# Contiguous and Non-Contiguous memory allocation in Operating System

**Memory management**

*Memory is central to the operation of a computer system. It consists of a large array of words or bytes each with its own address. In uniprogramming system, main memory has two parts one for the operating system and another part is for the program currently being executed. In the multiprogramming system, the memory part of the user is further divided into accommodate processes. The task of the subdivision is cannot out by the operating system and is known as memory management.*

**Memory management techniques**

The memory management techniques are divided into two parts...

1. **Uniprogramming:**
   In the uniprogramming technique, the RAM is divided into two parts ***one part is for the resigning the operating system*** *and* ***other portion is for the user process.*** Here the border register is used which contain the last address of the operating system parts. The operating system will compare the user data addresses with the fence register and if it is different that means the user is not entering in the OS area. Border register is also called boundary register and is used to prevent a user from entering in the operating system area. Here the CPU utilization is very poor and hence multiprogramming is used.
2. **Multiprogramming:**
   In the multiprogramming, the multiple users can share the memory simultaneously. By multiprogramming we mean there will be more than one process in the main memory and if the running process wants to wait for an event like I/O then instead of sitting ideal CPU will make a context switch and will pick another process.
   a. **Contiguous memory allocation**
   b. **Non-contiguous memory allocation**

## a) Contiguous memory allocation

The Contiguous memory allocation is one of the methods of memory allocation. In contiguous memory allocation, when a process requests for the memory, a **single contiguous section of memory blocks** is assigned to the process according to its requirement.

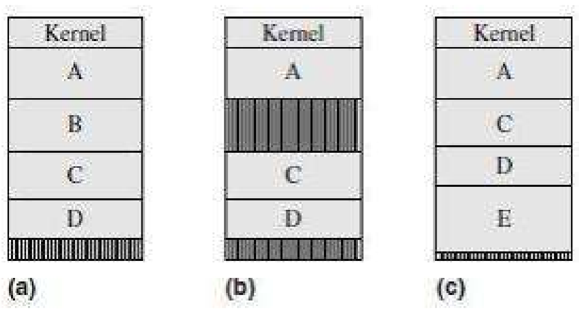
- All the available memory space remains together in one place. *It means freely available memory partitions are not expanded here and there across the whole memory space.*
- Both the operating system and the user must reside in the main memory. *The main memory is divided into two portions one portion is for the operating and other is for the user program.*
- When any user process request for the memory a single section of the contiguous memory block is given to that process according to its need. *We can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.*

A single process is allocated in that fixed sized single partition. But this will increase the degree of multiprogramming means more than one process in the main memory that bounds the number of fixed partition done in memory. Internal fragmentation increases because of the contiguous memory allocation.

**Contiguous Memory Allocation**

**Example**
Processes A, B, C, and D are in memory in Figure. Two free areas of memory exist after B terminates; however, neither of them is large enough to accommodate another process. The kernel performs compaction to create a single free memory area and initiates process E in this area. It involves moving processes C and D in memory during their execution.

Memory compaction involves *dynamic relocation*, which is not feasible without a relocation register. In computers not having a relocation register, the kernel must resort to reuse of free memory areas. However, this approach incurs delays in initiation of processes when large free memory areas do not exist, e.g., initiation of process E would be delayed in Example 4.8 even though the total free memory in the system exceeds the size of E.



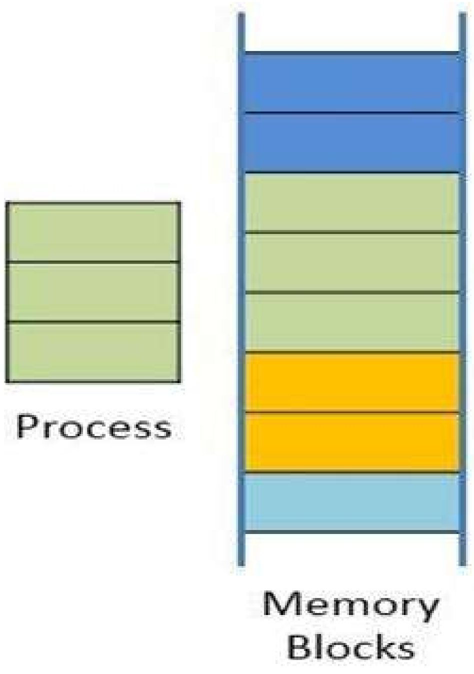
**Swapping**
The kernel swaps out a process that is not in the *running* state by writing out its code and data space to a *swapping area* on the disk. The swapped out process is brought back into memory before it is due for another burst of CPU time. A basic issue in swapping is whether a swapped in process should be loaded back into the same memory area that it occupied before it was swapped out. If so, it's swapping in depends on swapping out of some other process that may have been allocated that memory area in the meanwhile. It would be useful to be able to place the swapped-in process elsewhere in memory; however, it would amount to dynamic relocation of the process to a new memory area. As mentioned earlier, only computer systems that provide a relocation register can achieve it.

Swapping is a technique of temporarily removing inactive programs from the memory of the system. A process can be swapped temporarily out of the memory to a backing store and then brought back in to the memory for continuing the execution. This process is called swapping. Eg:-In a multi-programming environment with a round robin CPU scheduling whenever the time quantum expires then the process that has just finished is swapped out and a new process swaps in to the memory for execution.

A variation of swap is priority based scheduling. When a low priority is executing and if a high priority process arrives then a low priority will be swapped out and high priority is allowed for execution.
This process is also called as Roll out and Roll in. physical address is computed during run time. Swapping requires backing store and it should be large enough to accommodate the copies of all memory images. The system maintains a ready queue consisting of all the processes whose memory images are on the backing store or in memory that are ready to run. Swapping is constant by other factors:

To swap a process, it should be completely idle. A process may be waiting for an i/o operation. If the i/o is asynchronously accessing the user memory for i/o buffers, then the process cannot be swapped Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously. This depends upon address binding.

If the binding is done at load time, then the process is moved to same memory location. If the binding is done at run time, then the process is moved to different memory location. This is because the



**Contiguous Memory Allocation**

→ **Fixed sized partition**

In the fixed sized partition the system divides memory into fixed size partition (may or may not be of the same size) here entire partition is allowed to a process and if there is some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

**Advantage:** Management or book keeping is easy.

**Disadvantage:** Internal fragmentation

→ **Variable size partition**

In the variable size partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

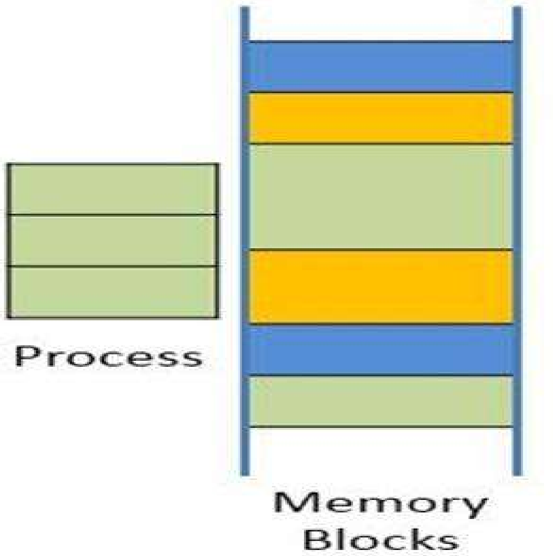
**Advantage:** There is no internal fragmentation.

**Disadvantage:** Management is very difficult as memory is becoming purely fragmented after some time.

## b) Non-contiguous memory allocation

The Non-contiguous memory allocation allows a process to **acquire the several memory blocks at the different location in the memory** according to its requirement. The non-contiguous memory allocation also **reduces** the **memory wastage** caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.

- The available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming.
- A process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement.
- It reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.



**Noncontiguous Memory Allocation**

**Non-contiguous memory allocation** is of different types,

1. **Paging**
2. **Segmentation**
3. **Segmentation with paging**

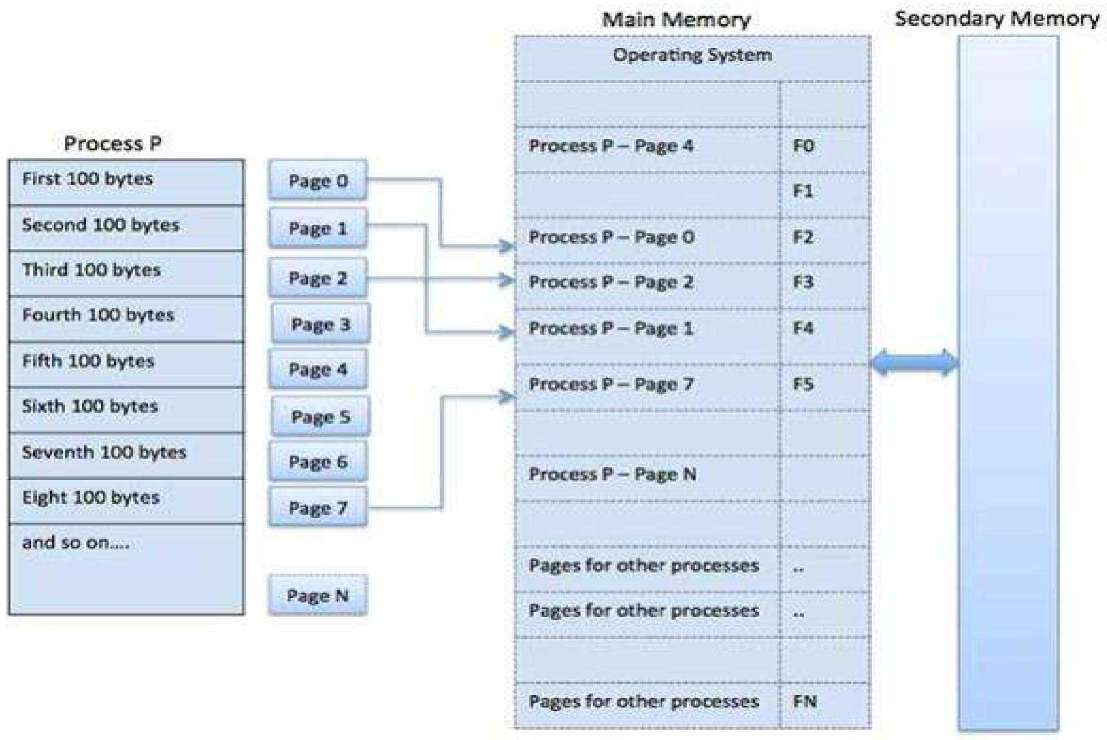
## (I) PAGING

A non-contiguous policy with a fixed size partition is called paging. A computer can address more memory than the amount of physically installed on the system. *This extra memory is actually called virtual memory*. Paging technique is very important in implementing virtual memory.

*Secondary memory is divided into equal size partition (fixed) called pages. Every process will have a separate page table. The entries in the page table are the number of pages a process. At each entry either we have an invalid pointer which means the page is not in main memory or we will get the corresponding frame number.*

*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.*

When the frame number is combined with instruction of set D than we will get the corresponding physical address. Size of a page table is generally very large so cannot be accommodated inside the PCB, therefore, PCB contains a register value PTBR( page table base register) which leads to the page table.

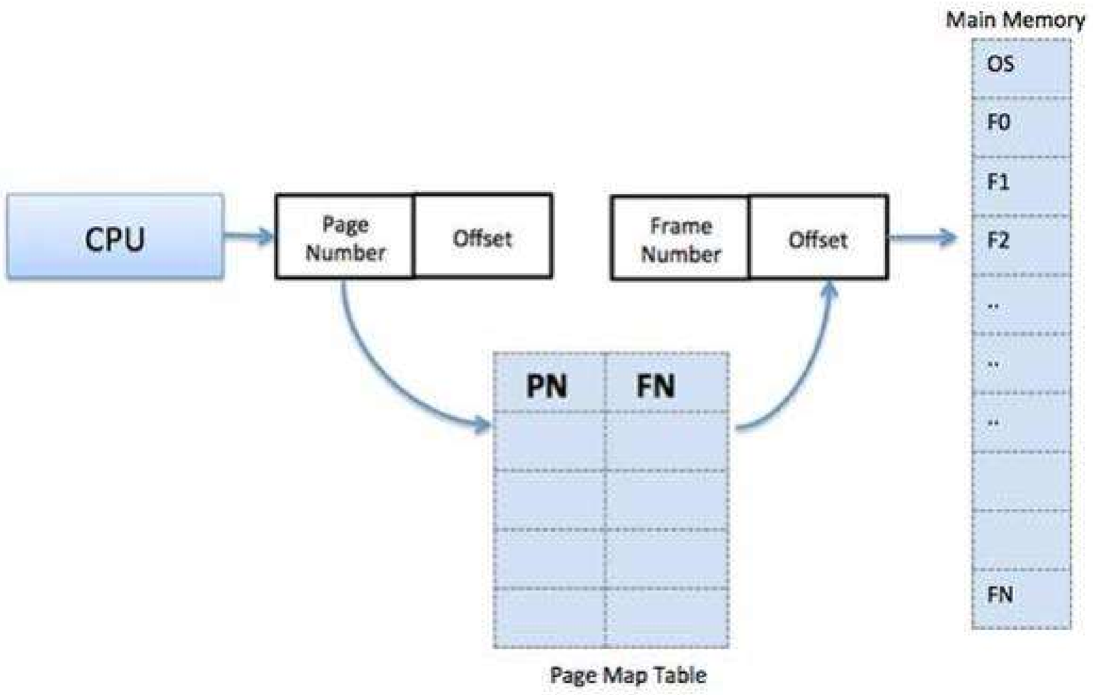


**Address Translation**

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 pages of a process to a frame in physical memory



When the system allocates a frame to any page, it translates this logical address into a physical address and creates entry into the page table to be used throughout execution of the program. When a process is to be executed, its corresponding pages are loaded into any available memory frames.

Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program. This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

**Advantages:**

→ It is independent of external fragmentation.
→ Paging reduces external fragmentation, but still suffers from internal fragmentation.
→ Paging is simple to implement and assumed as an efficient memory management technique.

**Disadvantages:**

→ It makes the translation very slow as main memory access two times.
→ A page table is a burden over the system which occupies considerable space.
→ Due to equal size of the pages and frames, swapping becomes very easy.
→ Page table requires extra memory space, so may not be good for a system having small RAM.

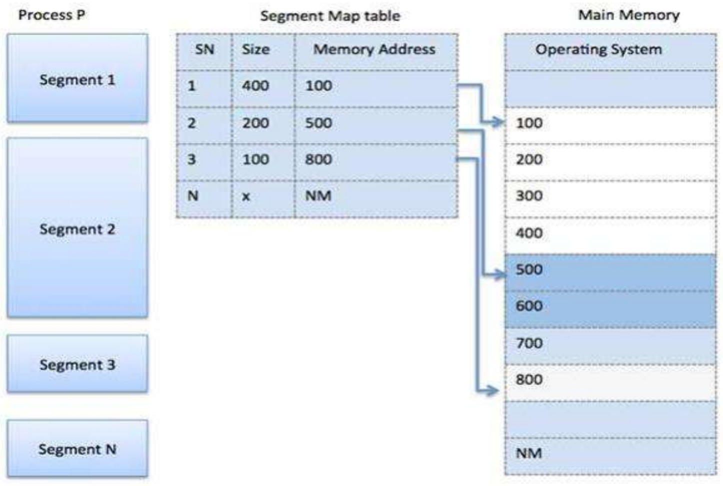
## (II) SEGMENTATION

Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation. Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.

*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 is 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.

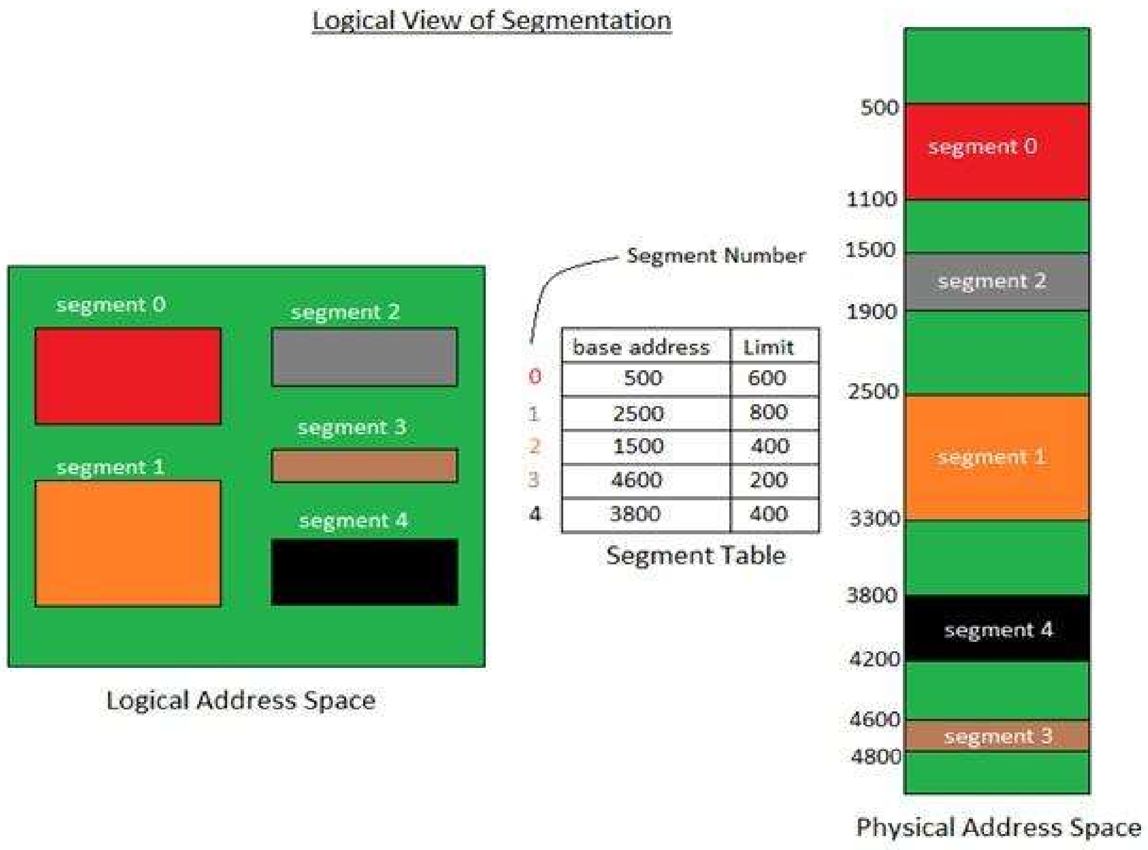
The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference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



| SN | Size | Memory Address |
|---|---|---|
| 1 | 400 | 100 |
| 2 | 200 | 500 |
| 3 | 100 | 800 |
| N | x | NM |

## (III) SEGMENTATION WITH PAGING

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages.



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