

# Memory Management

---

**Dr Julie M David**  
**Assistant Professor**

# Course Content:

- What is Memory management.
- What are its strategies.
- Basic hardware and Addressing Binding.
- Logical Vs Physical address space.
- Dynamic loading and Dynamic linking and libraries.
- Swapping ,Contiguous memory allocation.
- Segmentation ,Paging.

# What is Memory

- Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.
- Main memory is a repository of rapidly available information shared by the CPU and I/O devices.
- Main memory is the place where programs and information are kept when the processor is effectively utilizing them.
- Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.
- Main memory is also known as RAM (Random Access Memory). This memory is volatile. RAM loses its data when a power interruption occurs.

## What is Memory Management?

- In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes.
- The task of subdividing the memory among different processes is called Memory Management.
- Memory management is a method in the operating system to manage operations between main memory and disk during process execution

## What is address binding in the operating system?

- The Address Binding refers to the mapping of computer instructions and data to physical memory locations.
- Both logical and physical addresses are used in computer memory.
- It assigns a physical memory region to a logical pointer by mapping a physical address to a logical address known as a virtual address.
- It is also a component of computer memory management that the OS performs on behalf of applications that require

# Types of Address Binding in Operating System

- There are mainly three types of an address binding in the OS. These are as follows:
- **Compile Time Address Binding**
- **Load Time Address Binding**
- **Execution Time or Dynamic Address Binding**

# Compile Time Address Binding

- It is the first type of address binding. It occurs when the compiler is responsible for performing address binding, and the compiler interacts with the operating system to perform the address binding. In other words, when a program is executed, it allocates memory to the system code of the computer. The address binding assigns a logical address to the beginning of the memory segment to store the object code. Memory allocation is a long-term process and may only be modified by recompiling the program.

# Load Time Address Binding

- It is another type of address binding. It is done after loading the program in the memory, and it would be done by the operating system memory manager, i.e., loader. If memory allocation is specified when the program is assigned, no program in its compiled state may ever be transferred from one computer to another. Memory allocations in the executable code may already be in use by another program on the new system. In this case, the logical addresses of the program are not connected to physical addresses until it is applied and loaded into memory



# Execution Time or Dynamic Address Binding

- Execution time address binding is the most popular type of binding for scripts that aren't compiled because it only applies to variables in the program. When a variable in a program is encountered during the processing of instructions in a script, the program seeks memory space for that variable. The memory would assign the space to that variable until the program sequence finished or unless a specific instruction within the script released the memory address connected to a variable.

# Logical and Physical Address Space

- **Logical Address Space:** An address generated by the CPU is known as a “Logical Address”.
- It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.
- **Physical Address Space:** An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”.
- A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space.
- A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

# Static and Dynamic Loading

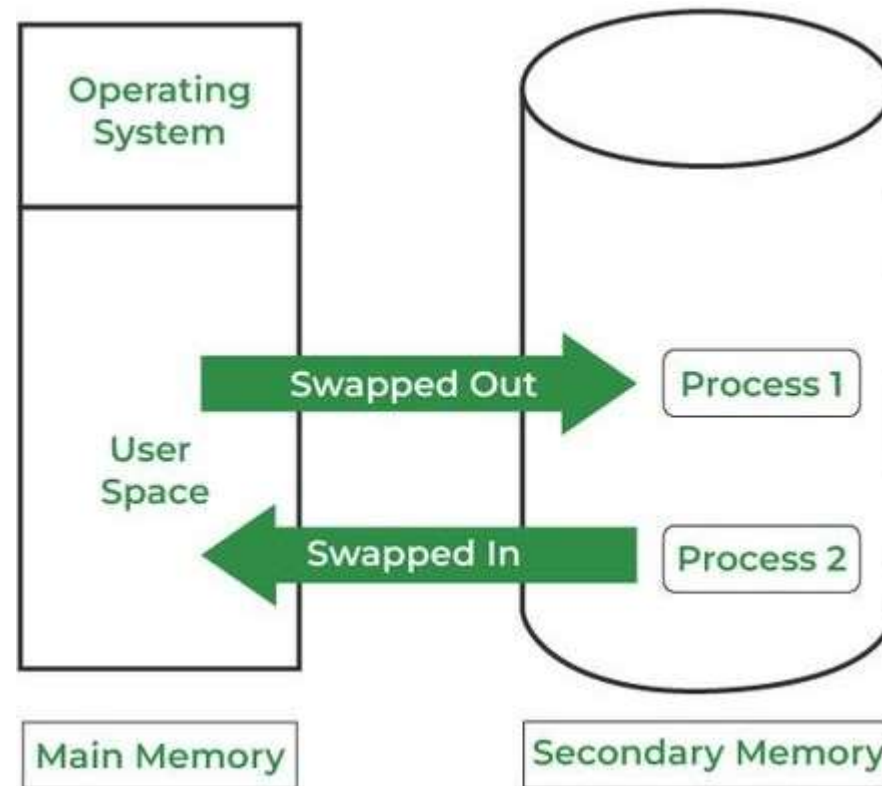
- Loading a process into the main memory is done by a loader. There are two different types of loading :
- **Static Loading:** Static Loading is basically loading the entire program into a fixed address. It requires more memory space.
- **Dynamic Loading:** The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that the unused routine is never loaded.

# Static and Dynamic Linking

- To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.
- **Static Linking:** In [static linking](#), the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
- **Dynamic Linking:** The basic concept of dynamic linking is similar to dynamic loading. In [dynamic linking](#), “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.
-

# Swapping

- When a process is executed it must have resided in memory.
- Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory.
- A swapping allows more processes to be run and can be fit into memory at one time.
- The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped.
- Swapping is also known as roll-out, or roll because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process



# Logical vs Physical Address

- An address generated by the CPU is commonly referred to as a logical address. the address seen by the memory unit is known as the physical address. The logical address can be mapped to a physical address by hardware with the help of a base register this is known as dynamic relocation of memory references.

# Contiguous Memory Allocation

- The main memory should accommodate both the operating system and the different client processes.
- Therefore, the allocation of memory becomes an important task in the operating system.
- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
- We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory.
- In adjacent memory allotment, each process is contained in a single contiguous segment of memory.





Process



Memory Blocks

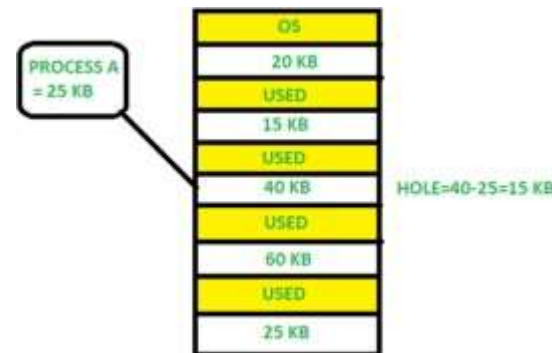
# Memory Allocation

- To gain proper memory utilization, memory allocation must be allocated efficient manner.
- One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process.
- Thus, the degree of multiprogramming is obtained by the number of partitions.

- **Multiple partition allocation:** In this method, a process is selected from the input queue and loaded into the free partition.
- When the process terminates, the partition becomes available for other processes.
- **Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory.
- This available memory is known as a “Hole”.
- When the process arrives and needs memory, we search for a hole that is large enough to store this process.
- If the requirement is fulfilled then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size  $n$  from a list of free holes.

- **First Fit**

- In the First Fit, the first available free hole fulfil the requirement of the process allocated.



- Here, in this diagram, a 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

# Best Fit

- In the Best Fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



- Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB). In this method, memory utilization is maximum as compared to other memory allocation techniques.
- **Worst Fit**
- In the Worst Fit, allocate the largest available hole to process. This method produces the largest leftover hole.

# Fragmentation

- Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole.
- These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process.
- To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems.
- In the operating systems two types of fragmentation:



- **Internal fragmentation:** Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is left over and creating an internal fragmentation problem.
- **External fragmentation:** In External Fragmentation, we have a free memory block, but we can not assign it to a process because blocks are not contiguous.

## Example Internal fragmentation.

- **Example:** Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

# External fragmentation **Example**

- **Example:** Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous. This is called external fragmentation.

- Both the first-fit and best-fit systems for memory allocation are affected by external fragmentation.
- To overcome the external fragmentation problem Compaction is used.
- In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

- Another possible solution to the external fragmentation is to allow the logical address space of the processes to be noncontiguous, thus permitting a process to be allocated physical memory wherever the latter is available.
- **Paging**
- Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous.

- Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging.
- The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory. This scheme permits the physical address space of a process to be non – contiguous.

- **Logical Address or Virtual Address (represented in bits):** An address generated by the CPU.
- **Logical Address Space or Virtual Address Space (represented in words or bytes):** The set of all logical addresses generated by a program.
- **Physical Address (represented in bits):** An address actually available on a memory unit.
- **Physical Address Space (represented in words or bytes):** The set of all physical addresses corresponding to the logical addresses.

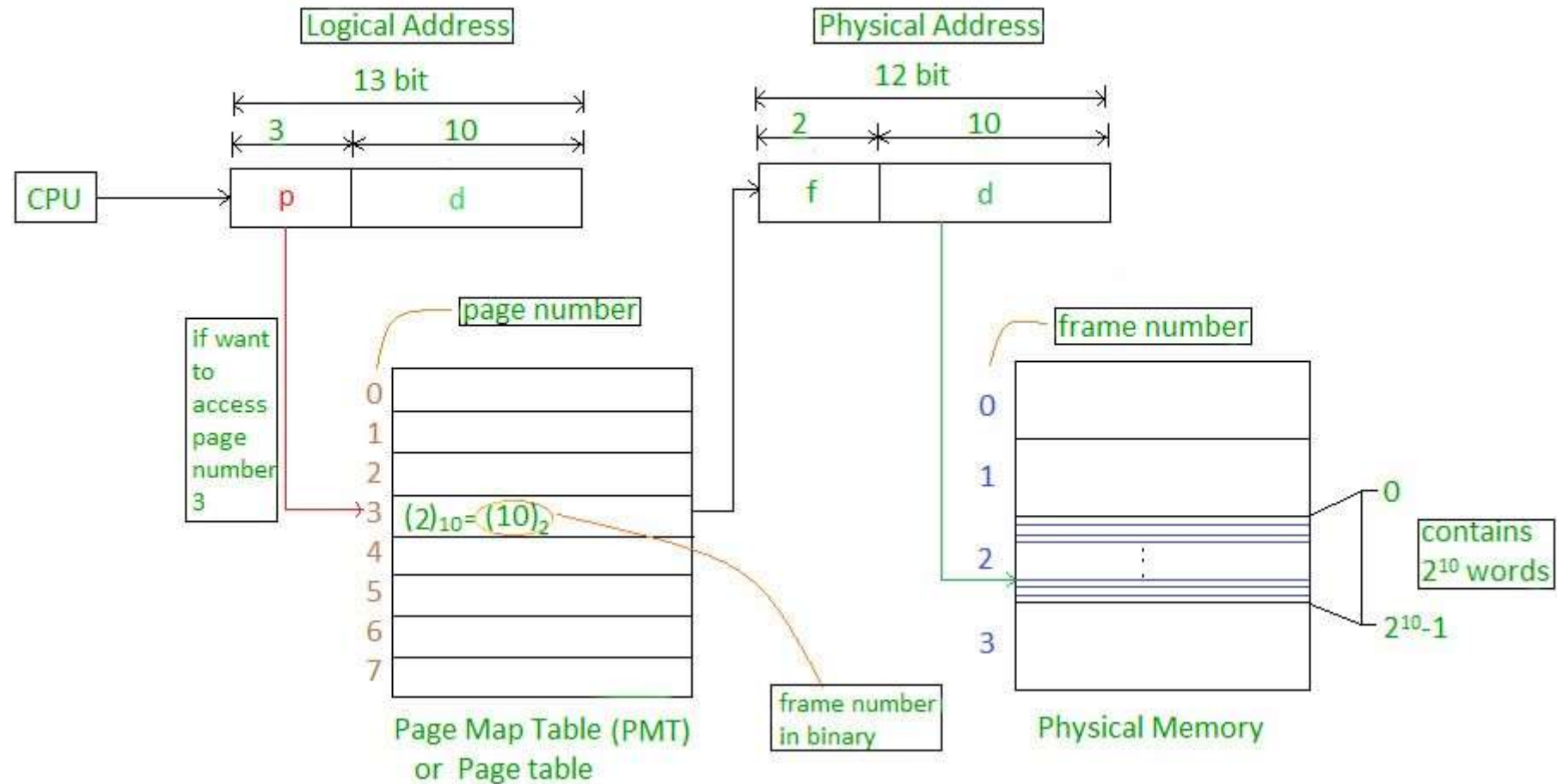
- **Example:**

- If Logical Address = 31 bits, then Logical Address Space =  $2^{31}$  words = 2 G words (1 G =  $2^{30}$ )
- If Logical Address Space = 128 M words =  $2^7 * 2^{20}$  words, then Logical Address =  $\log_2 2^{27} = 27$  bits
- If Physical Address = 22 bits, then Physical Address Space =  $2^{22}$  words = 4 M words (1 M =  $2^{20}$ )
- If Physical Address Space = 16 M words =  $2^4 * 2^{20}$  words, then Physical Address =  $\log_2 2^{24} = 24$  bits



- 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 several fixed-size blocks, called **frames**.
- The Logical Address Space is also split into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Number of frames = Physical Address Space / Frame size =  $4 \text{ K} / 1 \text{ K} = 4 = 2^2$   
 Number of pages = Logical Address Space / Page size =  $8 \text{ K} / 1 \text{ K} = 8 = 2^3$



- The address generated by the CPU is divided into:
- **Page Number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **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.
- Physical Address is divided into:
- **Frame Number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number frame
- **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.

- 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.

# Key components of the hardware that support paging

## 1. Page Table:

- The page table is a data structure maintained by the operating system and stored in memory.
- It maps virtual page numbers (VPN) to physical frame numbers (PFN).
- The page table allows for each process's virtual address space to map to the available frames in physical memory, handling non-contiguous memory allocation.

## Key components of the hardware that support paging

- **2. Page Table Base Register (PTBR)**
- The PTBR holds the physical address of the start of the page table for the current process.
- When a context switch occurs, the operating system updates the PTBR to point to the page table of the newly scheduled process

# Page Table Entries (PTEs):

Each entry in the page table corresponds to a page in the virtual address space and contains essential data:

The frame number where the page resides in physical memory.

Status bits, such as:

**Present/Absent bit:** Indicates if the page is currently loaded in memory.

**Modified bit (Dirty bit):** Shows if the page has been modified since being loaded (used during page replacement).

**Accessed bit:** Indicates if the page has been accessed recently.

**Protection bits:** Control the allowed access levels (read, write, execute).

## **Translation Lookaside Buffer (TLB):**

- The TLB is a small, fast, associative memory that caches recent translations of virtual page numbers to physical frame numbers to speed up the paging process.
- When a virtual address needs to be translated, the TLB is checked first. If the VPN is found in the TLB (a TLB hit), the translation happens quickly.
- If there's a TLB miss, the page table is accessed, and the translation is stored in the TLB for future access



- **Memory Management Unit (MMU):**
- The MMU is the hardware component responsible for handling the translation of virtual addresses to physical addresses, usually implemented as part of the CPU.
- The MMU uses the PTBR, page table, and TLB to manage address translation.
- It performs two main functions:
  - **Address Translation:** Using the VPN (from a logical address) to find the corresponding PFN and generate the physical address.
  - **Access Rights Management:** Checks the protection bits in the PTE to ensure that the requested access (read, write, or execute) is permitted.

## **Page Fault Handling Mechanism:**

- When the page table entry indicates that a page is not currently in physical memory (indicated by the Present bit being set to 0), a page fault occurs.
- The hardware triggers a page fault exception, allowing the operating system to load the required page into memory from secondary storage (like a hard disk or SSD).
- After the page is loaded, the page table is updated, and the program can continue execution.

## **Steps in Address Translation with Paging Hardware:**

**Virtual Address Formation:** The CPU generates a virtual address with a page number and an offset.

**TLB Lookup:** The MMU first checks the TLB for a mapping from the page number to a physical frame number.

If found (TLB hit), the physical address is generated immediately.

If not found (TLB miss), the MMU fetches the PTE from the page table in memory.

**Page Table Lookup:** The PTE is retrieved from the page table in memory if not cached in the TLB.

**Page Frame Determination:** The PTE provides the frame number where the page is stored in physical memory.

**Physical Address Calculation:** The frame number is combined with the page offset to form the complete physical address.

**Accessing Memory:** The physical address is sent to the main memory to retrieve or store data.

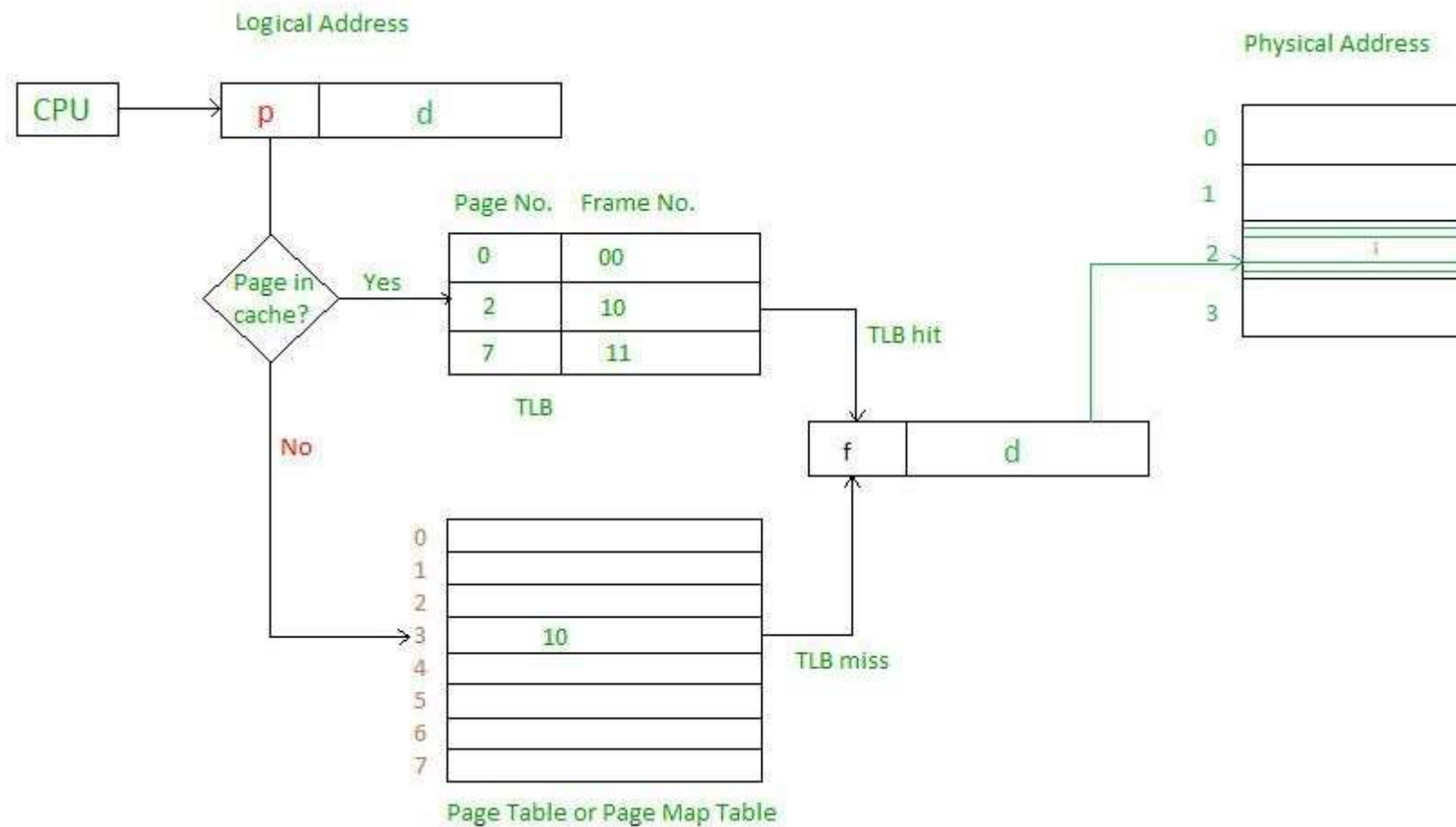
## **Optimizations in Hardware Paging:**

**Multi-level Page Tables:** Larger address spaces can be managed with multi-level page tables, reducing memory usage by creating page tables only for the parts of the address space in use.

**Inverted Page Tables:** Store one entry per physical page frame, reducing the size of the page table by mapping physical pages rather than virtual ones.

**Hardware-based Page Replacement Policies:** Some hardware implementations support page replacement policies (e.g., Least Recently Used) using accessed and modified bits.

- The TLB is an 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.



## Hit Ratio

•**Definition:** The hit ratio is the percentage of memory access requests that are successfully found in the cache.

•**Formula:** 
$$\text{Hit Ratio} = \frac{\text{Number of Cache Hits}}{\text{Total Number of Cache Accesses}} \times 100$$

•**Explanation:** When the CPU requests data or an address translation and it is already in the cache (a "hit"), the system can retrieve it quickly without needing to access slower memory or go through a more extended lookup process. A high hit ratio is desirable, as it indicates that the cache is effectively reducing the time required to access frequently used data.



## Miss Ratio

- Definition:** The miss ratio is the percentage of memory access requests that are not found in the cache and therefore require access to a slower memory tier (e.g., main memory or disk).

- Formula:** 
$$\text{Miss Ratio} = \frac{\text{Number of Cache Misses}}{\text{Total Number of Cache Accesses}} \times 100$$
$$\text{Miss Ratio} = \frac{\text{Number of Cache Misses}}{\text{Total Number of Cache Accesses}} \times 100$$

- Explanation:** When the requested data or address translation is not in the cache (a "miss"), the system must retrieve it from main memory or disk, causing a delay. A high miss ratio can indicate a less effective cache or a cache that is too small for the workload.

## Relationship Between Hit Ratio and Miss Ratio

- The hit ratio and miss ratio are complementary metrics, as they sum up to 100%:

$$\text{Hit Ratio} + \text{Miss Ratio} = 100\%$$

TLB access time =  $c$

TLB hit ratio =  $x$ , then miss ratio =  $(1-x)$

When hit occurs

Effective access time =  $\text{hit ratio} * (c+m) + \text{miss ratio} * (c+m+m)$

For page table access

for main memory access

TLB access time =  $c$

TLB hit ratio =  $x$ , then miss ratio =  $(1-x)$

When hit occurs

Effective access time =  $\text{hit ratio} * (c+m) + \text{miss ratio} * (c+m+m)$

For page table access

for main memory access

# Segmentation in Operating System

- A process is divided into Segments.
- The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments. Segmentation gives the user's view of the process which paging does not provide.
- Here the user's view is mapped to physical memory.

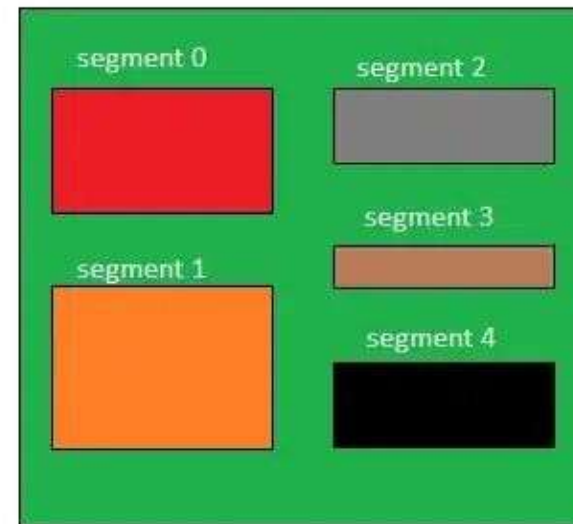
# Types of Segmentation in Operating System

- **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.

### Logical View of Segmentation

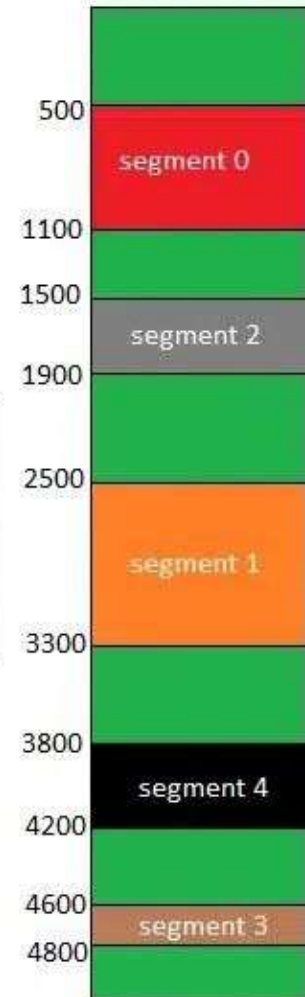


Logical Address Space

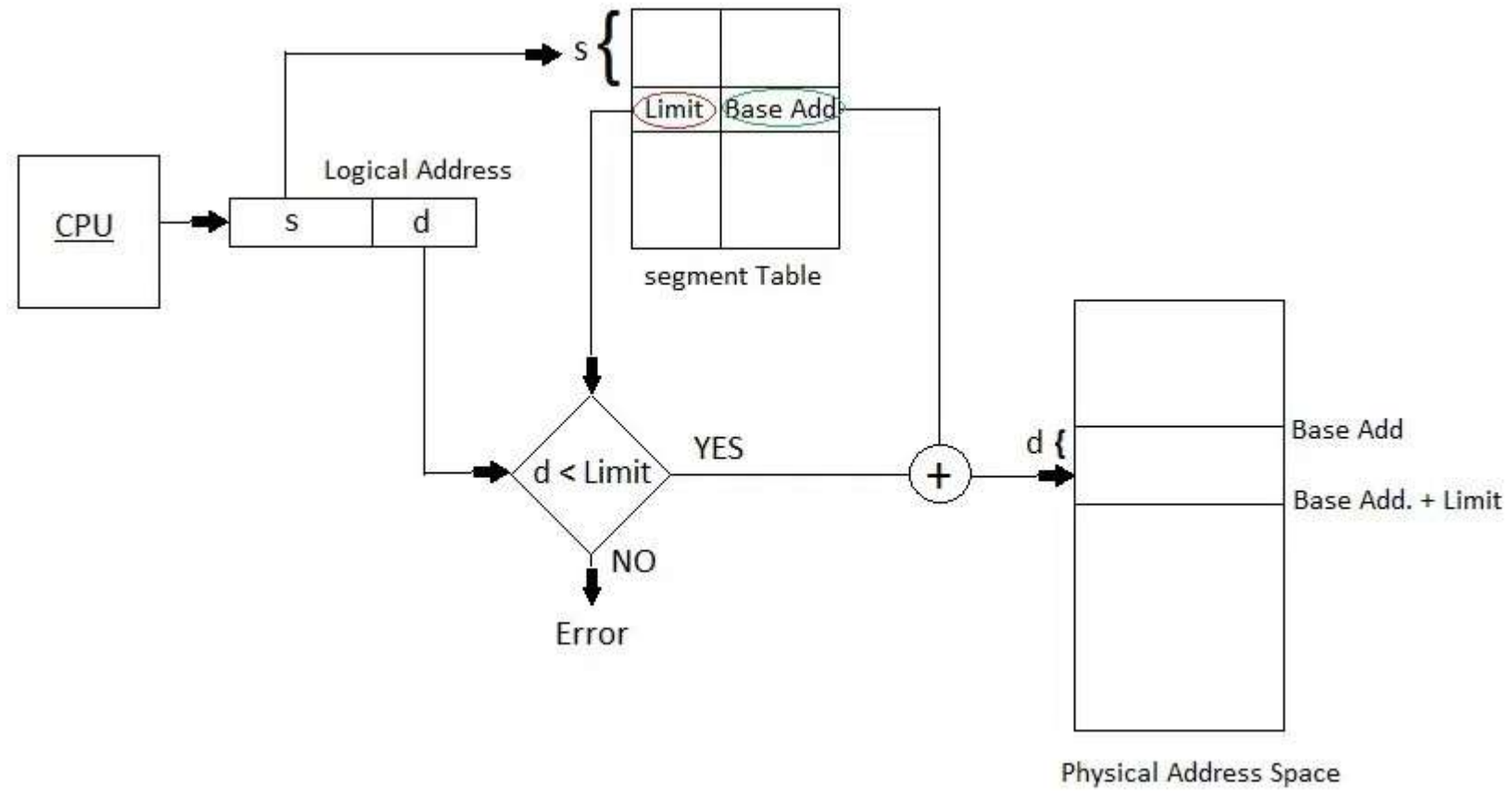
Segment Number

	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table



Physical Address Space





- 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 size of the segment.

# Advantages of Segmentation in Operating System

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.
- As a complete module is loaded all at once, segmentation improves CPU utilization.
- The user's perception of physical memory is quite similar to segmentation. Users can divide user programs into modules via segmentation. These modules are nothing more than separate processes' codes.
- The user specifies the segment size, whereas, in paging, the hardware determines the page size.
- Segmentation is a method that can be used to segregate data from security operations.
- **Flexibility:** Segmentation provides a higher degree of flexibility than paging. Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.
- **Sharing:** Segmentation allows for sharing of memory segments between processes. This can be useful for inter-process communication or for sharing code libraries.

# Page Replacement Algorithms in Operating Systems

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

**Page Fault:** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

# Page Replacement Algorithms:

**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.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

Page reference						
1, 3, 0, 3, 5, 6, 3						
1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**

when 3 comes, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.** Finally, when 3 come it is not available so it replaces 0 **1 page fault.**

# Belady's Anomaly in Page Replacement Algorithms

In Operating System, process data is loaded in fixed-sized chunks and each chunk is referred to as a page. The processor loads these pages in fixed-sized chunks of memory called frames. Typically the size of each page is always equal to the frame size.

Generally, on increasing the number of frames to a process' virtual memory, its execution becomes faster as fewer page faults occur. Sometimes the reverse happens, i.e. more page faults occur when more frames are allocated to a process. This most unexpected result is termed Belady's Anomaly.

A page fault occurs when a page is not found in the memory and needs to be loaded from the disk. If a page fault occurs and all memory frames have been already allocated, then the replacement of a page in memory is required on the request of a new page. **This is referred to as demand-paging.** The choice of which page to replace is specified by page replacement algorithms. The commonly used page replacement algorithms are FIFO, LRU, optimal page replacement algorithms, etc

**Bélády's anomaly** is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

This phenomenon is commonly experienced in the following page replacement algorithms:

First in first out (FIFO)

Second chance algorithm

Random page replacement algorithm

**Reason for Belady's Anomaly** —The other two commonly used page replacement algorithms are Optimal and LRU, but Belady's Anomaly can never occur in these algorithms for any reference string as they belong to a class of stack-based page replacement algorithms



# Optimal Page replacement

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

**Example-2:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

Page reference	7,0,1,2,0,3,0,4,2,3,0,3,2,3														No. of Page frame - 4													
7	0	1	2	0	3	0	4	2	3	0	3	2	3		7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2															
		1	1	1	1	1	4	4	4	4	4	4	4															
	0	0	0	0	0	0	0	0	0	0	0	0	0															
7	7	7	7	7	3	3	3	3	3	3	3	3	3															
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit															
Total Page Fault = 6																												

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

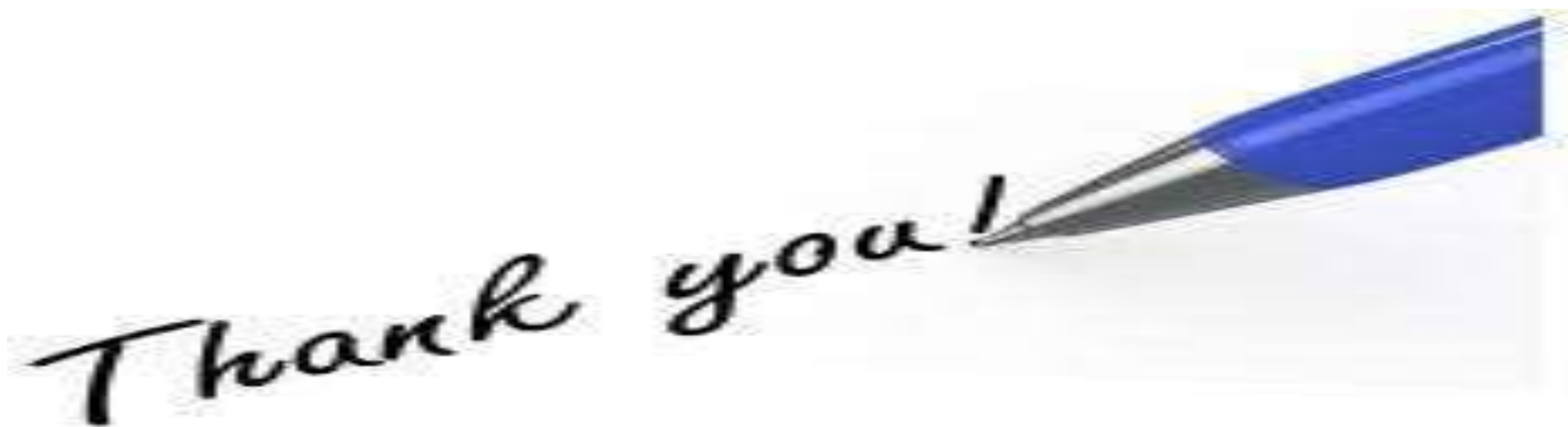
0 is already there so —> **0 Page fault**. when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault**. 0 is already there so —> **0 Page fault**. 4 will takes place of 1 —> **1 Page Fault**.

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests.

### **3. Least Recently Used**

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

A blue ballpoint pen is shown in the process of writing the words "Thank you!" in a cursive script on a white rectangular card. The pen is positioned at the end of the word "you!", with its tip touching the paper. The card is placed on a light-colored surface, and the background is white with a green vertical bar on the left and a dark blue horizontal bar at the top.

*Thank you!*