Week 8: Virtual Memory

Background

- Memory allocation techniques discussed so far, the general assumption was ➔ entire program or its entire executable parts must be in the memory before CPU can execute them
- However, entire program code may not be needed at the same time, which are rarely used
  - Error code, unusual routines, large data structures
- Why not consider the ability to execute partially-loaded program?

Virtual Memory ?

- Abstraction of main memory and hard-disk into massive storage
  - Illusion of unlimited memory
  - Enables programs to execute without completely loading in memory
    - Programs no longer constrained by physical memory size
  - Separates logical memory from physical memory
    - Programs with larger logical memory spaces can execute
    - Increased throughput and degree of multiprogramming

- Very high-level view:
  - Load only needed parts of program in memory for immediate and near-future execution
  - Continuously load new pages and replace existing ones in memory to only have what is needed ⇒ **Demand Paging**

Demand Paging ?

- Load needed pages on demand
  - Pure demand paging: Start/resume with no page in memory
    - Pages loaded when needed
    - Pages removed to load necessary ones
  - Similar to paging with swapping
- Two types of pages:
  - In memory and in secondary storage
  - Need to distinguish between them in page table
  - Use Variant of Valid-Invalid bits
    - Valid: Both valid as address and in memory
    - Invalid: Otherwise (either not valid as address or currently in the secondary storage)

