Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

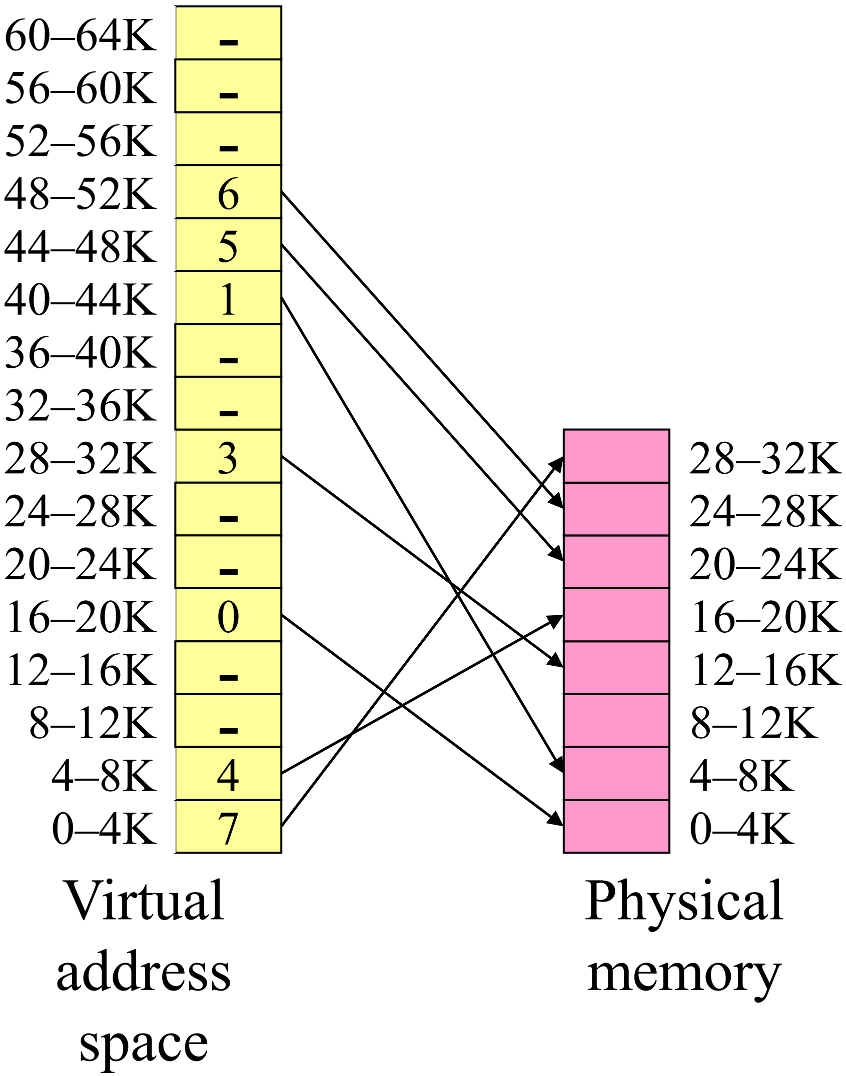
- Upcoming deadlines
  - Homework 8 is due this Friday: 2 extra attempts
  - Quiz 2 is this Friday at 11:59 pm
  - Lab 3 is due on Tuesday 3/28 at 11:59 pm
  - Project 3 is due Friday 4/7 at 11:59 pm

Previous Lecture …

- Memory allocation and protection
  - Take 1: Variable-size segments, base and limit registers
  - Take 2: Virtual memory
    - Fixed-size pages, on-demand, appear as if having more memory that physically in the system

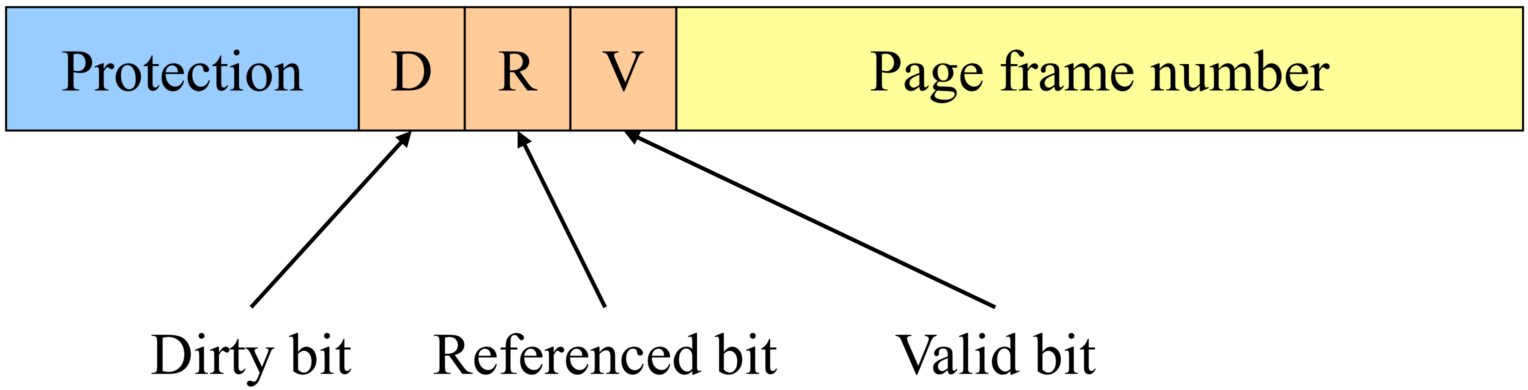
Paging and page tables

- Virtual addresses mapped to physical addresses
  - Unit of mapping is called a *page*
  - All addresses in the same virtual page are in the same physical page
  - *Page table entry* (PTE) contains translation for a single page
- Table translates virtual page number to physical page number
  - Not all virtual memory has a physical page
  - Not every physical page need be used
- Example:
  - 64 KB virtual memory
  - 32 KB physical memory



What's in a page table entry?

- Each entry in the page table contains
  - Valid bit: set if this logical page number has a corresponding physical frame in memory
    - If not valid, remainder of PTE is irrelevant
  - Page frame number: page in physical memory
  - Referenced bit: set if data on the page has been accessed
  - Dirty (modified) bit :set if data on the page has been modified
  - Protection information



Implementing page tables in hardware

- Page table resides in main (physical) memory
- CPU uses special registers for paging
  - Page table base register (PTBR) points to the page table
  - Page table length register (PTLR) contains length of page table: restricts maximum legal logical address
- Translating an address requires two memory accesses
  - First access reads page table entry (PTE)
  - Second access reads the data / instruction from memory
- Reduce number of memory accesses
  - Can’t avoid second access (we need the value from memory)
  - Eliminate first access by keeping a hardware cache (called a *translation lookaside buffer* or TLB) of recently used page table entries

Problem of the Day

- Page fault forces a choice
  - No room for new page (steady state)
  - A page must be removed to make room for an incoming page.
  - Which page to select?
    - Victim page
    - Evicted/purged

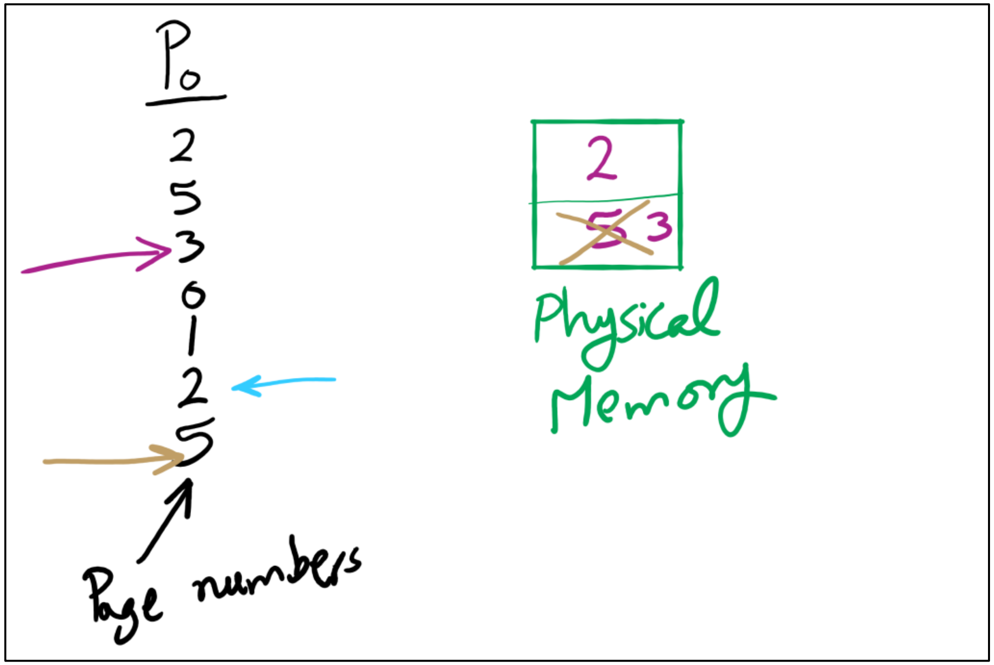
Page replacement algorithms

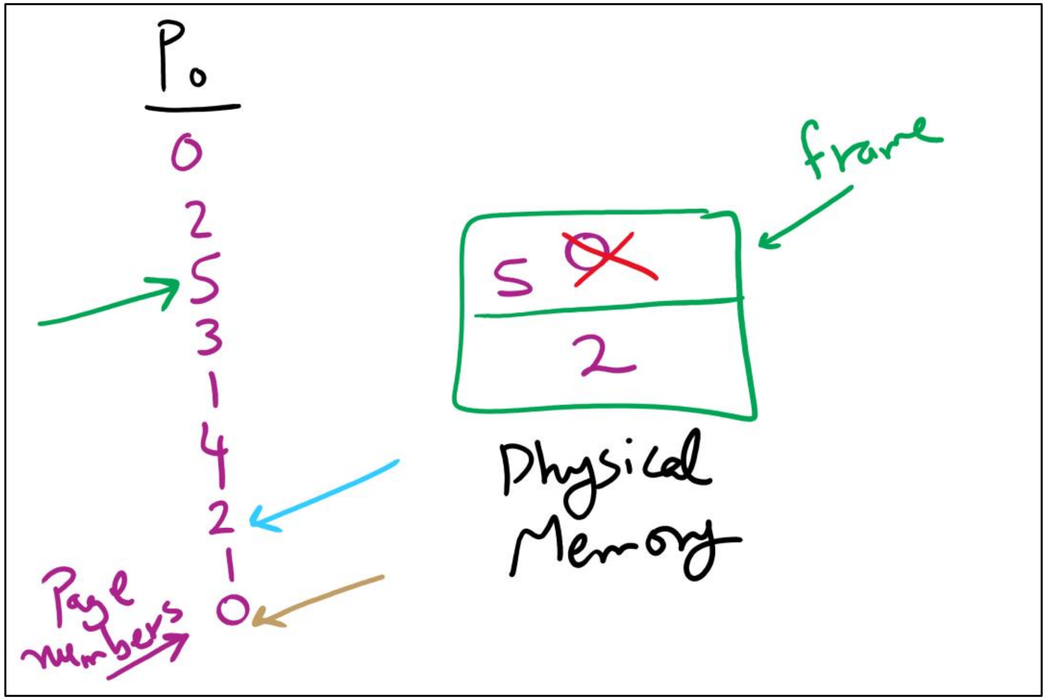
- How is a page removed from physical memory?
  - If the page is unmodified, simply overwrite it: a copy already exists on disk
  - If the page has been modified, it must be written back to disk: prefer unmodified pages?
- Better not to choose an often used page
  - It’ll probably need to be brought back in soon

Optimal page replacement algorithm

- What's the best we can possibly do?
  - Assume perfect knowledge of the future
  - Not realizable in practice (usually)
  - Useful for comparison: if another algorithm is within 5% of optimal, not much more can be done…
- Algorithm: replace the page that will be used furthest in the future
  - Only works if we know the whole sequence!
  - Can be approximated by running the program twice
    - Once to generate the reference trace
    - Once (or more) to apply the optimal algorithm
- Nice, but not achievable in real systems!

OPT Examples





Not-recently-used (NRU) algorithm

- Each page has reference bit and dirty bit
  - Bits are set when page is referenced and/or modified
- Pages are classified into four classes
  - 0: not referenced, not dirty
  - 1: not referenced, dirty
  - 2: referenced, not dirty
  - 3: referenced, dirty
- Clear reference bit for all pages periodically
  - Can’t clear dirty bit: needed to indicate which pages need to be flushed to disk
  - Class 1 contains dirty pages where reference bit has been cleared
- Algorithm: remove a page from the lowest non-empty class
  - Select a page at random from that class
- Easy to understand and implement
- Performance adequate (though not optimal)

NRU Operation

	every memory access	every clock interrupt	every victim selection
NRU	R, D bits are set (MMU)	R bits reset (OS)	(0,0) (0,1) (1,0) (1,1) (OS)

First-In, First-Out (FIFO) algorithm

- Maintain a linked list of all pages
  - Maintain the order in which they entered memory
- Page at front of list replaced
- Advantage: (really) easy to implement
- Disadvantage: page in memory the longest may be often used
  - This algorithm forces pages out regardless of usage
  - Usage may be helpful in determining which pages to keep

Page Replacement Algorithms Components

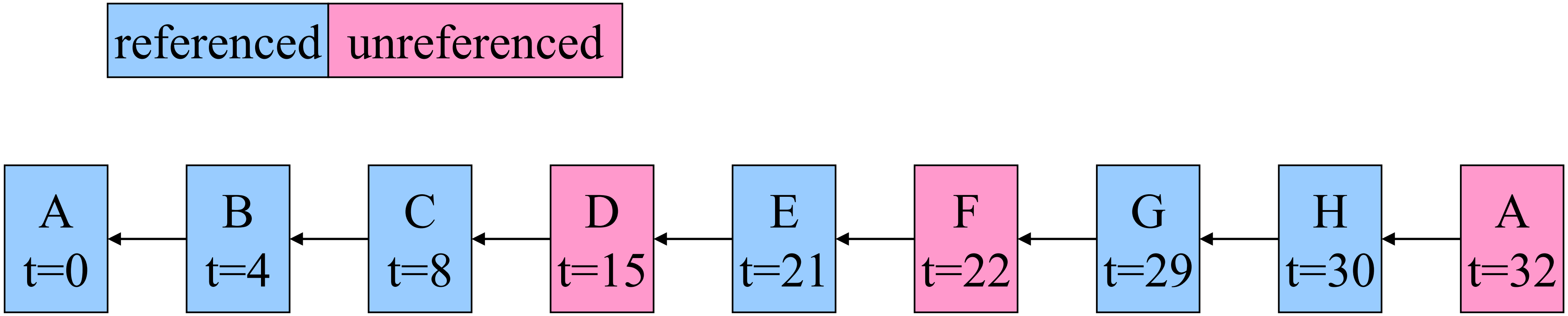
every memory access

every clock interrupt

every victim selection

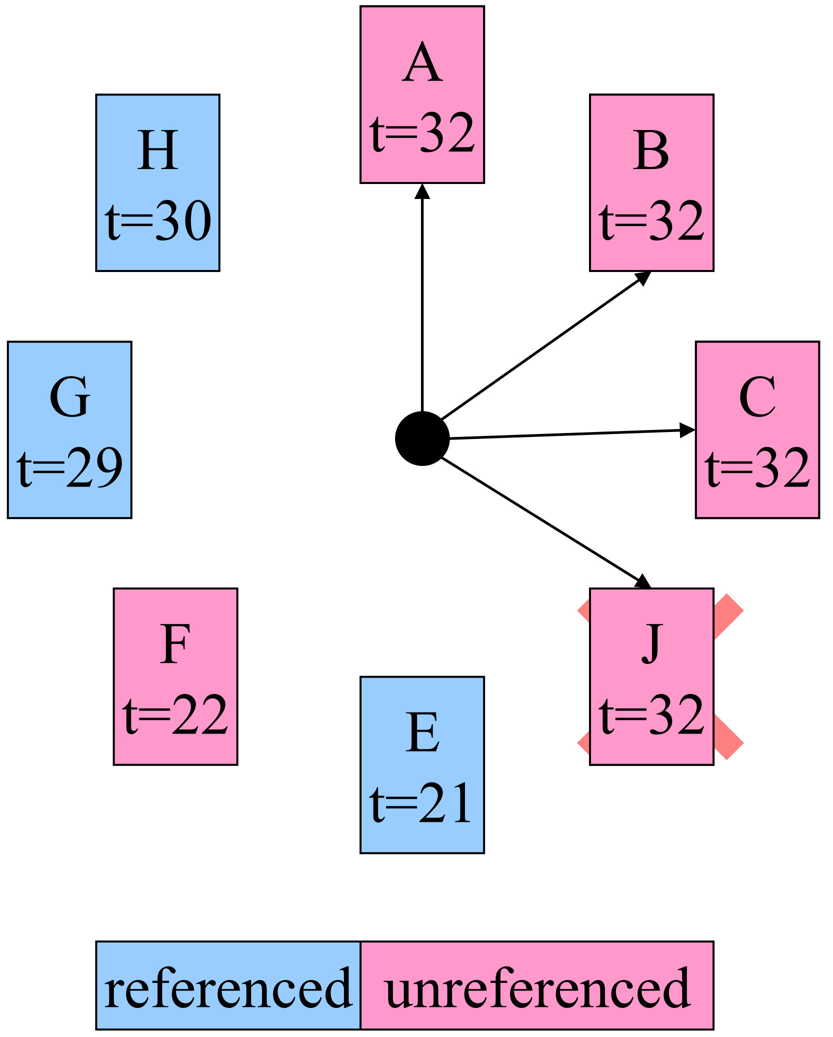
Second chance page replacement

- Modify FIFO to avoid throwing out heavily used pages
  - If reference bit is 0, throw the page out
  - If reference bit is 1
    - Reset the reference bit to 0
    - Move page to the tail of the list
    - Continue search for a free page
- Still easy to implement, and better than plain FIFO



Clock algorithm

- Same functionality as second chance
- Simpler implementation
  - “Clock” hand points to next page to replace
  - If R=0, replace page
  - If R=1, set R=0 and advance the clock hand
- Continue until page with R=0 is found
  - This may involve going all the way around the clock…



Least Recently Used (LRU)

- Assume pages used recently will be used again soon
    - Throw out page that has been unused for longest time
- Must keep a linked list of pages
    - Most recently used at front, least at rear
    - Update this list every memory reference!
        - This can be somewhat slow: hardware has to update a linked list on every reference!
- Alternatively, keep counter in each page table entry
    - Global counter increments with each CPU cycle
    - Copy global counter to PTE counter on a reference to the page
    - For replacement, evict page with lowest counter value

Simulating LRU in software

- Few computers have the necessary hardware to implement full LRU
  - Linked-list method impractical in hardware
  - Counter-based method could be done, but it’s slow to find the desired page
- Approximate LRU with Not Frequently Used (NFU) algorithm
  - At each clock interrupt, scan through page table
  - If R=1 for a page, add one to its counter value
  - On replacement, pick the page with the lowest counter value
- Problem: no notion of age—pages with high counter values will tend to keep them!

Aging replacement algorithm

- Reduce counter values over time
  - Divide by two every clock cycle (use right shift)
  - More weight given to more recent references!
- Select page to be evicted by finding the lowest counter value
- Algorithm is:
  - Every clock tick, shift all counters right by 1 bit
  - On reference, set leftmost bit of a counter (can be done by copying the reference bit to the counter at the clock tick)

Referenced this tick

	Tick 0	Tick 1	Tick 2	Tick 3	Tick 4
Page 0	10000000	11000000	11100000	01110000	10111000
Page 1	00000000	10000000	01000000	00100000	00010000
Page 2	10000000	01000000	00100000	10010000	01001000
Page 3	00000000	00000000	00000000	10000000	01000000
Page 4	10000000	01000000	10100000	11010000	01101000
Page 5	10000000	11000000	01100000	10110000	11011000