CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 32

Virtual Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

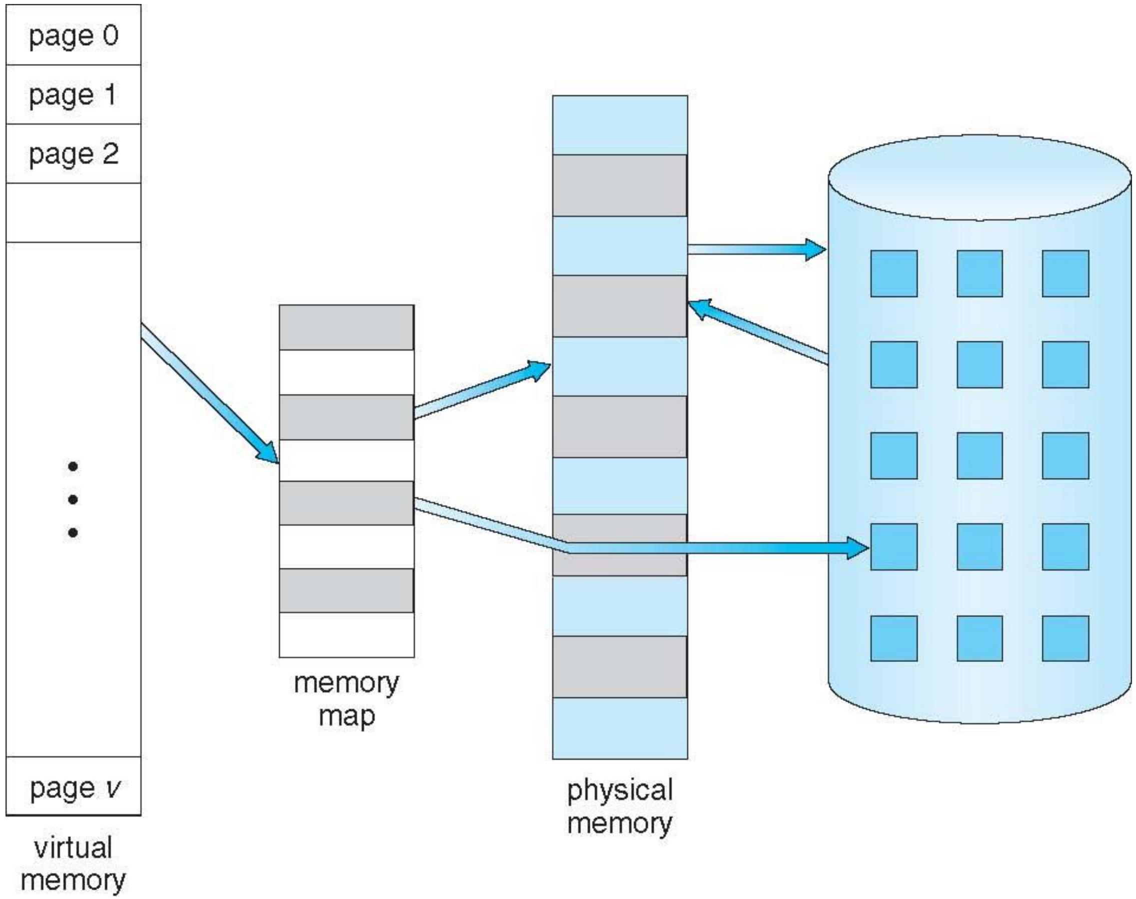
Questions for you

- What is disk space is full, physical memory is full, and the user launches a process?
- If physical memory (RAM) gets to be very big, do accesses to disk reduce?
- Is there ever a case where adding more memory does not help?

