MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Unit IV File & I/O management

Syllabus

File Management: Overview, File Organization and Access, File Directories, File Sharing, Record Blocking.

I/O Management: I/O Devices, Organization of the I/O Functions, I/O Buffering, Disk Scheduling.

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 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

Sequential File

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

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

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

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

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

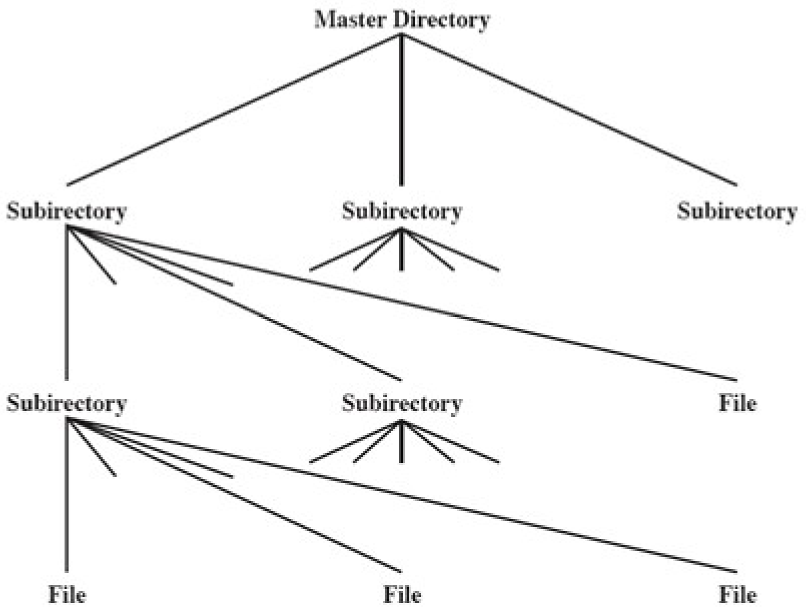
Hierarchical or Tree-Structured Directory

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

I/O Management

- I/O Devices
- I/O Buffering
- Disk Scheduling

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

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

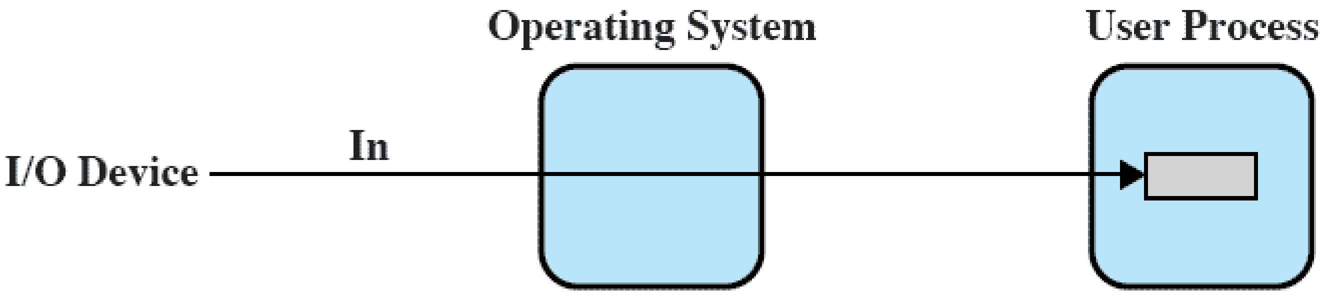
- Information is stored in fixed sized blocks
- Transfers are made a block at a time Eg. Used for disks

Stream-Oriented Buffering

- Transfer information as a stream of bytes
- Used for terminals, printers, communication ports, mouse, etc.

No Buffer

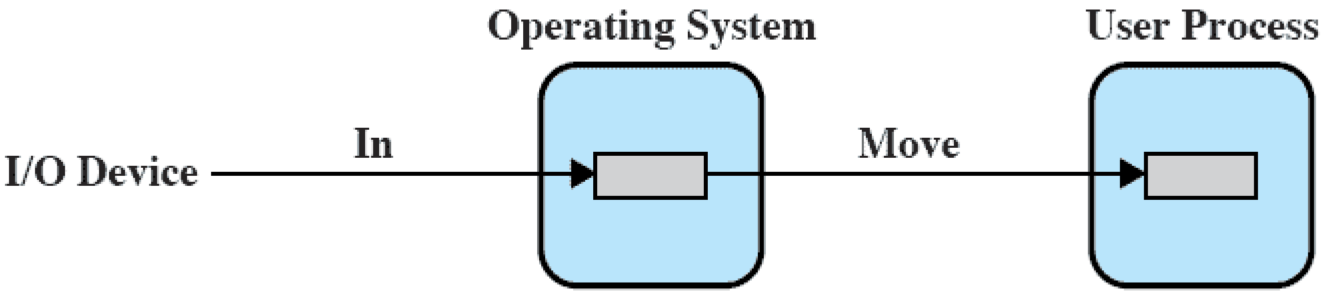
Without a buffer, the OS directly access the device as and when it needs



(a) No buffering

Single Buffer

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

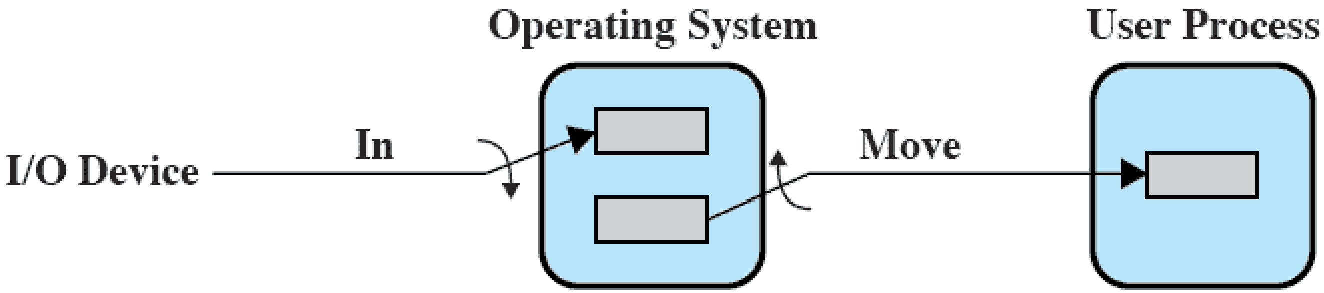


(b) Single buffering

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

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.

Disk Scheduling

- When the disk drive is operating, the disk is rotating at constant speed
- Positioning the Read/Write Head
- Track selection involves moving the head to a specific track
- **Disk Performance Parameters**

  -*Access Time* is the sum of:
  - ***Seek time:*** Time it takes to position the head at the desired track
  - ***Rotational delay*** or ***rotational latency*:** The time its takes for the beginning of the sector to reach the head

  -*Transfer Time* is the time taken to transfer the data.

Disk Scheduling Policies

To compare various schemes, consider a disk head is initially located at track 100.

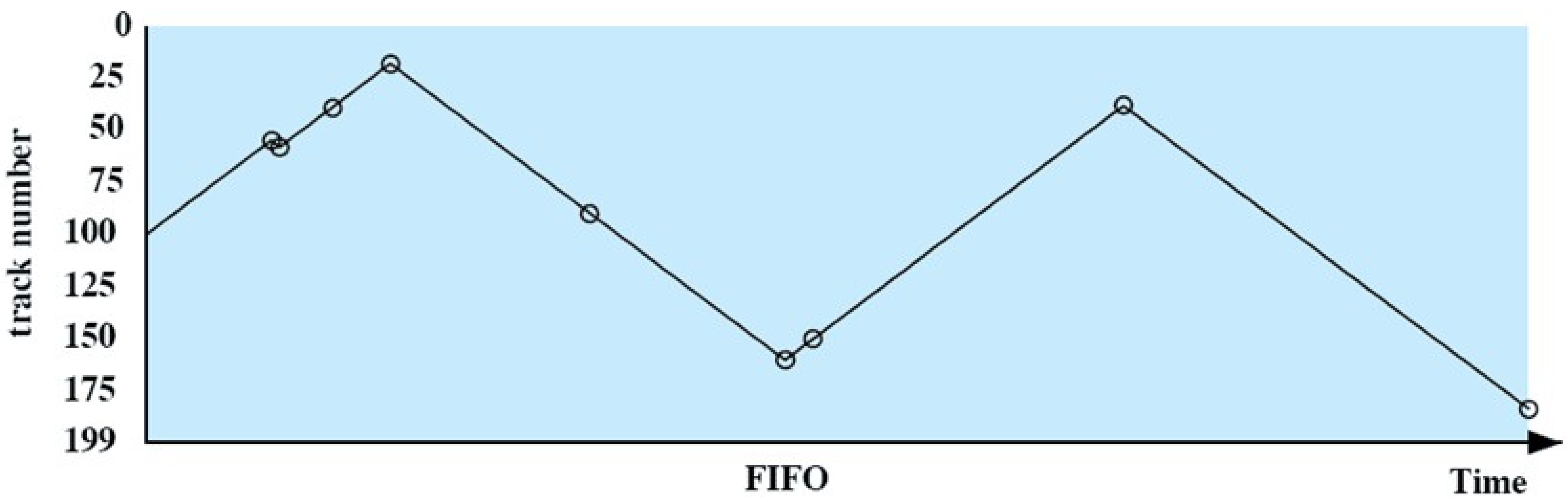
- assume a disk with 200 tracks and that the disk request queue has random requests in it

The requested tracks, in the order received by the disk scheduler, are

- 55, 58, 39, 18, 90, 160, 150, 38, 184.

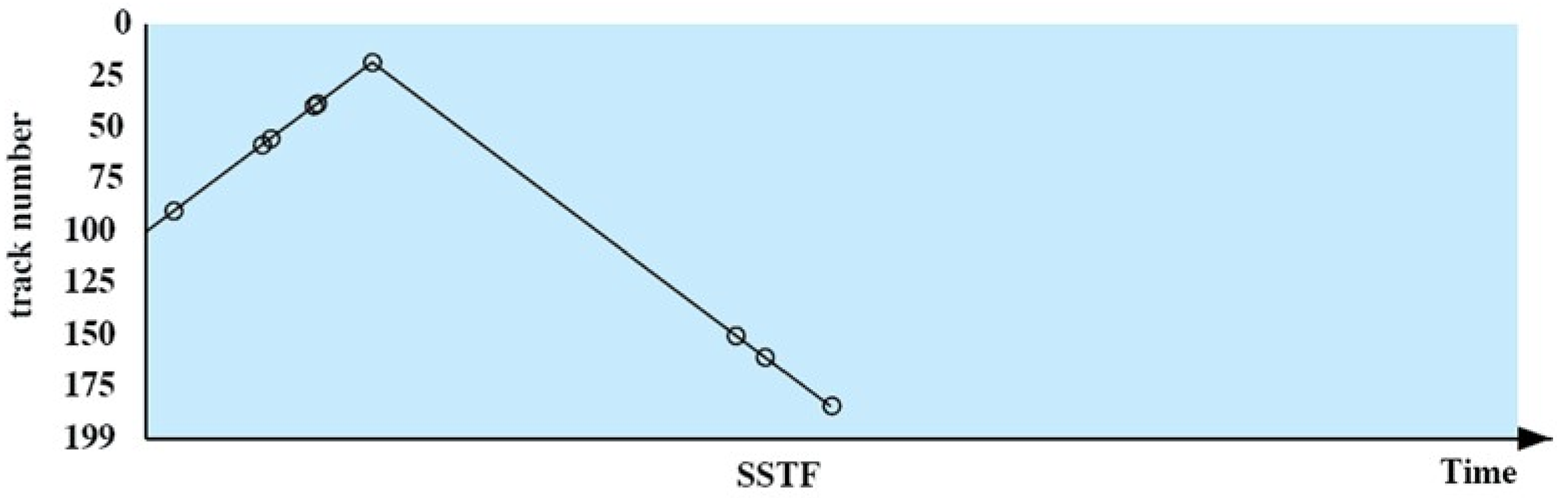
First-in, first-out (FIFO)

- Processes requests sequentially in the order received
- Fair to all processes
- Approaches random scheduling in performance



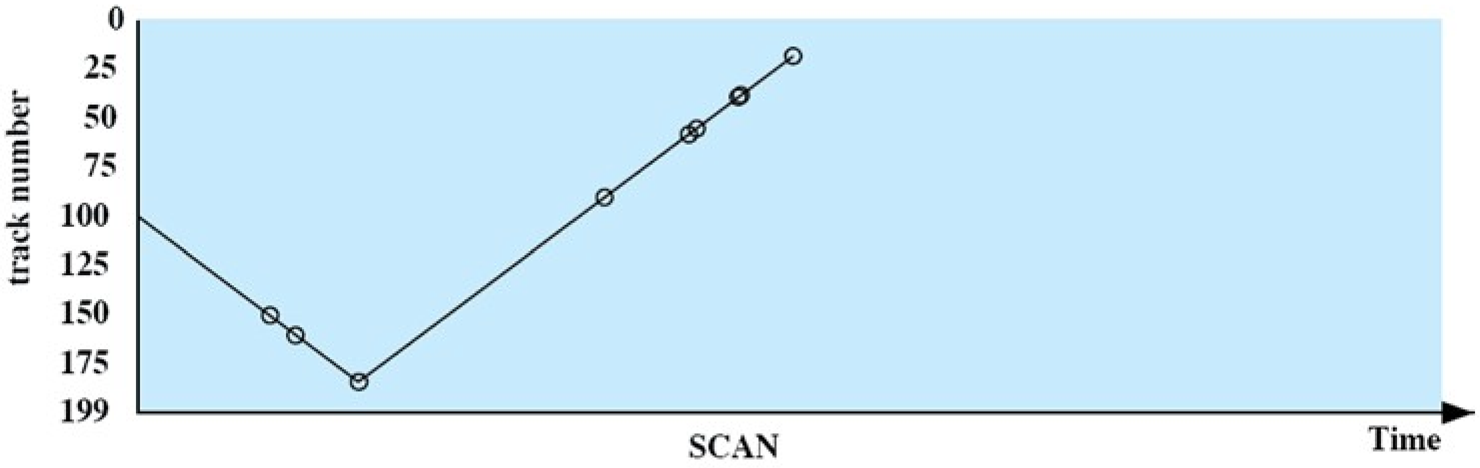
Shortest Service Time First

- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum seek time



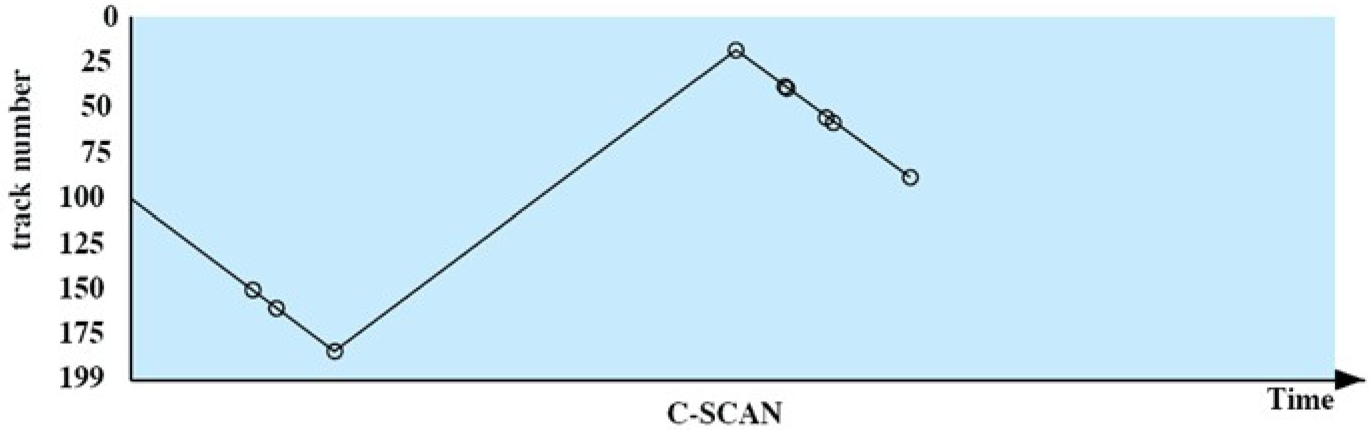
SCAN

- Arm moves in one direction only, processing all outstanding requests until it reaches the last track in that direction & then the direction is reversed



C-SCAN

- Restricts scanning to one direction only
- When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again



Performance Compared

Comparison of Disk Scheduling Algorithms

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

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

Unit IV Memory Management, File Management & I/O Management

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Syllabus Unit IV

Memory Management: Memory Management Requirements, Memory Partitioning: Fixed Partitioning, Dynamic Partitioning; paging, segmentation, virtual memory.

File Management: Overview, File Organization and Access, File Directories, File Sharing, Record Blocking.

I/O Management: I/O Devices, Organization of the I/O Functions, I/O Buffering, Disk Scheduling.

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

Memory Management Requirements

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

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 allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available

Memory Partitioning

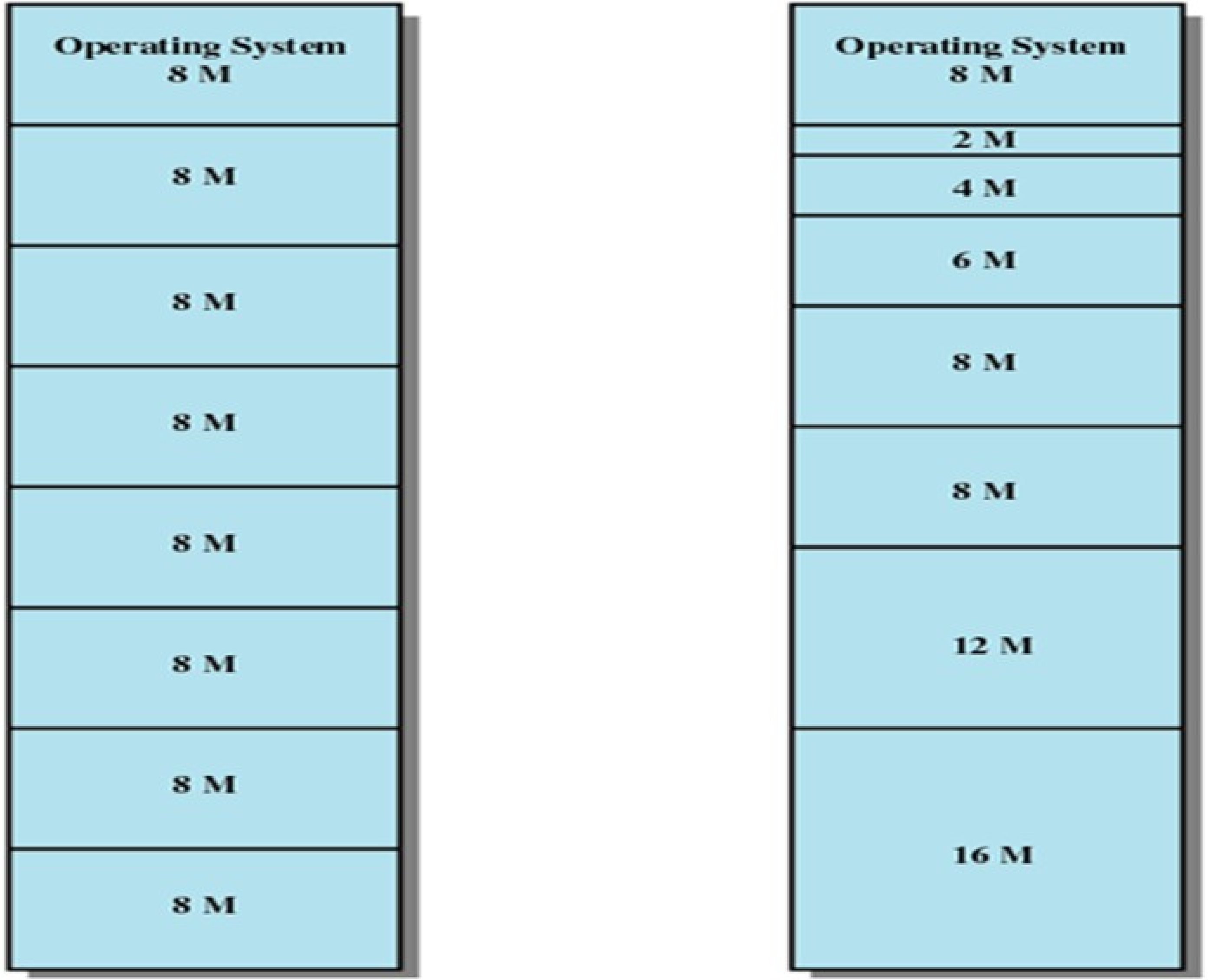
- OS occupies some fixed portion of main memory and that the rest of main memory is available for use by multiple processes.
- The simplest scheme for managing this available memory is to partition it into regions.
- Types-
  - Fixed Partitioning
  - Dynamic Partitioning

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.



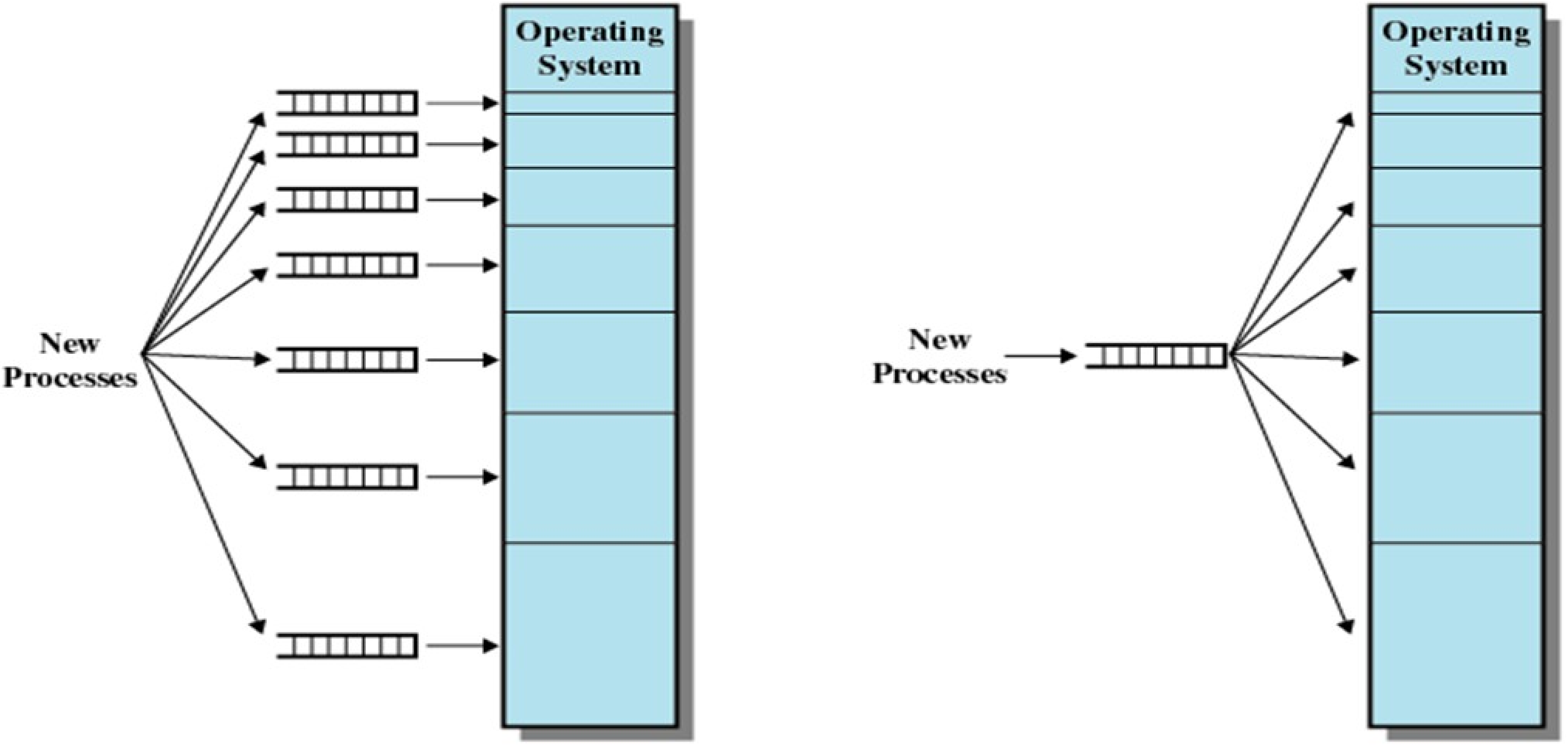
(a) Equal-size partitions

(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.

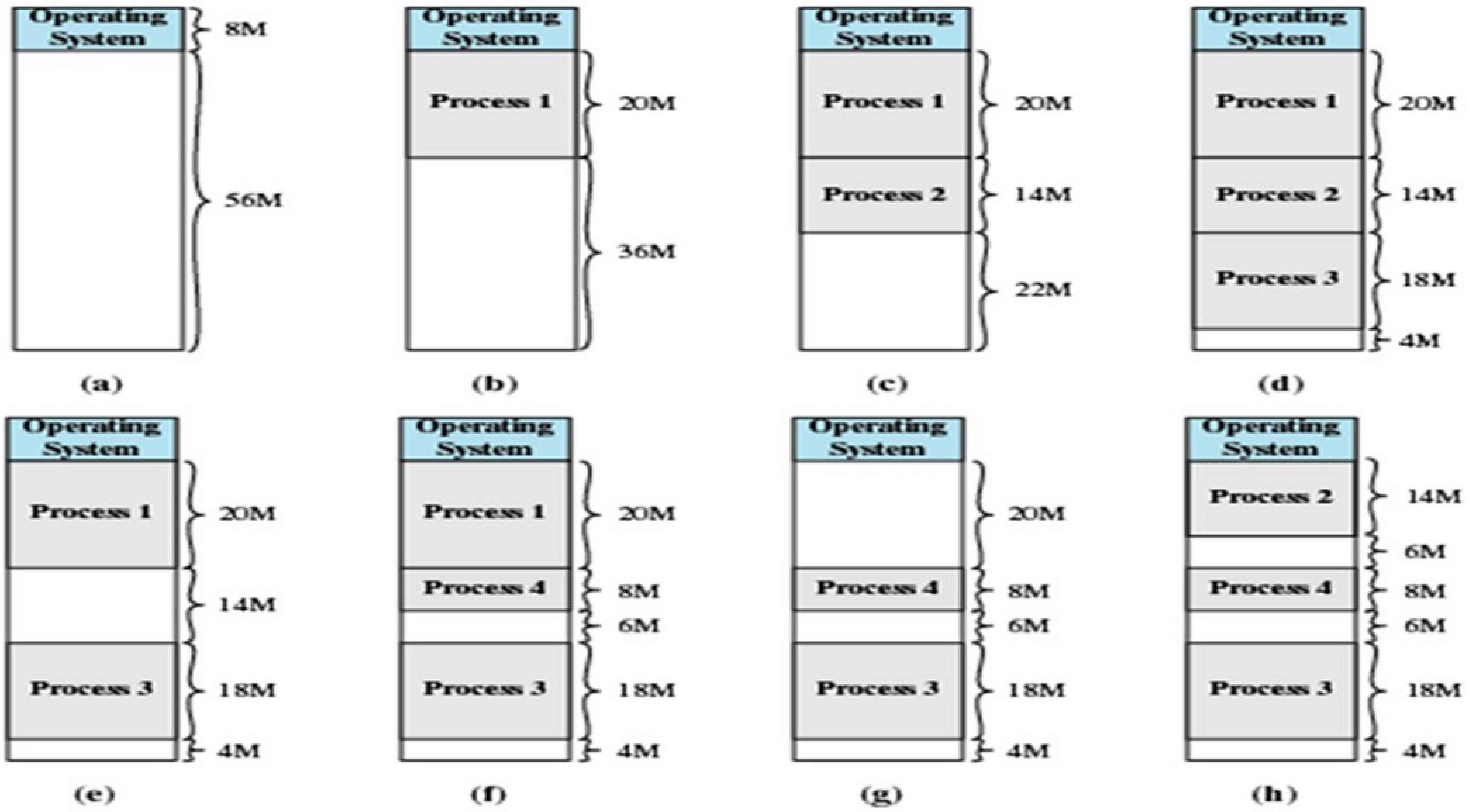
- As well as, it is not reasonable to know the requirement of the process beforehand.

