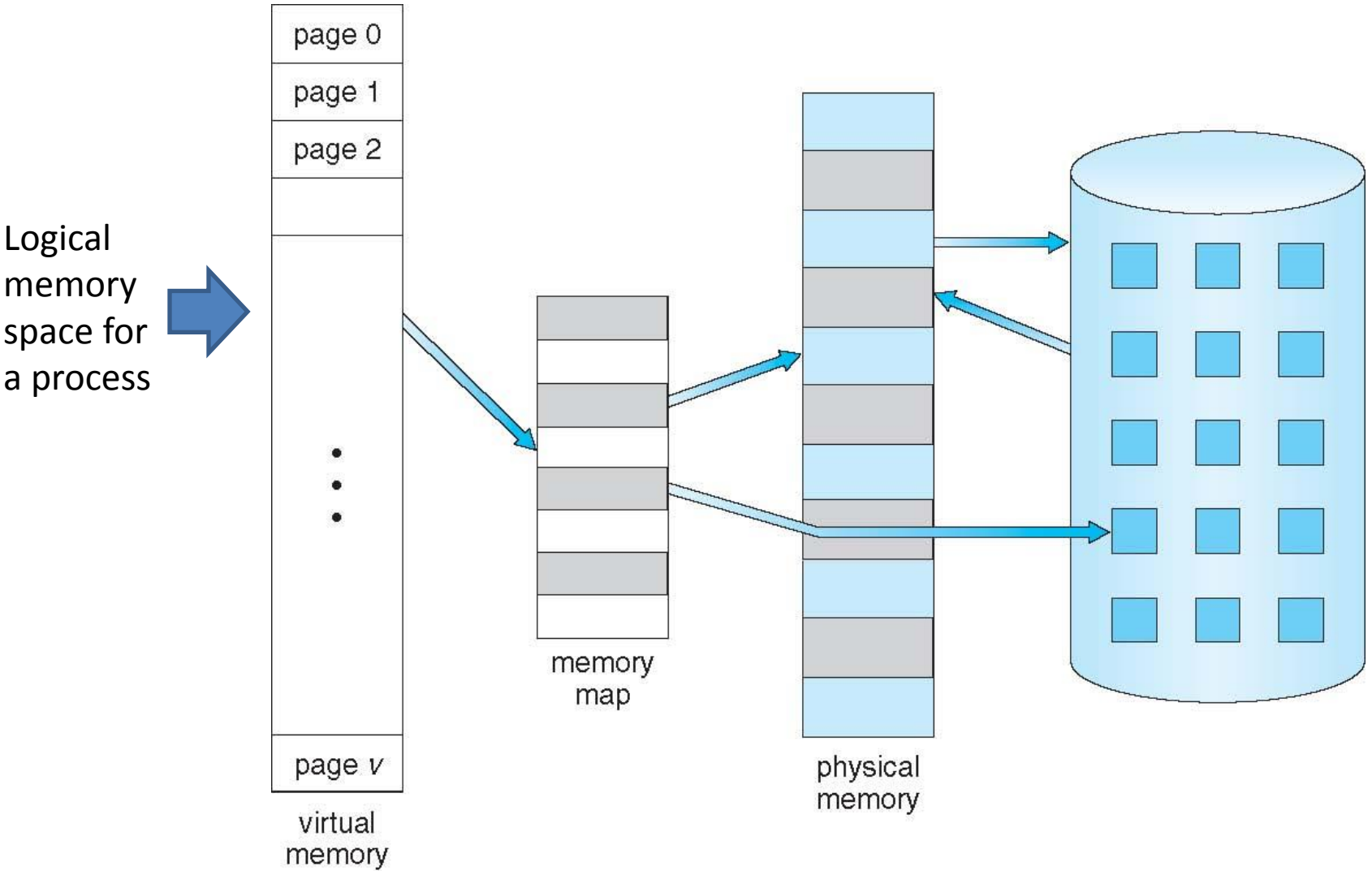
Background

- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes

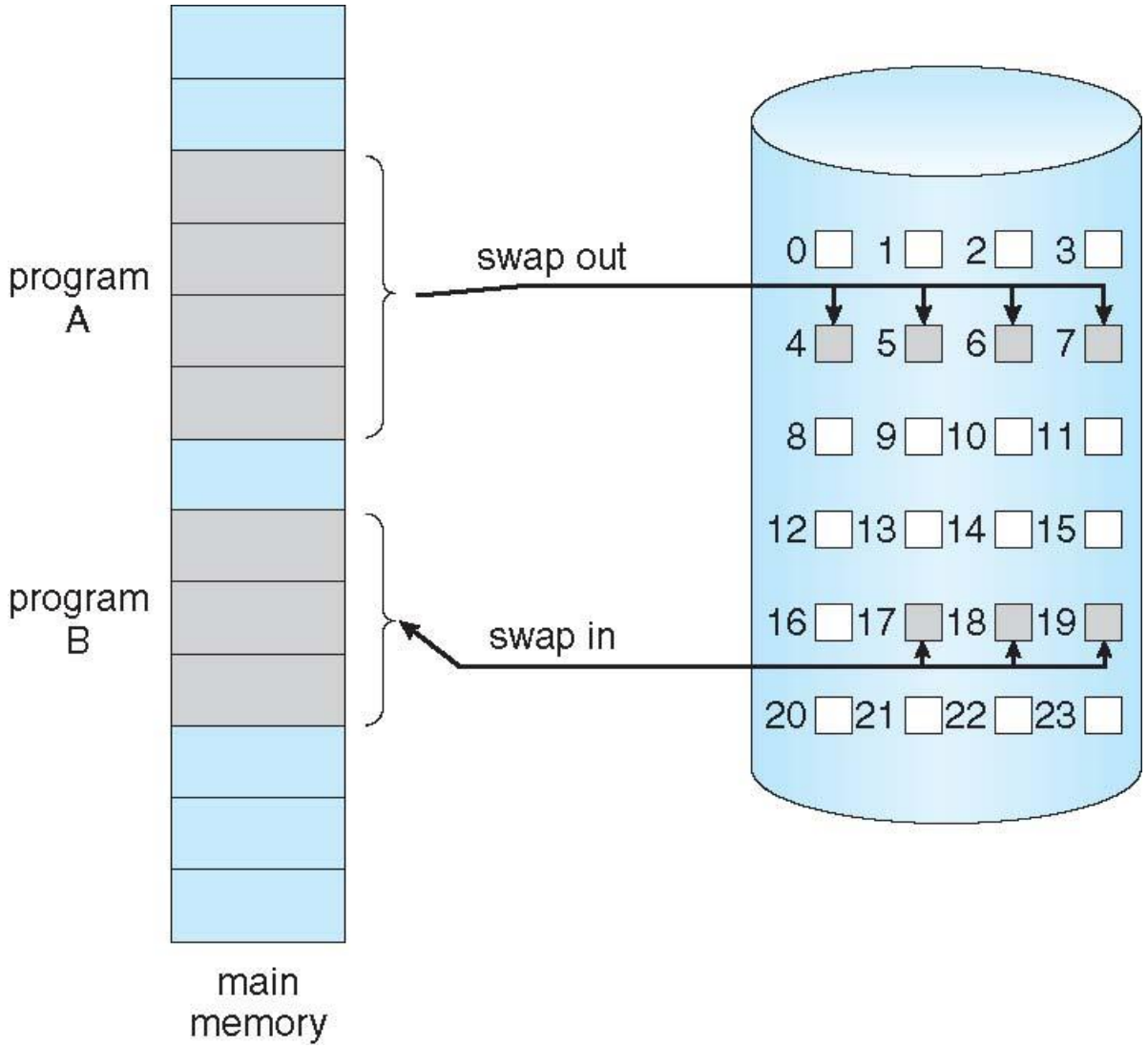
Virtual Memory That is Larger Than Physical Memory



Demand Paging

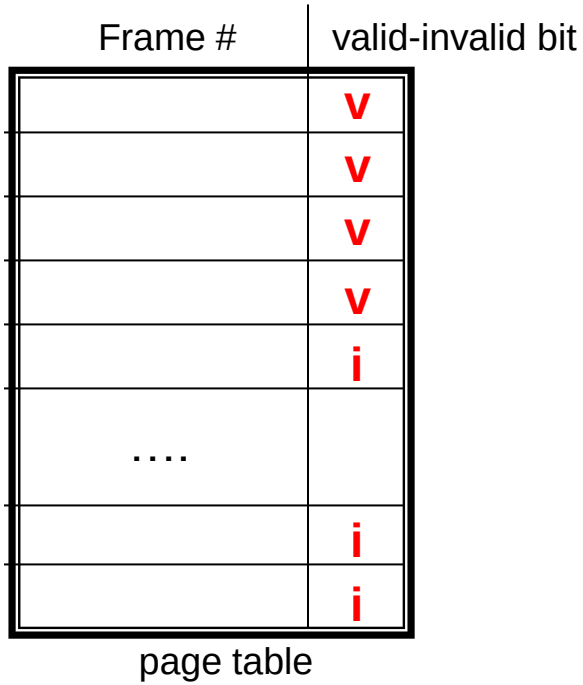
- Implementation way for VIRTUAL MEMORY
- Bring a page into memory only when it is needed
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  - **v** ⇒ in-memory – **memory resident**,
  - **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- During address translation, if valid–invalid bit in page table entry is **I** ⇒ **page fault**



Page Fault

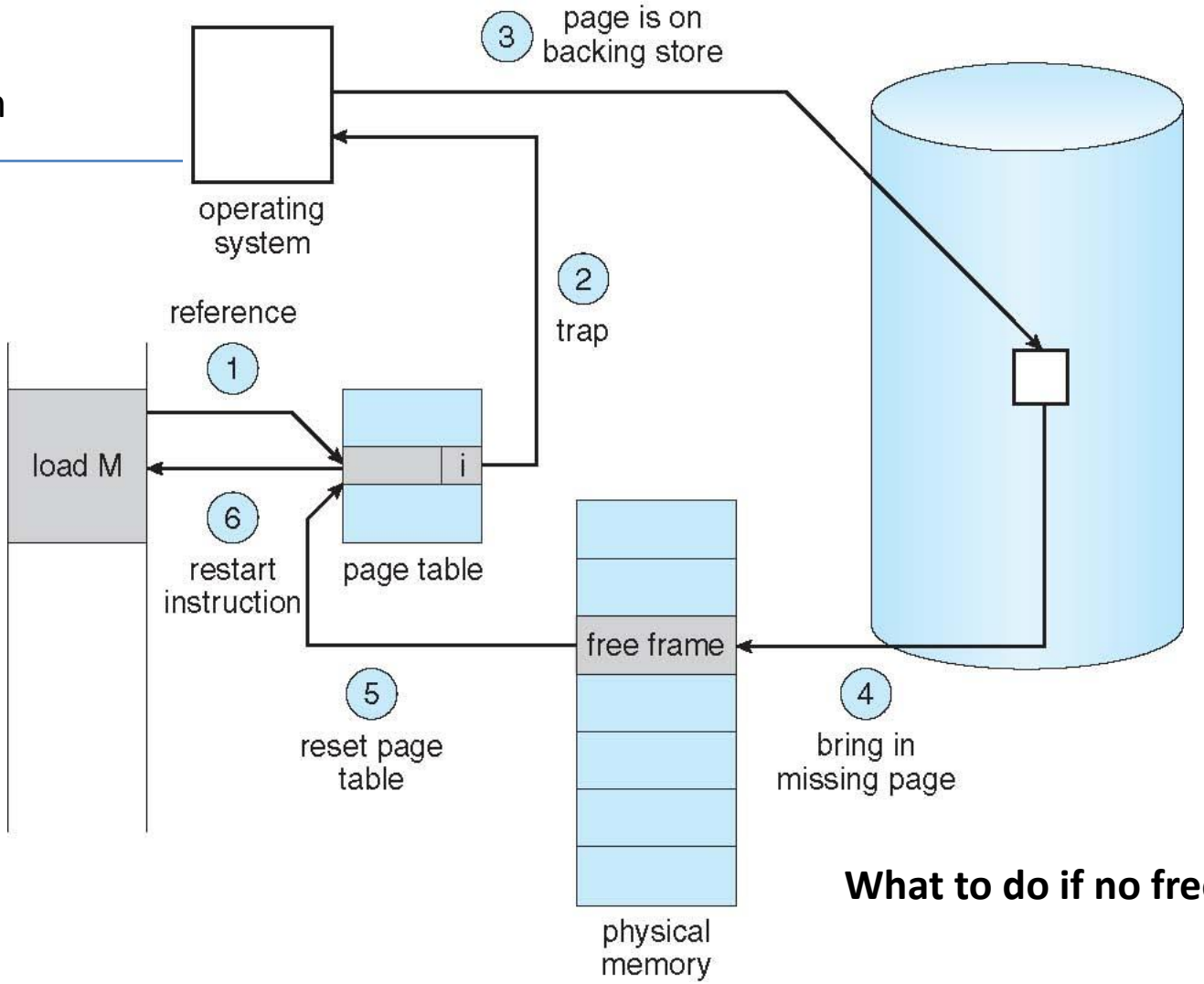
- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort (segmentation!!)
   - Just not in memory
2. Get empty 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

Steps in Handling a Page Fault

May be segmentation error!

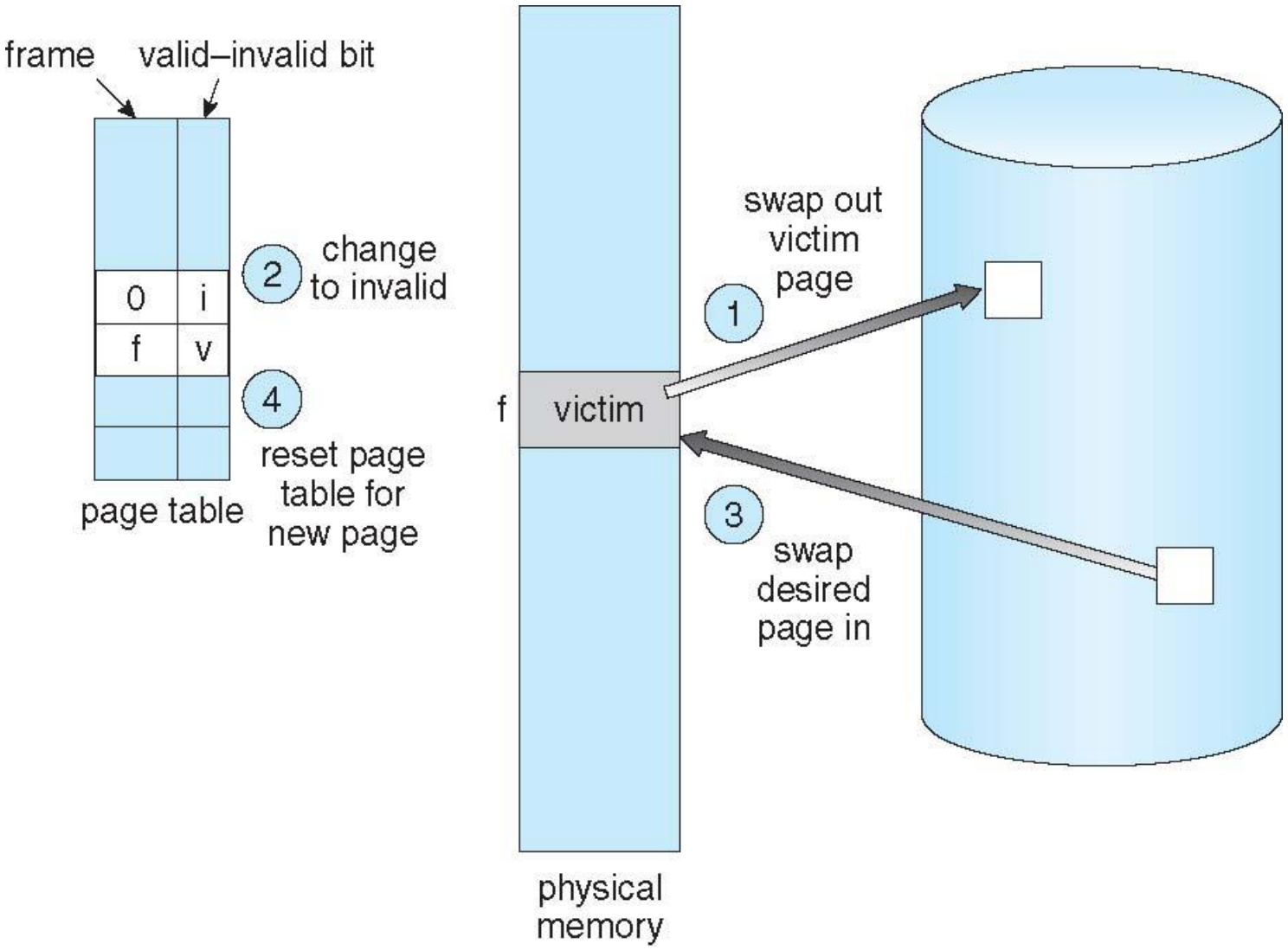


What to do if no free frame!!

What Happens if There is no Free Frame?

- 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

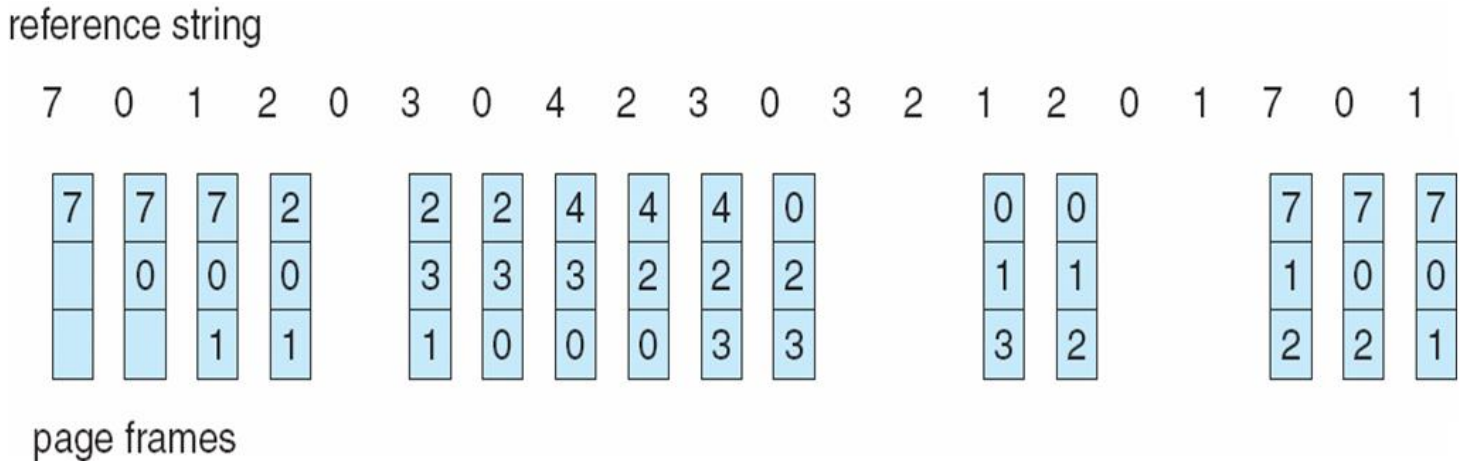
Page Replacement



First-In-First-Out (FIFO) Algorithm

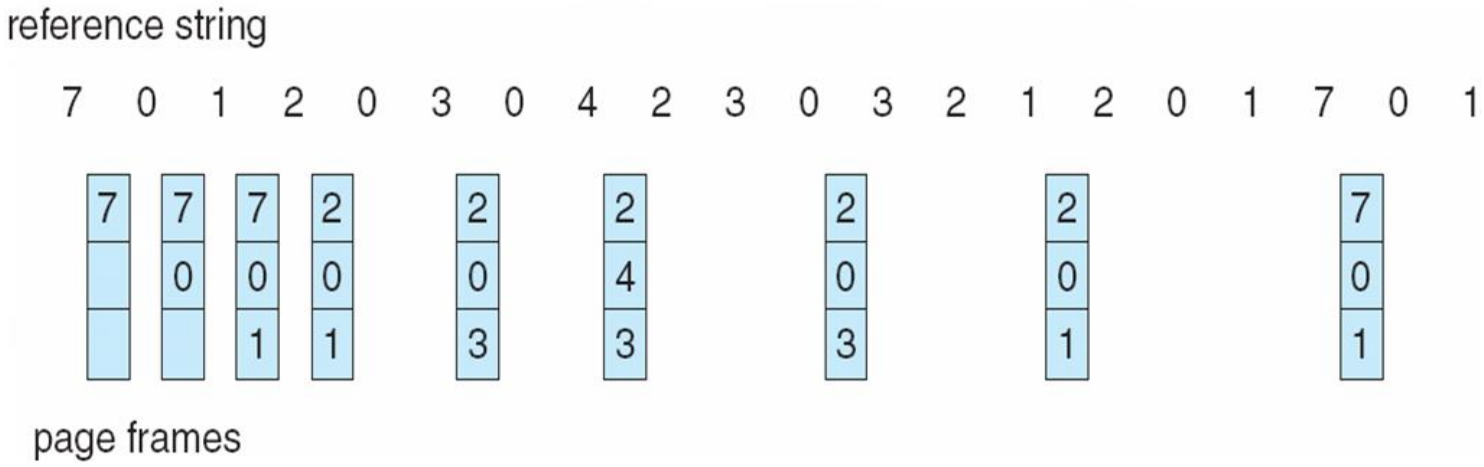
- Reference string:
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

15 page faults



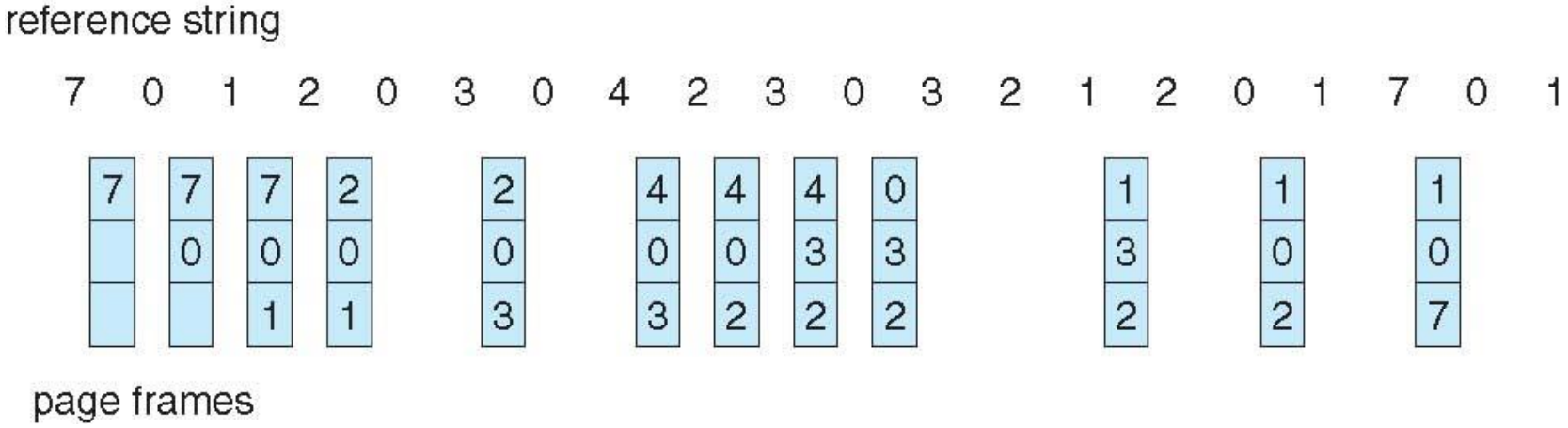
Optimal Algorithm

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example on the next slide
- 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
- 12 faults – better than FIFO but worse than OPT



Exercise

Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement