Memory Management

Dr Tarunpreet Bhatia

Assistant Professor

CSED, TIET

Disclaimer

This is NOT A COPYRIGHT MATERIAL

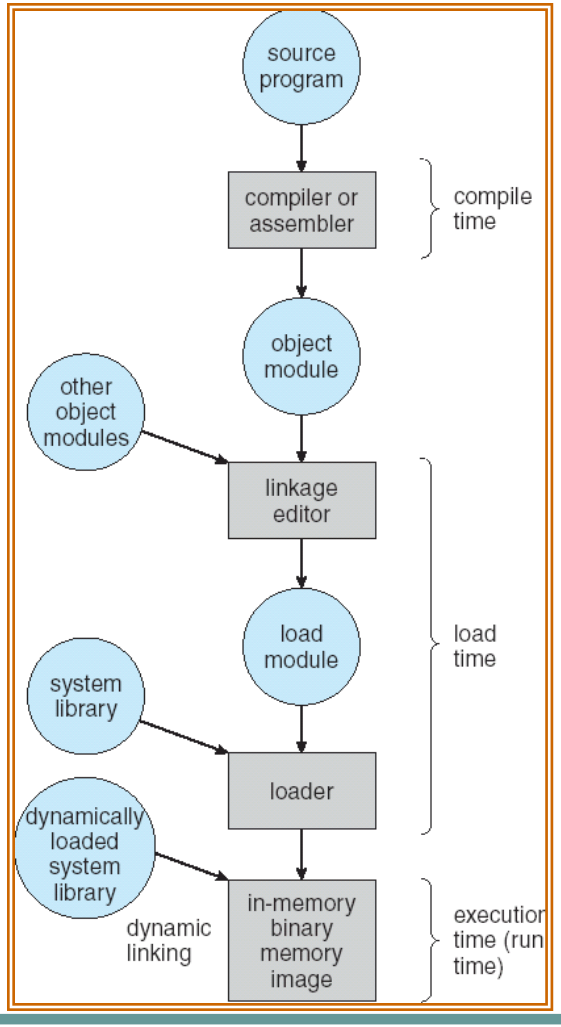
Content has been taken mainly from the following books:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings

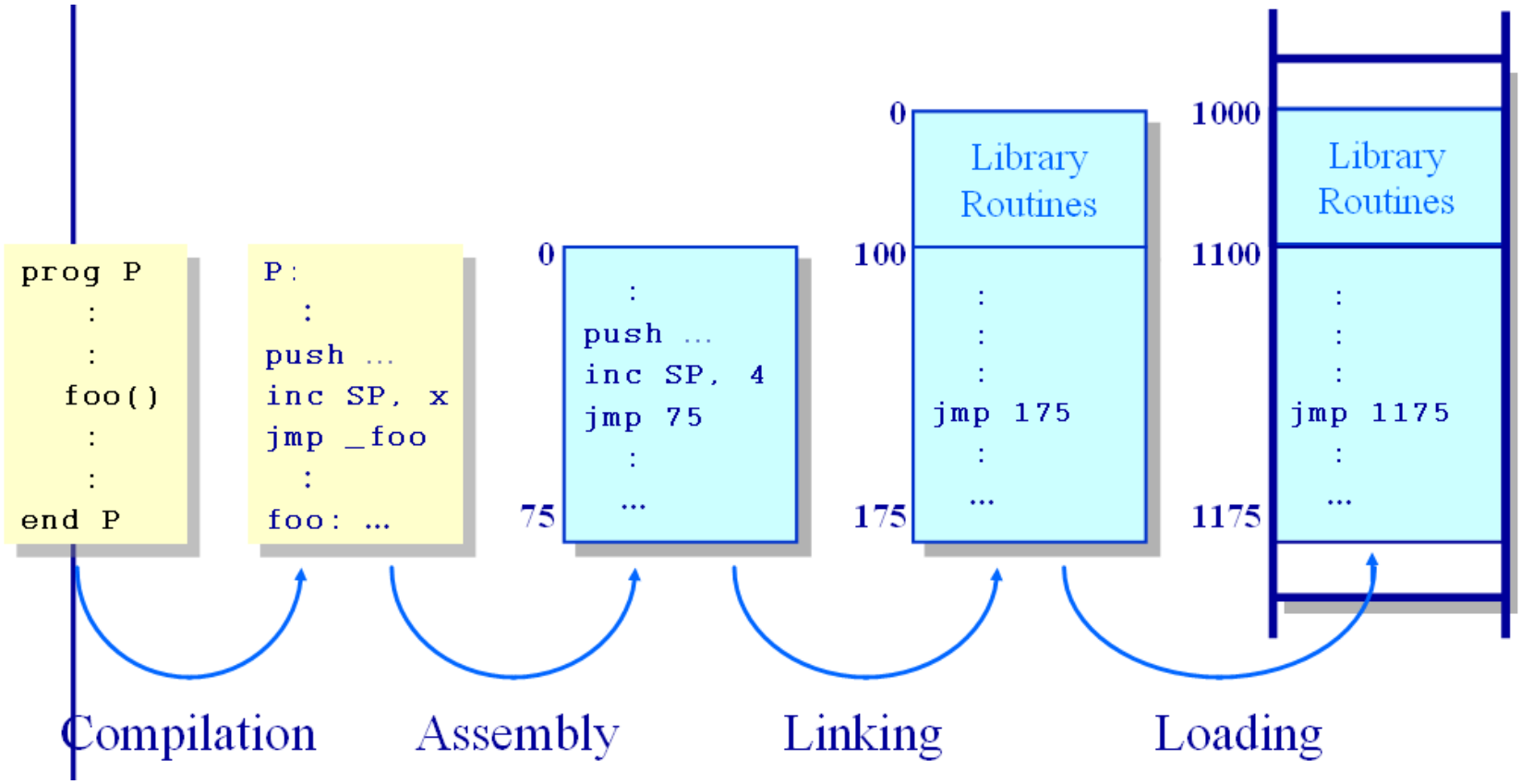
Introduction

- 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

Classically, the binding of instructions and data to memory addresses can be done at any step along the way:

a) **Compile time**. The compiler translates symbolic addresses to absolute addresses. If you know at compile time where the process will reside in memory, then absolute code can be generated (Static). If starting location changes, then it will be necessary to recompile this code.

b) **Load time**. The compiler translates symbolic addresses to relative (relocatable) addresses. The loader translates these to absolute addresses. If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code (Static).

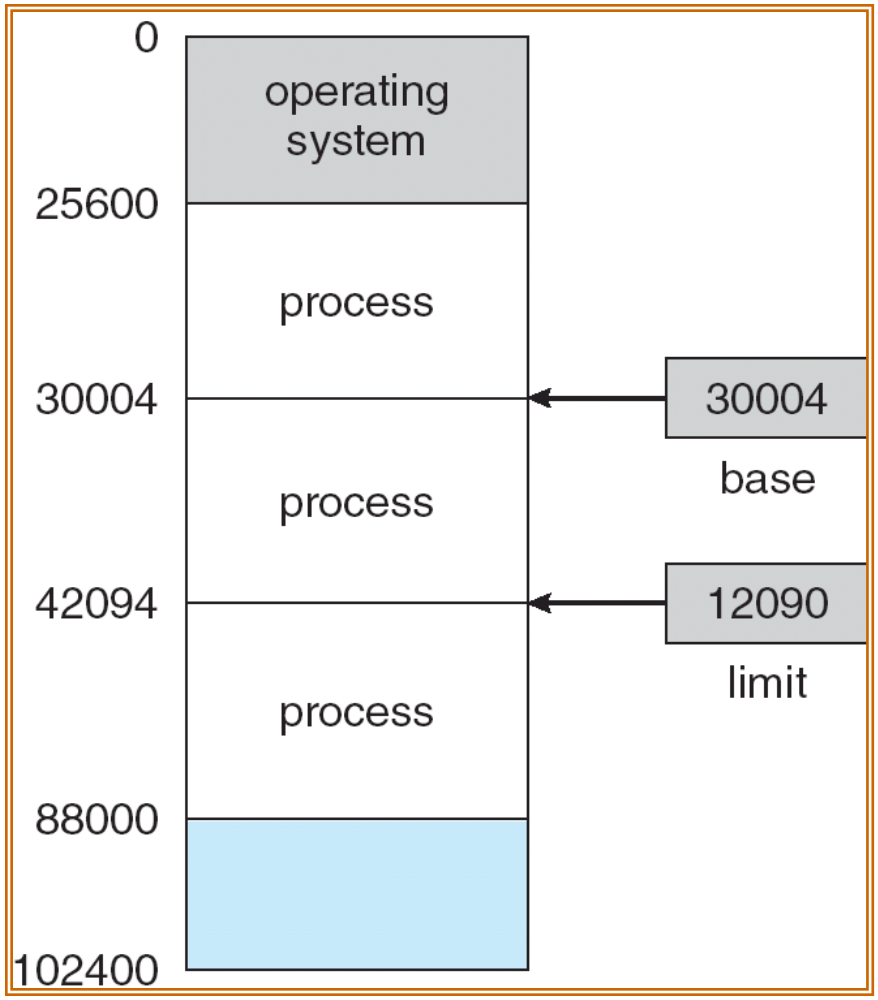
c) **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. The absolute addresses are generated by hardware. Most general-purpose OSs use this method (Dynamic).

Logical/Physical Address

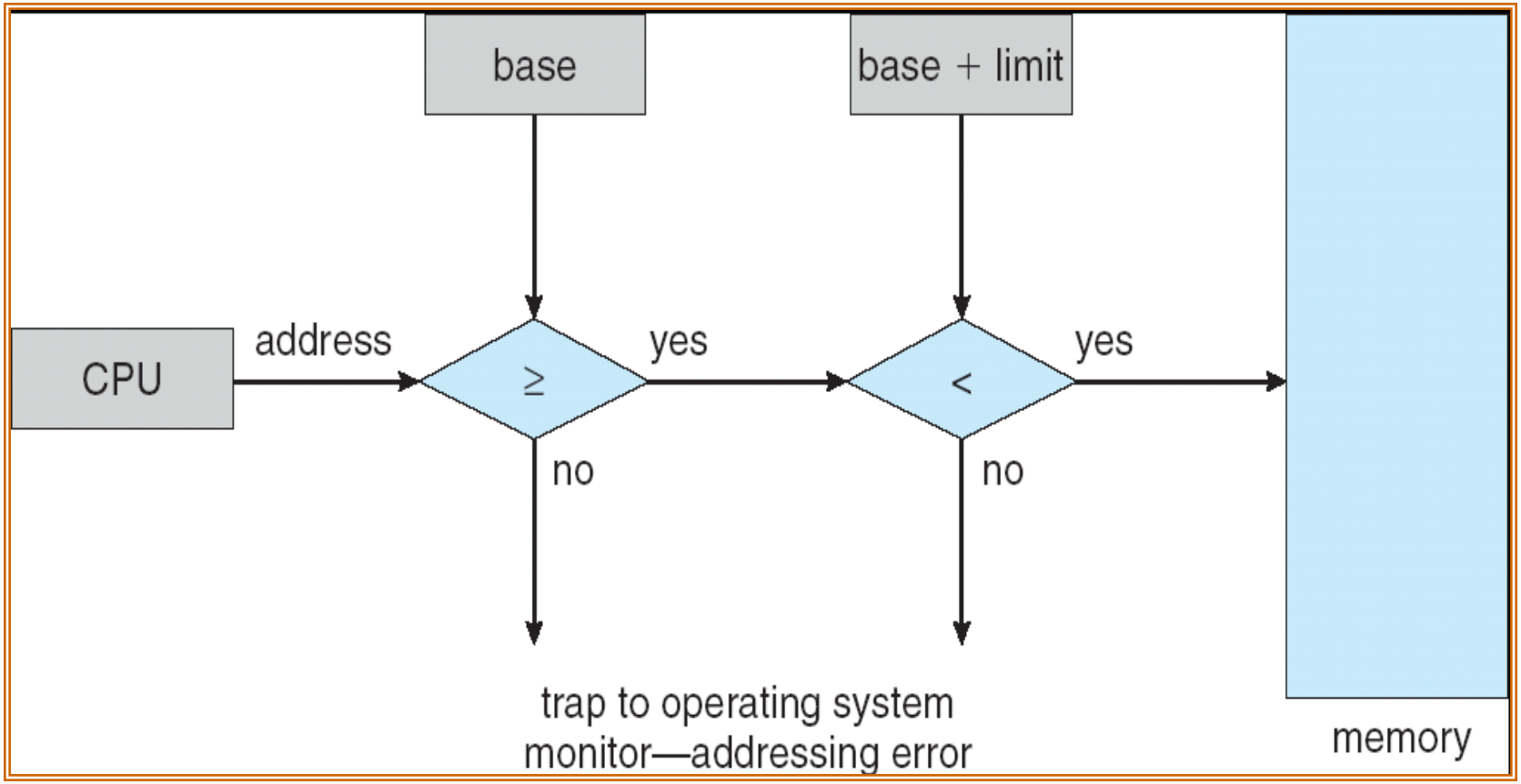
- Logical Address – Generated by the CPU, also referred to as virtual address
- Physical Address – Address seen by the memory unit

- 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



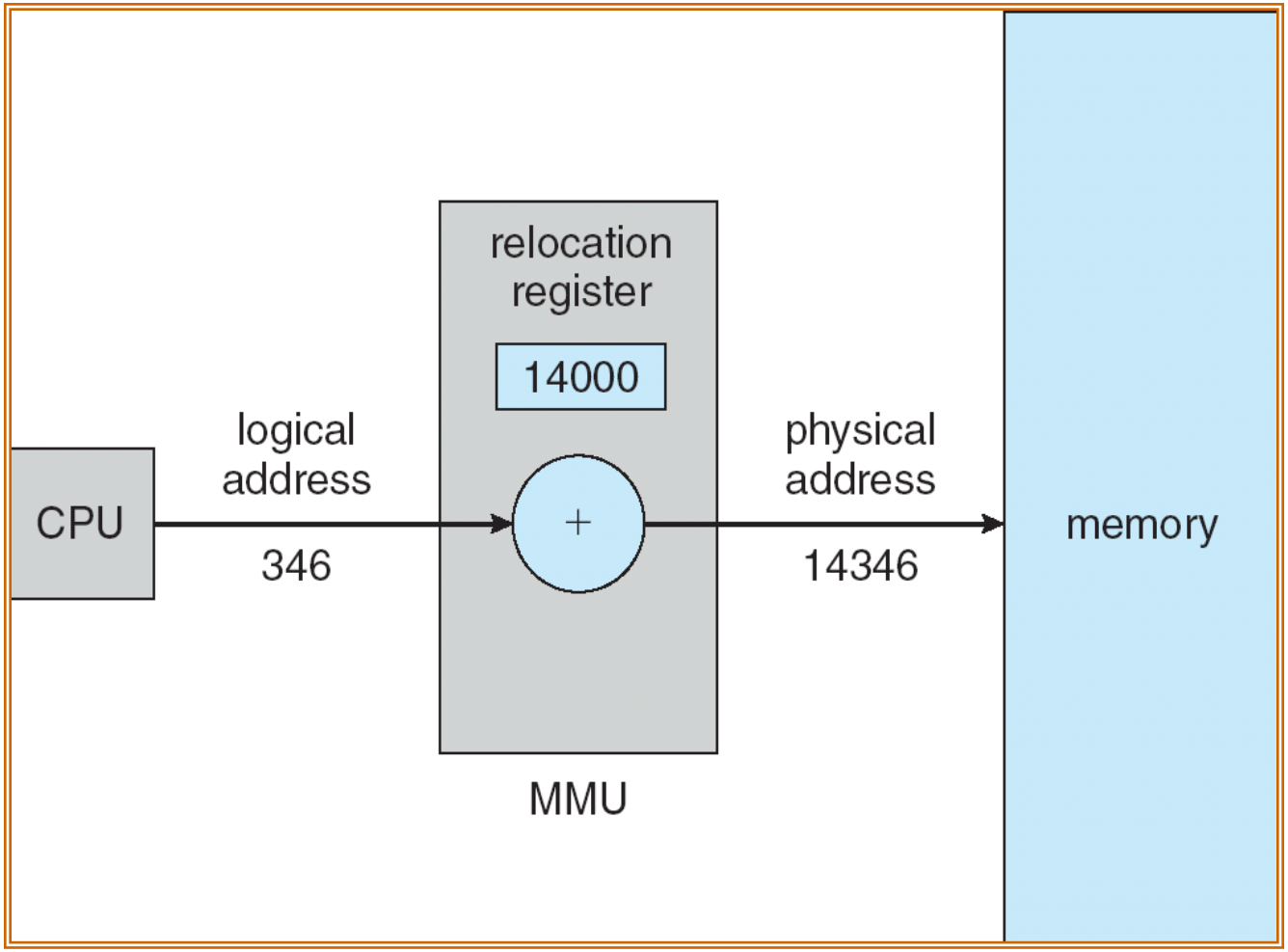
H/W Protection



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

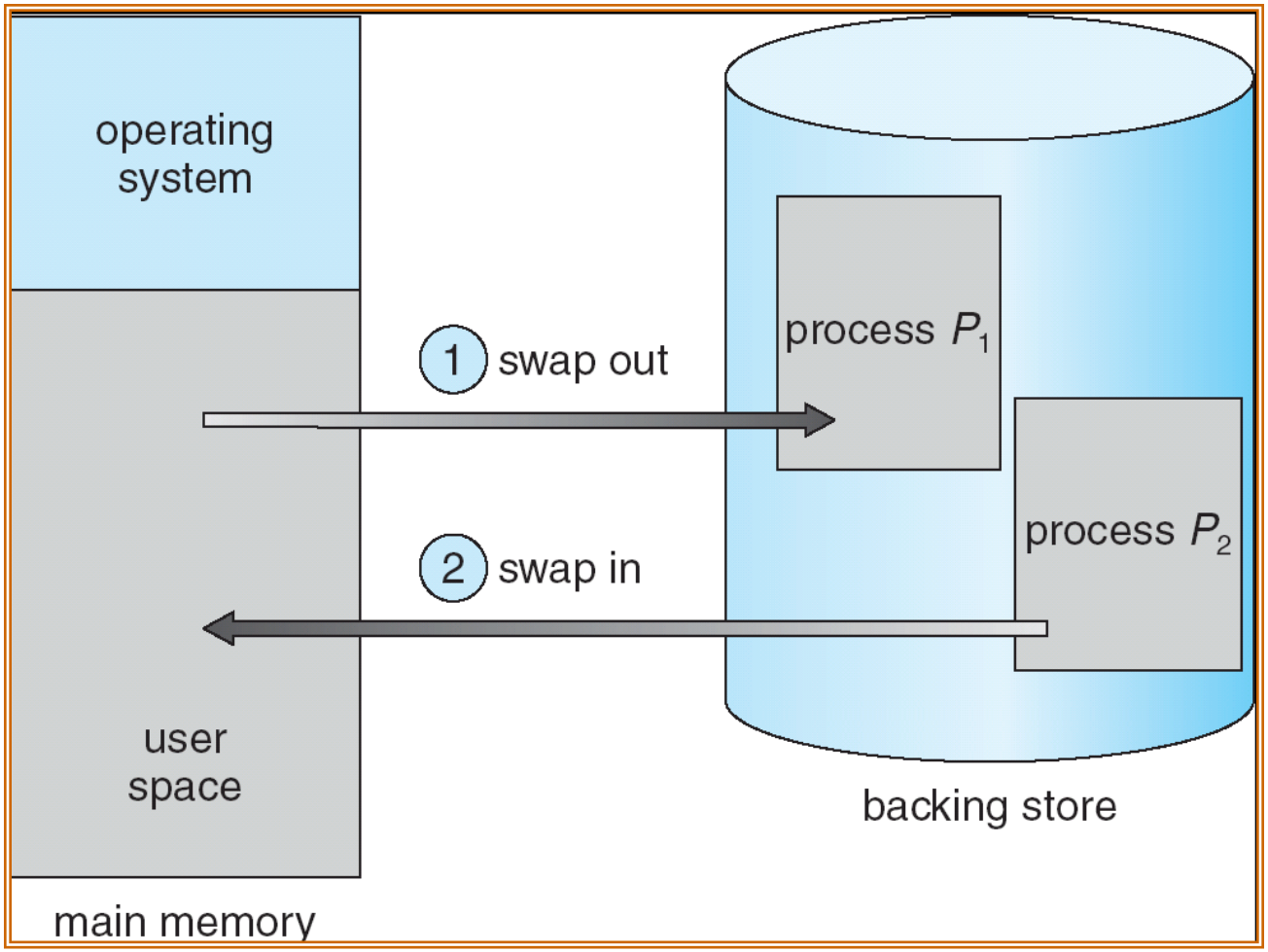
Memory Management Unit (MMU)



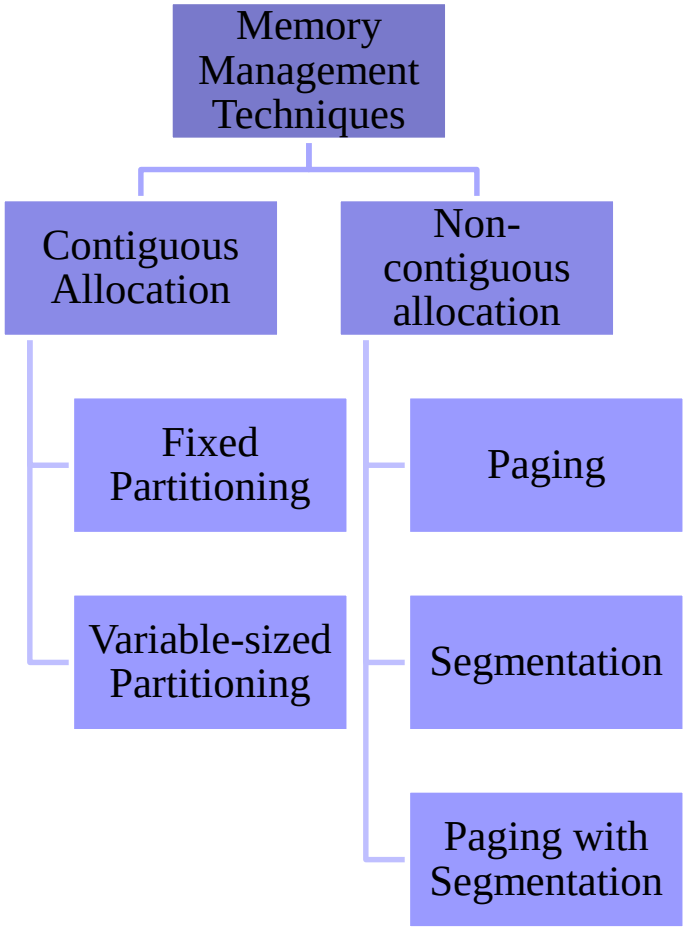
Swapping

- 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…)



Memory Management Techniques



Contiguous Allocation

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

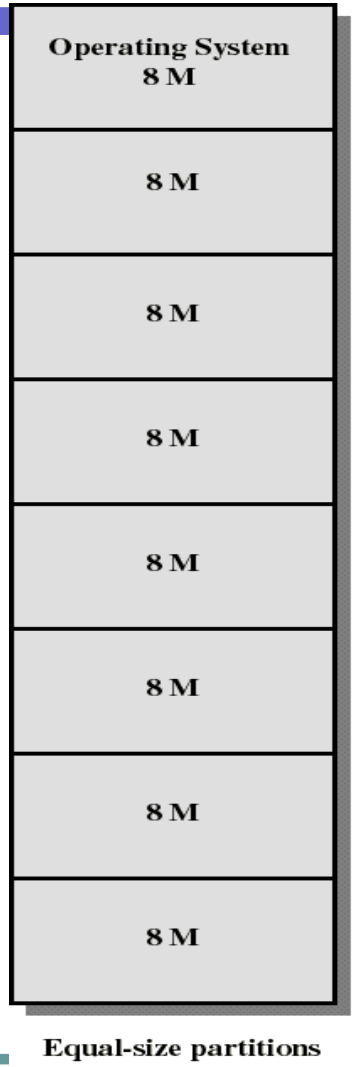
  Multiple-partition allocation

  - 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:

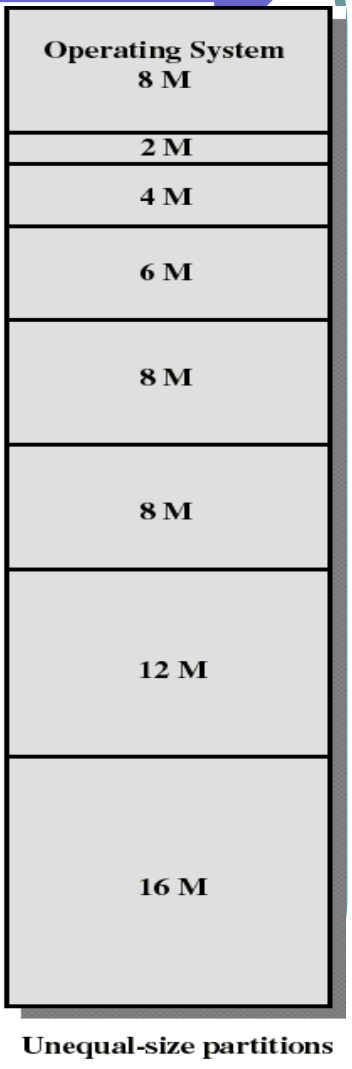
    a) Allocated Partitions b) Free Partitions (hole)

Fixed Partitioning

- Partition main memory into a set of non-overlapping memory regions called partitions.
- Fixed partitions can be of equal or unequal sizes.
- Processes are classified on entry to the system according to their memory requirements.



Equal-size partitions



Unequal-size partitions

Advantages and Disadvantages

Advantages

1. Easy to implement
2. Little OS overhead

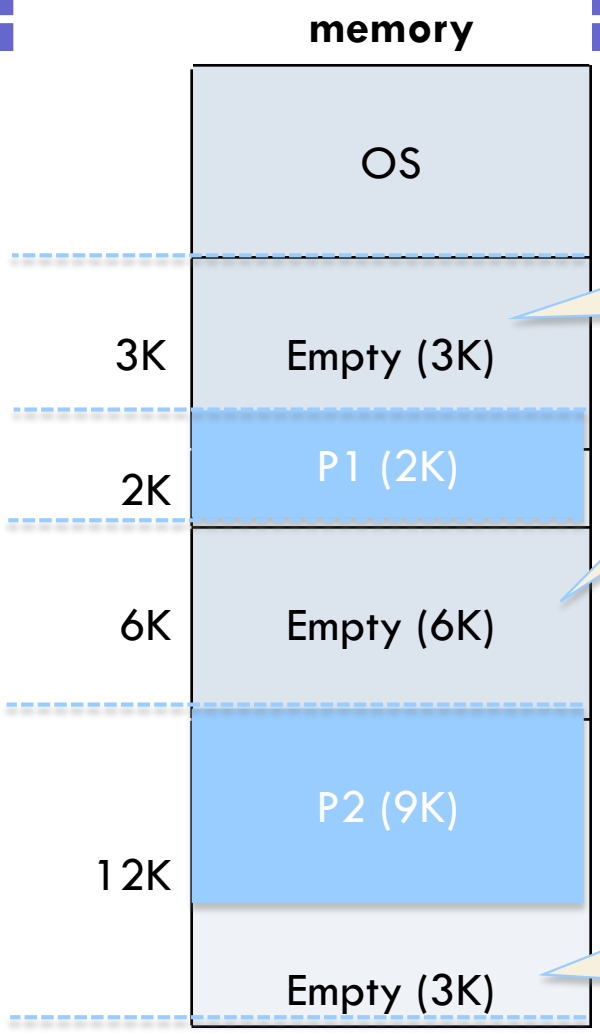
Disadvantages

1. Internal Fragmentation
2. External Fragmentation
3. Limit process size
4. Limitation on Degree of Multiprogramming

Fragmentation

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous. Although we have total space available that is needed by a process still we are not able to put that process in the memory because that space 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

Fragmentation



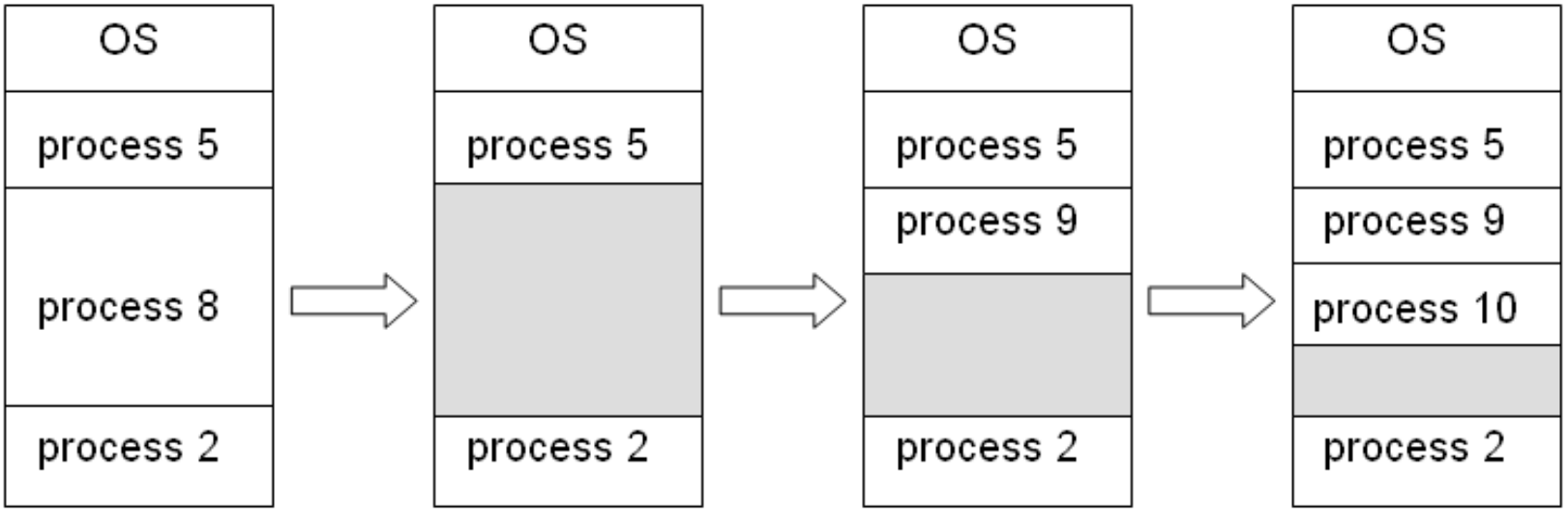
If P3 comes (9K), it is not given space as contiguous space of 9K is not available. This is ***external fragmentation.***

If a partition is being used by a process requiring some memory smaller than the partition size, then it is called an ***internal fragmentation.*** P2 needs 9K but 12K partition is given so 3K is internal fragmentation

Variable Partitioning

- Partition sizes may vary dynamically.
- In the variable partitioning method, we keep a table (linked list) indicating used/free areas in memory.
- Initially, the whole memory is free and it is considered as one large block.
- When a new process arrives, the OS searches for a block of free memory large enough for that process.
- We keep the rest available (free) for the future processes.

Variable Partitioning



Advantages and Disadvantages

Advantages

1. No Internal Fragmentation
2. No restriction on degree of Multiprogramming
3. No Limitation on the size of the process

Disadvantages

1. Difficult Implementation
2. External Fragmentation

Partition 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

Question 1

Given 5 memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, 600 KB (in order), how would the first fit, best fit and worst fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order) in fixed and variable partitioning?

Question 2

On a system with 2^24 bytes of memory and fixed partitions, all of size 65536 bytes, how many bits must the limit register have?

Question 3

On a system using fixed partitions with sizes 2^16, 2^24 and 2^32, how many bits must the limit register have?

Question 4

Consider the main memory allocation that allows variable sized partitioning without compaction with 2 free slots and 3 slots allocated to some system processes**.** Assume that there are 4 processes and their memory size requirements (in KB) are as given below:

P1: 300, P2: 25, P3: 125: P4: 50

Show the memory allocation for the following strategies:

- Best fit
- Worst fit

Allocated	Free	Allocated	Free	Allocated
50	150	300	350	600

Question 5

Consider 800 KB memory is managed using variable partitions without compaction. It currently has three processes which occupy partitions of sizes 200 KB, 134 KB and 90 KB respectively. What is the largest allocation and smallest allocation request in KB that could be denied?

Solution

With n processes, we can create a minimum 1 free partition.
Free partition = 800 – (200+134+90) = 376
The maximum process can be denied is 377 KB
Therefore if a process comes with size 377 KB you can't allocate.

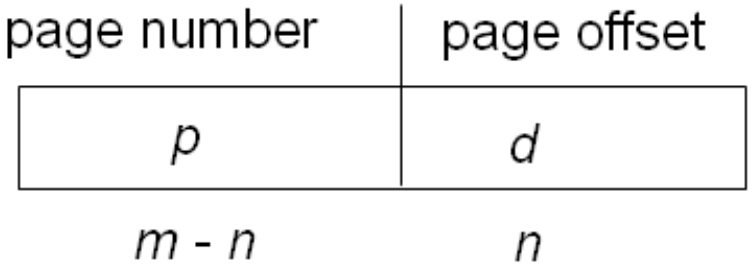
With n processes, we can create maximum n+ 1 free partitions.
Partition size = Total size available for free partitions/Available free partitions = 376/4 = 94 KB
Therefore if a process comes with the size 95 KB, we can’t allocate.

Paging

- Logical address space of a process can be non-contiguous
- 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

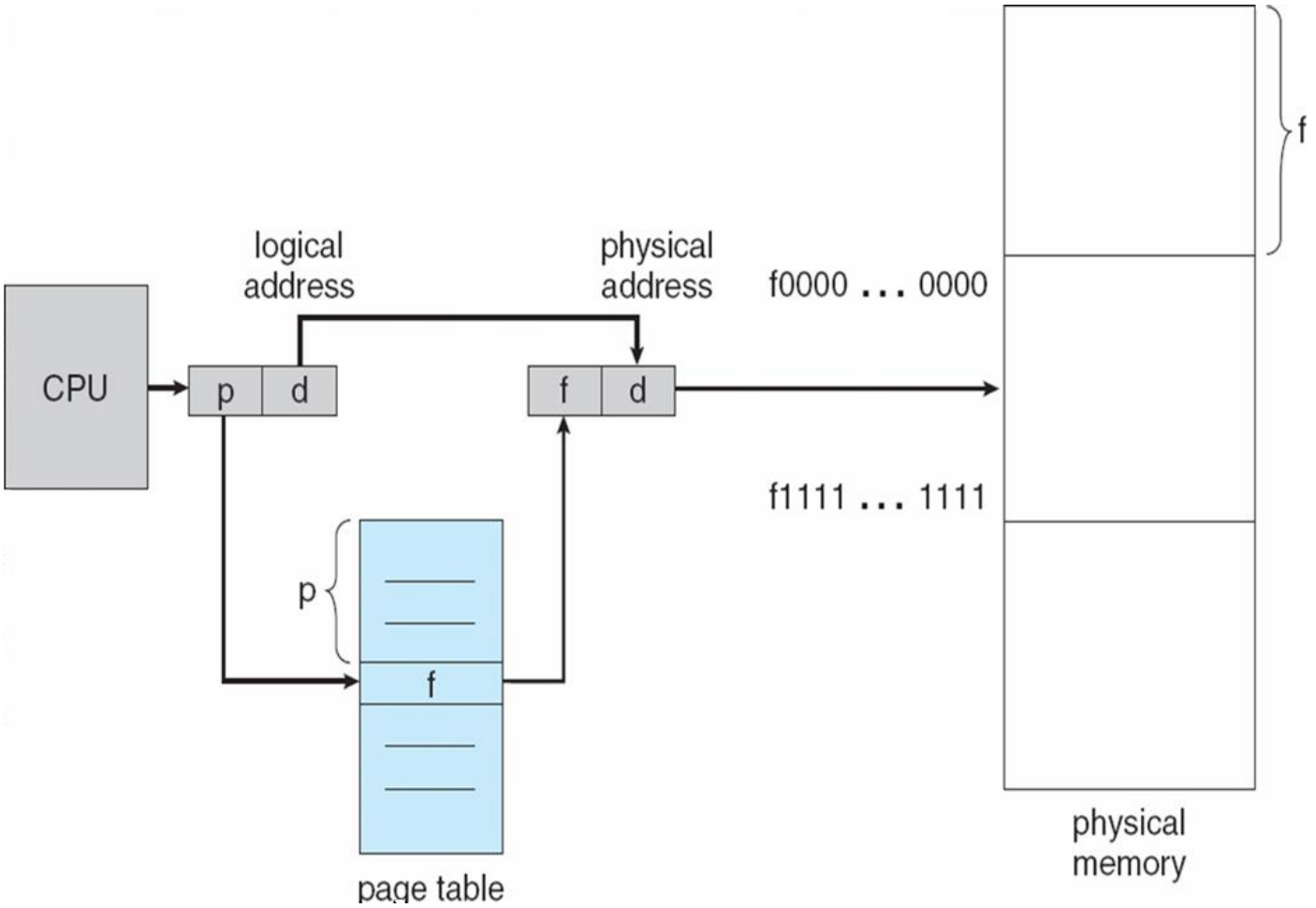
Address Translation Scheme

- 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

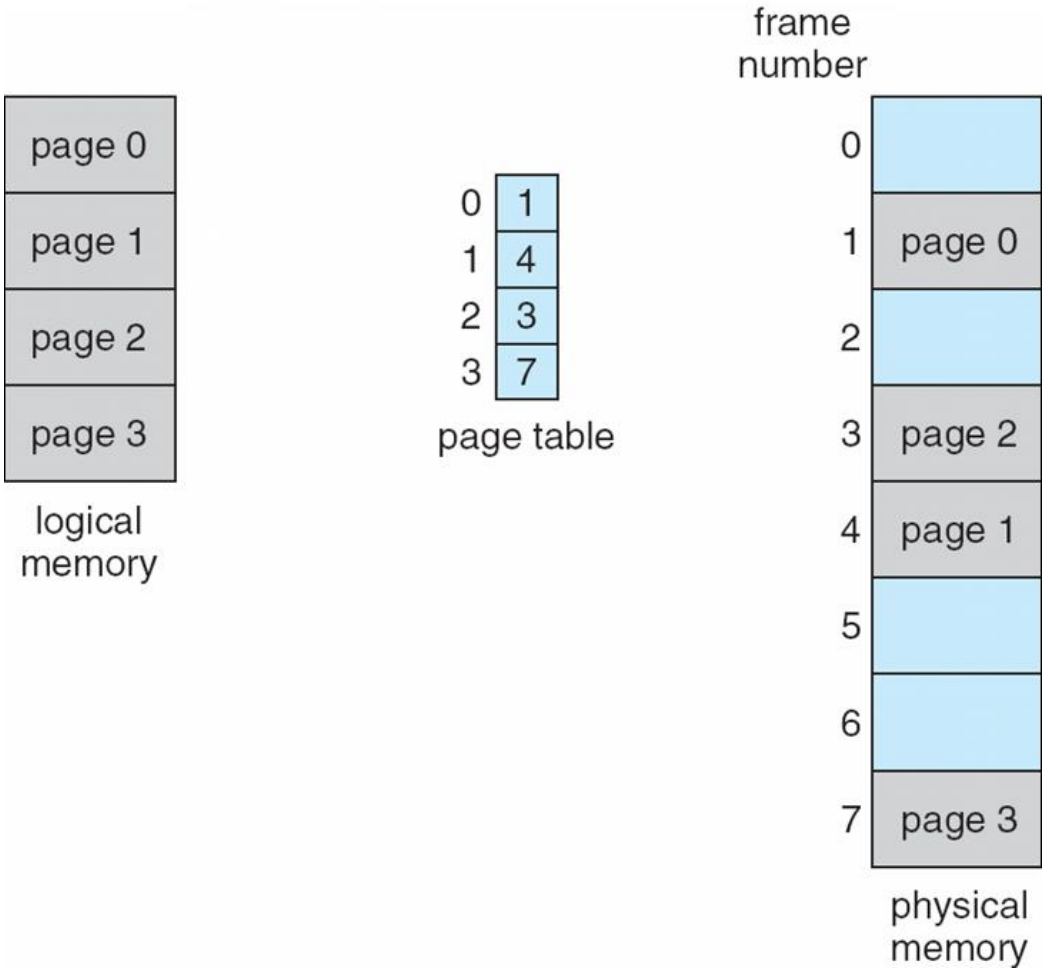


  - For given logical address space 2^m *and page size* 2^n

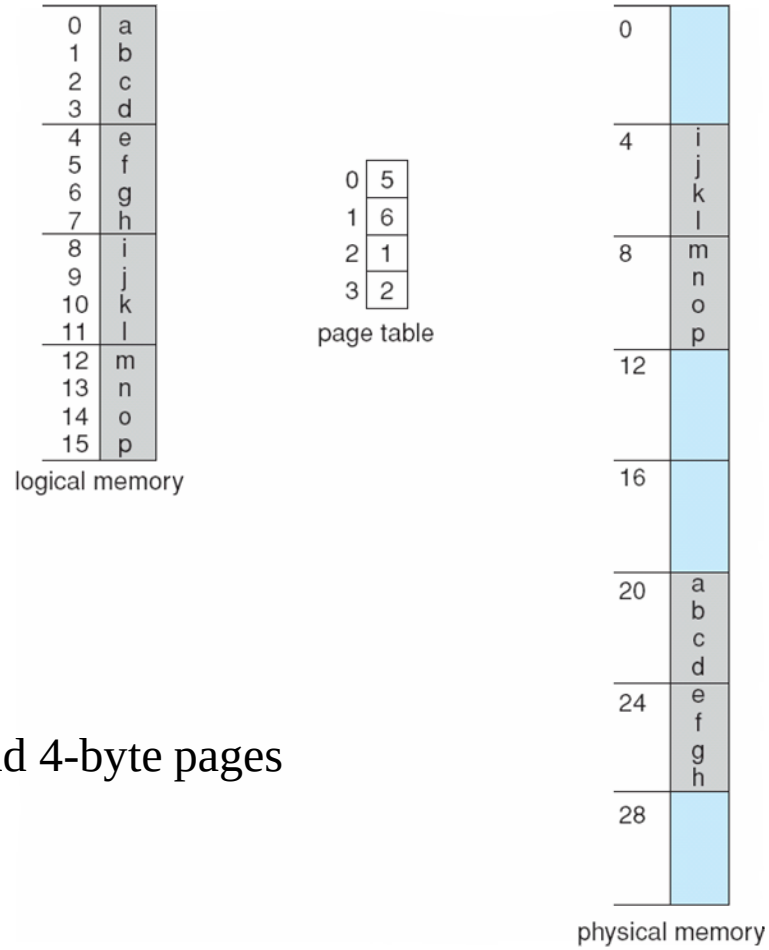
Paging Hardware



Page Modeling of Logical & Physical Memory



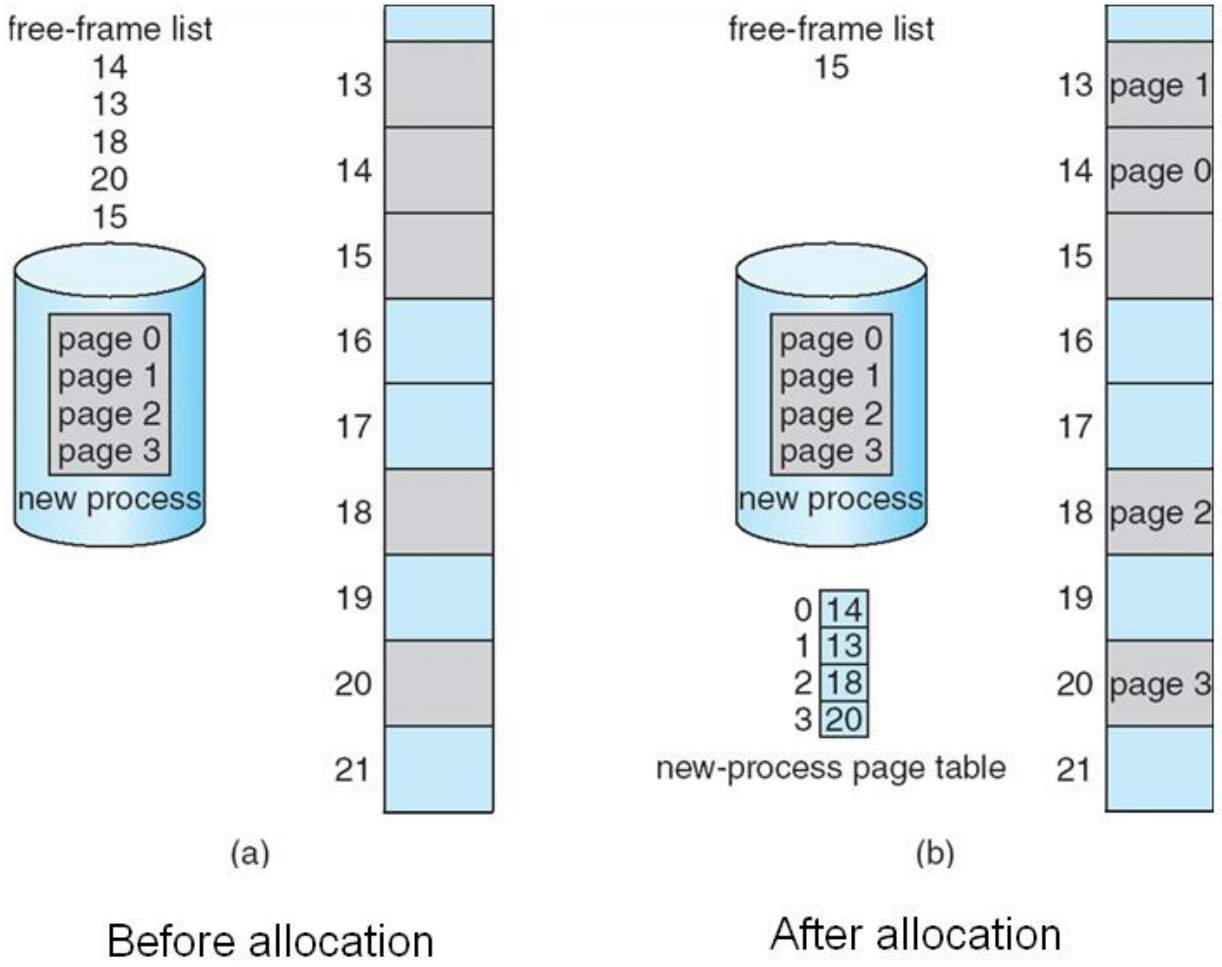
Paging Example



n=2 and m=4

32-byte memory and 4-byte pages

Free Frames



Before allocation

After allocation

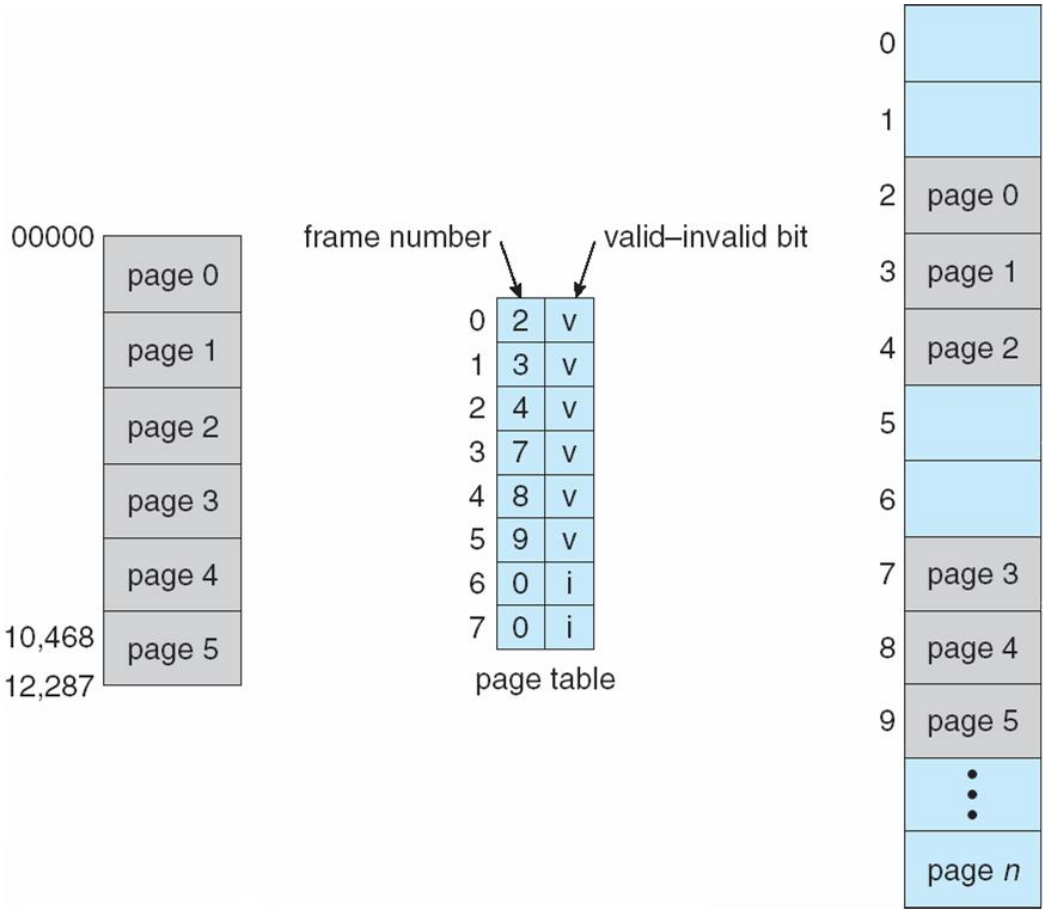
Limitations of Normal Paging

- Suffers from internal fragmentation
- Effective access time increases
- Page table occupies a significant amount of memory
- Since page table is per process, the page tables would also need to be switched during context switching.

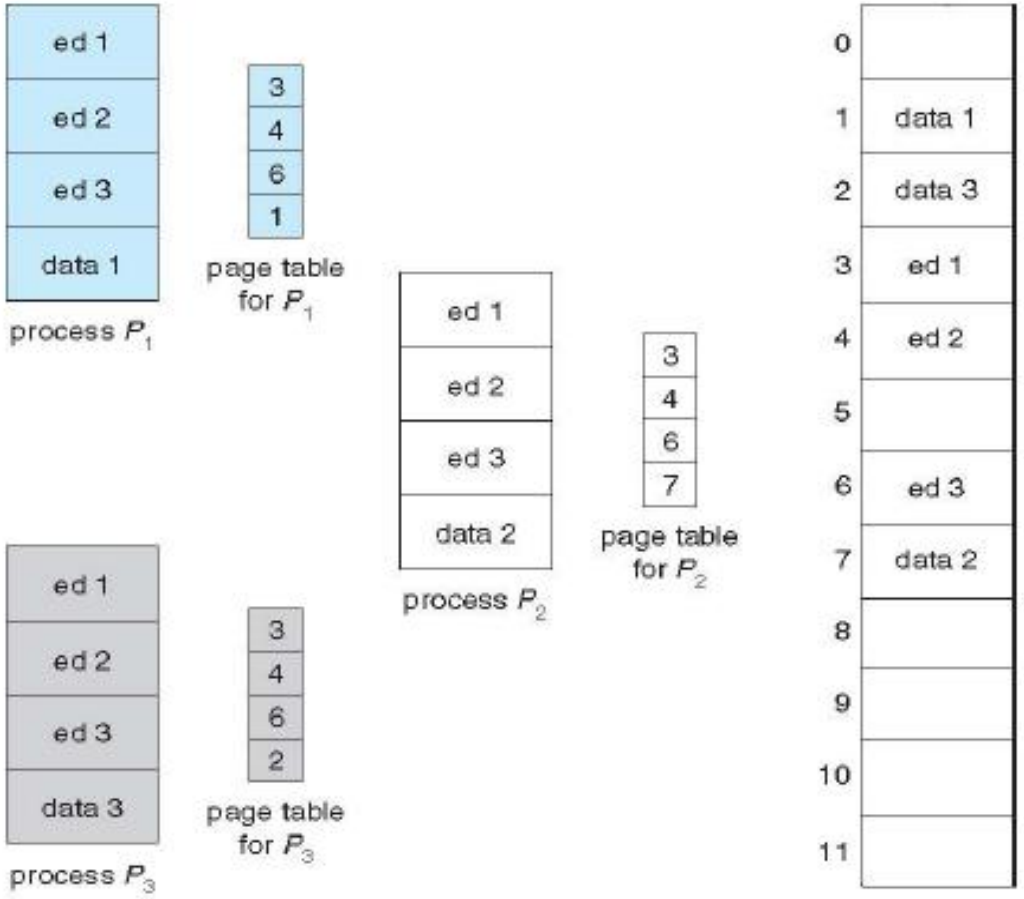
Valid Invalid Bit for Protection

- 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



Shared Pages Example



Question 6

Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size is 4KB and memory is byte addressable, what is the approximate size of the page table?

(A) 16 MB
(B) 8 MB
(C) 2 MB
(D) 24 MB

Question 7

On a simple paging system with a page table containing 64 entries of 11 bits (including valid/unvalid bit) each, and a page size of 512 bytes. Assume memory is byte addressable.

a) How many bits are there in logical address?

b) What is the size of logical address space?

c) How many bits are there in physical address?

d) What is the size of physical address space?

Question 8

Consider physical memory of 2^24 bytes, 256 pages of logical address space and page size of 1024 bytes. Assume each page table entry (PTE) contains a valid/unvalid bit in addition to the page frame number and memory is byte addressable.

a) How many bits are there in logical address?

b) How many bytes in a frame?

c) How many entries are in page table?

d) Calculate the size of page table.

Question 9

Consider a logical address space of 8 pages; each page is 2048 byte long, mapped onto a physical memory of 64 frames
(i) What are the sizes of the logical and physical address spaces?
(ii) How many bits are there in the logical address and how many bits are there in the physical address?
(iii) A 6284 bytes program is to be loaded in some of the available frames- 10, 8, 40, 25, 3, 15, 56, 18, 12, 35). Show the contents of the program's page table.
(iv) What is the size of the internal fragmentation?
(v) Convert the following logical addresses 2249, 5245, 10512 to physical addresses.

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 (PTLR) 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)

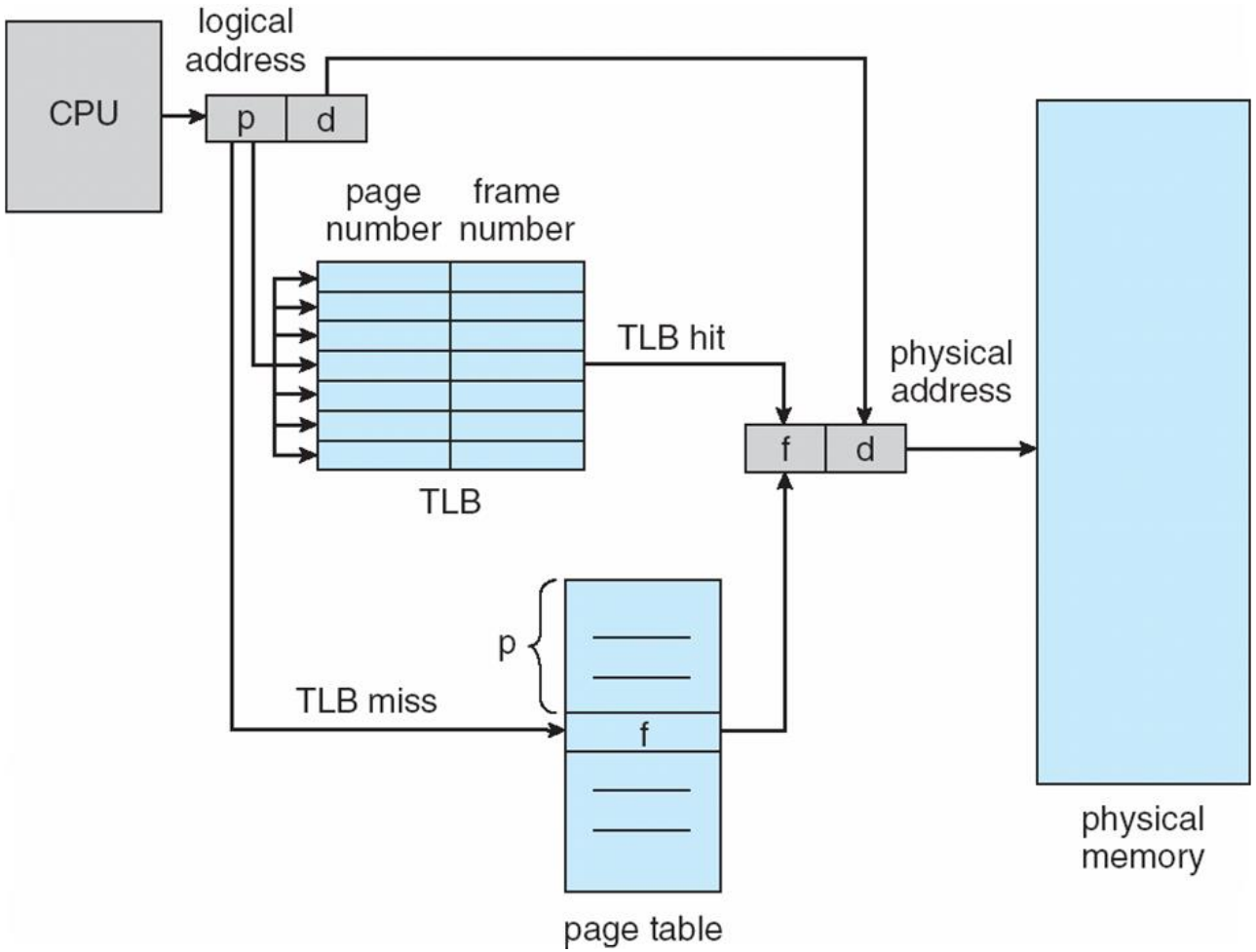
Associative Memory

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



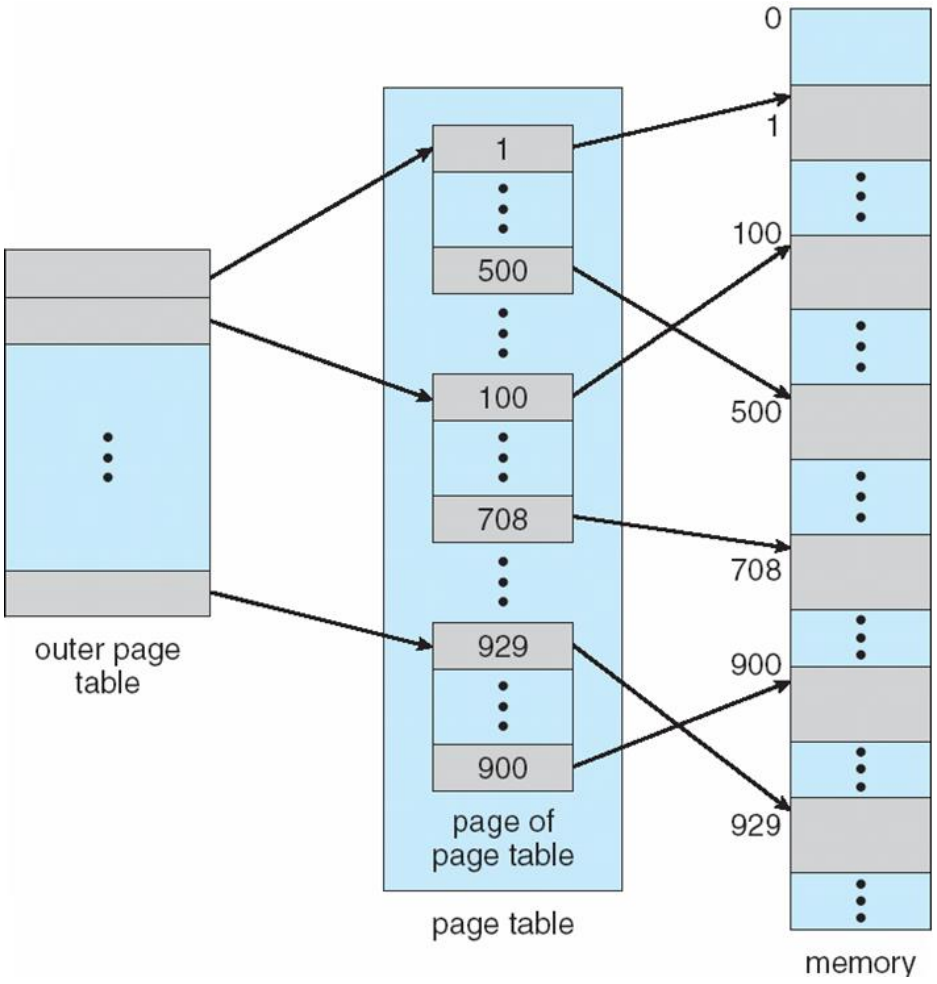
Example

Given TLB hit ratio as 0.9, RAM access time as 100 ns and TLB access time as 20 ns. Calculate reduction in effective access time with TLB.

Structure of PAGE TABLE

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

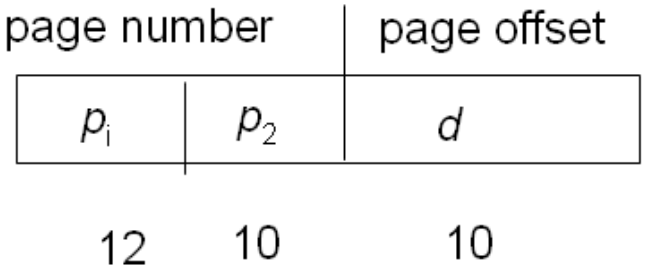
Two Level Page Table Scheme



Two Level Paging Example

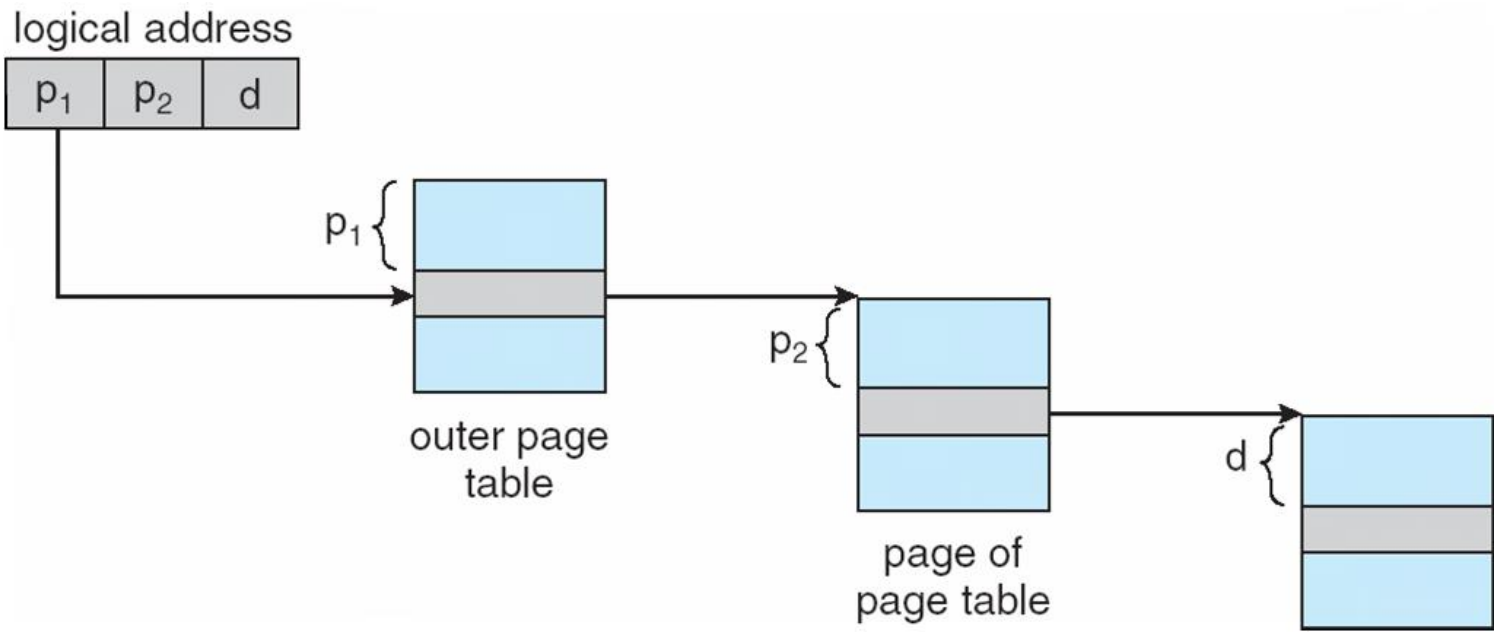
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - A Page number consisting of 22 bits
  - A Page offset consisting of 10 bits
- Since the PAGE TABLE IS PAGED, the page number is further divided into:
  - A 12-bit Page number
  - A 10-bit Page offset
- Thus, a Logical Address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

Two Level Paging Example (contd.)

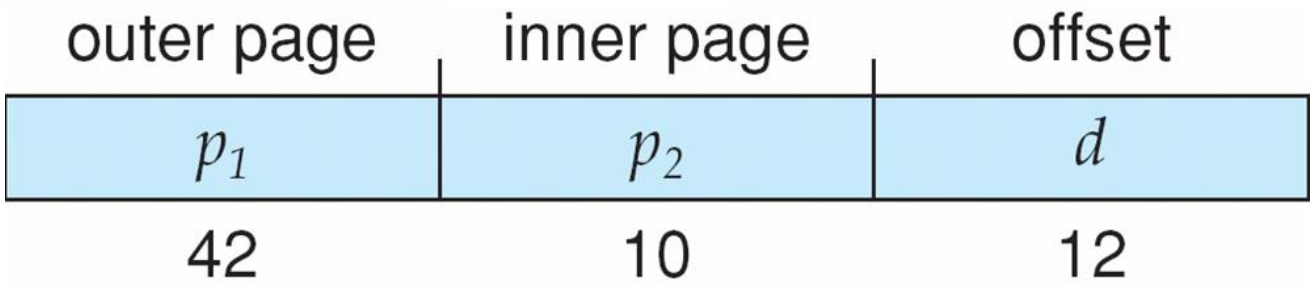


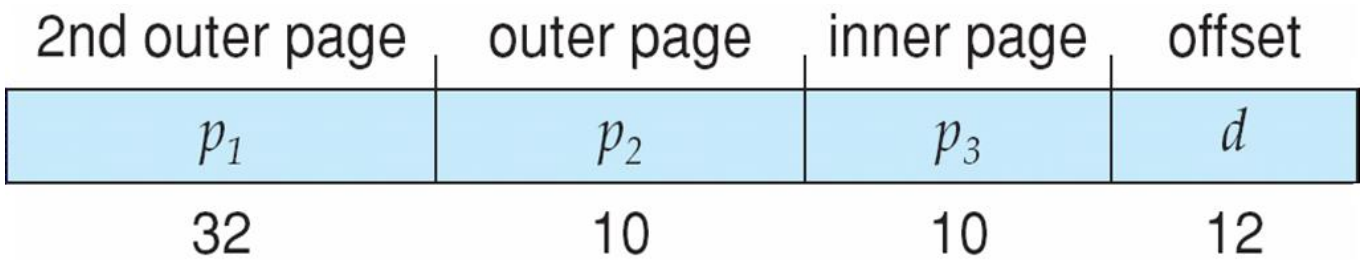
where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Address Translation Scheme



Three Level Paging Scheme

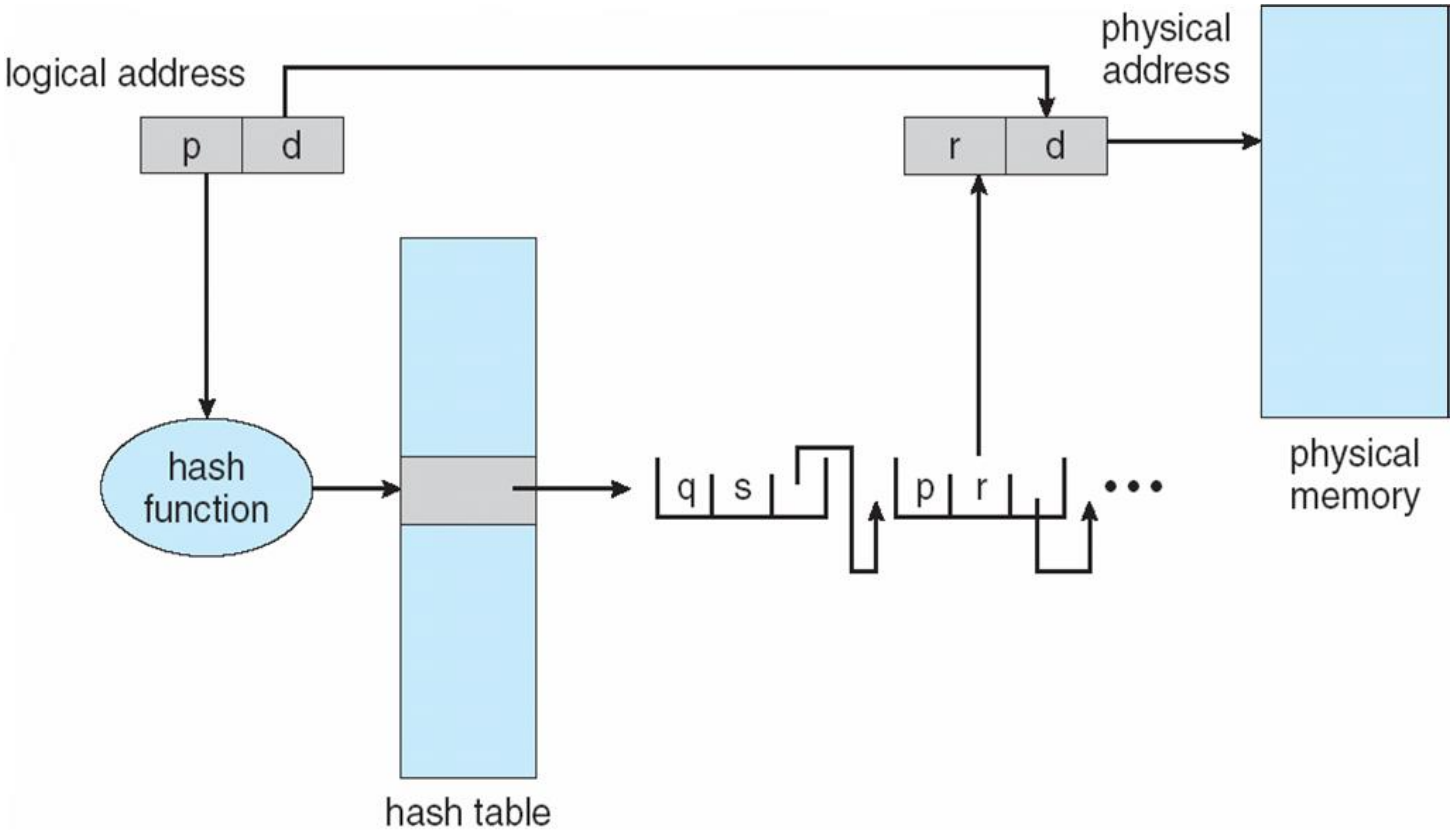




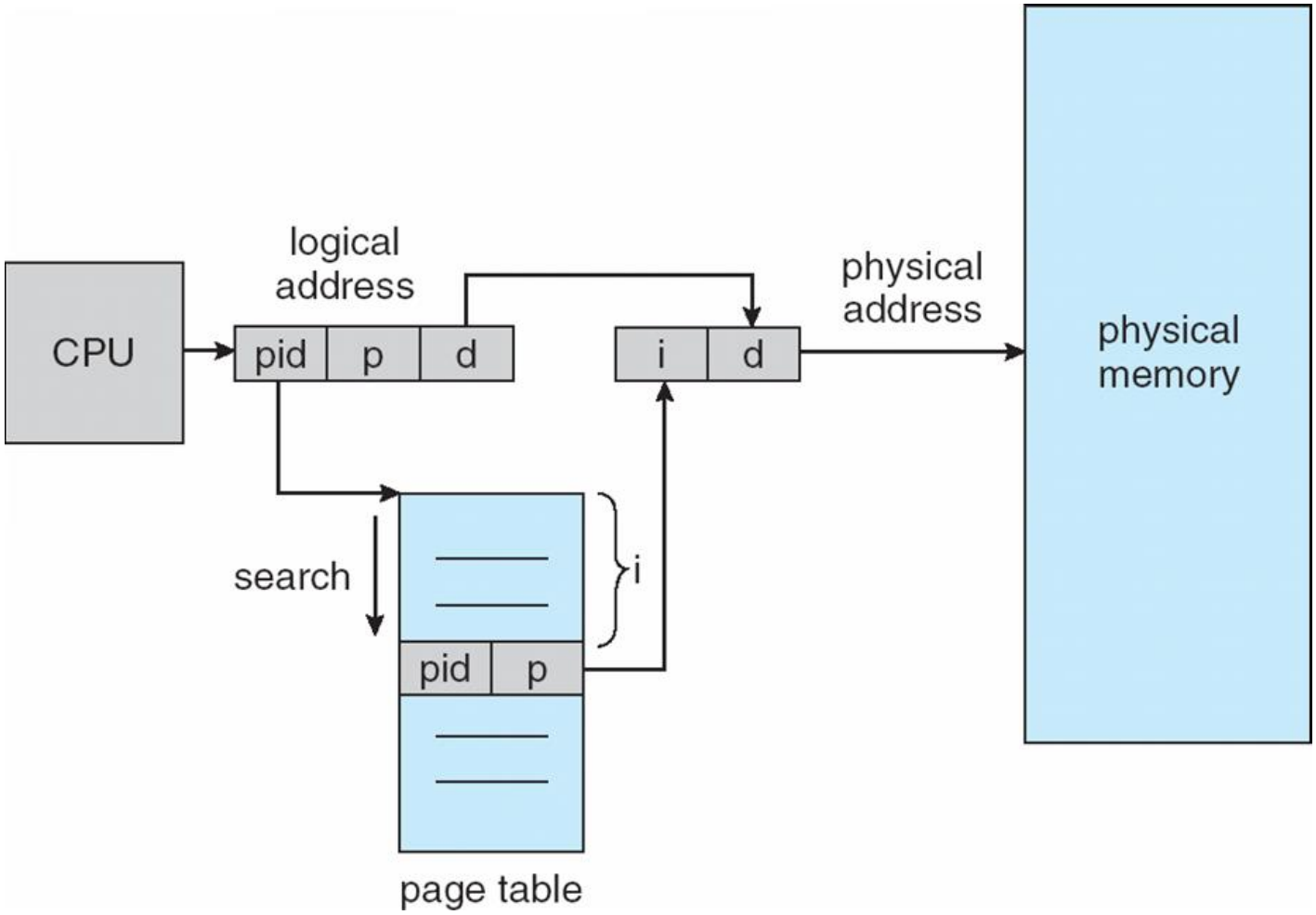
Question 10

A computer has 64 bit virtual address space and 2048-byte pages. PTE takes 4 bytes. A multi-level page table is used such that each table must be contained within a page. How many levels are required?

Hashed Page Table



Inverted Page Table



Question 11

In a 64 bit machine with 256 MB RAM and 4KB page size, how many entries will there be in the page table if it is inverted.

Question 12

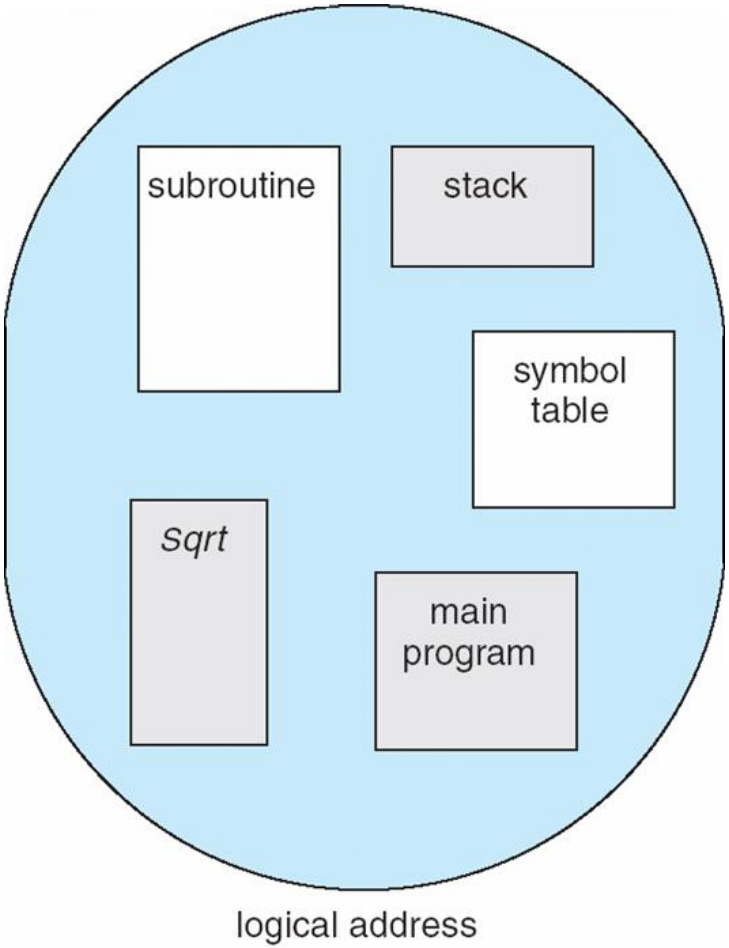
A machine has a 32-bit address space and 8KB page. The page table is entirely in hardware, with one 32-bit word per entry. When a process starts, the page table is copied to the hardware from memory, at one word every 100 nsec. If each process runs for 100 msec (including the time to load the page table), what fraction of the CPU time is devoted to loading the page tables?

Segmentation

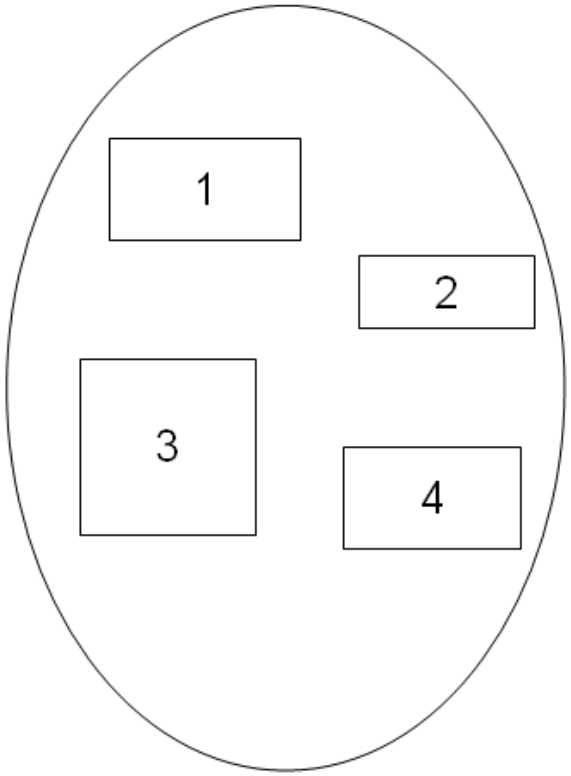
- *A Program is a collection of Segments*
  - A *SEGMENT* is a logical unit such as:

    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



user space

physical memory space

Segmentation Architecture

- Logical Address consists of a two tuple:

  <segment-number, offset>

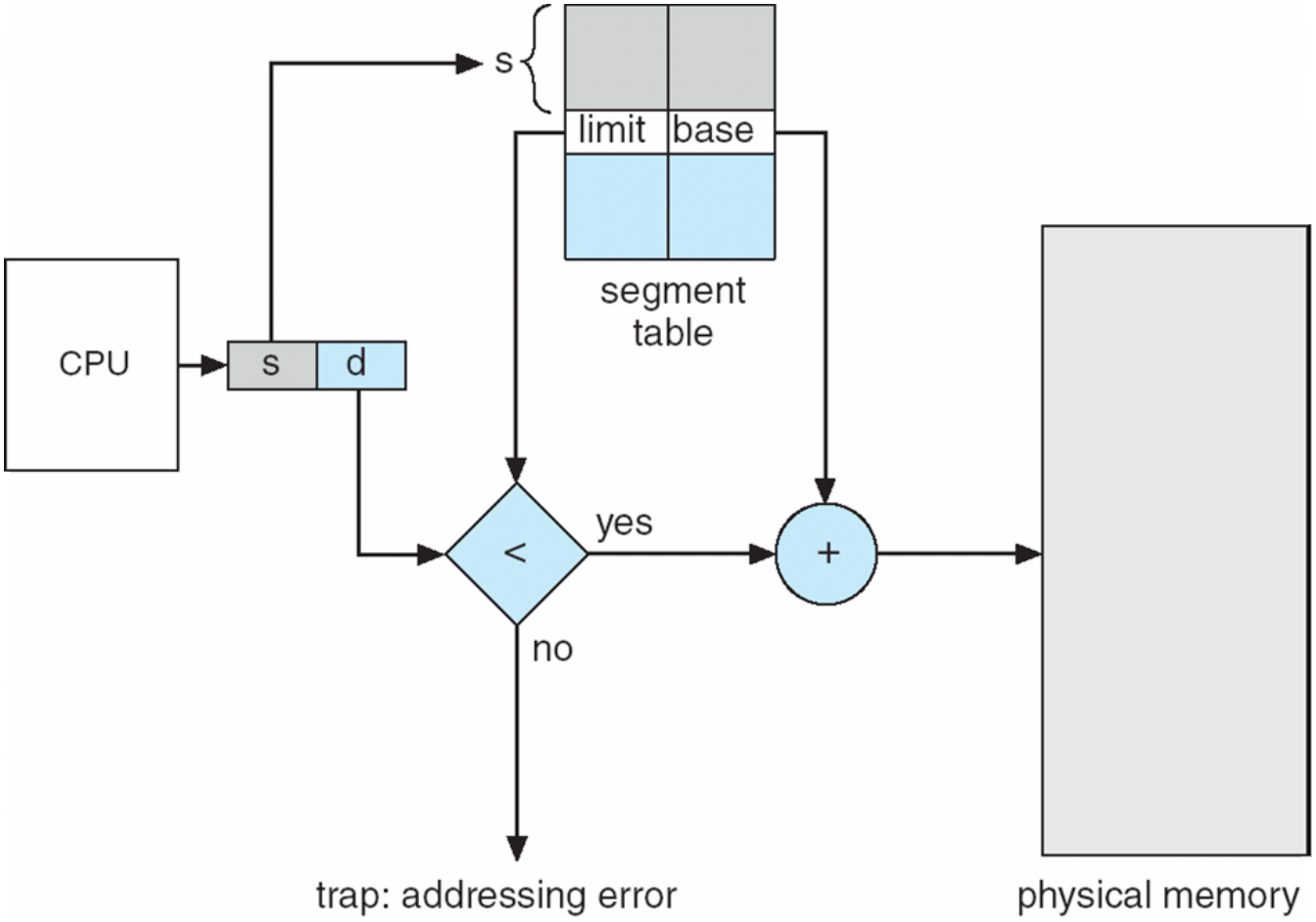
- Segment Table – Maps two-dimensional physical addresses.

- Each Table entry has:

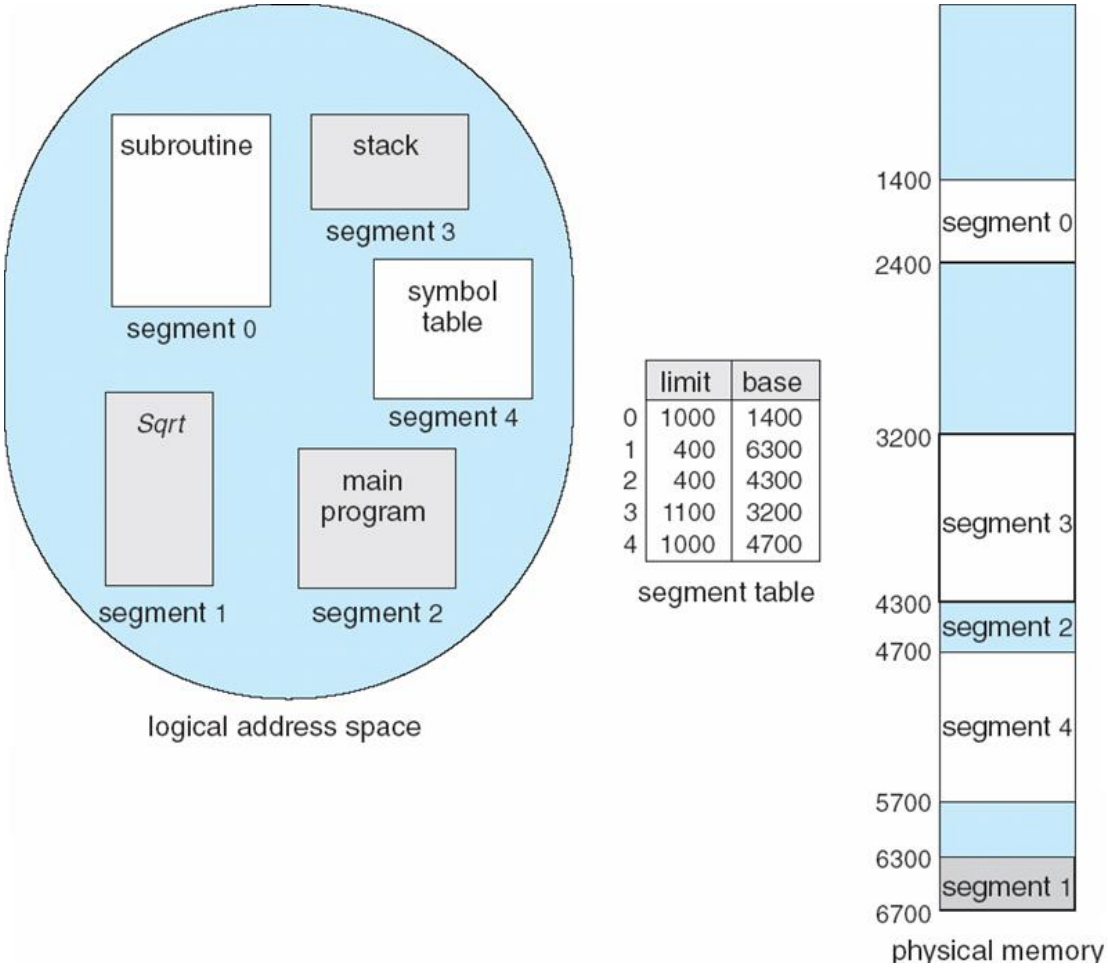
  - base – Contains the starting Physical Address where the segments reside in memory

  - limit – Specifies the Length of the Segment

Segmentation Hardware



Example of Segmentation



Question

On a system using segmentation, compute the physical addresses for each of the following logical addresses given the following segment table. Also indicate if the address generates a segment fault.

a) 0, 99 b) 2, 78 c) 1, 265 d) 3, 222 e) 0, 111

Segment No.	Base	Length
0	330	124
1	876	211
2	111	99
3	498	302