Requirements of Memory Management System

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed.

Memory management is meant to satisfy the following requirements:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.
2. When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.

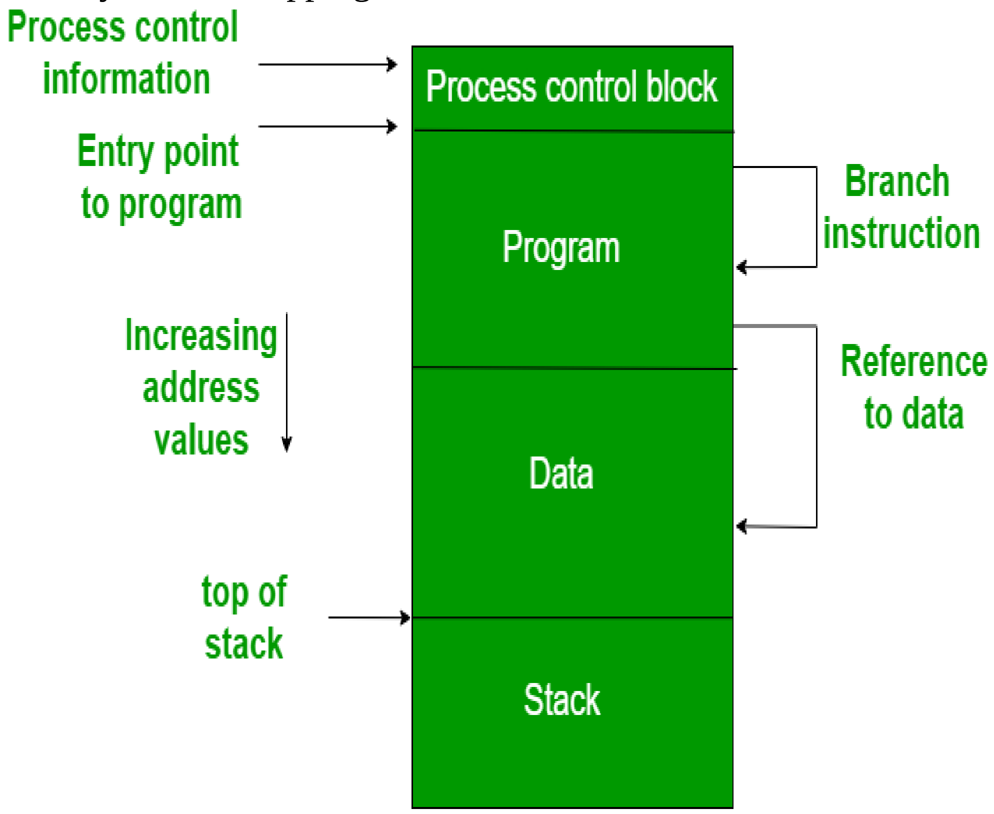


Figure 7.3.17.3.1: A process occupying a continuous region of main memory.

The figure depicts a process image. Every process looks like this in memory. Each process contains: 1) process control blocks; 2) a program entry point - this is the instruction where the program starts execution; 3) a program section; 4) a data section; and 5) a stack. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

3. **Protection –** There is always a danger when we have multiple programs executing at the same time - one program may write to the address space of another program. So every process must be protected against unwanted interference if one process tries to write into the memory space of another process - whether accidental or incidental. The operating system makes a trade-off between relocation and protection requirement: in order to satisfy the relocation requirement the difficulty of satisfying the protection requirement increases in difficulty.

4. It is impossible to predict the location of a program in main memory, which is why it is impossible to determine the absolute address at compile time and thereby attempt to assure protection. Most programming languages provide for dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system does not necessarily control a process when it occupies the processor. Thus it is not possible to check the validity of memory references.

5. **Sharing –** A protection mechanism must allow several processes to access the same portion of main memory. This must allow for each processes the ability to access the same copy of the program rather than have their own separate copy.

   This concept has an advantage. For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

6. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To

effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

- Modules are written and compiled independently and all the references from one module to another module are resolved by the system at run time.
- Different modules are provided with different degrees of protection.
- There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.

7. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

- The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
- In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

Memory Management in Operating System

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

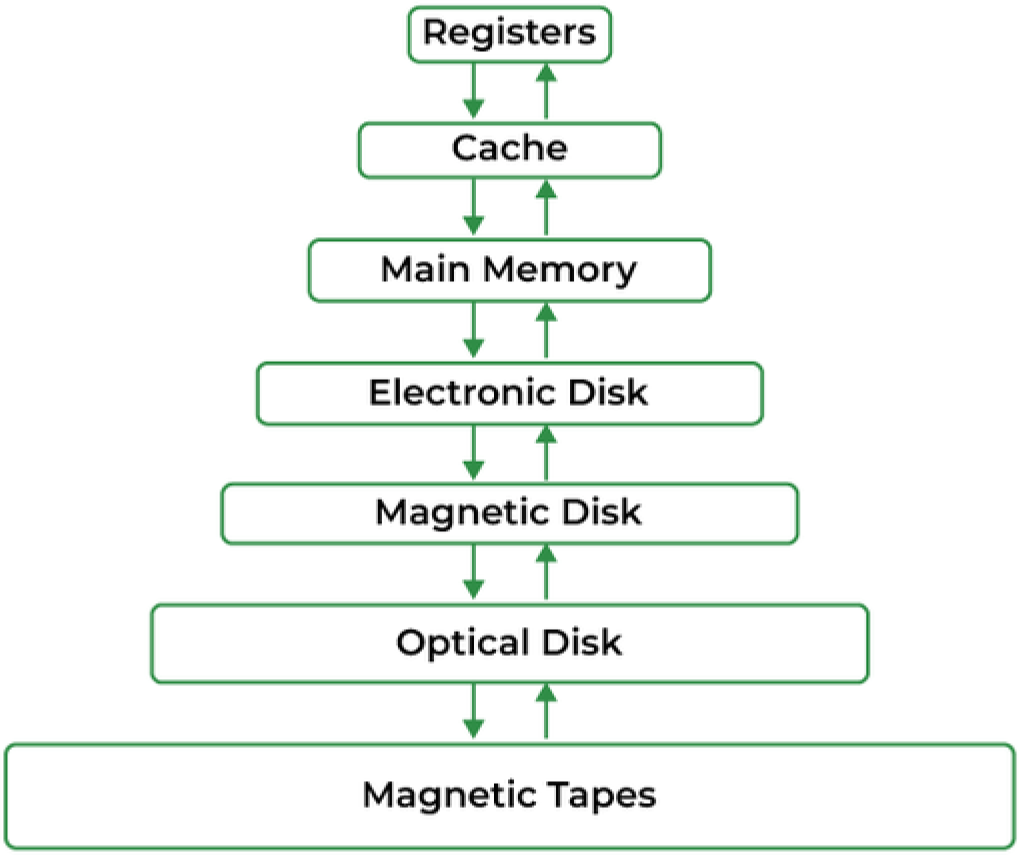
To achieve a degree of multiprogramming and proper utilization of memory, memory management is important. Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

Here, we will cover the following memory management topics:

- *What is Main Memory?*
- *What is Memory Management?*
- *Why Memory Management is Required?*
- *Logical Address Space and Physical Address Space*
- *Static and Dynamic Loading*
- *Static and Dynamic Linking*
- *Swapping*
- *Contiguous Memory Allocation*
    - *Memory Allocation*
        - *First Fit*
        - *Best Fit*
        - *Worst Fit*
    - *Fragmentation*
        - *Internal Fragmentation*
        - *External Fragmentation*
    - *Paging*

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 volatile. RAM loses its data when a power interruption occurs.



Main Memory

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.

Now we are discussing the concept of Logical Address Space and Physical Address Space

Logical and Physical Address Space

- **Logical Address Space:** An address generated by the CPU is known as a "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

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

- **Static Loading:** Static Loading is basically 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 the unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

