a pen at reception, which is precisely why renaming a file is fast no matter how large the file is.

12.9.3 PLAIN - a worked example

1. Here is what happens on an ext4 filesystem when you run four commands.

```
touch notes.txt
  1. allocate a free inode, say number 4192
  2. write inode: mode, owner, size 0, timestamps
  3. add entry ("notes.txt" -> 4192) to the directory file
  4. mark inode 4192 used in the inode bitmap

write 10 KB into it
  1. allocate 3 data blocks of 4 KB from the block bitmap
  2. write the bytes into those blocks
  3. record the extent (start block, length 3) in the inode
  4. set inode size = 10240, update mtime

mv notes.txt diary.txt
  1. add ("diary.txt" -> 4192) to the directory
  2. remove ("notes.txt" -> 4192)
  3. no data block is touched at all

rm diary.txt
  1. remove the directory entry
  2. decrement the inode link count, now 0
  3. free the 3 data blocks in the bitmap
  4. free inode 4192 in the inode bitmap
  5. the 10 KB of bytes are still physically there
```

2. Notice step 5 of the delete. Nothing overwrote your data. Only the bookkeeping changed. That is the whole basis of file recovery.
3. Notice also that renaming a 50 GB video is instant, because a rename edits a directory entry and never touches the data.

12.9.4 PLAIN - what is really happening inside

1. Blocks on disk are 512 or 4,096 bytes, but filesystems usually group them into a larger unit, called a **block** in Unix and a **cluster** in the Windows world. 4 KB is the common default.
2. A file always occupies a whole number of clusters. A 1-byte file consumes a full 4 KB. The wasted remainder is **internal fragmentation**, or slack.
3. Free space must be tracked. Three classic methods:
    1. A **bitmap**, one bit per block. Simple and compact. Used by ext4 and NTFS.
    2. A **linked chain**, where each entry names the next. Used by FAT.

3. **Extents**, recording start and length rather than every block, which is far more compact for big files. Used by ext4, NTFS, XFS, APFS.

4. Metadata typically includes file type, permission bits, owner and group IDs, size, link count, and three or four timestamps.
5. A directory is a file whose contents are `(name, inode number)` pairs. Modern filesystems index those pairs with a hash or a B-tree so that a directory with a million files is still fast to search.
6. Path lookup is therefore recursive. To open `/home/ana/a.txt`, the kernel reads the root directory to find `home`, reads that to find `ana`, reads that to find `a.txt`, then reads that inode. Each step is a real disk lookup, cached aggressively in memory.
7. Mounting attaches the root of one filesystem onto a directory of another, so the user sees one seamless tree.

12.9.5 TECHNICAL - the engineer's version

1. A classic Unix inode is 128 or 256 bytes and holds mode, uid, gid, size, link count, timestamps and either direct block pointers or an extent tree root. ext4 uses 256-byte inodes by default, which leaves room for nanosecond timestamps and inline extended attributes.
2. The original Unix File System used 12 direct pointers plus single, double and triple indirect blocks. With 4 KB blocks that reaches about 4 TB per file but costs up to four extra reads for a distant block. Extents replaced it because one extent record can describe up to 128 MiB in ext4.
3. Inodes are allocated at format time in ext2, ext3 and ext4. `mke2fs` defaults to roughly one inode per 16 KB of space, so a filesystem can run out of inodes while showing free space. `df -i` shows this, and it is a classic incident on mail and cache servers full of tiny files.
4. Filesystem block size is fixed at format time in most designs. XFS and ext4 are limited to the CPU page size, so 4 KiB on x86-64; APFS uses 4 KiB blocks; ZFS uses a variable record size up to 1 MiB.

```
stat notes.txt          # inode number, links, size, blocks, times
df -i /                 # inodes used and free, not just bytes
du -sh --apparent-size .   # real bytes, versus du -sh block usage
ls -di /home            # inode number of a directory
debugfs -R "stat <4192>" /dev/sda1   # raw inode dump, ext4
```

5. POSIX guarantees that `rename()` within one filesystem is atomic: an observer sees either the old name or the new one, never neither and never both. This is the foundation of the write-to-temporary-then-rename idiom used by nearly every well-written program that updates a file safely.
6. Hard links share an inode and cannot cross filesystems or, in most systems, point at directories. Symbolic links are separate inodes containing a path string, resolved at every use, and may dangle.

12.9.6 WORDS - remember these

1. Filesystem — the software that turns blocks into files — the on-disk format plus the driver implementing it.
2. Block or cluster — the allocation unit — the smallest number of sectors the filesystem will hand to a file, usually 4 KiB.
3. Metadata — everything about a file except its contents — mode, ownership, size, timestamps, link count, block map.
4. Inode — the file's record card — a fixed-size structure holding metadata and the map to data blocks, identified by number, not by name.
5. Directory — a file listing names — a mapping from names to inode numbers, usually indexed by hash or B-tree.
6. Extent — one run of consecutive blocks — a (start, length) record replacing per-block pointers.
7. Hard link — a second name for the same file — an additional directory entry referencing the same inode, raising its link count.

8. Slack — the wasted tail of the last cluster — internal fragmentation from rounding file size up to a whole allocation unit.

12.10 Real filesystems compared

12.10.1 PLAIN - in simple words

1. There is no single filesystem. Every operating system grew its own, and they all still exist because removable drives have to be readable everywhere.
2. **FAT** is the oldest survivor. Simple, small, understood by every device from cameras to car stereos, and very limited.
3. **exFAT** is FAT modernized to remove the size limits, and is now the normal choice for a big memory card or a shared USB stick.
4. **NTFS** is the Windows filesystem: journaled, permissioned, and full of extra features.
5. **ext4** is the standard Linux filesystem: reliable, fast, unexciting, and used on most servers on the internet.
6. **APFS** is Apple's, built for flash, and very good at making instant copies.
7. **Btrfs** and **ZFS** are the careful ones. They check every block against a stored fingerprint and can repair damage by themselves if you gave them a spare copy.
8. The rule of thumb: use your operating system's native filesystem for internal drives, and exFAT only for drives that must travel between systems.

12.10.2 PLAIN - a picture in your head

1. Think of a photocopier that is clever enough to be lazy.
2. You ask for a copy of a 500-page report. Instead of copying anything, it writes a note saying "this copy is the same as the original".
3. The copy appears instantly and takes no shelf space.
4. Later you edit page 12 of the copy. Only then does it actually duplicate page 12, and only page 12. The other 499 pages stay shared.
5. That is **copy-on-write**, and it is what APFS, ZFS and Btrfs do. It is why duplicating a 50 GB folder on a Mac can finish instantly.
6. A **snapshot** is the same trick applied to a whole filesystem: freeze the current state, and only start copying blocks when something changes them.

Where this comparison breaks: the photocopier note is a single record, while a real copy-on-write filesystem must keep a reference count on every shared block and free a block only when the last reference goes. And free space reporting becomes confusing: deleting a file may free nothing, because a snapshot still references it.

12.10.3 PLAIN - a worked example

1. Why will a 4 GB video not copy to a FAT32 memory stick, even with 200 GB free?
2. In a FAT32 directory entry, the file size is stored in a 32-bit unsigned field. That is a hard part of the on-disk format, not a bug.
3. The largest number that fits in 32 bits is 2 to the power 32, minus 1, which is 4,294,967,295.
4. So the largest possible FAT32 file is 4,294,967,295 bytes, one byte short of 4 GiB.
5. There is no workaround inside FAT32. The field is not big enough to write the number down.
6. The fixes are: reformat as exFAT or NTFS, or split the file into parts.
7. The related limit people meet is that the Windows graphical format tool refuses to make a FAT32 volume larger than 32 GB. That is a policy in the tool, not a limit of the format, which reaches 2 TiB with 512-byte sectors.

12.10.4 PLAIN - what is really happening inside

1. FAT keeps one table, the File Allocation Table, with one entry per cluster. Each entry either says "free", "bad", "end of file", or gives the number of the next cluster in this file. A file is a chain you follow.
2. That is why FAT is easy to implement and slow for large files: to reach the middle of a file you must walk the chain from the start.
3. NTFS keeps everything, including its own bookkeeping, in one enormous file called the **master file table**, or MFT. Each file gets a record, normally 1 KB.
4. A small file's contents live inside its own MFT record, so no data block is allocated at all. This is why NTFS handles thousands of tiny files well.
5. ext4 splits the volume into block groups, each with its own inode table and bitmaps, so related metadata and data stay physically close.
6. APFS, ZFS and Btrfs never overwrite a live block. A change writes new blocks, then new parent blocks pointing at them, up to the root, and finally one atomic pointer swap at the very top.
7. That is why they can offer instant snapshots and why they are always consistent after a crash. The old tree is still intact until the moment the root changes.
8. ZFS and Btrfs additionally store a checksum of every block in that block's parent. On read, the checksum is verified. If it fails and a redundant copy exists, the good copy is returned and the bad one is rewritten silently. That is **self-healing**, and it catches errors a drive never reports.

12.10.5 TECHNICAL - the engineer's version

1. Dates and origins. FAT12 came with Microsoft's 86-DOS work around 1980; FAT16 with PC DOS 3.0 in 1984; FAT32 with Windows 95 OSR2 in August 1996. exFAT appeared in Windows CE 6.0 in late 2006; Microsoft published the specification on 28 August 2019 and Linux gained a native driver in kernel 5.4 in November 2019.
2. NTFS shipped with Windows NT 3.1 in 1993. ext2 was written by Remy Card in 1993, ext3 added journaling in 2001 through work by Stephen Tweedie, and ext4 was marked stable in Linux 2.6.28 in December 2008. ZFS came from Sun Microsystems, principally Jeff Bonwick and Matt Ahrens, and shipped in OpenSolaris in 2005. Btrfs was started by Chris Mason at Oracle in 2007, with the on-disk format declared stable in Linux 3.10 in 2013. APFS was announced at Apple's WWDC in June 2016, shipped in iOS 10.3 in March 2017 and in macOS High Sierra in September 2017.

Filesystem	Max file	Max volume
FAT16	2 GiB	2 GiB at 32 KiB clusters
FAT32	4 GiB minus 1 byte	2 TiB at 512 B sectors
exFAT	16 EiB (spec)	128 PiB (spec)
NTFS	8 PiB minus 2 MiB	8 PiB at 2 MiB clusters
ext4	16 TiB at 4 KiB blocks	1 EiB
APFS	8 EiB	8 EiB
Btrfs	16 EiB	16 EiB
ZFS	16 EiB	256 ZiB per pool

Filesystem	Crash strategy	Main platform
FAT32 / exFAT	none (exFAT opt. TexFAT)	removable media
NTFS	metadata journal	Windows
ext4	journal, ordered mode	Linux
APFS	copy-on-write	macOS, iOS
Btrfs / ZFS	copy-on-write plus checksums	Linux, BSD, Solaris

3. The NTFS 8 PiB figure applies from Windows 10 version 1709 and Windows Server 2019; older releases capped volumes at 256 TiB with 64 KiB clusters. The theoretical on-disk limit is 16 EiB. Version and edition matter here, so quote the Windows release with any NTFS limit.
4. ext4's 16 TiB per-file limit follows from 32-bit extent block counts with 4 KiB blocks. The 1 EiB volume limit follows from 48-bit block addressing.
5. APFS space sharing means several volumes in one container draw from a shared free pool, so `df` shows the same free figure several times. A **clone** is a file-level copy-on-write duplicate created by the `clonefile` system call and used by `cp -c` and the Finder. A **snapshot** is the container-level equivalent and is what Time Machine local backups use.
6. macOS still cannot write to NTFS without third-party software, and Windows cannot read APFS or ext4 without third-party software. exFAT is the only format that all three major systems read and write out of the box, which is why cameras and large memory cards use it. SD cards above 32 GB, the SDXC class, specify exFAT in the SD standard.
7. Btrfs RAID 5 and RAID 6 profiles are still documented by the project as having unresolved issues and are not recommended for data you care about, as of 2026. Its RAID 0, 1 and 10 profiles are considered sound. Experts disagree on Btrfs stability generally; it is the default in some enterprise Linux distributions and avoided in others.

12.10.6 WORDS - remember these

1. FAT — the simple old chain-based format — File Allocation Table, one entry per cluster forming per-file linked lists.
2. exFAT — FAT with the limits removed — Microsoft's format for SDXC and large removable media, specification published 2019.
3. MFT — the giant index inside NTFS — Master File Table, one 1 KB record per file, small files stored resident.
4. Copy-on-write — never change a live block — write new blocks and swap the root pointer atomically.
5. Snapshot — a frozen view of the whole filesystem — a retained root pointer with reference counts preventing block reuse.
6. Clone — an instant duplicate that shares blocks — a file-level copy-on-write copy, `clonefile` on APFS, reflink on Btrfs and XFS.
7. Self-healing — the filesystem repairs bad blocks itself — checksum verification on read plus automatic repair from a redundant copy.

12.11 Journaling, consistency and crashes

12.11.1 PLAIN - in simple words

1. Almost nothing a filesystem does is a single write. Most operations are three or four writes that must all happen or none.
2. Appending to a file means: mark blocks used, write the data, update the file's record. A crash between any two leaves a half-finished mess.
3. If the blocks are marked used but the file record never learns about them, the space is lost forever.
4. If the file record claims blocks that were never marked used, the same blocks may later be handed to a second file. Now two files share bytes and both are ruined.
5. A **journal** fixes this by writing down what you are about to do, before you do it.
6. After a crash the system reads the journal. Anything fully written down is redone. Anything incomplete is thrown away.
7. Either way the filesystem is consistent afterwards, and it takes seconds rather than hours to check.
8. Consistent does not mean nothing was lost. It means nothing is contradictory. The last few seconds of your work can still be gone.

12.11.2 PLAIN - a picture in your head

1. Think of a bank teller moving money between two accounts.
2. Take 100 out of account A. Put 100 into account B. If the power fails in between, 100 has vanished from the world.
3. So before touching either account, the teller writes in a bound ledger: "about to move 100 from A to B", and only then starts.
4. If the lights go out, the next morning someone reads the ledger, sees an entry, and either finishes the move or undoes it.
5. The ledger is the journal. It is small, written in order, and never has to be searched.
6. Copy-on-write filesystems solve the same problem differently. They prepare a complete new version of everything off to one side, then change one single pointer to make it live.

Where this comparison breaks: a real journal does not have to hold the file data, only the bookkeeping, so a crash can leave a consistent filesystem containing a file full of rubbish. And the ledger only helps if the pen truly touched the paper before the lights went out, which is exactly the problem covered in the technical block below.

12.11.3 PLAIN - a worked example

1. Here is how a journaled metadata update runs, in order.

```
1. write journal: "intend: alloc blk 5000-5002, inode 4192,
                   size 10240"
2. FLUSH  -> force the drive to commit that to media
3. write journal commit record
4. FLUSH  -> the intent is now durable
5. write the real bitmap, inode and directory blocks
6. mark the journal entry complete

Crash after step 4 -> on mount, replay steps 5 and 6.
Crash before step 3 -> on mount, discard, nothing happened.
Crash between 5 and 6 -> replay is harmless, it is idempotent.
```

2. Notice steps 2 and 4. Without them the drive's own write cache may reorder everything and the whole scheme collapses.
3. Notice also that this costs two extra flushes and one extra write per operation. Journaling is not free; it buys recovery time with throughput.

12.11.4 PLAIN - what is really happening inside

1. Drives lie about writes for speed. A drive with a volatile write cache says "done" when the bytes reach its internal memory, not the platter or the flash.
2. That is safe for performance and lethal for ordering. A **flush** command forces the drive to actually commit everything it has acknowledged.
3. The kernel also has a cache, the page cache. A normal `write()` only copies bytes into memory and returns. The bytes may reach the drive seconds later.
4. `fsync()` is the system call that says "do not return until this file's data and metadata are truly on stable storage", and it triggers the drive flush.
5. Databases live or die on this. Every committed transaction must be provably on disk before the commit is reported to the client, or the database has lied about durability.
6. That is why database benchmarks care about flush latency more than about throughput, and why enterprise SSDs with capacitors are worth their price: they can honour a flush instantly because their cache is protected.
7. Journals normally protect metadata only. Your file's bookkeeping survives; the file's contents may be a mixture of old and new.

8. Copy-on-write designs need no journal for consistency, because the old tree remains valid until the atomic root swap. They still use a small log for speed when handling many tiny synchronous writes.

12.11.5 TECHNICAL - the engineer's version

1. ext4 offers three journal modes, set with the `data=` mount option: `journal` writes file data through the journal too, which is safest and slowest; `ordered` is the default and forces data blocks out before the metadata commit that references them; `writeback` orders nothing and can expose stale block contents after a crash.
2. NTFS journals metadata in `$LogFile` using write-ahead logging with undo and redo records. It never journals file data. XFS and JFS are also metadata-only journals.
3. Barriers are implemented as `FLUSH CACHE` in ATA, `SYNCHRONIZE CACHE` in SCSI and `Flush` in NVMe, or per-command with Force Unit Access. Mounting with `nobarrier` on a drive without power-loss protection is a known way to corrupt a filesystem, and should never be done outside a battery-backed array.
4. On macOS, `fsync()` deliberately does not force the drive to flush its cache. Programs that need real durability must call `fcntl(fd, F_FULLFSYNC, 0)`. This is a documented Apple-specific behaviour and a classic portability trap.
5. `fdatasync()` skips metadata that does not affect retrieval, such as mtime, which is measurably faster for database write-ahead logs.
6. Filesystem check tools. On a journaled filesystem a normal mount only replays the journal, taking under a second. A full structural check is a different operation: `fsck.ext4 -f` walks every inode, rebuilds the block and inode bitmaps, verifies link counts, and moves unreferenced-but-allocated inodes into `lost+found`. On a 20 TB ext4 filesystem with many files this can take hours and requires substantial memory. `xfs_repair` and `chkdsk /f` are the equivalents. ZFS deliberately has no fsck; it has `zpool scrub`, which verifies checksums on a live pool.

```
sudo tune2fs -l /dev/sda1 | grep -i 'features\|state'
sudo dumpe2fs -h /dev/sda1        # journal size and location
sudo zpool status ; sudo zpool scrub tank
sudo btrfs scrub start /mnt/data
```

7. The honest version: journaling protects filesystem structure, not your data. A power cut during a database write on a journaled filesystem can still lose committed-looking transactions if the application did not call fsync and the drive had no power-loss protection. Three separate parties must each do their job: the application, the kernel, and the drive.

12.11.6 WORDS - remember these

1. Journal — write down the plan before doing it — a write-ahead log of metadata changes, replayed or discarded at mount.
2. Atomicity — all of it or none of it — the guarantee that a multi-step update is never observed half-applied.
3. Write barrier — force the order — a cache flush or FUA operation preventing the device from reordering across a point.
4. fsync — really put it on the disk now — the system call forcing page cache and device cache to stable storage.
5. Ordered mode — data first, then the pointer to it — ext4's default, preventing a new inode from exposing stale block contents.
6. fsck — the full structural rebuild — offline consistency check reconstructing bitmaps and link counts, slow on large volumes.
7. Scrub — read everything and verify it — background checksum verification and repair in ZFS and Btrfs.

12.12 Partitions, formatting and booting

12.12.1 PLAIN - in simple words

1. One physical drive is usually cut into several independent regions called **partitions**.
2. The list of partitions is written in a small table at the very start of the drive.
3. There are two such table formats in use. The old one is **MBR**, the modern one is **GPT**.
4. MBR fits in one 512-byte sector, allows only four partitions, and cannot describe anything past 2 TB.
5. GPT allows a hundred and twenty-eight partitions, describes drives far larger than any that exist, and keeps a spare copy at the end of the drive in case the first is damaged.
6. **Formatting** a partition means writing an empty filesystem onto it: fresh bookkeeping structures saying "nothing is here".
7. A quick format writes only those structures. It never touches your old data, which is simply no longer referenced.
8. That is why a quick format is not a way to destroy anything, and why recovery software often gets almost everything back from a quick-formatted drive.
9. **Mounting** is attaching a filesystem so you can see it. Windows gives it a letter; macOS and Linux graft it onto a folder in one big tree.

12.12.2 PLAIN - a picture in your head

1. Think of a drive as a large empty warehouse and partitions as internal walls.
2. The partition table is the sign by the front door listing each room and where it starts and ends.
3. Formatting a room means putting up empty shelves and a fresh index card at the door. It does not mean sweeping the floor.
4. Quick formatting replaces the index card only. Everything on the shelves is still there, just unlisted.
5. Mounting is opening a door from the corridor into that room, so people can walk in.

Where this comparison breaks: the drive does not know or care about the walls. Partitions are purely a convention agreed between software, written in a table the drive itself never reads. You can wipe the table and every byte of the data is still physically present.

12.12.3 PLAIN - a worked example

1. Why exactly does MBR stop at 2 TB?
2. An MBR partition entry records the starting sector and the length as 32-bit numbers of sectors.
3. The largest count is 2 to the power 32, minus 1, that is 4,294,967,295 sectors.
4. With traditional 512-byte sectors: 4,294,967,295 times 512 equals about 2.199 times 10 to the 12 bytes, which is 2 TiB.
5. On a 4Kn drive with 4,096-byte sectors the same arithmetic gives 16 TiB, so the limit is a property of sector size, not of storage in general.
6. GPT stores those figures as 64-bit sector numbers. 2 to the power 64 sectors of 512 bytes is about 8 zebibytes, which no drive will reach this century.

```
An MBR: one 512-byte sector at LBA 0
+-----------------------------------------------+
| bootstrap code                          446 bytes|
| partition entry 1                        16 bytes|
| partition entry 2                        16 bytes|
| partition entry 3                        16 bytes|
| partition entry 4                        16 bytes|
| signature 0x55 0xAA                       2 bytes|
+-----------------------------------------------+
GPT: protective MBR at LBA 0, header at LBA 1,
128 entries of 128 bytes at LBA 2..33, backup at the end.
```

12.12.4 PLAIN – what is really happening inside

1. When an old BIOS machine starts, it loads the first sector of the boot drive into memory and jumps into it. That is the 446 bytes of bootstrap code in the MBR. Those 446 bytes then load a bigger loader from somewhere else.
2. A UEFI machine does something far more sensible. It reads the partition table, finds a partition marked as the **EFI System Partition**, which is formatted FAT32, and runs a normal file from it, such as `bootx64.efi`.
3. That is why the EFI partition is FAT32 on a modern Mac and PC: the firmware must be able to read it before any operating system exists.
4. Secure Boot adds a signature check on that file against keys stored in firmware.
5. A **logical volume manager** adds a layer between partitions and filesystems. It gathers whole drives into a pool and hands out volumes from that pool, which can be grown, shrunk, moved between drives and snapshotted while in use.
6. Without it, growing a filesystem means the free space must happen to sit directly after it on the same drive. With it, you just add another drive.
7. Mount points differ by system, but the idea is identical: the filesystem's root is grafted somewhere the user can reach.

12.12.5 TECHNICAL – the engineer's version

1. The MBR layout dates from PC DOS 2.0 in 1983. Four primary partition entries only; the extended partition trick chains further logical partitions through a linked list of extended boot records, which is fragile and now obsolete.
2. GPT is defined by the UEFI specification, descended from Intel's EFI work in the late 1990s for Itanium. Its header and entry array are each protected by a CRC32 checksum, and a full backup copy sits in the last sectors of the disk. LBA 0 holds a protective MBR describing one partition of type 0xEE covering the whole disk, so that MBR-only tools do not think the disk is blank and offer to initialize it.
3. Partition types are GUIDs in GPT. Common ones: EFI System Partition, Microsoft Basic Data, Linux filesystem, Linux LVM, Apple APFS Container.
4. Alignment matters. Partitions should start on a 1 MiB boundary so that filesystem 4 KiB blocks align with 4 KiB physical sectors and with SSD page and erase-block boundaries. Every current partitioner does this by default; tools from before about 2010 started at sector 63 and did not.
5. Formatting levels: a **low-level format**, meaning writing servo and sector structures, is a factory operation on modern drives and cannot be performed by a user. A **high-level format** writes filesystem metadata. Windows full format since Vista also writes zeros across the volume and runs a surface scan; Windows quick format does not.
6. Volume managers: LVM2 on Linux with physical volumes, volume groups and logical volumes; Storage Spaces on Windows; APFS containers on macOS; and ZFS, which merges volume management and filesystem into one layer deliberately.

System	Where drives appear	Table tool
Linux	/mnt, /media/user/LABEL	`lsblk`, `fdisk`, `parted`
macOS	/Volumes/NAME	`diskutil list`
Windows	drive letters, or folders	`diskpart`, Disk Management

```
lsblk -f                      # devices, filesystems, mountpoints
sudo parted /dev/sda print    # partition table type and entries
sudo blkid                    # UUIDs used by /etc/fstab
diskutil list                 # macOS, shows APFS containers
```

7. Mount by UUID or label, never by device name. Device names such as `/dev/sdb` are assigned in discovery order and can change between boots, which is a common cause of a machine failing to boot after adding a drive.

12.12.6 WORDS - remember these

1. Partition — one region of a drive — a contiguous LBA range described in a partition table.
2. MBR — the old 512-byte table — Master Boot Record, four primary entries, 32-bit sector counts, 2 TiB ceiling at 512-byte sectors.
3. GPT — the modern table — GUID Partition Table, part of UEFI, CRC-protected, with a backup copy at the end of the disk.
4. EFI System Partition — the small FAT32 partition the firmware boots from — holds signed `.efi` loader files.
5. Quick format — new bookkeeping, old data untouched — writing fresh filesystem metadata without erasing block contents.
6. Mounting — attaching a filesystem into the visible tree — associating a filesystem with a mount point or drive letter.
7. LVM — a layer that makes volumes flexible — logical volume manager pooling physical extents and serving resizable logical volumes.

12.13 RAID and redundancy

12.13.1 PLAIN - in simple words

1. **RAID** means using several drives together so that they behave as one, for speed, or for safety, or both.
2. **RAID 0** splits your data across drives in stripes. Twice the drives, twice the speed, all the capacity, and no safety at all. Lose one drive and you lose everything.
3. **RAID 1** writes the same data to two drives. Half the capacity, and either drive alone can carry on.
4. **RAID 5** spreads data across at least three drives and adds one drive's worth of **parity**, a checksum that lets any single missing drive be rebuilt. You lose one drive of capacity.
5. **RAID 6** does the same with two independent parities, so any two drives may die. You lose two drives of capacity.
6. **RAID 10** mirrors pairs and stripes across the pairs. Half the capacity, fast, and rebuilds quickly.
7. RAID protects against exactly one thing: a drive failing. That is all.
8. It does not protect against deletion, ransomware, corrupted files, a power surge, theft, fire or flood. Every one of those hits all drives at once because they are all in one box.
9. So RAID is about staying online, not about keeping data. It is not a backup.

12.13.2 PLAIN - a picture in your head

1. Imagine four people each holding one page of a four-page document, plus a fifth person holding a page of "checksums": for every character position, the sum of the four letters treated as numbers.
2. If one of the four is missing, you can work out their page exactly by subtracting the other three from the checksum page.
3. That is RAID 5 parity. It is not a copy of anything; it is the difference that lets you reconstruct whichever one you lost.
4. Now notice the danger. To reconstruct one page you must read every other page completely and correctly. If a second person makes even one mistake while you are reconstructing, the answer is wrong.

Where this comparison breaks: real parity uses exclusive-or rather than addition, and RAID 6 uses a second parity computed over a finite field so the two equations are independent. And the real risk is not a person making a mistake but a drive returning an unreadable sector during the rebuild.

12.13.3 PLAIN - a worked example

1. Eight 20 TB drives, and here is what you get from each layout.

Level	Usable	Survives
RAID 0	160 TB	no drive loss
RAID 1 (4 pairs)	80 TB	1 per pair
RAID 5	140 TB	any 1 drive
RAID 6	120 TB	any 2 drives
RAID 10	80 TB	1 per mirror pair

2. Now rebuild time, which is where RAID 5 loses its reputation.
3. Replacing a 20 TB drive means writing 20 TB onto it. At a generous sustained 150 MB/s that is 20,000,000 MB divided by 150, which is 133,333 seconds.
4. That is 37 hours, and only if the array is otherwise completely idle. On a busy array, two to five days is realistic.
5. During all that time you have zero redundancy and every surviving drive is being read from end to end, which is the hardest work they ever do.
6. Consumer drives are specified for one unrecoverable read error per 10 to the 14 bits, which is one per 12.5 TB read. Reading seven 20 TB drives in full is 140 TB. On the specified figure, hitting at least one error is likely.
7. That arithmetic is why RAID 6 or mirroring is the normal advice above about 4 TB per drive. The honest version: real drives usually beat their specified error rate by a wide margin, so the "RAID 5 is mathematically certain to fail" claim is too strong. Experts disagree on the size of the margin. Both camps agree RAID 6 is the safer choice at these capacities.

12.13.4 PLAIN - what is really happening inside

1. Parity is computed with exclusive-or, written XOR. XOR of a set of bits is 1 if an odd number of them are 1.
2. Its useful property: if you XOR all the data blocks together to make parity, then XOR any subset including parity but missing one block, you get the missing block back.
3. RAID 5 rotates which drive holds parity for each stripe, so no single drive becomes the write bottleneck.
4. Small writes to RAID 5 are expensive. To change one block, the controller must read the old block and old parity, compute the new parity, then write both. One logical write becomes two reads and two writes. This is the RAID 5 write penalty.
5. RAID 6's second parity uses Reed-Solomon arithmetic so that the two equations are independent and any two failures can be solved.
6. The **write hole** is a real hazard on parity RAID without a journal or battery: if power is lost between writing data and writing parity, the stripe is silently inconsistent, and a later rebuild will produce wrong data.
7. Hardware RAID controllers use a battery or capacitor-backed cache to close the write hole. Linux `md` uses a write-intent bitmap and an optional journal. ZFS avoids it entirely because RAID-Z stripes are copy-on-write and variable width.
8. RAID controllers rebuild from parity but usually cannot tell you which copy is right if the data is merely wrong rather than missing. Filesystems with checksums can, which is the argument for ZFS over classic RAID.

12.13.5 TECHNICAL - the engineer's version

1. The taxonomy comes from the 1988 SIGMOD paper "A Case for Redundant Arrays of Inexpensive Disks (RAID)" by David Patterson, Garth Gibson and Randy Katz at the University of California, Berkeley. The industry later renamed the I to "Independent".

