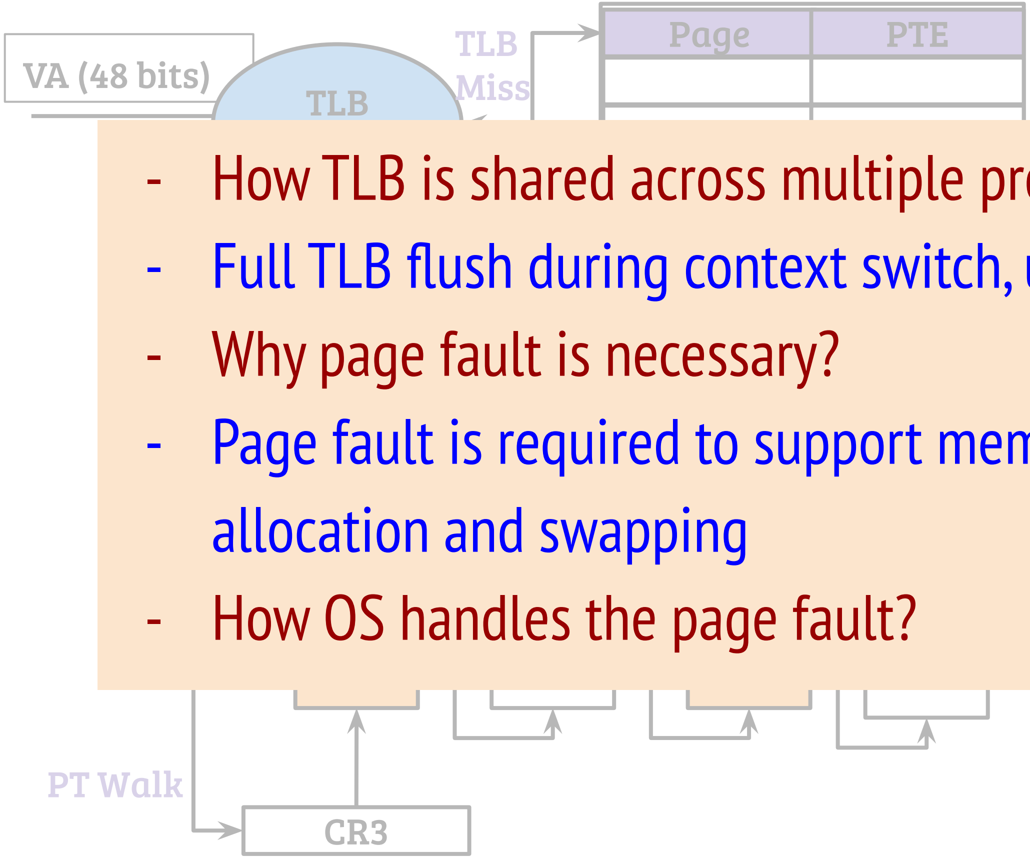
CS330: Operating Systems

Virtual memory: Page fault and Swapping

Recap: Address translation



- How TLB is shared across multiple processes?
- Full TLB flush during context switch, using ASID
- Why page fault is necessary?
- Page fault is required to support memory over-commitment through lazy allocation and swapping
- How OS handles the page fault?

Page fault handling in X86: Hardware

```
If(  !pte.valid ||
    (access == write && !pte.write) ||
    (cpl != 0 && pte.priv == 0)){
        CR2 = Address;
        errorCode = pte.valid
                  | access << 1
                  | cpl << 2;
        Raise pageFault;
} // Simplified
```

Page fault handling in X86: Hardware

```
If( !pte.valid ||
    (access == write && !pte.write) ||
    (cpl != 0 && pte.priv == 0)){
        CR2 = Address;
        errorCode = pte.valid
                    | access << 1
                    | cpl << 2;
        Raise pageFault;
} // Simplified
```

Error code

Other and unused	I	R	U	W	P

- **P** — Present bit, 1 $\Rightarrow$ fault is due to protection
- **W** — Write bit, 1 $\Rightarrow$ Access is write
- **U** — Privilege bit, 1 $\Rightarrow$ Access is from user mode
- **R** — Reserved bit, 1 $\Rightarrow$ Reserved bit violation
- **I** — Fetch bit, 1 $\Rightarrow$ Access is Instruction Fetch

- Error code is pushed into the kernel stack by the hardware

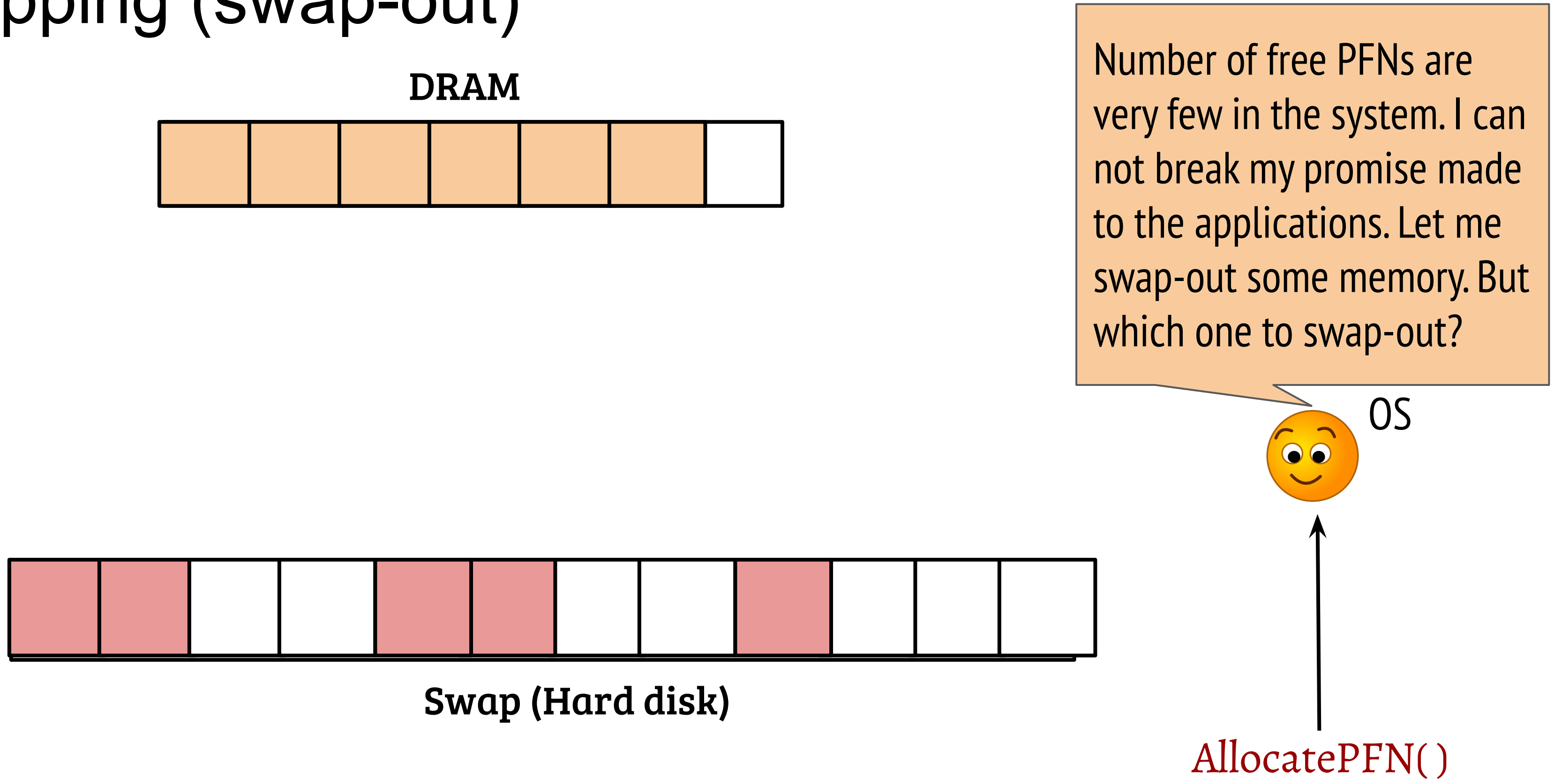
Page fault handling in X86: OS fault handler

```
HandlePageFault( u64 address, u64 error_code)
{
    If ( AddressExists(current → mm_state, address) &&
        AccessPermitted(current → mm_state, error_code) {
            PFN = allocate_pfn( );
            install_pte(address, PFN);
            return;
        }
    RaiseSignal(SIGSEGV);
}
```

