Parul® University

# Operating System

**Prof. Vibhuti Patel,** Assistant Professor
Parul Institute of Computer Application

# CHAPTER-3

## Memory Management

# Needs for Memory Management

- Memory is cheap today, and getting cheaper
  - But applications are demanding more and more memory, there is never enough!
- Memory Management, involves swapping blocks of data from secondary storage.
- Memory I/O is slow compared to a CPU
  - The OS must cleverly time the swapping to maximize the CPU's efficiency

# Memory Management

- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

- Memory Management Requirements:

    1. Relocation
    2. Protection
    3. Sharing
    4. Logical organization
    5. Physical organization

# Memory Management Requirements

## 1. Relocation:

- programmer cannot know where the program will be placed in memory when it is executed
- a process may be (often) relocated in main memory due to swapping
- swapping enables the OS to have a larger pool of ready-to-execute processes
- memory references in code (for both instructions and data) must be translated to actual physical memory address

# Memory Management Requirements

## Memory Management Terms

| Term | Description |
| --- | --- |
| Frame | ***Fixed***-length block of main memory. |
| Page | ***Fixed***-length block of data in secondary memory (e.g. on disk). |
| Segment | ***Variable-length*** block of data that resides in secondary memory. |

# Memory Management Requirements

## 2. 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

## 3. Sharing:

- must allow several processes to access a common portion of main memory without compromising protection
- 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

## 4. Logical Organization:

- users write programs in modules with different characteristics
- instruction modules are execute-only
- data modules are either read-only or read/write
- some modules are private others are public
- To effectively deal with user programs, the OS and hardware should support a basic form of module to provide the required protection and sharing

# Memory Management Requirements

## 5. Physical Organization:

- secondary memory is the long term store for programs and data while main memory holds program and data currently in use

- moving information between these two levels of memory is a major concern of memory management (OS)

- it is highly inefficient to leave this responsibility to the application programmer

# Partitioning

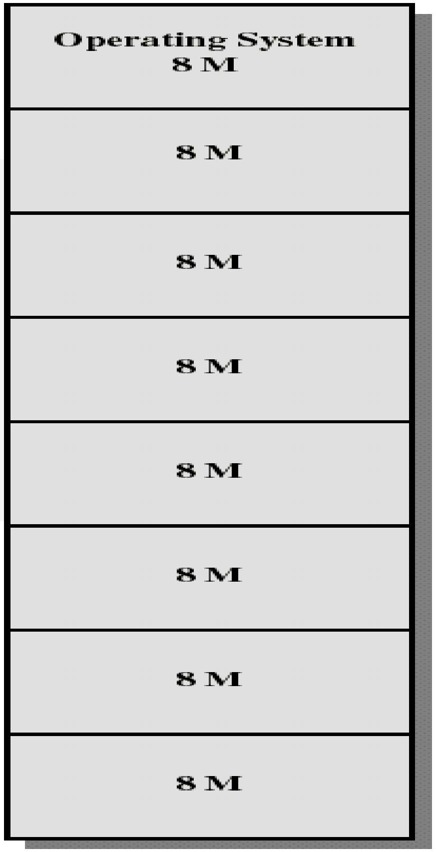
- A partition is a logical division of a hard disk that is treated as a separate unit by operating systems (OSes) and file systems.

- Types of Partitioning

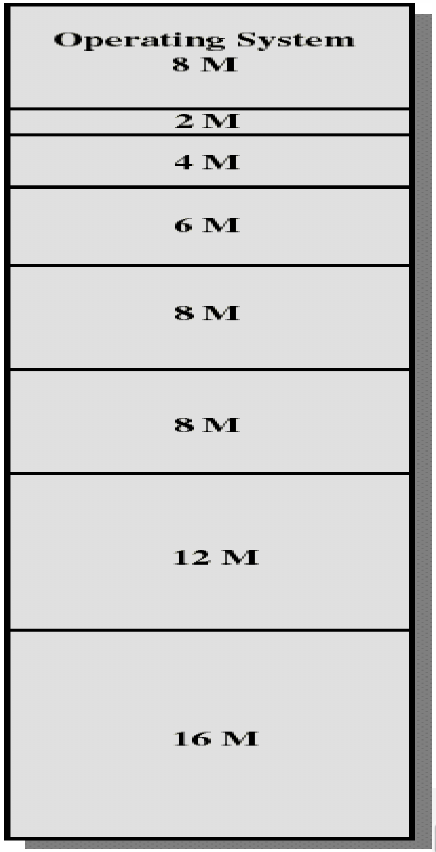
    1. Fixed Partitioning
    2. Dynamic Partitioning
    3. Simple Paging
    4. Segmentation

# Fixed Partitioning

- Partition main memory into a set of non overlapping regions called partitions
- Partitions can be of equal or unequal sizes




Equal-size partitions




Unequal-size partitions

Image source : Book

# Fixed Partitioning

- any process whose size is less than or equal to a partition size can be loaded into the partition.
- if all partitions are occupied, the operating system can swap a process out of a partition.
- a program may be too large to fit in a partition. The programmer must then design the program with overlays.
- when the module needed is not present the user program must load that module into the program's partition, overlaying whatever program or data are there.

# Fixed Partitioning

- Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This is called **internal fragmentation.**

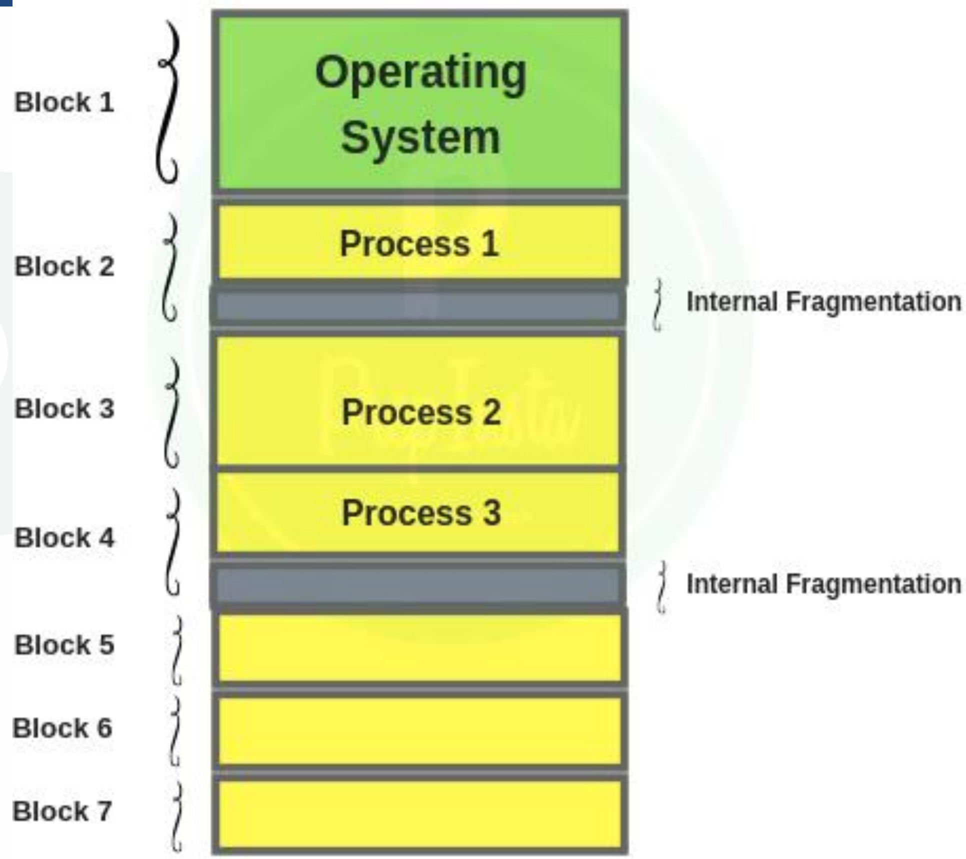



Image source : Google

# Fixed Partitioning

- To reduce **internal fragmentation** un-equal size partition is used.
- Example: Programs up to 16M can be accommodated without overlay in figure.

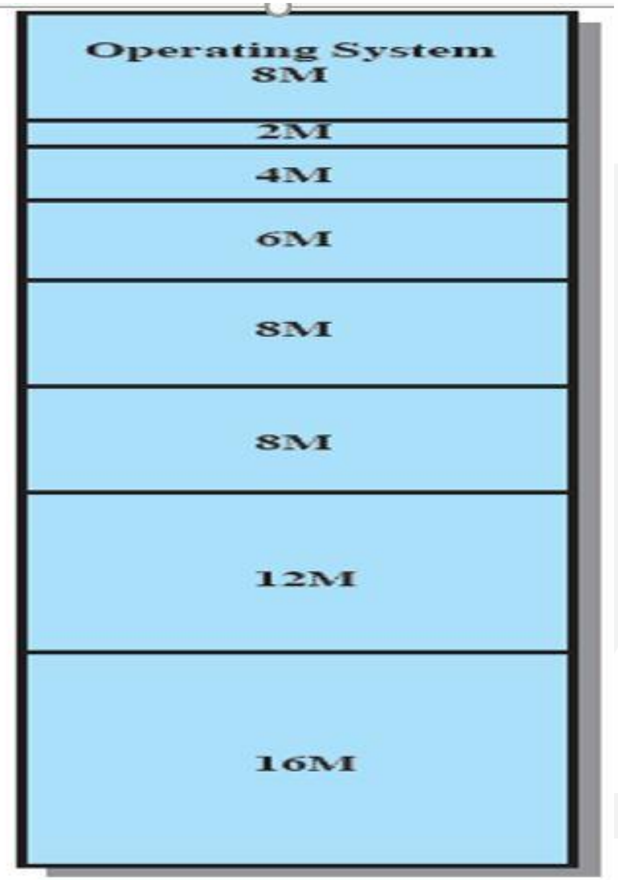



Image source : Book

# Fixed Partitioning Placement Algorithms

## Equal-size Partitions:

- If there is an available partition, a process can be loaded into that partition
- because all partitions are of equal size, it does not matter which partition is used
- If all partitions are occupied by blocked processes, choose one process to swap out to make room for the new process

# Fixed Partitioning Placement Algorithms

## Unequal-size Partitions (Use of multiple queues):

- assign each process to the smallest partition within which it will fit
- A queue for each partition size
- tries to minimize internal fragmentation

- **Problem:** some queues will be empty if no processes within a size range is present

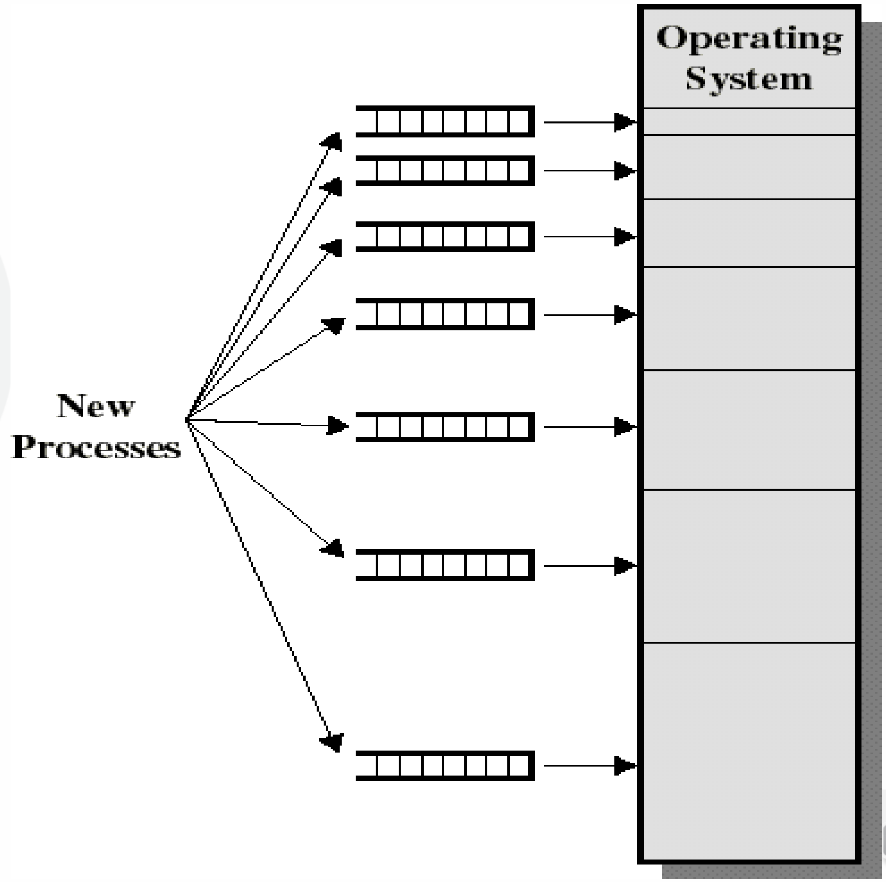



Image source : Google

# Fixed Partitioning Placement Algorithms

## Unequal-size Partitions (Use of single queue):

- When its time to load a process into main memory the smallest available partition that will hold the process is selected

- increases the level of multiprogramming at the expense of internal fragmentation

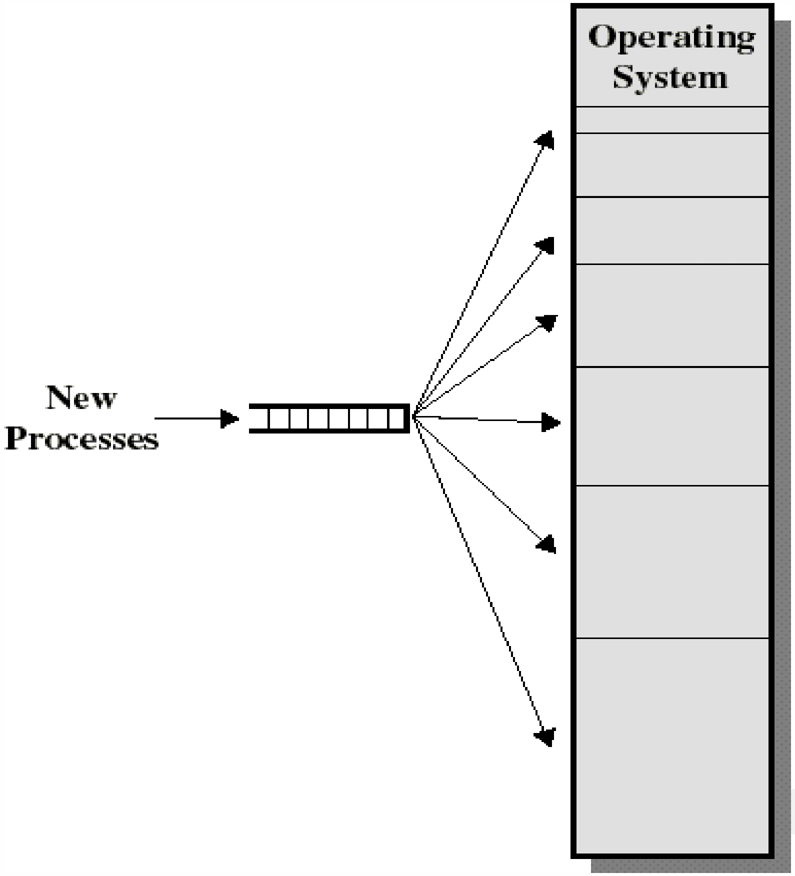



Image source : Google

# Dynamic Partitioning

- Partitions are of variable length and number
- Each process is allocated exactly as much memory as it requires
- Eventually holes are formed in main memory. This is called **external fragmentation.**
- To reduce external fragmentation **compaction** is used.
- Must use **compaction** to shift processes so they are contiguous and all free memory is in one block.

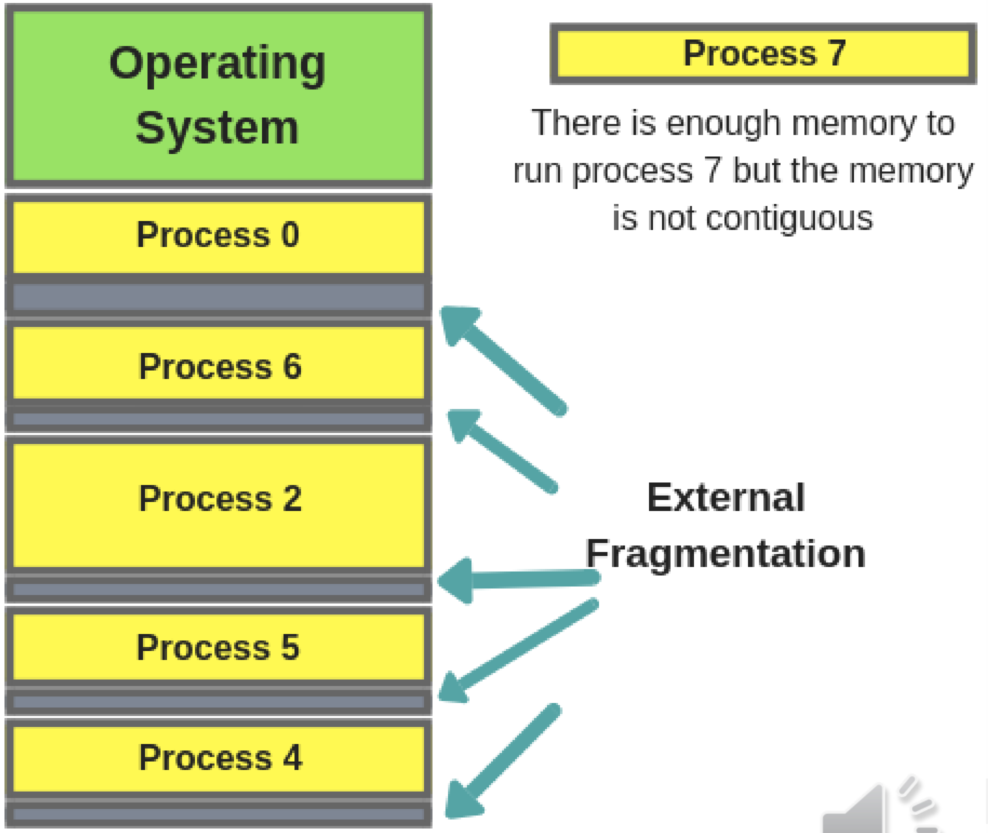



Image source : Google

# Dynamic Partitioning

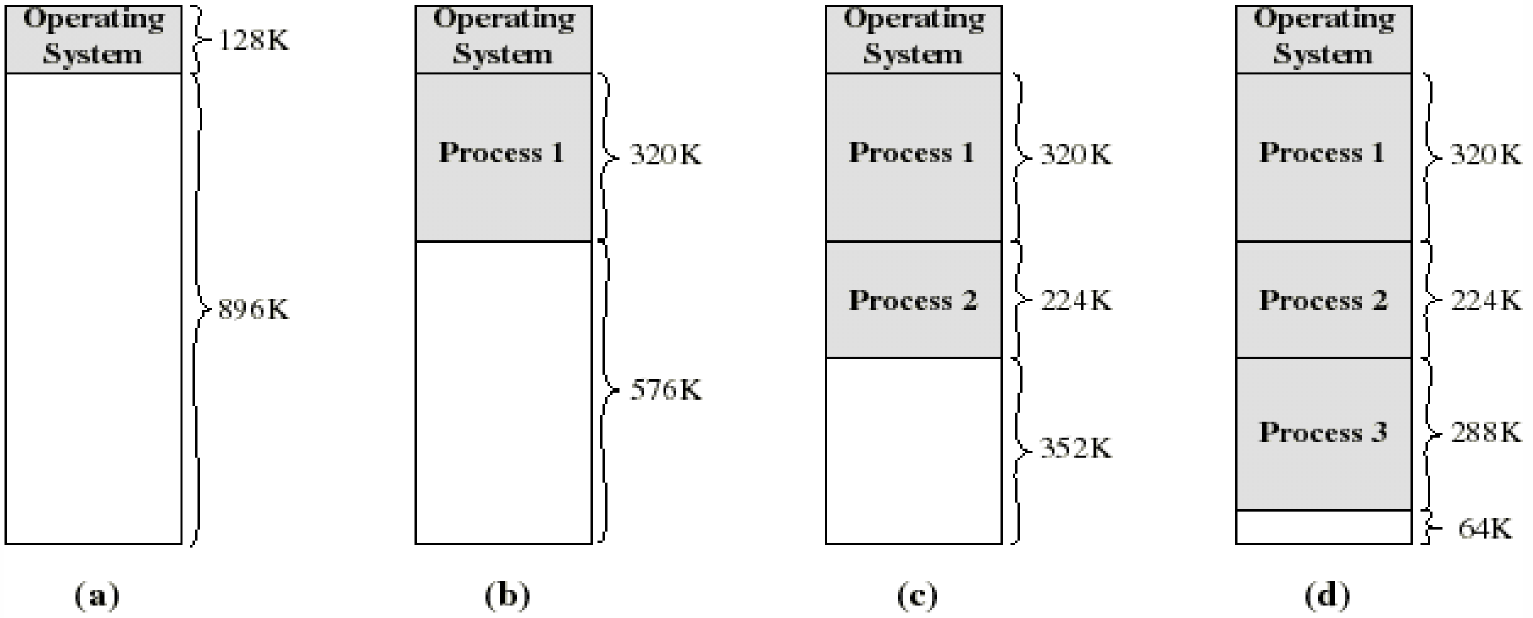



Image source : Google

- A hole of 64K is left after loading 3 processes: not enough room for another process
- Eventually each process is blocked. The OS swaps out process 2 to bring in process 4

# Dynamic Partitioning

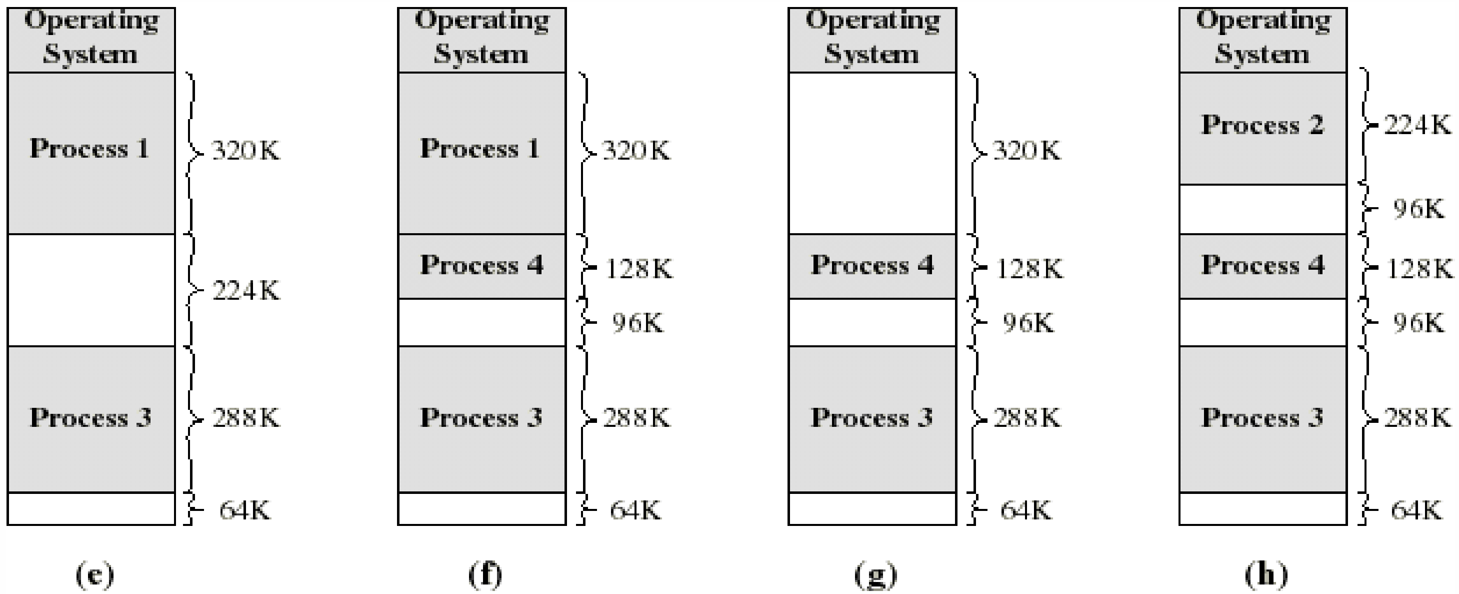



Image source : Google

- another hole of 96K is created
- Eventually each process is blocked. The OS swaps out process 1 to bring in again process 2 and another hole of 96K is created.
- Compaction would produce a single hole of 256K

# Dynamic Partitioning Placement Algorithm

- Used to decide which free block to allocate to a process
- Goal: to reduce usage of compaction (time consuming)
- Possible algorithms:
- Best-fit: choose smallest hole
- First-fit: choose first hole from beginning
- Next-fit: choose first hole from last placement

Example: Memory Configuration before & after allocation of 16 Kbytes Block

# Dynamic Partitioning Placement Algorithm

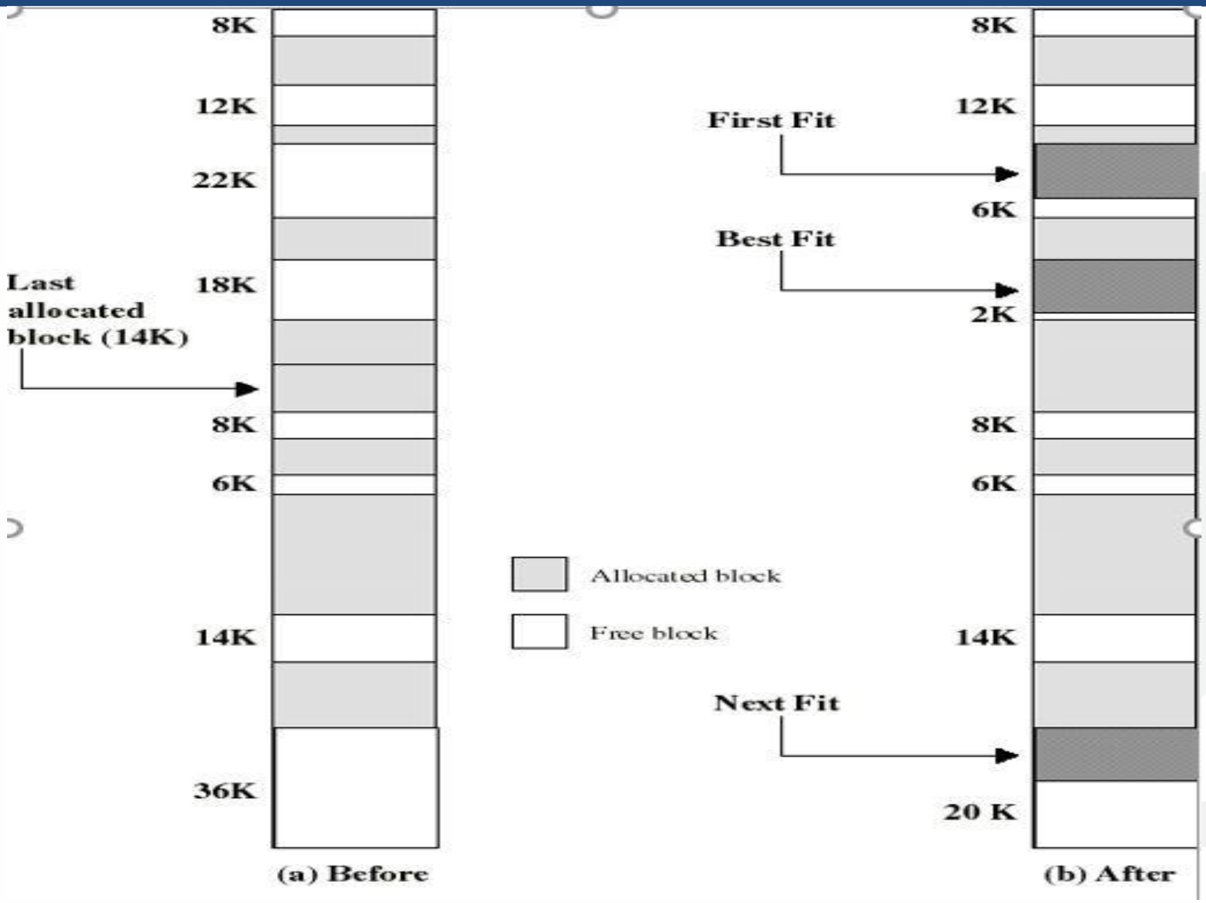



Image source : Google

# Buddy System

- In a fixed partitioning scheme limits the number of active processes and may use space inefficiently if there is a poor match between available partition sizes and process sizes.
- A dynamic partitioning scheme is more complex to maintain and includes the overhead of compaction.
- An interesting compromise is the buddy system.

# Buddy System

Example: 1-Mbyte initial block.

- The first request, A, is for 100 Kbytes, for which a 128K block is needed.
- The initial block is divided into two 512K buddies.
- The first of these is divided into two 256K buddies,
- and the first of these is divided into two 128K buddies,
- one of which is allocated to A.

# Buddy System

Example: 1-Mbyte initial block.

- The next request, B, requires a 256K block. Such a block is already available and is allocated.
- The process continues with splitting and coalescing occurring as needed.
- Note that when E is released, two 128K buddies are coalesced into a 256K block, which is immediately coalesced with its buddy