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.

MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।



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

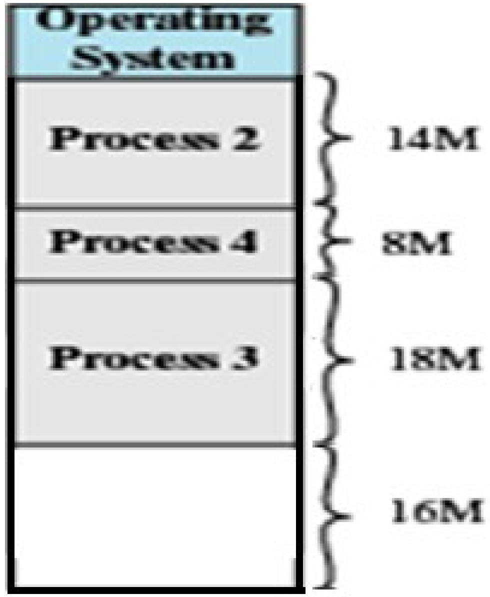
- Compaction helps to solve the problem of fragmentation, but it requires too much of CPU time.

- 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.



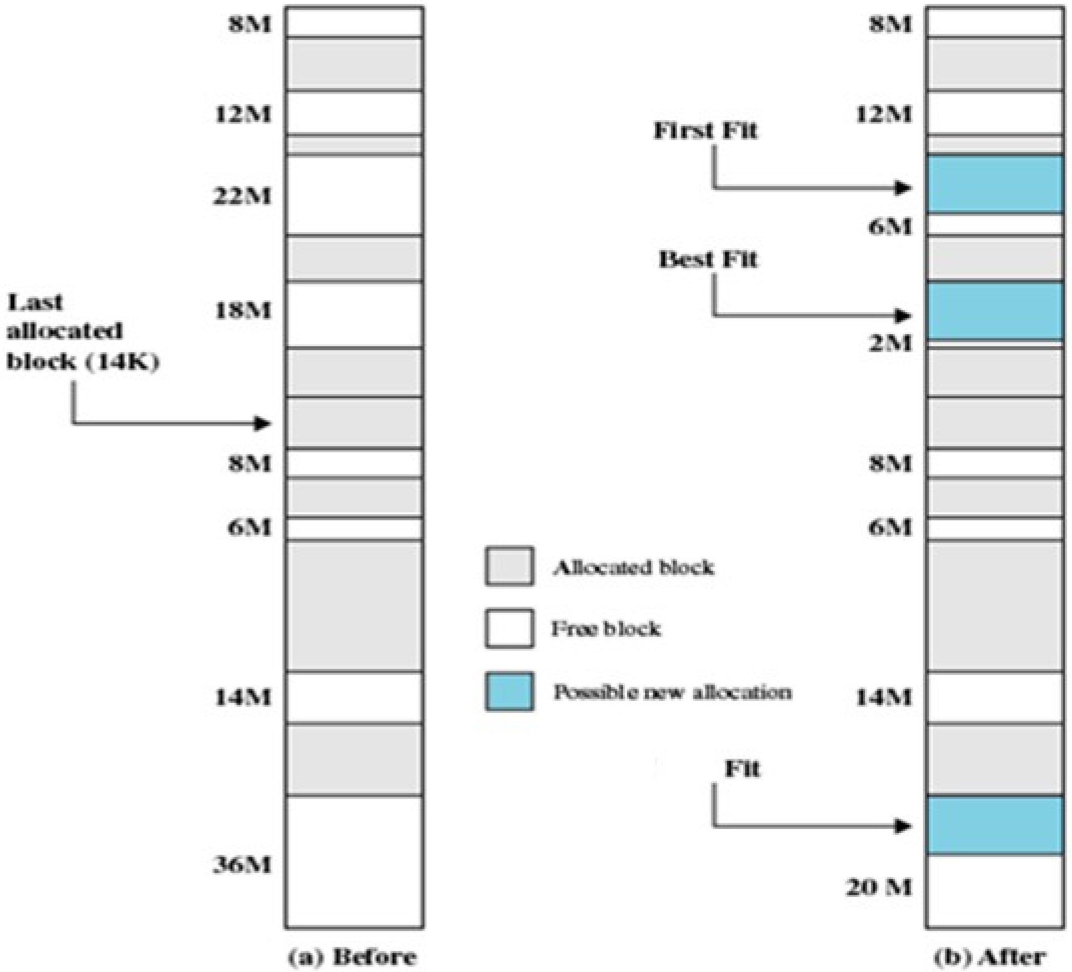
After Compaction

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

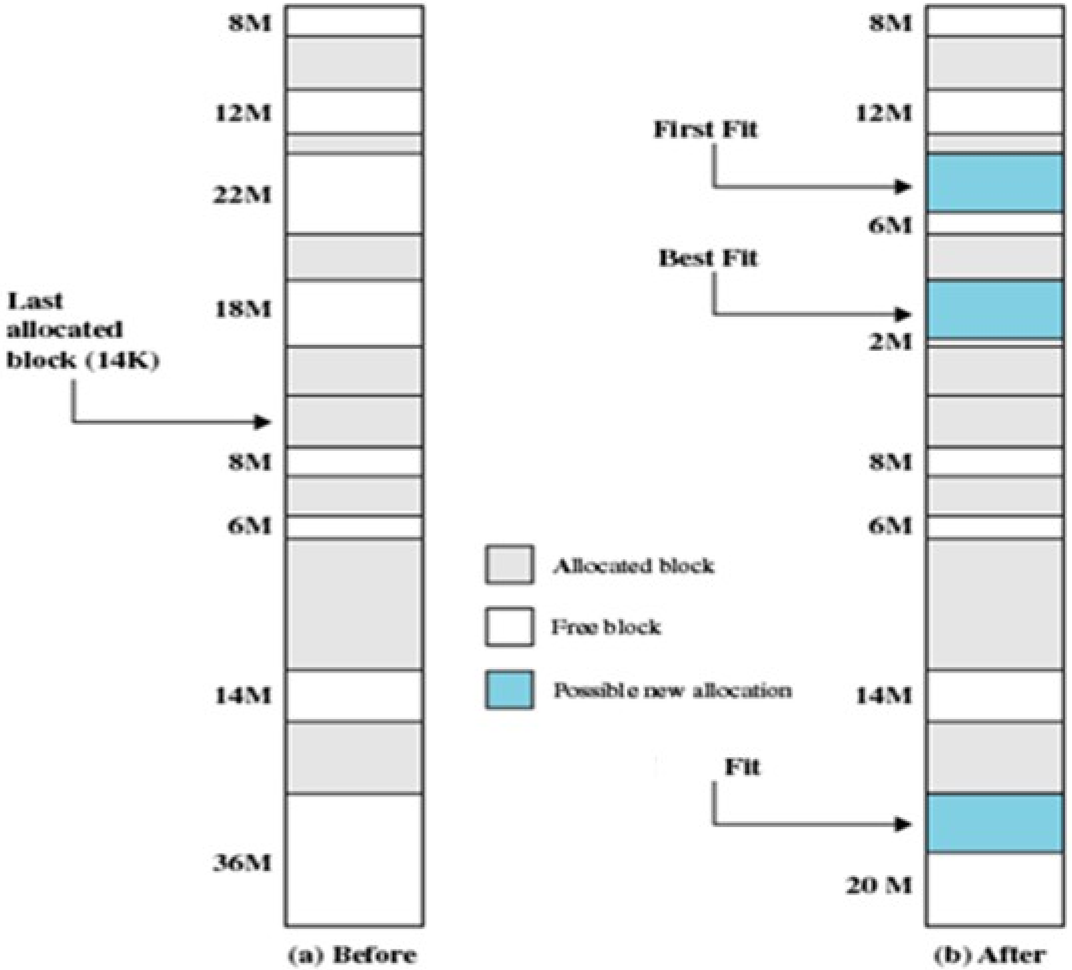
- Best-fit algorithm
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often



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



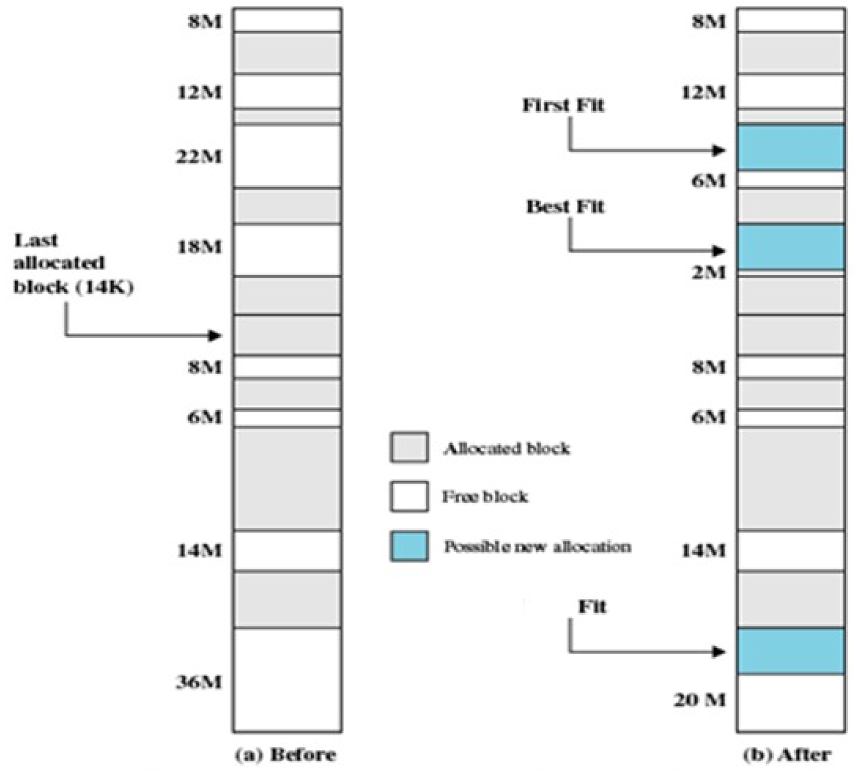
Example Memory Configuration Before and After Allocation of 16 Mbyte Block

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?

MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

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

Holes →

15	10	5	25	30	40

Processes

12

15-12=3

Modified list of holes - After assigning 15K hole to 12K process

03	10	5	25	30	40

2

3-2=1

Modified list of holes - After assigning 3K hole to 2K process

01	10	5	25	30	40

Continued ….

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

Holes → 01 10 5 25 30 40

Processes

25

25-25=0

Modified list of holes - After assigning 25K hole to 25K process

01 10 5 30 40

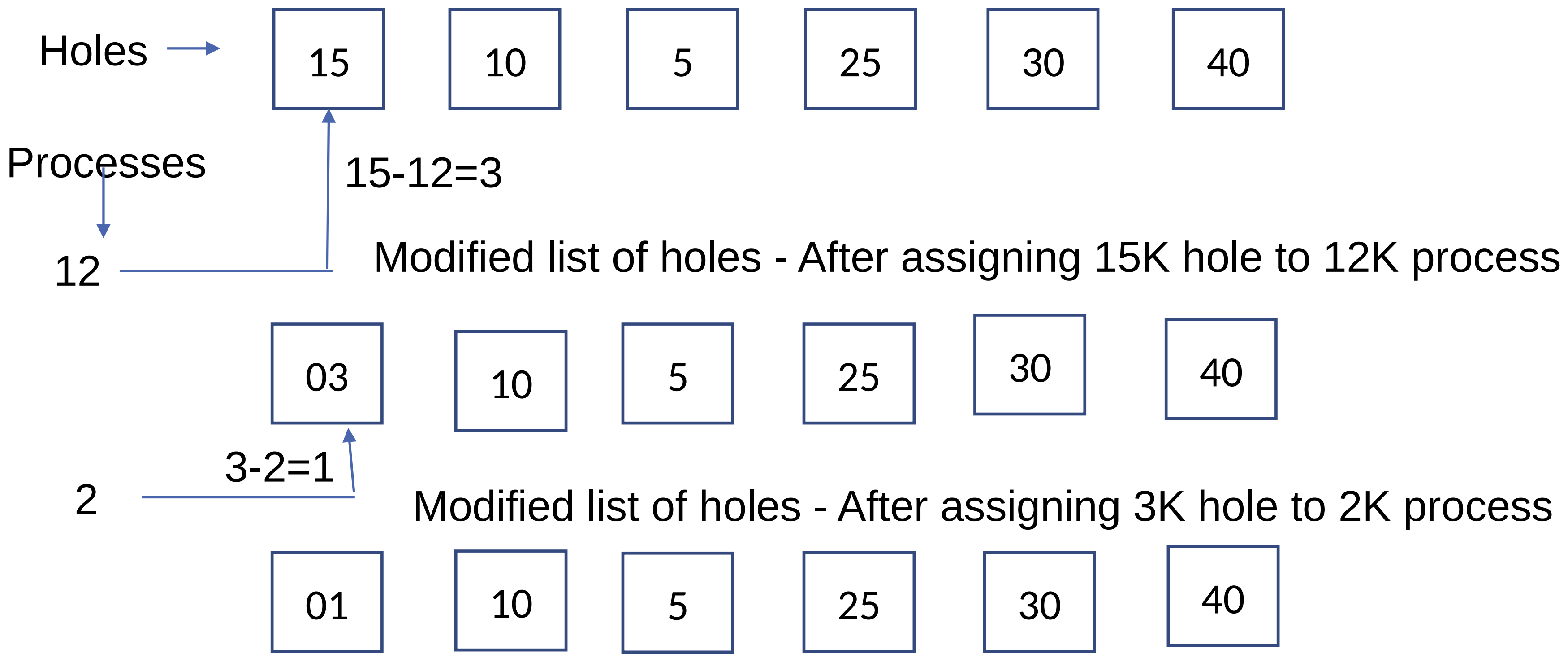
20

30-20=10

Modified list of holes - After assigning 30K hole to 20K process

01 10 5 10 40

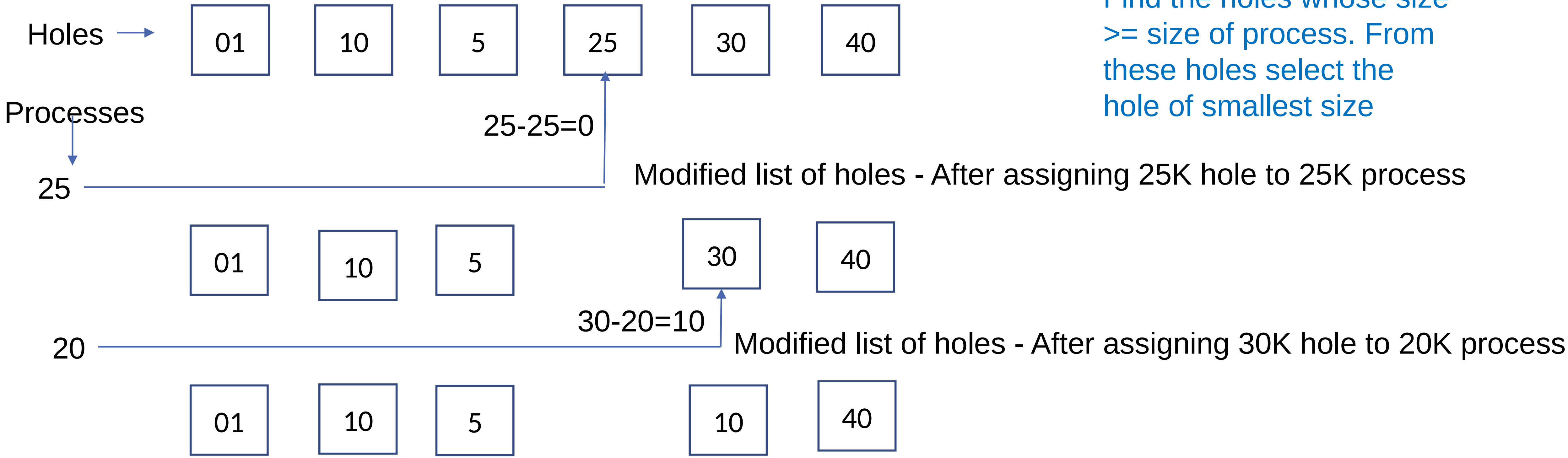
Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in **Best fit**



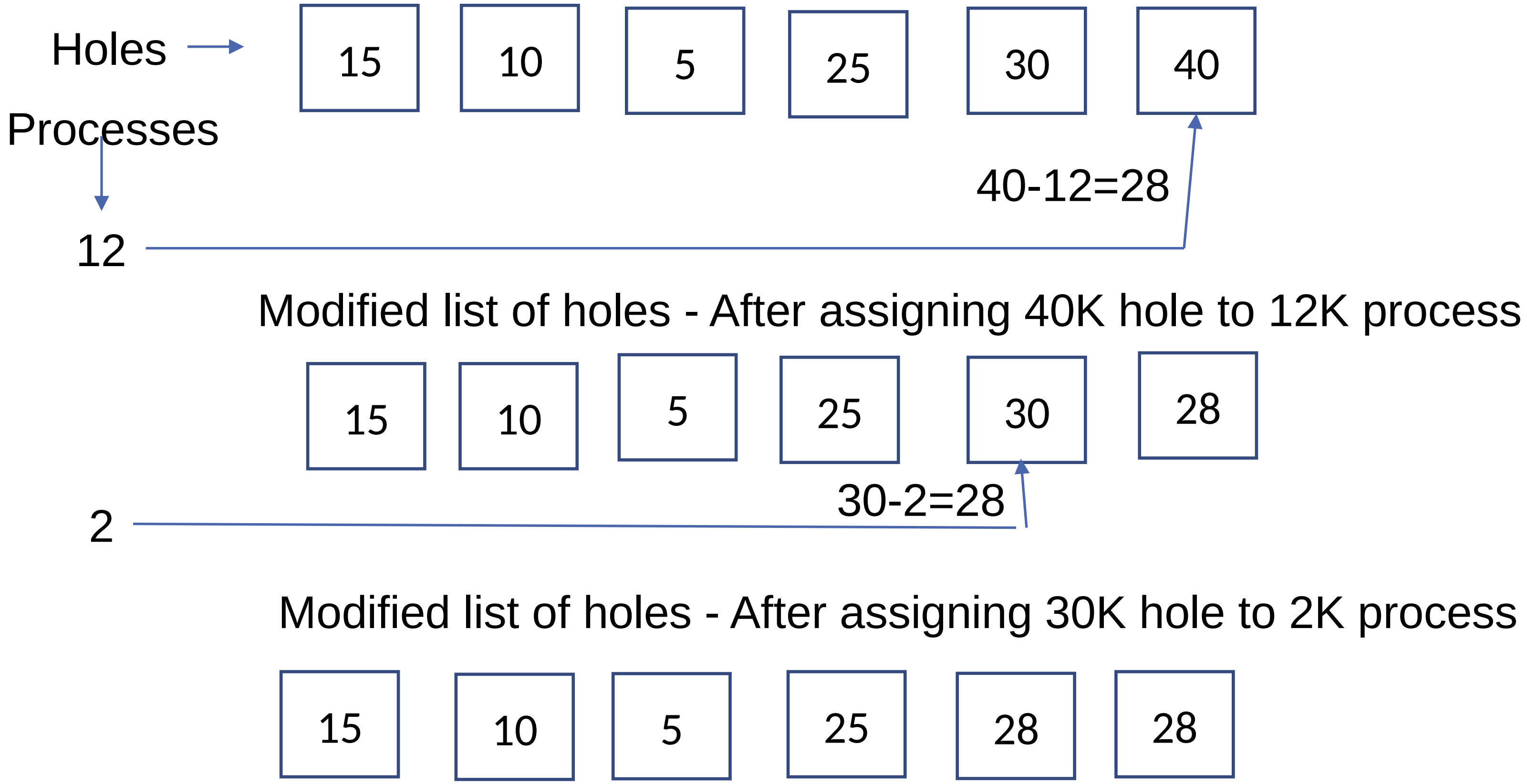
Best Fit: Search entire list
Find the holes whose size >= size of process. From these holes select the hole of smallest size

