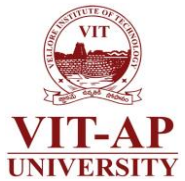
CSE2008: Operating Systems

L27 L28 & L29: Virtual Memory Concept



Dr. Subrata Tikadar

SCOPE, VIT-AP University

Recap

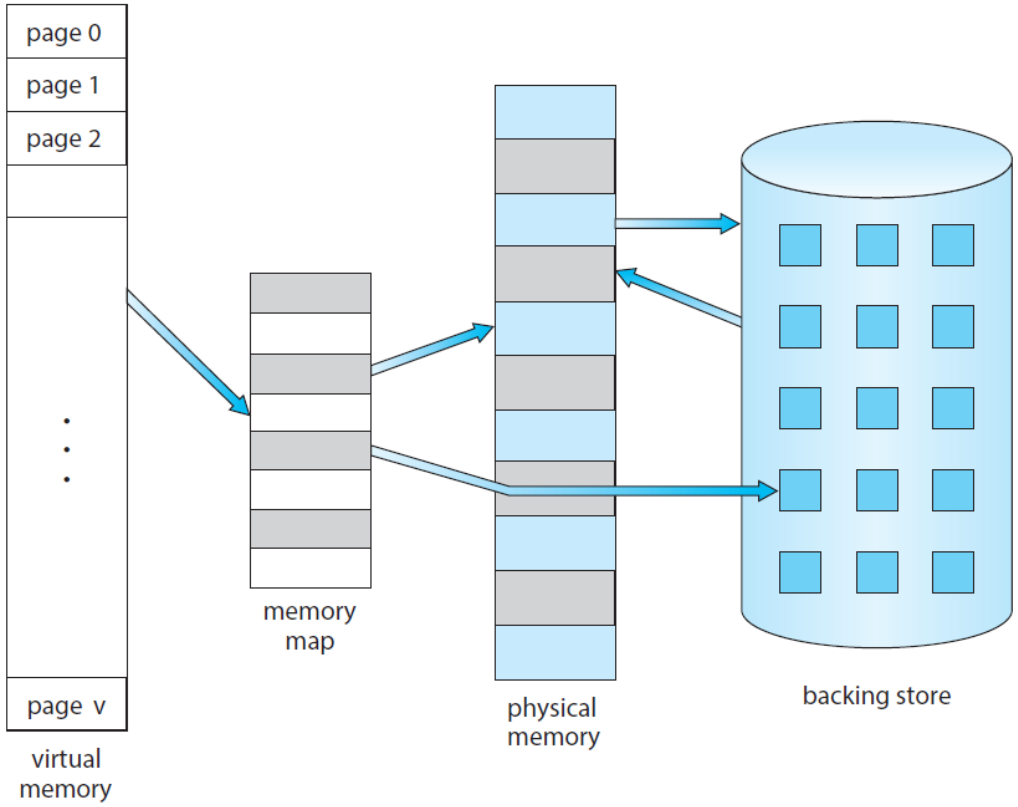
- Introductory Concepts
- Process Fundamentals
- IPC
- CPU Scheduling Algorithms
- Multithreading Concepts
- Synchronization
- Deadlock
- Memory Management

Outline

- Background
- Demand Paging
- Page Replacement
- Allocation of Frames
- Thrashing

Background

- Diagram showing virtual memory that is larger than physical memory

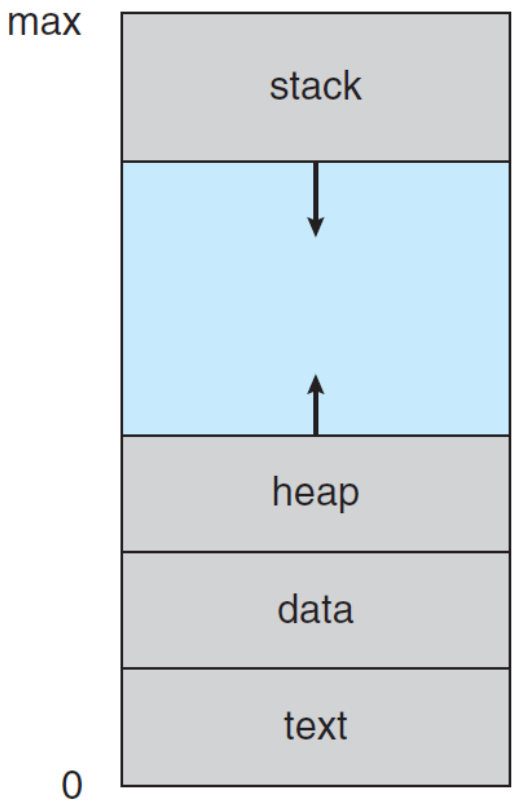


Benefits:

- A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space, simplifying the programming task.
- Because each program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput but with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap portions of programs into memory, so each program would run faster

