Operating Systems
Course Code: 71203002004
Page Replacement Strategies

by -
Asst. Prof. Minal Rajwar

Page Replacement Algorithms

Page replacement algorithms are techniques used in OS to manage memory efficiently when the physical memory is full.

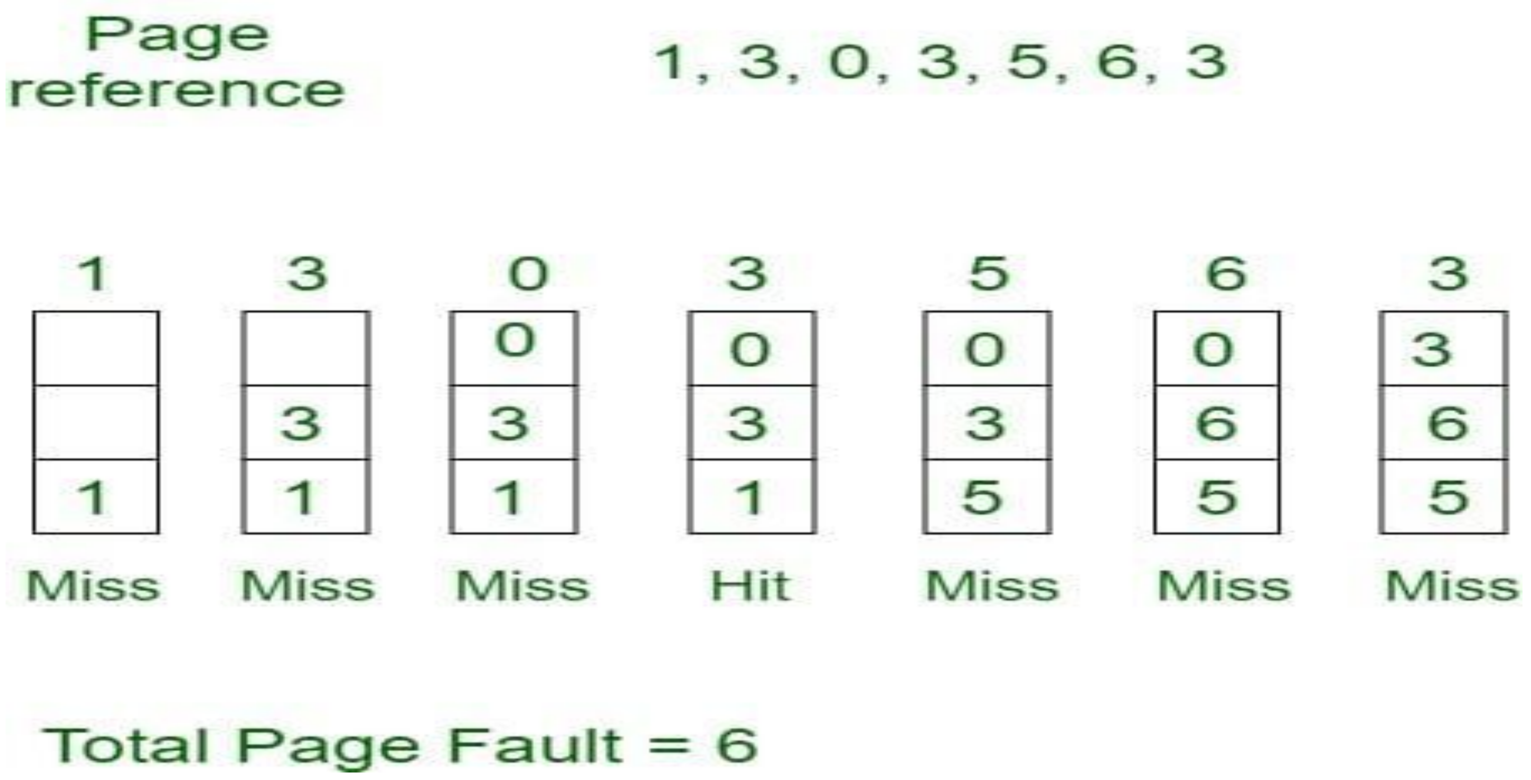
Common Page Replacement Techniques:

- First In First Out (FIFO)
- Optimal Page Replacement
- Least Recently Used (LRU)
- Most Recently Used (MRU)

FIFO - First In First Out

a. The oldest page in memory is replaced when a new page is needed.
b. Uses a queue structure - new pages are added to the tail, and the page at the head is replaced when needed.
c. Is simple to implement.
d. Doesn’t consider page usage frequency - might replace frequently used pages.

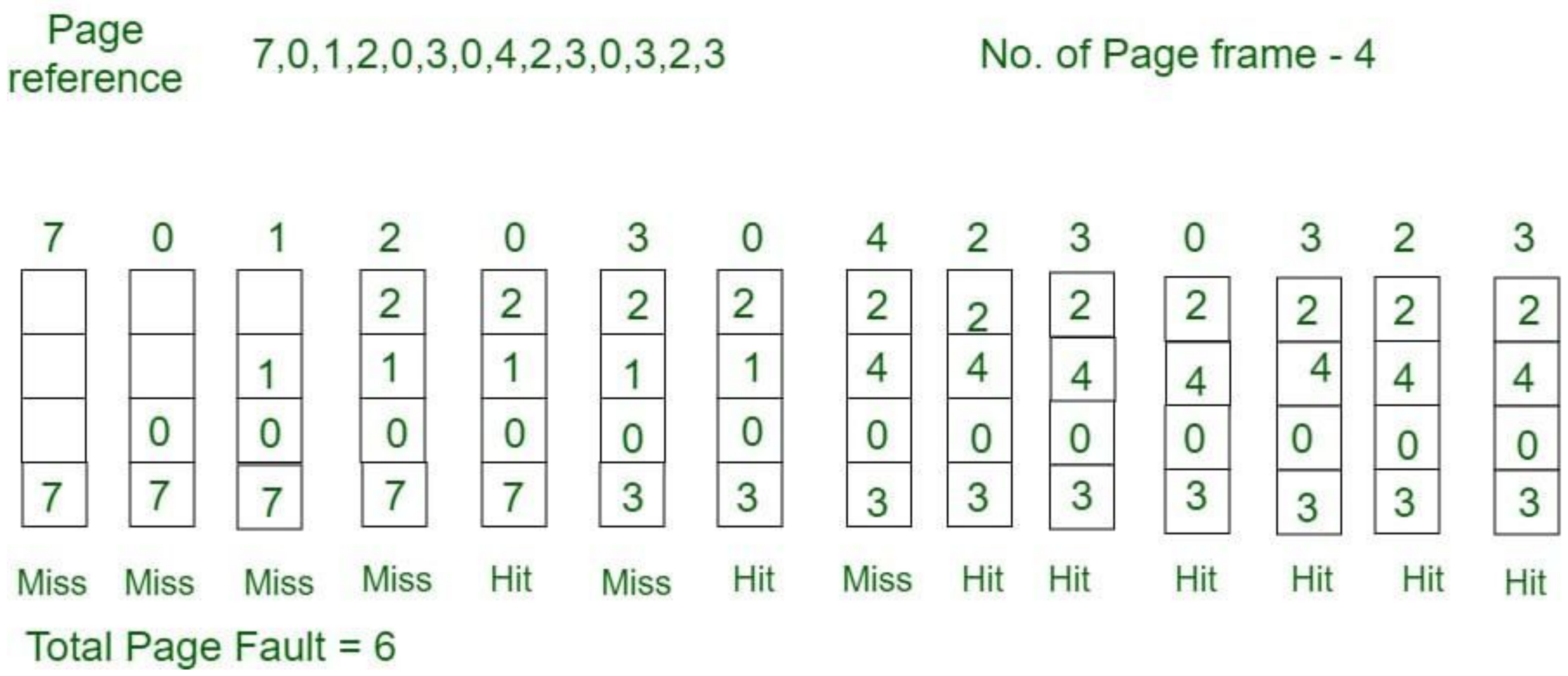
Example: Consider page reference string 1,3,0,3,5,6,3 with 3-page frames. Find the number of page faults using FIFO Page replacement Algorithm,



Optimal Page Replacement

a. Pages are replaces which would not be used for the longest duration of time in the future.
b. Acts as the theoretical benchmark for other algorithms.
c. Unrealistic in practice since it requires future knowledge.
d. How it works:
    i. When a page fault occurs, check future page references.
    ii. replace the page that either:
        1. will never be used again, OR
        2. Has the longest wait until its next use.

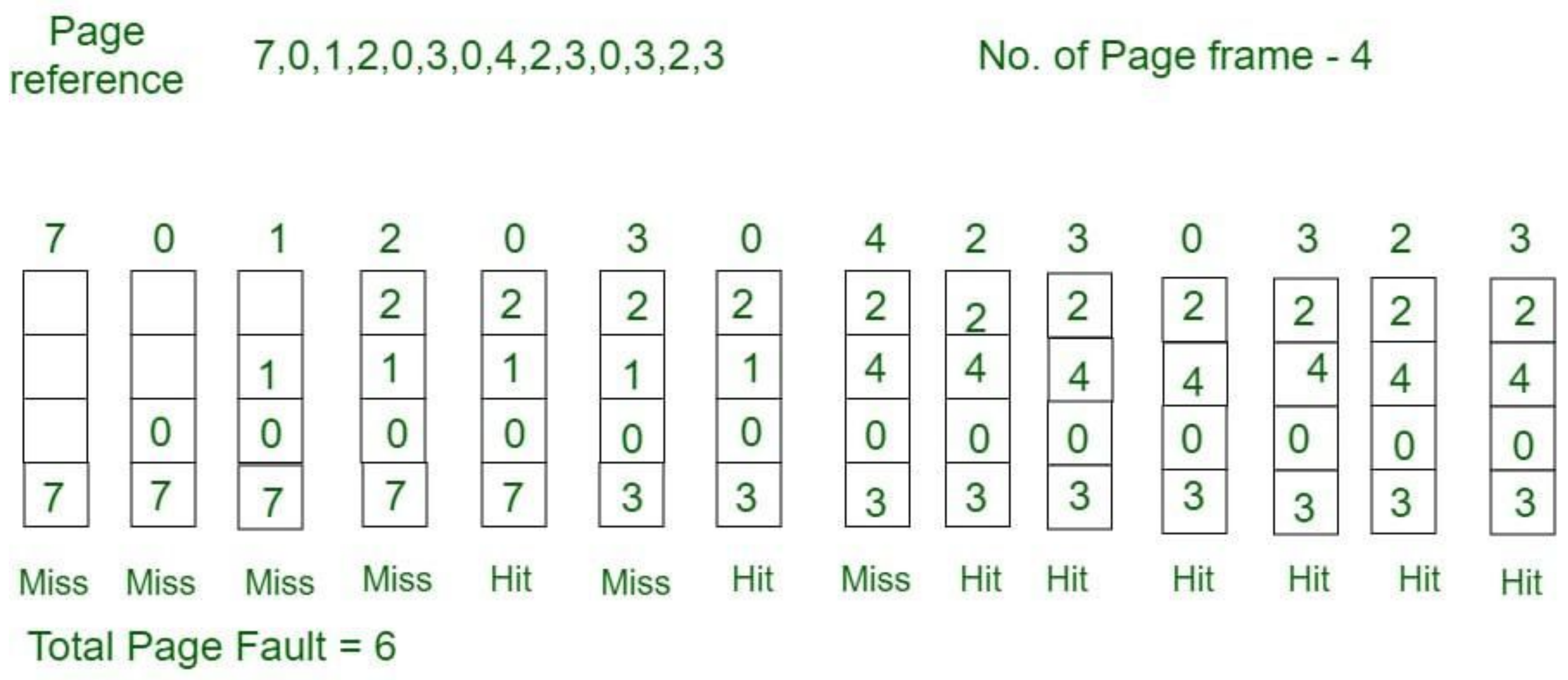
Example: Consider page reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,3 with 4 -page frames. Find the number of page faults using Optimal Page replacement Algorithm,



LRU - Least Recently Used

a. The page that hasn’t been used for the longest time is replaced.
b. Tracks when each page was last accessed and replaces the one with the oldest access time.
c. Better performance than FIFO as it considers page usage patterns.
d. More complex implementation as it requires tracking access times.

Example: Consider page reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,3 with 4 page frames. Find the number of page faults using LRU Page replacement Algorithm,

