

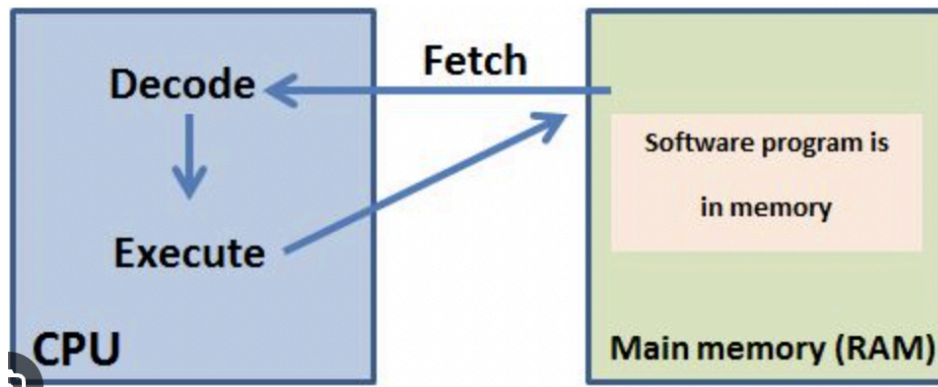
Computer Architecture

So far we've discussed the inner workings of computers from two different vantage points: memory and networking. But this isn't the full picture of how computers actually work. In fact, neither of these components holds the top spot for the most important piece of the puzzle. We could have all the memory and connectivity in the world, and none of it would matter if we couldn't give a program any instructions on how to use it. For that, we have to look at the most well known part of a computer, the CPU, or Central Processing Unit.

The CPU is the primary processor of a computer, responsible for executing all instructions it receives from a program at any given moment. It contains a core, which, as we discussed in the memory section, is where the registers and cache live. But the core does more than just hold data. Its main function is to execute processes on that data, fetching each instruction from memory, decoding what that instruction is asking it to do, and then executing it. This fetch, decode, execute cycle is the heartbeat of all computation.

The core accomplishes this through a network of transistors arranged into logic gates, which perform arithmetic, comparisons, and data movement entirely in binary. Depending on the type of core, it may support anywhere from around 50 to several thousand distinct instructions. This is called the instruction set architecture. Some of these instructions you've already encountered in your own code, like AND, OR, ADD, and subtract, since these are exactly how variables get combined and compared. One instruction we haven't touched on yet is called jump, and it's one of the most powerful of all, since it's what makes logic and loops possible in the first place, by adjusting something called the program counter, a piece of data held alongside the registers that

tracks which instruction in a program should run next.



Now, rather than build our own custom instruction set from scratch, let's talk about how you can actually observe this layer of your computer directly, using tools available right in your terminal. The core has a certain number of individual processing units, called cores, each capable of executing instructions independently. On most systems, you can check how many cores your computer has using a terminal command. On Mac, **sysctl -n hw.ncpu** will return the number of cores available. On Linux, the equivalent is **nproc**, or you can run **lscpu** for a more detailed breakdown, including core count, model name, and architecture.

```
[/Users/k2 %sysctl -n hw.ncpu
10
```

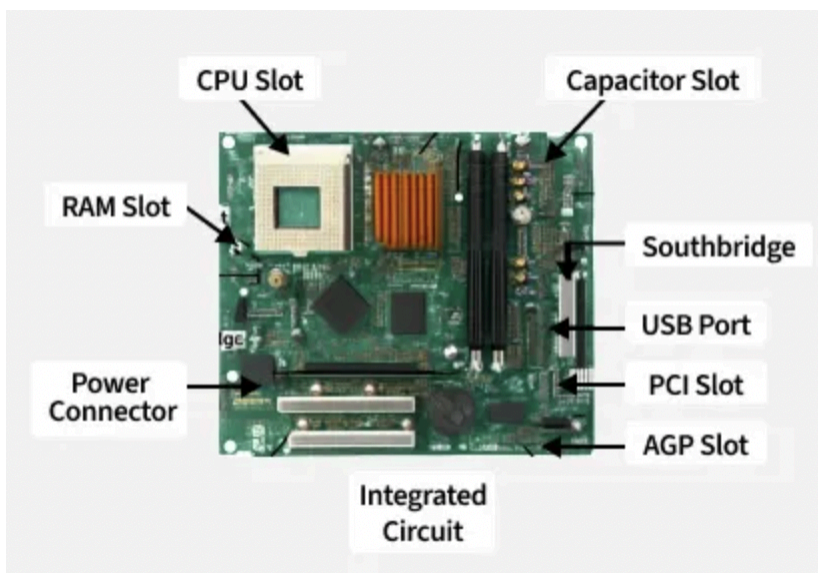
Beyond core count, it's also useful to see how hard those cores are working in real time, which is where CPU bandwidth, or usage, comes in. Tools like **top** or **htop**, which we introduced back in our very first session, give you a live, continuously updating view of CPU usage across all your cores, along with memory consumption and running processes. On Mac, you can also check current CPU load with a command like **top -l 1**, which takes a single snapshot rather than

continuously refreshing.

```
[/Users/k2 % top -l 1
Processes: 502 total, 5 running, 497 sleeping, 4433 threads
2026/09/19 16:47:31
Load Avg: 5.98, 4.95, 4.22
CPU usage: 14.2% user, 18.31% sys, 67.66% idle
```

Understanding these numbers matters more than it might seem at first. A high core count means a computer can genuinely run many tasks in true parallel, rather than just switching rapidly between them. High CPU usage on a specific core can point to a program that's poorly optimized, stuck in an inefficient loop, or simply doing legitimately heavy work. These same commands you're practicing on your own machine right now are exactly the tools you'll rely on later when monitoring the health of a real server or data center.

Moving past the core, let's go over some computer architecture components we already know, like main memory (RAM), which sits on physical sticks outside the CPU, along with hard disk drives and solid state drives, as well as PCIe. All of these critical pieces connect to what's called the motherboard, the main circuit board that keeps every component connected and allows communication to flow between them as needed.



There are a couple of concepts we haven't covered yet. NVMe is a protocol for solid state drives that allows them to communicate through the faster PCIe channels, rather than the slower SATA interface. There's also the GPU, or Graphics Processing Unit. A GPU differs architecturally from a CPU, since its purpose is delivering graphics rather than managing the full range of a computer's instructions and processes. Because of this, it pulls back on complex instruction scheduling in favor of many more cores, achieving higher bandwidth and performance for raw, repetitive calculations. This same design decision, originally made for rendering graphics, is exactly what makes GPUs the processing unit of choice for AI today, given AI's massive compute demand and relatively simple process management needs.

Finally, all of these components running at such high speeds inevitably produce heat as a byproduct of the energy they consume. This is why every computer also contains a fan, to prevent overheating and protect its internal components from damage.