•Memory Management

Background

- It is a method of managing various types of memory especially the primary memory (RAM).

- Program must be brought secondary memory (from disk) into primary memory and placed within a process for it to be run

- Main memory (RAM) and registers are only storage CPU can access directly

Multiprogramming

- Multiple Programs stored at the same time
  - into different areas of memory

- Processor needs to context switch

- Programs use memory addresses / references

- Proper addressing is required
 - no matter where the program is located in memory

Degree of Multiprogramming

- CPU utilization = $1 - P^n$

  - n process in memory
  - P is the I/O block time by one process
  - P^n probability of all process are waiting for I/O

- I/O waiting 80% of time
  - CPU utilization = $1 - 0.8^1$ =20% [1 process @ memory]
  - CPU utilization = $1 - 0.8^3$ =49%   [3 processes @ memory]
  - CPU utilization = $1 - 0.8^{10}$ =89% [10 processes @ memory]

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high

Schematic View of Swapping
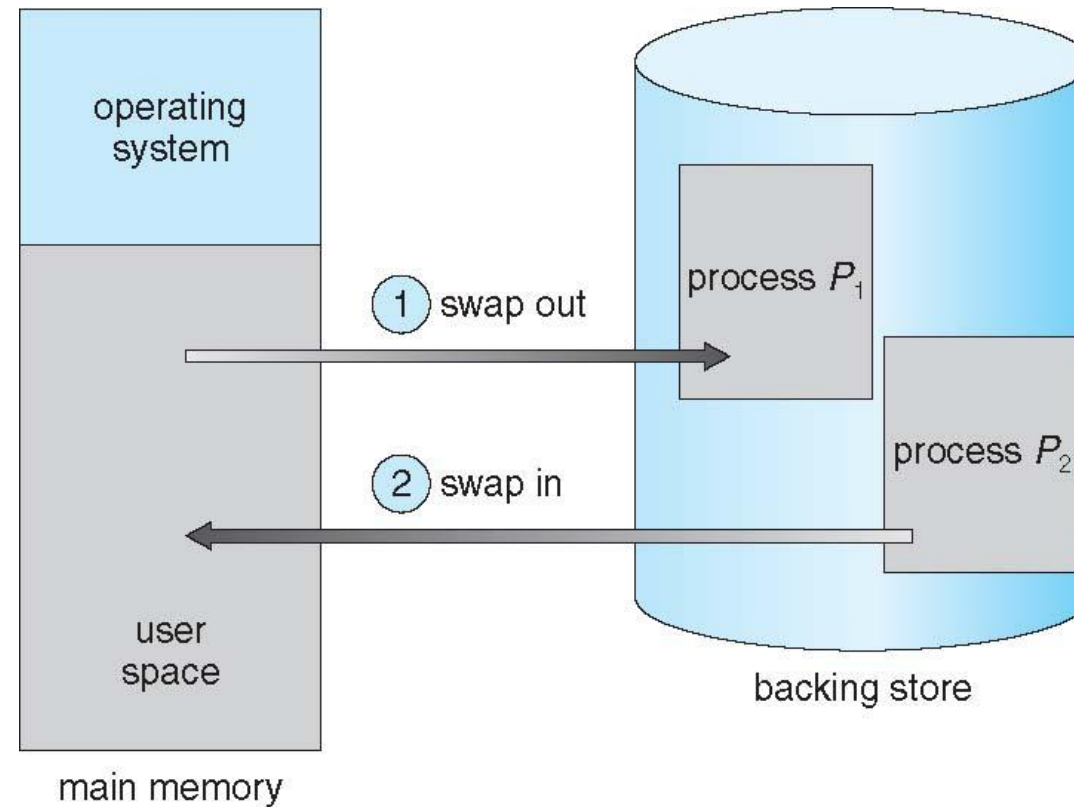
Memory Management Techniques

- Contiguous

  – Fixed partitioning (static)

  – Variable partitioning (dynamic)
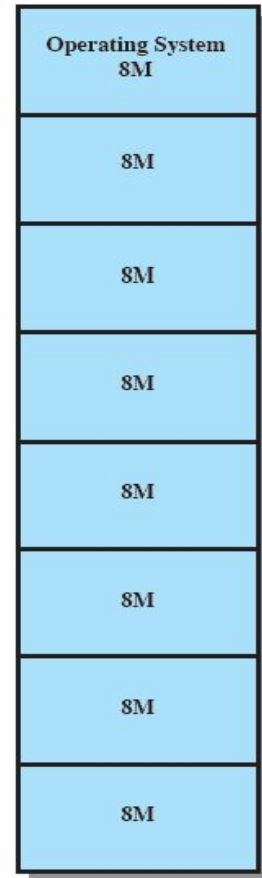
- Non-contiguous

