CSE 4/521 Introduction to Operating Systems

Lecture 16 – Virtual Memory II

(Copy-on-Write, Page Replacement)

Summer 2018

Overview

Objective:

- To explain the concept of copy-on-write.
- To understand common page replacement algorithms.

- Copy-on-Write
- Page Replacement

Recap

- Virtual Memory Background
  - Virtual Memory, Virtual Address Space, need for virtual memory.
- Demand Paging
  - Basic functionality, Valid-Invalid Bit, Page Fault, Demand Paging Example

Questions

1. What is a page fault? (Easy)

2. Assume that a program has just referenced an address in virtual memory. Describe a scenario in which each of the following can occur: (Medium)
   a. TLB miss with no page fault.
   b. TLB miss and page fault.
   c. TLB hit and no page fault.
   d. TLB hit and page fault.

3. Are you familiar with any other approach other than Demand Paging to facilitate virtual memory management? (Hard)

Demand Paging : Demand Paging Example

- Three major activities
  - Service the interrupt
  - Read the page
  - Restart the process
- 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)

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

Demand Paging : 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!!

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

Overview

- Copy-on-Write
- Page Replacement

Overview

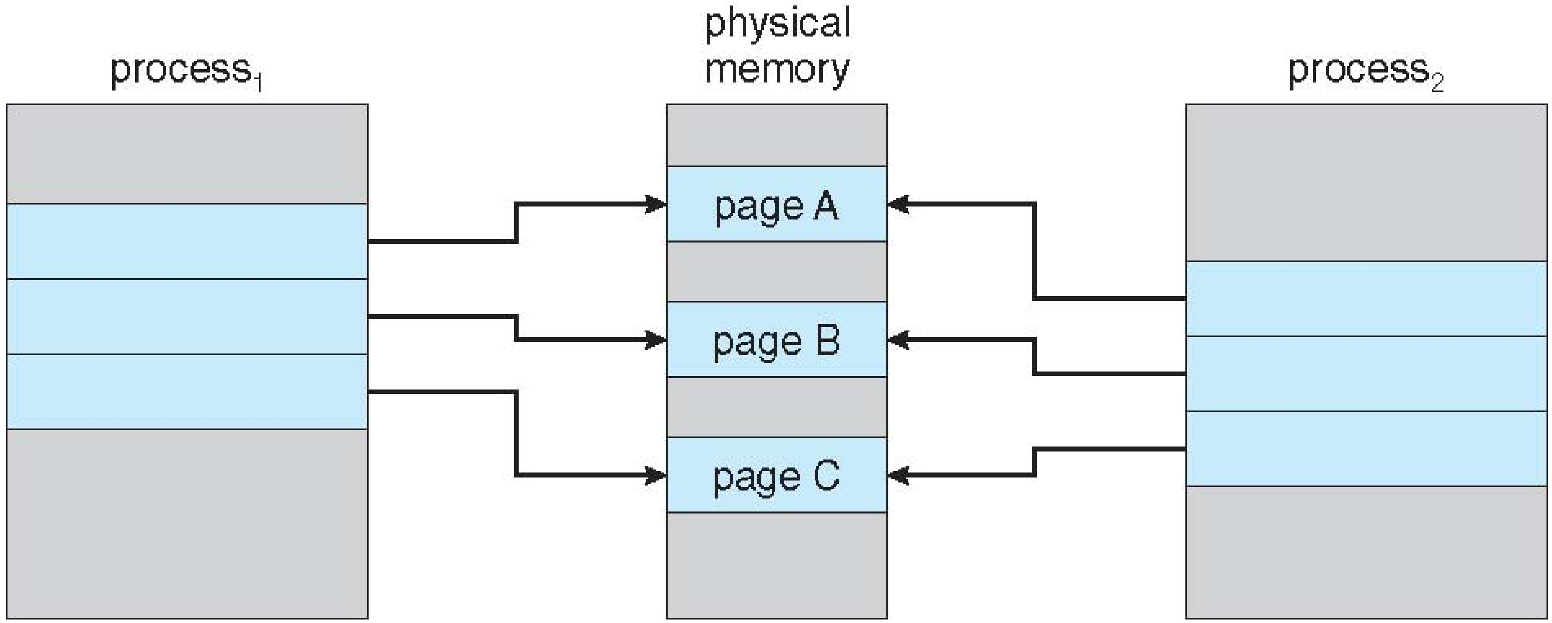
- Copy-on-Write
- Page Replacement

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a pool of zero-fill-on-demand pages
  - Pool should always have free frames for fast demand page execution
  - Why zero-out a page before allocating it?
- `vfork()` variation on `fork()` system call has parent suspend and child using copy-on-write address space of parent
  - Designed to have child call `exec()`
  - Very efficient

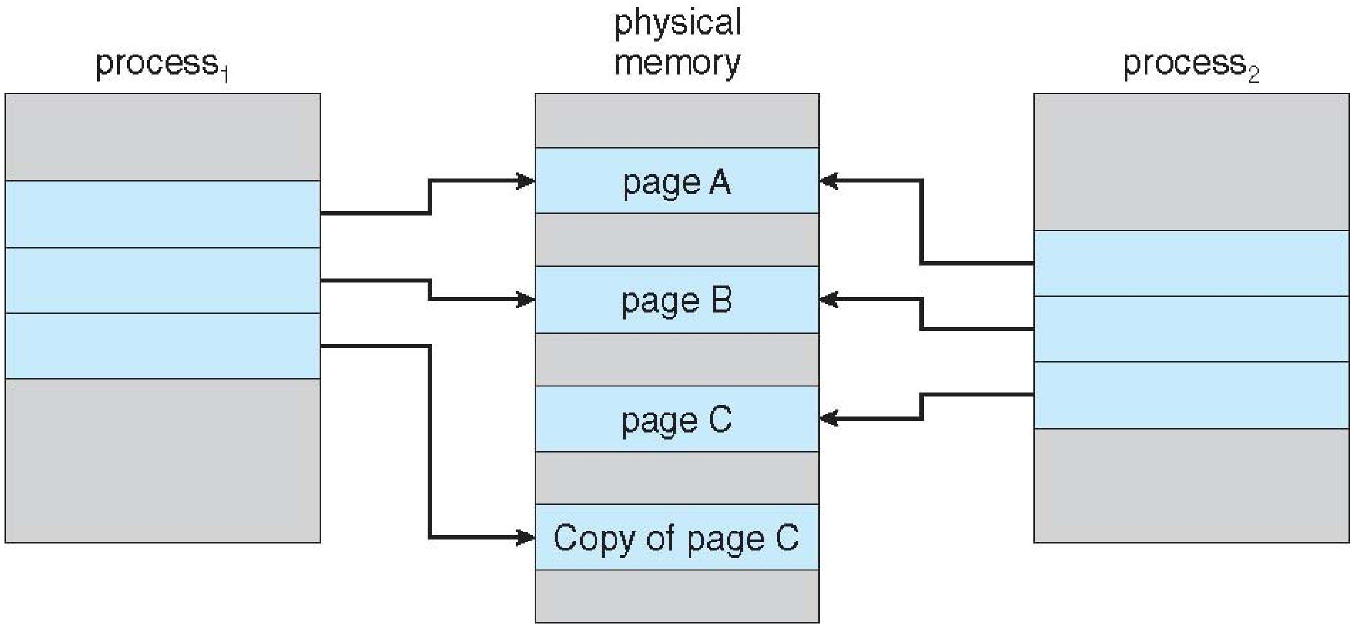
Copy-on-Write

Before Process 1 modifies page C



Copy-on-Write

After Process 1 modifies page C



Copy-on-Write : What happens if there is no Free Frame?

- 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

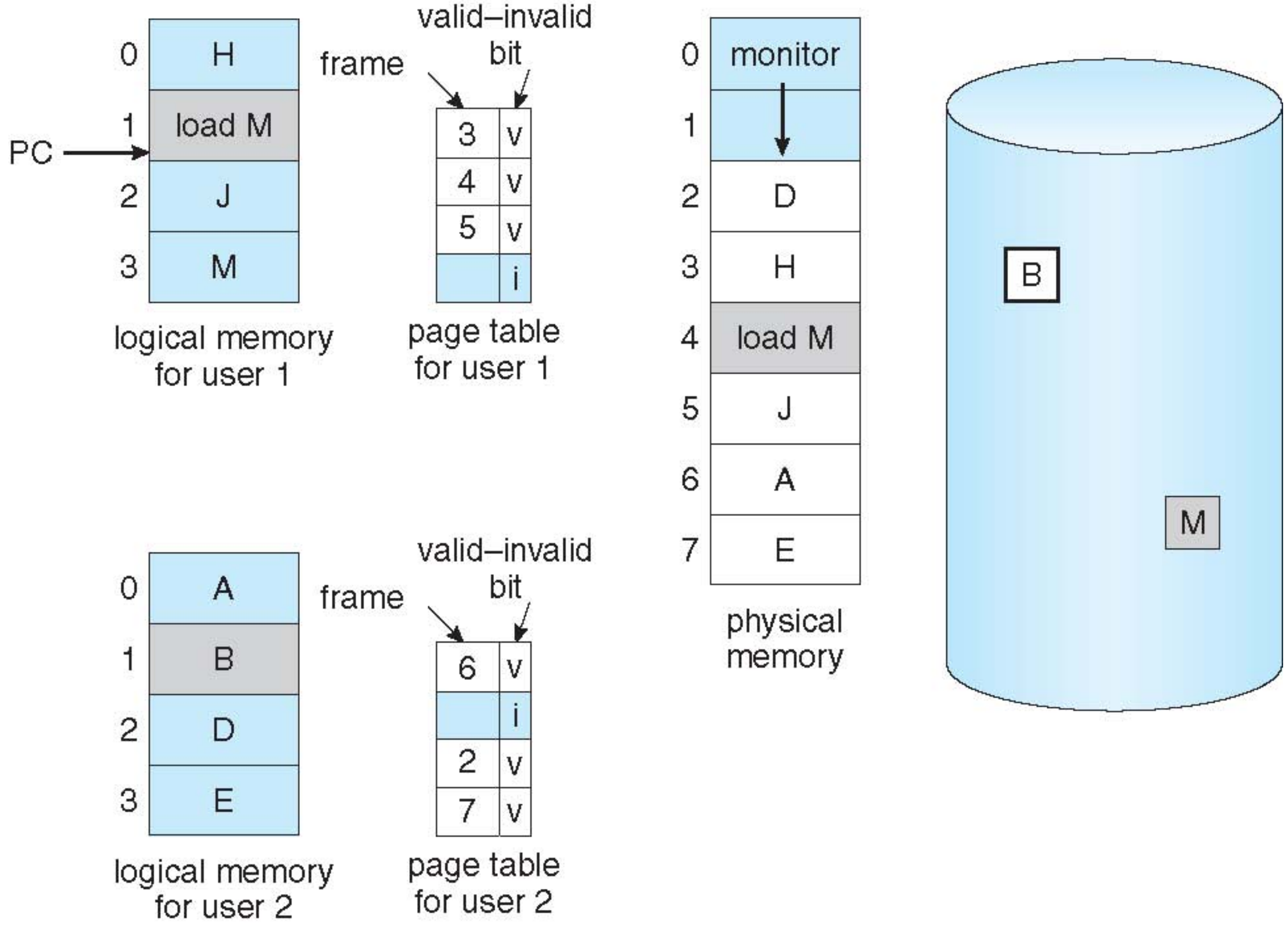
Overview

- Copy-on-Write
- Page Replacement

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

Page Replacement : Need for Page Replacement



Page Replacement : Basic Page Replacement

1. Find the location of the desired page on disk

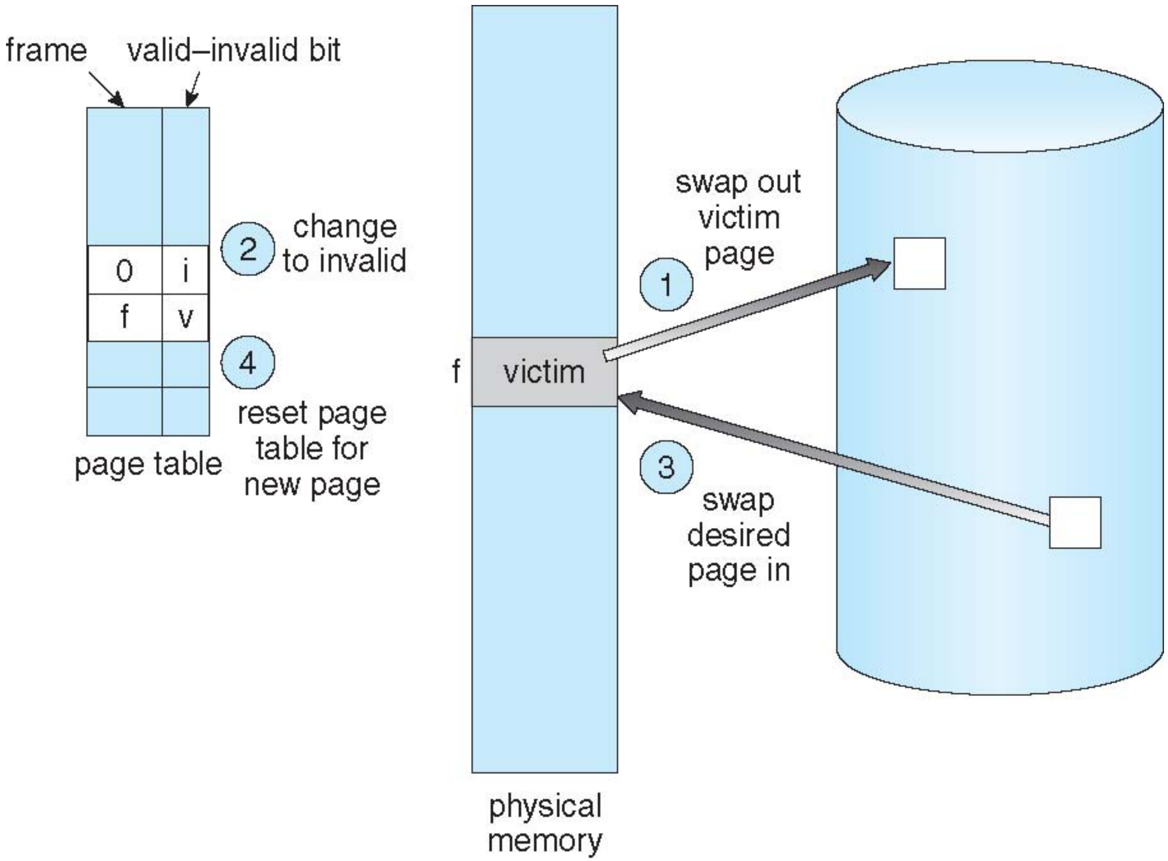
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame
     - Write victim frame to disk if dirty

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

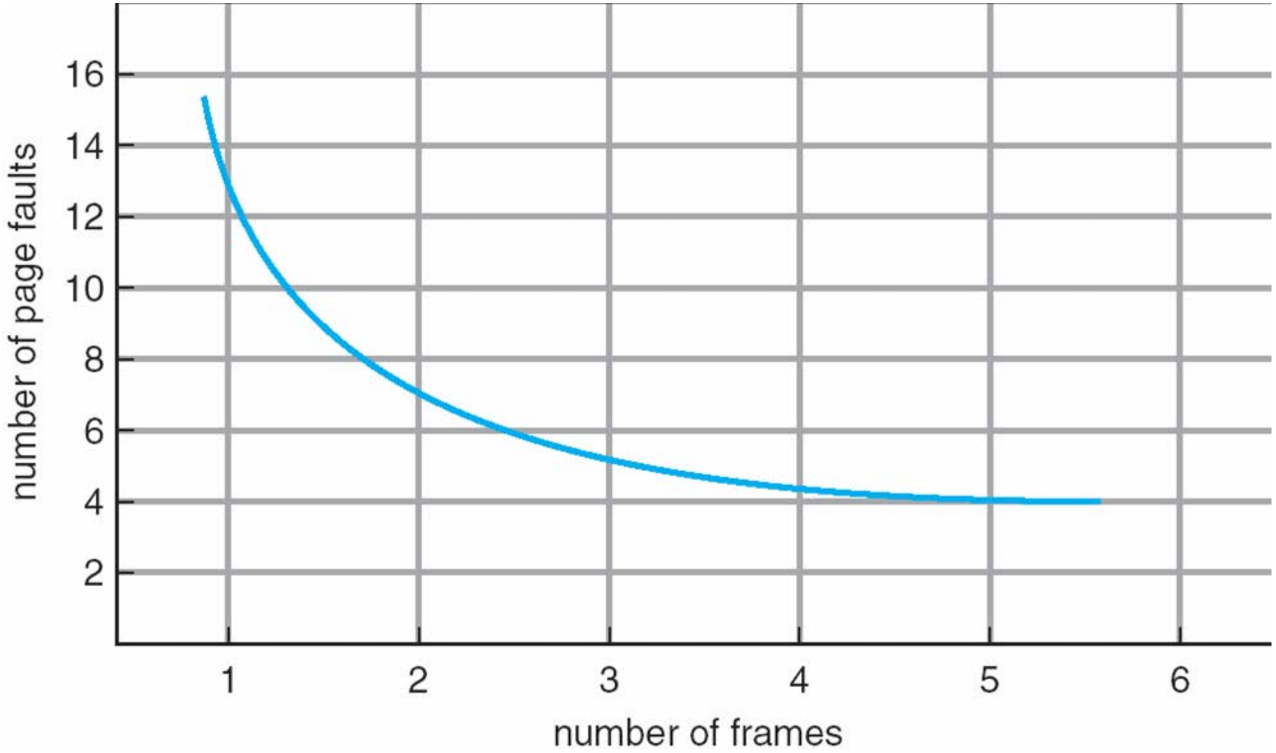


Page Replacement : Page and Frame Replacement Algorithms

- Frame-allocation algorithm determines
  - How many frames to give each process
  - Which frames to replace
- 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

