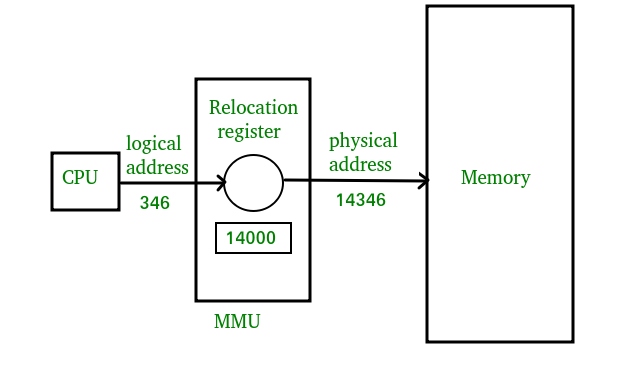
**UNIT - IV**

**Memory Management and Virtual Memory** - Logical versus Physical Address Space, Swapping, Contiguous Allocation, Paging, Segmentation, Segmentation with Paging, Demand Paging, Page Replacement, Page Replacement Algorithms.

**4.1 Logical and 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**](https://www.geeksforgeeks.org/memory-management-mapping-virtual-address-physical-addresses/)

Memory consists of large array of words or arrays, each of which has address associated with it. Now the work of CPU is to fetch instructions from the memory based program counter. Now further these instruction may cause loading or storing to specific memory address.

Address binding is the process of mapping from one address space to another address space. Logical address is address generated by CPU during execution whereas Physical Address refers to location in memory unit(the one that is loaded into memory).Note that user deals with only logical address(Virtual address). The logical address undergoes translation by the MMU or address translation unit in particular. The output of this process is the appropriate physical address or the location of code/data in RAM.

**An address binding can be done in three different ways:**

**Compile Time –** If you know that during compile time where process will reside in memory then absolute address is generated i.e physical address is embedded to the executable of the program during compilation. Loading the executable as a process in memory is very fast. But if the generated address space is preoccupied by other process, then the program crashes and it becomes necessary to recompile the program to change the address space.

**Load time –** If it is not known at the compile time where process will reside then relocatable address will be generated. Loader translates the relocatable address to absolute address. The base address of the process in main memory is added to all logical addresses by the loader to generate absolute address. In this if the base address of the process changes then we need to reload the process again.

**Execution time-** The instructions are in memory and are being processed by the CPU. Additional memory may be allocated and/or de-allocated at this time. This is used if process can be moved from one memory to another during execution(dynamic linking-Linking that is done during load or run time). e.g – Compaction.

**MMU(Memory Management Unit)-**  
The run time mapping between Virtual address and Physical Address is done by hardware device known as MMU.

In memory management, Operating System will handle the processes and moves the processes between disk and memory for execution . It keeps the track of available and used memory.

**Instruction-execution cycle Follows steps:**

1. First instruction is fetched from memory e.g. ADD A,B
2. Then these instructions are decoded i.e., Addition of A and B
3. And further loading or storing at some particular memory location takes place.

**Basic Hardware**

As main memory and registers are built into processor and CPU can access these only. So every instructions should be written in direct access storage  
devices.

1. If CPU access instruction from register then it can be done in one CPU clock cycle as registers are built into CPU.
2. If instruction resides in main memory then it will be accessed via memory bus that will take lot of time. So remedy to this add fast memory in between CPU and main memory i.e. adding cache for transaction.
3. Now we should insure that process resides in legal address.
4. Legal address consists of base register(holds smallest physical address) and limit register(size of range).

**For example:**

Base register = 300040

limit register = 120900

then legal address = (300040+120900)= 420940(inclusive).

legal address = base register+ limit register

**How processes are mapped from disk to memory**

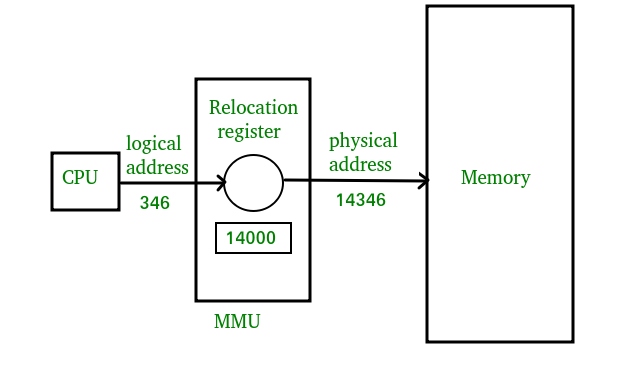
1. Usually process resides in disk in form of binary executable file.
2. So to execute process it should reside in main memory.
3. Process is moved from disk to memory based on memory management in use.
4. The processes waits in disk in form of ready queue to acquire memory.

**Procedure of mapping of disk and memory**

Normal procedure is that process is selected from input queue and loaded in memory. As process executes it accesses data and instructions from memory and as soon as it completes it will release memory and now memory will be available for other processes.

**MMU scheme –**

CPU------- MMU------Memory



1. CPU will generate logical address for eg: 346
2. MMU will generate relocation register(base register) for eg:14000
3. In Memory physical address is located eg:(346+14000= 14346)

**Differences Between Logical and Physical Address in Operating System**

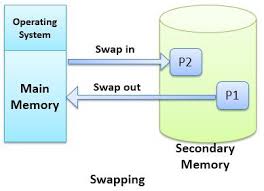
1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas 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.
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).

| **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. |

**4.2 Swapping**

Memory swapping is a computer technology that enables an operating system to provide more memory to a running application or process than is available in physical [random access memory](https://www.enterprisestorageforum.com/storage-hardware/random-access-memory-ram.html) (RAM). When the physical system memory is exhausted, the operating system can opt to make use of memory swapping techniques to get additional memory.

Memory swapping is among the multiple techniques for memory management in modern systems. Physical memory alone is sometimes not sufficient, which is why there are different ways of augmenting RAM in a system with these additional options.



Memory swapping works by making use of [virtual memory](https://www.enterprisestorageforum.com/storage-hardware/virtual-memory.html) and storage space in an approach that provides additional resources when required

With memory swapping, the operating system makes use of storage disk space to provide the functional equivalent of memory storage execution space. The space on the storage device is referred to as "swap space" and is used to run processes that have been swapped out of main physical memory.

The process of memory swapping is managed by an operating system or by a virtual machine hypervisor. Swapping is often enabled by default, though users can choose to disable the capability.

The actual memory swapping process and the creation of a swap file is automatically managed by the operating system. It is initiated when needed as physical RAM is used and additional capacity is required by processes and applications. As additional RAM is required, the state of the physical memory page is mapped to the swap space, enabling a form of virtual (non-physical RAM) memory capacity.

In other words, the main purpose of swapping in memory management is to enable more usable memory than held by the computer hardware.

There are times when physical memory will be allocated and a process needs additional memory. Rather than limiting a system to only having memory that is based on physical RAM, memory swapping enables operating systems and their users to extend memory to disk.

Memory swapping is an essential component of modern memory management helping to ensure availability and overall system stability.

**Swap Space or Swap File**

Swap space is storage space that is used as temporary memory capacity, when physical memory space is already exhausted. The swap file is the physical disk storage file for swap space that is used by an operating system to extend usable memory.

Understanding swap file and swap space is all about understanding memory management. Physical memory in a modern operating system is segmented in different ways, using virtual memory as an abstraction to combine both physical RAM and often swap space, as usable RAM for application processes. In memory management, operating systems make use of a page table to segment and define different memory locations. With memory swapping, the contents of memory stored in a physical element of the page table are copied to disk to maintain the same state for processes.

A swap file and its associated page of memory can be restored to different areas of a system's virtual memory as physical memory is reclaimed over time by the operating system.

The process of how to check swap memory can vary based on operating system. In Microsoft Windows operating systems, information about swap memory is listed under task manager as virtual memory. In Linux, swap space can be checked from the command line with by typing 'swapon-s', which will show allocated swap space usage.

**Advantages of Memory Swapping**

* **More Memory.** Memory swapping is a critical component of memory management, enabling an operating system to handle requests that would otherwise overwhelm a system.
* **Continuous Operations.** Swap file memory can be written to disk in a continuous manner, enabling faster lookup times for operations.
* **System Optimization.** Application processes of lesser importance and demand can be relegated to swap space, saving the higher performance physical memory for higher value operations.

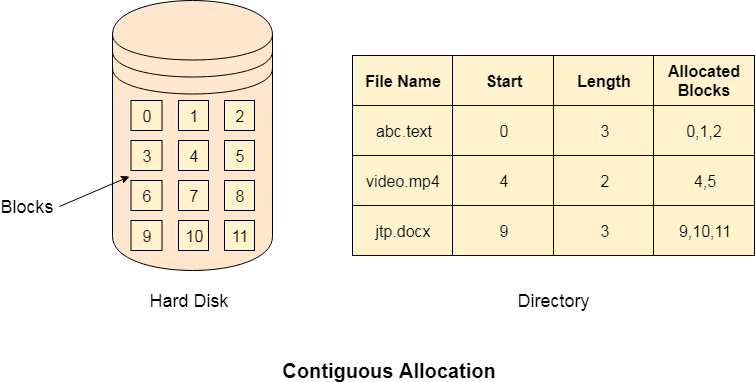
**Limitations of Memory Swapping**

* **Performance.** Disk storage space, when called up by memory swapping, does not offer the same performance as physical RAM for process execution.
* **Disk Limitations**. Swap files are reliant on the stability and availability of storage media, which might not be as stable as system memory.
* **Capacity.** Memory swapping is limited by the available swap space that has been allocated by an operating system or hypervisor.

# 4.3 Contiguous Allocation

If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.

In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



**Advantages**

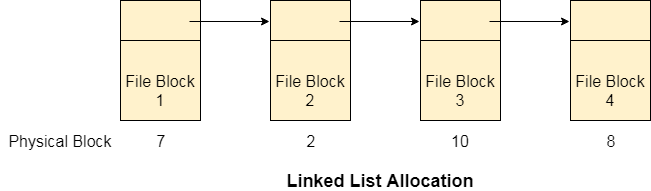
1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

**Disadvantages**

1. The disk will become fragmented.
2. It may be difficult to have a file grow.

**4.3.1 Linked List Allocation**

Linked List allocation solves all problems of contiguous allocation. In linked list allocation, each file is considered as the linked list of disk blocks. However, the disks blocks allocated to a particular file need not to be contiguous on the disk. Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.



**Advantages**

1. There is no external fragmentation with linked allocation.
2. Any free block can be utilized in order to satisfy the file block requests.
3. File can continue to grow as long as the free blocks are available.
4. Directory entry will only contain the starting block address.

**Disadvantages**

1. Random Access is not provided.
2. Pointers require some space in the disk blocks.
3. Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
4. Need to traverse each block.

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

### Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

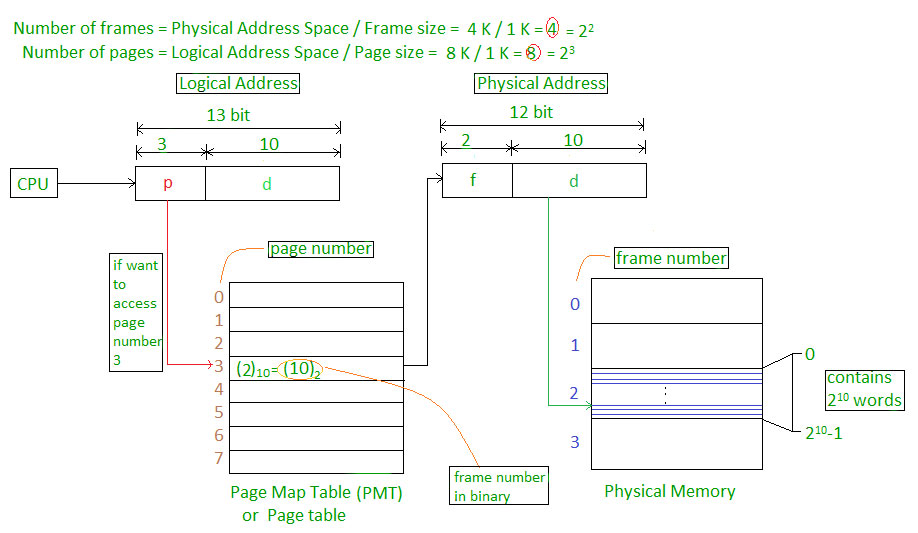
Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

**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://tutorialspoint.dev/image/paging.jpg)

**Address generated by CPU is divided into**

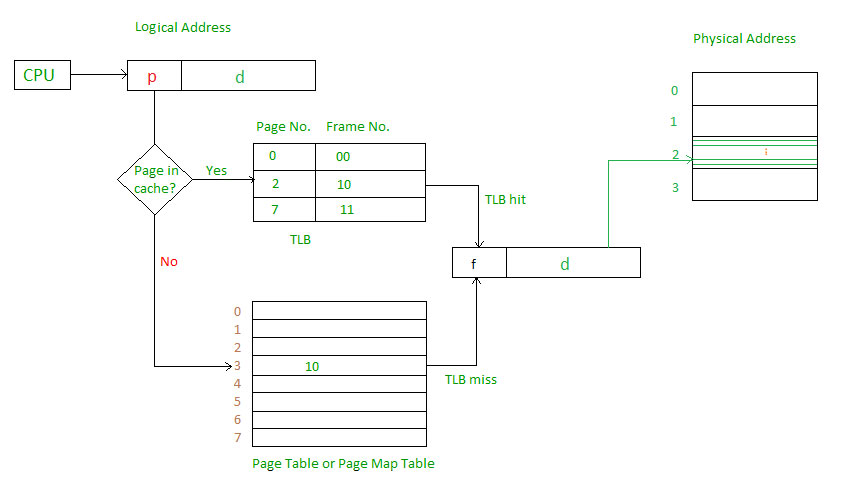
* **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
* **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

**Physical Address is divided into**

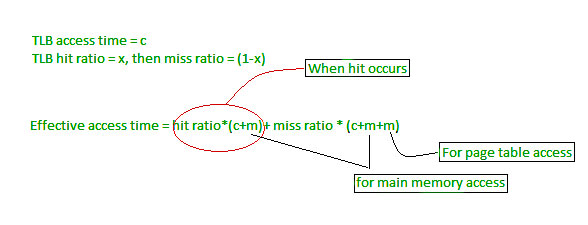
* **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
* **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

**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://tutorialspoint.dev/image/paging-2.jpg)

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).

[](https://tutorialspoint.dev/image/paging-3.jpg)

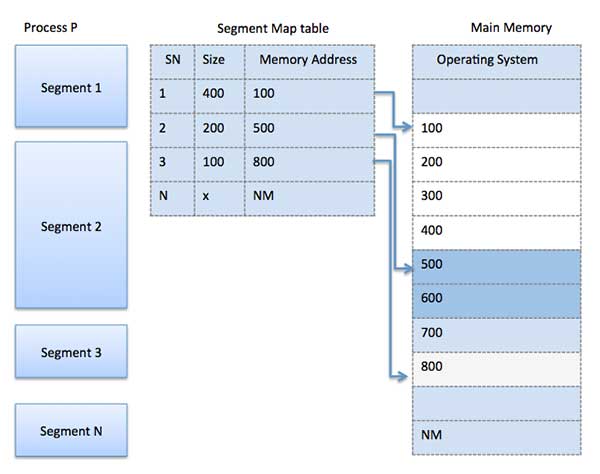
## 4.5 Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



**4.6 Segmented Paging**

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

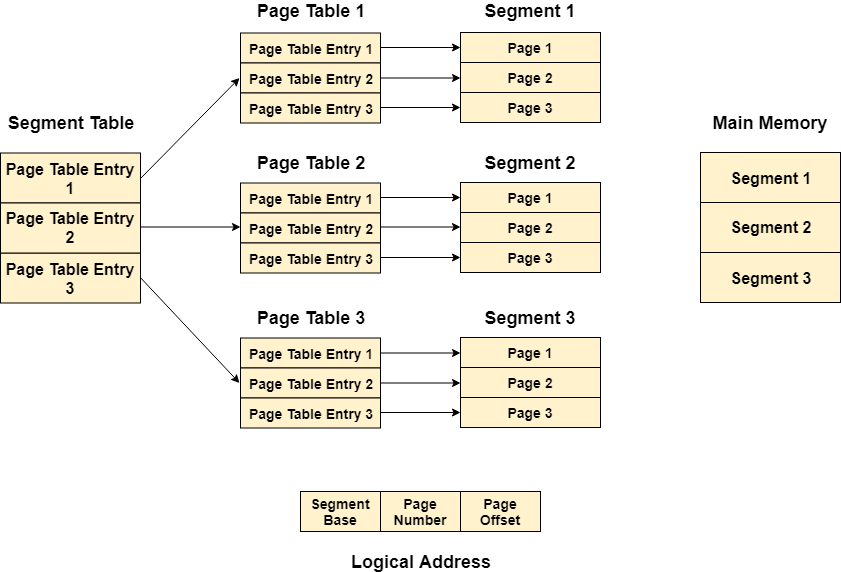
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

**Segment Number →** It points to the appropriate Segment Number.

**Page Number →** It Points to the exact page within the segment

**Page Offset →** Used as an offset within the page frame

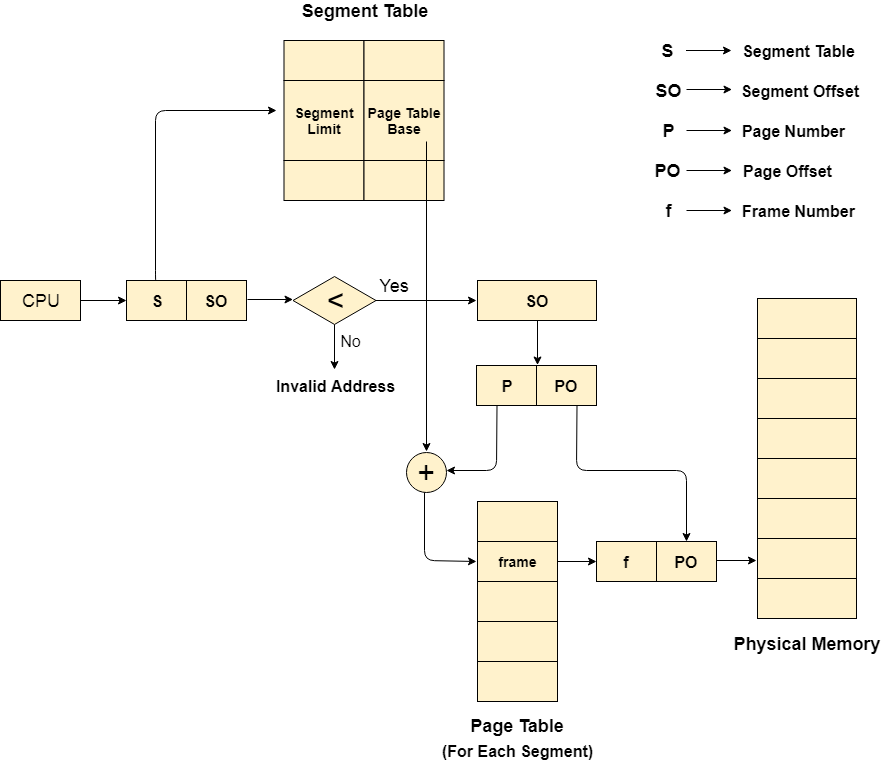
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



**Translation of logical address to physical address**

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



**Advantages of Segmented Paging**

1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

**Disadvantages of Segmented Paging**

1. Internal Fragmentation will be there.
2. The complexity level will be much higher as compare to paging.
3. Page Tables need to be contiguously stored in the memory.

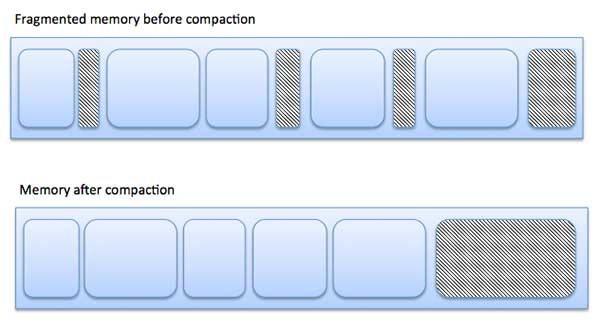
**4.6.1 Fragmentation**

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

**Fragmentation is of two types −**

|  |  |
| --- | --- |
| **S.N.** | **Fragmentation & Description** |
| 1 | **External fragmentation**  Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used. |
| 2 | **Internal fragmentation**  Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. |

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



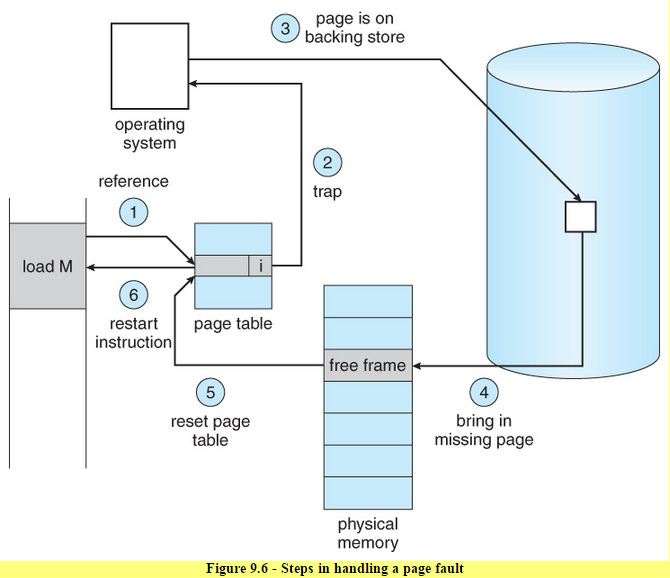
**External fragmentation** can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

**Internal fragmentation** can be reduced by effectively assigning the smallest partition but large enough for the process.

**4.7 Demand Paging**

The main steps involved in demand paging that is in between the page is requested and it is loaded into main memory are as follows:-

1. CPU refers to the page it needs.
2. The referred page is checked in page table whether that is present in main memory or not.If not an interrupt page fault is generated.
3. OS puts the interruptes  process in blocking state and  starts the process of fetching the page from memory so that process can be executed.
4. OS will search for it in logical address space.
5. Required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
6. Page table will be updated.
7. Interrupted process will be restarted.



These are  steps when a page required is missing from the memory.

Performance of Demand Paging

Demand paging can significantly affect the performance of a computer system.  
To see why, let’s compute the effective access time for a demand-paged  
memory. the memory-access time, denoted **ma**,ranges from 10 to 200 nanoseconds.

effective access time = (1 − p) × ma + p × page fault time.

where **p** is probability of page fault.

**Three major activities**  
– Service the interrupt – careful coding means just several hundred instructions needed  
– Read the page – lots of time  
– Restart the process – again just a small amount of time

# 4.8 Page Replacement

# The page replacement decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).

# OS Page Replacement Algorithms

# There are two main aspects of virtual memory, Frame allocation and Page Replacement. It is very important to have the optimal frame allocation and page replacement algorithm. Frame allocation is all about how many frames are to be allocated to the process while the page replacement is all about determining the page number which needs to be replaced in order to make space for the requested page. There are two important point if algorithm is not optimal. Such as

1. If the number of frames which are allocated to a process is not sufficient or accurate then there can be a problem of thrashing. Due to the lack of frames, most of the pages will be residing in the main memory and therefore more page faults will occur. However, if OS allocates more frames to the process then there can be **internal fragmentation**.
2. If the page replacement algorithm is not optimal then there will also be the problem of thrashing. If the number of pages that are replaced by the requested pages will be referred in the near future then there will be more number of swap-in and swap-out and therefore the OS has to perform more replacements then usual which causes performance deficiency. Therefore, the task of an optimal page replacement algorithm is to choose the page which can limit the **thrashing.**

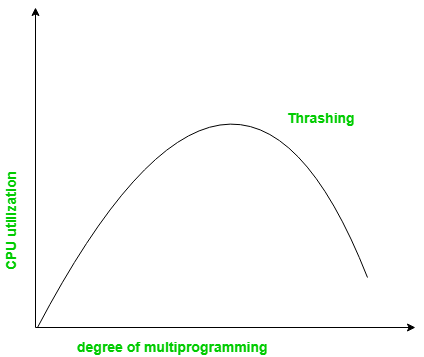
**4.8.1 Thrashing in Operating System**

When a program need space larger than RAM or it need space when RAM is full, Operating System will try to allocate space from secondary memory and behaves like it has that much amount of memory by serving to that program. This concept is called virtual memory. To know about thrashing we first need to know what is page fault and swapping.

**Page fault and swapping:**We know every program divided into some pages. When a program need a page which is not in RAM that is called page fault. Whenever a page fault happens, operating system will try to fetch that page from secondary memory and try to swap it with one of the page in RAM. This is called swapping.

## Thrashing  in OS

If this page fault and then swapping happening very frequently at higher rate, then operating system has to spend more time to swap these pages. This state is called thrashing. Because of this, CPU utilization is going to be reduced.

****

### Effect of Thrashing

Whenever thrashing starts, operating system tries to apply either **Global page replacement**Algorithm or **Local page replacement**algorithm.

**Global Page Replacement**

Since global page replacement can access to bring any page, it tries to bring more pages whenever thrashing found. But what actually will happen is, due to this, no process gets enough frames and by result thrashing will be increase more and more. So global page replacement algorithm is not suitable when thrashing happens.

**Local** **Page Replacement**

Unlike global page replacement algorithm, local page replacement will select pages which only belongs to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. So local page replacement is just alternative than global page replacement in thrashing scenario.

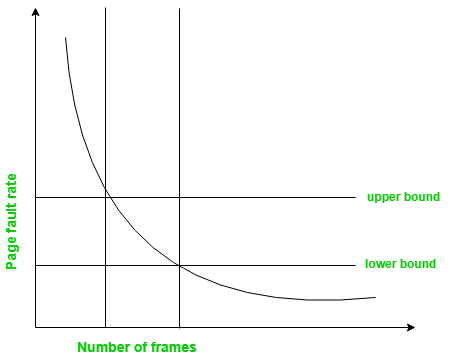
### Techniques to Handle Thrashing

We already seen Local replacement is better than Global replacement to avoid thrashing. But it also has disadvantages and not suggestable. Some more techniques are

**Working Set Model**

This model is based on locality. What locality is saying, the page used recently can be used again and also the pages which are nearby this page will also be used. Working set means set of pages in the most recent D time. The page which completed its D amount of time in working set automatically dropped from it. So accuracy of working set depends on D we have chosen. This working set model avoid thrashing while keeping the degree of multiprogramming as high as possible.

**Page Fault Frequency**



It is some direct approach than working set model. When thrashing occurring we know that it has few number of frames. And if it is not thrashing that means it has too many frames. Based on this property we assign an upper and lower bound for the desired page fault rate. According to page fault rate we allocate or remove pages. If the page fault rate become less than the lower limit, frames can be removed from the process. Similarly, if the page fault rate become more than the upper limit, more number of frames can be allocated to the process. And if no frames available due to high page fault rate, we will just suspend the processes and will restart them again when frames available.

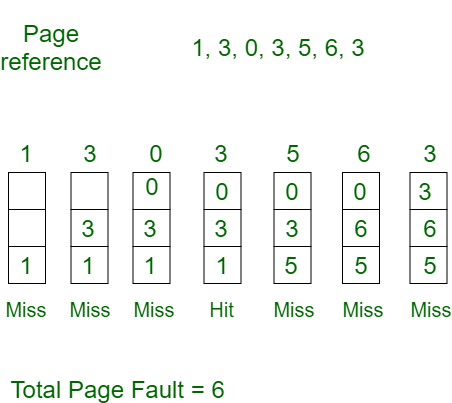
# 4.9 Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

**Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

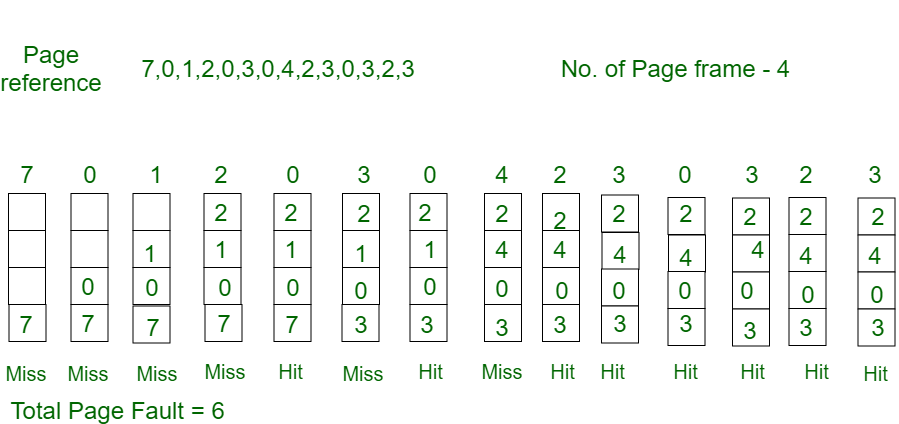
Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults. There are three types of page replacement algorithms such as:

**4.9.1 First In First Out (FIFO) –**  
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.  
  
Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**  
when 3 comes, it is already in  memory so —> **0 Page Faults.**  
Then 5 comes, it is not available in  memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**  
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**  
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

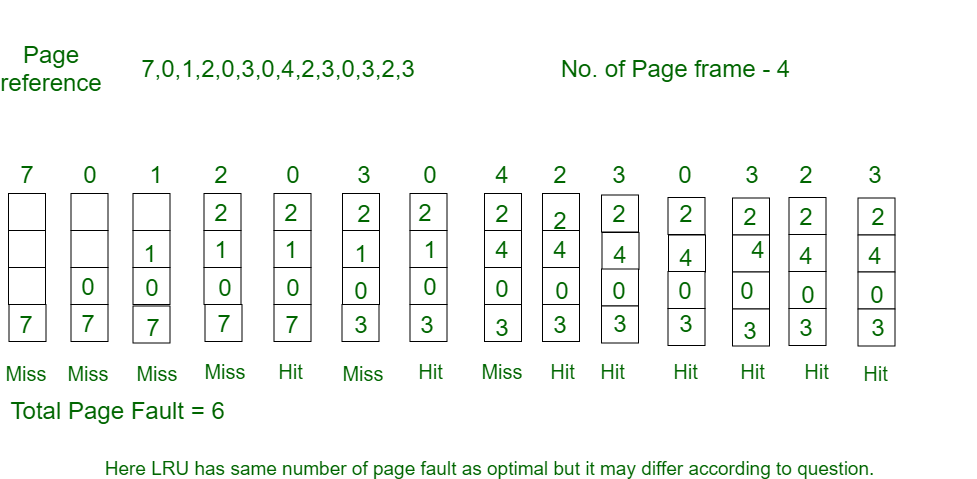
[**Belady’s anomaly**](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/)**–** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.  For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

**4.9.2 Optimal Page replacement –**  
In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Example-2:**Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.  
  
Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**  
0 is already there so —> **0 Page fault.**  
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**  
0 is already there so —>**0 Page fault.**.  
4 will takes place of 1 —>**1 Page Fault.**  
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

**4.9.3 Least Recently Used –**  
In this algorithm page will be replaced which is least recently used.

**Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.  


Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**  
0 is already their so —> **0 Page fault.**  
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**  
0 is already in memory so —>**0 Page fault**.  
4 will takes place of 1 —>**1 Page Fault**  
Now for the further page reference string —>**0 Page fault** because they are already available in the memory.