



OPERATING SYSTEM

Memory Management

Dr Rahul Nagpal
Computer Science

OPERATING SYSTEM

Memory Management

Dr. Rahul Nagpal
Computer Science

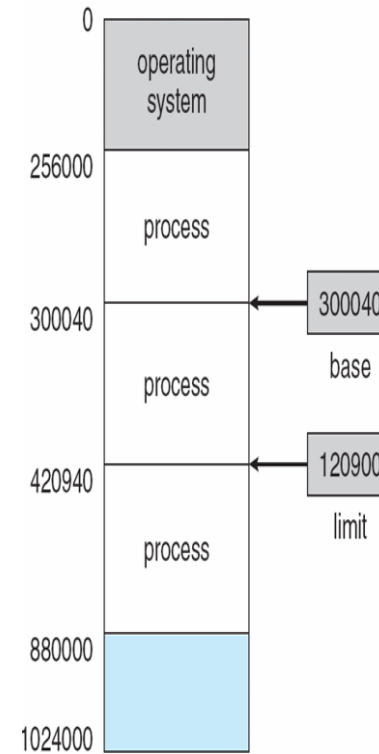
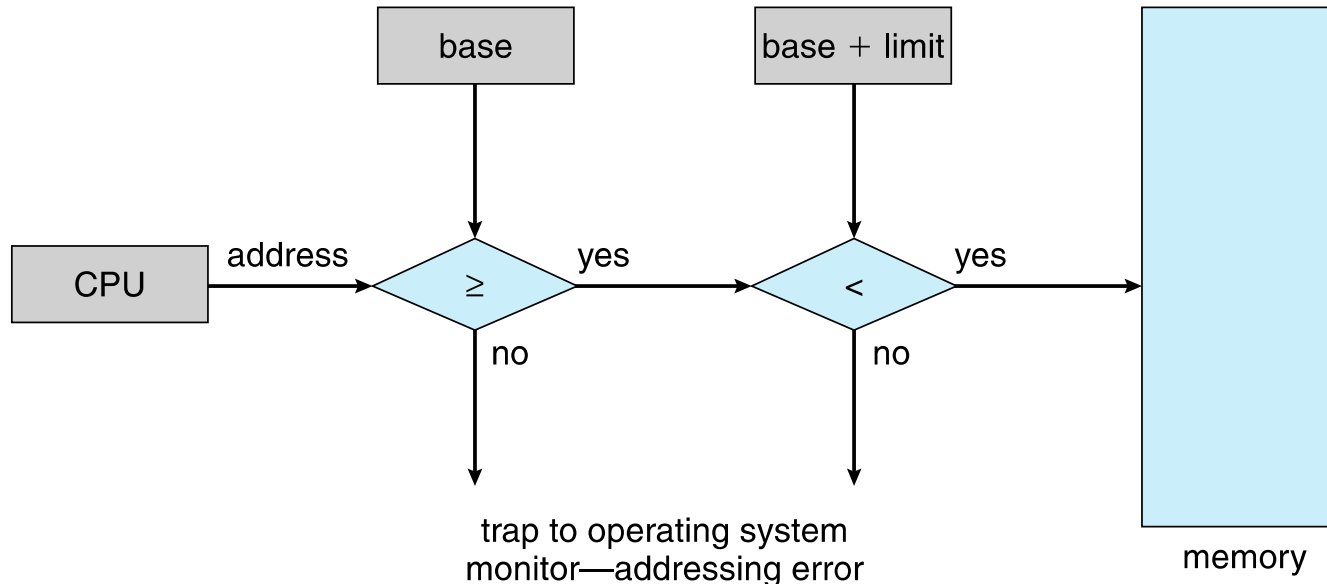
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

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

OPERATING SYSTEMS

Base, Limit Register and Address Translation

- 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

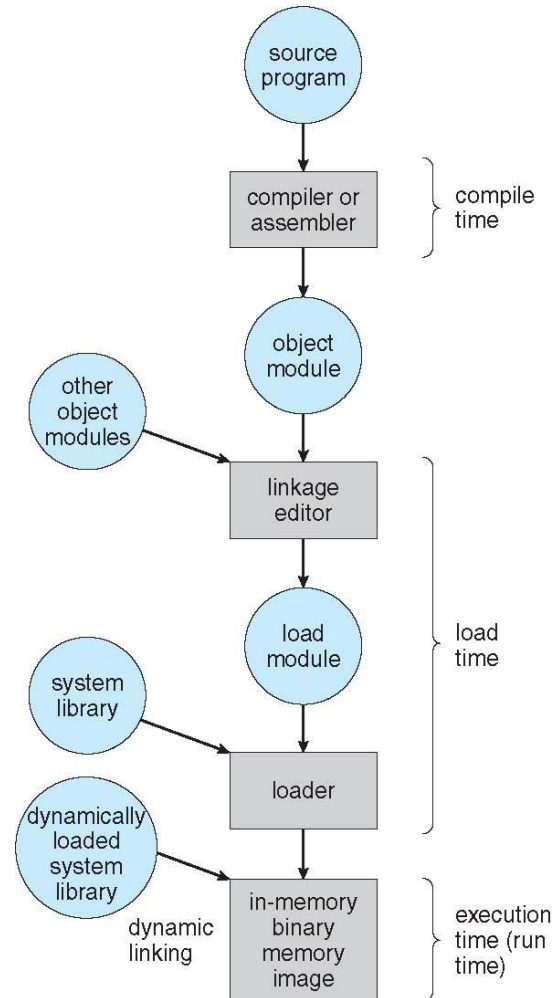


- Programs on disk, ready to be brought into memory to execute form an **input queue**
 - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
 - How can it not be?
- Further, addresses represented in different ways at different stages of a program's life
 - Source code addresses usually symbolic
 - Compiled code addresses **bind** to relocatable addresses
 - i.e. "14 bytes from beginning of this module"
 - Linker or loader will bind relocatable addresses to absolute addresses
 - i.e. 74014
 - 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)

OPERATING SYSTEMS

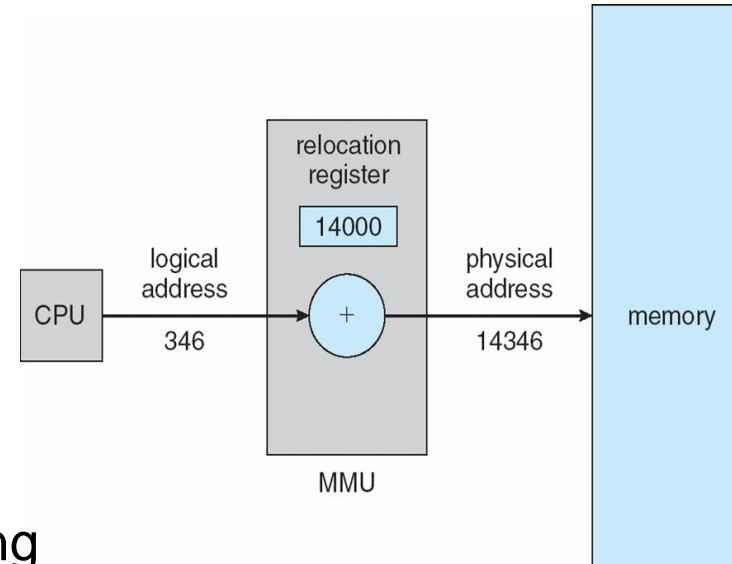
Multistep Processing of User Program



- 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 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

- Hardware device that at run time maps virtual to physical address
- Many methods possible, covered in the rest of this chapter
- 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**
 - MS-DOS on Intel 80x86 used 4 relocation registers
- 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

- ❑ Routine is not loaded until it is called
- ❑ Better memory-space utilization; unused routine is never loaded
- ❑ All routines kept on disk in relocatable load format
- ❑ Useful when large amounts of code are needed to handle infrequently occurring cases
- ❑ No special support from the operating system is required
 - ❑ Implemented through program design
 - ❑ OS can help by providing libraries to implement dynamic loading





THANK YOU

Dr Rahul Nagpal

Computer Science

rahulnagpal@pes.edu