2. RAID 2, 3 and 4 exist in the paper but are effectively unused: RAID 2 used Hamming codes and bit striping, RAID 3 byte striping with a dedicated parity drive, RAID 4 block striping with a dedicated parity drive. RAID 5's rotated parity removed RAID 4's bottleneck.
3. Nested levels are written as two digits: RAID 10 is a stripe over mirrors, RAID 01 is a mirror over stripes, and RAID 10 tolerates more failure patterns and rebuilds far faster because only one drive's worth of data must be copied.
4. ZFS RAID-Z1, Z2 and Z3 correspond roughly to single, double and triple parity, but with variable-width stripes that eliminate the write hole. A `zpool scrub` verifies every checksum and repairs from redundancy. Rebuilds in ZFS, called resilvering, copy only allocated blocks, so a half-empty pool resilvers in roughly half the time. Traditional RAID rebuilds the whole drive regardless.
5. Unrecoverable read error rate is specified as 1 in 10^14 bits for most consumer SATA drives and 1 in 10^15 for enterprise SATA, SAS and NL-SAS. 10^14 bits is 12.5 TB; 10^15 is 125 TB.
6. Practical guidance widely used in 2026: mirrors or RAID 10 for databases and virtual machines where write latency matters; RAID 6 or RAID-Z2 for bulk archives; RAID 5 only on small fast drives with a short rebuild window; RAID 0 only for scratch data you can regenerate.
7. The **3-2-1 rule**, popularized by the photographer Peter Krogh in his book "The DAM Book" in 2005: keep at least 3 copies of the data, on at least 2 different kinds of media, with at least 1 copy off site. A common modern extension is 3-2-1-1-0: one copy offline or immutable, and zero errors on a verified restore test.
8. The last part is the part people skip. A backup that has never been restored is a hypothesis, not a backup. Test restores on a schedule.

```
cat /proc/mdstat                  # Linux md array state
sudo mdadm --detail /dev/md0
sudo zpool status -v tank         # errors per device, scrub state
sudo smartctl -a /dev/sda | grep -i reallocated
```

12.13.6 WORDS - remember these

1. RAID — several drives behaving as one — Redundant Array of Independent Disks, from the 1988 Berkeley paper.
2. Striping — data spread across drives — RAID 0, dividing data into fixed-size chunks written in rotation.
3. Mirroring — the same data on two drives — RAID 1, full duplication.
4. Parity — the difference that rebuilds a lost drive — XOR for single parity, Reed-Solomon for the second.
5. Rebuild / resilver — reconstructing a replaced drive — a full-array read at maximum stress with no redundancy while it runs.
6. Write hole — data and parity out of step after a power cut — silent stripe inconsistency on parity RAID without a journal or backed cache.
7. URE — an unreadable sector — unrecoverable read error, specified at 1 in 10^14 bits for consumer drives.
8. 3-2-1 — the backup rule — three copies, two media types, one off site.

12.14 Data recovery, deletion and secure erase

12.14.1 PLAIN - in simple words

1. Deleting a file does not remove the file. It removes the entry that points to the file.
2. The bytes stay exactly where they were until something else happens to be given that space.
3. That is why recovery software works, and why an empty-looking second-hand drive can be full of somebody's private life.
4. On a hard disk, writing new data over the old data really does destroy it. One pass of zeros is enough on any drive made this century.
5. On an SSD, overwriting does not work, because the drive refuses to write in the same place. Your "overwrite" lands on a fresh page and the original is still sitting in a page you cannot address.

6. So SSDs need a different approach: a command that tells the drive itself to wipe everything, including the parts you cannot see.
7. The best approach of all is to encrypt the drive from the day you buy it. Then wiping it means destroying one key, which takes milliseconds.
8. Encrypted from the start, an unerased drive is already unreadable rubbish to anyone without the key.

12.14.2 PLAIN - a picture in your head

1. Think of a library index card system. Deleting a book means throwing away its card. The book is still on the shelf.
2. Recovery software walks the shelves ignoring the index, looking for anything that looks like a book. That is **file carving**.
3. Now the SSD problem. Imagine a librarian who, whenever you ask to replace a book, refuses to touch the original and instead shelves your replacement in a new spot, then quietly updates the index.
4. You can hand over replacement after replacement and the originals stay on the shelves, in a wing of the building you are not allowed to enter.
5. The only way to clear that wing is to ask the librarian, who alone has the keys, to burn everything.

Where this comparison breaks: file carving only finds file contents, not names, folders or dates, and cannot reassemble a file whose pieces were scattered. It works best on photos and videos, which have clear start and end markers and are usually written in one go.

12.14.3 PLAIN - a worked example

1. A JPEG image always starts with the bytes FF D8 FF and ends with FF D9.
2. Carving software reads the whole raw drive looking for those markers and writes out everything in between as a candidate file.
3. It needs no filesystem at all, which is why it survives a quick format.
4. Here is the difference between the erase options, on a 2 TB NVMe SSD.

Method	Time	Actually destroys data
Delete file	instant	no
Quick format	seconds	no
Overwrite with zeros	~3 min	not reliably, on flash
NVMe format, crypto erase	under 1 s	yes
NVMe sanitize, block erase	minutes	yes

5. The crypto erase is instant because it does not touch the data at all. It destroys the internal key the data was encrypted with, and the remaining bytes become noise.

12.14.4 PLAIN - what is really happening inside

1. Modern SSDs, and many hard drives, already encrypt everything they store, whether you asked for it or not. The key lives inside the drive.
2. That is called a self-encrypting drive. Normally the key is used automatically and you never notice.
3. A cryptographic erase simply replaces that key with a new random one. Every byte on the media instantly becomes undecryptable.
4. **ATA Secure Erase** is a command in the ATA standard telling the drive to erase its entire user area itself, including reallocated sectors that normal writes cannot reach. **Enhanced Secure Erase** additionally covers areas the basic command may skip.
5. NVMe has two: `Format NVM`, which can do a user-data erase or a cryptographic erase, and `Sanitize`, which is stronger and covers over-provisioned areas, caches and any unmapped data.

6. Full-disk encryption puts the key under your control rather than the drive maker's. BitLocker on Windows, FileVault on macOS, LUKS with dm-crypt on Linux, all normally using AES in XTS mode with a 256-bit key.
7. With full-disk encryption in place, disposal is trivial. Forget the passphrase, destroy the key slot, and the drive is a brick to everyone, including you.
8. Degaussing, a strong magnetic pulse, destroys hard disks and tapes permanently, and does absolutely nothing to flash, because flash stores charge and not magnetism.

12.14.5 TECHNICAL - the engineer's version

1. NIST Special Publication 800-88 Revision 1, "Guidelines for Media Sanitization", published in 2014, defines three levels: Clear, Purge and Destroy. It states explicitly that for modern magnetic media a single overwrite pass is adequate for Clear, and that cryptographic erase is an acceptable Purge method for self-encrypting media.
2. The 35-pass Gutmann method comes from Peter Gutmann's 1996 Usenix paper and targeted the specific MFM and RLL encodings of drives of that era. Gutmann himself has written that applying all 35 passes to a modern drive is pointless. Repeating it as current advice is a myth.
3. Commands and tools:

```
# ATA drives: check support, then issue a secure erase
sudo hdparm -I /dev/sda | grep -A6 Security
sudo hdparm --user-master u --security-set-pass p /dev/sda
sudo hdparm --user-master u --security-erase p /dev/sda

# NVMe: crypto erase (ses=2) or user-data erase (ses=1)
sudo nvme format /dev/nvme0n1 --ses=2
sudo nvme sanitize /dev/nvme0n1 --sanact=4   # crypto erase
sudo nvme sanitize-log /dev/nvme0n1          # progress
```

4. Drives are often left in a frozen security state by the firmware, and `hdparm` will report `frozen`. The usual workaround is a suspend and resume cycle, or hot-plugging the SATA data cable, both of which carry risk. Prefer `nvme sanitize` or vendor tools where available.
5. Recovery tooling: `photorec` and `scalpel` for carving, `testdisk` for rebuilding partition tables, `ddrescue` for imaging a failing drive with retry and error mapping. Always image first with `ddrescue` and work on the copy, never on the original.
6. On SSDs, TRIM interacts with recovery in a way that surprises people. On a TRIM-enabled drive with deterministic read-after-trim, reading a trimmed block returns zeros almost immediately after deletion. So on a modern SSD, deleting a file often does destroy it within seconds, and recovery frequently fails where it would have succeeded on a hard disk. This is a helpful property for privacy and a painful one for accidents.
7. Physical destruction remains the standard for the highest classifications: shredding to a specified particle size, disintegration, or incineration. For flash, particle size matters, because a surviving chip fragment can still hold readable pages.
8. The practical rule: encrypt at first use. Every erase question becomes easy afterwards, and the answer to "what if it fails before I can wipe it" becomes "it does not matter", which no amount of erasing software can give you.

12.14.6 WORDS - remember these

1. File carving — finding files by their shape, ignoring the index — signature based recovery scanning raw sectors.
2. ATA Secure Erase — the drive wipes itself — an ATA command covering the whole user area including reallocated sectors.
3. Sanitize — the strongest built-in erase — NVMe or ATA operation covering over-provisioned, cached and unmapped regions.

4. Cryptographic erase — throw away the key, not the data — destroying the media encryption key of a self-encrypting drive.
5. Self-encrypting drive — it was always encrypted — media encryption performed in the controller with an internally held key.
6. Full-disk encryption — the key is yours — BitLocker, FileVault, LUKS, normally AES-XTS with a 256-bit key.
7. Degaussing — a magnetic pulse that destroys magnetic media — effective on HDD and tape, useless on flash.

12.98 Common wrong ideas

1. Wrong: deleting a file erases it. Right: deleting removes a directory entry and frees the blocks. The bytes stay until something reuses the space, which is why recovery tools work on hard disks.
2. Wrong: an SSD has no moving parts, so it cannot fail. Right: flash wears out with every erase, leaks charge when unpowered, and SSDs commonly die suddenly from controller or firmware faults with no warning at all.
3. Wrong: defragmenting an SSD helps. Right: an SSD has no seek penalty, so defragmentation only burns write cycles. Windows deliberately runs TRIM instead when the drive is flash.
4. Wrong: RAID means my data is backed up. Right: RAID protects against a drive dying. Deletion, ransomware, corruption, fire and theft all reach every drive in the array at once.
5. Wrong: my 1 TB drive is faulty because the computer shows 931 GB. Right: the maker counted 10^12 bytes and the operating system is counting 2^40 bytes. Both numbers are correct, the units differ.
6. Wrong: a quick format wipes the drive. Right: it writes fresh empty bookkeeping and leaves every byte of old data physically in place.
7. Wrong: you must overwrite a drive seven or 35 times to be safe. Right: NIST SP 800-88 says one pass is enough on modern magnetic media, and on flash no number of overwrite passes is reliable. Use sanitize or cryptographic erase.
8. Wrong: more bits per flash cell is simply better value. Right: QLC gives more capacity per chip but roughly a tenth of the endurance of MLC and much slower sustained writes once the cache is exhausted.
9. Wrong: a filesystem journal protects my file contents. Right: most journals protect metadata only. After a crash the structure is consistent but a file may hold a mixture of old and new bytes.
10. Wrong: a 4 GB file will not fit because the USB stick is too small. Right: on FAT32 the file size field is 32 bits, so no file above 4 GiB minus one byte can be described, however much free space there is.

12.99 Chapter summary in 20 lines

1. Memory forgets without power and storage does not; every machine needs both because no technology is fast, cheap, huge and permanent at once.
2. The storage ladder runs from registers at under a nanosecond to tape at tens of seconds, spanning about eight orders of magnitude in latency.
3. Tape is not dead. LTO-10 reached 30 TB native in August 2025 and 40 TB in November 2025, at roughly a third of disk’s cost per terabyte.
4. A hard disk writes bits by flipping magnetic domains and reads them by sensing reversals, with the head flying a few nanometres up on an air bearing.
5. The IBM 350 of 1956 held about 3.75 MB on fifty 24-inch discs and weighed over a ton; a 2026 HAMR drive holds 44 TB on ten 3.5-inch platters.
6. Giant magnetoresistance, discovered in 1988 and awarded the 2007 Nobel Prize in Physics, is what made high-capacity drives possible.
7. Disk access time is seek plus rotational latency; average rotational latency is 30,000 divided by RPM in milliseconds.

8. That arithmetic caps a 7,200 RPM drive at roughly 80 random operations per second, a figure that has barely improved in thirty years.
9. Flash stores charge behind an insulator. Reading is cheap, writing damages the insulator, and more bits per cell means far less endurance.
10. NAND's defining rule: you read a page, you write a page, but you can only erase a whole block, so nothing is ever changed in place.
11. That rule forces an SSD to contain a small computer running a flash translation layer, doing wear levelling and garbage collection.
12. Write amplification measures the extra work; it rises sharply as the drive fills, which is why free space is working capital and not waste.
13. TRIM tells the drive which blocks are rubbish; without it an SSD slowly behaves as if permanently full.
14. NVMe over PCIe replaced one 32-command queue with up to 65,535 queues, which is why an SSD on PCIe 5.0 x4 reaches 14 GB/s and SATA stops at 550 MB/s.
15. A filesystem turns a numbered list of blocks into named files; the inode holds the metadata and the name lives in the directory, not in the inode.
16. Creating, writing, renaming and deleting are all metadata edits; renaming a 50 GB file is instant and deleting it overwrites nothing.
17. FAT32 cannot hold a file of 4 GiB or more because its size field is 32 bits wide; exFAT and NTFS remove that limit.
18. Journals make a crash leave a consistent structure, not intact contents; copy-on-write filesystems get the same guarantee by swapping one root pointer.
19. RAID keeps a service running when a drive dies, and protects against nothing else; the 3-2-1 rule, plus a tested restore, is the actual backup.
20. Encrypt every drive from first use, because then secure disposal is one discarded key rather than a hopeful overwrite that flash will quietly ignore.

13

CHAPTER

KB

Buses, Ports, Drivers and Firmware

13.0 What this chapter gives you

1. You will be able to point at any part of a motherboard and say what it does.
2. You will be able to explain what a bus is, and why fast links stopped being wide and became narrow instead.
3. You will be able to read "PCIe 4.0 x4" and work out the bandwidth yourself.
4. You will be able to describe every way a processor talks to a device: memory-mapped registers, ports, polling, interrupts and direct memory access.
5. You will be able to follow an interrupt from a wire to a running piece of code, and read the interrupt counters on a real machine.
6. You will be able to explain exactly what happens in the moment you plug in a USB device, step by step.
7. You will be able to answer the question "what is a driver" with a full chain of layers, not a slogan.
8. You will be able to say how Windows, Linux and macOS each load drivers, and how each one picks the right driver for a chip.
9. You will be able to separate firmware from driver from application, and say where each one lives.
10. You will be able to describe what happens in the first second after you press the power button, up to the point where the bootloader takes over.

Chapter 18 continues the story from where the bootloader hands off to the operating system kernel. This chapter stops at that handover.

13.1 The motherboard as a city

13.1.1 PLAIN - in simple words

1. The motherboard is a flat green board with everything plugged into it.
2. It is not one thing. It is a set of sockets, slots, wires and small chips.
3. The wires are not loose. They are thin copper lines baked into the board itself, called traces.
4. A modern board is a sandwich of six to sixteen thin layers of copper and glue, so traces can cross without touching.
5. Every part on the board does one job. Once you know the jobs, the board stops looking like a maze.
6. The big square socket in the middle holds the processor.
7. The long thin slots beside it hold the memory sticks.
8. The long slots below hold cards, such as a graphics card.
9. The small flat slots hold storage sticks.
10. The metal cluster at the back edge holds the connectors you can see from outside the case.

13.1.2 PLAIN - a picture in your head

1. Think of the board as a small city seen from above.
2. The processor socket is the city centre, where all the work is done.
3. The power circuits next to it are the power station, feeding the centre.
4. The memory slots are the warehouses, right next to the centre because the trips there are constant.
5. The chipset is the suburban traffic hub, handling everything that is not urgent enough to reach the centre directly.
6. The card slots are goods yards where big vehicles dock.
7. The back panel is the city gates, where things enter from outside.
8. The little battery is the town clock's backup, keeping time when the power is off.

Where this comparison breaks: a real city has traffic that goes anywhere. On a board almost every route is fixed at the factory and cannot change. A trace between two points is a permanent street, not a road that can be re-planned.

13.1.3 PLAIN - a worked example

1. Take a common desktop board layout and walk it top to bottom.
2. Here is what you would find, in order, on a typical mid-range board sold since about 2023.

```
    [ rear IO panel ]  [ CPU power socket ]
    +-----------------------------------------+
    | VRM  | CPU socket |  VRM   | D | D | D | D |
    |------+------------+--------| I | I | I | I |
    | PCIe x16 slot (from CPU)   | M | M | M | M |
    |   [ M.2 slot, under heatsink ]  M | M | M  |
    |----------------------------+--------------|
    | chipset chip under heatsink | SATA ports   |
    | PCIe x1 | PCIe x4 | M.2 | M.2 | BIOS chip  |
    | front panel + USB headers   | CMOS battery |
    +-----------------------------------------+
```

3. The board is read like a page. Power at the top, processor in the middle, slow devices at the bottom.
4. Fast things sit near the processor because the wires must be short.
5. Slow things sit far away because a longer wire does not hurt them.

13.1.4 PLAIN - what is really happening inside

1. **CPU socket**: a bed of spring pins that press against the processor. It carries power in and every high-speed signal out.
2. **VRM**, short for voltage regulator module: the small square chips, coils and cylinders next to the socket. They cut 12 volts down to under 1.5 volts for the processor.
3. **DIMM slots**: the long clipped slots holding memory sticks. Their wires go straight to the processor on any modern board.
4. **Chipset**: one large chip under a low heatsink. It is a fan-out switch, turning a few fast lanes from the processor into many slower ports.
5. **PCIe slots**: the long open-ended slots for cards. They carry high-speed serial links plus power.
6. **M.2 slots**: short flat slots for storage sticks. They usually carry the same kind of high-speed link as a card slot, in a smaller shape.
7. **SATA ports**: small flat right-angle sockets for older drives.
8. **Headers**: bare pin blocks with no plastic shell. The case's power button, front USB ports, fans and front audio plug into these.
9. **CMOS battery**: a coin cell that keeps the clock and settings alive when the machine is unplugged.
10. **BIOS flash chip**: a tiny eight-legged chip holding the startup code. Without it the board cannot even find the processor's first instruction.

11. **Super IO chip**: a small chip that runs fans, reads temperatures and voltages, and provides the last old-style ports.
12. **Rear IO panel**: the metal strip of outward-facing connectors.

13.1.5 TECHNICAL - the engineer's version

1. Sockets are LGA (land grid array, pins in the socket) or PGA (pin grid array, pins on the chip). Intel LGA1700 has 1700 contacts; AMD AM5 is LGA with 1718 contacts and replaced the pinned AM4 in 2022.
2. A large share of socket contacts are power and ground, not signals. On LGA1700 only a minority carry data.
3. VRMs are multiphase synchronous buck converters. A phase is one inductor plus a high-side and low-side MOSFET, driven by a PWM controller. Phases are switched out of step to smooth the current ripple.
4. A desktop processor core voltage is roughly 0.7 V to 1.4 V, and sustained current can exceed 200 A on high-end parts. That is why the VRM needs heatsinks.
5. DDR4 and DDR5 unbuffered DIMMs both have 288 pins, but the key notch sits in a different place, so they are not interchangeable. DDR5 moved the power management IC onto the DIMM itself.
6. On Intel platforms the chipset is the PCH (platform controller hub), attached over DMI (Direct Media Interface), which is electrically PCI Express. Recent desktop chipsets use a DMI 4.0 x8 link.
7. On AMD AM5 the chipset is called a Promontory device and attaches over a PCIe link; high-end boards chain two chipset chips together.
8. The firmware chip is SPI NOR flash in an 8-pin SOIC or WSON package, typically 16 MB or 32 MB (128 or 256 megabit) on boards since about 2020.
9. Super IO chips are made by Nuvoton and ITE, for example the ITE IT8689E. They hang off the LPC or eSPI bus and expose fan tachometers, PWM fan control, voltage and temperature sensing, and legacy serial ports.
10. The coin cell is a CR2032, 3 V lithium, about 225 mAh, and typically lasts three to seven years.
11. Board layer counts: four layers on budget boards, eight to twelve on mainstream boards, and more on server boards, because DDR5 and PCIe 5.0 signalling needs clean reference planes.

Part	Job in one line	Typical figure
VRM phase	Step 12 V down	50 to 110 A each
CMOS cell	Keep clock and setup	CR2032, 3 V
BIOS flash	Hold startup code	16 to 32 MB SPI
DMI 4.0 x8	CPU to chipset link	About 16 GB/s

12. Tools that show this without opening the case: `dmidecode -t 2` on Linux prints the board maker and model, and `lspci` lists the chipset devices.

13.1.6 WORDS - remember these

1. Trace — a printed wire on the board — a copper track etched in a PCB layer.
2. Socket — the seat the processor sits in — LGA or PGA contact array.
3. VRM — the power step-down circuit — multiphase synchronous buck converter.
4. Chipset — the traffic hub chip — PCH or southbridge fan-out switch.
5. Header — a bare block of pins — an unshrouded board-level connector.
6. Super IO — the fans and sensors chip — LPC or eSPI attached legacy IO controller.
7. Flash chip — where startup code lives — SPI NOR device holding the firmware image.

13.2 What a bus actually is

13.2.1 PLAIN - in simple words

1. A bus is a set of wires shared by several parts, so they can pass numbers to each other.
2. It carries three kinds of information: where, what, and when.
3. The “where” wires carry an address, meaning which device or which memory slot is being spoken to.
4. The “what” wires carry the data itself.
5. The “when” wires carry control signals: read or write, start now, I am ready, I am finished.
6. Only one part may talk at a time on a shared bus, or the messages would collide.
7. There are two ways to send many bits: side by side on many wires, or one after another on a few wires.
8. Side by side is called parallel. One after another is called serial.
9. Parallel sounds faster, and for a long time it was.
10. Above a certain speed, parallel stops working and serial wins. This chapter explains why.

13.2.2 PLAIN - a picture in your head

1. Imagine eight runners set off together carrying one letter each. The message is only complete when all eight arrive.
2. That is a parallel bus. Eight wires, eight bits, all at once.
3. If one runner is slightly slower, everyone waits for them. The whole message is only as fast as the worst runner.
4. Now make them run much faster. Tiny differences in path length become large differences in arrival time.
5. Worse, running close together, they knock into each other and drop letters.
6. Now imagine one runner, on one clean track, carrying letters one after another, but sprinting far faster than the group ever could.
7. That is a serial link. Fewer wires, but each wire far quicker.

Where this comparison breaks: real serial links are not truly one runner. They have a separate track in each direction, and often several tracks in parallel called lanes. The difference is that each lane carries its own timing, so the lanes do not have to arrive in perfect step.

13.2.3 PLAIN - a worked example

1. The bandwidth of a simple parallel bus is width times clock rate.
2. Formula: bytes per second = (bus width in bits / 8) x transfers per second.
3. Old PCI: 32 bits wide, 33.33 million transfers per second.
4. 32 / 8 = 4 bytes. 4 x 33,333,333 = 133,333,332 bytes per second.
5. That is about 133 MB/s, and it is the number the PCI specification quotes.
6. Now a 64-bit PCI slot at 66 MHz: 8 bytes x 66,666,666 = about 533 MB/s.
7. Now compare one PCI Express 3.0 lane. It is two wires each way, not 32.
8. It runs at 8 gigatransfers per second and delivers about 985 MB/s each way at the same time.
9. One tiny serial lane beats the whole 32-wire parallel bus, and does it in both directions at once.

Link	Wires for data	Bandwidth
PCI 32-bit 33 MHz	32 shared	133 MB/s shared
PCI 64-bit 66 MHz	64 shared	533 MB/s shared
PCIe 3.0 x1	4 (2 per way)	985 MB/s each

13.2.4 PLAIN - what is really happening inside

1. On a parallel bus every bit of one number leaves at the same instant, on its own wire.
2. Traces on a board are never exactly the same length, because they must bend around other parts.
3. A signal moves along a board trace at roughly half the speed of light, about 15 centimetres per nanosecond.
4. So a 1.5 centimetre difference in trace length is a 0.1 nanosecond difference in arrival. That gap is called skew.
5. At 33 million transfers per second each bit lasts 30 nanoseconds, so 0.1 nanosecond of skew does not matter at all.
6. At 8 billion transfers per second each bit lasts 0.125 nanoseconds. Now 0.1 nanoseconds of skew almost swallows the whole bit.
7. Board makers fix this by adding wiggles to short traces to match lengths, which is why you see squiggly patterns near memory slots.
8. Length matching gets harder as speed rises, and eventually impossible.
9. Second problem: a changing signal on one wire pushes a small copy of itself onto the wire beside it. That is crosstalk. More wires close together means more of it.
10. Third problem: on a shared bus, every device hanging off the wires adds electrical load, which rounds off the sharp edges of the signal.
11. Serial links dodge all three. Fewer wires means less crosstalk. Each lane carries its own timing, so skew between lanes can be corrected in the receiver. Point-to-point means only two ends, so the load is fixed and known.
12. Modern serial links do not send a separate clock. The clock is recovered from the data itself, which is why the data has to be encoded so it never stays flat for long.

13.2.5 TECHNICAL - the engineer's version

1. A classic bus has three groups: an address bus, a data bus and a control bus. The 8088 in the 1981 IBM PC had a 20-bit address bus and an 8-bit external data bus, reaching 1 MiB of address space.
2. Shared multi-drop buses are limited by settling time, reflections from unterminated stubs, and the capacitive load of every attached device.
3. Modern serial links are differential: each lane is a pair of wires driven to opposite voltages. The receiver reads the difference, so noise that hits both wires equally cancels out.
4. Signalling levels are low. PCI Express uses roughly 0.8 to 1.2 V differential swing, versus 5 V or 3.3 V single-ended on old PCI.
5. Clocking is embedded. Transmitters scramble and encode data so transitions occur often enough for a clock and data recovery (CDR) circuit to lock on.
6. Line codes cost bandwidth. 8b/10b sends 10 symbols for 8 data bits, a 20 percent overhead. 128b/130b sends 130 for 128, about 1.5 percent.
7. Raw rate is quoted in GT/s, gigatransfers per second, not GHz, because modern links may carry more than one bit per transfer.
8. Bandwidth formula for a serial link: bytes/s = (GT/s x lanes x coding efficiency) / 8.
9. Worked check for PCIe 3.0 x1: 8 GT/s x 1 x (128/130) / 8 = 0.985 GB/s.
10. Worked check for PCIe 5.0 x4: 32 x 4 x (128/130) / 8 = 15.75 GB/s each direction.

```
Parallel, shared:        Serial, point-to-point:

CPU ====================     CPU <--TX--> device
 ||   ||   ||   ||             (one pair each way,
dev  dev  dev  dev              own clock recovery)
(all share one set of wires)
```

11. Skew budget: at 32 GT/s a unit interval is 31.25 picoseconds. Board designers work to intra-pair skew targets in the low single-digit picoseconds. This is why parallel buses stopped scaling.

12. The honest version: serial did not beat parallel on raw physics alone. It won because it moved the hard problem into silicon, where equalization, scrambling and error correction are cheap, and away from the board, where length matching is expensive.
13. Parallel is not dead. Inside a chip, and between a processor and DRAM, wide parallel interfaces still win because the distances are short and fixed. DDR5 still uses a 64-bit data path per channel pair.

13.2.6 WORDS - remember these

1. Bus — shared wires between parts — a multi-drop interconnect with address, data and control groups.
2. Parallel — many bits at once — simultaneous transmission across a wide data path.
3. Serial — bits one after another — sequential transmission on a lane.
4. Skew — bits arriving at different times — timing difference between parallel signals at the receiver.
5. Crosstalk — one wire disturbing its neighbour — unwanted coupling between adjacent conductors.
6. Differential pair — two wires carrying opposites — balanced signalling with common-mode noise rejection.
7. GT/s — how many transfers per second — gigatransfers per second, before coding overhead is removed.
8. Line code — the packaging of bits on the wire — the mapping such as 8b/10b or 128b/130b.

13.3 The history of expansion buses

13.3.1 PLAIN - in simple words

1. From the beginning, computers needed a way to add hardware after buying the machine.
2. The answer was a slot: a row of contacts on the board, into which a card slides.
3. The first widely copied slot on personal computers came with the IBM PC in 1981.
4. It was slow, but it was documented, so anyone could make cards for it.
5. Graphics grew hungry, and each new slot design existed mainly because pictures needed more room.
6. Every generation was faster and had fewer wires than you would expect.
7. Today one design has replaced them all: PCI Express, usually written PCIe.
8. PCIe comes in different physical sizes and different speed generations, and the two are separate ideas.
9. A slot's size tells you how many lanes it can hold. A generation tells you how fast a lane runs.

13.3.2 PLAIN - a picture in your head

1. Think of a loading bay with a number of doors.
2. The size of the bay is how many doors it has: one, four, eight or sixteen.
3. The generation is how fast one door can move goods.
4. A sixteen-door bay of an old generation and a four-door bay of a new generation can move about the same amount.
5. That is why an older graphics card in a big slot can be matched by a newer card in a smaller one.
6. Also, a bay may have sixteen doors built, but only four of them connected to a road. The bay is full-size but only four doors are usable.

Where this comparison breaks: doors move goods one way. A PCIe lane moves data both ways at once, so the total for a link is double the one-way figure if both directions are busy.

13.3.3 PLAIN - a worked example

1. A graphics card that says "PCIe 4.0 x16" wants sixteen lanes at generation four speed.
2. Generation four gives about 1.969 GB/s each way per lane.
3. Sixteen lanes: 16 x 1.969 = about 31.5 GB/s each way.
4. Put the same card in a slot wired at x4. Now 4 x 1.969 = about 7.9 GB/s.

5. Put it in a PCIe 3.0 x16 slot instead. 16 x 0.985 = about 15.75 GB/s.
6. Notice that a generation four x8 link and a generation three x16 link give the same figure. That is not a coincidence: each generation roughly doubles the per-lane speed.

13.3.4 PLAIN - what is really happening inside

1. **ISA, 1981**: 62 contacts, 8 data bits, running at the processor clock of 4.77 MHz. In 1984 the IBM PC/AT added a second connector section for 16 bits, and the speed settled at 8 MHz.
2. Cards on ISA had switches and jumpers on them, because nothing was automatic. You set the card's address and interrupt number by hand.
3. **VESA Local Bus, 1992**: a hack. It bolted an extra connector onto the end of an ISA slot and wired it straight to the 486 processor's own bus.
4. It was fast for the time, up to about 200 MB/s, but the processor could only drive two or three such cards, and it died when the Pentium changed the processor's bus.
5. **PCI, 1992**: designed at Intel and then handed to a group called the PCI Special Interest Group. 32 bits at 33 MHz, about 133 MB/s, shared by every card in the machine.
6. PCI's real gift was not speed. It was that each card could describe itself, so the machine could set addresses and interrupts automatically. No more jumpers.
7. **AGP, 1997**: a private lane just for graphics, so the video card did not have to queue behind everything else on PCI.
8. **PCI Express, 2003**: the change of shape. It threw away the shared parallel bus and used point-to-point serial lanes with a switch in the middle.
9. Each PCIe device gets its own link. Nothing is shared, so one slow device no longer holds up the rest.

