VIRTUAL MEMORY

VIRTUAL MEMORY

- Virtual memory is a feature of an operating system that allows a computer to compensate for shortages of physical memory by temporarily **transferring pages of data from random access memory (RAM) to disk storage.**

- It is a technique that allows execution of processes that are not completely in memory.

- One of the major advantage of this scheme is that programs can be larger than the physical memory available.

VIRTUAL MEMORY

- Reason to move pages of data from RAM to disk storage - Because OS was running out of RAM.

- Solution: The operating system will need to move other pages to hard disk so it has space to bring back the pages it needs from temporary disk storage.

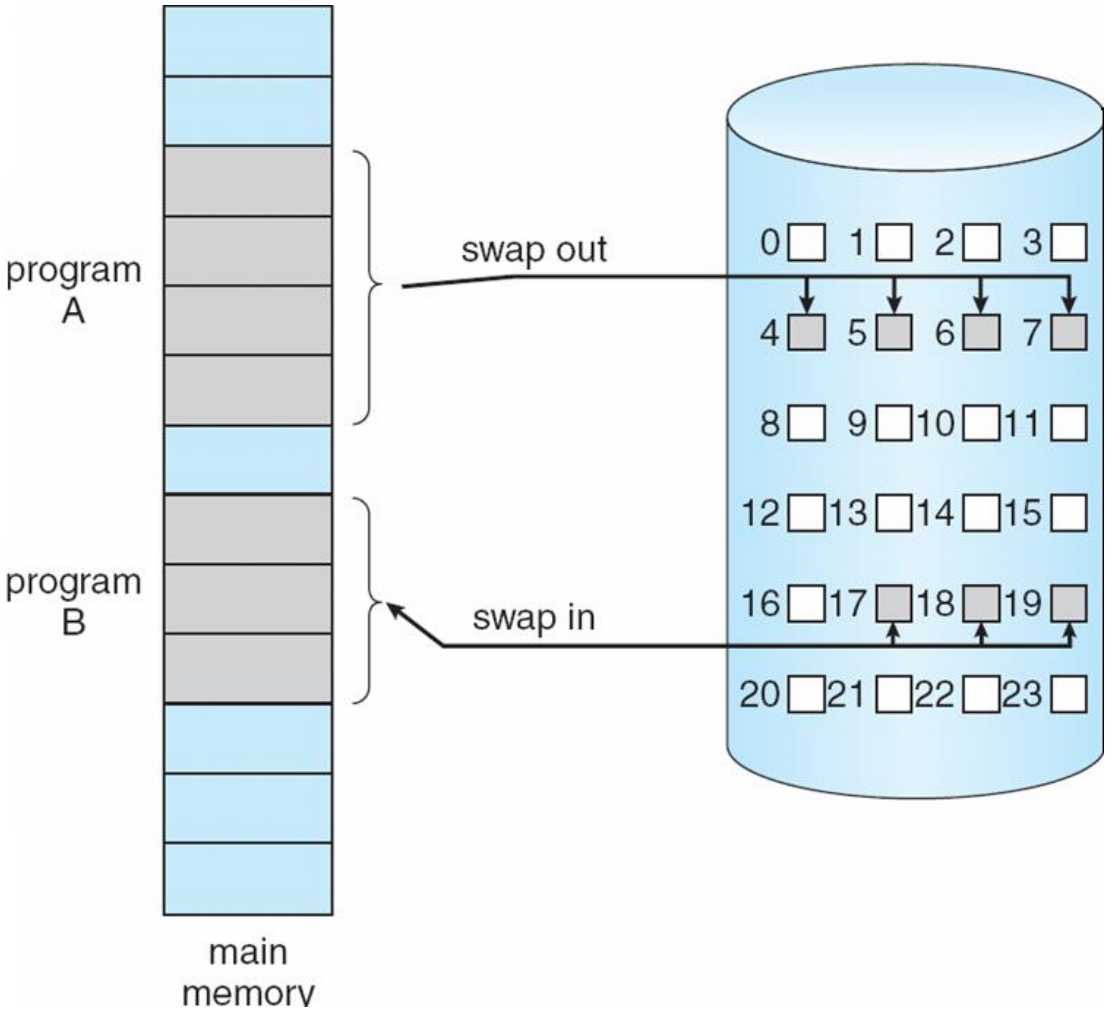
This process is known as *paging* or *swapping* and the temporary storage space on the hard disk is called a page file or a swap file.

VIRTUAL MEMORY

- Separation of user logical memory from physical memory.
- Program would no longer be constrained by the amount of physical memory that is available.
- Because each program takes less space, more program could be run at the same time.

- Less I/O would be needed to load or swap each user program into memory, Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Transfer of a Paged Memory to Contiguous Disk Space



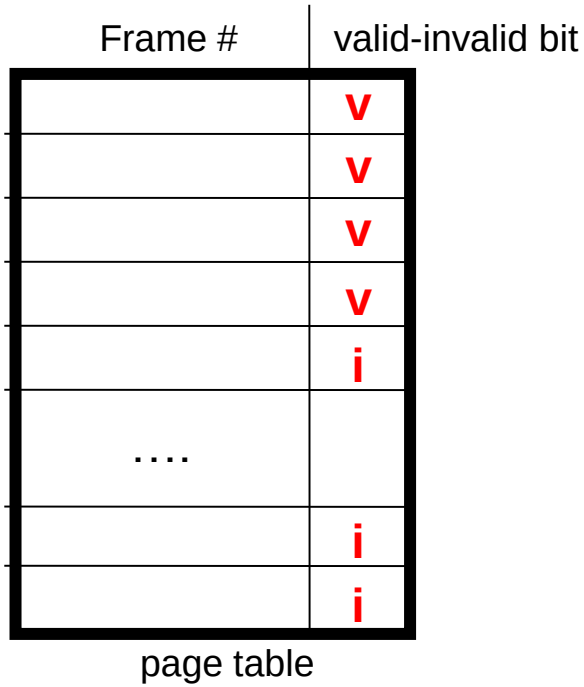
Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed $\Rightarrow$ reference to it
  - not-in-memory $\Rightarrow$ bring to memory
- **Lazy swapper** – never swaps a page into memory unless page will be needed
  - Swapper that deals with pages is a **pager**

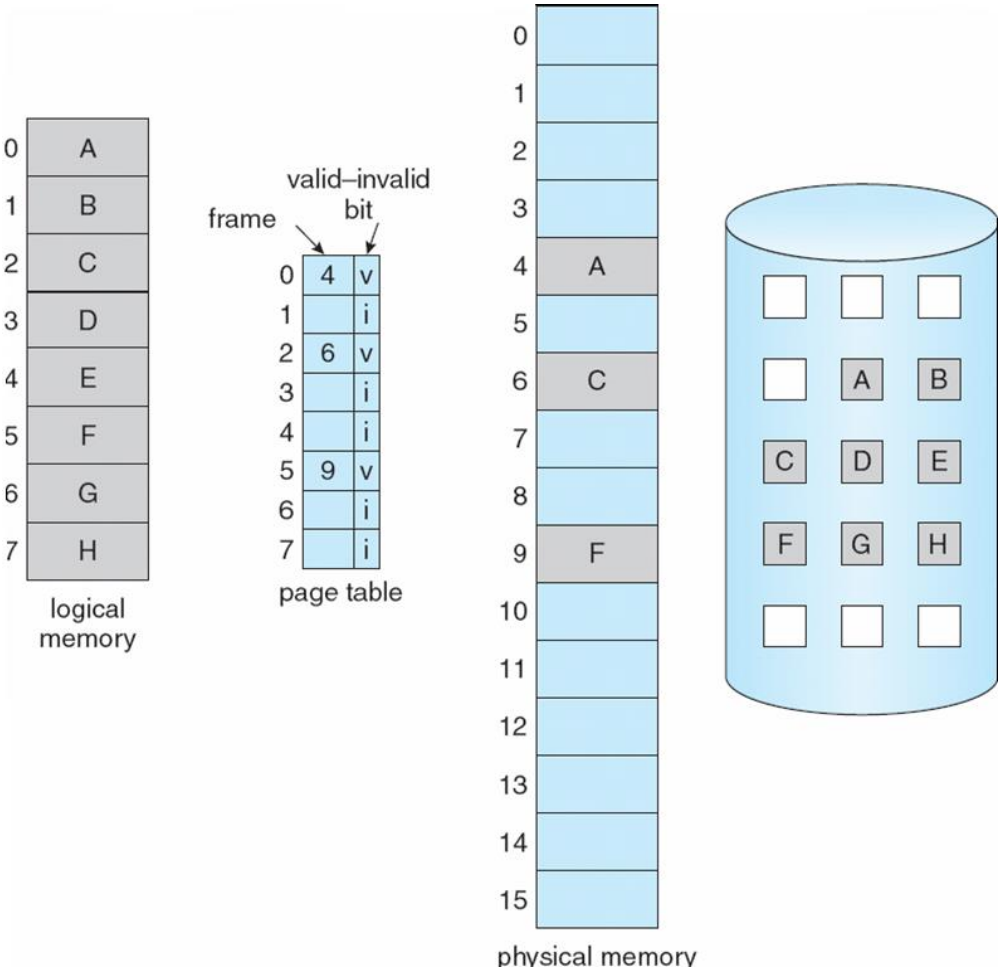
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**v** ⇒ in-memory, **i** ⇒ not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries
- Example of a page table snapshot:



During address translation, if valid–invalid bit in page table entry is **I** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory

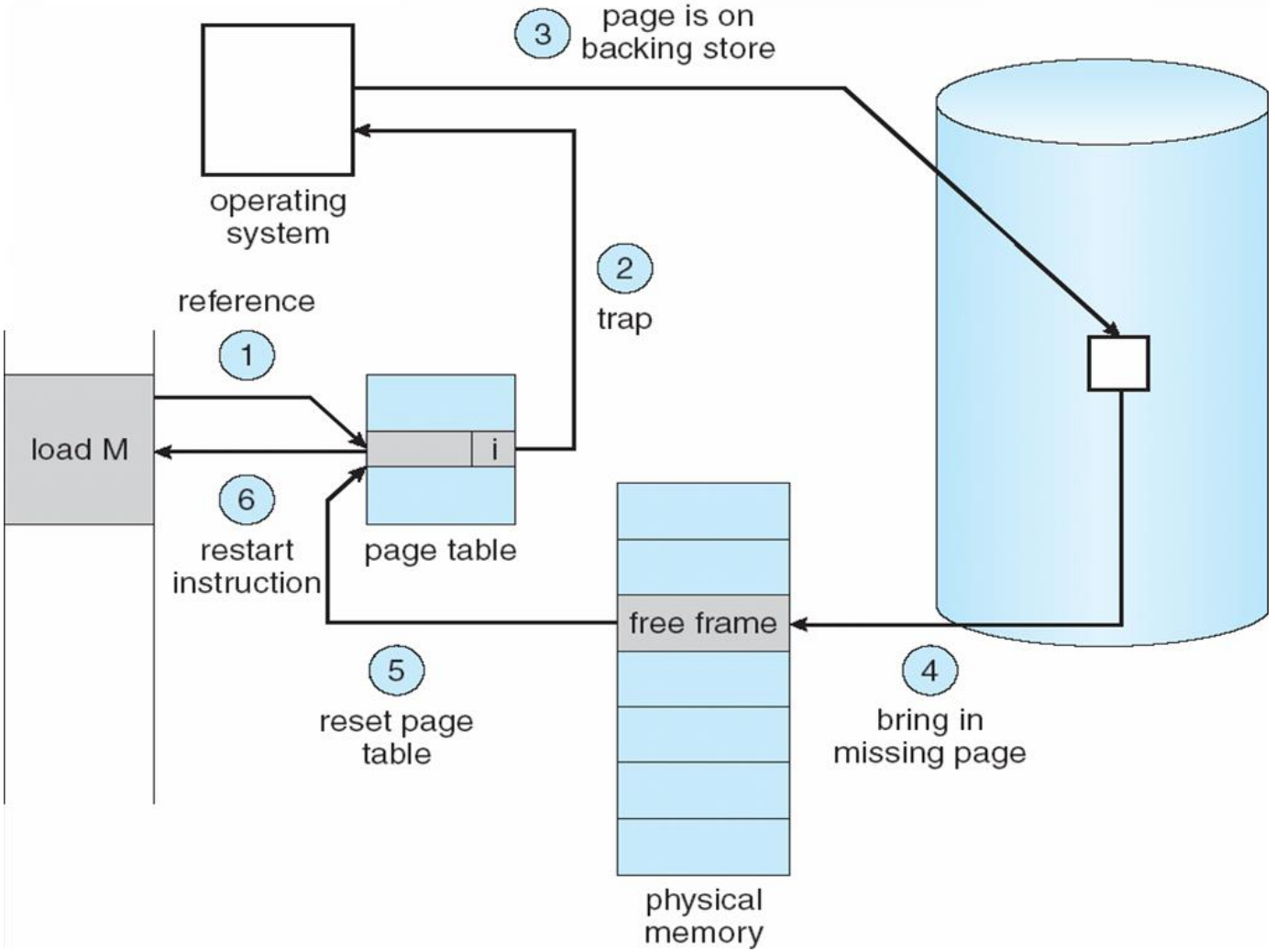


Page Fault

- If the required page is not available in Physical Memory: **Page Fault**
- If there is a reference to a page, first reference to that page (if page is not available in physical memory) will trap to operating system: **page fault**

1. Get empty frame
2. Swap page into frame
3. Reset tables
4. Set validation bit = **v**
5. Restart the instruction that caused the page fault

Steps in Handling a Page Fault



What happens if there is no free frame?

- Page replacement – find a page in memory and swap it out
  - Replacement algorithm selects the victim page
  - want an algorithm which will result in minimum number of page faults

Basic Page Replacement

1. Find the location of the desired page on disk

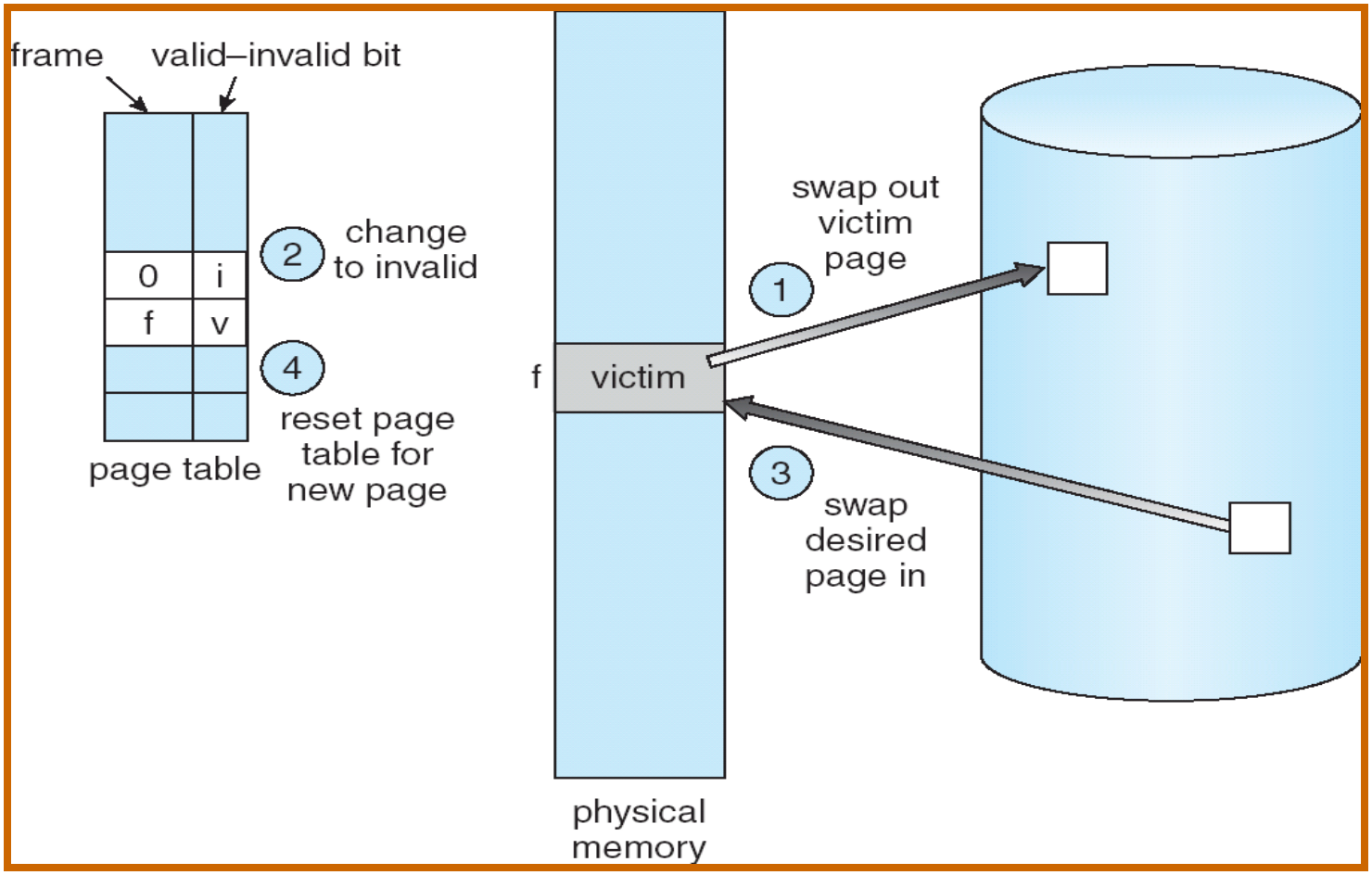
2. Find a free frame:
   - If there is a free frame, then use it
   - If there are no free frames, use a page replacement algorithm to select a **victim** frame

3. Bring the desired page into the (newly) freed frame, and update the page and frame tables

4. Restart the interrupted instruction of the process

Page Replacement

- Use **modify (dirty) bit** to reduce overhead of page transfers
  - If victim page is unmodified, then no need to write it back to disk

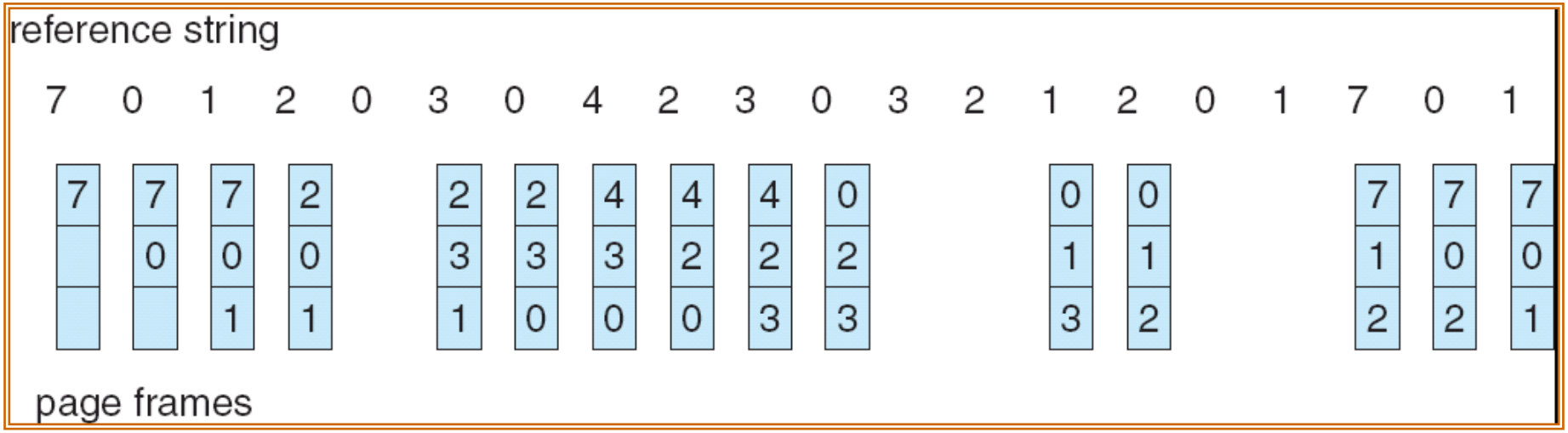


Page Replacement Algorithms

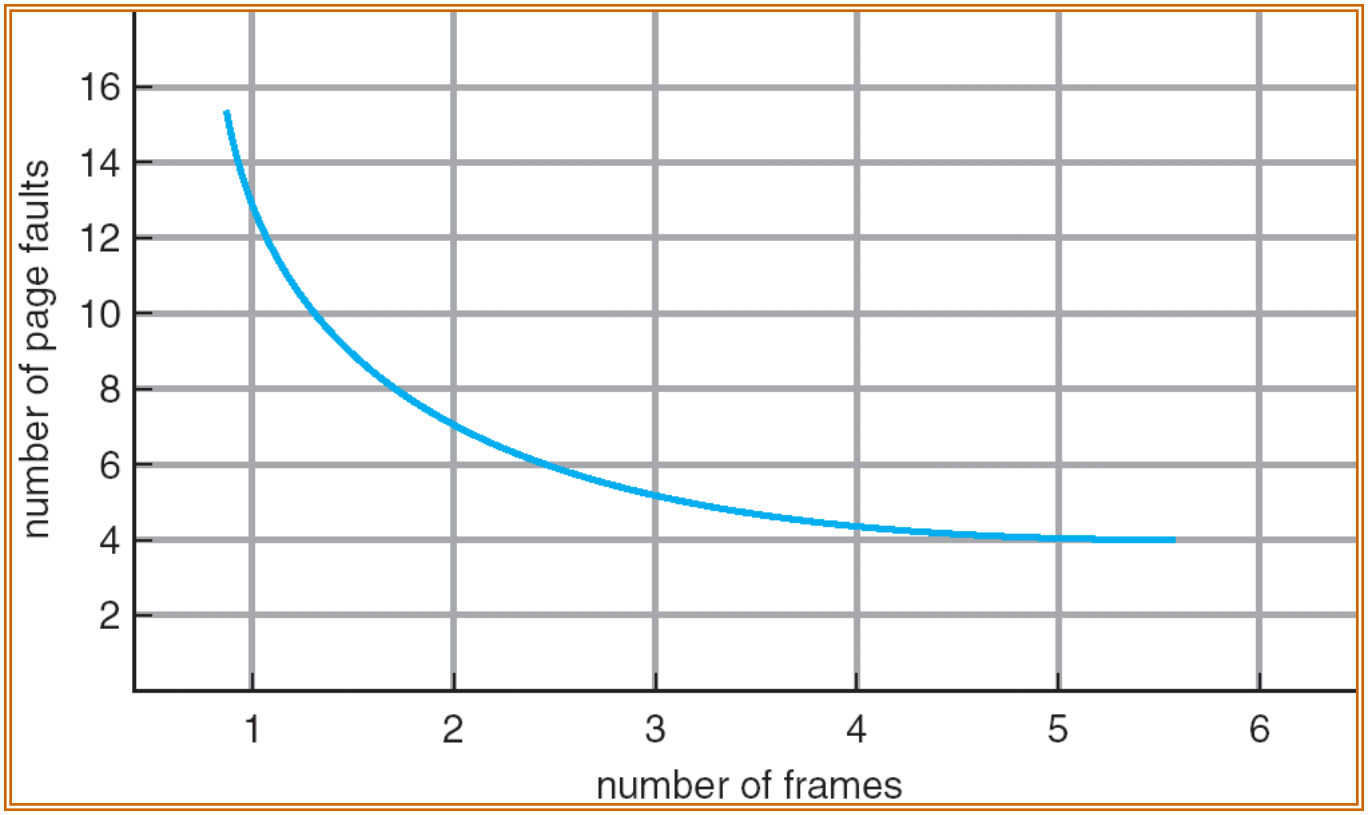
- Want lowest page-fault rate
  - Subsequently low page-fault service time too

- Evaluate algorithm by running it on a particular fixed string of memory references (reference string) and computing the number of page faults on that string
- FIFO page replacement
- Optimal page replacement
- LRU page replacement

FIFO Page Replacement

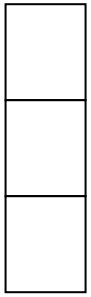


Graph of Page Faults vs Number of Frames

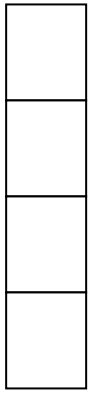


First-In-First-Out (FIFO) Algorithm

- Reference string (12 total refs / 5 unique): 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory per process)

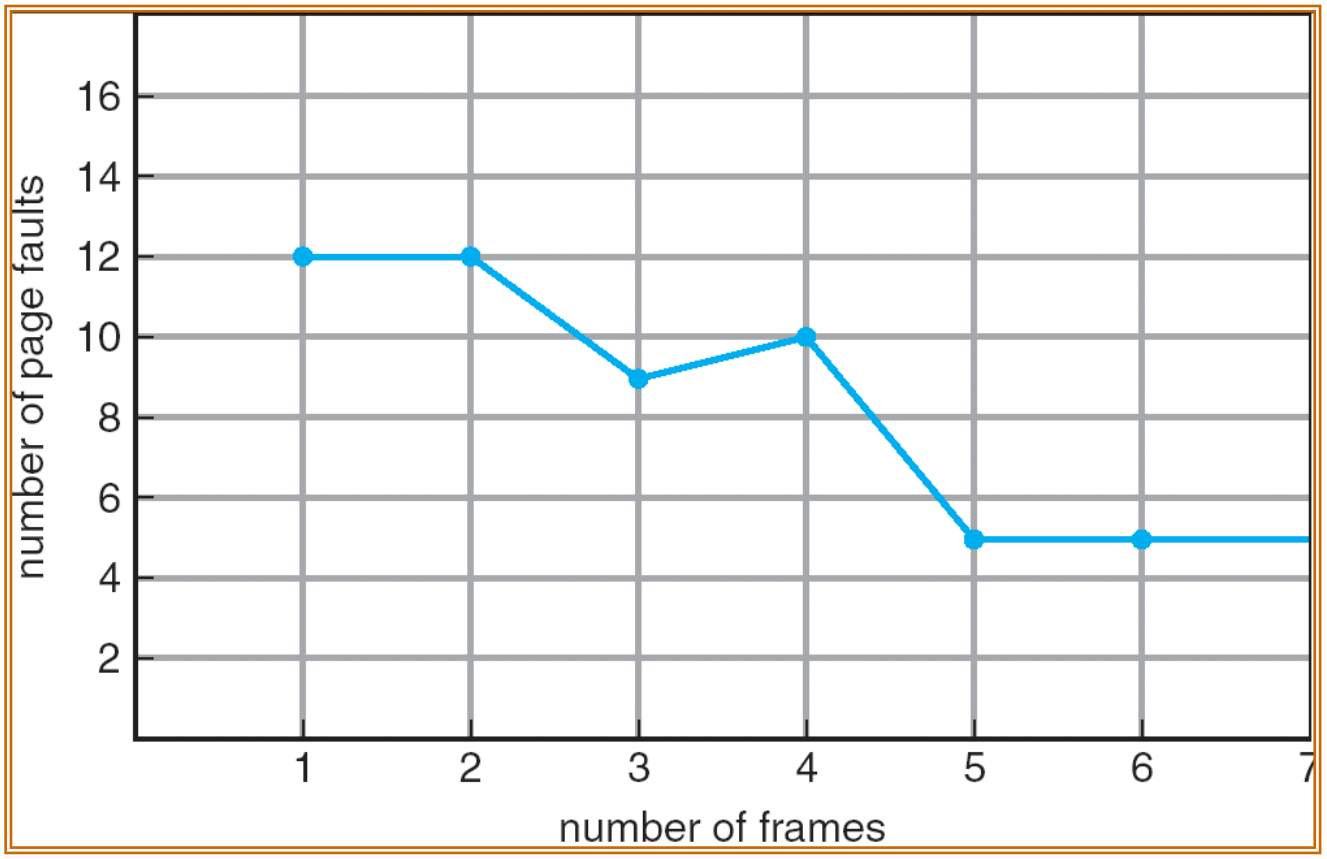


- 4 frames (3 pages can be in memory per process)



- Belady's Anomaly: More frames may yield more page faults

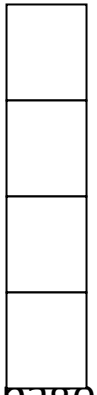
FIFO Illustration of Belady's Anomaly



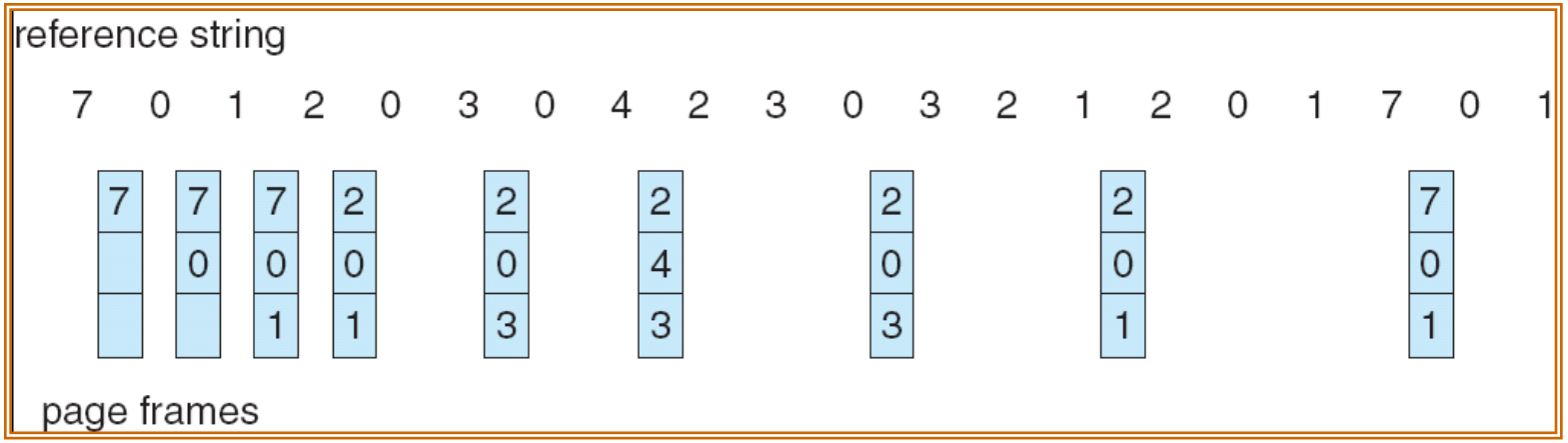
Optimal Algorithm (Called OPT or MIN)

- Replace page that will not be used for longest period of time in future
- 4 frames example

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



- How do we know which page will be used latest?
  - We don't! OPT is impossible to implement accurately unless the order of page references is known *a priori*



Reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14.

The reference to page 3 replaces page 1, as page 1 will be the last of the 3 pages in the memory to be referenced again.

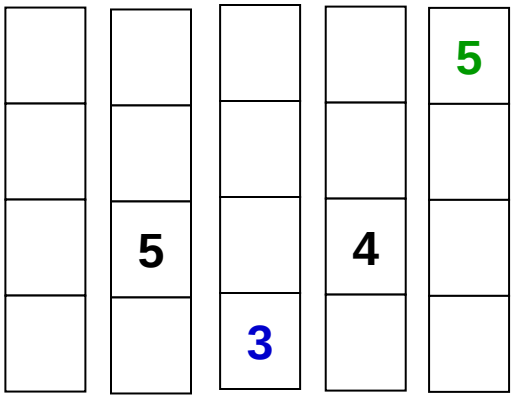
With only 9 page faults, OPT is much better than FIFO algo. Which results in 15 faults.

LRU replacement

- The key distinction between FIFO and OPT(other than looking backward versus forward in time) is that FIFO algo uses the time when a page was brought into the memory whereas Opt algo uses the time when a page is to be used.
- If **we use recent past as an approximation of the near future, thus we can replace a page that has not been used for the longest period of time in past.**
- This approach is LEAST RECENTLY USED(LRU) algorithm.
- LRU replacement associates with each page the time of the page's last use. When a page is to be replaced, LRU chooses the page that has not been used for the longest period of time.

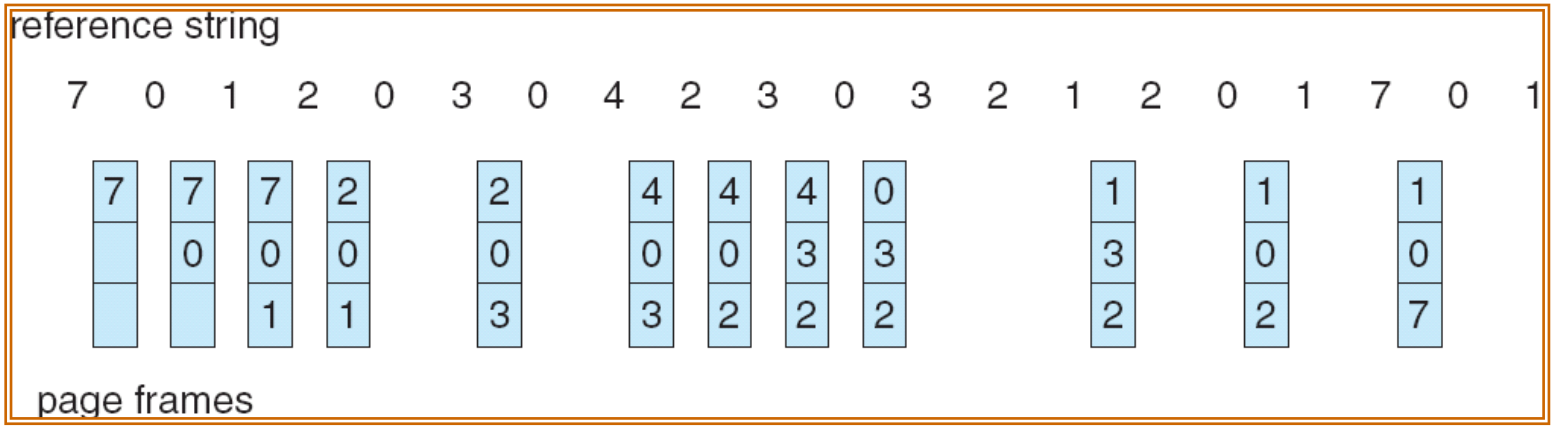
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**



- Considered good, but difficult to implement
- LRU does not suffer from Belady's anomaly

LRU Page Replacement

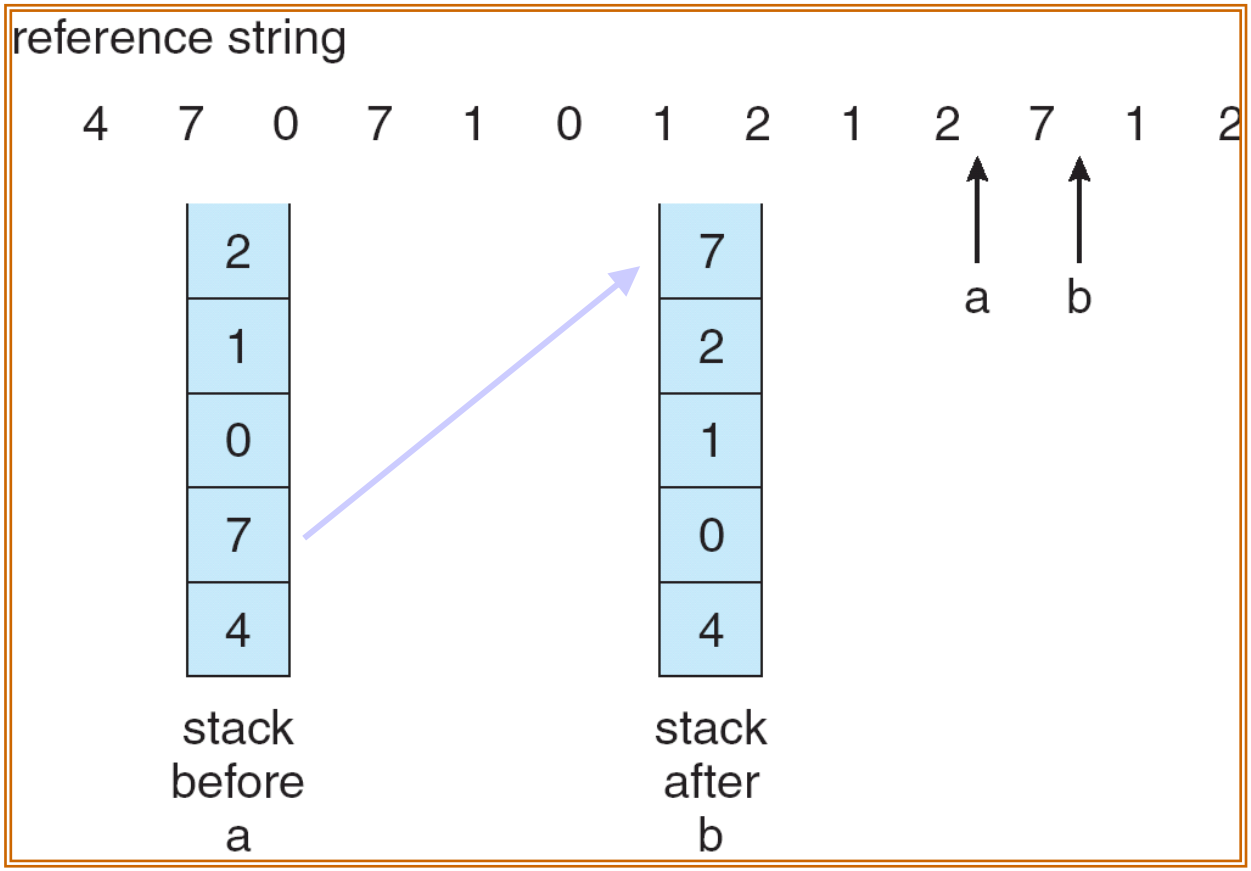


LRU Algorithm – Implementations

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock value into the counter
  - When a page needs to be replaced, linear search the counters and select the minimum value as the victim

- Stack implementation
  - Keep a stack of page numbers in a doubly-linked form:
  - Page referenced:
    - move it to the top

Stack Records Most Recent Page References



LRU Approximation Algorithms

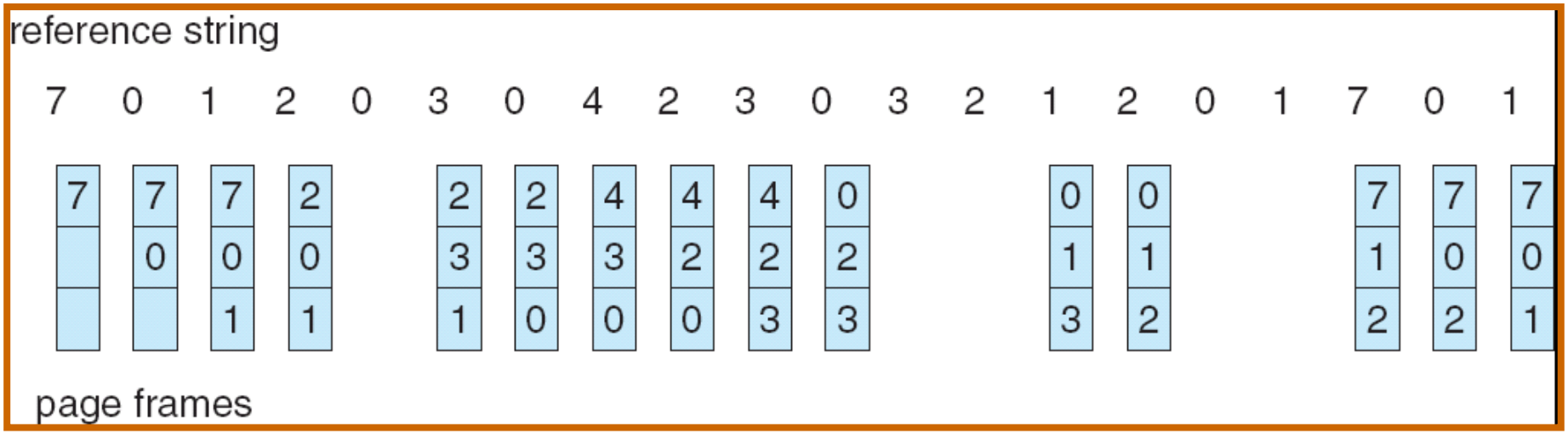
- Few computers provide sufficient hardware support for true LRU page replacement. Some systems provide no hardware support, and other page replacement algorithms must have to be used. Many systems provide help, however, in the form of reference bit.
- Reference bit (modified by hardware)
  - With each page associate a reference bit, initially = 0
  - When page is referenced, set reference bit to 1
  - Replace *any* victim page with reference bit = 0
    - The replacement sequence is partially ordered
- Can be enhanced to have a sequence of reference bits

LRU Approximation Algorithms

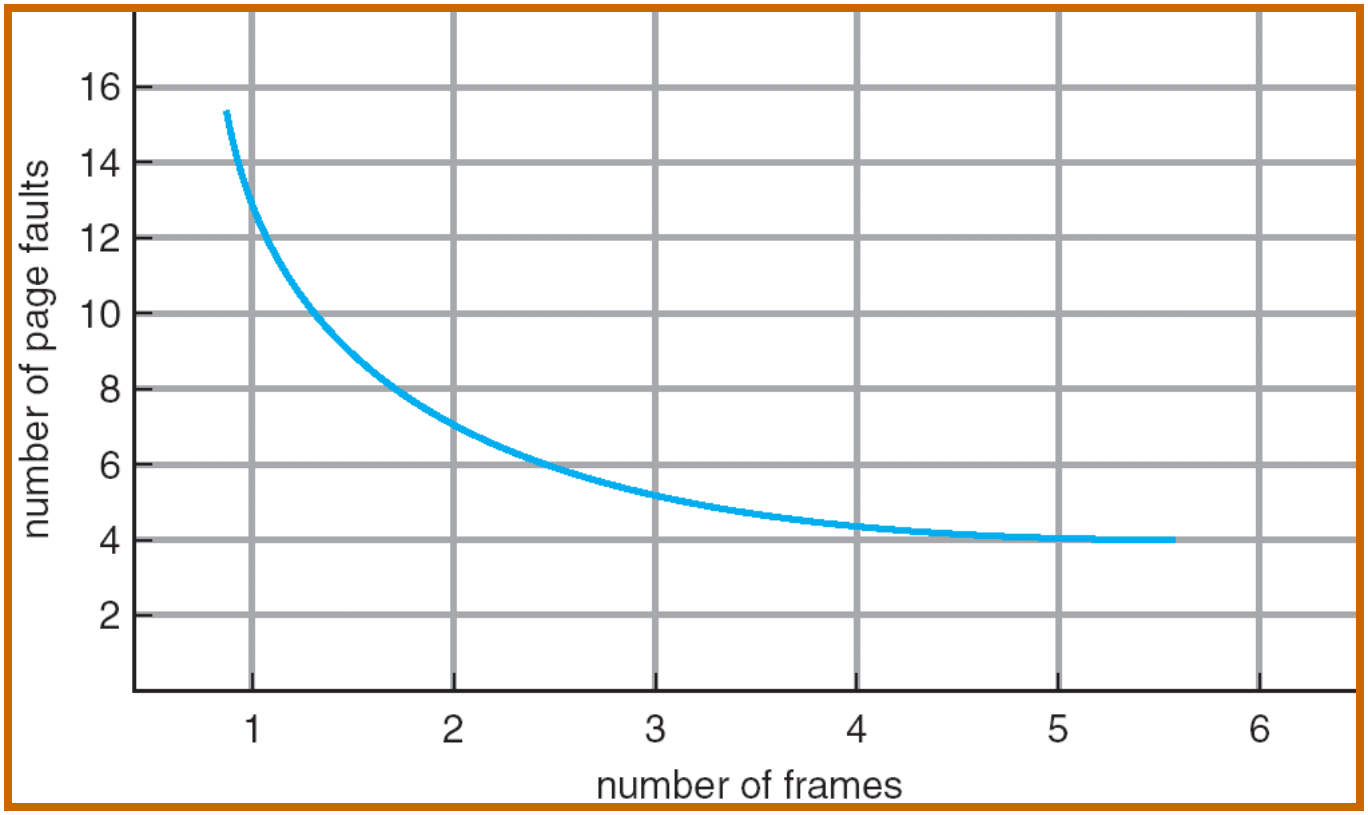
- Second chance
  - Need reference bit
  - Treat pages as a circular queue using clock replacement
  - If the page to be replaced has bit 0 then replace that page.
  - If page to be replaced (in clock order) has bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

- Can be enhanced be including a modify (dirty) bit as well
  - (0,0) – neither recently used, nor modified (best choice)
  - (0,1) – not recently used, but modified (need write-back)
  - (1,0) – recently used, but not modified (might need soon)
  - (1,1) – recently used and modified (worst choice)

FIFO Page Replacement



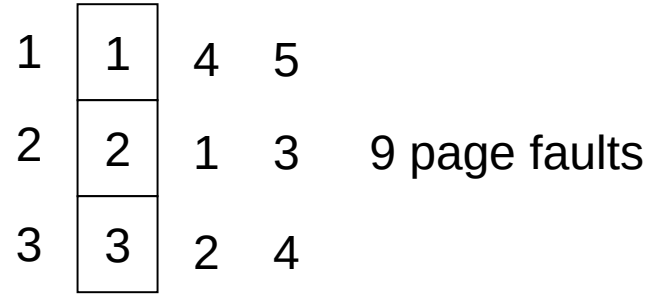
Graph of Page Faults vs Number of Frames



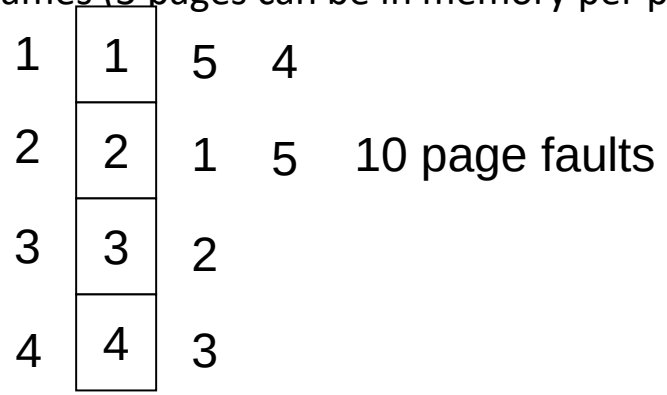
Expected behavior of a good page replacement algorithm.

First-In-First-Out (More frames doesnot decrease page fault

- Reference string (12 total refs / 5 unique): **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- 3 frames (3 pages can be in memory per process)

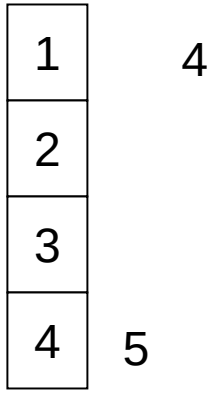


- 4 frames (3 pages can be in memory per process)



Optimal Algorithm (Called OPT or MIN)

We don't! OPT is impossible to implement accurately unless the order of page references is known *a priori*

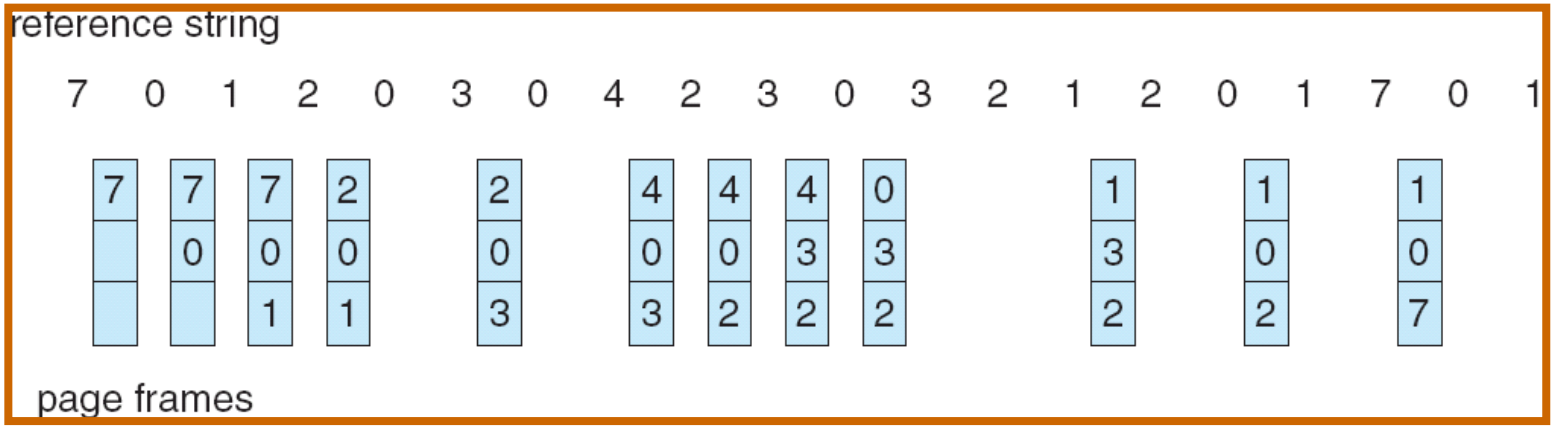


6 page faults

LRU replacement

- **Replace a page that has not been used for the longest period of time.**
- This approach is LEAST RECENTLY USED(LRU) algorithm.
- LRU replacement associates with each page the time of the page's last use. When a page is to be replaced, LRU chooses the page that has not been used for the longest period of time.

LRU Page Replacement Example



Thrashing

- **Thrashing** - a process is busy swapping pages in and out
- **This high paging activity is called thrashing.**

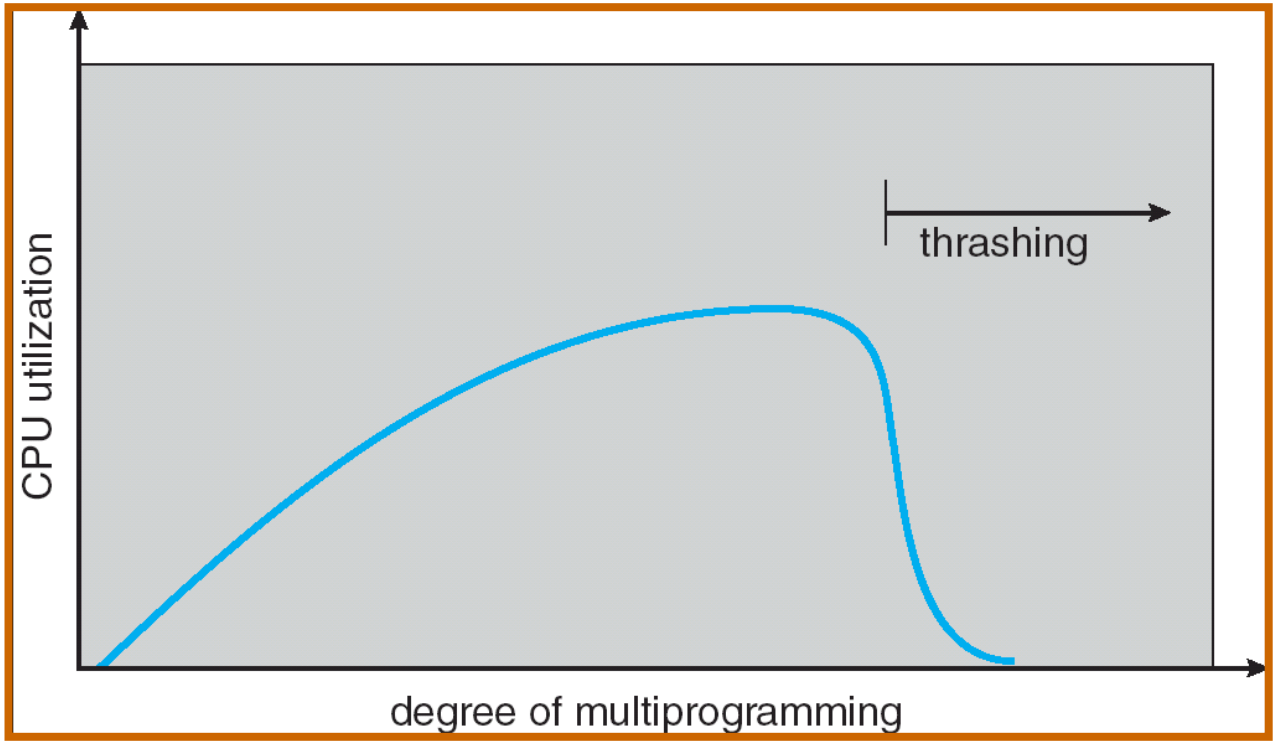
- If a process does not have "enough" pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system

- **Thrashing** ≡ a process is busy swapping pages

Thrashing

- **Thrashing** occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk.
- If a process does not have “enough” pages in memory, it will quickly page fault, at this point it must replace some page.
- Since all the pages are active in use, it must replace a page that will be needed again, may be after some time.
- This result to occurrence of Page fault
- This leads to:
  - Low CPU utilization and low throughput
  - OS attempts to increase the degree of multiprogramming
  - Another process added to the system
  - Even more page faults ensue.

Thrashing



End of Chapter 9