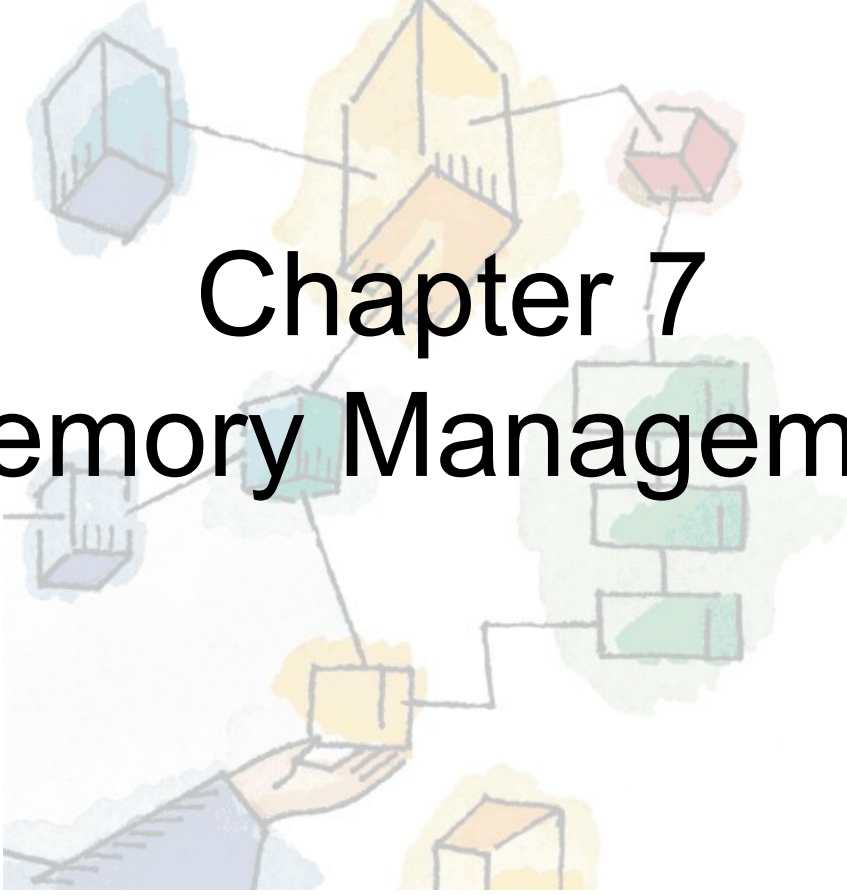


*Operating Systems:  
Internals and Design Principles, 8/E*  
William Stallings



# Chapter 7

## Memory Management

Patricia Roy  
Manatee Community College, Venice, FL  
©2015, Prentice Hall

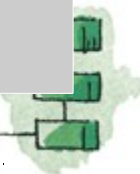
# Memory Management Terms

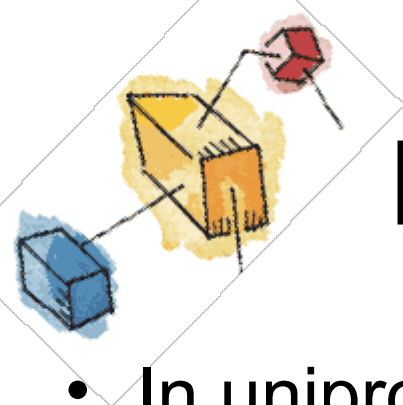
Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

# Related Command



Command	Description
Top	displays a list of the running tasks and various details about each application.
vmstat	shows summary information about memory, processes, interrupts, paging and block I/O information.
vmstat -S M	Shows memory in MB
free	shows a nicely summarized table containing information about the memory on your computer.
Ps	displays a list of running processes on the computer (similar to the top command)
ps aux	list of all the running processes for all users on the system
kill <pid>	close processes using their respective interfaces or through the command line

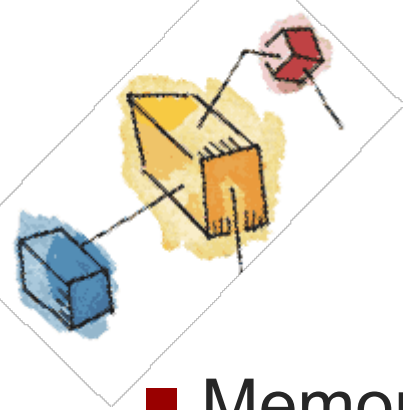




# Memory Management

- In uniprogramming, main memory is divided into two parts:
  - OS (kernel), and currently running program
- In multiprogramming, memory is further subdivided to accommodate multiple processes
  - The task of subdivision is called **memory management**
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time

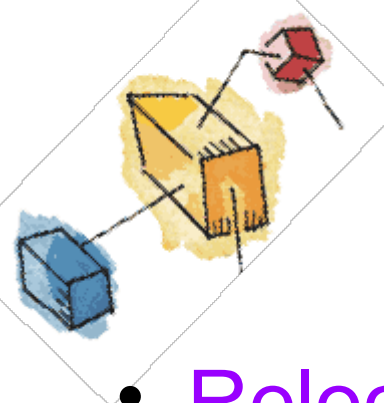




# Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization





# Memory Management Requirements

- Relocation

- In a multiprogramming system, the available main memory is shared among a number of processes.
- Programmer does not know where the program will be placed in memory when it is executed
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (**relocated**)
- Memory references must be translated in the code into actual physical memory address (**current location in main memory**)



# Addressing Requirement

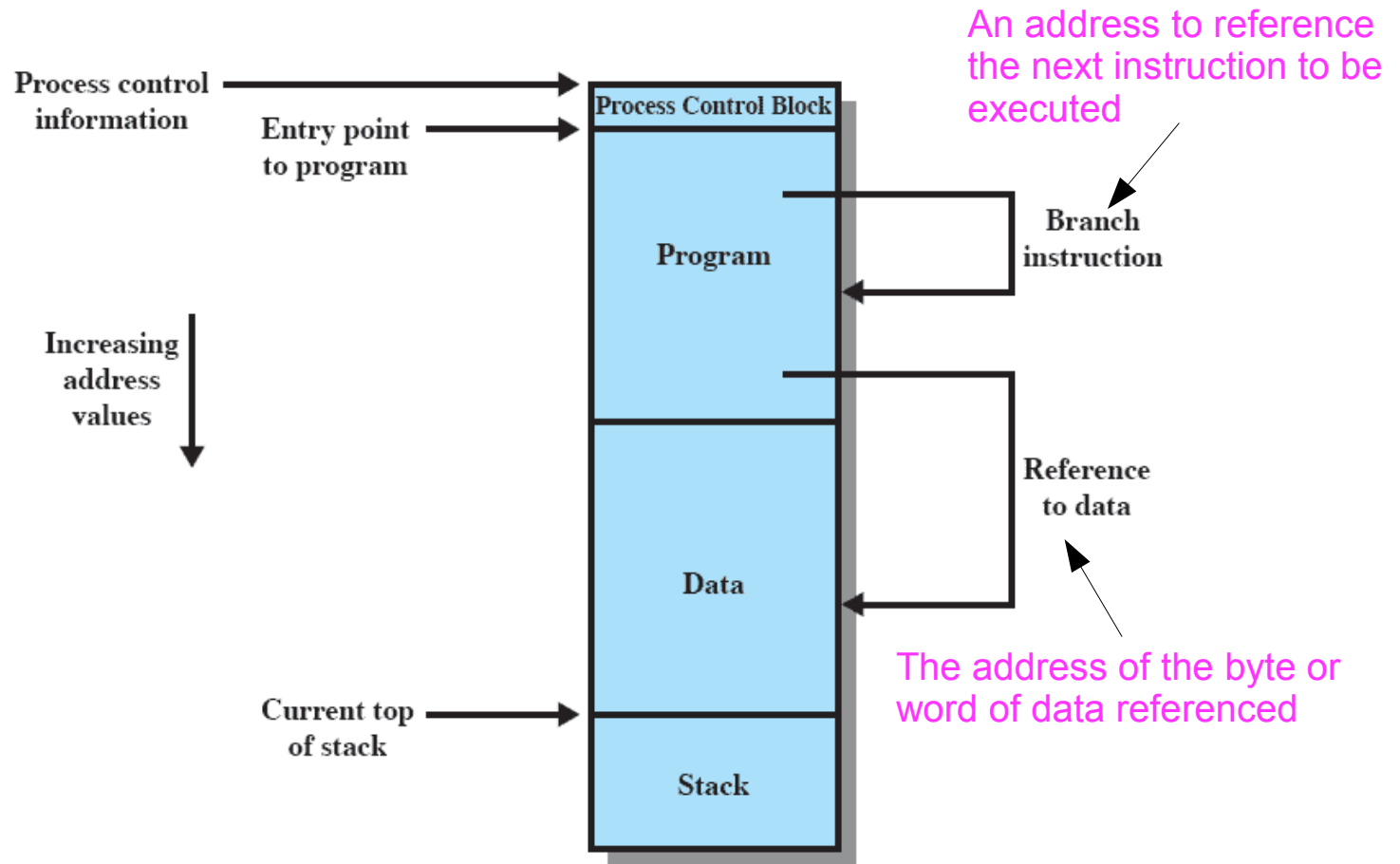
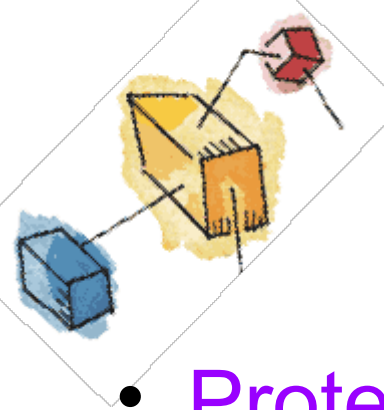


Figure 7.1 Addressing Requirements for a Process

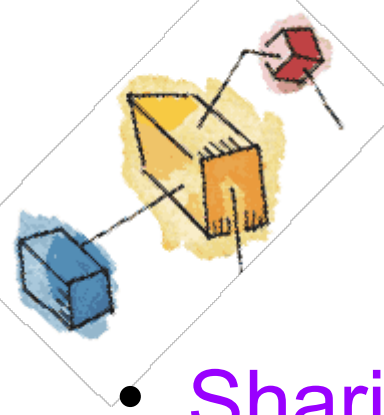


# Memory Management Requirements

- **Protection**
  - Each process should be protected against unwanted interference by other processes
  - Processes should not be able to reference memory locations in another process without permission
  - Relocation is unpredictable, make it difficult to satisfy the protection requirement.
    - Impossible to check **absolute addresses** at compile time
  - Thus, all memory reference must be checked at run time
  - Without arrangement, a program in one process cannot access the data area or another process.



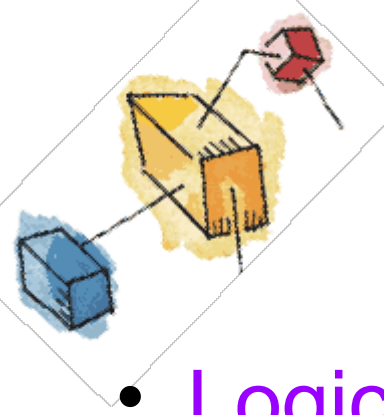




# Memory Management Requirements

- **Sharing**
  - Allow several processes to access the same portion of main memory
  - Better to allow each process access to the same copy of the program rather than have their own separate copy

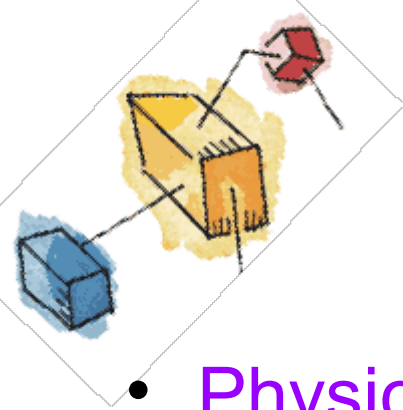




# Memory Management Requirements

- Logical Organization
  - Programs are written in modules
  - Modules can be written and compiled independently
  - Different degrees of protection given to different modules (read-only, execute-only)
  - Introduce a mechanism by which modules can be shared among processes
    - e.g: segmentation

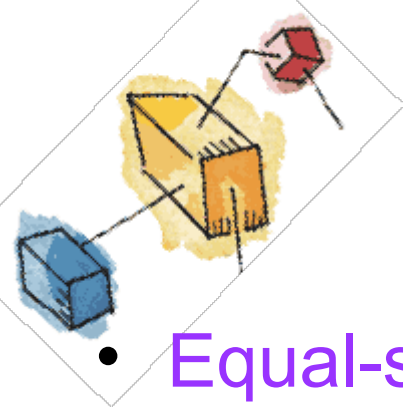




# Memory Management Requirements

- **Physical Organization**
  - Computer memory is divided into:
    - Main memory – fast access, high cost
    - Secondary memory – slower access, cheaper, long term storage
  - Impractical reasons for assigning responsibility for the flow organization of program to individual programmer:
    - Memory available for a program plus its data may be insufficient
      - Overlaying allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available





# Memory Partitioning:

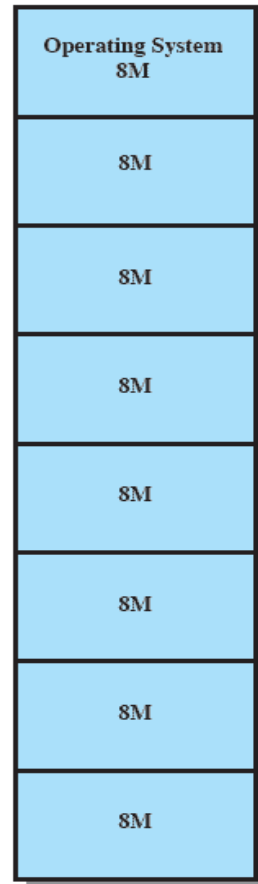
## Fixed Partitioning

- Equal-size partitions

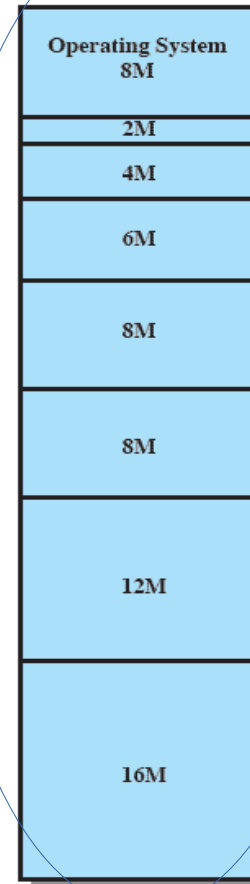
- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition and load in another process
- Disadvantages:
  - A program may not fit in a partition. The programmer must design the program with overlays
  - Main memory use is extremely inefficient. Any program, no matter how small, occupies an entire partition.
    - Program is 2 Mb, partition size is 8Mb
    - Wasted space - called **internal fragmentation**.



# Fixed Partitioning



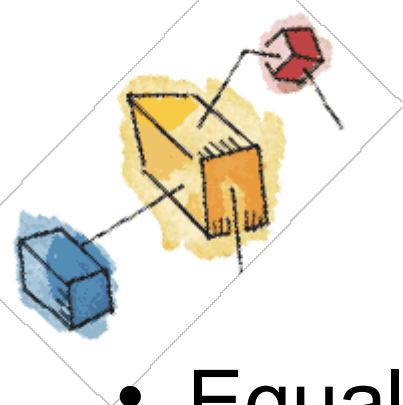
(a) Equal-size partitions



(b) Unequal-size partitions

Solution to equal-size

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory



# Placement Algorithm

- Equal-size
  - Placement of processes in memory is trivial
  - As long as there is any available memory, load the process
- Unequal-size
  - Can assign each process to the smallest partition within which it will fit
  - Scheduling queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition



# Fixed Partitioning

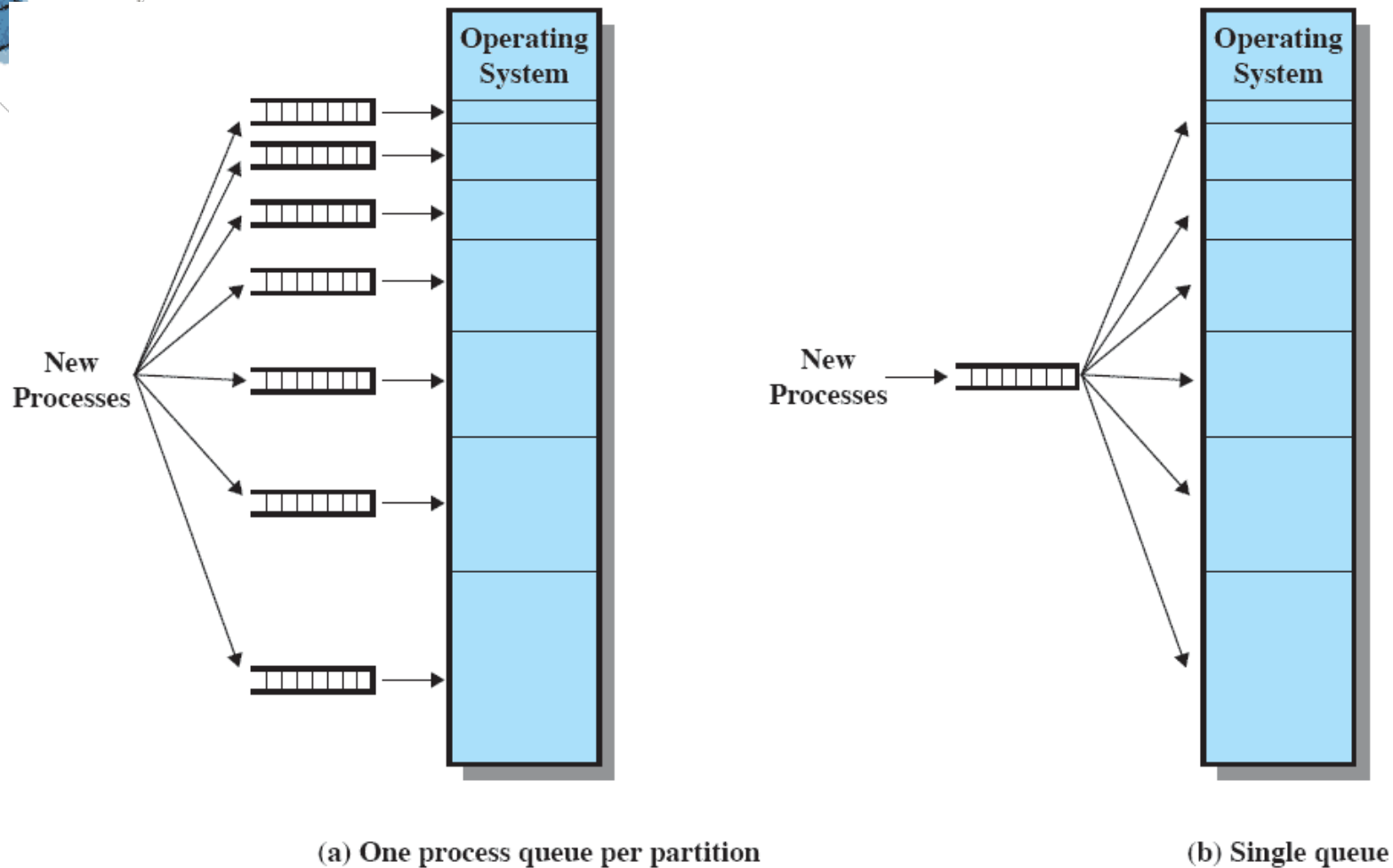
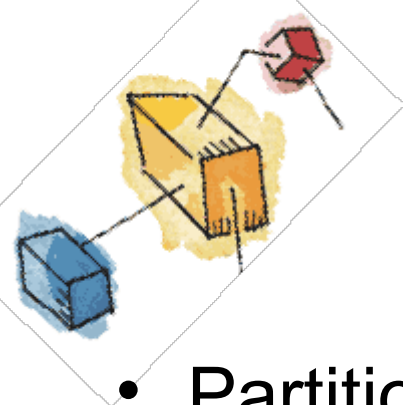


Figure 7.3 Memory Assignment for Fixed Partitioning

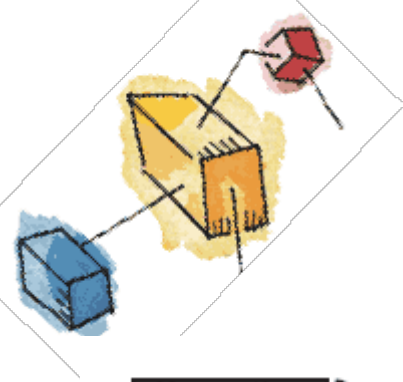


# Dynamic Partitioning

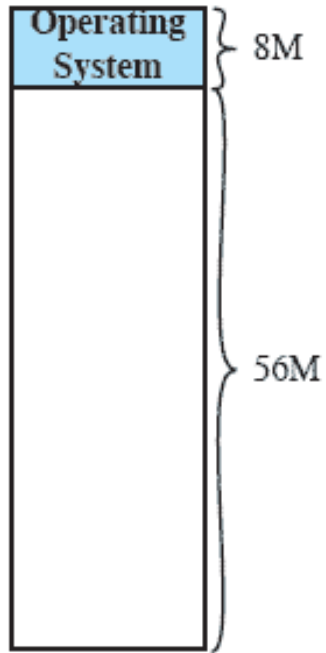
- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called **external fragmentation**
- Must use **compaction** to shift processes so they are contiguous and all free memory is in one block
  - But, it is time consuming



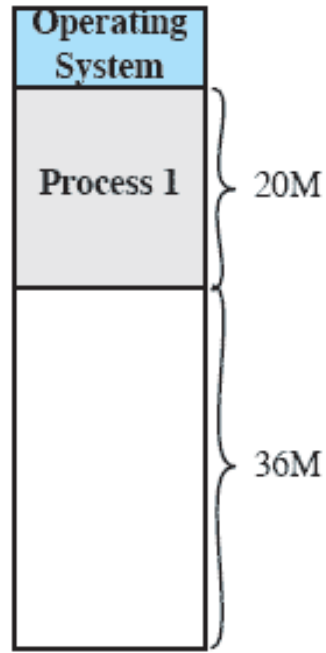




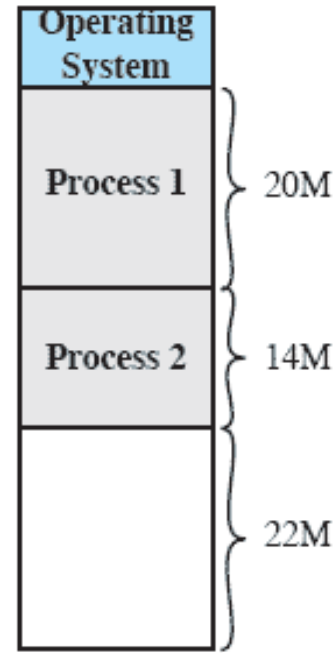
# Dynamic Partitioning



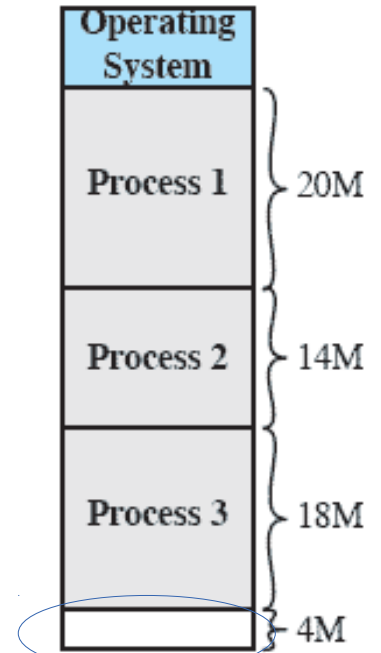
(a)



(b)



(c)

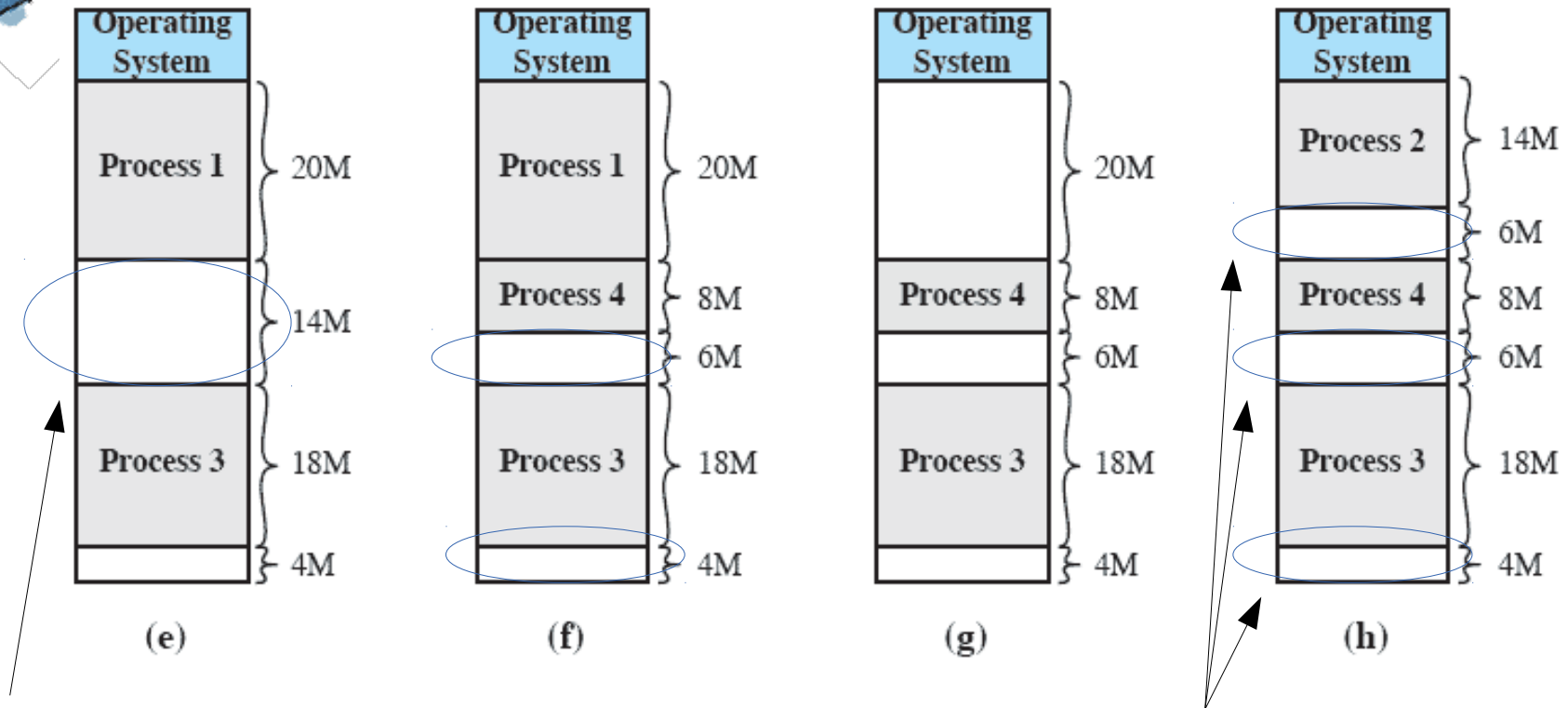


(d)

External fragmentation



# Dynamic Partitioning

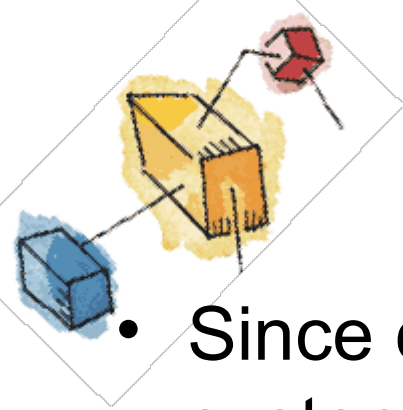


Swapped out

Should be shifted together

**Figure 7.4 The Effect of Dynamic Partitioning**

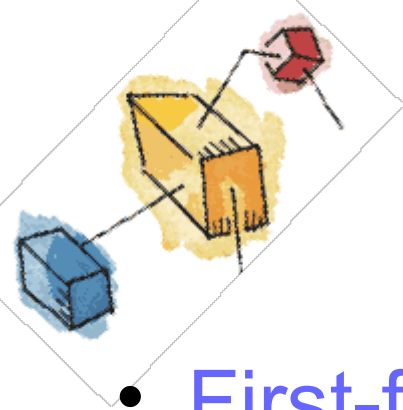




# Dynamic Partitioning

- Since compaction is time consuming, operating system must be clever to decide which free block to allocate to a process
- **Best-fit algorithm**
  - Chooses the block that is closest in size to the request
  - Worst performer overall
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often

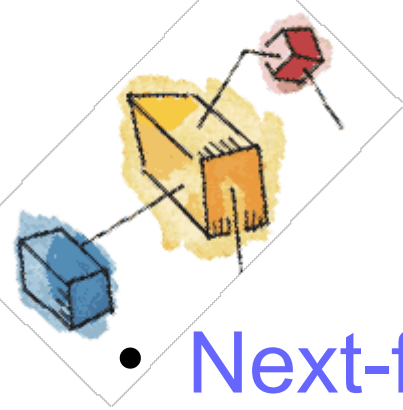




# Dynamic Partitioning

- **First-fit algorithm**
  - Scans memory from the beginning and chooses the first available block that is large enough
  - Fastest
  - May have many processes loaded in the front end of memory that must be searched over when trying to find a free block





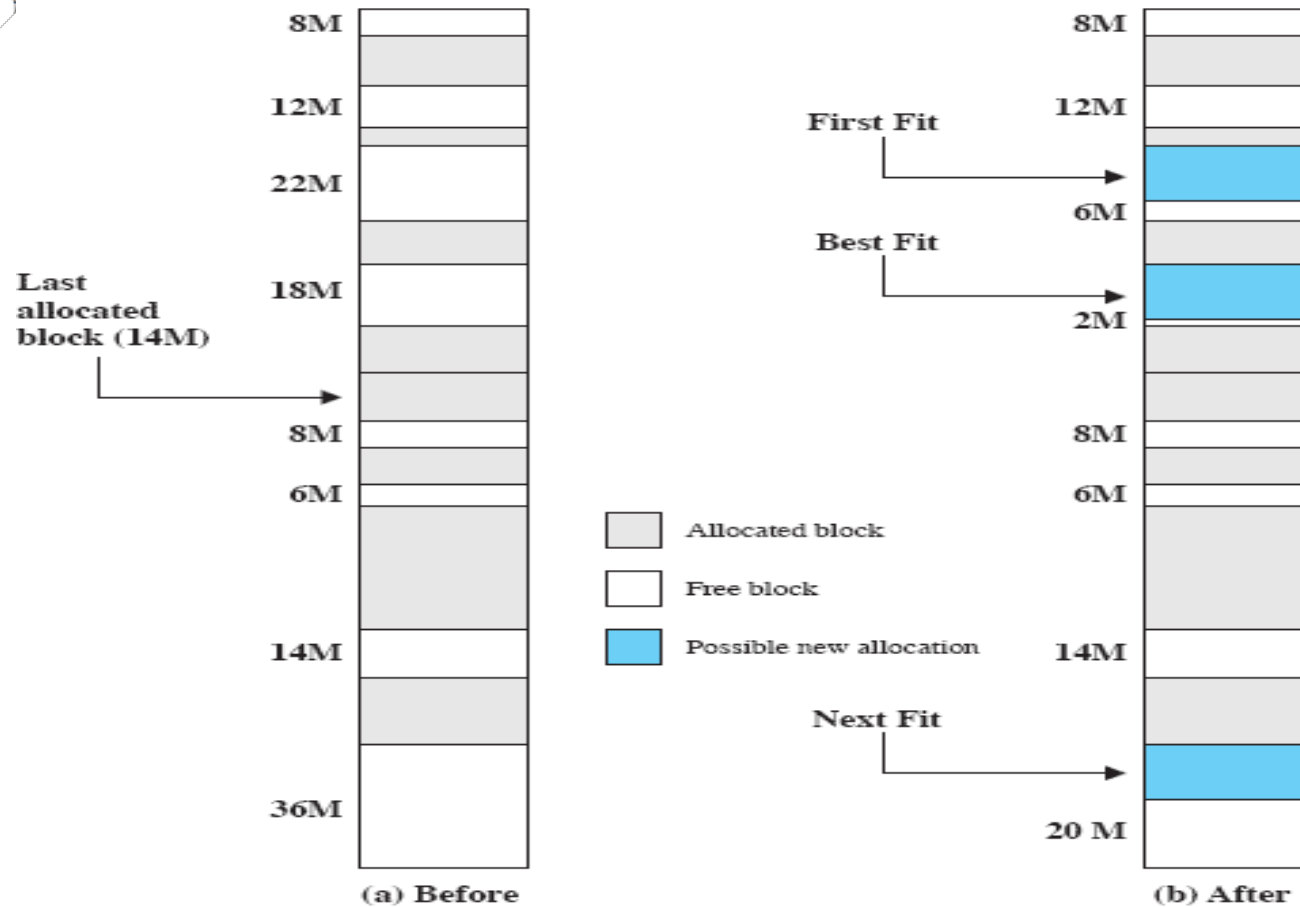
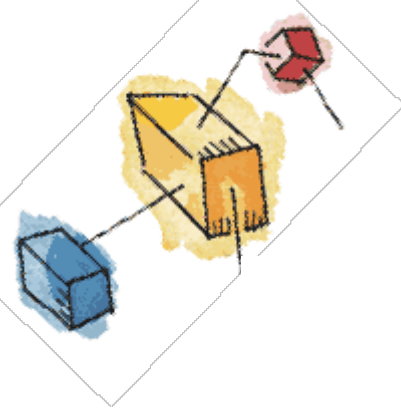
# Dynamic Partitioning

- Next-fit

- Scans memory from the location of the last placement, choose the one that is large enough
- More often allocate a block of memory at the end of memory where the largest block is found
- The largest block of memory is broken up into smaller blocks
- Compaction is required to obtain a large block at the end of memory



# Allocation



**Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block**