13.3.5 TECHNICAL - the engineer's version

1. Dates and figures for the main expansion buses:

Bus	Year	Peak bandwidth
ISA 8-bit	1981	About 4 to 8 MB/s
ISA 16-bit	1984	About 8 to 16 MB/s
VESA Local Bus	1992	Up to about 200 MB/s
PCI 32/33	1992	133 MB/s shared
AGP 8x	2002	2133 MB/s
PCIe 1.0 x16	2003	4 GB/s each way

2. AGP was introduced by Intel with the 440LX chipset on 26 August 1997. Its base clock was 66 MHz: 1x gives 266 MB/s, 2x gives 533 MB/s, 4x gives 1066 MB/s at 1.5 V, and 8x gives 2133 MB/s at 0.8 V.
3. Compaq coined the retronym Industry Standard Architecture for the IBM AT bus in the late 1980s, because IBM never gave it a public name.
4. PCI Express per-lane figures, one direction, after coding overhead:

Gen	Year	GT/s	Per lane each way
1.0	2003	2.5	250 MB/s
2.0	2007	5.0	500 MB/s
3.0	2010	8.0	985 MB/s
4.0	2017	16.0	1.969 GB/s
5.0	2019	32.0	3.938 GB/s
6.0	2022	64.0	7.563 GB/s
7.0	2025	128	15.125 GB/s

5. Generations 1.0 and 2.0 use 8b/10b, so the usable fraction is 80 percent. Generations 3.0 through 5.0 use 128b/130b, about 98.5 percent.
6. Generation 6.0, released in January 2022, changed the physics: PAM4 signalling carries two bits per symbol, with forward error correction and fixed 256-byte FLIT packets carrying 242 bytes of payload. Generation 7.0 was finalized in June 2025 and keeps PAM4.
7. First shipping PCIe 6.0 hardware appeared in 2025, three years after the specification. Specification date and product date are always different. Say which one you mean.
8. Physical versus electrical width: a slot with a x16 connector may only have four lanes wired. This is written as "x16 slot, x4 electrical". It happens because the processor has a fixed lane budget.
9. A typical desktop processor since 2021 exposes 20 to 28 usable lanes. A graphics card takes 16 and an NVMe drive takes 4, so the rest of the slots are wired through the chipset and share the chipset's uplink.
10. **Bifurcation** is splitting one wide link into several narrower ones: an x16 root port configured as x4/x4/x4/x4 to drive four NVMe drives on one adapter card. It must be supported by the processor's root complex and exposed in firmware setup. Cheap adapter cards that do not include a PCIe switch require bifurcation; cards with a switch chip do not.
11. Whether bifurcation is available at all is an implementation detail of one processor and one board's firmware, not something PCI Express guarantees. Two boards with the same chipset often differ.
12. Link training negotiates width and speed at power-on. If a lane fails, the link retrains narrower, so a card can silently fall back to x8 or x1.
13. Observe it on Linux with `lspci -vv` and read the LnkCap and LnkSta lines:

```
LnkCap: Speed 16GT/s, Width x16
LnkSta: Speed 2.5GT/s (downgraded), Width x16
```

14. A speed of 2.5 GT/s at idle is usually normal power saving, not a fault. Check under load before concluding anything.

13.3.6 WORDS - remember these

1. Slot — the place a card plugs in — a card edge connector on a defined pinout.
2. Lane — one serial path — a transmit pair plus a receive pair in PCIe.
3. Generation — how fast a lane runs — the PCIe specification revision setting the transfer rate.
4. Electrical width — how many lanes are actually wired — the negotiated link width, which may be less than the connector size.
5. Bifurcation — splitting one wide slot into several — configuring a root port as multiple narrower links.
6. Link training — the handshake at power-on — the LTSSM process that agrees width, speed and equalization.

13.4 How the processor talks to a device

13.4.1 PLAIN - in simple words

1. A device is not magic. It is a chip with a set of numbered boxes inside it, called registers.
2. Writing a number into one box makes the device do something.
3. Reading a number out of another box tells you what the device is doing.
4. So "controlling a device" means writing and reading small numbers at fixed places.
5. There are two ways for the processor to reach those boxes.
6. The first way pretends the device's boxes are ordinary memory addresses. Write to address such-and-such and the device hears it. This is called memory-mapped input and output.
7. The second way uses a separate, smaller set of addresses reached only by two special instructions. This is called port-mapped input and output.
8. Once you can reach the boxes, you still need to know when the device has finished a job.

9. You can keep asking, over and over. That is called polling, and it wastes the processor's time.
10. Or the device can tap the processor on the shoulder when it is ready. That is an interrupt.
11. Moving a lot of data through the processor one piece at a time is slow. So devices are allowed to move data to and from memory by themselves. That is direct memory access.

13.4.2 PLAIN - a picture in your head

1. Imagine a hotel reception desk with a row of pigeonholes on the wall behind it.
2. Each pigeonhole is a device register. Putting a card in box 3 means "start the lift". Reading box 4 tells you which floor the lift is on.
3. Memory-mapped input and output means those pigeonholes are simply part of the same wall as all the other room boxes. You use the same ladder.
4. Port-mapped input and output means the pigeonholes are in a locked side room with its own small ladder and its own numbering.
5. Polling is standing at the desk asking "is the lift here yet?" every two seconds, doing nothing else.
6. An interrupt is going back to work and letting a bell ring when the lift arrives.
7. Direct memory access is telling the porter to move the luggage between rooms without carrying each bag past the desk yourself.

Where this comparison breaks: a hotel porter can be trusted to only touch the right rooms. A device doing direct memory access can, unless stopped, write anywhere in memory. That risk is exactly why the IOMMU was invented.

13.4.3 PLAIN - a worked example

1. Here is a classic serial port, in the style of the original IBM PC, using port-mapped input and output.
2. The first serial port lives at port address 0x3F8.
3. Port 0x3F8 is the data box. Port 0x3FD is the status box.
4. Bit 5 of the status box means "the transmit box is empty, you may send".
5. Sending one character by polling looks like this.

```
/* wait until bit 5 of the status port is set */
while ((inb(0x3FD) & 0x20) == 0) {
    /* spin: the CPU does nothing useful here */
}
/* Linux order is outb(value, port). Some other
   headers use outb(port, value). Always check. */
outb('A', 0x3F8);    /* write the character */
```

6. At 9600 bits per second, one character takes about 1 millisecond.
7. A 3 GHz processor executes roughly 3 million cycles in that millisecond.
8. So sending one character by polling burns about 3 million cycles of doing nothing.
9. With an interrupt, the processor writes the character, goes off and runs other programs, and is called back when the port is free again.
10. With direct memory access, the processor hands over a pointer to 4 KB of text and is called back once, at the end, instead of 4096 times.

13.4.4 PLAIN - what is really happening inside

1. When the processor executes a write to a memory address, that address goes out on the interconnect.
2. Something has to decide who owns that address. That job belongs to address decoding logic in the processor's root complex and in bridges.
3. Ranges of the address space are handed out at start-up to devices. Each device says how big a window it needs, and firmware assigns it a base address.
4. So writing to address 0xF7C0_0010 might not touch memory at all. It might land in register 0x10 of a network card.

5. These device windows must not be cached, because reading the same address twice can legitimately give different answers, and a write must reach the device now and not sit in a cache.
6. Port-mapped input and output is different. On x86 the instructions IN and OUT carry a 16-bit port number, giving 65,536 ports in a space entirely separate from memory.
7. For direct memory access, the processor writes into the device's registers the address of a buffer in main memory and a length, then sets a "go" bit.
8. The device then becomes a bus master: it issues its own read and write requests to memory, without the processor being involved.
9. Real buffers are rarely one solid block of physical memory. A 1 MB buffer seen by a program as continuous may be scattered across 256 separate 4 KB physical pages.
10. So the driver builds a list of address and length pairs, and hands the list to the device. The device walks the list. This is scatter-gather.
11. Since the device writes straight into memory, the processor's cached copy of that memory can become stale. Either the hardware keeps caches coherent, or the driver must explicitly invalidate the range.
12. A device that can write anywhere in memory can overwrite the operating system. The IOMMU sits between devices and memory and translates and checks every device address, exactly as the MMU does for programs.

13.4.5 TECHNICAL - the engineer's version

1. MMIO regions are advertised through PCI Base Address Registers (BARs). A device has up to six 32-bit BARs; a 64-bit BAR consumes two.
2. At power-on, firmware writes all ones to a BAR and reads it back. The number of low bits that stay zero tells you the size of the window the device wants. Firmware then programs a base address.
3. On x86 the page attributes for MMIO are set to UC (uncacheable) or WC (write-combining) via the MTRRs and PAT, so accesses are not reordered or merged in ways the device would misread.
4. Port IO on x86 uses IN, OUT, INS and OUTS. Notable legacy addresses: 0x20 and 0xA0 for the two 8259A interrupt controllers, 0x60 and 0x64 for the keyboard controller, 0x70 and 0x71 for CMOS and the real-time clock, 0x3F8 for COM1, 0x378 for LPT1.
5. Port IO survives only for legacy compatibility. ARM and RISC-V have no separate IO space at all; everything is memory-mapped.
6. The original PC used an Intel 8237A DMA controller with four channels, and the PC/AT added a second for 16-bit channels 5, 6 and 7, with channel 4 used to cascade. This is third-party DMA, where a central controller moves the data.
7. Third-party DMA is obsolete on PCI Express. Every PCIe endpoint that moves bulk data is a bus master doing first-party DMA.
8. Bus mastering must be explicitly enabled by setting the Bus Master Enable bit in the device's PCI command register. A driver that forgets this sees a device that appears to do nothing.
9. Scatter-gather lists are called SGLs, or descriptor rings in networking. An NVMe drive supports both PRP (physical region page) lists and SGLs.
10. IOMMU implementations: Intel VT-d, AMD-Vi, and Arm SMMU. They provide an IO virtual address space per device, using the PCIe requester ID (bus:device.function) to select a page table.
11. The IOMMU defends against DMA attacks. The FireWire DMA attacks of the 2000s and the Thunderclap research of 2019 both exploited devices with unrestricted memory access. It also makes device passthrough to virtual machines safe.
12. IOMMU costs. Address translation adds latency, and IOTLB misses hurt. High-throughput setups often use large pages or, in some deployments, passthrough mode, trading protection for speed. Engineers genuinely disagree about where that line should be.

Mechanism	Who moves the data	Cost per byte
Polled PIO	The CPU, in a loop	Very high
IRQ + PIO	The CPU, on demand	High
DMA	The device itself	Near zero for CPU

13. Observe on Linux: `lspci -vv` shows BAR regions as "Memory at ..." and "I/O ports at ...", and `cat /proc/iomem` and `/proc/ioports` list the whole assignment.

13.4.6 WORDS - remember these

1. Register — a numbered box inside a chip — an addressable control or status location in a device.
2. MMIO — device boxes look like memory — memory-mapped IO through a BAR window, marked uncacheable.
3. Port IO — a separate small address space — x86 IN and OUT over a 16-bit port space.
4. Polling — asking again and again — busy-wait on a status register.
5. DMA — the device moves data itself — direct memory access by a bus master.
6. Scatter-gather — a list of memory pieces — a descriptor list of address and length pairs.
7. IOMMU — the guard between device and memory — an IO memory management unit translating and checking device addresses.

13.5 Interrupts, end to end

13.5.1 PLAIN - in simple words

1. An interrupt is a device saying "stop what you are doing and deal with me".
2. Without interrupts, the processor would have to check every device constantly, which would waste almost all of its time.
3. The processor finishes the instruction it is on, saves where it was, and jumps to a small piece of code written for that device.
4. That piece of code is called the interrupt handler, or interrupt service routine.
5. When the handler finishes, the processor restores what it saved and carries on exactly where it left off. The interrupted program never knows.
6. Many devices may want attention at once, so a helper chip collects and ranks the requests. That is the interrupt controller.
7. Each source has a number, so the processor knows which handler to run.
8. A table in memory maps each number to the address of its handler.
9. Interrupt handlers must be extremely short, because everything else is frozen while one runs.
10. So the work is split: a tiny urgent part now, and the slow part scheduled for a moment later.

13.5.2 PLAIN - a picture in your head

1. Imagine a clerk working through a pile of forms, with a bell on the desk.
2. Anyone needing attention rings the bell rather than standing in the queue.
3. The clerk puts a bookmark in the form they are on, deals with the bell, then goes back to the exact same line of the same form.
4. There are many bells, one per department, all wired into a small box on the wall that ranks them and passes the most important one through.
5. Beside the desk is a printed list: bell number 1 means the front door, bell number 4 means the fire door.
6. If a bell means "the delivery arrived", the clerk does not unload the van right then. They write "van outside" on a note, and go back to the forms. The unloading happens between forms.
7. That split is the top half and the bottom half.

Where this comparison breaks: a clerk can ignore a bell. A processor with interrupts enabled cannot, which is why a device stuck ringing its bell can freeze a machine completely. That failure is called an interrupt storm.

13.5.3 PLAIN - a worked example

1. You press the letter A on an old-style keyboard attached to a PC.
2. The keyboard controller sees a key change and raises interrupt request 1.
3. The interrupt controller checks that IRQ 1 is not masked and that nothing more important is being served, then signals the processor.
4. The processor finishes its current instruction and reads the interrupt number, 0x21 in the classic mapping.
5. It looks up entry 0x21 in the interrupt descriptor table and jumps to the handler address stored there.
6. The handler reads one byte from port 0x60. That byte is a scan code, not a letter. For A it is 0x1E on press and 0x9E on release.
7. The handler puts the scan code in a queue and returns immediately. Total time: a few microseconds.
8. Later, ordinary kernel code takes the scan code out of the queue, maps it through the current keyboard layout, and delivers the character A to whichever window has focus.
9. Note what did not happen inside the handler: no layout lookup, no window search, no drawing.

13.5.4 PLAIN - what is really happening inside

1. In the oldest design, an interrupt is literally a wire. The device pulls the wire and holds it until it is serviced. This is level-triggered.
2. A wire per device does not scale, so several devices shared one wire and the handler had to ask each of them "was it you?".
3. Message-signalled interrupts replaced the wire completely. The device does a normal memory write to a special address, and that write is the interrupt. No extra wire at all.
4. The interrupt controller's jobs are: collect requests, apply masks, rank by priority, and hand the processor a number.
5. The number is an index into a table of handler addresses.
6. Before jumping, the processor pushes enough state to return: the instruction pointer, the flags, and on x86 the code segment.
7. The handler runs with further interrupts of the same or lower priority blocked, so it must not wait for anything.
8. The handler acknowledges the interrupt, so the controller knows it may deliver the next one. Forgetting this leaves the machine wedged.
9. Then it does the smallest possible urgent job, and queues the rest.
10. The queued rest runs later with interrupts enabled again, so it may take locks, allocate memory, and wake up sleeping processes.
11. At very high rates, even a short handler costs too much. A network card receiving 1 million packets per second cannot afford 1 million interrupts.
12. So the device is told to wait: fire one interrupt per 32 packets, or one every 50 microseconds, whichever comes first. That is coalescing. It trades a little latency for a lot of processor time.

13.5.5 TECHNICAL - the engineer's version

1. The original IBM PC used one Intel 8259A PIC with 8 inputs. The 1984 PC/AT cascaded a second 8259A into IRQ 2, giving 15 usable lines.
2. Classic IRQ assignments, still visible in firmware and in old manuals. These are a convention set by the IBM PC and PC/AT, not a written standard, but every clone copied them and software depended on them:

IRQ	Assigned to	Note
0	System timer	8253/8254 PIT
1	Keyboard	Port 0x60
2	Cascade to slave	Not usable directly
6	Floppy controller	Legacy
12	PS/2 mouse	Port 0x60 also
14	Primary ATA	Legacy disk

3. The APIC (Advanced Programmable Interrupt Controller) arrived with the Pentium era. Each core has a local APIC; an IO APIC, such as the Intel 82093AA, routes external lines. This allows routing an interrupt to a chosen core and supports 24 inputs per IO APIC.
4. MSI (message-signalled interrupts) was added in the PCI 2.2 specification. A device performs a posted memory write to an address in the range 0xFEE00000 on x86, carrying a vector in the data payload.
5. MSI allows up to 32 vectors per device in the PCI specification, though Windows uses at most 16. MSI-X, added in PCI 3.0, allows up to 2048 vectors, each with its own address, data and mask bit in a table in device memory.
6. MSI-X is what makes multi-queue devices work: one vector per queue, each pinned to a different core, so a 32-core machine can service a NIC on all 32 cores without lock contention.
7. x86 has 256 interrupt vectors, 0 to 255. Vectors 0 to 31 are reserved for processor exceptions such as divide error (0), page fault (14) and general protection fault (13). Devices use 32 upward.
8. In real mode the table is the IVT at physical address 0, 256 entries of 4 bytes. In protected and long mode it is the IDT, 256 entries of 8 or 16 bytes, located by the IDTR register and loaded with LIDT.
9. Linux names the split top half and bottom half. The top half is the handler registered with `request_irq`. Bottom halves are softirqs, tasklets, threaded IRQ handlers, or workqueues, in increasing order of how much they are allowed to do.
10. Windows names it differently: the ISR runs at device IRQL, and defers work to a DPC (deferred procedure call) at DISPATCH_LEVEL.
11. Coalescing on Linux NICs is controlled with `ethtool -c eth0` to read and `ethtool -C eth0 rx-usecs 50` to set. NVMe uses a similar aggregation threshold and time window.
12. Read live counts on Linux:

```
cat /proc/interrupts          # per-CPU counts, per source
watch -n1 'grep nvme /proc/interrupts'
```

13. On macOS there is no /proc; use `ioreg -l` for device trees and the Instruments tool for interrupt activity. On Windows, Performance Monitor exposes “Interrupts/sec” and “DPCs Queued/sec”.
14. An interrupt storm shows as one line in /proc/interrupts climbing by hundreds of thousands per second while the machine is idle. The usual cause is a shared level-triggered line whose owner never deasserts.

13.5.6 WORDS - remember these

1. Interrupt — a device asking for attention — an asynchronous transfer of control to a handler.
2. IRQ — the request number — an interrupt request line or its assigned index.
3. Vector — the table entry number — an index into the IDT selecting a handler.
4. ISR — the code that runs on an interrupt — the interrupt service routine, running with interrupts partly masked.
5. Top half and bottom half — urgent bit now, slow bit later — hard IRQ context versus softirq, tasklet, workqueue or DPC.
6. MSI-X — an interrupt sent as a memory write — message-signalled interrupts with up to 2048 per-vector table entries.

7. Coalescing — batching interrupts — delaying notification by a packet count or time threshold to cut CPU overhead.

13.6 USB, properly

13.6.1 PLAIN - in simple words

1. Before 1996, every kind of device had its own kind of plug.
2. A mouse had one plug, a keyboard another, a printer a third, a modem a fourth, and a joystick a fifth.
3. Each of them needed its own settings, and each machine had only one or two of each socket.
4. USB, the Universal Serial Bus, was made to replace all of them with one plug that anyone could use.
5. Its three promises were: one connector for everything, plug it in while the machine is running, and no manual settings.
6. It also carries power, so small devices need no separate supply.
7. USB is not a shared party line. There is one boss, called the host, and everything else answers only when asked.
8. Devices never speak first. Even a mouse only replies when the host asks it for an update, which the host does about once every millisecond.
9. You can add more sockets by plugging in a hub, which is a splitter that passes the host's questions along.
10. When you plug something in, the machine goes through a fixed set of steps to find out what it is. That process is called enumeration.

13.6.2 PLAIN - a picture in your head

1. Think of a teacher in a classroom who alone decides who speaks.
2. The teacher goes around asking each pupil in turn: "anything for me?"
3. Most pupils say "nothing". A mouse says "nothing" a thousand times a second, then once says "moved left by three".
4. No pupil ever shouts out. If two spoke at once, neither would be heard.
5. Group leaders sit between the teacher and the pupils, passing questions down and answers back. Those are hubs.
6. A new pupil arriving at the door is noticed by the nearest group leader, who tells the teacher. The teacher then asks the newcomer their name, gives them a seat number, and asks what subject they study.

Where this comparison breaks: USB 3 and later added a second set of wires that lets a device signal readiness rather than being asked, which is closer to raising a hand. And USB On-The-Go lets a phone act as the teacher instead of the pupil. The strict one-boss model is true of USB 2 and earlier.

13.6.3 PLAIN - a worked example

1. Plug a keyboard into a running computer. Here is the exact sequence.

```
1 attach     device pulls one data line high through a
             resistor; the hub sees the change
2 report     hub tells host: something on port 3
3 reset      host tells hub to reset that port
4 default    device now answers at address 0, endpoint 0
5 read 8     host reads first 8 bytes of the device
             descriptor to learn the max packet size
6 address    host sends SET_ADDRESS with a free number
             from 1 to 127; device switches to it
7 descript   host reads full device, configuration,
             interface and endpoint descriptors
8 match      host looks up class, vendor and product
```

```
             IDs and picks a driver
9 configure  host sends SET_CONFIGURATION; device
             powers up its interfaces and starts work
```

2. In step 7 the host learns that this device reports interface class 3, which means human interface device, subclass 1 and protocol 1, which mean boot keyboard.
3. That is enough. No vendor driver is needed. The operating system's own HID keyboard driver handles it.
4. Total time from plug to working: typically well under a second.
5. Now the steady state. The host polls the keyboard's interrupt endpoint every 8 milliseconds by default for a low-speed keyboard.
6. Each poll returns 8 bytes: one byte of modifier keys, one reserved byte, and up to six key codes held down at once.
7. That six-key limit is why cheap keyboards cannot report many simultaneous key presses in boot protocol.

13.6.4 PLAIN - what is really happening inside

1. The shape is a tiered star. The host controller is the root. Hubs form the branches. Devices are the leaves.
2. The specification allows 7 tiers counting the root, which means at most 5 hubs between the host and a device, and up to 127 addresses in total.
3. Inside a device, the things the host talks to are called endpoints. An endpoint is a buffer with a number and a direction.
4. Endpoint 0 always exists, works both ways, and is used for setup and control. Every device has it.
5. A logical connection between the host's driver and one endpoint is called a pipe.
6. There are four kinds of transfer, and a device picks one per endpoint.
7. **Control**: short, guaranteed, used for setup and commands. Endpoint 0 only.
8. **Bulk**: large amounts, correctness guaranteed, timing not guaranteed. Used by drives and printers. It gets whatever bandwidth is left over.
9. **Interrupt**: small and regular. The host promises to ask at least this often. Used by mice, keyboards and touchpads. The name is misleading: nothing interrupts anything, the host simply polls on a schedule.
10. **Isochronous**: a guaranteed slice of bandwidth every frame, with no retries. Used by webcams, microphones and audio interfaces, where a late packet is worse than a lost one.
11. Time is divided into frames of 1 millisecond at full speed, and microframes of 125 microseconds at high speed. Bandwidth is reserved inside each frame.
12. Interrupt and isochronous endpoints get their reservation first. If not enough is left, the device is refused, which is the real reason a webcam sometimes fails to start when other devices share a controller.

13.6.5 TECHNICAL - the engineer's version

1. USB 1.0 was released in January 1996 by a promoter group including Compaq, DEC, IBM, Intel, Microsoft, NEC and Nortel. Ajay Bhatt led the effort at Intel. USB 1.1, in September 1998, was the version that actually worked in the field.
2. Speeds and dates:

Version	Year	Raw rate	Old marketing
1.0 Low	1996	1.5 Mbit/s	Low Speed
1.1 Full	1998	12 Mbit/s	Full Speed
2.0	2000	480 Mbit/s	High Speed
3.0	2008	5 Gbit/s	SuperSpeed
3.1	2013	10 Gbit/s	SuperSpeed+

Version	Year	Raw rate	Old marketing
3.2	2017	20 Gbit/s	SuperSpeed 20Gbps
USB4	2019	20 or 40 Gb/s	USB4 Gen 2 / 3
USB4 v2.0	2022	80 Gbit/s	USB 80Gbps

3. The naming is a genuine mess and the USB-IF made it worse twice. When USB 3.1 arrived, plain USB 3.0 was renamed USB 3.1 Gen 1. When USB 3.2 arrived, both were renamed again to USB 3.2 Gen 1 and USB 3.2 Gen 2. So a 5 Gbit/s port has had three official names for the same silicon.
4. As of 2026 the USB-IF's guidance is to drop the version numbers in consumer labelling and use speed names only: USB 5Gbps, USB 10Gbps, USB 20Gbps, USB 40Gbps and USB 80Gbps. Product pages still use the old names, so expect both.
5. Mapping: USB 3.2 Gen 1x1 is 5 Gbit/s, Gen 2x1 is 10 Gbit/s, Gen 2x2 is 20 Gbit/s using two lanes. USB4 Gen 3x2 is 40 Gbit/s and USB4 Gen 4x2 is 80 Gbit/s. USB4 Version 2.0 also defines an asymmetric mode of 120 Gbit/s one way and 40 Gbit/s the other, for high-resolution displays.
6. Encoding overhead is real. USB 3.0 at 5 Gbit/s uses 8b/10b, so the usable ceiling is 500 MB/s before protocol overhead, and real drives reach around 400 to 450 MB/s.
7. Power. USB 2.0 defines a unit load of 100 mA at 5 V, with up to 5 units, so 500 mA or 2.5 W. USB 3.x raises the unit load to 150 mA with up to 6 units, so 900 mA or 4.5 W.
8. USB Power Delivery negotiates far more. PD revision 1.0 reached 100 W. PD revision 3.1, published in 2021, added Extended Power Range with 28 V, 36 V and 48 V fixed levels, reaching 240 W at 48 V and 5 A. A cable must be electronically marked to carry 5 A.
9. USB Type-C is a **connector**, defined in a specification finalized in August 2014, with 24 pins and reversible insertion. It is not a speed and not a protocol.
10. This is the single most common confusion in the whole subject. A USB-C socket may run USB 2.0 at 480 Mbit/s and nothing else, or it may carry USB 80Gbps plus DisplayPort plus PCIe plus 240 W. The shape tells you nothing. Only the logo and the specification sheet tell you.
11. Device classes are defined by the USB-IF so that one driver serves many products. Class 1 audio, class 3 HID, class 6 still image, class 7 printer, class 8 mass storage, class 9 hub, class 14 video, class 255 vendor-specific.
12. HID, defined in 1996 and revised as HID 1.11 in 2001, is why a keyboard from any maker works with no download. It uses a self-describing report descriptor, so the device tells the host the layout of its own data.
13. Boot protocol is the cut-down fixed 8-byte HID keyboard format that firmware can drive before any operating system loads. That is why your keyboard works in the BIOS setup screen.
14. Vendor and product IDs are 16-bit each and assigned by the USB-IF. For example 0x046D is Logitech.
15. Observe on Linux with `lsusb -t` for the topology tree and `lsusb -v` for descriptors. On macOS, `system_profiler SPUSBDataType`. On Windows, Device Manager with "Devices by connection" selected.

```
/:  Bus 01.Port 1: Dev 1, Class=root_hub, 480M
    |__ Port 2: Dev 3, If 0, Class=HID, Driver=usbhid, 1.5M
    |__ Port 4: Dev 5, If 0, Class=Mass Storage, 480M
```

13.6.6 WORDS - remember these

1. Host controller — the boss chip on the machine — the xHCI controller issuing all transactions.
2. Hub — a splitter — a device that repeats host traffic to more ports and reports attachments.
3. Endpoint — a numbered buffer in a device — a source or sink with a direction and transfer type.
4. Pipe — a connection to one endpoint — the logical channel between host driver and endpoint.
5. Enumeration — finding out what was plugged in — reset, address assignment, descriptor read and configuration.
6. Descriptor — the device's self-description — the byte structures giving IDs, classes, endpoints and strings.

7. Device class — a family with a standard driver — the interface class code that selects a generic driver.
8. USB-C — the oval reversible plug — a 24-pin connector specification, not a speed or a protocol.

13.7 The other ports on the machine

13.7.1 PLAIN - in simple words

1. Not every socket on a machine is USB. Each remaining one exists because it does something USB did not do well enough at the time.
2. Thunderbolt looks exactly like USB-C but can carry far more, including a graphics card's connection and a screen's picture at the same time.
3. HDMI and DisplayPort carry picture and sound to a screen.
4. The square-ish clicky socket is Ethernet, for a network cable.
5. The small round holes are audio jacks, for headphones and microphones.
6. The thin slot is for SD cards, used mostly by cameras.
7. The old wide sockets, serial and parallel, have almost gone from home machines but are alive in factories and laboratories.

13.7.2 PLAIN - a picture in your head

1. Think of USB as a general delivery van that handles most parcels.
2. Thunderbolt is a motorway tunnel. Other roads drive through it unchanged and come out the other end still themselves.
3. That is the key idea: Thunderbolt does not convert the traffic. It wraps it, carries it, and unwraps it.
4. So a graphics card in a box on your desk really is talking to the processor over its normal card connection. The cable just carries it.

Where this comparison breaks: a tunnel has fixed lanes. Thunderbolt shares one budget between video and data, so a very high resolution screen leaves less room for the drive plugged into the same dock.

13.7.3 PLAIN - a worked example

1. You plug a laptop into one dock with a single USB-C cable, and get a 4K screen, a wired network, three USB ports and charging.
2. Inside the cable there is one Thunderbolt link.
3. The screen's picture rides inside it as DisplayPort.
4. The network chip and the USB ports on the dock sit behind a PCI Express switch, and that PCI Express traffic also rides inside it.
5. Power travels on separate power pins, negotiated by USB Power Delivery.
6. Take the same cable and plug in a second 4K screen at 120 Hz. Something will drop in quality or refresh rate, because the video share of the budget is now full.

13.7.4 PLAIN - what is really happening inside

1. Thunderbolt builds a small switched network in the cable. Each end has a controller chip that packs, routes and unpacks.
2. Because it tunnels PCI Express, a Thunderbolt device is, to the operating system, a card plugged into the machine. This is why external graphics boxes work and also why Thunderbolt security matters.
3. HDMI and DisplayPort differ in origin. HDMI grew from television and carries a clocked stream of pixels. DisplayPort grew from computing and sends packets, which is why it can drive several screens down one cable.
4. Ethernet's eight-pin socket uses two or four twisted pairs. Twisting the pairs is what cancels interference over 100 metres of cable.
5. A headphone jack with three black bands has four conductors: left, right, ground and microphone. That is what TRRS means: tip, ring, ring, sleeve.

6. An SD card is a small controller chip plus flash memory. The card, not the camera, decides where data physically goes.

13.7.5 TECHNICAL - the engineer's version

1. Thunderbolt began as Intel's Light Peak research and shipped in February 2011 on the MacBook Pro with a Mini DisplayPort connector.

Version	Year	Bandwidth	Connector
TB 1	2011	10 Gbit/s	Mini DisplayPort
TB 2	2013	20 Gbit/s	Mini DisplayPort
TB 3	2015	40 Gbit/s	USB-C
TB 4	2020	40 Gbit/s	USB-C
TB 5	2023	80 Gbit/s	USB-C

2. Thunderbolt 4 did not raise the headline speed over Thunderbolt 3. It raised the minimum requirements: 32 Gbit/s of PCIe tunnelling, two 4K displays, and mandatory DMA protection through the IOMMU.
3. Thunderbolt 5, announced by Intel in September 2023 with the controller codenamed Barlow Ridge, provides 80 Gbit/s in both directions and an asymmetric Bandwidth Boost mode of 120 Gbit/s out and 40 Gbit/s back. It tunnels PCIe 4.0 x4, about 64 Gbit/s. It is built on USB4 Version 2.0.
4. DMA protection is not optional history. The Thunderspy work published in 2020 showed pre-2019 Thunderbolt machines could be compromised through direct memory access from a malicious device.
5. Display standards: HDMI 1.0 in 2002; HDMI 2.1 in 2017 at 48 Gbit/s; HDMI 2.2, announced at CES in January 2025 and published in mid-2025, at 96 Gbit/s with "Ultra96" certified cables. DisplayPort 1.0 came from VESA in 2006; DisplayPort 2.0 in 2019 defines UHBR20 at 80 Gbit/s raw; DisplayPort 2.1b was announced alongside HDMI 2.2 with DP80LL cables.
6. HDMI is licensed by HDMI Licensing Administrator with per-unit royalties. DisplayPort is royalty-free from VESA. That commercial difference, not a technical one, is why televisions use HDMI and monitors often use both.
7. Ethernet on copper: 10BASE-T from 1990, 100BASE-TX from 1995, 1000BASE-T from IEEE 802.3ab in 1999, 10GBASE-T from 802.3an in 2006, and 2.5GBASE-T and 5GBASE-T from 802.3bz in 2016. The connector is an 8P8C modular plug, almost always called RJ45 by convention rather than by standard.
8. Audio jacks are 3.5 mm. TRS is three conductors and stereo out. TRRS is four and adds a microphone. Two incompatible TRRS wirings exist: CTIA, used by nearly everything today, and OMTP, used by some older phones, with microphone and ground swapped. The colour code, green for line out and pink for microphone, comes from the Microsoft and Intel PC 99 guide.
9. SD cards were introduced in August 1999 by SanDisk, Panasonic and Toshiba, with the SD Association formed in January 2000. Capacity tiers: SDSC to 2 GB, SDHC to 32 GB from 2006, SDXC to 2 TB from 2009, SDUC to 128 TB from
2018. Bus modes run from 12.5 MB/s default speed through UHS-I at 104 MB/s and UHS-II at 312 MB/s to SD Express, which puts PCIe and NVMe on the card and reaches into the gigabytes per second.
10. Serial: the EIA RS-232 standard dates from 1960. The PC/AT gave it a DE-9 plug in 1984. COM1 sits at port 0x3F8 on IRQ 4. It survives on network switch consoles, laboratory instruments, industrial controllers and embedded debug headers, usually now through a USB-to-serial adapter using an FTDI or Silicon Labs chip.
11. Parallel: the Centronics printer interface, standardized as IEEE 1284 in 1994, at port 0x378 on IRQ 7. It survives on old industrial machines and, for hobbyists, as a source of simple bit-banged input and output.

