CSc 360
Operating Systems
Virtual Memory

Wenjun Yang

Fall 2025

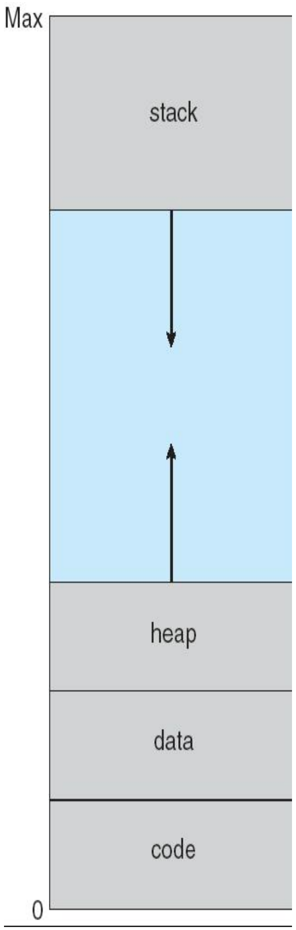
Review

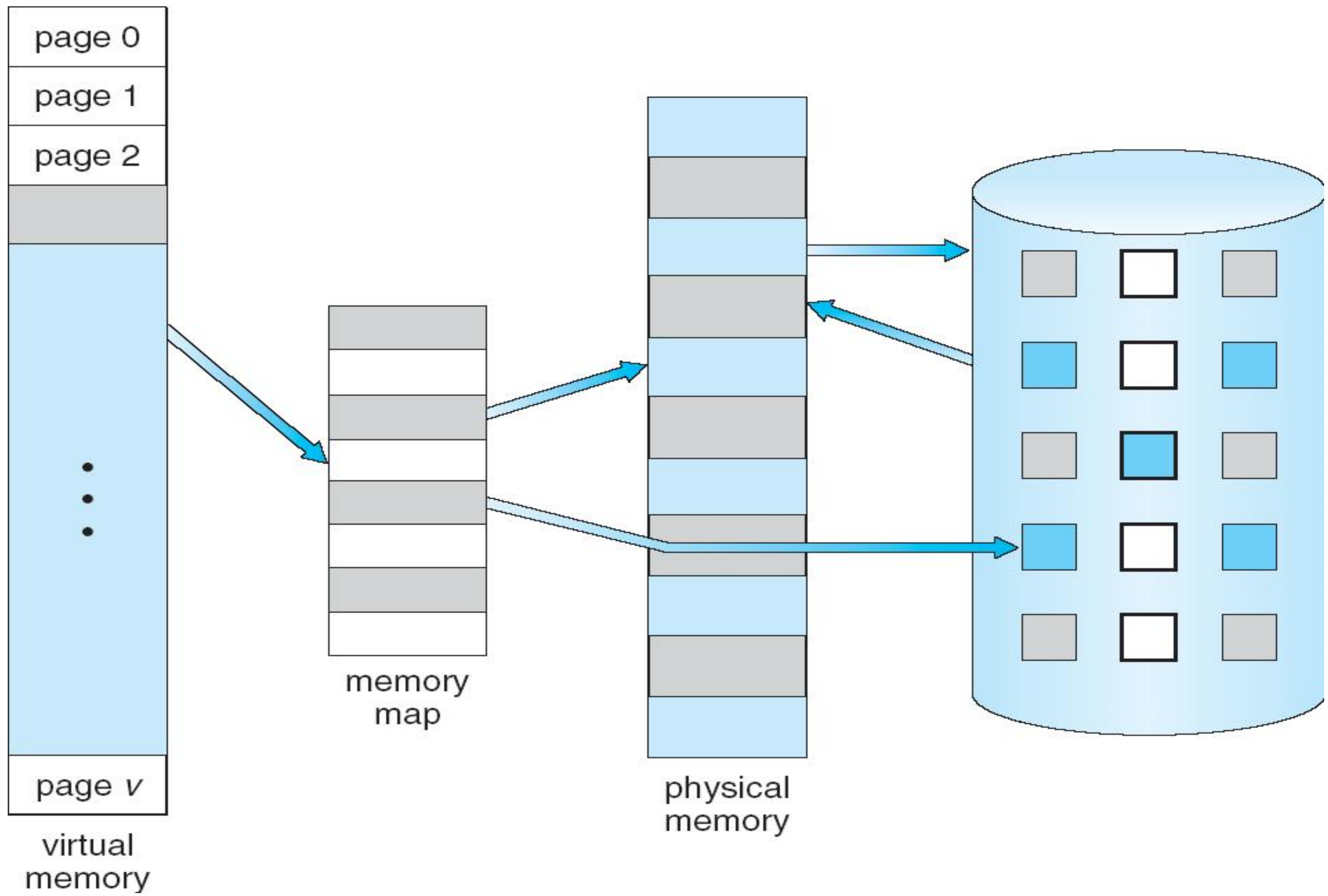
- Main memory management
  - addressing and address mapping
    - logical vs physical address
  - partition allocation
    - internal vs external fragmentation
  - paging and page table
    - TLB, hierarchical, hashed, inverted
  - segmentation

Virtual memory

- Load an entire process in physical memory?
  - not all addresses accessed at the same time
  - some even not used before swapped out!
- Virtual memory
  - logical space >> physical space
    - only load the portion being used or to be used
  - more (partial) processes in physical memory
  - faster swap in/out

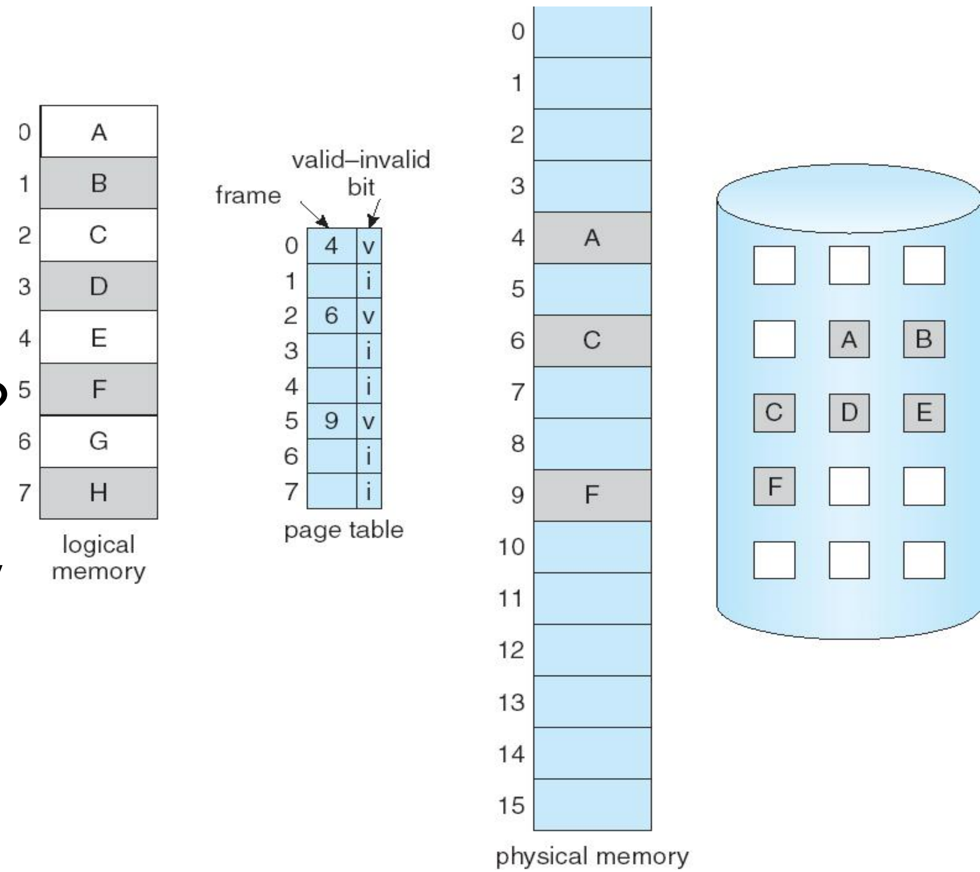
Virtual vs physical memory





Demand paging

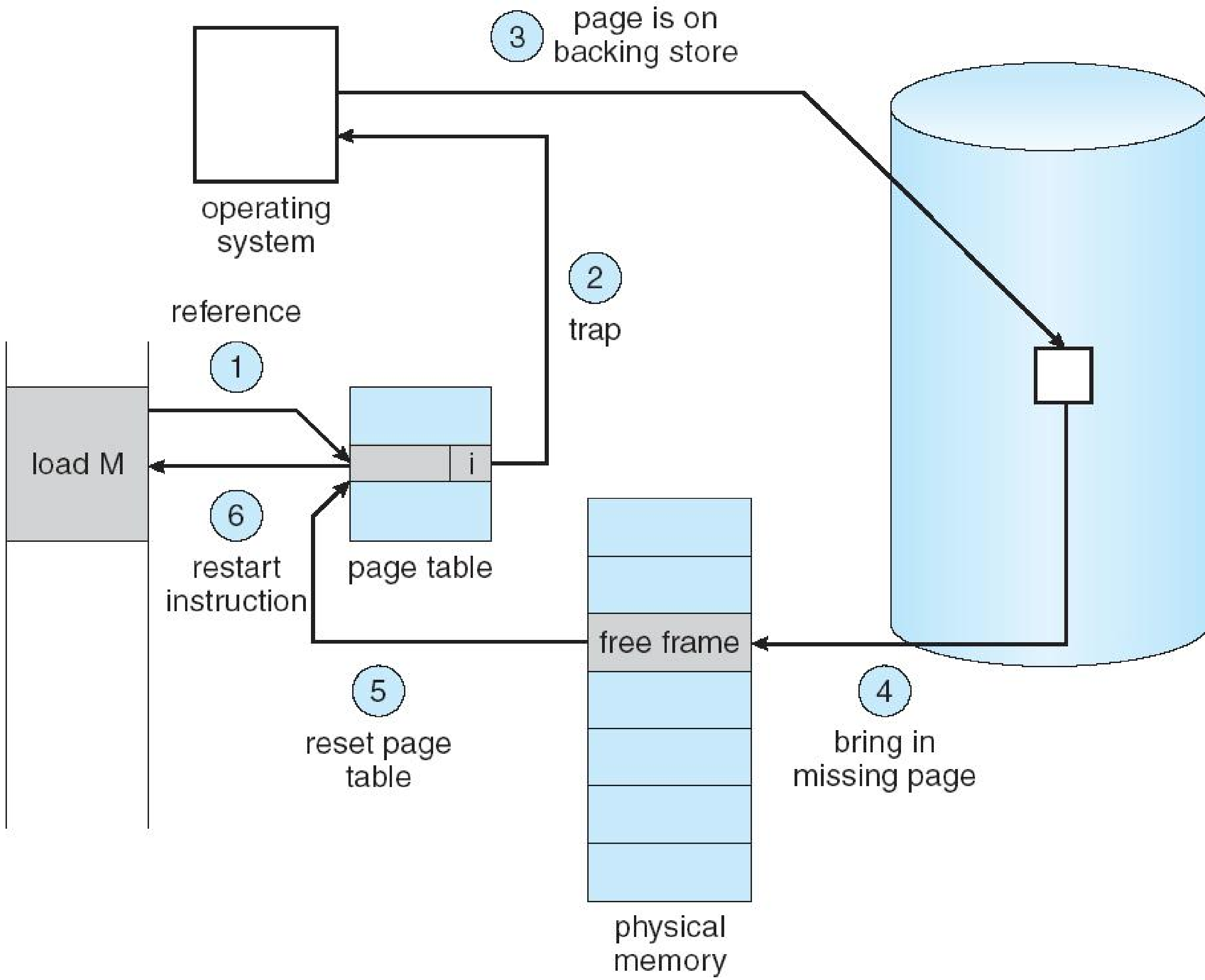
- On demand
  - when used
    - being referenced
    - really invalid reference?
      - error!
    - *valid* but not-in-memory
      - page in
  - lazy pager
  - valid-invalid bit



Q: Demand segmentation?

Page fault

- Reference
- Trap
  - to OS
- Page in
- Update
- Restart
  - issues



Q: block move instruction

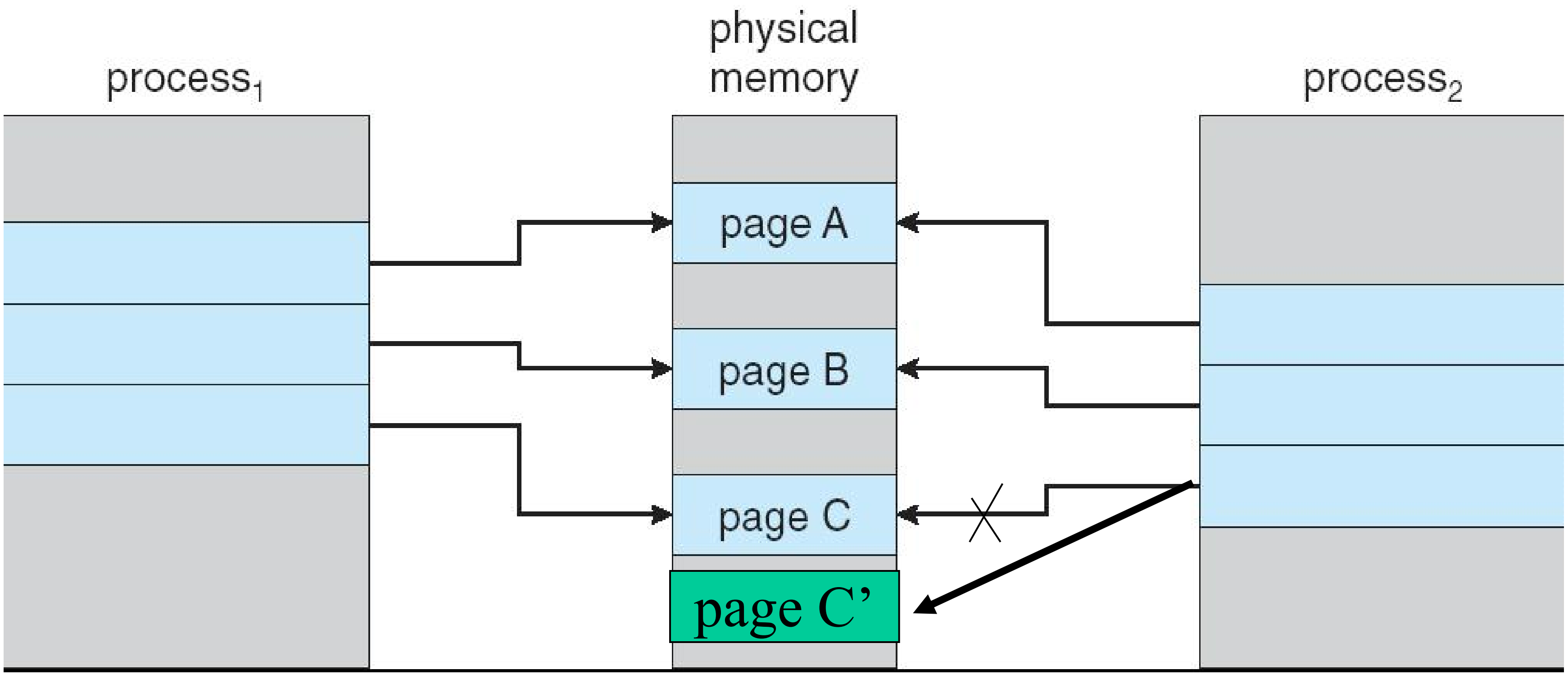
Paging performance

- Page Fault Rate 0 <= *p* <= 1.0
  - if *p* = 0, no page faults
  - if *p* = 1, every reference is a fault
- Effective Access Time (EAT)

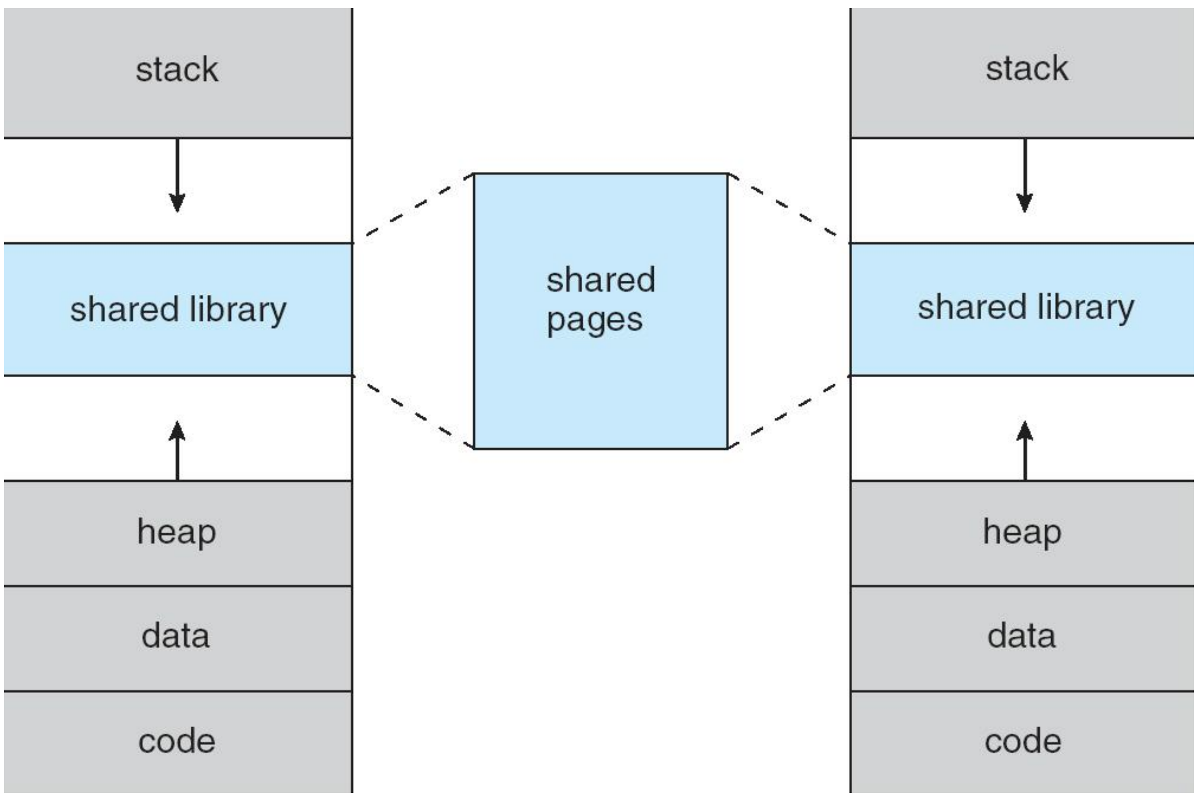
EAT = (1 – *p*) x memory access
+ *p* (page fault overhead
+ swap page out
+ swap page in
+ restart overhead)

Copy-on-write

- Share read-only pages



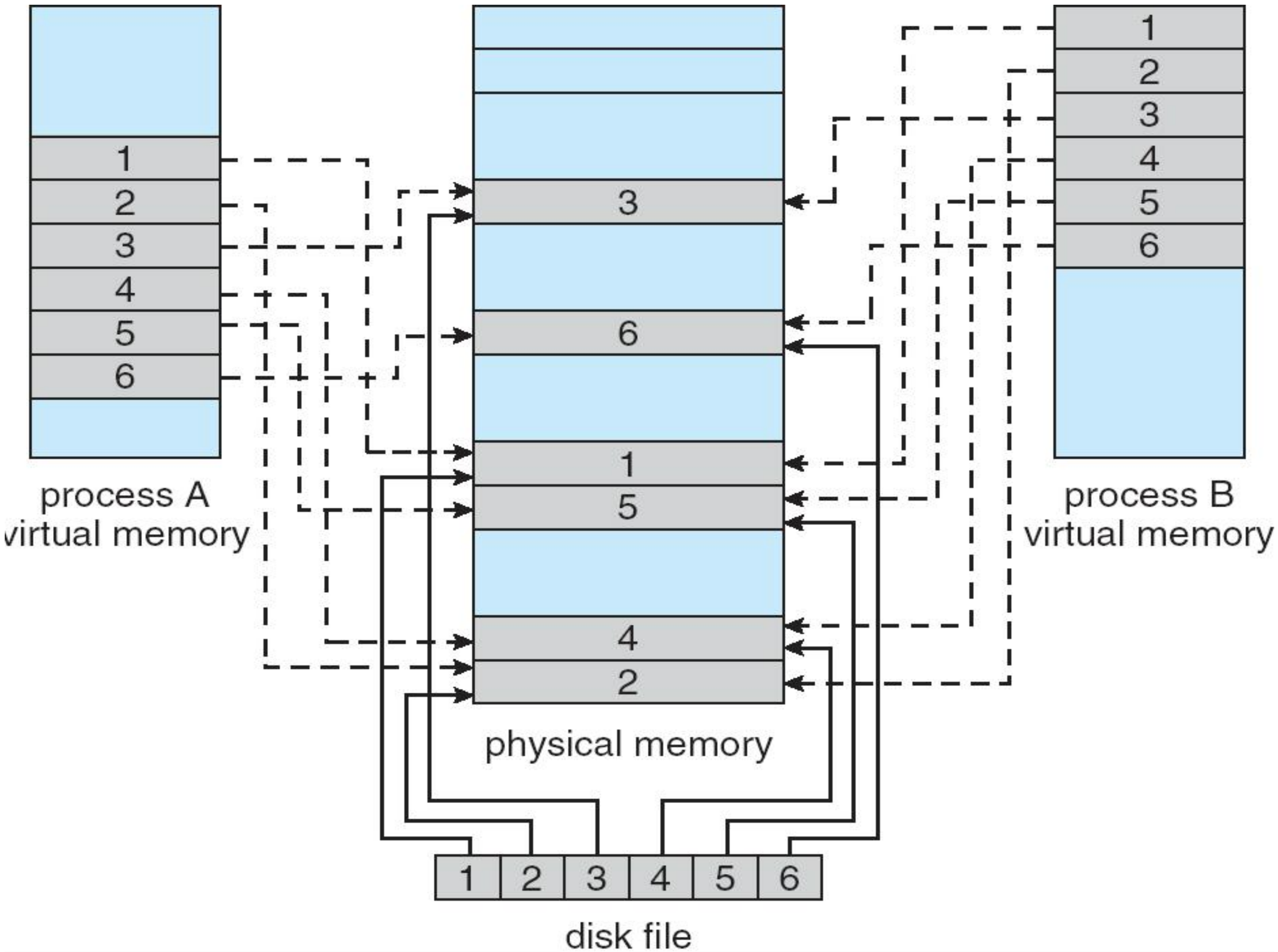
Memory sharing



Memory-mapped files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than **read() write()** system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared

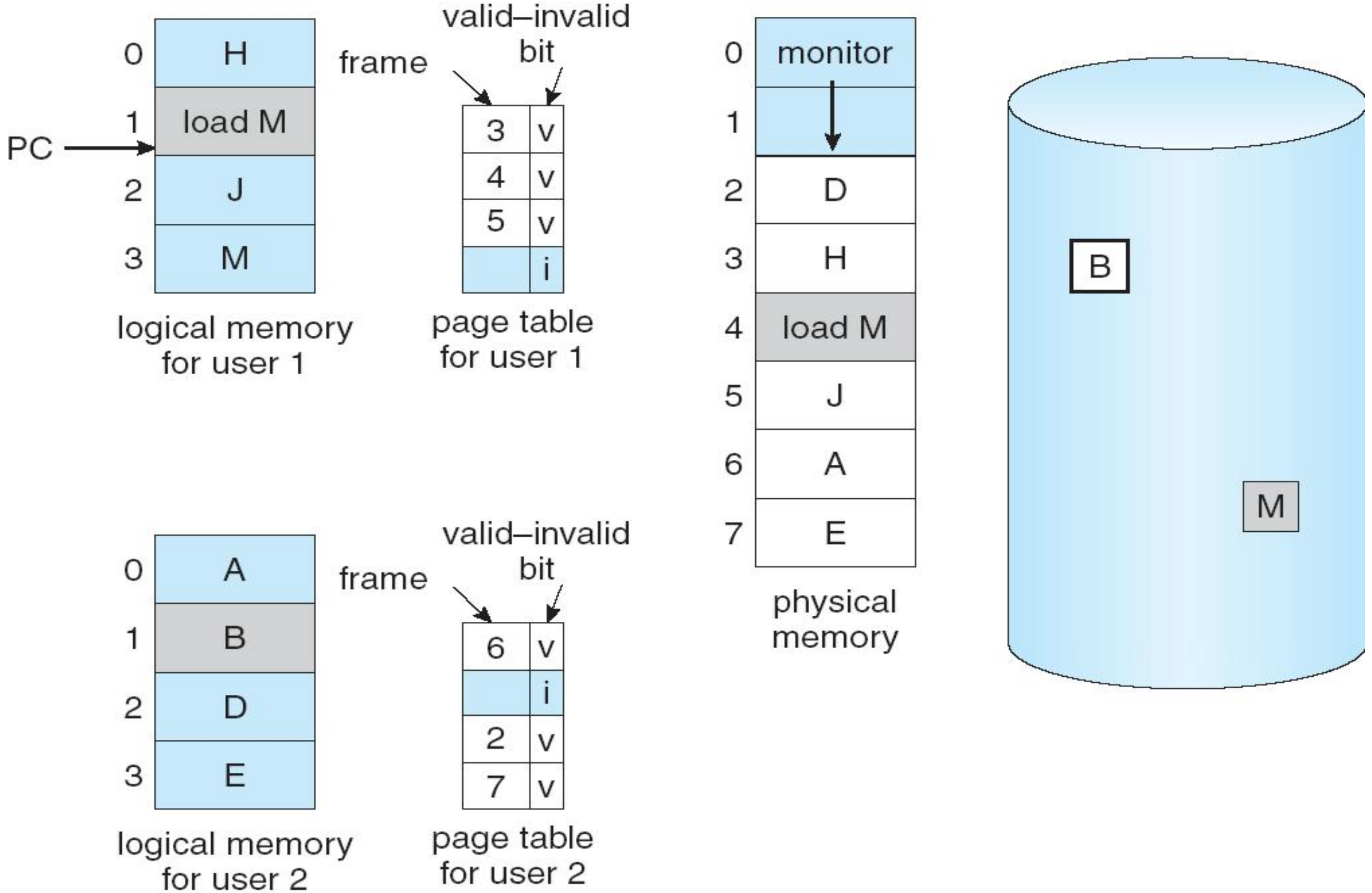
Memory-mapped file sharing



This lecture so far

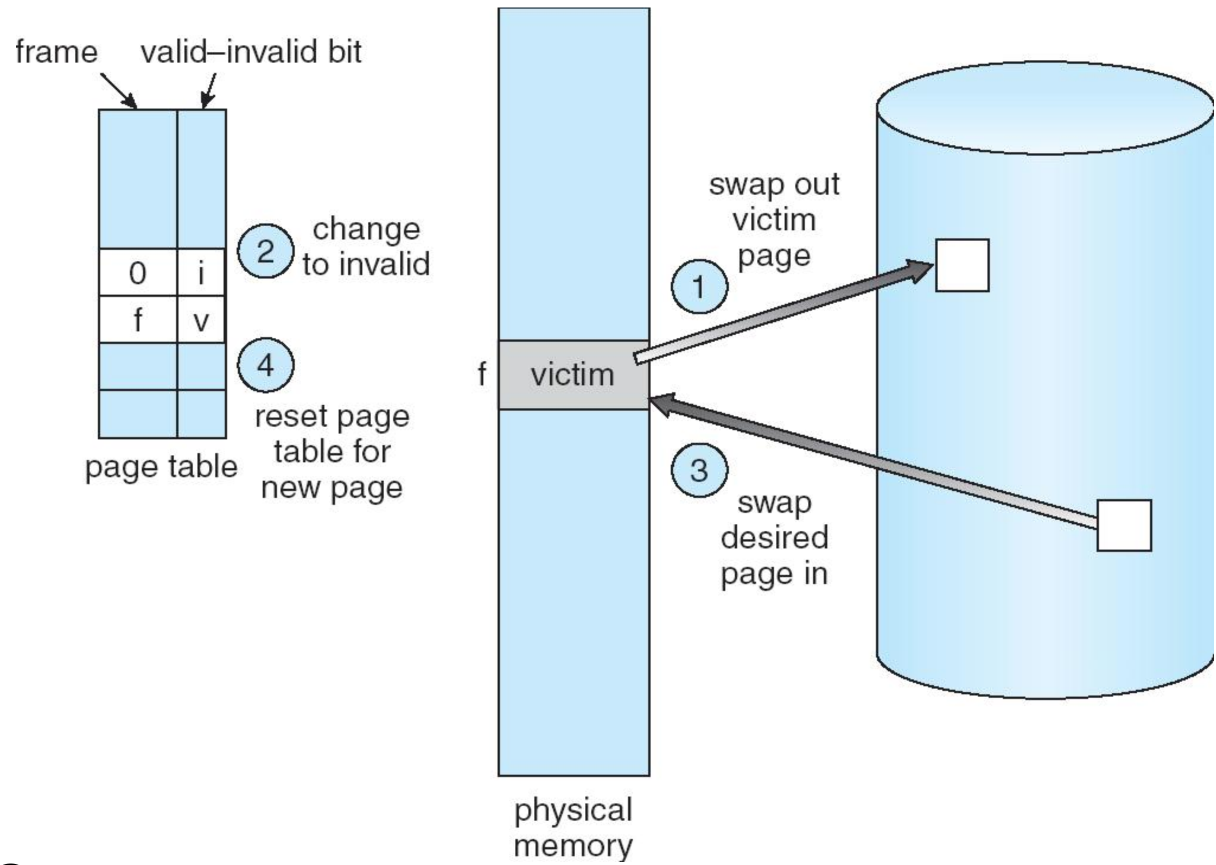
- Virtual memory
  – why virtual memory
  – how to support virtual memory
    - on-demand paging
  – other features
    - copy-on-write, memory-mapped files
- What if no free pages?

The need for page replacement



Basic page replacement

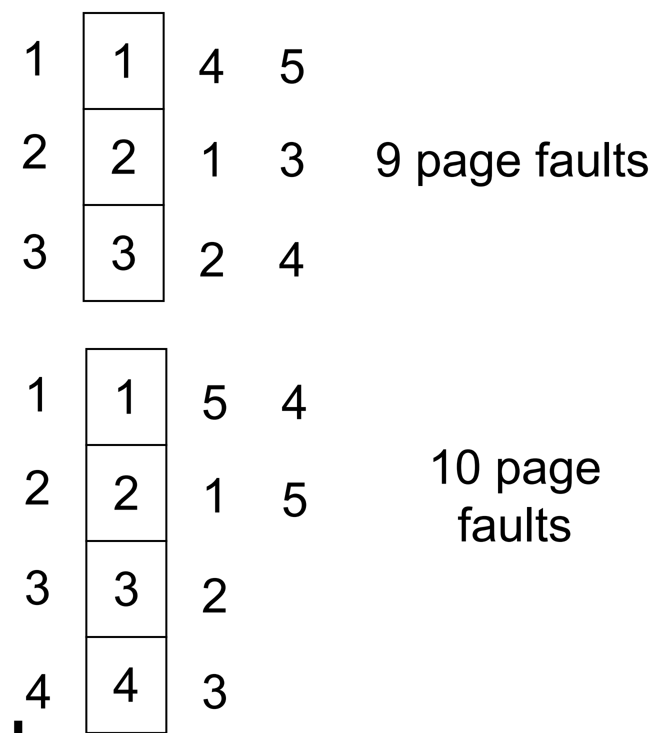
- Page in
  - no free frame?
    - choose a victim
    - swap out
      - dirty bit
- Algorithms
  - allocation
  - replacement
- Reduce fault rate



Q: modify bit

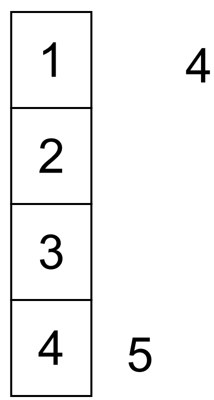
First-in-first-out (FIFO)

- Reference string
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames
- 4 frames
- Belady's Anomaly
  - more frames: more page faults!



Optimal algorithm

- Replace the page will not be used for the longest period of time
  - how do you know the future?!
    - if history is of any indication …



6 page faults

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

Q: how to achieve optimality?

Least recently used (LRU)

- Replace the least recently used
- Counter implementation
  - copy timestamp
  - search to replace
- Stack implementation
  - move to top
  - replace the bottom

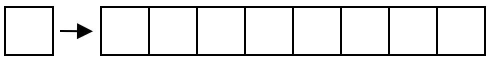
1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

1	2	3	4	1	2	**5**	1	2	**3**	**4**	**5**
	1	2	3	4	1	2	5	1	2	3	4
		1	2	3	4	1	2	5	1	2	3
			1	2	3	4	4	4	5	1	2

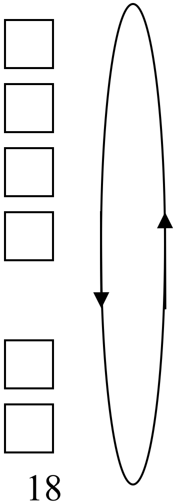
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

LRU approximation algorithms

- Additional-reference-bits algorithm
  - set reference bit on reference



  - shift into a history byte on timer
  - least-recently-used one has lowest history byte
- ***Second-chance algorithm***
  - if 0, replace
  - if 1, change to 0 and check the next page
  - circular queue



Counting-based algorithms

- Counter
  - number of references
- Least frequently used (LFU)
  - replace the one with the smallest counter
  - aging necessary
- Most frequently used (MFU)
  - new page has small counter
    - and might be referenced soon

This lecture so far

- Page replacement algorithms
  - FIFO
  - optimal
  - LRU
    - approximation
- Explore further
  - another reference string
    7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Page buffering

- Keep a pool of free pages
  - speed up swapping in desired pages
  - no need to wait a page *becomes* free
- Keep a list of modified pages
  - synch with disk when paging is idle
  - reduce overhead when swapping out
- Reuse "clean" pages from the pool
  - on page fault, check free pool first

Page allocation

- A process needs *minimum* number of pages
  - e.g., IBM 370: 6 pages to handle SS MOVE
    - instruction is 6 bytes, might span 2 pages
    - 2 pages to handle **from**
    - 2 pages to handle **to**
- Two major allocation schemes
  - fixed allocation
  - priority allocation

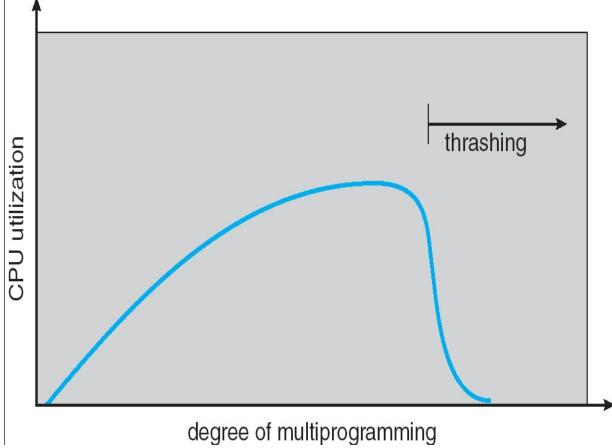
Fixed allocation

- Equal allocation
  - M free pages
  - N requesting processes
  - allocation: floor(M/N) each
    - some processes may request less than M/N
- Proportional allocation
  - each process requests s_i; all request $S=\Sigma s_i$
  - allocation: $s_i/S*M$

Priority allocation

- Allocation proportional to priority
  - higher priority process gets more pages allocated *when* necessary
- On page fault
  - select for replacement one of its frames.
  - select for replacement a frame from a process with lower priority number
- Global vs local replacement

Prevent thrashing

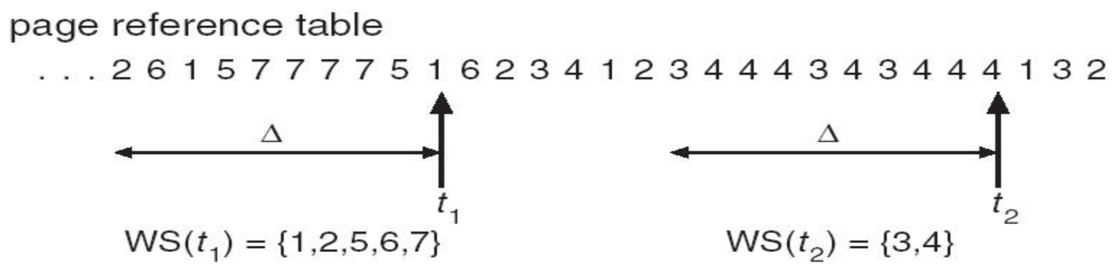


- Thrashing
  - more time on paging than executing
    - busy I/O, idle CPU
    - more processes admitted, more page faults
    - more processes in thrashing!
- Why thrashing
  - paging: explore locality
  - thrashing: locality explored too much!

Paging and thrashing

- Why does thrashing occur?
  - sum of size of locality > total memory size

Working-set model



- Prevent thrashing
  - local replacement
  - sufficient provisioning
  - working-set model
    - working-set window
      - most recent page references
    - working-set size (WSS)
      - number of unique page references
  - when sum $WSS_i > M$, reduce multiprogramming!

The last part of this module

- Page allocation
  - allocation algorithms
  - thrashing and thrashing prevention
- Explore further
  - OSC7 Section 9.8 and 9.9

Next lecture

- Mass storage
  - read OSC7Ch12