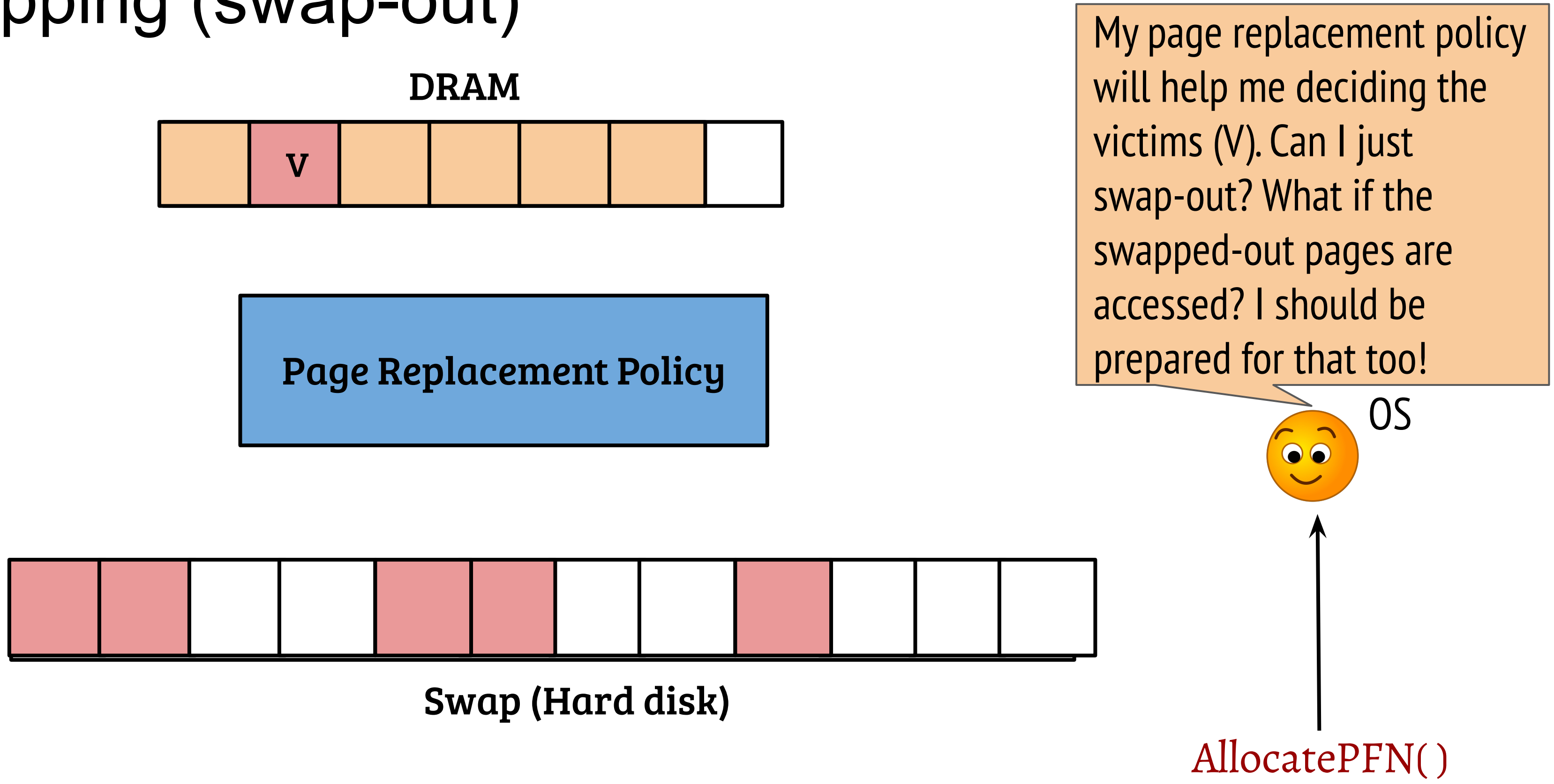
Address translation (TLB + PTW)

- How TLB is shared across multiple processes?
- Full TLB flush during context switch, using ASID
- Why page fault is necessary?
- Page fault is required to support memory over-commitment through lazy allocation and swapping
- How OS handles the page fault?
- The hardware invokes the page fault handler by placing the error code and virtual address. The OS handles the page fault either fixing it or raising a SEGFAULT.

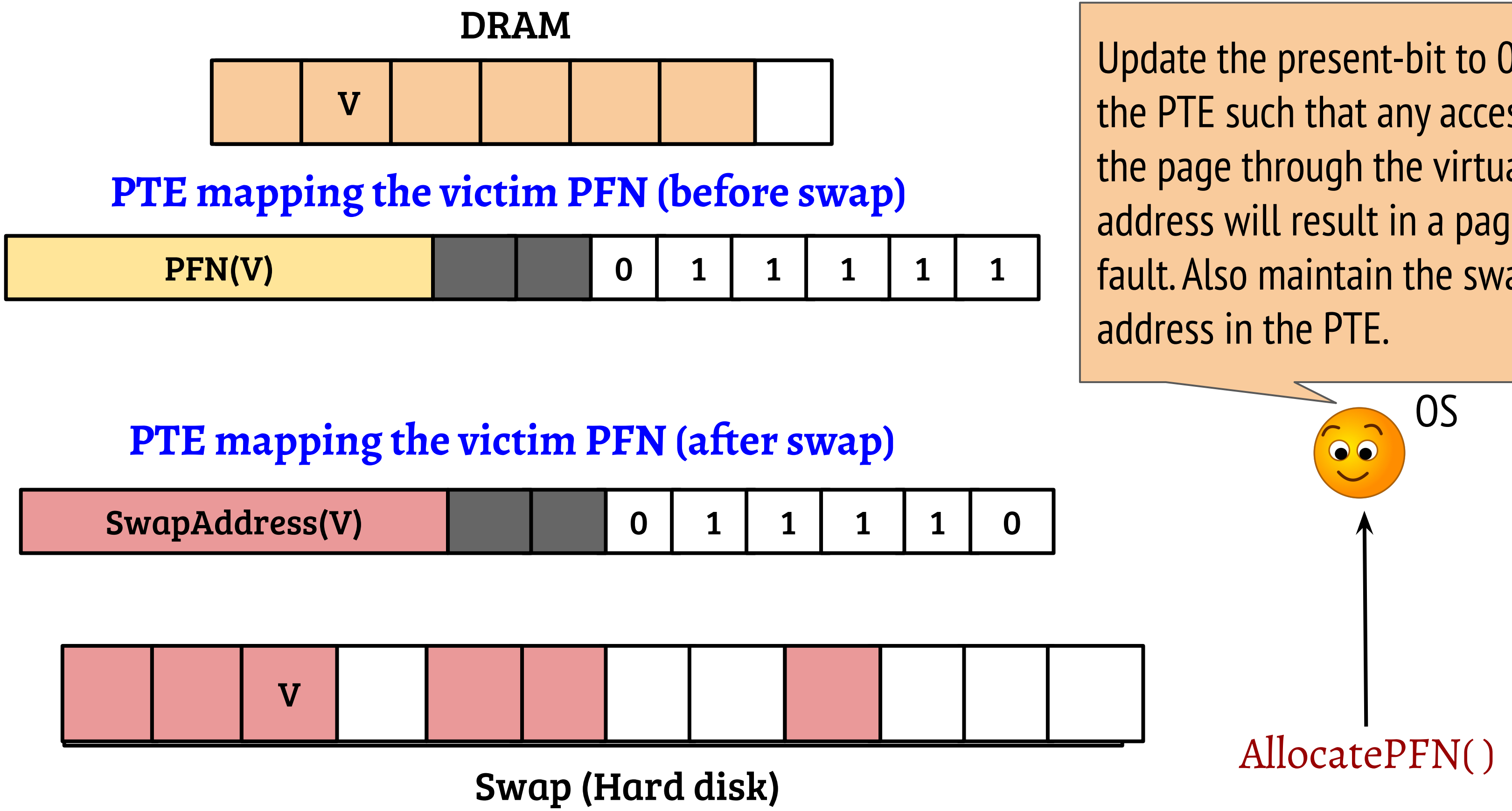
Swapping (swap-out)



Swapping (swap-out)

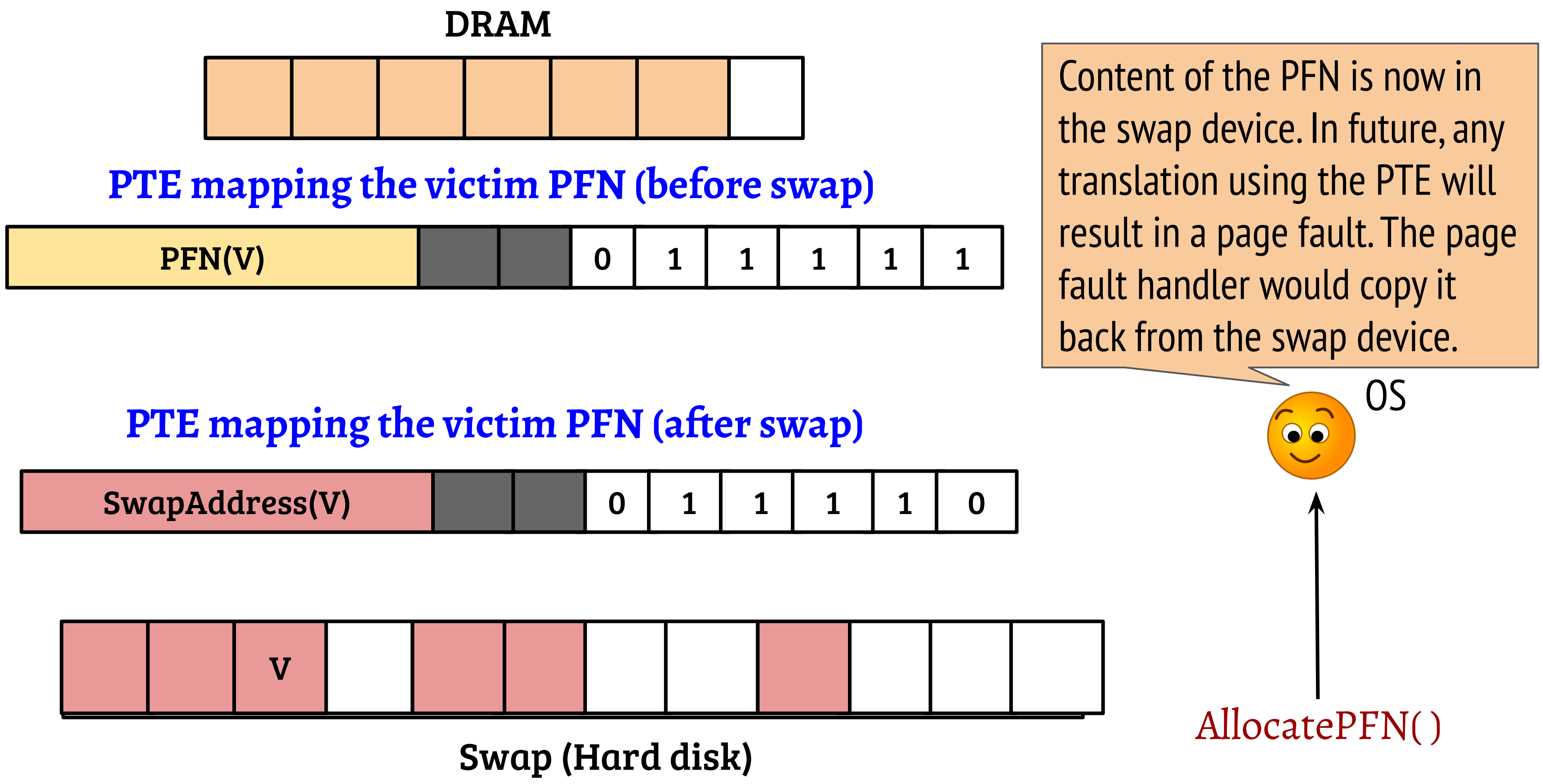


Swapping (swap-out)



Update the present-bit to 0 in the PTE such that any access to the page through the virtual address will result in a page fault. Also maintain the swap address in the PTE.

Swapping (swap-out)



Page fault: Swap-in

```
HandlePageFault( u64 address, u64 error_code)
{
    If ( AddressExists(current → mm_state, address) &&
        AccessPermitted(current → mm_state, error_code) {
            PFN = allocate_pfn( );
            If ( is_swapped_pte(address) )      // Check if the PTE is swapped out
              swapin(getPTE(address), PFN);   // Copy the swap block  to PFN
            install_pte(address, PFN);          // and update the PTE
            return;
        }
    RaiseSignal(SIGSEGV);
}
```

Page replacement

- Objective: minimize number of page faults (due to swapping)
- We can model this problem with three parameters
    - A given sequence of access to virtual pages
    - # of memory pages (Frames)
    - Page replacement policy
- Metrics to measure the effectiveness: # of page faults, page fault rate, average memory access time

Belady’s optimal algorithm (MIN)

- Strategy: Replace the page that will be referenced after the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = ?

Belady's optimal algorithm (MIN)

- Strategy: Replace the page that will be referenced after the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = 6 (3 cold-start misses result in page faults, no swapping)
- Belady's MIN is proven to be optimal, but impractical as it requires knowledge of future access

First In First Out (FIFO)

- Strategy: Replace the page that is in memory for the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = ?

First In First Out (FIFO)

- Strategy: Replace the page that is in memory for the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = 8 (3 cold-start misses)
- FIFO suffers from an anomaly known as Belady's anomaly
  - With increased #of frames, #of page fault may also increase!

First In First Out (FIFO)

- Strategy: Replace the page that is in memory for the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = 8 (3 cold-start misses)
- FIFO suffers from an anomaly known as Belady's anomaly
  - With increased #of frames, #of page fault may also increase!
  - Example access sequence: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4
  - #of page faults with 3 frames < #of page faults with 4 frames

Least recently used (LRU)

- Strategy: Replace the page that is not referenced for the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = ?

Least recently used (LRU)

- Strategy: Replace the page that is not referenced for the longest time
- Example:

  #of frames = 3

  Reference sequence (in temporal order)

  1, 3, 1, 5, 4, 1, 2, 5, 2, 2, 5, 3
- #of page faults = 7 (3 cold-start)
- LRU shown to be useful for workloads with access locality
- Implementation of LRU using the accessed-bit is not easy, approximated using CLOCK