CS 318 Principles of Operating Systems

Fall 2017

Lecture 11: Page Replacement

Ryan Huang

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Memory Management

Final lecture on memory management:

- **Goals of memory management**
  - To provide a convenient abstraction for programming
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead
- **Mechanisms**
  - Physical and virtual addressing (1)
  - Techniques: Partitioning, paging, segmentation (1)
  - Page table management, TLBs, VM tricks (2)
- **Policies**
  - Page replacement algorithms (3)

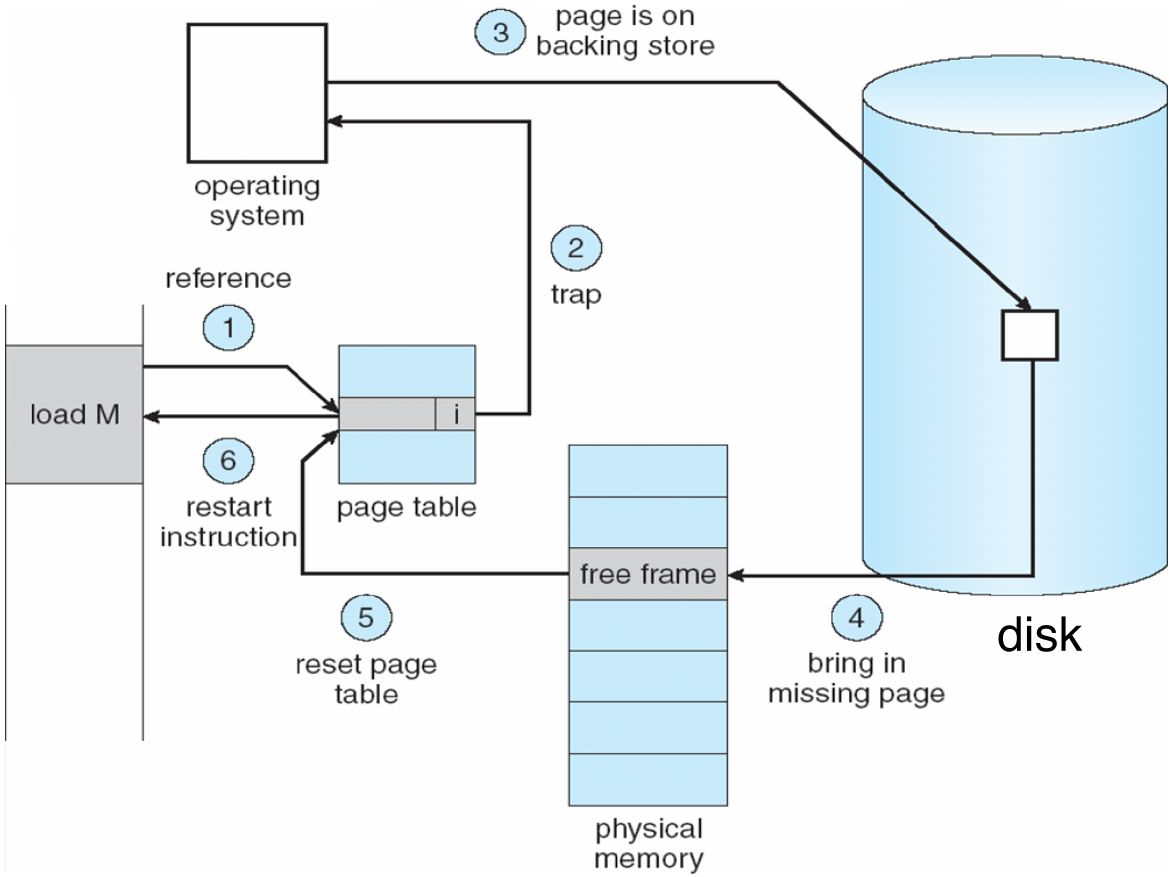
Lecture Overview

- **Review paging and page replacement**
- **Survey page replacement algorithms**
- **Discuss local vs. global replacement**
- **Discuss thrashing**

Review: Paging

- **Recall paging from the OS perspective:**
  - Pages are evicted to disk when memory is full
  - Pages loaded from disk when referenced again
  - References to evicted pages cause a TLB miss
    - PTE was invalid, causes fault
  - OS allocates a page frame, reads page from disk
  - When I/O completes, the OS fills in PTE, marks it valid, and restarts faulting process
- **Dirty vs. clean pages**
  - Actually, only dirty pages (modified) need to be written to disk
  - Clean pages do not – but you need to know where on disk to read them from again

Review: Paging

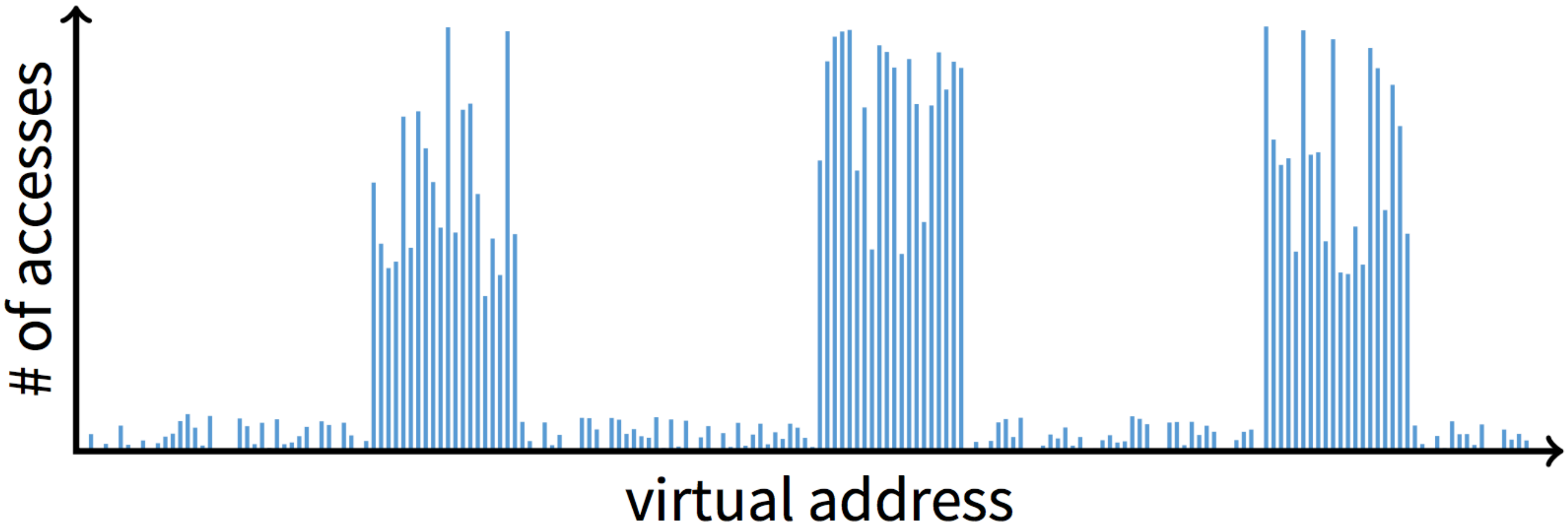


- **Use disk to simulate larger virtual than physical mem**

Locality

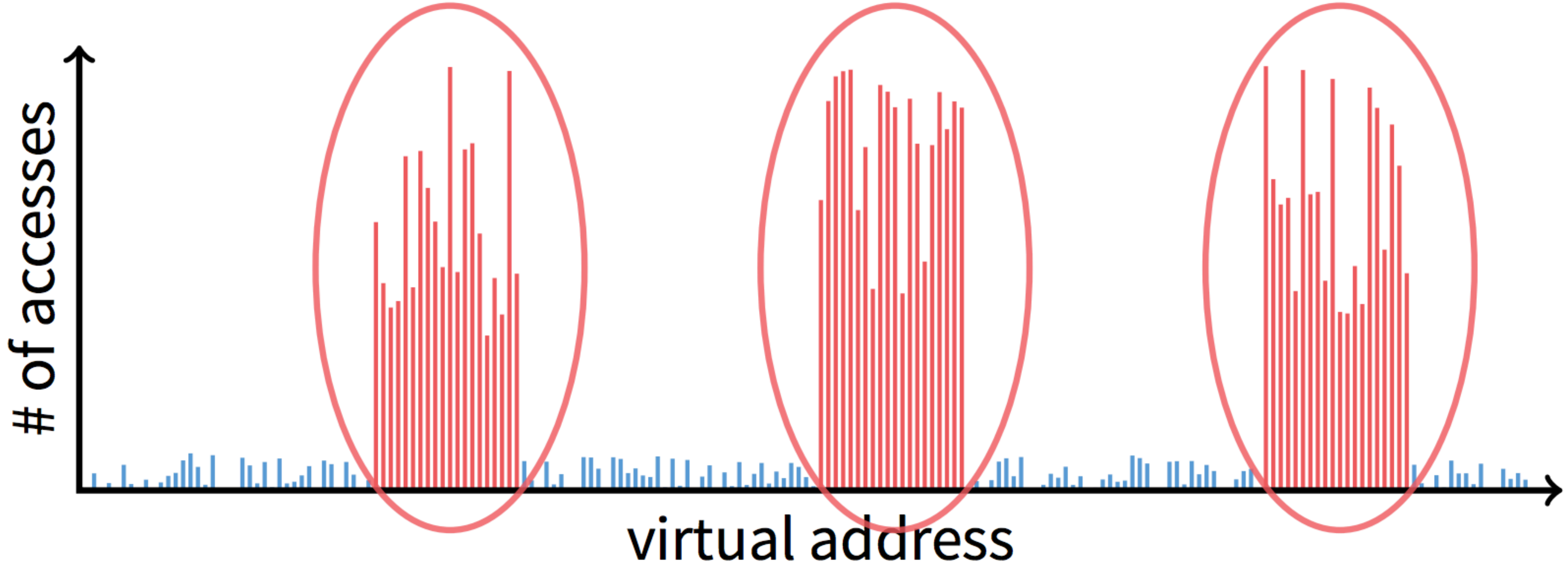
- **All paging schemes depend on locality**
  - Processes reference pages in localized patterns
- **Temporal locality**
  - Locations referenced recently likely to be referenced again
- **Spatial locality**
  - Locations near recently referenced locations are likely to be referenced soon
- **Although the cost of paging is high, if it is infrequent enough it is acceptable**
  - Processes usually exhibit both kinds of locality during their execution, making paging practical

Working Set Model (more later)



- **Disk much, much slower than memory**
  - Goal: run at memory speed, not disk speed
- **80/20 rule: 20% of memory gets 80% of memory accesses**
  - Keep the hot 20% in memory
  - Keep the cold 80% on disk

Working Set Model (more later)



- **Disk much, much slower than memory**
  - Goal: run at memory speed, not disk speed
- **80/20 rule: 20% of memory gets 80% of memory accesses**
  - Keep the hot 20% in memory
  - Keep the cold 80% on disk

Page Replacement

- **When a page fault occurs, the OS loads the faulted page from disk into a page frame of physical memory**
- **At some point, the process used all of the page frames it is allowed to use**
  - This is likely (much) less than all of available memory
- **When this happens, the OS must replace a page for each page faulted in**
  - It must evict a page to free up a page frame
- **The page replacement algorithm determines how this is done**
  - Greatly affect performance of paging (virtual memory)
  - Also called page eviction policies

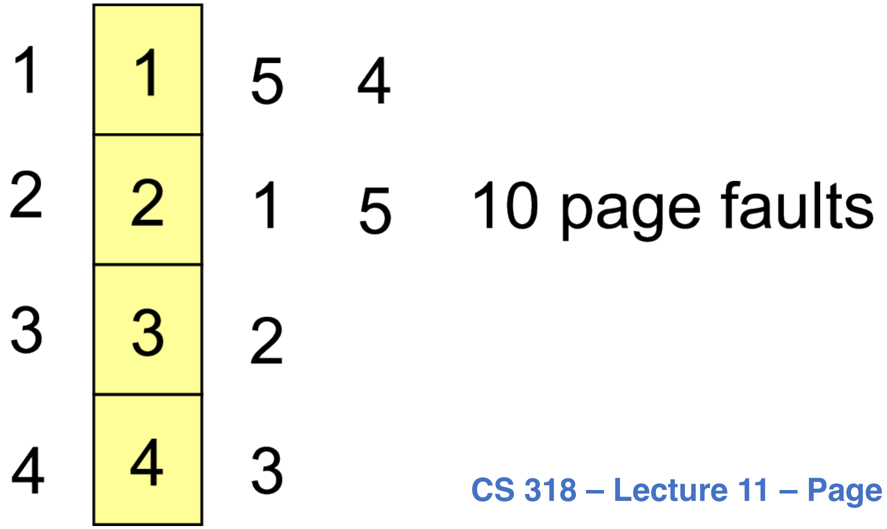
First-In First-Out (FIFO)

- **Evict oldest fetched page in system**
- **Example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **3 physical pages: 9 page faults**

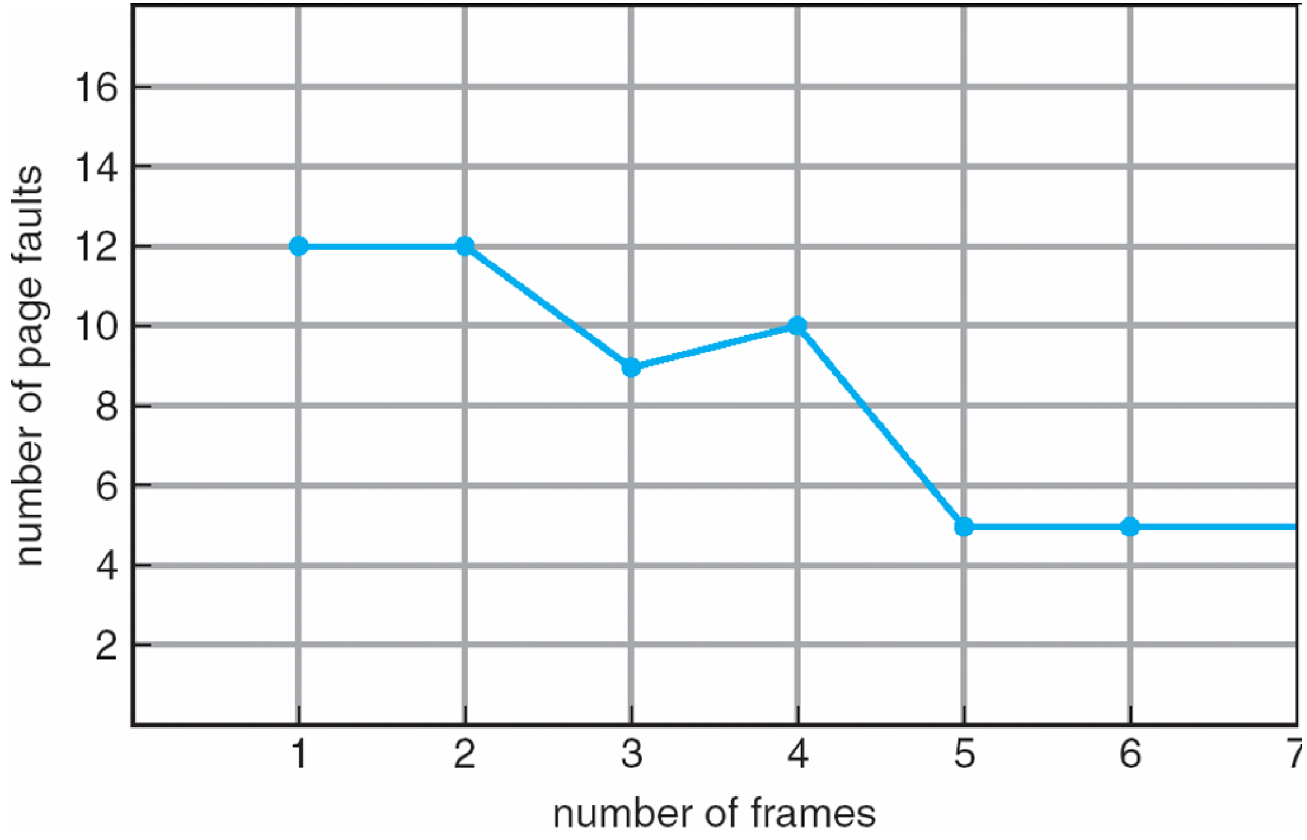
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

First-In First-Out (FIFO)

- **Evict oldest fetched page in system**
- **Example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **3 physical pages: 9 page faults**
- **4 physical pages: 10 page faults**



Belady's Anomaly



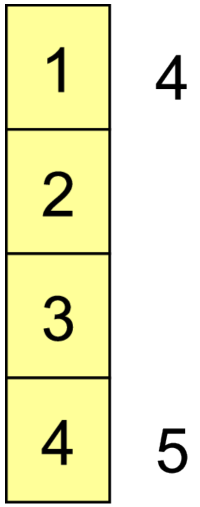
- **More physical memory doesn't always mean fewer faults**

Optimal Page Replacement

- **What is optimal (if you knew the future)?**

Optimal Page Replacement

- **What is optimal (if you knew the future)?**
  - Replace page that will not be used for longest period of time
- **Example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **With 4 physical pages:**



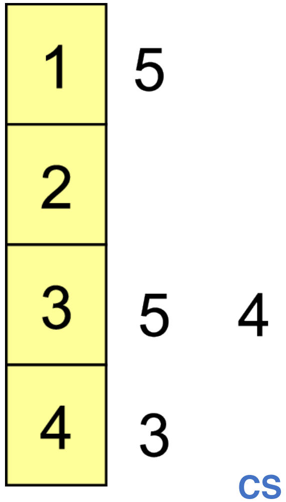
6 page faults

Belady's Algorithm

- **Known as the optimal page replacement algorithm**
  - Rationale: the best page to evict is the one never touched again
  - Never is a long time, so picking the page closest to "never" is the next best thing
  - Proved by Belady
- **Problem: Have to predict the future**
- **Why is Belady's useful then? Use it as a yardstick**
  - Compare implementations of page replacement algorithms with the optimal to gauge room for improvement
  - If optimal is not much better, then algorithm is pretty good
  - If optimal is much better, then algorithm could use some work
    - Random replacement is often the lower bound

Least Recently Used (LRU)

- **Approximate optimal with least recently used**
  - Because past often predicts the future
  - On replacement, evict the page that has not been used for the longest time in the past (Belady's: future)
- **Example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **With 4 physical pages: 8 page faults**



Least Recently Used (LRU)

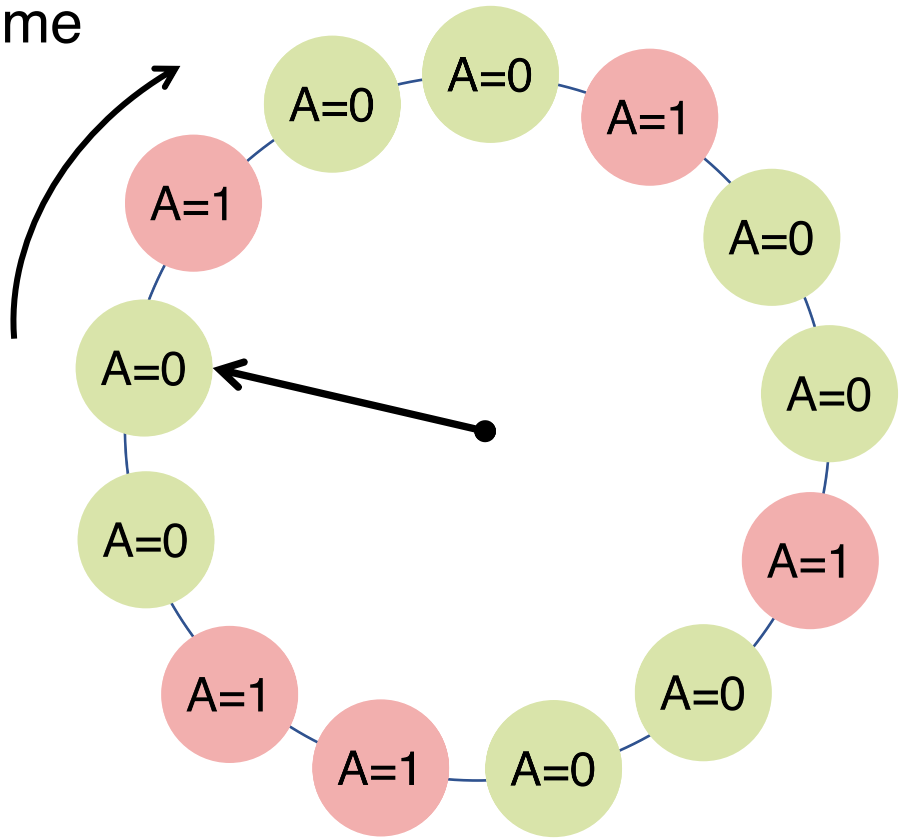
- **Approximate optimal with least recently used**
  - Because past often predicts the future
  - On replacement, evict the page that has not been used for the longest time in the past (Belady's: future)
- **Example—reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- **With 4 physical pages: 8 page faults**
- **Problem 1: Can be pessimal – example?**
  - Looping over memory (then want MRU eviction)
- **Problem 2: How to implement?**

Straw Man LRU Implementations

- **Stamp PTEs with timer value**
  - E.g., CPU has cycle counter
  - Automatically writes value to PTE on each page access
  - Scan page table to find oldest counter value = LRU page
  - Problem: Would double memory traffic!
- **Keep doubly-linked list of pages**
  - On access remove page, place at tail of list
  - Problem: again, very expensive
- **What to do?**
  - Just approximate LRU, don't try to do it exactly

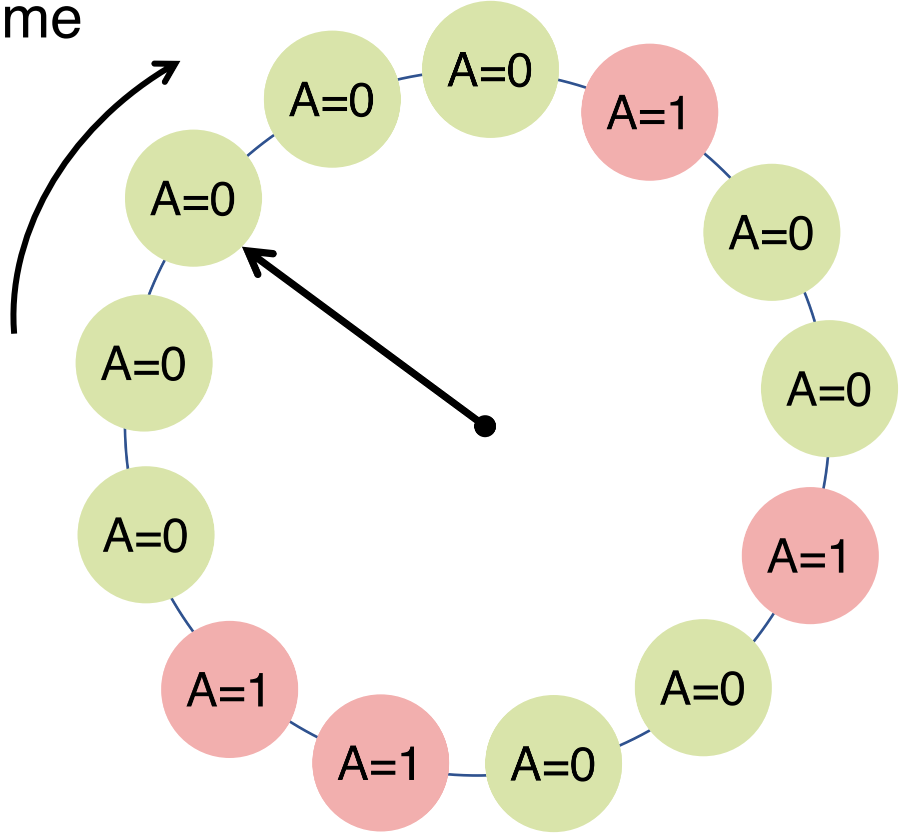
Clock Algorithm

- **Use accessed bit supported by most hardware**
  - E.g., Pentium will write 1 to A bit in PTE on first access
  - Software managed TLBs like MIPS can do the same
- **Do FIFO but skip accessed pages**
- **Keep pages in circular FIFO list**
- **Scan:**
  - page's A bit = 1, set to 0 & skip
  - else if A = 0, evict
- **A.k.a. second-chance replacement**



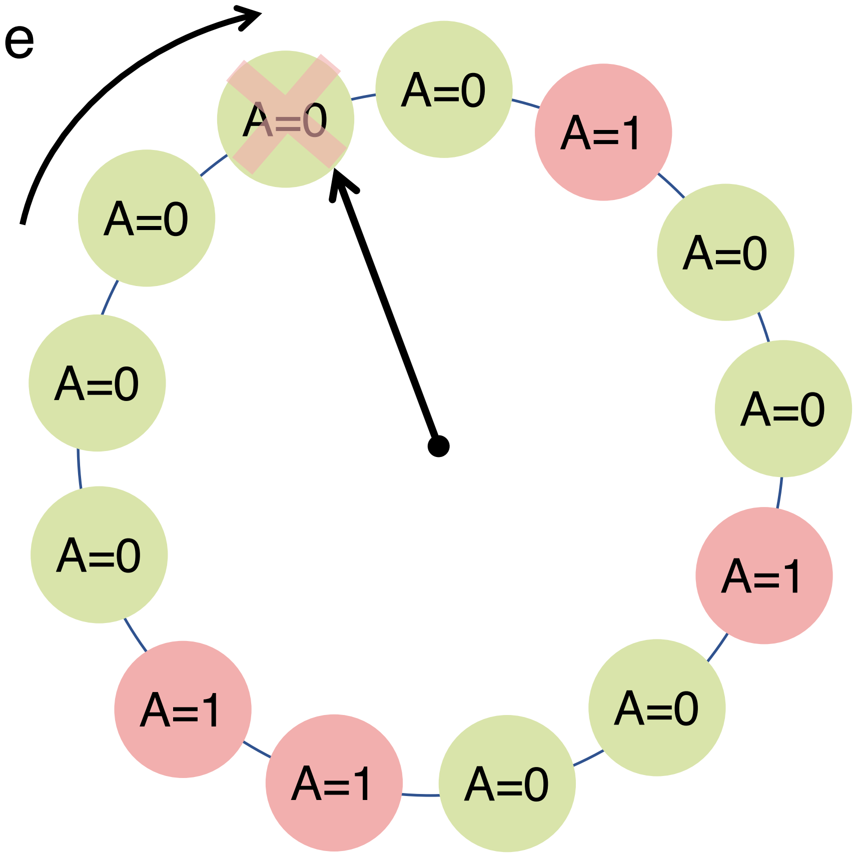
Clock Algorithm

- **Use accessed bit supported by most hardware**
  - E.g., Pentium will write 1 to A bit in PTE on first access
  - Software managed TLBs like MIPS can do the same
- **Do FIFO but skip accessed pages**
- **Keep pages in circular FIFO list**
- **Scan:**
  - page's A bit = 1, set to 0 & skip
  - else if A = 0, evict
- **A.k.a. second-chance replacement**



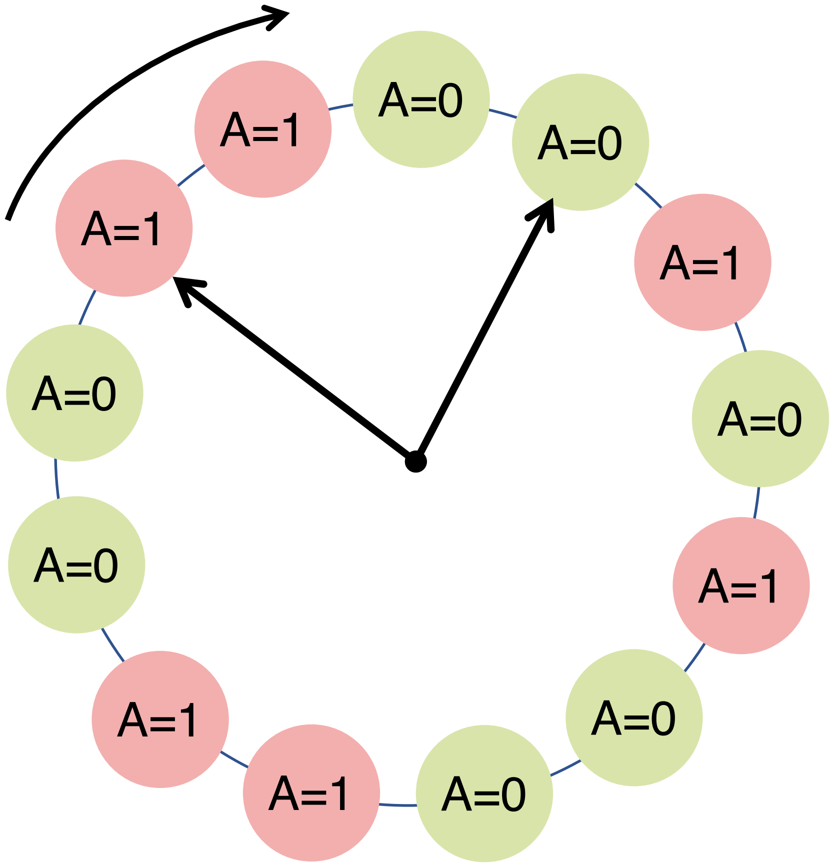
Clock Algorithm

- **Use accessed bit supported by most hardware**
  - E.g., Pentium will write 1 to A bit in PTE on first access
  - Software managed TLBs like MIPS can do the same
- **Do FIFO but skip accessed pages**
- **Keep physical pages in circular list**
- **Scan:**
  - page's A bit = 1, set to 0 & skip
  - else if A = 0, evict
- **A.k.a. second-chance replacement**



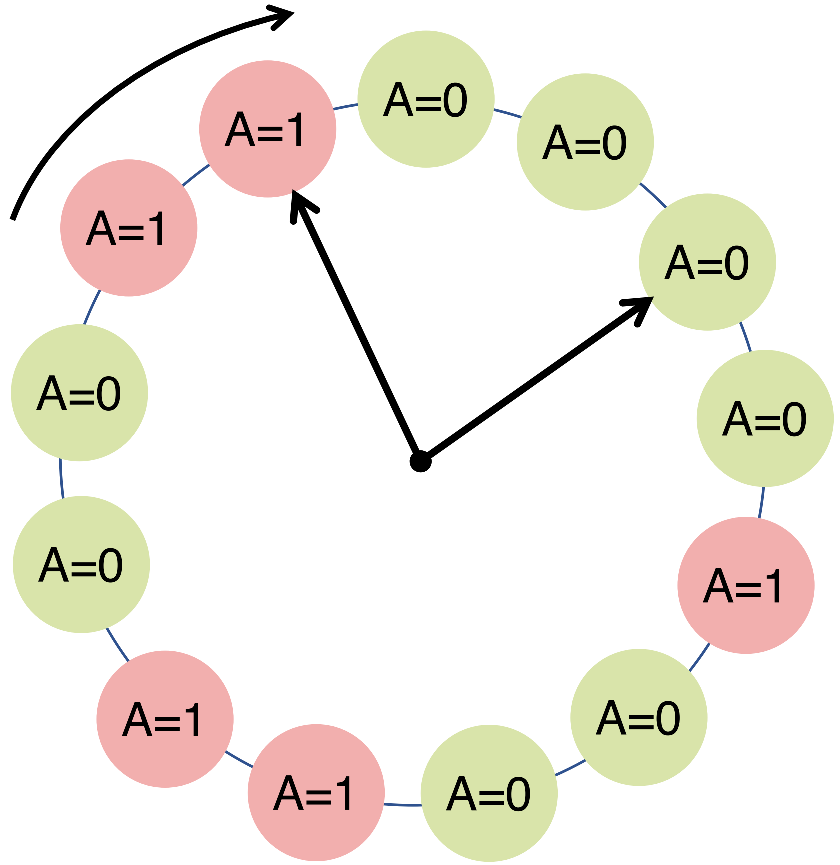
Clock Algorithm (continued)

- **Large memory may be a problem**
  - Most pages referenced in long interval
- **Add a second clock hand**
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0



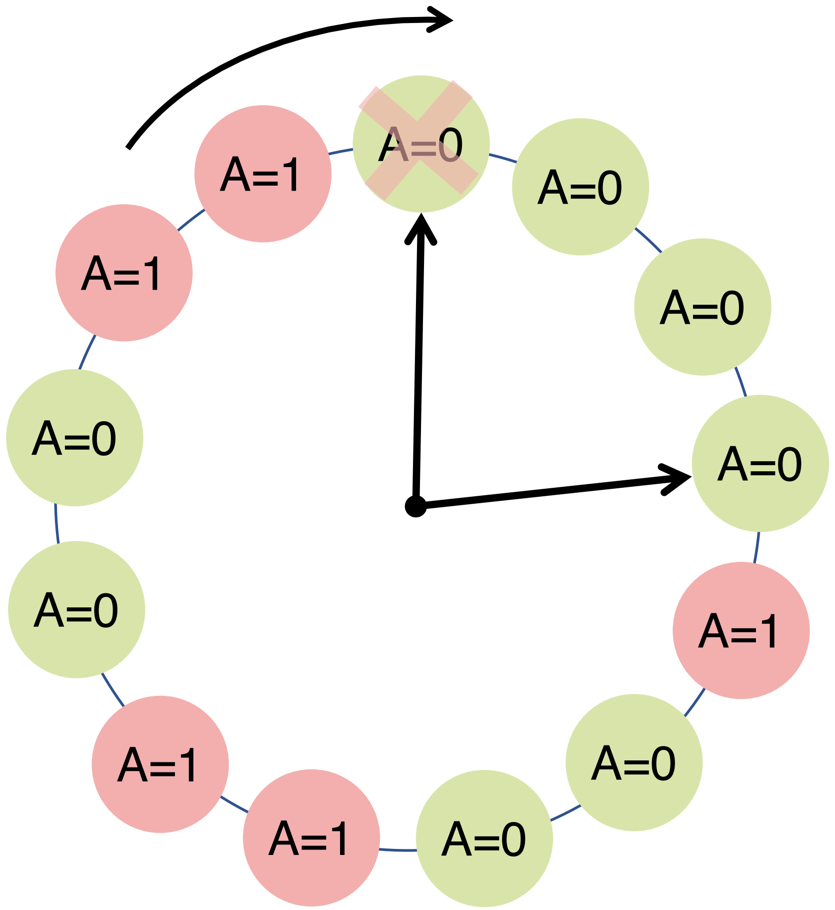
Clock Algorithm (continued)

- **Large memory may be a problem**
  - Most pages referenced in long interval
- **Add a second clock hand**
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0



Clock Algorithm (continued)

- **Large memory may be a problem**
  - Most pages referenced in long interval
- **Add a second clock hand**
  - Two hands move in lockstep
  - Leading hand clears A bits
  - Trailing hand evicts pages with A=0



Other Replacement Algorithms

- **Random eviction**
  - Dirt simple to implement
  - Not overly horrible (avoids Belady & pathological cases)
- **LFU (least frequently used) eviction**
  - Instead of just A bit, count # times each page accessed
  - Least frequently accessed must not be very useful (or maybe was just brought in and is about to be used)
  - Decay usage counts over time (for pages that fall out of usage)
- **MFU (most frequently used) algorithm**
  - Because page with the smallest count was probably just brought in and has yet to be used
- **Neither LFU nor MFU used very commonly**

Fixed vs. Variable Space

- **How to determine how much memory to give to each process?**
- **Fixed space algorithms**
  - Each process is given a limit of pages it can use
  - When it reaches the limit, it replaces from its own pages
  - Local replacement
    - Some processes may do well while others suffer
- **Variable space algorithms**
  - Process' set of pages grows and shrinks dynamically
  - Global replacement
    - One process can ruin it for the rest

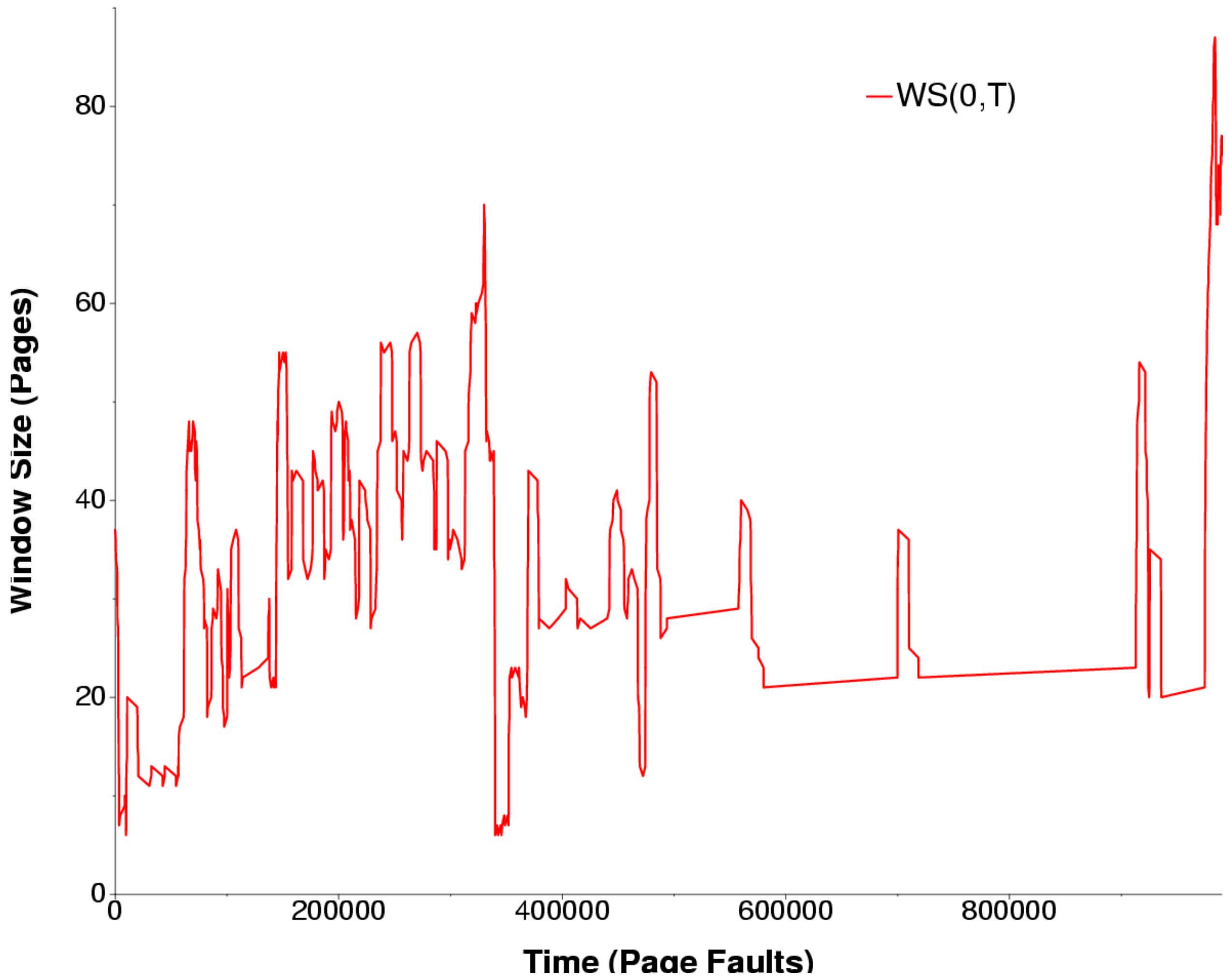
Working Set Model

- **A working set of a process is used to model the dynamic locality of its memory usage**
  - Defined by Peter Denning in 60s, published at the first SOSP conference
- **Definition**
  - WS(t,w) = {pages P such that P was referenced in the time interval (t, t-w)}
  - t – time, w – working set window (measured in page refs)
- **A page is in the working set (WS) only if it was referenced in the last w references**

Working Set Size

- **The working set size is the # of unique pages in the working set**
  - The number of pages referenced in the interval (t, t-w)
- **The working set size changes with program locality**
  - During periods of poor locality, you reference more pages
  - Within that period of time, the working set size is larger
- **Intuitively, want the working set to be the set of pages a process needs in memory to prevent heavy faulting**
  - Each process has a parameter w that determines a working set with few faults
  - Denning: Don't run a process unless working set is in memory

Example: gcc Working Set



Working Set Problems

- **Problems**
  - How do we determine w?
  - How do we know when the working set changes?
- **Too hard to answer**
  - So, working set is not used in practice as a page replacement algorithm
- **However, it is still used as an abstraction**
  - The intuition is still valid
  - When people ask, “How much memory does Firefox need?”, they are in effect asking for the size of Firefox’s working set

Page Fault Frequency (PFF)

- **Page Fault Frequency (PFF) is a variable space algorithm that uses a more ad-hoc approach**
  - Monitor the fault rate for each process
  - If the fault rate is above a high threshold, give it more memory
    - So that it faults less
    - But not always (FIFO, Belady's Anomaly)
  - If the fault rate is below a low threshold, take away memory
    - Should fault more
    - But not always
- **Hard to use PFF to distinguish between changes in locality and changes in size of working set**

Thrashing

- **Page replacement algorithms avoid thrashing**
  - When OS spent most of the time in paging data back and forth from disk
  - Little time spent doing useful work (making progress)
  - In this situation, the system is overcommitted
    - No idea which pages should be in memory to reduce faults
    - Could just be that there isn't enough physical memory for all of the processes in the system
    - Ex: Running Windows95 with 4 MB of memory…
  - Possible solutions
    - Swapping – write out all pages of a process
    - Buy more memory

Summary

- **Page replacement algorithms**
  - Belady's – optimal replacement (minimum # of faults)
  - FIFO – replace page loaded furthest in past
  - LRU – replace page referenced furthest in past
    - Approximate using PTE reference bit
  - LRU Clock – replace page that is "old enough"
  - Working Set – keep the set of pages in memory that has minimal fault rate (the "working set")
  - Page Fault Frequency – grow/shrink page set as a function of fault rate
- **Multiprogramming**
  - Should a process replace its own page, or that of another?

Next time…

- **Midterm review**