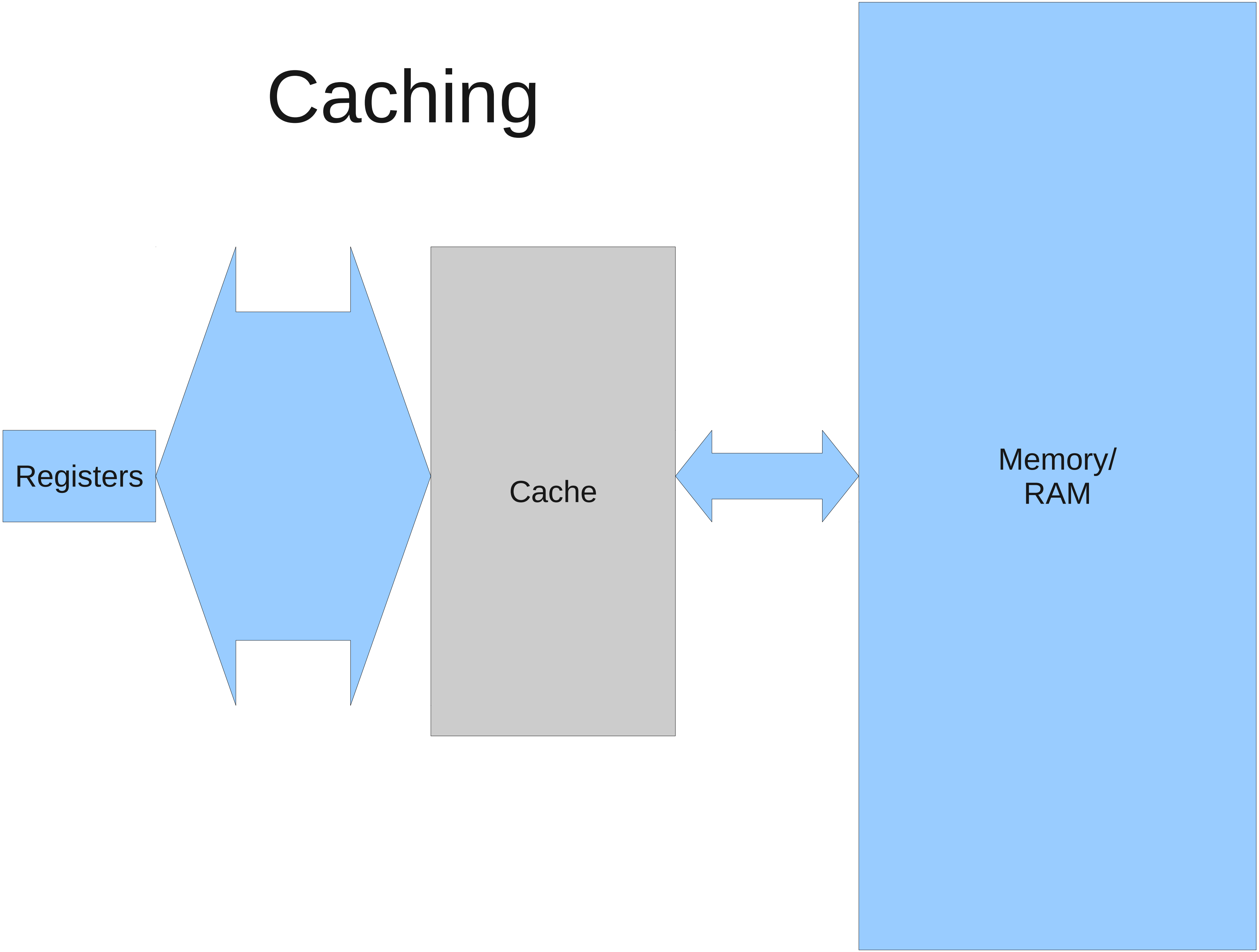
csci 3411: Operating Systems

Virtual Memory II

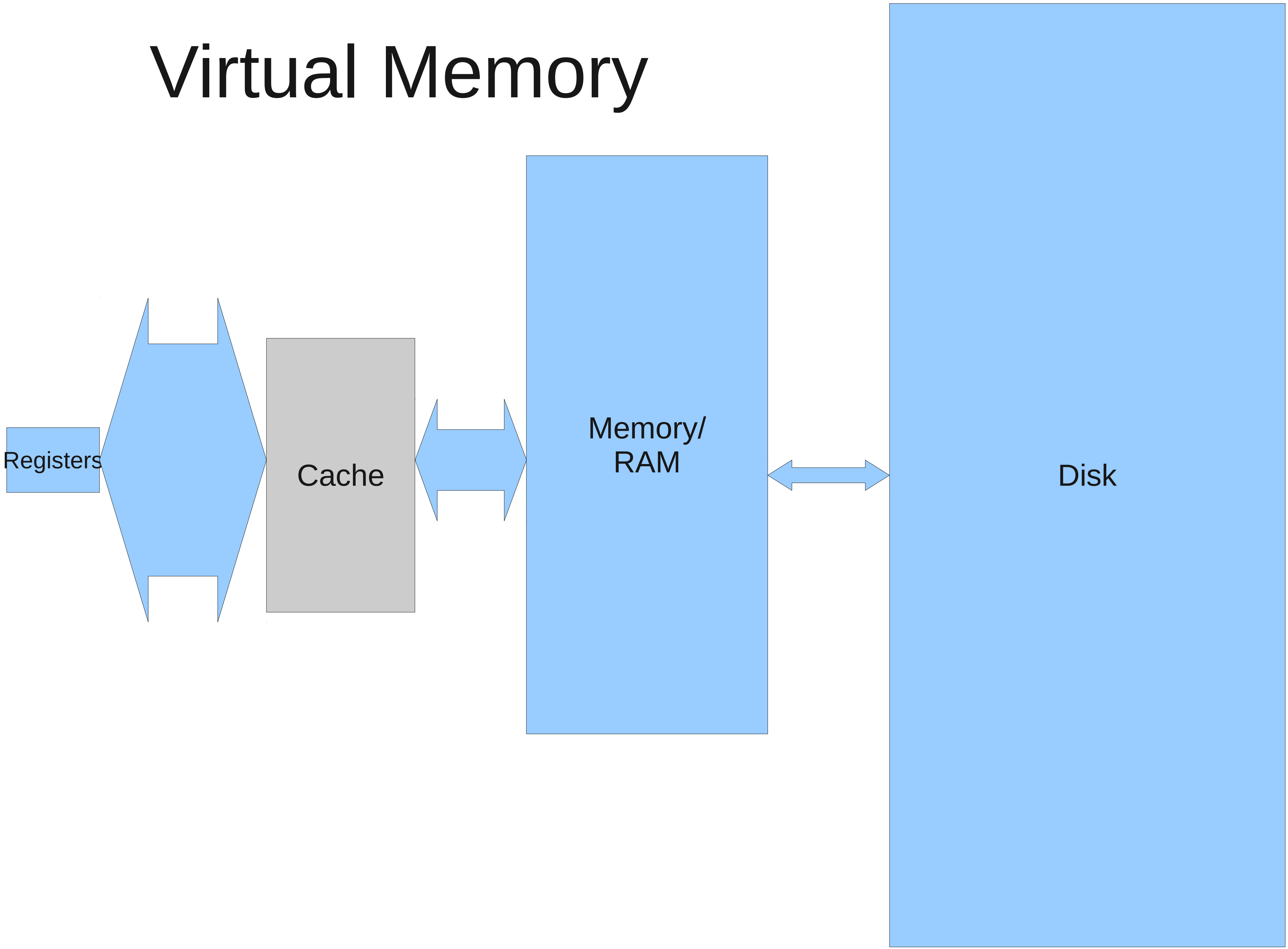
Gabriel Parmer

Slides adapted from Silberschatz and West

Caching



Virtual Memory



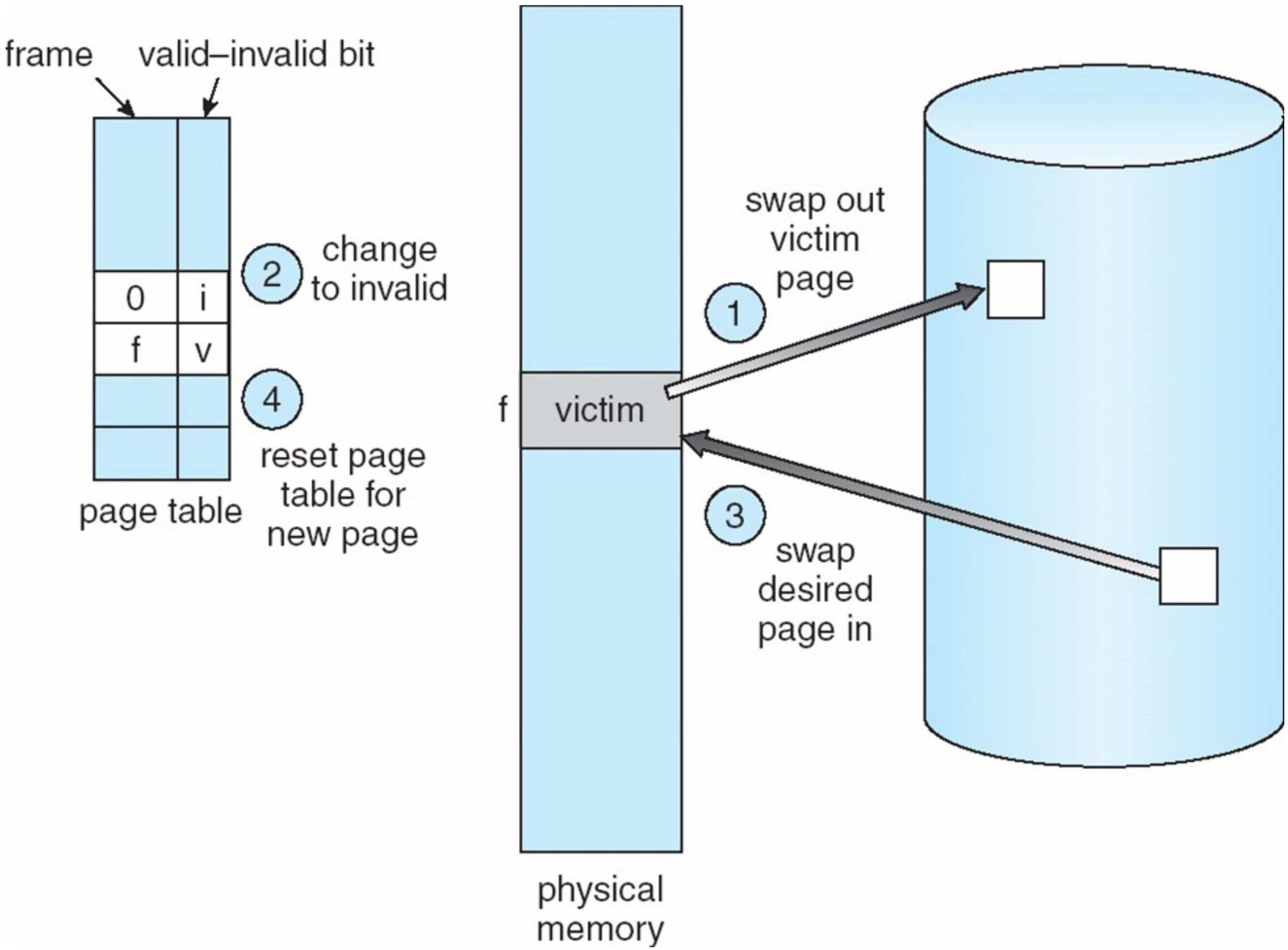
Page Replacement

- Use memory as a cache for disk

- Page replacement
  - Find a *victim* frame in memory
  - *Swap* it out – transfer it to disk to free that memory for other uses

- Swapping is the act of moving active memory back and forth from disk
  - Also called *paging* in page-based systems