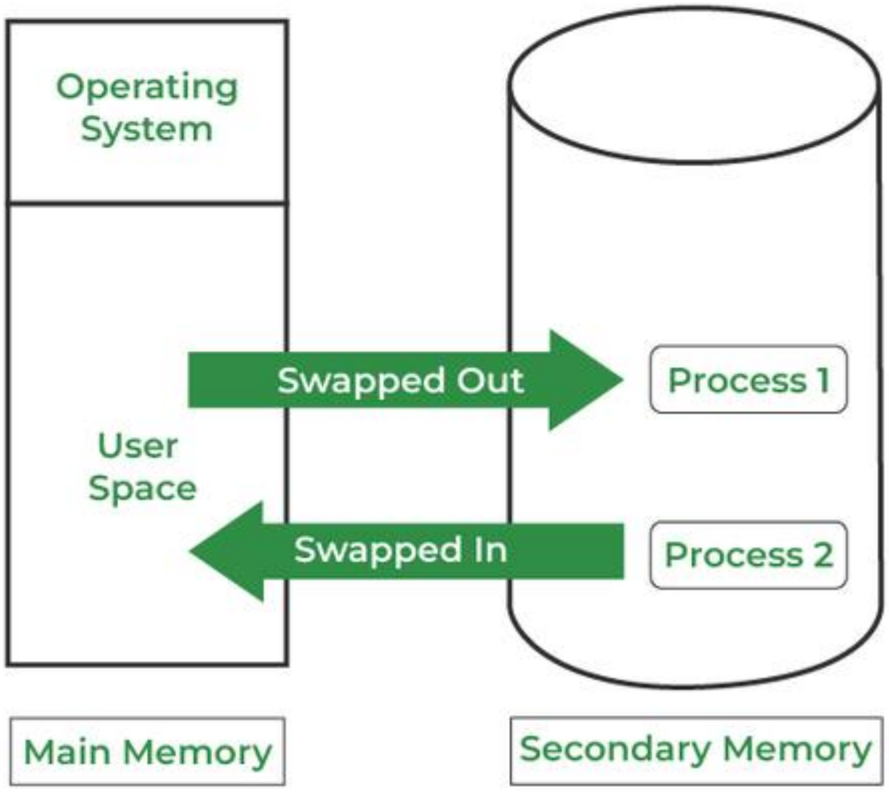
Static and Dynamic Linking

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 swapping a process temporarily into a secondary memory from the main memory, which is fast 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 is directly proportional to the amount of memory swapped. Swapping is also known as roll-out, or roll 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.



swapping in memory management

Memory Management with Monoprogramming (Without Swapping)
This is the simplest memory management approach the memory is divided into two sections:

- One part of the operating system
- The second part of the user program

Fence Register	
operating system	user program

- In this approach, the operating system keeps track of the first and last location available for the allocation of the user program
- The operating system is loaded either at the bottom or at top
- Interrupt vectors are often loaded in low memory therefore, it makes sense to load the operating system in low memory
- Sharing of data and code does not make much sense in a single process environment
- The Operating system can be protected from user programs with the help of a fence register.

Advantages of Memory Management

- It is a simple management approach

Disadvantages of Memory Management

- It does not support multiprogramming
- Memory is wasted

Multiprogramming with Fixed Partitions (Without Swapping)

- A memory partition scheme with a fixed number of partitions was introduced to support multiprogramming. this scheme is based on contiguous allocation
- Each partition is a block of contiguous memory
- Memory is partitioned into a fixed number of partitions.
- Each partition is of fixed size

Example: As shown in fig. memory is partitioned into 5 regions the region is reserved for updating the system the remaining four partitions are for the user program.

Fixed Size Partitioning

Operating System
p1
p2
p3
p4

Partition Table

Once partitions are defined operating system keeps track of the status of memory partitions it is done through a data structure called a partition table.

Sample Partition Table

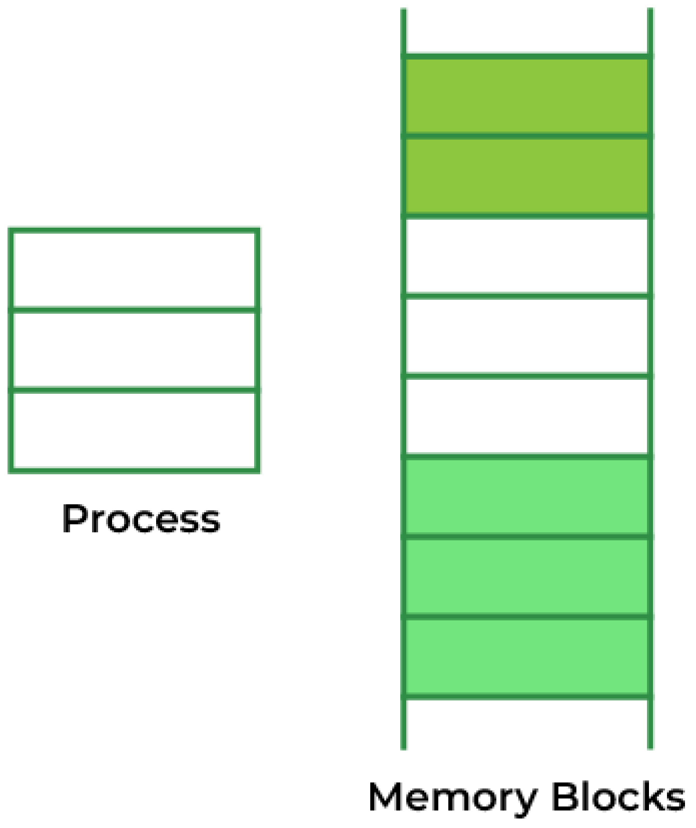
Starting Address of Partition	**Size of Partition**	**Status**
0k	200k	allocated
200k	100k	free
300k	150k	free
450k	250k	allocated

Logical vs Physical Address

An address generated by the CPU is commonly referred to as a logical address. the address seen by the memory unit is known as the physical address. The logical address can be mapped to a physical address by hardware with the help of a base register this is known as dynamic relocation of memory references.

Contiguous Memory Allocation

The main memory should accommodate 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.



Contiguous Memory Allocation

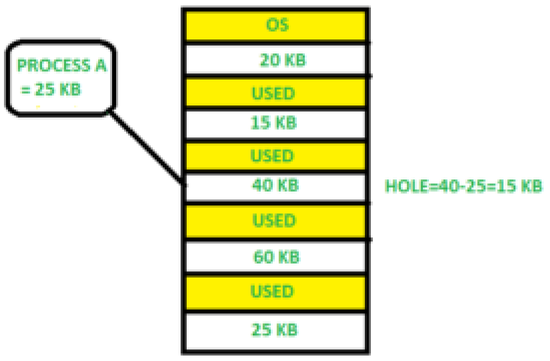
Memory Allocation

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 a "Hole". When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement is fulfilled 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 fulfil the requirement of the process allocated.

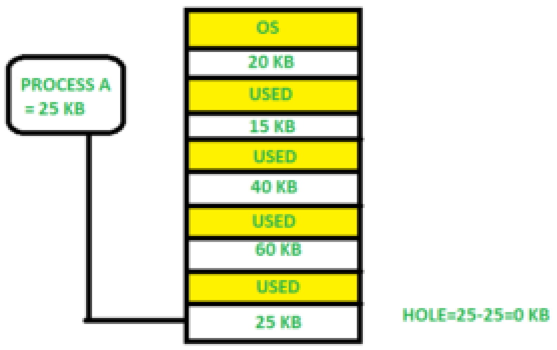


First Fit

Here, in this diagram, a 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.

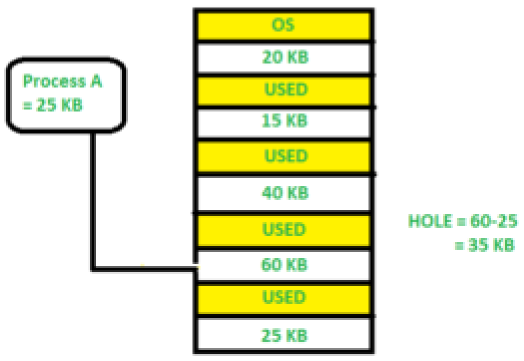


Best Fit

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.



Worst Fit

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

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 problems. In the operating systems two types of fragmentation:

1. **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 left over and creating an internal fragmentation problem.**Example:** Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.
2. **External fragmentation:** In External Fragmentation, we have a free memory block, but we can not assign it to a process because blocks are not contiguous. **Example:** Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the 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 are 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 permitting a process to be allocated physical memory wherever the latter is available.

Paging

Paging is a memory management scheme that eliminates the need for a 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 = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits

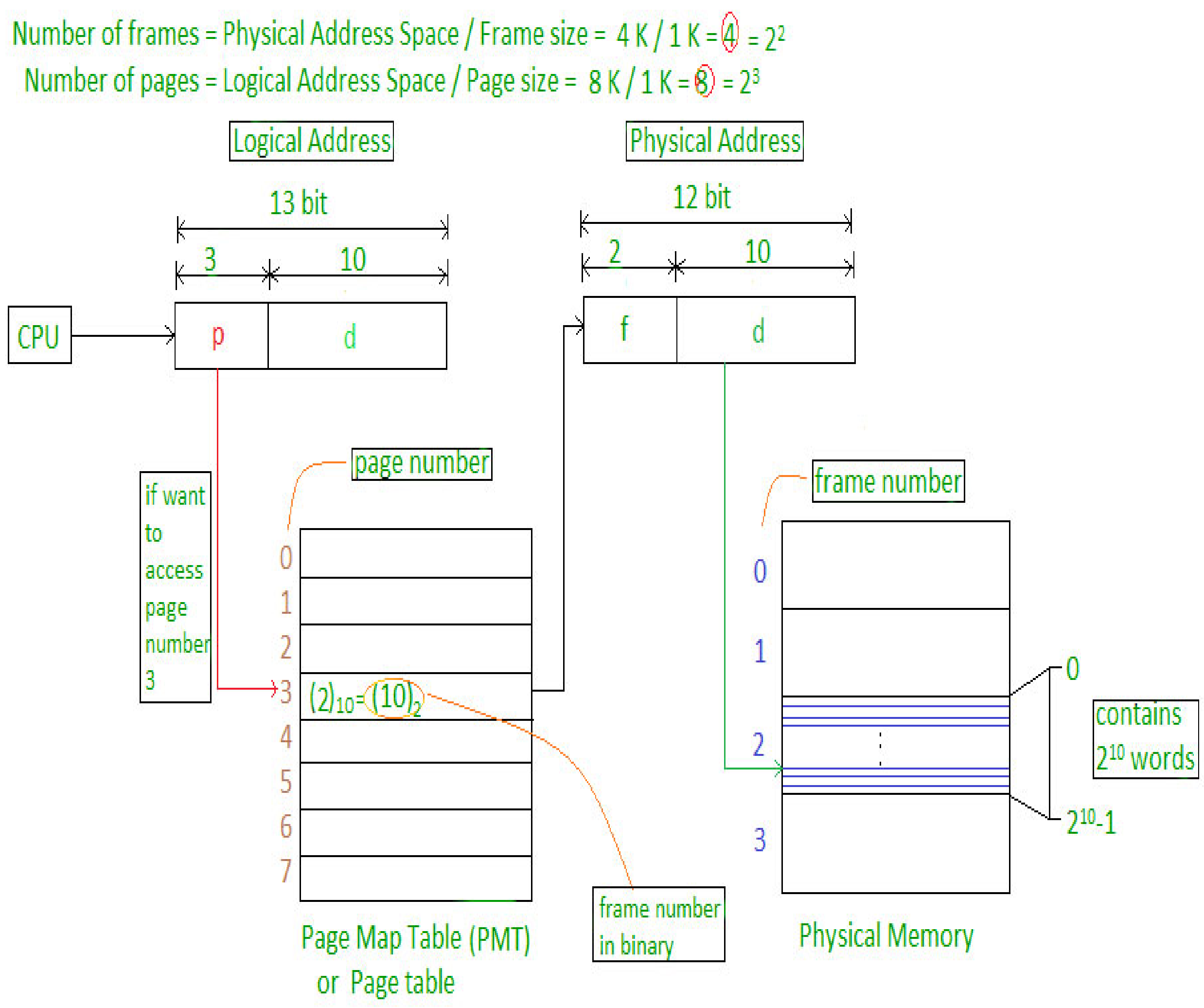
- If Physical Address = 22 bits, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^{4} * 2^{20}$ words, then Physical Address = $\log_2 2^{24}$ = 24 bits

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)



Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2
Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3
Logical Address
13 bit
3
10
Physical Address
12 bit
2
10
CPU
p
d
f
d
if want to access page number 3
page number
0
1
2
3
4
5
6
7
$(2)_{10} = (10)_2$
frame number
0
1
2
3
0
$2^{10}-1$
contains 2^{10} words
frame number in binary
Page Map Table (PMT) or Page table
Physical Memory

Paging

The address generated by the CPU is divided into:

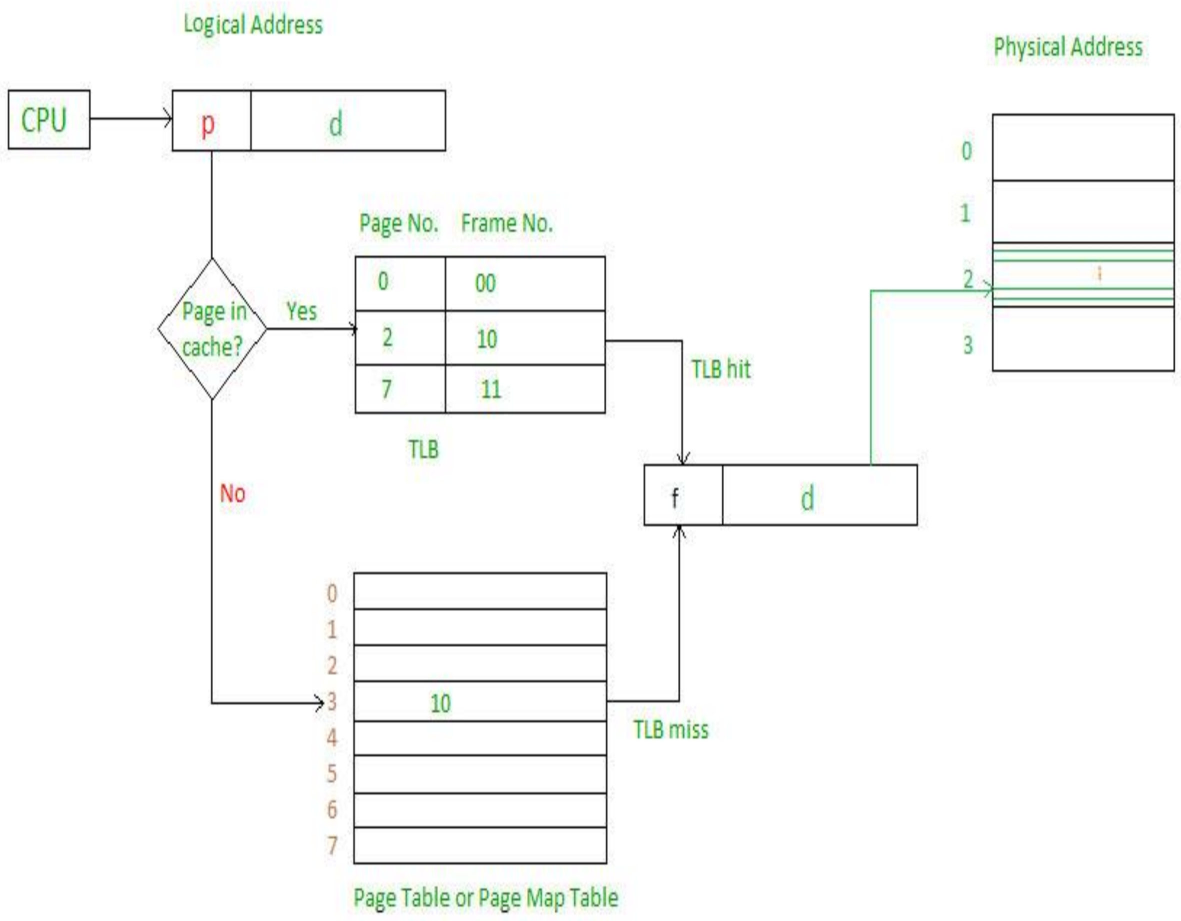
- **Page Number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page Offset(d):** Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

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.

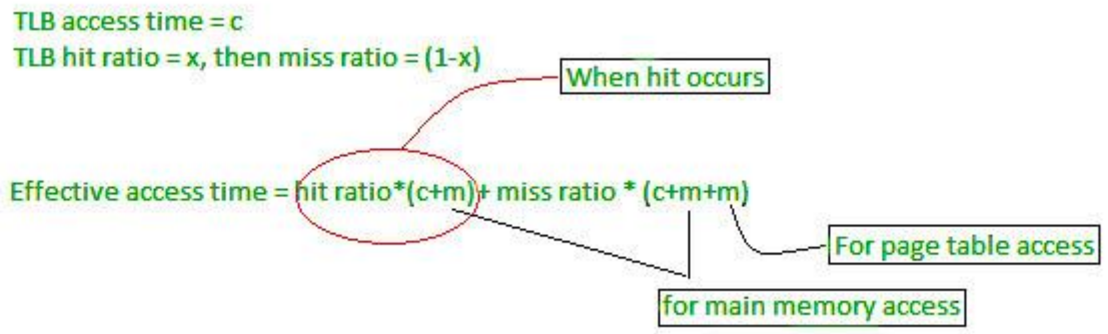
The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special,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



Page Map Table

```
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)
```


TLB access time = c
TLB hit ratio = x, then miss ratio = (1-x)
When hit occurs
Effective access time = hit ratio*(c+m) + miss ratio * (c+m+m)
For page table access
for main memory access

TLB Hit and Miss