# Contents

## Background

Operating Systems (OS) are responsible for at least the following capabilities:

1. Process Management
2. Memory Management
3. I/O Management
4. File System

From a programmer's point of view, originally, the system was the CPU used to execute instructions plus the memory used to hold instructions and data.

Now we have computers which:

- run multiple programs at the same time, and each needs its own memory space.
- switch the CPU quickly between programs to give the illusion that they are running at the same time.
- hold documents for several users, with the expectation that each user can protect their own files.
- allow multiple network connections simultaneously, where each connection may be dealing with sensitive data.

The OS memory management is responsible for transferring programs into and out of memory, allocating free space between programs, keeps track of each and every memory location, regardless of either it is allocated to some process or it is free (Remember a program must be brought into memory and placed within a process for it to be executed). It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

The OS manages how main memory is used. It decides:

- how memory is shared between processes
- what happens when there is not enough main memory to get the job done

If every program had full access to the CPU, main memory and the peripheral devices, all concepts of separation of programs and the data in memory, on disk etc. would not exist.

A program could look at all memory locations, including that of other programs, as well as read all the data on all of the attached disks, and read all the data being sent across the network.

> To prevent this, we need the CPU to have at least two privilege levels: user mode and kernel mode.

A user-mode program does need to cross into kernel mode.

- For example, when a program wants to read from disk, or get a line of text from the keyboard, it cannot do this in user mode.
- The only software which can is the operating system.
- The program needs to ask the operating system: "please access the hardware on my behalf and get me some input".

We need a mechanism, a ***system call***, whereby:

- a user-mode program can switch into kernel mode,
- but have no control over the instructions which will be performed in kernel mode.

Any software running in kernel mode has full access to everything. Therefore, we have to trust the system software which is the OS, or more precisely, the part of the OS which run in kernel mode, known unsurprisingly as the **kernel**.

We also need to ensure that, once running, the machine code which is the OS cannot be modified, or new code inserted into the OS which can be made to run in kernel mode.

Different processes running at the same time ***must not*** interfere with one another. This means they have to use different parts of the computer's memory.

The OS handles the transfer of data between processes by setting aside areas in memory where data values can be shared.

## Relocation as a Requirement of Memory Management

In a multiprogramming system, the main memory is generally shared among a number of processes, and generally, it's not possible for the programmer to know in advance which other programs will be resident in main memory at the time of execution of their program. Just as processes share the CPU, they also share physical memory

Computers need to be able to swap active processes in and out of main memory to maximise CPU utilisation by providing a large pool of READY processes to execute. When this process is swapped back into main memory, it does not need to be in the same memory region as it came from, therefore the process may be ***relocated*** to a different area of memory. Thus, ***swapping*** in and out of memory will occur and this can raise some technical concerns relating to addressing.

This requirement increases the difficulty of implementing protection.

## Protection as a Requirement of Memory Management

Each process should be protected against unwanted interference by other processes, whether accidental or intentional. Programs in other processes should not be able to reference memory locations in a process for read/writing without permission taking addressability and protection into account.

Because processes are swapped in/out between memory and secondary storage and stored in dynamic locations, all memory references generated by a process must be checked at run time to ensure that they refer only to the memory space allocated to that process. This is a responsibility of the processor hardware at the time of execution of an instruction making the reference (to ensure no memory reference violations) and not the OS, because the OS cannot anticipate all of the memory references that a program will make.

## Sharing as a Requirement of Memory Management

The above protection mechanism must be flexible enough to allow several processes to access the same portion of main memory i.e. if a no. of processes are executing the same program, each process should be allowed to access the same copy of the program rather than have its own separate copy.

Processes may therefore need to be allowed controlled access to shared areas of memory without compromising protection.

## Logical Organisation as a Requirement of Memory Management

Main memory and secondary memory in a computer system is organised as a linear, 1-dimensional address space (a sequence of bytes or words).

