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

Prof. Joseph Pasquale
University of California, San Diego

February 13, 2019

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

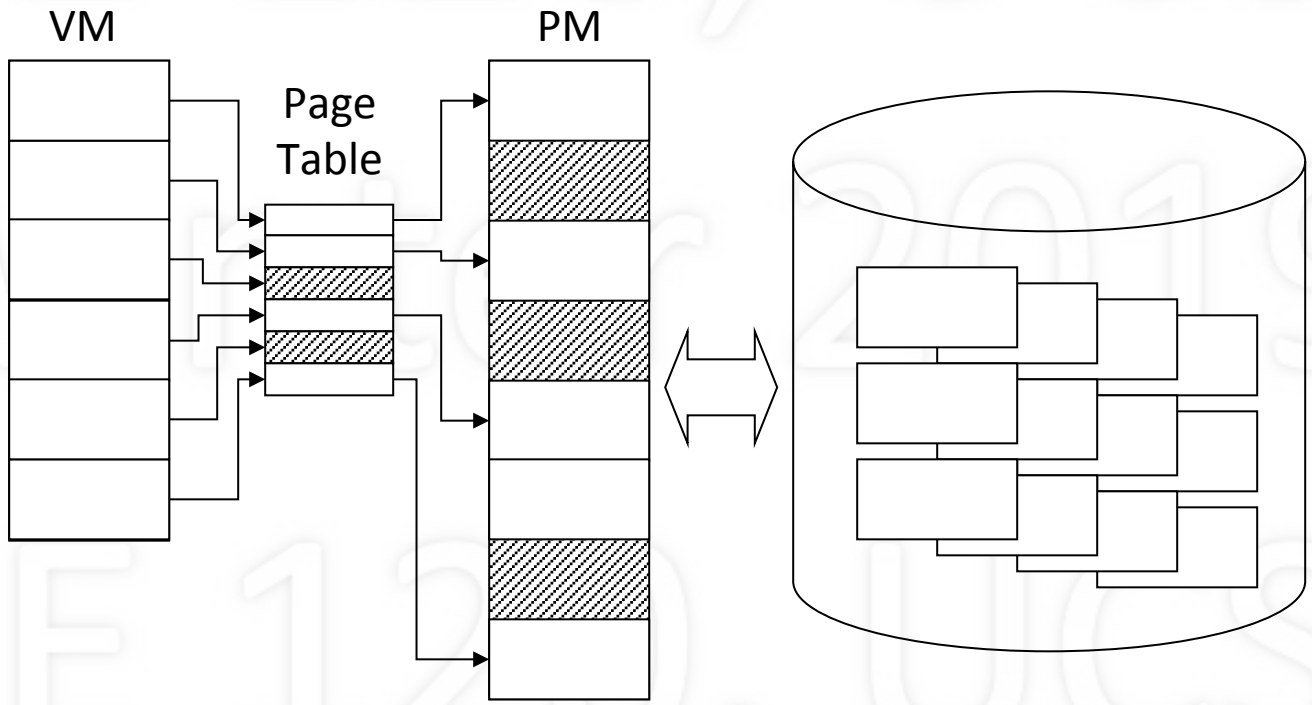
Implications

- 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

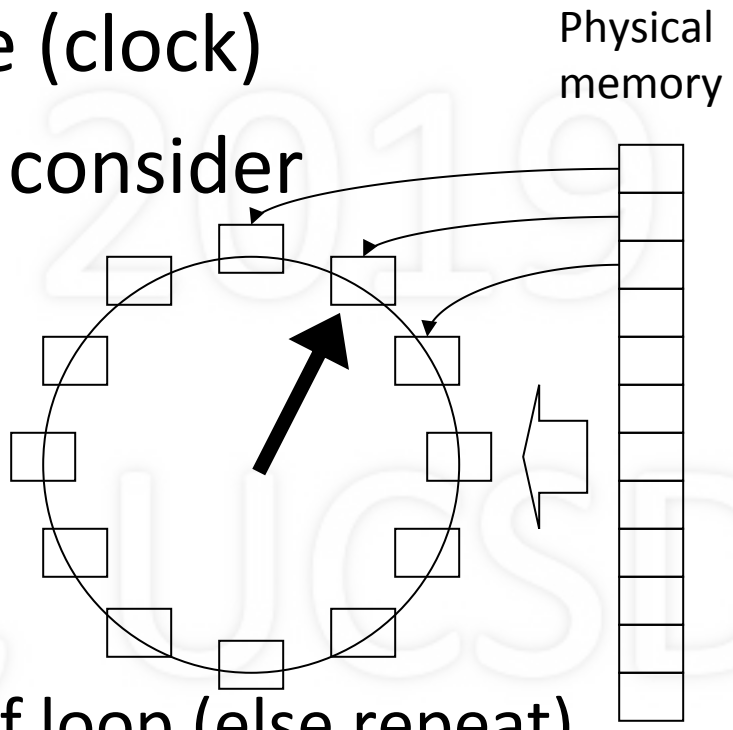
- OPT ≥ LRU (assuming locality) ≳ FIFO

Approximating LRU: Clock Algorithm

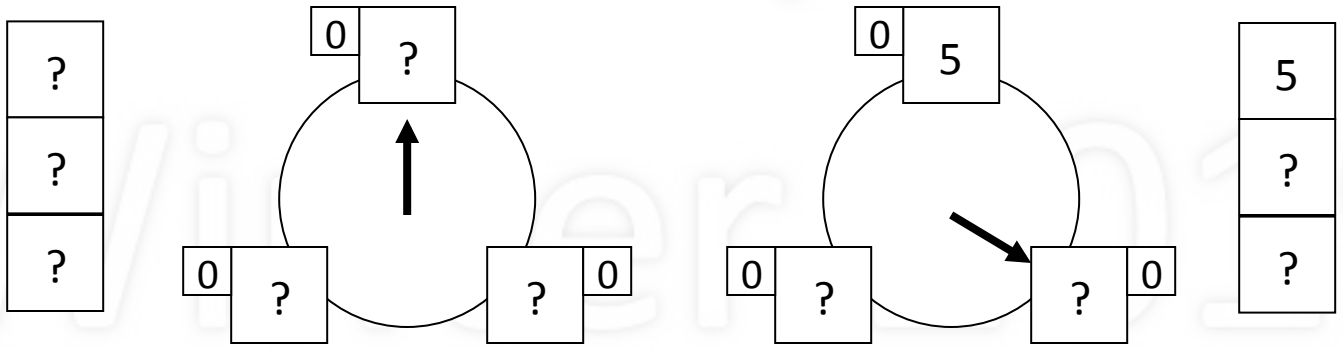
- Select page that is old and not recently used
  - Clock (second chance) is approximation of LRU
- Hardware support: reference bit
  - Associated with each frame is a reference bit
  - Actually, reference bit is in page table entry
- How reference bit is used
  - When frame filled with page, set bit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

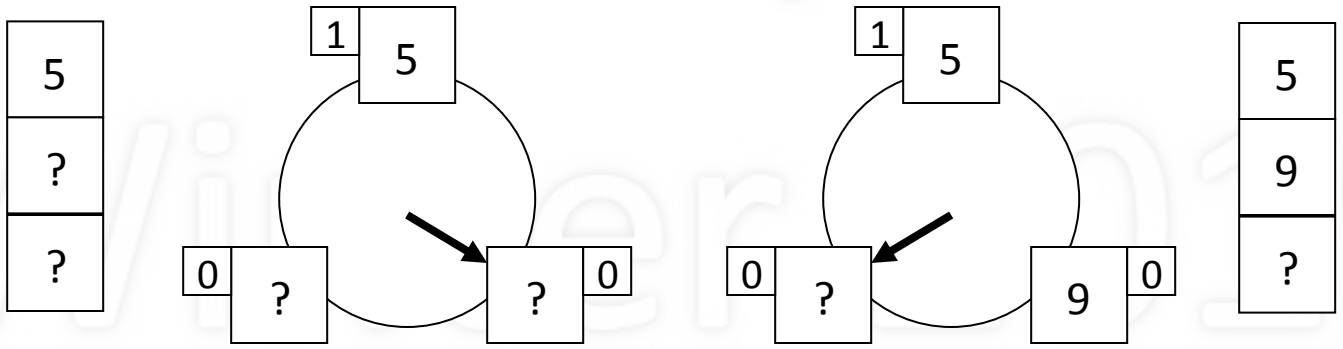


Example of Clock



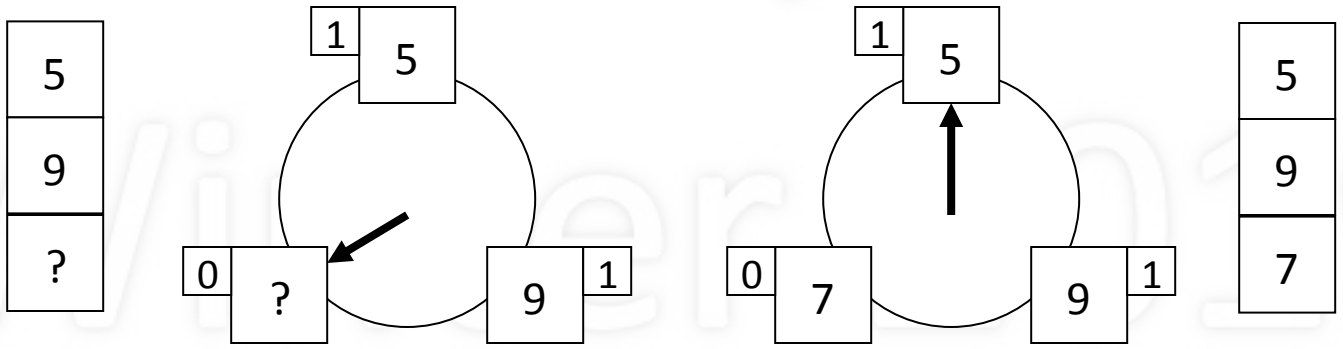
- Ref string: 5 9 7 1 9 5 9
- Reference page 5: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



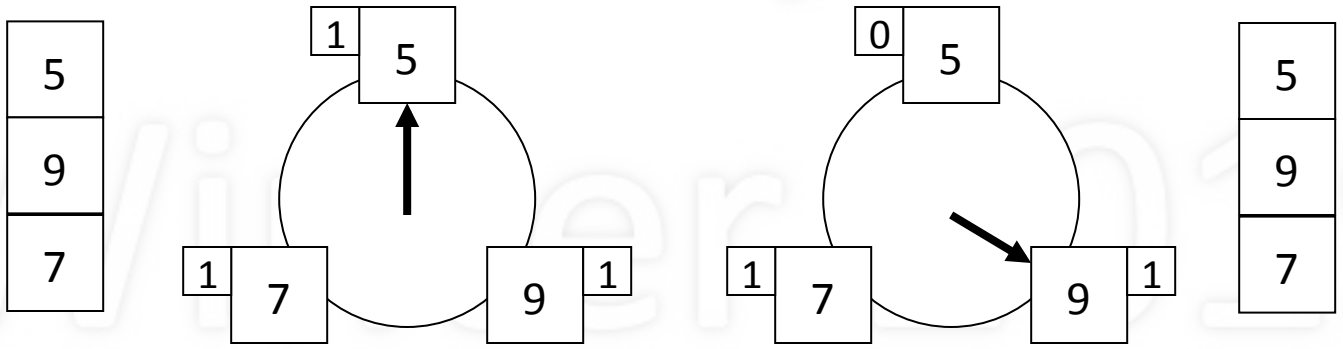
- Ref string: 5 9 7 1 9 5 9
- Reference page 9: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



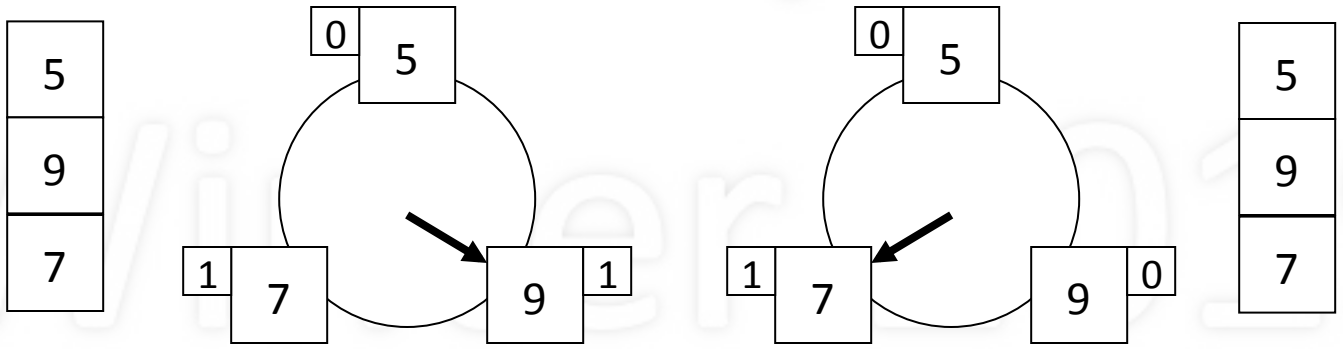
- Ref string: 5 9 7 1 9 5 9
- Reference page 7: page fault (unavoidable)
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



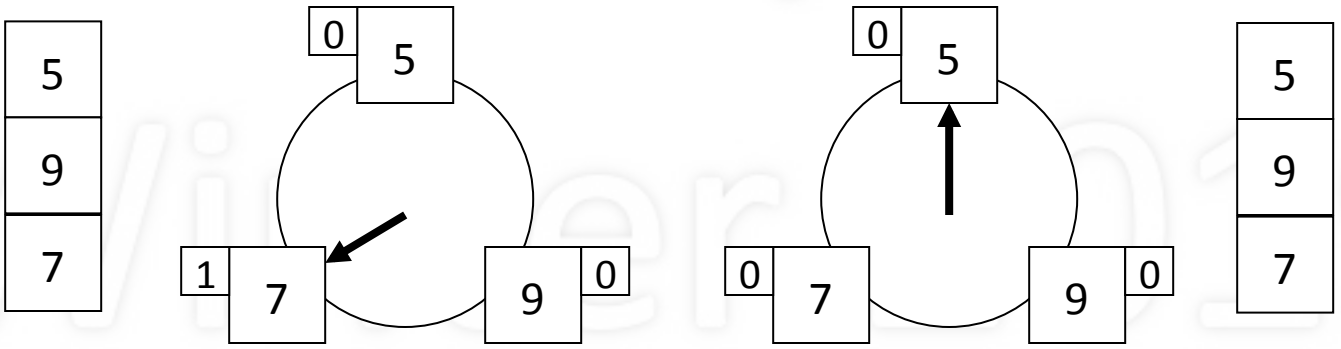
- Ref string: 5 9 7 1 9 5 9
- Reference page 1: page fault (1)
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



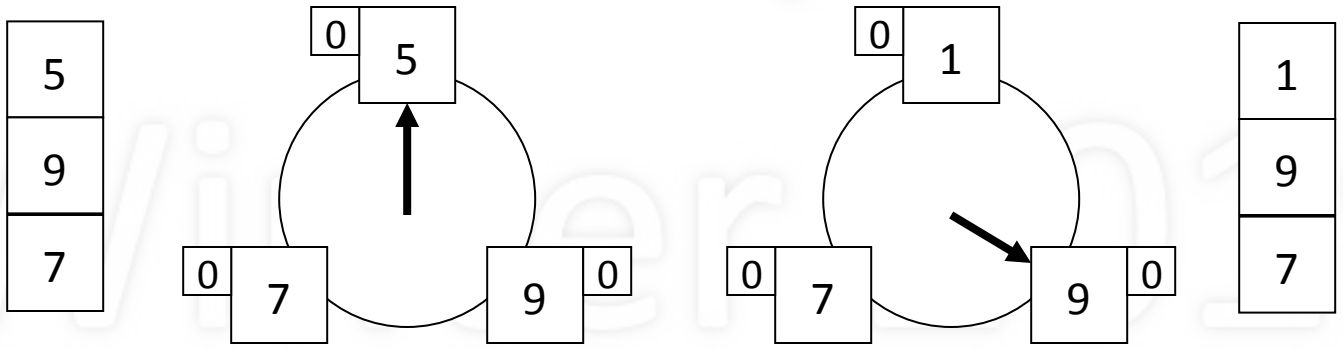
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



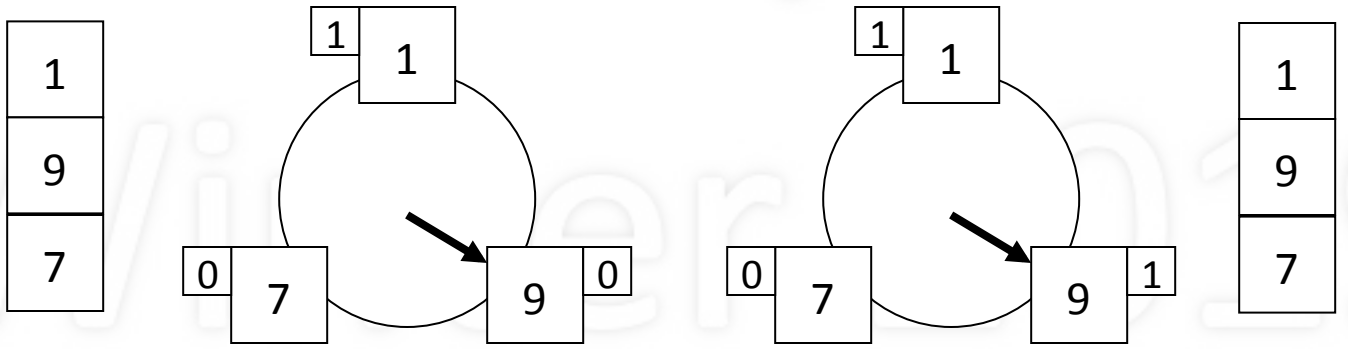
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

Example of Clock



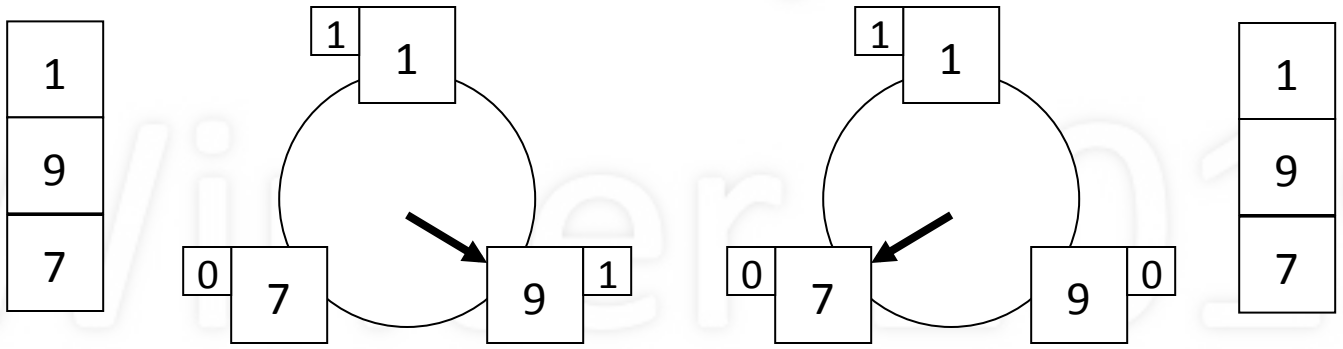
- Ref string: 5 9 7 1 9 5 9
- Trying to find unreferenced page
  - Hand points to an unreferenced page: use it
  - Advance hand

Example of Clock



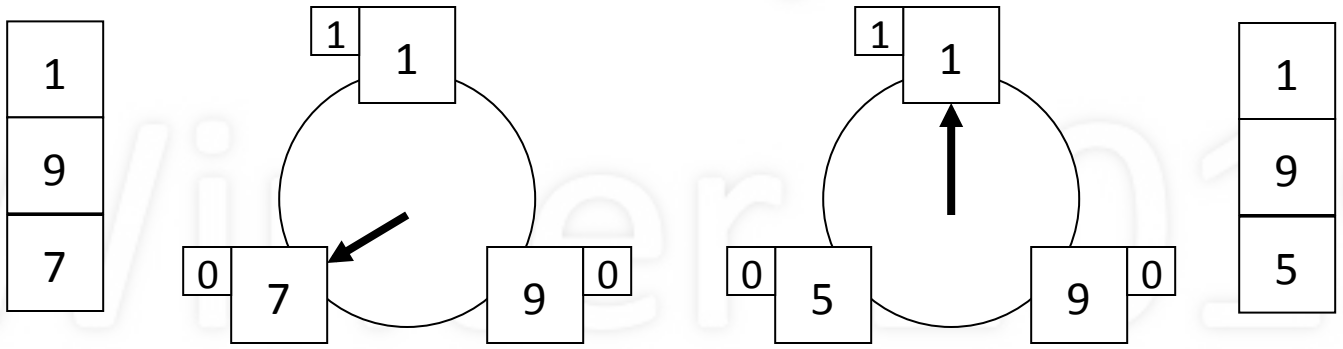
- Ref string: 5 9 7 1 9 5 9
- Reference page 9
  - Page 9 is already in memory: no page fault
  - OS does nothing, but *hardware* sets ref bit to 1

Example of Clock



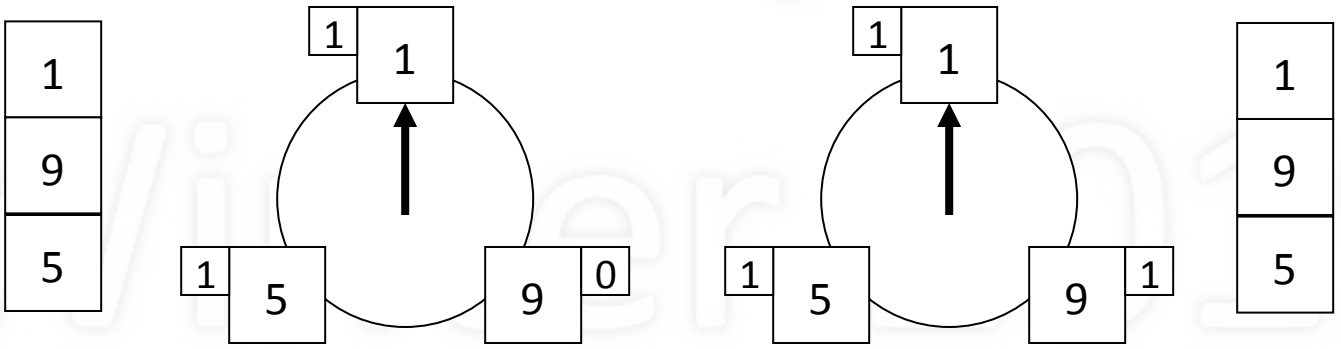
- Ref string: 5 9 7 1 9 5 9
- Reference page 5: page fault (2)
  - Hand points to a referenced page: skip it
  - Set ref bit to 0, advance hand, try again

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

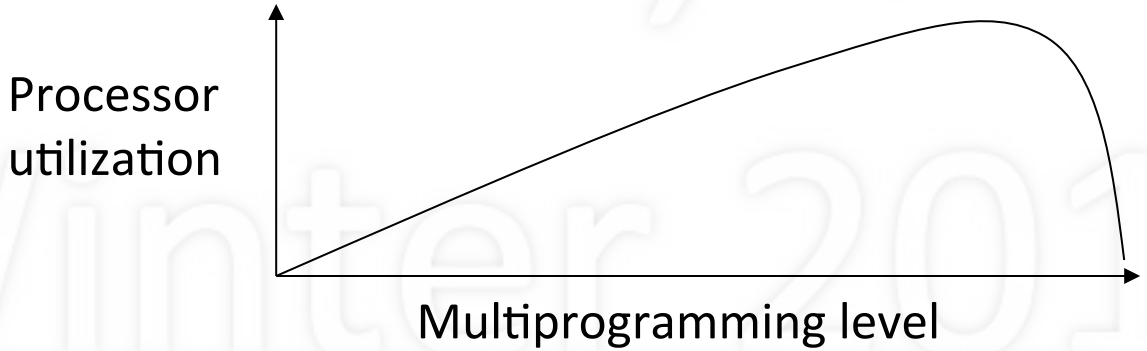


- Ref string: 5 9 7 1 9 5 9
- Reference page 9
  - Page 9 already in memory: no page fault
  - OS does nothing, but hardware sets ref bit to 1

Resident Set Management

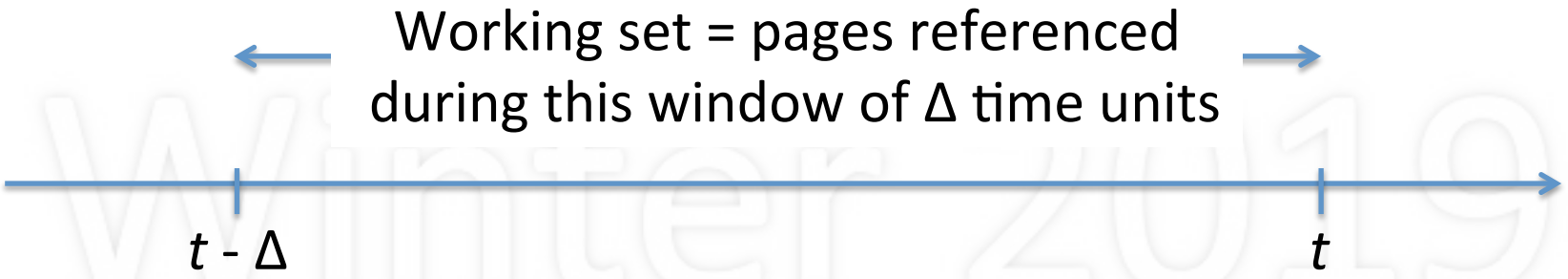
- Resident set: process's pages in physical memory
  - How big should resident set be? Which pages?
  - Who provides frame (same process or another)?
- Local: limit frame selection to requesting process
  - Isolates effects of page behavior on processes
  - Inefficient: some processes have unused frames
- 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

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

Review & Research

- How does memory structured as segments and pages allow partitioning of memory for convenient allocation?**
- How does memory structured as segments and pages allow reorganizing of memory for convenient usage?**
- What is meant by “convenient” in the above questions?***

R&R

- What is meant by a “logically” organized memory – how is the word “logical” being used?***
- One of the implications of a logically organized memory based on segments and pages is that no all the pieces (segments or pages) need to be in memory: precisely what is it that allows us to make this implication?***

R&R

- If not all pieces are in memory, where are they located (i.e., those not in memory)?*
- What are the factors that support the illusion that there is much more memory, and why?**
- To create this illusion, what mechanisms are needed?**

R&R

- How is a “logical memory” different from a “virtual memory”?*
- How does a virtual memory present the illusion of a large memory (and “large” relative to what)?**
- What is the difference between a “logical address space” and “virtual address space”?*

R&R

- In slide 5, why is there a sequential one-to-one mapping between virtual memory pages and page table entries?**
- Why is there NOT a sequential one-to-one mapping between page table entries and physical memory frames?**
- Are all the pages of virtual memory in frames of physical memory?* If not, where are they?

R&R

- How do we keep track of how to find pages of virtual memory that are in physical memory?*
- How do we keep track of how to find pages of virtual memory that are not in physical memory?***

R&R

- Slide 6 shows sample contents of a page table entry: can you explain the meaning of each field?*
- If the valid bit is off, do the other fields contain useful data?*
- If not, can you think of a use for the other fields (if the valid bit is off)?***

R&R

- What is a page fault?
- At what point during address translation is it determined that a page fault is to occur?
- Is a page fault determined by software (e.g., the kernel) or hardware?*
- What happens in hardware when a page fault is detected?** ... what happens in software? ***

R&R

- On slide 7, what does “Retry instruction” mean?**
- What are the fields of a virtual address for a virtual memory that is segmented and paged?
- What is a segment fault, and how is it different from a segmentation violation?**
- How is a segment fault different from a page fault?*

R&R

- Why are page faults expensive?
- How does the expense of page faults compare to the expense of TLB misses?**
- What is the point of the example on slide 9?*
- Can you think of any relationship between the time to resolve a page fault and the quantum time?***

R&R

- What is the Principle of Locality (of reference)?*
- What is meant by “thrashing”?*
- Why should we expect programs to exhibit locality?***
- What is meant by “page replacement”?
- Why is page replacement considered policy?*

R&R

- How does the FIFO page replacement policy work? ... OPT? ... LRU? What are the pros/ cons of each?*
- What is the key metric of performance for a page replacement policy?**
- Why doesn't FIFO generally perform well?*
- Why does OPT perform as best as possible?**
- Can FIFO ever perform as well as OPT?*

R&R

- Under what conditions will LRU perform well?
- Under what conditions will LRU not perform well?*
- What makes LRU so costly to implement?**
- What is a “reference string”?
- Do you expect the references in a small (say, 10-100 references) sequence in a reference string to be all the same: why or why not?***

R&R

- On slide 15, why is there an asterisk next to the 2?
- On slide 16, why is there an asterisk next to the 3, and not the 2?*
- On slides 16 and 17, why is the arrow pointing to the lowest row?*
- On slide 18, where did the 1 with the asterisk come from, and why did the arrow move?*

R&R

- Can you explain the rest of the FIFO example, in slides 19-26?*
- On slide 27, what is meant by “FIFO incurs 9 page faults (5 are obligatory)”?*
- Why is removing the oldest page not generally the best?*

R&R

- On slide 29, why are the first 4 columns filled without showing intervening steps?**
- On slide 30, why is the arrow pointing at the second row?*
- On slide 31, why is the arrow pointing at the first row, and why is there no asterisk in the sixth column?*
- Can you explain slides 32-37?*

R&R

- On slide 38, how is it known that 6 page faults is 1 beyond obligatory?*
- Why is OPT not realistic?
- If it is not realistic, why study it?*
- On slide 39, what clearly indicates that OPT does better than FIFO?*
- Is OPT always better than FIFO?*

R&R

- On slide 42, explain why there is a 2, 5*, and 1 in the rightmost column in the lower matrix, and why does the arrow point to the upper row?**
- On slide 43, why did the arrow move?
- On slide 44, explain the 2, 5, and 4* in the rightmost column in the lower matrix, and why does the arrow point to the middle row? **

R&R

- Can you explain the rest of the LRU example in slide 45-49?*
- In slide 50, can you explain why we know 7 page faults is 2 beyond obligatory?*
- Was there any locality in this example?**
- Why does LRU require hardware support, and what does this support correspond to in the example?***

R&R

- In slide 51, can you explain and justify the inequality "OPT ≥ LRU (assuming locality) ≳ FIFO"?***
- What is the goal of the Clock Algorithm?*
- What is the difference between "least recently used" and "not recently used"?**
- What is the purpose of the reference bit?

R&R

- The function of the reference bit in the Clock algorithm is most similar to what in LRU?***
- In the Clock algorithm, what happens if a memory reference results in a page fault?
- What happens if a memory reference does not result in a page fault?*
- What is the purpose of the “hand” in the Clock algorithm?*

R&R

- If a frame has been modified, why must it be written to the disk?*
- In slide 54, an you explain the contents of each of the boxes (including, why do all the little boxes have 0's), and why the arrow moved in the left and right diagrams?**
- In slide 55, why are the reference bits set to 1 for page 5, and to 0 for page 9?*

R&R

- In slide 56, why is the 3rd frame filled with page 7?
- In slide 57, which frame is eligible to hold page 1?*
- Can you explain slides 58-59?
- In slide 60, why is page 1 placed in the 1st frame, and why are all the reference bits set to 0?*

R&R

- In slide 61, what caused the reference bit for page 1 to be set to 1, and why was this done by the hardware rather than the kernel?*
- Can you explain slides 62-63?
- In slide 62, why is the hand pointing to page 1 while page 9's reference bit was set from 0 to 1?

R&R

- What is the resident set?*
- What's the difference between local replacement and global replacement policies, and what are the pros/cons of each?**
- What is meant by isolation vs. efficiency?*
- What is the Multiprogramming Level?*
- What does the graph on slide 66 show?**
- At what point does thrashing occur and why?*

R&R

- What is the working set, and why should the resident set contain it?**
- What should be done if the working set cannot be kept in memory in its entirety, and why?**
- Why is the working set a local replacement policy, and what is the significance of this?*

R&R

- What does it mean to time-stamp pages?**
- Why must pages be time-stamped in the working set?**
- How should the working set parameter Δ be determined?***
- How does the working set compare to Clock: what are the pros/cons?**