# CSE 4/521 Introduction to Operating Systems

Lecture 17 – Virtual Memory III

(Page Replacement Algorithms)

Summer 2018

# Overview

Objective:

- To understand different page replacement algorithms.

- Page Replacement Algorithms

# Recap

- Demand Paging Example
  - Basic functionality, Valid-Invalid Bit, Page Fault, Demand Paging Example
- Copy-on-Write
  - Copy on if write has taken place.

# Questions

1. What is copy-on-write? And why is it useful? (Easy)

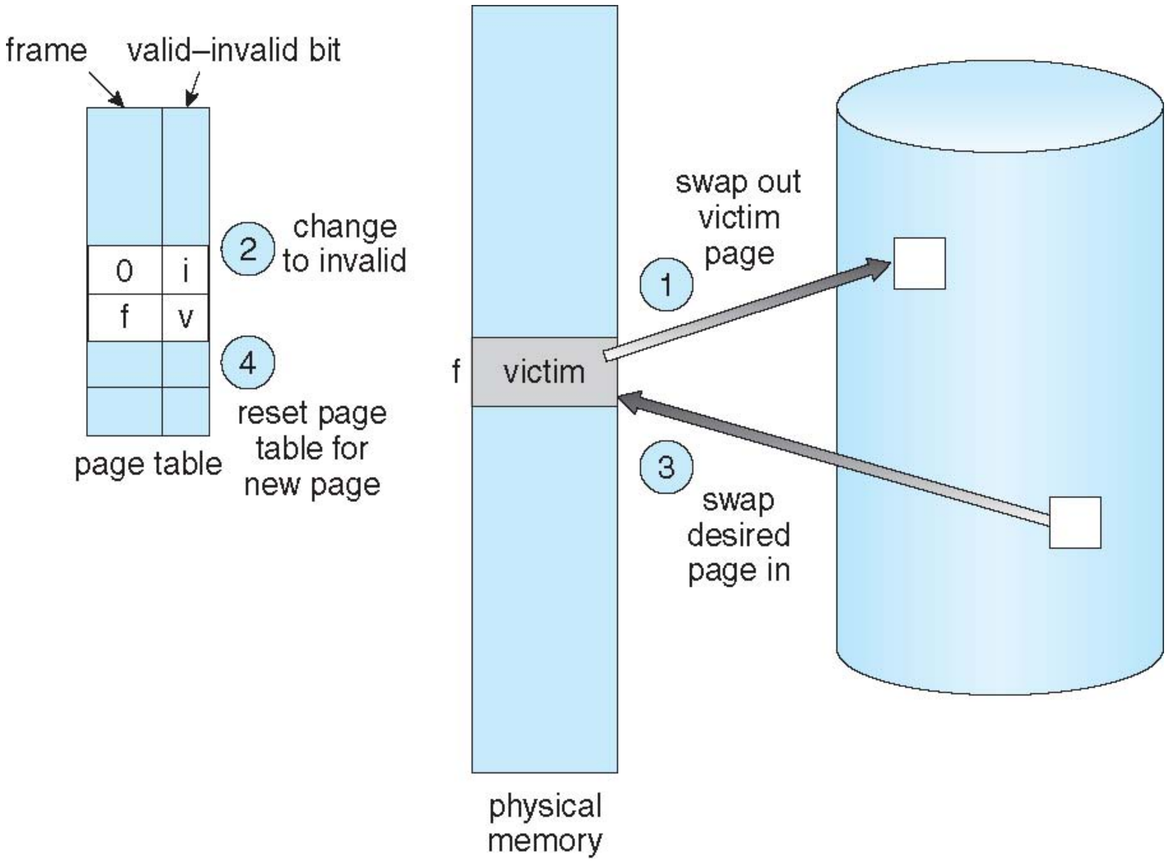
2. What is the maximum limit for virtual memory? (Easy)

3. What is the optimum number of page-faults to perceive no latency in real-time gaming environments? (Open-ended)

# Overview

- Page Replacement Algorithms

# Page Replacement

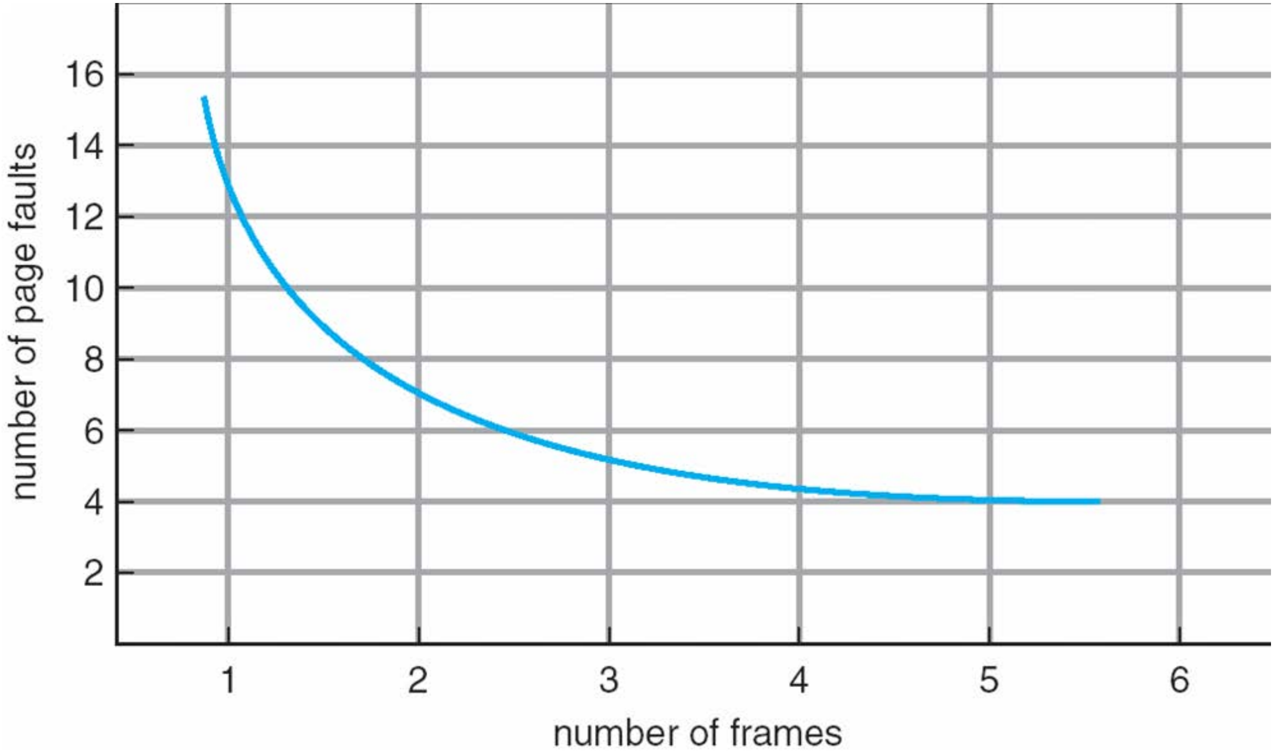


# Page Replacement : Page and Frame Replacement Algorithms

- Page-replacement algorithm
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a reference string and computing the number of page faults on that string
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the reference string of referenced page numbers is

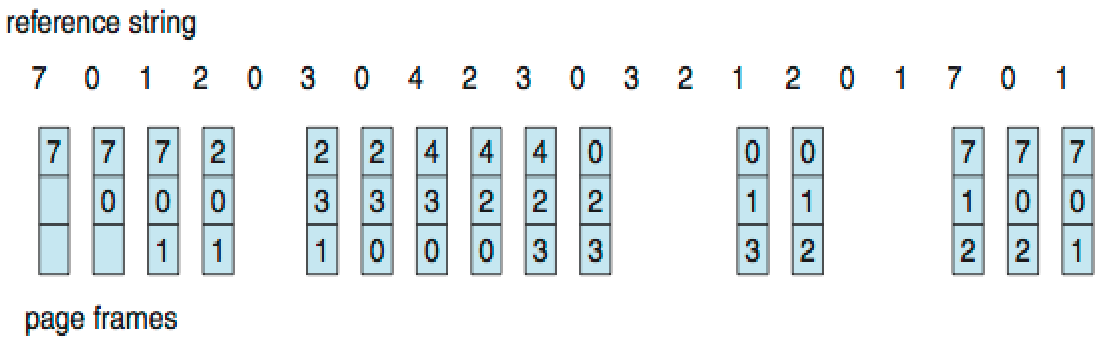
**7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