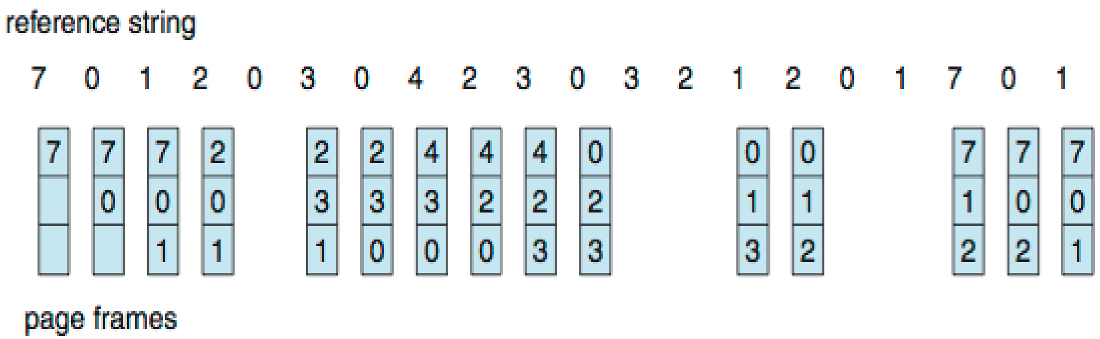
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

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



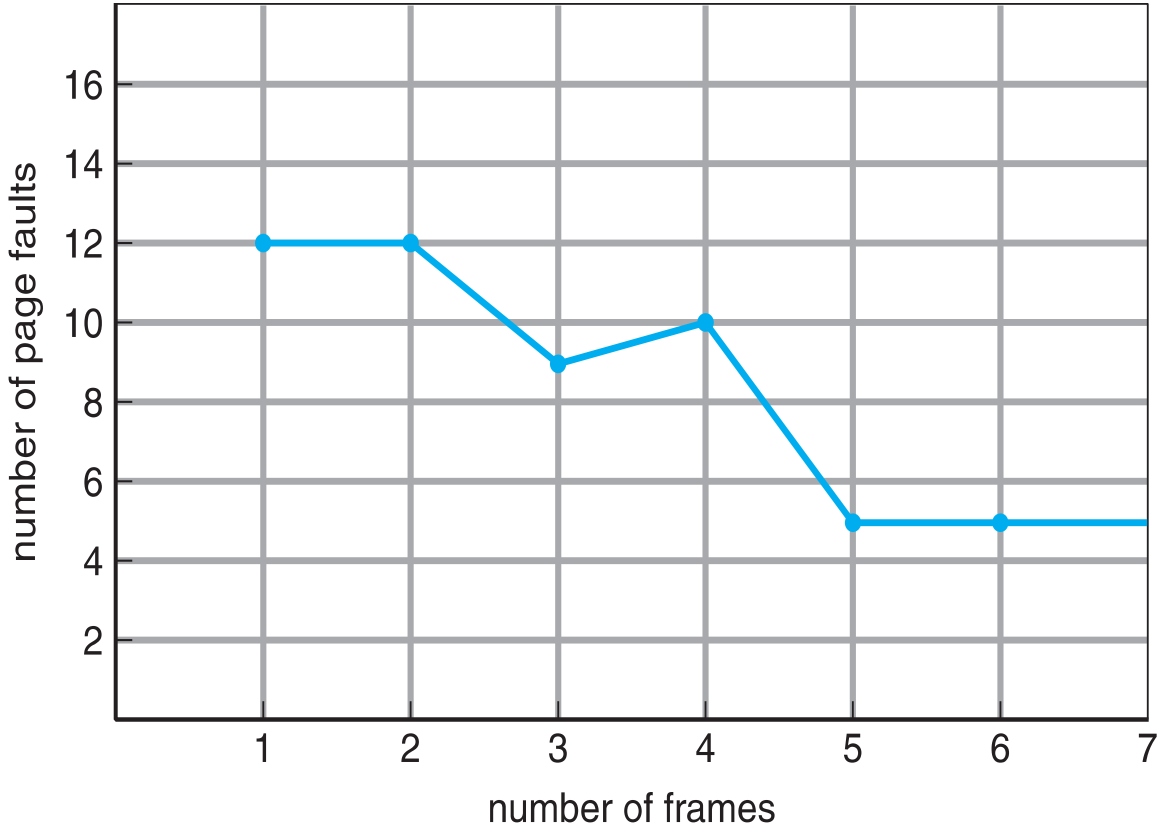
Page Replacement : First-In-First Out (FIFO) Algorithm

- 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



Credits for slides

Silberschatz, Galvin and Gagne