Unit-03

Memory Management and Virtual Memory

By

Balaram Das

EE & EEE Dept., SoET,

GIET University, Gunupur

Introduction

- **Memory management** is the function of an operating system which handles or manages primary memory during execution.
- It keeps track of each and every memory location, checks how much memory is to be allocated to processes, it decides which process will get memory at what time.

- A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory**.
- Virtual memory serves two purposes.
- First, it allows us to extend the use of physical memory by using disk.
- Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Logical & physical Address Space

- **Logical Address** is generated by CPU while a program is running.
- The logical address is also known as **virtual address** as it does not exist physically.
- This address is used as a reference to access the physical memory location by CPU.
- The Memory-Management Unit is used for mapping logical address to its corresponding physical address.

- **Physical Address** identifies a physical location of required data in a memory.

- The user never directly deals with the physical address but can access by its corresponding logical address for program execution.

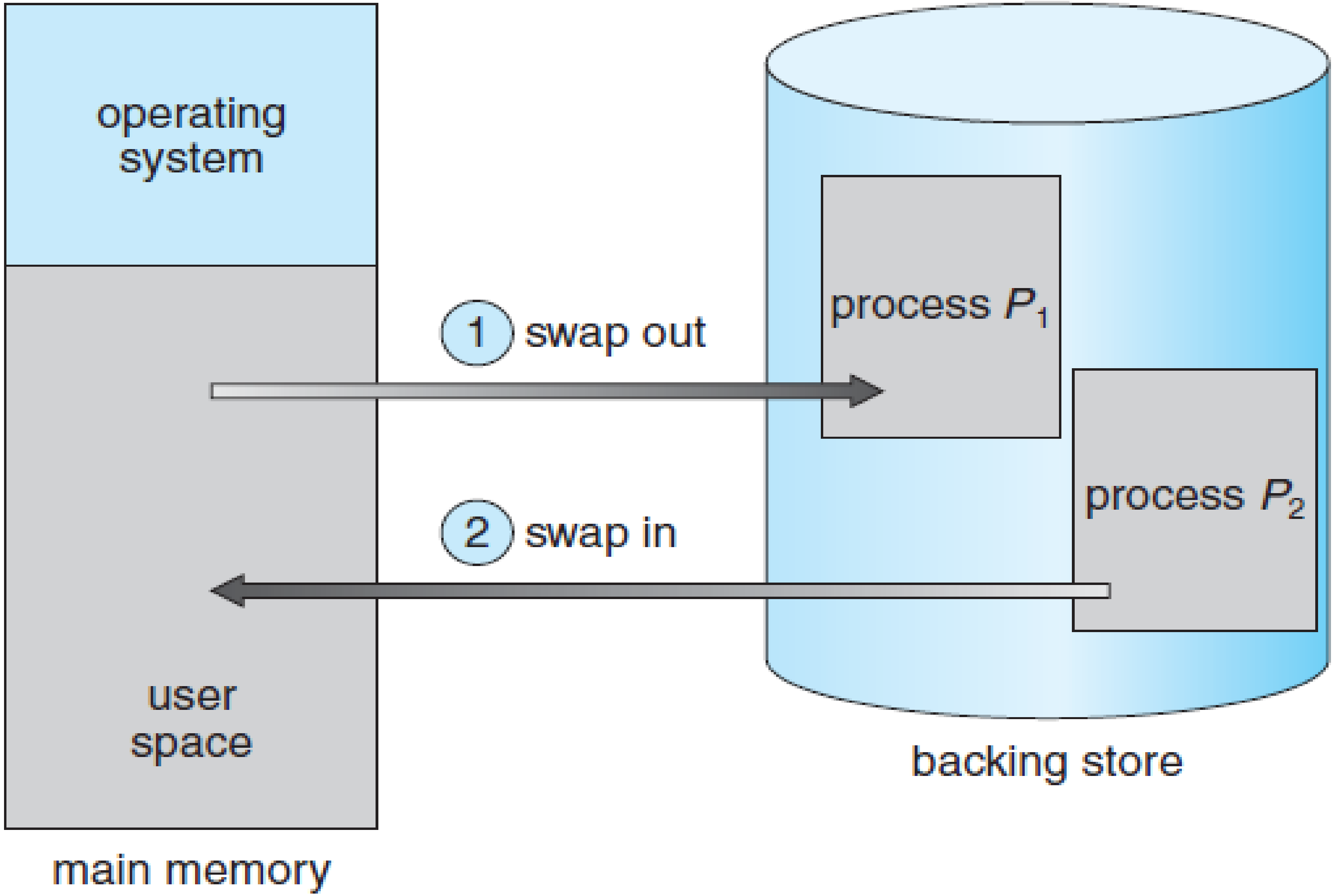
- The logical address must be mapped to the physical address by MMU before they are used.

- The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

Logical address	Physical address
The logical address is generated by CPU	The physical address is located in the memory unit
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.
User can view the logical address of a program.	User can never view physical address of program.
The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

Swapping

- Swapping is a memory management technique.
- It is used to temporarily remove the inactive programs from the main memory.
- Swapping is done so that other processes get memory for their execution.
- Due to the swapping technique **performance usually gets affected,** but it also helps in running multiple and big processes in parallel.



- *Fig.2: Swapping of two processes using a disk as a backing store.*

- A variant of the swapping technique is the **priority-based scheduling algorithm.**
- If any higher-priority process arrives and wants service, then the memory manager swaps out lower priority processes and then load the higher priority processes and then execute them.
- When the process with higher priority finishes .then the process with lower priority swapped back in and continues its execution. This variant is sometimes known as **roll in and roll out.**

- There are two more concepts that come in the swapping technique, and these are: **swap in** and **swap out.**

- The procedure by which any process gets removed from the **hard disk** and placed in the **main memory or RAM** commonly known as **Swap In.**

- On the other hand, **Swap Out** is the method of removing a process from the **main memory or RAM** and then adding it to the **Hard Disk.**

- **Advantages of Swapping**
- It helps the CPU to manage multiple processes within a single main memory.
- This technique helps to create and use virtual memory.
- With the help of this technique, the CPU can perform several tasks simultaneously. Thus, processes need not wait too long before their execution.
- This technique is economical.
- This technique can be easily applied to priority-based scheduling in order to improve its performance.

- **Disadvantages of Swapping**

- There may occur inefficiency in the case if a resource is commonly used by those processes that are participating in the swapping process.

- If the algorithm used for swapping is not good then the overall method can increase the number of page faults and thus decline the overall performance of processing.

- If the computer system loses power at the time of high swapping activity then the user might lose all the information related to the program.

Contiguous Allocation

- **Contiguous memory allocation** is a memory management technique.

- It is a method in which each process is assigned a single contiguous section of memory.

- In this memory allocation, all the available memory space remains together in one place which implies that the freely available memory partitions are not spread over here and there across the whole memory space.

- Whenever there is a request by the user process for the memory then a single section of the contiguous memory block is given to that process according to its requirement.
- The memory can be divided into 2 types:

a. **Fixed-sized partition scheme**

b. **Variable-sized partition scheme**

a. Fixed-size Partition Scheme

- It is also known as **static partitioning**.
- In this scheme, the system divides the memory into fixed-size partitions.
- The size of each partition is fixed and it cannot be changed.
- In this scheme, each partition may contain exactly one process.
- There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.
- Whenever any process terminates then the partition becomes available for another process.

- **Advantages of Fixed-size Partition Scheme**
- This scheme is simple and is easy to implement
- It supports multiprogramming.
- Management is easy using this scheme

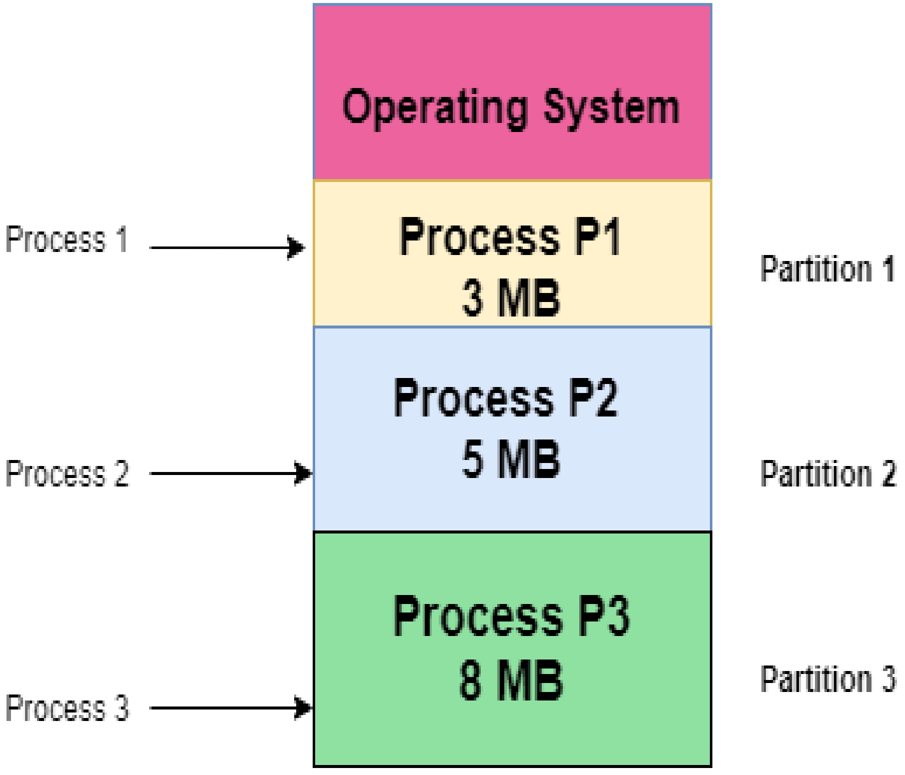
- **Disadvantages of Fixed-size Partition Scheme**

➢ **Internal Fragmentation:** Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as internal fragmentation.

➢ **Limitation on the size of the process:** If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory.

- **External Fragmentation:** Total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.
- **Degree of multiprogramming is less:** In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

- **b. Variable-size Partition Scheme**
- This scheme is also known as **dynamic partitioning.**
- The size of the partition is not declared initially.
- Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process.
- Thus the size of each partition is equal to the size of the process.
- As partition size varies according to the need of the process so in this partition scheme there is no **internal fragmentation.**



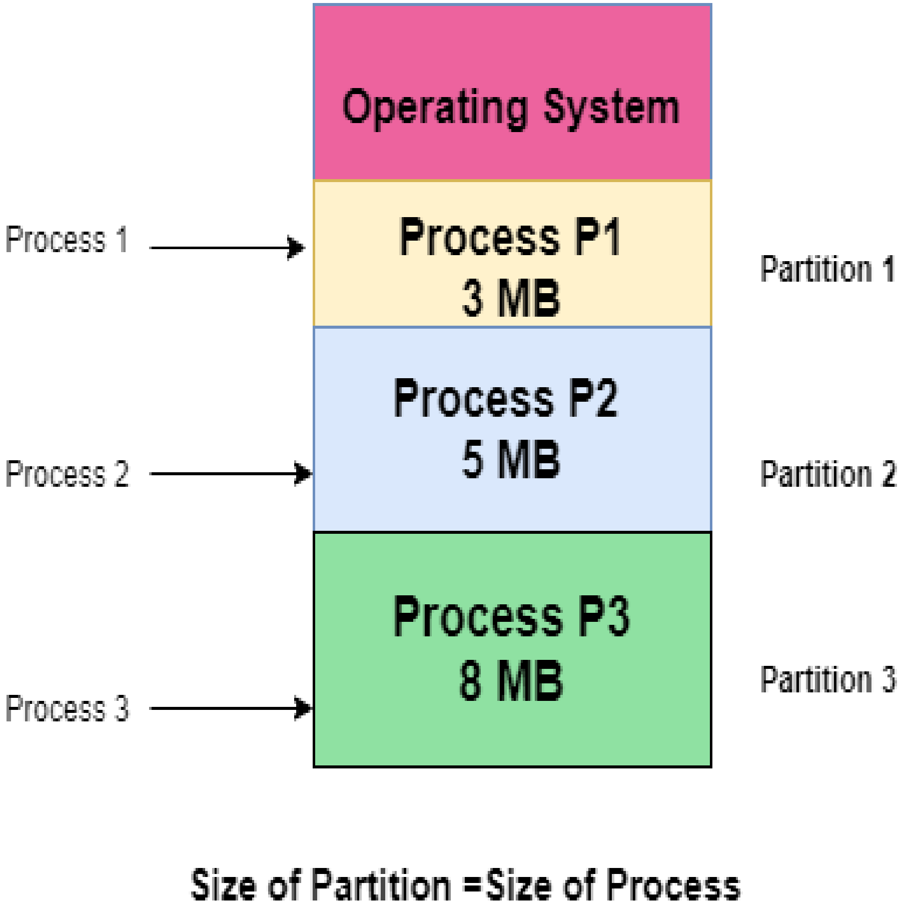
Size of Partition =Size of Process

- **Advantages of Variable-size Partition Scheme**

- **No Internal Fragmentation:** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
- **Degree of Multiprogramming is Dynamic:** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
- **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

- **Disadvantages of Variable-size Partition Scheme**

- **Results in External Fragmentation**: In the above diagram- process P1 (3MB) and process P3 (8MB) completed their execution.
- Hence there are two spaces left.
- Let’s there is a Process P4 of size 15 MB comes.
- But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation.
- Thus it results in External Fragmentation.



Disadv. Cont.

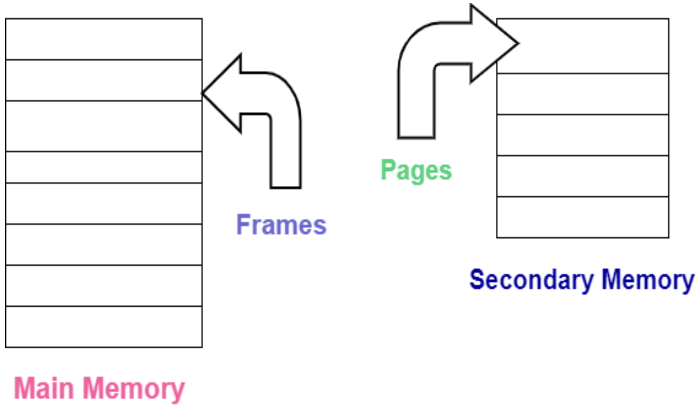
- **Difficult Implementation:** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration.

Paging

Introduction

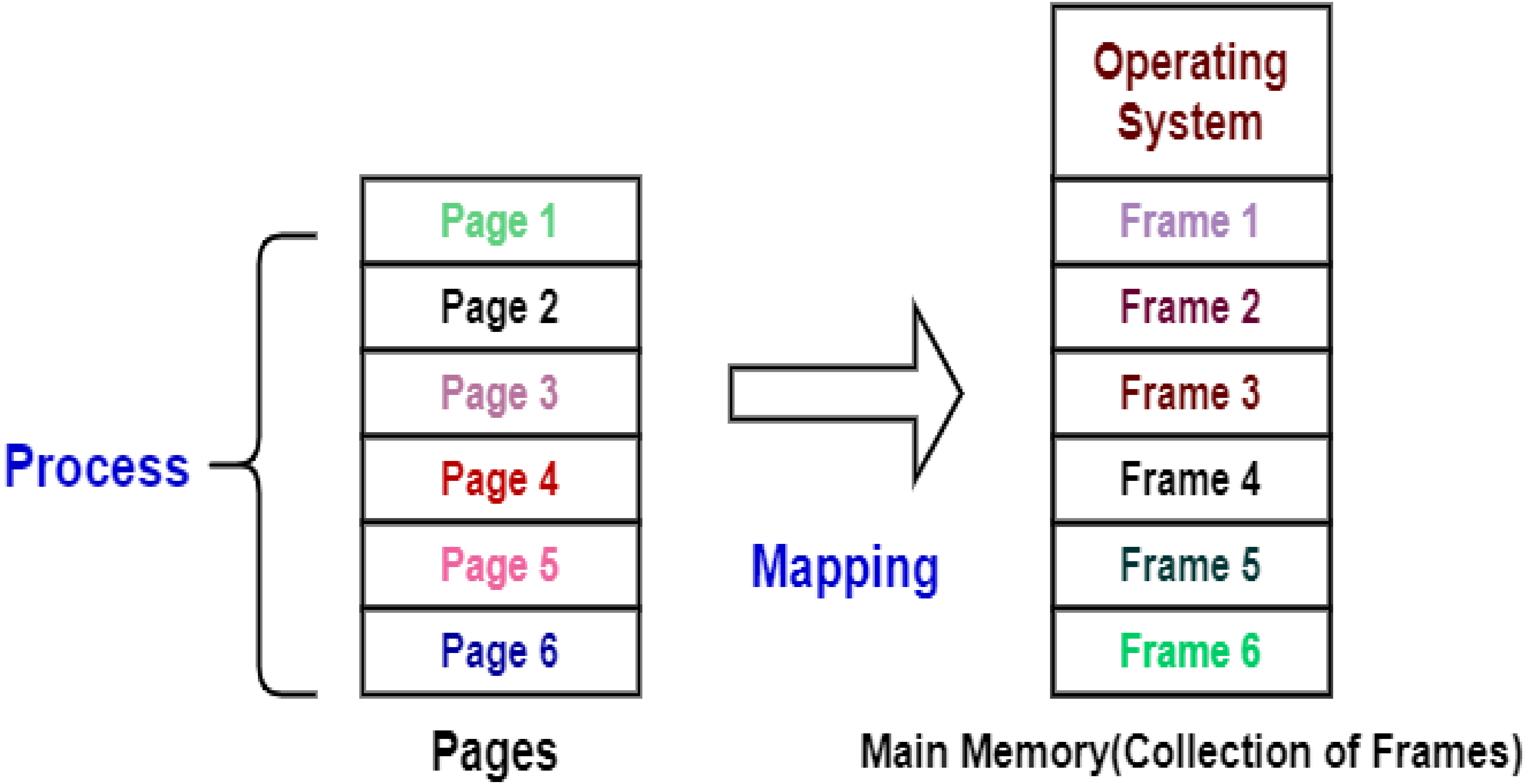
- It is a memory management scheme that permits the **physical address space** of a process to be **non-contiguous.**
- It is a **fixed-size partitioning scheme**.
- In the Paging technique, the secondary memory and main memory are divided into equal fixed-size partitions.
- Paging helps to **avoid external fragmentation** and the **need for compaction**.

Basic Method of Paging

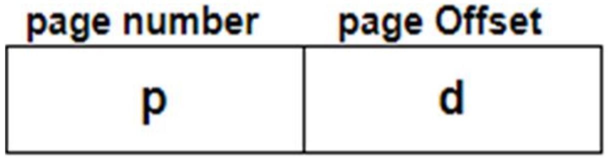


- The paging technique divides the physical memory(main memory) into fixed-size blocks that are known as **Frames** and divide the **logical memory(secondary memory) into blocks of the same size** that are known as **Pages.**
- This technique keeps the track of all the free frames.
- **The Frame has the same size as that of a Page.**
- A **frame is** basically **a place where a** (logical) **page can be** (physically) **placed.**

- Pages of a process are brought into the main memory only when there is a requirement otherwise they reside in the secondary storage.
- One page of a process is mainly stored in one of the frames of the memory.
- Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames.


Process
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Pages
Mapping
Operating System
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Main Memory(Collection of Frames)

- **Translation of Logical Address into Physical Address**
- The CPU always generates a logical address.
- The **logical address consists of two parts:**



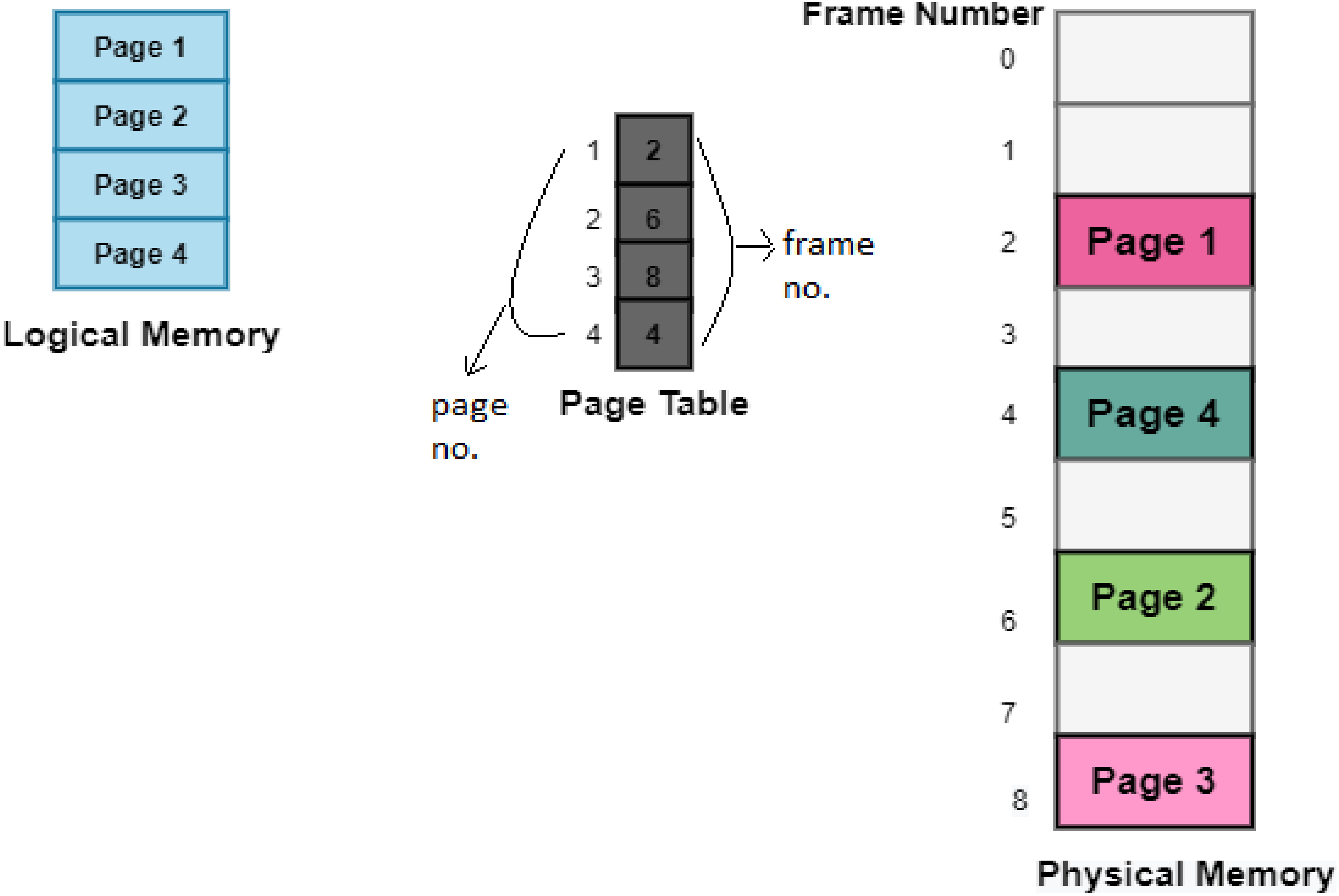
page number	page Offset
p	d

- **(i) Page Number(p):** is used to specify the specific page of the process from which the CPU wants to read the data and it is also used as an index to the page table.
- **(ii) Page offset(d):** is mainly used to specify the specific word on the page that the CPU wants to read or denotes displacement within the page.
- In order to access the main memory always a physical address is needed.

Page Table in OS

- The Page table mainly **contains the base address of each page** in the Physical memory.
- **Physical memory address =** frame number (or, base address of the frame) +   page offset
- The **Frame number(f)** is used to indicate the specific frame where the required page is stored.
- The size of the page is typically the power of 2 that varies between 512 bytes and 16 MB per page.

page size = frame size

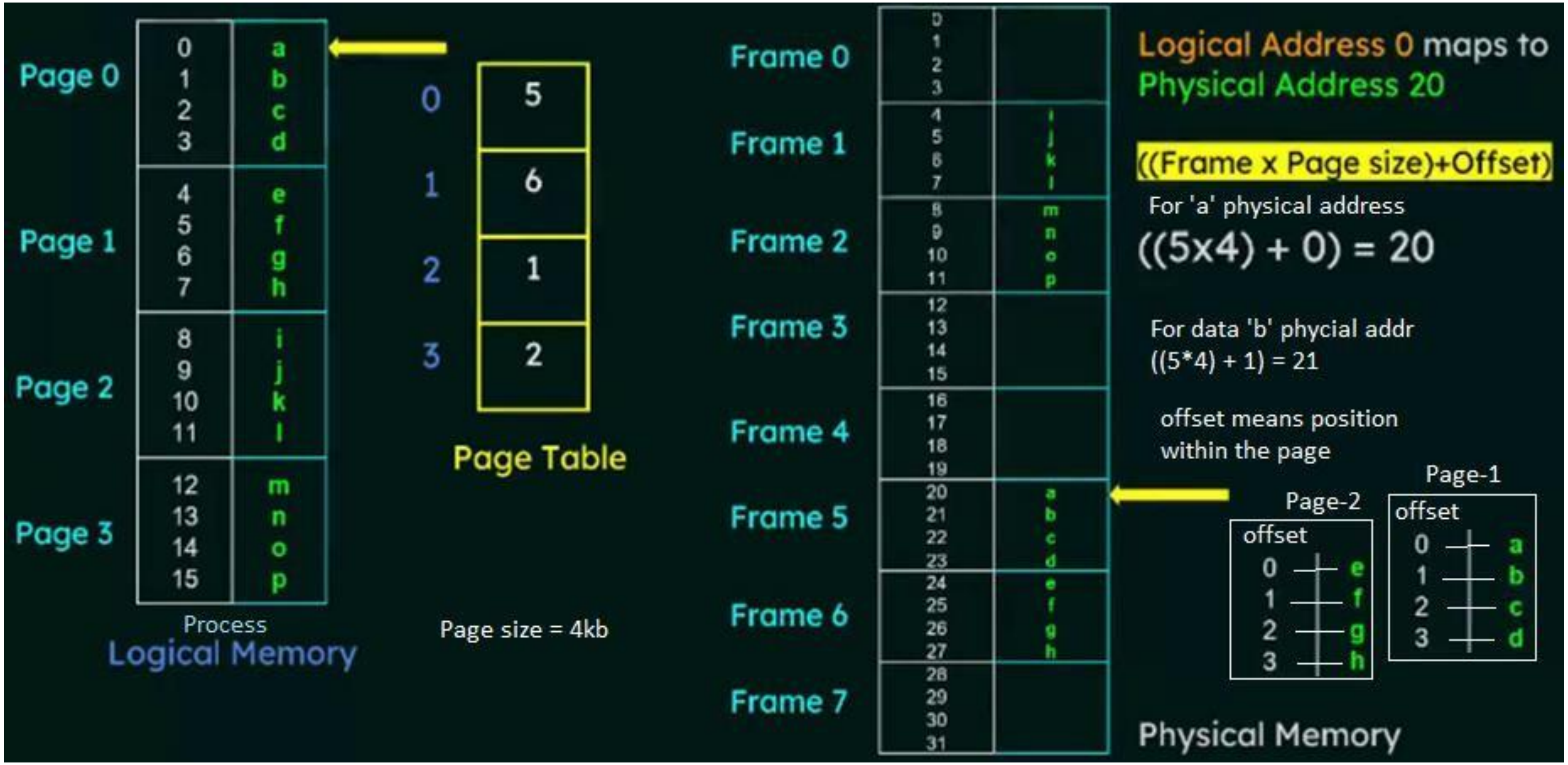


Translation of Logical address to Physical address using page table:-

- Every address generated by CPU is divided into two parts: A **page number (p)** & **page offset (d)**

Page number	Page offset
P	d

- **Each page 4bytes of data**



Logical Address 0 maps to Physical Address 20

((Frame x Page size)+Offset)

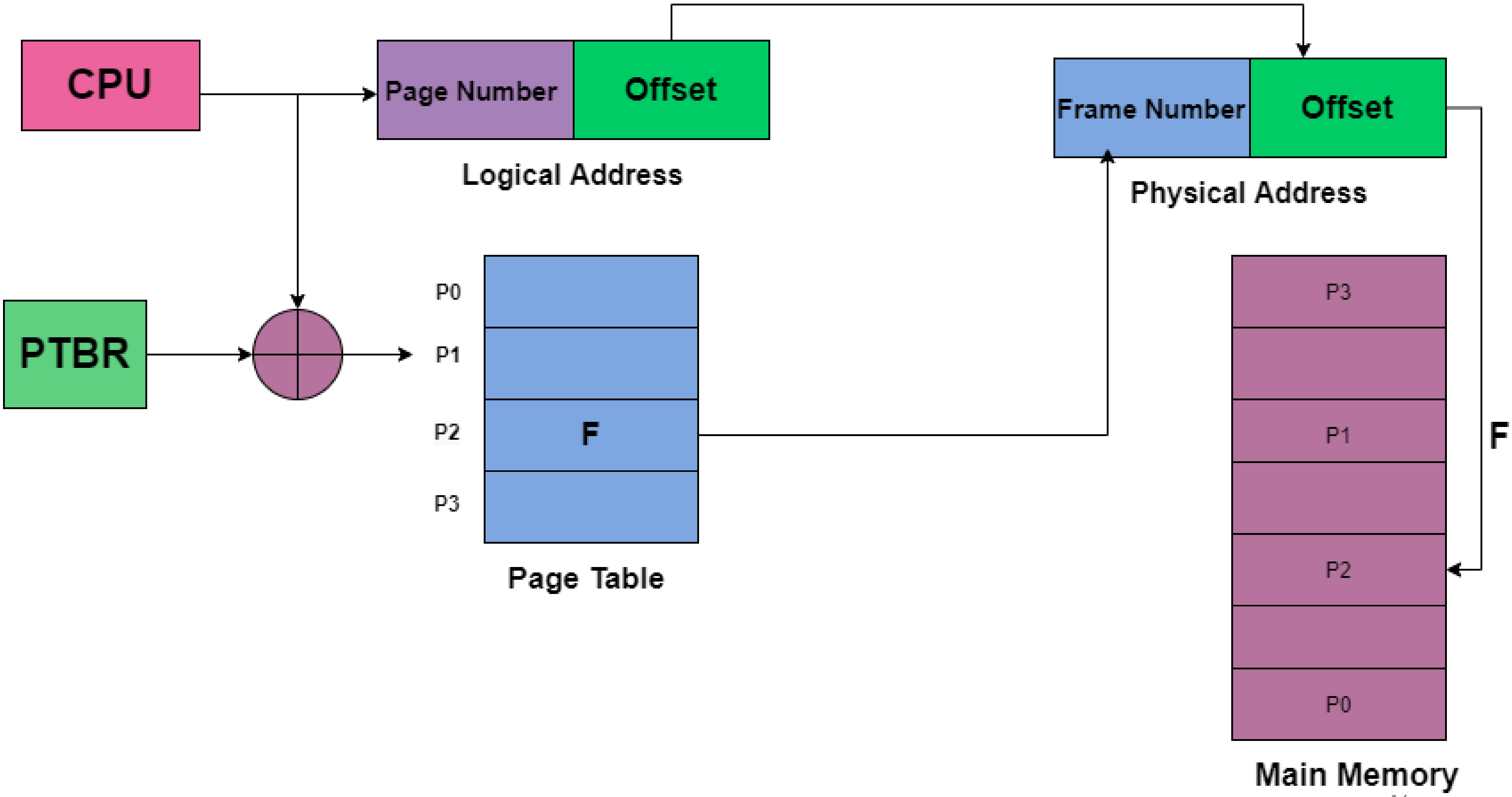
For 'a' physical address

((5x4) + 0) = 20

For data 'b' phycial addr

((5*4) + 1) = 21

offset means position within the page


CPU
Page Number
Offset
Logical Address
Frame Number
Offset
Physical Address
PTBR
P0
P1
P2
F
P3
Page Table
P3
P1
F
P2
P0
Main Memory

- The **Page Table Base Register (PTBR)** basically holds the base address for the page table of the current process.
- The **PTBR** is mainly a processor register and is managed by the operating system. Commonly, each process running on a processor needs its own logical address space.
- But there is a problem with this approach and that is with the time required to access a user memory location.

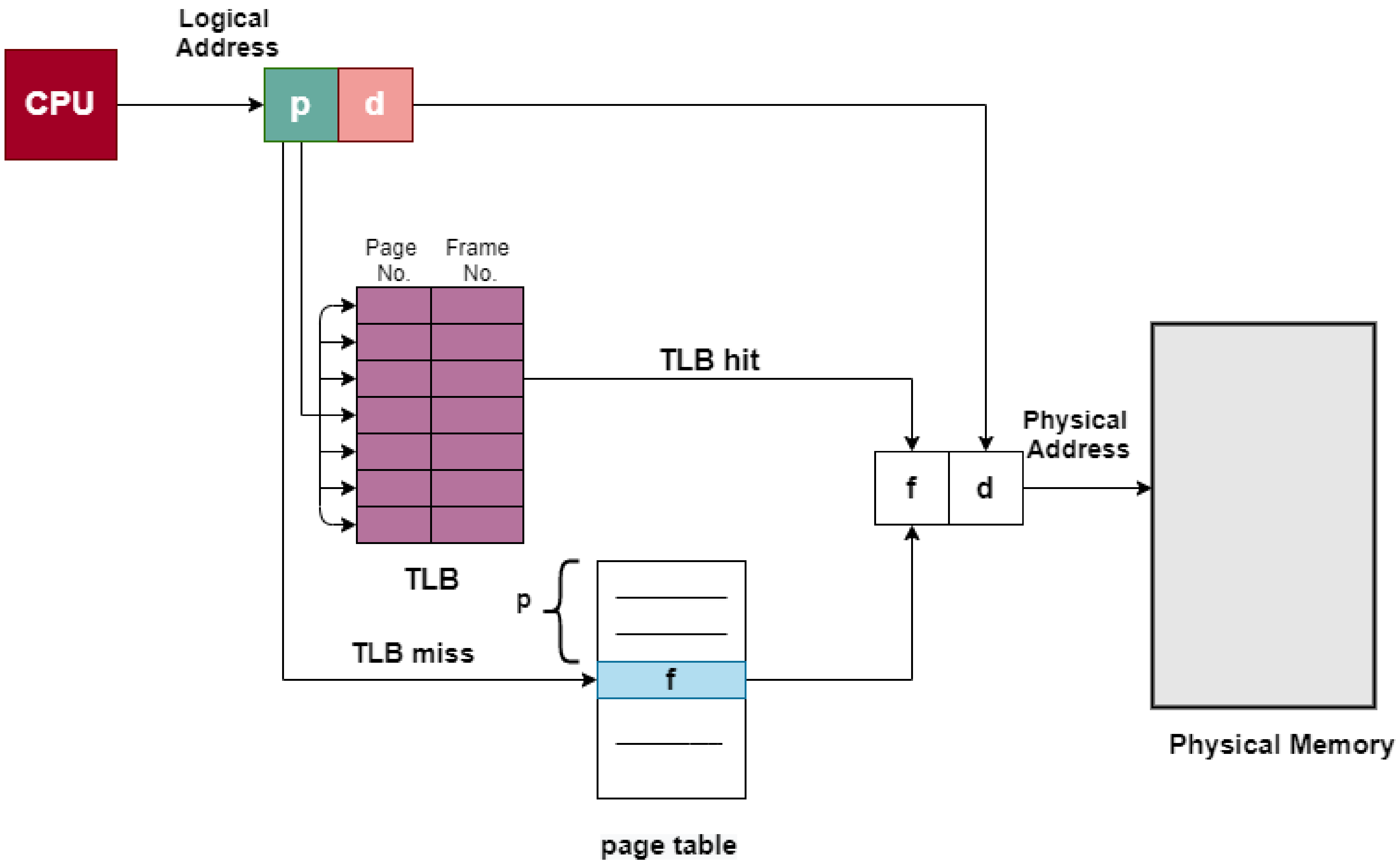
- **Translation of look-aside buffer(TLB)**
- There is the standard solution for the above problem that is to use a special, small, and fast-lookup hardware cache that is commonly known as Translation of look-aside buffer(TLB).
- TLB is associative and high-speed memory.
- Each entry in the TLB mainly consists of two parts: a key(that is the tag) and a value.
- .

- When associative memory is presented with an item, then the item is compared with all keys simultaneously. In case if the item is found then the corresponding value is returned.
- The search with TLB is fast though the hardware is expensive.
- The number of entries in the TLB is small and generally lies in between 64 and 1024

- **TLB** is used with **Page Tables** in the following ways:
- The TLB contains only a few of the page-table entries. Whenever the logical address is generated by the CPU then its page number is presented to the TLB.
- If the page number is found, then its frame number is immediately available and is used in order to access the memory. The above whole task may take less than 10 percent longer than would if an unmapped memory reference were used.
- In case if the page number is not in the TLB, then a memory reference to the Page table must be made.

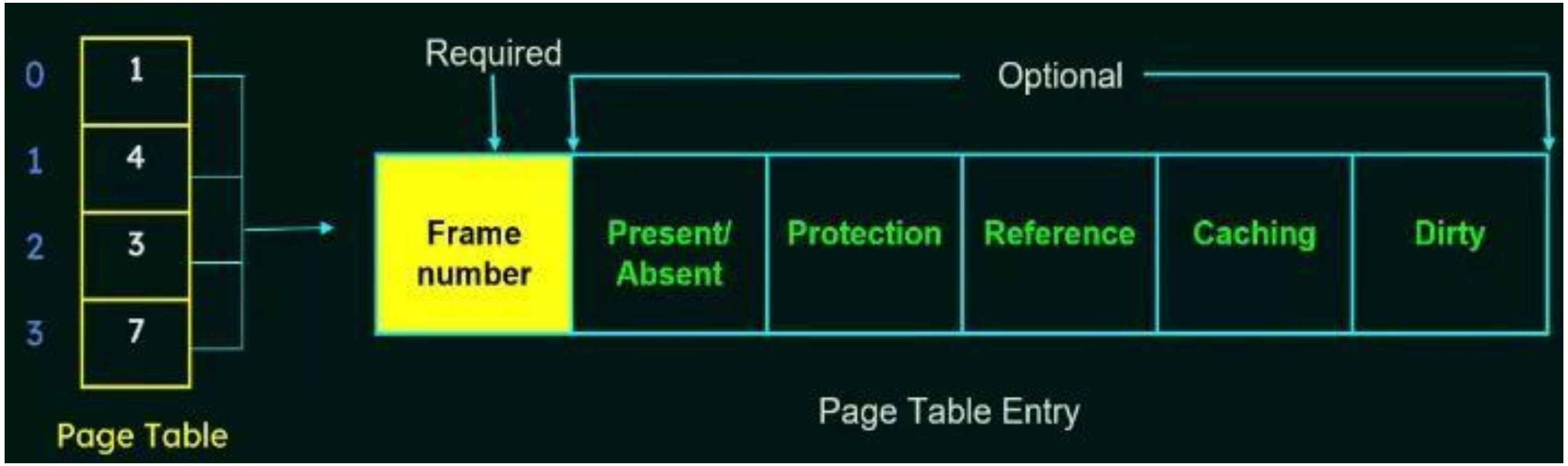
- When the frame number is obtained it can be used to access the memory. Additionally, page number and frame number is added to the TLB so that they will be found quickly on the next reference.
- In case if the **TLB is already full of entries** then the Operating system must select o**ne for replacement.**
- TLB allows some entries to **be wired down**, which means they cannot be removed from the TLB. Typically TLB entries for the kernel code are **wired down.**

Paging Hardware With TLB



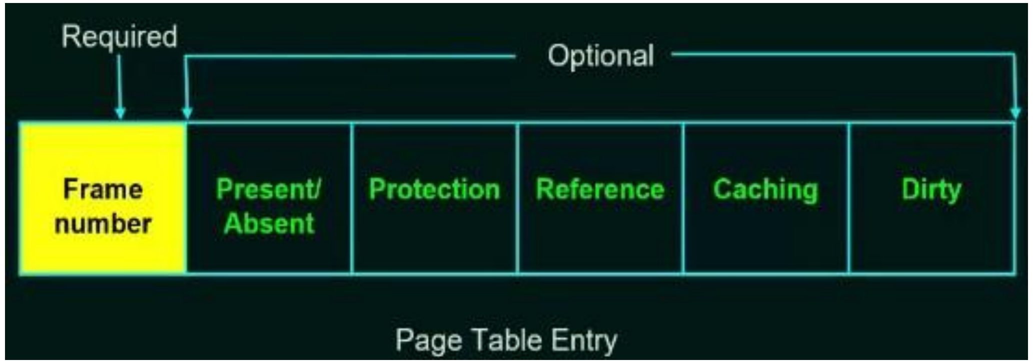
Page Table Entries:-

- Page table has entries where each page table entry stores a frame number and optional status (like protection) bits.



- **Frame number** specifies the frame where the page is stored in the main memory.
- **Present /Absent** bit specifies whether that page is present in the main memory or not. It is also called as Valid / Invalid bit.

Page Table Entries:-



3. Protection Bit

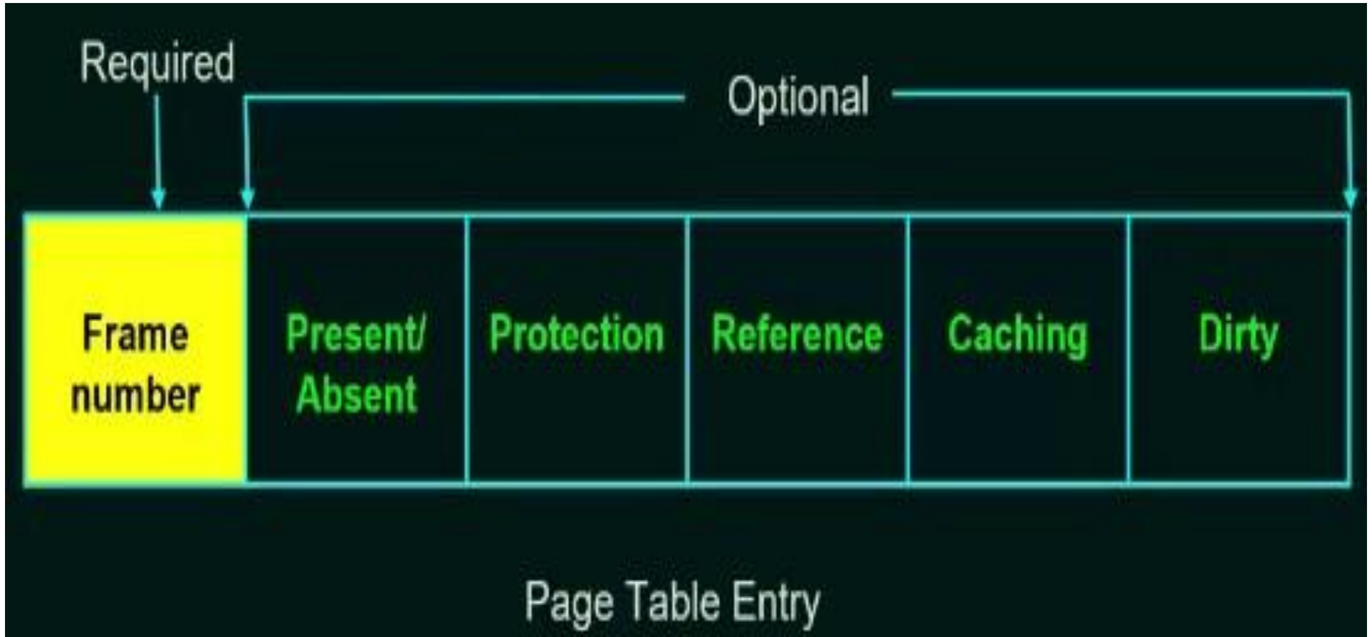
- This bit is also sometimes called as “**Read / Write bit**“ used for page protection.
- It specifies the permission to perform read and write operation on the page.
- If only read operation is allowed to be performed and no writing is allowed, then this bit is set to 0.
- If both read and write operation are allowed to be performed, then this bit is set to 1.

4. Reference Bit -

- Reference bit specifies whether that page has been referenced in the last clock cycle or not.
- If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.

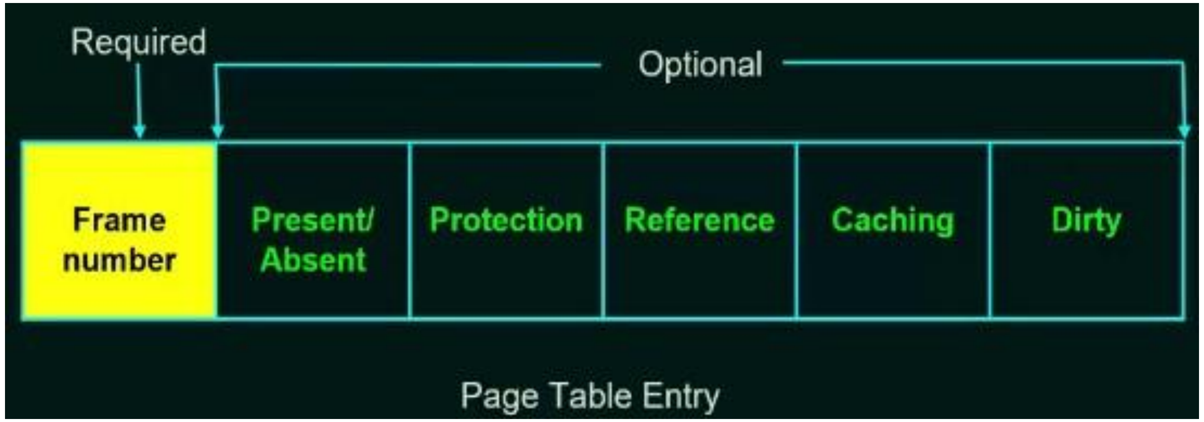
5. Caching

- The caching bit is used for enabling or disabling the caching of page.
- Whenever freshness in the data is required, then caching is disabled using this bit.
- If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.



6. Dirty Bit-

- This bit is also sometimes called as “**Modified bit**“.
- This bit specifies whether that page has been modified or not.
- If the page has been modified, then this bit is set to 1 otherwise set to 0.



Techniques used for Structuring the Page Table:-

- Some of the common techniques that are used for structuring the Page table are as follows:

1) Hierarchical Paging
2) Hashed Page Tables
3) Inverted Page Tables

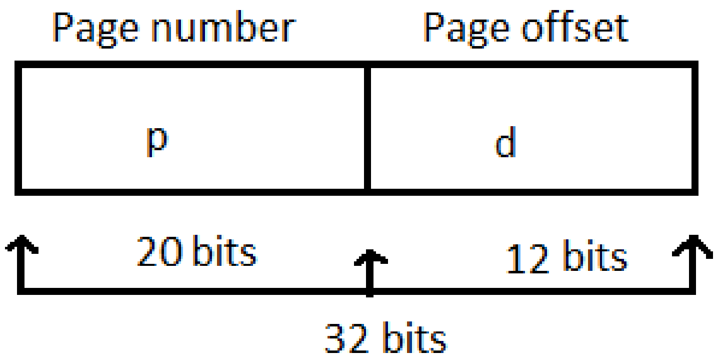
Hierarchical Paging

- **Hierarchical Paging is also known as multilevel paging.**
- **In this type of Paging the logical address space is broke up into Multiple page tables.**
- **Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.**

Two Level Page Table:-

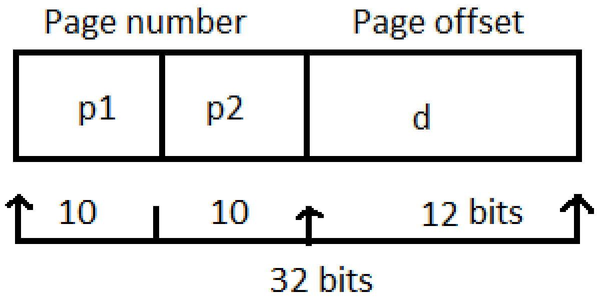
- Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:

1. Page Number consisting of 20 bits.
2. Page Offset consisting of 12 bits.

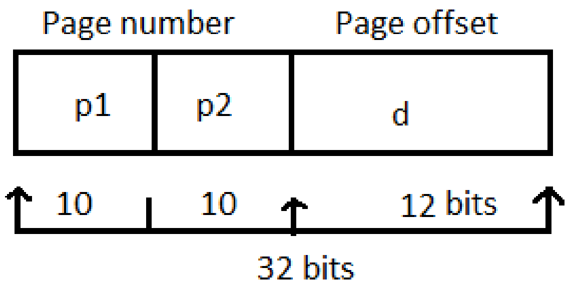


- Page number (p) denotes an index into a page table & Page offset(d) denotes displacement within the page.

- In the Page table, the page number is further divided into two:

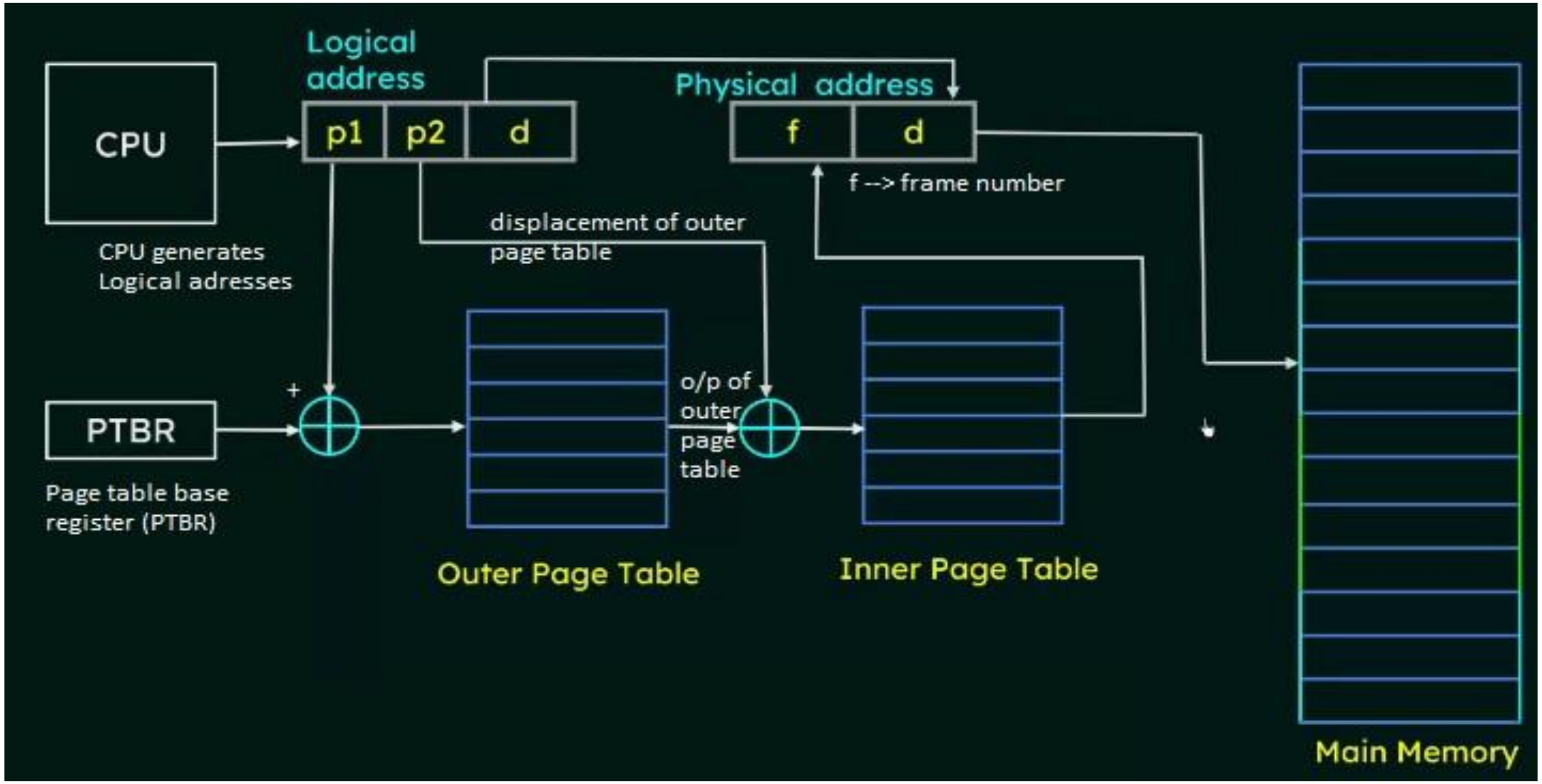


Two Level Page Table:-



- P1 is an index into the **Outer Page** table.
- P2 indicates the displacement within the page of the **outer page** Table.

Address Translation scheme for a two-level page table

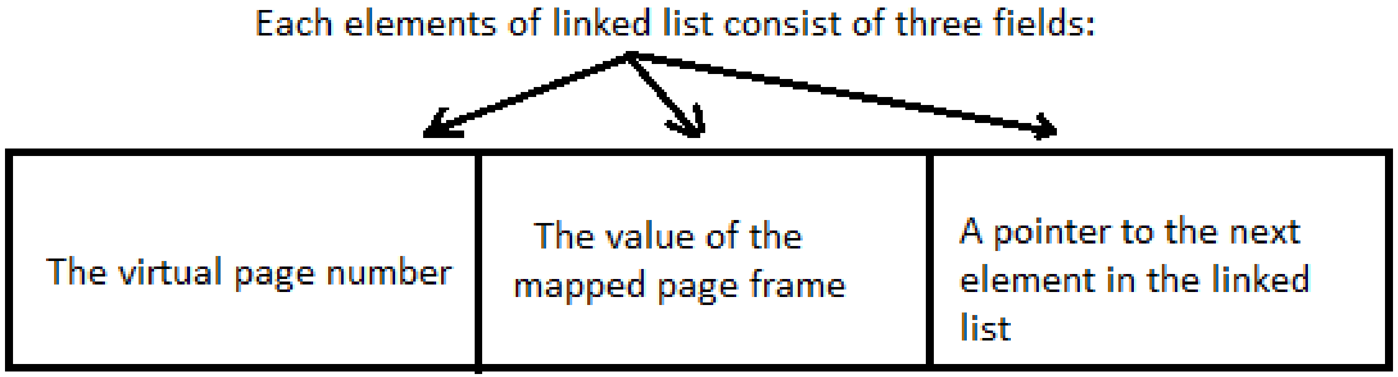


Hashed Page Tables:-

- This approach is used to handle address spaces that are **larger than 32 bits**.

Here,

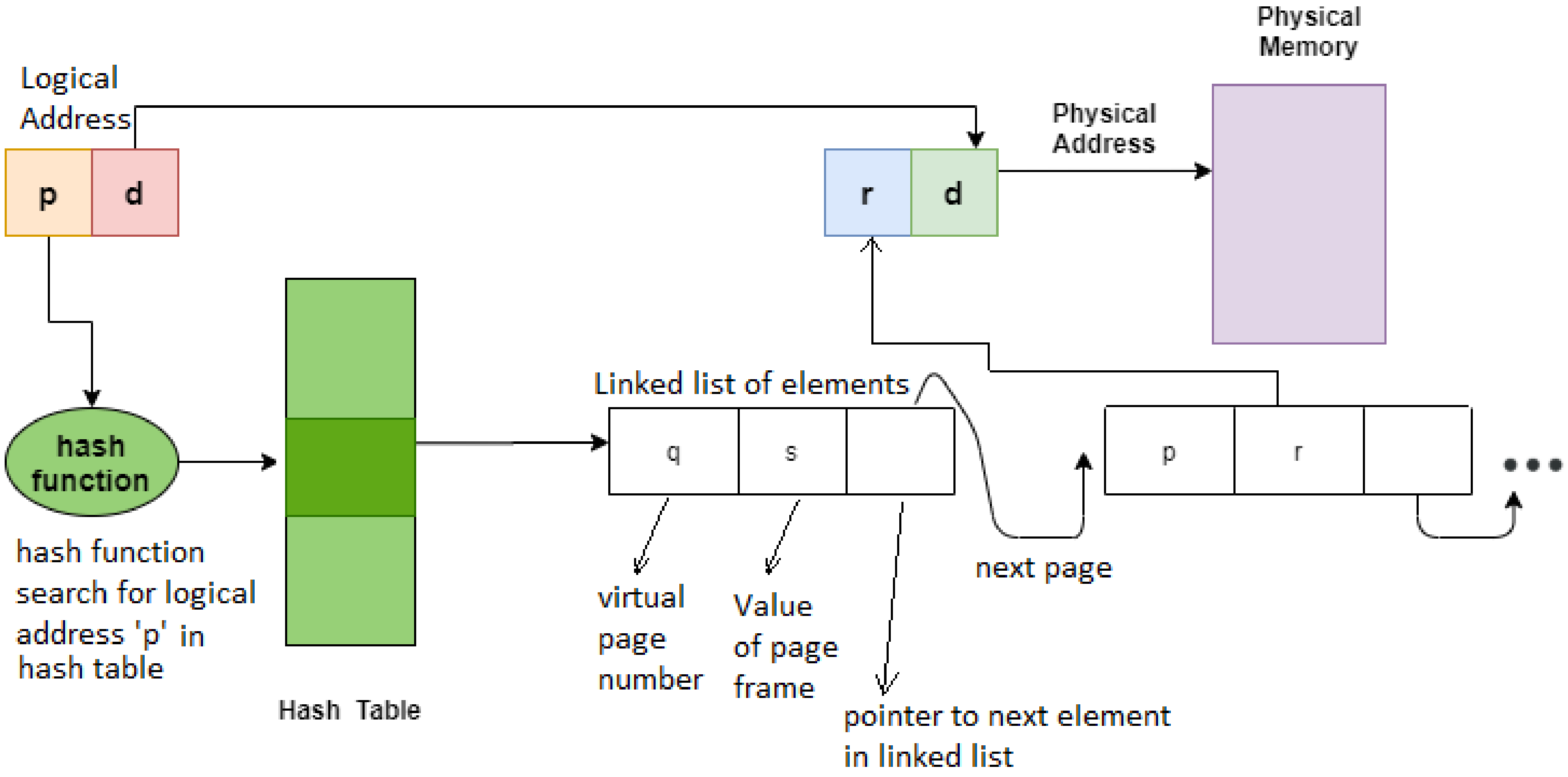
- The hash value is the virtual page number.
- Each entry in hash table contains a **linked list of elements** that hash to same location.



Each elements of linked list consist of three fields:

The virtual page number	The value of the mapped page frame	A pointer to the next element in the linked list

Address translation scheme of the Hashed Page Table:



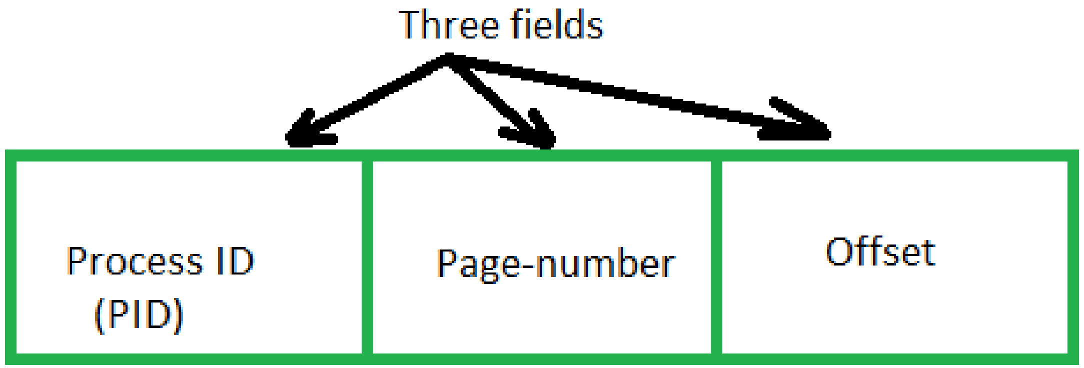
Inverted Page Tables:

- In most of the operating system, a separate page table is maintained for each processes.
- So, for ‘n’ number of processes, there will be ‘n’ number of page tables.
- For large processes, there would be many pages & for maintaining information about these pages, there would be too many entries in their page tables which itself will occupy lot of memory.
- Hence memory utilization is not efficient as lot of memory is wasted for maintaining page table itself.

- To avoid this problem, inverted page table is used.
- Only one page table in physical memory
- For each frame, it has one entry

Inverted Page Tables:

- Each virtual address in the system consist of three fields.



Three fields

Process ID (PID)

Page-number

Offset

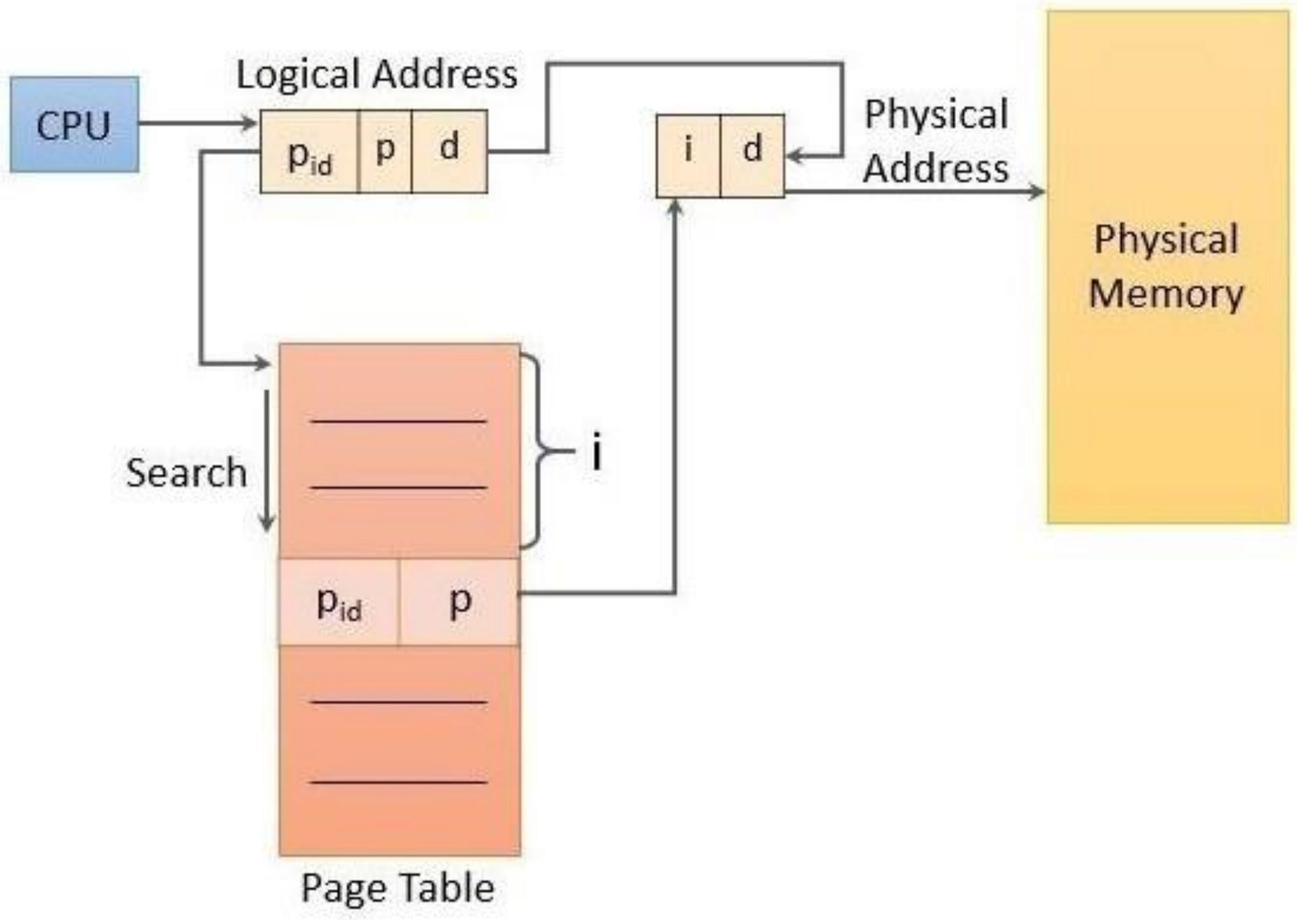
- Each inverted page-table entry is a pair of

Process ID (PID)	Page-number

Working:

- When a memory reference occurs, part of virtual address, consisting of **<process-id, page number>,** is presented to memory sub-system.
- The inverted page table is then searched for the match.
- If match is found-say at entry ‘i’, then physical address <i, offset> is generated.
- If no match is found, then an illegal address access has been attempted.

Inverted Page Tables:



- **Advantages of Paging**
- Paging mainly allows to storage of parts of a single process in a **non-contiguous fashion**.
- With the help of **Paging, the problem** of external fragmentation is solved.
- Paging is one of the simplest algorithms for memory management.

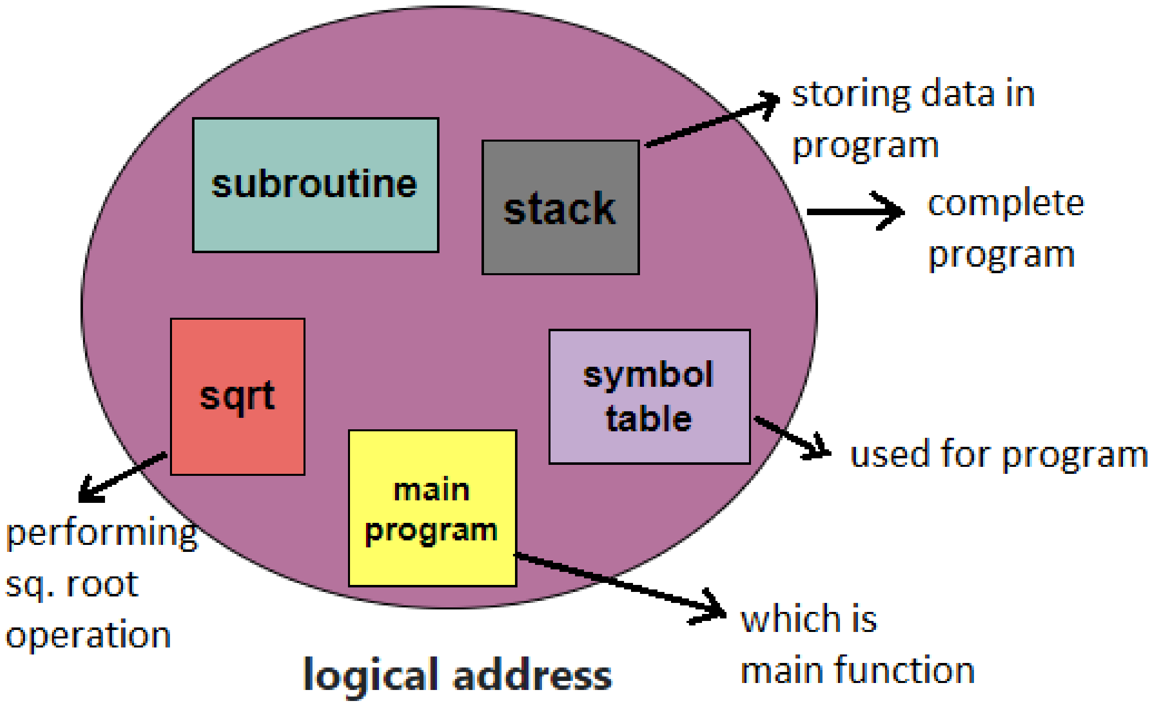
- **Disadvantages of Paging**
- Disadvantages of the Paging technique are as follows:
- In Paging, sometimes the page table consumes more memory.
- Internal fragmentation is caused by this technique.
- There is an increase in time taken to fetch the instruction since now two memory accesses are required.

Segmentation

- Segmentation is another non-contiguous memory allocation technique like paging.
- Unlike paging, in segmentation the processes are **not divided** into fixed size pages.
- Here, the processes are divided into several modules are **called segments.**
- So, here secondary memory & primary memory are divided into unequal sizes.

User's View of a Program

- Here user can understand subroutine is used for different task, sqrt module is for square root function. Etc.
- Logical address space is a collection of segments. Each segment has a name & length.



Segment Table:

- Thus the logical address consists of two tuples:

segment no.	offset

- A Table that is used to store the information of all segments of the process is commonly known as Segment Table.
- Generally, there is no simple relationship between logical addresses and physical addresses in this scheme.
- **In the segment table each entry has :**

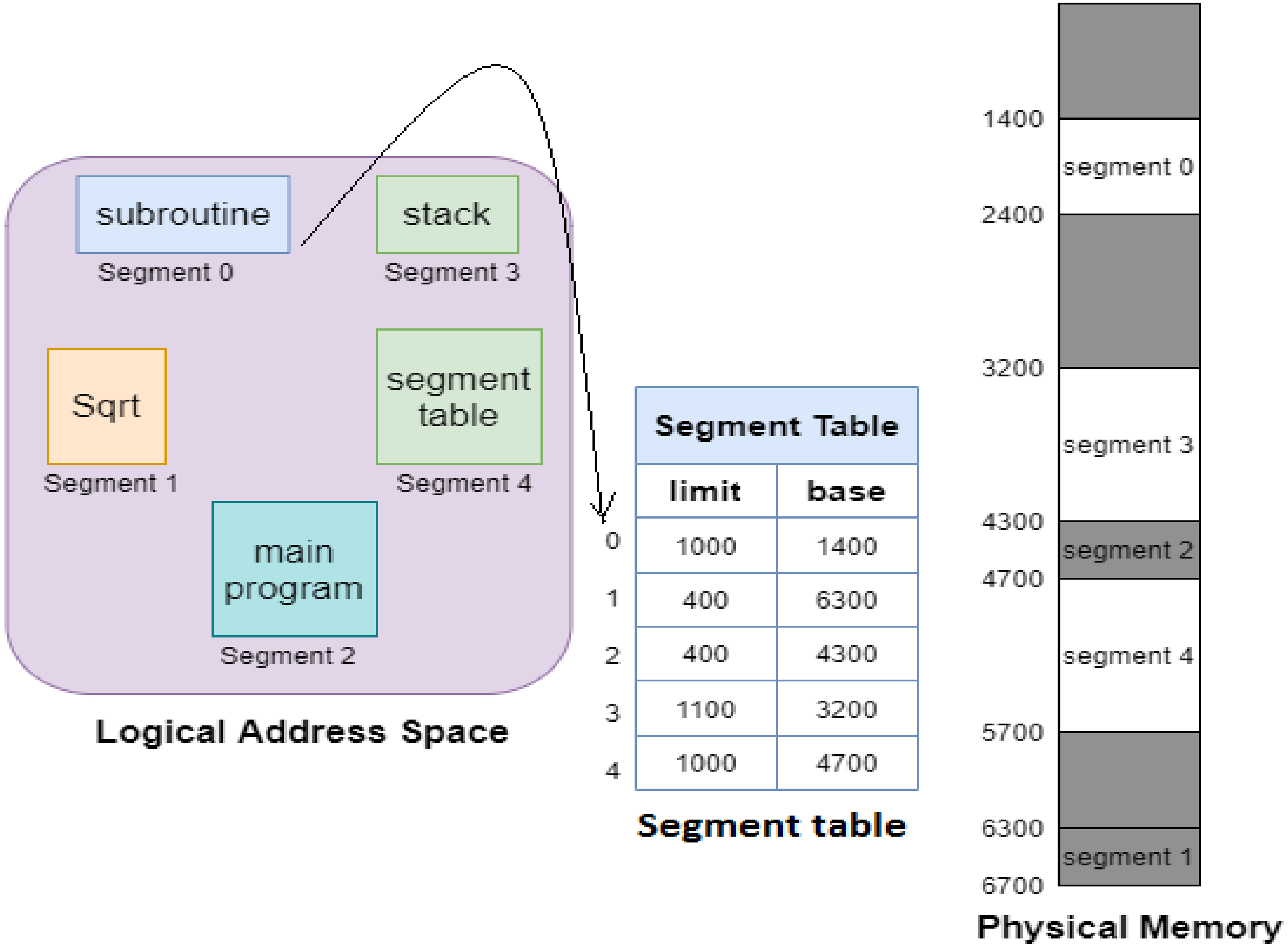
1. **Segment Base/base address:** The segment base mainly contains the starting physical address where the segments reside in the memory.
2. **Segment Limit:** mainly used to specify the length of the segment.

For example Segment-0 is present at base address of 1400 & limit 1000 means it can store up to 1400+1000=2400

	Limit	Base
Segment 0 →	1000	1400
Segment 1 →	400	6300
Segment 2 →	400	4300
Segment 3 →	1100	3200
Segment 4 →	1000	4700

Example of Segmentation:

- There are five segments numbered from 0 to 4. These segments will be stored in Physical memory as shown.



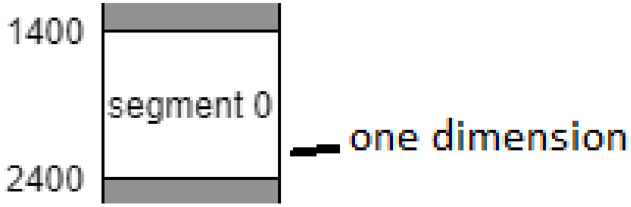
Segment Table

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

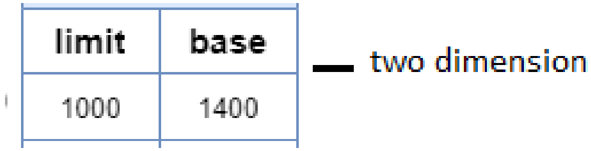
Segment table

Segmentation Hardware

- Physical memory has one dimensional →



- Logical address has two dimensional →



limit	base
1000	1400

two dimension

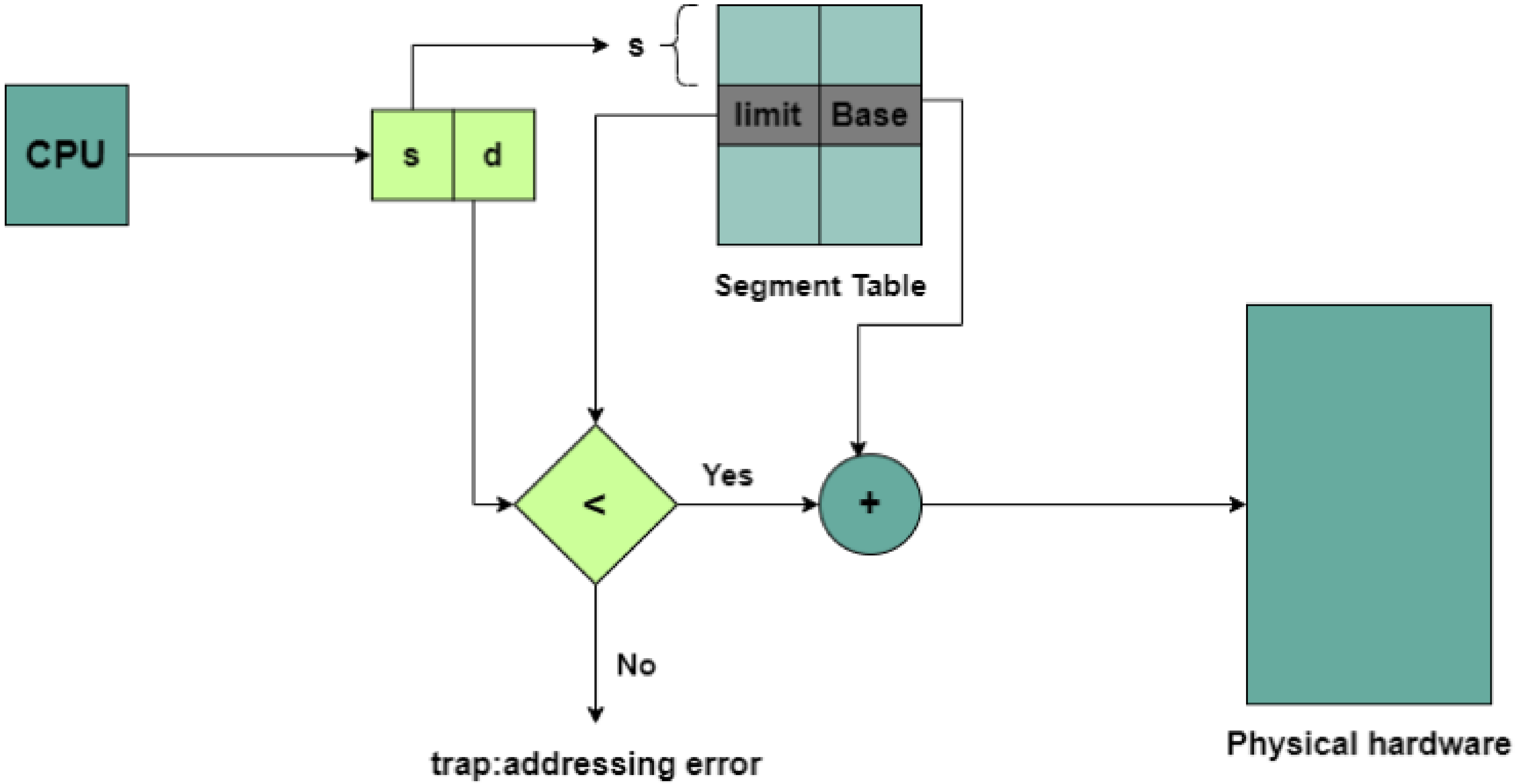
- Generally, there is no simple relationship between logical addresses and physical addresses in this scheme.
- The mapping of a two-dimensional Logical address into a one-dimensional Physical address is done using the segment table.
- This is done with the help of segment table.

Segmentation Hardware

- The logical address generated by CPU consist of two parts:

1. Segment Number(s): It is used as an index into the segment table.
2. Offset(d): It must lie in between '0' and 'segment limit'. In this case, if the Offset exceeds the segment limit then the trap is generated.

- If offset less than (<) limit then base address adds in offset in physical address.



Segmentation

Introduction

- Segmentation is another non-contiguous memory management technique like paging in which the memory is divided into the variable size parts.
- Each part is known as a segment which can be allocated to a process.
- The details about each segment are stored in a table called a segment table.

- Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

Importance of Segmentation

- Paging is closer to the Operating system rather than the User.
- It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of functions which need to be loaded in the same page.
- Operating system doesn't care about the User's view of the process.

- It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory.
- It decreases the efficiency of the system.
- It is better to have segmentation which divides the process into the segments.
- Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.

Translation of Logical address into physical address by segment table

- CPU generates a logical address which contains two parts:

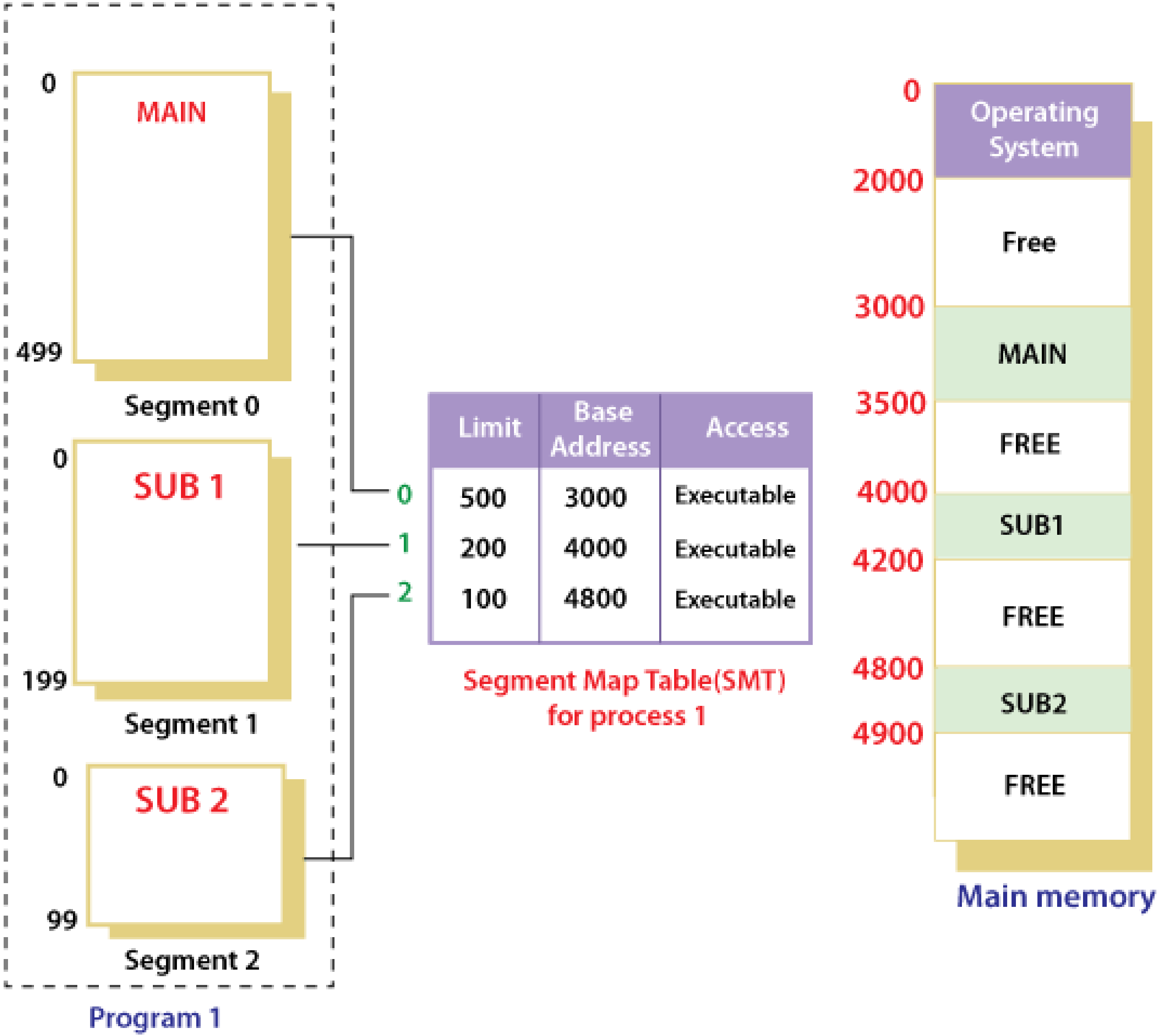
1. Segment Number

2. Offset

- **For Example:**
- Suppose a 16 bit address is used with 4 bits for the segment number and 12 bits for the segment offset so the maximum segment size is 4096 and the maximum number of segments that can be refereed is 16.

- When a program is loaded into memory, the segmentation system tries to locate space that is large enough to hold the first segment of the process, space information is obtained from the free list maintained by memory manager.
- Then it tries to locate space for other segments.
- Once adequate space is located for all the segments, it loads them into their respective areas.

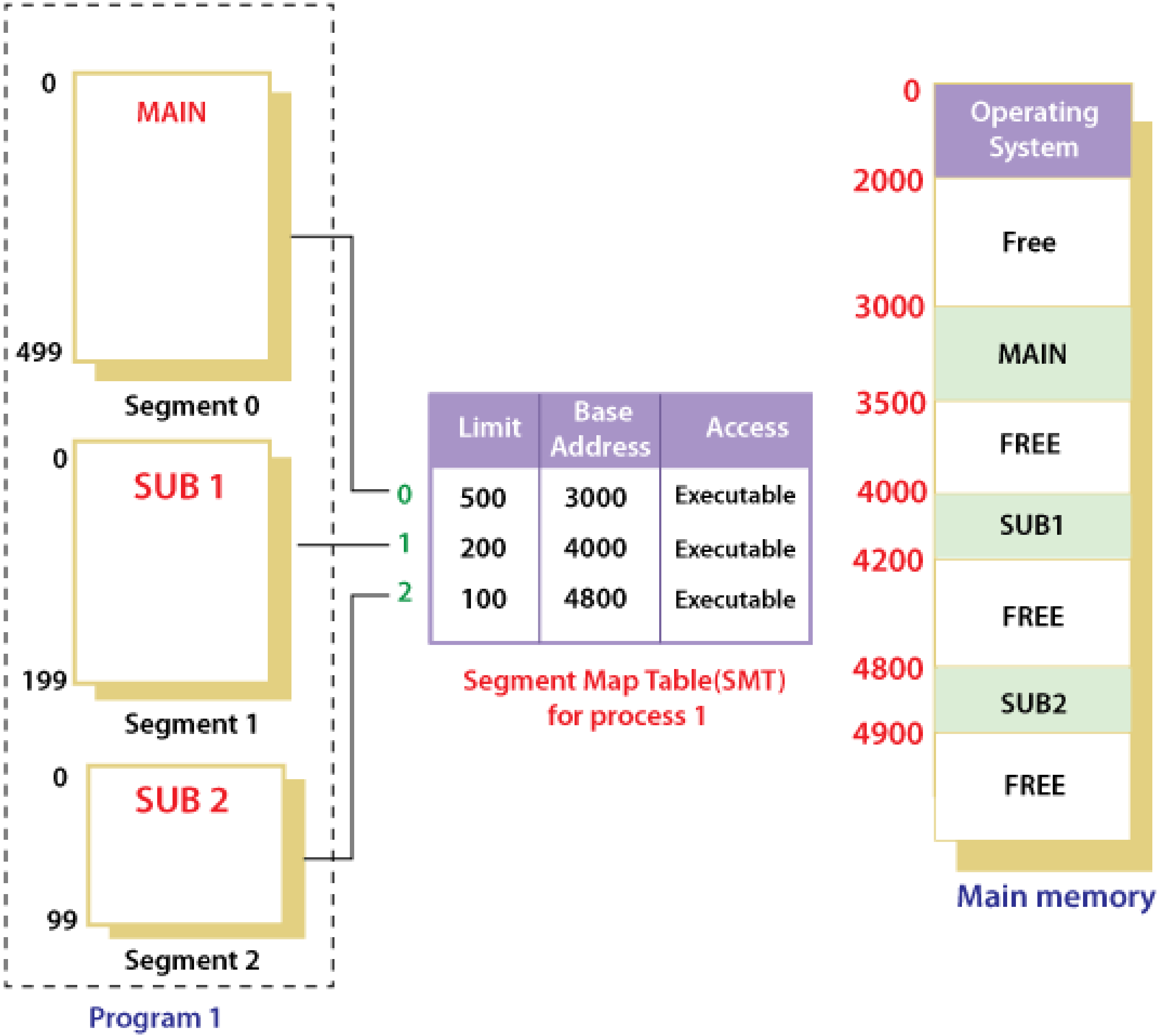
- The operating system also generates a segment map table for each program.



Limit	Base Address	Access
500	3000	Executable
200	4000	Executable
100	4800	Executable

Segment Map Table(SMT) for process 1

- With the help of segment map tables and hardware assistance, the operating system can easily translate a logical address into physical address on execution of a program.



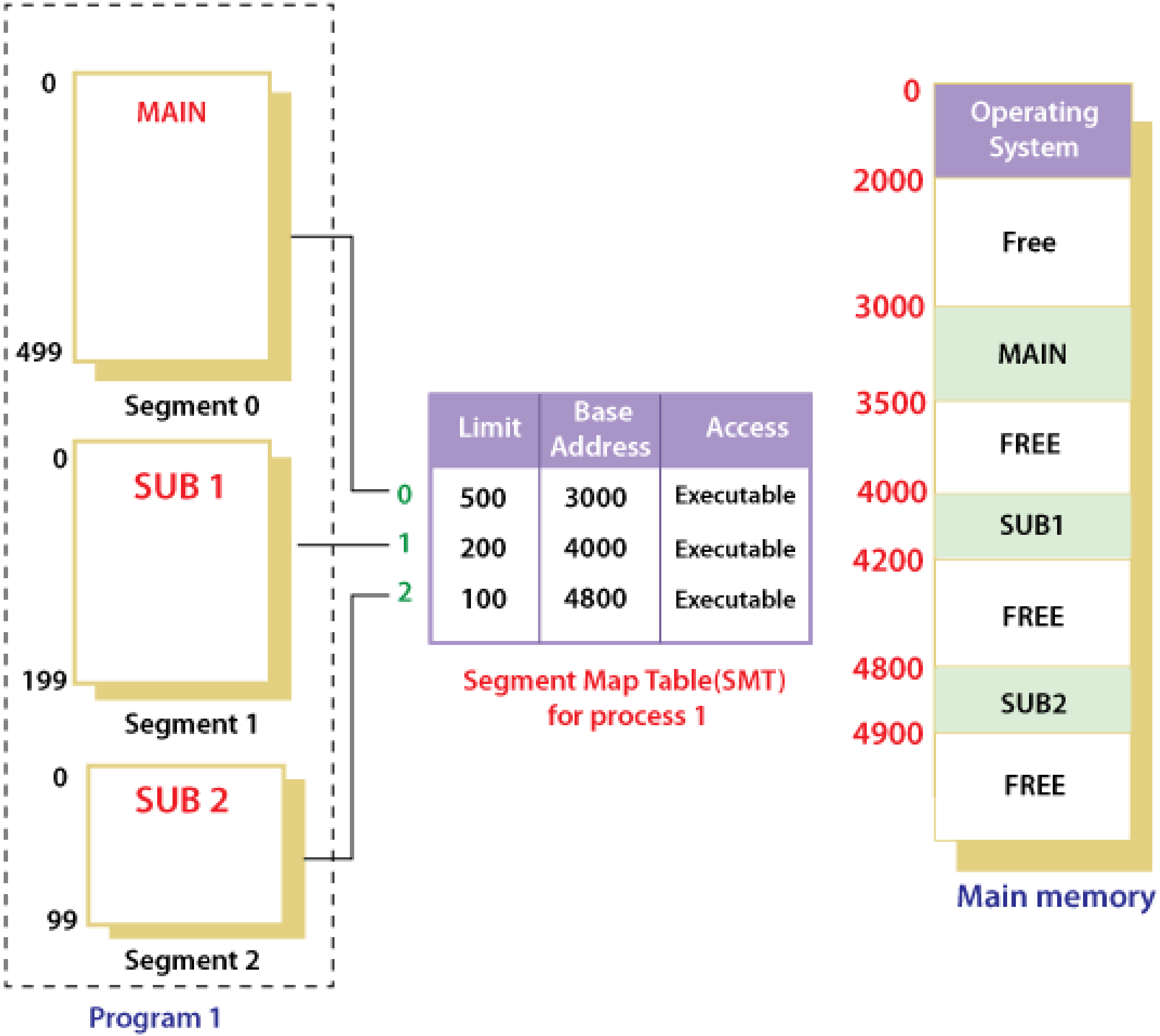
Limit	Base Address	Access
500	3000	Executable
200	4000	Executable
100	4800	Executable

Segment Map Table(SMT) for process 1

- The **Segment number** is mapped to the segment table.

- The limit of the respective segment is compared with the offset.

- If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.



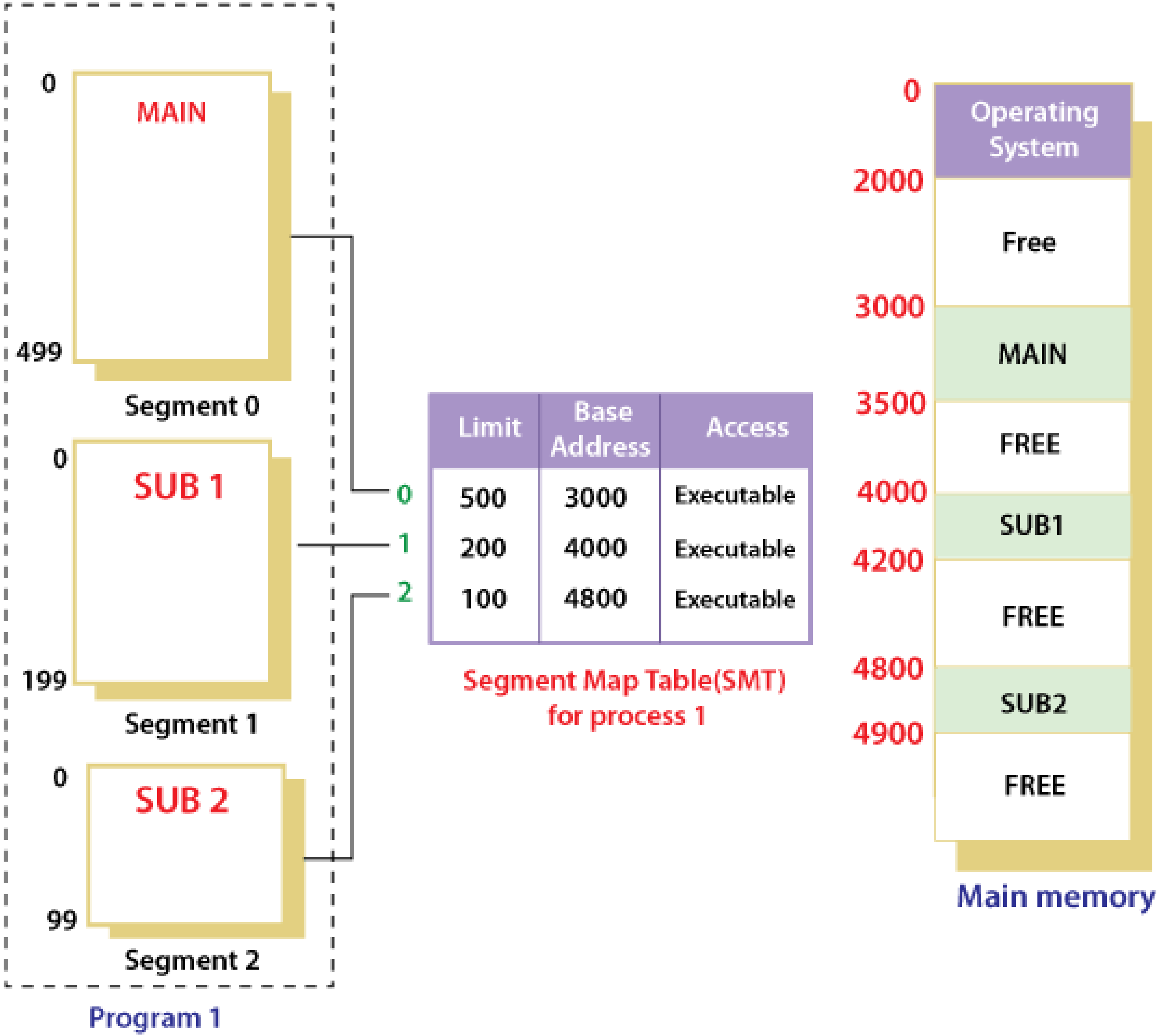
Program 1

Limit	Base Address	Access
500	3000	Executable
200	4000	Executable
100	4800	Executable

Segment Map Table(SMT) for process 1

Main memory

- In the case of valid addresses, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.



Limit	Base Address	Access
500	3000	Executable
200	4000	Executable
100	4800	Executable

Segment Map Table(SMT) for process 1

- **Advantages of Segmentation**

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compared to the page table in paging.

- **Disadvantages**

1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

Concept	Paging	Segmentation
Definition	Paging is a memory management scheme in which each process is divided into fixed-size pages.	Segmentation is a memory management scheme in which a process is divided into variable-sized segments.
Memory Allocation	Pages are allocated to processes as a whole unit.	Segments can be allocated to processes in a non-contiguous manner.
Memory Management	Paging is managed by hardware through a page table.	Segmentation is managed by software through a segment table.
Memory Utilization	Paging results in less internal fragmentation, but more external fragmentation.	Segmentation results in more internal fragmentation but less external fragmentation.
Page Size	Page size is fixed and determined by the system.	Segment size is variable and determined by the process.
Overhead	Paging has lower overhead due to a simpler memory management scheme.	Segmentation has higher overhead due to a more complex memory management scheme.
Address Translation	Address translation in paging is straightforward as it involves a single table lookup.	Address translation in segmentation is more complex as it involves two table lookups.
Flexibility	Paging is less flexible due to the fixed page size.	Segmentation is more flexible due to the variable segment size.
Performance	Paging has better performance due to the hardware-based management and simpler address translation.	Segmentation has lower performance due to the software-based management and complex address translation.

Segmentation with Paging

Introduction

- Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.
- In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

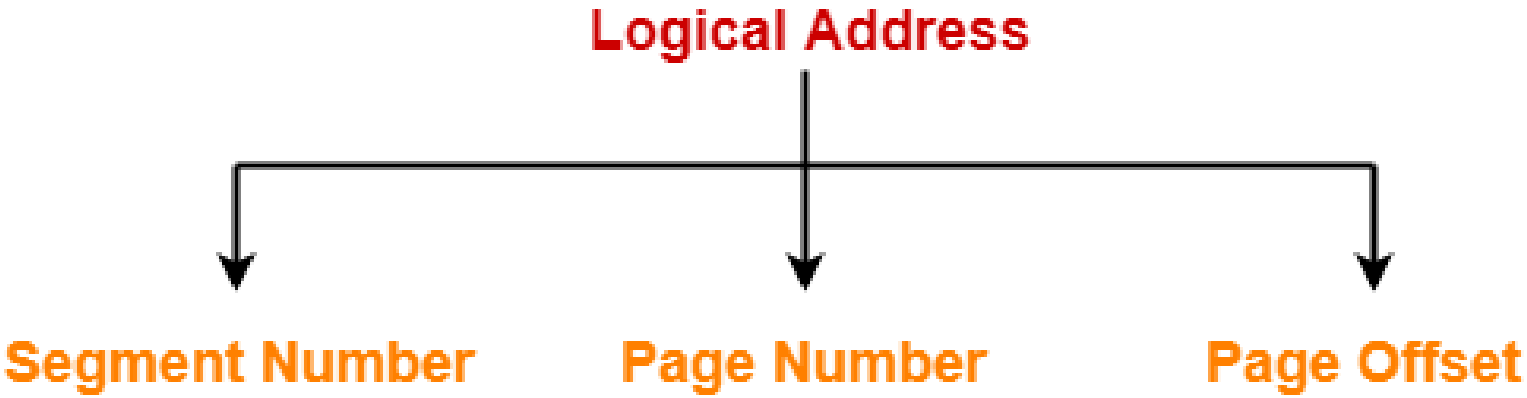
1. Pages are smaller than segments.
2. Each Segment has a page table.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

- **Segment Number** → It points to the appropriate Segment Number.
- **Page Number** → It Points to the exact page within the segment
- **Page Offset** → Used as an offset within the page frame

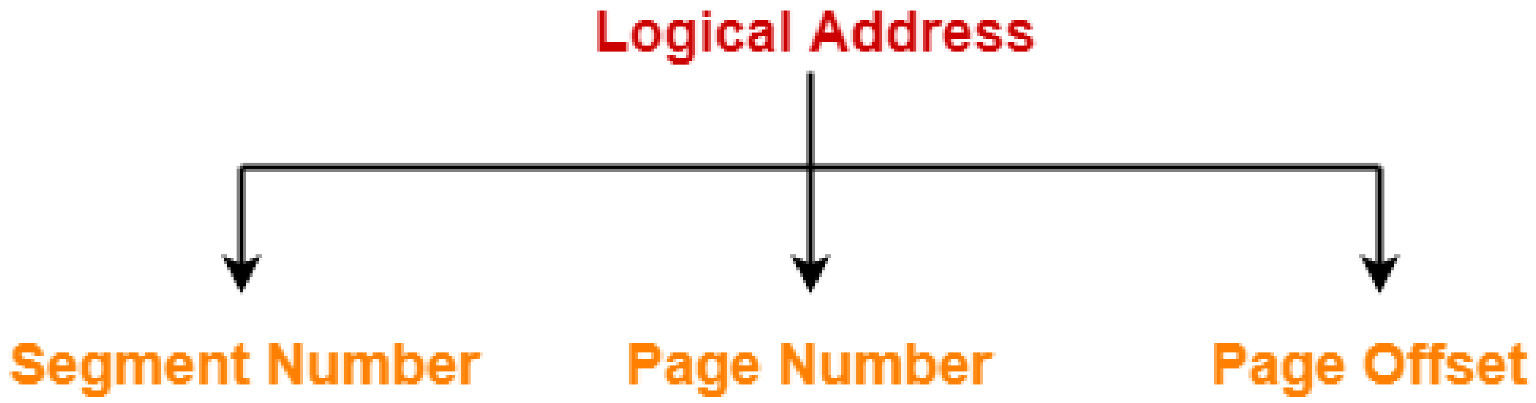
- Each Page table contains the various information about every page of the segment.
- The Segment Table contains the information about every segment.

Translation of logical address to physical address

- Step-01: The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset.
- The Segment Offset must be less than the segment limit.
- Offset is further divided into Page number and Page Offset.



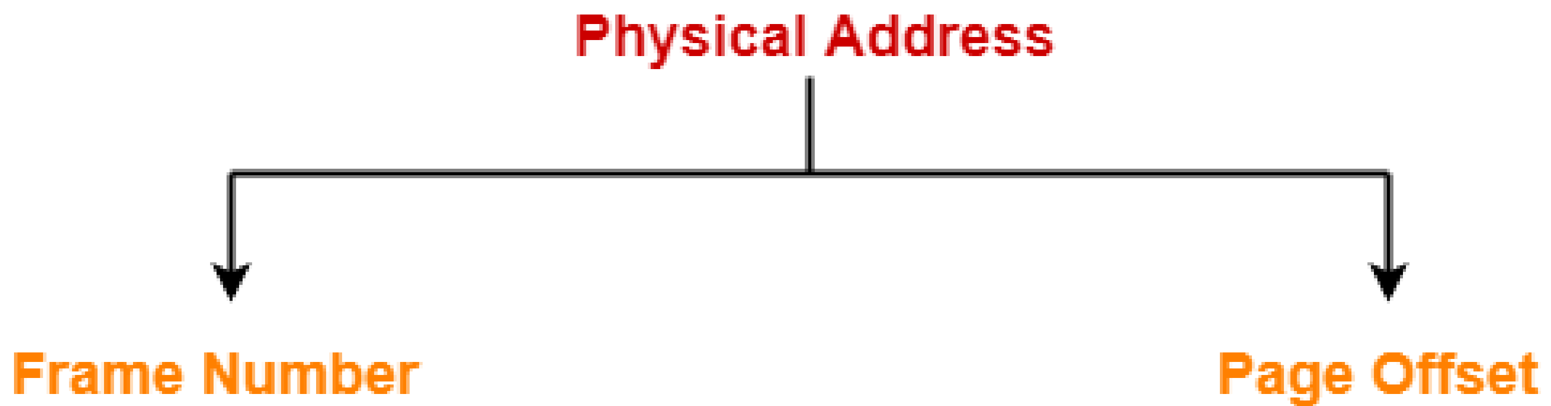
- Segment Number specifies the specific segment from which CPU wants to reads the data.
- Page Number specifies the specific page of that segment from which CPU wants to read the data.
- Page Offset specifies the specific word on that page that CPU wants to read.



- **Step-02:** For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.

- **Step-03:**
- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.

- **Step-04:**
- The frame number combined with the page offset forms the required physical address.
- For the generated page offset, corresponding word is located in the page and read.



Advantages of segmented paging

- Segment table contains only one entry corresponding to each segment.
- It reduces memory usage.
- The size of **Page Table** is limited by the segment size.
- It solves the problem of external fragmentation.

Disadvantages of segmented paging

- Segmented paging suffers from internal fragmentation.
- The complexity level is much higher as compared to paging.

Virtual Memory

- Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory.
- This is done by treating a part of secondary memory as the main memory.
- Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from RAM to disk storage.

- Today, most personal computers (PCs) come with at least 8 GB (gigabytes) of RAM.
- But, sometimes, this is not enough to run several programs at one time.
- This is where virtual memory comes in.
- Virtual memory frees up RAM by swapping data that has not been used recently over to a storage device, such as a hard drive or solid-state drive (SSD).

- Virtual memory is important for improving system performance, multitasking and using large programs.
- However, users should not overly rely on virtual memory, since it is considerably slower than RAM.
- If the OS has to swap data between virtual memory and RAM too often, the computer will begin to slow down. This is called thrashing.

How virtual memory works

- Virtual memory uses both hardware and software to operate.
- When an application is in use, data from that program is stored in a physical address using RAM.
- A memory management unit (MMU) maps the address to RAM and automatically translates addresses.

- If, at any point, the RAM space is needed for something more urgent, data can be swapped out of RAM and into virtual memory.
- The computer's memory manager is in charge of keeping track of the shifts between physical and virtual memory.
- If that data is needed again, the computer's MMU will use a context switch to resume execution.
- While copying virtual memory into physical memory, the OS divides memory with a fixed number of addresses into either pagefiles or swap files.

- Each page is stored on a disk, and when the page is needed, the OS copies it from the disk to main memory and translates the virtual addresses into real addresses.
- However, the process of swapping virtual memory to physical is rather slow.
- This means using virtual memory generally causes a noticeable reduction in performance.
- Because of swapping, computers with more RAM are considered to have better performance.

- **Types of virtual memory**
- A computer's MMU manages virtual memory operations.
- In most computers, the MMU hardware is integrated into the central processing unit (CPU).
- The CPU also generates the virtual address space.
- In general, virtual memory is either paged or segmented.

Advantages of virtual memory

- It can handle twice as many addresses as main memory.
- It enables more applications to be used at once.
- It frees applications from managing shared memory and saves users from having to add memory modules when RAM space runs out.
- It has increased speed when only a segment of a program is needed for execution.
- It has increased security because of memory isolation.

- It enables multiple larger applications to run simultaneously.
- Allocating memory is relatively inexpensive.
- It does not need external fragmentation.
- CPU use is effective for managing logical partition workloads.
- Data can be moved automatically.
- Pages in the original process can be shared during a fork system call operation that creates a copy of itself.

Limitations of virtual memory

- Applications run slower if they are running from virtual memory.
- Data must be mapped between virtual and physical memory, which requires extra hardware support for address translations, slowing down a computer further.

- The size of virtual storage is limited by the amount of secondary storage, as well as the addressing scheme with the computer system.
- Thrashing can occur if there is not enough RAM, which will make the computer perform slower.
- It may take time to switch between applications using virtual memory.
- It lessens the amount of available hard drive space.

Demand Paging

- The process of loading the page into memory on demand (whenever a page fault occurs) is known as demand paging.
- The process includes the following steps are as follows:

1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.

3. The OS will search for the required page in the logical address space.

4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.

5. The page table will be updated accordingly.

6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

- Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

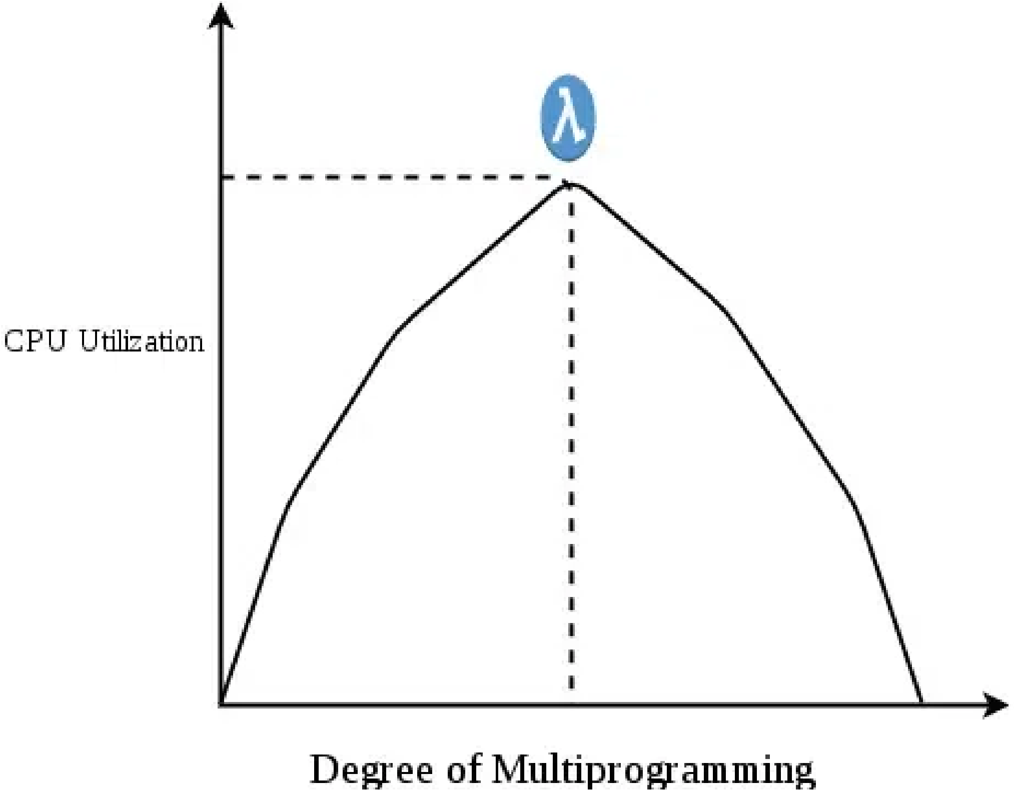
Thrashing

- At any given time, only a few pages of any process are in the main memory, and therefore more processes can be maintained in memory.
- In the steady state practically, all of the main memory will be occupied with process pages, so that the processor and OS have direct access to as many processes as possible.

- Thus when the OS brings one page in, it must throw another out.
- If it throws out a page just before it is used, then it will just have to get that page again almost immediately.
- Too much of this leads to a condition called Thrashing.
- The system spends most of its time swapping pages rather than executing instructions.
- So a good page replacement algorithm is required.

Causes of Thrashing

- **1. High Degree of Multiprogramming:** If the number of processes keeps on increasing in the memory then the number of frames allocated to each process will be decreased.
- So, fewer frames will be available for each process. Due to this, a page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.

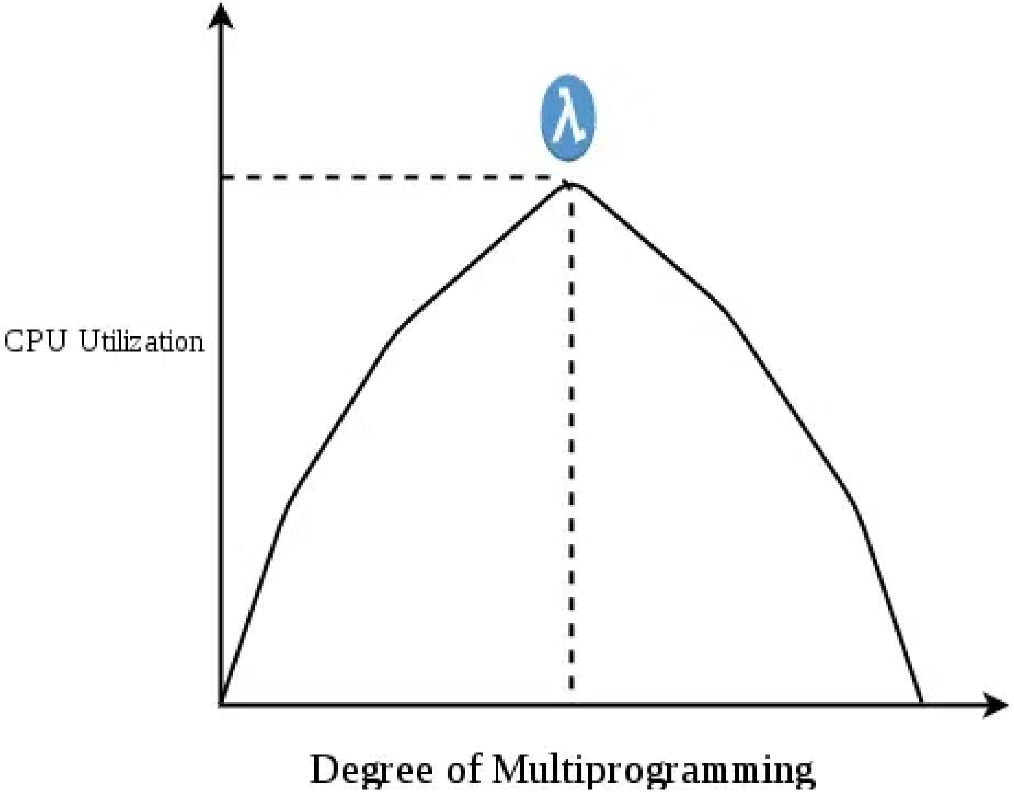


Causes of Thrashing

- For example:
- Let free frames = 400
- **Case 1**: Number of processes = 100
- Then, each process will get 4 frames.
- **Case 2**: Number of processes = 400

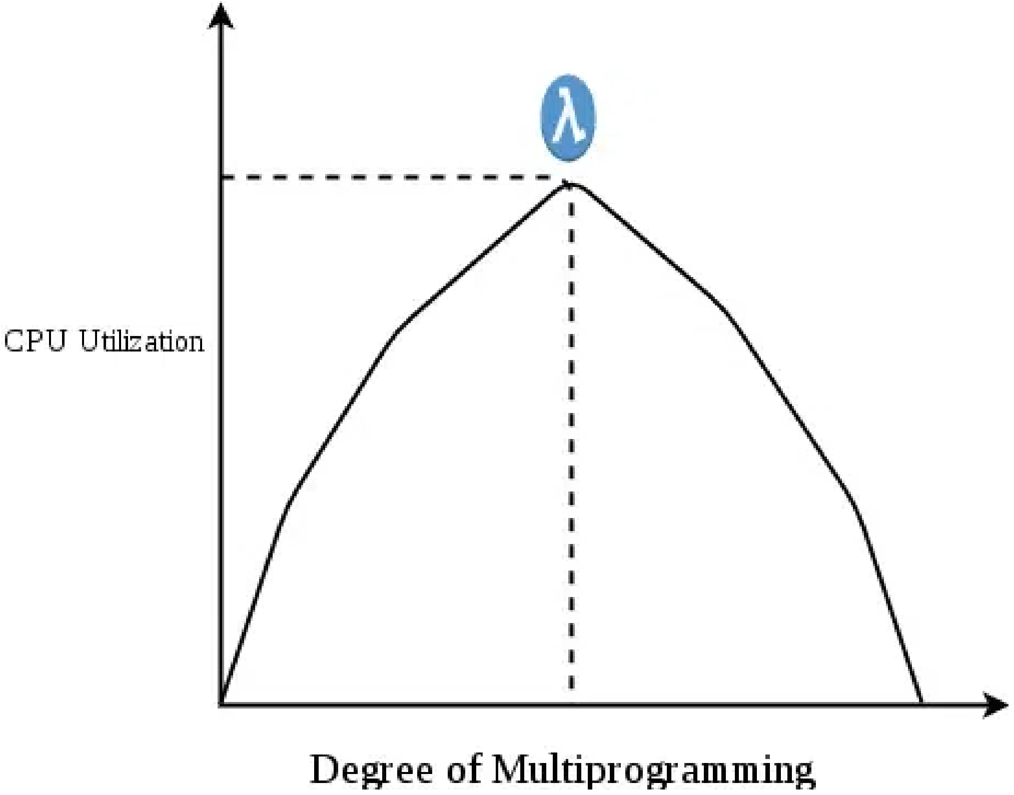
  Each process will get 1 frame.

  Case 2 is a condition of thrashing, as the number of processes is increased, frames per process are decreased.
- Hence CPU time will be consumed just by swapping pages.



Causes of Thrashing

- **2. Lacks of Frames:** If a process has fewer frames then fewer pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required.
- This may lead to thrashing.
- Hence a sufficient amount of frames must be allocated to each process in order to prevent thrashing.



- **Recovery of Thrashing**
- Do not allow the system to go into thrashing by instructing the long-term scheduler not to bring the processes into memory after the threshold.
- If the system is already thrashing then instruct the_mid-term scheduler to suspend some of the processes so that we can recover the system from thrashing.

Frame Allocation

- A number of frames allocated to each process in either static or dynamic.
- **Static Allocation:** The number of frame allocations to a process is fixed.
- **Dynamic Allocation:** The number of frames allocated to a process changes.
- **Paging Policies**
- **Fetch Policy:** It decides when a page should be loaded into memory.
- **Replacement Policy:** It decides which page in memory should be replaced.
- **Placement Policy:** It decides where in memory should a page be loaded.

- **Performance of Demand Paging**
- The performance of demand paging is often measured in terms of the *effective access time.*
- Effective access time is the amount of time it takes to access memory.
- In some sense it is an average or expected access time.

- ea = (1 - p) * ma + p*pft
- ea = effective access time

  ma = physical memory (core) access time

  pft = page fault time

  p = probability of a page fault occuring

  (1-p) = the probability of accessing memory in an available frame
- The page fault time is the sum of the additional overhead associated with accessing a page in the backing store.
- This includes additional context switches, disk latency and transfer time associated with page-in and page-out operations, the overhead of executing an operating system trap, &c.

Page Replacement Algorithms

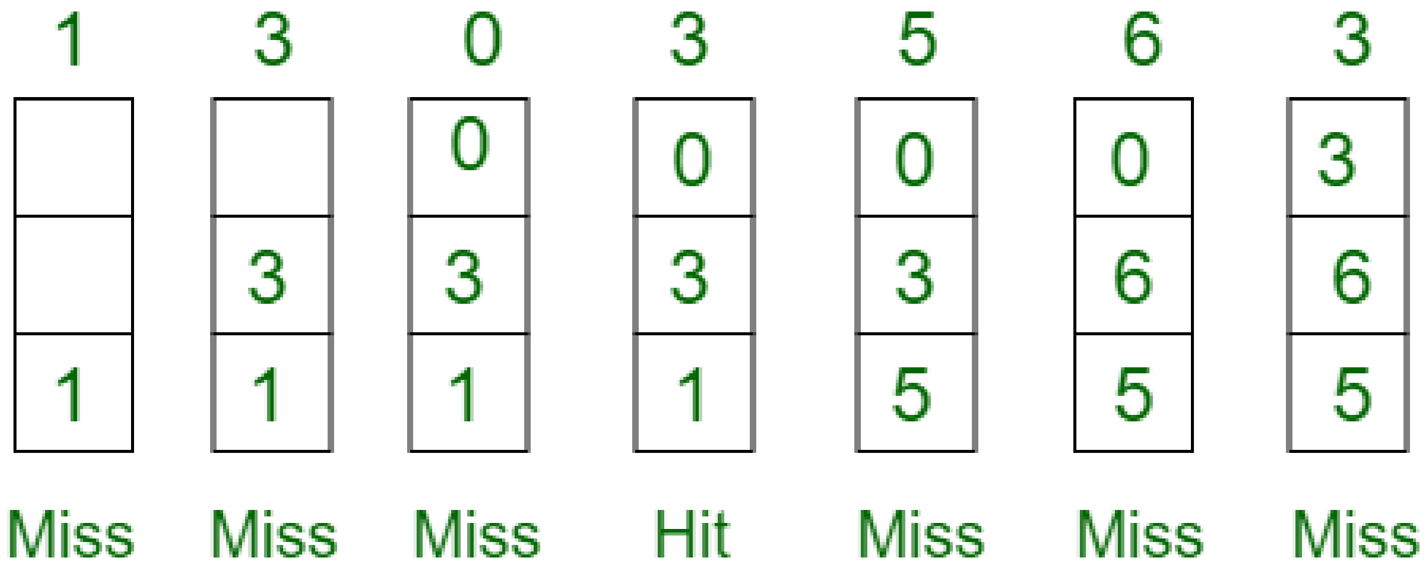
- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
- Page replacement becomes necessary when a page fault occurs and there are no free page frames in memory.
- However, another page fault would arise if the replaced page is referenced again.
- Hence it is important to replace a page that is not likely to be referenced in the immediate future.

- If no page frame is free, the virtual memory manager performs a page replacement operation to replace one of the pages existing in memory with the page whose reference caused the page fault.
- It is performed as follows:
- The virtual memory manager uses a page replacement algorithm to select one of the pages currently in memory for replacement, accesses the page table entry of the selected page to mark it as “not present” in memory, and initiates a page-out operation for it if the modified bit of its page table entry indicates that it is a dirty page.

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

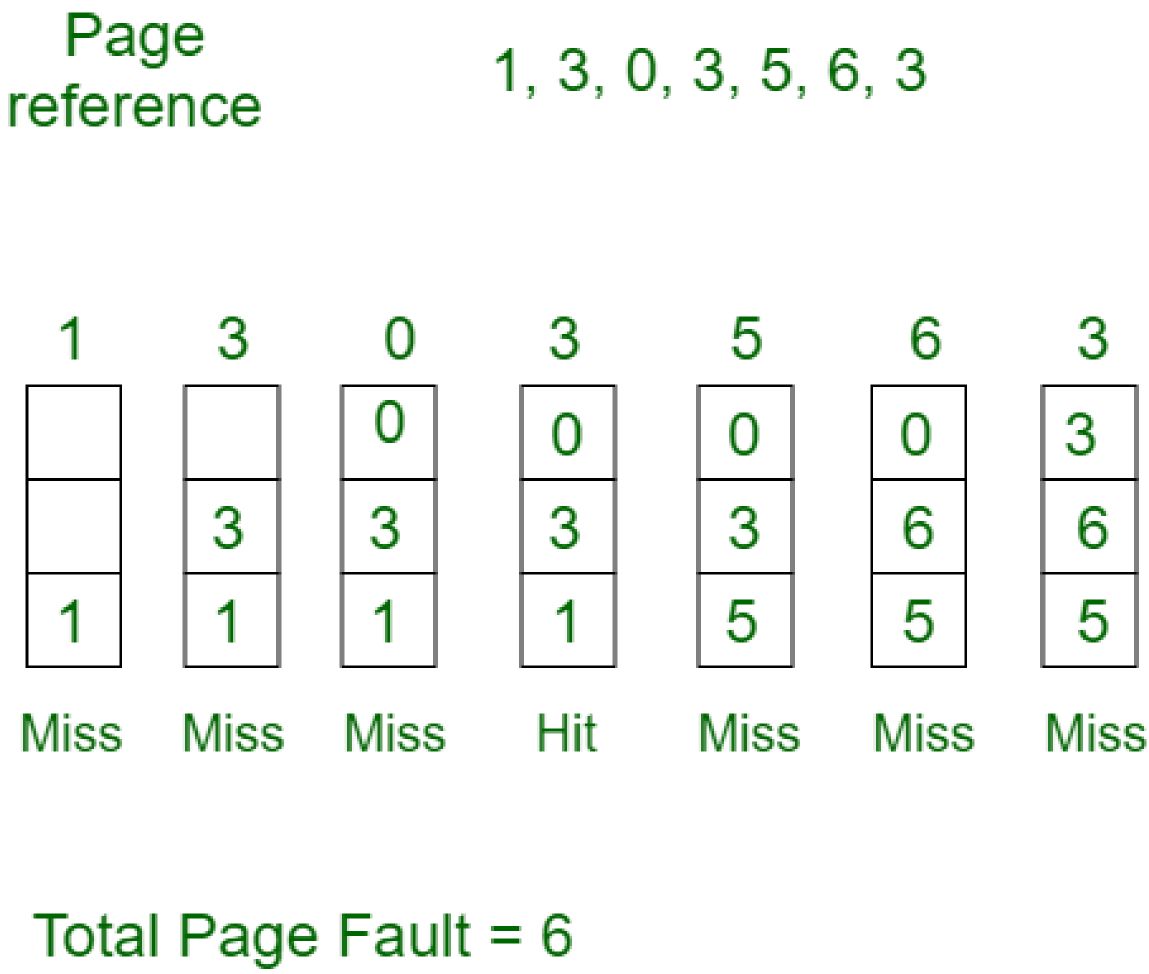
Page reference 1, 3, 0, 3, 5, 6, 3



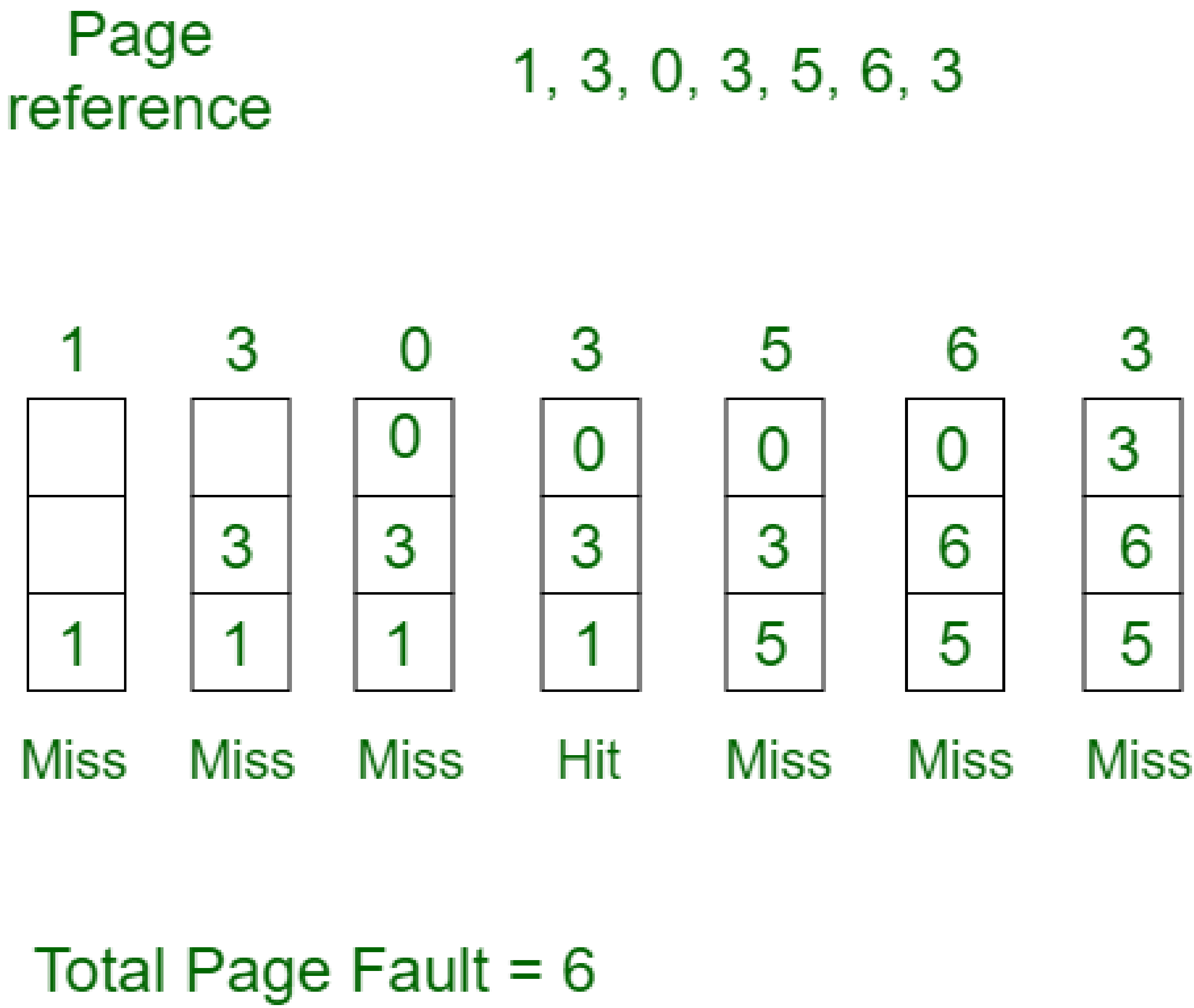
1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

- 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.**



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

Total Page Fault = 6

- **Belady's anomaly** proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.
- For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

- **2. 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.

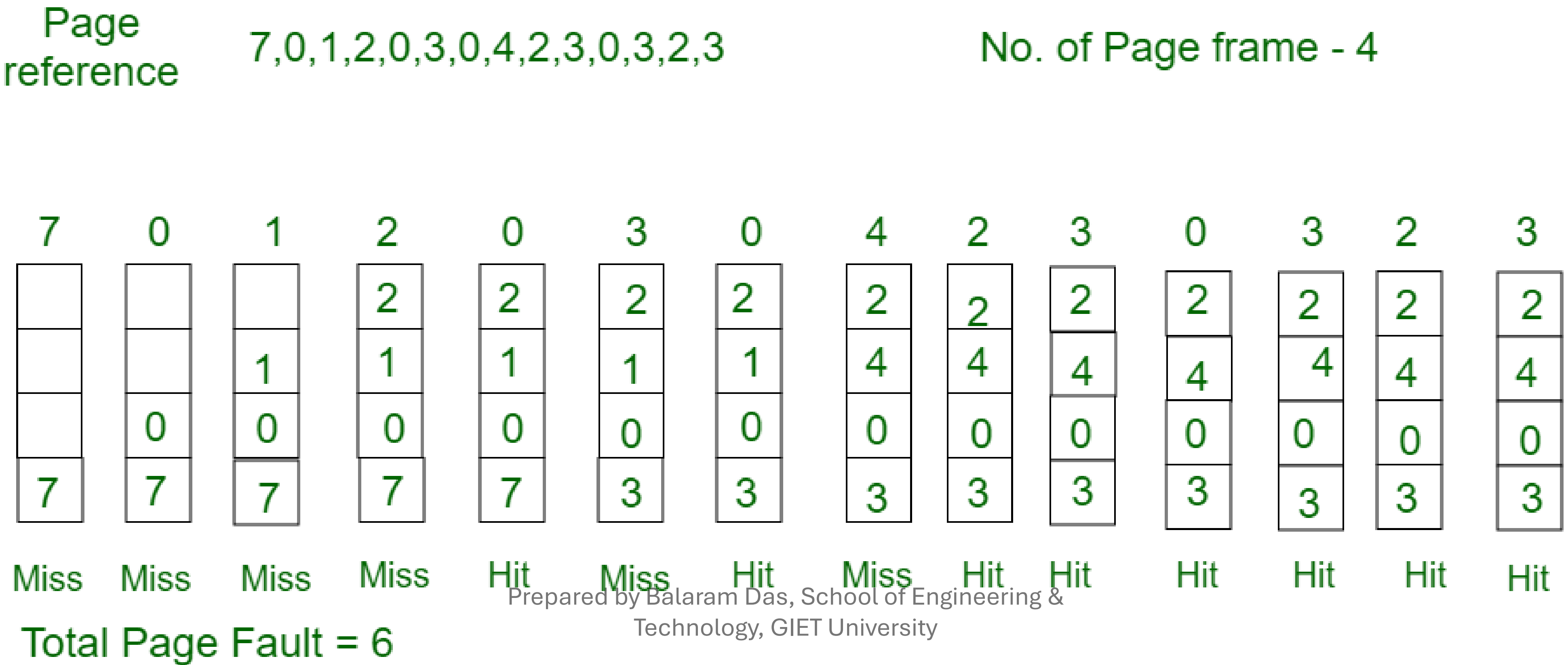
Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			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.**



Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

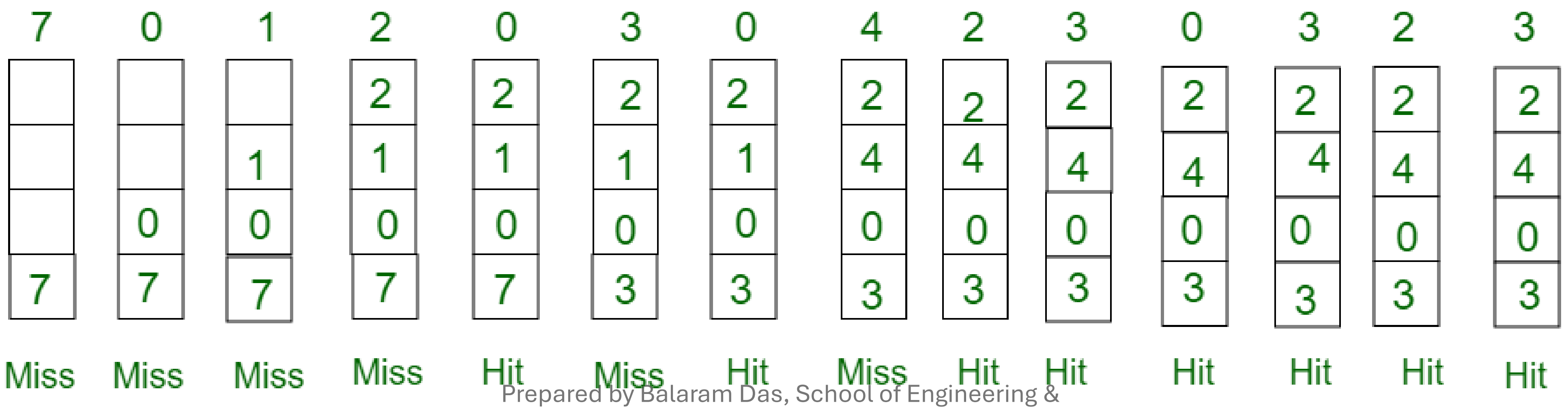
7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

- 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.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4



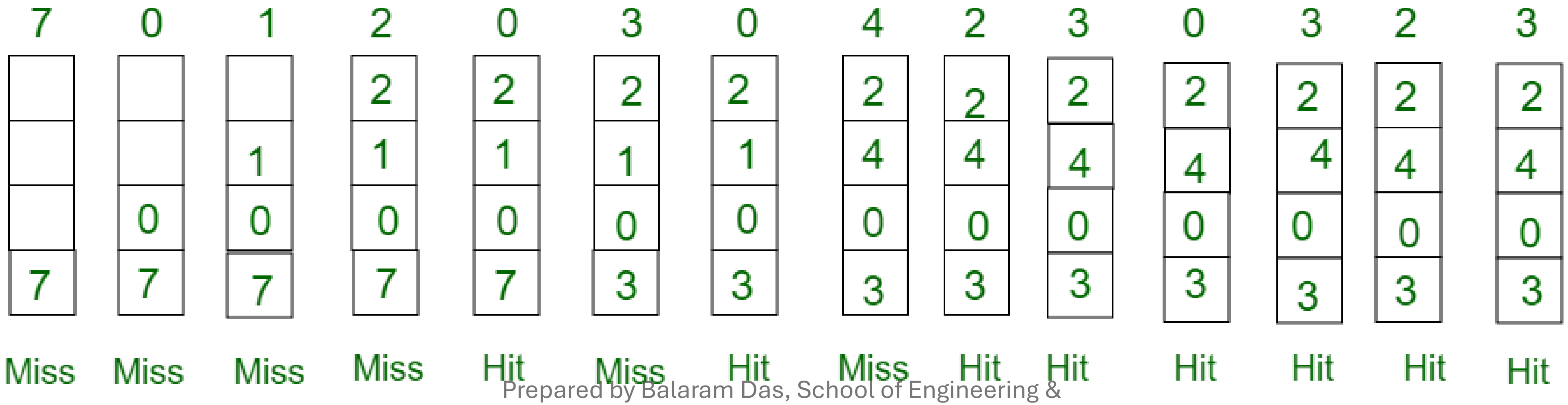
7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests.
- The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

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
			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

- **3. Least Recently Used:** In this algorithm, page will be replaced which is least recently used.
- **Example-3:** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

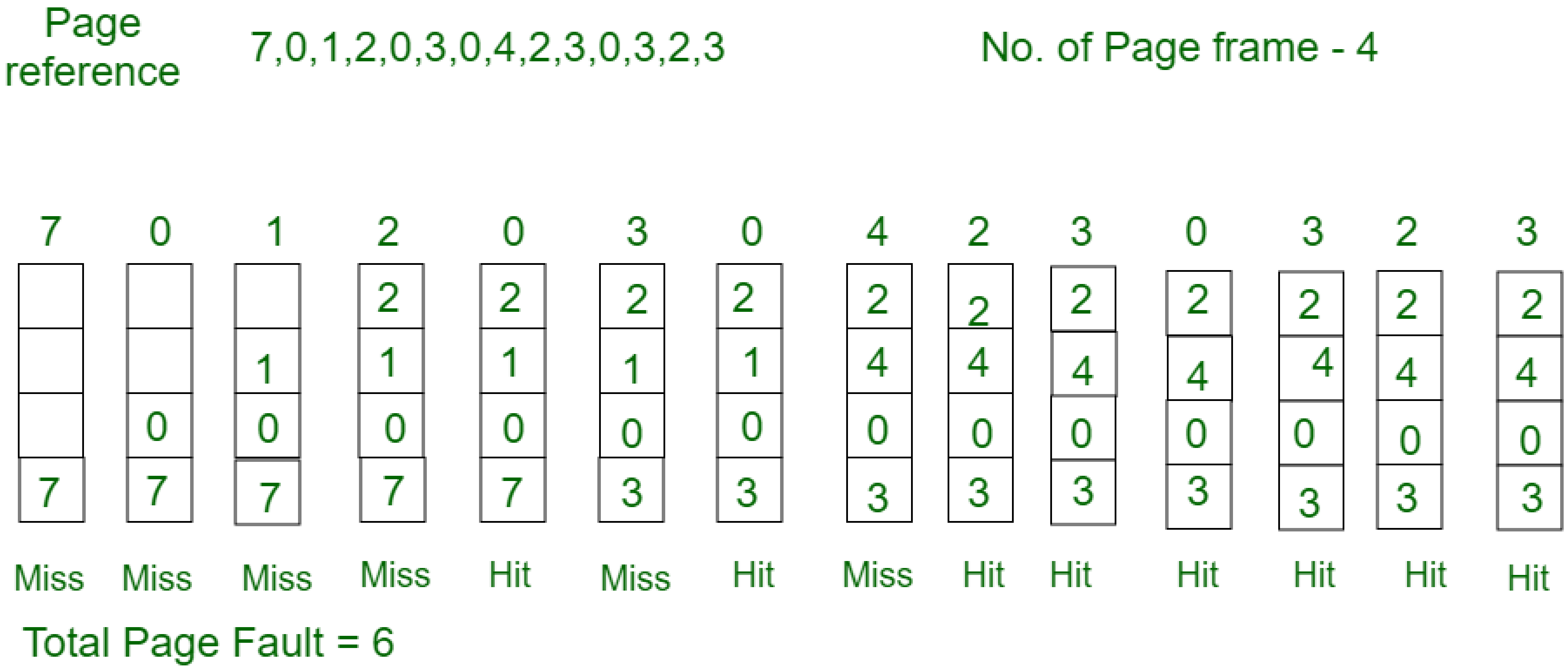
Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			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

Here LRU has same number of page fault as optimal but it may differ according to question.

- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
- 0 is already their so —> **0 Page fault.**



Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

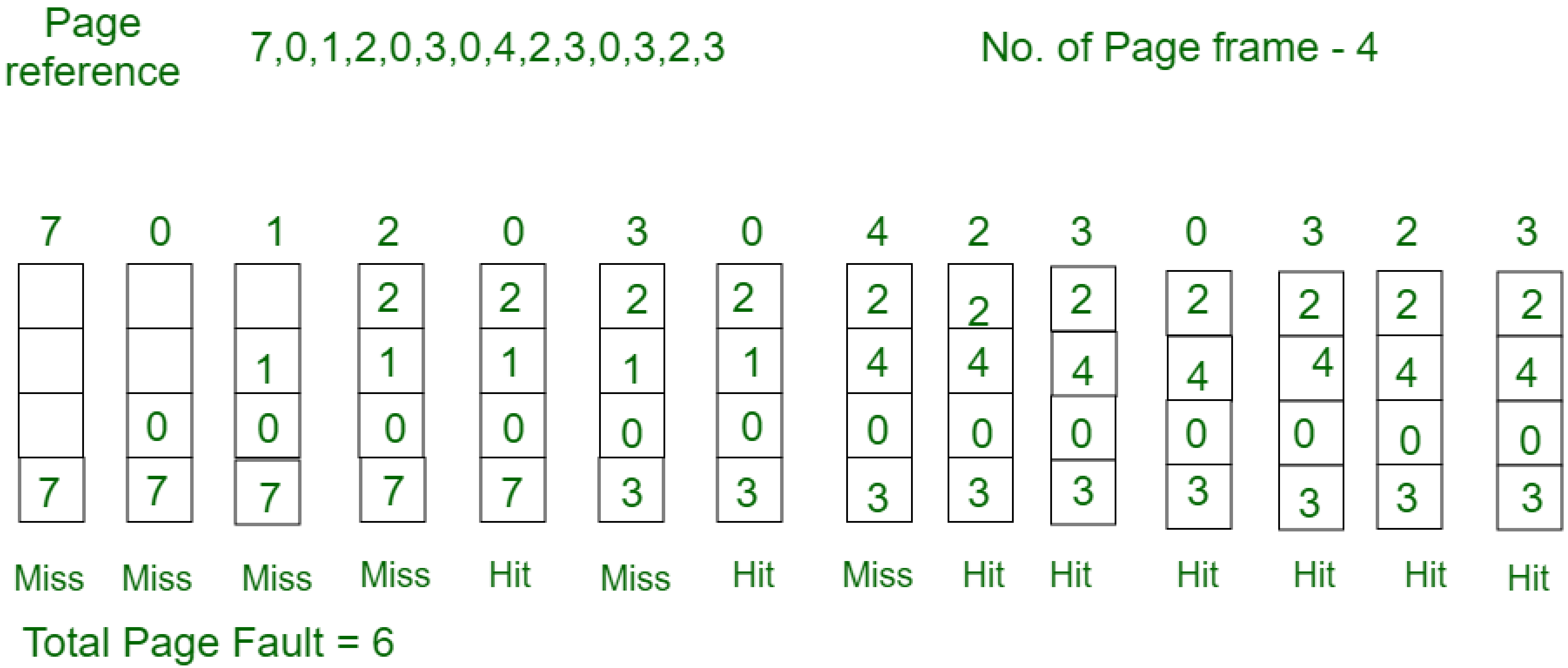
No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			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

Here LRU has same number of page fault as optimal but it may differ according to question.

- when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
- 0 is already in memory so —> **0 Page fault**.



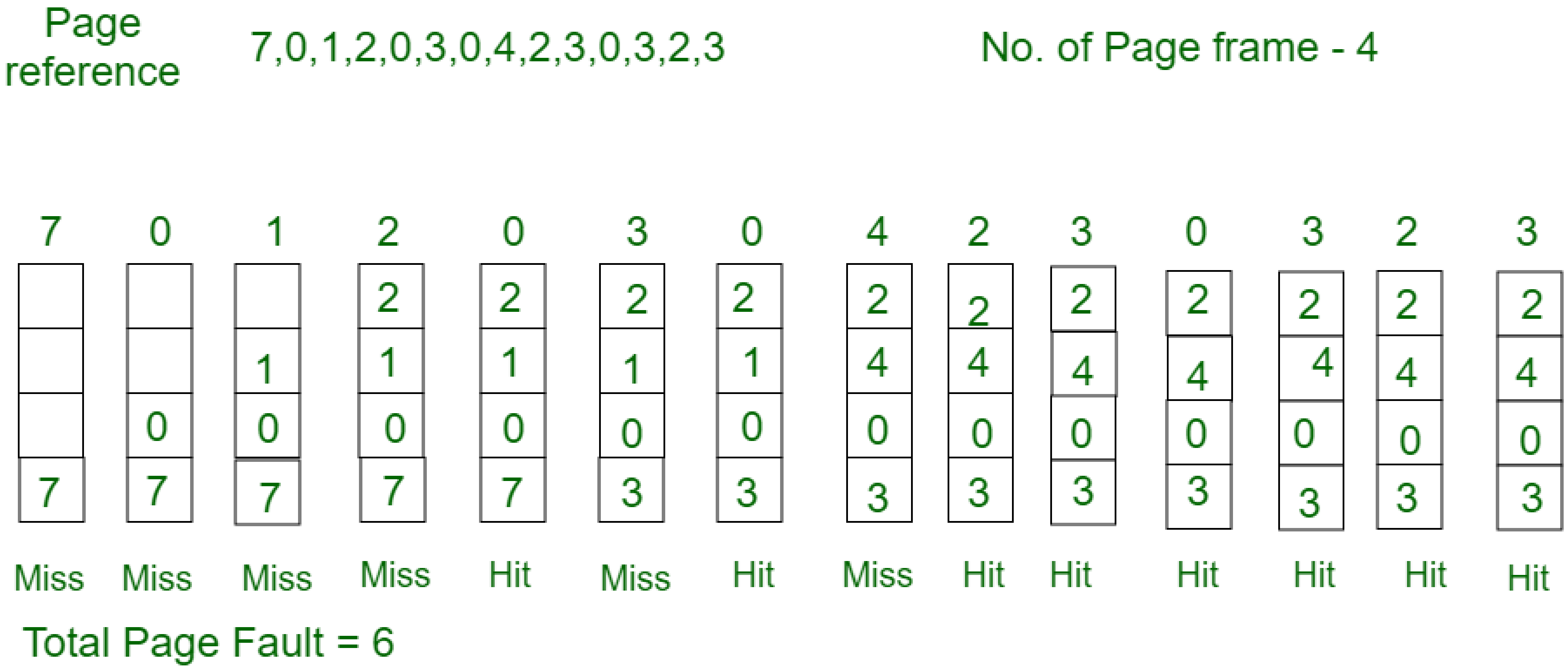
Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			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

Here LRU has same number of page fault as optimal but it may differ according to question.

- 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.



Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

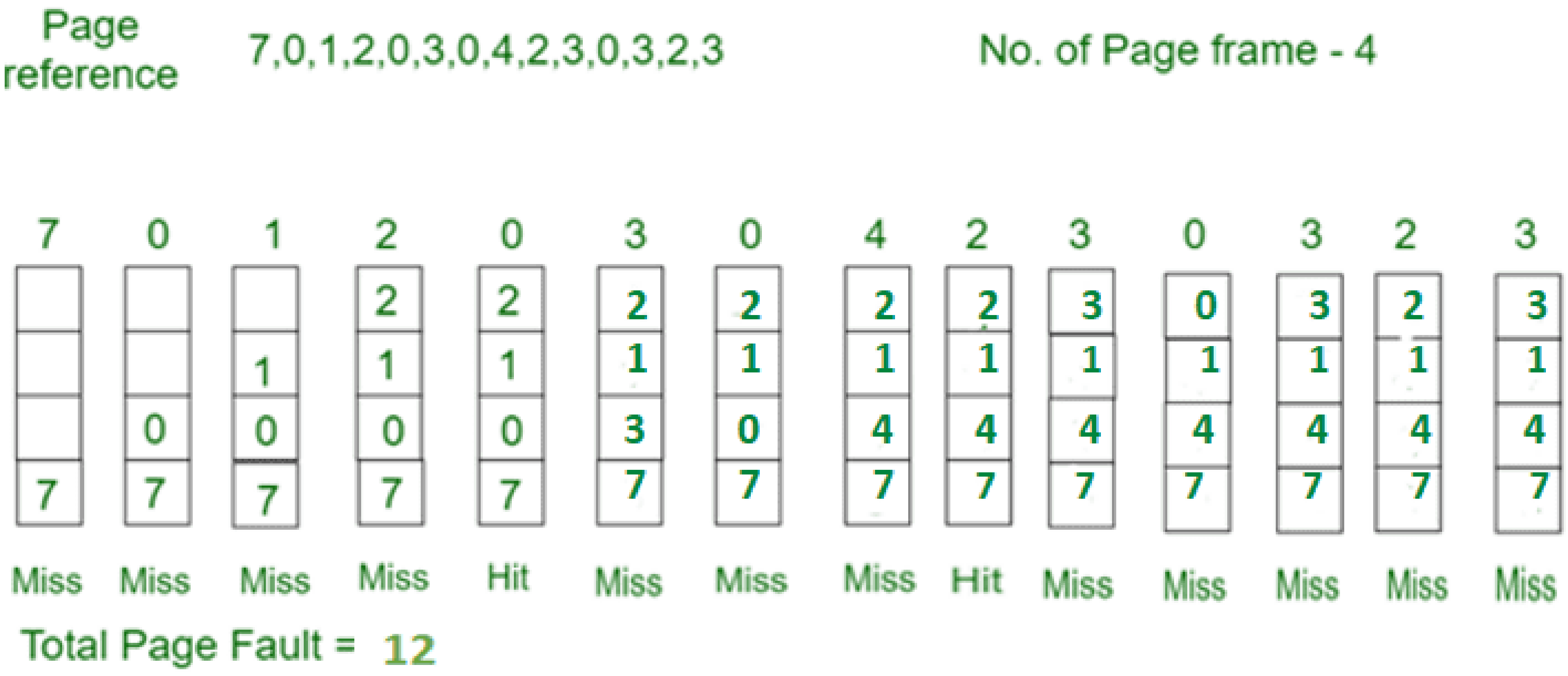
No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			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

Here LRU has same number of page fault as Optimal but it may differ according to question.

- **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.



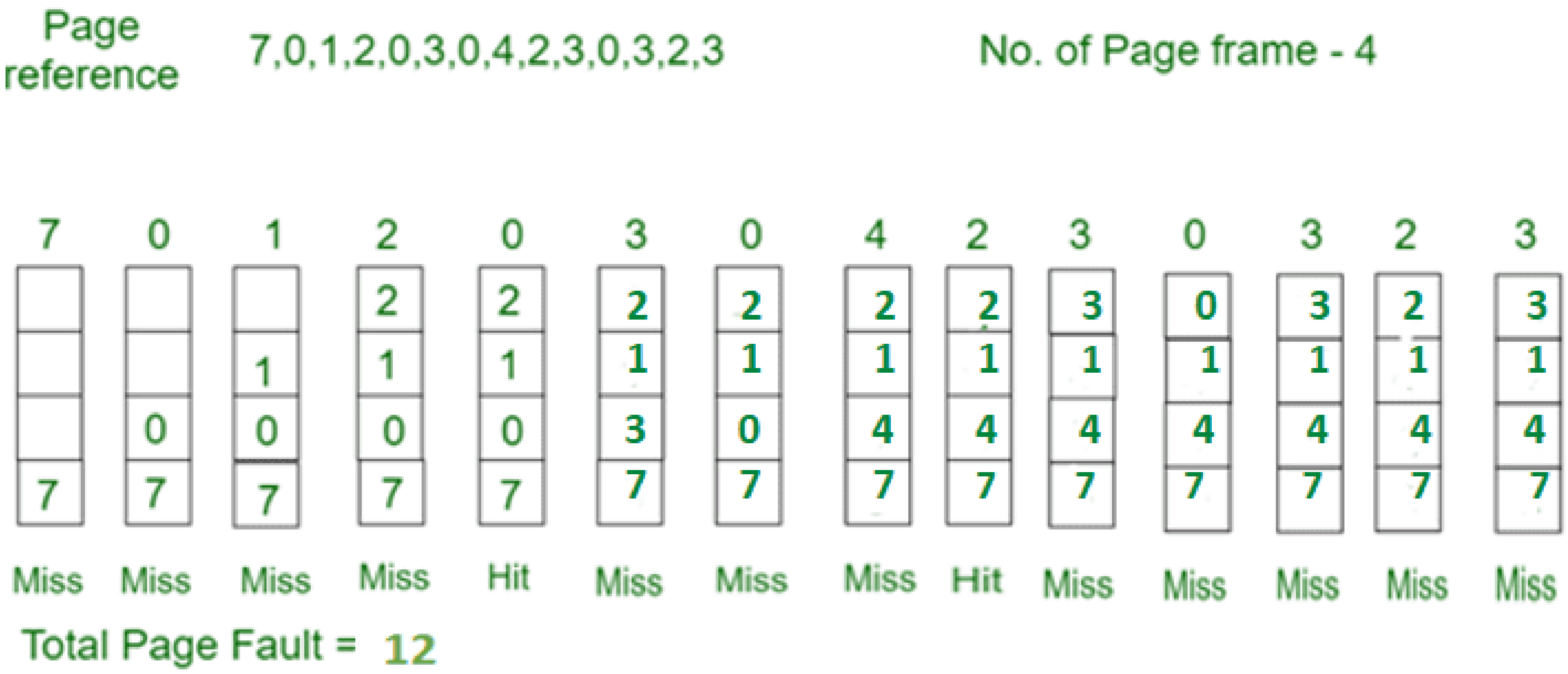
Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	3	0	3	2	3
		1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	3	0	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7	7
Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss

Total Page Fault = 12

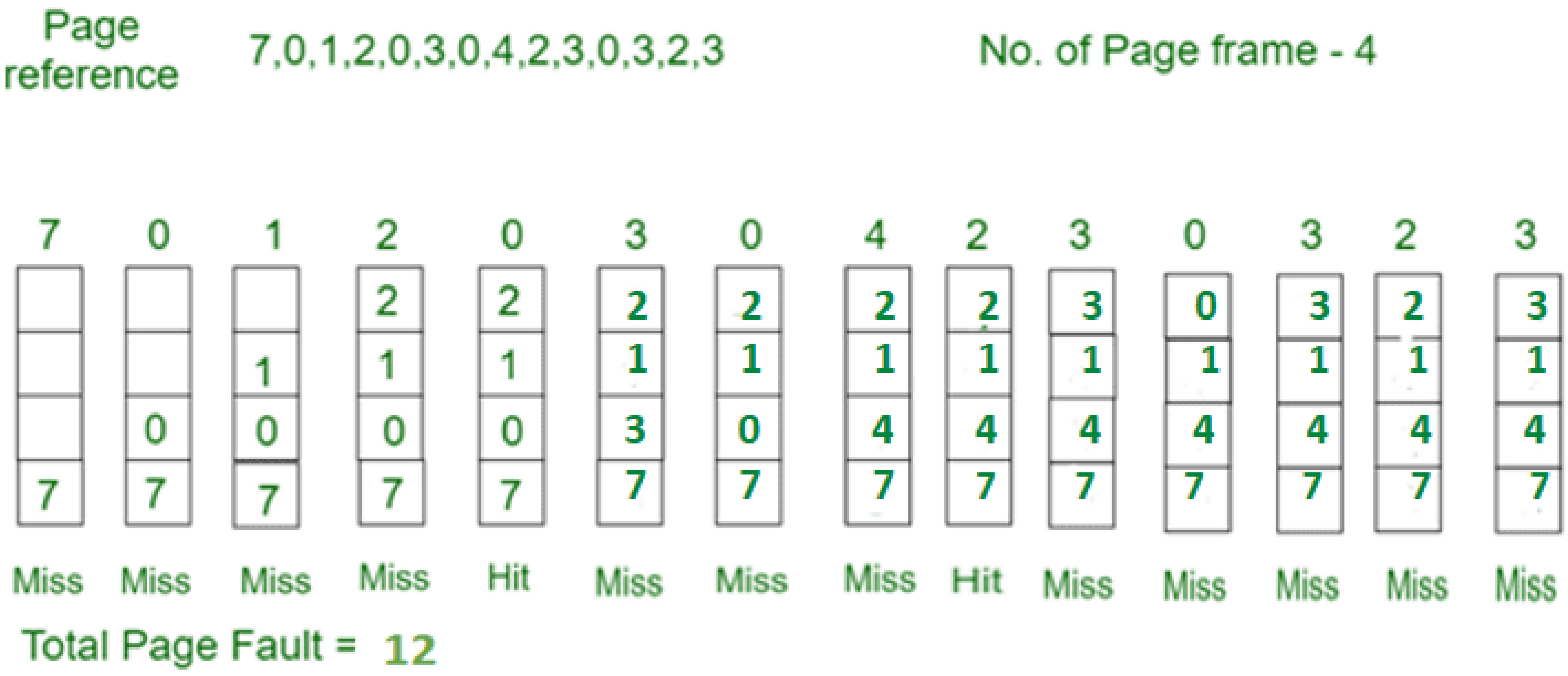
- Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
- 0 is already their so–> **0 page fault**
- when 3 comes it will take place of 0 because it is most recently used —>**1 Page fault**



- when 0 comes it will take place of 3 —>**1 Page fault**

- when 4 comes it will take place of 0 —>**1 Page fault**

- 2 is already in memory so —> **0 Page fault**



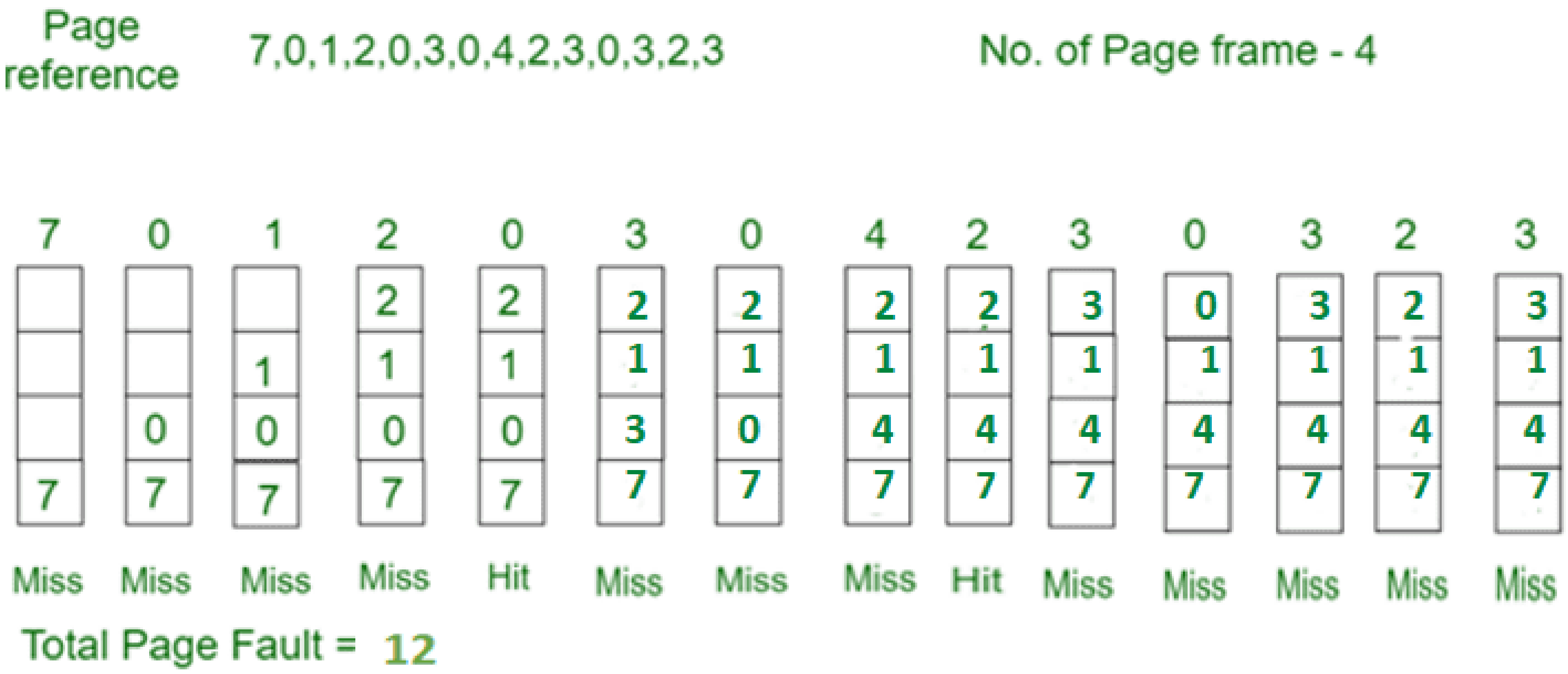
Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	3	0	3	2	3
		1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	3	0	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7	7
Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss

Total Page Fault = 12

- when 3 comes it will take place of 2 —>**1 Page fault**
- when 0 comes it will take place of 3 —>**1 Page fault**
- when 3 comes it will take place of 0 —>**1 Page fault**



Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

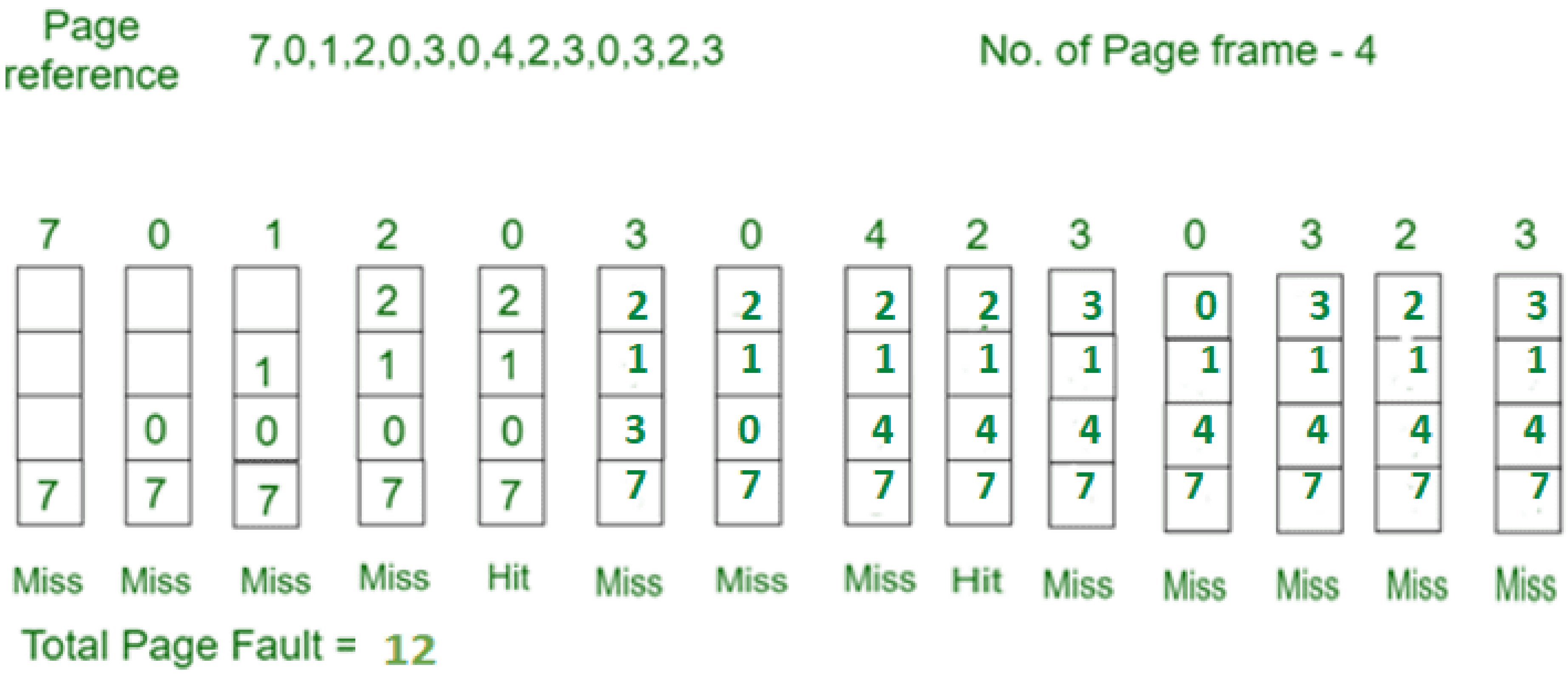
No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	3	0	3	2	3
		1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	3	0	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7	7
Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss

Total Page Fault = 12

- when 2 comes it will take place of 3 —>**1 Page fault**

- when 3 comes it will take place of 2 —>**1 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
			2	2	2	2	2	2	3	0	3	2	3
		1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	3	0	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7	7
Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss

Total Page Fault = 12

- Given page reference string: 1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4,3. Compare the number of page faults for LRU, FIFO and Optimal page replacement algorithm.

- Consider the following reference string
- 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. Assume there are three frames. Apply LRU replacement algorithm to the reference sting above and find out how many page faults are produced. Illustrate the LRU page replacement algorithm in detail

Operating System

By

Balaram Das

School of Engineering & Technology

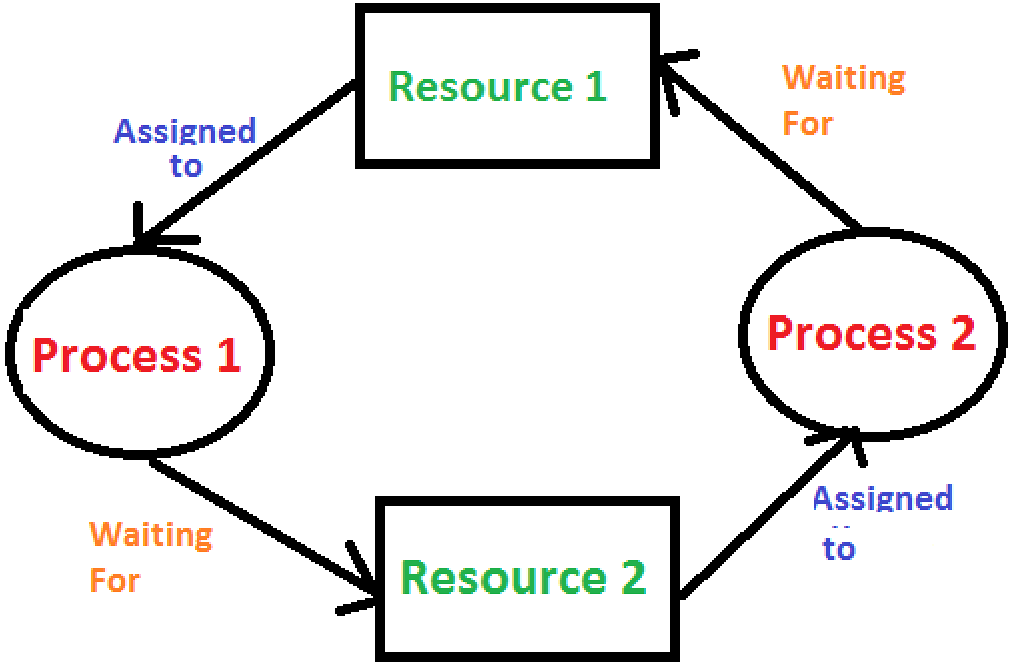
GIET University, Gunupur

Deadlocks

Introduction

- In a multiprogramming environment, several processes may compete for a finite number of resources.
- A process requests resources; if the resources are not available at that time, the process enters a waiting state.
- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes.
- This situation is called a **deadlock**.

- *“**Deadlock** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.”*



Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management

System Model

- A system can be modelled as a collection of limited resources that can be divided into different categories and allocated to a variety of processes, each with different requirements.
- Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.
- By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category.

- If there is some difference between the resources within a category, then that category must be subdivided further.
- For example, the term “printers” may need to be subdivided into “laser printers” and “color inkjet printers.”
- Some categories may only have one resource.

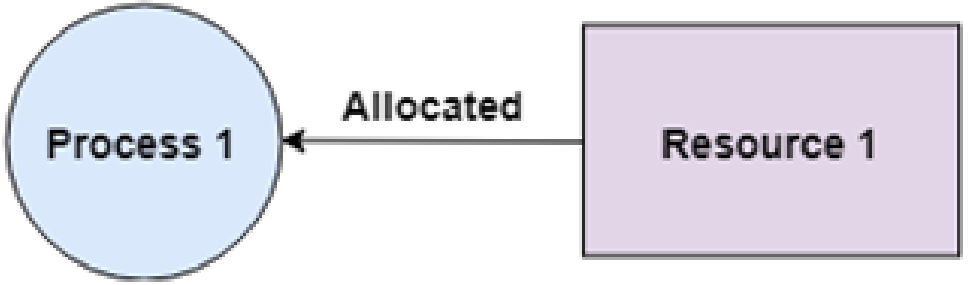
- The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources.
- Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores.)
- When every process in a set is waiting for a resource that is currently assigned to another process in the set, the set is said to be deadlocked.

- **Operations:** In normal operation, a process must request a resource before using it and release it when finished, as shown below.
- **Request:** If the request cannot be granted immediately, the process must wait until the resource(s) required to become available. The system, for example, uses the functions open(), malloc(), new(), and request ().
- **Use:** The process makes use of the resource, such as printing to a printer or reading from a file.
- **Release:** The process relinquishes the resource, allowing it to be used by other processes.

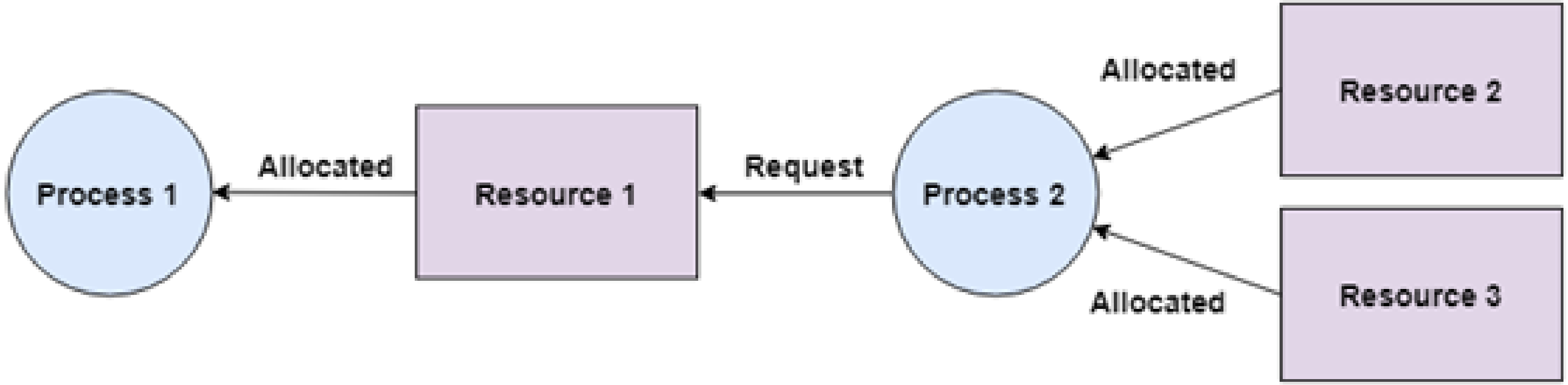
Deadlock Characterization

- In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.
- **Necessary Conditions :**
- There are four conditions that must be met in order to achieve deadlock as follows.
- **Mutual Exclusion:** At least one resource must be kept in a non-shareable state; if another process requests it, it must wait for it to be released.

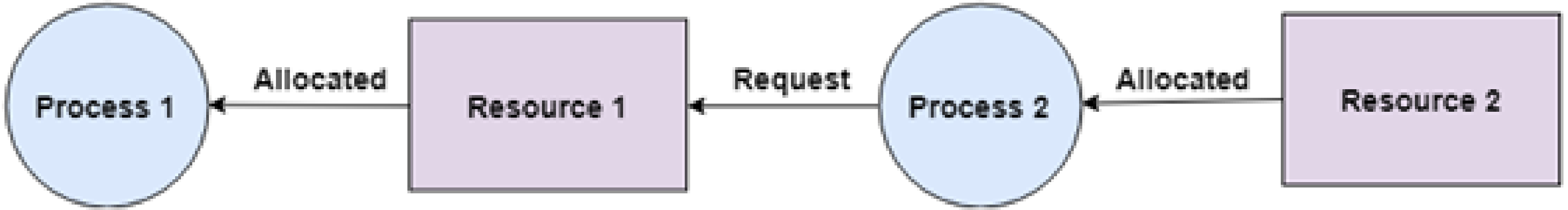
- **Mutual Exclusion:** There should be a resource that can only be held by one process at a time.
- In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



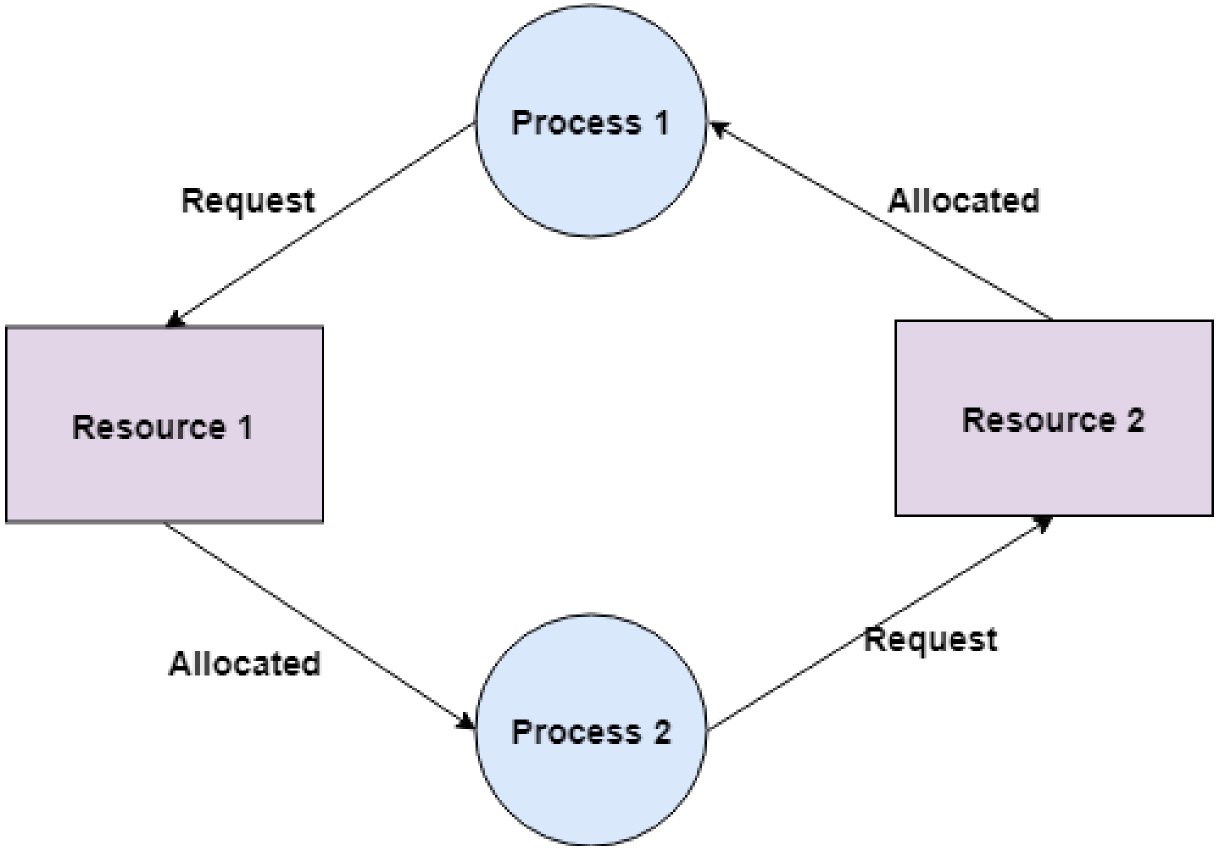
- **Hold and Wait**: A process can hold multiple resources and still request more resources from other processes which are holding them.
- In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



- **No Preemption:** Once a process holds a resource (i.e. after its request is granted), that resource cannot be taken away from that process until the process voluntarily releases it.
- In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



- **Circular Wait:** A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process.
- This forms a circular chain.
- For example: Process 1 is allocated Resource2 and it is requesting Resource 1.
- Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.

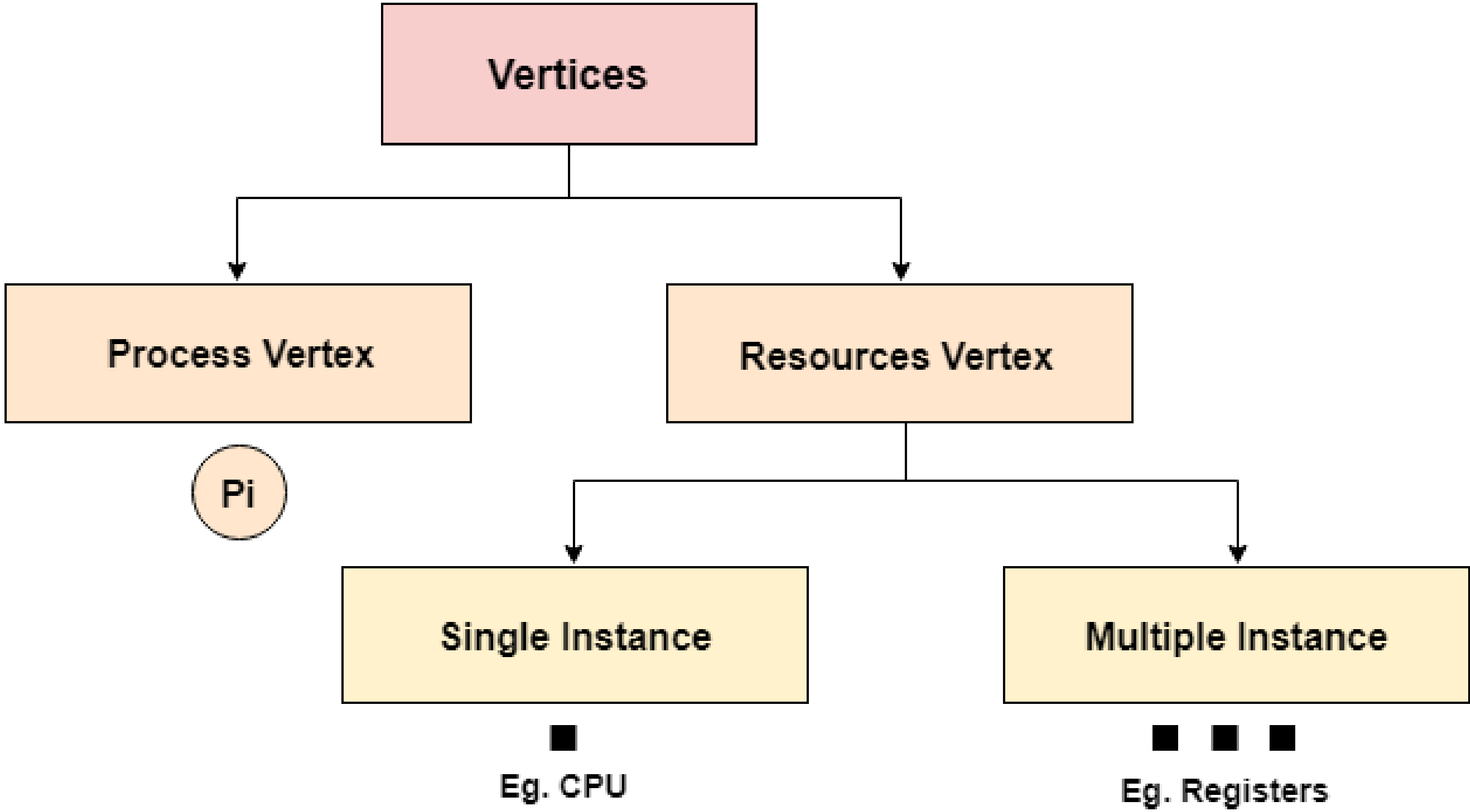


Resource-Allocation Graph (RAG)

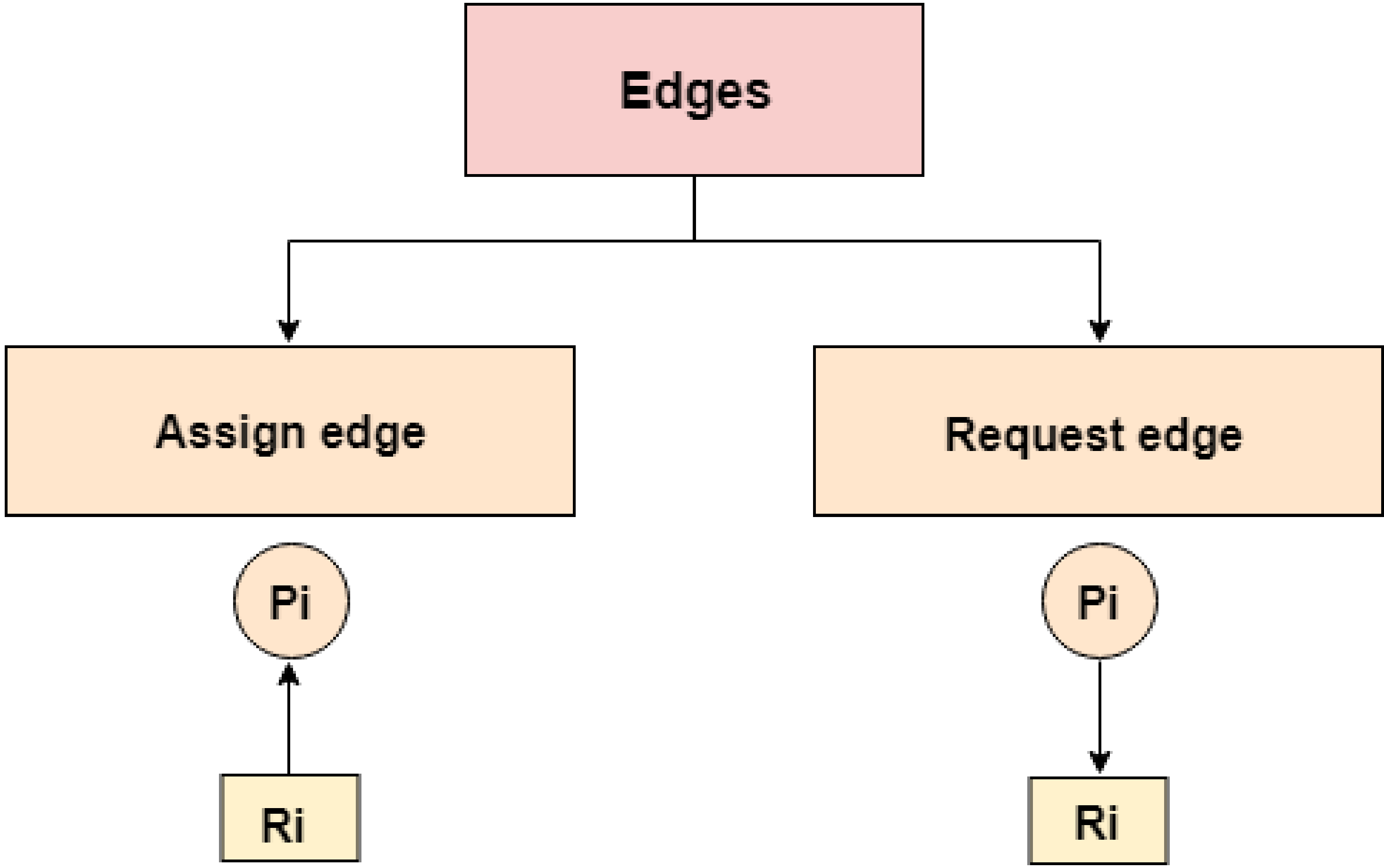
- As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.
- It also contains the information about all the instances of all the resources whether they are available or being used by the processes.
- In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle.

- Deadlocks can be described more precisely in terms of a directed graph called a **system resource-allocation graph**.
- This graph consists of a set of vertices *V* and a set of edges *E*.
- The set of vertices *V* is partitioned into two different types of nodes: $P = \{P1, P2, \ldots, Pn\}$, the set consisting of all the active processes in the system, and $R = \{R1, R2, \ldots, Rm\}$, the set consisting of all resource types in the system.

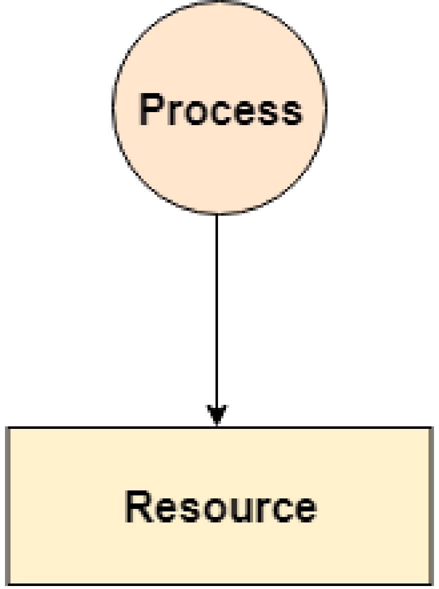
- Vertices are mainly of two types, Resource and process.
- Each of them will be represented by a different shape.



- A resource can have more than one instance.
- Each instance will be represented by a dot inside the rectangle.



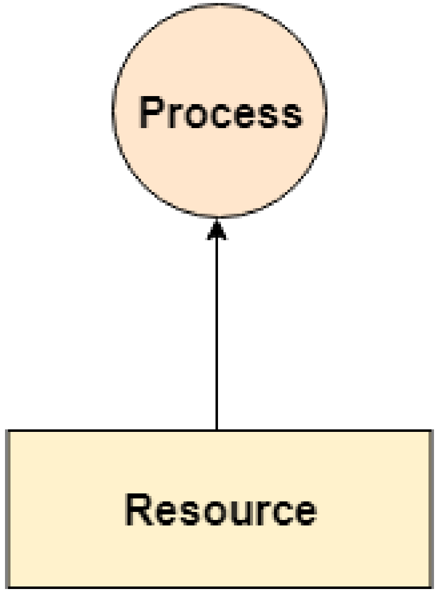
- Edges in RAG are also of two types, one represents **assignment** and other represents the **wait of a process** for a resource. The above image shows each of them.



Process is requesting for a resource

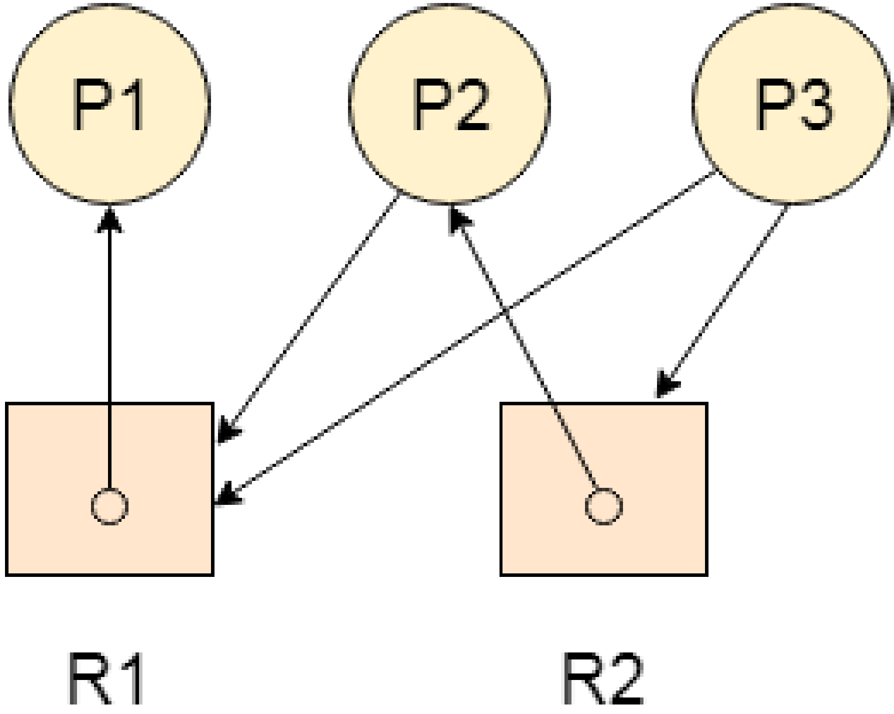
A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.



Resource is assigned to process

- Example
- Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.
- According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.
- The graph is deadlock free since no cycle is being formed in the graph.



Methods for Handling Deadlocks

- There are four approaches to deal with deadlocks
- Ignoring the Deadlock
- Deadlock Prevention
- Avoiding the Deadlock
- Deadlock detection and recovery

- **Ignoring the Deadlock**
- This approach is used by many operating systems where they assume that deadlock will never occur which means operating systems simply ignores the deadlock.
- This approach can be beneficial for those systems that are only used for browsing and for normal tasks.
- Thus ignoring the deadlock method can be useful in many cases but it is not perfect in order to remove the deadlock from the operating system.

- **Deadlock Prevention**
- As we have discussed in the above section, that all four conditions: Mutual Exclusion, Hold and Wait, No preemption, and circular wait if held by a system then causes deadlock to occur.
- The main aim of the deadlock prevention method is to violate any one condition among the four; because if any of one condition is violated then the problem of deadlock will never occur.
- As the idea behind this method is simple but the difficulty can occur during the physical implementation of this method in the system.

- **Avoiding the Deadlock**
- This method is used by the operating system in order to check whether the system is in a safe state or in an unsafe state.
- This method checks every step performed by the operating system.
- Any process continues its execution until the system is in a safe state.
- Once the system enters into an unsafe state, the operating system has to take a step back.
- Basically, with the help of this method, the operating system keeps an eye on each allocation, and make sure that allocation does not cause any deadlock in the system.

- **Deadlock detection and recovery**
- With this method, the deadlock is detected first by using some algorithms of the resource-allocation graph.
- This graph is mainly used to represent the allocations of various resources to different processes.
- After the detection of deadlock, a number of methods can be used in order to recover from that deadlock.

- One way is **preemption** by the help of which a resource held by one process is provided to another process.
- The second way is to **roll back**, as the operating system keeps a record of the process state and it can easily make a process roll back to its previous state due to which deadlock situation can be easily eliminate.
- The third way to overcome the deadlock situation is by killing one or more processes.

- **Advantages of Deadlock**
- Advantageous to the processes that perform a single burst of activity.
- No preemption is required.
- Convenient to apply to resources that can save and restore their states easily.
- Compile-time checks help apply it feasibly
- The system design solves problems so no run-time computation is required.

- **Disadvantages of Deadlock**
- Delay in process initiation.
- Knowledge of future resources is necessary.
- Frequent preemptions.
- Doesn’t allow incremental resource requests.
- There are inherent preemption losses.

Banker’s Algorithm

- Banker's Algorithm is a **resource allocation** and **deadlock avoidance algorithm.**
- It test all the request made by processes for resources and it checks for the safe state,
- If the system remains in the safe state it allows the request and if there is no safe state it doesn’t allow the request made by the process.

- **Inputs to Banker’s Algorithm:**
- Max need of resources by each process.
- Currently, allocated resources by each process.
- Max free available resources in the system.
- **The request will only be granted under the below condition:**
- If the request made by the process is less than equal to max need to that process.
- If the request made by the process is less than equal to the freely available resource in the system.

- Given A=10
- B=5
- C=7
- Check deadlock is there or not?

Example:

Process	Allocation			Maximum Need			Avilable			Remaining need		
										max. resources - allocated resources		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2	5	3	2	1	2	2
P3	3	0	2	9	0	2	7	4	3	6	0	0
P4	2	1	1	4	2	2	7	4	5	2	1	1
P5	0	0	2	5	3	3	7	5	5	5	3	1
	7	2	5				10	5	7			

Thank You