5.8 Free Space management techniques – Bitmap, Linked List.

The system keeps tracks of the free disk blocks for allocating space to files when they are created. Also, to reuse the space released from deleting the files, free space management becomes crucial. The system maintains a free space list which keeps track of the disk blocks that are not allocated to some file or directory. The free space list can be implemented mainly as:

1. **Bitmap or Bit vector –**
   A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: *0 indicates that the block is allocated* and 1 indicates a free block.
   The given instance of disk blocks on the disk in *Figure 1* (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

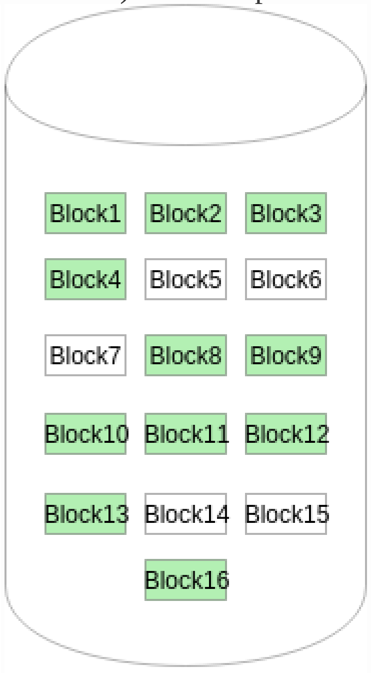


Figure - 1

Advantages –
- Simple to understand.
- Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.

The block number can be calculated as:
*(number of bits per word) *(number of 0-values words) + offset of bit first bit 1 in the non-zero word .*

For the *Figure-1*, we scan the bitmap sequentially for the first non-zero word.
The first group of 8 bits (00001110) constitute a non-zero word since all bits are not 0. After the non-0 word is found, we look for the first 1 bit. This is the 5th bit of the non-zero word. So, offset = 5.
Therefore, the first free block number = 8*0+5 = 5.

2. **Linked List –**
   In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

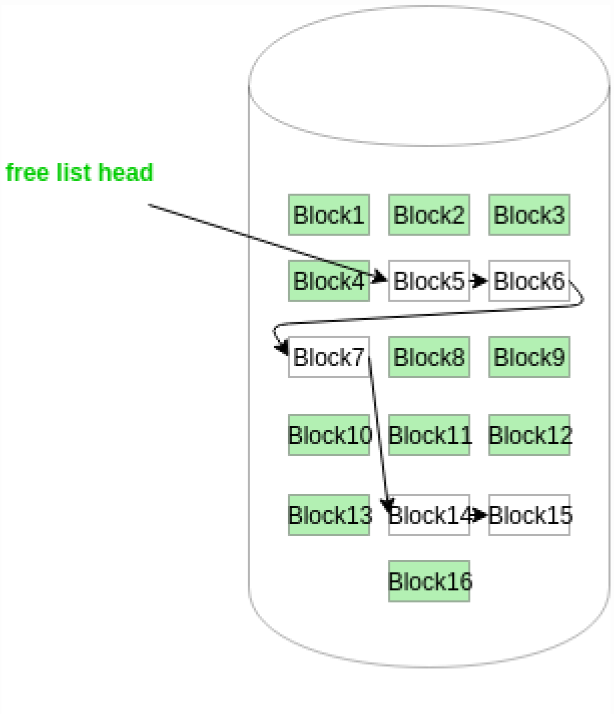


Figure - 2

In *Figure-2*, the free space list head points to Block 5 which points to Block 6, the next free block and so on. The last free block would contain a null pointer indicating the end of free list.
A drawback of this method is the I/O required for free space list traversal.

3. **Grouping** –
   This approach stores the address of the free blocks in the first free block. The first free block stores the address of some, say n free blocks. Out of these n blocks, the first n-1 blocks are actually free and the last block contains the address of next free n blocks. An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily.

4. **Counting –**
   This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.
   Every entry in the list would contain:
   1. Address of first free disk block
   2. A number n

For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

Variable (or dynamic) Partitioning in Operating System

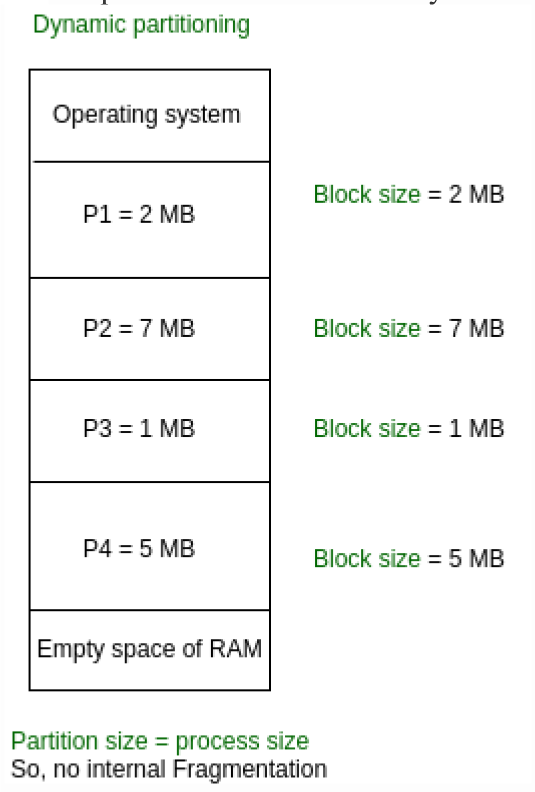
In operating systems, Memory Management is the function responsible for allocating and managing computer's main memory. Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.

There are two Memory Management Techniques: **Contiguous**, and **Non-Contiguous**. In Contiguous Technique, executing process must be loaded entirely in main-memory. Contiguous Technique can be divided into:

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

Variable Partitioning –
It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

1. Initially RAM is empty and partitions are made during the run-time according to process's need instead of partitioning during system configure.
2. The size of partition will be equal to incoming process.
3. The partition size varies according to the need of the process so that the internal fragmentation can be avoided
4. to ensure efficient utilisation of RAM.
5. Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.



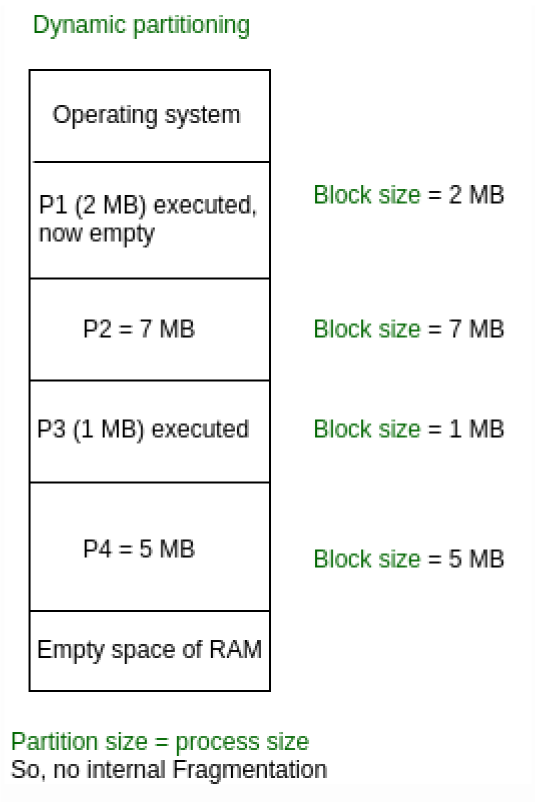
There are some advantages and disadvantages of variable partitioning over fixed partitioning as given below.

Advantages of Variable Partitioning –

1. **No Internal Fragmentation:**
   In variable Partitioning, space in main memory is allocated strictly according to the need of process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.
2. **No restriction on Degree of Multiprogramming:**
   More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is not empty.
3. **No Limitation on the size of the process:**
   In Fixed partitioning, the process with the size greater than the size of the largest partition could not be loaded and process can not be divided as it is invalid in contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Variable Partitioning –

1. **Difficult Implementation:**
   Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.
2. **External Fragmentation:**
   There will be external fragmentation inspite of absence of internal fragmentation.
   For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e., 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.



Now P5 of size 3 MB cannot be accommodated in spite of required available space because in contiguous no spanning is allowed.

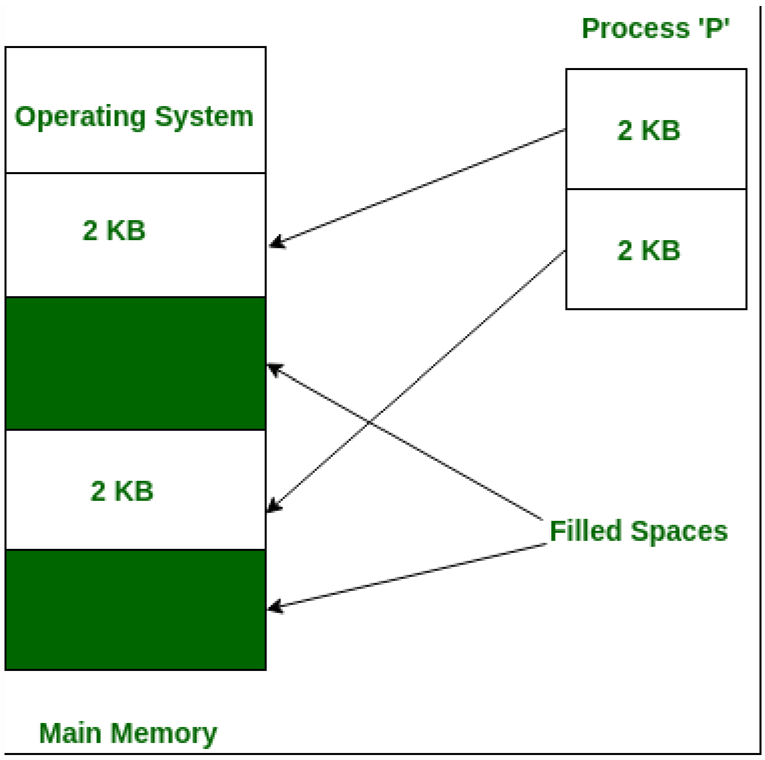
Non-Contiguous Allocation in Operating System

Prerequisite – Variable Partitioning, Fixed Partitioning

Paging and Segmentation are the two ways which allow a process's physical address space to be non-contiguous. It has **advantage** of reducing memory wastage but it increases the overheads due to address translation. It slows the execution of the memory because time is consumed in address translation.
In non-contiguous allocation, Operating system needs to maintain the table which is called **Page Table** for each process which contains the base address of each block which is acquired by the process in memory space. In non-contiguous memory allocation, different parts of a process is allocated different places in Main Memory. Spanning is allowed which is not possible in other techniques like Dynamic or Static Contiguous memory allocation. That's why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.

Working:
Here a process can be spanned across different spaces in main memory in non-consecutive manner. Suppose process P of size 4KB. Consider main memory have two empty slots each of size 2KB. Hence total free space is, 2*2= 4 KB. In contiguous memory allocation, process P cannot be accommodated as spanning is not allowed.

In contiguous allocation, space in memory should be allocated to whole process. If not, then that space remains unallocated. But in Non-Contiguous allocation, process can be divided into different parts and hence filling the space in main memory. In this example, process P can be divided into two parts of equal size – 2KB. Hence one part of process P can be allocated to first 2KB space of main memory and other part of processP can be allocated to second 2KB space of main memory. Below diagram will explain in better way:



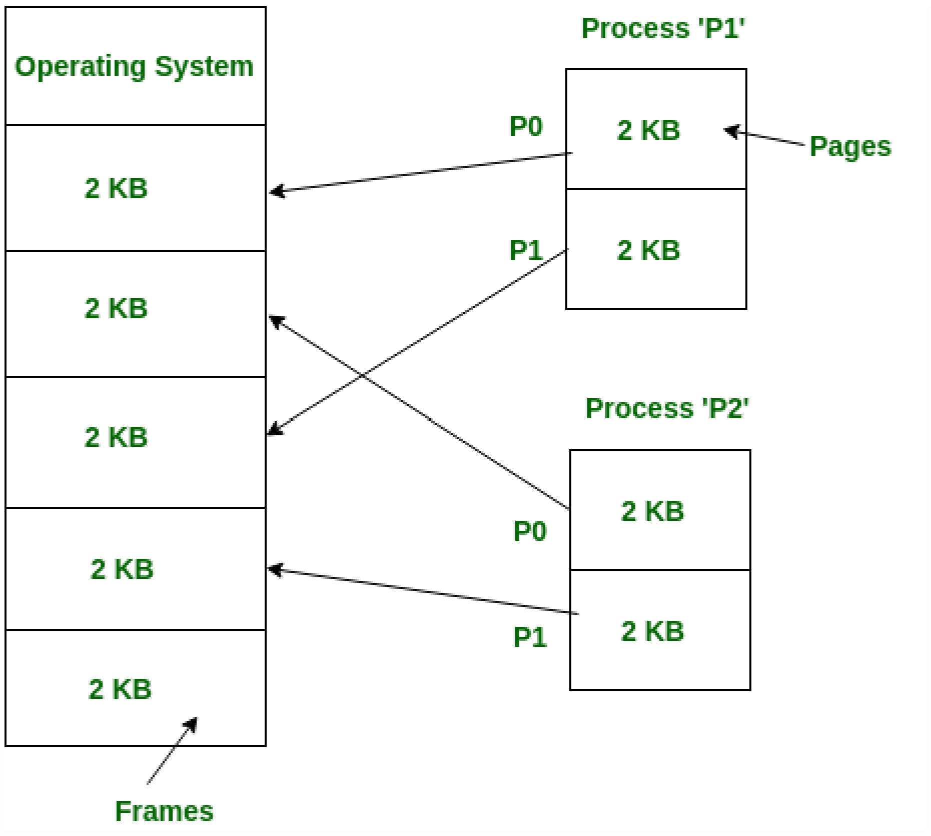
But, in what manner we divide a process to allocate them into main memory is very important to understand. Process is divided after analysing the number of empty spaces and their size in main memory. Then only we divide our process. It is very time-consuming process. Their number as well as their sizes changing every time due to execution of already present processes in main memory.

In order to avoid this time-consuming process, we divide our process in secondary memory in advance before reaching the memory for its execution. Every process is divided into various parts of equal size called Pages. We also divide our main memory into different parts of equal size called Frames. It is important to understand that:

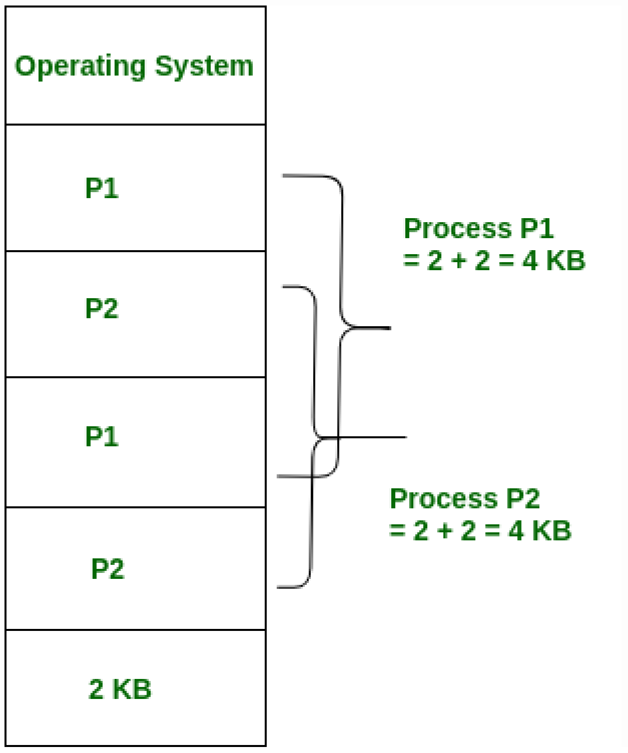
Size of page in process

= Size of frame in memory

Although their numbers can be different. Below diagram will make you understand in better way: consider empty main memory having size of each frame is 2 KB, and two processes P1 and P2 are 2 KB each.



Resolvent main memory,

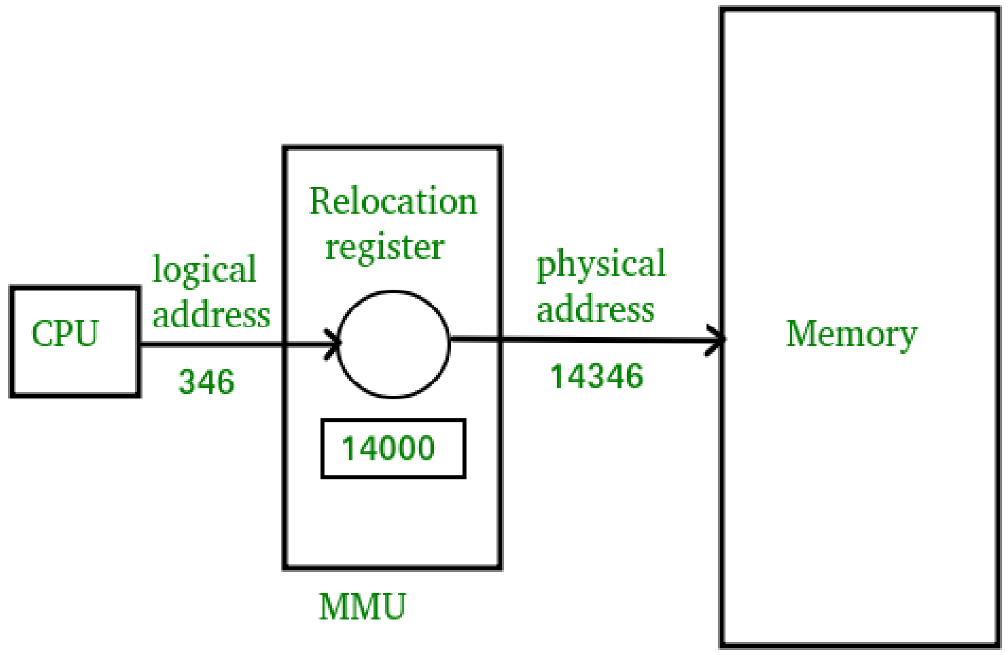


In conclusion we can say that, Paging allows memory address space of a process to be non-contiguous. Paging is more flexible as only pages of a process are moved. It allows more processes to reside in main memory than Contiguous memory allocation.

Logical vs Physical Address in Operating System

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Mapping virtual-address to physical-addresses

Differences Between Logical and Physical Address in Operating System

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.

5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Comparison Chart:

Paramenter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.

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 available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

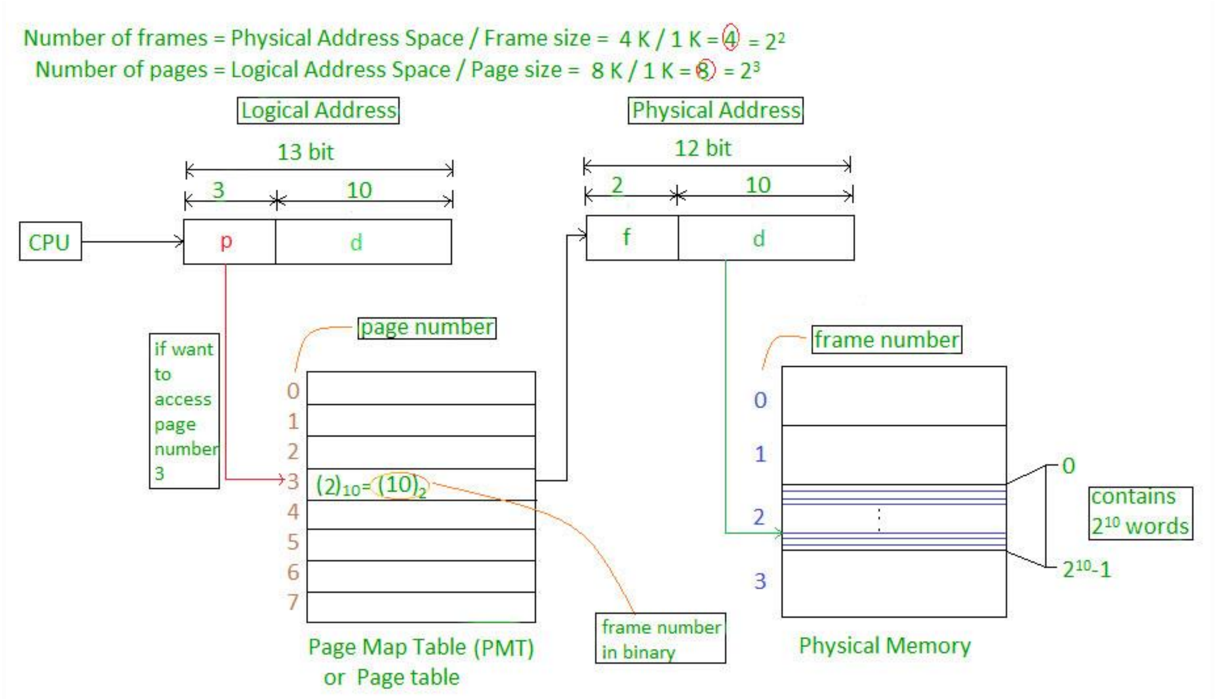
- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (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 paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted 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
CPU
p
d
Physical Address
12 bit
2
10
f
d
if want to access page number 3
page number
0
1
2
3
4
5
6
7
(2)10 = (10)2
Page Map Table (PMT) or Page table
frame number in binary
frame number
0
1
2
3
0
contains 2^10 words
2^10-1
Physical Memory

Address generated by CPU is divided into

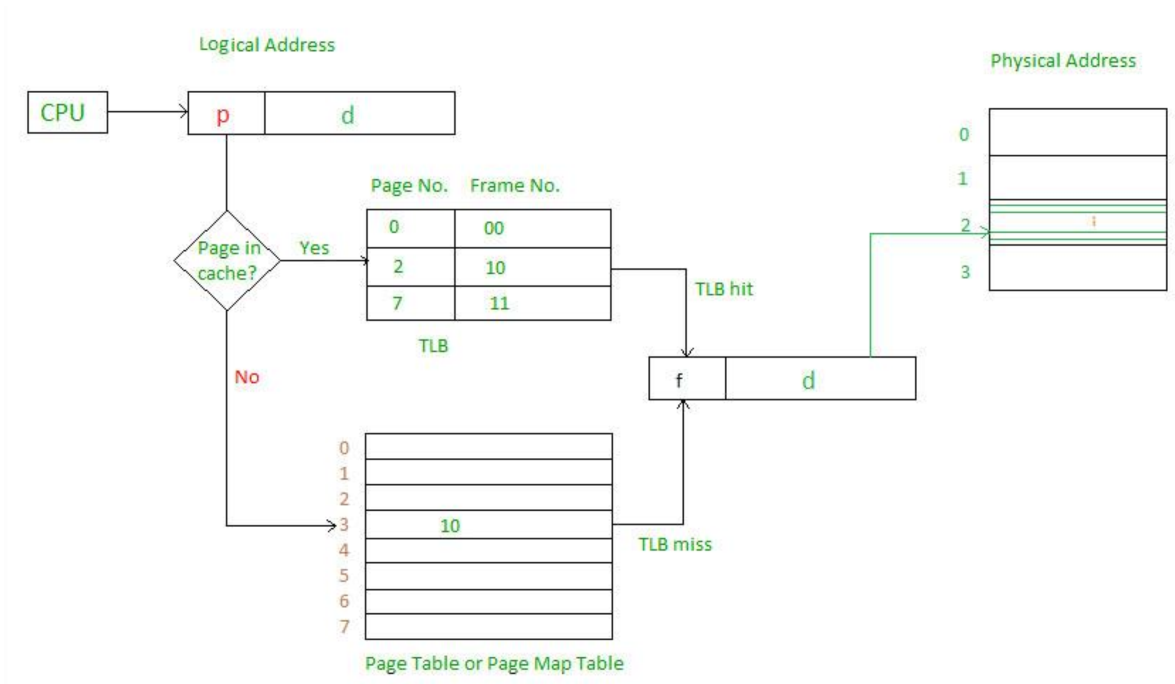
- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

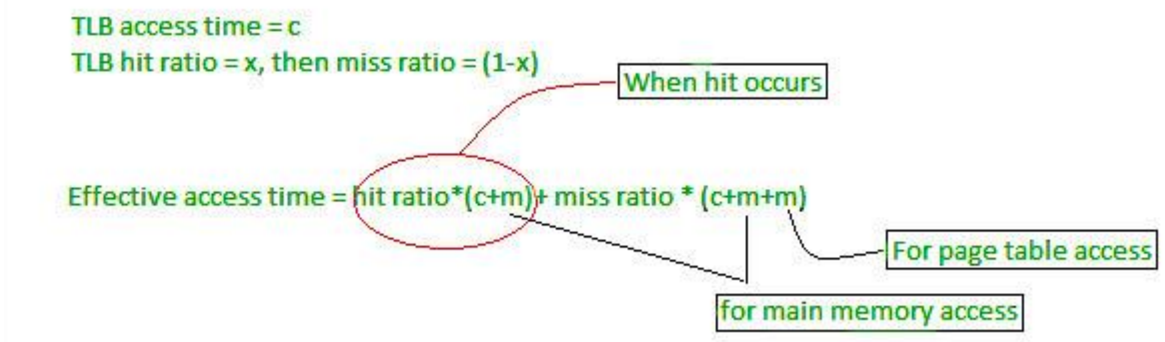
- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

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

- The TLB is 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 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)


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