

Operating Systems

Prepared by Dr. Alaa Eldeen Sayed Ahmed

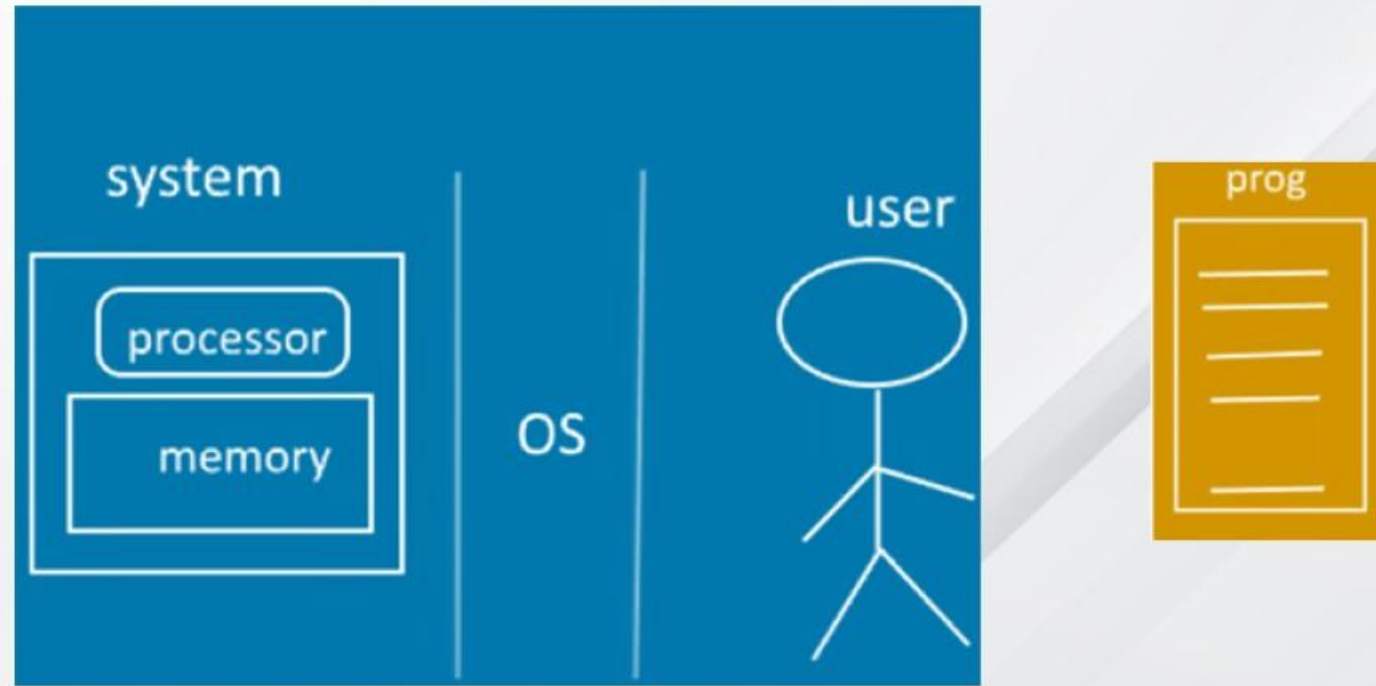
IT263 / CYB510
Memory Management- part1

Lecture 7

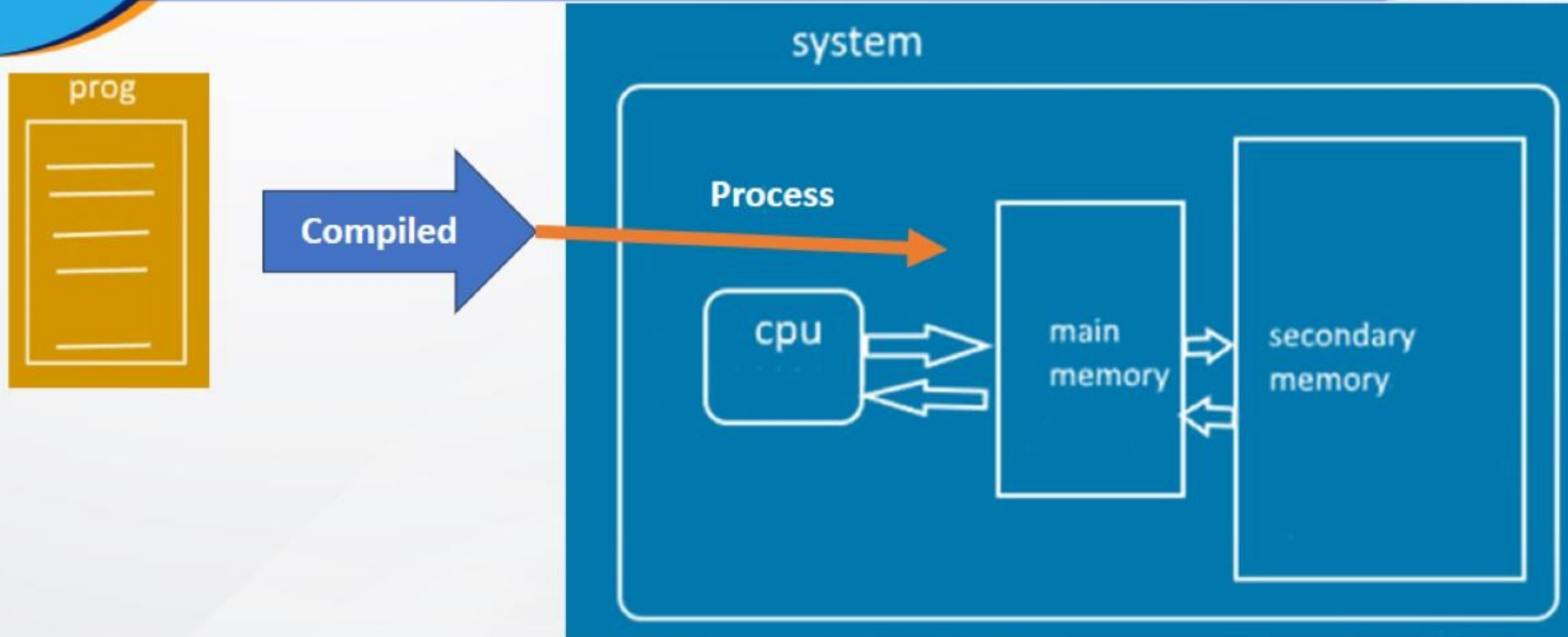


- Objectives:
 - Background
 - Contiguous Memory Allocation
 - Non-Contiguous Memory Allocation

- Topics To Be Covered:
 - To provide a detailed description of various ways of organizing memory hardware
 - To discuss various memory-management techniques (Segmentation and Paging).



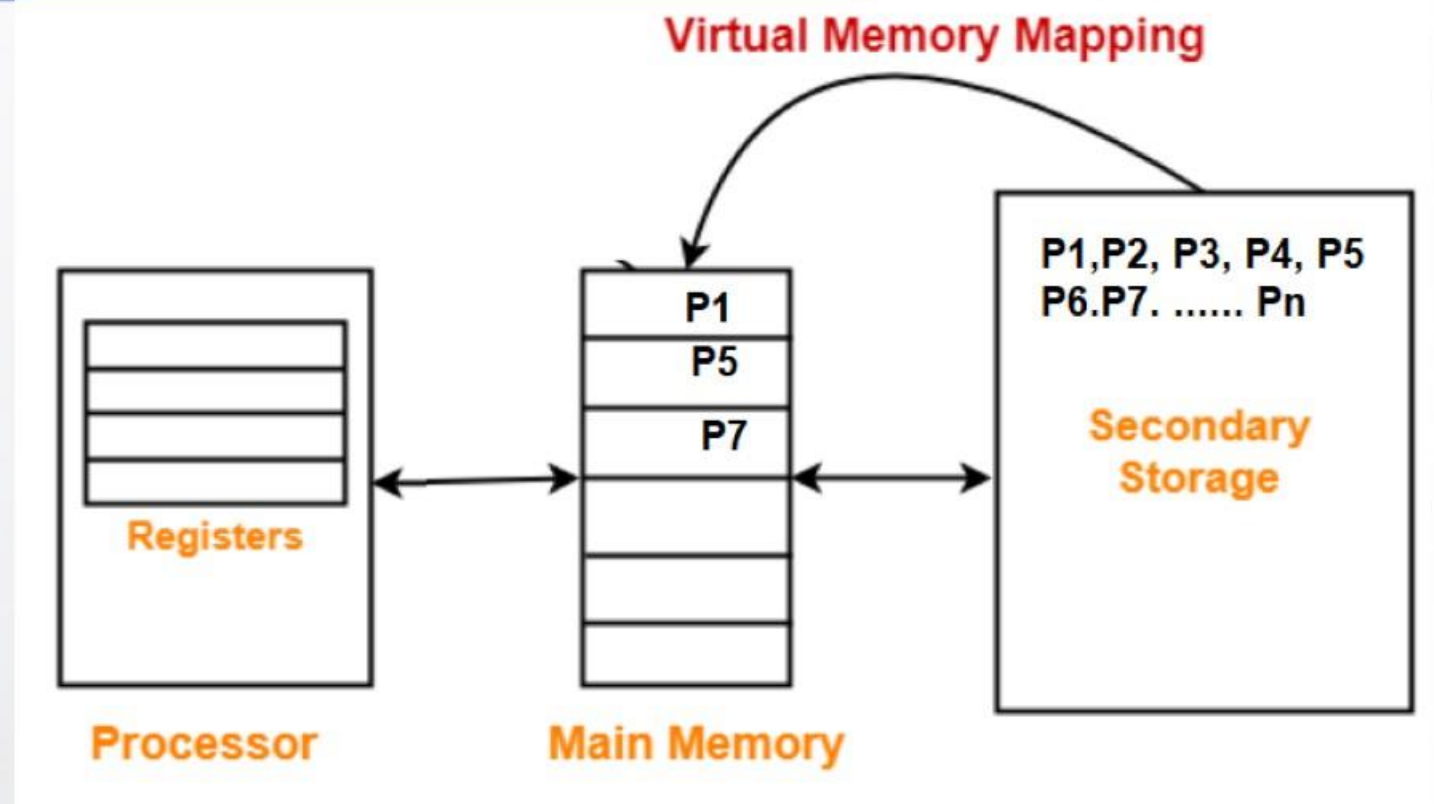
- Program must be brought (from disk) into memory and placed within a process for it to be run



- The memory is organized in the form of hierarchy and it mainly consists of two parts: **Main memory and Secondary Memory**
- Main memory and registers are only storage CPU can access directly

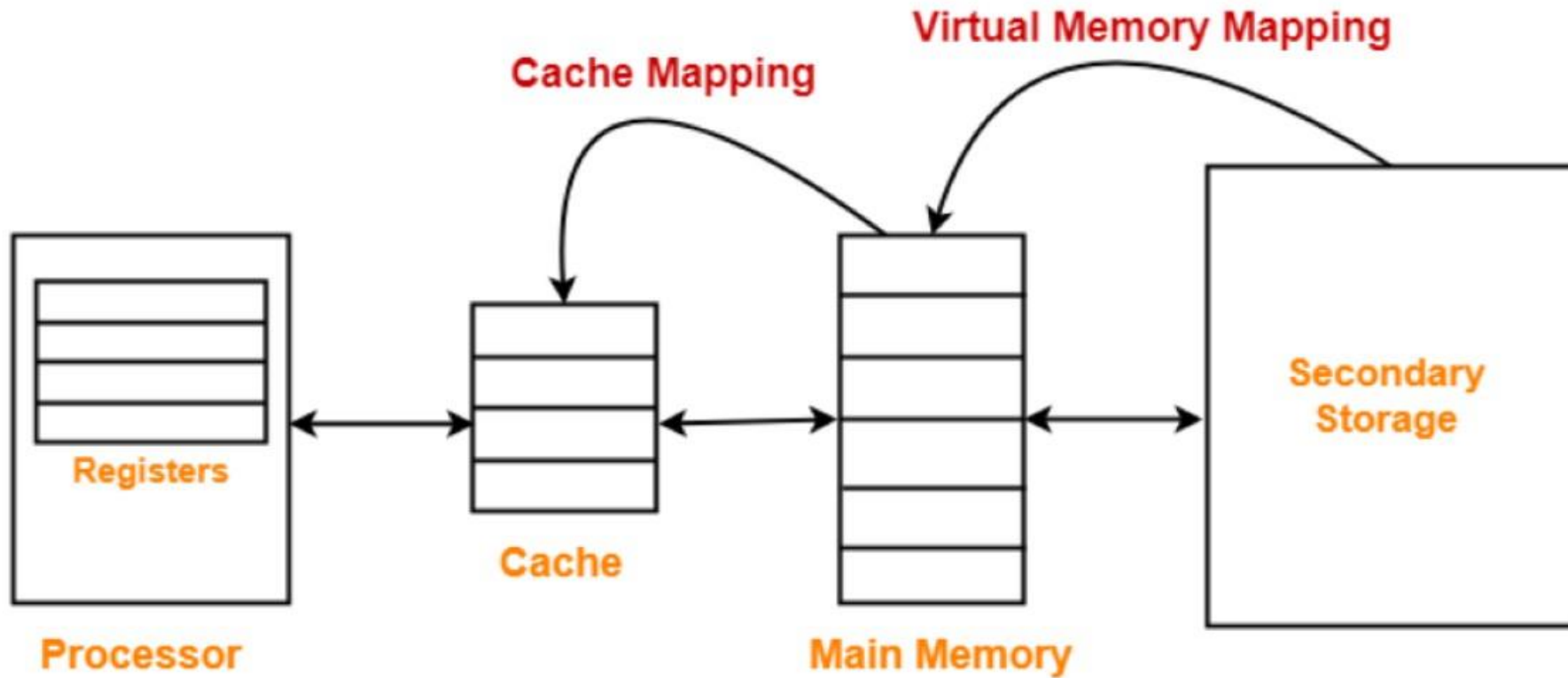
- To Keep more processes in the memory at the same time.
- The main memory size is small compared to Hard disk.
- Its size cannot be increased because of two reasons:
 - cost
 - Access time.
- The Secondary Memory size is bigger than main memory, so it has:
 - High Access time
 - Low cost

- OS, always tries to keep more processes in the memory. The number of processes in memory is called **degree of multiprogramming**.
- But since the memory size is limited, **memory management techniques** are used to give an illusion that there are more number of processes are placed in the memory at the same time.



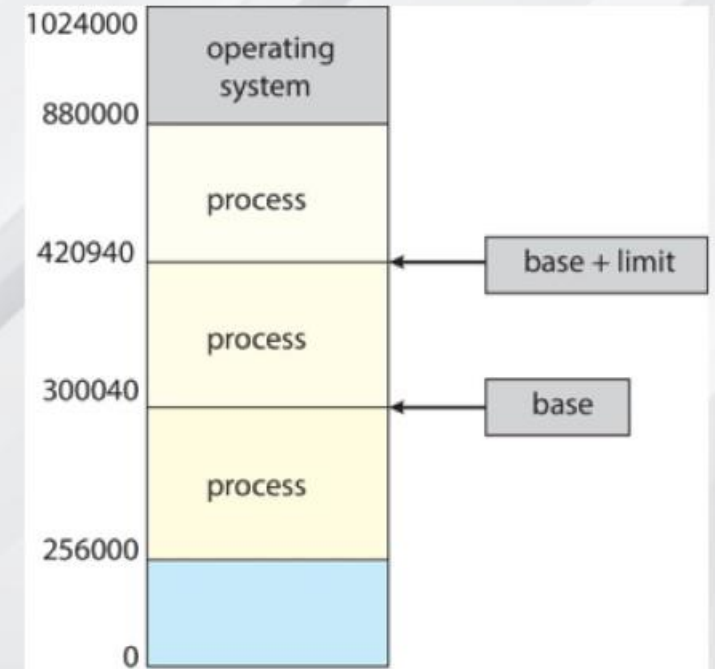
Background

- Memory unit only sees a stream of:
 - addresses + read requests, or
 - address + data and write requests
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation



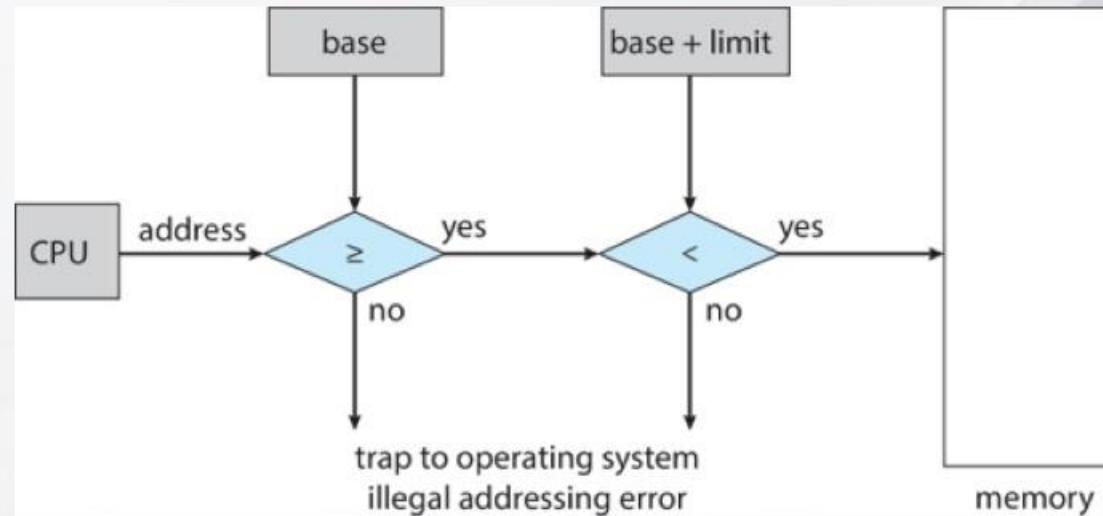
- Protection
- Address Binding, relocation and translation
- Allocation and de-allocation of memory
- Virtual Memory.
- Swapping

- Need to ensure that a process can access only those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit registers** to define the logical address space of a process



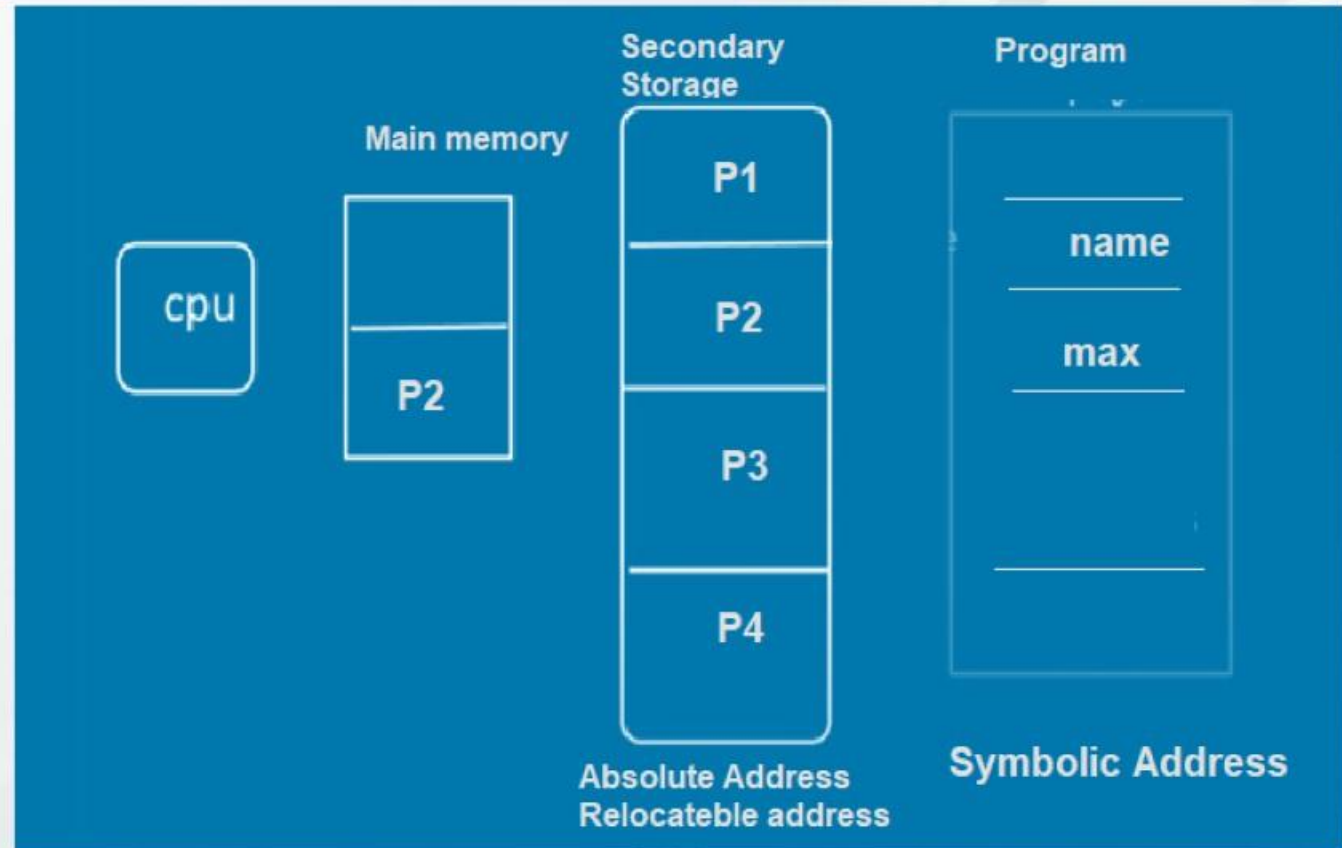
Hardware Address Protection

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



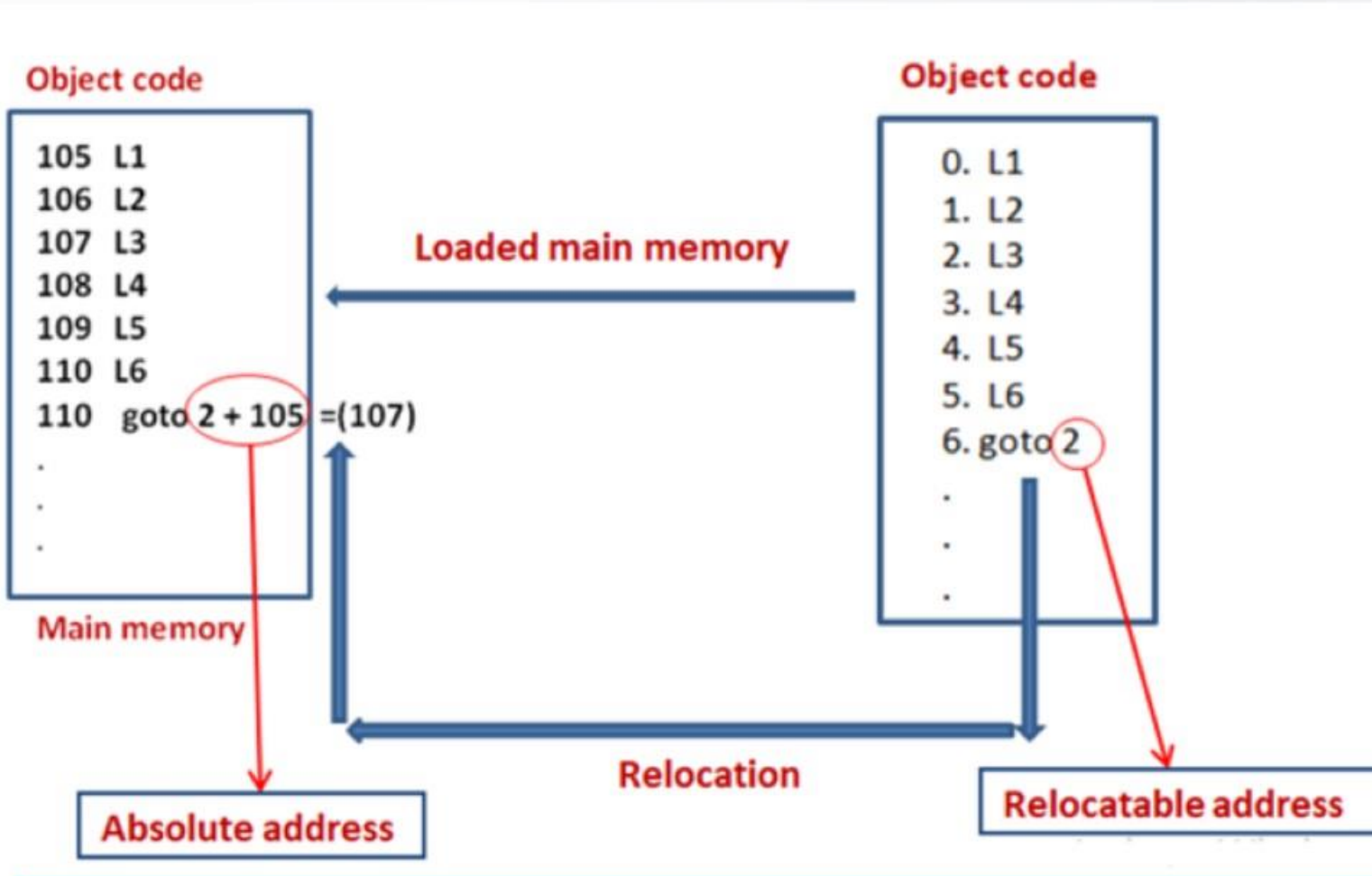
- the instructions to load the base and limit registers are privileged

- Programs on disk, ready to be brought into memory to execute form an **input queue**
 - Without OS support, programs must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000



- Addresses represented in different ways at different stages of a program's life
 - Source code addresses usually **symbolic Addresses**
 - symbolic Addresses is An addressing scheme whereby reference to an address is made by some convenient symbol that (preferably) has some relationship to the meaning of the data expected to be located at that address. It serves as an aid to the programmer.
 - After compiling the program, the symbolic addresses is converted to either **absolute** or **relocatable addresses**.

- expression.
- An address expression that has a fixed value, independent of run-time considerations such as where the program is in memory, is called an **absolute address**
- An address expression with a value that depends on run-time considerations is called a **relocatable address** expression. A relocatable expression with a value that depends only on where a process is located in memory.
- The Compiled code addresses **bind** to **relocatable addresses**
- Linker or loader will bind relocatable addresses to **absolute addresses**
- Each binding maps one address space to another



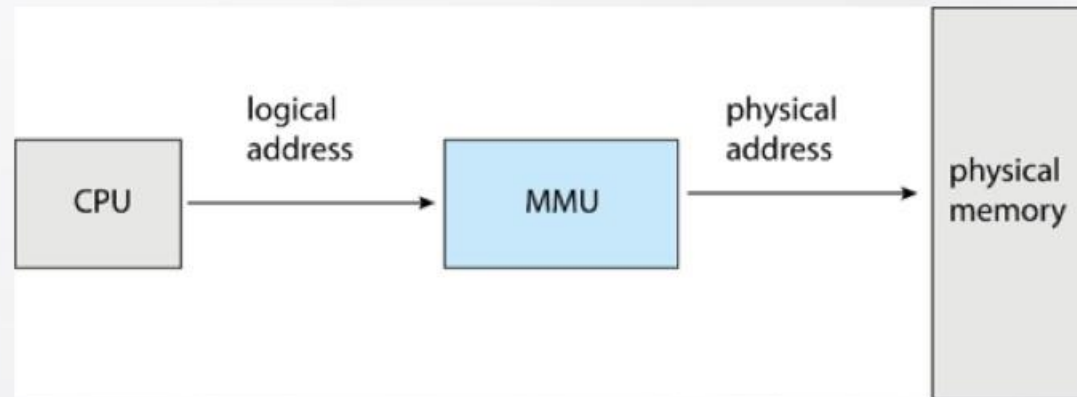
- Address binding of instructions and data to memory addresses can happen at three different stages
 - **Compile time:** If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
 - **Load time:** Must generate **relocatable code** if memory location is not known at compile time
 - **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 (e.g., base and limit registers)

Logical vs. Physical Address Space

- An address generated by the CPU is commonly referred to as a **logical address**, whereas an address seen by the memory unit—that is, the one loaded into the memory—is commonly referred to as a **physical address**.
- **Logical address** : also referred to as **virtual address**
- **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

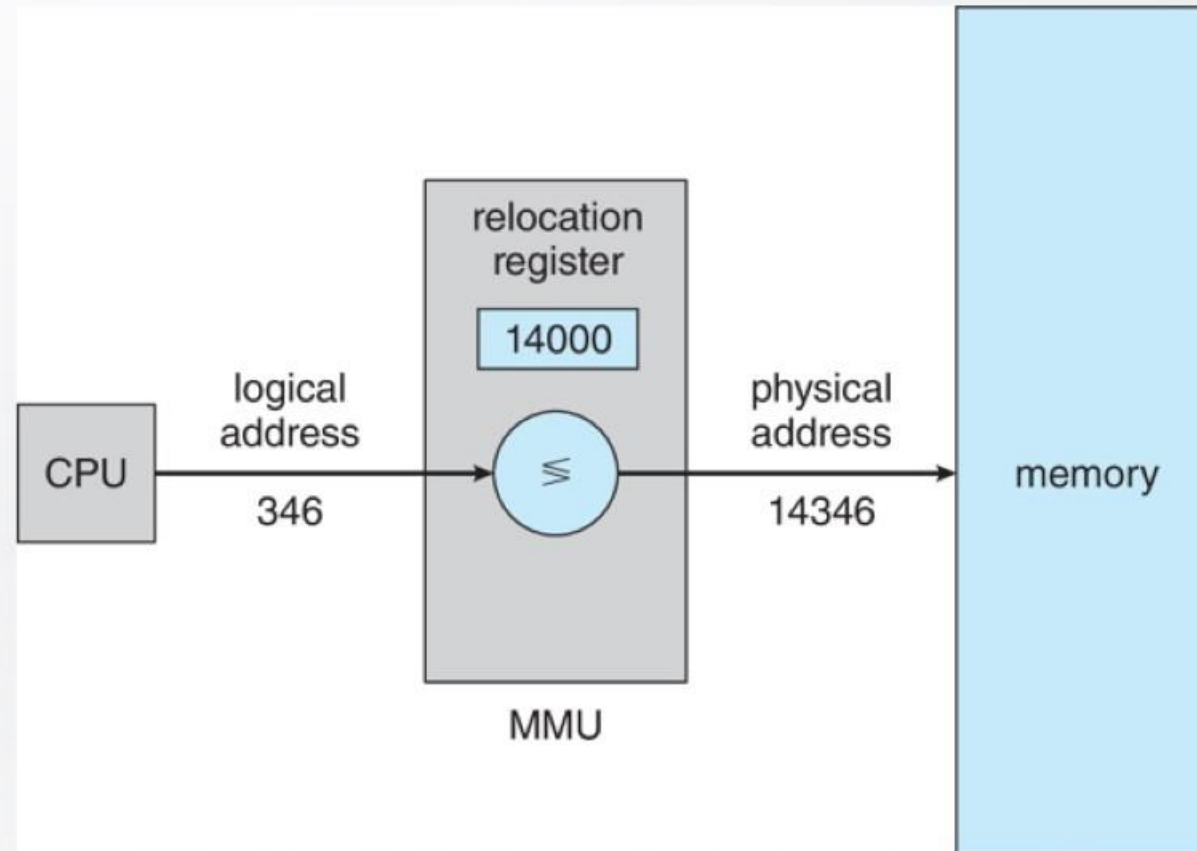
Memory-Management Unit (MMU)

- Hardware device that at run time maps logical or virtual to physical address



- Many methods possible, covered in the rest of this chapter

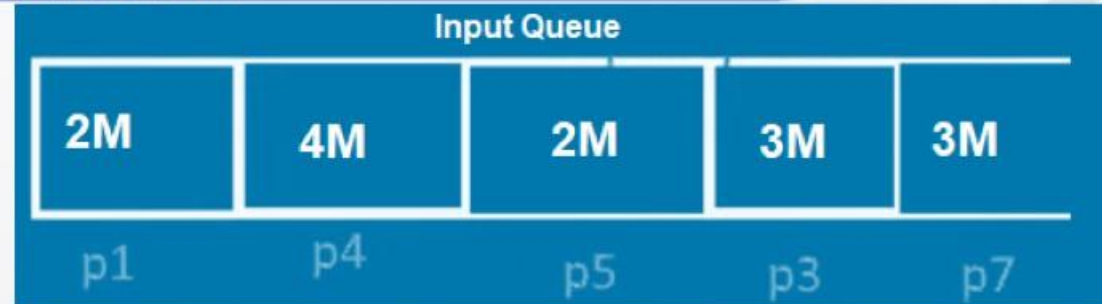
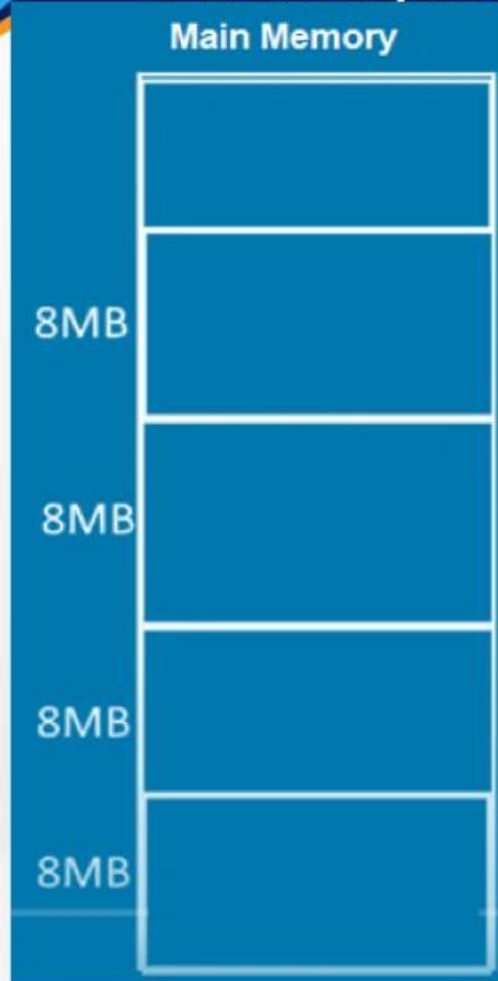
- Consider simple scheme. which is a generalization of the base-register scheme.
- The base register now is called **relocation register**
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* 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



- There are two types of memory allocations:
 - Contiguous
 - Fixed partitions
 - Dynamic Partitions
 - Non-contiguous
 - Paging
 - Segmentation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
 - Resident operating system, usually held in low memory with interrupt vector
 - User processes then held in high memory
 - Each process contained in single contiguous section of memory

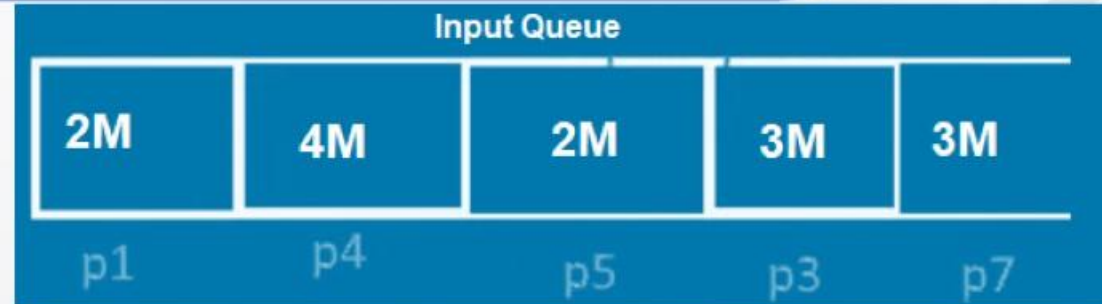
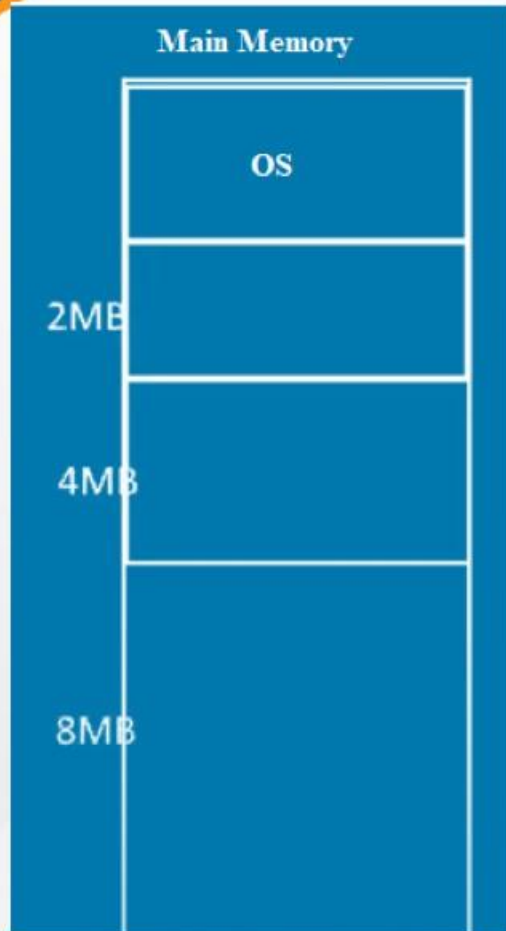
Contiguous Allocation: Fixed equal sized Partitions



Disadvantages:

1. Number of partitions is fixed → Degree of multiprogramming is fixed
2. Each partition can hold at most one process. The remaining part of the partition is wasted and this is called **Internal fragmentation**.
3. Total memory space may exist to satisfy a request, but it is not contiguous. This is called **External fragmentation**.
4. There is a limit on the size of the Process.

Contiguous Allocation: Fixed variable sized Partitions

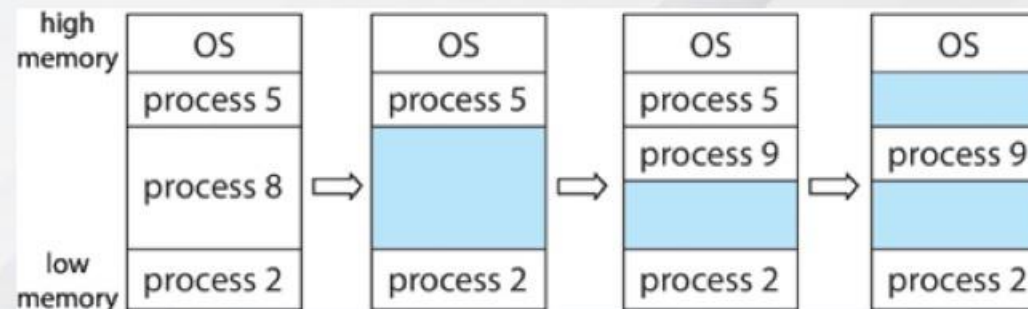


Disadvantages:

1. Number of partitions is fixed → Degree of multiprogramming is fixed
2. Total memory space may exist to satisfy a request, but it is not contiguous. This is called **External fragmentation**.
3. There is a limit on the size of the Process (based on the size of the largest partition)

Dynamic Storage-Allocation (variable 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)



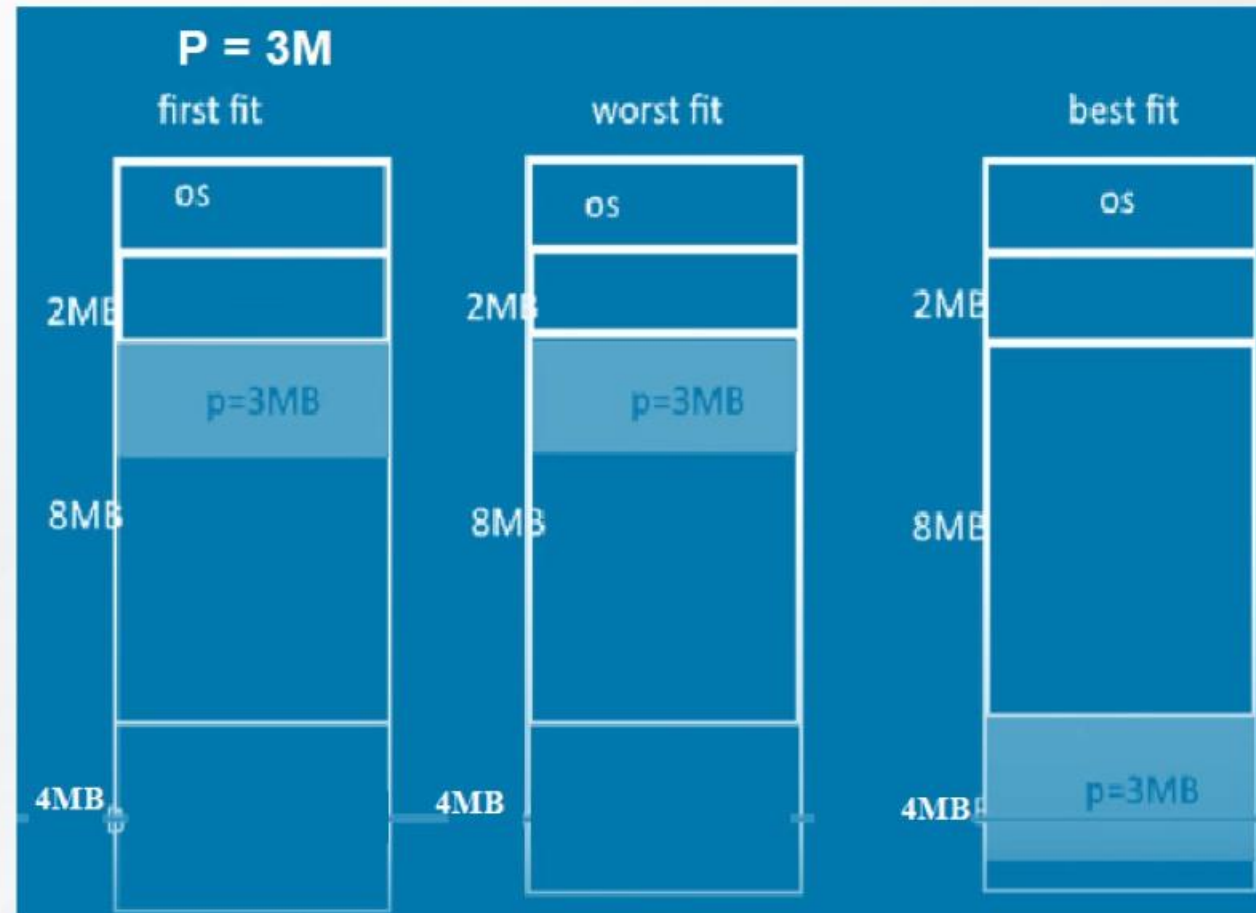
Dynamic Storage-Allocation (variable partitions)

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

Dynamic Storage-Allocation (variable partitions)

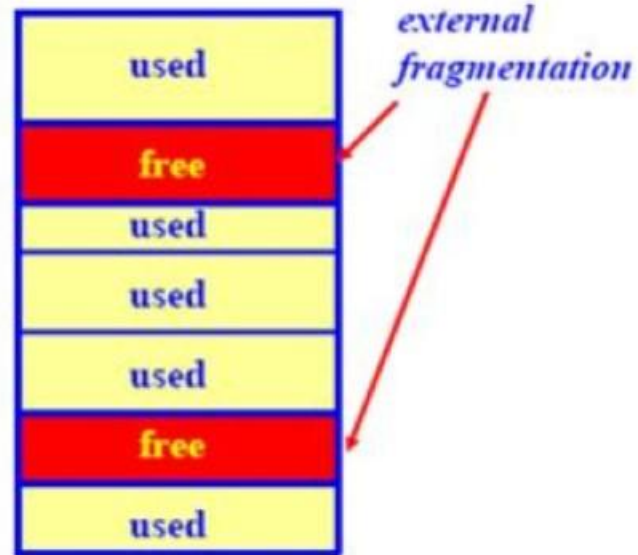


- **Limitations:**

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- Allocation and de-allocation is complex

- **Advantages:**

- **No Internal fragmentation** exists in this.
- Degree of multiprogramming is dynamic and not fixed.
- Process size is not limited to the partition size
- First fit analysis reveals that given N blocks allocated, $0.5 N$ blocks lost to fragmentation
 - $1/3$ may be unusable -> **50-percent rule**

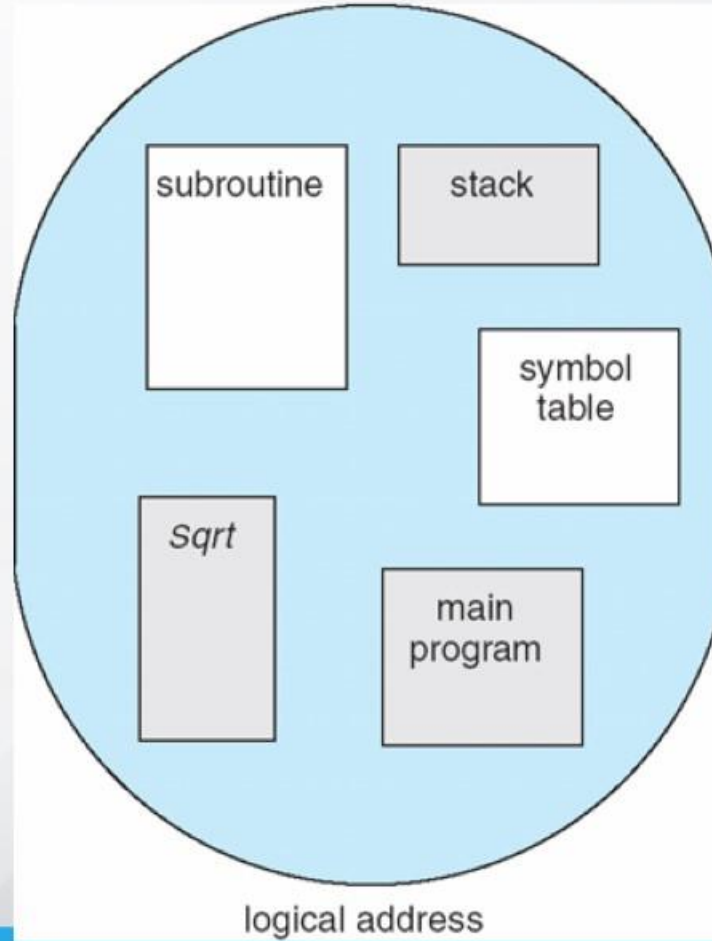


Fragmentation

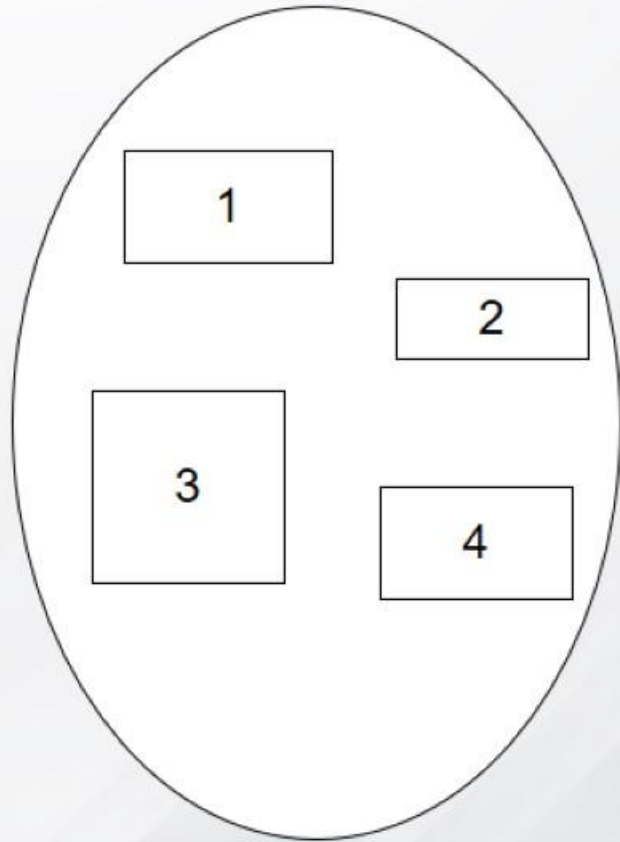
- Reduce external fragmentation by **compaction**
 - Shuffle memory contents to place all free memory together in one large block
- Compaction is possible *only* if relocation is dynamic, and is done at execution time
- I/O problem
 - Latch job in memory while it is involved in I/O
 - Do I/O only into OS buffers

Non- Contiguous Allocation. Segmentation

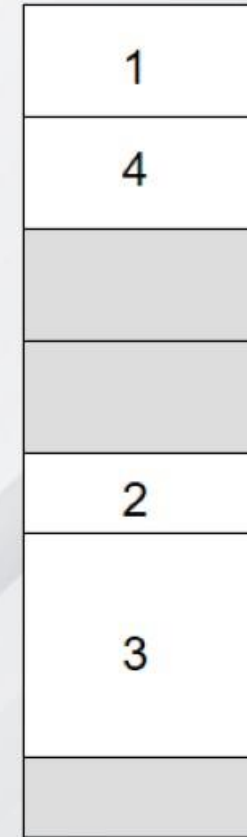
- Memory-management scheme that supports user view of memory
- A program is a collection of segments
 - A segment is a logical unit such as:
 - main program
 - procedure
 - function
 - method
 - object
 - arrays



Non-Contiguous Allocation. Segmentation



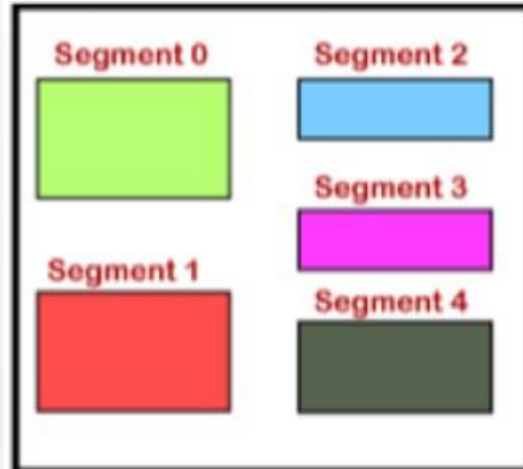
user space



physical memory space

- **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

Segmentation Architecture

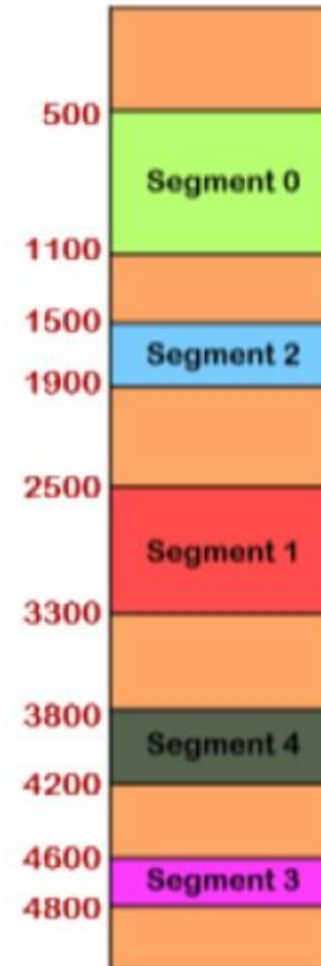


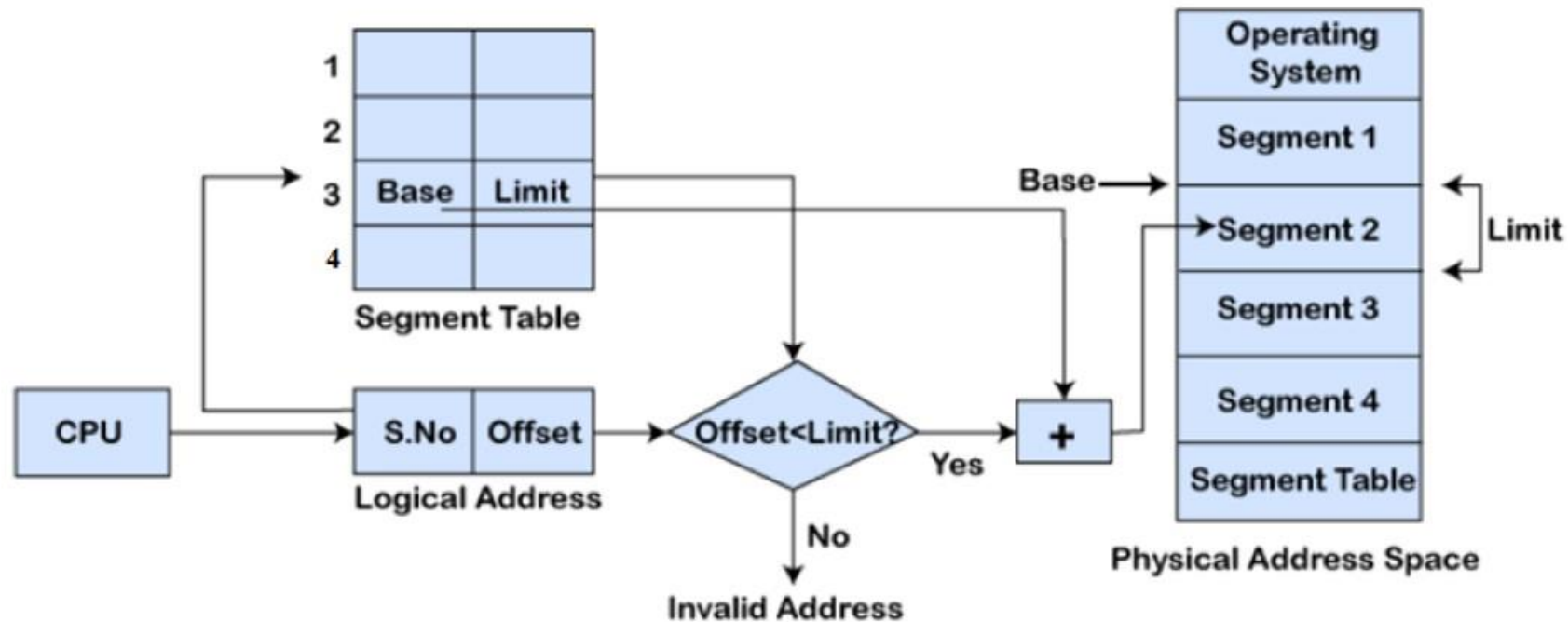
Logical Address Space

Segment Number

	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table





- Segmentation is a memory management technique.
- In segmentation memory is partitioned into the variable size segments.
- the user gives the size of the segments.
- There may be a chance of external fragmentation.

- The logical address is partitioned into the Segment number and the Segment offset.
- The list of holes is maintained by the operating system.

Advantages of Segmentation in Operating System

- No Internal Fragmentation: since there is no fixed size partitions.
- The user's perception of physical memory is quite similar to segmentation.
- **Flexibility:** Segmentation provides a higher degree of flexibility where Segments can be of variable size, and processes can be designed to have multiple segments, allowing for more fine-grained memory allocation.

- **Sharing:** Segmentation allows for sharing of memory segments between processes. This can be useful for inter-process communication or for sharing code libraries.
- **Protection:** Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process's memory segment. This can help increase the security and stability of the system.



Thank You