GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A Grade by NAAC

OPERATING SYSTEMS

Memory Management

Class Presentations on Operating System

by Subhash Chand Agrawal

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - Gigabytes slow, cheap disk storage

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
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a processor **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Basic Hardware

➢Each process has a separate memory space

➢We need to defined the range of legal addresses that a process may access.

➢This is ensured by two registers:

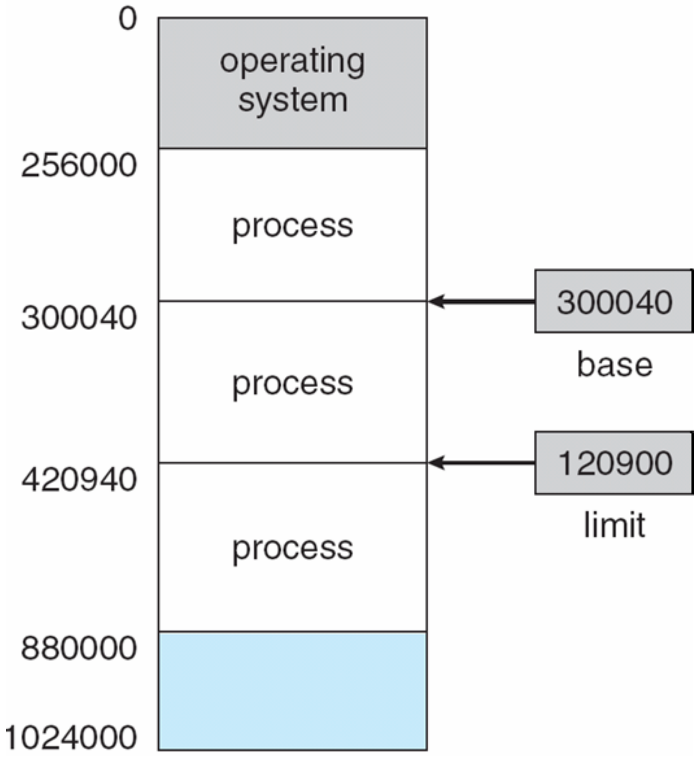
➢Base Register: Holds the smallest legal physical memory address.

➢Limit Register: specifies the size of the range.

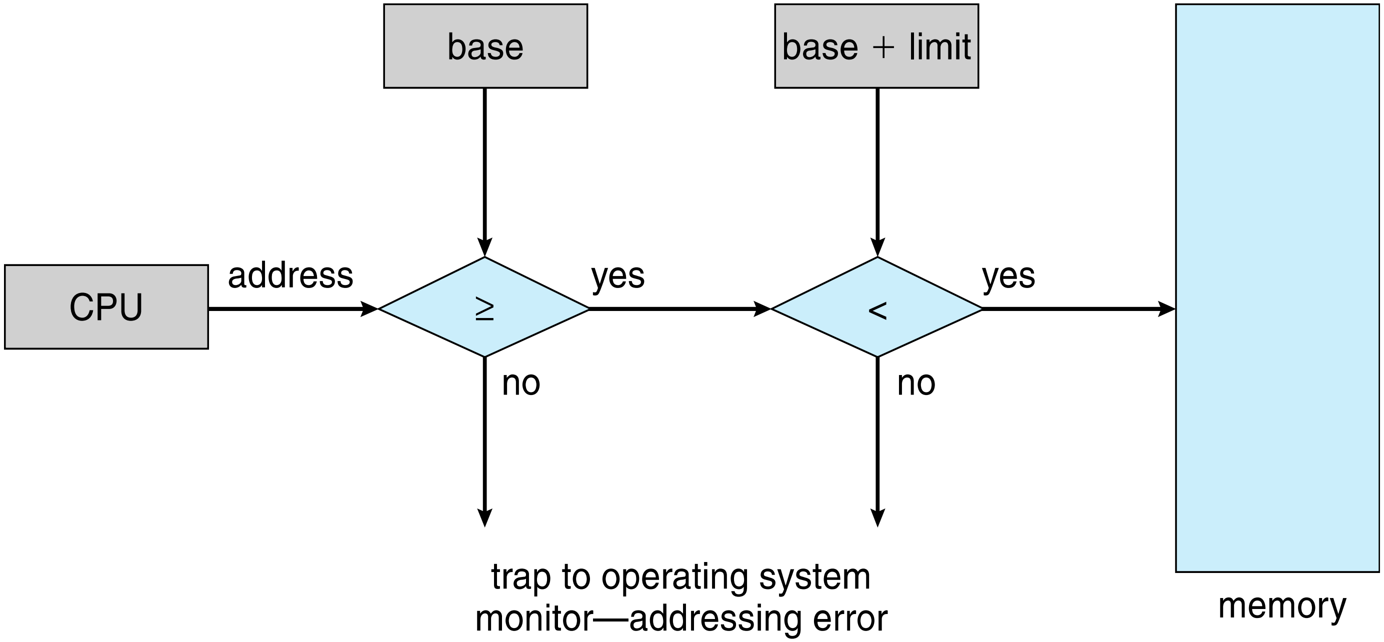
➢**For e.g. if the base register holds 1250 and limit register is 356, then the program can access all addresses from 1250 to 1605(inclusive)**

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

- **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

If, Physical Address Space(in words) = N Words then, Physical Address (in bits) = $\log_2 N$

If, logical address space = L words
Then, Logical Address = $\text{Log}_2 L$ bits

Review

- Physical Address Space = Size of the Main Memory
- If, physical address space = 4 K words then what will be the physical address?

12 bits

- Logical Address = 23 bits,then what will be the logical address space?

8 MB or 8 M words

Review

- If, physical address space = 64 KB and Let us consider word size = 8 Bytes
- Physical address space (in words)

2 ^ 13 Words

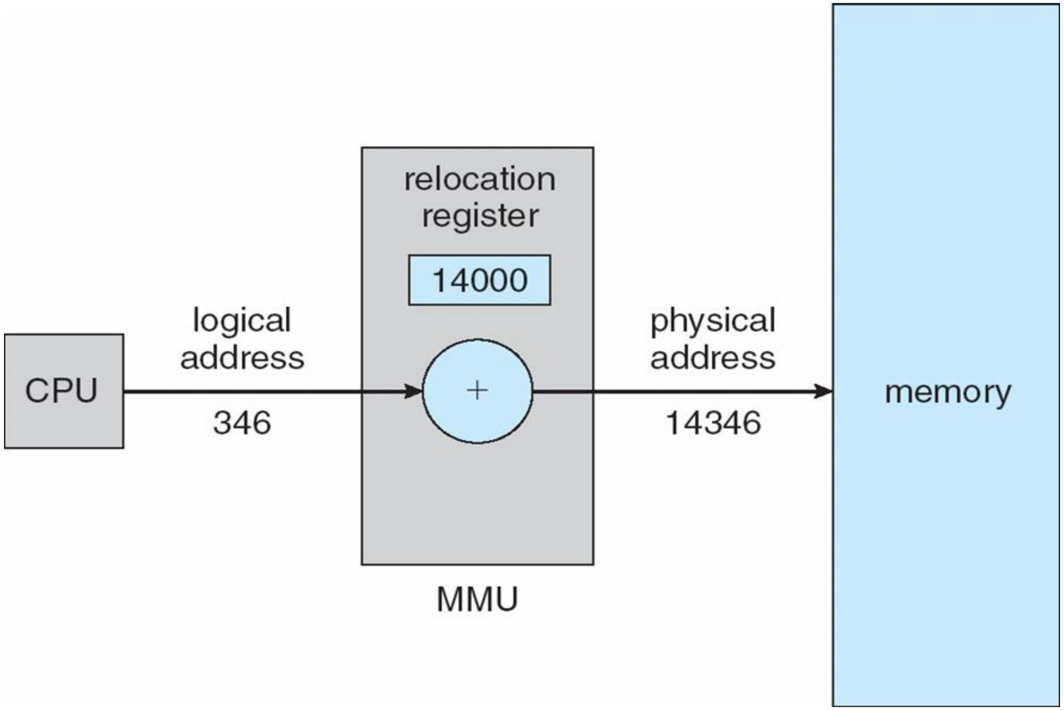
- Physical Address =

13 bits

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**
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Dynamic relocation using a relocation register



Memory Management Techniques

- There are two Memory Management Techniques: **Contiguous**, and **Non-Contiguous**.
- In Contiguous Technique, executing process must be loaded entirely in main-memory.
- Contiguous Technique can be divided into:
  - Fixed (or static) partitioning
  - Variable (or dynamic) partitioning

Fixed Partitioning:

- The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.
- In this technique, the main memory is divided into partitions of equal or different sizes.
- The operating system always resides in the first partition while the other partitions can be used to store user processes.
- In fixed partitioning,
  - The partitions cannot overlap.
  - A process must be contiguously present in a partition for the execution.

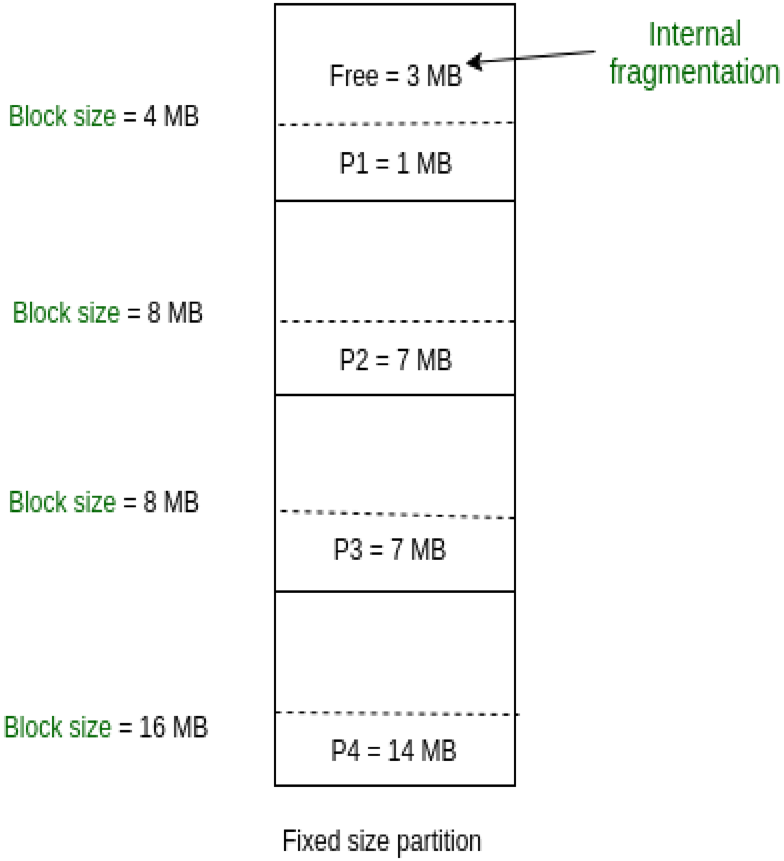
Disadvantages

Internal Fragmentation

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

External Fragmentation

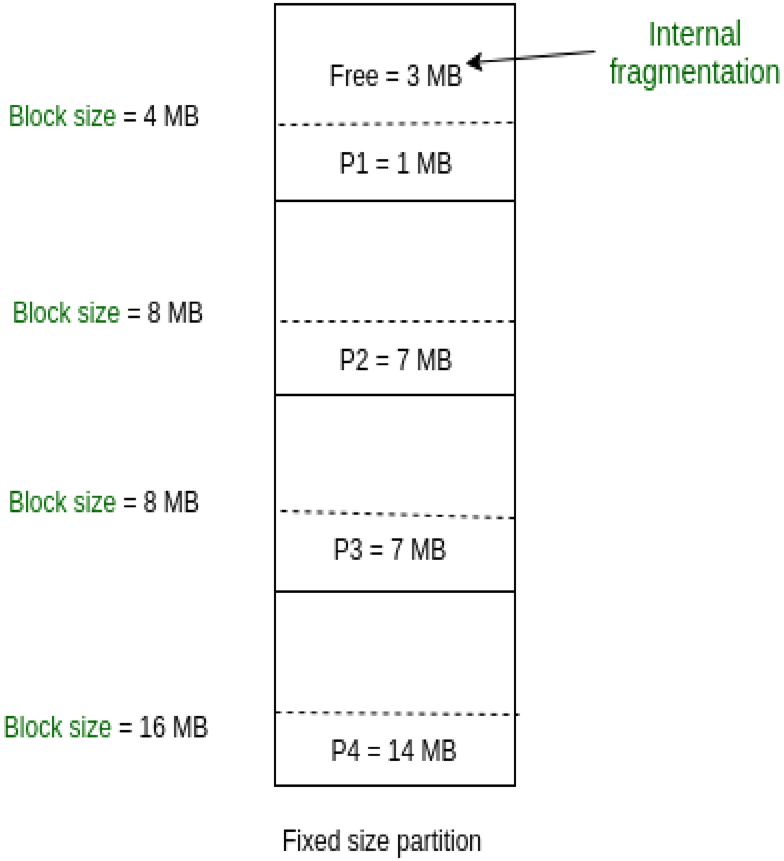
The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.



Disadvantages

Limitation on the size of the process
Process of size greater than size of partition in Main Memory cannot be accommodated.

Limitation on Degree of Multiprogramming:
Suppose if there are n1 partitions in RAM and n2 are the number of processes, then condition n2<=n1 must be fulfilled.



Dynamic Partitioning

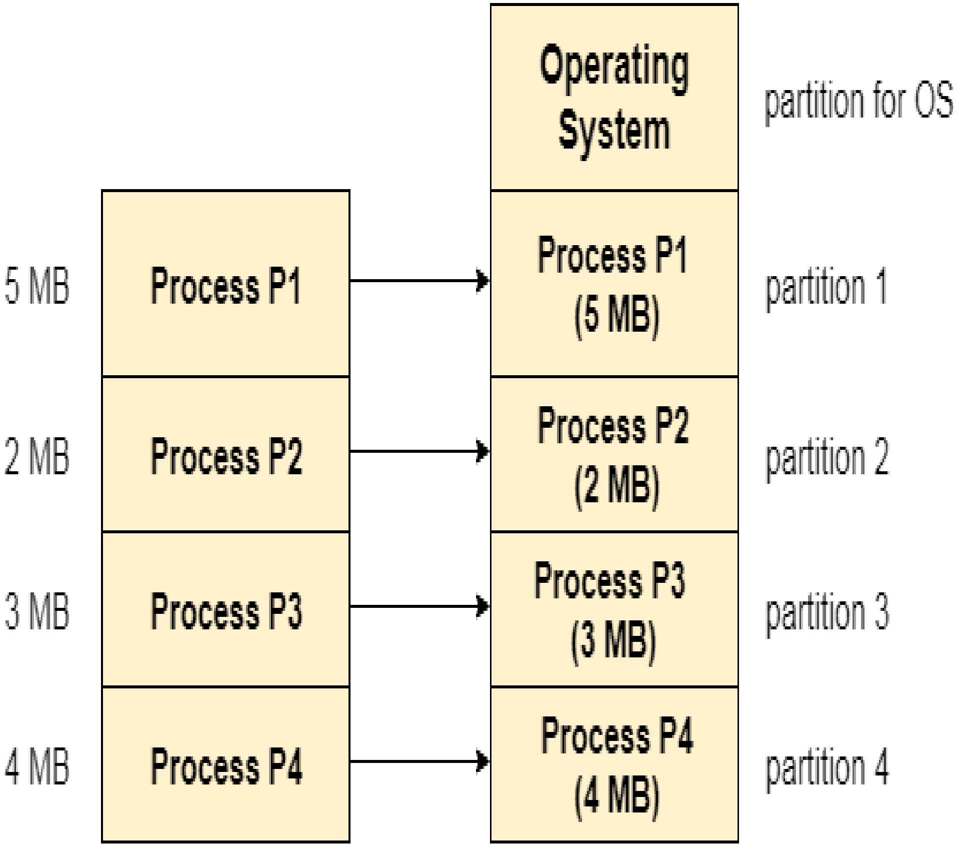
- Dynamic partitioning tries to overcome the problems caused by fixed partitioning.
- In this technique, the partition size is not declared initially.
- It is declared at the time of process loading.
- The first partition is reserved for the operating system.
- The remaining space is divided into parts.
- The size of each partition will be equal to the size of the process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided.

Advantages of Dynamic Partitioning over fixed partitioning

No Internal Fragmentation
there will not be any unused remaining space in the partition.

No Limitation on the size of the process

Degree of multiprogramming is dynamic
Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

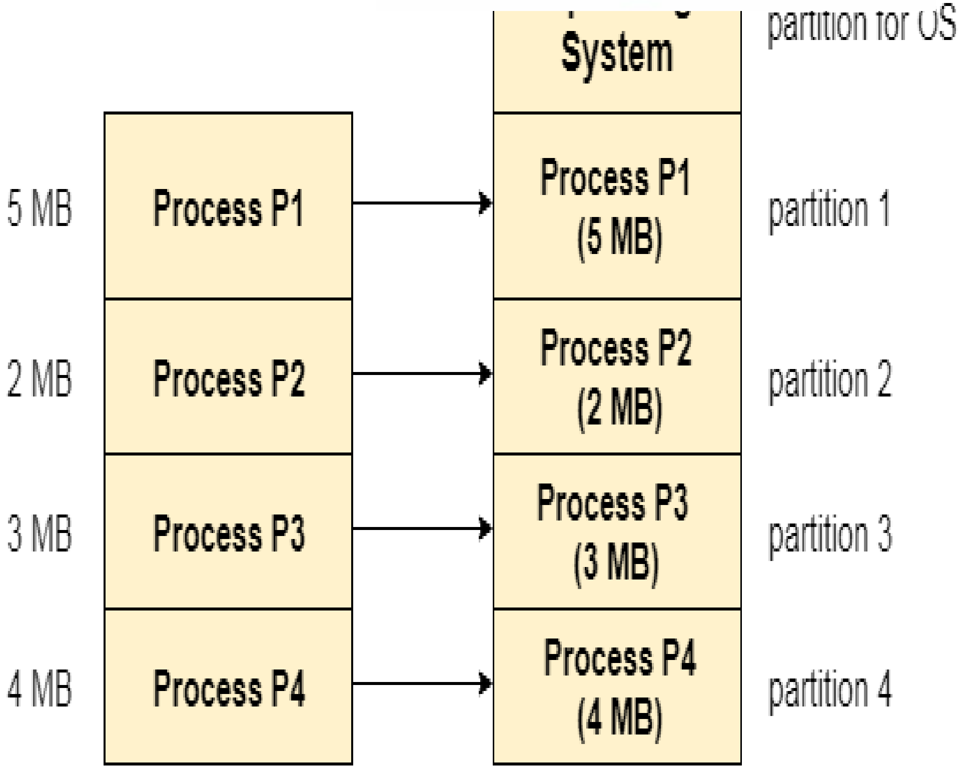


Dynamic Partitioning

(Process Size = Partition Size)

Disadvantages

External Fragmentation
Absence of internal fragmentation doesn't mean that there will not be external fragmentation.



Dynamic Partitioning

(Process Size = Partition Size)

Partitioning Algorithms

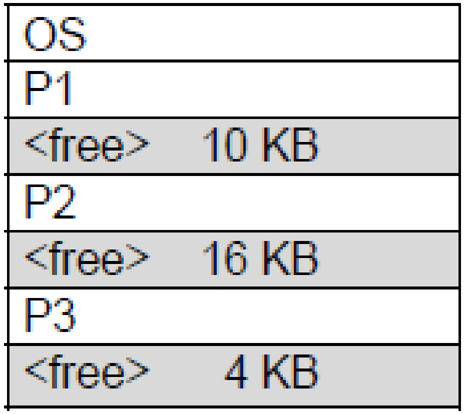
How to satisfy a request of size ***n*** from a list of free holes?

- **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 and best-fit better than worst-fit in terms of speed and storage utilization

Example

- Consider the following memory map and assume a new process P4 comes with a memory requirement of 3 KB. Locate this process.

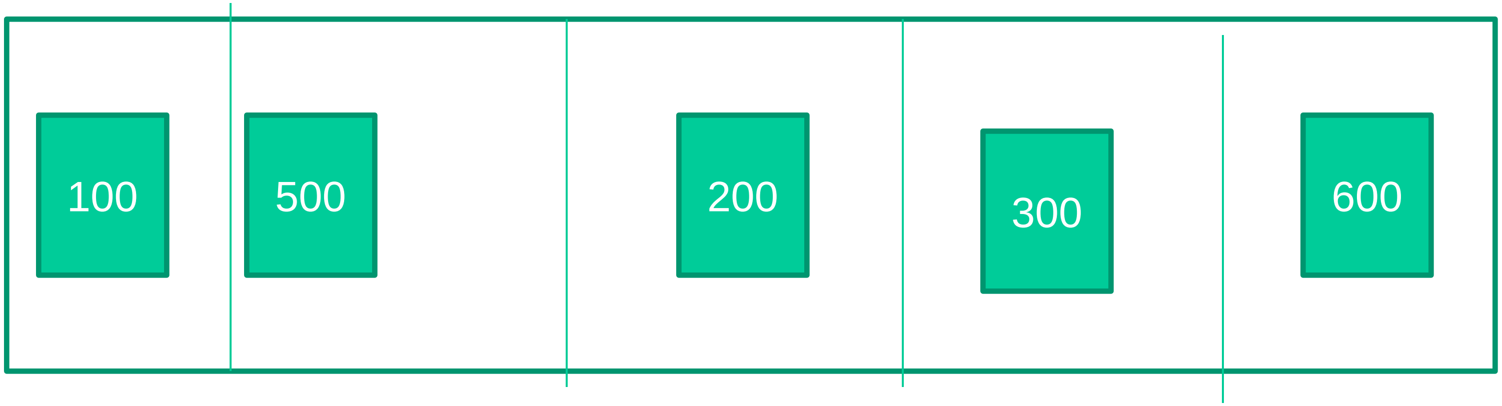


OS
P1
<free> 10 KB
P2
<free> 16 KB
P3
<free> 4 KB

a. First fit algorithm allocates from the 10 KB block.
b. Best fit algorithm allocates from the 4 KB block.
c. Worst fit algorithm allocates from the 16 KB block

Que

- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order),how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?



a. First-fit:

1. 212K is put in 500K partition
2. 417K is put in 600K partition
3. 112K is put in 288K partition (new partition 288K = 500K −212K)
4. 426K must wait

b. Best-fit:

1. 212K is put in 300K partition
2. 417K is put in 500K partition
3. 112K is put in 200K partition
4. 426K is put in 600K partition

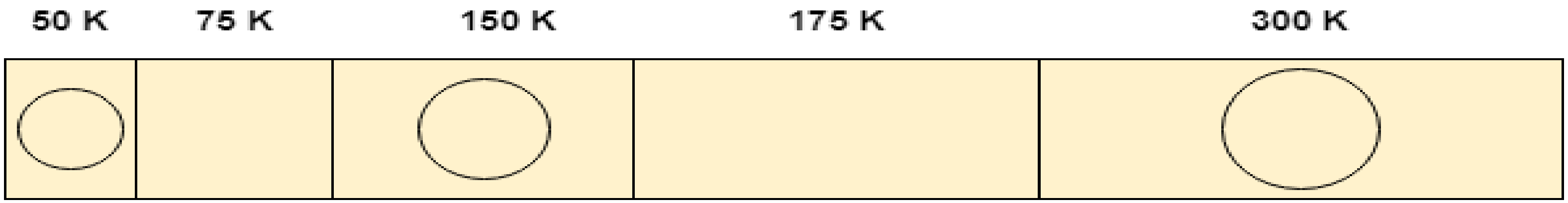
c. Worst-fit:

l. 212K is put in 600K partition

2. 417K is put in 500K partition
3. 112K is put in 388K partition
4. 426K must wait

In this example, best-fit turns out to be the best.

Q. Process requests are given as;
25 K , 50 K , 100 K , 75 K



Determine the algorithm which can optimally satisfy this requirement.

a. First Fit algorithm

b. Best Fit Algorithm

c. Neither of the two

d. Both of them

In the question, there are five partitions in the memory. 3 partitions are having processes inside them and two partitions are holes.

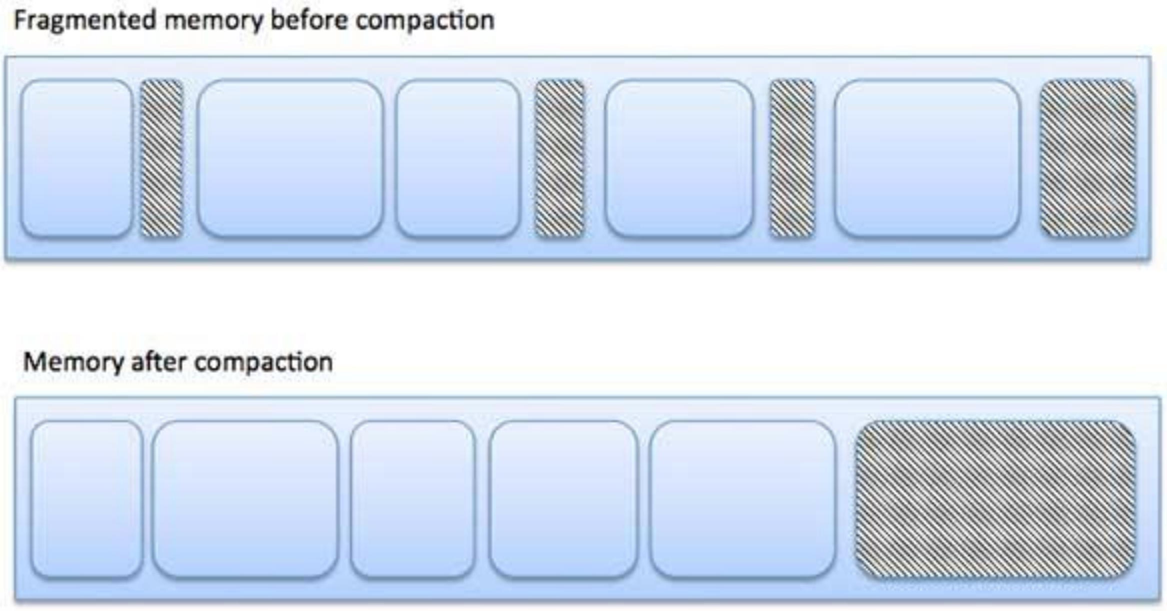
Our task is to check the algorithm which can satisfy the request optimally.

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

Fragmentation (Cont.)

- Use compaction to minimize the probability of external fragmentation.
- In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

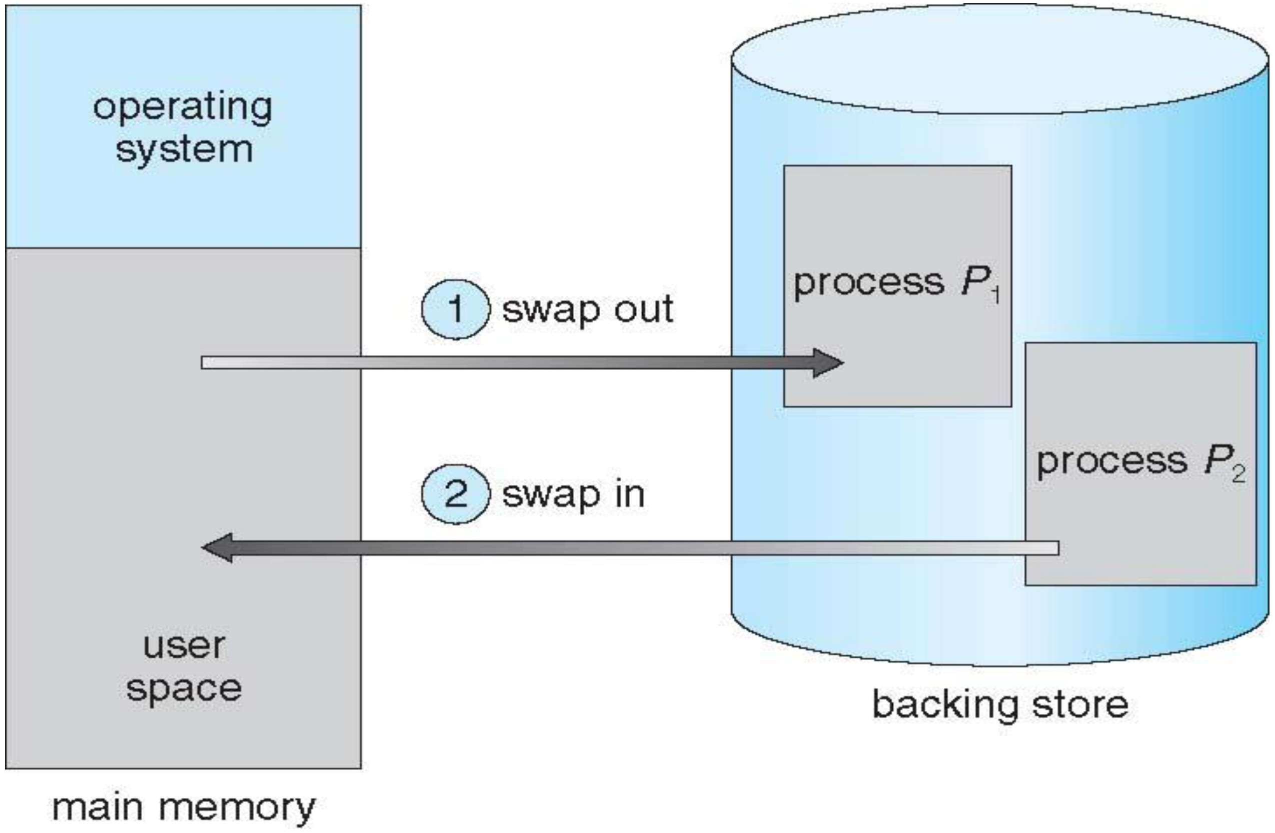


Reference:
https://www.tutorialspoint.com/operating_system/os_memory_management.htm

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
- **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; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Paging

- In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

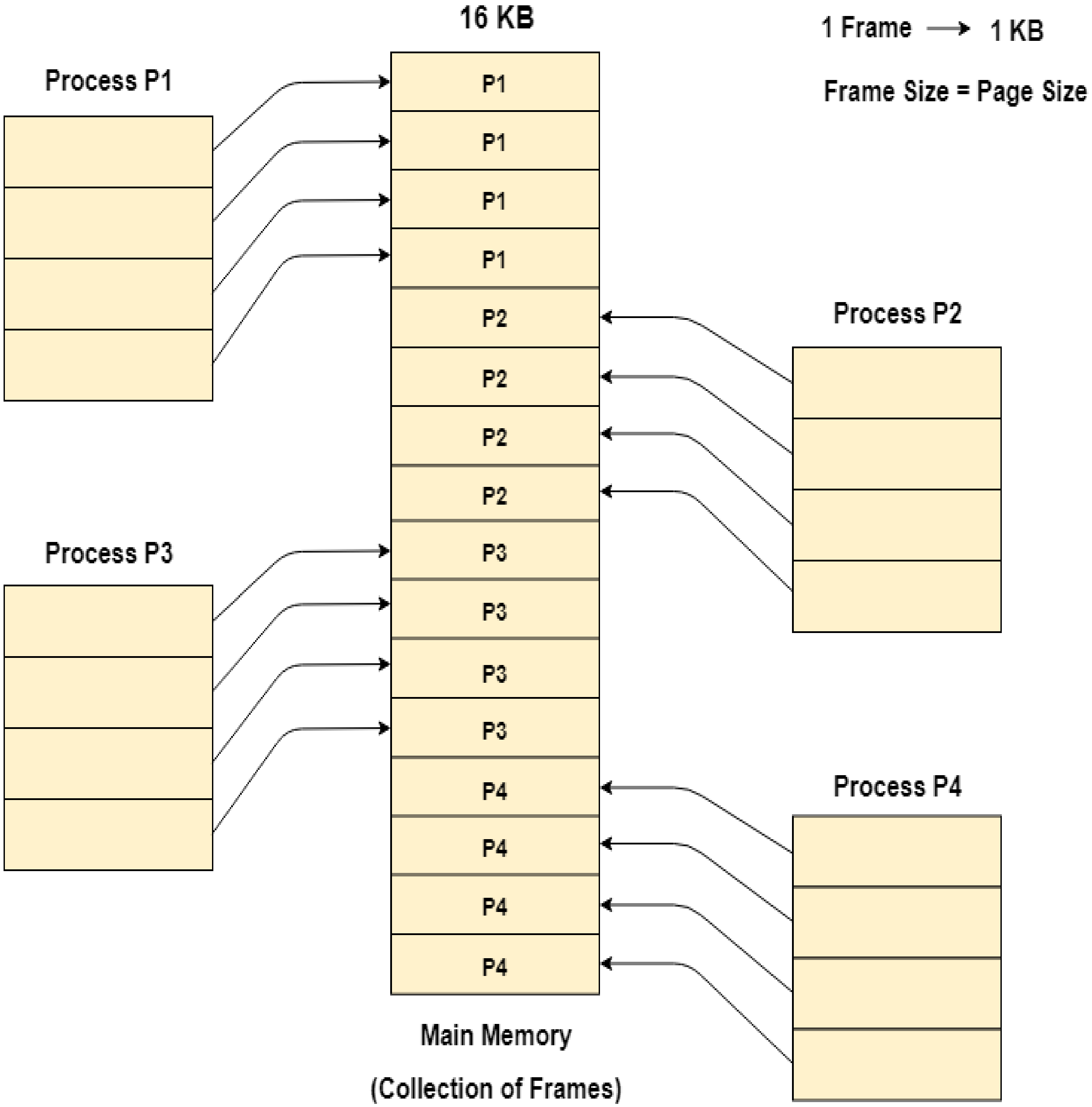
Paging

- Physical address space of a process can be noncontiguous;
  - 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

