MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Operating Systems

School of Computer Engineering and technology

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Unit IV

Unit – IV Memory Management:

Memory Partitioning: Fixed Partitioning, Dynamic Partitioning, Internal and External fragmentation, Compaction, Concept of virtual memory, Segmentation, Paging, Page allocation, Page fault, Concept of Locality of Reference, Working Set, Belady's Anomaly, Thrashing Page Replacement Algorithms: FIFO, LRU, Optimal.

I/O and File Management:

I/O Hardware: I/O devices, Device controllers, Direct Memory Access, Principles of I/O.

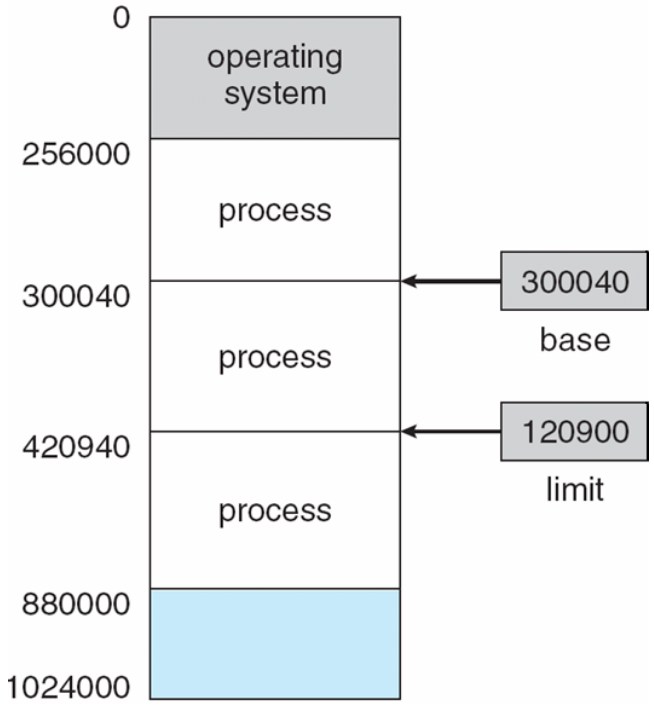
Memory Management

References

1. William Stallings, Operating System: Internals and Design Principles, Prentice Hall, ISBN-10: 0-13-380591-3, ISBN-13: 978-0-13-380591-8, 8th Edition

Memory Management

- Subdividing memory to accommodate multiple processes
  - Done by Memory Management module of OS
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time



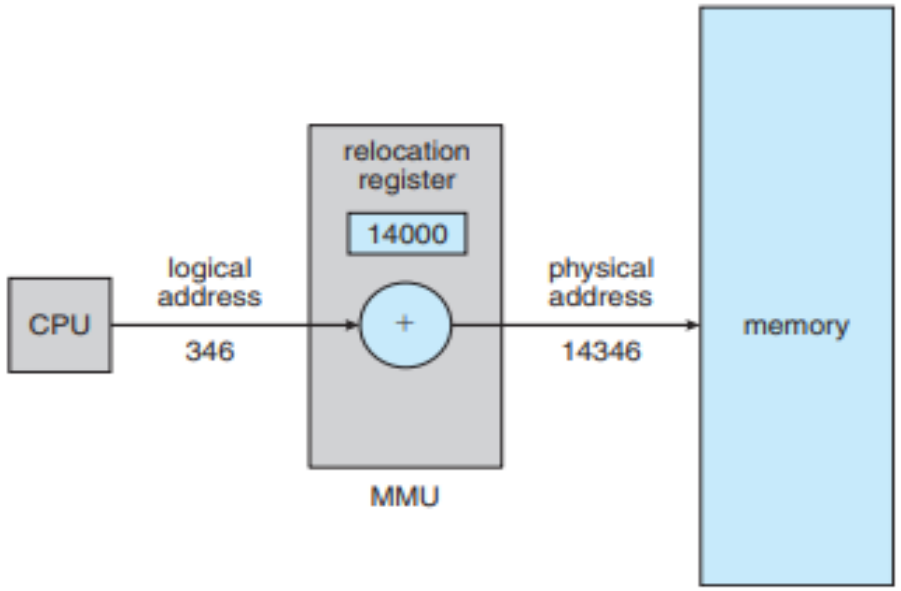
Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical Organization
- Physical Organization

Memory Management Requirements

- Relocation
  - Programmer does not know where the program will be placed in memory when it is executed
  - While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
  - Memory references must be translated in the code to actual physical memory address.
  - An address generated by the CPU is referred to as a logical address, whereas an address seen by the memory unit—that is, the one loaded into the memory-address register of the memory—is referred to as a physical address.

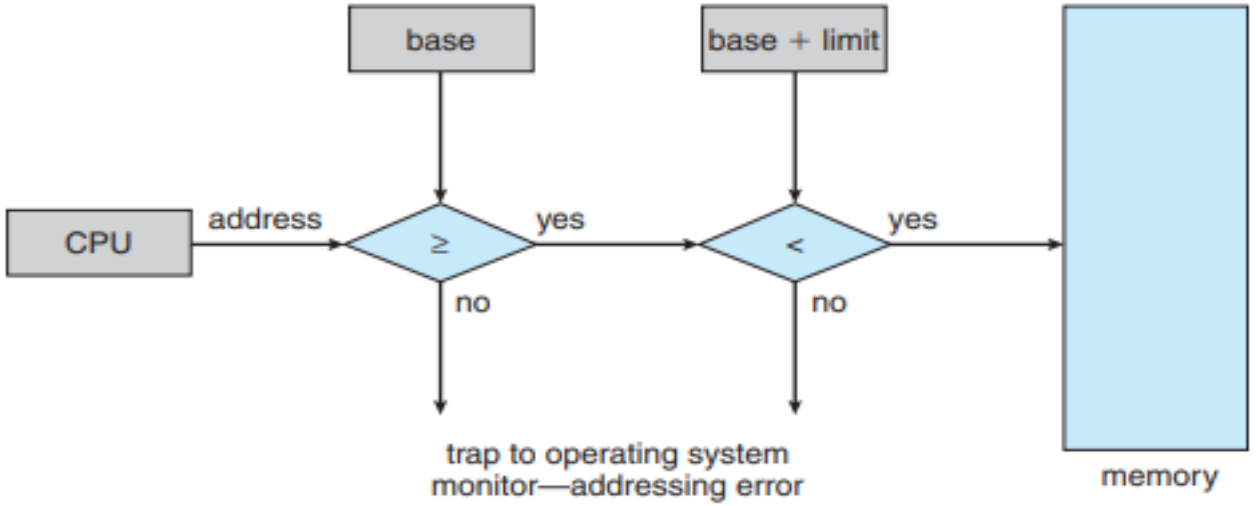
Dynamic Relocation using a relocation register



Memory Management Requirements

- Protection
  - Processes should not be able to refer memory locations in another process without permission
  - Impossible to check absolute addresses at compile time
  - Must be checked at run time

Example
•Base = 1000
•Limit = 500
•Process can access addresses **1000 to 1499** in RAM.
Case 1: CPU generates address 1200
•≥ 1000 ✔ and < 1500 ✔ → Valid access → allowed.
Case 2: CPU generates address 1550
•≥ 1000 ✔ but < 1500 ❌ → Invalid → trap to OS.



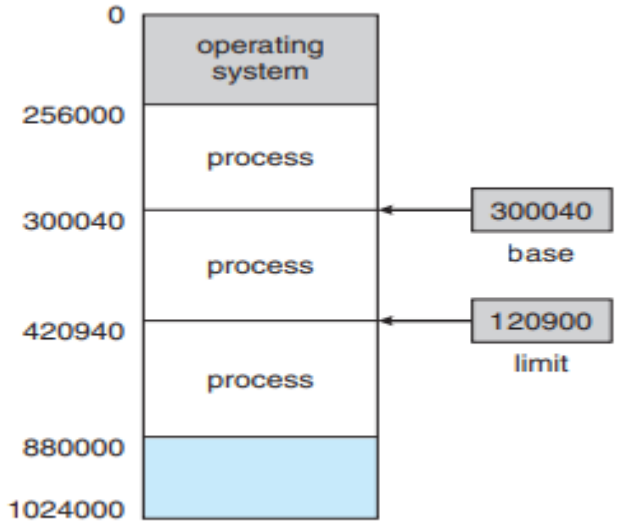
Hardware address protection with base and limit registers.

Protection can provided by using two registers,

1. Base: The base register holds the smallest legal physical memory address
2. Limit: The limit register specifies the size of the range.

For example, if the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).

Example
•Base = 1000
•Limit = 500
•Process can access addresses **1000 to 1499** in RAM.
Case 1: CPU generates address 1200
•≥ 1000 ✔ and < 1500 ✔ → Valid access → allowed.
Case 2: CPU generates address 1550
•≥ 1000 ✔ but < 1500 ❌ → Invalid → trap to OS.



A base and a limit register define a logical address space.

Memory Management Requirements

- Sharing
  - Allow several processes to access the same portion of memory
  - Better to allow each process access to the same copy of the program rather than have their own separate copy

Memory Management Requirements

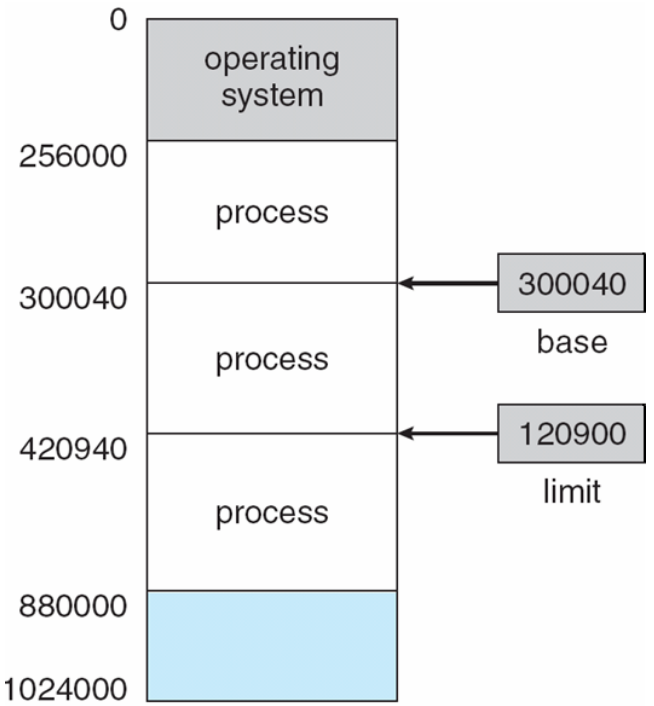
- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes

Memory Management Requirements

- Physical Organization
  - Memory available for a program plus its data may be insufficient
    - **Overlaying-** program and data are organized in such a way that various modules can be assigned the same region of memory with a main program responsible for switching the modules in and out as needed.
  - Programmer does not know how much space will be available

Memory Partitioning

- OS occupies some fixed portion of main memory and that the rest of main memory is available for use by multiple processes.
- The simplest scheme for managing this available memory is to partition it into regions.
- Types-
  - Fixed Partitioning
  - Dynamic Partitioning

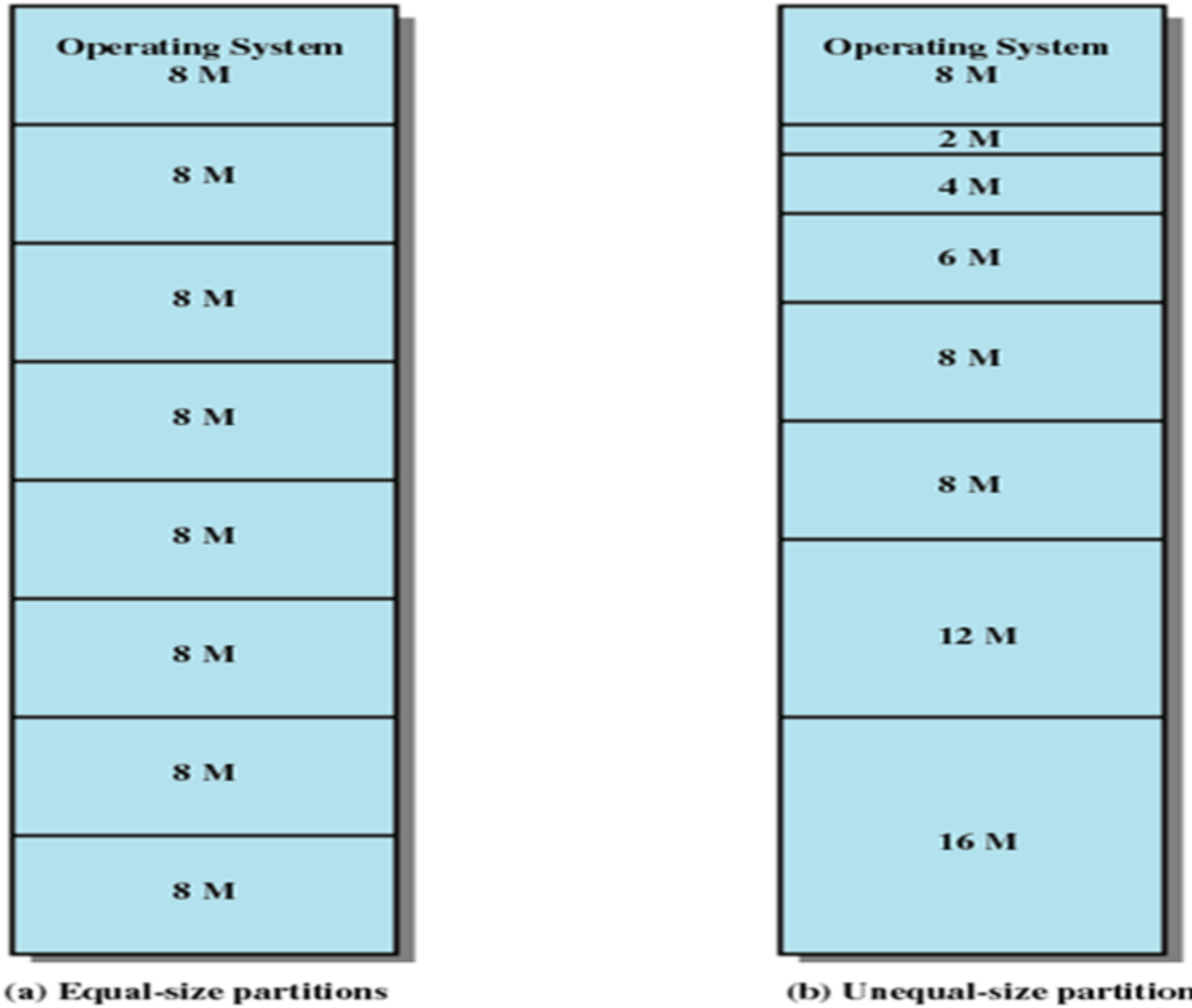


Fixed Partitioning

- Partition regions with fixed boundaries.
- Partition Sizes-
  - Two alternatives
    1. Equal-size fixed partitions
    2. Unequal-size fixed partitions

Equal-size partitions

- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition
- A program may not fit in a partition. The programmer must design the program with **overlays**
- Use of Main memory is inefficient in this case. Any program, no matter how small, occupies an entire partition. Here, there is wasted space internal to a partition. This is called **Internal fragmentation**.



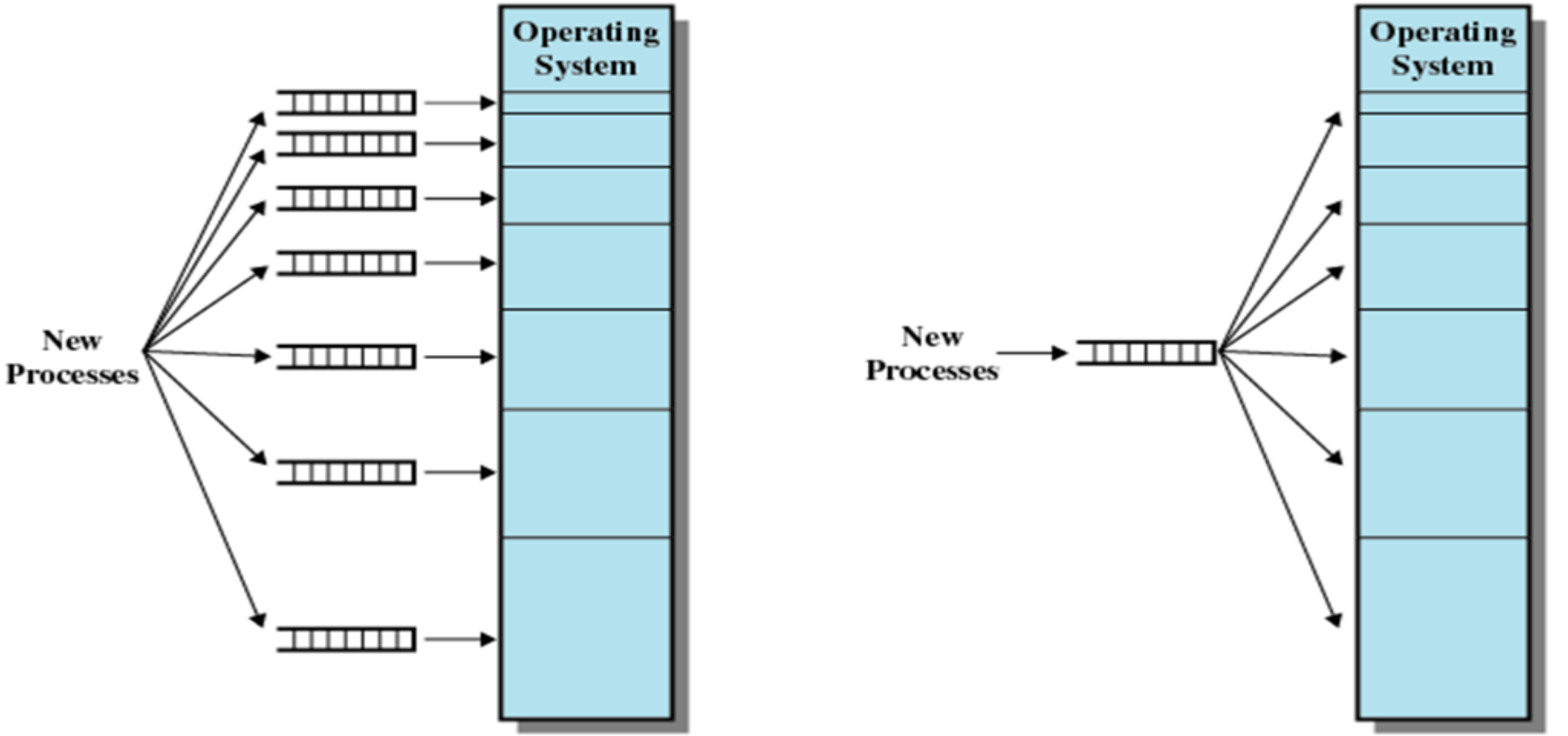
(a) Equal-size partitions

(b) Unequal-size partitions

Example of Fixed Partitioning of a 64-Mbyte Memory

Placement Algorithm with Partitions

- Equal-size partitions
  - Because all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - Can assign each process to the smallest partition within which it will fit
  - Queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition



(a) One process queue per partition

(b) Single queue

Memory Assignment for Fixed Partitioning

Disadvantages

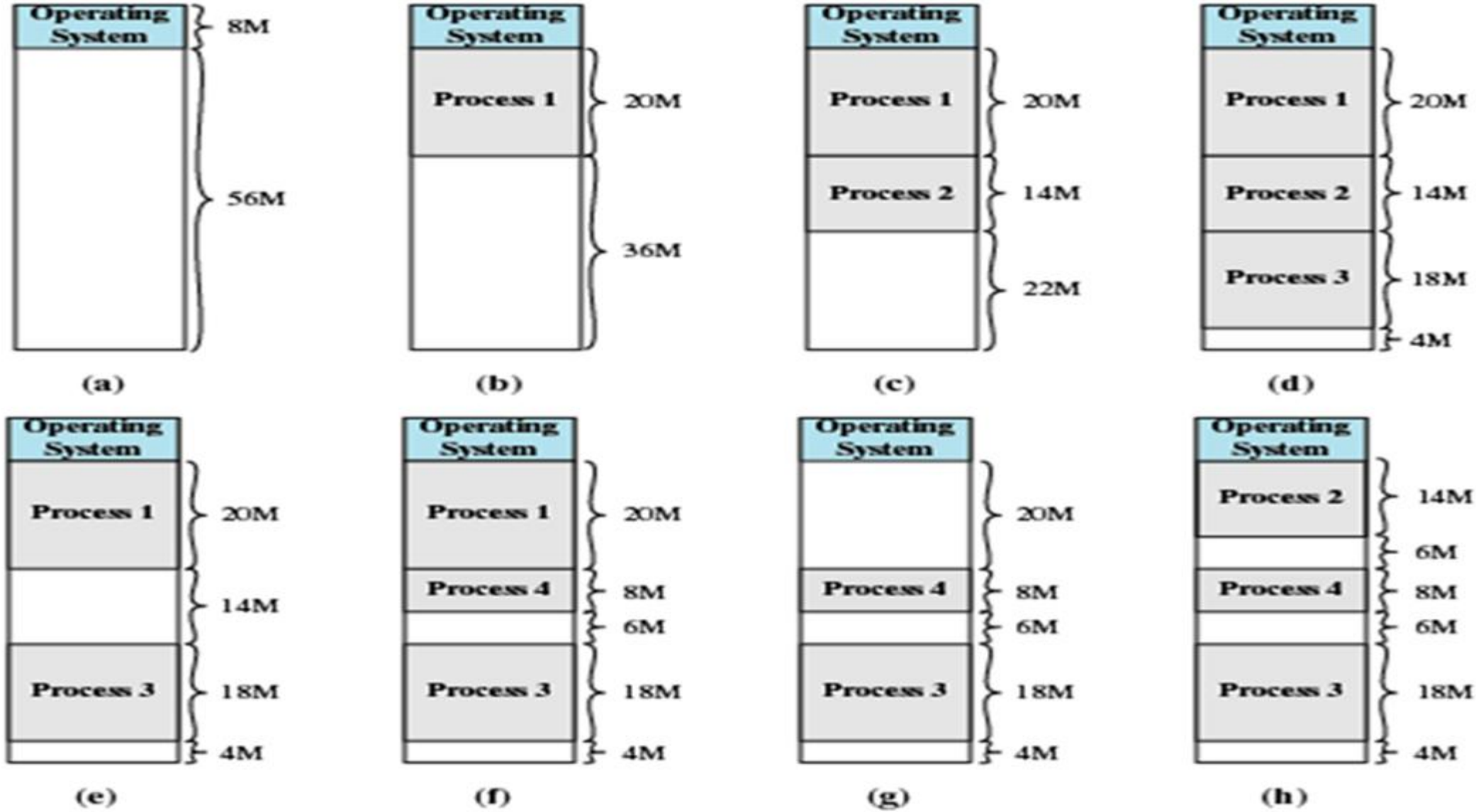
- The number of partitions specified at system generation time limits the number of active processes in the system.

Small processes will not utilize space efficiently.

- As well as, it is not reasonable to know the requirement of the process beforehand.

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually it leads to a situation in which there are a lot of small holes in the memory. This is called **external fragmentation.**
- Overcome :
  - Must use **compaction** to shift processes so they are contiguous and all free memory is in one block.



The Effect of Dynamic Partitioning

What is compaction ?

Compaction is a technique in which the free space is collected in a large memory chunk to make some space available for processes.

- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.

- Compaction refers to combining all the empty spaces together

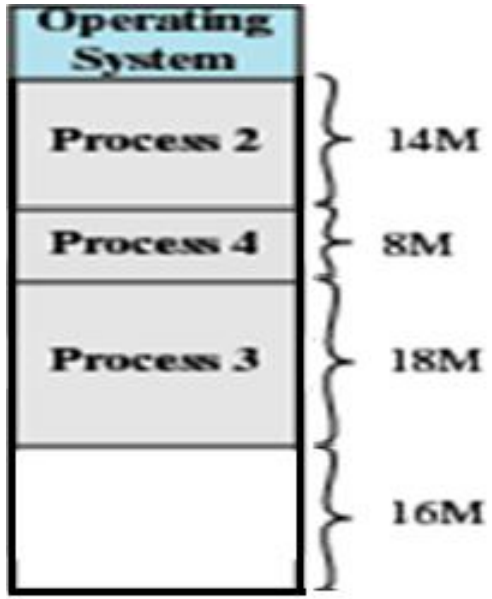
- Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time.

- It moves all the processes to one end and leaves one large free space for incoming jobs, instead of numerous small ones.

- In compaction, the system also maintains relocation information and it must be performed on each new allocation of process to the memory

Compaction Problems

- It wastes the processor time in shifting processes.
- Compaction needs the dynamic relocation capability.



After Compaction

Differences between internal and external fragmentation

Basis for Comparison	Internal Fragmentation	External Fragmentation
Basic	It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
Occurrence	When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
Solution	The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging and segmentation.

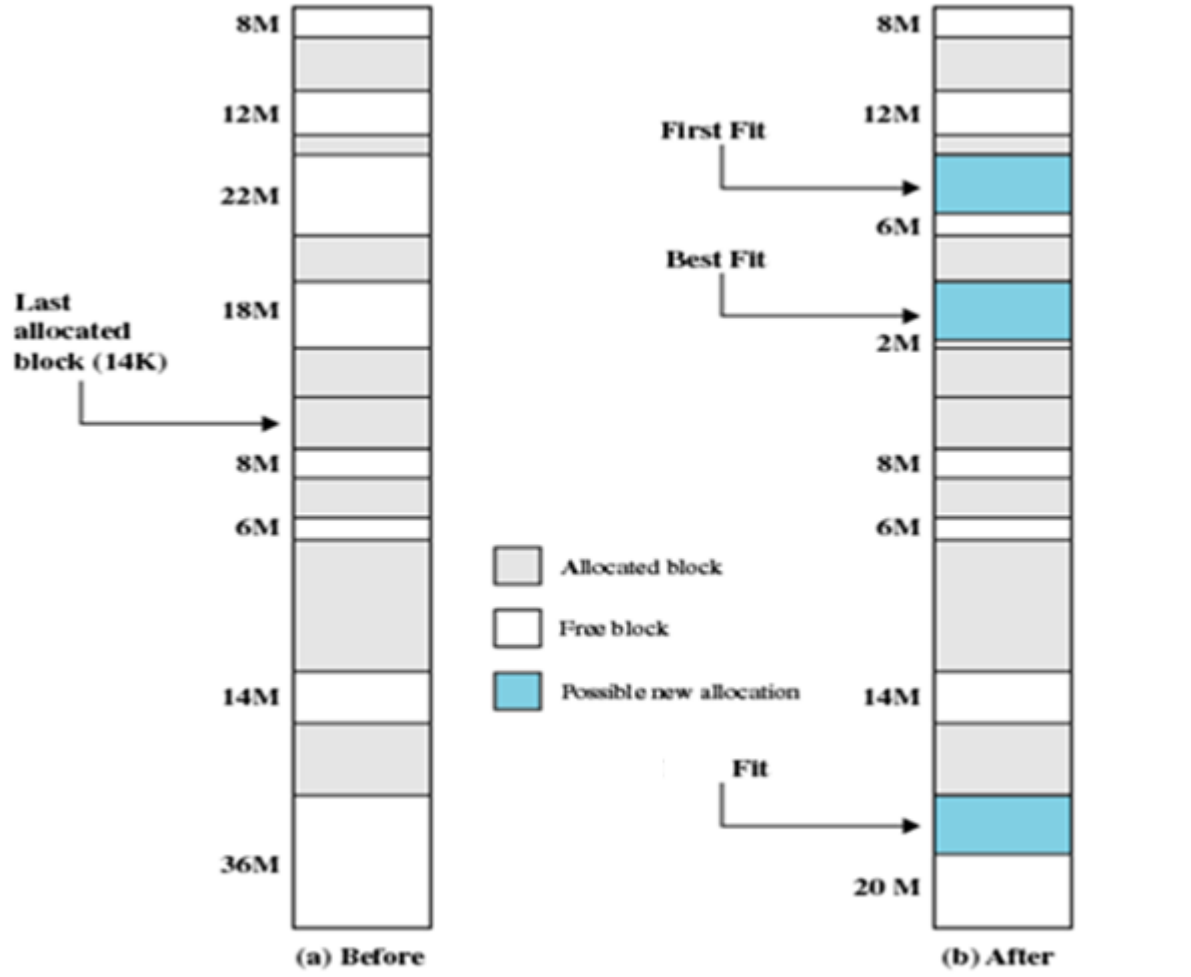
MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process.
- Types:
  - Best-fit algorithm
  - First-fit algorithm
  - Worst-fit algorithm

Dynamic Partitioning Placement Algorithm

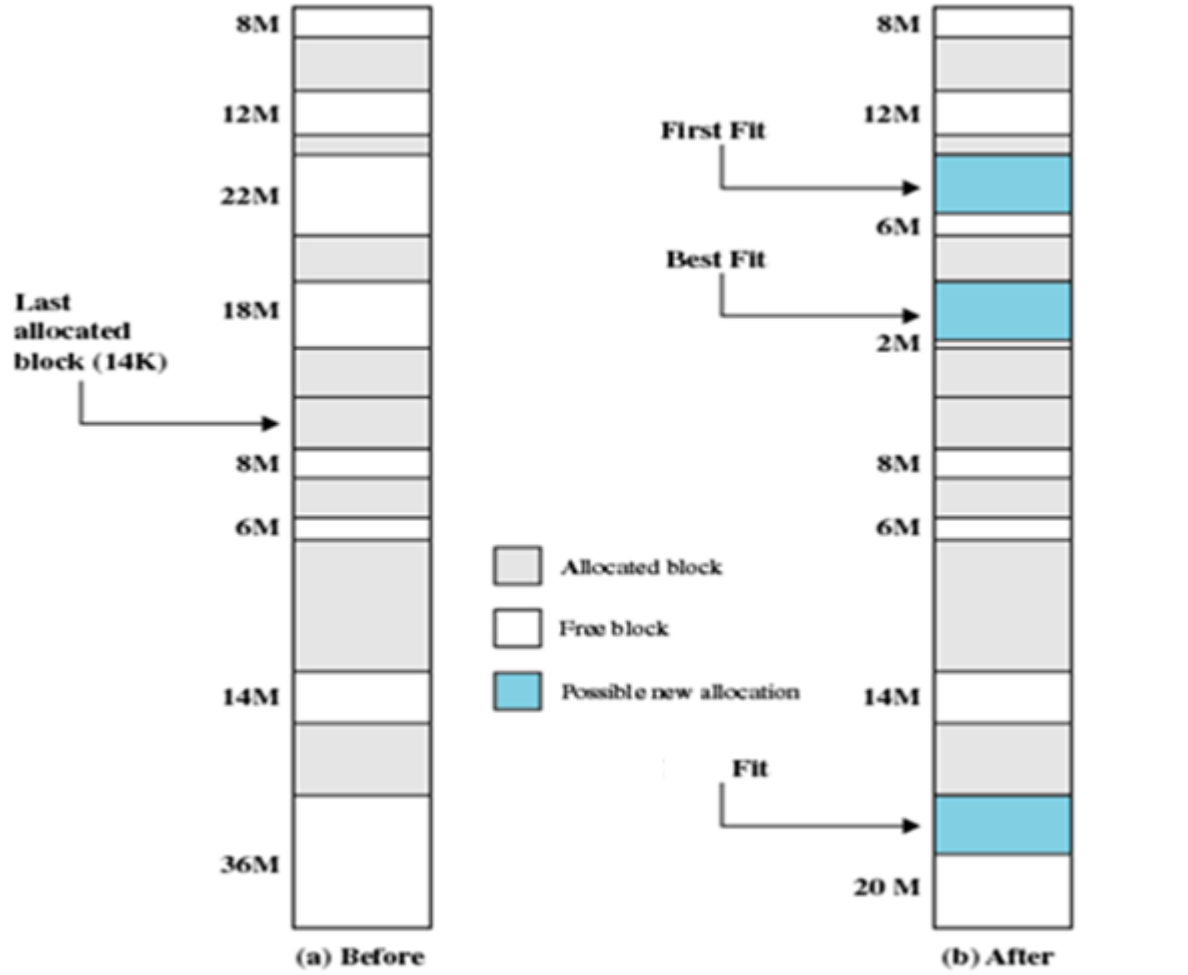
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often



Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning Placement Algorithm

- First-fit algorithm
  - Scans memory from the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block



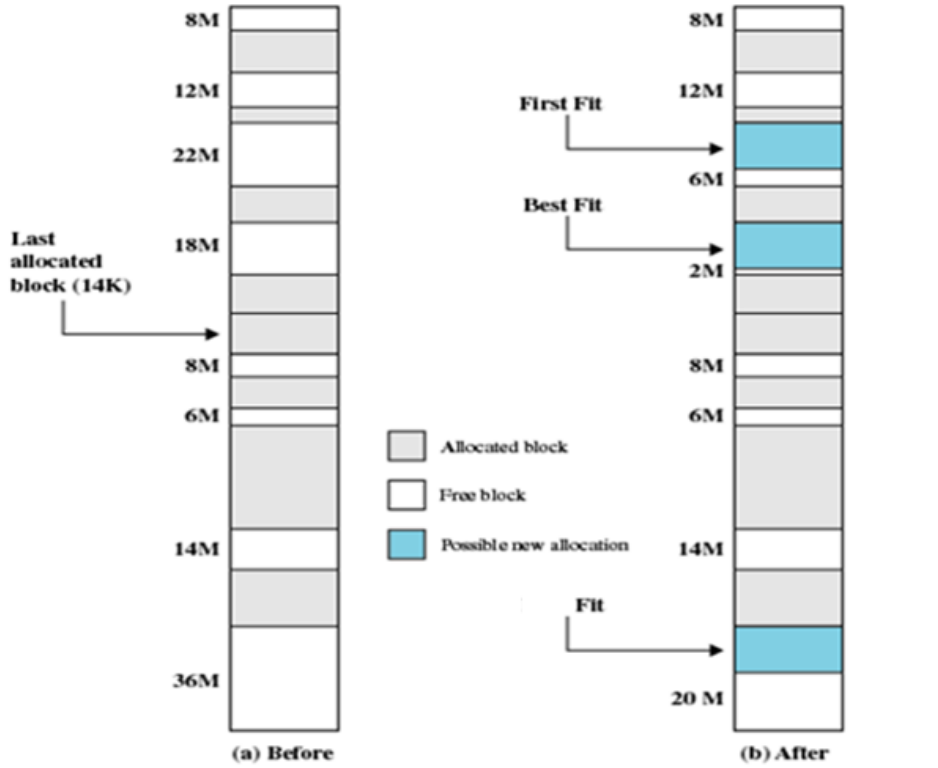
Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Dynamic Partitioning Placement Algorithm

- **Worst Fit**

Allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory.

If a large process comes at a later stage, then memory will not have space to accommodate it.

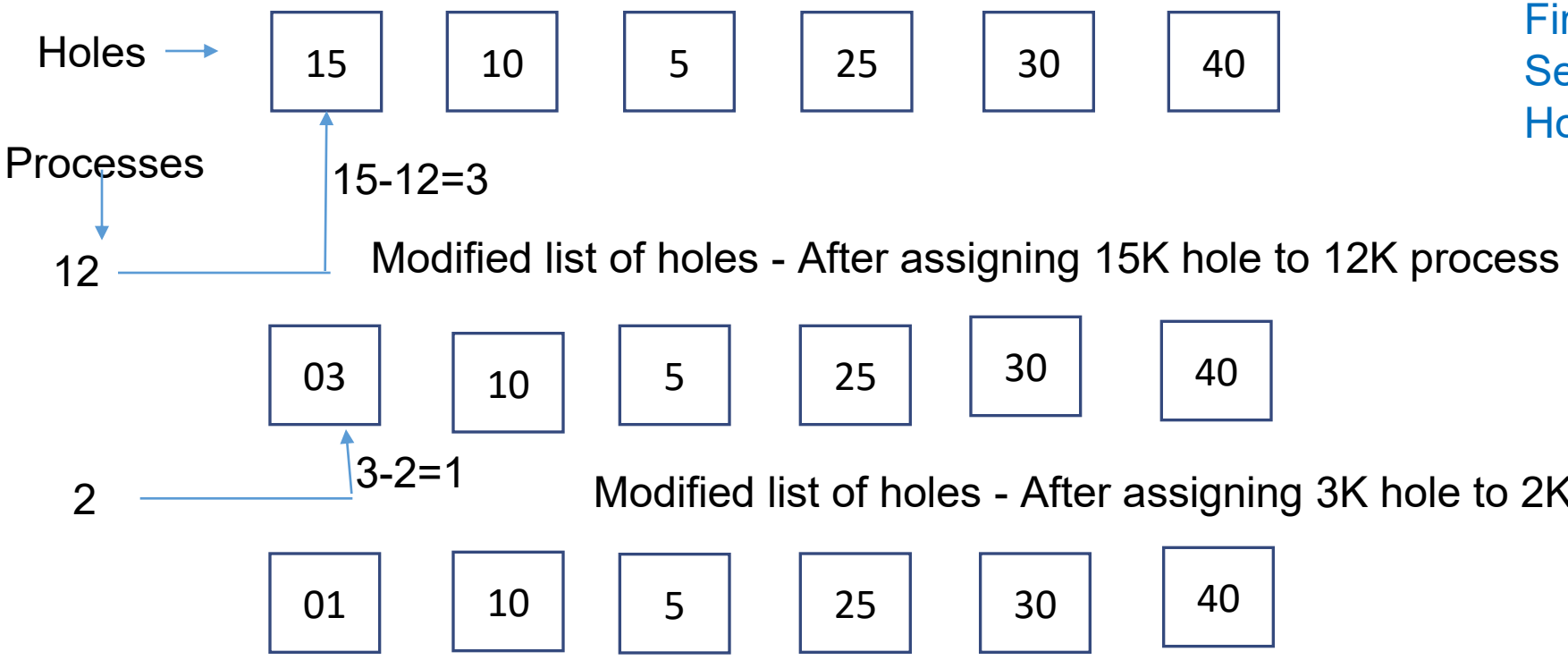


Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Example 1

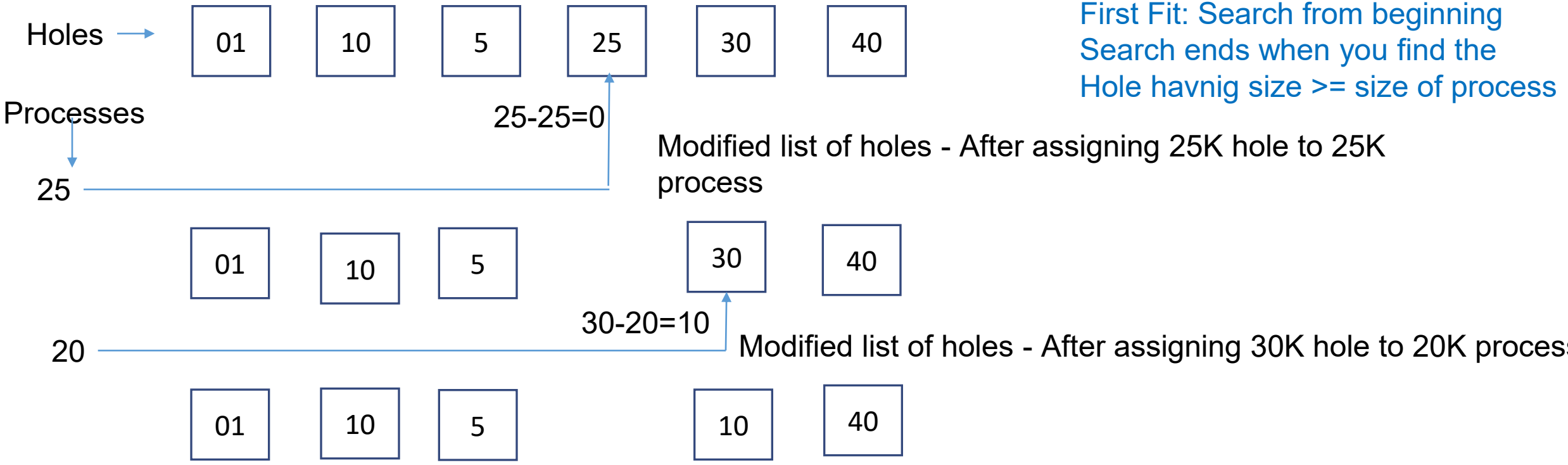
Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in first fit, best fit, worst fit?

Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in **first fit**



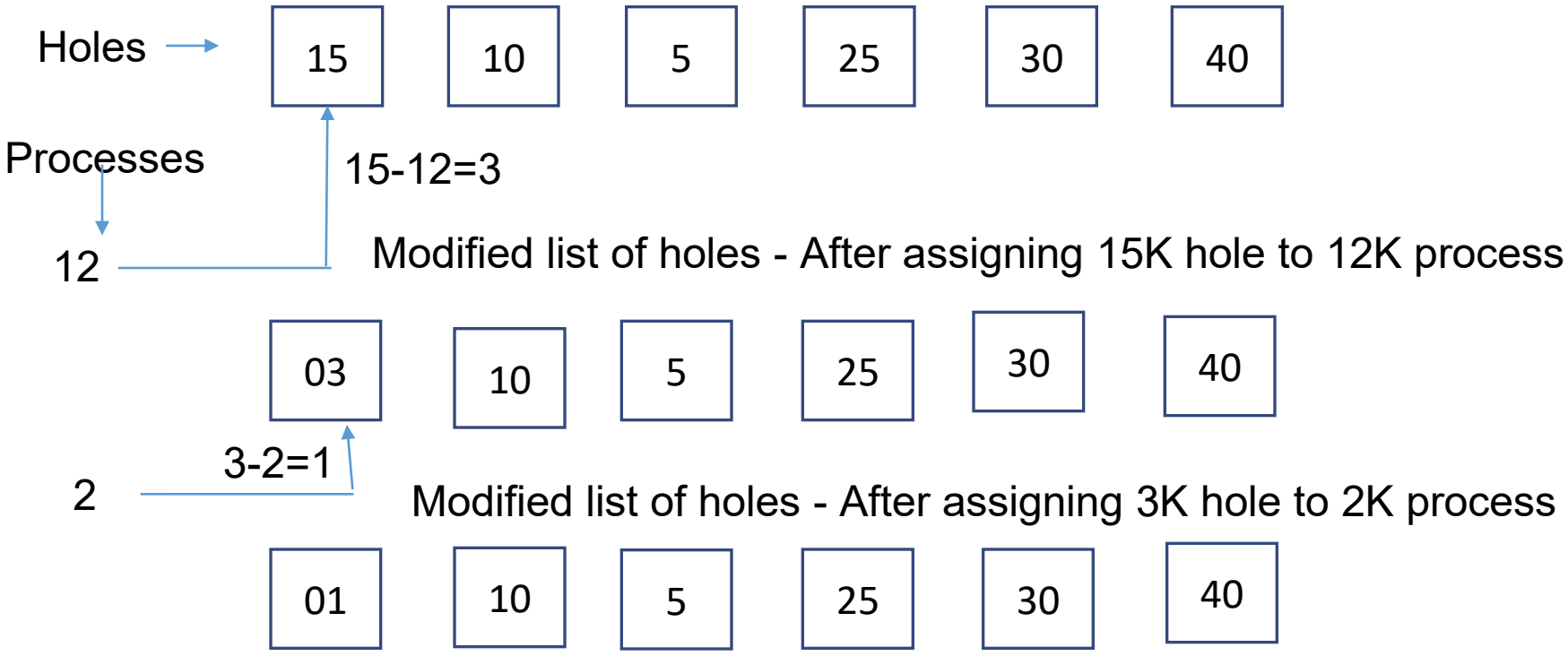
First Fit: Search from beginning
Search ends when you find the
Hole havnig size >= size of process

Continued ….



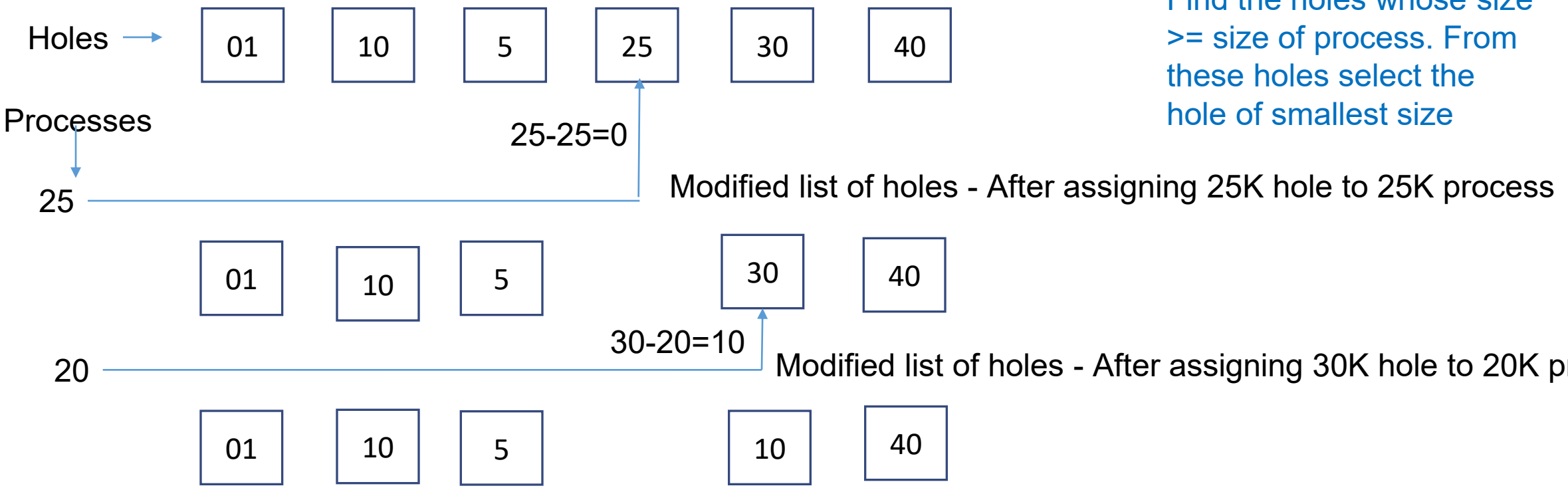
First Fit: Search from beginning
Search ends when you find the
Hole havnig size >= size of process

Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in **Best fit**



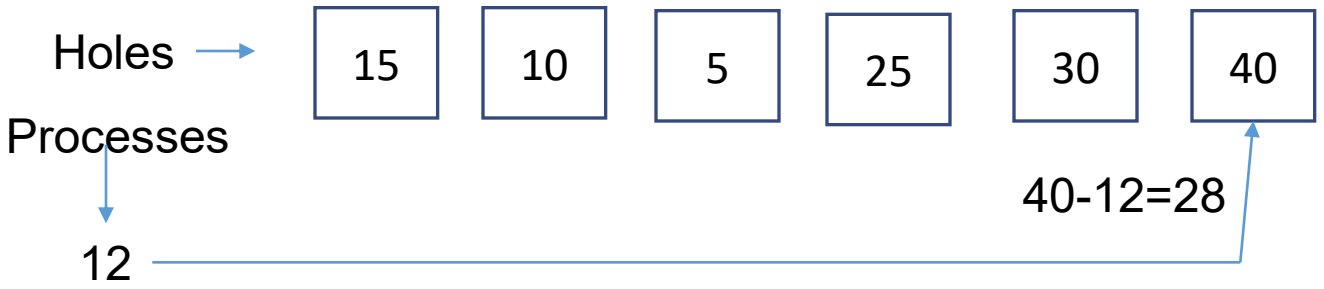
Best Fit: Search entire list Find the holes whose size >= size of process. From these holes select the hole of smallest size

Continued ….

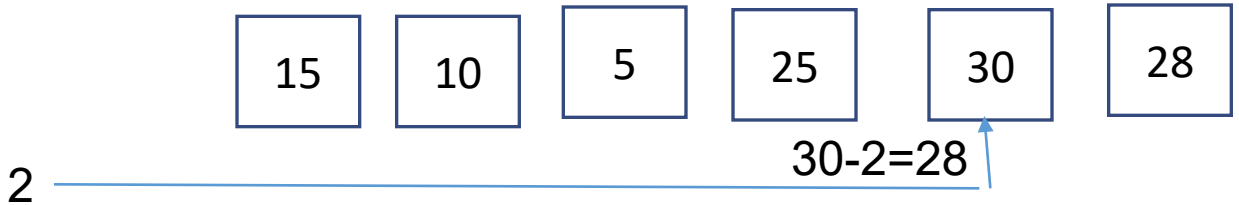


Best Fit: Search entire list Find the holes whose size >= size of process. From these holes select the hole of smallest size

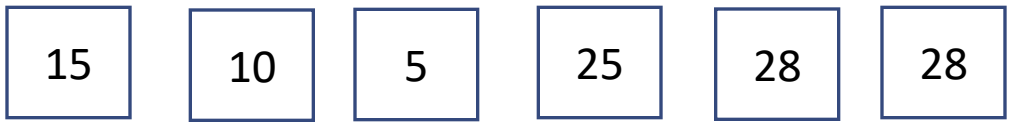
Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in **Worst fit**



Modified list of holes - After assigning 40K hole to 12K process

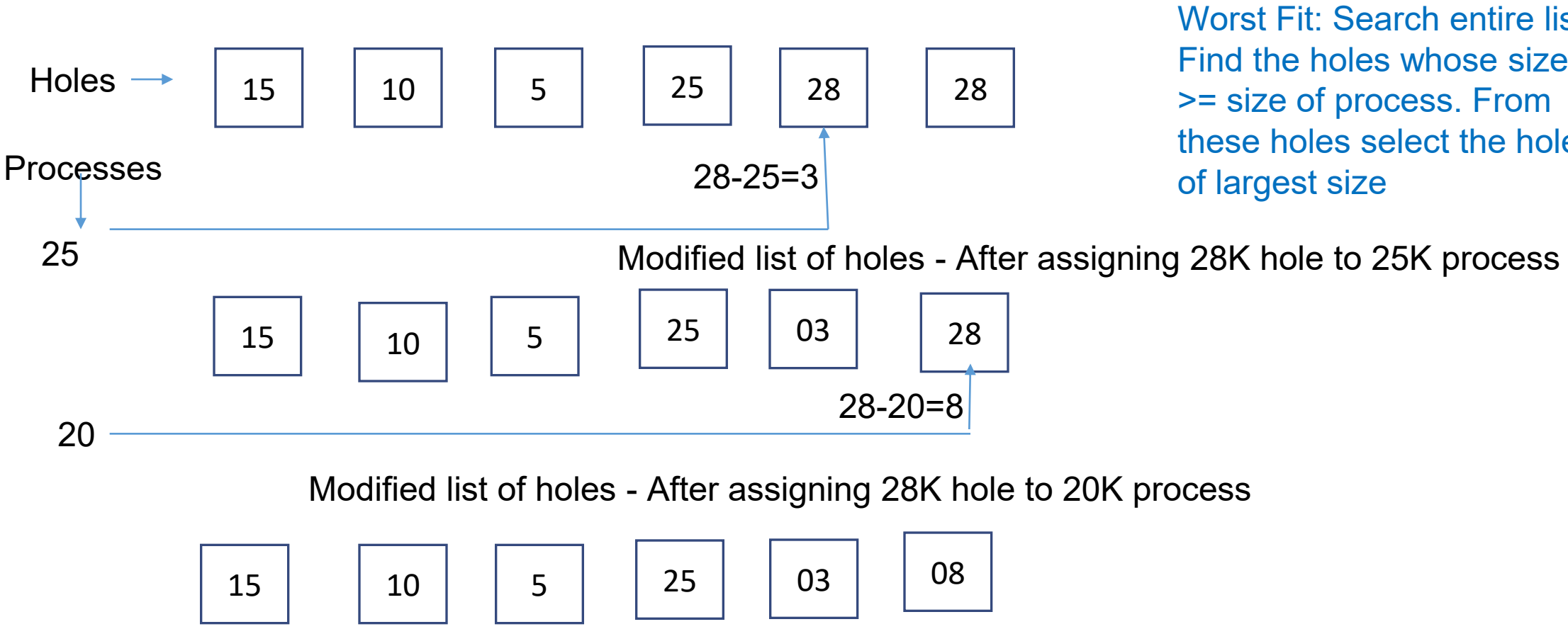


Modified list of holes - After assigning 30K hole to 2K process



Worst Fit: Search entire list Find the holes whose size >= size of process. From these holes select the hole of largest size

Continued ….



Worst Fit: Search entire list Find the holes whose size >= size of process. From these holes select the hole of largest size

Example 2

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?

Solution

Memory Partition

Partition	Memory Size Available
Partition 1	100 KB
Partition 2	500 KB
Partition 3	200 KB
Partition 4	300 KB
Partition 5	600 KB

Process Requires

Process	Memory Required
Process 1	212 KB
Process 2	417 KB
Process 3	112 KB
Process 4	426 KB

Example 2

Solution:
First-Fit:
Step 1->212K is put in 500K partition. (created new partition of size 288K = 500K - 212K).
Step 2- >417K is put in 600K partition. (created new partition of size 183K = 600K - 417K).
Step 3-> 112K is put in 288K partition

(Process 426K must wait.)

Best-Fit:
Step 1->212K is put in 300K partition. (created new partition of size 82K = 300K - 212K).
Step 2->417K is put in 500K partition. (created new partition of size 83K = 500K - 417K).
Step 3->112K is put in 200K partition. (created new partition of size 88K = 200K - 112K).
Step 3->426K is put in 600K partition. (created new partition of size 174K = 600K - 426K).

Example 2

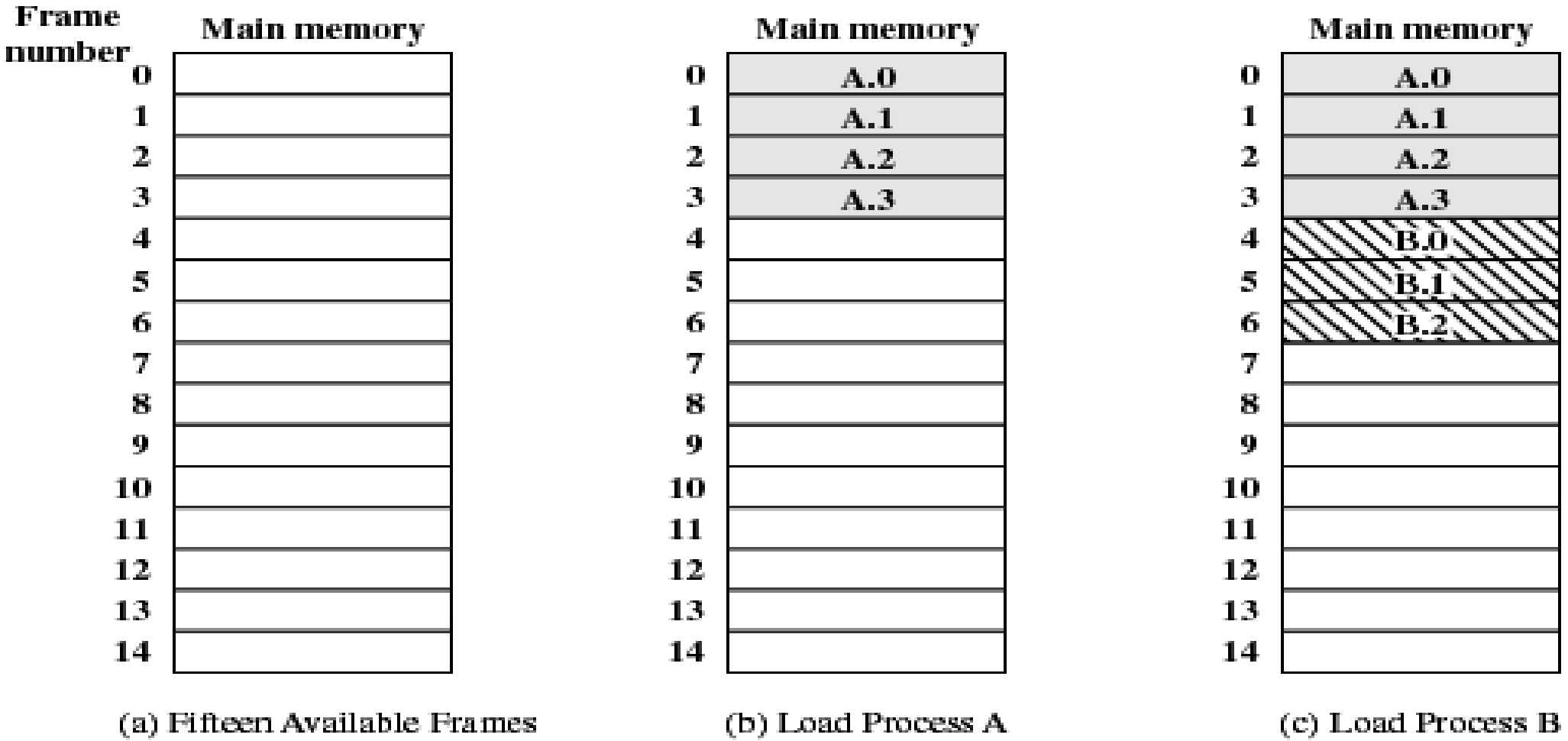
Solution:
Worst-Fit:
Step 1->212K is put in 600K partition. (created new partition of size 388K = 600K - 212K).
Step 2->417K is put in 500K partition. (created new partition of size 83K = 500K - 417K).
Step 3->112K is put in 388K partition. (created new partition of size 276K = 388K - 112K).
Step 4-> 426K must wait.

In this example, Best-Fit turns out to be the best.

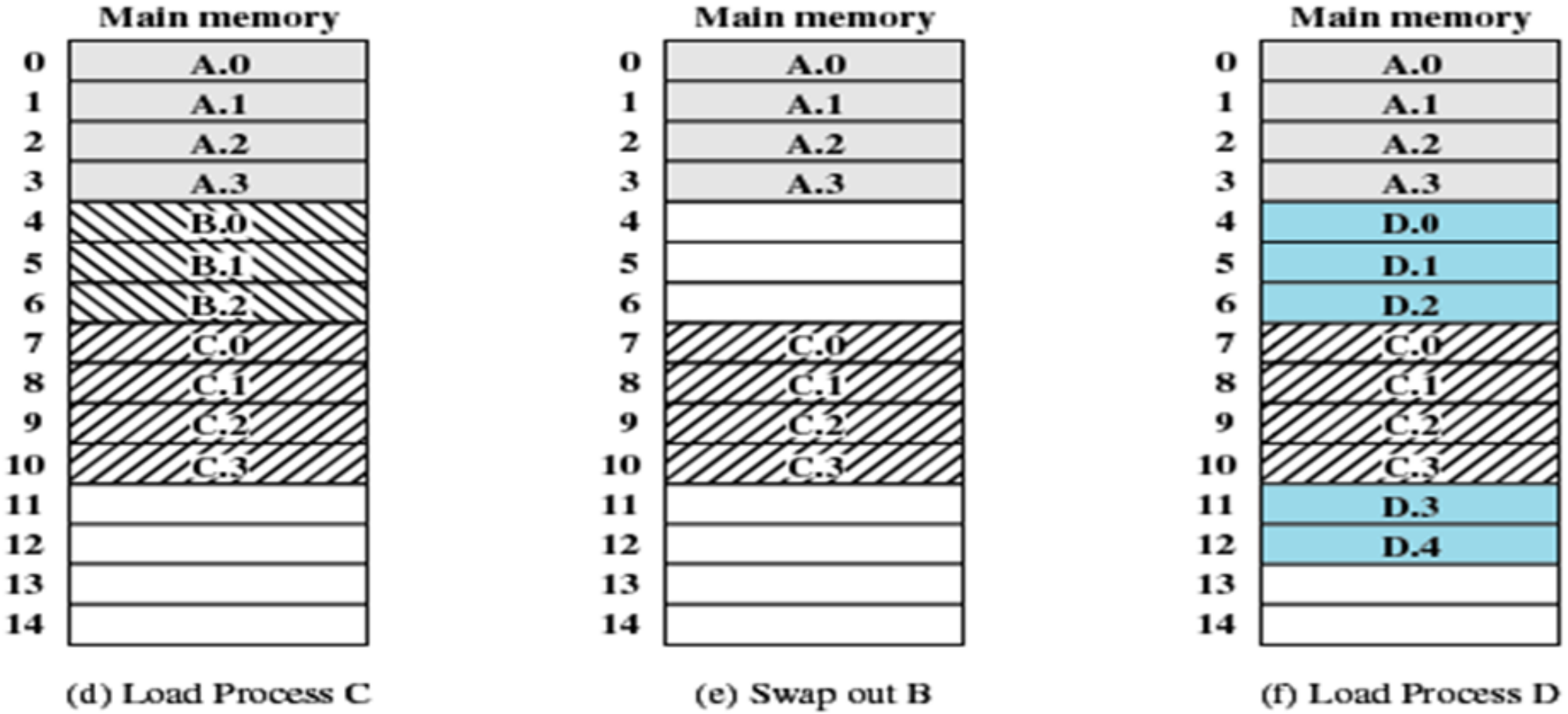
Paging

- Main memory is partitioned into equal fixed size chunks that are relatively small ---- called **frames**
- Each process is divided into small fixed sized chunks of the same size ---- called **pages**
- At a given point of time, some frames are in use & some are free
- Suppose process A stored on disk, consists of four pages.
- When process is to be loaded , OS finds four free frames & loads A's pages
- These frames need not be contiguous
- OS maintains a page table for each process
- Page table consists of frame location for each page of the process

Assignment of Process Pages to Free Frames

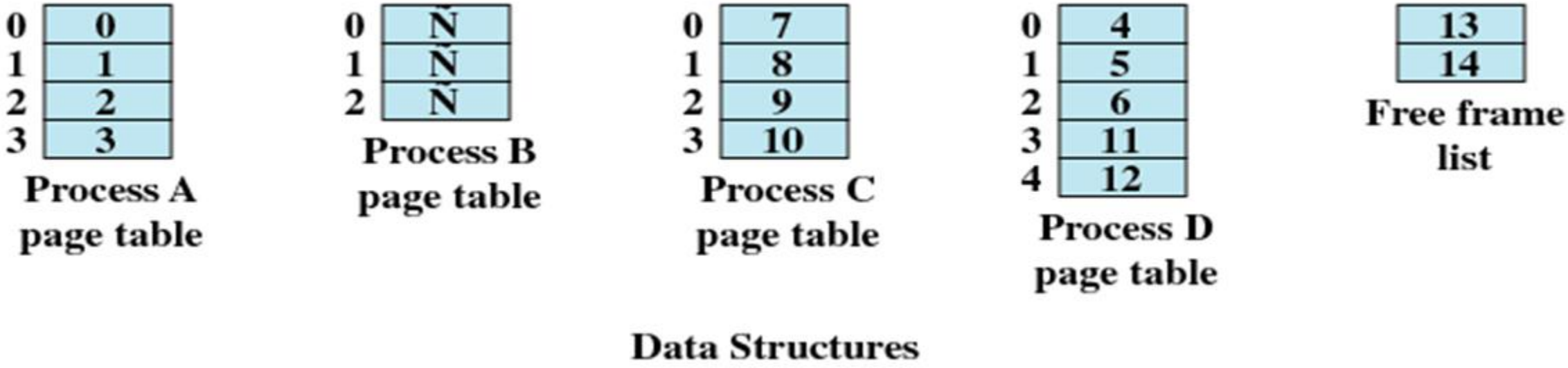


Assignment of Process Pages to Free Frames



Assignment of Process Pages to Free Frames

Page Tables for Example



Logical Address in Paging

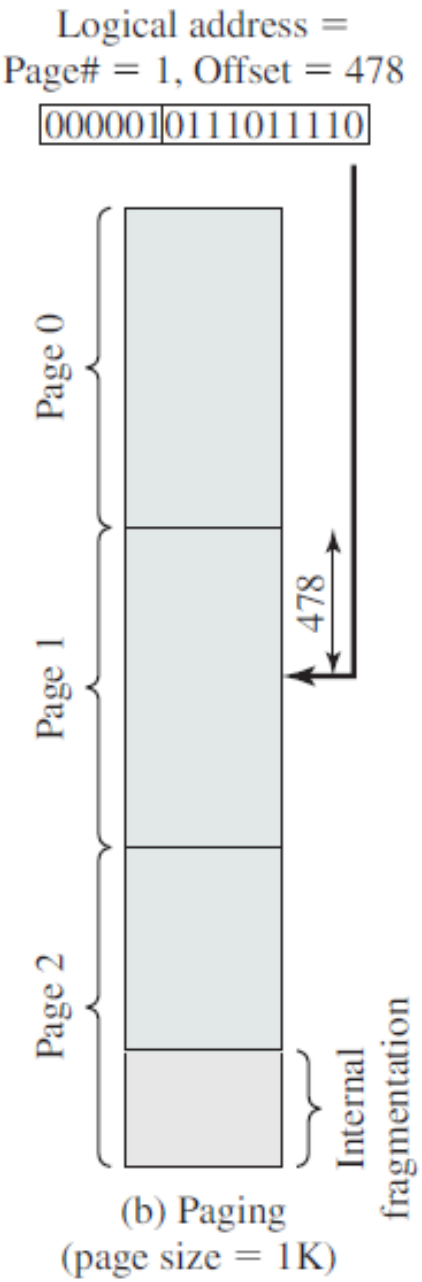
•Since the process size = 2700 bytes,

• it requires: [2700/1024]=3 pages
So, the process is divided into **Page 0, Page 1, and Page 2**.

•The last page (Page 2) is not fully filled (only 652 bytes used), which causes **internal fragmentation**

•With page size = 1024, split the logical address into:
•**Page number** = 1502 ÷ 1024 = **1**
•**Offset** = 1502 mod 1024 = **478**
•So the logical address is represented as:
•Page # = 1
•Offset = 478

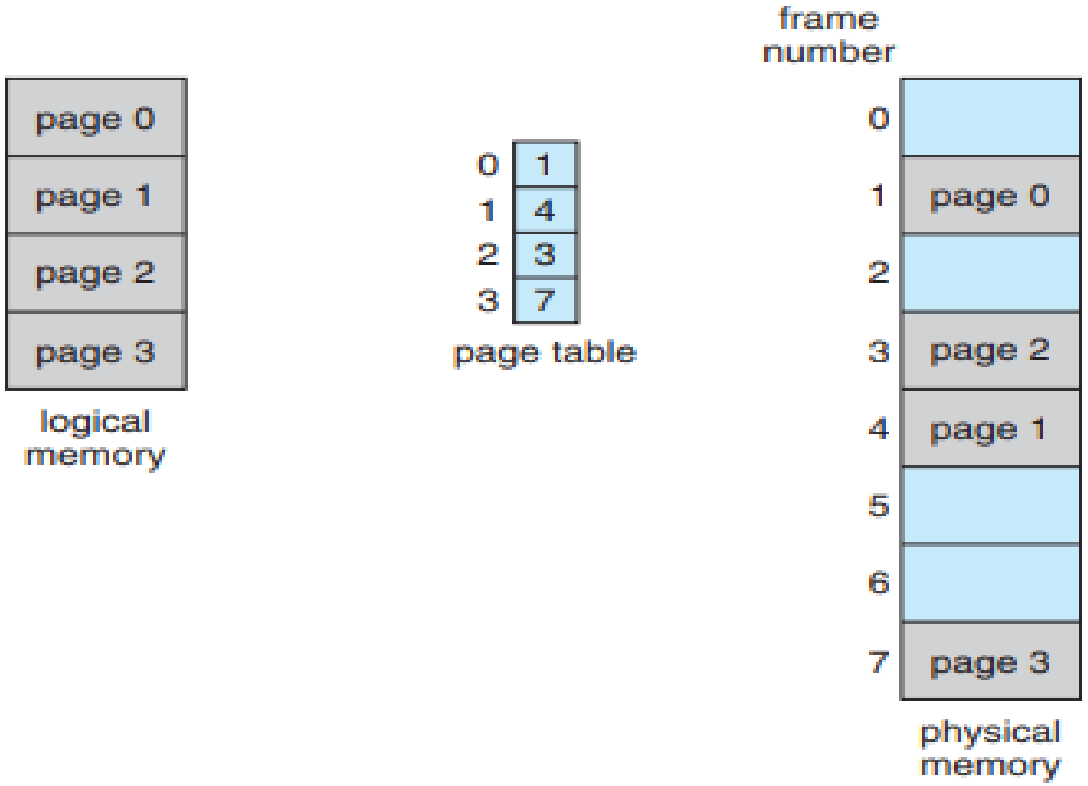
(a) Partitioning



(b) Paging
(page size = 1K)

- **Page number (p)** – tells *which page* of the process we need.
- **Offset (d)** – tells the *exact position* (or displacement) **inside that page**.
- The **offset** is the distance (or position) of a specific byte **within a page**.

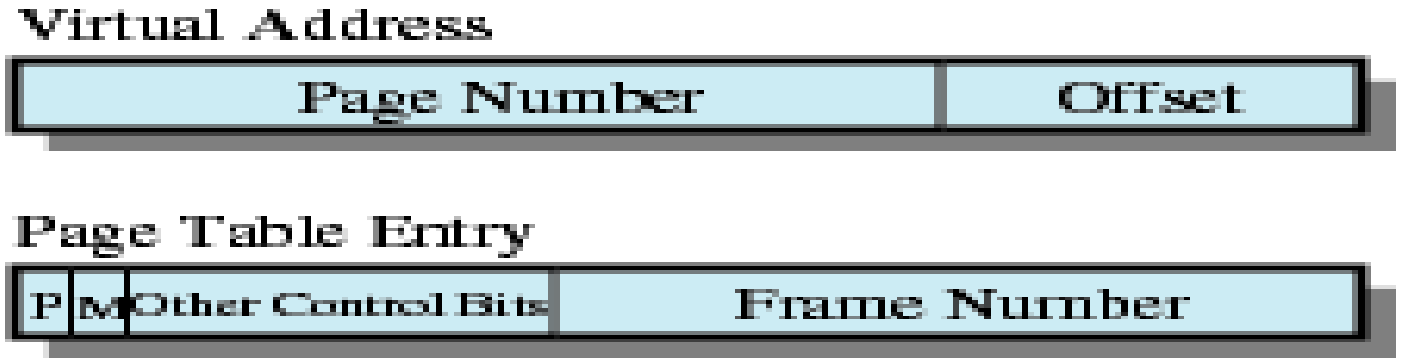
Paging model of logical and physical memory



Paging Continued…

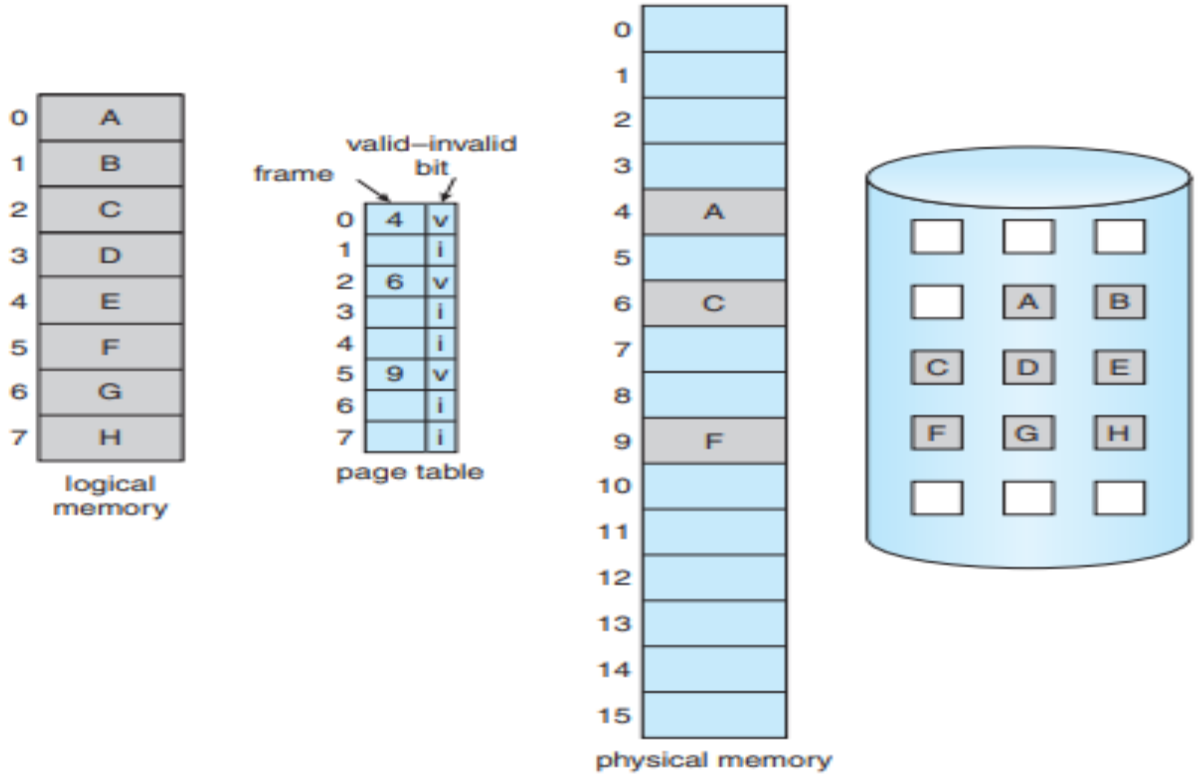
- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- A bit is needed to indicate whether the page is in main memory or not

Paging Continued…



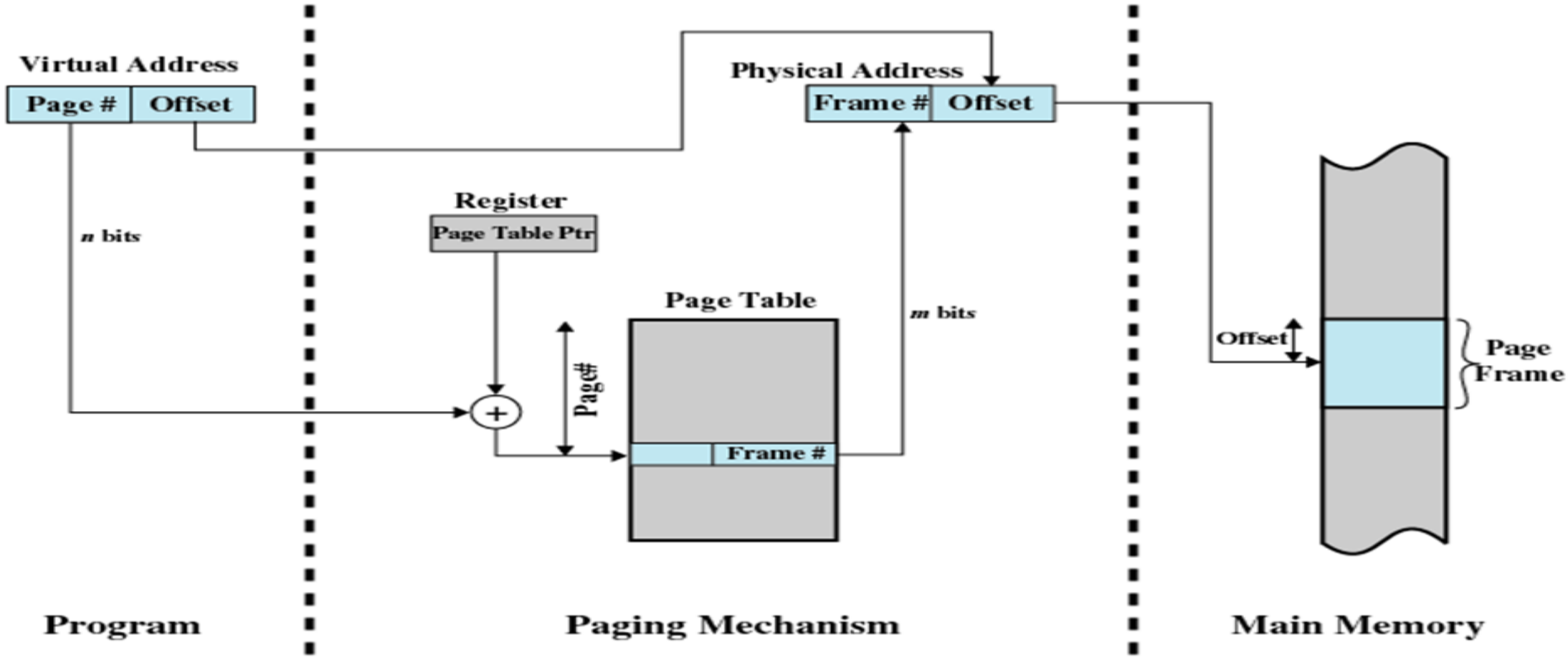
P /V:Present or Valid : Is the page in memory
M : Modify Bit
Other control bits : access control bits- read, write, exec

Page table when some pages are not in main memory.



Modify Bit in Page Table

- Modify bit is needed to indicate if the page has been altered since it was last loaded into main memory
- If no change has been made, the page does not have to be written to the disk when it needs to be swapped out



Address Translation in a Paging System

Differences between Logical and Physical addresses

Basis for Comparison	Logical Address	Physical Address
Basic	It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Address Space	Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
Visibility	The user can view the logical address of a program.	The user can never view physical address of program
Access	The user uses the logical address to access the physical address.	The user can not directly access physical address.
Generation	The Logical Address is generated by the CPU	Physical Address is Computed by MMU

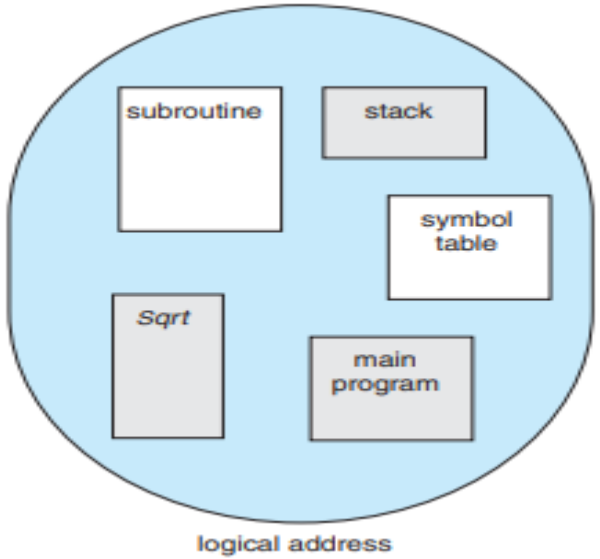
Page Tables

- The entire page table may take up too much main memory
- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

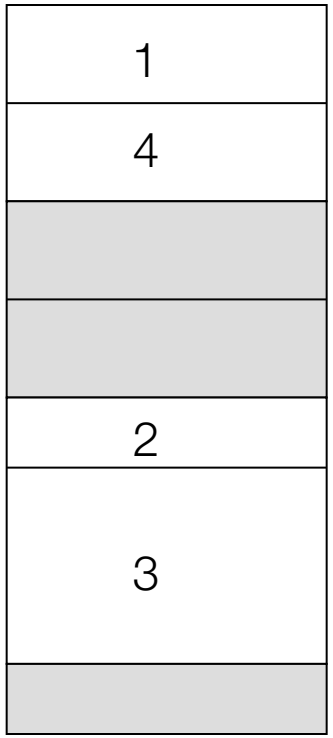
Segmentation

- A process is divided into number of segments that need not be of equal size
- Since segments are not equal, segmentation is similar to dynamic partitioning
- When a process is brought in, all of its segments are loaded into available regions of memory, and a segment table is set up.
- Addressing consist of two parts - a segment number and an offset

Logical View of Segmentation

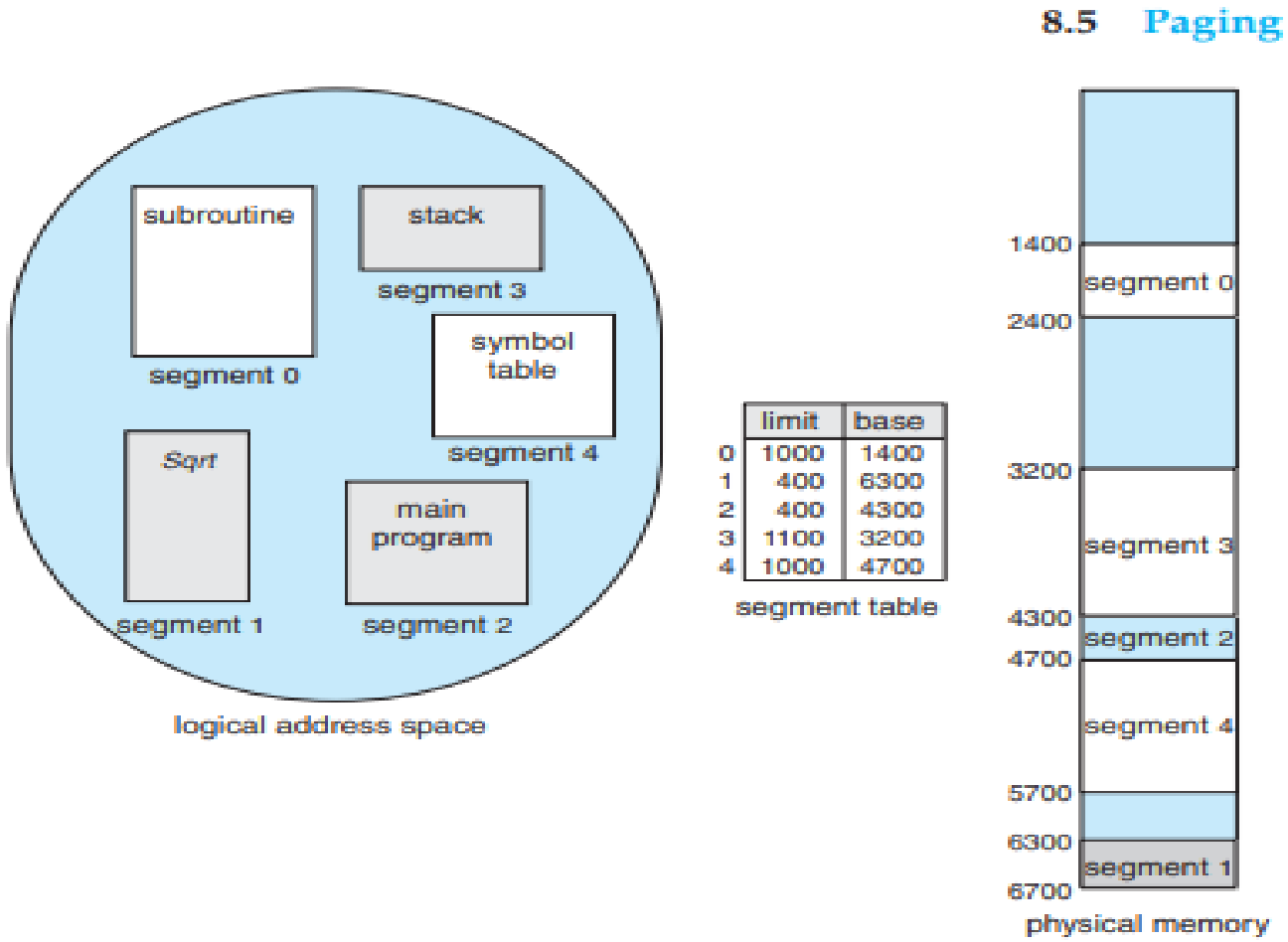


user space

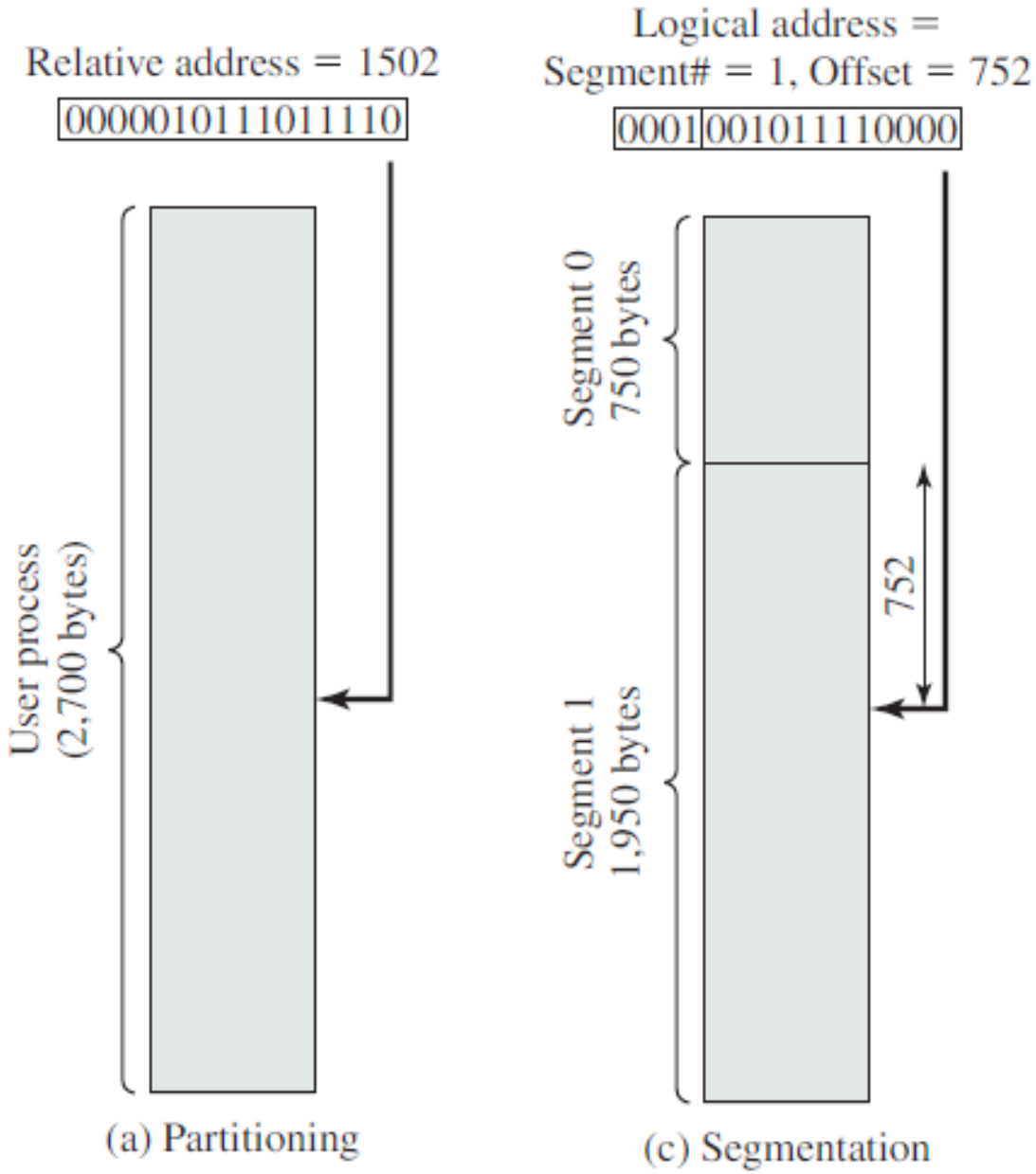


physical memory space

Segmentation Example



Logical Address in Segmentation



Differences between paging and segmentation

Basis for Comparison	Paging	Segmentation
Basic	A page is of fixed block size.	A segment is of variable size.
Fragmentation	Paging may lead to internal fragmentation.	Segmentation may lead to external fragmentation.
Address	The user specified address is divided by CPU into a page number and offset.	The user specifies each address by two quantities a segment number and the offset (Segment limit).
Size	The hardware decides the page size.	The segment size is specified by the user.
Table	Paging involves a page table that contains base address of each page.	Segmentation involves the segment table that contains segment number and offset (segment length).

MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Virtual Memory

Virtual Memory

- Memory references are dynamically translated into physical addresses at run time
  - A process may be swapped in and out of main memory such that it occupies different regions
- A process may be broken up into pieces that do not need to located contiguously in main memory
- All pieces of a process do not need to be loaded in main memory during execution

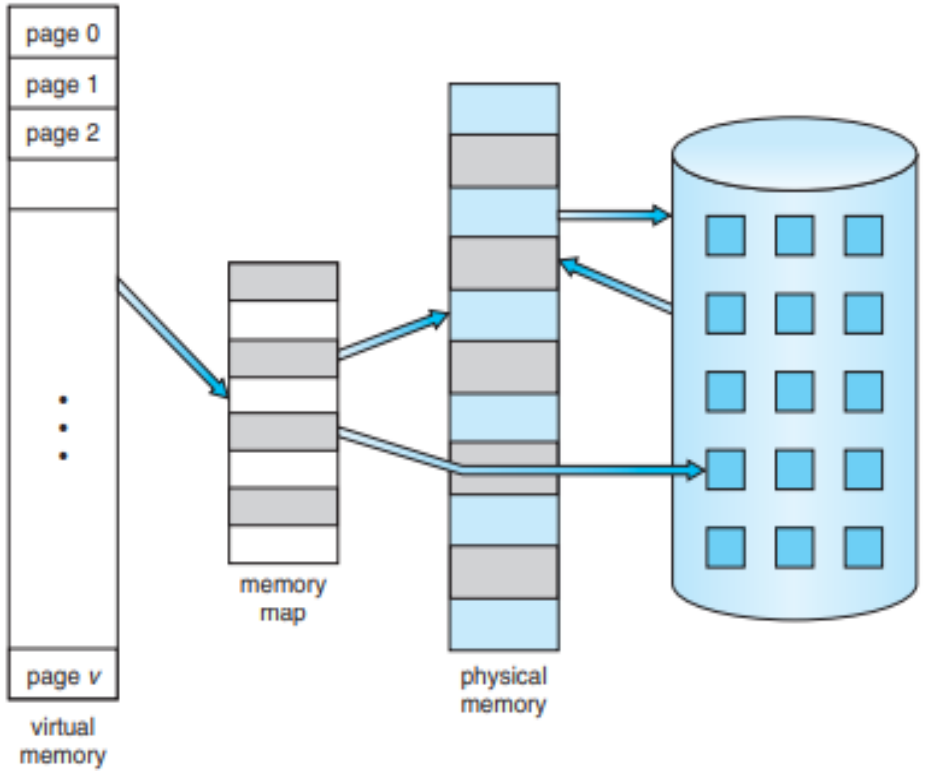


Diagram showing virtual memory that is larger than physical memory

Virtual memory involves the separation of logical memory as perceived by users from physical memory.

