CISC 3320

C20a Page Replacement

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Acknowledgement

- These slides are a revision of the slides provided by the authors of the textbook via the publisher of the textbook

Outline

- Page Replacement

- Allocation of Frames
- Thrashing
- Memory-Mapped Files
- Allocating Kernel Memory
- Other Considerations
- Operating-System Examples

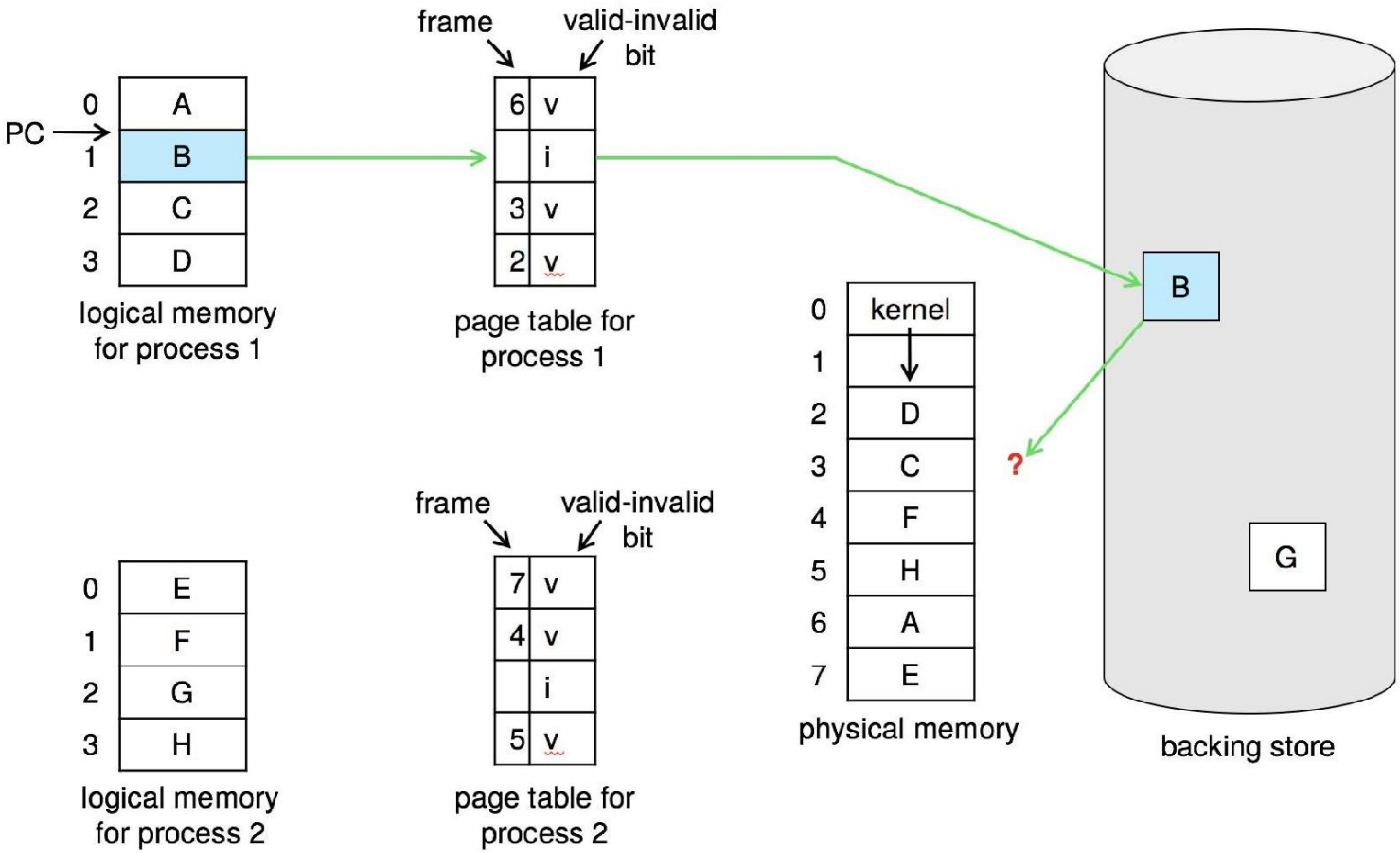
What Happens if There is no Free Frame?

