CS330: Operating Systems
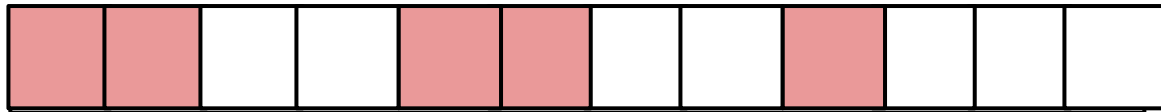
Virtual memory: Page fault and Swapping
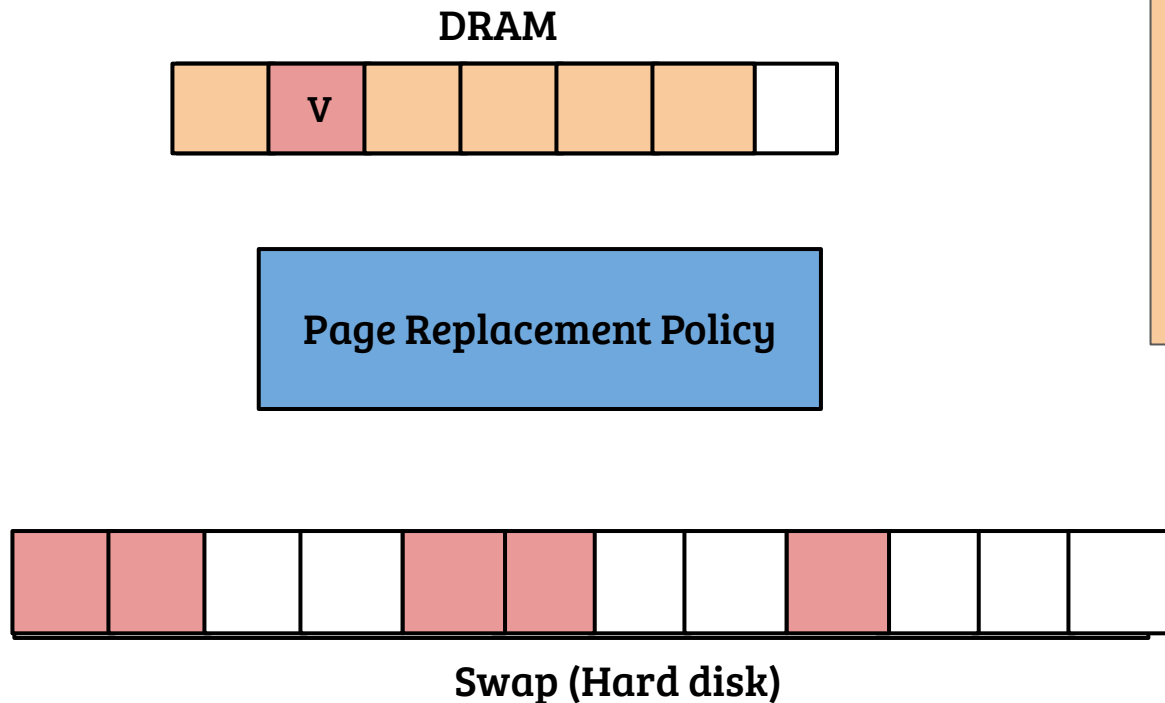
Swapping (swap-out)

DRAM

Number of free PFNs are very few in the system. I can not break my promise made to the applications. Let me swap-out some memory. But which one to swap-out?

OS

Swap (Hard disk)

AllocatePFN( )

Swapping (swap-out)

DRAM

Page Replacement Policy

Swap (Hard disk)

My page replacement policy will help me deciding the victims (V). Can I just swap-out? What if the swapped-out pages are accessed? I should be prepared for that too!
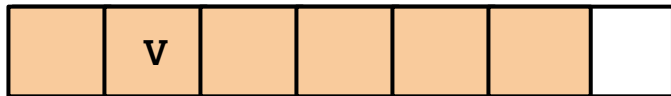
OS

AllocatePFN( )

Swapping (swap-out)

DRAM

	V					

PTE mapping the victim PFN (before swap)

PFN(V)			0	1	1	1	1	1

PTE mapping the victim PFN (after swap)

SwapAddress(V)			0	1	1	1	1	0

		V										

Swap (Hard disk)

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.
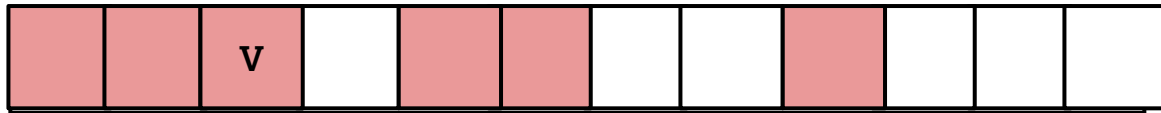
OS

AllocatePFN( )

Swapping (swap-out)

DRAM

PTE mapping the victim PFN (before swap)

| PFN(V) | | | 0 | 1 | 1 | 1 | 1 | 1 |

PTE mapping the victim PFN (after swap)

| SwapAddress(V) | | | 0 | 1 | 1 | 1 | 1 | 0 |

Swap (Hard disk)

Content of the PFN is now in the swap device. In future, any translation using the PTE will result in a page fault. The page fault handler would copy it back from the swap device.

OS

AllocatePFN( )

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