This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available

Execution of a Program

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state

Continued…

- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Advantages of Breaking up a Process

- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory

- **Virtual memory** is a memory management technique that separates **logical memory** (as perceived by users and programs) from **physical memory** (actual RAM).
- This separation enables the system to provide programmers with the illusion of a very large memory space, even though only a smaller physical memory is actually available.
- By doing so, virtual memory simplifies programming, since developers no longer need to manage memory limitations or worry about how much physical memory is present.

Types of Memory

- Real memory
  - Main memory
- Virtual memory
  - Memory on disk
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory
  - The two techniques used to implement the concept of Virtual Memory are **Paging** and **Segmentation**

Thrashing

- Swapping out a piece of a process just before that piece is needed
- The processor spends most of its time swapping pieces rather than executing user instructions

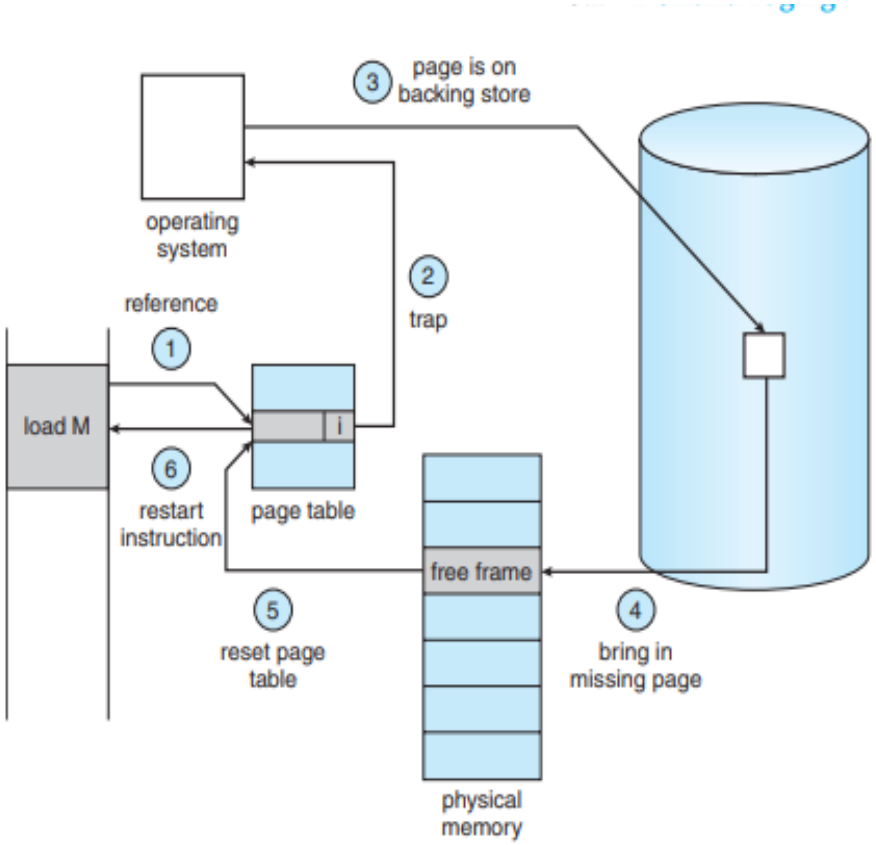
Fetch Policy

Determines when a page should be brought into memory. Two alternatives:

1. **Demand paging** only brings pages into main memory when a reference is made to a location on the page
   - Many page faults when process first started

2. **Prepaging** brings in more pages than needed
   - More efficient to bring in number of contiguous pages at one time rather than bringing one at a time
   - Ineffective as most of the extra pages that are brought in are not referenced.

Steps in handling a page fault



1.check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2. If the reference was invalid, terminate the process. If it was valid but have not yet brought in that page, then page it in.

3. Find a free frame (by taking one from the free-frame list, for example).

4. Schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory

Replacement Policy

- A page fault occurs when a program attempts to access data or code that is in its address space, but is not available in the main memory
- Frame Locking
  - Associate a lock bit with each frame
  - If frame is locked, it may not be replaced
  - Example:
    - Kernel of the operating system
    - Key Control structures

Locality of Reference

Several studies suggest a strong tendency of programs to favor subsets of their address spaces during execution

This phenomenon is known as locality of reference & there are 2 types

1. Spatial locality: Is the tendency for a program to reference clustered locations in preference to randomly distributed locations e.g. sequential processing of arrays.

It suggests that once an item is referenced, there's high probability that it or its neighboring items are going to be referenced in the near future

2. Temporal locality: Is the tendency for a program to reference the same location or a cluster several times during brief intervals of time e.g. loops

- When programs execute, they do **not access memory uniformly**. Instead, they show a strong tendency to use only certain parts of their address space at any given time.

  .

  This behavior is called **locality of reference**, and it is the reason why techniques like **caching** and **virtual memory** work effectively.

Locality of Reference

Research suggests that executing program moves from one locality to another in the course of its execution

Locality of reference suggests that a significant portion of memory references of the executing program may be made to a subset of its pages

There's increased probability that recently referenced pages, pages in the same locality are going to be referenced in the near future

These findings are utilized in implementing page replacement policies

Basic Page Replacement Algorithms

Deals with selection of a page in main memory to be replaced when a new page must be brought in

1. **First In First Out (FIFO)**
2. **Optimal Policy**
3. **Least Recently Used (LRU)**

First-in, first-out (FIFO)

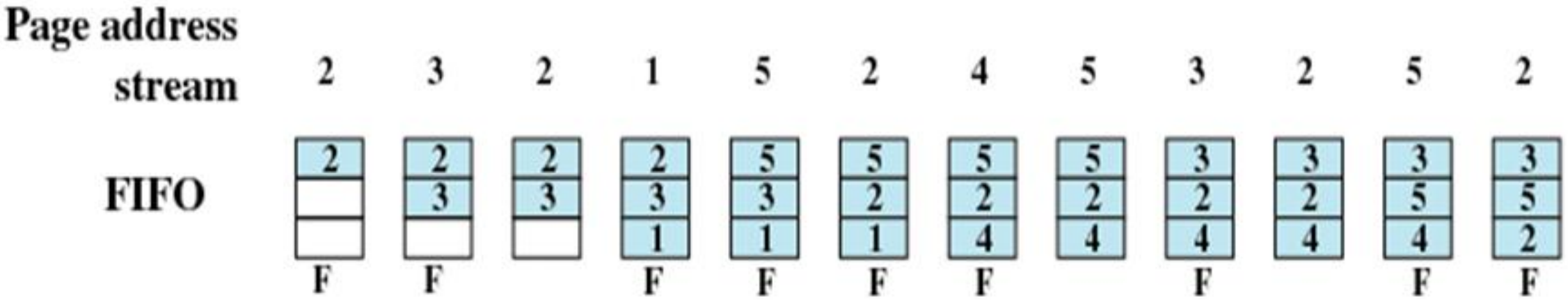
- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
- Simplest replacement policy to implement
- Page that has been in memory the longest is replaced
- These pages may be needed again very soon

Example

- We are considering following page reference string and three frames

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2

Example



Belady's Anomaly

The reference string is

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Find the number of page faults for 3 frames & 4 frames

Belady's Anomaly

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

FIFO Replacement – Belady's Anomaly

- more frames ⇒ less page faults

Belady's Anomaly

- This most unexpected result is known as Belady's Anomaly.
- For FIFO page-replacement algorithm, the page-fault rate may *increase* as the number of allocated frames increases.
- It is expected that giving more memory to a process would improve its performance.
- In some early research, investigators noticed that this assumption was not always true.
- **Discovery of Belady's Anomaly led to the discovery of Optimal Page Replacement algorithm which never suffers from Belady's Anomaly**

Optimal policy

- Selects for replacement that page for which the time to the next reference is the longest
- Impossible to have perfect knowledge of future events

Example

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
	F	F		F	F		F			F		

Least Recently Used (LRU)

- Replaces the page that has not been referenced for the longest time
- Each page could be tagged with the time of last reference. This would require a great deal of overhead

Example

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
	F	F		F	F		F		F	F		

Example

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames

4 frames

Example

Reference string: 5,2,3,2,7,1,3,4,5,1. Assumes there are 3 frames in main memory. Show the page replacement by FIFO and Optimal algorithms and calculate the number of page hits, page faults, hit ratio and miss ratio.

Miss/ fault ratio= Page Fault/total number of pages
Hit Ratio= Page Hit/total number of pages

Answer-
1 Using FIFO algorithm- Page Hit=3 Page Fault=7

2. Using Optimal algorithm- Page Hit=4 Page Fault=6

I/O and File Management

I/O Hardware

I/O Management

- I/O Devices
- I/O Buffering

Categories of I/O Devices

- **Three Categories:**

1. **Human readable**
   - Devices used to communicate with the user
     - Video display, Keyboard, Mouse, Printer, etc

2. **Machine readable**
   - Used to communicate with electronic equipment
     - Disk drives, Sensors, Controllers, Actuators, etc

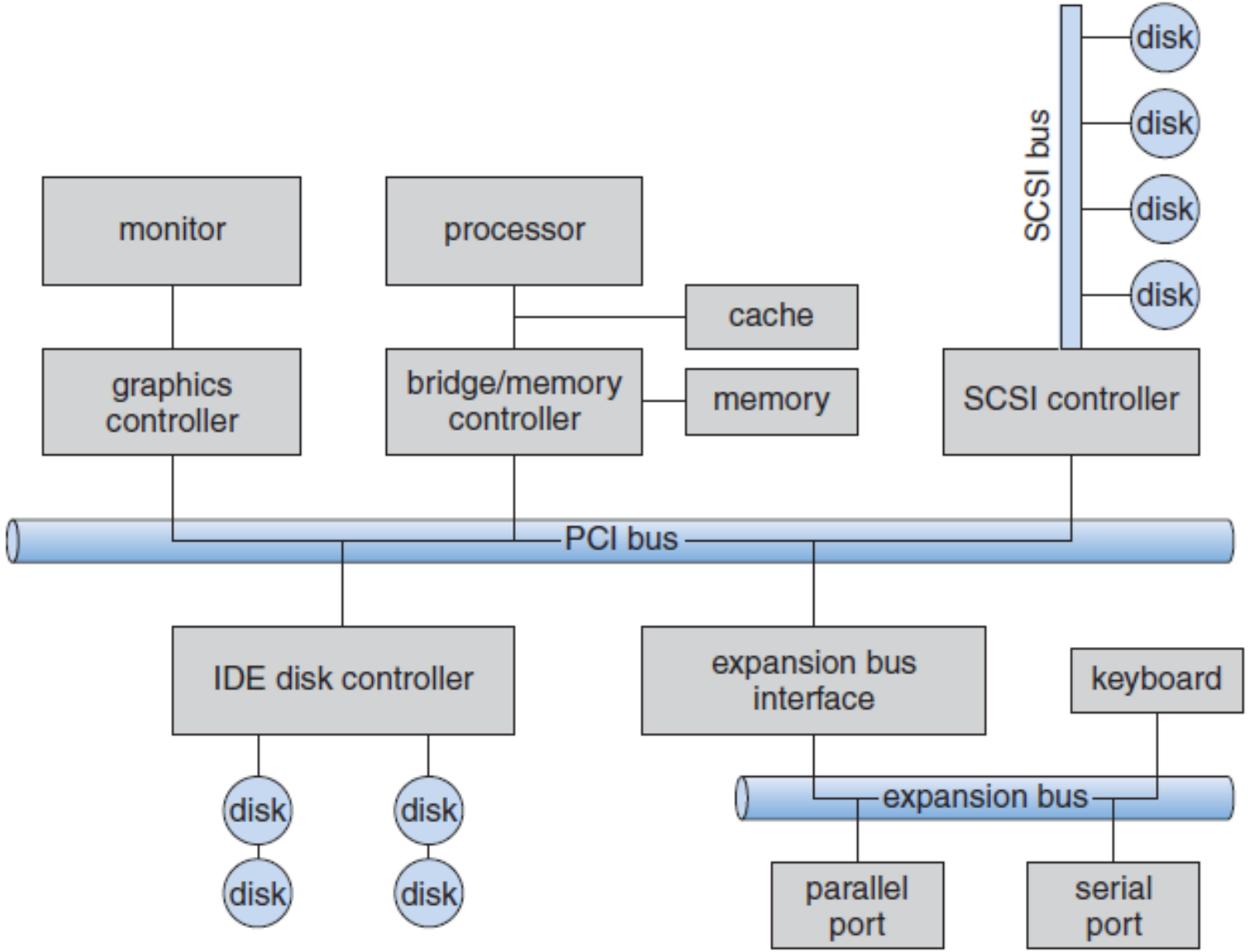
3. **Communications**
   - Used to communicate with remote devices
     - Modems, Digital line drivers, etc

Differences in I/O Devices

- Devices differ in a number of areas
  - Data Rate - Massive difference between the data transfer rates of devices
  - Application
  - Complexity of Control - Complexity of the I/O module that controls the device. Eg. Disk is much more complex as compared to printer
  - Unit of Transfer - Data may be transferred as a stream of bytes or characters (e.g., terminal I/O) or in larger blocks (e.g., disk I/O).
  - Data Representation - Different data encoding schemes are used by different devices
  - Error Conditions - nature of errors differ widely from one device to another

Terminology

- The device communicates with the machine via a connection point called as “**Port**”
- If devices share a common set of wires, the connection is called a “**bus**”
- **PCI (**Peripheral Component Interconnect**) bus** connects the processor–memory subsystem to fast devices, and an **expansion bus** connects relatively slow devices, such as the keyboard and serial and USB ports.
- When device *A* has a cable that plugs into device *B,* and device *B* has a cable that plugs into device *C,* and device *C* plugs into a port on the computer, this arrangement is called a **daisy chain**.



Typical PC Bus Structure

- **PCI – Peripheral Component Interconnect**
  A hardware bus used to connect peripheral devices (like network cards, sound cards, etc.) to a computer's motherboard.
- A **SCSI (**Small Computer System Interface**) controller** is a hardware device (or an interface card) that manages the connection and communication between the computer's **CPU** and **SCSI devices** such as:
- Hard drives
- CD/DVD drives
- Scanners
- Printers

Bridge/Memory Controller

Acts as a **bridge** between the **processor** and **memory** or **PCI bus**.

Controls data flow between the CPU, memory, and other peripheral devices.

Cache and Memory

Cache: Provides high-speed storage for frequently accessed data, reducing CPU waiting time.

Main Memory (RAM): Stores currently executing programs and data.

System Buses

PCI Bus (Peripheral Component Interconnect)

The **main system bus** connecting the CPU to high-speed components.
Allows multiple devices to communicate with the processor efficiently.

Devices connected to the PCI bus include:

Graphics Controller (for display/monitor)
IDE (Integrated Drive Electronics**) Disk Controller** (used for internal storage devices e.g., HDDs, CD-ROMs)
SCSI Controller (for high-speed devices)
Expansion Bus Interface (for connecting slower peripherals)

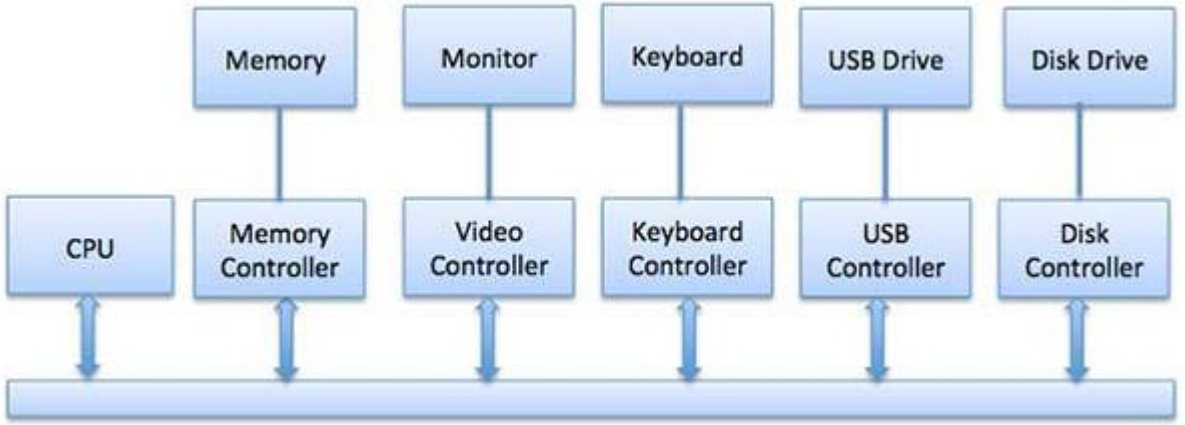
MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Device Controller

- A controller is a collection of electronics that can operate a port, a bus, or a device
- A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port.
- SCSI bus controller is often implemented as a separate circuit board that plugs into the computer.
- The controller has one or more registers for data and control signals. The processor communicates with the controller by reading and writing bit patterns in these registers.

Device Controller

- Device drivers are software modules that can be plugged into an OS to handle a particular device.
- The Device Controller works like an interface between a device and a device driver. There is always a device controller and a device driver for each device to communicate with the Operating Systems.



MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

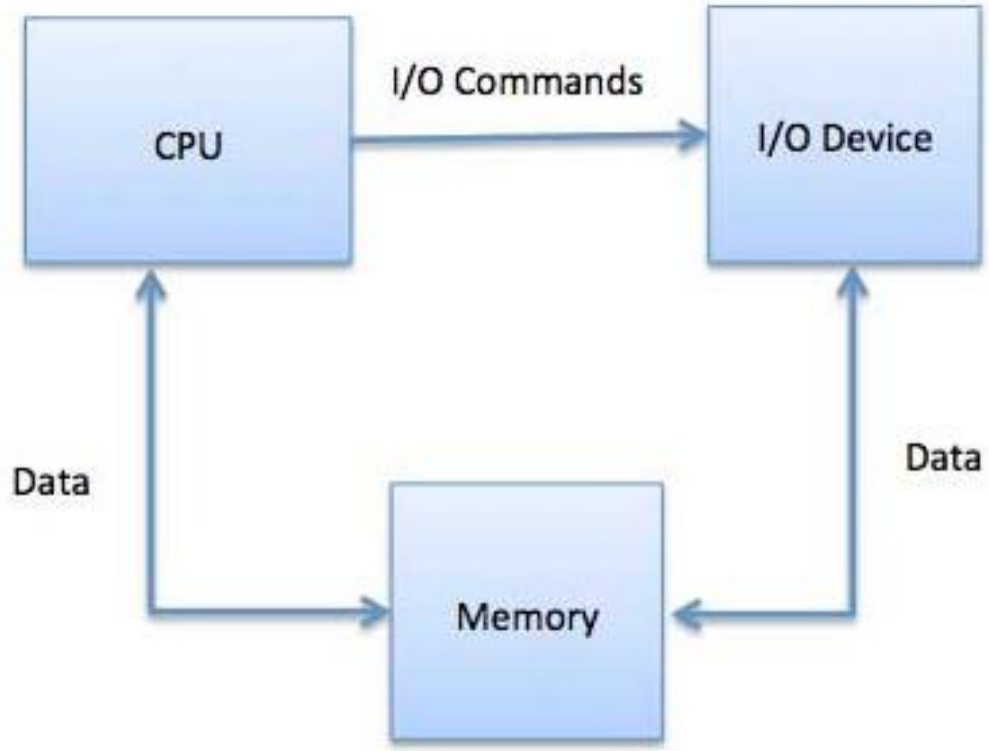
Communication to I/O Devices

- The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.
  - Special Instruction I/O
  - Memory-mapped I/O
  - Direct Memory Access (DMA)

- **Special Instruction I/O:** This uses CPU instructions (like IN and OUT) that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

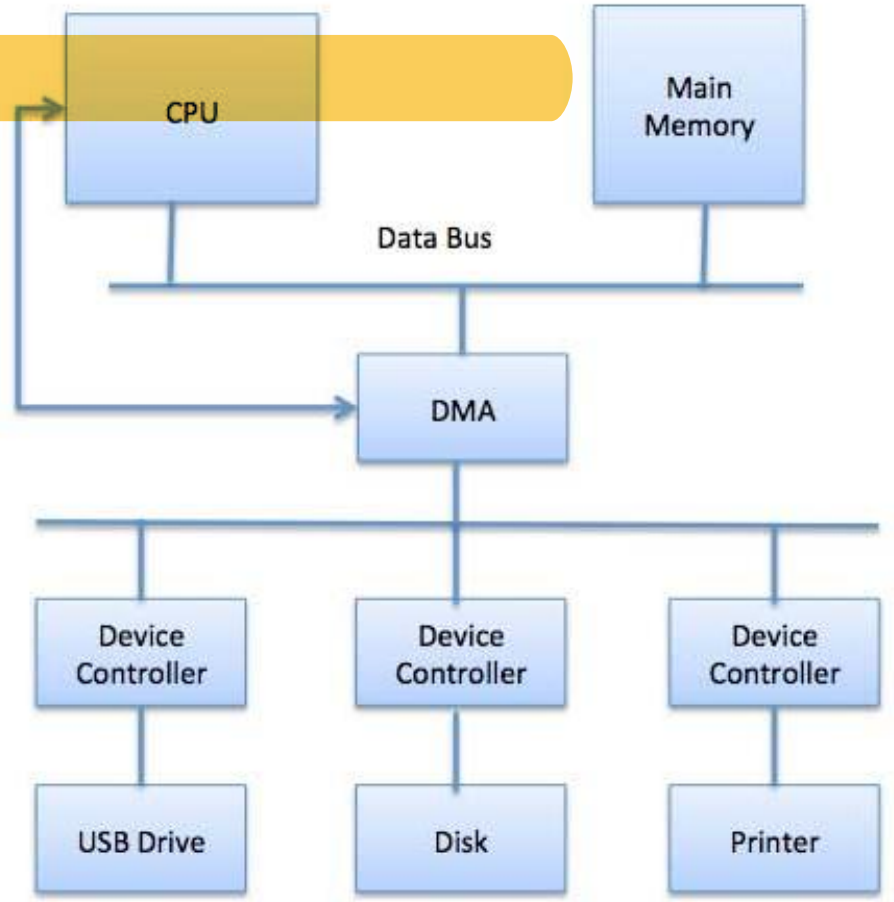
Memory-mapped I/O

- When using memory-mapped I/O, the same address space is shared by memory and I/O devices.
- The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.
- OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU.
- Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

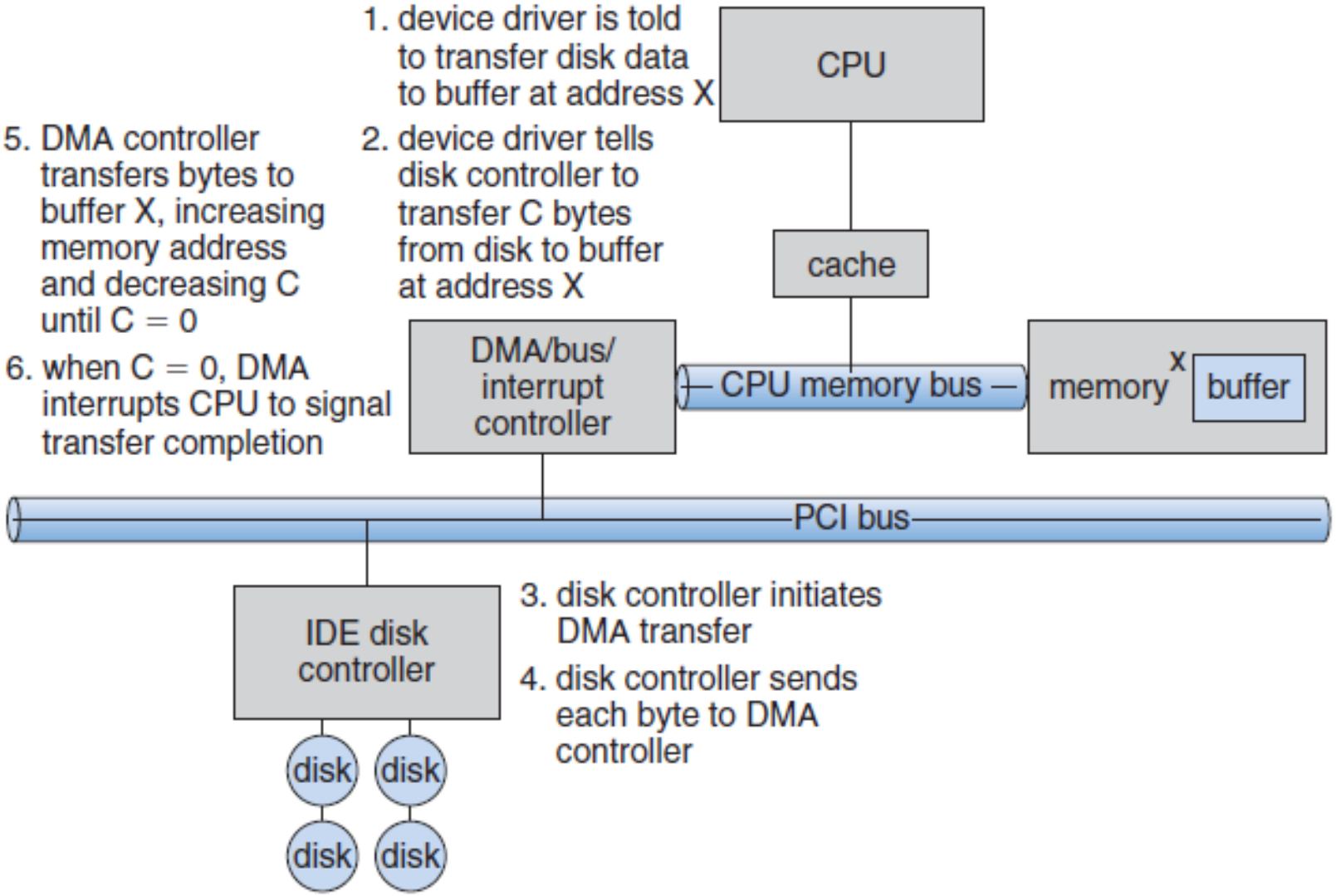


Direct Memory Access (DMA)

- CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module itself controls exchange of data between main memory and the I/O device.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.
- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus.
- The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



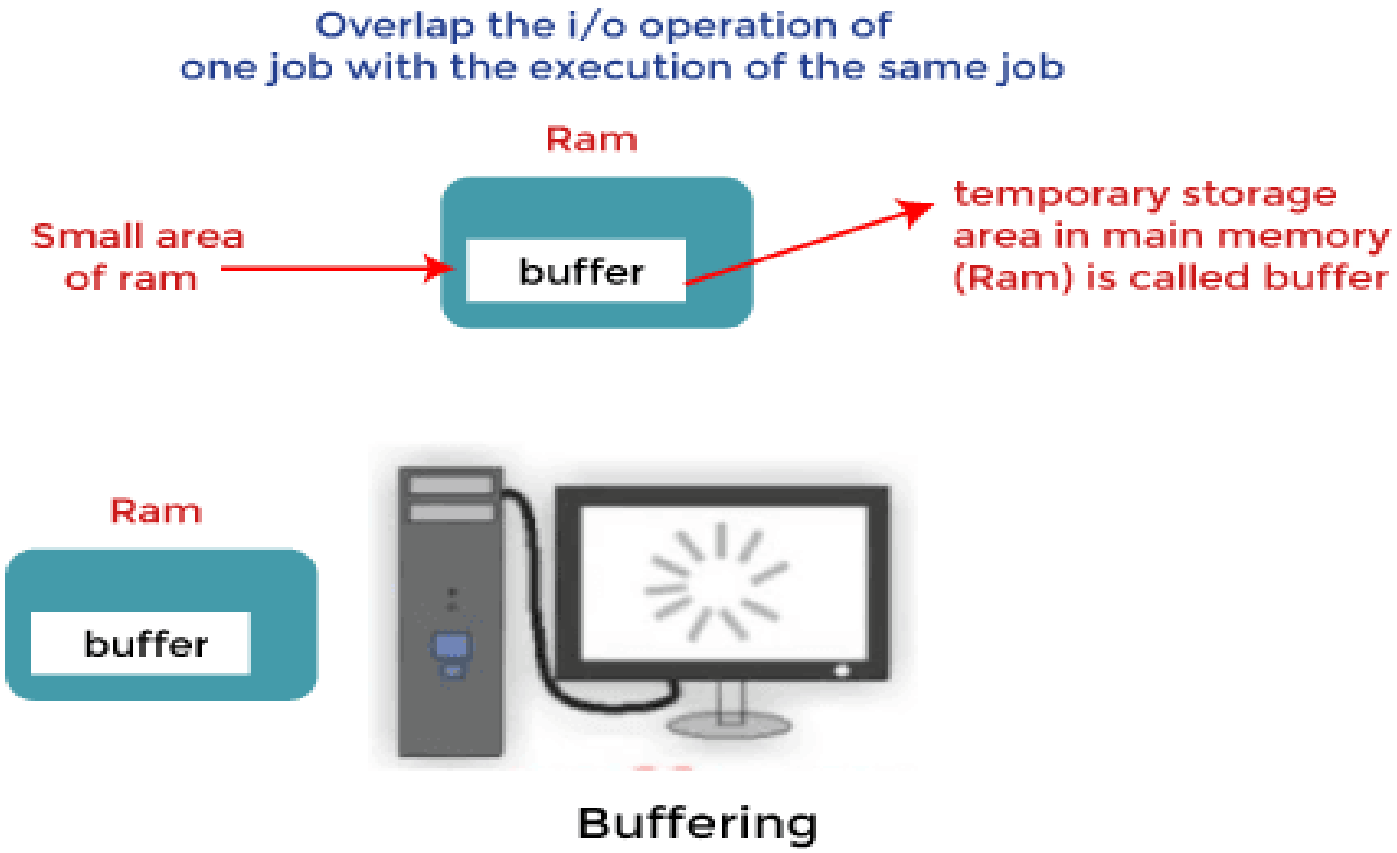
Steps in DMA Transfer



I/O Buffering

- Processes must wait for I/O to complete before proceeding
- It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made.
- **Block-oriented Buffering**
- Information is stored in fixed sized blocks
- Transfers are made a block at a time Eg. Used for disks
- **Stream-Oriented Buffering**
- Transfer information as a stream of bytes
- Used for terminals, printers, communication ports, mouse, etc.

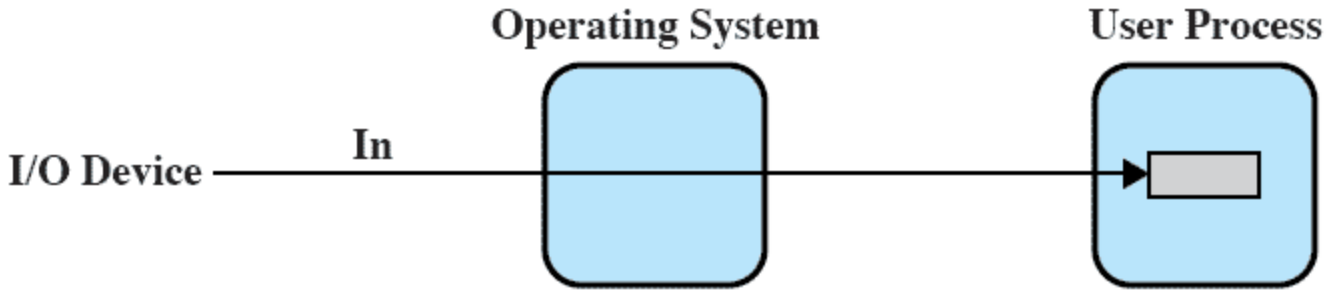
I/O Buffering



Buffering

No Buffer

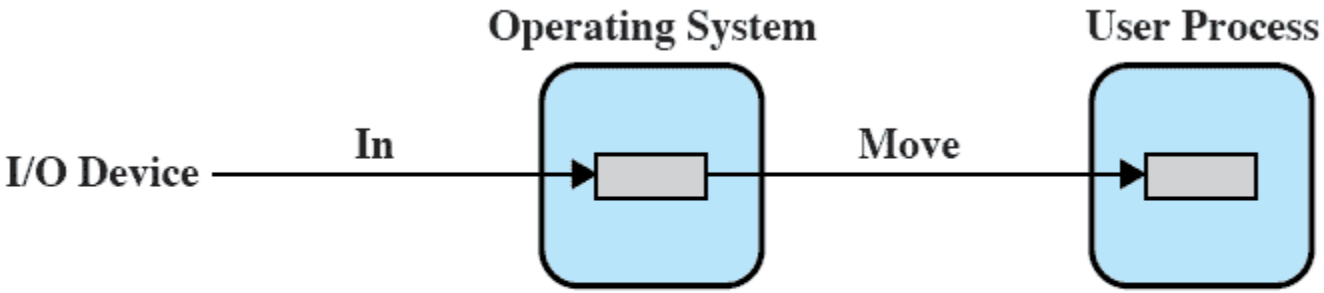
- Without a buffer, the OS directly access the device as and when it needs



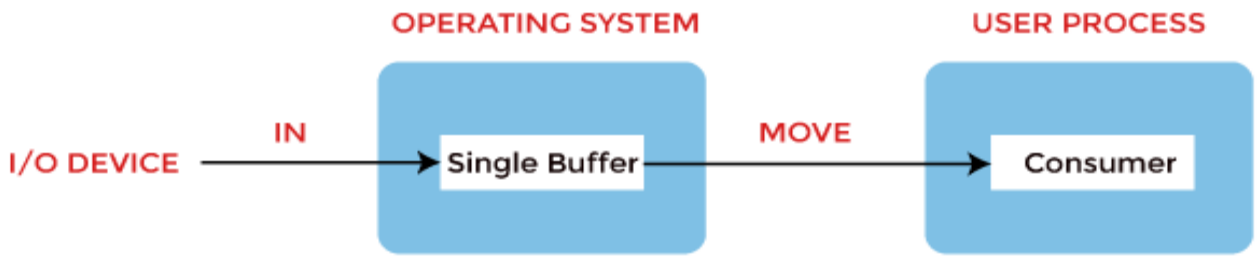
(a) No buffering

Single Buffer

- Operating system assigns a buffer in main memory for an I/O request



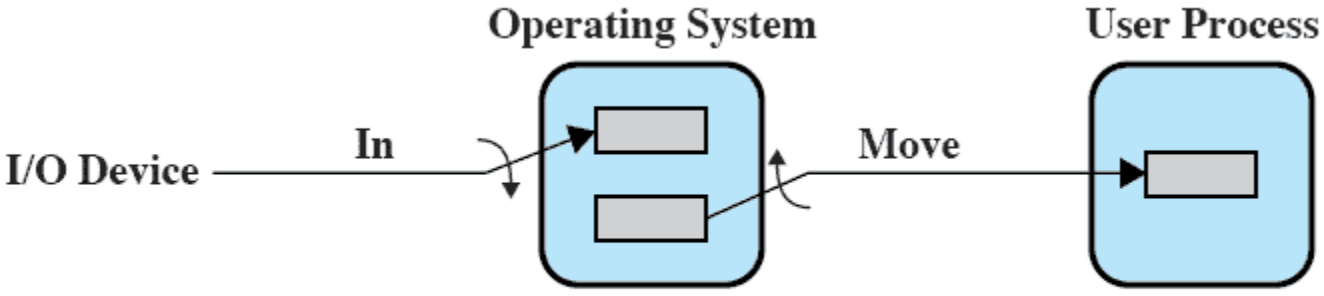
(b) Single buffering



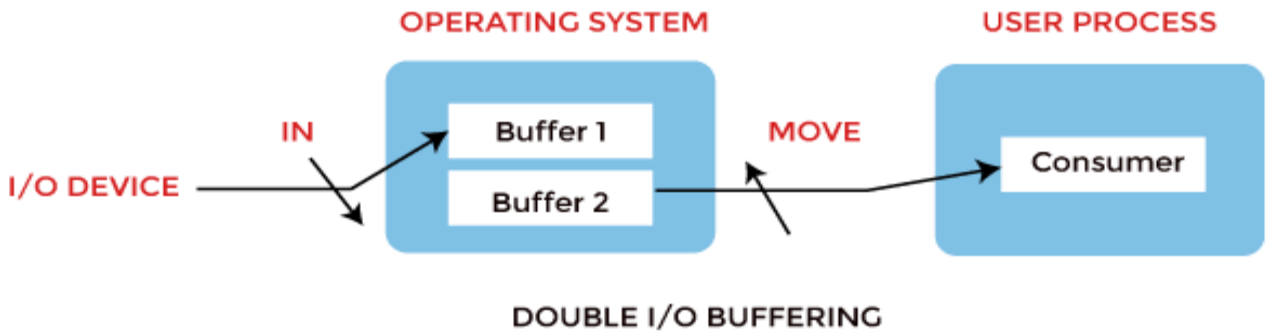
SINGLE I/O BUFFERING

Double Buffer

- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



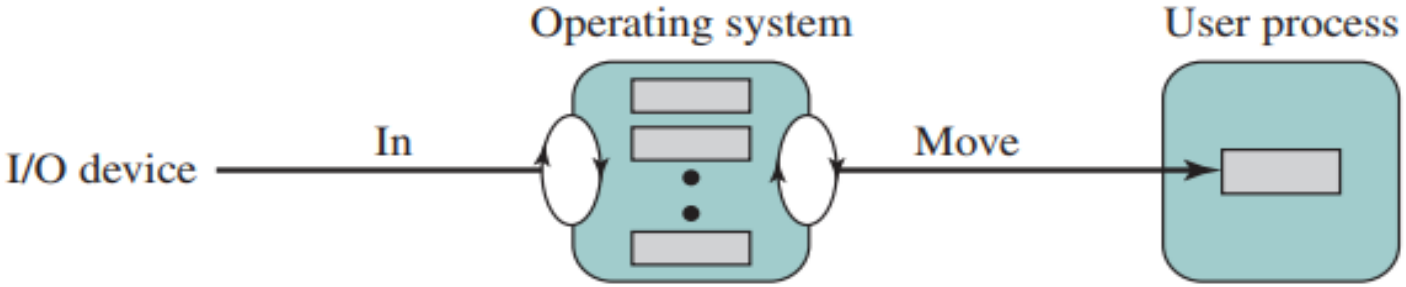
(c) Double buffering



DOUBLE I/O BUFFERING

Circular Buffer

- A double-buffer scheme should smooth out the flow of data between an I/O device and a process.
- If the performance of a particular process is the focus of our concern, then we would like for the I/O operation to be able to keep up with the process.
- Double buffering may be inadequate if the process performs rapid bursts of I/O
- The problem can often be alleviated by using more than two buffers. **When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer** with each individual buffer being one unit in the circular buffer.



Type	No. of Buffers	CPU–I/O Overlap	Efficiency	Complexity
Single Buffering	1	Low	Low	Simple
Double Buffering	2	Medium	Better	Moderate
Circular Buffering	3 or more	High	High	Complex

Buffer Limitations

- Buffering smoothens out peaks in I/O demand
- But with enough demand eventually all buffers become full and their advantage is lost
- Buffering can increase the efficiency of the OS and the performance of individual processes.

File Management

Overview

- Files are the central element to most applications
- Desirable properties of files:
  - Long-term existence
  - Sharable between processes
  - Structure

Terms in common use when discussing files

1. Fields

- Basic element of data
- Contains a single value
- Characterized by its length and data type

2. Records

- Collection of related fields
- Treated as a unit

3. File

- Have file names
- Is a collection of similar records
- Treated as a single entity
- May implement access control mechanisms

4. Database

- Collection of related data
- Relationships exist among elements
- Consists of one or more files

File System

- The File System is one of the most important part of the OS to a user
- Concerned with secondary storage
- File systems also provide functions which can be performed on files:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write

File Organization

- Refers to the logical structuring of records
- Determined by the ***way*** in which files are accessed
- Important criteria while choosing a file organization:
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability

File Organization

File Organization Types

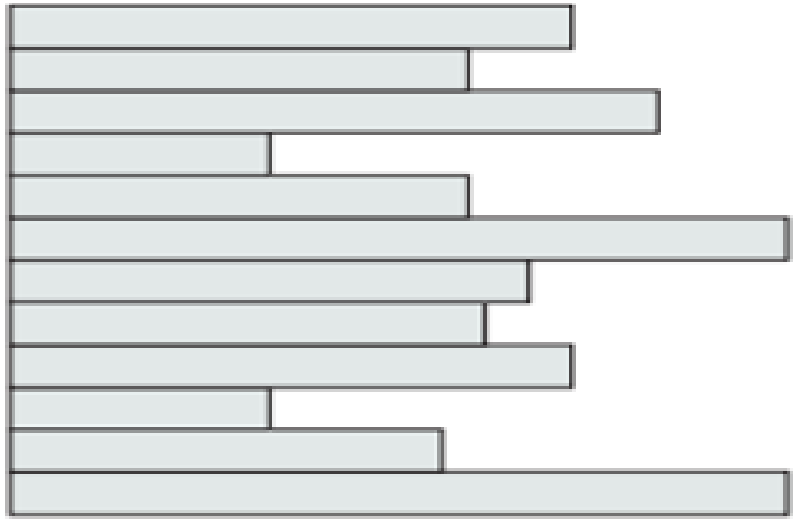
Many exist, but usually variations of:

- Pile
- Sequential file
- Indexed file
- Direct or hashed file

File Organization

Pile

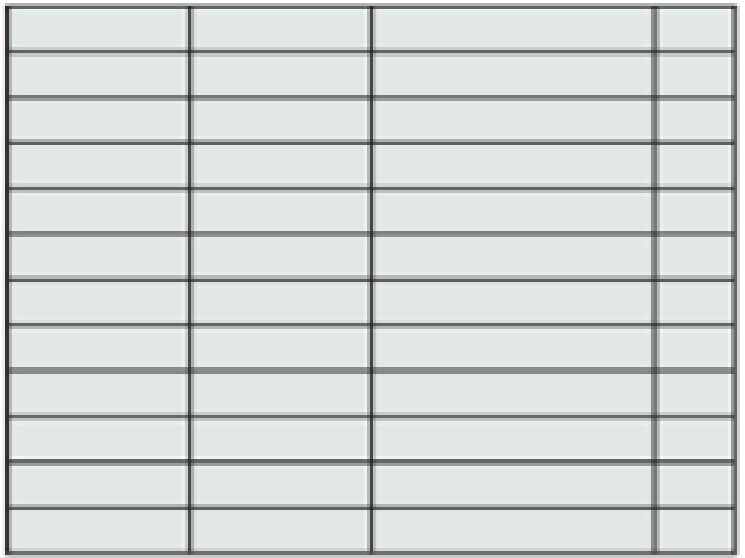
- Data are collected in the order they arrive
- No structure
- Purpose is to accumulate a mass of data and save it
- Records may have different fields
- Record access is by exhaustive search



Variable-length records
Variable set of fields
Chronological order

(a) Pile file

- **Sequential File**
- Fixed format used for records
- Records are the same length
- Key field
  - Uniquely identifies the record
  - Records are stored in key sequence



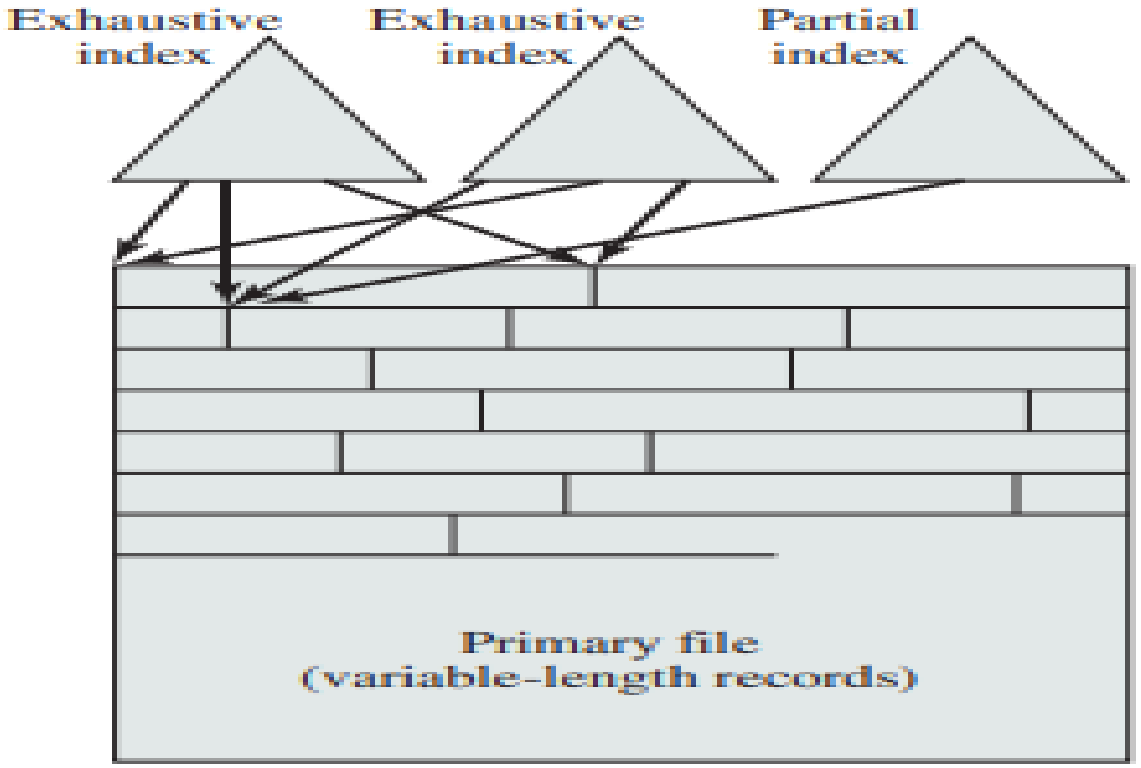
Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential file

File Organization

Indexed File

- Maintains the key characteristic of the sequential file: records are organized in sequence based on a key field
- An index is added to the file to support random access
- May contain an exhaustive index that contains one entry for every record in the main file
- May contain a partial index
- When a new record is added to the main file, all of the index files must be updated



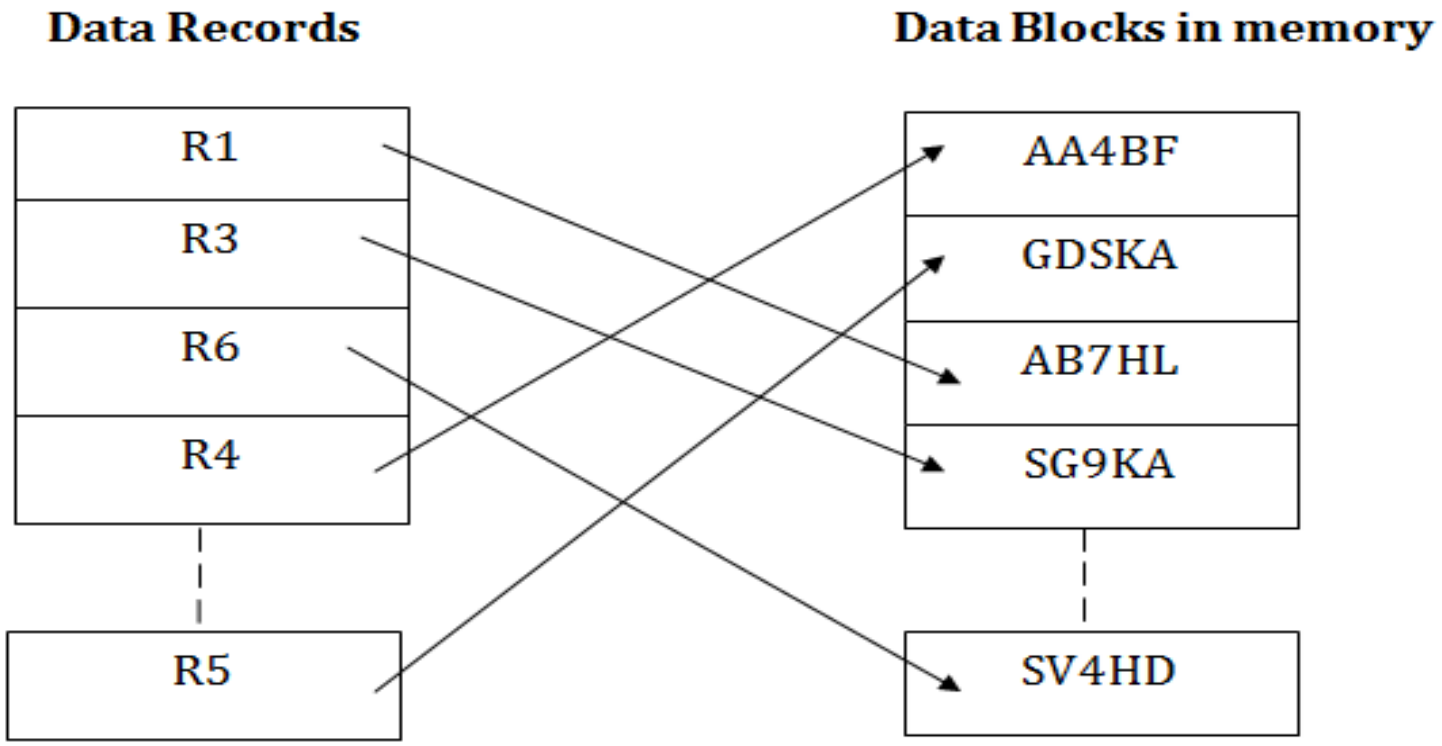
(d) Indexed file

File Organization

Direct or Hash File

- Directly access a block at a known address
- Key field required for each record
- But there is no concept of sequential ordering
- Makes use of hashing on the key value

Direct or Hash File



File Directories

- Contains information about files
  - Attributes
  - Location
  - Ownership
- Provides mapping between file names and the files themselves
- A directory system should support a number of operations including:
  - Search
  - Create files
  - Deleting files
  - Listing directory
  - Updating directory

Directory Elements

Basic Information

- **File Name**

  -Name as chosen by creator

  -Must be unique within a specific directory

Address Information

- **Volume** -Indicates device on which file is stored
- **Starting Address**
- **Size Used -**Current size of the file in bytes, words, or blocks
- **Size Allocated -** Maximum size of the file

Access Control Information

- **Owner**

  -Owner has control of this file & able to grant/deny access to other users and to change these privileges.

- **Access Permission**

-Specifies read, write & execute permissions on the file for owner, group & others

$ chmod 755 your-script-name

The chmod command modifies the permissions of a file or directory on a Linux system. ... The numbers 755 assign **read-write-execute permissions to the user owner and read-execute permissions to group owner and others**.

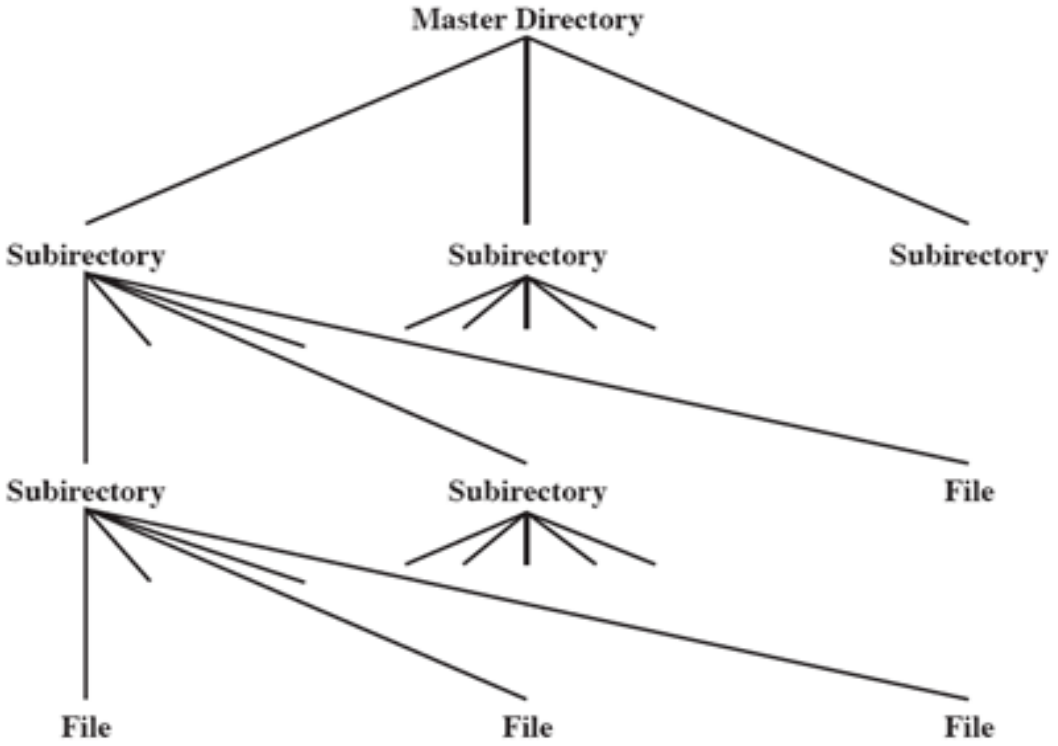
- **The sums of these numbers give combinations of these permissions:**
- 0 = no permissions whatsoever; this person cannot read, write, or execute the file.
- 1 = execute only.
- 2 = write only.
- 3 = write and execute (1+2)
- 4 = read only.
- 5 = read and execute (4+1)
- 6 = read and write (4+2)
- 7 = read and write and execute (4+2+1)

Directory Elements: Usage Information

- Date Created
- Identity of Creator
- Date Last Read Access
- Identity of Last Reader
- Date Last Modified
- Identity of Last Modifier
- Date of Last Backup
- Current Usage

Hierarchical or Tree-Structured Directory

- Master directory with user directories underneath it
- Each user directory may have subdirectories and files as entries
- **Naming**
- Users need to be able to refer to a file by name
- Files need to be named uniquely
- Tree structure allows users to find a file by following the directory path
- Duplicate filenames are possible if they have different pathnames



File Sharing

- In multiuser system, allow files to be shared among users
- Two issues
  - Access rights
  - Management of simultaneous access

-User may lock entire file when it is to be updated

-User may lock the individual records during the update

-Mutual exclusion and deadlock are issues for shared access

User Classes

- Owner - Usually the files creator, has full rights
- User Groups - A set of users identified as a group
- Others