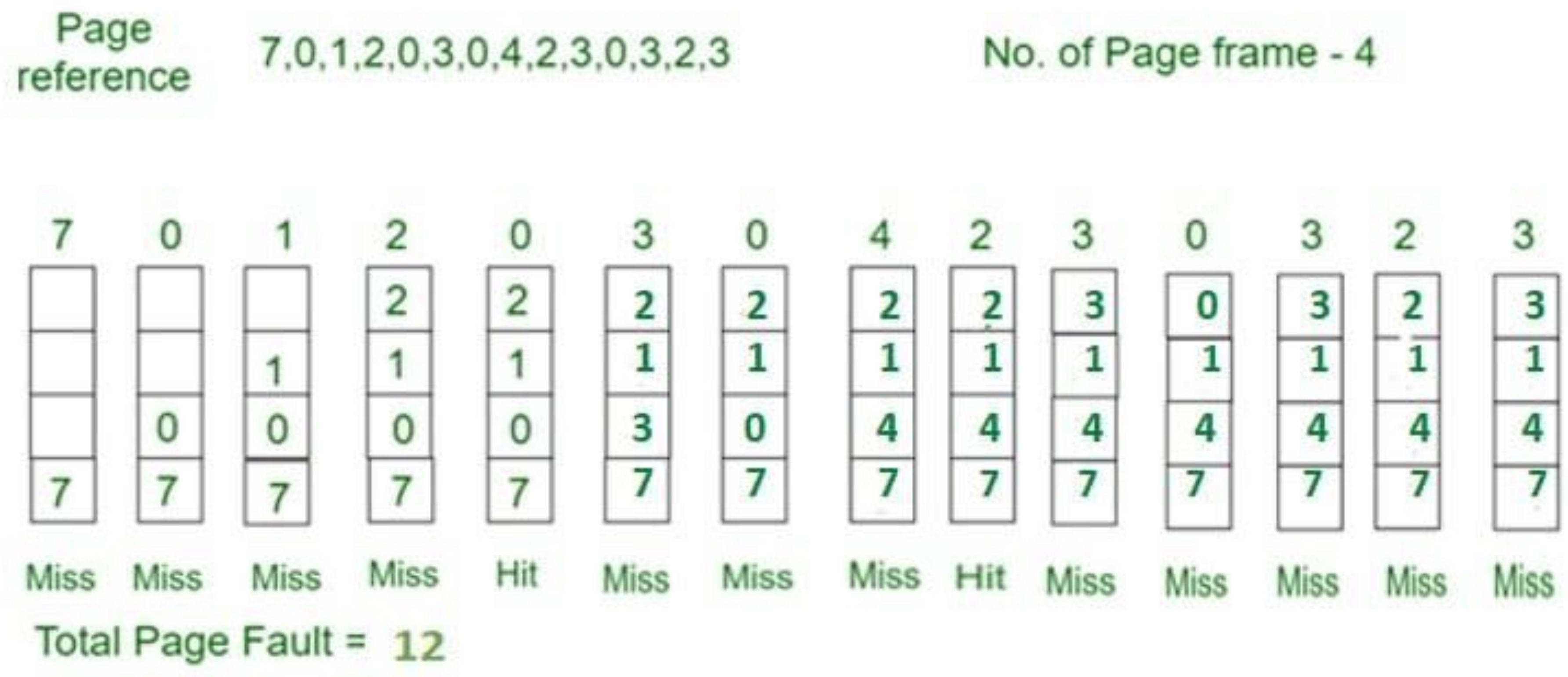


Here LRU has same number of page fault as optimal but it may differ according to question.

MRU - Most Recently Used

a. Replaces most recently used page when a new page is needed.
b. Fast for certain access patterns (e.g., when recent pages won't be needed soon).
c. Can suffer Belady's Anomaly
    i. Increasing available memory can sometime increase page faults instead of reducing them.

Example: Consider page reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,3 with 4 page frames. Find the number of page faults using MRU Page replacement Algorithm,



Demand Paging

It is a memory management technique where the OS loads only the required parts (pages) of a program into RAM when needed , rather than loading the entire program at startup.

How It Works:

- ★ Program runs > Only essential pages are loaded initially.
- ★ Page Fault > If the CPU requests a page not in RAM, the OS:
  - ○ Pauses the program.
  - ○ Fetches the missing page from disk.
  - ○ Loads it into memory (replacing another page if needed).
  - ○ Resumes execution.

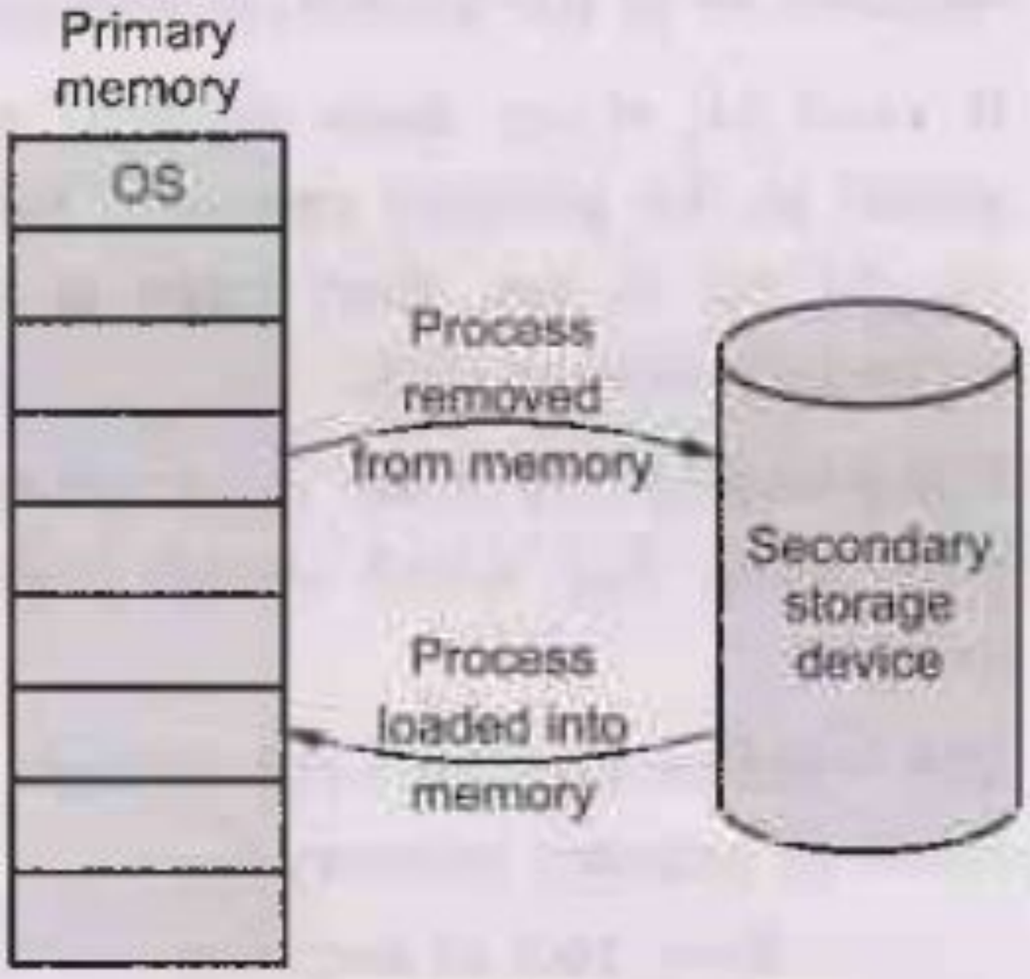


Fig. 4.10.1 Demand paging

Pros

★ Saves Memory (only active pages are loaded).

★ Faster startup (no need to load entire programs).

★ Supports large programs (even bigger than RAM).

Cons

★ Page faults slow execution (disk access is slower than RAM).

★ Thrashing (if too many page faults occur, system performance drops).

Performance of Demand Paging

Demand paging improves memory efficiency but has performance trade-offs. Key factors affecting performance:

- **Page Fault Rate (PFR)**
  - frequency at which a requested page is not in RAM (triggering a disk fetch).
  - LOW PFR > Smooth performance.
  - High PFR > Thrashing (system wastes time swapping pages).

- **Effective Access Time (EAT)**
  - EAT = (1 - p) x Memory Access Time + p x Page Fault Time

      - p = Page fault probability
      - Memory Access Time = time to access RAM (in nanoseconds).
      - Page Fault time = Time to handle page faults (including disk I/O, swapping, etc.)

- Example
  - If p = 10%, RAM access = 100ns , Disk access = 10ms: 10,000,000 ns

Apply the Formula:

$$EAT = (1-0.1)\times 100+0.1\times 10{,}000{,}000$$

$$EAT = 0.9\times 100+0.1\times 10{,}000{,}000$$

$$EAT = 90+1{,}000{,}000=1{,}000{,}090 \text{ ns}$$

- **Optimizations to Improve Performance**
  - Working Set Model : Keep actively used pages in RAM.
  - Prepaging ; Load anticipated pages in advance.
  - Good Page Replacement Algorithm (e.g., LRU over FIFO).
  - Fast Storage : SSDs reduce page fault penalty vs. HDDs.

- **Thrashing**
  - Too many processes > high page faults > system spends more time swapping then executing.
  - Solution:
    - Limit multiprogramming (fewer concurrent processes).
    - Increase RAM or optimize memory usage.

Discussion & Revision

- What does demand paging load into memory only when needed?
- Which algorithm replaces the page not used recently?
- What occurs when a needed page is not in RAM?
- What is the slowdown caused by excessive page faults called?
- Which storage device reduces page fault delays compared to HDDs?