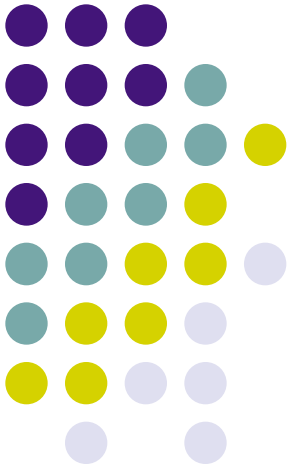
Page Replacement, Midterm Review 1

ECE469, March 8

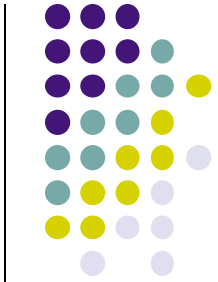
Yiying Zhang



Reading

- Chapter 9

Virtual Memory review

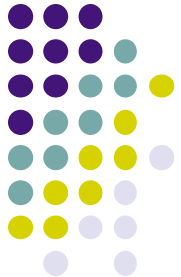


- What is *Virtual Memory*?
- Virtual address vs. virtual memory?

- How is Virtual Memory typically implemented?
  - What is *demand paging?*
  - It can also be implemented via *demand segmentation*
    - Double drawbacks?

- In Virtual Memory, what will the virtual address space be translated to?
- What is a page fault?
- Page fault vs. TLB miss?
- Does a TLB hit guarantee no page fault?

[lec16] Virtual Memory

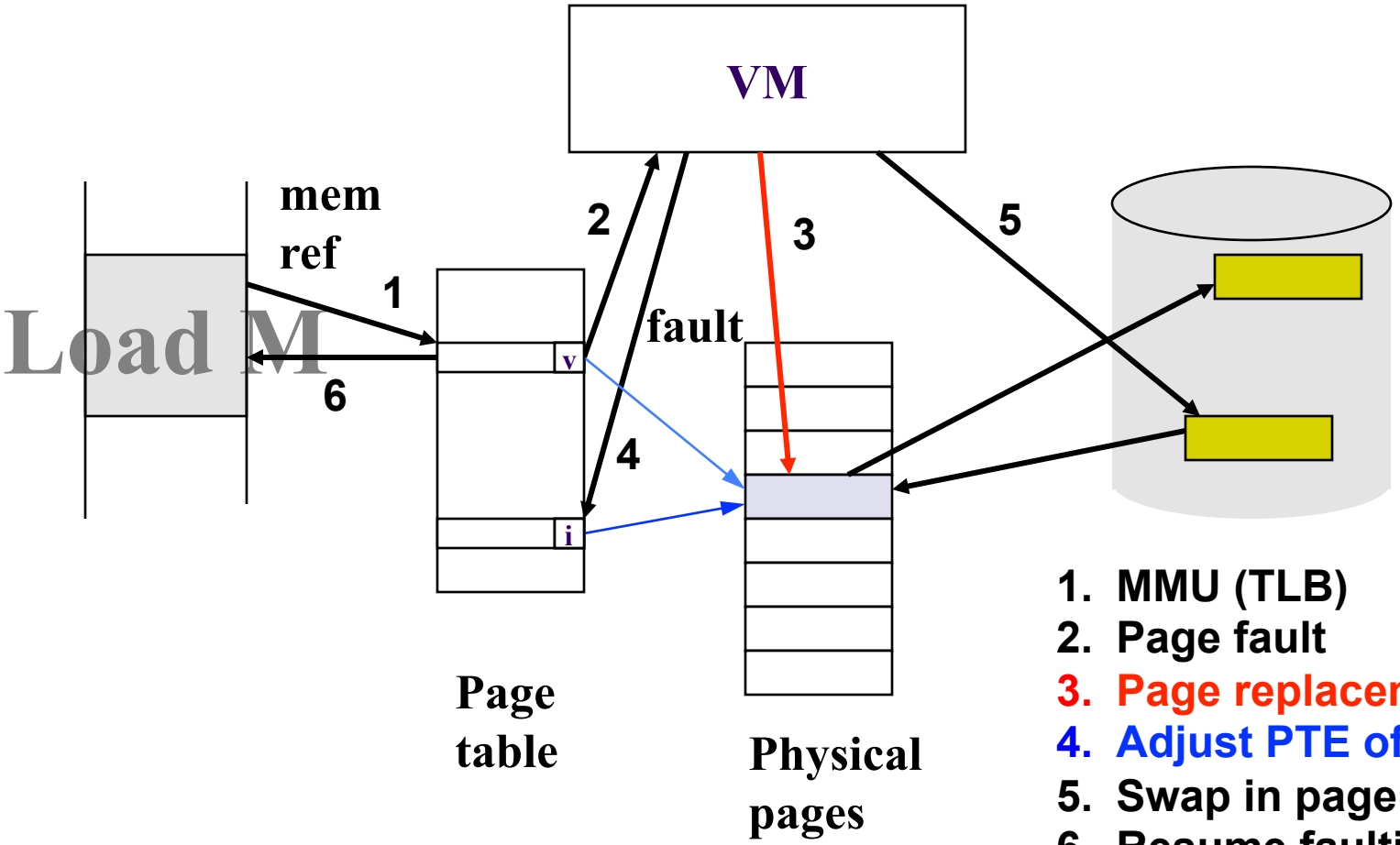


- Definition: *Virtual memory* permits a process to run with only some of its virtual address space loaded into physical memory

- Virtual address space translated to either
  - Physical memory (small, fast) or
  - Disk (backing store), large but slow

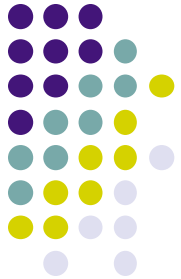
- Objective:
  - To produce the illusion of memory as big as necessary

[lec16] Page Fault Handling in demand paging



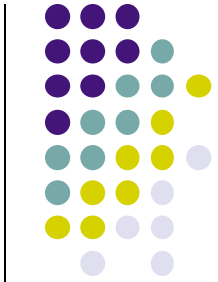
1. MMU (TLB)
2. Page fault
3. Page replacement
4. Adjust PTE of victim pg
5. Swap in page from swap
6. Resume faulting intr

[lec16] The Policy Issue: Page selection and replacement



- Once the hardware has provided basic capabilities for VM, the OS must make two kinds of decisions
  - Page selection: *when* to bring pages into memory
  - Page replacement: *which* page(s) should be thrown out
    - Try to kick out the least useful page

Page replacement problem



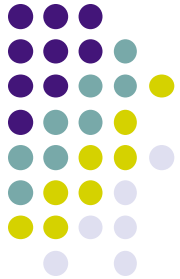
Definition:

- Expect to run with all physical pages in use
- Every "page-in" requires an eviction to swap space
- How to select the victim page?

- Performance goal:
  - Give a sequence of memory accesses, minimize the # of page faults
  - → given a sequence of virtual page references, minimize the # of page faults

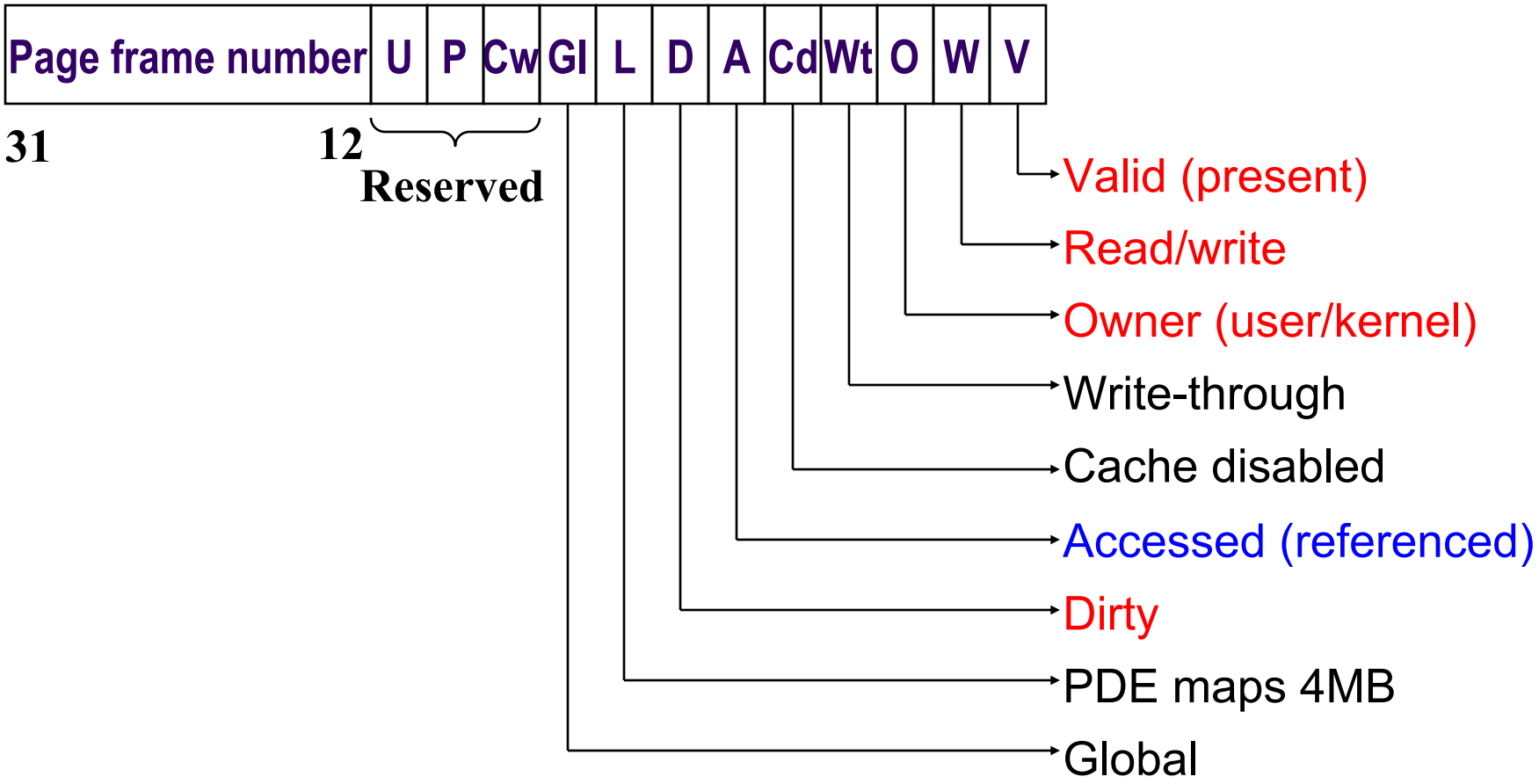
- Intuition: kick out the page that is least useful
- Challenge: how do you determine "least useful"?
  - Even if you know future?

Exploiting locality needs some hardware support

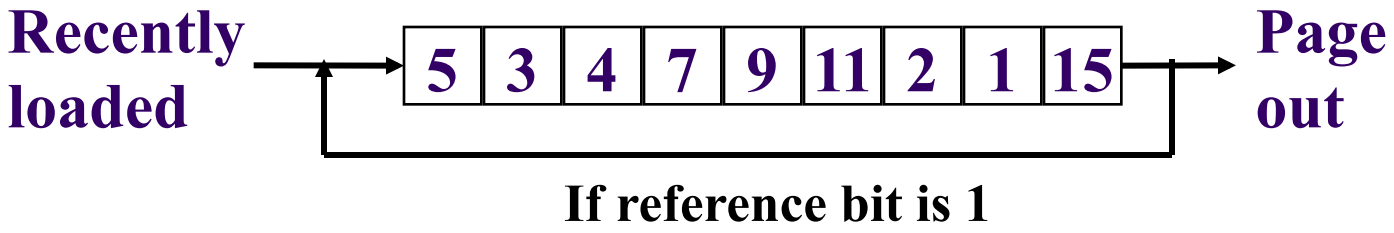


- Reference bit
  - A hardware bit that is set whenever the page is referenced (read or written)

x86 Page Table Entry

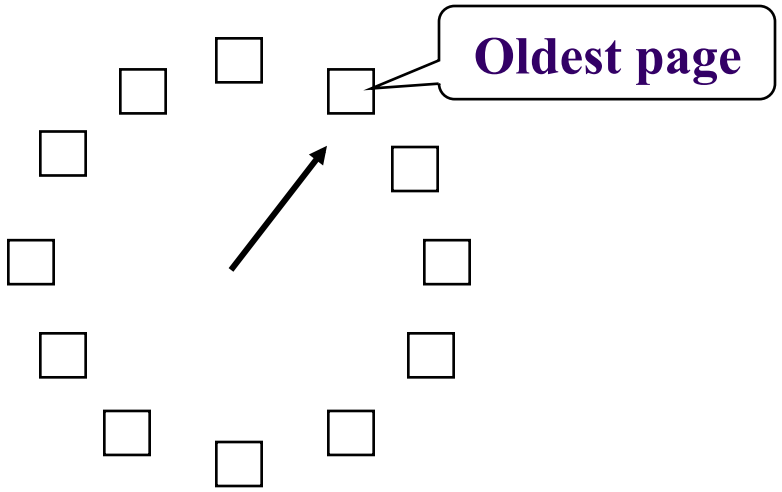


FIFO with Second Chance



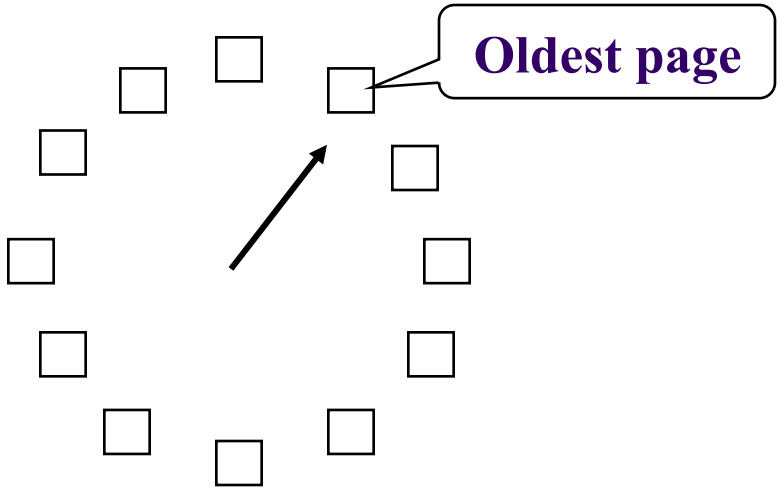
- Algorithm
  - Check the reference-bit of the oldest page (first in)
  - If it is 0, then replace it
  - If it is 1, clear the referent-bit, put it to the end of the list, and continue searching
- Pros
  - Fast
  - Frequency → do not replace a heavily used page
- Cons
  - The worst case may take a long time

Clock: a simple FIFO with 2nd chance



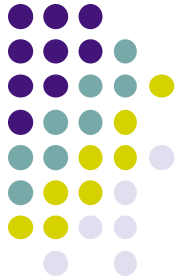
- FIFO clock algorithm
  - Hand points to the oldest page
  - On a page fault, follow the hand to inspect pages
- Second chance
  - If the reference bit is 1, set it to 0 and advance the hand
  - If the reference bit is 0, use it for replacement
- What is the difference between Clock and the previous one?
  - Mechanism vs. policy?

Clock: a simple FIFO with 2nd chance



- What happens if all reference bits are 1?
- What does it suggest if observing clock hand is sweeping very fast?
- What does it suggest if clock hand is sweeping very slow?

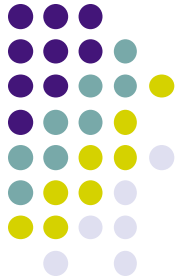
We've focused on miss rate. What about miss latency?



- Key observation: it is cheaper to pick a "clean" page over a "dirty" page
  - Clean page does not need to be swapped to disk

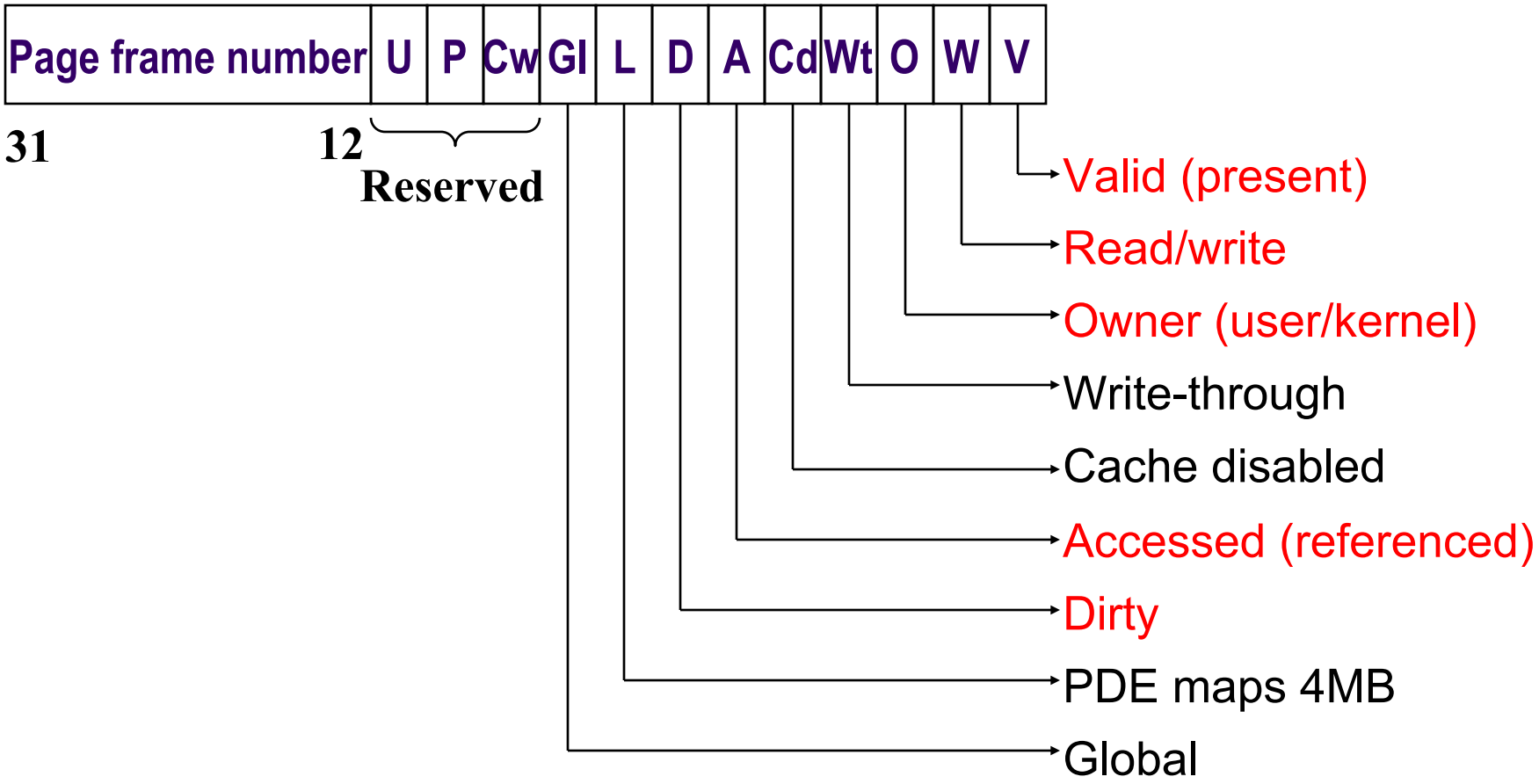
- Challenge:
  - How to get this info?

Refinement by adding extra hardware support

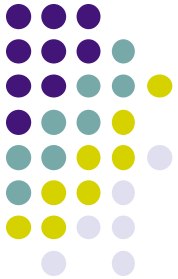


- Reference bit
  - A hardware bit that is set whenever the page is referenced (read or written)
- Modified bit (dirty bit)
  - A hardware bit that is set whenever the page is written into

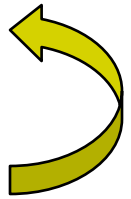
x86 Page Table Entry



Enhanced FIFO with 2nd-Chance Algorithm (used in Macintosh VM)

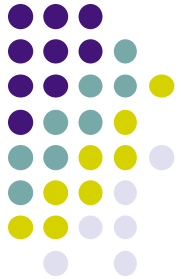


- Same as the basic FIFO with 2nd chance, except that it considers both (reference bit, modified bit)
  - (0,0): neither recently used nor modified (good)
  - (0,1): not recently used but dirty (not as good)
  - (1,0): recently used but clean (not good)
  - (1,1): recently used and dirty (bad)
  - When giving second chance, only clear reference bit



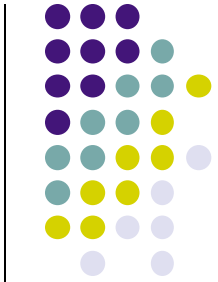
- Pros
  - Avoid write back
- Cons
  - More complicated, worse case scans multiple rounds

Enhanced FIFO with 2nd-Chance Algorithm – implementation



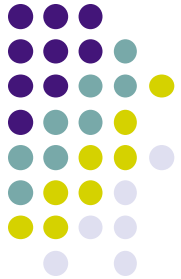
- On page fault, follow hand to inspect pages:
  - Round 1:
    - If bits are (0,0), take it
    - if bits are (0,1), record 1st instance
    - Clear ref bit for (1,0) and (1,1), if (0,1) not found yet
  - At end of round 1, if (0,1) was found, take it
  - If round 1 does not succeed, try 1 more round

Belady's anomaly



Belady's anomaly states that it is possible to have more page faults when increasing the number of page frames while using FIFO method of frame management. Laszlo Belady demonstrated this in 1970. Previously, it was believed that an increase in the number of page frames would always provide the same number or fewer page faults.

Example



Page Requests

3210324321 04

Example (Page Faults in Red)

Page Requests – 3 frames

Frame 1

Frame 2

Frame 3

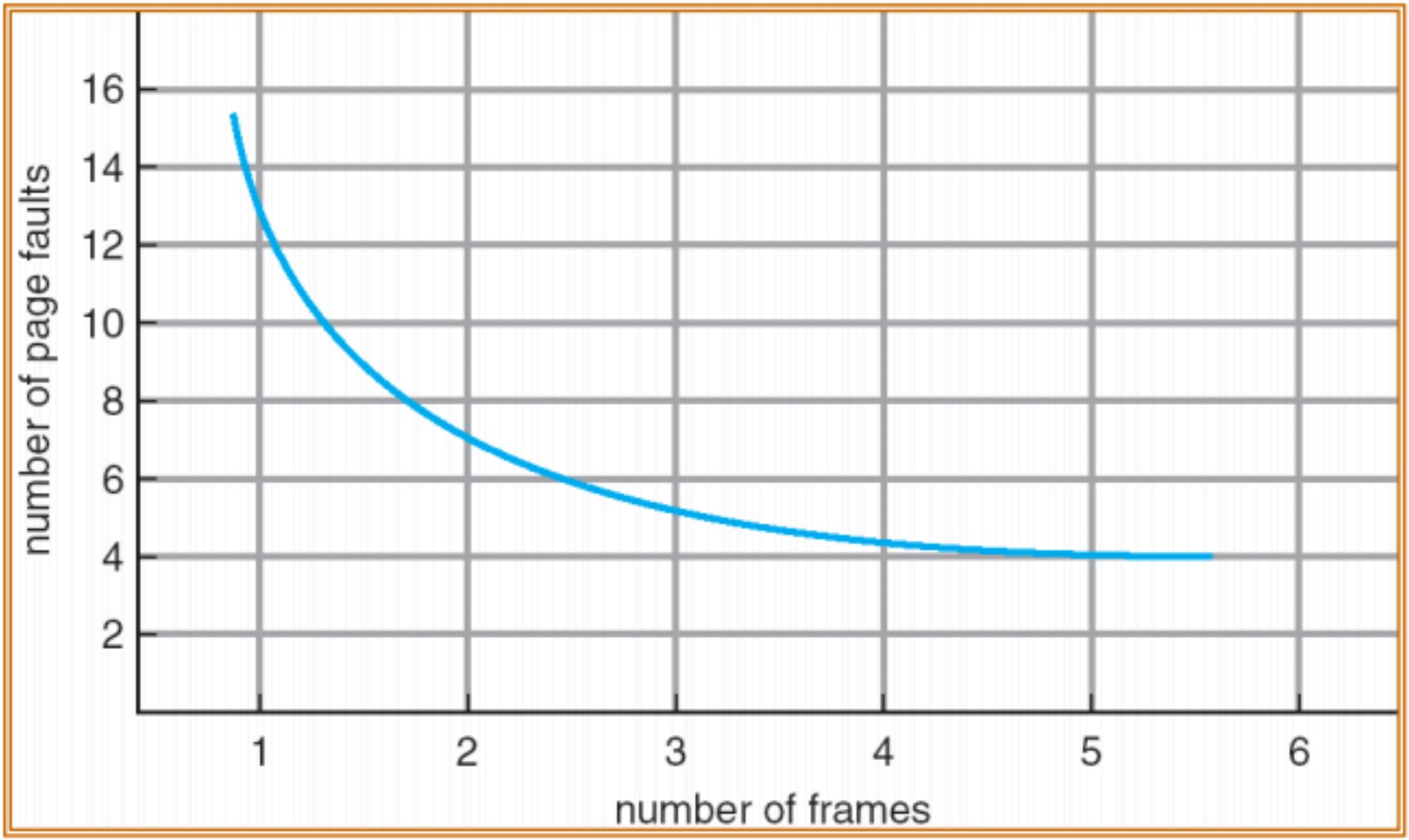
3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	0	0	0	4	4	4	4	4	4
	2	2	2	3	3	3	3	3	1	1	1
		1	1	1	2	2	2	2	2	0	0

Example (Page Faults in Red)

