CSE 120: Principles of Operating Systems
Lecture 10: Virtual Memory

Prof. Joseph Pasquale
University of California, San Diego
Mar 6, 2023

Segments and Pages

- Structuring memory as segments/pages allows
  - partitioning memory for convenient allocation
  - reorganizing memory for convenient usage
- How?
  - Relocation via address translation
  - Protection via matching operations with objects
- Result: a logically organized memory

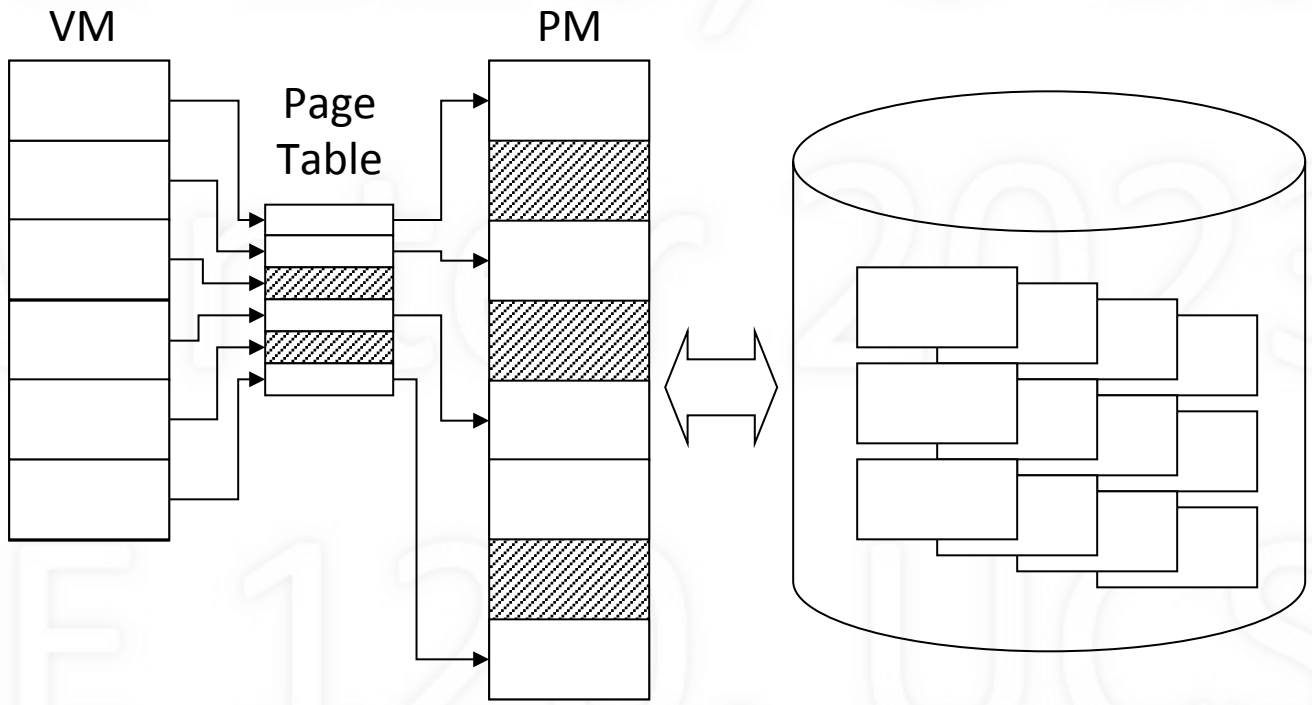
Implications

- Not all pieces need to be in memory
  - Need only piece being referenced
  - Other pieces can be on disk
  - Bring pieces in only when needed
- Illusion: there is much more memory
- What's needed to support this idea?
  - A way to identify whether a piece is in memory
  - A way to bring in a piece (from where, to where?)
  - Relocation (address translation, which we have)

From Logical to Virtual Memory

- Logical memory becomes virtual memory
  - Still logical (separate organization from physical)
  - Virtual: memory seems to exist, regardless of how
- Virtual memory: illusion of large memory
  - Keep only portion of logical memory in physical
  - Rest is kept on disk (larger, slower, cheaper)
  - Unit of memory is segment or page (or both)
- Logical address space → virtual address space

Virtual Memory based on Paging



- For all pages in virtual memory
  - All of them reside on disk
  - Some also reside in physical memory (which ones?)

Sample Contents of Page Table Entry

Valid	Ref	Mod	Frame number	Prot: rwx

- Valid: is entry valid (page in physical memory)?
- Ref: has this page been referenced yet?
- Mod: has this page been modified (dirty)?
- Frame: what frame is this page in?
- Prot: what are the allowable operations?

Address Translation and Page Faults

- Get entry: index page table with page number
- If valid bit is off, page fault – trap into kernel
  - Find page on disk (kept in kernel data structure)
  - Read it into a free frame
    - may need to make room: page replacement
  - Record frame number in page table entry
  - Set valid bit (and other fields)
- Retry instruction (return from page-fault trap)

Faults under Segmentation/Paging

- Virtual address: <segment s, page p, offset i>
- Use s to index segment table (gets page table)
  - May get a segment fault
- Check bound (Is p < bound?)
  - May get a segmentation violation
- Use p to index into page table (to get frame f)
  - May get a page fault
- Physical address: concatenate f and offset i

Page Faults are Expensive

- Disk: 5-6 orders magnitude slower than RAM
  - Very expensive; but if very rare, tolerable
- Example
  - RAM access time: 100 nsec
  - Disk access time: 10 msec
  - p = page fault probability
  - Effective access time: 100 + p × 10,000,000 nsec
  - If p = 0.1%, effective access time = 10,100 nsec !

If a memory access were to take 1 sec, a disk access would take 1-10 days!

Principle of Locality

- Not all pieces referenced uniformly over time
  - Make sure most referenced pieces in memory
  - If not, thrashing: constant fetching of pieces
- References cluster in time/space
  - Will be to same or neighboring areas
  - Allows prediction based on past

Page Replacement Policy

- Goal: remove page not in locality of reference
- Page replacement is about
  - which page(s) to remove
  - when to remove them
- How to do it in cheapest way possible, with
  - least amount of additional hardware
  - least amount of software overhead

Basic Page Replacement Algorithms

- FIFO: select page that is oldest
  - Simple: use frame ordering
  - Doesn't perform very well (oldest may be popular)
- OPT: select page to be used furthest in future
  - Optimal, but requires future knowledge
  - Establishes best case, good for comparisons
- LRU: select page that was least recently used
  - Predict future based on past; works given locality
  - Costly: time-stamp pages each access, find least

Reference String

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

- Reference string: sequence of page references
- Page reference: page of logical/virtual address

FIFO: First In First Out

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔												

- Arrow ➔ always points to next frame to fill
- For FIFO, this is always frame with oldest page
- But, is oldest page the best page to remove?

FIFO Example

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*											
1 fault												

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2										
		3*										
2 faults ➔												

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2									
		3*	3									
2 faults ➔												

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*	2	2	2								
		3*	3	3								
3 faults				1*								

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2	2	5*							
➔		3*	3	3	3							
4 faults				1*	1							

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2	2	5*	5						
		3*	3	3	3	2*						
5 faults ➔				1*	1	1						

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*	2	2	2	5*	5	5					
		3*	3	3	3	2*	2					
6 faults				1*	1	1	4*					

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*	2	2	2	5*	5	5	5				
		3*	3	3	3	2*	2	2				
6 faults				1*	1	1	4*	4				

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2	2	5*	5	5	5	3*			
➔		3*	3	3	3	2*	2	2	2			
7 faults				1*	1	1	4*	4	4			

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2	2	5*	5	5	5	3*	3		
➔		3*	3	3	3	2*	2	2	2	2		
7 faults				1*	1	1	4*	4	4	4		

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO	2*	2	2	2	5*	5	5	5	3*	3	3	
		3*	3	3	3	2*	2	2	2	2	5*	
8 faults ➔				1*	1	1	4*	4	4	4	4	

FIFO Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*	2	2	2	5*	5	5	5	3*	3	3	3
		3*	3	3	3	2*	2	2	2	2	5*	5
9 faults				1*	1	1	4*	4	4	4	4	2*

Summary of FIFO

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

FIFO ➔	2*	2	2	2	5*	5	5	5	3*	3	3	3
		3*	3	3	3	2*	2	2	2	2	5*	5
9 faults				1*	1	1	4*	4	4	4	4	2*

- FIFO incurs 9 page faults (5 are obligatory)
- FIFO is simple to implement
  - Just keep pointer to next frame after last loaded
- But, removing oldest page not generally best
  - Old does not imply useless; may still be in demand

OPT: Optimal Page Replacement

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT ➔												

- Optimal: replace page that will be accessed furthest in future (arrow ➔ points to frame)

OPT Example

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2								
		3*	3	3								
3 faults ➔				1*								

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2	2							
➔		3*	3	3	3							
4 faults				1*	5*							

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT ➔	2*	2	2	2	2	2						
		3*	3	3	3	3						
4 faults				1*	5*	5						

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT ➔	2*	2	2	2	2	2	4*					
		3*	3	3	3	3	3					
5 faults				1*	5*	5	5					

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT ➔	2*	2	2	2	2	2	4*	4				
		3*	3	3	3	3	3	3				
5 faults				1*	5*	5	5	5				

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT ➔	2*	2	2	2	2	2	4*	4	4			
		3*	3	3	3	3	3	3	3			
5 faults				1*	5*	5	5	5	5			

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2	2	2	4*	4	4	2*		
➔		3*	3	3	3	3	3	3	3	3		
6 faults				1*	5*	5	5	5	5	5		

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2	2	2	4*	4	4	2*	2	
➔		3*	3	3	3	3	3	3	3	3	3	
6 faults				1*	5*	5	5	5	5	5	5	

OPT Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2	2	2	4*	4	4	2*	2	2
➔		3*	3	3	3	3	3	3	3	3	3	3
6 faults				1*	5*	5	5	5	5	5	5	5

Summary of OPT

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

OPT	2*	2	2	2	2	2	4*	4	4	2*	2	2
➔		3*	3	3	3	3	3	3	3	3	3	3
6 faults				1*	5*	5	5	5	5	5	5	5

- OPT incurs 6 page faults, 1 beyond obligatory
  - This is the minimal number possible
- OPT is optimal, but not realistic
  - Requires predicting the future
  - Useful as a benchmark

FIFO vs. OPT

Reference string	2	3	2	1	5	2	4	5	3	2	5	2
FIFO ➔	2*	2	2	2	5*	5	5	5	3*	3	3	3
		3*	3	3	3	2*	2	2	2	2	5*	5
9 = 5 + 4 faults				1*	1	1	4*	4	4	4	4	2*

OPT	2*	2	2	2	2	2	4*	4	4	2*	2	2
➔		3*	3	3	3	3	3	3	3	3	3	3
6 = 5 + 1 faults				1*	5*	5	5	5	5	5	5	5

- OPT does much better than FIFO

LRU: Least Recently Used

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔												

- Replace page that was least recently used
  - LRU means used furthest in the past
- Takes advantage of locality of reference
- Must have some way of tracking LRU page

LRU Example

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU	2*	2	2	2								
➔		3*	3	3								
3 faults				1*								

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔	2*	2	2	2	2							
		3*	3	3	5*							
4 faults				1*	1							

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU	2*	2	2	2	2	2						
		3*	3	3	5*	5						
4 faults ➔				1*	1	1						

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU	2*	2	2	2	2	2	2					
➔		3*	3	3	5*	5	5					
5 faults				1*	1	1	4*					

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔	2*	2	2	2	2	2	2	2				
		3*	3	3	5*	5	5	5				
5 faults				1*	1	1	4*	4				

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU	2*	2	2	2	2	2	2	2	3*			
		3*	3	3	5*	5	5	5	5			
6 faults ➔				1*	1	1	4*	4	4			

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU	2*	2	2	2	2	2	2	2	3*	3		
➔		3*	3	3	5*	5	5	5	5	5		
7 faults				1*	1	1	4*	4	4	2*		

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔	2*	2	2	2	2	2	2	2	3*	3	3	
		3*	3	3	5*	5	5	5	5	5	5	
7 faults				1*	1	1	4*	4	4	2*	2	

LRU Example, continued

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔	2*	2	2	2	2	2	2	2	3*	3	3	3
		3*	3	3	5*	5	5	5	5	5	5	5
7 faults				1*	1	1	4*	4	4	2*	2	2

Summary of LRU

Reference string	2	3	2	1	5	2	4	5	3	2	5	2

LRU ➔	2*	2	2	2	2	2	2	2	3*	3	3	3
		3*	3	3	5*	5	5	5	5	5	5	5
7 faults				1*	1	1	4*	4	4	2*	2	2

- LRU incurs 7 page faults, 2 beyond obligatory
- Performs well, but only if there is locality
- Complex, requires hardware support
  - To keep track of frame with LRU page

FIFO vs. OPT vs. LRU

Reference string	2	3	2	1	5	2	4	5	3	2	5	2
FIFO ➔	2*	2	2	2	5*	5	5	5	3*	3	3	3
		3*	3	3	3	2*	2	2	2	2	5*	5
9 = 5 + 4 faults				1*	1	1	4*	4	4	4	4	2*
OPT	2*	2	2	2	2	2	4*	4	4	2*	2	2
➔		3*	3	3	3	3	3	3	3	3	3	3
6 = 5 + 1 faults				1*	5*	5	5	5	5	5	5	5
LRU ➔	2*	2	2	2	2	2	2	2	3*	3	3	3
		3*	3	3	5*	5	5	5	5	5	5	5
7 = 5 + 2 faults				1*	1	1	4*	4	4	2*	2	2

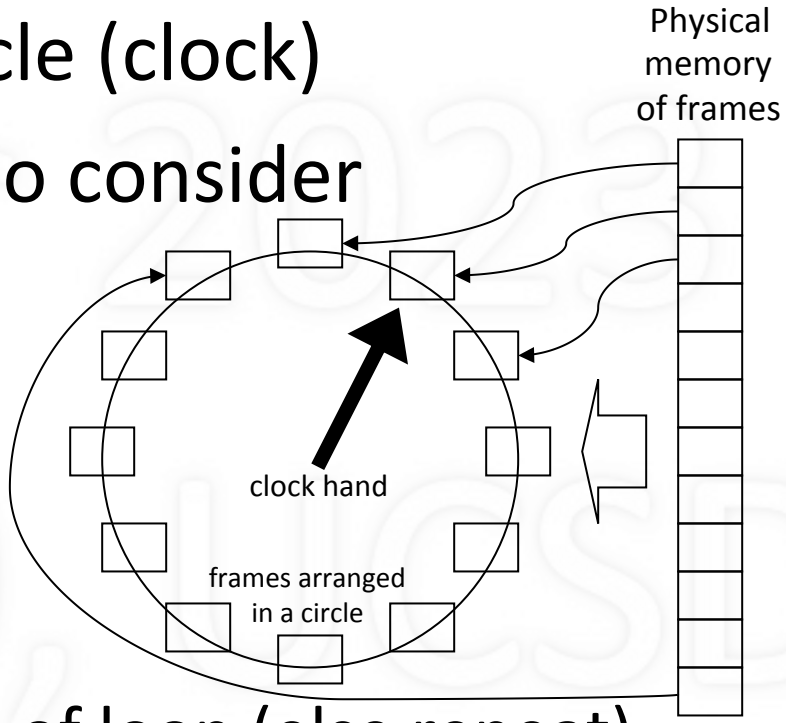
- OPT ≥ LRU (assuming locality) ≳ FIFO

Approximating LRU: Clock Algorithm

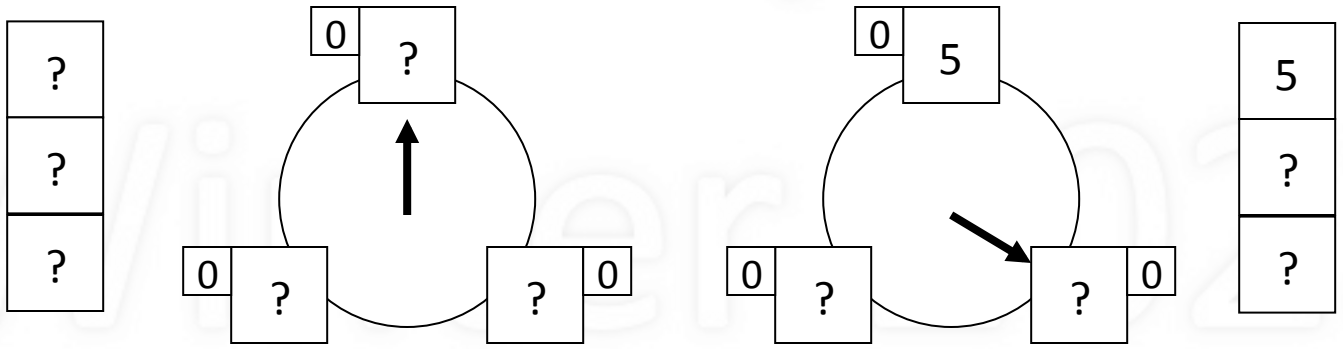
- Select page that is old and not recently used
  - Clock (second chance) is approximation of LRU
- Hardware support: reference bit
  - Associated with each frame is a reference bit
  - Actually, reference bit is in page table entry
- How reference bit is used
  - When frame filled with page, set bit to 0 (by OS)
  - If frame is accessed, set bit to 1 (by hardware)

How Clock Works

- Arrange all frames in circle (clock)
- Clock hand: next frame to consider
- Page fault: find frame
  - If ref bit 0, select frame
  - Else, set ref bit to 0
  - Advance clock hand
  - If frame found, break out of loop (else repeat)
- If frame had modified page, must write to disk

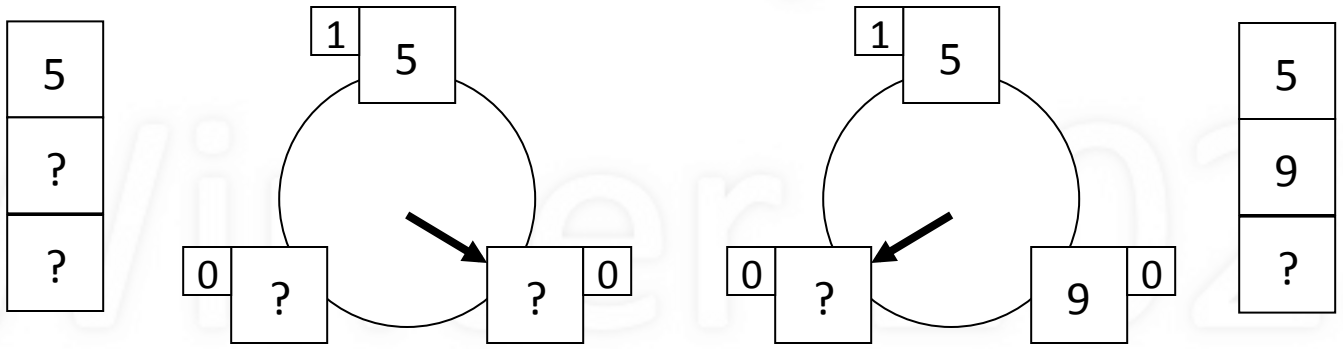


Example of Clock



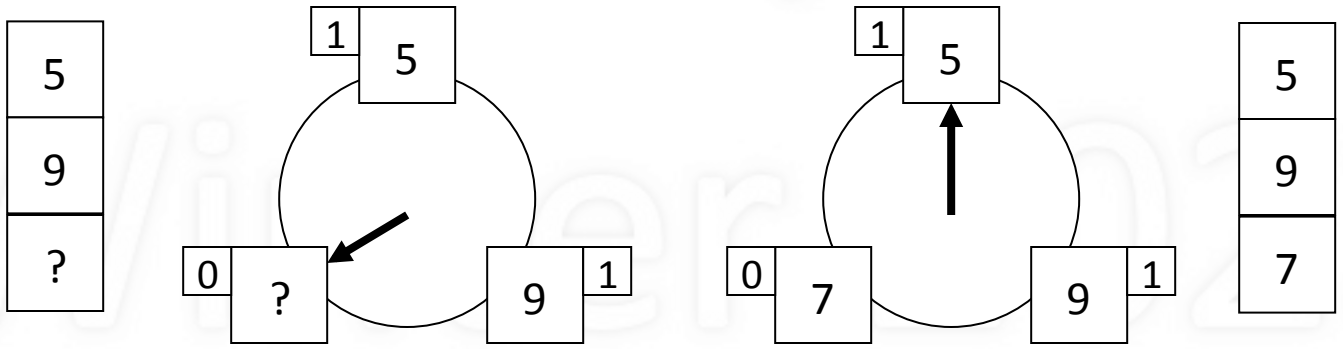
- Ref string: 5 9 7 1 9 5 9
- Reference page 5: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



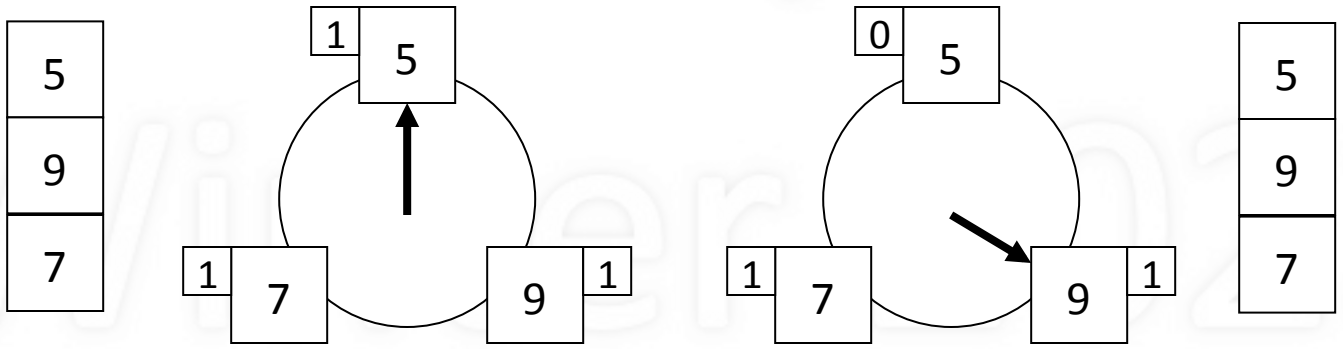
- Ref string: 5 9 7 1 9 5 9
- Reference page 9: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



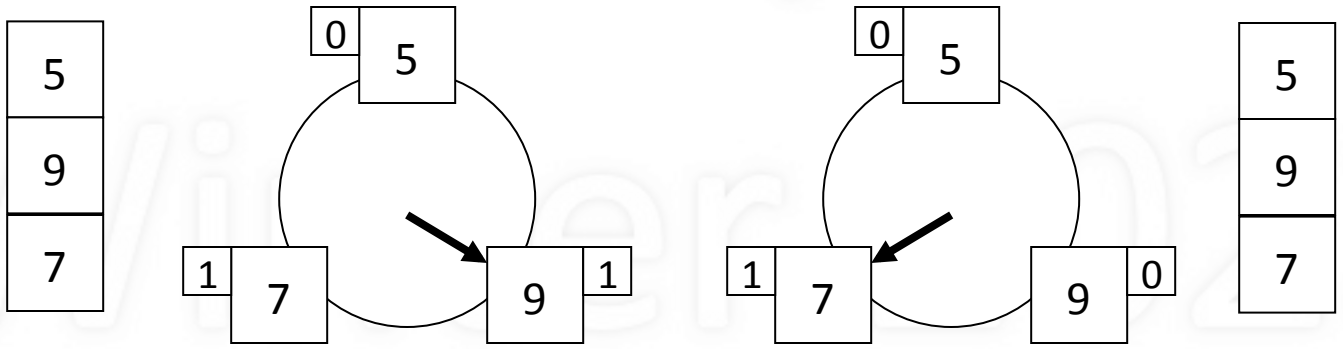
- Ref string: 5 9 7 1 9 5 9
- Reference page 7: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



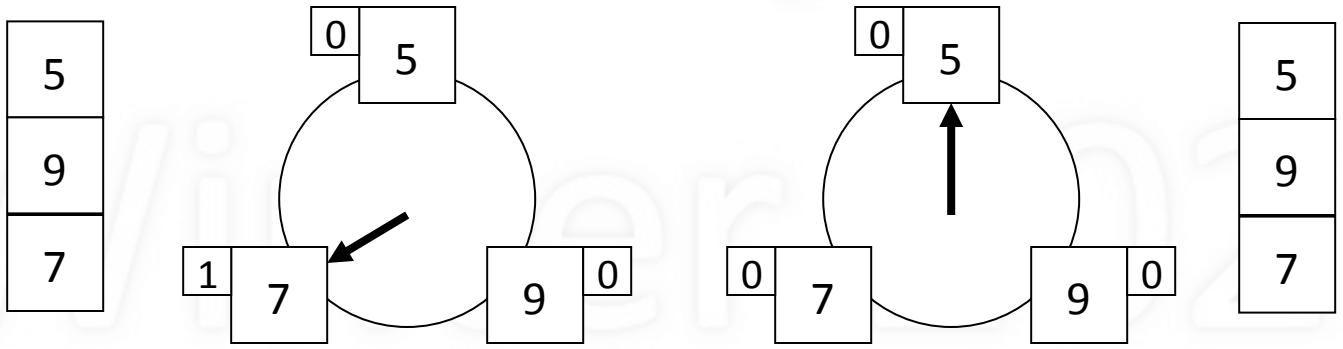
- Ref string: 5 9 7 1 9 5 9
- Reference page 1: page fault (1)
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



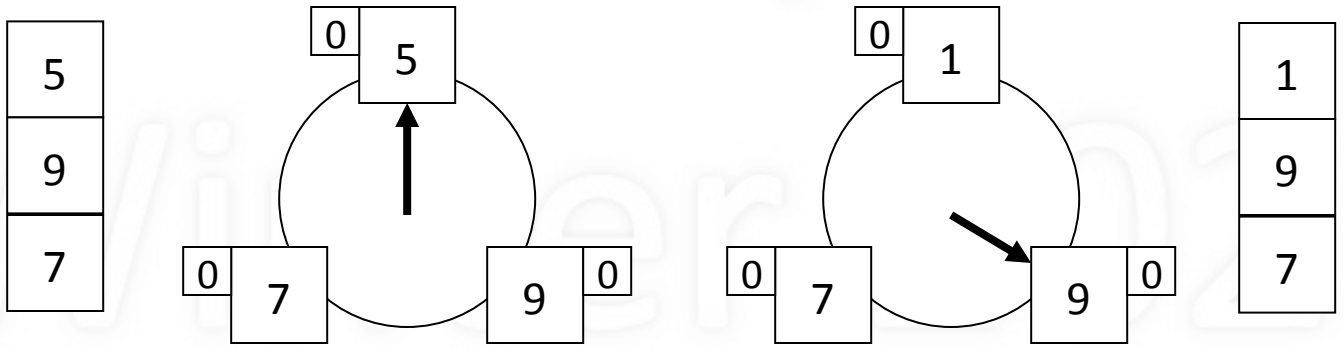
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



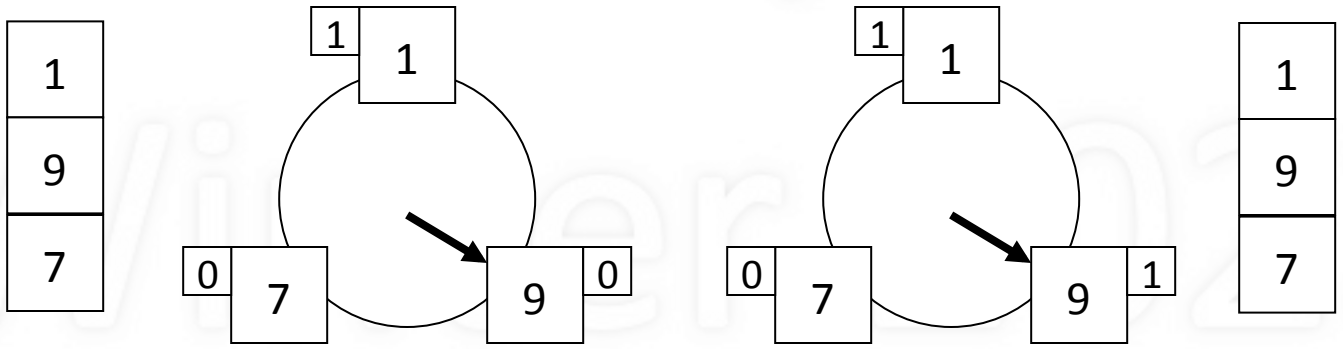
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



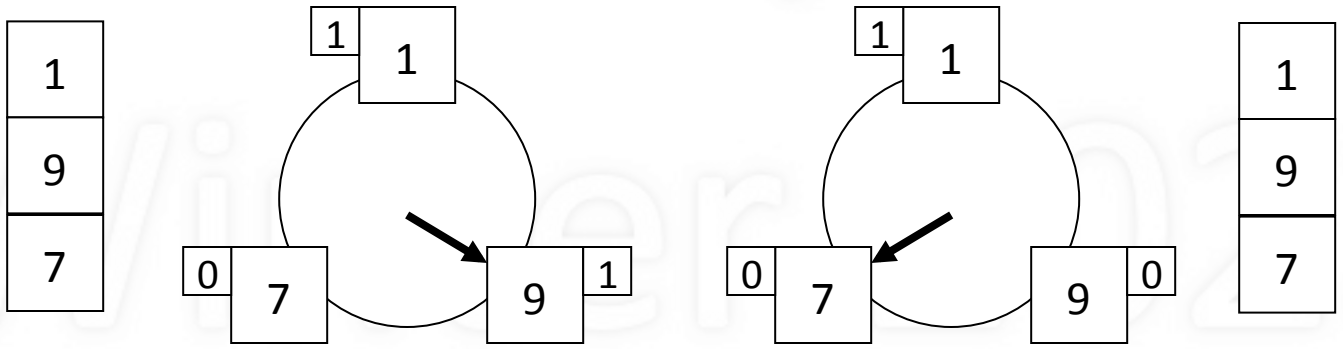
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



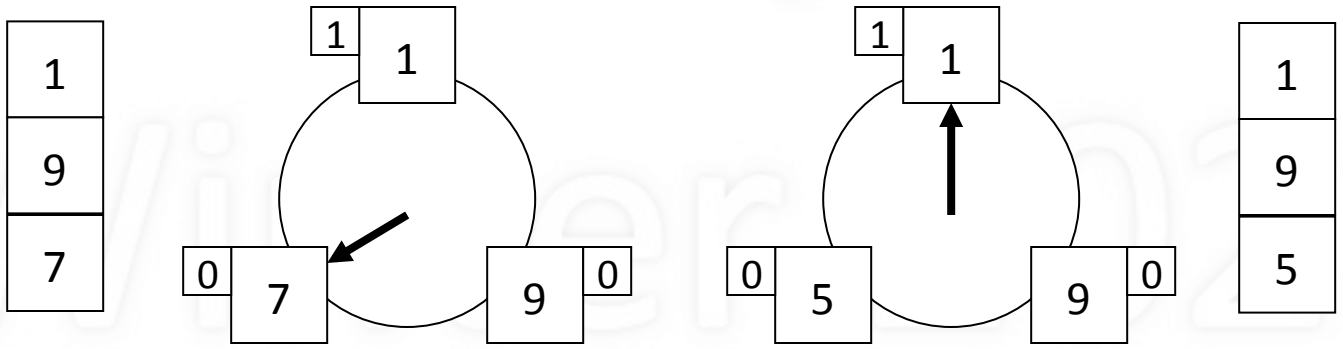
- Ref string: 5 9 7 1 9 5 9
- Reference page 9
  - Page 9 is already in memory: no page fault
  - OS does nothing, but *hardware* sets ref bit to 1

Example of Clock



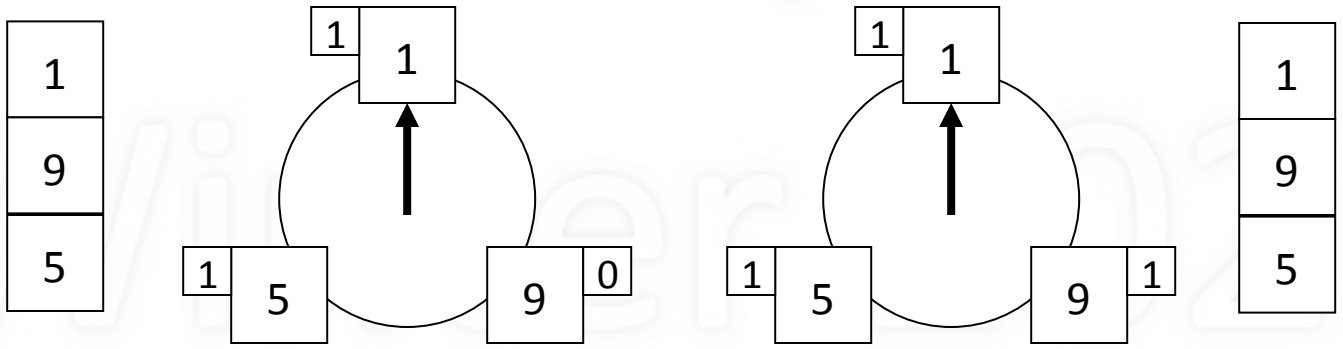
- Ref string: 5 9 7 1 9 5 9
- Reference page 5: page fault (2)
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock

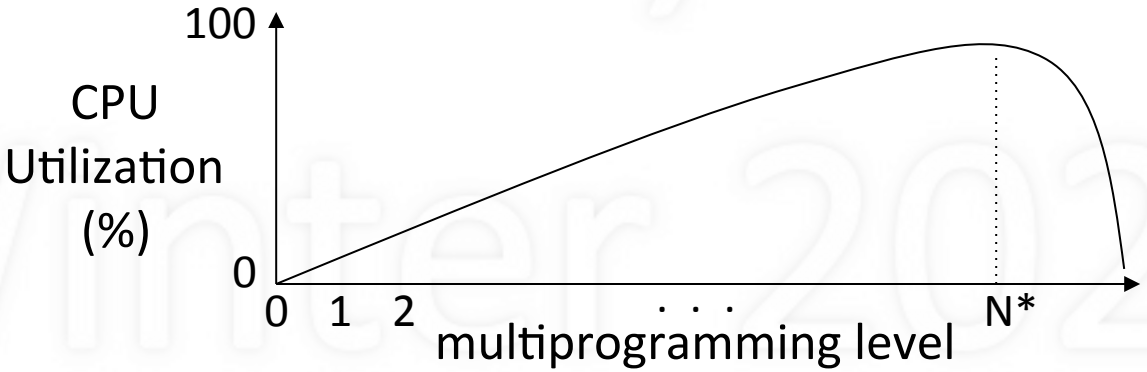


- Ref string: 5 9 7 1 9 5 9
- Reference page 9
  - Page 9 already in memory: no page fault
  - OS does nothing, but hardware sets ref bit to 1

Resident Set Management

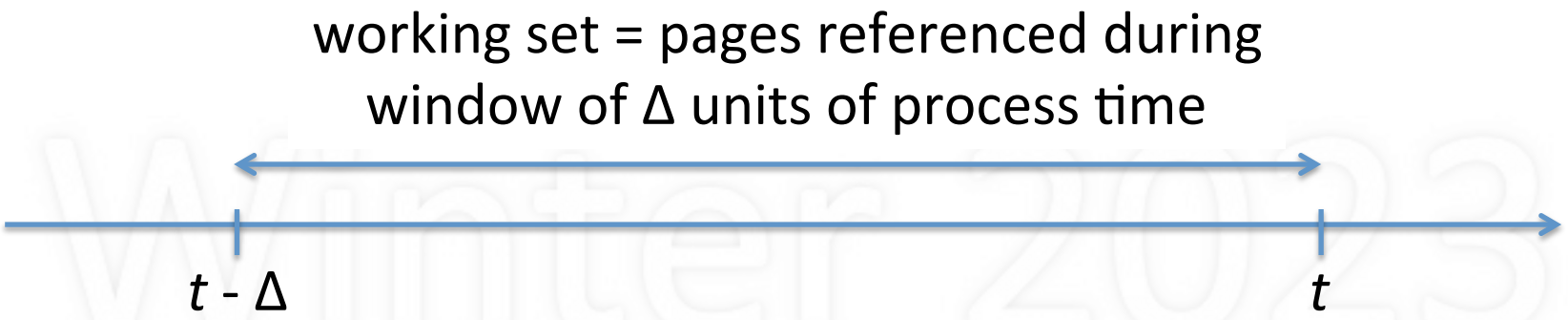
- Resident set: process's pages in physical memory
  - How big should resident set be? Which pages?
  - Who provides frame (same process or another)?
- Local: limit frame selection to requesting process
  - Isolates effects of page behavior on processes
  - Inefficient: some processes have unused frames
- Global: select any frame (from any process)
  - Efficient: resident sets grow/shrink accordingly
  - No isolation: process can negatively affect another

Multiprogramming Level



- Multiprogramming level: number of processes in physical memory (non-empty resident sets)
- Goal: increase multiprogramming level – how?
- However, beyond certain point: thrashing
- Resident set should contain the *working set*

Denning's Working Set Model



- Working set: W(t, Δ)
  - Pages referenced during last delta (process time)
- Process given frames to hold working set
- Add/remove pages according to W(t, Δ)
- If working set doesn't fit, swap process out

Denning's Working Set Model

- Working set is a local replacement policy
  - Process's page fault behavior doesn't affect others
- Problem: difficult to implement
  - Must timestamp pages in working set
  - Must determine if timestamp older than $t - \Delta$
  - How should Δ be determined?
- Contrast to Clock
  - Clock: simple, easy to implement, global policy

Summary

- Virtual memory
  - Logical memory: some in physical, all in secondary
  - Effective size of disk, effective speed of RAM
  - Efficient because of locality
- OPT ≥ LRU ≳ Clock ≳ FIFO
  - LRU and Clock work well assuming locality
- Goal: keep working set in memory
  - If working set cannot be resident, swap out

Textbook

- OSP: Chapter 10
- OSC: Read Chapter 10 (on Virtual Memory)
  - Lecture-related: 10.1-10.2, 10.4-10.6
  - Recommended: 10.3, 10.8-10.11