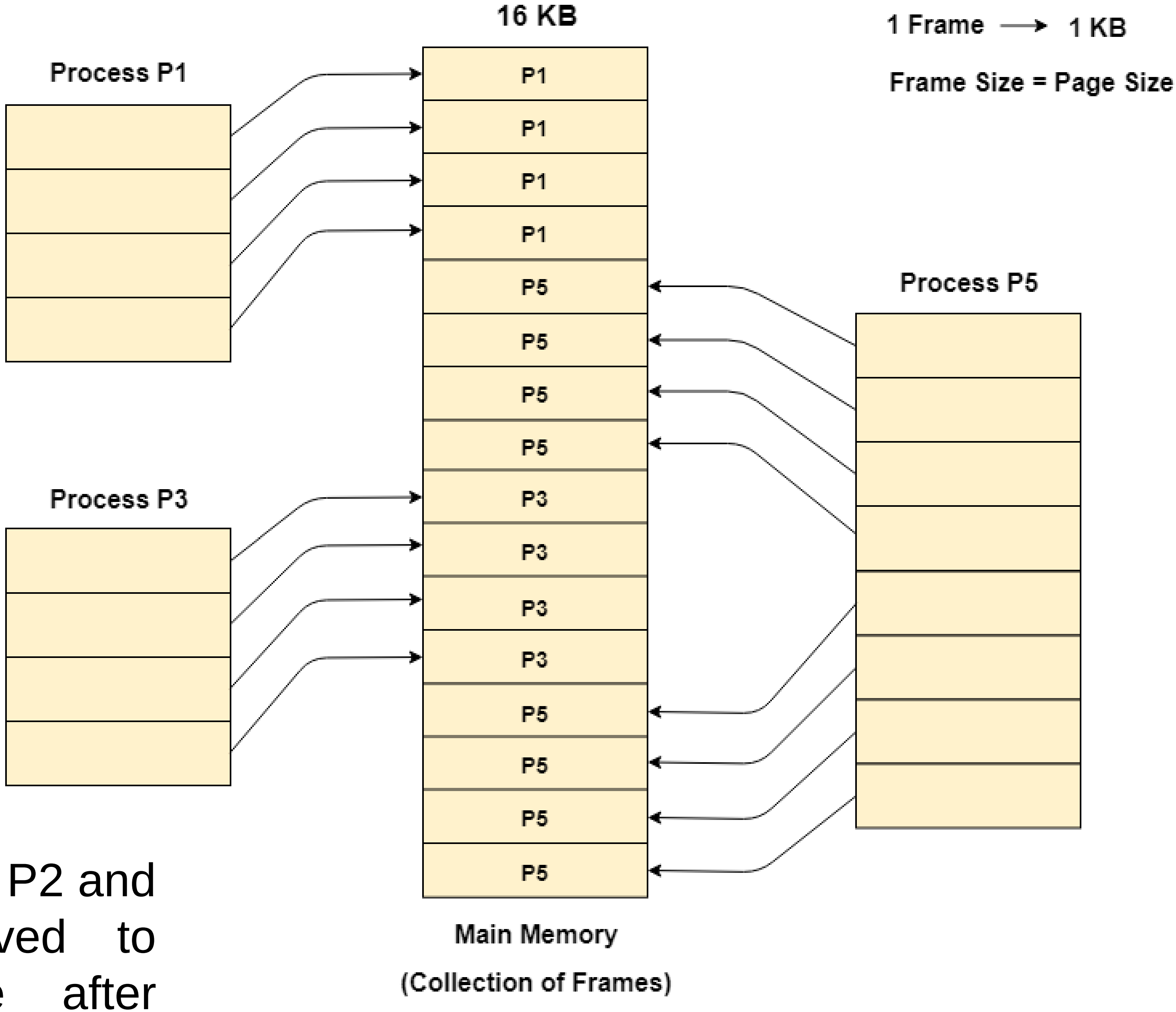
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

Assume main memory size 16 Kb and Frame size is 1 KB

P1, P2, P3 and P4 of 4 KB each



Paging

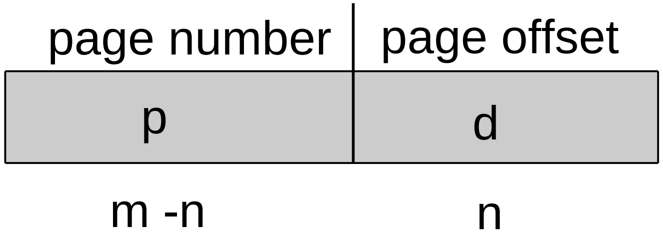


Paging

Consider that, P2 and P4 are moved to waiting state after some time.

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

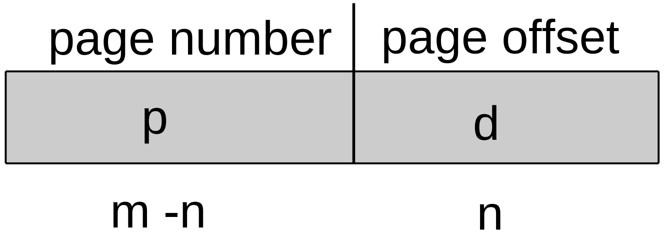
p | d

m -n | n

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

Terminology

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses



- Logical Address = 13 bits
- Physical Address = 12 bits
- Page size = frame size = 1 K words
- Logical Address Space?
- Physical Address Space
- How many pages?
- How many frames?
- What is the value of p and d?

2^13 bytes or words

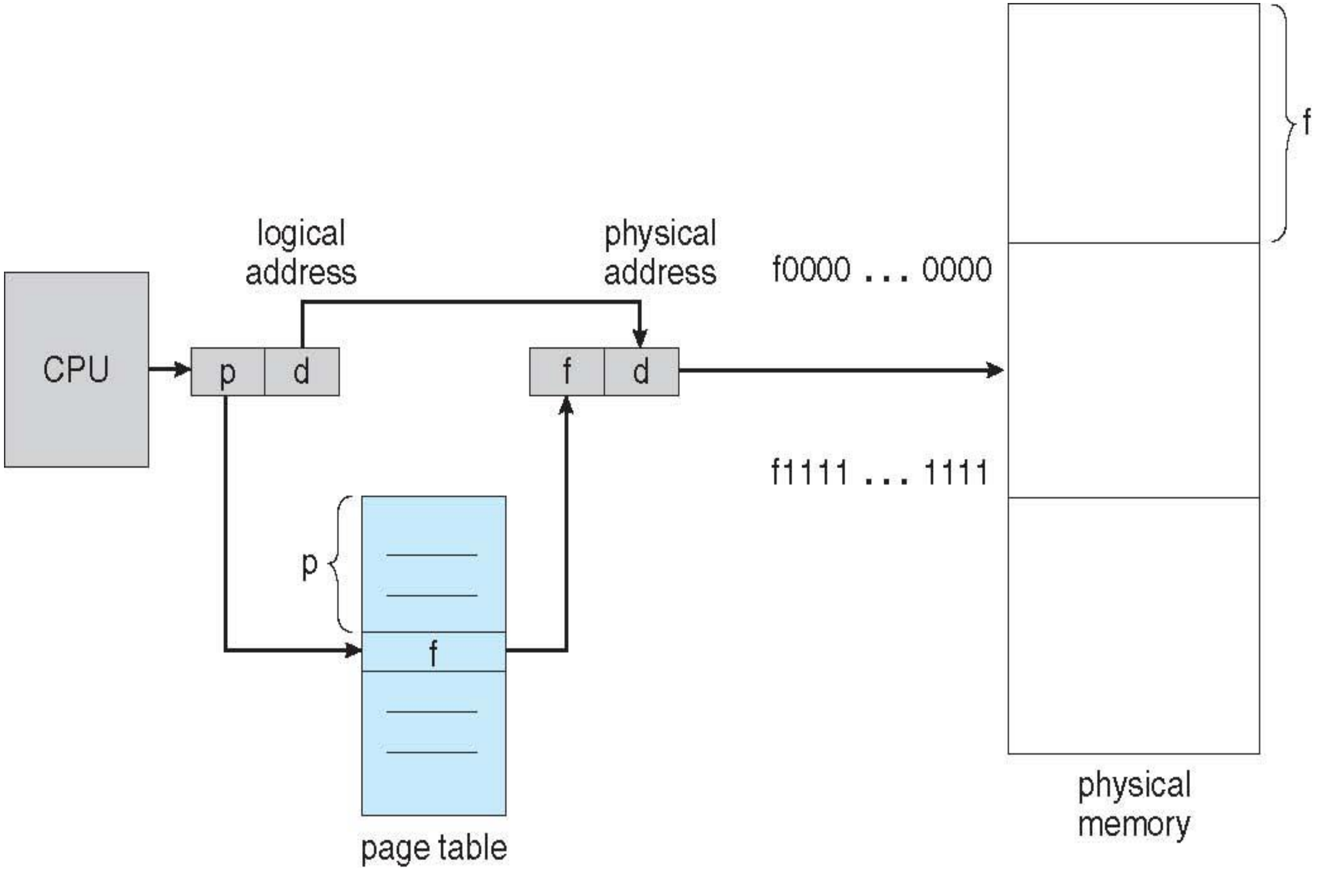
2^12 bytes or words

8

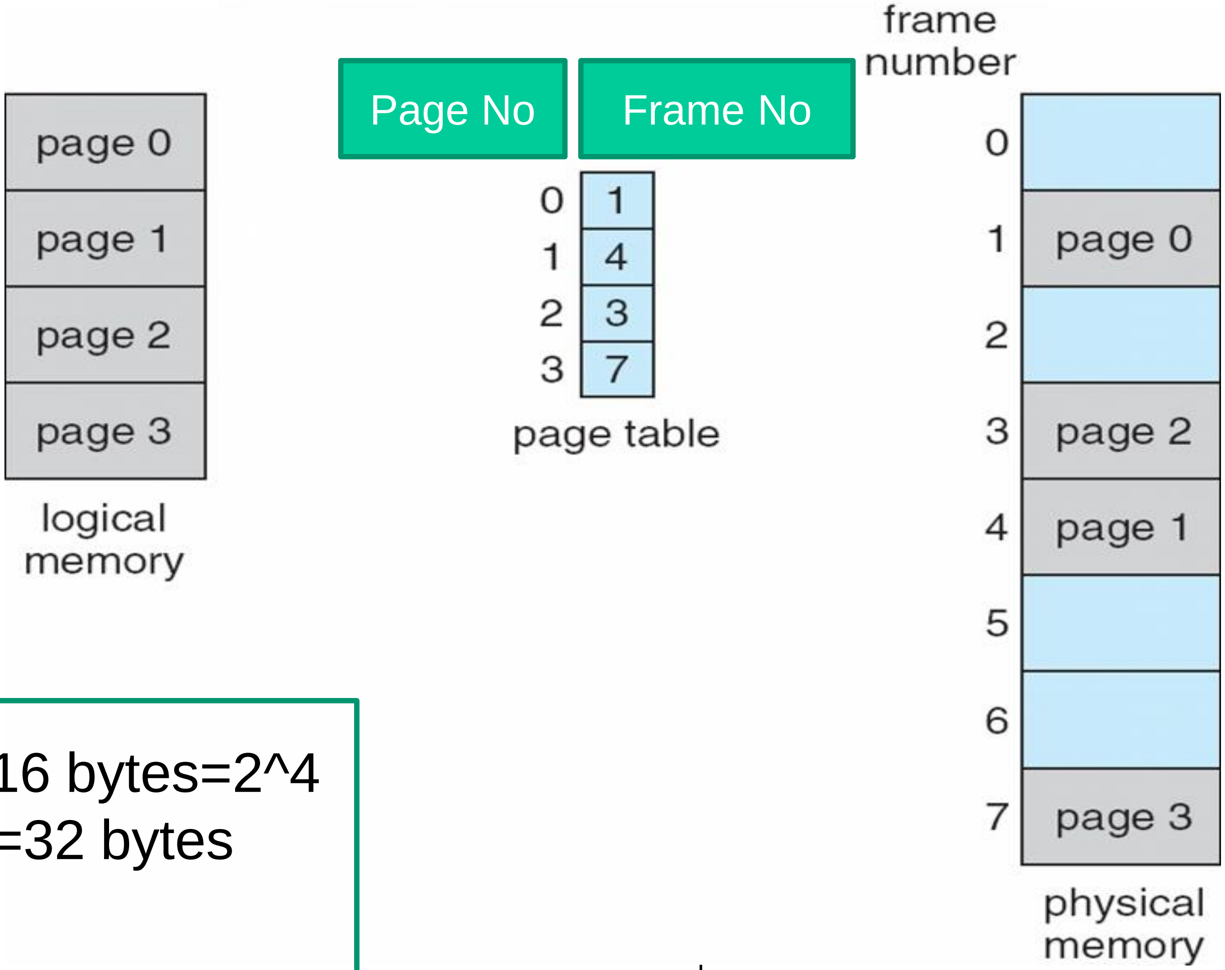
4

3, 10

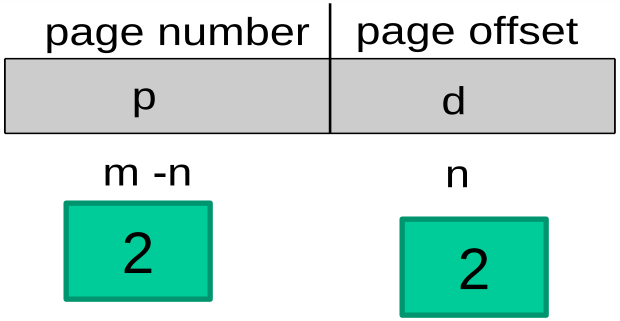
Paging Hardware



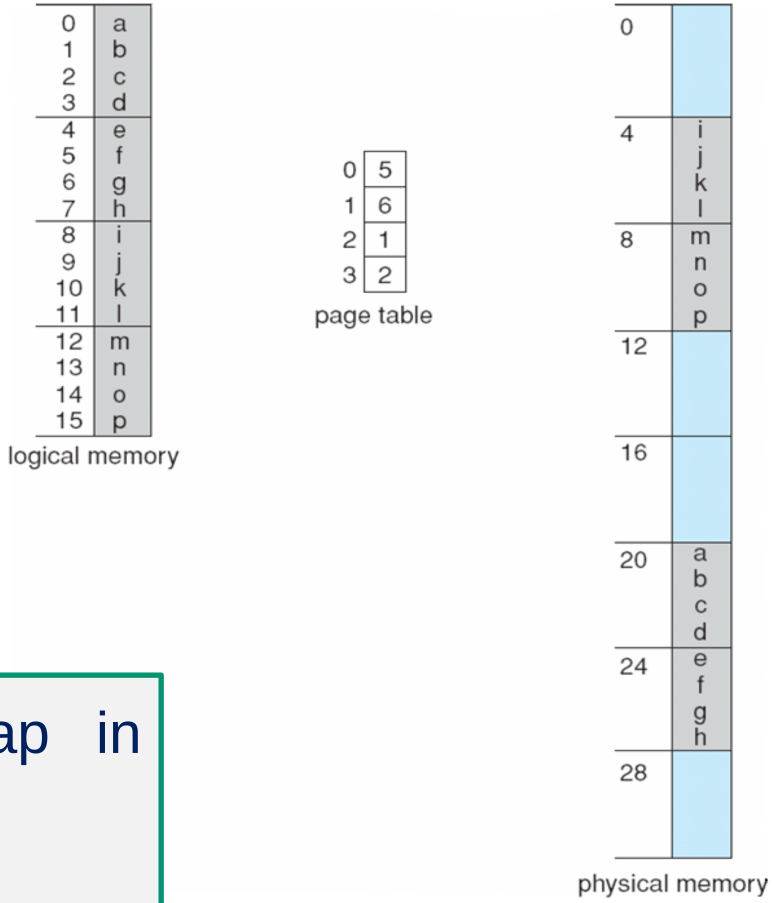
Paging Model of Logical and Physical Memory



Logical address space=16 bytes=2^4
Physical Address space=32 bytes
Page size=4 bytes=2^2
Logical Address? 4 bits
Physical address? 5 bits
No of pages=16/4=4 pages
No. of frames=32/4=8 frames



Paging Example



Logical address 0-> 0000 will map in physical address 20
Logical address 3->physical 23
Logical address 10->physical address 6
Logical address 13-> physical address 9
Logical address 4-> physical address 24

n=2 and *m*=4 32-byte memory and 4-byte pages

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - So small frame sizes desirable?

Q1

- Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

•Number of locations possible with 22 bits = 2^{22} locations
•It is given that the size of one location = 2 bytes

Thus, Size of memory
= 2^{22} x 2 bytes
= 2^{23} bytes
= 8 MB

Q2

- Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

Let ‘n’ number of bits are required. Then, Size of memory = 2^n x 4 bytes.

Since, the given memory has size of 16 GB, so we have-

2^n x 4 bytes = 16 GB

2^n x 4 = 16 G

$2^n \times 2^2 = 2^{34}$

$2^n = 2^{32}$

$\therefore$ n = 32 bits

Q3

- Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.

- Number of pages=$2^{32}/2^{12}=2^{20}$ pages
- Page table size
- = Number of entries in page table x Page table entry size
- =$2^{20}*4$=4 MB

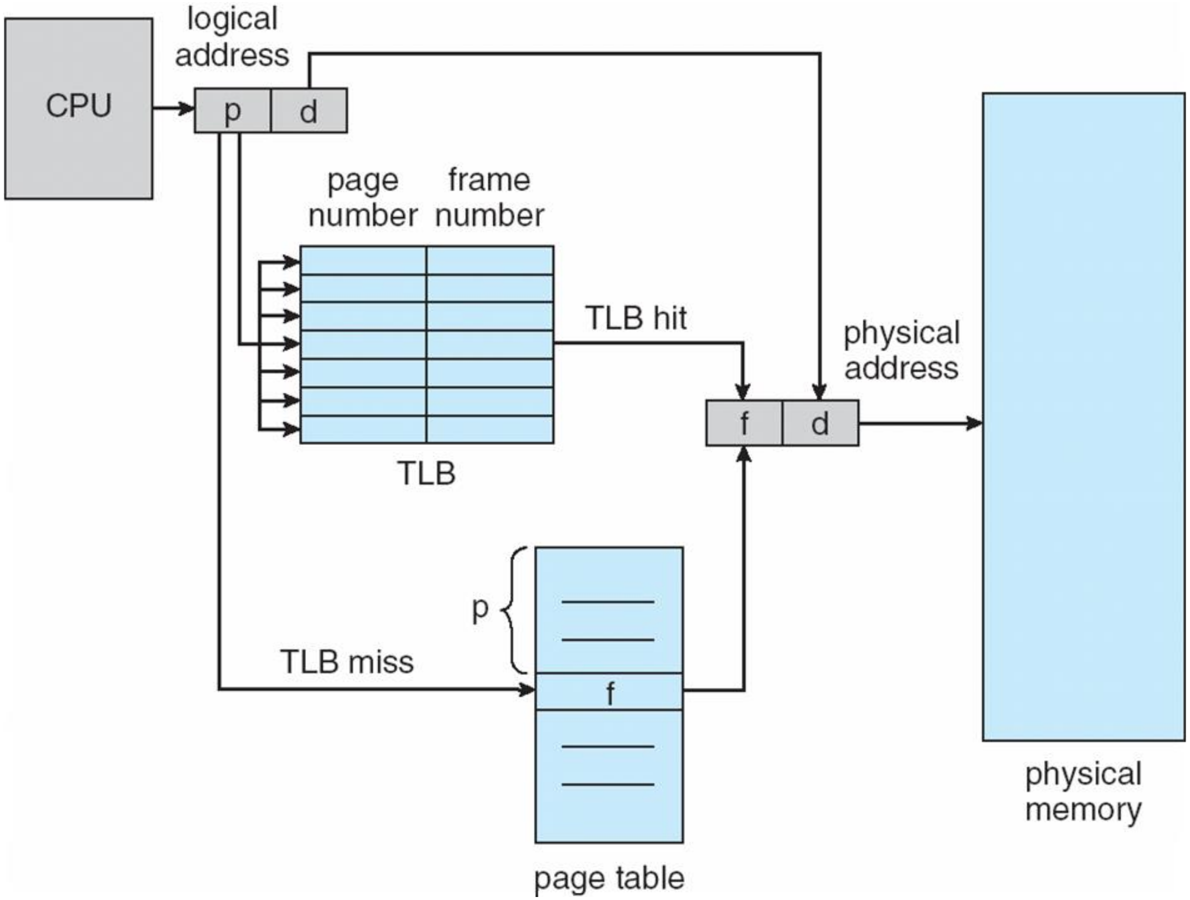
Q4

- Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?
- Number of pages=$2^{32}/2^{12}=2^{20}$
- Number of frames in main memory
- = Size of main memory / Frame size
- = 64 MB / 4 KB
- = 2^{26} B / 2^{12} B
- = 2^{14}
- Thus, Number of bits in frame number = 14 bits
- Page table size
- = Number of entries in page table x Page table entry size
- = Number of entries in page table x Number of bits in frame number
- = 2^{20} x 14 bits
- = 2^{20} x 16 bits (Approximating 14 bits ≈ 16 bits)
- = 2^{20} x 2 bytes
- = 2 MB

Q5

- In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?
- Number of bits in physical address = 30 bits
- Thus, Size of main memory = 2^{30} B = 1 GB
- Number of frames in main memory
- = Size of main memory / Frame size = 1 GB / 4 KB
- = 2^{30} B / 2^{12} B = 2^{18}
- Thus, Number of bits in frame number = 18 bits
- Maximum number of bits that can be used for storing protection and other information
- = Page table entry size – Number of bits in frame number
- = 32 bits – 18 bits
- = 14 bits

Paging Hardware With TLB(translation lookaside buffer)



Effective Access Time =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + 2 x Access time of main memory }

Q1

A paging scheme uses a Translation Lookaside buffer (TLB). A TLB access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page fault?

A.54

B.60

C.65

D. 75

Option C is correct

Q2

A paging scheme uses a Translation Lookaside buffer (TLB). The effective memory access takes 160 ns and a main memory access takes 100 ns. What is the TLB access time (in ns) if the TLB hit ratio is 60% and there is no page fault?

54

60

20

75

Option C is correct

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

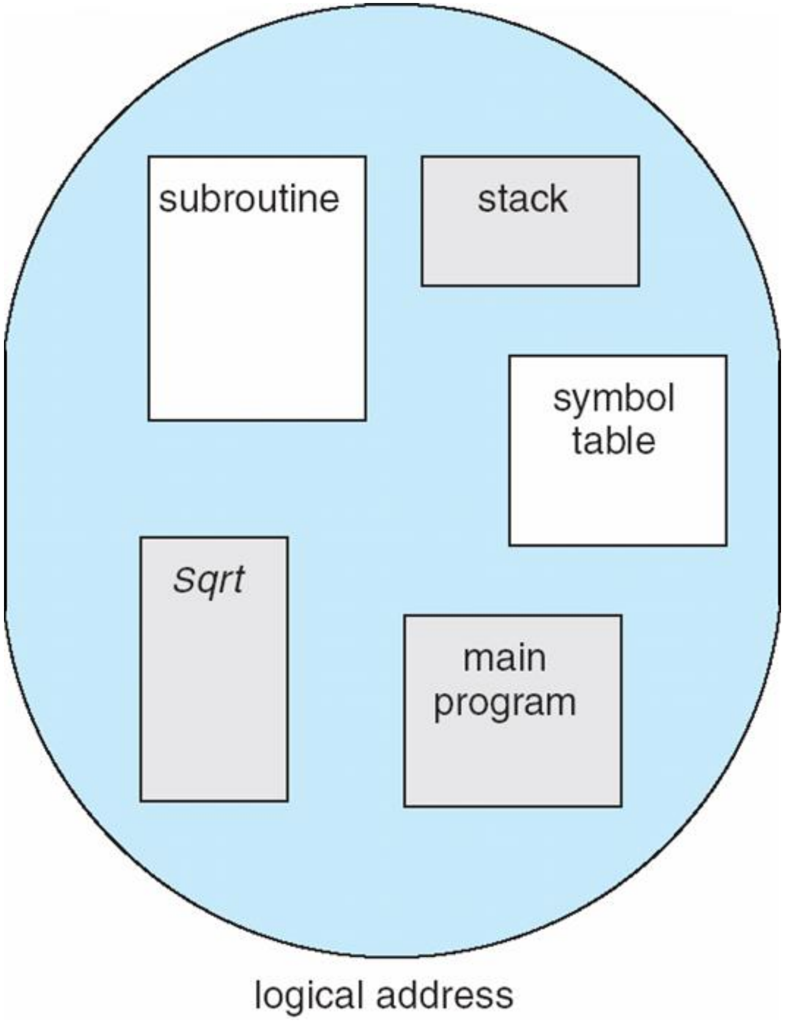
  symbol table

  arrays

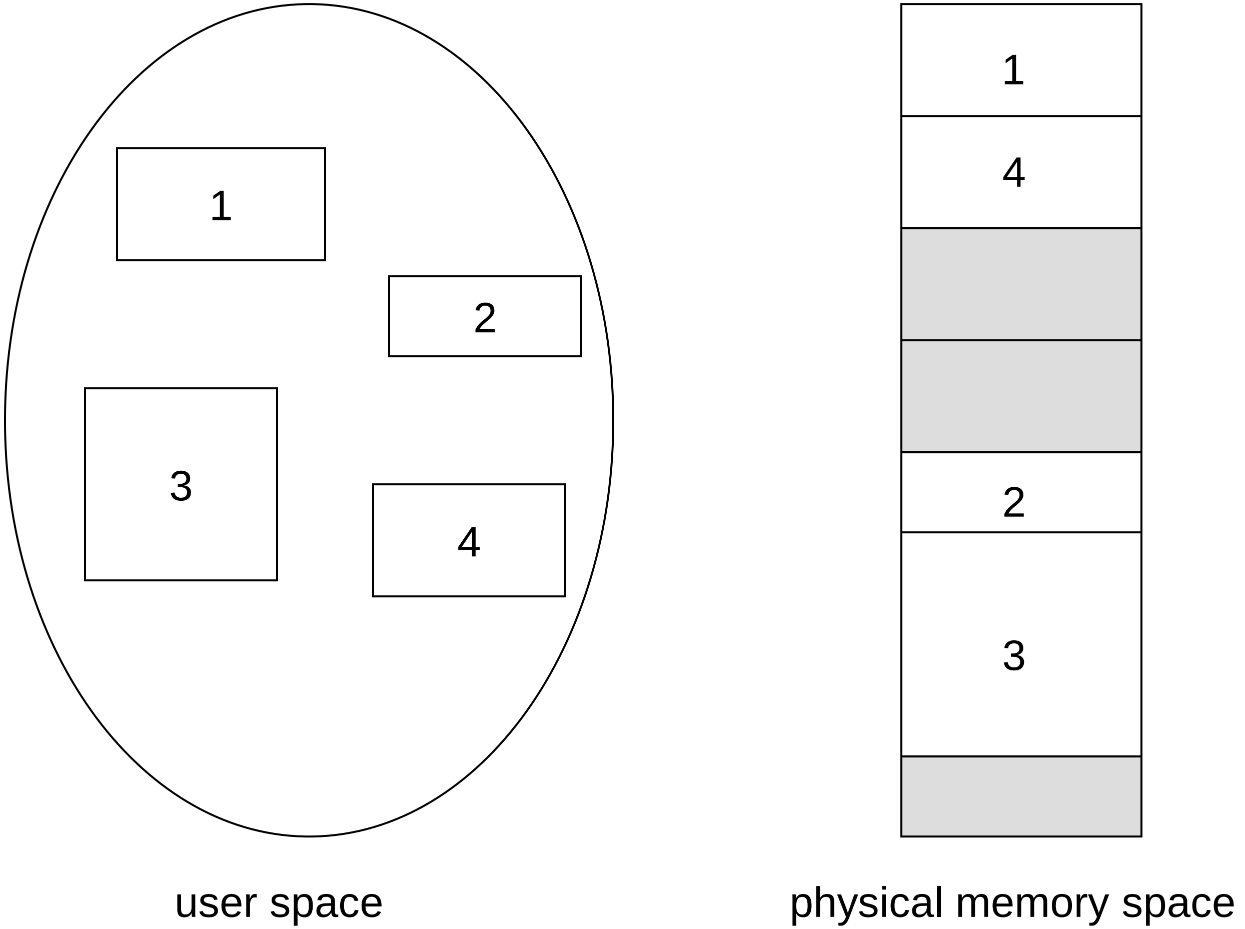
Why Segmentation is required?

