Memory Management

Organized By: *Vinay Arora*

Assistant Professor

CSED, TU

Disclaimer

This is NOT A COPYRIGHT MATERIAL

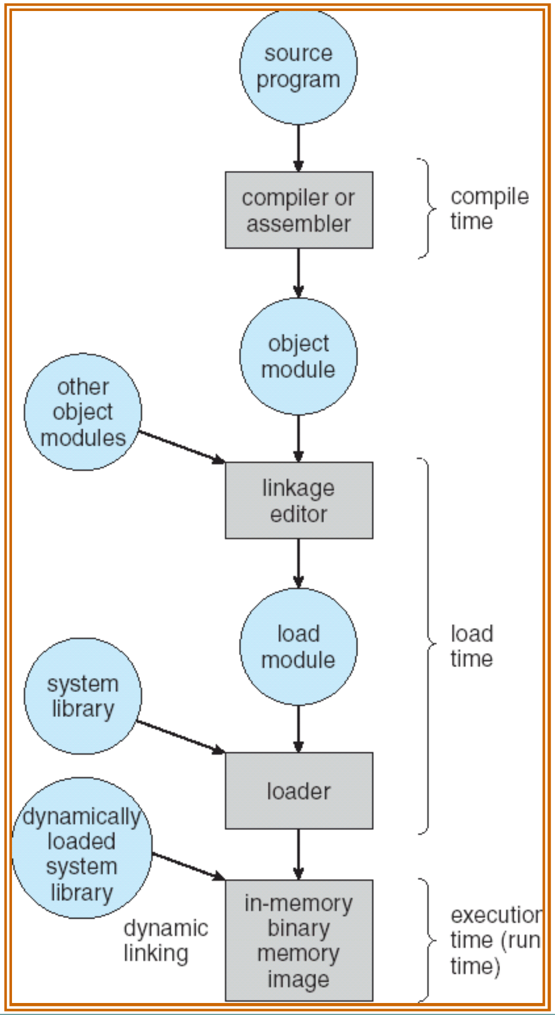
<u>Content has been taken mainly from the following books</u>:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings
www.os-book.com
www.cs.jhu.edu/~yairamir/cs418/os2/sld001.htm
www.personal.kent.edu/~rmuhamma/OpSystems/os.html
http://msdn.microsoft.com/en-us/library/ms685096(VS.85).aspx
http://www.computer.howsttuffworks.com/operating-system6.htm
http://williamstallings.com/OS/Animations.html
Etc…

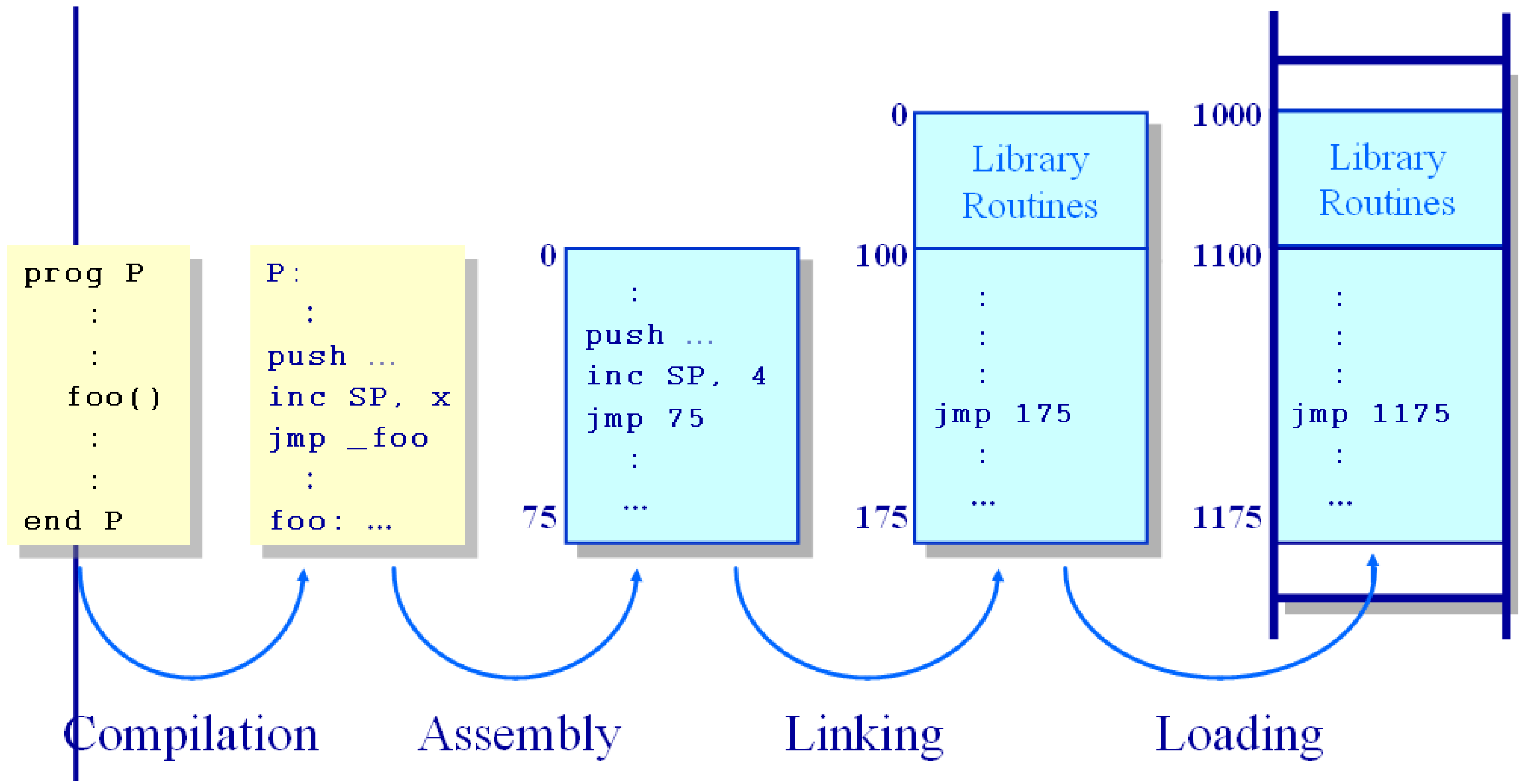
Introduction

- Program must be brought into memory and placed within a process for it to be run
- *Input Queue* – collection of processes on the disk that are waiting to be brought into memory to run the program
- User programs go through several steps before being run

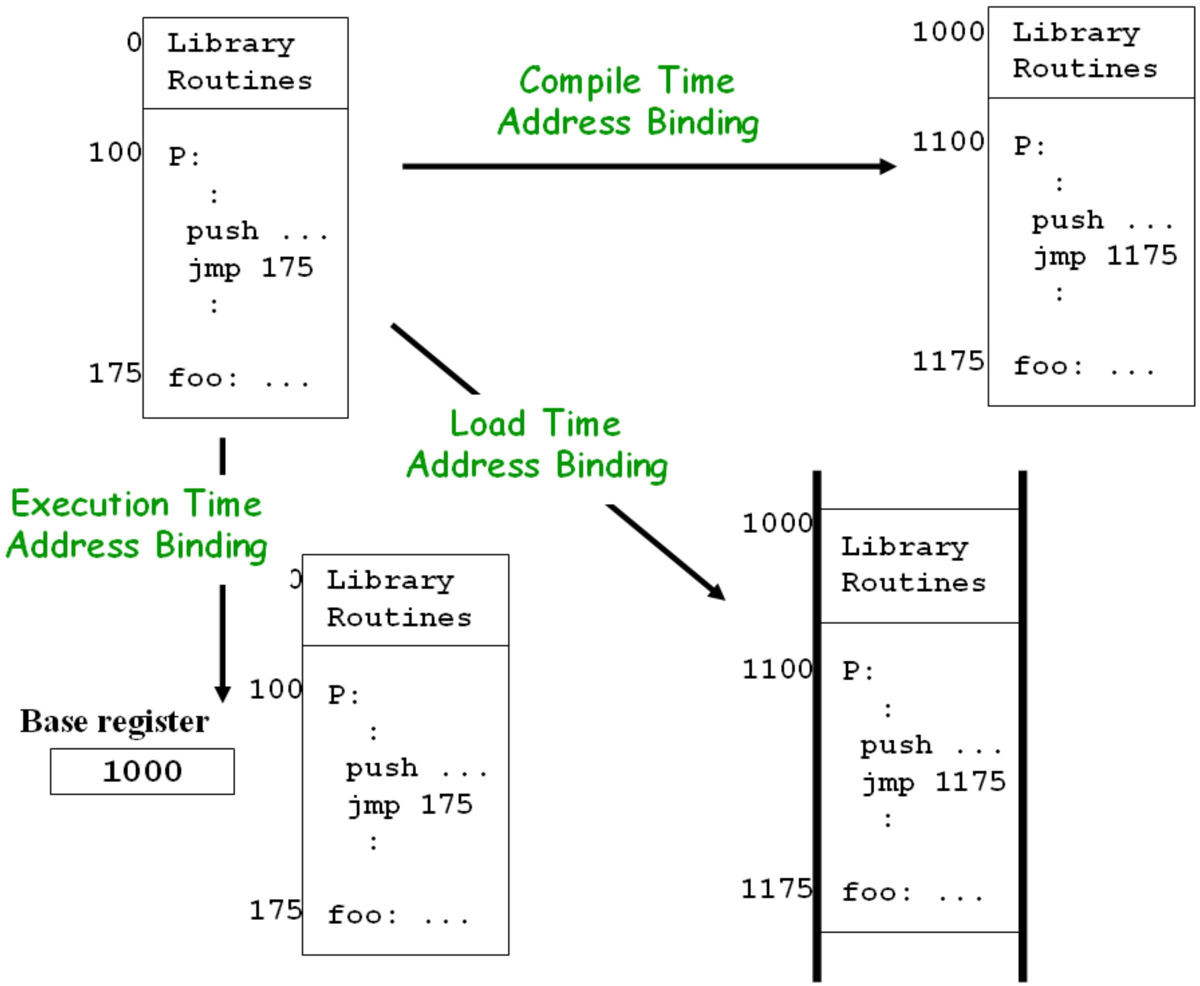
Multi-step Processing of User Program



Compilation Pipeline



Address Binding

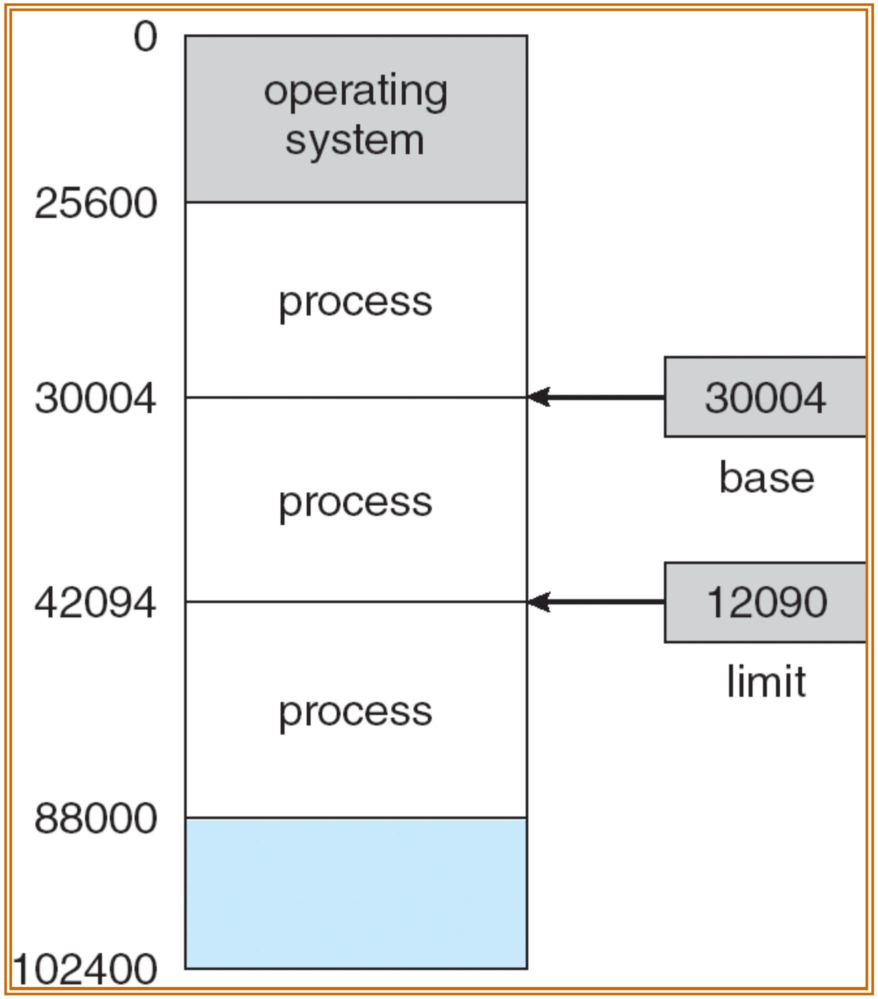


Logical/Physical Address

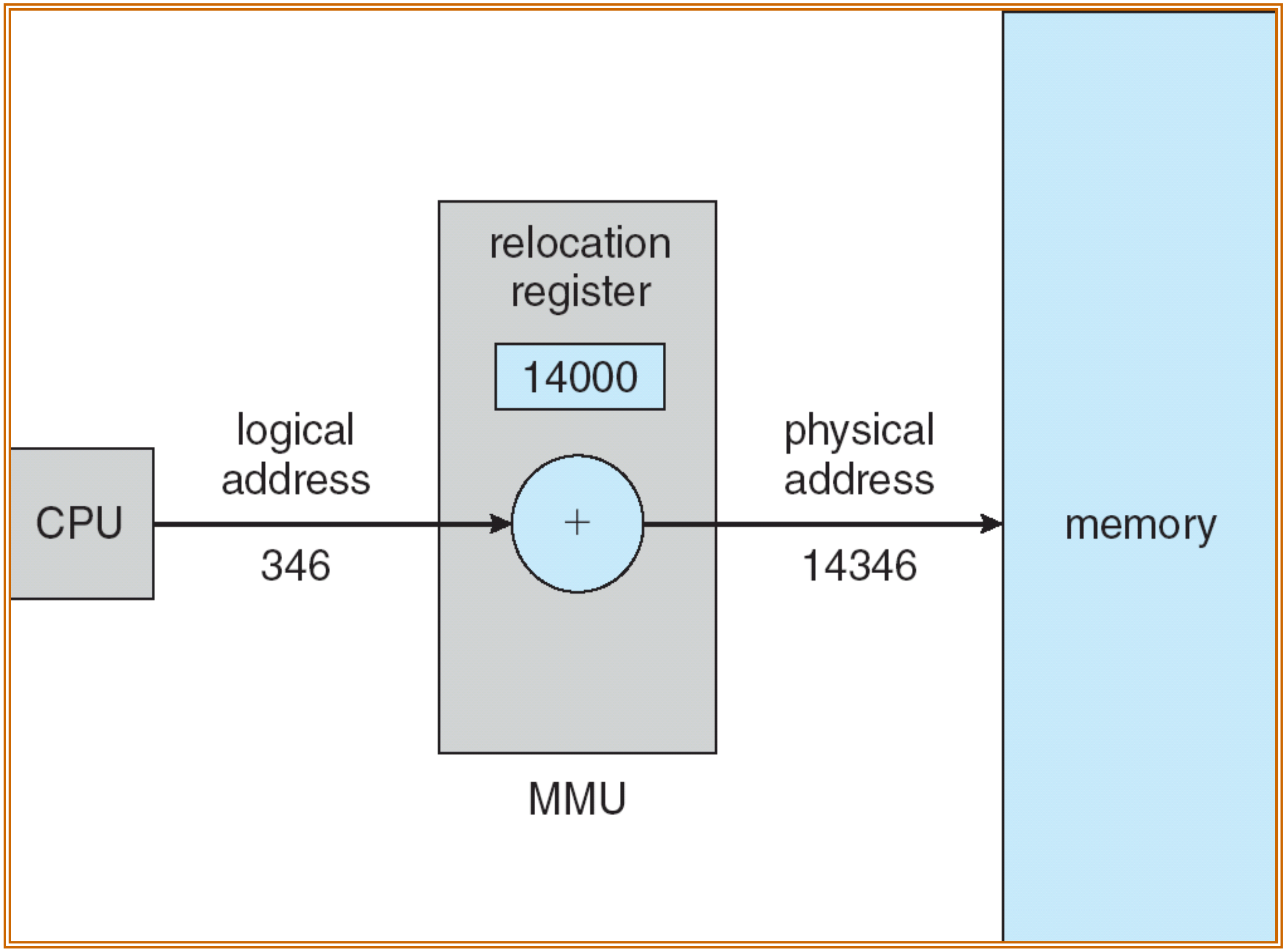
- Logical Address – Generated by the CPU, also referred to as virtual address
- Physical Address – Address seen by the memory unit

- Logical and Physical Addresses are the same in compile-time and load-time address-binding schemes.
- Logical (virtual) and Physical Addresses differ in execution-time address-binding scheme

Base & Limit Register

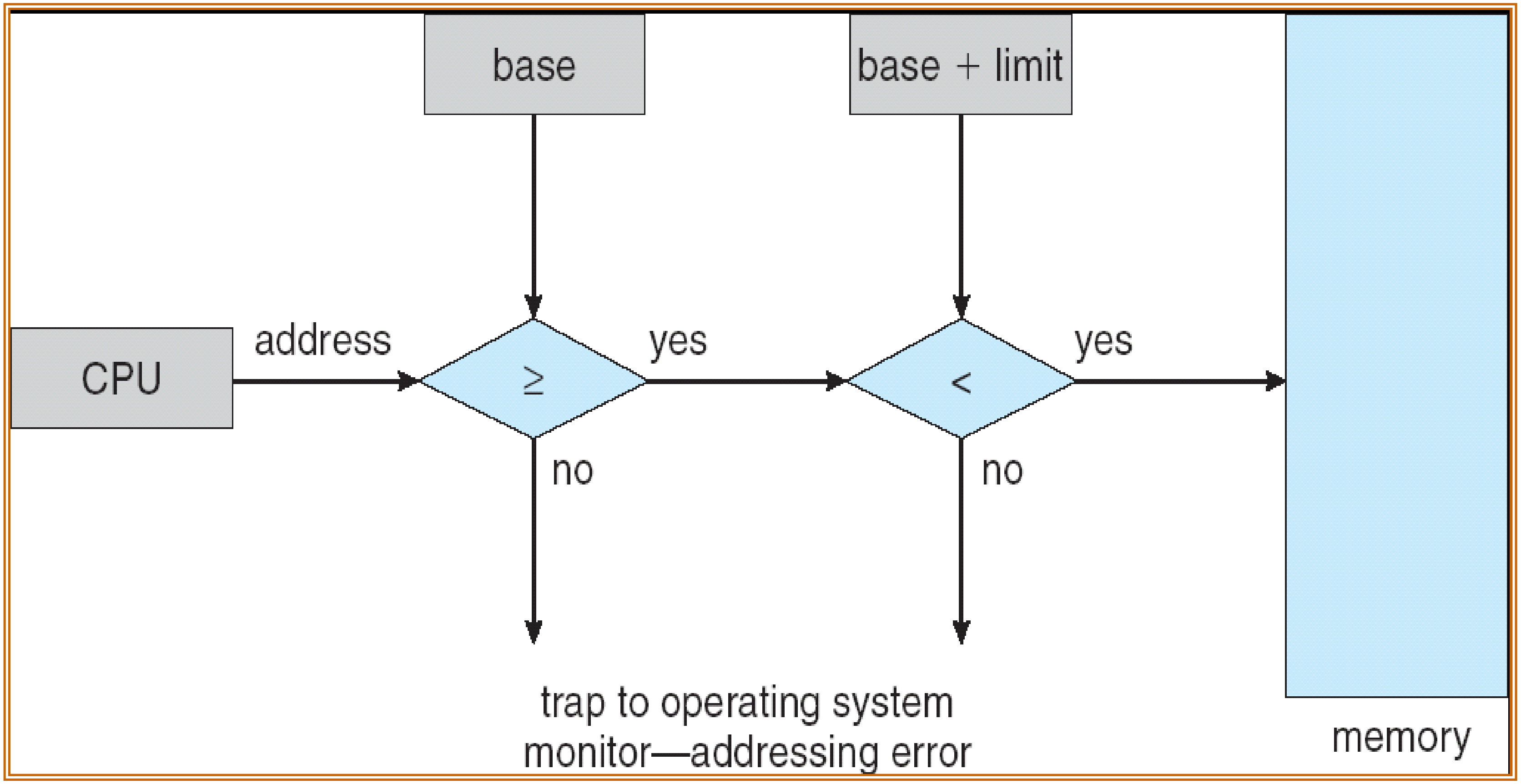


MMU



relocation register
14000
logical address
346
+
physical address
14346
CPU
memory
MMU

H/W Protection



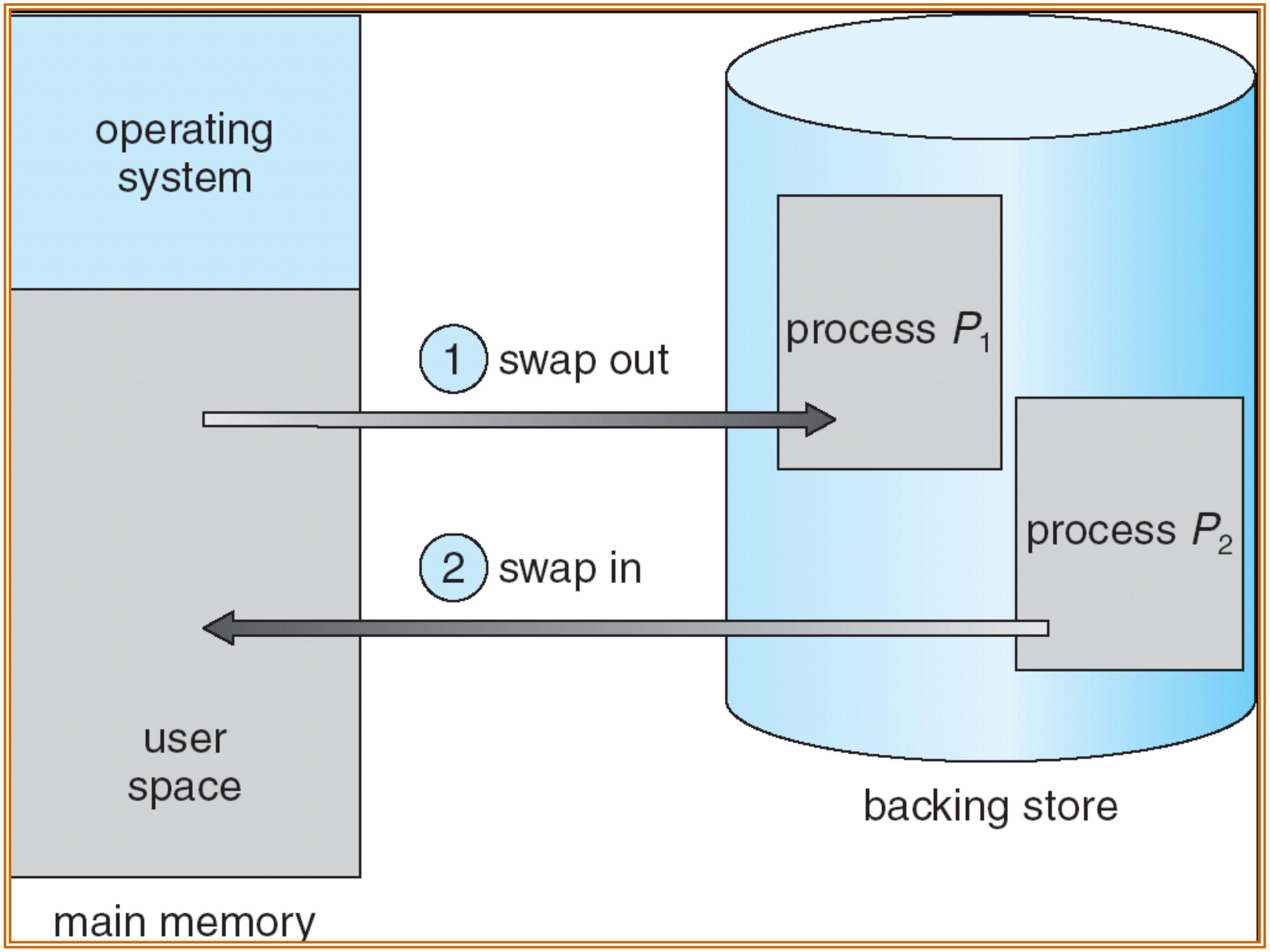
Dynamic Loading & Linking

- Routine is *not loaded until it is called*.
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- *Linking* postponed until Execution Time.
- *Dynamic Linking* is particularly useful for libraries

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- *Backing Store* – fast disk large enough to accommodate copies of all memory images for all users
- *Roll Out, Roll In* – Swapping variant used for priority-based scheduling algorithms.
- Major part of swap time is Transfer Time; Total Transfer Time is directly proportional to the amount of memory swapped.

Swapping (Conti…)



Contiguous Allocation

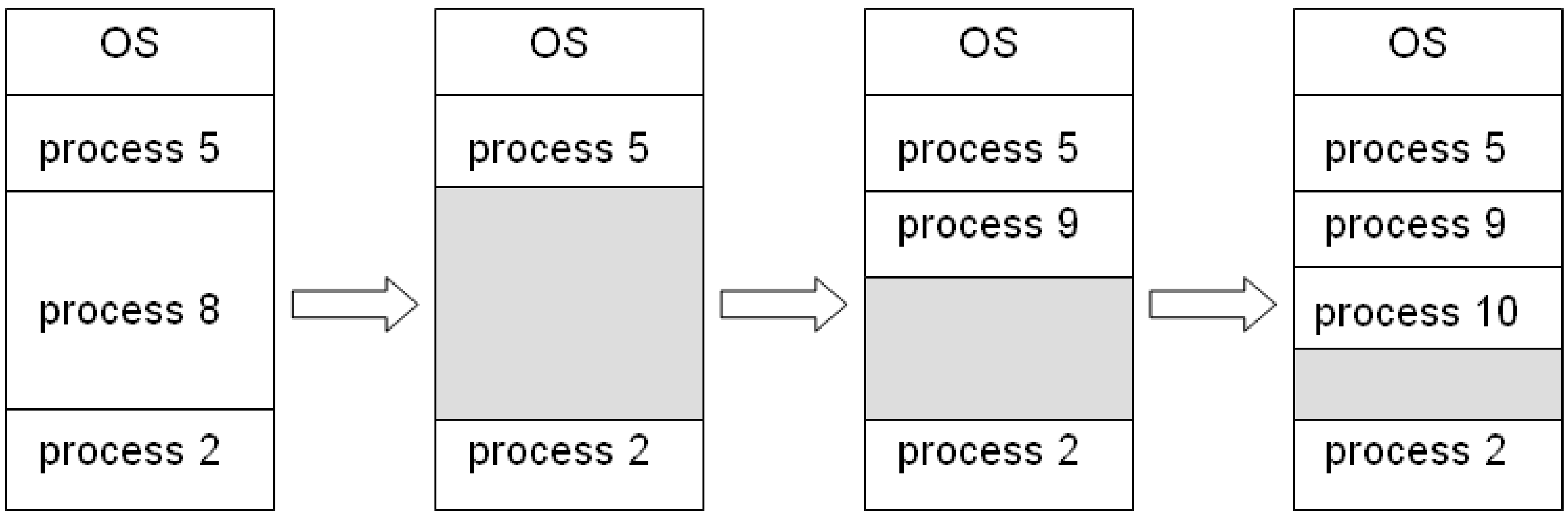
- Main memory is divided usually into two partitions:
  - Resident Operating System
  - User processes

Multiple-partition allocation

  - Hole – Block of available Memory
  - When a Process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:

    a) Allocated Partitions b) Free Partitions (hole)

Memory Allocation (Graphical Depiction)



Memory Allocation Strategies

- First-fit: Allocate the first hole that is big enough
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- Worst-fit: Allocate the largest hole; must also search entire list
  - Produces the largest leftover hole

Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous
- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if relocation is dynamic, and is done at execution time

Paging

- Logical address space of a process can be noncontiguous
- Divide *Physical Memory* into fixed-sized blocks called FRAMES
- Divide *Logical Memory* into blocks of same size called PAGES
- Keep track of all free frames
- To run a program of size n pages, need to find n free frames and load program
- Set up a page table to translate logical to physical addresses
- Internal Fragmentation

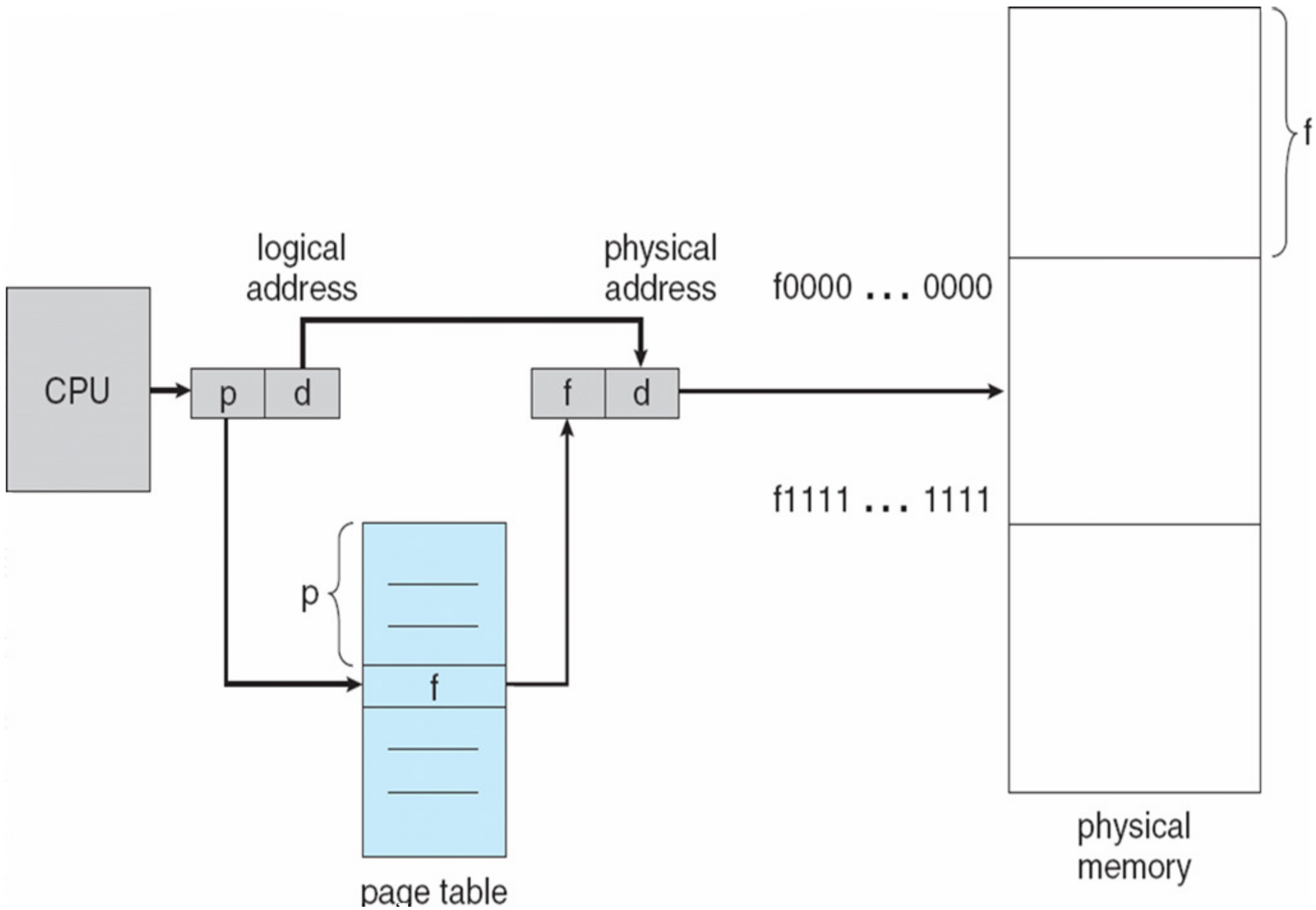
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

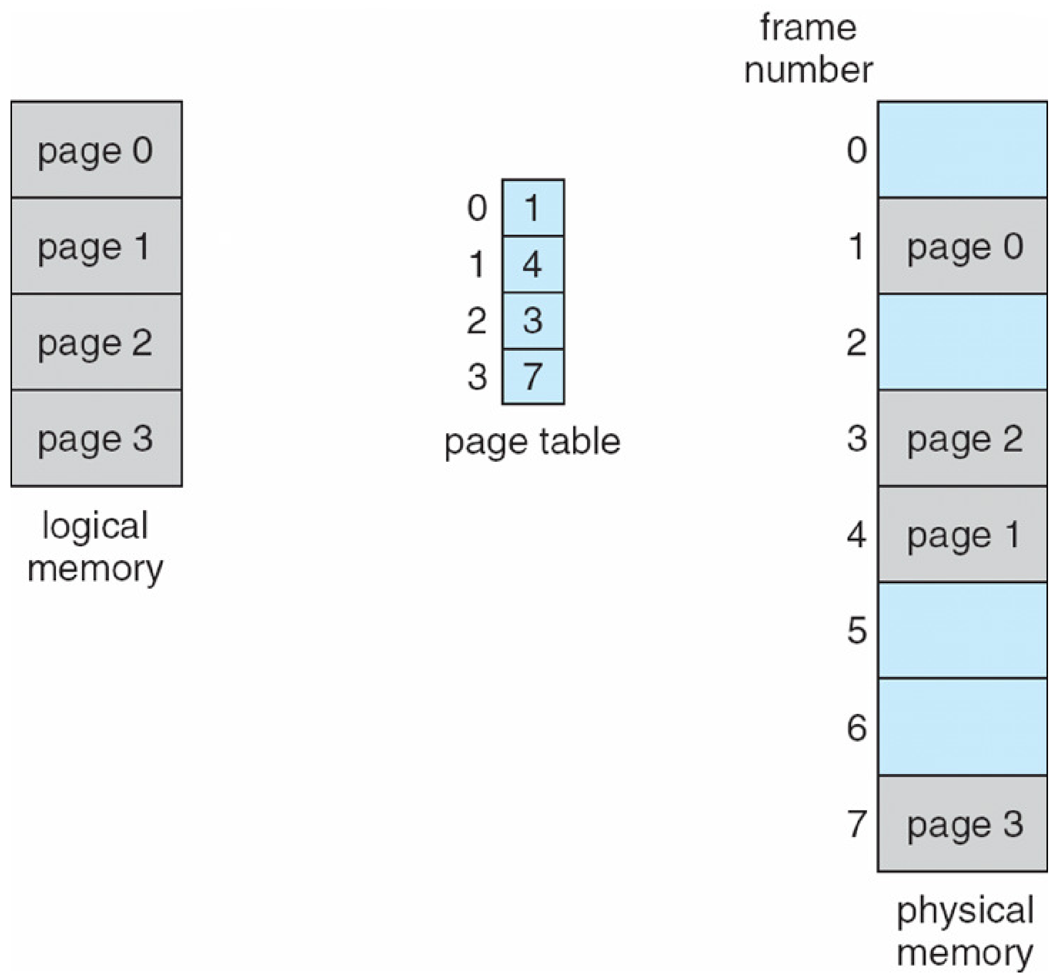
page number	page offset
p	d
$m - n$	n

  - For given logical address space 2^m *and page size* 2^n

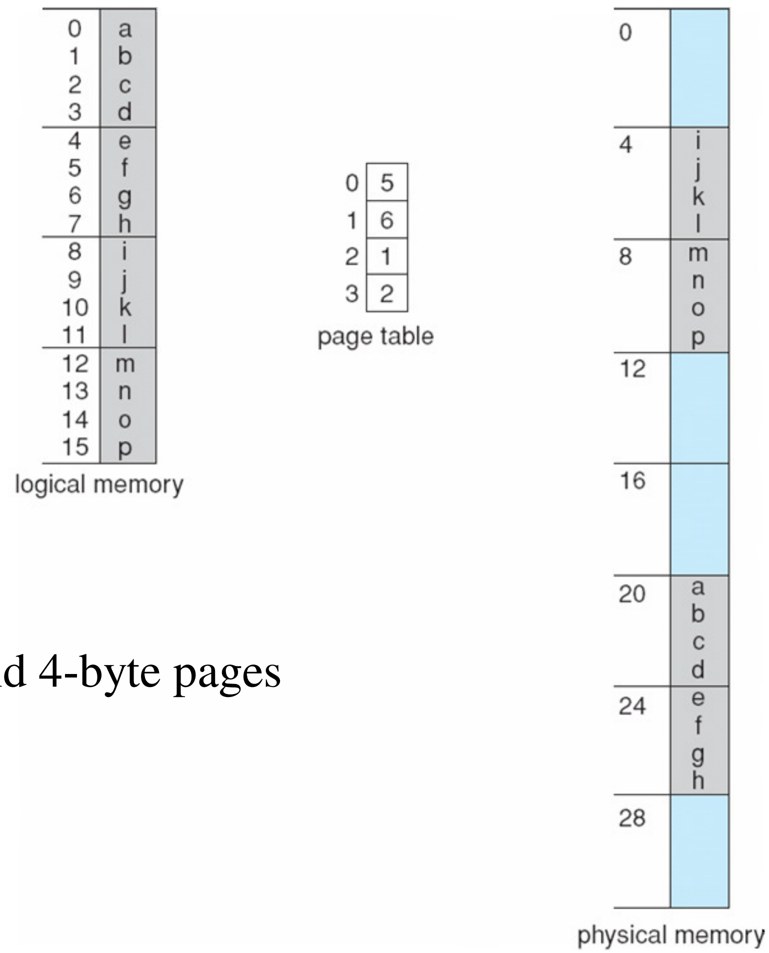
Paging Hardware



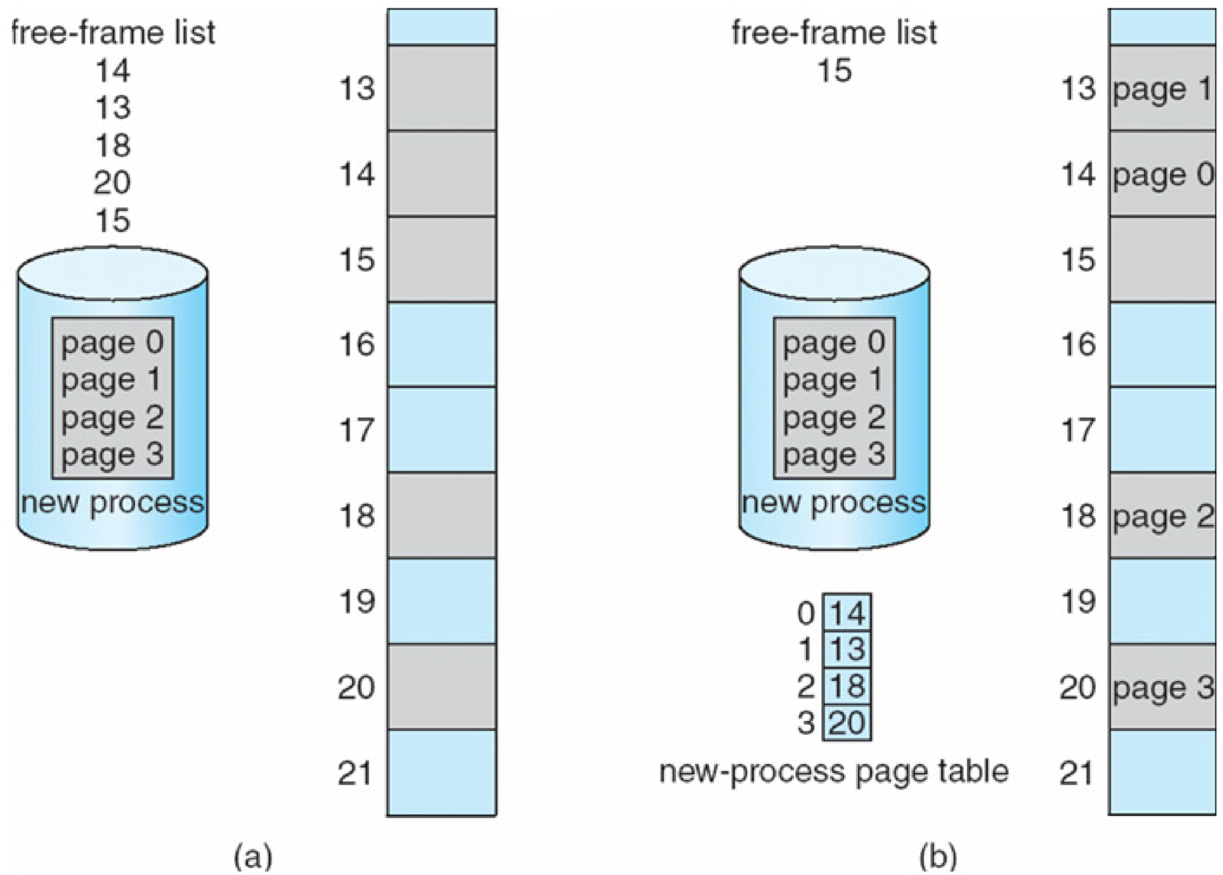
Page Modeling of Logical & Physical Memory



Paging Example



32-byte memory and 4-byte pages



Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory
- Page-table Base Register (PTBR) points to the Page Table
- Page-table Length Register (PRLR) indicates size of the Page Table
- In this scheme every data/instruction access requires two memory accesses. One for the *Page Table* and one for the *Data/instruction.*
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called *Associative Memory* or *Translation Look-aside Buffers* (TLBs)

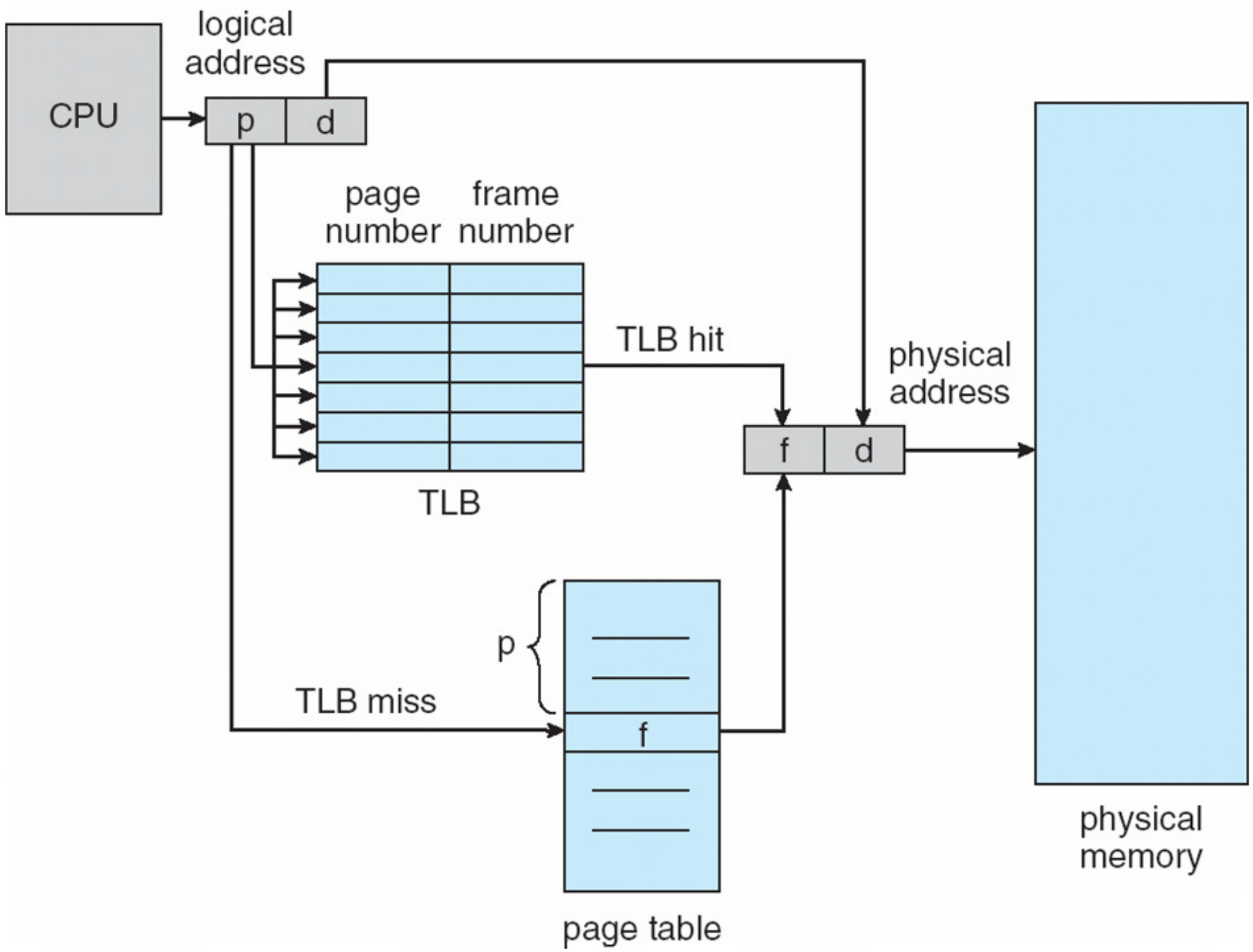
Associative Memory

Page #	Frame #

Address translation (p, d)

- If p is in associative register, get frame # out
- Otherwise get frame # from page table in memory

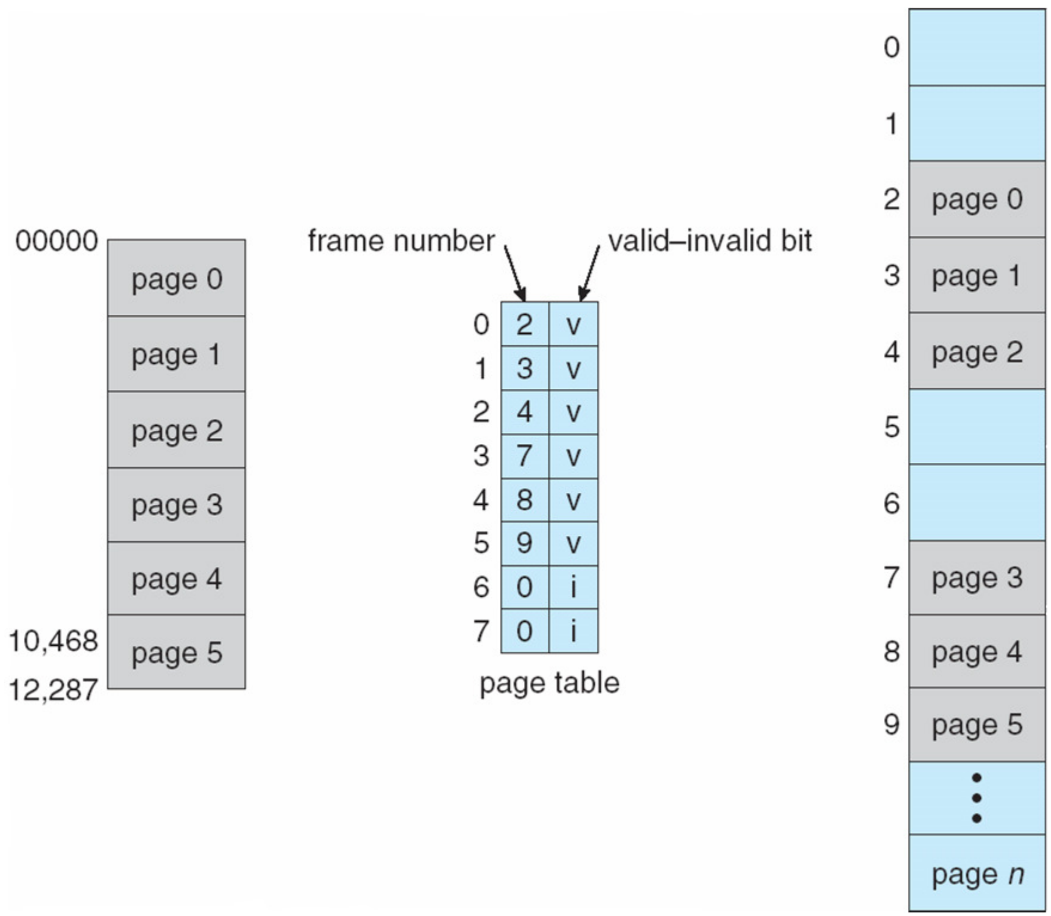
Paging Hardware with TLB



Valid Invalid Bit for Protection

- Memory protection implemented by associating protection bit with each frame
- Valid-invalid bit attached to each entry in the page table:
  - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page
  - “invalid” indicates that the page is not in the process’ logical address space

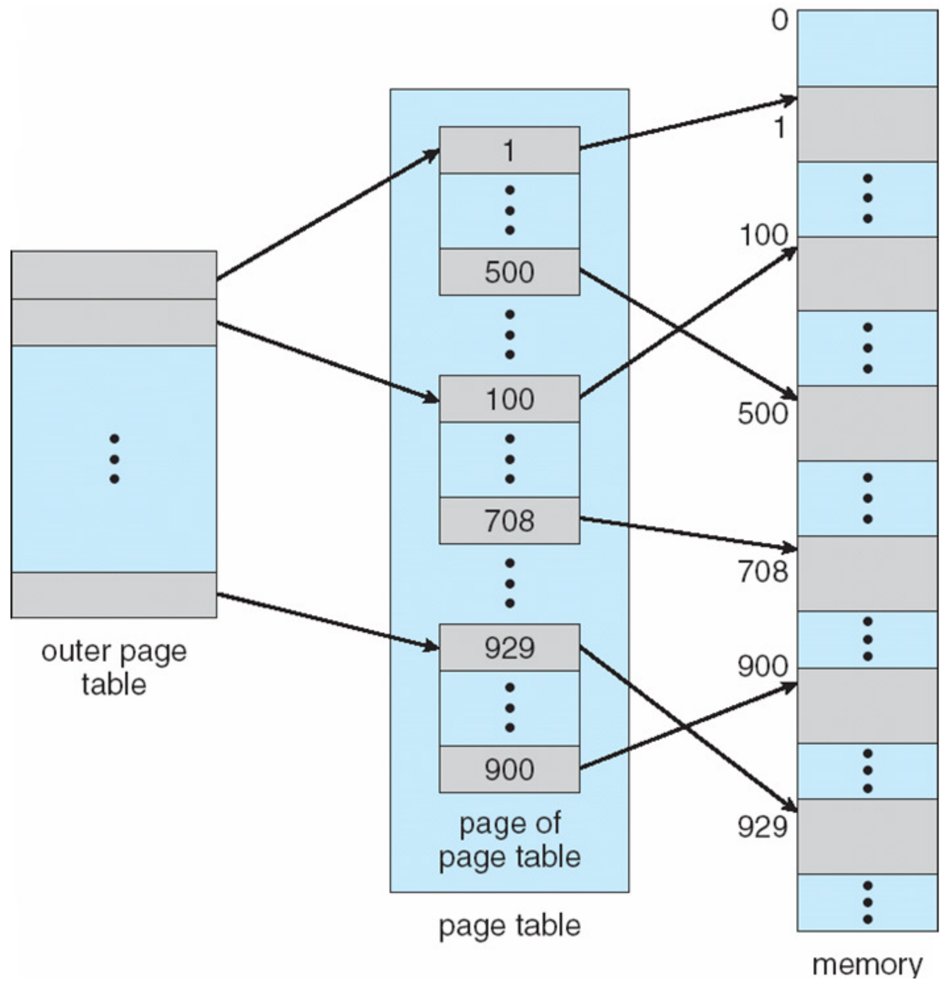
Valid Invalid Bit in Page Table



Structure of PAGE TABLE

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Two Level Page Table Scheme



Two Level Paging Example

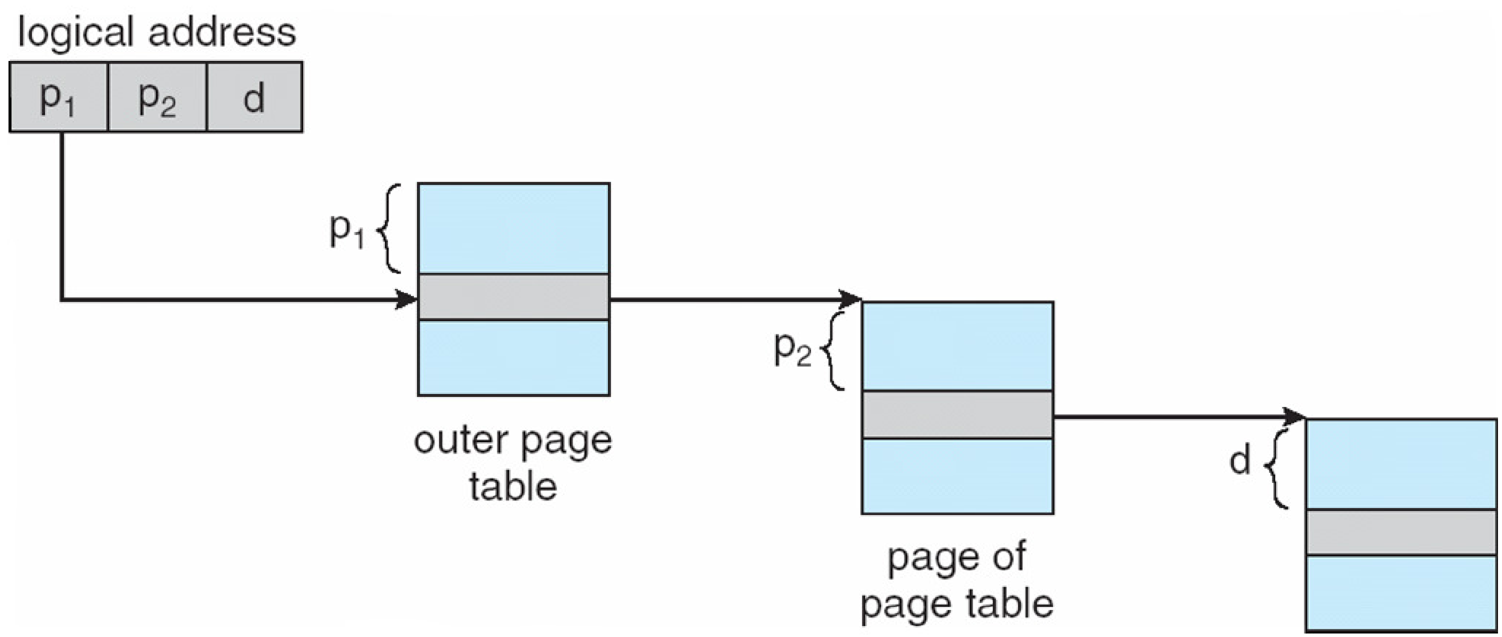
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - A Page number consisting of 22 bits
  - A Page offset consisting of 10 bits
- Since the PAGE TABLE IS PAGED, the page number is further divided into:
  - A 12-bit Page number
  - A 10-bit Page offset
- Thus, a Logical Address is as follows:

Two Level Paging Example (contd.)

page number		page offset
p_i	p_2	d
12	10	10

where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Address Translation Scheme

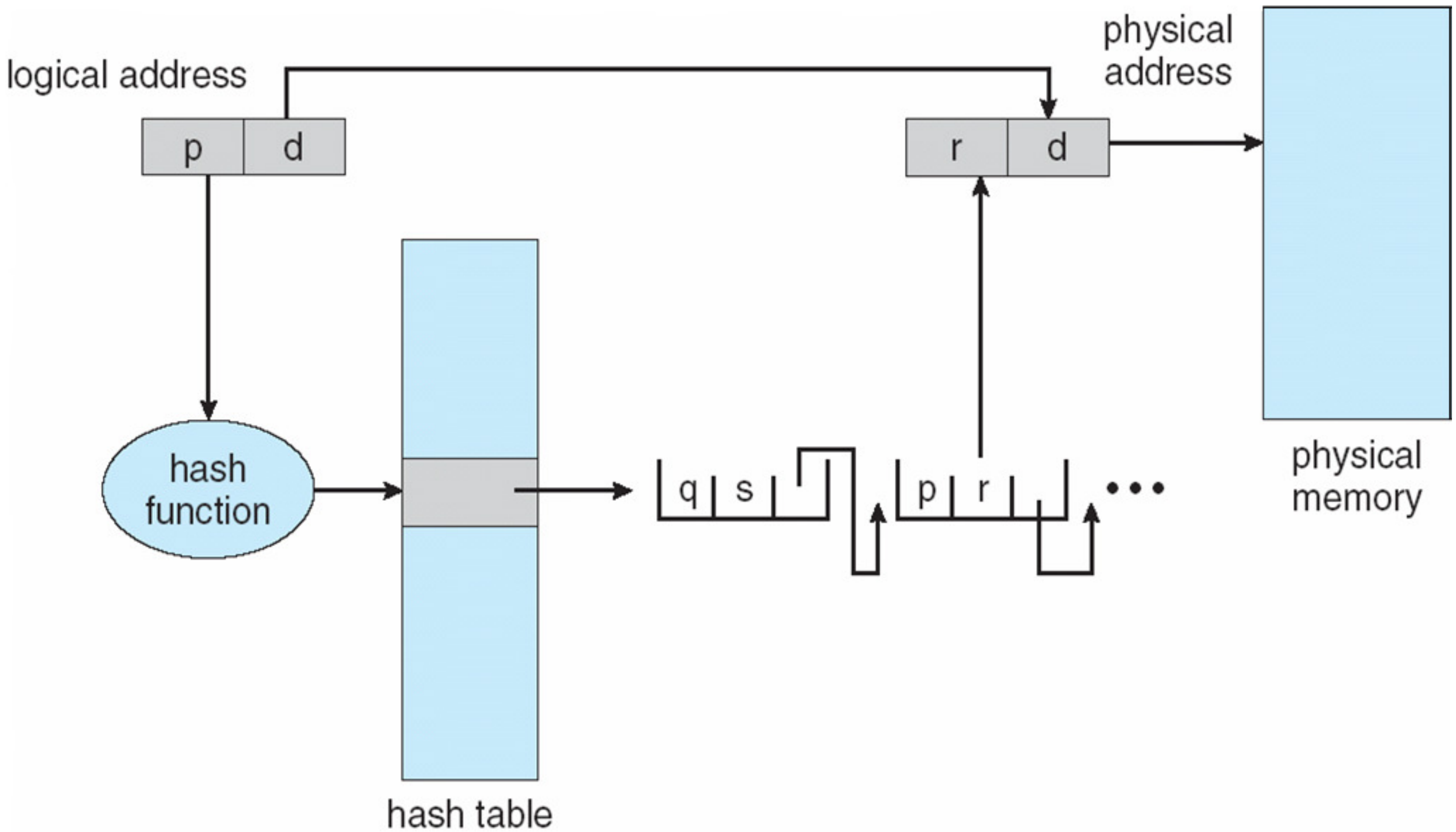


Three Level Paging Scheme

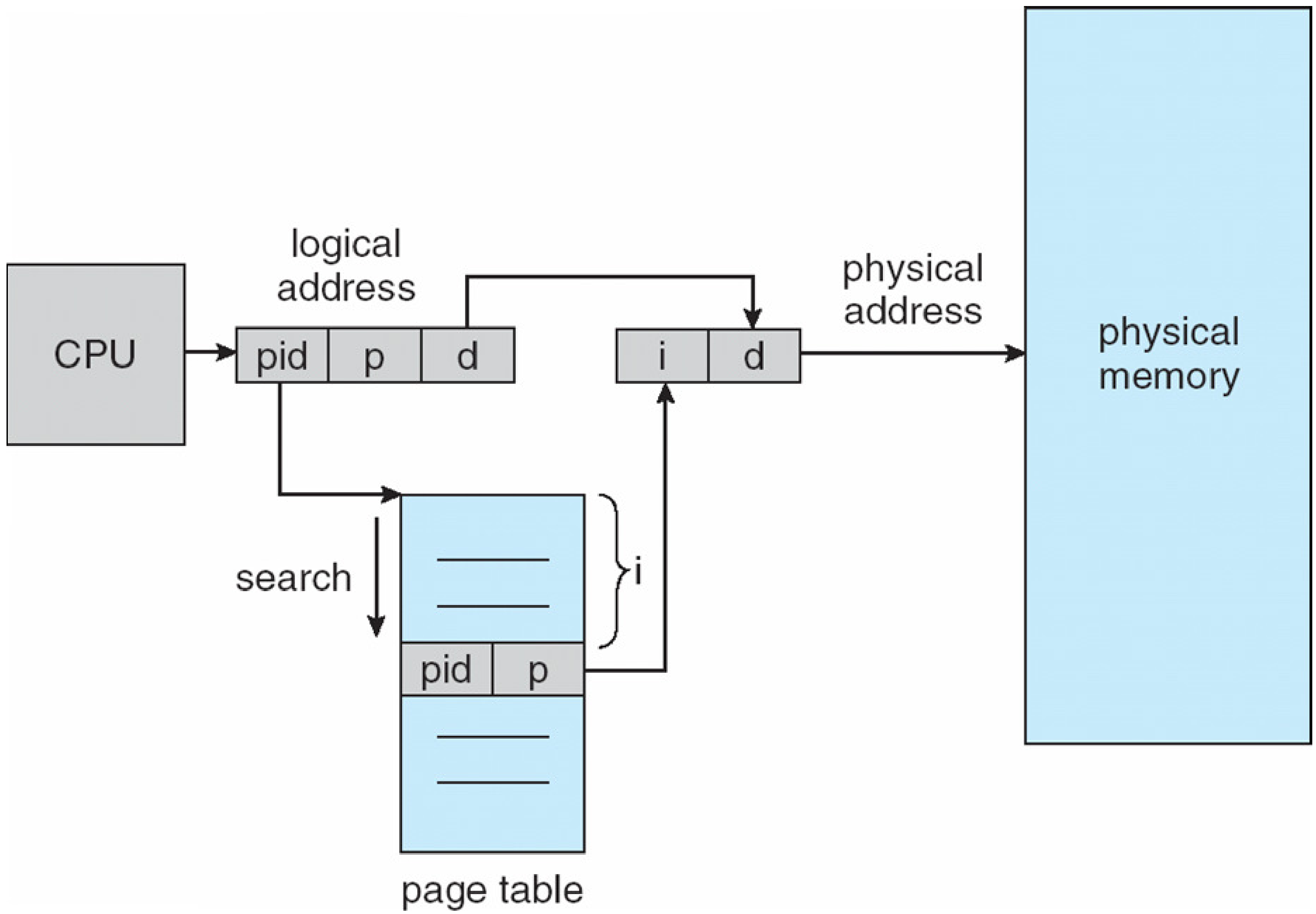
outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

Hashed Page Table



Inverted Page Table

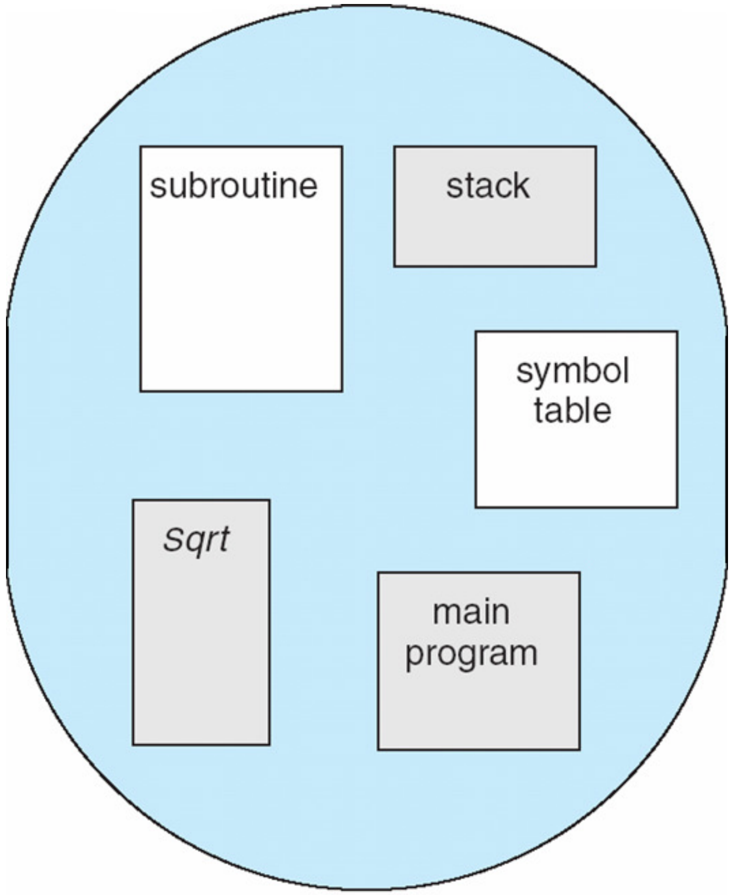


Segmentation

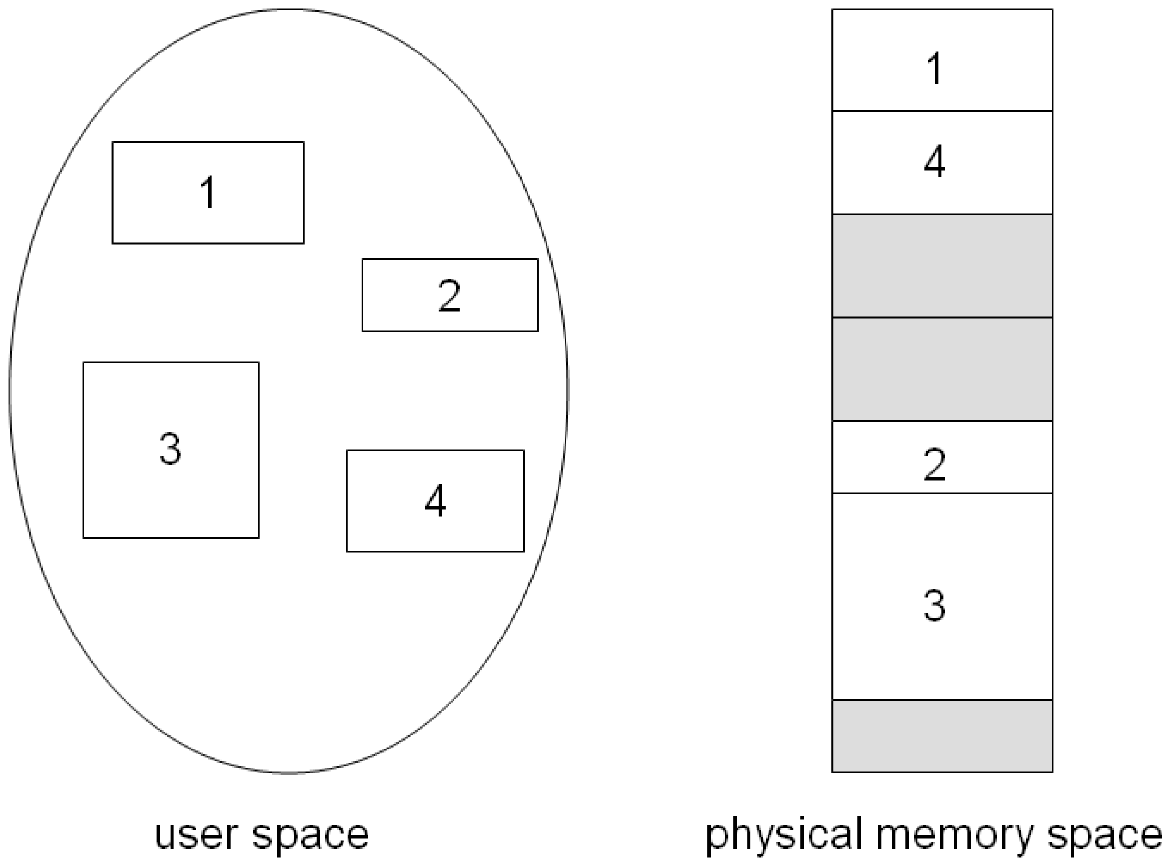
- *A Program is a collection of Segments*
  - A *SEGMENT* is a logical unit such as:

    Main Program
    Procedure
    Function
    Method
    Object
    Local Variables, Global Variables
    Common Block
    Stack
    Symbol Table
    Arrays

User's View of Program



Logical View of Segmentation



Segmentation Architecture

- Logical Address consists of a two tuple:

  $<$segment-number, offset$>$

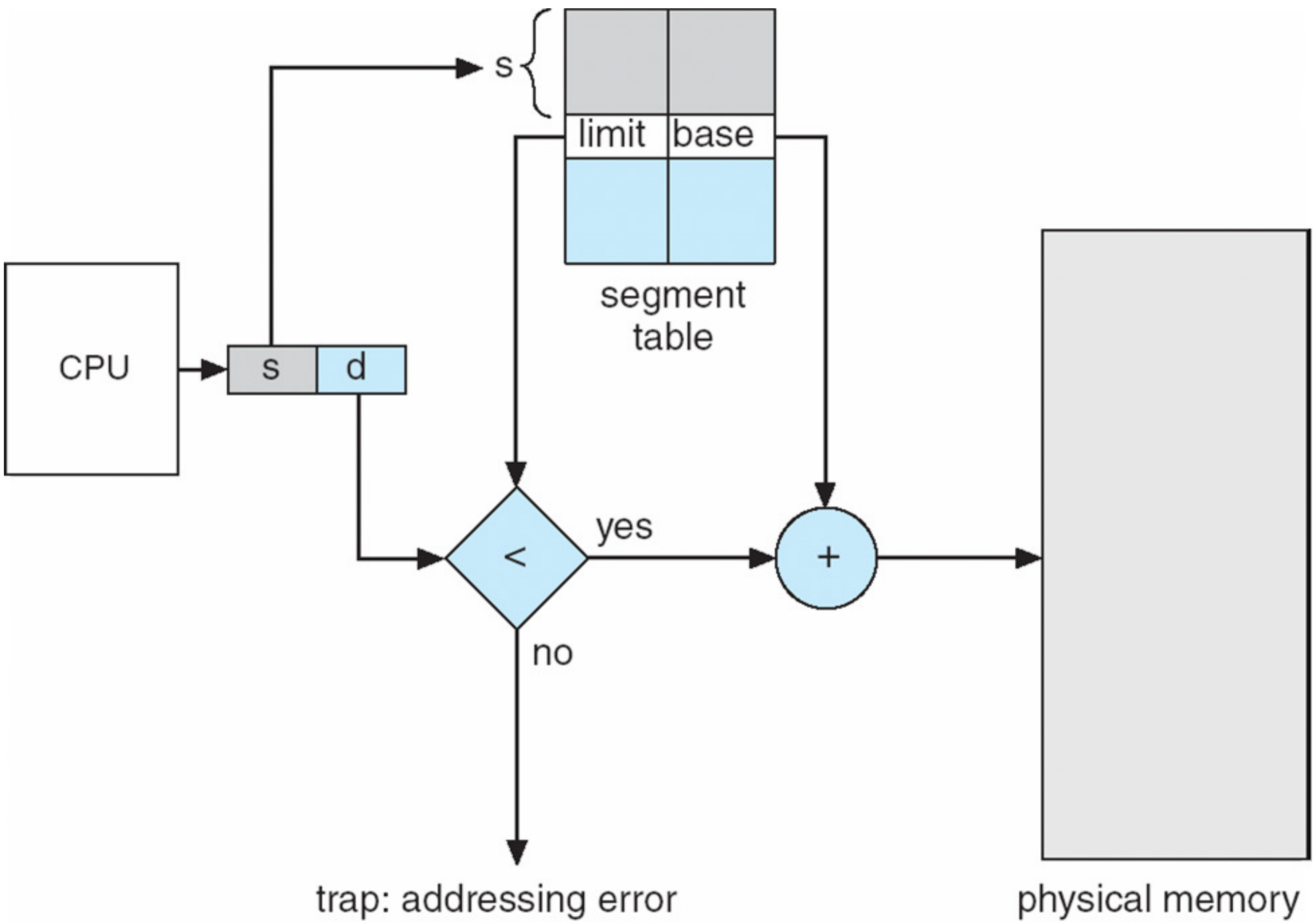
- Segment Table – Maps two-dimensional physical addresses.

- Each Table entry has:

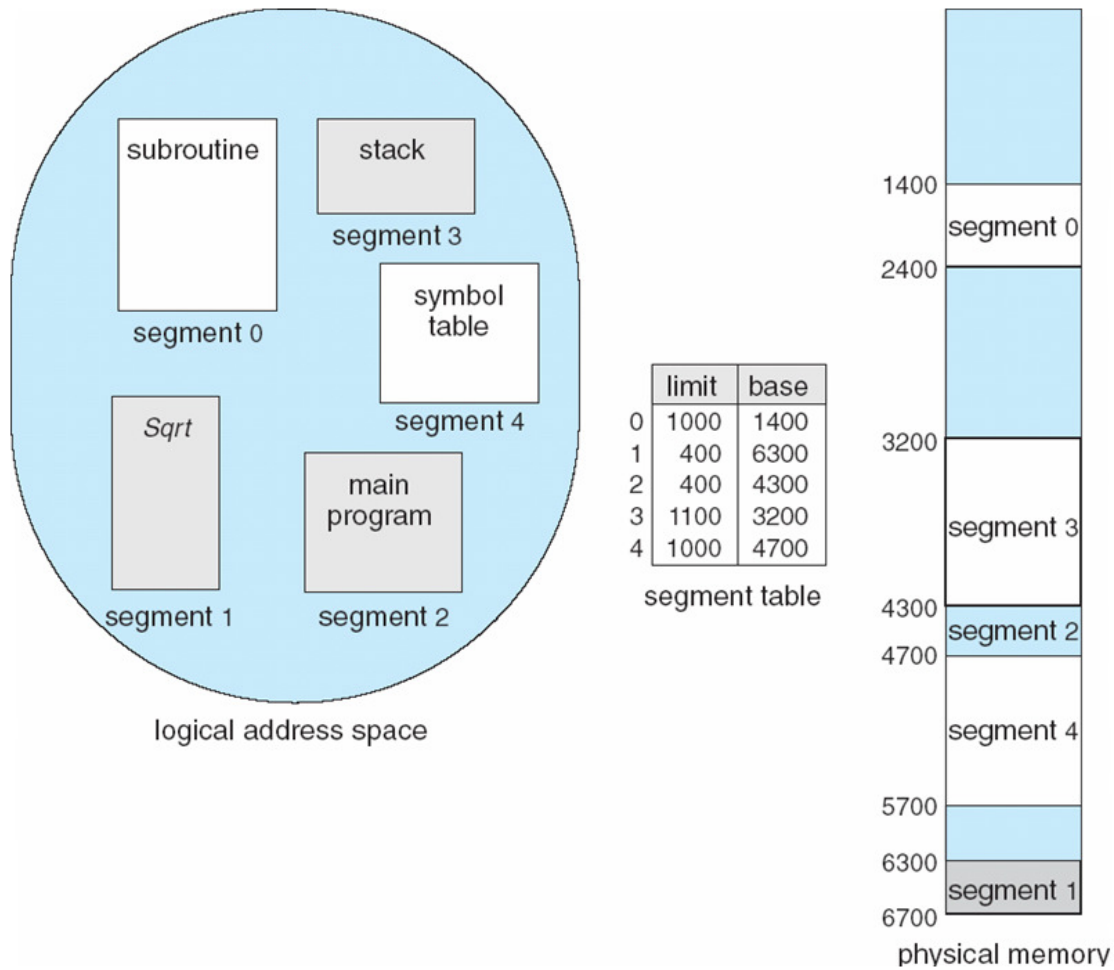
  - base – Contains the starting Physical Address where the segments reside in memory

  - limit – Specifies the Length of the Segment

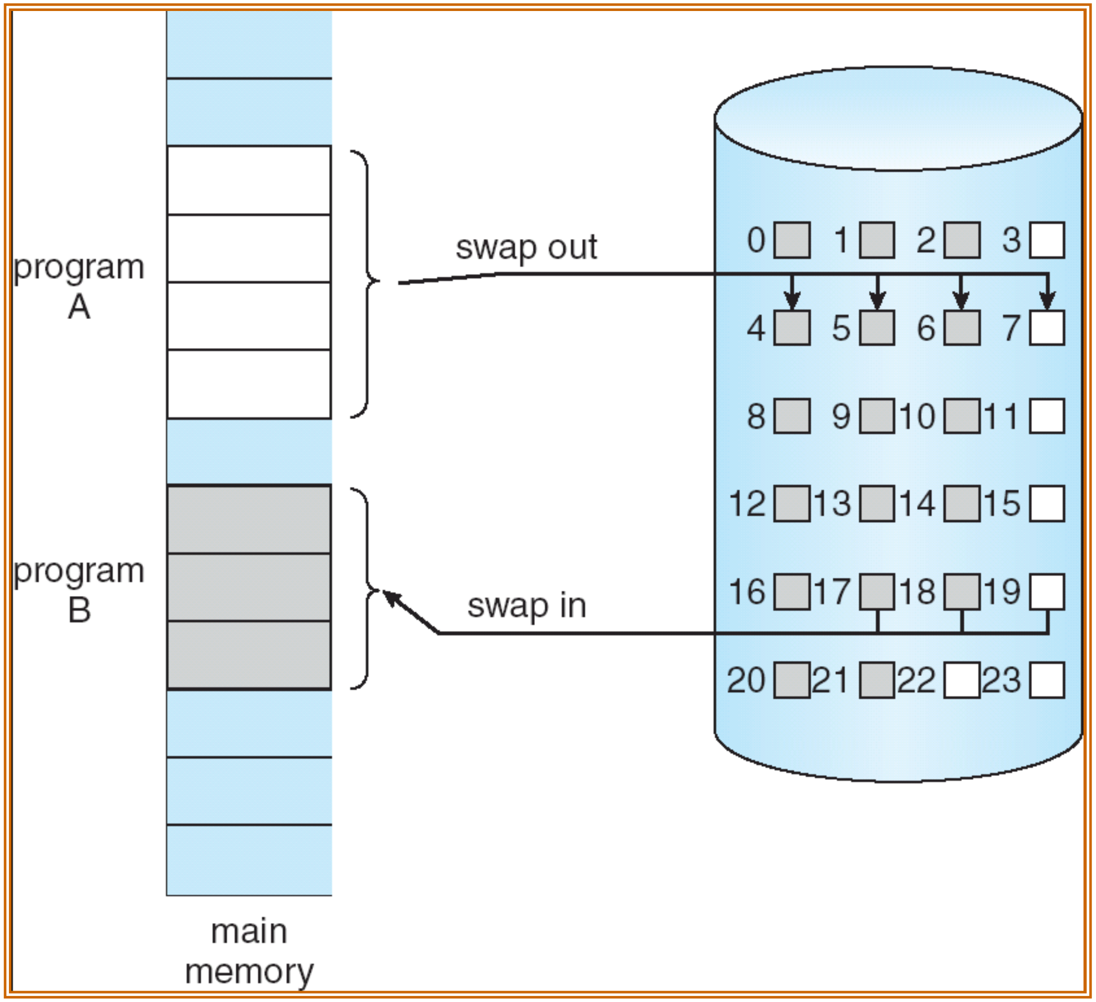
Segmentation Hardware



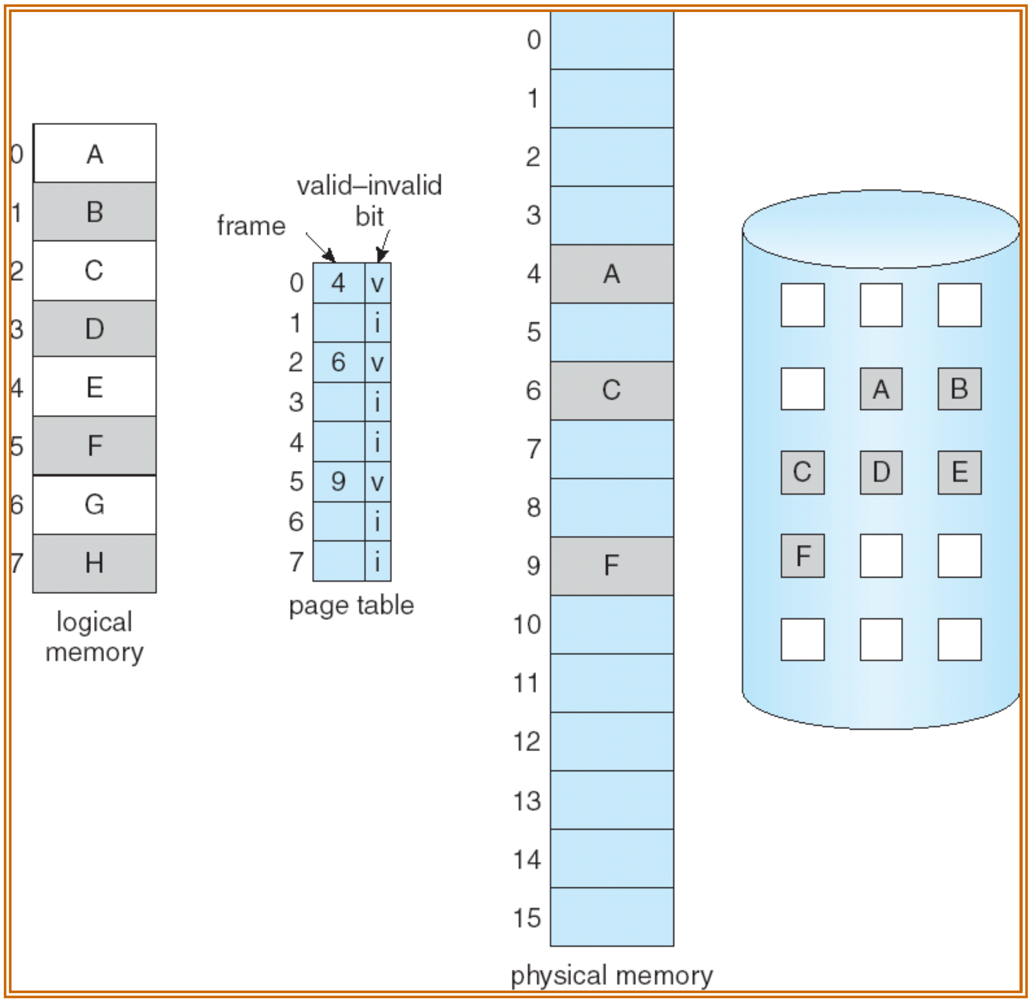
Example of Segmentation



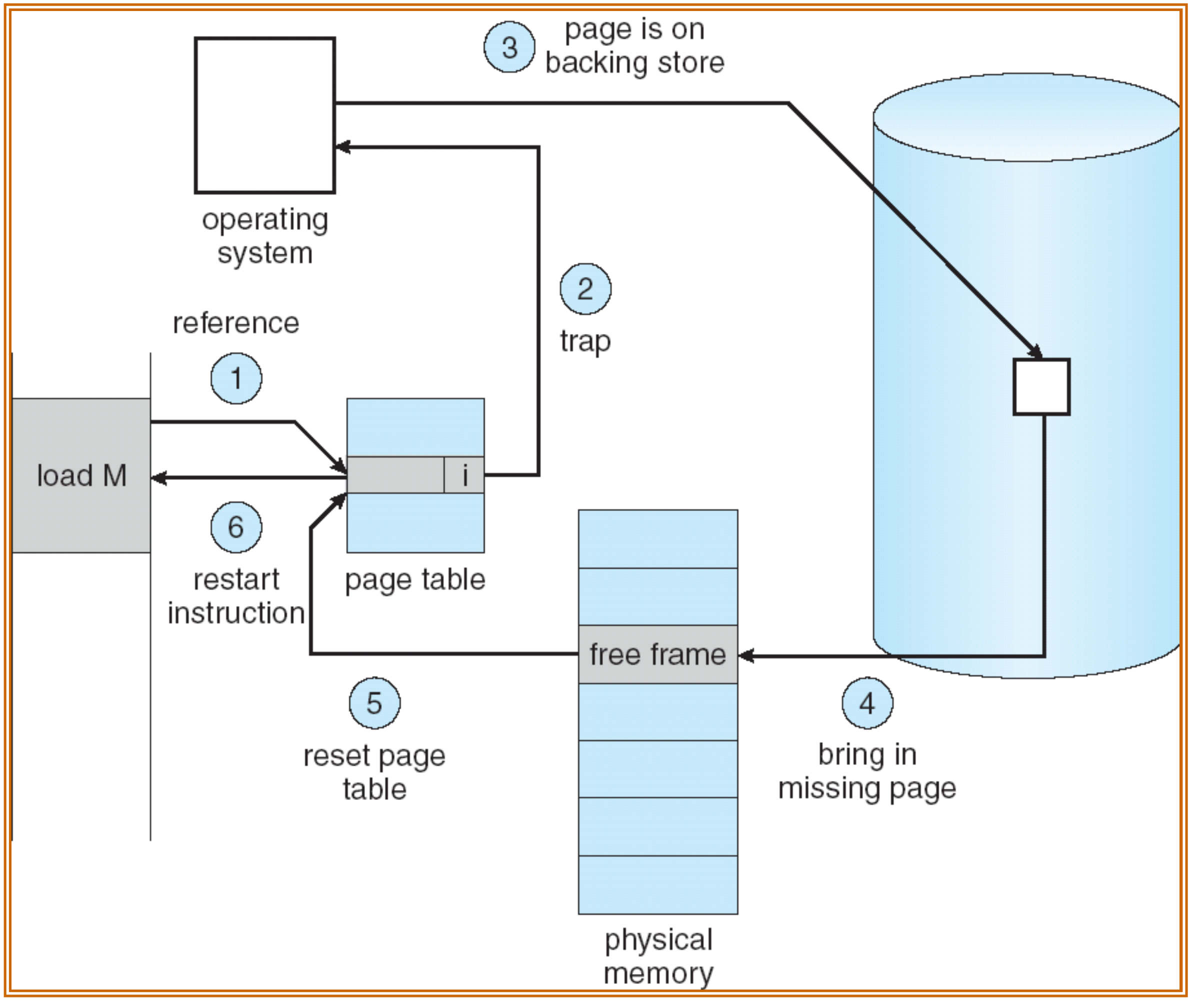
Transfer of a Paged Memory to Contiguous Disk Space



Page Table when some pages are not in Memory



Steps in Handling 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

Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- *Effective Access Time* (*EAT*)

$$EAT = (1 - p) \text{ x memory access}$$
$$+ p \text{ (page fault overhead}$$
$$+ [\text{swap page out }]$$
$$+ \text{swap page in}$$
$$+ \text{restart overhead)}$$

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement

- Use <u>*Modify (Dirty) Bit*</u> to reduce overhead of page transfers – only modified pages are written to disk

Basic Page Replacement

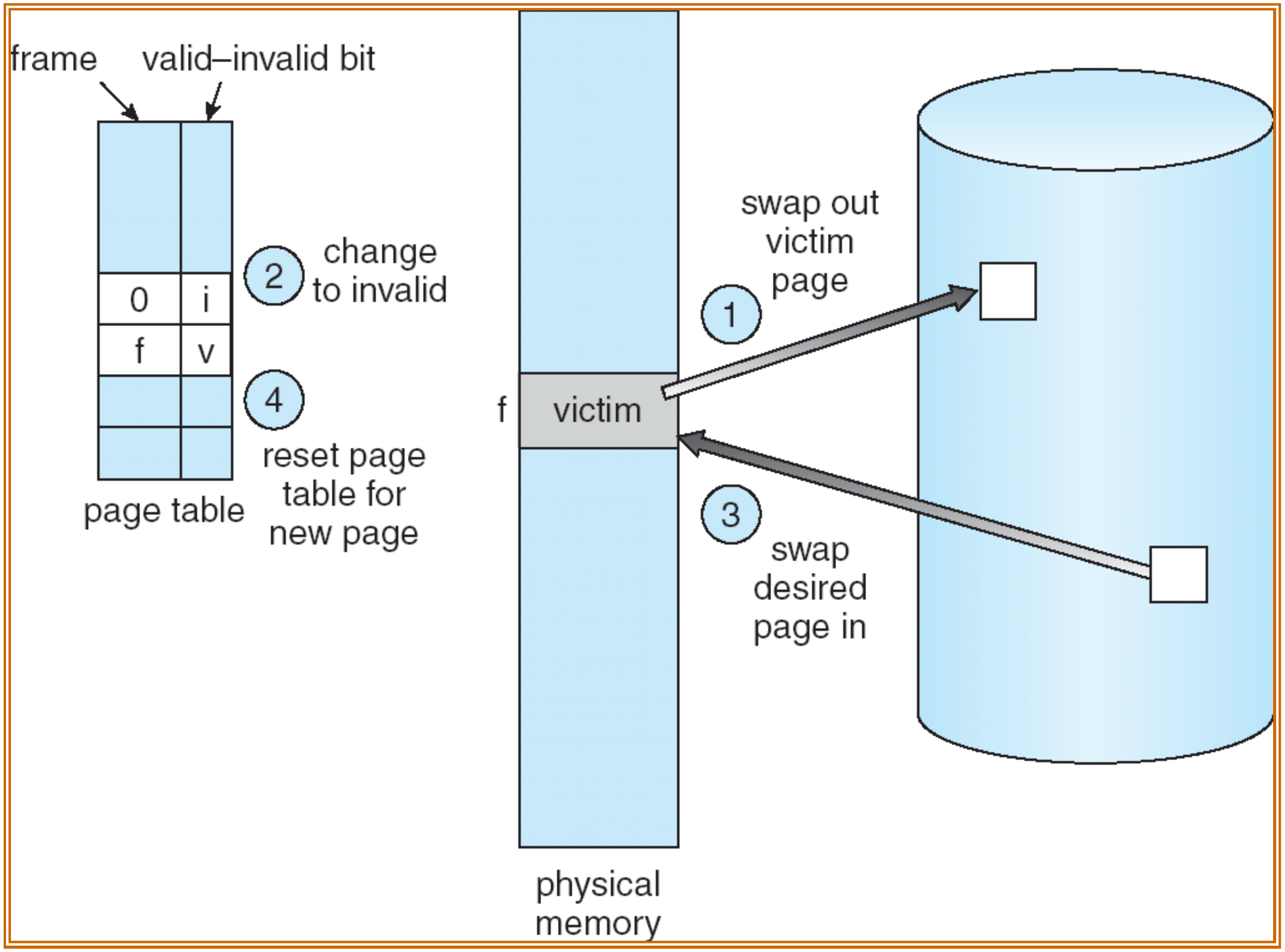
1. Find the location of the desired page on disk

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

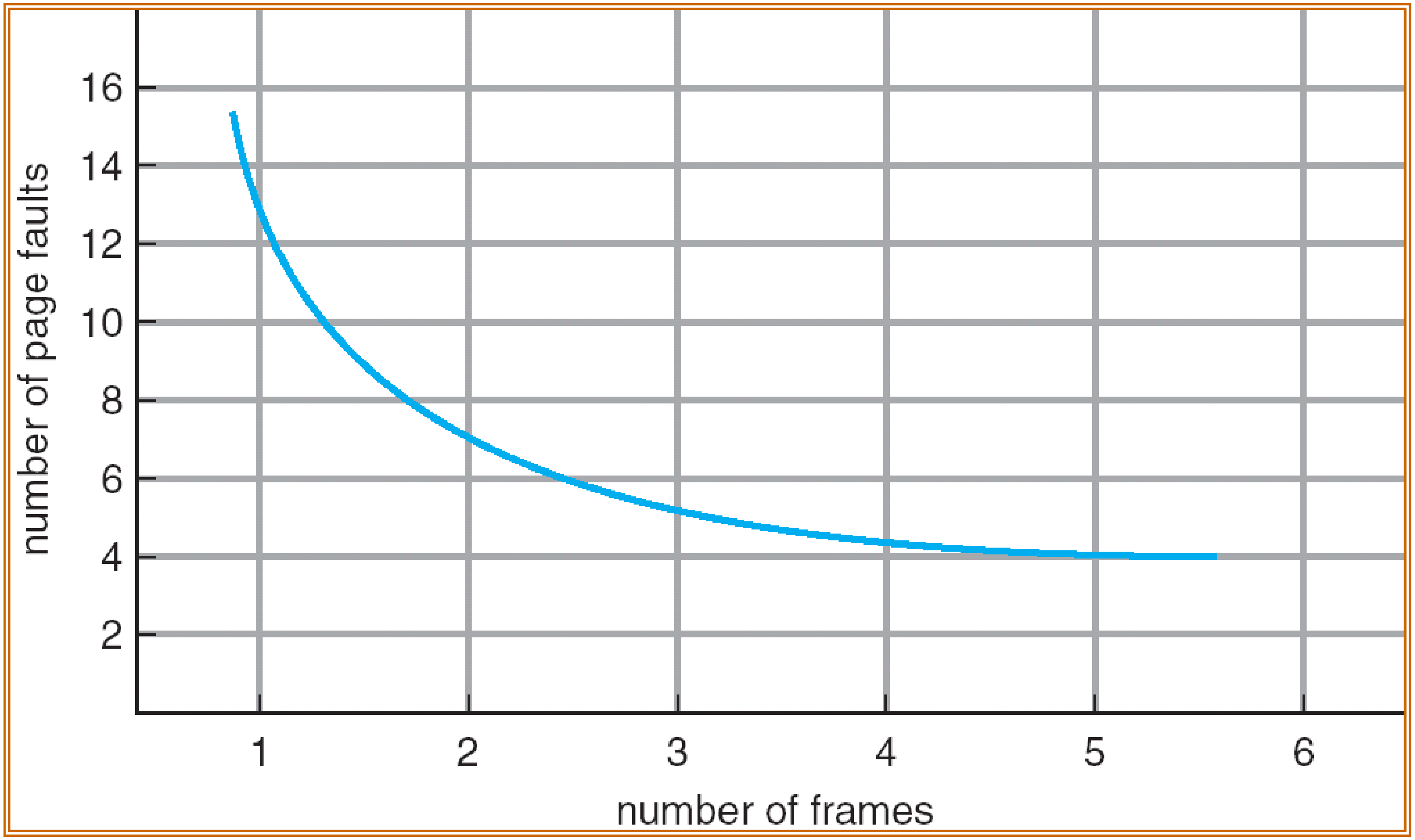
3. Read the desired page into the (newly) free frame. Update the page and frame tables.

4. Restart the process

Page Replacement



Page Fault versus No. of Frames



FIFO – First In First Out

- 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)

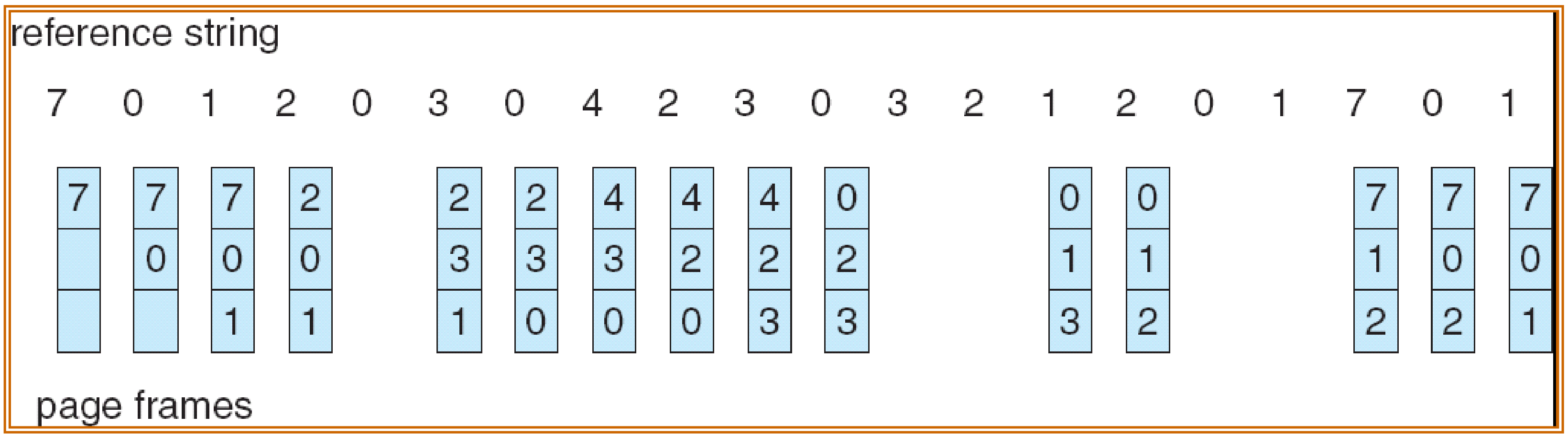
1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement – Belady's Anomaly
  - more frames $\Rightarrow$ more page faults

FIFO Page Replacement



Optimal Algorithm

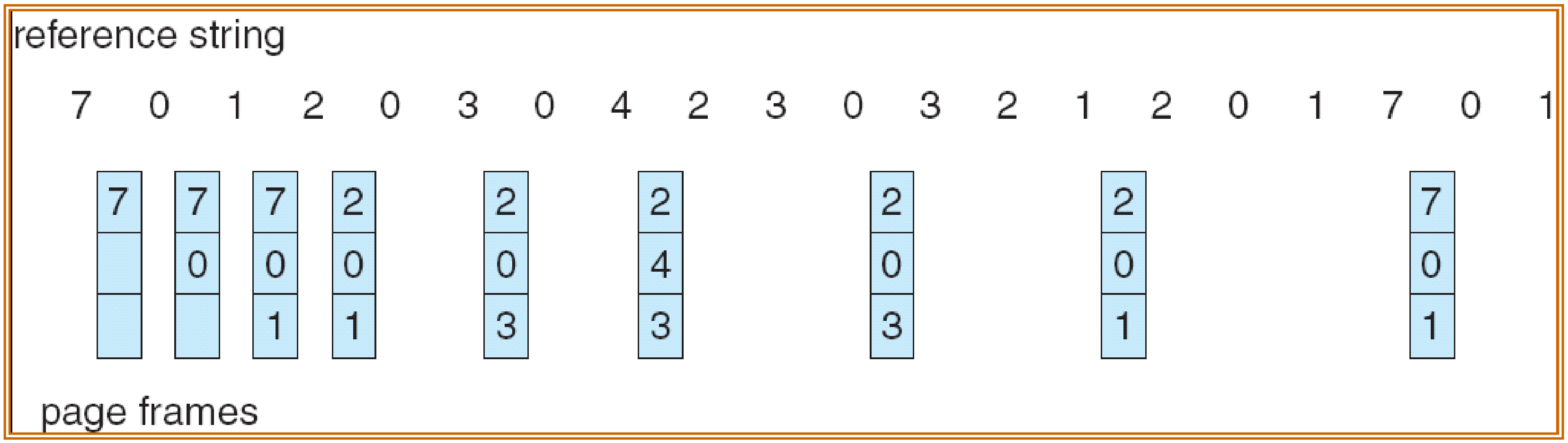
- Replace page that will not be used for longest period of time
- 4 frames example

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

1	4
2	
3	
4	5

6 page faults

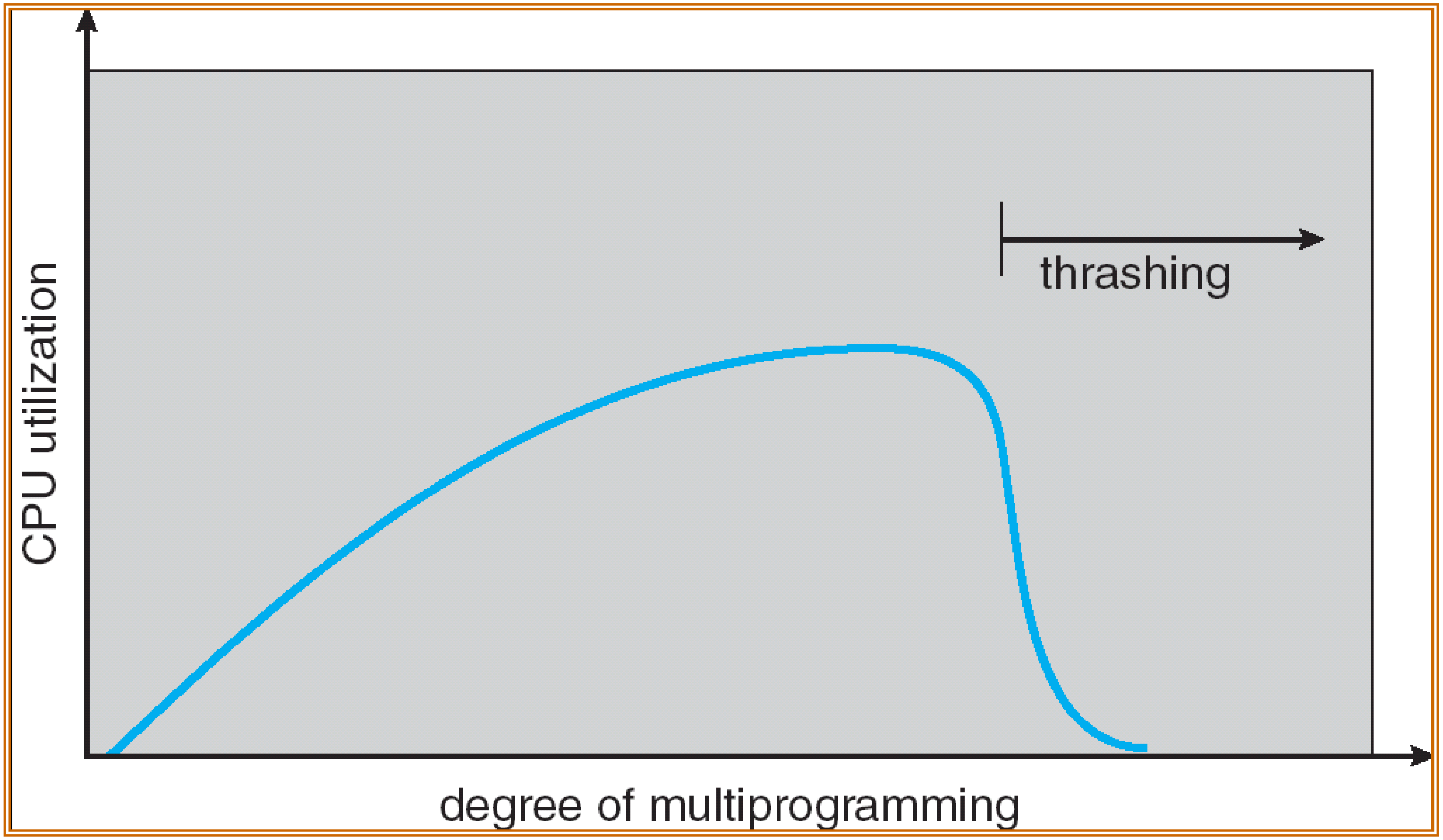
Optimal Page Replacement



Thrashing

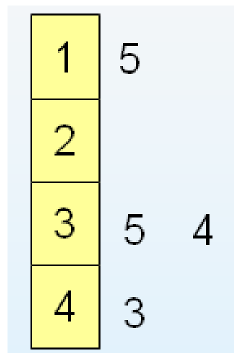
- If a Process does not have "enough" pages, the Page-fault rate is very high. This leads to:
  - Low CPU Utilization
  - Operating System thinks that it needs to increase the degree of multiprogramming
  - Another Process added to the system
- Thrashing ≡ a process is busy swapping pages in and out

Thrashing (contd.)



Least Recent Used Algorithm

- Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

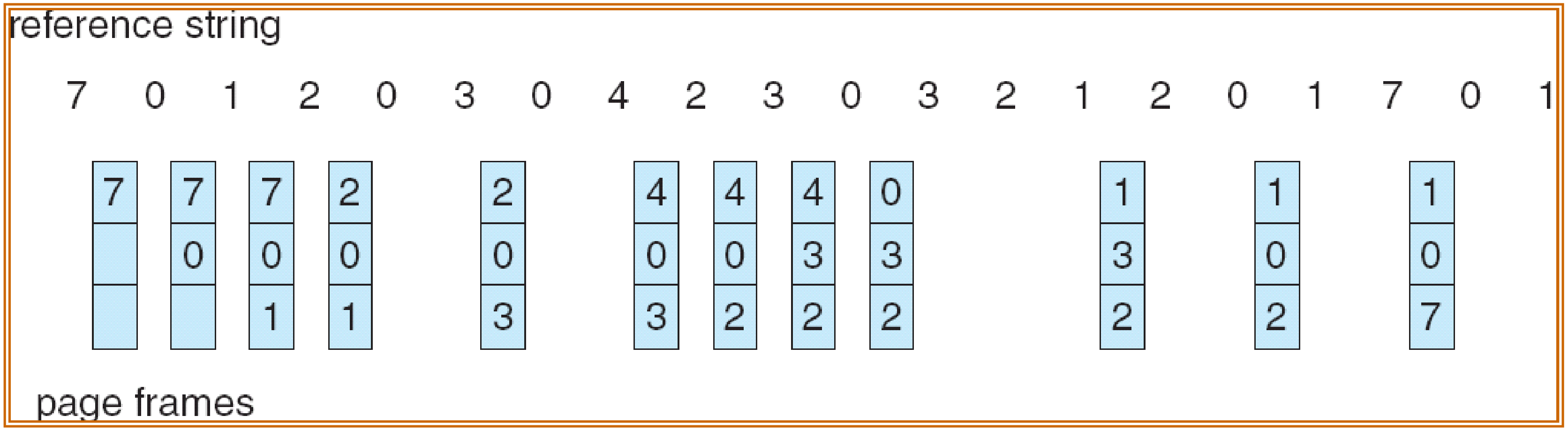


__Counter Implementation__

-- 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 determine which are to change

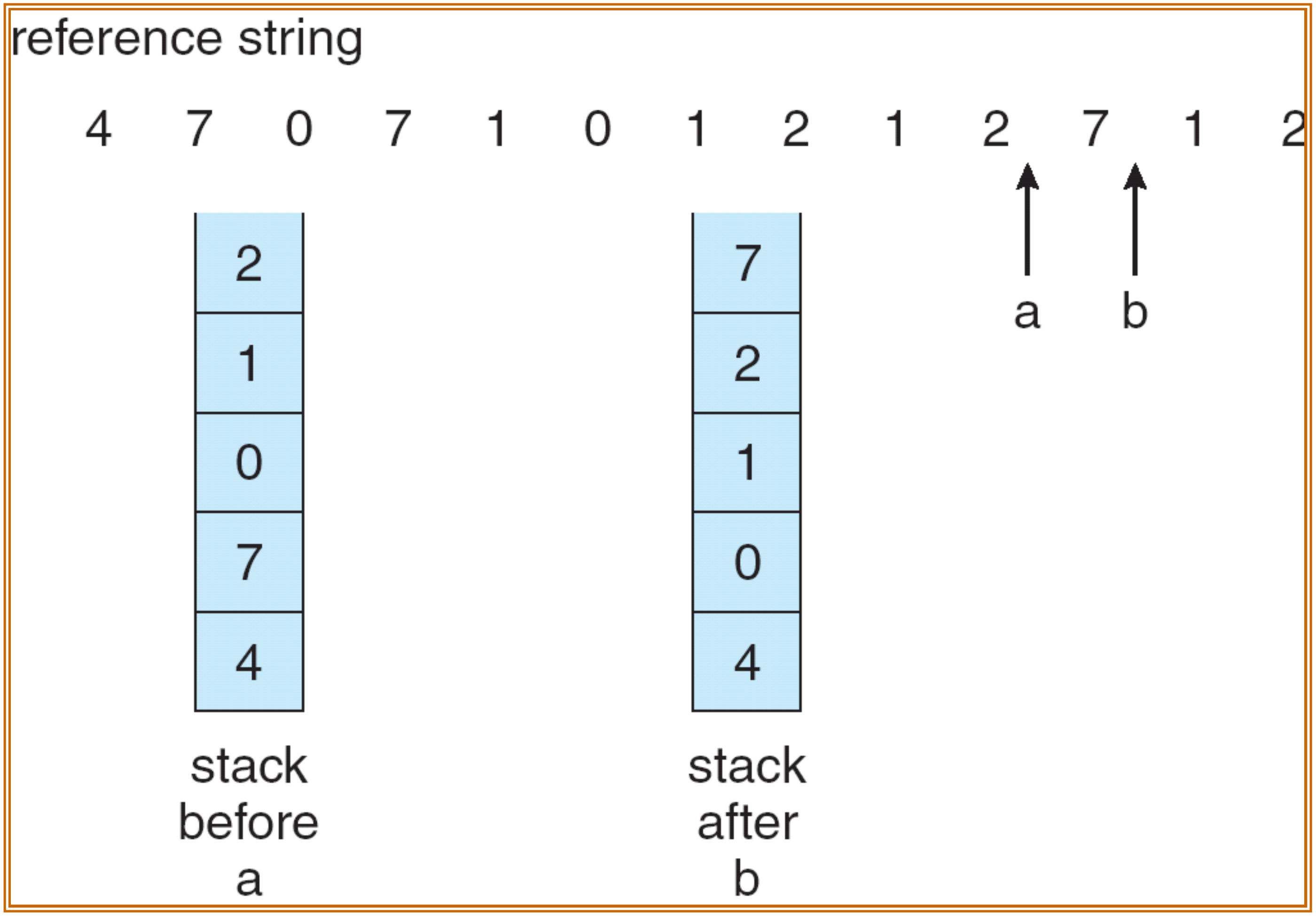
LRU Page Replacement



LRU Algorithm (contd.)

- ***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
  - No search for replacement

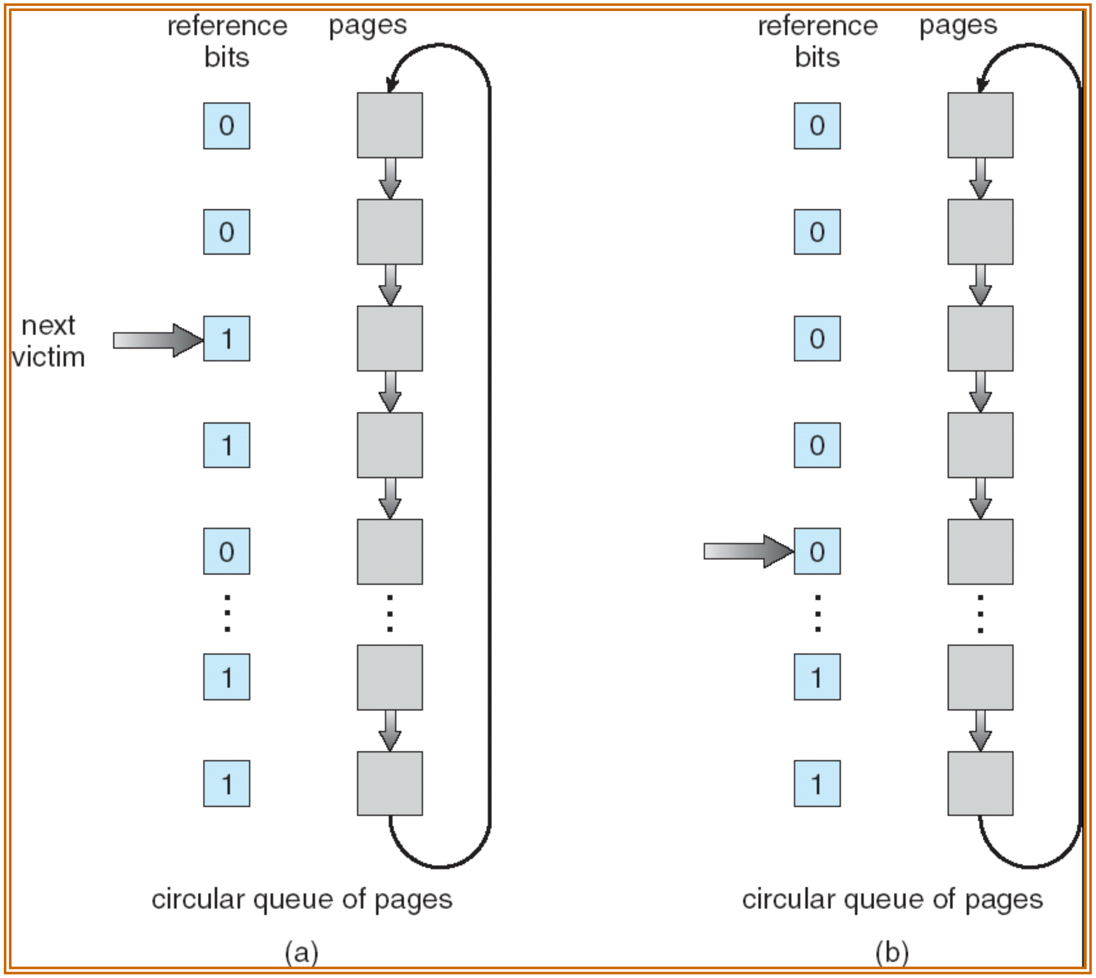
Use of Stack to record the most recent Page Reference



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit
  - Clock replacement
  - If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second Chance (Clock) Page Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.

- ***LFU Algorithm***: Replaces Page with smallest count.

- ***MFU Algorithm***: Based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Reference List

Operating Systems Concepts By Silberschatz & Galvin,

Operating systems By D M Dhamdhere,

System Programming By John J Donovan,

www.os-book.com

www.cs.jhu.edu/~yairamir/cs418/os2/sld001.htm

http://gaia.ecs.csus.edu/~zhangd/oscal/pscheduling.html

http://www.edugrid.ac.in/iiitmk/os/os_module03.htm

http://williamstallings.com/OS/Animations.html

etc…

Thnx…