OPERATING SYSTEMS

Subject code : CSC 404

Subject In-charge

Nidhi Gaur
Assistant Professor
email: nidhigaur@sfit.ac.in

Module 4 : Memory management
Topic: Virtual memory

Objectives

- To describe the benefits of a virtual memory system
- To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames

Background

Virtual memory

Only part of the program needs to be in memory for execution

- Logical address space can therefore be much larger than physical address space
- Allows address spaces to be shared by several processes

Virtual memory can be implemented via:

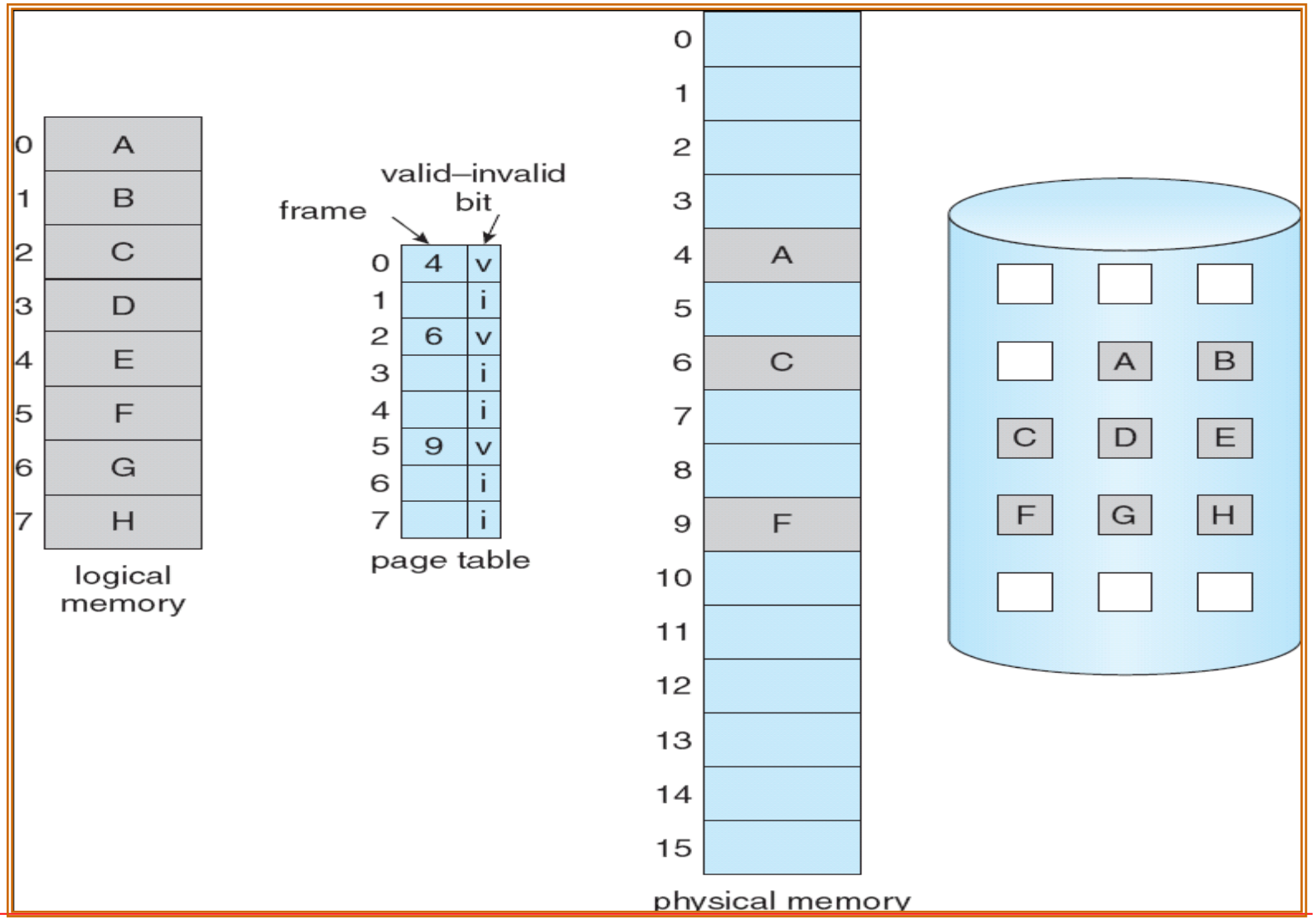
- Demand paging
- Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

  - Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- Lazy swapper – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

Page Table When Some Pages Are Not in Main Memory



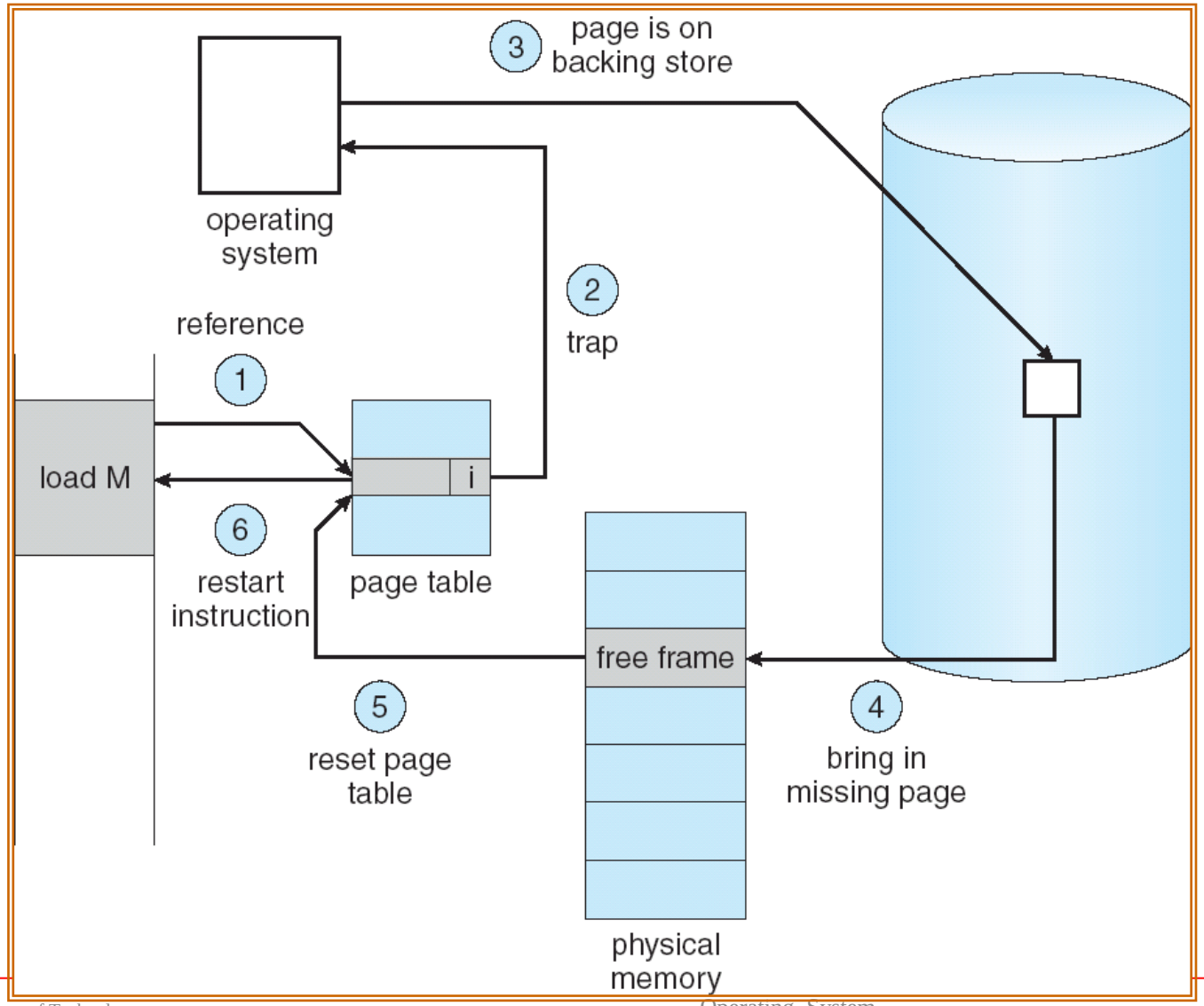
Page Fault

- If there is a reference to a page, first reference to that page trap to operating system:

  page fault

1. Operating system looks at table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = v
6. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



Pure demand paging

- In extreme case we could start executing a process with no page in memory.
- When OS sets the pointer to first instruction of the process, the process immediately faults for the page.
- The process continues to execute faulting as necessary until every page that it needs is in memory. Now it can execute with no more faults.
- This scheme is pure demand paging.
- Never bring a page into memory until it is required.

What happens if there is no free frame?

- Page replacement

find some page in memory, but not really in use, swap it out

  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults

- Same page may be brought into memory several times

Basic Page Replacement

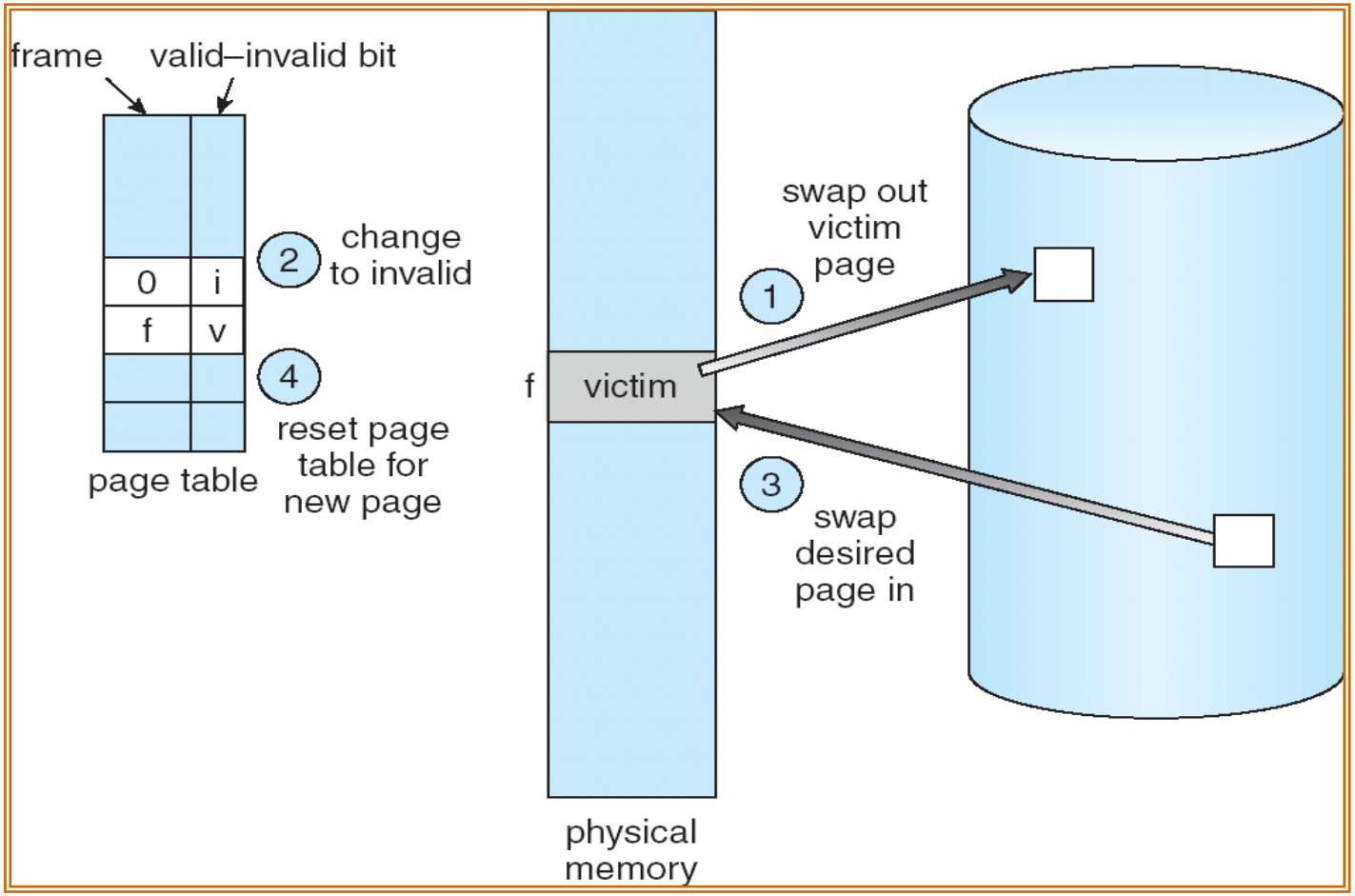
1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame

3. Bring the desired page into the (newly) free frame; update the page table.

4. Restart the process

Page Replacement



Page Replacement

- If no frames are free, two page transfers are required.
- This doubles the page fault service time and increases the effective access time accordingly.
- We can reduce this overhead by using a modify bit(dirty bit).
- We can avoid writing the page to the disk that has not been modified since it was read into memory, it reduces I/O time by one half if the page is not modified.