- Operating system doesn't care about the User's view of the process.
- It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory.
- It decreases the efficiency of the system.

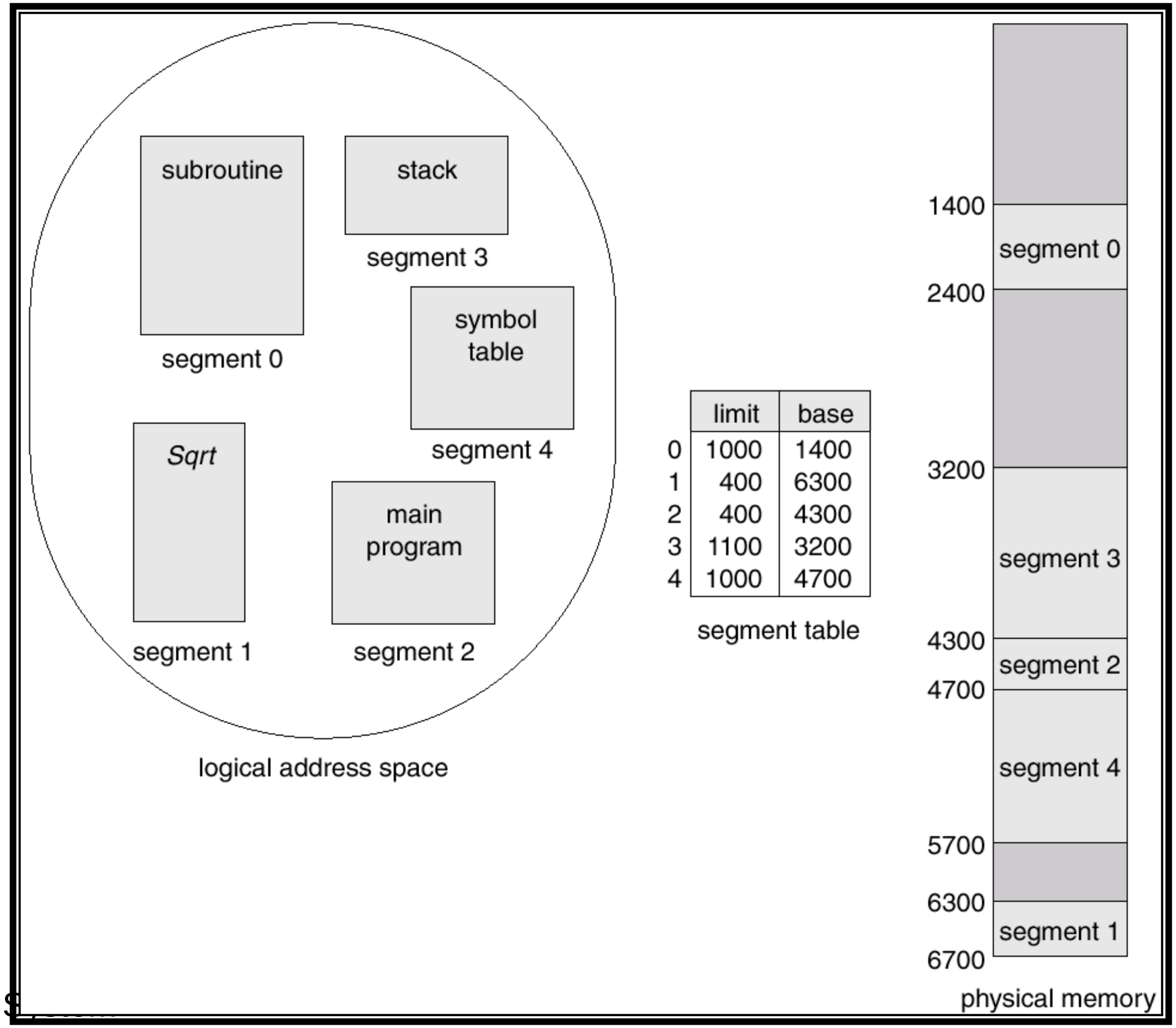
User’s View of a Program



Logical View of Segmentation



Segment Table Example



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

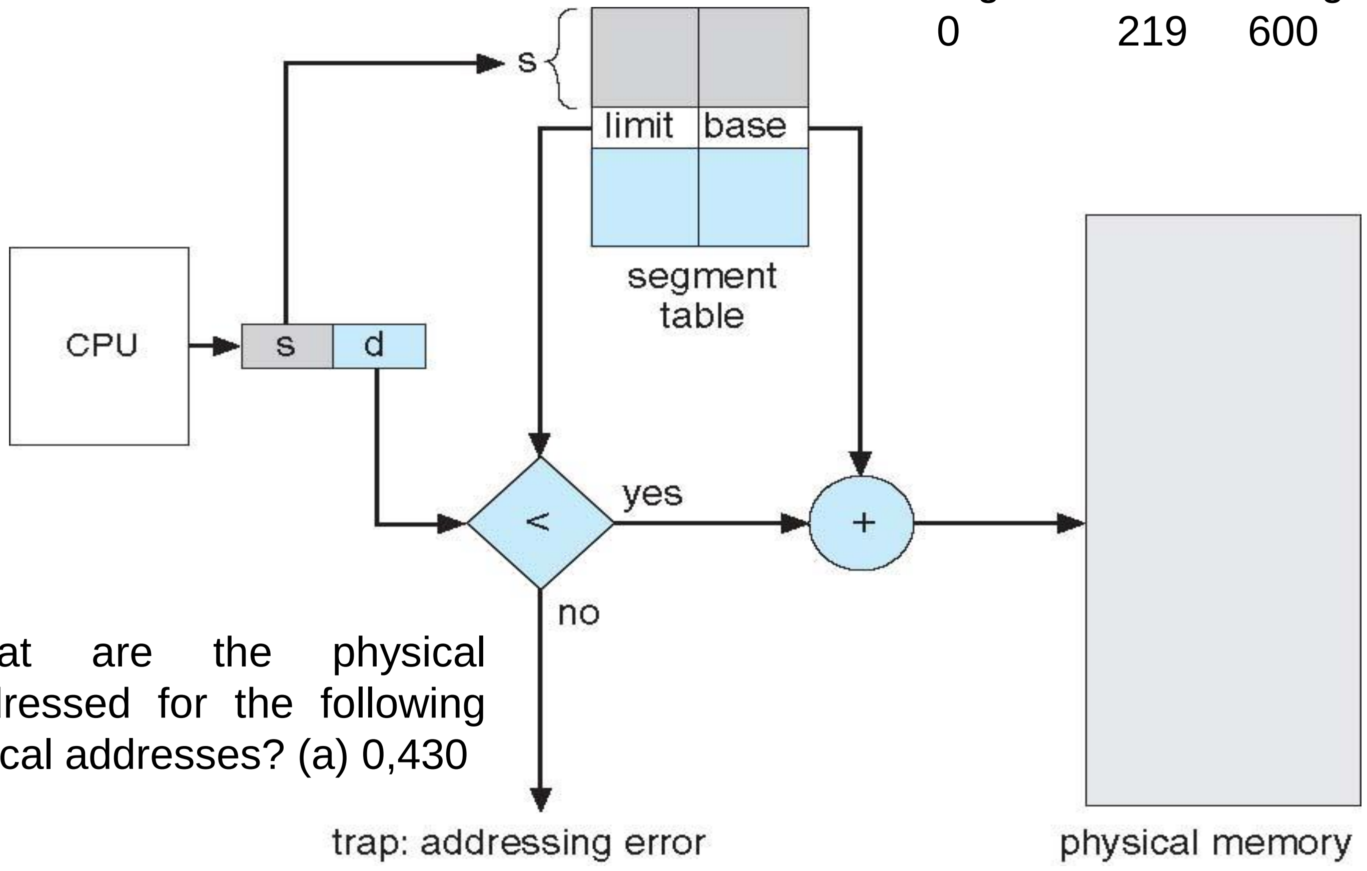
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

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

Segmentation Architecture (Cont.)

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem

Segment	Base	Length
0	219	600



What are the physical addressed for the following logical addresses? (a) 0,430

Question

- Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

- What are the physical addressed for the following logical addresses? (a) 0,430 (b) 1,10 (c) 2,500 (d) 3,400 (e) 4,112

- (a) 219 + 430 = 649
- (b) 2300 + 10 = 2310
- (c) illegal reference; traps to operating system
- (d) 1327 + 400 = 1727
- (e) illegal reference; traps to operating system

Advantages of Segmentation

- No internal fragmentation
- Average Segment Size is larger than the actual page size.
- Less overhead
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as compare to the page table in paging.

Disadvantages

- It can have external fragmentation.
- it is difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithms.

Refrences

- https://www.gatevidyalay.com/paging-formulas-practice-problems/