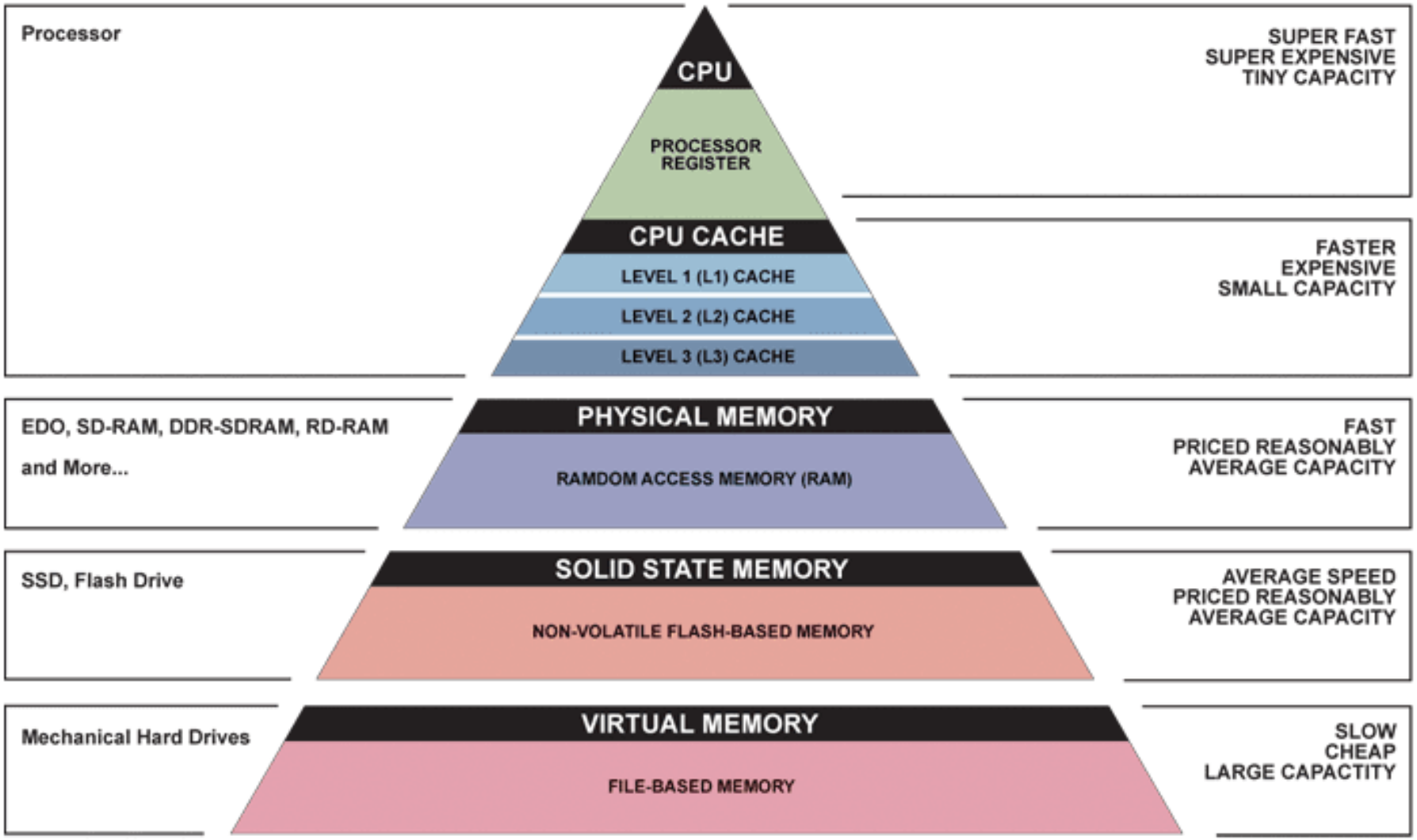
BCA Fourth Semester

Operating Systems

Unit -5 Memory Management [7 Hrs]

Memory Hierarchy



Memory Address

- In a computer system, the OS and the different processes reside in the memory unit of the system. **An address is used to identify a location in the memory unit uniquely.**

- **There are two types of addresses - logical address and physical address.**
- **The memory unit consists of physical addresses, while the CPU generates the logical addresses.**

Logical Address

- **The logical address refers to an address generated by the CPU during program execution**. They are used as a reference to access the physical address.
- **All logical addresses generated by a program are known as the logical address space.**
- As a logical address does not exist physically**, it is also referred to as a virtual address.**

Physical Address

- **The physical address refers to an actual physical location located in the memory unit.**
- **All physical addresses mapped to corresponding logical addresses are referred to as 'physical address space'.**
- User programs never see or deal with physical addresses.

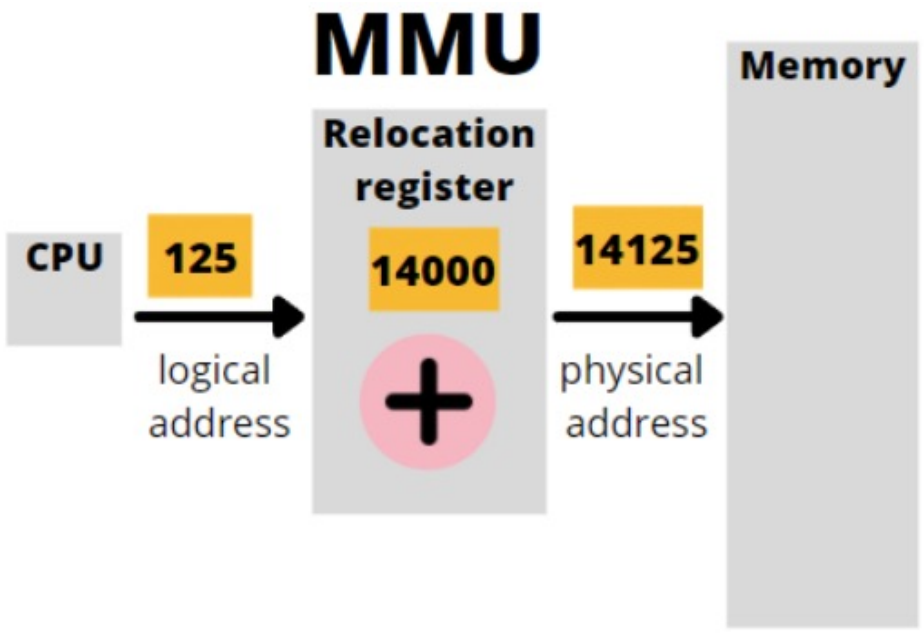
Logical vs Physical Address Space

Logical Address	Physical Address
It is an address generated by the CPU during program execution.	It refers to a physical location in the memory unit.
User programs deal with the logical address directly.	The user program never sees the physical address.
The user can access the physical address using the logical address.	Users cannot directly access the physical address.
The set of all logical addresses generated by a program are known as the 'logical address space'.	The set of all physical addresses are known as the 'physical address space'.
A logical address does not exist physically and is referred to as a 'virtual address'.	A physical address is a real location that exists in the memory unit.is

Memory Management Requirements - Relocation

- **Relocation is simply a mapping process of logical address to physical address.**

- **MMU or the memory-management unit is a hardware device that maps a logical address to its corresponding physical address.**
- Many memory mapping methods can be used for this purpose of mapping.



Dynamic relocation using relocation register

- In the above example, the mapping of logical to physical address is achieved using a simple MMU scheme known as **base-register scheme**.

- **The value of the base register or 'relocation register' is added with the logical address generated by the CPU.**

- **The sum of these two values is the physical address located in memory. Here the logical address '125' is added with relocation register value '14000' to obtain the physical address of '14125'.**

Memory Management Requirements

Protection

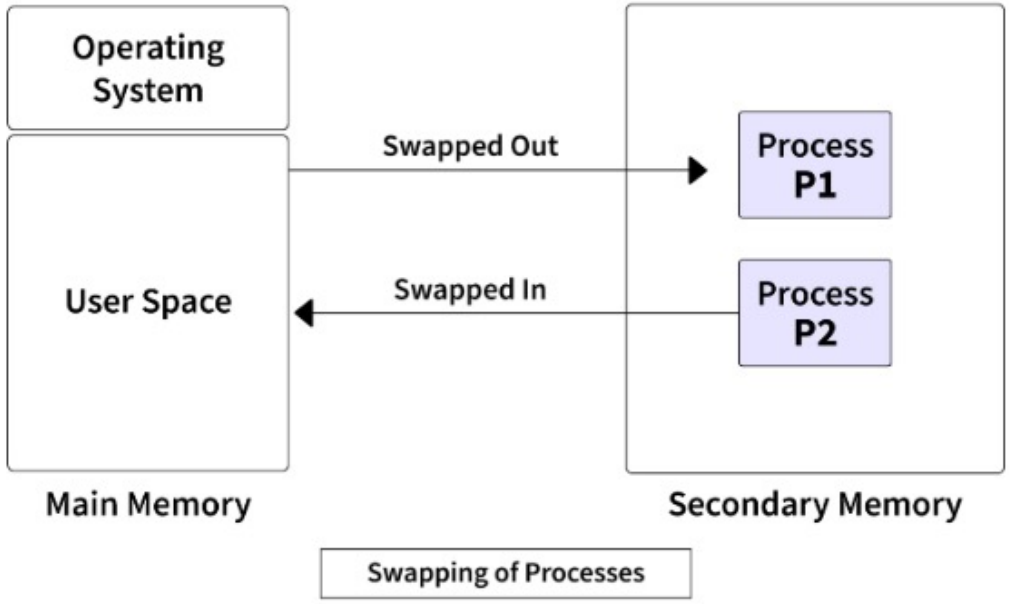
- Prevent processes from interfering with the OS or other processes.
- Processes should not be able to reference memory locations in another process without permission.
- Impossible to check absolute addresses in programs since the program could be relocated.
- Often integrated with relocation.

Sharing

- Allow several processes to access the same portion of memory (allow to share data).
- Better to allow each process (person) access to the same copy of the program rather than have their own separate copy.

Memory Management with Swapping

- **Swapping in OS is a memory management method that temporarily swaps out idle or blocked processes from main memory to secondary memory which ensures proper memory utilization.**
- **Swap In** is a method used to remove the process from secondary memory and restore it to primary memory.
- **Swap out** is a method used to remove the process from main memory and send it to secondary memory.



- **One of the main advantages of the swapping technique is that it provides proper RAM the utilization and ensures memory availability for every process.**
- **One of the main disadvantages of the swapping technique is that the algorithm used for swapping must be good enough otherwise it decreases the overall performance.**

Memory Management with Bitmaps and Linked List

Bitmaps

- With bitmap, memory is divided into allocation units.
- **Corresponding to each allocation unit there is a bit in a bitmap.**
  Bit is 0 if the unit is free.
  Bit is 1 if unit is occupied.
- The size of allocation unit is an important design issue, the smaller the size, the larger the bitmap.
- The main problem is that when it has been decided to bring a k unit process, the memory manager must search the bitmap to find a run of k consecutive 0 bits in the map.
- **Searching a bitmap for a run of a given length is a slow operation.**

Linked List

- **Another way to keep track of memory is to maintain a linked list of allocated and free memory segments, where segment either contains a process or is an empty hole between two processes.**

- **The memory of Fig1(a) is represented in Fig1(c) as a linked list of segments.**
- Each entry in the list specifies a hole (H) or process (P), the address at which it starts the length and a pointer to the next entry.

Memory Management with Bitmaps and Linked List

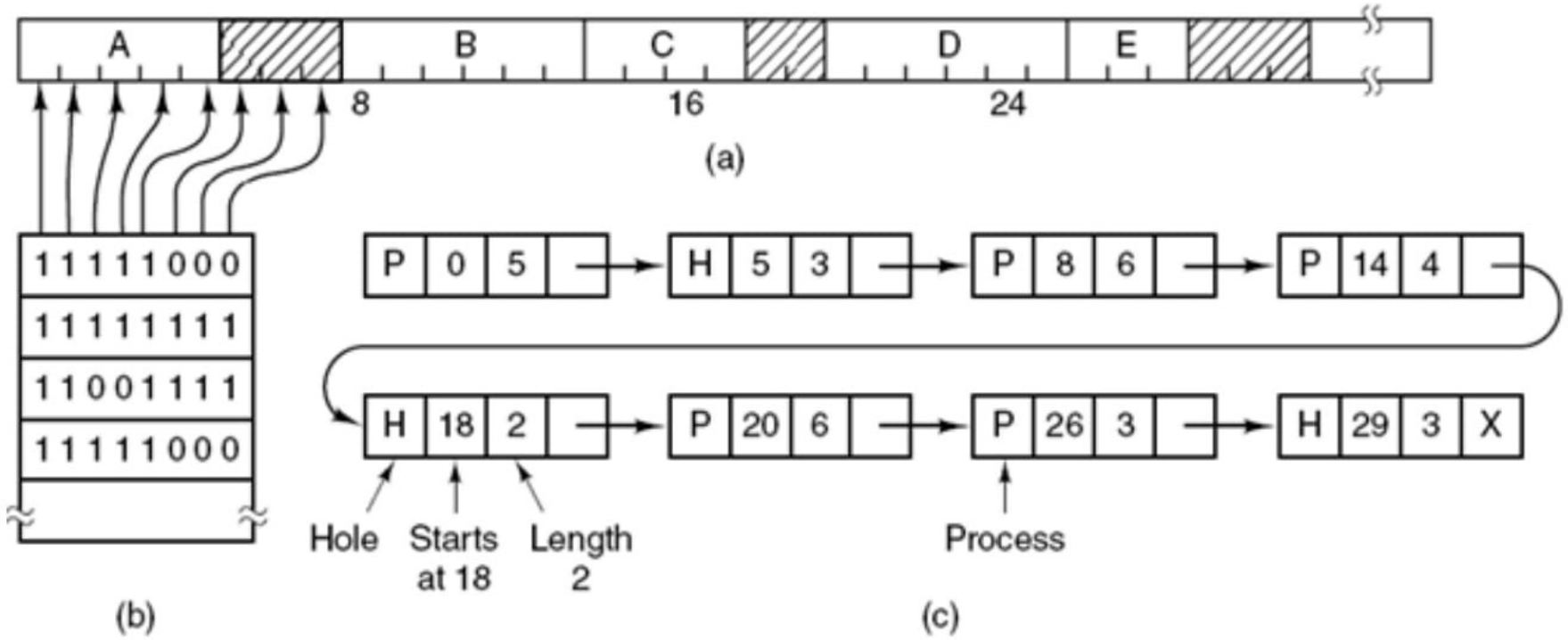


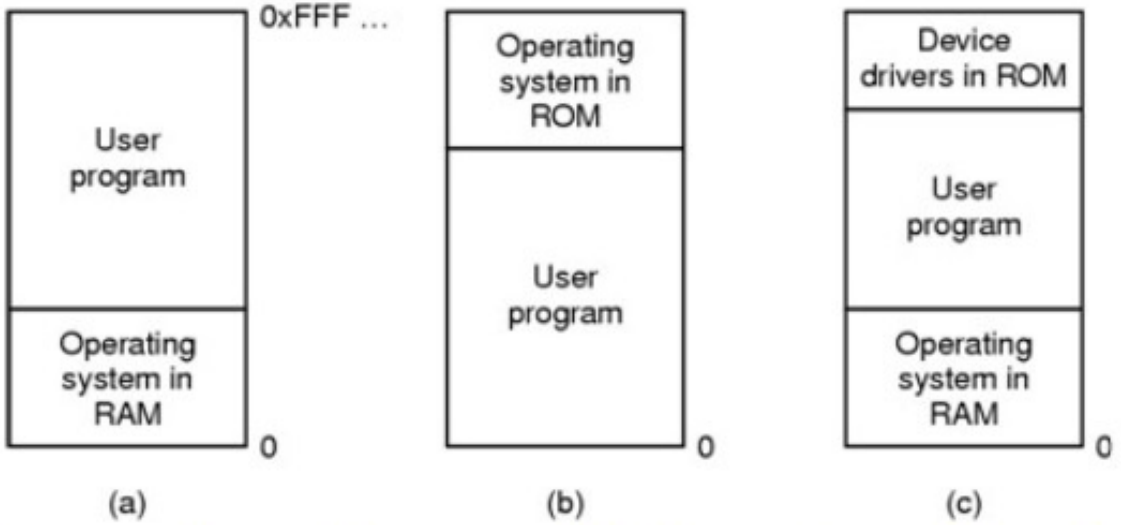
Figure1(a) A part of memory with five processes and three holes. The tick marks show the memory allocation units.
The shaded regions (0 in the bitmap) are free.

(b) The corresponding bitmap.
(c) The same information as a list.

Memory Management without Swapping

- **Entire process remains in memory from start to finish and does not move.**
- **The sum of the memory requirements of all jobs in the system cannot exceed the size of physical memory.**

Monoprogramming without Swapping or Paging



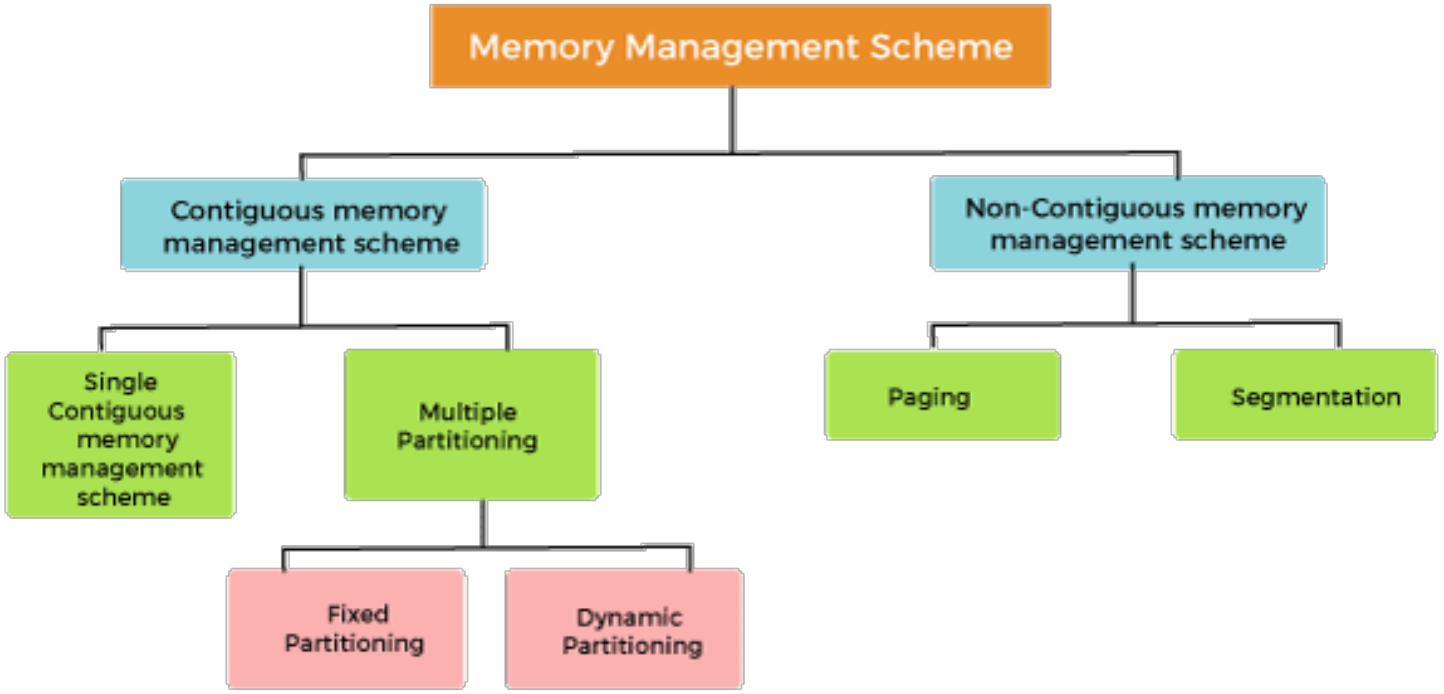
Model (a) was used on mainframes and minicomputers, and is rarely used any more.

Model (b) is used on some palmtop computers and embedded systems.

Model (c) was used by the early personal computers. The portion of the system in ROM is called BIOS (Basic Input Output System)

Except on simple embedded systems, monoprogramming is hardly used anymore.

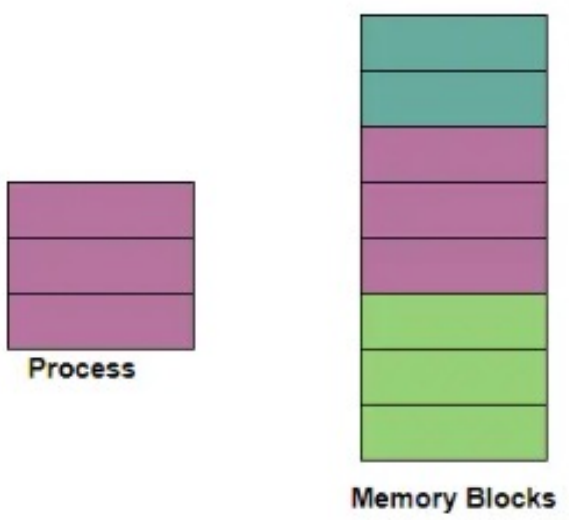
Memory Management Schemes



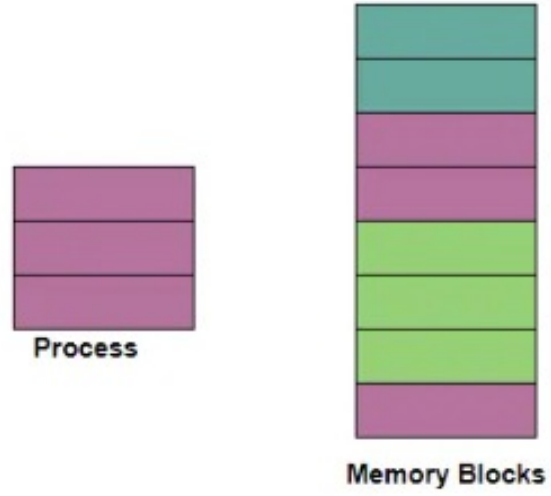
Classification of memory management schemes

Contiguous vs Non Contiguous Allocation

Basis	Contiguous Memory Allocation	Non – Contiguous Memory Allocation
Blocks of memory	Allocates consecutive blocks of memory to the process.	Allocates a single block of memory to the process.
Speed	It is faster.	It is slower.
Fragmentation	Both internal and external fragmentation.	Only external fragmentation.
Scheme	Fixed–sized partitions and variable-sized partitions are the two schemes of contiguous memory allocation.	Paging and segmentation are the schemes of non – contiguous memory allocation.
Wastage of memory	Memory is wasted here.	No memory wastage in this kind of allocation.



Contiguous Allocation



Non-Contiguous Allocation

Contiguous Memory Allocation Techniques

- Whenever a process has to be allocated space in the memory, following the contiguous memory allocation technique, we have to allot the process a continuous empty block of space to reside.

This allocation can be done in two ways:

1. Fixed-size Partition Scheme
2. Variable-size Partition Scheme

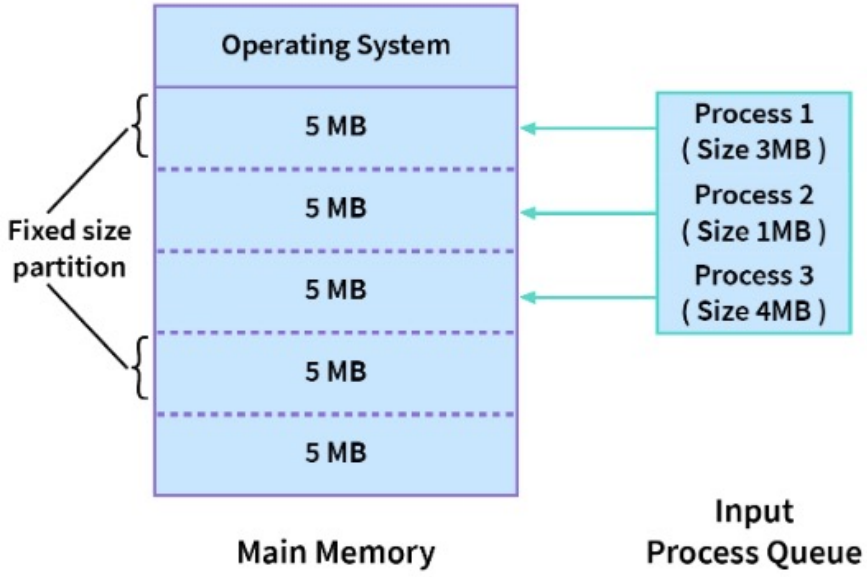
Fixed-size Paritition Scheme

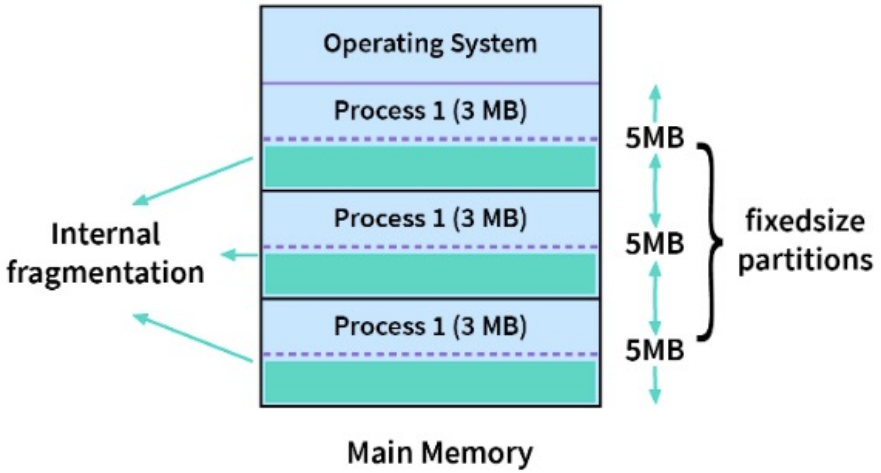
- In this type of contiguous memory allocation technique, **each process is allotted a fixed size continuous block in the main memory**.

- That means there will be continuous blocks of fixed size into which the complete memory will be divided, and each time a process comes in, it will be allotted one of the free blocks.

- Because irrespective of the size of the process, each is allotted a block of the same size memory space.

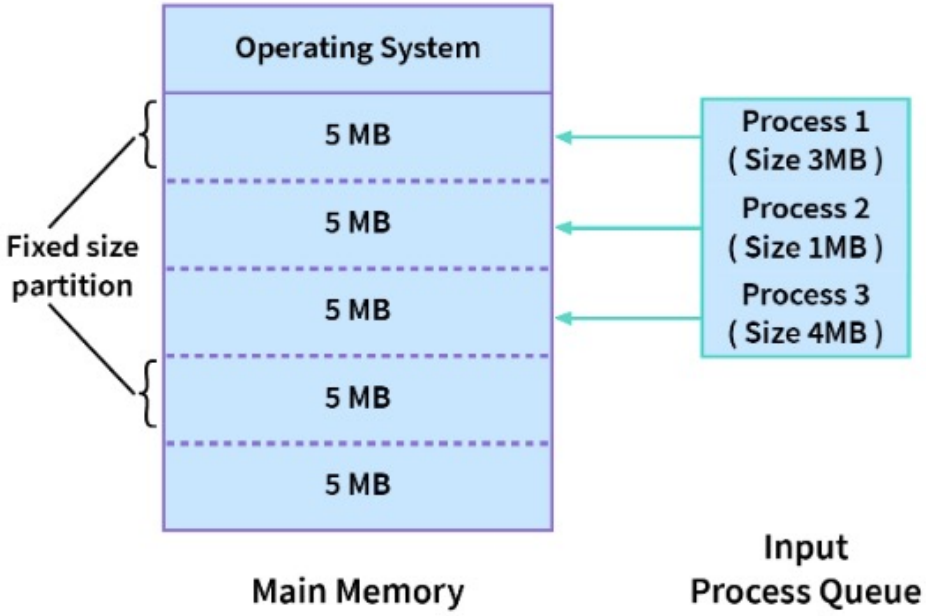
- **This technique is also called static partitioning.**





Variable-size Paritition Scheme

- In this type of contiguous memory allocation technique, **no fixed blocks or partitions are made in the memory.**
- **Instead, each process is allotted a variable-sized block depending upon its requirements.**
- **That means, whenever a new process wants some space in the memory, if available, this amount of space is allotted to it.**
- Hence, the size of each block depends on the size and requirements of the process which occupies it.



In the diagram above, there are no fixed-size partitions. Instead, the first process needs 3MB memory space and hence is allotted that much only. Similarly, the other 3 processes are allotted only that much space that is required by them.

As the blocks are variable-sized, which is decided as processes arrive, this scheme is also called Dynamic Partitioning.

Contiguous Memory Allocation Strategies

- Processes that have been assigned continuous blocks of memory will fill the main memory at any given time.
- **However, when a process completes, it leaves behind an empty block known as a hole.**
- **This space could also be used for a new process.**
- Hence, the main memory consists of processes and holes, and any one of these holes can be allotted to a new incoming process.
- **We have three strategies to allot a hole to an incoming process:**

First-Fit

Best-Fit

Worst-Fit

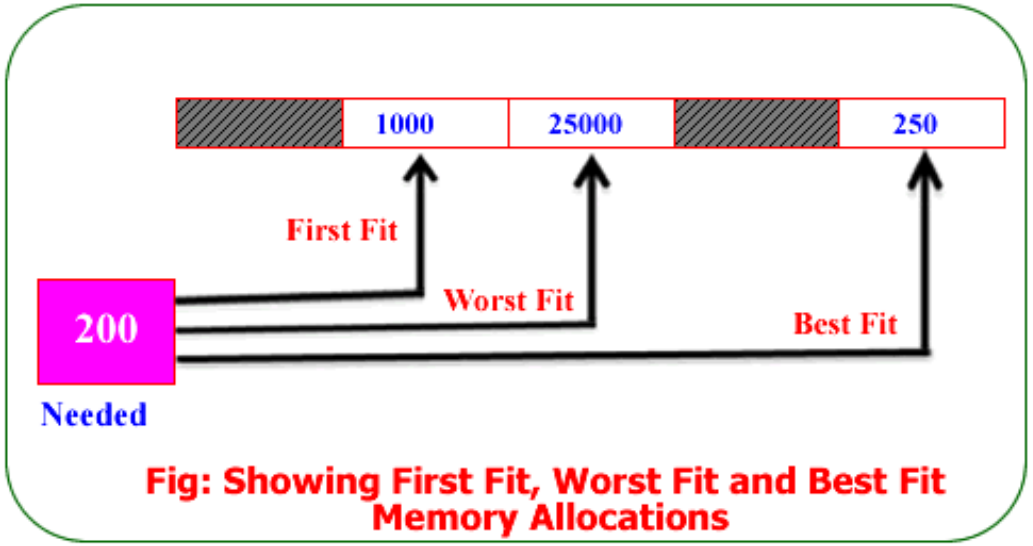


Fig: Showing First Fit, Worst Fit and Best Fit Memory Allocations

Contiguous Memory Allocation Strategies

First Fit

- **In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process.**
- It finishes after finding the first suitable free partition.

Best Fit

- **The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process.**
- This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

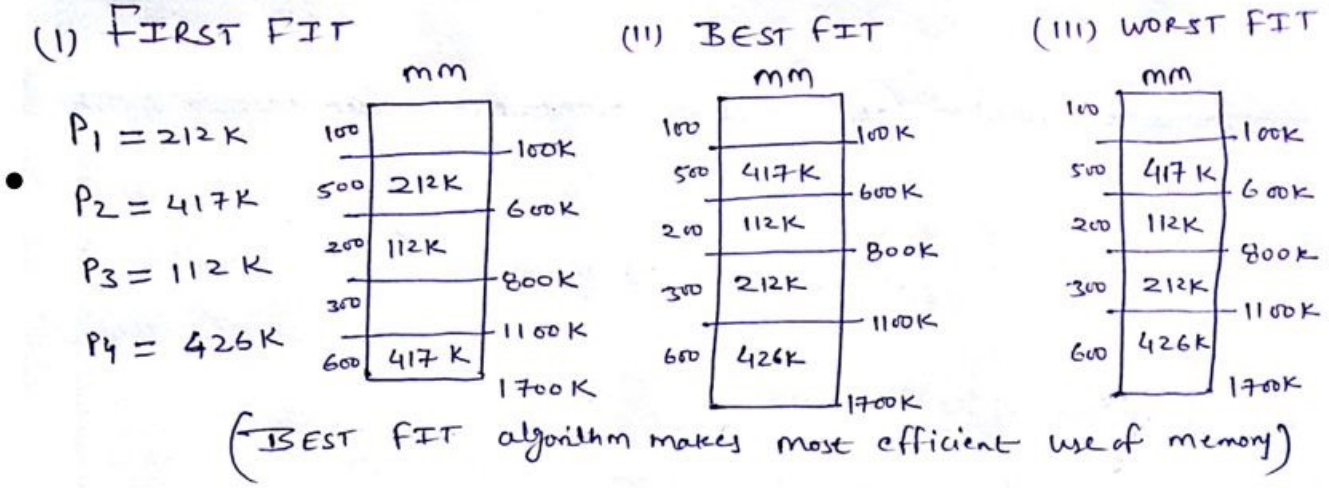
Worst fit

- **In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful.**
- It is the reverse of best fit.

Example 1

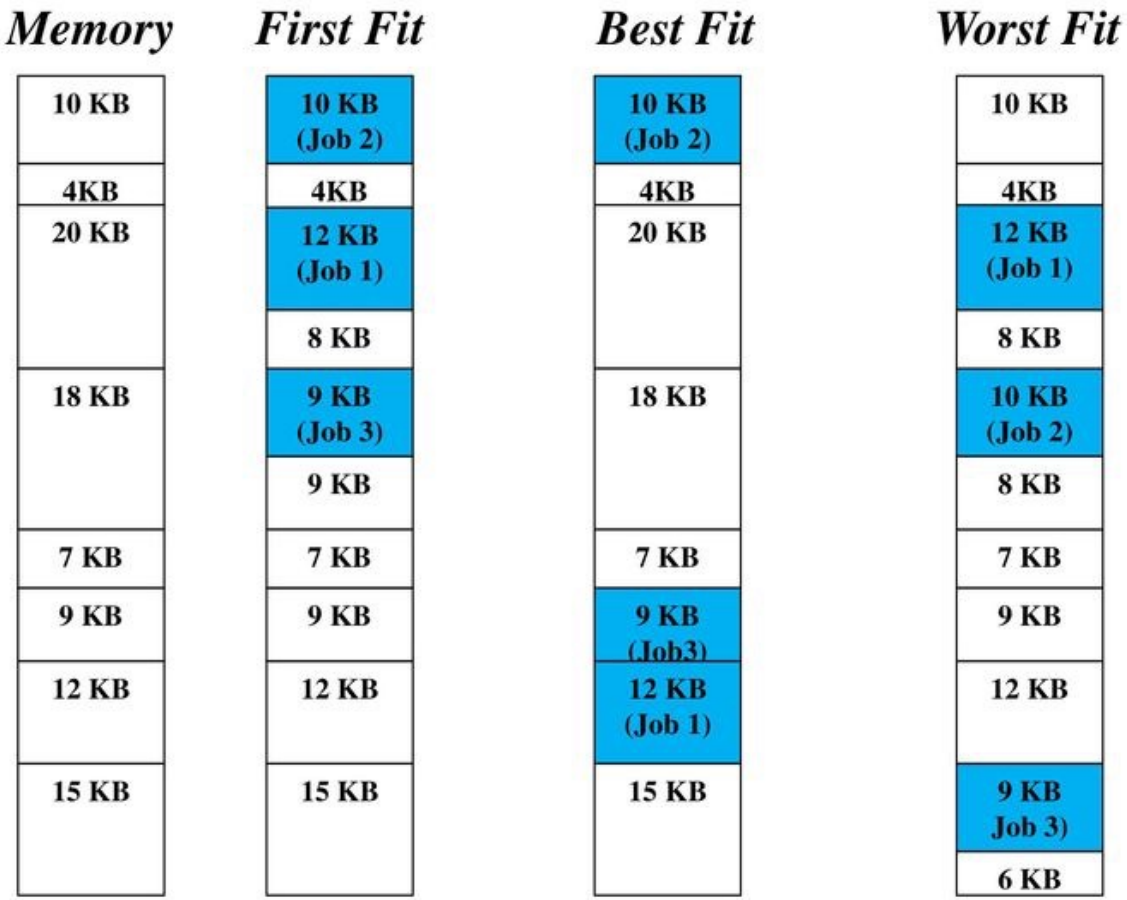
Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?

- Given memory partition
  - 100k, 500k, 200k, 300k, 600k (in order)
- Placed processes of
  - 212k, 417k, 112k, 426k (in order)



Example 2

Consider a swapping system in which memory consists of the following hole sizes in memory order: 10KB, 4KB, 20KB, 18KB, 7KB, 9KB, 12KB, and 15KB. Which hole is taken for successive segment requests of 12KB, 10KB, 9KB for *first fit*? Now repeat the question for *best fit, and worst fit*.



Fragmentation

- **In contiguous memory allocation, this loading and unloading of processes cause many fragments of the memory that can't be allotted to the incoming new process.**
- **This condition is called fragmentation in OS.**

Fragmentation in OS is of the following types:-

- Internal Fragmentation
- External Fragmentation

Internal Fragmentation

- **Internal Fragmentation is a problem that occurs when the process is allocated to a memory block whose size is more than the size of that process and due to which some part of the memory is left unused.**

- Thus the space wasted inside the allocated memory block is due to the restriction on the allowed sizes of allocated blocks.

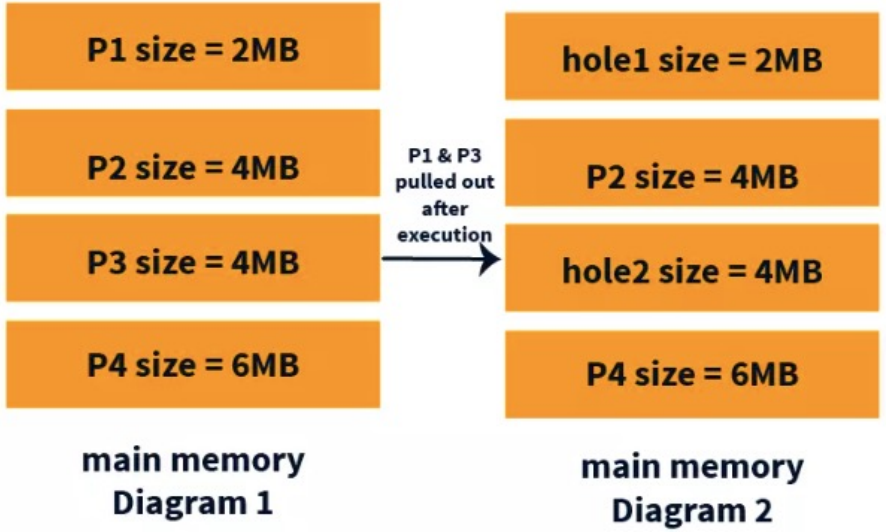
In the above diagram, **the difference between memory allocated and required space or memory by the process is called internal fragmentation.**

Solution: Best Fit Allocation

External Fragmentation

- **This is a condition where we have enough memory for the incoming process but that memory is not contiguous and hence cannot be allotted to the process. Hence the unused memory gets wasted. This causes external fragmentation.**

Example: **Suppose we have a RAM of size 16MB and 4 processes p1,p2,p3 and p4 with size 2MB,4MB,4MB and 6MB arrives respectively**. Since we don't have any partitions we will allocate memory to the processes in a contiguous manner. After allocation the RAM will look as in diagram1.
Now after that if process p1 and p3 gets pulled out from the RAM after they finished execution. The RAM will look as in diagram2.



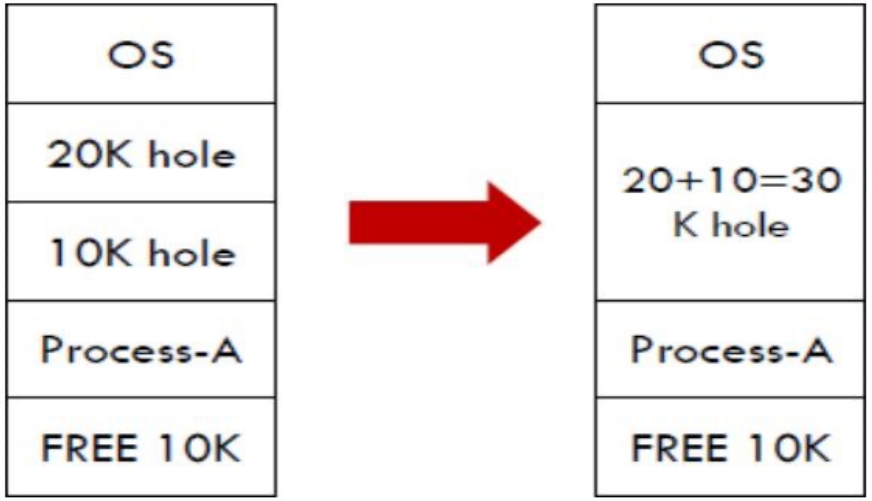
You can clearly see that we have two holes created of size 2MB and 4MB. Now if a process of size 6MB comes that will not be loaded into RAM because we don't have contiguous memory of size 6MB available although we have 6MB free in RAM. This is external fragmentation in OS.

Solution: Paging, Segmentation & Compaction

Coleasing and Compaction

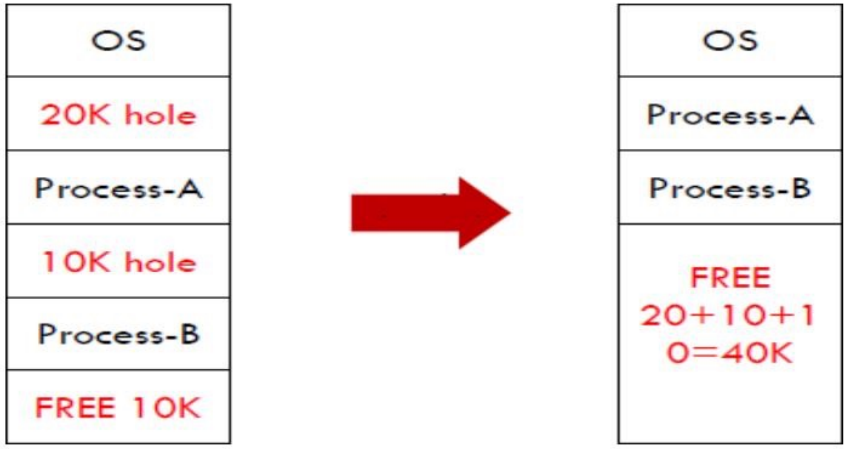
Coalescing

- **The process of merging two adjacent holes to form a single larger hole is called coalescing.**



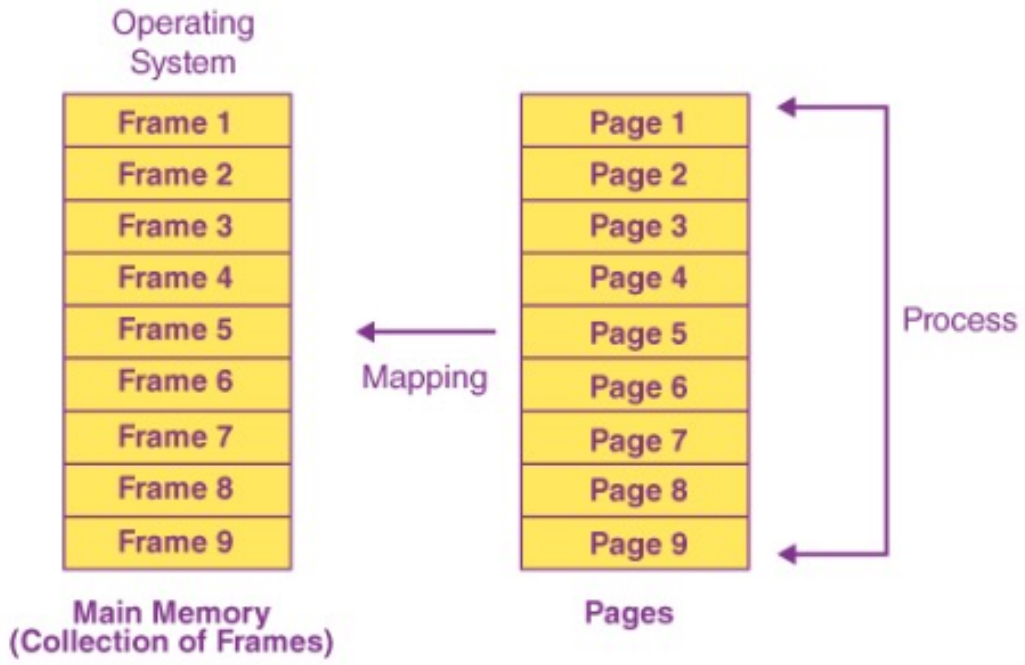
Compaction

- Even when holes coalesce, no individual hole may be large enough to hold the job, although the sum of holes is larger than the storage required for a process.
- **It is possible to combine all the holes into one big one by moving all the processes downward as far as possible; this technique is called memory compaction.**



Paging

- Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- **The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.**
- One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
- **Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.**



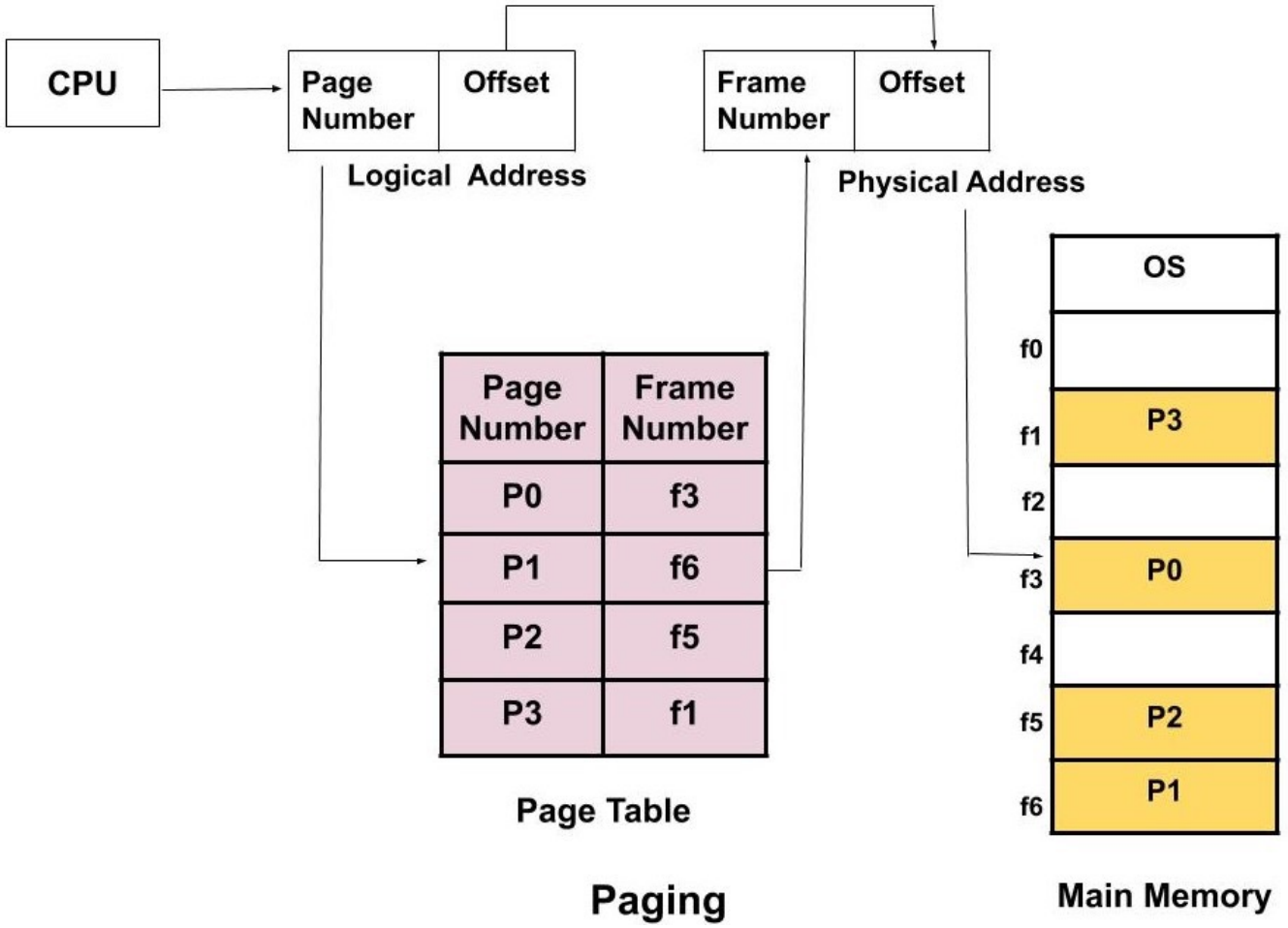
Features of paging:

- Mapping logical address to physical address.
- Page size is equal to frame size.
- Number of entries in a page table is equal to number of pages in logical address space.
- The page table entry contains the frame number.
- All the page table of the processes are placed in main memory.

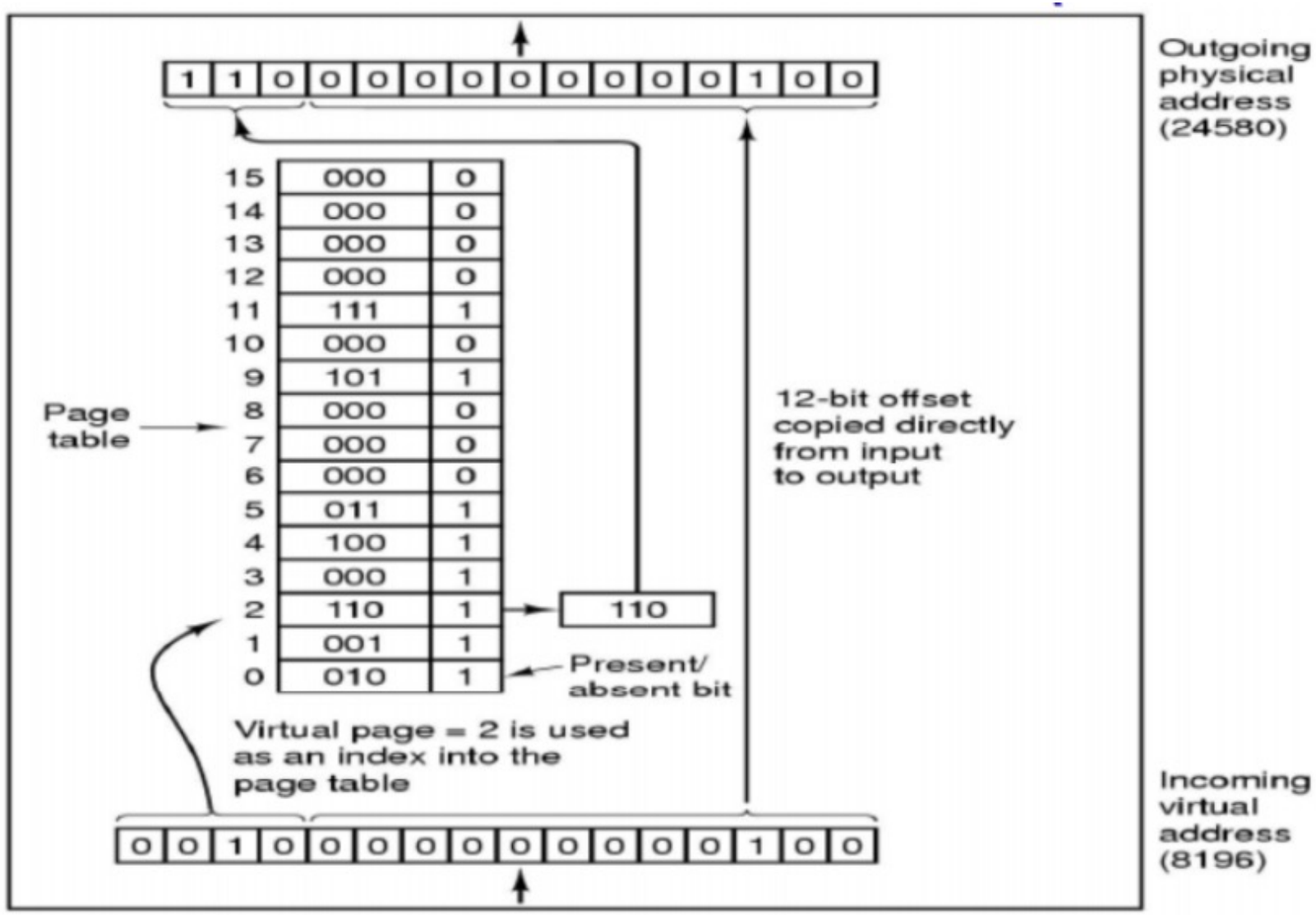
Paging Hardware

Address generated by CPU is divided into:
Page number (p) - used as an index into a page table which contains base address of each page in physical memory.
Page offset (d) - combined with base address to define the physical memory address that is sent to the memory unit.

page number	page offset
p	*d*



Address Translation



Page Table

- **The data structure that is used by the virtual memory system to store the mapping between the physical and logical addresses is known as Page Table.**

- **The page table provides the corresponding page number where the page is supposed to be stored.**

Characteristics of a Page Table:

- Page table is **stored in the main memory**.
- The **number of entries in the page table is equal to the number of pages** the processes are divided into.
- Each process has its page table.
- **Page Table Base Register or PTBR** holds the base address for the page table.

Page Table Issues

Page table is kept in main memory

Efficiency of mapping.

If a particular instruction is being mapped, the table lookup time should be very small than its total mapping time to avoid becoming CPU idle.
What would be the performance, if such a large table have to load at every mapping.

Size of page table

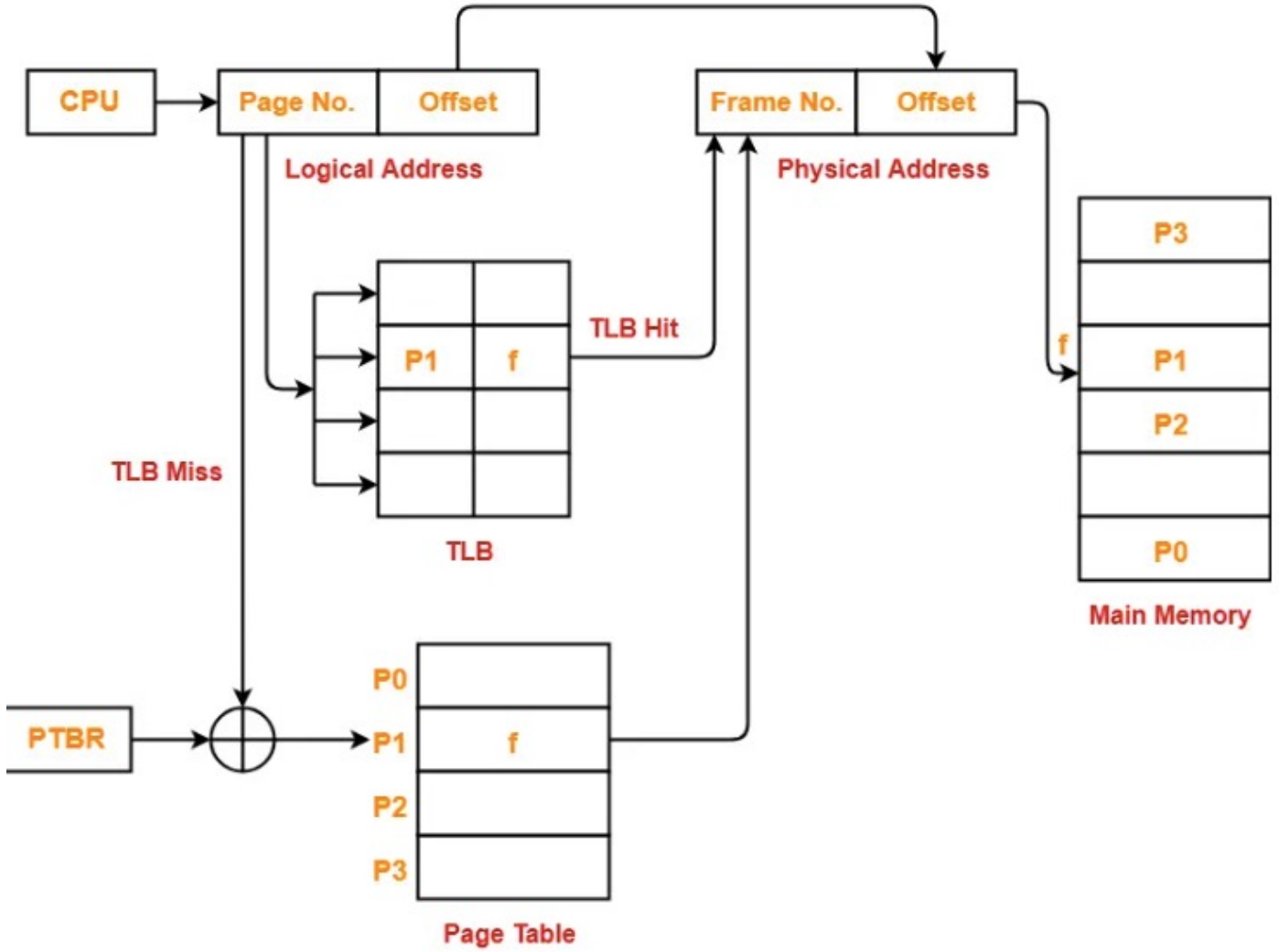
Most modern computers support a large virtual-address space (2^32 to 2^64). If the page size is 4KB, a 32-bit address space has 1 million pages. With 1 million pages, the page table must have 1 million entries.

think about 64-bit address space!

How to handle these issues?

Solution using TLB

- **Translation Lookaside Buffer (TLB)** is a solution that tries to reduce the effective access time. **Being a hardware, the access time of TLB is very less as compared to the main memory.**



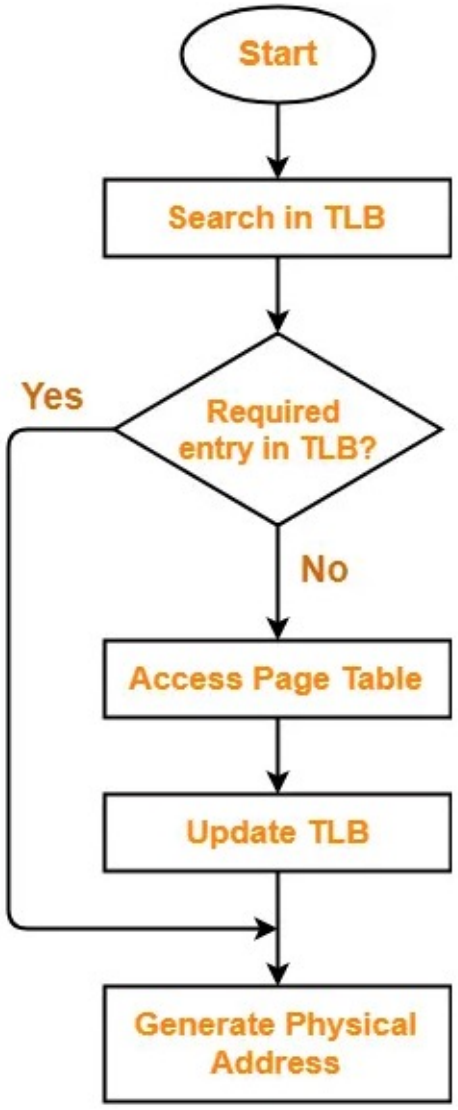
Solution using TLB

Case-01: If there is a TLB hit-

- If TLB contains an entry for the referenced page number, a TLB hit occurs.
- In this case, TLB entry is used to get the corresponding frame number for the referenced page number.

Case-02: If there is a TLB miss-

- If TLB does not contain an entry for the referenced page number, a TLB miss occurs.
- In this case, page table is used to get the corresponding frame number for the referenced page number.
- Then, TLB is updated with the page number and frame number for future references.



Flowchart

Structure of Page Table

- **Structure of page table simply defines, in how many ways a page table can be structured.**

- Well, the paging is a memory management technique where a large process is divided into pages and is placed in physical memory which is also divided into frames.

- Frame and page size is equivalent. The operating system uses a page table to map the logical address of the page generated by CPU to its physical address in the main memory.

Four common methods we use to structure a Page Table:

- **Hierarchical Page Table**
- **Hashed Page Table**
- **Inverted Page Table**

Hierarchical Page Table

- **There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.**

- **Hierarchical paging or multilevel paging is a type of paging where the logical address space is broken up into multiple page tables.**

- It is one of the simplest techniques and for this,
  a two-level or
  three-level page table is used.

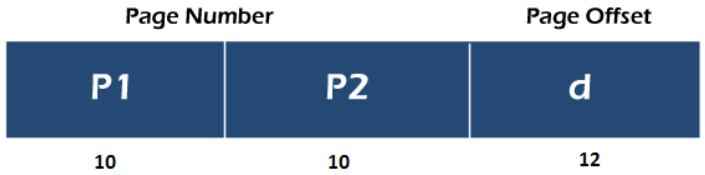
Two Level Page Table

- **Two-level paging in which a page table itself is paged. So we will have two-page tables' inner page table and outer page table.**

- **Consider a system having 32-bit logical address space and a page size of 1 KB. It is further divided into Page Number consisting of 20 bits and Page Offset consisting of 12 bits.**

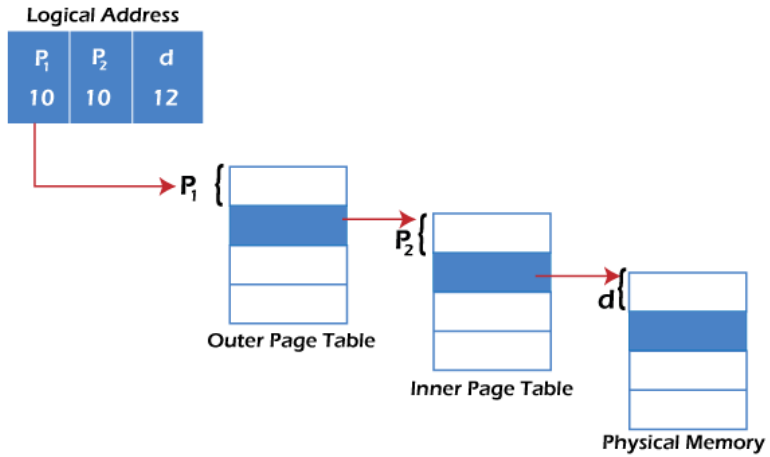
- **As we page the Page table, the page number is further divided into, Page Numbers consisting of 10 bits and Page Offset consisting of 12 bits.**

- **Thus the Logical address is as follows:**



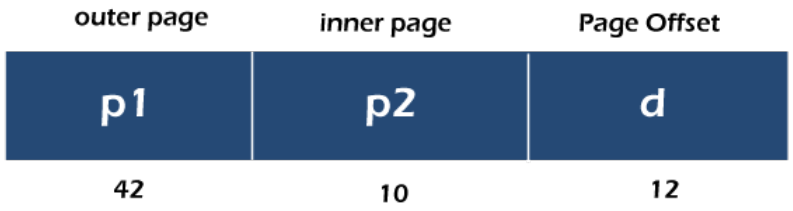
In the above image, P1 is an index into the ***Outer Page*** table, and P2 indicates the displacement within the page of the ***Inner page*** Table.

As address translation works from outer page table inward so is known as ***forward-mapped Page Table***.

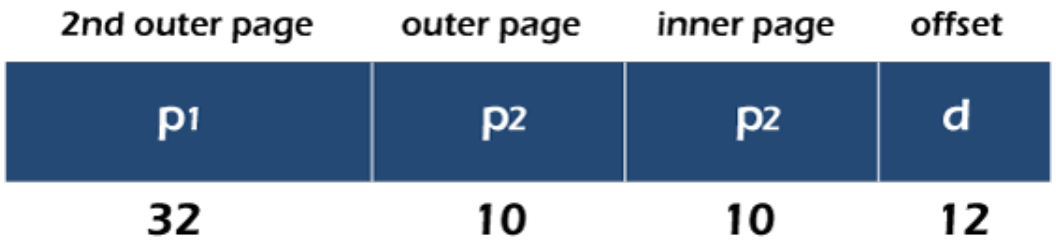


Three Level Page Table

- **A two-level paging scheme is not appropriate for a system with a 64-bit logical address space.**
- **Suppose that the page size is 4KB. If we use the two-page level scheme, then the addresses will look as follows:**



Thus, to avoid such a large table, there is a solution to divide the outer page table, and then it will result in a ***Three-level page table*** **as shown below.**



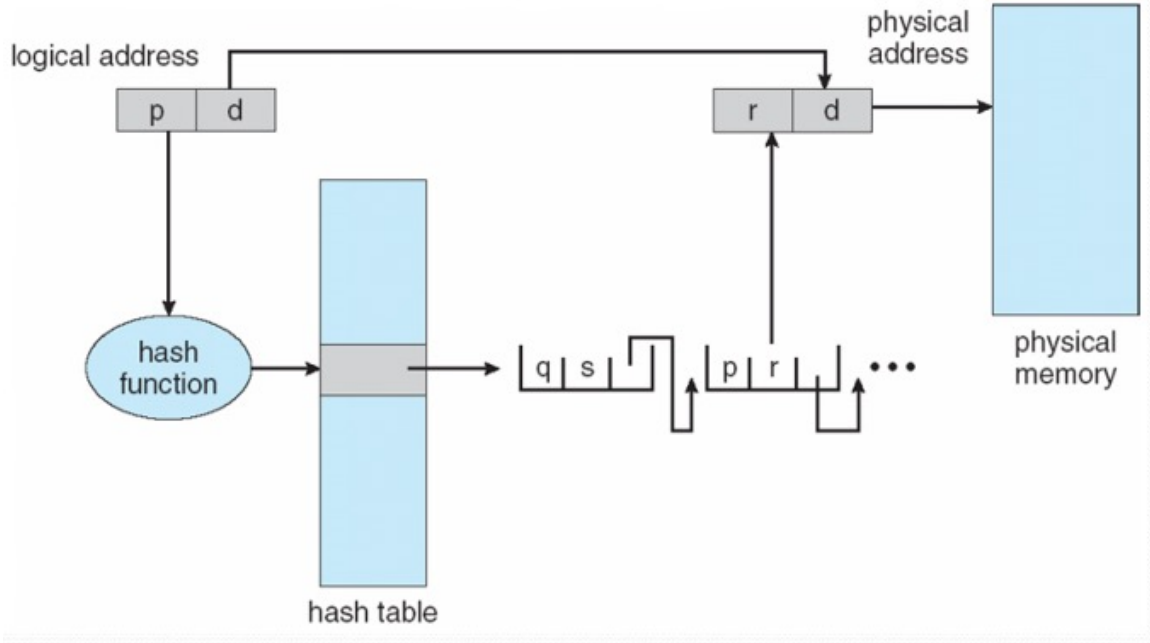
Hashed Page Table

A common approach for handling address space larger than 32-bit.

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

Each element consists of three fields: *virtual-page-number, value of mapped page frame, and a pointer to the next element.*

The virtual address is hashed into the hash table, if there is match the corresponding page frame is used, if not, subsequent entries in the linked list are searched.

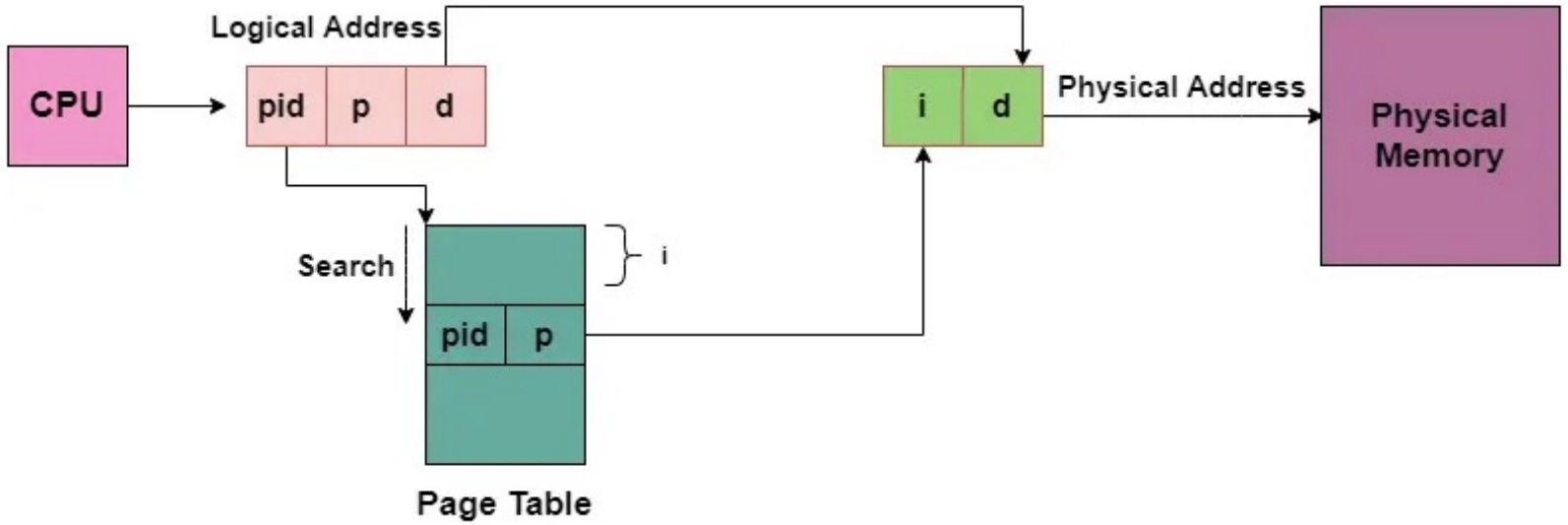


Inverted Page Table

The Inverted Page table basically combines A page table and A frame table into a single data structure.

- There is one entry for each virtual page number and a real page of memory
- And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
- Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

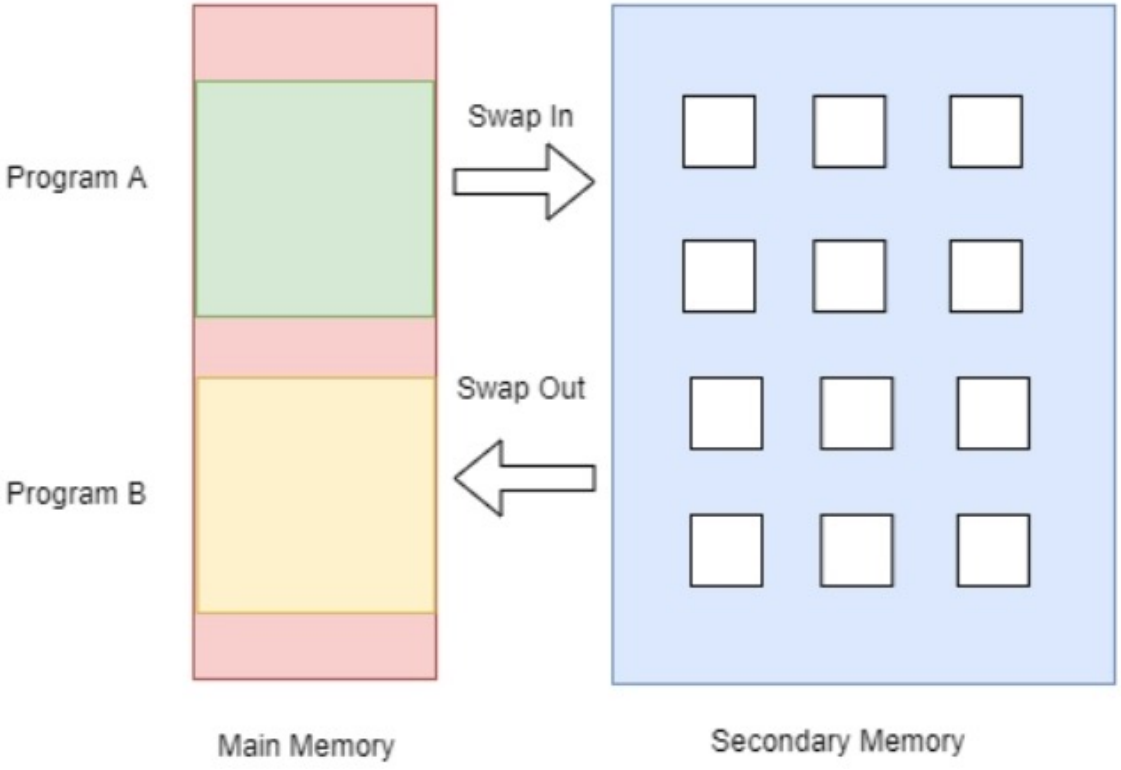
Given below figure shows the address translation scheme of the Inverted Page Table:



Demand Paging

Demand Paging is a method in which a page is only brought into main memory when the CPU requests it. At first, just those pages are loaded directly required by the operation. Pages that are never accessed are thus never loaded into physical memory.

- **The demand paging system is similar to the swapping paging system in that processes are mostly stored in the main memory (usually on the hard disk).**
- As a result, demand paging is a procedure that **addresses the problem above just by shifting pages on demand.**
- **Lazy swapper** is another name for this (It never swaps the page into the memory unless it is needed).
- A pager is a kind of swapper that deals with the individual pages of a process.



Demand Paging

Advantages

- **Memory** can be put to **better use**.
- If we use demand paging, then we can have a **large virtual memory**.
- By using demand paging, we can **run programs that are larger than physical memory**.
- In demand paging, **there is no requirement for compaction**.
- In demand paging, **the sharing of pages is easy**.

Disadvantages

- **Internal fragmentation** is a possibility with demand paging.
- It **takes longer to access memory** (page table lookup).
- **Memory requirements**

Working of Demand Paging

- **When the CPU requests access to any page, the page table is used to find the page in the main memory.**
- **If the page is found on the main memory, it is good, and if it is not, then a page fault occurs.**

- **Page fault is when the CPU wants to access the page from the main memory, but it is not present in the main memory.**

- **For this swapped-in is used, the swapped-in is used to swap from the secondary memory.**
- Swapped-in refers to moving a program back to the hard drive from the main memory, or RAM.
- However, if the page is already in the main memory, it is retrieved from there.
- The secondary memory is loaded with other pages.
- There are also valid and invalid bits, which are used to check whether the page is present in the main memory.
- Valid bits mean that the page is legal and present in the memory, and invalid bits mean the page is not valid or it is not present in the memory.

Page Replacement and Page Fault

- **There will be a miss if the referenced page is not present in the main memory; this is known as a page miss or page fault.**
- **The CPU must look up the missing page in secondary memory**.
- When the **number of page faults is significant, the system's effective access time increases** dramatically.
- **Page replacement is referred to a scenario in which a page from the main memory should be replaced by a page from secondary memory.**
- **Page replacement occurs due to page faults.**

Page Replacement Algorithms

- **Page Replacement Algorithm decides which page to remove, also called swap out when a new page needs to be loaded into the main memory.**

- **Page Replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.**

- **A page replacement algorithm tries to select which pages should be replaced so as to minimize the total number of page misses.**

- **The fewer is the page faults the better is the algorithm for that situation.**

Some Page Replacement Algorithms are:

First In First Out (FIFO)
Least Recently Used (LRU)
Optimal Page Replacement
Second Chance

First In First Out (FIFO)

- **This is the simplest page replacement algorithm.**
- **When there is a need for page replacement, the FIFO algorithm, swaps out the page at the front of the queue, that is the page which has been in the memory for the longest time.**

> *Advantages*
>
> - Simple and easy to implement.
> - Low overhead.

> *Disadvantages*
>
> - Poor performance.
> - Doesn't consider the frequency of use or last used time, simply replaces the oldest page.
> - Suffers from Belady's Anomaly(i.e. more page faults when we increase the number of page frames).

Example of FIFO

Consider the page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with frame size 4(i.e. maximum 4 pages in a frame).

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3
M	M	M	M	M	M	H	H	M	M	M	H

M = Miss
H = Hit

Total Page Fault = 9

Page Fault ratio = 9/12 i.e. total miss/total possible cases

Least Recently Used (LRU)

- Least Recently Used page replacement algorithm keeps track of page usage over a short period of time.
- It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.
- **In LRU, whenever page replacement happens, the page which has not been used for the longest amount of time is replaced.**

> ***Advantages***
>
> - Efficient.
> - Doesn't suffer from Belady's Anomaly.

> ***Disadvantages***
>
> - Complex Implementation.
> - Expensive.
> - Requires hardware support.

Example of LRU

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	3	3	3
			4	4	4	4	4	6	6	6	6
M	M	M	M	M	M	H	H	M	H	M	H

M = Miss
H = Hit

Total Page Fault = 8

Page Fault ratio = 8/12

Optimal Page Replacement (OPR)

- **Optimal Page Replacement algorithm is the best page replacement algorithm as it gives the least number of page faults.**

- **In this algorithm, pages are replaced which would not be used for the longest duration of time in the future, i.e., the pages in the memory which are going to be referred farthest in the future are replaced.**

> ***Advantages***
>
> - Easy to Implement.
> - Simple data structures are used.
> - Highly efficient.

> ***Disadvantages***
>
> - Requires future knowledge of the program.
> - Time-consuming.

Example of Optimal

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	1	1	1	1	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	5	5	5	5	5	5	5	5
M	M	M	M	M	H	H	H	M	H	H	H

M = Miss
H = Hit

Total Page Fault = 6

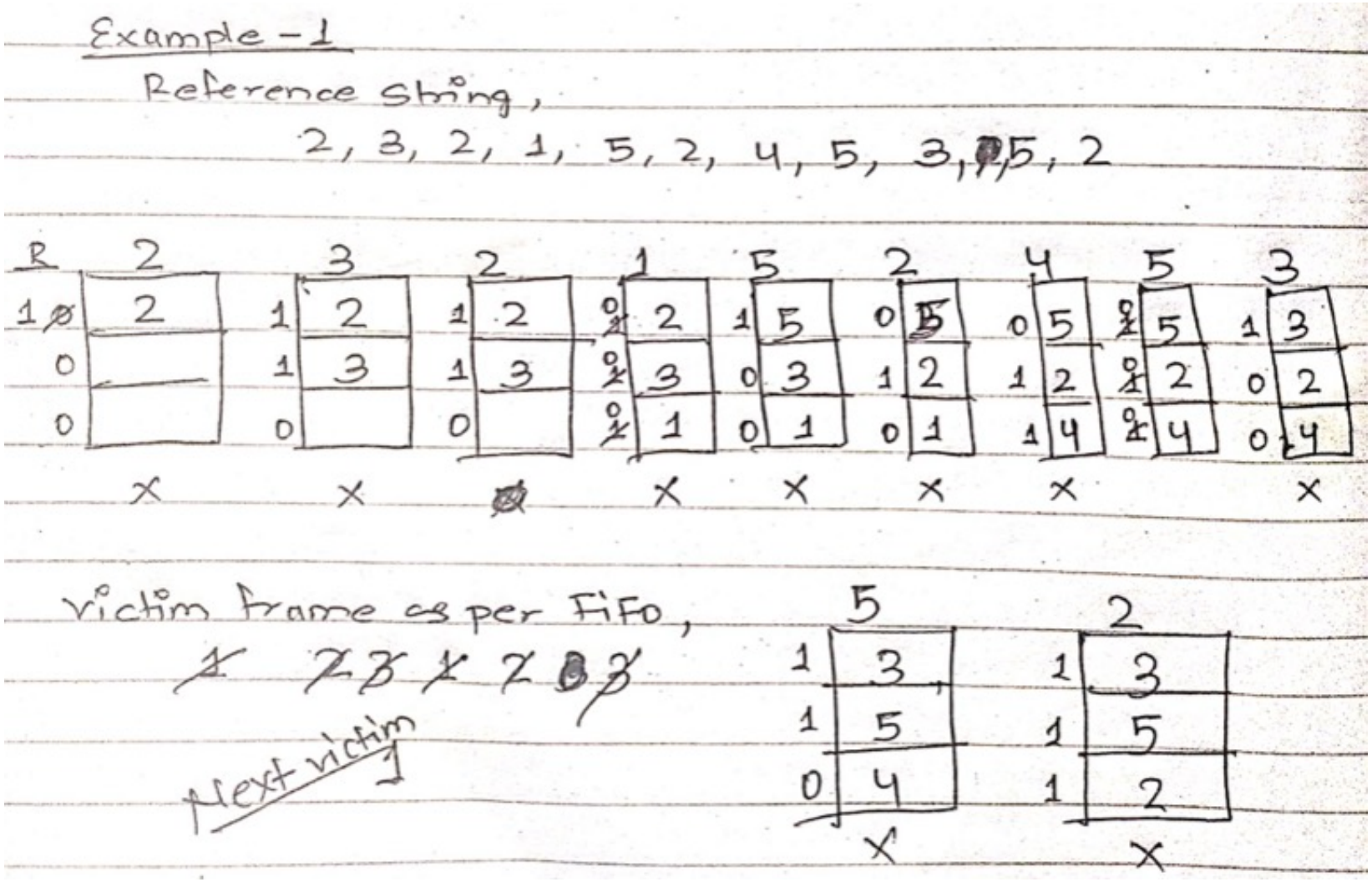
Page Fault ratio = 6/12

Second Chance/Clock Page Replacement (SCP)

Second chance / clock page Replacement:

- Also called clock page Replacement.
  - Set, Reference bit = 0, initially.
  * if page referenced set R bit to 1.
  * if page fault occurs,
    - if R = 1, set to 0, move to next frame.
    - if R = 0, Replace the page & set victim to next frame.
  * if page fault doesn't occur,
    - if R = 1, leave as it is.
    - if R = 0, set R = 1.

Example of SCP



Belady's Anomaly

- **In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames.**
- **However, Balady's found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.**
- This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady's Anomaly.

Let's examine such example :

The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

Case 1: Number of frames = 3

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 3			5	5	5	1	1	1	1	1	3	3
Frame 2		1	1	1	0	0	0	0	0	5	5	5
Frame 1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Belady’s Anomaly

Case 2: Number of frames = 4

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 4				3	3	3	3	3	3	5	5	5
Frame 3			5	5	5	5	5	5	1	1	1	1
Frame 2		1	1	1	1	1	1	0	0	0	0	4
Frame 1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady's Anomaly.

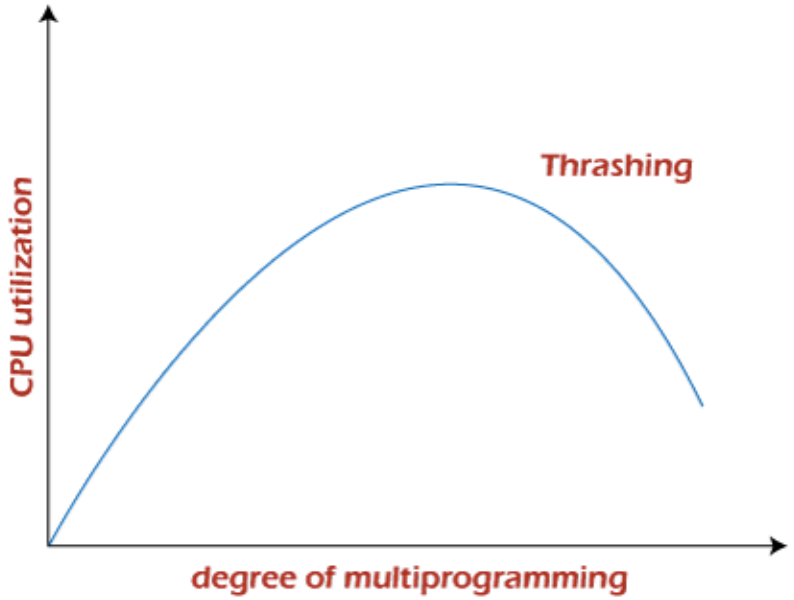
Thrashing

- ***Thrash** is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory.*

- ***Thrashing** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing.*

- **It causes the performance of the computer to degrade or collapse.**

- ***Thrashing* is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages.**
- This state in the operating system is known as thrashing.
- Because of thrashing, the CPU utilization is going to be reduced or negligible.



Segmentation

- **Segmentation is a memory management technique 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:**
  - **Base**: It is the base address of the segment
  - **Limit**: It is the length of the segment.

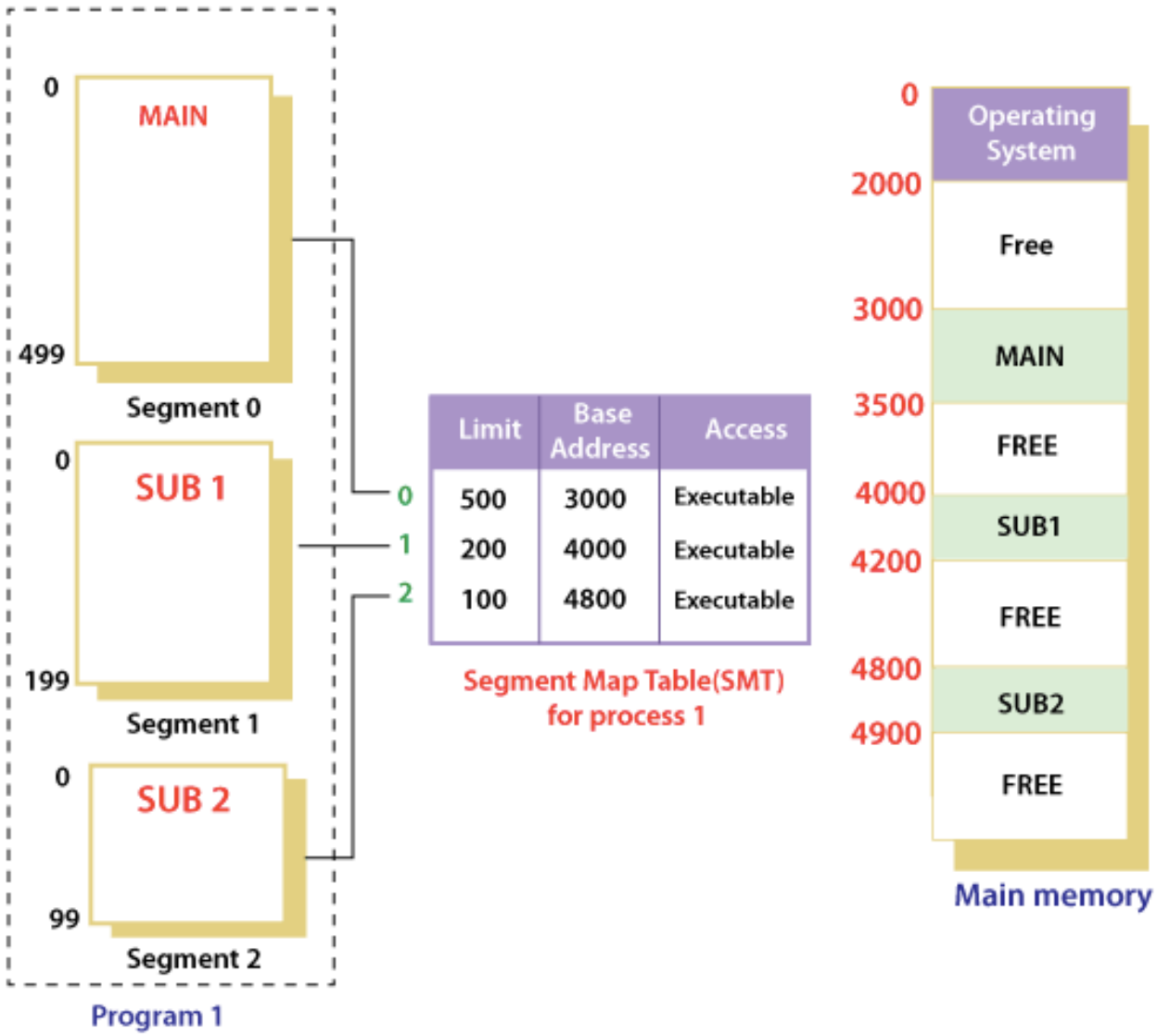
Advantages of Segmentation

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.

Segmentation Hardware



- **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.**

- 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.

- 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.

Paging vs Segmentation

Parameters	Paging	Segmentation
Individual Memory	In Paging, we break a process address space into blocks known as pages.	In the case of Segmentation, we break a process address space into blocks known as sections.
Memory Size	The pages are blocks of fixed size.	The sections are blocks of varying sizes.
Accountability	The OS divides the available memory into individual pages.	The compiler mainly calculates the size of individual segments, their actual address as well as virtual address.
Speed	This technique is comparatively much faster in accessing memory.	This technique is comparatively much slower in accessing memory than Paging.
Size	The available memory determines the individual page sizes.	The user determines the individual segment sizes.
Fragmentation	The Paging technique may underutilize some of the pages- thus leading to internal fragmentation.	The Segmentation technique may not use some of the memory blocks at all. Thus, it may lead to external fragmentation.
Logical Address	A logical address divides into page offset and page number in the case of Paging.	A logical address divides into section offset and section number in the case of Segmentation.
Data Storage	In the case of Paging, the page table leads to the storage of the page data.	In the case of Segmentation, the segmentation table leads to the storage of the segmentation data.

Segmentation with Paging (Multics)

- **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.**

- Pages are smaller than segments.

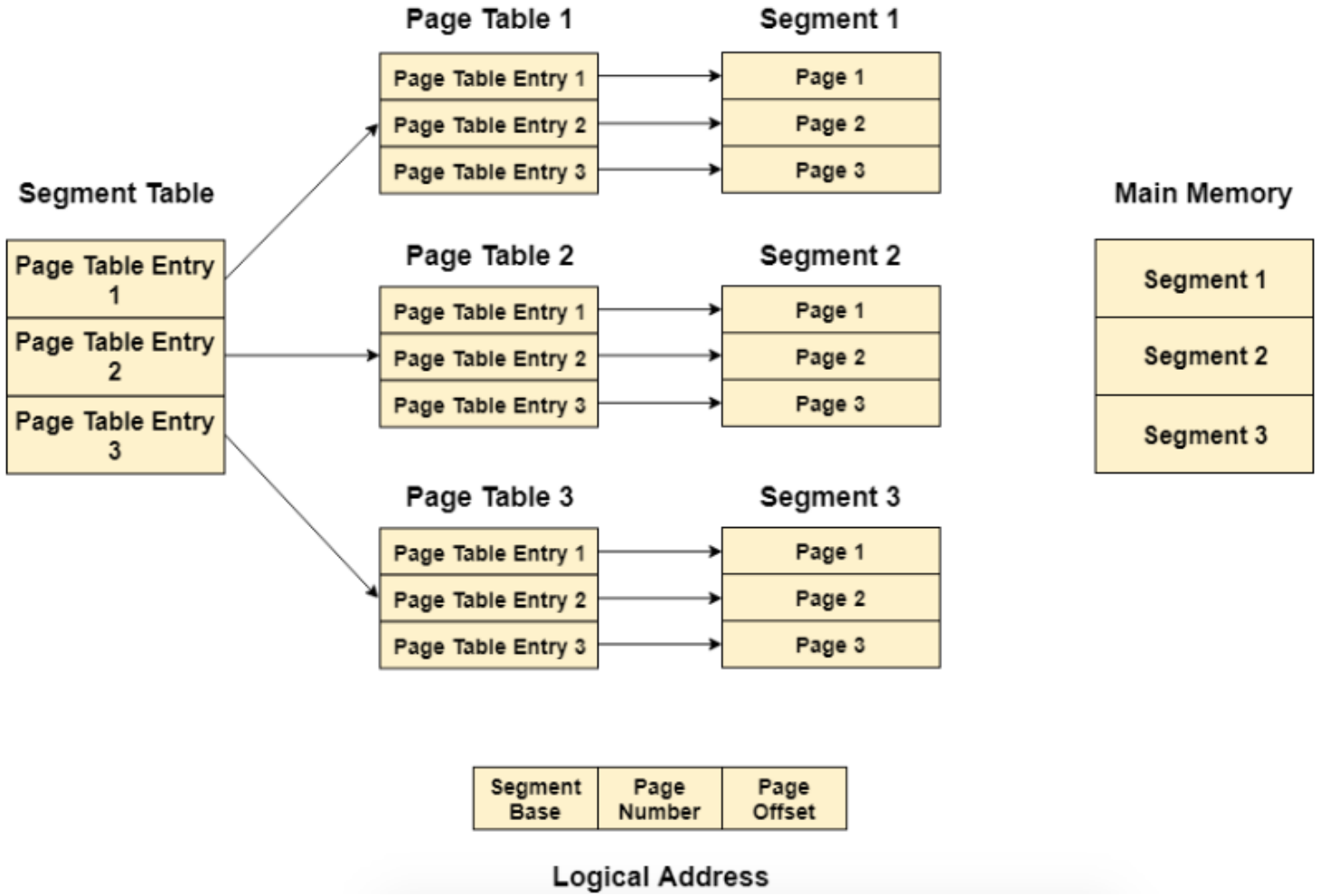
- **Each Segment has a page table which means every program has multiple page tables.**

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

Segmentation with Paging (Multics)

Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



Advantages:

- It reduces memory usage.
- Page table size is limited by the segment size.
- Segment table has only one entry corresponding to one actual segment.
- External Fragmentation is not there.
- It simplifies memory allocation.

Disadvantages:

- Internal Fragmentation will be there.
- The complexity level will be much higher as compare to paging.
- Page Tables need to be contiguously stored in the memory.