CSCI 4730/6730 OS
(Chap #9 Main Memory)

In Kee Kim

Department of Computer Science

University of Georgia

General OS Class

- Process, Thread
- CPU Scheduling
- Threads/Process Synchronization
- Memory Management
- Virtual Memory
- Storage Management
- File Systems
- Advanced Topics
  - Security, Virtualization, Networked and Distributed Systems

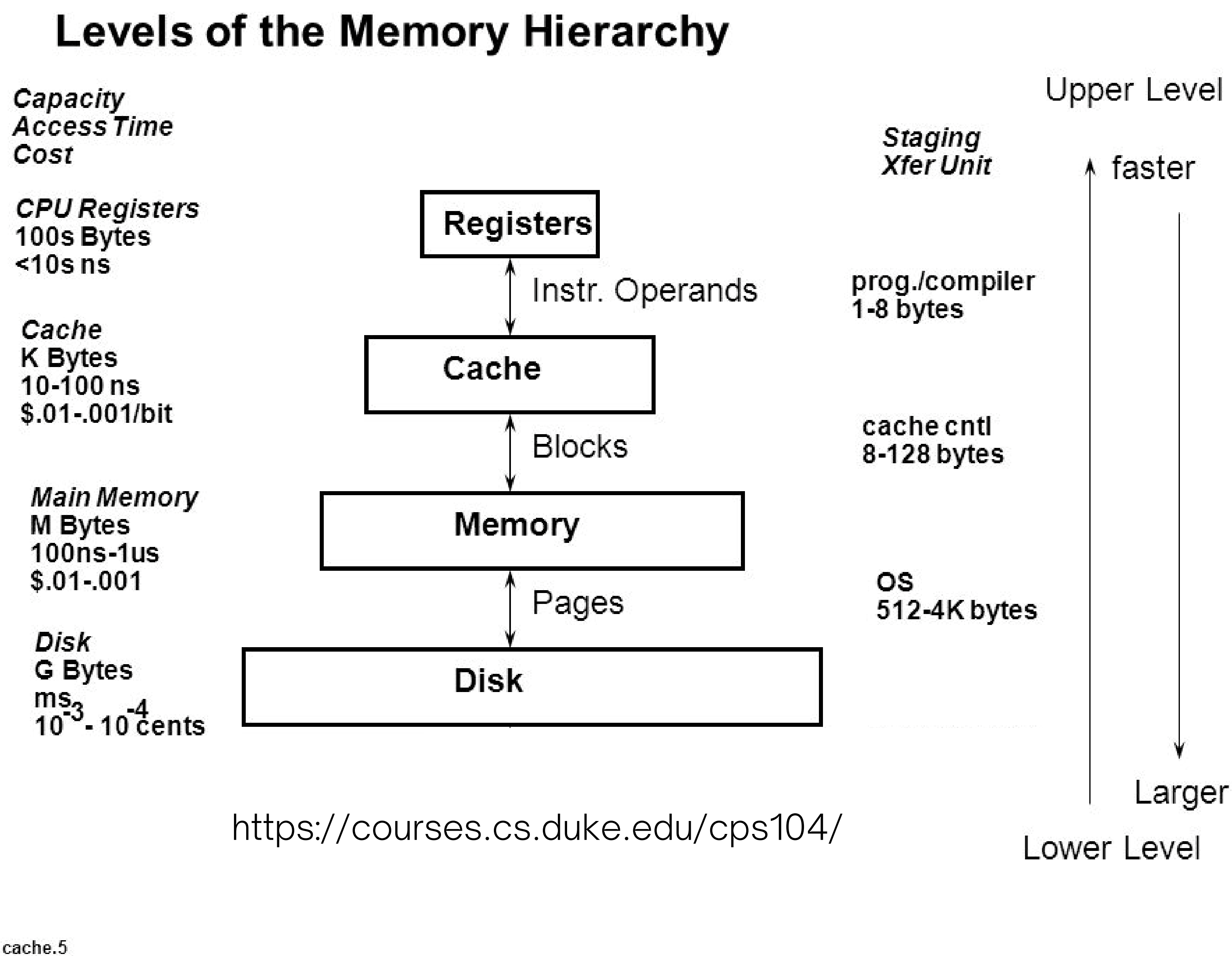
Chapter 8: Main Memory

- Background
- Contiguous Memory Allocation
- Paging and Page Table

Main Memory Questions

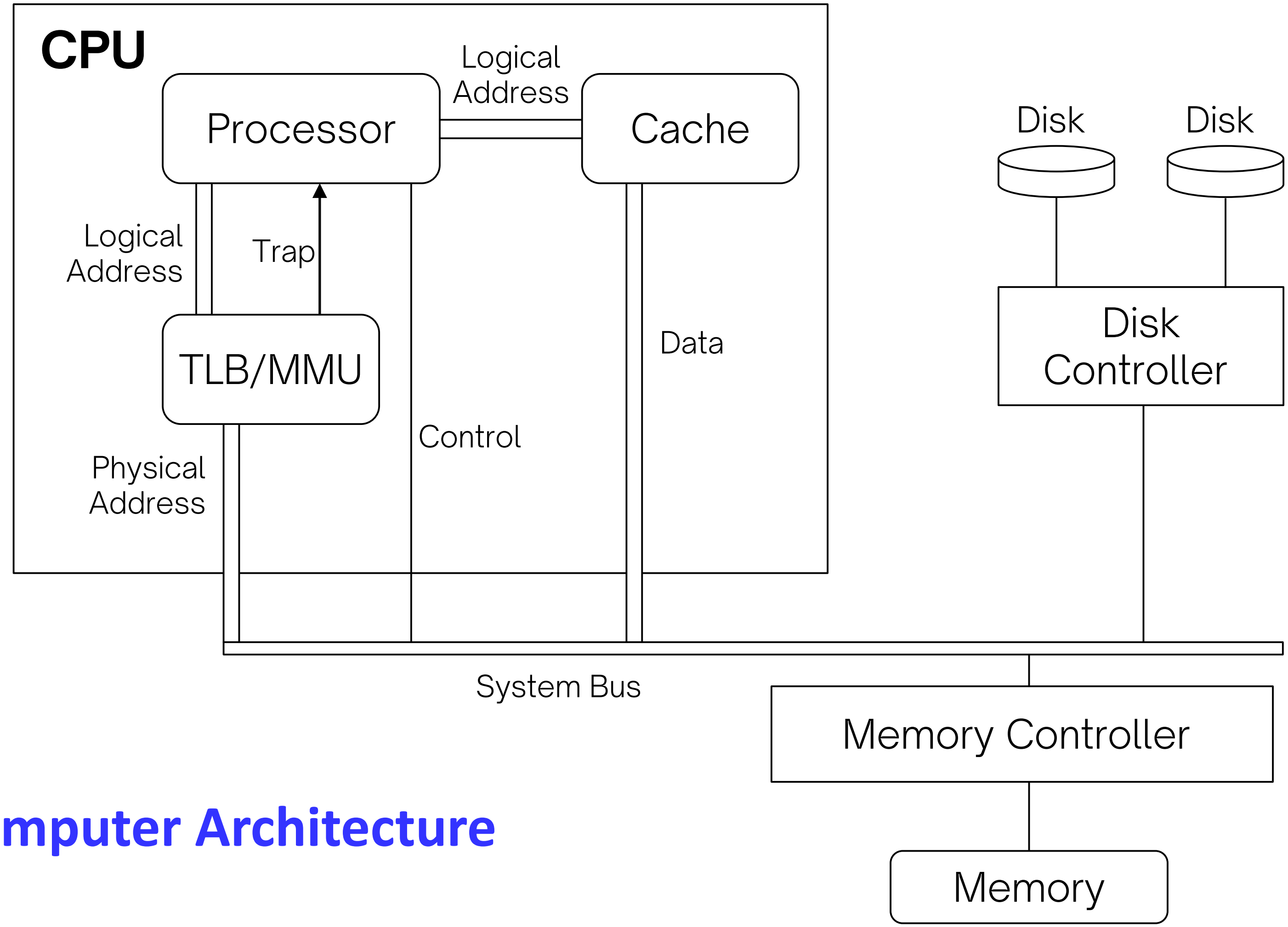
- Where is the executing processes (or programs)?
- What is main memory?
- How does OS allow multiple process to use main memory simultaneously
- What is an address in memory and how is it interpreted?

