SCS 1214: Operating Systems

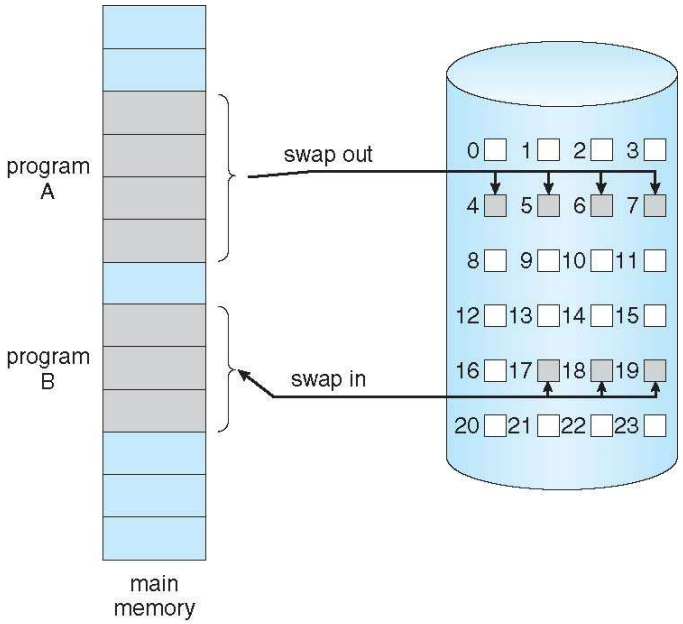
Dr. Dinuni Fernando, PhD

Based on Operating System Concepts by A.Silberschatz, P.Galvin, and G.Gagne

Page Replacement Algorithms II

Demand Paging

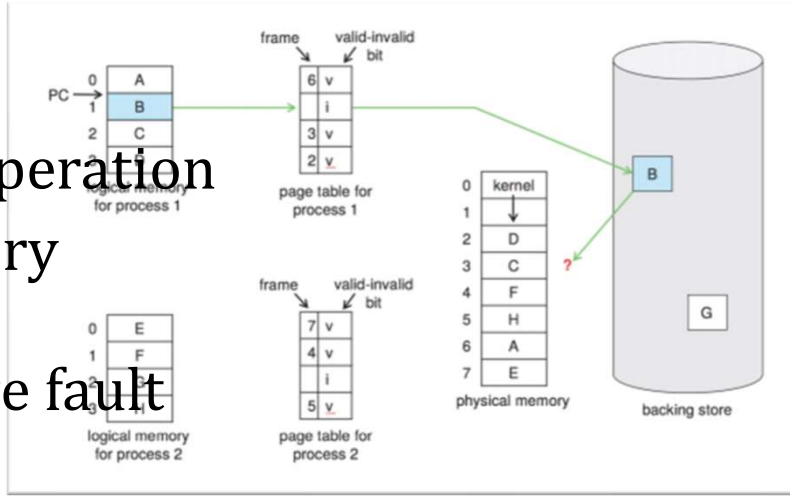
- Could bring entire process into memory at load time
- Or bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users
- Similar to paging system with swapping (diagram on right)
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**



Page Fault

- Accessing a page that was not brought into memory. Access to a page marked invalid causes a **page fault**

1. Operating system looks at internal table (in PCB) to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault



Copy-on-Write

- **Copy-on-Write** (COW) allows both parent and child processes to initially ***share*** the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- In general, free pages are allocated from a **pool** of **zero-fill-on-demand** pages
  - Pool should always have free frames for fast demand page execution
    - Don't want to have to free a frame as well as other processing on page fault
  - Why zero-out a page before allocating it?

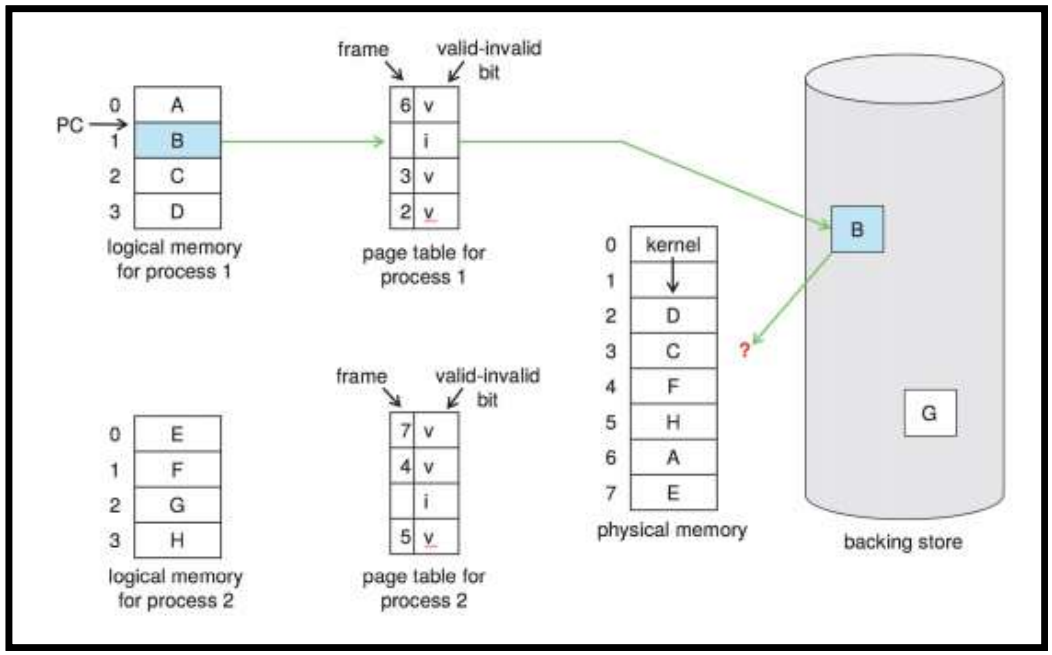
What happens if there is no free frame ?

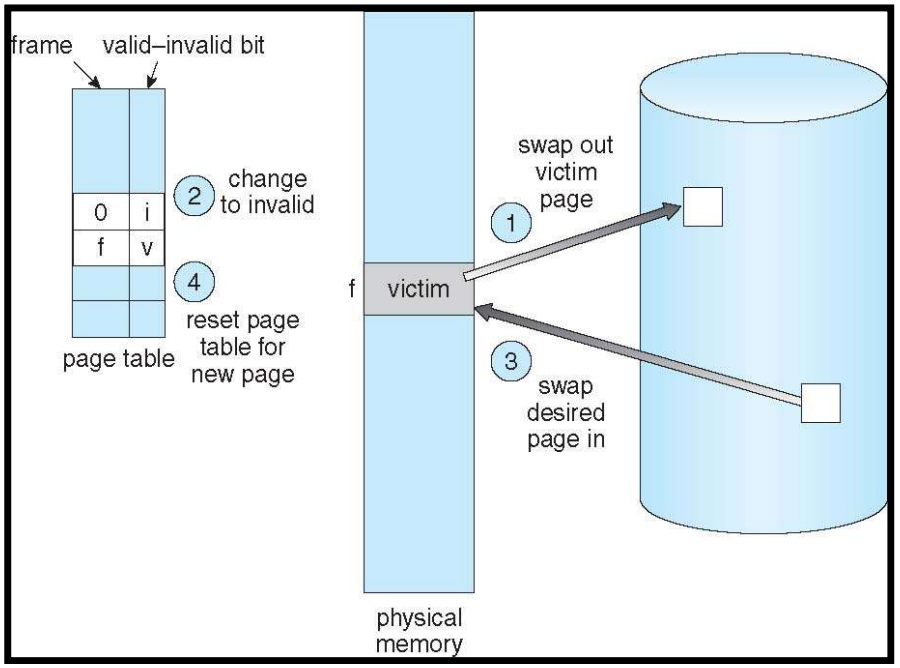
- Used up by process pages
- Also in demand from the kernel, I/O buffers, etc
- How much to allocate to each? For pages and I/O buffers
- Page replacement – find some page in memory, but not really in use, page it out
  - Algorithm – terminate? swap out? replace the page?
  - Performance – want an algorithm 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 Page Replacement

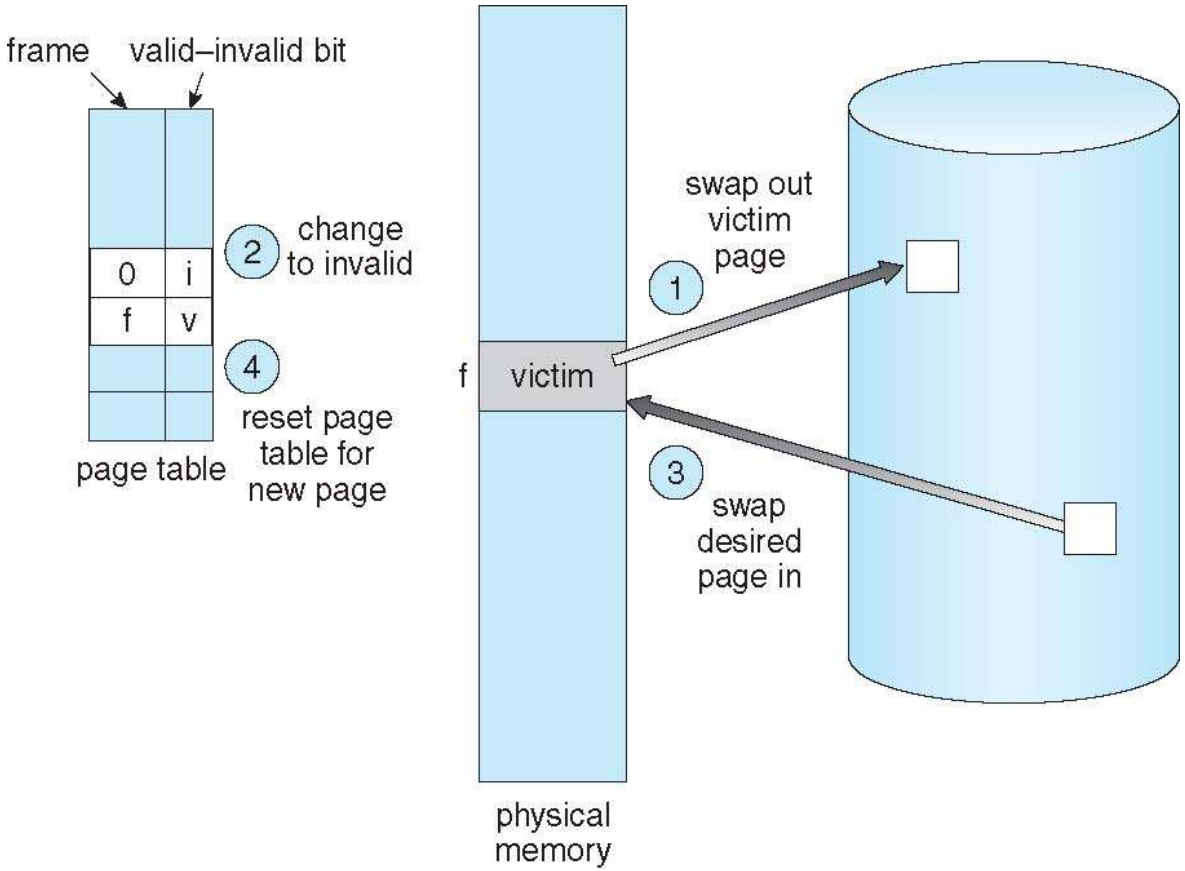
1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use 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

Page Replacement



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
- 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

Reference String

- Address sequence after tracing a particular process,

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,

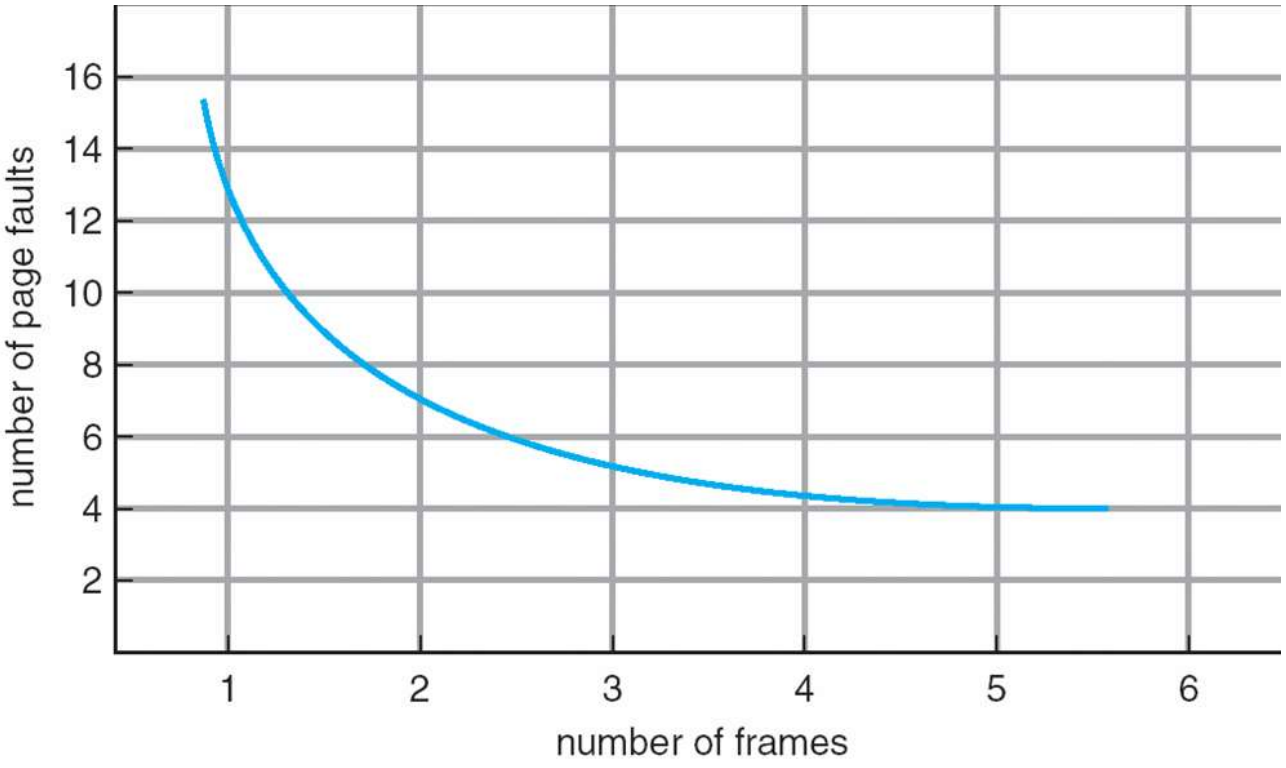
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

At 100 bytes per page, above sequence can reduce to following reference string.

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

Reference String										
1	4	1	6	1	6	1	6	1	6	1
	1	1	1	1	1	1	1	1	1	1
		4	4	4	4	4	4	4	4	4
				6	6	6	6	6	6	6
F	F		F							

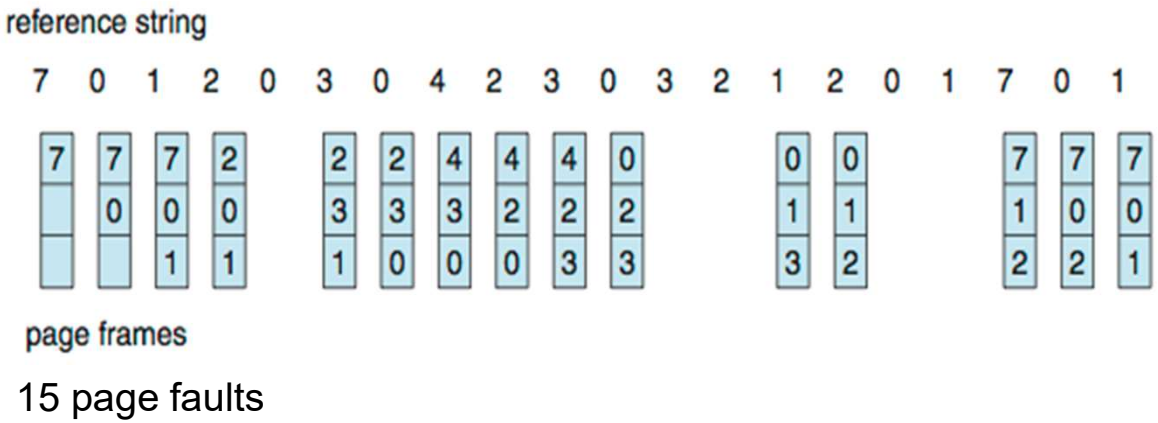
Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

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

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



15 page faults

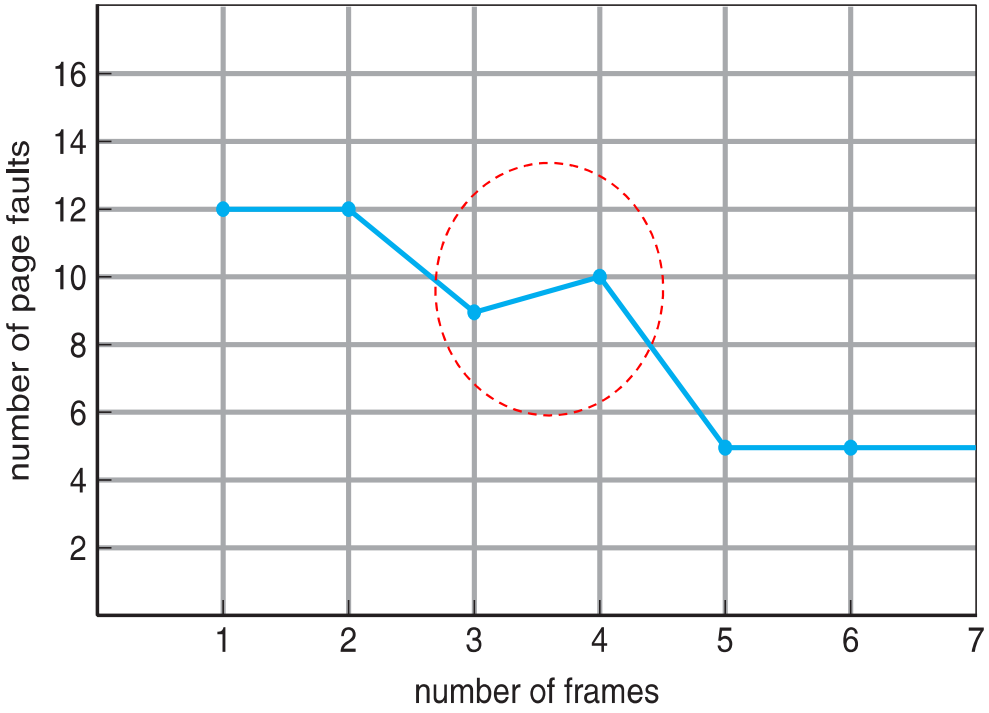
First-In-First-Out (FIFO) Algorithm (cont.)

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5

Ref string	1	2	3	4	1	2	5	1	2	3	4	5
3 frames	1	1	1	4	4	4	5			5	5	
		2	2	2	1	1	1			3	3	
			3	3	3	2	2			2	4	
# faults	1	2	3	4	5	6	7			8	9	
4 frames	1	1	1	1			5	5	5	5	4	4
		2	2	2			2	1	1	1	1	5
			3	3			3	3	2	2	2	2
				4			4	4	4	3	3	3
# faults	1	2	3	4			5	6	7	8	9	10

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

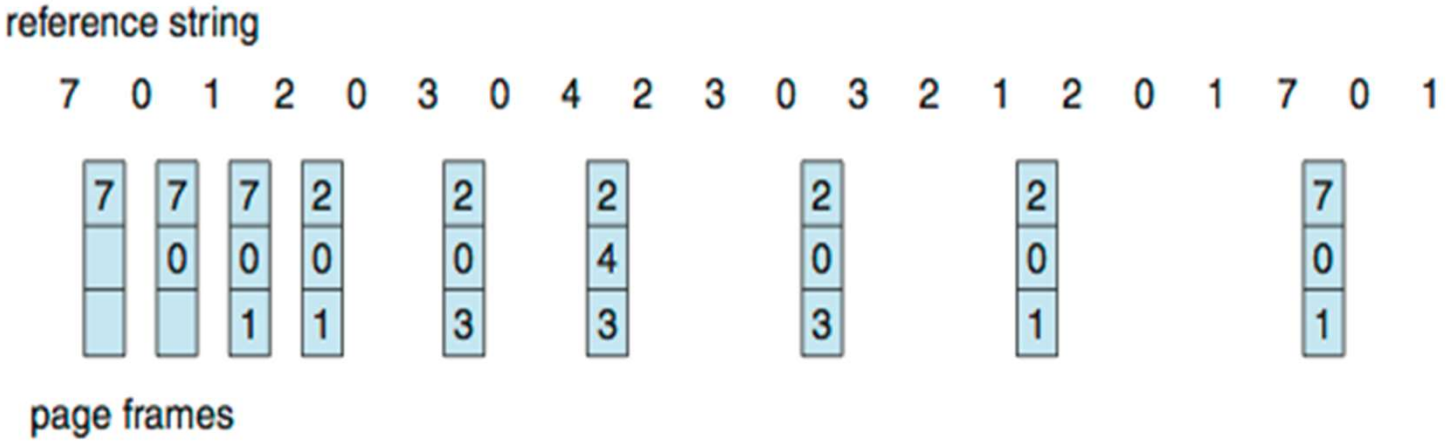
FIFO Illustrating Belady's Anomaly



Optimal Page Replacement Algorithm

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

  Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

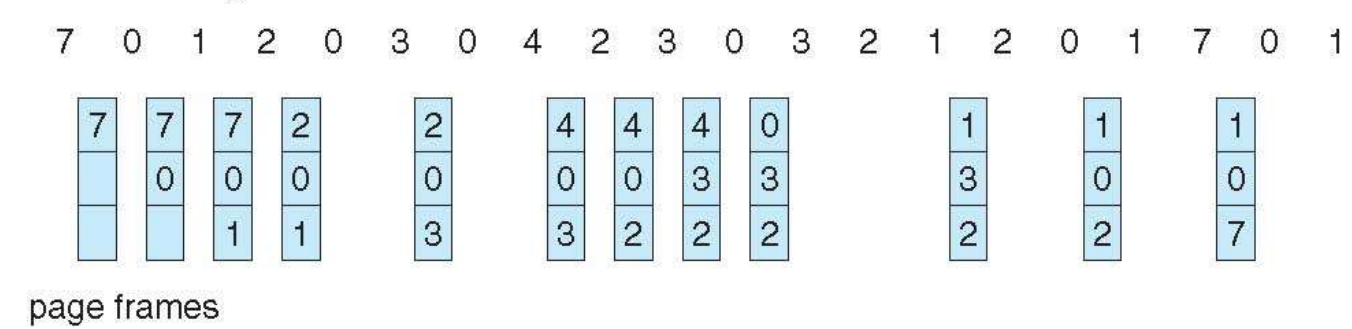


- 9 page faults is optimal for the example
- How do you know this?
  - Can't read the future

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

Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**



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

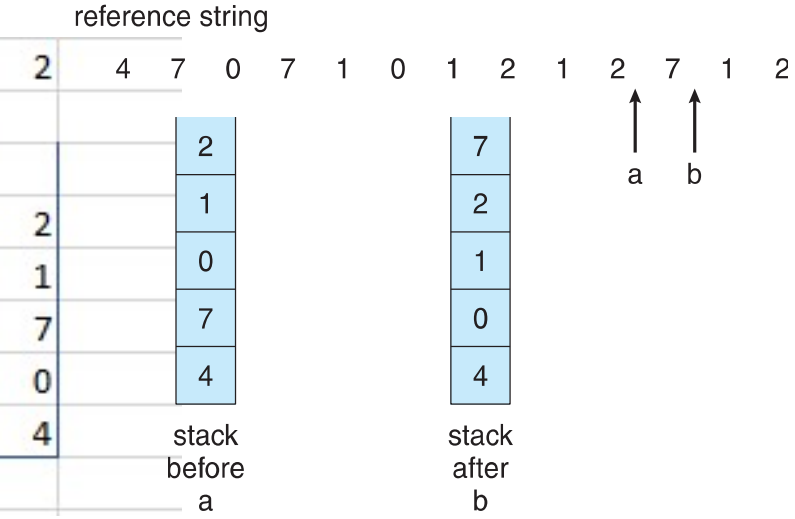
LRU Algorithm (Cont.)

- 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
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires top-bottom items to be changed
  - But each update more expensive
  - No search for replacement
- LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

Use Of A Stack to Record Most Recent Page References

Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

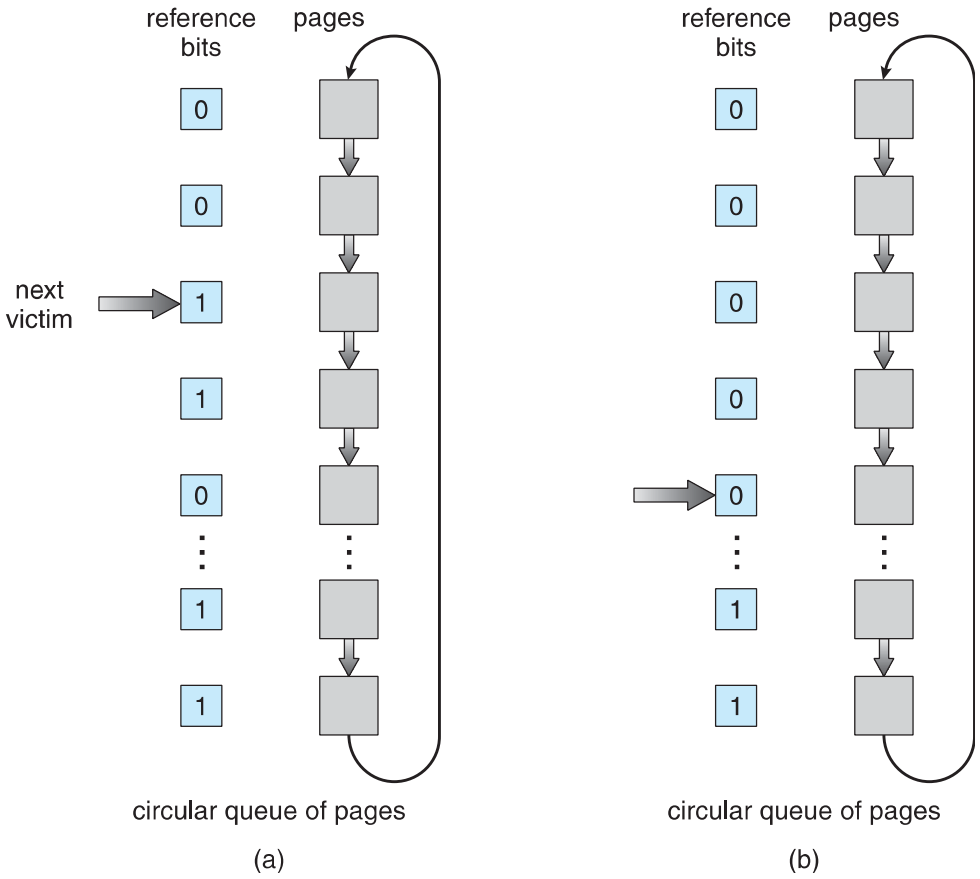
Ref string	4	7	0	7	1	0	1	2	1	2	7	1	2
Stack													
								2	1	2	7	1	2
					1	0	1	1	2	1	2	7	1
			0	7	7	1	0	0	0	0	1	2	7
		7	7	0	0	7	7	7	7	7	0	0	0
	4	4	4	4	4	4	4	4	4	4	4	4	4



LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **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
- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - **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 Algorithm



Enhanced Second-Chance Algorithm

- Can improve algorithm by using reference bit and modify bit (dirty bit – similar as in page tables)
- Ordered pairs (reference bit, modified bit)
  1. (0, 0) neither recently used not modified – best page to replace
  2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
  3. (1, 0) recently used but clean – probably will be used again soon
  4. (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 based page Replacements

- Keep a counter of the number of references

1. Least frequently used (LFU)
   - Page with the smallest counter will be replaced.
   - Ideally, an actively used page should have a large reference count.
   - Eg: when a page is heavily used in the initial phase and never used again. This will remain in memory even its no longer needed.
     - Solution – shift the counters right by 1 bit at regular intervals
2. Most frequently used (MFU)
   - Argument - Page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- How do we allocate the fixed amount of free memory among the various processes?
  - If we have 93 free frames and two processes, how many frames does each process get?
  - Consider a simple case of a system with 128 frames. The operating system may take 35, leaving 93 frames for the user process.
  - Under pure demand paging, all 93 frames would initially be put on the free-frame list. When a user process started execution, it would generate a sequence of page faults.
  - The first 93 page faults would all get free frames from the free-frame list. When the free-frame list was exhausted, a page-replacement algorithm would be used to select one of the 93 in-memory pages to be replaced with the 94th, and so on.
  - When the process terminated, the 93 frames would once again be placed on the free-frame list.

Minimum number of frames

- Cannot allocate more than the total number of available frames (unless there is page sharing).
- Allocating least a minimum number of pages – performance
  - #frames allocated per process decreases , page fault rate increases., slow process execution.
  - When a page fault occurs before an executing instruction is complete, instruction must be restarted.
  - Must have enough frames to hold all the different pages that any single instruction can reference.
- Minimum number of frames per process is defined by the architecture, the maximum number is defined by the amount of available physical memory

Allocation Algorithms

- Equal allocation
  - To split m frames among n processors, time everyone an equal share m/n frames (assume OS doesn't need frames at this moment).
- Proportional allocation
  - Allocate available memory to each process according to its size.
  - Eg: system with a 1KBframe size. If a small student process of 10KB and an interactive database of 127KB only two processes running in system with 62 free frames.
    - Giving 31 frames each doesn't make sense. Student process needs only 10 frames and other 21 is wasted.

Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Thank you!