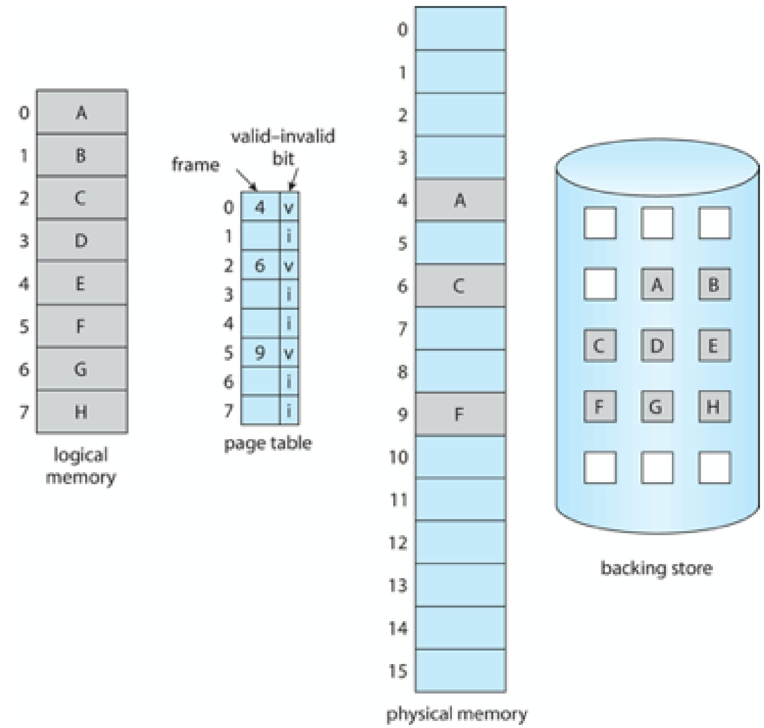
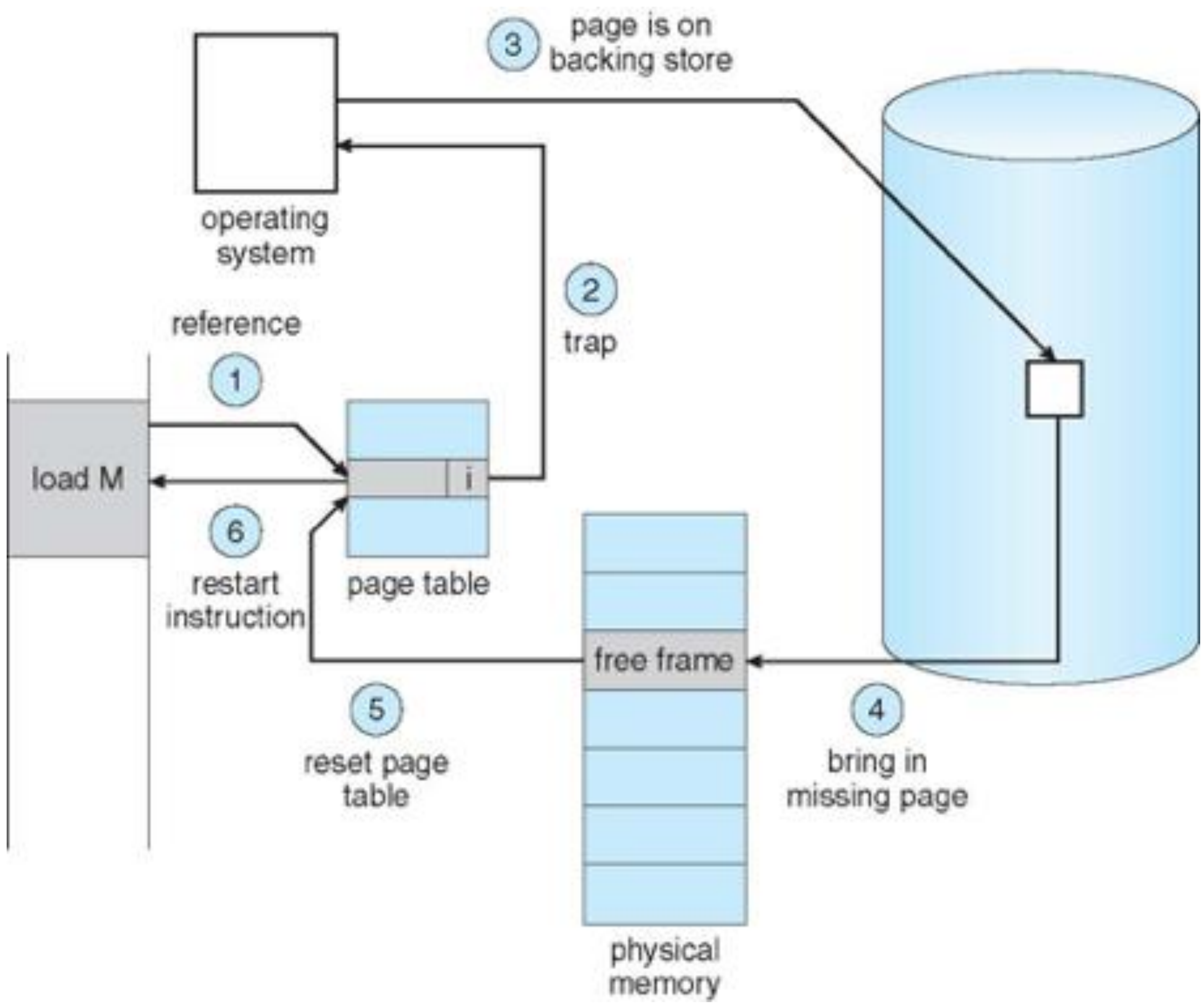


Fig. Page table when some pages are not in main memory

Page Fault

- Recall: CPU trying to access page not in memory
  - Page found with invalid bit at address translation
  - Trap to OS
- Handler: Check table in PCB to find whether:
  - Address is not part of process logical space (really invalid):
    - Terminate process
  - Address part of logical address ⇒ but process not in memory
    - If free frame exists in memory ⇒ Load page
    - If No free frames exist ⇒ Swap page with victim frame
    - Page table updated; valid-invalid bit set to valid
    - Access re-attempted

Page Fault



3 page is on backing store
operating system
2 trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory

Performance of Demand Paging (Cont.)

