Unit IV
Memory Management

Dr.R.Subhashini
Professor and Head
Dept. of Information Technology

Memory Management

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table

Memory Management

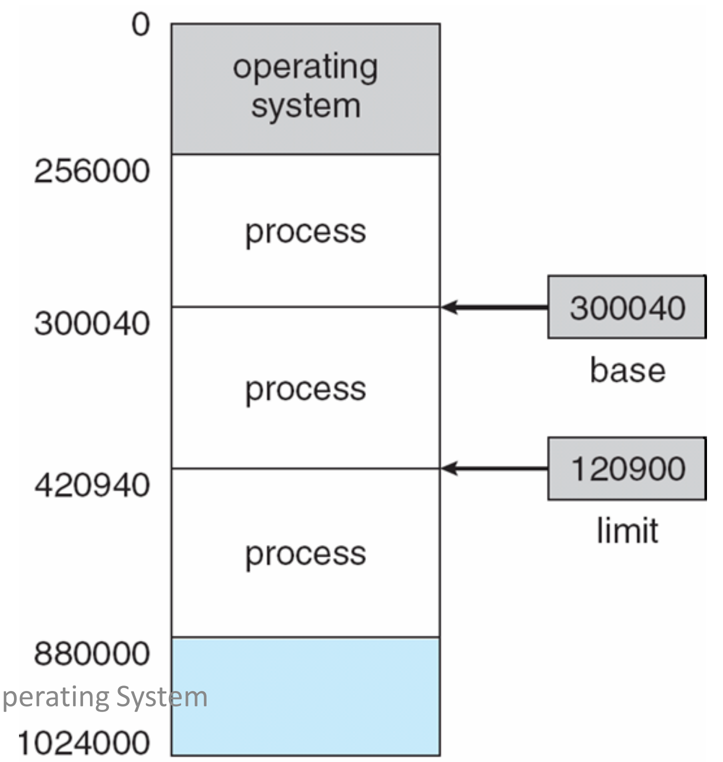
- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation

Background

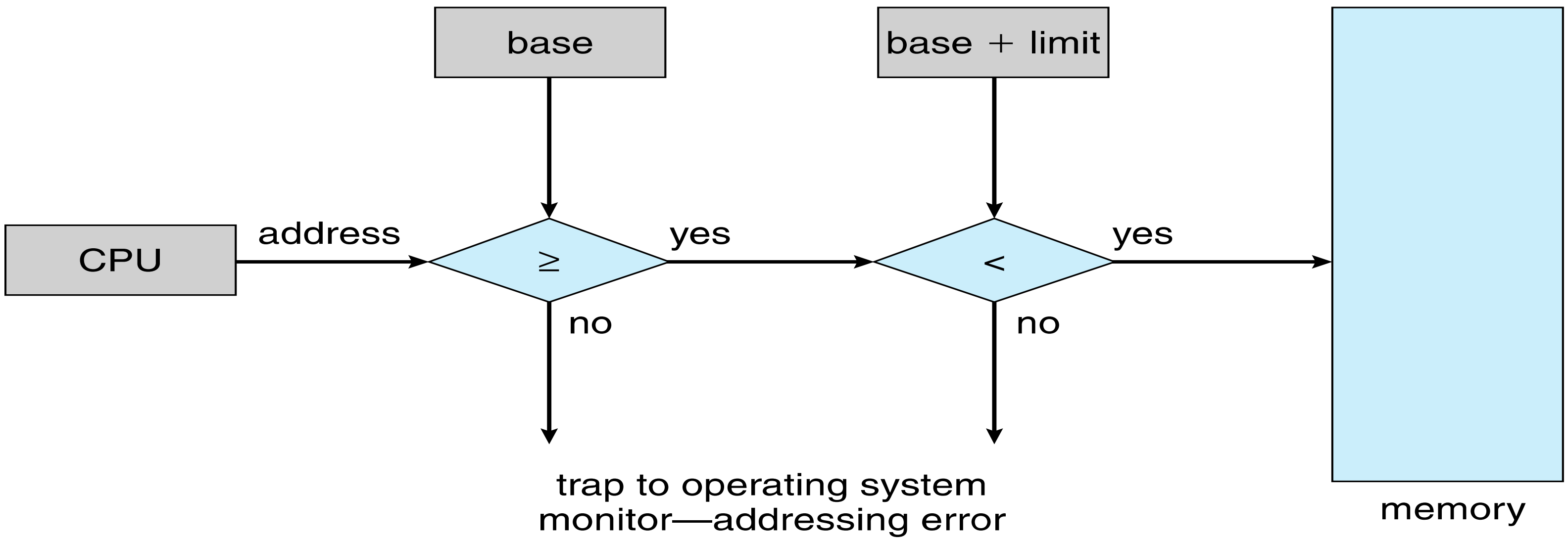
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Hardware Address Protection



Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **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
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

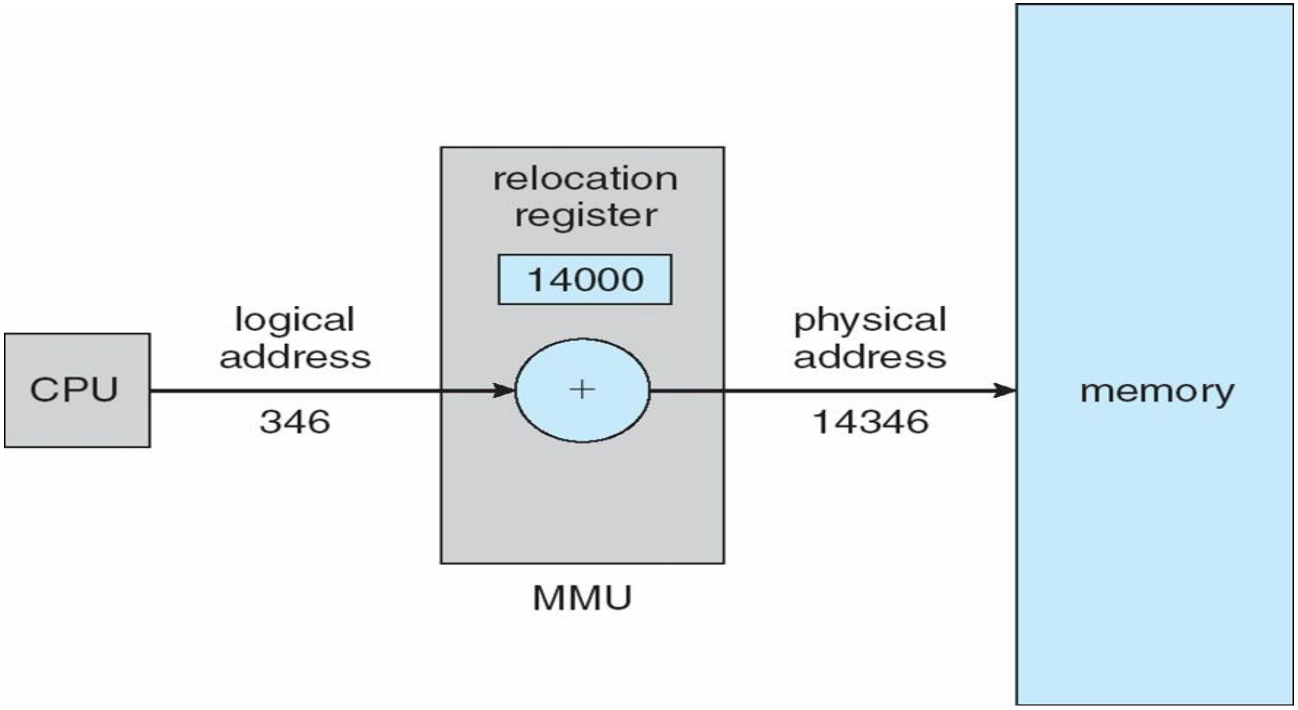
LOGICAL ADDRESS	PHYSICAL ADDRESS
It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
The user can view the logical address of a program.	The user can never view physical address of program
The user uses the logical address to access the physical address.	The user can not directly access physical address.
The Logical Address is generated by the CPU	Physical Address is Computed by MMU

Memory-Management Unit (MMU)

