



# **Operating Systems**

## **Main Memory-Part1**

Seyyed Ahmad Javadi

[sajavadi@aut.ac.ir](mailto:sajavadi@aut.ac.ir)

Spring 2022

# Chapter 9: Memory Management

---

- Background
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Swapping



# Objectives

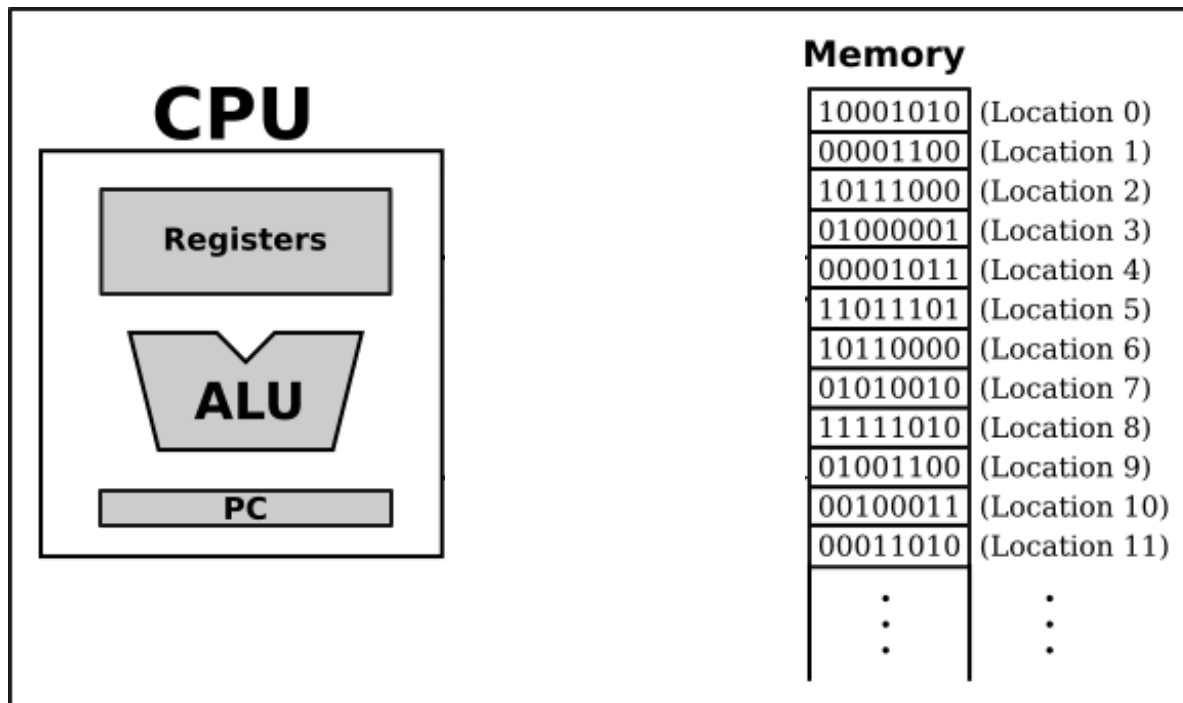
---

- To provide a detailed description of various ways of organizing memory hardware.
- To discuss various memory-management techniques.
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging.



# Background

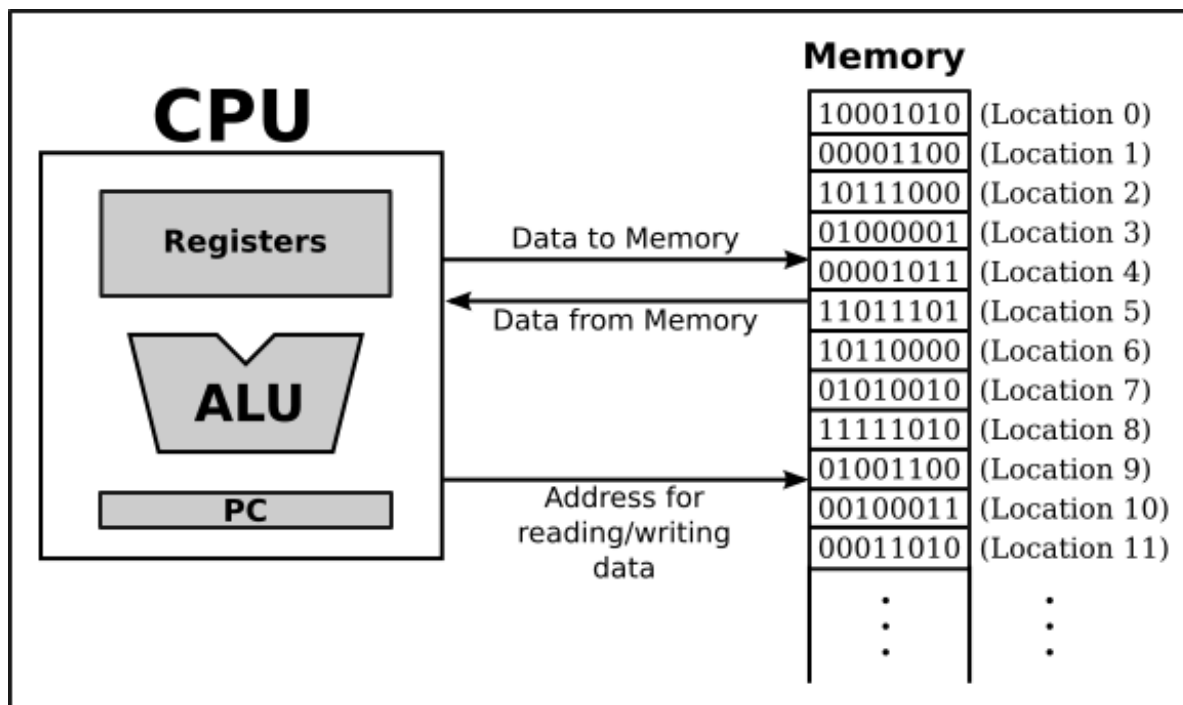
- Program must be brought (from disk) into memory and placed within a process for it to be run.
- Main memory and registers are only storage CPU can access directly.



<https://math.hws.edu/javanotes/c1/s1.html>

# Background (cont.)

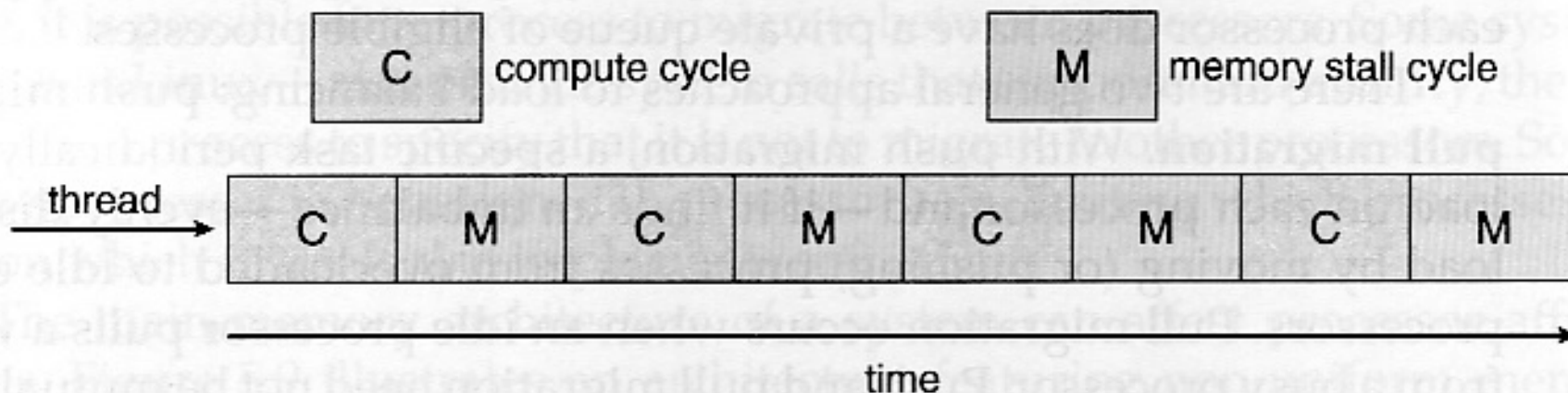
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests



<https://math.hws.edu/javanotes/c1/s1.html>

# Background (cont.)

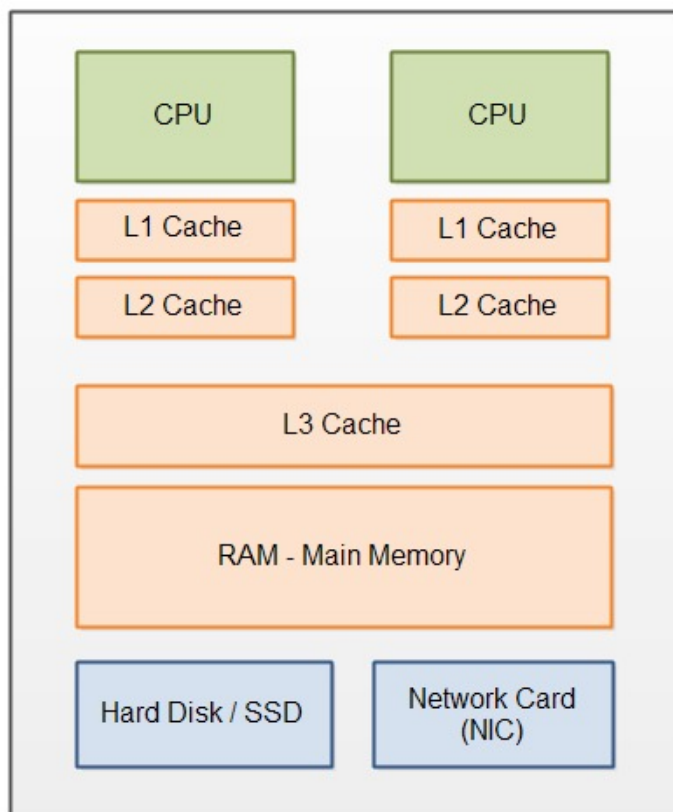
- Register access is done in one CPU clock (or less)
- Main memory can take many cycles, causing a **memory stall**



[https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5\\_CPU\\_Scheduling.html](https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html)

# Background (cont.)

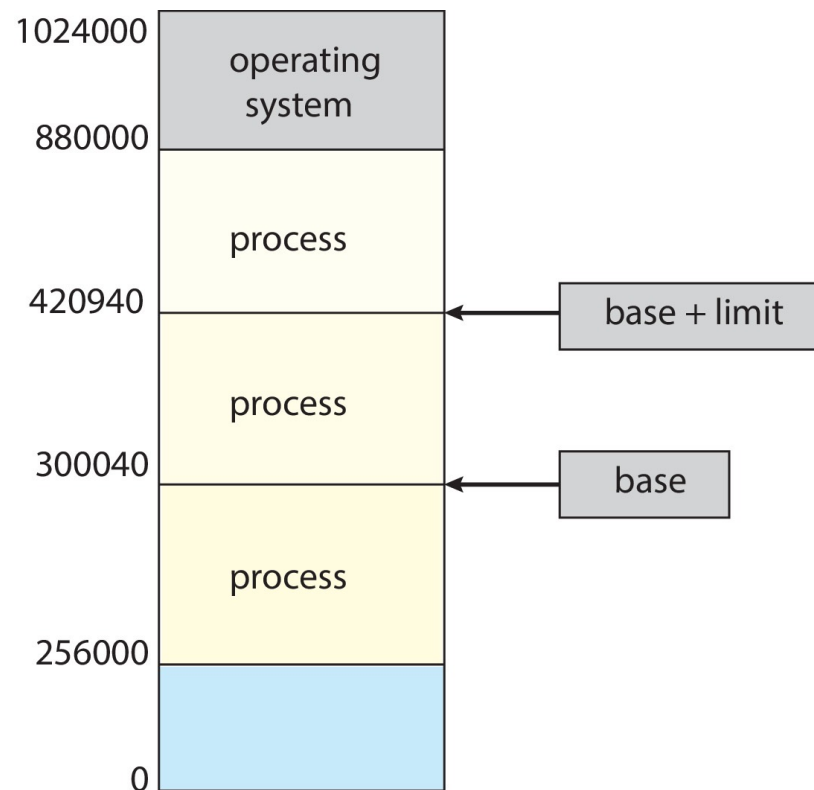
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation



<https://software.rajivprab.com/2018/04/29/myths-programmers-believe-about-cpu-caches/>

# Protection

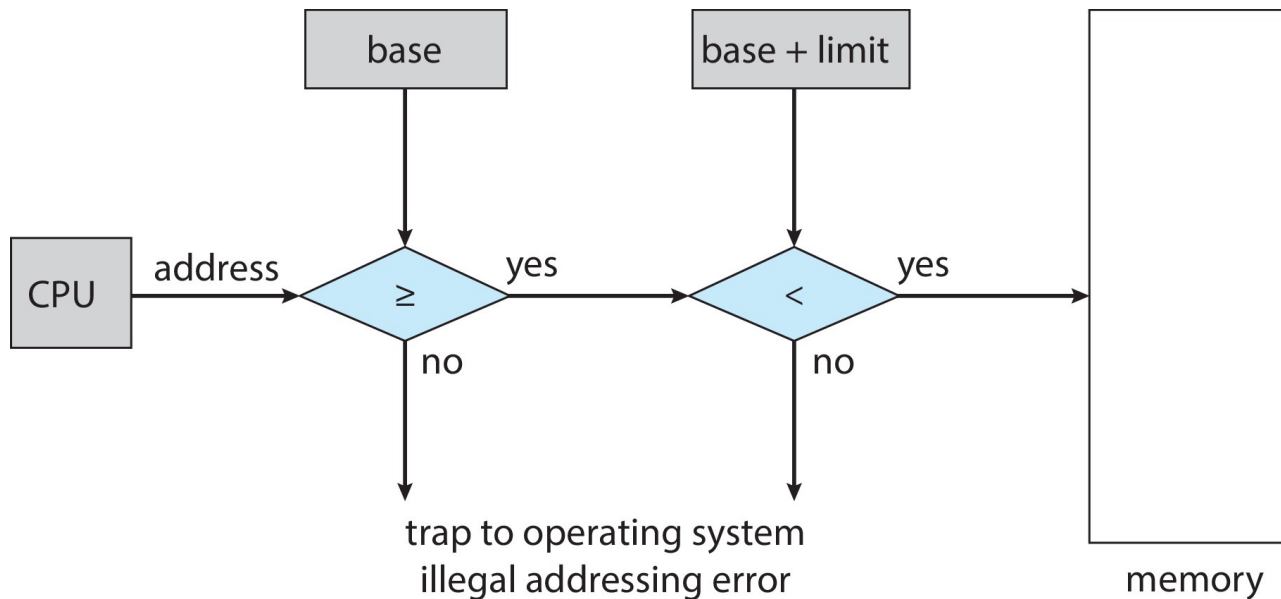
- Need to ensure that a process can access only those addresses in its address space.
- We can provide this protection by using a pair of *base* and *limit registers* define the logical address space of a process.





# Hardware Address Protection

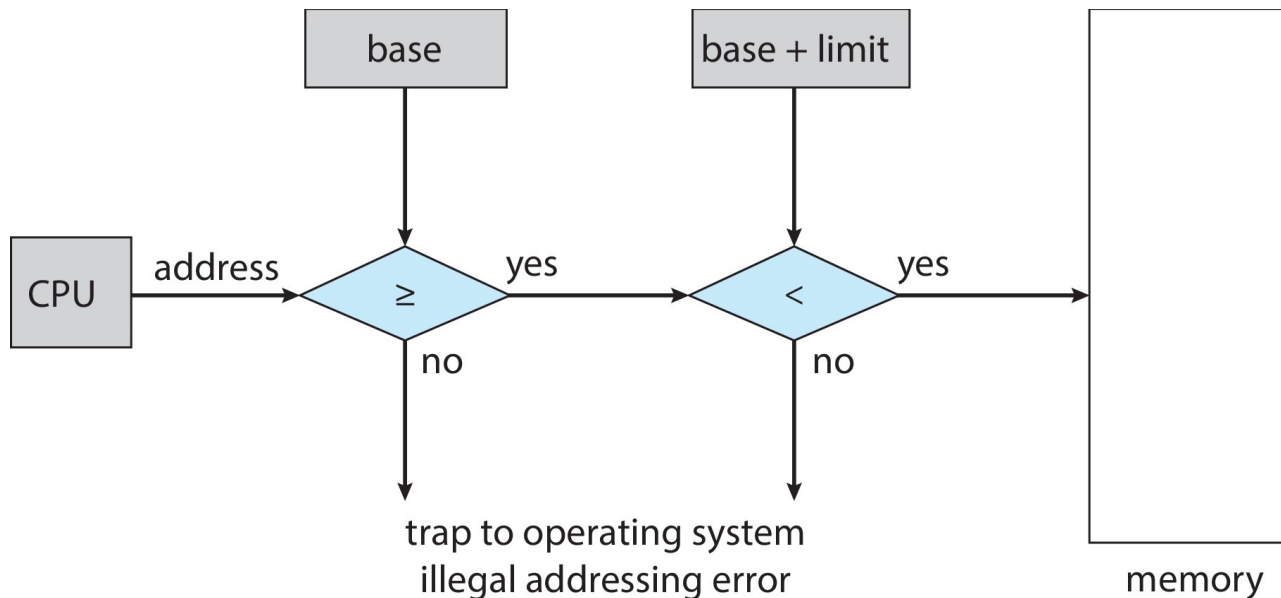
- CPU must check every memory access generated in user mode to be sure it is between **base** and **limit** for that user



- The instructions to loading the base and limit registers are **privileged**.

# Hardware Address Protection (cont.)

- This scheme prevents a user program from ***modifying the code or data structures*** of either the operating system or other users:
  - Accidentally or deliberately



# Address Binding

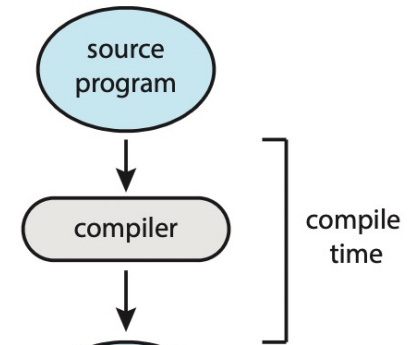
---

- Most systems allow a user process to reside in any part of the physical memory.
- Thus, although the address space of the computer may start at 00000, the first address of the user process need not be 00000.
- In most cases, a user program goes through several steps.



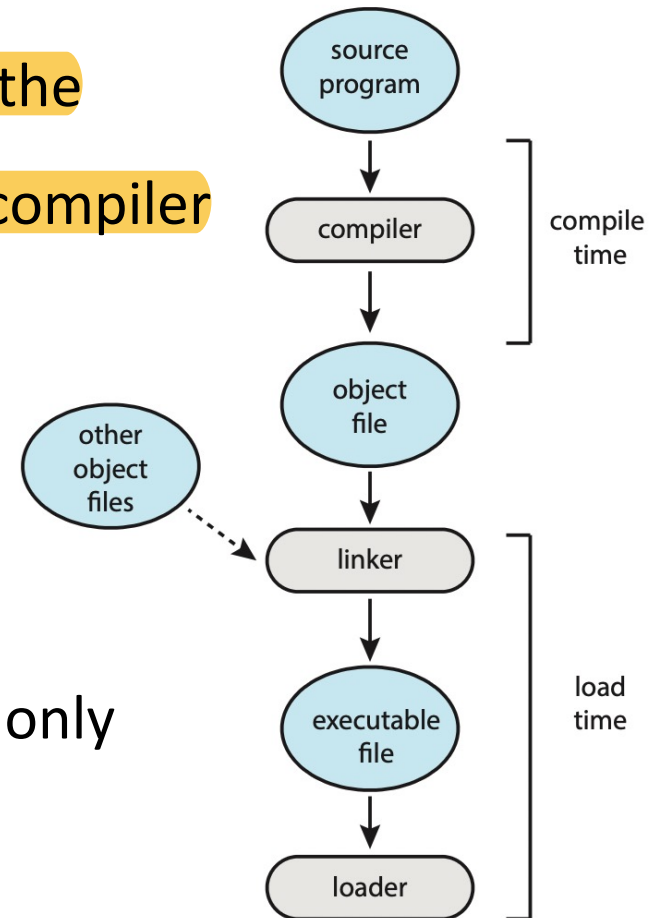
# Compile Time

- If you know at compile time where the process will reside in memory, then ***absolute code*** can be generated.
  - E.g., if a user process will reside starting at location ***R***, then the generated compiler code will start at ***R*** and extend up from there.
  - If, at some later time, the starting location changes, then it will be necessary to ***recompile*** this code.



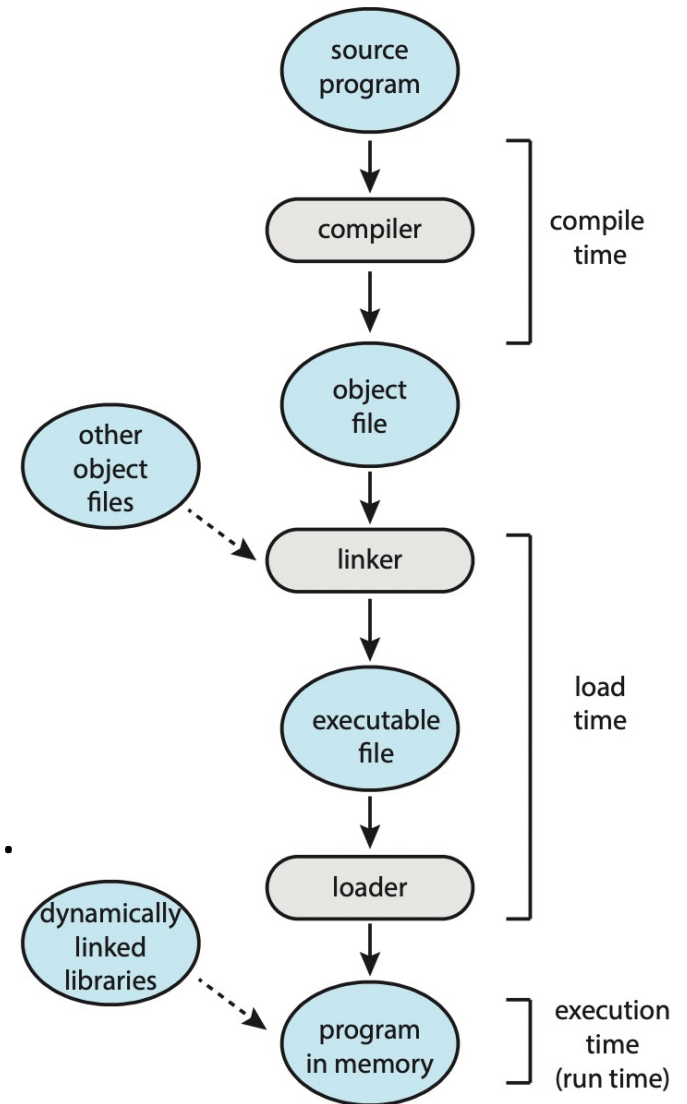
# Load Time

- If it is ***not known*** at compile time where the process will reside in memory, then the compiler must generate ***relocatable code***.
- In this case, final binding is delayed until load time.
- If the starting address changes, we need only reload the user code to incorporate this changed value.



# Execution Time

- If the process can be ***moved during its execution*** from one memory segment to another, then binding must be delayed until ***run time***.
- Special hardware must be available for this scheme to work.
- Most operating systems use this method.



# 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 vs. Physical Address Space (cont.)

---

## ■ Compile time

Logical Address	(= , !=)?	Physical address
-----------------	-----------	------------------

## ■ Load time

Logical Address	(= , !=)?	Physical address
-----------------	-----------	------------------

## ■ Execution time

Logical Address	(= , !=)?	Physical address
-----------------	-----------	------------------

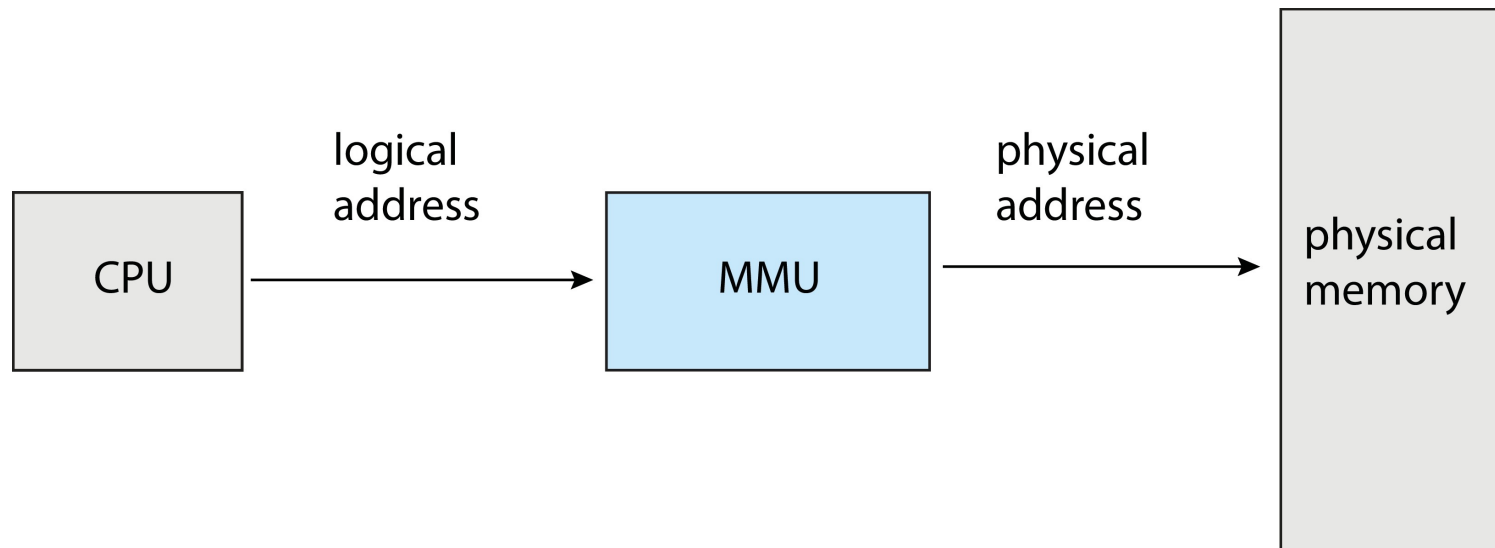




# Memory-Management Unit (MMU)

---

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



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

# Memory-Management Unit (cont.)

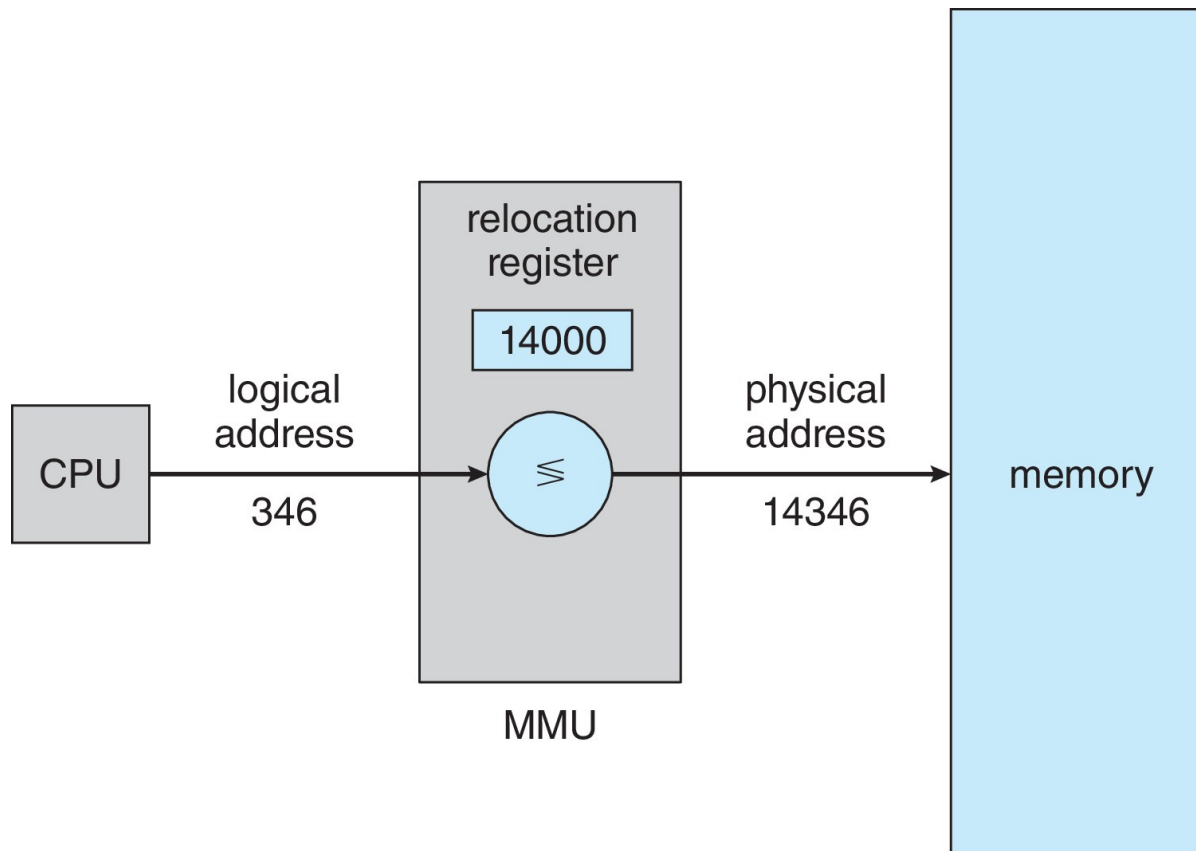
---

- Consider simple scheme.
  - Which is a generalization of the base-register scheme.
  
- The base register now called **relocation register**.



# Memory-Management Unit (cont.)

- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.



# Memory-Management Unit (cont.)

---

- 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



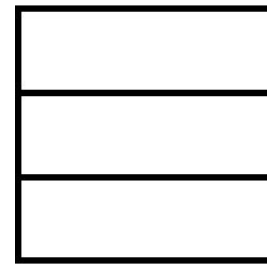
---

# Contiguous Allocation



# Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is **one early method**



Process

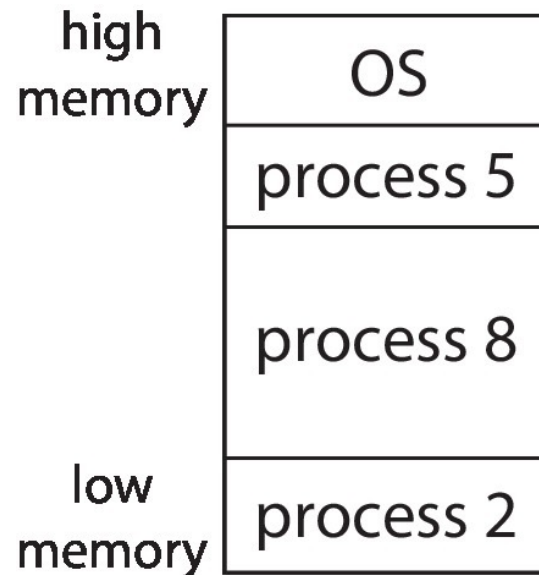
<https://www.geeksforgeeks.org/difference-between-contiguous-and-noncontiguous-memory-allocation/>



Memory blocks

# Contiguous Allocation (cont.)

- Main memory usually into two **partitions**:
  - Resident operating system
  - Each process contained in single contiguous section of memory



# Contiguous Allocation (cont.)

---

- Operating system placement
  - Low memory addresses
  - High memory addresses
- This decision depends on many factors
  - Such as the location of the interrupt vector.
- Many operating systems (including Linux and Windows) place the operating system in high memory.



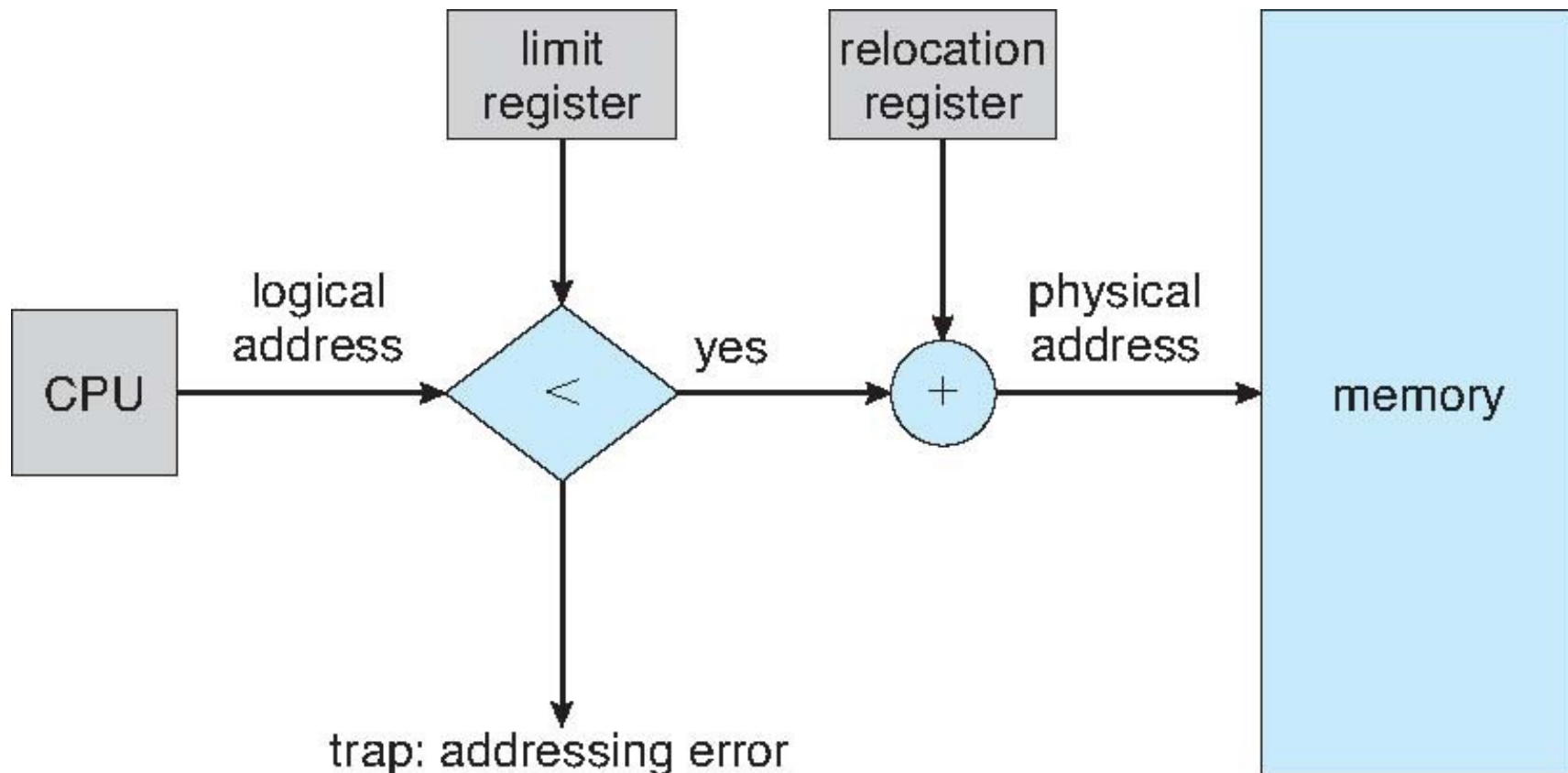
# Contiguous Allocation (cont.)

---

- **Relocation registers** used to protect user processes from each other, and from changing operating-system code and data
  - **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*

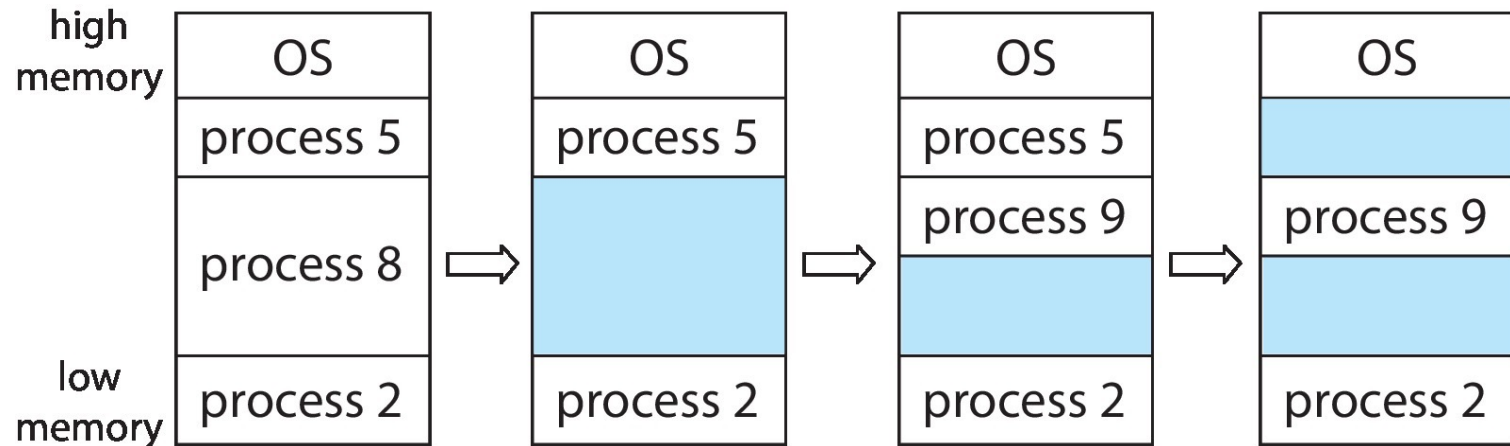


# Hardware Support for Relocation and Limit Registers



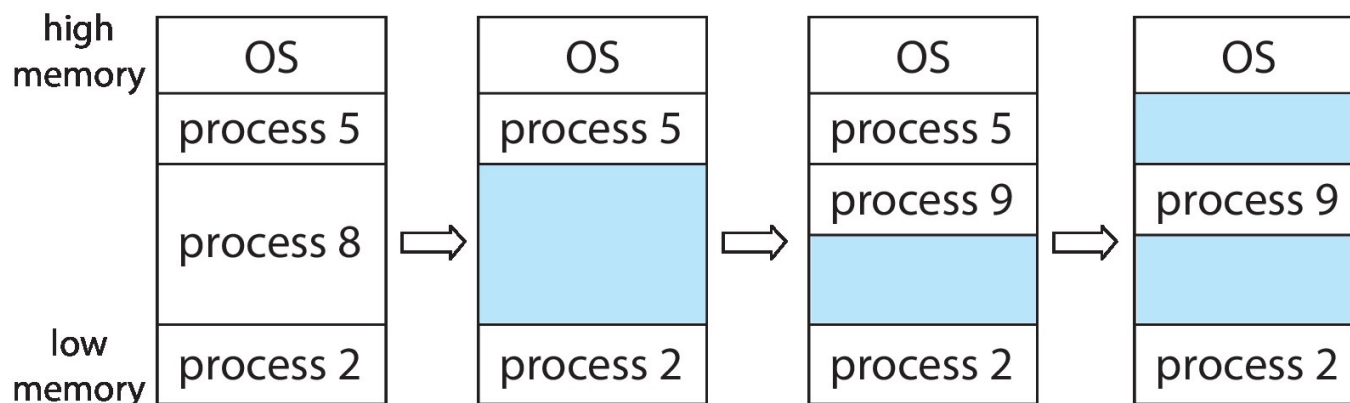
# Multiple-partition Allocation

- Degree of multiprogramming limited by number of 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



# Multiple-partition Allocation (cont.)

- 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 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

# Dynamic Storage-Allocation Problem

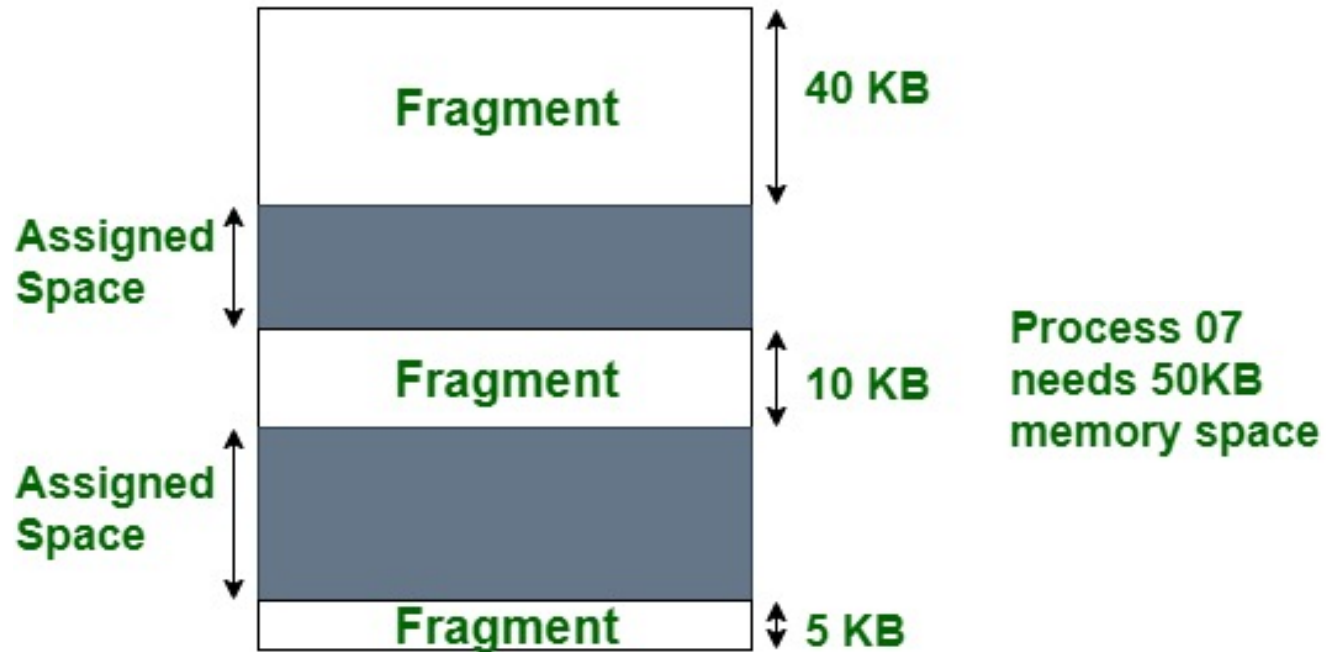
---

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

# Fragmentation

## ■ External Fragmentation

- Total memory space exists to satisfy a request, but it is not contiguous.

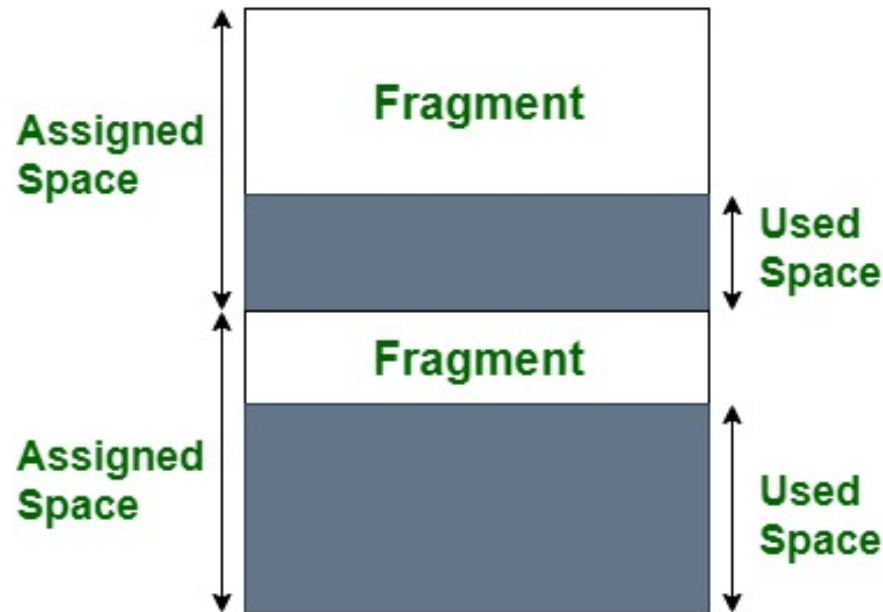


<https://www.geeksforgeeks.org/difference-between-internal-and-external-fragmentation/>

# Fragmentation (cont.)

## ■ Internal Fragmentation

- Allocated memory may be slightly larger than requested memory.
- This size difference is memory internal to a partition, but not being used.



<https://www.geeksforgeeks.org/difference-between-internal-and-external-fragmentation/>



# Fragmentation (cont.)

---

- First fit analysis reveals that given  $N$  blocks allocated, another  $0.5 * N$  blocks lost to fragmentation
  - $1/3$  may be unusable -> **50-percent rule**



# Fragmentation (cont.)

---

- 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.



# Fragmentation (cont.)

---

- The simplest compaction algorithm is to move all processes toward one end of memory.
- All holes move in the other direction, producing one large hole of available memory.
- This scheme can be expensive.
- Now consider that backing store has same fragmentation problems.

