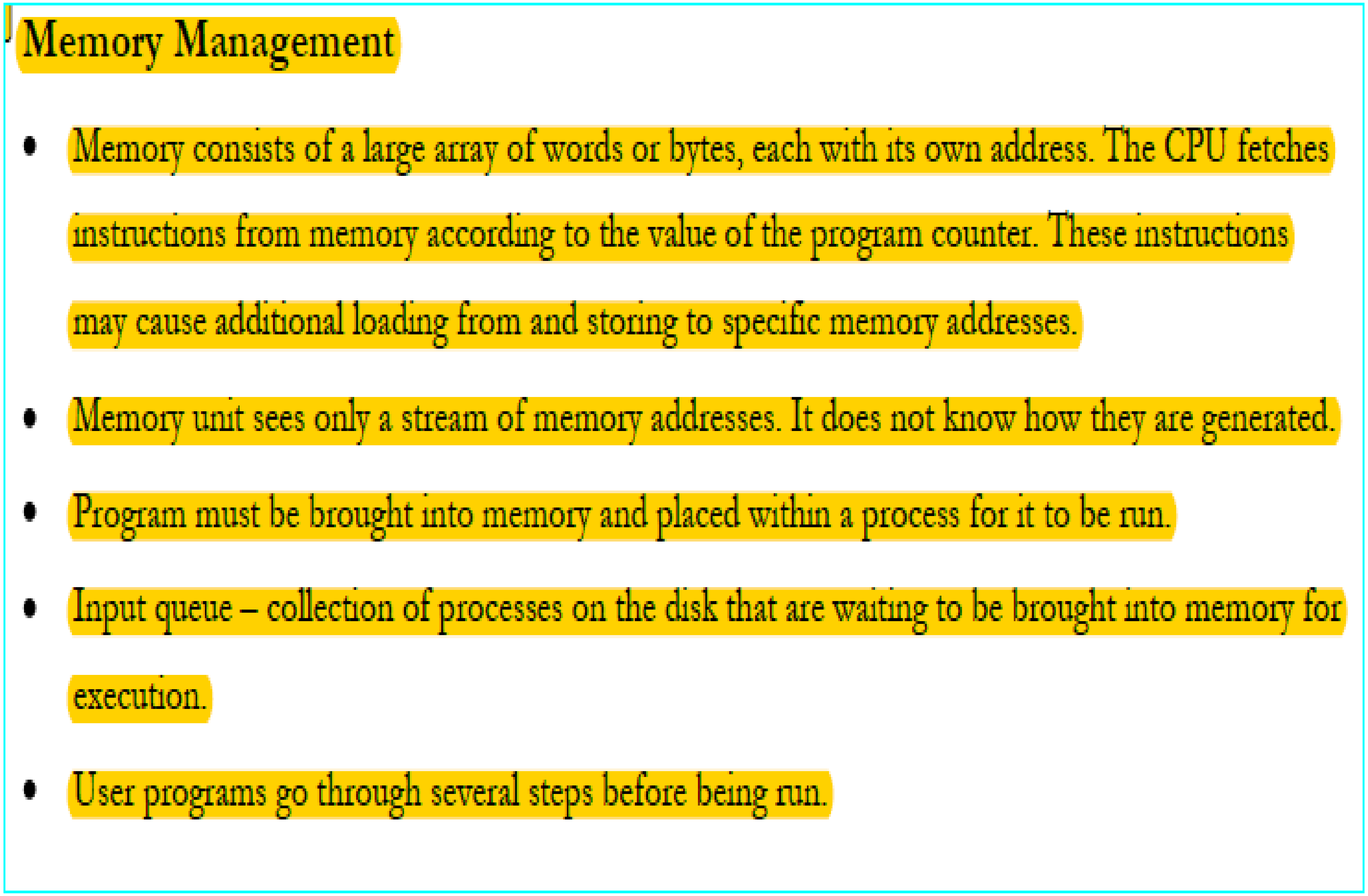
Operating system

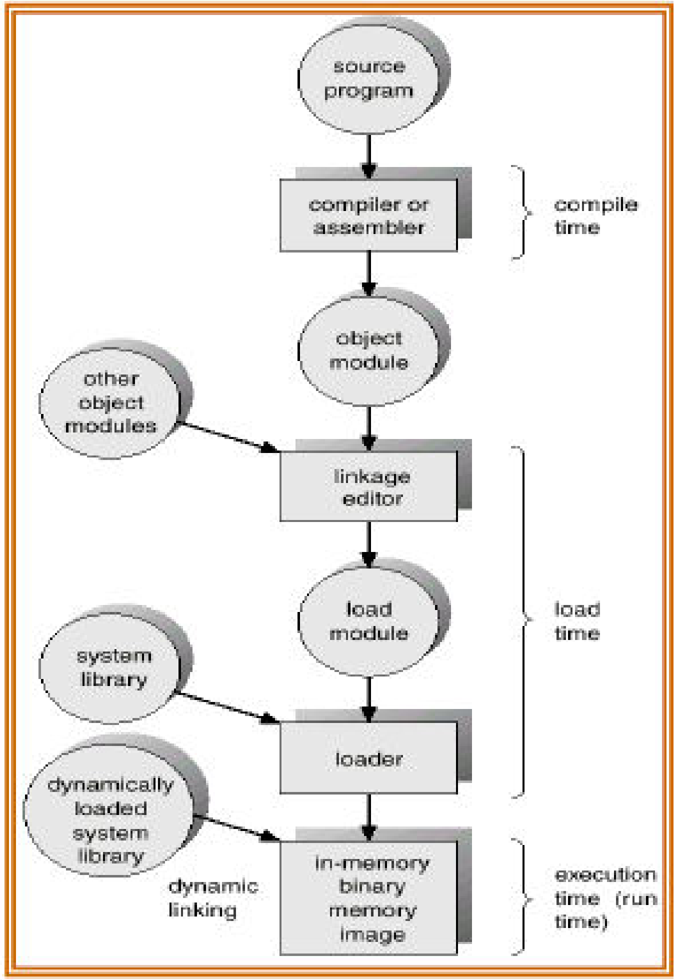
Mahesh Malkani
A.P in CSE Deptt.

Memory Management



Memory Management

- Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory unit sees only a stream of memory addresses. It does not know how they are generated.
- Program must be brought into memory and placed within a process for it to be run.
- Input queue – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being run.



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.

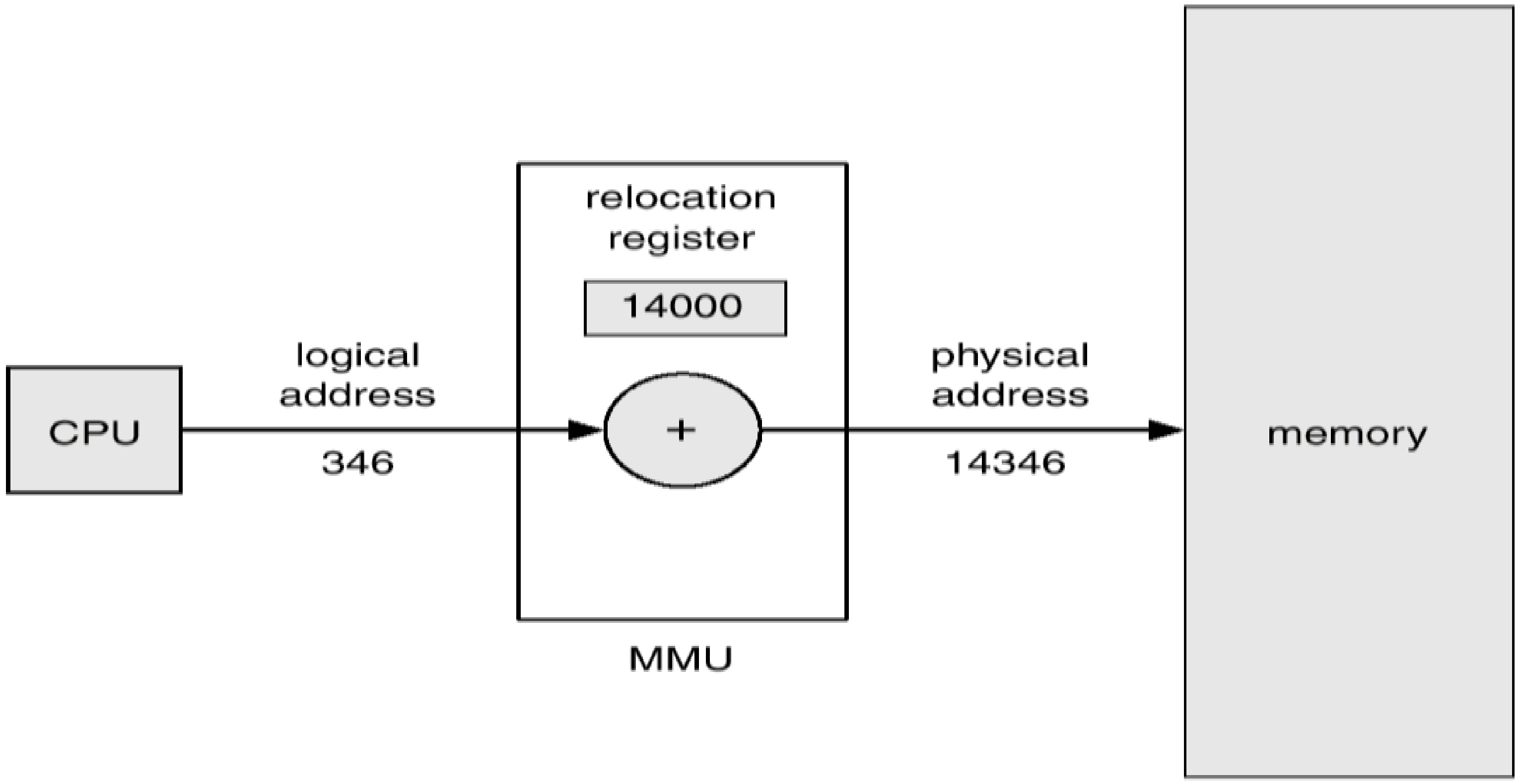
  Example: .COM-format programs in MS-DOS.

- **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., relocation registers).**

- **Logical Versus 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 – address generated by the CPU; also referred to as virtual address.
- Physical address – address seen by the memory unit.
- The set of all logical addresses generated by a program is a logical address space; the set of all physical addresses corresponding to these logical addresses are a physical address space.

- 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.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (MMU).



- This method requires hardware support slightly different from the hardware configuration.
- The base register is now called a 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 never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it and compare it to other addresses.
- The user program deals with logical addresses. The memory mapping hardware converts logical addresses into physical addresses.
- The final location of a referenced memory address is not determined until the reference is made.

Dynamic Loading

- **Routine is not loaded until it is called.**
- **All routines are kept on disk in a relocatable load format.**
- **The main program is loaded into memory and is executed.**
- **When a routine needs to call another routine, the calling routine first checks to see whether the other the desired routine into memory and to update the program's address tables to reflect this change.**
- **Then control is passed to the newly loaded routine.**
- **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 is postponed until execution time.
- Small piece of code, stub, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory.
- Stub replaces itself with the address of the routine, and executes the routine.
- Thus the next time that code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.
- Operating system is needed to check if routine is in processes‘ memory address.
- Dynamic linking is particularly useful for libraries.

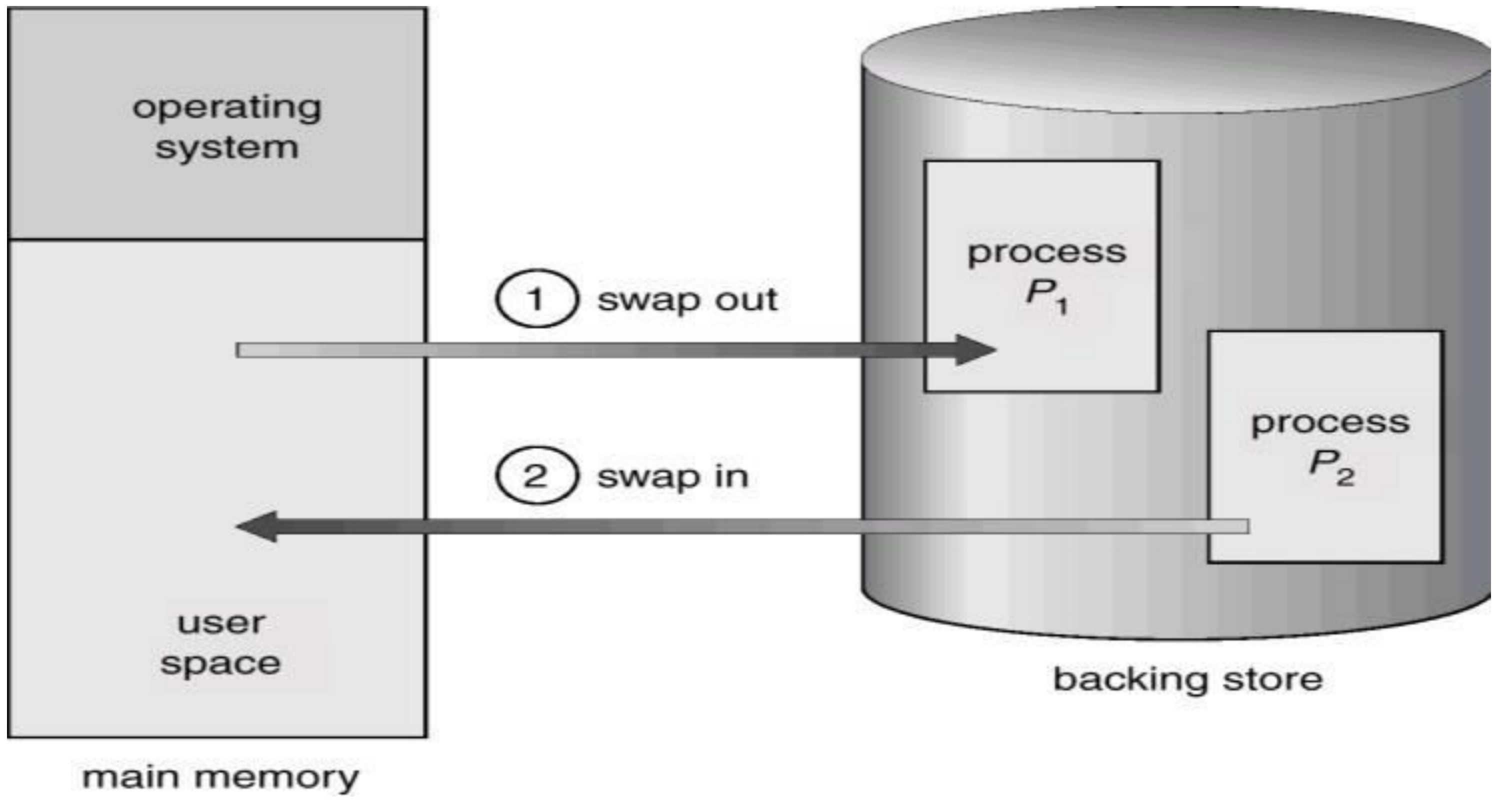
Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm.
- When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed.
- In the mean time, the CPU scheduler will allocate a time slice to some other process in memory.
- When each process finished its quantum, it will be swapped with another process.
- Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

- Roll out, roll in – swapping variant used for priority-based scheduling algorithms.
- If a higher priority process arrives and wants service, the memory manager can swap out the lower priority process so that it can load and execute lower priority process can be swapped back in and continued.
- This variant is some times called roll out, roll in. Normally a process that is swapped out will be swapped back into the same memory space that it occupied previously.
- This restriction is dictated by the process cannot be moved to different locations. If execution time binding is being used, then a process can be swapped into a different memory space, because the physical addresses are computed during execution time.

- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- The system maintains a ready queue consisting of all processes whose memory images are scheduler decides to execute a process it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory. If not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.
- It then reloads registers as normal and transfers control to the selected process.

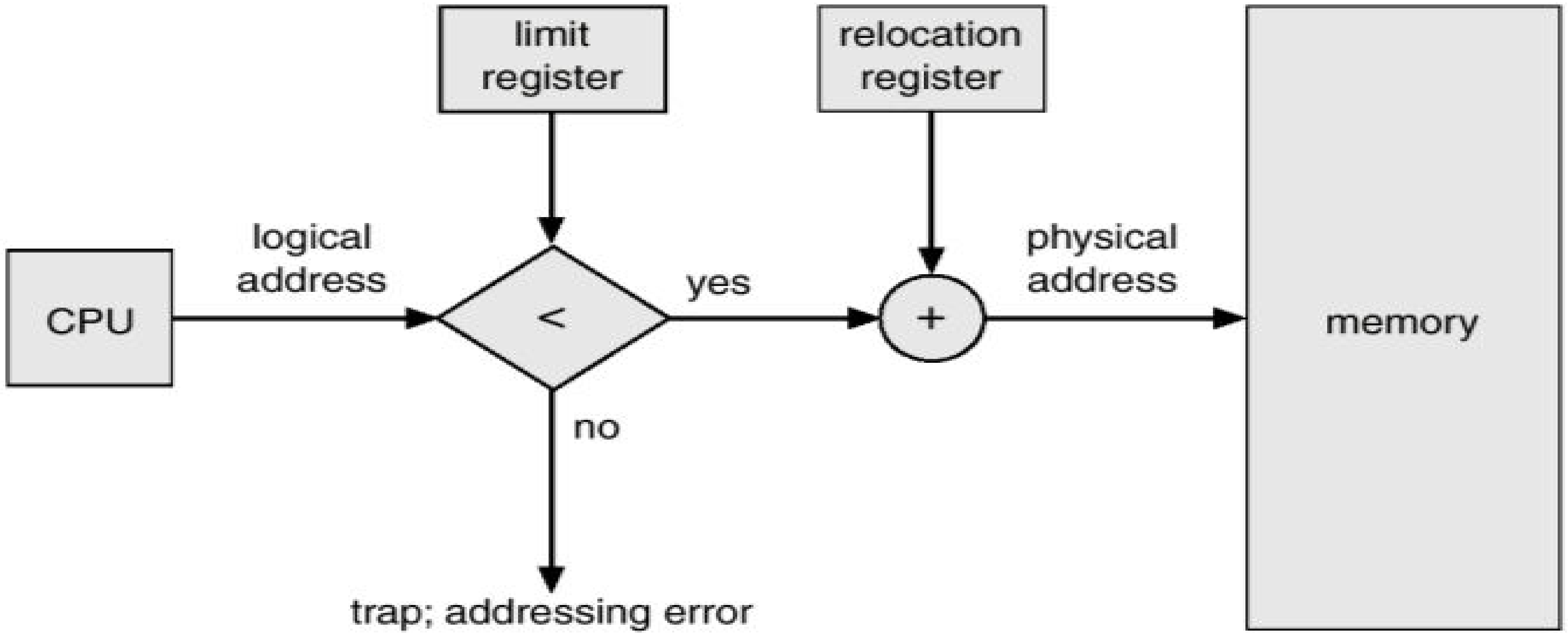
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows).



Contiguous Memory Allocation

- Main memory is usually divided into two partitions:
- o Resident operating system, usually held in low memory with interrupt vector.
- User processes, held in high memory.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Single-partition allocation
- Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.

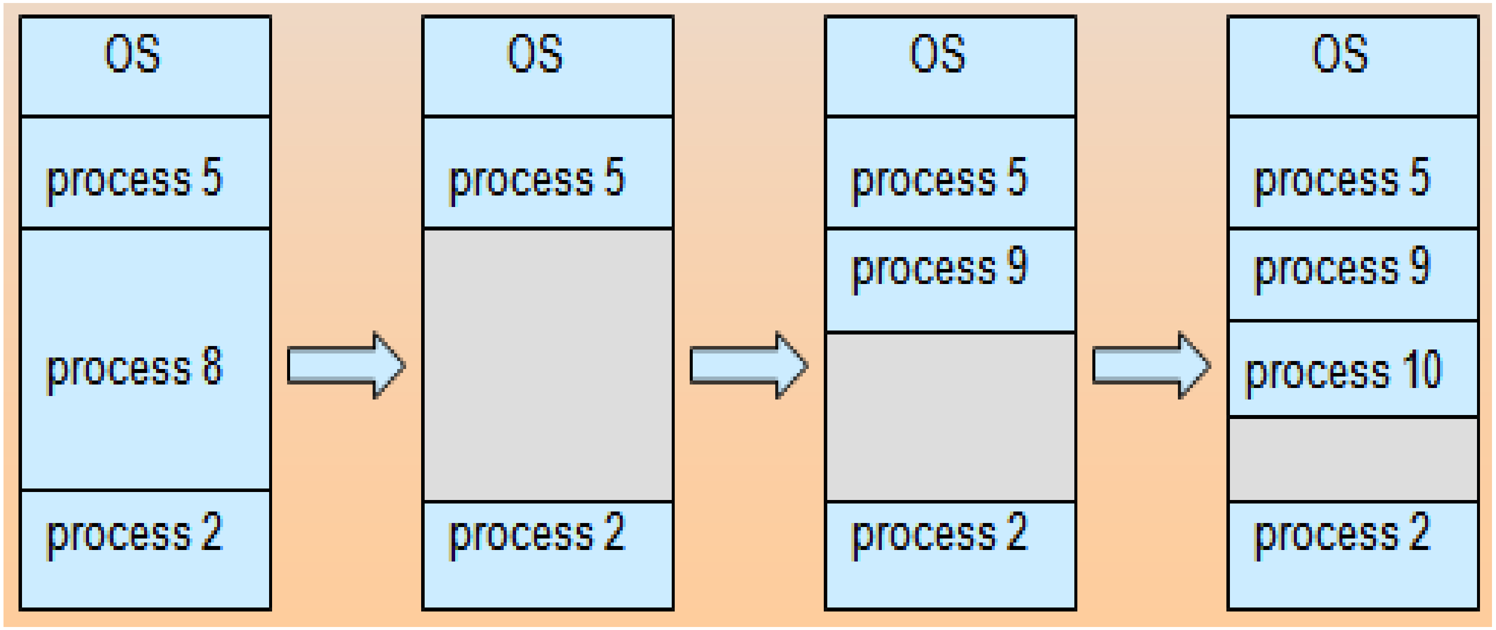
- Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register


limit register
relocation register
logical address
CPU
<
yes
+
physical address
memory
no
trap; addressing error

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)
- A set of holes of various sizes, is scattered throughout memory at any given time. When a process arrives and needs memory, the system searches this set for a hole that is large enough for this process. If the hole is too large, it is split into two: one part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hold is adjacent to other holes, these adjacent holes are merged to form one larger hole.

- This procedure is a particular instance of the general dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes.
- There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate.
- The first-fit, best-fit and worst-fit strategies are the most common ones used to select a free hole from the set of available 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.**
- **Worst-fit: Allocate the largest hole; must also search entire list.**