



Unit 4 – Memory Management

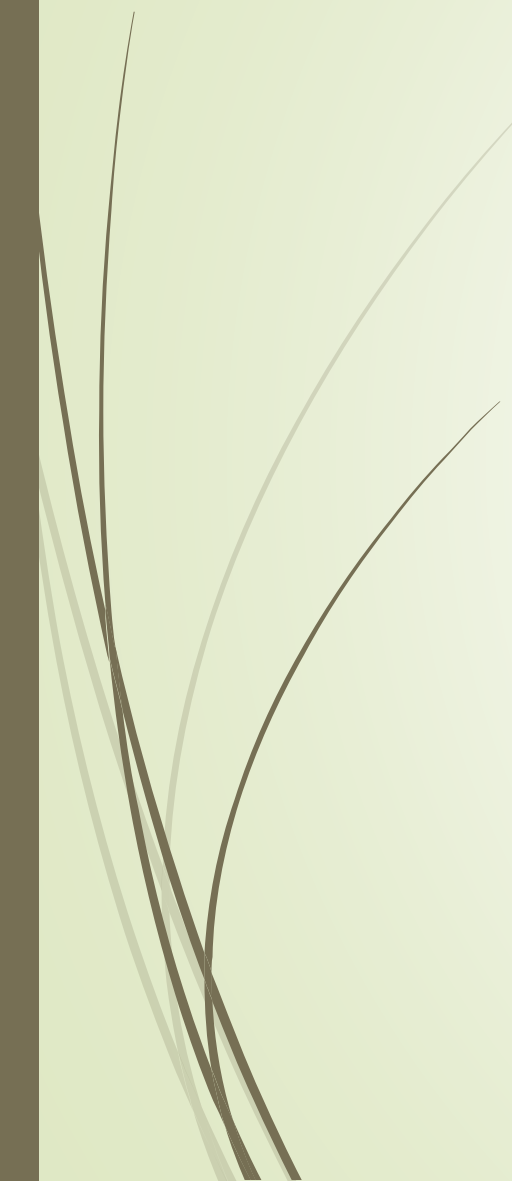


Introduction

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- A functionality of OS which manages memory



Introduction (contd.)

- The entire program and data of a process must be in main memory for the process to execute
 - It keeps track of the each and every memory allocation either it is allocated to some process or not
 - It decides which process gets the memory at what time
 - The part of OS which manages the memory is called memory manager
- 



Memory Management

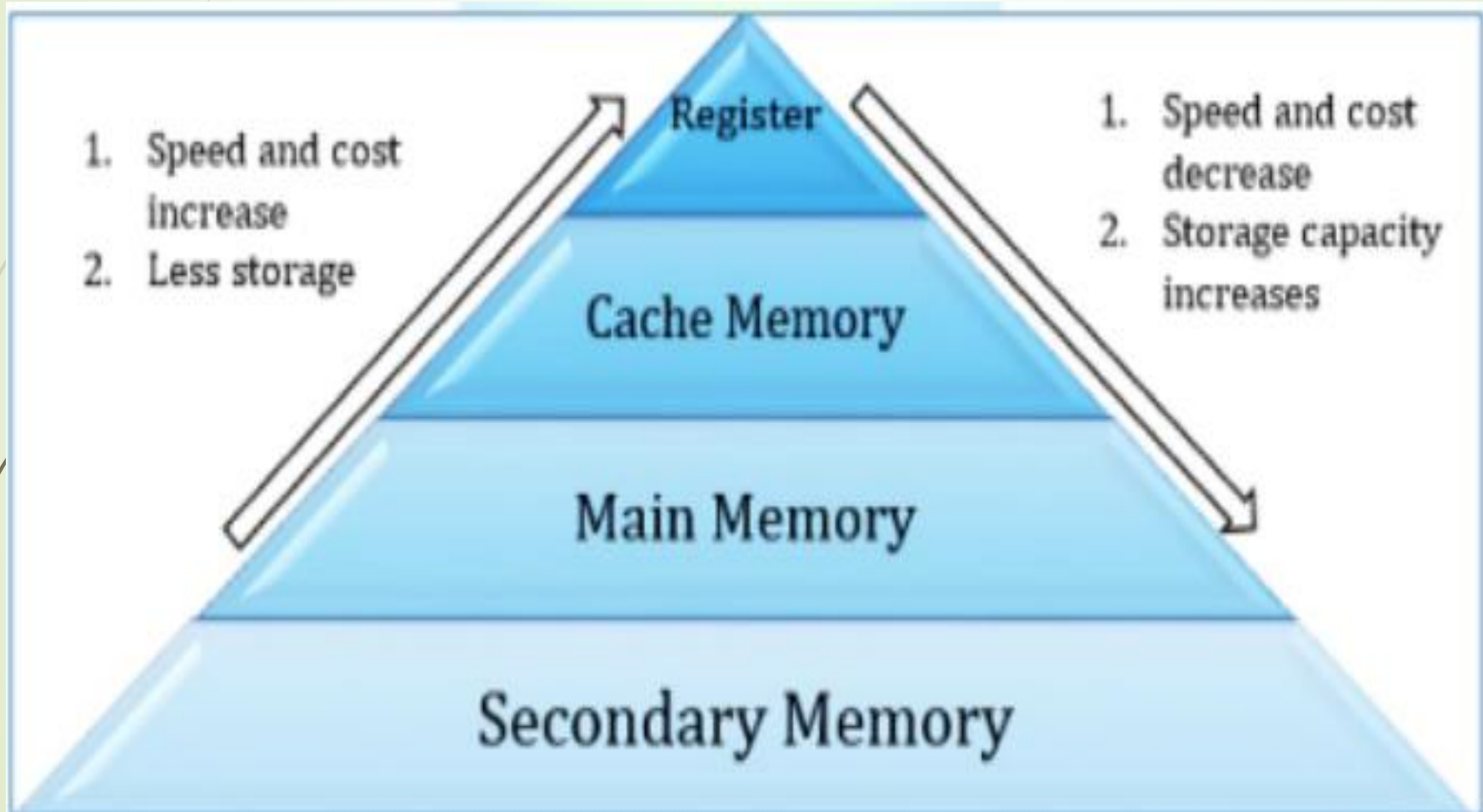
- RAM is an important resource that must be carefully managed.
- Nowadays memory size is heavily increase and programs are getting faster and bigger than memory.
- Memory management deal with how OS manage them.
- Once a program is loaded into memory , it remains there until it finishes.



Memory Manager

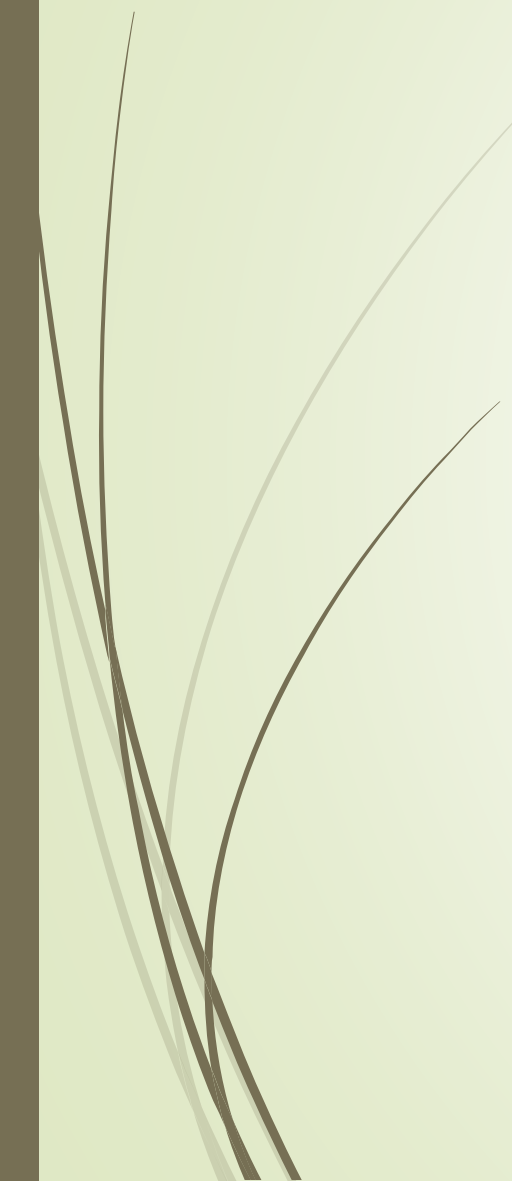
- Its job is to:
 - keep track of which parts of memory are in use
 - allocate memory to processes
 - deallocate memory when processes are done
 - manage swapping between main memory and disk

