UNIT-IV

Memory management and Virtual Memory :

- Logical versus physical address space
- Swapping
- Contiguous Allocation.
- Paging, Segmentation.
- Segmentation with paging
- Demand paging
- Page replacement
- page replacement algorithms.

Memory is central to the operation of a modern computer system.

Memory consists of a large array of bytes, each with its own address.

The CPU fetches instructions from memory according to the value of the program counter.

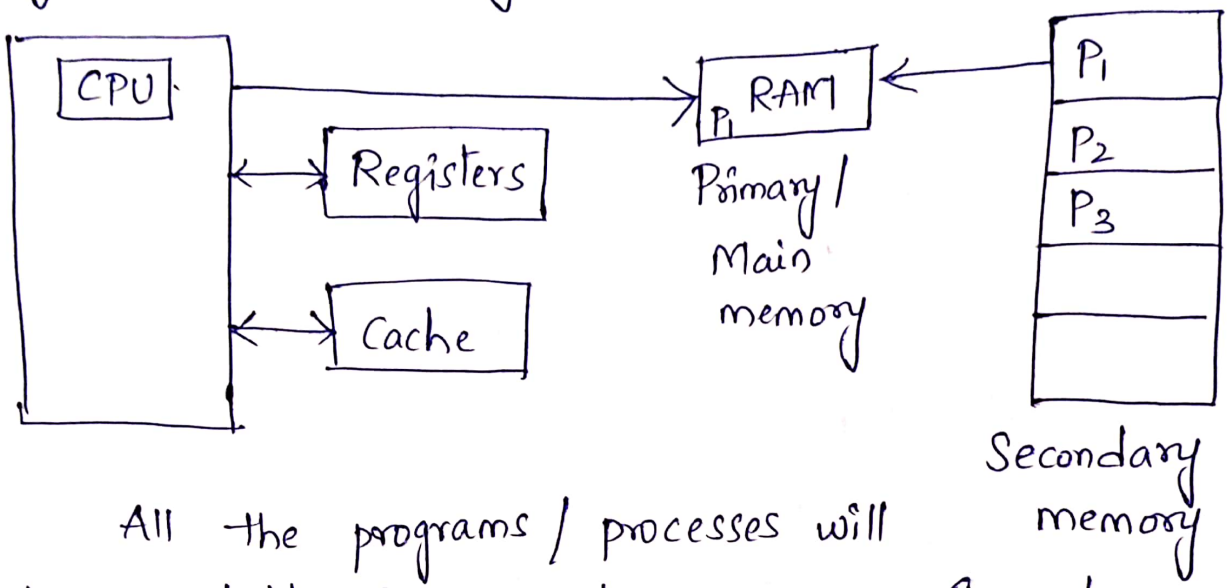
These instructions may cause additional loading from and storing to specific memory address.

Memory management in OS is a kind of method or functionality to manage the various kinds of memories. (RAM, HD, Registers and mainly primary memory).

Goals:

Efficient utilization of memory.

If the RAM increases then the cost of the system automatically increases.



All the programs / processes will be available in secondary memory. Secondary memory space is large but the CPU is not connected to it. So, the programs are moved to RAM from there. Then the CPU executes the program / process.

This process is called the degree of multiprogramming.

Main memory and the registers built into the processor itself and these are only storage devices that the CPU can access directly.

When the CPU utilization increases then automatically system performance also increases.

Logical and Physical addresses :-

An address generated by the CPU is commonly referred as Logical address.

Whereas the address seen by the memory unit, that is one loaded into memory address register of the memory is commonly referred as the Physical address.

The compile time and loadtime address binding generates the identical logical and physical addresses.

However, the execution time address binding scheme results in differing logical and physical addresses.

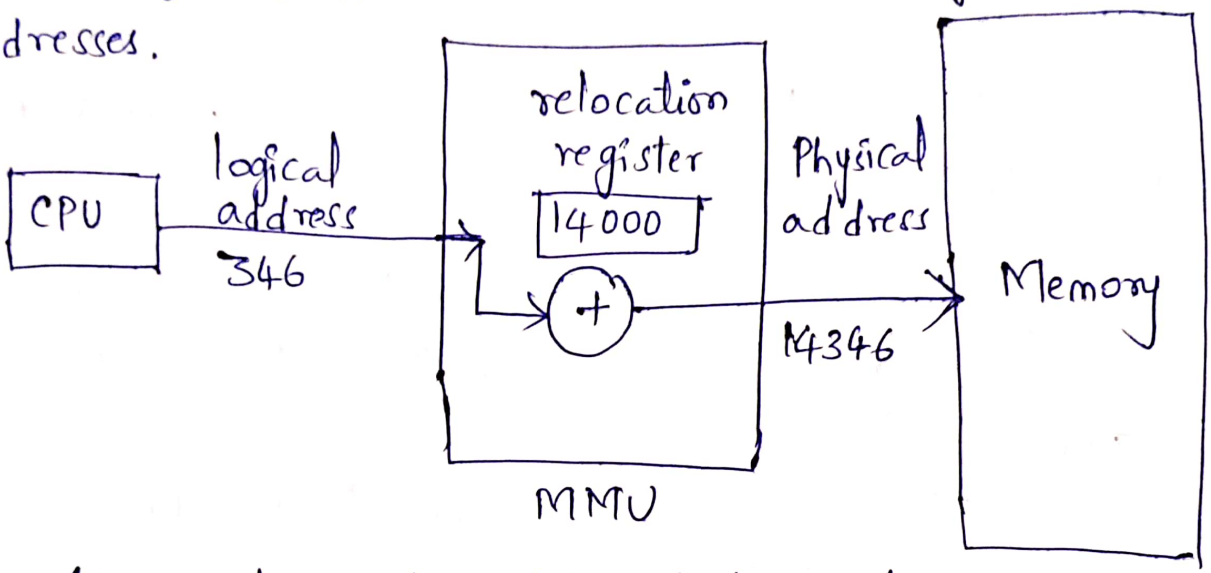
The set of all logical addresses generated by a program is known as Logical address space, whereas the set of all physical addresses corresponding to the logical address is physical address space. Now, the runtime mapping from virtual address to physical address is done by a hardware device is known as Memory management unit (MMU).

Memory management unit (MMU):

Hardware device that maps virtual to physical address.

In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

The user program deals with the logical addresses; it never sees the real physical addresses.



As we have two different type of addresses Logical address in the range (0 to max) and physical address in the range (R to R+max) where R is the value of relocation register.

A pair of base and limit registers define the logical address space.

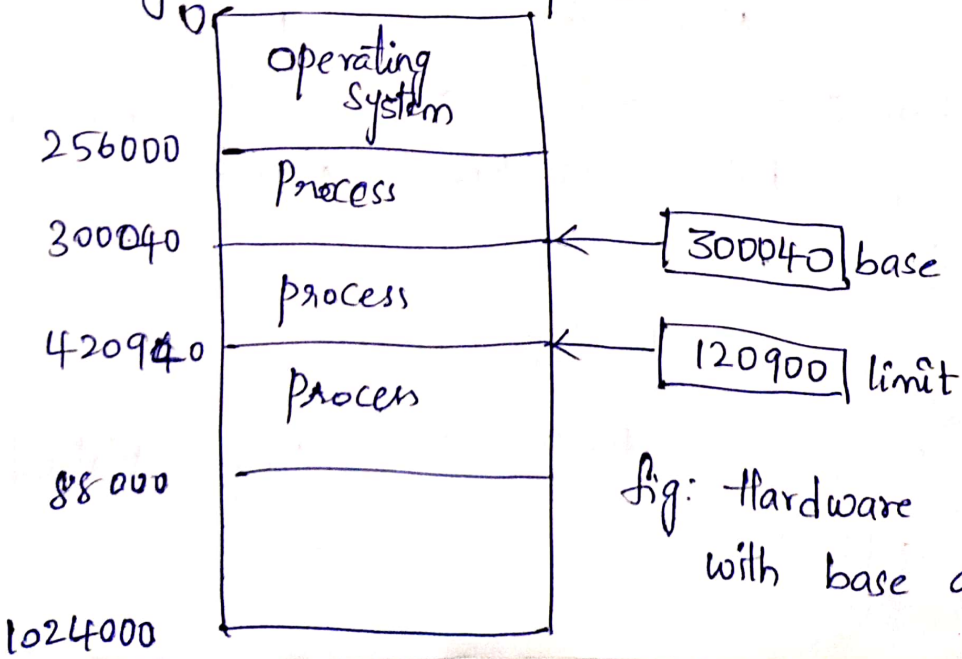


fig: Hardware protection with base and limit.

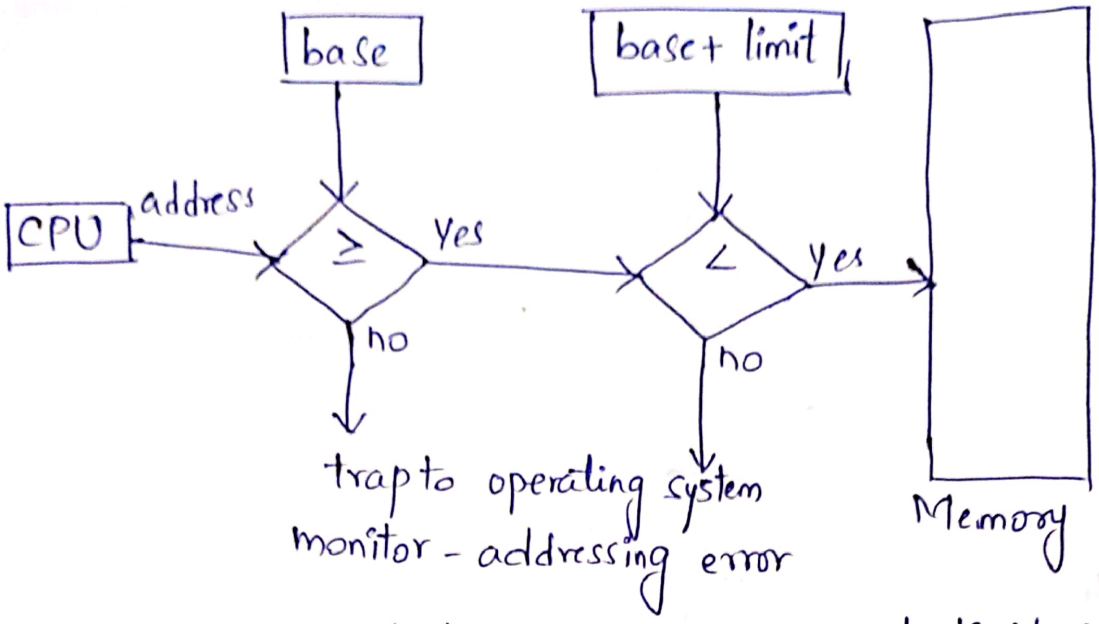


Fig:- Hardware protection with base and limit registers

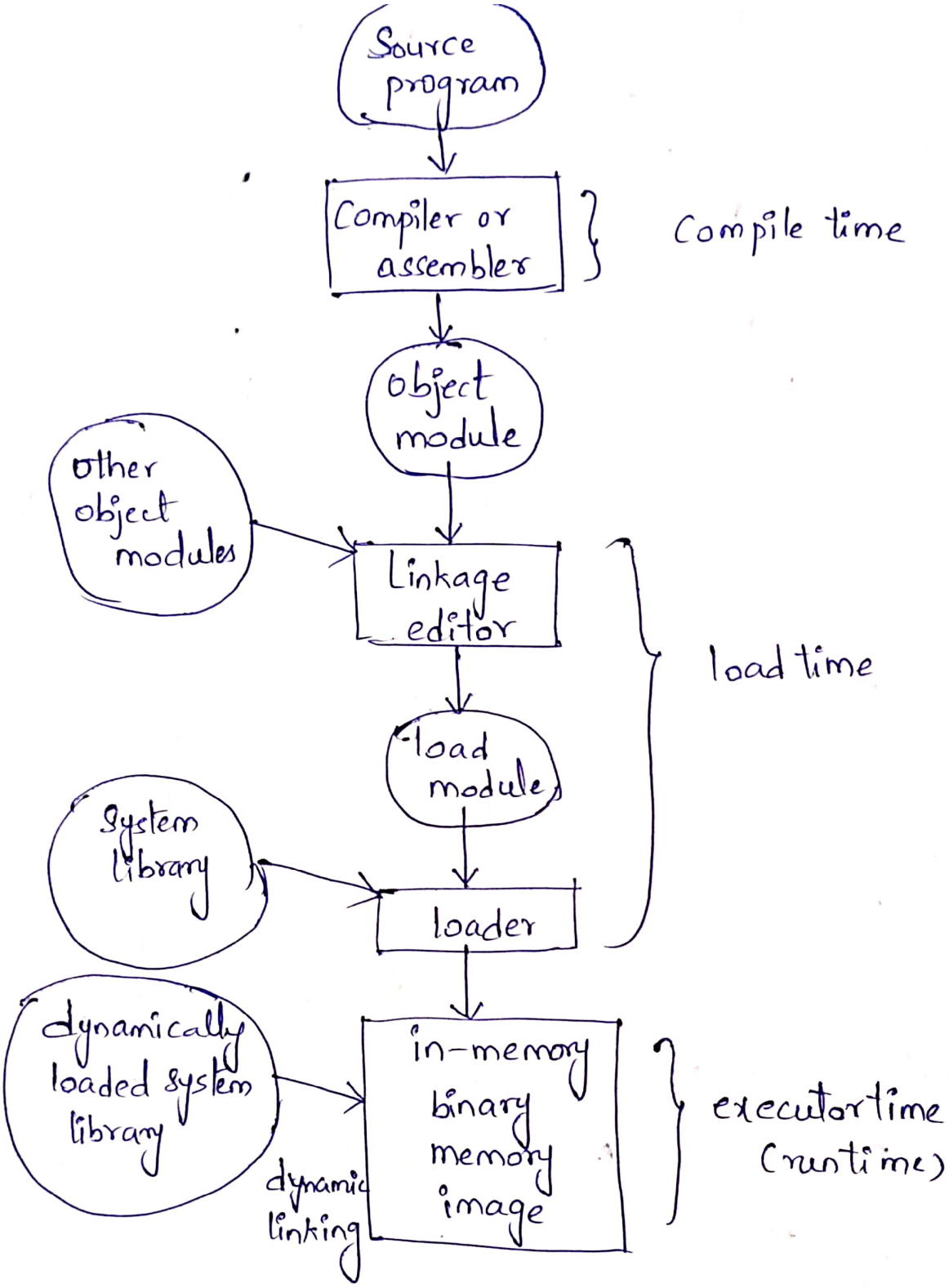
Address Binding :-

Address Binding of instructions and data to memory addresses can happen at three different stages.

Compile time : If memory location known a priori, absolute code can be generated, must recompile code if starting location changes.

Load time : must generate relocatable code if memory location is not known at compile time.

Execution time : Binding delayed until runtime if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g: base and limit registers).



Multi-step processing of a user program.

Dynamic loading :

- Routine is not loaded until it is called.
- Better memory space utilization; unused routine is never loaded.

- Useful when large amounts of code are needed to handle infrequently occuring cases.
- No special support from the operating system is required implemented through program design.

Dynamic Linking :

- Linking postponed until execution time.
- Small piece of code, stub, used to locate the appropriate memory resident library routine stub replaces itself with the address of the routine, and executes the routine.
- operating system needed to check, if routine is in processes memory address.
- Dynamic linking is particularly useful for libraries
- System also known as shared libraries.

Swapping :-

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. Backing store - fast disk large enough to accomodate copies of all memory images for all users; must provide direct

access to these memory images Roll out, roll in - Swapping variant used for priority-based scheduling Algorithms

- Lower priority process is swapped out so higher-priority process can be loaded and executed.

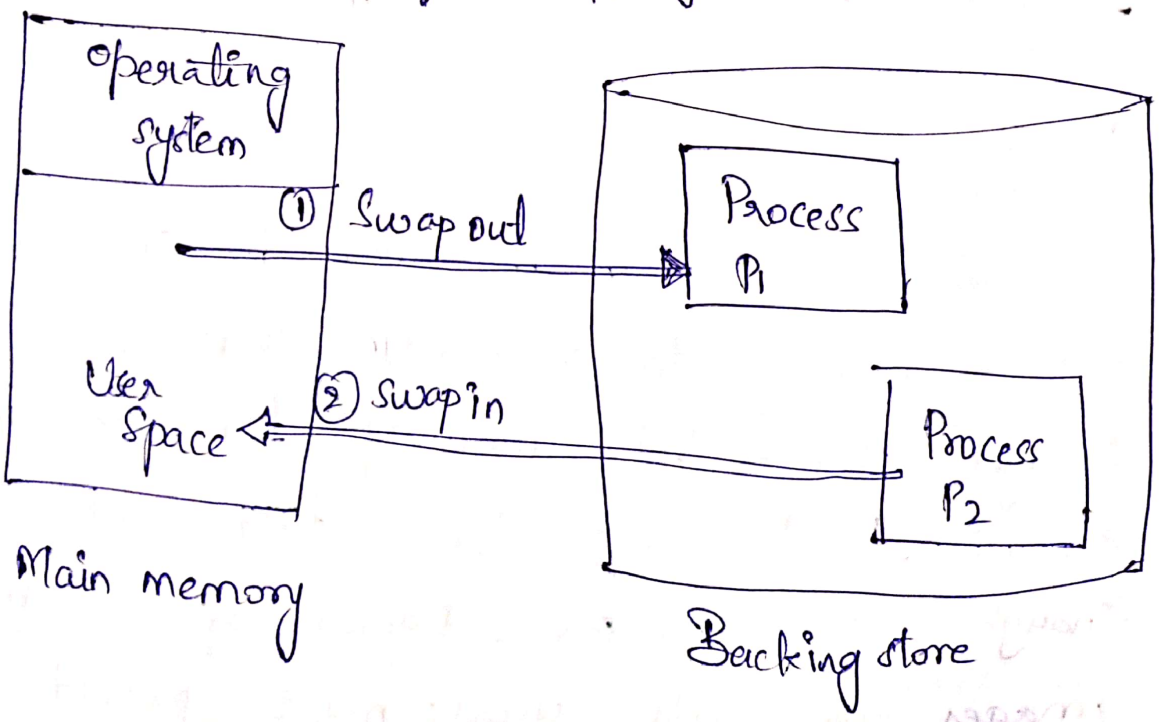
Major part of swap time is transfer time.

Total transfer time is directly proportional to the amount of memory swapped.

Modified versions of swapping are found on many systems (i.e. UNIX, Linux, & Windows)

Systems maintain a ready queue of ready to run processes which have memory images on disk.

Schematic view of swapping :-



Contiguous Allocation :- (Memory Allocation):

The main memory must accomodate both the operating system and the various user processes. Therefore, we need to allocate main memory in the most efficient way possible.

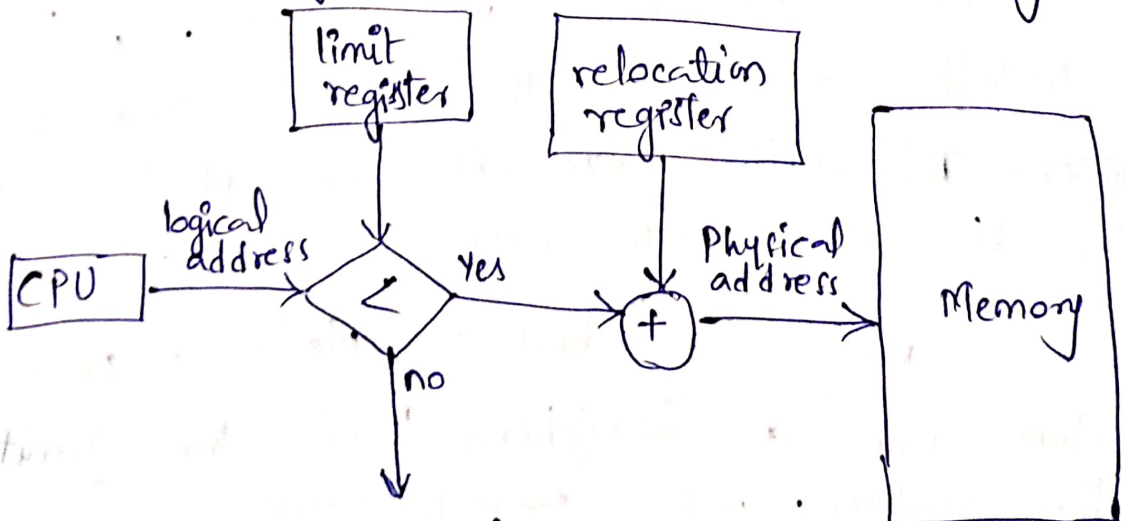
Main memory is usually divided into two partitions : one for the resident operating system and one for the user processes.

We can place the operating system and one in either low memory or high memory.

Major factor affecting this decision is the location of the interrupt vector.

Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well.

Operating system resides in low memory.



Memory protection :-

Memory protection is required to protect operating system from the user processes and user processes from one another.

A relocation register contains the value of the smallest physical address, for example say 100040. The limit register contains the range of logical address for example say 74600.

Each logical address must be less than the limit register.

If a logical address is greater than the limit register, then there is an addressing error and it is trapped.

The limit register hence offers memory protection.

The Memory management unit maps the logical address dynamically i.e., at runtime, by adding the logical address to the value in relocation register. This added value is the physical address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values.

When a process terminates, it releases its memory, which the OS may then fill with another process from the input queue.

This procedure is a particular instance of the general dynamic storage allocation problem, which concerns how to satisfy a request of size n from a list of free holes.

There are many solutions to the problem. The first-fit, best-fit and worst-fit strategies are the ones most commonly used to select a free hole from the set of available holes.

First fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first fit search ended. We can stop searching as soon as we find a free hole that is large enough.

Best fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

Worst fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

Memory allocation :-

Simplest method for allocating memory is to divide memory into several fixed size partitions.

Each partition may contain exactly one process. Thus, the degree of multi-programming is bound by the number of partitions.

In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition.

When the process terminates, the partition becomes available for another process.

In the variable-partition scheme, the OS keeps a table indicating which part of memory are available and which are occupied.

Initially, all memory is available for user processes and is considered one large block of available memory, a hole. Memory contains a set of holes of various sizes.

As processes enter the system, they are put into an input queue. The OS takes into account the memory requirements of each processes and the amount of available memory space in determining which processes are allocated memory.

When a process is allocated space, it is loaded into memory, and it can then compete for CPU time

Fragmentation :-

Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.

As processes are loaded and removed from memory, free memory space is broken into little pieces.

External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contagious. contiguous.

Storage is fragmented into a large number of small holes.

This fragmentation problem can be severe.

In worst case, we could have a block of free memory between every two processes. If all these small pieces of memory were big in one big free block instead, we might be able to run several more processes.

Whether we are using the first-fit or best-fit strategy can affect the amount of fragmentation.

Another factor is which end of a free block is allocated.

No, matter which algorithm is used, however external fragmentation will be a problem.

Memory fragmentation can be internal as well as external.

Consider a multiple partition allocation scheme with a hole of 18464 bytes.

Suppose that the next process requests 18462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes.

The overhead to keep track of this hole will be substantially larger than the hole itself.

The general approach to avoiding this problem is to break the physical memory into fixed sized blocks and allocate memory in units based on block size.

With this approach, the memory allocated to a process may be slightly larger than the requested memory.

The difference between these two numbers is internal fragmentation — unused memory that is internal to a partition.

One solution to the external fragmentation is Compaction.

The goal is to shuffle the memory contents so as to place all free memory together in one large block.

Compaction is not always possible, however.

If relocation is static and is done at assembly or load time, compaction cannot be done.

It is possible only if relocation is dynamic and is done at execution time.

Paging and Segmentation :- (Non-Contiguous Allocation)

Paging and segmentation are the two ways which allow a processes physical address space to be non-contiguous.

It has advantage of reducing memory wastage but it increases the overheads due to address translation.

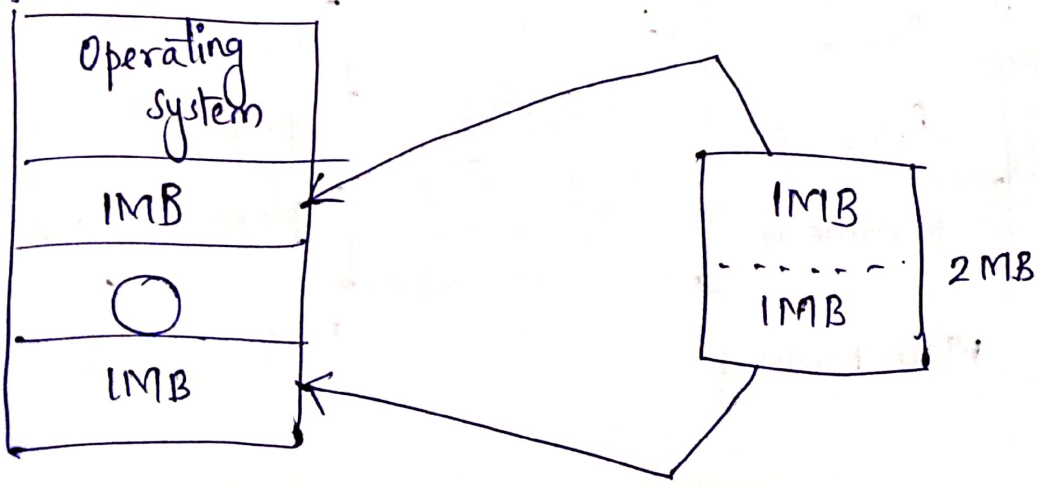
It slows down the execution of the memory because the time is consumed in address translation.

In Non-contiguous allocation OS need to maintain the table which is called page table for each process which contains the base address of each block which is acquired by the process in memory space.

Different parts of process allocated at different places in main memory.

Paging :-

The idea behind the paging is to divide the pages so that, we can store them in the memory at different holes.



In OS, the paging is a storage mechanism used to retrieve process from the secondary storage in the main memory in the form of pages.

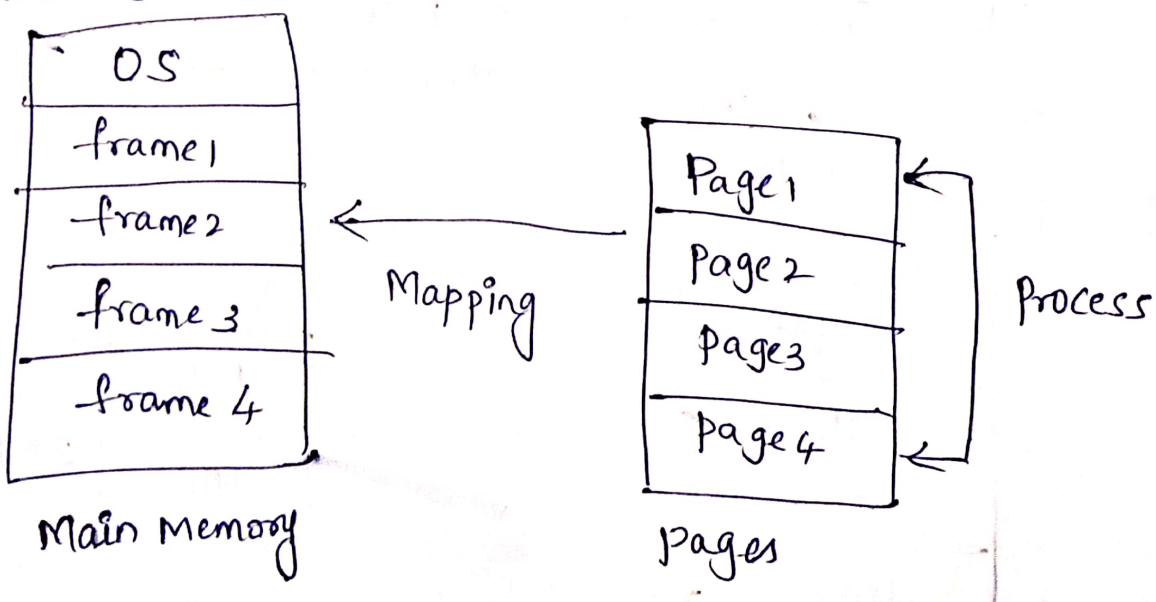
It divides each process in the form of pages. The main memory will also be divided in the form of frames.

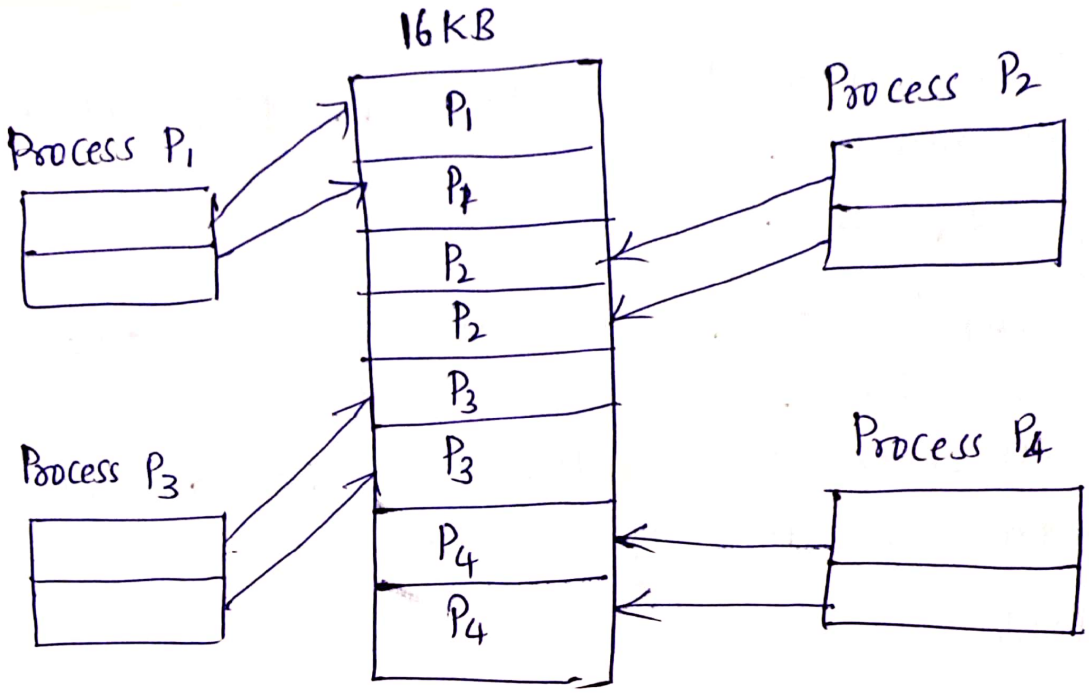
One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find contiguous frames or holes.

Pages of the Process are brought into the main memory when they are required. Otherwise they reside in the secondary storage.

Different OS defines different frame sizes. The sizes of each frame must be equal.

Considering the fact the pages are mapped to the frames in paging, page size need to be as same as frame size.





Paging Example.

Memory Management Unit (MMU):-

The purpose of 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 OS needs to obtain the physical address to access the page physically.

The logical address has two parts:

1. page number
2. offset.

MMU of OS needs to convert the page number into frame number.

Binary addresses :-

Computer assigns the binary addresses to the memory locations.

However, the system uses the amount of bits to address a memory location

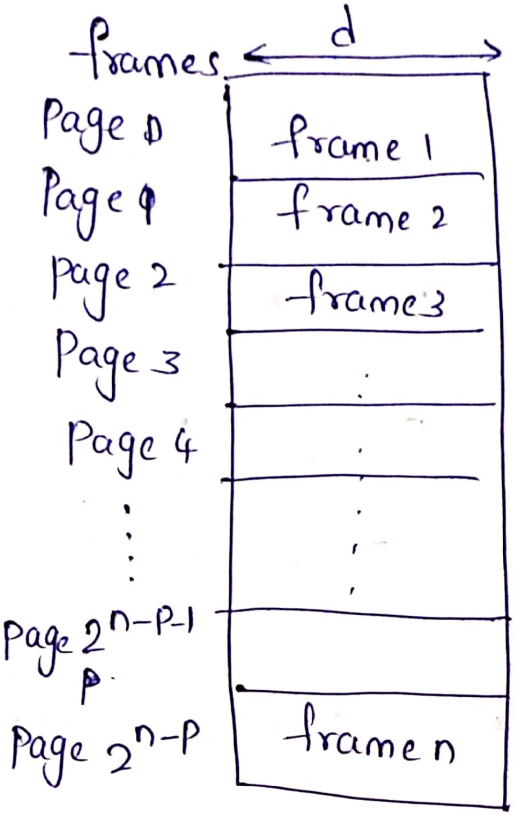
Page table :-

Page table is a data structure used by the virtual memory system to store the mapping between the logical address space and physical address space.

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

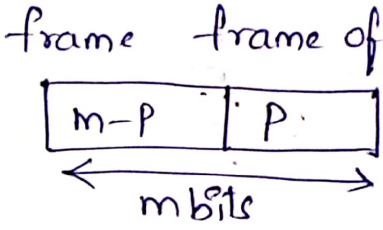
Physicall addresses are the actual frame address of the memory. They are generally used by the hardware or more specifically by RAM subsystems.

Physical address space = m words
Logical address space = n words
page size = p words.
Physical address = $\log_2 M$ = m bits
Logical address = $\log_2 n$ = n bits
Page offset = $\log_2 P$ = p bits.

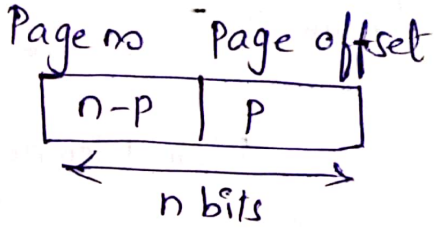


Page table

P.A :



L.A :



d = m-p (frame size) bits.

No. of entries in the page = no. of pages in the process.

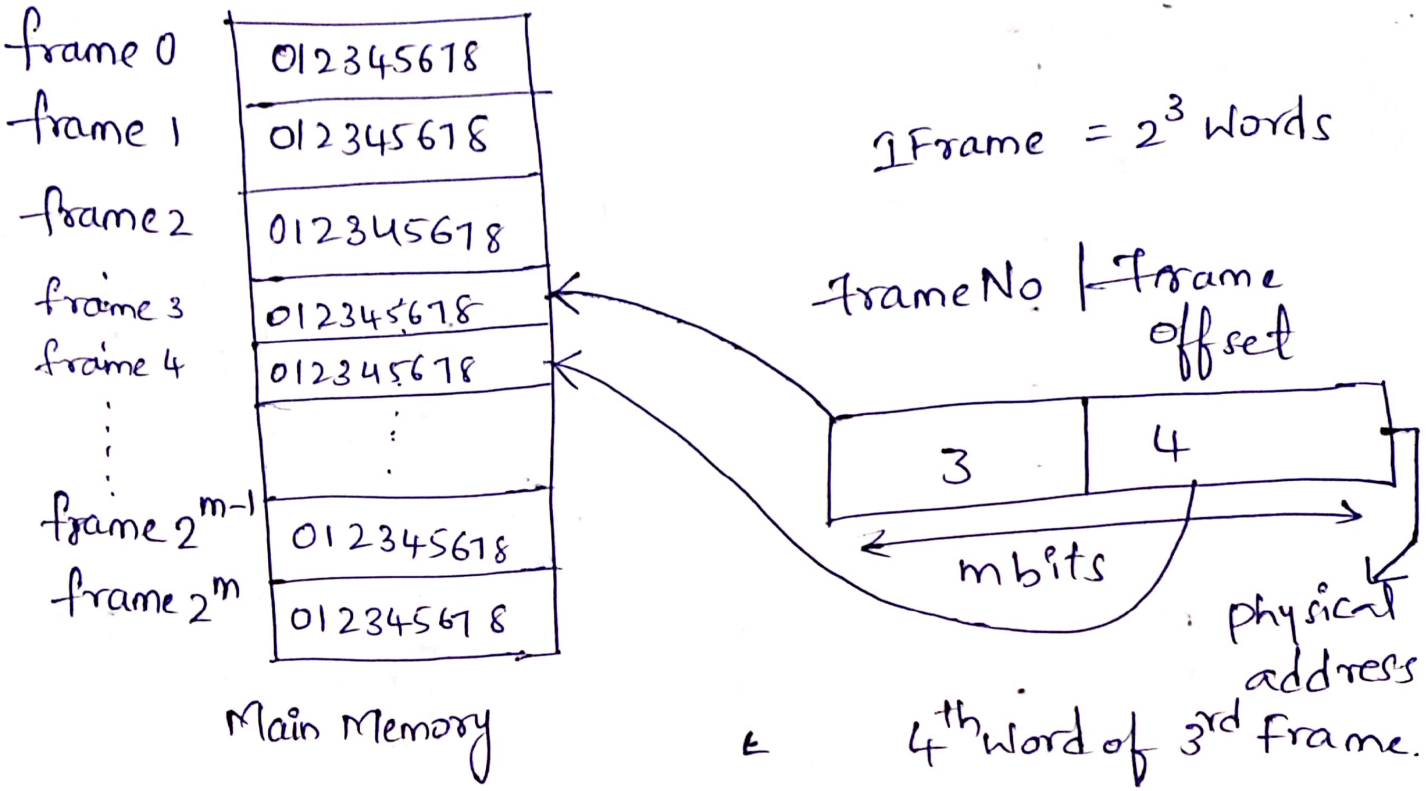
Pagetable size = 2^{n-p} x d bytes.

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

In this situation MMU comes into picture.

It converts the page number of the logical address to the frame number of the physical address. The offset remains the same.

To perform this task, MMU needs a special kind of mapping which is done by page table.

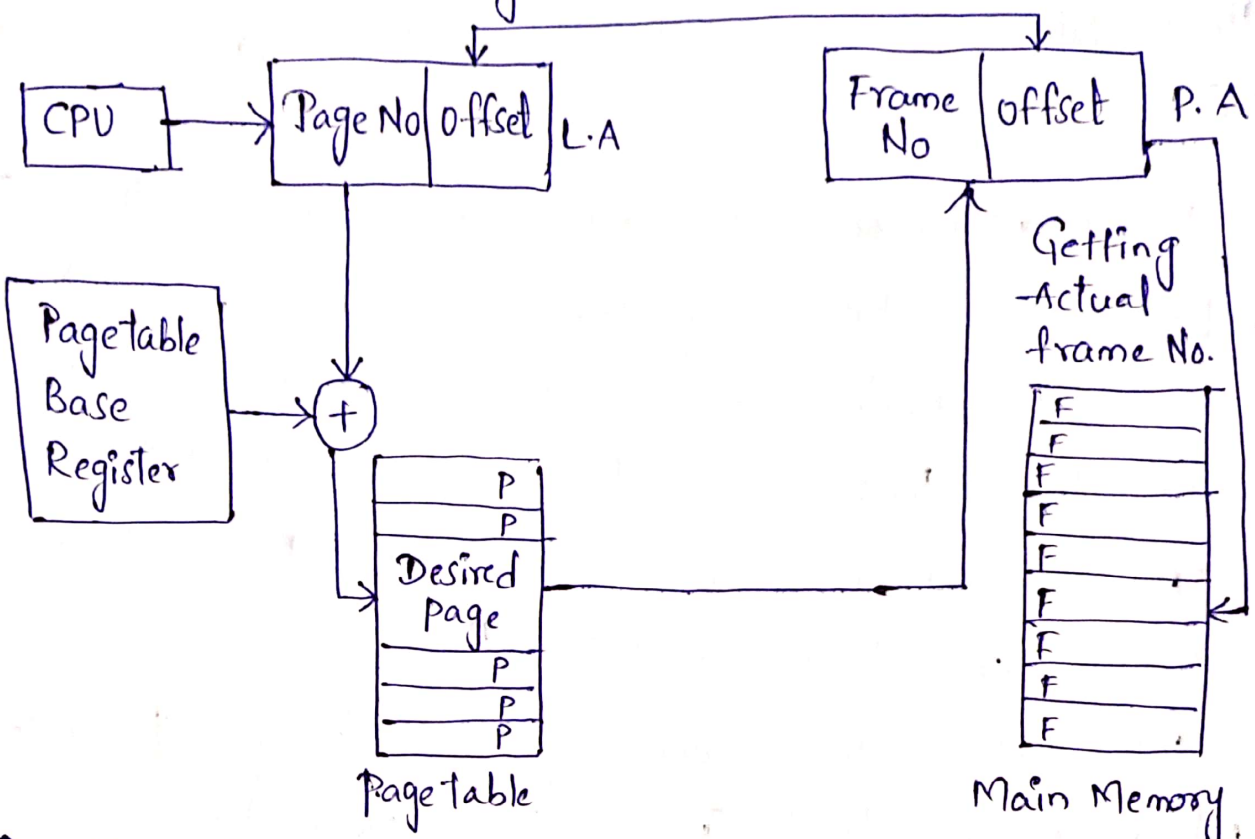


Mapping from page table to Main Memory :-

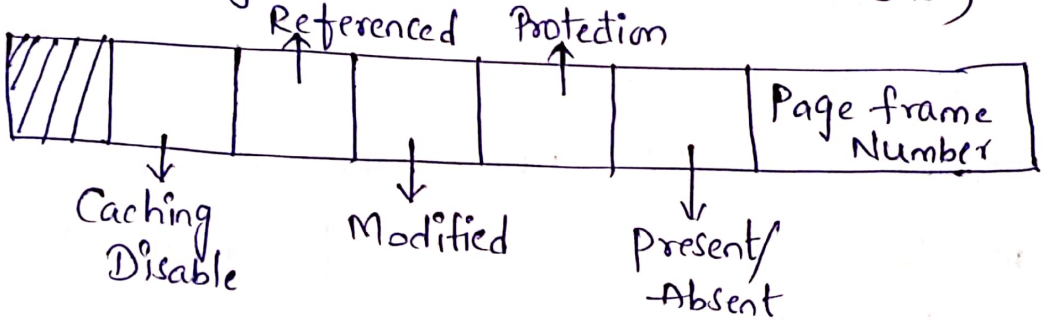
In OS, there is always a requirement of mapping from logical address to physical address.

The Mapping involves certain steps :

1. Generation of logical address
2. scaling.
3. Generation of physical address.
4. Getting actual frame number.



Page Table Entry :- (Contains some Extra bits)



Translation Look aside buffer :-

The drawbacks of paging are :

- Size of page table can be very big and therefore it wastes main memory.
- CPU will take more time to read a single word from the main memory.

A Translation Look aside buffer can be defined as a memory cache 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 the CPU to access the page table again and again TLB which is lesser than that taken to access main memory.

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.

TLB hit is a condition where the desired entry is found in TLB. If entry is not found in TLB then it is TLB miss.

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

Therefore effective access time can be defined as

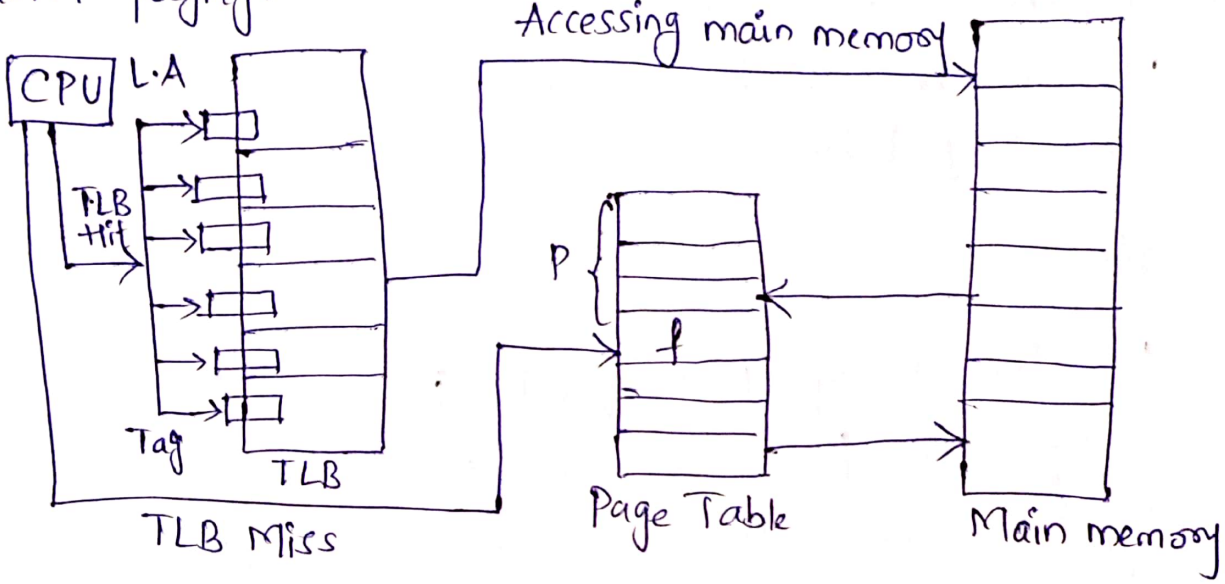
$$E.A.T = P(t+m) + (1-P)(t + k.m + m)$$

where E.A.T - Effective access time
P - TLB Hit rate
(1-P) - TLB miss rate.
t - time taken to access TLB
m - time taken to access memory.

By this formula we came to know that

1. E.A.T will be decreased if TLB hit rate is increased.

2. E.A.T will be increased in case of multi-level paging.



Locality of Reference :-

In OS, 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.

What is 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.

Inverted page table :-

It is the global page table which is maintained by the OS for all the processes.

In Inverted page table the no. of entries is equal to the no. of frames in the main memory.

It can be used to overcome the drawbacks of the page table.

Thrashing :-

If the no. of page faults is equal to the no. of referred pages or the no. of page faults are so high so that the CPU remains busy in just reading the pages from the secondary memory and it will 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. This concept is called thrashing.

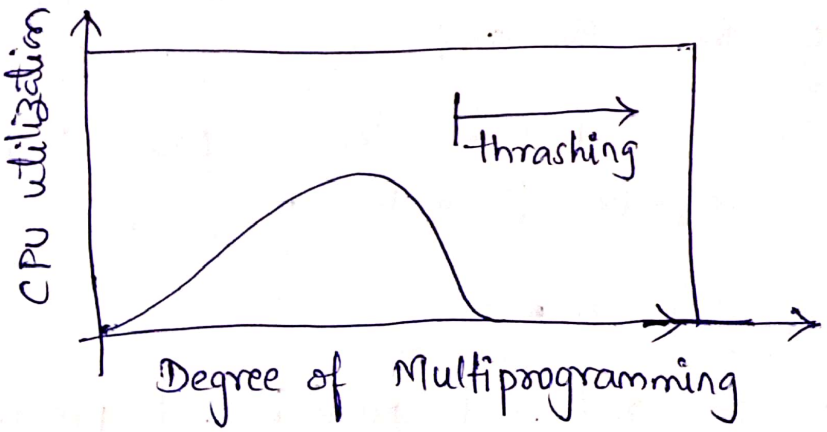


fig :- Thrashing

In fact, look at any process that does not have "enough" frames. If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault.

At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again and again, and again, replacing pages that it must bring back in immediately.

This high paging activity is called Thrashing. A process is thrashing if it is spending more time paging than executing.

Thrashing results in severe performance problems.

Consider the following scenario, which is based on the actual behaviour of early paging systems.

The OS monitors CPU utilization. If the CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system.

A global replacement algorithm is used. It replaces pages without regard to the process to which they belong.

Now suppose a process enters a new phase in its execution and needs more frames. It starts faulting and taking away frames from other processes.

These processes need those pages however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap in and out.

As they queue up for the paging device, the ready queue empties.

As processes wait for the paging device, CPU utilization decreases.

The CPU Scheduler sees the decreasing CPU utilization and increases the degree of multi-programming as a result.

The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device.

As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more.

Thrashing has occured and system throughput plunges.

The page-fault rate increases tremendously.

As a result, the effective memory access time increases.

No working is getting done, because the processes are spending all their time paging.

We can limit the effects of thrashing by using a local replacement algorithm.

Segmentation :-

In OS, 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 process.

The details about each segment are stored in a table called a segment table.

Segment table is a table contains mainly information about two segments.

1. Base : It is the base address of the segment.
2. Limit : It is the length of the segment.

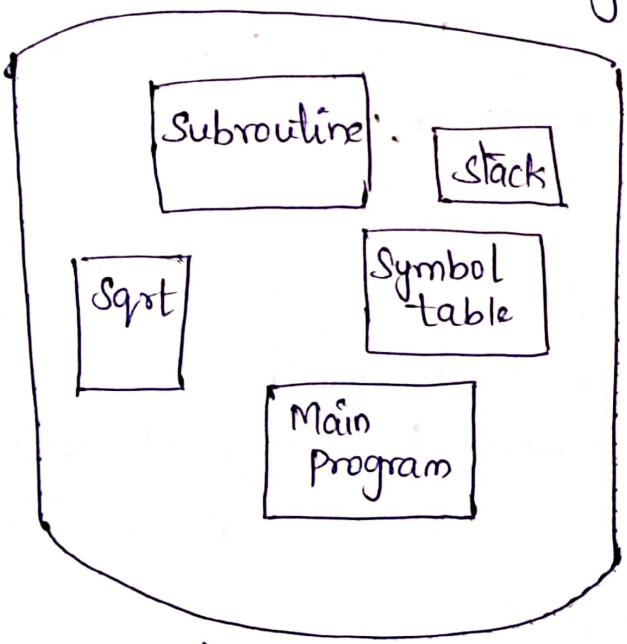


fig : Programmer's view of a program.

Segmentation is a memory management scheme that supports the programmer view of memory.

A logical address space is a collection of segments.

Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment.

The programmer therefore specifies each address by two quantities: a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.

Thus, a logical address consists of a two tuple:

<segment_number, offset>

Segmentation Hardware :-

Translation of logical address into physical address by segment table. (or Mapping)

Each entry in the segment table has a segment base and a segment limit.

The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

A CPU generates a logical address which contains two parts : a segment number s, and an offset into that segment, d.

The segment number is used as an index to the segment table.

Segment number is mapped to the segment table.

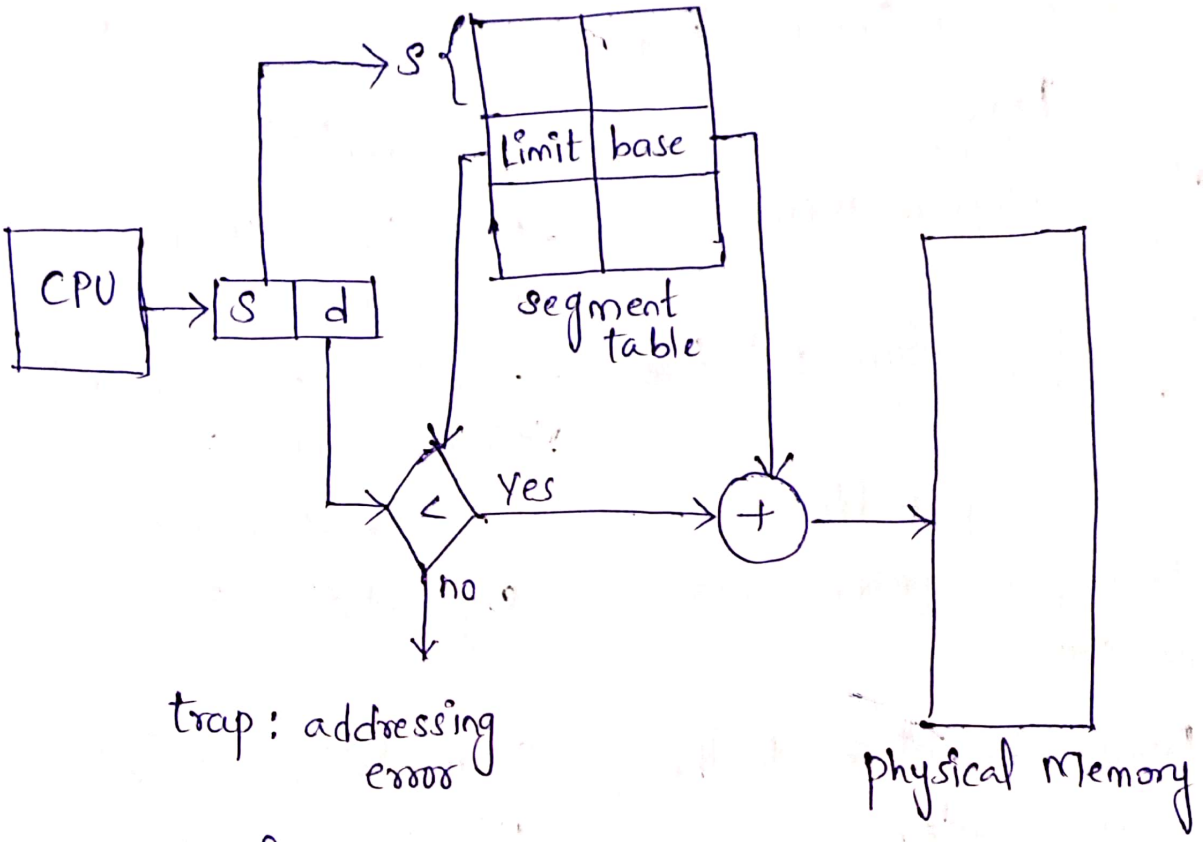


fig: Segmentation Hardware.

Advantages :-

- No internal fragmentation.
- Average segment size is larger than actual page size.
- Less overhead.
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as Compare to the pagetable in paging.

Disadvantages :-

- It can have external fragmentation.
- It is difficult to allocate contiguous memory to variable sized partition.

— Costly Memory Management Algorithms.

Difference between Paging and Segmentation ?

Paging	Segmentation.
1. Non-Contiguous memory allocation	- Non-Contiguous memory allocation.
2. Dividing program into fixed size pages.	- Dividing program into Variable size segments.
3. OS is responsible	- Compiler is responsible.
4. paging is faster than Segmentation	- Segmentation is slower.
5. Paging is closer to the Operating system	- Segmentation is closer to user.
6. It suffers from Internal fragmentation	4. It suffers from external fragmentation.
7. Logical address is divided into page no and offset.	- Logical address is divided into segment number and offset.
8. 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 of segment.

Segmented paging :- (Segmentation with paging) :-

Pure Segmentation is not very popular and not being used in many of the operating systems.

However, the segmentation combined with the 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 sized 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 appropriate segment Number.

Page Number : It points to the exact page within the segment.

page offset : used as an offset within the page frame.

Translation of logical address to physical address :-

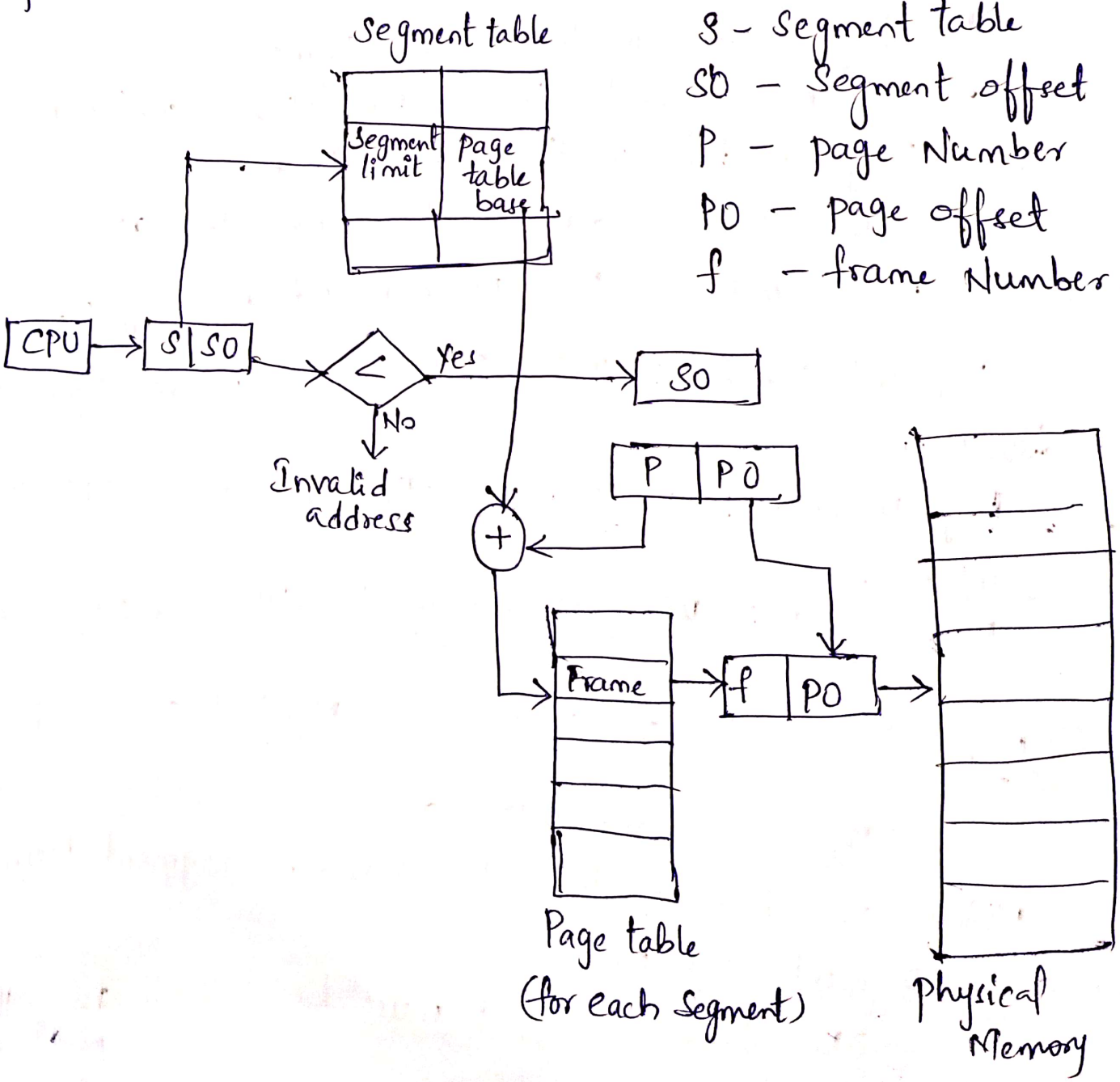
The CPU generates a logical address which is divided into two parts. Segment Number and segment offset.

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 within the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



Virtual Memory Management :-

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

User can load the bigger size processes than the available main memory by having an illusion that the memory is available to load the process.

By doing this, the degree of multi-programming increased and therefore, the CPU ulilization will also be increased.

How virtual memory works :-

In modern world, the 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 unlimited RAM.

Advantages of Virtual Memory :-

- Degree of multiprogramming will be increased
- User can run large applications with less real RAM.
- There is no need to buy more RAM memory.

Disadvantages of virtual memory management :-

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

Demand paging :-

Demand paging is a popular method of virtual memory management.

In demand paging, the page 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 are replaced.

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 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 the pages of the frames in the secondary memory they are required.

In other words, it says that do not load any page in the main memory until it is required.

Page replacement :-

Page replacement is needed in the operating systems that use virtual memory using Demand paging. As we know that in Demand paging, only a set of pages of a process is loaded into the memory. This is done so that we can have more processes in the memory at the same time.

When a page that is residing in virtual memory is requested by a process for its execution, the operating system needs to decide which page will be replaced by this requested page. This process is known as page replacement and is a vital component in virtual memory management.

Why we need page replacement algorithms :-

To understand why we need page replacement algorithms, we first need to know about page faults. Let's see what is a page fault.

Page Fault :- A page fault occurs when a program running in CPU tries to access a page that is in the address space of that program, but the requested page is currently not loaded into the main physical memory, the RAM of the system.

Since the actual RAM is much less than the virtual memory the page faults occurs, the operating system has to replace an existing page in RAM with the newly requested page.

In this scenario, page replacement algorithms help the operating system in deciding which page to replace. The primary objective of all the page replacement algorithms is to minimize the number of page faults.

Page replacement Algorithms in OS :-

- FIFO
- LRU
- optimal.

FIFO : [First-in-first-out] :-

FIFO algorithm is the simplest of all the page replacement algorithms.

In this, we maintain a queue of all the pages that are in the memory currently.

The oldest page in the memory is at the front-end of the queue and the most recent page is at the back or rear end of the queue.

Whenever a page fault occurs, the operating System looks at the front end of the queue to know the page to be replaced by the newly requested page. It also adds this newly requested page at the rear end and removes the oldest page from the front end of the queue.

Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames. Lets try to find the number of page faults.

Pages →	3	1	2	1	6	5	1	3
Frames →			2	2	2	5	5	5
		1	1	1	1	6	6	3
	3	3	3	3	6	2	1	1
	m	m	m	H	m	m	m	m

Pagefaults = 7.

Queue →

3	3	3	3	1	2	6	5
	1	1	1	2	6	5	1
		2	2	6	5	1	3

→ Initially all slots are empty so page faults occur at 3, 1, 2.
Pagefaults = 3

→ When page 1 comes, it is in the memory so no page fault occurs.
Page faults = 3.

→ When page 6 comes, it is not present and a page fault occurs. Since there are no empty slots, we move the front of the queue i.e 3.
Page faults = 4

→ When page 5 comes, it is not present and hence a page fault occurs. The front of the queue i.e., 1 is removed.
Page faults = 5.

→ When page 1 comes, it is again not found in memory, again a page fault occurs. The front of the queue i.e 2 is removed.
Page faults = 6.

→ When page 3 comes, it is again not found in memory, a page fault occurs, and page 6 is removed being on top of the queue.

Total page faults = 7.

Belady's Anamoly :-

Generally if we increase the number of frames in the memory, the number of page faults should decrease to obvious reasons.

Belady's Anamoly refers to the phenomena where increasing the number of frames in memory, increases the page faults as well.

In FIFO's page replacement algorithm, the no. of page faults will gets increased with the increment in no. of frames.

Advantages :-

- Simple to understand and implement.
- Does not cause more overhead.

Disadvantages :-

- poor performance
- Doesn't use the frequency of the last used time and just simply replaces the old page.
- suffers from Belady's Anamoly.

Optimal page replacement algorithm in Operating system :-

optimal page replacement is the best page replacement algorithm as this algorithm results in the least number of page faults.

In this Algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future.

In simple terms, the pages that will be referred farthest in the future are replaced in this algorithm.

Eg: Lets take the page reference string 3, 1, 2, 1, 6, 5, 1, 3 with 3 page frames as we saw in FIFO.

This also helps you understand how optimal page replacement works the best.

Pages →	3	1	2	1	6	5	1	3
frames →			2	2	6	5	5	5
		1	1	1	1	1	1	1
	3	3	3	3	3	3	3	3
	m	m	m	H	m	m	H	H

Initially, since all the slots are empty, pages 3, 1, 2 cause a page fault and take the empty slots.

Page faults = 3

→ When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

→ When page 6 comes, it is not in the memory, so a page fault occurs and 2 is removed as it is not going to be used again.

Page faults = 4.

→ When page 5 comes, it is also not in the memory and cause a page fault. Similar to above 6 is removed as it is not going to be used again.

Page faults = 5.

→ When page 1 and page 3 come, they are in the memory so no page fault occurs.

Total page faults = 5.

Advantages :

- Excellent efficiency.
- Less complexity.
- Easy to use and understand.
- Simple data structures can be used to implement.
- Used as the benchmark for other algorithms.

Disadvantages :

- More time consuming.
- Difficult for error handling.
- Need future awareness of the programs, which is not possible everytime.

Least Recently Used Algorithm (LRU) :-

The Least Recently used page replacement algorithm keeps the track of usage of pages over a period of time.

This algorithm works on the basis of the principle of locality of a reference which states that a program has a tendency to access the same set of memory locations repetitively over a short period of time.

So, pages that have been used heavily in the past are most likely to be used heavily in the future also.

In this algorithm, when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page.

Eg: Lets see the performance of the LRU on the same reference string of 3, 1, 2, 1, 6, 5, 1, 3 with 3 - page frames.

Pages →	3	1	2	1	6	5	1	3
frames →			2	2	2	2	1	1
		1	1	1	1	5	5	5
	3	3	3	3	6	6	6	3
	m	m	m	H	m	m	m	m

Initially, since all the slots are empty, pages 3, 1, 2 cause a page fault and take the empty slots.

Page faults = 3.

When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3.

When page 6 comes, it is not in the memory, so, a page fault occurs and least recently used page 3 is removed.

Page faults = 4

When page 5 comes, it again causes a page fault and page 1 is removed as it is now the least recently used page.

Page faults = 5

When page 1 comes again, it is not in the memory and hence page 2 is removed according to LRU.

Page faults = 6

When page 3 comes, the page fault occurs again and this time page 6 is removed as the least recently used one.

Total page faults = 7.

Now in the above example, the LRU causes the same Page faults as the FIFO, but this may not always be the cas as it will depend upon the series the number of frames available in memory, etc. In fact, on most occasions, LRU is better than FIFO.

Advantages :-

- It is open for full analysis.
- Doesn't suffer from Belady's Anamoly.
- often more efficient than other algorithms.

Disadvantages :-

- It requires additional data structures to be implemented.
- More Complex.
- High hardware assistance is required.