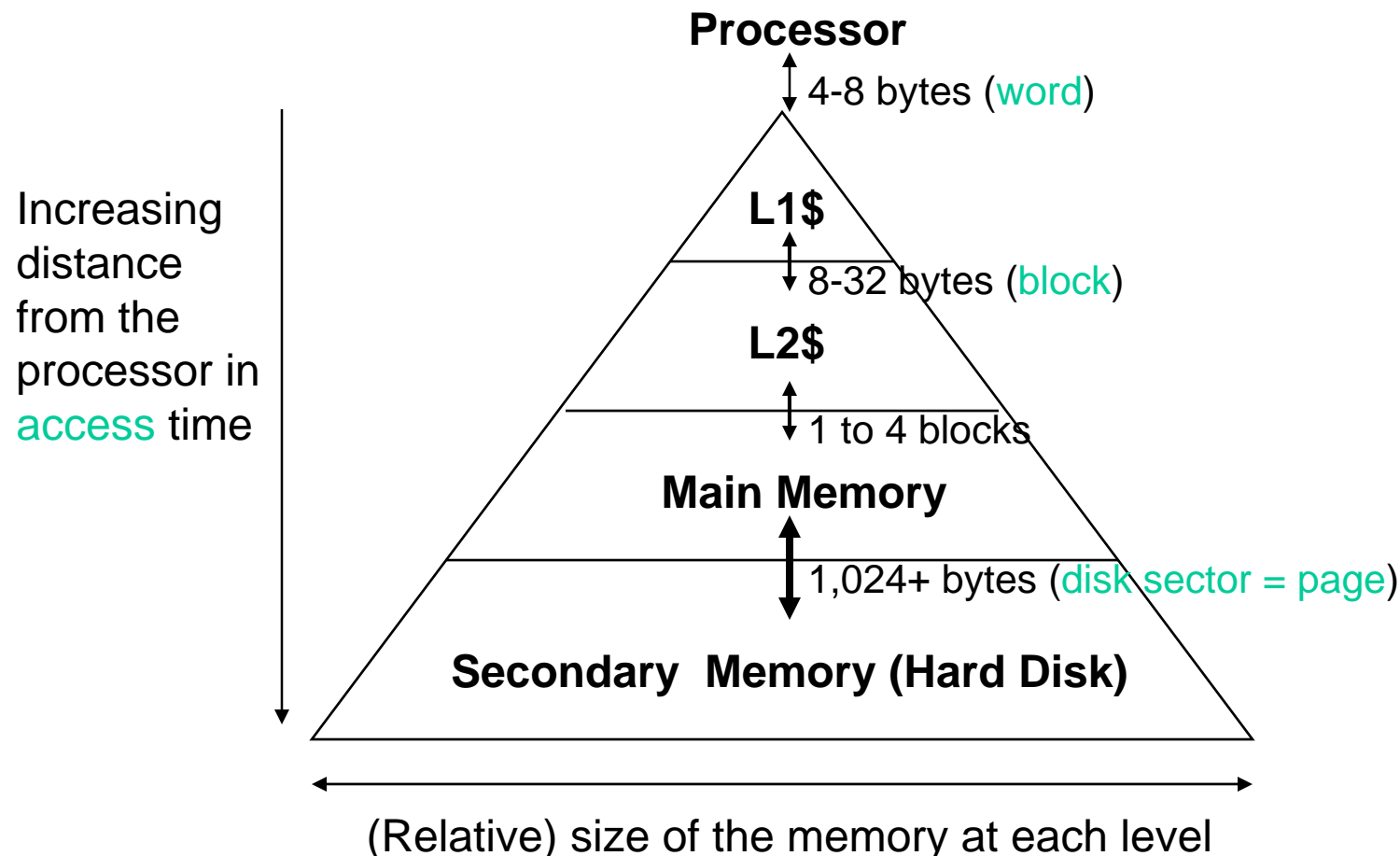


ECE7995 Caching and Prefetching Techniques in Computer Systems

Lecture 8: Buffer Cache in Main Memory (I)

Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology



Management of Buffer Cache in Main Memory

- Less constraints on buffer cache in main memory (managed by software)
 - Allow fully-associative mapping
 - ✓ Any disk blocks can be placed at any memory block address in a buffer cache
 - Allow complex data structure
 - ✓ Processor cache: address manipulation for indices to tables
 - ✓ Buffer cache: use advanced data structure such as hash table and radix tree to search blocks/pages of a give address.
 - Allow complex algorithm for replacement decision
 - ✓ Processor cache: LRU for 2-way, random or approximate LRU for higher ways
 - ✓ Buffer cache: LRU and other more advanced replacement algorithms.
- The objective of buffer cache management
 - Leverage intelligent replacement algorithms to minimize I/O time.

Basic Page Replacement

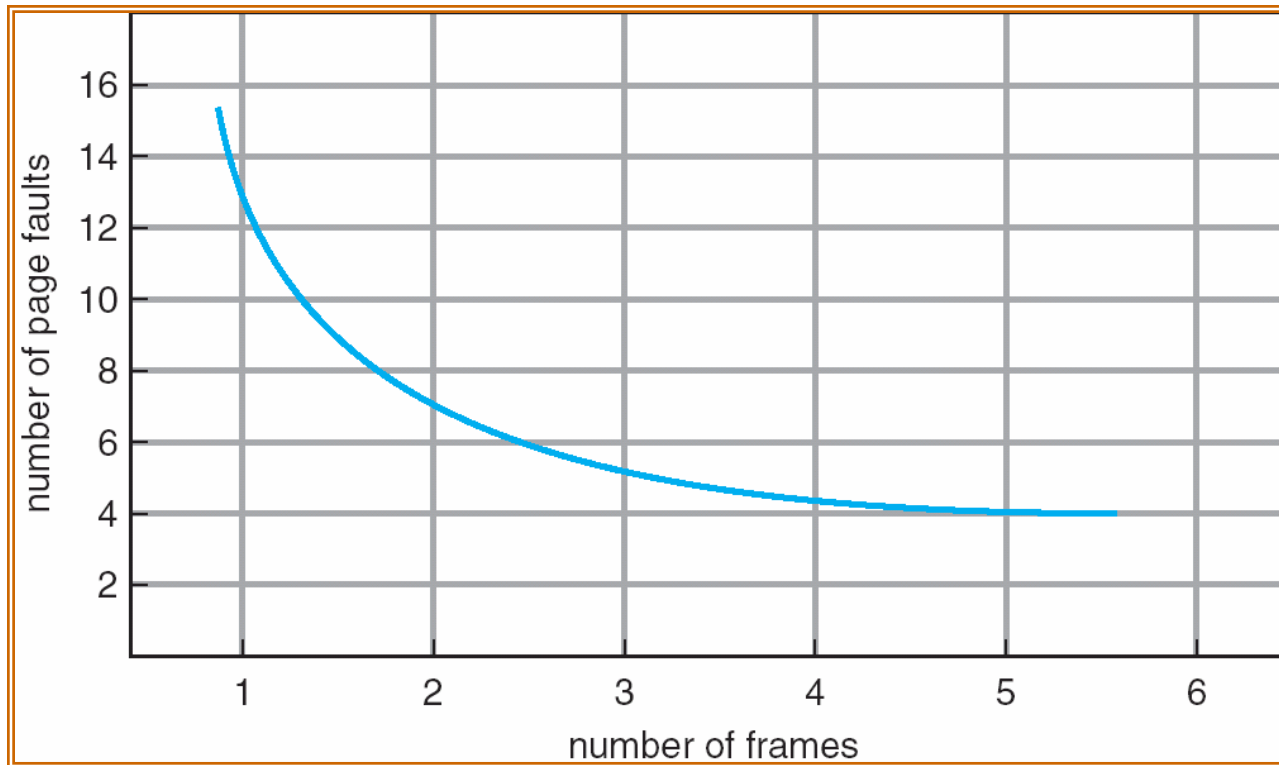
- 1) Find the location of the desired page on disk
- 2) Find a free frame:
 - If there is a free frame, use it
 - If there is no free frame, use its page replacement algorithm to select a victim frame (if the frame is dirty, it needs to be written back)
- 3) Bring the desired page into the (newly) free frame; update the page and frame tables
- 4) Restart the process

Page Replacement Algorithms

- One objective: lowest page-fault rate (or miss 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
- The reference string is like:

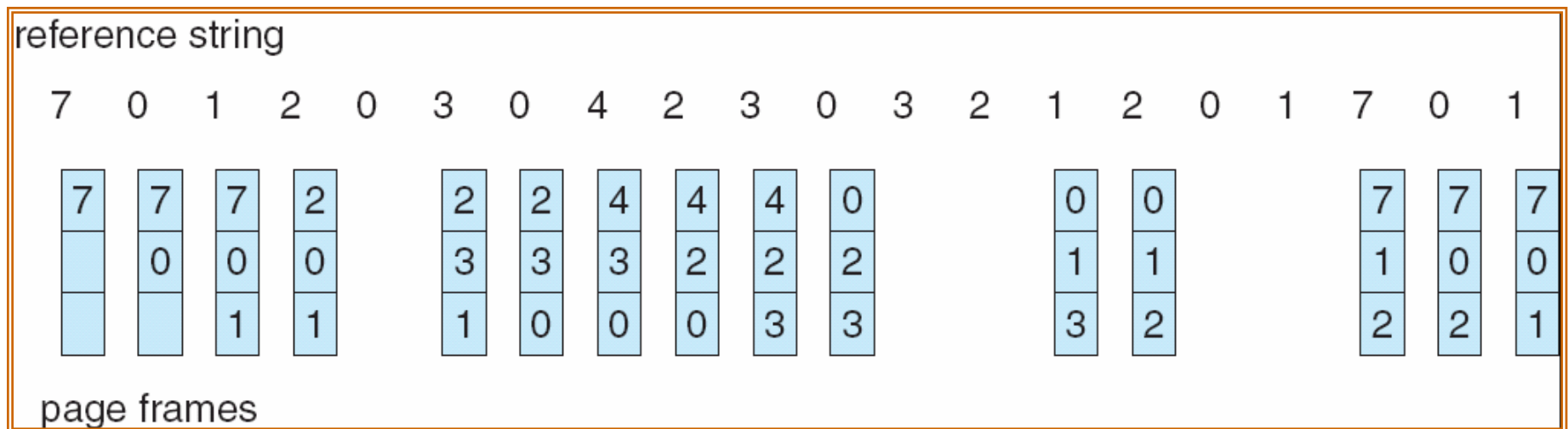
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Graph of Page Faults Versus The Number of Frames



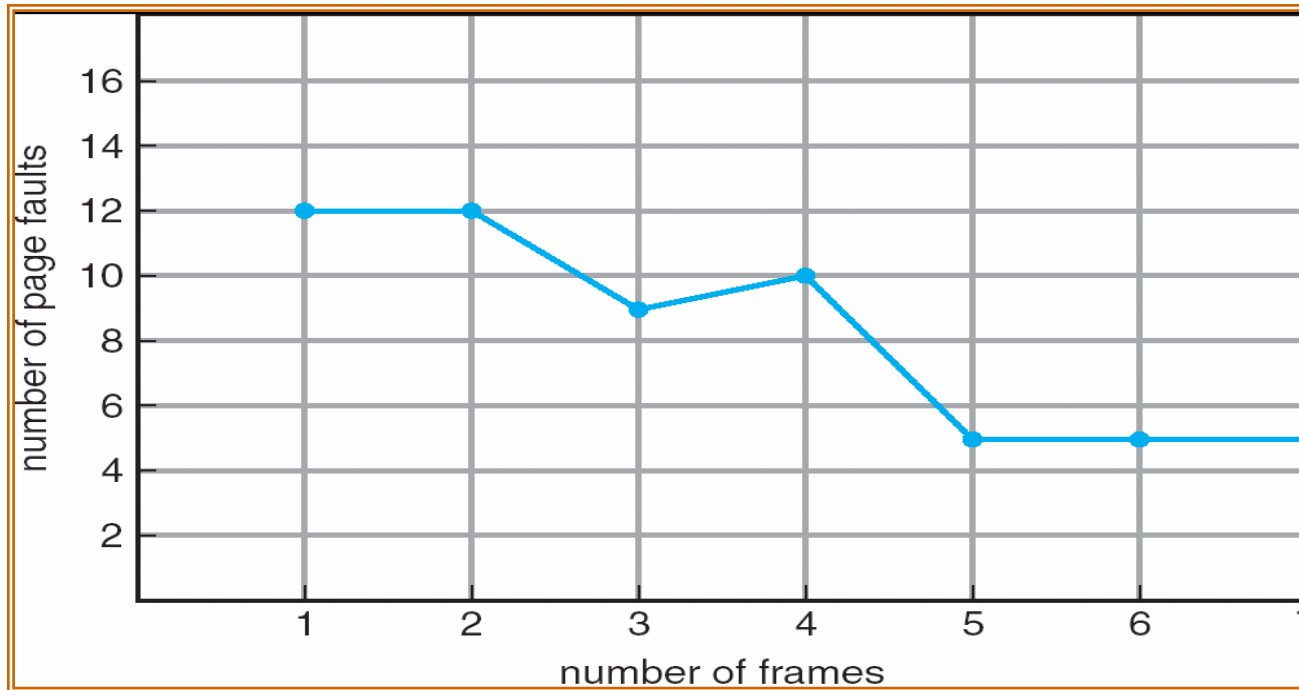
FIFO Page Replacement

- FIFO: Evict oldest page:
 - Problem: completely ignores usage pattern
 - first pages loaded are often frequently accessed



Belady's Anomaly

For some page replacement algorithms, the page fault rate may increase as the number of page frames increases.

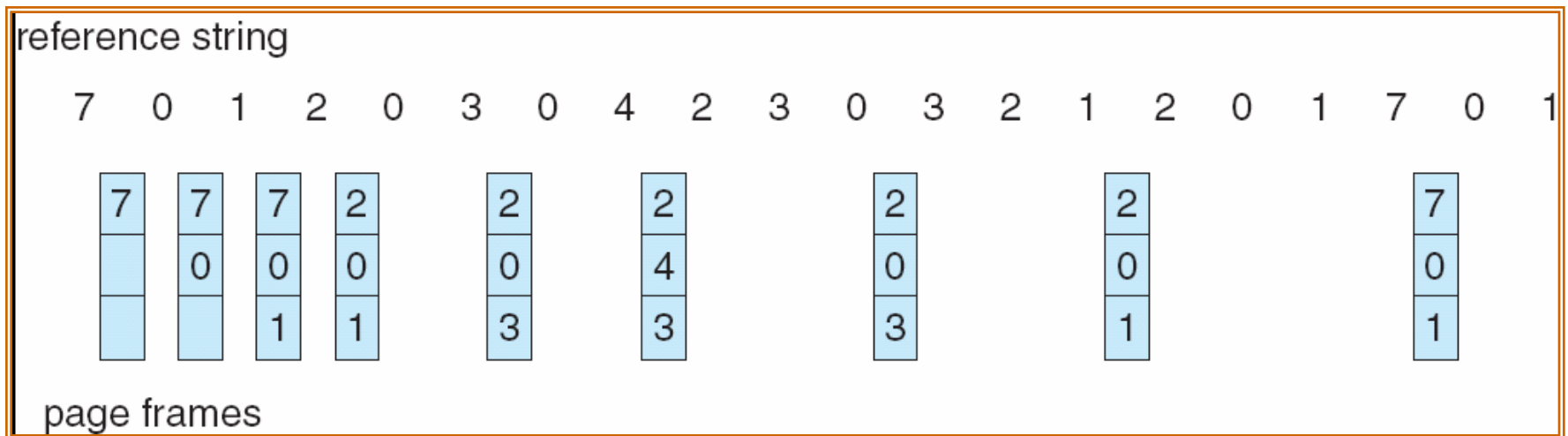


Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
with FIFO:

- (1) Using 3 page frames, there are 9 page faults
- (2) Using 4 page frames, there are 10 page faults

Optimal Algorithm

- The MIN algorithm: replace the page that is accessed the farthest in the future, e.g. that won't be accessed for the longest time
- Problem: don't know what are the future references
 - Used for measuring how well a practical algorithm performs

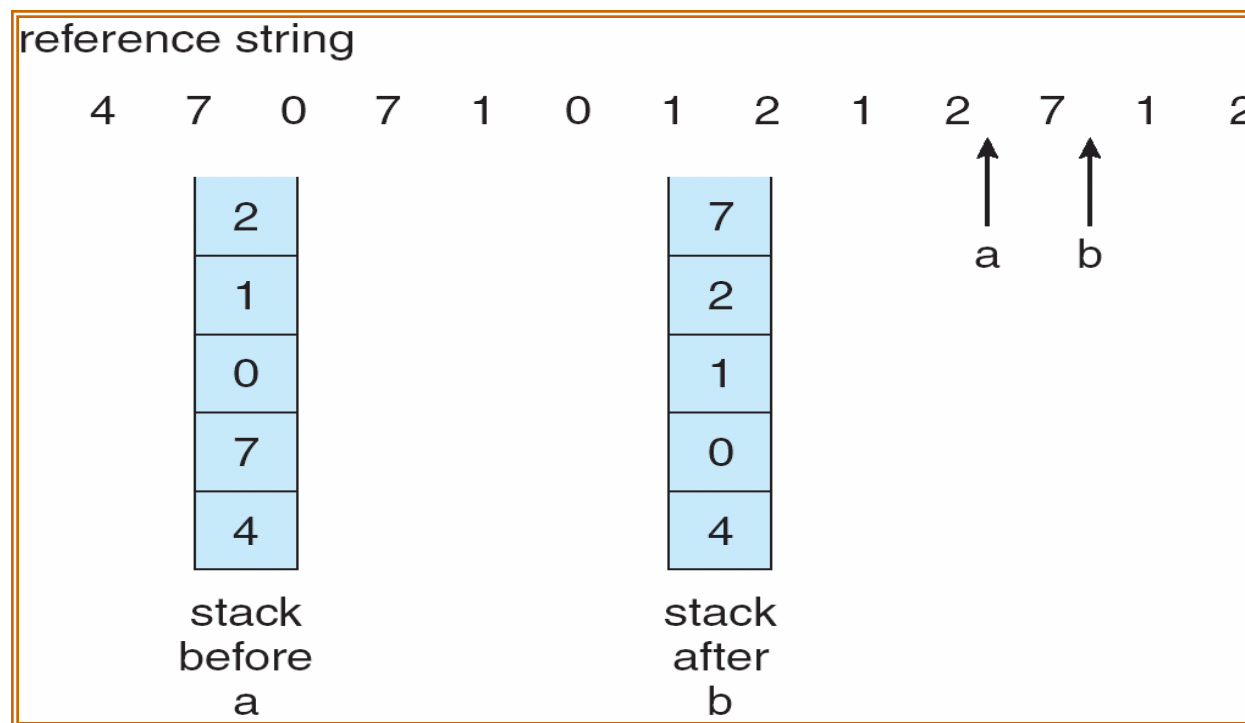


Least Recently Used (LRU) Algorithm

- LRU:
 - OS maintains ordered list of physical pages by reference time.
 - When page is referenced: move the page to front of list
 - When need victim: pick page at back of list (least-recently-used page)
 - Trade-off: slow on memory reference, fast on replacement
- LRU is designed according to the temporal locality
 - A page recently accessed is likely to be accessed again in the near future
- Good performance if past is predictive of future.
- Major problem: would have to keep track of “recency” on every access, either timestamp, or move to front of a list
 - OK with I/O access via sys-call
 - unaffordable with hits in process address space

LRU Algorithm Implementation

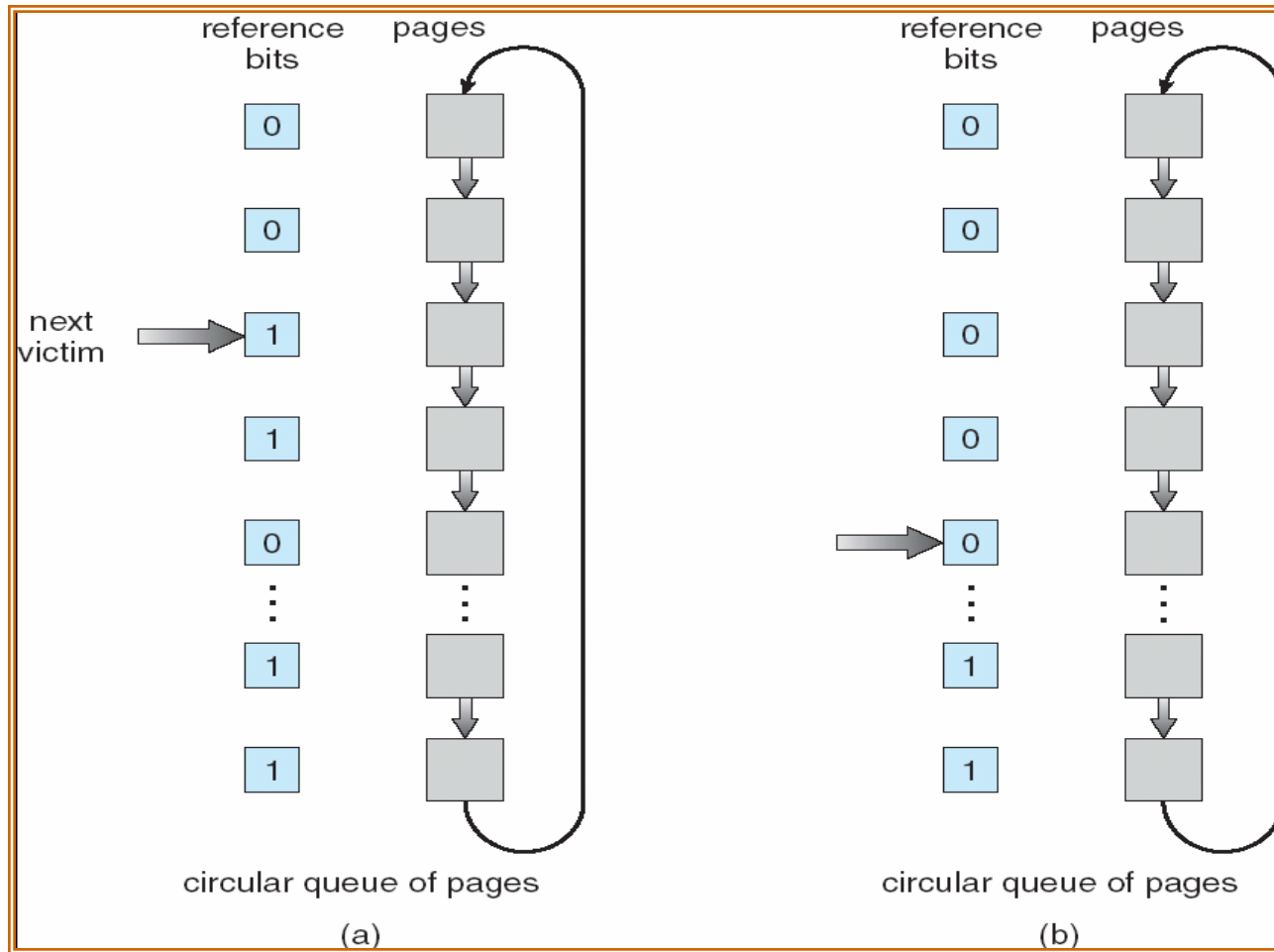
- Stack implementation – keep a stack of page numbers in a double link form:
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - No search for replacement
- Use of a stack to record the most recent page references



Clock Algorithm – an LRU Approximation

- Hardware support
 - Keep reference bit for each page frame
 - When page is referenced, set the bit
- Operating System
 - Page replacement: look for page with reference bit cleared (has not been referenced for a while)
 - Implementation:
 - ✓ Treat pages as circular buffer
 - ✓ Keep pointer to last examined frame
 - ✓ Traverse pages in circular buffer
 - ✓ Clear reference bits as search
 - ✓ Stop when find page with cleared reference bit, replace this page.

Illustration of the Clock Algorithm



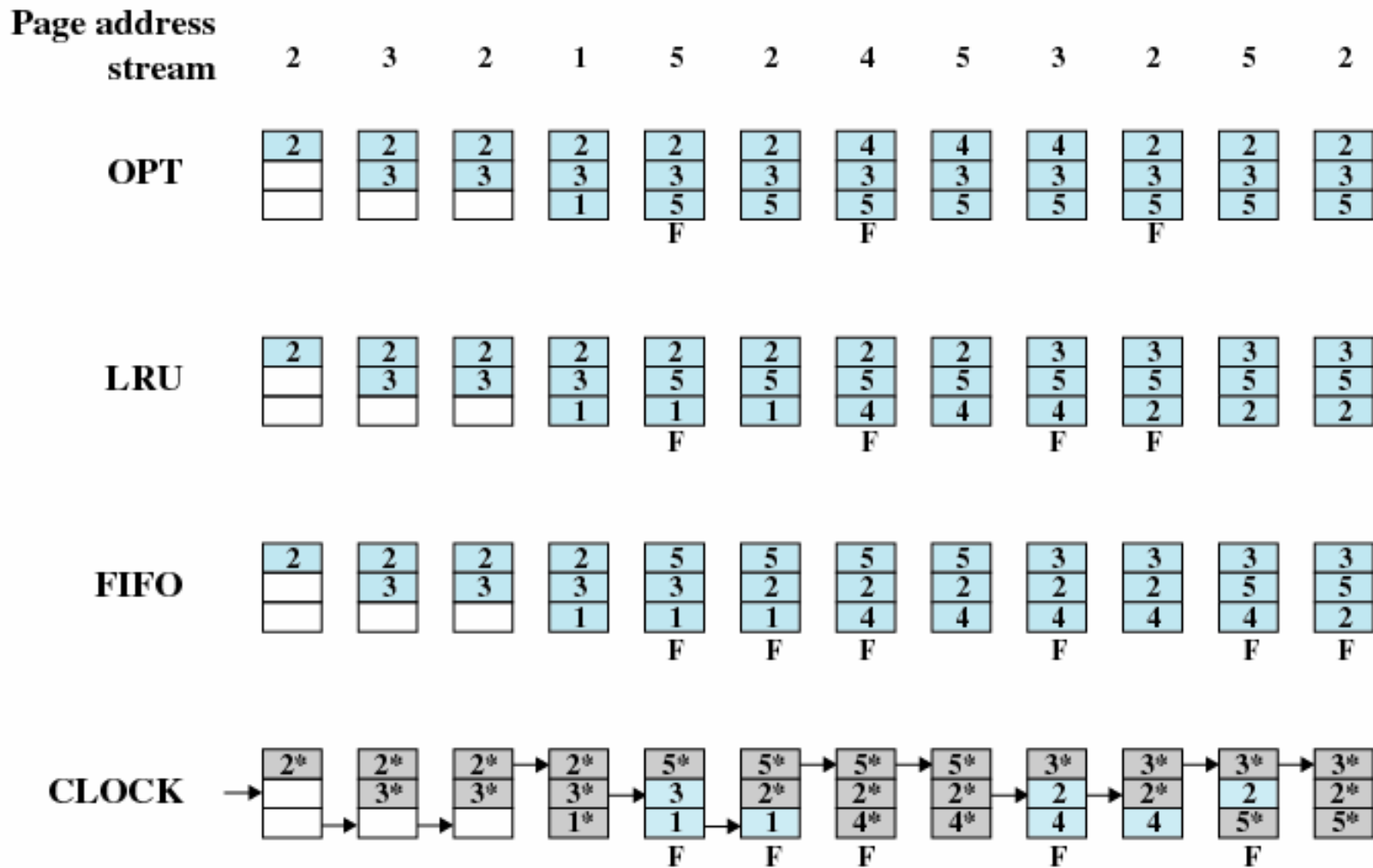
In what cases, the clock hand sweeps very slow? And very fast?

What are the issues with the cases?

Clock Extensions

- Replace multiple pages at once
 - Intuition: Expensive to run replacement algorithm to get a single page frame or to write single page to disk
 - Find multiple victims each time
- Two-handed clock
 - Intuition:
 - ✓ If takes long time for clock hand to sweep through pages, then all uses might be set
 - ✓ Traditional clock cannot differentiate between usage of different pages in this case.
 - Allow smaller time between clearing reference bit and testing
 - ✓ First hand: clears reference bit periodically (timing not determined solely by miss events)
 - ✓ Second hand: looks for victim page with reference bit still cleared.

Replacement Examples



F = page fault occurring after the frame allocation is initially filled

Other Replacement Algorithms

- **LFU Algorithm:** Keep a counter of the number of references that have been made to each page and replaces page with smallest count
- **MRU Algorithm:** replace most-recently-used page