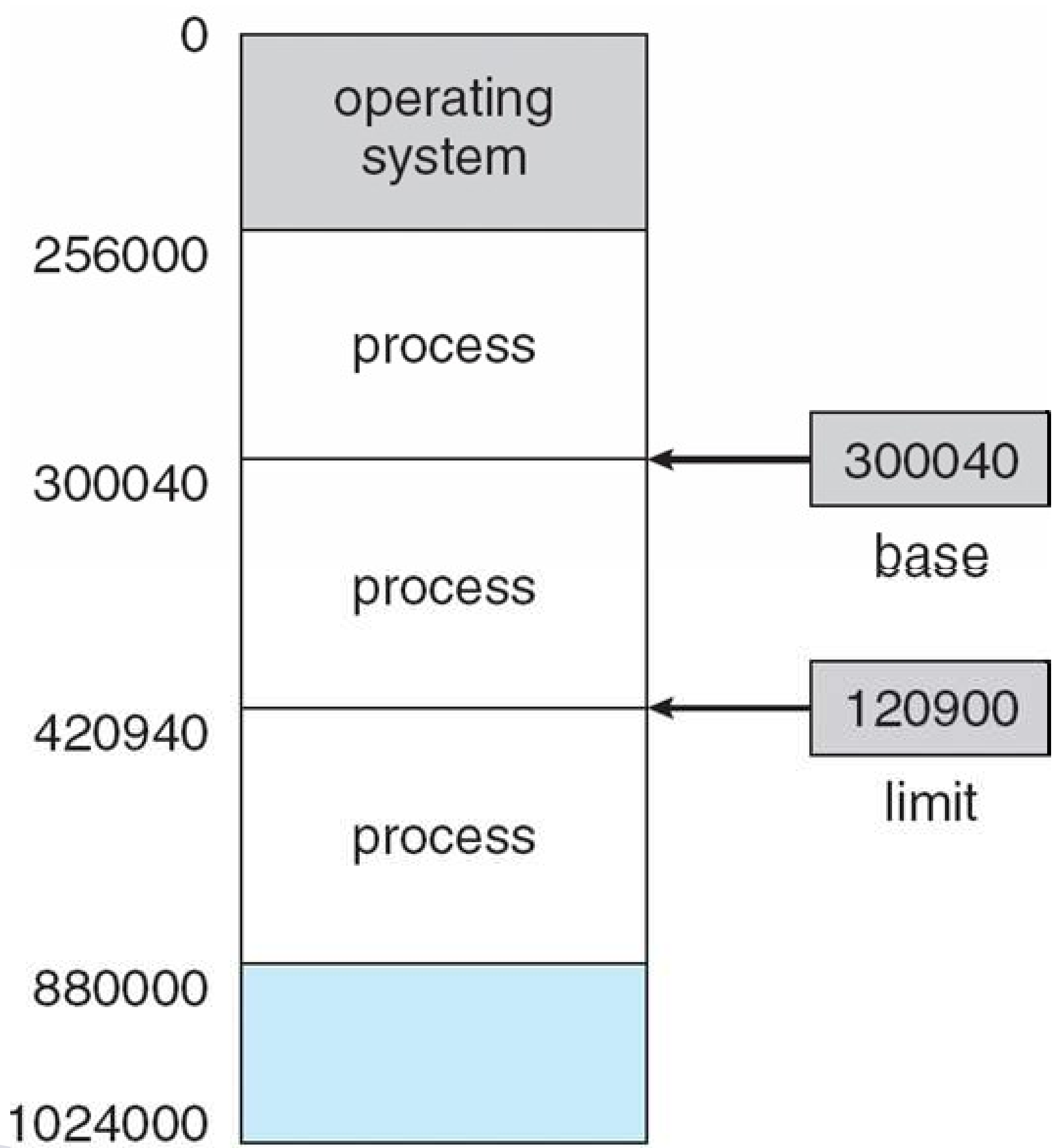
Memory Management

Ms.Hitha Paulson
Assistant Professor, Dept of Computer Science
Little Flower College, Guruvayoor

Background

- Memory --- large array of words or bytes with its own address
- CPU fetches instructions from memory according to the value of program counter
- Instructions cause additional loading from and storing to specific memory address
- Fetches -- Decode -- Execute


0
operating system
256000
process
300040
process
420940
process
880000
1024000
300040
base
120900
limit

Address Binding

- The collection of processes on the disk that is waiting to be brought in to memory for execution forms the input queue.
- Select one from input queue and load to memory.
- Inconvenient to have first user process physical address always at 0000

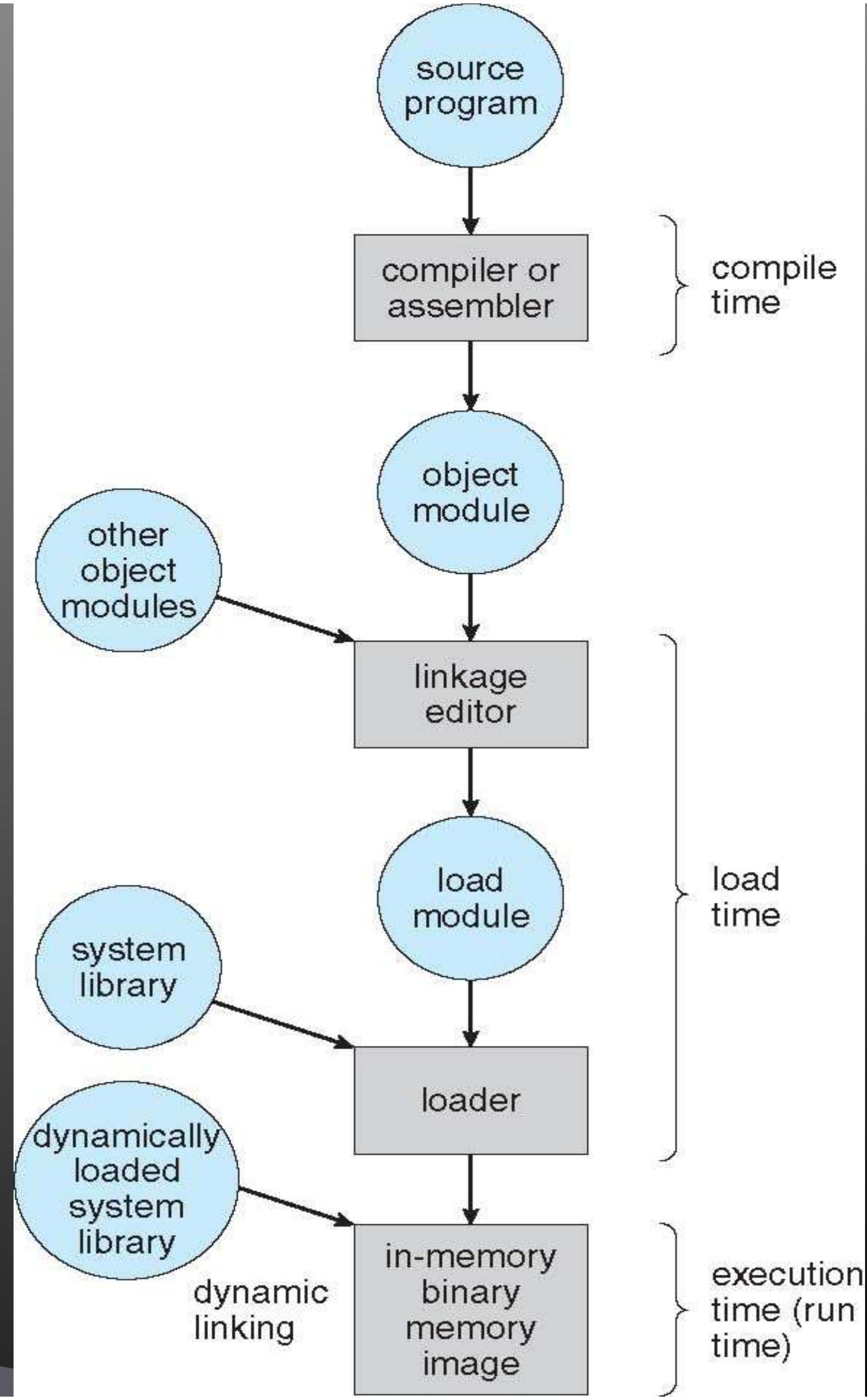
Address Binding

- 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

Binding of Instructions and Data

- 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 (early binding)
  - Load time: Must generate relocatable code if memory location is not known at compile time (delayed binding)
  - Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another (late binding)
    - Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



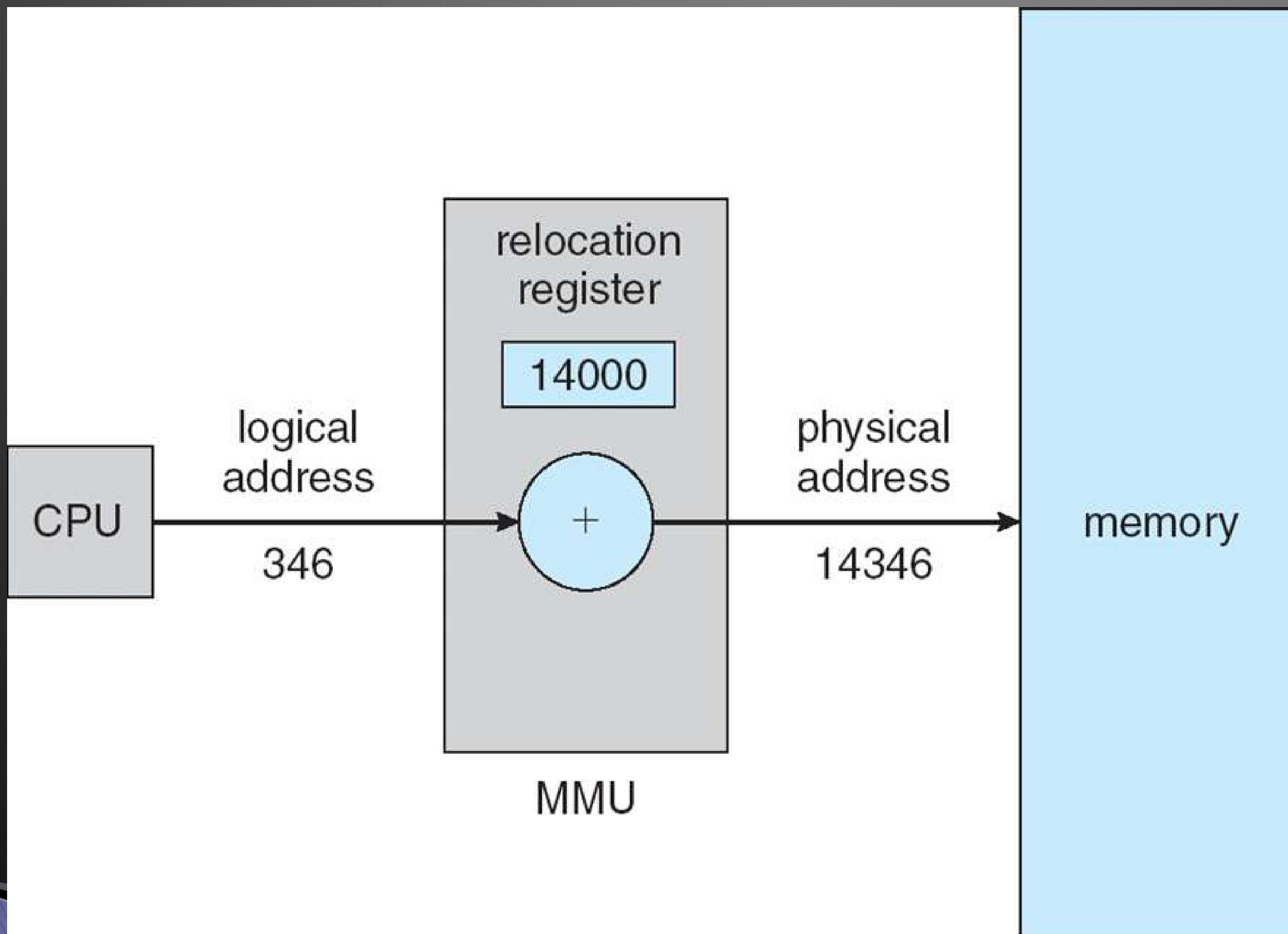
Logical vs. Physical Address Space

- 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

Memory-Management Unit (MMU)

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

Dynamic relocation using a relocation register



Dynamic Loading

- Still we considered that entire program and data of a process must be in physical memory for the process of execute.
- The size of the process is limited to size of physical memory.
- For better utilization – Dynamic loading, ie a routine is not loaded until it is called.
- Unused routines are never loaded.
- It is the responsibility of the users to design their programs to take advantage of this method