Background



https://courses.cs.duke.edu/cps104/

Background



Computer Architecture

Background

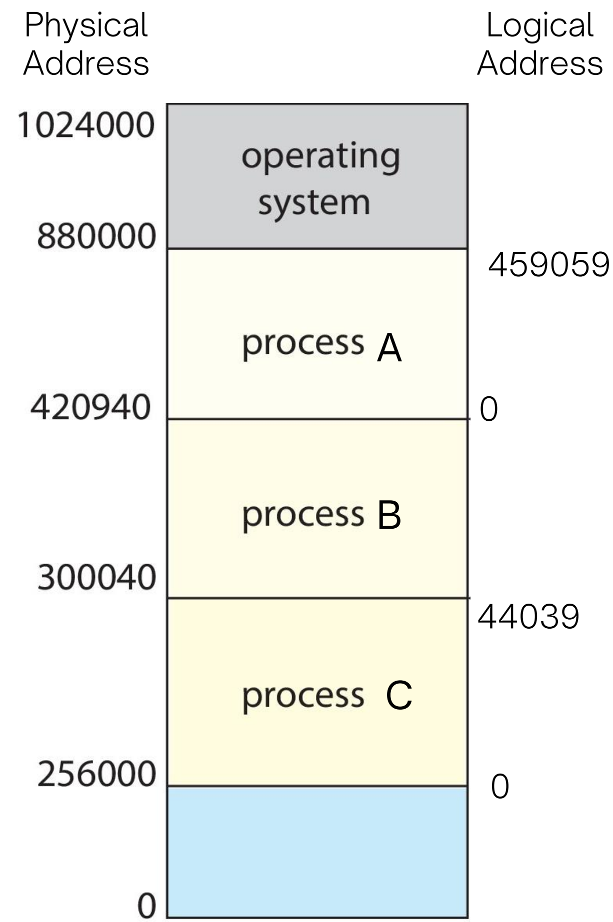
- Program must start on **where???**
- OS loads the program into **where???**
- CPU fetches instructions and data from **where???** while executing the program
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests
- Register access is done in one CPU clock (or less)

Background

- Completing memory access can take multiple cycles
  - **Processor stall**
  - Processor does not have the data required to complete the instruction.
- **Cache** sits between main memory and CPU registers
  - Fast, on the CPU chip
- Protection of memory required to ensure correct operation (we will revisit this)

Memory Management: Terminology

- **Segment**: a chunk of memory assigned to a process
  - Contiguous allocation
- **Physical address**: a real address in memory
- **Logical (Virtual) address**: an address relative to the start of a processor's address space



Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **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 generated by a program

BTW, Why do we need logical address?

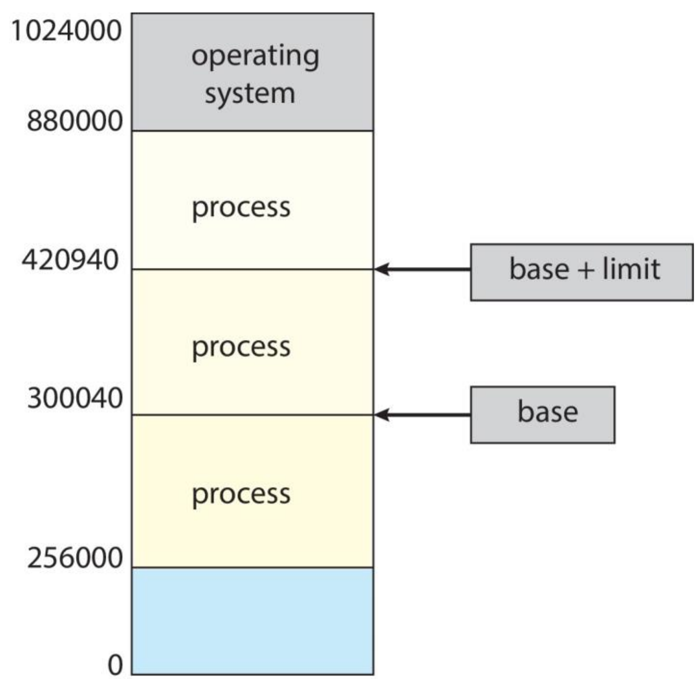
- Why not simply using physical address?
  - Processes can grow with dynamic memory allocation
    - e.g., malloc
  - Virtual memory has no idea about actual physical limit
    - processes should have illusion that they can obtain more memory space

BTW, Size Comparison

- Logical address space != Physical address space
- Logical address space == Physical address space
- Q. If your machine has 62-bit Ubuntu 20.04 LTS on Intel i9-11900K, 64GB RAM, and 1T SSD.
- Logical address space ?= Physical address space

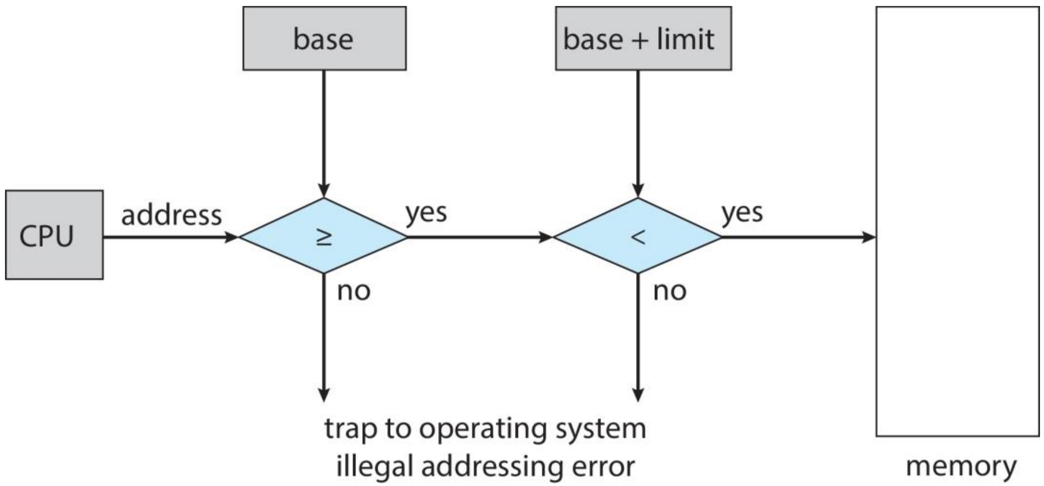
Memory Management: Protection

- Need to ensure that a process can access only access those addresses in its address space.
- OS 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



- Kernel can change values for base and limit register
  - The instructions to loading the base and limit registers are ***privileged***
  - Why/When OS needs to change base and limit register?

Address Binding

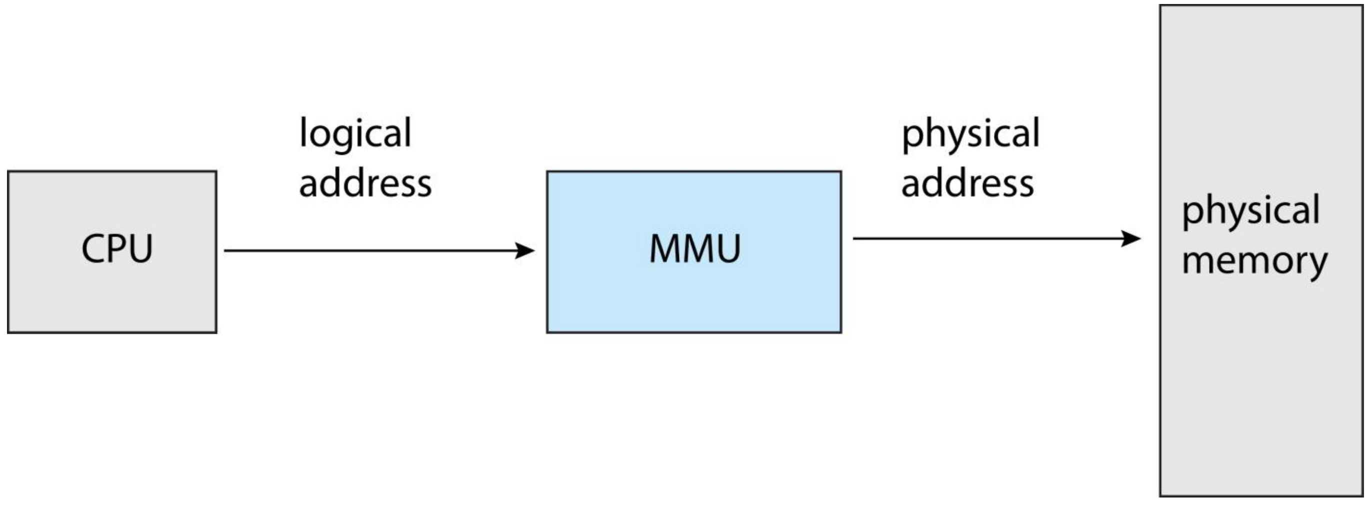
- When will a program's address be determined?
- Three stages
  - Compile Time (by compiler)
  - Loading Time (by loader and linker)
  - Execution Time (by OS)

Address Binding

- **Compile Time**
  - Compiler generates the exact physical location in memory (**absolute code**). OS does nothing.
- **Loading Time**
  - Compiler generates an address (**relocatable code**), but at load time, loader and linker determine the process' starting position. Once the process loads, it does not move in memory
- **Execution Time**
  - Compiler generates an address, and OS can place it any where it wants in memory
  - **Need HW support (registers + MMU)**

Memory-Management Unit (MMU)

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

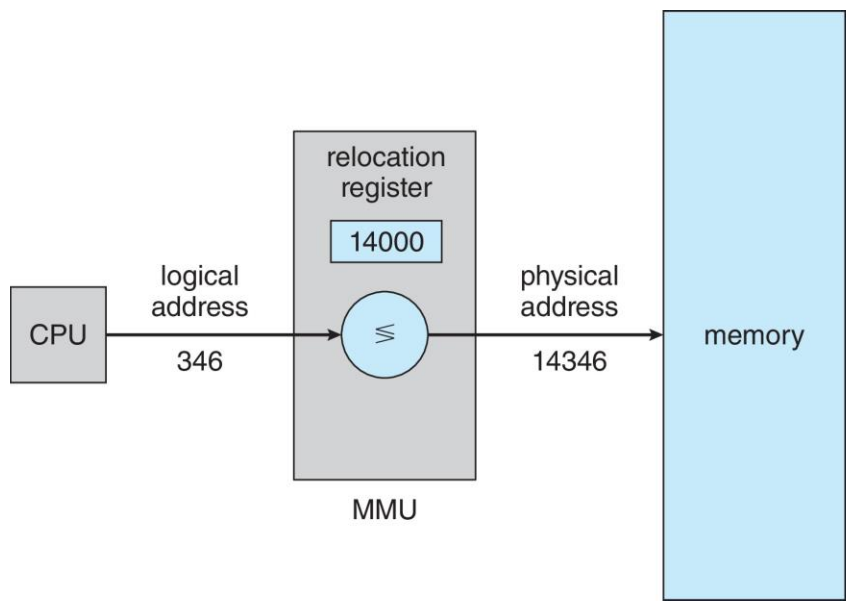


Memory-Management Unit (MMU)

- Consider simple scheme, which is a generalization of the base-register scheme.
  - The base register now 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

Memory-Management Unit (MMU)

- Consider simple scheme, which is a generalization of the base-register scheme.
  - The base register now 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



HW Support for Relocation and Limit Registers

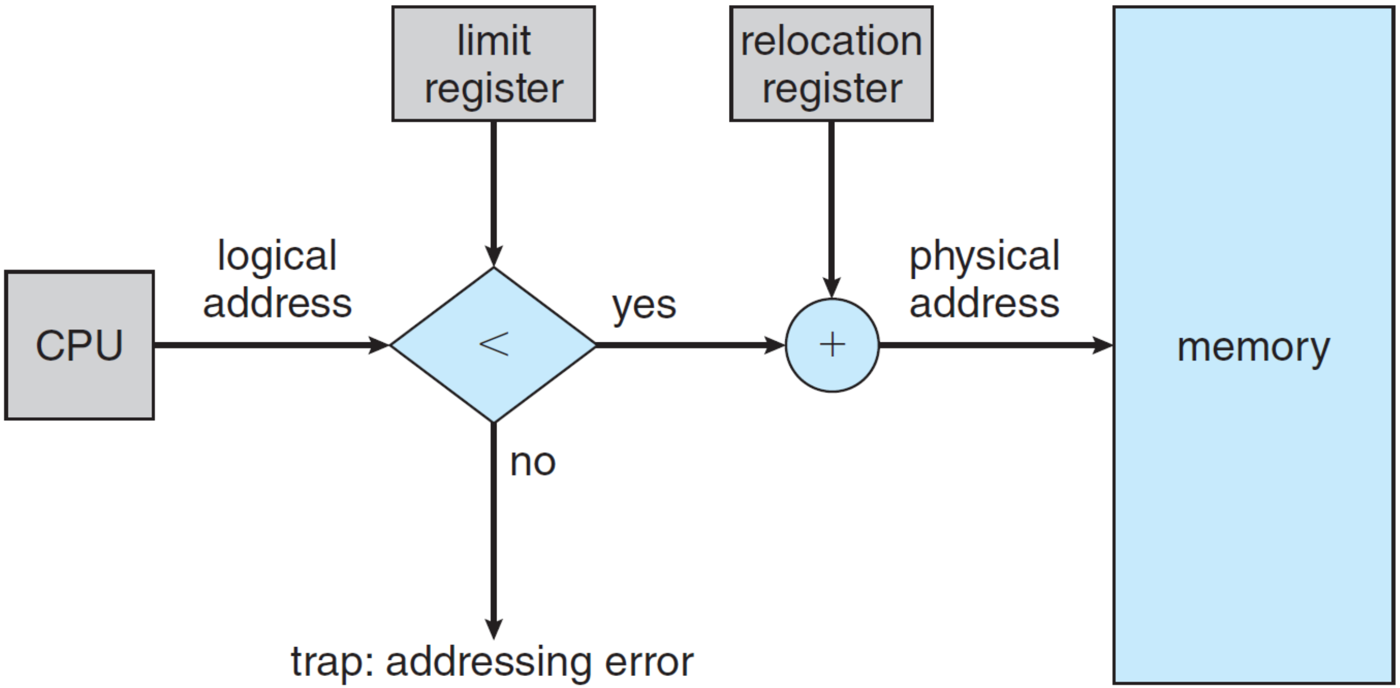


Figure 9.6 Hardware support for relocation and limit registers.

HW Support for Relocation and Limit Registers

- HW adds "relocation register" to logical address to get a physical address
- HW compares address with limit register
  - Address must be less than limit
- If test fails, the process takes an address trap and ignore the physical address

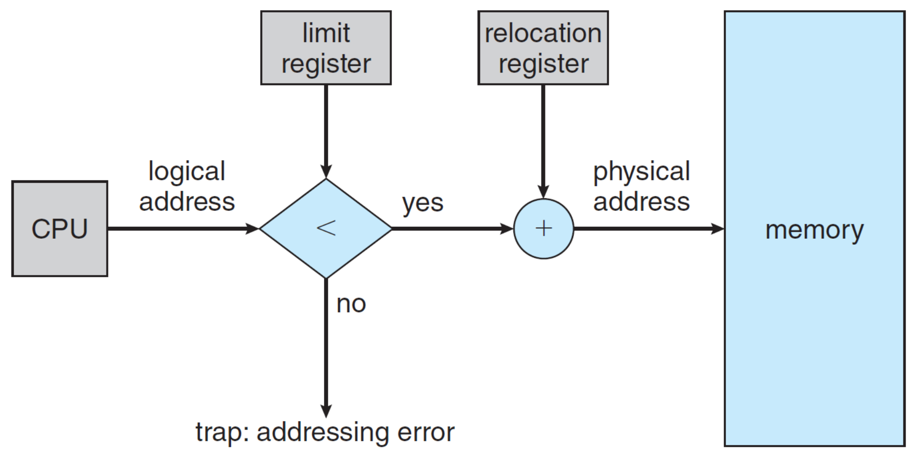
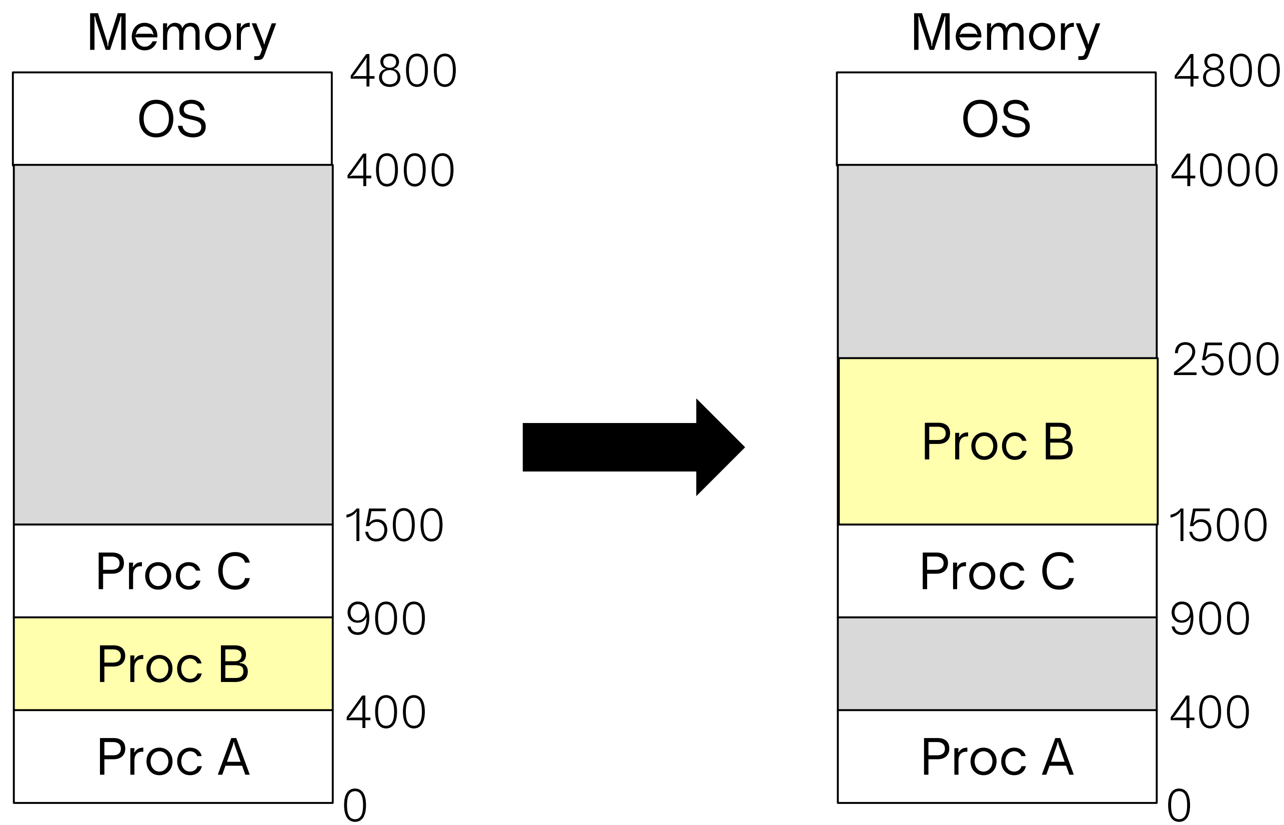
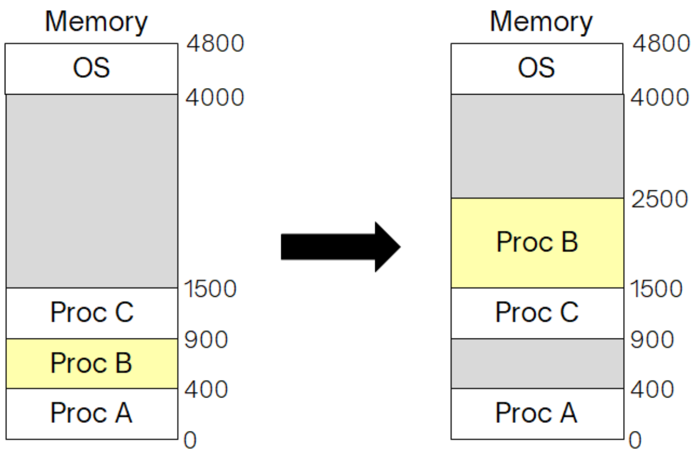


Figure 9.6 Hardware support for relocation and limit registers.

Dynamic (Process) Relocation



Dynamic Relocation



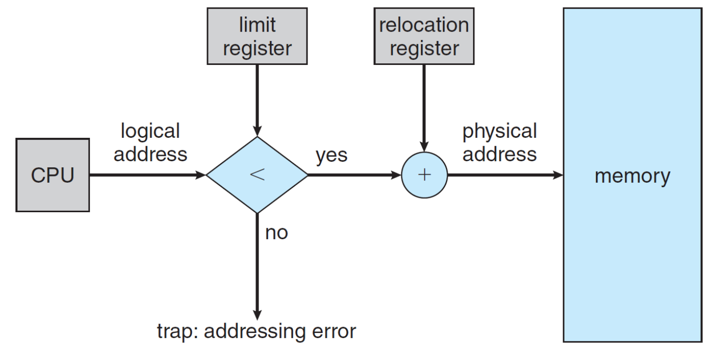


Figure 9.6 Hardware support for relocation and limit registers.

- When Proc B needs to be moved from addr 400 to addr 1500, what OS needs to do?
  - Change the value of relocation register. Proc B continues the exec.
- What if proc A needs more memory space?
  - Change the value of limit register

Pros and Cons of Dynamic Relocation

Pros	Cons
- OS can easily move a proc during its exec. - OS can allow a proc to grow over time - Simple, fast hardware: two special registers, an add, and a compare	- Slow down HW due to the add on every memory reference - Complicated memory management - # proc are limited to the size of memory

Contiguous Allocation

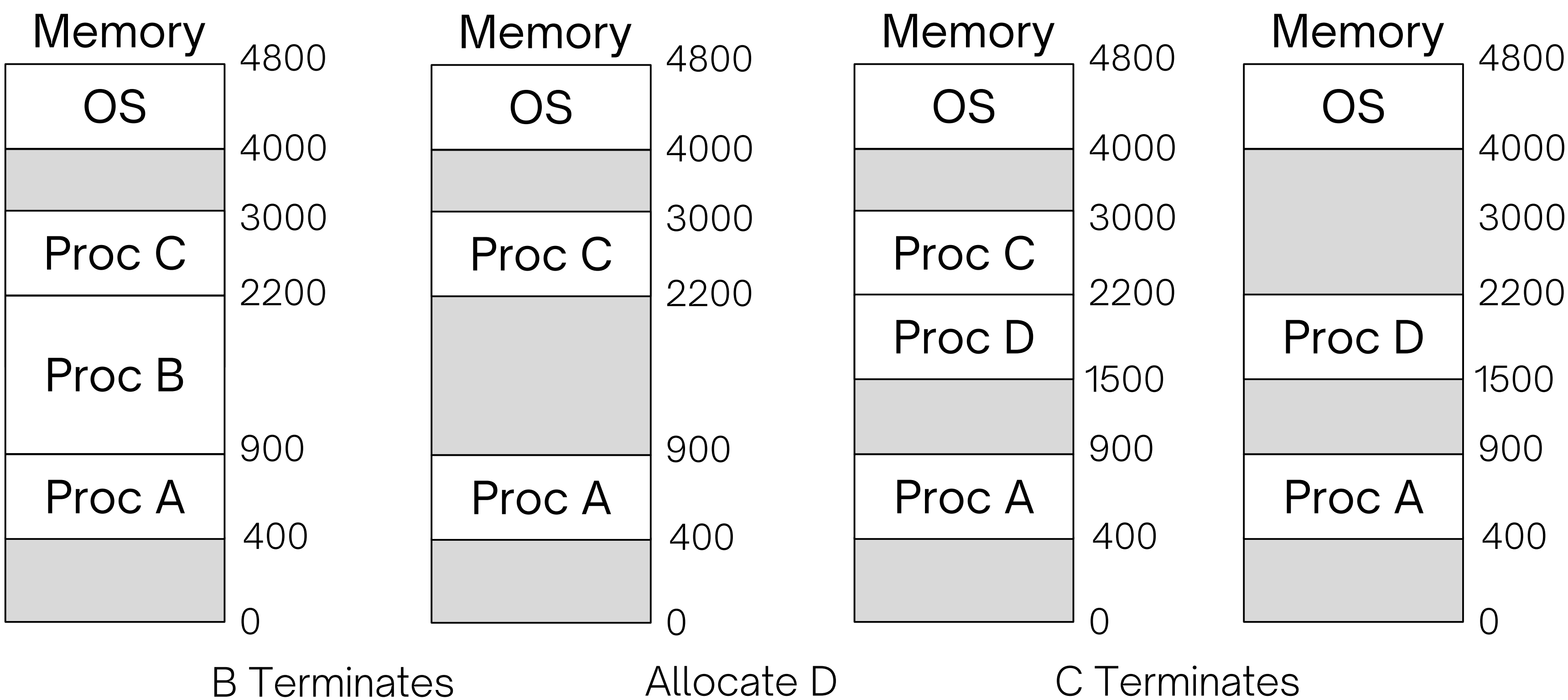
- Main memory must support both OS and user procs.
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - OS + Processes
  - Resident operating system, usually place on one end (high or low mem). User processes then held in the other end.
  - Each process contained in single contiguous section of memory

Contiguous Allocation

- Protection is mostly done by relocation register
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Relocation (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*

Memory Allocation

- **Problem –** Dynamic Nature of Processes



- As the processes enter the system, grow, and terminate, OS must keep track of which memory is available and utilized.

Variable Partition in Memory Allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **What is degree of multiprogramming? (chap 3)**
    - The number of processes currently in memory

Variable Partition in Memory Allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Q. Why variable partition sizes?**