Memory Hierarchy



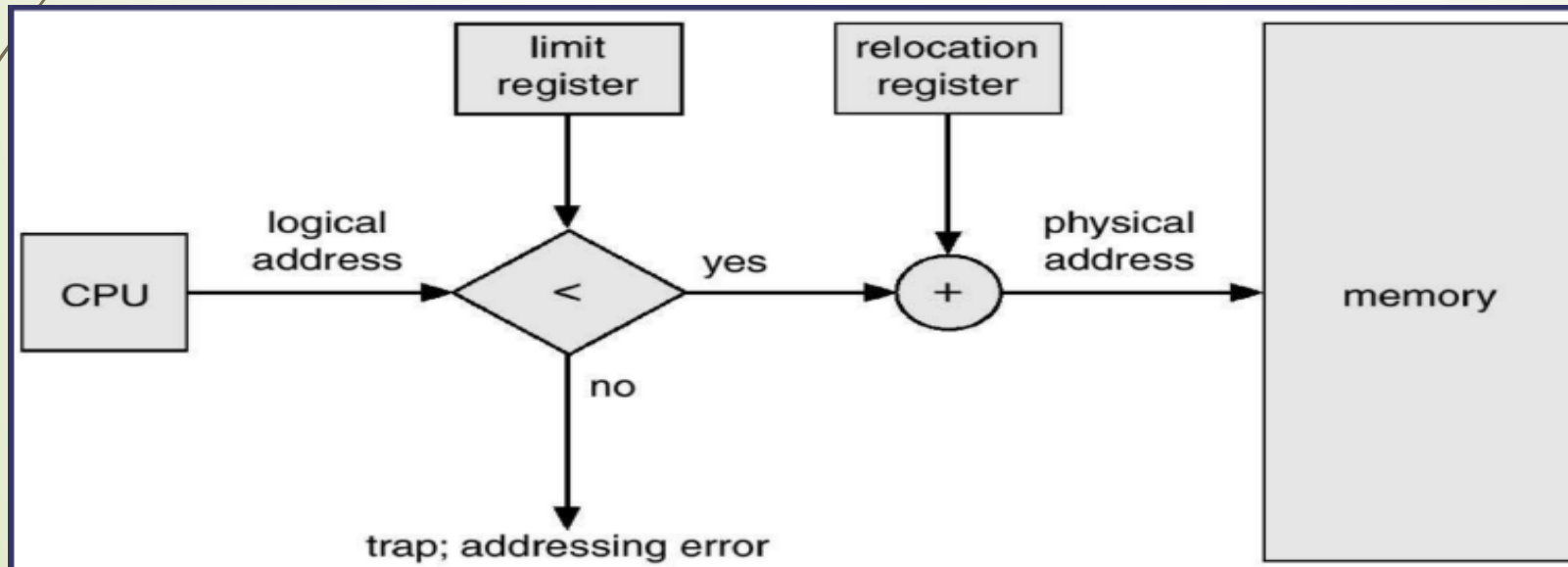


Logical VS Physical Address

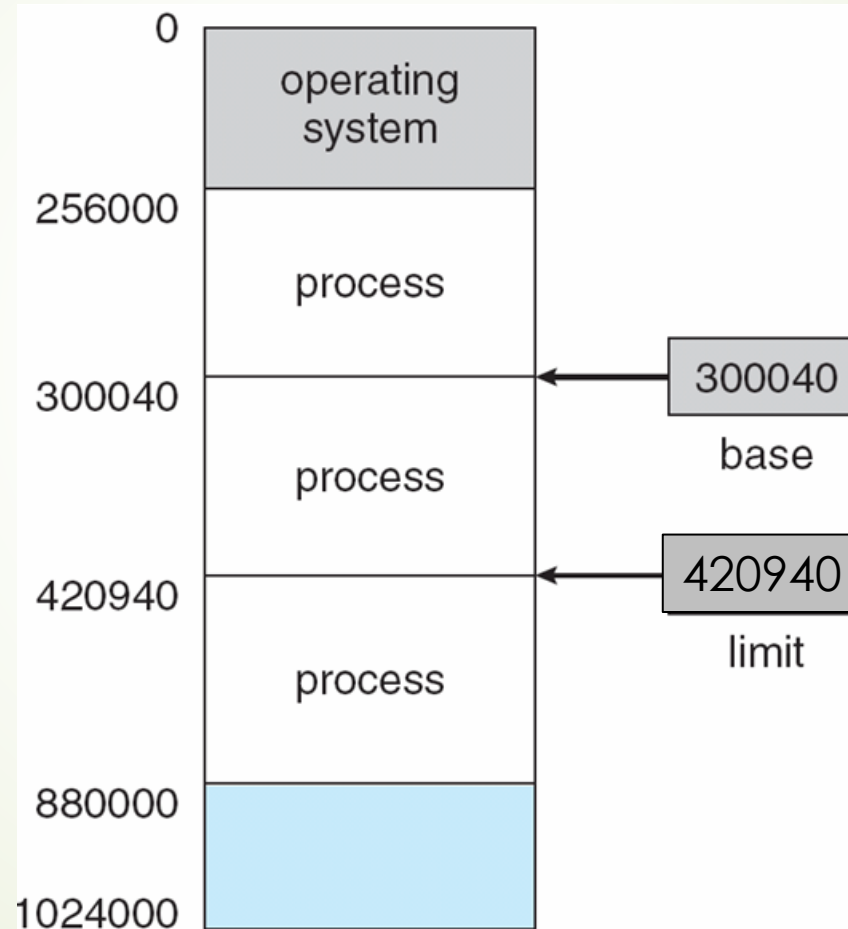
- ▶ The address generated by the CPU is a logical address.
 - ▶ Address actually seen by the memory hardware is a physical address.
 - ▶ Logical address is also known as virtual addresses.
 - ▶ The set of all logical addresses used by a program composes the logical address space and the set of all corresponding physical addresses composes the physical address space.
 - ▶ The run time mapping of logical to physical addresses is handled by the memory management unit, MMU.
- 

Addresses

- **Logical Address:** These addresses are generated by CPU and also known as virtual address
- **Physical Address:** These address are actual addresses loaded into memory address register
- Mapping uses two register: **Base** and **Limit** registers
- **Base:** Specifies the smallest legal physical memory address.
- **Limit:** Specifies the size of the range.

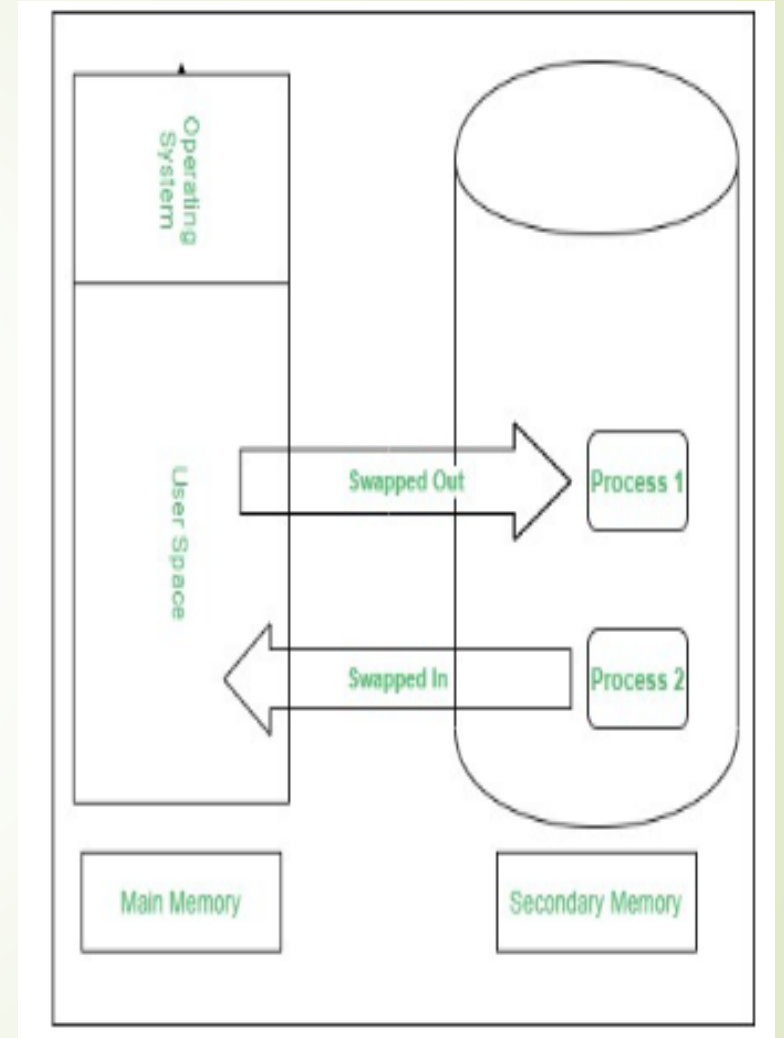


Base and Limit Registers



SWAPPING

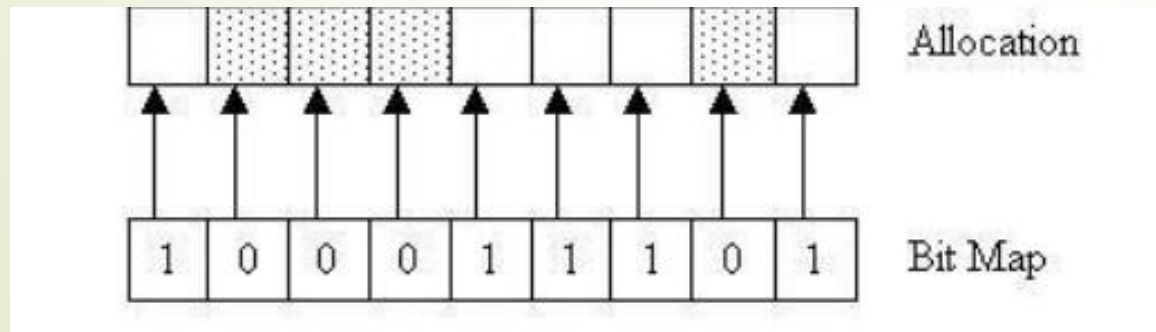
- Mechanism in which a process can be swapped temporarily out of main memory to secondary storage and make that memory available to other processes.
- When higher priority process enters the input queue, a low –priority process is swapped out , so the higher-priority process can be loaded and executed .
- **Swap-out** is a technique for moving a process from RAM to the hard disc.
- **Swap-in** is a method of transferring a program from a hard disc to main memory, or RAM.



Memory management with Swapping

Bitmap or bit-vector

- Series or collection of bits where each bit corresponds to a disk block
- The bit takes two values 0 and 1
 - where, 1 is allocated block and 0 is free block
- If the allocation unit will be chosen large, we could waste memory as we may not use all the space allocated in each allocation unit.
- Slow operation and for this reason bit maps are not often used.



Memory management with Swapping Linked List

- The free disk blocks are linked together i.e. a free block contains a pointer to the next free block
- The block number of the very first disk block is stored at a separate location on disk and is also cached in memory
- 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.
- Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward.
- A terminating process normally has two neighbors (except when it is at the very top or very bottom of memory).

Memory management with Swapping Linked List

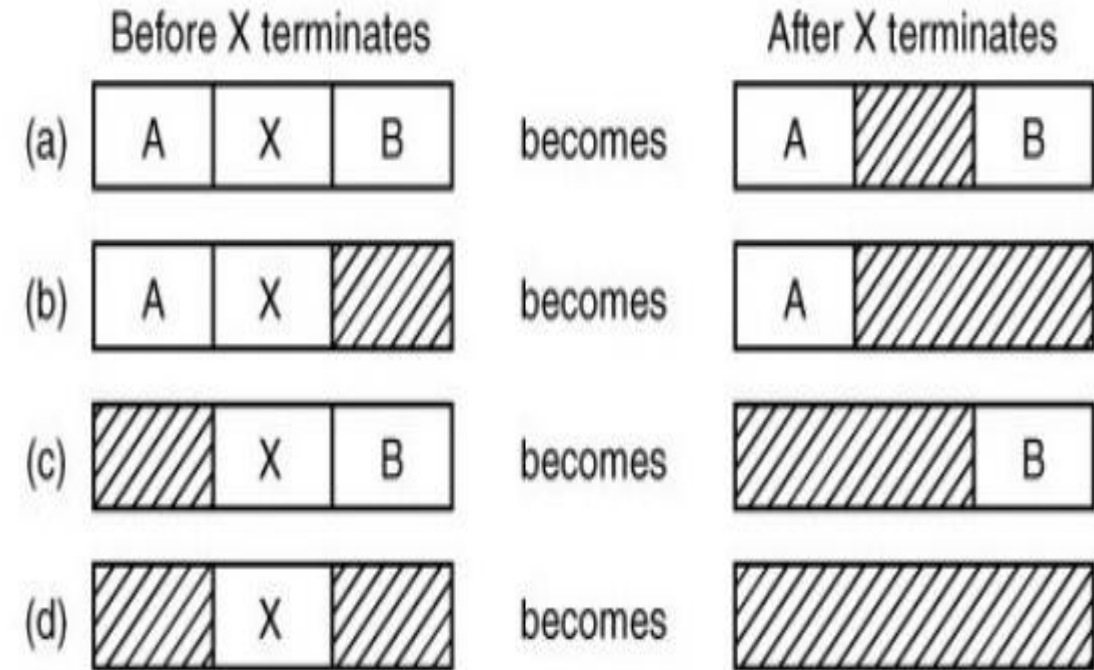
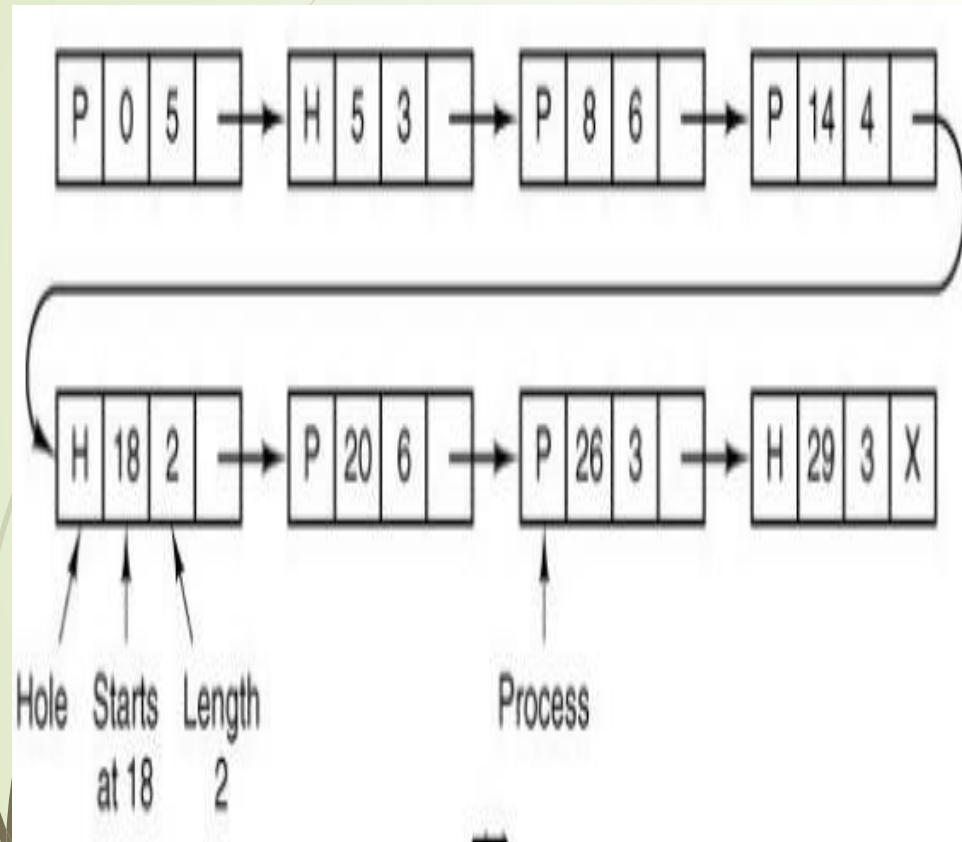
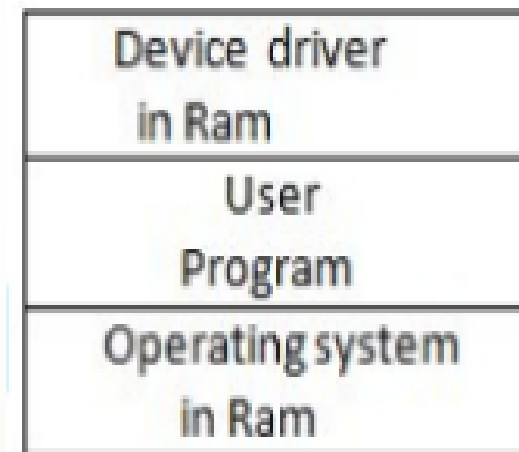
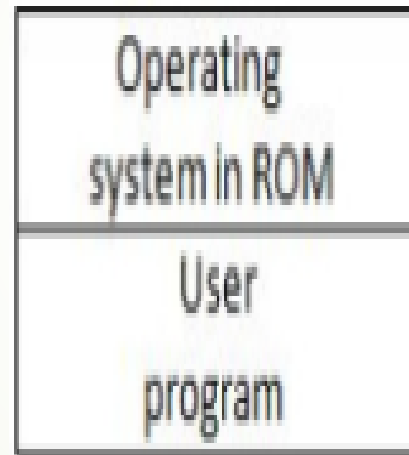
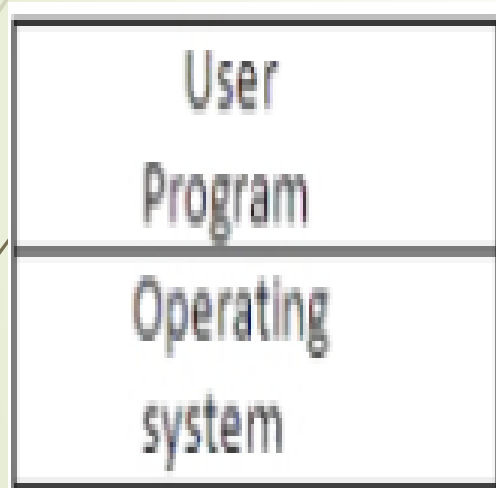


Fig: Four neighbor combinations for the terminating process, X.

Memory Management Without Swapping

Mono Programming

- Simplest memory management in which only one program is run at a time , sharing the memory between the program and operating system.



Memory Management Without Swapping

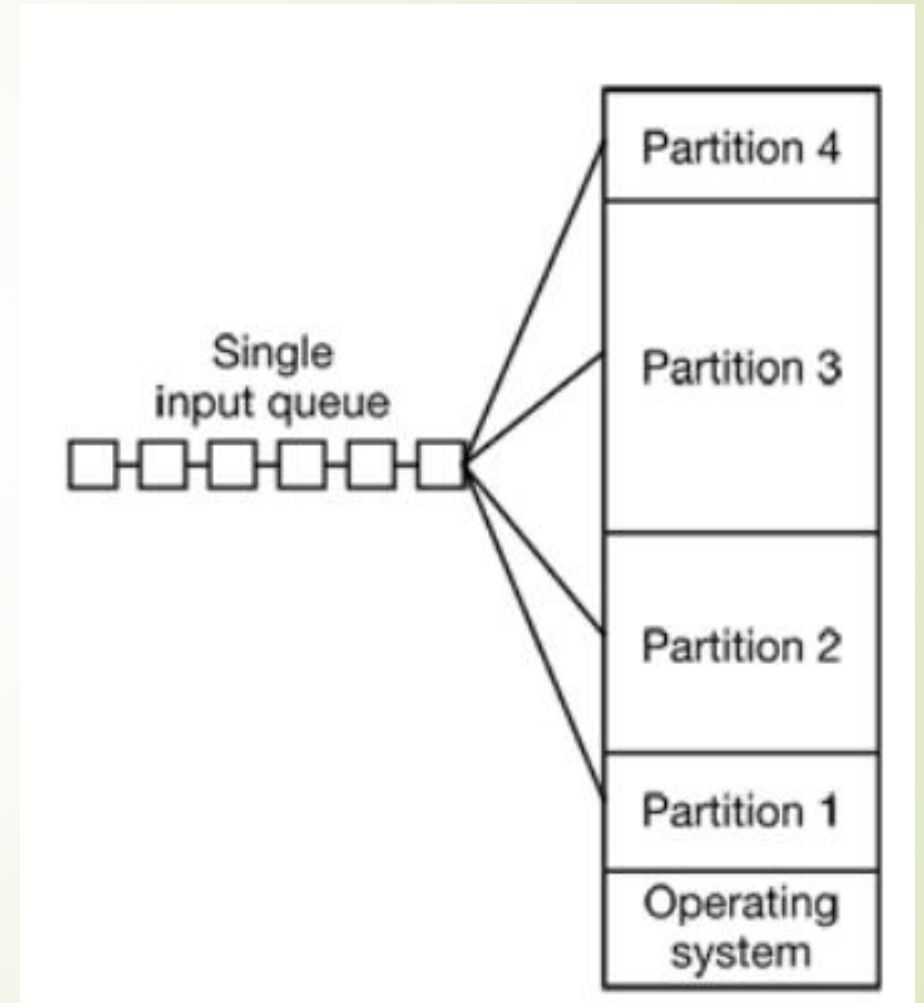
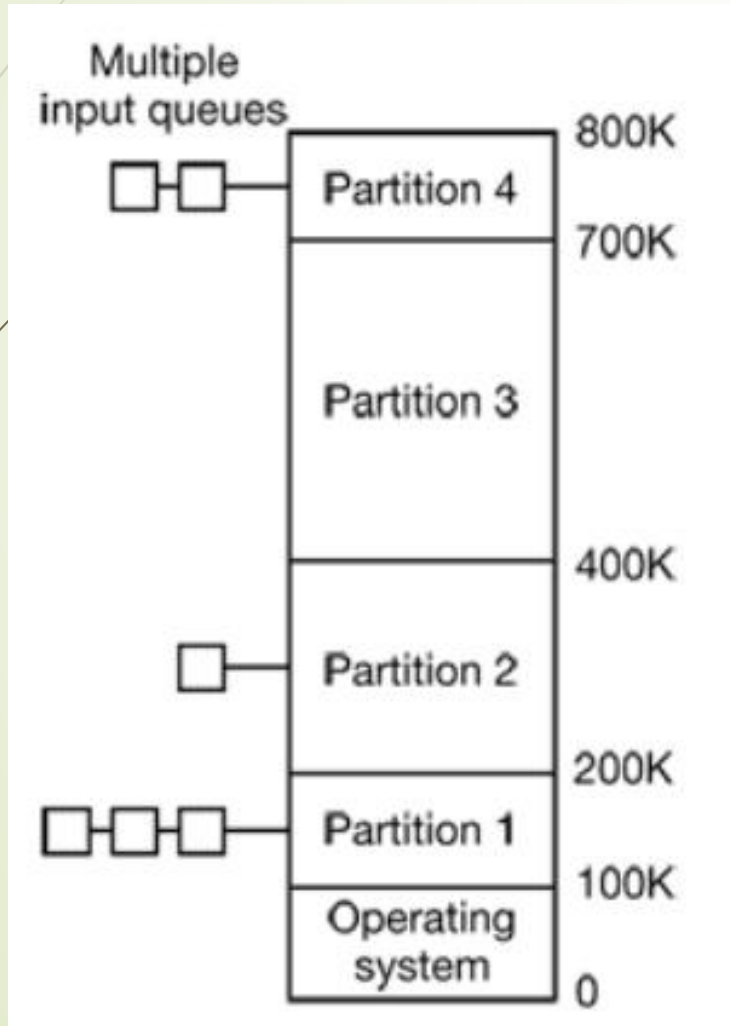
Mono vs Multi Programming

Mono-Programming	Multi-Programming
Only one program at any point of time	Contains more than one program at a time
CPU executes the program but when I/O operation is encountered, CPU sits ideal. So, CPU isn't efficiently used	CPU executes the program but when one program encounters I/O operation, CPU switches to next program. So, CPU is efficiently used
Memory needs less space	Memory needs more space
Fixed size partition is used	Both fixed and variable sized are used
Example: Old mobile OS	Example: Windows 7/8/10/11

Multiprogramming with fixed partition

- Multiple processes run at the same time
- Memory partitioned into fixed no. of partitions
- In this , number of partitions in RAM is fixed but size of each partition may or may not be same.
- Spanning is not allowed.
- Advantage :
 - Easy to implement
 - Little OS Overhead
- Disadvantage:
 - Internal Fragmentation
 - External Fragmentation
 - Limit in process size
 - Limitation on degree of multiprogramming

Multiprogramming with fixed partition (contd.)



Multiprogramming with dynamic partition

- Multiple processes run at the same time
- Memory partitioned dynamically
- Meets the requirement of each requesting process – **variable partitioning**
- When a process terminates or swapped, memory manager free up the space

Free Space for RAM
P1 = 30MB
P2 = 3MB
Operating System

Contiguous Memory Allocation

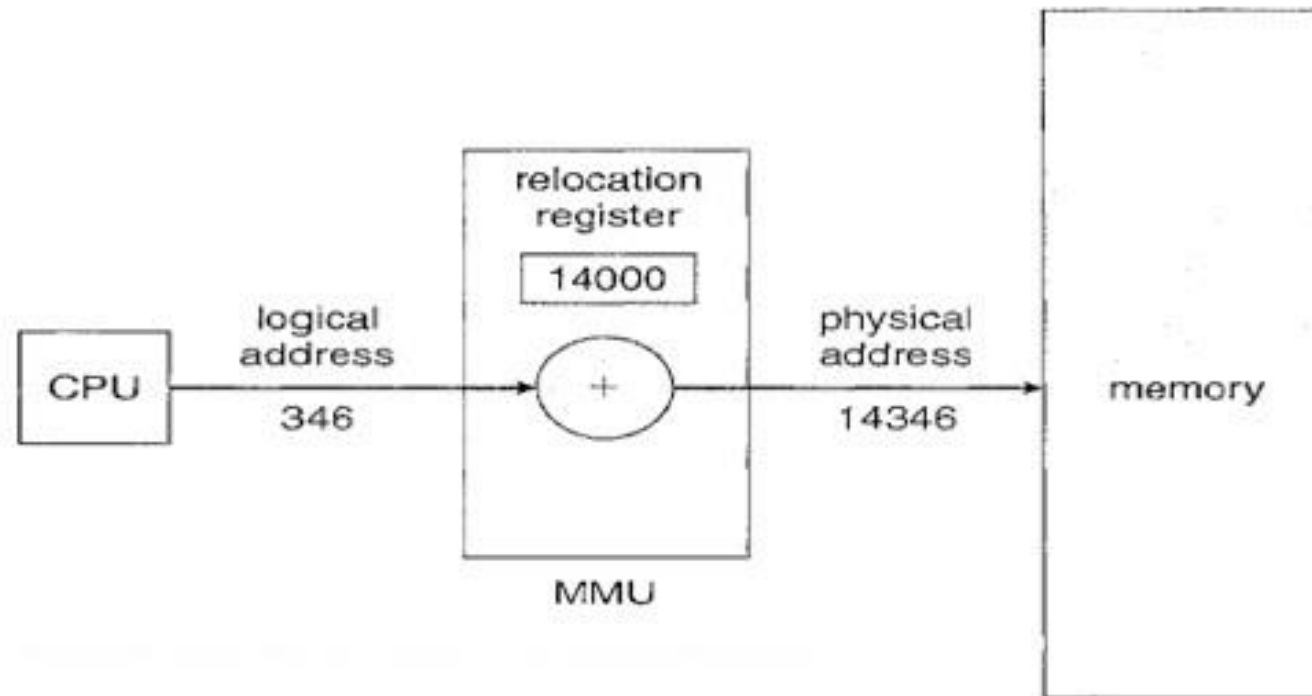
- Refers to the allocation of a continuous block of memory to a process or program.
- The entire memory requirement of a process is satisfied by a single contiguous block of memory.
- The operating system keeps track of the memory blocks that are allocated and those that are free.
- When a process requests memory, the operating system searches for a large enough contiguous block of free memory to satisfy the request.
- Advantages:
 - It is simple and easy to implement.
 - It minimizes external fragmentation because memory is allocated in contiguous blocks.
- Disadvantages:
 - can lead to internal fragmentation

Non-Contiguous Memory Allocation

- Non-contiguous memory allocation involves allocating memory to a process or program in non-continuous or scattered blocks.
- memory space of a process can be divided into multiple smaller fragments or pages, and these fragments can be allocated in different locations in the physical memory.
- Advantages:
 - It allows efficient utilization of memory by allocating memory in smaller, non-contiguous blocks..
 - It helps overcome external fragmentation because memory is allocated in smaller units
- Disadvantages:
 - It helps overcome external fragmentation because memory is allocated in smaller units
 - It can lead to slower access times compared to contiguous memory allocation due to the need for page swapping between secondary storage and physical memory.

Relocation

- The ability to load and execute a given program into memory
- the program may be loaded at different memory locations, which are called physical addresses
- Relocation is way to map logical addresses into physical addresses



Types of Relocation – Static and Dynamic

➤ Static

- Program must be relocated before or during loading of process into memory
- Program must be loaded into same memory address space in memory

➤ Dynamic

- Process can be freely moved around the memory
- Logical-to-Physical address mapping can be done at run time



Protection

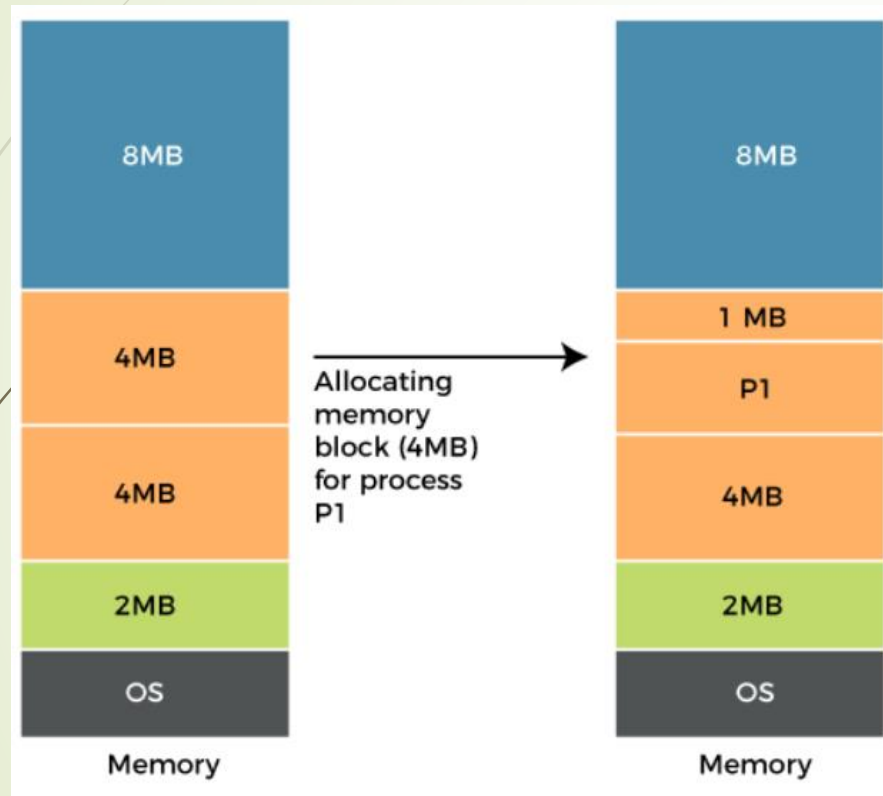
- A way to control memory access rights on a computer – **memory protection**
- Main purpose is to prevent the process or program from accessing the memory that has not been allocated to it
- Prevents malware or bugs present in the process to affect other process in the memory
- Prevent data and instructions to be overwritten – **write protection**
- Ensure the privacy of data and instruction – **read protection**



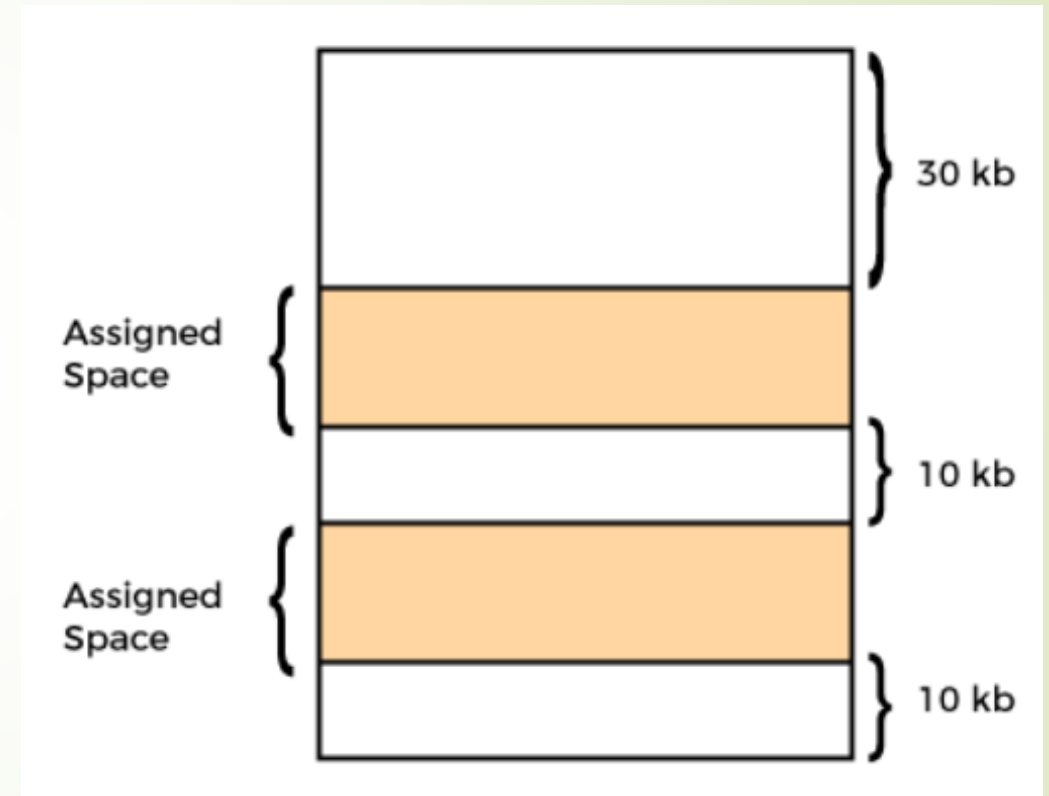
Fragmentation

- Divided into multiple small parts
- the process is loaded and unloaded from memory, there exists free space which are fragmented that cannot be allocated to incoming processes
- It is an unwanted problem in OS
- Two types of fragmentation:
 - **Internal Fragmentation** - if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes internal fragmentation
 - **External Fragmentation** - when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable

Types of Fragmentation



Internal



External

Memory Allocation Strategies

➤ First Fit

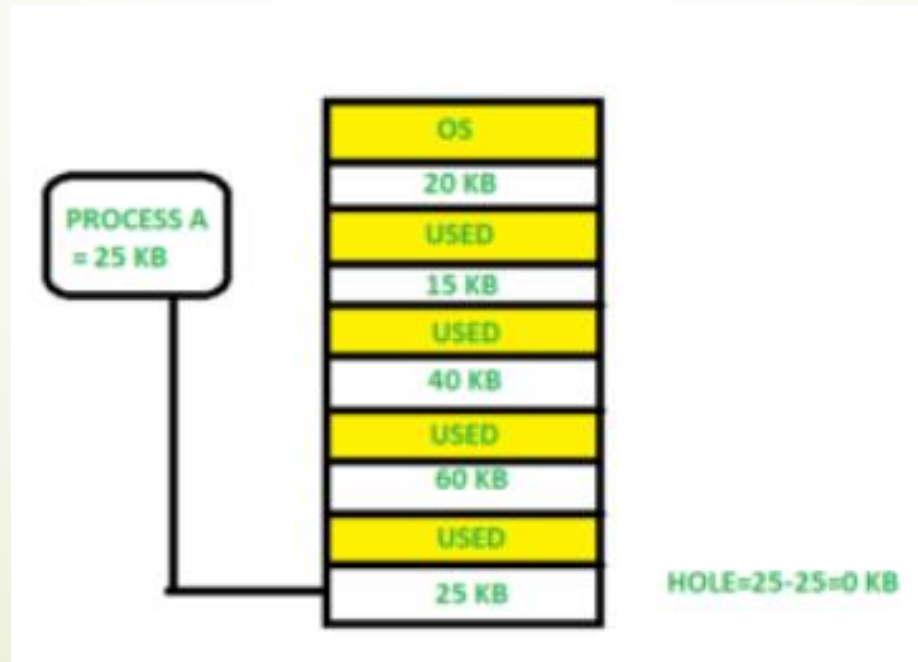
- The partition is allocated which is the first sufficient block from the top of memory
- It scans memory from the beginning and chooses the first available block that is large enough
- Thus it allocates the first hole that is large enough



MAS (contd.)

➤ Best Fit

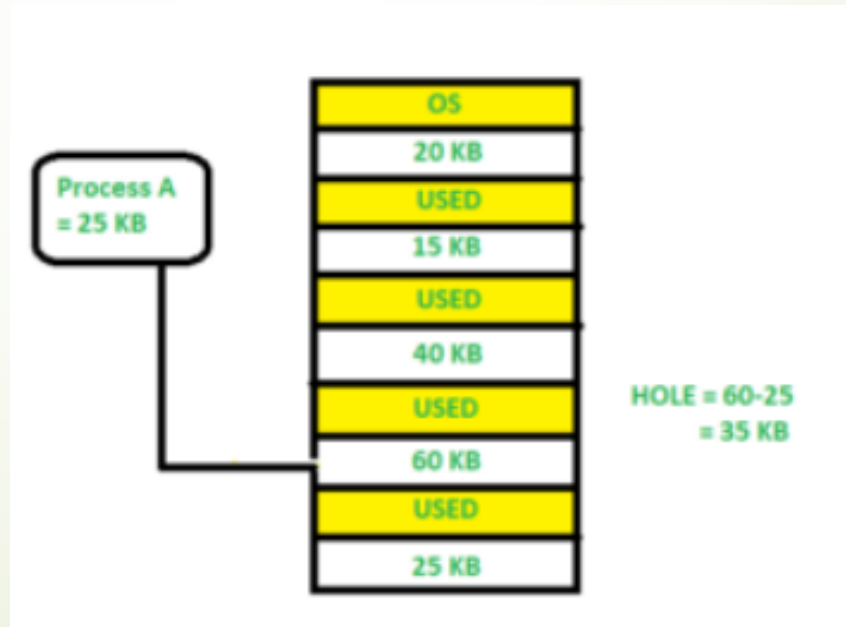
- Allocate the process to the partition which is the first smallest sufficient partition among the free available partition
- It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process



MAS (contd.)

➤ Worst Fit

- Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the memory
- It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process



MAS - Numerical

- Given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, 600 KB in order. How would it fit under best, worst and first algorithms when processes of 212 KB, 417 KB, 112 KB and 426 KB in order needs to be executed. Which algorithm makes the most efficient use of the memory?

- First

- 212- 500
- 417- 600
- 112- 288 (500-212)
- 426 must wait

- Best

- 212 - 300
- 417 - 500
- 112 - 200
- 426 - 600

- Worst

- 212 - 600
- 417 - 500
- 112 - 388
- 426 must wait



MAS - Numerical

- Given memory partitions of 100KB, 200KB, 300KB, 400KB, 500KB in order. How would it fit under best, worst and first algorithms when processes of 202KB, 404KB and 90KB in order needs to be executed. Which algorithm makes the most efficient use of the memory?



Compaction

- Technique to collect and combine the free spaces and relocate it so that the space is big enough to be allocated by some programs
- The merging of those free partitions and allocating according to the needs of process - **defragmentation**

Coalescing

- Merging two adjacent free blocks of memory.
- When a program no longer requires certain blocks of memory, these blocks of memory can be freed.
- Used to reduce external fragmentation, but is not totally effective.

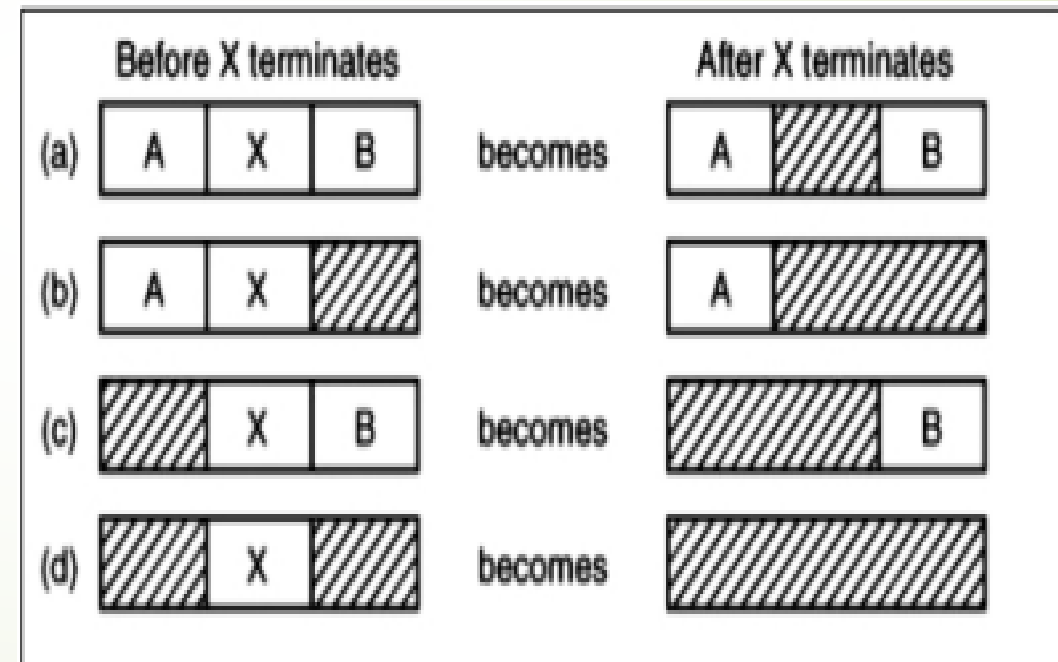
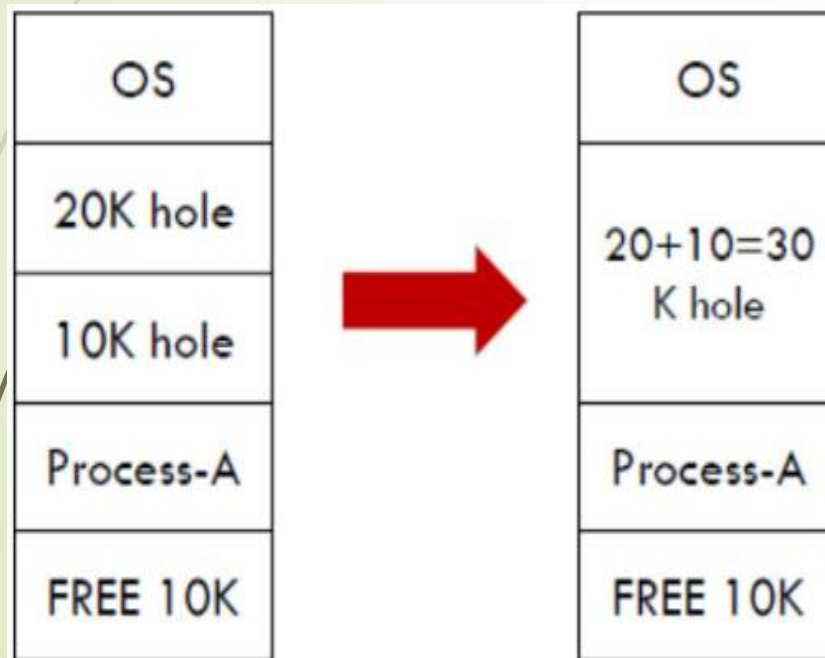


Fig: Coalescing Process

Virtual Memory

- A computer can add more memory than the amount of physical installed on system this extra memory is called virtual memory.
- Combined size of the program, data, and stack may exceed the amount of physical memory available for it
- The operating system keeps those parts of the program currently in use in main memory, and the rest on the disk
 - Only part of the program needs to be in memory for execution
 - Logical address space can therefore be much larger than physical address space
 - Allows address spaces to be shared by several processes
 - Allows for more efficient process creation
 - More programs running concurrently
 - Less I/O needed to load or swap processes

Virtual Memory (contd.)

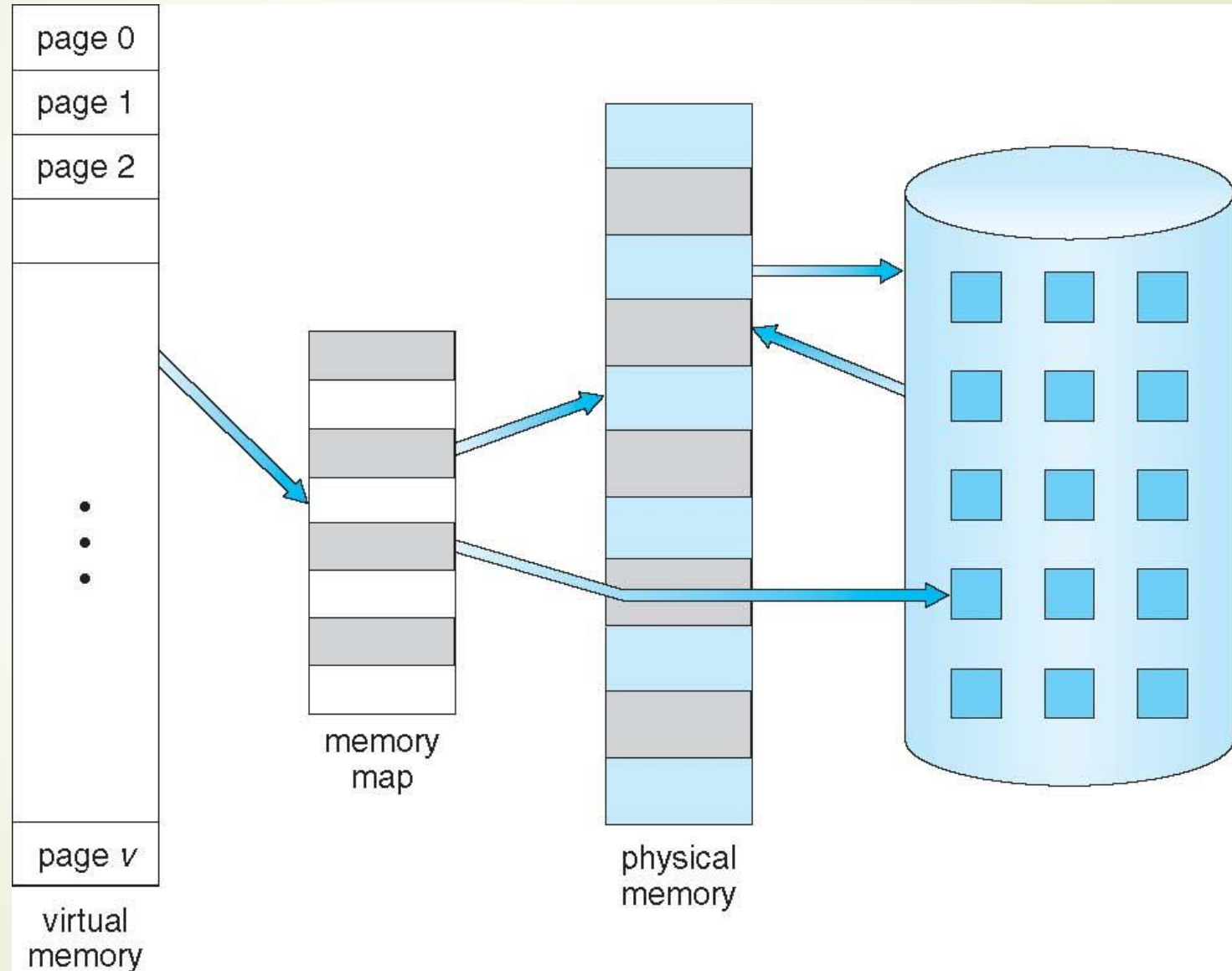
- For example, a 16-MB program can run on a 4-MB machine by carefully choosing which 4 MB to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed
- **Benefits:**
 1. It allows to execute a process whose size is larger than physical memory.
 2. It is used to increase the range of multiprogramming by loading more no of process into physical memory.
- Virtual memory can be implemented by two most commonly used methods :
 - Paging
 - Segmentation
 - or mix of both



Virtual Address Space

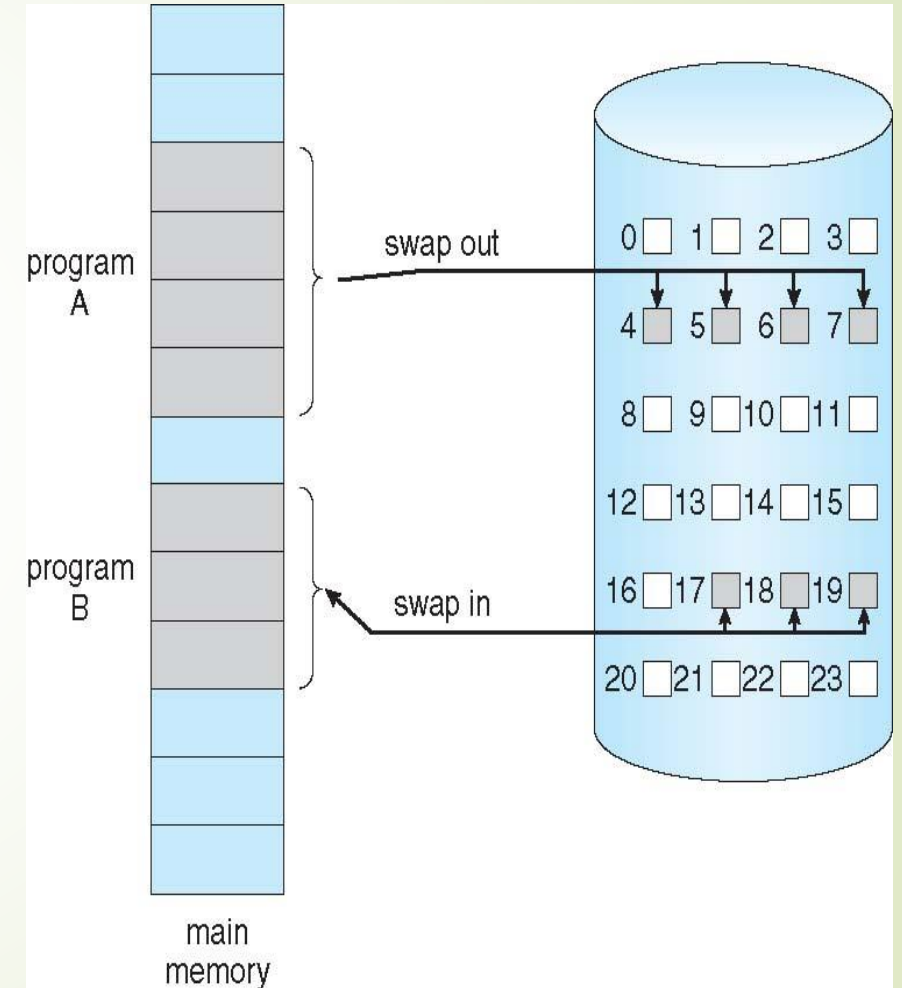
- Logical view of how process is stored in memory
 - Usually start at address 0, contiguous addresses until end of space
 - Meanwhile, physical memory organized in page frames
 - MMU must map logical to physical

Virtual Memory That is Larger Than Physical Memory



Paging

- Memory management technique in which process address space is broken into blocks of same size.
- The virtual address space is divided up into units - **pages**
- The corresponding units in the physical memory - **page frames**
- The pages and page frames are always the same size
- Transfers between RAM and disk are always in units of a page



Address Translation Scheme

- 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 memory
 - Page Offset (d)
 - The part of a virtual address that specifies the location of a byte within a page

Page Number(p)

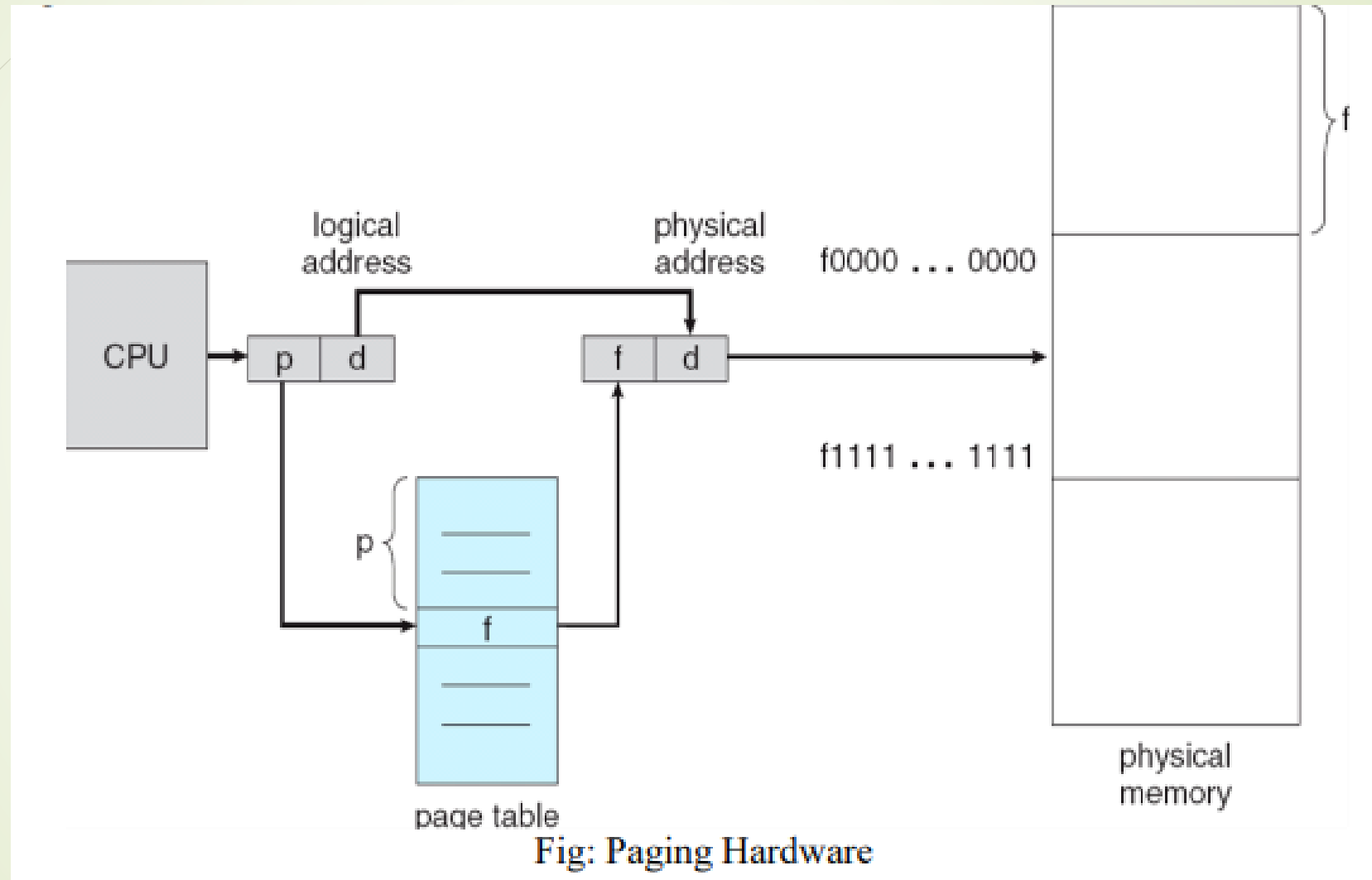
m-n bit

Page offset (d)

n bit

- Logical address is 2^m and page size 2^n

Paging (contd.)



Page Table

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H

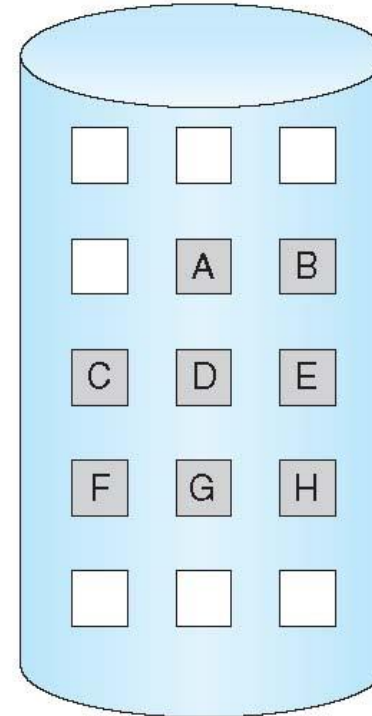
logical
memory

valid-invalid bit		
frame		
0	4	v
1		i
2	6	v
3		i
4		i
5	9	v
6		i
7		i

page table

0	
1	
2	
3	
4	A
5	
6	C
7	
8	
9	F
10	
11	
12	
13	
14	
15	

physical memory



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** \Rightarrow in-memory – **memory resident**, **i** \Rightarrow not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries

Frame #	valid-invalid bit
	v
	v
	v
	i
...	
	i
	i

page table



Types of page Table

Hierarchical

- It is also called Multilevel Paging and it is a very simple methodology.
- When the page table is too big to fit in a contiguous space then this hierarchical paging system is used with several levels.
- In this, the logical address space is broken up into Multiple page tables.
- Hierarchical paging uses the following two types of page tables:
 - **Two-Level Page Table:** On this page table, itself is paged. Therefore, here two-page tables' inner page table and outer page table are present.
 - **Three-Level Page Table:** In this, divide the outer page table first, and then it will result in a Three-level page table

Types of page Table Hierarchical

P1	P2	d
10	10	12
Outer Page Table	Inner Page Table	Offset

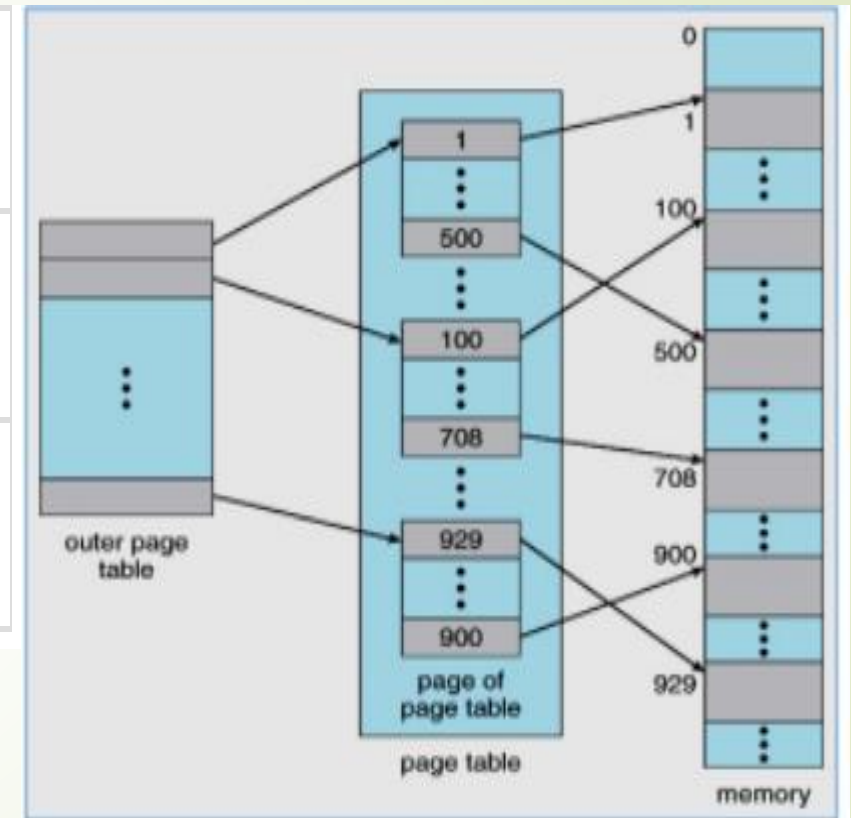
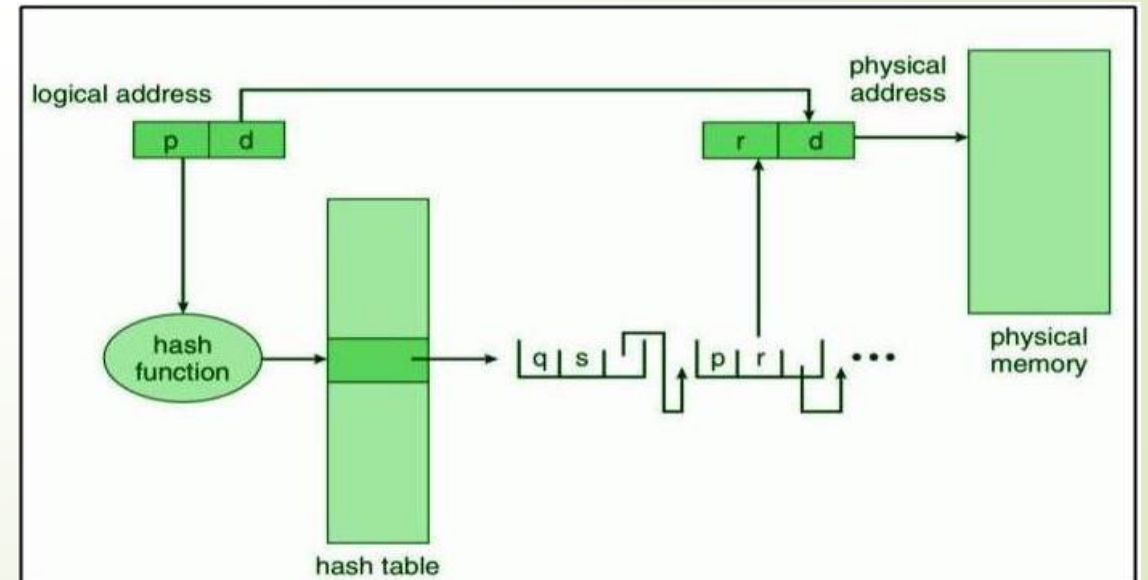


Fig: Two Level Page Table Scheme

Hash Page Table

- Address space greater than 32 bit
- Virtual page numbers are compared in this chain searching for a match . If a match is found , the corresponding physical frame is extracted.
- Each element consist three field ,Virtual page number, value of mapped page frame, pointer to next element in linked list.

- Searching is done faster
- Hashed value as virtual page no.
- Each entry contain linked list



Inverted Page Table

- Each process has an associated page table.
- Page tables may have large number of entries wasting the physical memory.
- To avoid this inverted page table is used.
- Only one page table in physical memory.
- For each frame it has a entry

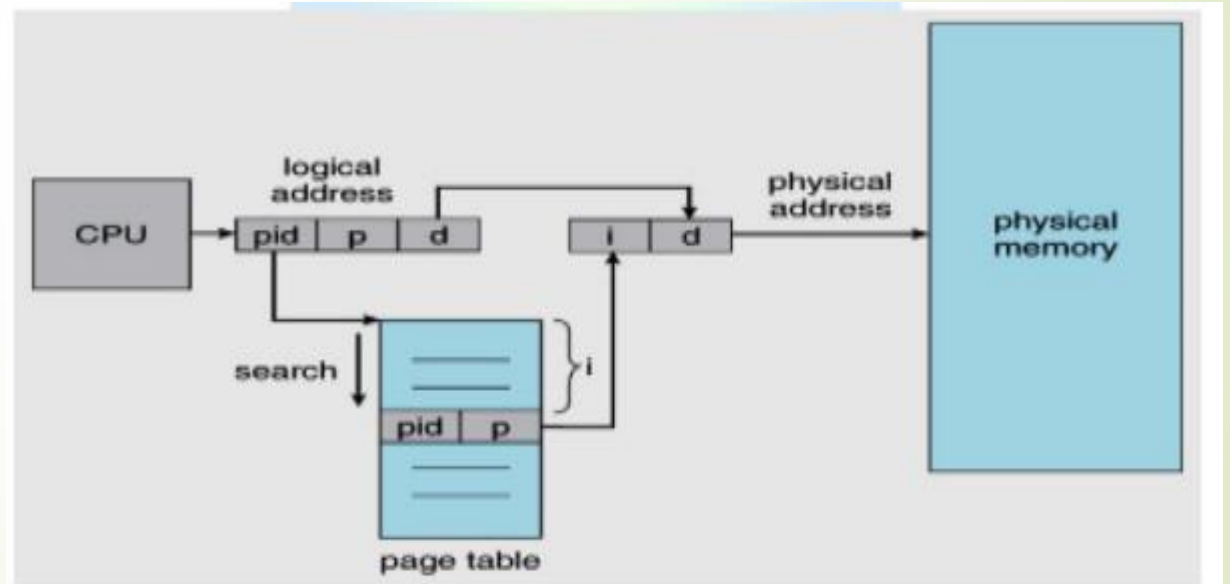


Fig: Inverted Page Table

Demand Paging

- Technique used in virtual memory management that enable computer to use its physical memory more efficient.
- Basic Idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once .
- They are swapped in only when the process needs them (on-demand).
- This is term as lazy swapper,

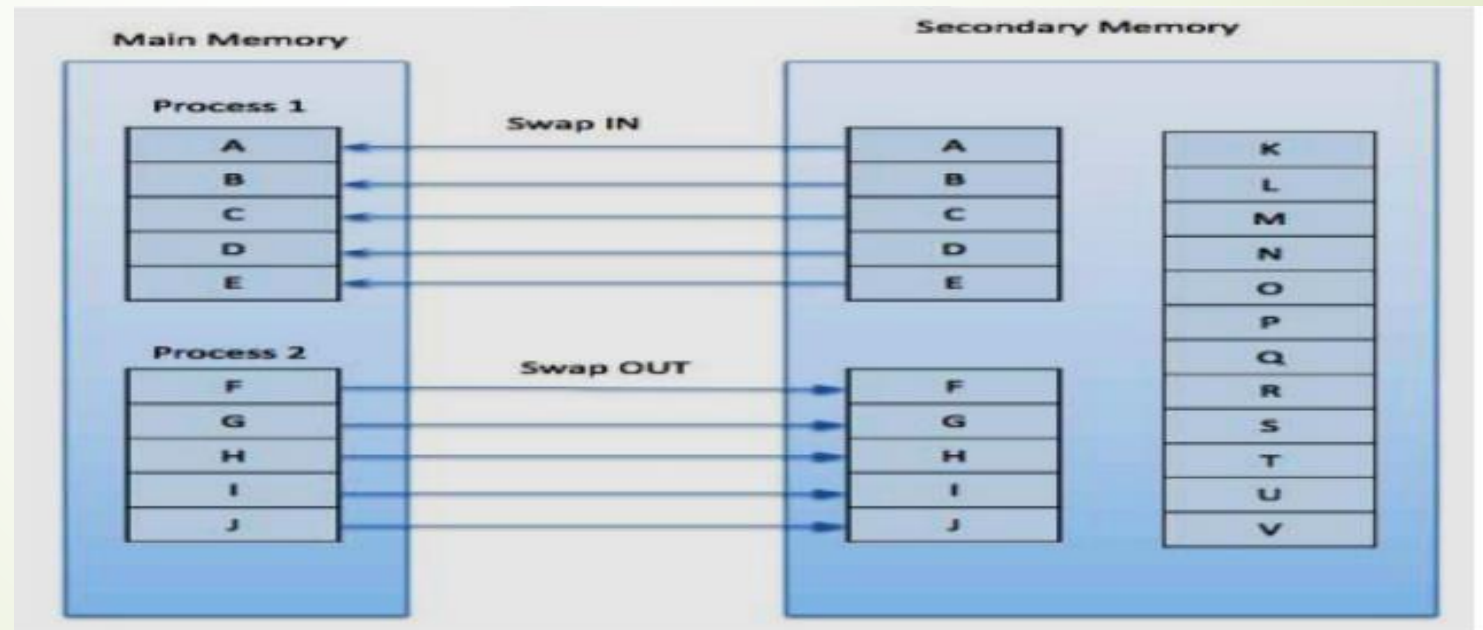



Fig: Demand Paging

Demand Paging

- ▶ While executing the program , if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference.
- ▶ And transfer control from the program to the operating system to demand the page back into the memory.
- ▶ Advantage:
 - ▶ Large Virtual memory
 - ▶ More efficient use of memory
 - ▶ There is no limit on degree of multiprogramming
- ▶ Disadvantage:
 - ▶ Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.



Page Fault

- A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM.
 - If the program is of five pages 1,2,3,4,5 and out of these pages 1,4, and 2 are loaded into main memory at the time of execution.
 - If the program tries to address the address of page number 3 then, there will be an error, as this page is not swapped into main memory at the time of loading the program.
- 



Page Replacement Algorithms

- Algorithms decides which page is replace by which page
- The main objective of all the Page replacement policies is to decrease the maximum number of page faults.
- When a page fault occurs, the OS has to choose a page to remove from memory to make the room for the page that has to be brought in
- Pages should exist in order to not have page fault

FIFO

- It is a very simple way of Page replacement and is referred to as First in First Out.
- This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.
- This algorithm is implemented by keeping the track of all the pages in the queue.
- As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.

Advantages:

- This algorithm is simple and easy to use.
- FIFO does not cause more overhead.

Disadvantages:

- This algorithm does not make the use of the frequency of last used time rather it just replaces the Oldest Page.
- There is an increase in page faults as page frames increases.
- The performance of this algorithm is the worst.

FIFO

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

3 frames (3 pages can be in memory at a time per process)

reference string

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

7	7	7	2																		
	0	0	0																		
		1	1																		

page frames

15 page faults



Belady's Anomaly



OPR (Optimal Page Replacement)

- This algorithm mainly replaces the page that will not be used for the longest time in the future.
- Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.
- This algorithm leads to less number of page faults and thus is the best-known algorithm
- Advantages:
 - This algorithm is easy to use.
 - his algorithm provides excellent efficiency and is less complex.
- Disadvantages:
 - future awareness of the program is needed.
 - Practical Implementation is not possible because the operating system is unable to track the future request

OPR (Optimal Page Replacement)

- Replace page that will not be used for longest period of time

reference string

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

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

page frames

Least Recently Used (LRU)

- ▶ page which has not been used for the longest time in memory is the one which will be selected for replacement.
- ▶ Easy to implement , keep a list , replace pages by looking back into time.

reference string

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

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

page frames

Second Chance Page Replacement (SCPR)

- A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect reference bit of the oldest pages.

Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

3 frames (3 pages can be in memory at a time per process)



Least Frequently Used (LFU)

Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

3 frames (3 pages can be in memory at a time per process)



Segmentation

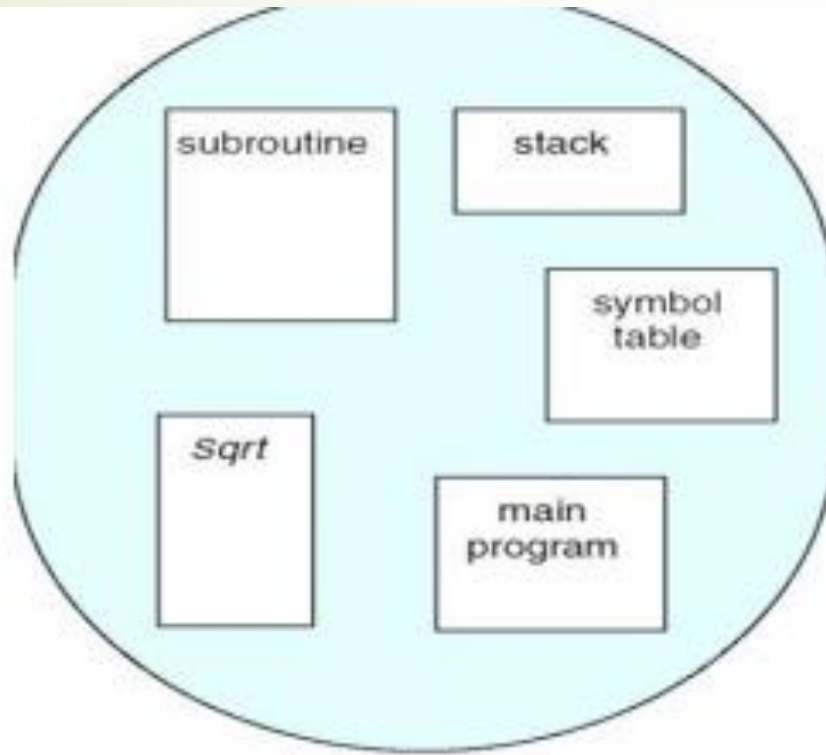
- The memory is divided into the variable size parts.
- Each parts is known as segments
- These segments are allocated to a process
- Details about the segments are stored in a table called **segment table**
- Segment table mainly contains:
 - Base Address
 - Limit



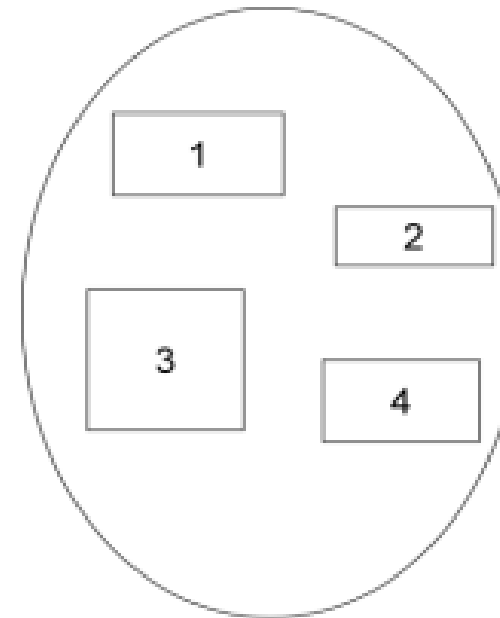
Why Segmentation?

- A process is divided into Segments
- The chunks that a program is divided into which are not necessarily all of the same sizes are called segments
- Segmentation gives the user's view of the process which paging does not give because paging is more close to the OS more than user
- It may divide the same computation function into multiple parts and those pages may or may not be loaded at same time into the memory

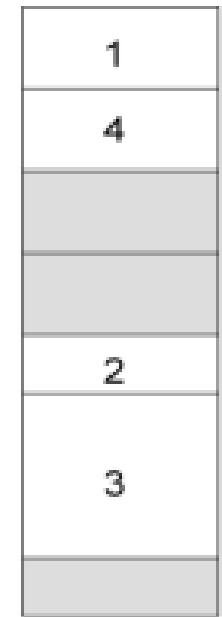
Segmentation (contd.)



logical address
Fig: User's view of program



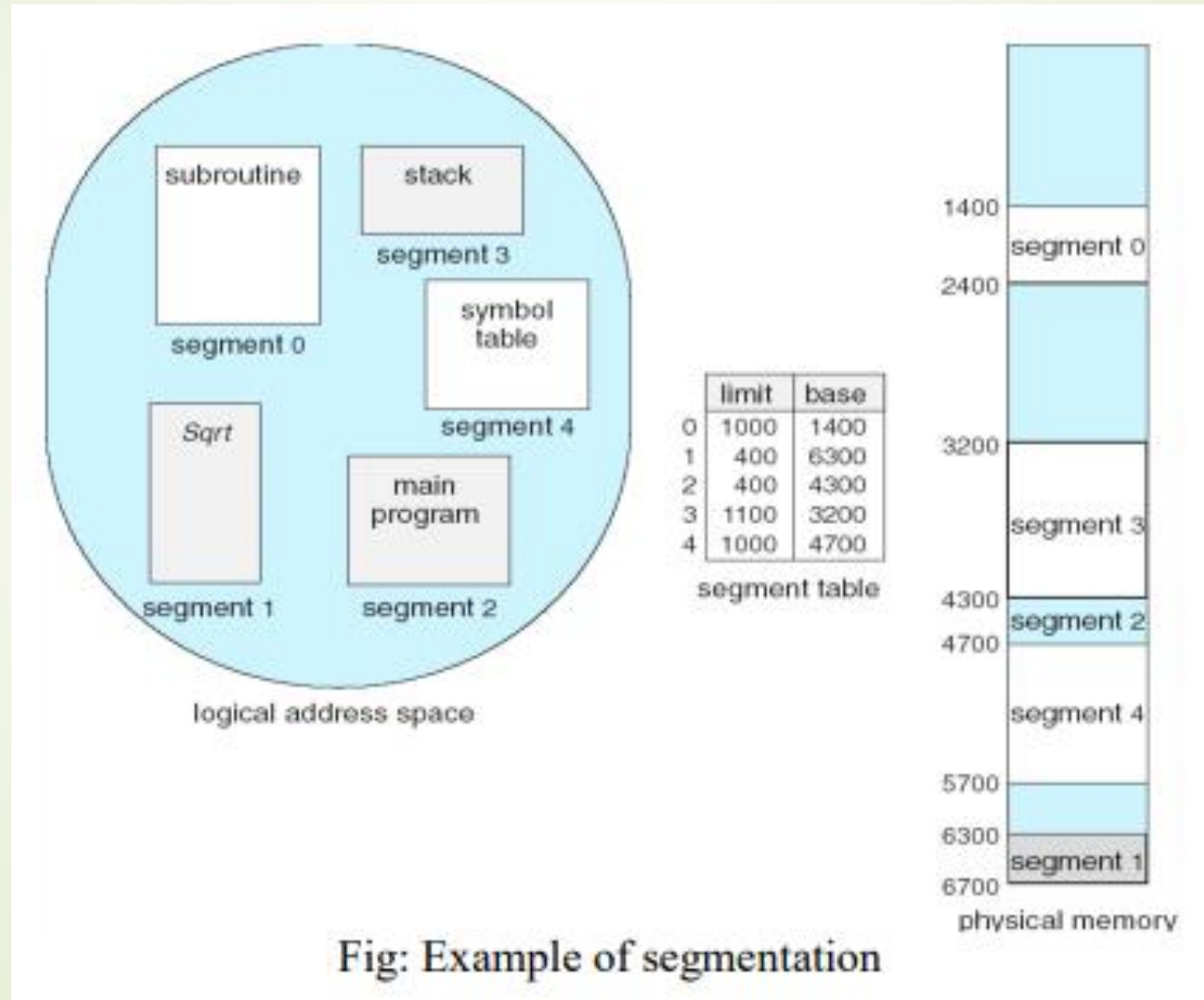
user space



physical memory space

Fig: Logical view of segmentation

Segmentation (contd.)





Adv. / Dis Adv. Of Segmentation

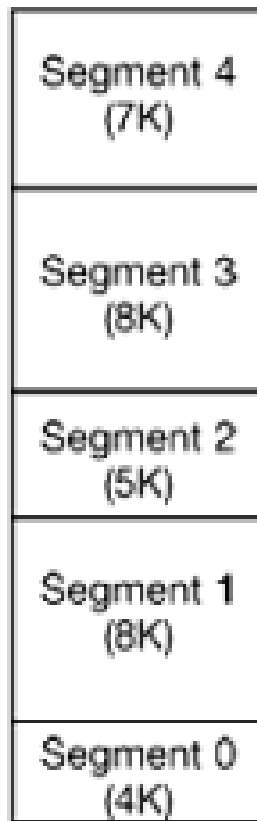
➤ Advantages

- No internal fragmentation
- Less overhead
- Easier to reallocate segments than entire address space
- Segmentation table is lesser in size than page table

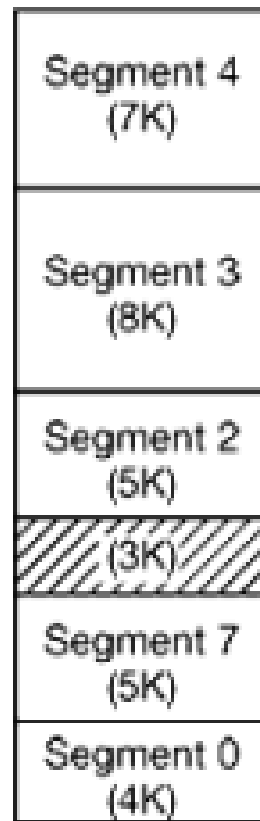
➤ Disadvantages

- May have external fragmentation
- Difficult to allocate contiguous memory to variable sized partition

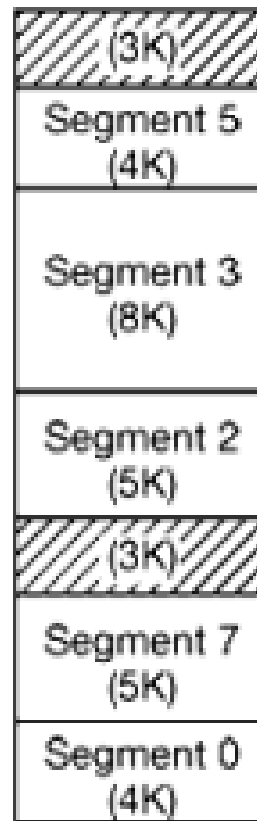
Implementation of Segmentation



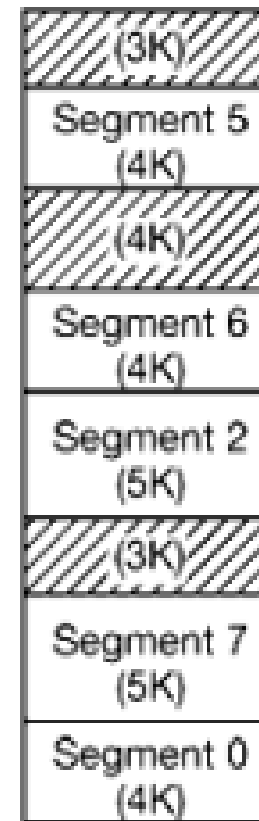
(a)



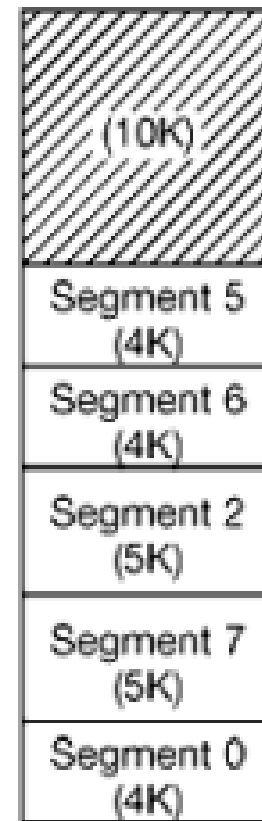
(b)



(c)

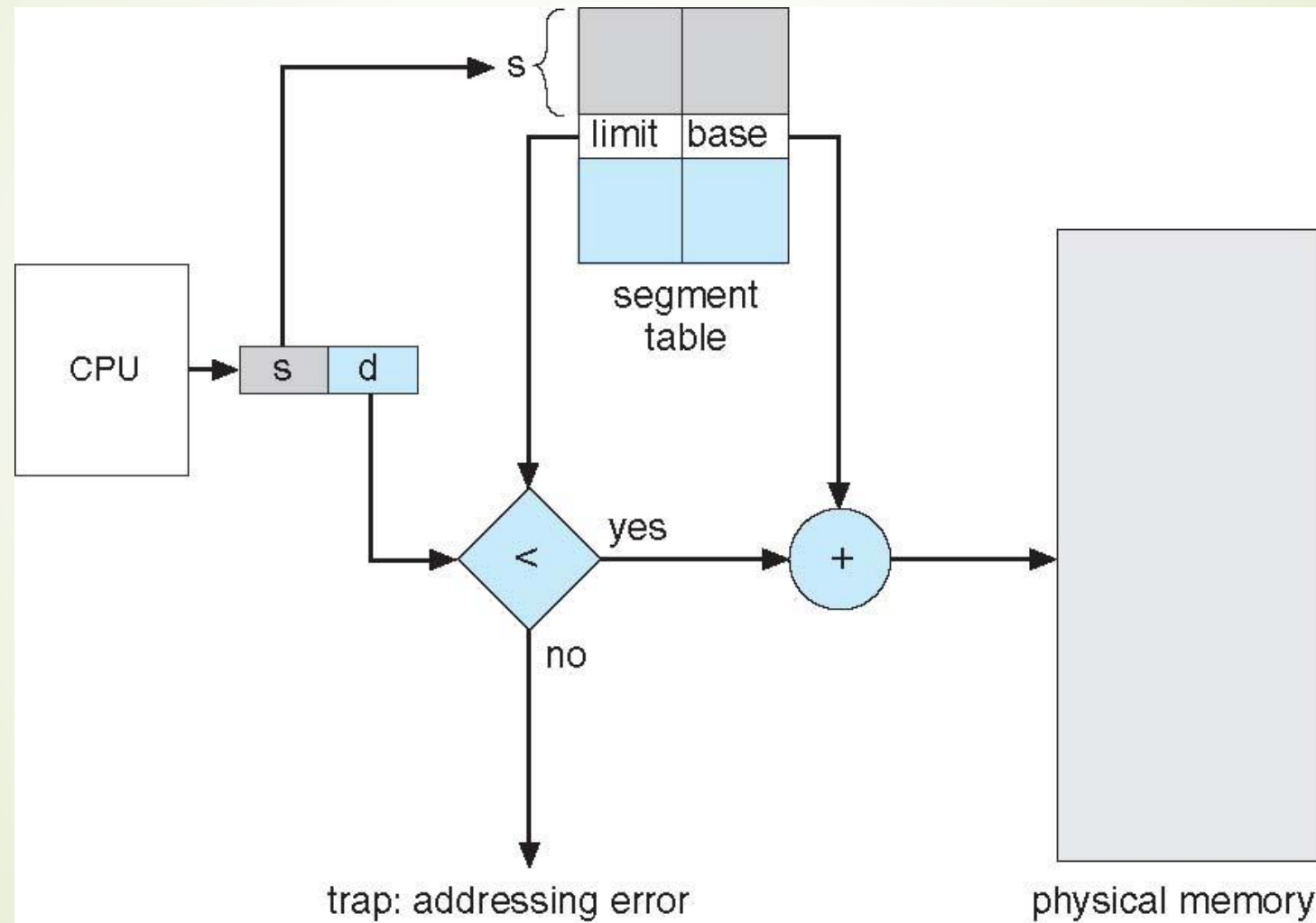


(d)



(e)

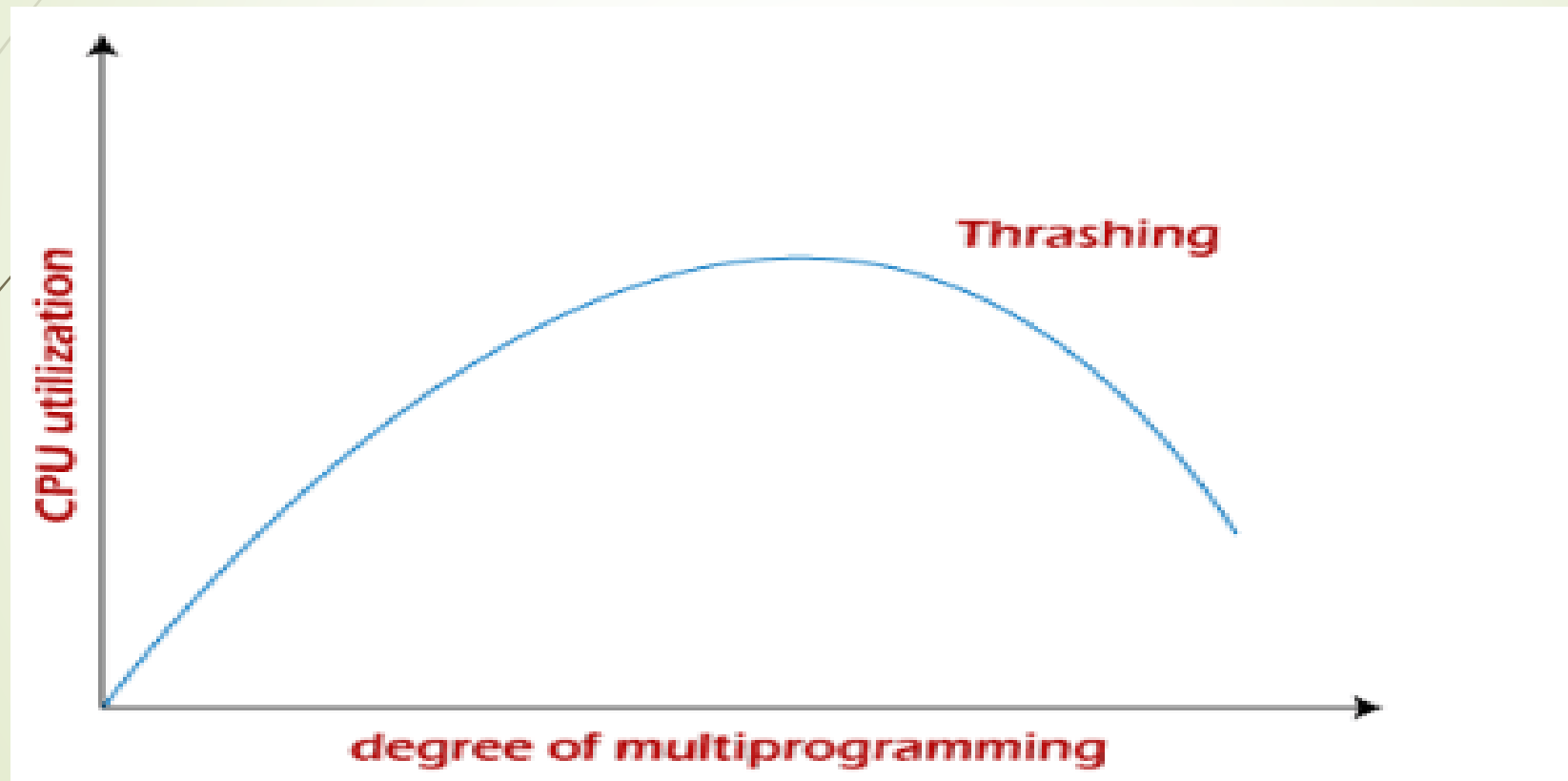
Segmentation with Paging: MULTICS



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.
- In computer science, thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults.
- 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.

Thrashing



Banker Algorithm (Practice)

Process	Allocation				Available				Max			
	A	B	C	D	A	B	C	D	A	B	C	D
Po	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				