Background

- Virtual address space of a process in memory

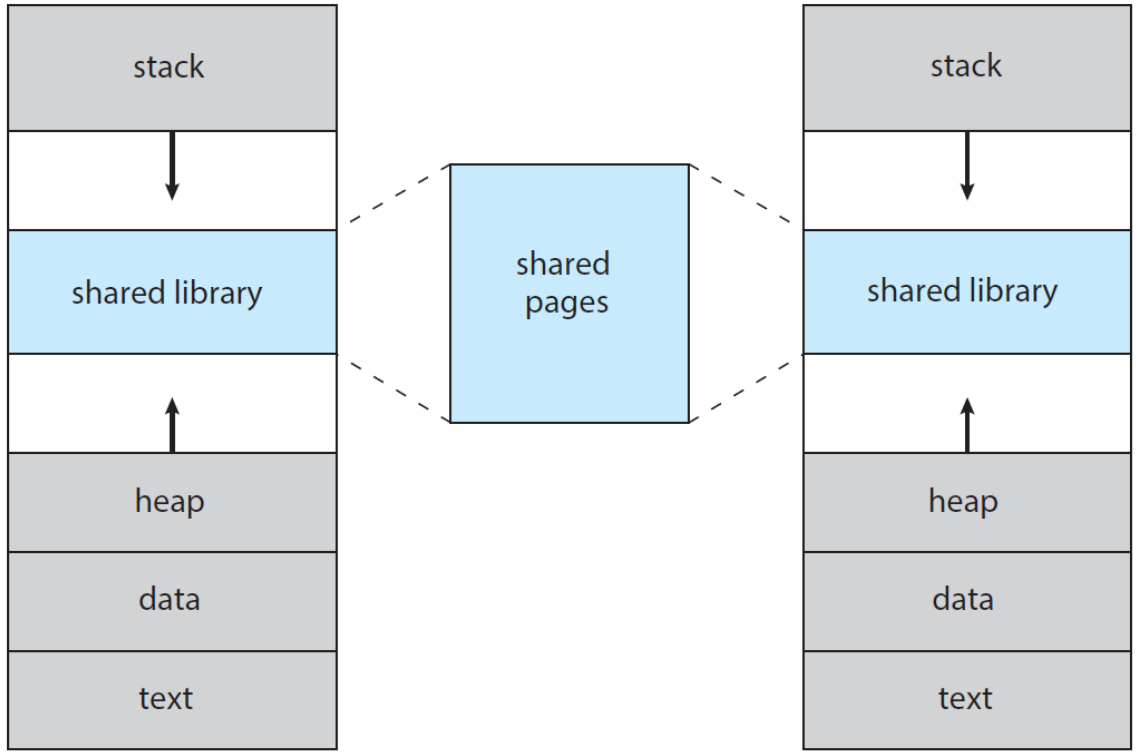


Virtual address spaces that include holes are known as **sparse address spaces**.

Using a sparse address space is beneficial because the holes can be filled as the stack or heap segments grow or if we wish to dynamically link libraries (or possibly other shared objects) during program execution.

Background

- Shared library using virtual memory

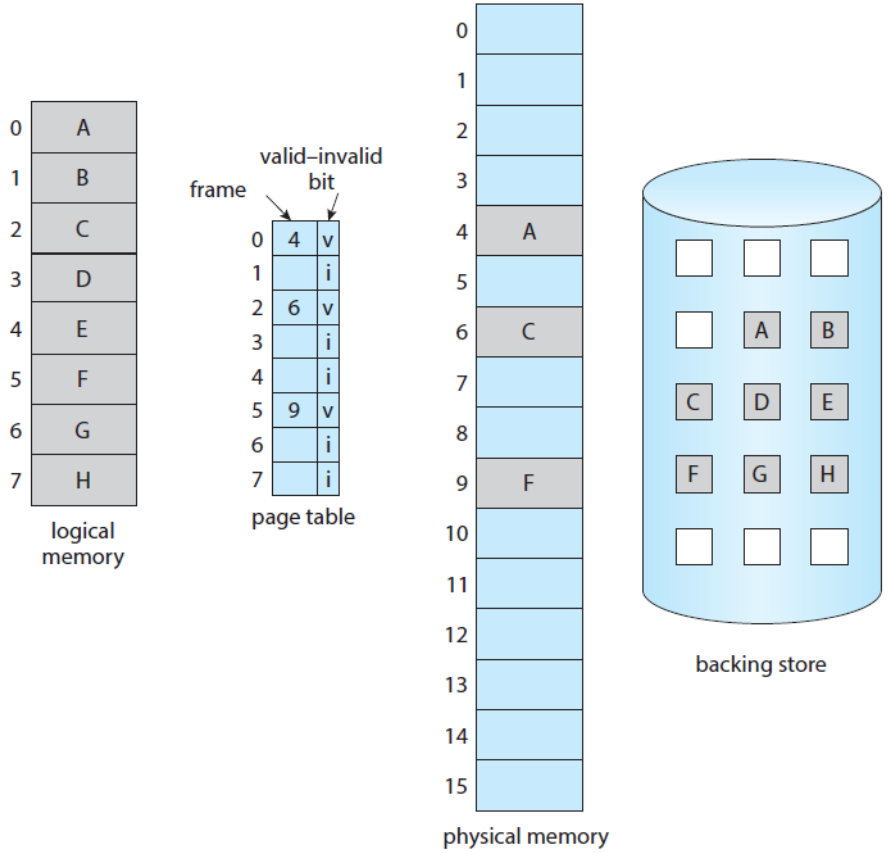


Benefits:

- System libraries such as the standard C library can be shared by several processes through mapping of the shared object into a virtual address space.
- Virtual memory allows one process to create a region of memory that it can share with another process.
- Pages can be shared during process creation with the fork() system call, thus speeding up process creation

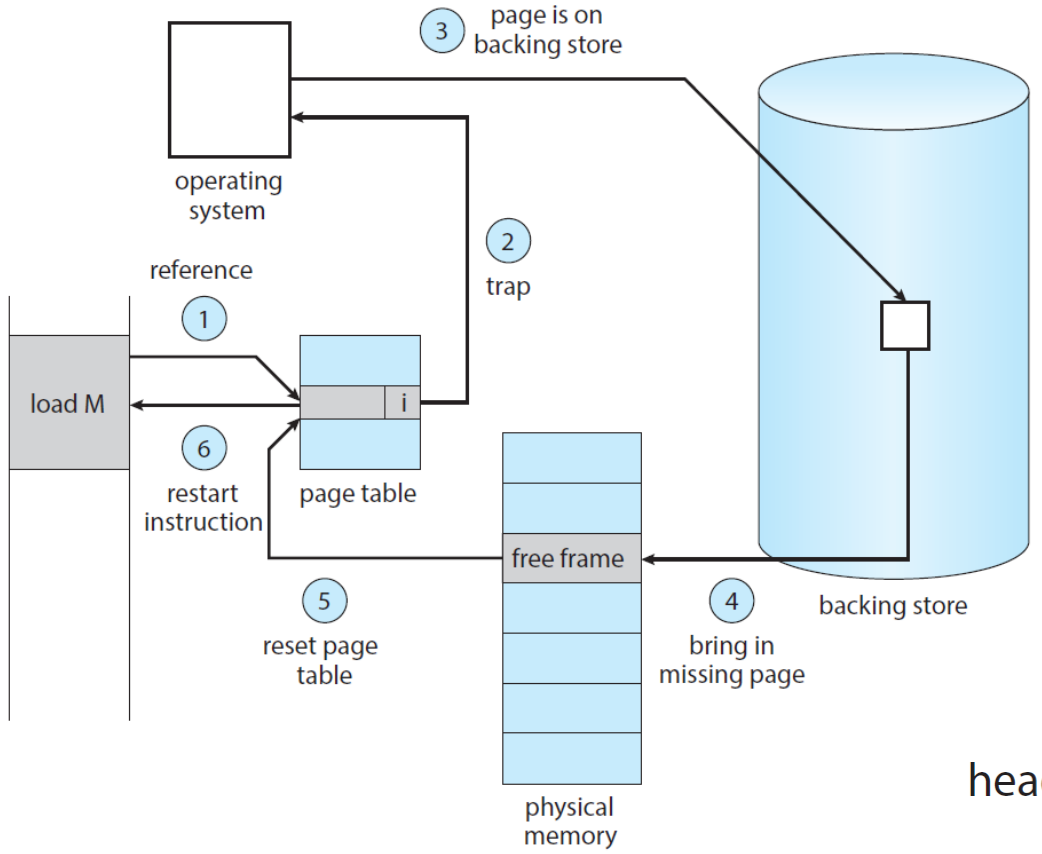
Demand Paging

- Load a page in memory only when it is needed
  - Possible Problem: Page table when some pages are not in main memory



Demand Paging

- Steps in handling a page fault



Demand Paging

- Performance of Demand Paging
  - Assume that
    - The memory-access time (*ma*) = 10 nanoseconds
    - The probability of a page fault = p ($0 \leq p \leq 1$)

effective access time = $(1 - p) \times ma + p \times$ page fault time

Demand Paging

- Performance of Demand Paging
  - A page fault causes the following sequence to occur:
    1. Trap to the operating system.
    2. Save the registers and process state.
    3. Determine that the interrupt was a page fault.
    4. Check that the page reference was legal, and determine the location of the page in secondary storage.
    5. Issue a read from the storage to a free frame:
       a) Wait in a queue until the read request is serviced.
       b) Wait for the device seek and/or latency time.
       c) Begin the transfer of the page to a free frame.
    6. While waiting, allocate the CPU core to some other process.

Demand Paging

- Performance of Demand Paging
  - A page fault causes the following sequence to occur:
    7. Receive an interrupt from the storage I/O subsystem (I/O completed).
    8. Save the registers and process state for the other process (if step 6 is executed).
    9. Determine that the interrupt was from the secondary storage device.
    10. Correct the page table and other tables to show that the desired page is now in memory.
    11. Wait for the CPU core to be allocated to this process again.
    12. Restore the registers, process state, and new page table, and then resume the interrupted instruction.

Demand Paging

- Performance of Demand Paging
  - Three major task components of the page-fault service time:
    1. Service the page-fault interrupt.
    2. Read in the page.
    3. Restart the process.

  With an average page-fault service time of **8 milliseconds*** and a memory access time of 200 nanoseconds, the effective access time in nanoseconds is

  effective access time = $(1 - p) \times (200) + p$ (8 milliseconds)

  $= (1 - p) \times 200 + p \times 8{,}000{,}000$

  $= 200 + 7{,}999{,}800 \times p.$

  - **effective access time is directly proportional to the page-fault rate**

If we want performance degradation to be less than 10 percent, we need to keep the probability of page faults at the following level:

$220 > 200 + 7{,}999{,}800 \times p,$

$20 > 7{,}999{,}800 \times p,$

$p < 0.0000025.$

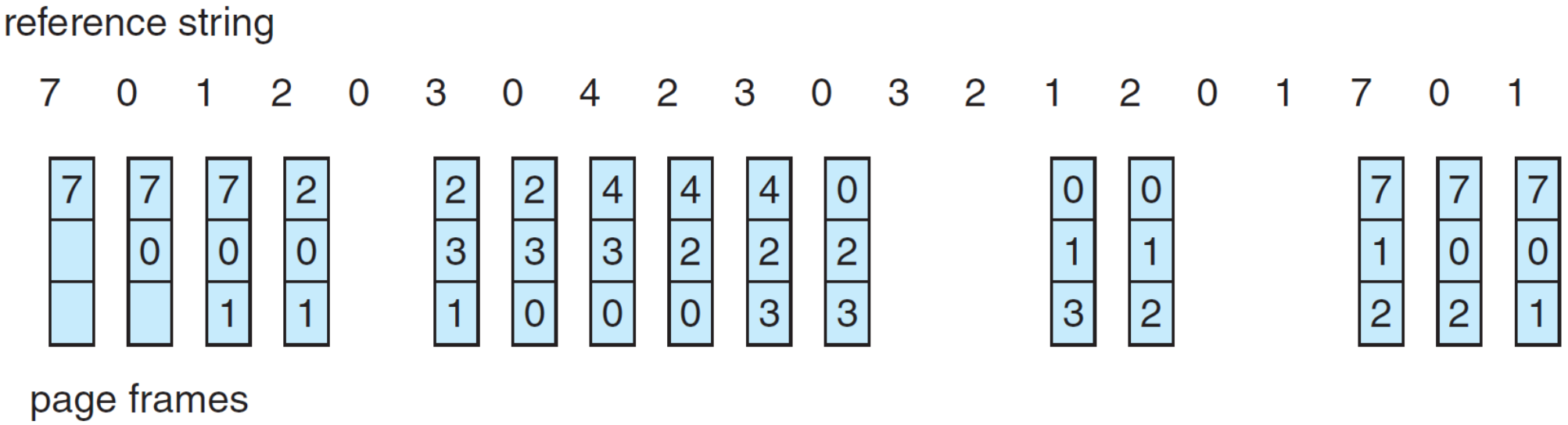
A typical hard disk has an average latency of 3 milliseconds, a seek of 5 milliseconds, and a transfer time of 0.05 milliseconds.

Page Replacement

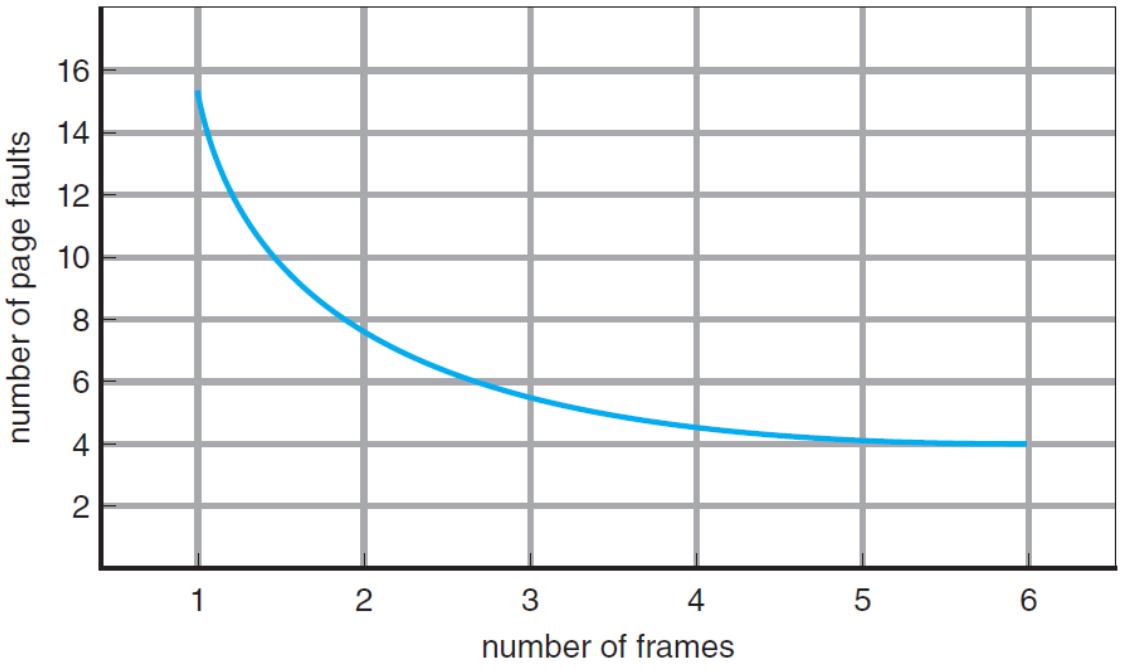
- FIFO Page-Replacement Algorithm
- Optimal Page-Replacement Algorithm
- LRU Page-Replacement Algorithm

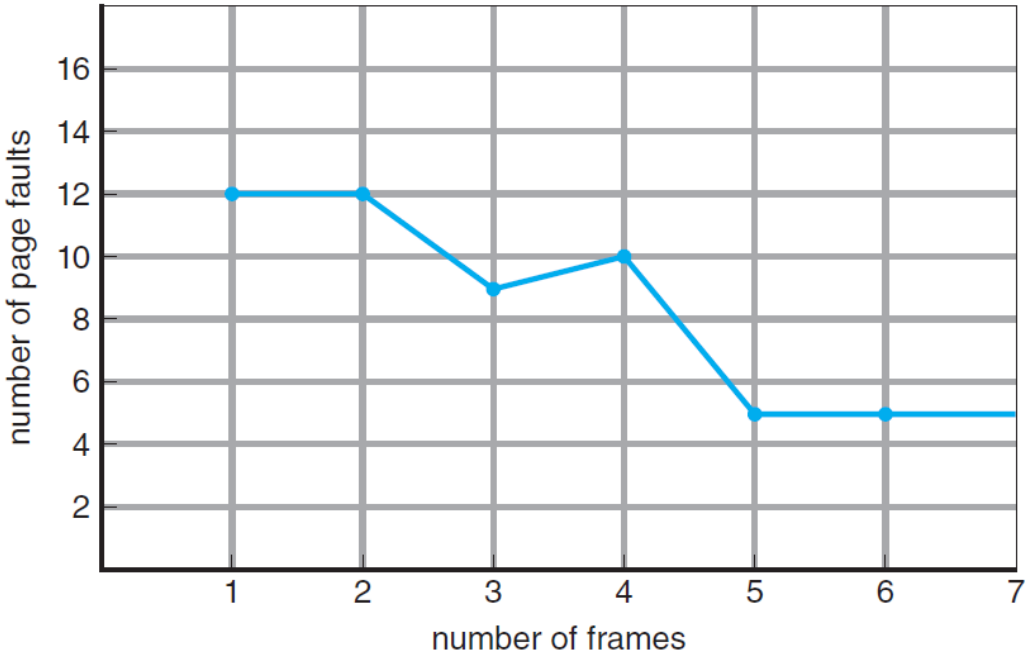
FIFO Page-Replacement Algorithm

- FIFO Page-Replacement Algorithm
  - Associates with each page the time when that page was brought into memory.
  - When a page must be replaced, the oldest page is chosen



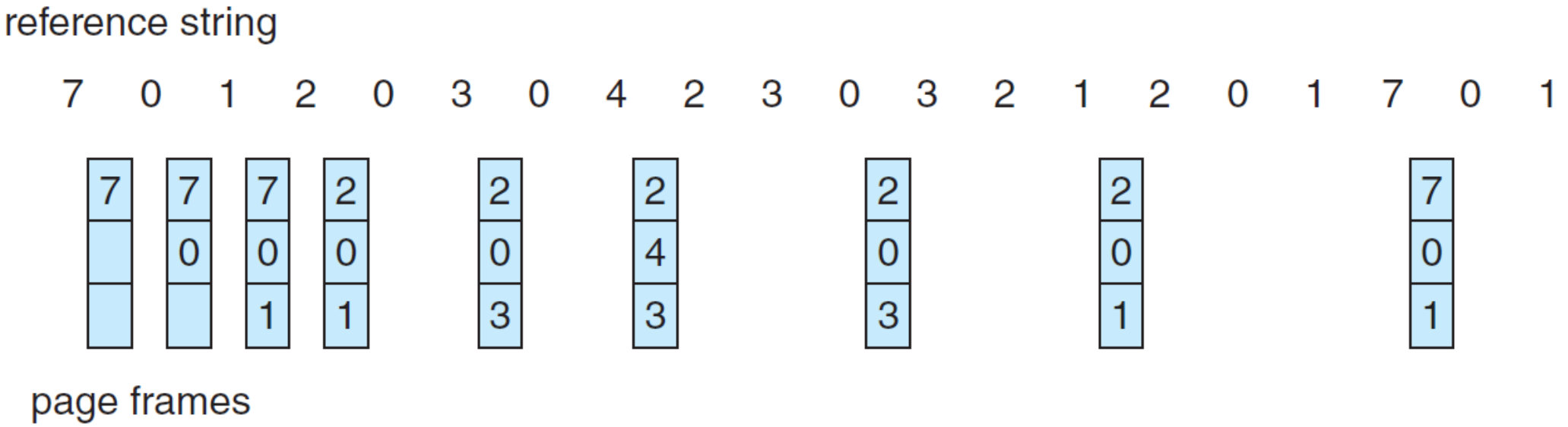
Belady's Anomaly





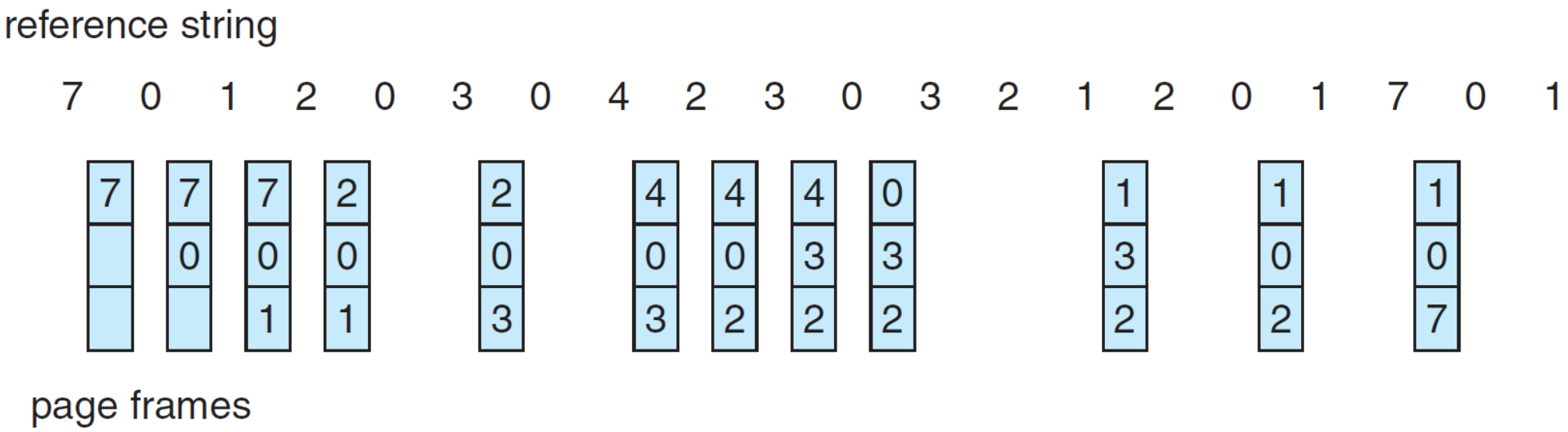
Optimal Page-Replacement Algorithm

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



LRU Page-Replacement Algorithm

- Replace the page that has not been used for the longest period of time

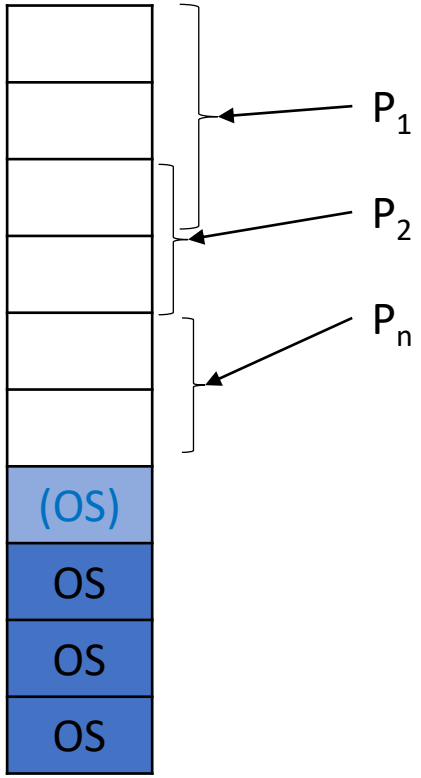


Other Page-Replacement Algorithm

- Counting-Based Page Replacement
  - The **least frequently used** (LFU) page-replacement algorithm
  - The **most frequently used** (MFU) page-replacement algorithm

Allocation of Frames

- Minimum Number of Frames



Allocation of Frames

- Allocation Algorithms
  - Equal Allocation
    - Split *m* frames among *n* processes is to give everyone an equal share, m/n frames

      Example:

      If there are 93 frames and 5 processes, each process will get 18 frames.

      [The 3 leftover frames can be used as a free-frame buffer pool]

Allocation of Frames

- Allocation Algorithms
  - Proportional Allocation
    - Allocate available memory to each process according to its size.

Let the size of the virtual memory for process p_i be s_i, and define

$$S = \Sigma s_i.$$

Then, if the total number of available frames is m, we allocate a_i frames to process p_i, where a_i is approximately

$$a_i = s_i/S \times m.$$

Example:

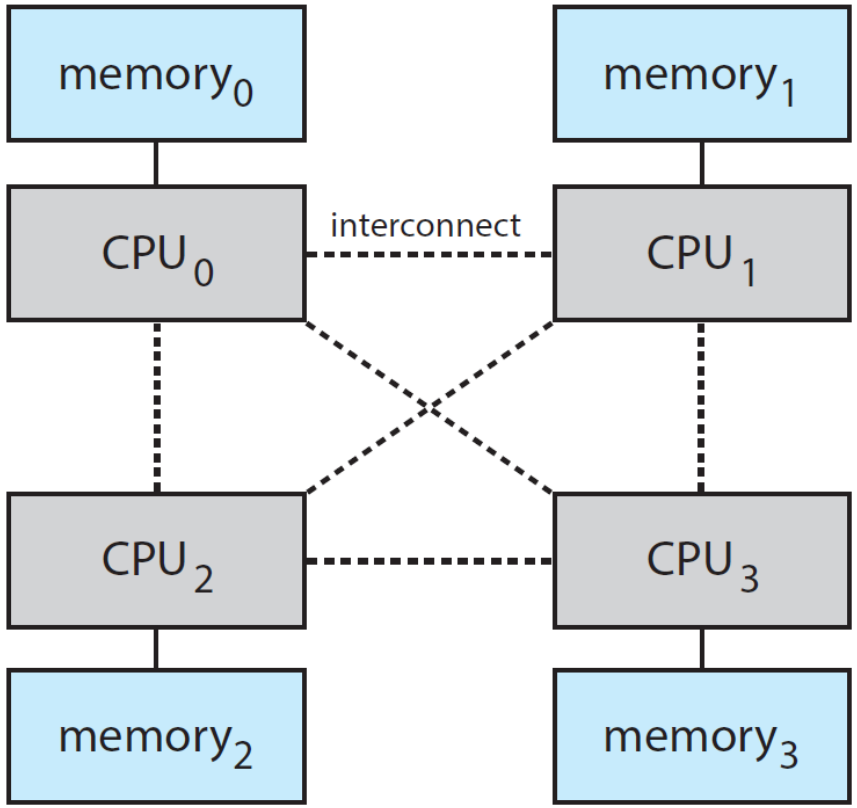
Consider a system with a 1-KB frame size. If a small student process of 10 KB and an interactive database of 127 KB are the only two processes running in a system with 62 free frames.

For Student Process → $10/137 \times 62 \approx$ **4**

For Interactive Database → $127/137 \times 62 \approx$ **57**

Allocation of Frames

- Non-Uniform Memory Access (NUMA)

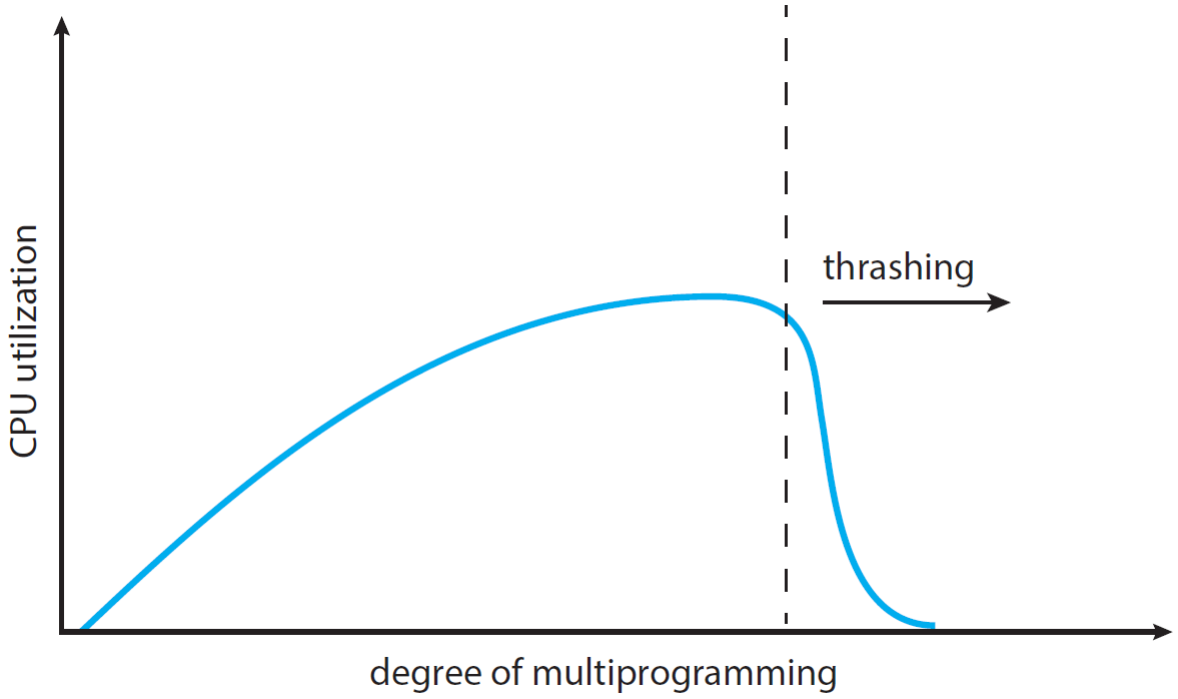


Access time for each of the memory units by each CPU is not exactly the same.

It is required to have memory frames allocated "as close as possible" to the CPU on which the process is running. (The definition of close is "with minimum latency," which typically means on the same system board as the CPU).

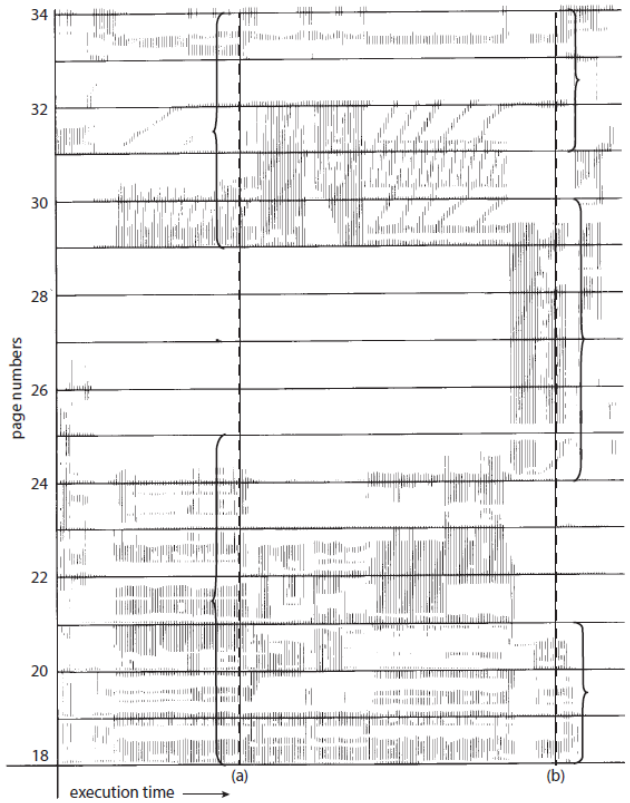
Thrashing

- Cause of Thrashing

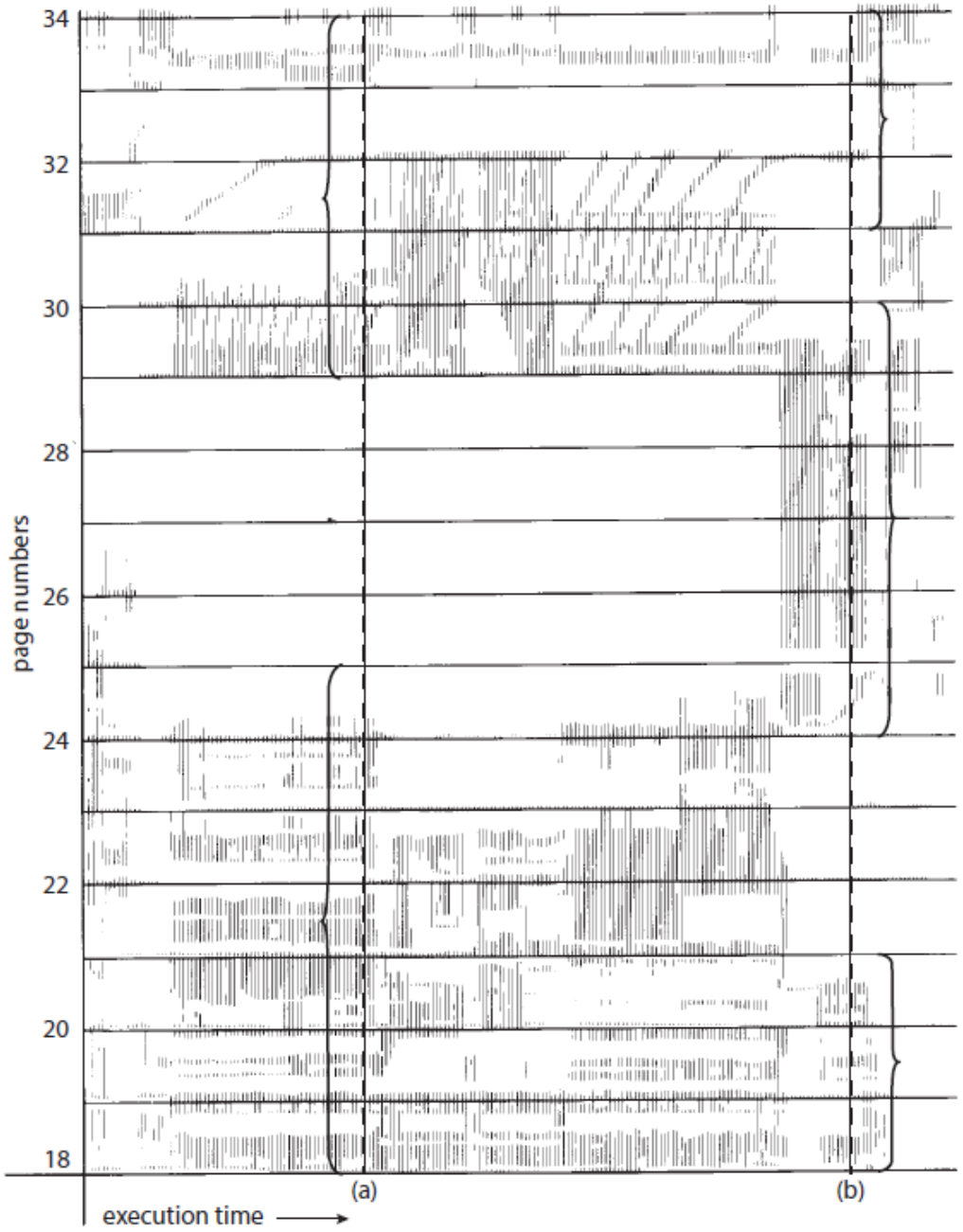


Thrashing

- limiting the effects of thrashing – Local Replacement Algorithm



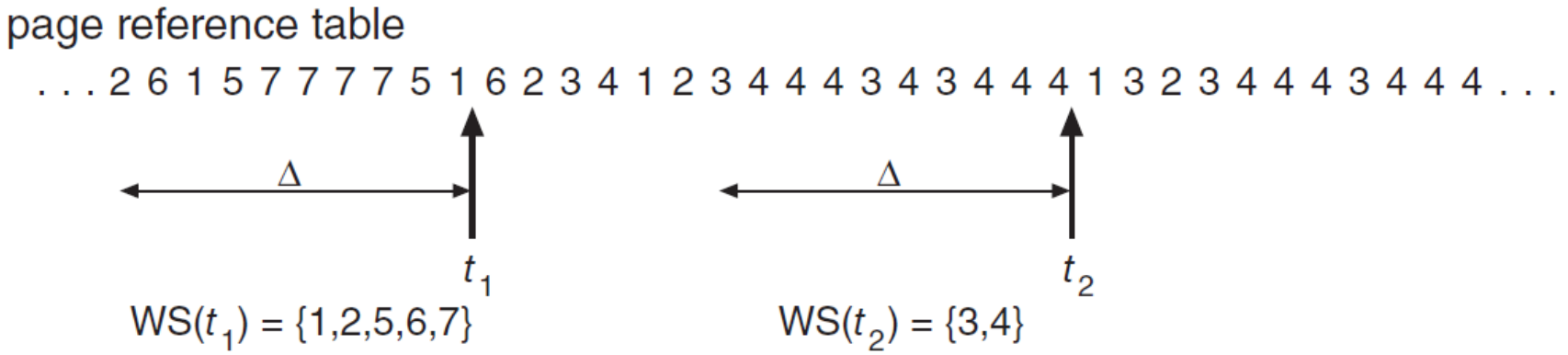
Th



VIT-AP UNIVERSITY

Thrashing

- Working-Set Model



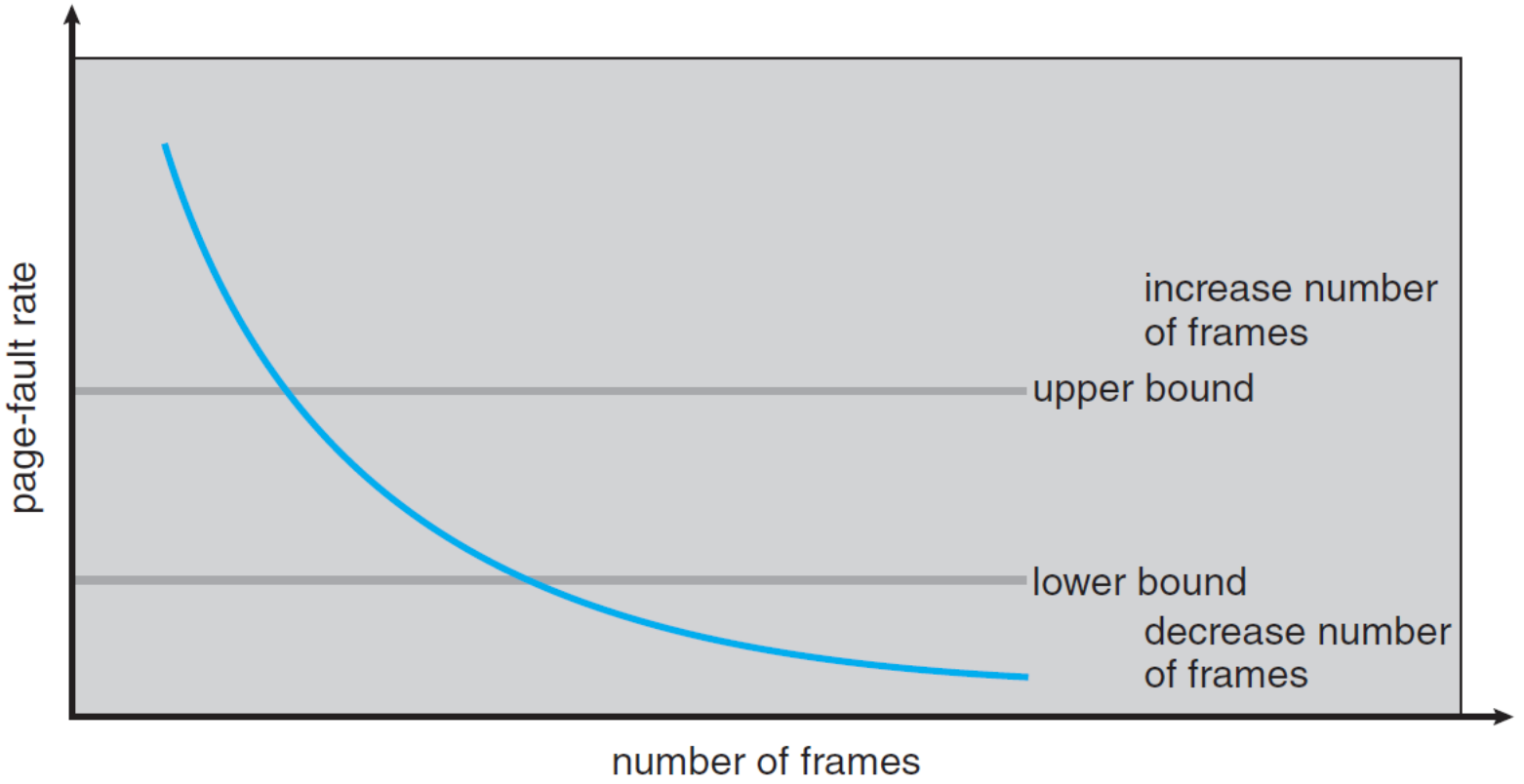
$$D = \sum WSS_i,$$

D is the total number of frames.

If D > m, then thrashing

Thrashing

- Page-Fault Frequency – A direct approach to solve thrashing issue



Memory Compression

- An alternative to paging **<Self Study>**

Reference

- Abraham Silberschatz , Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison Wesley, 10th edition, 2018
  - Chapter 10: Section 10.1 – 10.6

Next

- Storage management and security

VIT-AP UNIVERSITY

Thank You