Paging

Virtual Memory

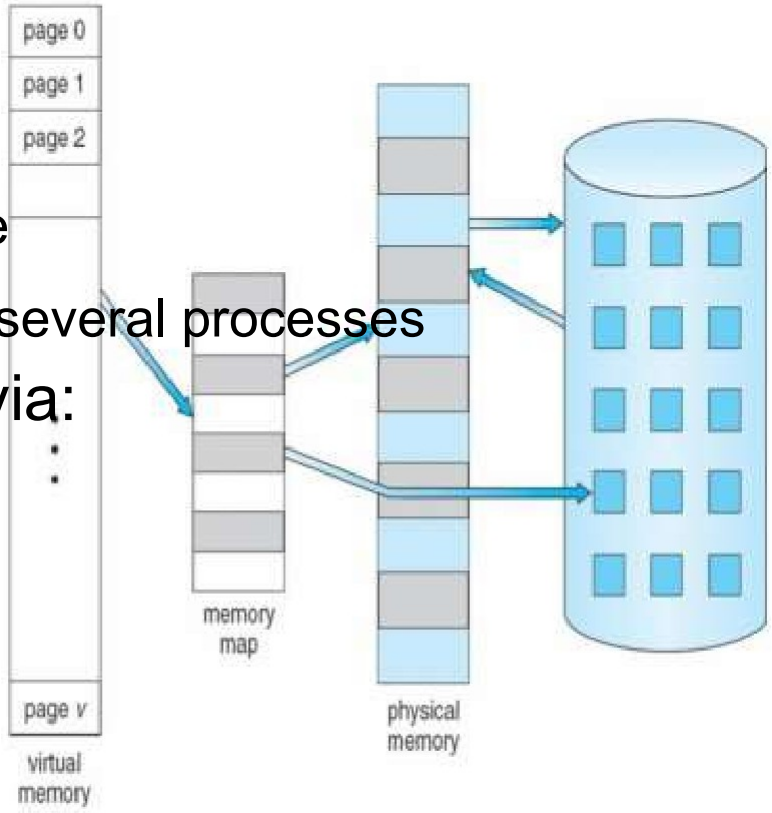
- Background
- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing
- Operating-System Examples

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
- To discuss the principle of the working-set model

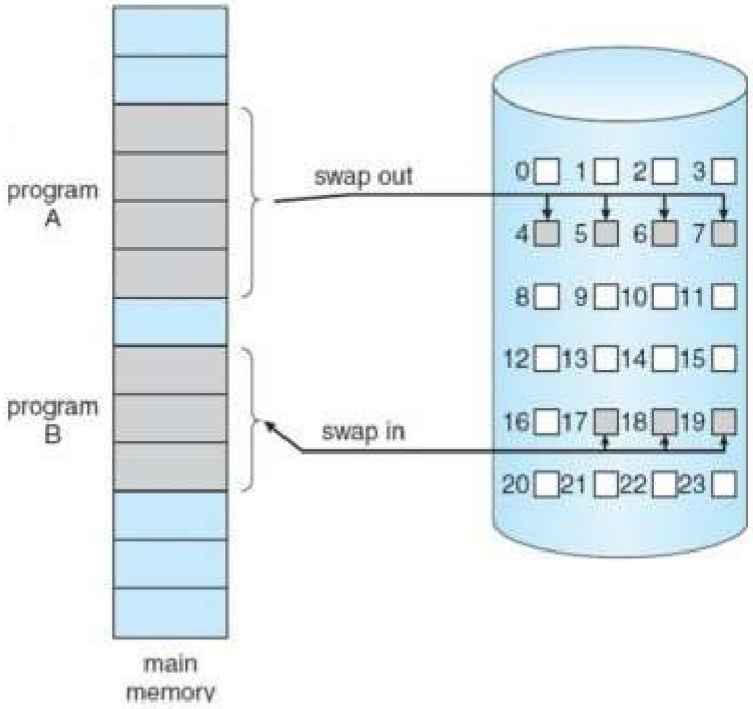
Background

- **Virtual memory** - separation of user logical memory from physical 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

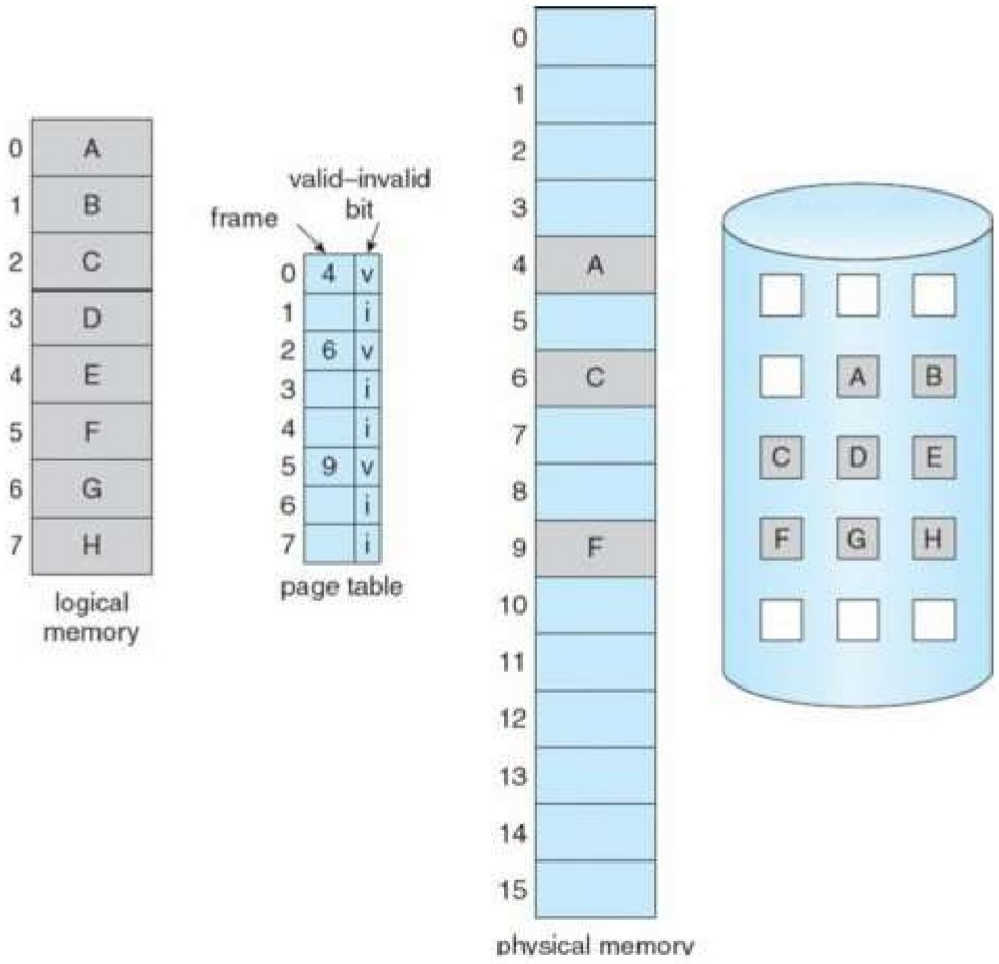
- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users



- 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



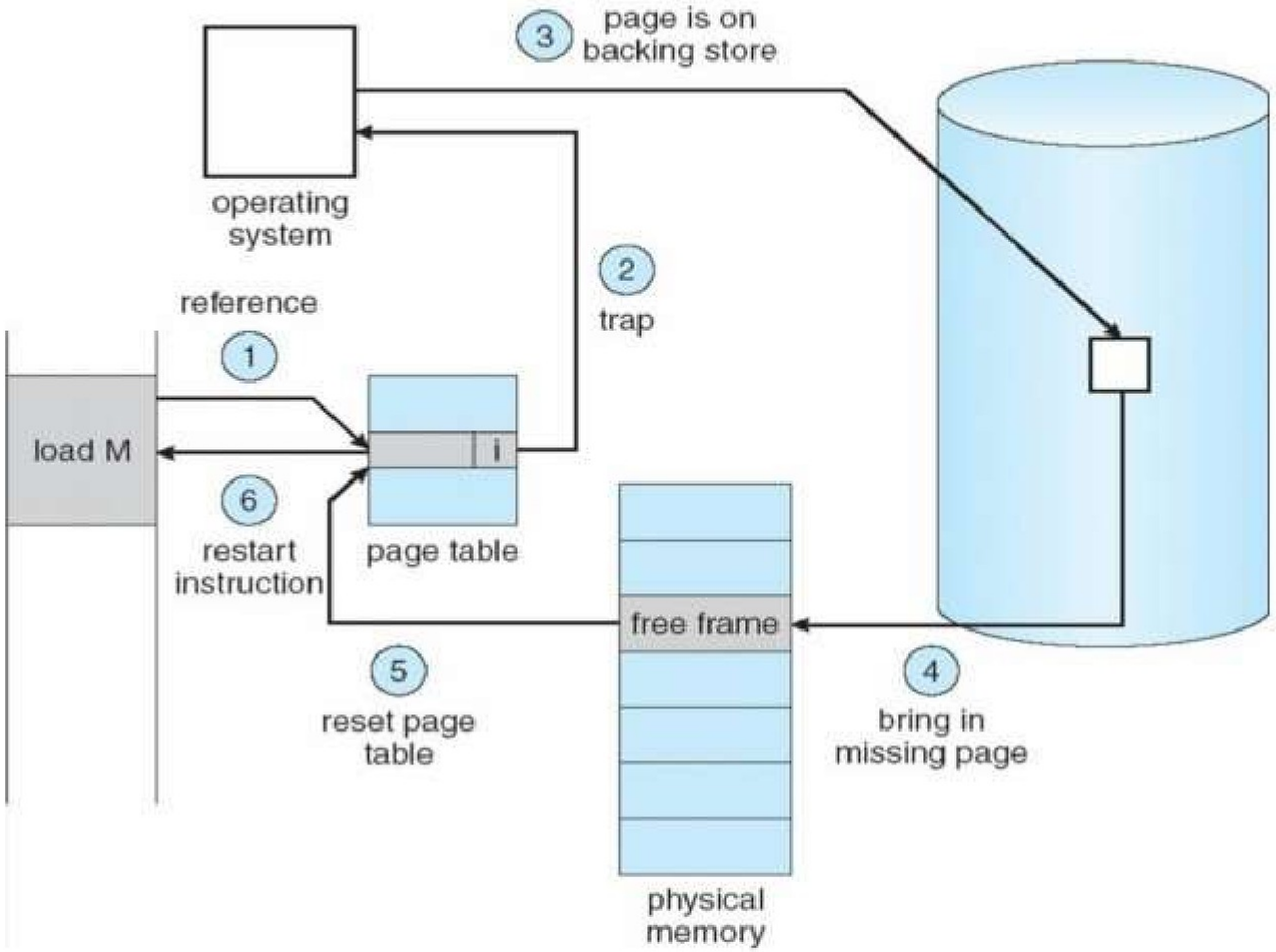
Page Fault

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

page fault

1. Operating system looks at another table to decide:
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

We start executing a process with no pages in the memory. When the Operating System sets the instruction pointer to the first pointer instruction of the process, which is on a non memory resident page, the process immediately faults for the process. After this page is brought into the memory, the process continues to execute, faulting as necessary until every page that it needs is in memory.

At this point, it can execute with no more faults. This is Pure Demand Paging.

HARDWARE SUPPORT

The hardware to support demand paging is the same as the hardware for paging and swapping:

- **PAGE TABLE : This table has the ability to mark an entry invalid through a valid invalid bit or a special value of protection bits.**

- **SECONDARY MEMORY : The memory holds those pages that are not present in the main memory. The secondary memory is usually a high speed disk.**

It is known as Swap Device and the section of the disk used for this purpose is known as the Swap Space.

Performance of Demand Paging

- Page Fault Rate $0 < p < 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Effective Access Time (EAT)

$$EAT = (1 - p) \times \text{memory access}$$
$$+ p \text{ (page fault overhead}$$
$$+ \text{swap page out}$$
$$+ \text{swap page in}$$
$$+ \text{restart overhead)}$$

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 - p) x 200 + p (8 milliseconds)

  = (1 - p x 200 + p x 8,000,000

  = 200 + p x 7,999,800

EAT is directly proportional to the page fault rate.

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

Page Replacement

- Basic to demand paging
- When a user process executes and page fault occurs and there are no free frames in memory
  - One choice for OS is to terminate the process
  - Another choice is to swap out one process
  - One frame is freed that is not currently being used… called *Page Replacement*
- *Page replacement completes separation between logical memory and physical memory - large virtual memory can be provided on a smaller physical memory*

STEPS IN PAGE REPLACEMENT :

REPLACEMENT :

- Find the location of the desired page on the disk.

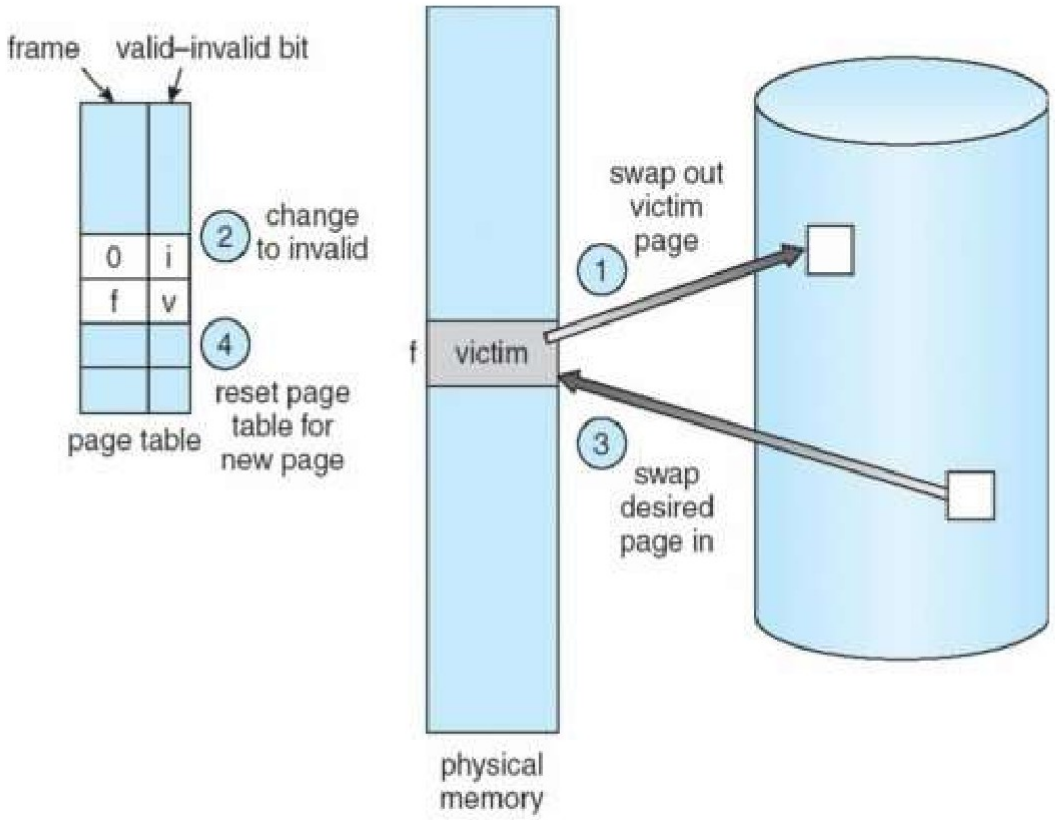
- Find a free frame :
  - ✔ If there is a free frame, use it.
  - ✔ If there is no free frame, use a page replacement algorithm to select a victim frame.
  - ✔ Write the victim frame to the disk, change the page and frame tables accordingly.

