Unit-4

Memory Management

Memory management is a functionality of operating system which manages primary memory. It keeps track of each and every memory allocation either it is allocated to some process or not. It also decides which process will get memory at what time. It tracks whenever some memory get freed or unallocated and correspondingly updates the state. The part of operating system that manages the memory hierarchy is called the memory manager.

⊛. Monoprogramming vs Multiprogramming:

Monoprogramming	Multiprogramming
i) In monoprogramming memory contains only one program at any point of time.	i) In multiprogramming memory contains more than one user program.
ii) In monoprogramming CPU executes the program and I/O operation is encountered, during that time CPU sits ideal. So, CPU is not efficiently used.	ii) In multiprogramming when one user program contains I/O operation CPU switches to next user program. So, CPU is efficiently used.
iii) In monoprogramming main memory needs less space as only one program sits in main memory.	iii) In multiprogramming main memory needs more space as it contain more than one program in its main memory.
iv) Fixed size partition is used in monoprogramming.	iv) Both fixed and variable size partition can be used in multiprogramming.
Example: Old mobile operating systems	Example: Windows 7,8,10.
Advantages: → Allocation of memory is easy & cheap. → Eliminates external fragmentation. → More efficient swapping.	Advantages: → CPU never becomes ideal. → Supports multiple users. → Resources are wisely used.
Disadvantages: → Longer memory access time. → Internal fragmentation. → Inverted page tables.	Disadvantages: → Difficult to program a system. → Tracking of all tasks is sometimes difficult to handle.

⊛. Modeling Multiprogramming:-

Assume that a process spend p% of its time waiting for I/O. With n processes in memory the probability that all processes are waiting for I/O (meaning the CPU is idle) is p^n. The CPU utilization is then given by;

$$CPU.\ utilization = 1 - p^n$$

Example:- If p=0.5 & n=4 then,

$$1-p^n = 1-0.5^4$$
$$= 0.9375$$

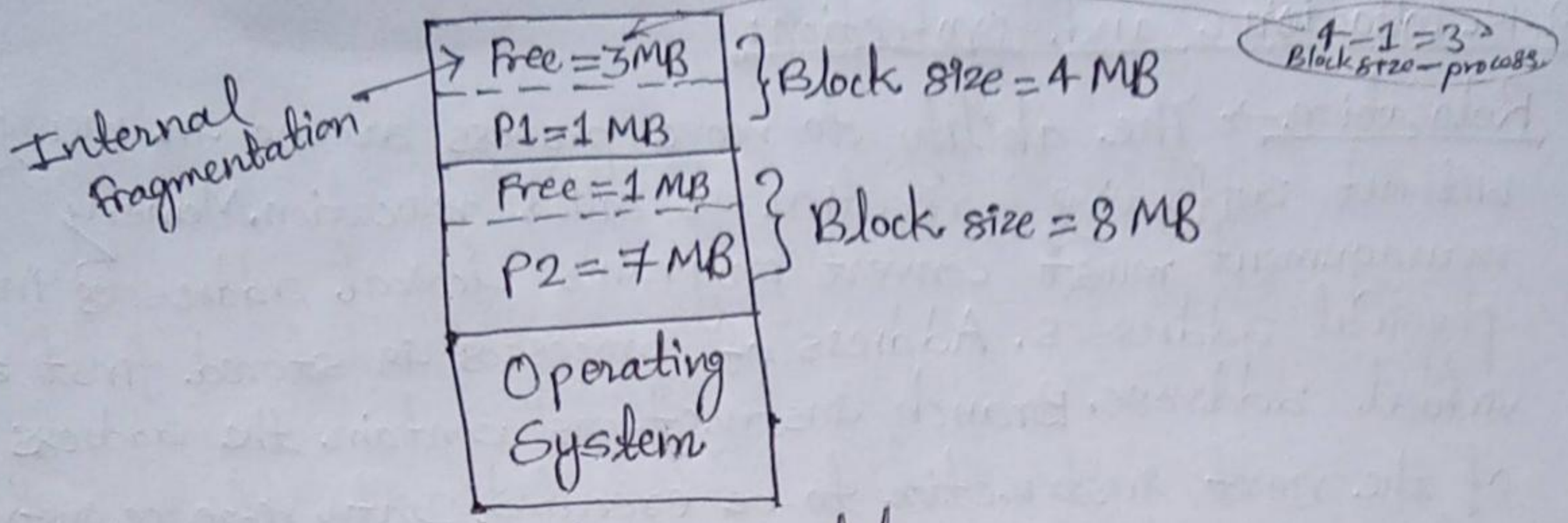
Note: In monoprogramming CPU utilization is 1-p. p is often >0.5 which shows CPU utilization is poor in monoprogramming. But above example of multiprogramming shows that CPU utilization is much better.

⊛. Multiprogramming with fixed and variable partitions;

In multiprogramming environment several programs reside in primary memory at a time and CPU passes its control rapidly between these programs. One way to support multiprogramming is to divide the main memory into several partitions each of which is allocated to single process. Depending on how and when partition are created there may be two types of partition: static & dynamic. (i.e, fixed & variable).

a) Fixed partition:

→ Memory is divided into several fixed size partition where each partition will accomodate only one program for execution.
→ The number of programs residing in the memory will be bounded by the number of partition.
→ When a program terminates the partition is freed for another program waiting in queue.
→ When job arrives it can be put into the input queue for the smallest partition large enough to hold it.
→ Since, the partitions are fixed in this schema, any space in a partition can not be used by job is wasted while that job runs.



Fixed size partition

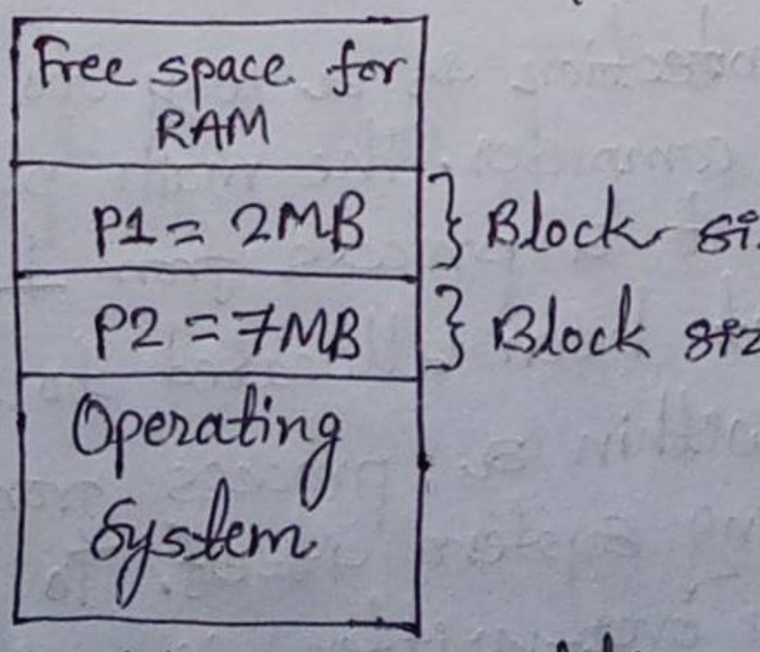
Advantages:
- → Easy to implement.
- → Little OS overhead.
- → Efficient utilization of process and I/O devices.

Disadvantages:
- → Internal & External Fragmentation.
- → Limit process size.
- → Limitation on degree of multiprogramming.

b) Dynamic partition: (variable partition):

→ Memory management approach is to create partition dynamically to meet the requirements of each requesting process, this technique is called variable partitioning.

→ When a process terminates or is swapped out then the memory manager can return the vacant space to pull off free memory area from which partition allocation are made.



variable size partition

Advantages:
- → No internal fragmentation.
- → No limitation on process size.
- → No limitation on degree of multiprogramming.

Disadvantages:
- → Difficult implementation.
- → External fragmentation.

⊛. Relocation and protection:

Relocation → The ability to move process around in memory without affecting execution is called relocation. Memory management must convert programs logical addresses into physical addresses. Address of processes is stored first as virtual address. Branch instructions contain the address of the next instruction to be executed. Data refrence instructions contain the address of byte or word of data referenced.

There are two types of relocations: static and dynamic. In static relocation program must be relocated before or during loading of process into memory. Program must always be loaded into same address space in memory, or relocator must be run again. In dynamic relocation process can be freely moved around in memory. Virtual-to-physical address space mapping is done at run-time.

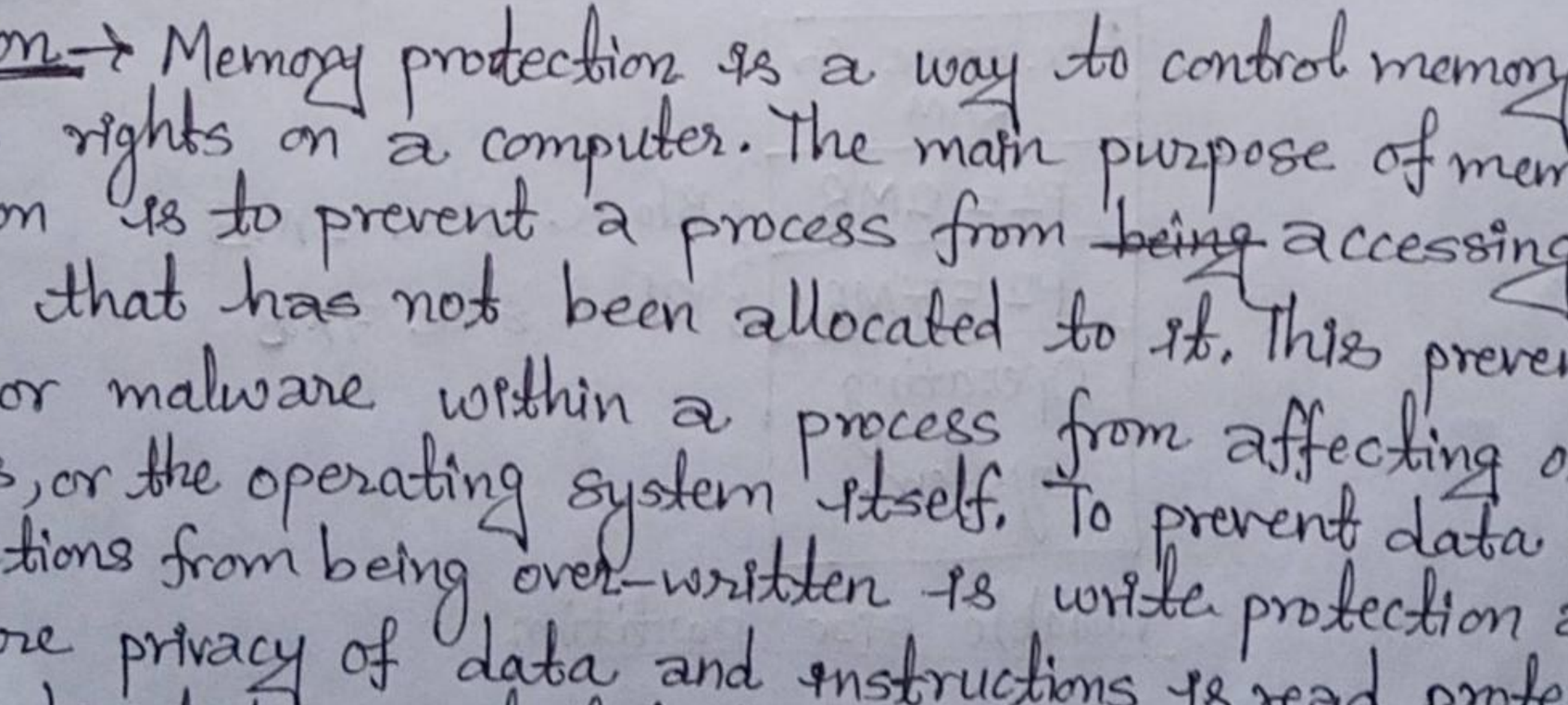


Fig: Memory relocation.

Protection → Memory protection is a way to control memory access rights on a computer. The main purpose of memory protection is to prevent a process from ~~being~~ accessing memory that has not been allocated to it. This prevents a bug or malware within a process from affecting other processes, or the operating system itself. To prevent data and instructions from being over-written is write protection and to ensure privacy of data and instructions is read protection. OS needs to be protected from user processes and user processes need to be protected from each other.

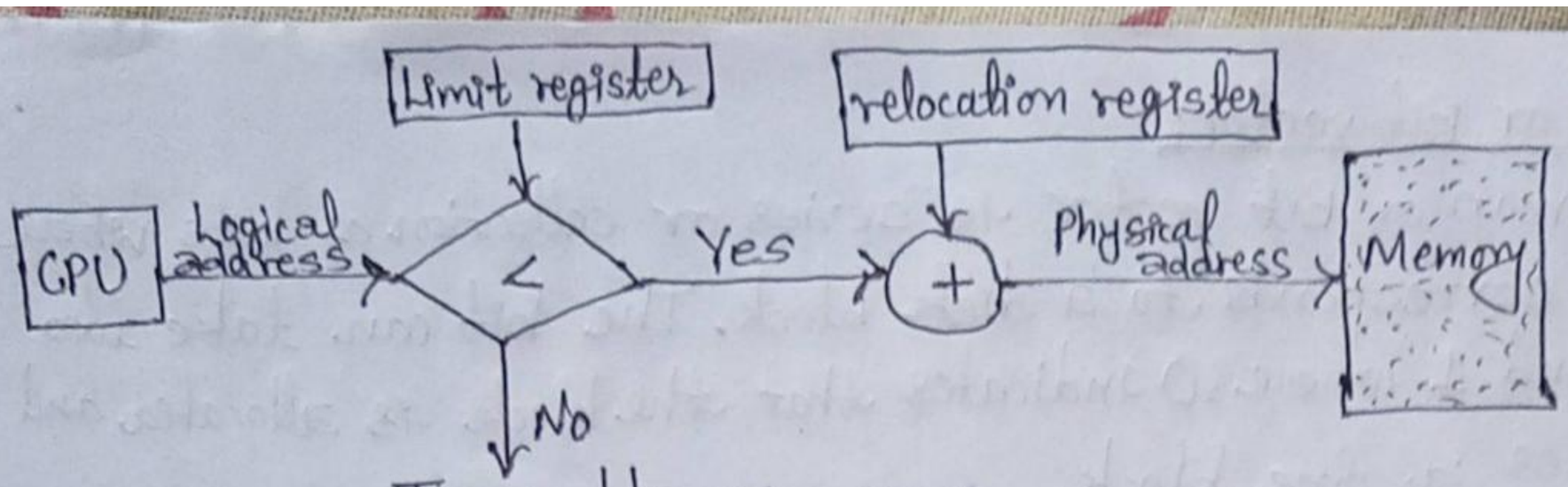


Fig. Memory protection and relocation.

#Space Management

⊛. Fragmentation:

As a process are loaded and removed from the memory, the free memory space is broken into little pieces. It happens after sometime that process cannot be allocated to memory block considering their small size and memory blocks remain unused, this problem is known as fragmentation.

There are two types of fragmentation: Internal and External. In Internal fragmentation memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. In External fragmentation total memory space is enough to satisfy a request or to reside a process in it, but it is not continious so it can't be used.

⊛. Compaction:

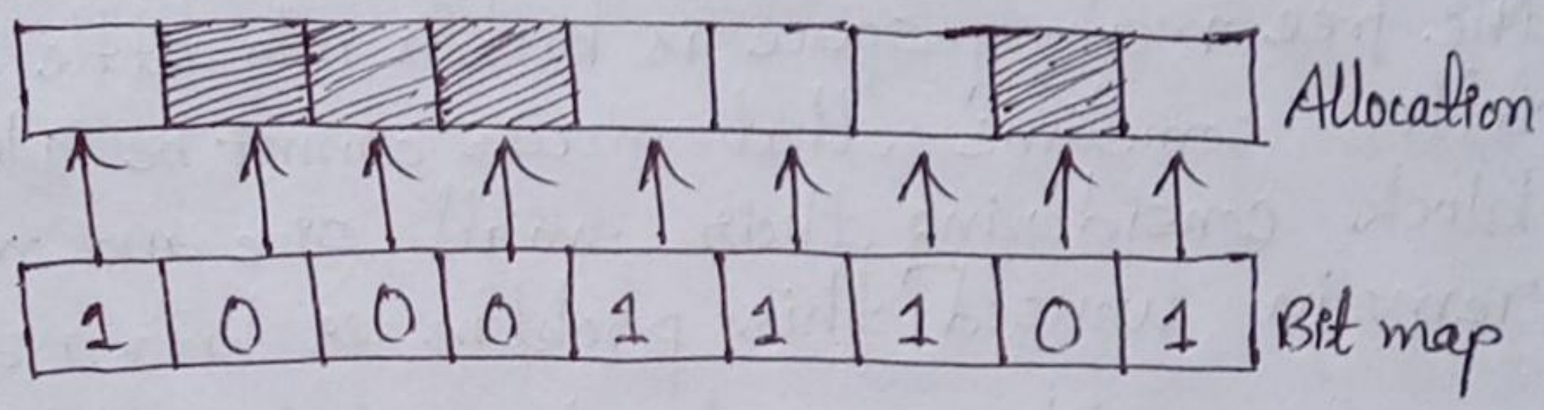
Compaction is a technique by which the programs are relocated in such a way that the small chunks of free memory space are made continious to eachother & clubbed together into a single free partition, that may be big enough to accomodate some more programs.

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.

⊛. Bitmap or Bit vector:

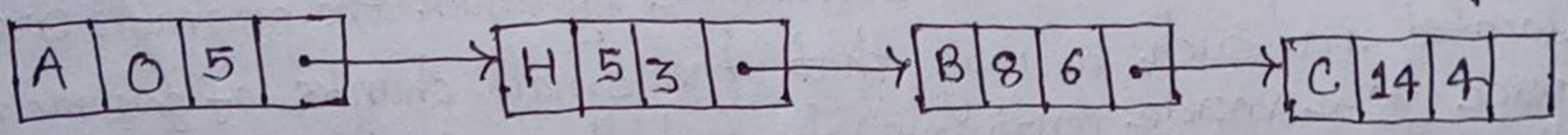
A Bitmap or Bit vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1 where, 0 indicates that the block is allocated and 1 indicates a free block.

The main problem with this scheme is the size of the allocation unit. The smaller the allocation unit, the larger the bit map has to be. But if we choose larger allocation unit, we may waste memory. The other problem with this scheme is to allocate memory to process. For this reason bit maps are not often used.



⊛. Linked Lists:

It is the another way to keep track of memory of allocated and free memory segments, where segment either contain a process or is a empty hole between two processes. In this approach, the free disk blocks are linked together i.e, a free block contains a pointer to the next free block. The block number of very first disk block is stored at a seperate location on disk and is also cached in memory.



⊛. Memory Allocation Strategies:-

a) first fit → In this approach the process manager scans along the list of segements until the first free partition with large enough hole which can accomodate the process is found. It finishes after finding the first suitable free partition.

Advantage → Fastest algorithm because it searches as little as possible.

Disadvantage → Wastage of remaining unused memory left after allocation.

b) Next fit → It works same as first fit, except that it keeps track of where it is, whenever it finds a suitable hole. The next time, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does.

this can be escaped

c) Best fit → The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first search the entire list of free partition and consider the smallest hole that is sufficient for the requesting process.

Advantage → Memory utilization is much better than first fit.

Disadvantage → It is slower & may even tend to fill up memory with tiny useless holes.

d) Worst fit → In worst fit, approach is to allocate largest available free partition so that the partition left will be big enough to be useful. It is the reverse of the best fit.

Advantage → Reduce the rate of production of small gaps.

Disadvantage → If a process requiring larger memory arrives at a later stage then it cannot be accomodated as the largest hole is already split & occupied.

Numerical Problem:- Given memory partitions of 100K, 500K, 200K, 300K and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K and 426K (in order)? Which algorithm makes the most efficient use of memory?

Solution:

For First-fit

1) 212K is put in 500K partition.
2) 417K is put in 600K partition.
3) 112K is put in 288K partion (new partition 288K = 500K - 212K)
4) 426K must wait.

For Best-fit

i) 212K is put in 300K partition.
ii) 417K is put in 500K partition.
iii) 112K is put in 200K partition.
iv) 426K is put in 600K partition.

For Worst-fit

i) 212K is put in 600K partition.
ii) 417K is put in 500K partition.
iii) 112K is put in 388K partition.
iv) 426K must wait.

⇒ In this example, Best-fit turnsout to be the best.

#Virtual Memory

Virtual memory is a technique of executing programs/instructions that may not fit entirely in the system memory. It is implemented with the help of secondry storage device by transferring data from the main memory to secondry memory and vice-versa. The main objective of this method is to provide multiprogramming. The use of virtual memory has its own limitations mainly with speed. So, it's generally better to have as much physical memory as possible so programs work directly from RAM or physical memory.

⊛. Paging :

Paging is a non-contiguous memory allocation technique which allows the physical address space of a process to be non-contigious. Logical address space is divided into equal size pages but physical address space is divided into equal size frames. A computer can address more memory than the amount physically installed on the system, this extra memory is actually called virtual memory and it is a section of a hard disk that set upto emulate the computers RAM.

Paging is a memory management technique in which process address space is broken into blocks of same size

called pages (size is power of 2). Similarly main memory is divided into small fixed size block of memory called frames & the size of frame is kept the same as that of page to have optimum utilization of the main memory & to avoid external fragmentation.

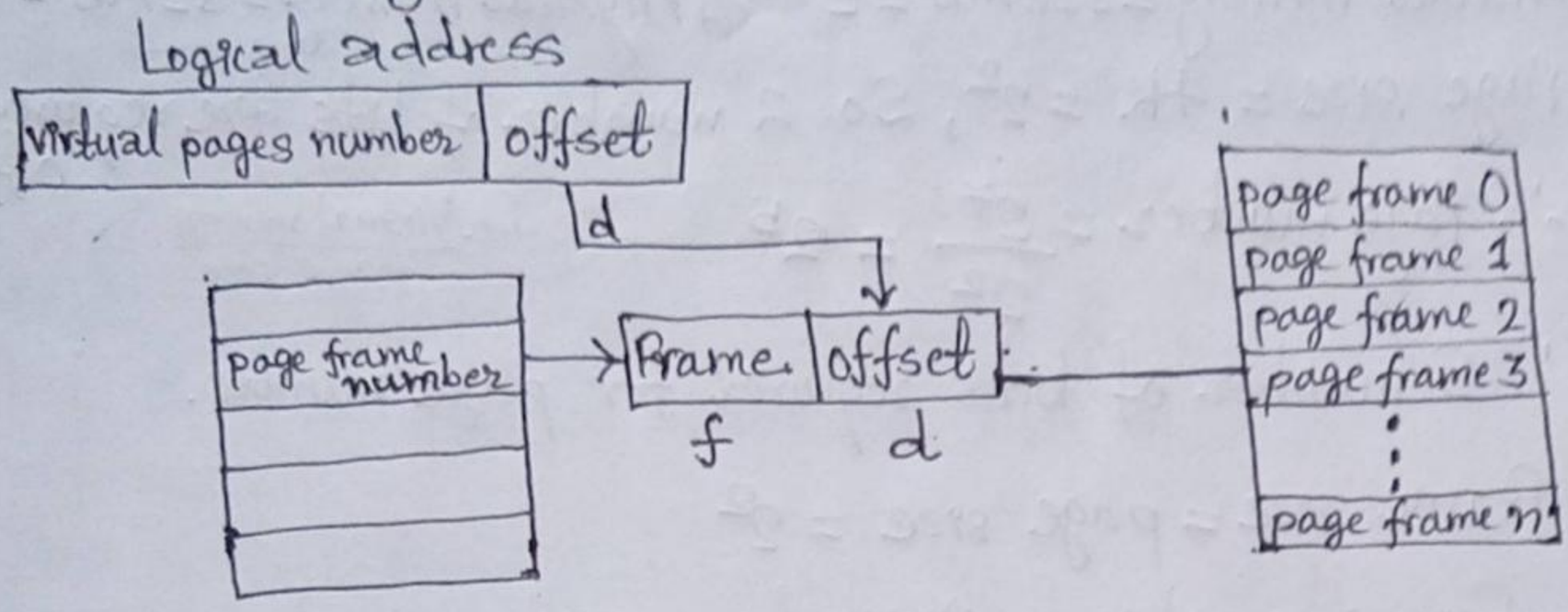


fig: paging hardware.

⊛. Page Table:

A page table is a data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. It holds information such as page number, offset, present/absent bit, modified bit etc. It acts as a bridge between virtual pages and page frames thus mathematically we can say that the page table is a function, with the virtual page number as argument and the physical frame number as result. Page table entry has the following information:

Frame Number	Present/Absent	Protection	Refrence	Caching	Dirty

The number of bits required depends on the number of frames. Number of bits for frame = $\frac{\text{Size of physical memory}}{\text{Frame size}}$.

Virtual address space = size of process.

Process size = 2^x bytes, no. of bits in virtual address space = x bits.

fraime size = page size.

Page number = $\frac{\text{virtual memory}}{\text{page size}}$

Q. Consider a virtual memory and physical memory of size 128 MB and 32 MB respectively. Assume that page size is 4K, what will be the number of bits required for the page number, frame number and offset?

Solution:

Virtual memory = 128 MB = 2^7 & Physical memory = 32 MB = 2^5

Page size = 4K = 2^2, So. 2 number of bits are required for offset.

∴ Page number = $\frac{2^7 K}{2^2} = 2^5$ (i.e. $\frac{\text{Virtual memory}}{\text{page size}}$)

∴ 5 number of bits required for page number.

Frame size = page size = 2^2

∴ Frame number = $\frac{\text{Physical memory}}{\text{frame size}}$

$= \frac{2^5}{2^2}$

$= 2^3$

∴ 3 number of bits required for frame number.

⊛. Handling Page Faults:-

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So, when page fault occurs then the following sequence of event happens:

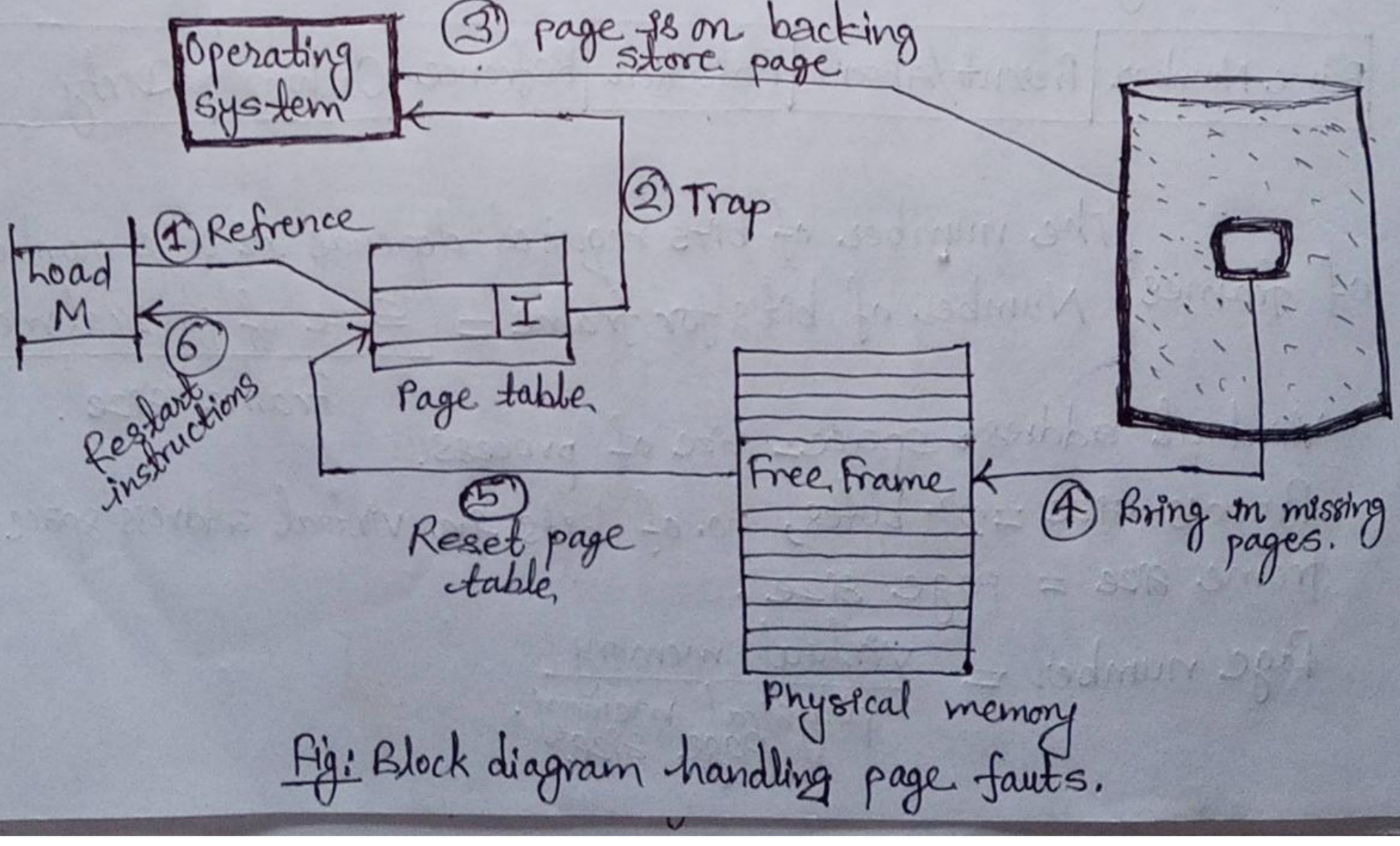


Fig: Block diagram handling page faults.

→ The memory address requested is first checked, to make sure it was a valid memory request.
→ If the refrence was invalid, the process is terminated. Otherwise, the page must be paged in.
→ A frame is located, possibly from a free-frame list.
→ A disk operation is scheduled to bring in the necessary page from disk to allow or block processes on an I/O.
→ When the I/O operation is complete, the processe's page table is updated with the new frame number and the invalid bit is changed to indicate that this is now a valid page refrence.
→ The instruction that caused the page fault must now be restarted from the beginning.

⊛ TLB's:

On the cache miss, there will be two memory accesses. Each virtual memory refrence can cause two physical memory accesses: one to fetch the page table and other to fetch the data. To overcome this problem a high-speed cache is set up for page table entries called a Translation Look aside Buffer (TLB). TLB is nothing but a special cache used to keep track of recently used transactions.

Fig: Paging hardware with TLB.

#Page Replacement Algorithms:

A page replacement algorithm is needed to decide which page needs to be replaced when new page comes in. Since physical memory is much smaller than virtual memory, page faults happen. So, OS might have to replace one of the existing page with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace so that no. of page faults may be reduced.

1) First In First Out (FIFO) Page Replacement Algorithm:

It is the simplest page replacement algorithm. The idea behind this is replace a page which is the oldest page of all pages of main memory.

For Example:- Refrence string sequence (7,0,1,2,0,3,0,4,2,3,0,3,1,2,0) (15 pages/refrences in disk) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
F_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
F_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
F_3			1	1	1	1	0	0	0	3	3	3	3	2	2
					H							H			H

(3 frames)

(Any page staying for longest time in memory is to be replaced)

(Since 3 is already present in memory so no page placed at same place replacing 2 late)

Note:

Hit → If demanded/requested page is already present in the main memory then we call it as hit. We denote hit by H as in above example. So, no. of page hits in above example are 3.

Page fault → (other than page hits i.e, page miss). There are 12 page faults in above example.

Hit ratio → $\frac{\text{no. of hits}}{\text{Total no. of refrence}}$

Miss/Page fault ratio → $\frac{\text{no. of page fault}}{\text{Total no. of refrence}}$

If we want Hit ratio or page fault ratio in percentage then we multiply result by 100.

Advantages:
→ Easy to understand and program
→ Distribute fair chance to all.

Disadvantages:
→ FIFO is likely to replace heavily used pages and they are still needed for further processing.
→ System needs to keep track of each frame.

*) Belady's anomaly:- Belady's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns. This phenomenon is commonly experienced when using FIFO page replacement algorithm.

For example:- If we consider refrence string 1,2,3,4,1,2,5,1,2,3,4,5 and no. of frames = 3, we get 9 total page faults, but if we increase and make no. of frames = 4, we will get 10 page faults.
[We can solve and show this by FIFO technique as we did before].

2) Second chance page replacement algorithms:

It is the modified form of the FIFO page replacement algorithm. One way to implement is to have circular queue. If the value is equal to 0, then we proceed to replace the page. But if the refrence bit is equal to 1, then we give the page second chance. When the page gets second chance, its refrence bit is cleared.

Example:- Consider refrence string 2,3,2,1,5,2,4,5,3,2,5,2 and no. of frames = 3, we get 7 total page faults by using this algorithm.

	2	3	2	1	5	2	4	5	3	2	5	2
F_1	2 (0)	2 (0)	2 (1)	2 (1)	2 (0)	2 (1)	2 (0)	2 (0)	3 (0)	3 (0)	3 (0)	3 (0)
F_2		3 (0)	3 (0)	3 (0)	5 (0)	5 (0)	5 (0)	5 (1)	5 (1)	5 (0)	5 (1)	5 (1)
F_3				1 (0)	1 (0)	1 (0)	4 (0)	4 (0)	4 (0)	2 (0)	2 (0)	2 (1)
			H			H		H			H	H

Total page faults = 7
& no. of hits = 5

Advantages:
→ Improvement over FIFO
→ Allows commonly used pages to stay in queue.

Disadvantage:
→ Suffers from Belady's anomaly.

3) Optimal page replacement algorithm:

In this algorithm pages are replaced which would not be used for the longest duration of time in the future. It has lowest page fault rate of all the algorithm.

Example: Refrence sequence (7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F_1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
F_2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
F_3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
					H		H		H	H		H	H		H	H	H		H	H

no. of refrences = 20
no. of hits = 11
no. of misses = 9
∴ Hit rate = $\frac{11}{20} \times 100\% = 55\%$
& Miss rate = $\frac{9}{20} \times 100\% = 45\%$

Advantages:
→ lowest page fault rate.
→ Never ~~suffrs~~ suffers Belady's anomaly.

Disadvantages:-
→ Not all OS can implement this algorithm.
→ Error detection is harder.

4) LRU Page Replacement Algorithms:-

In this algorithm, the page that has not been used for the longest period of time has to be replaced. This page replacement algorithm looks backward in time, rather than forward.

Example:- Refrence sequence (7,0,1,2,0,3,0,4,2,3,2,1,2,0,1,7) with 3 frames.

Solution:

	7	0	1	2	0	3	0	4	2	3	2	1	2	0	1	7
F_1	7	7	7	2	2	2	2	4	4	4	4	1	1	1	1	1
F_2		0	0	0	0	0	0	0	0	3	3	3	3	0	0	0
F_3			1	1	1	3	3	3	2	2	2	2	2	2	2	7
					H		H				H		H		H	

5) Least frequently Used (LFU) Page Replacement Algorithm:

This algorithm selects a page for replacement if the page has not been used often in the past or replace page that has smallest count. In LFU we check the old page as well as the frequency of that page and if the frequency of the page is larger than the old page we cannot remove it and if all the old pages are having same frequency then take FIFO method for that and remove that page.

Example:- Refrence sequence (7,0,1,2,0,3,0,4,2,3,0,3,2,1,2) with 3 frames.

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
F_1	7	7	7	2	2	2	2	4	4	3	3	3	3	1	1
F_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0
F_3			1	1	1	3	3	3	2	2	2	2	2	2	2
					H		H				H	H	H		H

Frequency table

(Initially frequencies of all are set to 0)

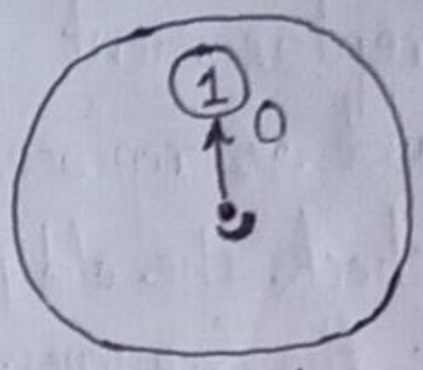
Refrence	frequencies					
7	0	1	0			
0	0	1	2	3	4	
1	0	1	0	1		
2	0	1	0	1	2	3
3	0	1	0	1	2	0
4	0	1	0			

6) Clock Page Replacement Algorithm:-

The clock algorithm keeps a circular list of pages in the memory, with the "hand" pointing to the last examined page frame in the list. When a page fault occurs and no empty frames exist, then the R (refrenced) bit is inspected at the hand's location. If R is 0, the new page is put in place of the page the "hand" points to, and the hand is advanced one position. Otherwise, the R bit is cleared, then the clock hand is incremented and the process is repeated until a page is replaced.

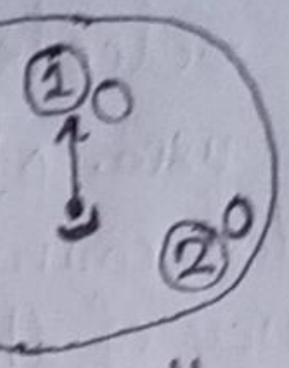
Example:- Input Refrences: {1,2,3,4,3,1,2,1,5,4}
Size of memory: 3

Upon accessing 1



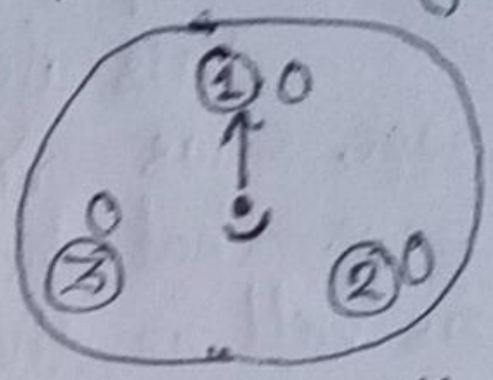
Page fault

Upon accessing 2



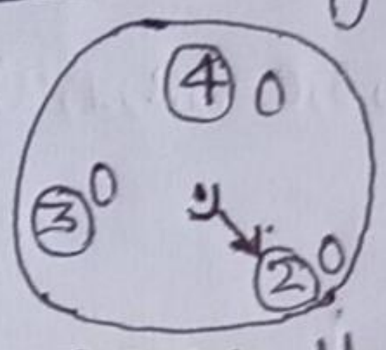
Page fault.

Upon accessing 3



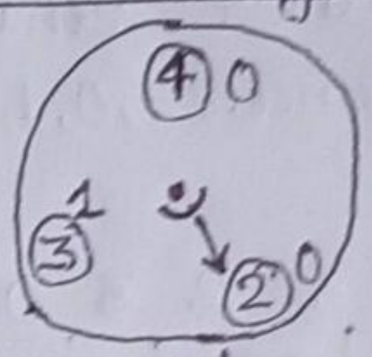
Page fault.

Upon accessing 4



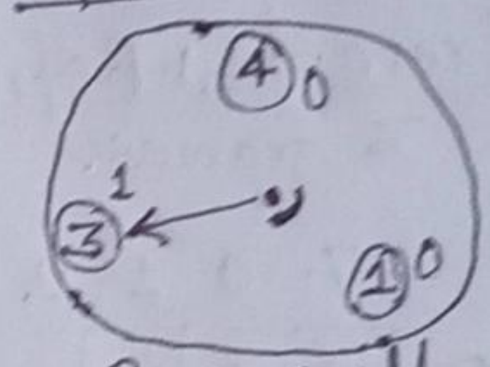
Page fault

Upon accessing 3



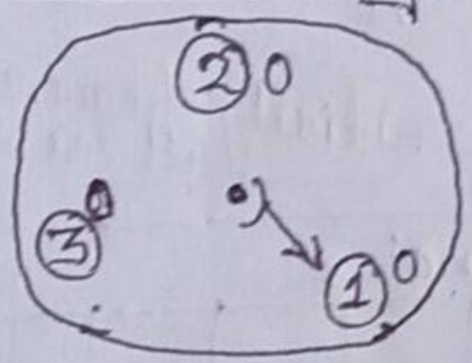
Hit

Upon accessing 1



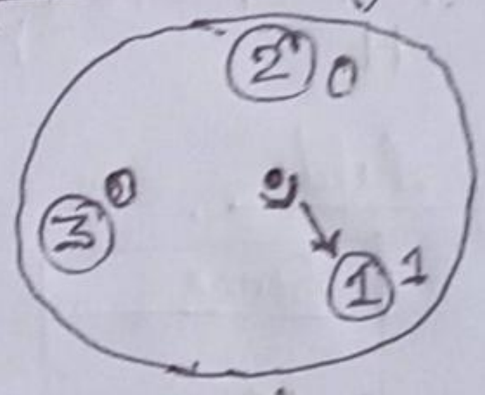
Page fault.

Upon accessing 2



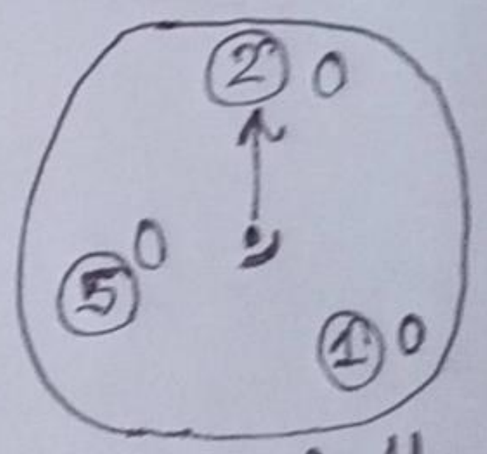
Page fault

Upon accessing 1



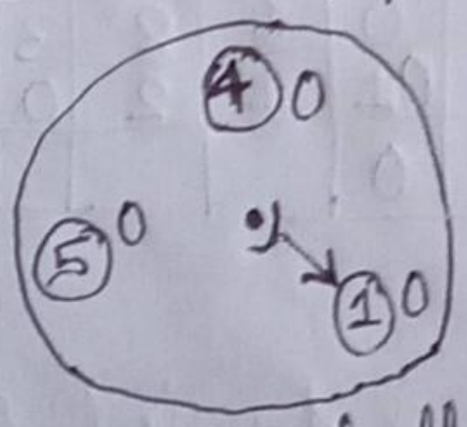
Hit.

Upon accessing 5



Page fault.

Upon accessing 4



Page fault.

Hence

total no. of page faults = 8

& total no. of page hits = 2.

7) WS-Clock Page Replacement Algorithm:

It is an improved form of clock page replacement algorithm. Lesser important in comparision to other. We can study this from youtube if we want.

[I have escaped this].

⊛. Locality of refrence:-

Locality of refrence is the tendency of a processor to access the same set of memory locations repetitively over a short period of time. Subroutines tend to localize the refrences to memory for fetching instructions, this is the reason for this property. There are two basic types of locality of refrence: temporal and spatial. Locality of refrence makes cache memory to work.

i) Temporal locality → The information which is used currently is likely to be in use in near future. For e.g, Reuse of information in loops.

ii) Spatial locality → If a word is accessed, adjacent (near) words are likely to be accessed soon. For e.g, Related data items (arrays) are usually stored together, Instructions are executed sequentially.

Segmentation

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 also stored in one (or many) of the segments. Segment table contains mainly two information about segments:

i) Base → It is the base address of the segment.

ii) Limit → It is the length of the segment.

⊛. Why segmentation is required ?

⇒ Paging is more close to operating system rather than the user. 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.

Advantages of segmentation:

→ No internal fragmentation
→ Less overhead.
→ It is eaiser to reallocate segments than entire address space.
→ The segment table is of lesser size as compared to the page table in paging.

Disadvantages of segmentation:

→ It can have external fragmentation.
→ It is difficult to allocate contigious memory to variable sized partition.
→ Costly memory management algorithms.

Paging vs Segmentation

Paging	Segmentation.
i) Page is always of fixed block size.	i) Segment is of variable size.
ii) Paging may lead to internal fragmentation as page is of fixed size block.	ii) Segmentation may lead to external fragmentation as the memory is filled with the variable sized blocks.
iii) In paging the user only provides a single integer as the address which is divided by hardware ~~number~~ into a page number & offset.	iii) In segmentation the user specifies the address into two quantities i.e, segment number and offset.
iv) The size of the page is decided or specified by the hardware.	iv) The size of the segment is specified by the user.

⊛. Segmentation with Paging (MULTICS):

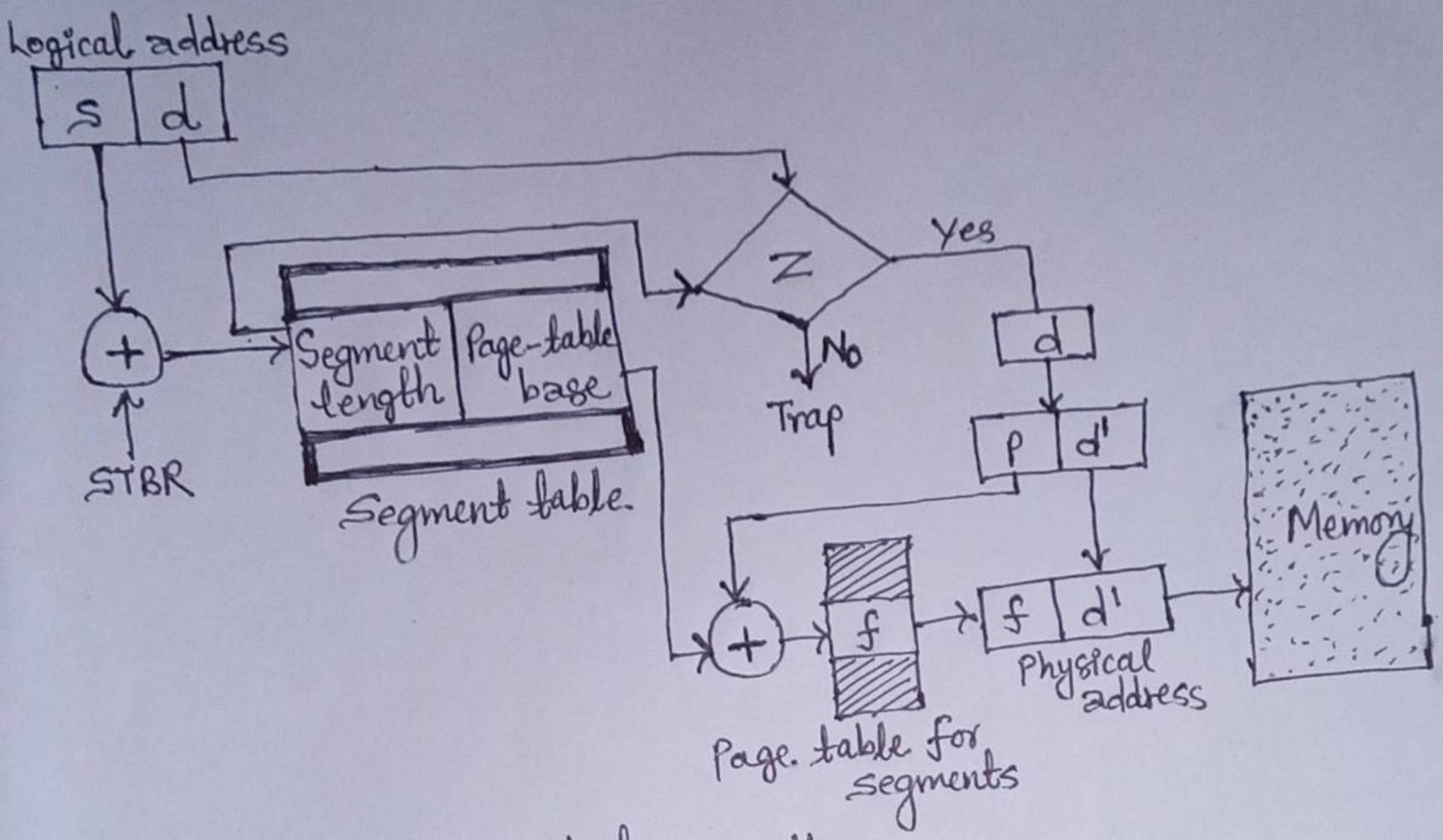


Fig: Segmentation with paging

→ The segment number is used to find segment descriptor.

→ A check is made if the segment's page table is in memory. If it is, it is located. If not, a segment fault occured.

→ The page table entry for the requested virtual page is examined. If the page itself is not in memory, a page fault was triggered. If it is in memory, the main-memory address of the start of the page is extracted from the page table entry.

→ The offset is added to the page origin to give the main memory address where the word is located.

→ The read or write finally takes place.