Memory Management
Lecture 1

Memory Management Requirement

Minakshi Retharekar

OPERATING SYSTEMS: INTERNALS AND DESIGN PRINCIPLES
William Stalling 6th edition

Memory Management

- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

Objectives

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

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

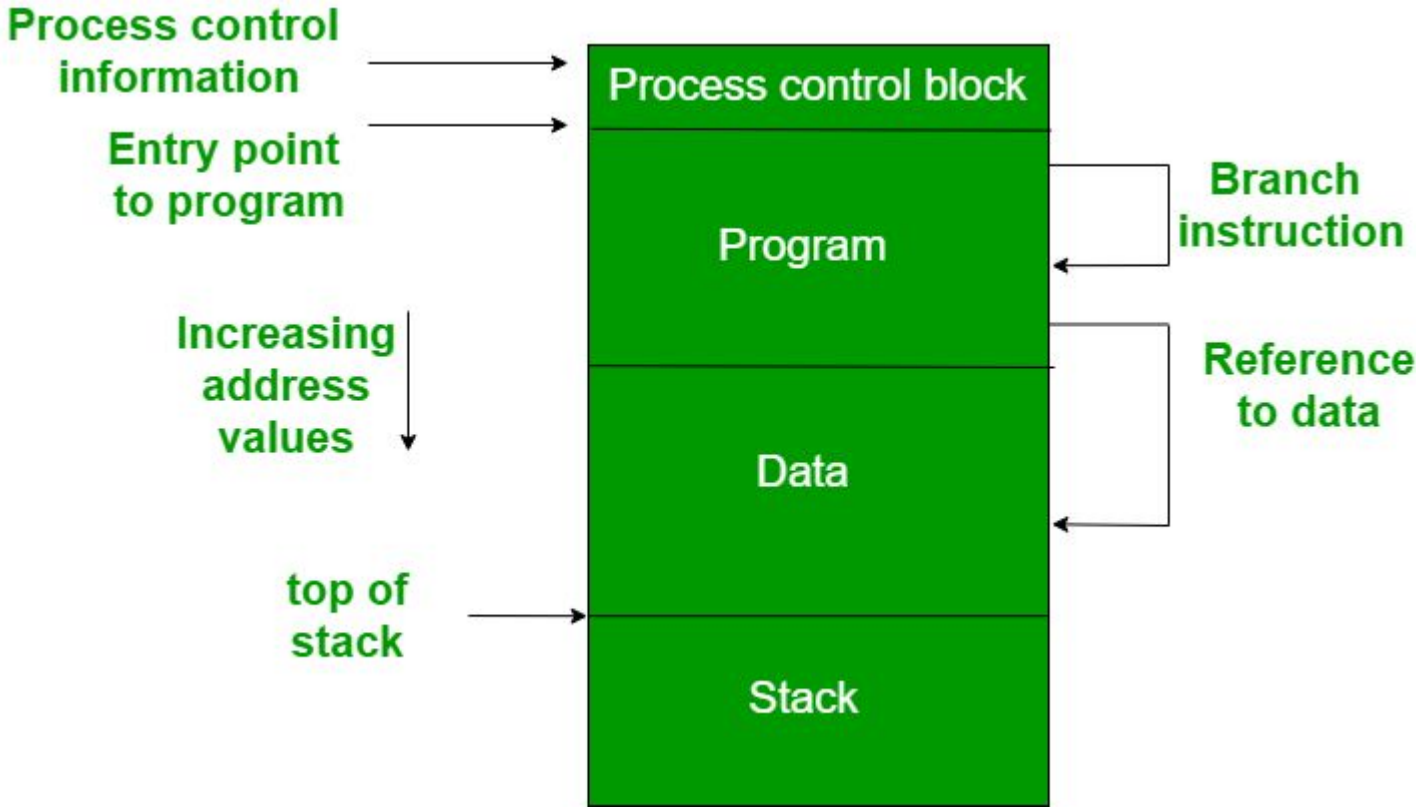
MEMORY MANAGEMENT REQUIREMENTS

- ✔ Relocation
- ✔ Protection
- ✔ Sharing
- ✔ Logical organization
- ✔ Physical organization

Requirements: Relocation

- ✔ In Multiprogramming system, the available main **memory is generally shared** among a number of processes.
- ✔ Typically, it is not possible for the programmer to know in advance which other programs will be resident in main memory at the time of execution of his or her program.
- ✔ In addition, we would like **to be able to swap active processes in and out of main memory** to maximize processor utilization by providing a large pool of ready processes to execute.
- ✔ Once a program has been swapped out to disk, it would be quite limiting to declare that when it is next swapped back in, it must be placed in the same main memory region as before.
- ✔ Instead, we may need to **relocate** the process to a different area of memory.

Relocation


Process control information
Entry point to program
Increasing address values
top of stack
Process control block
Program
Data
Stack
Branch instruction
Reference to data

Requirements: Protection

- ✔ Each process should be **protected against unwanted interference** by other processes, whether accidental or intentional.
- ✔ Thus, programs in other processes should not be able to reference memory locations in a process for reading or writing purposes without permission.

Requirements: Sharing

- ✔ Allow several processes to **access the same portion of memory**
- ✔ Better to allow each **process access to the same copy** of the program rather than have their own separate copy.
- ✔ The memory management system must therefore allow controlled access to shared areas of memory without compromising essential protection.

Requirements: Logical Organization

- ✔ Main memory in a computer system is organized as a **linear, or one-dimensional, address space**, consisting of a sequence of **bytes or words**.
- ✔ While this organization closely mirrors the actual machine hardware, it does not correspond to the way in which programs are typically constructed.
- ✔ Most programs are **organized into modules**, some of which are un-modifiable (read only, execute only) and some of which contain data that may be modified.

Requirements: Logical Organization

- ✔ **Modules** can be written and compiled independently, with all references from one module to another resolved by the system at run time.
- ✔ With modest additional overhead, **different degrees of protection** (read only, execute only) can be given to different modules.
- ✔ It is possible to introduce mechanisms by which modules can be shared among processes.

Requirements: Physical Organization

- ✔ Computer memory is organized into at least two levels, referred to as **main memory** and **secondary memory**.
- ✔ Main memory provides **fast access** at relatively **high cost**.
- ✔ In addition, main memory is **volatile**; that is, it does not provide permanent storage.
- ✔ Secondary memory is **slower** and **cheaper** than main memory and is usually not volatile.
- ✔ Thus secondary memory of large capacity can be provided for long-term storage of programs and data, while a smaller main memory holds programs and data currently in use.

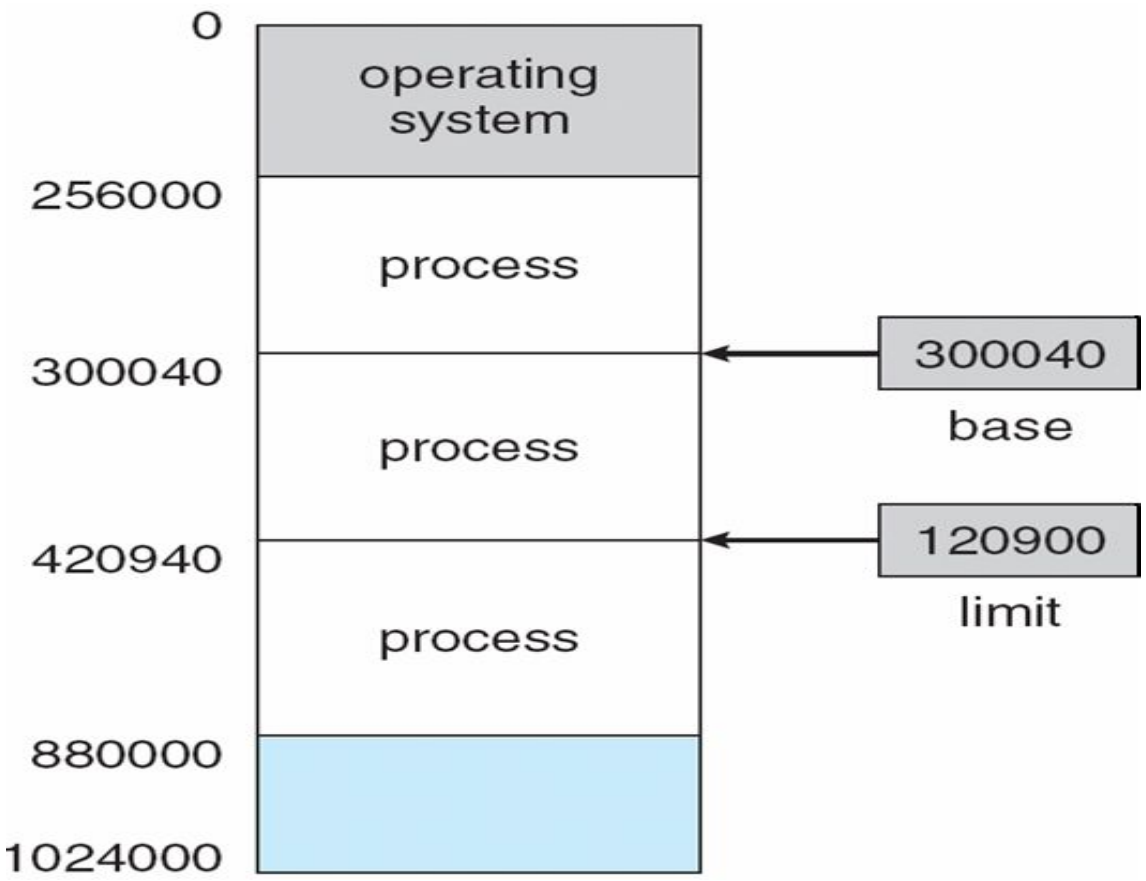
Requirements: Physical Organization

- ✔ In this two-level scheme, the organization of the flow of information between main and secondary memory is a major system concern.
- ✔ The responsibility for this flow could be assigned to the individual programmer, but this is impractical & undesirable for two reasons:

1. **The main memory available for a program plus its data may be insufficient. In** that case, the programmer must engage in a practice known as **overlaying,**
2. **In a multiprogramming environment, the programmer does not know at the** time of coding how much space will be available or where that space will be.

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



Address Binding

- The Association of program **instruction and data to the actual physical memory locations** is called the Address Binding
- 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

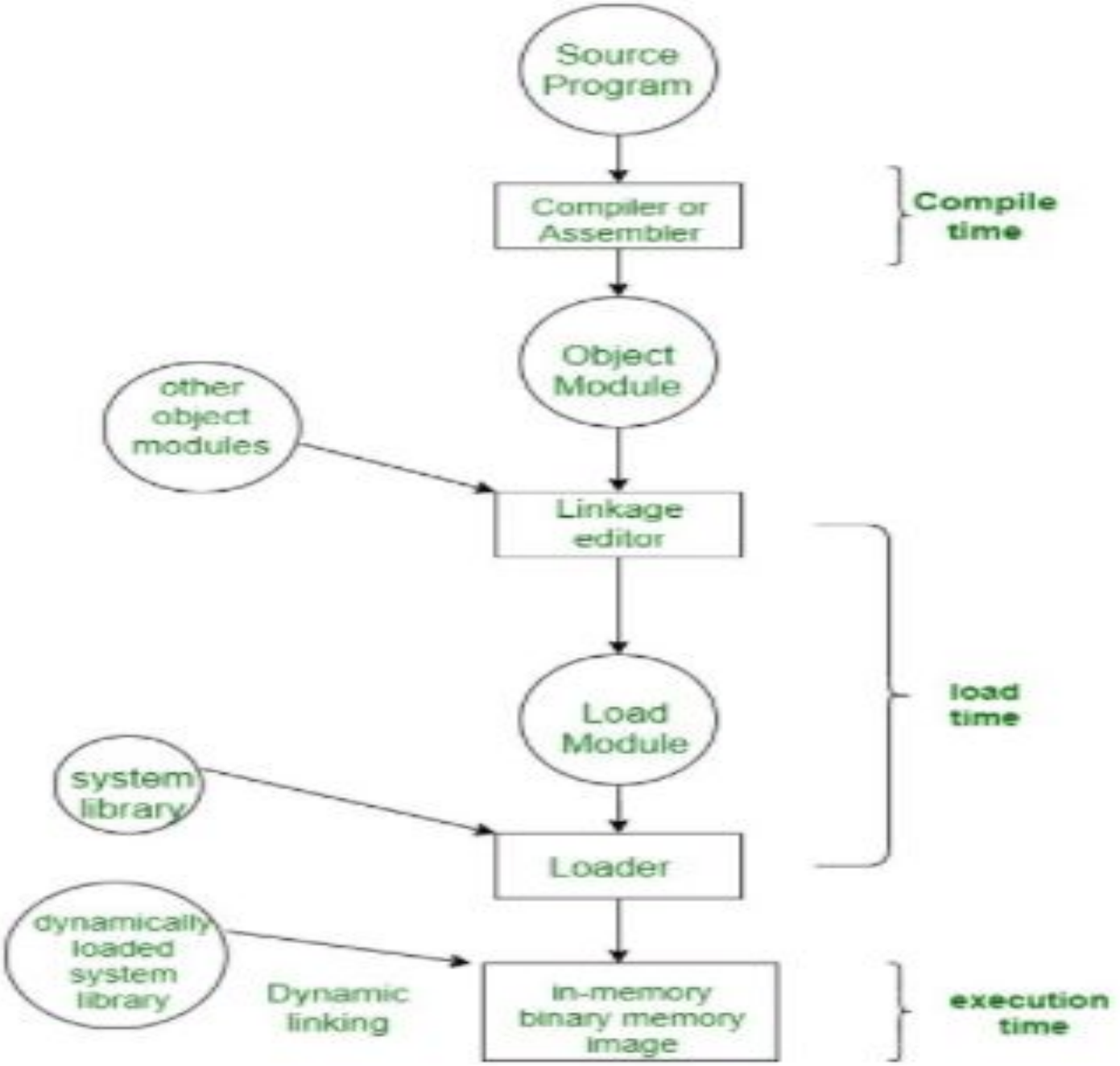
    i.e. "14 bytes from beginning of this module"
  - Linker or loader will bind relocatable addresses to absolute addresses

    i.e. 74014
  - Each binding maps one address space to another

Binding of Instructions and Data to Memory

- 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
  - In this case final binding is delayed until load time, the base address of the process in the main memory is added to all logical addresses by the loader to generate the absolute address
  - **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)


Source Program
Compiler or Assembler
Compile time
Object Module
other object modules
Linkage editor
Load Module
load time
system library
Loader
dynamically loaded system library
Dynamic linking
in-memory binary memory image
execution time

Logical vs. Physical Address Space

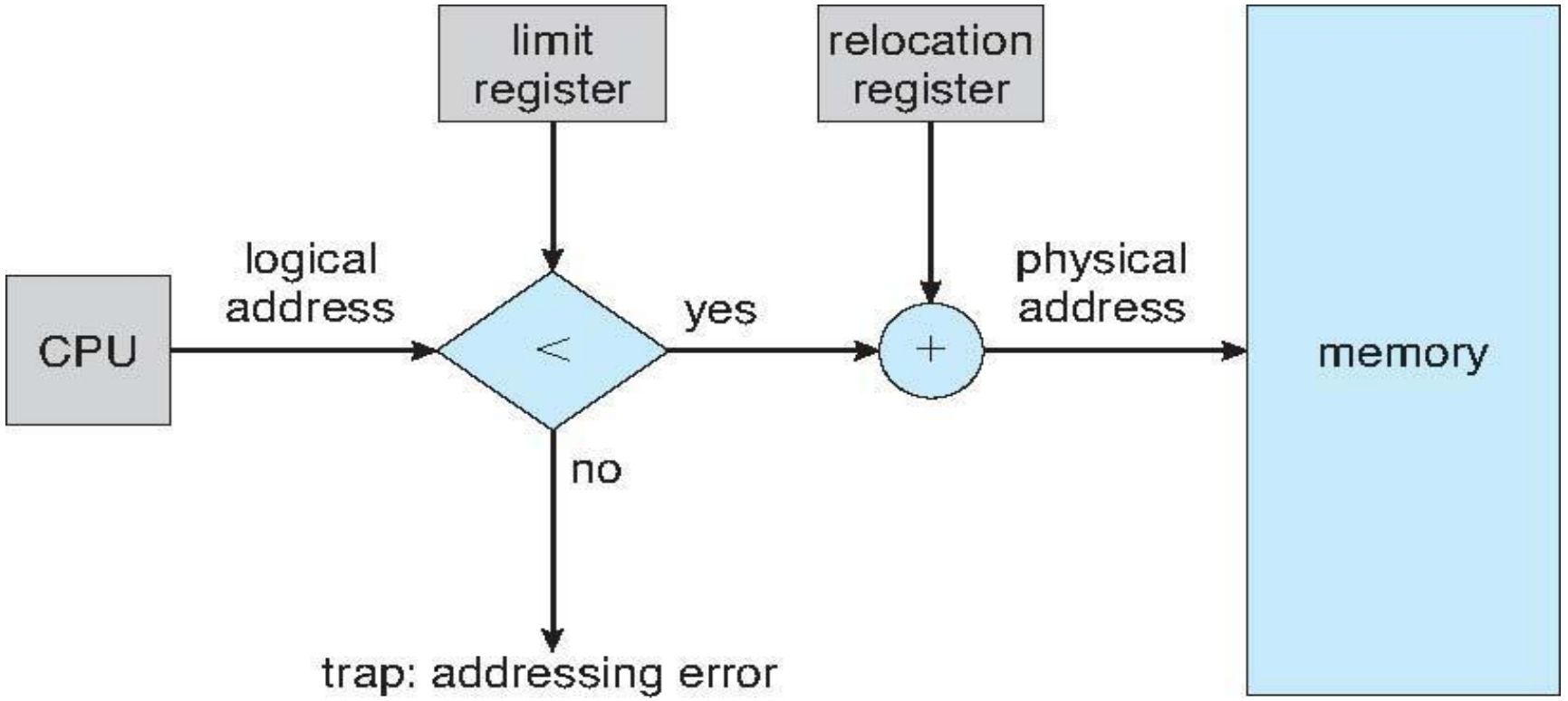
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit

- Logical 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

S.No	Logical Address	Physical Address
1.	Users can access the **logical address** of the Program.	User can never access the **physical address** of the Program
2.	The logical address is generated by the CPU.	The physical address is located in the memory unit.
3.	The user can access the physical address with the help of a logical address.	A physical address can be accessed by a user indirectly b ut not directly.
4.	The logical address does not exist physically in the memory and thus termed as a Virtual address.	On the other hand, the physical address is a location in the memory. Thus it can be accessed physically.
5.	The set of all logical addresses that are generated by any program is referred to as **Logical Address Space.**	The set of all **physical addresses** corresponding to the **Logical addresses** is commonly known as **Physical Address Space.**

Mapping Virtual Addresses to Physical Addresses

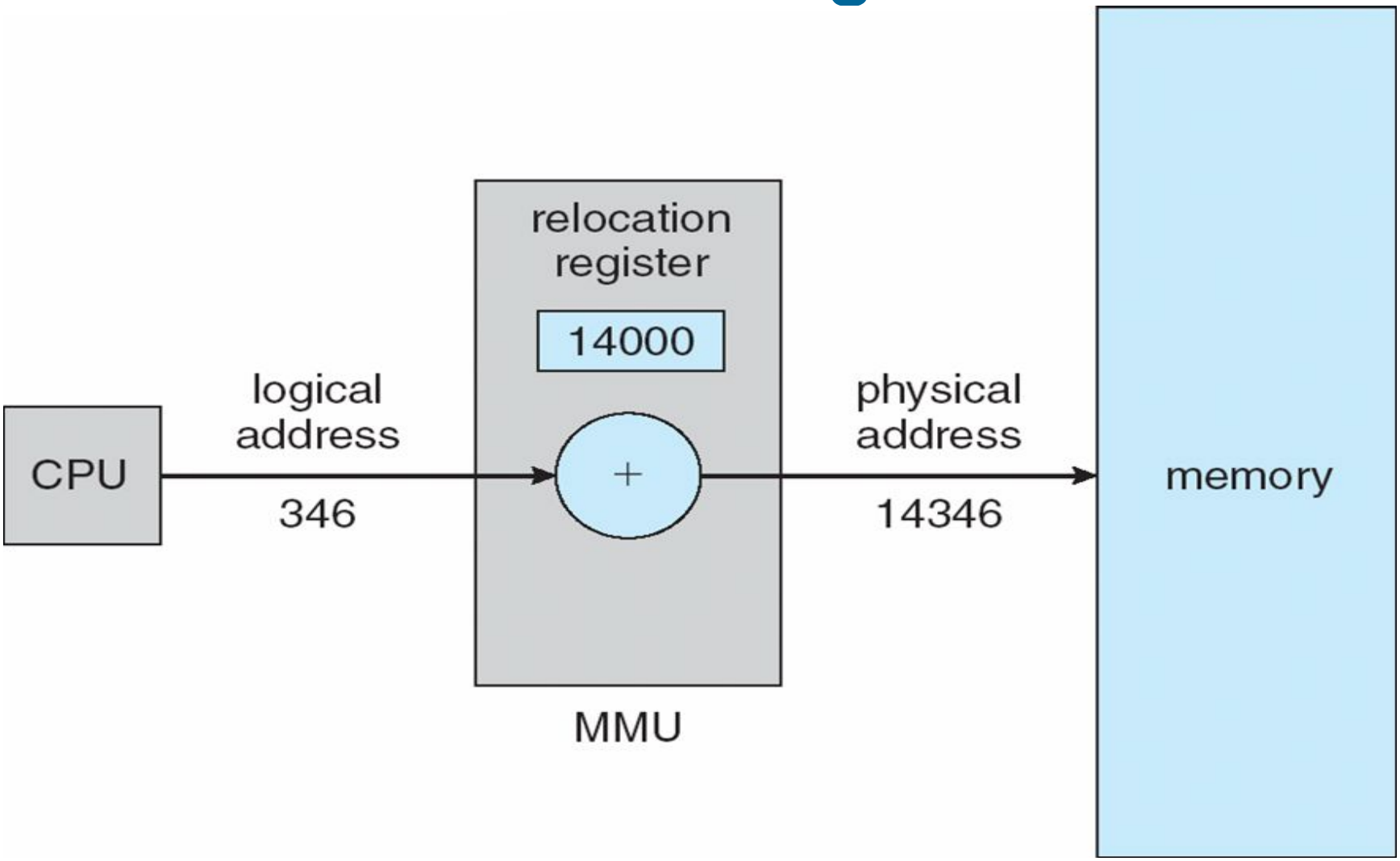
- **Base Register –** contains the starting physical address of the process.
- **Limit Register** -mentions the limit relative to the base address on the region occupied by the process.



Memory-Management Unit (MMU)

- The run time mapping between Virtual address and Physical Address is done by a hardware device known as MMU.
- In memory management, the Operating System will handle the processes and move the processes between disk and memory for execution .
- It keeps track of available and used memory.

Dynamic relocation using a relocation register



Dynamic Loading

- 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

Advantage of dynamic loading -

- unused routine is never loaded.

-Dynamic loading does not require special support from the OS

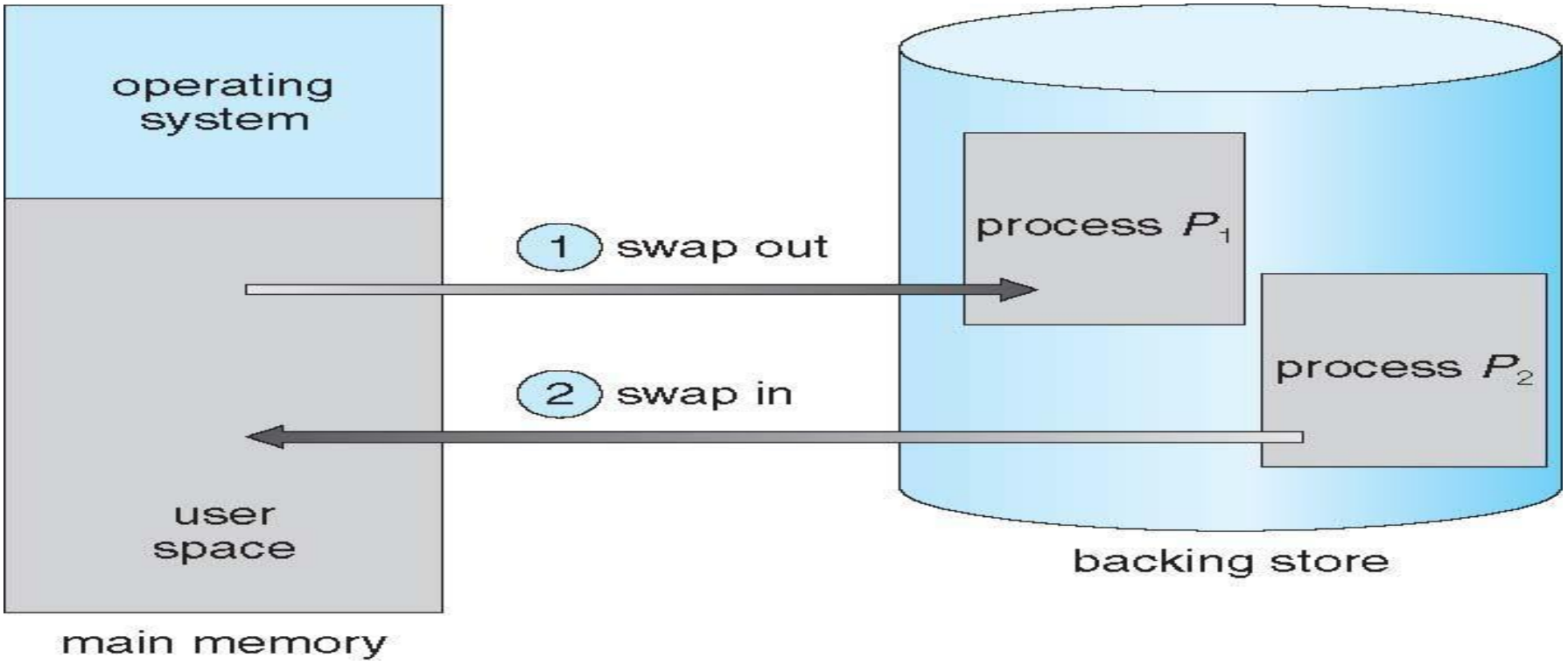
Dynamic Linking

- **Static linking** – When we click the .exe (executable) file of the program and it starts running, all the necessary contents of the binary file have been loaded into the process's virtual address space.
- system libraries and program code combined by the loader into the binary program image
- **Dynamic linking –**linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- **Dynamic linking is particularly useful for libraries**
- System also known as **shared libraries**

Swapping

- A process can be **swapped temporarily** out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; **lower-priority process is swapped out so higher-priority process can be loaded and executed**
- Major part of swap time is **transfer time;** total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



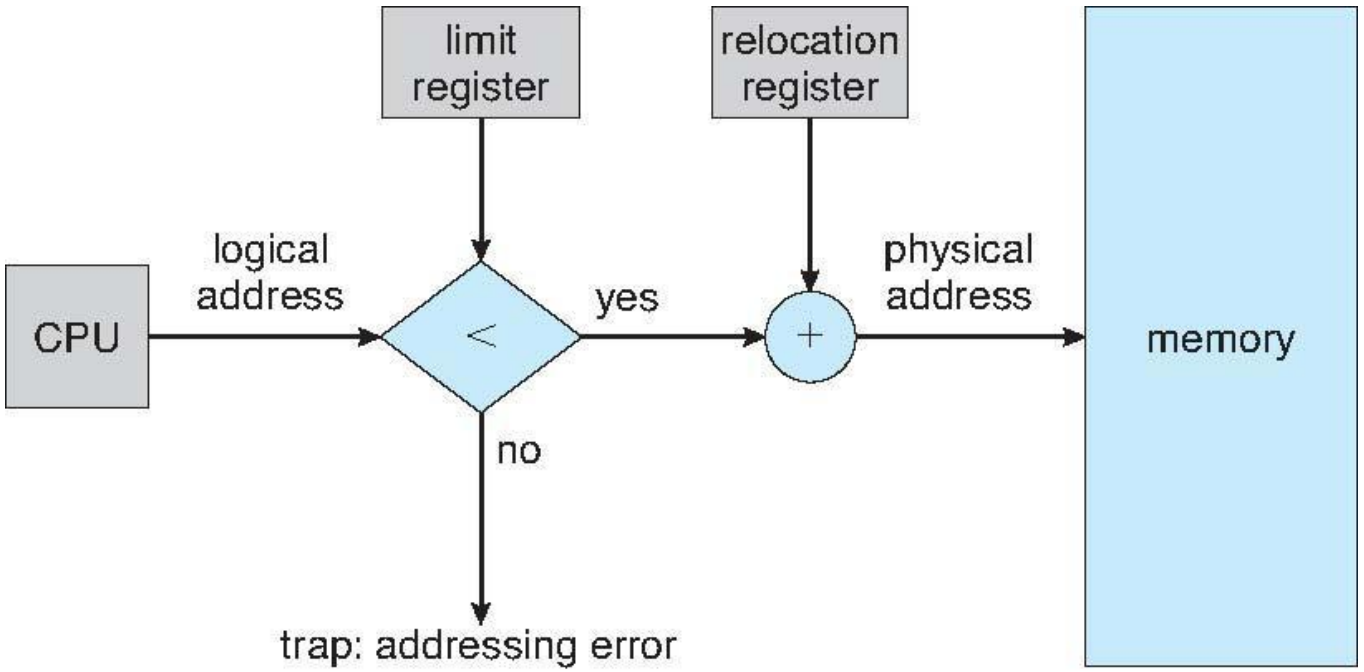
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Contiguous Allocation

- 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**

- 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*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



THANK YOU