Technical Perspective: Multiprogramming

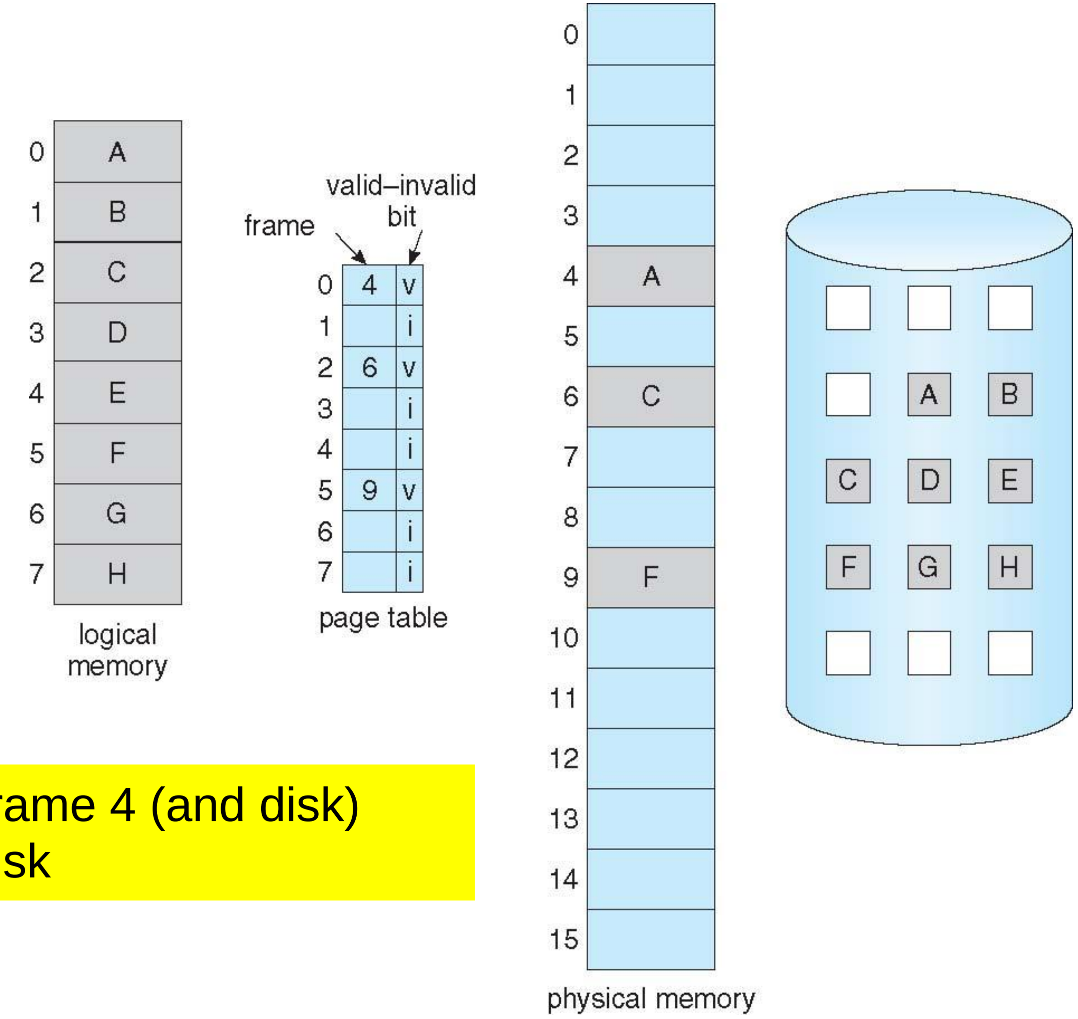
- Contiguous allocation. Problem: external fragmentation
- Non-contiguous, but entire process in memory: Problem: Memory occupied by stuff needed only occasionally. Low degree of Multiprogramming.
- Demand Paging: Problem: page faults
- How to minimize page faults?