# Page Replacement :
# Graph of Page Faults vs. The Number of Frames



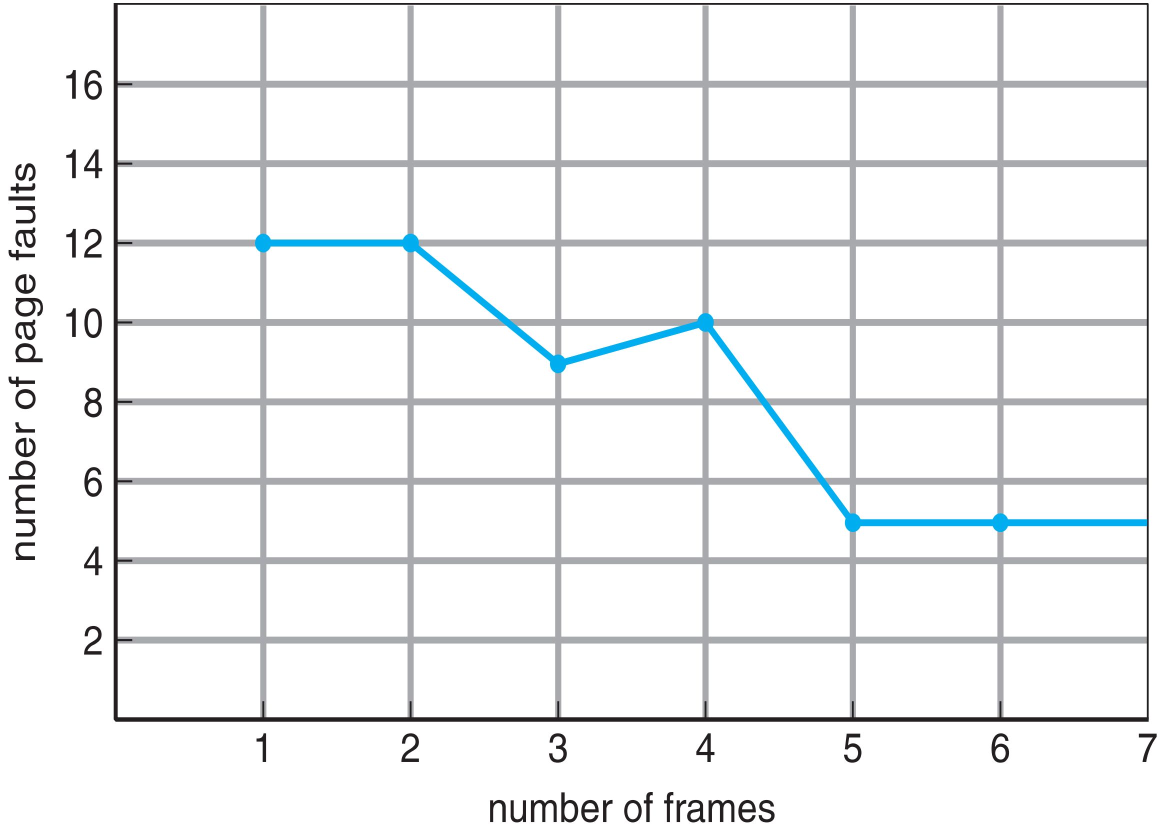
# Page Replacement : First-In-First Out (FIFO) Algorithm

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



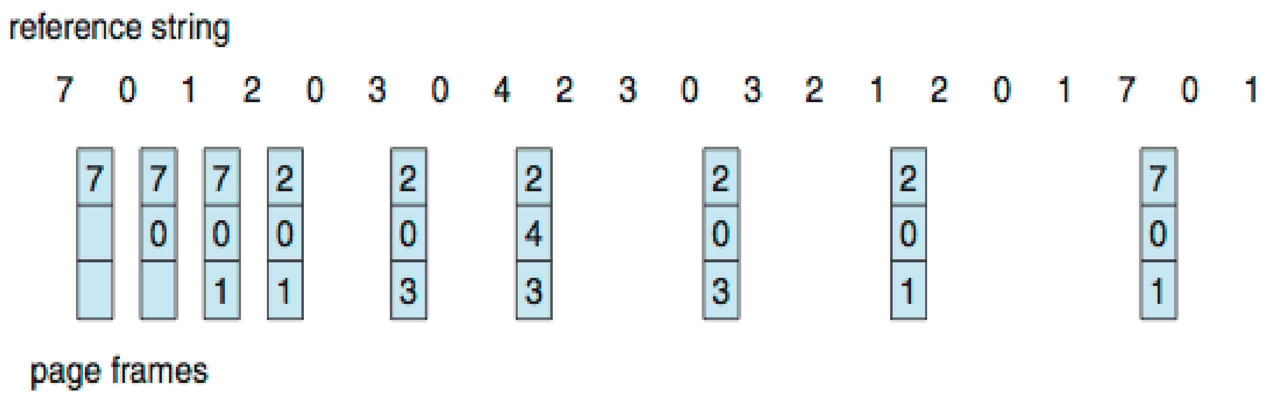

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - Belady's Anomaly
- How to track ages of pages?
  - Just use a FIFO queue

# Page Replacement : FIFO Illustrating Belady's Anomaly



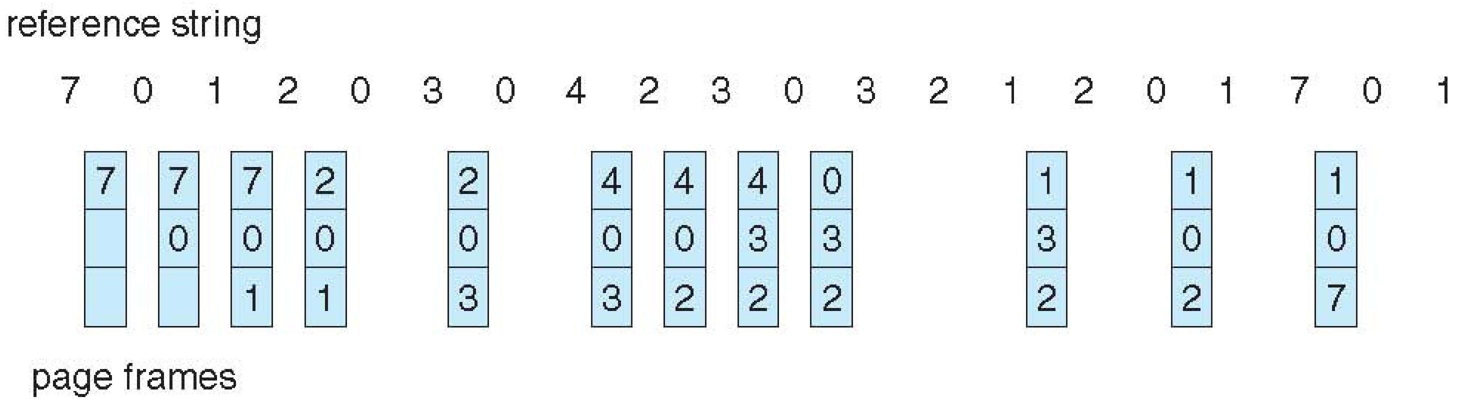
# Page Replacement : Optimal Algorithm

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



# Page Replacement : Least Recently Used (LRU) Algorithm

- 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

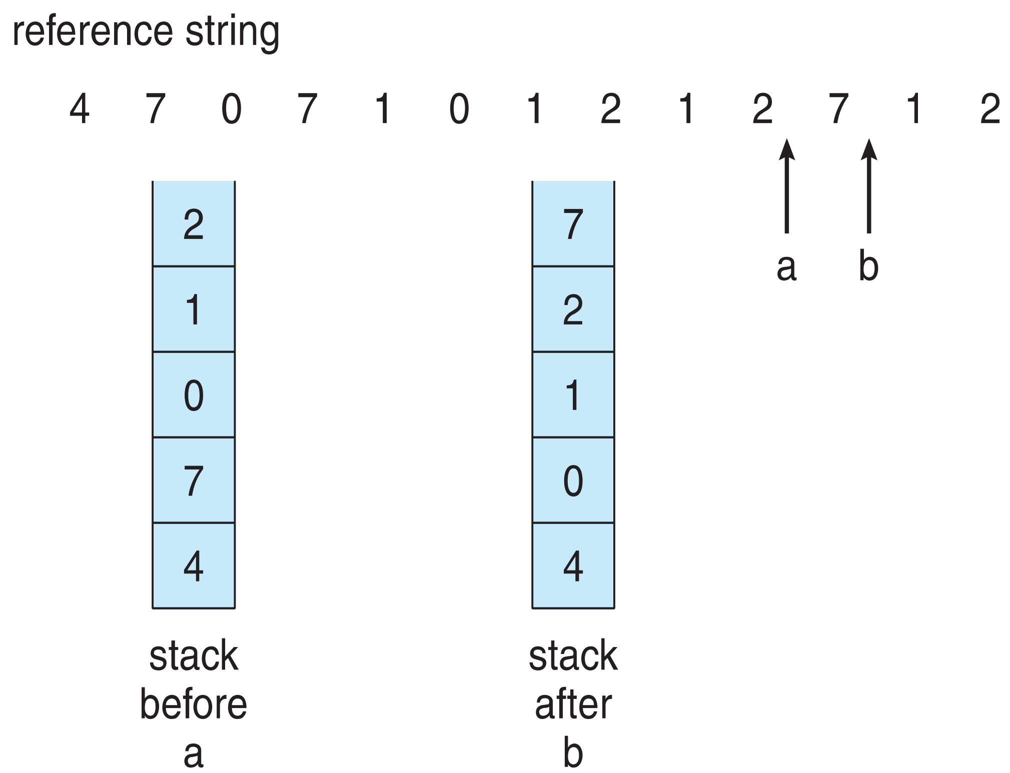



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

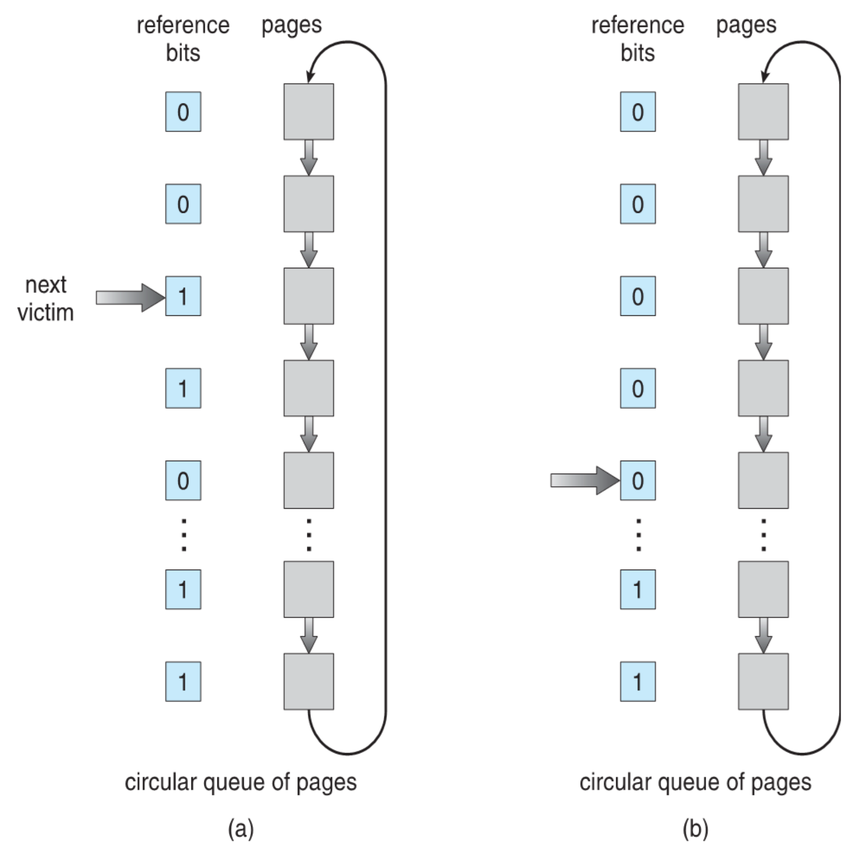
# Page Replacement : LRU Algorithm

- 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

# Page Replacement : Use of a stack to record most recent page reference



# Page Replacement : Second-Chance (clock) Page-Replacement Algorithm



# Page Replacement : Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- 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 replace page in lowest non-empty class
  - Might need to search circular queue several times

# Page Replacement : Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- Lease Frequently Used (LFU) Algorithm: replaces page with smallest count
- Most Frequently Used (MFU) Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

# Page Replacement : Page-Buffering Algorithms

- Keep a pool of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Possibly, keep list of modified pages
  - When backing store otherwise idle, write pages there and set to non-dirty
- Possibly, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

# Page Replacement : Applications and Page Replacement

- All of these algorithms have OS guessing about future page access
- Some applications have better knowledge – i.e. databases
- Memory intensive applications can cause double buffering
  - OS keeps copy of page in memory as I/O buffer
  - Application keeps page in memory for its own work
- Operating system can given direct access to the disk, getting out of the way of the applications
  - Raw disk mode
- Bypasses buffering, locking, etc

# Credits for slides

Silberschatz, Galvin and Gagne