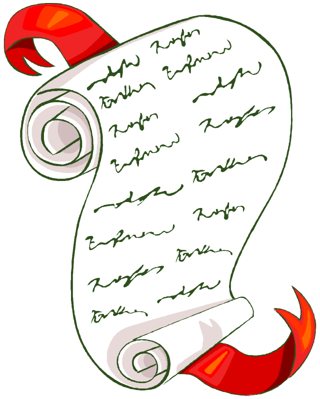
CMP320
Operating Systems

Virtual Memory

Muhammad Adeel Nisar

Note: Slides Courtesy
- Dr Syed Mansoor Sarwar.
- Mr Arif Butt

Today’s Agenda



- Review of Previous Lecture
- Introduction to Virtual Memory
  - Demand Paging
  - Servicing the page faults
  - Performance of Demand Paging
  - Memory Mapped Files
  - Page Replacement Algorithms
    - FIFO
    - Optimal
    - LRU
    - Page Buffering
- Thrashing

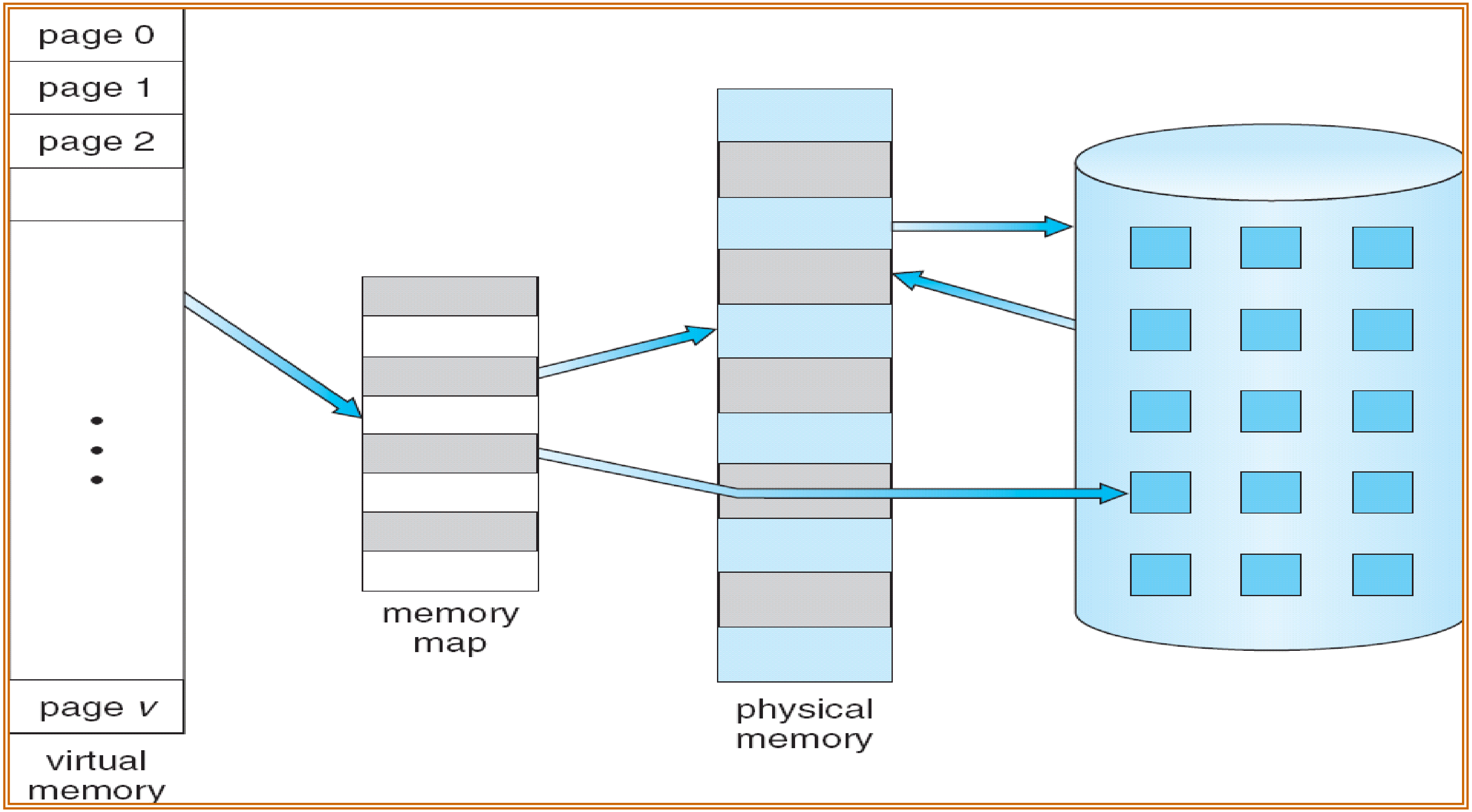
INTRODUCTION TO VIRTUAL MEMORY

- We have seen that a user process had to be completely loaded into memory before it could run
- This is wasteful since a process only needs a small amount of its total address space at any one time (locality)
- Knuth's estimate that 90% of the time, a process executes in 10% of its code
- Virtual memory is a technique used by operating systems that allows a process to execute programs having size larger than the available main memory
- Only a part of a process needs to be loaded in the main memory for it to execute

INTRODUCTION TO VIRTUAL MEMORY

- If an entire process is not loaded while it executes, it may be possible that the process requests for an instruction/data in a page that is not in the main memory rather is there in the backing store. Hardware and software cooperated to make things work
- Extend the page tables with extra bit “present bit” or “valid bit”; if it is one that means the page in loaded and vice versa
- In such situations, the operating system need to take two kinds of scheduling decisions:
  - **Page Selection.** When to bring pages into memory
    - **Demand Paging**. Start up process with no pages loaded, load a page when a page fault for it occurs
    - **Request Paging**. Let user say which pages are needed. Limitation is user don’t always knows best
    - **Pre-paging.** Bring a page into memory before it is referenced. When one page is referenced , bring in the next one as well
  - **Page Replacement**. Which page(s) should be thrown out, and when

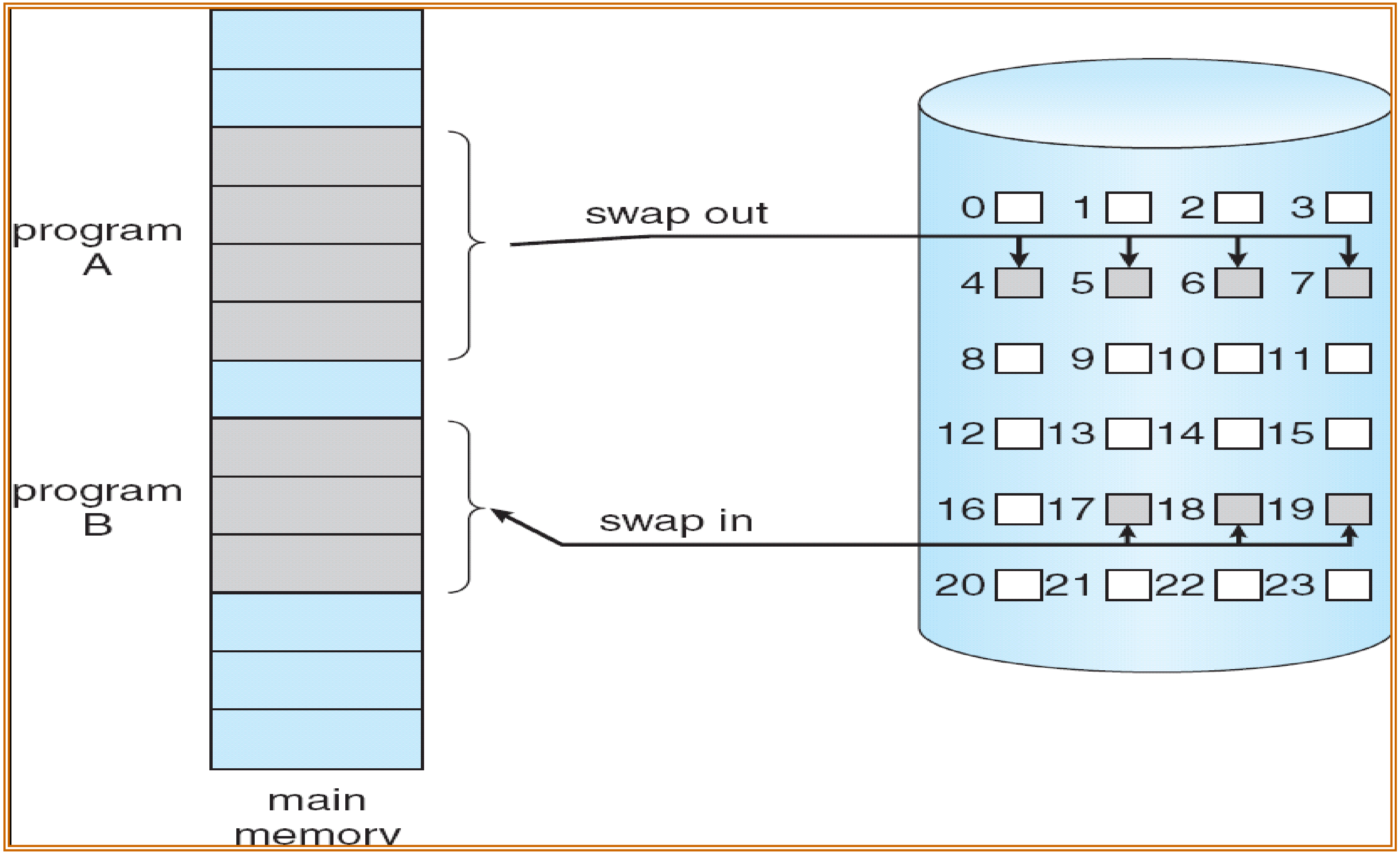
INTRODUCTION TO VIRTUAL MEMORY



DEMAND PAGING

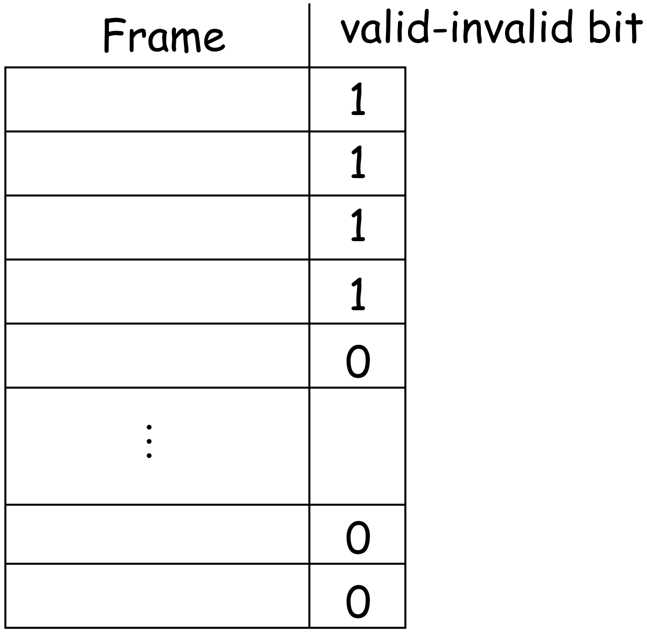
- Demand paging says; bring a page into memory only when it is needed
- When ever a process make a request for a page a reference is made to it
  - —invalid reference ⇒ abort. (The page a process has requested is not in its process address space)
  - —not-in-memory ⇒ page fault occurs and bring the page to memory
  - —No free frame ⇒ Swapping
- Advantages of demand paging are:
  - —Potentially Less I/O needed
  - —Potentially Less memory needed
  - —Faster response
  - —High degree of multiprogramming

TRANSFER OF A PAGE FROM MEMORY TO DISK



VALID – INVALID BIT

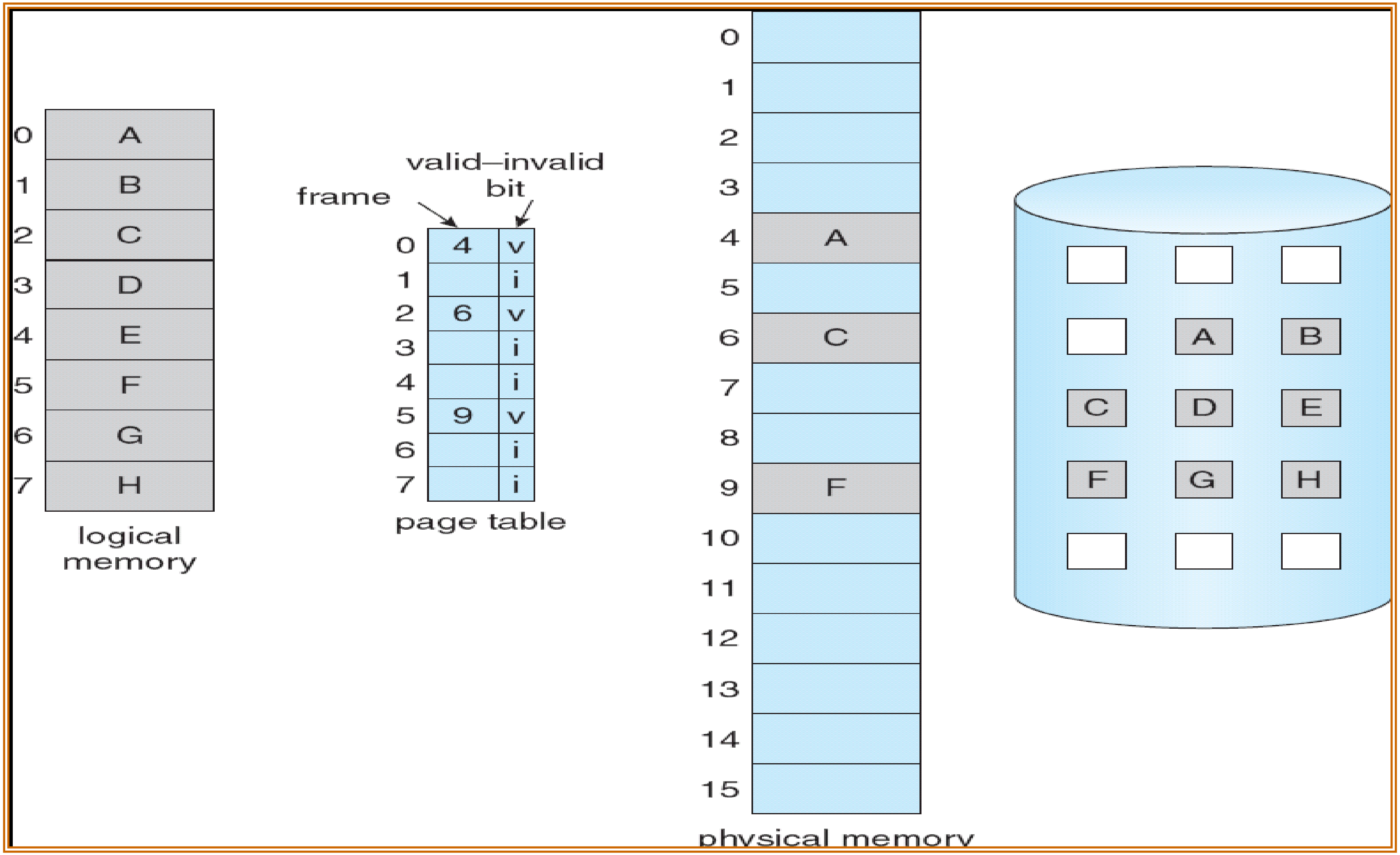
- With each page table entry a valid–invalid bit is associated
  (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries
- Example of a page table snapshot
- During address translation, if valid–invalid bit in page table entry is 0, a page fault occurs



Frame	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

page table

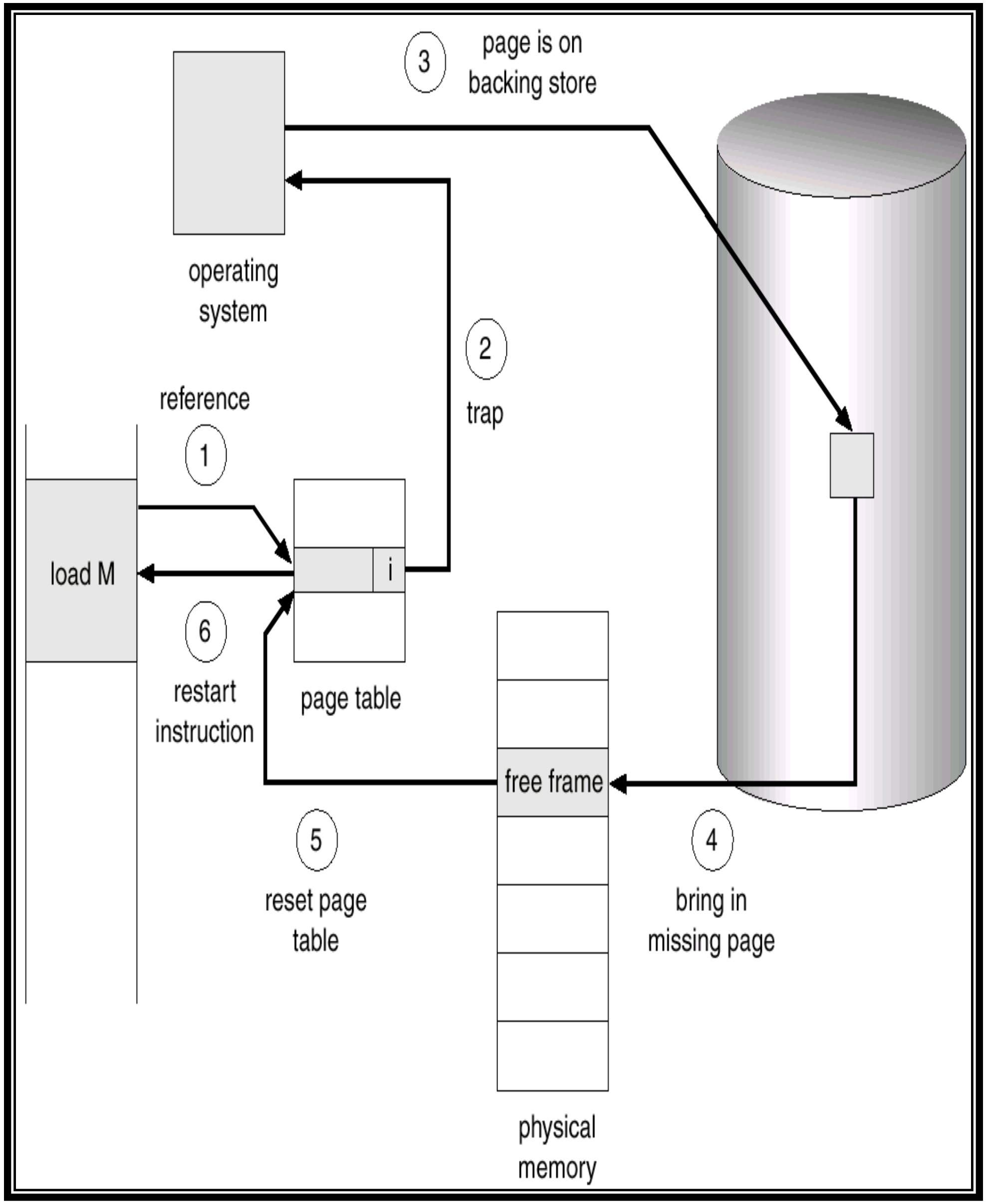
PAGE TABLE WHEN SOME PAGES ARE NOT IN MAIN MEMORY



SERVICING A PAGE FAULT

When ever a page reference is made, the operating system checks:

1. If it is an invalid reference a trap to the OS is made and the process is aborted
2. If it is a valid reference then the valid bit of the page table is checked. If the valid bit is zero that means the page is not loaded in the memory, so a page fault is generated. To service this page fault following steps are taken:
   a. Locate the page on the disk
   b. Allocate an empty frame from the frames allocated to that process and load the page into that frame
   c. If there is no empty frame available with that process then select a victim frame and swap out that victim frame/page back to the hard disk
   d. Swap in the desired page into the selected frame
   e. Store the frame number in the appropriate page table entry, and set valid bit to one
   f. Restart instruction



PERFORMANCE OF DEMAND PAGING

- Page Fault Rate $0 \le p \le 1.0$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- **Effective Access Time (EAT)**

  EAT = (No page fault) * mat + (page fault) * (page fault service time)

  EAT = (1 – p) x mat + p * (page fault overhead + swap page out + swap page in+ restart overhead)

Problem 30

Consider a system with Memory Access Time of 100 nanosecond. Total Page fault Service Time is 25 millisecond. Calculate Effective Access Time. Discuss how much the system will be slowed down if one out of 1000 accesses causes a page fault.

$T_{effective} = (1 - p) * 100 + p\ (25 \times 10^6)$

Let p = 1/1000

$T_{effective} = (1 - 1/1000) * 100 + 1/1000 * (25 \times 10^6)$

$T_{effective} = 99.9 + 25000$

$T_{effective} = 25099.9\ nsec = 25000\ nsec$

So the system is slowed down by a factor of 250 times, by just one page fault out of 1000 page accesses.

PERFORMANCE OF DEMAND PAGING

- If we want less than 10 percentage degradation in effective memory access time then we have the following inequality

$$T_{effective} = (1 - p) * 100 + p\,(25 * 10^6)$$

$$110 > (1 - p) * 100 + 25000000 * p$$

$$110 > 100 - 100 * p + 25000000 * p$$

$$10 > -100 * p + 25000000 * p$$

$$10 > 24999900 * p$$

$$10/24999900 > p$$

$$p < 1/2499990$$

- This means we can allow only one page fault every 2,500,000. (2.5 million)

PERFORMANCE OF DEMAND PAGING

Problem 31

Consider a system with MAT of 200 nanosecond. Average Page fault service time 8 ms. Compute Effective Access Time and discuss how much the system will be slowed down if one out of 1000 accesses causes a page fault. Also compute the page fault rate (p), if we want a slow down by less than 10%

PAGE FAULT AND NO FREE FRAME

- 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
- We are going to discuss various page replacement algorithms as coming up on next slides
  - **Local replacement** – If a process P1 having no free frame demands a page and the victim frame selected is of the same process P1 then it is called Local replacement
  - **Global replacement** – If a process P1 having no free frame demands a page and the victim frame selected is of some other process then it is called Global replacement

PAGE REPLACEMENT

ALGORITHMS

PAGE REPLACEMENT

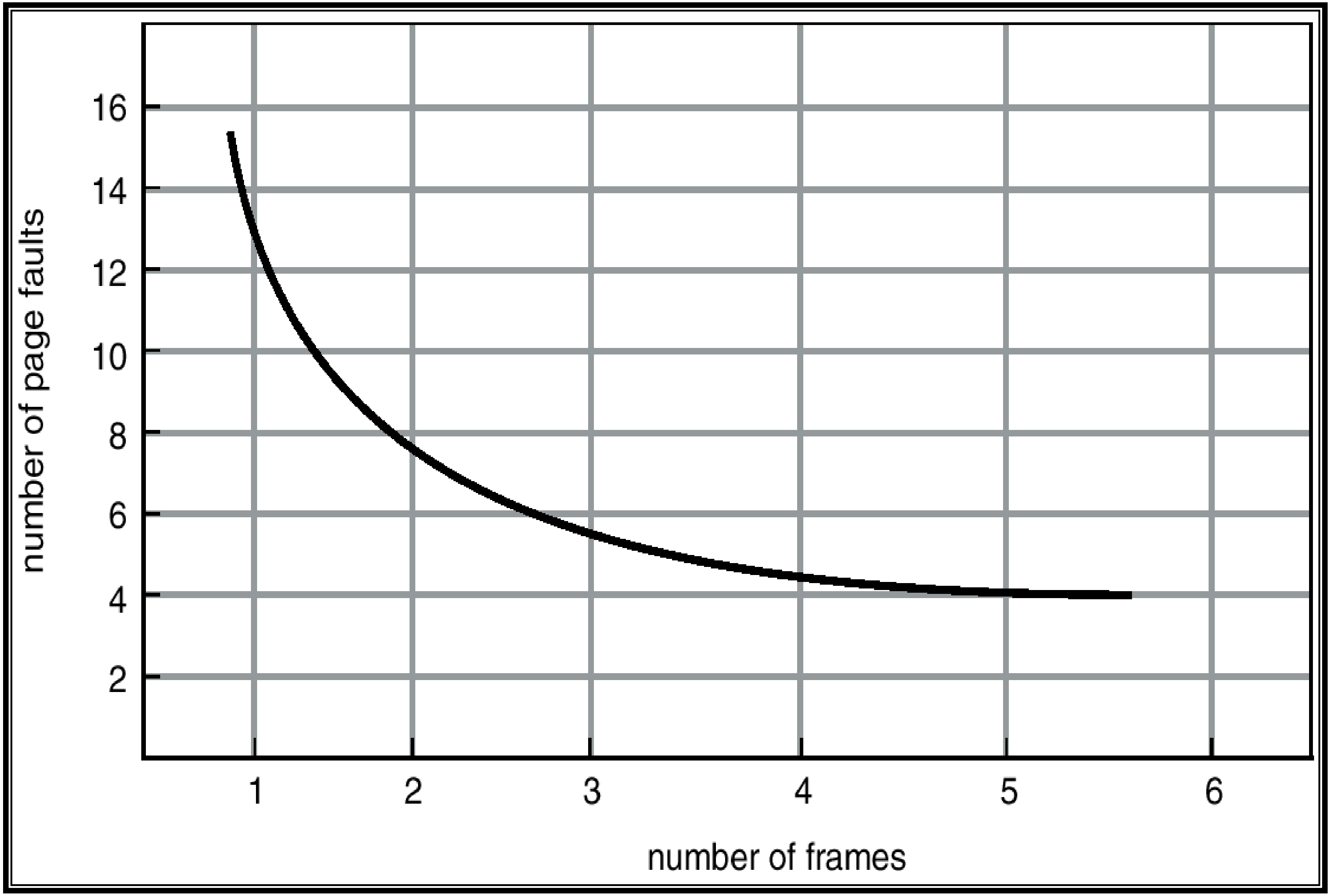
- When doing a page replacement the operating system has to make decisions regarding the fetching, placement and replacement of the pages
  - **Fetch Strategies** decides **when** a page should be brought into main memory from secondary storage (demand, request, & pre-paging)
  - **Placement Strategies** decides **where** a page should be placed in the main memory (Placement algos, e.g., next fit, best fit, …)
  - **Replacement Strategies** decides **which** page (victim page) from main memory should be swapped out on disk if there is no more free frame available with that process (Replacement algos, e.g., FIFO, Optimal, LRU, …)
    - Page fault service routine is modified to include page replacement as well
    - Dirty bit is used to reduce overhead of page transfers, i.e., only modified pages are written back to disk. This bit is set by hardware when data is written to a page and is checked by operating system at page replacement time

PAGE REPLACEMENT ALGORITHMS

- Principle of Optimality
  - Replace the page that will not be used again the farthest time in the future
- FIFO - First in First Out
  - Replace the page that has been in main memory the longest
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time
- LFU - Least Frequently Used
  - Replace the page that is used least often
- MFU – Most Frequently Used
  - Replace the page that is most frequently used
- Page Buffering
  - Uses FIFO
- Working Set
  - Keep in memory those pages that the process is actively using

NOTE: 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

PAGE FAULTS VS NUMBER OF FRAMES



FIRST IN FIRST OUT

1. Simplest algorithm
2. Associates with each page the time when that page was brought into memory. When a page is to be replaced the oldest is chosen
3. Instead of recording time when a page is brought in, we use a FIFO queue
4. Suffers with Belady's anomaly. “For some page replacement algorithms, by increasing the allocated frames to a process, the page fault rate may also increase”

FIRST IN FIRST OUT (cont…)

A sample string showing the Belady's anomaly in FIFO algorithm

- Number of frames allocated = 3
- Reference string:
- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 9

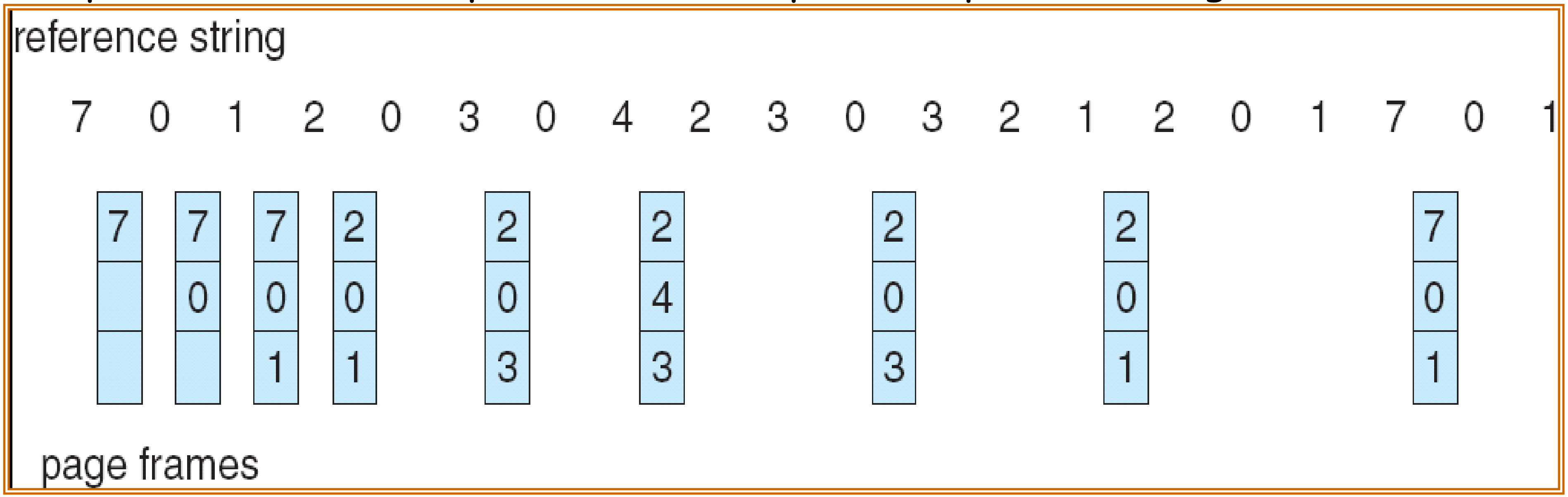
1	1	1	4	4	4	5	5	5
	2	2	2	1	1	1	3	3
		3	3	3	2	2	2	4

- Number of frames allocated = 4
- Reference string:
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Number of page faults = 10

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

OPTIMAL

1. Best possible algorithm, but impossible to implement
2. Replace the page that will not be used for the longest period of time. (Look forward but how?)
3. Has the lowest page fault rate
4. Never suffer from Belady's anomaly
5. Unrealizable, because at the time of page fault, the OS has no way of knowing when each of the pages will be referenced next
6. Used as Bench Mark, whenever an algorithm is to be evaluated its performance is compared with the optimal replacement algorithm



LRU

- An approximation of optimal algorithm; assumes that pages that have not been used for ages will probably remain unused for a long time
- LRU policy replaces the page in memory that has not been referenced for the longest time
- Example
  - Number of frames allocated = 3
  - Reference string:

    1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - Number of page faults = 10

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

LRU IMPLEMENTATION

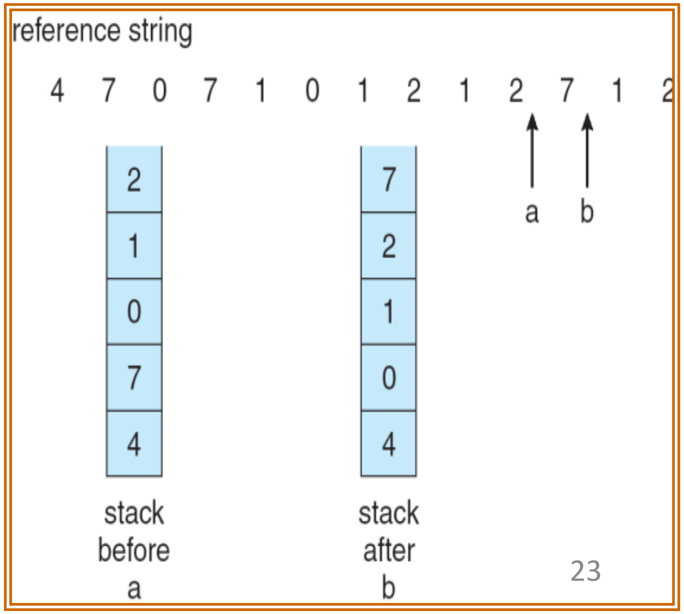
- **Counter Based Implementation.** Every page entry has a counter, every time page is referenced through this entry, copy the clock into the counter. When a page needs to be replaced, choose the one with minimum counter value
- **Stack Based Implementation.** Keep a stack of page numbers. Whenever a page is referenced remove it form its current location and push it at the top of stack. The LRU page is at the bottom of the stack. So when you need to replace a page, you replace the page whose number is at the bottom of the stack
- *A doublee linked list implementation of stack requires six pointer changes ?*

Reference string

4 7 0 7 1 0 1 2 1 2 ↑ 7 ↑ 1 2

a b

Give the contents of stack before a and after b



COUNTING BASED PAGE REPLACEMENT

- Keep a counter of the number of references that have been made to each page and develop the following two schemes:
- **Least Frequently Used (LFU)**
  - This algo is based on locality of reference concept; the least frequently used page is not in the current locality
  - Replace the page with the smallest reference count. The reason of this selection is that an actively used page should have a large reference count. The problem is if a page is used heavily during the initial phase of a process but then is never used again. Since it was heavily used in the beginning it has a large count and remains in memory even though it is no longer needed. Solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average user count
- **Most Frequently Used (MFU)**
  - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used; i.e., it will be in the locality that has just started
  - Neither of these is used as they are expensive and they do not approximate Optimal replacement algorithm well.

PAGE REPLACEMENT ALGORITHMS (cont…)

PAGE BUFFERING

The OS keeps a pool of free frames, when a page fault occurs a victim page is chosen as before. However, the desired page is read into a free frame from the pool before the victim is written out. This allows the process to restart as soon as possible, without waiting for the victim to be written out. When the victim is later written out, its frame is taken from the process and added to the free frame pool. Thus a process in need can be given a frame quickly and while victims are selected, free frames are added to the pool in the background

PAGE REPLACEMENT ALGORITHMS (cont…)

PAGE BUFFERING

1. Follows FIFO. To improve performance, a replaced page is not lost but rather is assigned to one of two lists.
    1. Free page list if the page has not been modified.
    2. Modified page list if the page is modified.
2. When a page is not there in frames the two buffers are checked if the page exist in any one of them, a page fault is not declared and the page is taken from that particular buffer to the appropriate frame. Also the victim frame is swapped with the required frame (buffer1 or buffer 2).
3. If the page that is not there in frames, also does not exist in any of the buffers, a page fault is declared. Now the page is brought from secondary memory to the appropriate victim frame and that victim frame moves to one of the buffers using FIFO rule.

Example.

Consider the following reference string:

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

Use a single buffer with 3 frames. Repeat with 4 frames and compare page faults

PAGE REPLACEMENT ALGORITHMS (cont…)

Drawbacks of FIFO

1. A page which is being accessed quite often may also get replaced because it arrived earlier than those present
2. Ignores locality of reference. A page which was referenced last may also get replaced, although there is high probability that the same page may be needed again

Drawbacks of LRU

1. A good scheme because focuses on replacing dormant pages and takes care of locality of reference, but a page which is accessed quite often may get replaced because of not being accessed for some time, although being a significant page may be needed in the very next access

Drawbacks of NFU

1. Does not take care of locality of reference, and reacts slowly to changes in it. If a program changes the set of pages it is currently using, the frequency counts will tend to cause the pages in the new location to be replaced even though they are being used currently

Pages like page directory are very important and should not at all be replaced, thus some factor indicating the context of the page should be considered in every replacement policy, which is not done in any of above

SAMPLE PROBLEMS

Problem 32 Compute total page faults for the given page Trace for 4 frames using following algorithms:

- FIFO
- OPTIMAL
- LRU
- Page Trace/Reference String is given below:

2, 3, 4, 5, 6, 4, 5, 2, 7, 8, 9, 4, 9, 0, 8, 9, 1, 6, 5, 6, 5, 3

Problem 33 Compute total page faults for the given page Trace for 3 frames using Page Buffering algorithm with:

- Buffer Size = 1
- Buffer Size = 2

- Page Trace

6, 5, 4, 3, 0, 6, 5, 1, 3, 2, 5, 6, 4, 3, 2, 1, 6, 0, 3, 4, 1, 6

Problem 34 Compute total page faults for the given page Trace:

8, 9, 0, 1, 2, 0, 1, 8, 3, 4, 5, 0, 5, 6, 4, 5, 7, 2, 1, 2, 1,9

- Do it using 4 frames and then using 3 frames.
- Use following algorithms.

a. FIFO
b. LRU
c. OPTIMAL
d. Buffering with buffer size =1
e. Buffering with buffer size =2

SAMPLE PROBLEMS

Problem 35

- Consider a paging system with physical memory of 12 Bytes & Page size of 4 Bytes. A program stores and accesses characters A to Z. Compute the physical addresses of logical addresses where following program instructions are loaded:

  F, S, V, A, O, K, D, J, X, Z, H, A
- Use FIFO while constructing the Page Table
- Use Optimal while constructing the Page Table
- Use LRU while constructing the Page Table
- Use Page Buffering while constructing the Page Table

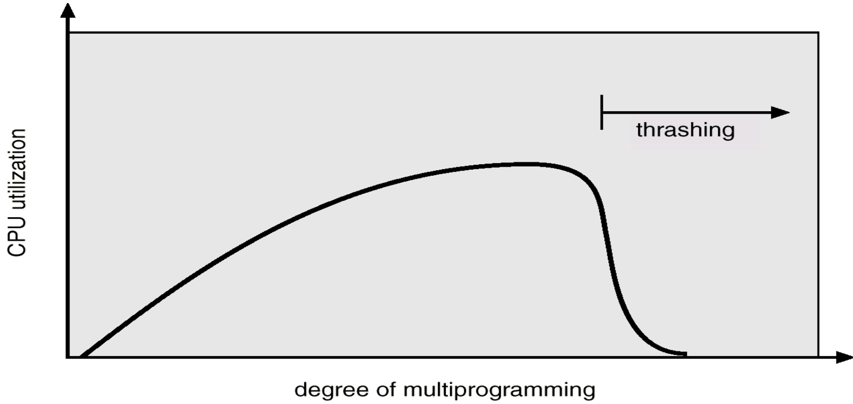
Problem 36

- Consider a paging system with physical memory of 12 Bytes & Page size of 4 Bytes. A program stores and accesses characters A to Z. Compute the physical addresses of logical addresses where following program instructions are loaded:

  Z, I, L, O, R, I, Y, K, P, U, C, I, T
- Use FIFO while constructing the Page Table
- Use Optimal while constructing the Page Table
- Use LRU while constructing the Page Table
- Use Page Buffering while constructing the Page Table

THRASHING

- If a process does not have “enough” frames, the page-fault rate is very high. This leads to:
  - Low CPU utilization
  - OS thinks that it needs to increase the degree of multi - programming
  - Another process is added to the system and that cause serious problems
- Thrashing is a phenomenon in which CPU spends much of its time swapping pages in and out, rather than executing instructions
- Thus in order to stop thrashing:
  - The degree of multiprogramming needs to be reduced
  - The effects of thrashing can be reduced by using a local page replacement. So if one process starts thrashing it cannot steal frames from another process and cause the later to thrash also
- Thrashing results in severe performance problems:
  - Low CPU utilization
  - High disk utilization
  - Low utilization of other I/O devices



ALLOCATION OF FRAMES

- Minimum number of frames that can be allocated to a process is dependent upon architecture. Maximum is dependent upon amount of available memory
- Each process needs a minimum number of frames so that its execution is guaranteed on a given machine. E.g. consider the following **mov** instruction on an IBM 370:

  MOV X,V

  - Instruction itself is 6 bytes, might span 2 pages
  - 2 pages to handle **from**
  - 2 pages to handle **to**
- So to execute the above instruction, we need to allocate a minimum of six frames, so that it can execute fully, else a page fault will keep occurring and the instruction will never be able to execute.
- Three major allocation schemes:
  - **Fixed Allocation**. Free frames are equally divided among processes
  - **Proportional Allocation**. Number of frames allocated to a process is proportional to its size

  Frames to be alloacate to P_i = {(No pages requested by P_i)/(Total demand of pages)} * (Available no of frames in the system)

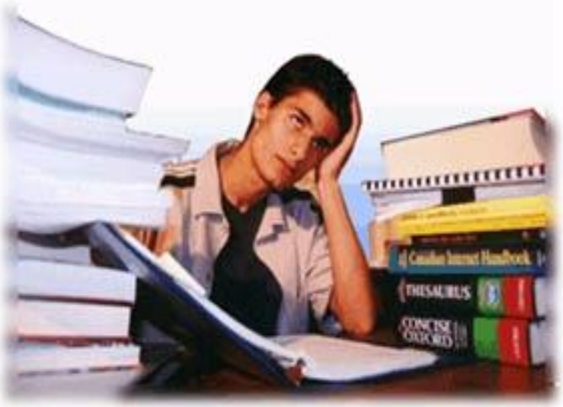
  - **Priority Allocation**. Number of frames allocated as per the process priority

Problem 37

Consider a system with 64 frames. At a particular instant of time there are three processes P1(10 pages), P2(40 pages) and P3(127 pages).

Allocate frames to processes using Fixed and prop

We're done for now, but Todo's for you after this lecture…



- Go through the slides and Book Sections: 9.1 to 9.7, 9.8.1, 9.9
- Solve all the sample problems given in slides to understand the concepts discussed in class