- Read the desired page into the newly freed frame, change the page and frame tables.

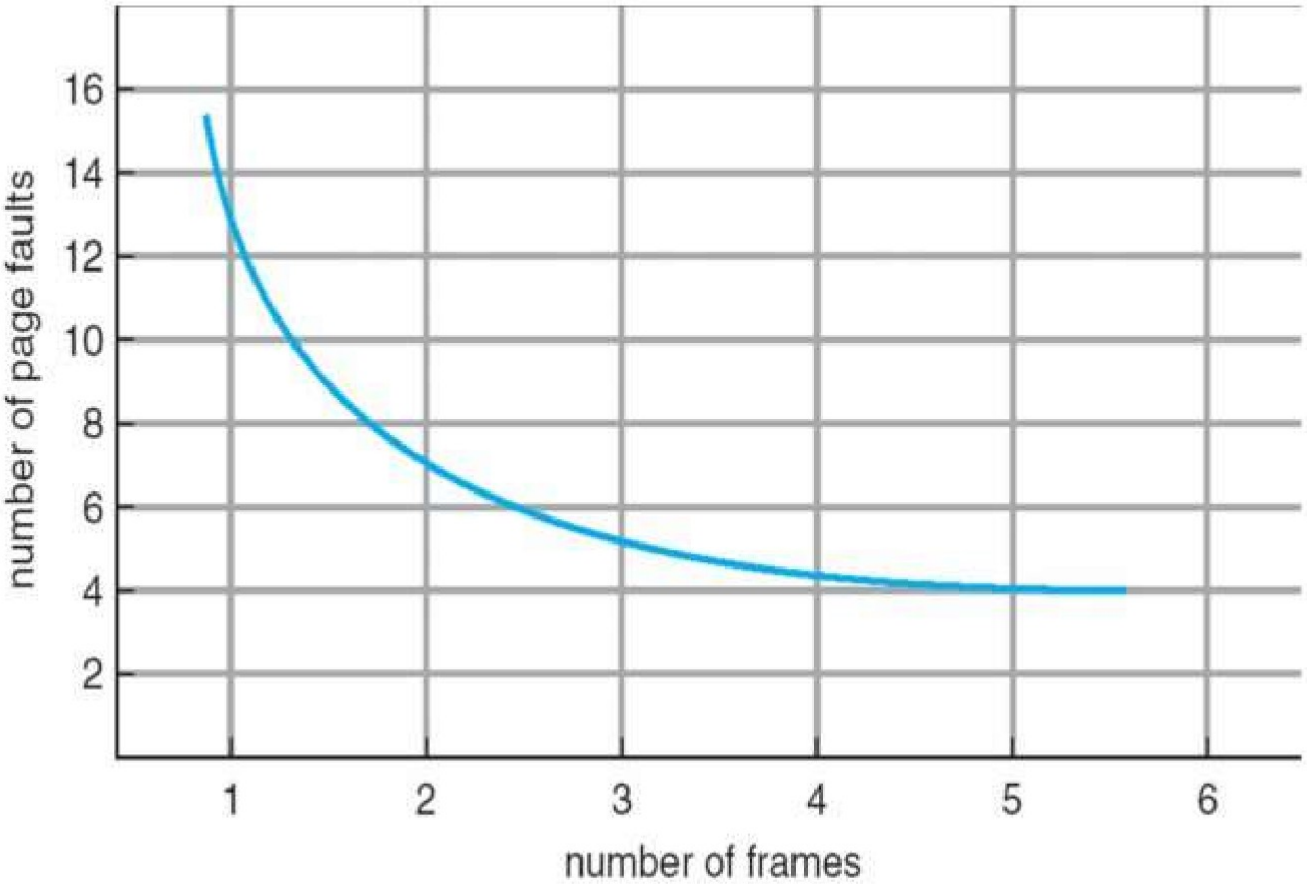
- Restart the user process.

Page Replacement

Use **modify (dirty) bit** to reduce overhead of page transfers - only modified pages are written to disk



Graph of Page Faults Versus The Number of Frames



Page Replacement Algorithms

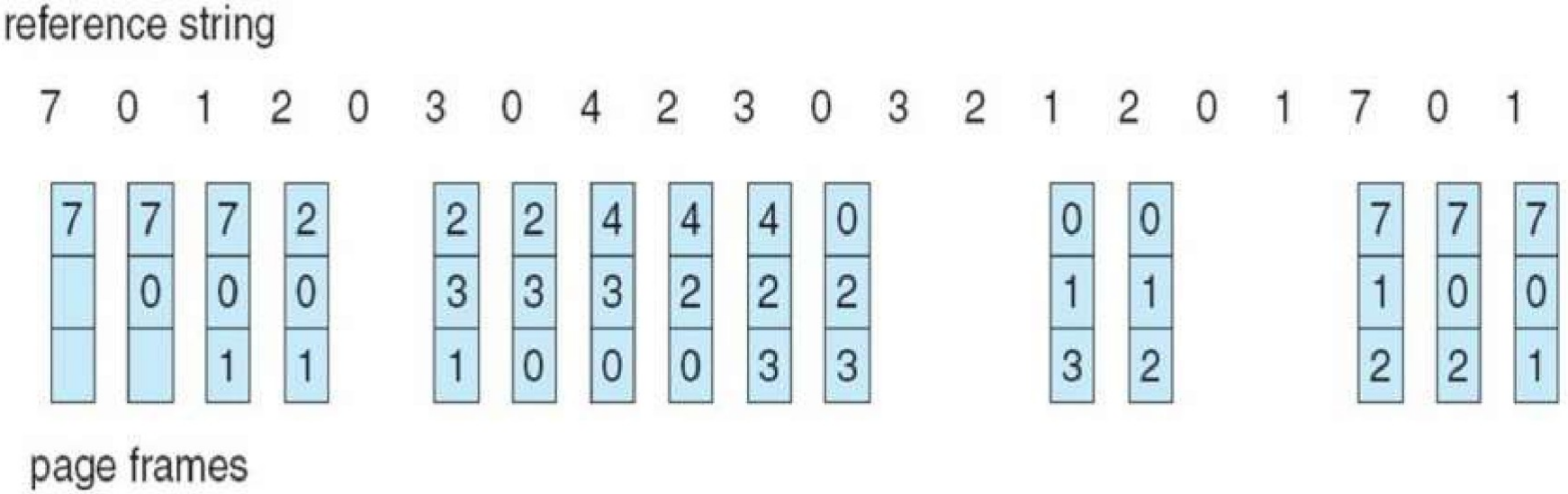
- Want lowest page-fault rate

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

- In all our examples, the reference string is

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

FIFO Page Replacement



First-In-First-Out (FIFO) Example

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

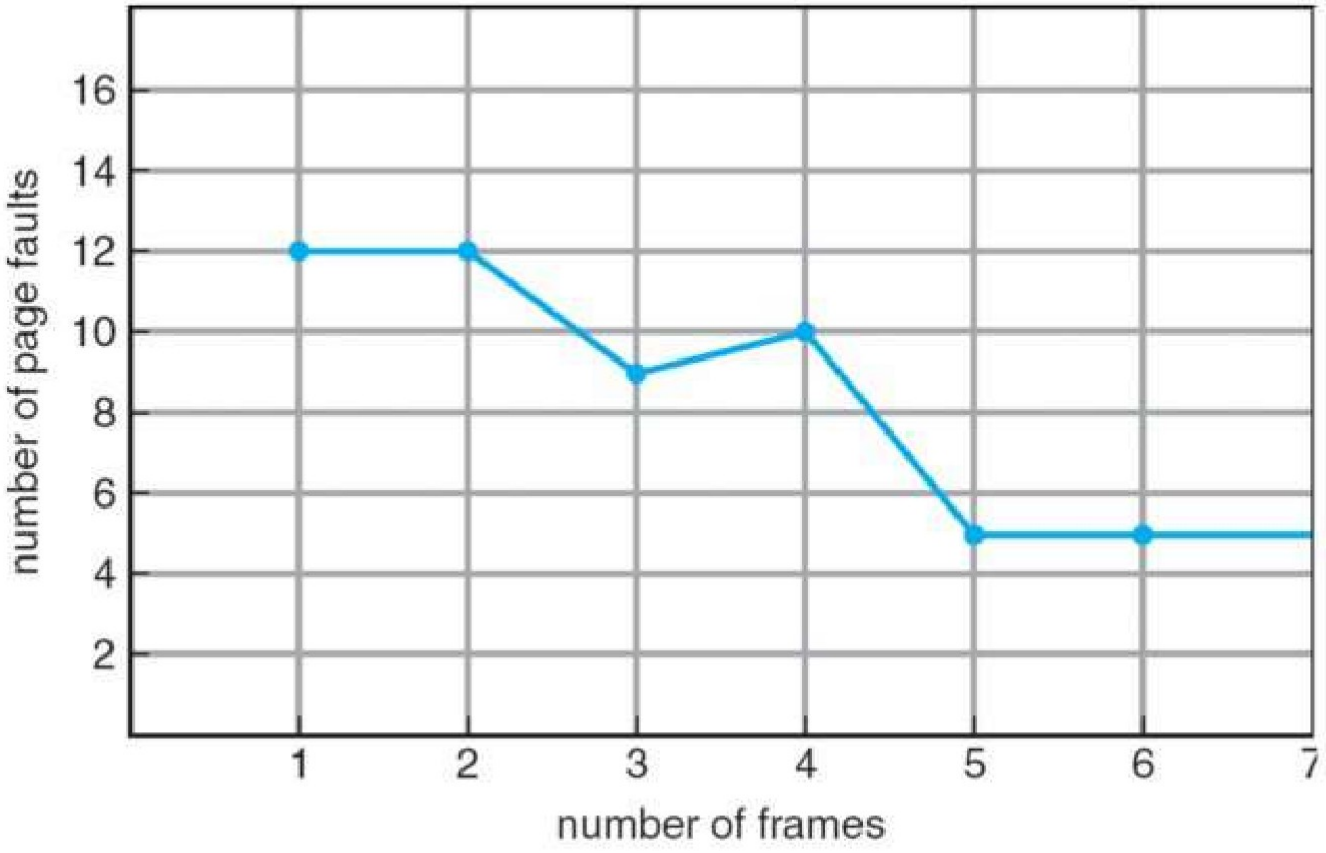
3 frames (3 pages can be in memory at a time per process)

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

4 frames

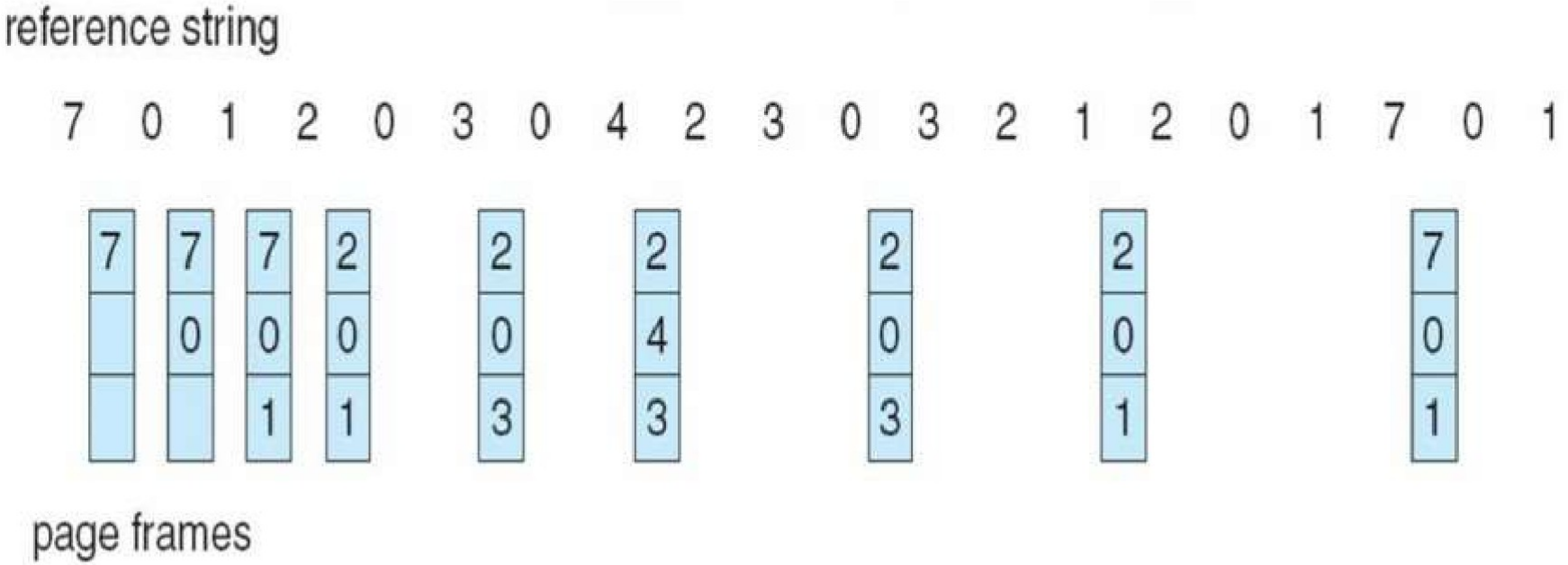
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

FIFO Illustrating Belady's Anomaly



Optimal Page Replacement

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



Unfortunately, the optimal page-replacement is difficult to implement, because it requires future knowledge of the reference string

Least Recently Used (LRU) Algorithm

- LRU replacement associates with each page the time of that page's last use

- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time

Least Recently Used (LRU) Algorithm

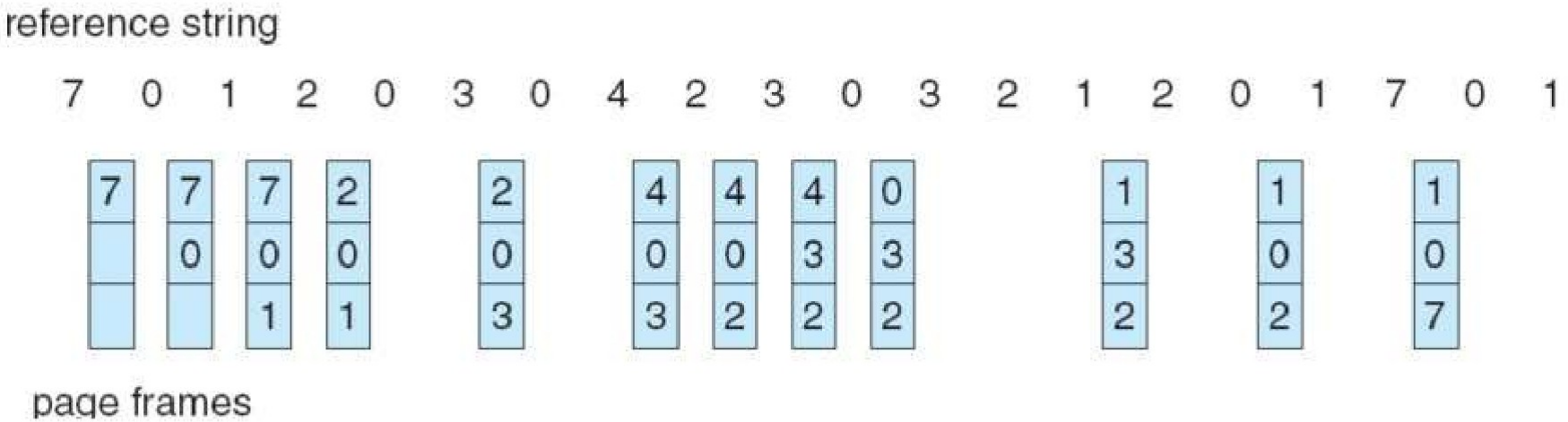
- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

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

LRU Implementation

- The major problem is how to implement LRU replacement:
  - **Counter**: whenever a reference to a page is made, the content of the clock register are copied to the time-of-use field in the page table entry for the page. We replace the page with the **smallest** time value
  - **Stack**: Whenever a page is referenced, it is removed from the stack and put on the top. In this way, the most recently used page is always at the top of the stack

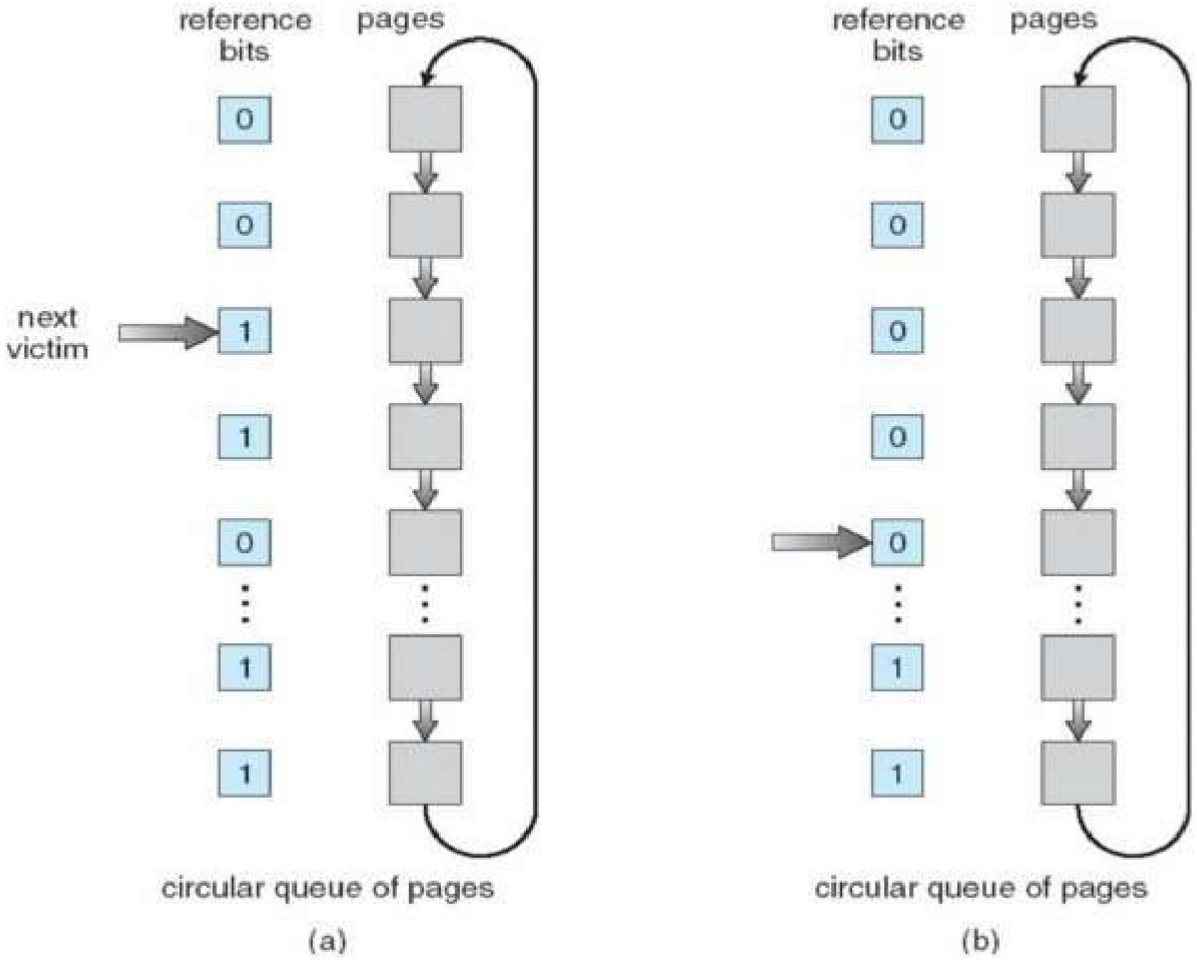
Example: LRU Page Replacement



Second Chance LRU

- Second-Chance algorithm is actually a FIFO replacement algorithm with a small modification that causes it to approximate LRU.
- When a page is selected according to a FIFO order, we check its reference bit.
  - If it is set (the page was referenced), we clear it and look for another page.
- Also called the clock algorithm.
- In the worst case, when all bits are set, the pointer cycles through all of the pages, giving each page a second chance and clearing its bit.
- Requires hardware support for reference bit.

Second-Chance (clock) Page-Replacement Algorithm



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

Allocation of Frames

- Each process needs *minimum* number of pages

- The minimum no. of frames per process is defined by the architecture, the maximum no. is defined by the amount of physical memory.

- Two major allocation schemes :-

  - fixed allocation
  - priority allocation

Fixed Allocation

- **Equal allocation** - For example, if there are 100 frames and 5 processes, give each process 20 frames.
- **Proportional allocation** - Allocate according to the size of process

s_i size of process p_i

S is sum of all the process

m is total number of free frames

a_i allocation for p_i $(s_i / S) * m$

$s_1 = 10$

$s_2 = 127$

$m = 64$

$a_1 = (10/137) * 64 = 4$

$a_2 = (127/137) * 64 = 57$

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size

- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

- **Global replacement** - process selects a replacement frame from the set of all frames; one process can take a frame from another

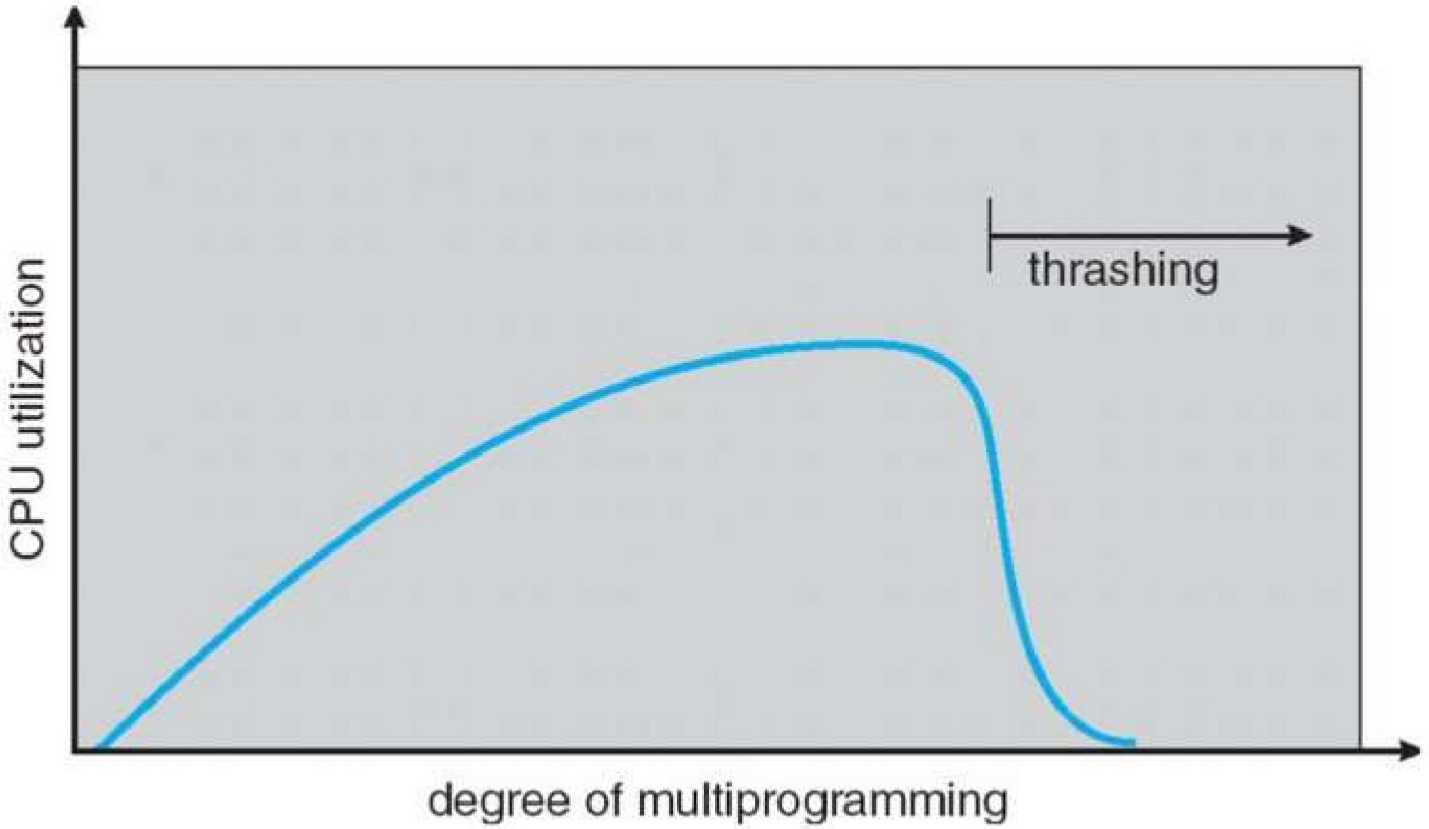
- **Local replacement** - each process selects from only its own set of allocated frames

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



Working-Set Model

- working-set window = a fixed number of page references

  Example: 10,000 instruction
- WSS_i (working set of Process P_i) =

  total number of pages referenced in the most recent (varies in time)
  - if too small will not encompass entire locality
  - if too large will encompass several localities
  - If is equal will encompass entire program
- $D = \sum WSS_i$, D= total demand frames, m= available frames
- if $D > m$ Thrashing
- Policy if $D > m$, then suspend one of the processes

Working Set Model

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

$WS(t_1) = \{1,2,5,6,7\}$

$WS(t_2) = \{3,4\}$

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