- Used up by process pages
- Also in demand from the kernel, I/O buffers, etc
- How much to allocate to each?
- Page replacement – find some page in memory, but not really in use, page it out
  - Algorithm – terminate? swap out? replace the page?
  - 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 written to disk
- Page replacement completes separation between logical memory and physical memory
  - large virtual memory can be provided on a smaller physical memory

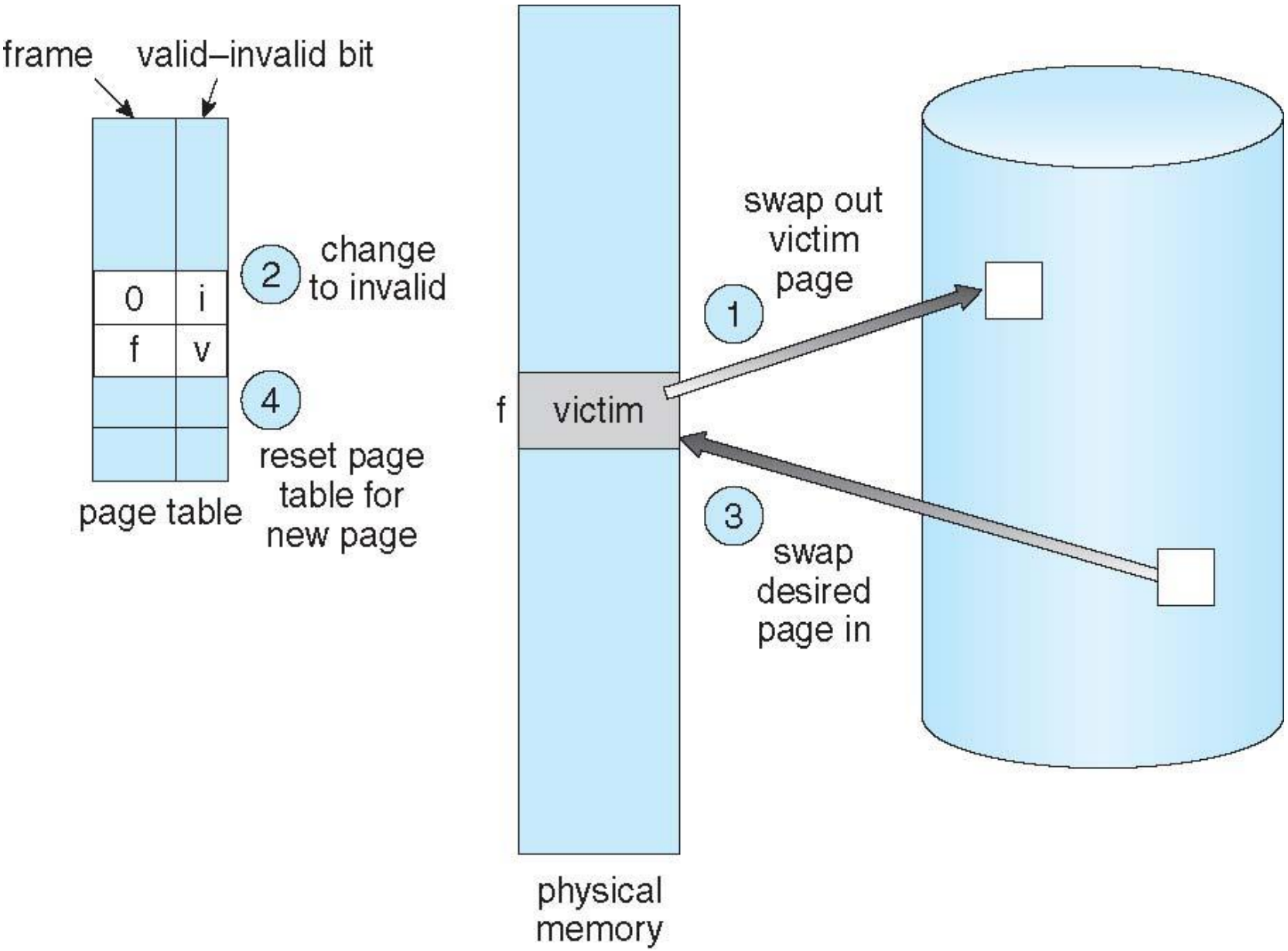
Need For Page Replacement



Basic Algorithm

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 a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

- Note now potentially 2 page transfers for page fault – increasing EAT


frame
valid–invalid bit
0
i
f
v
page table
2
change
to invalid
4
reset page
table for
new page
f
victim
1
swap out
victim
page
3
swap
desired
page in
physical
memory

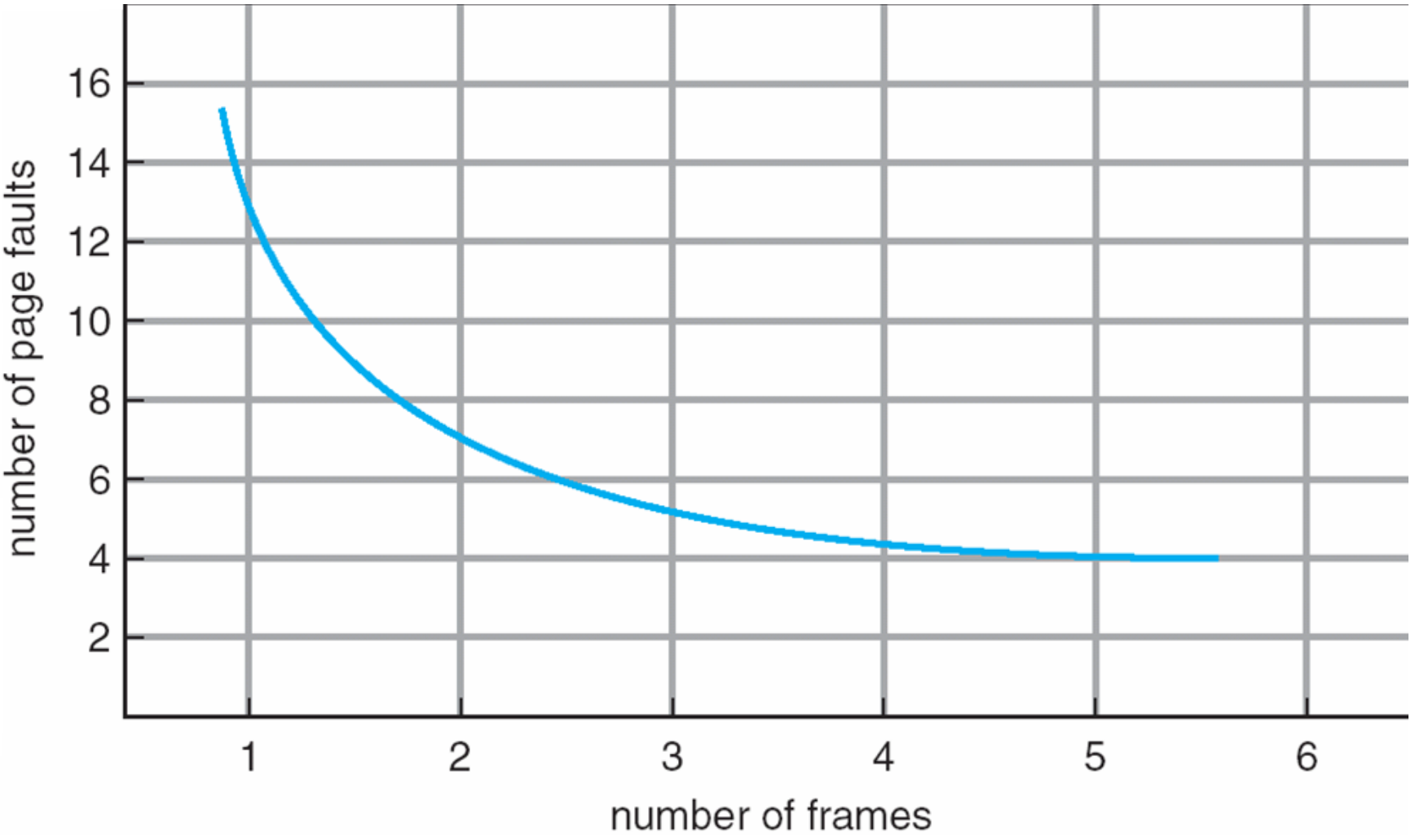
Page and Frame Replacement Algorithms

- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access

Evaluation Consideration

- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the **reference string** of referenced page numbers is

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1


number of page faults
16
14
12
10
8
6
4
2
1
2
3
4
5
6
number of frames

Page Replacement Algorithms

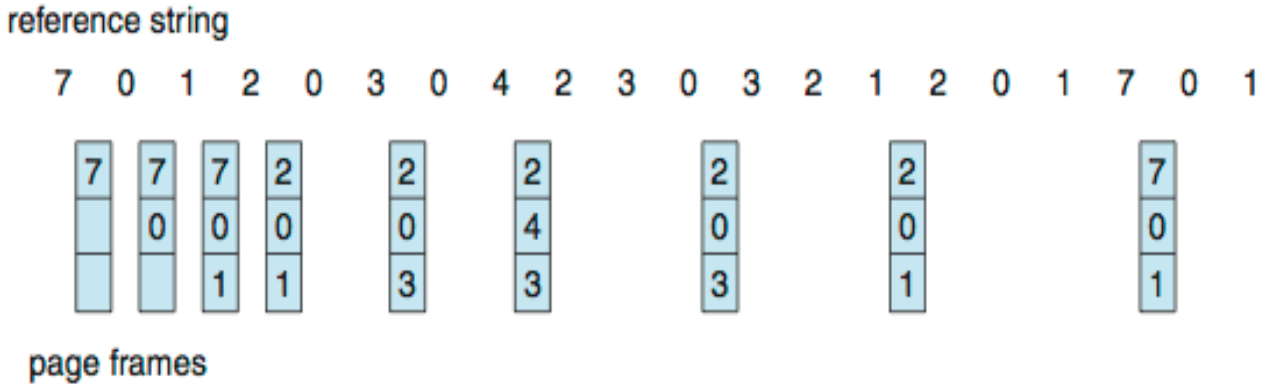
- Optimal Algorithm
- First-In-First-Out (FIFO) Algorithm
- Least Recently Used (LRU) Algorithm
- LRU Approximation Algorithms
- Second-Chance Algorithm
- Enhanced Second-Chance Algorithm
- Counting Algorithm
- Page-Buffering Algorithms

Optimal Algorithm

- Replace page that will not be used for longest period of time

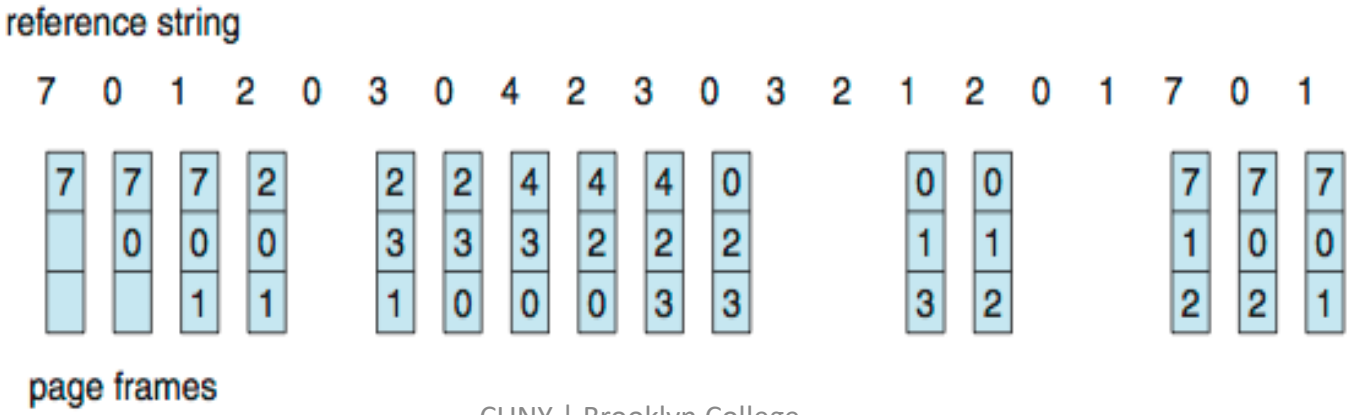
Optimal Algorithm: Example

- Reference string:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- Assume 3 frames



First-In-First-Out (FIFO) Algorithm

- Reference string:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- Assume 3 frames
- 15 page faults

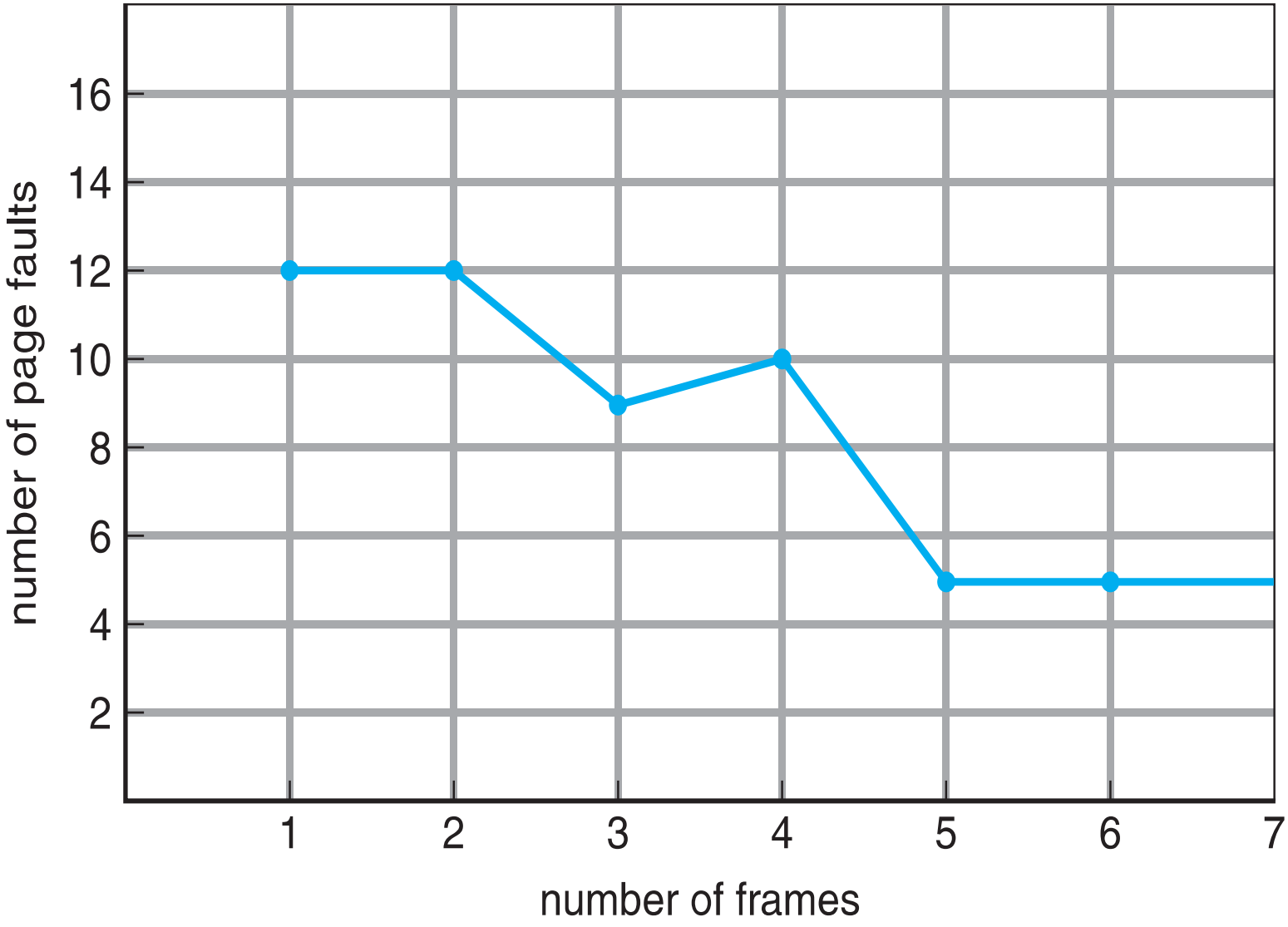


First-In-First-Out (FIFO) Algorithm

- Page faults can vary by reference string
- Now consider 1,2,3,4,1,2,5,1,2,3,4,5

Belady's Anomaly

- Adding more frames can cause more page faults!
    - **Belady's Anomaly**


number of page faults
16
14
12
10
8
6
4
2
1
2
3
4
5
6
7
number of frames

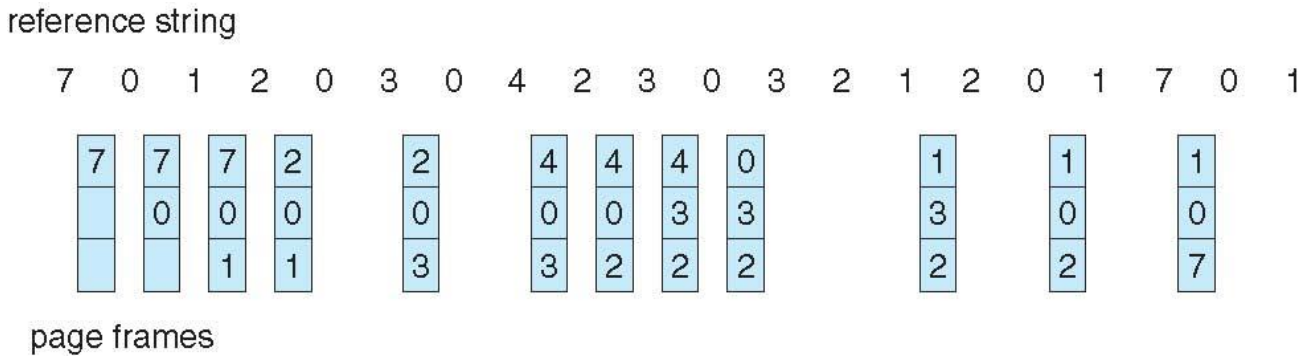
Age of Page

- How to track ages of pages?
  - Just use a FIFO queue

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

Least Recently Used (LRU) Algorithm: Example



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

LRU Algorithm: Implementation Strategies

- Consider the following two strategies
  - Counter implementation
  - Stack implementation

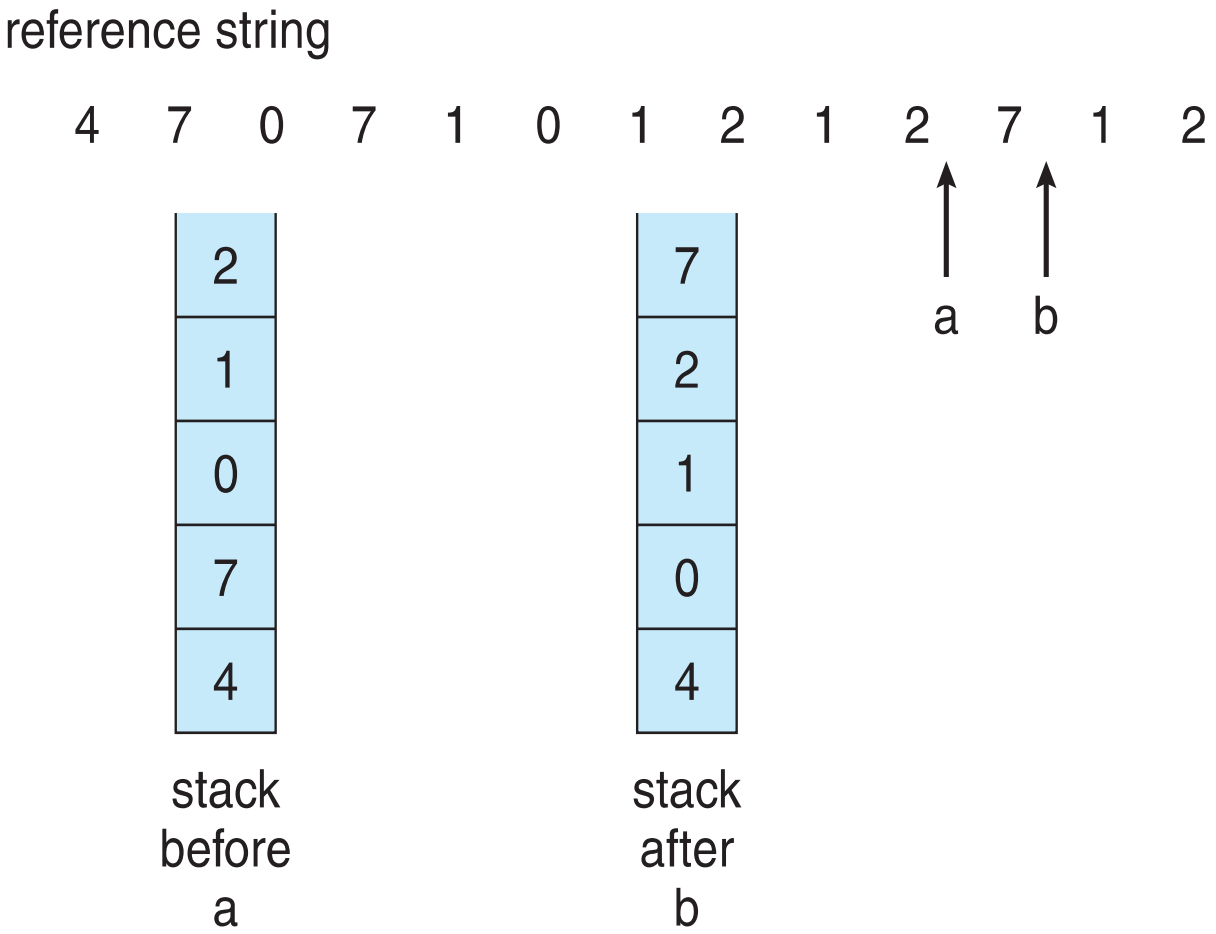
LRU Algorithm: Counter Implementation

- 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

LRU Algorithm: Stack 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
  - But each update more expensive
  - No search for replacement

Use Of A Stack to Record Most Recent Page References



Belady's Anomaly

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

LRU Approximation Algorithms

- LRU needs special hardware and still slow
- Use approximation

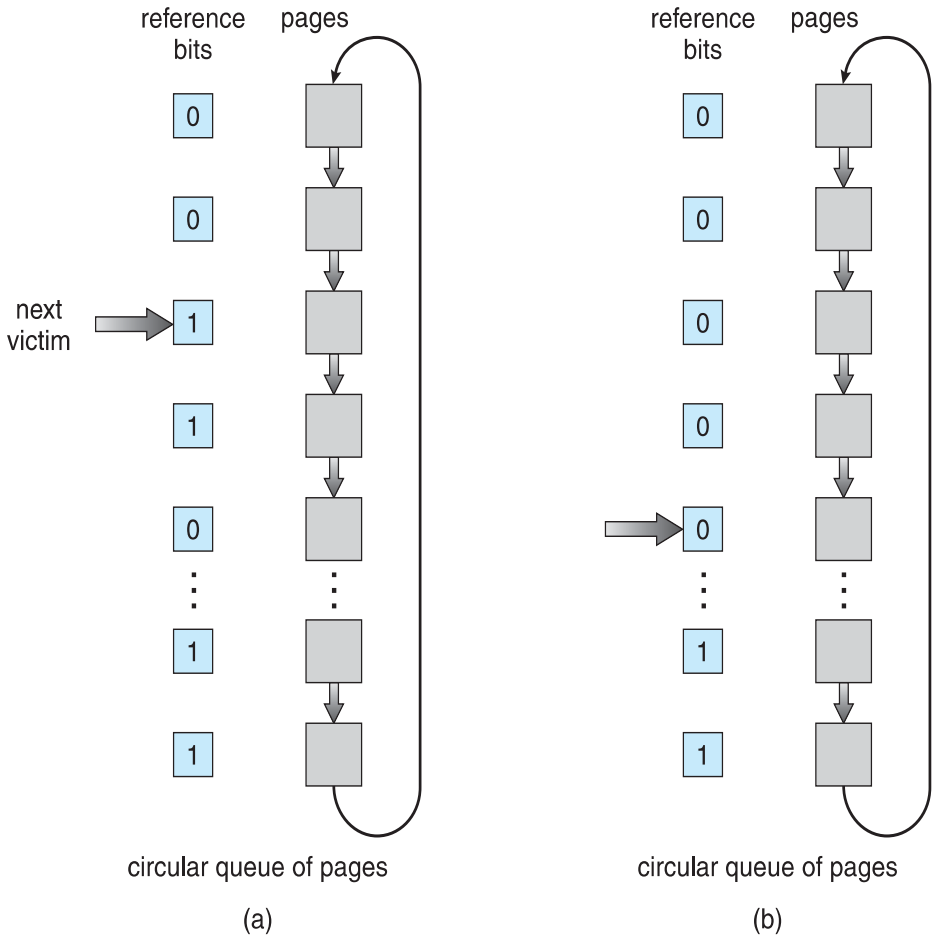
Reference Bit

- Algorithm: **reference bit**
- With each page associate a bit, initially = 0
- When page is referenced bit set to 1
- Replace any with reference bit = 0 (if one exists)
  - We do not know the order, however

Second-Chance Algorithm

- Generally FIFO, plus hardware-provided reference bit
- Also called Clock replacement
- If page to be replaced has
  - Reference bit = 0 -> replace it
  - reference bit = 1 then:
    - set reference bit 0, leave page in memory
    - replace next page, subject to same rules

Second-Chance (Clock) Page-Replacement: Example



Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- Take ordered pair (reference, modify):
  - (0, 0) neither recently used not modified – best page to replace
  - (0, 1) not recently used but modified – not quite as good, must write out before replacement
  - (1, 0) recently used but clean – probably will be used again soon
  - (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  - Might need to search circular queue several times

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **Lease Frequently Used (LFU) Algorithm**: replaces page with smallest count
- **Most Frequently Used (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

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

Questions?

- Page replacement algorithms