Page Replacement III



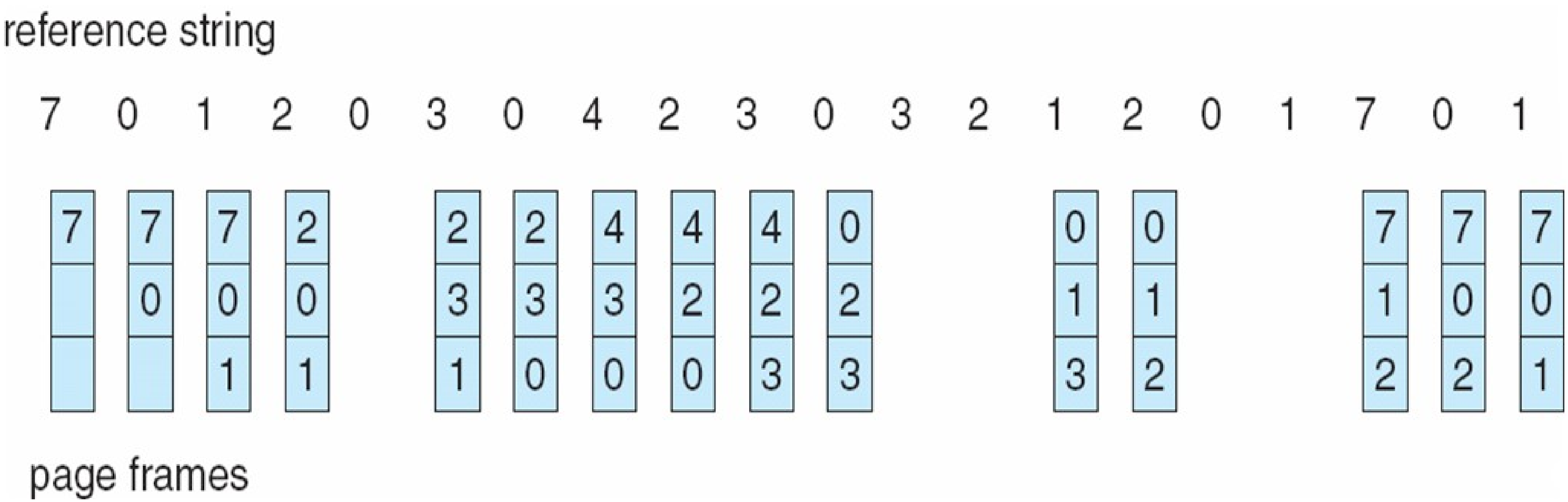
How do we Choose a Victim Frame?

- Going to disk is *expensive*
  - Want to swap as infrequently as possible
- Find frame that is *least likely* to be referenced in the near future
- Optimization: consider frames that already exists on disk, and haven't been modified in RAM!
  - How is this more efficient?
  - Page tables include *modified* bit (set by hardware when a store is made to the page)
- Today: Algorithms for finding victim frame
  - Questions so far?

Page Replacement Algorithms

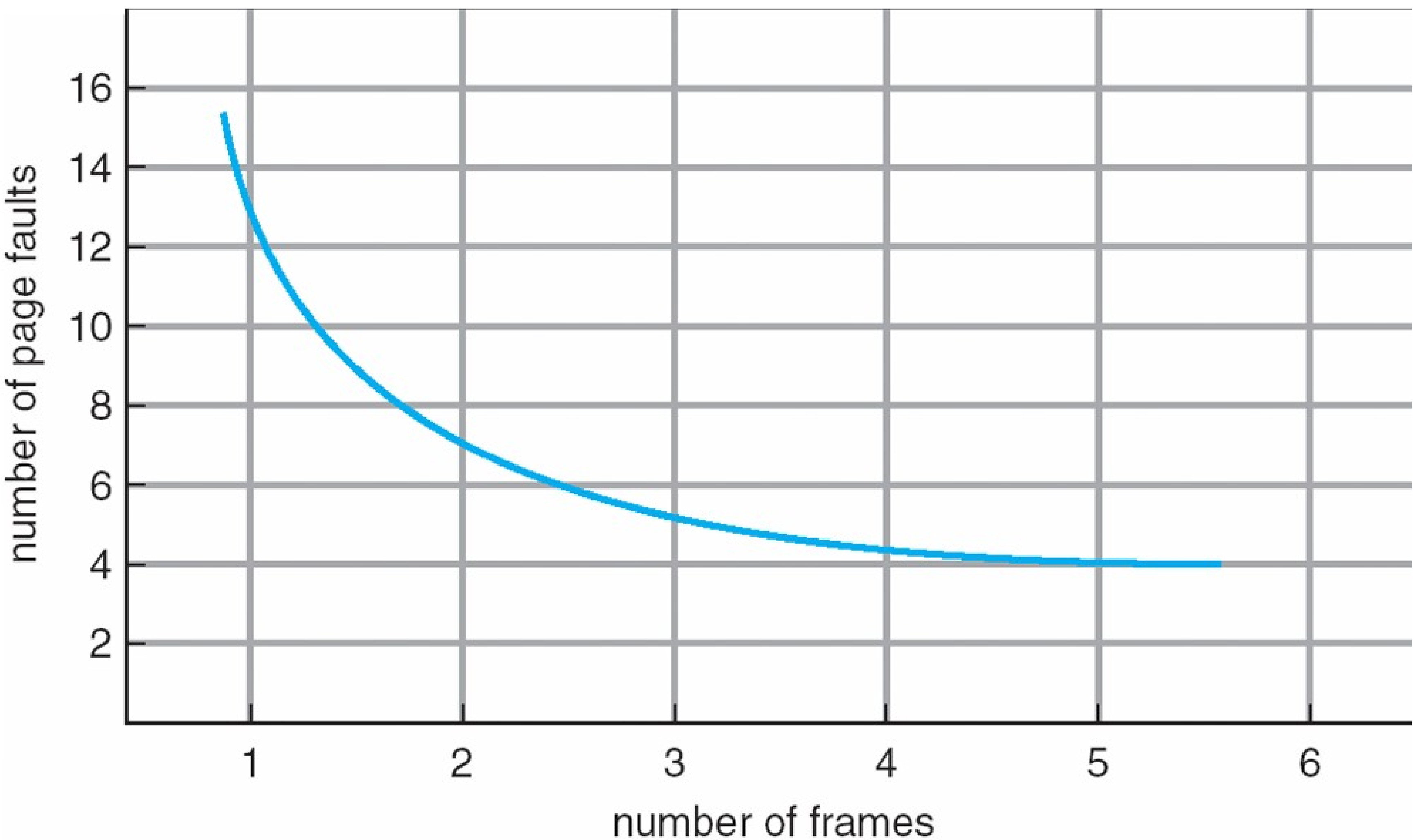
- Goal: lowest page fault rate
  - Remember – 1/1000 memory accesses going to disk → 40x slower EAT
- Evaluate algorithms for a given string of references made by program execution
  - For simplicity, memory references will refer to page numbers of virtual address
- We need an algorithm to determine the frame to page out to disk
  - *Any algorithms come to mind???*

FIFO Page Replacement



- *Are the page fault rate and number of frames available correlated? In what way?*

Page Faults Versus Number of Frames



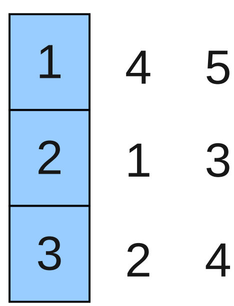
FIFO

- Memory/page reference sequence:
  - *1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5*

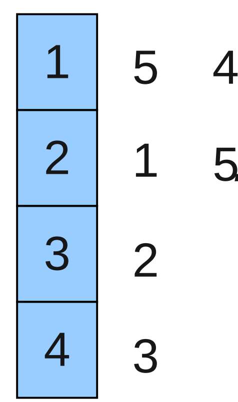
- 3 Frames?
- 4 Frames?

