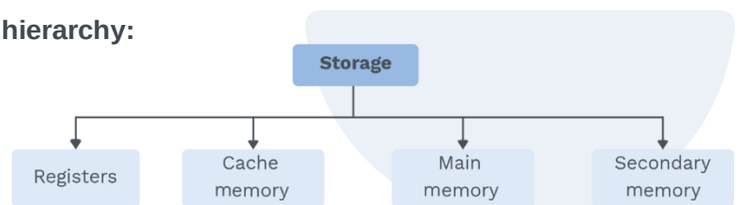
5 Memory Management

5.1 BASICS OF MEMORY MANAGEMENT

- The main purpose of a computer system is to execute programs. These programs, along with their data, must at least be partially present in the main memory during execution.
- To greatly enhance the utilisation of the CPU and its response time to users, many programs exist in the main memory. Thus various memory management methods are present to manage memory in different situations.
- Memory Management provides the functionality of allocating and de-allocating the main memory to the processes. It helps the operating system to keep a record of all the memory locations which are allocated to any process or are freely available.

Memory hierarchy:



Different places at which data is stored in a computer system in a hierarchical manner.

1) Registers:

- A CPU register is one of a small number of data storage areas found within the computer processor.
- Registers are a type of computer memory that is used to execute programs quickly and efficiently by storing data that is often used. The sole function of a register is to allow for quick retrieval of data for processing.

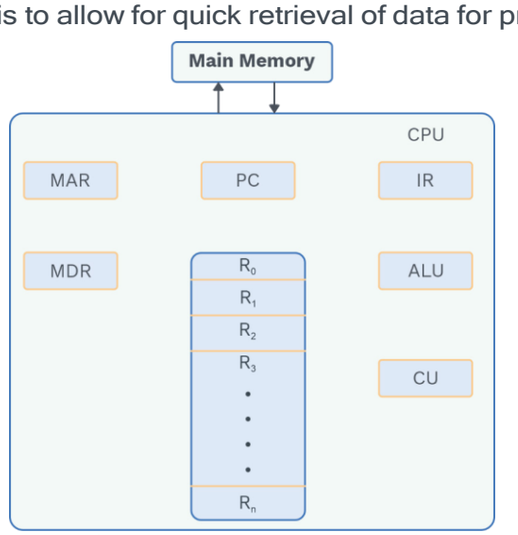
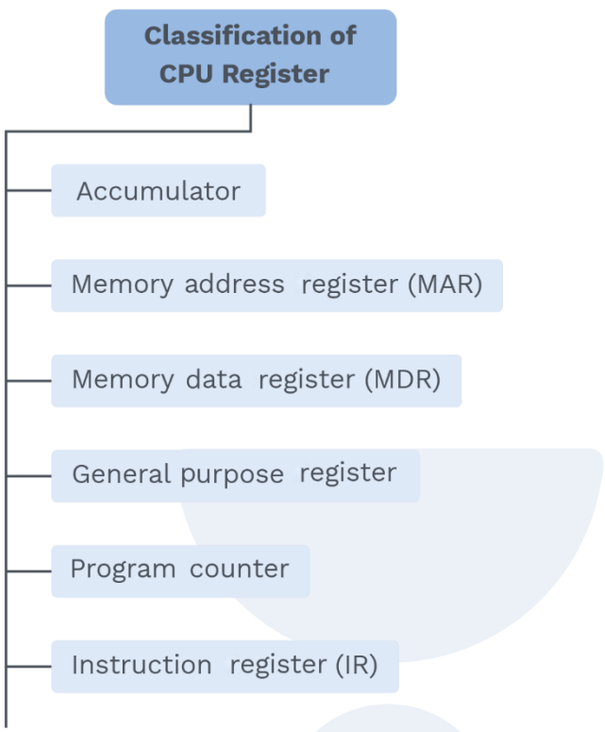


Fig. 5.1 Different Types of Register



- The CPU has direct access to registers; it can retrieve data from registers in a single clock cycle.

2) Cache memory:

- Cache memory is a volatile computer memory that saves frequently used applications and enables high-speed data access to a CPU.
- Cache memory is more expensive than main memory or disc memory, but it is less expensive than registers. It serves as a buffer between the processor and the main memory.
- Cache memory is used to lower the average time to access data from the main memory by storing copies of data from frequently accessed main memory locations. It is smaller and faster than the main memory.

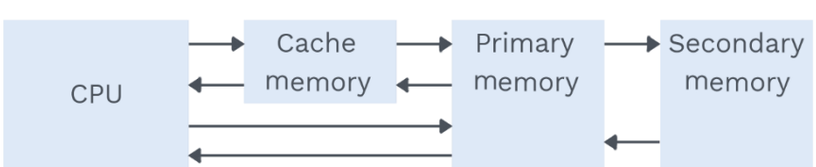


Fig. 5.2 Levels of memory hierarchy

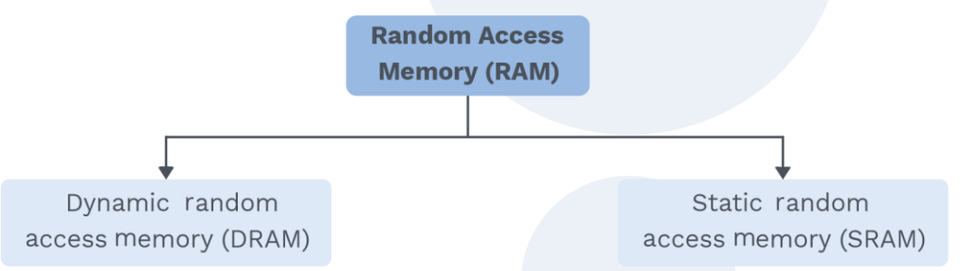
Note:

Data transfer is in words between CPU and cache memory and in block between cache and main memory.

