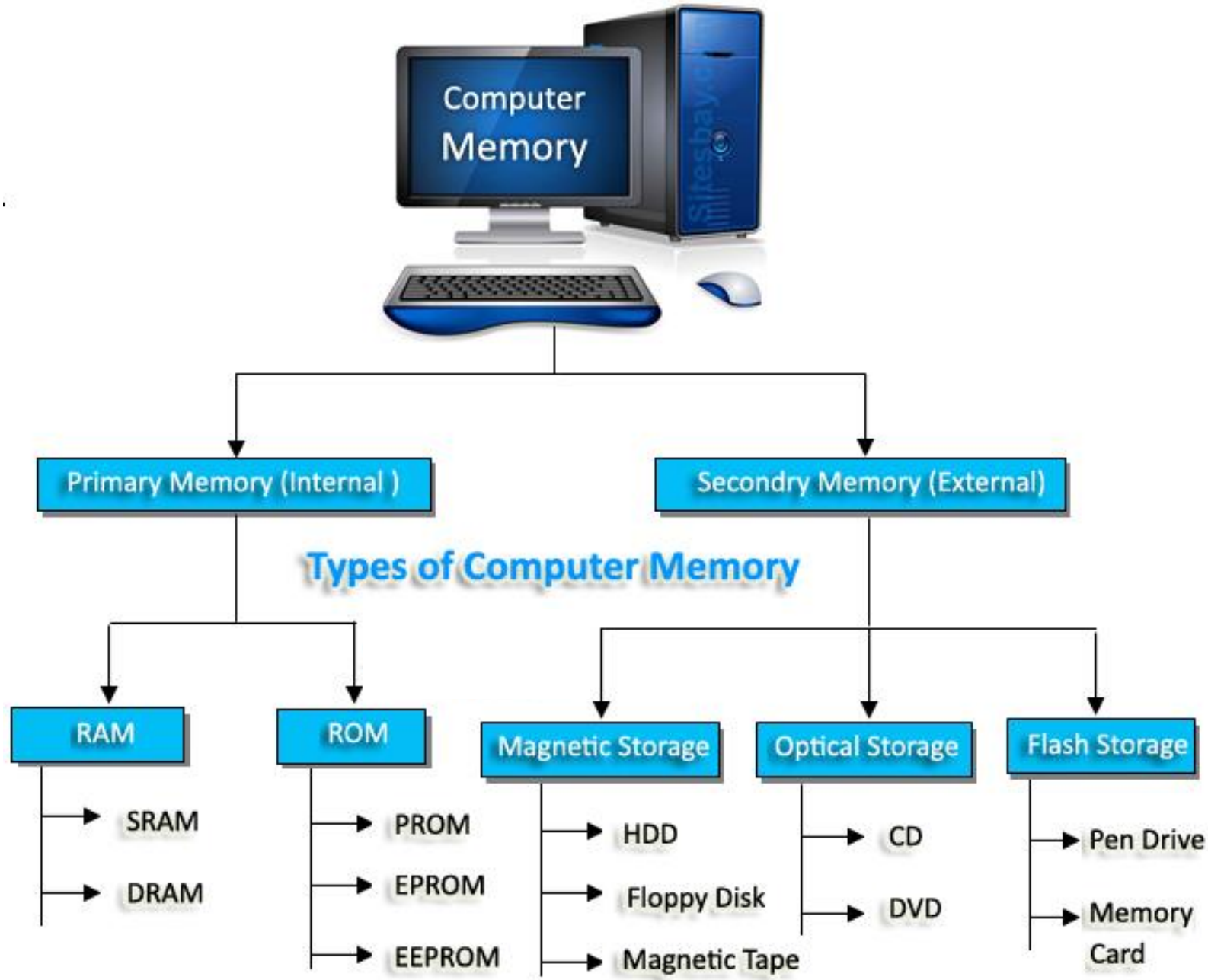
Chapter 5:
Memory Management

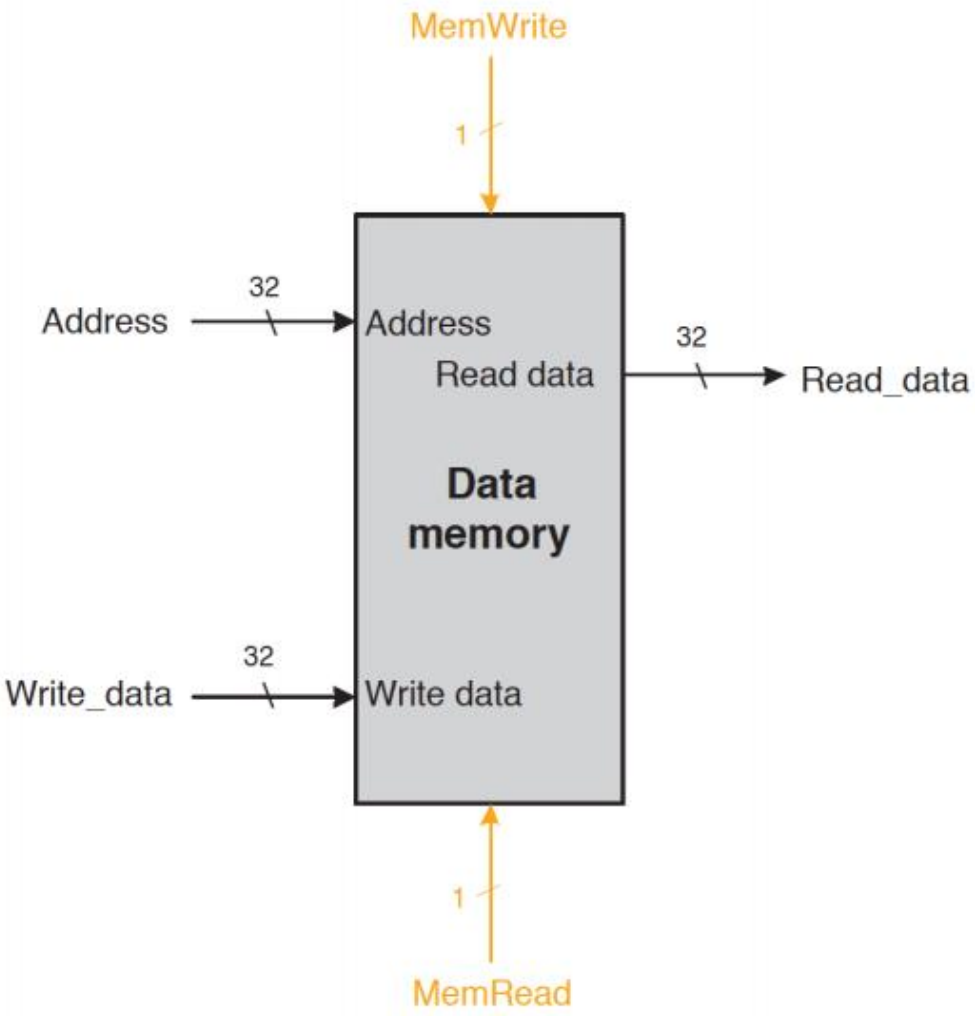

Computer Memory
Primary Memory (Internal)
Secondry Memory (External)
Types of Computer Memory
RAM
ROM
Magnetic Storage
Optical Storage
Flash Storage
SRAM
DRAM
PROM
EPROM
EEPROM
HDD
Floppy Disk
Magnetic Tape
CD
DVD
Pen Drive
Memory Card

Chapter 5: Memory Management

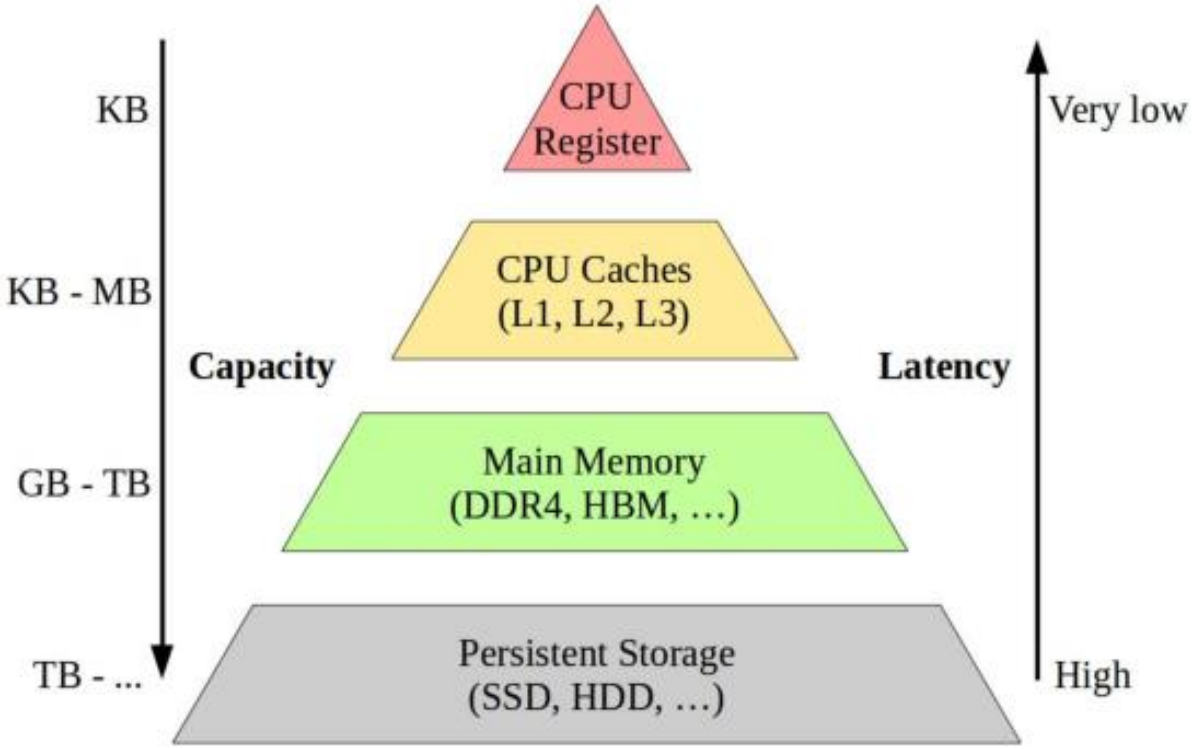
- Background
- Swapping
- Contiguous Memory Allocation
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging

Background

- Memory unit only sees a stream of
  - addresses + read requests, or
  - addresses + data and write requests



Background



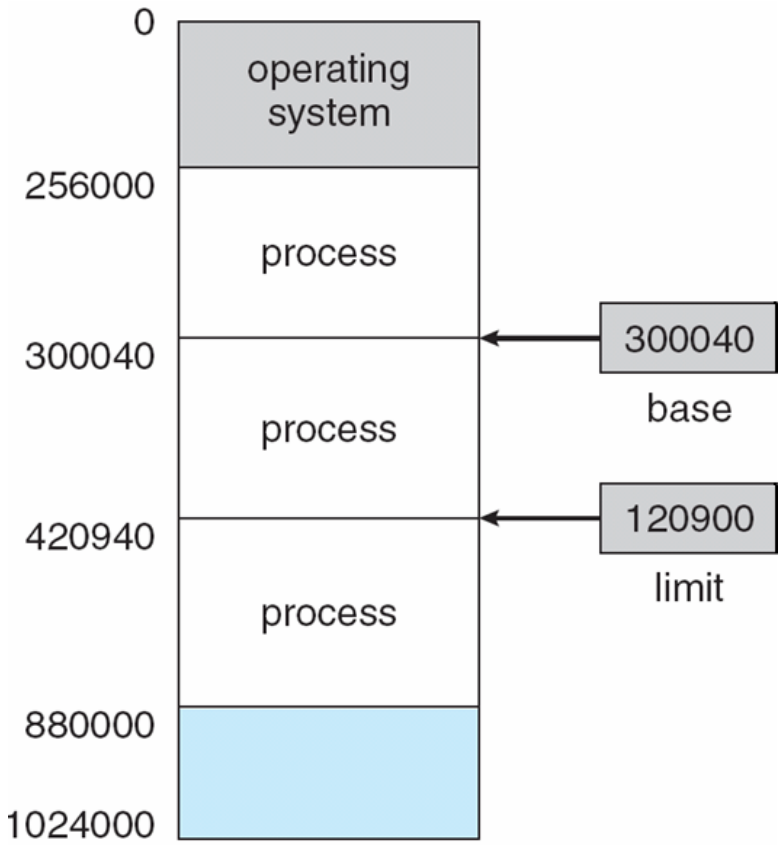
- Register access is faster than memory access:
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles

Background

- To speed up memory access **cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

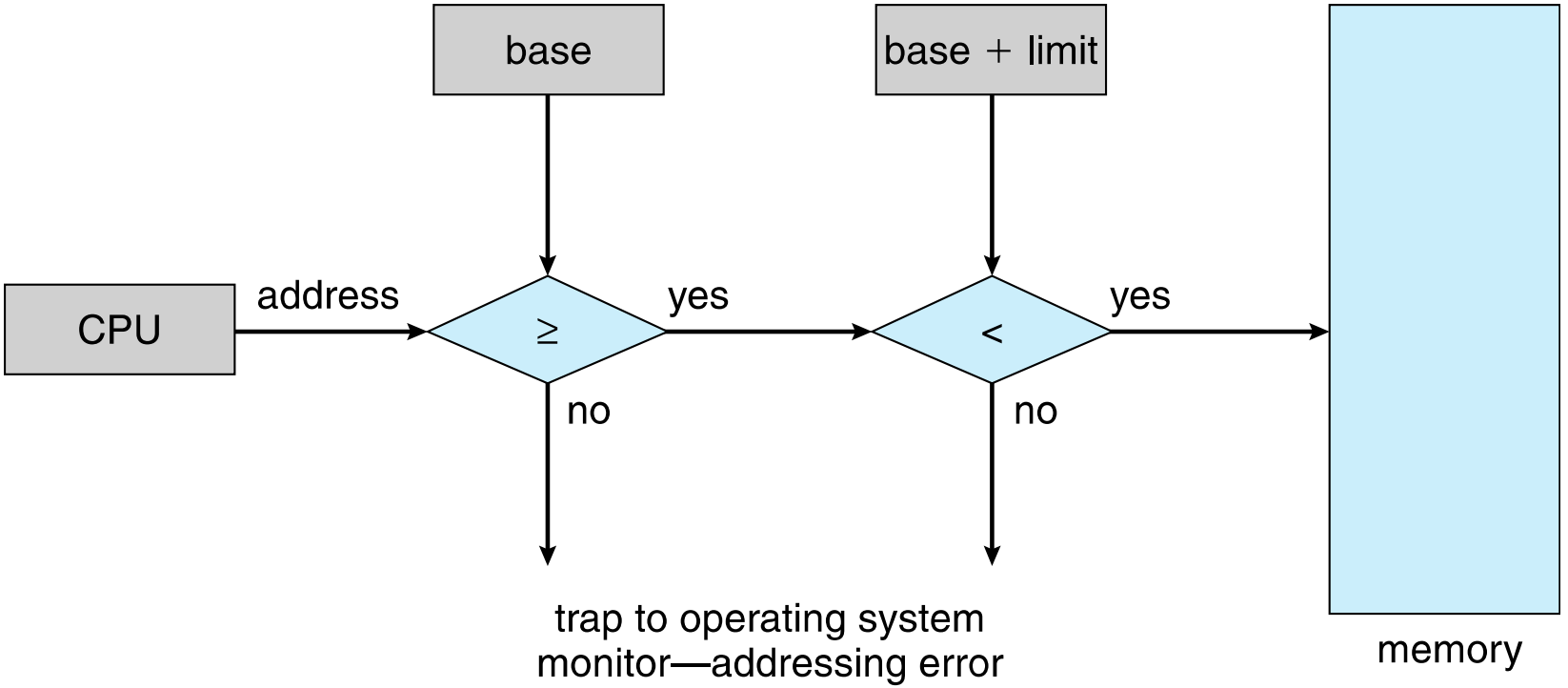
Address Space Protection: Base and Limit Registers

- Memory protection needed in multi-process systems. Each process has a separate memory space.
- A pair of **base** and **limit registers** define the logical address space

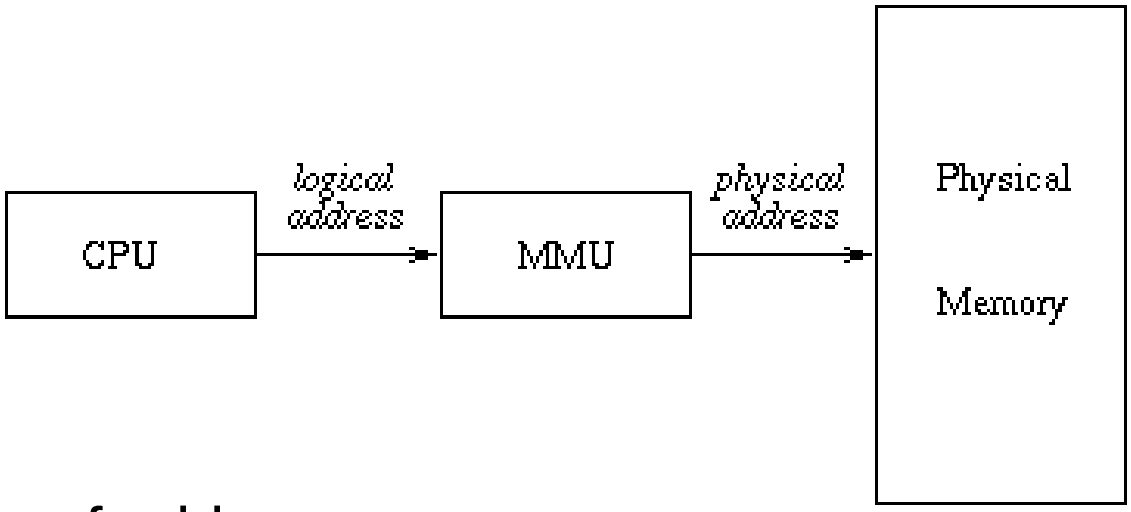


Hardware Address Protection

CPU must check every memory access generated in user mode to be sure it is between base and limit for that user

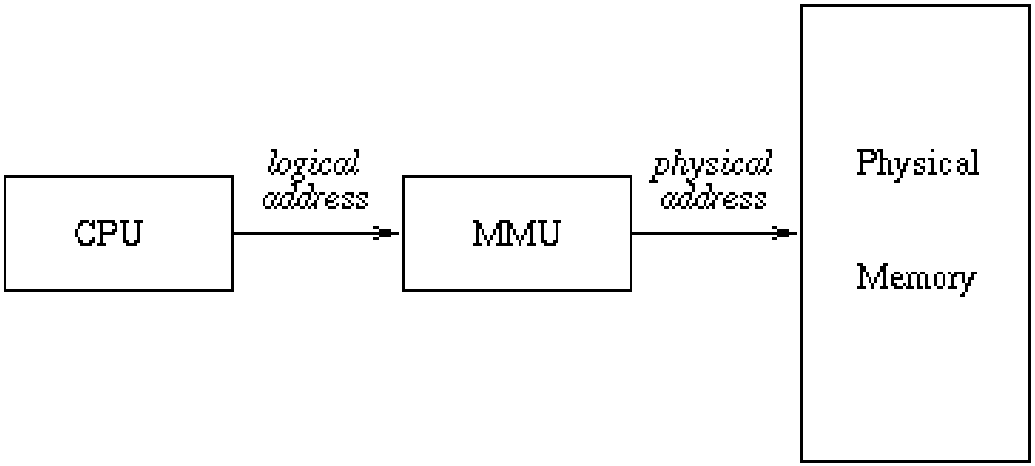


Logical vs. Physical Address Space



- Two types of addresses
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses corresponding to the logical addresses

Memory-Management Unit (MMU)



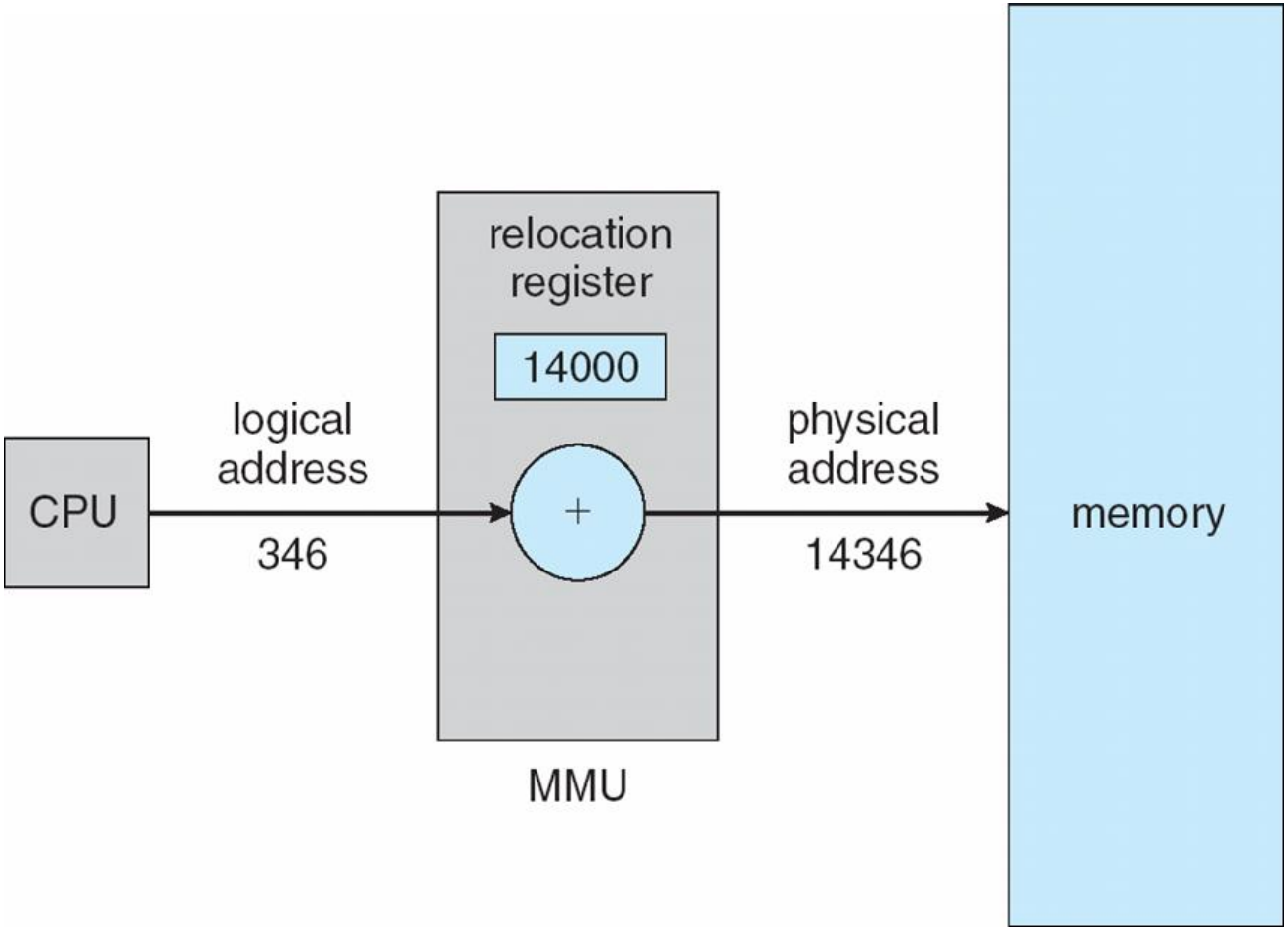
- The user program deals with logical addresses; it never sees the real physical addresses
- The run-time mapping from virtual to physical addresses is done by a hardware called the MMU.
- Many methods possible for mapping, covered in the rest of this chapter

Memory-Management Unit (MMU)

- MMU is a computer hardware unit that examines all memory references on the memory bus, translating these requests, known as virtual memory addresses, into physical addresses in main memory.
- Base register now called **relocation register**
  - E.g. value of the relocation register = R
  - The user program generates only logical addresses and thinks that the process runs in locations 0 to MAX.
    - Range of the logical address generated by a user process 0 to MAX
    - → Physical address space for that process = R+0 to R+MAX

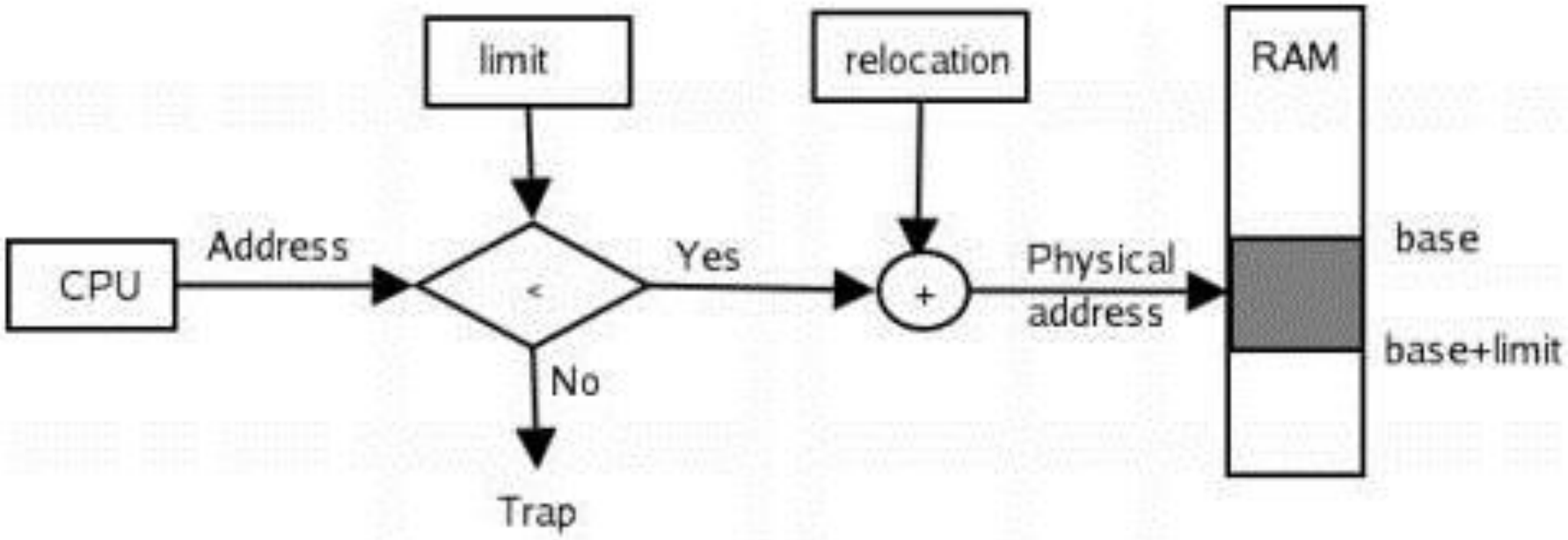
Memory-Management Unit (MMU)

Relocation register

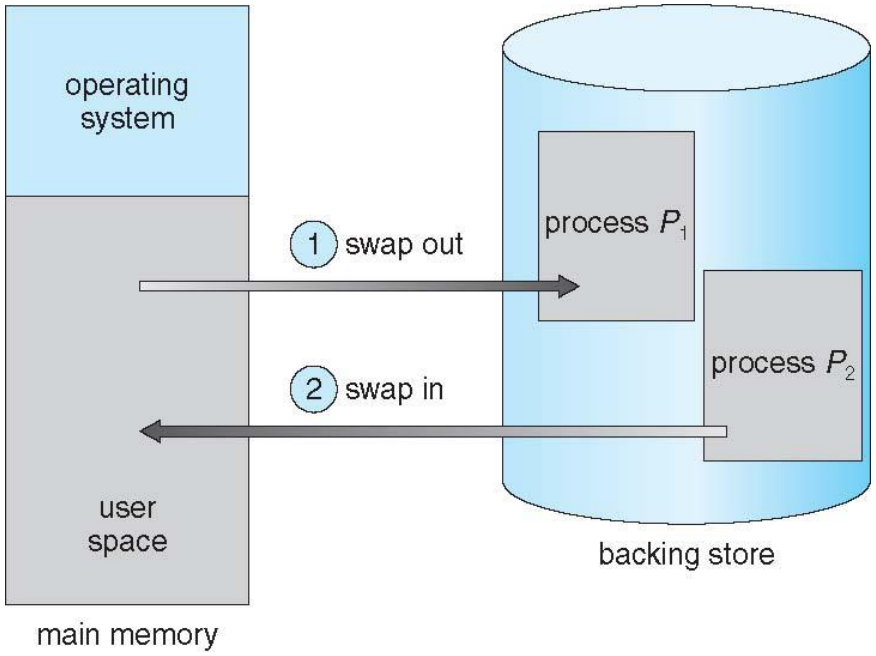


Memory-Management Unit (MMU)

Relocation register



Swapping



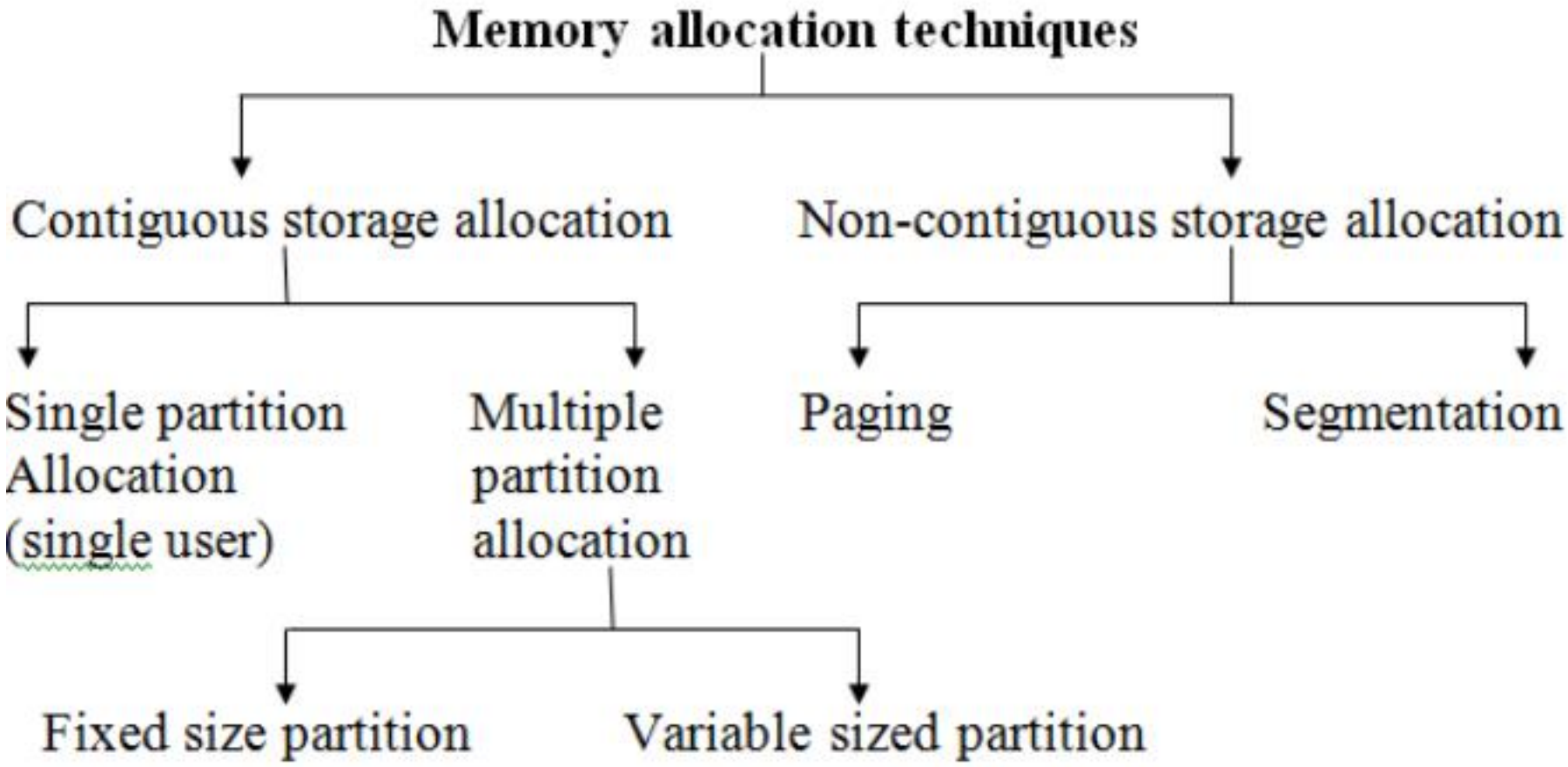
- A process must be in memory to be executed.
- However, it can be **swapped** temporarily out of memory to a backing store (e.g. hard disk), and then brought back into memory for continued execution
  - Swapping makes it possible for the total physical memory space of processes can exceed physical memory, thus increase the multiprogramming in a system

Swapping

- The context-switch time in such a swapping system is fairly high. The major part of the swap time is transfer time.
  - Assume that the user process is 100 MB in size and the backing store is a standard hard disk with a transfer rate of 50 MB/second.
  - The actual transfer of the 100-MB process to or from main memory takes

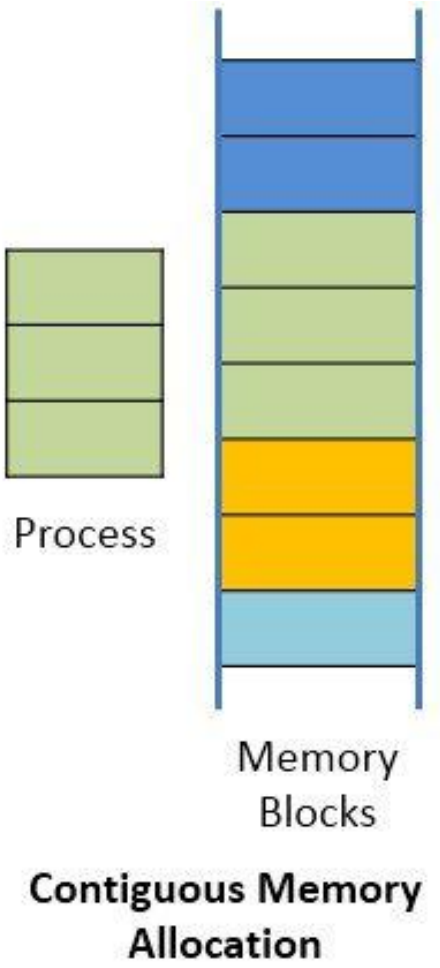
    100 MB/50 MB per second = 2 seconds
  - Since we must swap both out and in, the total swap time is about 4 seconds.

- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped


Memory allocation techniques
Contiguous storage allocation
Non-contiguous storage allocation
Single partition Allocation (single user)
Multiple partition allocation
Paging
Segmentation
Fixed size partition
Variable sized partition

Contiguous Allocation

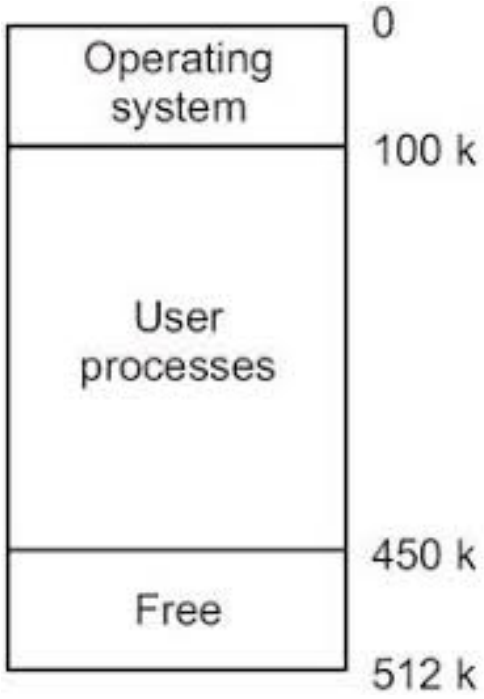
- Contiguous allocation is one early method
- Contiguous storage allocation implies that a program's data and instructions are assured to occupy a single contiguous memory area.



Contiguous Memory Allocation

Single-Partition Allocation

- The simplest possible memory management scheme is to run just one program at a time.
- In this scheme, the memory is divided into two parts.
  - One part holds the OS and
  - the remaining holds the user processes which are loaded and executed one at a time.



- When user process completes its task, it is taken out of main memory and another requesting process is brought into the memory by the OS.

Single-Partition Allocation

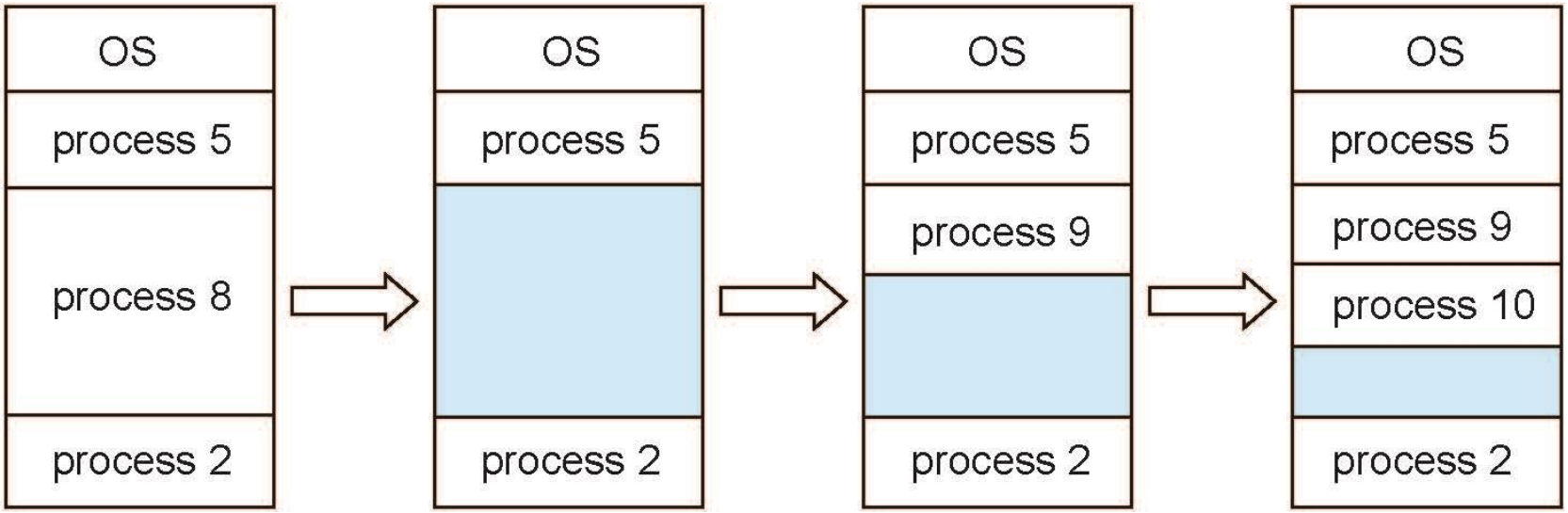
- **Advantages:**
  - It is simple.
  - It does not require much understanding of the system.

- **Disadvantages:**
  - Poor utilization of memory.
  - Poor utilization of processors.

Multiple-Partition Allocation

- Divide the memory into a set of partitions, each partition will hold one process
  - Degree of multiprogramming limited by number of partitions
  - When a partition is free, a process is selected from the input queue and is loaded into the free partition.
  - When the process terminates, the partition becomes available for another process.

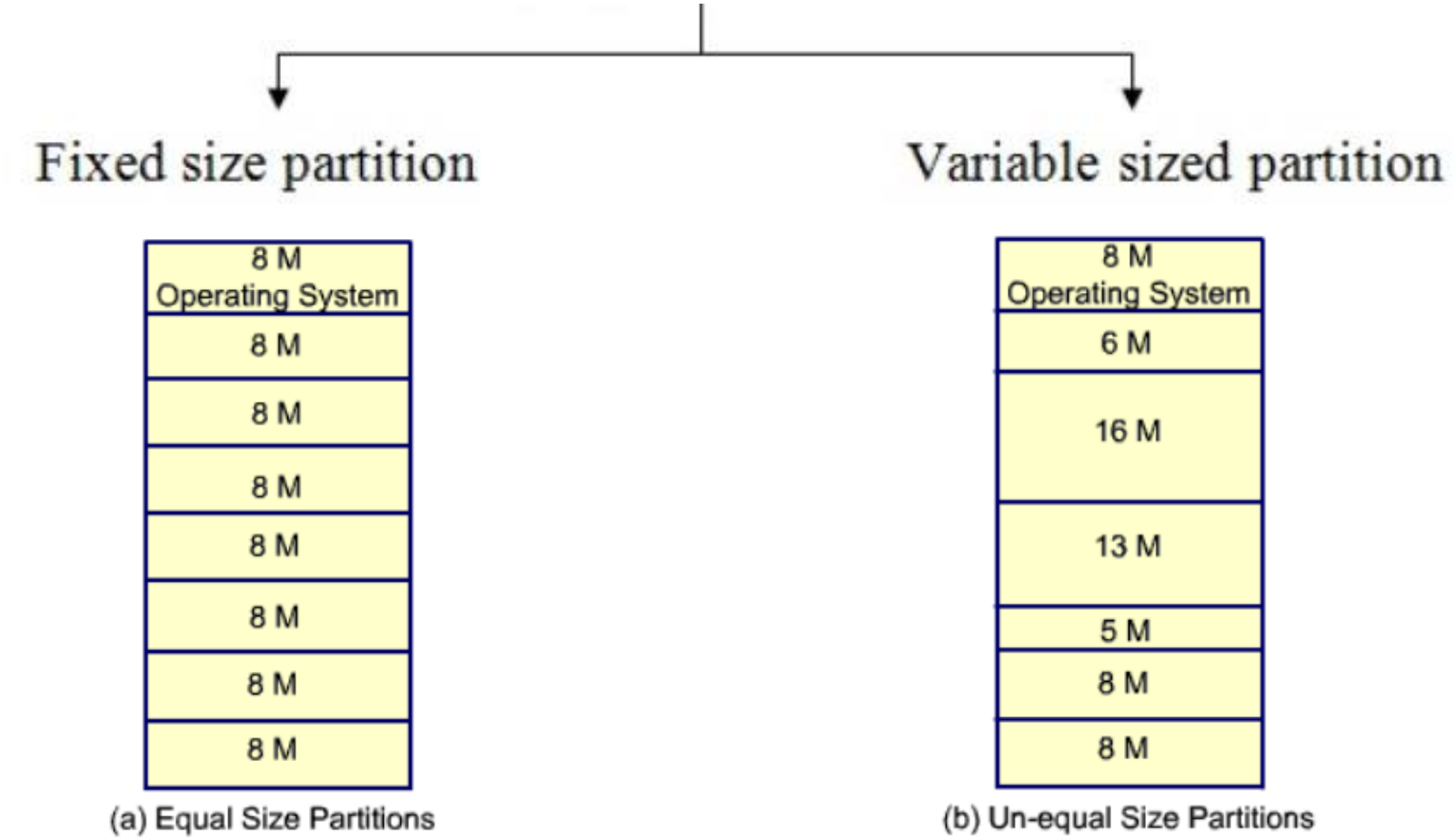
Multiple-Partition Allocation



- When a process arrives, it will be placed into a hole that is large enough to accommodate it
- Process exiting frees its partition, adjacent free partitions are combined
- OS maintains information about:

  a) allocated partitions b) free partitions (hole)