Virtual Memory That is Larger Than Physical Memory



Colorado State University

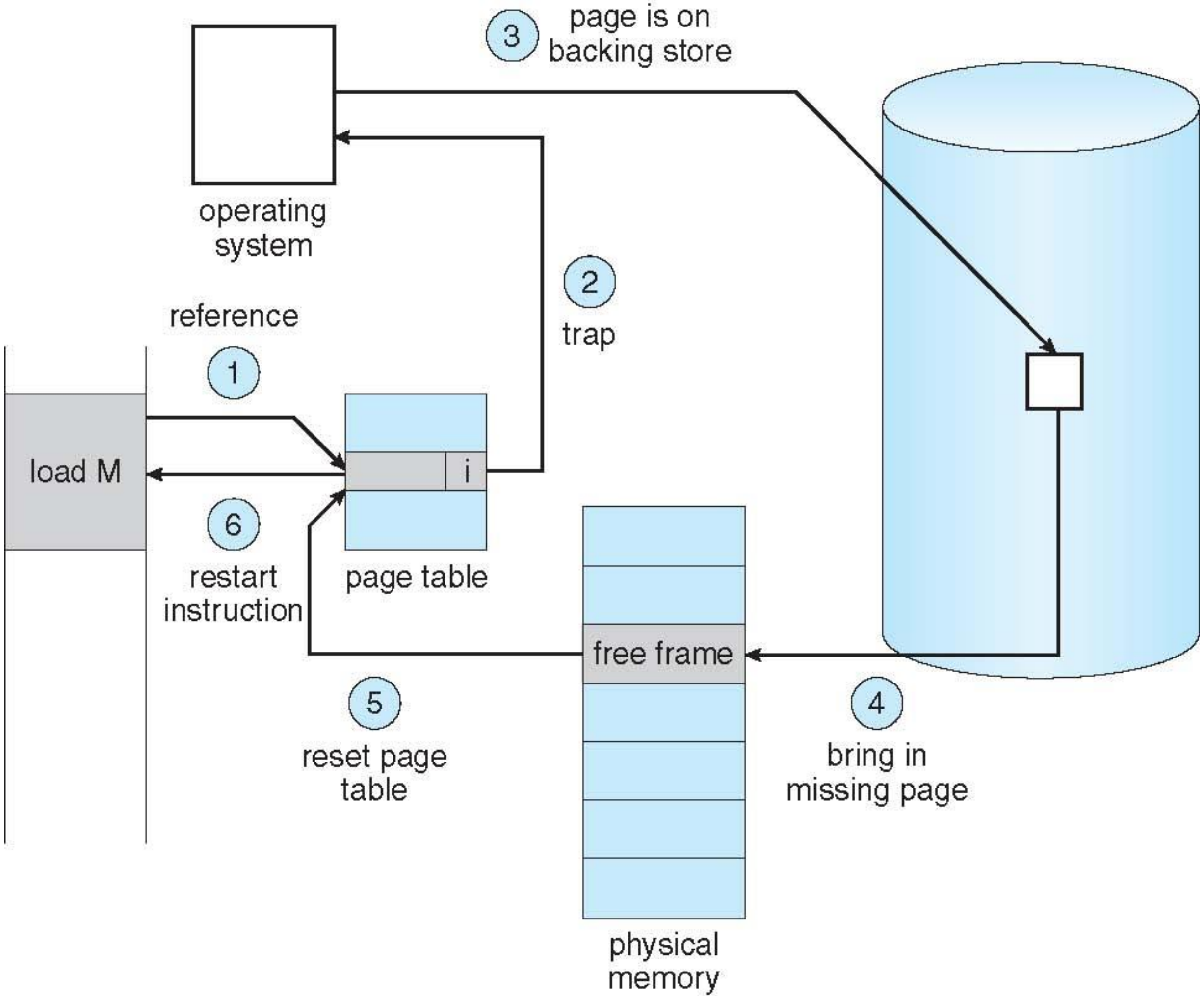
Page Table When Some Pages Are Not in Main Memory



Page 0 in Frame 4 (and disk)
Page 1 in Disk

Colorado State University

Steps in Handling a Page Fault



Performance of 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 (Cont.)

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read the page – lots of time
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)

Hopefully p <<1

EAT = $(1 – p)$ x memory access time
+ p (page fault overhead
+ swap page out + swap page in)

Page swap time = seek time + latency time

Colorado State University

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 nanosec.

  = 200 + p x 7,999,800 ns

  Linear with page fault rate

- 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!!

- If want performance degradation < 10 percent, **p = ?**
  - 220 > 200 + 7,999,800 x p
    20 > 7,999,800 x p
  - p < .0000025
  - < one page fault in every 400,000 memory accesses

Issues: Allocation of physical memory to I/O and programs

- Memory used for holding **program** pages
- **I/O buffers** also consume a big chunk of memory
- Solutions:
  - Fixed percentage set aside for I/O buffers
  - Processes and the I/O subsystem compete

Demand paging and the limits of logical memory

- Without demand paging
  - All pages of process **must be** in physical memory
  - Logical memory **limited** to size of physical memory
- With demand paging
  - All pages of process **need not be** in physical memory
  - Size of logical address space is **no longer constrained** by physical memory
- Example
  - 40 pages of physical memory
  - 6 processes each of which is 10 pages in size
    - Each process only needs 5 pages as of now
  - Run 6 processes with 10 pages to spare

Higher degree of multiprogramming

Coping with over-allocation of memory

Example

- Physical memory = 40 pages
- 6 processes each of which is of size 10 pages
  - But are using 5 pages each as of now
- What happens if each process needs all 10 pages?
  - 60 physical frames needed
- **Terminate** a user process
  - But paging should be transparent to the user
- **Swap out** a process
  - Reduces the degree of multiprogramming
- **Page replacement:** selected pages. Policy? soon

What Happens if there is no Free Frame?

- Could be all used up by process pages or
- kernel, I/O buffers, etc
  - How much to allocate to each?
- Page replacement – find some page in memory, but not really in use, page it out
  - Algorithm – terminate? swap out? replace the page?
  - Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Continued to Page replacement etc...

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
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Basic Page Replacement

1. Find the location of the desired page on disk

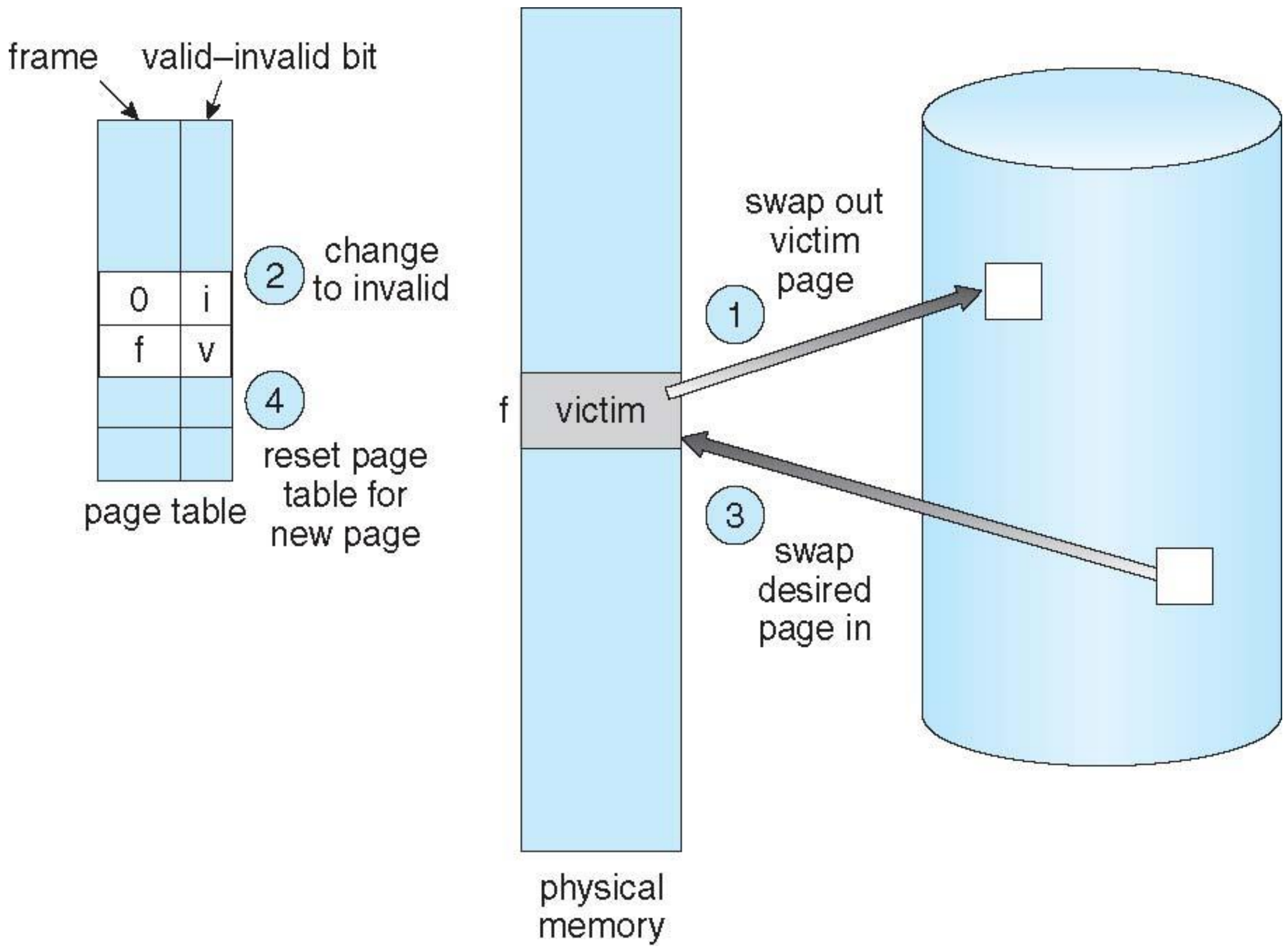
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty

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

4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Page Replacement



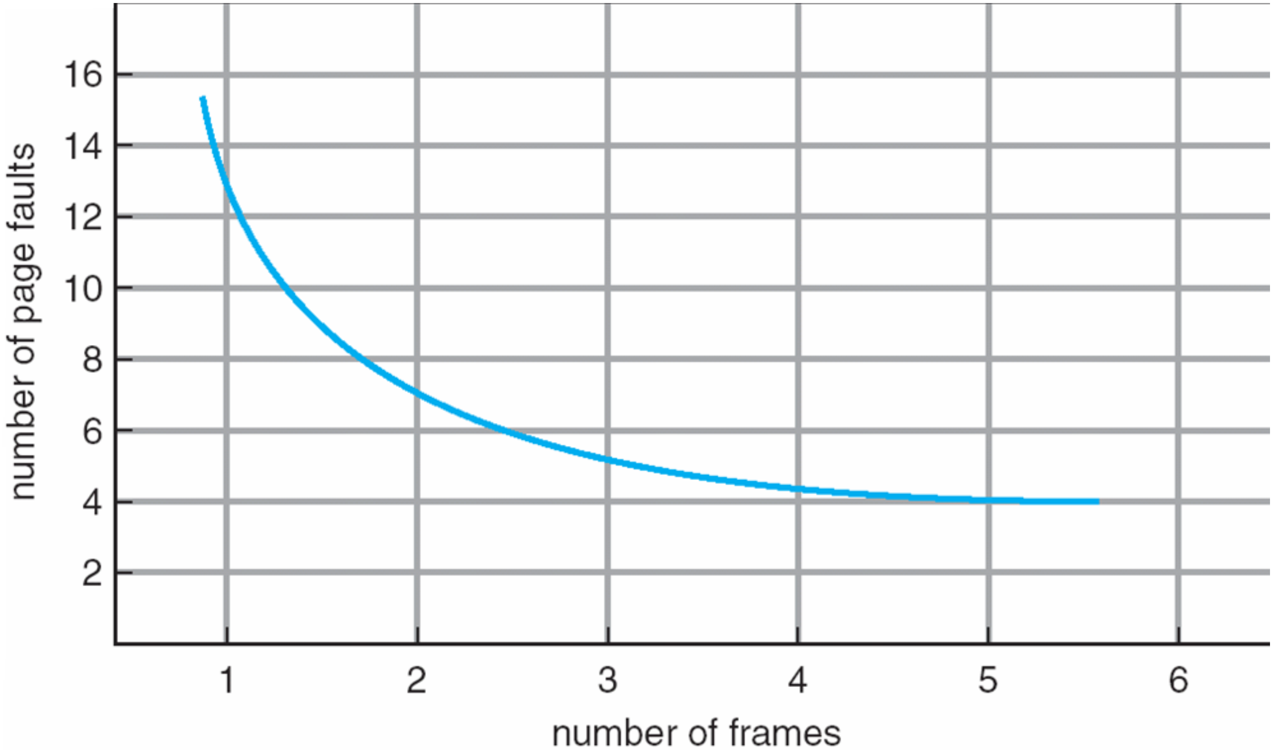
Colorado State University

Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
- **Page-replacement algorithm**
  - Which frames to replace
  - 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
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, we use 3 frames and 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

Graph of Page Faults Versus The Number of Frames



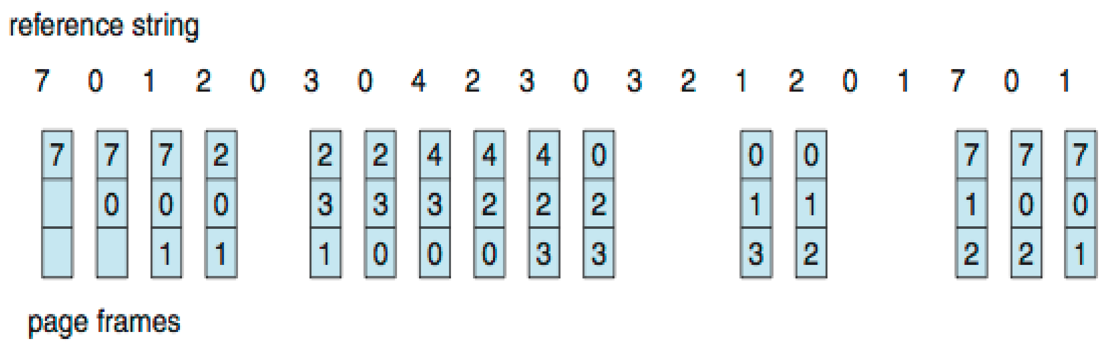
Colorado State University

FIFO page replacement algorithm: Out with the old; in with the new

- When a page must be replaced
  - Replace the oldest one
- OS maintains list of all pages currently in memory
  - Page at head of the list: Oldest one
  - Page at the tail: Recent arrival
- During a page fault
  - Page at the head is removed
  - New page added to the tail

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



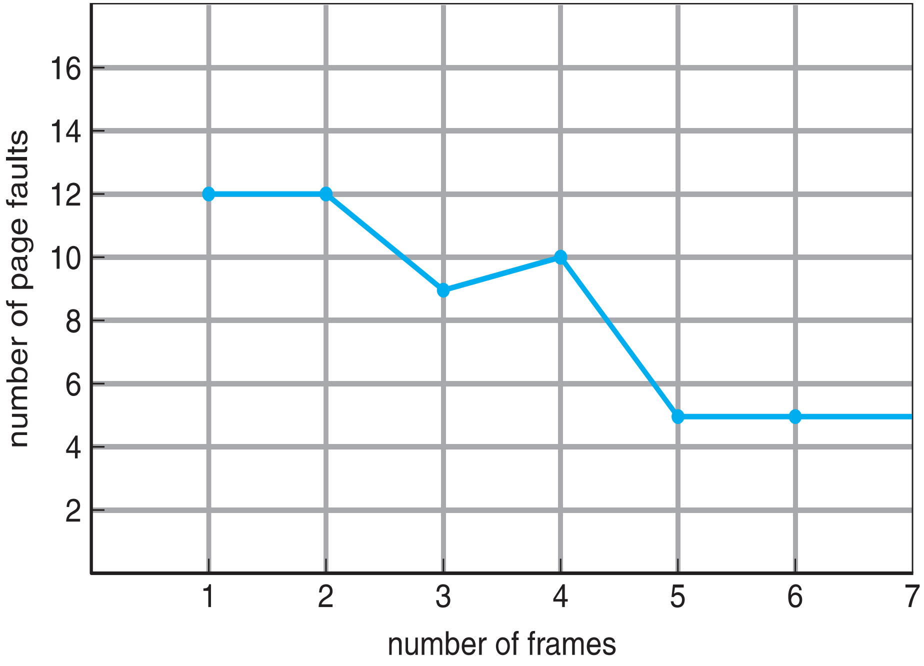
- 15 page faults
- Sometimes a page is needed soon after replacement 7,0,1,2,0,3,0, ..

