CSE 421/521 - Operating Systems
Fall 2014

LECTURE - XVI

VIRTUAL MEMORY - I

Tevfik Koşar

University at Buffalo

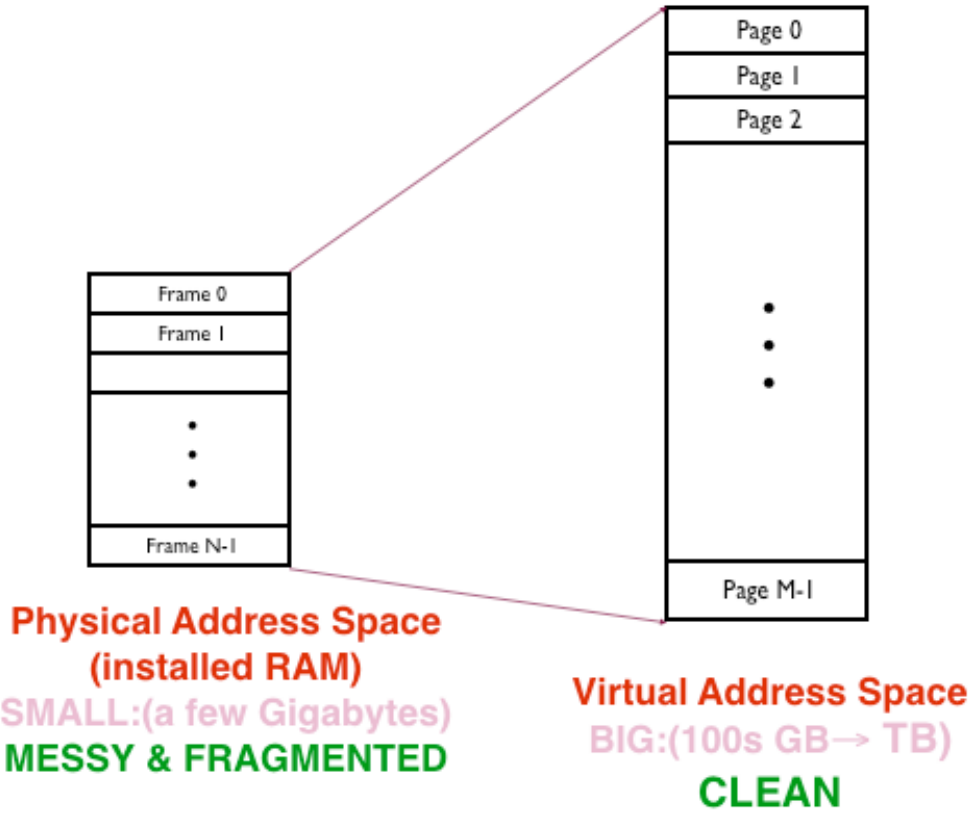
October 21st, 2014

Roadmap

- Virtual Memory
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms
    - FIFO
    - Optimal Algorithm

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.

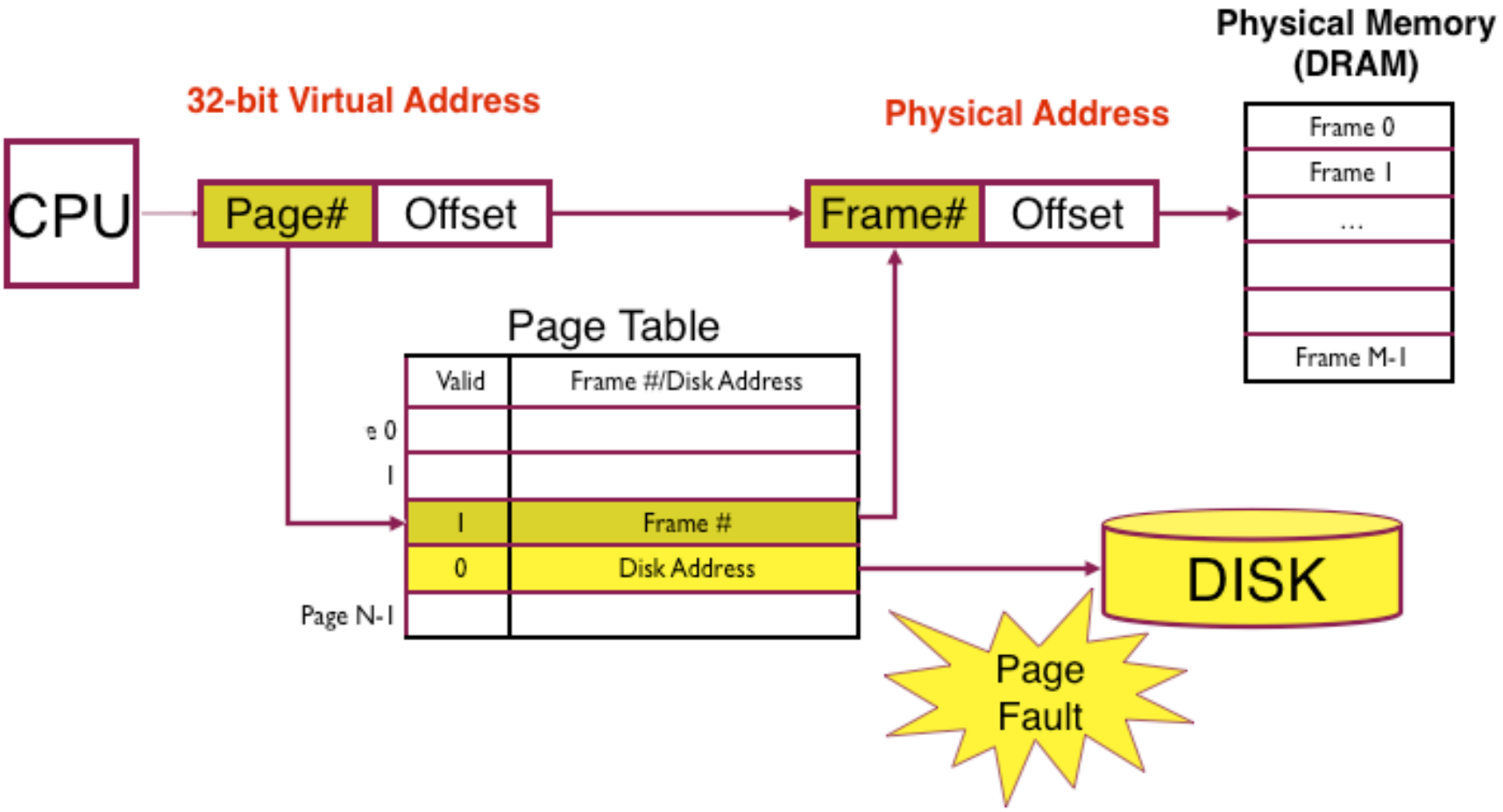


Goals

- Make programmers job easier
  - Can write code without knowing how much DRAM is there
  - Only need to know general memory architecture
    - (e.g., 32-bit address space)

- Enable Multiprogramming
  - Keep several programs running concurrently
    - Together, these programs may need more DRAM than we have.
    - Keep just the actively used pages in DRAM.
  - Share when possible
    - When one program does I/O switch CPU to another.

How it works?


Physical Memory
(DRAM)
32-bit Virtual Address
Physical Address
CPU
Page#
Offset
Frame#
Offset
Frame 0
Frame 1
...
Frame M-1
Page Table
Valid
Frame #/Disk Address
e 0
1
1
Frame #
0
Disk Address
Page N-1
DISK
Page
Fault

Implementation

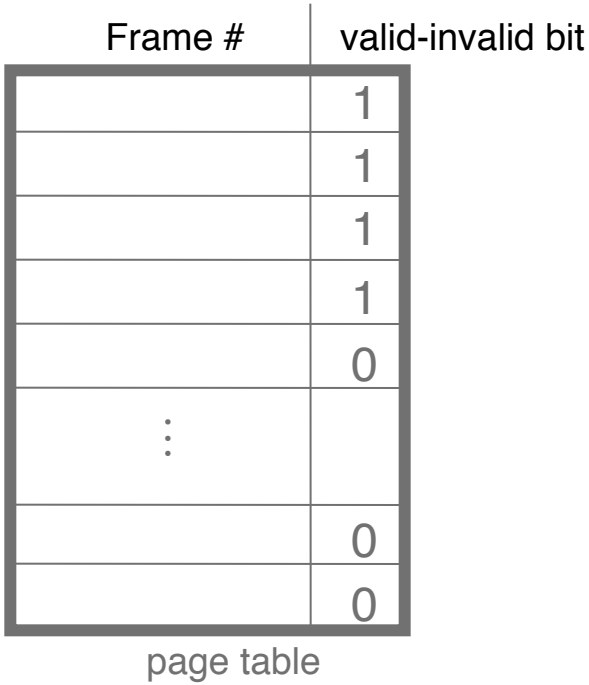
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