FIFO

- Memory/page reference sequence:
  - *1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5*

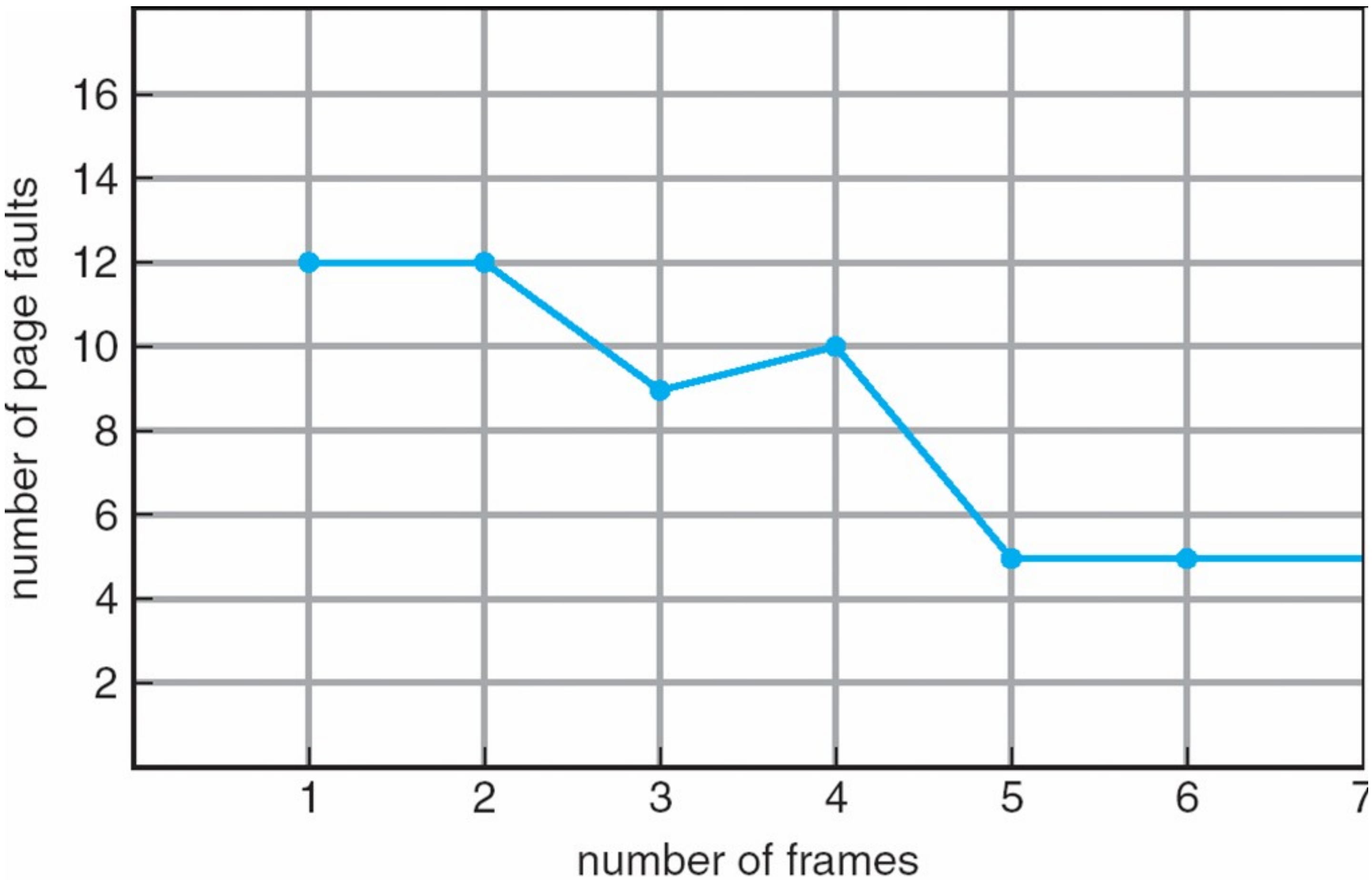


9 page faults

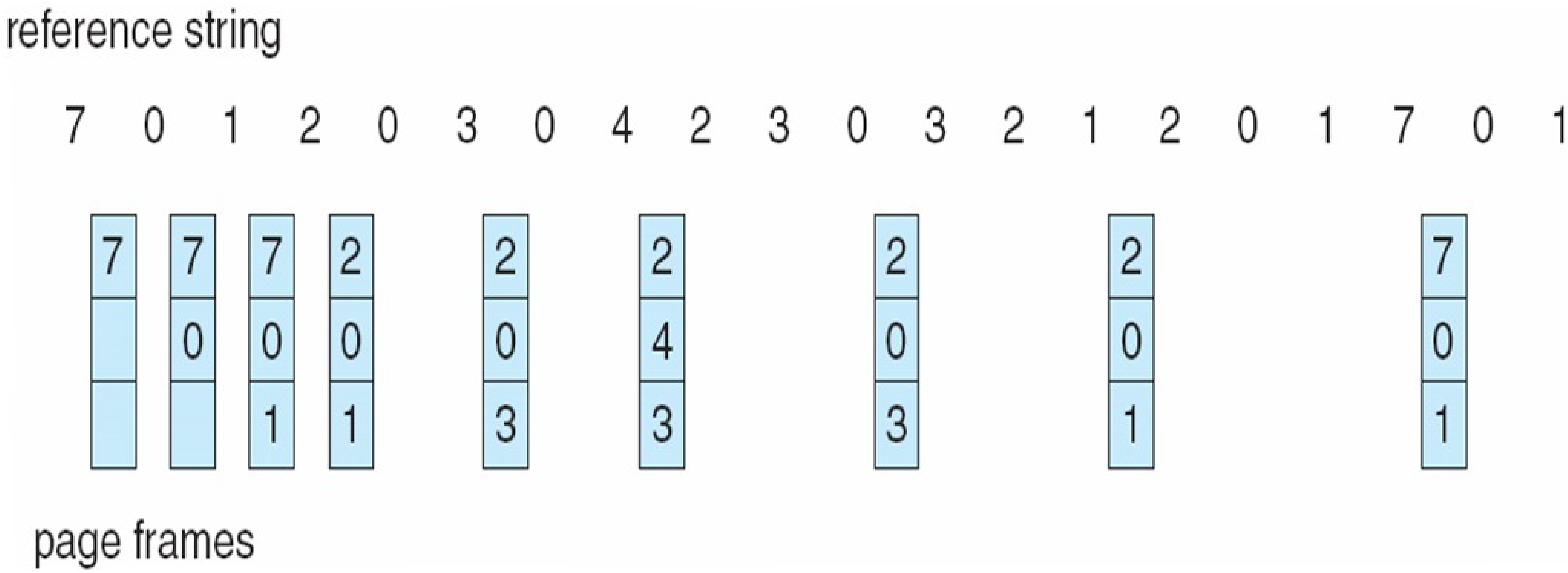


10 page faults