Colorado State University

Belady's Anomaly

- Consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - **Belady's Anomaly**

Belady was here at CSU. Guest in my CS530!

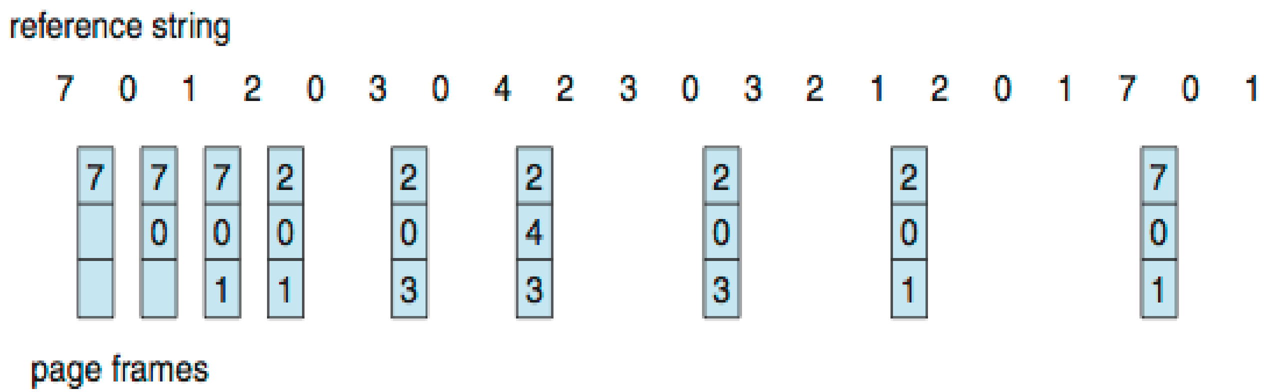


3 frames: 9 page faults
4 frames: 10 page faults

Colorado State University

"Optimal" Algorithm

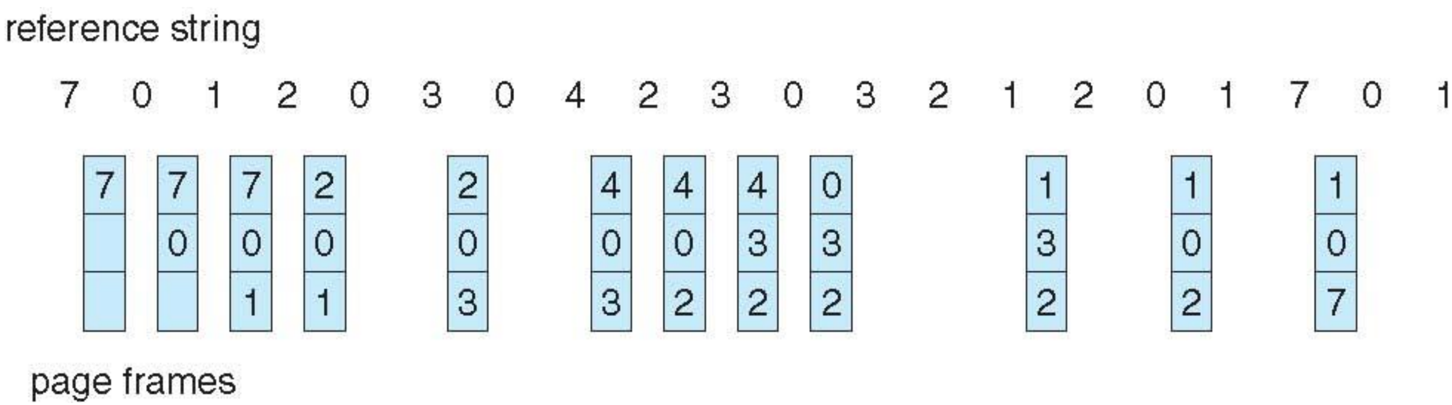
- Replace page that will not be used for longest period of time
  - 9 page replacements is optimal for the example
    - 4th access: replace 7 because we will not use if got the longest time…
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs



Colorado State University

Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time (4th access – page 7 is least recently used …_)
- 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?

Colorado State University