Memory Management

Department of Computer Science & Engineering (Data Science)

R. C. Patel Institute of Technology, Shirpur

(An Autonomous Institute)

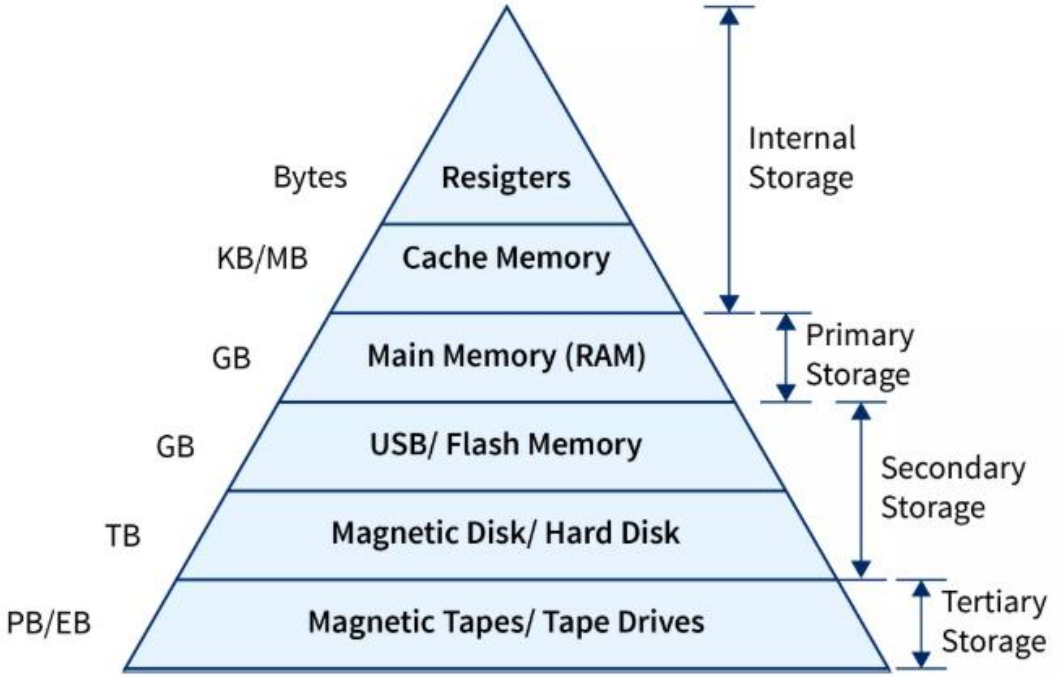
Unit-III 08 Hrs.

- Memory Organization: Memory Hierarchy, Main Memory, Cache Memory, Memory Mapping, cache coherence, Pentium IV cache organization, ARM cache organization.
- Memory Management: Memory partitioning: Fixed and Variable Partitioning,
- Memory Allocation: Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping, Virtual Memory, Paging. Segmentation,
- Demand paging and Page replacement policies.
- Comparative study of memory management in Windows, Linux and Android OS.

Memory Hierarchy

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time.

Memory Hierarchy helps in optimizing the memory available in the computer. There are multiple levels present in the memory, each one having a different size, different cost, etc. Some types of memory like cache, and main memory are faster as compared to other types of memory but they are having a little less size and are also costly whereas some memory has a little higher storage value, but they are a little slower. Accessing of data is not similar in all types of memory, some have faster access whereas some have slower access.

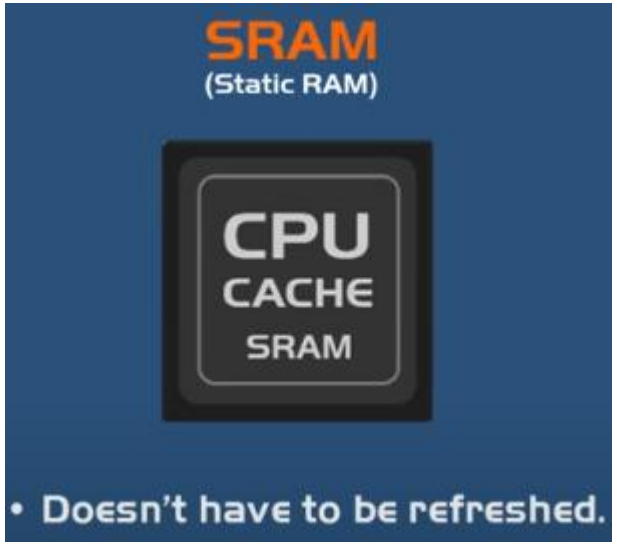


Registers

- These are very small memory devices, generally holding around bytes of data nearly 32-bit to 64-bit, and are of various types, for example - general-purpose registers, address registers, etc.
- The registers are directly associated with the CPU which means they are the fastest memory available in a system, which also means they are the costliest of all, Due to this they are used in very small sizes and possess the fastest access time.

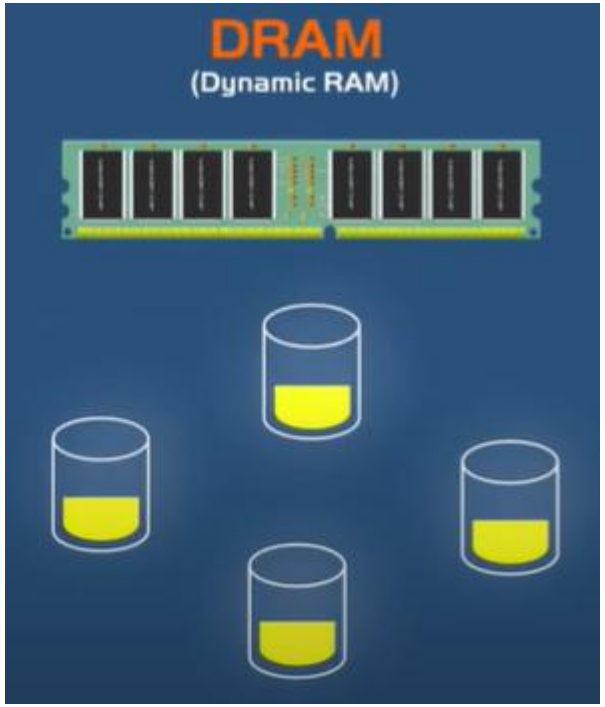
Cache Memory

- Cache memories are also called **SRAM** (Static Random Access Memory), they are measured in kilobytes or megabytes, and are used to store the segments of programs that are frequently accessed by the processor.
- Registers and Cache memories are embedded on the CPU itself, and hence are the fastest and are collectively referred to as internal memory.



Main Memory (RAM)

- This is the most commonly used memory, also called **DRAM** (Dynamic Random Access Memory), the main memory directly communicates with the CPU and auxiliary memory devices through input and output.
- These are less expensive as compared to cache memory and are measured in Gigabytes.



All the above three memories i.e., Registers, Cache Memory, and Main memory are volatile, meaning all the data inside them will be erased if the power is cut off.

Flash Memory

- Memories like Pendrives and Solid State Drives come in this category and are generally measured in Gigabytes.

Magnetic Disks

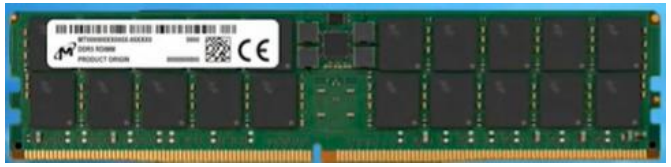
- Most commonly available memories, are slower as compared to flash memories, Memory storage like hard drives come in this category, and are measured in Gigabytes or Terabytes.

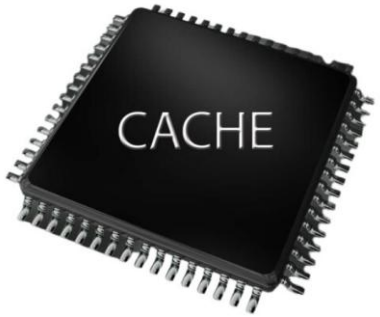
Flash Memory and Magnetic Disks are collectively referred to as **Secondary** Storage.

Magnetic Tapes

- Magnetic tapes are the largest in size provide the slowest access time, and are generally measured in Petabytes, they are mainly used as backup storage and are not directly connected to the CPU.

Magnetic tapes are referred to as **tertiary** storage.





Characteristics of Memory Hierarchy Design

Access Time

Access time is the time interval between the availability of the data and the subsequent read/write requests.
Access time increases if we travel from top to bottom in the memory hierarchy design.

Cost Per Bit

The ratio of cost per bit increases as we go from bottom to top, meaning the registers are the costliest memory and secondary and tertiary storage are the cheapest.

Frequency

If we talk about the frequency ie, which memory is used most frequently by the **CPU**, And the least used memory device is Magnetic tapes, as they are generally used for keeping the backups, and we know that on a regular basis, backups are not accessed, so they are the least used memory in a computer system.

Capacity

As we travel from top to bottom, the capacity increases, it simply means the volume of data the memory can store.

Memory	Type	Access Time (Approx)
Registers	Internal Memory	1ns
Cache Memory (SRAM)	Internal Memory	10ns
Main Memory (DRAM)	Primary Memory	100ns
Flash Drives	Secondary Storage	1µs
Magnetic Disks	Secondary Storage	10ms
Magnetic Tapes	Tertiary Storage	100ms

Types of Memory Hierarchy in Operating System

The Memory Hierarchy is divided into two types.

Internal Memory

Internal memory is also referred to as primary memory, in a computer system, internal memory is memory that stores the data that needs to be accessed frequently and quickly, this type of memory is directly reachable by the system and other processes through **input/output** modules. It consists of

- Registers
- Cache Memories
- RAM
- ROM

External Memory

Also known as Secondary Memory, and are used to store huge amounts of data, They are non-volatile memories, meaning they will be storing the data even if the power gets off. They consist of

- Magnetic Disks
- Optical Disks
- Magnetic Tapes

Advantages of Memory Hierarchy:

There are several advantages of using a memory hierarchy in computer systems, including:

- **Faster Access**: By having multiple levels of the memory hierarchy, computer systems can provide faster access to frequently accessed data. The fastest memory such as cache memory is used to store the most frequently used data, which can be accessed much faster than other types of memory.
- **Cost-Effective**: The use of a memory hierarchy enables the computer system to be cost-effective, as the cost of implementing high-speed memory such as cache memory is more expensive compared to other types of memory like main memory or secondary storage. With the use of a memory hierarchy, the most expensive memory can be used where it is needed the most, without needing to use it everywhere.
- **Efficient use of resources**: Memory hierarchy allows efficient use of resources by storing frequently accessed data in fast memory and less frequently accessed data in slower memory. This ensures that the computer system does not waste resources by using high-speed memory for data that is rarely accessed.
- **Increases the processing speed**: The use of a memory hierarchy allows computer systems to perform operations much faster. By using the fastest memory for frequently used data, the CPU can access the data quickly and complete the operations faster, increasing the processing speed of the system.
- **Increased capacity**: Memory hierarchy increases the overall capacity of the computer system by allowing the system to store large amounts of data in the secondary storage while keeping the most frequently accessed data in the high-speed memory. This allows the system to handle large amounts of data without slowing down its performance.

Cache Memory

Cache memory is a small, fast storage space within a computer. It holds duplicates of data from commonly accessed locations in the main memory. The CPU contains several separate caches that store both instructions and data.

The **key function** of cache memory is to **reduce the average time** needed to retrieve data from the main memory.

Working of Cache Memory

To understand the working of the cache, we must understand a few points:

1.Fast but Small: Cache is way faster than RAM but can only hold a small amount of data because it's limited in size.

2.Checking the Cache First: When the CPU needs data, it looks in the cache first because it's so quick. If the data is there ,the CPU grabs it and gets to work.

3.What Happens if Data Isn't in Cache? : If the data isn't in the cache, the CPU looks in the slower RAM, gets the data, and copies it to the cache for next time.

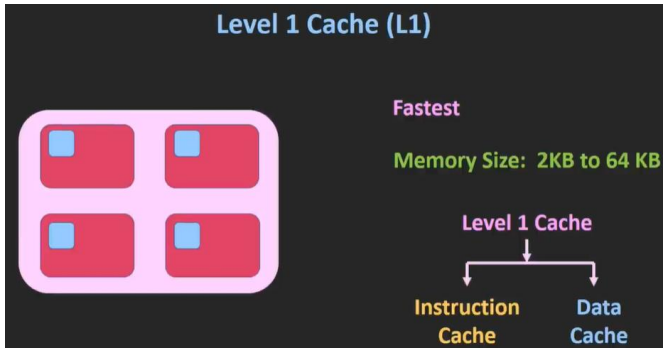
4.Speeding Things Up: By keeping frequently used data in the cache, the CPU spends less time waiting for data from RAM. This makes your computer run faster.

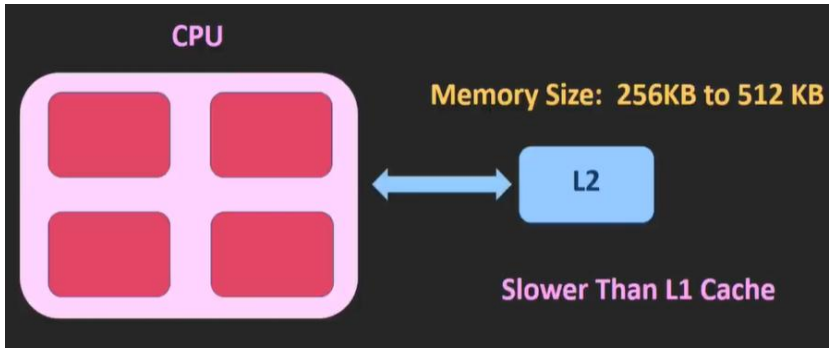
Types of Cache Memory

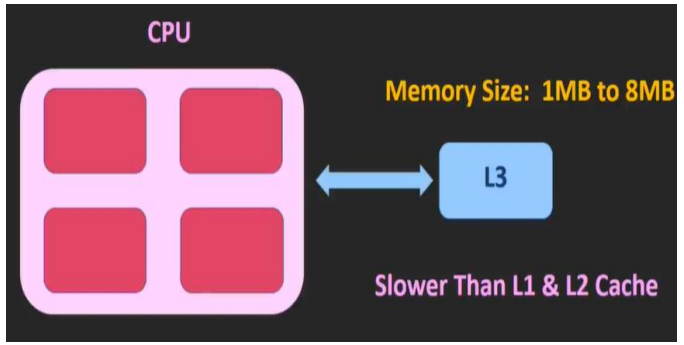
1.L1 or Level 1 Cache: It is the first level of cache memory that is present inside the processor. It is present in a small amount inside every core of the processor separately. The size of this memory ranges from 2KB to 64 KB.

2.L2 or Level 2 Cache: It is the second level of cache memory that may present inside or outside the CPU. If not present inside the core, It can be shared between two cores depending upon the architecture and is connected to a processor with the high-speed bus. The size of memory ranges from 256 KB to 512 KB.

3.L3 or Level 3 Cache: It is the third level of cache memory that is present outside the CPU and is shared by all the cores of the CPU. Some high processors may have this cache. This cache is used to increase the performance of the L2 and L1 cache. The size of this memory ranges from 1 MB to 8MB.







Why Cache Memory is Important

Cache memory acts as a bridge between the CPU and RAM, helping the CPU access data more quickly. It stores frequently used data so that the CPU doesn’t have to go all the way to the slower RAM. By keeping this data close, cache memory speeds up the CPU’s work and improves the overall performance of the computer.

How Cache Memory Improves CPU Performance

Cache memory helps improve the CPU's performance by reducing the time it takes to fetch data. By keeping the most frequently accessed data closer to the CPU, cache minimizes the need to access slower main memory (RAM). This reduction in wait time results in a much faster and more efficient system.

What is a Cache Hit and a Cache Miss?

Cache Hit: When the CPU finds the required data in the cache memory, allowing for quick access. On searching in the cache if data is found, a cache hit has occurred.

Cache Miss: When the required data is not found in the cache, forcing the CPU to retrieve it from the slower main memory. On searching in the cache if data is not found, a cache miss has occurred

Cache Mapping

Cache mapping refers to the method used to store data from main memory into the cache. It determines how data from memory is mapped to specific locations in the cache.