Continued ….

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



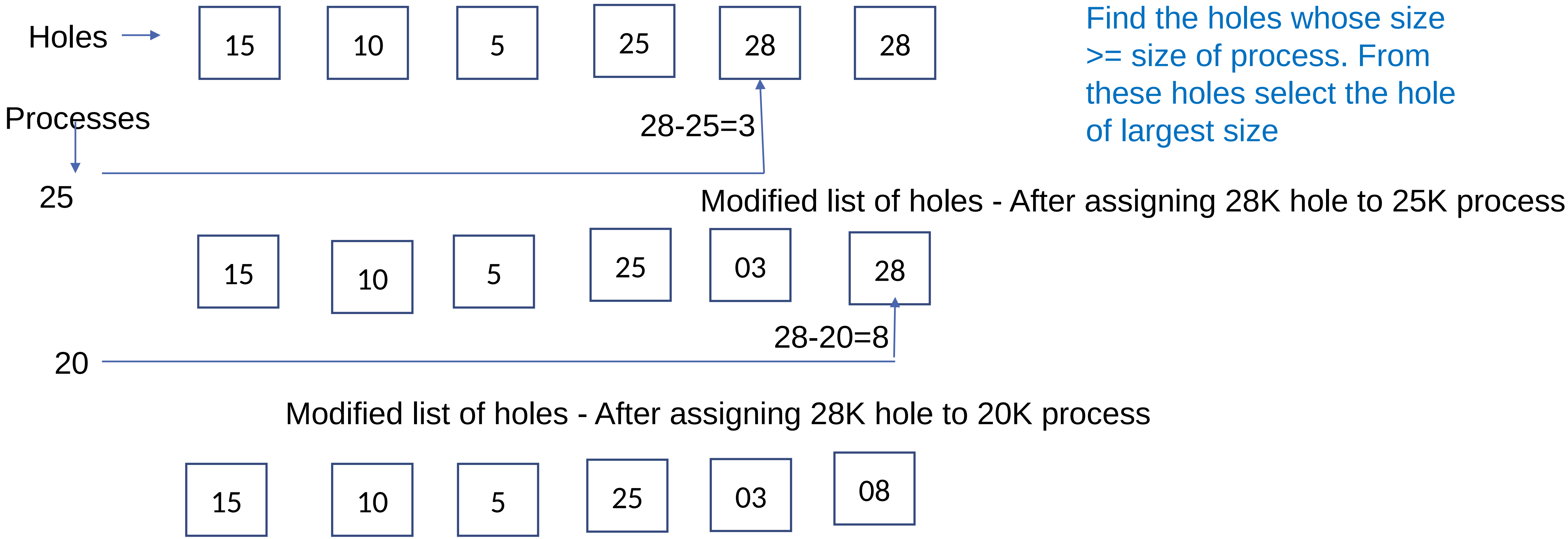
Free memory holes of sizes 15K, 10K, 5K, 25K, 30K, 40K are available. The processes of size 12K, 2K, 25K, 20K are to be allocated. How processes are placed in **Worst fit**



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

Continued ….

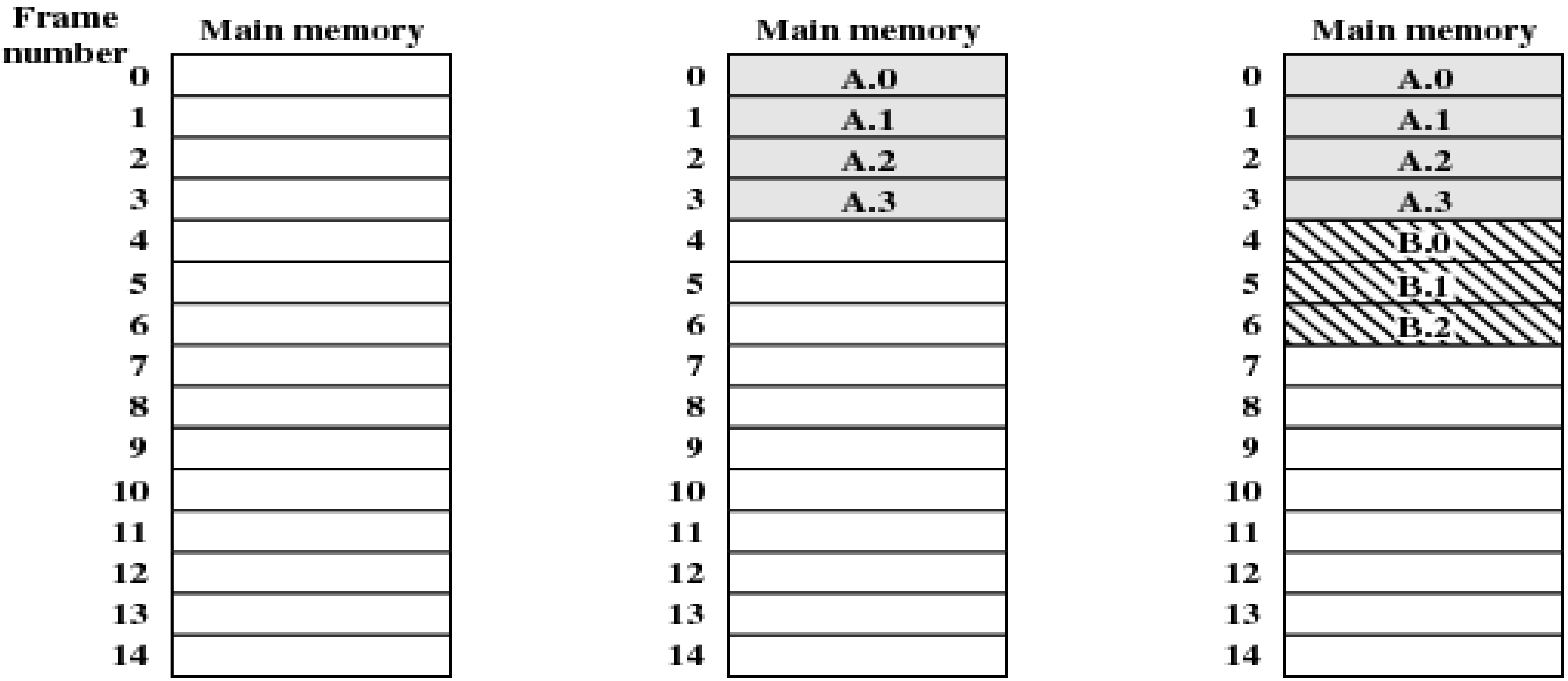
Worst Fit: Search entire list
Find the holes whose size >= size of process. From these holes select the hole of largest size



Paging

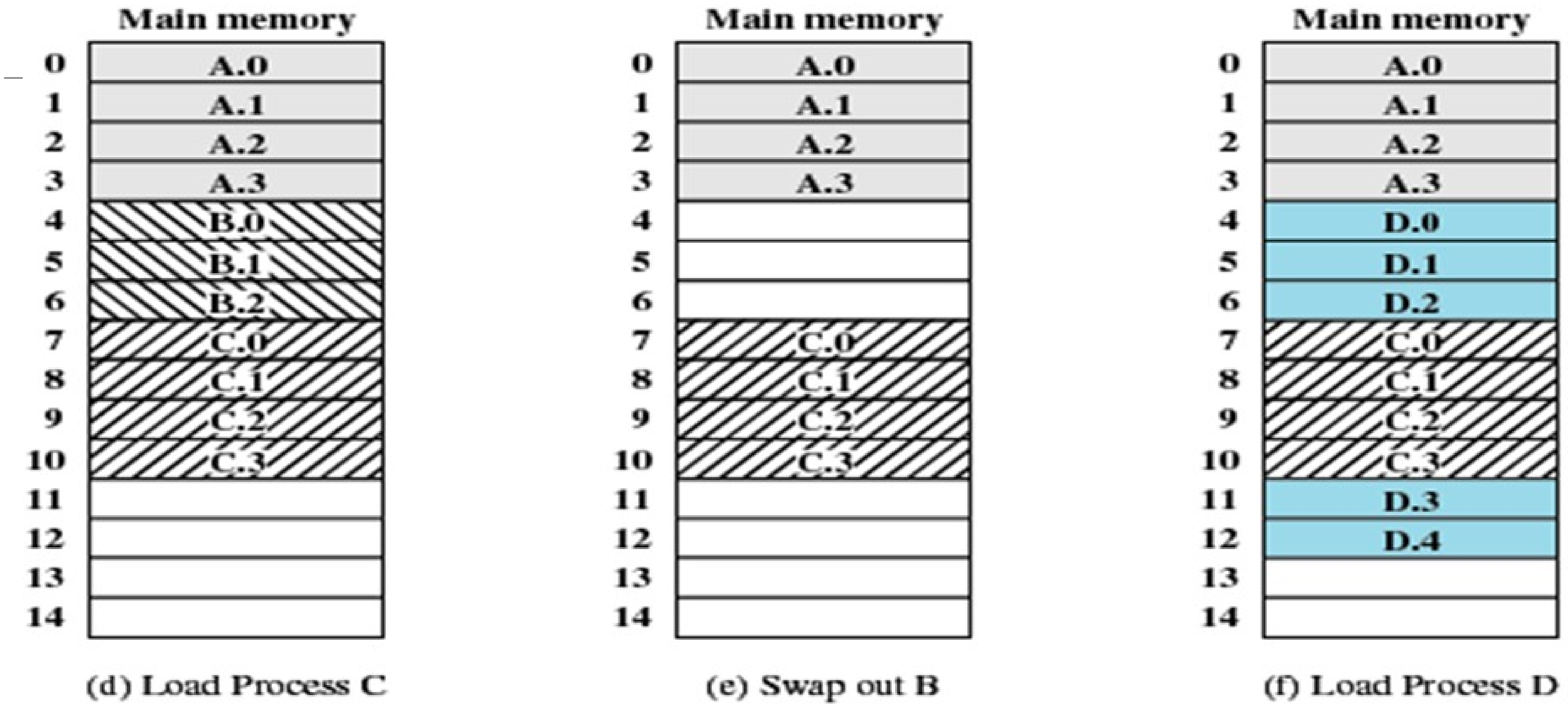
- Main memory is partitioned into equal fixed size chunks that are relatively small ---- called **frames**
- Each process is divided into small fixed sized chunks of the same size ---- called **pages**
- At a given point of time, some frames are in use & some are free
- Suppose process A stored on disk, 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



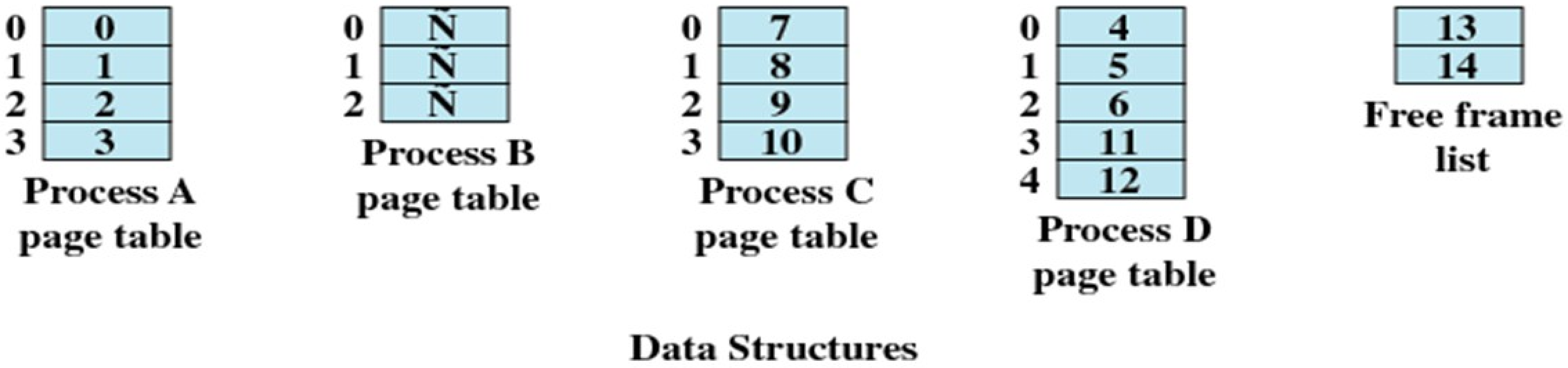
(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

Assignment of Process Pages to Free Frames



Assignment of Process Pages to Free Frames

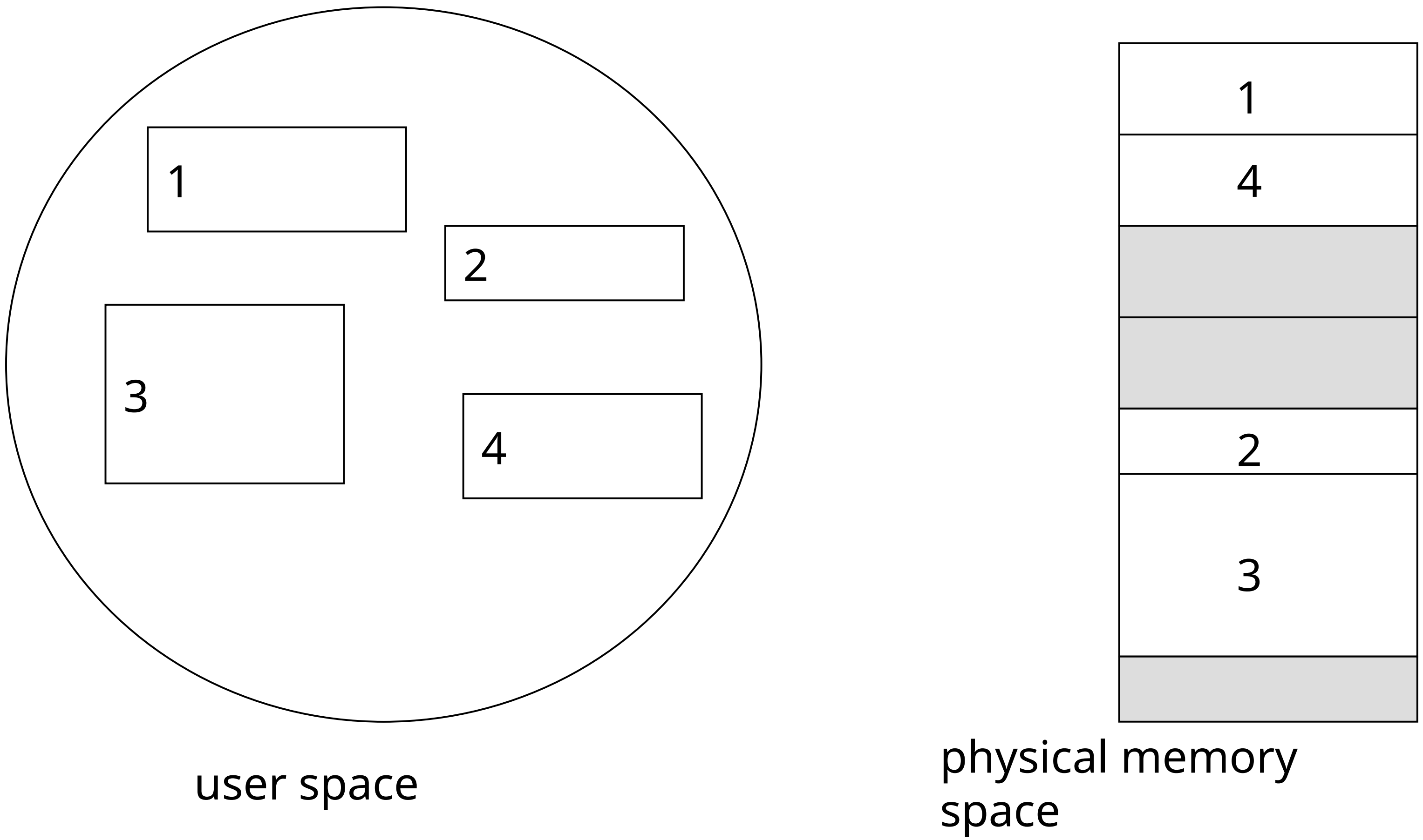
Page Tables for Example



Segmentation

- All segments of all programs do not have to be of the same length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning

Logical View of Segmentation



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

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

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**

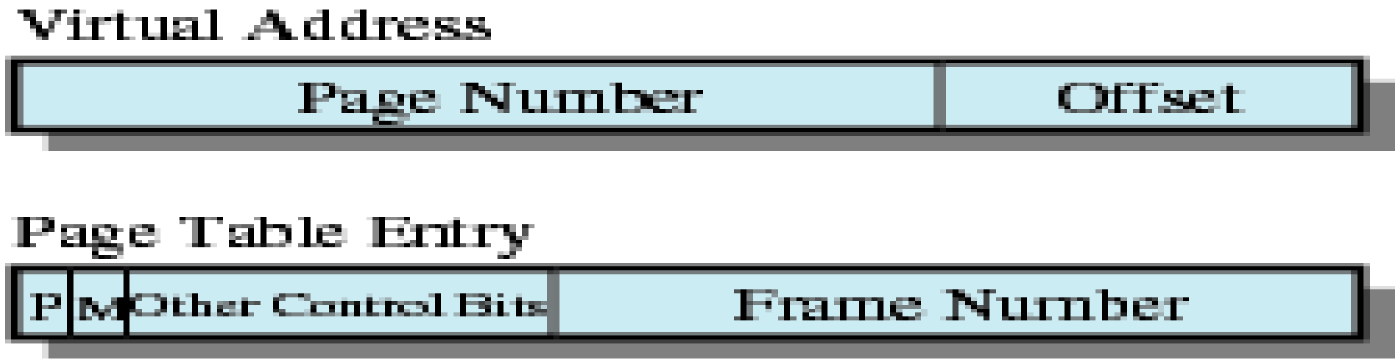
Thrashing

- 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

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

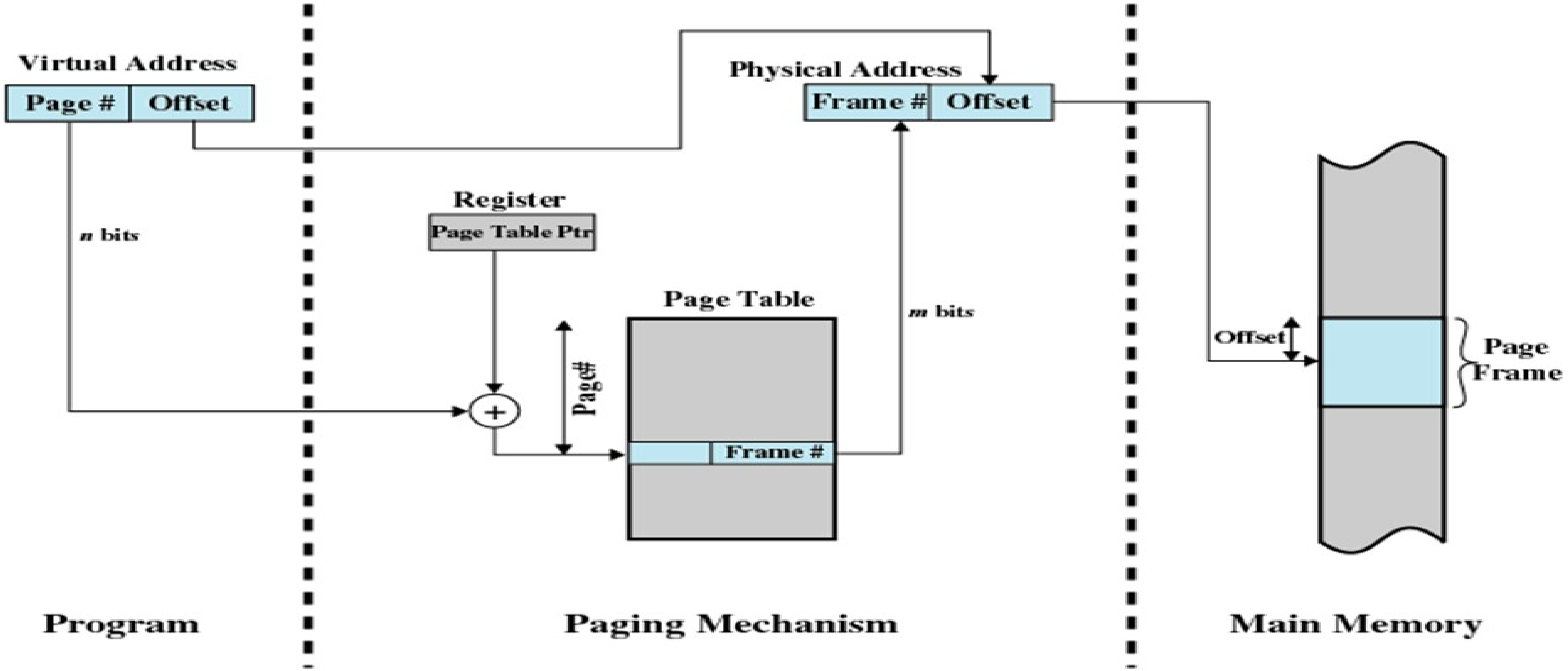
Paging Continued…



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

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

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

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.

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

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)

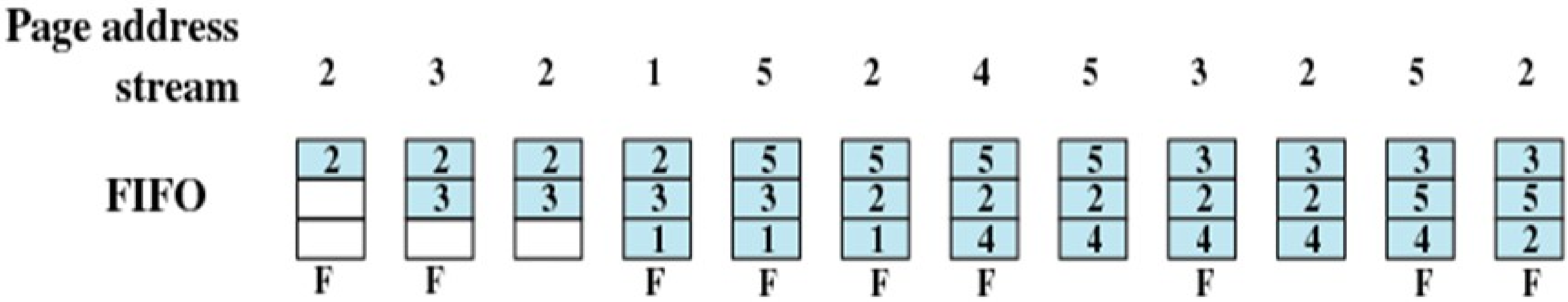
- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
- Simplest replacement policy to implement
- Page that has been in memory the longest is replaced
- These pages may be needed again very soon

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


Page address stream
2 3 2 1 5 2 4 5 3 2 5 2
FIFO
2 | 2 3 | 2 3 | 2 3 1 | 5 3 1 | 5 2 1 | 5 2 4 | 5 2 4 | 3 2 4 | 3 2 4 | 3 5 4 | 3 5 2
F F F F F F F F F

Belady’s Anomaly

The reference string is

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

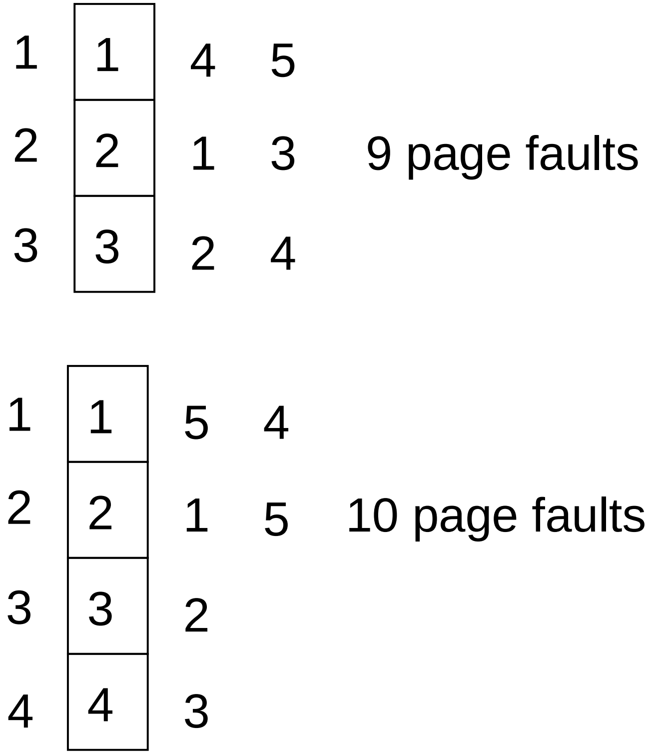
Find the number of page faults for 3 frames & 4 frames

Belady’s Anomaly

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

3 frames

4 frames



FIFO Replacement – Belady’s Anomaly

- more frames $\Rightarrow$ 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
- Page Address Stream: 2 3 2 1 5 2 4 5 3 2 5 2

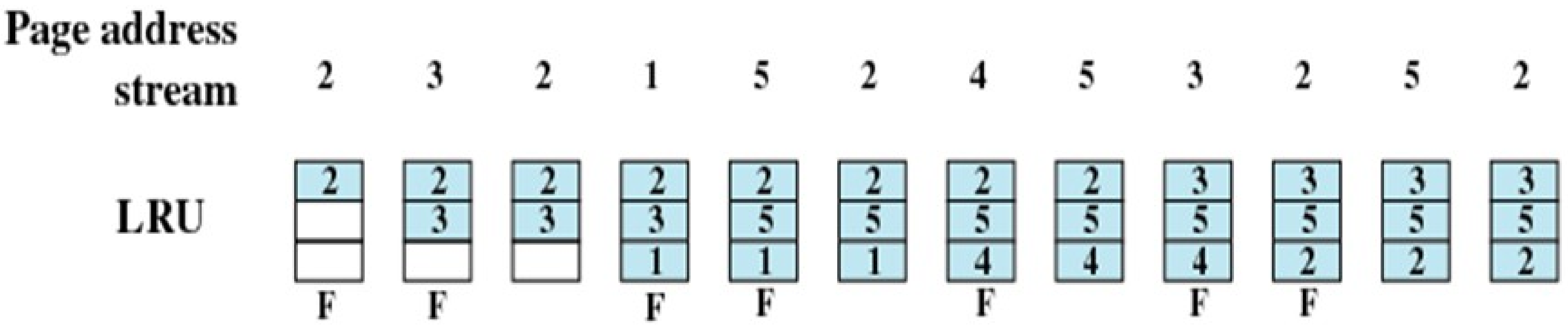
Example

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

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