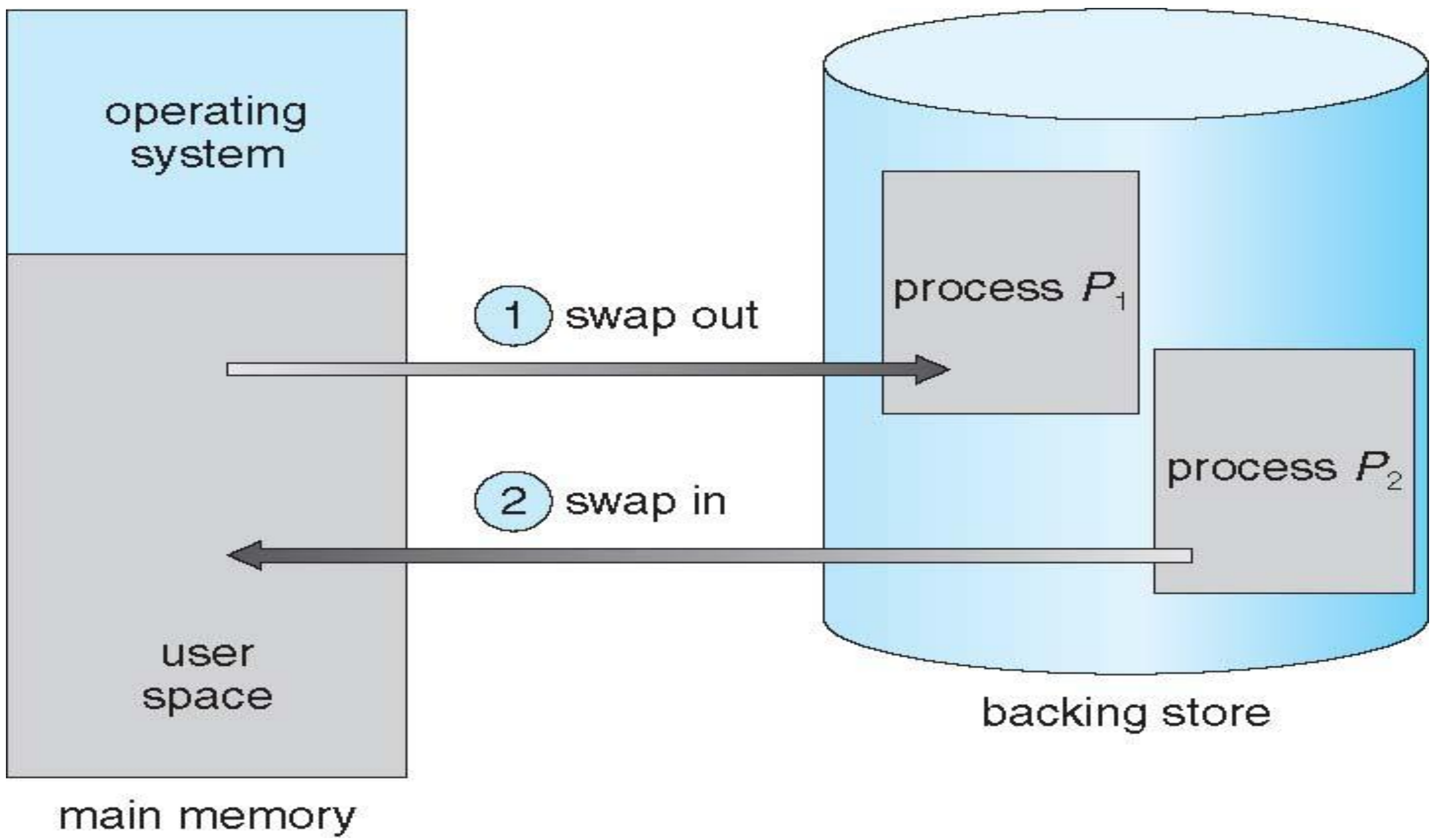
- Hardware device that at run time maps virtual to physical address
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Dynamic relocation using a relocation register

Mapping logical to physical address

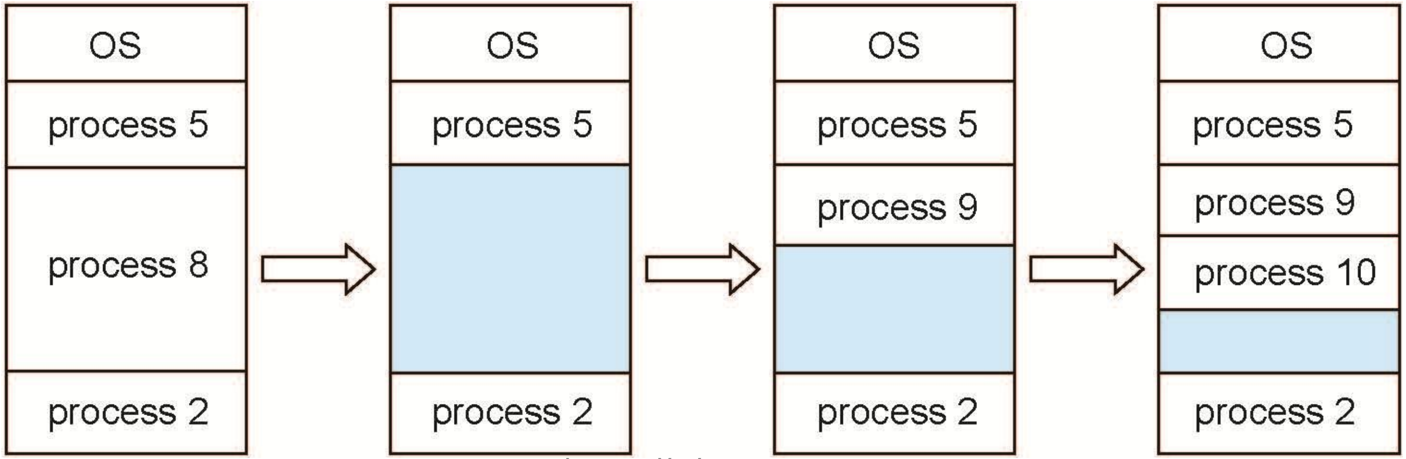


Schematic View of Swapping



Multiple-partition allocation

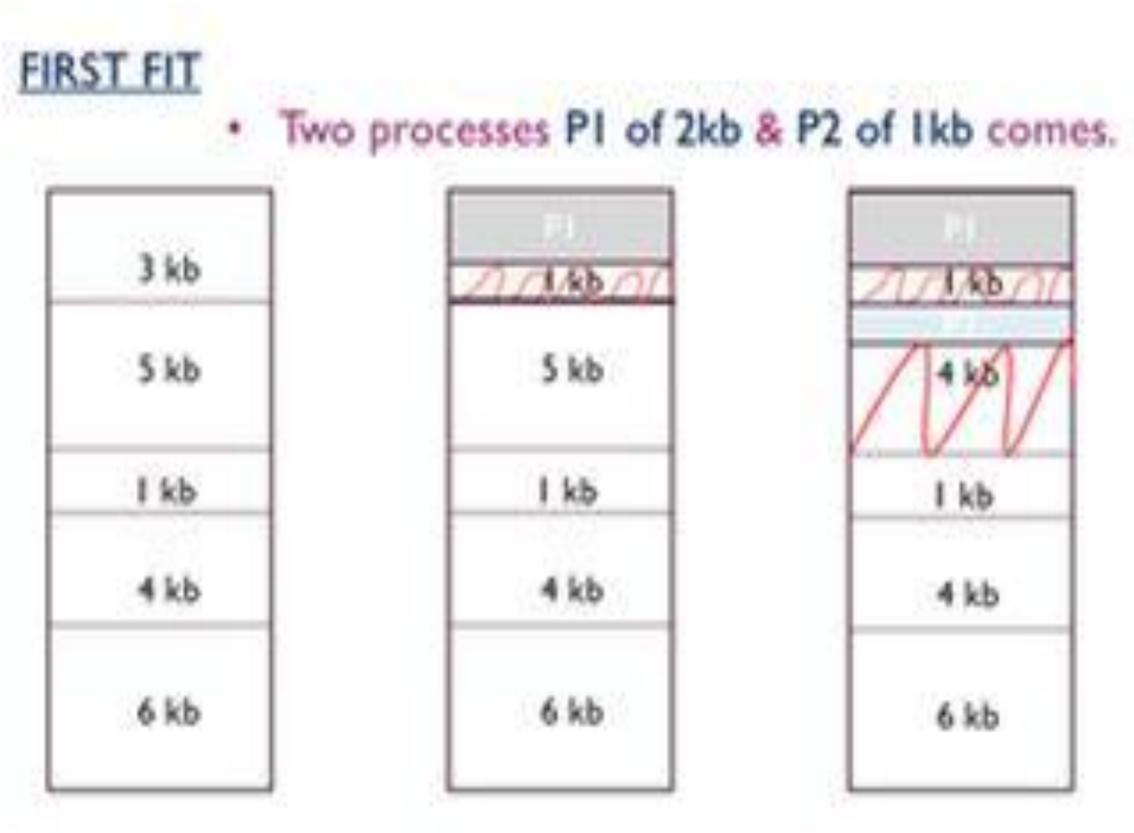
- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

- How to satisfy a request of size *n* from a list of free holes?
  - **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

First Fit


FIRST FIT
• Two processes P1 of 2kb & P2 of 1kb comes.
3 kb
5 kb
1 kb
4 kb
6 kb
1 kb
5 kb
1 kb
4 kb
6 kb
1 kb
4 kb
1 kb
4 kb
6 kb

Partition Selection Algorithms

Available blocks

A: 1500
B: 950
C: 750
D: 1100
E: 300
F: 350

Requests come in for blocks of the following sizes:

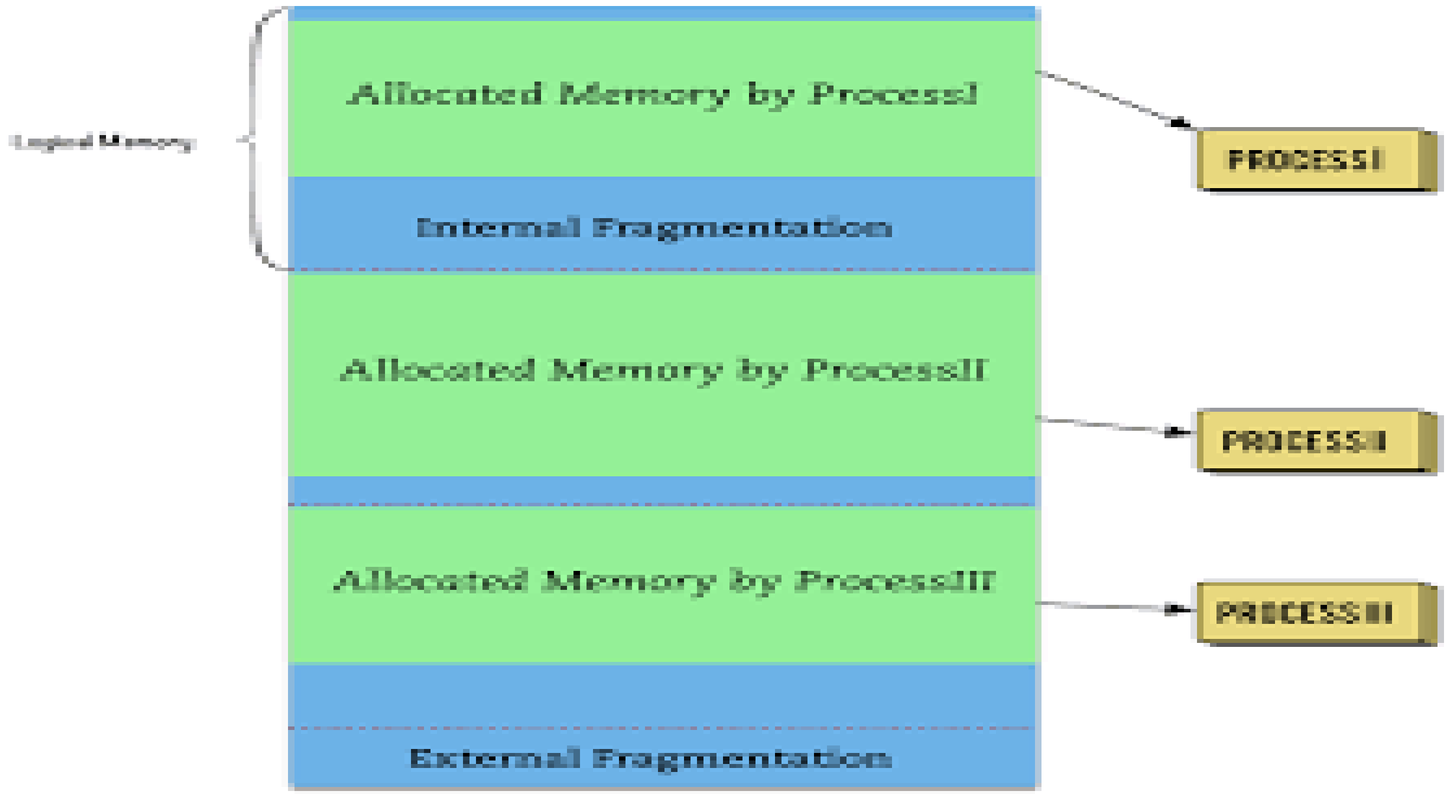
P1	P2	P3	P4	P5
900	25	780	1450	325

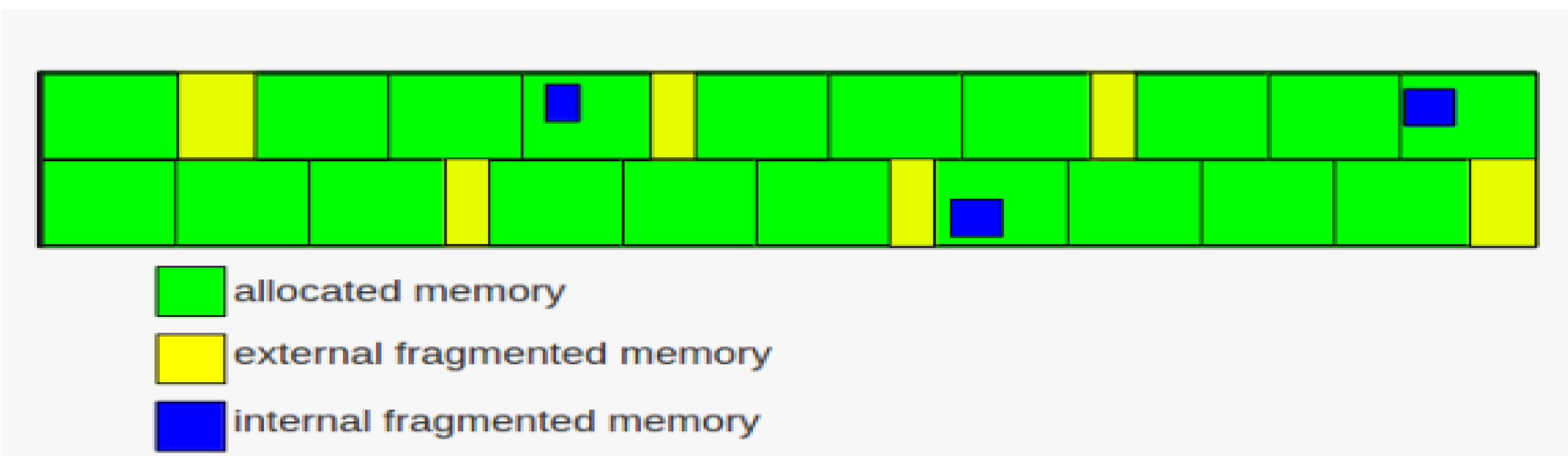
What block will be assigned to each request if the

- ***first-fit*** *algorithm is used?*
- ***best-fit*** *algorithm is used?*
- ***worst-fit*** *algorithm is used?*

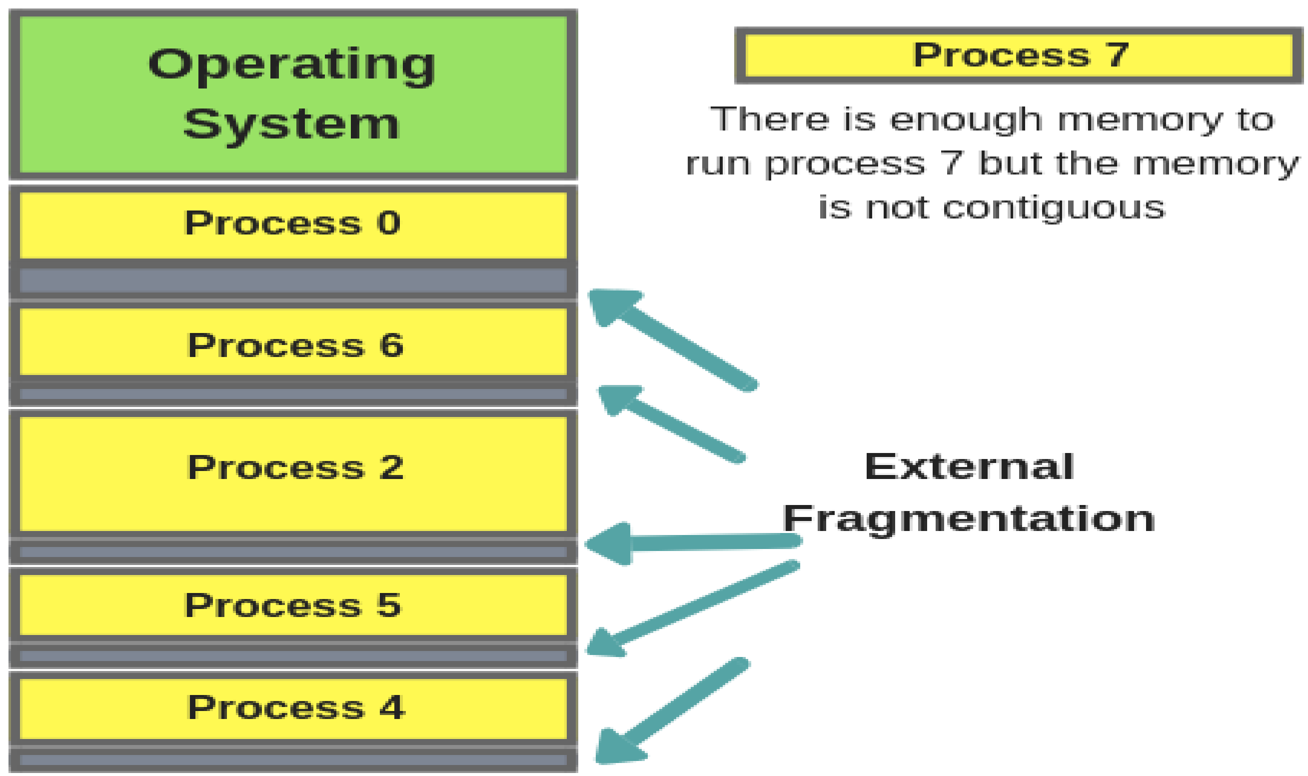
Fragmentation

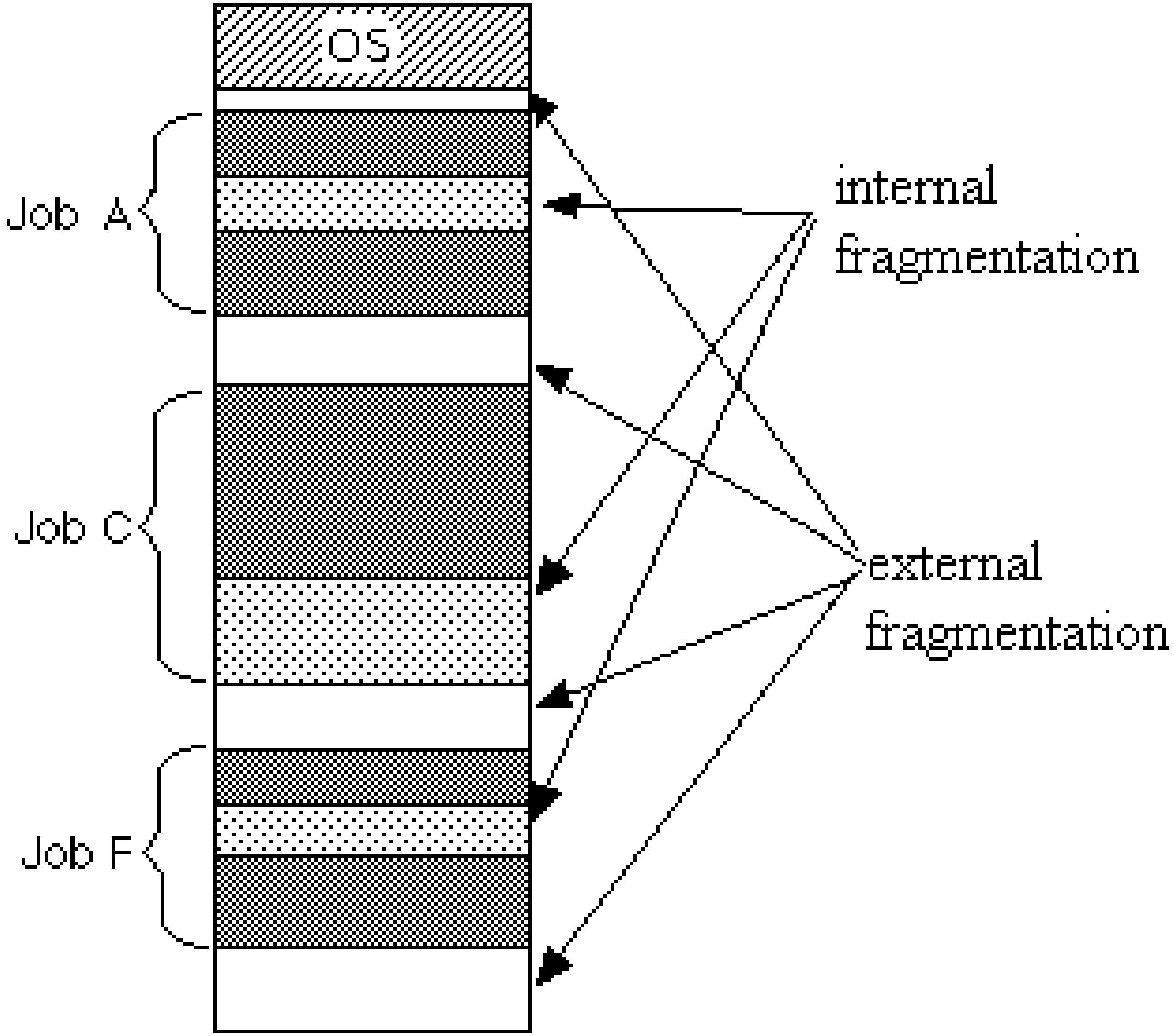
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used


Logical Memory
Allocated Memory by ProcessI
Internal Fragmentation
Allocated Memory by ProcessII
Allocated Memory by ProcessIII
External Fragmentation
PROCESS I
PROCESS II
PROCESS III

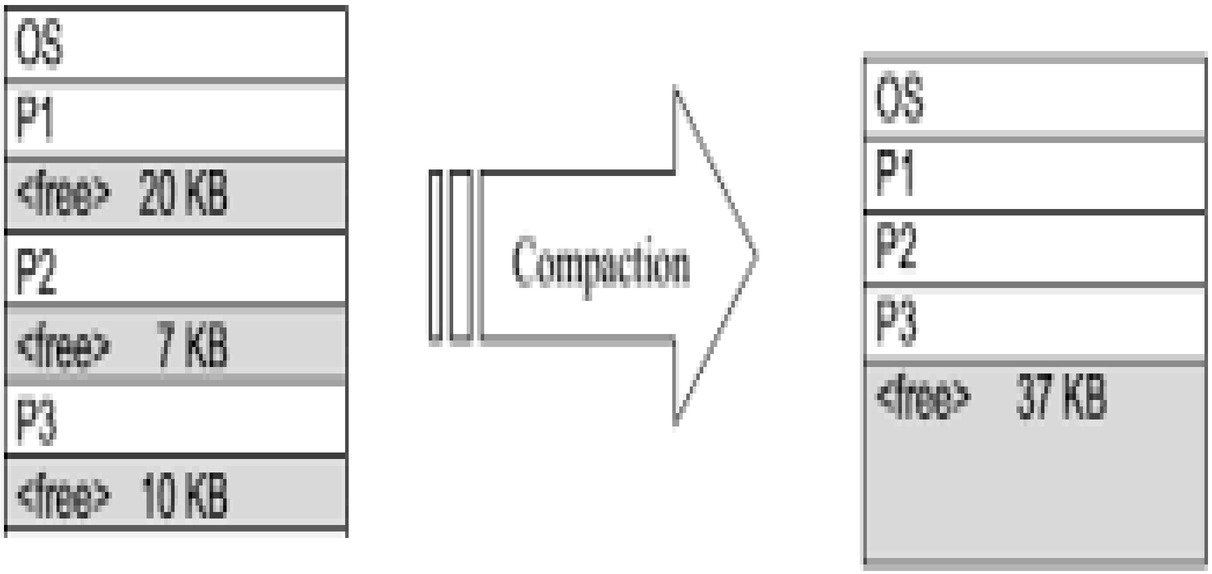

allocated memory
external fragmented memory
internal fragmented memory

External Fragmentation




OS
Job A
Job C
Job F
internal fragmentation
external fragmentation

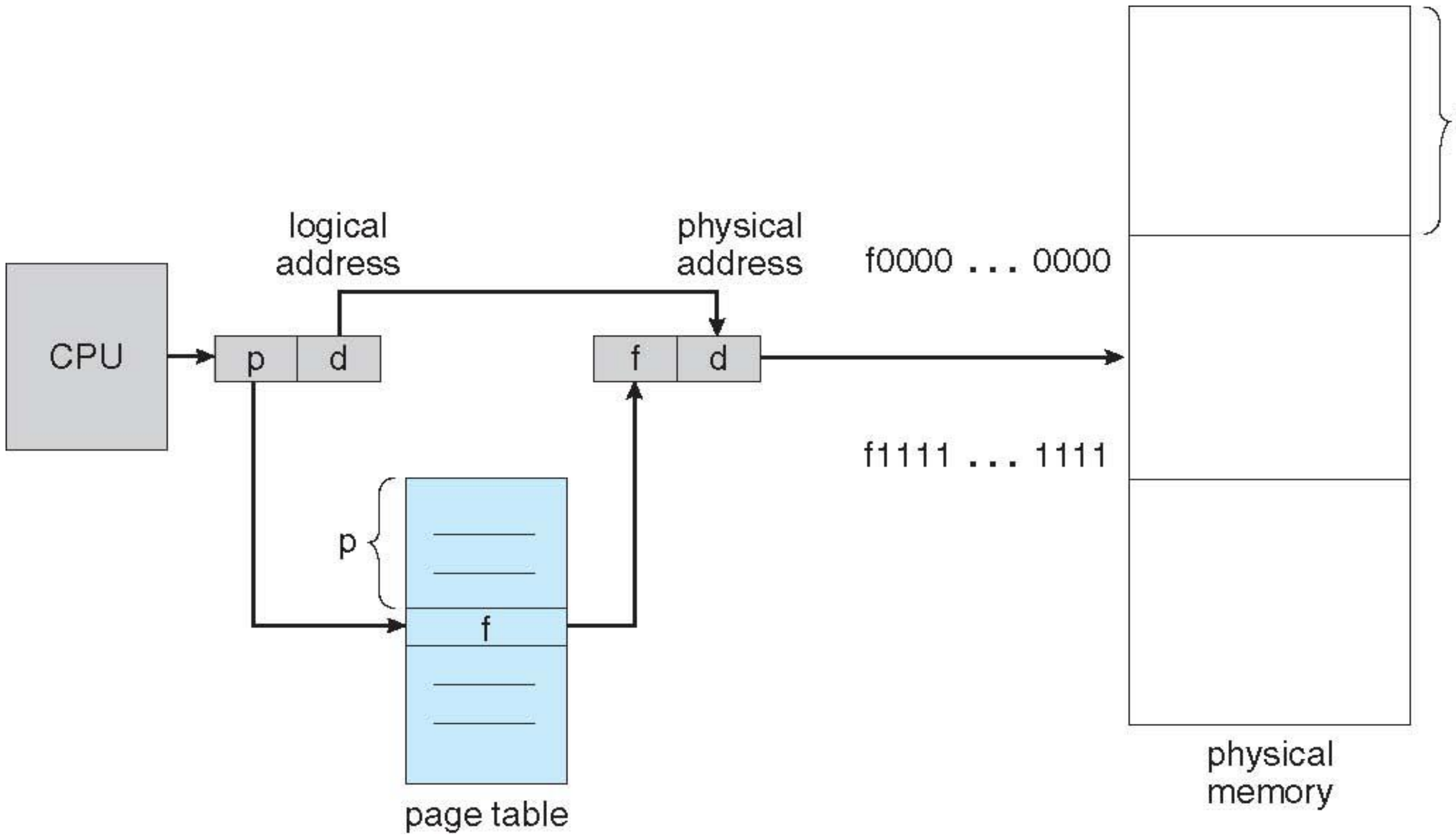
INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging and segmentation.


OS
P1
<free> 20 KB
P2
<free> 7 KB
P3
<free> 10 KB
Compaction
OS
P1
P2
P3
<free> 37 KB

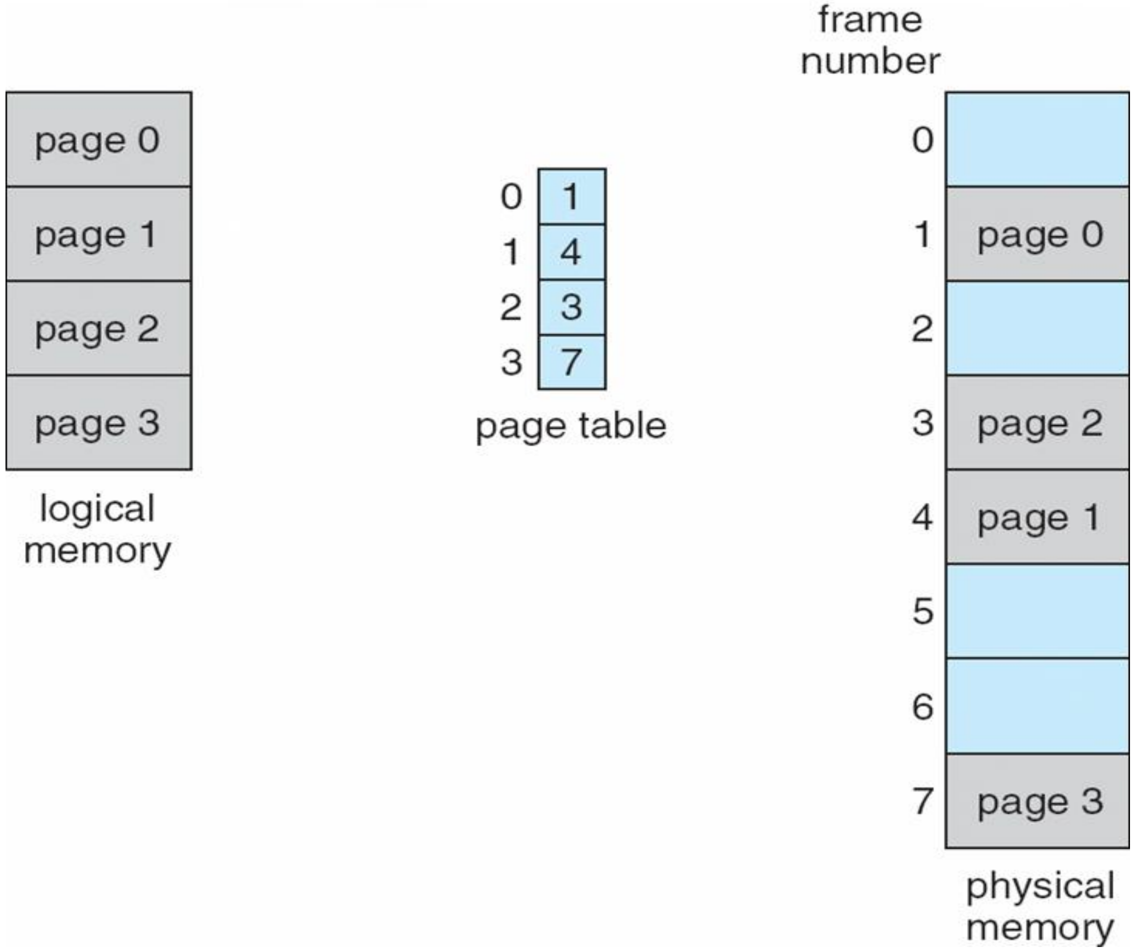
Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
  – Avoids external fragmentation
- Divide physical memory into fixed-sized blocks called **frames**
  – Size is power of 2, between 512 bytes and 16 Mbytes
- 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
- Backing store likewise split into pages
- **Still have Internal fragmentation**

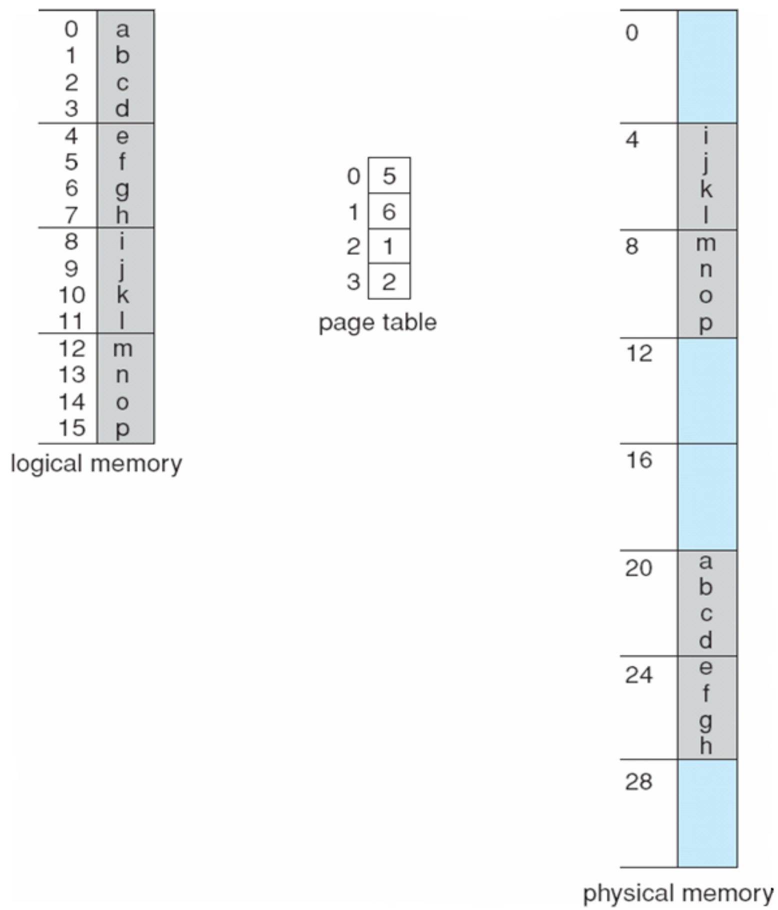
Paging Hardware



Paging Model of Logical and Physical Memory



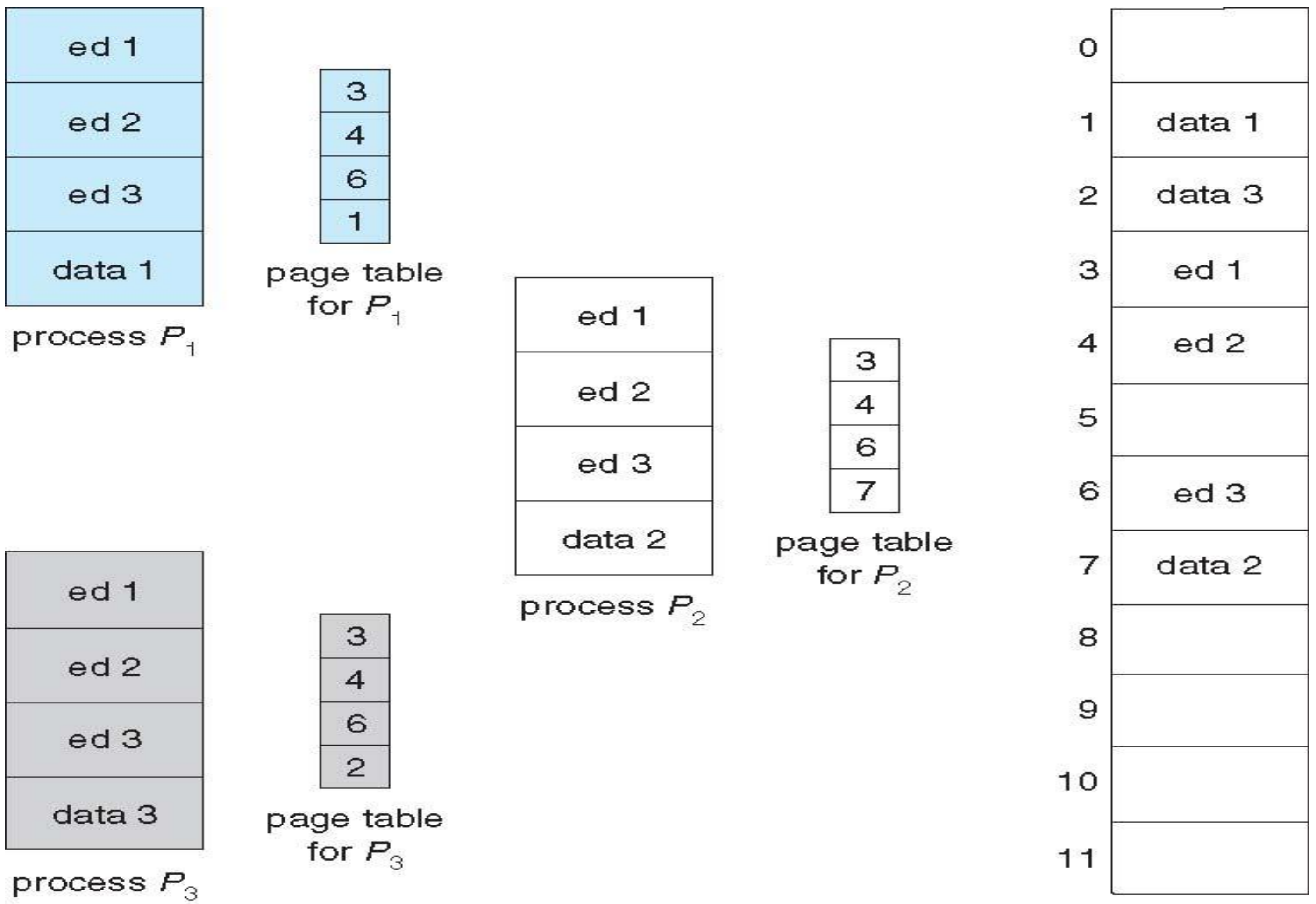
Paging Example



n=2 and m=4

32-byte memory and 4-byte pages

Shared Pages Example



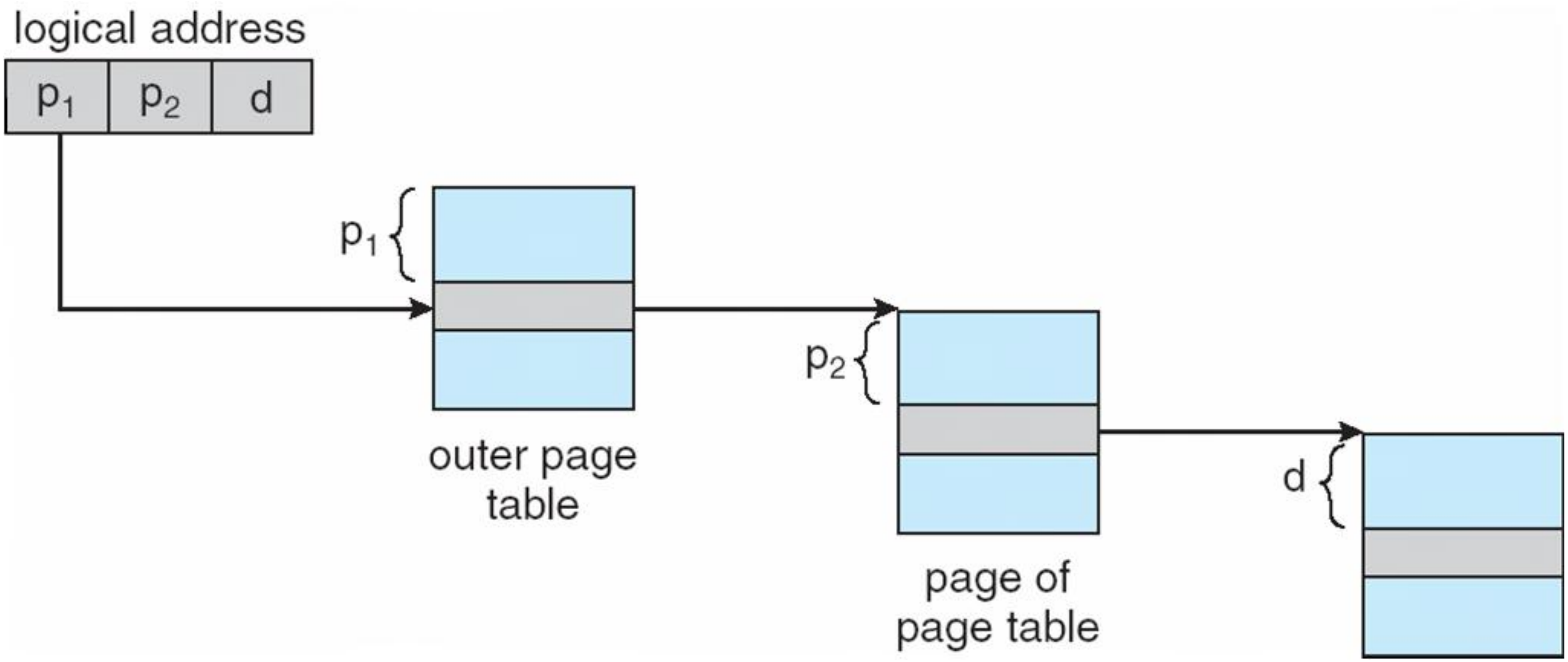
Types of Page Tables

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

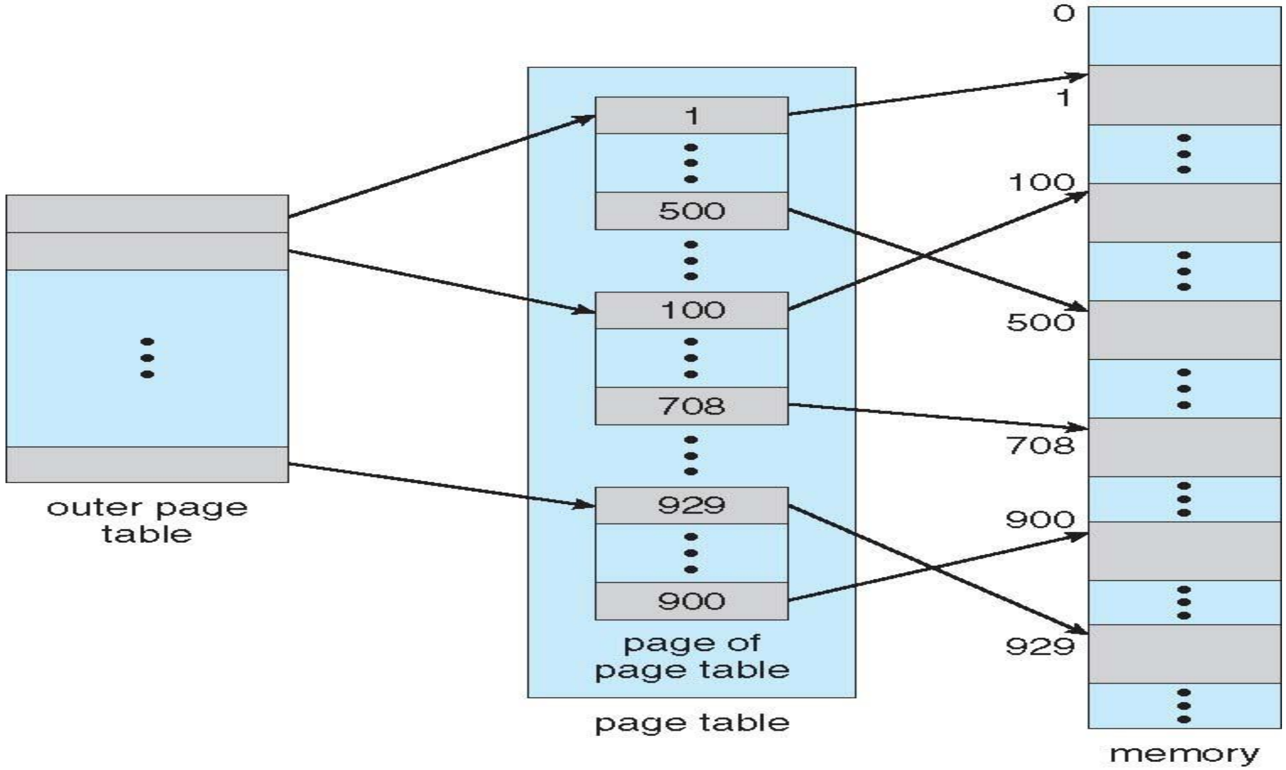
Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table
- We then page the page table

Two-Level Page-Table



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:

page number		page offset
p_1	p_2	d
12	10	10

- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

Three-level Paging Scheme

For 64bit logical address space, two level paging is not sufficient

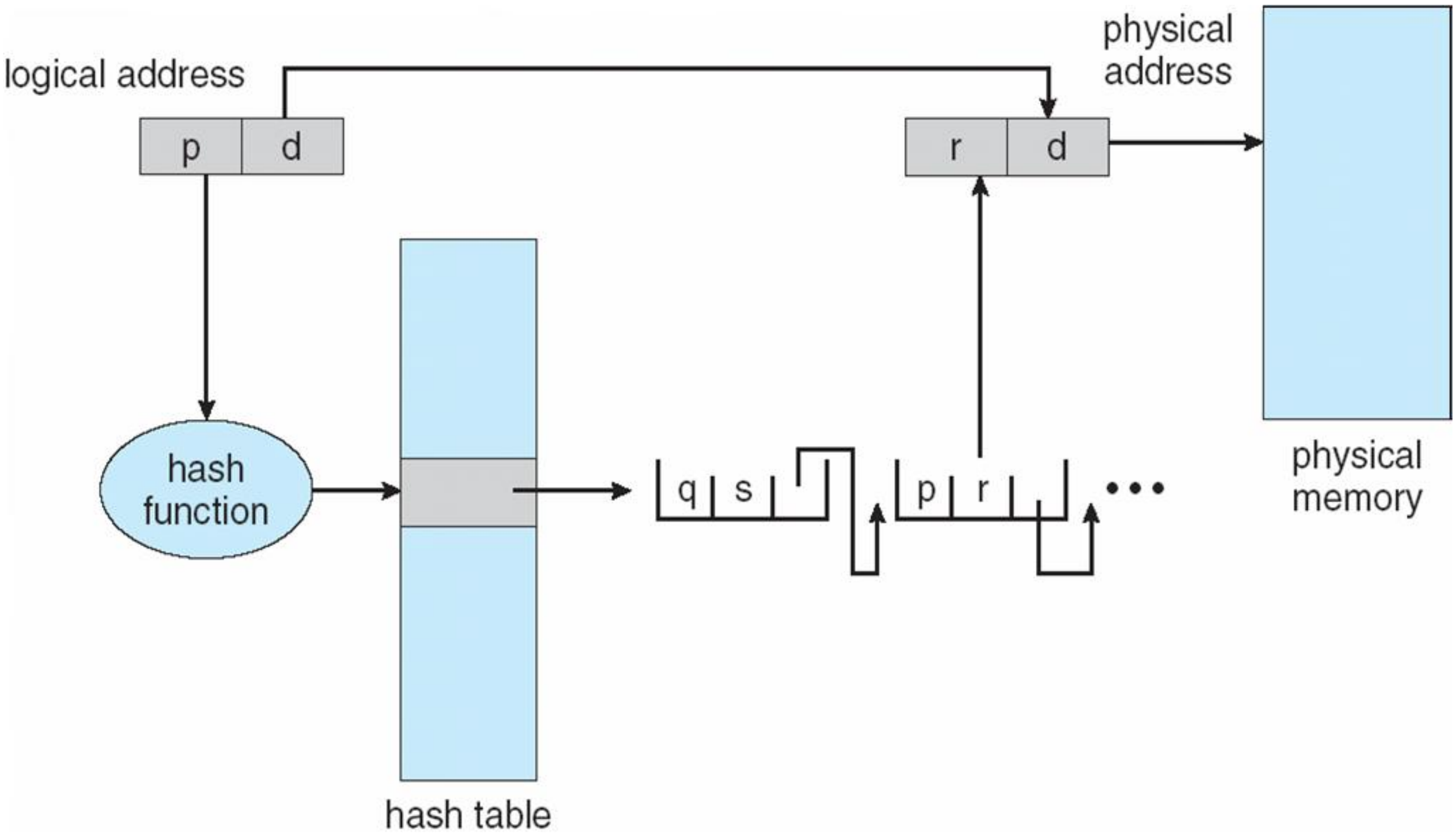
outer page	inner page	offset
p_1	p_2	d
42	10	12

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

Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
  - This page table contains a **chain of elements hashing to the same location**
- **Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element**
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted

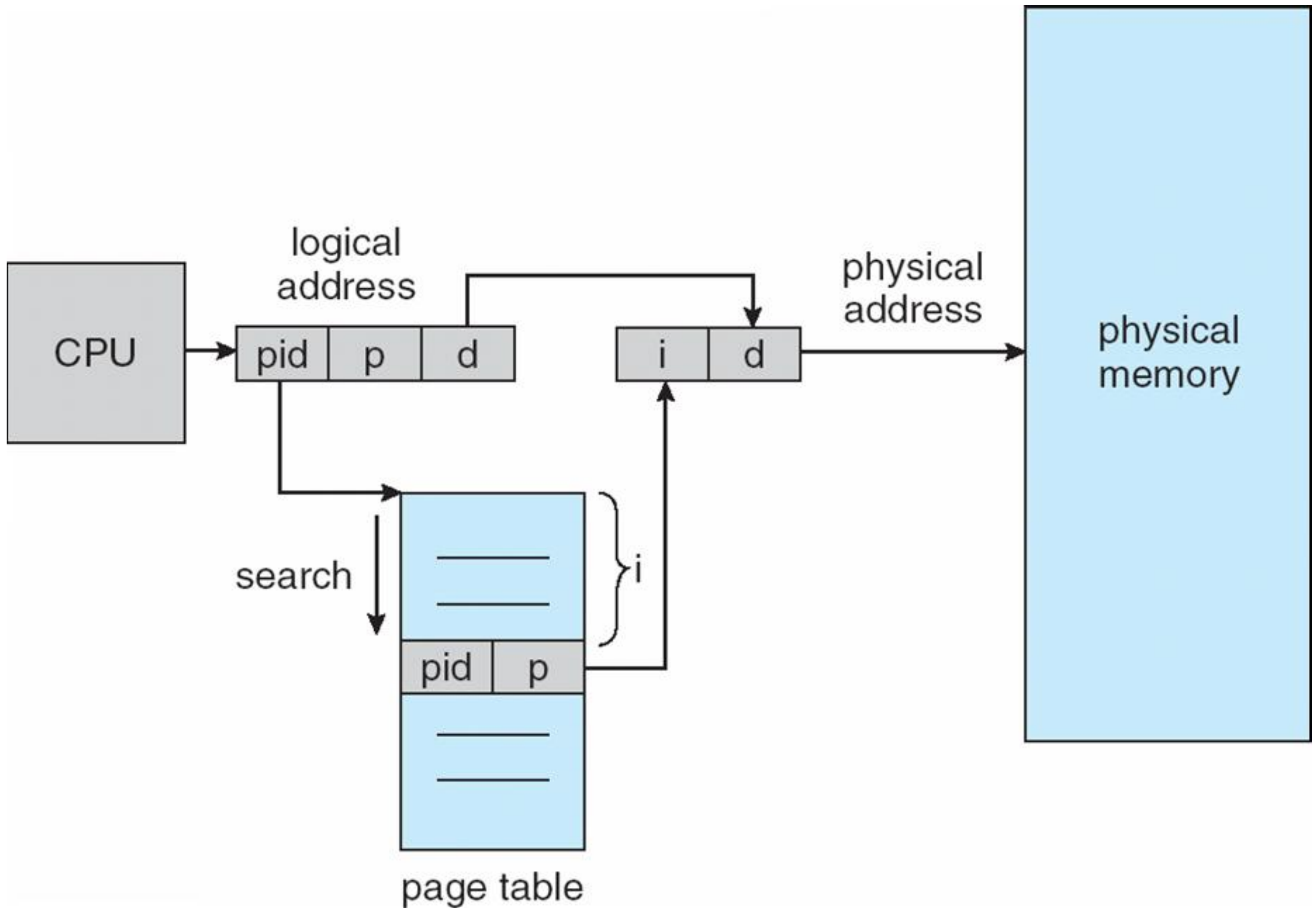
Hashed Page Table



Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page

Inverted Page Table Architecture

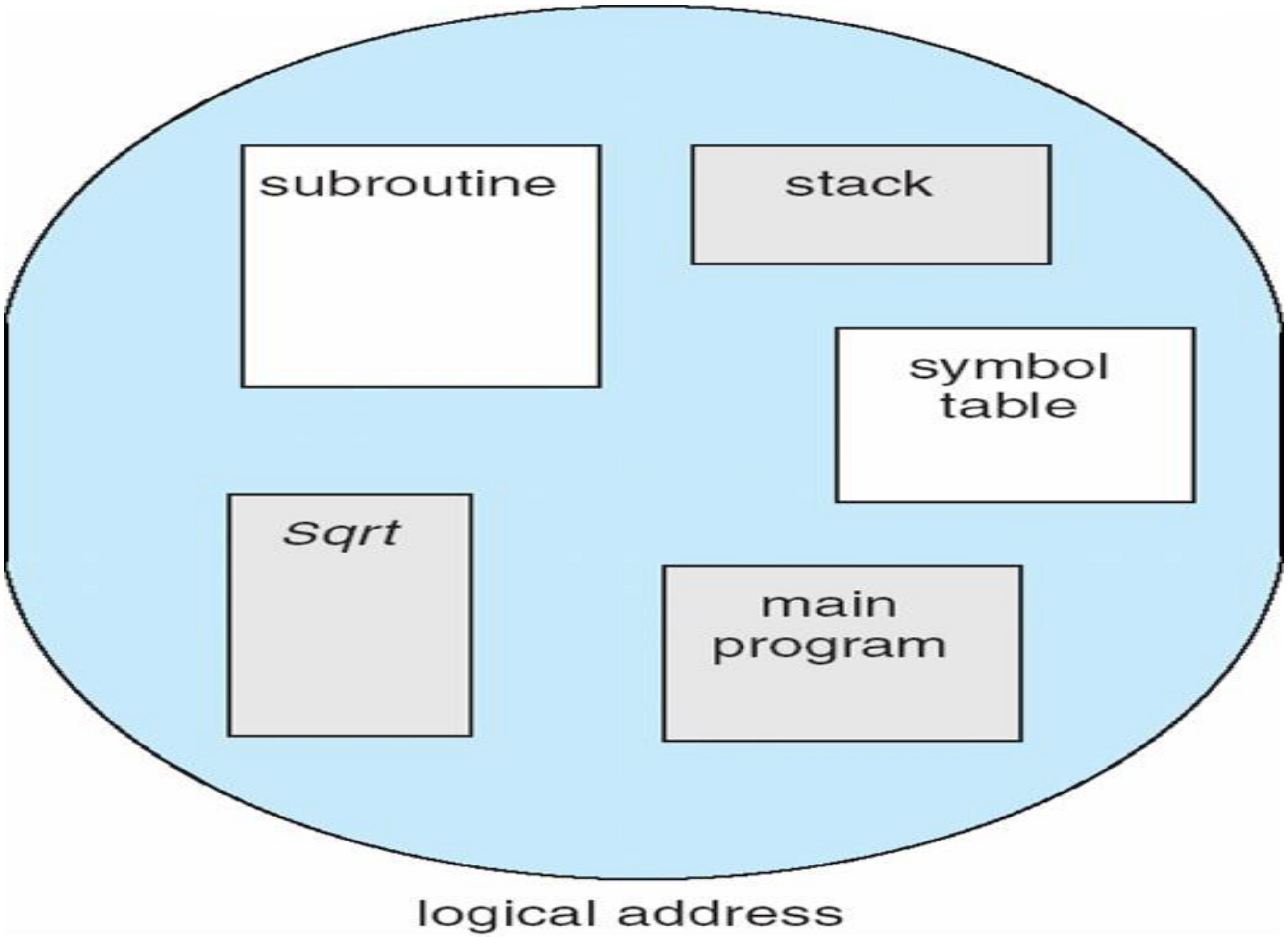


Segmentation

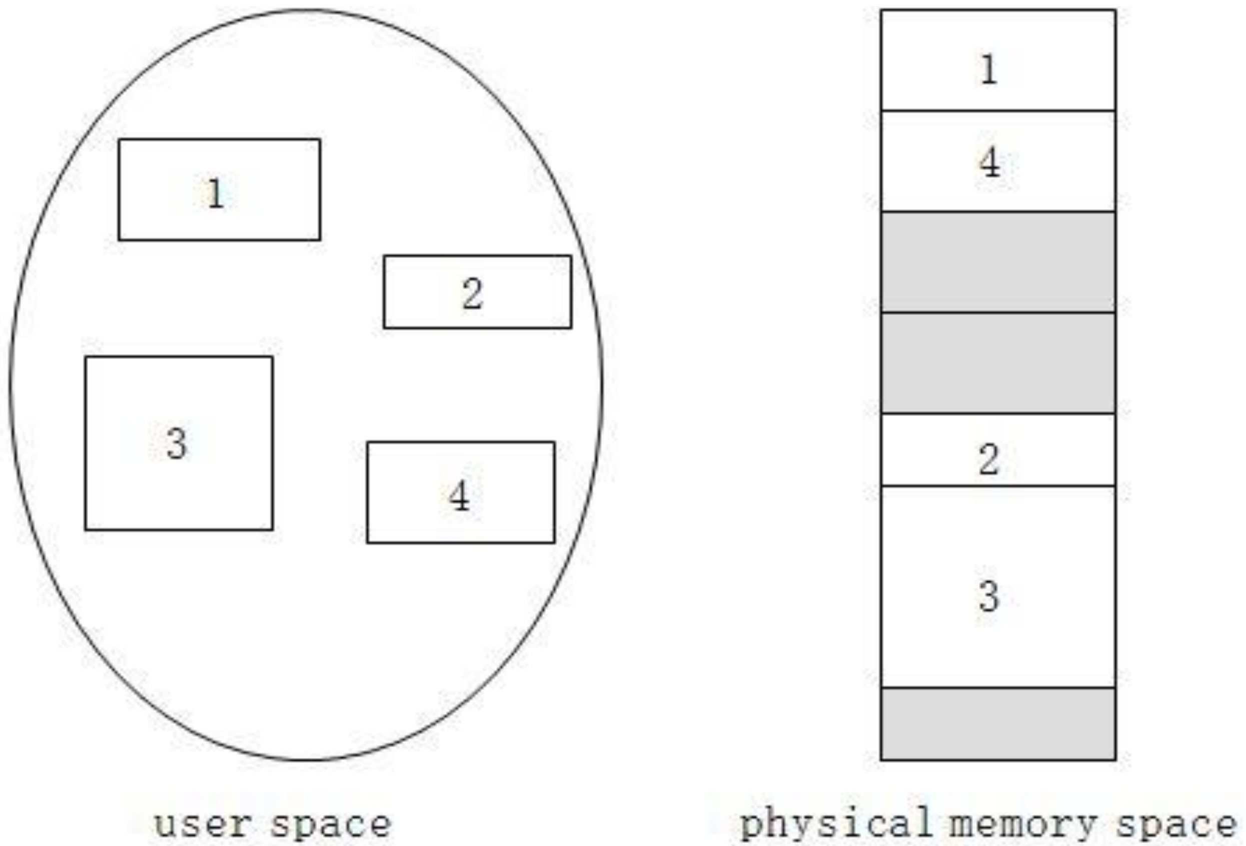
- Memory-management scheme that supports user view of memory
- No internal Fragmentation
- A program is a collection of segments (Variable sized Blocks)
  - A segment is a logical unit such as:

    main program
    procedure
    function
    method
    object
    local variables, global variables
    common block
    stack
    symbol table
    arrays

User’s View of a Program



Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

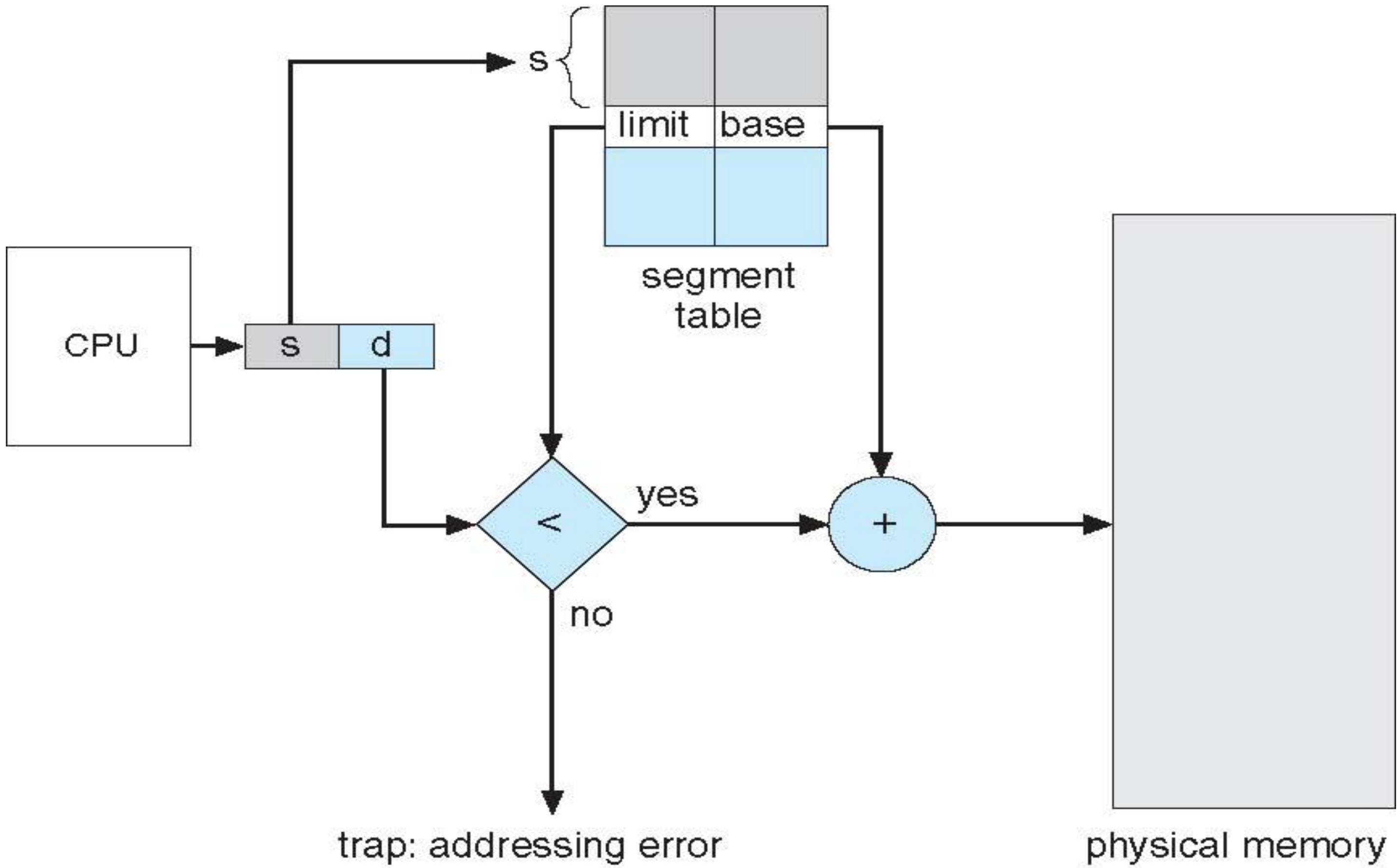
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

- **Segment-table base register (STBR)** points to the segment table's location in memory

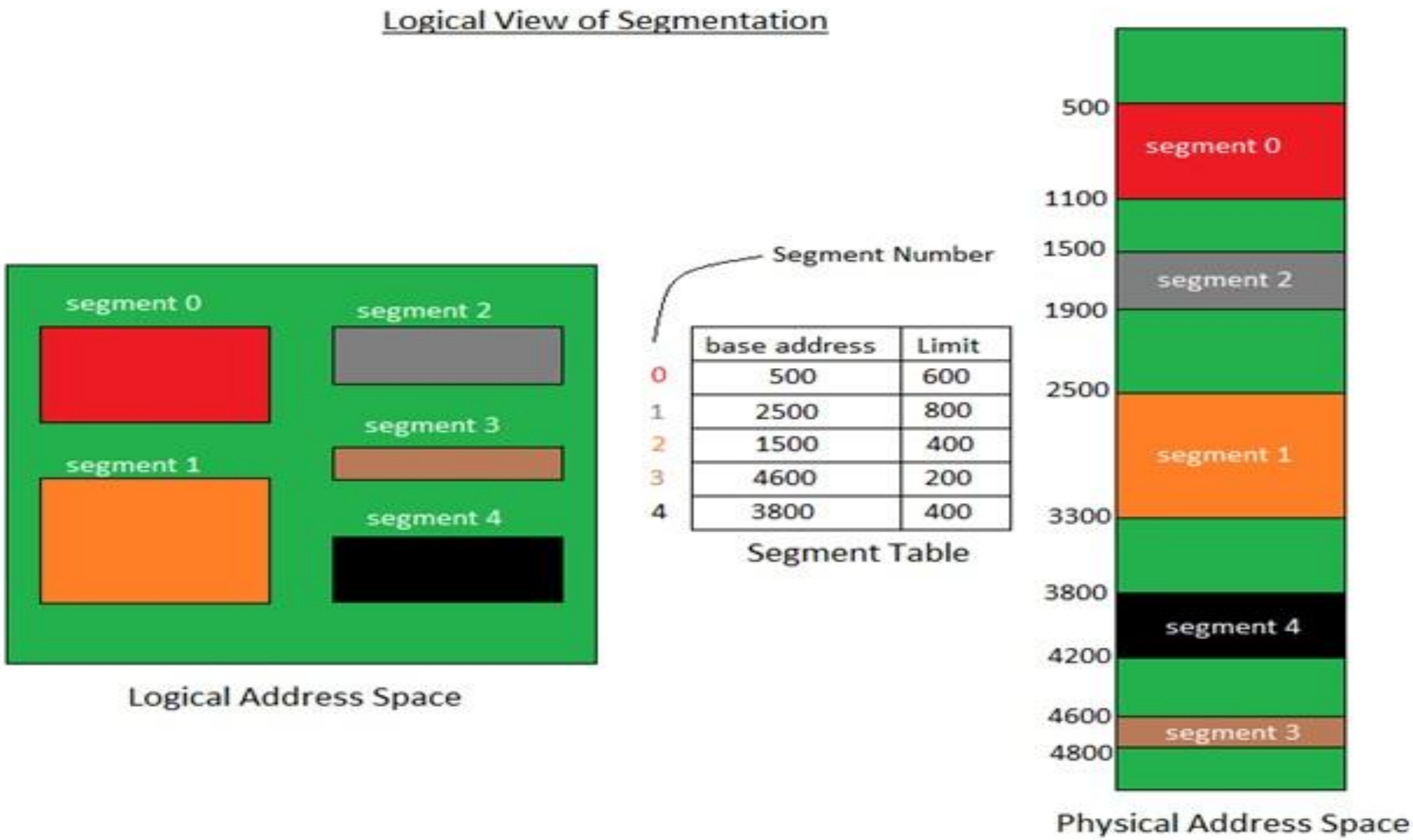
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Hardware



Segmentation



base address	Limit
500	600
2500	800
1500	400
4600	200
3800	400

Segment Table

References

- A. Silberschatz Galvin, Operation System Concepts, 3rd edition, Willey publications