Unit 4- Memory Management

Requirements of Memory Management System

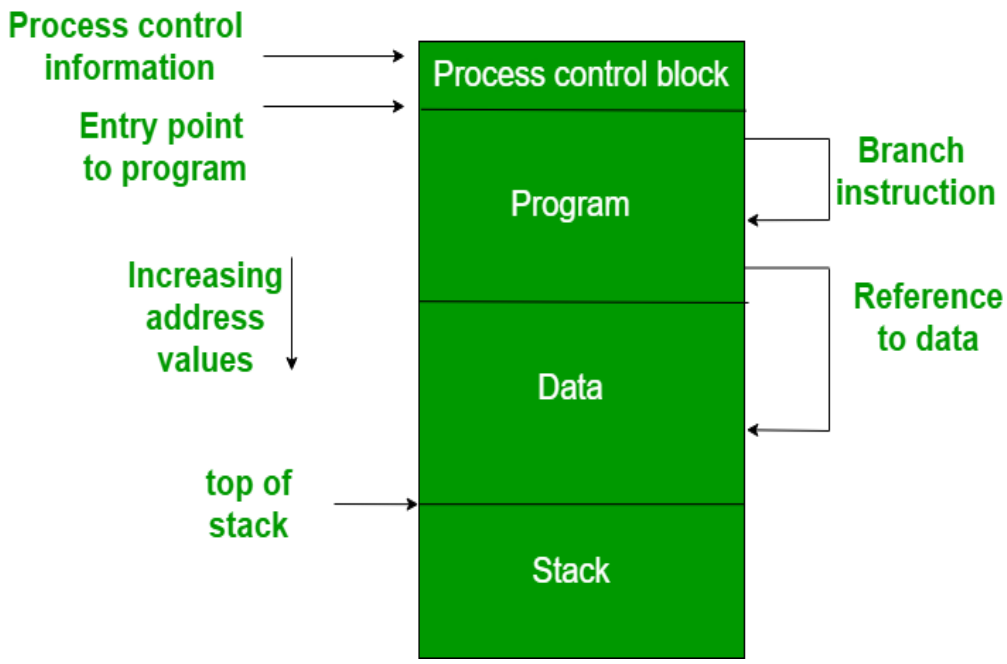
Memory management keeps track of the status of each memory location, whether it is allocated or free. It **allocates** the memory dynamically to the programs at their request and **frees** it for reuse when **it is no longer needed**. Memory management meant to satisfy some requirements that we should keep in mind.

These **Requirements of memory management** are:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of this program.

   Swapping the active processes **in and out** of the main memory enables the operating system to have a larger **pool** of ready-to-execute process.

   When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the **location of process control information, the execution stack, and the code entry.** Within a program, there are **memory references in various instructions and these are called logical addresses**.

After loading of the program into main memory, the processor and the **operating system must be able to translate logical addresses into physical addresses**. **Branch instructions** contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

2. **Protection** – There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program.
   So every process must **be protected against unwanted interference** when other process tries to write in a process

whether accidental or incidental..

Prediction of the **location of a program in main memory** is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. The memory protection requirement must be satisfied by the processor rather than the operating system because the **operating system can hardly control a process when it occupies the processor**.

3. **Sharing – A protection mechanism must have to allow several processes to access the same portion of main memory.** Allowing each processes access to the same copy of the program.
   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of **Memory management to allow controlled access to the shared areas of memory without compromising the protection**.

4. **Logical organization –** Main memory is organized as **linear** or it can be a **one-dimensional address space** which consists of a **sequence of bytes** or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

1. Modules are **written and compiled independently** and all the references from one module to another module are resolved by `the system **at run time**.
2. Different modules are provided with **different degrees of protection.**
3. There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.

5. **Physical organization –** The structure of computer memory has two levels referred to **as main memory and secondary memory**. **Main memory is relatively very fast and costly** as compared to the secondary memory. **Main memory is volatile**. Thus secondary memory is provided for storage of data on a long-term basis while the **main memory holds currently used programs.**

   The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

   1. The programmer may engage in a practice known as **overlaying** when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is **time-consuming** for the programmer.
   2. In a multiprogramming environment, the programmer does not know how much space will be available **at the time of coding and where that space will be located inside the memory.**

Logical and Physical Address in Operating System
Logical address = virtual address

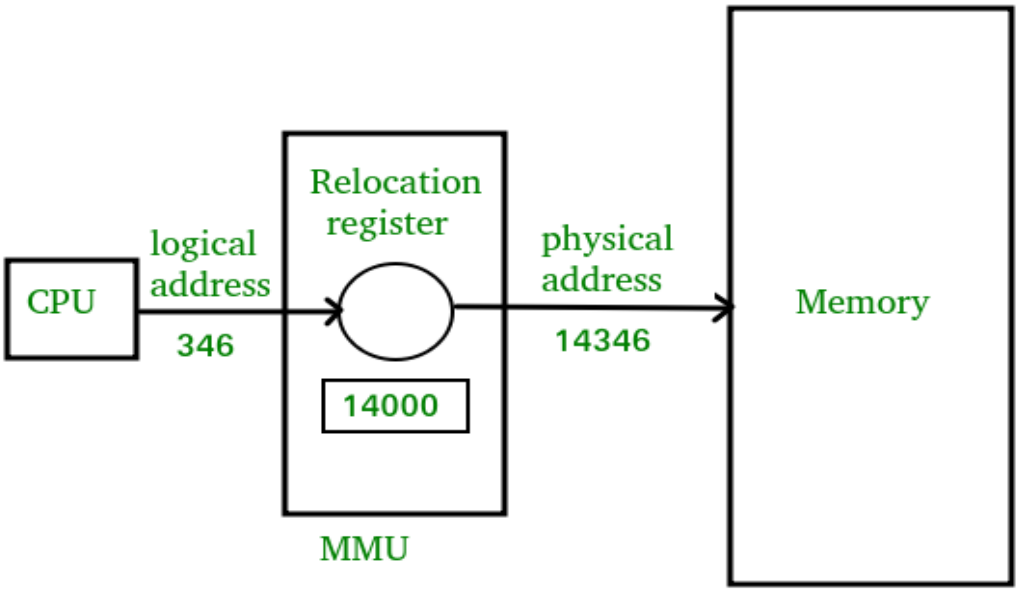
Logical Address is generated **by CPU** while a program is running. The **logical address is virtual address** as it does not exist physically, therefore, it is also known as **Virtual Address.**

This address is **used as a reference** to access the physical memory location by CPU.

The hardware device called **Memory-Management Unit**(MMU)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.