  – Paging

  – Segmentation

Contiguous Allocation

- Base register contains value of smallest physical address
- Limit register contains range of logical addresses – each logical address must be less than the limit register
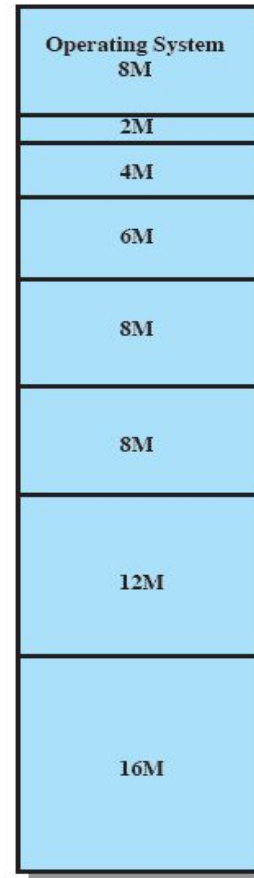- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers

Fixed Partitioning

Operating System 8M	Operating System 8M
8M	2M
8M	4M
8M	6M
8M	8M
8M	8M
8M	12M
8M	16M
8M	
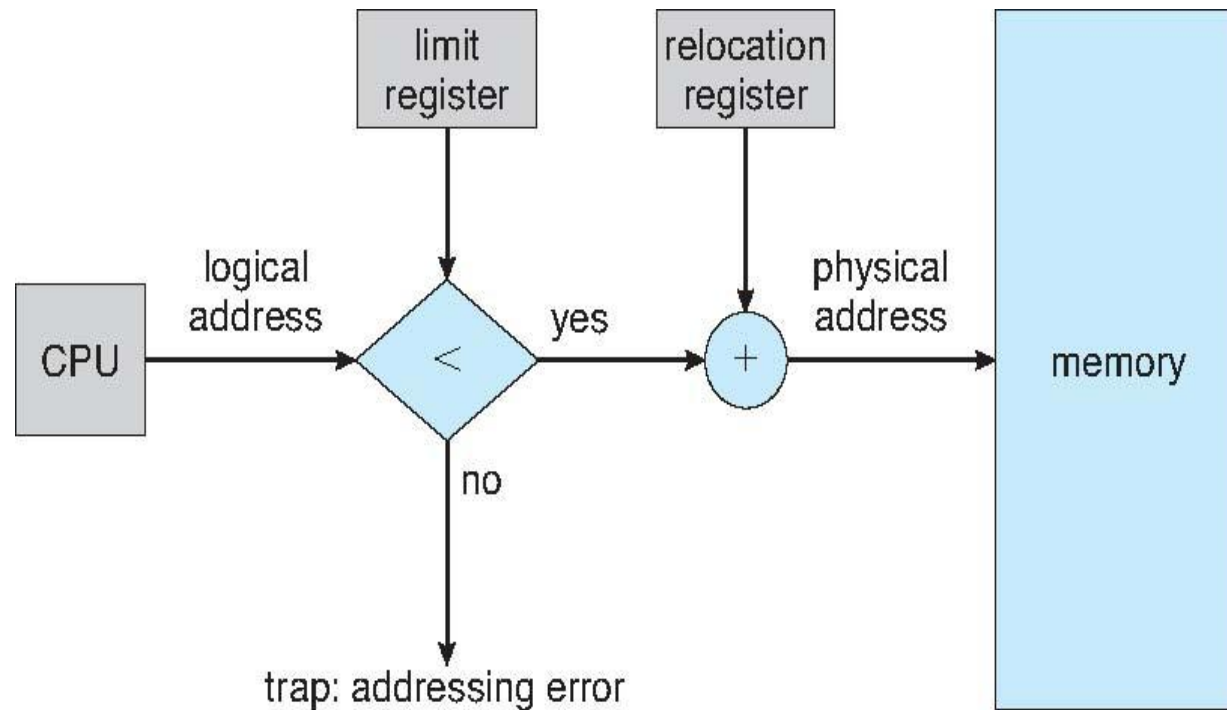
(a) Equal-size partitions   (b) Unequal-size partitions

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Fixed Partitioning

- Number of partitions are fixed
- Size of each partition may or may not be same
- Since contiguous allocation, so spanning is not allowed

Disadvantages:

- Internal fragmentation
- Process size is limited
- Limitations on degree of multiprogramming

Variable Partitioning

- There are no partitions beforehand.
- Partition will be created according to the process size during runtime.

Advantages:

- No internal fragmentation
- No limitation on number of processes
- No limitation on process size

Disadvantages:

- External fragmentation (hole) will be created.
- Allocation and reallocation is complex.
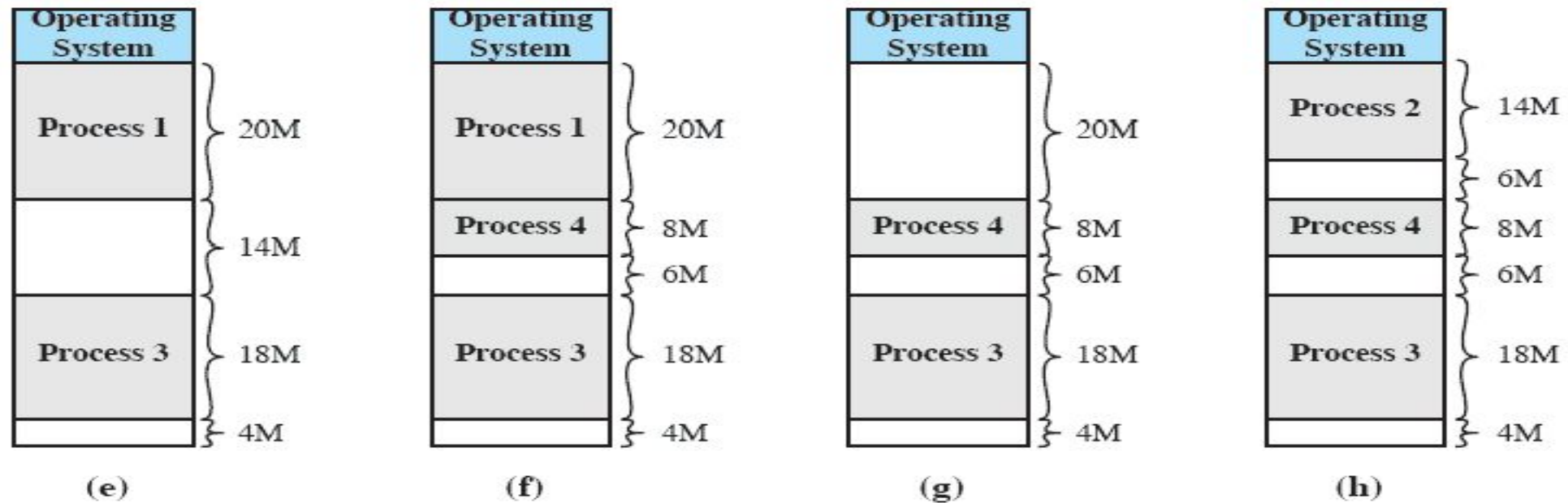
Effects of Variable Partitioning