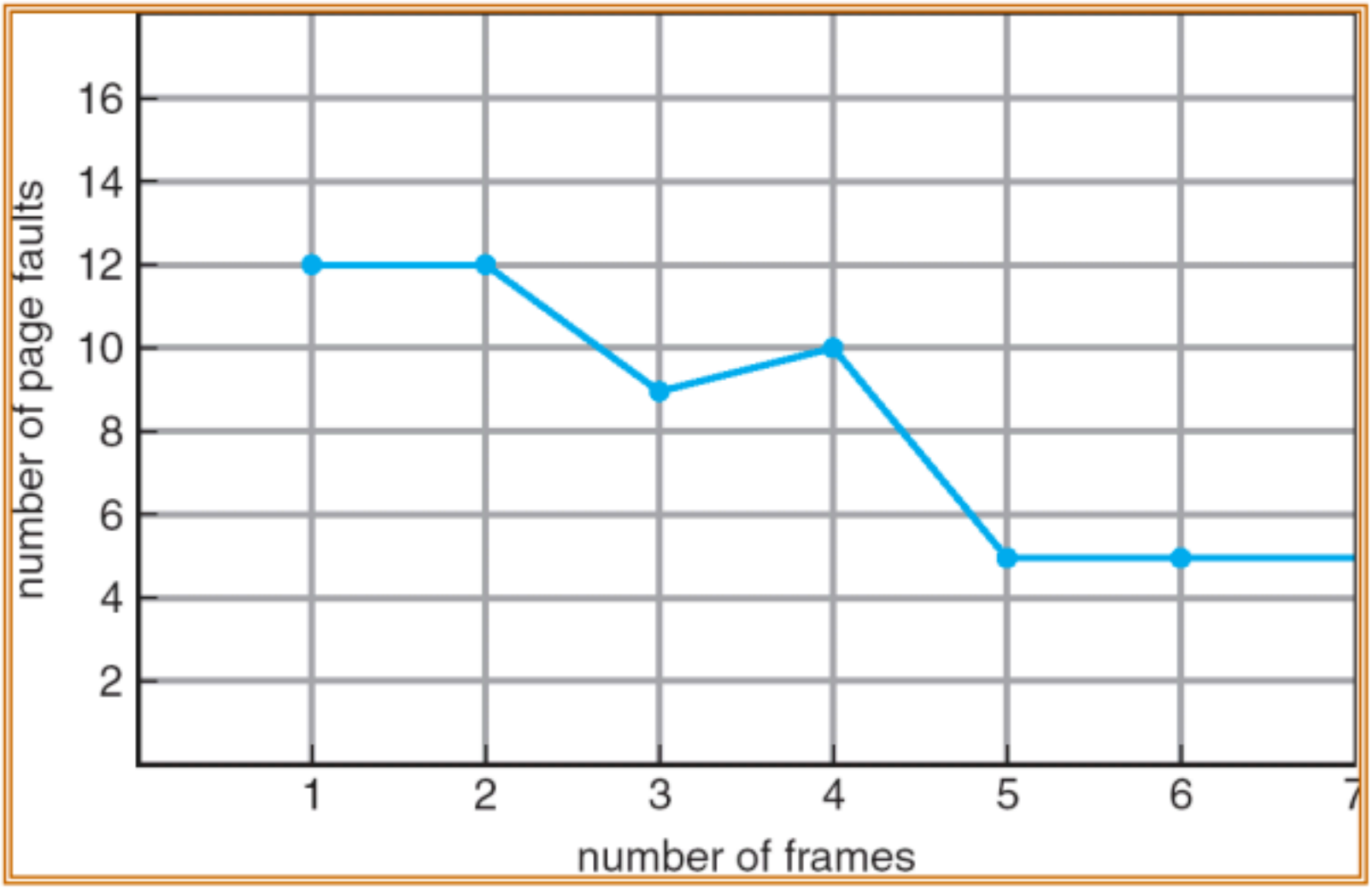
- Page Requests – 4 frames
- Frame 1
- Frame 2
- Frame 3
- Frame 4

3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	3	3	3	4	4	4	4	0	0
	2	2	2	2	2	2	3	3	3	3	4
		1	1	1	1	1	1	2	2	2	2
			0	0	0	0	0	0	1	1	1

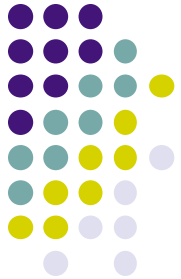
Ideal curve of # of page faults v.s. # of physical pages



FIFO illustrating Belady's anomaly



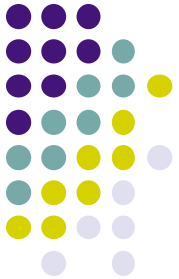
break



Dennis Ritchie

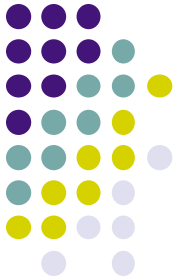
- Father of C and UNIX

Ken Thompson



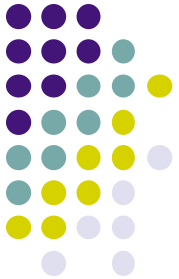
- Co-creator of UNIX

Midterm topics



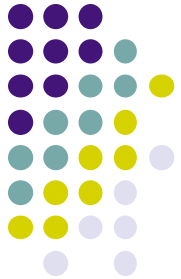
- Introduction
- Processes
- Threads
- Synchronization (and wait-free sync)
- IPC with messages
- CPU scheduling
- Deadlocks
- Memory management (including today's lec)

Format of midterm

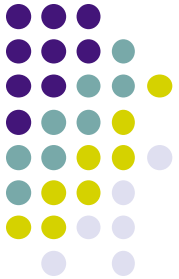


- 16 True or False (32 pts)
- 5 multiple choices (10 pts)
- 3 short-answer questions (12 pts)
- 3 big problems (16+20+10 pts)
  - Each big problem split into few smaller problems
  - Write (debug) some code

Midterm Review 1 – Key Concepts

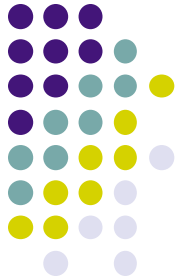


Intro



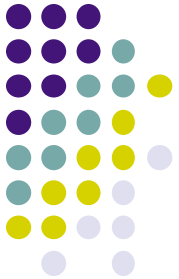
- Batch, multiprogramming, timesharing systems
- Two definitions of OS
- System calls
- Interrupts

Process and Thread Management



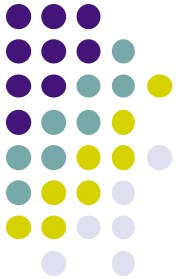
- Process creation
  - fork, exec
- Process/thread management metadata
  - PCB/TCB
  - Context/thread switch
- Why thread and what is thread?
- User-level and kernel-level thread implementation
- Inter-process communication
  - Shared data
  - Message passing

Synchronization



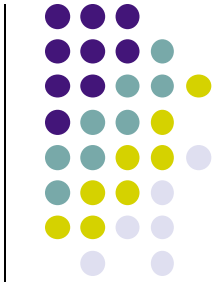
- Mutual exclusion and critical section
- Locks
- Semaphore
- Conditional variables
- Monitor

Synchronization Problems



- Producer/Consumer
  - Semaphore solution
  - Conditional variable solution
  - Monitor solution
- Readers/Writers
  - Reader preference solution
  - Writer preference solution
  - Fair solution
- Dining philosopher

Synchronization Implementation

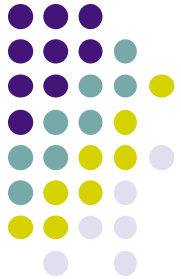


- Uniprocessor implementation

- Multiprocessor implementation
  - Test-and-set
  - Load-linked and store-conditional

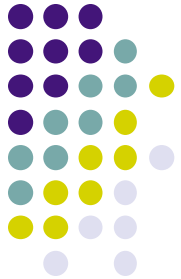
  - Busy wait
  - Wait-free synchronization

Concurrency bugs

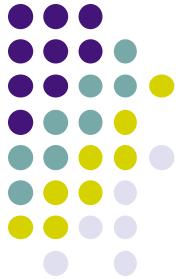


- Deadlock, starvation, livelock
- Conditions of deadlock
- Strategies to prevent/break deadlocks
- Non-deadlock concurrency bugs

CPU Scheduling



- Goals
  - Minimize turnaround time
  - Maximize throughput
  - Short response time
  - Fairness
- Scheduling algorithms
  - FIFO
  - RR
  - SJCF and SRTCF
  - Priority scheduling and MLFQ
  - Lottery



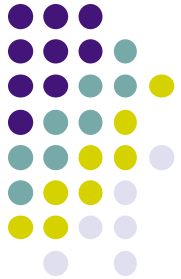
Memory Management

- The process of compiling, linking, loading, and running a program (big picture)

- How linker works
  - segment relocation
  - address translation

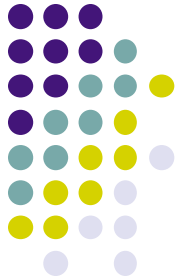
- Allocation and free
  - Segregated lists
  - Buddy allocation
  - Garbage collection

Memory Management



- Segmentation
  - Base & bound
  - Multiple segments
  - External fragmentation
- Paging
  - Multi-level paging
  - Inverted page table
  - TLB

Memory Management



- Virtual memory
  - Demand paging
  - Swapping