Most programs are organised into modules, some being read-only, some being execute-only, and some with data that can be modified.

*Segmentation* which is one of the memory management techniques explored here.

## Physical Organisation as a Requirement of Memory Management

With computer memory organised into at least two levels: main memory and secondary memory, the organisation of the flow of information between these levels of memory is a major system concern and the task of moving information between the two levels of memory is a system responsibility, the essence of memory management.

## Processes Requirements for Every Process

Every **process** needs some memory to store its variables and code. But if there is more than one process in memory at any one time, then the OS must manage memory (as a resource) to prevent processes from reading/damaging each other's memory, and to ensure that each process has enough memory (not too much, not too little), bearing in mind that a process' memory requirements may vary with time, and users schedule processes unpredictably.

That memory is hardware and has an address decoder.

Before we continue, we are concerned in this set of notes with the OS as a memory manager so what about memory…

## Memory

In computing, memory refers to the computer hardware devices used to store information for immediate use in a computer. Conventional Von-Neumann computing architectures consist of a pure computational part (central processor unit - CPU) and a memory part in which the computing programs and the input/output data of the calculations are stored

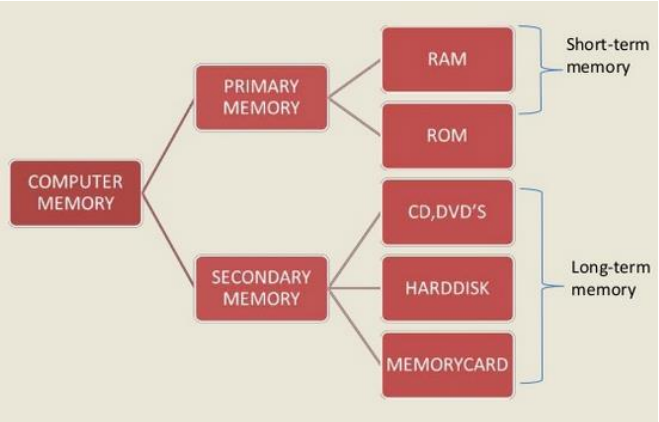
As applications become increasingly memory-hungry, a computer without enough RAM will quickly hit its limit. Because no matter how fast the processor or hard disk might be, insufficient RAM will slow down the whole system.

A fast office computer should have at least 8GB of RAM, while video editing and games require 16GB or more.

For mobiles, for example, Android uses approximately 830MB to operate while iOS uses approximately 200MB. That is why most Android phones have more RAM than the iPhones. If you decide to use Android, choose a phone with at least 3GB of RAM. You should also be okay with an iPhone of about 2GB of RAM, iPhone7 and iPhone X have 3GB RAM.

> Currently, Google's Pixel 3 is reported widely to be suffering with a memory management issue, which is Pixel 3's apparent inability to shuffle more than a few apps at a time: taking a photo is apparently enough to kill Spotify if it's playing music in the background. Cycling more than 3-4 apps can force some out of memory. Maybe 4GB of RAM wasn't enough for a flagship phone in 2018 after all, Google?

There are two types of memories:



### 1. Long-term memory

E.g. Hard drive, CD or any external storage device. It is a system for permanently storing, managing, and retrieving information for later use

### 2. Short-term memory

#### o *Random Access Memory (RAM)*

RAM is your system's short-term memory. Whenever your computer performs calculations, it temporarily stores the data in the RAM until it is needed. This is also known as **working memory** - it stores input data, intermediate results, programs, and other information temporarily. It can be read and/or written. It is volatile.

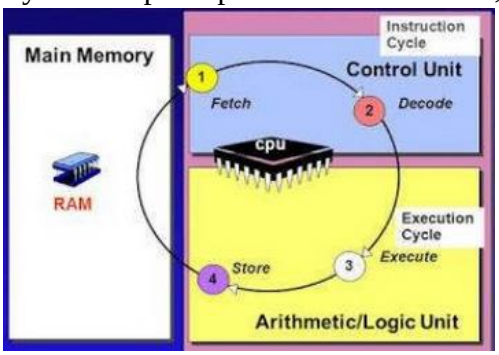



*Figure 1 - RAM in the Fetch-Execute cycle*

There are two types of RAM:

1. **Static RAM (SRAM)**

- Semi conductor memory
- Use flip-flop to store each bit of memory so does not need to be periodically refreshed
- Faster and consumes low power
- Expensive and have complex structure (6 transistors) so not use in high capacity applications

2. **Dynamic Random Access (DRAM)**

- Store each bit of memory in capacitor in an integrated circuit
- Real capacitors leak charge so capacitors need to be refreshed periodically
- Slower and cheaper than SRAM.
- Simple structure (1 transistor and 1 capacitor per bit) so it has very high density

There are then variations of RAM such as *LPDDR3 SDRAM*, *DDR4* and so on. Familiarising yourself with these will be useful when buying a new machine for yourself!

- *Read-Only-Memory (ROM)*

RAM requires a power source to retain its information, whereas ROM can retain its information when its power source is removed.

ROM contains the programming that allows your computer to be "booted up" or regenerated each time you turn it on.

There are various types of ROM:

**Programmable ROM (PROM)**

- Empty of data when the chip is manufactured, can be programmed by the user. Once programmed the data cannot be erased.

**Erasable PROM (EPROM)**

- Like PROM only the chip can be removed from the computer and the program erased and another stored in its place using ultraviolet light.

**Electrically EPROM (EEPROM)**

- Like EPROM but electricity is used to erase and reprogram

Such complex systems have a memory hierarchy comprising different semiconductor memory types.

Dense, slow and non-volatile storage memory with limited endurance is combined with fast, volatile, power and area consuming SRAM/DRAM working memory (located close to the CPU) in order to ensure both rapid accessibility and data non-volatility.

However, this sort of design hierarchy requires complex control.

> Start-up (booting) and shut- down procedures usually take a long time and waste a significant amount of power since they imply extensive data traffic (from storage memories to working memories and vice- versa).

In recent years, both the limited clock frequency of the processor and the emergence of multi-core architectures led to a significant increase in working memory capacity. As a result, the performance and the power of the computing system became determined by working, SRAM-based, memory.

## Cache Memory

Cache memory, also called CPU memory, is high-speed SRAM that a computer microprocessor can access more quickly than it can access regular RAM as it sits between main memory and the CPU registers.

*Recall: every program must be brought into memory and placed within a process for it to be run and there will be a collection of processes on disk waiting to be brought into memory to be run.*

- Faster and more expensive than RAM
- It improves the computer's performance
- Processor can use it to store frequently accessed data and program instructions

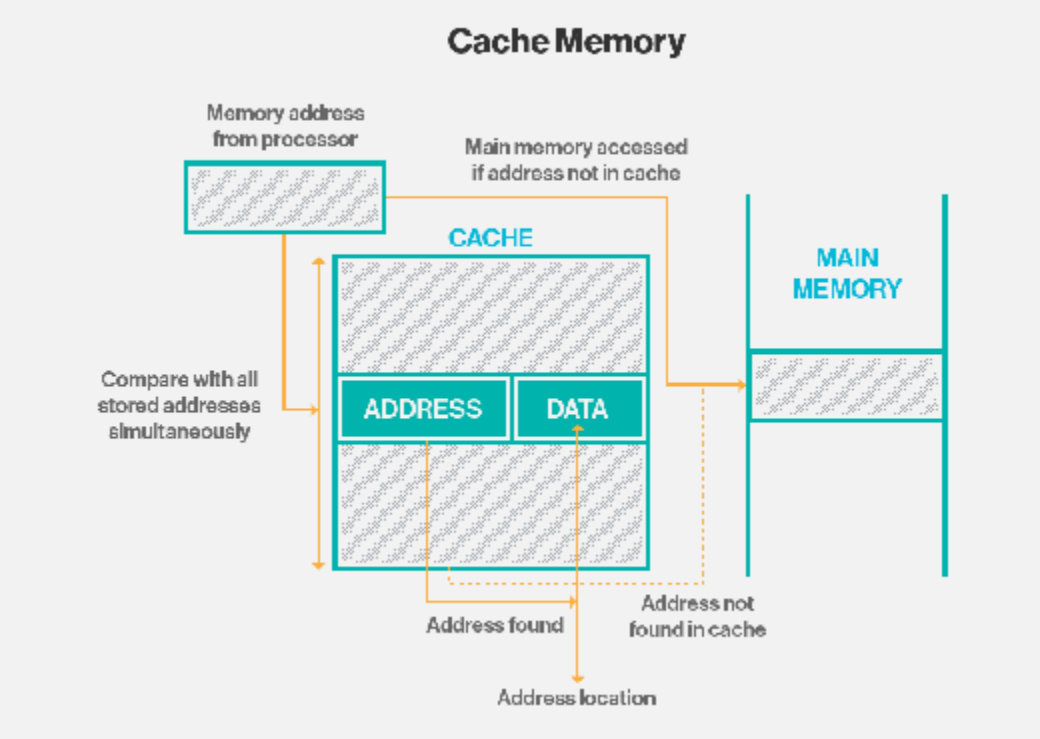


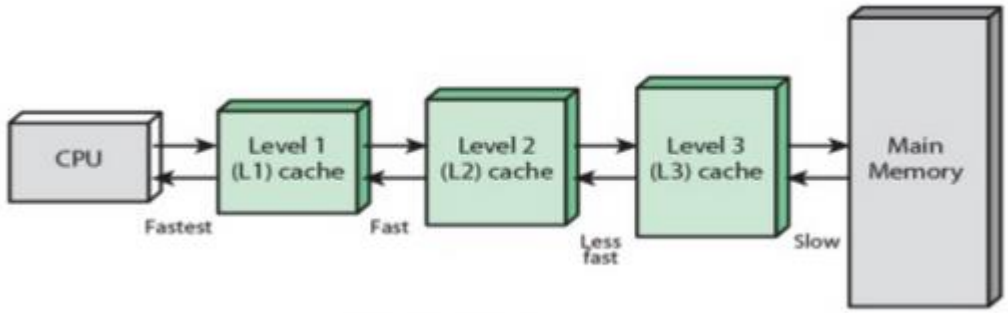

*Figure 2 - Cache Hierarchy*

There are three types of cache memory found in a CPU:

1. L1 : primary cache is embedded on the processor as CPU cache
2. L2 : secondary cache may be embedded on the CPU, or it can be on a separate chip or coprocessor and have a high-speed alternative system bus connecting the cache and CPU

3. L3: developed to improve the performance of L1 and L2. With multicore processors, each core can have dedicated L1 and L2 cache, but they can share an L3 cache. If an L3 cache references an instruction, it is usually elevated to a higher level of cache.

If active segments of a program are placed in a fast cache memory, then the execution time can be reduced.

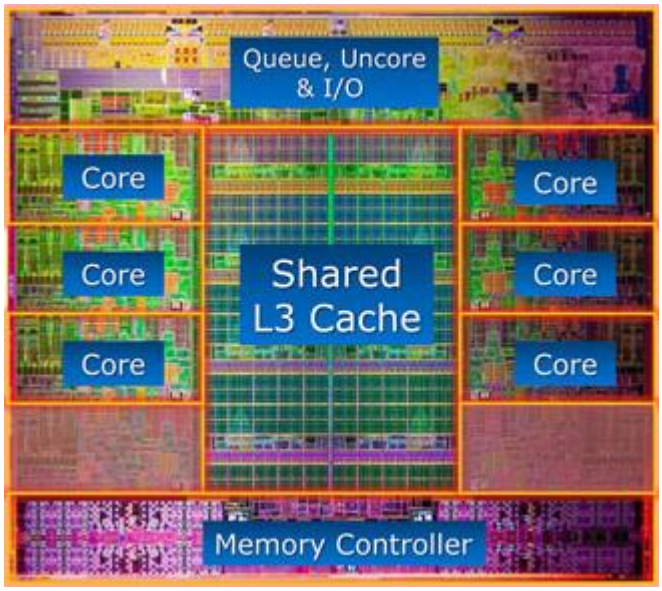



Cache is usually located on the computer processor chip and not on the motherboard and is based on the property of computer programs known as "***locality of reference***"

> **Temporal locality of reference:** Whenever an instruction or data is needed for the first time, it should be brought into a cache. It will hopefully be used again repeatedly.
>
> **Spatial locality of reference:** Instead of fetching just one item from the main memory to the cache at a time, several items that have addresses adjacent to the item being fetched may be useful.

The address issued by the processor may correspond to that of an element that exists currently in the cache (***cache hit***); otherwise, it may correspond to an element that is currently residing in the main memory. Therefore, address translation has to be made in order to determine the location of the requested element. This is one of the functions performed by the Memory Decoder, also known as the Memory Management Unit, MMU.

**Intel's use of Cache Memory**

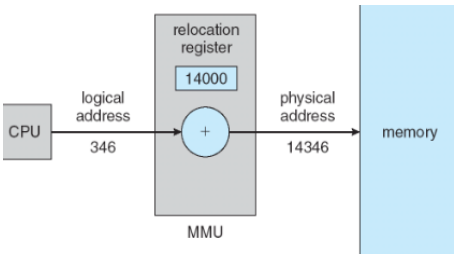



Intel's Core i7 processors have maintained an **8MB** L3 since the debut of Nehalem (codename for an Intel processor microarchitecture) in 2008 (roughly 2MB of L3 for every CPU core) but the highest-end parts are typically pegged at 2.5MB of cache per CPU core

*Figure 3 - Intel Core i7-3960X processor with shared L3 Cache*

## Memory-Management Unit (MMU)

In brief, the **Memory-Management Unit (**MMU) is a hardware device that maps virtual to physical address.




For example, in a partitioned memory allocation scheme, the MMU adds a value in the relocation register (starting address of the segment allocated to the program) to every address (offset) generated by a user process at the time it is sent to memory.

OSs see the virtual address of the user programs as well as the physical memory address, so that certain MMU registers are loaded by proper values, during the program execution.

Early memory cache controllers used a write-through cache architecture, where data written into cache was also immediately updated in RAM. This approached minimised data loss, but also slowed operations. With later 486-based PCs, the write-back cache architecture was

developed, where RAM isn't updated immediately. Instead, data is stored on cache and RAM is updated only at specific intervals or under certain circumstances where data is missing or old.

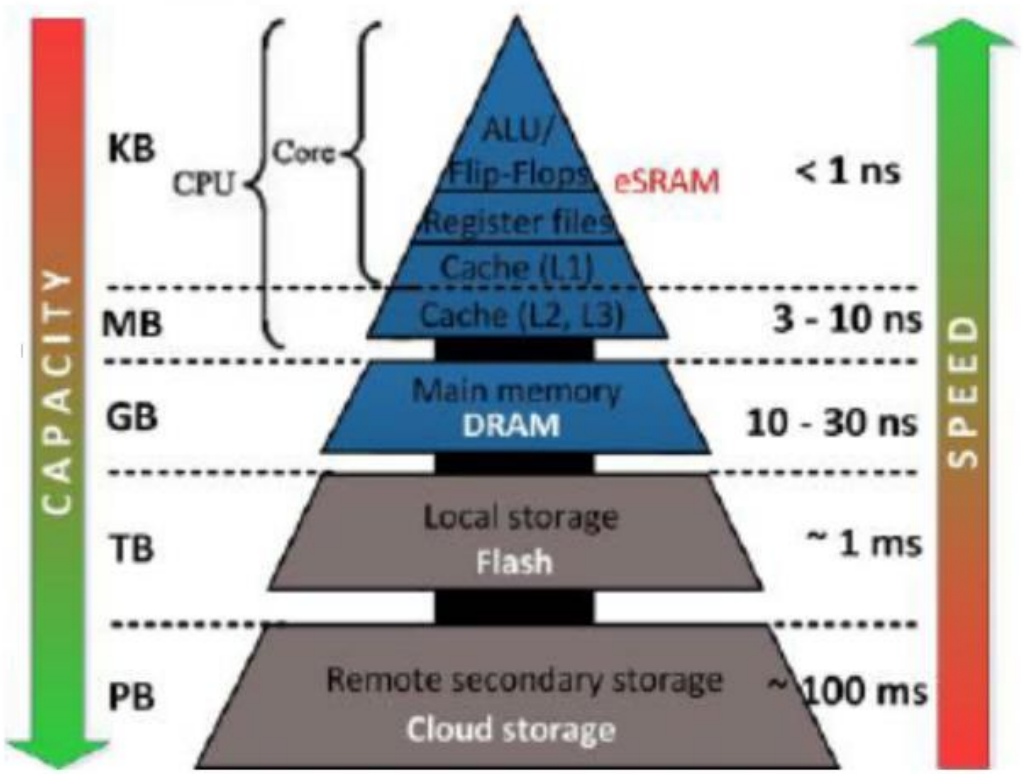



*Figure 4 - Memory Hierarchy*

The above figure demonstrates how the largest amount of memory (in bytes) at the end of the hierarchy is also the slowest for the CPU to access. The memory at the top is the smallest, most expensive and has the fastest access speed.

As you can see from the memory hierarchy, a computer has a limited amount of RAM and even less cache memory. When a large program or multiple programs are running, it's possible for memory to be fully used. To compensate for a shortage of physical memory, the computer's OS can create ***virtual memory***. *More on this later*

## Tools of Memory Management

- Swapping
- Paging, page tables and TLB
- Base and limit registers for Partitions
- Segmentation
- Virtual memory

## Memory Locations

High level languages are converted to machine code in several steps:

1. compilation,
2. linking to form an executable file, and at a later time
3. loading of the executable into main memory to run.

### Process Address Space: Logical and Physical Addresses

1. **Logical address** – generated by the CPU unique to individual programs; also referred to as ***virtual address***. To make it easier to manage memory of multiple processes, each program must have their own virtual address.

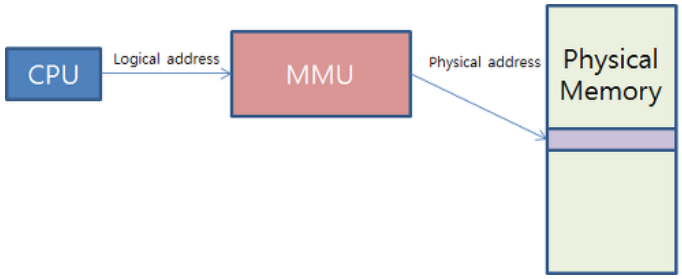
The set of all logical addresses generated by a program is referred to as a **logical address space**.

2. **Physical address** – address seen by the memory unit, unique to the system. Normally, two virtual addresses will not bound to the same physical address, if it is not specially intended.

The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The OS takes care of mapping the logical addresses to physical addresses (sometimes called ***address binding***) at the time of memory allocation to the program (logical address space that is bound to a separate physical address space) and this is central to proper memory management.

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, $2^{31}$ possible numbers, for a total theoretical size of 2GB.




The user program deals with ***logical addresses***; it never sees the real physical addresses.

Logical (virtual) and physical addresses are ***the same*** in compile-time and load-time address-binding schemes; Logical and physical addresses ***differ*** in execution-time address-binding scheme