Virtual Memory

Background

Need of Virtual memory : in many cases, the entire program is not needed.

1. Programs often have code to handle unusual error conditions. Since these errors seldom, this code is almost never executed.
2. An array may be declared 100 × 100 elements, even though it is but used only 10 × 10 elements.
3. An assembler symbol table may have room for 3,000 symbols, although the average program has less than 200 symbols.

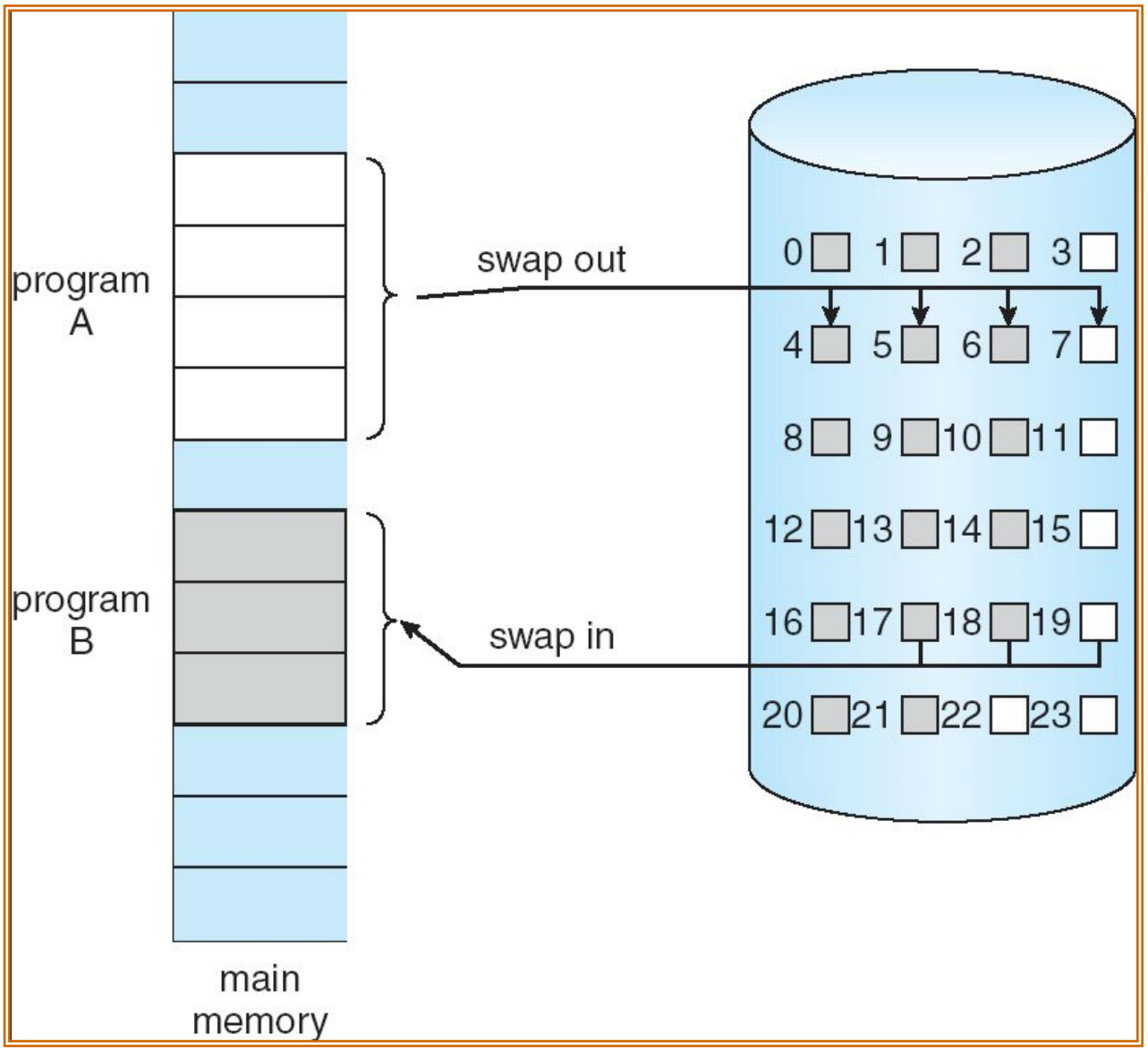
Demand Paging: paging with swapping

- Process is a sequence of pages hence pager swaps in and out pages from disk space.
- Bring a page into memory only when it is needed.
- Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

Demand Paging: paging with swapping

- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

Access to page marked invalid causes a page-fault trap.

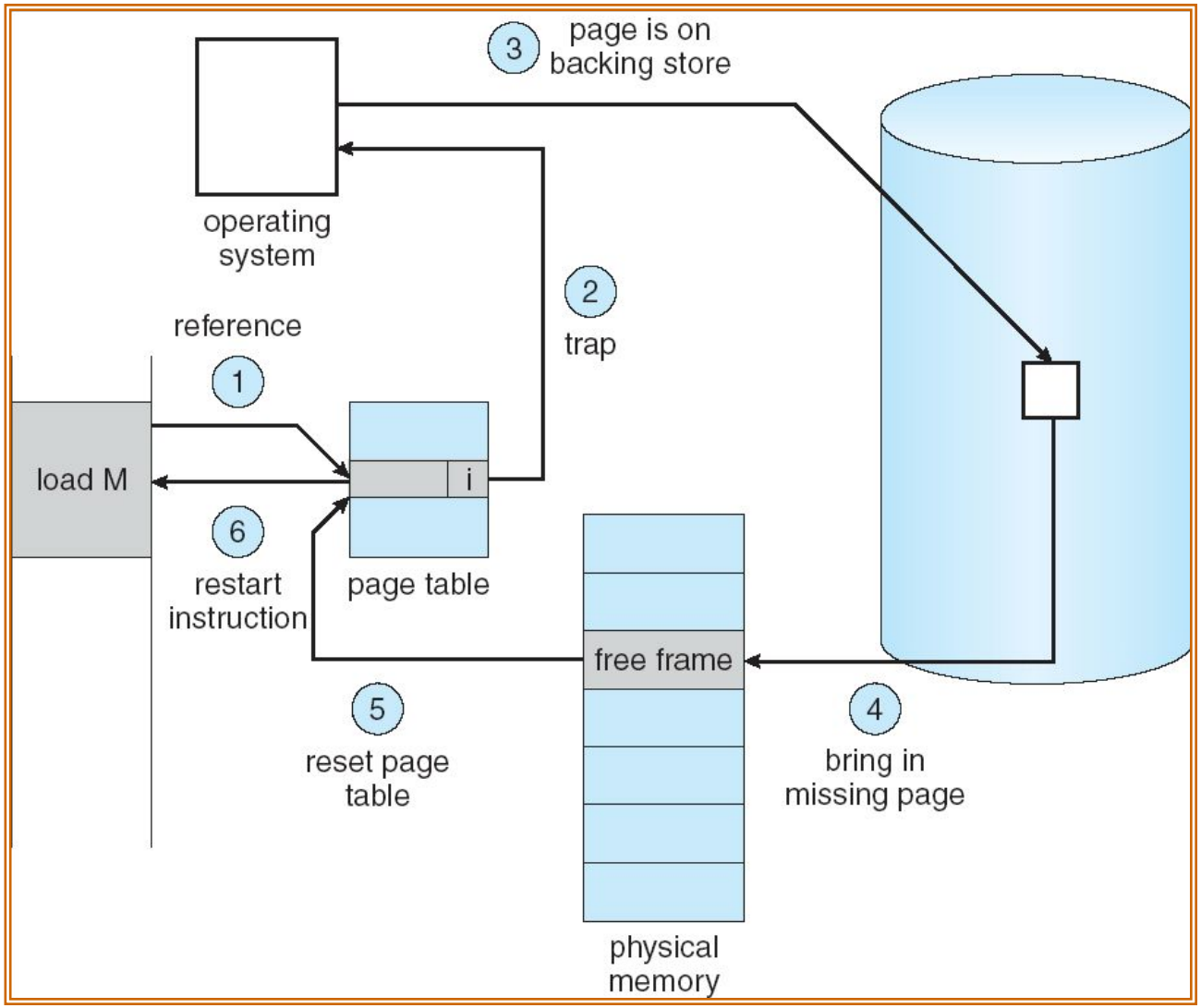
1. When bit is set to “valid” indicates the associated page is both legal and in memory.
2. If set to invalid, indicates either is page not valid (not present in logical space) or is valid but currently on disk.

Page Fault: steps to handle page fault

- If there is a reference to a page that was not brought into memory, first reference to that page will trap to operating system: **page fault**

1. Check internal table, to determine whether reference was valid or invalid.
2. If the reference was invalid, terminate the process. If it was valid, but not yet brought in memory, then bring page in.
3. Get empty frame from the free-frame list
4. Swap page into frame .
5. Reset tables , Set validation bit = **v .**
6. 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 some page in memory, but not really in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times.

Basic Page Replacement

1. Find the location of the desired page on disk.

2. Find a free frame:

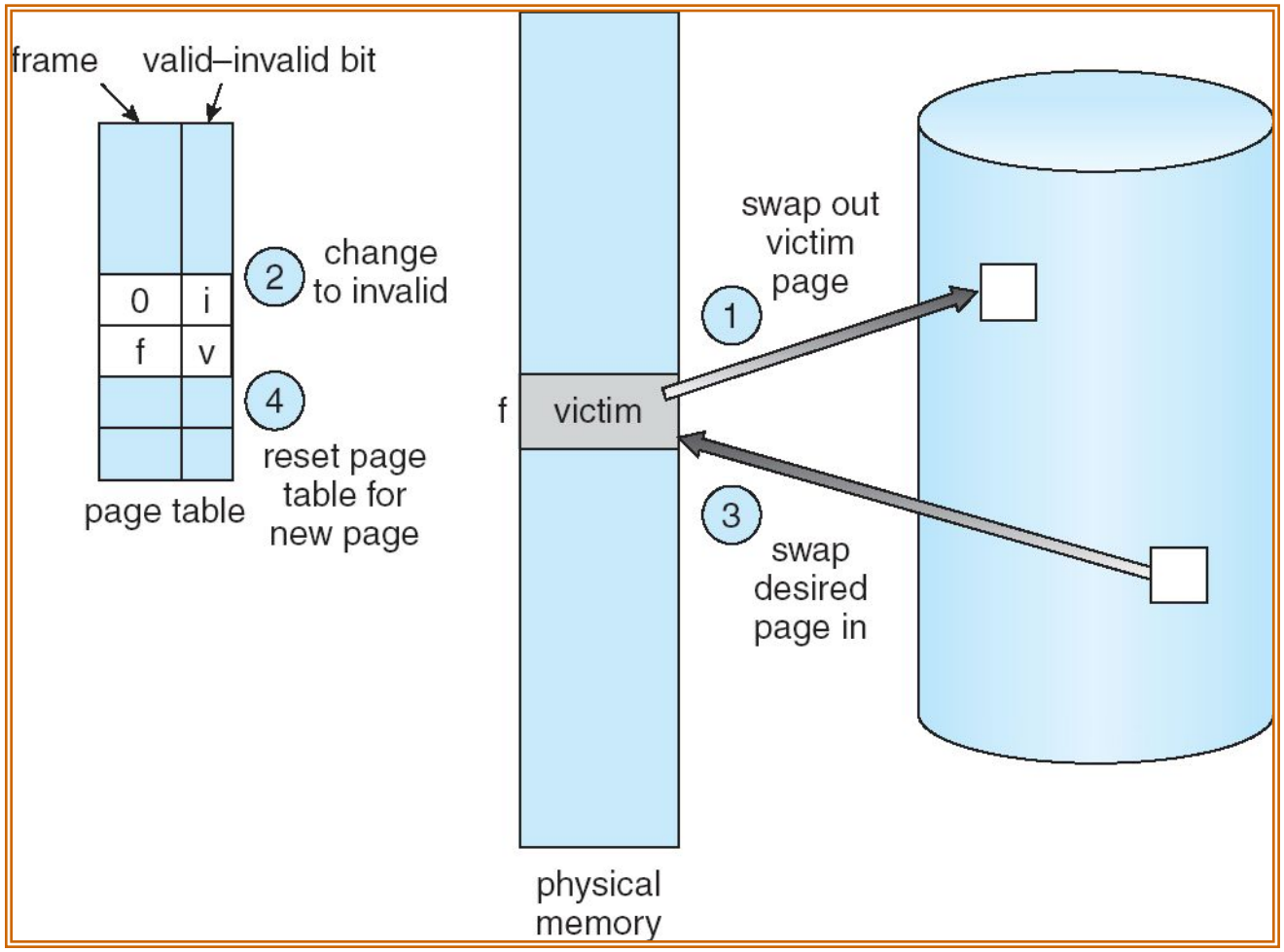
   - If there is a free frame, use it.

   - If there is no free frame, use a page replacement algorithm to select a *victim* frame.

3. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process.

Page Replacement



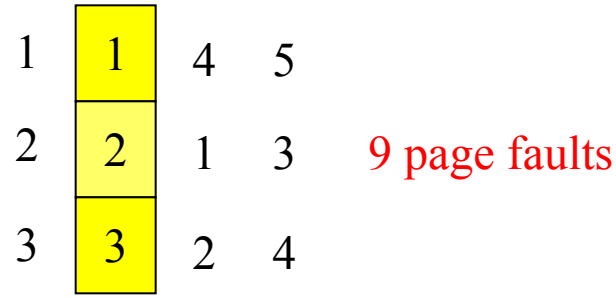
Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- The reference string is

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

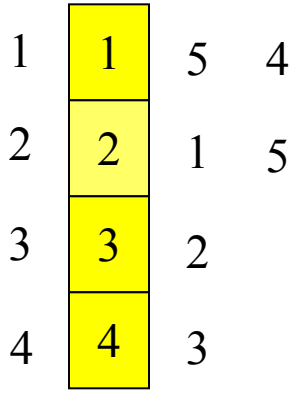
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

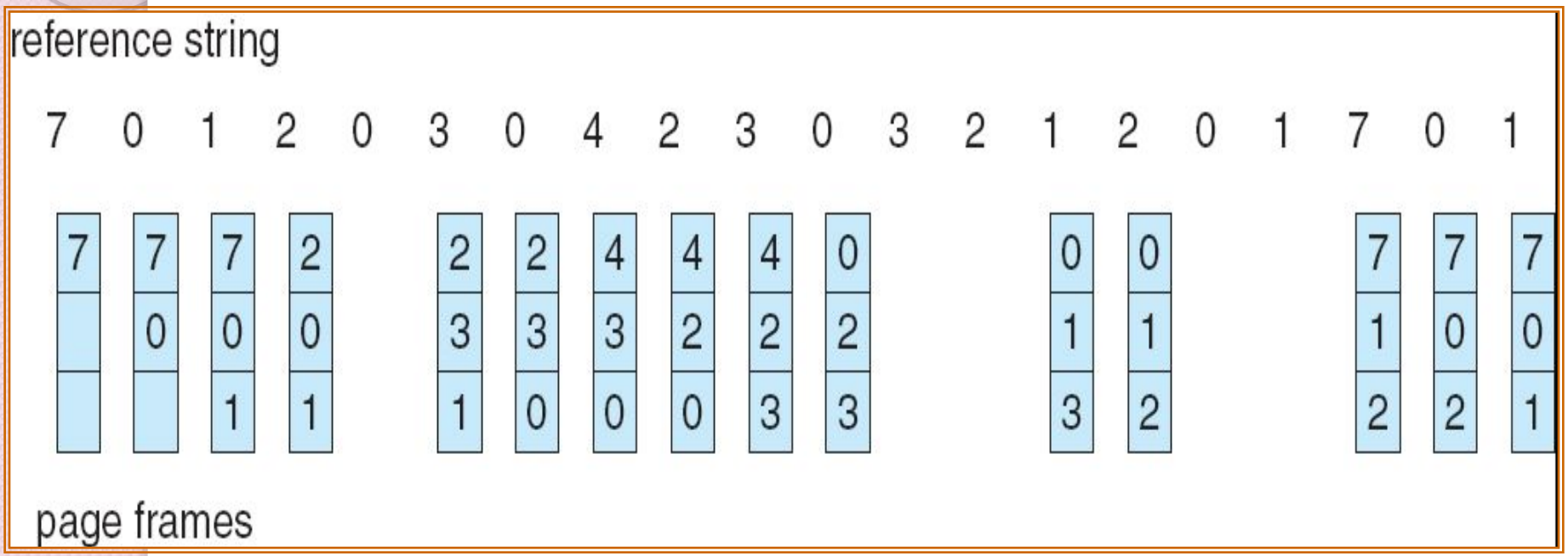
- 4 frames



10 page faults

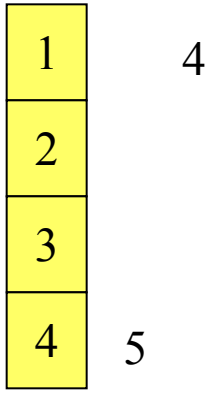
- **Belady's Anomaly:** more frames ⇒ more page faults

FIFO Page Replacement



Optimal Algorithm

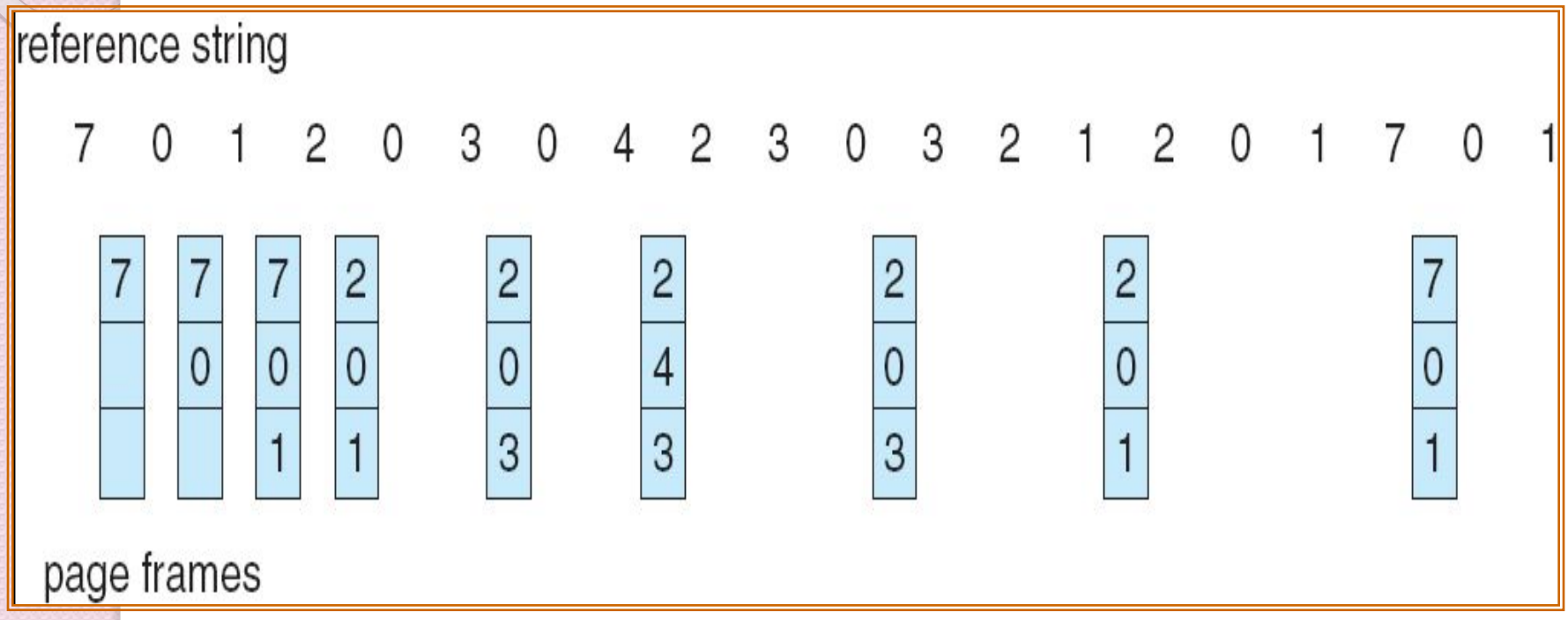
- Replace page that will not be used for longest period of time
- 4 frames example : **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**



6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



Least Recently Used (LRU) Algorithm

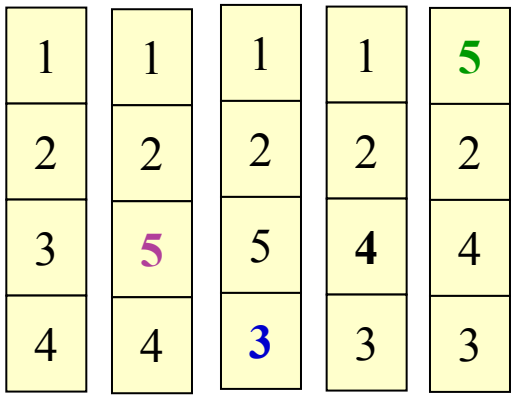
- LRU replacement associates with each page the time of that page last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- The major problem is how to implement LRU replacement.
- Two implementations are feasible:

    1. counter implementation
    2. stack implementation.

Least Recently Used (LRU) Algorithm

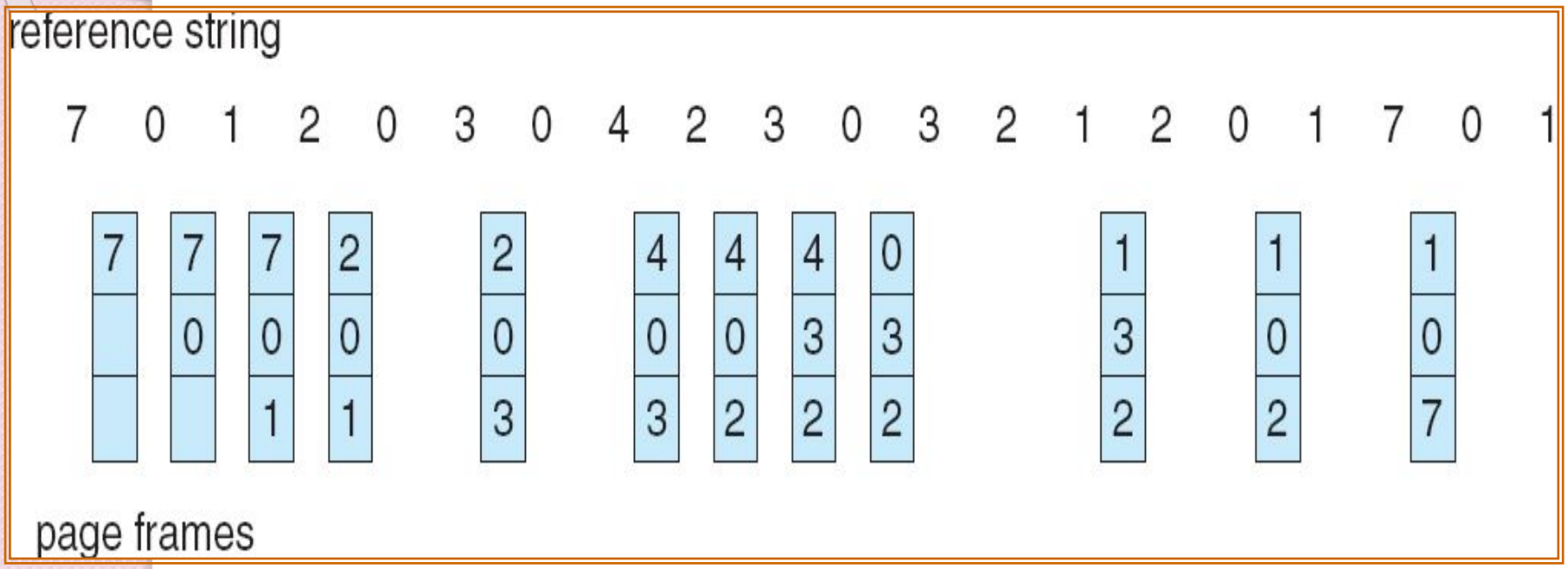
- Reference string:

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



- Note: both implementation of LRU require hardware support.
- The updating of the clock fields or stack must be done for *every memory reference.*
- If we were to use an interrupt for every reference to allow software to update such data structures, it would slow every memory reference

LRU Page Replacement

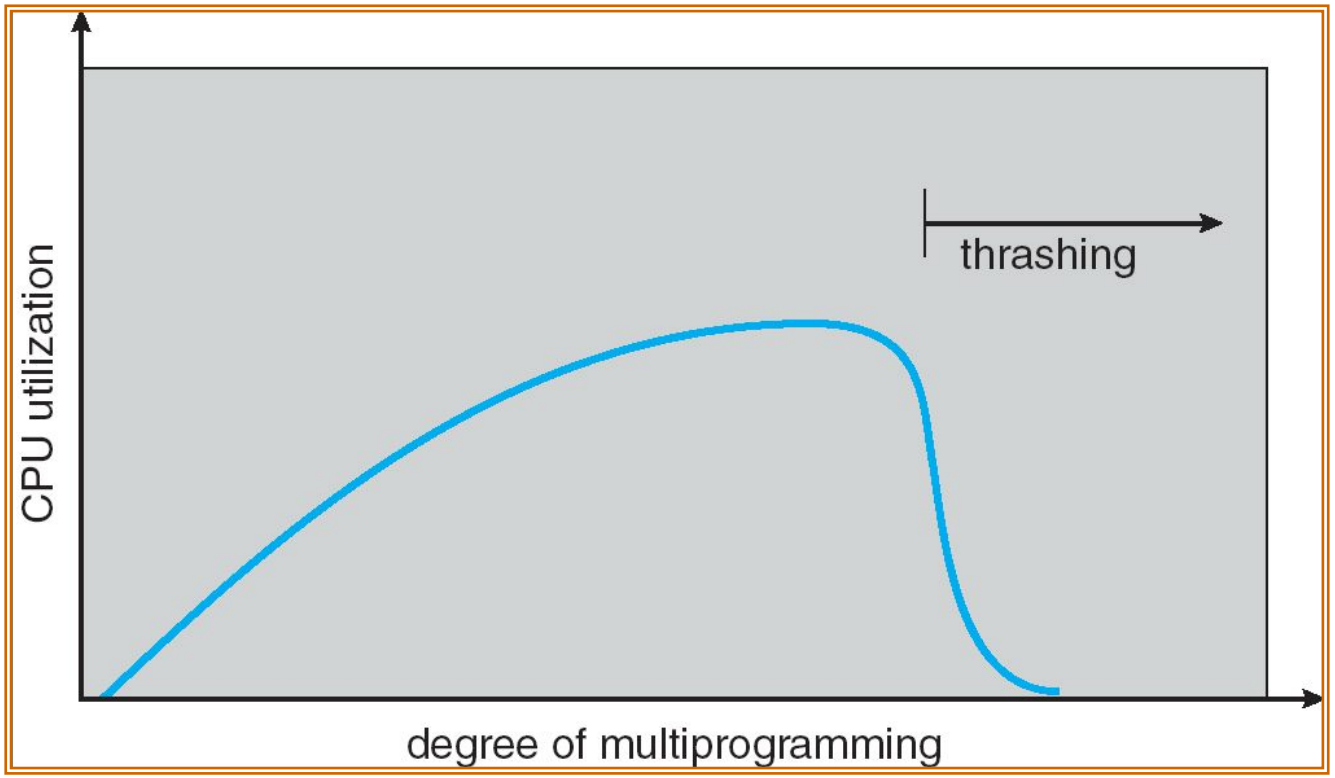


Thrashing : high paging activity

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- **Thrashing** ≡ a process is busy swapping pages in and out (than execution)

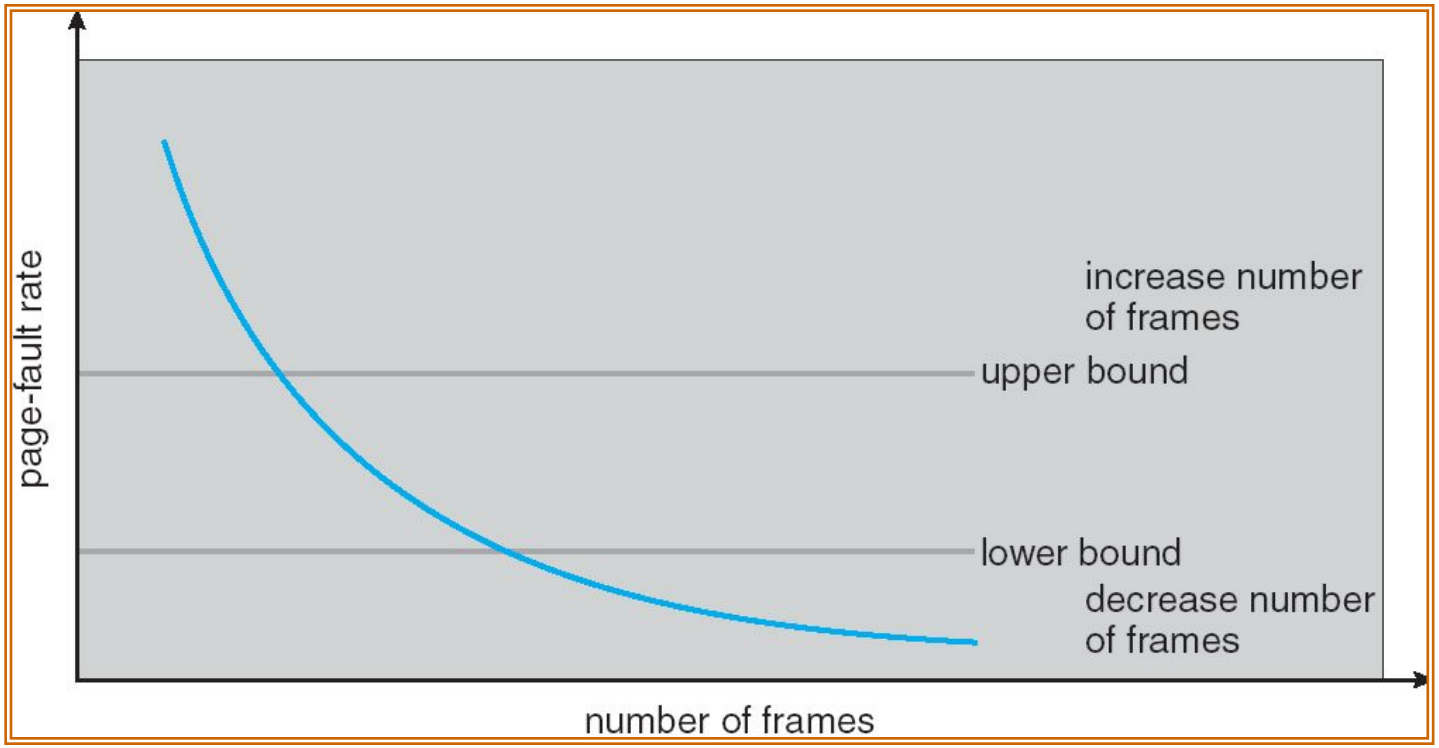
Thrashing (Cont.)



CPU utilization *increases* with degree of multiprogramming slowly, until maximum, if degree of programming increased than that CPU utilization suddenly drops.

Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Difference between paging and Segmentation

Paging	Segmentation
A page is a contiguous range of memory addresses which is mapped to physical memory.	A segment is an independent address space. Each segment has addresses in a range from 0 to maximum value.
It has only one linear address space	It has many address spaces
Procedures and data cannot be separated	Procedures and data can be separated
Procedures cannot be shared between users	Procedures can be shared between users
Procedures and data cannot be protected separately	Procedures and data can be protected separately
Compilation cannot be done separately A page is a fixed size	Compilation can be done separately Segment is of arbitrary size.
A page is a physical unit	A segment is a logical unit