CS 4410
Operating Systems

Page Replacement (2)

Summer 2016

Cornell University

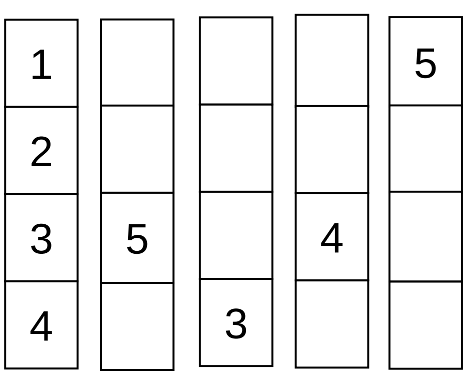
Today

- Algorithm that approximates the OPT replacement algorithm.

Least Recently Used (LRU) Page Replacement

- A recently used page is likely to be used again in the future.
- Replace the page that has not been used for the longest period of time.
- Use the recent past as an approximation of the near future.

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

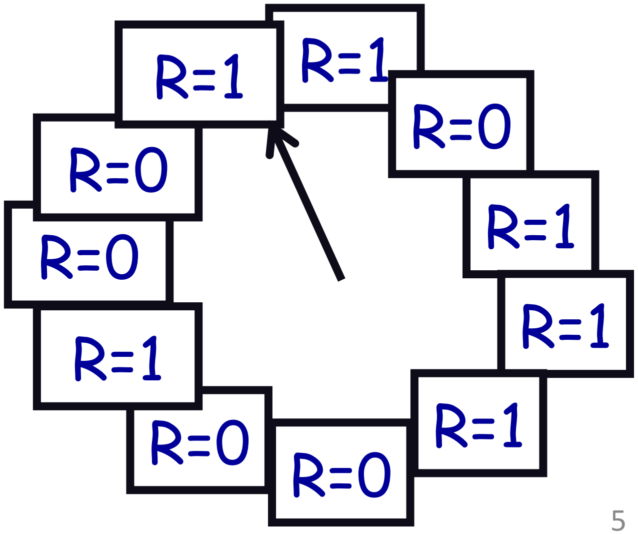


Distinguish recently used from not recently used pages

- Counters
  - Each page-table entry is associated with a time-of-use field.
  - Add to the CPU a logical clock.
    - The clock is incremented at every memory access.
  - At every memory access, the field of the referenced page is updated with the clock.
  - Scan the page table to find the LRU page.
- Stack
  - Whenever a page is referenced, it is removed from the stack and put on the top.
  - The LRU page is always at the bottom.
  - The update is expensive.

LRU: Clock Algorithm

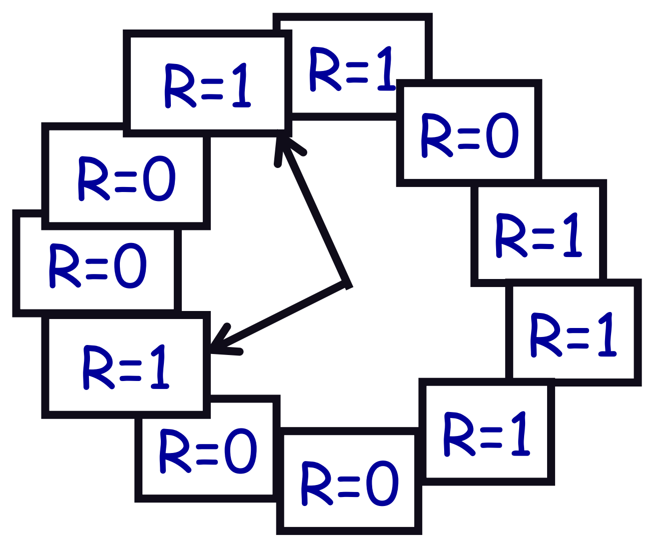
- Each page entry is associated with a reference bit.
  - Set on use, reset periodically by the OS.
- Algorithm:
  - Scan: if ref bit is 1, set to 0, and proceed. If ref bit is 0, stop and evict.
- Problem:
  - Low accuracy for large memory



LRU: Clock Algorithm

- Solution: Add another hand
  - Leading edge clears ref bits
  - Trailing edge evicts pages with ref bit 0

- What if angle small?
- What if angle big?



Global vs Local Allocation

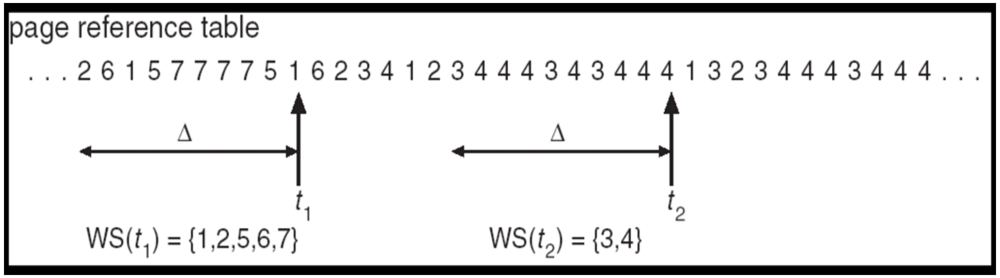
- Global replacement
  - Single memory pool for entire system.
  - On page fault, evict oldest page in the system.
  - Problem: lack of performance isolation.
- Local (per-process) replacement
  - Have a separate pool of pages for each process.
  - Page fault in one process can only replace pages from its own process.
  - Problem: might have idle resources.

Thrashing

- Excessive rate of page replacement
  - Keep throwing out page that will be referenced soon.
  - Keep referencing pages that are not in memory.
- Why does it occur?
  - Too many processes in the system.
- How can we solve this problem?
  - *Locality* model of process execution.
    - A locality is a set of pages that are actively used together.

Working Set

- Estimate locality → Identify useful pages → Do not evict these pages, because they are likely to be referenced again.
- Working Set = An approximation of the program's locality.
  - The set of pages in the most recent Δ page references.
  - Δ: working-set window
- As a process executes, it moves from locality to locality.
- Example (Δ = 10):
  - t1 → WS = {1,2,5,6,7}
  - t2 → WS = {3,4}



- If allocated frames do not accommodate current locality, the process will thrash.

Computing the working set

- Working set = sets of pages in the working set window.
- Difficulty: the working set window is a moving window. At each memory reference:
  - a new reference appears at one end -> the corresponding page should be marked as a member of the working set.
  - The oldest reference drops off the other end -> the corresponding page should be unmarked.

Computing the working set

… 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 … Δ=10

Δ

t1 t2 t3 t4 t5

ti	WS
t1	{1,2,5,6,7}
t2	{1,5,6,7}
t3	{1,2,5,6,7}
t4	{1,2,3,5,6,7}
t5	{1,2,3,4,5,6,7}

How can we compute WS

- without recording the reference history, but
- with specialized bits associated to page table entries?
  - These bits signify whether a page belongs to the WS.

Computing the working set

- Each page table entry is associated with 1 reference bit and Δ WS-bits.
- At every page reference:
  - Set the corresponding reference bit to 1.
  - Update the working set:
    - Shift WS-bits one bit to the right.
    - Put reference bit to the most significant WS-bit.
  - Reset reference bits to 0.
- If some WS-bits of a page are set to 1, then the page belongs to the WS.
- If all WS-bits of a page are 0, then the page does not belong to WS.
  - This page can be evicted.

Computing the working set

… 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 … $\Delta=10$

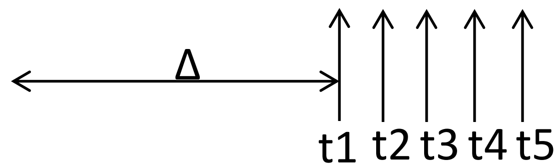
Δ

t1 t2 t3 t4 t5

Page number	WS-bits after access ti				
	t1	t2	t3	t4	t5
1					
2					
3					
4					
5					
6					
7					
WS					

Computing the working set

… 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 … Δ=10



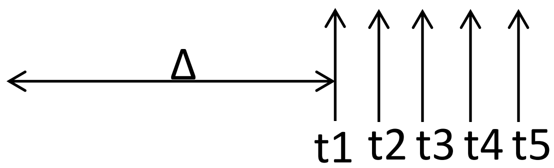
Page number	WS-bits after access ti				
	t1	t2	t3	t4	t5
1	**1**000000100	0100000010	0010000001	0001000000	0000100000
2	0000000001	0000000000	**1**000000000	0100000000	0010000000
3	0000000000	0000000000	0000000000	**1**000000000	0100000000
4	0000000000	0000000000	0000000000	0000000000	**1**000000000
5	0100001000	0010000100	0001000010	0000100001	0000010000
6	0000000010	**1**000000001	0100000000	0010000000	0001000000
7	0011110000	0001111000	0000111100	0000011110	0000001111
WS	{1,2,5,6,7}	{1,5,6,7}	{1,2,5,6,7}	{1,2,3,5,6,7}	{1,2,3,4,5,6,7}

Working Set Approximation

- It is expensive to update the WS at every page reference.
- We can approximate the WS by updating it after Δ/n references.
- Need n WS-bits per page table entry.
- After Δ/n references there will be an interrupt.
- At every page reference:
    - Set the corresponding reference bit to 1.
- At every interrupt:
    - Update the working set:
        - Shift WS-bits one bit to the right.
        - Put reference bit to the most significant WS-bit of each page.
    - Reset reference bits to 0.

Working Set Approximation

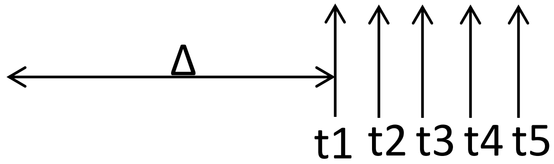
… 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 … Δ=10, n=5



Page number	WS-bits after access ti		
	t1	t3	t5
1			
2			
3			
4			
5			
6			
7			
WS			

Working Set Approximation

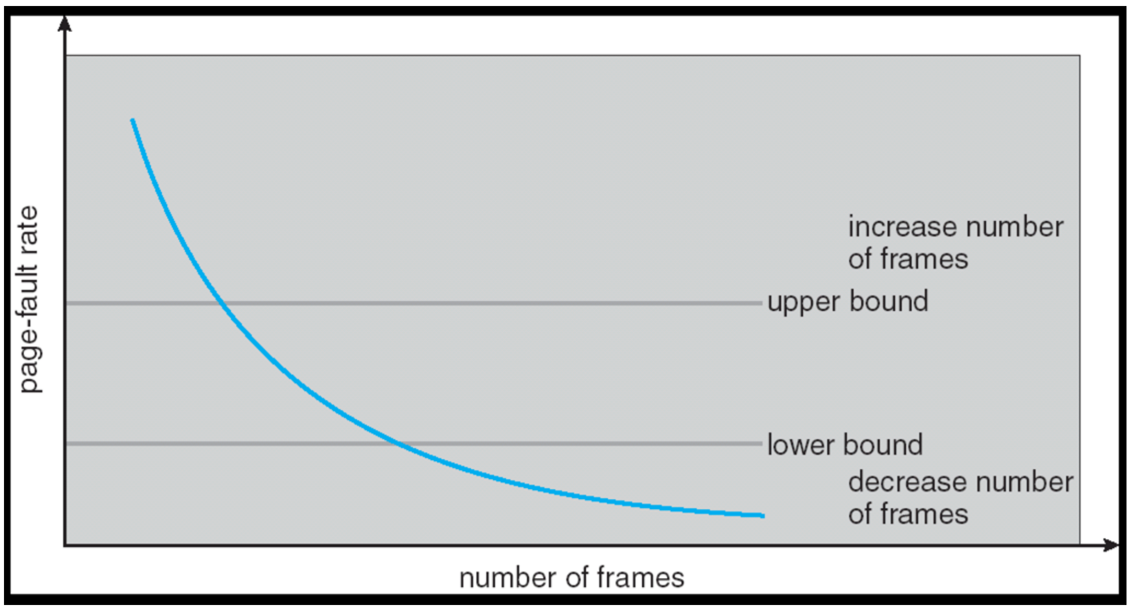
… 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 … Δ=10, n=5



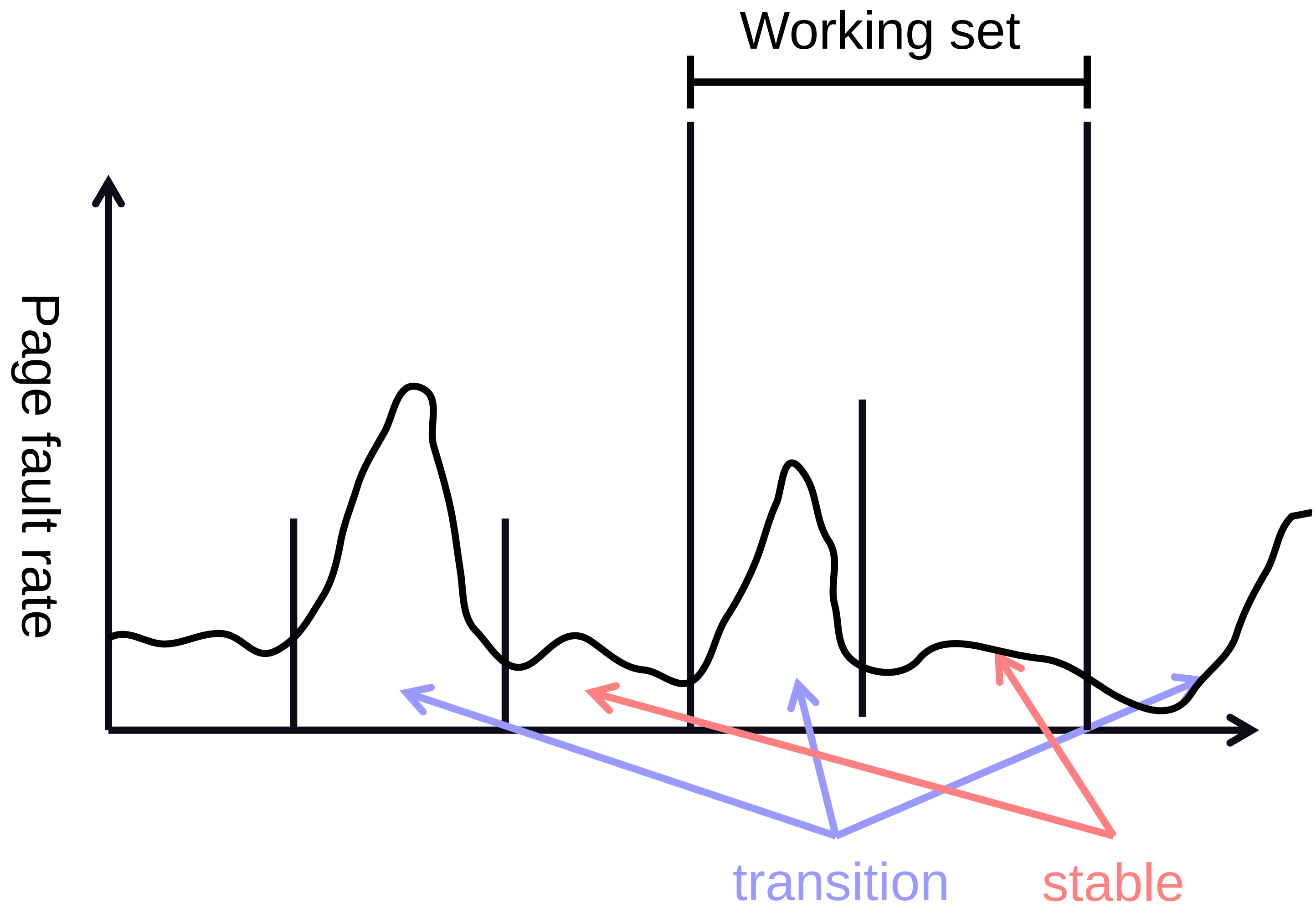
Page number	WS-bits after access ti		
	t1	t3	t5
1	**1**0010	01001	00100
2	00001	**1**0000	01000
3	00000	00000	**1**0000
4	00000	00000	**1**0000
5	10010	01001	00100
6	00001	**1**0001	01000
7	01100	00110	00011
WS	{1,2,5,6,7}	{1,2,5,6,7}	{1,2,3,4,5,6,7}

Page Fault Frequency

- PFF = page faults / instructions executed.
- If PFF rises above threshold, process needs more memory.
  - Not enough memory on the system? → Swap out.
- If PFF sinks below threshold, memory can be taken away.



Working Sets and Page Fault Rates



Today

- Algorithm that approximates the OPT replacement algorithm.

Coming up…

- Next lecture: Review
- HW3: due today
- Exam2: Wednesday, last 30mis of class