Memory Management

Background

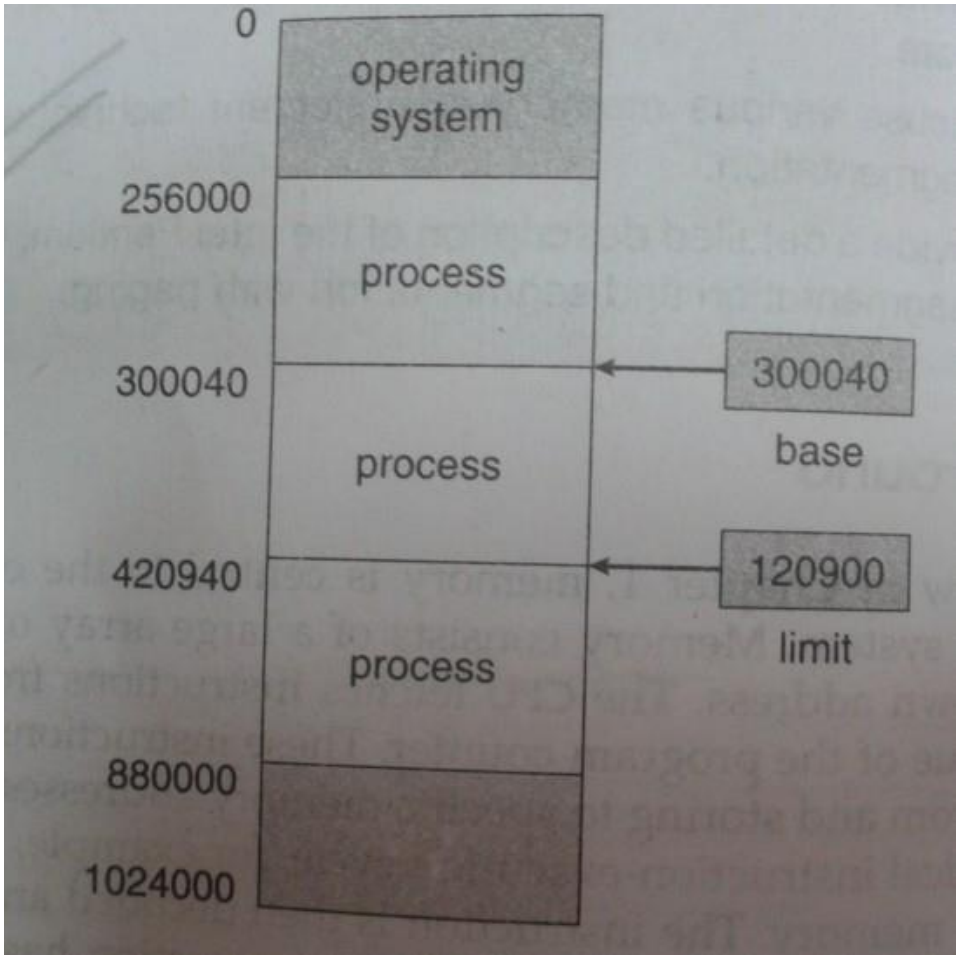
- Memory consists of large array of words or bytes, each with its own address.
- CPU fetches instructions from the memory as per the value of program counter.
- Memory unit only sees the addresses and is not concerned with how they are generated

Basic Hardware

- CPU can directly address – Main memory and registers
- Instructions only take main memory address and registers as operands
- So the data or any instructions must be in direct access device to be executed
- CPU can access registers with one clock cycle.
- But main memory access requires many clock cycles.
- Hence *cache* is used to increase the access rate

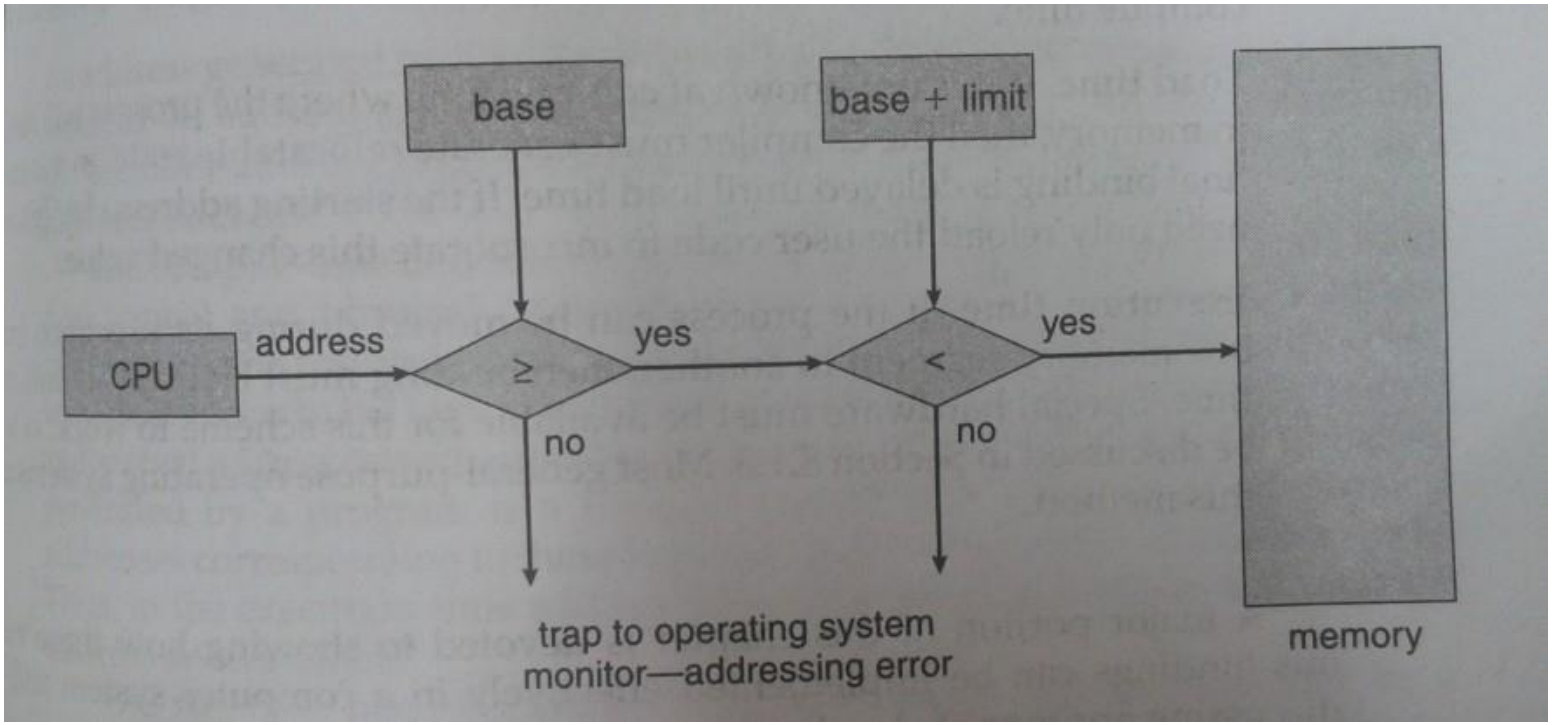
Address space of a process

- Base register – holds the smallest legal physical address
- Limit register – specifies the size of the range



Protection

- CPU hardware compares every address generated in user mode.



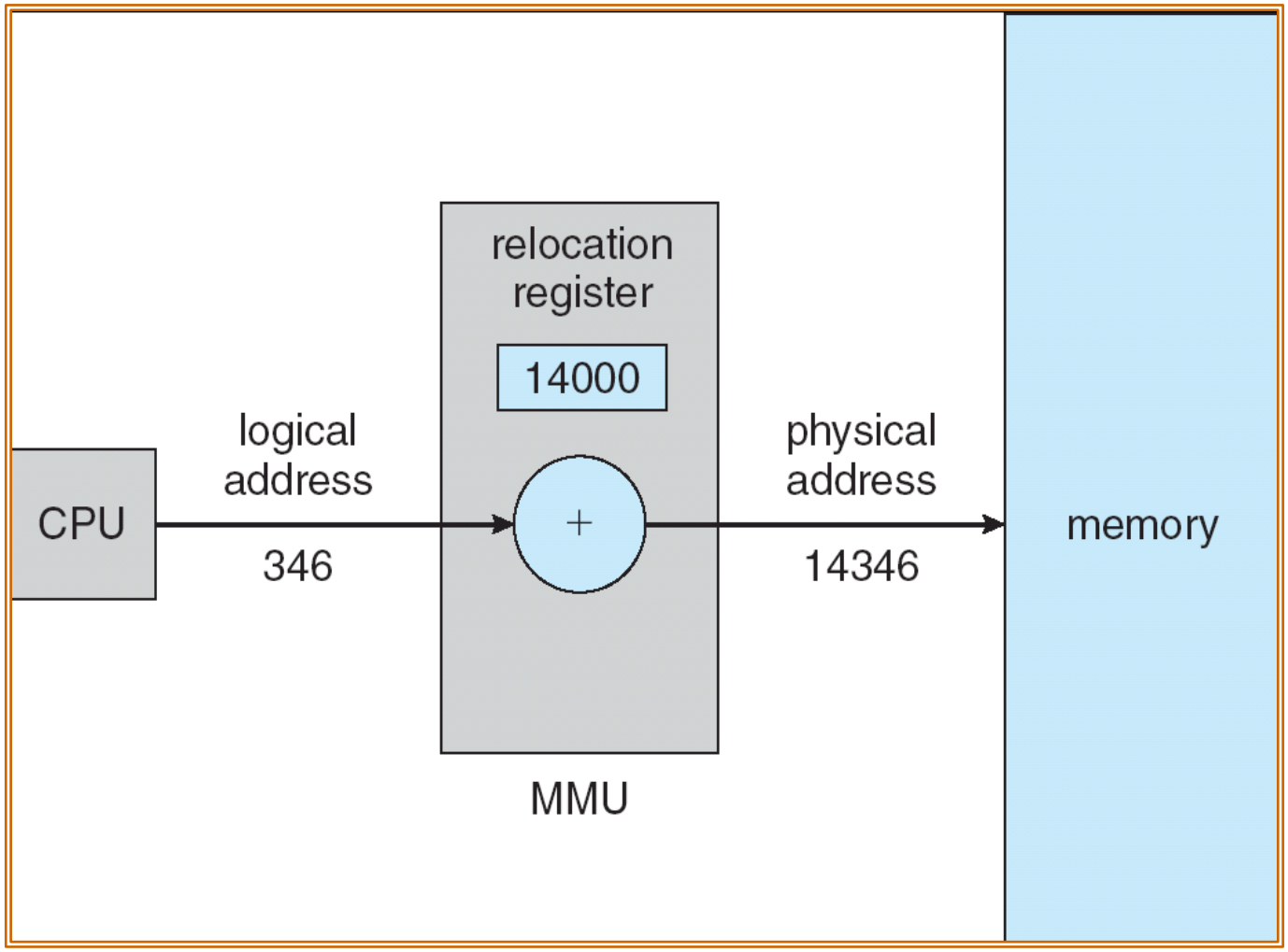
- Base and limit registers can only be updated by OS in kernel mode

