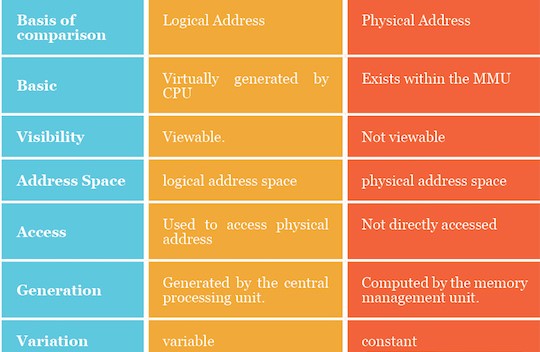
**PART E**

**15.a**



**15.b**

Each process is contained in a single contiguous section of memory.

There are two methods namely :

**Fixed–Partition Method**

**Variable–Partition Method**

**Fixed – Partition Method :**

 Divide  memory  into  fixed  size  partitions,  where  each  partition  has exactly one process.

 The   drawback   is    memory   space    unused   within    a    partition    is

wasted. (eg. when process size < partition size)

**Variable-partition method:**

Divide memory into variable size partitions, depending upon the size of the incoming process.

        When a process terminates, the partition becomes available for another process.

 As  processes complete and  leave they create holes in  the main memory.

        Hole – block of available memory; holes of various size are scattered throughout memory.

**Dynamic Storage-Allocation Problem**:-How to satisfy a request of size =n from a list of free holes?

**Solutions:**

**First-fit:**Allocate the first hole that is big enough.

**Best-fit:**Allocate the smallest hole that is big enough; must search entire list,unless ordered by size. Produces the smallest leftover hole.

**Worst-fit:**Allocate the largest hole; must also search entire list. Produces thelargest leftover hole.

**16.a**

The **Demand Paging** is also same with the Simple Paging. But the Main Difference is that in the Demand Paging Swapping is used. Means all the Pages will be in and out from the Memory when they are required. When we specify a Process for the Execution then the Processes is stored firstly on the Secondary Memory which is also known as the Hard Disk.

But when they are required then they are Swapped Backed into the Memory and when a Process is not used by the user then they are Temporary Swapped out from the Memory. Means they are Stored on the Disk and after that they are Copied into the Memory.

So Demand Paging is the Concept in which a Process is Copied into the Logical Memory from the Physical Memory when we needs them. A Process can load either Entire, Copied into the Main Memory or the part of single Process is copied into the Memory so that is only the single Part of the Process is copied into the Memory then this is also called as the Lazy Swapping.

For Swapping the Process from the Main Memory or from the Physical Memory, a Page Table must be used. The Page Table is used for Storing the Entries which Contains the Page or Process Number and also the offset Number which indicates the address of the Process where a Process is Stored and there will also be the Special or Extra Bit which is also Known as the Flag Bit which indicates whether the Page is Stored into the Physical Memory.

The Page Table Contains two Entries those are used as valid and invalid means whether the Process is Stored into the Page Table. Or Whether the Demand Program is Stored into the Physical Memory So that they can be easily swapped.  If the Requested Program is not stored into the Page Table then the Page Table must Contains the Entries as v and I means valid and invalid along the Page Number.

When a user Request for any Operation then the Operating System perform the following instructions:-

1) First of all this will fetch all the instructions from the Physical Memory into the Logical Memory.  
2) Decode all the instructions means this will find out which Operation has to be performed on the instructions.  
3) Perform Requested Operation.  
4) Stores the Result into the Logical Memory and if needed the Results will be Stored into the Physical Memory.

**Advantages**

* Only loads pages that are demanded by the executing process.
* As there is more space in main memory, more processes can be loaded, reducing the [context switching](https://en.wikipedia.org/wiki/Context_switch) time, which utilizes large amounts of resources.
* Less loading latency occurs at program startup, as less information is accessed from secondary storage and less information is brought into main memory.
* As main memory is expensive compared to secondary memory, this technique helps significantly reduce the bill of material cost in smart phones.

**17.a**

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.

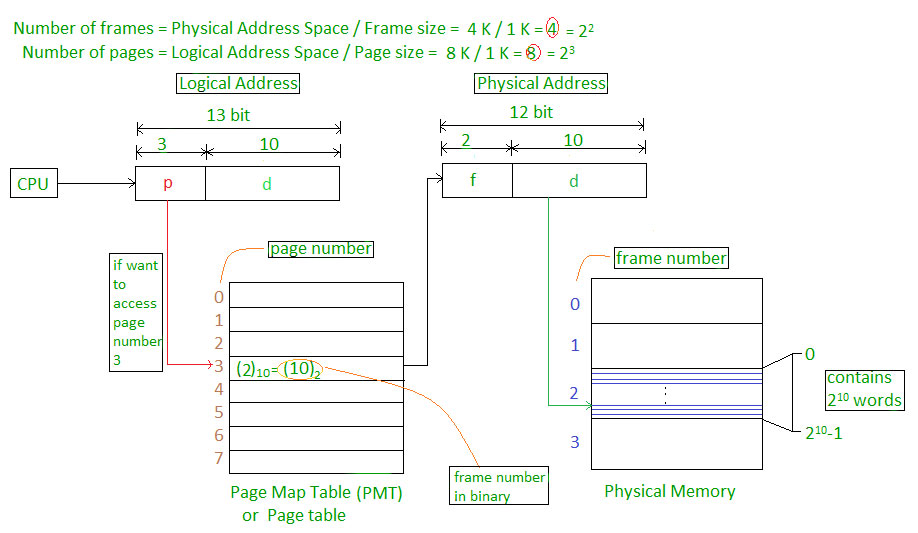
* Logical Address or Virtual Address (represented in bits): An address generated by the CPU
* Logical Address Space or Virtual Address Space( represented in words or bytes): The set of all logical addresses generated by a program
* Physical Address (represented in bits): An address actually available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

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)

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/paging.jpg)

Address generated by CPU is divided into

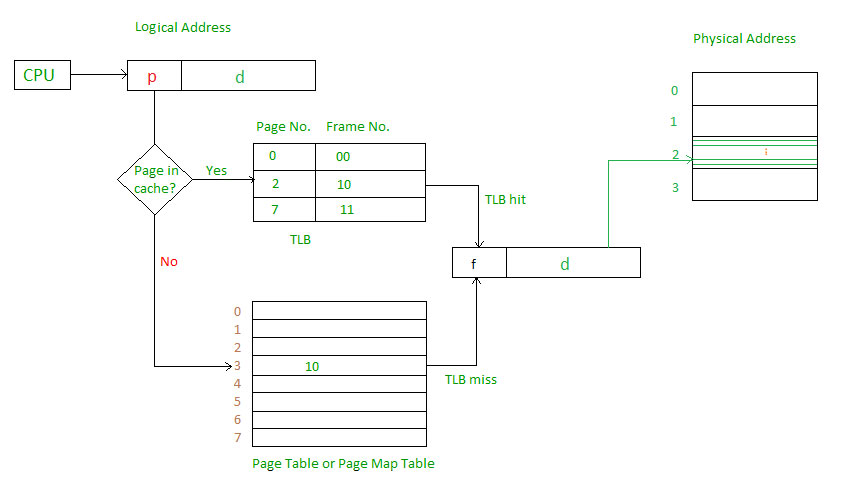
* **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
* **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

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.

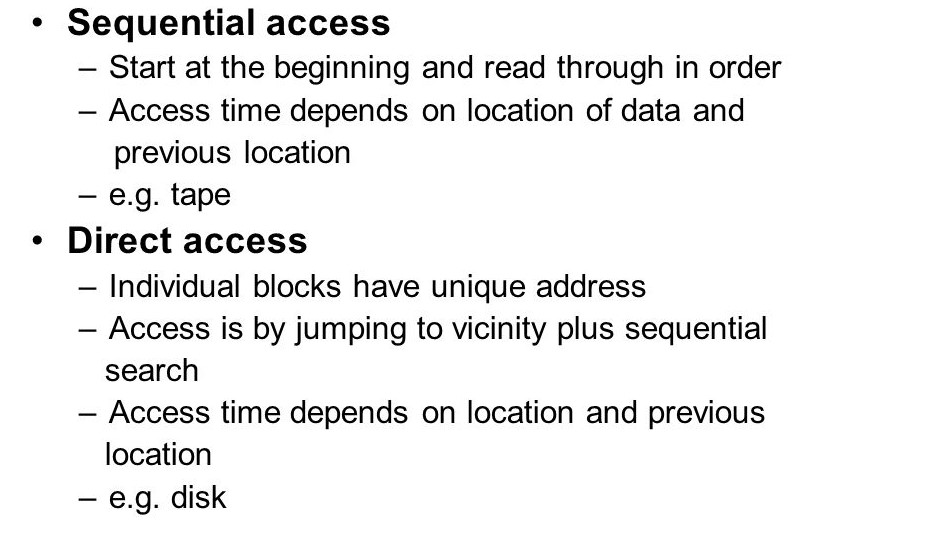
* The TLB is associative, high speed memory.
* Each entry in TLB consists of two parts: a tag and a value.
* When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then corresponding value is returned.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/paging-2.jpg)

**17.b**

When we use a paging scheme, we have no external fragmentation: any free frame can be allocated to a process that needs it. However, we may have some internal fragmentation. Notice that frames are allocated as units. If the memory requirements of a process do not happen to coincide with page boundaries, the last frame allocated may not be completely full. For example, if page size is 2,048 bytes, a process of 72,766 bytes will need 35 pages plus 1,086 bytes. It will be allocated 36 frames, resulting in internal fragmentation of 2,048 − 1,086 = 962 bytes. In the worst case, a process would need n pages plus 1 byte. It would be allocated n + 1 frames, resulting in internal fragmentation of almost an entire frame. If process size is independent of page size, we expect internal fragmentation to average one-half page per process. This consideration suggests that small page sizes are desirable. However, overhead is involved in each page-table entry, and this overhead is reduced as the size of the pages increases. Also, disk I/O is more efficient when the amount data being transferred is larger (Chapter 10). Generally, page sizes have grown over time as processes, data sets, and main memory have become larger. Today, pages typically are between 4 KB and 8 KB in size, and some systems support even larger page sizes. Some CPUs and kernels even support multiple page sizes. For instance, Solaris uses page sizes of 8 KB and 4 MB, depending on the data stored by the pages. Researchers are now developing support for variable on-the-fly page size.

**18.a**



**18.b**

File Access Control functions much like a bank. Inside your local bank is a vault, with safety deposit boxes. You can store your valuables, such as the deed to your home inside the safety deposit box knowing that no one can access that deed without access to the vault and the key to your safety deposit box. In a similar manner, important computer files can be protected by the operating system's File Access Control feature. In this lesson we will explore how operating system can protect or allow access to files and directories.

Operating system controls the file access by setting permissions to files and directories. Permissions can be set to grant or denyaccess to specific files and directories. When a permission is granted, you can access and perform any function on the file or directory. When permission is denied, you are prevented from accessing that file or directory. The most common permissions are Read, Write, Delete, and Execute.

* **Read** permission to allows a user to open and read a file or directory.
* **Write** permission allows you to open the file or directory, make changes, and save those changes.
* **Delete** permission allows you to delete the file or directory.
* **Execute** permission allows you to run an executable file. Certain files are executable files, usually ending in .exe or .com which starts an application on your computer.

**19.a**

A new magnetic disk is a blank slate: it is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting, or physical formatting. Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually512 bytes in size), and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC). When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area. When the sector is read, the ECC is recalculated and compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad. The ECC is an error-correcting code because it contains enough information, if only a few bits of data have been corrupted, to enable the controller to identify which bits have changed and calculate what their correct values should be. It then reports a recoverable soft error. The controller automatically does the ECC processing whenever a sector is read or written.

For many hard disks, when the disk controller is instructed to low-level-format the disk, it can also be told how many bytes of data space to leave between the header and trailer of all sectors. It is usually possible to choose among a few sizes, such as 256, 512, and 1,024 bytes. Formatting a disk with a larger sector size means that fewer sectors can fit on each track; but it also means that fewer headers and trailers are written on each track and more space is available for user data. Some operating systems can handle only a sector size of 512 bytes. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. The second step is logical formatting, or creation of a file system. In this step, the operating system stores the initial file-system data structures onto the disk. To increase efficiency, most file systems group blocks together into larger chunks, frequently called clusters. Some operating systems give special programs the ability to use a disk partition as a large sequential array of logical blocks, without any file-system data structures.

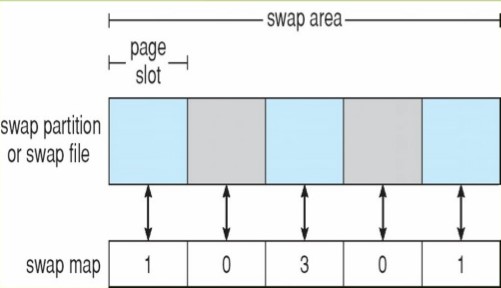
**19.b**

It is moving entire processes between disk and main memory. It occurs when the amount of physical memory reaches a critically low point and processes are moved from memory to swap space to free available memory. In practice most systems combine swapping with virtual memory techniques.

That is the merging of these two concepts “swapping” and “paging”. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system. In this section, we discuss how swap space is used, where swap space is located on disk, and how swap space is managed.

Swap space is used in various ways by different operating systems, depending on the memory-management algorithms in use. The amount of swap space needed on a system can therefore vary from a few megabytes of disk space to gigabytes, depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used.

These swap spaces are usually placed on separate disks so that the load placed on the I/O system by paging and swapping can be spread over the system’s I/O Bandwidth.The traditional UNIX kernel started with an implementation of swapping that copied entire processes between contiguous disk regions and memory. UNIX later evolved to a combination of swapping and paging as paging hardware became available.

In Solaris 1 (SunOS), the designers changed standard UNIX methods to improve efficiency and reflect technological developments. When a process executes, text-segment pages containing code are brought in from the file system, accessed in main memory, and thrown away if selected for page out. 

Swap space is only used as a backing store for pages of anonymous memory, which includes memory allocated for the stack, heap, and uninitialized data of a process. More changes were made in later versions of Solaris. The biggest change is that Solaris now allocates swap space only when a page is forced out of physical memory, rather than when the virtual memory page is first created. This scheme gives better performance on modern computers, which have more physical memory than older systems and tend to page less. Linux is similar to Solaris in that swap space is used only for anonymous memory—that is, memory not backed by any file. Linux allows one or more swap areas to be established. A swap area may be in either a swap file on a regular file system or a dedicated swap partition. Each swap area consists of a series of 4-KB page slots, which are used to hold swapped pages. Associated with each swap area is a swap map—an array of integer counters, each corresponding to a page slot in the swap area. If the value of a counter is 0, the corresponding page slot is available. Values greater than 0 indicate that the page slot is occupied by a swapped page. The value of the counter indicates the number of mappings to the swapped page. For example, a value of 3 indicates that the swapped page is mapped to three different processes (which can occur if the swapped page is storing a region of memory shared by three processes).