Dynamic Linking

- Static linking – 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
- Consider applicability to patching system libraries
  - Versioning may be needed

Dynamic Linking

- Static linking – 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
- Consider applicability to patching system libraries
- Dynamic linking needs help from Operating system.

Overlays

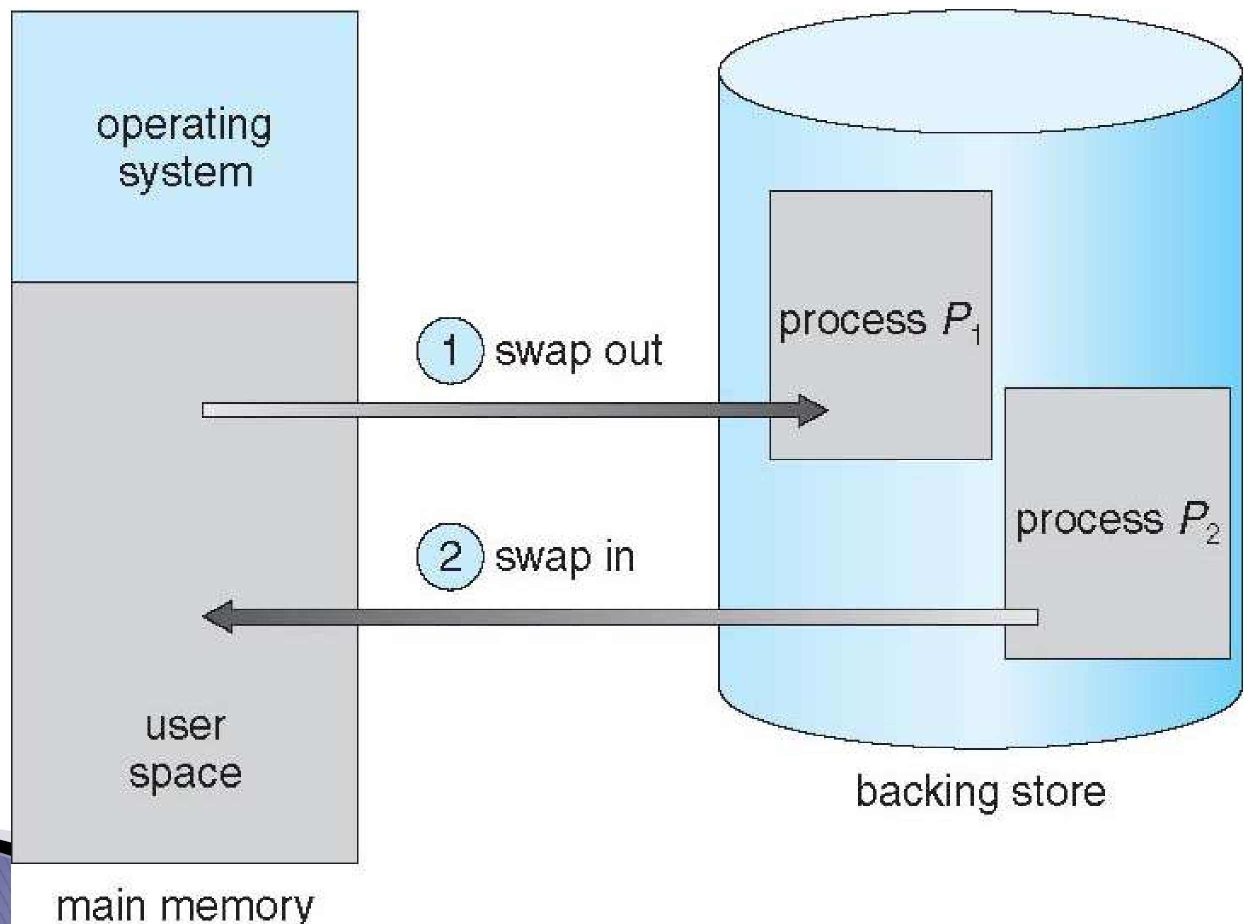
- To keep in memory only those instructions and data that are needed at any given time.
- Preparation of Mutually exclusive instruction and data set to be loaded.
- Need not require any support from OS

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
- System maintains a ready queue of ready-to-run processes which have memory images on disk

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space



Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- 100MB process swapping to hard disk with transfer rate of 50MB/sec
  - Swap out time of 2000 ms
  - Plus swap in of same sized process
  - Total context switch swapping component time of 4000ms (4 seconds)

Contiguous Allocation

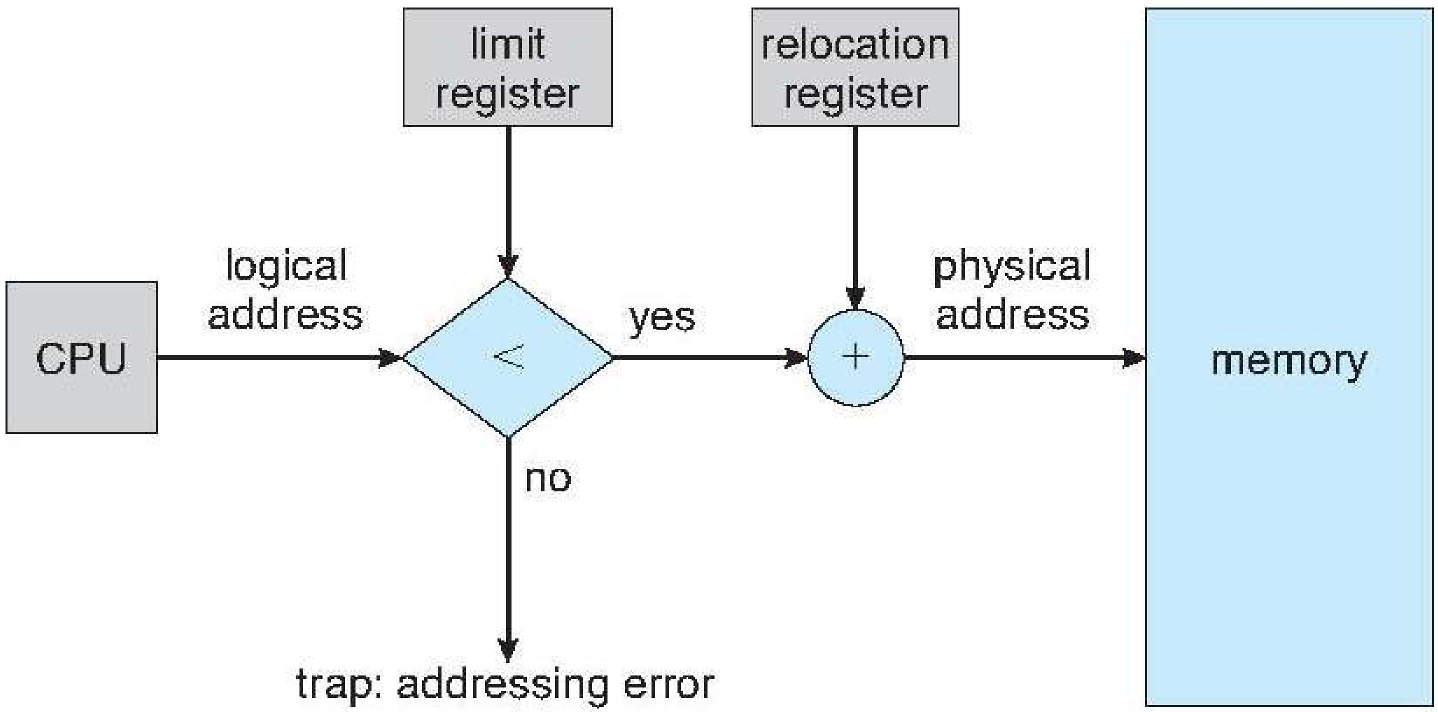
- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

1. Single Contiguous
2. Multiple Fixed Partitioned
3. Multiple Variable partitioned

Contiguous Allocation (Cont.)

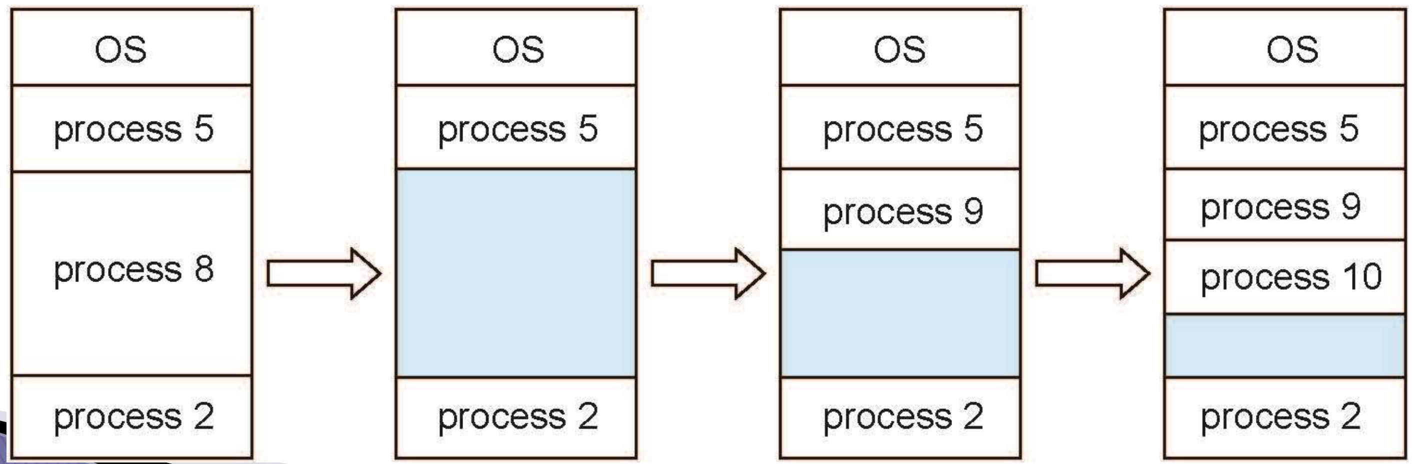
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

- 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
  - 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

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

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
- First fit analysis reveals that given *N* blocks allocated, 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
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers