Virtual Memory

Dr Tarunpreet Bhatia

Assistant Professor

CSED, TIET

Disclaimer

This is NOT A COPYRIGHT MATERIAL

Content has been taken mainly from the following books:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings

Virtual Memory

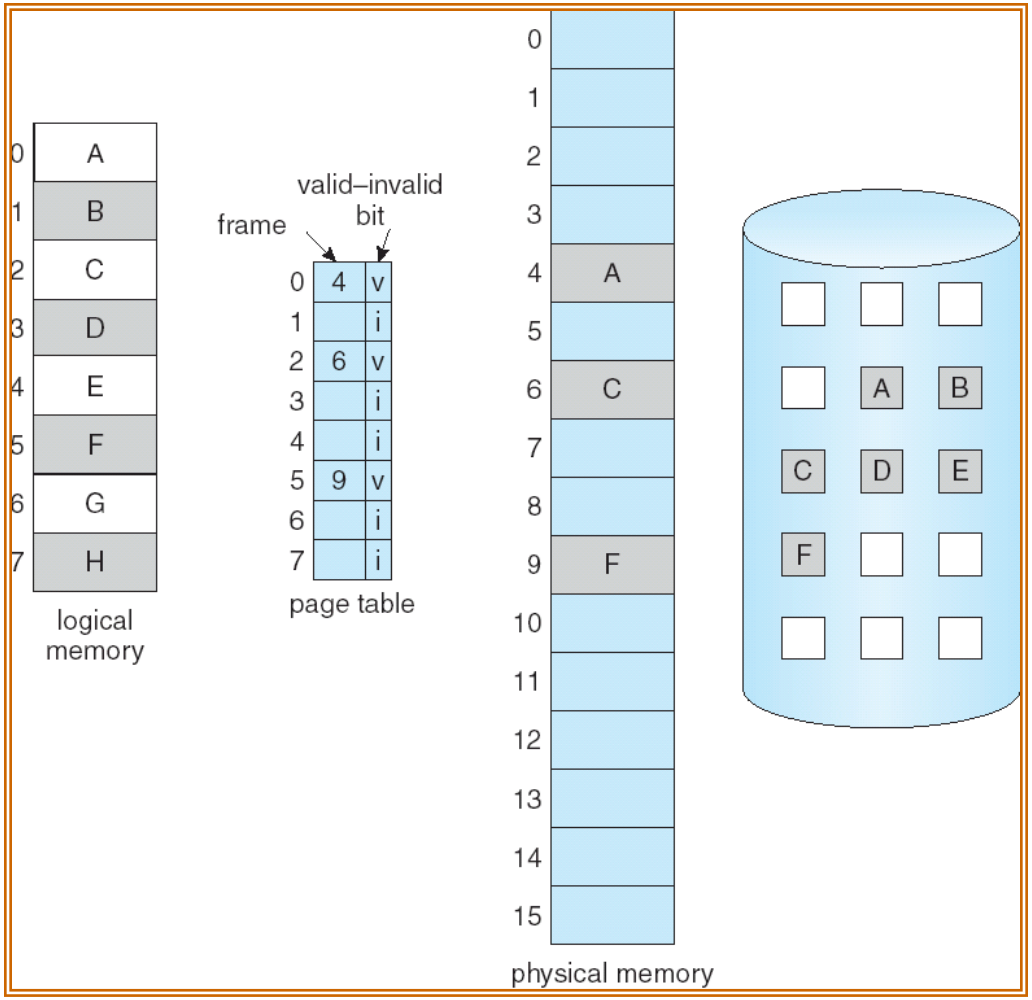
- Code needs to be in memory to execute, but entire program rarely used
  - Error code, unusual routines, large data structures
- Entire program code not needed at same time
- Consider ability to execute partially-loaded program
  - Program no longer constrained by limits of physical memory
  - Each program takes less memory while running -> more programs run at the same time
    - Increased CPU utilization and throughput with no increase in response time or turnaround time
  - Less I/O needed to load or swap programs into memory -> each user program runs faster

