Cloud Infrastructure

Cache Memory and the Need for Memory Hierarchy

1. Why do we need memory?

Imagine the CPU as the brain of the computer. It can perform calculations extremely fast, but it cannot hold all the data it needs inside itself. That is why we have **memory**.

- Programs and data are stored in memory.
- The CPU fetches instructions, decodes them, executes them, and stores results back.
- Without memory, the CPU would have nothing to work on.

This model is called the **Von Neumann Architecture**. It connects Input → CPU (Control Unit + ALU) → Memory → Output.

2. How programs actually run

1. Programs are stored permanently on **secondary memory** (like hard disks or SSDs).
2. When you run a program, the **Operating System** loads it into **RAM (main memory)**.
3. The CPU then executes instructions from RAM in a loop called the **machine cycle**:
    - **Fetch** an instruction from memory.
    - **Decode** it into commands.
    - **Execute** it using the ALU.
    - **Store** results back into memory.

So, the CPU constantly talks to memory while running a program.

3. The speed problem

CPU has become much, much faster than memory over the years.

- If the CPU had to go to main memory for every instruction and every piece of data, programs would run **20–30 times slower**.
- The gap is growing because CPU speeds improve by ~50% per year, while memory speeds only improve by ~9%.
- This is called the **memory wall** problem — the CPU is hungry for data, but memory cannot keep up.

4. The solution: Memory hierarchy

To fix this speed gap, computers use a **hierarchy of memory**.

- Keep **small amounts of active data** in very fast but expensive memory (registers, cache).
- Keep **all data** in slower, cheaper memory (RAM, disk).
- The trick is to move data up and down this hierarchy so that the CPU always sees fast access.

5. Memory hierarchy levels

- **Registers**: Inside the CPU, fastest, very small.
- **L1 Cache**: Very small (hundreds of KB), extremely fast.
- **L2 Cache**: Larger (a few MB), a bit slower compared to L1 cache.
- **L3 Cache**: Even larger (tens of MB), slower than L2 cache but still faster than RAM.
- **Main Memory (RAM)**: Much larger (GBs), but much slower than caches.
- **Secondary Storage**: Hard drives or SSDs, enormous size, but thousands of times slower.

The rule of thumb: **the faster the memory, the smaller and more expensive it is.**

6. Characteristics of memory systems

When we describe memory, we look at:

- **Location**: Internal (registers, cache, RAM) or External (disk, optical).
- **Capacity**: How much data it can hold (bytes/words).
- **Unit of transfer**: The chunk of data transferred at a time (word or block).
- **Access method**:
  - *Sequential* (tape-like, must go in order).
  - *Direct* (block-level, like hard drives).
  - *Random* (instant access, like RAM).
  - *Associative* (find data by content, used in caches).
- **Performance**: measured by Access time, cycle time, transfer rate.
- **Physical type**: Semiconductor, magnetic, optical.
- **Volatility**: Volatile (RAM loses data when power is off) or non-volatile (disk, flash).

7. Methods of accessing memory

- **Sequential access**: Data is read in order, like a tape. Slow and linear.
- **Direct access**: Blocks are located by physical addresses (like hard drives).
- **Random access**: Any location can be accessed instantly and directly (like RAM).
- **Associative access**: Data is found by its content, not by its address (used in caches for fast lookups).

Recap

- CPU is very fast, memory is slower → creates a bottleneck.
- Solution: **layer memory in a hierarchy**, with small, fast caches near the CPU and large, slow storage farther away.
- Cache memory is most important of this hierarchy — it keeps the CPU “fed” with the most frequently used instructions and data, preventing the slowdown.

Capacity and Performance of Memory

When we study memory, two major aspects stand out: **capacity**, which tells us how much information the memory can hold, and **performance**, which tells us how quickly the memory can be accessed. Performance is usually described in terms of three parameters: **access time, memory cycle time, and transfer rate**.

Access Time (Latency)

Access time, often referred to as latency, is the delay before data becomes available. For **random-access memory (RAM)**, it is the time required to complete a read or write operation. For **non-random-access memory**, such as magnetic disks or tapes, it represents the time taken to position the read–write mechanism at the desired location. In short, it is the waiting time before the system can actually begin using the data.

Memory Cycle Time

Memory cycle time includes the access time plus any additional delay before a new request can begin. This extra time may be required for two reasons:

- **Signal stabilisation**: waiting for transients on signal lines to die out.
- **Data regeneration**: restoring data if it is destructively read, as in the case of DRAM.

Cycle time therefore reflects how long the memory unit needs before it can be accessed again. Unlike access time, which is processor-focused, cycle time is more closely tied to the timing of the **system bus**.

Transfer Rate

Transfer rate measures the speed at which data moves once the memory has been accessed. It describes how quickly information can flow into or out of the memory unit. For random-access memory, the transfer rate is essentially the inverse of cycle time:

- Faster cycle time → higher transfer rate.
- Slower cycle time → lower transfer rate.

This makes transfer rate a direct measure of how much useful data throughput the memory can deliver once the access overhead has been paid.

Memory

Memory exists in different forms, each with its own technology and characteristics. The most common types include **semiconductor memory**, **magnetic surface memory**, **optical memory**, and **magneto-optical memory**. These technologies form the foundation of modern storage systems.

Several physical characteristics of memory are especially important. One of these is **volatility**. Volatile memory loses its information once power is switched off or may naturally decay over time; RAM is the most common example.

In contrast, **non-volatile memory** retains data even when electrical power is removed. Information stored here remains stable until it is deliberately changed, and no electrical power is required to preserve it. Examples include flash drives, hard disks, and ROM.

Magnetic-surface memories, such as hard disks, are inherently non-volatile, making them suitable for long-term storage. **Semiconductor memory**, on the other hand, may be volatile (such as DRAM and SRAM) or non-volatile (such as flash memory).

Another important property is **erasability**. Some memory types are **non-erasable**, meaning the data cannot be altered once written, except by physically destroying the storage medium. A well-known example is **Read-Only Memory (ROM)**, which is a type of non-erasable semiconductor memory.

Finally, for **random-access memory**, the way memory is organised is a key design factor. Organisation refers to how bits are physically arranged into words, which determines how efficiently the processor can read or write information.

Memory Hierarchy

When designing computer memory, three questions always arise: **how much memory do we need, how fast should it be, and how expensive can it be?** These three factors — capacity, speed, and cost — are always in tension.

- Faster memory has a **higher cost per bit**.
- Greater capacity usually means **lower cost per bit**, but also **slower access**.
- No single memory type can satisfy all needs, so computers combine different kinds into a **hierarchy**.

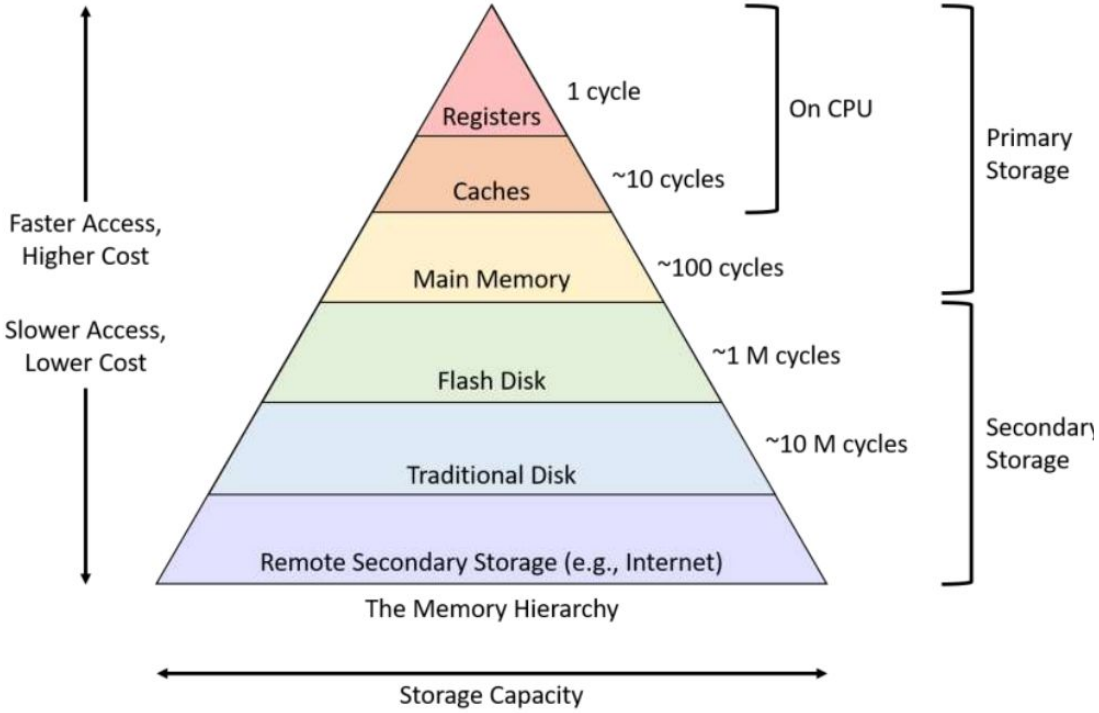
The memory hierarchy balances speed, size, and cost by using multiple levels of storage, ranging from the fastest and smallest (CPU registers) to the slowest but largest (remote storage).

Structure of the Hierarchy

At the very top are **registers**, which are built directly into the CPU and can be accessed in a single cycle. Next come the **caches** (L1, L2, L3), which store recently used data to speed up processing. Below the caches lies **main memory (RAM)**, which has much greater capacity but is significantly slower. Moving further down, we have **secondary storage**: flash disks and traditional hard drives. At the bottom are **remote storage systems**, such as cloud or internet-based storage, which offer virtually unlimited capacity but the slowest speed.

The key idea is simple:

- **Upper levels (registers, cache)** → very fast but very small and expensive.
- **Lower levels (disks, remote storage)** → very large and cheap but much slower.


Faster Access,
Higher Cost
Slower Access,
Lower Cost
Registers
1 cycle
Caches
~10 cycles
Main Memory
~100 cycles
Flash Disk
~1 M cycles
Traditional Disk
~10 M cycles
Remote Secondary Storage (e.g., Internet)
On CPU
Primary
Storage
Secondary
Storage
The Memory Hierarchy
Storage Capacity

Speed at Different Levels

Different cache levels have distinct performance characteristics:

- **L1 Cache**: The fastest, with access times around **1–2 nanoseconds**. It is closest to the CPU core and stores the most frequently used data.
- **L2 Cache**: Slower than L1 (about **3–10 nanoseconds**) but larger. It stores additional data that L1 cannot hold and may be dedicated per core or shared across cores.
- **L3 Cache**: Slower again (around **15–30 nanoseconds**) but much larger. It is shared across all CPU cores and helps balance performance.
- **RAM (DRAM)**: Much slower than cache, with latency of **50–100 nanoseconds**, but provides the main working memory for the system.

The pattern is clear: **closer to the CPU = faster and smaller**, while **farther away = slower but larger**.

Cost Across the Hierarchy

The cost of memory mirrors this pattern:

- **Registers**: Extremely expensive, costing thousands to tens of thousands of dollars per MB.
- **Cache**: Still very expensive, hundreds to thousands per MB.
- **RAM**: Relatively affordable, a few dollars per MB.
- **SSDs**: Less than a dollar to a few dollars per MB.
- **Hard Drives (HDDs)**: Cheapest of all, costing only pennies per MB.

Thus, the hierarchy is not just about speed but also about making storage economically feasible.

Cache Levels in Detail

Caches form the most important middle layer of the hierarchy, bridging the huge speed gap between CPU registers and main memory. They are organised into **levels**:

- **L1 Cache**: Smallest but fastest, tightly coupled with each CPU core.
- **L2 Cache**: Larger, slower than L1, can be per-core or shared.
- **L3 Cache**: Even larger, shared across all cores, with higher latency.

As we go from L1 → L2 → L3:

- **Bandwidth decreases** (less data per second).
- **Latency and capacity increase** (more data stored, but slower to reach).

Modern processors use multiple caches to ensure that the CPU spends less time waiting for data and more time computing.

Disk Cache

Because disk access is relatively slow, systems use a disk cache to speed things up. A portion of main memory is set aside as a fast buffer that temporarily stores data being read from or written to the disk. This allows the system to combine many small, time-consuming transfers into fewer, larger ones. Later, when the same data is needed again, it can often be retrieved directly from memory instead of repeating the slow disk access. In effect, the cache acts as a shortcut, reducing delays and improving overall performance.

Cache and Main Memory Structure

The organisation of cache and main memory is based on **blocks**.

- Main memory is divided into blocks, each containing several words.
- Cache is also divided into lines (or slots), each capable of storing a block from main memory.
  Each cache line has a **tag** that identifies which block of main memory it currently holds. This structure allows the CPU to quickly check whether a requested word is present in the cache or must be fetched from main memory.

How a Cache Read Works

The cache read operation follows a simple sequence:

1. The CPU provides an address (RA = requested address).
2. The system checks whether the block containing RA is already in the cache.
   - **If yes (cache hit):** the word is fetched directly and delivered to the CPU.
   - **If no (cache miss):** the block is read from main memory. It is then loaded into a cache line, and the requested word is delivered to the CPU.

Cache in the System

In a typical system, the processor connects to the cache through **address, data, and control lines**. The cache itself connects to the system bus, which gives it access to main memory. Buffers (address and data buffers) help manage the flow of information between these components.