Address Binding

- **Compile time**: If memory location known at compile time, *absolute code* can be generated; must recompile code if starting location changes
- **Load time**: Must generate *relocatable code* if memory location is not known at compile time. In this case final binding is delayed until load time.
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps.

Logical vs physical address

- Logical address – generated by the CPU; also referred to as virtual address
- Physical address – address seen by the memory management unit
- Compile time and load time address binding generate identical logical and physical addresses.
- Execution time generates different logical and physical addresses.
- Memory management unit(MMU) is used to map logical address to physical address.



Dynamic Loading

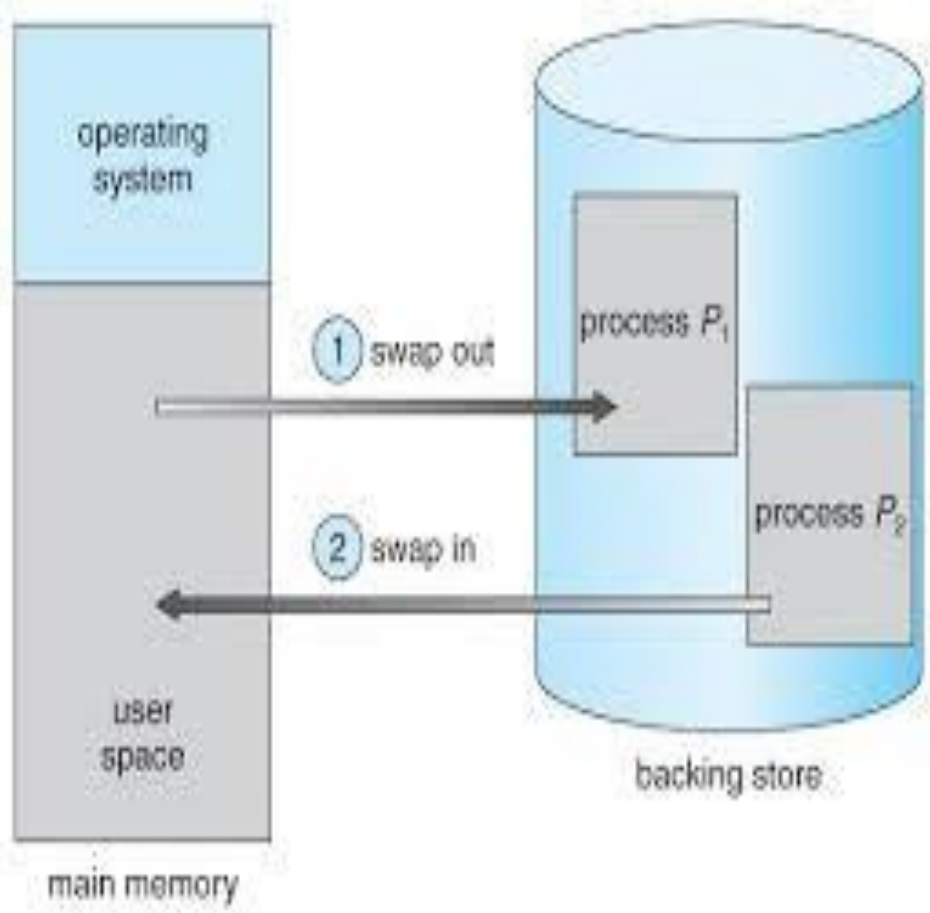
- Entire program and its entire data should be in memory for process to execute.
- Therefore process size <= memory size

Dynamic loading

- Load a routine only when it is called

Swapping

- Used in priority based scheduling algorithms.
- To make space for higher priority process.
- Where to swap in the process?
  - Same space
  - Different

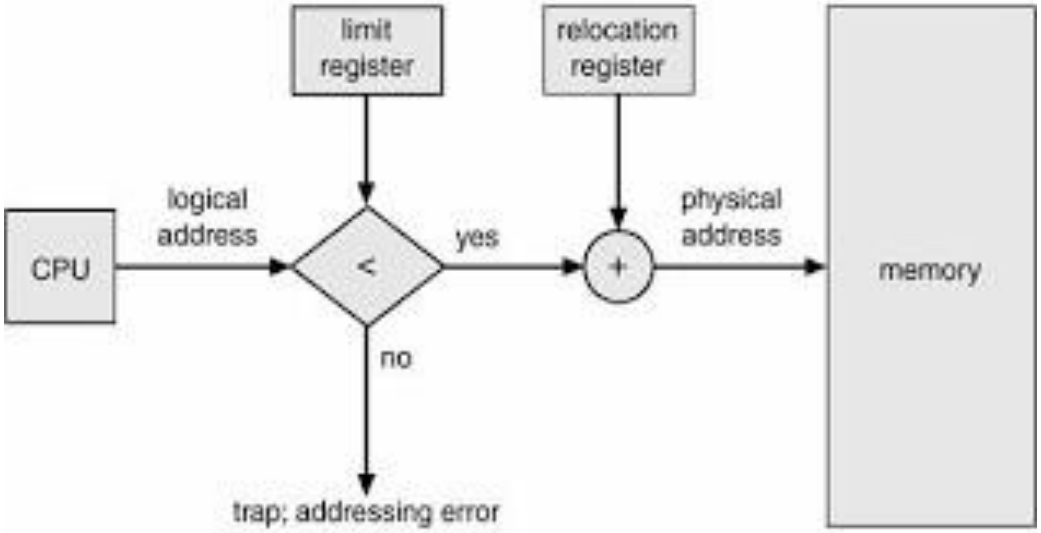


Memory Allocation Schemes

- Contiguous memory allocation
- Paging
- Segmentation

Contiguous Memory allocation

- Each process is contained in **a single contiguous section** of memory.
- Memory mapping and protection
  - Relocation register – contains value of smallest physical address
  - Limit register – contains the range of logical addresses



Problem

- Suppose that the value of relocation register is 100040 and limit register is 74600. What are the physical addresses for the following logical addresses?
  1. 70000
  2. 75000

- Ans:

1.Check whether logical address < limit register value.

70000<74600. TRUE.

Physical address = logical address + relocation register value

i.e. Physical address = 70000+100040=170040

2.Check whether logical address < limit register value.

75000<74600. FALSE

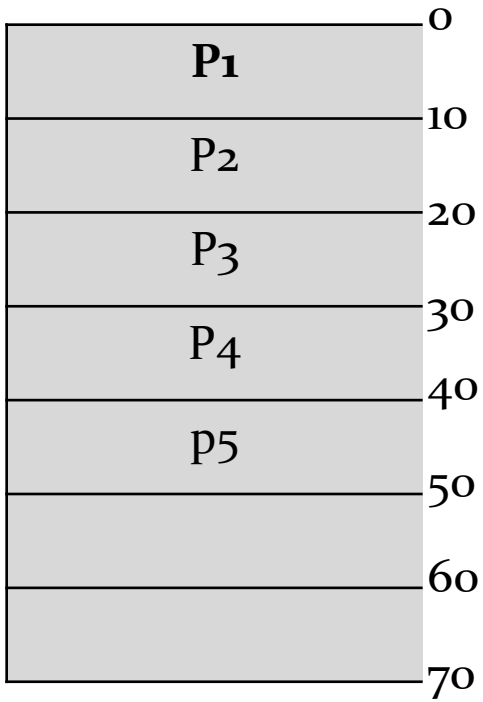
Trap: addressing error

Contiguous Memory allocation

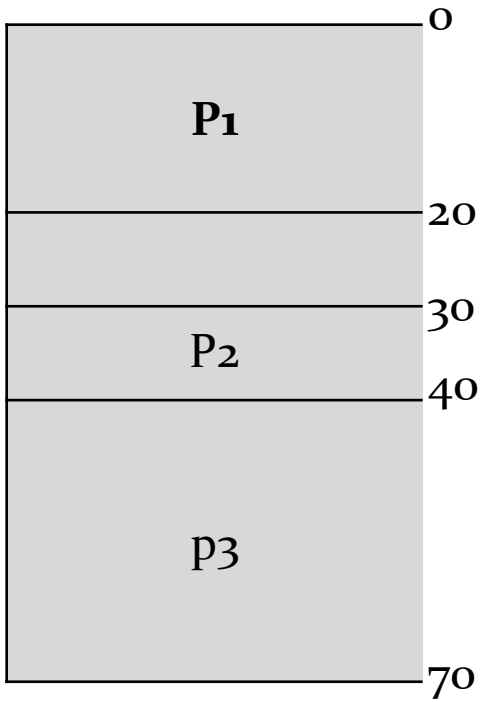
1. Fixed size partition
   - Each partition may contain exactly one process.
   - Degree of multiprogramming = number of partitions
   - Used in IBM OS/360
   - Not used now

2. Variable partition scheme
   - Partitions of variable size
   - Initially all the memory is free.
   - OS maintains a table indicating which part of memory is free.

Fixed partition

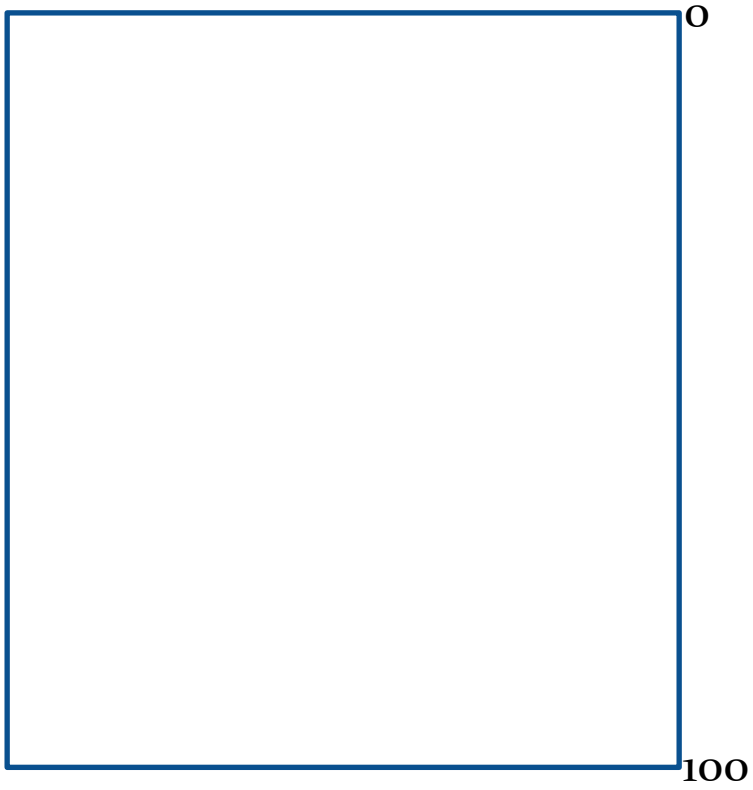


Variable partition



Variable partition working

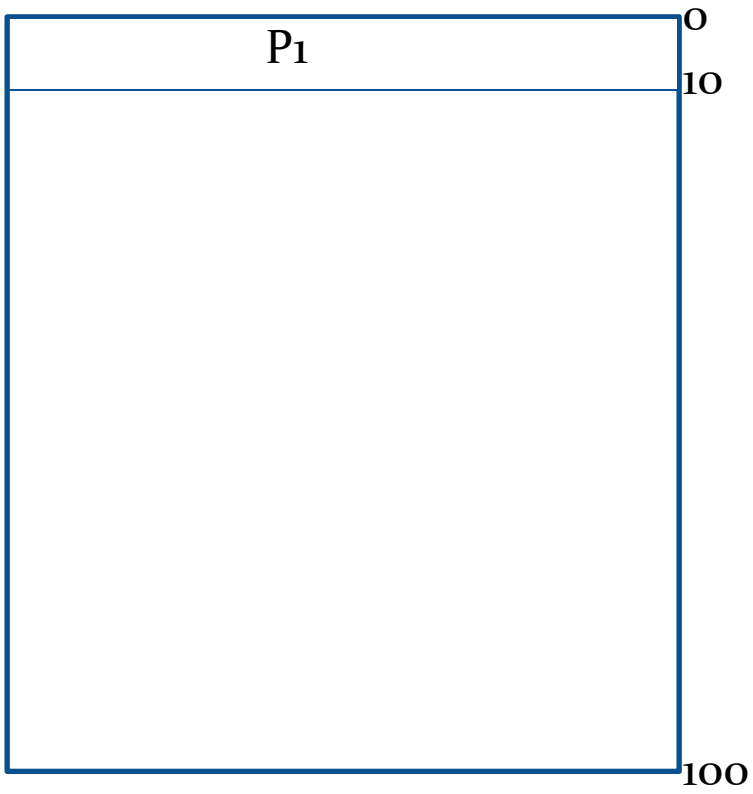
- Suppose total memory is 100Mb



Memory	Status
0-100	Free

Variable partition working

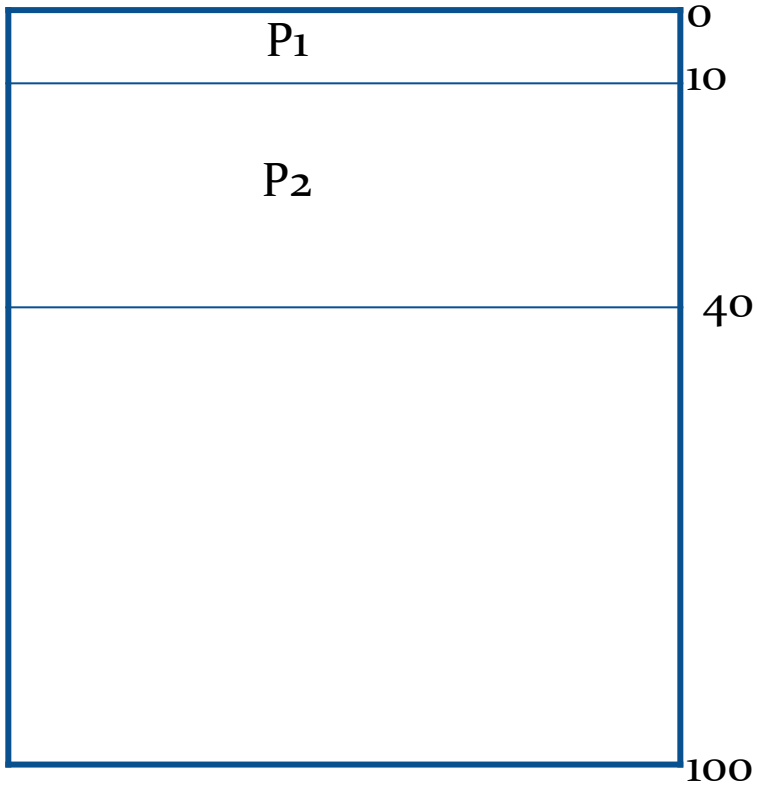
- Process P1 requires 10Mb space



Memory	Status
0-9	P1
10-100	free

Variable partition working

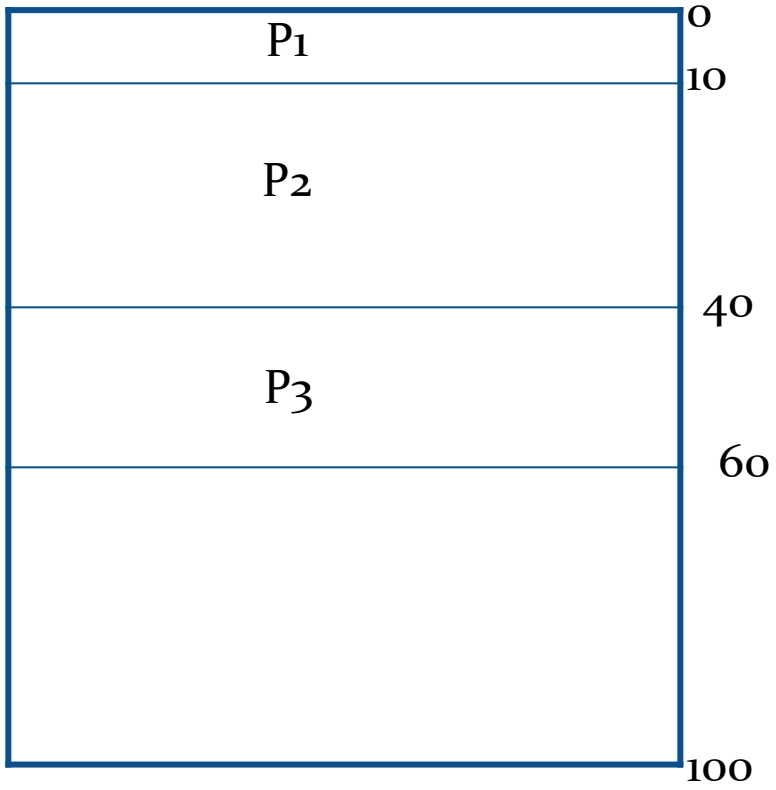
- Process P2 requires 30Mb space



Memory	Status
0-9	P1
10-100	free

Variable partition working

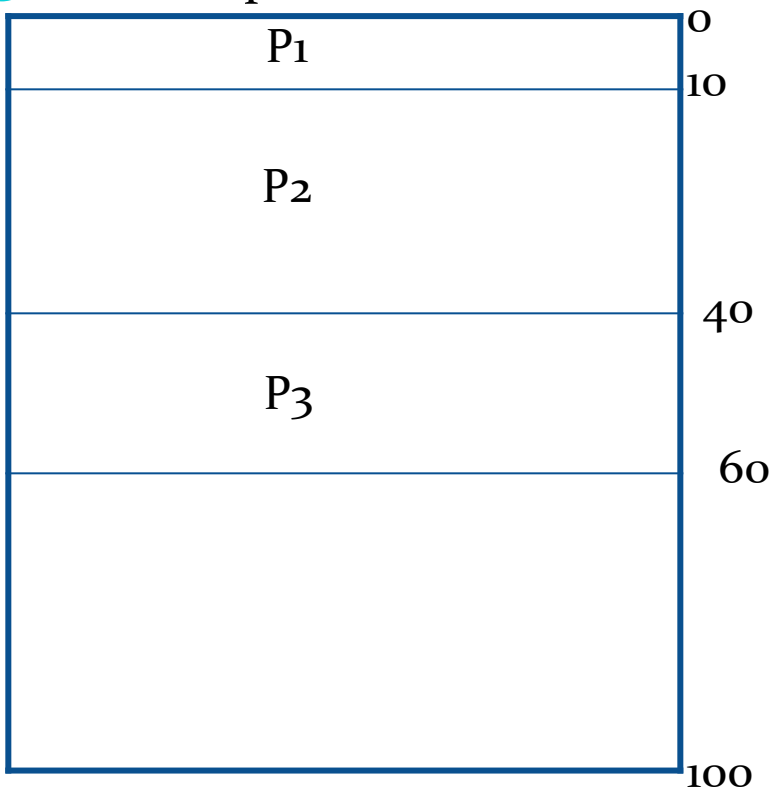
- Process P3 requires 20Mb space



Memory	Status
0-9	P1
10-39	P2
40-59	P3
60-100	Free

Variable partition working

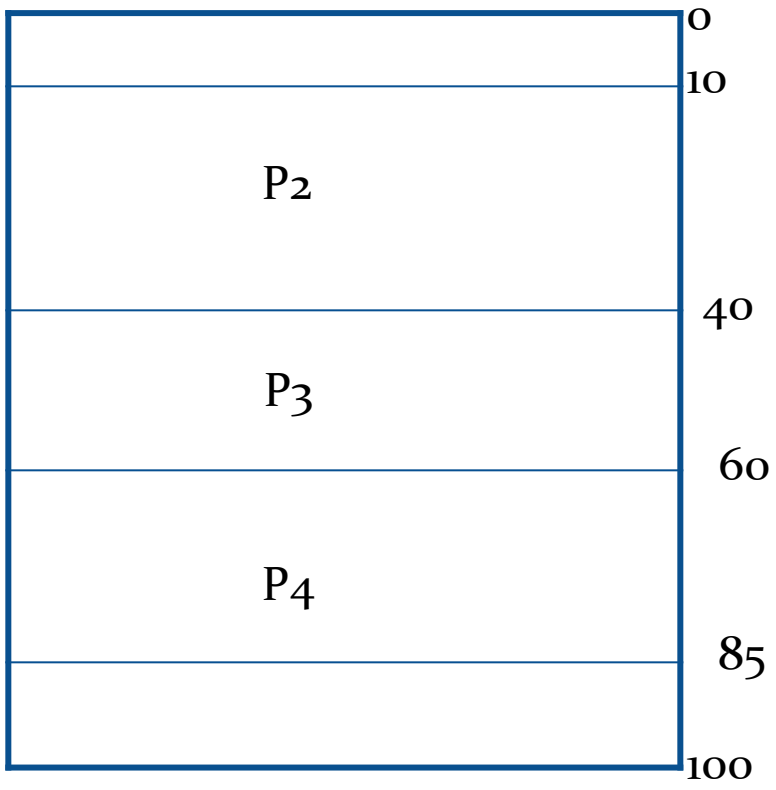
- Process P1 finishes its execution.
- So the space is freed



Memory	Status
0-9	Free
10-39	P2
40-59	P3
60-100	Free

Variable partition working

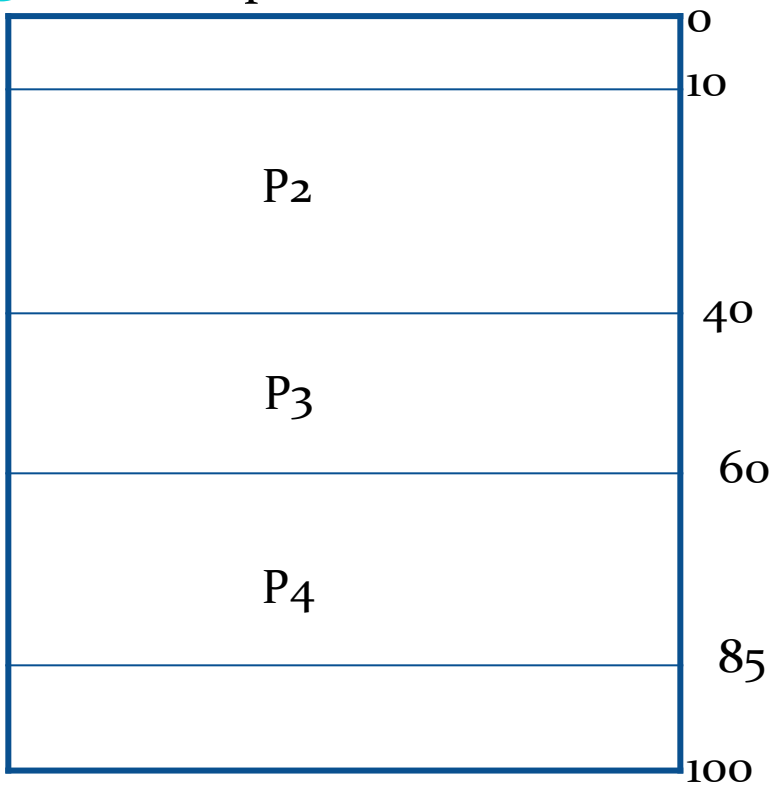
- Process P4 requires 25 Mb space



Memory	Status
0-9	Free
10-39	P2
40-59	P3
60-84	P4
85-100	Free

Variable partition working

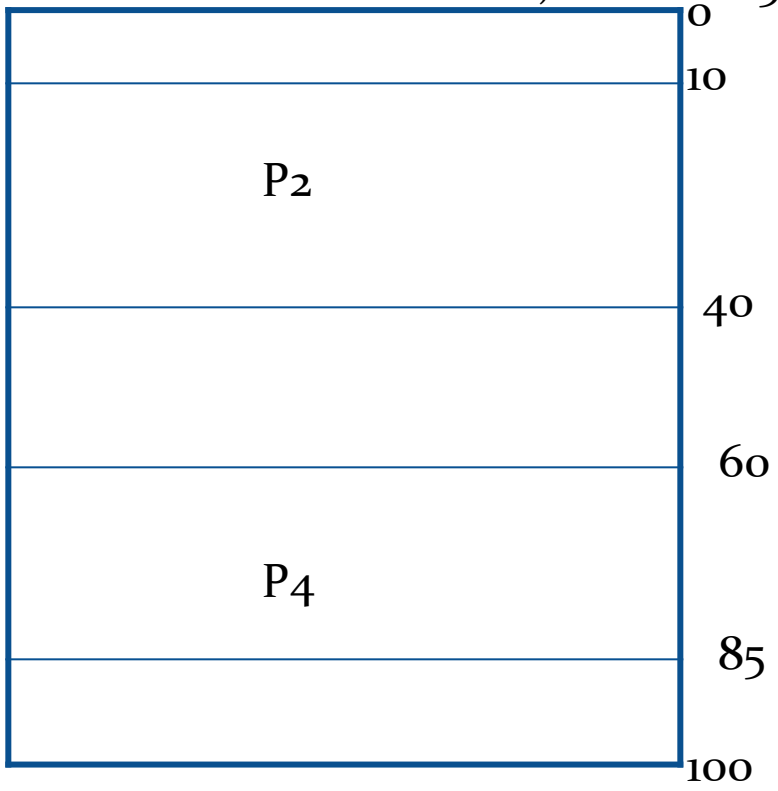
- Process P3 finishes its execution.
- So the space is freed



Memory	Status
0-9	Free
10-39	P2
40-59	Free
60-84	P4
85-100	Free

Variable partition working

- Process P5 requires 25 Mb space
- Can it be granted? No, because 25Mb is not free in a single slot(hole)



This problem is called "External Fragmentation"

Memory	Sta
0-9	Free
10-39	P2
40-59	Free
60-84	P4
85-100	Free

10 Mb free

20 Mb free

16 Mb free

Total free space = 10+20+16=46Mb

External Fragmentation

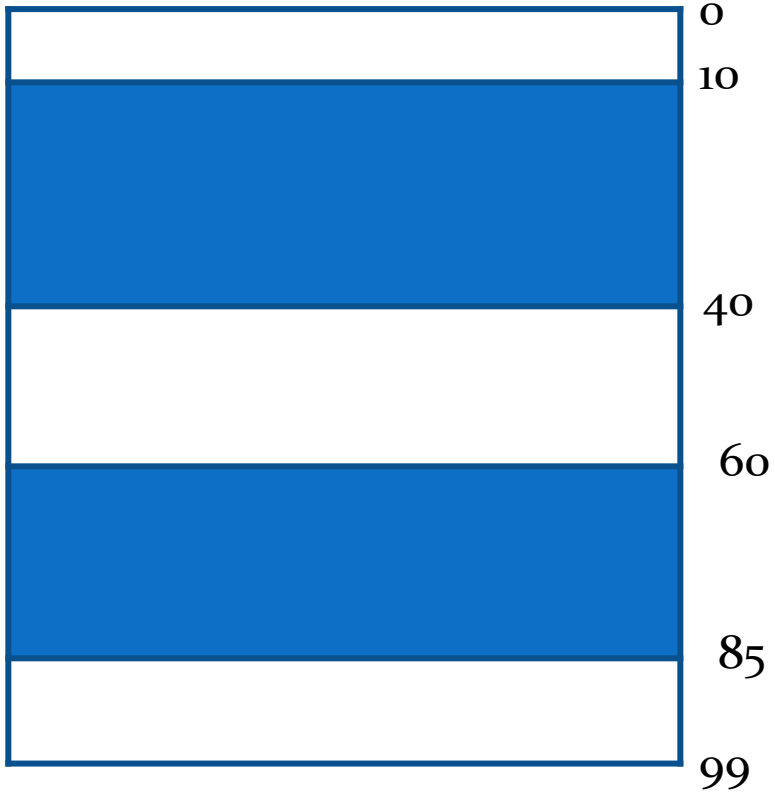
- When there is enough total memory space to satisfy a request, but the available space is not contiguous.
- Solution
  - Compaction – shuffle the memory contents so as to place all free memory together in one large block.

Block(hole) allocation

- First fit – allocate the first hole that is big enough
- Best fit – allocate the smallest hole that is big enough
- Worst fit – allocate the largest hole

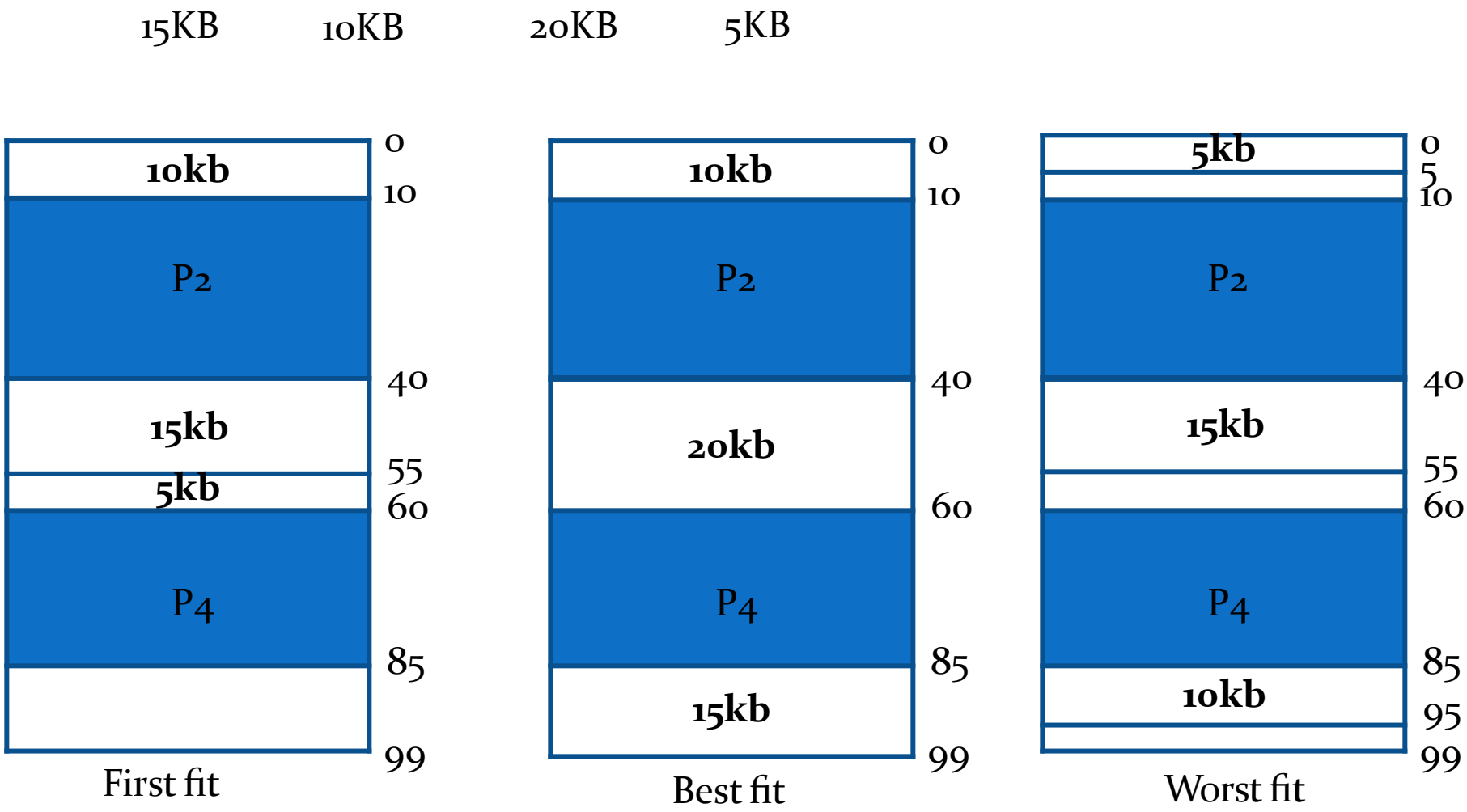
Block(hole) allocation- Example

- Given three free memory partitions of 10 KB, 20 KB and 15 KB(in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 15 KB, 10 KB, 20 KB and 5 KB (in order)?



Block(hole) allocation- Example

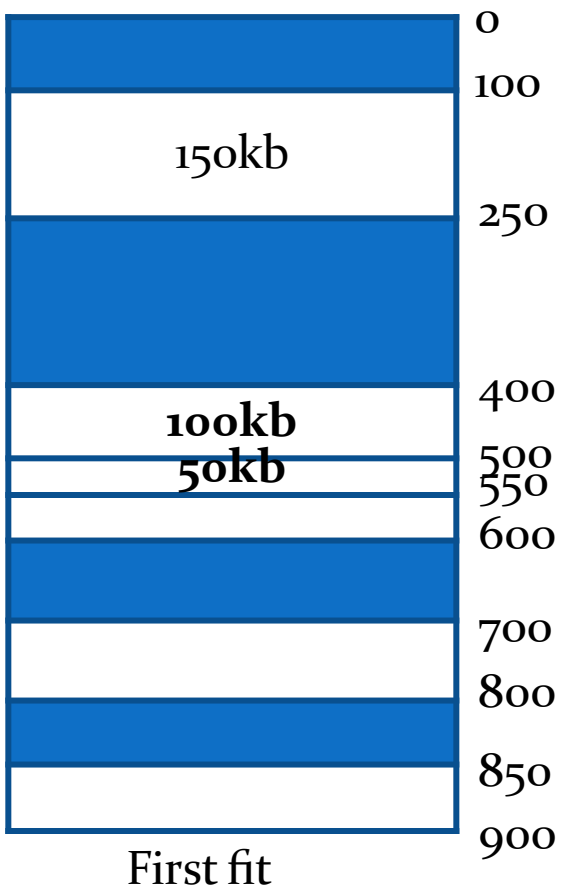
- 15 KB, 10 KB, 20 KB and 5 KB (in order)



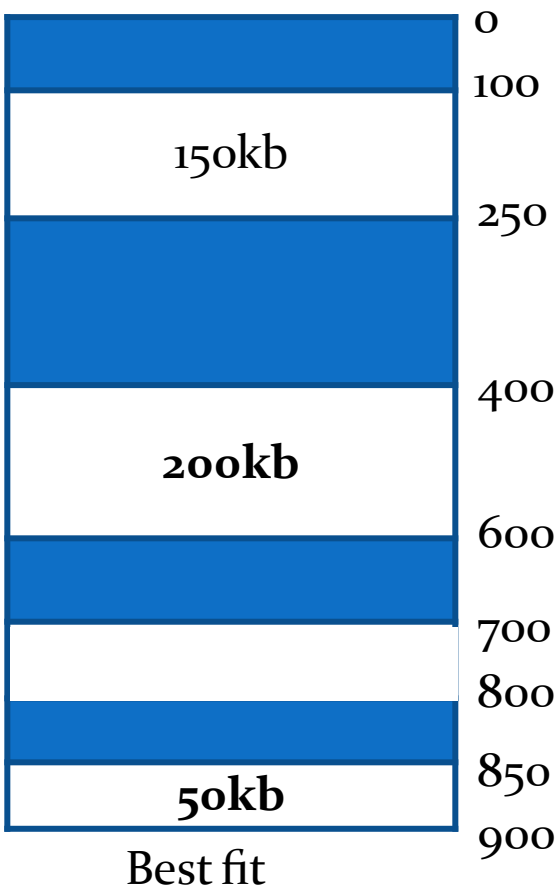
Practice Problem

CODERINDEED

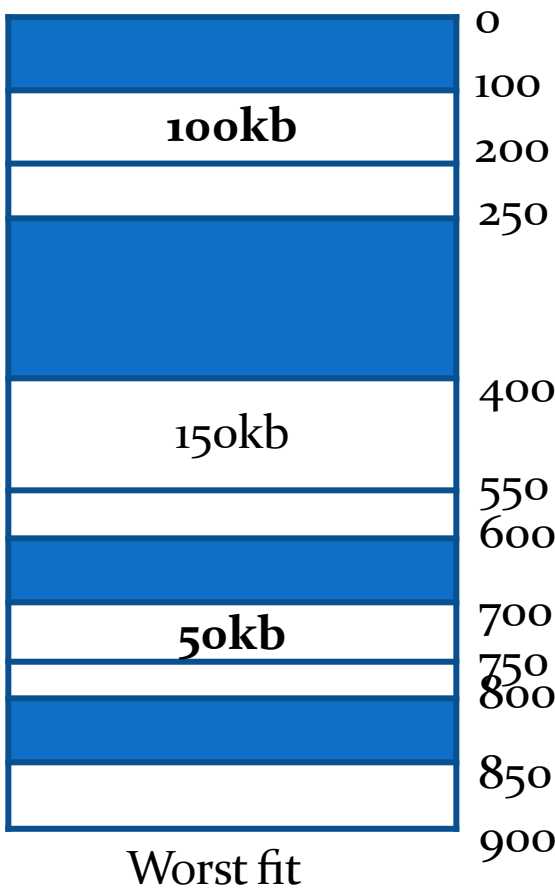
- Given four free memory partitions of 150 KB, 200 KB , 100 KB and 50 KB(in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 150 KB, 100 KB, 200 KB and 50 KB (in order)? Which algorithm performs the best?



First fit



Best fit



Worst fit

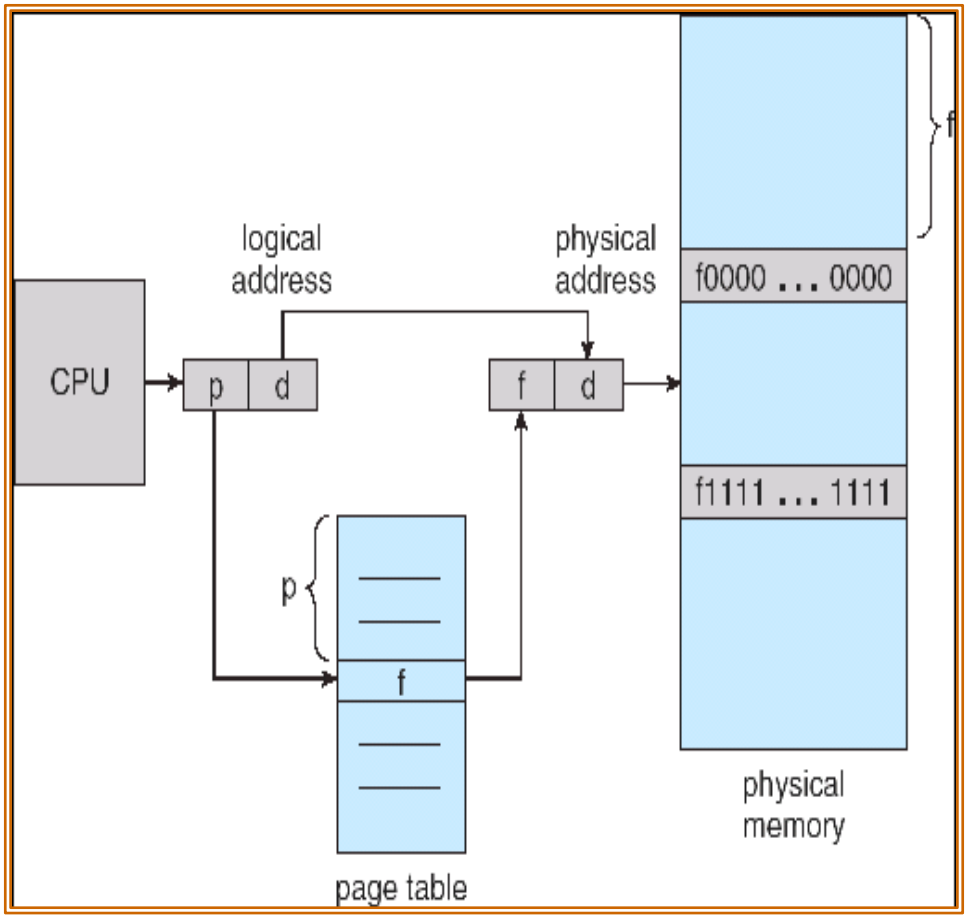
Paging

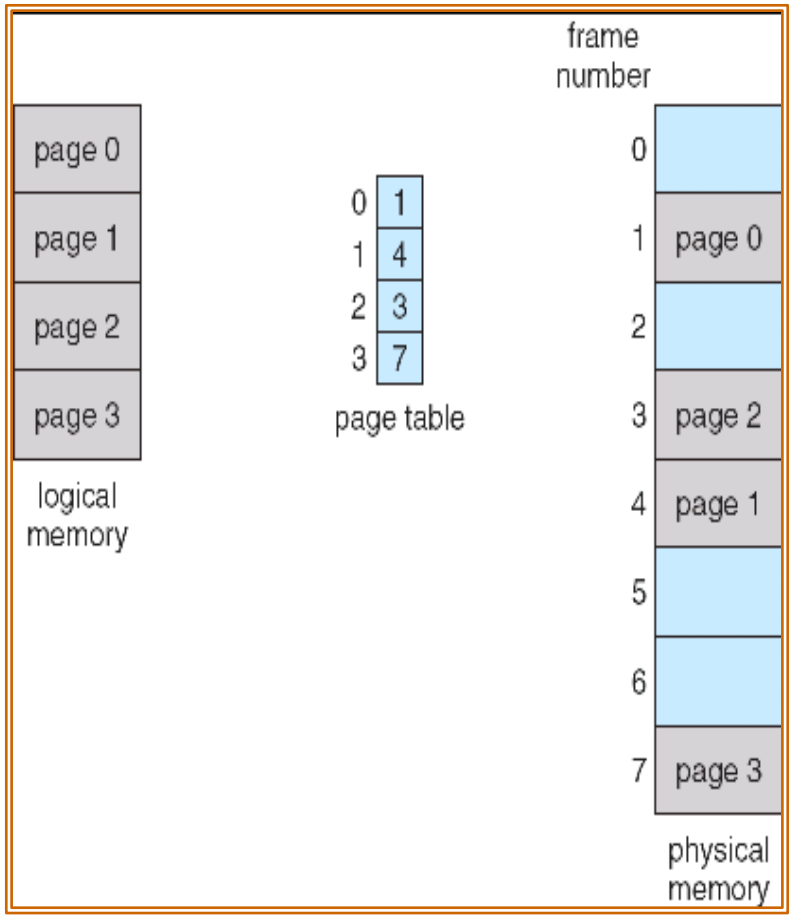
- Physical address space of a process is **non-contiguous.**
- Implementation
  - Frames – fixed size blocks of physical memory
  - Pages – fixed size slots of logical memory
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- Page Table – to translate logical address to physical address

Address Translation

- 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

Address Translation





Address Translation

- How to break logical address into page number and page offset?

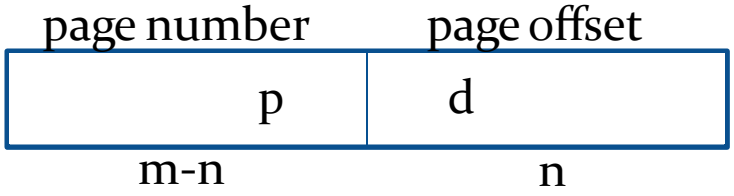
If address space size = 2^m

Page size = 2^n

Then

Page number = higher order (m-n) bits

Page offset = n low-order bits



Example

Q. Using a page size of 4bytes and physical memory of 32 bytes, find the physical address if the logical address is

a) 4

b)10

Solution:

a) Page size = 4bytes = 2^2 (i.e. n=2)

Address space = 32 = 2^5 (i.e. m=5)

Logical address = 4

In binary = 00100

Page number=(higher)(m-n)bits=5-2=3 bits

i.e. 001 = 1

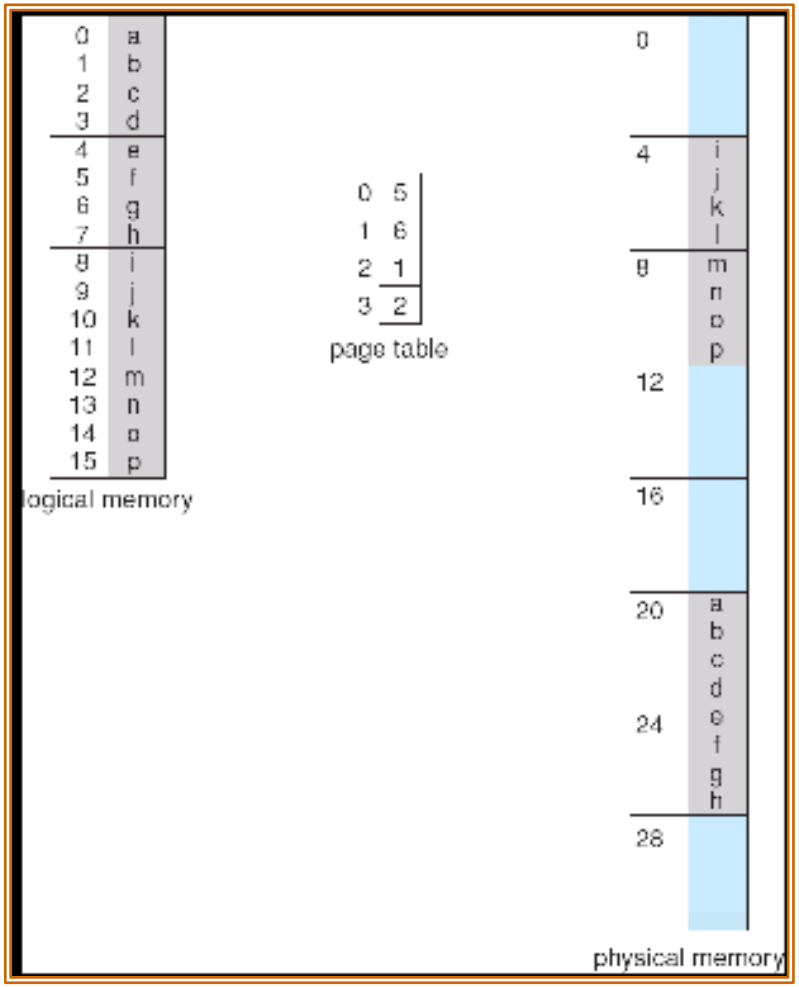
Offset = (lower)n bits = 2 bits i.e. 00 = 0

From page table, if page number = 1

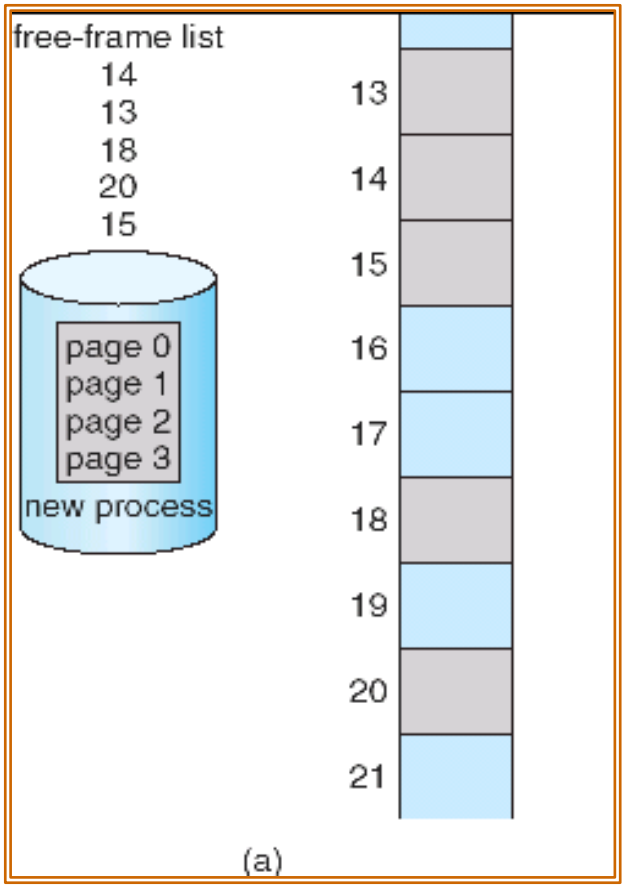
Then frame = 6

Physical address = frame*page size+offset

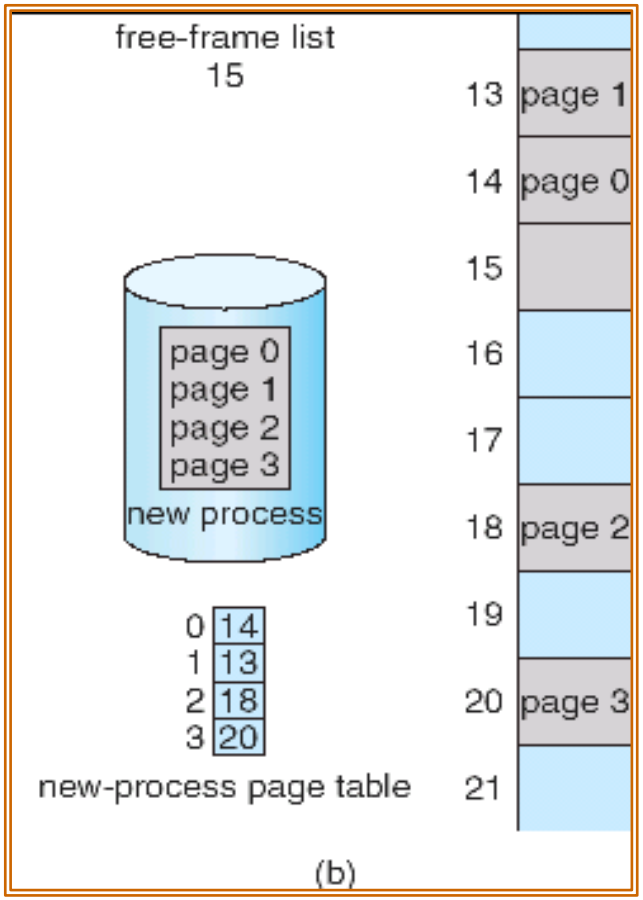
= 6*4+0 = 24



Frame allocation



Before allocation



After allocation

Hardware Support

Problem

Q. If it takes 20 nanoseconds to search the TLB , 100 nanoseconds for PT and 100 nanoseconds to access memory, then find the effective access time if the hit-ratio is 80%.

Solution:

Time taken to access data if page is found in TLB = 20+100=120

Time taken to access data if page is not found in TLB = 20+100+100 = 220

Effective access time = .80*120 + .20*220

= 140 nanoseconds

Problem

Q. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then find the effective access time if the miss-ratio is 2%.

Solution:

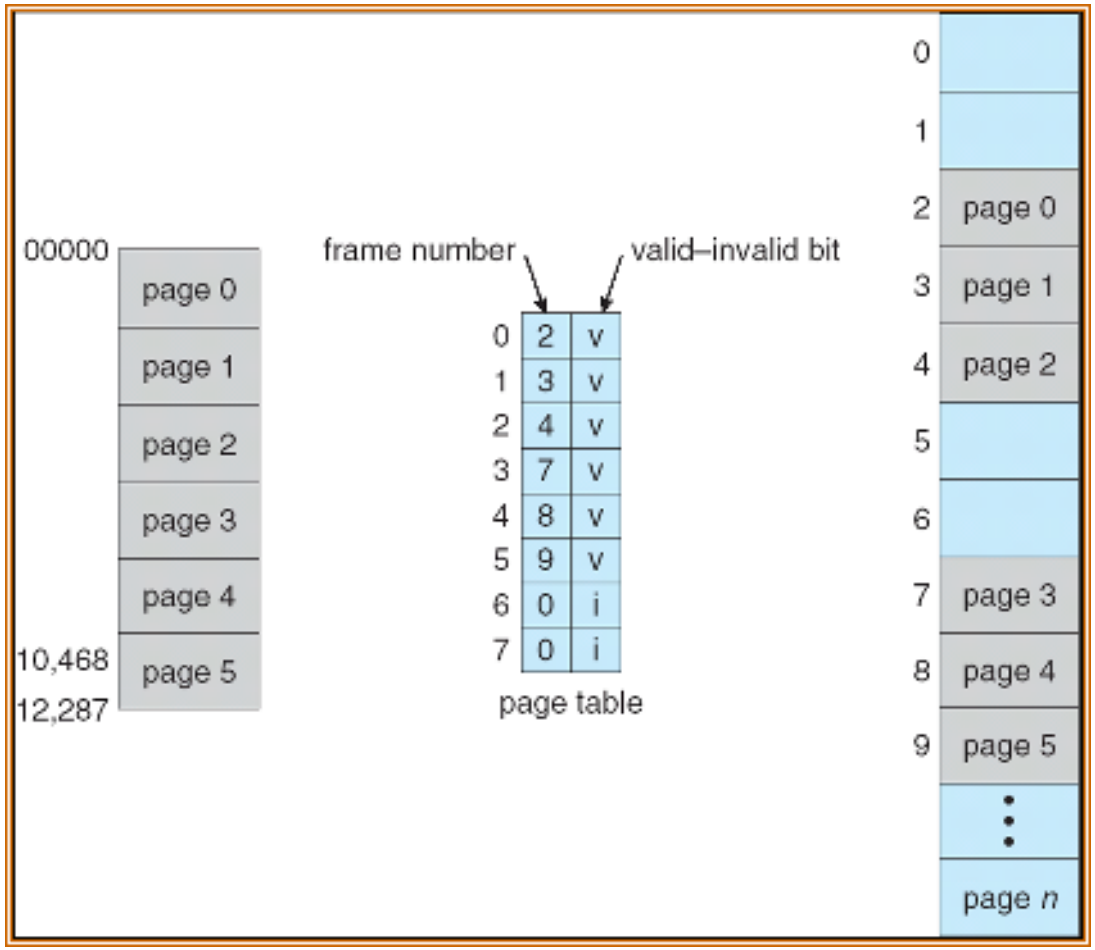
Effective access time = .98*120 + .02*220

= 122 nanoseconds

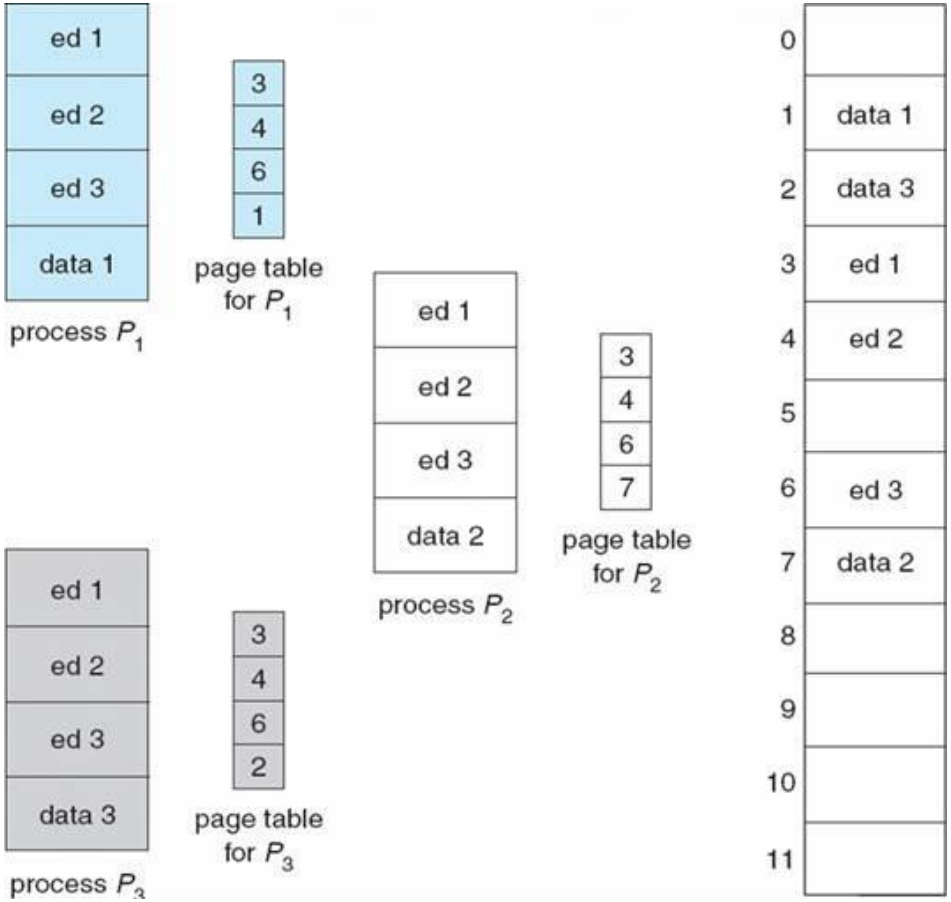
Memory Protection

- 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



Shared Page tables

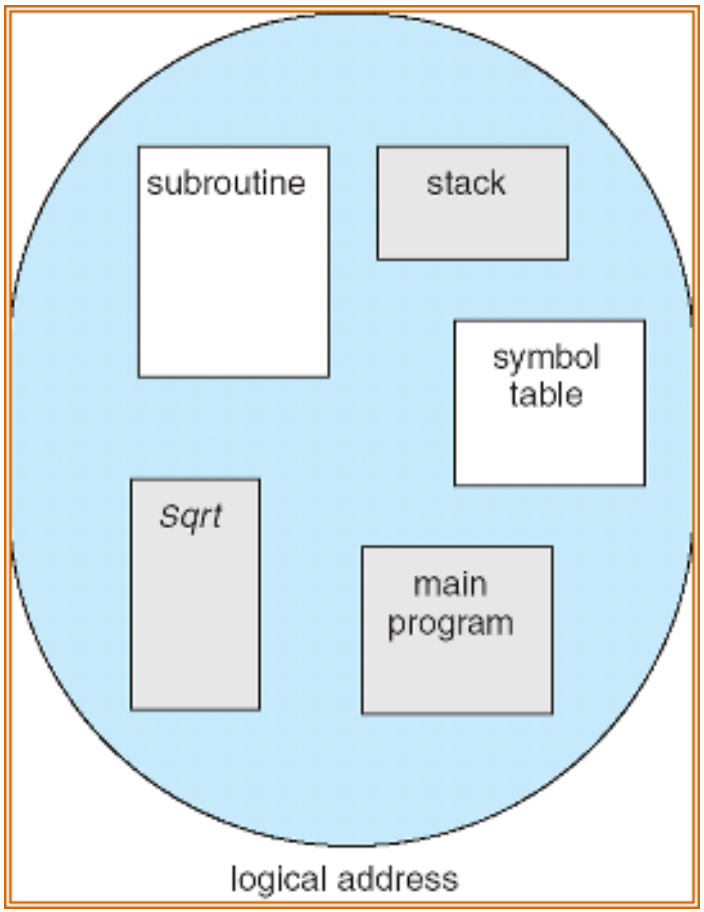


Segmentation

- Users view memory as a collection of variable size segments. With no necessary ordering of these segments
- Segmentation is memory-management scheme that supports user view of memory
- 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 a program

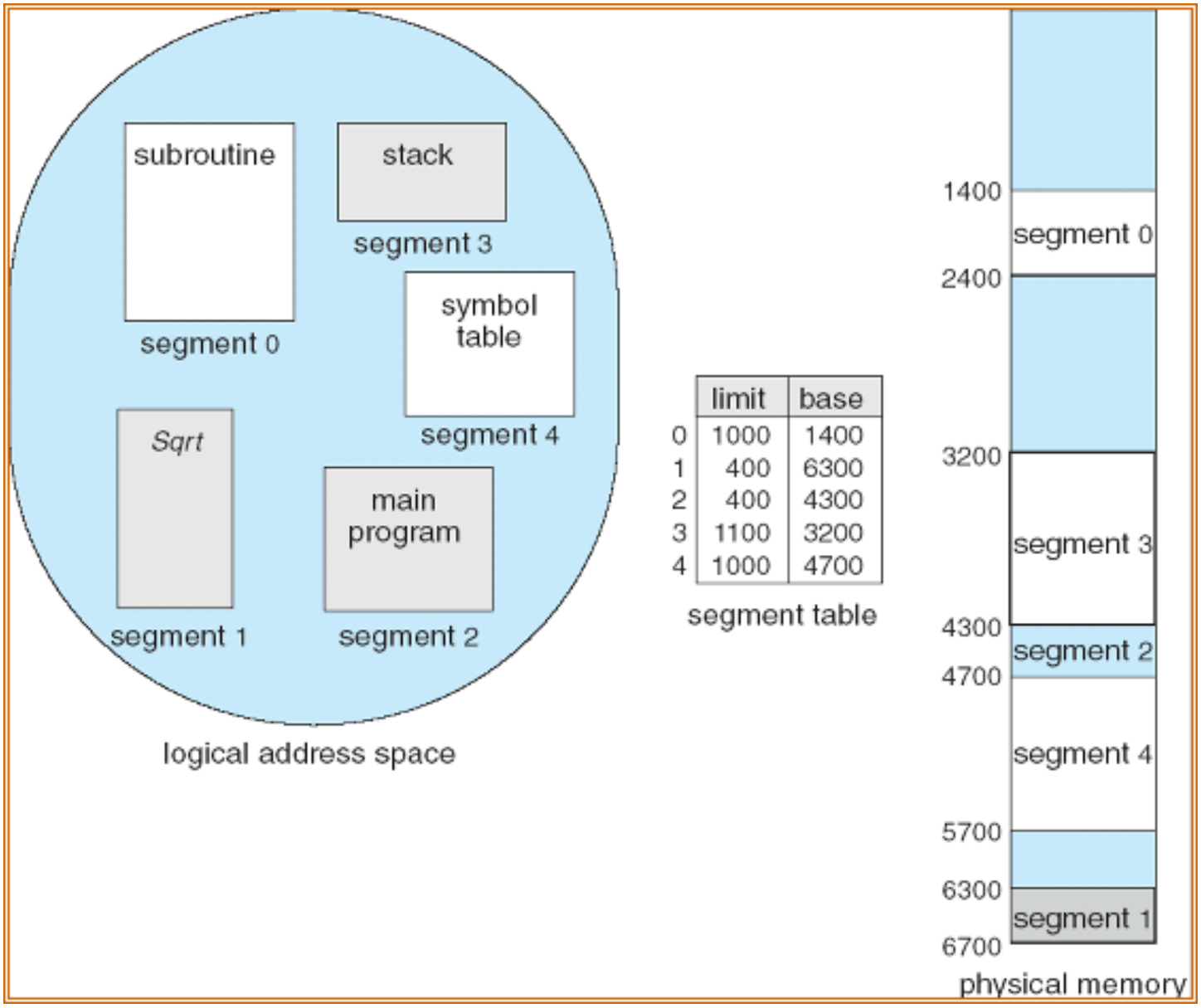


Segmentation hardware

- Each segment has a name and its length.
- 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



Example

- Segment 2 is 400 bytes long and begins at location 4300.
- Thus a reference to byte 53 of segment 2 is mapped onto location 4300+53=4353.

Problem

Q. What is the physical address if the byte is 852 of segment 3?

Ans:

3200+852=4052.

Q. What is the physical address if the byte is 1222 of segment 0 ?

Ans: Would result in a trap to OS, as this segment is only 1000 bytes long.