UNIT NO: - 4
MEMORY MANAGEMENT

WHAT IS MEMORY?

Computer memory can be defined as a collection of some data represented in the binary format. On the basis of various functions, memory can be classified into various categories. We will discuss each one of them later in detail.

A computer device that is capable to store any information or data temporally or permanently is called storage device.

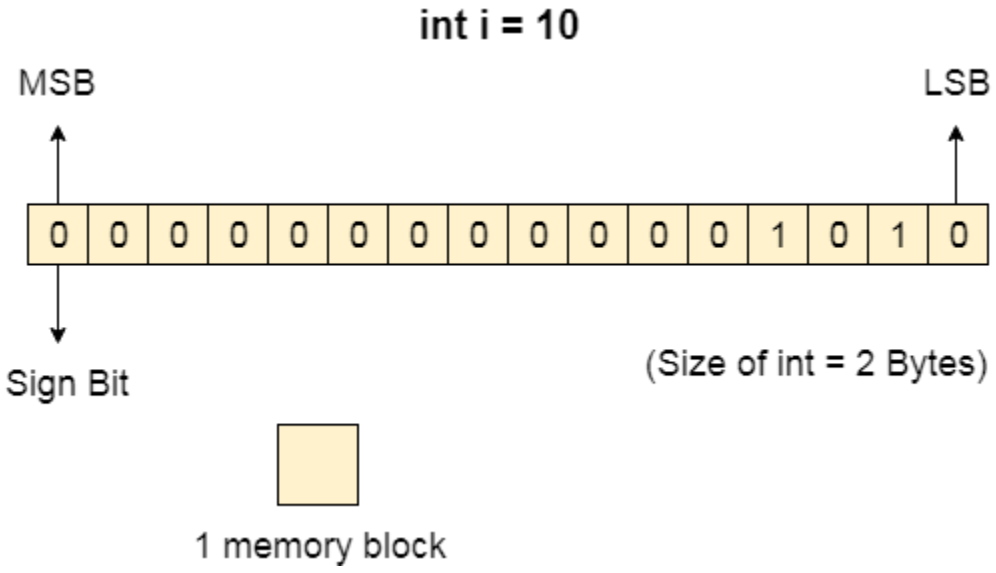
HOW DATA IS BEING STORED IN A COMPUTER SYSTEM?

In order to understand memory management, we have to make everything clear about how data is being stored in a computer system.

Machine understands only binary language that is 0 or 1. Computer converts every data into binary language first and then stores it into the memory.

That means if we have a program line written as **int α = 10** then the computer converts it into the binary language and then store it into the memory blocks.

The representation of **inti = 10** is shown below.



The binary representation of 10 is 1010. Here, we are considering 32 bit system therefore, the size of int is 2 bytes i.e. 16 bit. 1 memory block stores 1 bit. If we are using signed integer then the most significant bit in the memory array is always a signed bit.

Signed bit value 0 represents positive integer while 1 represents negative integer. Here, the range of values that can be stored using the memory array is -32768 to +32767.

well, we can enlarge this range by using unsigned int. in that case, the bit which is now storing the sign will also store the bit value and therefore the range will be 0 to 65,535.

NEED FOR MULTI PROGRAMMING

However, The CPU can directly access the main memory, Registers and cache of the system. The program always executes in main memory. The size of main memory affects degree of Multi programming to most of the extant. If the size of the main memory is larger than CPU can load more processes in the main memory at the same time and therefore will increase degree of Multi programming as well as CPU utilization.

1. Let's consider,
2. Process Size = 4 MB
3. Main memory size = 4 MB
4. The process can only reside in the main memory at any time.
5. If the time for which the process does IO is P,
6. Then,
7. CPU utilization = (1-P)
8. let's say,
9. P = 70%
10. CPU utilization = 30 %
11. Now, increase the memory size, Let's say it is 8 MB.
12. Process Size = 4 MB
13. Two processes can reside in the main memory at the same time.
14. Let's say the time for which, one process does its IO is P,
15. Then
16. CPU utilization = (1-P^2)
17. let's say P = 70 %
18. CPU utilization = (1-0.49) =0.51 = 51 %

Therefore, we can state that the CPU utilization will be increased if the memory size gets increased.

FIXED PARTITIONING

The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

1. Internal Fragmentation

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

2. External Fragmentation

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.

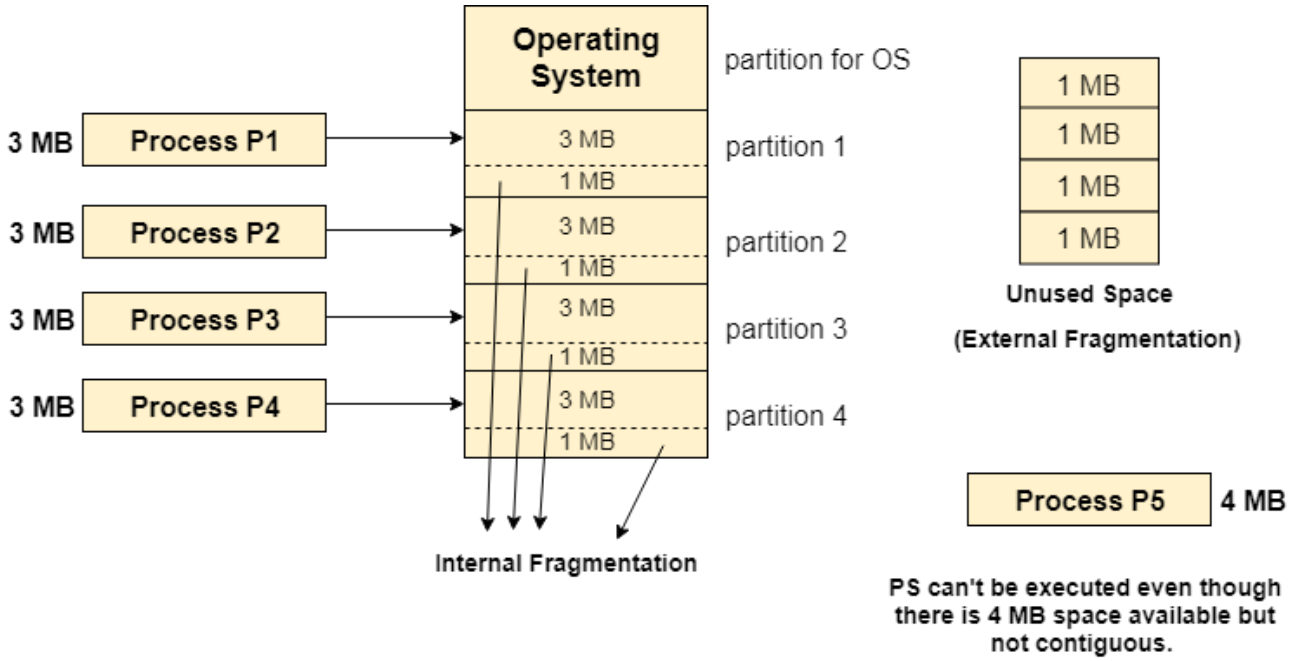
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

3. Limitation on the size of the process

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Degree of multiprogramming is less

By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.



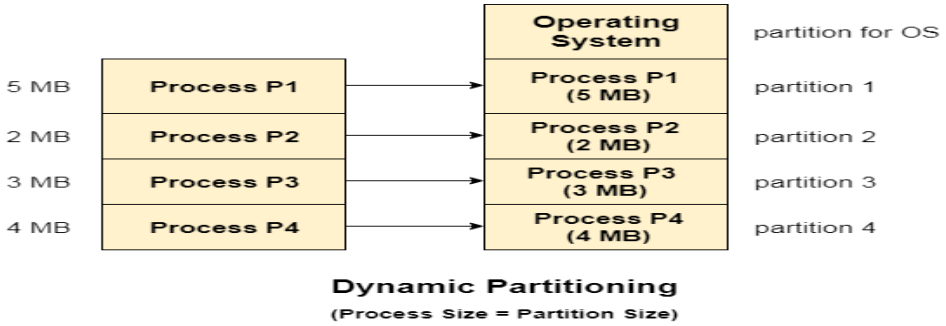
Fixed Partitioning

(Contiguous memory allocation)

DYNAMIC PARTITIONING

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.



Dynamic Partitioning

(Process Size = Partition Size)

Advantages of Dynamic Partitioning over fixed partitioning

1. No Internal Fragmentation

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

2. No Limitation on the size of the process

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

3. Degree of multiprogramming is dynamic

Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

Disadvantages of dynamic partitioning

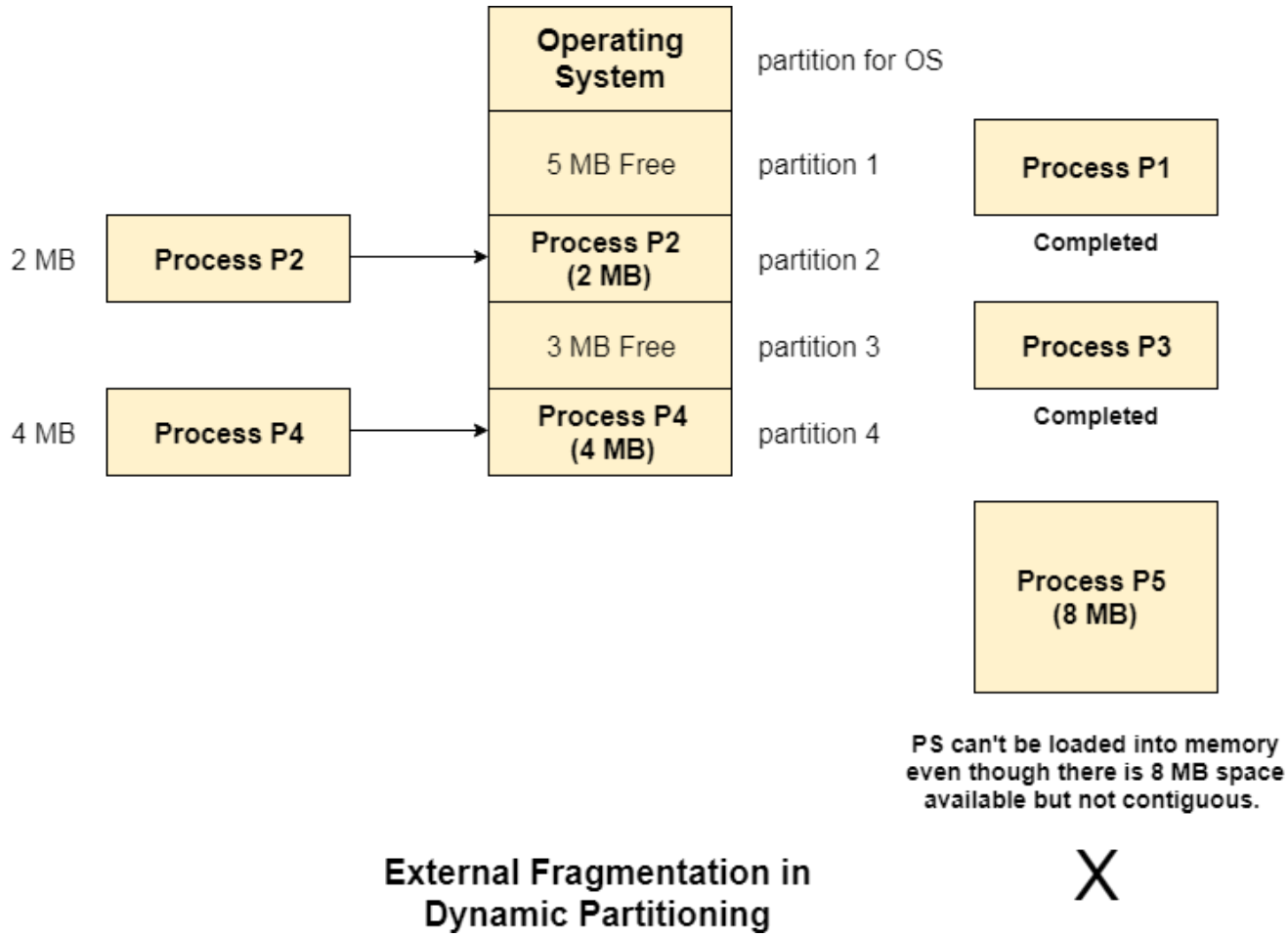
External Fragmentation

Absence of internal fragmentation doesn't mean that there will not be external fragmentation.

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.

After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



Complex Memory Allocation

In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.

Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

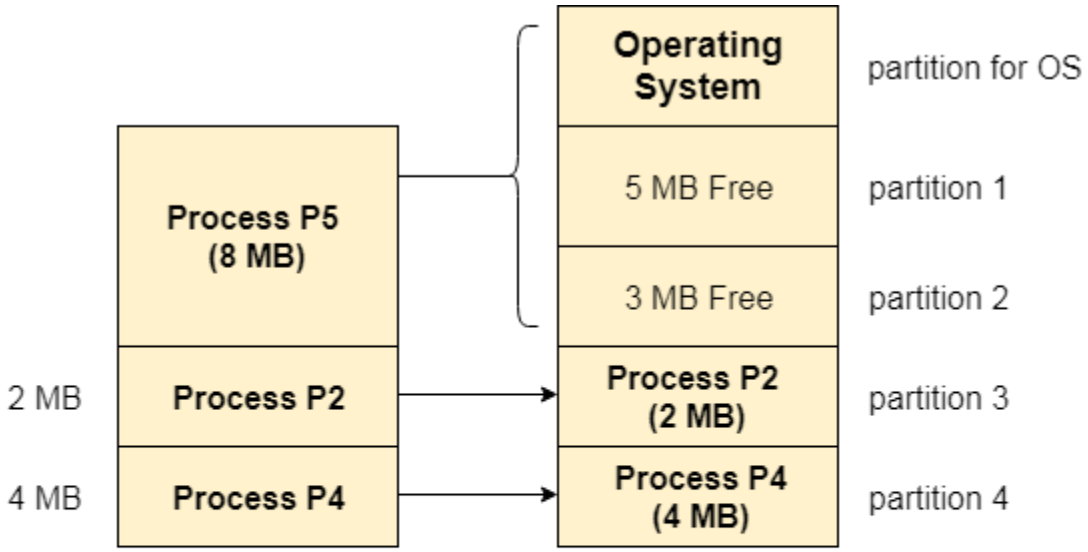
Compaction

We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.

To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.



Compaction

As shown in the image above, the process P5, which could not be loaded into the memory due to the lack of contiguous space, can be loaded now in the memory since the free partitions are made contiguous.

Problem with Compaction

The efficiency of the system is decreased in the case of compaction due to the fact that all the free spaces will be transferred from several places to a single place.

Huge amount of time is invested for this procedure and the CPU will remain idle for all this time. Despite of the fact that the compaction avoids external fragmentation, it makes system inefficient.

Let us consider that OS needs 6 NS to copy 1 byte from one place to another.

1. 1 B transfer needs 6 NS
2. 256 MB transfer needs 256 X 2^20 X 6 X 10 ^ -9 secs

hence, it is proved to some extent that the larger size memory transfer needs some huge amount of time that is in seconds.

Bit Map for Dynamic Partitioning

The Main concern for dynamic partitioning is keeping track of all the free and allocated partitions. However, the Operating system uses following data structures for this task.

1. Bit Map
2. Linked List

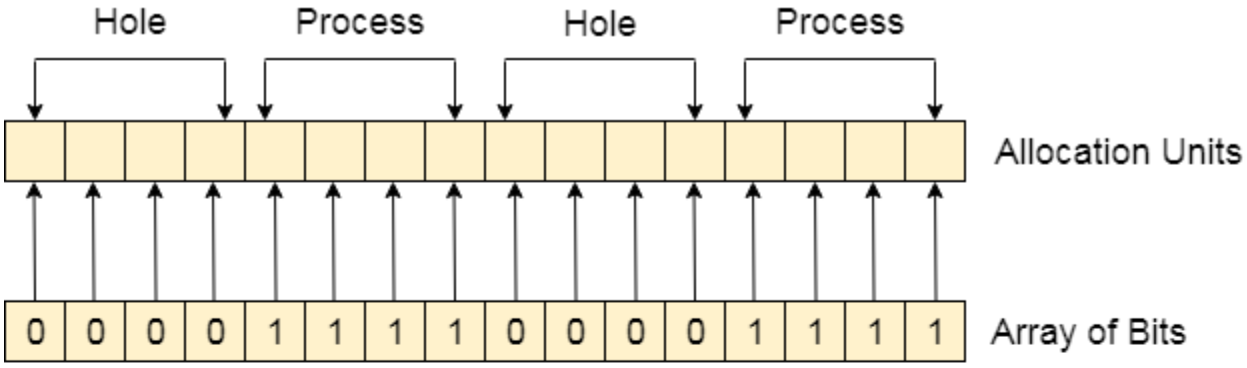
Bit Map is the least famous data structure to store the details. In this scheme, the main memory is divided into the collection of allocation units. One or more allocation units may be allocated to a process according to the need of that process. However, the size of the allocation unit is fixed that is defined by the Operating System and never changed. Although the partition size may vary but the allocation size is fixed.

The main task of the operating system is to keep track of whether the partition is free or filled. For this purpose, the operating system also manages another data structure that is called bitmap.

The process or the hole in Allocation units is represented by a flag bit of bitmap. In the image shown below, a flag bit is defined for every bit of allocation units. However, it is not the general case, it depends on the OS that, for how many bits of the allocation units, it wants to store the flag bit.

The flag bit is set to 1 if there is a contiguously present process at the adjacent bit in allocation unit otherwise it is set to 0.

A string of 0s in the bitmap shows that there is a hole in the relative Allocation unit while the string of 1s represents the process in the relative allocation unit.



1 Allocation Unit = 1/2 byte

1 Bit of Bit Map ⟶ 1 Bit of Allocation Unit

1/5 of the total memory is taken by Bit Map

0 ⟶ Hole

0 ⟶ Process

Bit Map for Dynamic Partitioning

Disadvantages of using Bitmap

1. The OS has to assign some memory for bitmap as well since it stores the details about allocation units. That much amount of memory cannot be used to load any process therefore that decreases the degree of multiprogramming as well as throughput.

In the above image,

The allocation unit is of 4 bits that is 0.5 bits. Here, 1 bit of the bitmap is representing 1 bit of allocation unit.

1. Size of 1 allocation unit = 4 bits
2. Size of bitmap = 1/(4+1) = 1/5 of total main memory.

Therefore, in this bitmap configuration, 1/5 of total main memory is wasted.

2. To identify any hole in the memory, the OS need to search the string of 0s in the bitmap. This searching takes a huge amount of time which makes the system inefficient to some extent

Linked List for Dynamic Partitioning

The better and the most popular approach to keep track the free or filled partitions is using Linked List.

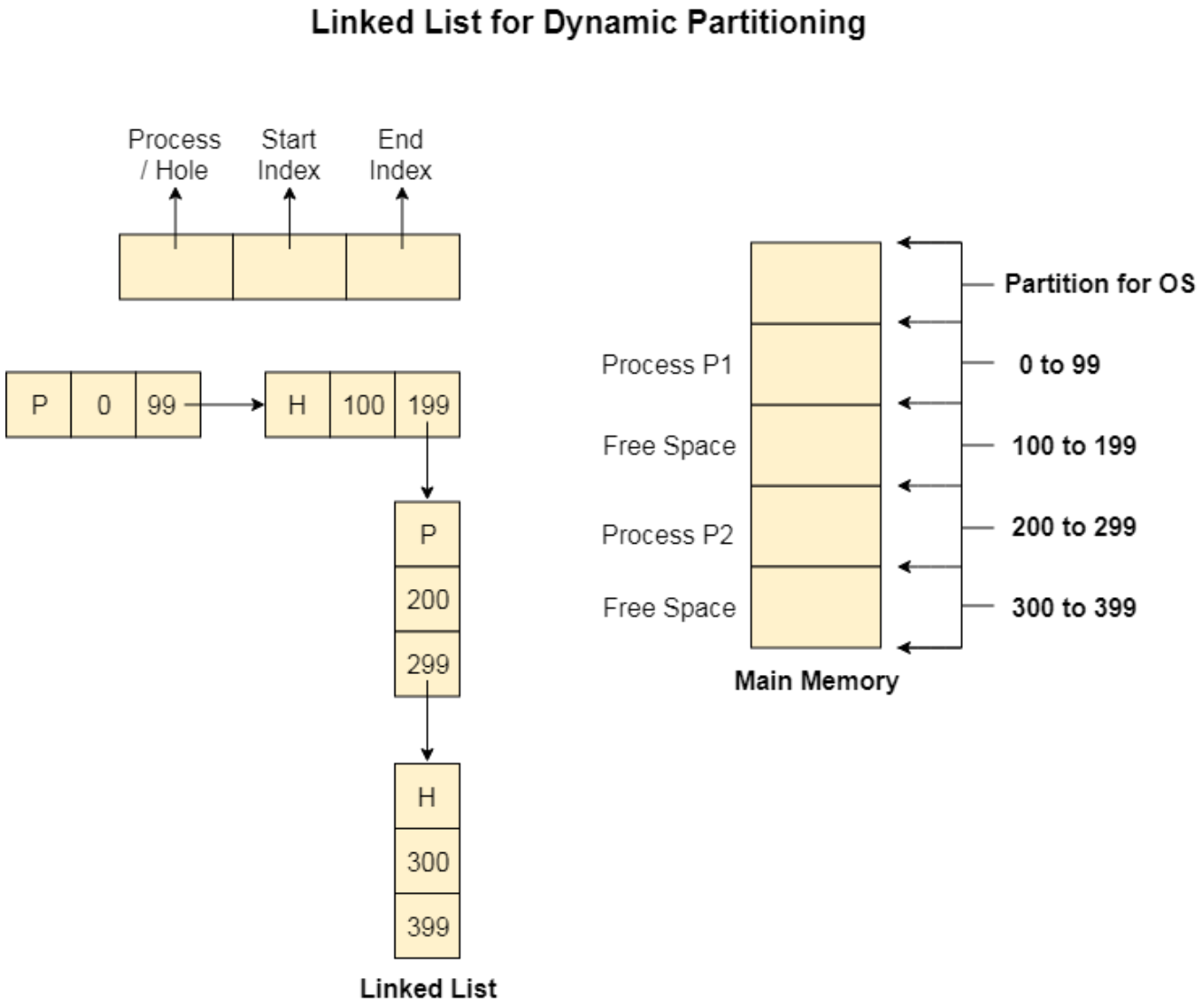
In this approach, the Operating system maintains a linked list where each node represents each partition. Every node has three fields.

1. First field of the node stores a flag bit which shows whether the partition is a hole or some process is inside.
2. Second field stores the starting index of the partition.
3. Third filed stores the end index of the partition.

If a partition is freed at some point of time then that partition will be merged with its adjacent free partition without doing any extra effort.

There are some points which need to be focused while using this approach.

1. The OS must be very clear about the location of the new node which is to be added in the linked list. However, adding the node according to the increasing order of starting index is suggestible.
2. Using a doubly linked list will make some positive effects on the performance due to the fact that a node in the doubly link list can also keep track of its previous node.



PARTITIONING ALGORITHMS

There are various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.

The explanation about each of the algorithm is given below.

1. First Fit Algorithm

First Fit algorithm scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole. This procedure produces two partitions. Out of them, one partition will be a hole while the other partition will store the process.

First Fit algorithm maintains the linked list according to the increasing order of starting index. This is the simplest to implement among all the algorithms and produces bigger holes as compare to the other algorithms.

2. Next Fit Algorithm

Next Fit algorithm is similar to First Fit algorithm except the fact that, Next fit scans the linked list from the node where it previously allocated a hole.

Next fit doesn't scan the whole list, it starts scanning the list from the next node. The idea behind the next fit is the fact that the list has been scanned once therefore the probability of finding the hole is larger in the remaining part of the list.

Experiments over the algorithm have shown that the next fit is not better then the first fit. So it is not being used these days in most of the cases.

3. Best Fit Algorithm

The Best Fit algorithm tries to find out the smallest hole possible in the list that can accommodate the size requirement of the process.

Using Best Fit has some disadvantages.

1. 1. It is slower because it scans the entire list every time and tries to find out the smallest hole which can satisfy the requirement the process.
2. Due to the fact that the difference between the whole size and the process size is very small, the holes produced will be as small as it cannot be used to load any process and therefore it remains useless.
   Despite of the fact that the name of the algorithm is best fit, It is not the best algorithm among all.
3.

4. Worst Fit Algorithm

The worst fit algorithm scans the entire list every time and tries to find out the biggest hole in the list which can fulfill the requirement of the process.

Despite of the fact that this algorithm produces the larger holes to load the other processes, this is not the better approach due to the fact that it is slower because it searches the entire list every time again and again.

5. Quick Fit Algorithm

The quick fit algorithm suggests maintaining the different lists of frequently used sizes. Although, it is not practically suggestible because the procedure takes so much time to create the different lists and then expending the holes to load a process.

The first fit algorithm is **the best algorithm** among all because

1. It takes lesser time compare to the other algorithms.
2. It produces bigger holes that can be used to load other processes later on.
3. It is easiest to implement.

Disadvantage of Dynamic Partitioning

The main disadvantage of Dynamic Partitioning is External fragmentation. Although, this can be removed by Compaction but as we have discussed earlier, the compaction makes the system inefficient.

We need to find out a mechanism which can load the processes in the partitions in a more optimal way. Let us discuss a dynamic and flexible mechanism called paging.

Need for Paging

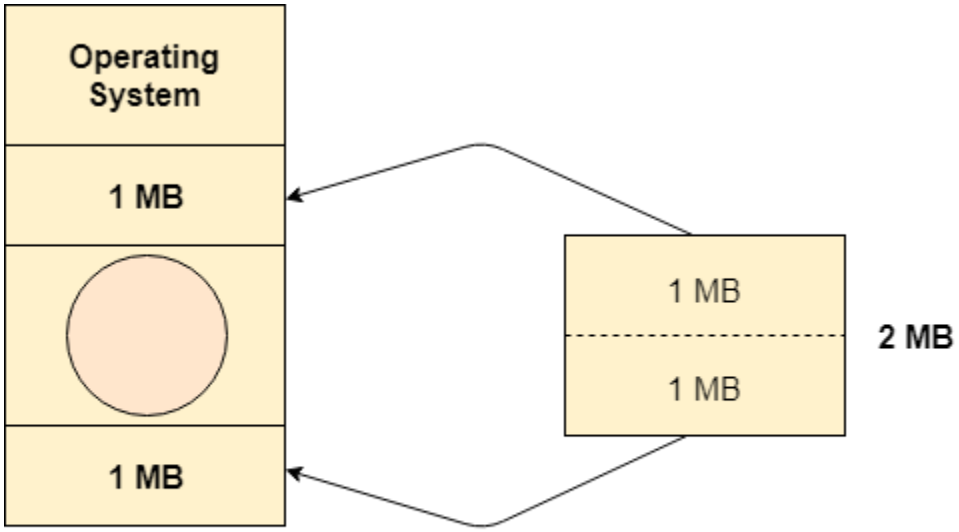
Lets consider a process P1 of size 2 MB and the main memory which is divided into three partitions. Out of the three partitions, two partitions are holes of size 1 MB each.

P1 needs 2 MB space in the main memory to be loaded. We have two holes of 1 MB each but they are not contiguous.

Although, there is 2 MB space available in the main memory in the form of those holes but that remains useless until it become contiguous. This is a serious problem to address.

We need to have some kind of mechanism which can store one process at different locations of the memory.

The Idea behind paging is to divide the process in pages so that, we can store them in the memory at different holes. We will discuss paging with the examples in the next sections.



The process needs to be divided into two parts to get stored at two different places.

Paging with Example

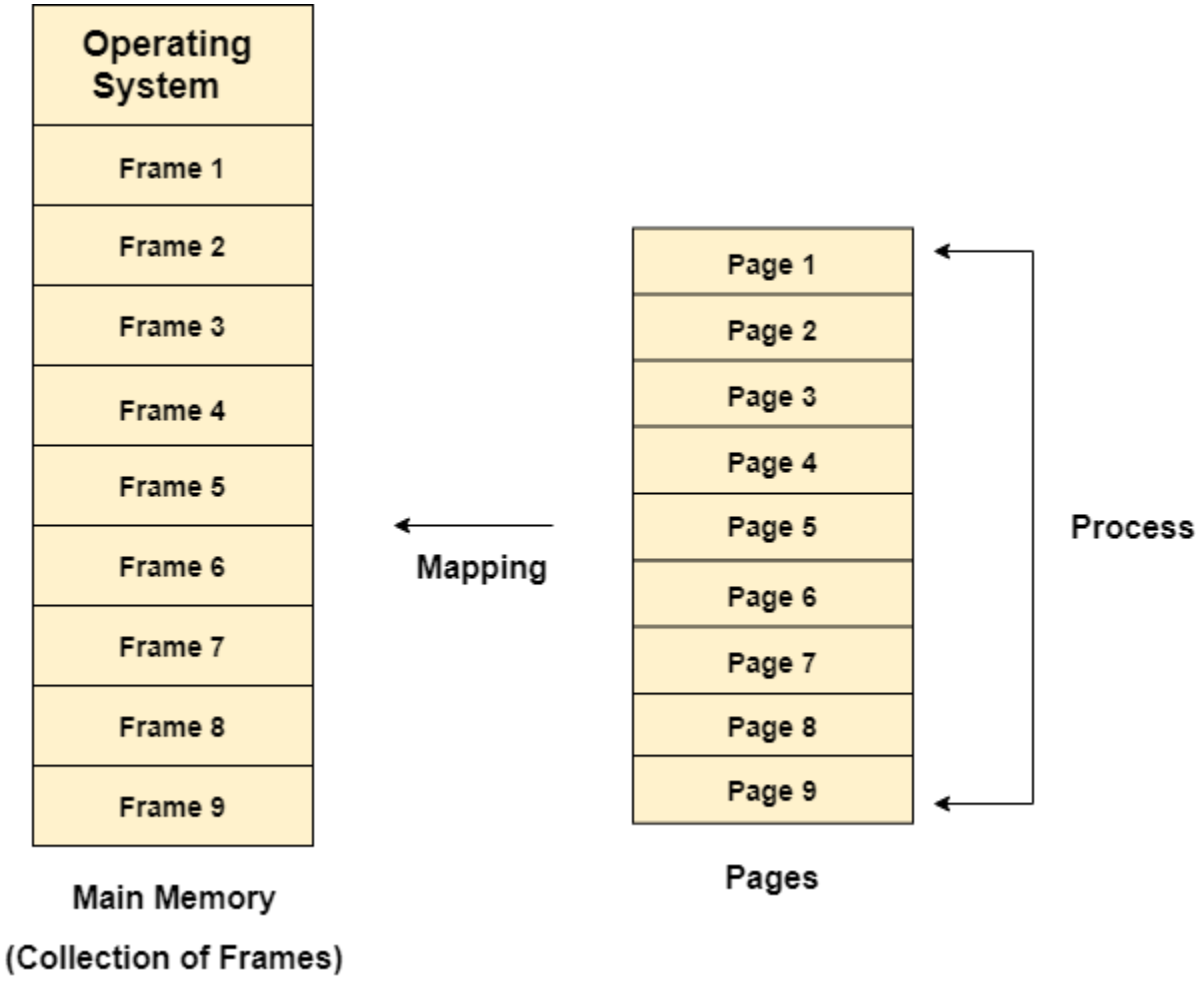
In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



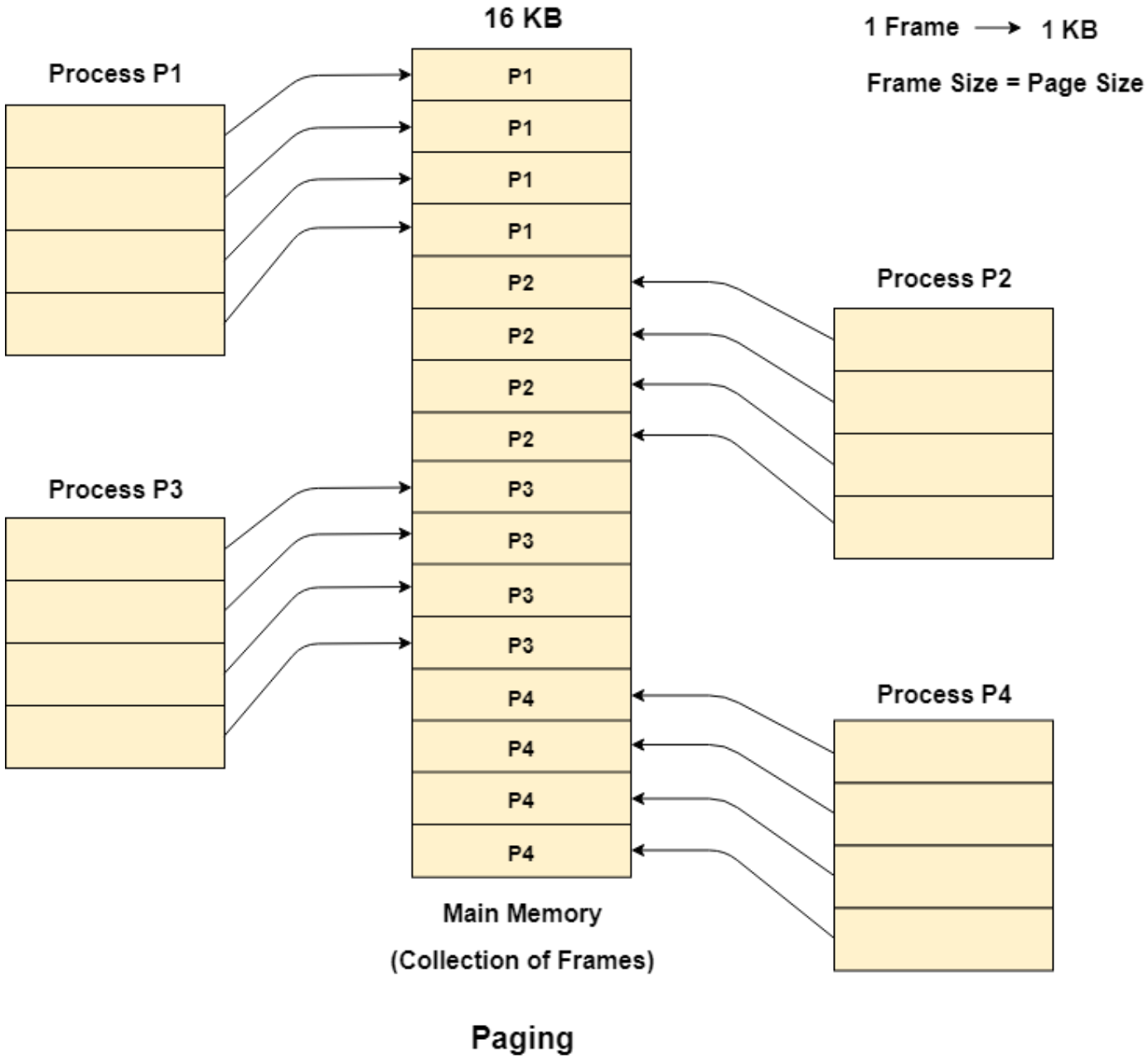
Example

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

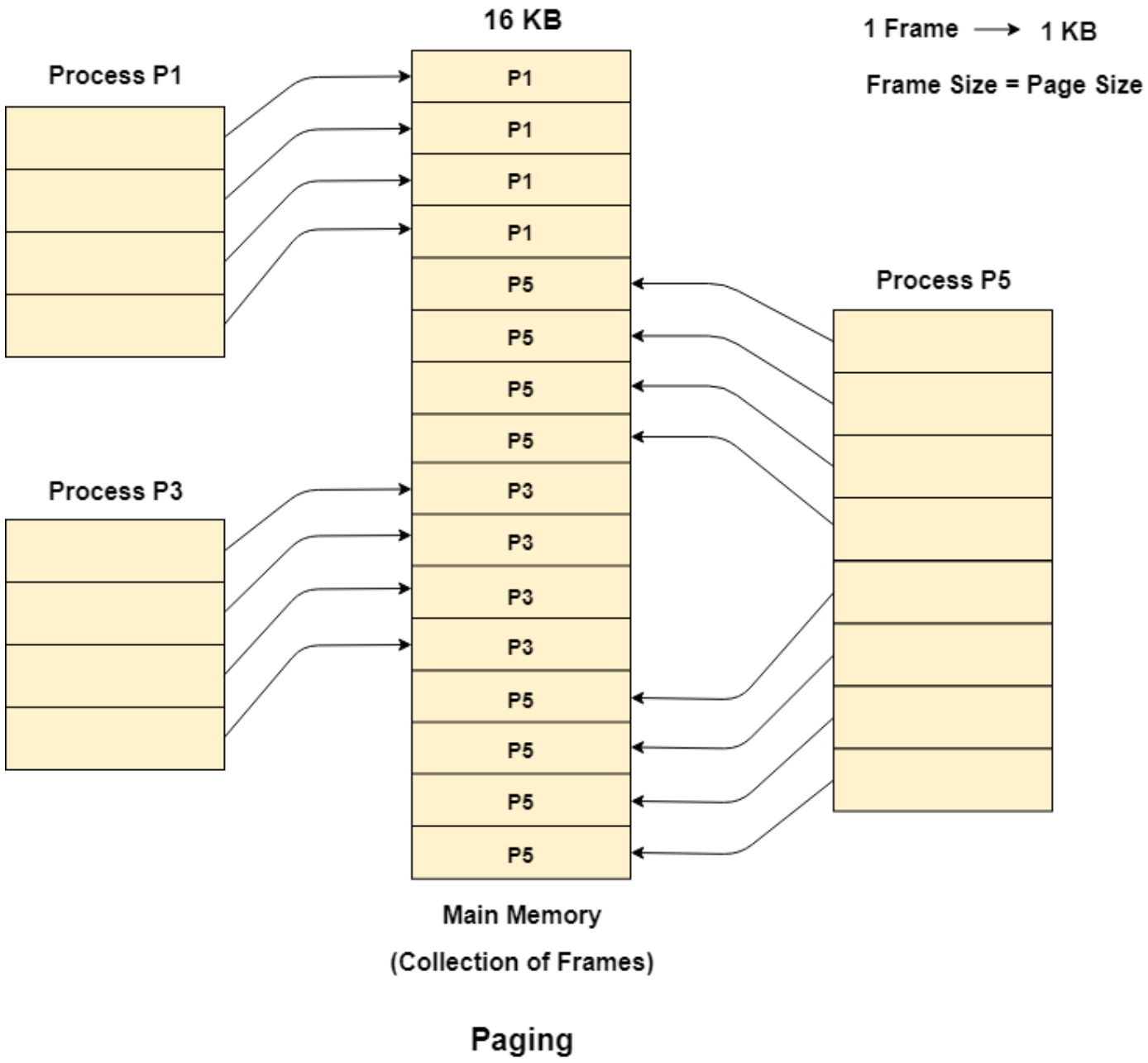
Frames, pages and the mapping between the two is shown in the image below.



Paging

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.



Paging

Memory Management Unit

The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address. The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.

When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

1. Page Number
2. Offset

Memory management unit of OS needs to convert the page number to the frame number.

Example

Considering the above image, let's say that the CPU demands 10th word of 4th page of process P3. Since the page number 4 of process P1 gets stored at frame number 9 therefore the 10th word of 9th frame will be returned as the physical address.

Basics of Binary Addresses

Computer system assigns the binary addresses to the memory locations. However, The system uses amount of bits to address a memory location.

Using 1 bit, we can address two memory locations. Using 2 bits we can address 4 and using 3 bits we can address 8 memory locations.

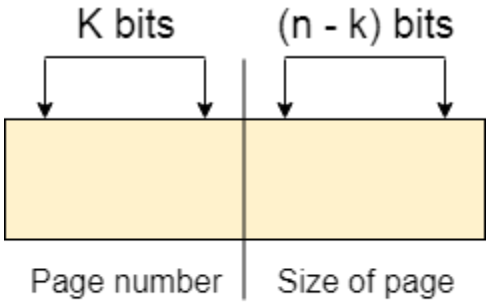
A pattern can be identified in the mapping between the number of bits in the address and the range of the memory locations.

We know,

1. Using 1 Bit we can represent 2^1 i.e 2 memory locations.
2. Using 2 bits, we can represent 2^2 i.e. 4 memory locations.
3. Using 3 bits, we can represent 2^3 i.e. 8 memory locations.
4. Therefore, **if** we generalize **this**,
5. Using n bits, we can assign 2^n memory locations.
6.
7. n bits of address → 2 ^ n memory locations

1 Bit	2 Bits		3 Bits		
0	0	0	0	0	0
1	0	1	0	0	1
	1	0	0	1	0
	1	1	0	1	1
			1	0	0
			1	0	1
			1	1	0
			1	1	1

these n bits can be divided into two parts, that are, **K** bits and **(n-k)** bits.



$$2^n = 2^n \times 2^{n-k}$$

Physical and Logical Address Space

PHYSICAL ADDRESS SPACE:-

Physical address space in a system can be defined as the size of the main memory. It is really important to compare the process size with the physical address space. The process size must be less than the physical address space.

Physical Address Space = Size of the Main Memory

If, physical address space = 64 KB = 2 ^ 6 KB = 2 ^ 6 X 2 ^ 10 Bytes = 2 ^ 16 bytes

Let us consider,
word size = 8 Bytes = 2 ^ 3 Bytes

Hence,
Physical address space (in words) = (2 ^ 16) / (2 ^ 3) = 2 ^ 13 Words

Therefore,
Physical Address = 13 bits

In General,
If, Physical Address Space = N Words

then, Physical Address = $\log_2 N$

Logical Address Space

Logical address space can be defined as the size of the process. The size of the process should be less enough so that it can reside in the main memory.

Let's say,

Logical Address Space = 128 MB = (2 ^ 7 X 2 ^ 20) Bytes = 2 ^ 27 Bytes
Word size = 4 Bytes = 2 ^ 2 Bytes

Logical Address Space (in words) = (2 ^ 27) / (2 ^ 2) = 2 ^ 25 Words
Logical Address = 25 Bits

In general,
If, logical address space = L words
Then, Logical Address = $Log_2 L$ bits

What is a Word?

The Word is the smallest unit of the memory. It is the collection of bytes. Every operating system defines different word sizes after analyzing the n-bit address that is inputted to the decoder and the 2 ^ n memory locations that are produced from the decoder.

PAGE TABLE:-

Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.

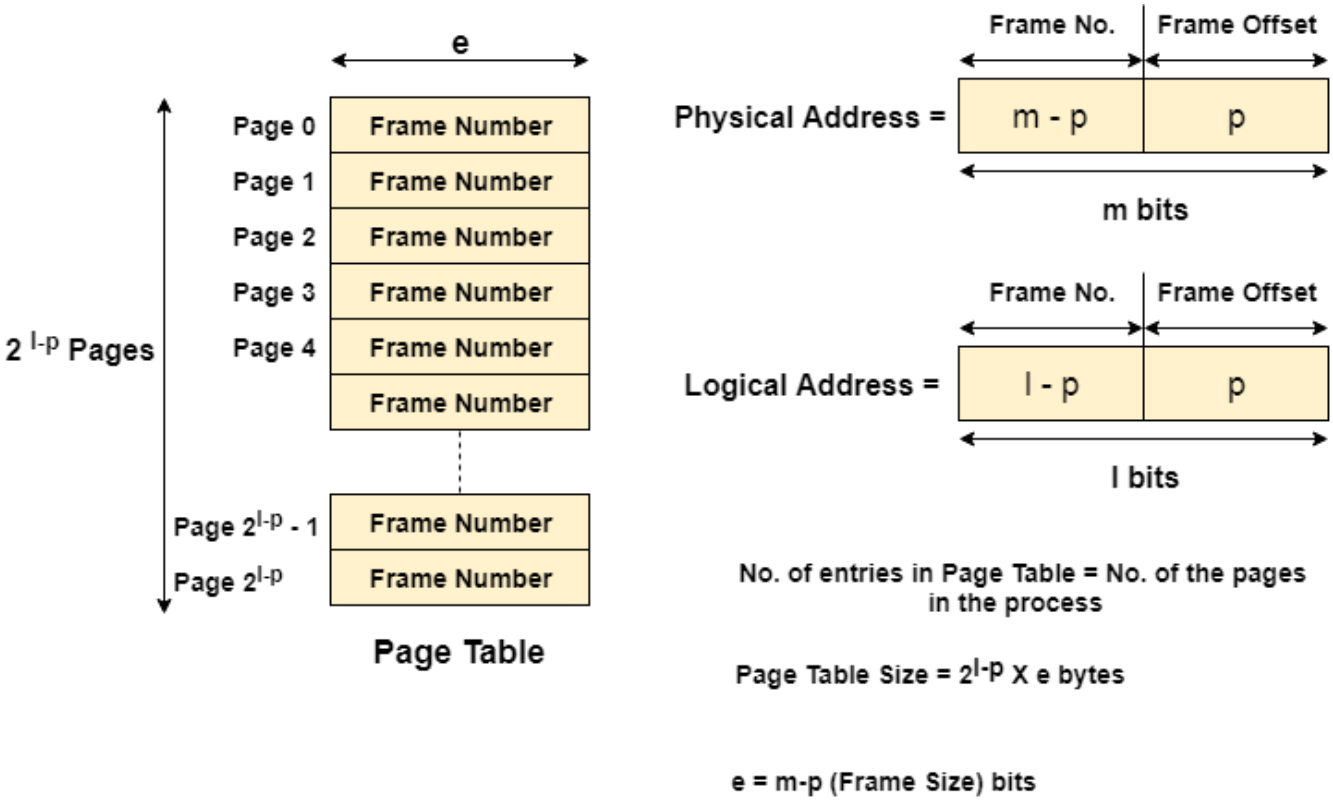
Logical addresses are generated by the CPU for the pages of the processes therefore they are generally used by the processes.

Physical addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

The image given below considers,

Physical Address Space = M words
Logical Address Space = L words
Page Size = P words

Physical Address = $\log_2 M$ = m bits
Logical Address = $\log_2 L$ = l bits
page offset = $\log_2 P$ = p bits



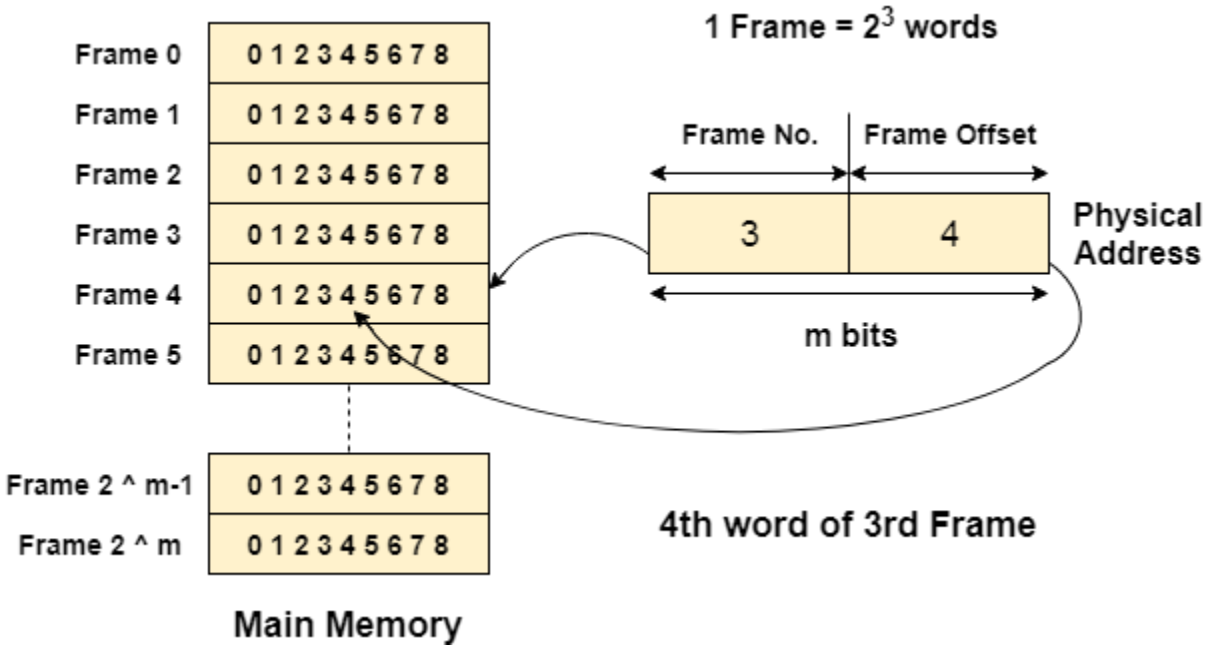
The CPU always accesses the processes through their logical addresses. However, the main memory recognizes physical address only.

In this situation, a unit named as Memory Management Unit comes into the picture. It converts the page number of the logical address to the frame number of the physical address. The offset remains same in both the addresses.

To perform this task, Memory Management unit needs a special kind of mapping which is done by page table. The page table stores all the Frame numbers corresponding to the page numbers of the page table.

In other words, the page table maps the page number to its actual location (frame number) in the memory.

In the image given below shows, how the required word of the frame is accessed with the help of offset.



Mapping from page table to main memory

In operating systems, there is always a requirement of mapping from logical address to the physical address. However, this process involves various steps which are defined as follows.

1. Generation of logical address

CPU generates logical address for each page of the process. This contains two parts: page number and offset.

2. Scaling

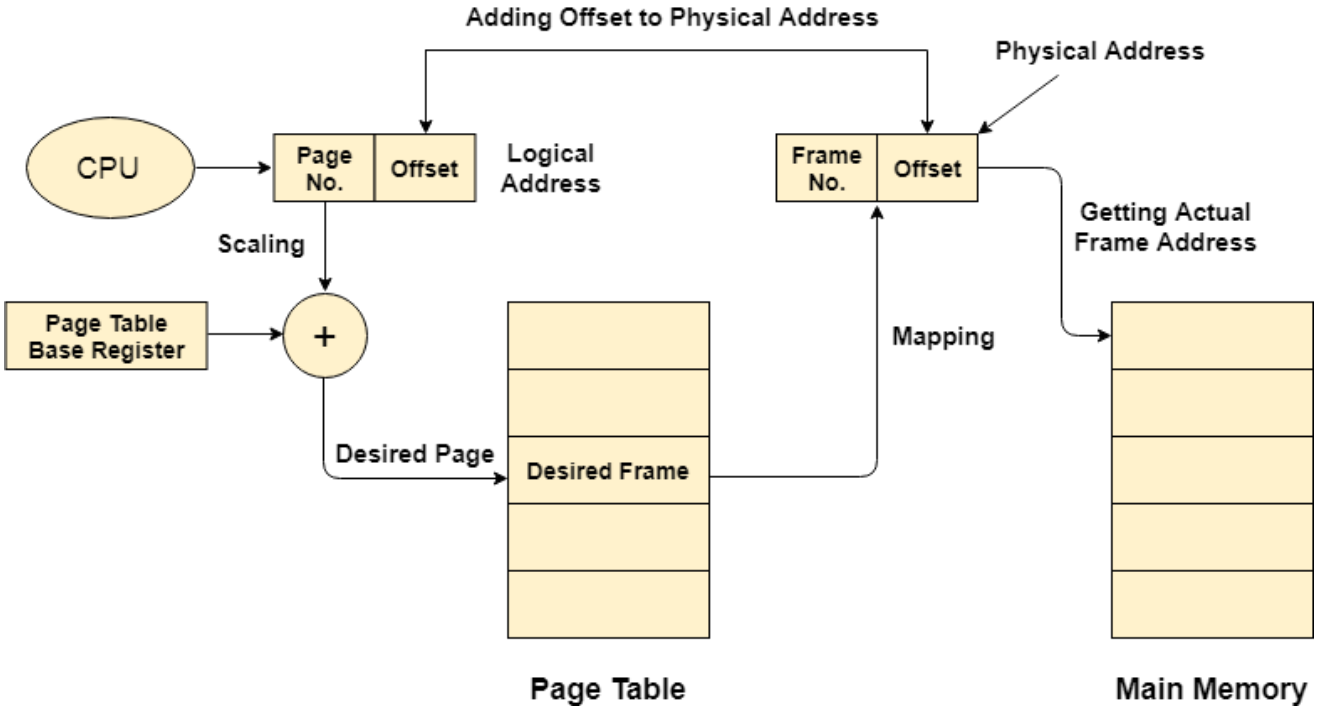
To determine the actual page number of the process, CPU stores the page table base in a special register. Each time the address is generated, the value of the page table base is added to the page number to get the actual location of the page entry in the table. This process is called scaling.

3. Generation of physical Address

The frame number of the desired page is determined by its entry in the page table. A physical address is generated which also contains two parts : frame number and offset. The Offset will be similar to the offset of the logical address therefore it will be copied from the logical address.

4. Getting Actual Frame Number

The frame number and the offset from the physical address is mapped to the main memory in order to get the actual word address.



Page Table Entry

Along with page frame number, the page table also contains some of the bits representing the extra information regarding the page.

Let's see what the each bit represents about the page.

1. Caching Disabled

Sometimes, there are differences between the information closest to the CPU and the information closest to the user. Operating system always wants CPU to access user's data as soon as possible. CPU accesses cache which can be inaccurate in some of the cases, therefore, OS can disable the cache for the required pages. This bit is set to 1 if the cache is disabled.

2. Referenced

There are variouspage replacement algorithms which will be covered later in this tutorial. This bit is set to 1 if the page is referred in the last clock cycle otherwise it remains 0.

3. Modified

This bit will be set if the page has been modified otherwise it remains 0.

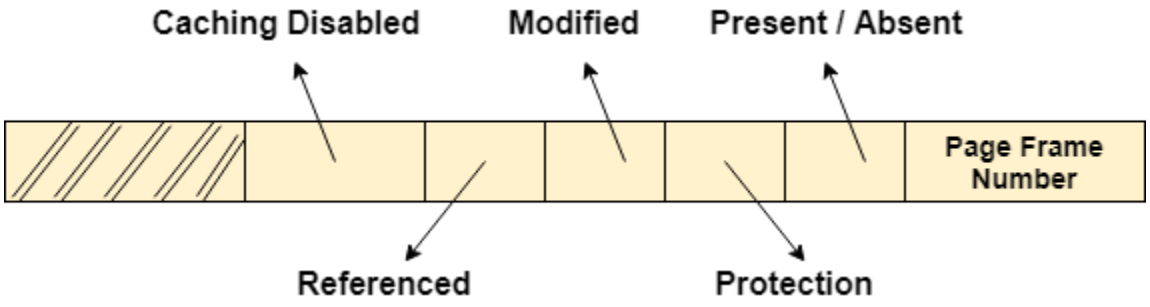
4. Protection

The protection field represents the protection level which is applied on the page. It can be read only or read & write or execute. We need to remember that it is not a bit rather it is a field which contains many bits.

5. Present/Absent

In the concept of demand paging, all the pages doesn't need to be present in the main memory Therefore, for all the pages that are present in the main memory, this bit will be set to 1 and the bit will be 0 for all the pages which are absent.

If some page is not present in the main memory then it is called page fault.



Size of the page table

However, the part of the process which is being executed by the CPU must be present in the main memory during that time period. The page table must also be present in the main memory all the time because it has the entry for all the pages.

The size of the page table depends upon the number of entries in the table and the bytes stored in one entry.

Let's consider,

1. Logical Address = 24 bits
2. Logical Address space = 2 ^ 24 bytes
3. Let's say, Page size = 4 KB = 2 ^ 12 Bytes
4. Page offset = 12
5. Number of bits in a page = Logical Address - Page Offset = 24 - 12 = 12 bits
6. Number of pages = 2 ^ 12 = 2 X 2 X 10 ^ 10 = 4 KB
7. Let's say, Page table entry = 1 Byte
8. Therefore, the size of the page table = 4 KB X 1 Byte = 4 KB

Here we are lucky enough to get the page table size equal to the frame size. Now, the page table will be simply stored in one of the frames of the main memory. The CPU maintains a register which contains the base address of that frame, every page number from the logical address will first be added to that base address so that we can access the actual location of the word being asked.

However, in some cases, the page table size and the frame size might not be same. In those cases, the page table is considered as the collection of frames and will be stored in the different frames.

Finding Optimal Page Size

We have seen that the bigger page table size cause an extra overhead because we have to divide that table into the pages and then store that into the main memory.

Our concern must be about executing processes not on the execution of page table. Page table provides a support for the execution of the process. The larger the page Table, the higher the overhead.

We know that,

1. Page Table Size = number of page entries in page table X size of one page entry
2. Let's consider an example,
3. Virtual Address Space = 2 GB = 2 X 2 ^ 30 Bytes
4. Page Size = 2 KB = 2 X 2 ^ 10 Bytes
5. Number of Pages in Page Table = (2 X 2 ^ 30)/(2 X 2 ^ 10) = 1 M pages

There will be 1 million pages which is quite big number. However, try to make page size larger, say 2 MB.

Then, Number of pages in page table = (2 X 2 ^ 30)/(2 X 2 ^ 20) = 1 K pages.

If we compare the two scenarios, we can find out that the page table size is anti proportional to Page Size.

In Paging, there is always wastage on the last page. If the virtual address space is not a multiple of page size, then there will be some bytes remaining and we have to assign a full page to those many bytes. This is simply a overhead.

Let's consider,

1. Page Size = 2 KB
2. Virtual Address Space = 17 KB
3. Then number of pages = 17 KB / 2 KB

The number of pages will be 9 although the 9th page will only contain 1 byte and the remaining page will be wasted.

In general,

1. If page size = p bytes
2. Entry size = e bytes
3. Virtual Address Space = S bytes
4. Then, overhead O = (S/p) X e + (p/2)

On an average, the wasted number of pages in a virtual space is p/2(the half of total number of pages).

For, the minimal overhead,

1. $\partial O/\partial p = 0$
2. $-S/(p^2) + ½ = 0$
3. $p = \sqrt{(2.S.e)}$ bytes

Hence, if the page size $\sqrt{(2.S.e)}$ bytes then the overhead will be minimal.

Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works?

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

Demand Paging

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.

A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. We will discuss each one of them later in detail.

Snapshot of a virtual memory management system

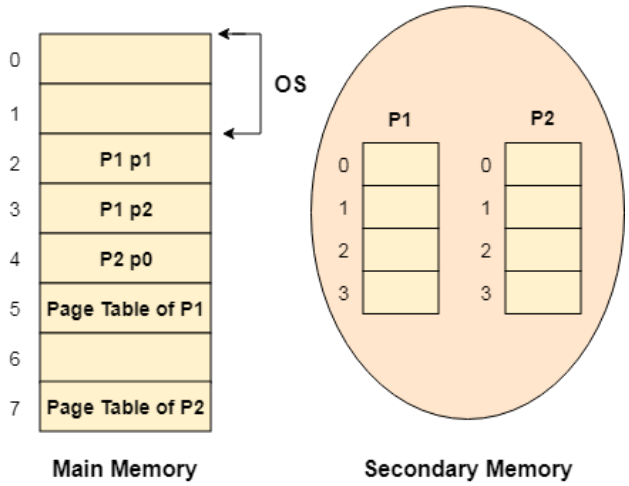
Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.

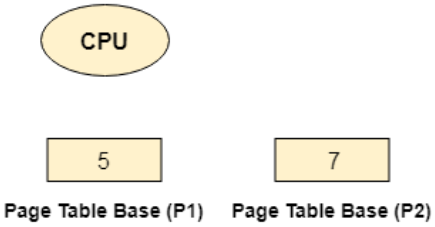
	Frame	Present / Absent	D Bit	Reference bit	Protection
0		0	0	0	
1	2	1	0	1	
2	3	1	1	0	
3		0	0	0	

Page Table of P1



	Frame	Present / Absent	D Bit	Reference bit	Protection
0	4	1	0	1	
1		0	0	0	
2		0	0	0	
3		0	0	0	

Page Table of P2



Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

TRANSLATION LOOK ASIDE BUFFER:-

Drawbacks of Paging

1. Size of Page table can be very big and therefore it wastes main memory.

2. CPU will take more time to read a single word from the main memory.

How to decrease the page table size

1. The page table size can be decreased by increasing the page size but it will cause internal fragmentation and there will also be page wastage.
2. Other way is to use multilevel paging but that increases the effective access time therefore this is not a practical approach.

How to decrease the effective access time

1. CPU can use a register having the page table stored inside it so that the access time to access page table can become quite less but the register are not cheaper and they are very small in compare to the page table size therefore, this is also not a practical approach.
2. To overcome these many drawbacks in paging, we have to look for a memory that is cheaper than the register and faster than the main memory so that the time taken by the CPU to access page table again and again can be reduced and it can only focus to access the actual word.

Locality of reference

In operating systems, the concept of locality of reference states that, instead of loading the entire process in the main memory, OS can load only those number of pages in the main memory that are frequently accessed by the CPU and along with that, the OS can also load only those page table entries which are corresponding to those many pages.

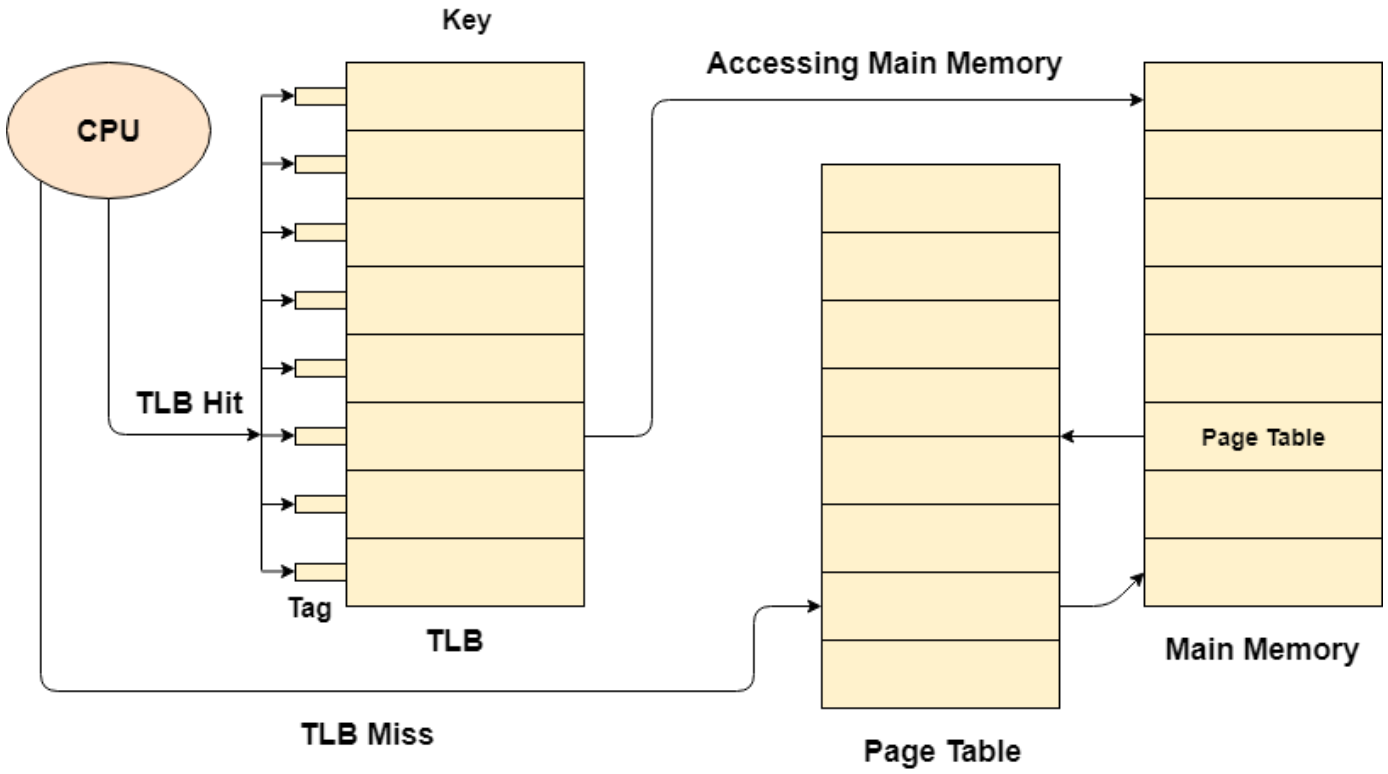
Translation look aside buffer (TLB)

A Translation look aside buffer can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again.

It is a memory cache which is closer to the CPU and the time taken by CPU to access TLB is lesser then that taken to access main memory.

In other words, we can say that TLB is faster and smaller than the main memory but cheaper and bigger than the register.

TLB follows the concept of locality of reference which means that it contains only the entries of those many pages that are frequently accessed by the CPU.



In translation look aside buffers, there are tags and keys with the help of which, the mapping is done.

TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory.

However, if the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory.

Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

If the probability of TLB hit is P% (TLB hit rate) then the probability of TLB miss (TLB miss rate) will be (1-P) %.

Therefore, the effective access time can be defined as;

1. EAT = P (t + m) + (1 - p) (t + k.m + m)

Where, p → TLB hit rate, t → time taken to access TLB, m → time taken to access main memory k = 1, if the single level paging has been implemented.

By the formula, we come to know that

1. Effective access time will be decreased if the TLB hit rate is increased.
2. Effective access time will be increased in the case of multilevel paging.

Demand Paging

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

Whenever any page is referred for the first time in the main memory, then that page will be found in the secondary memory.

After that, it may or may not be present in the main memory depending upon the page replacement algorithm which will be covered later in this tutorial.

What is a Page Fault?

If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault.

The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

What is Thrashing?

If the number of page faults is equal to the number of referred pages or the number of page faults are so high so that the CPU remains busy in just reading the pages from the secondary memory then the effective access time will be the time taken by the CPU to read one word from the secondary memory and it will be so high. The concept is called thrashing.

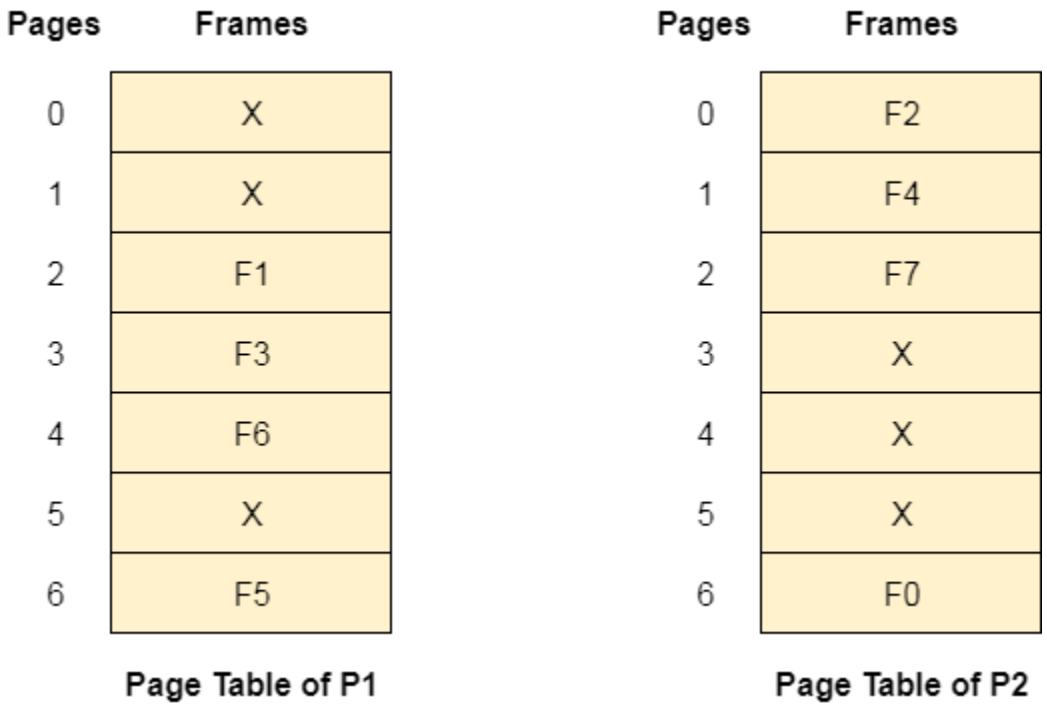
If the page fault rate is PF %, the time taken in getting a page from the secondary memory and again restarting is S (service time) and the memory access time is ma then the effective access time can be given as;

1. EAT = PF X S + (1 - PF) X (ma)

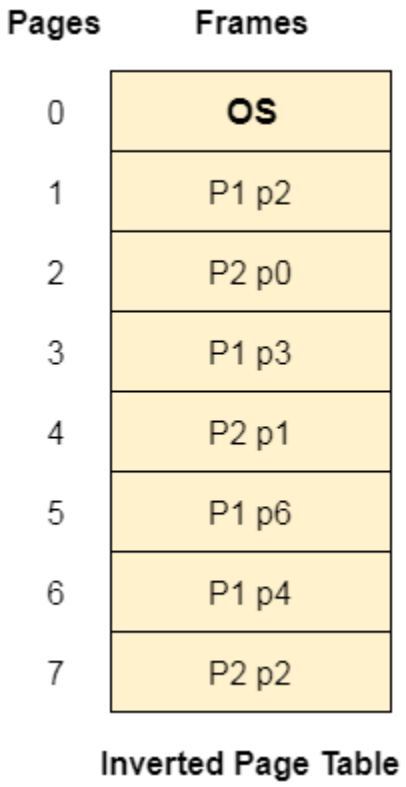
Inverted Page Table

Inverted Page Table is the global page table which is maintained by the Operating System for all the processes. In inverted page table, the number of entries is equal to the number of frames in the main memory. It can be used to overcome the drawbacks of page table.

There is always a space reserved for the page regardless of the fact that whether it is present in the main memory or not. However, this is simply the wastage of the memory if the page is not present.

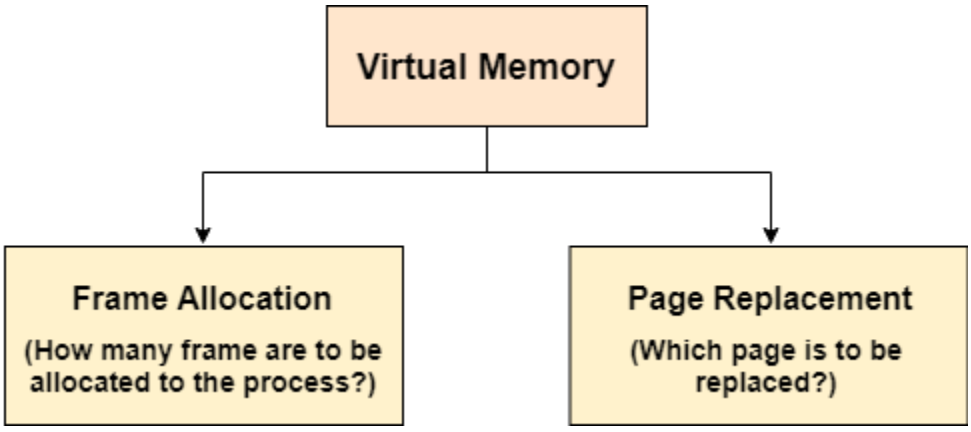


We can save this wastage by just inverting the page table. We can save the details only for the pages which are present in the main memory. Frames are the indices and the information saved inside the block will be Process ID and page number.



Page Replacement Algorithms

The page replacement algorithm 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).



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.

What If the algorithm is not optimal?

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.

Types of Page Replacement Algorithms

There are various page replacement algorithms. Each algorithm has a different method by which the pages can be replaced.

1. **Optimal Page Replacement algorithm** → this algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but it can be used as a benchmark. Other algorithms are compared to this in terms of optimality.
2. **Least recent used (LRU) page replacement algorithm** → this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
3. **FIFO** → in this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

Numerical on Optimal, LRU and FIFO

Q. Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm
2. FIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm

Optimal Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

BELADY'SANOMALY:-

In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames. However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.

This is the strange behavior shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady'sAnomaly.

Let's examine such example :

The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behavior of FIFO algorithm in two cases.

Case 1: Number of frames = 3

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 3			5	5	5	1	1	1	1	1	3	3
Frame 2		1	1	1	0	0	0	0	0	5	5	5
Frame 1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Case 2: Number of frames = 4

Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 4				3	3	3	3	3	3	5	5	5
Frame 3			5	5	5	5	5	5	1	1	1	1
Frame 2		1	1	1	1	1	1	0	0	0	0	4
Frame 1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady's Anomaly.

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 segment which can be allocated to a process.

The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. 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 Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which needs to be loaded in the same page.

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 contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment,

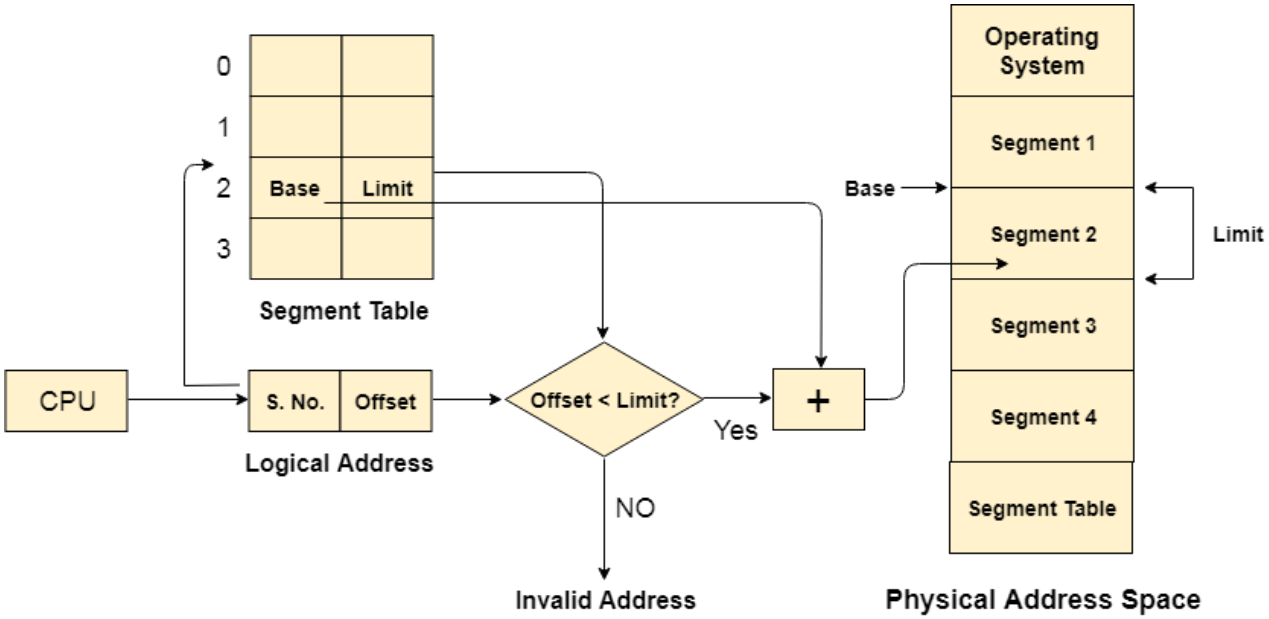
Translation of Logical address into physical address by segment table

CPU generates a logical address which contains two parts:

1. Segment Number
2. Offset

The Segment number is mapped to the segment table. 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 address, the base address of the segment is added to the offset to get the physical address of actual word in the main memory.



Advantages of Segmentation

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compare to the page table in paging.

Disadvantages

1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

Paging VS Segmentation

Sr No	Paging	Segmentation
1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
5	Paging is closer to Operating System	Segmentation is closer to User
6	It suffers from internal fragmentation	It suffers from external fragmentation
7	There is no external fragmentation	There is no external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information

10	Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.

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.

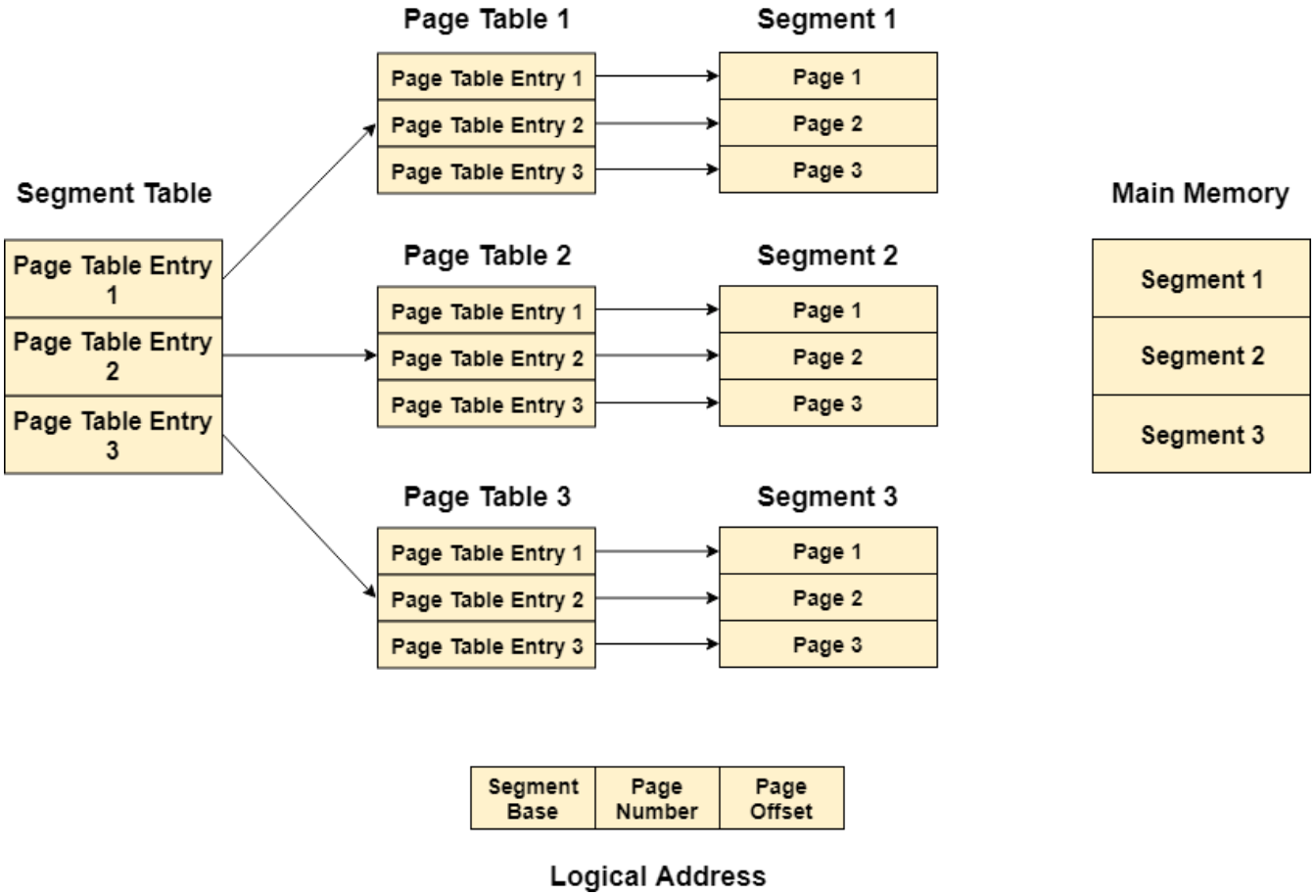
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

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

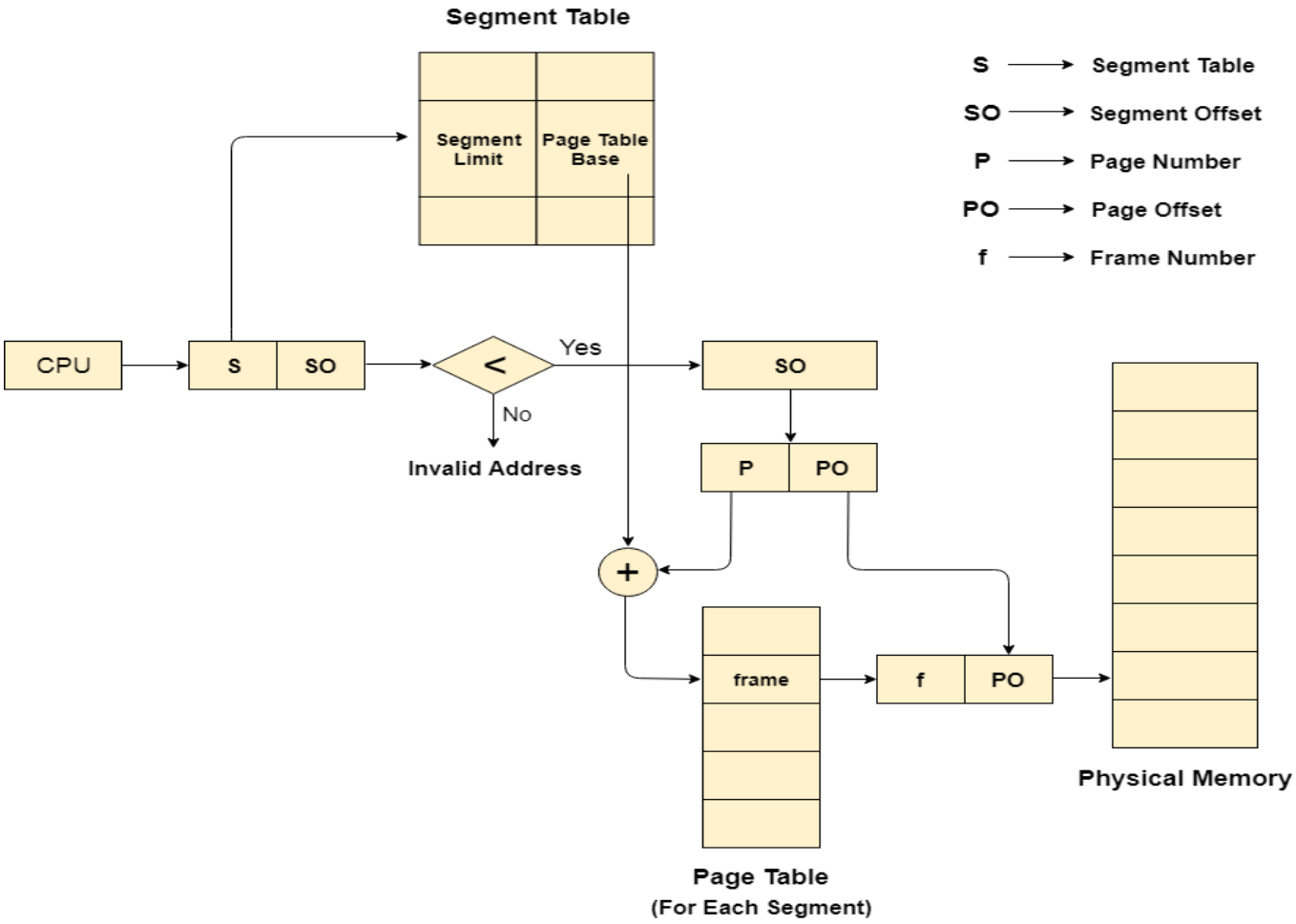
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



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.

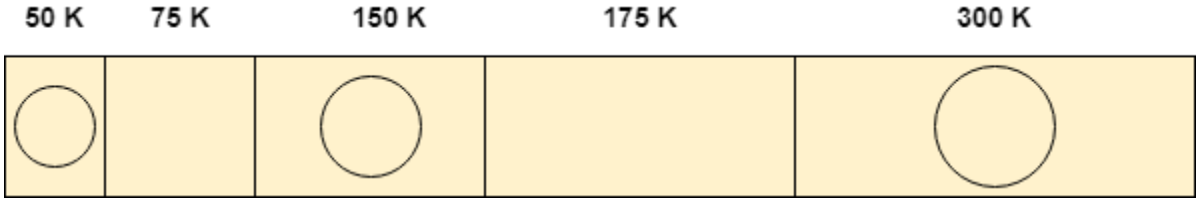
GATE QUESTION

GATE question on best fit and first fit

From the GATE point of view, Numerical on best fit and first fit are being asked frequently in 1 mark. Let's have a look on the one given as below.

Q. Process requests are given as;

25 K , 50 K , 100 K , 75 K



Determine the algorithm which can optimally satisfy this requirement.

1. First Fit algorithm
2. Best Fit Algorithm
3. Neither of the two
4. Both of them

In the question, there are five partitions in the memory. 3 partitions are having processes inside them and two partitions are holes.

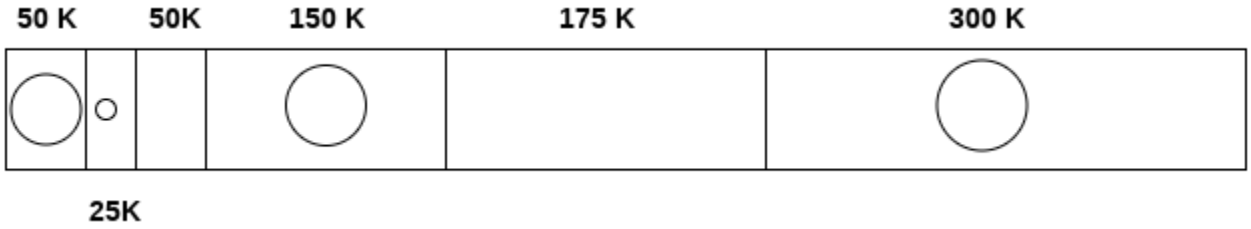
Our task is to check the algorithm which can satisfy the request optimally.

Using First Fit algorithm

Let's see, how first fit algorithm works on this problem.

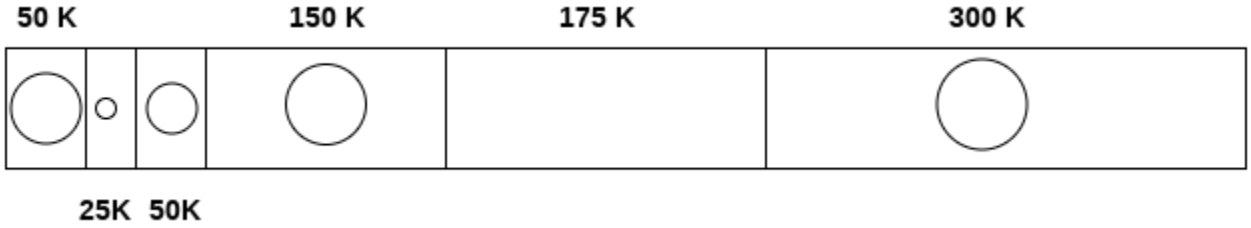
1. 25 K requirement

The algorithm scans the list until it gets first hole which should be big enough to satisfy the request of 25 K. it gets the space in the second partition which is free hence it allocates 25 K out of 75 K to the process and the remaining 50 K is produced as hole.



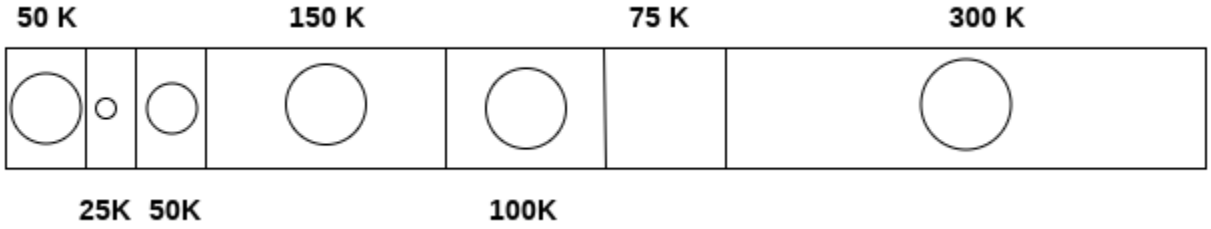
2. 50 K requirement

The 50 K requirement can be fulfilled by allocating the third partition which is 50 K in size to the process. No free space is produced as free space.



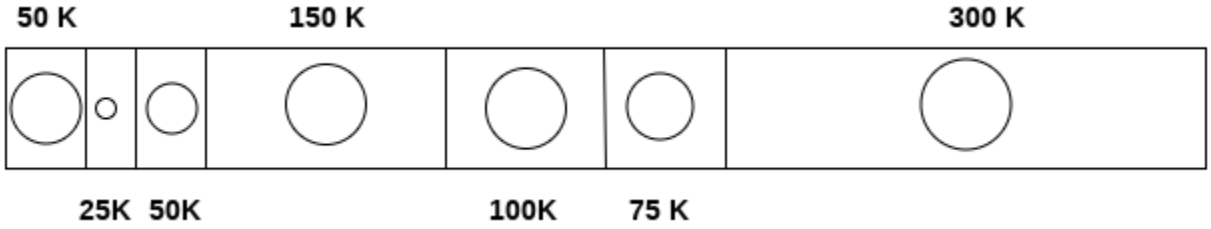
3. 100 K requirement

100 K requirement can be fulfilled by using the fifth partition of 175 K size. Out of 175 K, 100 K will be allocated and remaining 75 K will be there as a hole.



4. 75 K requirement

Since we are having a 75 K free partition hence we can allocate that much space to the process which is demanding just 75 K space.



Using first fit algorithm, we have fulfilled the entire request optimally and no useless space is remaining.

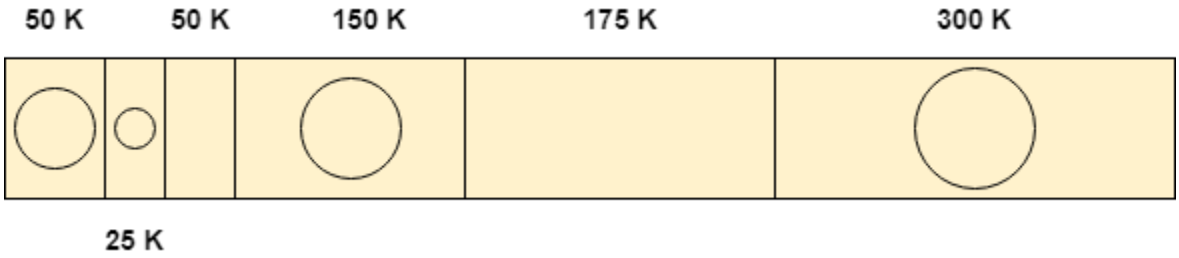
Let's see, How Best Fit algorithm performs for the problem.

Using Best Fit Algorithm

1. 25 K requirement

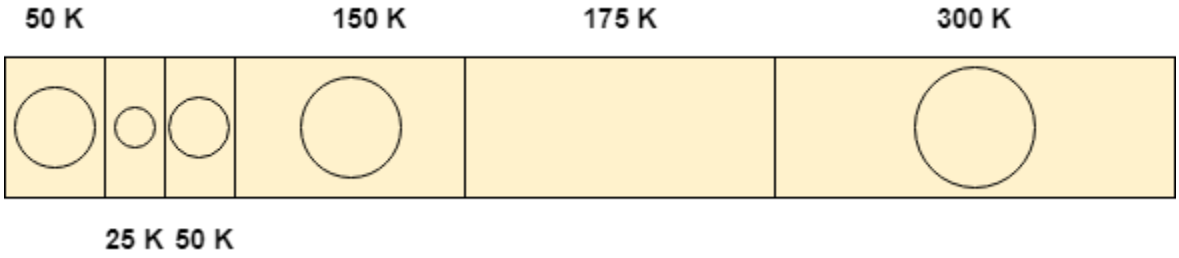
To allocate 25 K space using best fit approach, need to scan the whole list and then we find that a 75 K partition is free and the smallest among all, which can accommodate the need of the process.

Therefore 25 K out of those 75 K free partition is allocated to the process and the remaining 5o K is produced as a hole.



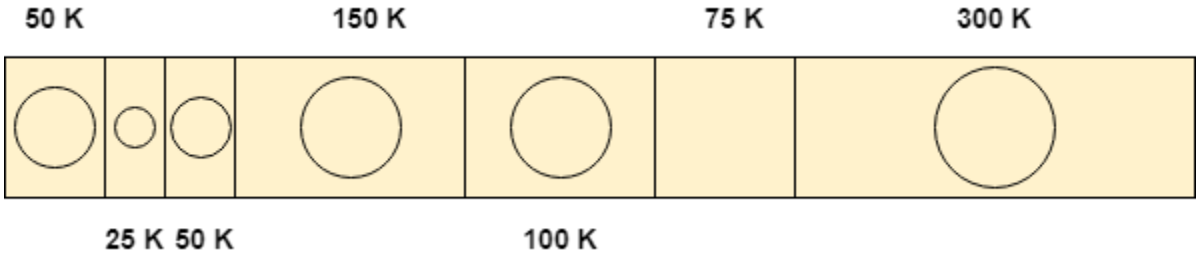
2. 50 K requirement

To satisfy this need, we will again scan the whole list and then find the 50 K space is free which the exact match of the need is. Therefore, it will be allocated for the process.



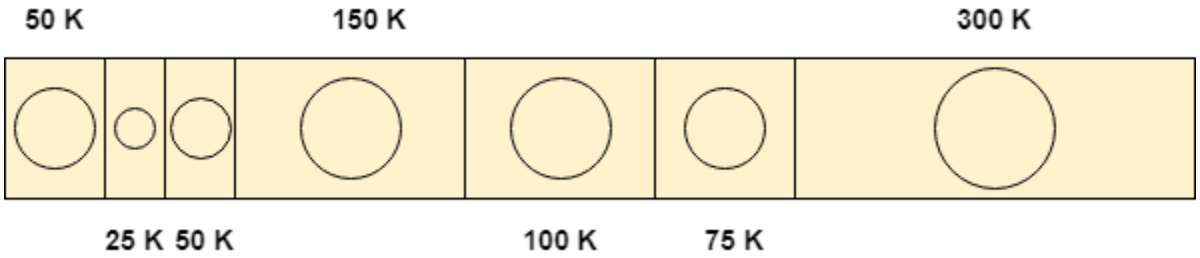
3. 100 K requirement

100 K need is close enough to the 175 K space. The algorithm scans the whole list and then allocates 100 K out of 175 K from the 5th free partition.



4. 75 K requirement

75 K requirement will get the space of 75 K from the 6th free partition but the algorithm will scan the whole list in the process of taking this decision.



By following both of the algorithms, we have noticed that both the algorithms perform similar to most of the extant in this case.

Both can satisfy the need of the processes but however, the best fit algorithm scans the list again and again which takes lot of time.

Therefore, if you ask me that which algorithm performs in more optimal way then it will be **First Fit algorithm** for sure.

Therefore, the answer in this case is A.

GATE Question on TLB

GATE | GATE-CS-2014-(Set-3)

Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _____.

A. 120
B. 122

C. 124
D. 118

Given,

1. TLB hit ratio = 0.6
2. Therefore, TLB miss ratio = 0.4
3. Time taken to access TLB (t) = 10 ms
4. Time taken to access main memory (m) = 80 ms

Effective Access Time (EAT) = 0.6 (10 + 80) + 0.4 (10 + 80 + 80) = 90 X 0.6 + 0.4 X 170 = 122

Hence, the right answer is option B.

GATE 2015 question on LRU and FIFO

Q. Consider a main memory with five page frames and the following sequence of page references: 3, 8, 2, 3, 9, 1, 6, 3, 8, 9, 3, 6, 2, 1, 3. which one of the following is true with respect to page replacement policies First-In-First-out (FIFO) and Least Recently Used (LRU)?

A. Both incur the same number of page faults
B. FIFO incurs 2 more page faults than LRU
C. LRU incurs 2 more page faults than FIFO
D. FIFO incurs 1 more page faults than LRU

Solution:

Number of frames = 5

FIFO

According to FIFO, the page which first comes in the memory will first goes out.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	1	1	1
Frame 4					9	9	9	9	9	9	9	9	2	2	2
Frame 3			2	2	2	2	2	2	8	8	8	8	8	8	8
Frame 2		8	8	8	8	8	8	3	3	3	3	3	3	3	3
Frame 1	3	3	3	3	3	3	6	6	6	6	6	6	6	6	6
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults = 9
Number of hits = 6

LRU

According to LRU, the page which has not been requested for a long time will get replaced with the new one.

Request	3	8	2	3	9	1	6	3	8	9	3	6	2	1	3
Frame 5						1	1	1	1	1	1	1	2	2	2
Frame 4					9	9	9	9	9	9	9	9	9	9	9
Frame 3			2	2	2	2	2	2	8	8	8	8	8	1	1
Frame 2		8	8	8	8	8	6	6	6	6	6	6	6	6	6
Frame 1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Miss/Hit	Miss	Miss	Miss	Hit	Miss	Miss	Hit	Hit	Miss	Hit	Miss	Hit	Miss	Miss	Hit

Number of Page Faults = 9
Number of Hits = 6

The Number of page faults in both the cases is equal therefore the Answer is **option (A)**.