Page Replacement Algorithms

- Want lowest page-fault rate

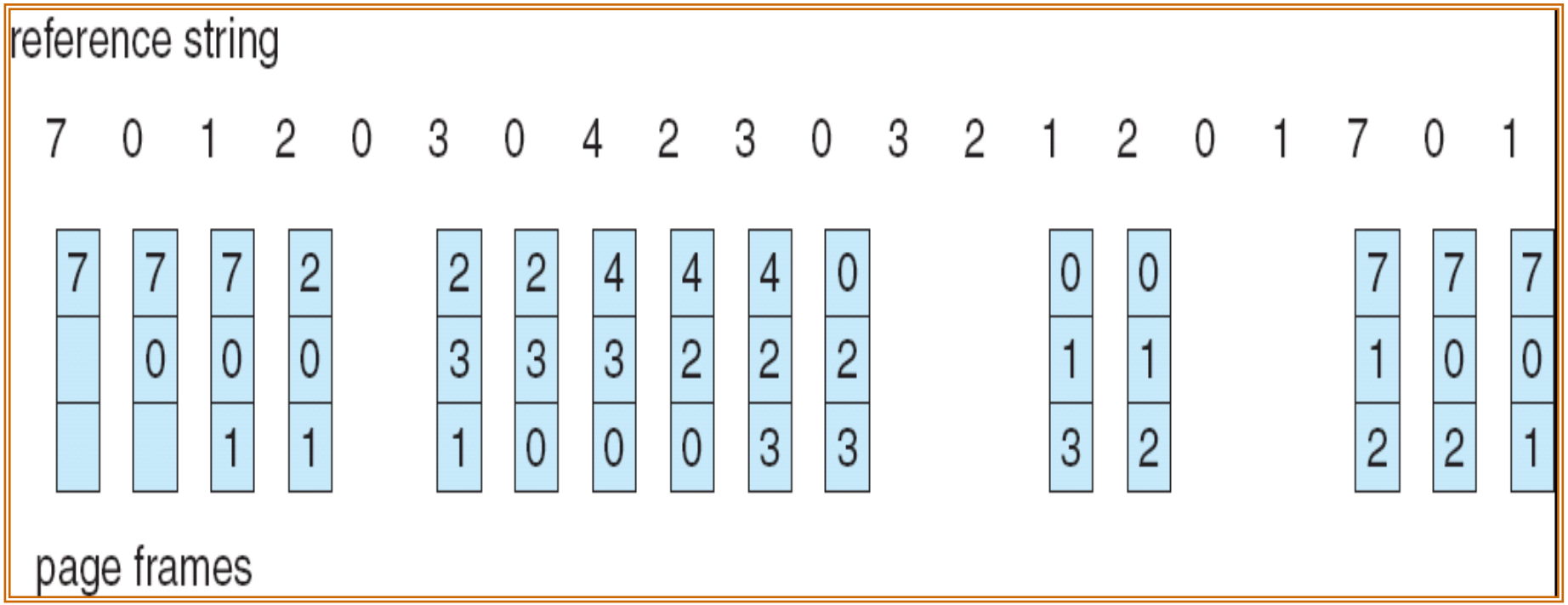
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

7, 0, 1, 2, 0, 3, 0 , 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0,1

FIFO

- The oldest page among all pages in memory is used for page replacement.
- OS attaches timestamp with page while storing in memory.
- Easiest way is to store pages in FIFO queue.
- The page at the head of the queue is paged out first and new page will be inserted at the tail.

FIFO Page Replacement



15 page faults

First-In-First-Out (FIFO) Algorithm

Reference string:

5, 0, 2, 1, 0, 3, 0, 2, 4, 3, 0, 3, 2, 1, 3, 0, 1, 5

for 3 frames and 4 frames using FIFO. Find out the number of page faults.

Ans:

for 3 frames: 15 page faults

for 4 frames: 11 page faults

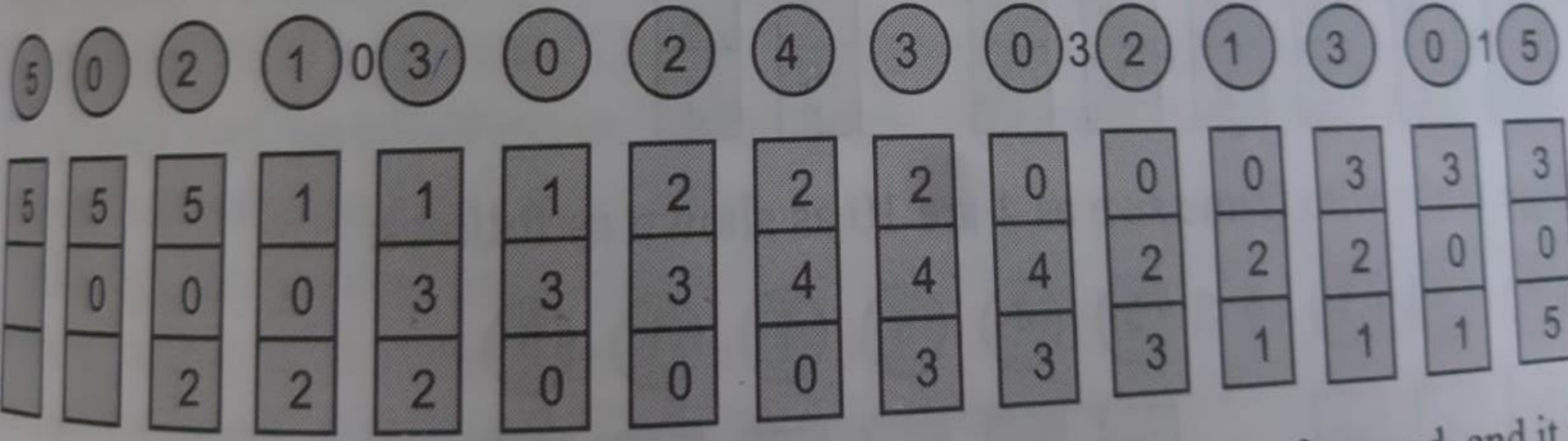
with some examples.

Example 11.6

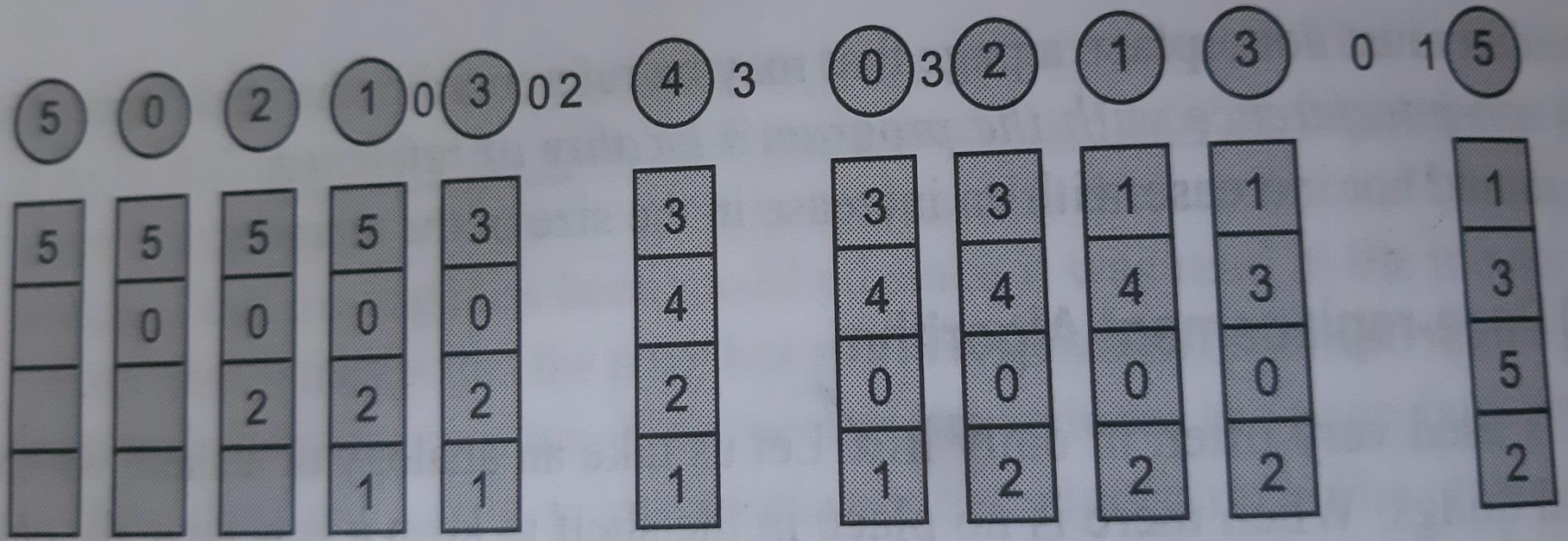
Calculate the number of page faults for the following reference string using FIFO with frame size as 3.

5 0 2 1 0 3 0 2 4 3 0 3 2 1 3 0 1 5

Solution



Initially, all the three frames are empty. Page number 5 is first referenced, and it
After handling the page fault, the page is brought into the memory in one of the fra
Next, Page number 1 is



faults decreases as discussed earlier that the number of page faults will decrease in the number of page frames.

Example 11.8

Calculate the [illegible] using

First-In-First-Out (FIFO) Algorithm

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 for 3 frames and 4 frames. Find out the number of page faults.

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

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

- 4 frames

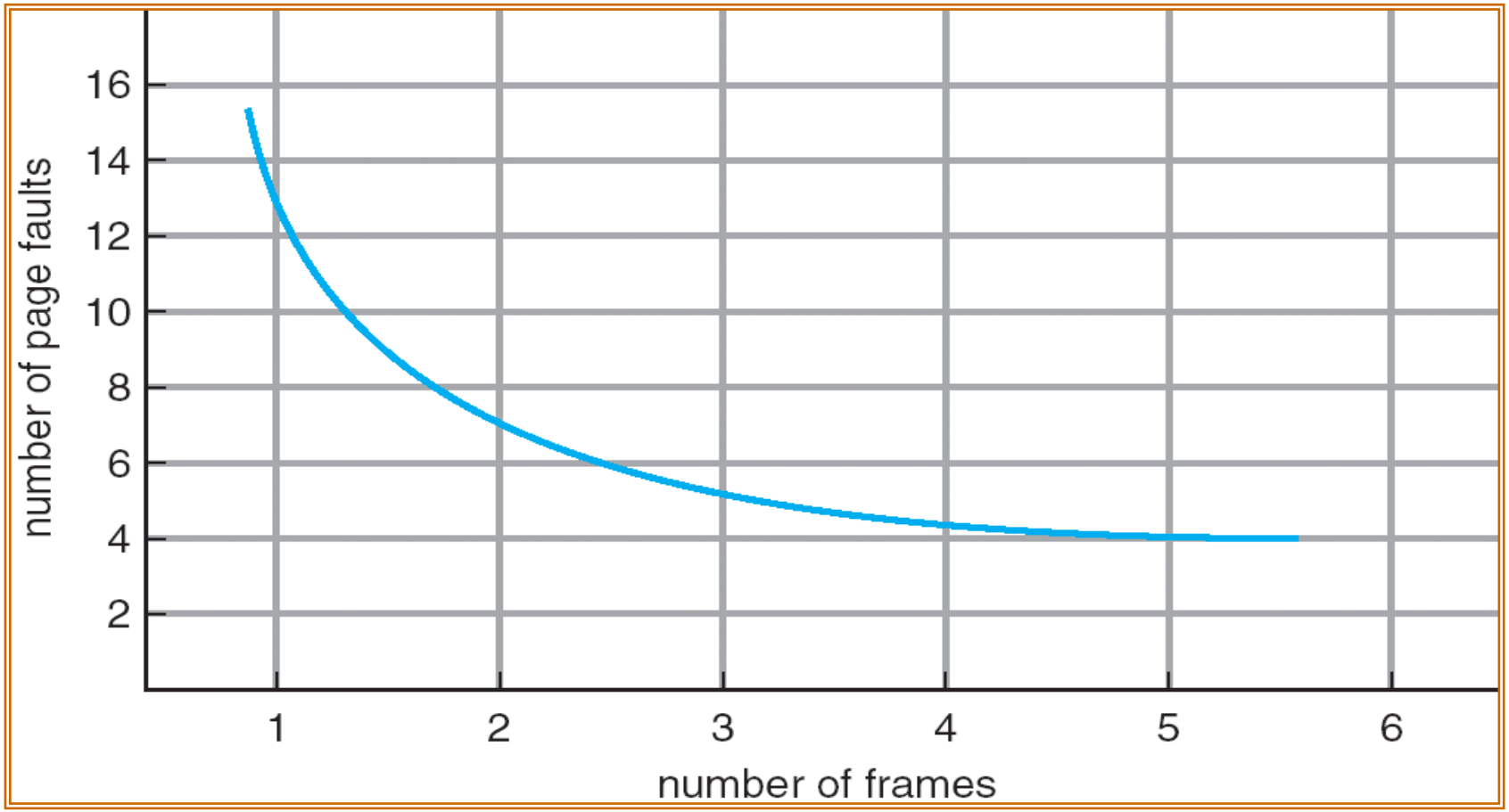
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

First-In-First-Out (FIFO) Algorithm

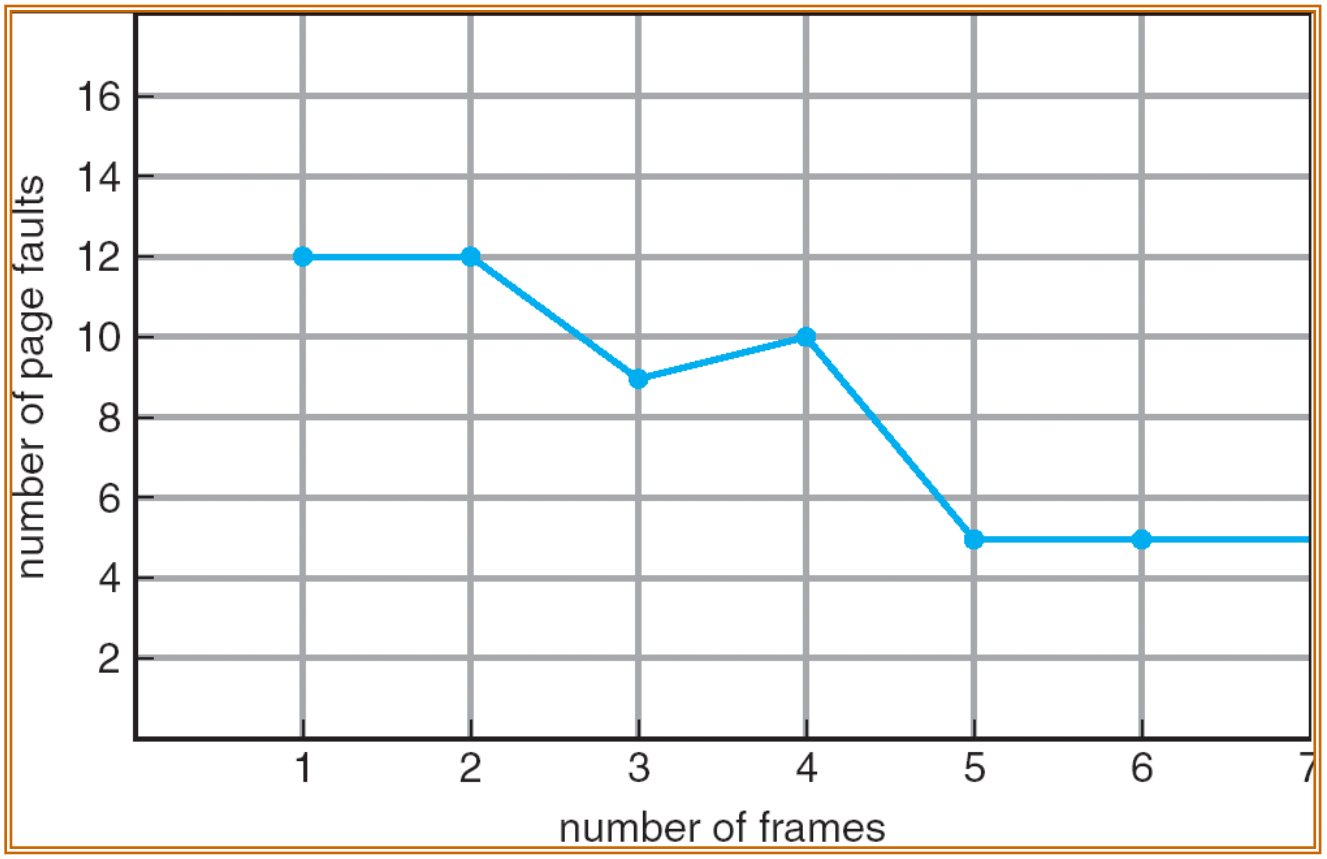
Belady's **anomaly** is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns.

This phenomenon is commonly experienced when using the First in First Out (FIFO) page replacement algorithm.

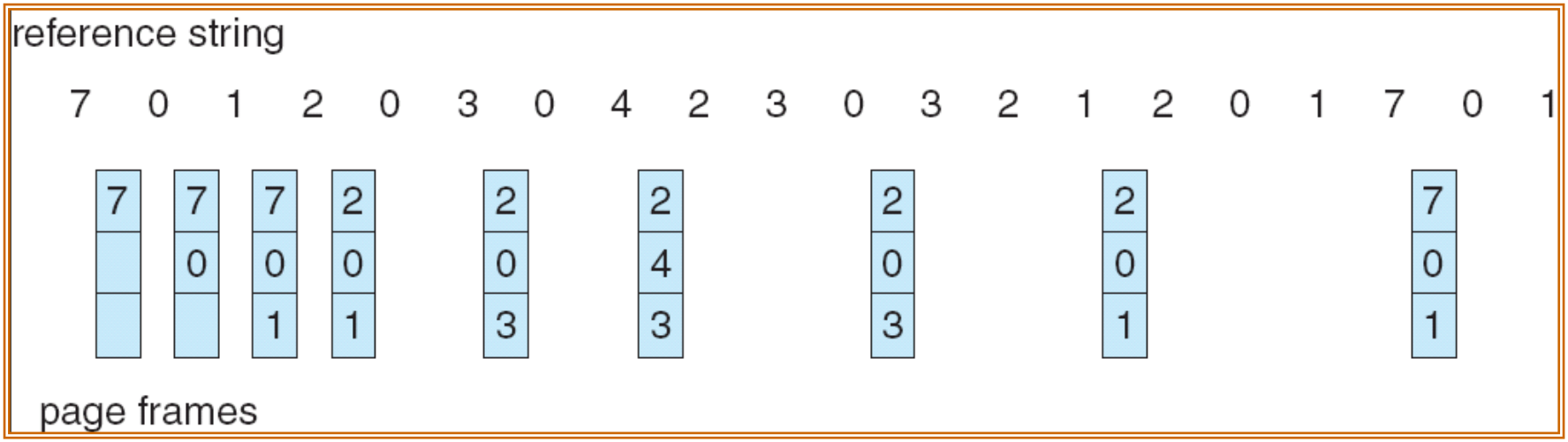
Graph of Page Faults Versus The Number of Frames



FIFO Illustrating Belady's Anomaly



Optimal Page Replacement



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

9 page faults

Optimal Algorithm

Reference string:

5, 0, 2, 1, 0, 3, 0, 2, 4, 3, 0, 3, 2, 1, 3, 0, 1, 5

for 3 frames using Optimal algorithm. Find out the number of page faults.

Ans:

for 3 frames: 9 page faults

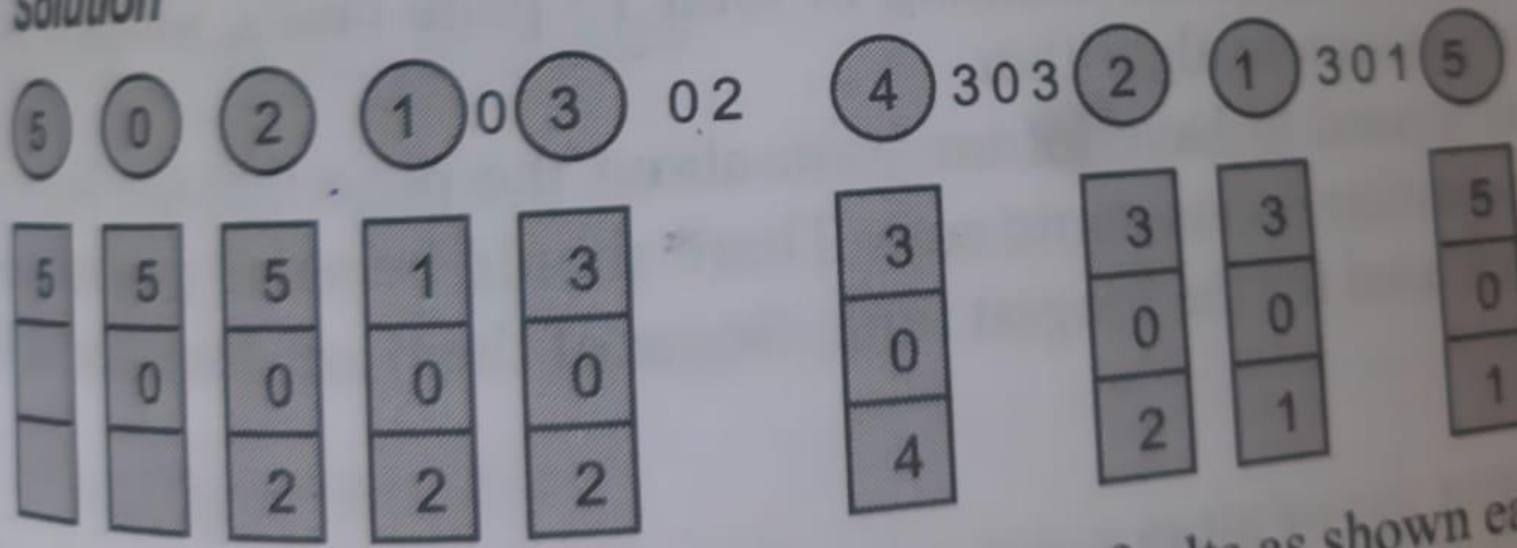
This policy
an example.

Example 11.9

Calculate the number of page faults for the following reference string using optimu
with frame size as 3.

5 0 2 1 0 3 0 2 4 3 0 3 2 1 3 0 1 5

Solution



page faults as shown earlier. The next pa
which page out of 5,

Optimal Algorithm

- Reference string given. Solve for 4 frames using OPT algorithm.

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

Optimal Algorithm

- Reference string given. Solve for 4 frames using OPT algorithm.

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

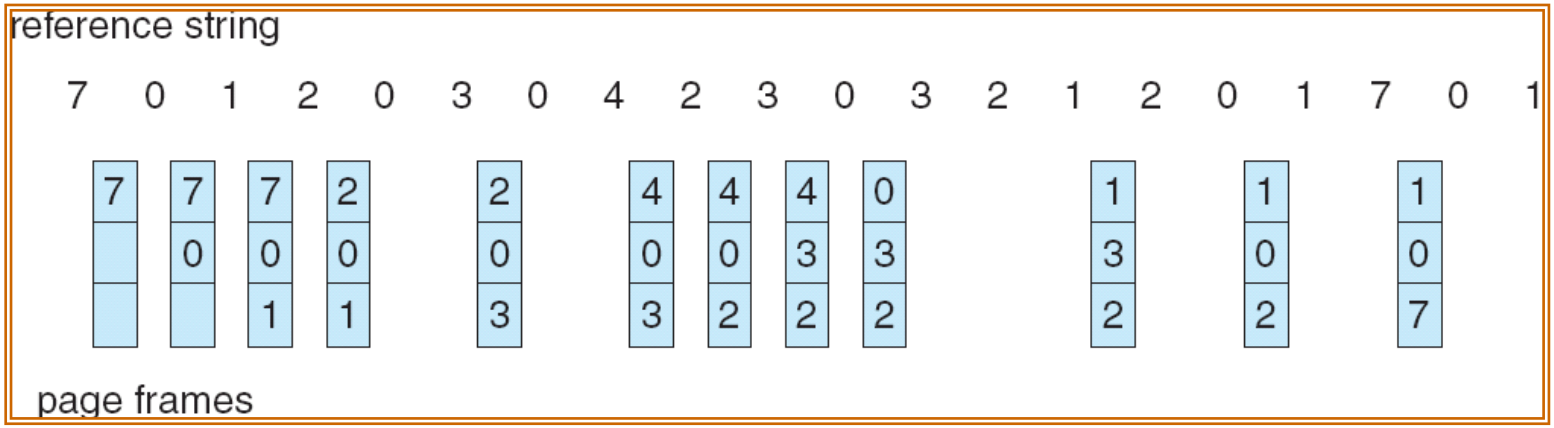
1	4
2	
3	
4	5

6 page faults

Least Recently Used (LRU) Algorithm

- If we use recent past as an approximation of the near future, then we will replace the page that has not been used for the longest period of time.
- This approach is least recently used algorithm
- FIFO uses the time when a page was brought into memory.
- The optimal uses the time when a page is to be used.
- LRU associates with each page the time of that page's last use.

LRU Page Replacement



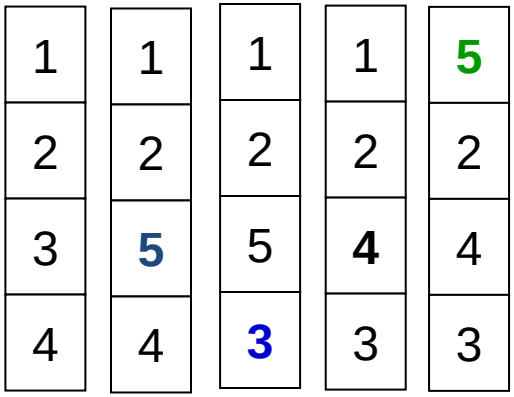
12 page faults

LRU example

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Solve it for LRU, using 4 frames.

Least Recently Used (LRU) algorithm

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



8 page faults

LRU Algorithm

Reference string:

5, 0, 2, 1, 0, 3, 0, 2, 4, 3, 0, 3, 2, 1, 3, 0, 1, 5

for 3 frames and using LRU algorithm. Find out the number of page faults.

Ans:

for 3 frames: 13 page faults

LRU Algorithm (Cont.)

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to be changed

LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - Implemented using doubly linked list.
  - Optimal and LRU does not suffer from Belady's anomaly.

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- LFU Algorithm: replaces page with smallest count
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Resident Set

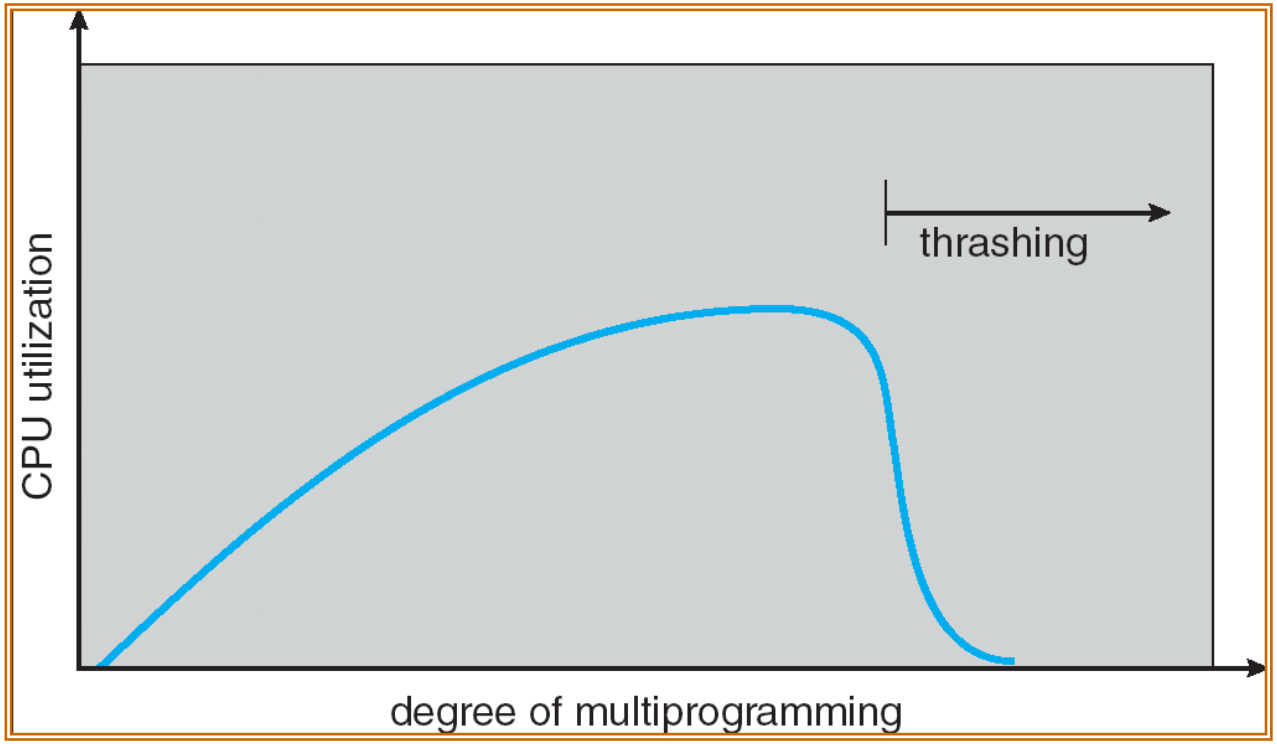
- In a virtual memory system, a process's resident set is that part of a process' s address space which is currently in main memory . If this does not include all of the process's working set, the system may thrash.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- Thrashing ≡ a process is busy swapping pages in and out

Thrashing (Cont.)



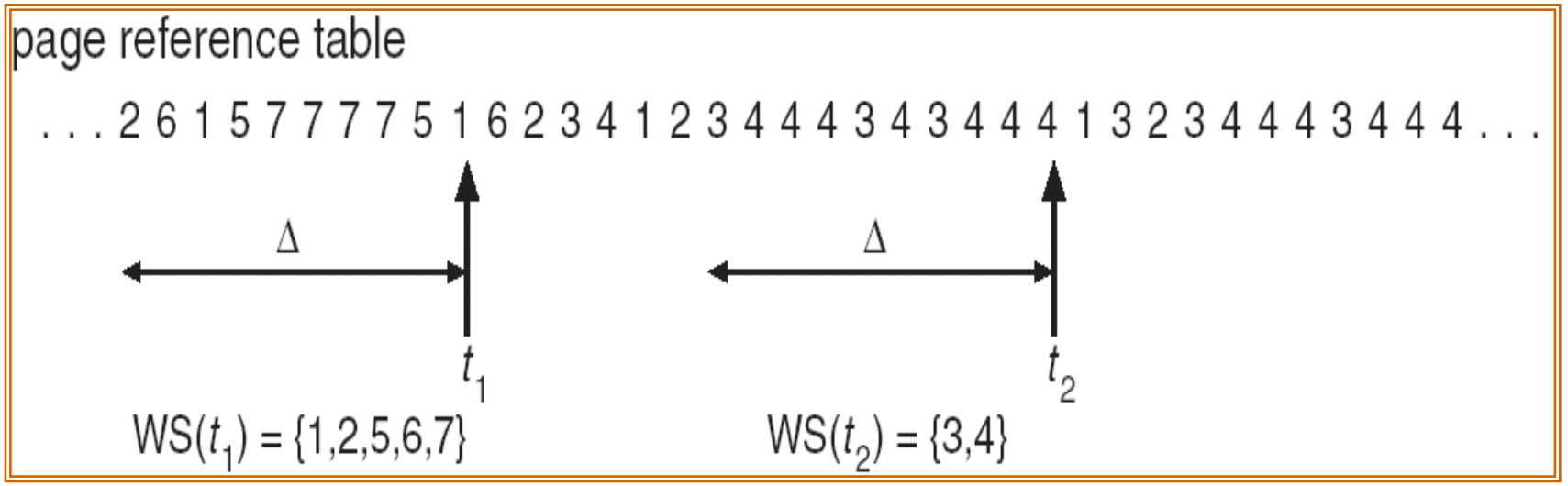
Working-Set Model

- The working set of a program or system is that memory or set of addresses which it will use in the near future.
- The working set should fit in the storage level otherwise the system may thrash

Working-Set Model

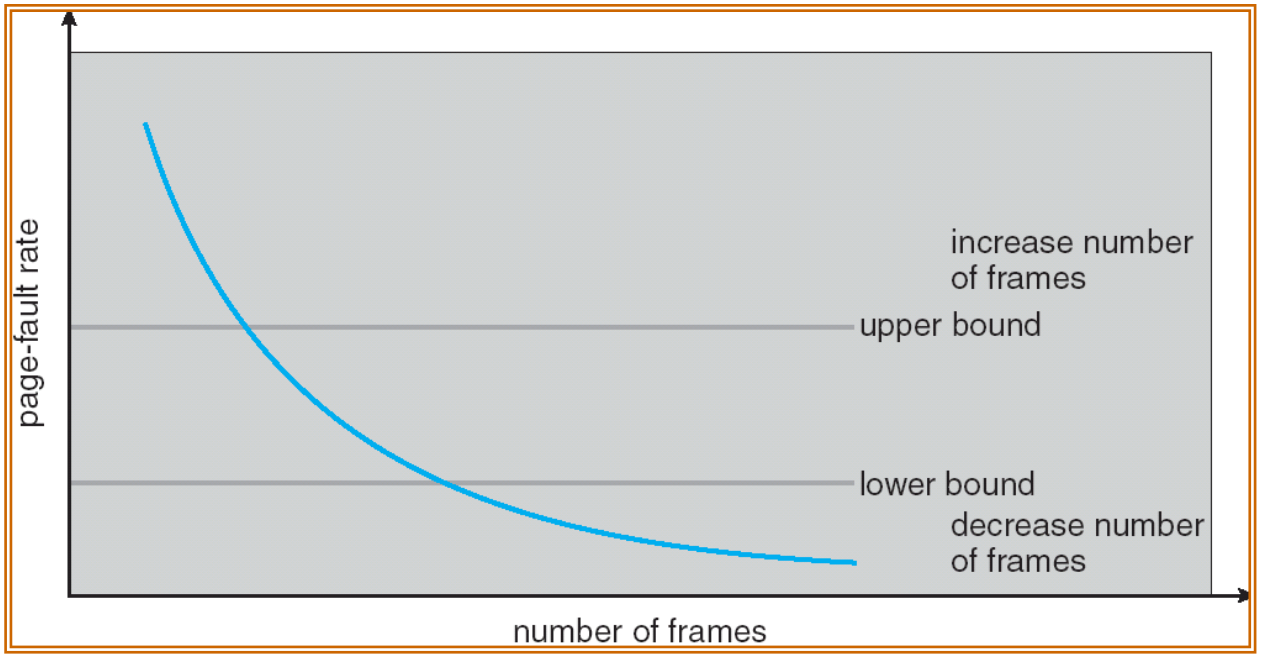
- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set size of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing (m=total available frames)
- Policy if $D >$ m, then suspend one of the processes

Working-set model



Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Cleaning Policy

- Demand cleaning
  - a page is written out only when it has been selected for replacement
- Precleaning
  - pages are written out in batches

Cleaning Policy

- Best approach uses page buffering
  - Replaced pages are placed in two lists
    - Modified and unmodified
  - Pages in the modified list are periodically written out in batches
  - Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Solve for FIFO, OPT, LRU

Reference string: 1, 0, 5, 1, 1, 3, 5, 1, 5, 3, 4, 5, 2, 1, 3, 0, 1, 4, 0, 5

Frame size 3 and 4

End of Chapter