# Operating Systems Concepts

## Virtual Memory: Page Faults and Memory Swapping

CS 4375, Fall 2025

**Instructor:** MD Armanuzzaman (***Arman***)
*marmanuzzaman@utep.edu*

October 22, 2025

# Summary

- Main Memory:
    - Implementation of Paging
    - Effective Access Time
    - Multi-level Paging
    - Example: Intel 32-bit Architecture

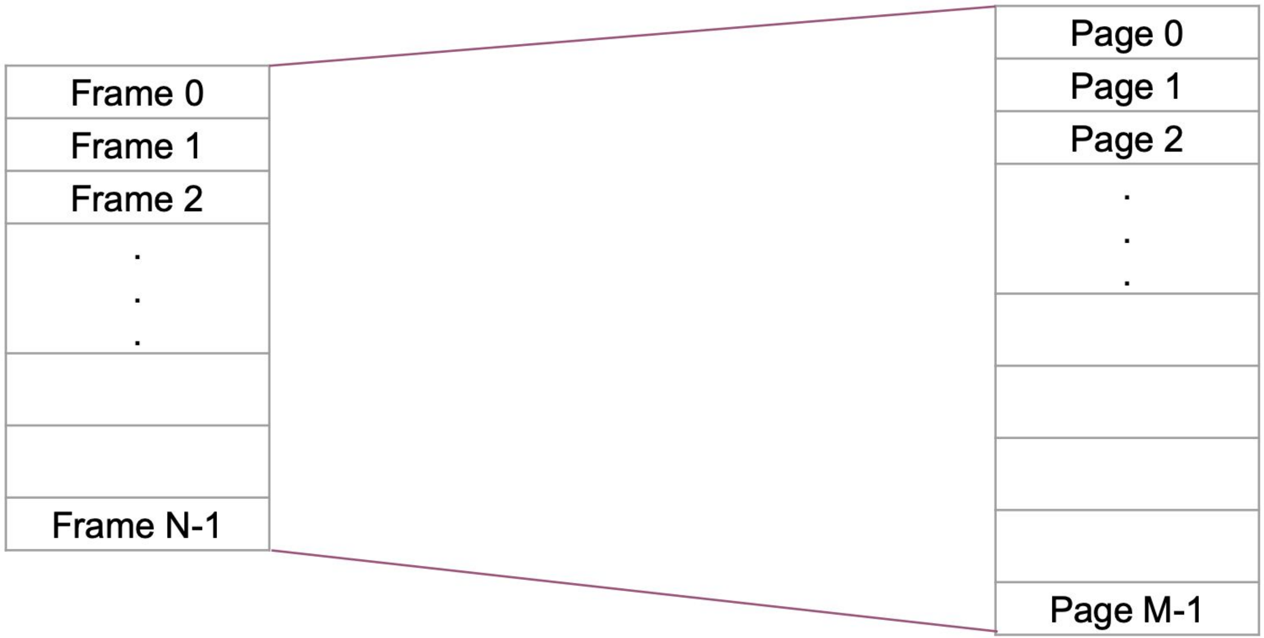
# Agenda

- Virtual Memory:
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms
    - FIFO
    - Optimal Algorithm
    - LRU & LRU Approximations
    - Counting Algorithms

# Virtual Memory

- Separation of user logical memory from physical memory
  - Only part of the program needs to be in the memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation

# Virtual Memory




**Physical Address Space (installed RAM)**

Small (2-8 GB)

Messy & Fragmented

**Virtual Address Space**

Large (100 GB)

Clean

# Virtual Memory Goals

- Making programmers' job **easier**
  - Can write code without knowing how much DRAM is there
  - Only need to know general memory architecture
    - E.g., 32-bit address space
- Enable **multiprogramming**
  - Keep several programs running concurrently
    - Together, these programs may need more DRAM than we have
    - Keep just the actively used pages in DRAM
  - Share when possible
    - When one program does I/O, switch CPU to another

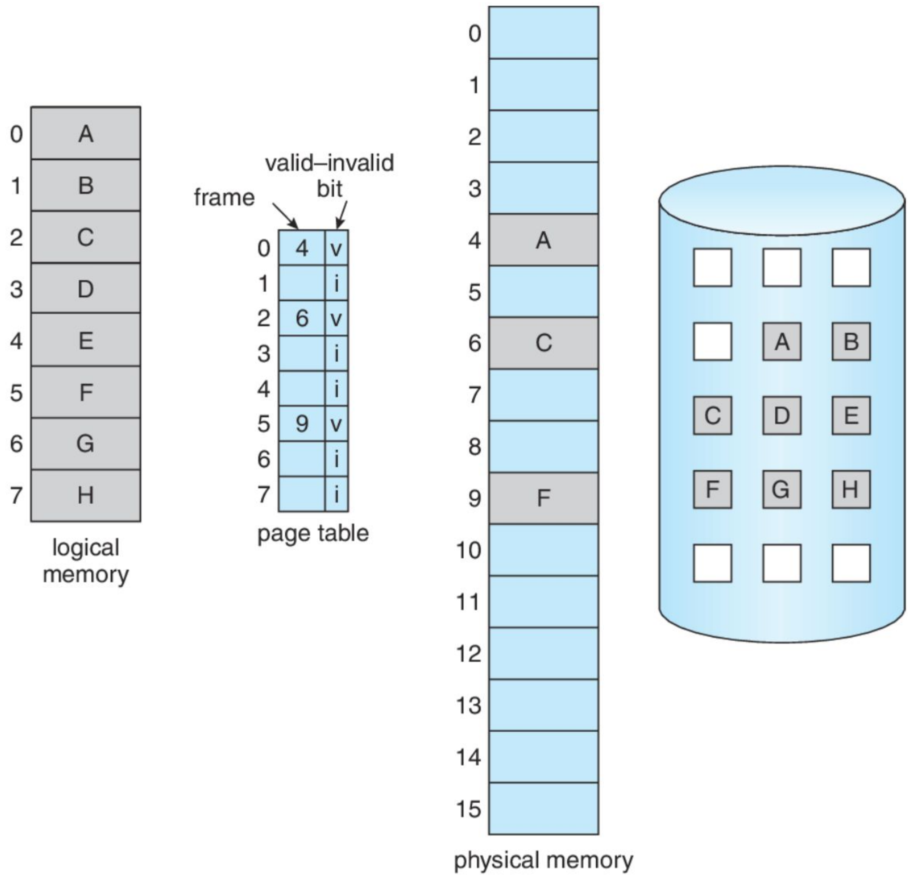
# Virtual Memory Implementation

- Virtual Memory can be implemented by:
  - Demand paging
  - Demand segmentation

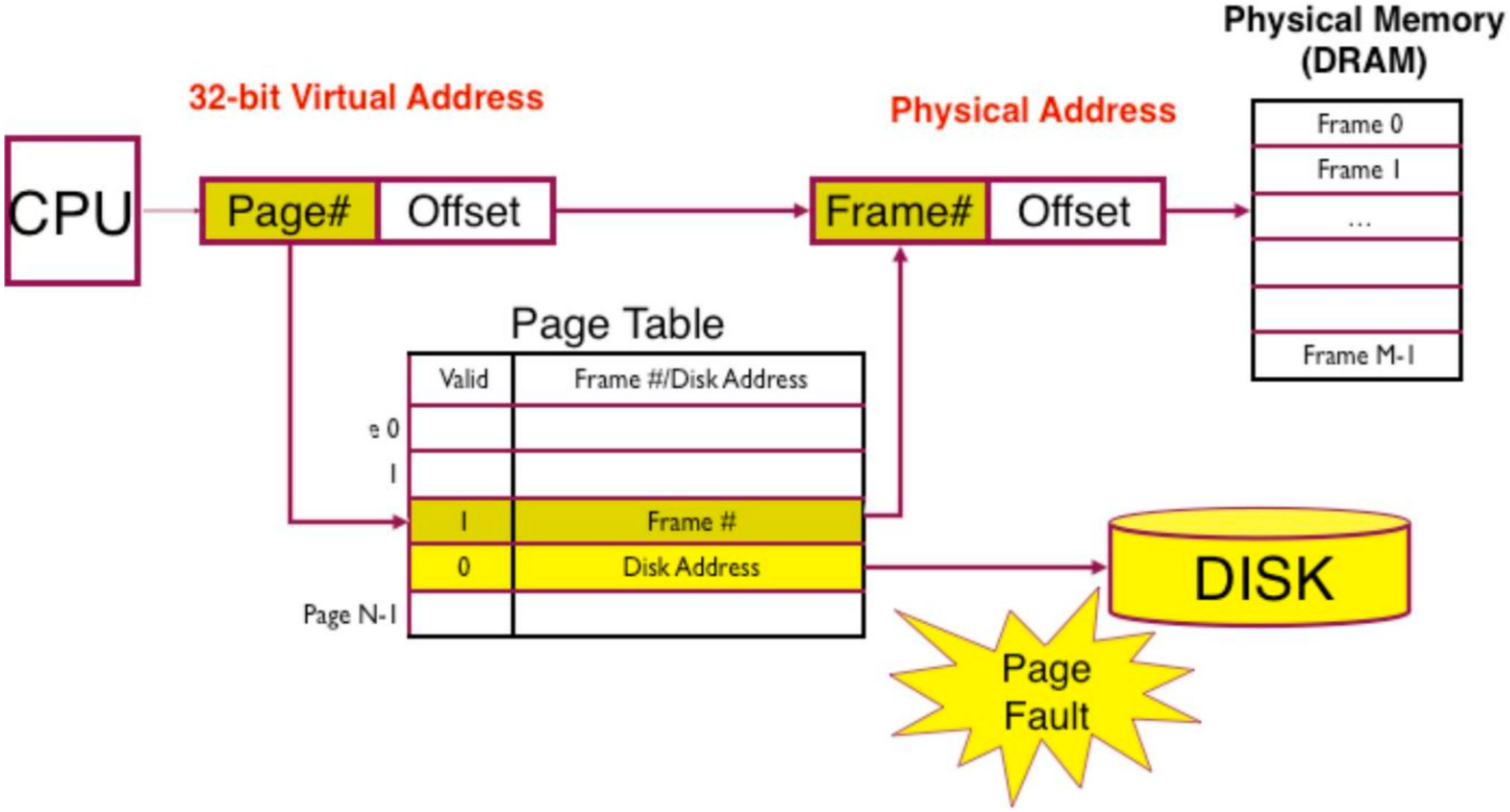
# Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed → Reference to it
  - Invalid reference? → Abort!
  - Not-in-memory? → Bring to memory

# Demand Paging - Valid/Invalid Bit



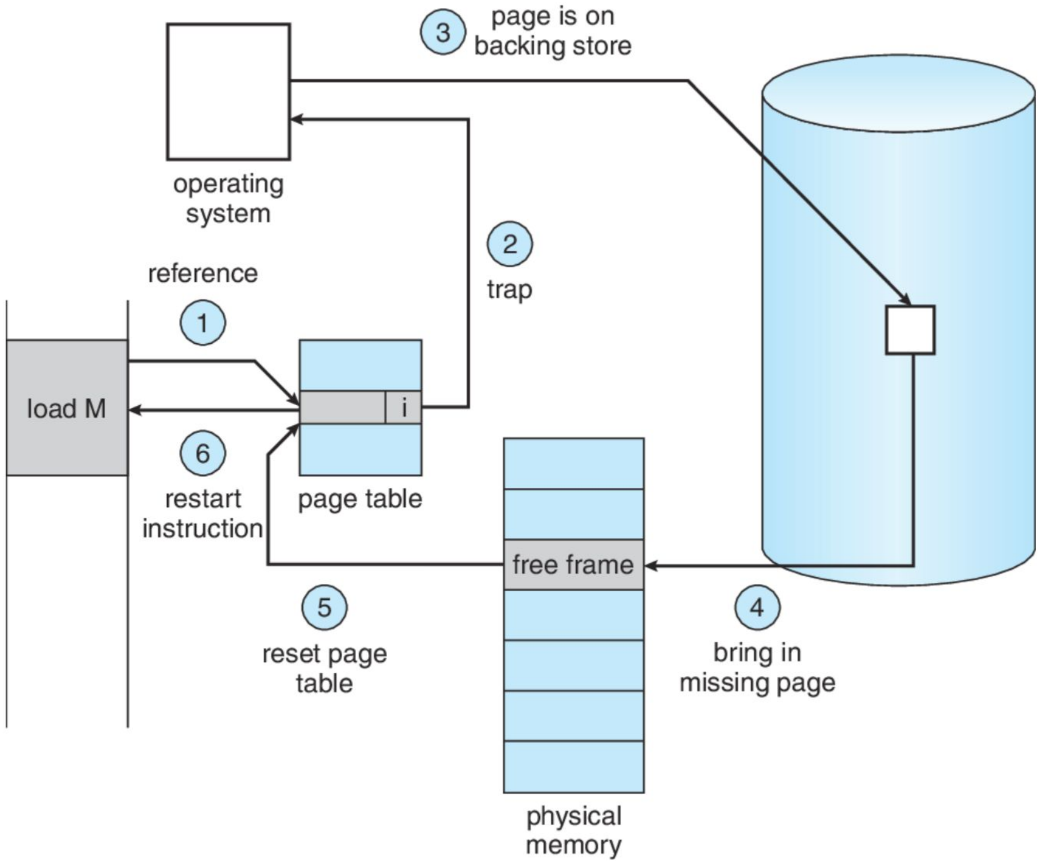
# Demand Paging - Valid/Invalid Bit



# Demand Paging - Page Fault

- If there is ever a reference to a page not in memory, first reference will trap to OS **(Page fault)**
- OS looks at another table (in PCB) to decide:
  - Invalid reference (out of bounds or etc.) → Abort the process
  - Valid reference but not in memory → Swap-in
    - Get an empty frame
    - Swap (read) page into the new frame
    - Set the page table, and validation bit=1
    - Restart instruction

# Demand Paging - Page Fault



# Page Replacement

## What happens if there is no free frame?

- Page replacement: Find some page that resides in memory, but is not really in use, swap it out to free up space.
    - Algorithms (FIFO, LRU, ...)
    - Performance: Want an algorithm which will result in minimum number of page faults.
    - Same page may be brought into memory several times

# Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers

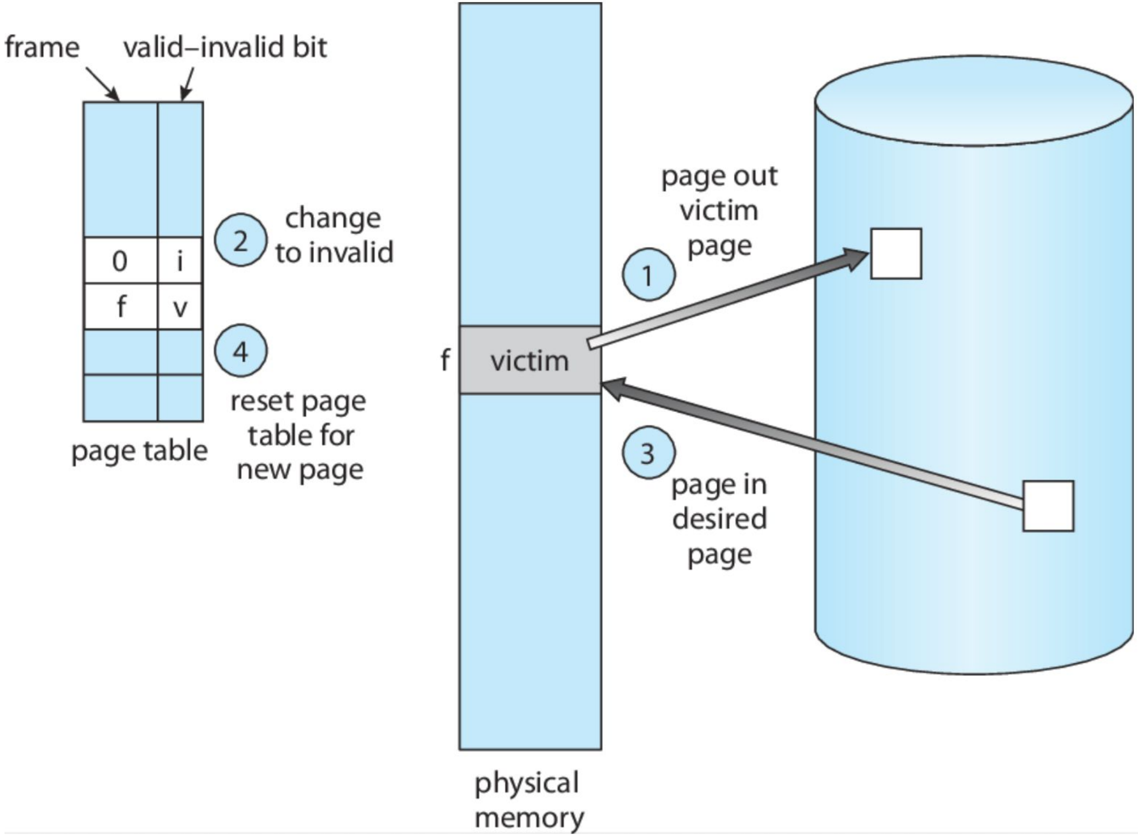
  Only modified pages are swapped out (written to disk)

- Page replacement completes separation between logical memory and physical memory
- Large virtual memory can be provided on a smaller physical memory

# Page Replacement - Simplified

1. Find the location of the desired page on disk
2. Find a free frame:
    a. If there is a free frame, use it
    b. If there is no free frame, use a page replacement algorithm to select a **victim**
3. Read the desired page into the [newly] free frame. Update the page and frame tables
4. Restart the process

# Page Replacement



# Page Replacement Algorithms

- We want lowest page fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In our examples, the reference string is:
- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

# Page Replacement Algorithms - FIFO

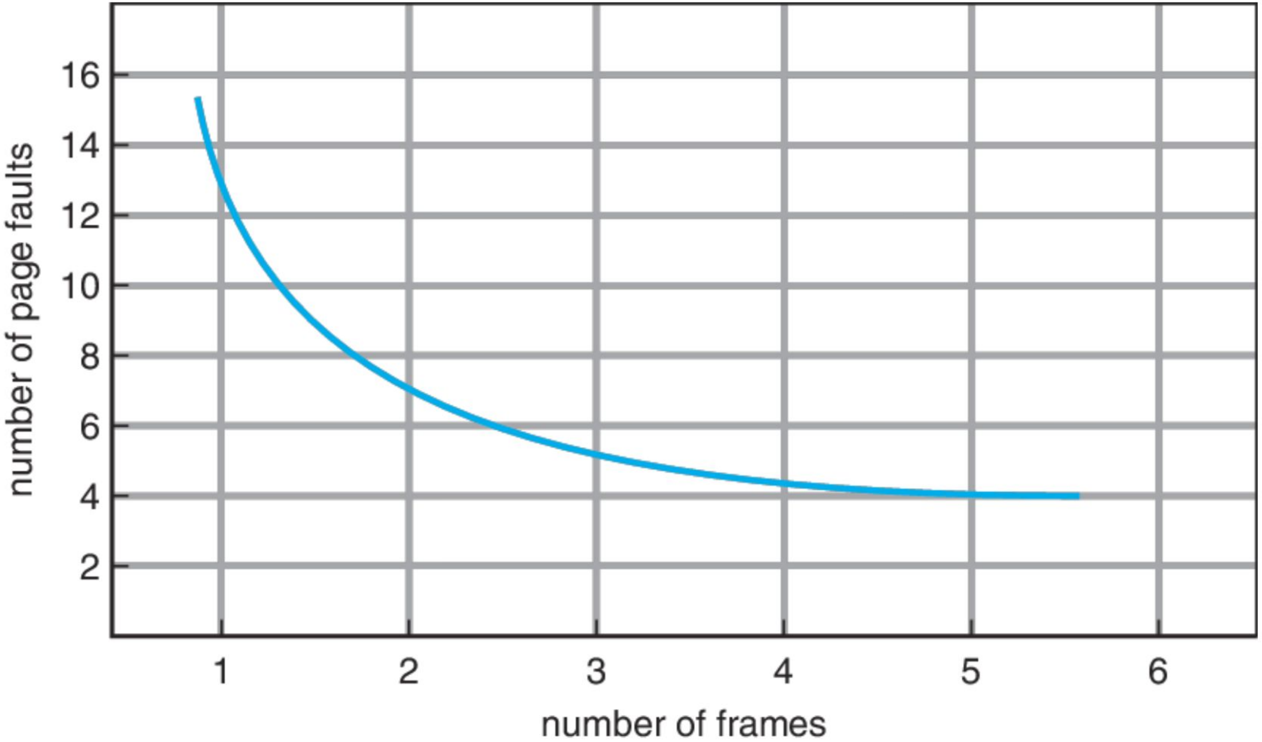
- First-In, First-Out (FIFO)
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time)
- The blue arrow represents the next victim ➡

| Referenced Page | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | ➡ | 1 | 1 | 1 ➡ | 4 | 4 | 4 ➡ | 5 | 5 | 5 | 5 | 5 ➡ | 5 ➡ |
| Frame-2 | | ➡ | 2 | 2 | 2 ➡ | 1 | 1 | 1 ➡ | 1 ➡ | 1 ➡ | 3 | 3 | 3 |
| Frame-3 | | | ➡ | 3 | 3 | 3 ➡ | 2 | 2 | 2 | 2 | 2 ➡ | 4 | 4 |

**9 page faults**

# Page Replacement Algorithms - FIFO

- Expected Graph of Page Faults vs The Number of Frames



# Page Replacement Algorithms - FIFO

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 4 frames (4 pages can be in memory at a time)

| Referenced Page | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | ➡ | 1 | 1 | 1 | 1 ➡ | 1 ➡ | 1 ➡ | 5 | 5 | 5 | 5 ➡ | 4 | 4 |
| Frame-2 | | ➡ | 2 | 2 | 2 | 2 | 2 | 2 ➡ | 1 | 1 | 1 | 1 ➡ | 5 |
| Frame-3 | | | ➡ | 3 | 3 | 3 | 3 | 3 | 3 ➡ | 2 | 2 | 2 | 2 ➡ |
| Frame-4 | | | | ➡ | 4 | 4 | 4 | 4 | 4 | 4 ➡ | 3 | 3 | 3 |

10 page faults → Belady's Anomaly (more frames, more page faults)

# Page Replacement Algorithms - FIFO

- FIFO is obvious, and simple to implement
  - When you page in something, put it on the tail of a list
  - Evict page at the head of the list
- Why might this be good?
  - Maybe the one brought in longest ago is not being used
- Why might this be bad?
  - Maybe the one brought a while ago ***is being used***!
  - **No information** either way!
- In fact, FIFO's performance is typically lousy
- FIFO also suffers from Belady's anomaly

# Page Replacement Algorithms - Optimal

- Replace page that **will not be used** for the longest time in future
- 4 frames available. Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

| Referenced Page | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 |
| Frame-2 | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame-3 | | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Frame-4 | | | | | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |

6 page faults

- **How would you know in advance?**

# Page Replacement Algorithms - Optimal

- Belady's optimal algorithm is **provably optimal**, with lowest fault rate
    - Evict the page that won't be used for the longest time in future
    - It is impossible to predict the future!
- Why Belady's optimal algorithm useful?
    - As a reference to compare other algorithms
- Is there a best practical algorithm?
    - No, it depends on the workload
- Is there a worst algorithm?
    - No, but random replacement is pretty bad!
        - However there are some situations where OSs use near-random algorithms quite effectively

# Page Replacement Algorithms - LRU

- Least Recently Used (LRU)
- Replace the page that has not been used for the longest amount of time in the past
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

| Referenced Page | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 |
| Frame-2 | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame-3 | | | | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 4 | 4 |
| Frame-4 | | | | | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |

8 page faults

# Page Replacement Algorithms - LRU

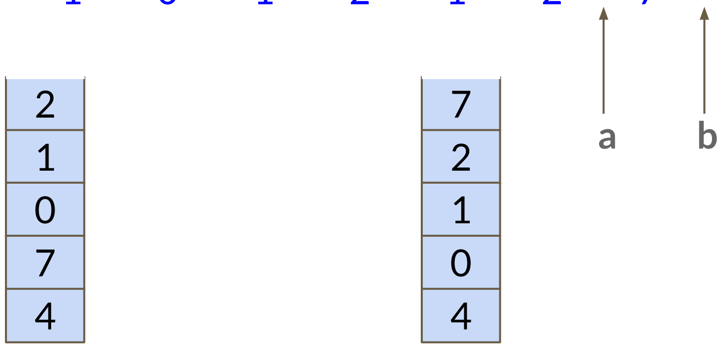
- LRU uses reference information to make a more informed replacement decision
- Idea: past experience gives us a guess of future behavior
- Evict the page that hasn't been used for the longest time
    - LRU looks at the past (Belady's optimal algorithm looks at future)
    - How is LRU different from FIFO?
- Implementation
    - To be perfect, must grab a timestamp on every memory reference, then order or search based on the timestamps …
    - Way too costly!
    - We need a cheap approximation

# Page Replacement Algorithms - LRU Implementation

- **Stack implementation:** Keep a stack of page numbers in a double link form
  - Page referenced:
    - Move it to the top
    - Requires 6 pointers to be changed
  - No search for replacement

reference string:

4 7 0 7 1 0 1 2 1 2 7 1 2




stack before a

stack before b

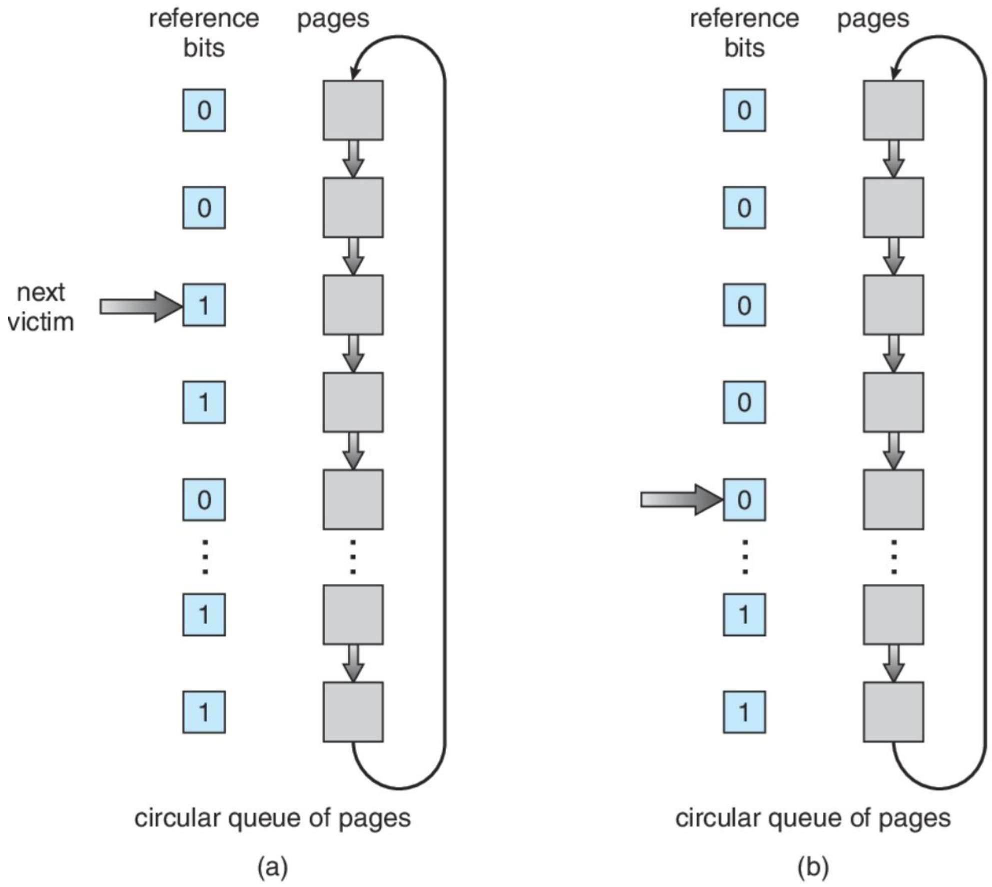
# Page Replacement Algorithms - LRU Approximation

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced, bit set to 1
  - Replace the one which is 0 (if one exists). however , we do not know the order.
- Additional reference bits
  - 1 byte for each page. E.g. 00110011
  - Shift right at each time interval

# Page Replacement Algorithms - Second Chance

- LRU-Clock algorithm, also known as Not Recently Used (NRU) or Second Chance
  - Replace page that is old enough
  - Logically, arrange all physical page frames in a big circle (clock)
    - A circular linked list
  - A clock hand is used to select a good LRU candidate
    - Sweep through the pages in circular order like a clock
    - If reference bit is off, it hasn't been used recently → we have a victim!
    - If the reference bit is on, turn it off and go to next one → second chance
  - Arm moves quickly when pages are needed
  - Low overhead if we have plenty of memory
  - We can add more clock hands to improve

# Page Replacement Algorithms - Second Chance



# Page Replacement Algorithms - Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- Least Frequently Used (LFU)
    - Replaces page with smallest count
- Most Frequently Used (MFU)
    - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used

# Page Replacement Algorithms - Exercise

Consider the following page reference string:

1, 2, 3, 4, 4, 3, 2, 1, 5, 6, 2, 1, 2, 3, 7, 8, 3, 2, 1, 5

Assuming 4 memory frames and LRU, LFU, or Optimal page replacement algorithms:

- How many page faults, page hits, and page replacements would occur?
- Show your page assignments to frames

| Referenced Page | | 1 | 2 | 3 | 4 | 4 | 3 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 | 7 | 8 | 3 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | | | | | | | | | | | | | | | | | | | | | |
| Frame-2 | | | | | | | | | | | | | | | | | | | | | |
| Frame-3 | | | | | | | | | | | | | | | | | | | | | |
| Frame-4 | | | | | | | | | | | | | | | | | | | | | |

**# of page faults:** **# of page hits:** **# of page replacements:**

# Page Replacement Algorithms - Exercise

Consider the following page reference string:

1, 2, 3, 4, 4, 3, 2, 1, 5, 6, 2, 1, 2, 3, 7, 8, 3, 2, 1, 5

Assuming 4 memory frames and LRU-Clock page replacement algorithm:

1. When a page is brought to the memory, reference bit is initialized to 0
2. Advance the victim pointer only if you need to find a victim to replace.

| Referenced Page | | 1 | 2 | 3 | 4 | 4 | 3 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 | 7 | 8 | 3 | 2 | 1 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame-1 | | | | | | | | | | | | | | | | | | | | | |
| Frame-2 | | | | | | | | | | | | | | | | | | | | | |
| Frame-3 | | | | | | | | | | | | | | | | | | | | | |
| Frame-4 | | | | | | | | | | | | | | | | | | | | | |

**# of page faults:** **# of page hits:** **# of page replacements:**

# Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from University of Nevada, Reno
- Farshad Ghanei from Illinois Tech
- T. Kosar and K. Dantu from University at Buffalo

# Announcement

- Quiz 4
    - Will be released tomorrow
    - Blackboard
- Homework 3
    - Due on Monday, 27th at 11.59PM