- Three major activities
  - Service the interrupt – careful coding means just several hundred instructions needed
  - Read the page – lots of time
  - Restart the process – again just a small amount of time
- Page Fault Rate $0 \le p \le 1$
  - if $p = 0$, no page faults
  - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)
- EAT = $(1 - p)$ x memory access + p x (page fault overhead + swap page out + swap page in)
  - Memory access time is in ns (usually 10 to 200 ns)
  - If the probability of page fault is high, EAT will increases
  - EAT is proportional to page fault rate

Probability if there is no page fault

Probability of page fault

- EAT = only memory access time, when no page fault

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p) x 200 + p x 8,000,000 (in ns)

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then

  EAT = 8199.8 ns = 8.2 microseconds (where, p = 0.001).

Page Replacement

- Mechanism and selection criteria of pages to swap out and replace them with demanded pages
  - Involves the transfer of two pages, one out and another in
  - Swapped in page already determined and must use I/O
  - Swapped out page must be selected smartly to reduce two overheads
    - Selection Overhead- which one to swap out
    - Immediate I/O Overhead

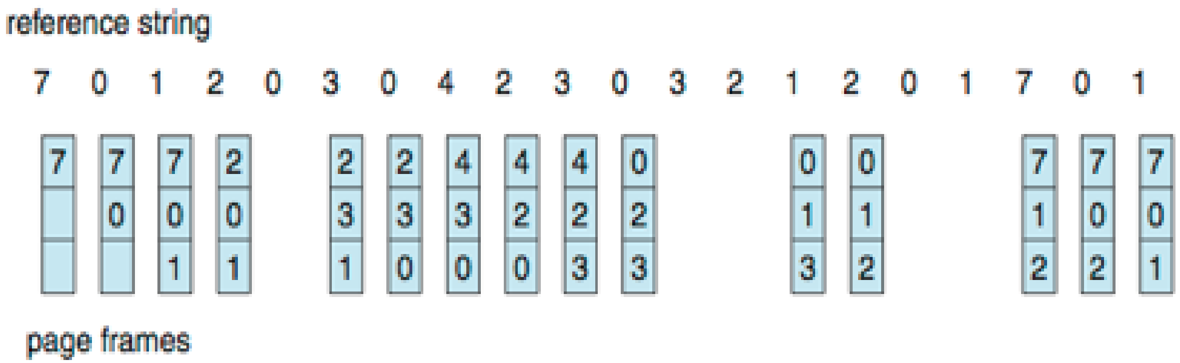
- Immediate I/O Overhead- to write the page in the secondary storage
  - I/O not needed if page not modified since loaded or last update
  - Choosing those for swap-out reduces I/O overhead
  - Done based on Modify/Dirty Bits:
    - One bit associated per each page and frame in memory
    - Set only if at least one byte in page is modified since loaded or updated

Page Selection for Swap-Out

- Overhead
  - Selection **procedure**: Result in information and algorithmic overhead
  - Future I/O overhead if "wrong" page swapped out
    - "Wrong" page: If page currently/frequently referenced/demanded is selected
    - Must be reloaded shortly if swapped-out ⇒ Avoidable expensive I/O
- Replacement Algorithms
  - First-In First-Out (FIFO)
  - Optimal (OPT)
  - Least Recently Used (LRU)
  - Counting-based (CB)
- Reference String: Used to test replacement algorithms
  - E.g., **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

FIFO Replacement

- Simplest
  - When a page is brought into memory, it is inserted at the tail of queue
  - Associate entry time to page ⇒ when a page must be replaced, replace the oldest page (page at the head of the queue)
- Example



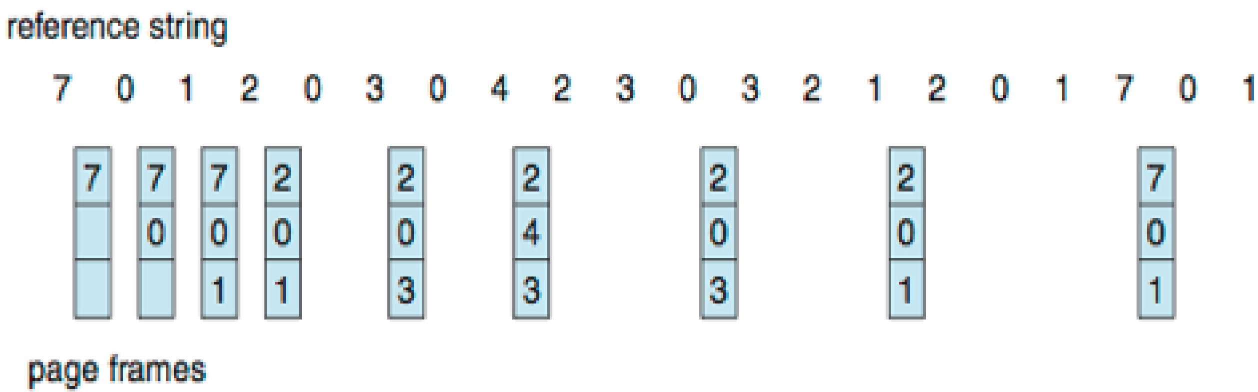
15 Page Faults

- Pros: Simple, easy to understand and program
- Drawback: Selection does not use any info on the usage of the paper or future page faults ⇒ Performance unpredictable

Optimal Replacement

- Optimality Criterion and Method
  - Lowest page-fault rate of all algorithms
  - Replace the page that **will not be used** for the longest period (future)

- Example



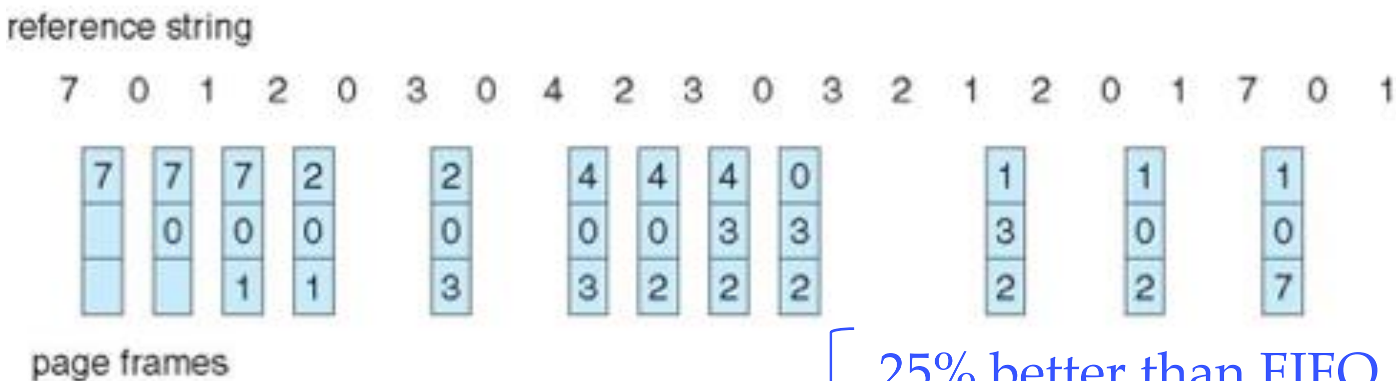
9 Page Faults

- Pros: Well Optimal!
- Drawback: Very difficult to implement ⇒ Anti-causal

(requires knowledge of quite a good number of future references in the string)

LRU -Least recently used Replacement

- Approximation of optimal
  - Use recent past to anticipate near future
  - Replace page that has not be used for the longest period (prior)
- Example



12 Page Faults — 25% better than FIFO, 50% worse than OPT

- Implementation of LRU
  - Counters: Incremented per references, appended to referenced page
  - Stack: Pulls referenced page to stack top. Least used at stack bottom

Counting-Based Replacement

- Count number of reference per page
- Replace:
  - Least Frequently Used (LFU) Page
    - Replaces page with smallest count
    - Does not factor time of usage into selection
      - E.g., Page with high initial count but no recent activities; Won't be replaced
      - Solution: Shift counts right by 1 every interval
  - Most Frequently Used (MFU) Page
    - Replaces page with largest count
    - Pages with smallest count just swapped in ⇒ Yet to be referenced more
    - May again be totally inaccurate
- Both expensive and do not approximate OPT