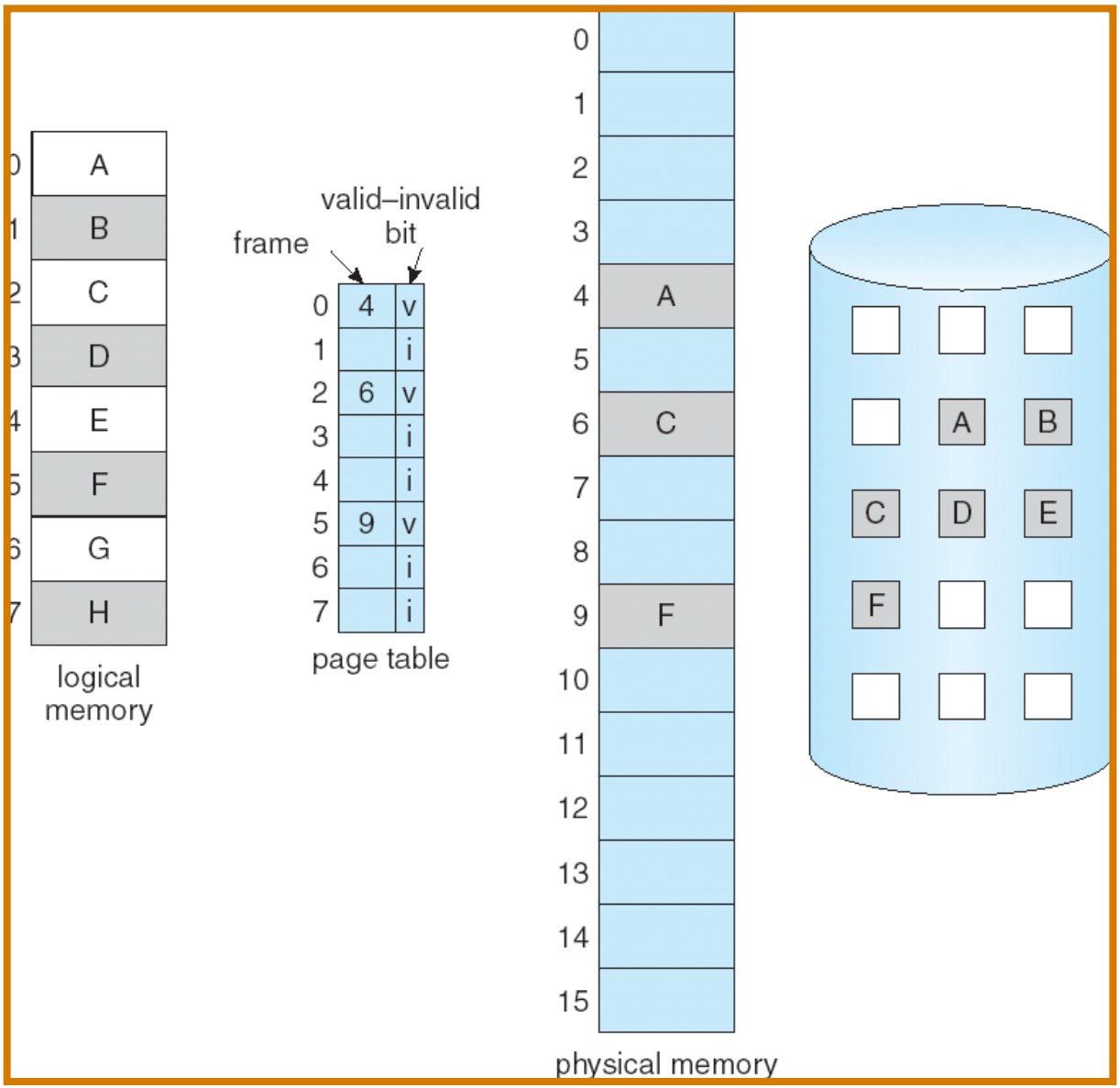
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory and legal, 0 ⇒ not-in-memory or invalid)
- Initially valid–invalid bit is set to 0 on all entries
- Example of a page table snapshot:

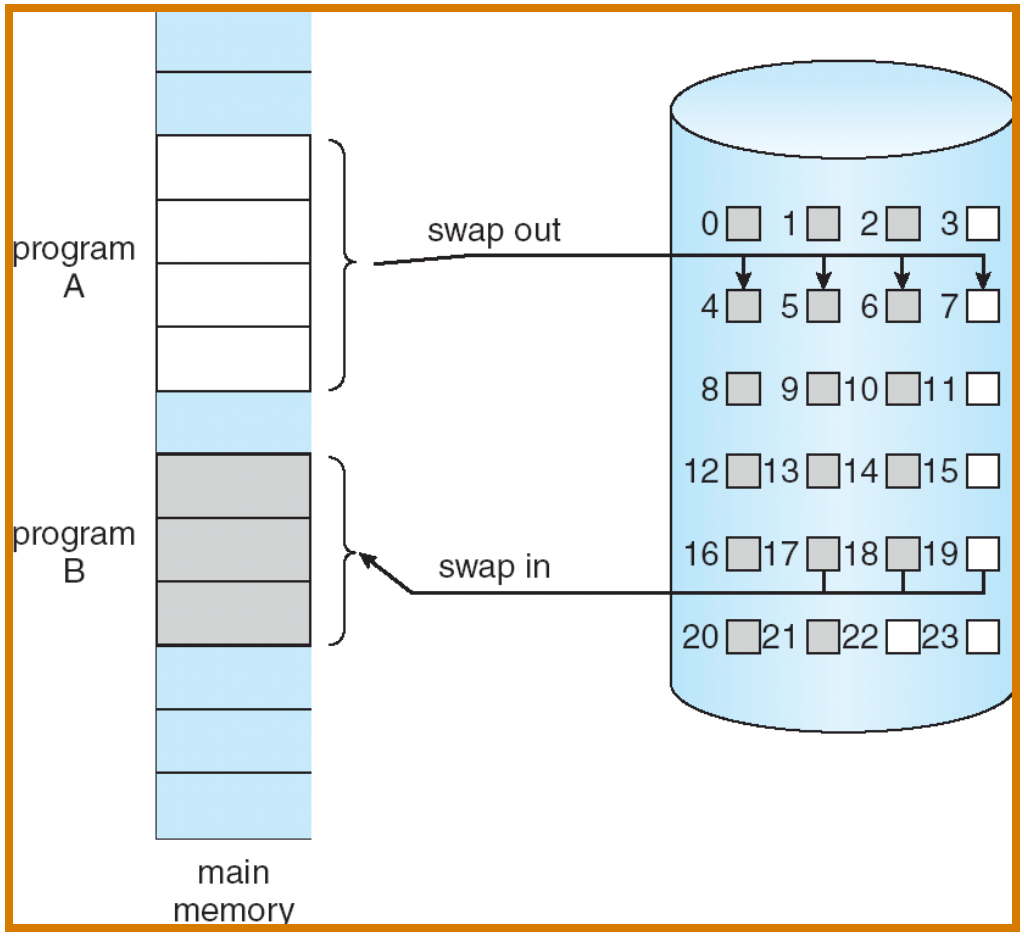


- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



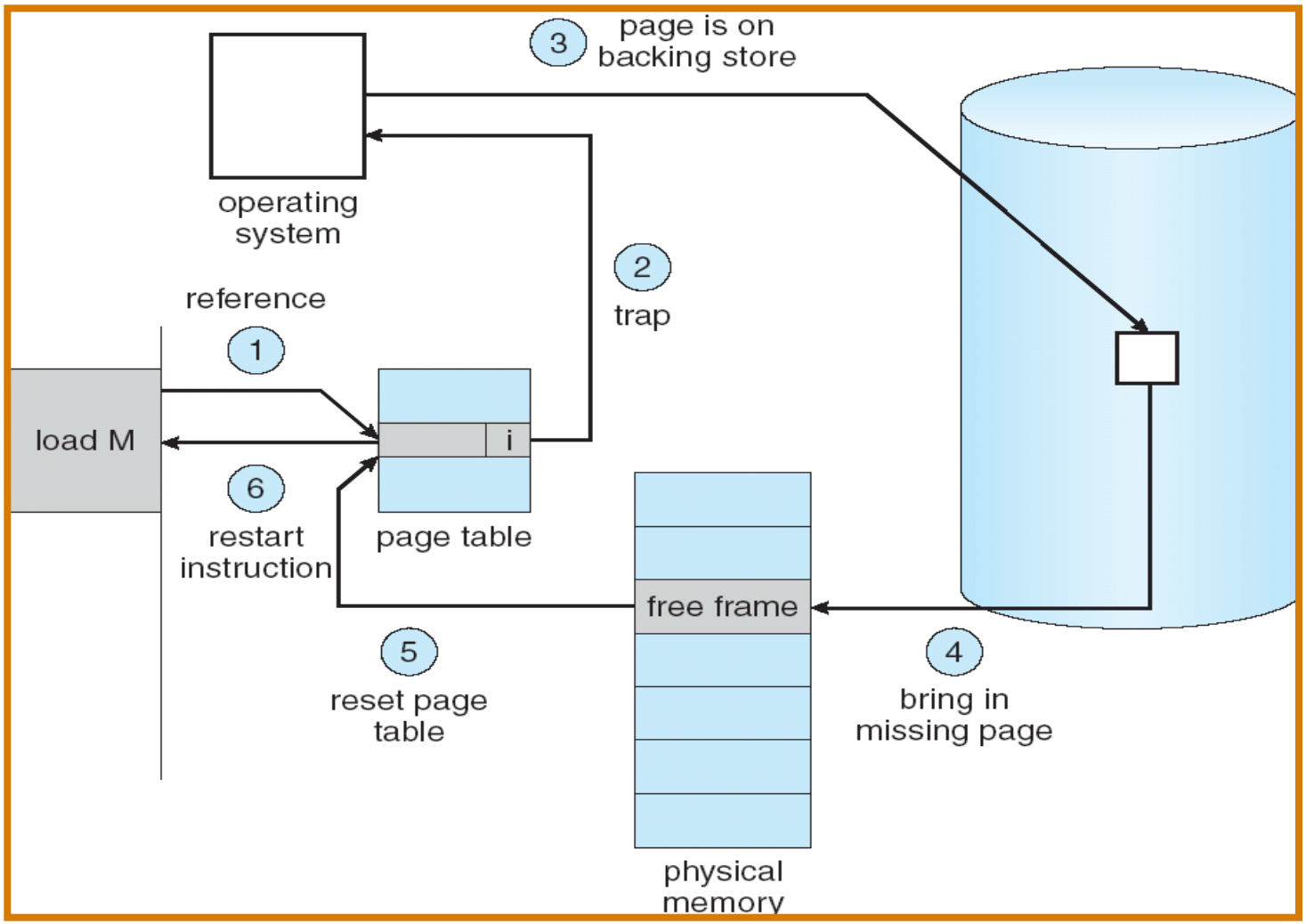
Transfer of a Paged Memory to Contiguous Disk Space