13.7.6 WORDS - remember these

1. Tunnelling — carrying one protocol inside another — encapsulating PCIe and DisplayPort inside Thunderbolt or USB4 transport.
2. Thunderbolt — the high-capacity USB-C link — an Intel interconnect tunnelling PCIe and DisplayPort, now aligned with USB4.
3. TRRS — the four-part headphone plug — tip, ring, ring, sleeve, adding a microphone conductor, in CTIA or OMTP wiring.
4. 8P8C — the network socket — the eight-position eight-contact modular connector commonly called RJ45.
5. UHS — the faster SD card mode — Ultra High Speed bus modes I, II and III.

13.8 What a driver actually is

13.8.1 PLAIN - in simple words

1. A driver is a piece of code that translates.
2. On one side it accepts a general request that says nothing about hardware, such as “write these bytes”.
3. On the other side it produces the exact numbers that one particular chip needs, written into that chip’s exact registers, in that chip’s exact order.
4. That is the whole job. A translator between the general and the specific.
5. It exists because there are thousands of different chips that do the same job in different ways.
6. Without drivers, every program would need code for every chip ever made.
7. With drivers, a program says “write these bytes” and never learns what kind of drive it is.
8. A driver is not a program you run. It is code the operating system loads and calls when needed.
9. Most drivers live inside the operating system’s core, so a fault in one can bring down the whole machine.
10. The operating system chooses which driver to load by matching numbers the device reports about itself.

13.8.2 PLAIN - a picture in your head

1. Imagine an international company where head office issues one instruction: “ship this box to the customer”.
2. Each country’s office knows its own local rules, forms, languages and couriers.
3. Head office never learns those rules. It only knows the one instruction.
4. Each local office is a driver. It turns one general instruction into the specific local procedure.
5. Replace a courier in one country and only that one office changes. Head office is untouched.
6. Now imagine a local office that files a form wrongly and locks the whole company’s systems. That is what a bad kernel driver does.

Where this comparison breaks: a local office can refuse an instruction it does not understand. A driver is usually trusted, running with the same power as the operating system core, so it is believed even when it is wrong.

13.8.3 PLAIN - a worked example

1. Follow one key press from finger to screen. This is the exact chain.

```
finger presses key
  -> keyboard matrix scan in keyboard's own chip
  -> USB HID report, 8 bytes, on an interrupt endpoint
  -> host controller (xHCI) driver takes the transfer
  -> USB core hands the report to the matching driver
  -> HID class driver decodes the report descriptor
  -> input subsystem turns it into a key event
  -> keymap turns key code into a character
```

```
   -> window system sends the character to the focused app
   -> the app draws the letter
```

2. Now follow writing a file. The chain runs the other way.

```
app calls write(fd, buf, 4096)
   -> system call: user mode traps into the kernel
   -> VFS picks the filesystem for that mount
   -> ext4 or APFS or NTFS turns it into block writes
   -> block layer queues and merges the requests
   -> NVMe class driver builds a command in a queue
   -> PCIe driver has already mapped the device's BARs
   -> doorbell register write tells the SSD to look
   -> SSD firmware does wear levelling and writes flash
   -> completion queue entry plus MSI-X interrupt
   -> the kernel wakes the waiting application
```

3. Note how many layers are generic and how few are device-specific.
4. Only two boxes in each chain know what chip this actually is.

13.8.4 PLAIN - what is really happening inside

1. The layers, from top to bottom, always look like this.
2. **Application**: a normal program. Knows nothing about hardware.
3. **System call**: the doorway into the kernel. The processor switches from user mode to kernel mode.
4. **Kernel subsystem**: the generic manager for a category, such as the file layer, the network stack, or the input layer. Defines the general request.
5. **Class driver**: code for a whole family of devices that follow one standard, such as USB mass storage or NVMe. It knows the standard, not the brand.
6. **Bus driver**: code that knows how to find and reach devices on a particular bus, such as PCIe or USB. It enumerates devices, reads their IDs, and hands out address windows and interrupts.
7. **Device driver**: code for one particular chip or chip family. It knows the register map, the quirks, and the initialization sequence.
8. **Hardware**: the chip itself.
9. Not every chain has every layer. A well-standardized device may need no vendor-specific driver at all, because the class driver is enough.
10. That is why a USB stick works instantly and a graphics card does not: the stick follows a class standard, the graphics card is a custom machine with a proprietary register map.
11. Requests travel down and completions travel up. Down is usually a call. Up is usually an interrupt followed by a callback.

13.8.5 TECHNICAL - the engineer's version

1. A driver is defined by the interface it implements, not by what it contains. On Linux it registers a `struct file_operations` or a bus- specific `struct pci_driver` with `probe` and `remove` callbacks.
2. Minimal shape of a Linux PCI driver:

```
static struct pci_device_id ids[] = {
    { PCI_DEVICE(0x8086, 0x100e) },   /* Intel 82540EM */
    { 0, }
};
MODULE_DEVICE_TABLE(pci, ids);

static struct pci_driver drv = {
    .name     = "mydrv",
    .id_table = ids,
    .probe    = my_probe,
```

```
    .remove   = my_remove,
};
```

3. `probe` is called by the bus core once a device matching the ID table is found. It maps BARs with `pci_iomap`, requests an interrupt with `pci_alloc_irq_vectors` and `request_irq`, sets bus mastering with `pci_set_master`, and registers with the relevant subsystem.
4. Vendor ID 0x8086 is Intel, chosen as a reference to the 8086 processor. 0x10DE is NVIDIA, 0x1002 is AMD graphics, 0x10EC is Realtek.
5. Windows layers the same idea as a device stack of driver objects: a bus driver creates a physical device object, and function and filter drivers attach device objects above it. Requests travel as IRPs (IO request packets) down the stack.
6. macOS and iOS use IOKit, an object-oriented C++ driver framework with a matching dictionary per driver, and a live registry visible with `ioreg`.
7. The system call boundary is the hard security line. On x86-64 Linux the instruction is `syscall`; on ARM64 it is `svc #0`. Everything above it runs unprivileged; everything below can touch any memory.
8. Timings that explain the layering, order of magnitude only:

Step	Typical cost
Function call	About 1 nanosecond
System call	50 to 400 nanoseconds
Interrupt entry/exit	Around 1 microsecond
NVMe read completion	20 to 100 microsecond

9. This cost table is why fast devices use polling again in some paths. Linux NAPI polls a busy network card instead of taking an interrupt per packet, and `io_uring`, merged in Linux 5.1 in 2019, exists to avoid system calls per operation. Polling returned not because interrupts got worse but because devices got faster than interrupt handling.
10. Observe the stack on Linux with `lsmod`, `modinfo nvme`, and `ls /sys/bus/pci/drivers/`. On Windows, `pnputil /enum-drivers` and the Device Manager driver tab. On macOS, `kmutil showloaded` and `ioreg -l`.

13.8.6 WORDS - remember these

1. Driver — the translator for one device — code implementing a kernel interface and driving a specific register map.
2. Class driver — one driver for a whole family — a driver implementing a device class standard such as HID or NVMe.
3. Bus driver — the code that finds devices — enumerates a bus, reads IDs and allocates resources.
4. System call — the doorway into the kernel — a controlled privilege transition such as `syscall` or `svc`.
5. Probe — the moment a driver claims a device — the callback the bus core makes when an ID matches.
6. Device stack — the chain of code over one device — layered driver objects passing requests down and completions up.

13.9 Driver mechanics

13.9.1 PLAIN - in simple words

1. Code on a machine runs in one of two worlds.
2. In user space, code is fenced in. If it goes wrong, only it dies.
3. In kernel space, code can touch everything. If it goes wrong, the machine stops.
4. Most drivers live in kernel space, because they need direct access to hardware and must respond in microseconds.

5. Modern systems move as many drivers as possible into user space, giving up a little speed for a lot of safety.
6. Drivers are usually not built into the operating system. They are separate files loaded when a matching device appears.
7. Because a driver has total power, operating systems now refuse to load one unless it carries a valid digital signature.
8. The signature does not prove the driver is good. It proves who made it and that nobody changed it since.
9. Each operating system has its own driver framework, with different names for the same ideas.
10. Matching is done by numbers: the device says "I am vendor 0x8086, product 0x100E", and the system looks for a driver that claims that pair.

13.9.2 PLAIN - a picture in your head

1. Think of a hospital. Visitors stay in the public corridors. Surgeons enter the operating theatre.
2. User space is the corridor. A visitor who faints causes a small incident.
3. Kernel space is the theatre. A surgeon who faints mid-operation is a disaster for the patient.
4. So hospitals check credentials at the theatre door, and keep as much work as possible out in the corridors.
5. Driver signing is the credential check. It says this person is who they claim, and their badge has not been forged.
6. It does not say they are a competent surgeon.

Where this comparison breaks: a hospital can eject a bad surgeon mid- operation. An operating system usually cannot stop a kernel driver that has already corrupted memory. By the time the fault is visible, the damage is done.

13.9.3 PLAIN - a worked example

1. On Linux, you plug in a USB-to-serial adapter. Watch what happens.

```
$ lsusb
Bus 001 Device 007: ID 0403:6001 Future Technology
   Devices International, Ltd FT232 Serial (UART) IC

$ dmesg | tail -3
usb 1-2: new full-speed USB device number 7
usbcore: registered new interface driver ftdi_sio
usb 1-2: FTDI USB Serial Device now attached to ttyUSB0

$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188, 0 Aug 13 10:22 /dev/ttyUSB0
```

2. Read that carefully. 0403 is the vendor ID for FTDI. 6001 is the product ID for the FT232 chip.
3. The kernel formed a modalias string from those numbers, looked it up, and loaded the module named ftdi_sio.
4. The module then created a device node, /dev/ttyUSB0.
5. `188, 0` are the major and minor numbers. 188 identifies the driver, 0 identifies which of its devices this is.
6. From now on, any program can just open /dev/ttyUSB0 like a file. It never needs to know the chip is an FT232.

13.9.4 PLAIN - what is really happening inside

1. A loadable module is compiled machine code with a table of symbols it needs and a table of what it provides.

2. Loading it means: copy it into kernel memory, resolve its references to kernel functions, run its init function, and register it with the bus core.
3. The bus core then walks its list of known devices and calls the new driver's probe function for each match.
4. Unloading runs the reverse, but only if nothing is using the driver.
5. A kernel driver shares one address space with the whole kernel. There is no memory protection between it and everything else.
6. So a wild pointer write in a driver silently corrupts unrelated kernel data, and the crash may appear minutes later somewhere else entirely.
7. A user-space driver runs as an ordinary restricted process. It reaches hardware only through a narrow, checked path the kernel opens for it.
8. If it crashes, the kernel closes the handle, resets the device, and can restart the driver. The machine survives.
9. The trade is latency and copies. Every crossing between user space and the kernel costs time, so the fastest devices keep kernel drivers.

13.9.5 TECHNICAL - the engineer's version

1. **Windows.** WDM (Windows Driver Model), introduced with Windows 98 and Windows 2000, is the low-level model of IRPs and device objects. It is powerful and unforgiving.
2. WDF (Windows Driver Frameworks) sits above WDM and handles the hard parts, chiefly power management and plug-and-play state. It comes in two forms: KMDF (Kernel-Mode Driver Framework) and UMDF (User-Mode Driver Framework). As of Windows 11 the shipping versions are KMDF 1.33 and UMDF 2.33. Microsoft publishes the framework source on GitHub.
3. UMDF version 2 uses the same object model and function names as KMDF, so a driver can be moved between kernel and user mode with modest changes. UMDF version 1, which used COM, is superseded.
4. Windows drivers are described by an INF text file listing hardware IDs, which look like `PCI\VEN_10DE&DEV_2484&SUBSYS_...&REV_A1` or `USB\VID_046D&PID_C52B`. Matching is by longest, most specific ID first.
5. Windows driver signing history: Windows Vista x64 in 2006 first required kernel-mode drivers to be signed. Since Windows 10 version 1607, released in August 2016, new kernel-mode drivers must be signed by Microsoft through the hardware developer portal, submitted with an EV certificate, either as WHQL-certified or attestation-signed.
6. **Linux.** Drivers are usually loadable kernel modules, files ending in `.ko`, loaded with `modprobe` or automatically by udev. `lsmod` lists them; `modinfo` shows their aliases and parameters.
7. Character and block devices appear as nodes under `/dev`, created automatically by devtmpfs and managed by udev, in practice `systemd-udevd` on most distributions. Rules live in `/etc/udev/rules.d` and `/lib/udev/rules.d`.
8. `/sys`, the sysfs filesystem introduced in Linux 2.6 in 2003, exposes the whole device model as directories: `/sys/bus/pci/devices`, `/sys/class/net`, `/sys/devices/...`. It is how userspace discovers hardware without ioctls.
9. Matching uses modalias strings. Read one directly:

```
$ cat /sys/bus/usb/devices/1-2/modalias
usb:v0403p6001d0600dc00dsc00dp00ic FFisc FFip FFin00
```

10. Linux has no stable in-kernel driver ABI, by deliberate policy. Drivers are expected to be merged into the mainline tree, where they are updated with the rest of the kernel. Out-of-tree drivers, such as NVIDIA's, must be rebuilt for each kernel, usually through DKMS.
11. Experts genuinely disagree here. Kernel developers argue the unstable ABI forces good code into the tree and lets interfaces improve. Vendors argue it makes proprietary drivers painful. Both are describing the same fact, valuing it differently.

12. Linux also supports user-space drivers: FUSE for filesystems, libusb and usbfs for USB devices, VFIO and UIO for direct device access with IOMMU protection, and SPDK and DPDK for polled user-space storage and network drivers.
13. **macOS.** Kexts (kernel extensions) built on IOKit were the traditional model. Apple announced their deprecation at WWDC in June 2019 and introduced DriverKit, which builds driver code as a user-space system extension using the same IOKit-style class model.
14. On Apple silicon Macs, loading a kext requires switching to Reduced Security in recoveryOS and approving it, then rebooting. As of 2026 kexts still load on macOS 26 under that reduced security setting, but Apple's documented guidance is to use DriverKit and system extensions.
15. Inspect on macOS with `kmutil showloaded`, `systemextensionsctl list` and `ioreg -l`.

System	Kernel driver	User-space driver
Windows	WDM, KMDF	UMDF 2
Linux	Kernel module .ko	FUSE, libusb, VFIO
macOS	Kext (IOKit)	DriverKit extension

16. The honest version: "user-space driver" rarely means no kernel code at all. A small kernel stub still owns the interrupt and the IOMMU mapping. What moves out is the large, complex, bug-prone part.

13.9.6 WORDS - remember these

1. Kernel space — where trusted code runs — the privileged address space with no memory protection between components.
2. Kernel module — a driver file loaded on demand — a relocatable object linked into the running kernel.
3. Device node — the file that represents a device — a character or block special file with major and minor numbers.
4. sysfs — the folder view of hardware — the kernel object hierarchy exported under `/sys`.
5. udev — the thing that creates device files — the userspace device manager acting on kernel uevents.
6. Driver signing — a proof of who wrote it — a cryptographic signature the loader verifies against trusted keys.
7. DriverKit — Apple's safer driver system — a user-space system extension framework replacing kexts.
8. Hardware ID — the number that picks the driver — a vendor and product ID string used for driver matching.

13.10 Firmware: code that lives on the device

13.10.1 PLAIN - in simple words

1. Firmware is software that lives inside a piece of hardware rather than on your disk.
2. It runs on a small processor inside the device itself, not on your main processor.
3. It is there before your operating system starts, and it keeps running after your operating system stops.
4. Almost every device has some. A drive, a keyboard, a network chip, a screen, a mouse, a charger.
5. It is called firm-ware because it sits between hardware, which cannot change, and software, which changes all the time.
6. Long ago it truly could not be changed. Today it is stored in flash memory and can be rewritten.
7. That is useful, because bugs in firmware are common and often serious.
8. It is also dangerous, because a failed rewrite can leave a device that cannot start at all.

13.10.2 PLAIN - a picture in your head

1. Think of a vending machine. The metal box is the hardware.
2. The instructions inside its controller, deciding what happens when you press B4, are the firmware.

3. The price list you load in is data. The app you use to pay is software on your phone.
4. Changing the firmware changes what the machine does, permanently, even with nobody around.
5. And if the engineer's update fails halfway, the machine will not open its door for anyone, because the instructions are now half old and half new.

Where this comparison breaks: a broken vending machine can be opened with a key. Many devices have no recovery route once their only firmware image is damaged, which is what "bricked" means.

13.10.3 PLAIN - a worked example

1. Here is where firmware sits in one ordinary laptop.

Device	Firmware does what
Motherboard flash	Starts the machine
SSD controller	Wear levelling, mapping
Keyboard chip	Scans the key matrix
WiFi module	Runs the radio timing
GPU (VBIOS)	Powers up, sets display
Monitor scaler	Scaling, menus, inputs

2. Count them. That is six separate processors running six separate programs, before your operating system even loads.
3. A modern laptop typically contains between ten and thirty small processors you never see.

13.10.4 PLAIN - what is really happening inside

1. Firmware is stored in non-volatile memory, usually NOR flash for code that must run in place, or NAND flash with a small loader.
2. When power arrives, the device's own small processor starts at a fixed address inside that flash and begins executing.
3. Some devices do not store their firmware at all. They start empty, and the driver on your machine uploads the firmware image at every boot.
4. That is why some WiFi and graphics cards need a firmware package installed separately from the driver, and do nothing without it.
5. Updating firmware means erasing a block of flash and writing a new image, then verifying it.
6. If power is lost between the erase and the verify, the device has no valid image, and the small processor has nothing valid to run.
7. Careful designs avoid this with two images: write to the spare slot, verify it, then flip a pointer. If anything fails, the old image is untouched.

13.10.5 TECHNICAL - the engineer's version

1. Firmware update mechanisms in use today: UEFI capsule updates for system firmware, vendor tools for SSD and GPU firmware, DFU (device firmware upgrade) class for USB devices, and I2C or DDC/CI for monitor scalers.
2. On Linux, `fwupd` with the Linux Vendor Firmware Service delivers signed firmware for participating vendors. Commands: `fwupdmgr get-devices` and `fwupdmgr update`. On Windows, firmware arrives through Windows Update as a firmware driver package. On macOS, firmware ships inside system updates.
3. Real firmware bugs with real consequences. In November 2019 Hewlett Packard Enterprise issued an urgent bulletin: certain SAS solid state drives would fail permanently, with total data loss, at exactly 32,768 hours of power-on time. That number is 2 to the power 15, a signed 16-bit counter overflowing.

4. Samsung's 840 EVO drives, from 2013, slowed dramatically when reading old data, and the fix was a firmware change to the drive's read-retry and refresh behaviour, shipped in 2014 and again in 2015.
5. Intel and AMD ship processor microcode updates, which are firmware for the processor itself, loaded by the platform firmware or by the operating system at every boot. The Spectre and Meltdown mitigations of 2018 arrived partly this way.
6. Firmware is a security surface. It runs before any operating system protection exists, and it survives reinstalling the operating system. Attacks that persist in firmware are the reason for Boot Guard, signed capsules and measured boot into a TPM.
7. Redundancy schemes: dual-BIOS boards with a backup chip, A/B slots on phones and network gear, and USB BIOS Flashback, which rewrites the flash using a small always-on controller with no processor or memory installed.
8. The plain difference, stated once and clearly:

Thing	Where it lives	Who runs it
Firmware	Flash on the device	The device's chip
Driver	OS kernel or system	Your main CPU
Software	Your disk	Your main CPU

9. The honest version: the boundary is not sharp. A GPU driver contains blobs that get uploaded to the card and run there, which makes them firmware delivered by a driver. Classification follows what runs where, not which file it arrived in.

13.10.6 WORDS - remember these

1. Firmware — code inside the device — non-volatile program executed by an embedded controller.
2. Flashing — rewriting firmware — erasing and reprogramming non-volatile memory with a new image.
3. Bricked — dead after a failed update — an unbootable device with no valid firmware image and no recovery path.
4. Microcode — firmware for the processor — patches to the CPU's internal instruction decoding, loaded at boot.
5. DFU — the standard USB update mode — Device Firmware Upgrade class.
6. A/B slots — two copies of the firmware — redundant images with an atomic pointer switch for safe updates.

13.11 BIOS and UEFI

13.11.1 PLAIN - in simple words

1. When you press power, your processor has no operating system, no drivers, and no idea what is attached.
2. Something must wake it, test the basics, find a disk, and load the first piece of the operating system.
3. That something is the platform firmware, stored on the small flash chip on the board.
4. The old version, from 1981, was called BIOS, short for Basic Input Output System.
5. The modern replacement is called UEFI, short for Unified Extensible Firmware Interface.
6. BIOS was simple and limited: it loaded 512 bytes from the start of a disk and jumped into them.
7. UEFI is far larger. It can read a real filesystem, keep a list of things it can boot, and check signatures before running them.
8. Both do the same job. Only UEFI can handle large disks, many boot options and modern security.

13.11.2 PLAIN - a picture in your head

1. BIOS is a caretaker with one instruction: go to the front desk of the building, read the note pinned there, and do whatever it says.
2. The note is tiny, so it usually says "go to room 4 and read the longer note there".
3. UEFI is a caretaker with a real address book. It knows several buildings, remembers which one you preferred, and checks each visitor's identity card before letting them in.

Where this comparison breaks: the identity check, Secure Boot, only verifies the visitor at the door. Once the operating system kernel is running, it can do whatever it likes. Secure Boot does not keep watching.

13.11.3 PLAIN - a worked example

1. Here is the first second after you press the power button.

```
0 ms     power supply stabilizes, asserts POWER_GOOD
         the CPU is held in reset until then
~1 ms    reset released; CPU starts at its reset
         vector, the top of the address space
~2 ms    firmware runs from flash; no RAM works yet
         so cache is used as temporary memory
10-2000  memory training: the controller tunes the
  ms     timing of every DDR5 signal, then tests RAM
then     POST: check CPU, RAM, basic chips
then     enumerate PCIe and USB, assign addresses
then     read boot order from NVRAM settings
then     load the boot file from the EFI partition
then     hand control to the bootloader
```

2. Memory training is why a machine with new DDR5 memory can appear dead for thirty seconds on its very first boot, and start in three seconds after.
3. The result of the training is cached, so it only happens once.

13.11.4 PLAIN - what is really happening inside

1. POST means power-on self test. The firmware checks the processor, tests enough memory to work, and initializes the basic chips.
2. If it fails before a screen exists, it must report through beeps, board LEDs or a two-digit code display. That is why beep codes exist.
3. Then it enumerates buses. It walks every PCIe and USB port, reads the ID of whatever is there, and assigns address windows and interrupts.
4. Then it looks for something to boot.
5. Old BIOS did this by reading the first 512 bytes of each disk in a configured order, checking for the two marker bytes 0x55 0xAA at the end, and jumping into that code.
6. 512 bytes is almost nothing, so that code's only job was to find and load a bigger loader.
7. UEFI does it differently. It reads a real FAT-formatted partition on the disk, called the EFI System Partition, and runs a normal executable file from it.
8. UEFI also keeps a list of boot entries in its own non-volatile memory, each naming a device and a file path, plus an order to try them in.
9. Settings and the clock survive power loss because of the coin cell. Remove it and the settings return to defaults, which is the standard fix for a forgotten firmware password on older boards.

13.11.5 TECHNICAL - the engineer's version

1. The IBM PC of 1981 shipped an 8 KB BIOS ROM, and IBM printed its full assembly listing in the technical reference manual. Compaq in 1982 and Phoenix Technologies in 1984 produced clean-room compatible versions, which is what created the PC clone industry.

2. BIOS ran in 16-bit real mode with a 1 MiB address space and offered services through software interrupts, notably INT 13h for disk and INT 10h for video.
3. The replacement effort began at Intel in 1998 as the Intel Boot Initiative, for Itanium, becoming EFI. Intel stopped at EFI 1.10 in 2005 and gave it to the Unified EFI Forum. UEFI 2.0 was published on 31 January
2006. Version 2.11 was published in December 2024.
4. UEFI runs in 32-bit or 64-bit mode, has a driver model of its own, a shell, and services split into boot services and runtime services. Runtime services survive into the running operating system, which is how a system can set its own next boot entry.
5. The EFI System Partition is FAT12, FAT16 or FAT32, conventionally at least 100 MB and often 512 MB, with GUID type C12A7328-F81F-11D2-BA4B- 00A0C93EC93B. The default fallback path is `\EFI\BOOT\BOOTX64.EFI`.
6. Boot entries are UEFI variables named Boot0000, Boot0001 and so on, with BootOrder listing the sequence and BootCurrent naming the one used.

```
$ efibootmgr -v
BootCurrent: 0002
BootOrder: 0002,0000,0001
Boot0000* Windows Boot Manager  HD(1,GPT,...)/File(\EFI\...)
Boot0002* ubuntu  HD(1,GPT,...)/File(\EFI\ubuntu\shimx64.efi)
```

7. On Windows the equivalent tool is `bcdedit /enum firmware`.
8. GPT (GUID Partition Table) is part of the UEFI specification. MBR with 512 byte sectors cannot address beyond 2 TiB; GPT uses 64-bit LBAs, keeps a backup table at the end of the disk, protects both with CRC32, and supports 128 entries by default. The UEFI specification requires firmware to support GPT; booting from MBR was handled by the Compatibility Support Module, which Intel removed from client platforms around 2020.
9. Secure Boot was added in UEFI 2.3.1, released on 6 April 2011, and enforced on Windows 8 logo machines from 2012. It verifies the signature of each executable the firmware loads against a key hierarchy: PK, the platform key; KEK, key exchange keys; db, allowed signatures; and dbx, the revocation list.
10. What Secure Boot actually verifies: the bootloader, the firmware drivers it loads, and, if the operating system continues the chain, the kernel. What it does not verify: your hardware, your applications, your data, or anything after the kernel takes over. It is a chain of trust for boot, not an antivirus.
11. Linux distributions boot under Secure Boot through `shim`, a small loader signed by Microsoft's third-party UEFI CA, which then verifies the distribution's own key.
12. Current and dated: Microsoft's 2011-generation Secure Boot certificates reach the end of their validity through 2026, and are being replaced by 2023-dated certificates via firmware and operating system updates. If you read this after 2026, check the current state rather than trusting this line.
13. Chapter 18 picks the story up here: from the moment the bootloader is running and begins to load an operating system kernel.

13.11.6 WORDS - remember these

1. POST — the start-up self test — power-on self test of CPU, memory and core chipset before boot.
2. Reset vector — the first instruction address — the fixed address the CPU fetches from after reset.
3. BIOS — the old start-up firmware — 16-bit real-mode firmware with INT-based services and MBR booting.
4. UEFI — the modern replacement — a specification with its own executables, FAT system partition and boot variables.
5. ESP — the boot partition — the EFI System Partition holding UEFI executables.
6. GPT — the modern partition table — GUID Partition Table with 64-bit addressing and CRC-protected redundant headers.

7. Secure Boot — signature checking at boot — verification of loaded images against the PK, KEK, db and dbx key stores.

13.12 Plug and play, resources and conflicts

13.12.1 PLAIN - in simple words

1. Every device needs three things reserved for it: an address range, an interrupt number, and sometimes a data-moving channel.
2. If two devices are given the same one, both misbehave, often in confusing ways.
3. In the 1980s and early 1990s you set these by hand, by moving tiny plastic caps called jumpers on the card itself.
4. You had to know what everything else in the machine already used, and read the manual for each card.
5. Plug and play was the promise that the machine would work this out by itself.
6. It works by having each device describe what it needs, rather than demand a fixed setting.
7. The firmware and the operating system then assign resources so nothing overlaps.
8. Today conflicts are rare, and when they appear they are almost always a driver problem rather than a real hardware clash.

13.12.2 PLAIN - a picture in your head

1. Old cards were like guests who each insisted on a specific seat number printed on their own ticket.
2. If two guests printed the same number, you had to reprint one ticket by hand before the meal could start.
3. Plug and play is guests who say "I need one seat near a window" and let the host do the seating plan.
4. The host has the full list and can always find an arrangement.

Where this comparison breaks: a host can seat anyone anywhere. Some hardware still only works at fixed addresses, so the seating plan has permanent reserved places it must work around.

13.12.3 PLAIN - a worked example

1. A 1993 machine with a sound card and a network card, set by jumpers.

Device	IO port	IRQ	DMA
Sound Blaster	0x220	5	1
Network card	0x300	10	-
LPT1 printer	0x378	7	-
COM1 serial	0x3F8	4	-

2. Set the network card to IRQ 5 by mistake and both cards break: the sound stutters and the network drops, because two devices pull one shared line.
3. The fix was physical. Power off, open the case, move a jumper, close, boot, and test.
4. Today the same information is assigned automatically at every boot, and can change between boots without anything breaking.

13.12.4 PLAIN - what is really happening inside

1. A PCI or PCIe device has a small configuration space that anyone can read before the device is configured.
2. It contains the vendor ID, product ID, class code, and a set of base address registers describing how much address space it wants.
3. The firmware reads all of these, builds a map with no overlaps, and writes the chosen addresses back into the device.

4. Interrupts are assigned the same way, and with message-signalled interrupts there is no physical line to share at all, so classic conflicts simply cannot happen.
5. ACPI adds a description of everything that is not discoverable: which interrupt the embedded controller uses, what the power buttons are, how the fans and sleep states work.
6. So the operating system reads devices from the buses and reads non-discoverable parts from ACPI tables, and combines both into one tree.

13.12.5 TECHNICAL - the engineer's version

1. ISA Plug and Play was specified by Intel and Microsoft in 1993, using an isolation protocol at ports 0x279 and 0xA79 to enumerate cards that could not otherwise be discovered. It was fragile enough to earn a bad reputation in the Windows 95 era.
2. PCI configuration space is 256 bytes per function, extended to 4096 bytes in PCIe. Access is through ports 0xCF8 and 0xCFC in the legacy mechanism, or through the memory-mapped ECAM region described by the ACPI MCFG table.
3. ACPI 1.0 was published in December 1996 by Intel, Microsoft and Toshiba. It is now maintained by the UEFI Forum, which took it over in 2013. Current revisions are 6.x.
4. ACPI tables to know: DSDT and SSDT contain the AML bytecode describing devices, MADT describes interrupt controllers, MCFG describes PCIe configuration space, and FADT holds fixed hardware descriptions.
5. Windows Device Manager error codes you will actually meet: Code 10, the device cannot start, usually a driver or firmware fault; Code 12, not enough free resources, the modern trace of a genuine resource conflict; Code 28, no drivers installed; Code 43, the driver reported a failure.
6. Code 12 today is usually not two devices fighting over one IRQ. It is normally exhausted PCIe lanes or, on older 32-bit systems, exhausted address space below 4 GiB.
7. Inspect resource assignment on Linux with `cat /proc/iomem`, `cat /proc/ioports` and `lspci -vv`. On Windows use `msinfo32` and look under Hardware Resources for Conflicts/Sharing.
8. The honest version: interrupt sharing is still normal and correct on legacy PCI lines. Several devices on one line is not a conflict. A conflict is when the same exclusive resource is claimed twice.

13.12.6 WORDS - remember these

1. Jumper — a manual setting cap — a shorting block across two header pins selecting a hardware option.
2. Resource — something a device must be given — an IO range, memory window, interrupt or DMA channel.
3. Plug and play — automatic setup — enumeration and dynamic resource allocation without user configuration.
4. Configuration space — the device's self-description — the 256-byte or 4096-byte PCI header with IDs and BARs.
5. ACPI — the description of the rest of the board — tables and AML bytecode describing power, and non-discoverable devices.

13.13 System-on-chip: when the whole board fits on one die

13.13.1 PLAIN - in simple words

1. A phone has no slots, no card connectors and no chipset you can point at.
2. Nearly everything discussed in this chapter has been folded onto a single piece of silicon called a system-on-chip, or SoC.
3. The processor, graphics, memory controller, camera processor, radio controller, video decoder and security block all sit on one die.
4. The memory is usually stacked directly on top of the same package.
5. Nothing is removable, so nothing needs a slot, a connector or a plug and play discovery process.

6. The parts still talk to each other over buses, but those buses are inside the chip, made of metal layers, not board traces.
7. Because nothing can be discovered by asking, the software must be told in advance what exists and where. That description is called a device tree.

13.13.2 PLAIN - a picture in your head

1. A desktop is a town with separate buildings and roads between them.
2. A system-on-chip is a single very large building, where every department is a room on the same floor.
3. Walking between rooms is far quicker than driving between buildings, and uses far less energy.
4. But you cannot add a new department. The building is finished.
5. And there is no directory at the entrance, because the plan was fixed when it was built. Everyone must be handed a map instead.

Where this comparison breaks: parts of the chip can still be discovered. A PCIe controller inside an SoC enumerates its own devices exactly as a desktop does. It is the fixed on-chip blocks that need the map.

13.13.3 PLAIN - a worked example

1. A device tree entry describing a serial port on an ARM board looks like this.

```
uart0: serial@fe215040 {
    compatible = "brcm,bcm2835-aux-uart";
    reg = <0xfe215040 0x40>;
    interrupts = <0 93 4>;
    clocks = <&aux_clk>;
    status = "okay";
};
```

2. Read it line by line. `compatible` is the matching key, the equivalent of a vendor and product ID.
3. `reg` says the registers start at address 0xFE215040 and occupy 0x40, that is 64, bytes.
4. `interrupts` says which interrupt line it uses.
5. `clocks` says which clock must be switched on before it works.
6. The kernel reads this, finds the driver whose compatible string matches, and calls its probe function with the address and interrupt filled in.
7. That is exactly the same probe mechanism as PCIe, with the tree taking the place of bus enumeration.

13.13.4 PLAIN - what is really happening inside

1. Inside the chip, blocks connect to a shared on-chip fabric: a set of wide internal buses with an arbiter deciding who gets access.
2. Fast blocks such as the processor and graphics sit on a wide, high-speed part of the fabric. Slow blocks such as a serial port sit on a narrow, cheap part behind a bridge.
3. Wide parallel buses are fine here, because the distance is millimetres and fixed at design time, so skew is controlled by the layout tools.
4. The device tree is compiled into a binary blob and handed to the kernel by the bootloader at start-up.
5. So the boot chain is: chip's internal ROM, then a small first-stage loader, then a full bootloader, then the kernel plus the device tree.
6. On x86 machines the same job is done by ACPI tables from the firmware. Arm servers use ACPI too; Arm phones and embedded boards use device trees.

13.13.5 TECHNICAL - the engineer's version

1. On-chip interconnect standards come mostly from Arm's AMBA family: AXI for high-performance memory traffic, AHB for medium speed, and APB for slow peripheral registers. Larger chips use a network-on-chip with routers rather than a single bus.

2. Device tree came from Open Firmware, standardized as IEEE 1275 in 1994 and used by Sun and Apple PowerPC machines. Linux adopted it for PowerPC and then for Arm, source files ending `.dts`, compiled by `dtc` into `.dtb`.
3. Arm Linux switched to device trees around Linux 3.x, from 2011 onward, replacing thousands of lines of hardcoded board files.
4. Why phone drivers age badly. An SoC vendor ships a board support package: a kernel fork with hundreds of out-of-tree drivers. Because Linux has no stable in-kernel driver ABI, moving to a newer kernel means porting all of them. Vendors support a chip for a few years and then stop.
5. Google's Project Treble, introduced with Android 8.0 in 2017, split the vendor implementation from the Android framework behind a stable interface called the HAL, so the framework could be updated without the vendor's code changing.
6. The Generic Kernel Image work, from Android 11 onward, went further: one common kernel binary with vendor code confined to loadable modules across a defined module interface. Google Pixel and Samsung Galaxy devices now advertise seven years of updates, which was not possible under the old model.
7. Inspect on a running Linux SoC:

```
ls /proc/device-tree/            # the live tree
cat /proc/device-tree/model      # the board name
dtc -I fs -O dts /proc/device-tree | less
```

8. The honest version: "no slots" is not quite true. Many SoCs expose PCIe and USB, and a laptop built on an SoC still has USB-C ports that enumerate normally. What disappeared is the socketed, replaceable, discoverable internal hardware, not discoverable buses in general.

13.13.6 WORDS - remember these

1. SoC — the whole computer on one chip — a die integrating CPU, GPU, memory controller and peripherals.
2. Fabric — the buses inside the chip — an on-chip interconnect such as AMBA AXI or a network-on-chip.
3. Device tree — the map of fixed hardware — a data structure describing non-discoverable devices, compiled to a DTB.
4. Compatible string — the matching key — the device tree property a driver binds to, in place of a vendor and product ID.
5. Board support package — the vendor's kernel fork — a chip-specific kernel plus out-of-tree drivers.

13.98 Common wrong ideas

1. Wrong: a driver is a program you install and run. Right: it is code the operating system loads and calls; you never start it yourself, and it has no window.
2. Wrong: more wires means faster, so parallel beats serial. Right: above a few hundred megahertz, skew and crosstalk make wide buses unusable, which is why every fast modern link is narrow and serial.
3. Wrong: a USB-C port supports fast data, video and 240 W charging. Right: USB-C is only a connector shape. A USB-C port may be limited to USB 2.0 at 480 Mbit/s and 7.5 W. Only the specification sheet tells you.
4. Wrong: a card in an x16 slot always gets sixteen lanes. Right: the connector size and the wired electrical width are separate. A full-length slot may be wired x4, or drop to x8 when another slot is used.
5. Wrong: firmware and drivers are the same thing. Right: firmware runs on the device's own processor and lives in the device's flash. A driver runs on your processor inside your operating system.
6. Wrong: Secure Boot stops malware. Right: it verifies signatures on boot-time executables against firmware key stores. It does nothing about anything that runs after the kernel starts.
7. Wrong: interrupts are always better than polling. Right: at very high event rates, interrupt overhead dominates. Fast network and storage paths deliberately poll, using NAPI, DPDK or SPDK.

8. Wrong: two devices sharing an interrupt is a conflict. Right: shared level-triggered lines are normal and correct. A conflict is two claims on the same exclusive resource, and message-signalled interrupts removed most of that possibility.
9. Wrong: USB 3.2 Gen 2x2 is newer and therefore faster than USB4. Right: Gen 2x2 is 20 Gbit/s. USB4 reaches 40 Gbit/s, and USB4 Version 2.0 reaches 80 Gbit/s. The version numbers do not sort by speed.
10. Wrong: signed drivers are safe drivers. Right: a signature proves origin and integrity, not correctness. Signed drivers with serious bugs and exploitable flaws are shipped regularly.

13.99 Chapter summary in 20 lines

1. A motherboard is a fixed set of sockets, slots, chips and printed wires, each with one job.
2. Fast parts sit close to the processor because high-speed wires must be short and length-matched.
3. A bus carries address, data and control, and a shared bus lets only one talker speak at a time.
4. Parallel buses died above a few hundred megahertz because of skew, crosstalk and electrical loading.
5. Serial point-to-point links won by moving the hard problem into silicon: scrambling, equalization and clock recovery.
6. Bandwidth is transfers per second times width times coding efficiency; PCIe 3.0 x1 is 8 GT/s x 128/130 / 8, about 985 MB/s each way.
7. The bus story runs ISA in 1981, VESA Local Bus in 1992, PCI in 1992, AGP in 1997, then PCI Express from 2003 to now.
8. Connector size and electrical width are separate; bifurcation splits one wide link into several narrower ones.
9. A processor reaches a device through memory-mapped registers or, on x86 only, a separate port space.
10. Polling burns cycles, interrupts free them, and direct memory access removes the processor from bulk transfers entirely.
11. The IOMMU translates and checks every device memory access, which is what makes Thunderbolt and virtual machine passthrough safe.
12. Interrupts run from line or message, through a controller, to a vector, to a short handler, with the slow work deferred to a bottom half.
13. USB has one host, a tiered star of hubs, endpoints, pipes and four transfer types, and enumerates in a fixed nine-step sequence.
14. USB naming has been rewritten twice; as of 2026 the guidance is speed labels: USB 5Gbps through USB 80Gbps.
15. USB-C is a 24-pin connector from 2014, not a protocol and not a speed.
16. Thunderbolt tunnels PCI Express and DisplayPort inside one link, reaching 80 Gbit/s in Thunderbolt 5 from 2023.
17. A driver translates a general operating system request into one specific chip's registers, sitting under a class driver and above the hardware.
18. Windows uses WDM and WDF with KMDF and UMDF, Linux uses kernel modules with sysfs and udev, macOS is replacing kexts with DriverKit.
19. Firmware runs on the device's own processor from its own flash, before and after your operating system, and a failed update can brick the device.
20. Platform firmware, BIOS then UEFI, runs POST, enumerates buses, reads its boot entries and loads a bootloader; Chapter 18 continues from there.

14

CHAPTER

KB

The Long Road — A History of Computing Machines

14.0 What this chapter gives you

1. You will be able to tell the story of computing from tally sticks to the machines of 2026, in order, with correct years and names.
2. You will be able to explain what a punched card did in a loom in 1804, and why that counts as the first machine following stored instructions.
3. You will be able to say what Charles Babbage designed, what Ada Lovelace wrote in 1843, and where the honest scholarly argument about her sits.
4. You will be able to explain a Turing machine, the halting problem and the phrase "Turing complete" to somebody who knows no mathematics.
5. You will be able to name the wartime machines of 1941 to 1945 and say exactly what each one could and could not do.
6. You will be able to explain why the stored-program idea of 1945 changed everything, and who deserves credit for it.
7. You will be able to trace the chain: transistor, integrated circuit, microprocessor, personal computer, graphical screen, network, phone.
8. You will be able to say why the industry stopped raising clock speed around 2005 and started adding cores instead.
9. You will be able to correct the five most common history mistakes people repeat about computers.
10. You will have one master timeline you can read in five minutes.

14.1 Counting before machines

14.1.1 PLAIN - in simple words

1. For most of human history, counting aids were sticks, stones and beads.
2. A **tally stick** is a stick with notches cut into it. One notch, one thing.
3. Some tally sticks were split lengthwise so two people each kept half. The matching notches proved the debt. England used such sticks for tax records for centuries, right up to 1826.
4. An **abacus** is a frame of beads on rods. Each rod is a decimal place.
5. The beads do not calculate. The person calculates. The beads hold the partial answer so the person does not have to remember it.
6. That is the first big idea in this whole chapter: a machine that remembers for you is already useful, even if it does no thinking.
7. In 1901 sponge divers found a corroded lump of bronze in a shipwreck off the Greek island of Antikythera.
8. Inside were at least 30 meshing gears. It is called the **Antikythera mechanism**, and it modelled the sky.

9. Turn a handle and pointers showed the positions of the Sun and Moon, the phase of the Moon, and when eclipses were due.
10. Nothing of that mechanical complexity is known again for over a thousand years. It was a dead end, not a starting point.

14.1.2 PLAIN - a picture in your head

1. Think of a kitchen with no notepad.
2. You are adding a long shopping list in your head. By item nine you have lost item three.
3. Now somebody hands you a row of ten cups and a bag of stones.
4. You drop stones into cups to hold each part of the sum. Your head is free.
5. The cups do not add. You add. The cups only refuse to forget.
6. Napier's bones, the slide rule and the abacus are all cups and stones. They store a partial result, or they turn a hard job into an easy one.
7. The Antikythera mechanism is different. It is a clockwork sky. The gear ratios *are* the astronomy, baked into metal once and for all.

Where this comparison breaks: cups and stones can hold any number you like, but a gear train can only ever do the one calculation its teeth were cut for. To change the question you must cut new gears. That difference, fixed function versus changeable instructions, is the whole story of the next 2,000 years.

14.1.3 PLAIN - a worked example

1. John Napier published **Napier's bones** in 1617, in a book called *Rabdologiae*, printed in Edinburgh. He died that same year.
2. Each bone is a rod carrying one column of the multiplication table.
3. To multiply 4,875 by 7, lay out the bones for 4, 8, 7 and 5, and read row 7.
4. Row 7 gives, in each bone, the digits of 7 times that digit, split by a diagonal line into tens and units.

```
Bones for   4     8     7     5
Row 7:    2/8   5/6   4/9   3/5

Add along the diagonals, right to left:
  units:            5
  next:       3 + 9 = 12  -> write 2, carry 1
  next:   1 + 4 + 6 = 11  -> write 1, carry 1
  next:   1 + 5 + 8 = 14  -> write 4, carry 1
  next:       1 + 2 = 3
Result: 34125
```

5. Check: 4,875 times 7 is 34,125. The bones turned four multiplications into four small additions.
6. That is the trick every pre-electronic aid uses. Convert a hard operation into an easier one you can already do.

14.1.4 PLAIN - what is really happening inside

1. Napier's other invention, logarithms, published in 1614, does the same trick at a deeper level.
2. A logarithm turns multiplication into addition. Multiply two numbers by adding their logarithms and looking up the answer.
3. Edmund Gunter engraved a logarithmic scale on a ruler around 1620. William Oughtred put two such scales side by side around 1622 and slid one along the other. That is the **slide rule**.
4. Sliding one scale by a distance *is* adding the logarithms. The mechanism does the addition by geometry.
5. In 1623 Wilhelm Schickard, a professor at Tübingen, described a "calculating clock" in letters to the astronomer Johannes Kepler.
6. It used Napier's bones mounted on cylinders for multiplication and a geared adder underneath. The machine being built for Kepler burned in a fire before delivery. Schickard died of plague in 1635.

7. In 1642 the nineteen-year-old Blaise Pascal built the **Pascaline** to help his father, a tax official, add long columns of money.
8. It added and subtracted with toothed wheels, and it carried automatically: when a wheel passed nine, a weighted lever tipped and nudged the next wheel.
9. Around fifty were made. About nine survive.
10. Gottfried Wilhelm Leibniz went further. He showed a wooden model of his **stepped reckoner** to the Royal Society in London on 1 February 1673.
11. It was the first design that could do all four operations: add, subtract, multiply and divide.
12. In 1703 Leibniz published a short paper, *Explication de l'Arithmétique Binaire*, setting out arithmetic using only 0 and 1.
13. He did not build a binary machine. But every machine in the rest of this book runs on the notation he wrote down in 1703.

14.1.5 TECHNICAL - the engineer's version

1. The Antikythera mechanism carries at least 30 surviving bronze gears; the largest has 223 teeth, encoding the 223-month Saros eclipse cycle.
2. Dating is contested. Published estimates cluster around 150 to 100 BCE, with arguments for 205 BCE and for about 87 BCE. The shipwreck itself dates to roughly 70 to 60 BCE. Experts disagree; treat "around 100 BCE" as a reasonable midpoint, not a settled fact.
3. Its lunar train uses a pin-and-slot on two gears of 53 teeth to produce a varying angular rate, modelling the Moon's elliptical motion. That is a mechanical implementation of a first-order anomaly correction.
4. The Pascaline's carry mechanism, the *sautoir*, is gravity-driven. Its drawback is that a long carry chain, such as 999,999 plus 1, must lift several weights at once, so torque rises with carry length.
5. Leibniz's key part is the **stepped drum** (Staffelwalze): a cylinder with nine teeth of increasing length. Shifting a gear along the drum selects how many teeth engage per turn, which multiplies by a chosen digit.
6. The stepped drum stayed in production hardware until the twentieth century, in the Thomas Arithmometer from 1820 and later in the Curta of 1948.

Device	Year	Operations it did
Napier's bones	1617	Multiply, divide aid
Slide rule	c. 1622	Multiply, divide, roots
Schickard clock	1623	Add, subtract, aid mult
Pascaline	1642	Add, subtract
Stepped reckoner	1673	All four operations

7. Precision limits, not ideas, blocked all of these. Leibniz's machine had a carry fault that was only fully understood when it was studied in the twentieth century.
8. Standard versus convention: none of this is standardized. These are one-off artefacts. The first true standard in computing history is the punched card format, and that comes in section 14.4.

14.1.6 WORDS - remember these

1. Tally stick — a notched stick counting things — a unary record medium.
2. Abacus — beads on rods — a positional manual accumulator.
3. Antikythera mechanism — an ancient geared sky model — a bronze analogue astronomical calculator, roughly 100 BCE.
4. Napier's bones — rods holding times tables — lattice multiplication aid,

1617.

5. Logarithm — a number that turns times into plus — the exponent to which a fixed base must be raised.
6. Slide rule — two sliding log scales — an analogue multiplier, about 1622.

7. Pascaline — Pascal's adding box of 1642 — geared decimal adder with gravity carry.
8. Stepped reckoner — Leibniz's four-function machine, 1673 — stepped-drum mechanical calculator.
9. Binary — counting with only 0 and 1 — base-2 positional notation, set out by Leibniz in 1703.

14.2 Machines that follow instructions

14.2.1 PLAIN - in simple words

1. A calculator does one job. To change the job you change the machine.
2. The next idea is bigger: leave the machine alone, and change a stack of cards instead.
3. In 1804 Joseph Marie Jacquard, in Lyon in France, patented an attachment that sat on top of a weaving loom.
4. Weaving a picture into cloth means lifting a different set of threads for every single row of the picture.
5. Before Jacquard, a boy sat inside the loom pulling cords by hand, one row at a time, all day, for weeks.
6. Jacquard replaced the boy with a chain of stiff cards, each punched with holes.
7. Rods press against a card. Where there is a hole, a rod goes through and its thread lifts. Where there is card, the rod is blocked and the thread stays.
8. One card equals one row of the picture. The chain of cards equals the whole picture.
9. To weave a different picture you do not rebuild the loom. You hang a different chain of cards.
10. That separation, machine on one side and instructions on the other, is the single most important idea in this book.

14.2.2 PLAIN - a picture in your head

1. Think of a music box with a spiked cylinder.
2. The spikes pluck the metal teeth. The cylinder is the tune.
3. A cheap music box has the cylinder fixed. It plays one tune for ever.
4. A better music box lets you pull the cylinder out and push a new one in. Same box, new tune.
5. Jacquard's loom is that better music box, except the tune is a picture and the cylinder is a loop of punched cards.
6. A player piano works exactly this way too, with a punched paper roll.

Where this comparison breaks: a music box or piano roll only ever plays forwards. It cannot skip back, repeat a bar four times, or choose a different bar depending on what happened. The Jacquard chain has the same limit. Real programs need to jump, loop and decide. That missing power is exactly what Babbage tried to add in 1837, and what nobody achieved in working hardware until the 1940s.

14.2.3 PLAIN - a worked example

1. Take a tiny loom with 8 threads, weaving a picture 5 rows tall.
2. Punch 1 means "lift this thread". No punch, shown as 0, means "leave it".

```
Card 1:  0 0 1 1 1 1 0 0
Card 2:  0 1 1 0 0 1 1 0
Card 3:  1 1 0 0 0 0 1 1
Card 4:  0 1 1 0 0 1 1 0
Card 5:  0 0 1 1 1 1 0 0
```

3. Read the 1s down the page and a diamond shape appears in the cloth.
4. Five cards, eight holes each: 40 bits of information decide a picture.
5. Real Jacquard cards of the period commonly carried several hundred hole positions per card, and a single elaborate portrait could need tens of thousands of cards.
6. The famous woven portrait of Jacquard himself, made in the 1830s, used around 24,000 cards for one image.

7. Charles Babbage owned a copy of that portrait and showed it to visitors. That is a direct, documented link from weaving to computing.

14.2.4 PLAIN - what is really happening inside

1. Jacquard did not invent this from nothing. He combined three earlier French ideas.
2. Basile Bouchon used a punched paper tape for loom control in 1725.
3. Jean-Baptiste Falcon replaced the fragile tape with a chain of stiff cards in 1728.
4. Jacques de Vaucanson built an automatic loom around 1740 using a perforated cylinder.
5. Jacquard's 1804 machine put the strong parts together and made it reliable and cheap enough for real workshops.
6. Mechanically: a rectangular card presses against a bank of spring-loaded needles. Each needle controls a hook. Each hook lifts one warp thread.
7. Hole present: needle passes through, hook stays engaged with the lifting knife, thread rises.
8. No hole: needle is pushed back, hook is displaced sideways, the knife misses it, thread stays down.
9. The card is therefore a row of yes/no decisions, read in parallel, once per row of cloth. It is a memory device made of cardboard.
10. Weavers understood the threat immediately. There were riots in Lyon, and looms were destroyed. This is where the word Luddite belongs, though the English Luddites of 1811 to 1816 were mostly attacking other machines.

14.2.5 TECHNICAL - the engineer's version

1. The Jacquard head is a parallel-read, sequential-advance control store. Card width sets the number of parallel control lines.
2. Typical nineteenth-century heads came in sizes such as 400, 600 and 900 hooks. A 900-hook head is a 900-bit-wide control word.
3. Advance is mechanical and unconditional: a four-sided prism rotates one quarter turn per pick, presenting the next card. There is no branch, no conditional and no addressing.
4. In formal terms the Jacquard head is a finite-state sequencer with an external, replaceable program tape, and no conditional transfer. It is not Turing complete and was never meant to be.
5. The honest version: calling a Jacquard card chain "the first program" is a useful teaching device, not a precise claim. It is the first widely used **stored, replaceable, machine-readable instruction sequence**. It is not a program in the modern sense because it cannot branch or loop.
6. The lineage is nonetheless direct and documented. Babbage planned card input for the Analytical Engine from 1837, Herman Hollerith used cards for the 1890 census, and IBM's punched card business ran on that idea into the 1970s.

Machine	Year	Program medium
Bouchon loom	1725	Punched paper tape
Falcon loom	1728	Chain of cards
Vaucanson loom	c. 1740	Perforated cylinder
Jacquard head	1804	Chain of stiff cards
Analytical Engine	1837	Punched cards, planned

14.2.6 WORDS - remember these

1. Punched card — stiff card with holes meaning yes or no — a parallel-read binary storage medium.
2. Warp — the threads held tight lengthwise on a loom — the addressable lines the control head selects.
3. Jacquard head — the card-reading box on top of a loom, 1804 — a wide parallel control-store sequencer.
4. Program — a list of instructions a machine follows — an encoded sequence of operations held separately from the mechanism.
5. Branch — choosing what to do next — a conditional transfer of control, which the Jacquard head lacks.

14.3 Charles Babbage and Ada Lovelace

14.3.1 PLAIN - in simple words

1. In the early 1800s, sailors, bankers and astronomers all used printed books of numbers called tables.
2. The tables were worked out by hand by people whose job title was **computer**. That word meant a person until the 1940s.
3. People make mistakes. Printers make more. A wrong number in a navigation table could put a ship on rocks.
4. Charles Babbage, an English mathematician, wanted to remove the humans from both steps: calculate by machine, and print by machine, so no typesetter could introduce an error.
5. On 14 June 1822 he described his **Difference Engine** to the Royal Astronomical Society.
6. It could not multiply. It only added. That was enough, because of a mathematical trick explained in the worked example below.
7. The British government funded it from 1823. By 1842 it had spent over 17,000 pounds, a huge sum, and the machine was not finished.
8. Meanwhile Babbage had a much bigger idea. From 1837 he designed the **Analytical Engine**, a general-purpose machine.
9. It had a **mill** to do the arithmetic, a **store** to hold numbers, a card **reader** for instructions, and a **printer** for output.
10. Those four parts map exactly onto a modern computer: processor, memory, input, output.
11. Neither machine was completed in his lifetime. Babbage died in 1871.

14.3.2 PLAIN - a picture in your head

1. Think of a very large mechanical cash register, the size of a small room, made of brass and iron, turned by a handle or a steam engine.
2. Inside are columns of numbered wheels. Each column holds one number.
3. Turning the handle makes the columns add themselves together in a fixed pattern, and a new answer appears at the front.
4. Keep turning and a whole table of numbers pours out, and a printing plate is stamped at the same time.
5. That is the Difference Engine: one fixed pattern, endlessly repeated.
6. Now imagine a second machine where a stack of cards tells the columns what to do this time, and where the machine can look at a result and choose a different card next. That is the Analytical Engine.

Where this comparison breaks: a cash register has one accumulator. Babbage's store was designed for 1,000 numbers of 40 decimal digits each, which is about 16.6 kilobytes. And no cash register can change its own next action based on a result. That ability to decide is what makes the Analytical Engine a computer and the Difference Engine a calculator.

14.3.3 PLAIN - a worked example

1. Here is the trick that lets a machine that can only add produce a table of squares.
2. Write the squares: 1, 4, 9, 16, 25, 36.
3. Subtract each from the next. These are the first differences: 3, 5, 7, 9, 11.
4. Subtract those from each other. The second differences: 2, 2, 2, 2. Constant.
5. So you can run the process backwards using only addition.

Step	2nd diff	1st diff	Value
start	2	3	1
1	2	5	4
2	2	7	9

Step	2nd diff	1st diff	Value
3	2	9	16
4	2	11	25

6. Each row: add the second difference to the first difference, then add the first difference to the value. Two additions per table entry.
7. Any polynomial of degree n has constant nth differences, so this works for a whole family of useful functions.
8. Logarithms and sines are not polynomials, but they can be approximated by polynomials over short ranges, which is how real tables were produced.
9. Difference Engine No. 1 was designed for 20-digit numbers and sixth-order differences. Difference Engine No. 2, designed between 1847 and 1849, handled 31-digit numbers and seventh-order differences.

14.3.4 PLAIN - what is really happening inside

1. Now Ada Lovelace. She was born in 1815, the daughter of the poet Lord Byron, and was taught mathematics from childhood.
2. She met Babbage in 1833 and followed the Analytical Engine work closely.
3. In 1840 Babbage lectured in Turin. An Italian engineer, Luigi Federico Menabrea, wrote the lecture up in French and published it in 1842.
4. Lovelace translated Menabrea's article into English over about nine months in 1842 and 1843, and added seven notes of her own, labelled A to G.
5. Her notes are roughly three times longer than the article she translated.
6. **Note G** contains a step-by-step table showing how the Analytical Engine would compute the Bernoulli numbers, a sequence used in advanced arithmetic.
7. That table has variables, operations, and a loop. It is widely called the first published computer program.
8. Note A contains something arguably more important. Lovelace saw that the numbers in the machine need not stand for quantities at all.
9. She wrote that if the relations of pitched sounds could be expressed as numbers, the engine might compose music. That is the idea of general computing, written down in 1843.
10. She also wrote that the machine has no pretensions to originate anything, and can only do what we know how to order it to perform. That line is still quoted in arguments about artificial intelligence today.

14.3.5 TECHNICAL - the engineer's version

1. Analytical Engine specification, as designed from 1837: store of 1,000 numbers of 40 decimal digits; decimal, not binary; mill performing the four operations; separate operation cards and variable cards; conditional branching via a mechanism Babbage called the "combinatorial" or barrel control; output to printer, curve plotter and bell.
2. Estimated speed, from Menabrea's account of Babbage's claims: an addition in about one second, and the product of two 20-digit numbers in about three minutes.
3. Babbage anticipated microprogramming: the barrel held sequences of micro-operations that a single instruction card triggered. Maurice Wilkes, who invented microprogramming in 1951, noted the resemblance.
4. The scholarly argument about Lovelace, stated fairly, both sides:
    1. Against: Allan Bromley wrote in 1990 that all but one of the programs in her notes had been prepared by Babbage three to seven years earlier. Bruce Collier, also in 1990, judged that she publicized rather than advanced the design. Eugene Eric Kim and Betty Alexandra Toole argued in 1999 that calling her the first programmer is incorrect.

2. For: Doron Swade, who led the Science Museum's Babbage work, holds that she alone saw that numbers could represent things other than quantity, which is the conceptual leap to general-purpose computing. Stephen Wolfram, writing in 2016, judged her Bernoulli computation as sophisticated as anything Babbage produced.

5. The defensible statement: Babbage wrote earlier program-like traces that were not published; Lovelace produced the first published, fully worked algorithm intended for a machine, and the clearest statement of what a general-purpose computer would mean. Both facts are true at once.
6. Neither engine was finished, for three reasons: cost, Babbage's 1833 quarrel with his engineer Joseph Clement over tool ownership, and Babbage's habit of redesigning before finishing.
7. The vindication came late. The Science Museum in London built Difference Engine No. 2 to the original 1847 to 1849 drawings, completing the calculating section in 1991 for Babbage's 200th birthday, and the printer in

2002. It has about 8,000 parts and weighs around 5 tonnes. It works.

Item	Babbage's design	Modern equivalent
Mill	Arithmetic unit	ALU
Store	1,000 x 40 digits	RAM, about 16.6 kB
Operation cards	What to do	Instruction stream
Variable cards	Which store cell	Operand addresses

14.3.6 WORDS - remember these

1. Difference Engine — Babbage's table-printing adder, 1822 — a fixed-function polynomial evaluator using finite differences.
2. Analytical Engine — Babbage's general machine, 1837 — a programmable decimal computer with separate mill, store and card input.
3. Mill — the part that does the sums — the arithmetic and logic unit.
4. Store — the part that remembers numbers — main memory.
5. Method of differences — building a table by adding only — evaluating a degree-n polynomial from constant nth differences.
6. Note G — Lovelace's Bernoulli number table, 1843 — the first published algorithm written for a machine.

14.4 Hollerith, the 1890 census, and the birth of IBM

14.4.1 PLAIN - in simple words

1. The United States counts its whole population every ten years. The constitution requires it.
2. The 1880 count took about eight years to process by hand.
3. The country was growing so fast that the 1890 count risked not being finished before the 1900 count began. That is a real crisis, not a joke.
4. Herman Hollerith, a young engineer who had worked on the 1880 census, had an idea borrowed from railway tickets and from Jacquard's loom.
5. Punch each person's answers as holes in a card. One card, one person.
6. Then build a machine that reads the holes electrically and counts them.
7. He was granted his key United States patent, number 395,782, on 8 January

1889.

8. His machines were used for the 1890 census. The main population count was announced within about six weeks, and the full processing took roughly six years instead of eight, on a much larger and more detailed job.

9. Hollerith founded the Tabulating Machine Company in 1896.
10. In 1911 it merged with three other firms to form the Computing-Tabulating-Recording Company, or CTR.
11. In 1924, under Thomas J. Watson, CTR was renamed International Business Machines. That is IBM.

14.4.2 PLAIN - a picture in your head

1. Imagine a hotel with 200 rooms and a wall of pigeonholes behind reception.
2. Every guest hands in a card with holes punched in it: nationality, age group, room type.
3. The receptionist drops each card into a slot. A lid comes down. Little metal pins press against the card.
4. Where a hole is, a pin dips through and touches mercury in a cup underneath. That completes an electric circuit and a dial clicks up by one.
5. At the same moment, a lid pops open on one pigeonhole, telling the clerk where to file that card.
6. A hundred dials, a hundred questions, one card at a time, all day.

Where this comparison breaks: the real machine was not automatic in the way we would expect. An operator fed every card by hand. The sorting box only opened the correct lid; the human still moved the card. Fully automatic card feeding came later, with Hollerith's own 1900 machines and their successors.

14.4.3 PLAIN - a worked example

1. Say we want the count of people in a city who are both over 60 and born abroad.
2. Each card has a position for "age band 60 plus" and a position for "born abroad".
3. The tabulator is wired so a counter advances only when both pins make contact at once.
4. That wiring is a physical AND gate, built from relays, decades before anyone used the word.

```
Card position 34 punched  -> pin 34 dips -> circuit A closed
Card position 61 punched  -> pin 61 dips -> circuit B closed

Counter 7 advances only if A AND B are closed.
```

5. Change the question by rewiring the plugboard, not by rebuilding the machine.
6. Speed in practice: a trained operator could process roughly 80 cards a minute on the 1890 equipment, so about 1,000 cards an hour with breaks.
7. Roughly 62.9 million people were counted in the 1890 census. The scale is what made the machinery pay for itself.

14.4.4 PLAIN - what is really happening inside

1. The card is the data. The machine is a fixed reader. The plugboard is the program.
2. This is the same split as the Jacquard loom, but now the cards hold facts rather than instructions, and a separate wiring panel holds the instructions.
3. Hollerith's business insight mattered as much as his engineering. He rented the machines rather than selling them, and he sold the blank cards.
4. That model, cheap machine, endless consumable, is the razor-and-blades business. It funded IBM for sixty years.
5. IBM standardized an 80-column rectangular-hole card in 1928. The 80-column card is a genuine industry **standard**, and it is why terminal screens were 80 characters wide for decades, and why many text tools still default to 80.
6. There is a dark chapter that honesty requires naming. IBM's German subsidiary Dehomag supplied punched card equipment used by the Nazi state in the 1930s and 1940s, including for censuses and record-keeping. Historian Edwin Black documented this in his 2001 book *IBM and the Holocaust*. The degree of knowledge and control at IBM headquarters is disputed by historians; the supply of equipment is not.

14.4.5 TECHNICAL - the engineer's version

1. The 1890 card was about 3.25 by 7.375 inches, chosen to match the United States dollar bill of the day so existing cash drawers could store it.
2. The 1890 census card carried up to 288 punch positions in a 12 by 24 grid.
3. Reading was electromechanical: a spring-loaded pin array descends; a pin passing through a hole contacts a mercury cup, closing a circuit that steps a clockwork-style counter dial reading 0 to 9,999.
4. The sorter box had 26 compartments with electrically released lids.
5. The IBM card of 1928 has 80 columns by 12 rows, rectangular holes, and encodes one character per column using Hollerith code: a zone punch in rows 12, 11 or 0 combined with a digit punch in rows 1 to 9.
6. The card measures 7.375 by 3.25 inches by 0.007 inches thick. That thickness is why a box of 2,000 cards is about 14 inches tall.

Year	Machine or event	Significance
1889	US patent 395,782	Electric card tabulating
1890	US census run	First large data processing
1896	Tabulating Machine Co	Hollerith's firm
1911	CTR formed	Four-company merger
1924	Renamed IBM	Name that dominated 60 yrs
1928	80-column card	Format standard to 1970s

7. Card equipment was not a dead end. Unit record machines, sorters, collators, reproducers and accounting machines, ran commercial data processing until magnetic tape displaced them through the 1960s.
8. The last widespread public use of punched cards was in voting machines. The 2000 United States presidential election recount in Florida turned on partly detached card chads, which is why the word "chad" briefly became famous.

14.4.6 WORDS - remember these

1. Tabulator — a machine that counts punched cards — an electromechanical counting unit driven by card-hole contacts.
2. Plugboard — a wiring panel that sets the question — a removable control panel defining data paths, the program of a unit record machine.
3. Unit record — one card, one thing — the data model where each card is a complete record.
4. Hollerith code — the hole pattern for a character — zone plus digit punch encoding on an 80-column card.
5. CTR — the 1911 merger — Computing-Tabulating-Recording Company, renamed IBM in 1924.
6. Chad — the bit punched out of a card — the removed rectangle of card stock, famous from the 2000 Florida recount.

14.5 The theory arrives: Hilbert, Gödel, Church and Turing

14.5.1 PLAIN - in simple words

1. Before anyone built a working computer, mathematicians worked out what a computer could never do. That is a strange and wonderful order of events.
2. Around 1920 the German mathematician David Hilbert proposed a plan for all of mathematics.
3. He wanted a fixed set of starting rules from which every true statement could be proved, with no contradictions, and with a mechanical method for deciding whether any given statement was provable.
4. That last part, the mechanical decision method, had a German name: the **Entscheidungsproblem**, the decision problem. Hilbert and Wilhelm Ackermann stated it clearly in a 1928 textbook.

5. In 1931 Kurt Gödel, a 25-year-old Austrian, destroyed the first two parts.
6. His **incompleteness theorems** showed that any consistent set of rules strong enough to describe ordinary arithmetic must contain true statements it cannot prove, and cannot prove its own consistency.
7. The third part, the decision problem, was still open. Two people answered it independently in 1936.
8. Alonzo Church at Princeton invented a symbol-rewriting system called the **lambda calculus** and used it to prove no such decision method exists.
9. Alan Turing in Cambridge invented an imaginary machine and proved the same thing, in a way that anybody can picture.
10. Turing's imaginary machine turned out to describe every computer that has ever been built.

14.5.2 PLAIN - a picture in your head

1. Picture an endless paper tape, ruled into squares.
2. Each square is blank or holds one symbol from a small fixed alphabet.
3. A small box sits over one square. The box can read that square, rub it out and write a new symbol, and move one square left or right.
4. The box has a **state**, which you can think of as a mood. It has a small fixed number of moods, written on a card inside it.
5. The card is a table of rules of the form: if my mood is 3 and I see a 1, then write 0, move left, and change to mood 7.
6. That is the whole machine. Tape, head, moods, rule table.
7. There is nothing else. No screen, no memory chip, no operating system.
8. Turing's shocking claim is that anything a human being can compute by following a written procedure, this box can compute too.

Where this comparison breaks: the tape is infinite, and no real machine has infinite memory. Real computers are finite-state machines. In practice the tape just needs to be long enough for the problem, so the model is useful anyway. The second break: the model says nothing about speed. A Turing machine takes absurdly many steps for tasks a real processor does in one instruction. It answers "possible or impossible", never "fast or slow".

14.5.3 PLAIN - a worked example

1. Here is a complete Turing machine that adds 1 to a binary number.
2. The number is written on the tape, most significant bit on the left. The head starts on the leftmost digit. Blank squares are shown as _.

```
State  Read  ->  Write  Move  New state
right   0         0      R     right
right   1         1      R     right
right   _         _      L     carry
carry   0         1      L     done
carry   1         0      L     carry
carry   _         1      L     done
done    any      same    -     halt
```

3. Run it on the tape `_1011_`, which is 11 in decimal.
4. In state `right` the head walks over 1, 0, 1, 1 and hits the blank.
5. It steps left into state `carry` on the last 1. Rule: write 0, move left, still carrying.
6. Next square is 1. Write 0, move left, still carrying.
7. Next square is 0. Write 1, move left, done.
8. The tape now reads `_1100_`, which is 12. Correct.
9. Six rules and two moods add 1 to any binary number of any length. That is the power of the model in one small picture.

14.5.4 PLAIN - what is really happening inside

1. Turing's paper is called *On Computable Numbers, with an Application to the Entscheidungsproblem*. He submitted it on 28 May 1936. It was read to the London Mathematical Society on 12 November 1936 and printed in their Proceedings across 1936 and 1937.
2. Church's paper *An Unsolvable Problem of Elementary Number Theory* appeared in the American Journal of Mathematics in April 1936, a few months earlier.
3. Turing added something Church did not have: the **universal machine**.
4. A universal Turing machine is a Turing machine whose rule table is written to read *another machine's rule table* off the tape and imitate it.
5. That is the stored-program computer, described in 1936 as a piece of mathematics, nine years before anybody built one.
6. Now the famous impossibility. Ask: can we write a program that reads any program plus its input, and always answers correctly whether that program will eventually stop or run for ever?
7. Suppose such a checker exists. Build a troublemaker program that runs the checker on itself, and then does the opposite: if the checker says "stops", the troublemaker loops for ever; if the checker says "loops", it stops.
8. Now ask the checker about the troublemaker. Either answer makes the checker wrong. So no such checker can exist.
9. That is the **halting problem**, and it is not a limit of today's computers. It is a limit of computing itself, for ever.
10. The **Church-Turing thesis** says that lambda calculus, Turing machines, and every other reasonable definition of "effectively computable" describe exactly the same set of problems.
11. It is a thesis, not a theorem. It cannot be proved, because "what a human could compute with paper and pencil" is not a formal object. Ninety years of failed attempts to beat it are the evidence.

14.5.5 TECHNICAL - the engineer's version

1. Formally a deterministic single-tape Turing machine is a 7-tuple: states Q, input alphabet, tape alphabet, transition function, start state, accept state, reject state.
2. Turing's own 1936 proof was about "circle-free" machines and the printing problem, not literally about halting. The name **halting problem** and the modern formulation are due to Martin Davis, in his 1958 book *Computability and Unsolvability*.
3. The name **Church-Turing thesis** was coined by Stephen Kleene in his 1952 book *Introduction to Meta-mathematics*.
4. **Turing complete** means a system can simulate a universal Turing machine, given unbounded storage. **Turing equivalent** means two systems simulate each other.
5. Turing completeness needs surprisingly little: sequencing, a way to store and read unbounded data, and a conditional branch. Anything with those three is almost always complete.
6. Because the bar is so low, systems become Turing complete by accident. Real documented examples, with attributions:

System	Shown by	Year
Conway's Game of Life	Conway, then others	1970 on
C++ templates	Erwin Unruh, others	1994 on
Rule 110 automaton	Matthew Cook	2004
x86 MOV instruction	Stephen Dolan	2013
PowerPoint animations	Tom Wildenhain	2017
Magic: The Gathering	Churchill, Biderman, Herrick	2019

7. Read that table carefully. The PowerPoint result was presented at SIGBOVIK, a deliberately humorous conference, but the construction is real. The Magic result is a genuine peer-reviewed style construction

using legal cards.

8. Practical consequences of undecidability, which working engineers meet:
    1. No compiler can decide in general whether your loop terminates, so termination checkers are conservative and refuse some correct programs.
    2. No antivirus can decide in general whether a program is malicious, which is why detection is heuristic and signature-based.
    3. Rice's theorem, proved by Henry Gordon Rice in 1951, generalizes this: every non-trivial property of a program's behaviour is undecidable.
9. The honest version: "undecidable" does not mean "hopeless in practice". It means no method works for every input. Tools such as static analysers, model checkers and the Coq and Lean proof assistants succeed on huge classes of real programs by giving up on the rest.

14.5.6 WORDS - remember these

1. Entscheidungsproblem — Hilbert's question of 1928, is there a mechanical test for provability — the decision problem for first-order logic, answered no in 1936.
2. Incompleteness — some true things cannot be proved — Gödel's 1931 theorems about consistent recursively axiomatized systems containing arithmetic.
3. Turing machine — tape, head, moods, rules — an abstract automaton with unbounded tape and a finite transition function.
4. Universal machine — a machine that imitates any other from its description — the theoretical stored-program computer, Turing 1936.
5. Halting problem — you cannot always tell if a program will stop — the undecidability of the halting set, named by Davis in 1958.
6. Church-Turing thesis — all reasonable definitions of computing agree — an unprovable but unbeaten identification of effective calculability with Turing computability.
7. Turing complete — powerful enough to compute anything computable — able to simulate a universal Turing machine given unbounded memory.

14.6 War machines, 1939 to 1945

14.6.1 PLAIN - in simple words

1. The Second World War pushed several countries to build calculating machines at a scale nobody would have funded in peacetime.
2. Germany encrypted its radio messages with the **Enigma**, a typewriter-sized machine with rotating wheels.
3. Press a key and an electrical path runs through the wheels and lights a different letter. The wheels then step, so the next letter is enciphered differently.
4. Polish mathematicians broke into Enigma first. Marian Rejewski worked out the internal wiring mathematically in December 1932.
5. In July 1939, weeks before the invasion, Poland handed everything it knew to British and French intelligence. That gift saved years.
6. Britain's codebreaking moved to **Bletchley Park**, a country house north of London, in August 1939.
7. Alan Turing and Gordon Welchman designed the **Bombe**, a machine that hunted for the day's Enigma settings.
8. A second, much harder German cipher, called Lorenz by the Germans and Tunny by the British, protected messages between high command and field marshals.
9. To attack Tunny, a Post Office engineer named Tommy Flowers built **Colossus**, the first large-scale electronic digital machine.
10. At the same time Konrad Zuse in Berlin, John Atanasoff in Iowa and Howard Aiken at Harvard were each building very different machines.

14.6.2 PLAIN - a picture in your head

1. Think of a hotel with a combination lock on the front door, and the combination is changed every midnight.
2. You cannot try all combinations, because there are about 159 million million million of them.
3. But you know one thing: the porter always writes the current date on the pad by the door in the same format.
4. So instead of testing combinations against the lock, you test them against that known scrap of text. Any setting that could not have produced it is thrown away instantly.
5. The Bombe is a machine that throws away wrong settings at high speed. It does not decrypt anything. It hands humans a short list to try by hand.
6. The known scrap of text has a name in this trade: a **crib**.

Where this comparison breaks: the Bombe did not simply test and compare. It exploited a specific weakness, that Enigma never enciphered a letter as itself, and it used Welchman's diagonal board to propagate a single contradiction through the whole wiring at once, killing thousands of settings per test rather than one. It was a logic-propagation engine, not a brute-force counter.

14.6.3 PLAIN - a worked example

1. Enigma's setting count, for the standard army machine of the war years:

Choice	Count	Notes
Rotor order	60	3 chosen from 5, in order
Rotor start positions	17,576	26 x 26 x 26
Ring settings	676	Two that matter
Plugboard, 10 pairs	150.7 billion	The dominant term

2. Multiply those and you get about 1.59 times 10 to the power 20 settings.
3. Written out: roughly 159,000,000,000,000,000,000.
4. Testing one per second would take longer than the age of the universe.
5. Now use the crib. Suppose the intercepted text contains `WETTERBERICHT`, German for weather report, and you can guess where.
6. Enigma never maps a letter to itself. Slide the crib along the ciphertext. Any position where a letter lines up with itself is impossible, and is dropped without any machine at all.
7. The Bombe then took the surviving positions, wired up the logical chain of letter-to-letter implications, and spun until the electrical current found no contradiction. It stopped, and a human checked the stop.
8. A single Bombe run took around 20 minutes. By May 1945 Britain had about 155 three-rotor Bombes plus later four-rotor machines, running day and night.

14.6.4 PLAIN - what is really happening inside

1. The first British Bombe, code-named Victory, was installed at Bletchley Park on 18 March 1940.
2. The improved version with Gordon Welchman's diagonal board, called Agnus and then Agnes, ran from 8 August 1940. That addition made the whole approach practical.
3. Bombes were built by the British Tabulating Machine Company at Letchworth, with Harold Keen as chief engineer. About 200 were built in Britain.
4. Be clear on this: the Bombe was electromechanical, not electronic, and it was not a computer. It ran one fixed search. It could not be programmed.
5. Colossus is a different animal. The Lorenz cipher used twelve wheels and was attacked statistically, not by simulation.
6. Bill Tutte worked out the complete internal structure of the Lorenz machine in 1941 and 1942 without ever seeing one, from intercepted traffic alone. It is one of the great feats of the war.

7. Max Newman designed a statistical attack. The first machine to run it, in 1943, was electromechanical and nicknamed Heath Robinson. It was too slow and its paper tapes tore.
8. Tommy Flowers, of the Post Office Research Station at Dollis Hill, argued that valves, what Americans call vacuum tubes, could do the job if you never switched them off. His superiors doubted it. He built it anyway, partly with his own money.
9. Colossus Mark 1, with 1,600 valves, was working at Bletchley Park in January 1944 and attacked its first real message on 5 February 1944.
10. Colossus Mark 2, with 2,400 valves, started work at 8 in the morning on 1 June 1944, five days before the Normandy landings. Ten Colossi were running by the end of the war.
11. Colossus was programmable only by switches and plugs, and it had no stored program and no general arithmetic. It was a special-purpose electronic machine. That is still a first.
12. Most Colossi were broken up on Churchill's orders. The work stayed secret until the mid-1970s, which is why the machine is missing from older history books. A working rebuild led by Tony Sale was completed in 2007.

14.6.5 TECHNICAL - the engineer's version

1. Konrad Zuse completed the **Z3** in Berlin and presented it on 12 May 1941.

Z3 property	Value
Technology	2,600 relays
Word length	22 bits, floating point
Clock	5 to 10 Hz
Memory	64 words
Add / multiply time	0.8 s / 3 s

2. The Z3 used binary floating point with a 14-bit mantissa, 7-bit exponent and sign bit, decades ahead of American practice. Program came from punched 35 mm film. It had no conditional branch.
3. Raúl Rojas showed in 1998 that the Z3 is Turing complete in principle, but only by an artificial construction that computes all branches. That is a theoretical result, not a description of how it was used.
4. The original Z3 was destroyed on 21 December 1943 in an Allied air raid on Berlin. A working replica was built in 1961.
5. The **Atanasoff-Berry Computer** was built at Iowa State College by John Vincent Atanasoff and Clifford Berry between 1939 and 1942.
6. It used about 280 vacuum tubes for arithmetic, was binary, held 60 numbers of 50 bits in rotating capacitor drums, and did about 30 additions per second.
7. Its regenerative capacitor memory, refreshed each rotation, is the direct ancestor of the refresh cycle in modern DRAM.
8. It solved systems of up to 29 simultaneous linear equations. It was not programmable and it never worked fully automatically; the card output unit was unreliable.
9. On 19 October 1973, in *Honeywell v. Sperry Rand*, United States District Judge Earl R. Larson invalidated the ENIAC patent, ruling that Eckert and Mauchly "derived that subject matter from one Dr. John Vincent Atanasoff".
10. The honest version: that ruling is a legal finding about patent validity, not a historical consensus about who invented the computer. Historians still argue about how much Mauchly took from his 1941 visit to Atanasoff.
11. The **Harvard Mark I**, built by IBM as the Automatic Sequence Controlled Calculator, was presented to Harvard on 7 August 1944.
12. It was 51 feet long, 8 feet high, weighed about 4.7 short tons, and did 3 additions per second, a multiplication in 6 seconds and a division in 15.3 seconds. It was decimal, driven by a 50-foot shaft, and read its program from punched paper tape.

13. Howard Aiken led it. Grace Hopper, Richard Bloch and Robert Campbell were among its first programmers. John von Neumann ran implosion calculations on it for the Manhattan Project from 29 March 1944.

14.6.6 WORDS - remember these

1. Enigma — the German rotor cipher machine — a stepping-rotor polyalphabetic substitution device with a plugboard.
2. Crib — a guessed piece of the plain message — known plaintext used to constrain a key search.
3. Bombe — the Bletchley setting-hunting machine, 1940 — an electromechanical parallel logic engine testing rotor hypotheses for contradiction.
4. Lorenz or Tunny — the high command teleprinter cipher — a 12-wheel additive stream cipher on 5-bit Baudot code.
5. Colossus — the 1943 to 1944 valve machine — the first large-scale electronic digital special-purpose computer, 1,600 then 2,400 valves.
6. Z3 — Zuse's 1941 relay machine — the first working programmable automatic binary floating-point computer.
7. ABC — the Atanasoff-Berry Computer of 1942 — the first electronic binary digital calculating device, non-programmable, with regenerative memory.

14.7 ENIAC and EDVAC

14.7.1 PLAIN - in simple words

1. Artillery needs firing tables: for this gun, this shell, this angle, this wind, where does it land.
2. In 1943 the United States Army was drowning in the arithmetic. Each table needed thousands of trajectory calculations.
3. The Army funded John Mauchly and J. Presper Eckert at the Moore School of Electrical Engineering, University of Pennsylvania, to build a fully electronic calculator.
4. The result was **ENIAC**, the Electronic Numerical Integrator and Computer.
5. It filled a room, used nearly 18,000 vacuum tubes, weighed 30 tons and drew 150 kilowatts.
6. It did 5,000 additions a second, roughly a thousand times faster than the relay machines of the day.
7. It was finished in 1945 and shown to the press on 14 February 1946.
8. It had one enormous flaw: to change the problem, you rewired the machine.
9. That work took days or weeks of moving cables and setting thousands of switches by hand.
10. The people who did that work were six women, and for forty years almost nobody knew their names.

14.7.2 PLAIN - a picture in your head

1. Think of an old telephone exchange, the kind with a wall of sockets and a human operator pushing plugs into them.
2. Now imagine that instead of connecting callers, each plug decides which number flows into which adding unit.
3. To ask the machine a different question you do not type anything. You unplug forty cables and plug them somewhere else, and turn 3,000 switches.
4. Then you spend two days finding the one cable you put in the wrong hole.
5. That is ENIAC. The program was the physical shape of the wiring.

Where this comparison breaks: a telephone exchange connects two points at a time and then releases them. ENIAC's plugboard connections were the permanent data paths for the whole run, and the timing of pulses down those cables was the control flow. It is closer to laying out a factory production line than to placing a call.

14.7.3 PLAIN - a worked example

1. Here are ENIAC's real numbers, from the Moore School records.

Property	Value
Vacuum tubes	17,468
Crystal diodes	7,200
Relays	1,500
Resistors	70,000
Weight	Over 30 short tons
Floor space	About 1,800 sq ft
Power	150 kW
Additions per second	5,000
Cost	About $487,000

2. You will often see 18,000 tubes quoted. The precise figure is 17,468. Both appear in reputable sources; the rounder number came from press releases.
3. Arithmetic detail that surprises people: ENIAC was **decimal**, not binary. It had 20 accumulators, each holding 10 decimal digits and a sign.
4. A digit was stored as a ring of ten flip-flops, one of which was on. That is ten tubes to hold what one bit-pair would hold in binary.
5. Reliability was the real problem. With 17,468 tubes, a failure every day or two was expected. Eckert ran the tubes far below rated voltage and never switched the machine off, since switch-on is when tubes die.
6. ENIAC's first serious job, in December 1945, was a feasibility calculation for the hydrogen bomb, run for six weeks. Its very first task was not ballistics at all.
7. It was switched off for the last time at 11:45 in the evening on 2 October

1955.

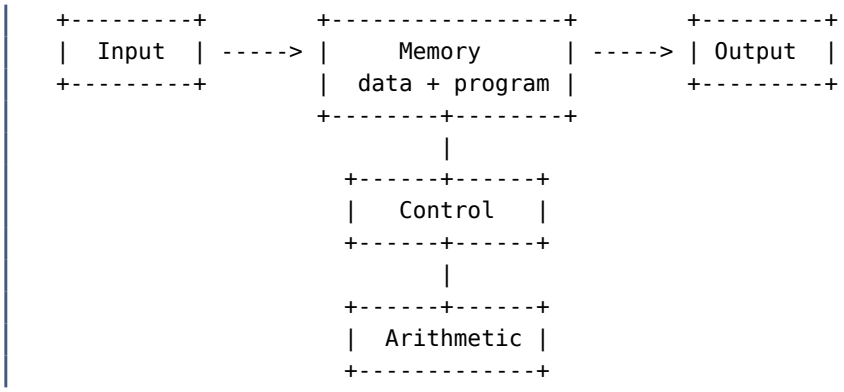
14.7.4 PLAIN - what is really happening inside

1. The Army had already hired about eighty women as human computers, working out trajectories on desk calculators.
2. Six of them were selected in 1945 to program ENIAC: Kathleen McNulty, Frances Bilas, Betty Jean Jennings, Ruth Lichterman, Elizabeth Snyder and Marlyn Wescoff.
3. They are also known by their later married names: McNulty Mauchly Antonelli, Bilas Spence, Jennings Bartik, Lichterman Teitelbaum, Snyder Holberton, and Wescoff Meltzer.
4. They were given wiring diagrams and told to work it out. There was no manual, no programming language and no course, because none existed.
5. They learned the machine from its blueprints, invented debugging techniques, and worked out how to break a calculation into parallel units.
6. Betty Snyder Holberton went on to write the first sort-merge generator and worked on the standards for COBOL and Fortran. Kathleen McNulty helped devise subroutine techniques on later machines.
7. On the day ENIAC was shown to the press in February 1946, the men who built the hardware were named. The women were not invited to the celebratory dinner.
8. For decades their photographs were captioned as models posing with the equipment. The phrase used later to describe this is "refrigerator ladies", because women in technical photographs were assumed to be advertising props.
9. Kathryn Kleiman began tracking them down from 1986. Their story reached a wide audience with the 1997 Women in Technology International Hall of Fame inductions and the 2014 documentary *The Computers*.

10. This is not a footnote about fairness. Programming as a distinct discipline, separate from building hardware, starts here, and these six people invented much of its practice.

14.7.5 TECHNICAL - the engineer's version

1. While ENIAC was still being wired, its team already knew rewiring was unacceptable. The successor project was called EDVAC.
2. John von Neumann joined the group as a consultant in 1944. On 30 June 1945 Herman Goldstine distributed a 101-page document titled *First Draft of a Report on the EDVAC*.
3. It described a machine with five parts: a central arithmetic unit, a central control unit, memory, input and output.
4. Its crucial claim was that instructions and data live in the same read-write memory, so a program is just numbers, and a program can therefore be loaded, changed and even modified by another program.



5. The naming controversy is real and worth stating precisely.
    1. The document carried only von Neumann's name. It was a draft circulated for comment, not a finished credited paper.
    2. Eckert and Mauchly said the stored-program concept came out of Moore School group discussions in 1944, before von Neumann arrived, and that he wrote up shared ideas in logical notation.
    3. Because the draft was circulated publicly more than a year before the patent filing, it counted as prior public disclosure. That helped invalidate the ENIAC patent claims later.
    4. Turing's 1936 universal machine had already stated the idea mathematically, and von Neumann had read Turing's paper.
6. Many historians therefore prefer the neutral terms "stored-program computer" or "Princeton architecture". The label "von Neumann architecture" is a **convention**, and a slightly unfair one.
7. What the name refers to today is a specific structural property: a single address space and a single bus for both instructions and data. The alternative, separate instruction and data memories, is the **Harvard architecture**, named after the Harvard Mark I.
8. The consequence engineers still fight is the **von Neumann bottleneck**, a phrase John Backus coined in his 1977 Turing Award lecture: the processor can only go as fast as the single path to memory allows.
9. Nearly every modern chip is a hybrid. Level 1 caches are usually split into separate instruction and data caches, which is Harvard-like, over a unified main memory, which is von Neumann. You can see this on Linux with `lscpu`, which lists L1d and L1i separately.
10. EDVAC itself was delivered in 1949 and became properly operational in 1951, by which time British machines had beaten it into service.

14.7.6 WORDS - remember these

1. ENIAC — the 1945 American electronic calculator — a decimal, plugboard programmed, 17,468-tube general-purpose electronic computer.
2. Accumulator — a box that holds a running total — a register that stores an arithmetic result, ten decimal digits on ENIAC.

3. Stored program — instructions kept in memory like data — a single writable address space holding both code and operands.
4. von Neumann architecture — the 1945 five-part design — a stored-program machine with unified instruction and data memory.
5. Harvard architecture — separate stores for code and data — physically distinct instruction and data memories and buses.
6. von Neumann bottleneck — the narrow pipe to memory — the throughput limit imposed by a single shared processor-memory path, named by Backus in 1977.

14.8 The first stored-program computers

14.8.1 PLAIN - in simple words

1. After 1945 the race was to build a machine that kept its program in memory.
2. The hard part was not logic. It was memory. Nobody had a fast, cheap, rewritable store.
3. Two solutions appeared. One used sound. One used a television tube.
4. The sound one is the **mercury delay line**. Send a pulse of sound down a tube of mercury, catch it at the far end, and send it round again. The data is literally in flight.
5. The television one is the **Williams tube**. Draw dots on the inside of a cathode ray tube. The charge left behind is the bit. Read it with a metal plate on the glass.
6. At Manchester, Freddie Williams and Tom Kilburn made the tube work, and then built the smallest possible computer to prove it.
7. That machine, the Small-Scale Experimental Machine, nicknamed the **Baby**, ran the world's first stored program on 21 June 1948.
8. At Cambridge, Maurice Wilkes built **EDSAC**, which first ran on 6 May 1949 and was the first machine to give a regular computing service to other people.
9. Manchester's full-size machine became the Manchester Mark 1, and the company Ferranti turned it into a product.
10. The **Ferranti Mark 1**, delivered in February 1951, was the first commercially available general-purpose electronic computer in the world.
11. In America, **UNIVAC I** was accepted by the Census Bureau on 31 March 1951 and became famous on election night in 1952.

14.8.2 PLAIN - a picture in your head

1. Picture a long queue of people passing a whispered message down the line and back to the front, over and over, so it is never forgotten.
2. That is a mercury delay line. The message exists only as sound travelling through liquid metal.
3. To read a particular bit, you wait for it to come round. If you just missed it, you wait a full lap.
4. Now picture instead a chalk mark on a blackboard that fades in a fraction of a second, so a helper walks along constantly redrawing every mark.
5. That is a Williams tube. Any dot you can see, you can read at once, in any order.

Where this comparison breaks: the queue is not just slow, it is strictly sequential, which changed how programs were written. On EDSAC, and even more on drum-memory machines, programmers placed instructions at positions on the loop so the next one arrived just as it was needed. That practice is called optimum coding, and it is the ancestor of today's cache-aware programming.

14.8.3 PLAIN - a worked example

1. The Baby's first program, written by Tom Kilburn, found the highest proper factor of 2 to the power 18, which is 262,144.
2. The method was the crudest possible: try every candidate downwards from 262,143 and test by repeated subtraction.