Multiple-Partition Allocation



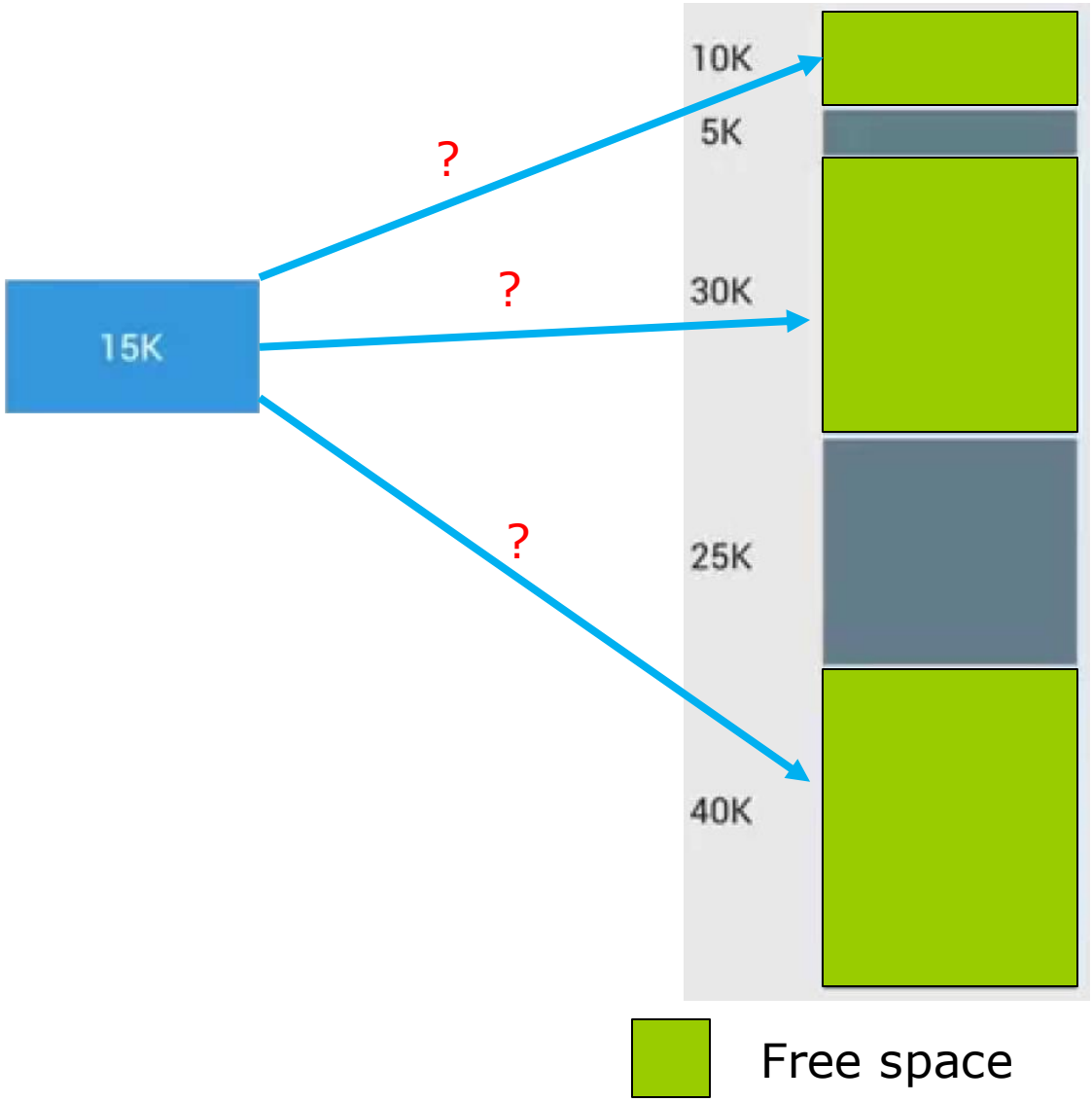
- Partitions are same size
- Was originally used by the IBM OS/360 operating system but is no longer in use

- Allocation is based on the need of the processes

Fixed-size partitions

Variable Size Partition: Dynamic Storage-Allocation

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



Variable Size Partition: Dynamic Storage-Allocation

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

- **First-fit**: Allocate the ***first*** hole that is big enough

  advantage - faster

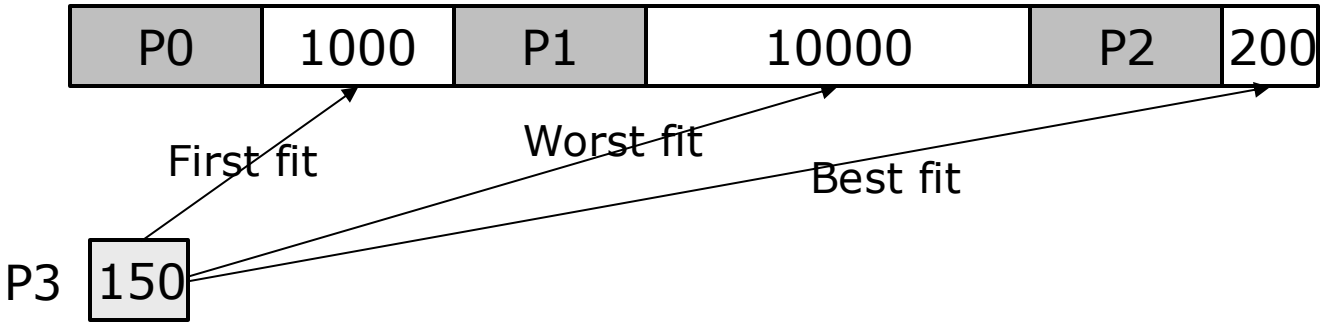
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size

  advantage – better storage utilization

- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list

Dynamic Storage-Allocation Problem

Place a process of size 150 using first, best and worst fits.



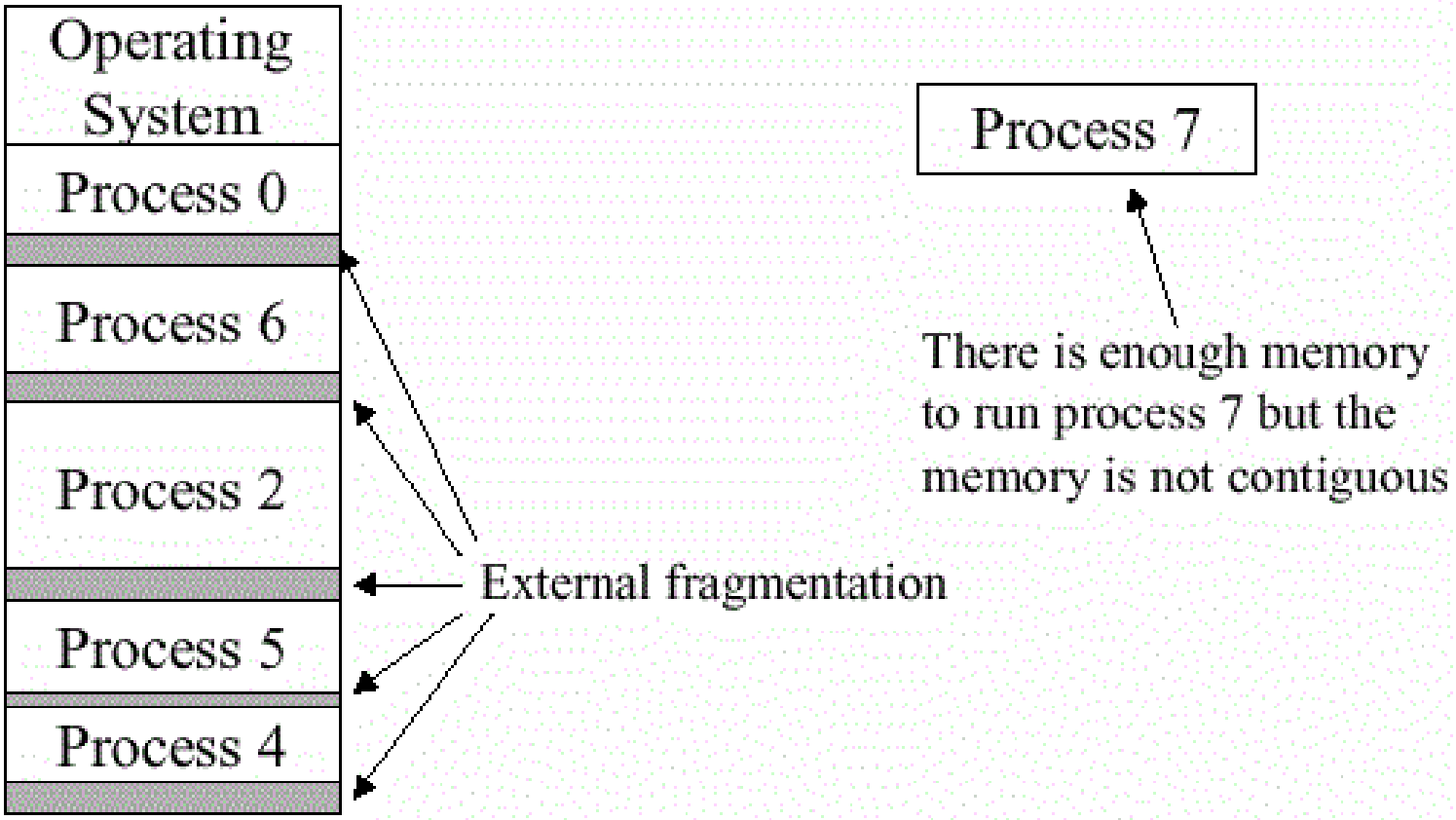
Both the first-fit and best-fit strategies for memory allocation suffer from **external fragmentation**.

Fragmentation

1. External Fragmentation
2. Internal fragmentation

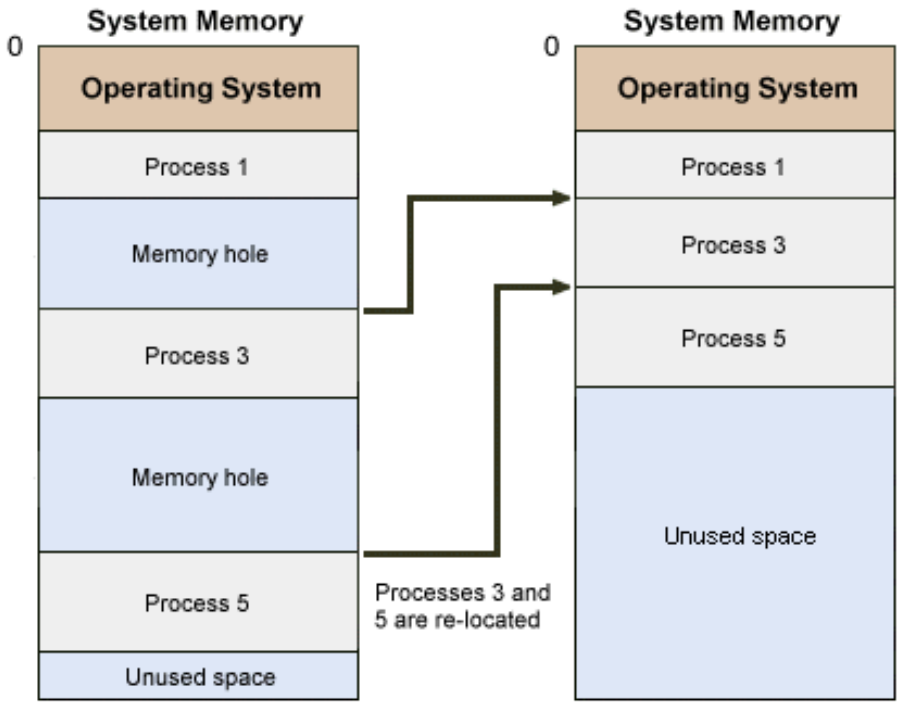
External Fragmentation

- **Total memory space exists to satisfy a request, but it is not contiguous, so it cannot be used**
- Memory is fragmented into a large number of small holes



Variable sized partitions lead to external fragmentation

Compaction

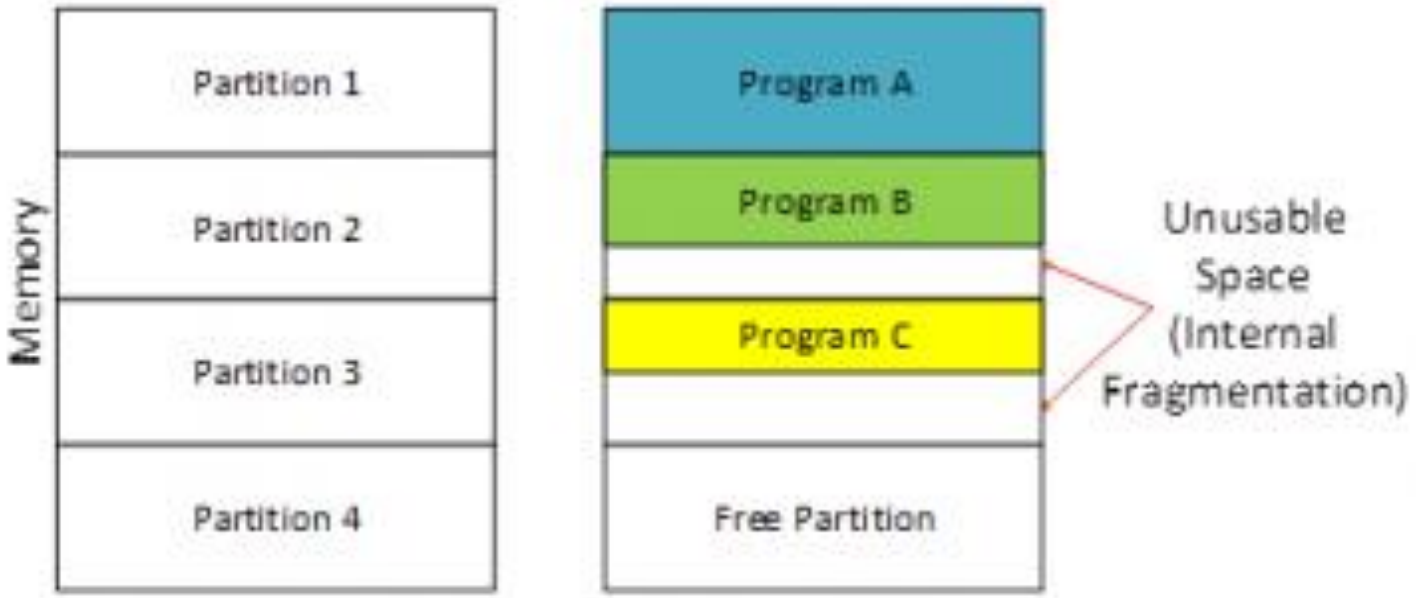


A way to reduce external fragmentation

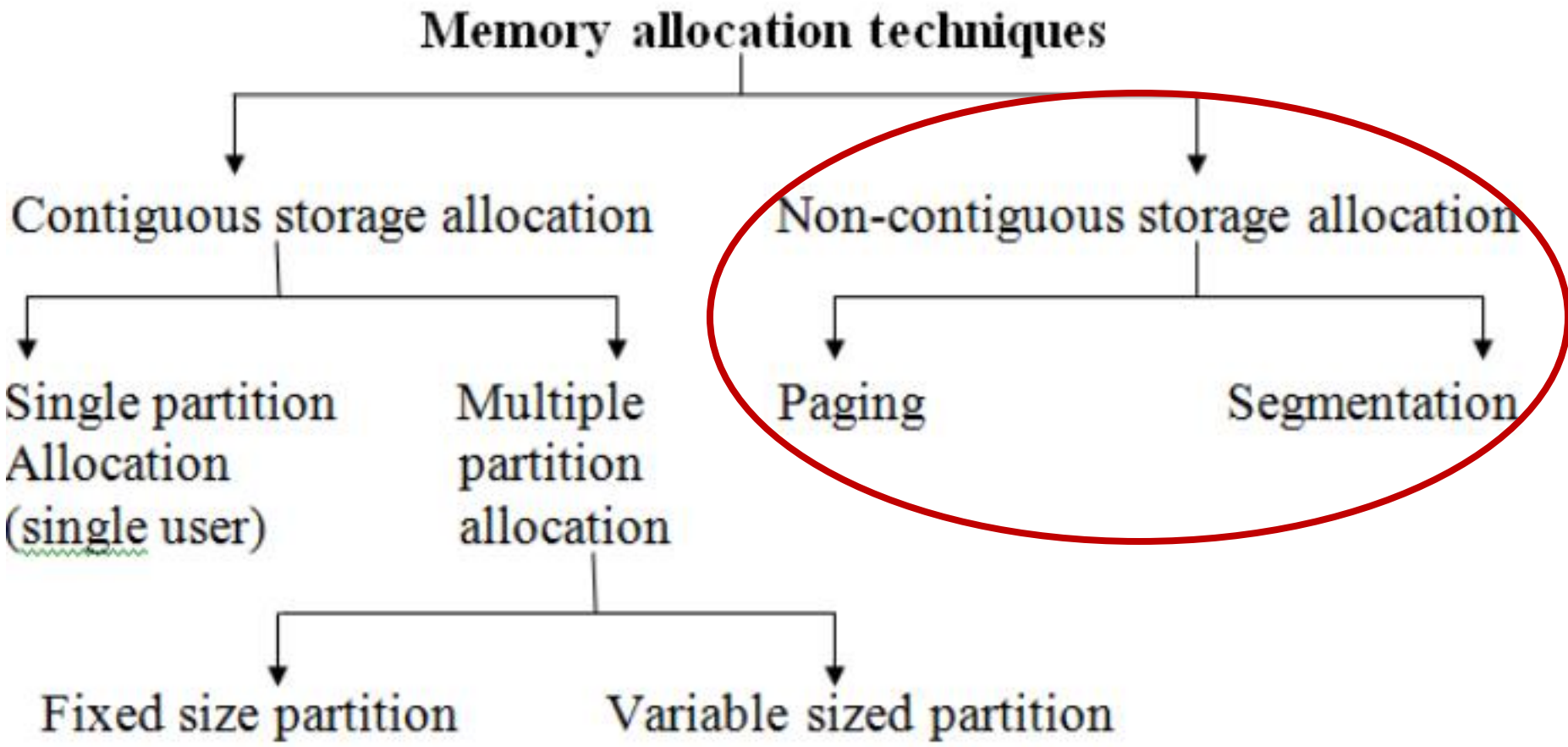
- Shuffle memory contents to place all free memory together in one large block
- Compaction is possible *only* if relocation is dynamic, and is done at execution time

Internal Fragmentation

Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used (e.g. in fixed size partitions)



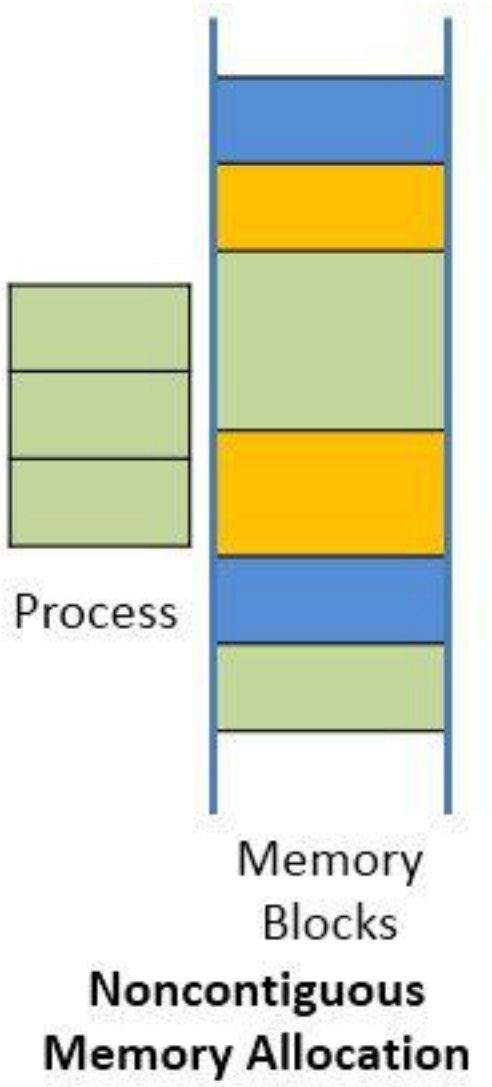
Fixed sized partitions lead to internal fragmentation


Memory allocation techniques
Contiguous storage allocation
Non-contiguous storage allocation
Single partition Allocation (single user)
Multiple partition allocation
Paging
Segmentation
Fixed size partition
Variable sized partition

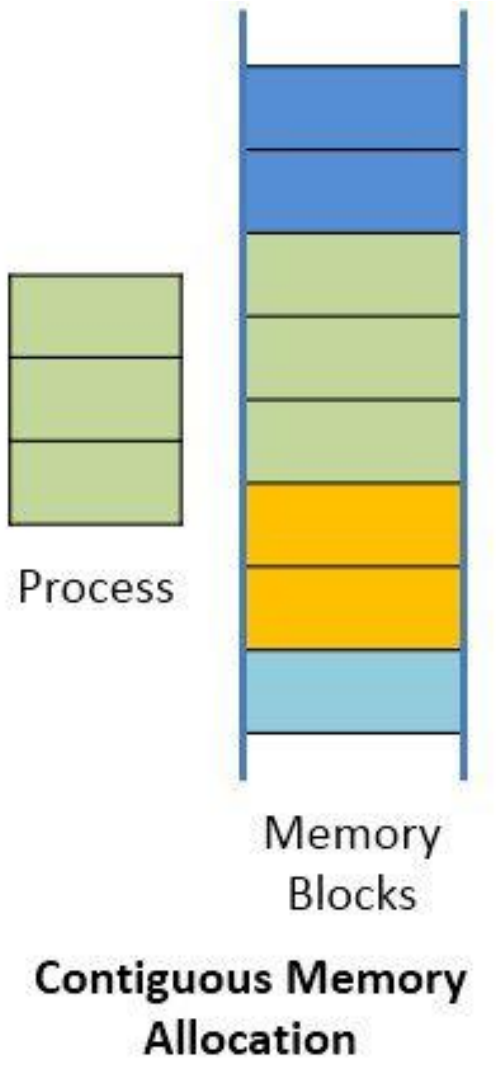
Non-contiguous allocation

- In non-contiguous storage allocation, a program's data and instructions may occupy non-contiguous area of memory.

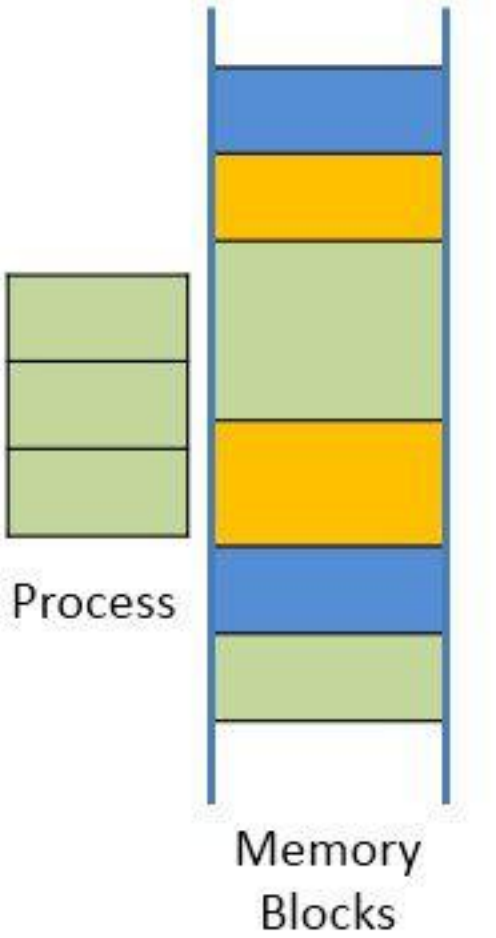
- Two approaches
  - Segmentation
  - Paging



Contiguous vs Noncontiguous allocation



Contiguous Memory Allocation



Noncontiguous Memory Allocation

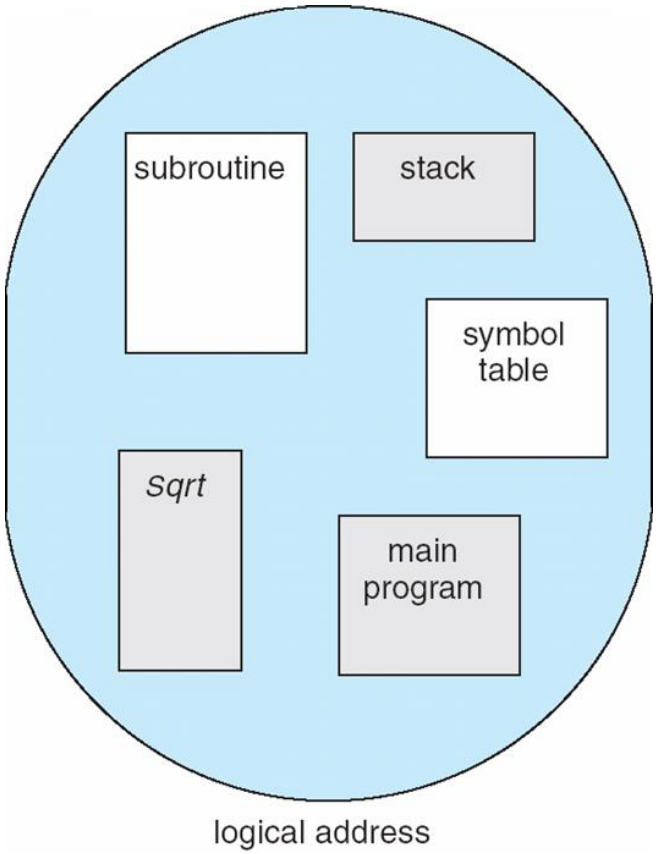
Contiguous vs. Non-contiguous allocation

- **Contiguous Allocation**
  - Program must exist as a single block of contiguous physical addresses
  - Sometimes it is impossible to find a large enough block
  - Low overhead

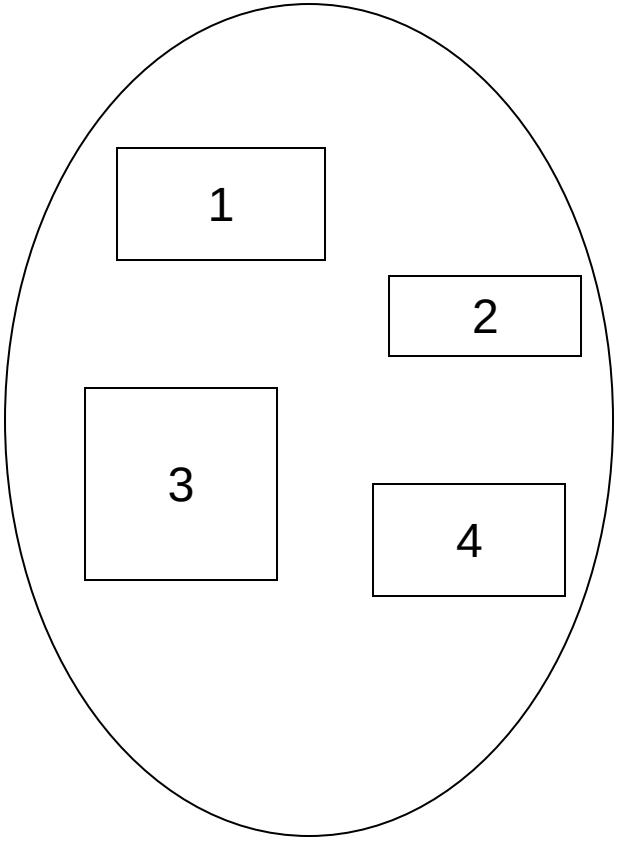
- **Non-contiguous Allocation**
  - Program divided into chunks called pages/segments
  - Each page/segment can be placed in different part of memory
  - Easier to find holes in which a segment will fit
  - Increased number of processes that can exist simultaneously in memory.

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
    - object
    - local / global variables
    - common block
    - stack
    - symbol table
    - arrays



Logical View of Segmentation



user space

physical memory space

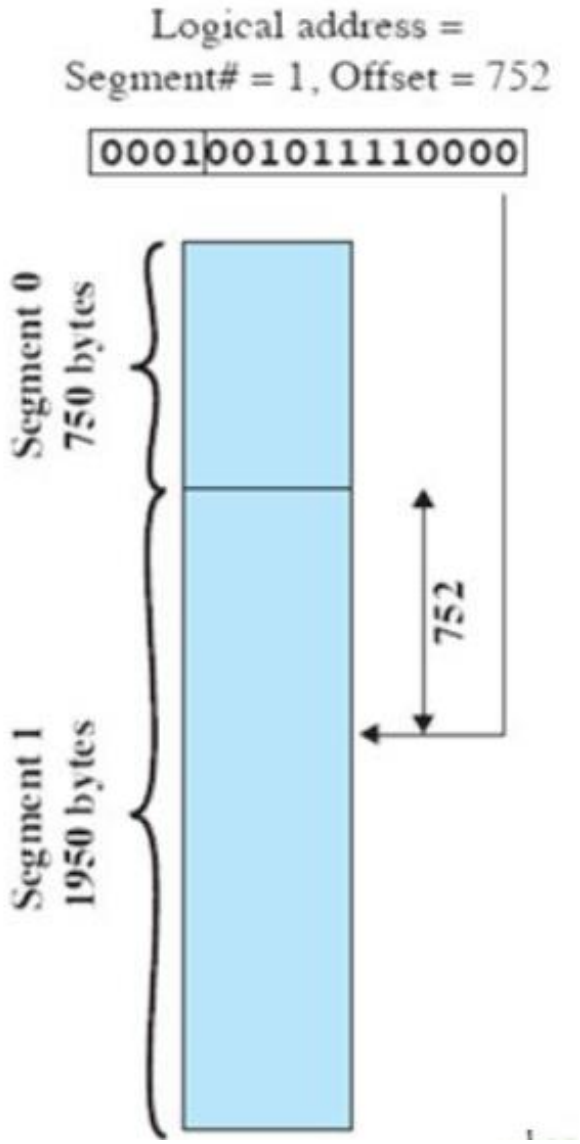
Segmentation Architecture

- Logical address consists of a two tuple:

<segment-number, offset>

segment-number – This identifies the segment to which the memory location belongs.

offset – is a value representing the distance (in terms of memory units) between the start of the segment and the actual memory location of a particular data or instruction.

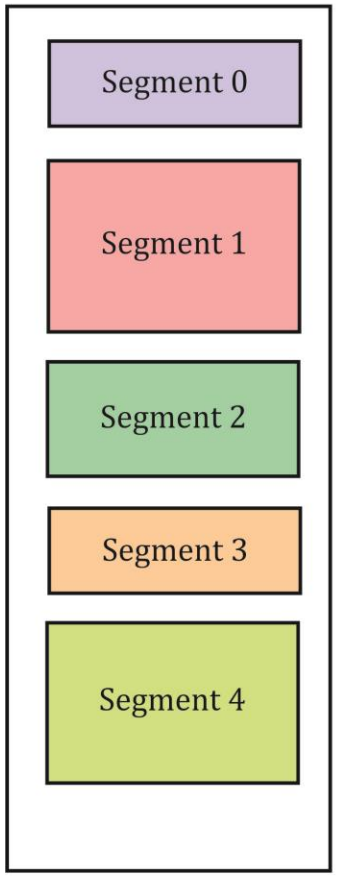


Segmentation Architecture

Segment table – maps logical to physical addresses.

- Logical: <segment-number, offset>
- Physical: base and limit values
  - **base** – the starting physical address where the **segments** reside in memory
  - **limit** – specifies the length of the **segment**

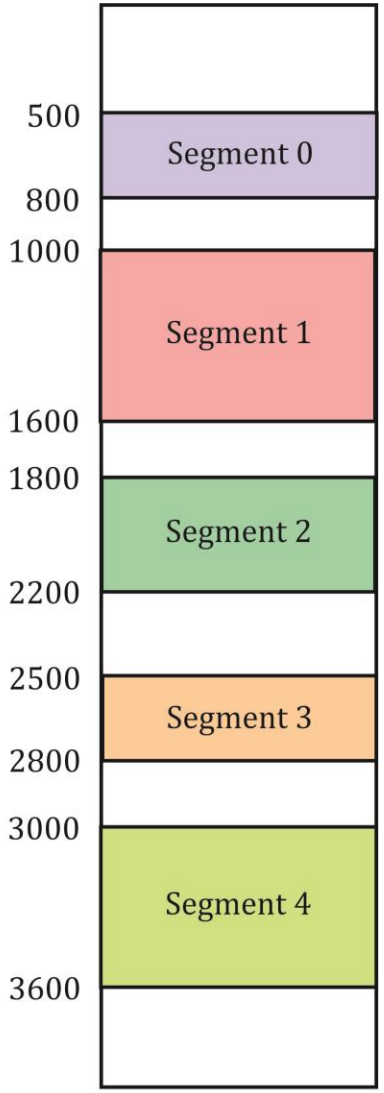
Segmentation Architecture



Logical Memory

Segment No.	Base Address	Segment Limit
0	500	300
1	1000	600
2	1800	400
3	2500	300
4	3000	600

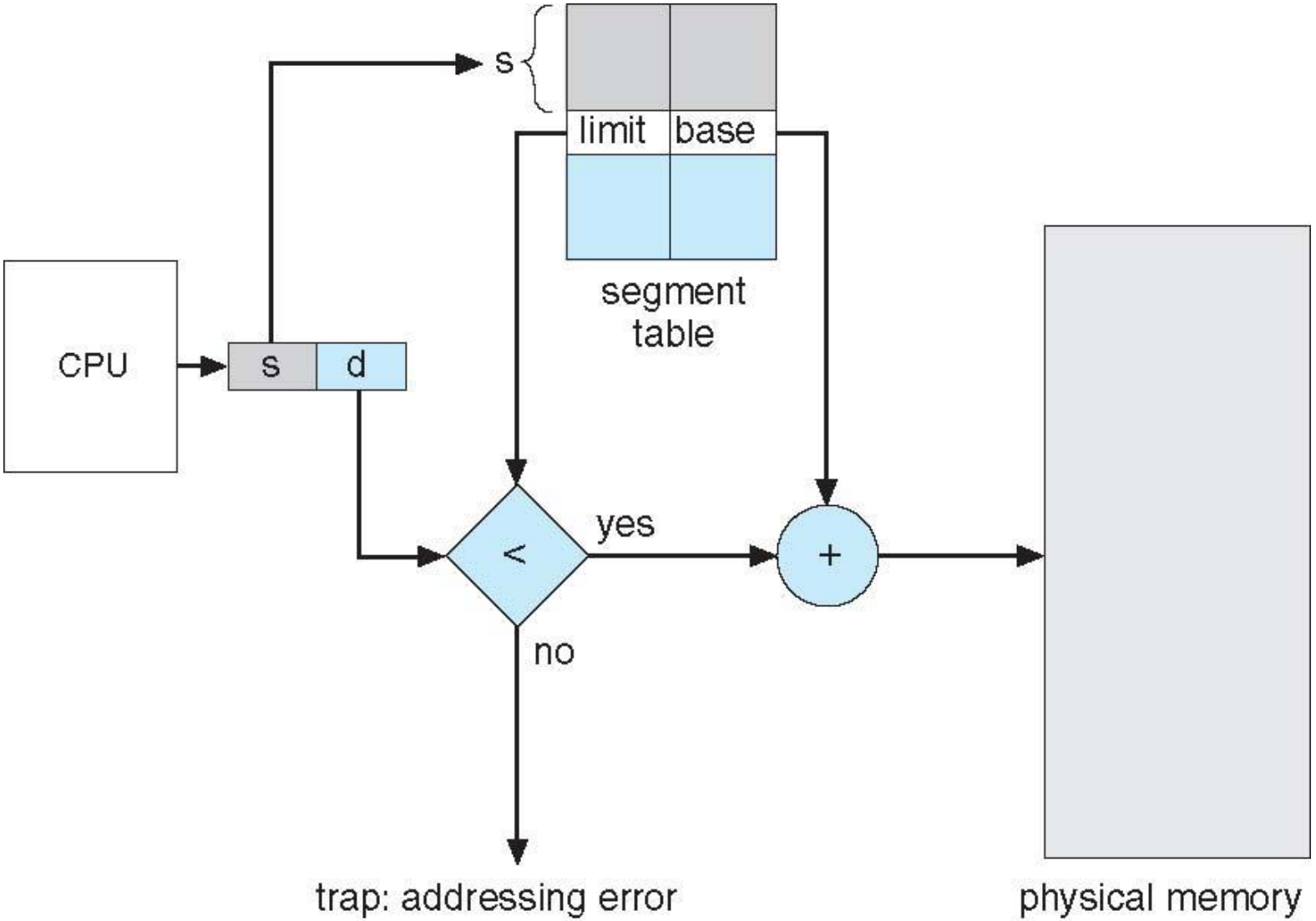
Segment Table



Physical Memory

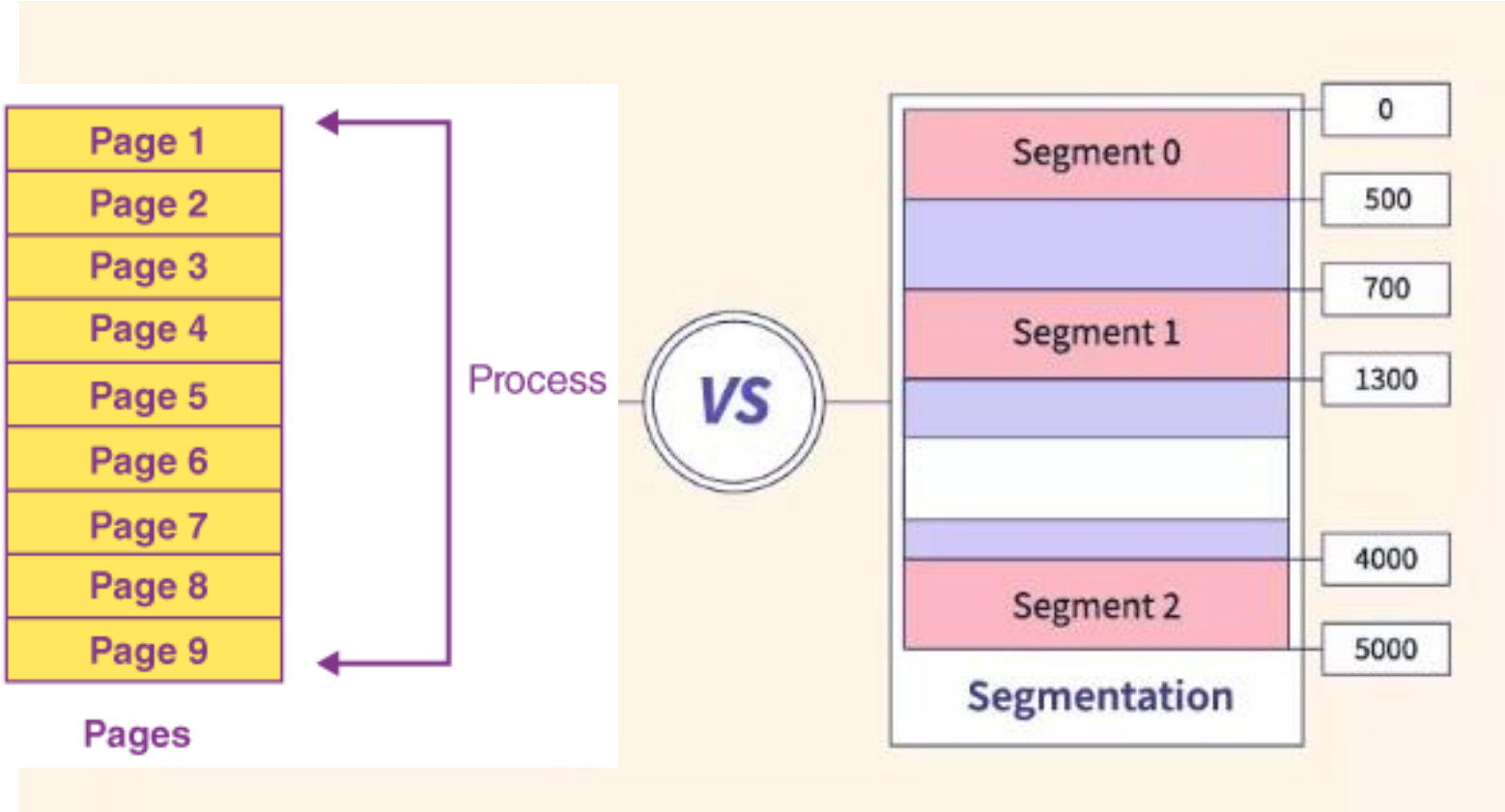
Segmentation Hardware

Maps two dimensional logical address (segment no s, offset d) to one dimensional physical address



Paging

Paging vs Segmentation



Paging

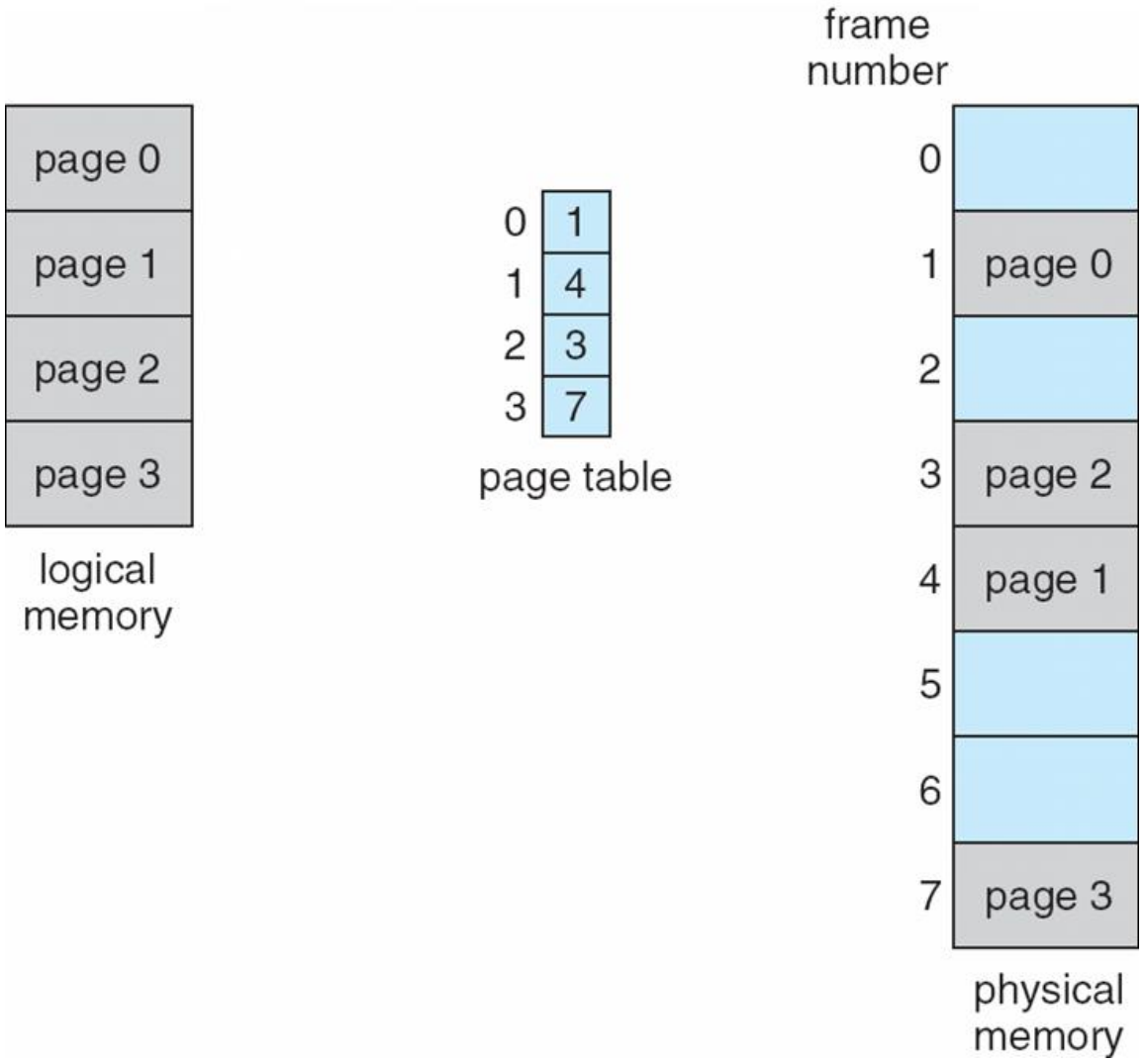
- Segmentation : Allows external fragmentation
  → need for compaction

- Paging :
  - Avoids external fragmentation
  - Avoids the need for compaction
  - Avoids problem of varying sized memory chunks
  - Still have Internal fragmentation

Paging

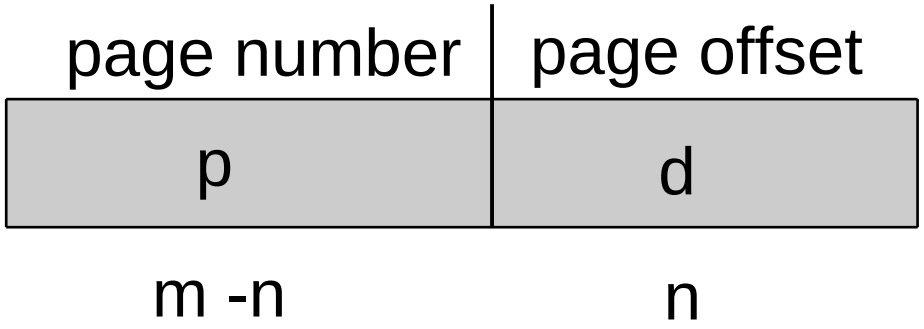
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages

Paging Model of Logical and Physical Memory



Address Translation Scheme

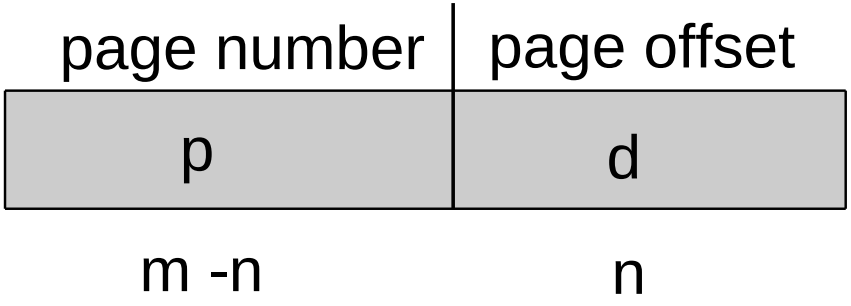
- Consider m bit logical address space. This space is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



For given logical address space 2^m and Page size 2^n bytes, number of pages = 2^{m-n}

Byte addressing

Address Translation Scheme



E.g. What will be the maximum no of pages in a page table with 16 bit addresses and 8 KB page size.

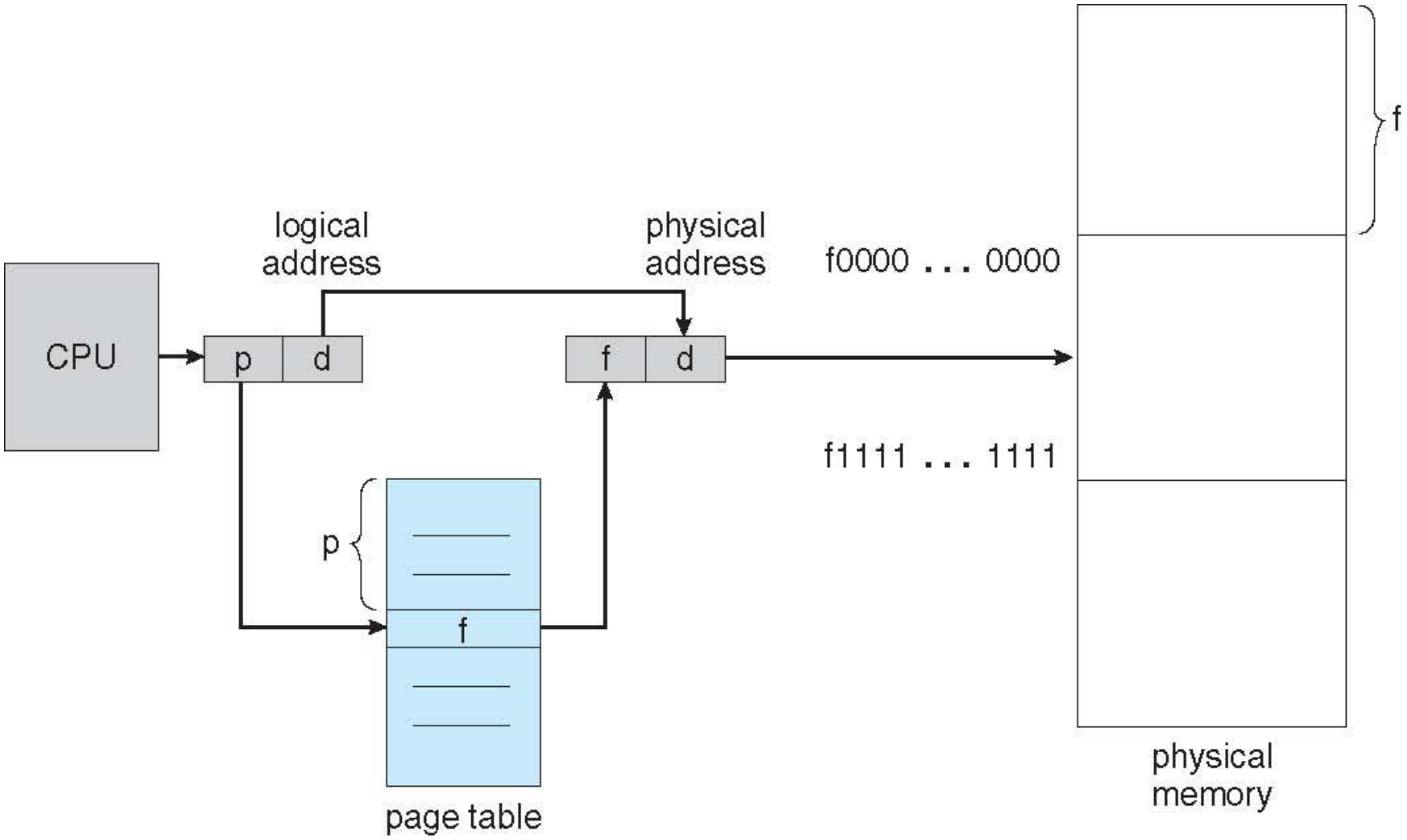
$8\ KB = 2^3 \times 2^{10} = 2^{13}$

13 bits are allocated for offset

→16-13 bits are for page number

→2^3 = 8 pages

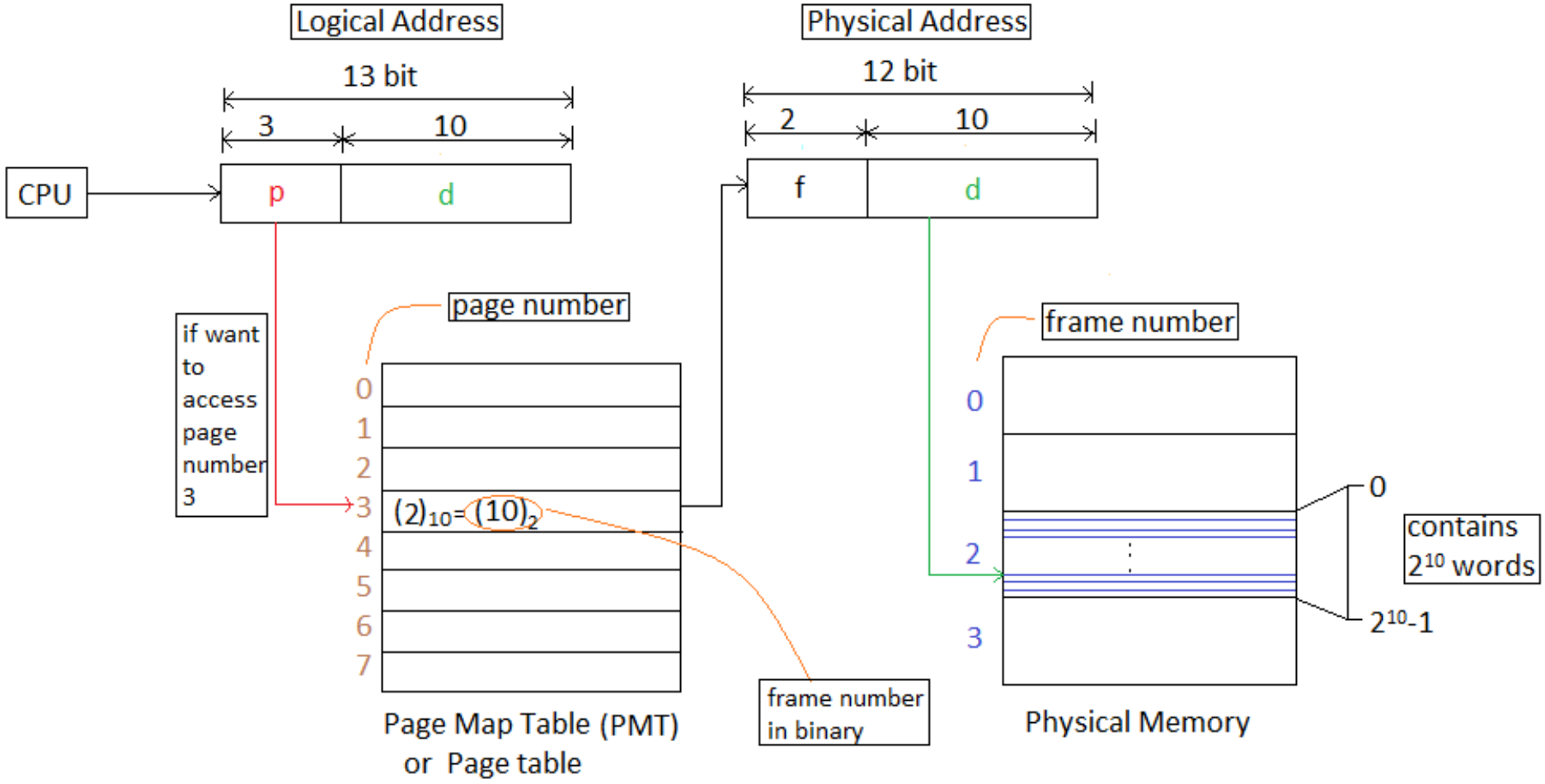
Paging Hardware



Example

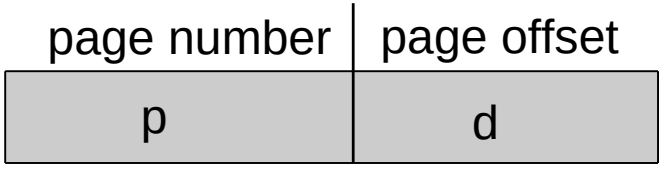
- Consider 13 bit logical address space, and 12 bit physical address space.
- Assume page size = frame size = 2^10 = 1KB
- Page table entry size is different than page size
- No of pages = 2^3 = 8, No of frames = 2^2 = 4
- Usually logical address space is > physical address space, physical address space is limited by the size of the RAM.

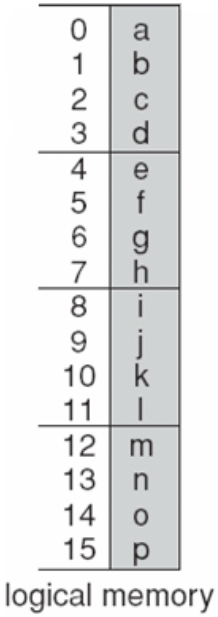
Example

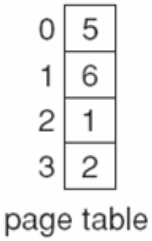

Logical Address
13 bit
3
10
CPU
p
d
Physical Address
12 bit
2
10
f
d
if want to access page number 3
page number
0
1
2
3
4
5
6
7
$(2)_{10}=(10)_2$
Page Map Table (PMT)
or Page table
frame number in binary
frame number
0
1
2
3
0
$2^{10}-1$
contains 2^{10} words
Physical Memory

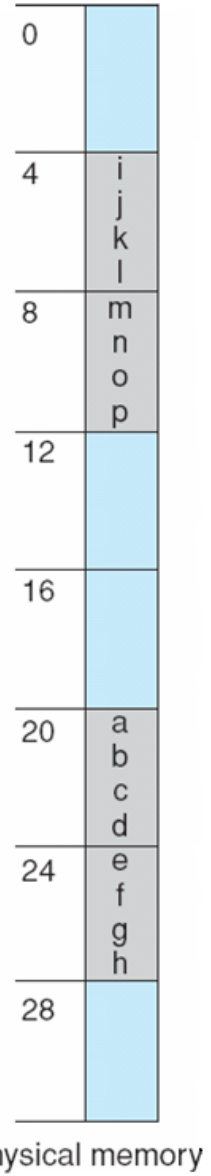
Paging Example

4 bit logical address space,
32 byte of physical memory
Page size = 4 bytes









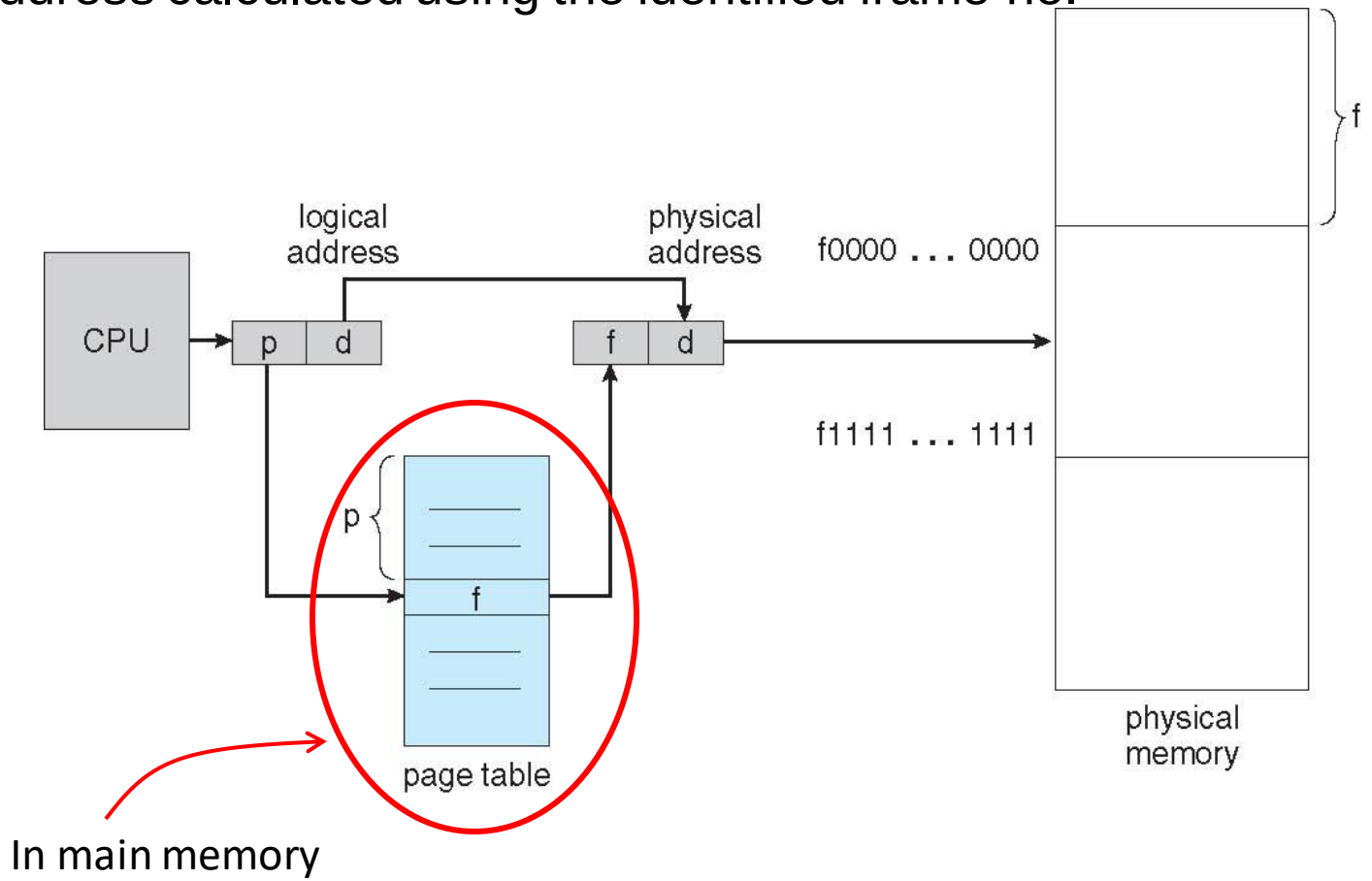
→ Page size = 4 bytes (4 blocks, each with 1 byte of size.)
→ n = 2
→ No of pages = 2^(4-2) = 4

Size of the physical memory = 32 bytes = 2^5.
m = 5, n = 2 ➔ number of frames = 2^(5-2) = 8

Size of the page table = no of pages X size of each entry in the table
= 4 x size of each entry in the table

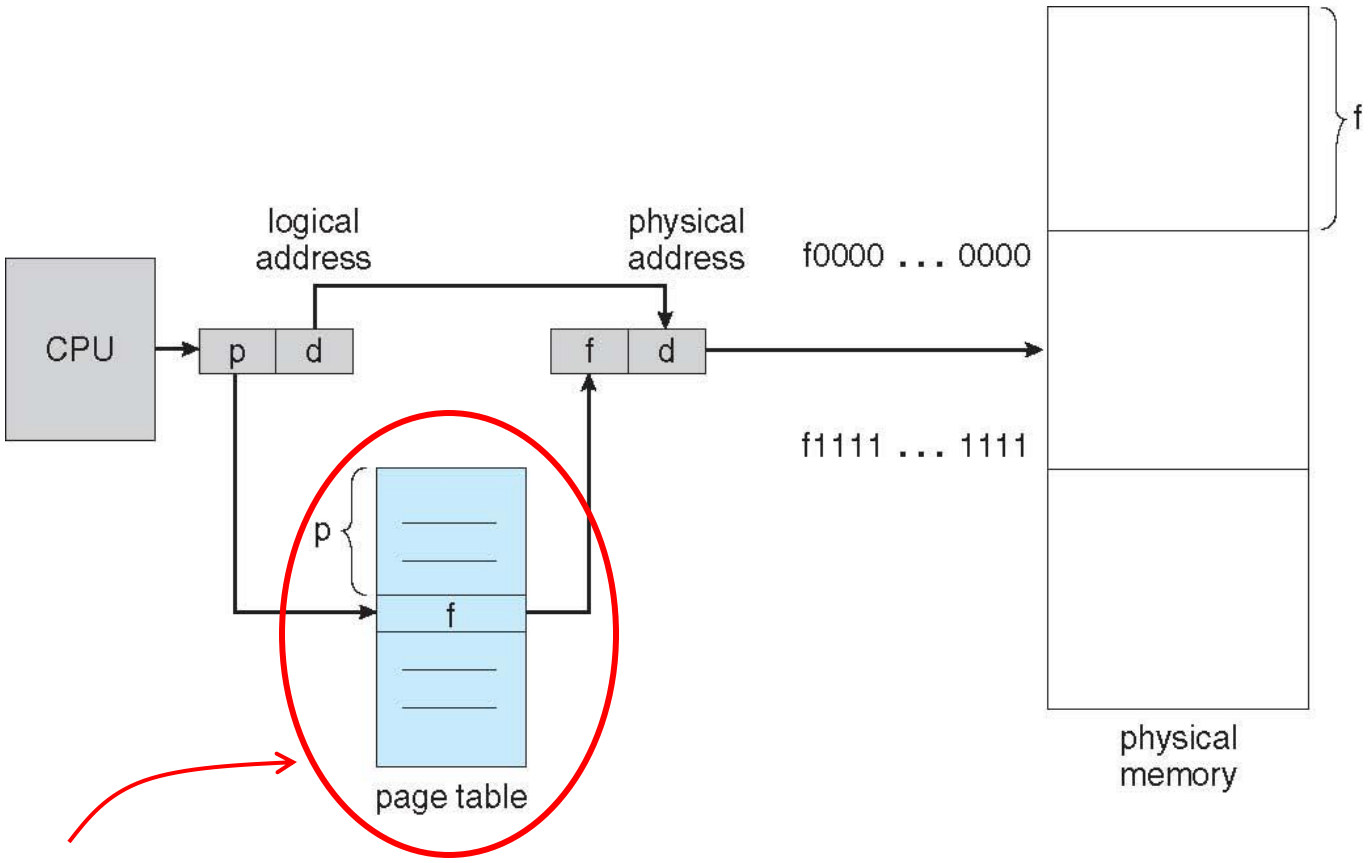
Page Table

- Page table is kept in main memory
- Every data access requires two memory accesses
  - First one is to access the page table to get the frame number.
  - Second is to access the data in memory using the physical address calculated using the identified frame no.



Translation Lookaside Buffer (TLB)

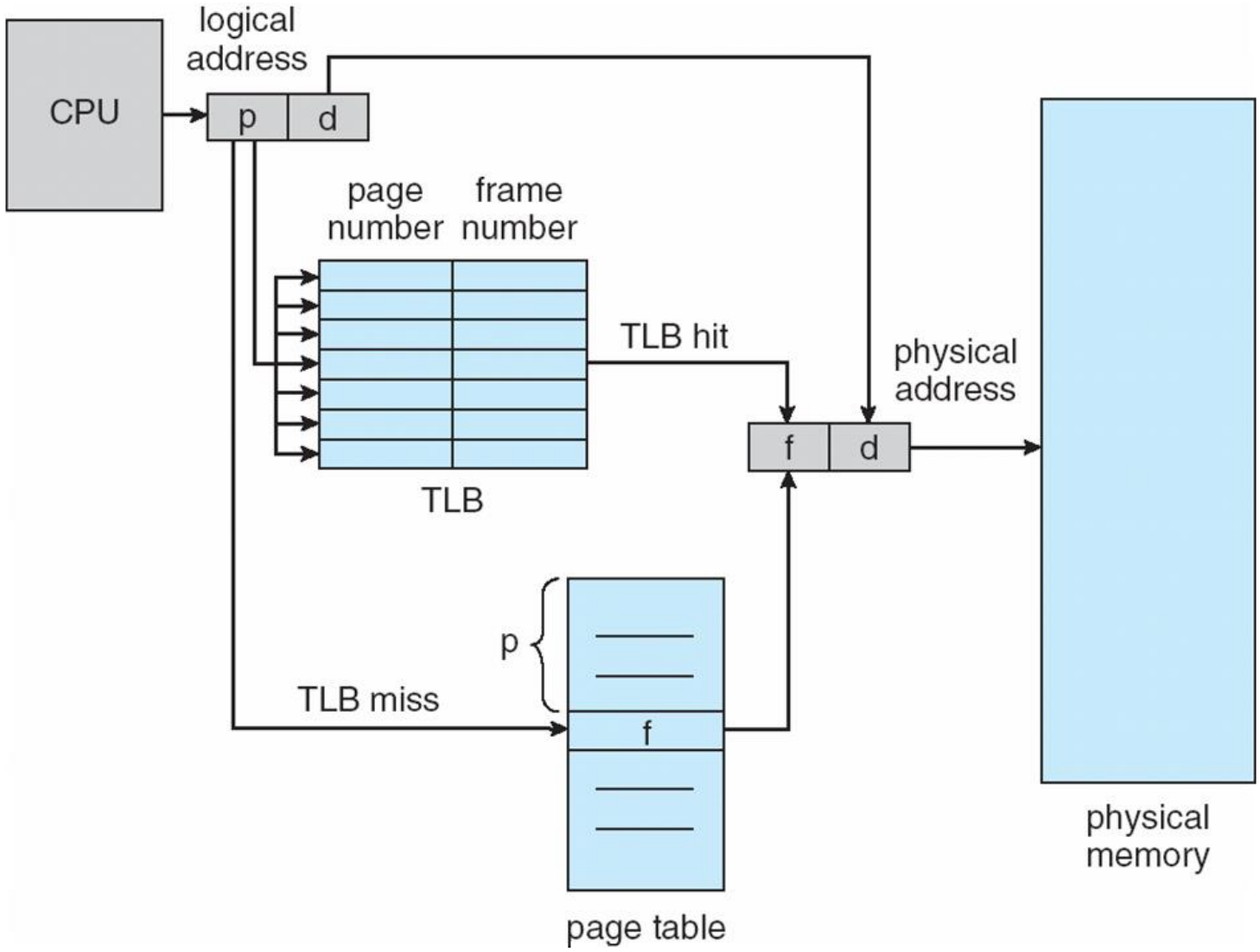
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)



Translation Lookaside Buffer (TLB)

- A cache that stores recent translations of logical memory to physical addresses for faster retrieval.
- TLB stores a few (32 to 1024) entries (page no. to frame no. mappings).
- How does it work?
  - When a logical address is generated by the CPU, its page no. is first checked to see whether it is in the TLB or not.
  - If it is in the TLB (TLB hit), frame no. is returned and the frame can be fetched from the memory – only one memory access.
  - If it is not (TLB miss), two memory accesses are needed.
- On a TLB miss, value is loaded into the TLB for faster access next time

Paging Hardware With TLB



End of Part-1