

Outline

- ▶ Module 1 and 2 dealt with processes, scheduling and synchronization
- ▶ Next two modules will deal with memory and storage
- ▶ Processes require data to perform useful work
 - Stable storage is slow
 - Transient storage (memory) is faster
 - multiple levels with different speeds
- ▶ Goal is to get as much performance out of the system by moving data between the levels



Memory management problem

- ▶ Physical memory large contiguous address
 - Need to load multiple programs into the address
 - protect each from others
 - Require hardware support - managed by OS
 - Ideally the presence of multiple programs is transparent to the application
 - Use logical rather than physical addresses
 - Require hardware support - managed by OS
 - Swap contents to lower levels in memory hierarchy
 - OS manages allocation of memory resources

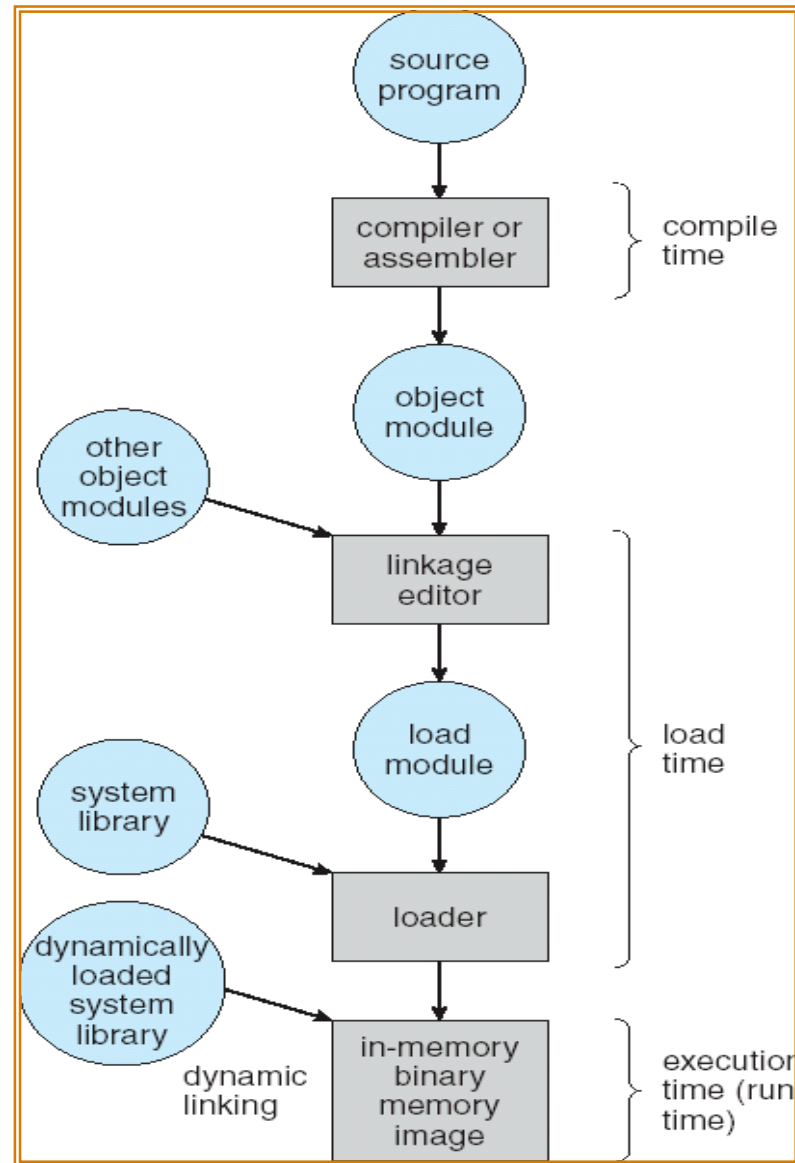


Chapter 8: Memory management

- ▶ Program must be brought into memory and placed within a process for it to be run
- ▶ Memory hierarchy: Processor -> Registers, Cache (L1, L2, L3 ..), Main memory, hard disk
 - The closer to processor, the faster and expensive
- ▶ **Input queue** – collection of processes on the disk that are waiting to be brought into memory to run the program
- ▶ User programs go through several steps before being run



Multistep Processing of a User Program



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
- ▶ **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*).



Loading programs

► Dynamic loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- 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

► 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 needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries



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

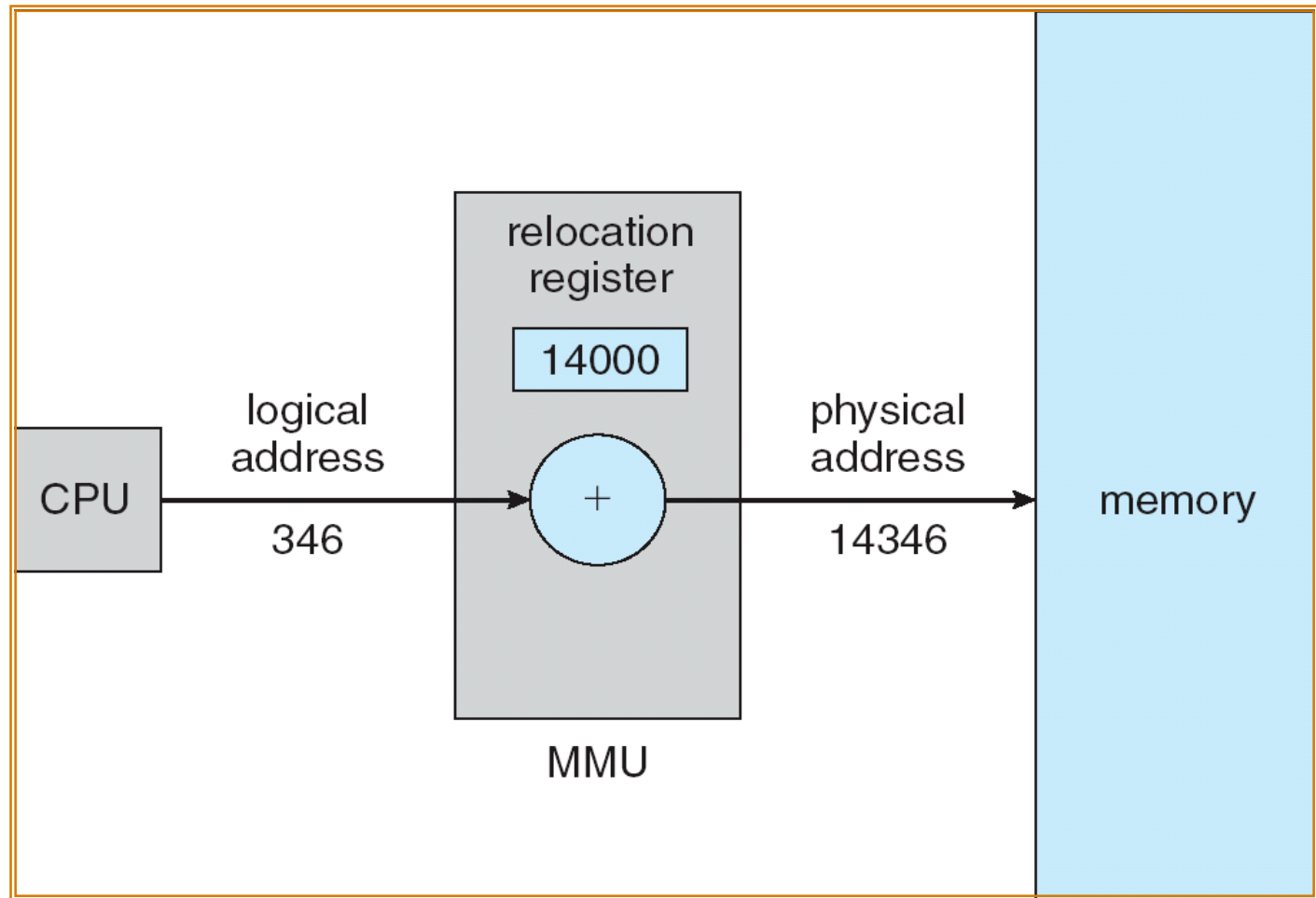


Memory-Management Unit (MMU)

- ▶ Hardware device that maps virtual to physical address
- ▶ In MMU scheme, 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



Dynamic relocation using a relocation register



MMU is a hardware component
OS manages the relocation register

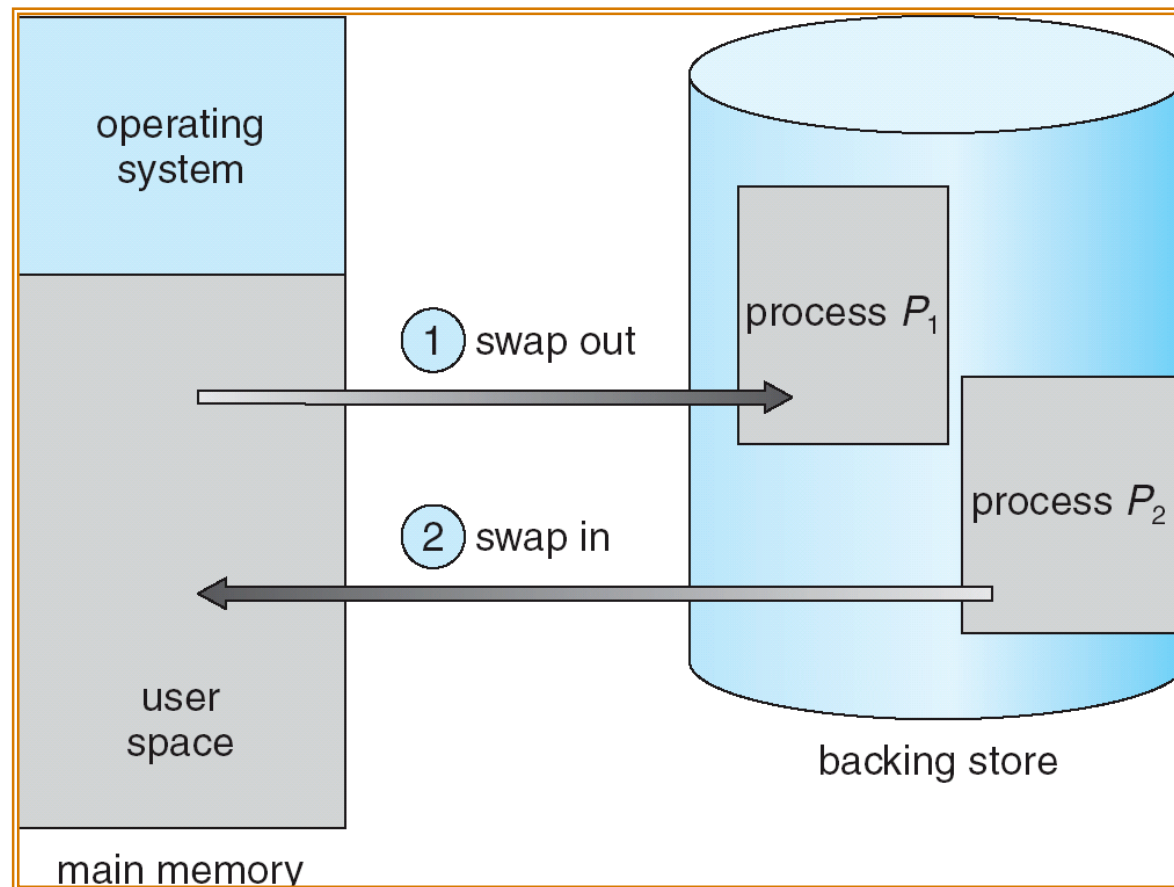


Swapping

- ▶ A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- ▶ 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
- ▶ Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)



Schematic View of Swapping

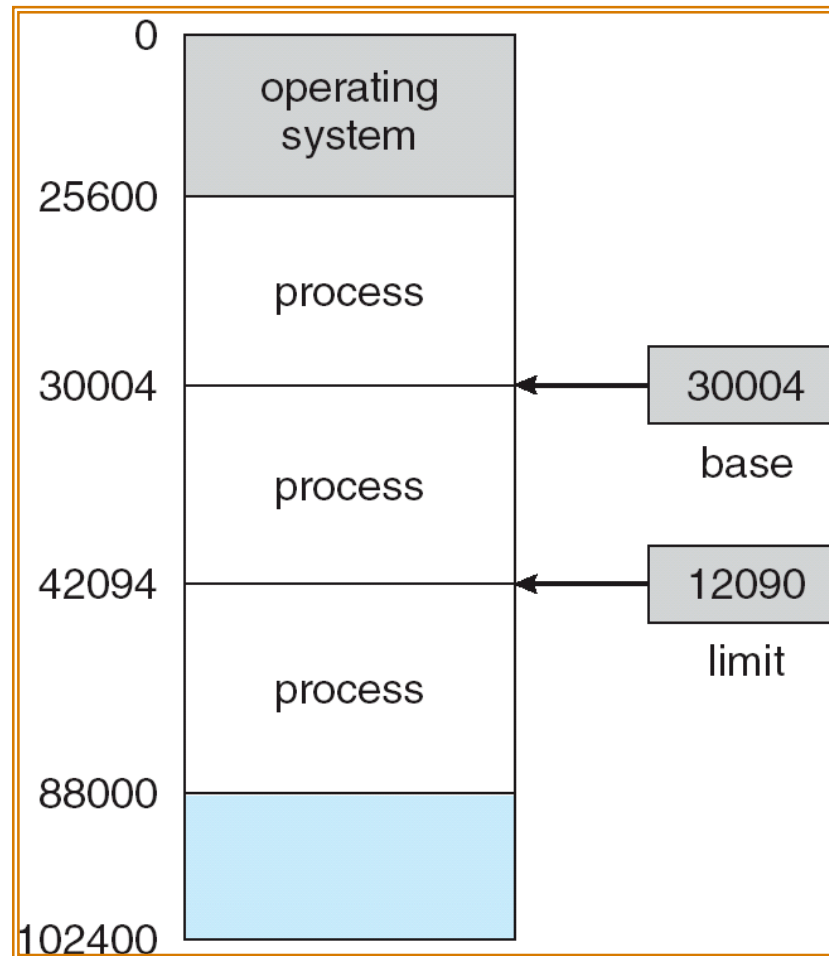


Contiguous Allocation

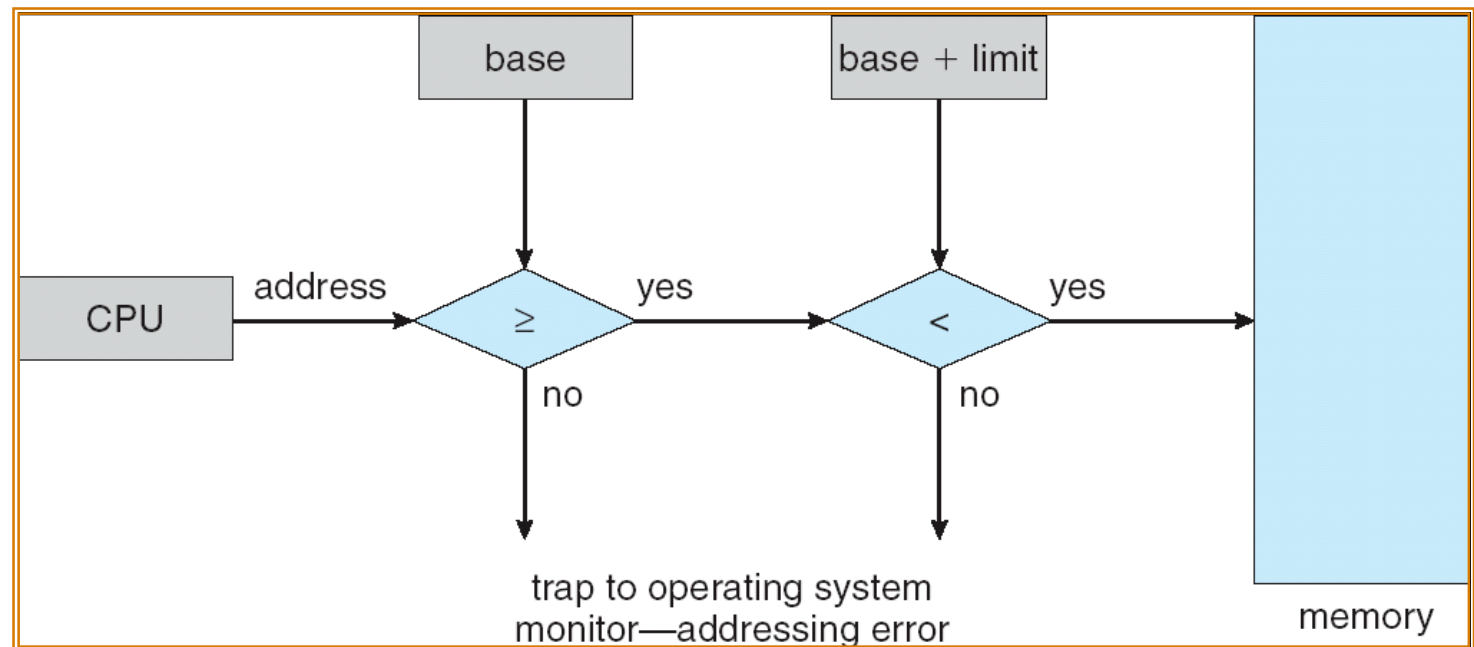
- ▶ Main memory usually into two partitions:
 - Resident operating system, usually held in low memory with interrupt vector
 - User processes then held in high memory
- ▶ Single-partition allocation
 - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data
 - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register



A base and a limit register define a logical address space



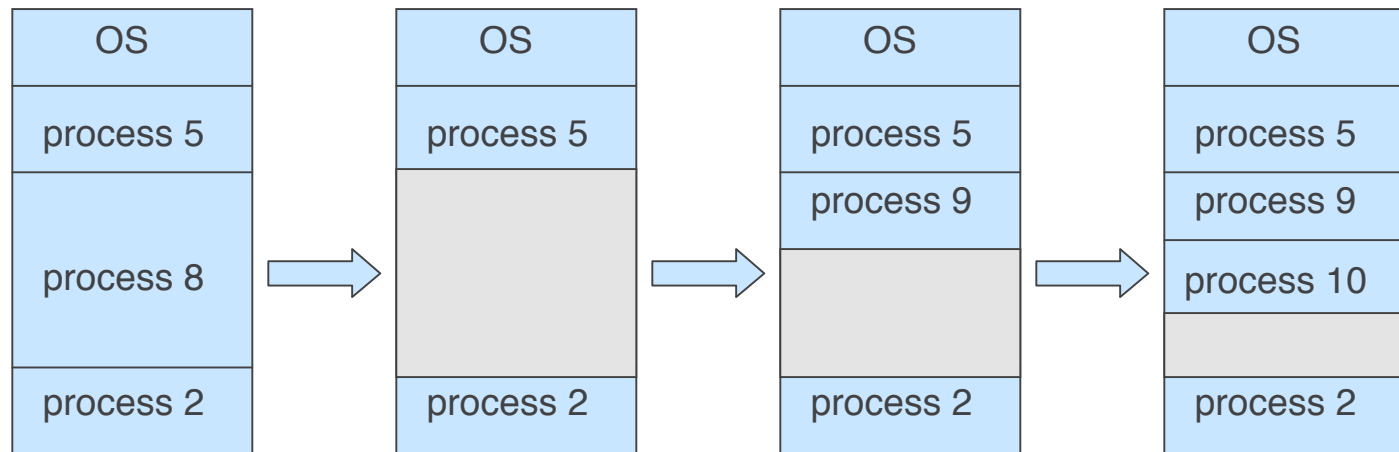
HW address protection with base and limit registers



Contiguous Allocation (Cont.)

► Multiple-partition allocation

- *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
- 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.

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
- ▶ **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- ▶ 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 - I/O may be performed by a DMA controller
 - Latch job in memory while it is involved in I/O
 - Do I/O only into OS buffers