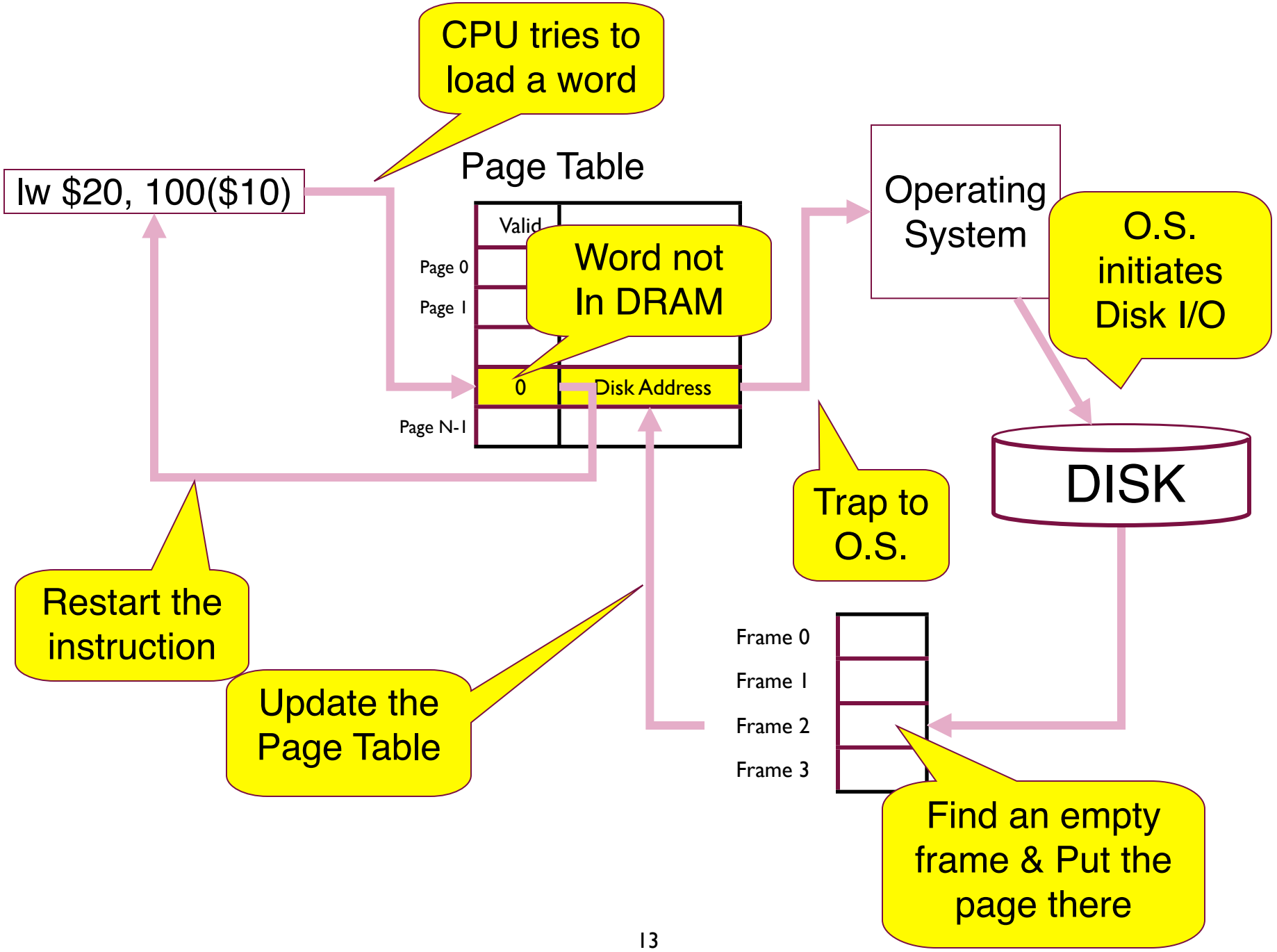


Page Fault

- If there is ever a reference to a page not in memory, first reference will trap to OS ⇒ page fault
- OS looks at another table (in PCB) to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory. ==> page-in
- Get an empty frame.
- Swap (read) page into the new frame.
- Set validation bit = 1.
- Restart instruction

Steps in Handling a Page Fault




CPU tries to load a word
lw $20, 100($10)
Page Table
Valid
Page 0
Page 1
Word not In DRAM
0
Disk Address
Page N-1
Operating System
O.S. initiates Disk I/O
DISK
Trap to O.S.
Restart the instruction
Update the Page Table
Frame 0
Frame 1
Frame 2
Frame 3
Find an empty frame & Put the page there

What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out
  - Algorithms (FIFO, LRU ..)
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

