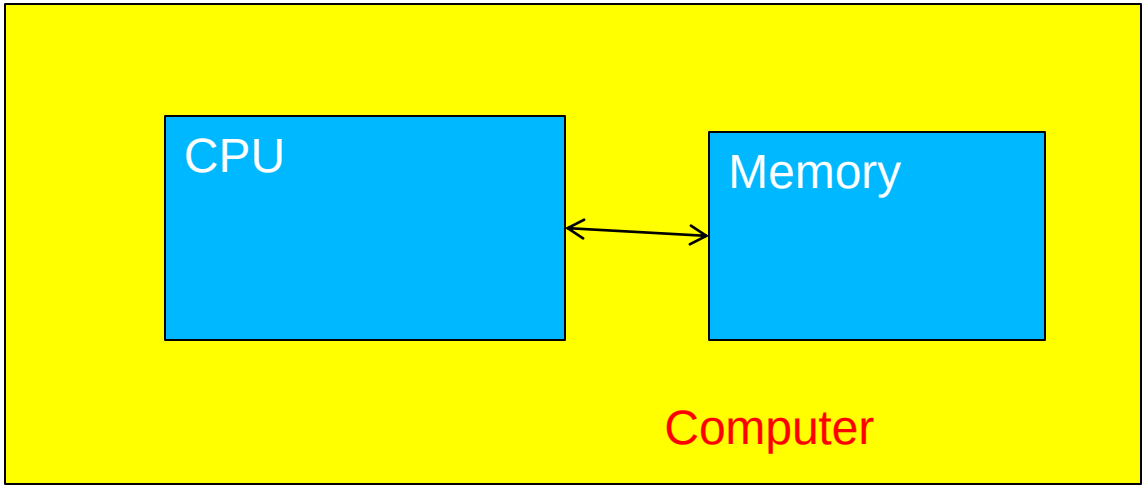
Unit V

Memory Management

Contents

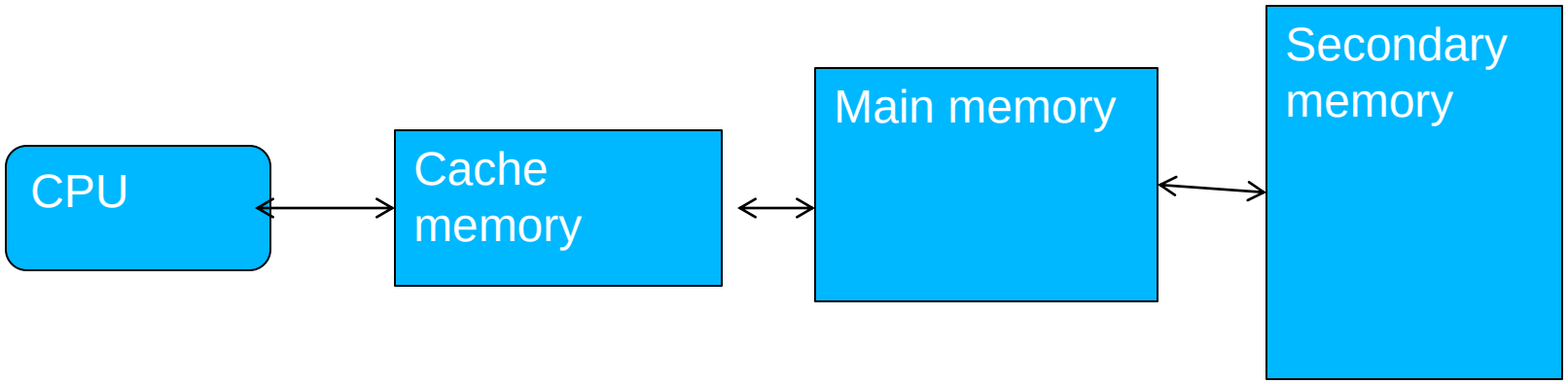
- Memory Management Requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

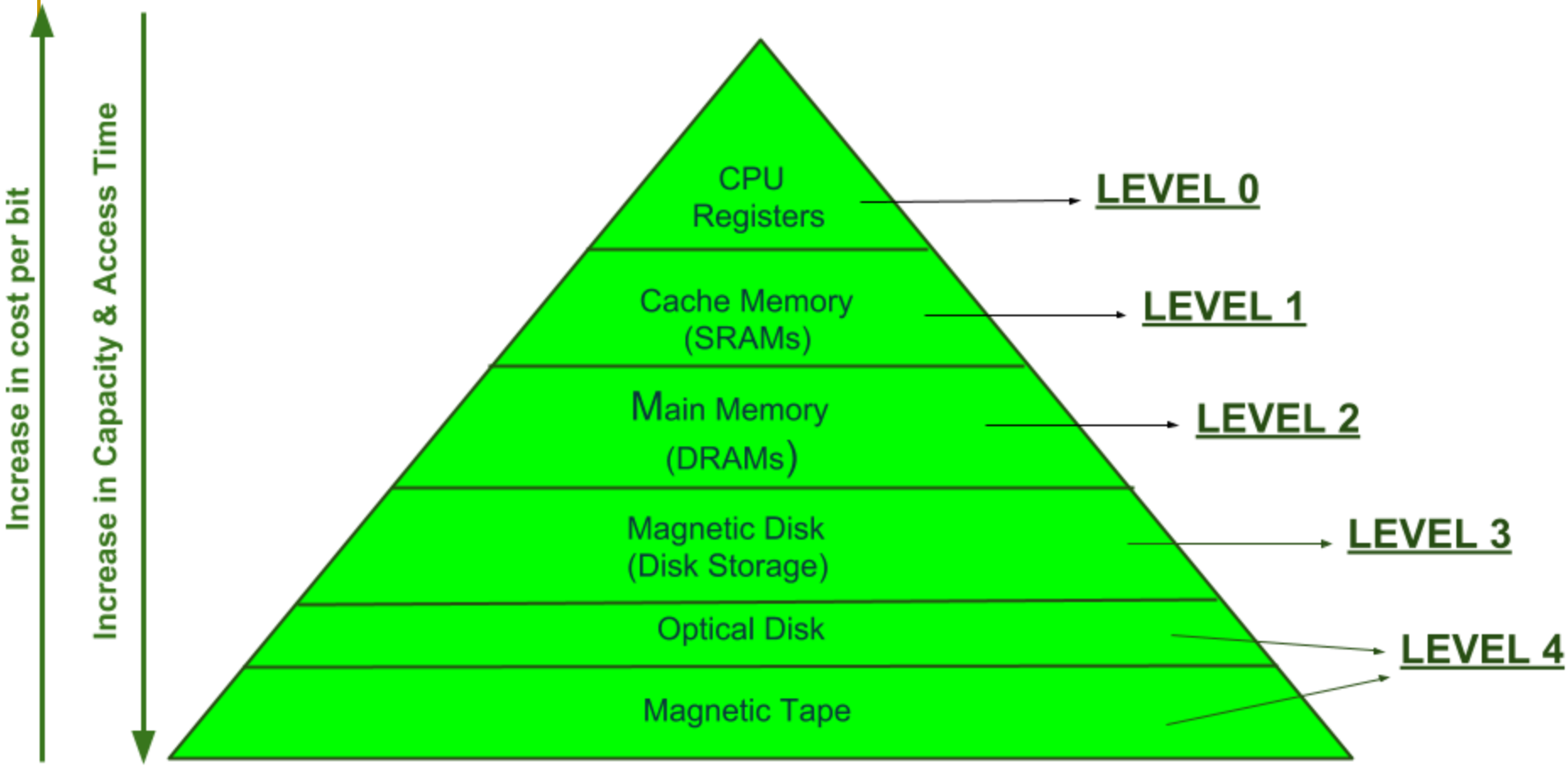
Introduction



- Desirable properties of memory
- Size -more
- Access time -less
- Per unit cost -less

Introduction





MEMORY HIERARCHY DESIGN

- This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory –**

   Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

2. **Internal Memory or Primary Memory –**

   Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

- **characteristics of Memory Hierarchy Design**
- **Capacity:**

  It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
- **Access Time:**

  It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
-

- Performance:

  Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

- Cost per bit:

  As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Memory Management

- In a uniprogramming system, main memory is divided into two parts:
  - one part for the operating system (resident monitor, kernel) and
  - one part for the program currently being executed.
- In a multiprogramming system, the “user” part of memory must be further subdivided to accommodate multiple processes.
- The task of subdivision is carried out dynamically by OS is known as **Memory Management.**
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.
- Memory Management, involves swapping blocks of data from secondary storage

Execution of a Program

- Operating system brings into main memory a few pieces of the program.
- Resident set - portion of process that is in main memory.
- An interrupt is generated when an address is needed that is not in main memory.
- Operating system places the process in a blocking state.

Memory Management Requirements

1. Relocation
2. Protection
3. Sharing
4. Logical organisation
5. Physical organisation

Memory Management Terms

Term	Description
Frame	***Fixed***-length block of main memory.
Page	***Fixed***-length block of data in secondary memory (e.g. on disk).
Segment	***Variable-length*** block of data that resides in secondary memory.

Memory Management Requirements

1. Relocation:

- 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 to actual physical memory address.

Addressing

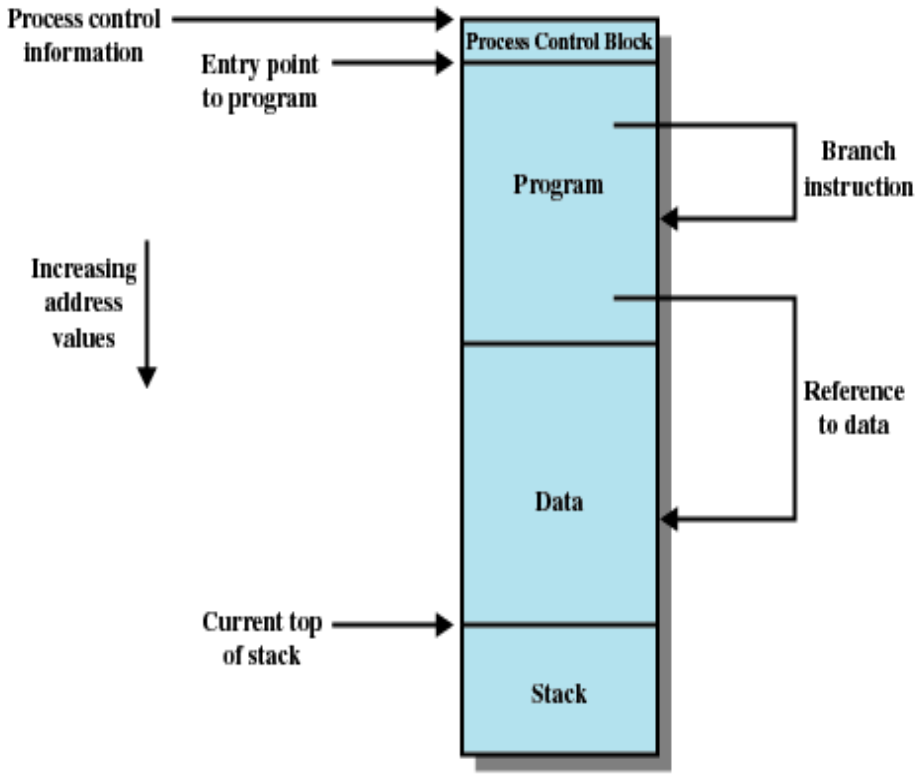


Figure 7.1 Addressing Requirements for a Process

The OS needs to know the location of:

- process control information
- the execution stack,
- the entry point to begin execution of the program for this process.

There will be memory references within the program

Eg. BT, GD, PD

Processor HW and OS should be able to translate these into actual physical memory addresses

Changes every time

Memory Management Requirements

2. Protection

- Processes should not be able to reference memory locations in another process without permission.
- Impossible to check absolute addresses at compile time.
- Must be checked at run time.
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
  - Operating system cannot anticipate all of the memory references a program will make

Memory Management Requirements

3. Sharing

- Allow several processes to access the same portion of memory
  - Ex. Number of processes executing same program
- Better to allow each process access to the same copy of the program rather than have their own separate copy
- Protection mechanism must have flexibility to allow several processes to access the same portion of memory
- Processes that are cooperating on some task may need to share access to the same data structure.

Memory Management Requirements

4. Logical Organization

- Memory is organized linearly (usually)

- Main memory is usually organized as a linear, or 1-D address space, consisting of a sequence of bytes or words.

- Secondary memory, at its physical level, is similarly organized.

If memory can also be dealt with in form of modules, there are several advantages(Programs are written in modules)

- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules among processes.

- Ex: use of segmentation uses module concept

Memory Management Requirements

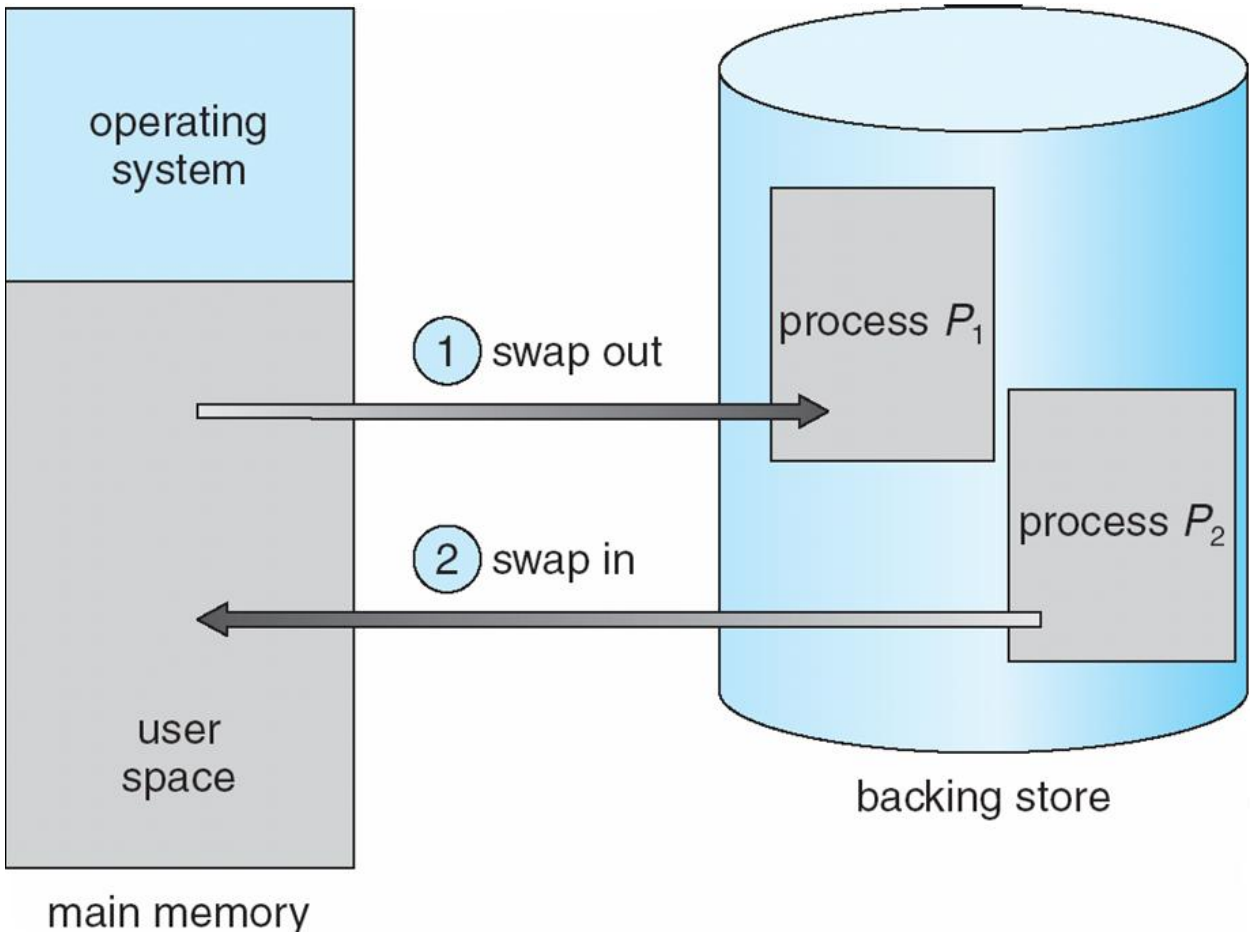
5. Physical Organization

- Memory is organized into Two –levels
- Main memory :fast , expensive, volatile
- Secondary memory : Slow , less expensive, non-volatile
- Flow of information b/n main & secondary memory is major concern
- Can it be programmer's responsibility?
- No
- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory
  - Programmer dependent
  - Wastage of time of programmer
  - Programmer does not know how much space will be available and where that space will be.

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

Swapping


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Swapping

- Benefits:

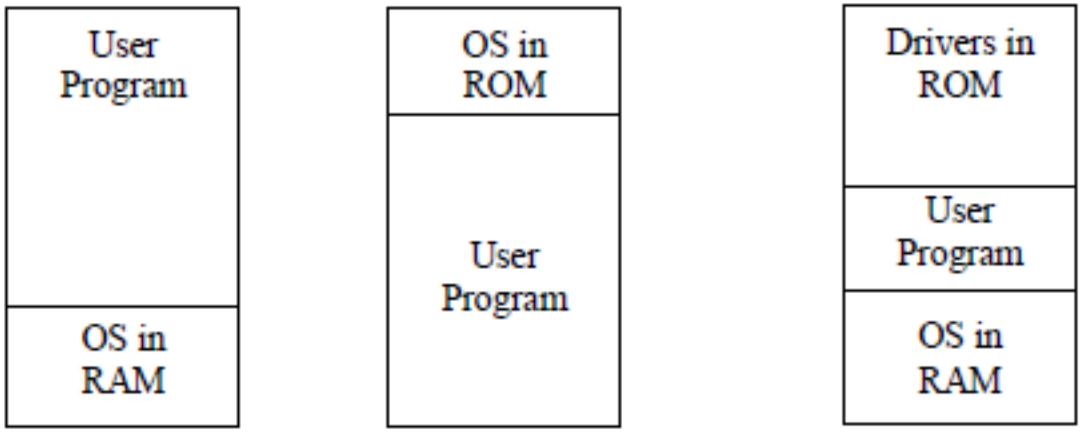
1. Allows higher degree of multiprogramming
2. Allow dynamic relocation
3. Better memory utilization
4. Less wastage of CPU time on compaction
5. Can easily be applied on priority based scheduling algorithms to improve performance

- Limitations:

1. Entire program must be resident in store when it is executing
2. Processes with changing memory requirements will need to issue system calls for requesting & releasing memory.

Single process monitor(Monoprogramming)

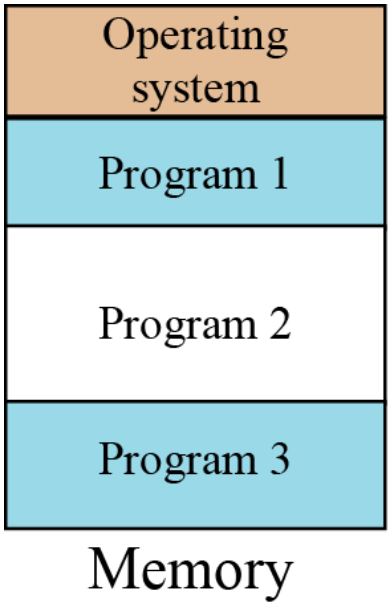
- Only one process in main memory
- No address translation during execution of program
- Protection of OS
- Ex: MS-DOS
- Limited capacity & performance



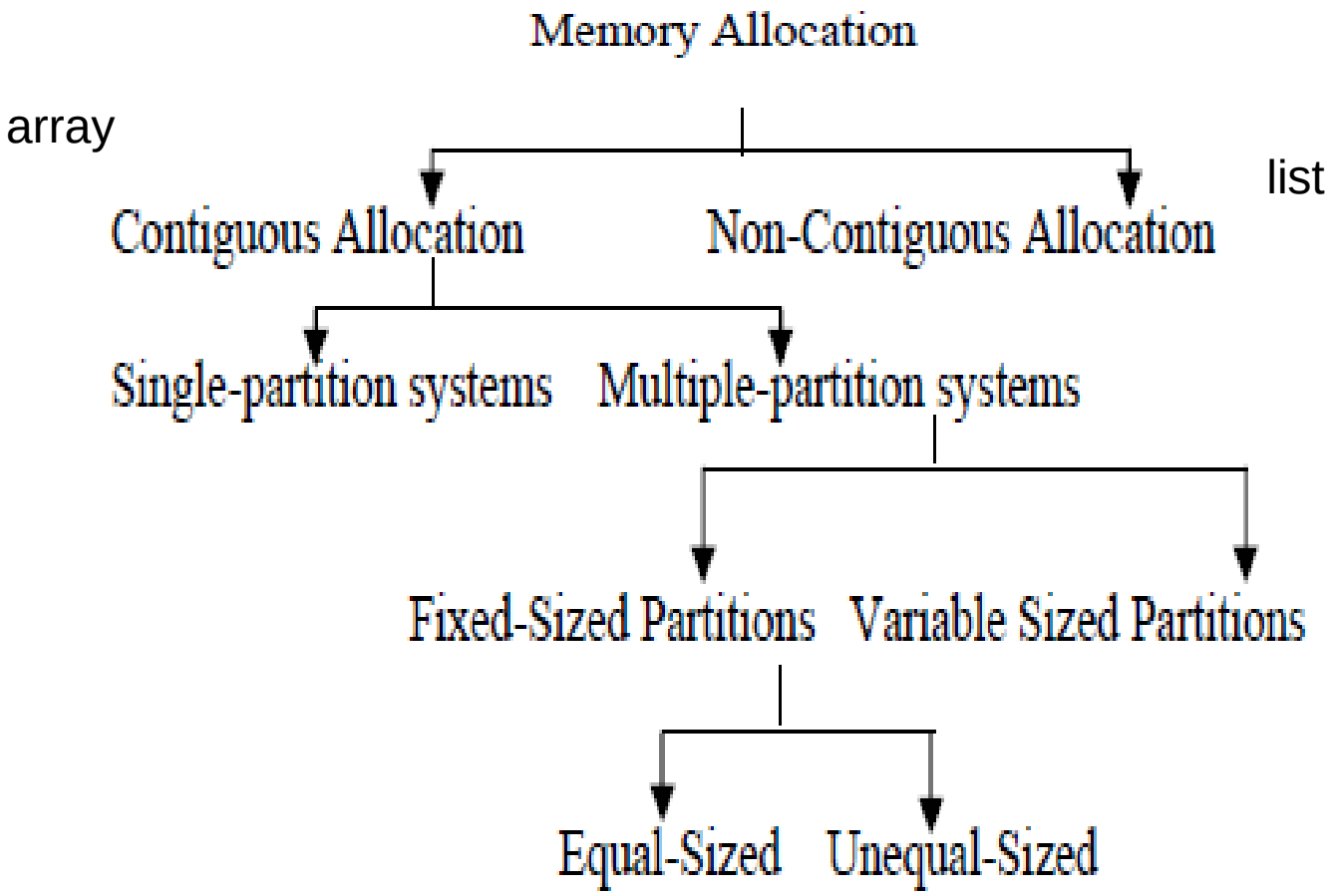
Contents

- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

Multiprogramming

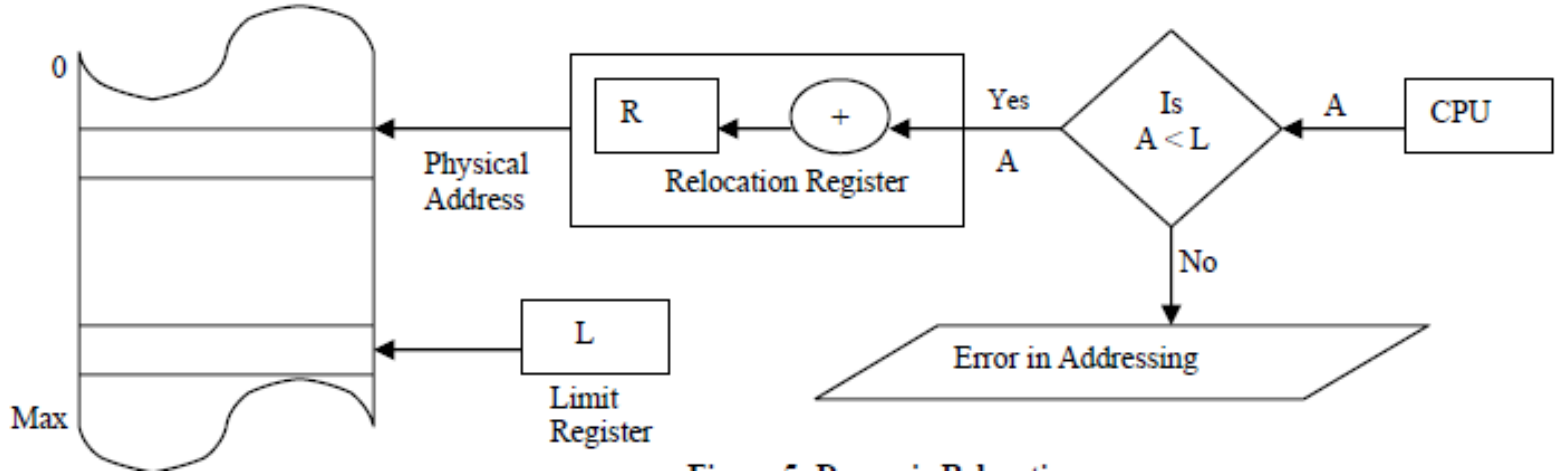


Memory Allocation Methods



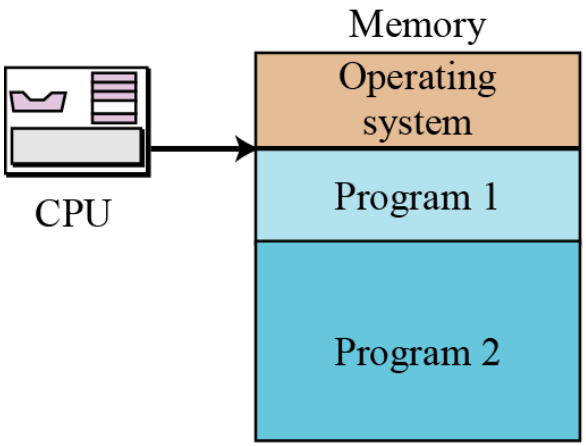
Single partition System

- OS can be protected by keeping it in lower part and user processes in upper part of memory
- Dynamic relocation is used using relocation register & base register
- Relocation register contains smallest physical address
- Limit register contains logical address range
- If logical address space is 0 to MAX then physical address space is R+0 to R+MAX

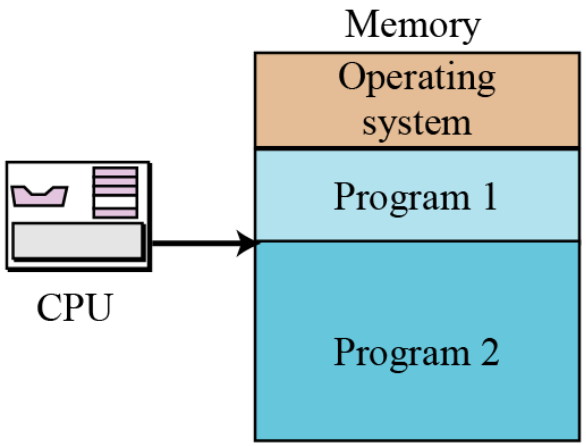


Memory Management

- Principal operation of MM is to bring process from secondary memory to main memory
- This typically involves Virtual Memory which in turn involves segmentation and/or paging
- Prior to Virtual Memory, other simpler techniques were used
  - Partitioning
    - Fixed
    - Variable / Dynamic
  - Concept of paging
  - Concept of segmentation



a. CPU starts executing program 1



b. CPU starts executing program 2

Contiguous Memory Partitioning

- Partition the available memory into regions with fixed boundaries.

1. Fixed Partitioning:

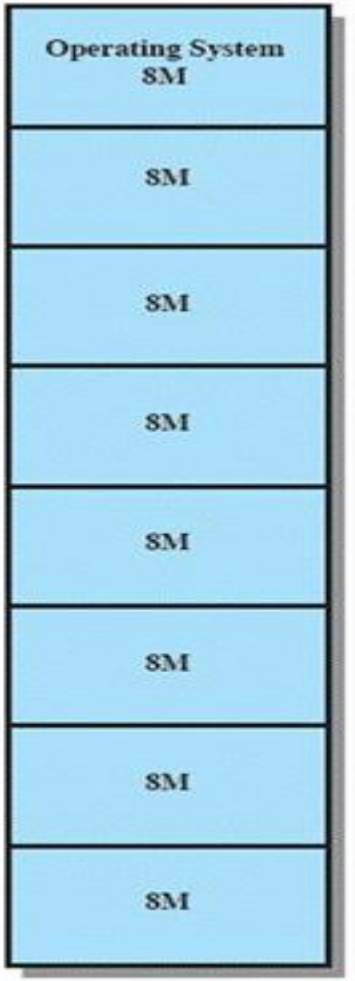
a) Equal-size Partitions

b) Unequal-size Partitions

2. Variable 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



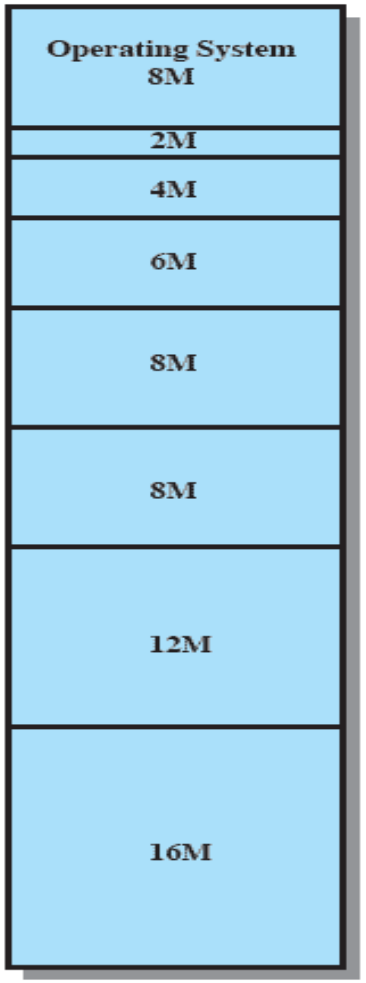
(a) Equal-size partitions

Fixed Partitioning

- Problems:
- A program may be too big to fit in any partition.
  - Use of overlays is compulsory.
- Main memory use is inefficient.
- Any program, no matter how small, occupies an entire partition.
- Wasted space internal to the partition called internal fragmentation

Solution: Unequal Size Partitions

- Lessens both problems
  - but doesn't solve completely
- In Figure
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation



(b) Unequal-size partitions

Placement Algorithm with Partitions

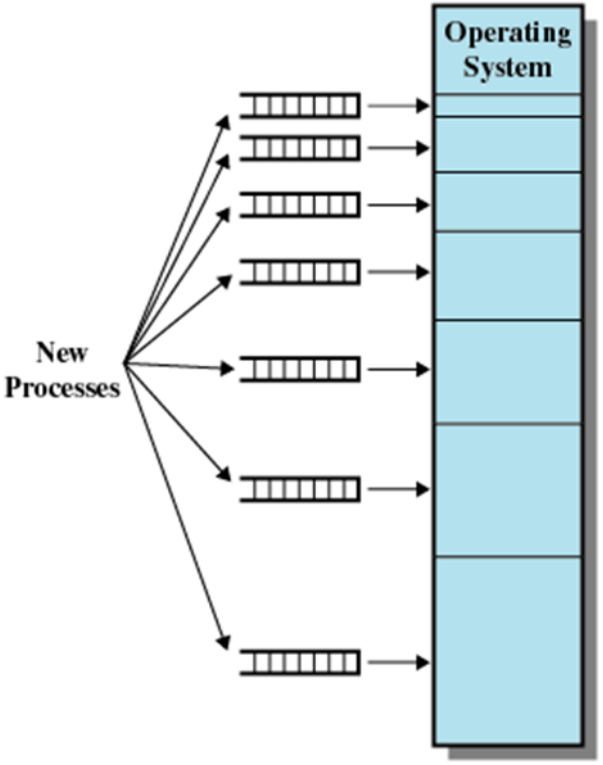
- Equal-size partitions
  - Because all partitions are of equal size, it does not matter which partition is used.
- Unequal-size partitions
  - Can assign each process to the smallest partition within which it will fit.
  - Queue for each partition.
  - Processes are assigned in such a way as to minimize wasted memory within a partition.

Placement Algorithm

- Equal sized
  - If any partition is available, load the process there
    - Does not matter which partition is used
  - If all partitions occupied with processes not ready to run, swap one out and load this guy in
    - Which one to swap out is a scheduling decision
- Unequal sized
  - Two possible strategies

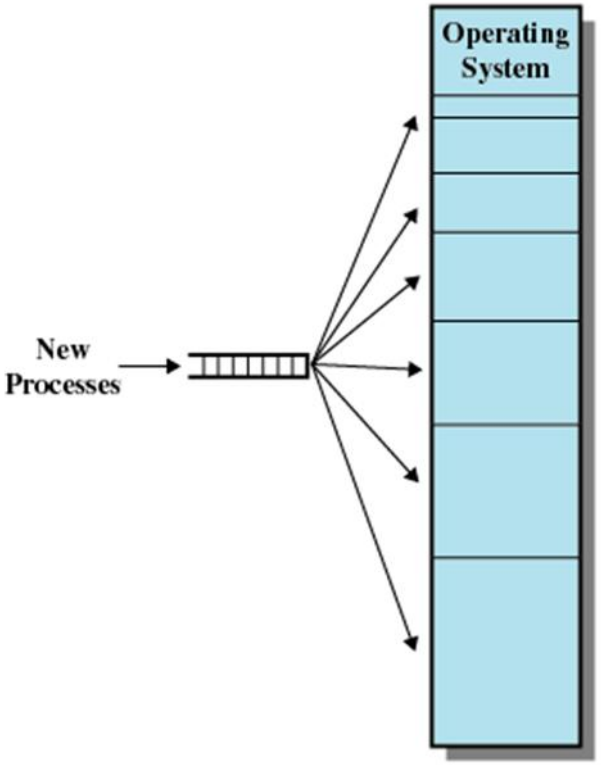
Strategy 1

- Assign to smallest partition possible
  - May have to wait because somebody else is already there
- Needs a scheduling queue for each partition to hold swapped out processes destined for that partition
- If memory requirement is not clear, overlaying or virtual memory are the only solutions
- Advantage: minimum internal fragmentation
- Disadvantage?
  - Small process cannot go to bigger partition even if available



Strategy 2

- Single queue
- Smallest available partition is selected when needed
- If all are occupied, then swap out somebody
  - Smallest partition process that is enough to hold this guy
  - Priority
  - Process state



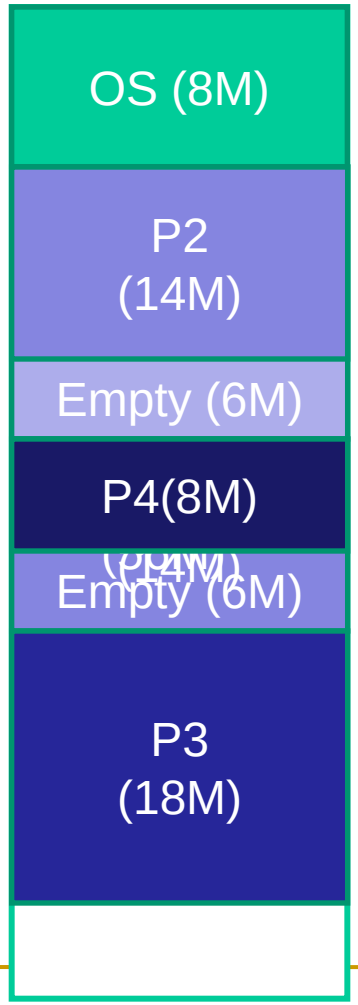
Fixed Partitioning

- Advantages
  - Simple, require minimal OS and processing overhead
- Disadvantages
  - Number of partitions specified at system generation time limits the number of active processes system can support
  - Internal fragmentation cannot be completely eliminated
    - It is always possible to get small jobs which do not utilize partitions fully
- Fixed partitioning is no where to be seen today
- Example: IBM Mainframe's OS/MFT(Multiprogramming with fixed number of tasks)

Dynamic Partitioning

- Purpose:to overcome the difficulties of fixed 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
- Example:I BM Mainframe's OS/MVT (Multiprogramming with variable number of tasks)

Dynamic Partitioning Example



- ***External Fragmentation***
- Memory external to all processes is fragmented
- Can resolve using **coalescing holes** & ***compaction***

Dynamic Partitioning Example

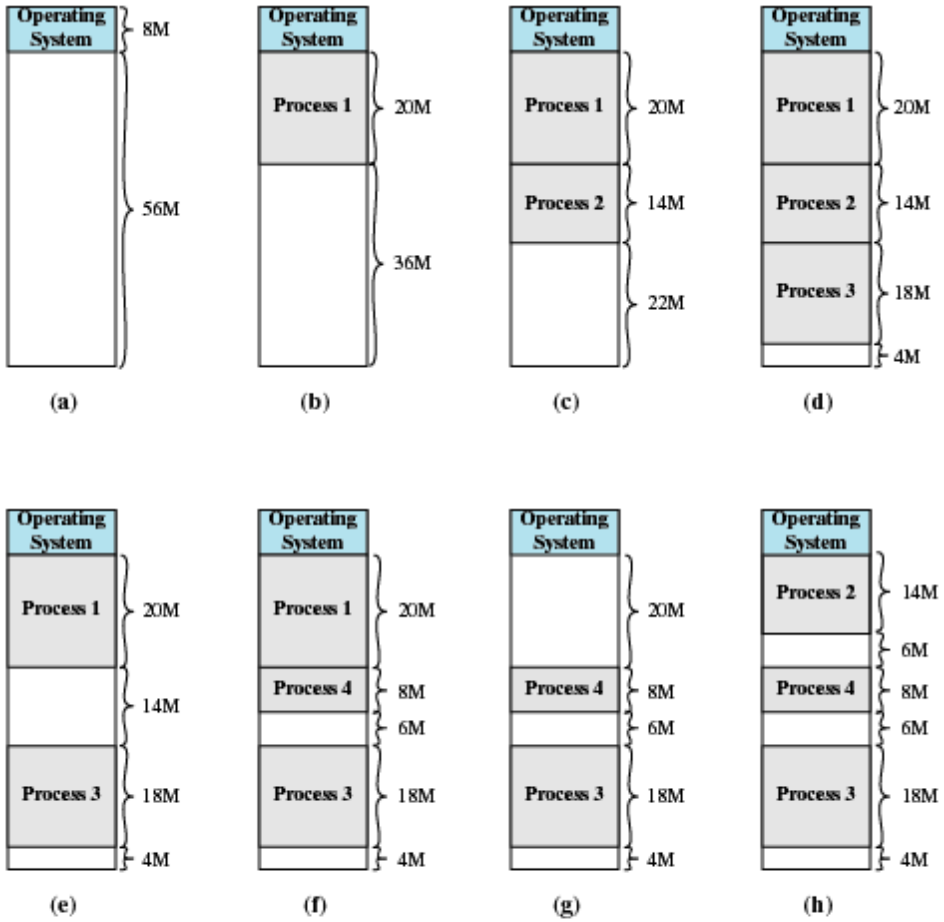


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

- Coalescing holes is the process of merging existing hole adjacent to a process that will terminate and free its allocation space. So that new adjacent hole & existing holes can be viewed as a single large hole and can be efficiently used.
- In Compaction, shuffle all occupied areas of memory to one end and leave all free memory space as a large block

- Problem with compaction?
  - Extra overhead in terms of resource utilization & large response time
  - Expensive
  - Needs dynamic relocation of processes in memory

Contents

- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

Dynamic Partitioning Placement Algorithm

- Purpose: To overcome on problem of compaction
- Operating system must decide which free block to allocate to a process
- When more than one choice available, OS must decide cleverly which hole to fill
- Hole must be big enough to accommodate the to-be-loaded process
- Placement algorithms:

1. First fit
2. Best fit
3. Next fit
4. Worst fit

Dynamic Partitioning Placement Algorithm

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

Dynamic Partitioning Placement Algorithm

Best-fit algorithm

- scan all holes to see which is best
- 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 Placement Algorithm

- Next-fit
  - Scans memory from the location of the last placement
  - 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

•The last block that was used was a 22-Mbyte block from which a 14-Mbyte partition was created.

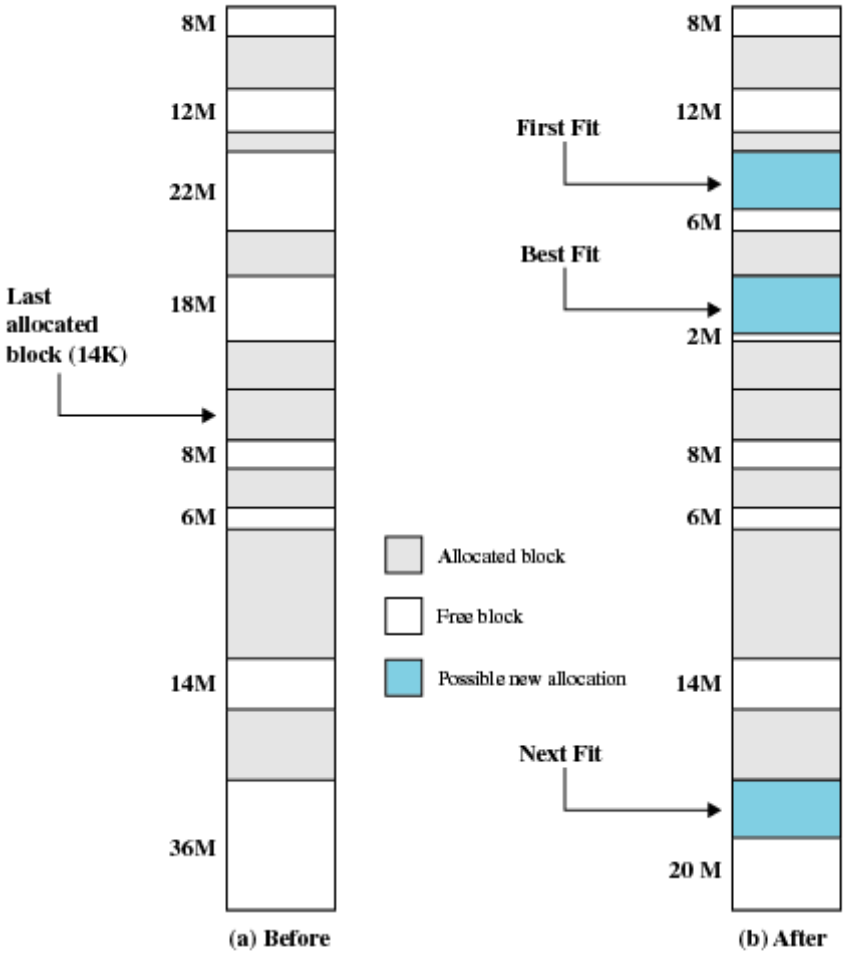


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Comparison

- Depends on exact sequence of process swapping and size of processes
- First fit
  - Simplest and usually the best and fastest
  - Splits regions towards beginning requiring more searches
- Next fit produces slightly worse results than first fit
  - Tends to allocate more frequently towards the end of the memory, thus largest block of free memory which usually appears at the end is quickly fragmented requiring more frequent compaction
- Best fit is usually the worst performer
  - Every allocation leaves behind smallest fragment of no good use
  - Requires compaction even more frequently
- How about worst-fit strategy?

- 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

Example

- Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 212 KB, 417KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

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 in memory is 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
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers