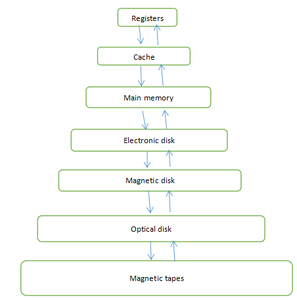
The term Memory can be defined as a collection of data in a specific format. It is used to store instructions and processed data. The memory comprises a large array or group of words or bytes, each with its own location.

The primary motive of a computer system is to execute programs. These programs, along with the information they access, should be in the main memory during execution. The CPU fetches instructions from memory according to the value of the program counter.

To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.



**What is Main Memory:**

The main memory is central to the operation of a modern computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs and information are kept when the processor is effectively utilizing them.  Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.  Main memory is also known as RAM(Random Access Memory). This memory is a volatile memory.RAM lost its data when a power interruption occurs.

### What is Memory Management :

In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

### Why Memory Management is required:

* Allocate and de-allocate memory before and after process execution.
* To keep track of used memory space by processes.
* To minimize fragmentation issues.
* To proper utilization of main memory.
* To maintain data integrity while executing of process.

### Logical and Physical Address Space:

**Logical Address space:**An address generated by the CPU is known as “Logical Address”. It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.

**Physical Address space:**An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”. A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

**Static and Dynamic Loading:**

To load a process into the main memory is done by a loader. There are two different types of loading :

* **Static loading**:- loading the entire program into a fixed address. It requires more memory space.
* **Dynamic loading**:- The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

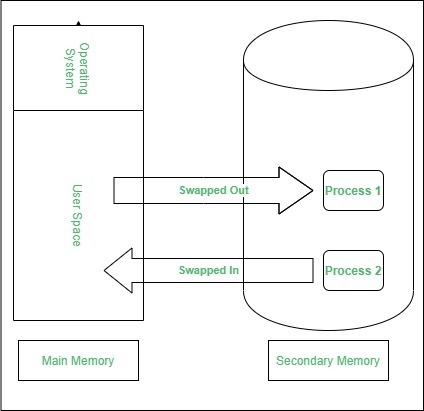
**Static and Dynamic linking:**

To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

* **Static linking:**In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
* **Dynamic linking:** The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

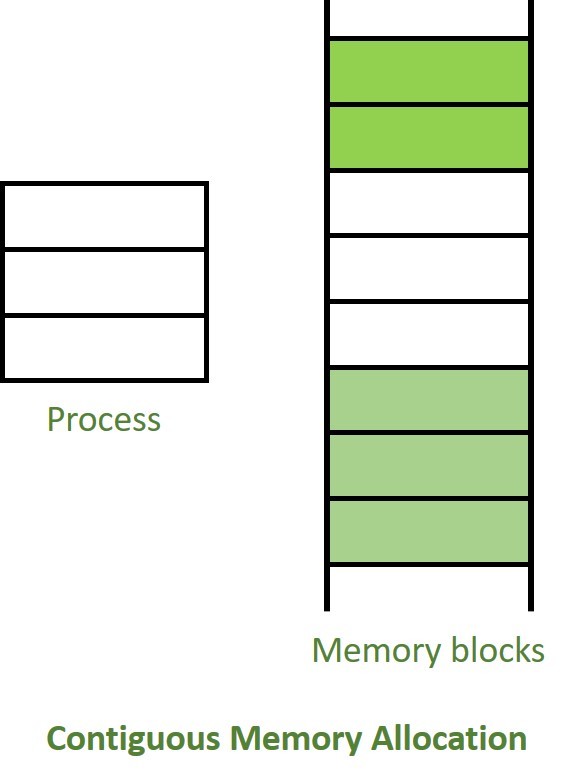
### Swapping :

When a process is executed it must have resided in memory. Swapping is a process of swap a process temporarily into a secondary memory from the main memory, which is fast as compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time directly proportional to the amount of memory swapped. Swapping is also known as roll-out, roll in, because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.



### Contiguous Memory Allocation :

The main memory should oblige both the operating system and the different client processes.  Therefore, the allocation of memory becomes an important task in the operating system.  The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.



**Memory allocation:**

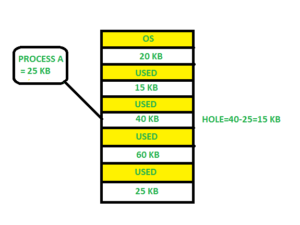
To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.

**Multiple partition allocation**: In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

**Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as “Hole”. When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement fulfills then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:

**First fit:-**

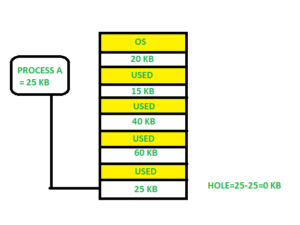
In the first fit, the first available free hole fulfills the requirement of the process allocated.



Here, in this diagram 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

**Best fit:-**

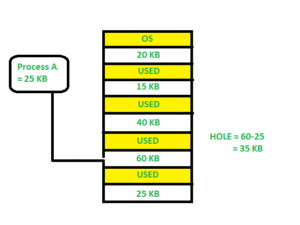
In the best fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB).

In this method memory utilization is maximum as compared to other memory allocation techniques.

**Worst fit:-**In the worst fit, allocate the largest available hole to process. This method produces the largest leftover hole.



Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB. Inefficient memory utilization is a major issue in the worst fit.

**Fragmentation:**

A Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process.  To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problem. In operating system two types of fragmentation:

**Internal fragmentation:**

Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is leftover and creates an internal fragmentation problem.

 Example: Suppose there is a fixed partitioning is used for memory allocation and the different size of block 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demand for the block of memory. It gets a memory block of 3MB but 1MB block memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

**External fragmentation:**

In external fragmentation, we have a free memory block, but we can not assign it to process because blocks are not contiguous.

Example: Suppose (consider above example) three process p1, p2, p3 comes with size 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating process p1 process and p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous.  This is called external fragmentation.

Both the first fit and best-fit systems for memory allocation affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

Another possible solution to the external fragmentation is to allow the logical address space of the processes to be noncontiguous, thus permit a process to be allocated physical memory where ever the latter is available.

**Paging:**

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous.

* Logical Address or Virtual Address (represented in bits): An address generated by the CPU
* Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program
* Physical Address (represented in bits): An address actually available on a memory unit
* Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

**Example:**

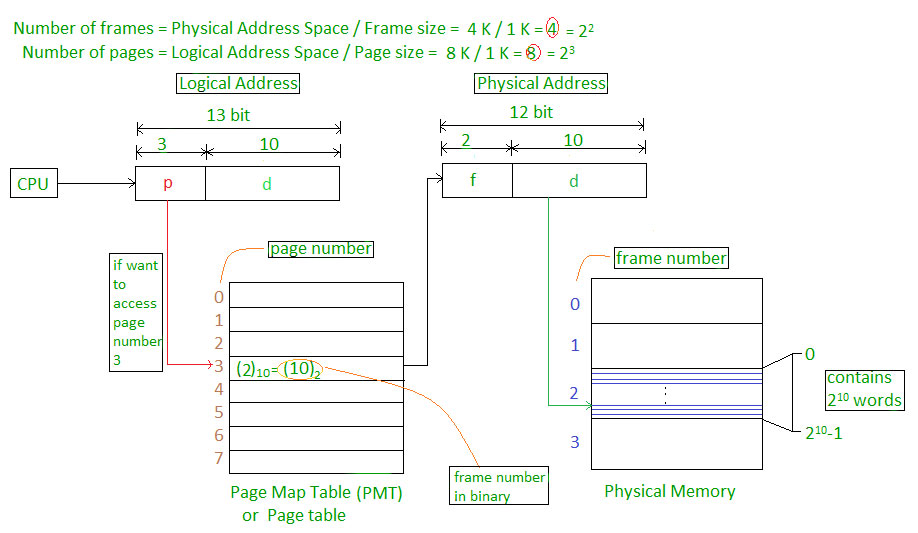
* If Logical Address = 31 bits, then Logical Address Space = 231 words = 2 G words (1 G = 230)
* If Logical Address Space = 128 M words = 27 \* 220 words, then Logical Address = log2 227 = 27 bits
* If Physical Address = 22 bits, then Physical Address Space = 222 words = 4 M words (1 M = 220)
* If Physical Address Space = 16 M words = 24 \* 220 words, then Physical Address = log2 224 = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as the paging technique.

* The Physical Address Space is conceptually divided into several fixed-size blocks, called **frames**.
* The Logical Address Space is also split into fixed-size blocks, called **pages**.
* Page Size = Frame Size

Let us consider an example:

* Physical Address = 12 bits, then Physical Address Space = 4 K words
* Logical Address = 13 bits, then Logical Address Space = 8 K words
* Page size = frame size = 1 K words (assumption)



The address generated by the CPU is divided into

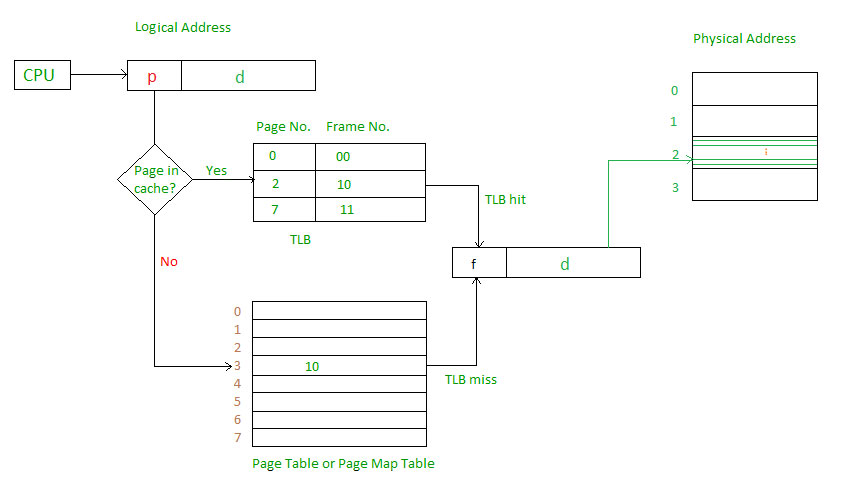
* **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
* **Page offset(d):** Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

* **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number frame
* **Frame offset(d):** Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of the page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.

* The TLB is an associative, high-speed memory.
* Each entry in TLB consists of two parts: a tag and a value.
* When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



Main memory access time = m

If page table are kept in main memory,

Effective access time = m(for page table) + m(for particular page in page table)

