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.
- A typical instruction-execution cycle,

  --for example, first fetches an instruction from memory.

  --The instruction is then decoded and may cause operands to be fetched from memory.

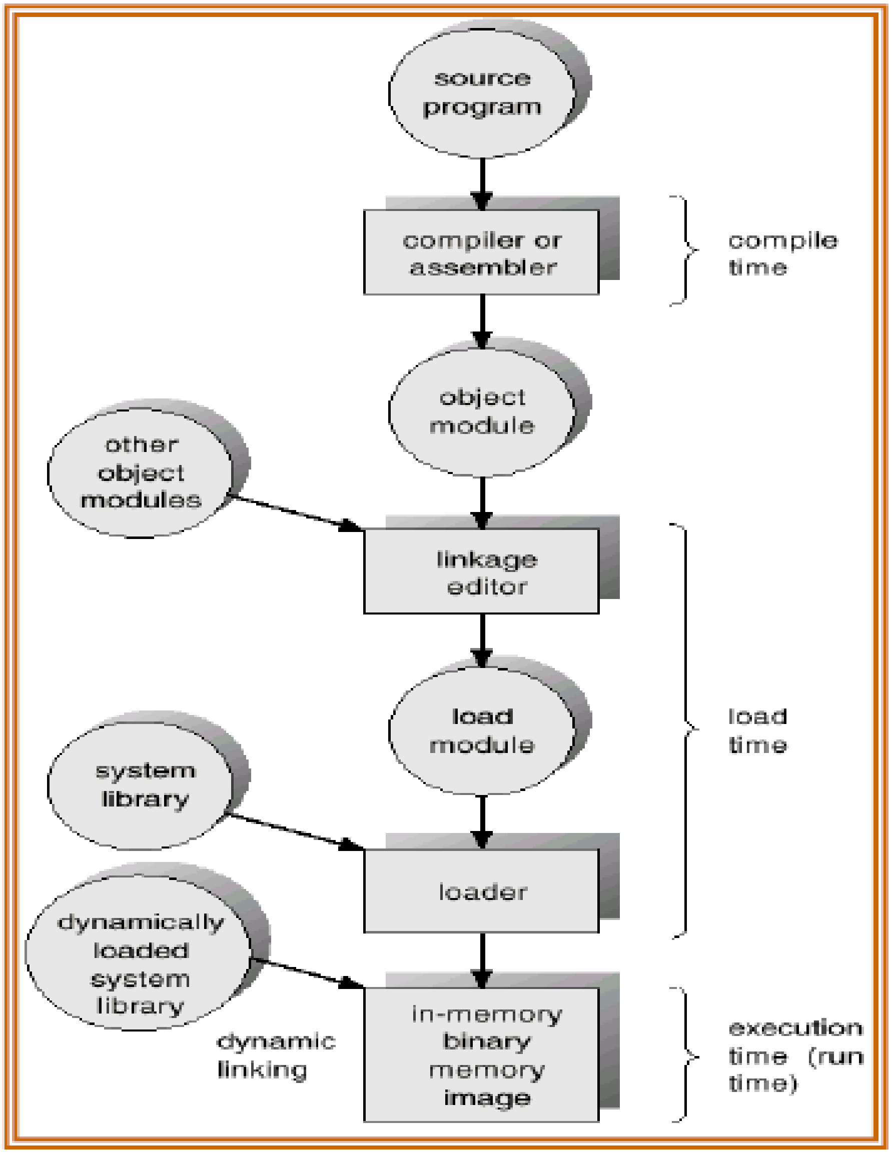
  --After the instruction has been executed on the operands, results may be stored back in memory.

- **Memory Management**
- 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.

- **Memory Management**
- The normal procedure is to select one of the processes in the input queue and to load that process into memory.
- As the process is executed, it accesses instructions and data from memory.
- Eventually, the process terminates, and its memory space is declared available.

- a user program goes through several steps before being executed.
- Addresses may be represented in different ways during these steps.
- Addresses in the source program are generally symbolic (such as the variable count).
- A compiler typically binds these symbolic addresses to relocatable addresses (such as “14 bytes from the beginning of this module”).
- The linkage editor or loader in turn binds the relocatable addresses to absolute addresses (such as 74014).
- Each binding is a mapping from one address space to another.

Multistep processing of a user program.



- Classically, the binding of instructions and data to memory addresses can be done at any step along the way:
- **Compile time**. If you know at compile time where the process will reside in memory, then absolute code can be generated.
- For example, if you know that a user process will reside starting at location *R, then the generated* compiler code will start at that location and extend up from there.
- If, at some later time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS .COM-format programs are bound at compile time.

- Load time. If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code.
- In this case, final binding is delayed until load time.
- If the starting address changes, we need only reload the user code to incorporate this changed value.

- **Execution time:** If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time.
- Special hardware must be available for this scheme to work

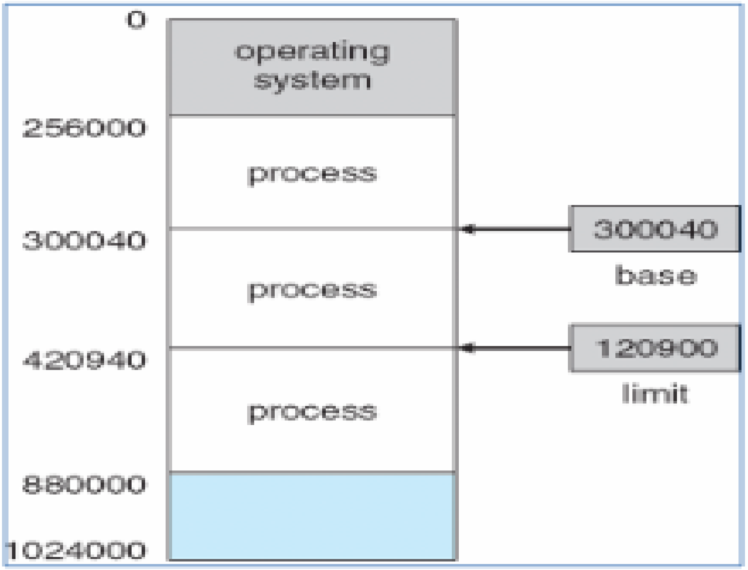
Logical Versus Physical Address Space

- An address generated by the CPU is commonly referred to as a logical address,
- whereas an address seen by the memory unit—that is, the one loaded into the memory-address register of the memory—is commonly referred to as a physical address.
- The compile-time and load-time address-binding methods generate identical logical and physical addresses.
- the execution-time address binding scheme results in differing logical and physical addresses.

- 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.

What is the Base register and what is the Limit register?

- **Base register:** Specifies the smallest legal physical memory address.
- **Limit register:** Specifies the size of the range.
- A pair of base and limit registers specifies the logical address space.
- The base and limit registers can be loaded only by the **operating system.**
- Ex: If the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).



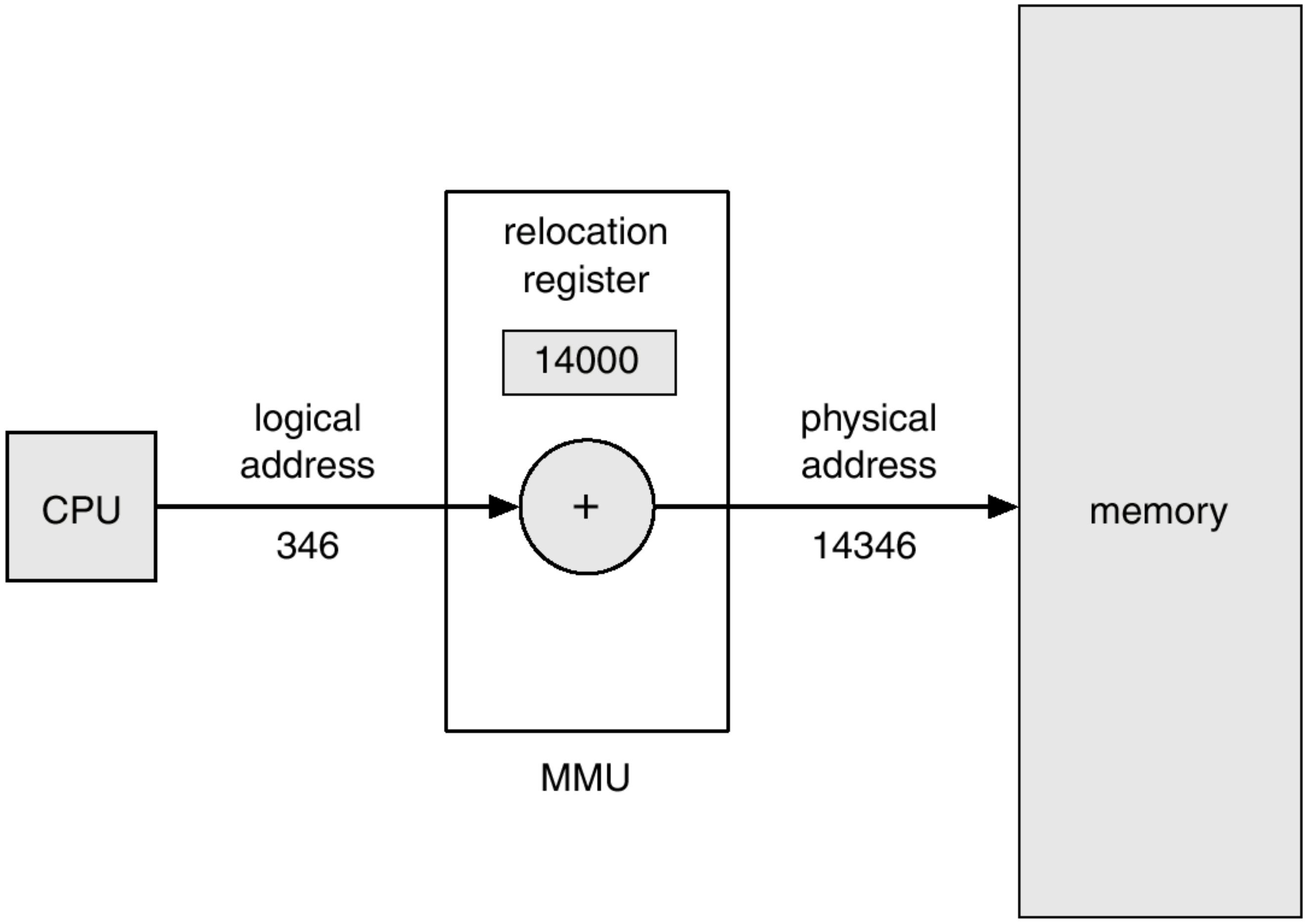
Logical Versus Physical Address Space

- we usually refer to the logical address as a virtual address.
- *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 is a physical address space.
- Thus, in the execution-time address-binding scheme, the logical and physical address spaces differ.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU).**

Logical Versus Physical Address Space

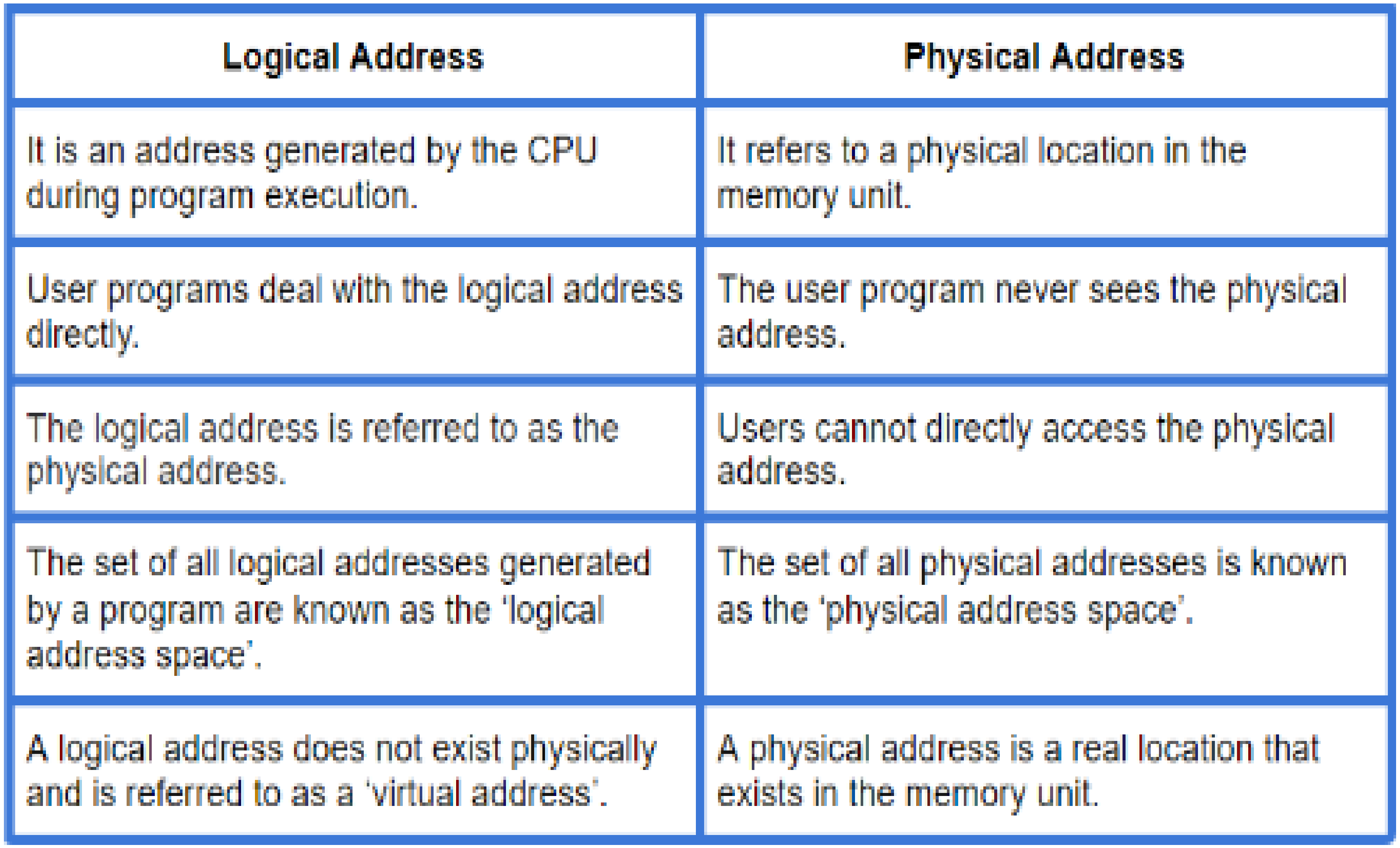
- The base register is now called a **relocation register.**
- The value in the relocation register is added to every address generated by user process at the time the address is sent to memory .
- For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location14000;
- An access to location346 is mapped to location 14346.

Dynamic relocation using a relocation register



- 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.

Comparison



Logical Address	Physical Address
It is an address generated by the CPU during program execution.	It refers to a physical location in the memory unit.
User programs deal with the logical address directly.	The user program never sees the physical address.
The logical address is referred to as the physical address.	Users cannot directly access the physical address.
The set of all logical addresses generated by a program are known as the 'logical address space'.	The set of all physical addresses is known as the 'physical address space'.
A logical address does not exist physically and is referred to as a 'virtual address'.	A physical address is a real location that exists in the memory unit.

Swapping

- Swapping is a memory management scheme
- any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes.
- It is used to improve main memory utilization.
- In secondary memory, the place where the swapped-out process is stored is called swap space.
- The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it.
- The thing to remember is that swapping is used only when data is not present in RAM.

Swapping

- A process must be in memory to be executed.
- A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution .
- Swap-out is a method of removing a process from RAM and adding it to the hard disk.
- Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

Swapping

- Standard swapping involves moving processes between main memory and a backing store.
- The backing store is commonly a fast disk.
- 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 on the backing store or in memory and are ready to run.

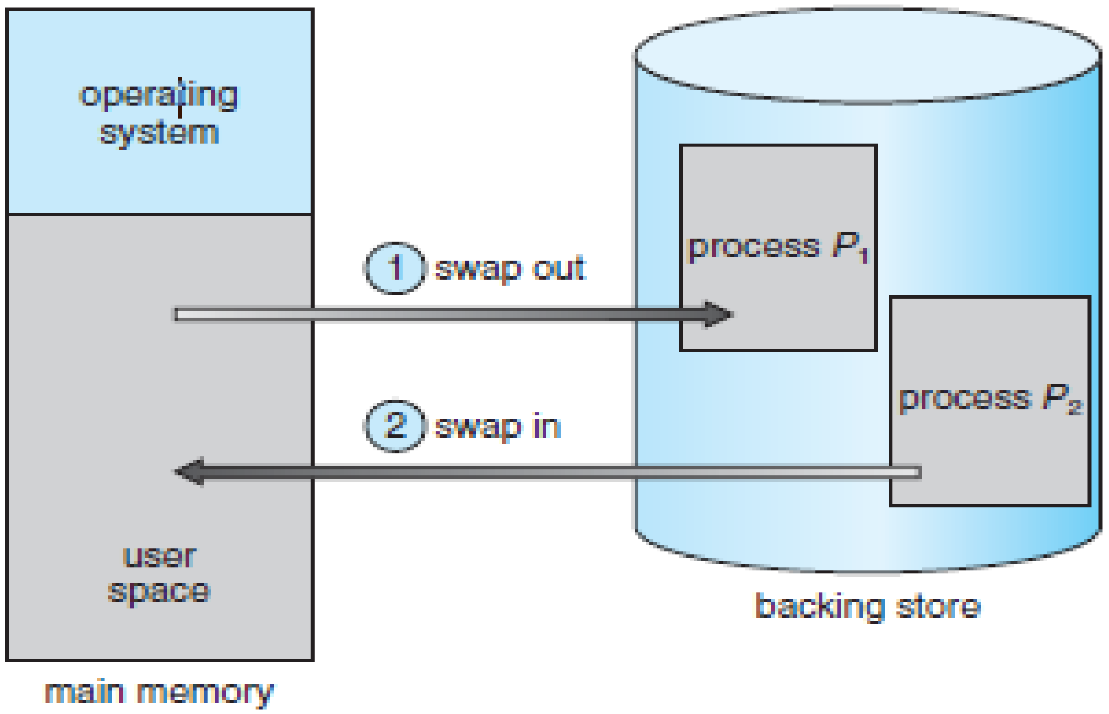


Figure 8.5 Swapping of two processes using a disk as a backing store.

- Whenever the CPU scheduler decides o execute a process, it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory.
- If it is not, and if 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 and transfers control to
- the selected process.
- The context-switch time in such a swapping system is fairly high.

- 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.

- Advantages of Swapping
- It helps the CPU to manage multiple processes within a single main memory.
- It helps to create and use virtual memory.
- Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
- It improves the main memory utilization.

Contiguous Memory Allocation

- The main memory must accommodate both the operating system and the various user processes.
- We therefore need to allocate main memory in the most efficient way possible.
- ***In contiguous memory allocation method,*** When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.
- In contiguous memory allocation, each process is contained in a single section of memory that is contiguous to the section containing the next process.
- The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.
- We can place the operating system in either low memory or high memory.
- The major factor affecting this decision is the location of the interrupt vector.
- Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well.

Memory protection

- Memory protection--Protecting the OS from user processes and protecting user processes from one another
- If we have a system with a relocation register , together with a limit register , we accomplish our goal.
- The relocation register contains the value of the smallest physical address;
- the limit register contains the range of logical addresses (for example, relocation = 100040 and limit = 74600).
- Each logical address must fall within the range specified by the limit register.
- The MMU maps the logical address dynamically by adding the value in the relocation register.
- This mapped address is sent to memory

Memory protection

- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- Because every address generated by a CPU is checked against these registers, we can protect both the operating system and the other users ’programs and data from being modified by this running process.
- The relocation-register scheme provides an effective way to allow the operating system’s size to change dynamically.

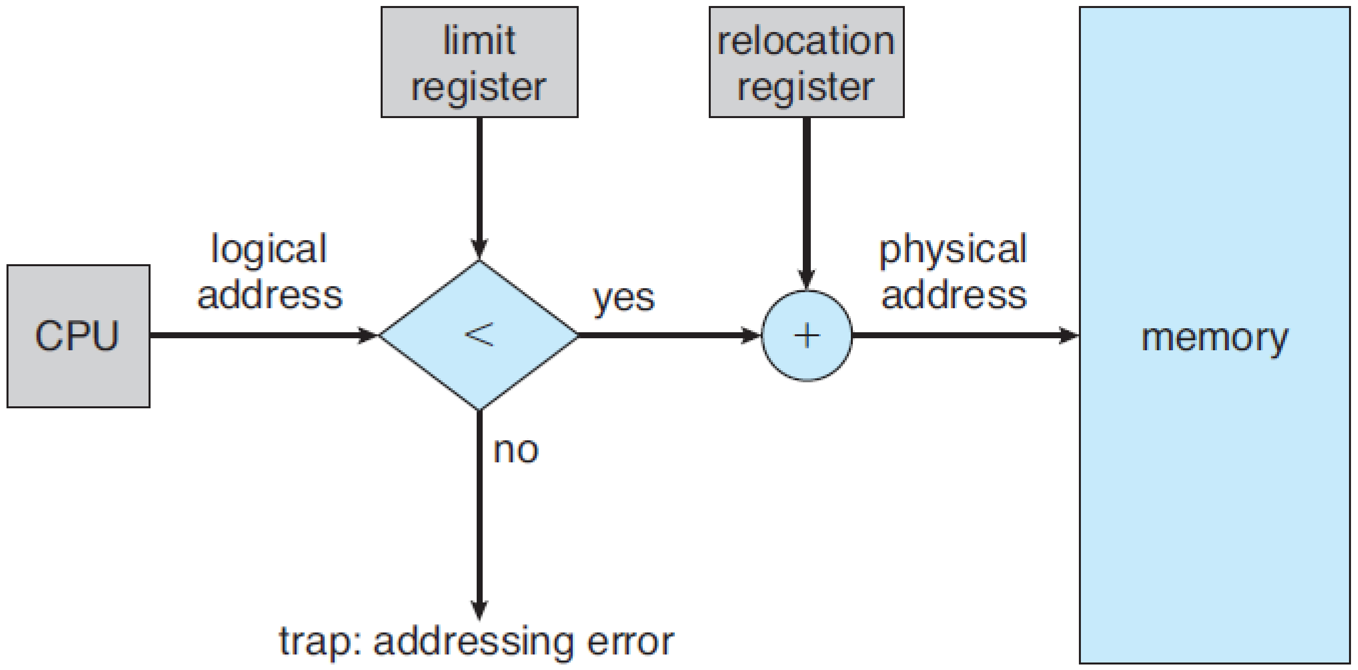


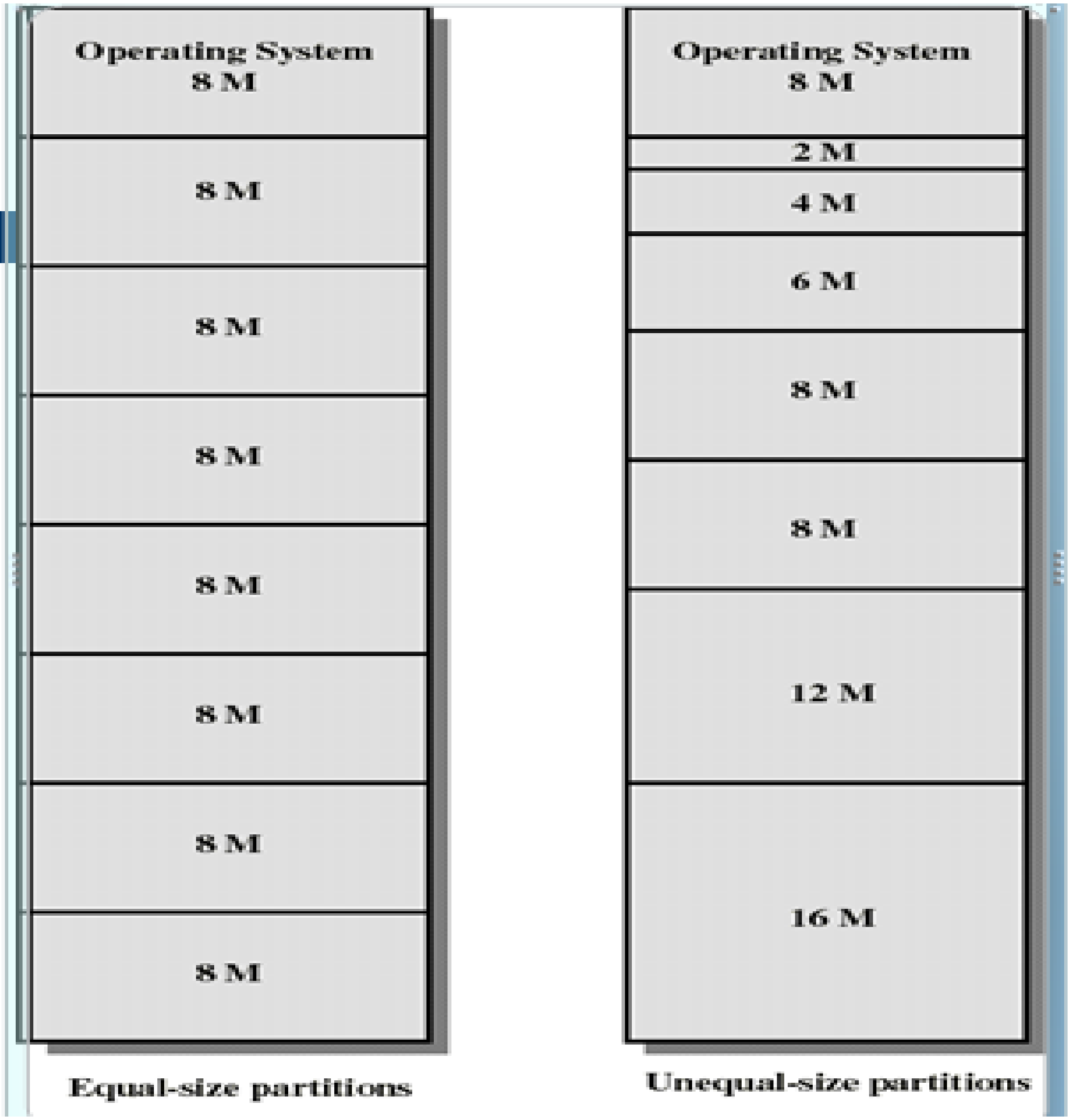
Figure 8.6 Hardware support for relocation and limit registers.

Memory allocation

- One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions.
- Each partition may contain exactly one process.
- Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition.
- When the process terminates, the partition becomes available for another process.
- This method was originally used by the IBM OS/360 operating system (called MFT)but is no longer in use.

Fixed partition

- Partition main memory into a set of non-overlapping memory regions called partitions.
- Fixed partitions can be of equal or unequal sizes.
- Leftover space in partition, after program assignment, is called internal fragmentation

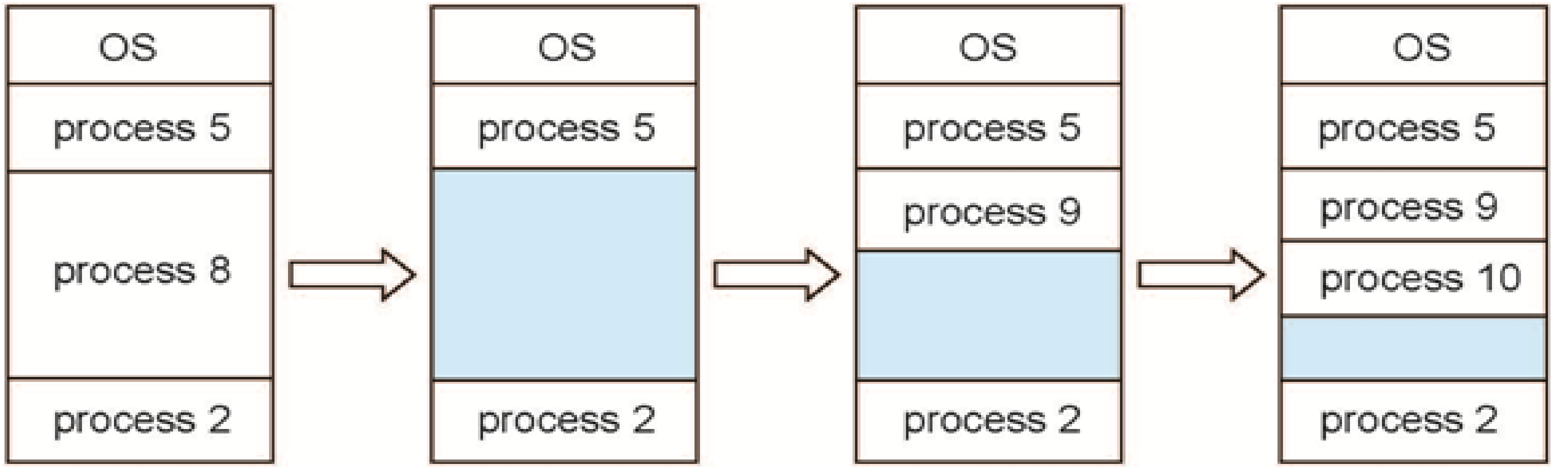

Operating System
8 M
8 M
8 M
8 M
8 M
8 M
8 M
8 M
Equal-size partitions
Operating System
8 M
2 M
4 M
6 M
8 M
8 M
12 M
16 M
Unequal-size partitions

Memory allocation

- In the variable-partition scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied.
- Initially, all memory is available for user processes and is considered one large block of available memory, a hole.
- Eventually,memory contains a set of holes of various sizes.
- As processes enter the system, they are put into an input queue.
- The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory.

Variable partition

- 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)



Memory allocation

- When a process is allocated space, it is loaded into memory, and it can then compete for CPU time.
- When a process terminates, it releases its memory, which the operating system may then fill with another process from the input queue.
- At any given time, then, we have a list of available block sizes and an
- input queue.
- The operating system can order the input queue according to a scheduling algorithm.
- Memory is allocated to processes until, finally, the memory requirements of the next process cannot be satisfied—that is, no available block of memory (or hole) is large enough to hold that process
- The operating system can then wait until a large enough block is available, or it can skip down the input queue to see whether the smaller memory requirements of some other process can be met.

Memory allocation

- In general, the memory blocks available comprise a ***set of*** holes of various sizes scattered throughout memory.
- When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process.
- If the hole is too large, it is split into two parts. 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 hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.
- At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.

Memory allocation

- **dynamic storage allocation problem, which concerns how to satisfy a request of size *n from a*** list of free holes.
- There are many solutions to this problem.
- The **first-fit, best-fit,** and **worst-fit strategies are the ones most commonly used to select a free hole** from the set of available holes.

- **First fit**. Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended .We can stop searching as soon as we find a free hole that is large enough.
- **Best fit**. Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- **Worst fit**. Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

- Simulations have shown that both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization.
- Neither first fit nor best fit is clearly better than the other in terms of storage utilization, but first fit is generally faster.

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.

Fragmentation

- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- **External fragmentation** exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous:
- storage is fragmented into a large number of small holes.
- This fragmentation problem can be severe.
- In the worst case, we could have a block of free (or wasted) memory between every two processes.
- If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

- Memory fragmentation can be internal as well as external.
- Consider a multiple-partition allocation scheme with a hole of 18,464 bytes.
- Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes.
- The overhead to keep track of this hole will be substantially larger than the hole itself.
- The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size.
- With this approach, the memory allocated to a process may be slightly larger than the requested memory.
- The difference between these two numbers is **internal fragmentation**—unused memory that is internal to a partition.

- One solution to the problem of external fragmentation is **compaction.**
- The goal is to shuffle the memory contents so as to place all free memory together in one large block.
- Compaction is not always possible, however.
- If relocation is static and is done at assembly or load time, compaction cannot be done.
- It is possible only if relocation is dynamic and is done at execution time.

- The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory.
- This scheme can be expensive.
- Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be noncontiguous, thus allowing a process to be allocated physical memory wherever such memory is
- available.
- Two complementary techniques achieve this solution: segmentation and paging. These techniques can also be combined