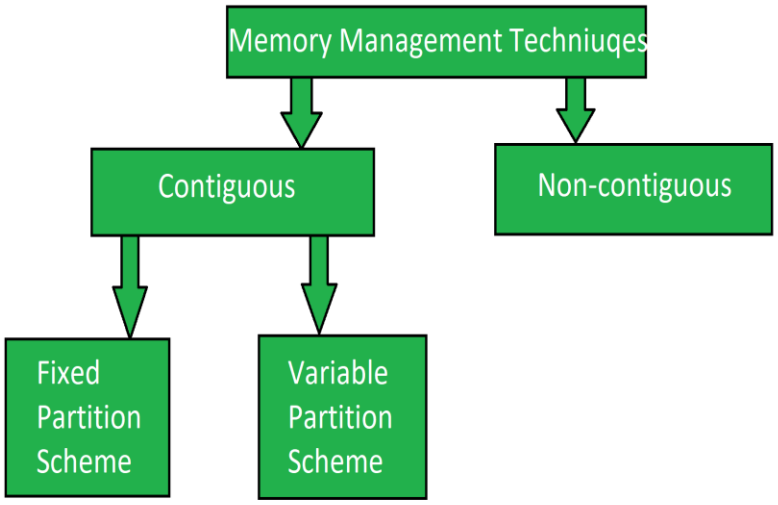
Memory Allocation Techniques:
To store the data and to manage the processes, we need a large-sized memory and, at the same time, we need to access the data as fast as possible. But **if we increase the size of memory, the access time will also increase** and, as we know, the CPU always generates addresses for secondary memory, i.e. logical addresses.

But we want to access the main memory, so we need **Address translation of logical address into physical address**.
The main memory interacts with both the user processes and the operating system.So we need to efficiently use the main memory.

Main memory is divided into non-overlapping memory regions called partitions.

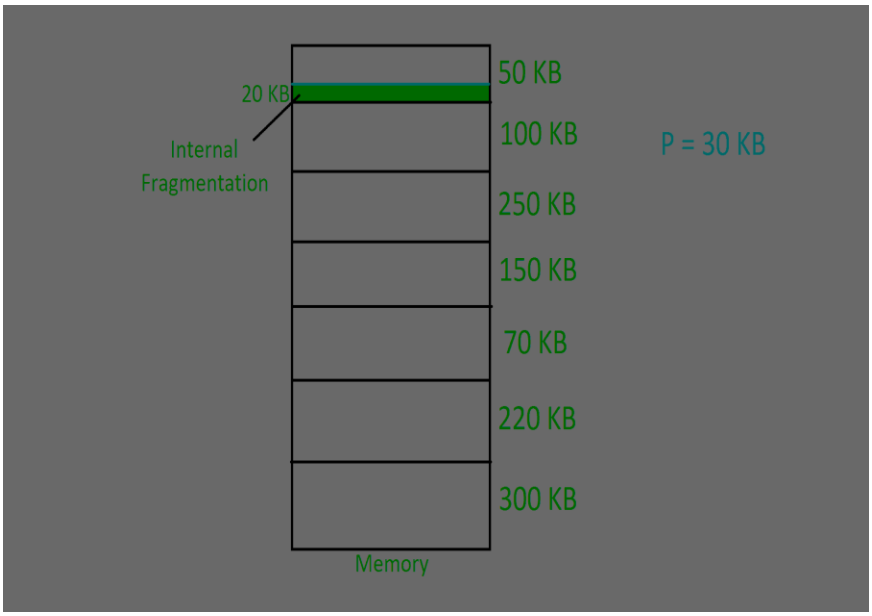
The main memory can be broadly allocated in two ways –
1. Contiguous memory allocation
2. Non-Contiguous memory allocation

Contiguous Memory Management Techniques:
In this technique, **memory is allotted in a continuous way to the processes.** It has two types:

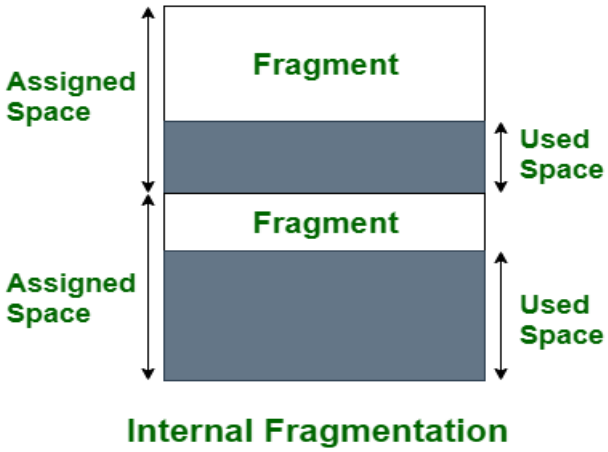


Fixed Partition Scheme**:**
- Memory is divided into fixed number of partitions.
-In every partition only one process will be accommodated.
- **Drawback** : Maximum size of the process is restricted by maximum size of the partition.
-Every partition is associated with the *limit registers*.

- **Limit Registers:** It has two limit:
- **Lower Limit:** Starting address of the partition.
- **Upper Limit:** Ending address of the partition.

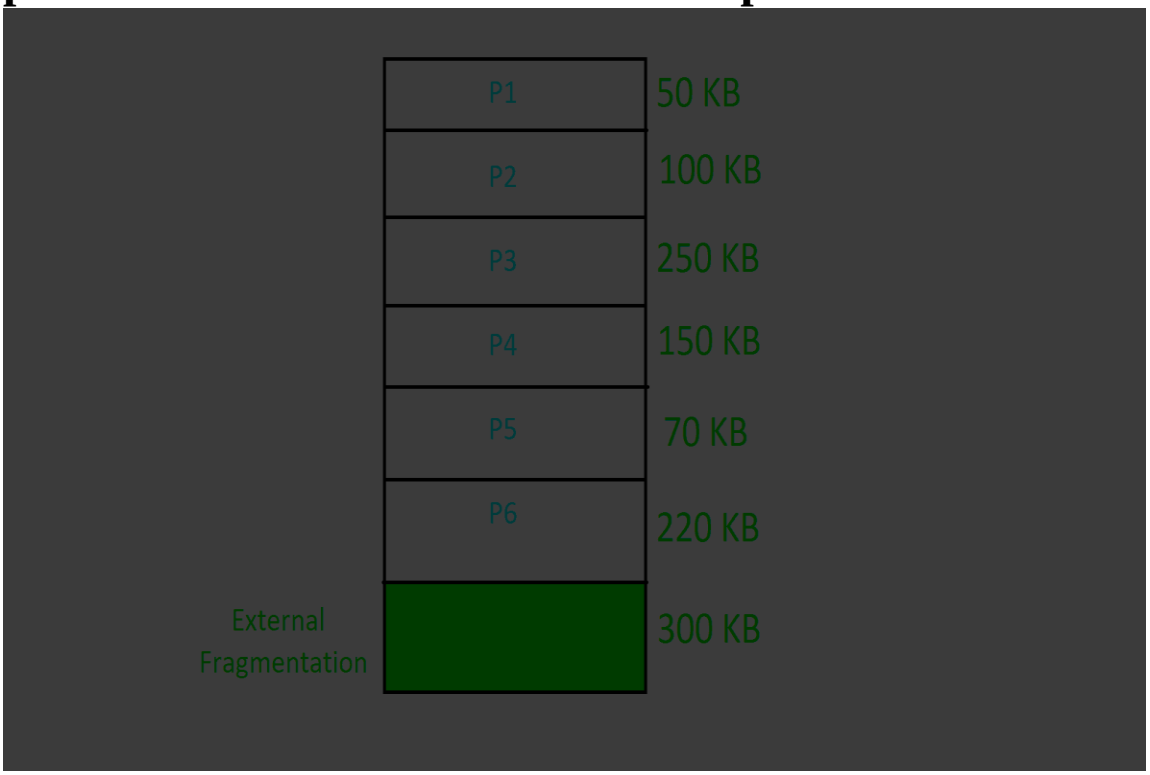


- **<u>Internal Fragmentation</u> is found in fixed partition scheme.**
  To overcome the problem of internal fragmentation, instead of fixed partition scheme, **variable partition scheme is used.**
-

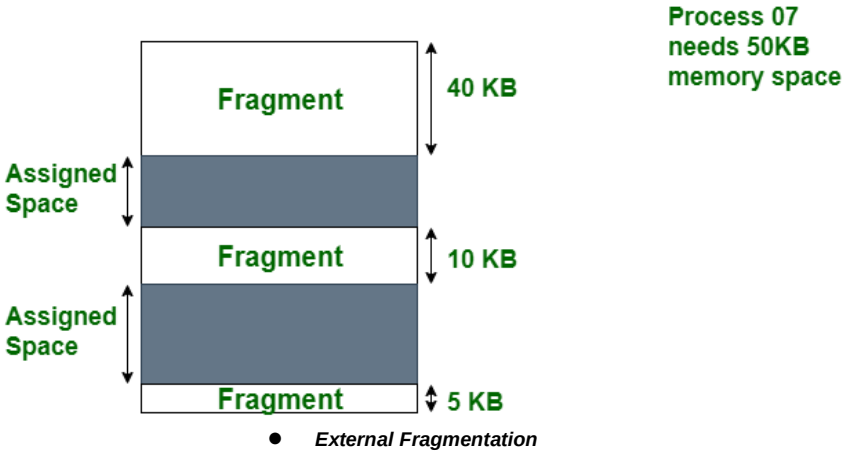


Internal Fragmentation

- Variable Partition Scheme **(External Fragmentation):** In the variable partition scheme, initially memory will be **single continuous free block**. Whenever the request by the process arrives, accordingly partition will be made in the memory.
- **If the smaller processes keep on coming then the larger partitions will be made into smaller partitions**.



- External Fragmentation (variable partition)
  To overcome the problem of external fragmentation, **non-contiguous memory management** techniques are used.



- *External Fragmentation*

- In the above diagram, we can see that, there is enough space (55 KB) to run a process-07 (required 50 KB) but the memory (fragment) is not contiguous.

- Here, we use **paging, or segmentation** to use the free space to run a process.

- For both fixed and dynamic memory allocation schemes, the operating system **must keep list of each memory location;which are free and which are busy**. Then as new jobs come into the system, the free partitions must be allocated.

These are **Contiguous** memory allocation techniques

- These partitions may be allocated by 4 ways:
- **1.** First-Fit Memory Allocation
- **2.** Best-Fit Memory Allocation
- **3.** Worst-Fit Memory Allocation
- **4.** Next-Fit Memory Allocation

- First-Fit Memory Allocation**:**
  This method **keeps the free/busy list** of jobs organized by memory location, low-ordered to high-ordered memory. In this method, **first job claims the first available memory with space more than or equal to it's size.** The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	200 K	J1	20 K	Busy	180 K
30457	30 K			Free	30
300875	700 K	J2	200 K	Busy	500 K
809567	50 K	J4	50 K	Busy	None
Total available :	980 K	Total used :	270 K		710 K

- As illustrated above, the system assigns **J1 the nearest partition in the memory**. As a result, there is no partition

with sufficient space is available for J3 and it is placed in the waiting list.

- **Advantages of First-Fit Memory Allocation:**
  It is **fast** in processing. As the processor allocates the **nearest available memory** partition to the job, it is very fast in execution.
- **Disadvantages of First-Fit Memory Allocation:**
  It **wastes a lot of memory**. The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a **lot of memory is wasted and many jobs may not get space in the memory**, and would have to wait for another job to complete.

Best-Fit Memory Allocation:

This method keeps the **free/busy list in order by size – smallest to largest**. In this method, the operating system first searches the **whole of the memory according to the size of the given job** and allocates it to the **closest-fitting free partition** in the memory, making it able **to use memory efficiently**. Here the jobs are in the order from **smallest job to largest job**.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J4	50 K	Busy	None
300875	200 K	J2	200 K	Busy	None
809567	700 K	J3	500 K	Busy	200 K
Total available :	980 K	Total used :	770 K		210 K

As illustrated in above figure, the operating system first search throughout the memory and allocates the **job to the minimum possible memory partition**, making the memory allocation efficient.

Advantages of Best-Fit Allocation:

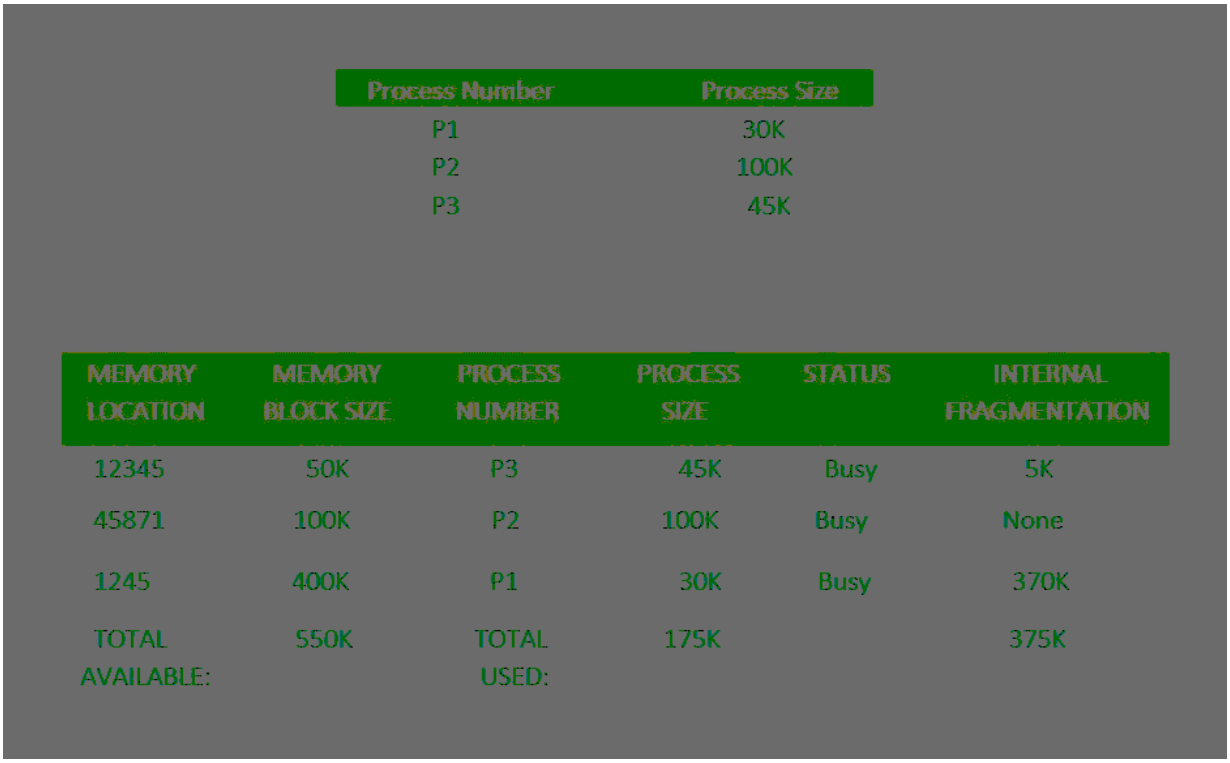
Memory Efficient. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To **save memory from getting wasted**, it is the best method.

Disadvantages of Best-Fit Allocation:

It is a **Slow Process**. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

Worst-Fit Memory Allocation :
In this allocation technique, the process **traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition.** It is a slow process because it has to traverse the entire memory to search the largest hole.

Here is an example to understand Worst Fit-Allocation –



Process Number	Process Size
P1	30K
P2	100K
P3	45K

MEMORY LOCATION	MEMORY BLOCK SIZE	PROCESS NUMBER	PROCESS SIZE	STATUS	INTERNAL FRAGMENTATION
12345	50K	P3	45K	Busy	5K
45871	100K	P2	100K	Busy	None
1245	400K	P1	30K	Busy	370K
TOTAL AVAILABLE:	550K	TOTAL USED:	175K		375K

Here Process P1=30K is allocated with the Worst Fit-Allocation technique, so it traverses the entire memory and selects memory size 400K which is the largest, and hence there is an internal fragmentation of 370K which is very large and so many other processes can also utilize this leftover space.

Advantages of Worst-Fit Allocation:
Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, **this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.**

Disadvantages of Worst-Fit Allocation:
It is a **slow process** because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a **time-consuming process.**

What is Next Fit?

The next fit is a modified version of 'first fit'. It begins as the first fit to find a free partition but when called next time it starts searching from where it left off, not from the beginning. This policy makes use of a roving pointer. The pointer moves along the memory chain to search for the next fit. This helps in, to avoid the usage of memory always from the head (beginning) of the free blockchain.

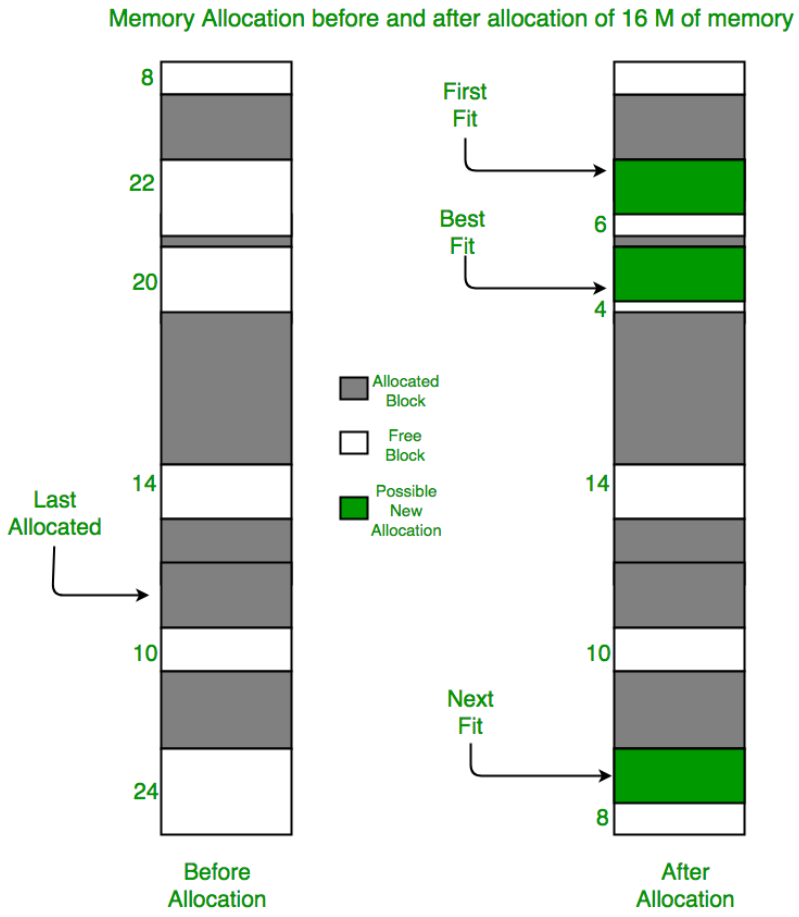
What is its advantage over first fit?

- First fit is a straight and fast algorithm, but tends to cut a large portion of free parts into small pieces due to which, processes that need a large portion of memory block would not get anything even if the sum of all small pieces is greater than it required which is so-called external fragmentation problem.
- Another problem of the first fit is that it tends to allocate memory parts at the beginning of the memory, which may lead to more internal fragments at the beginning. Next fit tries to address this problem by starting the search for the free portion of parts not from the start of the memory, but from where it ends last time.

- Next fit is a very fast searching algorithm and is also comparatively faster than First Fit and Best Fit Memory Management Algorithms.

Algorithm:

1. Input the number of memory blocks and their sizes and initializes all the blocks as free.
2. Input the number of processes and their sizes.
3. Start by picking each process and check if it can be assigned to the current block, if yes, allocate the required memory and check for next process but from the block where we left not from starting.
4. If the current block size is smaller then keep checking the further blocks.



How does the Next Fit algorithm help in reducing memory fragmentation?

The Next Fit algorithm helps in reducing memory fragmentation by leaving larger gaps between allocated partitions, which can be used for future allocations. This reduces the number of small gaps and helps in allocating larger contiguous blocks of memory to processes.

Page Replacement Algorithms in Operating Systems

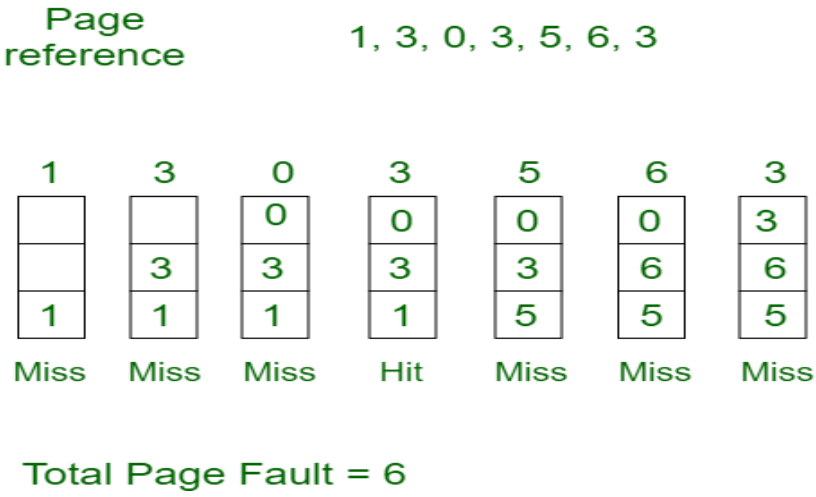
In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

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 a 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.

Page Replacement Algorithms:

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, 3 with 3 page frames.Find the number of page faults.

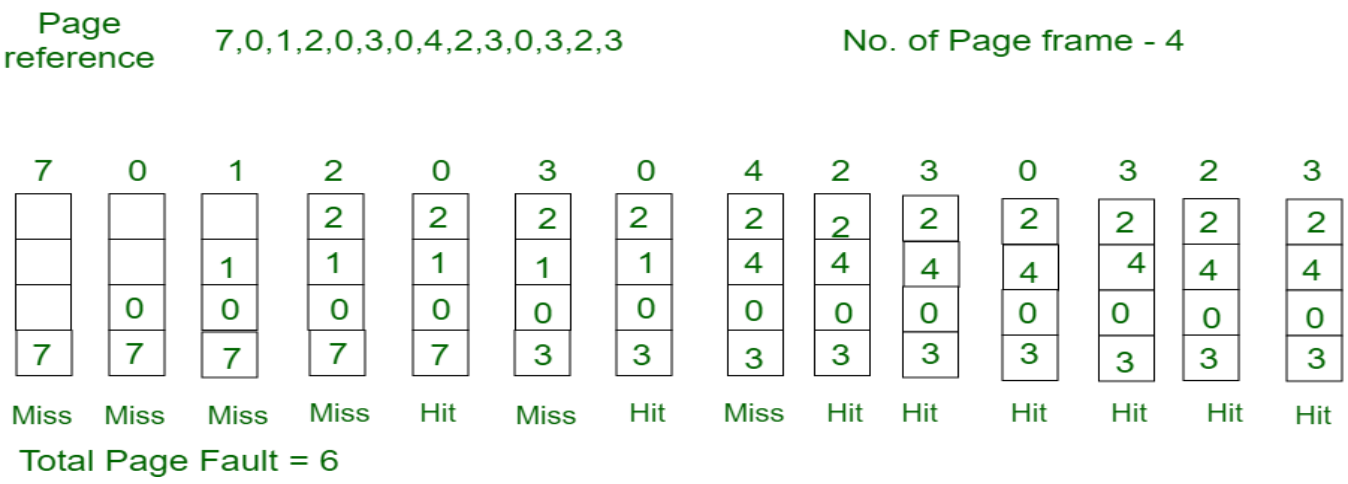


Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.** when 3 comes, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.** Finally, when 3 come it is not available so it replaces 0 **1 page fault.**

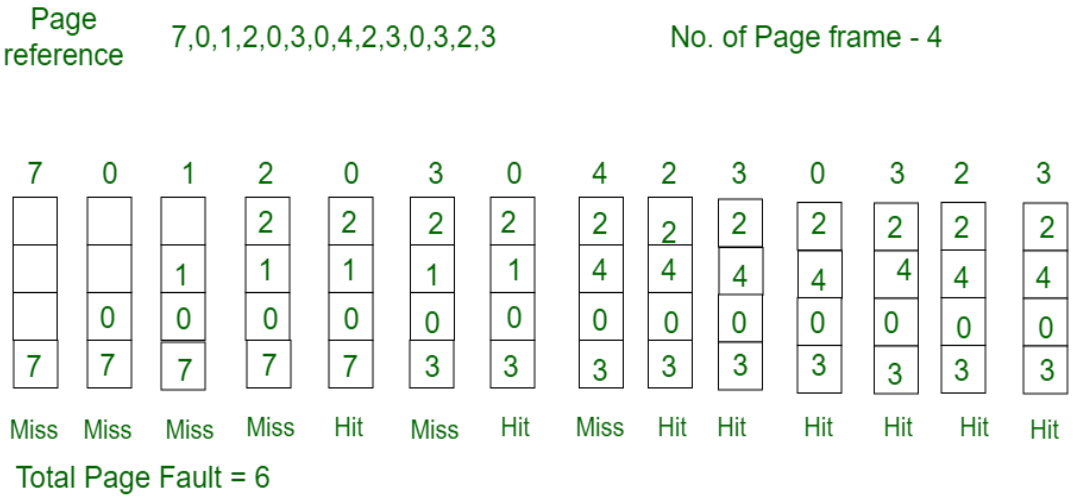
<u>Belady's anomaly</u> proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

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, 3 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.
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, 3 with 4 page frames. Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

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.

1. **Most Recently Used (MRU):** In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

What is the difference between LRU and optimal page replacement?

The LRU page replacement algorithm keeps track of page usage in the memory over a short time period. In contrast, In the LFU page replacement algorithm, the page with the least visits in a given period of time is removed. LRU removes the page that has not been utilized in the memory for the longest period of time.

Virtual Memory in Operating System

Virtual Memory is a storage allocation scheme in which **secondary memory can be addressed as though it were part of the main memory.**The program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is **limited by the addressing scheme of the computer system** and the amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

2. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of

the main memory such that it occupies different places in the main memory at different times during the course of execution.

3. A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

Demand Paging :

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps :

1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to precede the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The 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.
5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Advantages:

- More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- A process may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

Page Fault Service Time:

The time taken to service the page fault is called page fault service time. The page fault service time includes the time taken to perform all the above six steps.

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is: p

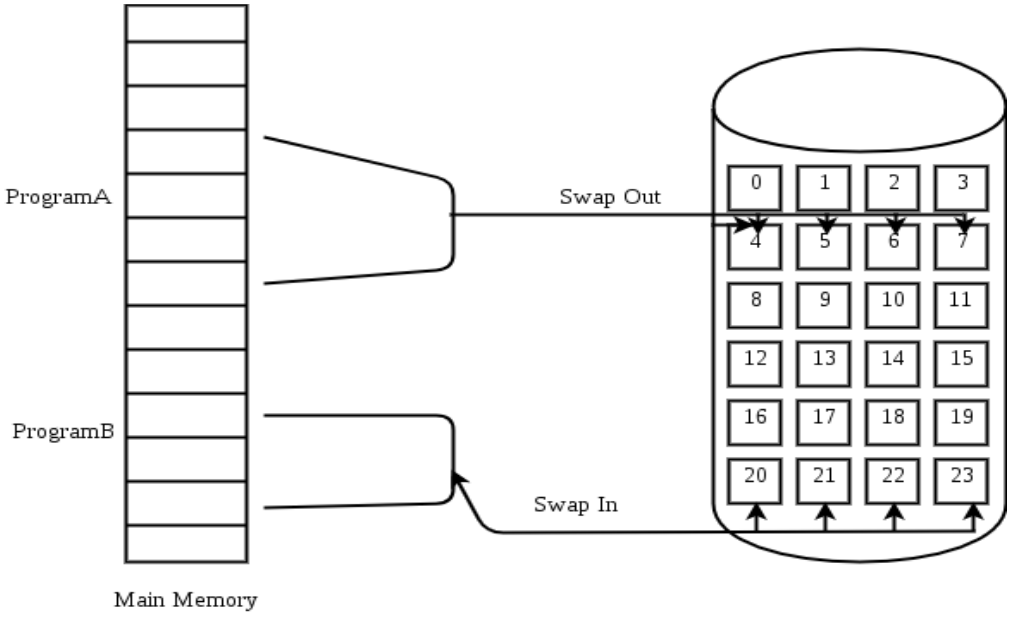
Then, Effective memory access time = (p*s) + (1-p)*m

Swapping:

Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the

normal page replacement process. Suspending a process ensures that it is not runnable while it is swapped out.

At some later time, the system swaps back the process from the secondary storage to the main memory. When a process is busy swapping pages in and out then this situation is called **thrashing**.



GATE CS Corner Questions

Practicing the following questions will help you test your knowledge. All questions have been asked in GATE in previous years or in GATE Mock Tests. It is highly recommended that you practice them.

1. Memory Management | Question 1
2. Memory Management | Question 10
3. GATE CS 2014 (Set-1), Question 65
4. GATE CS 2012, Question 40
5. GATE CS 2007, Question 56
6. GATE CS 2007, Question 82
7. GATE CS 2007, Question 83
8. GATE CS 2014 (Set-3), Question 65
9. GATE CS 2002 Question 23

10. GATE CS 2001, Question 21
11. GATE CS 2010, Question 24

Segmentation:

Segmentation

In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

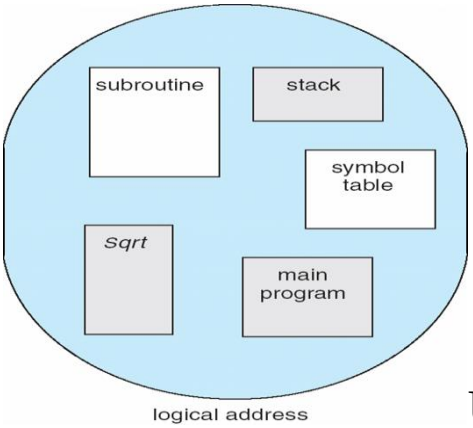
The **details about each segment are stored in a table called a segment table.** Segment table contains mainly two information about segment:

Base: It is the base address of the segment

Limit: It is the length of the segment.

Why Segmentation is required?

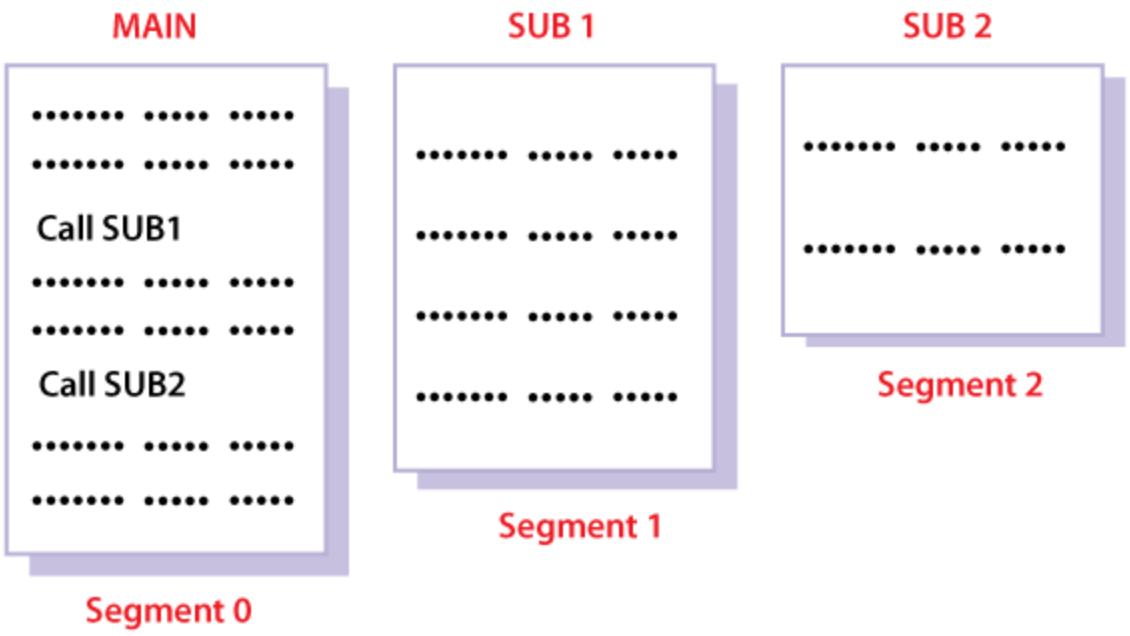
Till now, we were using Paging as our main memory management technique. Paging is more close to the Operating system rather than the User. It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of functions which need to be loaded in the same page.



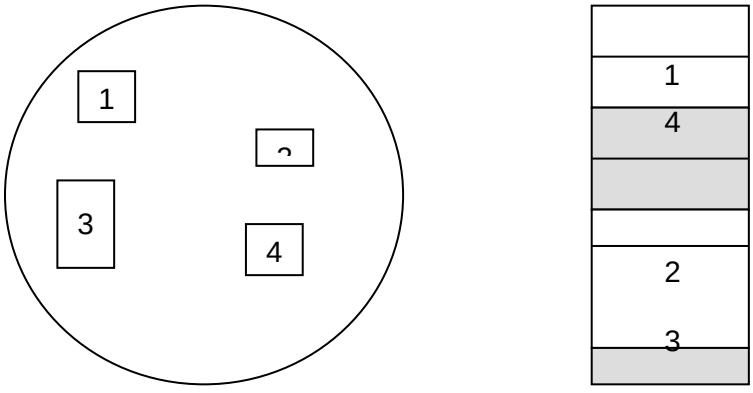
Users View of program:

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. **It decreases the efficiency of the system**.

It is **better to have segmentation which divides the process into the segments**. **Each segment contains the same type of functions** such as the **main function can be included in one segment and the library functions can be included in the other segment.**



Logical View of Segmentation



Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

Segment Number

Offset

For Example:

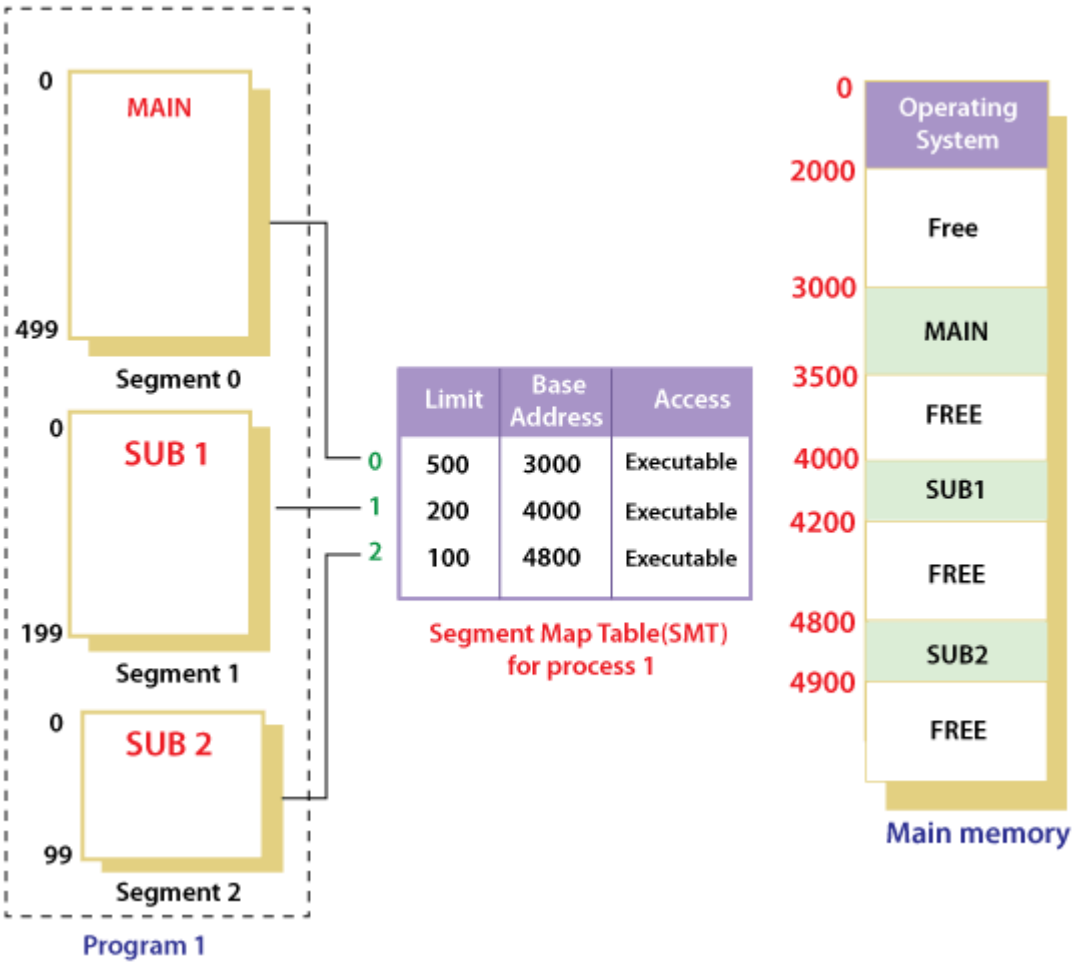
Suppose a 16 bit address is used with 4 bits for the segment number and 12 bits for the segment offset so the

maximum segment size is 2^12 = 4096 and

maximum number of segments that can be refereed is 2 ^ 4 =16.

When a program is loaded into memory, the segmentation system tries to locate space that is large enough to hold the first segment of the process, space information is obtained from the free list maintained by memory manager. Then it tries to locate space for other segments. Once adequate space is located for all the segments, it loads them into their respective areas.

The operating system also generates a segment map table for each program.



With the help of segment map tables and **hardware assistance,** the operating system can easily translate a logical address into physical address on execution of a program.

The **Segment number** is mapped to the segment table.

Ex:

Seg 1 is 200 bytes long so to refer the 59^{th} byte from seg 1 we have to mapped the location $4000+59^{th}$ =4059.

The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid.

In the case of valid addresses, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.

The above figure shows how address translation is done in case of segmentation.

Advantages of Segmentation

No internal fragmentation

Average Segment Size is larger than the actual page size.

Less overhead

It is easier to relocate segments than entire address space.

The segment table is of lesser size as compared to the page table in paging.

Disadvantages

It can have external fragmentation.

it is difficult to allocate contiguous memory to variable sized partition.

Costly memory management algorithms.

Paging in Operating System

Paging is **a memory management scheme** that eliminates the need for contiguous allocation of physical memory.

-Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available

- Avoids external fragmentation

- Avoids problem of varying sized memory chunks

- Conceptually Divide **physical memory into fixed-sized blocks** called **frames**

- **Page Size is power of 2**, between 512 bytes and 1 GB per page

- Divide **logical memory into blocks of same size** called **pages**

- Keep track of all free frames

- To run a program of size ***N***pages, need to find ***N*** free frames and load program

- Set up a **page table** to translate logical to physical addresses

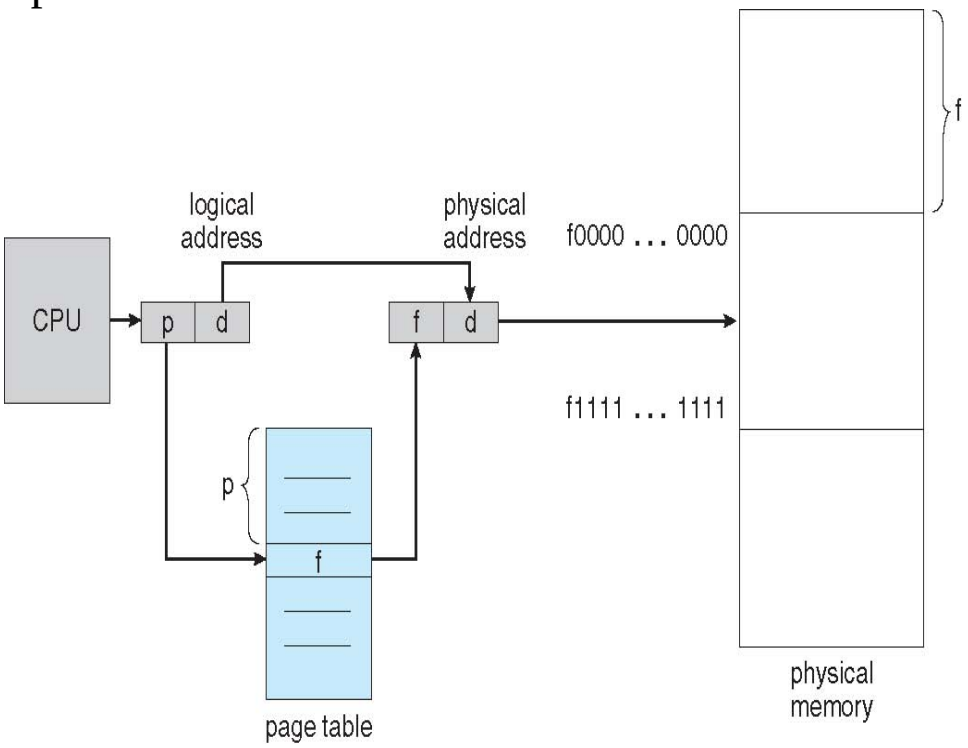
-Still have internal fragmentation

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**.

Address generated by CPU is divided into:

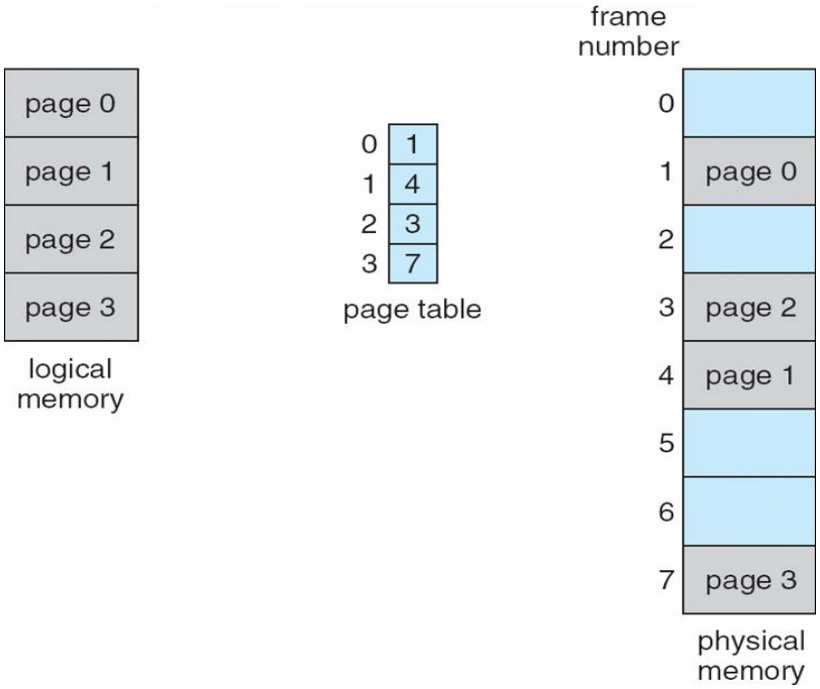
- **Page number** (***p***) – used as an **index into a page table** which contains base address of each page in physical memory
- **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit

Physical Address is divided into

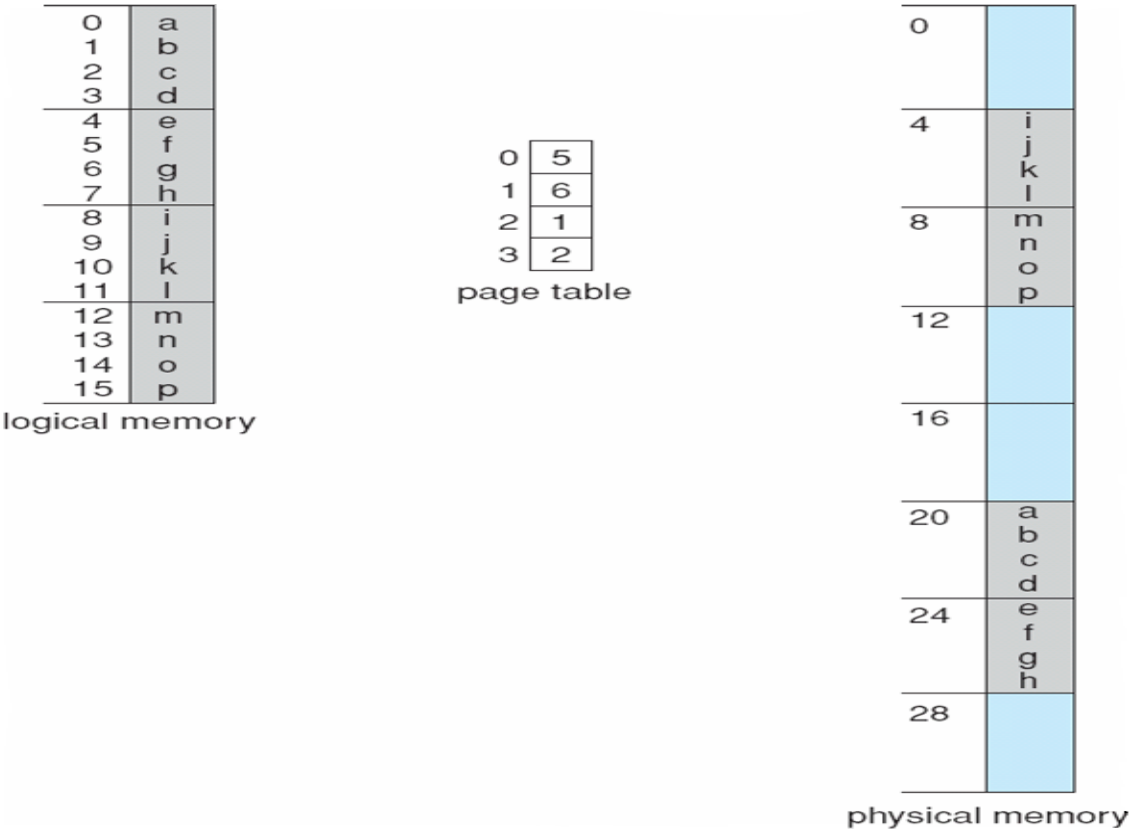
- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or **Frame number**.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame **or frame offset.**
- 

P=page number in page table

Page table contains the base address of each page in physical memory. This page address is combined with the page offset to define the physical memory address sent to the memory unit.



Paging Model of Logical and Physical Memory



Paging Example : *page size is n bytes* =2 i.e 2^2 = 4 bytes

Size of the logocal address space is m=4 i.e. 2^4

32-byte memory and 4-byte pages

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.

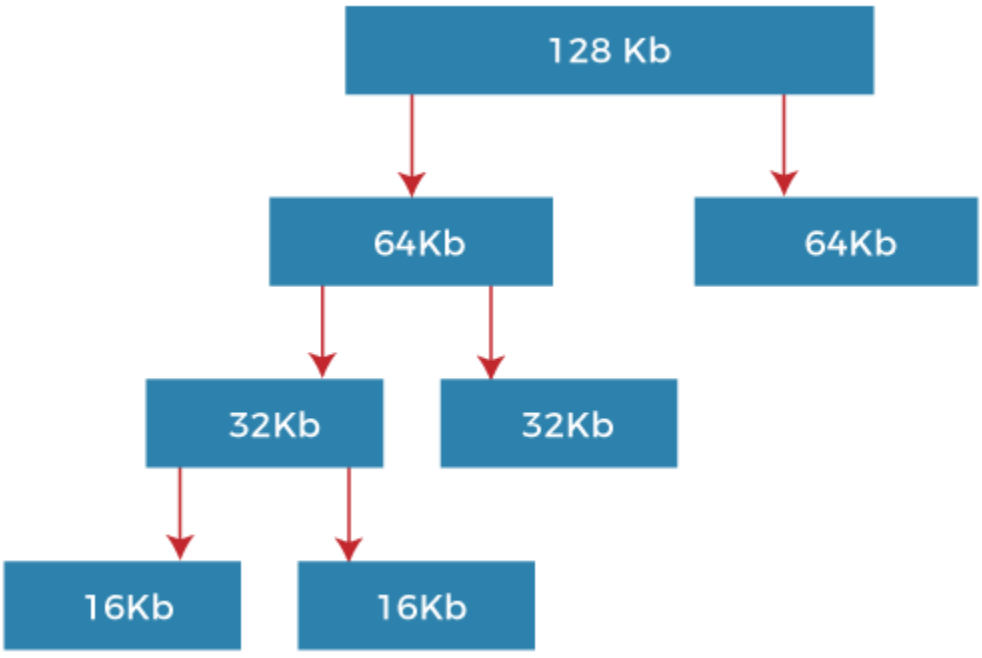
- **What is Buddy System?**
- The **two smaller parts of the block are of equal size** and called buddies.
- The buddy system **is a procedure in which two individual buddies operate together as a single unit so that they can monitor and help each other.**
- Similarly, one of the two buddies will further divide into smaller parts until the request is fulfilled.
- ***Merriam-Webster*** is the first known use of the phrase **buddy system in 1942**. According to Webster buddy system is an arrangement in which **two individuals are paired** (as for mutual safety in a hazardous situation).
- The buddy system is basically working together in pairs in a large group or alone. Both the individuals have to do the job.

The job could **ensure that the work is finished safely** or **transferred effectively** from one individual to the other.

- **Types of Buddy System**

1. Binary Buddy System

- The buddy system **maintains a list of the free blocks of each size** (called a free list) so that it is easy to find a block of the desired size if one is available. If no block of the requested size is available, Allocate searches for the first non-empty list for blocks of atleast the size requested. In either case, a block is removed from the free list.
- For example, suppose the size of the memory segment is initially 256kb, and the kernel requests 25kb of memory. The segment is initially divided into two buddies. Let's say A1 and A2, each 128kb in size. One of these buddies is further divided into two 64kb buddies, B1 and B2. But the next highest power of 25kb is 32kb so, either B1 or B2 is further divided into two 32kb buddies (C1 and C2), and finally, one of these buddies is used to satisfy the 25kb request. A split block can only be merged with its unique buddy block, which then reforms the larger block they were split from.



Weighted Buddy System

The weighted buddy system is similar to the original buddy system. **The large blocks are split iteratively to provide the desired smaller blocks in this system**. When blocks are released, they are combined with their buddy if the buddy is available or, failing this, are attached to an available space list. The binary and weighted buddy system has the following similarities.

- The ease of calculating the address of the buddy of a block and giving the block's address. The address calculation for the binary and weighted buddy systems is straight forward.
- And the allocation of blocks from an available space list.