# Buddy System

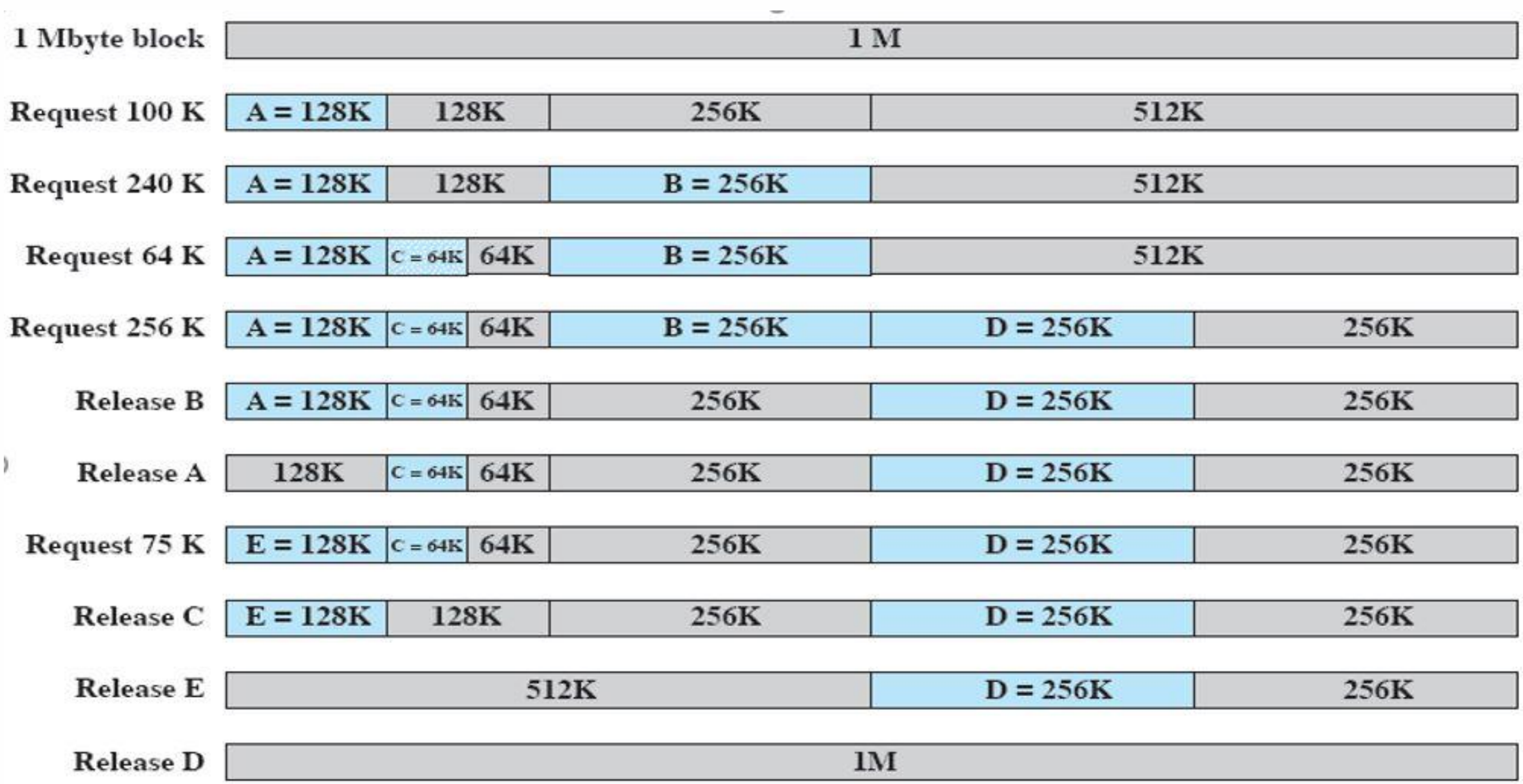



Image source : Book

# Tree Representation of Buddy System

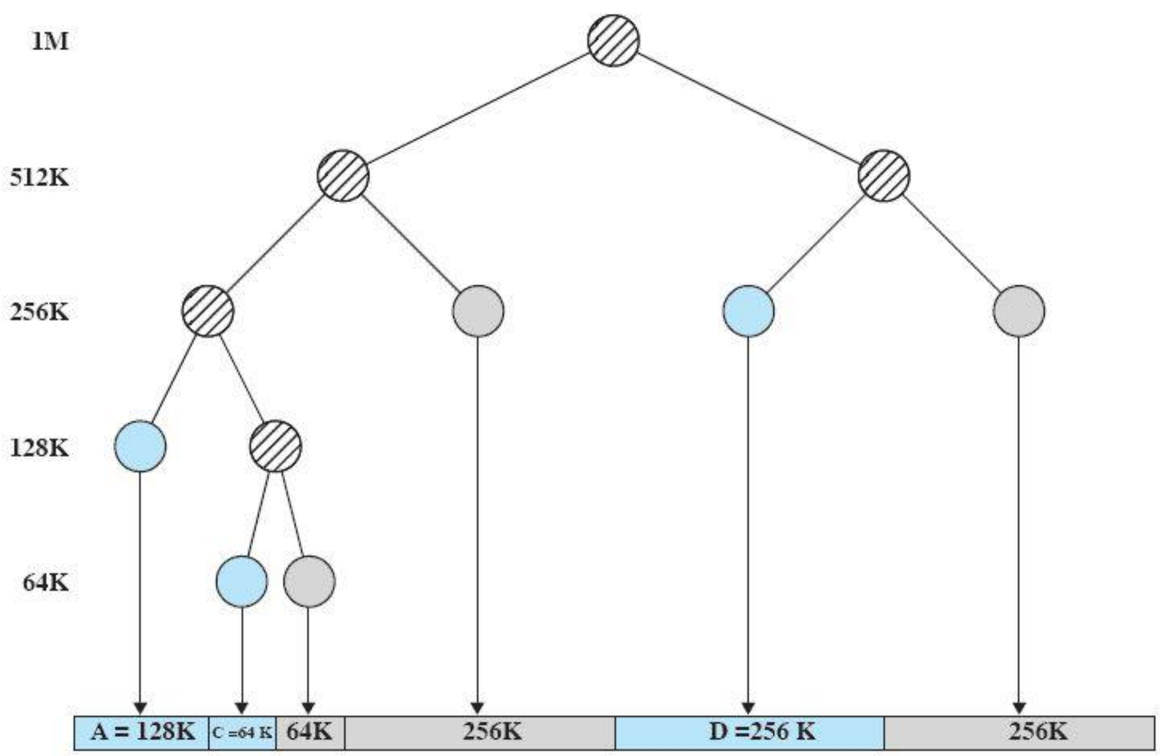



Image source : Book

# Simple Paging

- Main memory is partition into equal fixed-sized chunks (of relatively small size)
- Track: each process is also divided into chunks of the same size called pages
- The process pages can thus be assigned to the available chunks in main memory called frames (or page frames)
- Consequence: a process does not need to occupy a contiguous portion of memory

# Example of Process Loading – Simple Paging

| Frame number | Main memory | Main memory | Main memory | Main memory |
|---|---|---|---|---|
| 0 | | A.0 | A.0 | A.0 |
| 1 | | A.1 | A.1 | A.1 |
| 2 | | A.2 | A.2 | A.2 |
| 3 | | A.3 | A.3 | A.3 |
| 4 | | | B.0 | B.0 |
| 5 | | | B.1 | B.1 |
| 6 | | | B.2 | B.2 |
| 7 | | | | C.0 |
| 8 | | | | C.1 |
| 9 | | | | C.2 |
| 10 | | | | C.3 |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| | (a) Fifteen Available Pages | (b) Load Process A | (b) Load Process B | (d) Load Process C |

Image source : Book

- Now suppose that process B is swapped out

# Example of Process Loading – Simple Paging

- When process A and C are blocked, the pager loads a new process D consisting of 5 pages
- Process D does not occupied a contiguous portion of memory
- There is no external fragmentation
- Internal fragmentation consist only of the last page of each process

| | Main memory |
|---|---|
| 0 | A.0 |
| 1 | A.1 |
| 2 | A.2 |
| 3 | A.3 |
| 4 | |
| 5 | |
| 6 | |
| 7 | C.0 |
| 8 | C.1 |
| 9 | C.2 |
| 10 | C.3 |
| 11 | |
| 12 | |
| 13 | |
| 14 | |

(e) Swap out B

| | Main memory |
|---|---|
| 0 | A.0 |
| 1 | A.1 |
| 2 | A.2 |
| 3 | A.3 |
| 4 | D.0 |
| 5 | D.1 |
| 6 | D.2 |
| 7 | C.0 |
| 8 | C.1 |
| 9 | C.2 |
| 10 | C.3 |
| 11 | D.3 |
| 12 | D.4 |
| 13 | |
| 14 | |

(f) Load Process D

Image source : Book

# Page Table

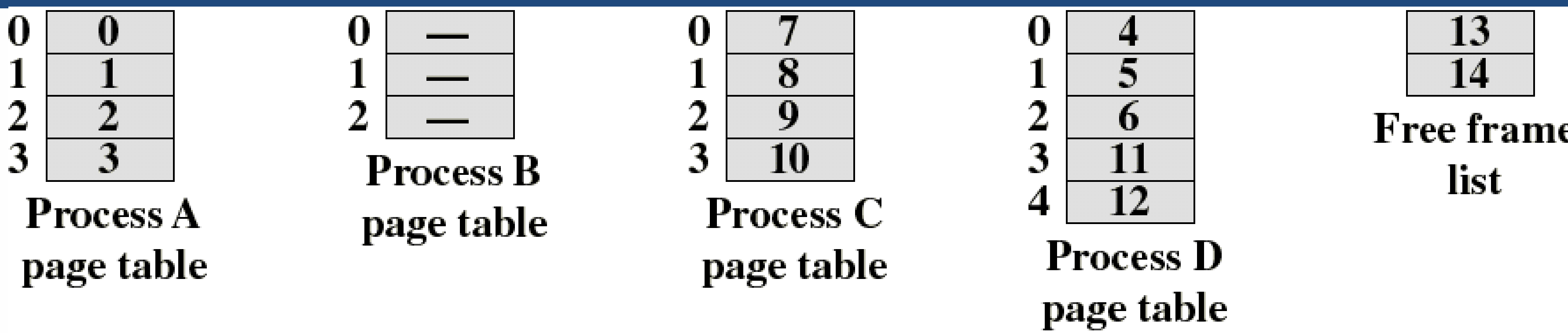



Image source : Book

- The OS now needs to maintain (in main memory) a page table for each process
- Each entry of a page table consist of the frame number where the corresponding page is physically located
- The page table is indexed by the page number to obtain the frame number
- A free frame list, available for pages, is maintained

# Segmentation

- Each program is subdivided into blocks of non-equal size called segments
- When a process gets loaded into main memory, its different segments can be located anywhere
- Each segment is fully packed with instructs/data: no internal fragmentation
- There is external fragmentation; it is reduced when using small segments

# Segmentation

- In contrast with paging, segmentation is visible to the programmer
  - provided as a convenience to organize logically programs (ex: data in one segment, code in another segment)
  - must be aware of segment size limit

- The OS maintains a segment table for each process. Each entry contains:
  - the starting physical addresses of that segment.
  - the length of that segment (for protection)

# Virtual Memory

- Virtual Memory: A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory & it is a section of a hard disk that's set up to emulate the computer's RAM.

- Virtual Address: The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.

- Virtual Address Space: The virtual storage assigned to a process.

- Address Space: The range of memory addresses available to a process.

- Real Address: The address of a storage location in a main memory.

# Paging

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- Two extra bits are needed to indicate:
    - whether the page is in main memory or not
    - Whether the contents of the page has been altered since it was last loaded
- (see next slide)

# Page Table

- It is typical to associate a unique page table with each process.
- A bit is needed in each page table entry to indicate whether the corresponding page is present (P) in main memory or not.
- If the bit indicates that the page is in memory, then the entry also includes the frame number of that page.
- The page table entry includes a modify (M) bit, indicating whether the contents of the corresponding page have been altered since the page was last loaded

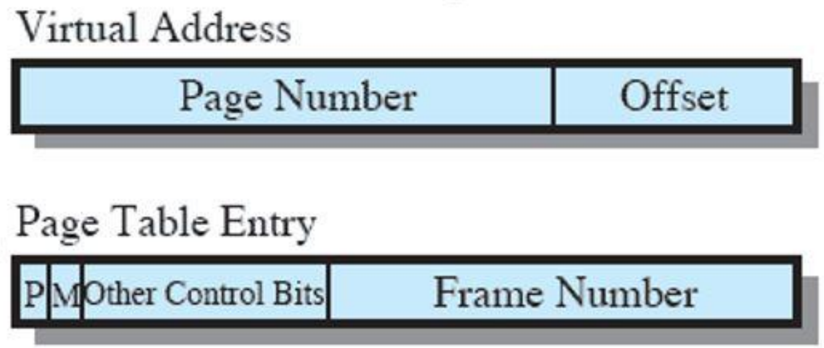



Image source : Book

# Page Table

- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

# Replacement Policy

- When all of the frames in main memory are occupied and it is necessary to bring in a new page, the replacement policy determines which page currently in memory is to be replaced.

## Which page is replaced?

- Page removed should be the page least likely to be referenced in the near future

- How is that determined?

- Principal of locality again

# Replacement Algorithms

There are certain basic algorithms that are used for the selection of a page to replace, they include

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)

# Replacement Algorithms

Example:

•An example of the implementation of these policies will use a page address stream formed by executing the program is

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

•Which means that the first page referenced is 2,

•the second page referenced is 3,

•And so on.

# Optimal Policy

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

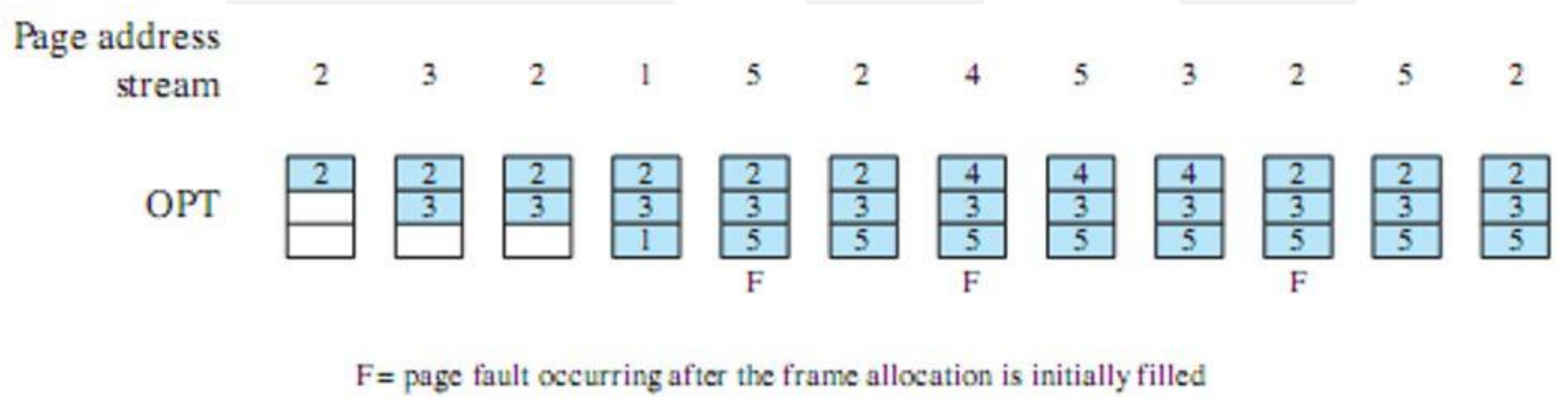


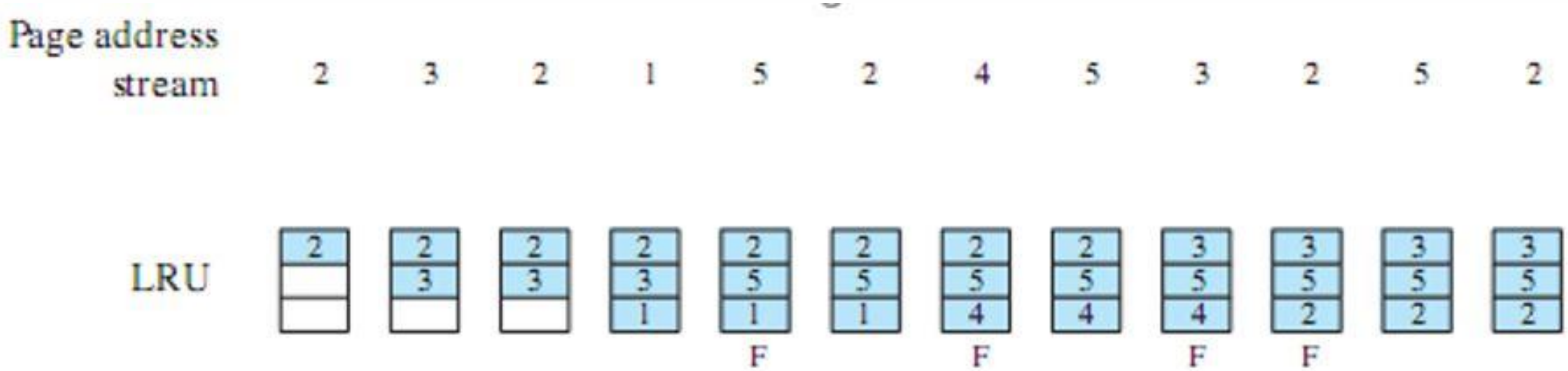

Image source : Book

- The optimal policy produces three page faults after the frame allocation has been filled.

# Least Recently Used (LRU) Policy

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement
- One approach is to tag each page with the time of last reference.
- This requires a great deal of overhead.

# Least Recently Used (LRU) Policy




F= page fault occurring after the frame allocation is initially filled

Image source : Book

- The LRU policy does nearly as well as the optimal policy.
- In this example, there are four page faults

# First In First Out (FIFO) Policy

- 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
  - But, these pages may be needed again very soon if it hasn’t truly fallen out of use

# First In First Out (FIFO) Policy

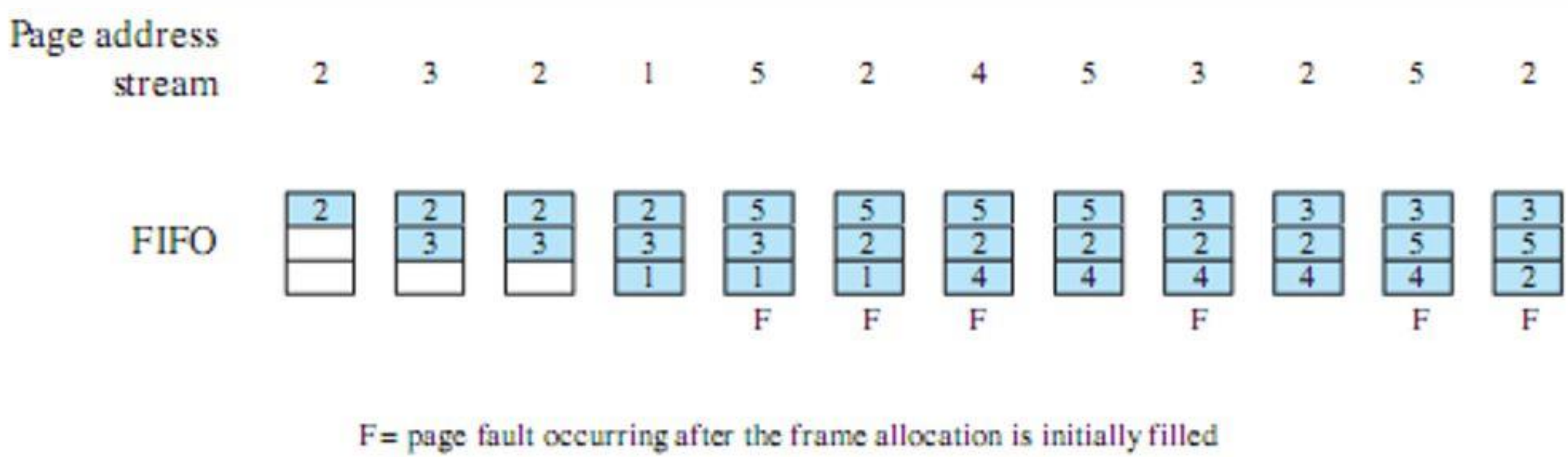



Image source : Book

- The FIFO policy results in six page faults.
- Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

# DIGITAL LEARNING CONTENT


Parul® University

www.paruluniversity.ac.in