3) Main memory:

It's also known as read-write memory, primary memory, or main memory.

- Because the data is lost when the power is switched off, it is a volatile memory.
- Random Access Memory is another name for it (RAM). It is the portion of the computer that contains the operating system, software applications, and other data for the processor. The term "random access" refers to the fact that the CPU can go to any portion of the main memory without having to proceed in sequence.



Note:

Data transfer between main memory and secondary memory is done by the operating system.

i) DRAM:

The most prevalent type of main memory in a computer is DRAM. In PCs, it is a common memory source. DRAM is continually recovering whatever data is currently stored in memory. It sends millions of pulses per second to the memory cells to refresh the data.

ii) SRAM:

It's a popular choice for embedded devices. SRAM data does not need to be refreshed on a regular basis. When the power is turned off, the information in this RAM stays as a static image until it is overwritten or destroyed. When not in use, it is less dense and more energy efficient.

4) Secondary memory:

- It is non-volatile and persistent computer memory that cannot be accessed directly by a computer.
- Primary memory has limited storage capacity and is volatile; this limitation is overcome by secondary memory.
- It is slower in data accessing. Typically main memory is at least six times faster than the secondary memory.

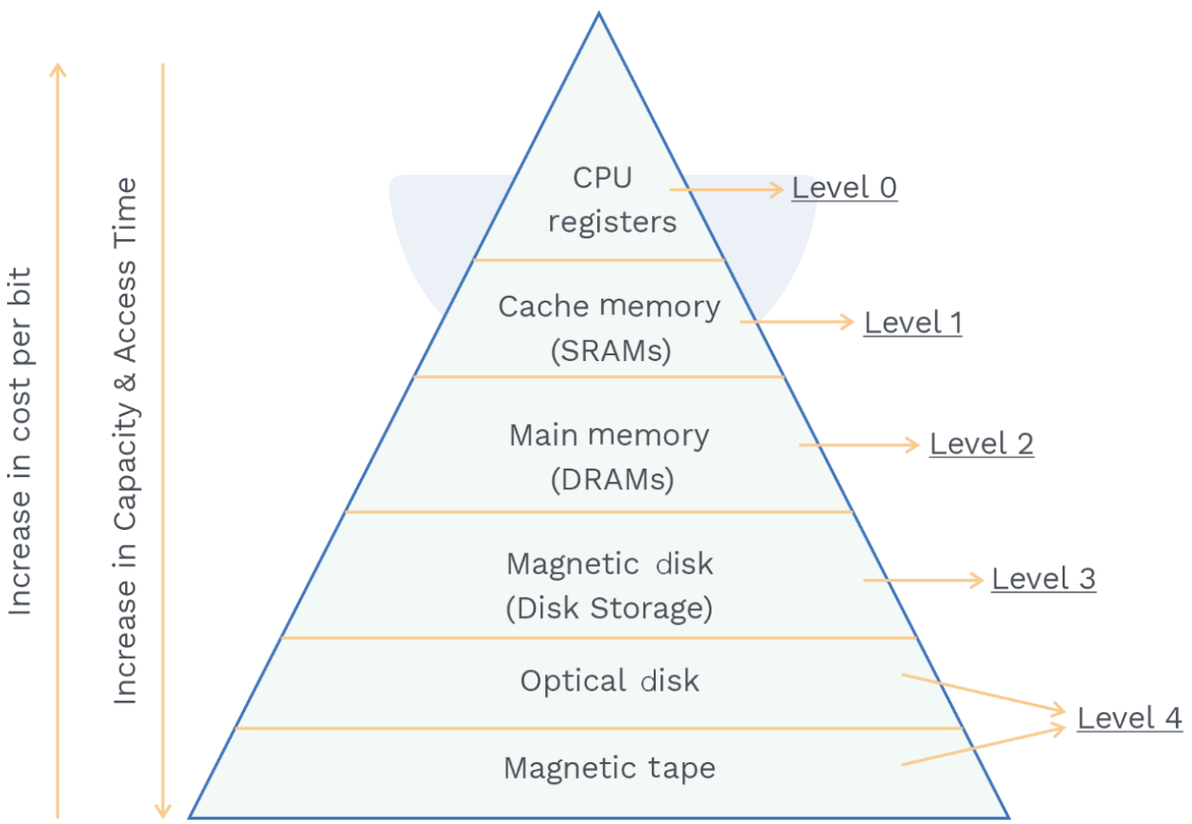


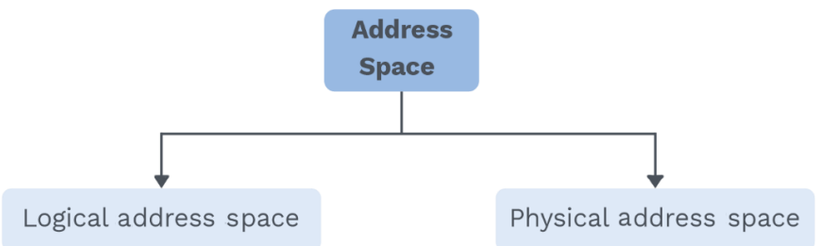
Fig. 5.3 Memory Hierarchy Design

Address space:

- An address space is the collection of all the addresses that are allocated to the program for its storage and execution. The memory locations in the address space can be accessed by the programs or the processes.

Classification of address space:

- Address space can be logical address space for storing the program or can be physical address space for executing the program.

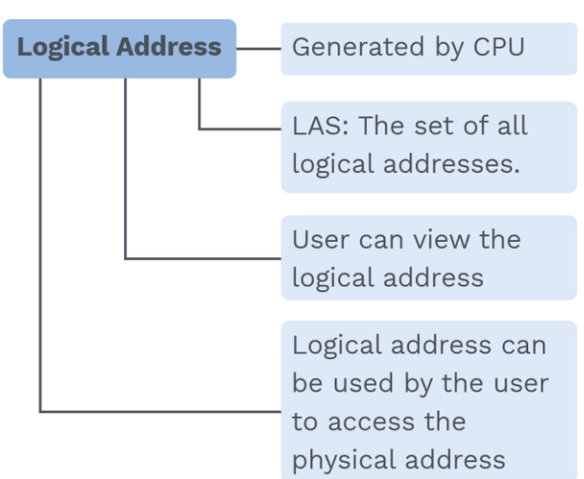


1) Logical address space:

- CPU generates logical addresses.
- It is used by the CPU to access the physical memory location as a reference.
- The set of all logical addresses generated from a program's point of view is known as the logical address space.

2) Physical address space:

- In memory, a physical address represents the physical location of requested data. The user never interacts with the physical address directly but can access it via its logical address.
- The logical address is generated by the user software.
- The hardware mechanism that converts a logical address to its physical address is the Memory-Management Unit (MMU).
- MMU must first map the logical address to the physical address.



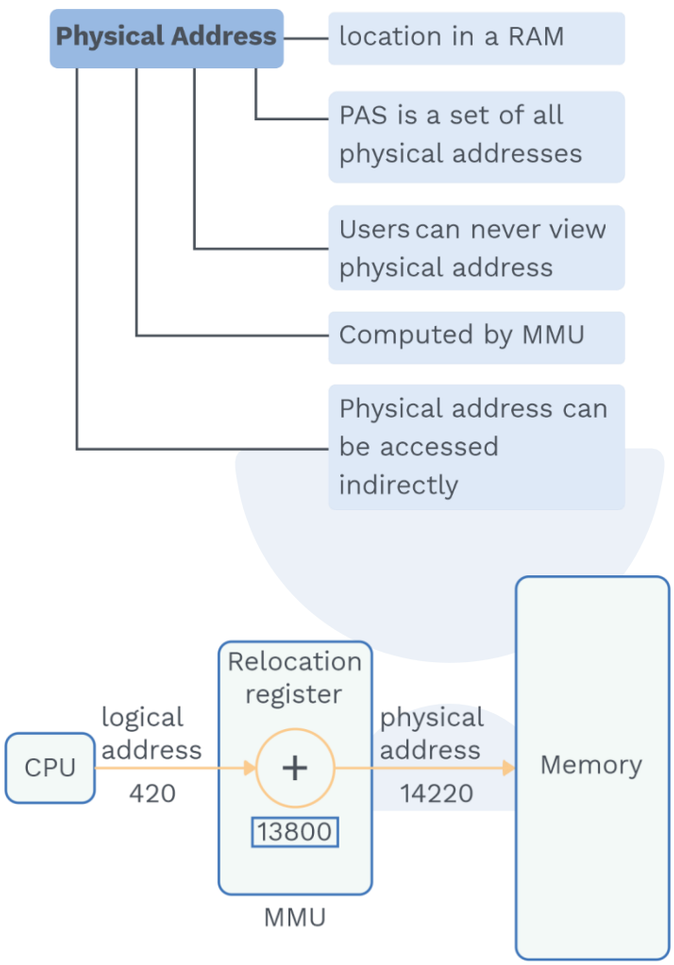


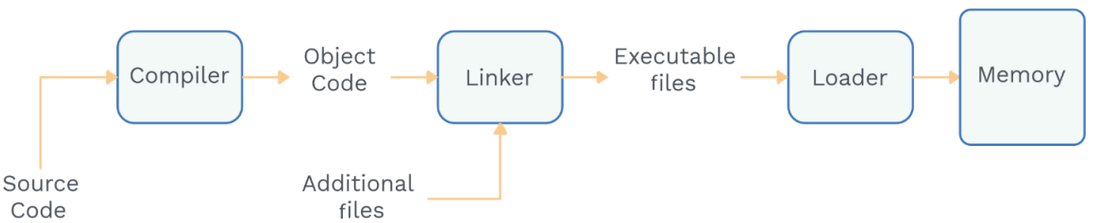
Fig. 5.4 Translation of Logical Address into Physical Address

Linker and loader:

Definition

The utility programs Linker and Loader play an important part in the execution of a program. Before being executed, a program's source code passes via the compiler, assembler, linker, and loader in that order.

Linking and loading:



1) Linking:

A program has multiple modules in it, which we compile together in order to generate a single executable code that can be loaded into the main memory for its execution. There are some codes which are reused, again and again; they are known as library functions. All different object files and library functions are combined into a single executable code; this process is known as linking. It can be done in two ways:

i) Static linking:

- Static linking is done during the compile time. In static linking, all the object modules and the library functions are combined in an executable file.
- Static linking is performed by special programs called linkers.
- Linker appends, all the library functions needed to execute the object file, which is generated by the compiler/assembler.

Advantage:

Programs in which all the modules and library functions are linked statically are faster than the programs in which all the modules and library functions are linked during run time.

Disadvantage:

- Executable file becomes very large, so loading time becomes more.
- If any library is replaced by a new version, all the programs must be re-compiled and re-linked to the new library version.

ii) Dynamic linking:

- Dynamic linking is a mechanism for linking a library into memory at runtime, retrieving variable and function addresses, executing the functions, and unloading the program from memory.
- It's frequently used to create software plugins. This method is used by Apache Web Server to load a dynamic shared object (*.dso) files at runtime.

Advantage:

- If any library is replaced by a new version, all the programs that reference the library will automatically use the new version without the need for recompilation.

Disadvantage:

- Program startup time is slower because during the execution, it checks all linked files if they are in main memory or not.

2) Loading:

- Initially executable code of every program is present in the secondary storage. But for the execution of the executable file, it must be present in the main memory.
- Bringing the code from secondary memory and storing it in the main memory is known as loading.
- It is done by the loader.
- Loader is a special program that takes the executable code present in the secondary memory, which is generated by the linker, and loads it into main memory. It can be done in two ways:

i) Static loading:

- Static loading refers to loading the entire executable file in the main memory before its execution starts.

Advantage:

- Program execution will be fast.

Disadvantage:

- Inefficient utilisation of main memory because even if the entire code need not be executed at once, loader still loads the whole code in main memory, thus occupying the main memory unnecessarily.
- Also, the size of the process which can be loaded into the main memory is limited by the size of the main memory. Thus, any process whose size is more than the size of the main memory cannot be executed.

ii) Dynamic loading:

- A routine in dynamic loading is not loaded until it is called.
- The main program is loaded into the memory and gets executed; other routines are kept on the disk in a relocatable format.
- When a routine needs to call another routine, the calling routine first checks whether the other routine has been loaded.

- If it is not, the relocatable linking loader is called to load the desired routine into the memory.
- This type of loading is useful when the large number of codes are needed to handle infrequently occurring cases, such as error routines.

Advantage:

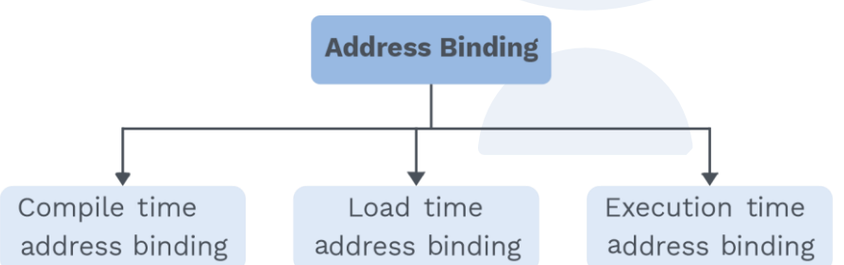
- Unused routine is never loaded into the main memory; thus main memory can be efficiently used.

Disadvantage:

- Program execution will be slower.

Address binding:

- Address Binding is the process of connecting application program instructions and data into physical memory locations.
- It refers to the process of converting one address space to another.
- The physical address relates to the location in the memory unit, but the logical address is generated by the CPU during execution.



1) **Compile time address binding:**

- If you know where the process will live in memory during compile time, an absolute address is generated, i.e. the physical address is given to the program's executable fire during compilation.
- Address binding is the responsibility of the compiler.
- It will be completed before the application is loaded into memory.
- To achieve compile-time address binding, the compiler must communicate with an OS memory management.

2) Load time address binding:

- If the location of the process is unknown at compile-time, a relocatable address will be produced. The relocatable address is converted to an absolute address by the loader.
- The loader generates an absolute address by adding the process's base address in the main memory to all logical addresses.
- It will be completed after the application has been loaded into memory.
- The OS memory manager, or loader, will handle this form of address binding.

Note:

If the process's base address changes during load time, address binding, the process must be reloaded.

3) Execution time/dynamic address binding:

- The CPU executes instructions of a process which are stored in memory.
- If a process can be relocated from one memory location to another during execution, this is employed.
- Even after the program has been loaded into memory, the address binding will be delayed. Until the program is finished, the application program will keep modifying the memory locations.
- The CPU does this type of address binding during program execution.
- It's compatible with dynamic absolute addresses.

Note:

Hardware support for address mappings, such as base and limit registers are required for dynamic address binding.

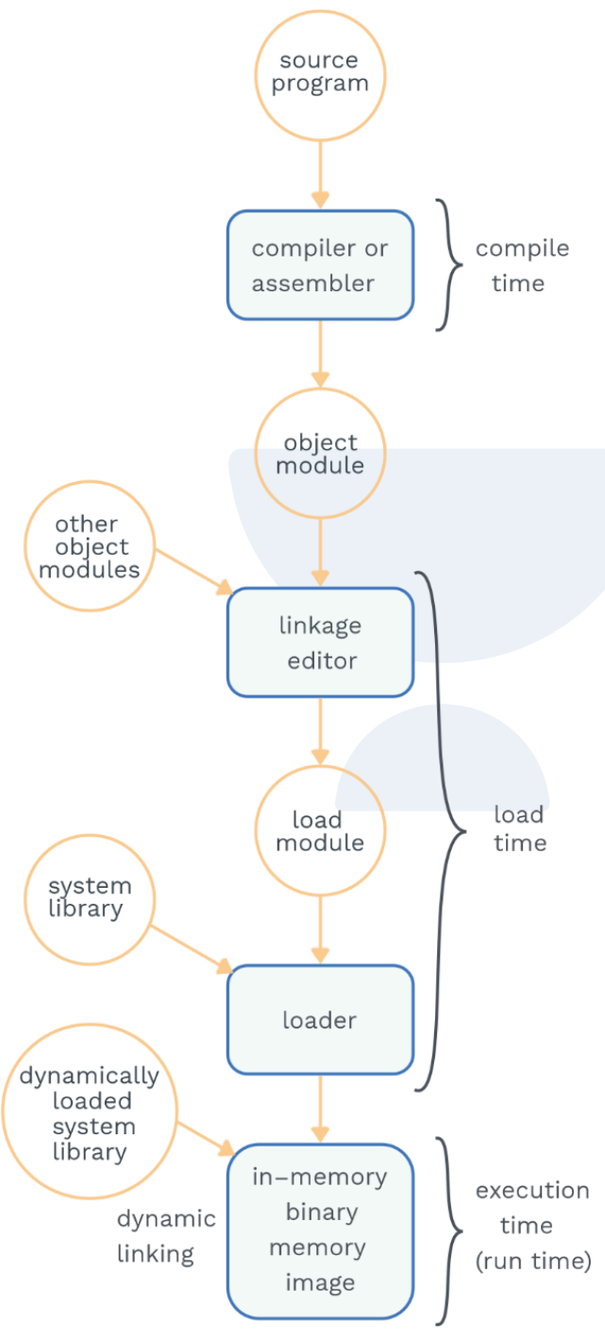


Fig. 5.5 Multistep Processing of a User Program

Swapping:

- A process must be in memory to be executed however, it can be temporarily moved to a backing store/secondary memory/nonvolatile storage, and then returned to main memory for further execution.
- Swapping allows the entire physical address space of all processes to surpass the system's actual physical (RAM) memory.
- Swapping is another technique for increasing the degree of multiprogramming in a system.

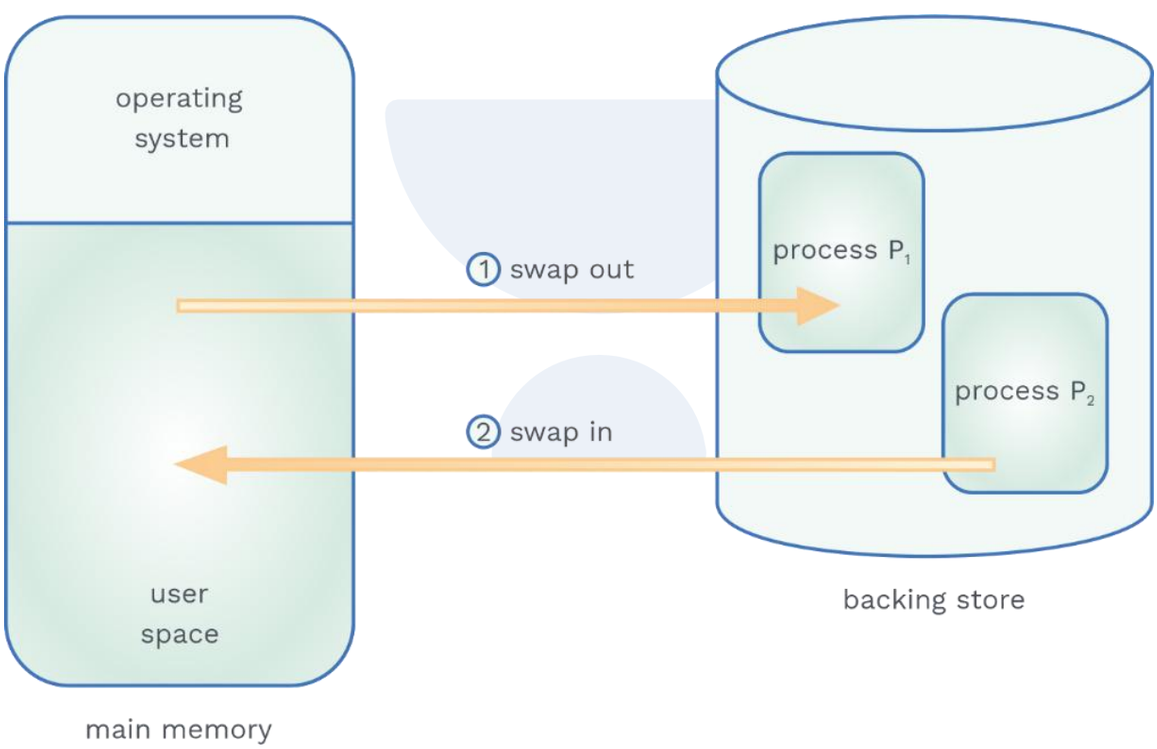
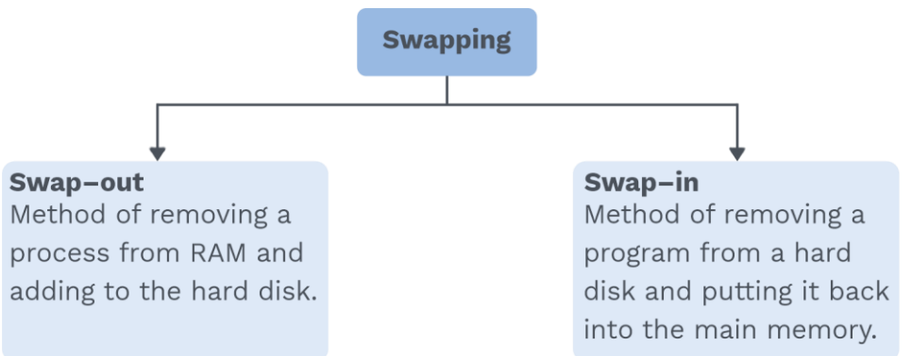


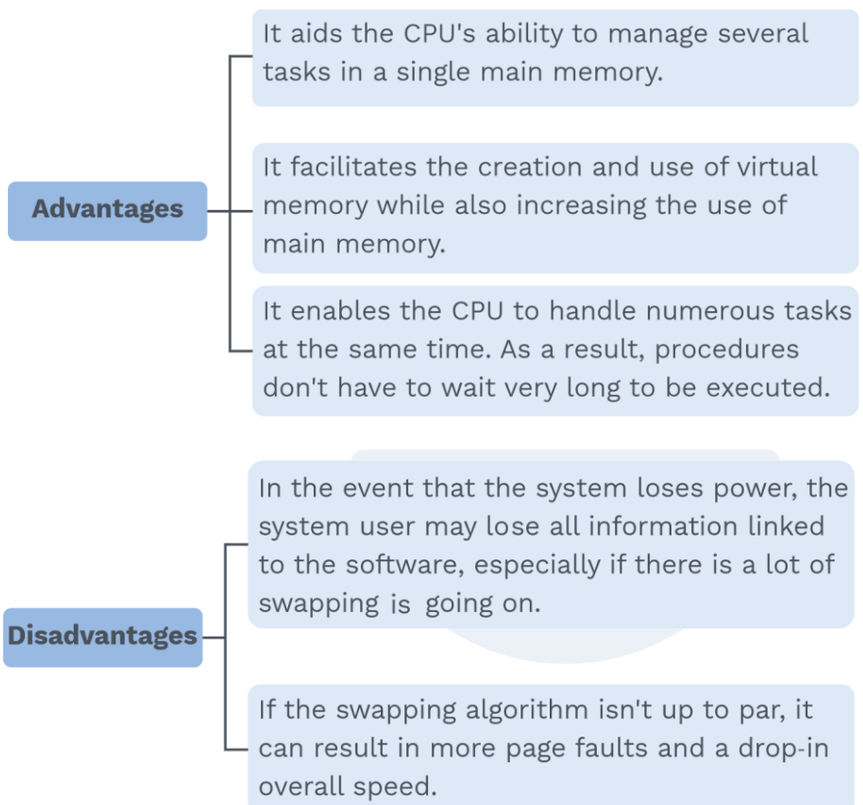
Fig. 5.6 Swapping of Two Processes Using a Disk as a Backing Store

Definition

Swapping is a memory management technique that allows any the process to be temporarily switched from main memory to secondary memory to free up main memory for other processes.

The concept of swapping has divided into two parts:

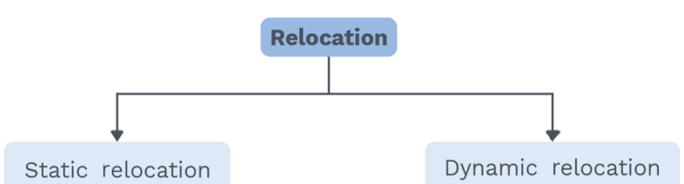




Objectives of memory management system:

1) Relocation:

- The ability to shift processes around in memory without having any impact on their execution.
- Memory is managed by the operating system rather than the programmer, and processes can be moved into the memory.
- Memory Management (MM) is responsible for translating logical addresses to physical addresses.



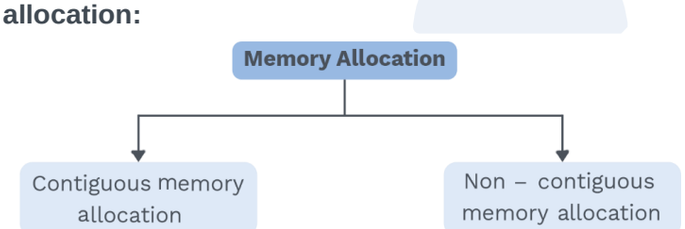
Static relocation:
Before or during the loading of the process into memory, the program must be relocated.
Relocator must be executed again if the program is not loaded into the same address space in memory.

Dynamic relocation:
In memory, processes can freely move about. At runtime, a virtual–to–physical address space mapping is performed.

2) Protection:
- To prevent data and instructions from being over–written.
- For data and instructions security purposes, OS is protected from user processes and on the same note, user processes are protected from other user processes.
- The reason behind this is many of the languages compute address of memory during runtime.

3) Sharing:
- Different processes may need to run the same operation or even access the same data at times.
- Different processes must access the same memory address when they signal or wait for the same semaphore.

Memory allocation:



Contiguous memory allocation:
The RAM is frequently partitioned into two parts: one for the operating the system, and the other for user processes.

Note:
The operating system can be installed in either a low-memory or a high-memory location. It is determined by the Interrupt Vector's location.

Definitions

It is basically a method in which a single contiguous part of memory is allocated to a process.

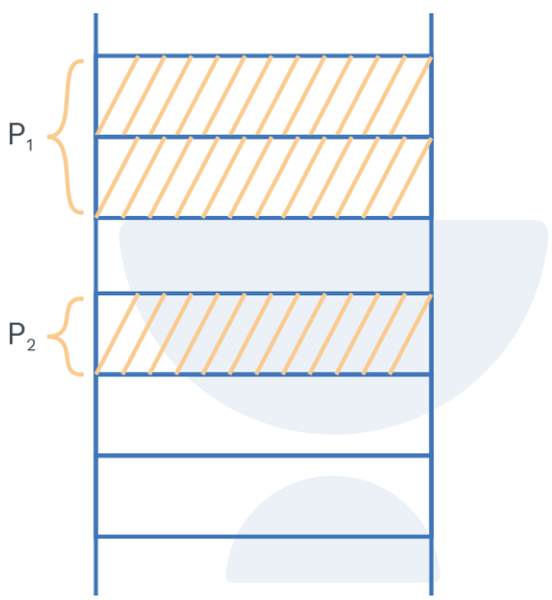


Fig. 5.7 Contiguous Memory Allocation

- Unused memory space is allocated to the same location in contiguous memory.
- Processes are faster in execution.
- Processes are easier for the OS to control.
- Address translation is minimum while executing a process.
- It suffers from both Internal and external fragmentation.

Grey Matter Alert!

Internal fragmentation:
When memory blocks associated with a process have unused space within the block, it is called internal fragmentation.

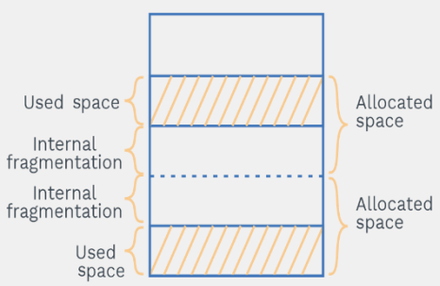


Fig. 5.8 Internal Fragmentation

External fragmentation:
When there is enough space within the memory to satisfy a method's memory request, but the process' memory request cannot be completed because the memory is spread in a non-contiguous manner, several approaches, such as compaction, are applied to solve external fragmentation.

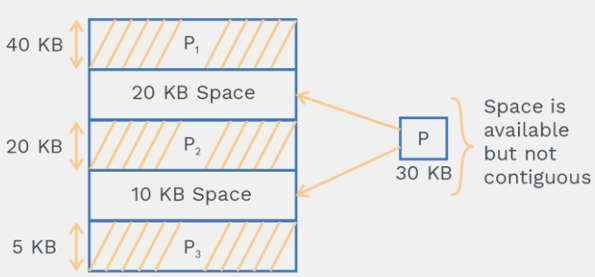
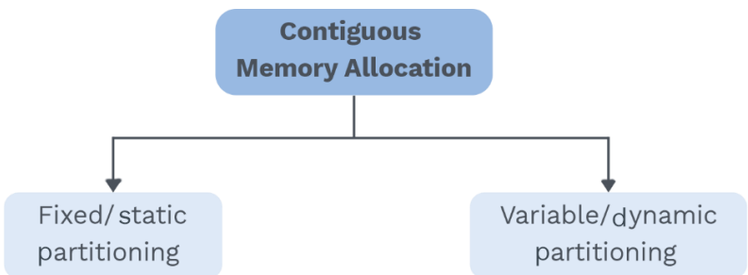


Fig. 5.9 External Fragmentation

In the above figure P1, P2 and P3 have been allocated in RAM, leaving holes of 20KB and 10KB which are not contiguous. When a process P of 30KB request for space in memory; it cannot be allocated because available space is not contiguous.



Fixed partitioning:

- This is the simplest technique used to store more than one process at a time in the main memory.
- In this partitioning, a number of non-overlapping partitions in physical memory is fixed, but the size of each partition may or may not be the same.
- In this, only one process is allowed per partition.

Note:

The operating system is always installed in the first partition, with the remaining partitions being utilised to store user processes.

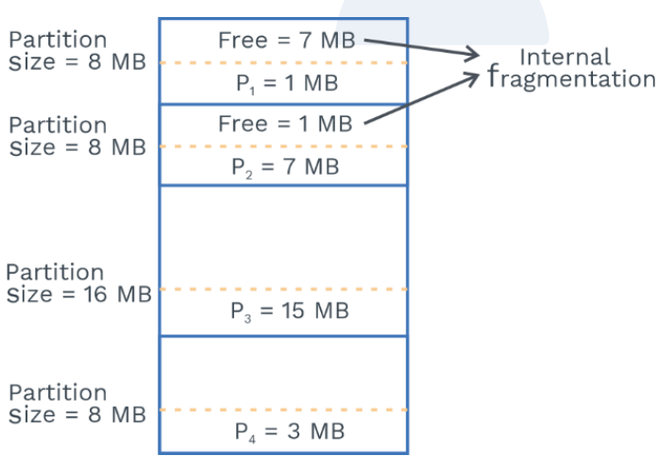


Fig. 5.10 Fixed Size Partition

Advantages:

1) Implementation is easy
2) Fixed partitioning processing necessitates less CPU resources, and minimal OS overhead.
3) Fixed partitioning does not suffer from external fragmentation.

Disadvantages:

1) Internal Fragmentation is suffered in the fixed partition.
2) Because partitions are made before execution, it limits the degree of multiprogramming. For example, if there are 'x' partitions in RAM and 'y' is the number of processes, then the 'y' = 'x' requirement must be met. In fixed partitioning, processes that are larger than partitions (in terms of quantity) are invalid.
3) It restricts the size of the process since it cannot support processes larger than the partition size.
4) Spanning a process into two partitions is not possible.

Variable partitioning:

Initially, the whole user space of memory is free, and partitions are created as needed after the arrival of processes.

- The total number of partitions is not fixed and is determined by the number of incoming processes and the amount of RAM available.
- The operating system takes up the first partition, and the remaining space is dynamically partitioned.

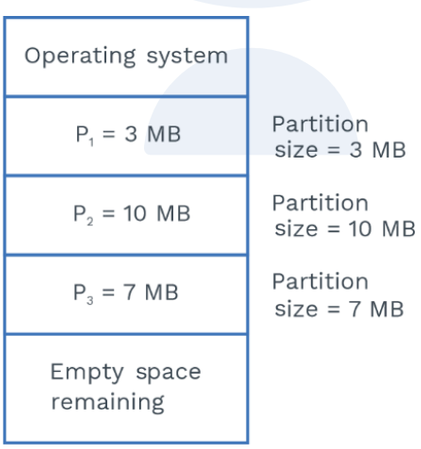


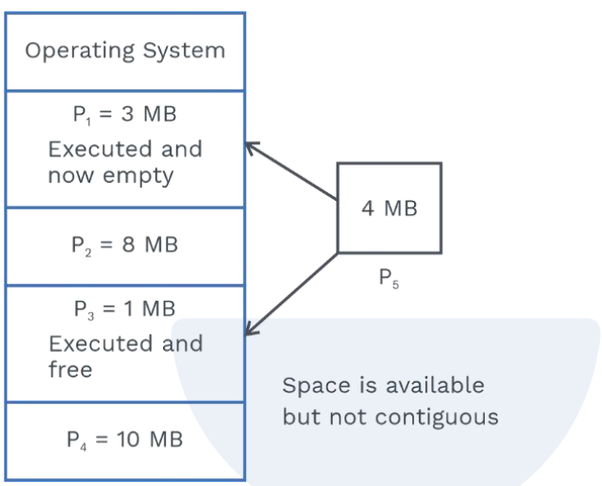
Fig. 5.11 Variable Partitioning

Advantages:

1) There is no internal fragmentation because the main memory space is allocated based on the needs of the process. There will be no unused partition space remaining.
2) There are no restrictions on the amount of multiprogramming that can be done.
3) There is no limit on the size of the process requesting RAM.

Disadvantages:

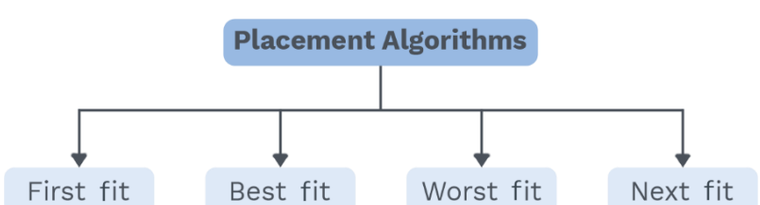
1) Implementation is complex as the allocation of memory is at runtime.
2) It suffers from external fragmentation.



In the above figure P_1, P_2, P_3 and P_4 has been allocated in RAM. Process P_1 and Process P_3 have been executed and freed up space of 3MB and 1MB, which are not contiguous. When process P_5 of 4MB requests for space in memory, but it cannot be allocated because available space is not contiguous.

Allocation policies:

When there are multiple partitions available to accommodate a process, a partition must be chosen using one of the partition allocation policies.



1) First fit:

It assigns the first available free partition (hole) that is large enough. It begins scanning the memory from the beginning and selects the first large enough block available. Among the several allocation policies, it is the quickest.

2) Best fit:

- It allocates the smallest sufficient free block that can hold the process.
- It gives the worst performance overall in terms of allocation time, as every time the smallest hole is found by searching in the whole memory.
- It will give least internal fragmentation.
- Compaction is done here more often.
- Best fit is an efficient allocation policy for fixed partitioning.

3) Worst fit:

- It allocates the largest block among all available blocks that is big enough.
- It is the opposite of best fit allocation.
- Worst fit is efficient allocation policy for variable partitioning as it will create holes of larger size so that other small processes can be placed in those memory holes. .

4) Next fit:

Next fit will start searching from last allocated partition; rest is the same as that of the first fit.

Rack Your Brain

Is Best–fit really the best allocation technique among others in fixed partitioning?

Grey Matter Alert!

Compaction:

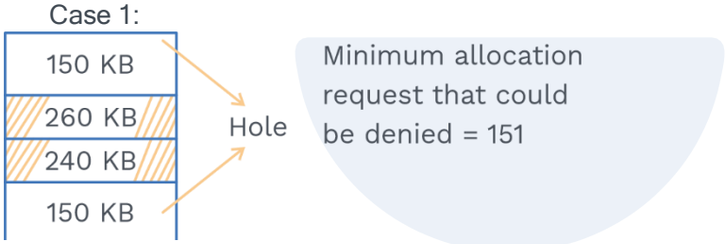
Compaction or shuffling memory contents is a solution for external fragmentation; all free memory is gathered into one single block. Moving should be dynamic in order to make compaction possible. Using a paging or segmentation technique, external fragmentation can be resolved.

SOLVED EXAMPLES

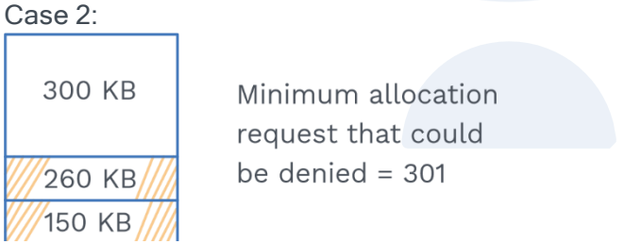
Q2 **A memory of size 800 KB is managed using variable partitioning with no compaction method. Memory currently has the following partitions:**
Partition I – 260 KB
Partition II – 240 KB
What is the smallest allocation request size (in KB) that could be denied?

Sol: **Range: 101–101**

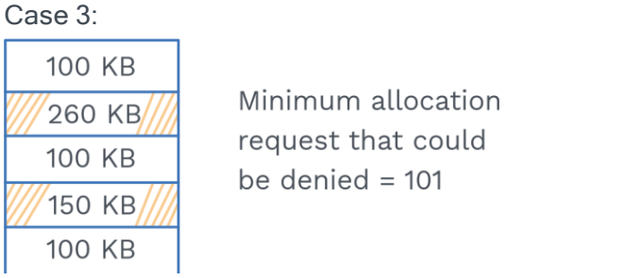
Case 1:



Case 2:



Case 3:



Non–contiguous allocation:
In contrast to contiguous allocation, it is a mechanism that allocates memory space in different locations to the process according to its needs.
All of the available vacant space is dispersed across.
This memory allocation strategy helps to prevent memory waste, which eventually leads to internal and external fragmentation.