Figure 7.4 The Effect of Dynamic Partitioning

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
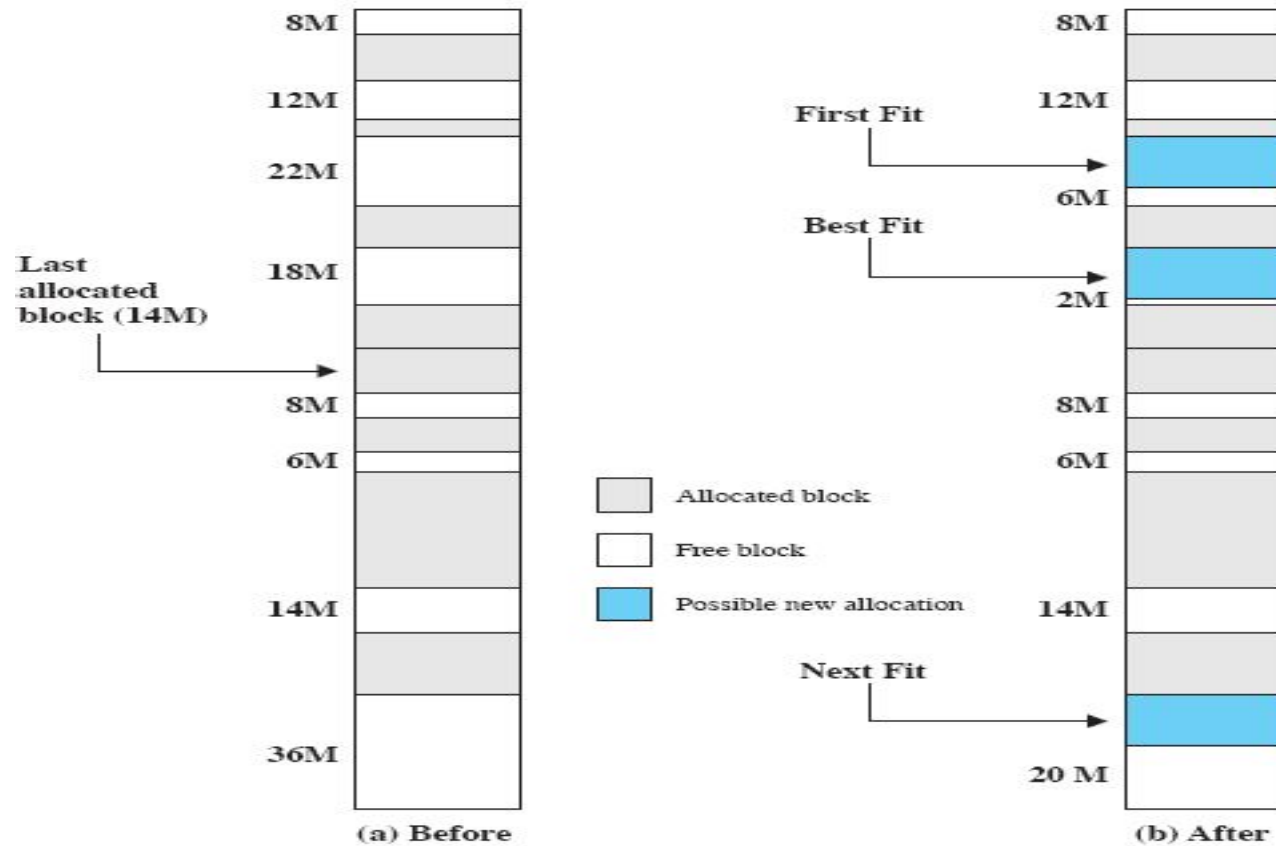 - Produces the largest leftover hole
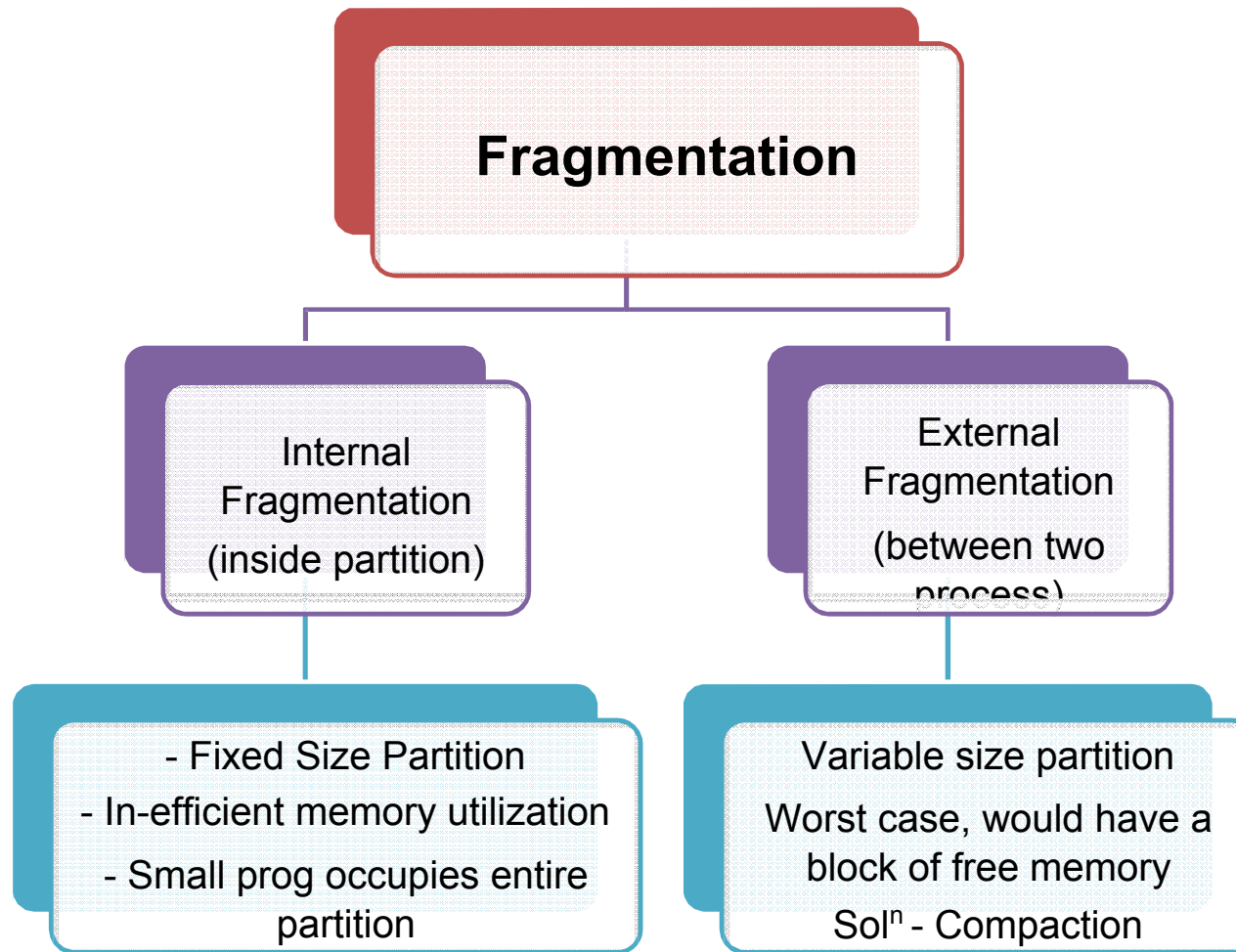
Allocation Schemes

Figure 7.5   Example Memory Configuration before
and after Allocation of 16-Mbyte Block

Fragmentation

Internal Fragmentation
(inside partition)

External Fragmentation
(between two process)

- Fixed Size Partition
- In-efficient memory utilization
- Small prog occupies entire partition

Variable size partition

Worst case, would have a block of free memory

Sol^n - Compaction

Compaction

- 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

Disadvantages
- Before shifting, the processes at running states need to be halted
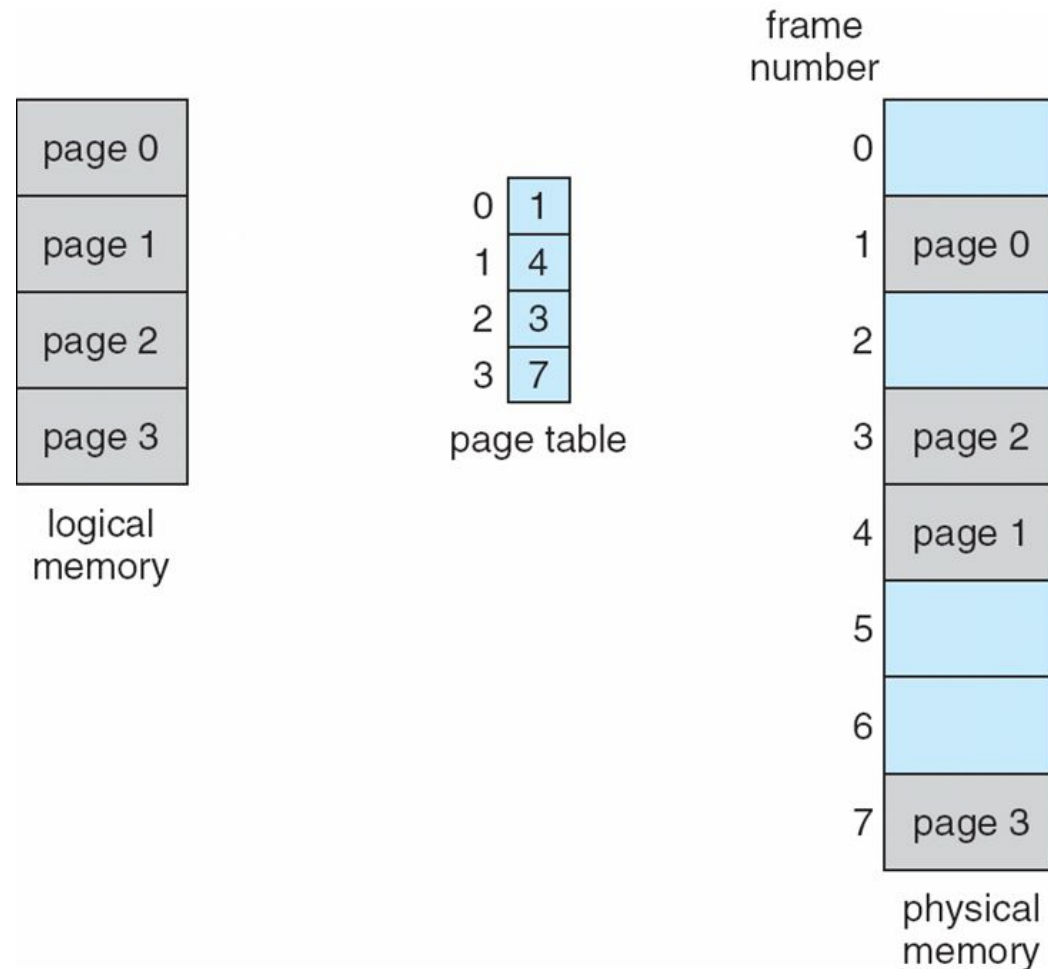- Time consuming

Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available

  - Avoids external fragmentation

  - Avoids problem of varying sized memory chunks

- 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
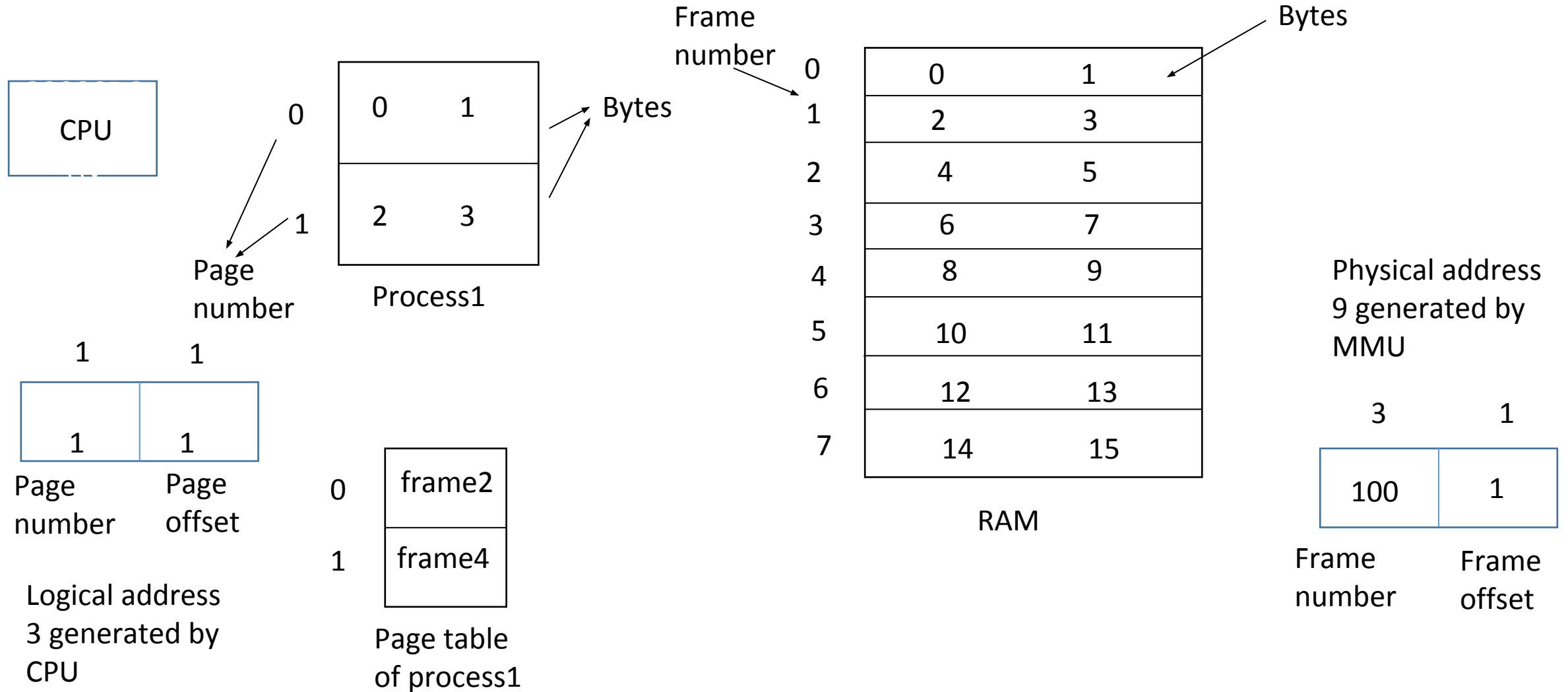
- Backing store likewise split into pages

Paging Hardware
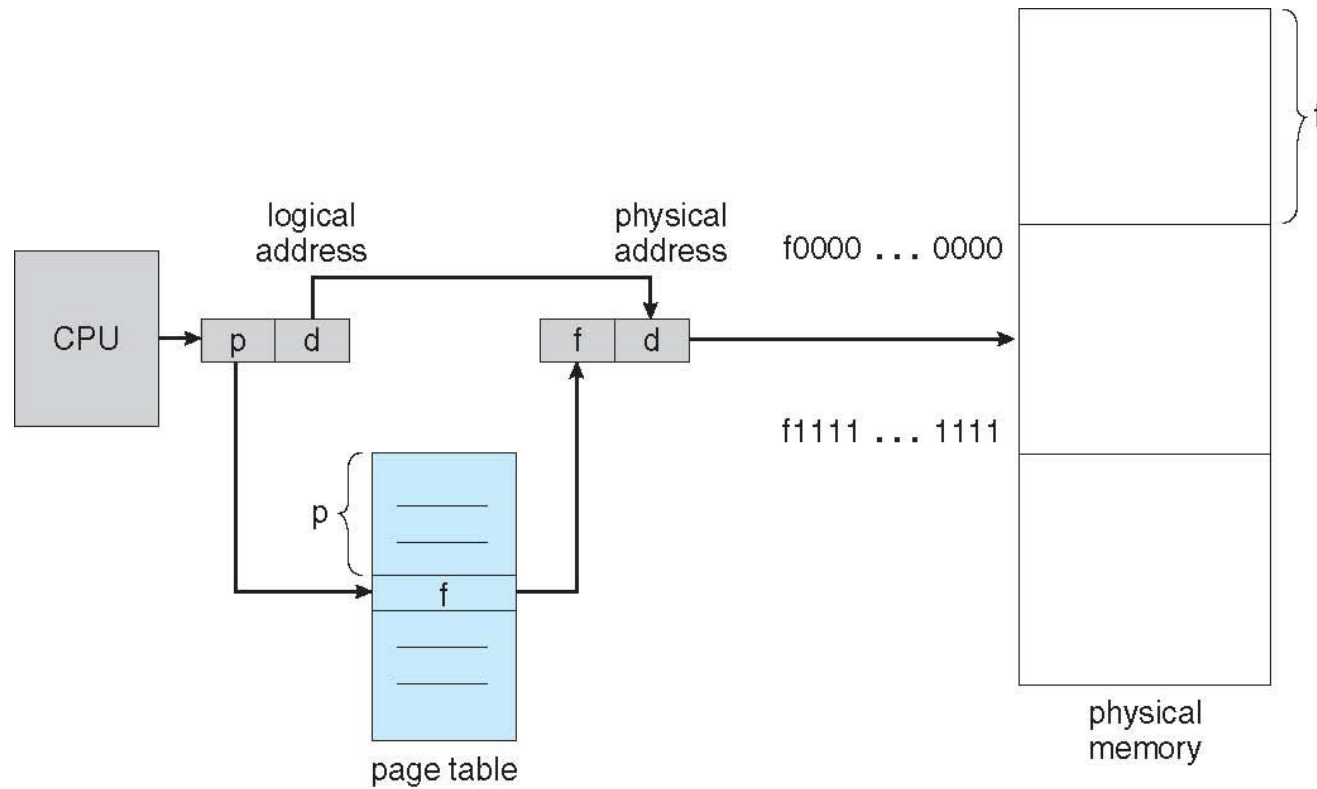
Paging Model of Logical and Physical Memory

Paging Explained With an Example

CPU

Process1

0   1

2   3

Page number

Bytes

Page table of process1

| 0 | frame2 |
| 1 | frame4 |

Logical address 3 generated by CPU

Page number	Page offset
1	1

RAM

Frame number

	Bytes	
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	11
6	12	13
7	14	15

Physical address 9 generated by MMU

Frame number	Frame offset
100	1

Paging Explained With an Example(cont.)

- Process size is 4B

- Page size is 2B

- Number of pages is 2 (4/2).

- RAM size is 16B

- Frame size is 2B

- Number of frames is 8 (16/2).

Paging Calculations

Consider a system which has logical address which is represented by 7 bits, physical address represented by 6 bits, page size of 8 bytes, then calculate number of pages and number of frames.

4	3

Page number   Page offset

Logical address

3	3

Frame number   Frame offset

Physical address

Paging Calculations (Cont.)

- Total number of pages= 2^4=16

- Total number of frames= 2^3=8

- Logical address space=2^7=128

- Physical address space=2^6=64

- Number of entries in a page table of a process= number of pages in a process =16

# Powers of 2

- $2^1=2$ $2^8=256$
- $2^2=4$ $2^9=512$
- $2^3=8$ $2^{10}=1K$
- $2^4=16$ $2^{20}=1M$
- $2^5=32$ $2^{30}=1G$
- $2^6=64$ $2^{40}=1T$
- $2^7=128$

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**)

Paging Hardware With TLB
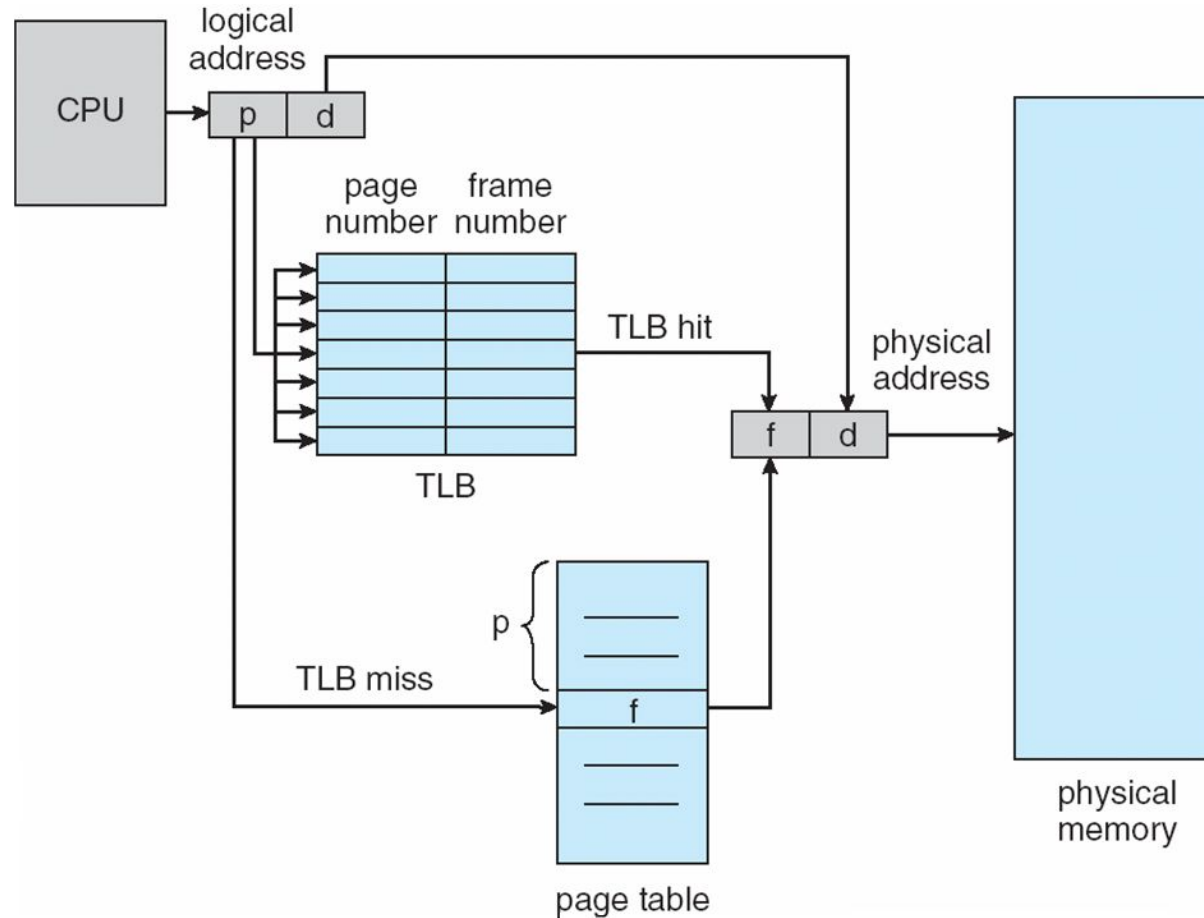
Effective Memory Access Time

Effective access time=

#TLB hit x (TLB access time + memory access time) +

#TLB miss x   (TLB access time + Page table access time + memory access time)

Memory Access Time 100 nanoseconds

100% hit ratio:1.0 x (20+100) nanoseconds =120 nanoseconds
80% hit ratio:0.8 x (20+100) + 0.20x (20+100+100) nanoseconds=140 nanoseconds
98% hit ratio:0.98 x (20+100) + 0.02x (20+100+100) nanoseconds
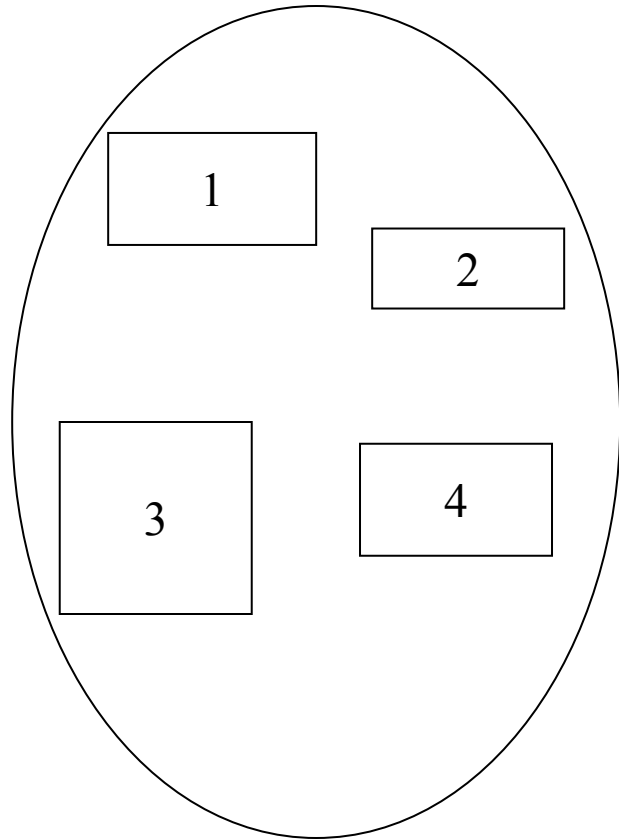=122 nanoseconds

Segmentation

- 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

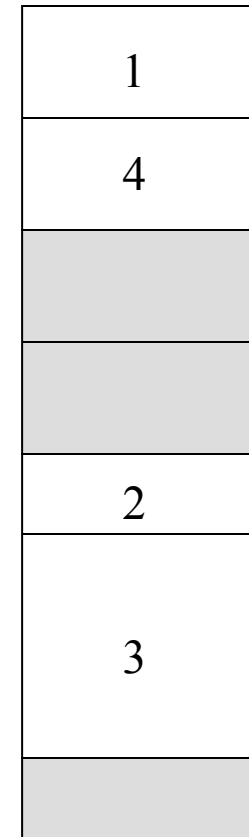stack

symbol table

arrays

User's View of a Program
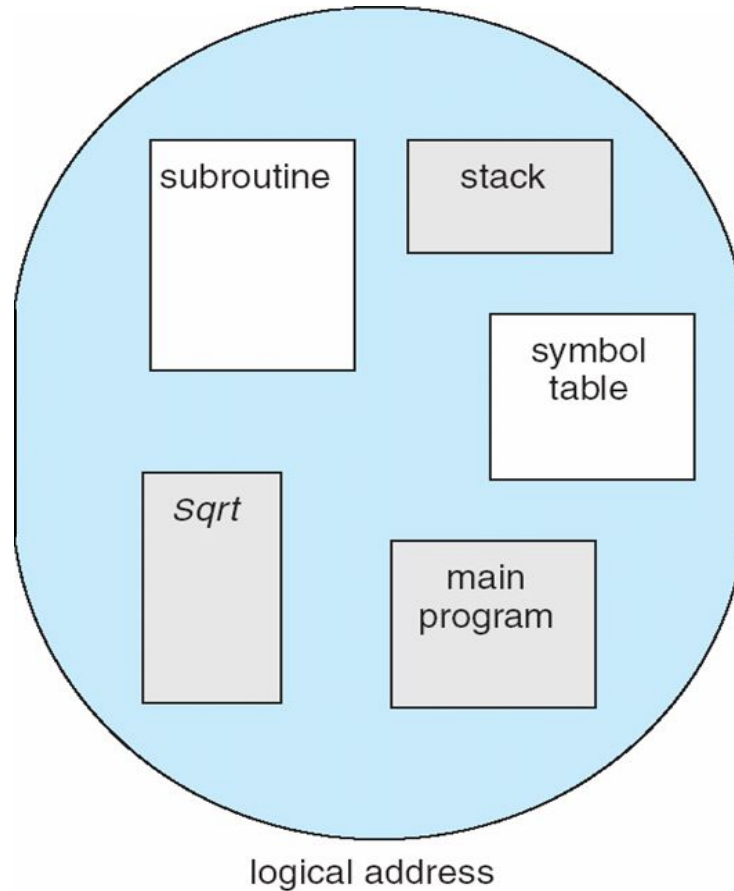
Logical View of Segmentation

user space

physical memory space

Segmentation Hardware