3. It was 17 instructions long and it lived in a Williams tube holding 32 words of 32 bits, which is 1,024 bits in total.
4. It ran for about 52 minutes and performed roughly 3.5 million operations, which is around 1,100 operations per second.
5. The answer, of course, is 131,072. The point was never the answer. The point was that the program was in memory.
6. Compare that with EDSAC's first run on 6 May 1949, which printed a table of the squares of 0 to 99, and then a list of prime numbers.

Machine	First ran	Memory
Manchester Baby	21 Jun 1948	32 words, Williams tube
EDSAC	6 May 1949	512 words, mercury
Manchester Mark 1	Apr 1949	Tube plus drum
Ferranti Mark 1	Feb 1951	256 words, plus drum
UNIVAC I	1951	1,000 words, mercury

14.8.4 PLAIN - what is really happening inside

1. EDSAC's real importance is not the hardware. It is that Cambridge treated computing as a service and a discipline.
2. David Wheeler invented the **subroutine** there: a block of instructions you can call from anywhere and return from. The return mechanism is still called the Wheeler jump.
3. Wilkes, Wheeler and Gill published *The Preparation of Programs for an Electronic Digital Computer* in 1951. It is the first programming textbook, and it introduced the idea of a library of reusable routines.
4. Wheeler also received what is generally described as the world's first doctorate in computer science, in 1951.
5. EDSAC also produced the first business computing. The catering company J. Lyons and Company built a copy called LEO, the Lyons Electronic Office, which ran its first routine business job on 17 November 1951, valuing a bakery's output.
6. A tea shop company thus ran payroll on a computer before most governments did. That is a genuine and under-told fact.
7. The Manchester Mark 1 contributed the **index register**, which Manchester called a B-line: a register whose value is added to an address before use.
8. That one idea makes loops over arrays possible without a program rewriting its own instructions, and every processor since has it.

14.8.5 TECHNICAL - the engineer's version

1. Williams-Kilburn tube: a standard CRT storing charge on the phosphor. Typical capacity 1,024 to 2,048 bits per tube. Access time about 10 to 30 microseconds and, crucially, random access. It needed constant refresh and was sensitive to nearby electrical noise, even to someone opening a door.
2. Mercury delay line: a tube roughly 1.5 metres long. Sound travels in mercury at about 1,450 metres per second, giving a loop time near 1 millisecond and holding several hundred bits. Average latency is half a loop.
3. EDSAC: 3,000 valves, about 11 kW, 512 words of 17 bits initially, cycle time 1.5 ms for most instructions and 6 ms for multiplication. It ran until 11 July 1958.
4. Ferranti Mark 1: delivered to the University of Manchester in February 1951, publicly demonstrated in July 1951. Around nine machines including the improved Mark 1 star were built between 1951 and 1957.
5. Priority claims, stated carefully, because people argue about them:
    1. First stored program executed: Manchester Baby, 21 June 1948.

   2. First regular computing service: EDSAC, from 1949.
   3. First commercially available general-purpose computer delivered: Ferranti Mark 1, February 1951.
   4. First American commercial delivery: UNIVAC I, accepted by the Census Bureau on 31 March 1951, physically installed later that year.
   5. Zuse's Z4 was delivered to ETH Zurich in July 1950 and was electromechanical, so it is usually excluded from electronic firsts.
6. UNIVAC I specifications: 5,200 vacuum tubes, about 125 kW, 1,000 words of 12 characters in mercury memory, about 1,905 operations per second, 46 machines built, and a price that rose from a quoted $159,000 to well over a million dollars.
7. The 1952 election. On CBS, on 4 November 1952, UNIVAC was given 5.5 per cent of the returns and predicted a landslide for Dwight Eisenhower over Adlai Stevenson, at 438 electoral votes to 93.
8. Nobody believed it, because pollsters expected a close race. CBS staff adjusted a trend factor from 40 per cent to 4 per cent to produce a respectable-looking 268 to 263, and broadcast that instead.
9. The actual result was 442 to 89. The original prediction was within four electoral votes. Late that night Remington Rand's Arthur Draper admitted on air that the machine had been right and had been overruled.
10. That broadcast, more than any technical paper, is what put the word "computer" into ordinary speech.

14.8.6 WORDS - remember these

1. Mercury delay line — memory made of sound in a tube — a sequential-access acoustic recirculating store, about 1 ms loop time.
2. Williams tube — memory made of dots on a screen — a CRT charge-storage random-access memory, about 1 to 2 kilobits per tube.
3. Baby — the Manchester test machine of 1948 — the Small-Scale Experimental Machine, first to execute a program held in memory, 21 June 1948.
4. Subroutine — a reusable block of instructions — a callable routine with a return mechanism, invented by David Wheeler on EDSAC.
5. Index register — a register that shifts an address — a B-line, added to an operand address before memory access, from the Manchester Mark 1.
6. LEO — the tea company's computer, 1951 — Lyons Electronic Office, the first computer used for routine commercial work.

14.9 Transistors, mainframes and minicomputers

14.9.1 PLAIN - in simple words

1. A vacuum tube is a small glass bulb that controls electricity. It works, but it is hot, fragile, power-hungry and it burns out.
2. On 23 December 1947, John Bardeen and Walter Brattain at Bell Labs demonstrated the **transistor**: a lump of germanium doing the same job with no glass, no heater and almost no power.
3. William Shockley followed with the more manufacturable junction transistor, published in 1951. The three shared the Nobel Prize in Physics in 1956.
4. Machines built from transistors were smaller, cooler and ran for months instead of hours between failures.
5. Through the 1950s IBM sold its 700 series, built from tubes, and then the 7000 series, built from transistors.
6. Every one of those machines had its own instruction set, so software written for one would not run on the next.
7. On 7 April 1964 IBM announced the **System/360**: one family of machines, from small to huge, all running the same programs.

8. That single decision made software an asset you keep instead of a cost you repeat, and it made IBM dominant for twenty years.
9. Meanwhile a small company called Digital Equipment Corporation went the other way, building machines cheap enough for one laboratory to own.
10. Those were **minicomputers**, and they took computing out of the glass room and put it next to the people doing the work.

14.9.2 PLAIN - a picture in your head

1. A mainframe is a city power station. It is enormous, it is behind a fence, and you apply for a connection.
2. Programs arrive as decks of cards, are run overnight in a queue, and results come back as printout the next morning. That is **batch processing**.
3. A minicomputer is a generator in your own workshop. Smaller, noisier, less efficient, but yours, and switched on when you want it.
4. Nobody schedules you. You sit at the machine, you type, it answers.
5. System/360 is a third idea: not one power station but a whole product range, from a small generator to a station, all taking the same plug.

Where this comparison breaks: minicomputers were not personal. A PDP-11 cost tens of thousands of dollars and served a whole department, usually through several terminals at once. The truly personal machine is still a decade away in this story.

14.9.3 PLAIN - a worked example

1. Here is why compatibility mattered, in money.
2. Say a bank has 200 programs, each taking 6 months of one programmer's time to write. That is 100 programmer-years of investment.
3. Before 1964, buying a bigger machine meant rewriting most of that. The software bill could exceed the hardware bill.
4. With System/360, the same program ran on a Model 30 and on a Model 75, which was roughly fifty times faster.
5. You bought a bigger box and your software simply ran faster. Nothing else changed.

Machine	Year	Note
IBM 701	1952	Tubes, 19 built
IBM 650	1954	Drum, about 2,000 built
IBM 704	1954	Fortran's first host
IBM 7090	1959	Transistorized 709
IBM S/360 Model 30	1965	34,500 instr/sec
IBM S/360 Model 91	1967	16.6 million instr/sec

14.9.4 PLAIN - what is really happening inside

1. The trick that made one family possible is **microprogramming**, invented by Maurice Wilkes at Cambridge in 1951.
2. Instead of building each instruction directly out of logic gates, you build a small, fast inner machine, and store a tiny program for each instruction in read-only memory.
3. A cheap model runs the same instruction set with slow, simple hardware and more microcode steps. An expensive model uses fast hardware and fewer steps.
4. Same behaviour, wildly different cost. That is how one architecture spanned a fifty-to-one performance range.
5. Digital Equipment Corporation, founded in 1957 by Ken Olsen and Harlan Anderson, attacked from below.

6. The **PDP-1** of 1959 cost $120,000 when mainframes cost millions. Fifty-three were built. On one of them, in 1962, Steve Russell and friends at MIT wrote the game Spacewar, one of the first video games.
7. The **PDP-8**, launched on 22 March 1965 at $18,500, was the first computer sold for under $20,000. Over 50,000 were sold, and more than 300,000 across the whole family.
8. The **PDP-11**, announced in January 1970, sold around 600,000 units over its life. Unix was first written for it, and the C language grew up on it.

14.9.5 TECHNICAL - the engineer's version

1. System/360 announcement, 7 April 1964: six models initially, numbered 30, 40, 50, 60, 62 and 70. The last three were replaced before shipping by the 65 and 75.
2. Development cost is usually quoted as about $5 billion, from a September 1966 Fortune article calling it IBM's five-billion-dollar gamble. Adjusted for inflation that is roughly $50 billion today.
3. Architectural decisions from System/360 that are still with us:
    1. The 8-bit **byte** as the standard addressable unit. Before 360, machines used 6-bit characters and word sizes of 36, 48 or 60 bits.
    2. 32-bit words, 24-bit addressing, and general-purpose registers.
    3. A separate channel subsystem for input and output, which is the ancestor of DMA controllers.
4. Gene Amdahl was chief architect. Fred Brooks managed the OS/360 software project, and later wrote *The Mythical Man-Month* in 1975 about how badly it went. Its most quoted line: adding people to a late software project makes it later.
5. IBM's dominance drew the United States Department of Justice, which filed an antitrust suit on 17 January 1969. It ran for thirteen years and was dropped on 8 January 1982 as being without merit.
6. The rest of the industry was known as the BUNCH: Burroughs, UNIVAC, NCR, Control Data and Honeywell. The older joke was "IBM and the seven dwarfs".
7. Mainframes did not die. IBM z16 systems, announced in April 2022, still run code compiled for System/360 in 1964. That is 60 years of binary compatibility, the longest unbroken run in the industry.

14.9.6 WORDS - remember these

1. Transistor — a solid switch with no glass bulb — a semiconductor device controlling current, demonstrated at Bell Labs in December 1947.
2. Mainframe — a big shared central computer — a high-throughput, high-reliability system optimized for input and output volume.
3. Minicomputer — a machine a department can afford — a mid-range system, typically 12 to 16 bits, priced from about $18,500 in 1965.
4. Batch processing — hand in your job, come back tomorrow — non-interactive scheduled execution of queued jobs.
5. Microprogram — a tiny program inside the processor — microcode in control store implementing each machine instruction, invented by Wilkes in 1951.
6. Binary compatibility — old programs still run — the ability to execute unmodified object code on a newer implementation.

14.10 The integrated circuit and Silicon Valley

14.10.1 PLAIN - in simple words

1. By 1958 a computer was thousands of separate parts joined by hand-soldered wires. Every joint was a chance to fail.
2. Engineers called this the tyranny of numbers: to make a machine more capable you needed more parts, and more parts meant it broke sooner.
3. On 12 September 1958, Jack Kilby at Texas Instruments demonstrated a different idea: build all the parts out of one piece of semiconductor, so there are no joints at all.

4. His device worked but used tiny gold wires bonded by hand across the surface. It was a proof, not a product.
5. In 1959 Robert Noyce at Fairchild Semiconductor solved the manufacturing problem, using a silicon wafer and printing the connections on as a flat layer of aluminium.
6. That is the **integrated circuit**, or chip.
7. Both men are credited. Kilby received the Nobel Prize in Physics in 2000; Noyce had died in 1990 and Nobel prizes are not awarded posthumously.
8. Noyce's version won commercially because it could be made in quantity.
9. The reason it could be made in quantity was a process invented by his colleague Jean Hoerni, also in 1959, called the **planar process**.
10. All of this happened in a strip of orchards south of San Francisco, which a journalist named Silicon Valley in 1971.

14.10.2 PLAIN - a picture in your head

1. Think of printing a newspaper instead of writing each copy by hand.
2. Old way: draw every letter individually, then glue the pages together.
3. New way: make one master plate, then print thousands of identical copies at almost no extra cost per copy.
4. A chip is printed. Light is shone through a mask onto a light-sensitive coating on the silicon, and the pattern is etched in.
5. Printing 10 components costs the same as printing 10 million. That is the entire economics of the computer industry in one sentence.

Where this comparison breaks: newspapers do not care about a single dust speck. A chip does. One particle in the wrong place kills the die, so yield, the fraction of good chips per wafer, dominates cost. That is why chips are made in cleanrooms with fewer particles per cubic metre than an operating theatre.

14.10.3 PLAIN - a worked example

1. The planar process in five steps, which is still roughly what happens today.

```
1. Oxidize   : grow a glass layer of SiO2 on the silicon
2. Mask      : coat with resist, shine light through a mask
3. Etch      : dissolve the exposed oxide, opening windows
4. Diffuse   : drive dopant atoms in through the windows
5. Metallize : evaporate aluminium on top, then pattern it
```

2. The key point is step 1. The oxide stays on as a permanent protective and insulating layer.
3. Because the surface is flat and insulated, metal wiring can be laid across the top without shorting to anything underneath.
4. Kilby's germanium device had no such layer, so his connections were flying wires. Hoerni's flat oxide is what made mass production possible.
5. Kurt Lehovec at Sprague Electric added the other missing piece in 1959: isolating neighbouring components with reverse-biased junctions.

14.10.4 PLAIN - what is really happening inside

1. William Shockley left Bell Labs and set up Shockley Semiconductor Laboratory in Mountain View, California, in 1956, the year he won the Nobel Prize.
2. He hired brilliant people and managed them badly. He posted salaries publicly and required staff to take lie detector tests.
3. On 18 September 1957 eight of them resigned together. Shockley called them the **traitorous eight**.
4. They were Julius Blank, Victor Grinich, Jean Hoerni, Eugene Kleiner, Jay Last, Gordon Moore, Robert Noyce and Sheldon Roberts.

5. Sherman Fairchild financed them with a loan of $1.38 million, and Fairchild Semiconductor was born.
6. Fairchild then leaked people the way Shockley had. Companies founded by ex-Fairchild staff are called Fairchildren, and there are dozens.
7. On 18 July 1968 Noyce and Moore left to found Intel, backed by the venture capitalist Arthur Rock.
8. On 1 May 1969 Jerry Sanders, Fairchild's head of marketing, founded Advanced Micro Devices with seven colleagues.
9. Eugene Kleiner went on to co-found the venture firm Kleiner Perkins in 1972, which is how the engineering diaspora became a financial one too.
10. The name arrived on 11 January 1971, when the journalist Don Hoefler ran a series titled "Silicon Valley U.S.A." in the trade paper Electronic News.
11. Silicon is the element the chips are made from; the valley is the Santa Clara Valley. The name did not become common speech until the early 1980s.

14.10.5 TECHNICAL - the engineer's version

1. Kilby's 12 September 1958 device: a phase-shift oscillator on a germanium bar about 11 by 1.6 millimetres, with gold wire interconnect. Texas Instruments filed the patent in February 1959.
2. Noyce's Fairchild device: silicon, oxide-passivated by Hoerni's planar process, with evaporated aluminium interconnect. Patent filed 30 July 1959.
3. The two companies litigated for a decade and settled with cross-licensing in
1966. Courts eventually favoured Noyce on the interconnect claims and Kilby on the integration concept.
4. Gordon Moore's observation was published in Electronics magazine on 19 April
1965. He noted the number of components per chip had roughly doubled each year, and predicted it would continue for a decade.
5. In 1975 Moore revised the doubling period to about two years. The often-quoted eighteen months is a later paraphrase attributed to Intel colleague David House, combining transistor count and speed.
6. Standard, convention or observation? Moore's law is none of a standard, a law of physics or a theorem. It is an economic observation that became a planning target the industry then organized itself to meet.

Item	Year	Transistors
Kilby prototype IC	1958	About 1
Intel 4004	1971	2,300
Intel 80386	1985	275,000
Intel Pentium	1993	3.1 million
Apple M1	2020	16 billion
Nvidia Blackwell B100	2024	208 billion

14.10.6 WORDS - remember these

1. Integrated circuit — many parts on one piece of silicon — a monolithic circuit fabricated on a single semiconductor die.
2. Planar process — the flat oxide-layer way of making chips — Hoerni's 1959 oxide passivation and diffusion method enabling deposited interconnect.
3. Wafer — the silicon disc chips are printed on — a monocrystalline substrate, today 300 mm in diameter.
4. Yield — how many chips per wafer actually work — the ratio of functional die to total die, the dominant cost factor.
5. Moore's law — chips double in capacity every couple of years — the 1965 observation on component count per die, revised to two years in 1975.
6. Traitorous eight — the group who quit Shockley in 1957 — the founders of Fairchild Semiconductor and, indirectly, of Silicon Valley.

14.11 The microprocessor

14.11.1 PLAIN - in simple words

1. By 1969 you could put thousands of transistors on one chip, but a computer's processor was still spread over many chips and several boards.
2. In April 1969 a Japanese calculator company called Busicom asked Intel, then a young memory company, for a set of chips for a printing calculator.
3. Busicom's own design used twelve custom chips, each hardwired for one part of the job.
4. An Intel engineer, Ted Hoff, said that was backwards. Build one small general-purpose processor, and put the calculator's behaviour in a program in read-only memory.
5. Stanley Mazor helped work out the instruction set. Busicom's engineer Masatoshi Shima wrote much of the calculator software and checked the design.
6. Nobody at Intel could actually build it until Federico Faggin joined in April 1970 and invented the circuit techniques needed to fit it on one die.
7. The result was the **Intel 4004**, announced on 15 November 1971.
8. It had 2,300 transistors, a 4-bit word, ran at up to 740 kilohertz, and cost $60.
9. That single chip had roughly the computing power of ENIAC, which had filled a room in 1946.
10. Intel then bought back the exclusive rights from Busicom, which was in financial trouble, by refunding $60,000 of development cost. That is how a calculator part became a product anyone could buy.

14.11.2 PLAIN - a picture in your head

1. Think of a kitchen that makes one dish. Every worktop, jig and mould is shaped for that dish alone. Change the menu and you rebuild the kitchen.
2. Now think of a kitchen with one general chef, a knife, a pan and a recipe card.
3. The second kitchen is slower at the one dish, but it can cook anything, and changing the menu costs the price of a card.
4. The 4004 is the general chef. The read-only memory chip is the recipe card.
5. Because one design serves a thousand products, you can afford to make ten million of them, and the price collapses.

Where this comparison breaks: general-purpose chips did not simply beat custom ones. Custom silicon is still faster and cheaper per unit for very high volume fixed jobs, which is why your phone contains dozens of fixed-function blocks for video, radio and camera work alongside its general processors.

14.11.3 PLAIN - a worked example

1. Compare the 4004 with ENIAC, which is a fair fight on arithmetic rate.

Property	ENIAC, 1946	Intel 4004, 1971
Size	About 1,800 sq ft	12 square mm
Weight	Over 30 tons	Under 1 gram
Power	150 kW	Under 1 watt
Additions per second	5,000	About 92,000
Cost	About $487,000	$60

2. Twenty-five years took a room-sized machine to a fingernail-sized chip, at about one eight-thousandth of the price.
3. The 4004 was made on a 10 micrometre pMOS silicon-gate process. Today's leading process is around 3 nanometres, which is roughly 3,000 times finer in each direction.

14.11.4 PLAIN - what is really happening inside

1. The 4004 was a starting gun, not a finish line. The useful chips came next.
2. The **Intel 8008** arrived in April 1972. It was 8-bit and came from a separate contract for the Datapoint 2200 terminal, whose maker rejected it.
3. The **Intel 8080** arrived in April 1974, with about 4,500 transistors and a 2 megahertz clock. It was fast enough to be the heart of a real computer, and it powered the Altair 8800.
4. Motorola answered with the **6800** in 1974. It needed only a single 5-volt supply, which made board design far simpler.
5. Federico Faggin left Intel at the end of 1974 and founded Zilog. With Masatoshi Shima he designed the **Z80**, released in July 1976.
6. The Z80 ran all 8080 software, added instructions and registers, needed one power supply, and refreshed dynamic memory itself. It became the most widely used 8-bit processor of the era.
7. The turning point for ordinary people was price. In August 1974 the Motorola 6800 and Intel 8080 were introduced at around $360 each.
8. Chuck Peddle had worked on the 6800 at Motorola. In August 1974 he and seven colleagues left for MOS Technology.
9. In September 1975, at the Wescon show in San Francisco, they sold the **MOS 6502** for $25, out of jars on a table, because nobody believed the price otherwise.
10. Twenty-five dollars instead of three hundred and sixty is what made a home computer possible. The 6502 went into the Apple I and II, the Commodore PET and 64, the BBC Micro, the Atari 2600 and the Nintendo Entertainment System.

14.11.5 TECHNICAL - the engineer's version

1. Faggin's enabling technique was **silicon gate MOS**, which he had developed at Fairchild in 1968. Using polysilicon rather than aluminium for the gate makes it self-aligning, which shrinks parasitic capacitance and allows buried contacts. Without it the 4004 would not have fitted on one die.
2. The 4004 shipped as part of the MCS-4 chip set: 4001 ROM, 4002 RAM, 4003 shift register and 4004 CPU. On its own the processor is not a computer.
3. It used a 4-bit data bus, a 12-bit address space of 4 kilobytes of program memory, 46 instructions, and a 16-level address stack in hardware.

Processor	Introduced	Bits	Transistors
Intel 4004	Nov 1971	4	2,300
Intel 8008	Apr 1972	8	3,500
Intel 8080	Apr 1974	8	4,500
Motorola 6800	1974	8	About 4,100
MOS 6502	Sep 1975	8	About 3,510
Zilog Z80	Jul 1976	8	About 8,500

4. The 6502 was cheap for two engineering reasons, not just aggressive pricing. First, MOS Technology fixed mask defects rather than discarding the wafer, raising yield dramatically. Second, the design is minimal: three 8-bit registers, a hardware stack fixed at page one, and no microcode.
5. Its zero-page addressing mode treats the first 256 bytes of memory as quasi-registers, which is how a chip with so few registers stayed fast.
6. Legal note: MOS Technology first shipped the 6501, which was pin-compatible with the Motorola 6800. Motorola sued in November 1975. MOS withdrew the 6501 and settled in 1976, paying $200,000. The 6502 was not pin-compatible and survived.

14.11.6 WORDS - remember these

1. Microprocessor — a whole processor on one chip — a monolithic CPU, first commercially released as the Intel 4004 in November 1971.
2. Silicon gate — a way of building transistors with polysilicon gates — self-aligned MOS process reducing gate capacitance, Faggin 1968.
3. Instruction set — the list of things a chip understands — the architectural contract between hardware and compiled code.
4. Read-only memory — the chip holding the fixed program — ROM, non-volatile storage written at manufacture or once thereafter.
5. Zero page — the first 256 bytes of memory on a 6502 — a short-addressed region used as extended register space.

14.12 The personal computer revolution

14.12.1 PLAIN - in simple words

1. In December 1974 a magazine cover changed everything. The January 1975 issue of Popular Electronics showed a blue box called the **Altair 8800**.
2. It was made by MITS, a small company in Albuquerque run by Ed Roberts, and it cost $439 as a kit or $621 assembled.
3. It had an Intel 8080, 256 bytes of memory, no keyboard, no screen and no storage. You entered programs with switches and read answers as lights.
4. MITS hoped to sell a few hundred. They had 1,000 orders in February 1975 alone, and had shipped over 5,000 by August.
5. Two young men in Boston saw the cover. Bill Gates and Paul Allen phoned MITS and claimed to have a BASIC language for it. They did not.
6. They wrote it in eight weeks on a simulated 8080 running on a PDP-10, having never touched an Altair. Paul Allen flew to Albuquerque, and it worked first time.
7. They founded **Microsoft** on 4 April 1975 to sell it.
8. On 5 March 1975 about thirty hobbyists met in Gordon French's garage in Menlo Park, California. That was the first **Homebrew Computer Club**.
9. Two members were Steve Wozniak and Steve Jobs. Apple came out of that room.
10. From there the story accelerates: Apple in 1976 and 1977, three rival home computers in 1977, the first spreadsheet in 1979, and IBM's entry in 1981.

14.12.2 PLAIN - a picture in your head

1. Think of the first cheap cars. They were sold as parts, bought by mechanics, and driven by people who could fix them at the roadside.
2. Then somebody sold a car with a roof, a starter motor and a key, and ordinary people bought it.
3. The Altair is the kit. The Apple II with a plastic case and a power supply is the car with a roof.
4. But nobody buys a car because it has a roof. They buy it to get somewhere.
5. The place people wanted to get to turned out to be a spreadsheet.

Where this comparison breaks: cars had obvious uses before they were easy to drive. In 1977 nobody could say what a home computer was for. That question was genuinely open, and shop owners struggled to answer it, until VisiCalc gave a concrete answer for businesses in 1979.

14.12.3 PLAIN - a worked example

1. The three machines of 1977, often called the 1977 trinity, arrived within months of each other and defined the market.

Machine	Launched	Price and CPU
Apple II	10 Jun 1977	$1,298, 6502
Commodore PET 2001	Oct 1977	$795, 6502
TRS-80 Model I	Aug 1977	$599.95, Z80

2. All three included BASIC in read-only memory, so the machine was usable the moment it was switched on.
3. The TRS-80 sold about 55,000 in its first year, helped by Radio Shack's 3,000 stores. Over 200,000 were sold in total.
4. The Apple II sold about 6 million across all models between 1977 and 1993.
5. Apple Computer had been founded on 1 April 1976 by Steve Jobs, Steve Wozniak and Ronald Wayne. Wayne sold his ten per cent share back for $800 twelve days later.
6. The Apple I went on sale in July 1976 at $666.66. About 200 were built. The first order was 50 units from Paul Terrell's Byte Shop at $500 each, on condition they were assembled boards.

14.12.4 PLAIN - what is really happening inside

1. VisiCalc was released on 17 October 1979 for the Apple II, at under $100.
2. Dan Bricklin, a Harvard business student, imagined it. Bob Frankston wrote it. Personal Software, later VisiCorp, published it.
3. It is the first **killer application**: a program so useful that people buy the hardware in order to run it.
4. More than 700,000 copies sold in six years. Reports at the time suggested over a quarter of Apple IIs sold in 1979 were bought to run it.
5. That commercial signal is what brought IBM in. IBM normally took years to develop a product. This time it gave a team in Boca Raton, Florida, one year.
6. The project was code-named Chess, led by William C. Lowe and then Don Estridge. To hit the deadline they broke every IBM rule and bought parts from outside.
7. The **IBM PC model 5150** was announced on 12 August 1981: an Intel 8088 at 4.77 megahertz, 16 kilobytes of memory as standard, from $1,565.
8. The decision that made history was openness. IBM published a technical reference manual containing the full circuit diagrams and the complete source listing of the BIOS, the start-up firmware.
9. That let anyone build add-in cards and software. It also let anyone build a copy of the machine, provided they could produce a BIOS without copying IBM's code.
10. Compaq did exactly that. One team of engineers documented what the IBM BIOS did; a second team, who had never seen IBM's code, wrote a new one from that description alone. That procedure is called **clean room** design.
11. The Compaq Portable was announced in November 1982 and shipped in March 1983 at $2,995. Compaq sold 53,000 units and made $111 million in its first year, an American record at the time.
12. Once Phoenix Technologies began selling a compatible BIOS to anyone in 1984, the clone industry exploded, and the PC stopped being IBM's product and became an industry standard.

14.12.5 TECHNICAL - the engineer's version

1. How Microsoft got MS-DOS, with dates, because this is constantly told wrong.
    1. IBM approached Digital Research, maker of CP/M, the dominant 8-bit operating system. No deal was reached. The famous story that Gary Kildall went flying instead of meeting IBM is disputed by participants and should not be stated as fact.
    2. Microsoft agreed to supply an operating system it did not have.
    3. Tim Paterson at Seattle Computer Products had written 86-DOS, nicknamed QDOS for Quick and Dirty Operating System, starting in April 1980 and shipping in August 1980. It was modelled on CP/M but for the 8086.
    4. In December 1980 Microsoft licensed it non-exclusively for $25,000.

   5. In July 1981 Microsoft bought all rights for a further $50,000, and hired Paterson.
   6. IBM shipped it in August 1981 as PC DOS 1.0.
2. The commercial masterstroke was not the purchase. It was the contract: Microsoft licensed to IBM non-exclusively and kept the right to sell MS-DOS to anyone else. Within a year it had licensed it to more than 70 companies.
3. So Bill Gates did not write MS-DOS, and Microsoft did not originally own it. Both facts are commonly got wrong.
4. IBM PC 5150 hardware, for the record: Intel 8088, 4.77 MHz, 8-bit external bus with 16-bit internal registers, 1 megabyte address space, five expansion slots, cassette port, and 16 to 64 kilobytes on the system board.
5. The 8088 was chosen over the faster 8086 precisely because its 8-bit external bus let IBM reuse cheap existing 8-bit support chips.
6. Three operating systems were offered at launch: PC DOS at $40, CP/M-86 at $240, and UCSD p-System. Price decided the outcome.
7. The legal foundation for clones is that copyright protects the expression of the BIOS code, not the interface it presents. Clean room procedure documents that the new expression was created independently. This principle was reinforced in *Computer Associates v. Altai* in 1992.

14.12.6 WORDS - remember these

1. Kit computer — a computer you solder together — an unassembled system such as the Altair 8800 of January 1975.
2. Killer application — the program worth buying a computer for — software whose demand drives hardware adoption, first VisiCalc in 1979.
3. BIOS — the start-up firmware in a PC — Basic Input/Output System, the low-level routines between hardware and operating system.
4. Clean room design — copying behaviour without copying code — reimplementation from a specification written by a separate team with no access to the original code.
5. Clone — a machine that runs the same software — a hardware-compatible system using an independently written BIOS.
6. Open architecture — published plans anyone can build for — documented interfaces and bus specifications permitting third-party hardware.

14.13 The graphical interface

14.13.1 PLAIN - in simple words

1. Until the 1980s you talked to a computer by typing commands and reading text.
2. The alternative was demonstrated on 9 December 1968 in San Francisco, by Douglas Engelbart of the Stanford Research Institute.
3. In ninety minutes he showed a mouse, windows on screen, clickable linked text, shared editing of a document with a colleague miles away, and video of that colleague on the same screen.
4. His system was called NLS, the oN-Line System. The demonstration is now called the Mother of All Demos.
5. Almost nothing he showed was available to buy for fifteen years.
6. In 1970 Xerox opened a research centre in Palo Alto, called PARC, and hired many of Engelbart's people.
7. On 1 March 1973 PARC had the **Alto** working: a personal computer with a screen made of individually controlled dots, a mouse, and a network socket.
8. PARC also produced **Smalltalk**, a programming system by Alan Kay's group in which overlapping windows, pop-up menus and the whole desktop idea were worked out.
9. Steve Jobs visited PARC in December 1979 and saw all of it.
10. Apple shipped the **Lisa** on 19 January 1983 at $9,995, and the **Macintosh** on 24 January 1984 at $2,495.

11. Microsoft shipped **Windows 1.0** on 20 November 1985 at $99, and Apple sued in 1988.

14.13.2 PLAIN - a picture in your head

1. A command line is like ordering food by writing a note in a language the kitchen understands, and getting a note back.
2. A graphical interface is like walking to a counter where the dishes are laid out, and pointing.
3. Pointing needs three things: a picture of the choices, a way to point, and instant feedback when you do.
4. The picture needs a screen where every dot is separately controlled, which is called a **bitmap** display. That is far more memory than a text screen.
5. The pointing needs a mouse. The feedback needs the machine to redraw fast enough that the arrow feels attached to your hand.
6. That combination is why the idea waited from 1968 to 1984. It needed memory to become cheap.

Where this comparison breaks: pointing at a counter shows you only what is on display. A command line can express things no menu contains, such as "rename every file recorded before March and move it". Neither replaced the other, which is why every professional operating system in 2026 still has a terminal.

14.13.3 PLAIN - a worked example

1. Memory is the whole reason for the delay. Count the bits.
2. A 1970s text terminal: 80 columns by 24 rows of characters, one byte each. 80 times 24 is 1,920 bytes.
3. The Alto's screen: 606 by 808 dots, one bit each. That is 489,648 bits, about 61 kilobytes.
4. So a graphical screen needs about 32 times the memory of a text screen just to hold the picture, before any program runs.
5. The original Macintosh screen is 512 by 342, one bit per dot, which is 21,888 bytes out of a total of 128 kilobytes of memory.
6. Seventeen per cent of the whole machine was the picture on the screen.

Machine	Screen	Frame buffer
Text terminal	80 x 24 chars	About 2 kB
Xerox Alto, 1973	606 x 808 mono	About 61 kB
Apple Lisa, 1983	720 x 364 mono	About 33 kB
Macintosh, 1984	512 x 342 mono	About 22 kB

14.13.4 PLAIN - what is really happening inside

1. Engelbart and Bill English built the first mouse prototype in 1964, a wooden box with two wheels at right angles. The patent, number 3,541,541, was granted on 17 November 1970.
2. Xerox did sell a graphical machine. The Xerox Star, formally the 8010 Information System, launched in April 1981 with icons, windows, folders and a two-button mouse.
3. It cost $16,595 per workstation and needed a network of them to be useful. It sold poorly. Xerox invented the future and failed to price it.
4. The Apple visit of December 1979 was not theft. Xerox was given the right to buy 100,000 Apple shares before Apple's public offering, in exchange for showing its work.
5. Apple's engineers then built things PARC had not: the pull-down menu bar fixed at the top, click-and-drag direct manipulation, overlapping windows on a cheap machine, and the trash can.
6. The Lisa failed commercially. It was slow, its own disk format was unreliable, and it cost as much as a house deposit. Estimates of units sold range from about 10,000 to 60,000.
7. The Macintosh succeeded, helped by the "1984" advertisement directed by Ridley Scott, shown once during the Super Bowl on 22 January 1984.

8. Microsoft's Windows 1.0 could not overlap its windows. They were tiled, side by side, partly for technical reasons and partly because of Apple's licensing terms. Overlapping arrived with Windows 2.0 in December 1987.

14.13.5 TECHNICAL - the engineer's version

1. Alto specification: introduced 1 March 1973, designed largely by Charles P. Thacker; 128 kilobytes of memory expandable to 512; a 2.5 megabyte removable disk cartridge; a portrait bitmap display; Ethernet at 2.94 megabits per second, invented at PARC by Robert Metcalfe and David Boggs, whose founding memo is dated 22 May 1973. About 2,000 were built.
2. Smalltalk-72, Smalltalk-76 and Smalltalk-80 came from Alan Kay's Learning Research Group. Smalltalk is where object-oriented programming, the model-view-controller pattern and the modern integrated development environment were worked out together.
3. Bravo, written by Charles Simonyi in 1974 for the Alto, is the first what-you-see-is-what-you-get document editor. Simonyi later joined Microsoft and led Word and Excel.
4. Windows release history, with dates:

Version	Released	Notable change
Windows 1.0	20 Nov 1985	Tiled windows only
Windows 2.0	Dec 1987	Overlapping windows
Windows 3.0	22 May 1990	First big commercial hit
Windows 3.1	Apr 1992	TrueType fonts
Windows 95	24 Aug 1995	Start menu, taskbar

5. Windows 95 sold about one million copies in four days and was supported by an advertising campaign costing hundreds of millions of dollars. It is the moment the graphical interface became the default for ordinary computers.
6. *Apple Computer, Inc. v. Microsoft Corp.*, the look-and-feel case:
    1. Filed in 1988 against Microsoft and Hewlett-Packard.
    2. Apple listed 189 interface elements it said were copied.
    3. Microsoft's defence rested on a 1985 agreement licensing Apple interface elements for Windows 1.0. The district court found 179 of the 189 elements were covered by that licence.
    4. For the rest, the court applied a "virtual identity" standard, because the remaining similarities were unprotectable ideas or the only practical way of expressing them.
    5. The Ninth Circuit affirmed in 1994. The Supreme Court declined to hear the case in 1995.
    6. Xerox sued Apple during the same period and lost, mainly on timing.
    7. All remaining disputes were settled in August 1997, with Microsoft investing $150 million in non-voting Apple stock.
7. The lasting legal principle: an interface idea, such as overlapping windows or icons for files, is not copyrightable. Its specific artwork and code are.

14.13.6 WORDS - remember these

1. Graphical user interface — pointing at pictures instead of typing — a GUI, using windows, icons, menus and a pointer.
2. Bitmap display — a screen where every dot is separately set — a raster framebuffer with one or more bits per pixel.
3. Mouse — the pointing box — an X-Y position indicator, prototyped by Engelbart and English in 1964, patented in 1970.
4. WYSIWYG — the screen looks like the printout — what you see is what you get, first in Bravo on the Alto in 1974.

5. Desktop metaphor — files as paper on a desk — the interface model of the Xerox Star, 1981, and its successors.
6. Look and feel — the overall feel of an interface — the contested subject of Apple v Microsoft, largely held unprotectable in 1994.

14.14 Networks, the web and mobile: the spine only

14.14.1 PLAIN - in simple words

1. This section is deliberately short. Chapter 30 tells the story of networks and the web in full, and Chapter 31 tells the story of mobile.
2. Here you get only the spine, so the timeline in this chapter is not missing a limb.
3. Networks: in 1969 the United States funded ARPANET, a network joining four university computers so researchers could share machines.
4. The first message was sent on 29 October 1969, from UCLA to Stanford Research Institute, by a student named Charley Kline. He typed L, then O, and the system crashed before the G.
5. The web: in March 1989 Tim Berners-Lee, a British engineer at CERN in Switzerland, wrote a proposal for a linked document system. His boss's note on it read "vague but exciting".
6. Mobile: phones went from car-boot-sized analogue radios in 1983 to pocket computers with a screen in 2007.

14.14.2 PLAIN - a picture in your head

1. A network is a postal system that never asks who you are, only where the letter goes.
2. The web is a library laid on top of that postal system, where every book can contain the address of another book.
3. Mobile is the same postal system with the address being a moving person rather than a fixed building.
4. Each layer is built on the one below and does not know how it works.

Where this comparison breaks: the post office delivers whole letters, while a network chops your message into small numbered pieces that may travel by different routes and arrive out of order. Reassembly is the receiver's job.

14.14.3 PLAIN - a worked example

1. You can watch this history with one command on your own machine.

```
traceroute github.com
 1  192.168.0.1
 2  172.31.0.17
 3  137.97.29.249
 ...
 7  ae66-0.del01-96cbe-1b.ntwk.msn.net
 8  be23.rwa02.bom01.ntwk.msn.net
10  be5.ibr02.pnq21.ntwk.msn.net
```

2. Every line is a machine that looked at your packet, decided where next, and passed it on.
3. That "look and forward" behaviour is exactly what the first ARPANET nodes did in 1969, running on Honeywell minicomputers called Interface Message Processors.
4. The addresses beginning 172 are private ranges reserved by RFC 1918. The names ending msn.net are one company's naming convention, not a standard.
5. Fifty-seven years separate the first hop and the last, and the basic idea has not changed.

14.14.4 PLAIN - what is really happening inside

1. The spine of networking, in eight dates.
2. 1969: ARPANET's first four nodes.
3. 1974: Vinton Cerf and Robert Kahn publish the design of TCP, the protocol that lets separate networks join into one internet.
4. 1 January 1983: the whole ARPANET switches to TCP/IP on a single day, still called the flag day.
5. 1983 and 1984: Paul Mockapetris designs the Domain Name System, so people can type names instead of numbers.
6. March 1989: Berners-Lee's proposal at CERN. December 1990: the first web server and browser run.
7. 30 April 1993: CERN places the web software in the public domain, charging nothing, for ever. That decision is why the web is not a product.
8. 1993: the Mosaic browser from NCSA, written by Marc Andreessen and Eric Bina, puts images and text on one page and the public arrives.
9. 1998: Google is founded, and finding things becomes the main problem.

14.14.5 TECHNICAL - the engineer's version

1. Mobile generations, with the years the first commercial networks opened:

Generation	From	Key change
1G analogue	1979 to 1983	Voice only, no security
2G GSM	1991	Digital, SMS, SIM cards
3G	2001	Packet data at scale
4G LTE	2009	All-IP, no voice circuit
5G	2019	Low latency, dense cells

2. The first GSM call was made in Finland in 1991. The first SMS text message was sent in December 1992 and read "Merry Christmas".
3. 3G opened commercially with NTT DoCoMo's FOMA service in Japan in October 2001. The first commercial 4G LTE networks opened in Stockholm and Oslo in December 2009.
4. Two structural facts worth carrying forward. First, the internet has no centre and no owner; it is an agreement between networks to pass traffic. Second, the web is one application running on it, alongside email, video calls and file transfer.
5. Chapter 30 covers packets, routing, TCP, DNS, HTTP and TLS properly. Chapter 31 covers radio, cells, handover and the mobile stack.

14.14.6 WORDS - remember these

1. ARPANET — the 1969 research network — the packet-switched predecessor of the internet, funded by the US Advanced Research Projects Agency.
2. Packet — a small numbered piece of your message — a routed protocol data unit with header and payload.
3. TCP/IP — the agreement that joins networks together — the Transmission Control Protocol and Internet Protocol suite, universal from 1 January 1983.
4. DNS — the phone book turning names into numbers — the Domain Name System, designed by Paul Mockapetris in 1983 and 1984.
5. World Wide Web — linked pages on top of the internet — the hypertext system proposed by Berners-Lee at CERN in March 1989, released freely in 1993.

14.15 The last twenty years

14.15.1 PLAIN - in simple words

1. For thirty years computers got faster in the simplest way: the clock ticked faster every year.
2. Around 2004 that stopped. Pushing the clock higher made chips too hot to cool. Intel cancelled its next high-clock designs in May 2004.
3. The answer was to put several complete processors, called **cores**, on one chip. Mainstream dual-core desktop chips arrived in May 2005.
4. This shifted the burden onto software. A single-threaded program gets no faster on a four-core chip.
5. On 9 January 2007 Apple announced the **iPhone**, which shipped on 29 June 2007. In September 2008 the first Android phone was announced.
6. From 2006, Amazon began renting computing by the hour. Storage, called S3, opened on 14 March 2006, and rentable servers, called EC2, in August 2006. That is **cloud computing**.
7. Around 2011 the world began shipping more smartphones than personal computers, and never went back.
8. Graphics chips, built to draw triangles for games, turned out to be ideal for the arithmetic behind machine learning.
9. Since 2012 that has driven the largest hardware build-out in the industry's history.
10. In 2026 the fastest computer on the public TOP500 list is a Chinese machine called LineShine, at about 2.2 exaflops.

14.15.2 PLAIN - a picture in your head

1. Imagine a restaurant kitchen with one chef who cooks faster every year.
2. Eventually the chef cannot move any faster without catching fire.
3. So you hire four chefs. The kitchen can now serve four tables at once.
4. But one complicated dish still takes one chef the same time. Four chefs cannot chop one onion four times as fast.
5. That is why your computer has eight or sixteen cores and some programs still feel exactly as slow as they did in 2010.

Where this comparison breaks: chefs share one kitchen. Cores share caches and one path to memory, so adding cores can even slow things down through contention. Splitting work also costs time in coordination, which is why Amdahl's law, stated by Gene Amdahl in 1967, says the sequential part of a program sets a hard ceiling on any speed-up.

14.15.3 PLAIN - a worked example

1. Amdahl's law in numbers. Suppose 90 per cent of a program can be split across cores and 10 per cent cannot.
2. Speed-up with N cores is 1 divided by (0.1 plus 0.9 over N).

Cores	Speed-up	Efficiency
1	1.00	100%
2	1.82	91%
4	3.08	77%
16	6.40	40%
1,000	9.91	1%

3. With a tenth of the work stuck in sequence, a thousand cores buy you less than ten times the speed.
4. That table explains the whole software industry since 2005. It is also why machine learning suits GPUs so well: its core operation, multiplying large matrices, is nearly all parallel.

14.15.4 PLAIN - what is really happening inside

1. Why the clock stopped rising: a rule called Dennard scaling used to say that as transistors shrank, power per unit area stayed constant. It broke down around 2005 because leakage current stopped shrinking with size.
2. So power density rose with frequency, and the heat became unremovable with air cooling.
3. Clock speed comparison: a Pentium 4 reached 3.8 gigahertz in 2004. High-end desktop chips in 2026 boost to roughly 5 to 6 gigahertz. That is under twice as fast in twenty-two years.
4. Core counts in the same period went from 1 to 16 or 24 on a desktop, and to well over 100 on server parts.
5. Cloud computing changed who owns machines. Before 2006, needing 500 servers for one week meant buying 500 servers. After 2006 it meant a credit card and ten minutes.
6. Phones changed what a computer looks like. The processor in a modern phone uses the ARM instruction set, designed in Cambridge in 1985 for the Acorn Archimedes, and licensed rather than manufactured by its owner.
7. Apple moved its own laptops and desktops to ARM chips of its own design with the M1 in November 2020, ending nearly fifteen years of Intel Macs.

14.15.5 TECHNICAL - the engineer's version

1. Multicore timeline: IBM POWER4, the first commercial dual-core general purpose processor, shipped in 2001. Mainstream x86 dual-core parts, the AMD Athlon 64 X2 and the Intel Pentium D, both launched in May 2005.
2. GPU computing timeline: Nvidia released the CUDA software development kit on 15 February 2007 for the G80 generation, starting with the GeForce 8800.
3. The result that changed the field was AlexNet, which won the ImageNet competition in 2012 by a large margin, trained on two consumer graphics cards. The 2017 paper *Attention Is All You Need* introduced the transformer architecture behind current language models.
4. Where things stand in 2026, separated honestly:
    1. **Established fact**: Nvidia's Blackwell B100, from 2024, carries about 208 billion transistors. Apple's M3 Ultra, from 2025, carries about 184 billion. TSMC's 3-nanometre class process is in high-volume production and 2-nanometre class parts are entering it.
    2. **Established fact**: on the June 2026 TOP500 list the leading machine is LineShine at the National Supercomputing Centre in Shenzhen, China, at 2,198 petaflops, ahead of El Capitan at Lawrence Livermore at 1,809 and Frontier at Oak Ridge at 1,353.
    3. **Active research**: whether current model architectures continue to improve with more data and compute, and what the true limits of low precision arithmetic are for training.
    4. **Active research**: quantum computing. Real machines exist and real results are published, but no general commercial workload runs faster on one today.
    5. **Marketing claim**: process node names such as "3 nanometre" no longer describe any physical dimension on the chip. They are commercial labels. Compare vendors by transistor density, not by node name.
5. Process node names are the clearest example in the industry of a **marketing convention** dressed as a specification. Say so when you hear it.
6. Tools that let you observe your own machine's place in this story:

```
lscpu                # cores, threads, caches, flags
cat /proc/cpuinfo    # per-core model and speed
nvidia-smi           # GPU model, memory, utilization
sysctl -n machdep.cpu.brand_string   # macOS CPU name
```

14.15.6 WORDS - remember these

1. Core — a complete processor inside the chip — an independent execution unit with its own registers and level-1 caches.
2. Dennard scaling — smaller transistors used to mean cooler chips — the constant power-density rule that failed around 2005.
3. Amdahl's law — the slow part sets the limit — the speed-up bound imposed by the non-parallel fraction of a program, stated in 1967.
4. Cloud computing — renting computers by the hour — on-demand elastic provisioning of remote compute and storage, from AWS S3 in March 2006.
5. GPU — the graphics chip now used for mathematics — a throughput-oriented many-core processor, made general-purpose by CUDA in 2007.
6. Exaflop — a billion billion sums a second — 10 to the power 18 floating point operations per second.

14.16 One master timeline, 1801 to 2026

14.16.1 PLAIN - in simple words

1. A timeline is not a list of gadgets. It is a list of moments when something became possible that had not been possible the day before.
2. Read it once for the shape, not for the detail.
3. You will see three kinds of entry: an idea written down, a machine that worked, and a product that sold.
4. All three are needed. An idea nobody builds is philosophy. A machine nobody buys is a museum piece.
5. You will also see long gaps. Between Babbage in 1837 and the Z3 in 1941 there are 104 years in which almost nothing happened in general computing.

14.16.2 PLAIN - a picture in your head

1. Think of a relay race where the baton is dropped for a century and picked up by somebody who never met the first runner.
2. Babbage's designs were largely forgotten. Turing did not build on them. Zuse had not heard of them.
3. The same idea was found independently three or four times.
4. That is normal in this history. Ideas arrive when the parts to build them become cheap, not when someone first thinks of them.

Where this comparison breaks: it was not a single race. Several teams ran in parallel in different countries, often in secret, and the winner was often decided by who was allowed to publish.

14.16.3 PLAIN - a worked example

1. Take one row and unpack it: 1948, Manchester Baby runs.
2. The idea came from Turing in 1936 and the design from the EDVAC report in 1945, so the idea is 12 years older than the machine.
3. The enabling part was the Williams tube, a memory made from a war-surplus radar display.
4. So the row hides three things: a mathematician's paper, a wartime component, and an engineering team.
5. Every row in the table below hides a story of that shape.

14.16.4 PLAIN - what is really happening inside

1. Four forces recur throughout this timeline.
2. **Cost per operation** falls, roughly by a factor of ten every five to seven years across the whole period.
3. **Physical size** falls, from a room, to a cabinet, to a board, to a chip, to part of a chip.

4. **Who is allowed to use it** widens: governments, then corporations, then departments, then individuals, then everyone with a pocket.
5. **What it is used for** widens from arithmetic, to records, to text, to pictures, to communication, to everything.
6. If you can name those four trends you can place any unfamiliar machine in this story within a decade of its real date.

14.16.5 TECHNICAL - the engineer's version

Year	Event	Why it mattered
1801	Jacquard shows loom in Paris	Cards begin to steer machines
1804	Jacquard head patented	Program separate from machine
1822	Babbage's difference engine	Machine-made number tables
1837	Analytical Engine designed	Mill, store, cards, printer
1843	Lovelace publishes Note G	First published algorithm
1854	Boole's Laws of Thought	Logic becomes algebra
1889	Hollerith patent 395,782	Electric card counting
1890	US census tabulated	First mass data processing
1911	CTR formed by merger	The firm that becomes IBM
1924	CTR renamed IBM	Sixty years of dominance
1931	Gödel's incompleteness	Limits of proof shown
1936	Turing and Church papers	Limits of computing shown
1937	Shannon's relay thesis	Boolean algebra meets wires
1941	Zuse's Z3 runs	First program-driven machine
1942	Atanasoff-Berry Computer	Electronic binary arithmetic
1943	Colossus Mark 1 built	First big electronic machine
1944	Harvard Mark I presented	US sequence-controlled machine
1945	First Draft on EDVAC	Stored-program design set out
1946	ENIAC shown to press	Electronic speed made public
1947	Transistor at Bell Labs	End of the vacuum tube
1948	Manchester Baby runs	First stored program executed
1949	EDSAC service starts	Subroutines and a real service
1951	Ferranti Mark 1 delivered	First commercial computer
1951	UNIVAC I accepted	US commercial computing
1952	UNIVAC predicts election	Computer enters common speech
1953	IBM 701 shipped	IBM enters electronic computing
1956	IBM 350 disk drive	Random-access storage arrives
1957	Fortran released	Programming above the machine
1957	Eight leave Shockley	Fairchild and Silicon Valley
1958	Kilby's integrated circuit	Parts without wires
1959	Noyce's planar IC	Chips can be mass produced
1959	DEC PDP-1 delivered	Interactive computing begins
1964	IBM System/360 announced	One compatible product family
1964	BASIC at Dartmouth	Beginners can program
1965	Moore's law published	The industry gets a roadmap
1965	DEC PDP-8 at $18,500	Minicomputer era opens
1968	Intel founded	The chip giant begins
1968	Engelbart's demo in SF	Mouse and windows shown
1969	ARPANET first message	Networking begins
1969	Unix begun at Bell Labs	Portable operating system
1969	AMD founded	A second source for chips

Year	Event	Why it mattered
1970	DEC PDP-11 announced	Unix and C grow up on it
1971	Intel 4004 announced	A processor on one chip
1971	Silicon Valley named	The region gets its identity
1972	C language at Bell Labs	Portable systems software
1973	Xerox Alto working	Graphical personal computer
1973	Ethernet memo at PARC	Local networks
1974	Intel 8080 released	The first hobby computer CPU
1975	Altair 8800 on sale	Personal computing starts
1975	Microsoft founded	Software becomes an industry
1975	MOS 6502 at $25	Cheap CPUs for home machines
1976	Apple I on sale	Apple begins
1976	Zilog Z80 released	The dominant 8-bit CPU
1977	Apple II, PET, TRS-80	Ready-to-use home computers
1978	Intel 8086 released	The x86 line starts
1979	VisiCalc released	The first killer application
1981	IBM PC 5150 announced	The business standard
1981	Xerox Star launched	Desktop metaphor for sale
1982	Compaq Portable announced	Clean-room clones arrive
1983	ARPANET moves to TCP/IP	One internet
1983	Apple Lisa launched	GUI reaches the market
1984	Macintosh launched	GUI becomes affordable
1985	Windows 1.0 released	Microsoft enters graphics
1985	ARM design starts at Acorn	The CPU in your phone
1989	Web proposed at CERN	Linked documents
1991	Linux kernel announced	Free operating system
1993	Mosaic browser released	The public web
1993	CERN frees web software	No owner, no licence fee
1995	Windows 95 released	GUI becomes the default
1998	Google founded	Search organizes the web
2001	IBM POWER4 dual core	Multicore begins
2003	AMD Opteron ships x86-64	64-bit on commodity chips
2005	Dual-core desktop chips	Clock speed race ends
2006	AWS S3 and EC2 launch	Computing rented by the hour
2007	iPhone released	The pocket computer
2007	CUDA SDK released	GPUs become general
2008	First Android phone	Smartphones become universal
2011	Smartphones outsell PCs	The centre of gravity moves
2012	AlexNet wins ImageNet	GPUs become learning engines
2017	Transformer paper	Modern language models
2020	Apple M1 ships	ARM reaches the desktop
2022	ChatGPT released	Language models reach public
2024	Nvidia Blackwell announced	208 billion transistors
2025	Apple M3 Ultra ships	184 billion transistors
2026	LineShine tops TOP500	2.2 exaflops in Shenzhen

1. A caution on any timeline: dates in computing are slippery because announcement, first working unit, first delivery and general availability can be years apart.
2. Where they differ in this chapter, the body text says which one is meant.
3. Where a claim is contested, such as who first built a computer, the chapter says so rather than picking

a winner.

14.16.6 WORDS - remember these

1. Announcement date — when it was told to the world — the marketing date, often the earliest and least meaningful.
2. First working unit — when it actually ran — the engineering milestone used for priority claims.
3. General availability — when anyone could buy one — the commercial date, the one that changes an industry.
4. Priority claim — the argument about who was first — a disputed assertion that depends entirely on the definition chosen.

14.98 Common wrong ideas

1. Wrong: Babbage built the Analytical Engine. Right: he designed it from 1837 and built only fragments. Neither of his engines was completed in his lifetime; he died in 1871. The Science Museum in London built Difference Engine No. 2 from his drawings, finishing the calculating section in 1991.
2. Wrong: the first computer bug was a moth. Right: the moth found in a Harvard Mark II relay on 9 September 1947 was taped into a logbook with the note "First actual case of bug being found". The joke only works because engineers already said "bug" for a fault. Thomas Edison used the word that way in 1878.
3. Wrong: Grace Hopper coined the word bug. Right: she popularized the moth story. She did lead the team that built the first compiler, the A-0 system in 1952, and shaped COBOL from 1959, which is achievement enough.
4. Wrong: Bill Gates wrote MS-DOS. Right: Tim Paterson at Seattle Computer Products wrote 86-DOS in 1980. Microsoft licensed it for $25,000 in December 1980 and bought it outright for $50,000 in July 1981. Gates and Allen did write Altair BASIC themselves in 1975.
5. Wrong: Apple invented the graphical interface. Right: Douglas Engelbart demonstrated the core ideas on 9 December 1968 and Xerox PARC built them into the Alto in 1973 and sold the Star in 1981. Apple licensed, refined and priced it for ordinary buyers with the Lisa in 1983 and the Macintosh in 1984.
6. Wrong: Apple stole the interface from Xerox. Right: the December 1979 visit was arranged, and Xerox received the right to buy Apple shares in exchange. Xerox later sued Apple over the interface and lost, largely on timing.
7. Wrong: Al Gore said he invented the internet. Right: on 9 March 1999, on CNN, he said "I took the initiative in creating the Internet", meaning legislation. Vinton Cerf and Robert Kahn, who designed TCP/IP, publicly defended his record. His 1991 High Performance Computing Act helped fund the Mosaic browser.
8. Wrong: ENIAC was the first computer. Right: that depends entirely on the definition. The Z3 of 1941 was program-driven, the Atanasoff-Berry machine of 1942 was electronic, Colossus of 1943 was electronic and digital, and the Manchester Baby of 1948 was the first to run a stored program. ENIAC was the first general-purpose electronic machine put to broad use.
9. Wrong: computers have always used binary. Right: ENIAC, the Harvard Mark I, the IBM 650 and many other early machines were decimal. Binary won because it halves the components needed per digit and matches switching circuits.
10. Wrong: Silicon Valley is named after a company. Right: the journalist Don Hoefler used the phrase in Electronic News on 11 January 1971. Silicon is the element the chips are made from, and the valley is the Santa Clara Valley in California.

14.99 Chapter summary in 20 lines

1. Counting aids came first: tally sticks, the abacus, and the geared Antikythera mechanism of roughly 100 BCE, found in a wreck in 1901.
2. Napier's bones of 1617, the slide rule of about 1622 and Leibniz's 1703 paper on binary set the mathematical stage.
3. Schickard in 1623, Pascal in 1642 and Leibniz in 1673 built calculators that worked but could not be reprogrammed.
4. Jacquard's loom head, patented in 1804, separated the machine from its instructions. That is the founding idea of the whole field.
5. Babbage designed the Difference Engine from 1822 and the Analytical Engine from 1837, with mill, store, card reader and printer. Neither was finished.
6. Ada Lovelace's 1843 notes contain the first published algorithm and the first clear statement that a computer could handle symbols, not just quantities.
7. Hollerith's punched card machines processed the 1890 United States census. His company merged in 1911 and was renamed IBM in 1924.
8. Gödel showed in 1931 that proof has limits. Church and Turing showed in 1936 that computing has limits too, and Turing's model described every later machine.
9. Turing complete means able to simulate a universal Turing machine, and the bar is so low that card games and spreadsheet formulas clear it by accident.
10. The war produced the Bombe in 1940, Zuse's Z3 in 1941, the Atanasoff-Berry machine in 1942, Colossus in 1943 and 1944, and the Harvard Mark I in 1944.
11. ENIAC, shown in 1946, was electronic and general-purpose but was programmed by rewiring, work done by six women who went uncredited for forty years.
12. The 1945 First Draft of a Report on the EDVAC set out the stored-program design that everything since has followed, under a disputed name.
13. The Manchester Baby ran the first stored program on 21 June 1948, EDSAC gave the first service in 1949, and the Ferranti Mark 1 was sold in 1951.
14. UNIVAC I predicted the 1952 election correctly and was overruled live on air, which is how the public learned what a computer was.
15. The transistor of 1947 and IBM's System/360 of 1964 turned computing into an industry with compatible product families and portable software.
16. Kilby in 1958 and Noyce in 1959 put whole circuits on one chip, and Hoerni's planar process made them manufacturable.
17. The Intel 4004 of 1971 put a processor on one chip; the $25 MOS 6502 of 1975 made home computers affordable.
18. The Altair of 1975, Apple II of 1977, VisiCalc of 1979 and the IBM PC of 1981 with its published design created the personal computer industry.
19. Engelbart in 1968 and Xerox PARC in 1973 invented the graphical interface; Apple sold it in 1984 and Windows 95 made it universal in 1995.
20. Since 2005 progress has come from many cores, rented cloud machines, phones and graphics chips repurposed for machine learning, with the June 2026 TOP500 led by LineShine at about 2.2 exaflops.

ABOUT THE AUTHOR

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

Shikhar Singh is the Founder & Chairman of KedByte Technologies Private Limited.

He wrote this book to solve a problem he kept meeting: capable people concluding they were not technical, when in truth nobody had ever shown them the bottom of the stack. His conviction is that computing has no genuinely hard idea in it, only a very long chain of simple ones, and that anybody who is walked along that chain in order can hold the whole of it in their head.

That conviction shapes the method used here. Explain it plainly, with a comparison and a worked example, so that it lands. Then explain it again in the exact professional vocabulary, with the real units and the real figures, so that it is usable. Say clearly where the plain version stops being true. Never let a reader carry away a confident misunderstanding.

Under his direction, KedByte Technologies Private Limited treats technical education as infrastructure rather than as a product feature: knowledge that should be transferable, verifiable, and free of the vocabulary barriers that keep capable people out of the field.

ABOUT THE PUBLISHER

KedByte Technologies Private Limited

KedByte Technologies Private Limited publishes technical work under the KedByte Press imprint.

The editorial standard applied to this volume is simple to state and difficult to meet. Every explanation must work twice: once for a reader who has never opened a terminal, and once for an engineer who needs the exact figure. Every claim that can be checked must be checked. Every figure that will go stale must carry a date. Where the field genuinely disagrees, both positions are given rather than one being quietly chosen.

Where this book uses real evidence, it uses it unaltered. The network trace in Part F, the failed pushes in Part G and the error messages quoted throughout are reproduced exactly as they were observed. Diagnosis is taught against ambiguous real data, because that is the only kind there is.

KEDBYTE

TECHNOLOGIES PRIVATE LIMITED

COLOPHON

This first edition of *How Computers Work* runs to nine parts and fifty-three chapters, and was set entirely from plain text.

The body text is set in TeX Gyre Pagella, a digital revival of Hermann Zapf's Palatino, chosen for long-form reading. Headings, tables and the KedByte identity are set in Poppins. Code, command output and diagrams are set in DejaVu Sans Mono, and every diagram in this book is drawn in plain ASCII so that it survives being copied anywhere.

The book was composed with pandoc and typeset with XeLaTeX on A4 stock. The single accent colour used on rules, chapter numbers and section headings is KedByte navy.

Shikhar Singh

Founder & Chairman, KedByte Technologies Private Limited

KB

First Edition • KedByte Press