Virtual Memory

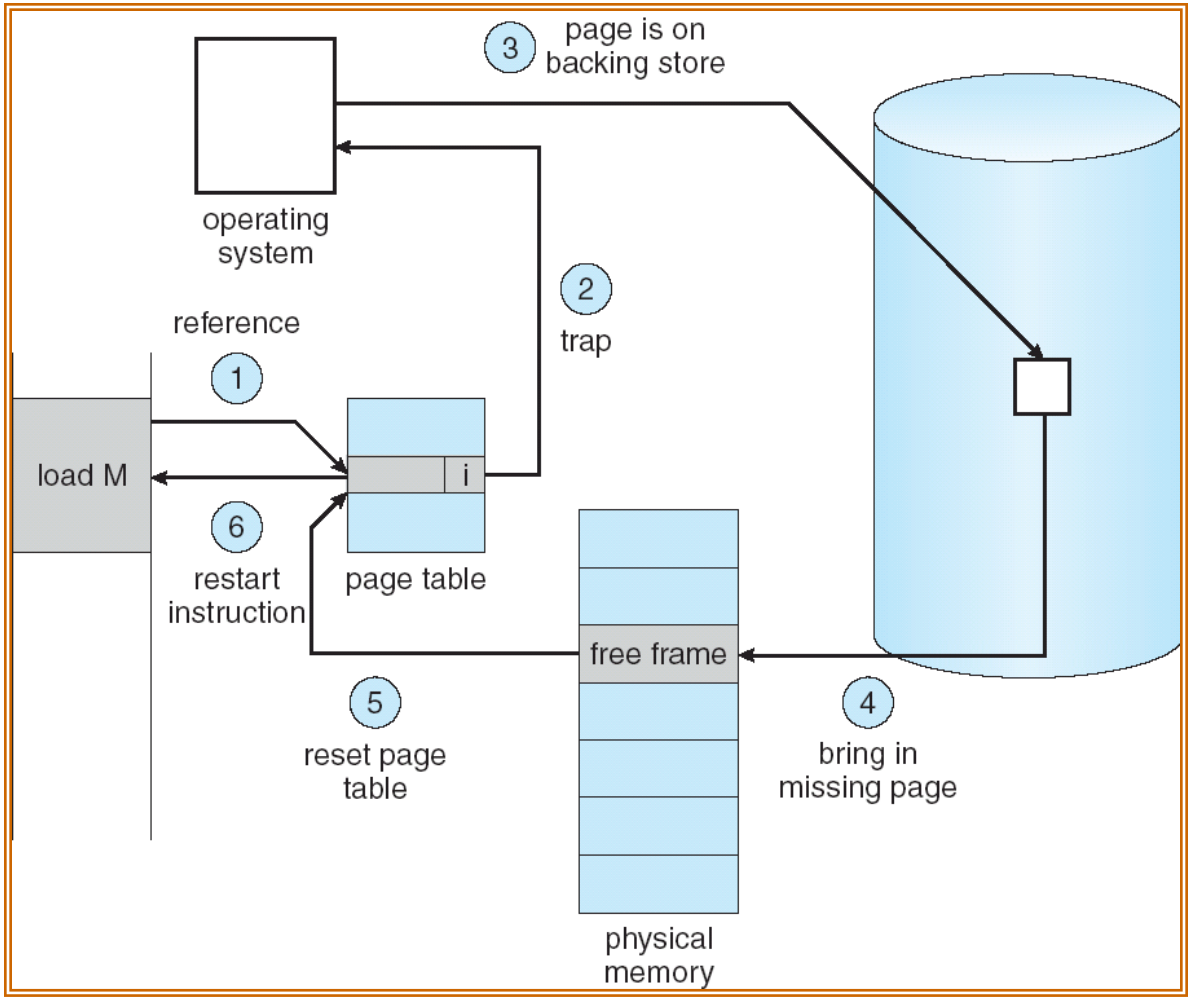
Virtual memory – separation of user logical memory from physical memory

- 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
- Allows for more efficient process creation
- More programs running concurrently
- Less I/O needed to load or swap processes

Page Table when some pages are not in Memory



Steps in Handling Page Fault



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

Extreme case – start process with *no* pages in memory

- OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault
- And for every other process pages on first access
- **Pure demand paging**

Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user 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 on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted 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 = (1 – p) x memory access
  + p (page fault overhead
  + [swap page out]
  + swap page in
  + restart overhead)

  EAT = (1 – p) x memory access time without page fault
  + p x (page fault service time)

More accurate

EAT = (1 – p) x memory access time without page fault

+ p x (memory access time + page fault service time)

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
- If one access out of 1,000 causes a page fault, then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!

Question 1

Let the page fault service time be 10 ms in a computer with average memory access time being 20 ns. If one page fault is generated for every 10^6 memory accesses, what is the effective access time for the memory?

A. 21 ns

B. 30 ns

C. 23 ns

D. 35 ns

Question 2

Suppose the time to service a page fault is on the average 10 milliseconds, while a memory access takes 1 microsecond. Then, a 99.99% hit ratio results in average memory access time of-

A. 1.9999 milliseconds
B. 1 millisecond
C. 9.999 microseconds
D. 1.9999 microseconds
E. None of these

Correct answer is E (2 microseconds)

Note: Sometimes you may not get exact option and none of these is also not mentioned then you can choose the closest one because it will be solved neglecting memory access time then answer is 1.999 microseconds

EAT with TLB but without page fault

In a multilevel paging scheme using TLB without any possibility of page fault, effective access time is given by:

Effective Access Time (without page faults) =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + (L+1) x Access time of main memory }

where L = Number of levels of page table

Question 3

Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%, the effective memory access time is _____ ns.

Consider a two level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%, the effective memory access time is _____ ns.

Question

Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB. If TLB hit ratio is 50% and effective memory access time is 170 ns, main memory access time is _____.

Ans 100 ns

Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If effective memory access time is 130 ns, TLB hit ratio is _____.

Ans 0.9 or 90%

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use *Modify (Dirty) Bit* to reduce overhead of page transfers – only modified pages are written to disk

Basic Page Replacement

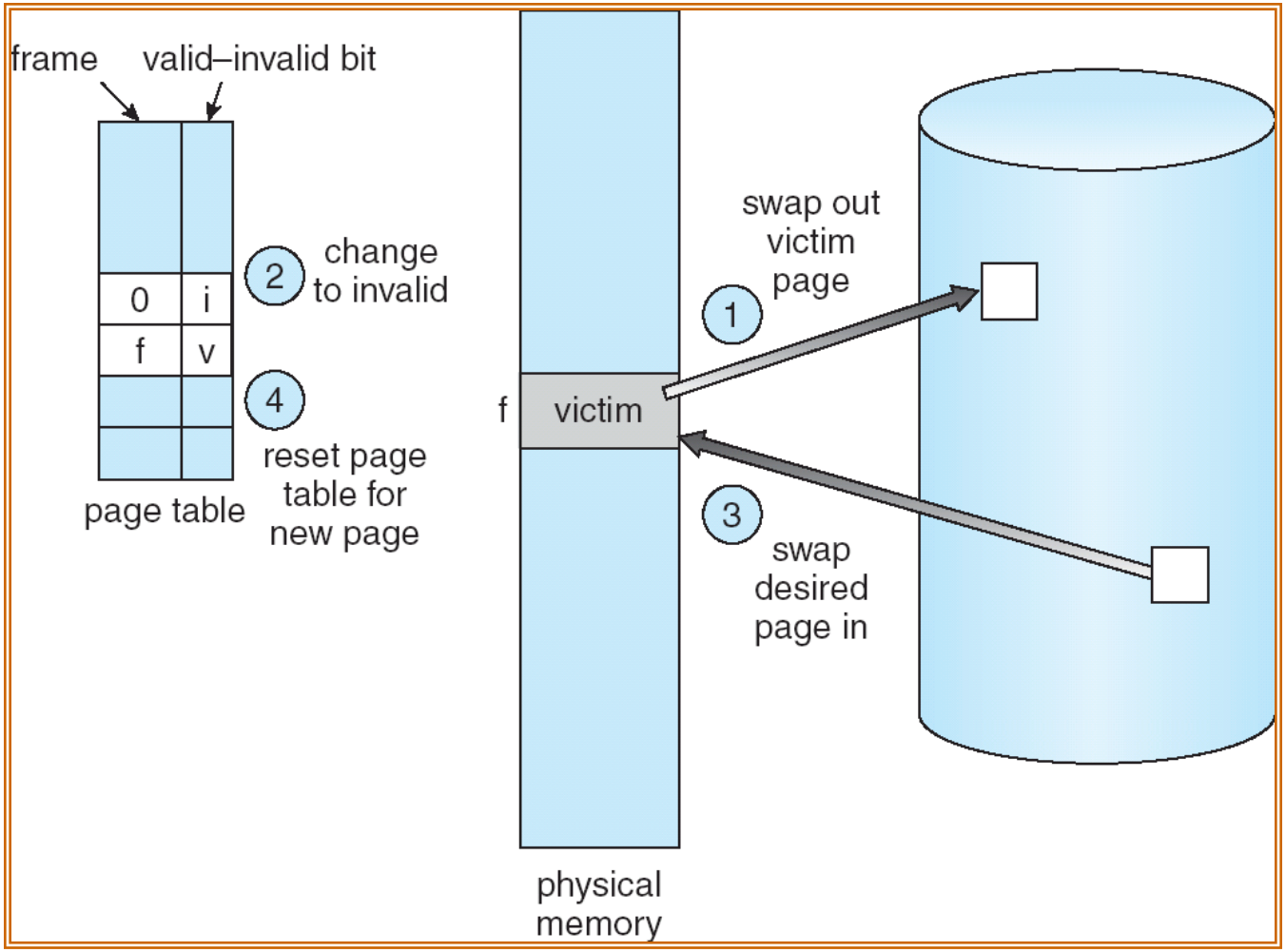
1. Find the location of the desired page on disk

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. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process

Page Replacement



Question 4

TLB lookup time = 20 ns
TLB hit ratio = 80%
Memory access time = 75 ns
Swap page time = 500,000 ns
50% of pages are dirty
OS uses a single level page table
What is the effective access time (EAT) if we assume the page fault rate is 10%? Page fault occurs when there is TLB miss. Assume the cost to update the TLB, the page table, and the frame table (if needed) is negligible.

Ans 15110 nanoseconds

Question 5

Consider a paging system that uses 1-level page table residing in main memory and a TLB for address translation. Each main memory access takes 100 ns and TLB lookup takes 20 ns. Each page transfer to/from the disk takes 5000 ns. Assume that the TLB hit ratio is 95%, page fault rate is 10%. Assume that for 20% of the total page faults, a dirty page has to be written back to disk before the required page is read from disk. Assume the cost to update the TLB and the page table is negligible. Page fault occurs when there is TLB miss. The average memory access time in ns (round off to 1 decimal places) is _____

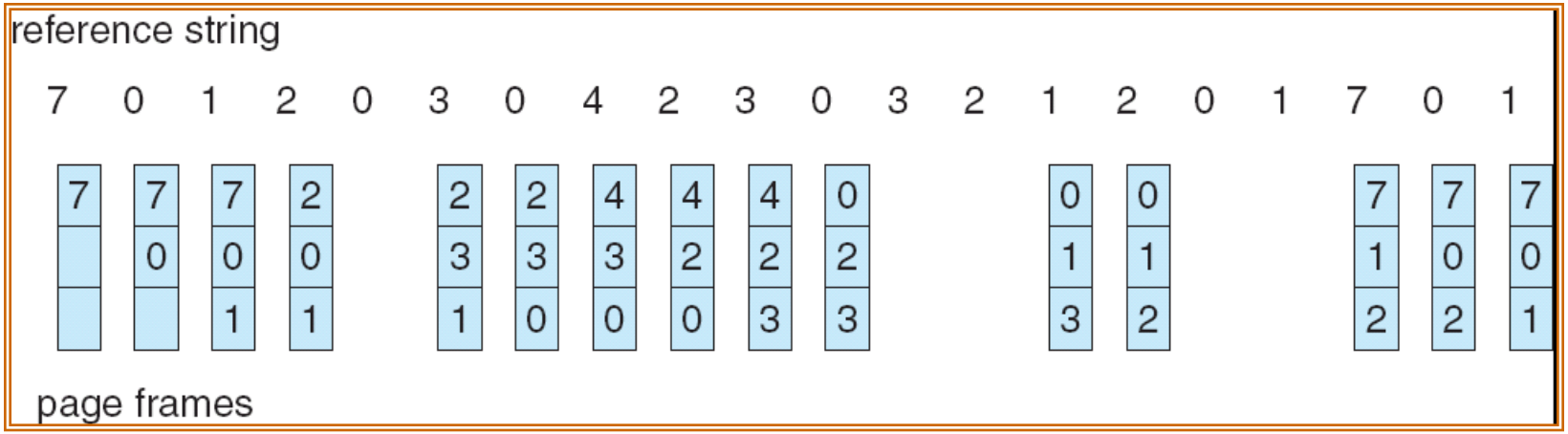
Ans 155 ns

Question 6

Suppose we have a demand-paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault without a dirty-page write, and 20 milliseconds with dirty-page write. Memory access time is 100 nanoseconds. Assume that the dirty-page write occurs on 70 percent of all page faults. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds? **Assume page fault service time includes memory access time also.**

Ans: 0.000006

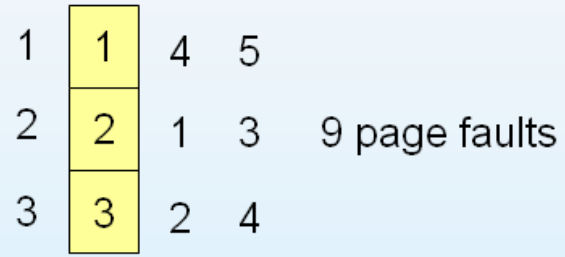
FIFO Page Replacement



15 page faults

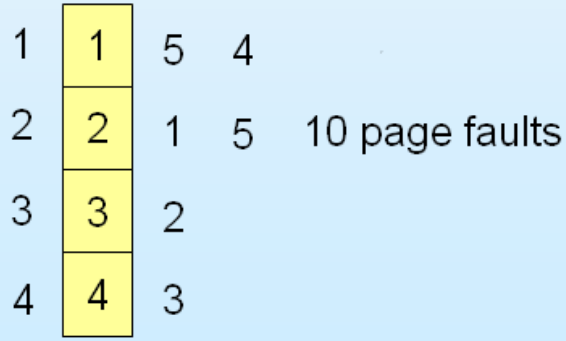
FIFO – First In First Out

- 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)



9 page faults

- 4 frames

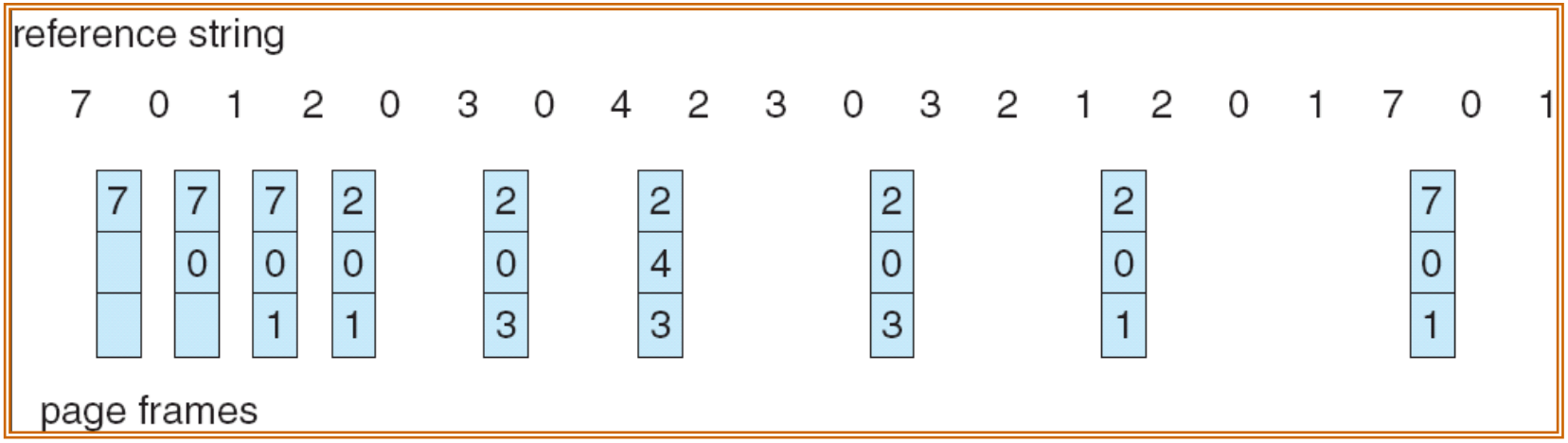


10 page faults

- FIFO Replacement – Belady's Anomaly
  - more frames ⇒ more page faults

Optimal Page Replacement

- Replace page that will not be used for longest period of time
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

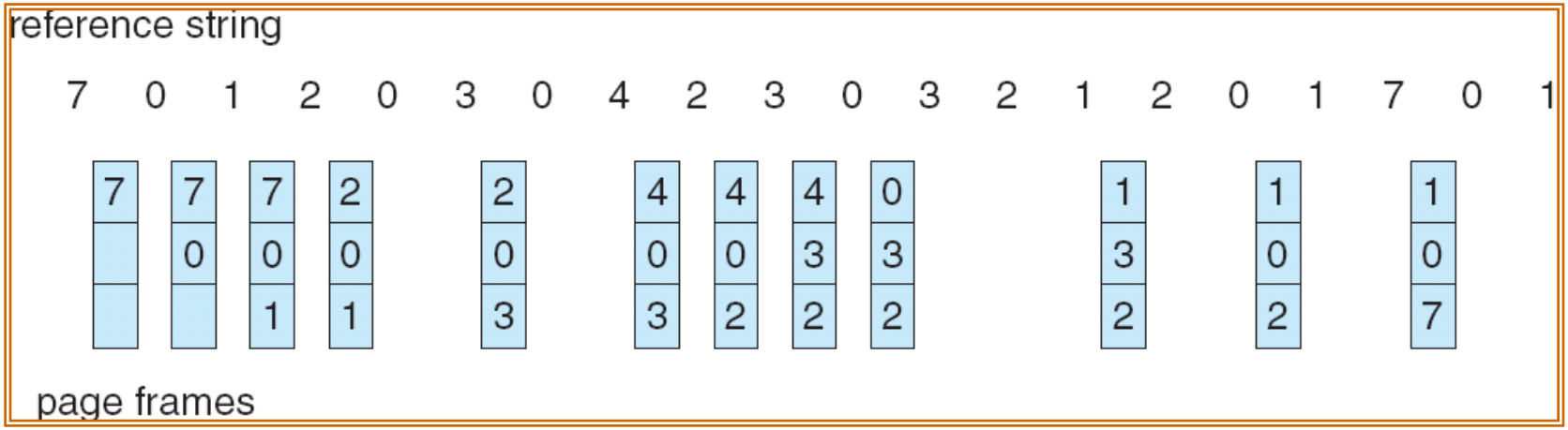


9 page faults

LRU Page Replacement

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page
- Generally good algorithm and frequently used

  But how to implement?

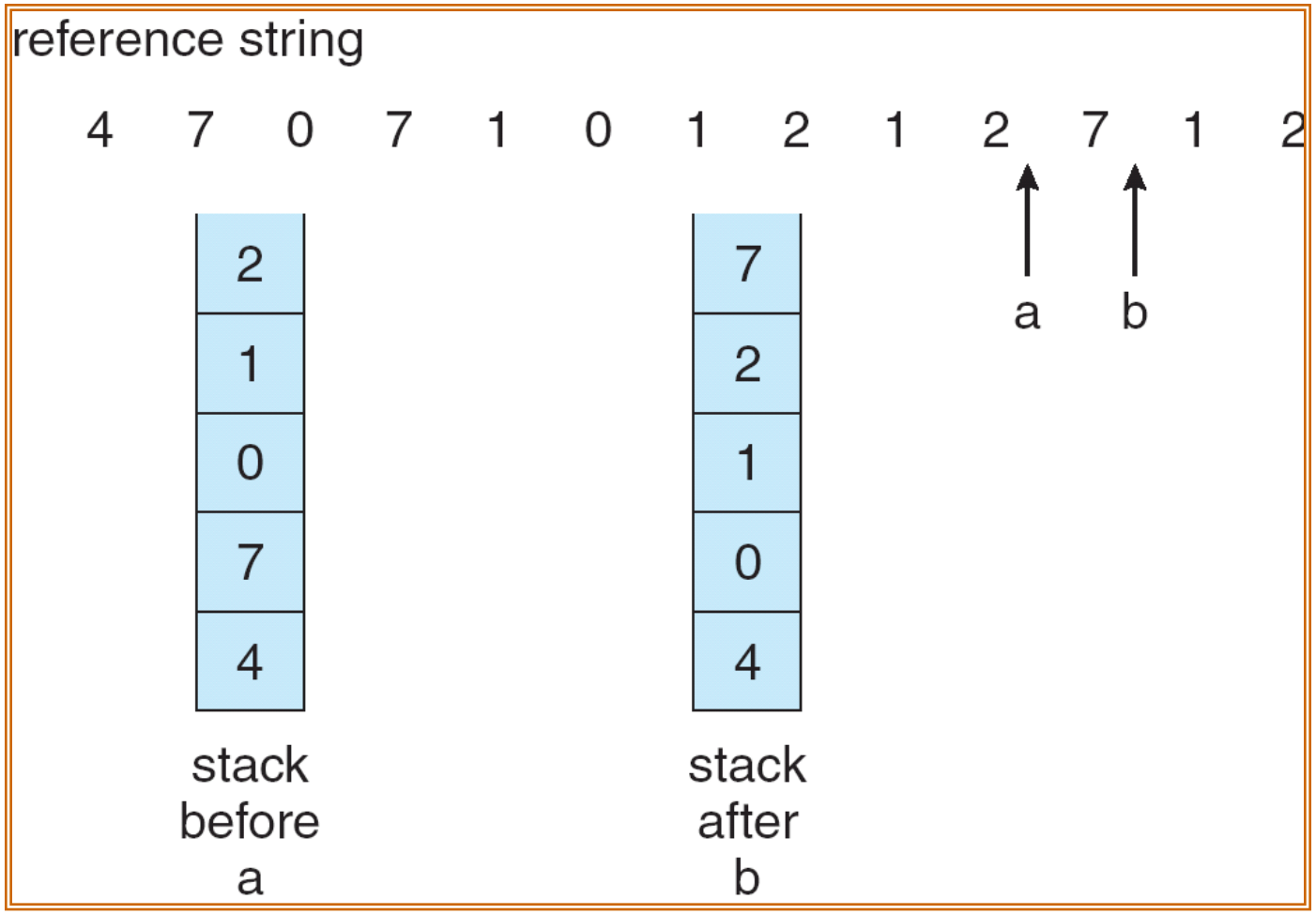


12 faults – better than FIFO but worse than OPT

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 find smallest value
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - But each update more expensive
  - No search for replacement
- LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

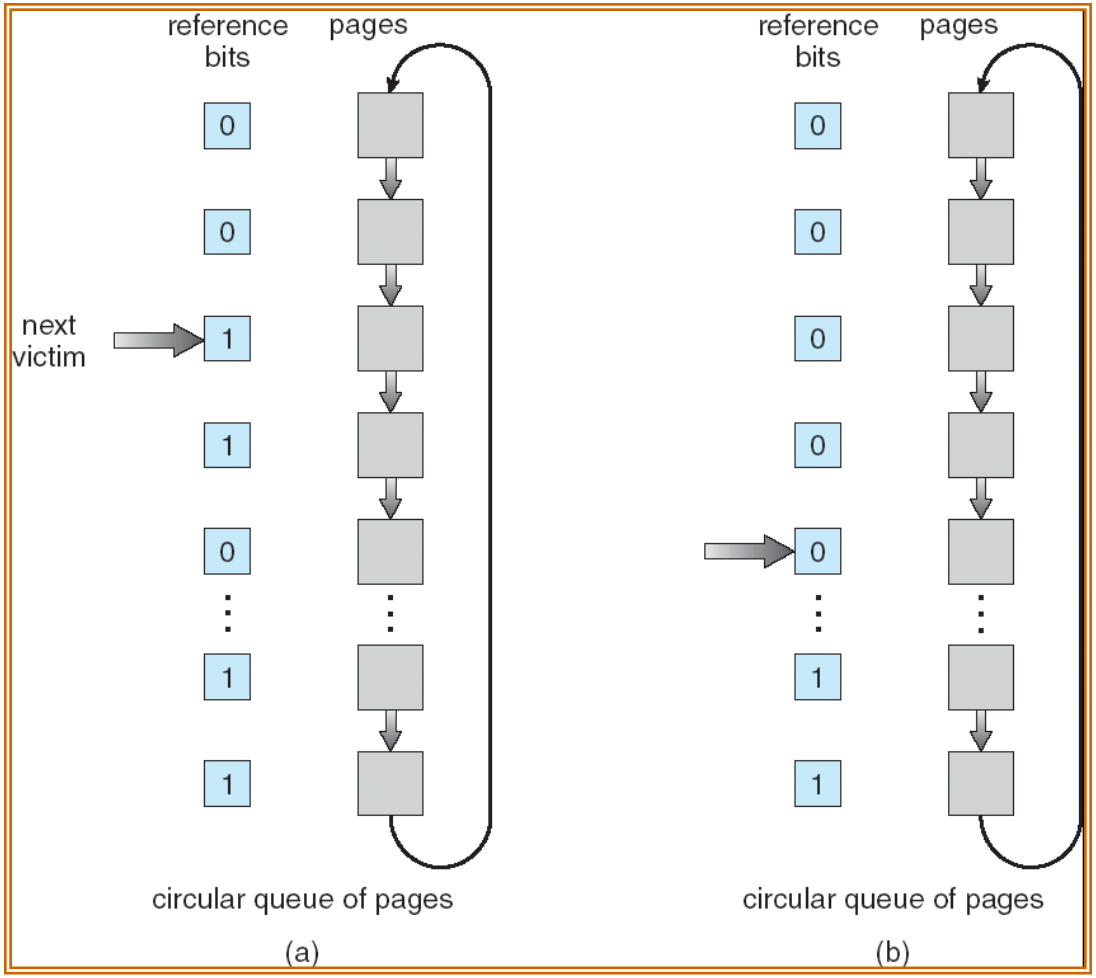
Use of Stack to record the most recent Page Reference



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Additional Reference bits
  - Every time a page is referenced, shift the reference bits to the right by 1
  - Place the reference bit (1 if being visited, 0 otherwise) into the high order bit of the reference bits
  - The page with the lowest reference bits value is the one that is Least Recently Used, thus to be replaced. E.g., the page with ref bits 11000100 is more recently used than the page with ref bits 01110111
- Second chance or clock
  - Need reference bit
  - If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second Chance (Clock) Page Replacement Algorithm



3, 2, 3, 0, 8, 4, 2, 5, 0, 9, 8, 3, 2

5 frames

Try this!

Frames 3

Reference string

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

Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available)
- Take ordered pair (reference, modify)

1. (0, 0) neither recently used not modified – best page to replace
2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
3. (1, 0) recently used but clean – probably will be used again soon
4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement

- When page replacement called for, use the clock scheme but use the four classes to replace page in lowest non-empty class
  - Might need to search circular queue several times

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: Fixed Allocation

- Equal allocation – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- Proportional allocation – Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 62$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 4$

$a_2 = \frac{127}{137} \times 62 \approx 57$

Allocation of Frames: 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

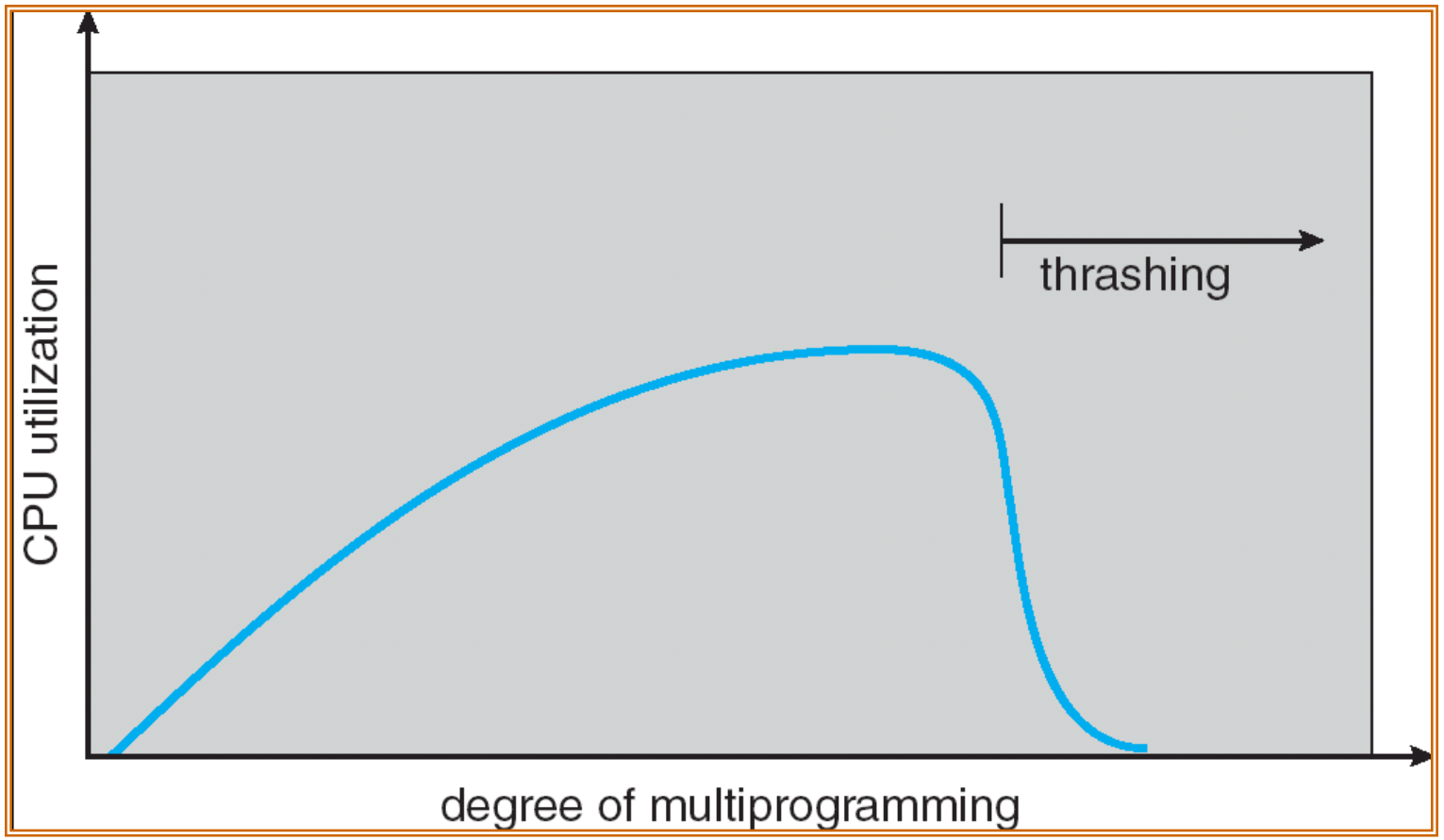
Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

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 (contd.)



Question 7

Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1. On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Calculate number of page faults under the LRU, FIFO and OPTIMAL page replacements policy.

Question 8

A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below? 4, 7, 6, 1, 7, 6, 1, 2, 7, 2