Memory Management in Operating System

Memory is the important part of the computer that is used to store the data and instructions. Its management is critical to the computer system because the amount of main memory available in a computer system is limited and at any time, many processes are competing for it.

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status. The MMU (Memory Management Unit) is responsible for the management of memory.

Role of Memory management

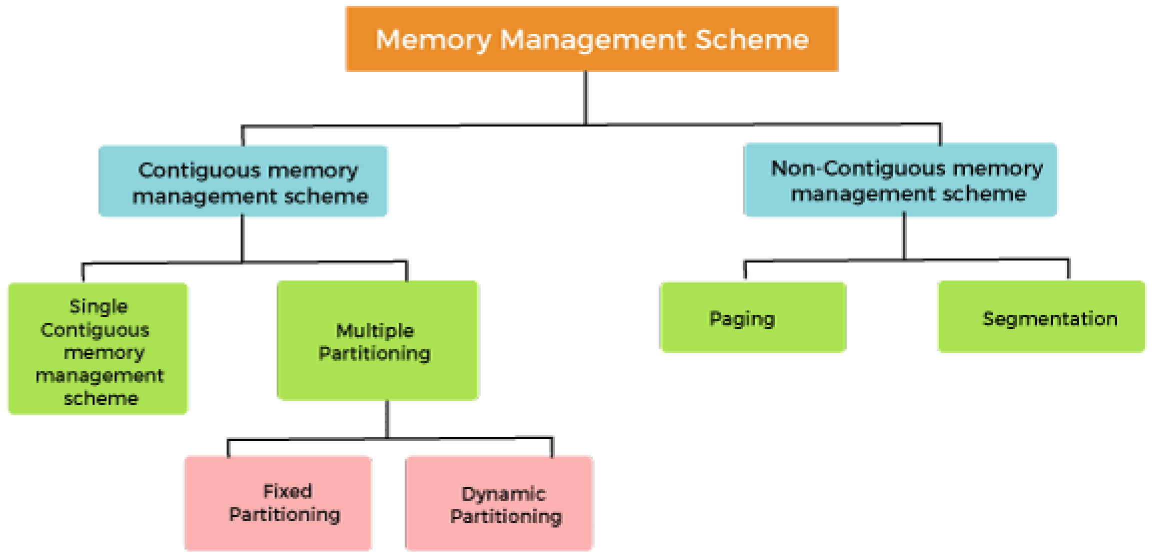
Following are the important roles of memory management in a computer system:

- Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.
- Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.
- The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.
- Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

Memory Management Techniques:

The memory management techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes


Memory Management Scheme
Contiguous memory management scheme
Non-Contiguous memory management scheme
Single Contiguous memory management scheme
Multiple Partitioning
Paging
Segmentation
Fixed Partitioning
Dynamic Partitioning

Classification of memory management schemes

Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition known as high memory.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It cannot be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to be loaded concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus, multiple processes can reside in the main memory simultaneously.

The multiple partitioning schemes can be of two types:

- Fixed/Static Partitioning
- Variable/Dynamic Partitioning

Fixed Partitioning

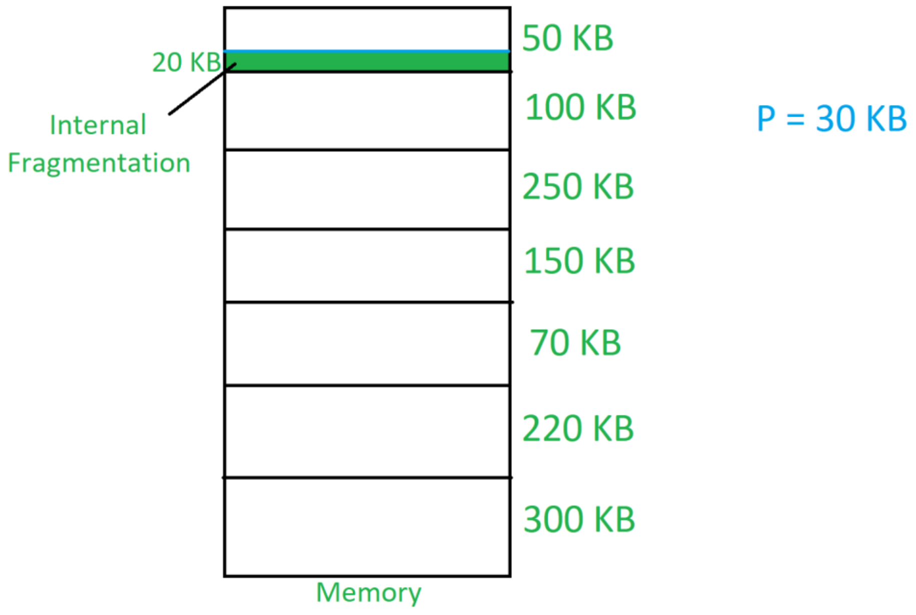
The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system generation.



Dynamic Partitioning

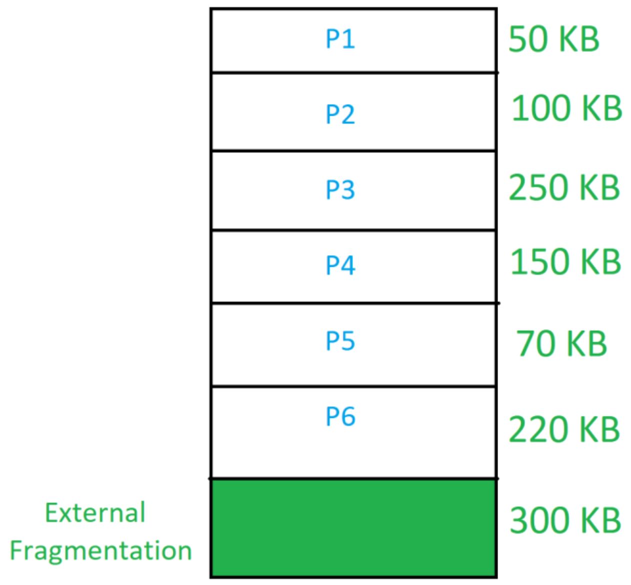
The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

Advantages of Dynamic Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Dynamic Partitioning memory management schemes:

- This scheme also suffers from external fragmentation.
- The number of partitions is specified at the time of system segmentation.



Fragmentation

Fragmentation is an unwanted problem in the OS in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

Types of Fragmentation

There are mainly two types of fragmentation in the operating system. These are as follows:

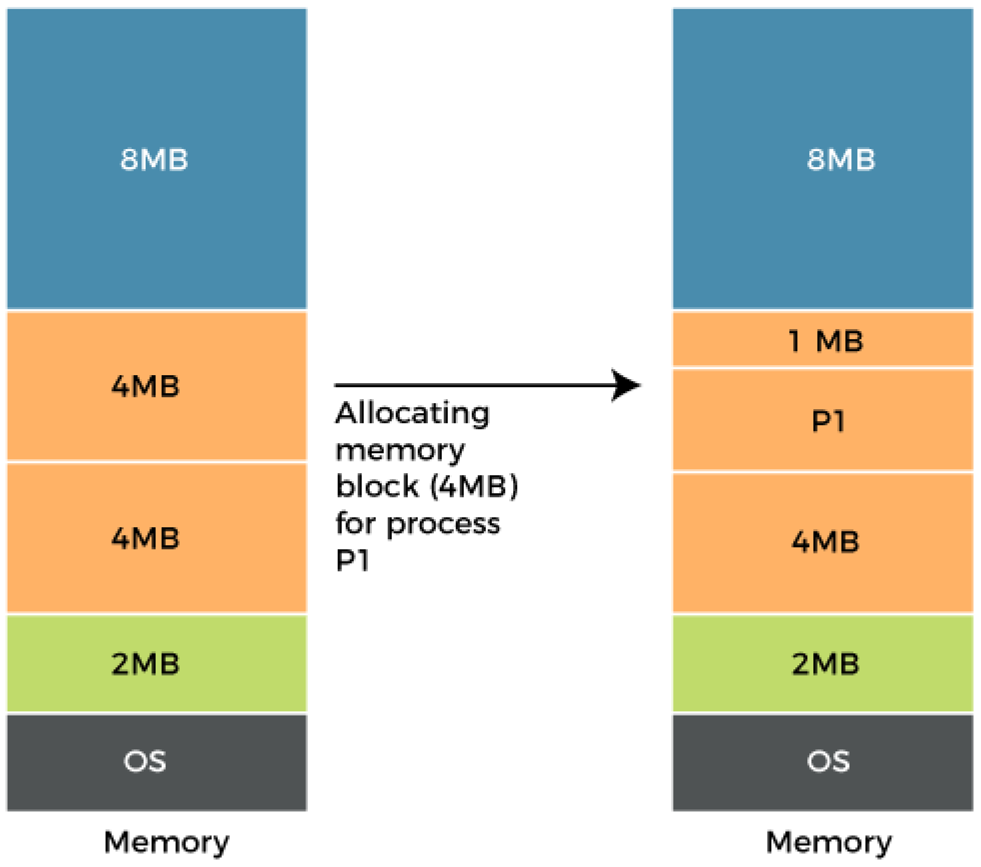
1. **Internal Fragmentation**
2. **External Fragmentation**

Internal Fragmentation

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which is known as **internal** fragmentation.

For Example:

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.



Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as **internal fragmentation**.

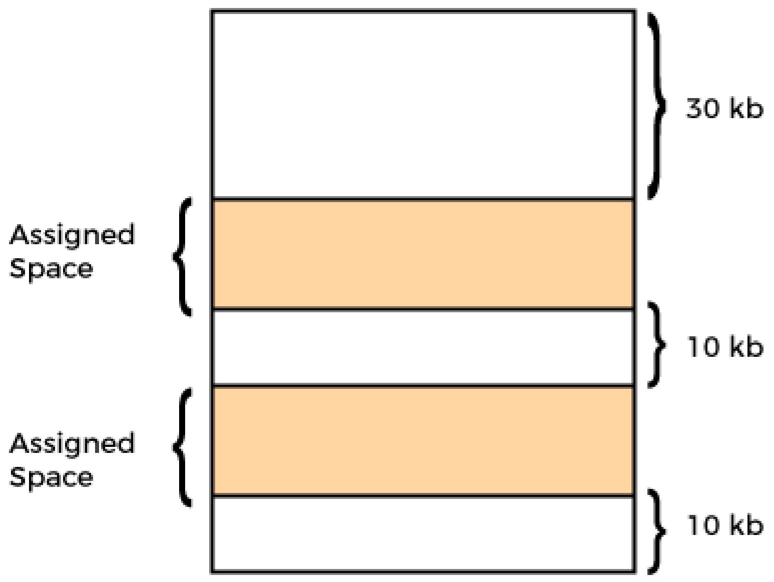
How to avoid internal fragmentation?

The problem of internal fragmentation may arise due to the fixed sizes of the memory blocks. It may be solved by assigning space to the process via dynamic partitioning. Dynamic partitioning allocates only the amount of space requested by the process. As a result, there is no internal fragmentation.

External Fragmentation

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as **external** fragmentation.

For Example:



Process 05 needs 45kb memory space

Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space **(50 KB)** to run a process **(05) (need 45KB)**, but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

How to remove external fragmentation?

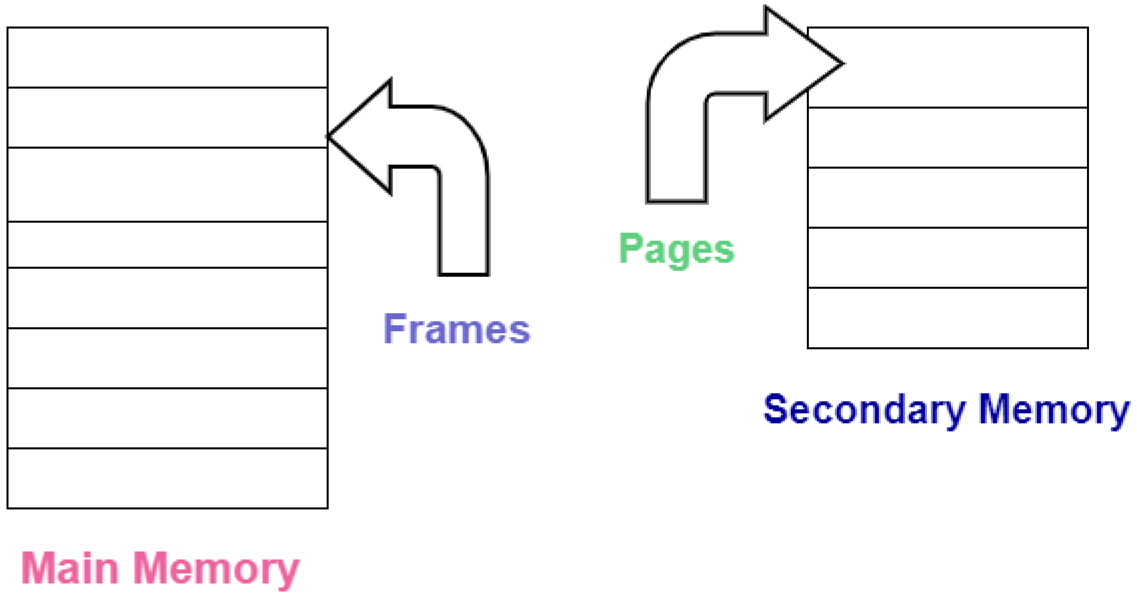
Compaction is used for removing external fragmentation. External fragmentation may be decreased when dynamic partitioning is used for memory allocation by combining all free memory into a single large block. The larger memory block is used to allocate space based on the requirements of the new processes. This method is also known as defragmentation.

The problem of external fragmentation occurs whenever we allocate the RAM to process continuously. It can be done in segmentation and paging, in which the memory is allocated non-contiguously to the processes. And as a result, in case we happen to remove this very condition, then the external fragmentation may ultimately be decreased.

Paging

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages. Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

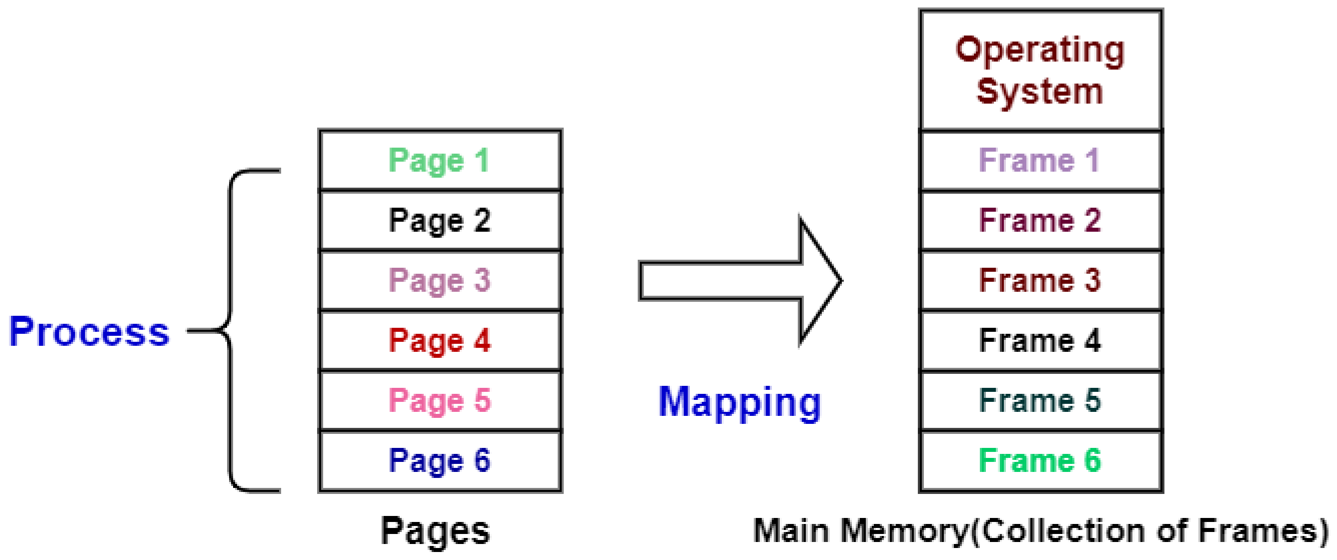
The Frame has the same size as that of a Page. A frame is basically a place where a (logical) page can be (physically) placed.



Each process is mainly divided into parts where the size of each part is the same as the page size.

There is a possibility that the size of the last part may be less than the page size.

- Pages of a process are brought into the main memory only when there is a requirement otherwise they reside in the secondary storage.
- One page of a process is mainly stored in one of the frames of the memory. Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames.



Segmentation

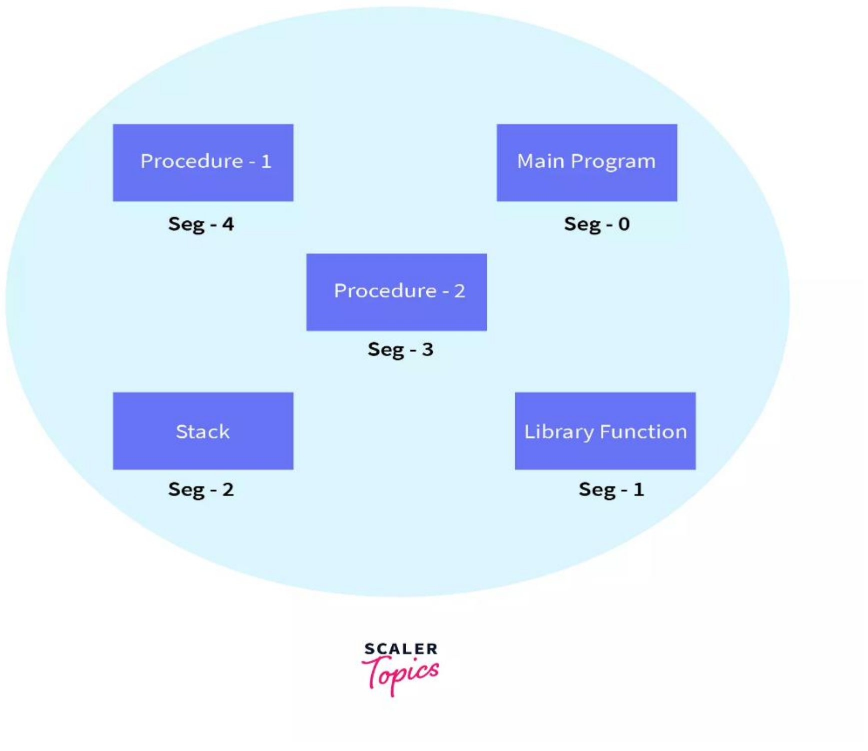
Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program. When a process is to be executed, its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

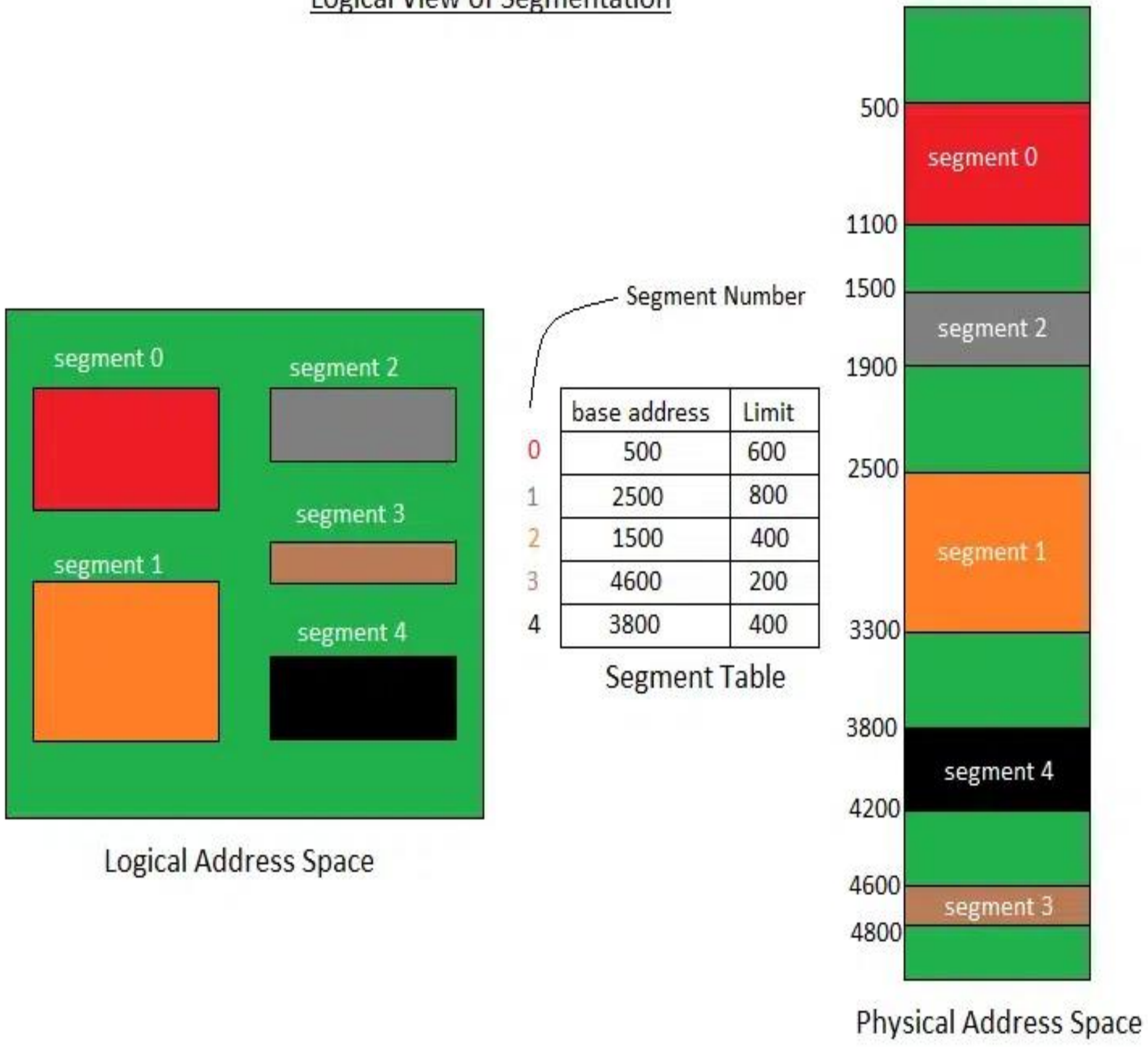
Types of Segmentation

Segmentation can be divided into two types:

1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into **n** number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into **n** number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other (in a noncontinuous manner).

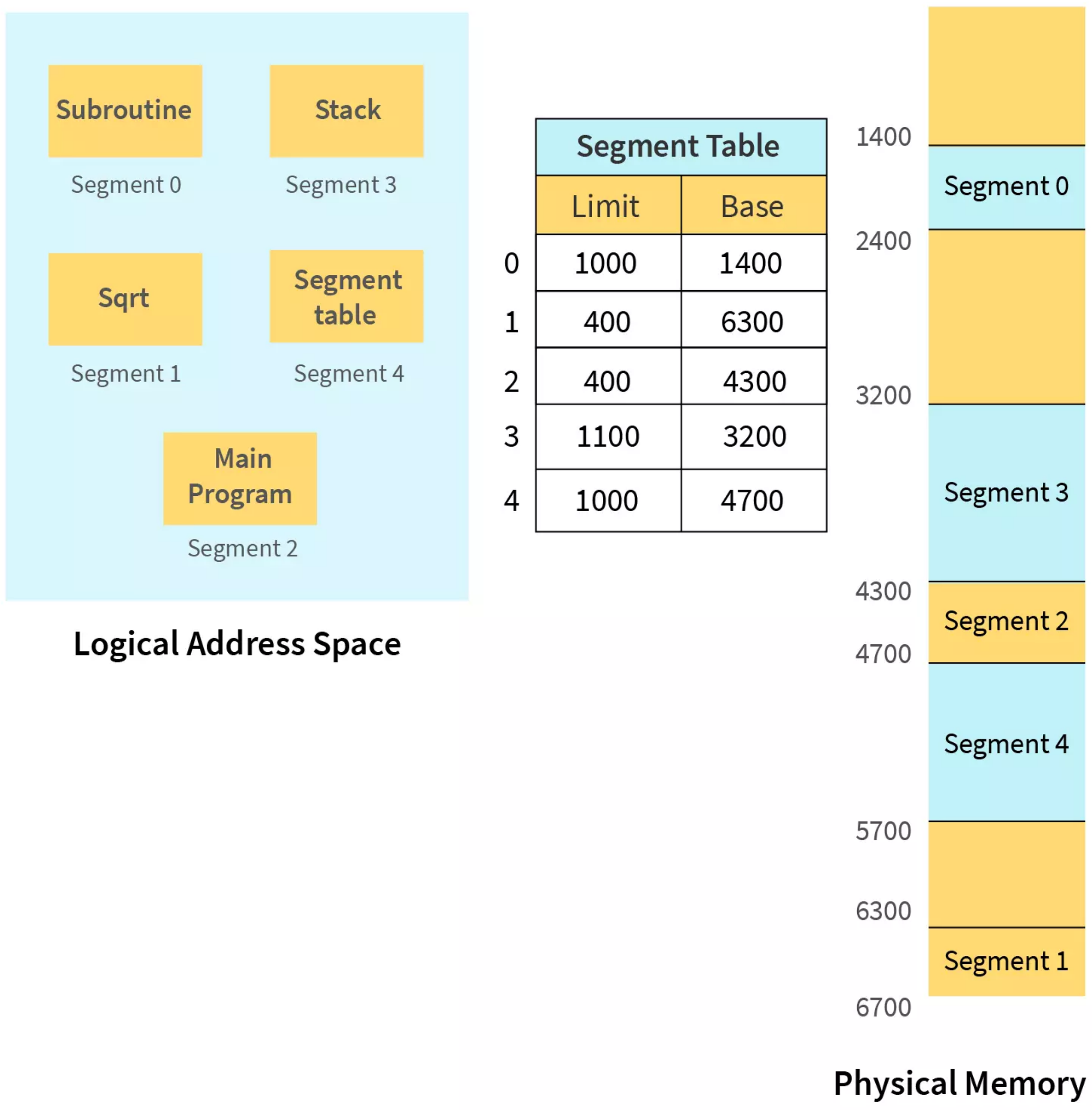


Logical View of Segmentation



	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table



SCALER Topics

Following are the important differences between Paging and Segmentation.

Sr. No.	Key	Paging	Segmentation
1	Memory Size	In Paging, a process address space is broken into fixed sized blocks called pages.	In Segmentation, a process address space is broken in varying sized blocks called sections.
2	Accountability	Operating System divides the memory into pages.	Compiler is responsible to calculate the segment size, the virtual address and actual address.
3	Size	Page size is determined by available memory.	Section size is determined by the user.
4	Speed	Paging technique is faster in terms of memory access.	Segmentation is slower than paging.
5	Fragmentation	Paging can cause internal fragmentation as some pages may go underutilized.	Segmentation can cause external fragmentation as some memory block may not be used at all.
6	Logical Address	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.

Sr. No.	Key	Paging	Segmentation
7	Table	During paging, a logical address is divided into page number and page offset.	During segmentation, a logical address is divided into section number and section offset.
8	Data Storage	Page table stores the page data.	Segmentation table stores the segmentation data.