There are three different types of mapping used for the purpose of cache memory which is as follows:

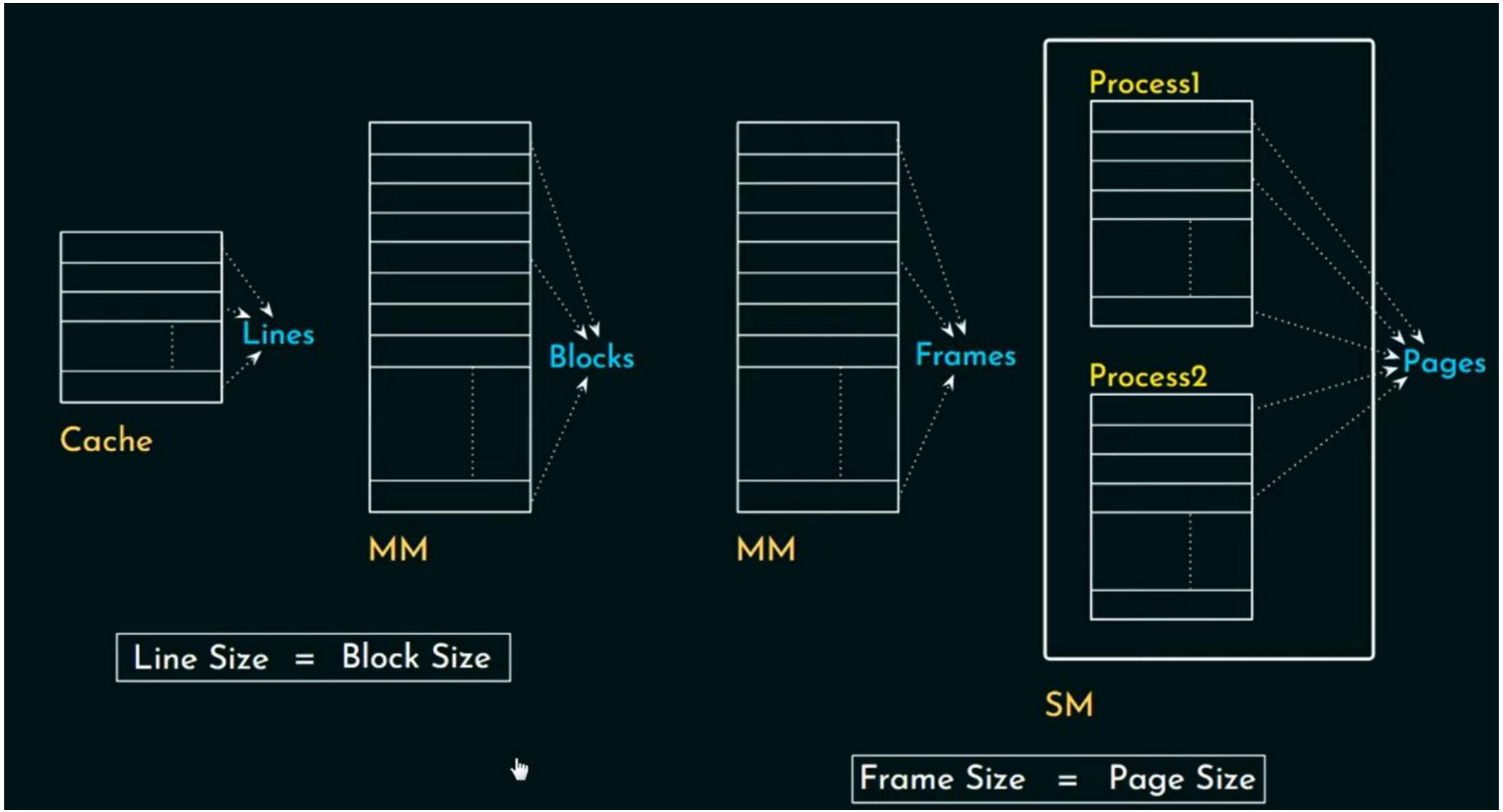
- Direct Mapping
- Fully Associative Mapping
- Set-Associative Mapping

The **Main Memory** consists of memory blocks and these blocks are made up of fixed number of words. A typical address in main memory is split into two parts:

2. **Index Field:** It represent the block number. Index Field bits tells us the location of block where a word can be.
3. **Block Offset:** It represent words in a memory block. These bits determines the location of word in a memory block.

The **Cache Memory** consists of cache lines. These cache lines has same size as memory blocks. The address in cache memory consists of:

2. **Block Offset:** This is the same block offset we use in Main Memory.
3. **Index:** It represent cache line number. This part of the memory address determines which cache line (or slot) the data will be placed in.
4. **Tag:** The Tag is the remaining part of the address that uniquely identifies which block is currently occupying the cache line.


Cache
Lines
MM
Blocks
MM
Frames
Process1
Process2
Pages
SM
Line Size = Block Size
Frame Size = Page Size

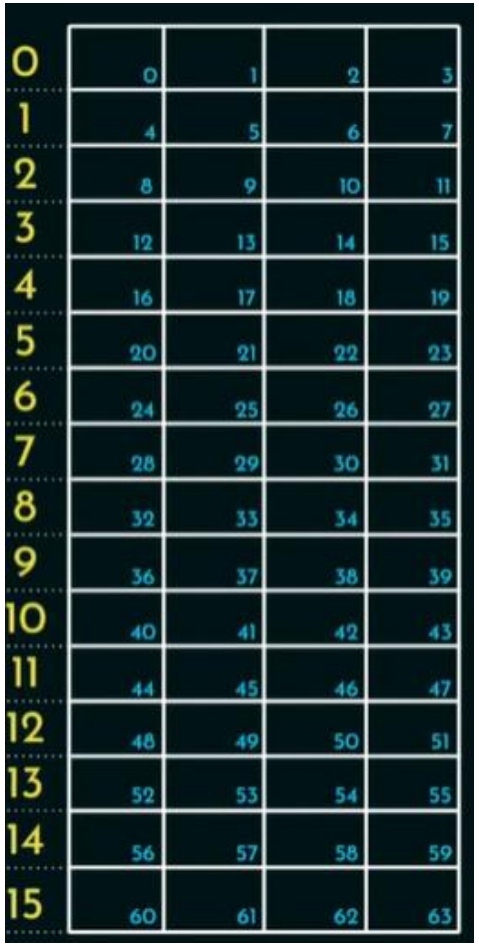
Word: Smallest Addressable Unit of Memory.

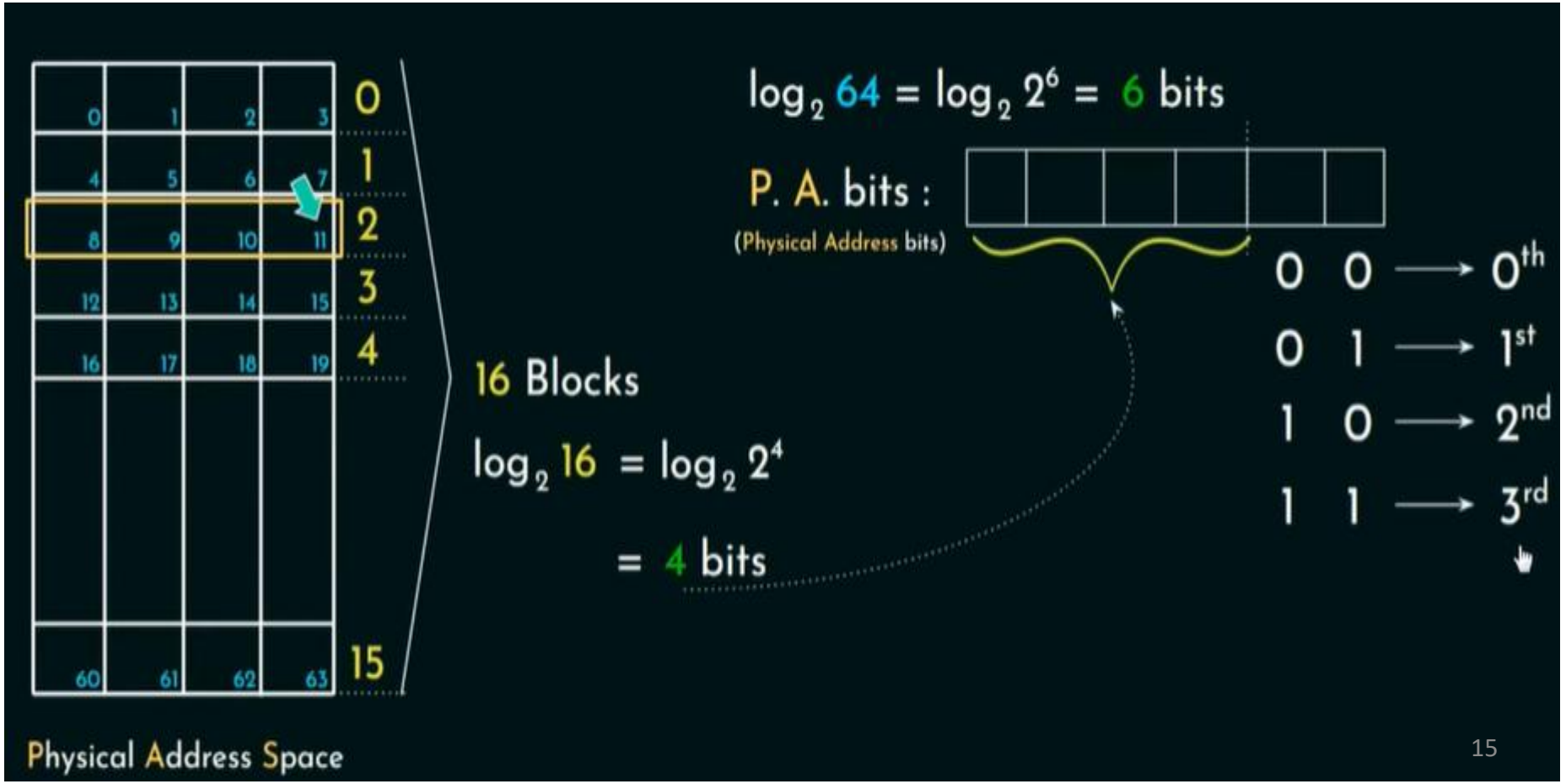
- Byte-Addressable Memory: 1 word = 1 Byte

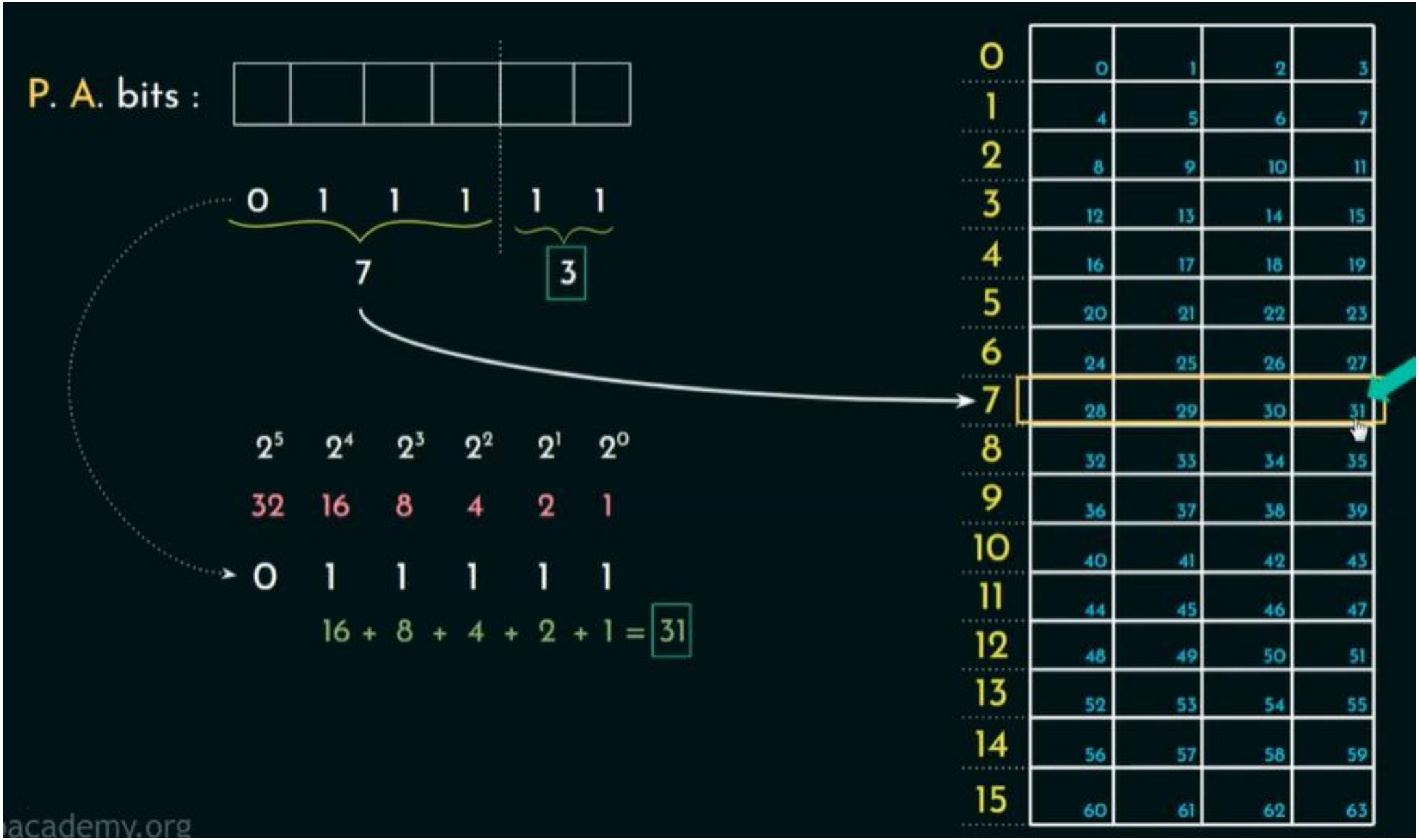
Main Memory Size : 64 words

Block Size : 4 words

No. of Blocks in MM : 64 / 4 = 16 i.e. 0, 1, 2, 3, ... , 15






P. A. bits :
0 1 1 1 1 1
7
3
2^5 2^4 2^3 2^2 2^1 2^0
32 16 8 4 2 1
0 1 1 1 1 1
16 + 8 + 4 + 2 + 1 = 31
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63

Cache Size : 16 words

Block Size : 4 words

Line Size : 4 words

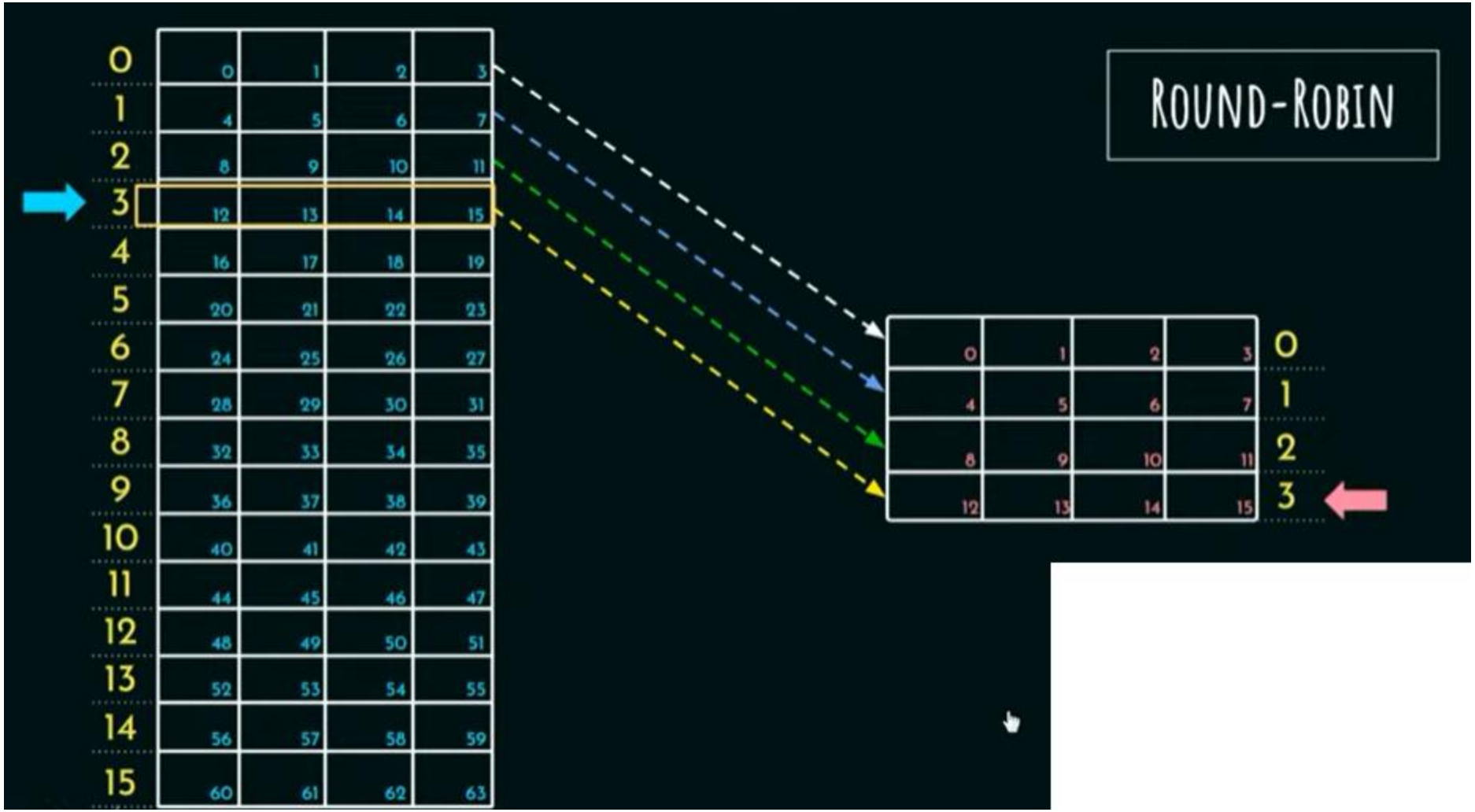
Block Size = Line Size

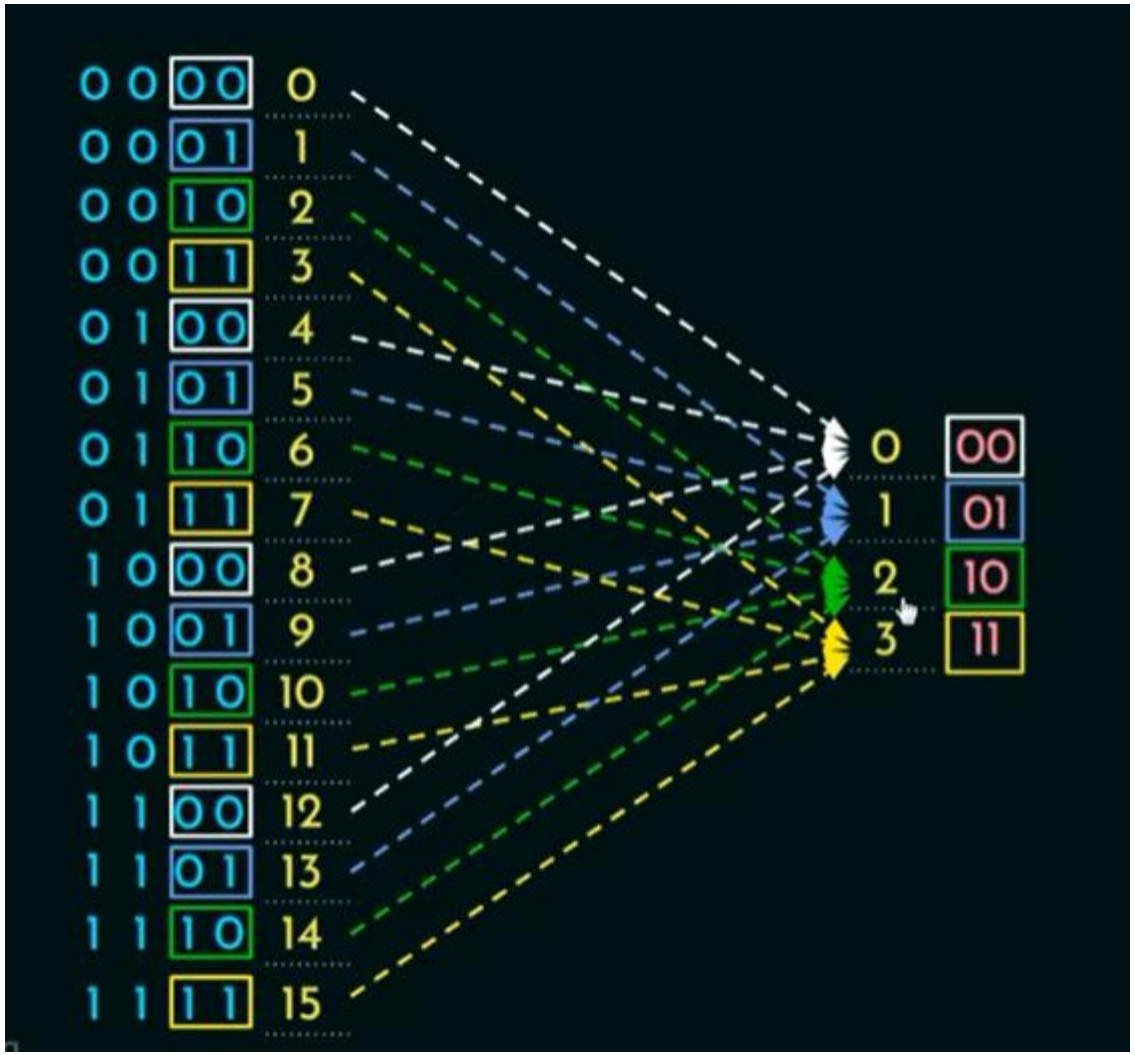
No. of Lines in Cache : 16 / 4 = 4 i.e. 0, 1, 2, 3

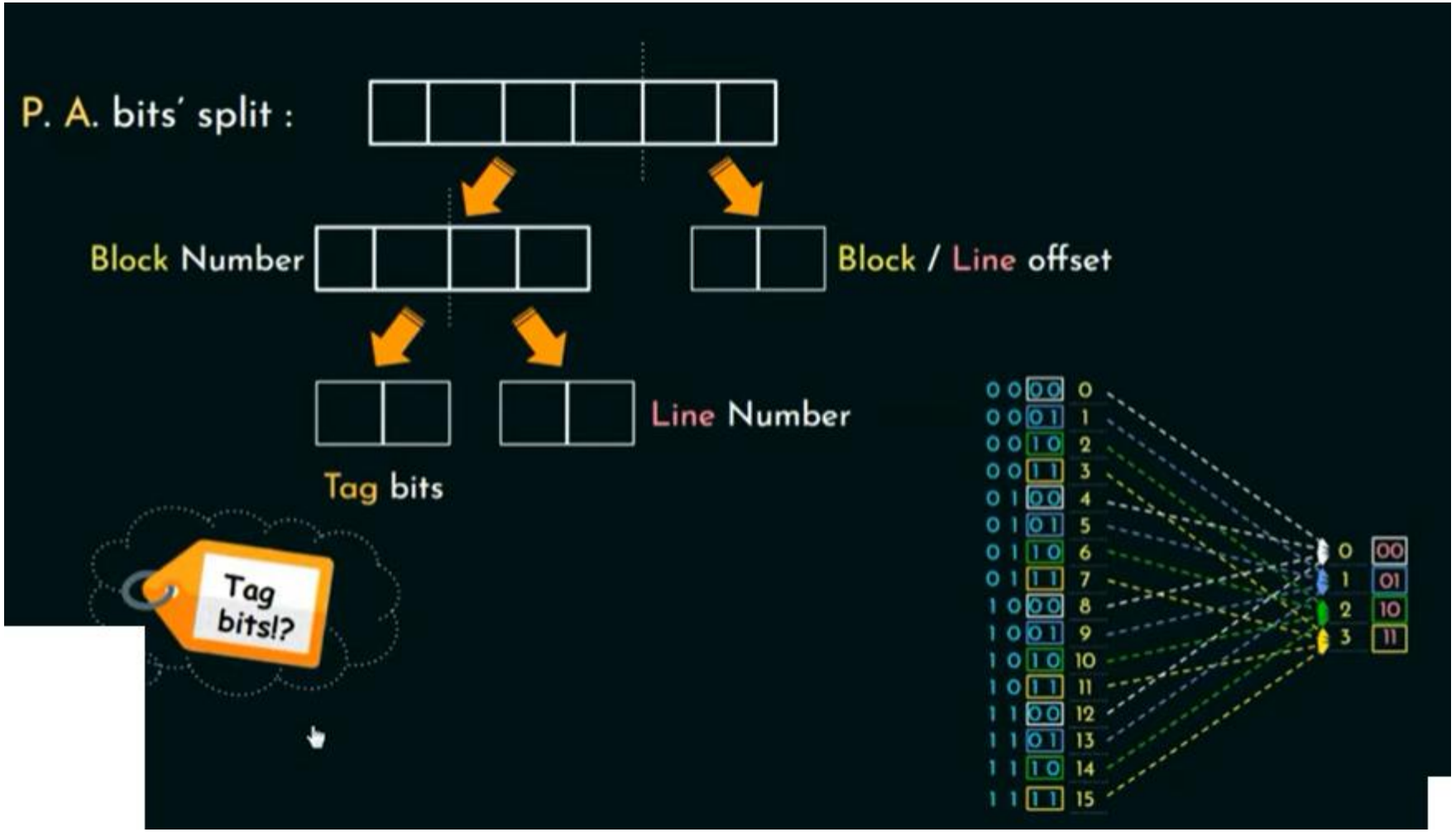
4 Lines

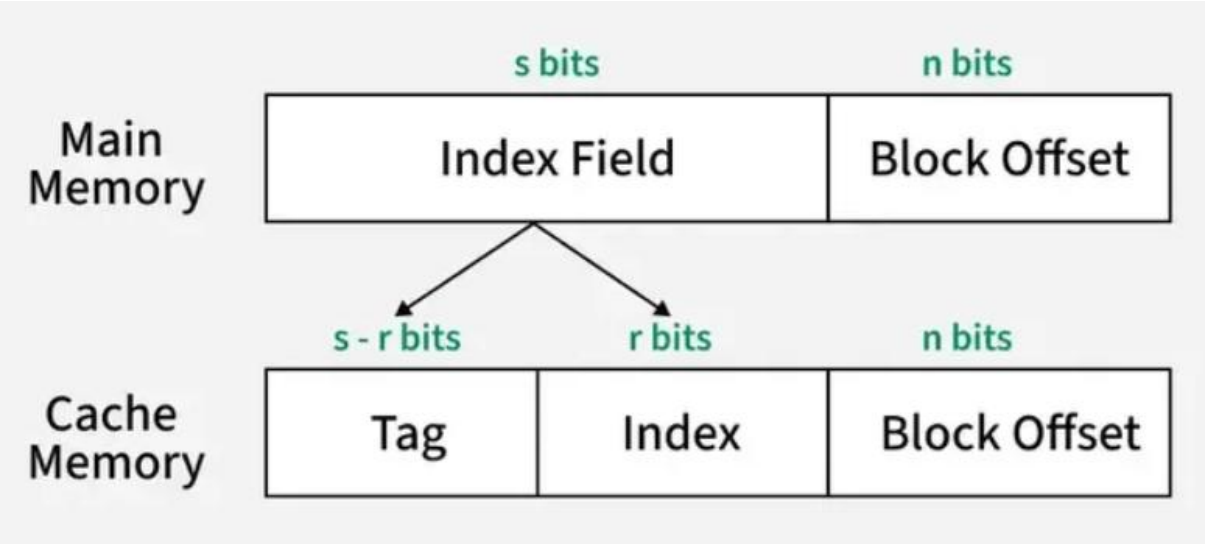
$\log_2 4 = \log_2 2^2$

= 2 bits


ROUND-ROBIN
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15


0 0 00 0
0 0 01 1
0 0 10 2
0 0 11 3
0 1 00 4
0 1 01 5
0 1 10 6
0 1 11 7
1 0 00 8
1 0 01 9
1 0 10 10
1 0 11 11
1 1 00 12
1 1 01 13
1 1 10 14
1 1 11 15
0 00
1 01
2 10
3 11


P. A. bits' split :
Block Number
Block / Line offset
Line Number
Tag bits
Tag bits!?
0 0 0 0 0
0 0 0 1 1
0 0 1 0 2
0 0 1 1 3
0 1 0 0 4
0 1 0 1 5
0 1 1 0 6
0 1 1 1 7
1 0 0 0 8
1 0 0 1 9
1 0 1 0 10
1 0 1 1 11
1 1 0 0 12
1 1 0 1 13
1 1 1 0 14
1 1 1 1 15
0 00
1 01
2 10
3 11

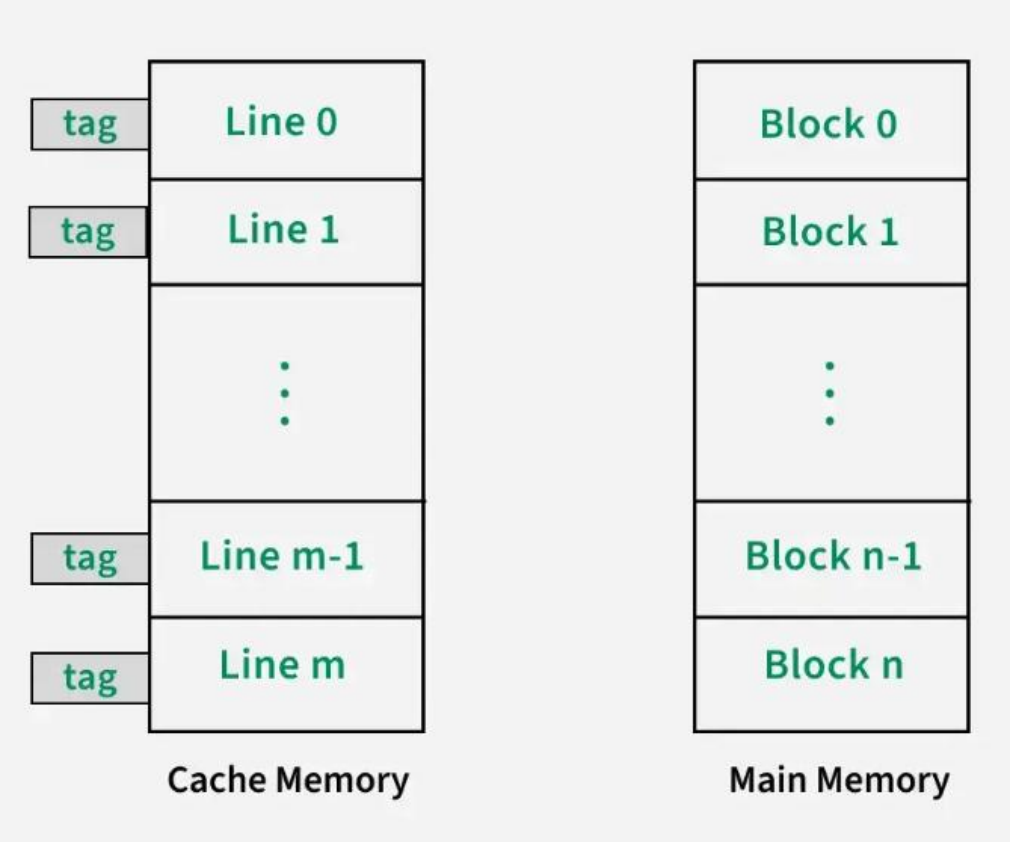


The index field in main memory maps directly to the index in cache memory, which determines the cache line where the block will be stored. The block offset in both main memory and cache memory indicates the exact word within the block. In the cache, the tag identifies which memory block is currently stored in the cache line. This mapping ensures that each memory block is mapped to exactly one cache line, and the data is accessed using the tag and index while the block offset specifies the exact word in the block.

2. Fully Associative Mapping

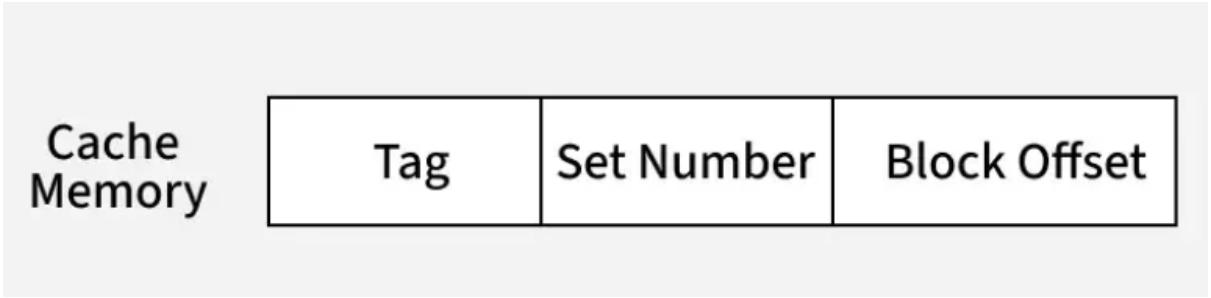
- Fully associative mapping is a type of cache mapping where any block of main memory can be stored in any cache line.
- Unlike direct-mapped cache, where each memory block is restricted to a specific cache line based on its index, fully associative mapping gives the cache the flexibility to place a memory block in any available cache line.
- This improves the hit ratio but requires a more complex system for searching and managing cache lines.
- The address structure of Cache Memory is different in fully associative mapping from direct mapping.
- In fully associative mapping, the cache does not have an index field.
- It only have a tag which is same as Index Field in memory address.
- Any block of memory can be placed in any cache line. This flexibility means that there’s no fixed position for memory blocks in the cache.

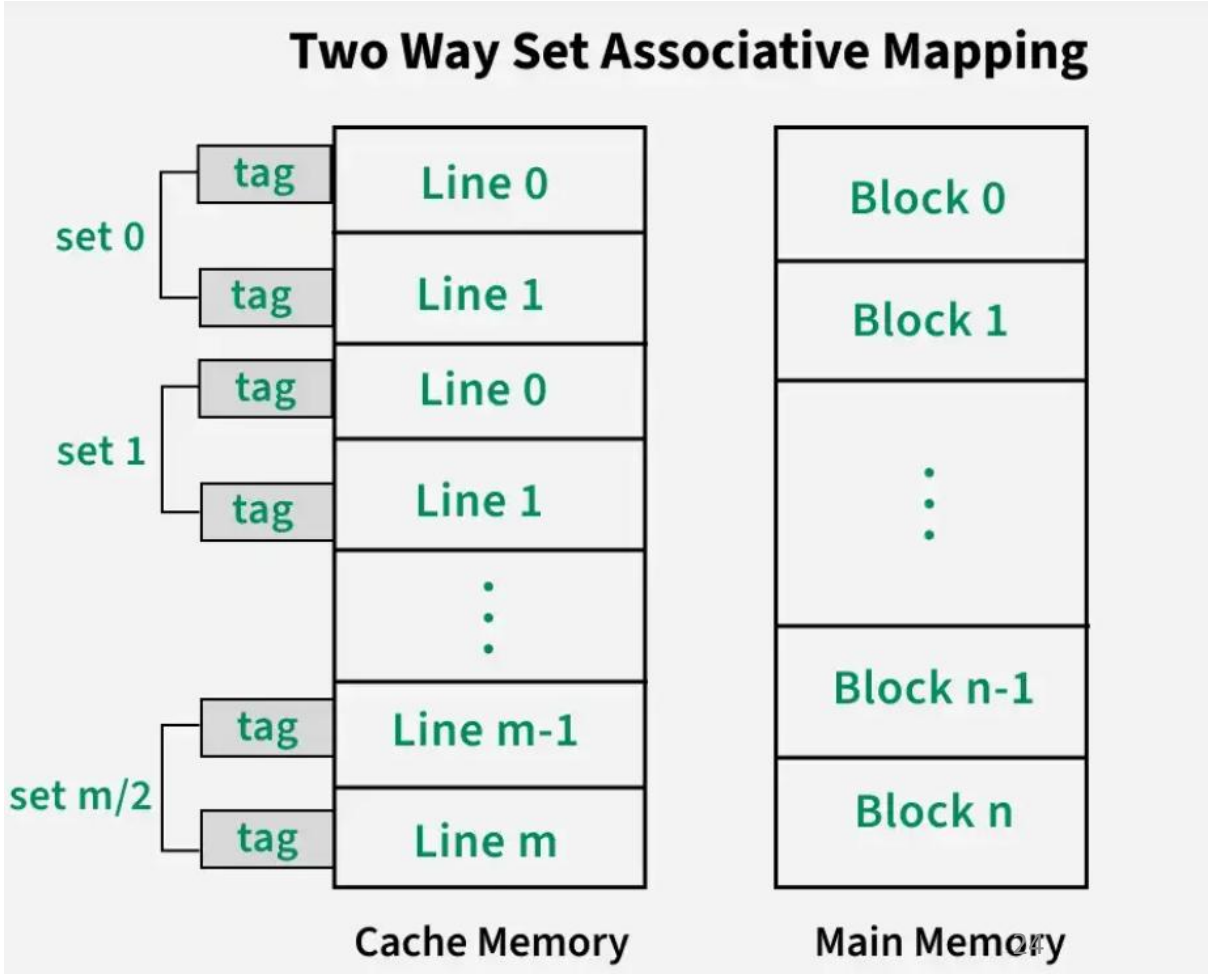
- To determine whether a block is present in the cache, the tag is compared with the tags stored in all cache lines.
- If a match is found, it is a cache hit, and the data is retrieved from that cache line. If no match is found, it's a cache miss, and the required data is fetched from main memory.



3. Set-Associative Mapping

- Set-associative mapping is a compromise between direct-mapped and fully-associative mapping in cache systems.
- It combines the flexibility of fully associative mapping with the efficiency of direct mapping.
- This reduces the conflict misses that occur in direct mapping while still limiting the search space compared to fully-associative mapping.

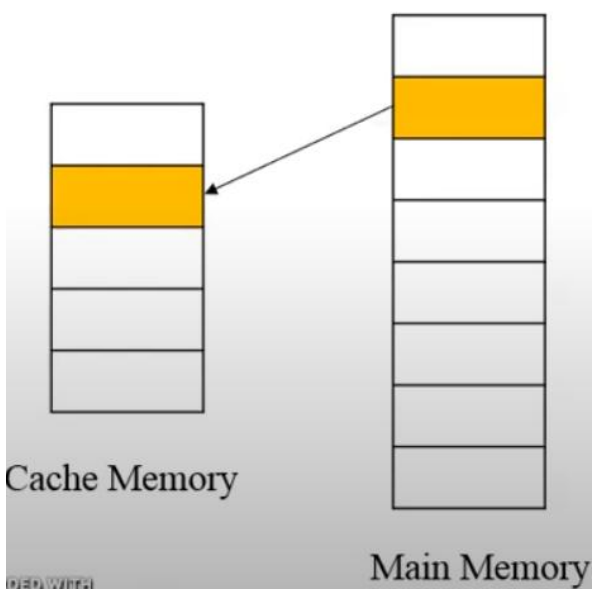


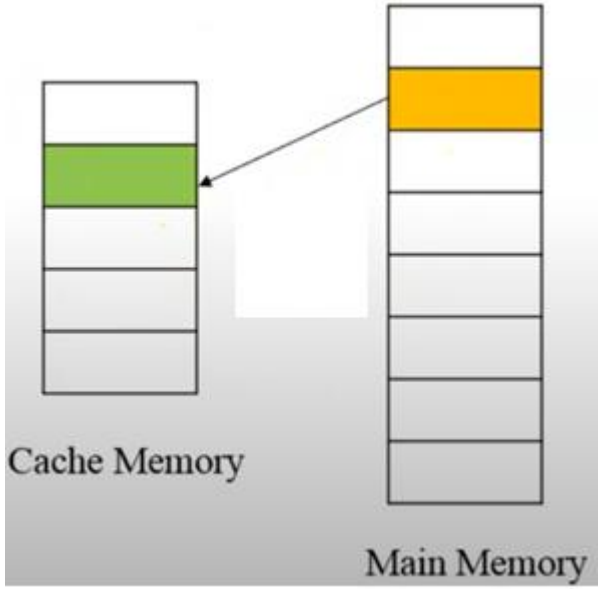


Cache Coherence

❑What is Cache Coherence Problem?

- In a single CPU system, two copies of same data, one in cache memory and another in main memory may become different. This data inconsistency is called as ***cache coherence problem.***

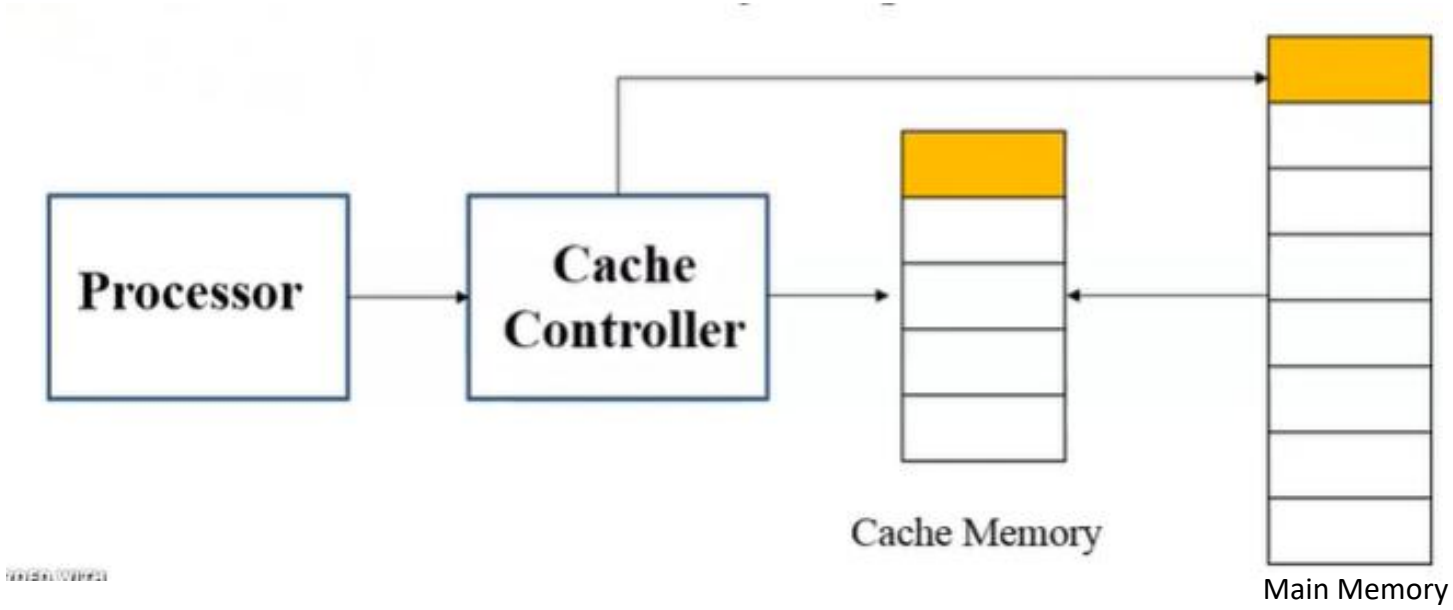




- Different strategies to writing into cache.

Write through cache: The cache controller copies data to the main memory immediately after it is written to the cache. Due to this, main memory always contains a valid data and thus any block in the cache can be overwritten immediately without data loss.

This approach require time to write data in main memory with increase in bus traffic. It is reduce system performance.



Write-back system:

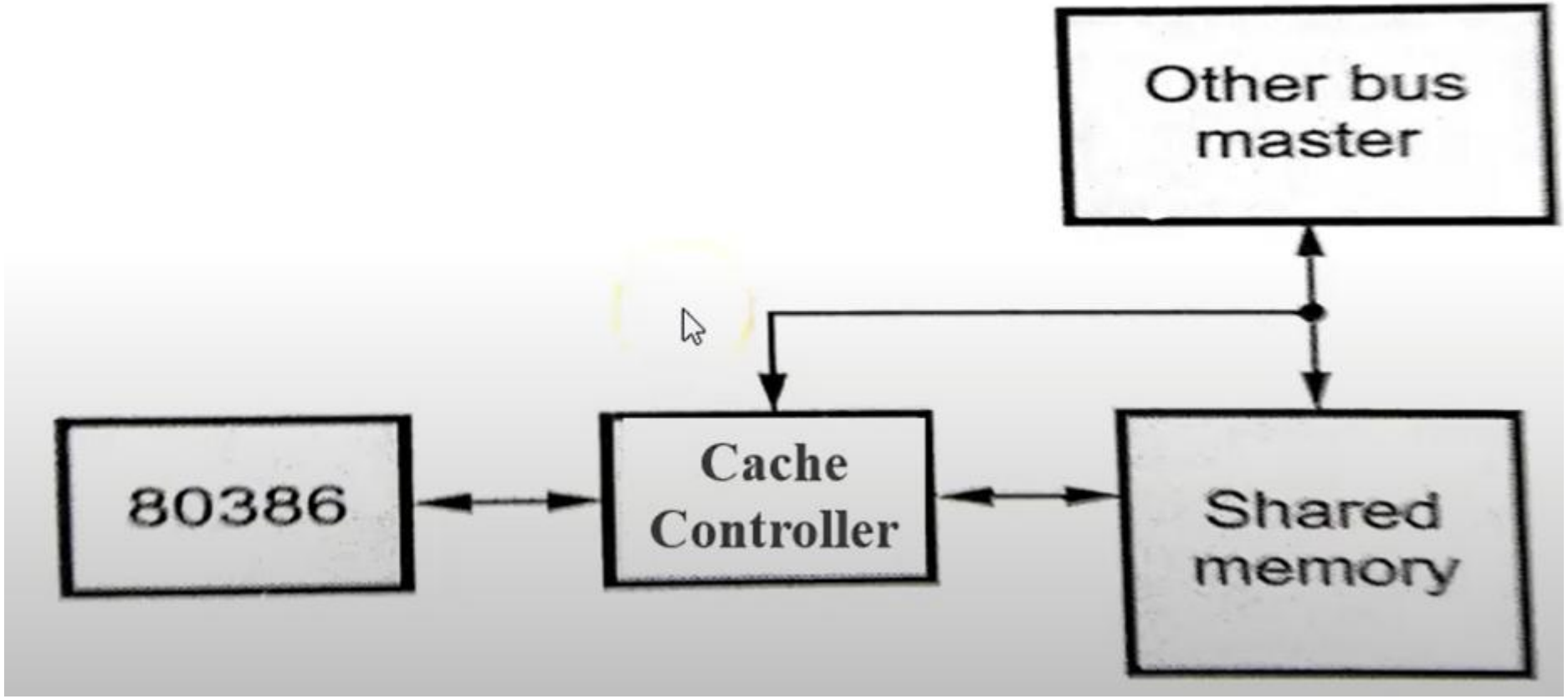
- In a write back system, the alter bit in the tag field is used to keep information of the new data. If it is set, the controller copies the block to main memory before loading new data into the cache.
- Due to one time write operation, number of write cycles are reduced in write-back system. But this system has following disadvantage.
  - It is necessary that, all altered blocks must be written to the main memory before another device can access these blocks in main memory.

Buffered write through system:

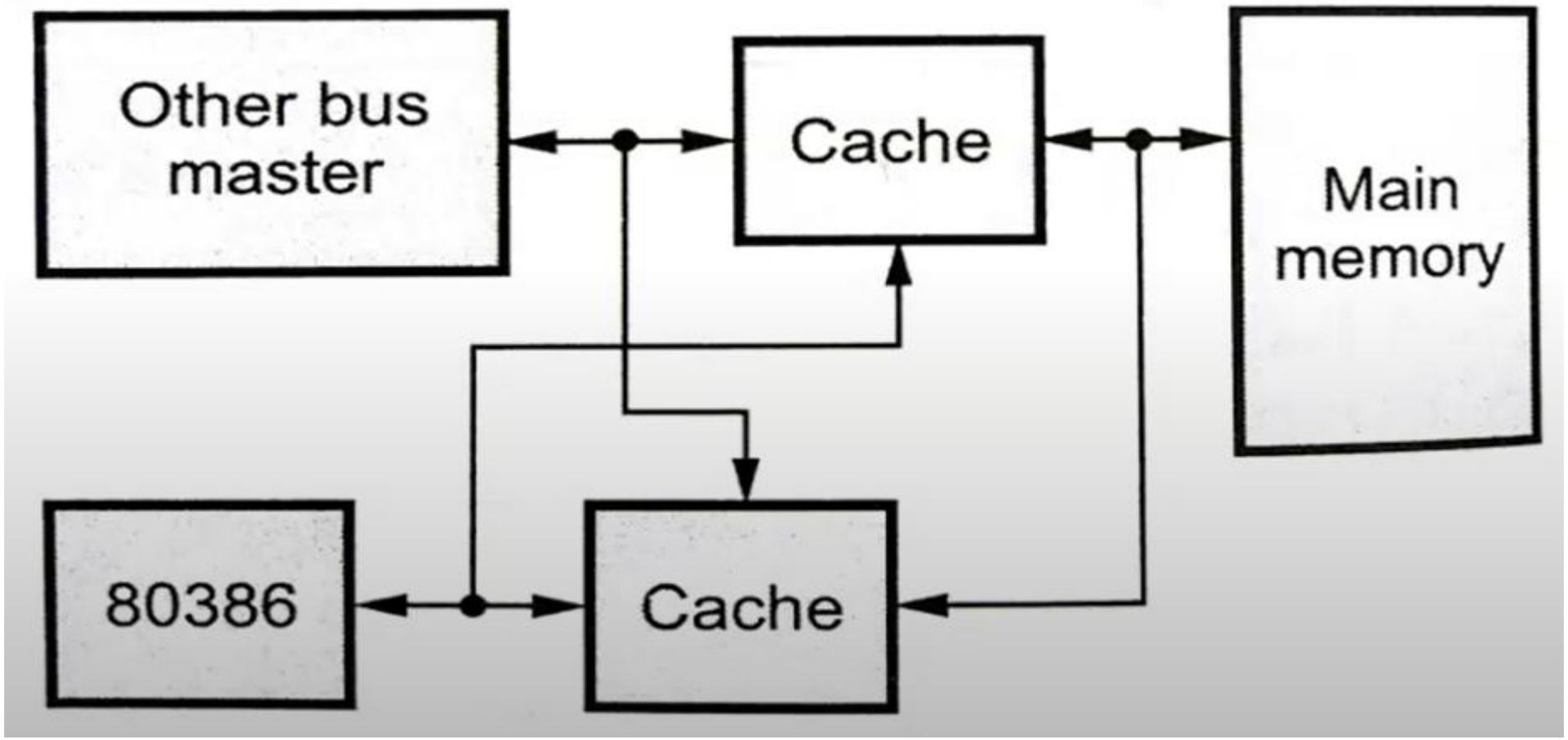
- In buffered write through system, the processor can start a new cycle before the write cycle to the main memory is completed. This means that the write accesses to the main memory are buffered.
- In such systems, read access which is a "cache hit" can be performed simultaneously when main memory is updated.
- However two consecutive write operations to the main memory and read operation with cache "miss" require to processor the wait.

- There are four different approaches to prevent data inconsistency, that is to protect cache coherency :

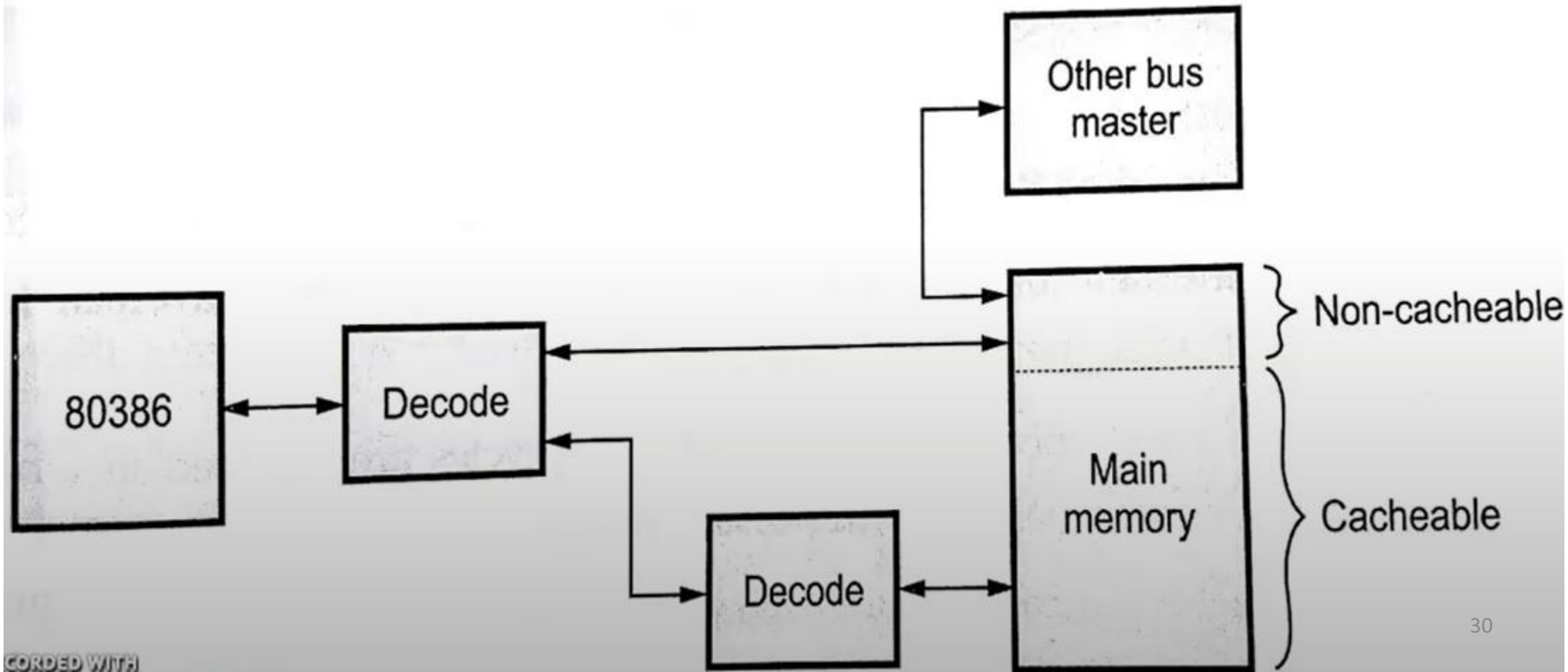
- **Bus watching :** In bus watching, cache controller invalidates the cache entry, if another master writes to a location in shared memory which also resides in the cache memory. Below diagram shows bus watching.



Hardware Transparency: In hardware transparency, accesses of all devices to the main memory are routed through the same cache or by copying all cache writes both to the main memory and to all other caches that share the same memory. Below diagram shows hardware transparency.



Non-cacheable memory: Processor can partition its main memory into a cacheable and non-cacheable memory. By designing shared memory as non-cacheable memory cache coherency can be maintained, since shared memory is never copied into cache.



Cache flushing: To avoid data inconsistency, a cache flush means, writes any altered data to the main memory and caches in the system are flushed before a device writes to shared memory.

Pentium IV cache organization

The Pentium 4 has three levels of cache:

- Level 1 cache
- Level 2 cache
- Level 3 cache

Level 1 Cache

- The level 1 cache is a split cache.
- 8KB in size.
- four-way set associative. This means that each set is made up of four lines in cache.
- The replacement algorithm used for this is a “least recently used” algorithm.
- The line size is 64 bytes.

Level 1 Cache

- The level 1 cache is small to reduce latency, taking 2 cycles for an integer data cache hit and 6 cycles for a floating point.

Instead of a classic level 1 instruction cache, the Pentium 4 uses a trace cache which takes advantage of the advanced branch prediction algorithms.

Trace Cache

- After the instructions have been decoded into micro-ops, they are stored in the trace cache.
- Six micro-ops are stored for each trace line. The trace cache can store up to 12K micro-ops.
- Since the instructions have already been decoded, the hardware knows about any branches and fetches instructions that follow the branch.

Level 2 Cache

- The level 2 cache is a unified cache.
- 256KB in size.
- eight-way set associative. This means that each set is made up of eight lines in cache.
- The line size is 128 bytes
- The replacement algorithm used for this is a "least recently used" algorithm.

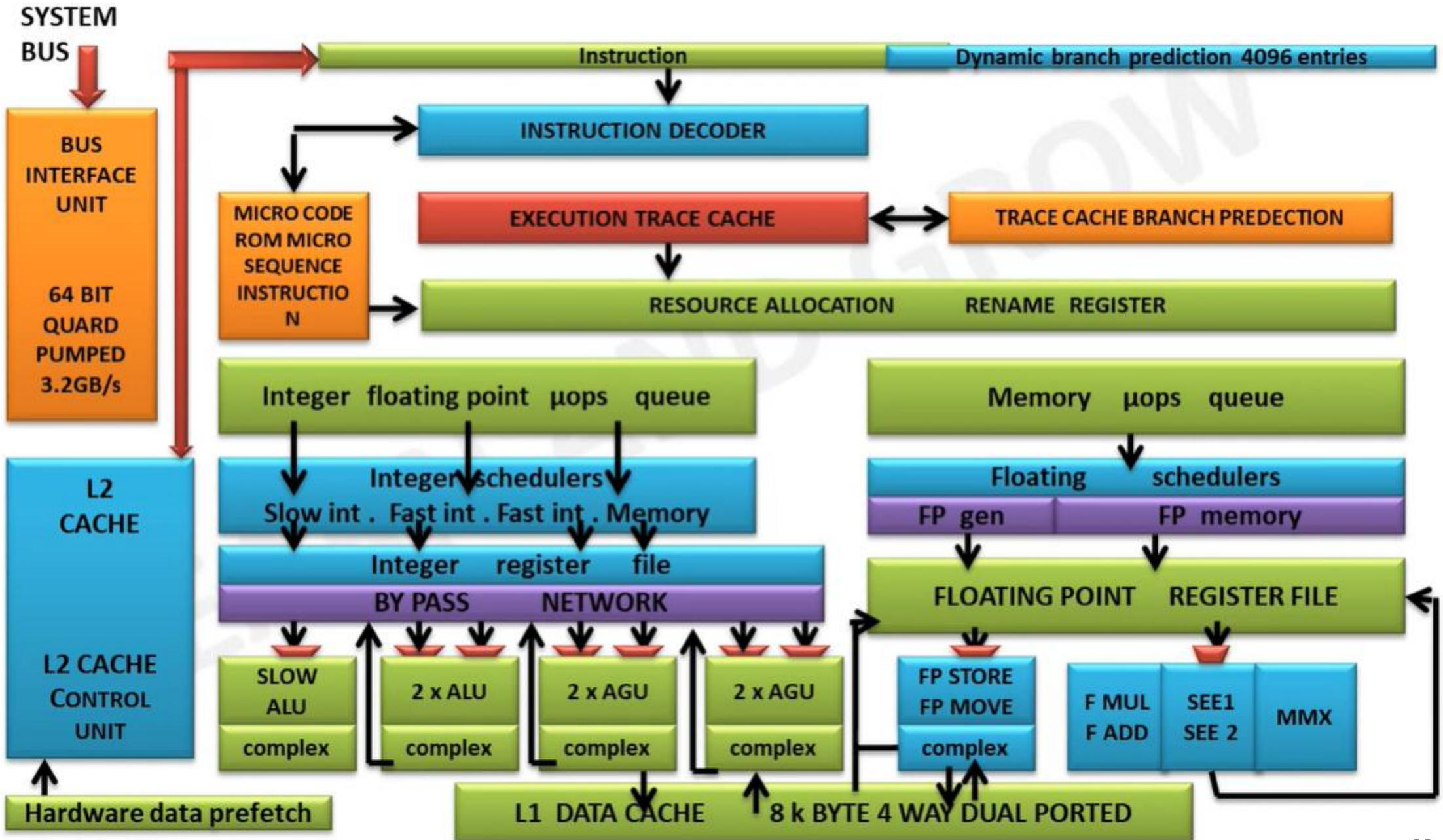
Level 2 Cache

- The increase in size and set size means that it will reduce the chances of a miss occurring when accessing this cache, increasing its effectiveness.
- The increase in line size can cause higher latency, so the Pentium 4 employs a 400MHz system bus using a 100MHz clock that delivers a data rate of 3.2GB/s to make up for the latency.

Level 3 Cache

- eight-way set associative .
- line size of 128 bytes.
- make use of a “least recently used” replacement algorithm.

This provides a large on-processor tertiary memory storage area that the processor uses for keeping information nearby. Thus, the contents of the Level 3 cache are faster to access than main memory, but slower than other types of cached information.


SYSTEM BUS
BUS INTERFACE UNIT
64 BIT QUARD PUMPED 3.2GB/s
L2 CACHE
L2 CACHE CONTROL UNIT
Hardware data prefetch
Instruction
Dynamic branch prediction 4096 entries
INSTRUCTION DECODER
MICRO CODE ROM MICRO SEQUENCE INSTRUCTIO N
EXECUTION TRACE CACHE
TRACE CACHE BRANCH PREDECTION
RESOURCE ALLOCATION RENAME REGISTER
Integer floating point µops queue
Memory µops queue
Integer schedulers
Slow int . Fast int . Fast int . Memory
Floating schedulers
FP gen
FP memory
Integer register file
BY PASS NETWORK
FLOATING POINT REGISTER FILE
SLOW ALU
complex
2 x ALU
complex
2 x AGU
complex
2 x AGU
complex
FP STORE FP MOVE
complex
F MUL F ADD
SEE1 SEE 2
MMX
L1 DATA CACHE
8 k BYTE 4 WAY DUAL PORTED

ARM (Advanced RISC Machine) cache organization

ARM processors use a **hierarchical cache memory** system to reduce memory latency and improve performance. The cache system is typically **multi-level** (e.g., L1, L2, and sometimes L3), and its organization can vary based on the specific ARM architecture.

1. **Cache Levels in ARM**

Level	Type	Features
L1	Instruction (I-Cache) and Data (D-Cache), typically separate (Harvard architecture)	Small (8KB–64KB), fastest, per core
L2	Unified or separate	Larger (128KB–2MB), slower than L1, can be shared or per core
L3	Unified	Found in high-performance cores shared across cores

2. Cache Types

- **Instruction Cache (I-Cache):** Caches instructions to be executed
- **Data Cache (D-Cache):** Caches data being read or written
- **Unified Cache:** Stores both data and instructions (typically L2 or L3)

3. Cache Architecture

a. Harvard vs. Von Neumann:

•**Harvard Architecture:** Separate instruction and data caches (common in L1)

•**Von Neumann Architecture:** Unified cache (common in L2 and L3)

b. Associativity:

•**Direct-mapped**

•**Set-associative (e.g., 2-way, 4-way, 8-way)** — more common

•**Fully-associative** — rare, more complex

c. Write Policies:

•**Write-through:** Writes data to both cache and memory simultaneously

•**Write-back:** Writes only to cache initially, then writes to memory later (more efficient)

d. Replacement Policies:

LRU (Least Recently Used)

Where is the ARM processor used?

- **Smartphones** (e.g., Snapdragon, Apple A-series)
- **Tablets and Laptops** (e.g., Apple M1/M2/M3)
- **Gaming Consoles** (e.g., Nintendo Switch)
- **Automotive Systems**
- **Smart Home Devices**
- **IoT Devices**

Memory Management

Memory management is a critical aspect of operating systems that ensures efficient use of the computer's memory resources. It controls how memory is allocated and deallocated to processes, which is key to both performance and stability.

Logical and Physical Address Space

Logical Address Space: The logical address space is the set of all addresses that a process can generate using its CPU. It defines the range of memory locations available to the process from its perspective.

Physical Address Space: The physical address space is the set of all actual memory addresses in the main memory (RAM). It represents the real locations where data and instructions are stored.

Logical Address

A logical address is generated by the CPU while a program runs. It represents the address from the process's perspective and does not exist physically, hence it is also called a virtual address.

.

Physical Address

A physical address is the real location in main memory (RAM) where data or instructions are stored. The physical address space consists of all physical addresses corresponding to logical addresses.

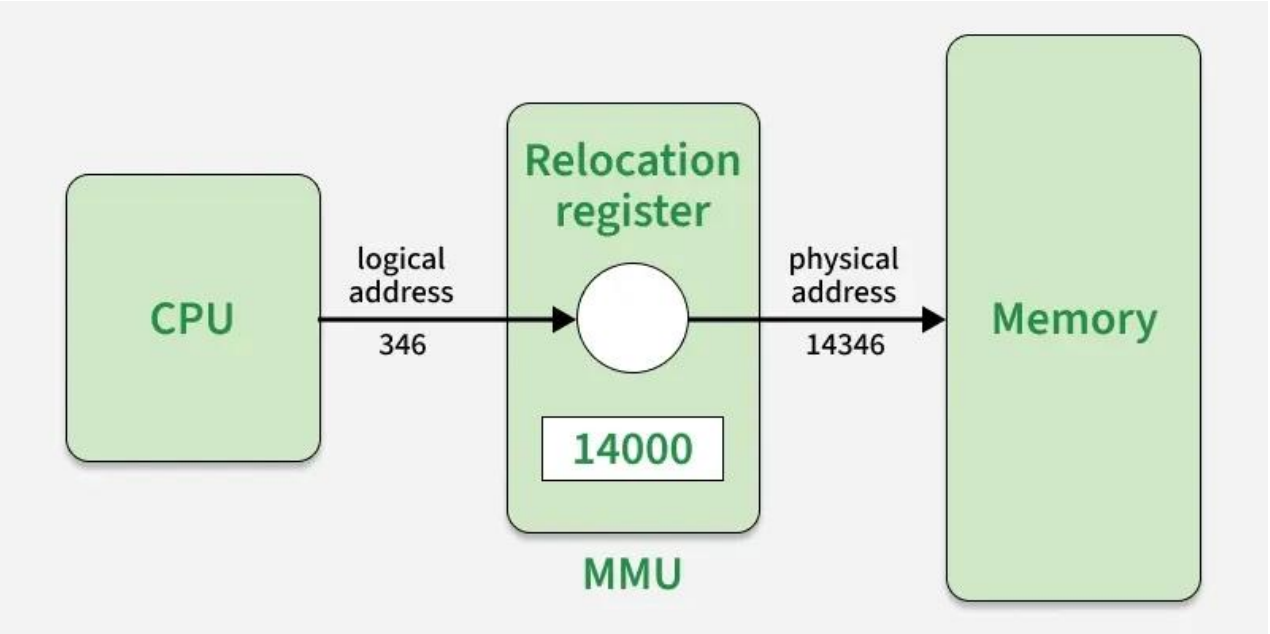
The MMU performs address translation using a page table, mapping each logical page to a physical frame. This allows processes to access memory transparently, without knowing actual memory locations.

Memory Management Unit

The **Memory Management Unit (MMU)** is a hardware component in the computer system that handles all memory and caching operations associated with the CPU. Its main function is to translate **logical (virtual) addresses** generated by the CPU into **physical addresses** in main memory.

Important Points about Logical and Physical Addresses in Operating Systems

•Logical addresses provide abstraction, so processes don’t need to know physical locations.
•Logical addresses are mapped to physical addresses via the page table.
•Translation is transparent and handled by hardware (MMU).
•Enables efficient memory management through paging and segmentation.



Difference Between Logical address and Physical Address

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Static and Dynamic Loading

Loading a process into the main memory is done by a loader. There are two different types of loading :

Static Loading: Static Loading is basically loading the entire program into a fixed address. It requires more memory space.

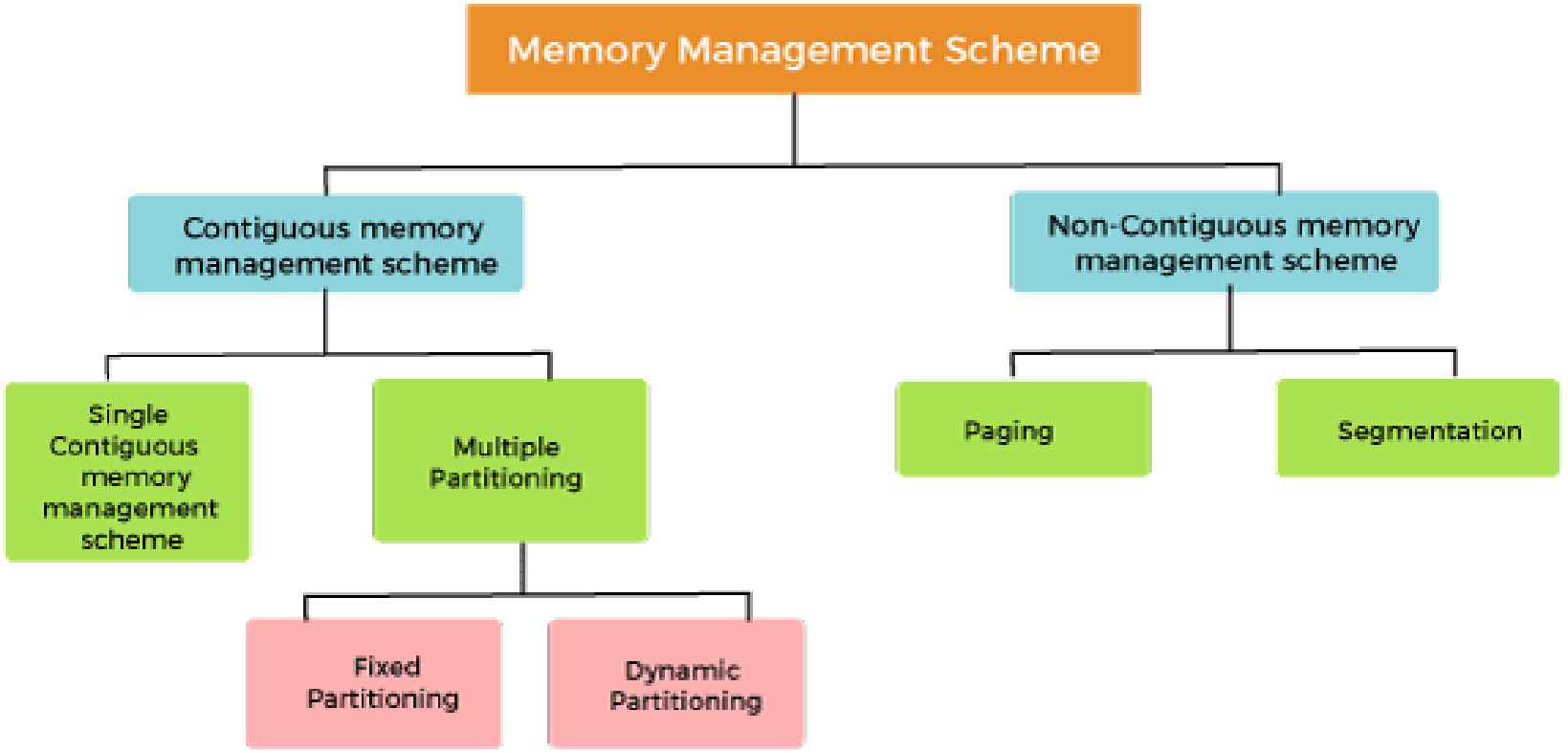
Dynamic Loading: Dynamic loading loads program routines into memory only when they are needed. This saves memory by not loading unused routines.

Swapping

Swapping moves processes between main memory and secondary memory to manage limited memory space. It allows multiple processes to run by temporarily swapping out lower priority processes for higher priority ones.

Memory Management Techniques

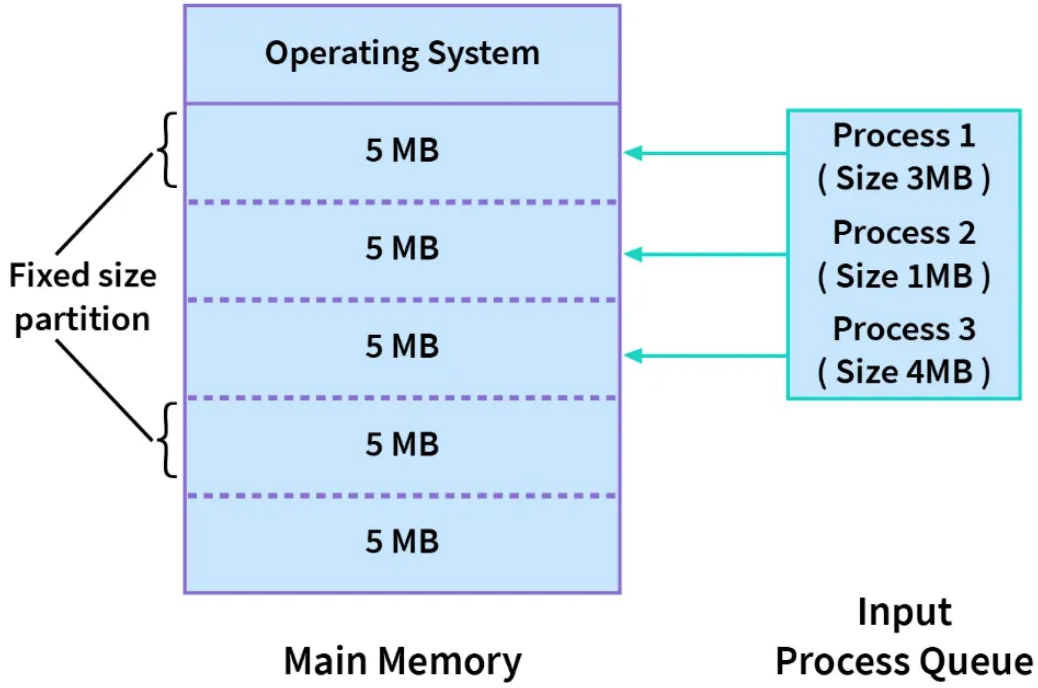
Memory management techniques are methods used by an operating system to efficiently allocate, utilize, and manage memory resources for processes. Various techniques help the operating system manage memory effectively.



Classification of memory management schemes

Contiguous memory management schemes

Contiguous memory management schemes are memory allocation techniques that involve allocating a contiguous block of memory to a process or program. In these schemes, each process is given a single, contiguous block of memory in which it can load and execute.



Contiguous memory management schemes

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Multiple Partitioning:

1.Fixed Partitioning:

- In fixed partitioning, the memory is divided into a fixed number of partitions or segments, each of a predefined size.
- Each partition can hold one process or program. The size of the partitions is determined during system configuration.
- Processes are assigned to partitions based on their size. Small processes may share a partition, while larger processes require entire partitions.
- Fixed partitioning is relatively simple to implement but can lead to inefficient memory utilization, as there may be internal fragmentation (unused memory within a partition).
- It is typically used in older systems where memory requirements were relatively small and fixed

2. Variable Partitioning:

- Variable partitioning is a more flexible version of contiguous memory management, where partitions can vary in size.
- Memory is divided into variable-sized partitions, and processes are allocated memory based on their actual size, with no fixed partition sizes.
- A process is allocated the smallest available partition that can accommodate it.
- Variable partitioning helps reduce internal fragmentation, as processes are allocated memory more precisely. However, it requires dynamic memory allocation and management.
- This scheme is commonly used in modern operating systems to handle varying memory requirements of processes efficiently.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.
There are different Placement Algorithm:

- First Fit
- Best Fit
- Worst Fit
- Next Fit

1. **First Fit**: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.
2. **Best Fit** Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.
3. **Worst Fit** Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.
4. **Next Fit:** Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Fragmentation

Fragmentation is the inefficient use of memory that occurs when free space is divided into small, scattered blocks, making it difficult to allocate larger chunks of memory

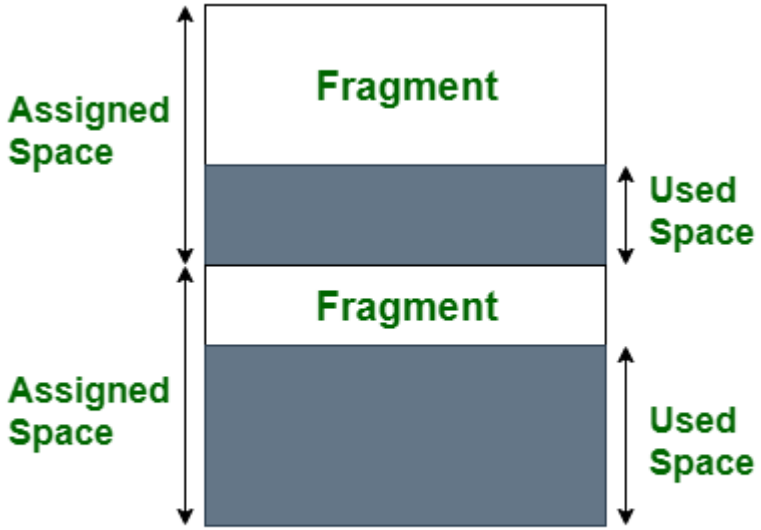
Types of Fragmentation

There are two main types of fragmentation:

- Internal Fragmentation
- External Fragmentation

1. Internal Fragmentation

Internal fragmentation occurs when there is unused space within a memory block. For example, if a system allocates a 64KB block of memory to store a file that is only 40KB in size, that block will contain 24KB of internal fragmentation. When the system employs a fixed-size block allocation method, such as a memory allocator with a fixed block size, this can occur.

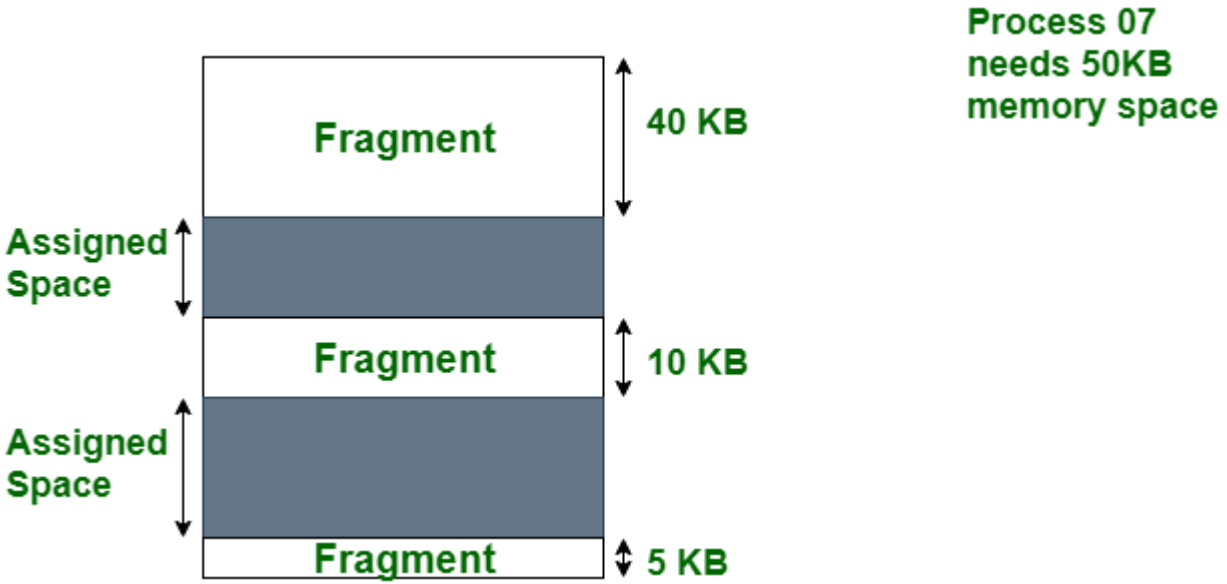


Internal Fragmentation

Fragmentation

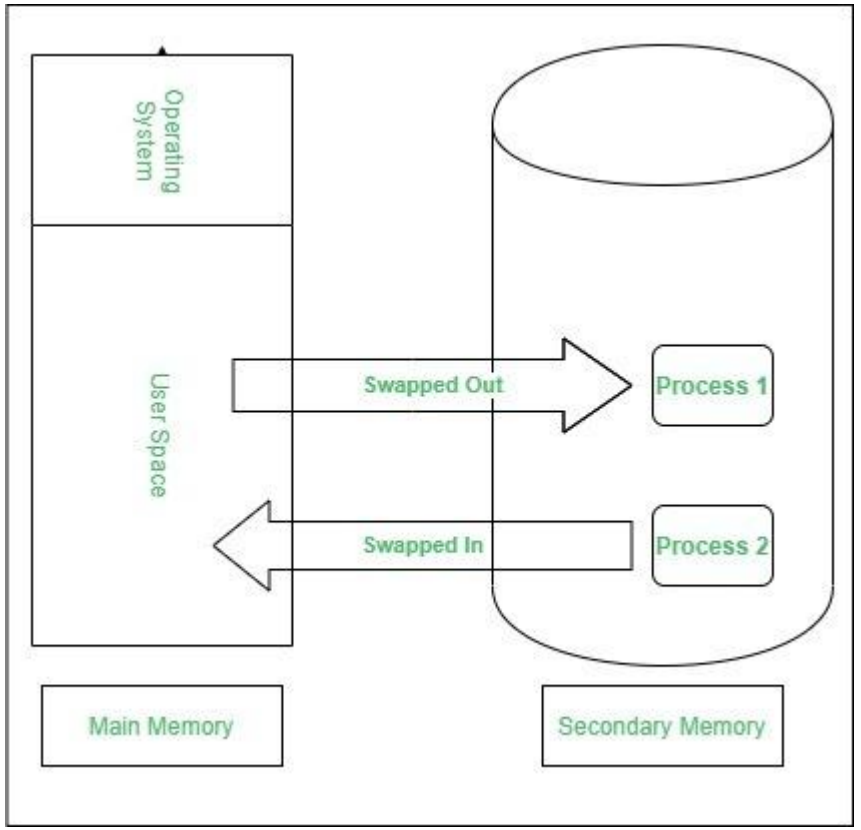
2.**External fragmentation:**

This occurs when there is enough total free memory, but it is broken up into small, non-contiguous "holes" or blocks. A new process that requires a contiguous block of memory cannot be allocated, even though the total free space is sufficient.



Swapping

- Swapping in an operating system is a process that moves data or programs between the computer's main memory (RAM) and a secondary storage (usually a hard disk or SSD).
- This helps manage the limited space in RAM and allows the system to run more programs than it could otherwise handle simultaneously.
- It's important to remember that swapping is only used when data isn't available in RAM.
- Although the swapping process degrades system performance, it allows larger and multiple processes to run concurrently. Because of this, swapping is also known as memory compaction.
- The CPU scheduler determines which processes are swapped in and which are swapped out.
- Consider a multiprogramming environment that employs a priority-based scheduling algorithm.
- When a high-priority process enters the input queue, a low-priority process is swapped out so the high-priority process can be loaded and executed. When this process terminates, the low-priority process is swapped back into memory to continue its execution.



Swapping has been subdivided into two concepts:

swap-in and swap-out.

- Swap-out is a technique for moving a process from RAM to the hard disc.
- Swap-in is a method of transferring a program from a hard disc to main memory, or RAM.

Process of Swapping

•When the RAM is full and a new program needs to run, the operating system selects a program or data that is currently in RAM but not actively being used.

•The selected data is moved to secondary storage, freeing up space in RAM for the new program

•When the swapped-out program is needed again, it can be swapped back into RAM, replacing another inactive program or data if necessary.

Advantages

- Swapping minimizes the waiting time for processes to be executed by using the swap space as an extension of RAM. Swapping allows the operating system to free up space in the main memory (RAM).
- Using only single main memory, multiple process can be run by CPU using swap partition.
- It allows larger programs or applications to run on systems with limited physical memory
- By swapping out inactive processes, the operating system can prevent the system from becoming overloaded.

Disadvantages

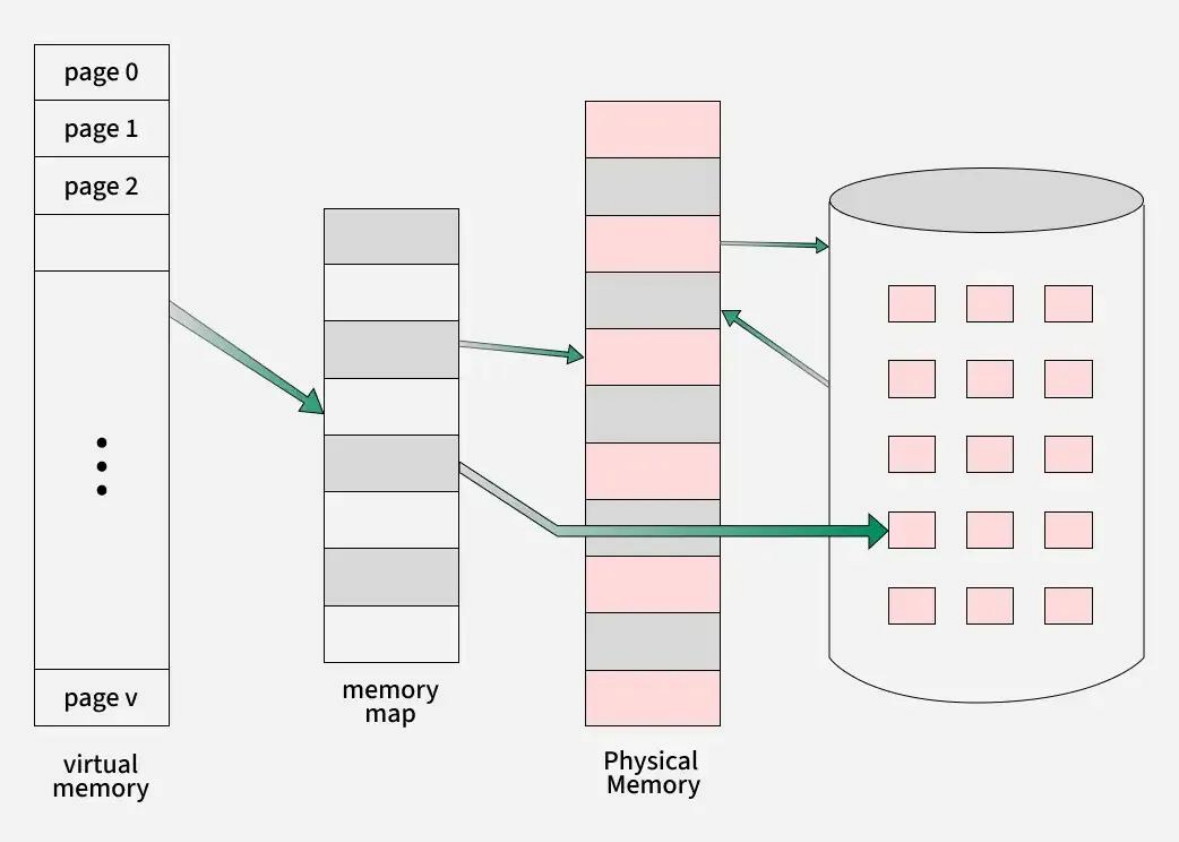
- Risk of data loss during swapping arises because of the dependency on secondary storage for temporary data retention.
- The **number of page faults** increases as the system frequently swaps pages in and out of memory.
- If the system Swaps-in and out too often, the performance of system can severely decline as CPU will spend more time swapping then executing processes.

Virtual Memory

- Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited and not necessarily allocated in contiguous manner.
- The main idea is to divide the process in pages, use disk space to move out the pages if space in main memory is required and bring back the pages when needed.

Objectives of Virtual Memory

- A program doesn't need to be fully loaded in memory to run. Only the needed parts are loaded.
- Programs can be bigger than the physical memory available in the system.
- Virtual memory creates the illusion of a large memory, even if the actual memory (RAM) is small.
- It uses both RAM and disk storage to manage memory, loading only parts of programs into RAM as needed.
- This allows the system to run more programs at once and manage memory more efficiently.



Virtual Memory

How Virtual Memory Works

- Virtual memory uses both hardware and software to manage memory.
- When a program runs, it uses virtual addresses (not real memory locations).
- The computer system converts these virtual addresses into physical addresses (actual locations in RAM) while the program runs.

Types of Virtual Memory

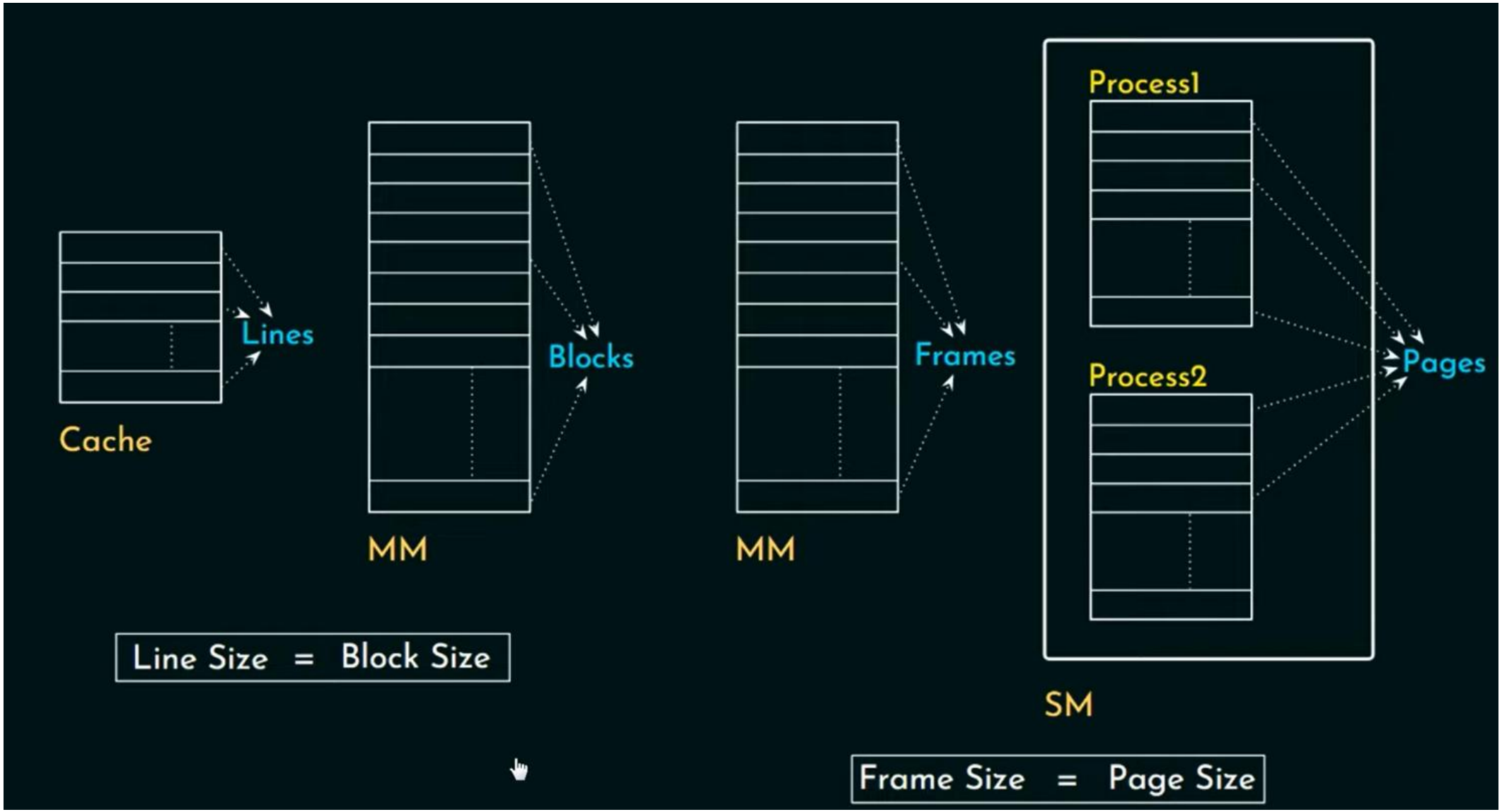
In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU. The CPU generates virtual addresses that the MMU translates into physical addresses. There are two main types of virtual memory:

1.Paging

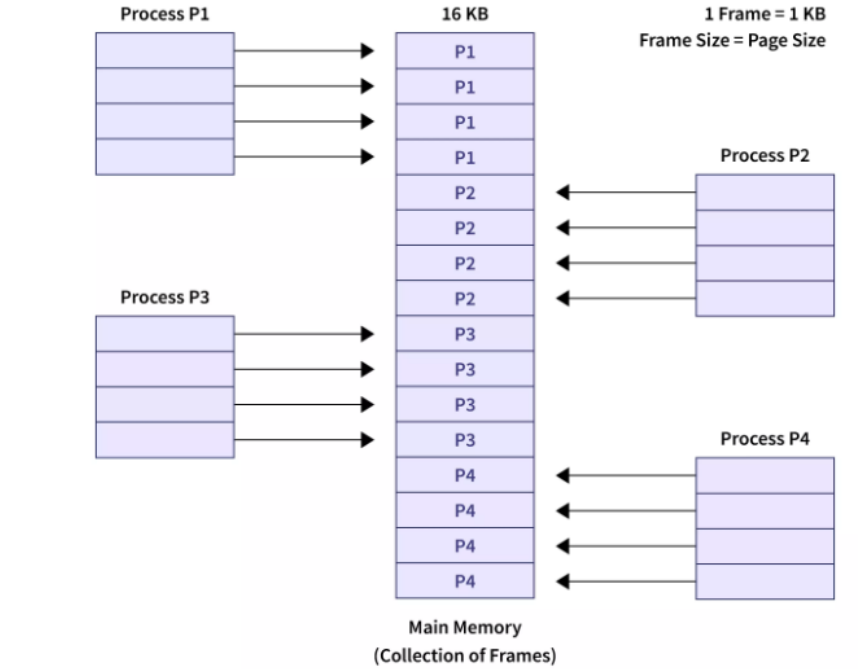
2.Segmentation

Paging:

- In paging, both physical memory and the process's logical address space are divided into fixed-size blocks called "pages."
- Physical memory is divided into page frames, which are also of the same size as pages.
- When a process is loaded into memory, it is divided into fixed-size blocks, or pages, and these pages can be scattered throughout physical memory.
- A page table is used to map logical pages to physical page frames. Each entry in the page table contains the mapping information.
- Paging eliminates external fragmentation because pages can be allocated in any available page frame, and internal fragmentation is minimal.
- It allows for efficient memory allocation, and it simplifies memory management. However, it may incur some overhead due to the page table.

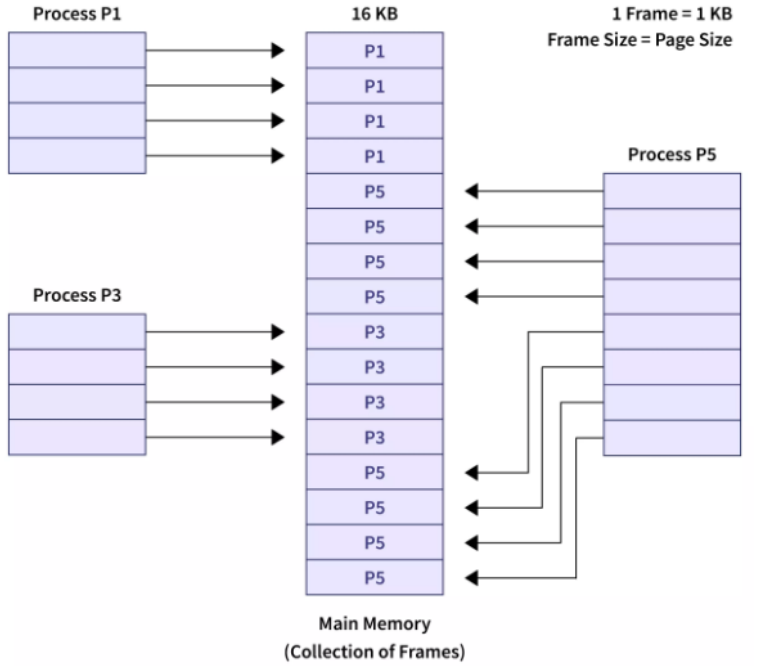

Cache
Lines
MM
Blocks
MM
Frames
Process1
Process2
Pages
SM
Line Size = Block Size
Frame Size = Page Size

Case-1



As we can see in the image, we have main memory divided into 16 frames of the size of 1KB each. Also, there are 4 processes available in the secondary (local) memory: P1, P2, P3, and P4 of a size of 4KB each. Clearly, each process needs to be further subdivided into pages of size of 1KB each, so that one page can be easily mapped to one frame of the main memory. This divides each process into 4 pages and the total for 4 processes gives 16 pages of 1KB each. Initially, all the frames were empty and therefore, pages will be allocated here in a contiguous manner.

Case-2



Let us assume that in Case-1, processes P2 and P4 are moved to the waiting state after some time and leave behind the empty space of 8 frames. **In Case-2, we have another process P5 of size 8KB (8 pages) waiting inside the ready queue to be allocated.** We know that with Paging, we can store the pages at different locations of the memory, and here, we have 8non-contiguous frames available. Therefore, we can easily load the 8 pages of the process P5 in the place of P2 and P4 which we can observe in the image.

Important Features of Paging

Logical to physical address mapping: Paging divides a process's logical address space into fixed-size pages. Each page maps to a frame in physical memory, enabling flexible memory management.

Fixed page and frame size: Pages and frames have the same fixed size. This simplifies memory management and improves system performance.

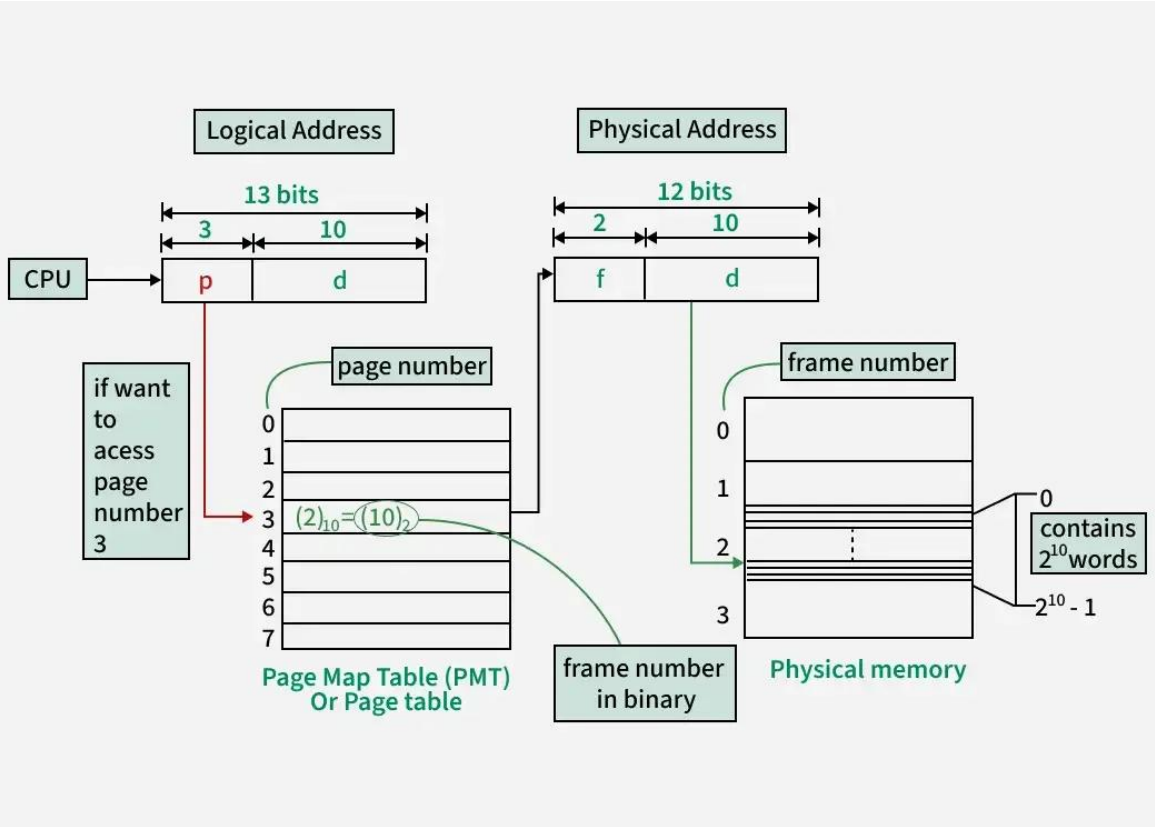
Page table entries: Each logical page is represented by a page table entry (PTE). A PTE stores the corresponding frame number and control bits.

Number of page table entries: The page table has one entry per logical page. Thus, its size equals the number of pages in the process's address space.

Page table stored in main memory: The page table is kept in main memory. This can add overhead when processes are swapped in or out.

Working of Paging

When a process requests memory, the operating system allocates one or more page frames to the process and maps the process's logical pages to the physical page frames. When a program runs, its pages are loaded into any available frames in the physical memory.



Each program has a page table, which the operating system uses to keep track of where each page is stored in physical memory. When a program accesses data, the system uses this table to convert the program's address into a physical memory address.

Steps Involved in Paging :

- **Step 1 Divide Memory :** Logical -> Pages, Physical -> Frames .
- **Step 2 Allocate Pages :** Load pages into available frames.
- **Step 3 Page Table :** Map logical pages to physical frames.
- **Step 4 Translate Address :** Convert logical to physical address.
- **Step 5 Handle Page Fault :** Load missing pages from disk.
- **Step 6 Run Program :** CPU uses page table during execution.

The mapping from virtual to physical address is done by the Memory Management Unit (MMU) which is a hardware device and this mapping is known as the paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called frames.
- The Logical Address Space is also split into fixed-size blocks, called pages.
- Page Size = Frame Size

The address generated by the CPU is divided into:

1.Page number(p): Number of bits required to represent the pages in Logical Address Space or Page number

2.Page offset(d): Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

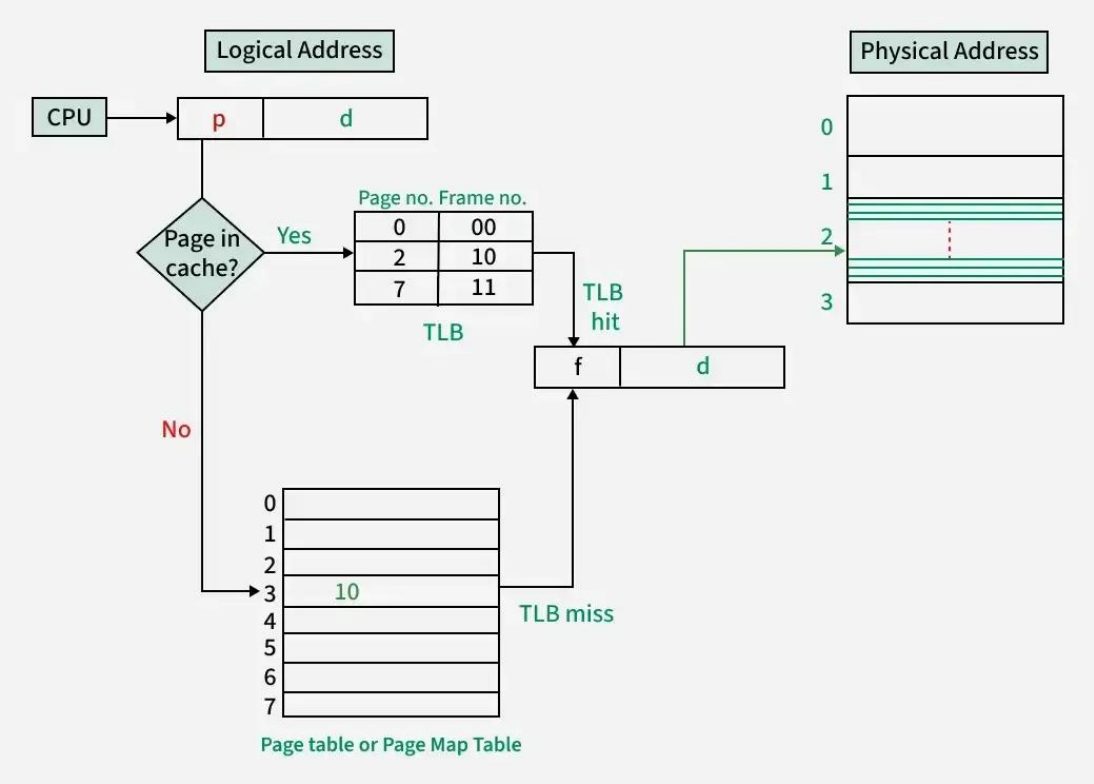
A Physical Address is divided into two main parts:

1.Frame Number(f): Number of bits required to represent the frame of Physical Address Space or Frame number

Frame Offset(d): Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Hardware implementation of Paging

The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.



The TLB is associative, high-speed memory.
Each entry in TLB consists of two parts: a tag and a value.
When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.

Advantages of Paging

- **Eliminates External Fragmentation:** Paging divides memory into fixed-size blocks (pages and frames), so processes can be loaded wherever there is free space in memory. This prevents wasted space due to fragmentation.
- **Efficient Memory Utilization:** Since pages can be placed in non-contiguous memory locations, even small free spaces can be utilized, leading to better memory allocation.
- **Supports Virtual Memory:** Paging enables the implementation of virtual memory, allowing processes to use more memory than physically available by swapping pages between RAM and secondary storage.
- **Ease of Swapping:** Individual pages can be moved between physical memory and disk (swap space) without affecting the entire process, making swapping faster and more efficient.
- **Improved Security and Isolation:** Each process works within its own set of pages, preventing one process from accessing another's memory space.

Disadvantages of Paging

- **Internal Fragmentation:** If the size of a process is not a perfect multiple of the page size, the unused space in the last page results in internal fragmentation.
- **Increased Overhead:** Maintaining the Page Table requires additional memory and processing. For large processes, the page table can grow significantly, consuming valuable memory resources.
- **Page Table Lookup Time:** Accessing memory requires translating logical addresses to physical addresses using the page table. This step increases memory access time, although TLBs can help reduce the impact.
- **I/O Overhead During Page Faults:** When a required page is not in physical memory (page fault), it needs to be fetched from secondary storage, causing delays and increased I/O operations.
- **Complexity in Implementation:** Paging requires sophisticated hardware and software support, including the MMU and algorithms for page replacement, which add complexity to the system.

Segmentation

Segmentation is a memory management technique where a process is divided into variable-sized chunks called **segments**. Unlike paging, segmentation matches the user's logical view of a program (functions, arrays, modules) to physical memory.
It reflects the user's view of memory rather than the computer's physical organization, making it easier to manage and protect processes.

Key Features of Segmentation

- **Variable-sized divisions:** Segments can have different lengths, depending on the program's requirements.
- **Logical division of memory:** Segments represent meaningful units like code, stack, data, or modules.
  Two-part address: A logical address consists of:

1. Segment number (s): Identifies which segment is being referred to.
2. Offset (d): Specifies the exact location within that segment.

Logical Address = ⟨Segment number, Offset⟩

- **Protection and sharing:** Different segments can have access rights (read, write, execute) and can be shared among processes.
- **No internal fragmentation**: Since segments are not fixed in size.
- **External fragmentation**: Can occur when free memory is divided into small scattered blocks.

Types of Segmentation in Operating Systems

Virtual Memory Segmentation: Each process is divided into a number of segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.

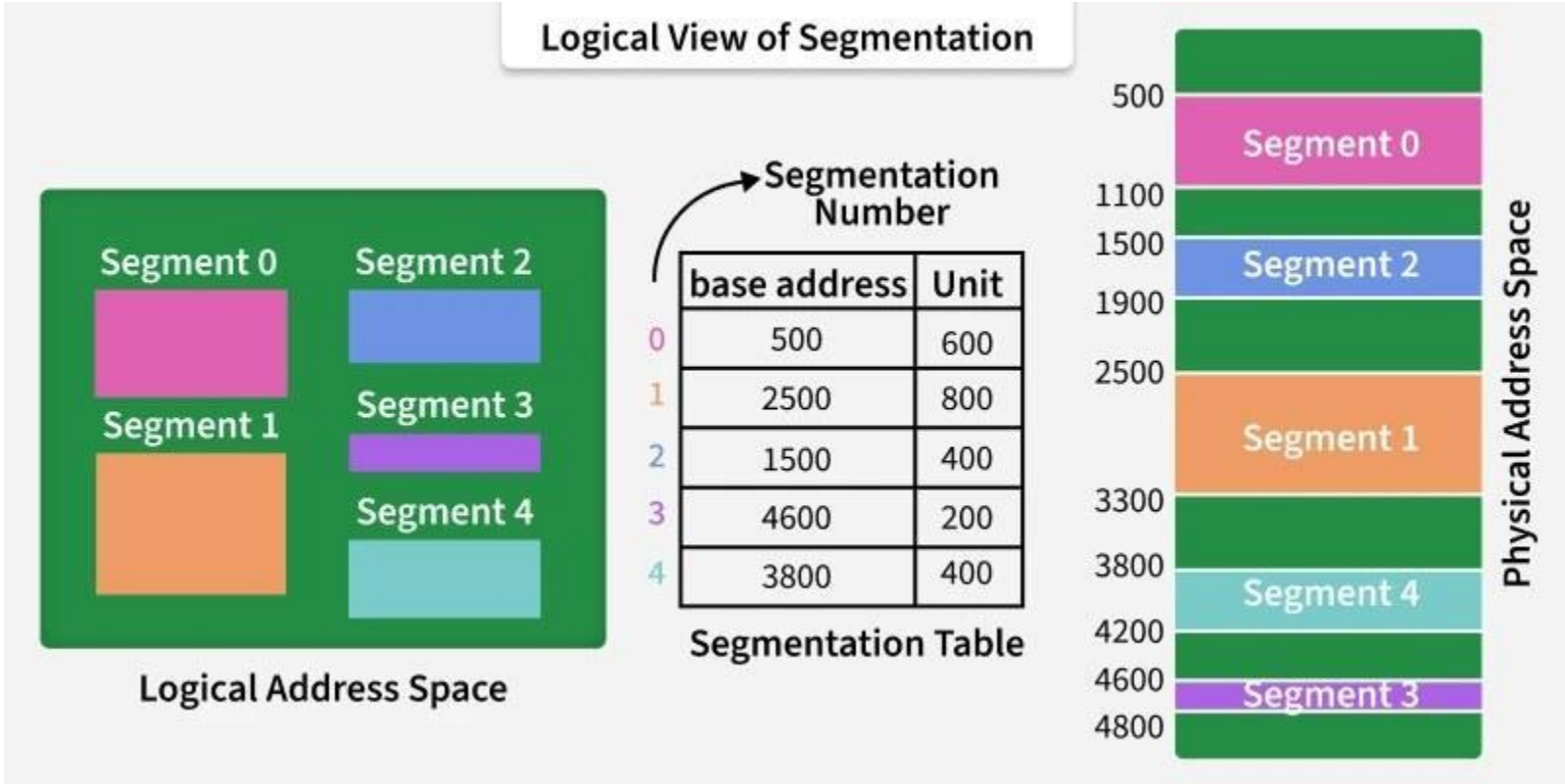
Simple Segmentation: Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

What is Segment Table?

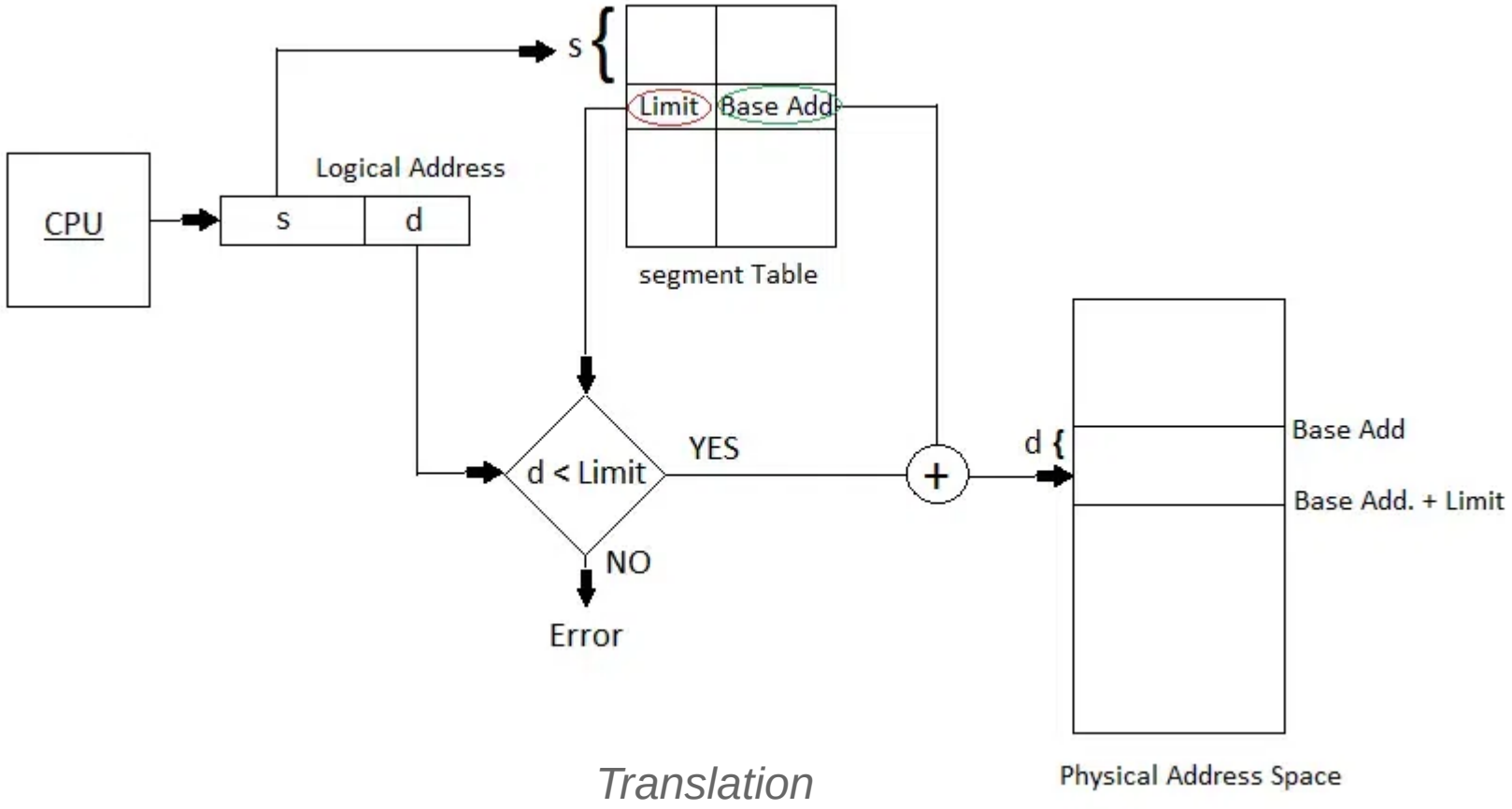
It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Segment Limit:** Also known as segment offset. It specifies the length of the segment.



Segmentation

Translation of Two-dimensional Logical Address to Dimensional Physical Address



Translation

The address generated by the CPU is divided into:

Segment number (s): Number of bits required to represent the segment.

Segment offset (d): Number of bits required to represent the position of data within a segment.

Advantages of Segmentation in Operating System

- **Reduced Internal Fragmentation:** Segments are sized as per program needs, minimizing wasted space.
- **Smaller Segment Table:** Requires less space compared to page tables.
- **Better CPU Utilization:** Entire modules are loaded at once, improving performance.
- **Closer to User's View:** Programs can be divided into logical modules, matching how users think.
- **User-Controlled Size:** Segment size is defined by the user, unlike fixed page size in paging.
- **Security & Separation:** Segments help isolate sensitive data and operations.

Disadvantages of Segmentation in Operating System

External Fragmentation: Free memory gets scattered, leading to wasted space.

Overhead: Maintaining segment tables requires extra memory and management.

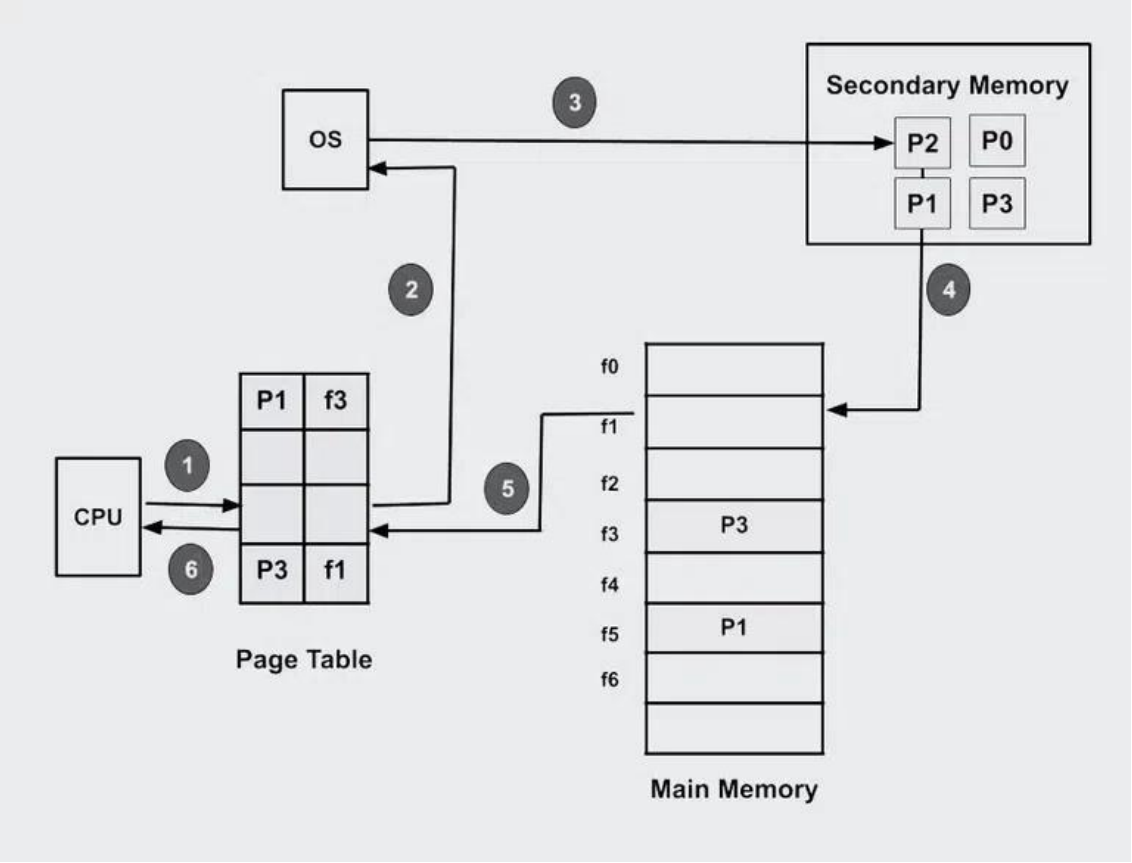
Slower Access: Two memory lookups (segment table + main memory) increase access time.

Complexity: Managing multiple variable-sized segments is harder than paging.

Segmentation Faults: Errors may occur if a program tries to access memory outside its segment.

Demand paging

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. OS loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start.



Working Process of Demand Paging

Let us understand this with the help of an example. Suppose we want to run a process P which have four pages P0, P1, P2 and P3. Currently, in the page table, we have pages P1 and P3.

The operating system's demand paging mechanism follows a few steps in its operation:

•**Program Execution:** Upon launching a program, the operating system allocates a certain amount of memory to the program and establishes a process for it.

•**Creating Page Tables:** To keep track of which program pages are currently in memory and which are on disk, the operating system makes page tables for each process.

•**Handling Page Fault:** When a program tries to access a page that isn't in memory at the moment, a page fault happens. In order to determine whether the necessary page is on disk, the operating system pauses the application and consults the page tables.

•**Page Fetch:** The operating system loads the necessary page into memory by retrieving it from the disk if it is there.

•The page's new location in memory is then reflected in the page table.

•**Resuming The Program:** The operating system picks up where it left off when the necessary pages are loaded into memory.

•**Page Replacement:** If there is not enough free memory to hold all the pages a program needs, the operating system may need to replace one or more pages currently in memory with pages currently in memory. on the disk. The page replacement algorithm used by the operating system determines which pages are selected for replacement.

•**Page Cleanup:** When a process terminates, the operating system frees the memory allocated to the process and cleans up the corresponding entries in the page tables.

Algorithms Used for Page Replacement in OS

1. First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

2. Optimal Page Replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

3. Least Recently Used

In this algorithm, page will be replaced which is least recently used.

4. Most Recently Used (MRU)

In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

Comparative study of memory management in Windows, Linux and Android OS

Feature / OS	Windows	Linux	Android
Memory Model	Virtual memory-based	Virtual memory-based	Virtual memory-based (Linux kernel)
Memory Allocation	Uses Heap, Stack, and Pool allocators	malloc(), brk(), mmap()	Uses Linux methods + Dalvik/ART heap
Virtual Memory Management	Paging + Demand Paging	Paging + Demand Paging	Same as Linux + Low Memory Killer (LMK)
Page Replacement Algorithm	Modified Clock Algorithm (WS Clock)	LRU-based algorithms	Android's LMK (approximate LRU)
Swap Space	Uses pagefile.sys	Uses swap partitions or swap files	Supported but discouraged
Memory Protection	Per-process address space, DEP, ASLR	Segmentation, Paging, ASLR	ASLR, per-app sandboxing
Garbage Collection (GC)	.NET applications use GC	Manual memory management (C/C++)	Java/Kotlin apps use GC (ART/Dalvik)

Comparative study of memory management in Windows, Linux and Android OS

Feature / OS	Windows	Linux	Android
Shared Memory	Memory-mapped files, file mapping APIs	shmget, mmap, POSIX shared memory	Binder IPC, Ashmem (Android Shared Mem)
Kernel Memory Allocation	Pool-based (ExAllocatePool)	kmalloc, vmalloc, slab allocator	Same as Linux kernel
OOM Handling	Triggers OOM and writes to logs	OOM Killer terminates process	LMK or OOM Killer kills background apps
Caching & Buffering	File system cache, prefetching	Page cache, slab cache	Uses Linux caches + app cache control
Performance Optimization	SuperFetch, ReadyBoost	Swappiness, HugePages, Transparent Huge Pages (THP)	LMK tuning, memory profiling tools

🔍 Detailed Comparison

1. Memory Allocation

- **Windows**: Uses `HeapAlloc`, `VirtualAlloc`, and system pools. .NET apps use managed heap.
- **Linux**: Uses system calls like `malloc`, `mmap`, `brk`. Kernel uses `kmalloc`, `slab`.
- **Android**: Built on Linux, with additional heap management for ART/Dalvik.

2. Virtual Memory & Paging

- All three use **virtual memory** and **paging** to separate physical memory from user processes.
- **Android** uses Linux mechanisms but adds **Low Memory Killer** to aggressively reclaim memory.

3. Page Replacement Algorithms

- **Windows**: WS Clock, considers working set and aging.
- **Linux**: LRU and its variants (e.g., Clock-Pro).
- **Android**: LMK (not true paging but proactive process killing based on memory pressure).

4. Garbage Collection

- **Windows (.NET)**: Has managed runtime with generational GC.
- **Linux (C/C++)**: No automatic GC; relies on manual memory handling.
- **Android**: ART/Dalvik VMs use generational and concurrent garbage collectors.

5. Memory Protection

- All provide memory isolation via per-process address spaces.
- Use ASLR (Address Space Layout Randomization) and DEP (Data Execution Prevention) for security.

6. Shared Memory

- **Windows:** File mapping APIs.
- **Linux:** POSIX & System V shared memory (`shmget` , `mmap`).
- **Android:** Ashmem, Binder (for inter-process communication).

7. Out of Memory (OOM) Handling

- **Windows:** Logs and user-level error handling.
- **Linux:** OOM Killer selects and kills processes based on priority.
- **Android:** Low Memory Killer prioritizes user experience, kills background or less important apps.

Thank You