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

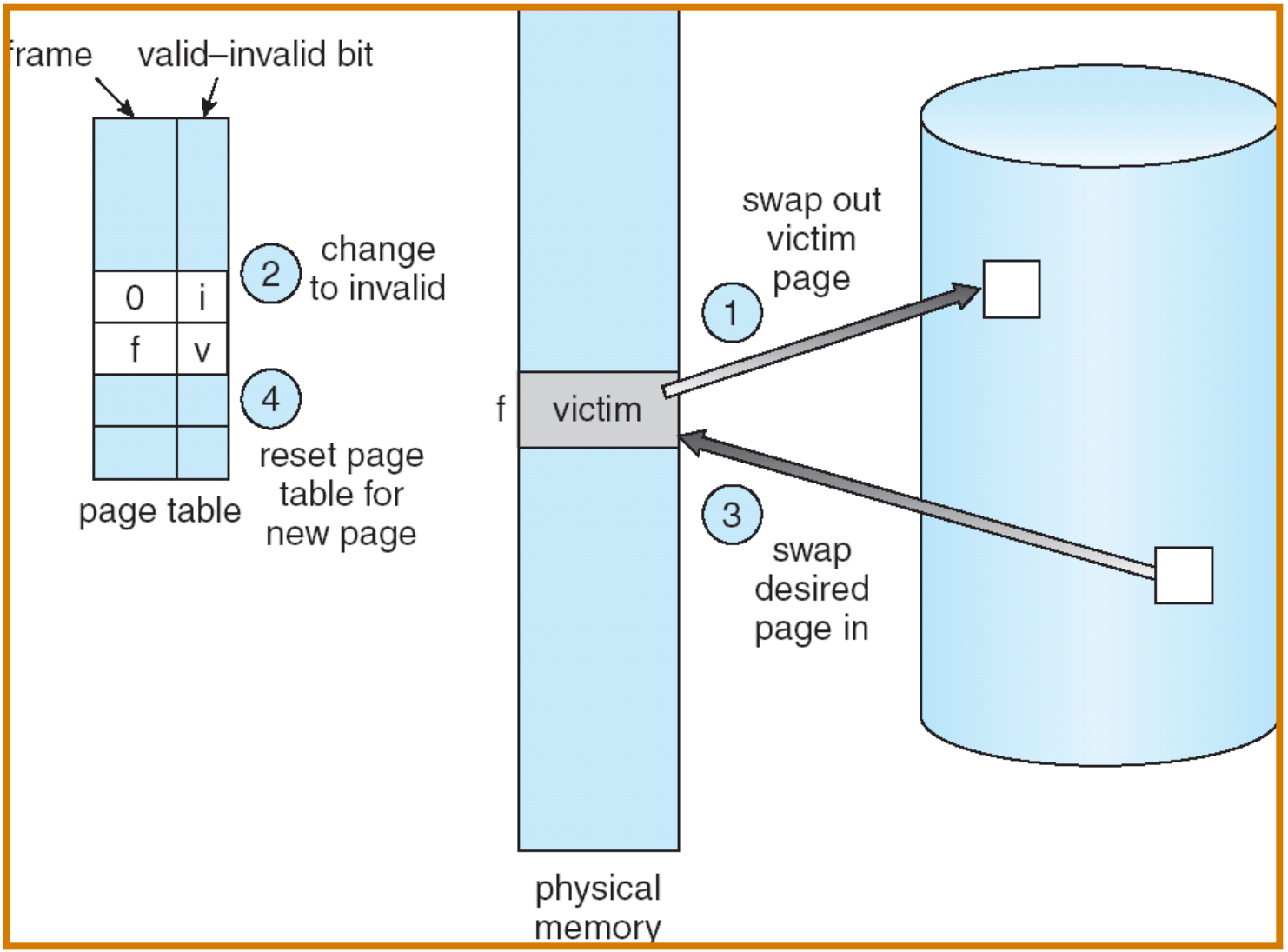
1. Find the location of the desired page on disk

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

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

4. Restart the process

Page Replacement

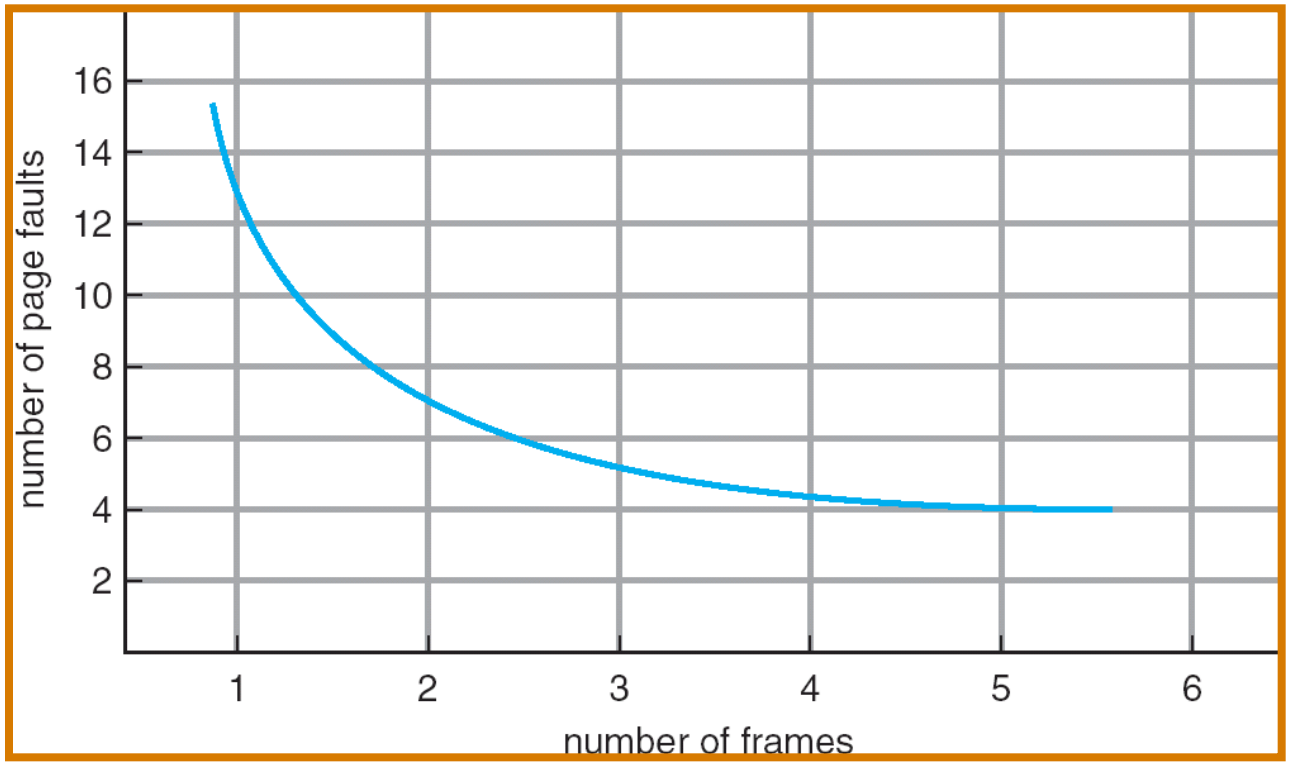


Page Replacement Algorithms

- 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
- In all our examples, the reference string is

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

Expected Graph of Page Faults vs The Number of Frames

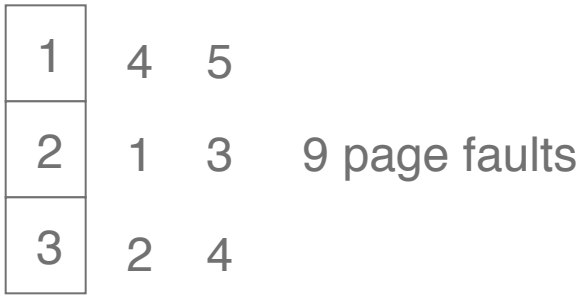


First-In-First-Out (FIFO) Algorithm

- 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)

First-In-First-Out (FIFO) Algorithm

- 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)



First-In-First-Out (FIFO) Algorithm

- 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)

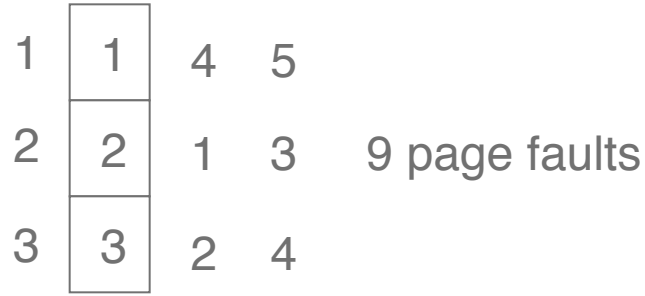
1	4	5	
2	1	3	9 page faults
3	2	4	

- 4 frames

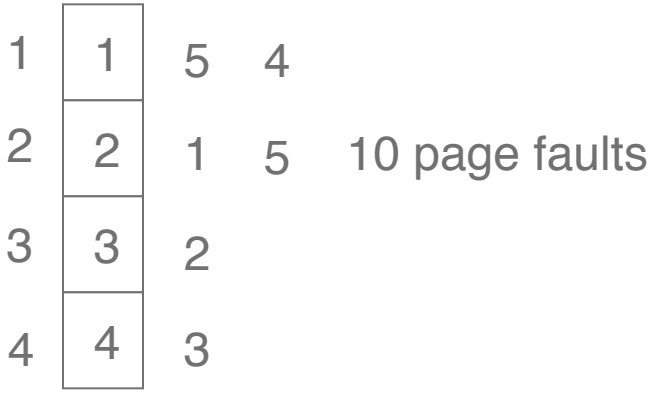
-

First-In-First-Out (FIFO) Algorithm

- 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)

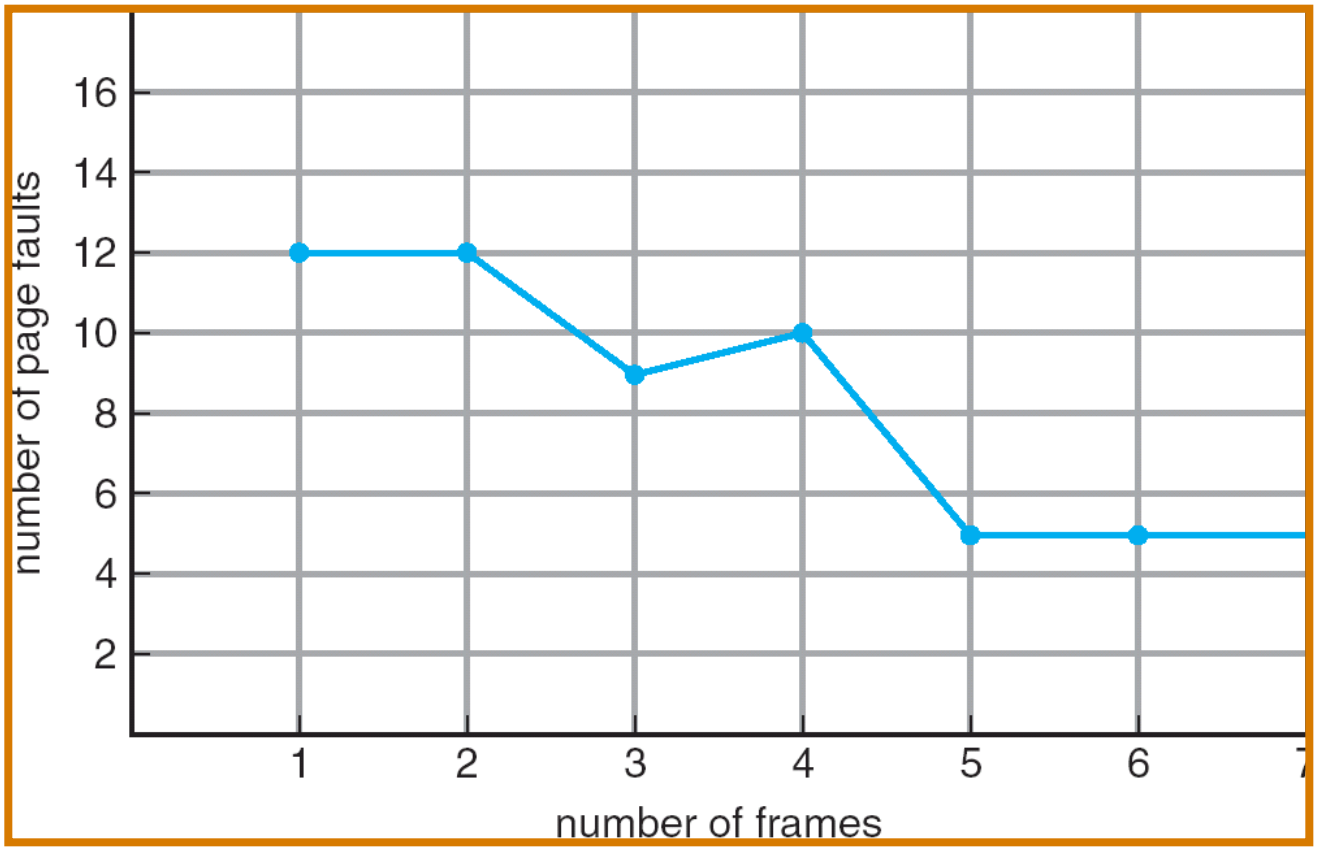


- 4 frames



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

FIFO Illustrating Belady's Anomaly



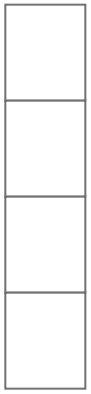
FIFO

- FIFO is obvious, and simple to implement
  - when you page in something, put it on the tail of a list
  - evict page at the head of the list
- Why might this be good?
  - maybe the one brought in longest ago is not being used
- Why might this be bad?
  - then again, maybe it *is* being used
  - have absolutely no information either way
- In fact, FIFO's performance is typically lousy
- In addition, FIFO suffers from Belady's Anomaly
  - there are reference strings for which the fault rate *increases* when the process is given more physical memory

Optimal Algorithm

- Replace page that will not be used for the longest time in future
- 4 frames example

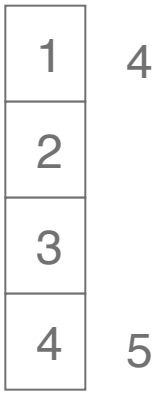
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

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



6 page faults

- How would you know this in advance?

Optimal (Belady's) Algorithm

- Provably optimal: lowest fault rate (remember SJF?)
  - evict the page that won't be used for the longest time in future
  - problem: impossible to predict the future
- Why is Belady's Optimal algorithm useful?
  - as a yardstick to compare other algorithms to optimal
    - if Belady's isn't much better than yours, yours is pretty good
      - how could you do this comparison?
- Is there a best practical algorithm?
  - no; depends on workload
- Is there a worst algorithm?
  - no, but random replacement does pretty badly
    - there are some other situations where OS's use near-random algorithms quite effectively!

Summary

- Virtual Memory
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms
    - FIFO
    - Optimal Algorithm

- Next Lecture: Virtual Memory - II
- Reading Assignment: Chapter 9 from Silberschatz.

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR