Mechanism of Page Eviction

- flush TLB, clear PTE present bit for the evicted page
- allocate space in swap, write evicted page to swap
- track page => swap loc
- zero out old content before mapping it to a new page

Today: which page to evict?

Question in class: why write page to swap instead of using disk directly (map page to swap)

-> architecture:
   code and data needs to be in memory to be accessed by the CPU
   disk is block addressable, much much slower in access latency

   There's a non traditional persistent device: non volatile memory (byte addressable, slower access than DRAM, but can be accessed as memory directly), offers an interesting design space
Eviction Policies: What page/frame to evict?

- FIFO
  - pick the page that's brought in first (longest time in memory)
  - a queue of frames in order of allocation
  - doesn’t care about access patterns

Belady's Anomaly

More frames may cause more page faults with FIFO policy

Page accessed in this order: A, B, C, D, A, B, E, A, B, C, D, E

FIFO 3 frames

Frame 1: A, A, A, D, D, D, E, E, E, E, E

FIFO 4 frames

Frame 4: - - - D, D, D, D, D, D, D, C, C, C

(9 PFS)

(10 PFS)
Least Recently Used
- evict page that’s least recently used
- good when access exhibits locality
- worst case: N frames N+1 pages accessed in order

How to implement this?
- SW: queue, when a page is used, bring it to the back, head of queue is the LRU page
- HW: implement HW queue, or add timestamp update to PTE
  needs to be done on every access (TLB miss & TLB hits)

Clock Algorithm
- approximates LRU
- uses access bit (PTE) to estimate if the page is used recently
- algorithm starts at clock hand (stateful, advances each time the algo runs)
  
  if (access bit == 0) { found our page to evict; advance clock hand 3 }
  else { set access bit to 0 & advance clock hand 3 }

needs to also do a TLB shutdown to ensure future access to the page updates the access bit back to 1