Belady's Anomaly



Optimal Page Replacement

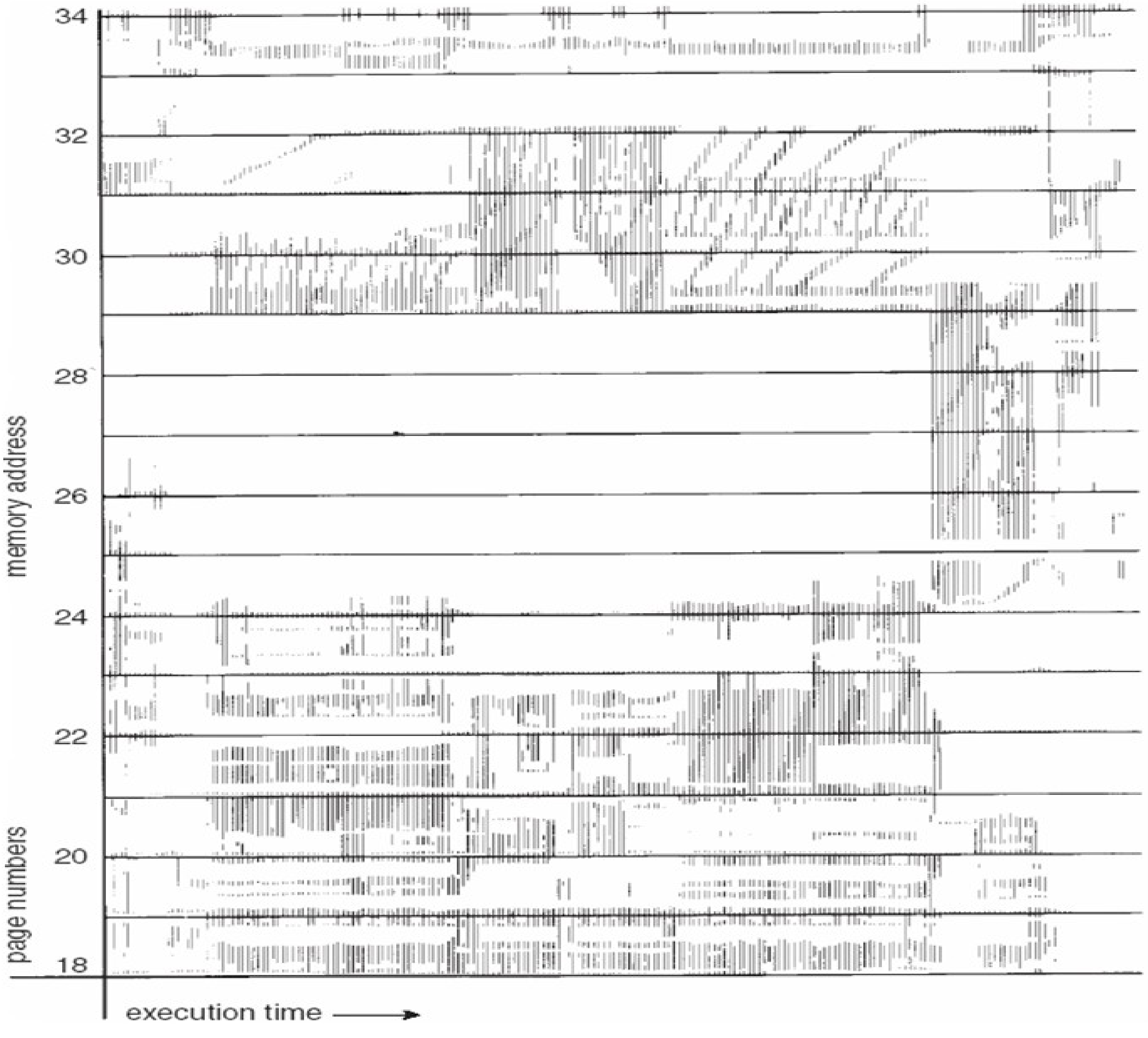


- Replace frame whose next reference is *furthest* in the *future*
- FIFO yields 15 page faults vs. 9 for optimal
  - Lets do better! *Ideas*?

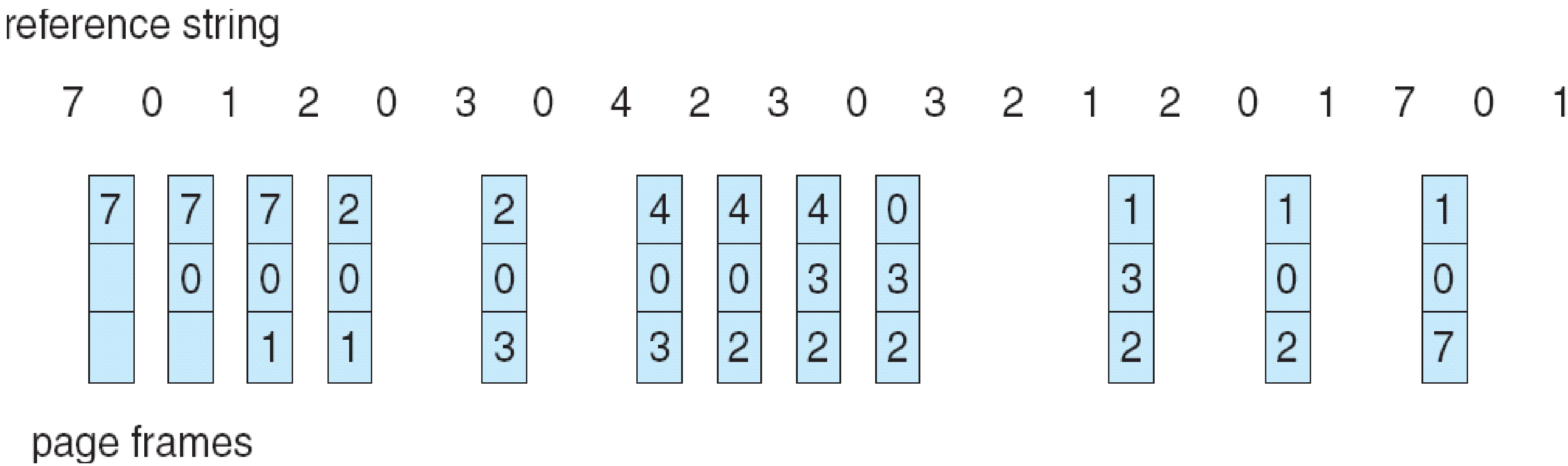
Caching Based on Historical Behavior

- Use past behavior to predict future
- Fundamental assumptions for effective caching
  - Spacial Locality
  - Temporal Locality
- Why does FIFO perform badly?

Memory Reference Trace



LRU Page Replacement



- Least Recently Used – evict the frame that was accessed furthest into the past
  - Takes advantage of temporal locality
  - Uses past to predict future

LRU Page Replacement II

- Reference string:
  - 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

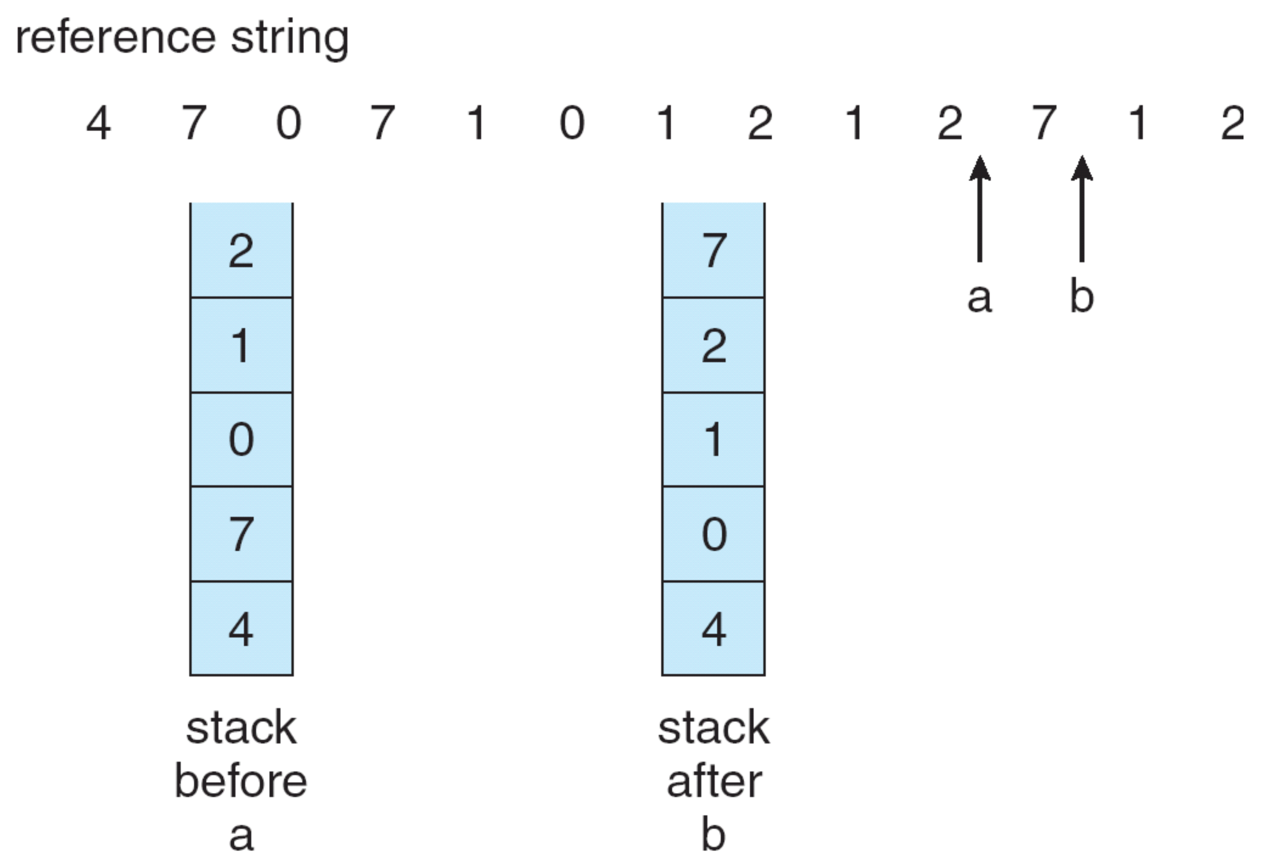
LRU Implementation I

- Use system-wide counter
  - Each memory reference increment counter
  - Each page has a *time of use* field
    - *Time of use* field updated with counter value on access
- *Which page should we replace?? Which is least recently used?*

- *Problems/limitations with this approach?*

LRU Implementation II

- Using a linked list to track references
  - Upon reference, frame placed at top of list
  - *Which is the least recently used frame?*
- *Problems/limitations with this approach?*



