# Memory Management
# Lecture 9

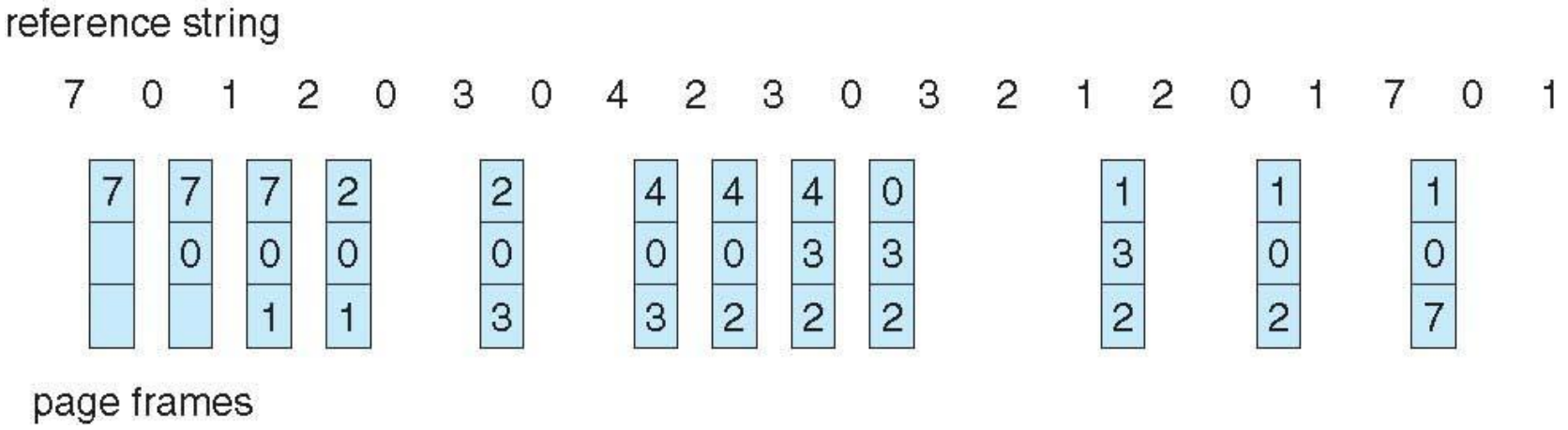
LRU Page Replacement Algorithm

Minakshi R.

**Operating System Concepts 8th edition silberschatz Galvin**

# Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page
- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

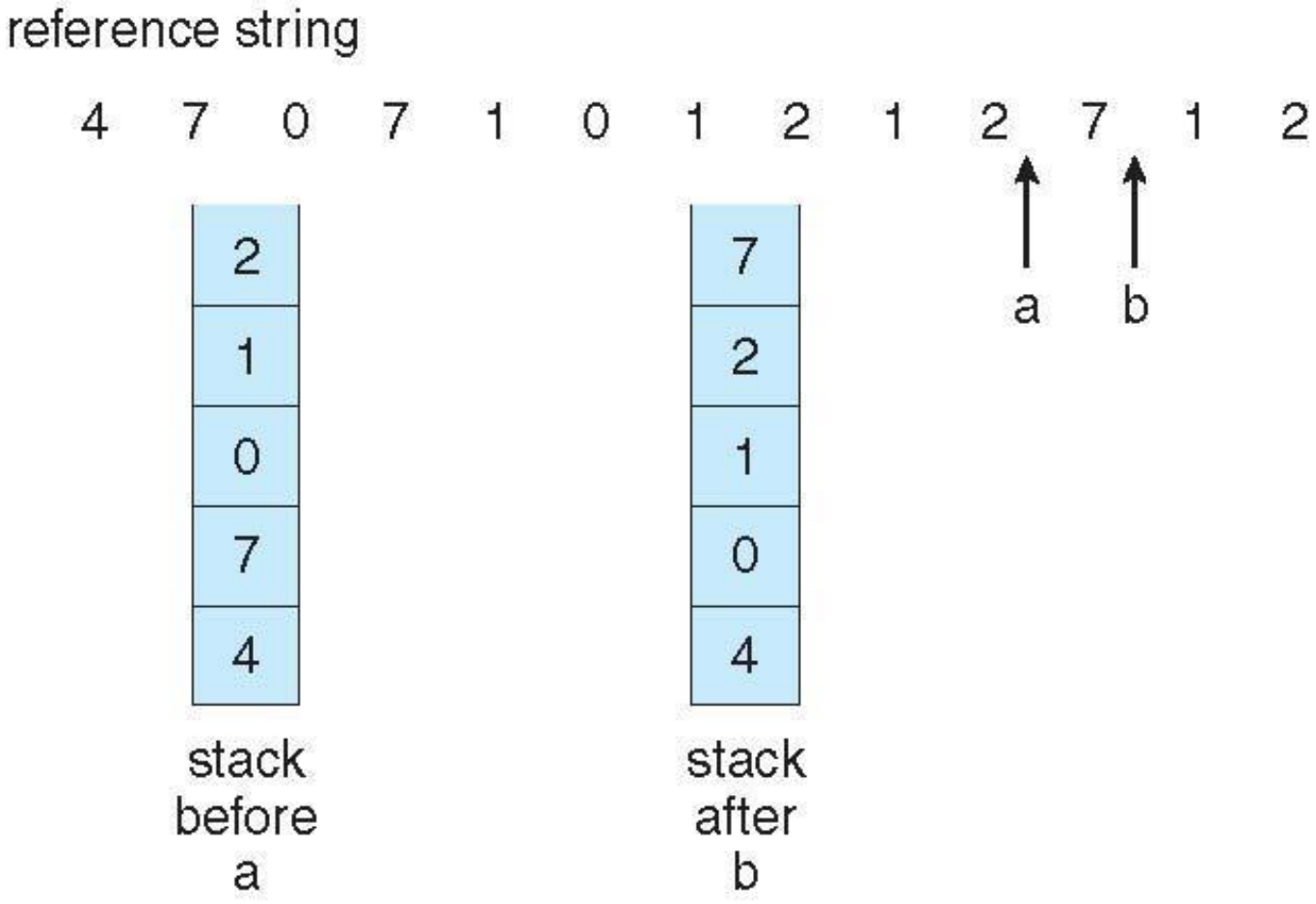


# LRU Algorithm (Cont.)

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, **copy the clock into the counter**
  - When a page needs to be changed, look at the counters to find smallest value
    - Search through table needed
- Stack implementation
  - Top is most recently used. Bottom is least recently used.
  - Since stack entries are removed from the middle of the stack, a **doubly-linked list** implementation is best
  - (**6 pointer changes at worst**--2 for internal neighbors, top of stack, previous top's pointer, entry's 2 pointers

- LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

# Use Of A Stack to Record The Most Recent Page References



# LRU Approximation Algorithms

- Implementation of LRU can be expensive, so in practice may approximate LRU
- **Reference bit**
  - every entry in page table has reference bit
  - bit set (to **1) when page accessed**
  - **periodically the OS "cleans"** the reference bits (resets all to 0)
  - pages with bit reset (i.e., **0) have higher priority** to be replaced
  - Reference bits tell us what pages have been accessed but **do not tell us the order of use**
- **Additional reference bits algorithm**
  - keep record of reference bits at regular time intervals
  - associate **8 bit byte** with each page table entry
  - at regular intervals (e.g., 100 miliseconds) **OS shifts reference bit** for each page into high-order bit of 8-bit byte, shifting other bits right by one.
  - low-order bit discarded

    00000000 not used in 8 time units

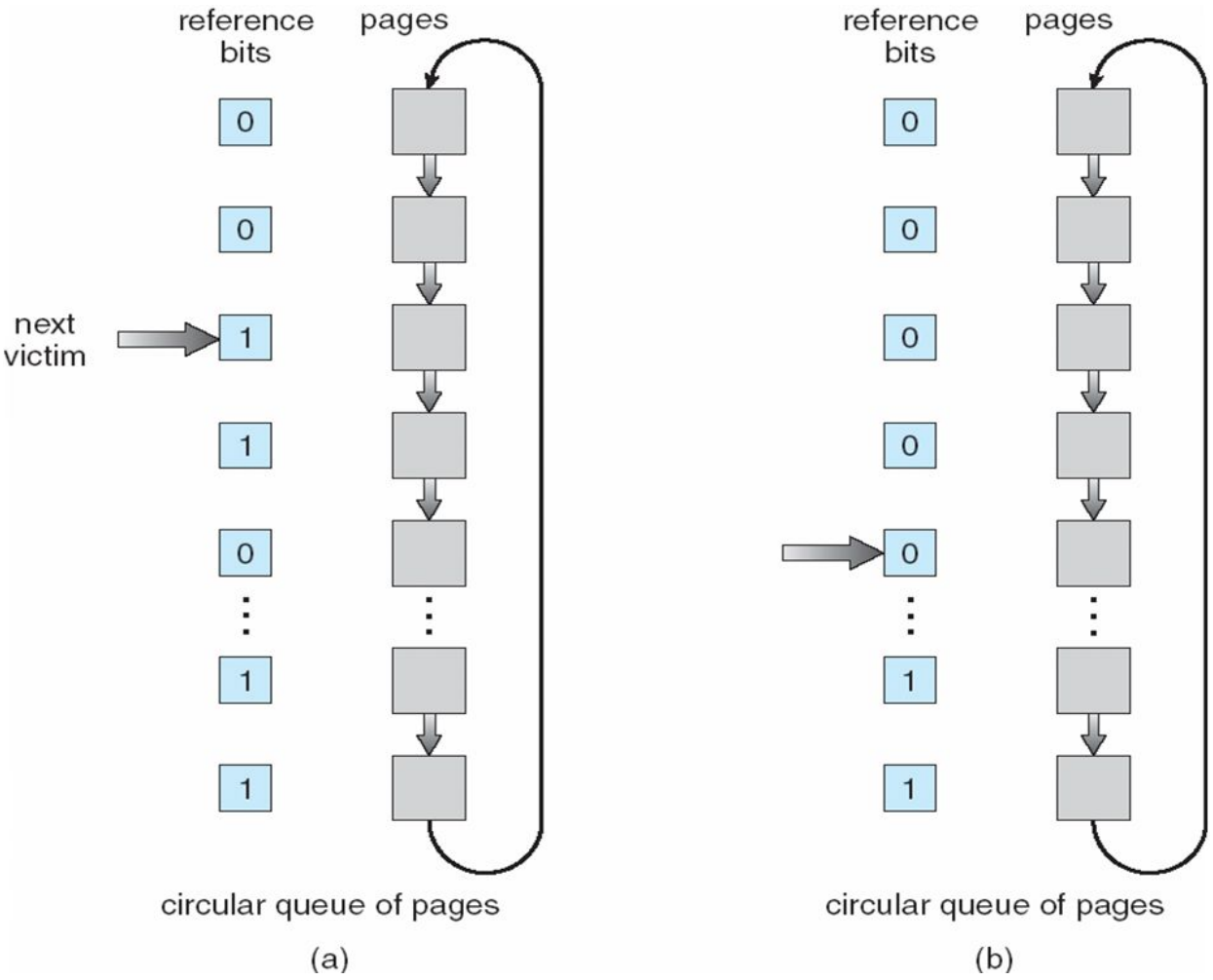
    11111111 used in each of past 8 time units

    00111111 not used in last 2 time units but used in each of 6 previous ones
  - If view numbers as unsigned integers, the **page(s) with the lowest number is the one least recently used** (not guaranteed to be one page--can be ambiguous then)

# LRU Approximation Algorithms

- **Second-chance algorithm (aka clock algorithm)**
  - FIFO replacement but inspect reference bit before actually replacing
  - if reference bit == **0 then replace**
  - if reference bit == **1 then clear reference bit and move on to inspect the next FIFO page**
  - **Implementation: circular queue**; advance pointer, **clearing reference bits until find one with reference bit == 0**
  - Worst case: cycle through all entries (degenerates to FIFO in this case)
  - Slowly sweeping clock hand is good---means there is plenty of memory, not many page faults

# Second-Chance (clock) Page-Replacement Algorithm

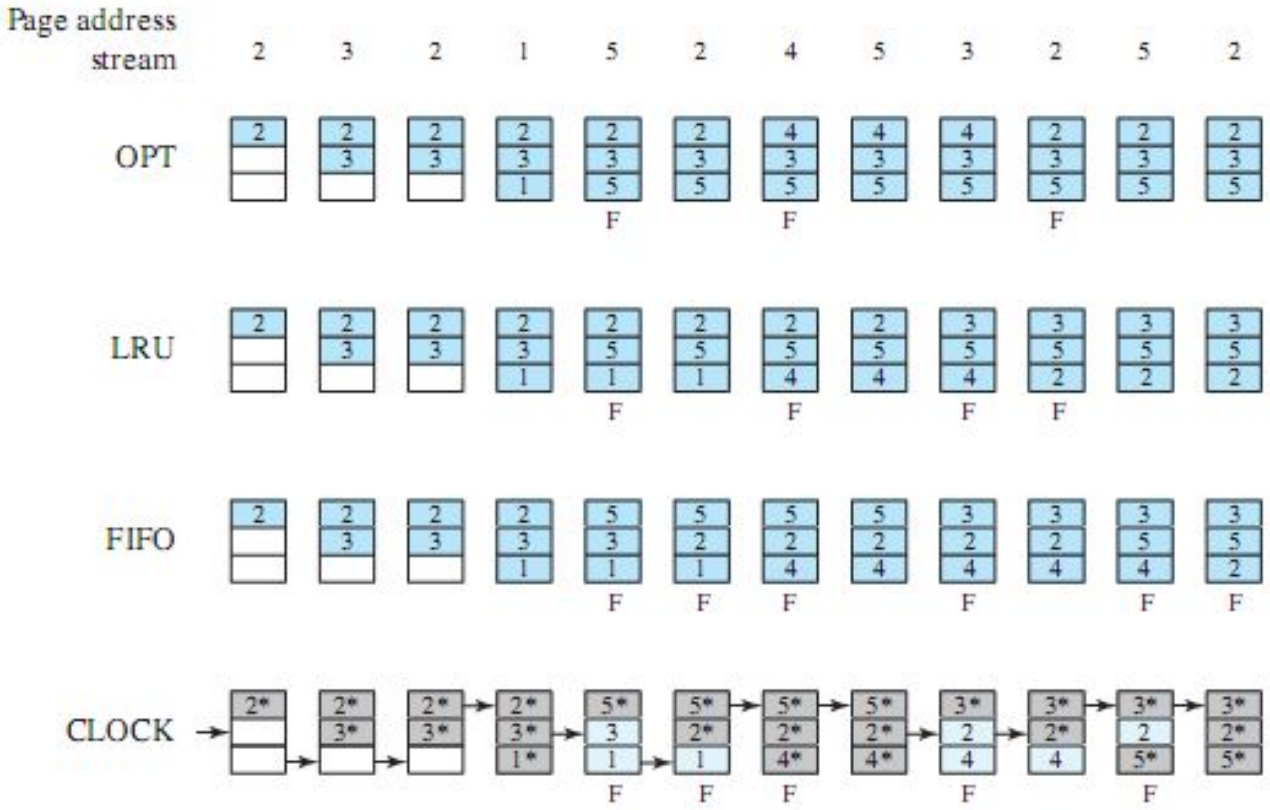


# Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

# Page-Buffering Algorithms

- Keep a **pool of free frames**, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Possibly, **keep list of modified pages**
  - When backing store otherwise idle, write pages there and set to non-dirty
- Possibly, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected




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

**Figure 8.15** **Behavior of Four Page Replacement Algorithms**

# Comparison

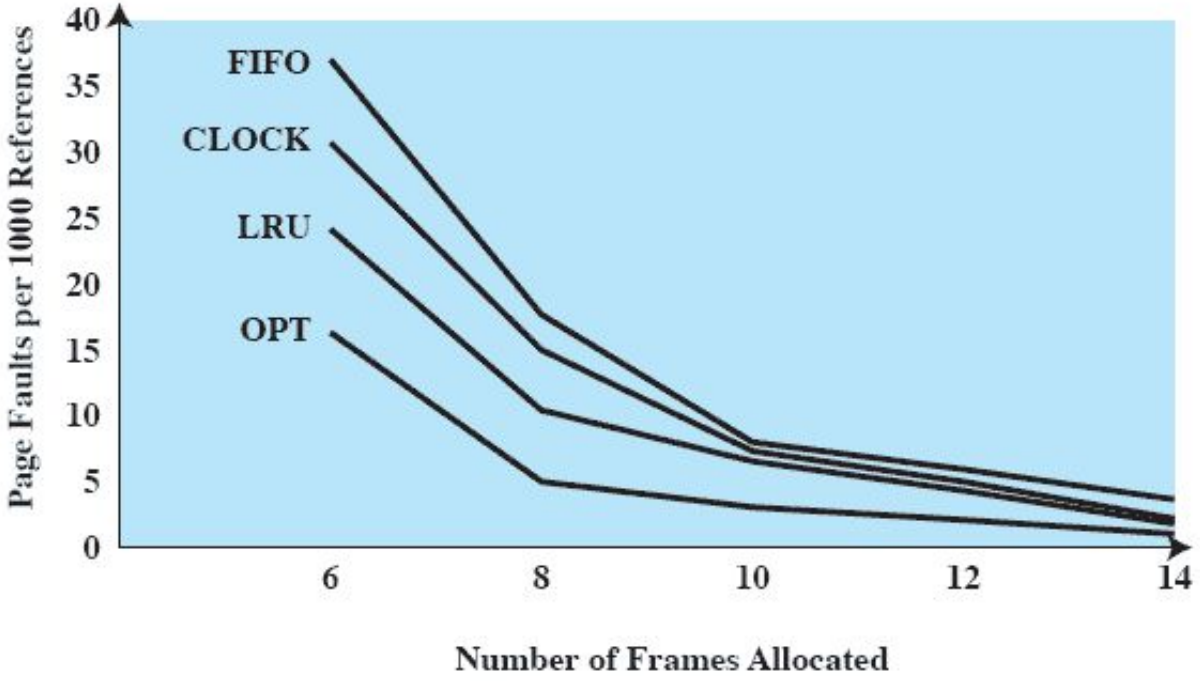



**Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms**

# Applications and Page Replacement

- All of these algorithms have OS guessing about future page access
- Some applications have better knowledge – i.e. databases
- Memory intensive applications can cause double buffering
  - OS keeps copy of page in memory as I/O buffer
  - Application keeps page in memory for its own work
- Operating system can given direct access to the disk, getting out of the way of the applications
  - **Raw disk** mode
- Bypasses buffering, locking, etc

# THANK YOU