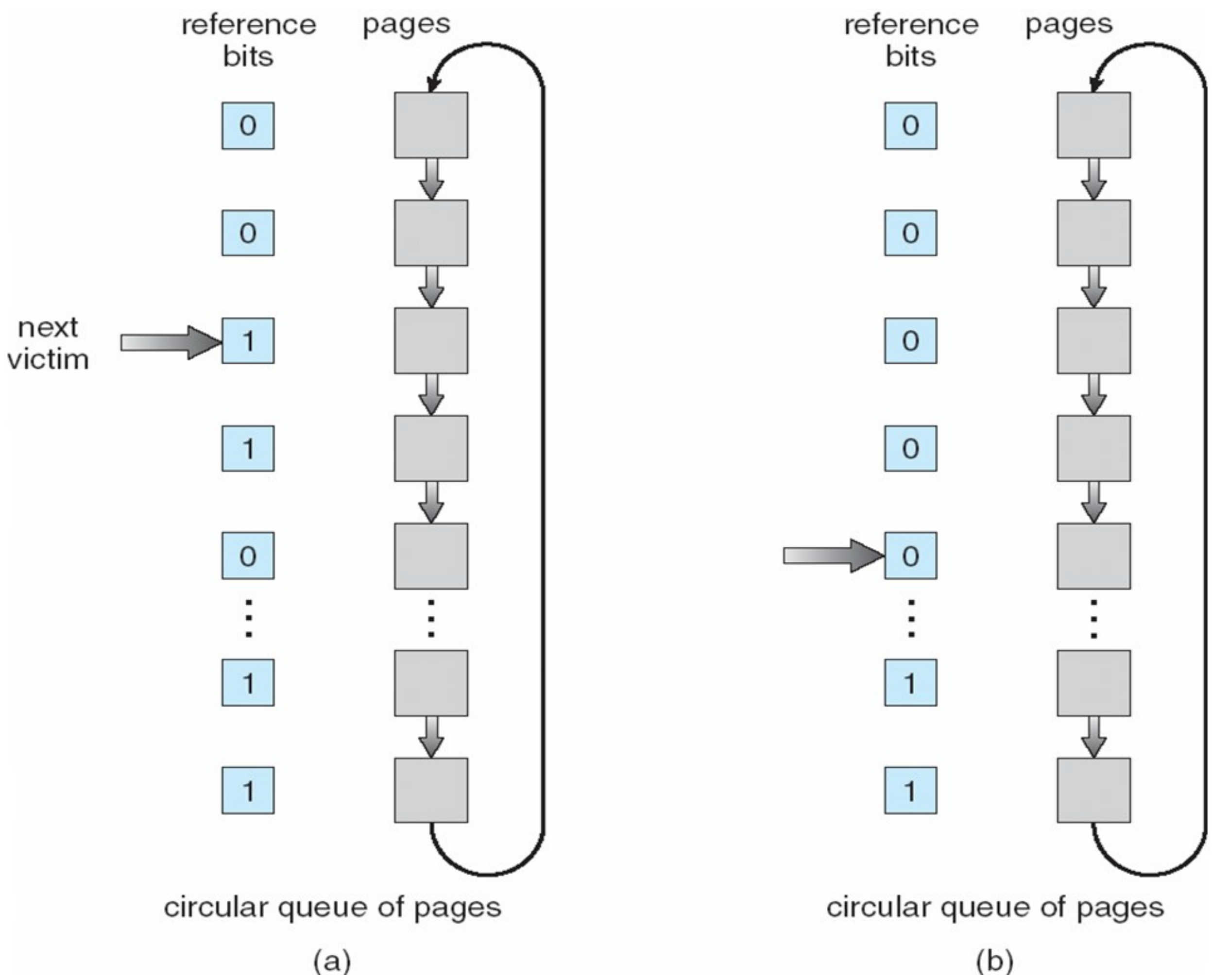
LRU Approximation

- Previous approaches require hardware assistance
  - Memory references in the many millions/second so hardware must manage list/counter
- Limited hardware support requires approximation
  - Page tables include *reference* bit
    - Set by hardware when page is referenced (like *modified)*
    - Basic idea: replace pages that haven't been referenced
    - Loses ordering information

Clock (Second Chance) Algorithm

- Similar to FIFO, but considers *referenced* bit for each page
  - If bit == 0, replace page
  - If bit == 1, clear bit and check next page
    - Give page a second chance to stay in memory
- Worst case: all pages are referenced
  - Cycle through all pages
- Closest to *not frequently used* (NFU)

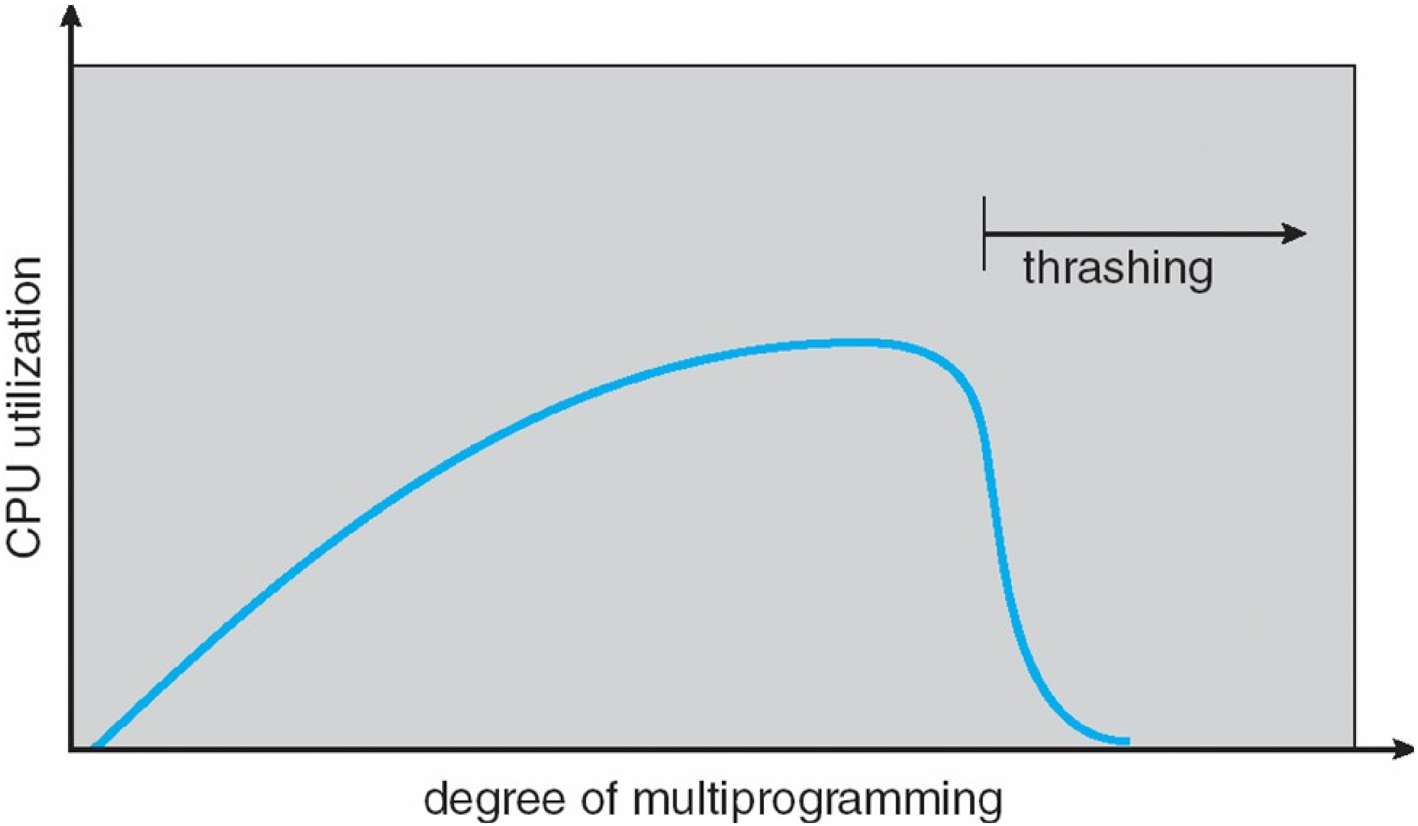
Clock Page Replacement Algorithm



Thrashing I

- When paging activity overwhelms normal execution
- Suppose process includes 2 pages: P1, P2
  - Access to P2 brings it into memory and swaps P1 to disk
  - What happens if P1 is access shortly thereafter?

Thrashing II



Thrashing III

- Decrease degree of Multiprogramming
- *Should one process that uses many pages cause all processes to slow down due to thrashing?*
  - *Are there other options?*

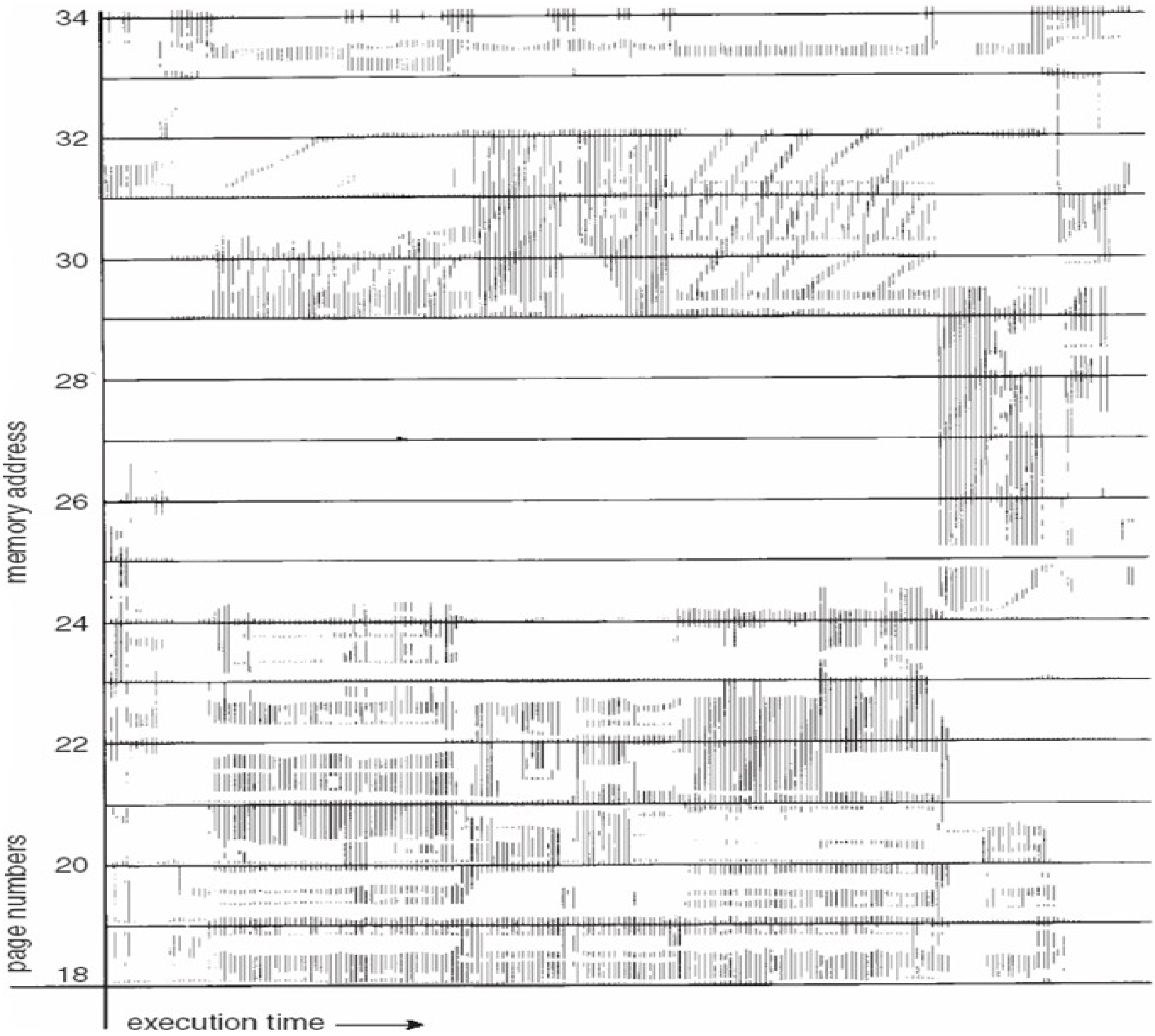
Global vs. Local Replacement

- Global replacement
  - Page replacement algorithm considers all frames in the system for replacement
    - i.e. across all processes
- Local replacement
  - Consider only pages of the process that requires a page to be swapped in
  - Pages allocated separately to different processes
    - Priority based, proportional, ...
- *What are the Pros/Cons of each of these?*

Locality of Reference

- 90:10 rule
  - 90% of its time, a program executes 10% of its code
  - 90% of the data accesses are to 10% of the data-structure memory locations
  - Why?
- Implication:
  - We can achieve a good hit rate (low page fault rate) if we keep that 10% of pages in memory

Memory Reference Trace



Working Set

- A processes working set is the set of most actively referenced pages at a given point in time
  - Can change
  - Subset of total memory possibly requested by process
- Thrashing: $\sum_{\text{forall } i} WSS_i > M$
  - M = total memory in system
  - WSS_i = working set size for process i

Working Set II

- OS monitors working set of each process
  - Allocate enough frames to fit the working set
- If there are spare frames
  - New processes can begin
- If OS cannot accommodate WSS of all processes
  - Suspend process to disk

Monitoring WSS

- How does the OS monitor the WSS of each process? (Esp. since WSS varies with time!)
  - Allocate frames to process if high page fault freq.
  - Take frames from process if low page fault freq.