Bare Machine and Resident Monitor

* These two are important Part of Computer system.
* the Bare Machine and Resident Monitor are not directly related to O.S but while we study about memory management these components are really important to study.

Bare Machine → Basically Bare Machine is logical hardware which is used to execute the Program in the Processor without using O.S.

— As of now, we have studied that we can't execute any Process without the O.S. But yet with the help of the Bare machine we can do that.

— Initially, when the O.S are not developed, the execution of an instruction is done by directly on hardware without using any interfacing hardware, at that time the only drawback was that the Bare Machines accepting the instruction is only machine language, due to this those Persons who has sufficient

knowledge about computer field are able to operate a computer.
So, after the development of the O.S Base machine is referred to as inefficient.

<u>Resident Monitor</u> — The Resident Monitor is a code that runs on Base Machine.

The resident monitor works like an O.S that controls the instructions and perform all necessary functions. It also works like job sequencer because it also sequences the job and sends them to the Processor.

— After scheduling the job Resident monitors loads the programs one by one into the main memory according to their sequences. There is no gap between the program execution and processing is going to be faster.

The resident monitor is divided into 4 parts as,

1 → Control Language Interpreter
2 → Loader
3 → Device Driver
4 → Interrupt Processing.

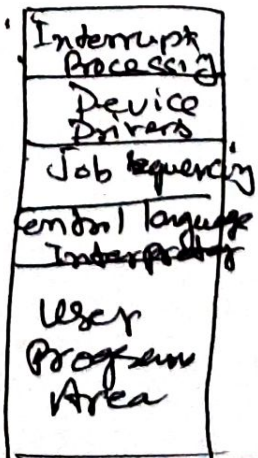


fig: Memory

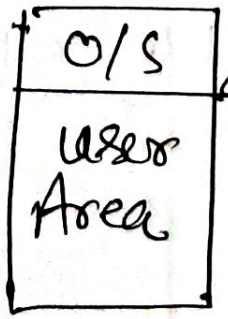
(1) Control or language Interpreter :- The first Part of the resident monitor is control language interpreter which is used to read and carry out instructions from one level to the next level.

(2) Loader :- It loads all necessary System and application Programs into the main memory.

(3) Device Driver :- It is used to manage the connecting input-output devices to the system. So, basically it is a the interface between the user and the System

(4) Interrupt Processing :- It Processes all occurred interrupt to the system

Main Memory



(* one process at one time)

(1) Bare Machine

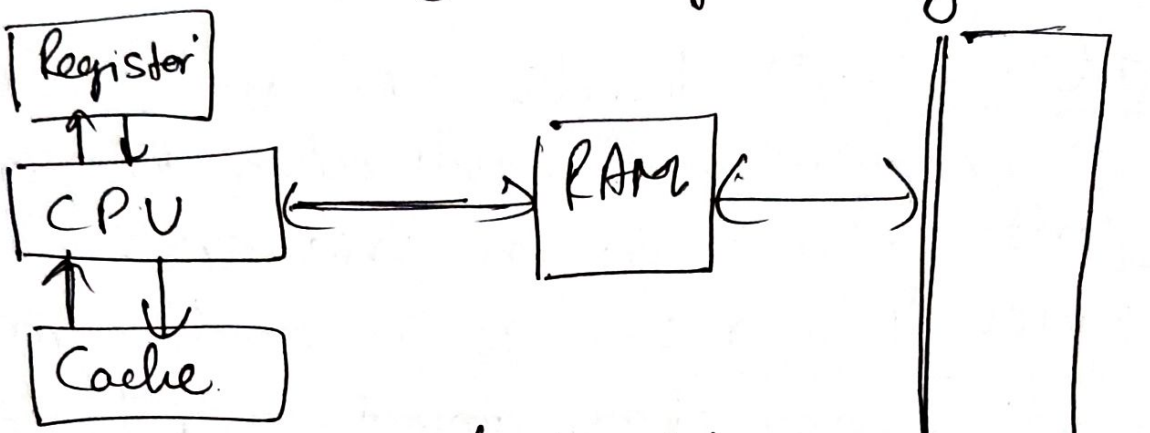
* no Reloaded OS
* Application Specific
* Sequential instruction Execution in Machine language.
* Task Dedicated
* Task Handled by designer.

(2) Resident Monitor

* A Part of OS resides in MM
* Fence Address is used to Provide Security to OS from user Programs
* Useful to Single user System

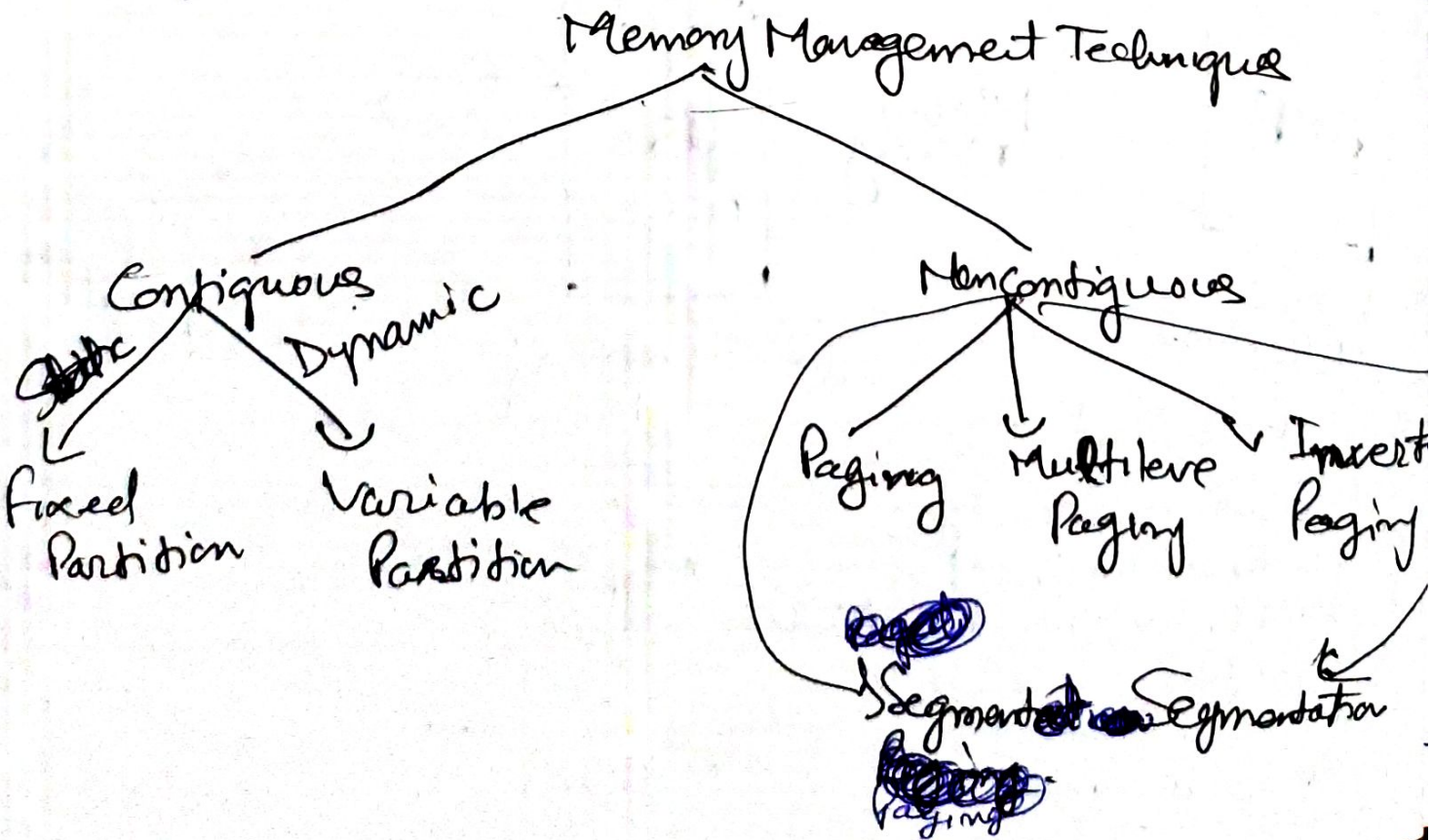
Memory Management

Is the method of managing Main memory (Primary Memory)
ie Efficient utilization of Memory

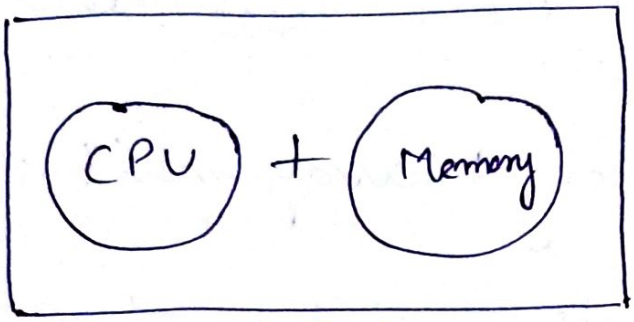


(* Say 'A Memory Management is functionality of OS which manages Main Memory and keeps tracks of processes moving from secondary memory to Main memory. *)

So, Memory Manage includes methods of managing Degree of Multiprogramming.



Introduction to Memory Management →



Computer

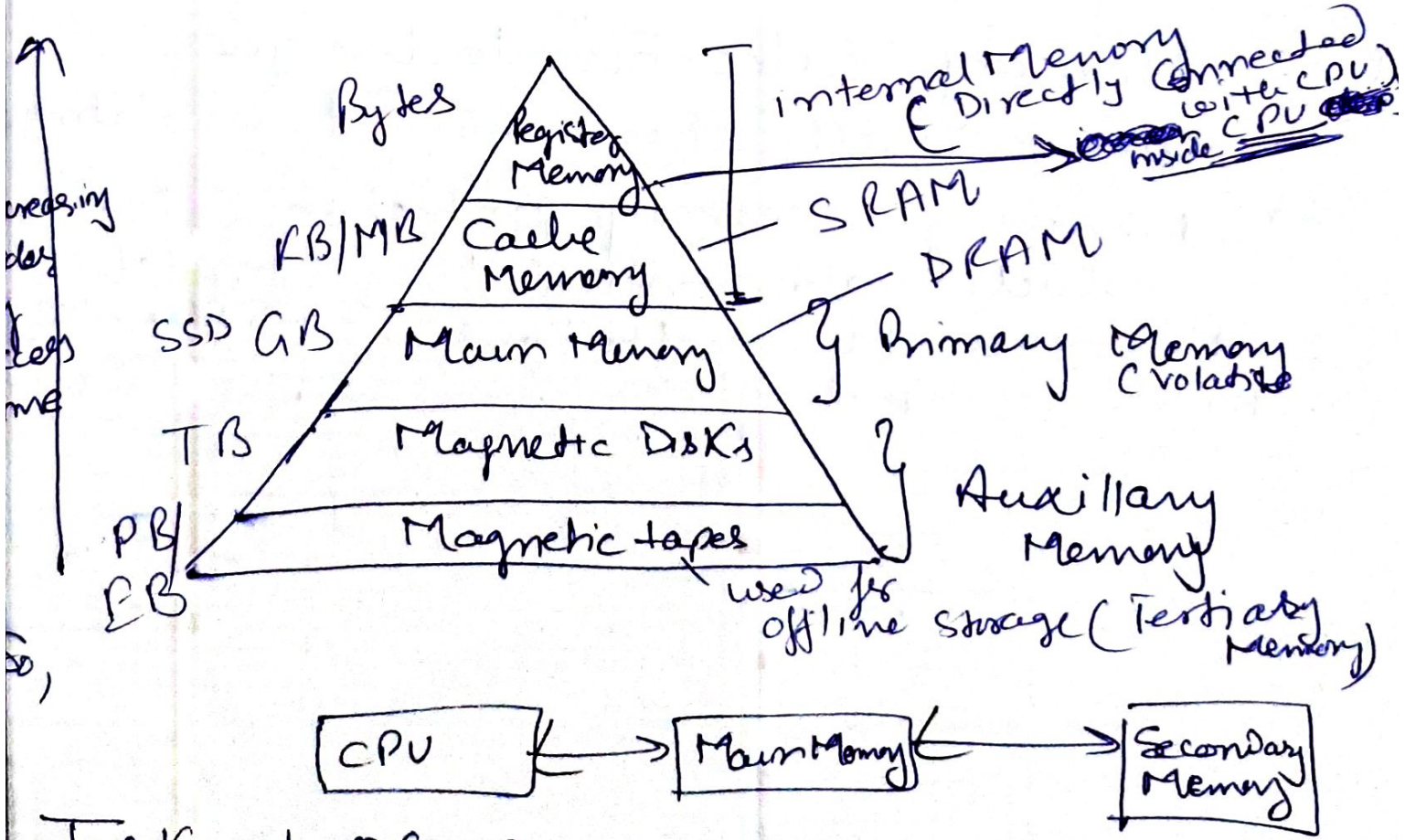
Memory

Size: Large Memory Size

Access Time: As less as Possible

Per Unit Cost: As less as Possible

Background of Memory Management →



Task of OS →

1 - which data from the secondary memory should come to main memory and at which

location.

2- CPU generates address for data at secondary memory, but have access to main memory, thus O.S performs the address translation.

Functions/Goals of Memory Management :

(1) Keep track of the status of each memory location where the memory is allocated or free.

(2) Determining allocation Policies for the memory

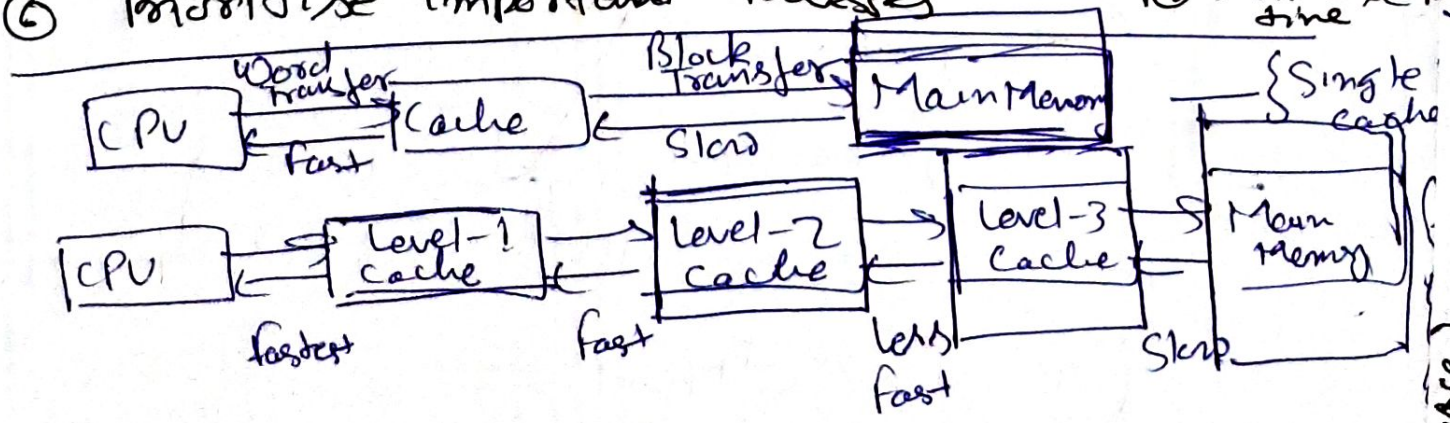
(3) Memory allocation techniques, memory allocation information updating

(4) Providing memory protection schemes which are critical to accomplish

(5) Maximise Memory Utilization.

(6) Prioritise important Processes

(1) Maximise CPU Utili
(8) Minimise R time



Cache organization

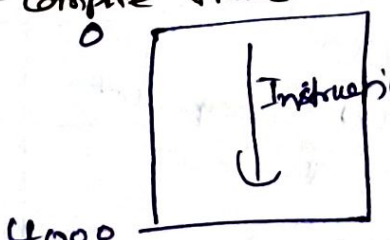
Address Binding of instructions and data to Memory →

Compile time : absolute address (Confirm train ticket)

advantage :– Minimum setup time.

disadvantage : – if the memory space is preoccupied by a program then there is a chance of collision.

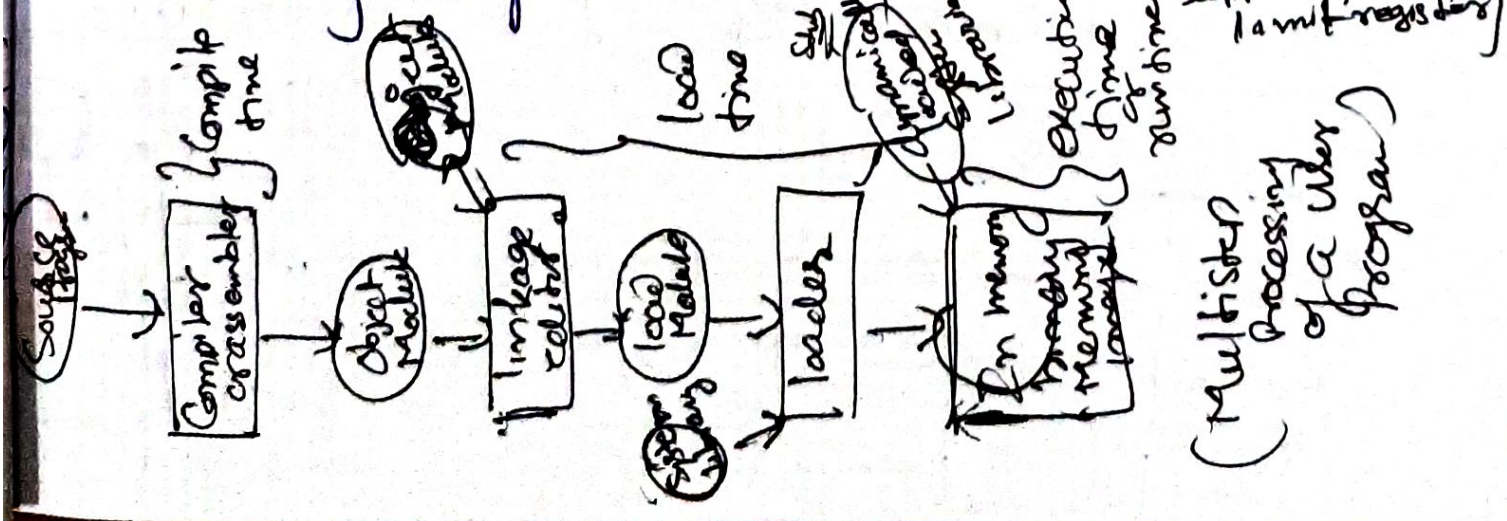
Load time : relocatable code (Confirm flight ticket)
if memory location is not known at compile time.

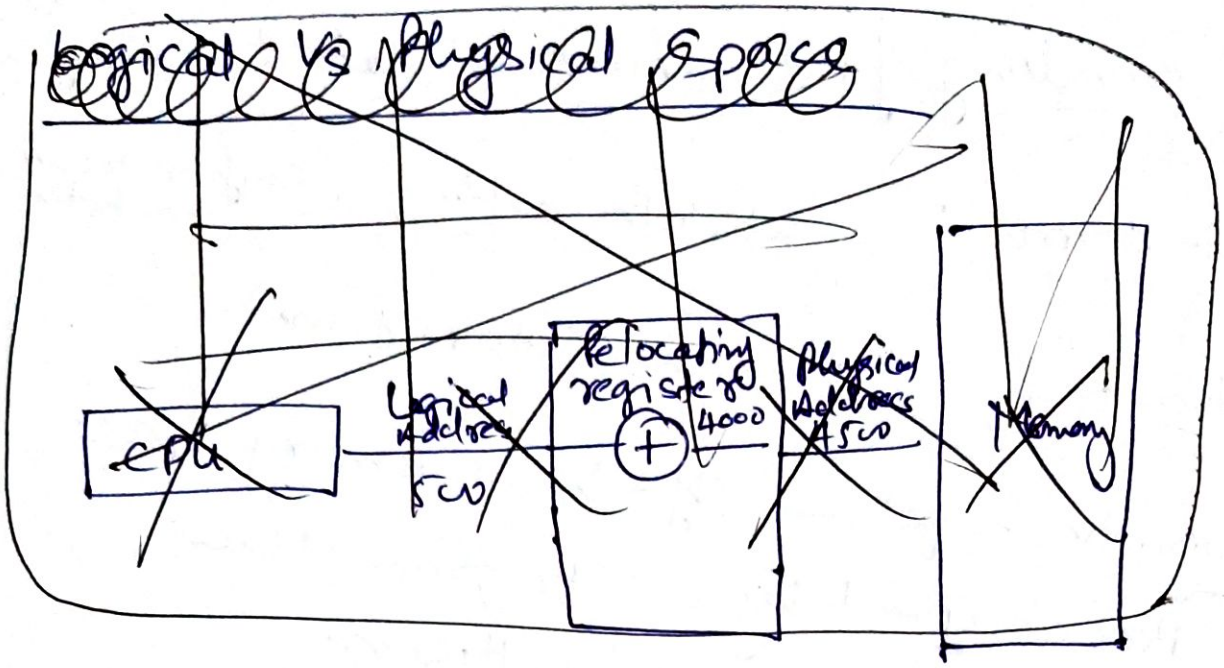


then converted to 0 + 2000
1 + 2000

at load time

Execution time → Process can move during its execution from one memory segment to another (Needs hardware support, base/limit registers)





Swapping →

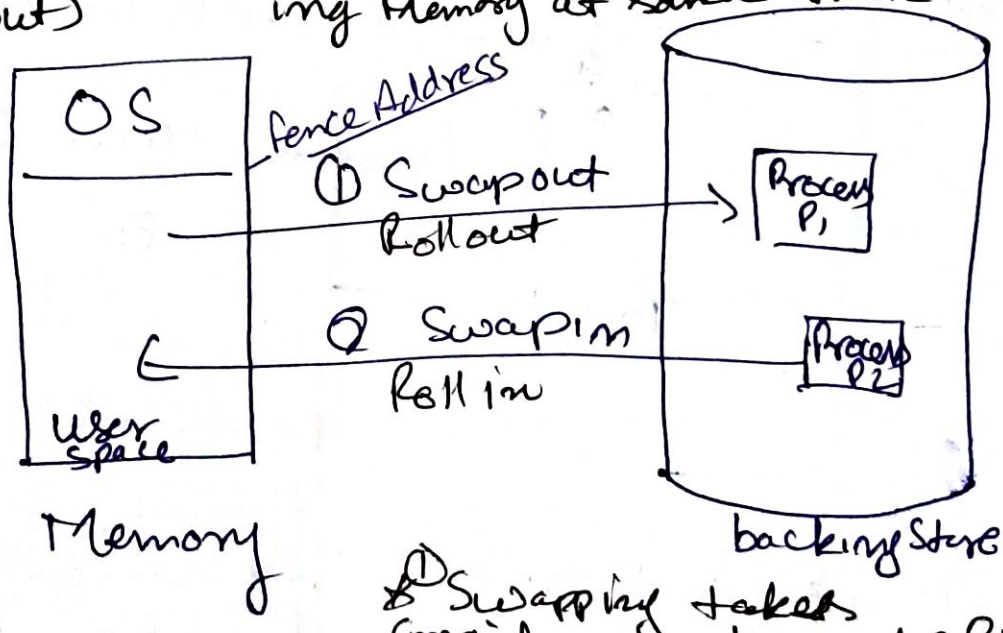
Problem with Swapping → If all of them are acc ing Memory at same time.

Reasons: → (Swapout)

- Completed Task
- Abnormal termination i. ETRAP
- For I/O.

Swapout could be —

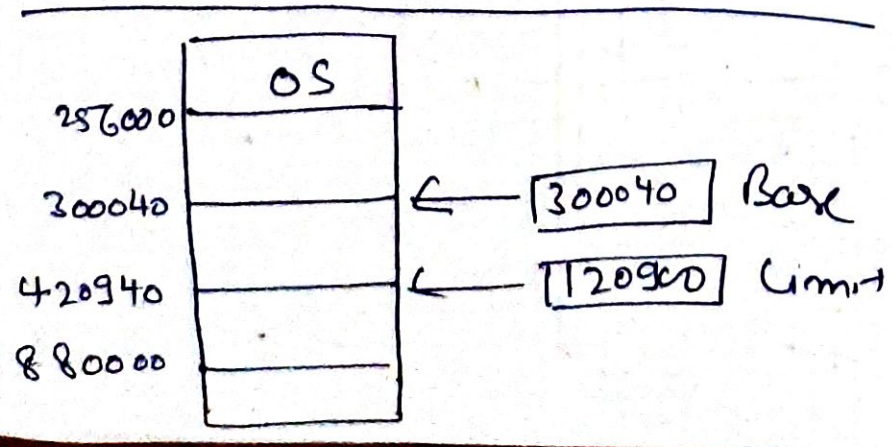
1. write to disk
2. or do nothing in case of read only process

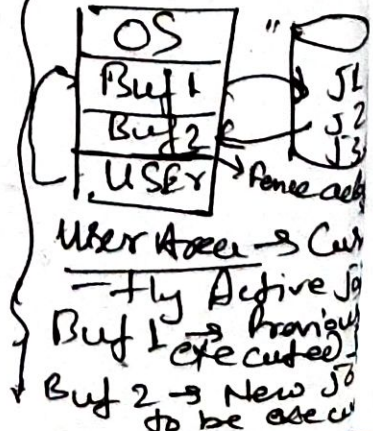


Swapping takes considerable time of CPU for new process become to CPU

Solution to this

Base and Limit Register →





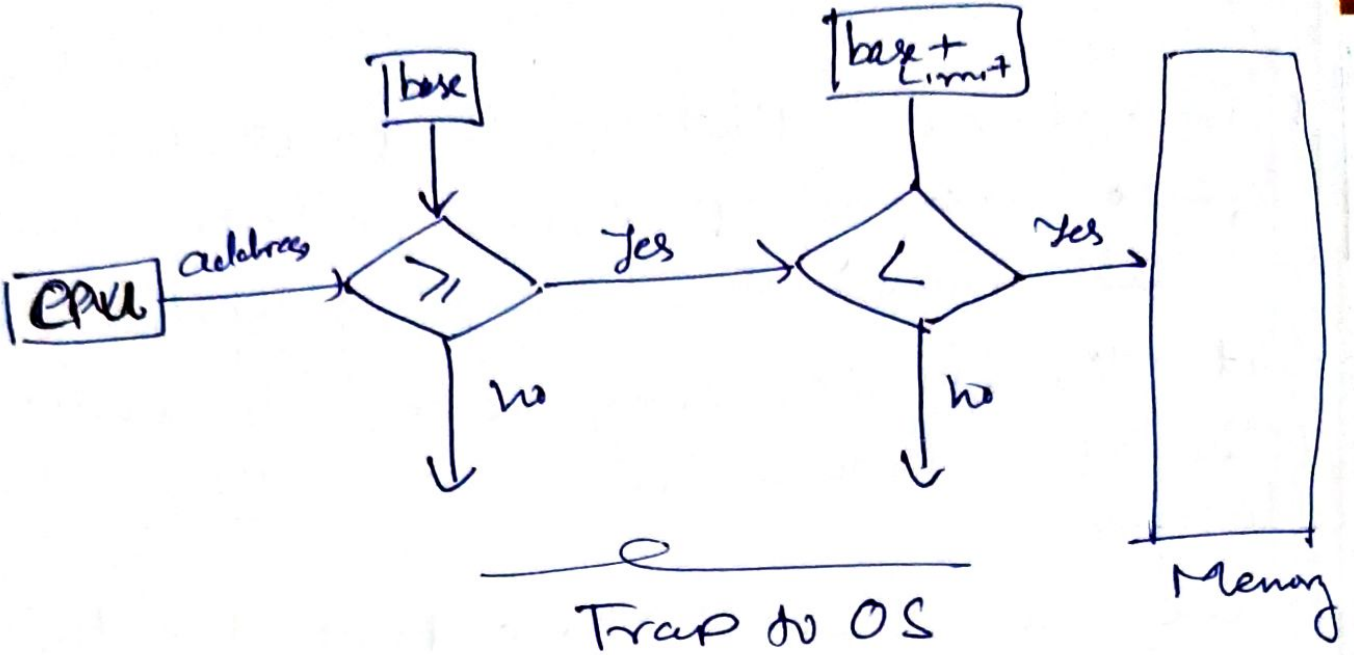
* To make sure that each process has a separate memory space, we need the ability to determine the range of legal addresses that the process may access to ensure that the process can access these legal addresses.

We can provide this protection by using two registers, usually a base and a limit.

So, Here in fig. the program can legally access all addresses from 300040 through 420939 (inclusive)

Hardware address protection with base and limit registers

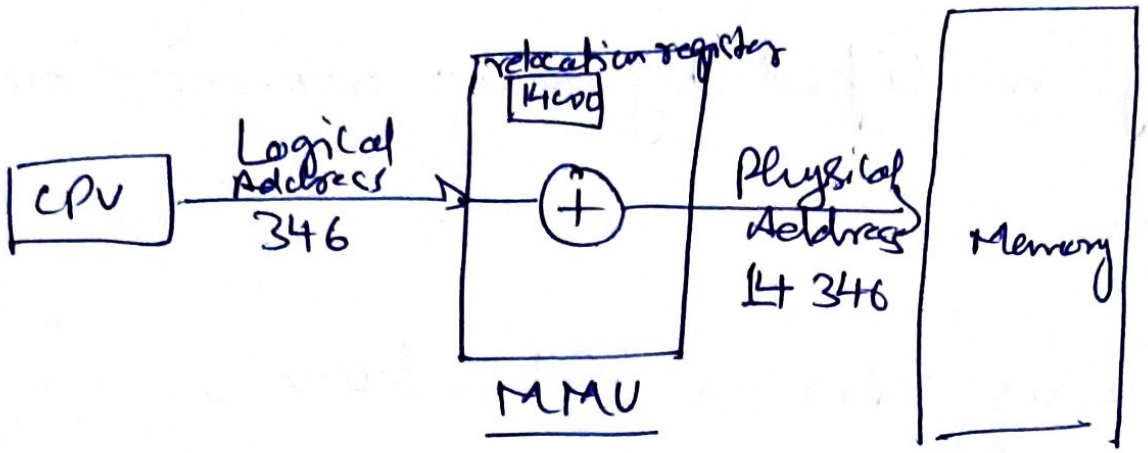
* Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with registers.
* Any attempt by a program executing in user mode to access O.S memory or other user's memory results in a trap to O.S, which treats it as a fatal error.



Logical Vs Physical Address Space

* The set of all logical addresses generated by a program is a logical address space.
* The set of all physical address corresponding to these logical addresses is a physical address space.

Note:- The run-time mapping from virtual logical Address space to physical address space is done by a hardware device called memory-management unit (MMU).



Memory Mapping and Protection :-

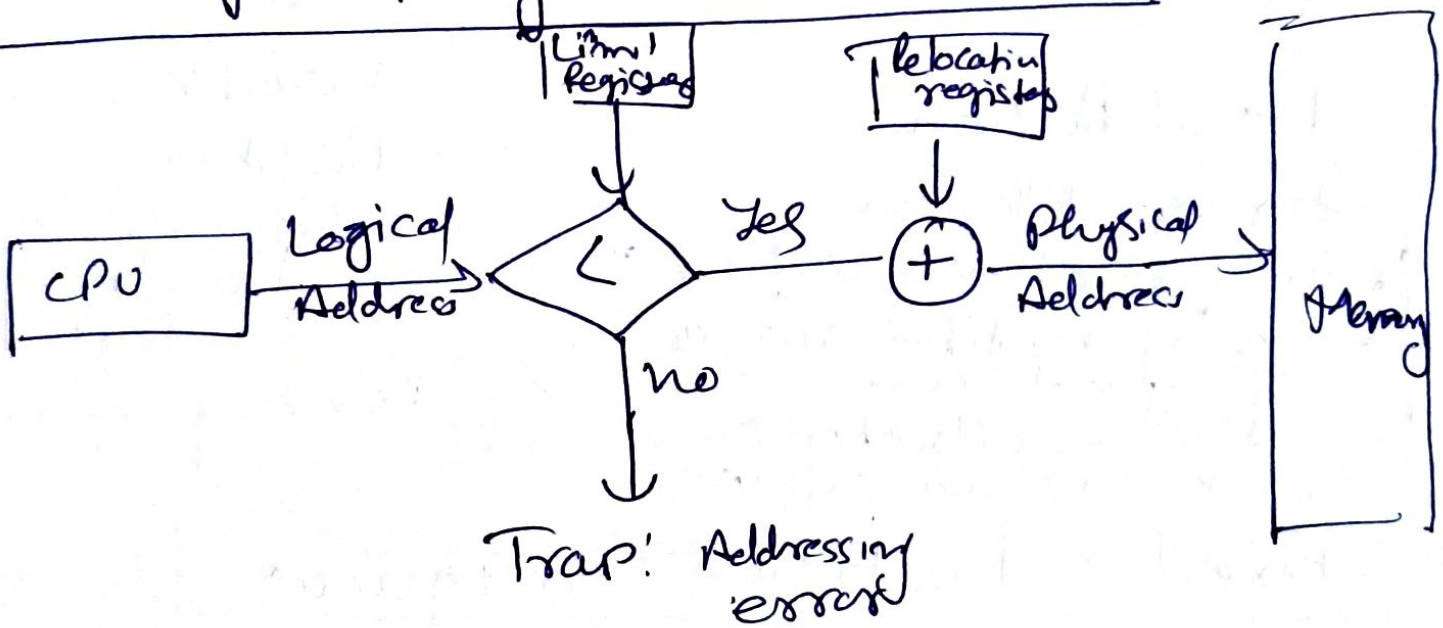


fig. — Hardware support for relocation and limit registers

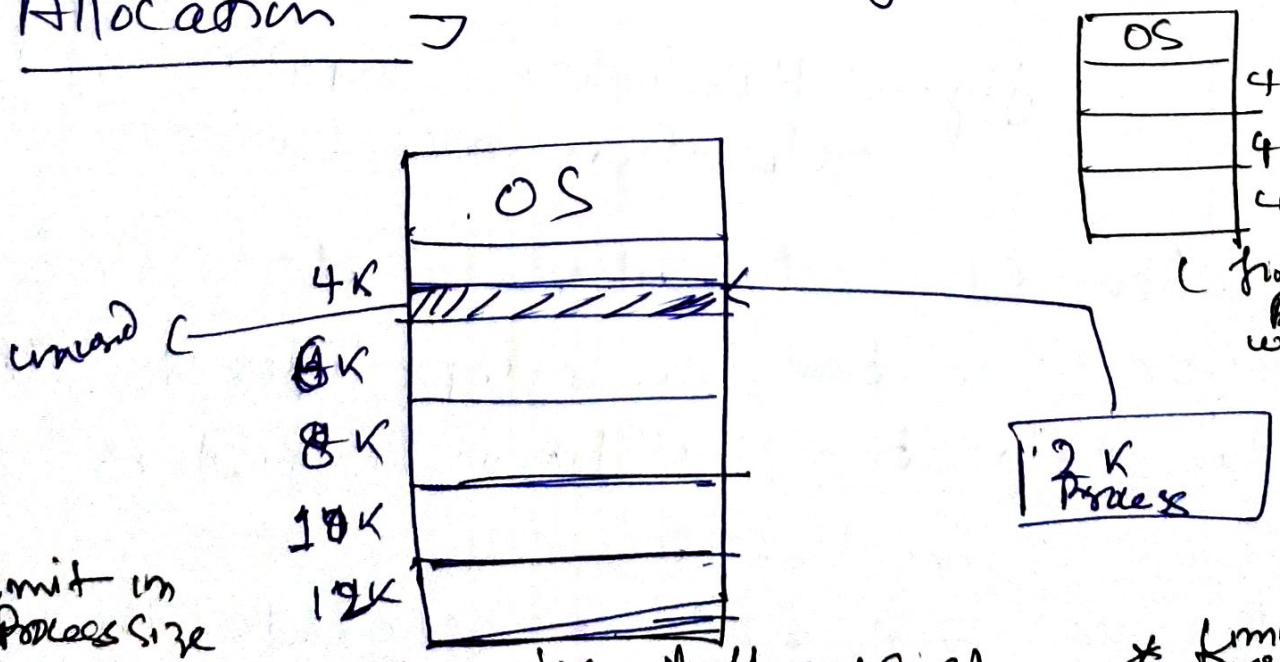
* When CPU scheduler selects a Process for execution, the dispatcher loads the relocation and limit registers with correct values.
* Because every address generated by CPU is checked against these registers, we can Protect both the OS and the other user's Programs and data from

being modified by this running process

Contiguous Storage Allocation :

- Static
  - Fixed Partition Storage Allocation
    - * no of Partitions are fixed
    - * Size of Partition may or may not same
    - * Contiguous allocation so spanning is not allowed
  - Variable - Partition Storage Allocation Strategy
    - Partition set runt

fixed - Partition Contiguous Storage Allocation :

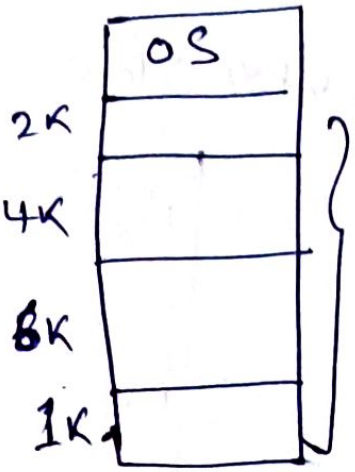


(with different sizes)

* Suffer to internal fragmentation.
* This Schema was used for early batch systems because space requirement of a process is kno

Variable – Partition Contiguous Storage allocation :-

* No Limitation on Process Size



Variable Partitions at runtime

* There is no internal fragmentation
* No limitation on no of processes or degree of multiprogramming

* Suffer to External fragmentation (hole)

Three Strategies ——

(i) Best – fit (ii) Worst - Fit

(iii) First – fit

Solution to External fragmentation :- Compaction, Coalescing → No data movement

both will be merged to get a Contiguous space

First Fit :- Allocate the first hole that is big enough. // Simple & fast

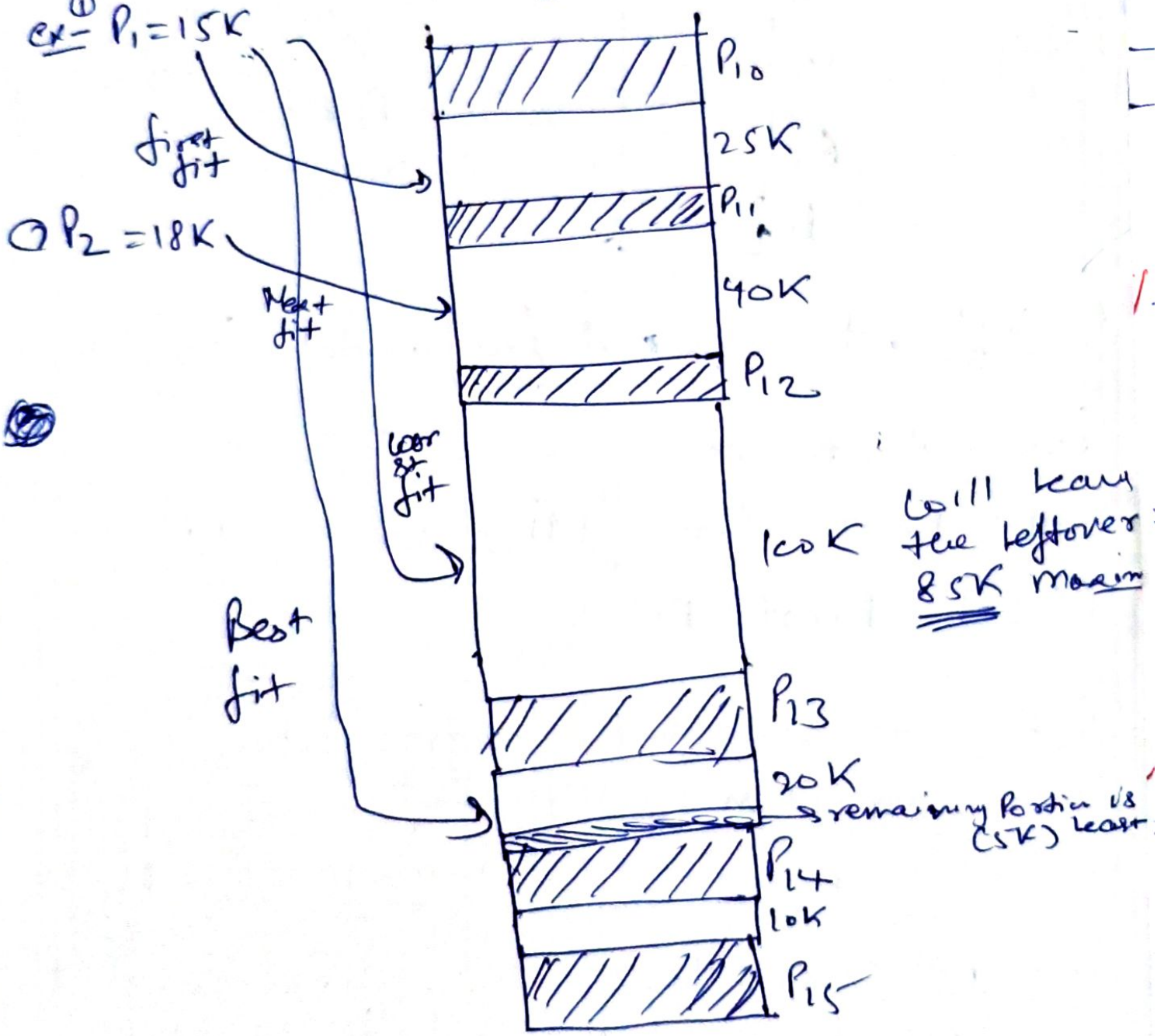
Next - fit :- Same as first fit but start search always from last allocated hole. // advantage is same as first fit with pointer pointing next

Disadvantage // It will search the entire List it is slow
advantage // Internal fragmentation is less to accom a new ho

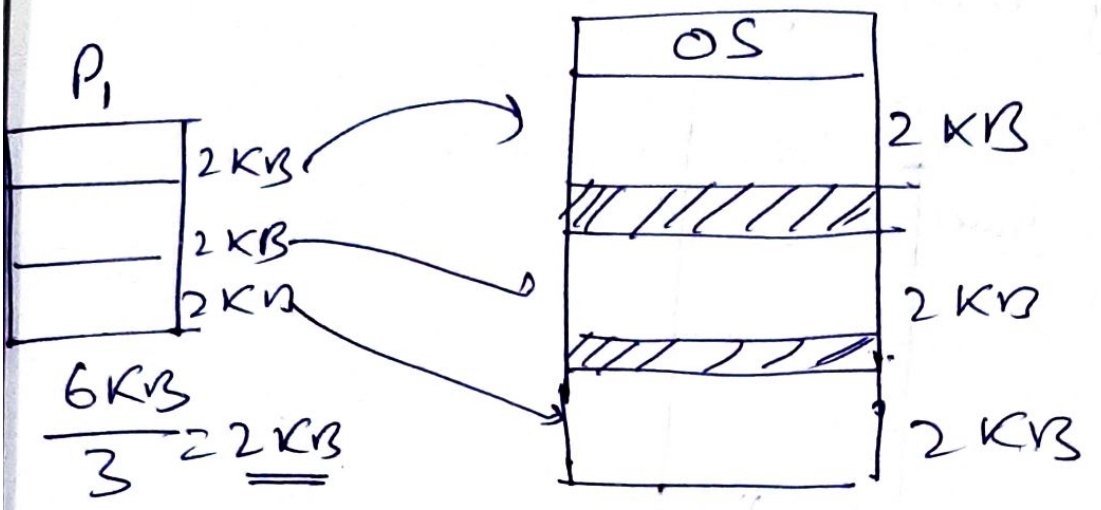
Best fit :- Allocate the smallest hole that is big enough

Worst fit :- Allocate the largest hole.

Disadvantage // slow



Non Contiguous Memory Allocation → (Need of Paging)

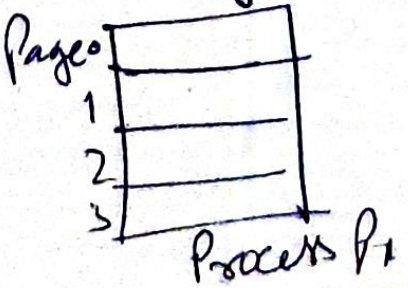


* To Remove external fragmentation we use non Contiguous memory allocation

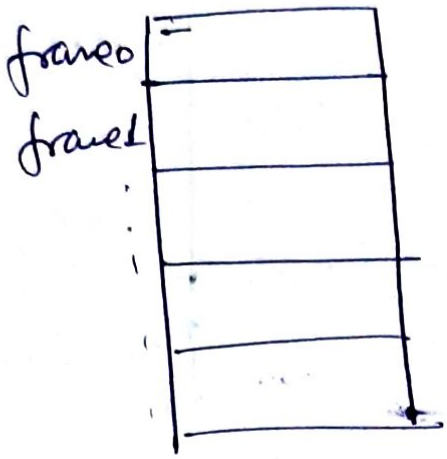
* So, to allocate the memory dynamically at runtime is costly because we need to check the availability of holes and there size and accordingly we can divide the size of Process.

* The hole size will continuously change.

So, what we can do is we can create a Pages of a Process in Secondary Memory



* Like we will divide the Physical memory into frames.



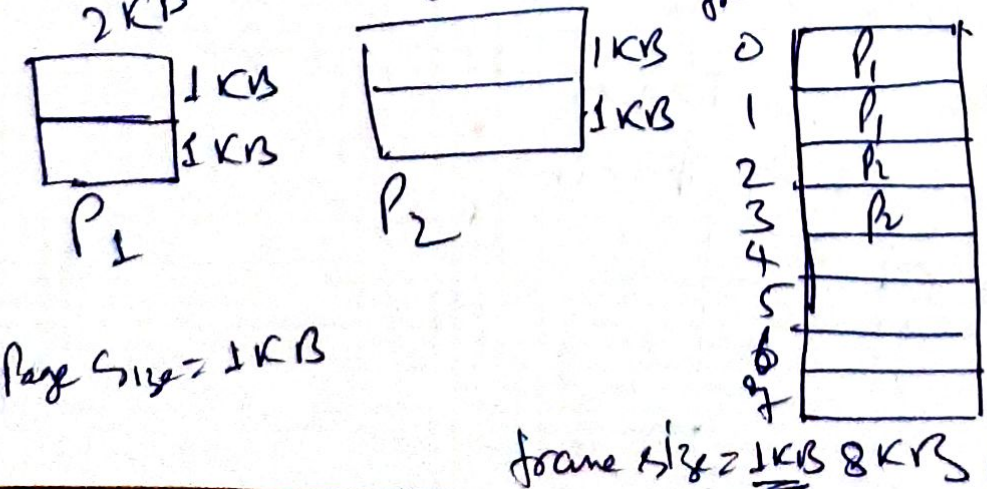
Main Memory

* Imp — Page Size = frame Size

* we can allocate those of Pages to different frames (non contiguously)

* Advantage is that External fragmentation is completely removed.

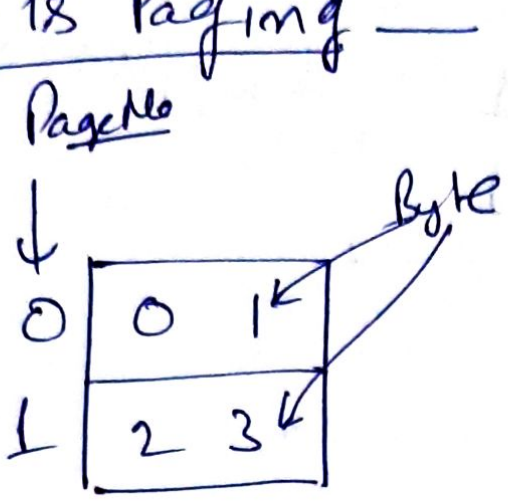
Ex



Page Size = 1KB

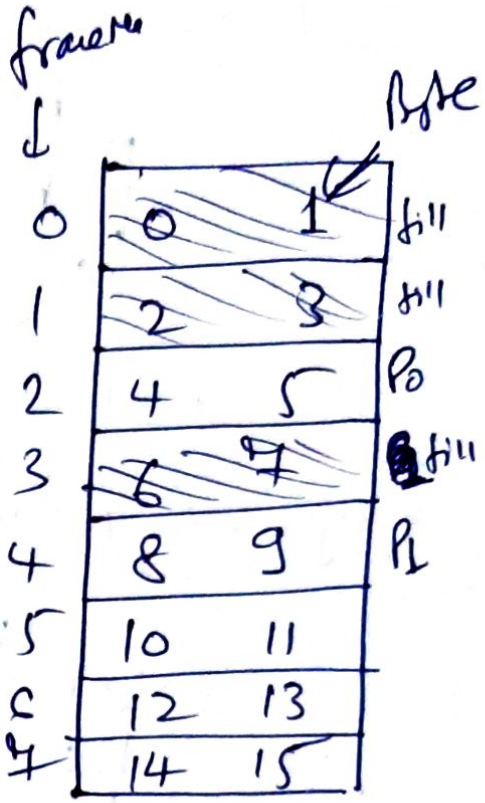
frame size = 1KB & 8KB (RAM)

What is Paging —



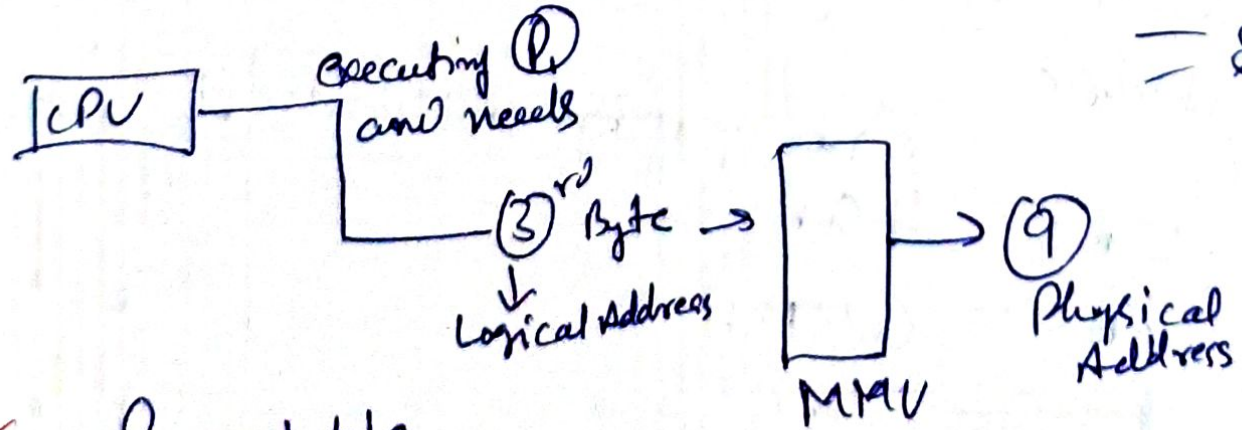
- Process Size = 4B
- Page Size = 2B

formula: No. of Pages = $\frac{4B}{2B}$ = 2

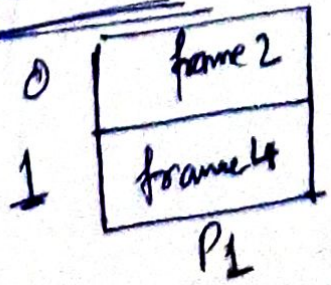


- M/M Size = 16B
- frame Size = 2B

formula: No of frames = $\frac{16B}{2B}$ = 8 frames

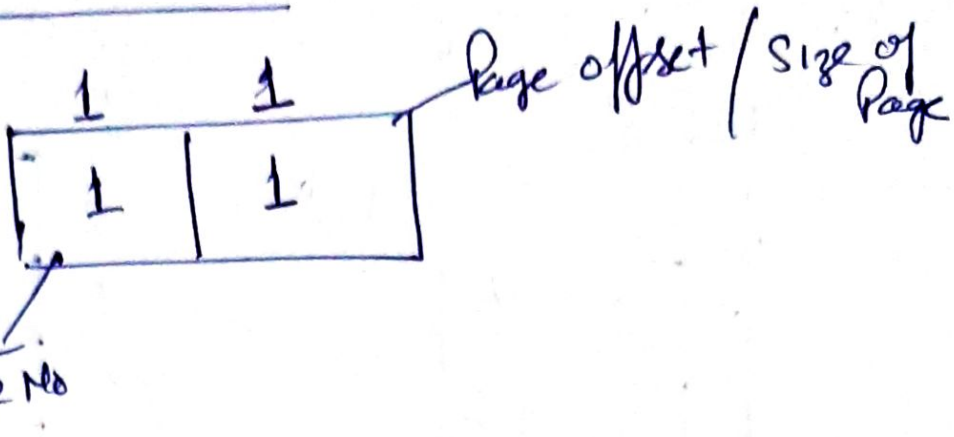


Page table

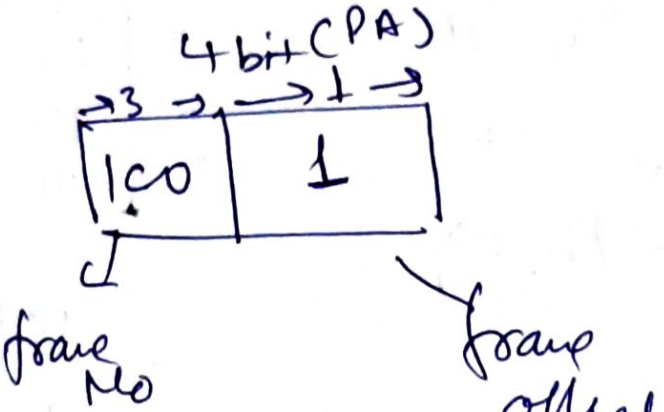


// MMU converts Logical address to Physical address by using Page table.

Logical Address



Physical Address →



// because Memory Size = 16 B
$2^4 = 16$

Question on Logical Address and Physical Address →

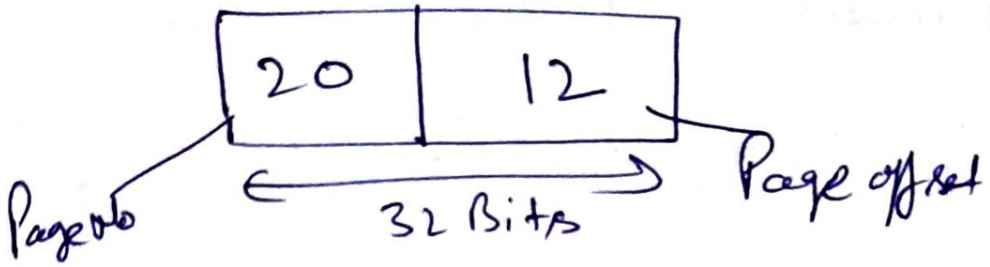
* Memory is Byte Addressable

Given
LAS = 4GB
PAS = 64MB
Page Size = 4KB
No. of Pages = ?
No. of frames = ?
No of entries in Page table = ?
Size of Page table = ?

$2^1 = 2$	2^{20} =
$2^2 = 4$	2^{30} =
$2^3 = 8$	2^{40} =
$2^4 = 16$	
$2^5 = 32$	
$2^6 = 64$	
$2^7 = 128$	
$2^8 = 256$	
$2^9 = 512$	
$2^{10} = 1K$	

Solution:-

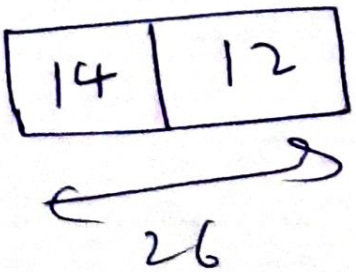
$$LA = 4GB$$
$$= 2^2 \times 2^{30} B$$
$$= 2^{32} B$$



$$\text{Page Size} = 4\,KB = 2^2 \times 2^{10}$$
$$= 2^{12} B$$

$$\text{So, No of Pages} = 2^{20}$$

$$PA = 64\,MB = 2^6 \times 2^{20}$$
$$= 2^{26} B$$



$$\text{So, No of frames} = 2^{14}$$

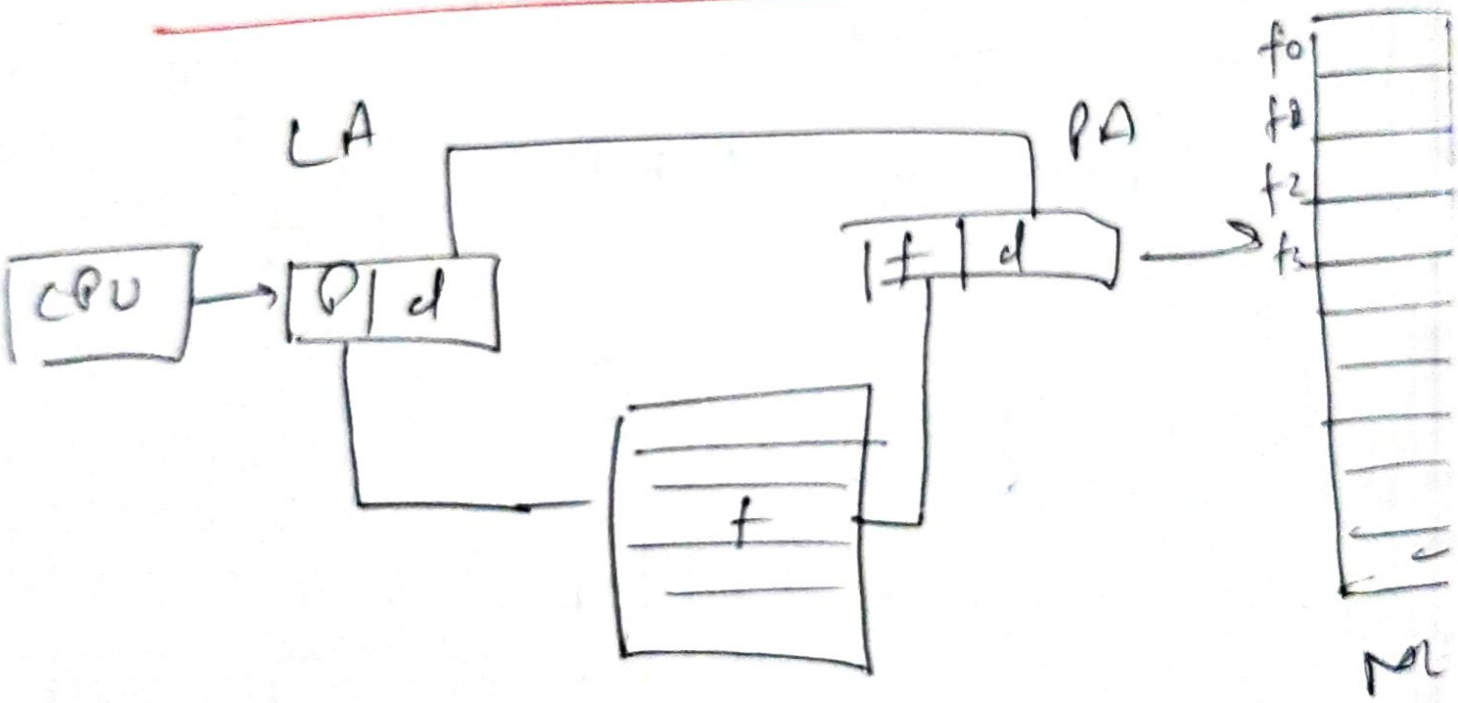
No of entries in Page table = No of Pages in a process

So, No of entries in PT $= 2^{20}$

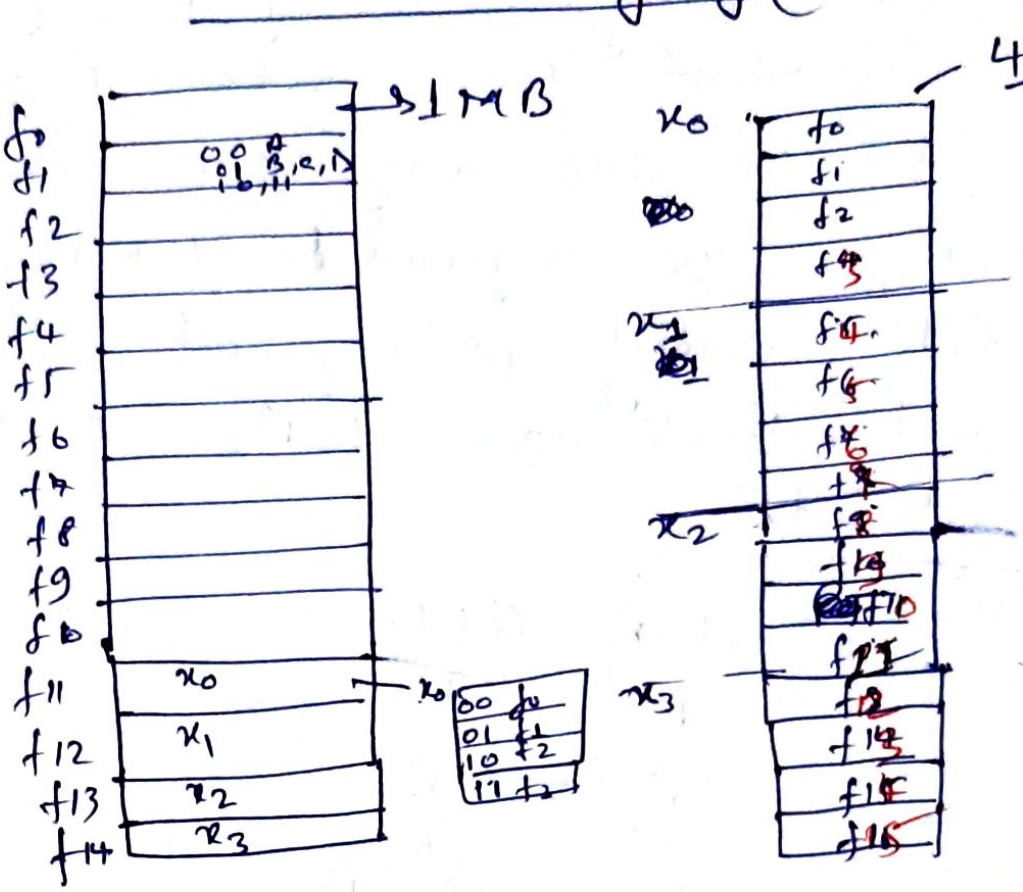
Size of P.T $\equiv$ No. of entries in PT $\times$ Size of frame
$2^{20} \times 14$ bits

* Page-table base register (PTBR)
  Points to Page table
* Page-Table length Register (PTLR)
  indicates Size of Page table.

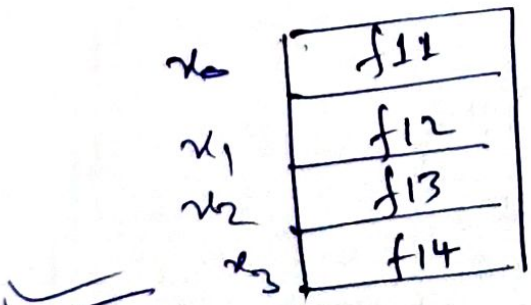
Address Translation Architecture



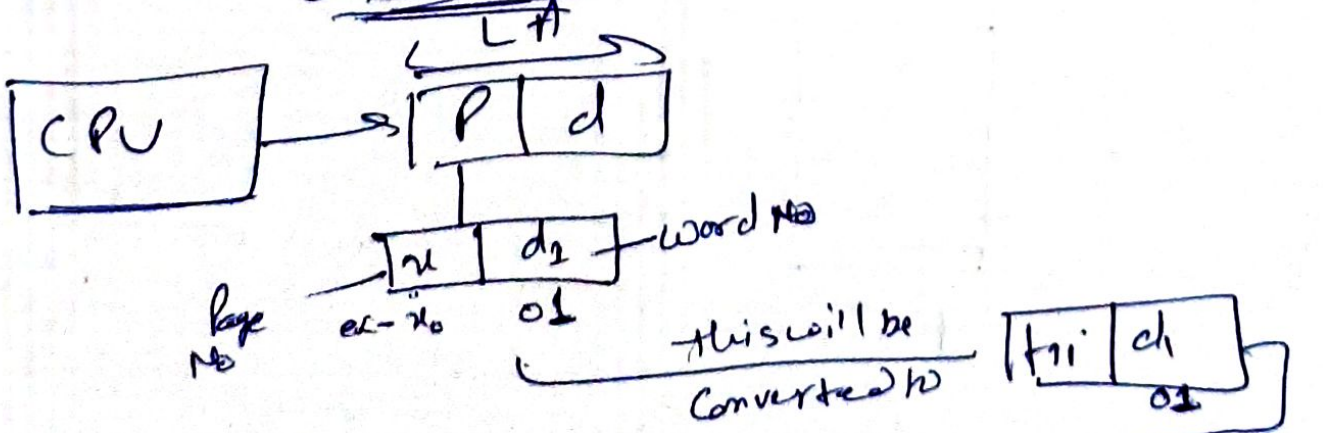
Multilevel Paging (2 level)



MM



Outer P.T



Now which data ? ie  ie 10 — C Ans.

Q- How many levels of Paging is required such that the outer Page table fit in one Page size assuming Page size equal to 4KB and Page table entry is equal to 4B at all levels and LA is equal to 64 bits

Soluⁿ

LA = 64 bits

P.S = 4KB

PTE = 4B

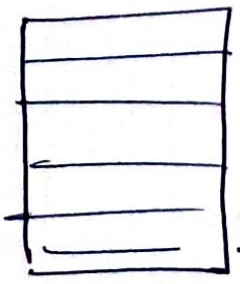
LAS = 2^{64} B

PS = $2^2 \times 2^{10}$ B

= 2^{12} B

No of Pages = $\frac{2^{64}\text{ B}}{2^{12}\text{ B}}$

= 2^{52} Pages



PT — No of entries in P.T = 2^{52}

Size = $2^{52} \times 4$B

= 2^{54} B

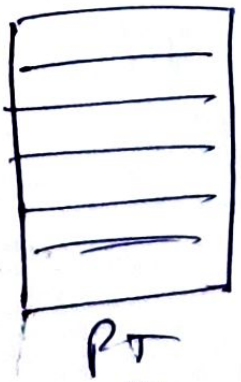
So, we can't put 2^{64}B in 2^{12}B P.T Size

So, Let divide the P.T

$$\text{Pages} = \frac{2^{54}}{2^{12}} = 2^{42} \text{ Pages (No of entries)}$$

$$\text{Size} = 2^{42} \times 4B$$
$$= 2^{44} B$$

again this 2^{44} is much bigger than 2^{12}B P.T Size, so another division is required



$$\frac{2^{44}}{2^{12}} = 2^{32} \text{ Pages (no of entries)}$$

$$\text{Size} = 2^{32} \times 4B = 2^{34} B \text{ Size}$$

again its much bigger, So, we need to divide it again

$$2^{54} \longrightarrow 2^{44} \longrightarrow 2^{34} \longrightarrow 2^{24} \longrightarrow 2^{14} \longrightarrow 2^{4}$$

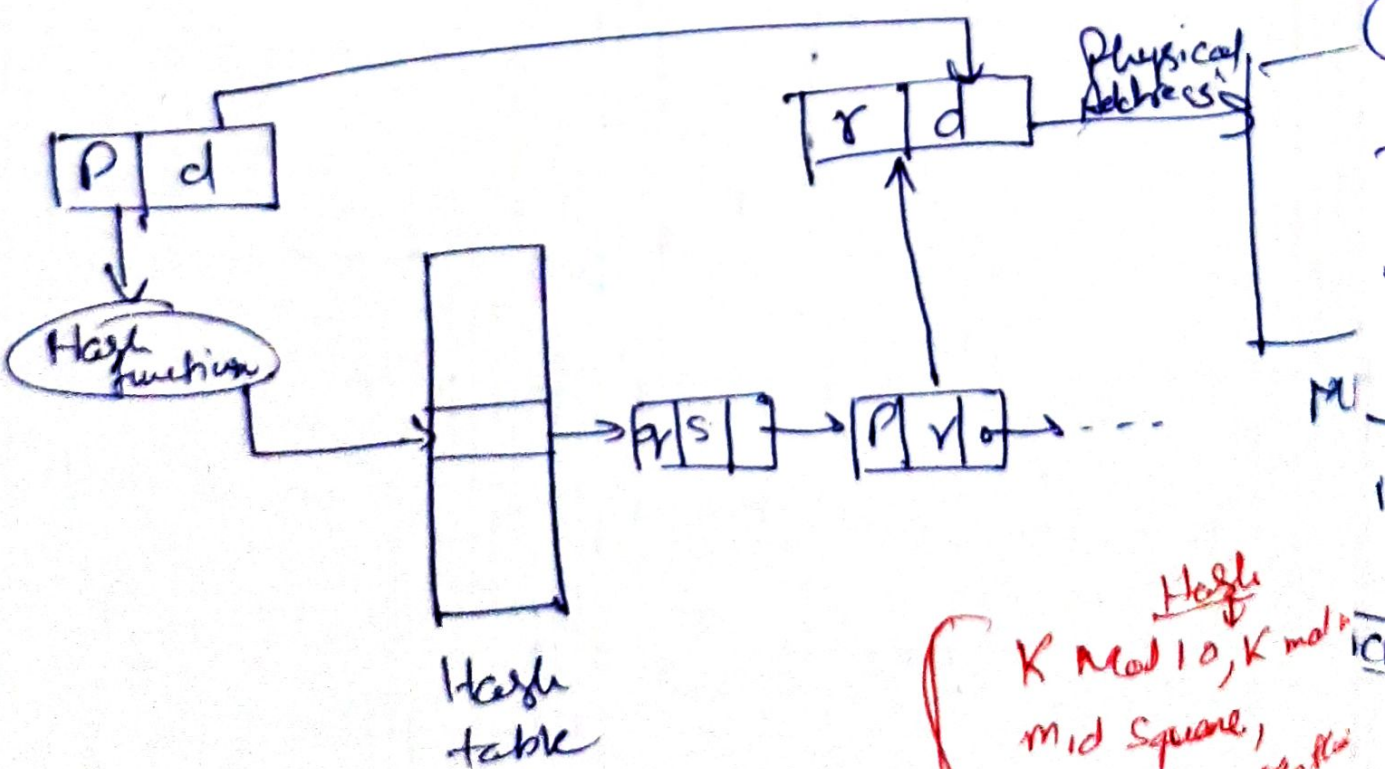
① ② ③ ④ ⑤ ⑥

ie 6 level of Paging.

Hashed P.T → A common approach

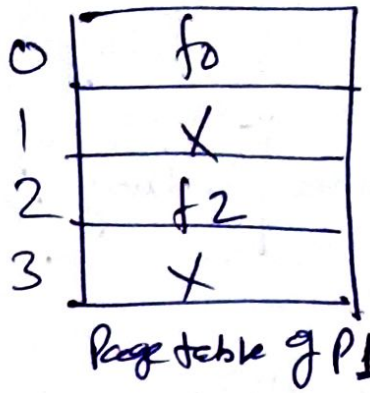
for handling address space larger than 32 bits is to use a hashed Page table, with hash value being the virtual Page no.

* Each entry in the hash table contains a linked list of elements that hash to the same location
* Each element consists of three field,
  (i) the virtual Page no.
  (ii) the value of the mapped Page frame.
  (iv) a Pointer to the next element in the linked list

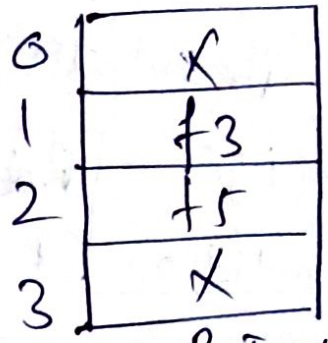


Hash
K Mod 10, K mod
mid Square,
folding Metho

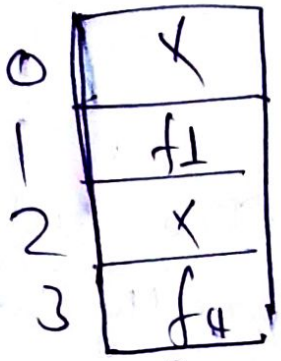
Inverted Paging



Page table of P1



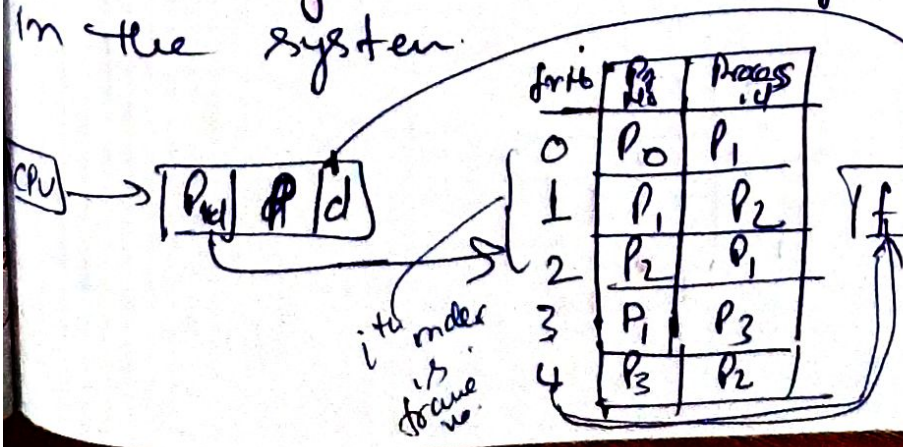
P.T of P3



P.T of P2

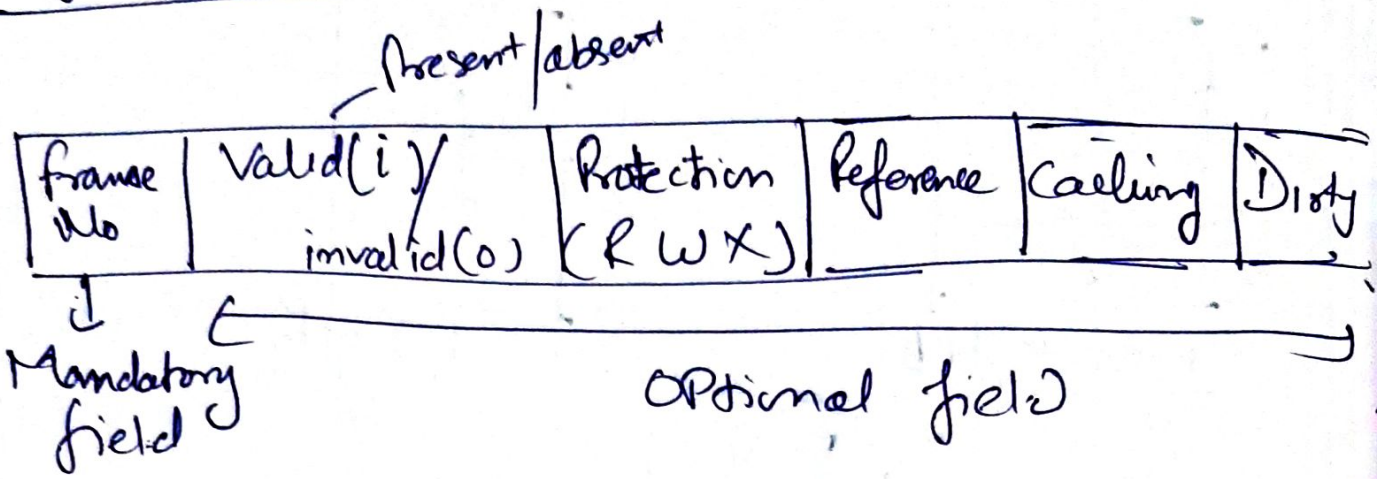
(1) Each Process has its own P.T
(2) P.T will be in main Memory
} Normal Paging

To Solve this, we can use inverted P.T.
An inverted P.T has one entry for each real Page (or frame) of memory. Thus in this way we are using only one P.T in the system.



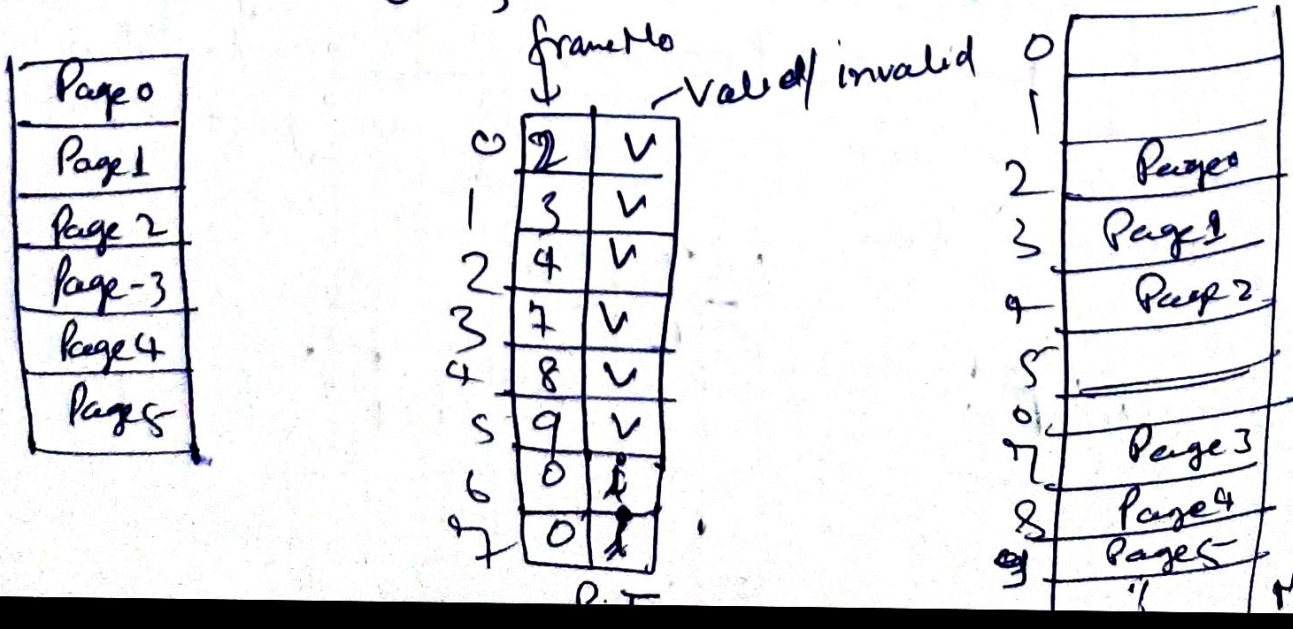
Problems Searching time (L.S)

Page table entry



Valid / invalid → One additional bit is generally attached to each entry in the Page table : a valid/invalid bit.

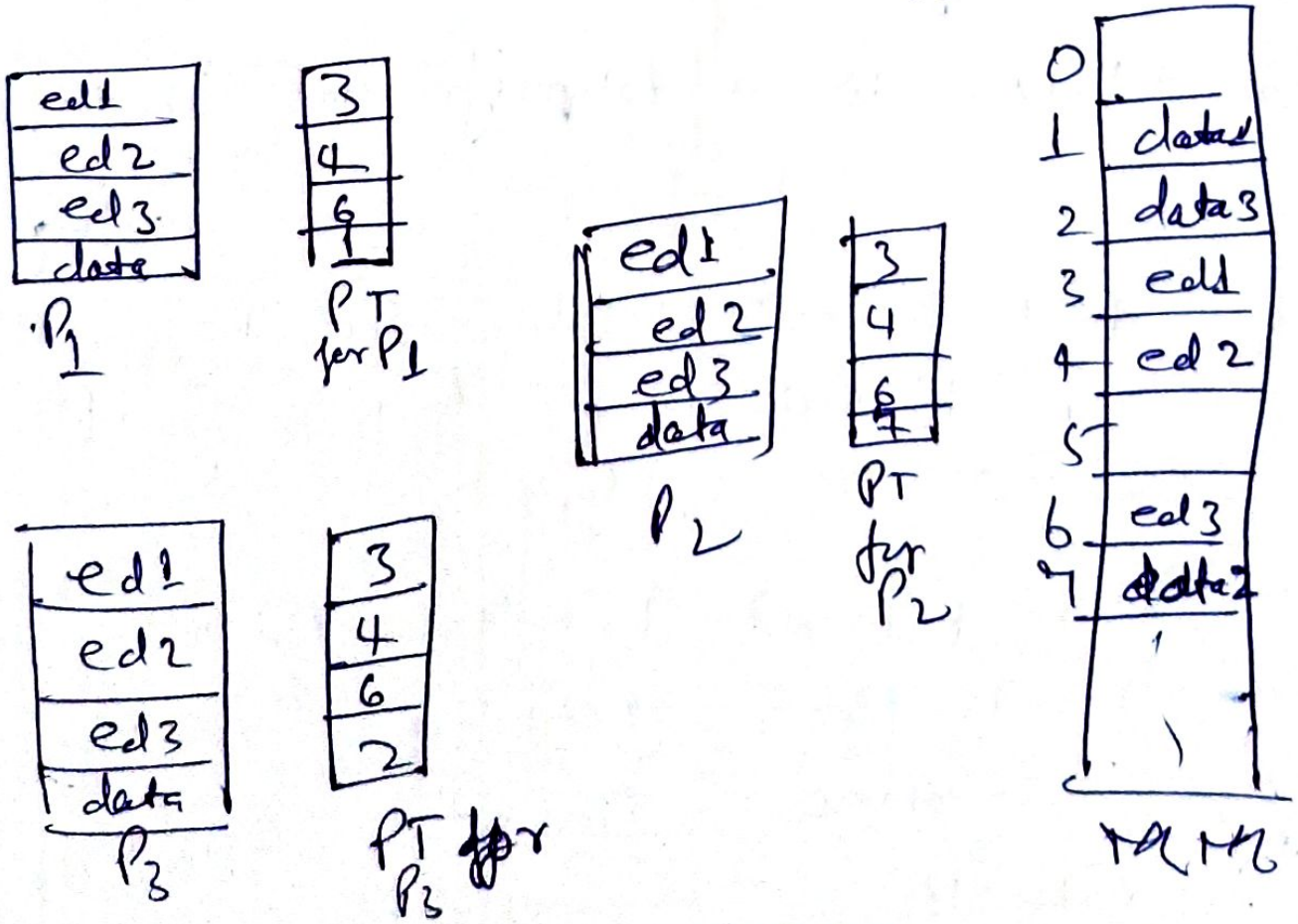
* When this bit is set to "valid" the associated Page is in the Process's logical address space and is thus legal (or valid) Page; otherwise it is **invalid**.
* Illegal attempts will be trapped to the O.S.



Shared Pages :- An advantage of Paging is the possibility of sharing common code. This consideration is particularly important in a time-sharing environment.

→ Consider a system that support 40 users each of whom executes a text editor. If the editor consists of 150KB of code and 50 KB of data space, we need 8000 KB space to support 40 users

→ but if the code is reentrant code (or Pure code), it can be shared like below (Sharing of code in Paging environment)



* Reentrant Code is not - self-modifying
It never changes during execution.
Thus, two or more Process can execute the same code at the same time.
Each Process has its own copy of registers and data storage to hold the data for the Process's execution.

* Thus, to support 40 users, we need only one copy of the editor (150KB), Plus 40 Copies of 50 KB of data space Per user.
So total Space required = 2150 KB instead of 8000 KB

So, this is a significant Saving.

Q- M.M accesstime = 100 μsec, Page fault Rate = 40%. & Page fault Service time = 4 sec

Solu^n

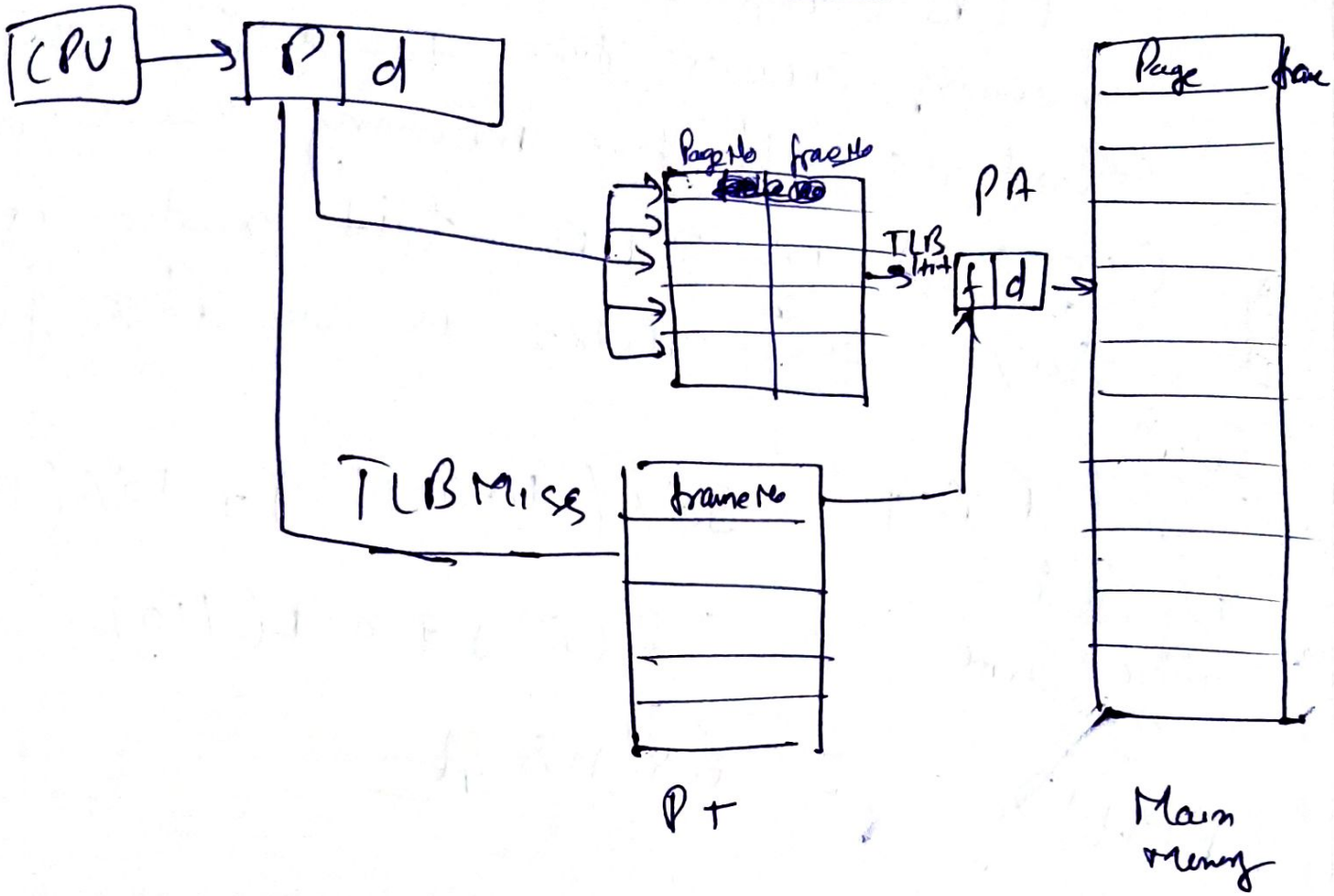
$$EMAT = P*S + (1-P)*m$$

$m = 100\ \mu sec$, $P = 40\% = \frac{40}{100} = 0.4$

$S = 4\ msec = 4000\ \mu sec$ (1 milisec = 1000 μsec)

$(0.4 * 4000)\ \mu sec + (0.60 * 100) = 1660\ \mu sec$

Translation Lookaside Buffer (TLB) is a Cache



* TLB is faster than RAM

Effective Access time

$$EAT = Hit\left(TLB_{time} + x\right) + Miss\left(TLB + x + x\right)$$

(x = memory access time)

i.e. if $Hit = \alpha$

then $$EAT = \alpha\left(TLB + x\right) + (1-\alpha)\left(TLB + 2x\right)$$

Q— A Paging Scheme using T.LB. TLB access time 10 ns and main memory access time takes 50 ns. What is effective memory access time (in ns). If TLB hit ratio is 90% and there is no Page fault

Soluⁿ

EAT = 90%(10 + 50) + 10%(10 + TS

or

Effective memory Access time (EMAT) = 0.9(60) + 0.1(110)

= 65 ns

Q— Suppose an Instruction has Page fault Service time = 10 msec and memory access time 20 nsec if one Page fault is generated for every 10^6 memory access then what is the effective access time for the memory?

Soluⁿ — PFST = 10 ms

MAT = 20 nsec

Page fault rate = $\frac{1}{10^6}$

$EMAT = \frac{1}{10^6} \times 10 \times 10^{-3} + \left(1 - \frac{1}{10^6}\right) \times 20 \times 10^{-9}$

$= 3 \times 10^{-8} - 2 \times 10^{-14}$

$= ?$

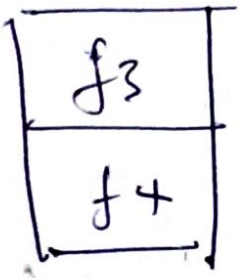
Segmentation Vs Paging

Problem with Paging

- Paging divides the process into no. of pages without caring for what is written in code of process.

Ex —

Add()

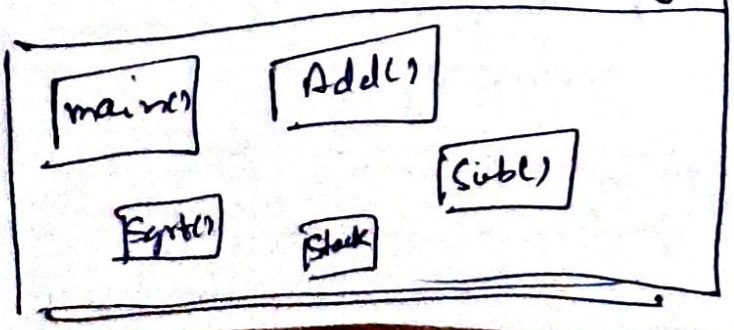


Now when CPU executes f3 it will not get all code of Add(), because it has been combined into f3 as f4.

It means after f3 we must execute f4, otherwise instructions of Add(), will not be executed.

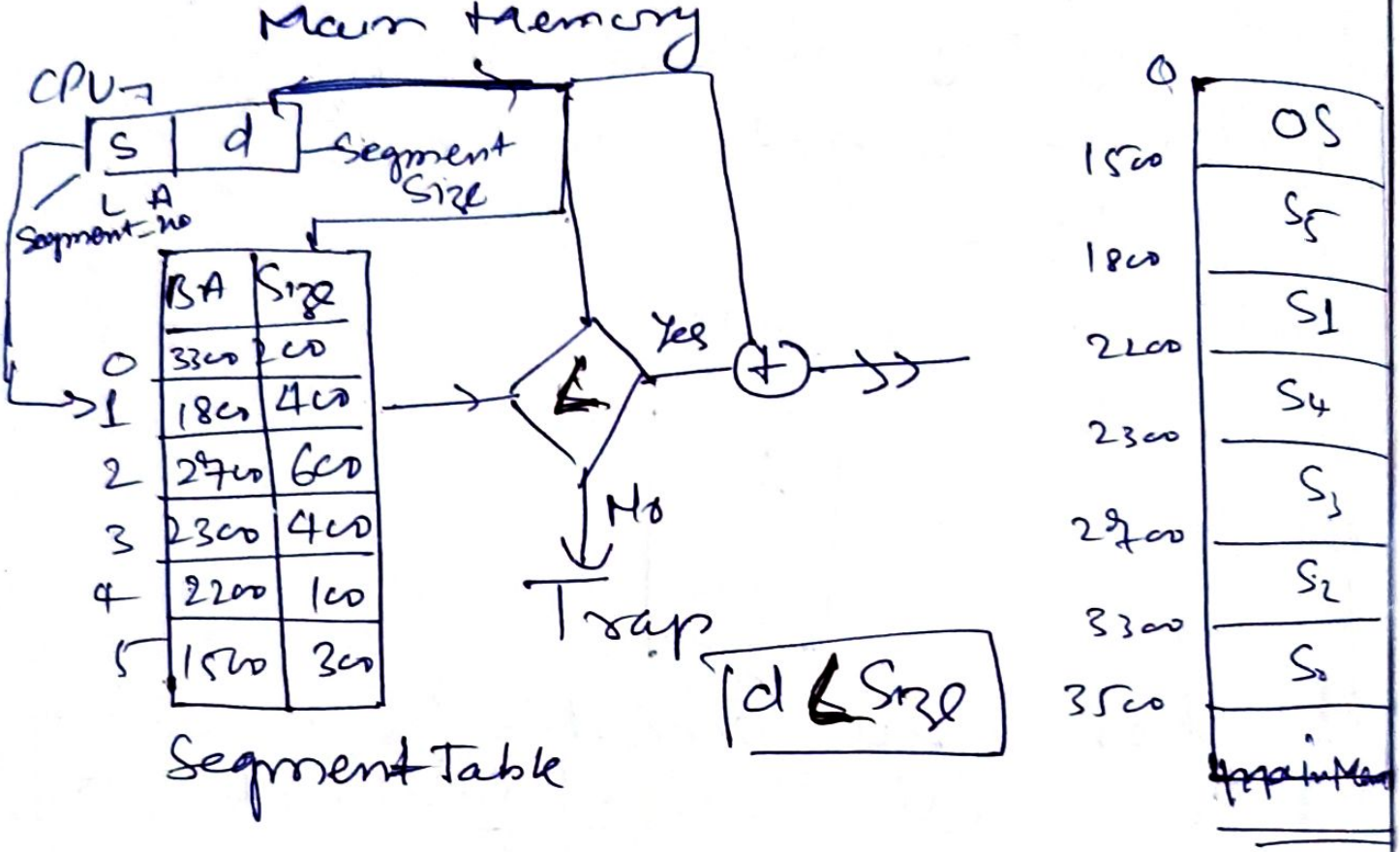
Segmentation works on user view.

* A Program is a collection of Segments



* Different Segments has different size. (But Pages are always of same size)

* we save these segments in Main memory



Segment Table

* Segment Table base register (STBR) points to the segment table's location in memory

* Segment table length register (STLR) indicates number of Segments used by a program.

Segment No. < STLR

Q- Consider the following segment table

Seg_no	Base	length
0	1219	7cv
1	23cv	14
2	90	1cv
3	1327	580
4	1952	96

which following Logical Address will Produce TRAP address in $_{seg\,no}$ error $_{offset}$

(1) 0,430
(2) 1, 11 (3) 2, 1cv
(4) 3, 425 (5) 4, 95

Soluⁿ 1 – Physical address = 1219 + 430 (No error)

2 – Physical address = 23cv + 11 (no error)

3 – TRAP (1cv < 1cv false)

4 – P.A = 1327 + 425 (No error)

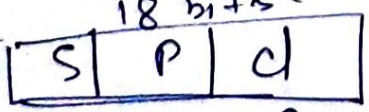
5 – PA = 1952 + 95 (No error)

(Paging with Segmentation)

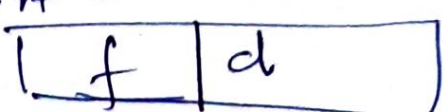
Q— A Certain Computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and Physical address spaces contain 2^{18} bytes each. The virtual address space is divided into 16 equal size segments. Assume that Page size is 512 bytes. The memory management unit (MMU) has a segment table, each entry of which contains the physical address of the Page table for the segment. Page tables are stored in the main memory and consists of 2-bytes Page table entries. How many bits are available in Page Table entry for storing the Paging information for the Page?

Soln

CA



PA



Because 16 segments $= 2^4 = 4$ bits to represent segment no

offset = Page size = 512 bytes $= 2^9 = 9$ bits

So, 9 bits for Paging information and 7 bits for various extra information

Here PT entry = 16 bits

Segmentation

→ Segmentation is a memory management technique.

→ In this process is divided in to segments

→ These segments need not to be of same size.

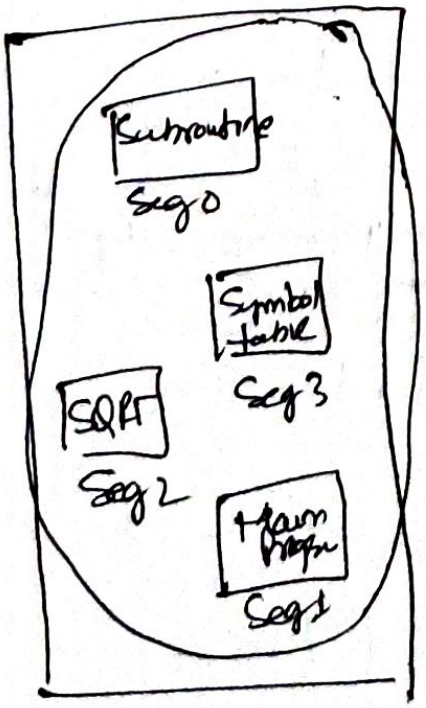
→ Segmentation use variable Partitioning.

one segment = one complete memory Block

→ Details of each segments is stored in to a 'Segment Table'

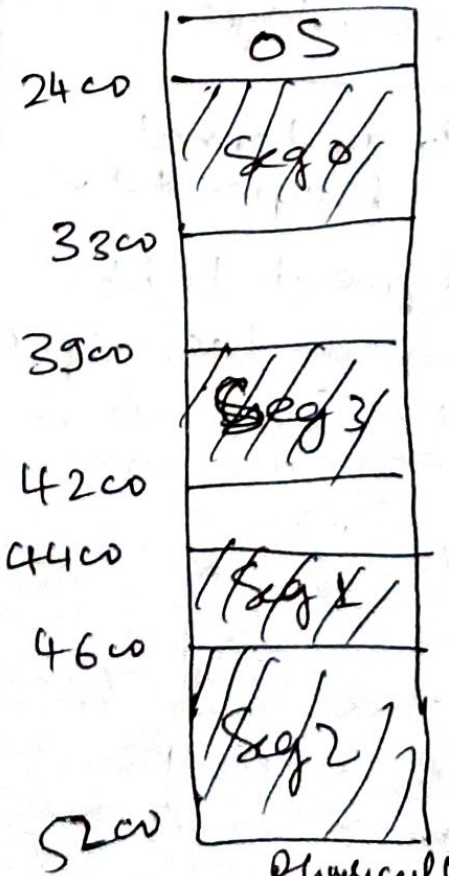
→ It contains 2 information

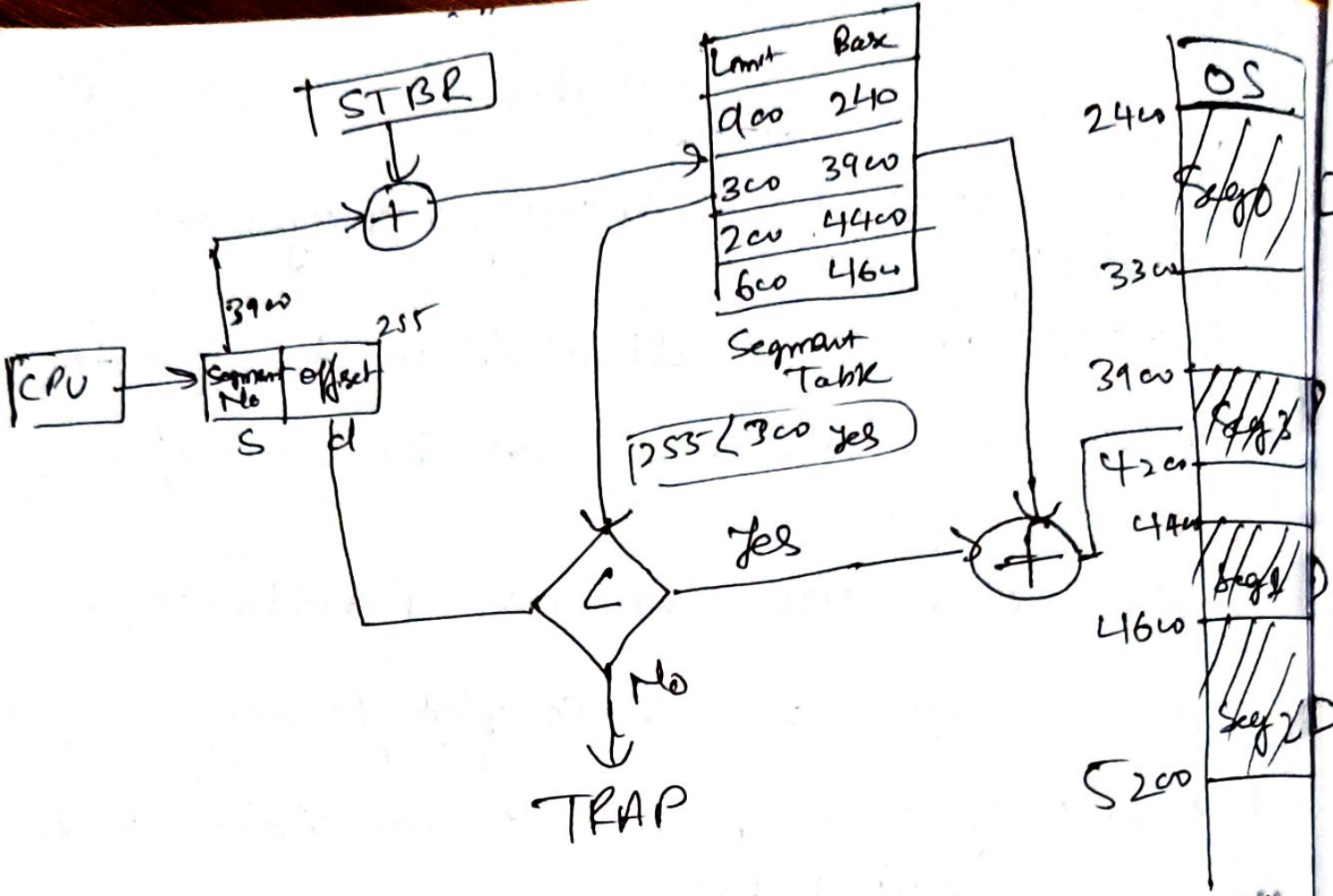
- Base — Base address of segment
- Limit — Length of the segment



Segment Table

Base	Limit
2400	900
3900	300
4400	200
4600	1000





Segmentation Hardware

Segmentation [Segment Size >>> Page Size]

Advantages	Disadvantages
① No internal fragmentation	① External fragmentation is there
② Segment table consume less space in comparision Page table in Paging.	② Costly Memory Management (To keep track of which pieces and how much memory is free)
③ Sharing of Code is easy	③ Memory is not being utilized
④ Very good protection to keep process from other processes	④ Swapping of process is not easy (To bring back same segment sized process to memory)

Paging with Segmentation

(To overcome from external fragmentation)

Process → Divided → Segments → Divided → Pages → Stored → Main Memory

1 Page table for 1 segment which keep track of frames storing pages

Each Page table occupies one frame in main memory

No of entries in Page table of Segment = No of Pages that segment is Divided

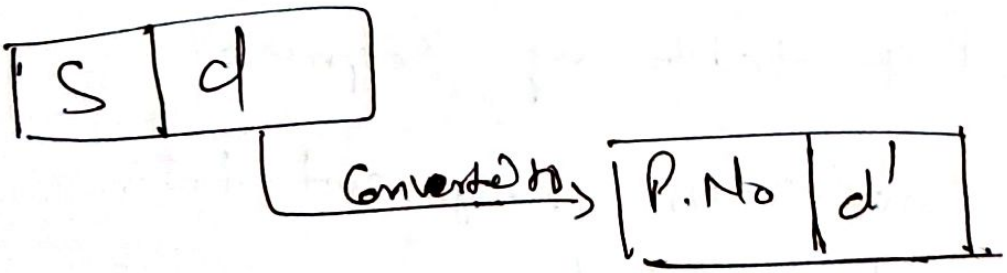
Segment table keeps tracks of frame storing Page table of segment

No of entries in segment table = No of segment

Base address of Segment table is Stored in STBR → Segment table Base Register

---

Q- On a system using Paging and segmentation the virtual address space consists of upto 16 segments where each segment can be upto 2^{16} byte long. The hardware pages each segment into 512 byte Pages. How many bits in virtual address space



Specify the following

(i) Segment No (ii) Page no (iii) offset within Page (iv) Entire virtual address

Soln → (i) 16 segments = 2^4 i.e 4 bits are needed to specify segment No.

S	P	d
4 bit	7 bit	9 bit

(ii) Page No → 512 byte Page = 2^9 byte Page offset

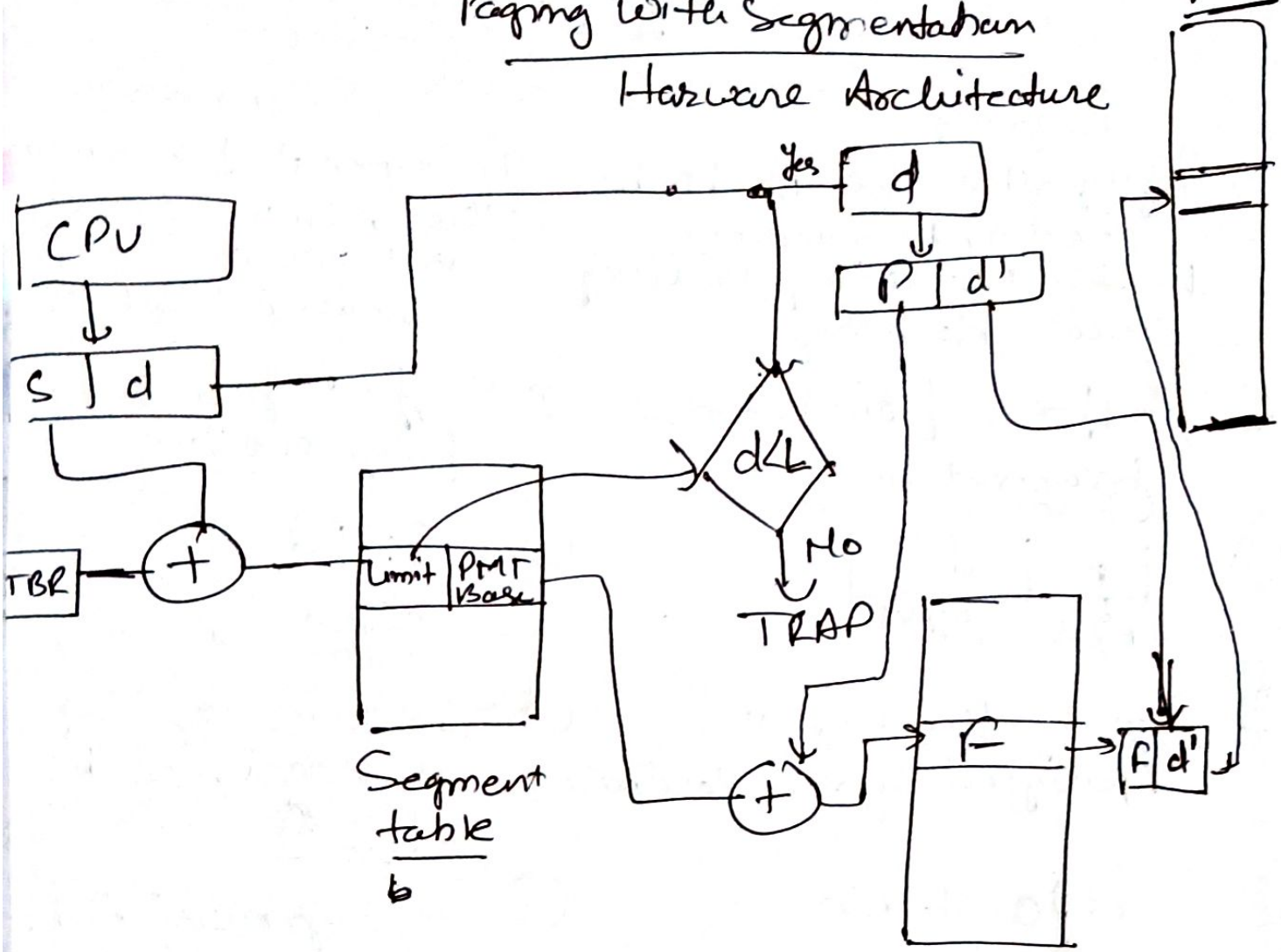
No of Pages in each segment = $\frac{\text{Size of each segment}}{\text{Size of each page}}$

$= \frac{2^{16}}{2^9} = 2^7$

i.e 7 bits are required to specify Page no

Paging With Segmentation

Hardware Architecture

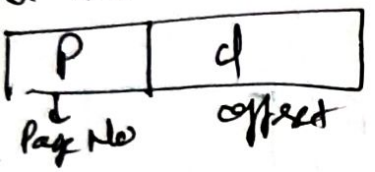
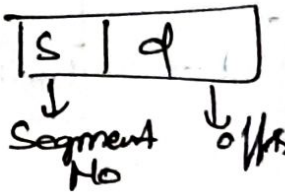


H/W Architecture { Page Map table of Segment

Paging With Segmentation

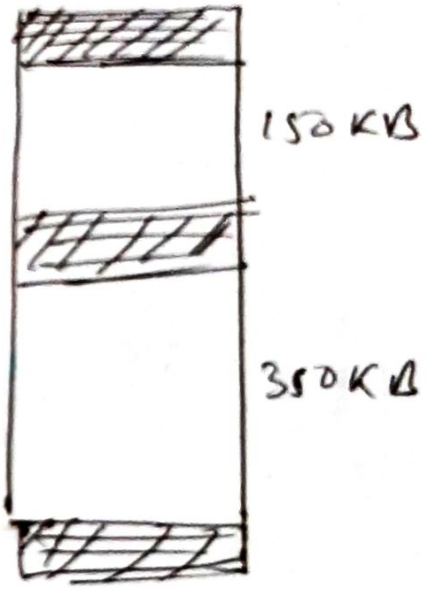
Advantage	Disadvantage
External fragmentation is not there	→ It Suffered from Internal fragmentation
Segment table has only one entry corresponding to one actual Segment	→ Complexity level is higher than others
Page table size is limited by Segment Size	→ Costlier than Segmentation & Paging
Protection to user	
Sharing of Data	

Difference Between Paging and Segmentation

Paging	Segmentation
(1) Paging allows memory to be divided into fixed sized blocks i.e. a Page is always fixed block size.	(1) Segmentation divides the memory space into segments of variable block size.
(2) Suffers from internal fragmentation	(2) Suffers from external fragmentation
(3) 	(3) 
(4) Size of the Page is specified by the Hardware	(4) The size of Segment is specified by User
(5) Page table	(5) Segment table
(6) Paging is invisible to the user	(6) Segmentation is is visible to user
(7) Paging is done by OS	(7) Segmentation is done by user.
(8) Paging is faster than Segmentation	(8) Segmentation is slower than Paging.
(9) There is no external fragmentation	(9) It suffers from external fragmentation
(10) frame of Pages ↓ Physical memory	(10) Variable sized blocks of logical memory are called segments.

Q- Req. for various Processes are
300K, 25K, 125K, 50K respectively
the above request could be satisfy
with

(A) Best fit but not first fit
(B) First fit but not best fit
(C) Both
(D) None



Ans First fit

Solve the same using worst fit ~ yes!
~ It's working

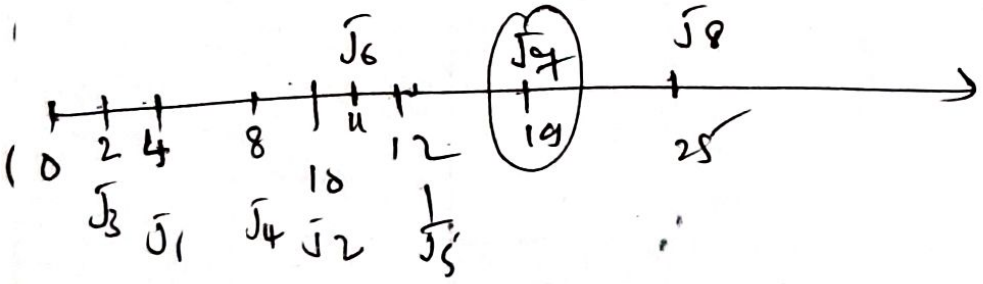
(III) Entire virtual address = Segment no + Page no + offset
= 4 + 7 + 9 = 20 bit ✓

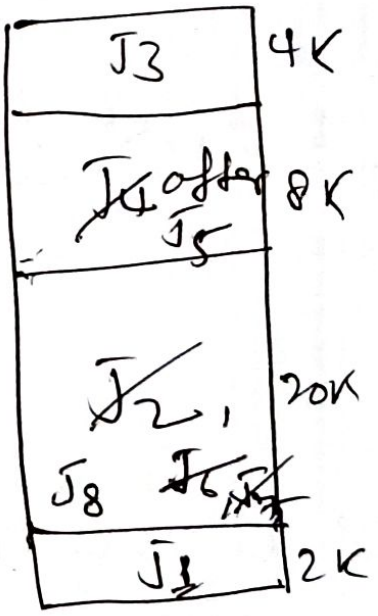
Q-

Reques No	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Request Size	2K	14	3K	6K	6K	10K	7K	20K
usage time	4	10	2	8	4	1	8	6

Best fit

Calculate the time at which J_7 will be completed





⇒ J_4 at 8 time will be completed and J_5 will arrive and replace the place of J_4

⇒ J_2 at time 10 will be completed and J_6 will be put in at that place

⇒ J_7 will be put at 20K Slot because it is completed at time 11 and we can put J_7 at in this slot.

We use overlay in generally in embedded system or working Machine

Overlays :- It is a method by which a large size process can be put into the main memory.

9- Consider a two Pass assembler

Pass 1 : 80 KB, Pass 2 : 90 KB

Symbol table :- 30 KB

Common routine : 20 KB

At a time only one Pass is used.

What is min Partition size required if overlay driver is 10 KB size.

Total Size

80
90
30
20
10

230 KB (Max Memory size)

Pass 1	Pass 2
80	90
30	30
20	20
10	10
140 KB	150 KB

So, we need 150 KB Minimum to Perform the ~~code~~ function

Virtual Memory in OS

* Virtual memory in O.S is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory.
* The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage.
* It is a technique that is implemented using both hardware and software. It maps memory address used by a program called virtual addresses, into physical addresses in computer memory.

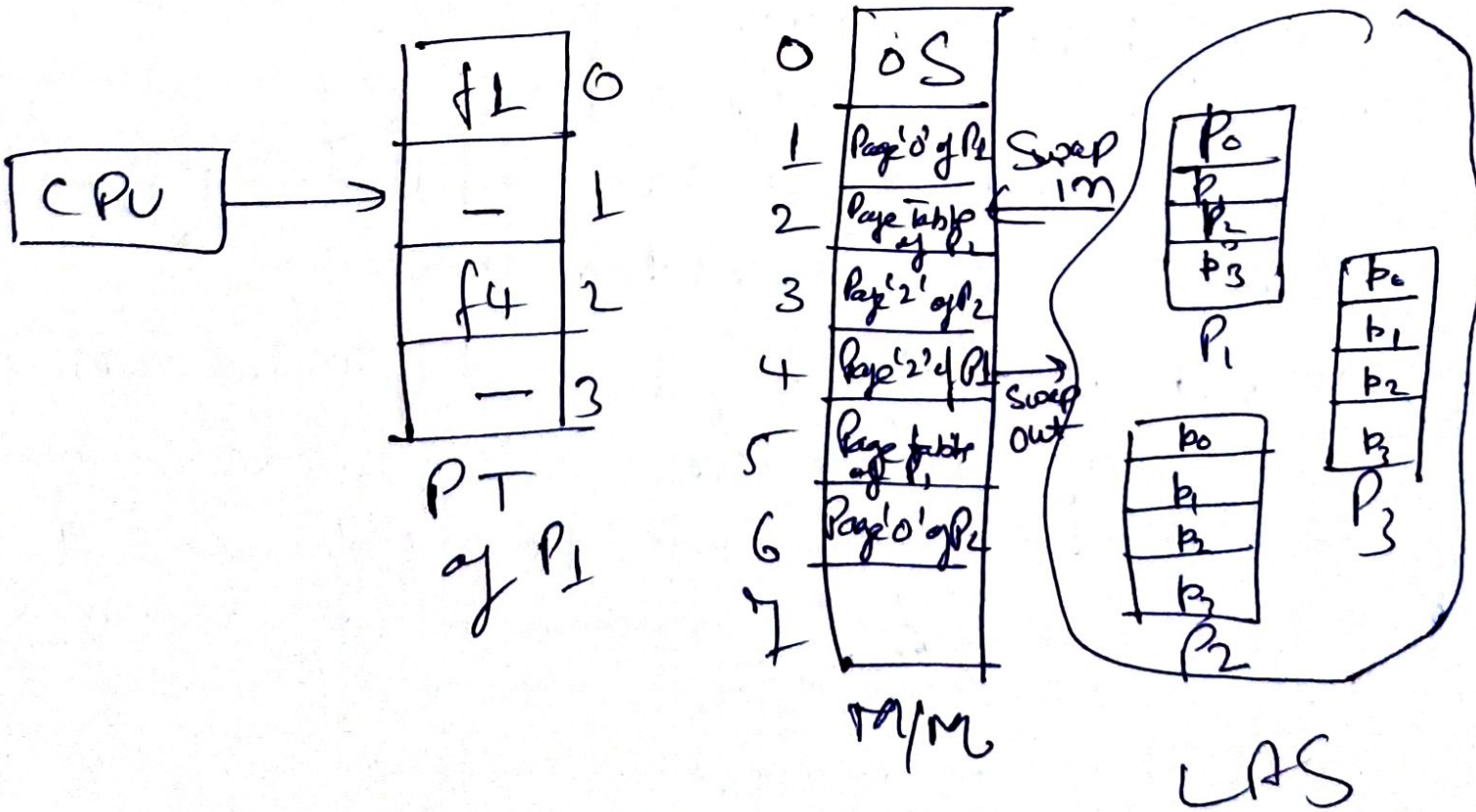
(1) All memory references within a process are logical address that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of memory such that it occupies different places in main memory at different times during the course of execution.

(2) A process may be broken into a no.

of Pieces and these pieces need not be Continuously located in the main memory during execution.

So, the required Pages need to be located into memory whenever required.

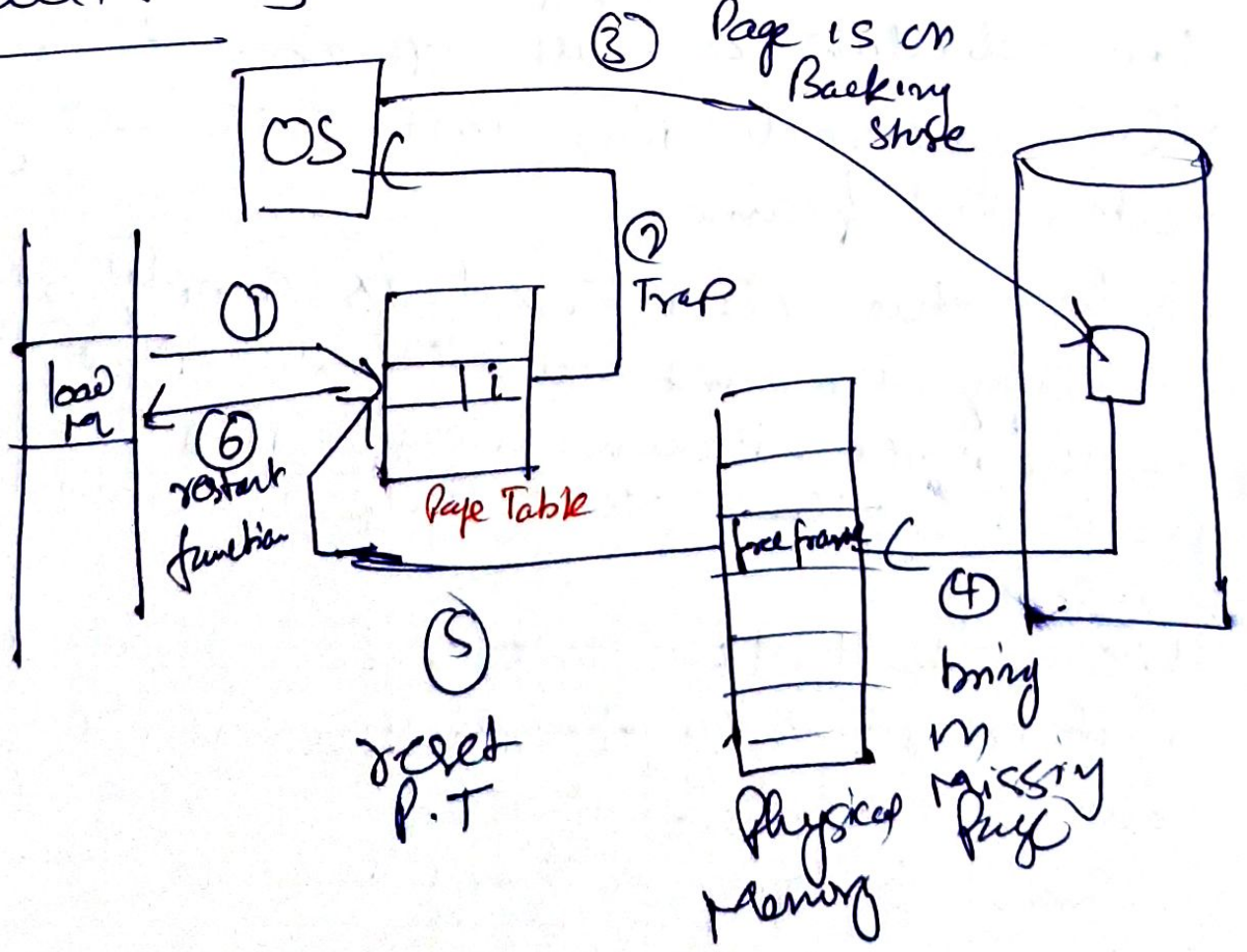
* Tip Virtual memory is implemented using Demand Paging or Demand Segmentation.



Demand Paging → load Pages only as they are needed.

* Pages are only loaded when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.

Steps in handling a Page fault →



(1) We check an internal table (Usually we kept with PCB) for this Process to determine whether the reference was a valid or an invalid memory access.

(2) If the reference was invalid, we terminate the Process. If it was valid, but we have not yet brought in that Page, we now Page it in.

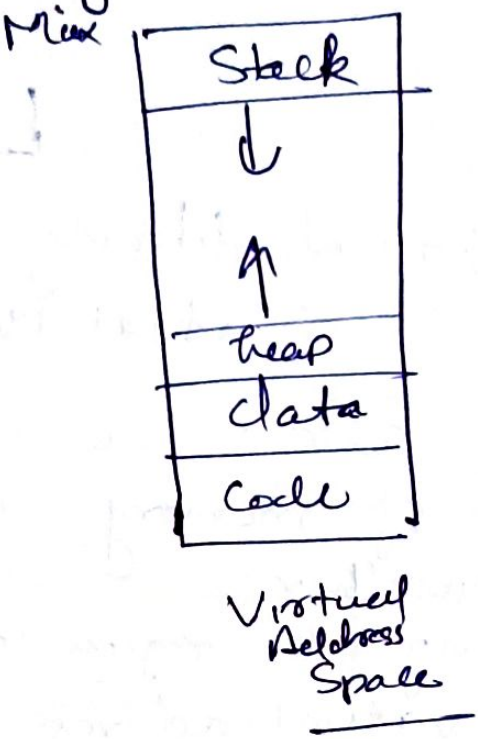
(3) We find a frame (by taking one from free frame list, for example)

(4) We schedule a disk operation to read the desired Page into the newly allocated frame.

(5) When the disk read is complete, we modify the internal table kept with the Process and the P.T to indicate that the Page is now in memory.

(6) We restart the instruction that was interrupted by trap. The Process can now access the Page as though it had always been in memory.

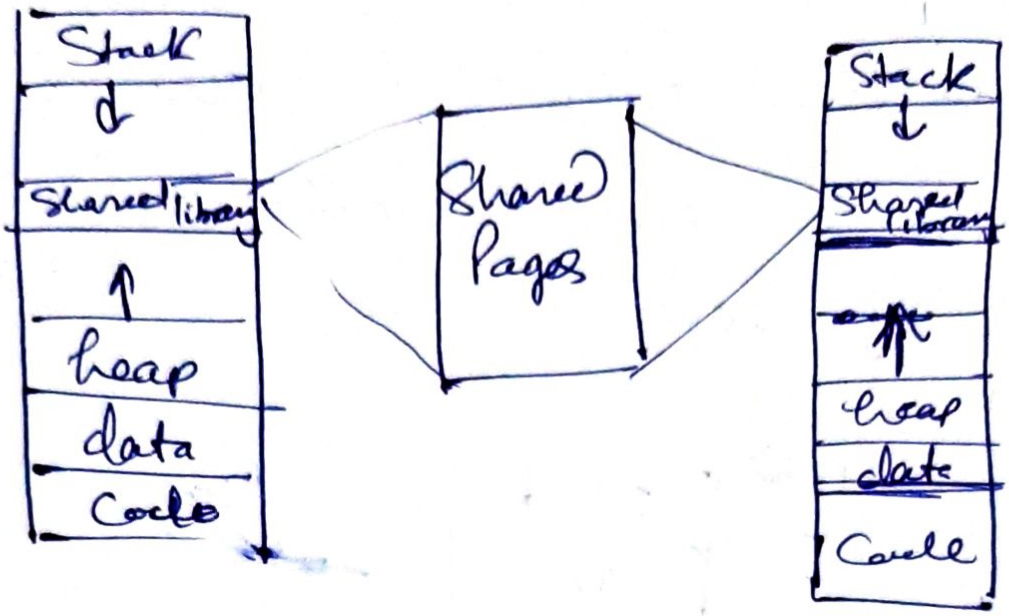
The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.



* In addition to separating logical memory from physical memory, virtual memory allows files and memory to be shared by two or more processes through Page sharing. This leads to following benefits:

* System libraries can be shared by several processes through mapping of shared object into a virtual address space.

• → Typically, a library is mapped read only into the space of each process that is linked with it



Shareed library using Virtual Memory.

* Virtual memory allows one process to create a region of memory that it can share with another. Processes sharing this region consider it part of their virtual address space.

Performance of Demand Paging :

Let 'P' be the Probability of a Page fault ($0 \leq P \leq 1$). We would expect 'P' to be close to zero — that is a few Page faults. The effective access time.

$$EAT = (1 - P) \times ma + P \times \text{Page fault}$$

ma → Memory Access time

P → Page fault Rate

Page fault → servicing

Page Replacement Algorithms :

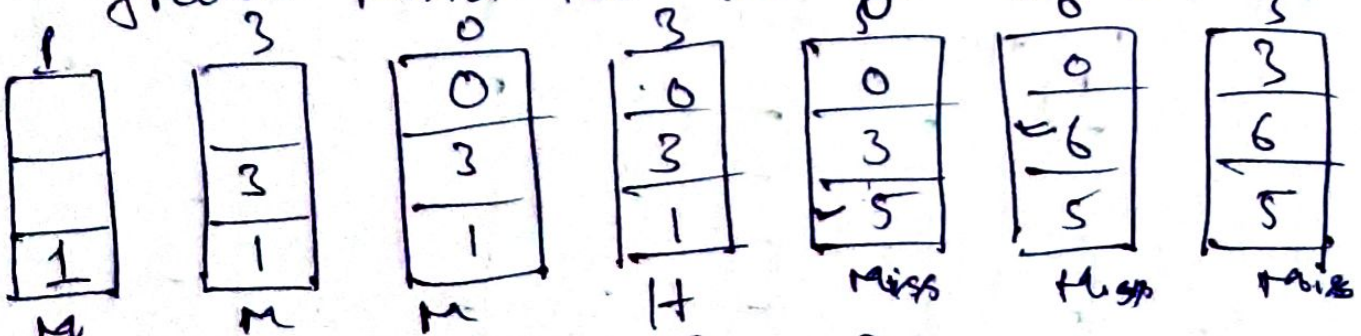
Page fault - A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.

In case of page fault, O.S might have to replace one of the existing pages with the newly needed pages.

1- First in First Out → (FIFO)

In this algorithm, the O.S keeps track of all pages in the memory in a queue, the oldest page is in front of the queue. When a page needs to be replaced page in front of the queue is selected for removal.

Ex— Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the no of page faults.



Total page fault = 6

Belady's Anomaly :- Belady's anomaly

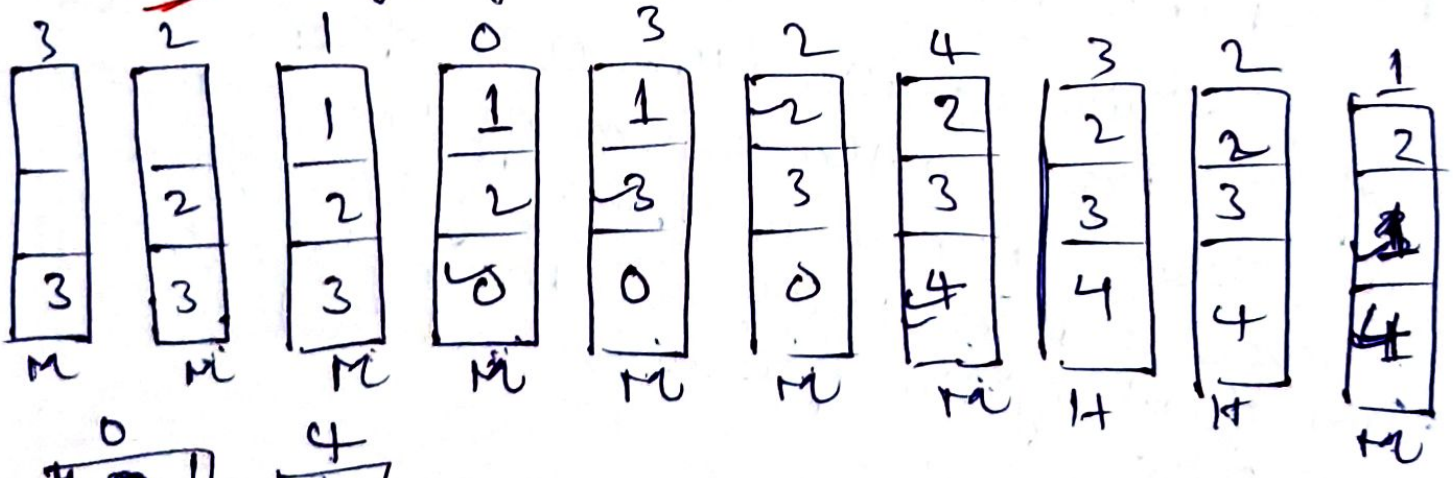
Proves that it is possible to have more Page faults when increasing the no. of frames while using the (FIFO) Page replacement algo.

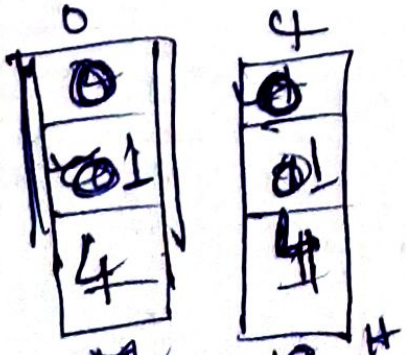
Ex

String : 3 2 1 0 3 2 4 3 2 1 0 4

Slot = 3 , we get total 9 Page fault

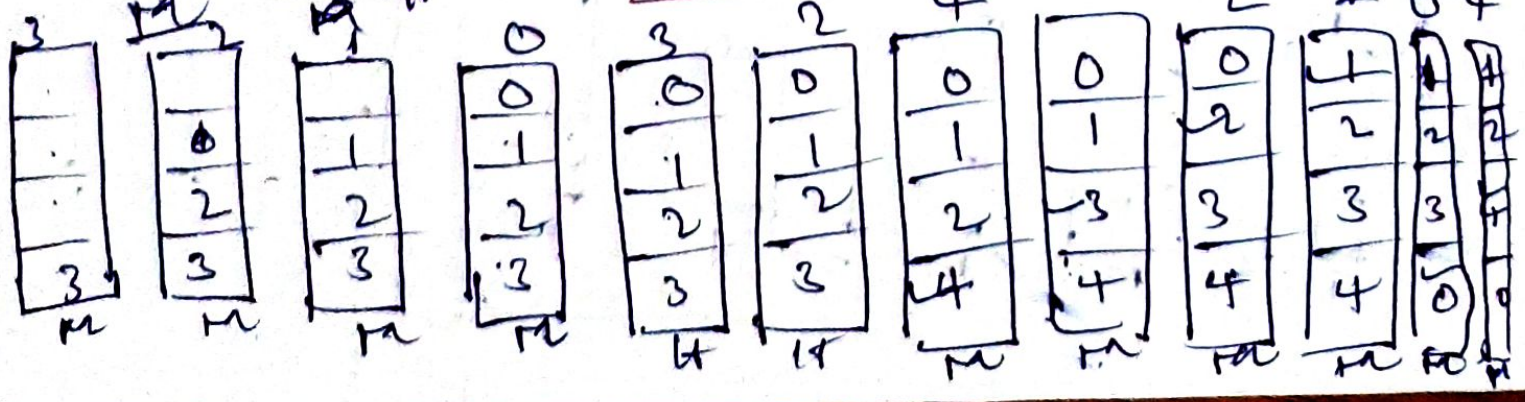
⇒ If we increase slots to 4, we get 10 - Page faults.





Page fault = 9

but with 4 slot P. fault = 10

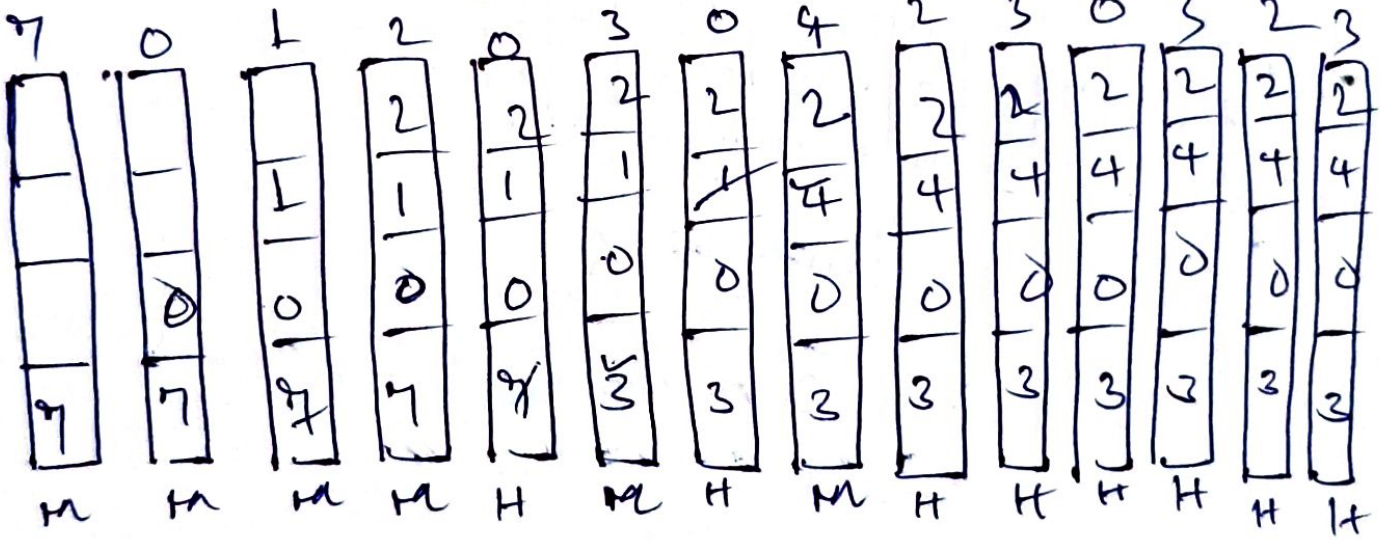


Optimal Page Replacement: In this algorithm, pages are replaced which would not be used in the longest dimension of time in the future.

Scan pages →

Page reference: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3

No of frame = 4



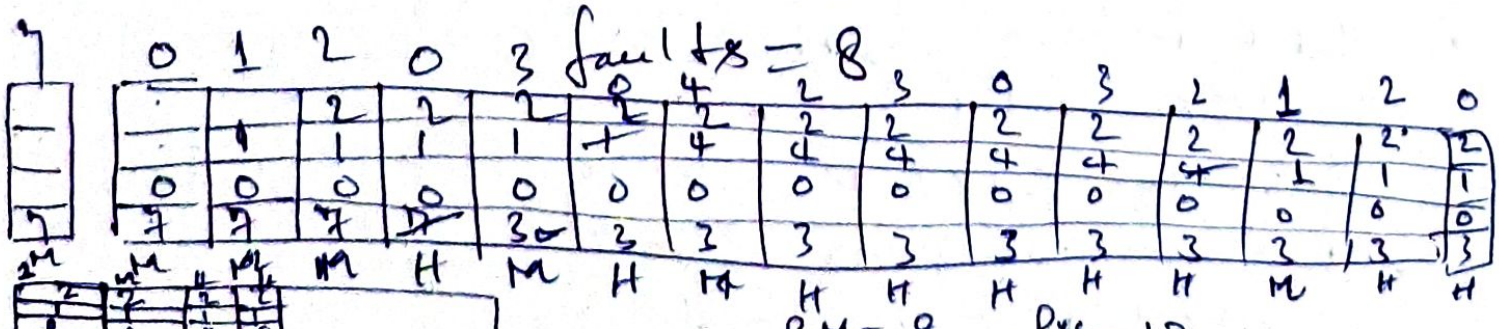
Total P. Fault = 6

Ex— 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

No of frame = 4

Solⁿ → Hit = 12

fault = 8



PM = 8, PH = 12

(3) Least Recently Used - In this algorithm, Page will be replaced with the least recently Used page in Past

Ex- String : 7 0 1 2 0 3 0 4 2 3 0 3 2

frame = 4

7	0	1	2	0	3	0	4	2	3	0	3	2
			2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3
M	M	M	M	H	M	H	M	H	H	H	H	H

Page fault = 6

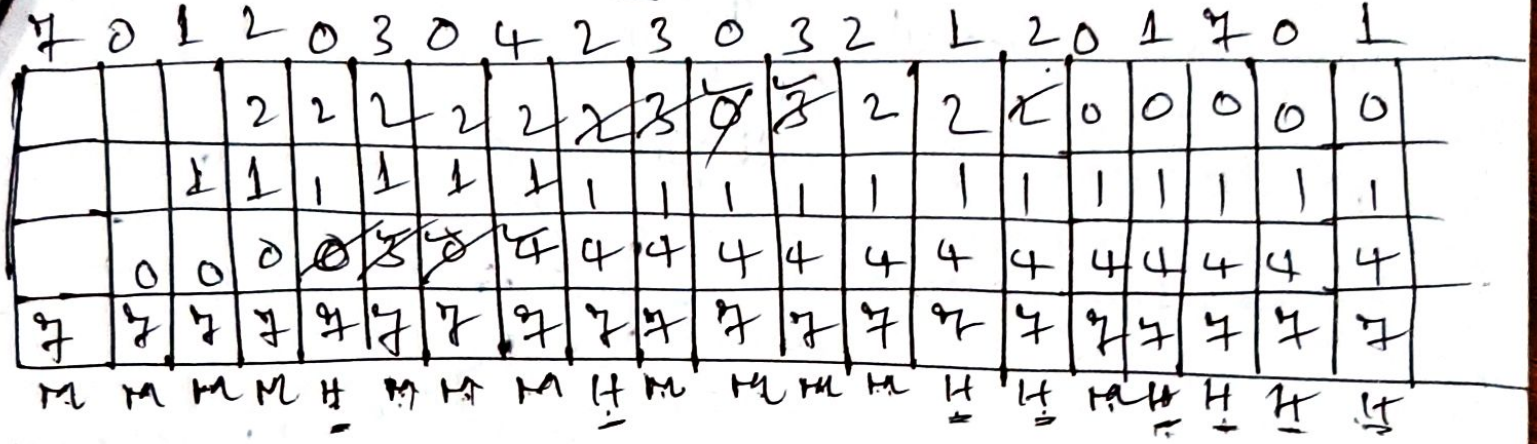
Ex - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Soln Page fault = 8
Hit = 12

* LRR performs better in case in Page fault then FIFO
* But its speed is slower then FIFO because it searches for Least Recently used Page in Past.

Most Recently Used : Replace the most recently used Page in Past.

Ex — 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



Page fault = 12

Hit = 8

Belady's Anomaly → Ref = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

frame = 3

Hit = 3

Pf = 9

the frame = 4

Hit = 2

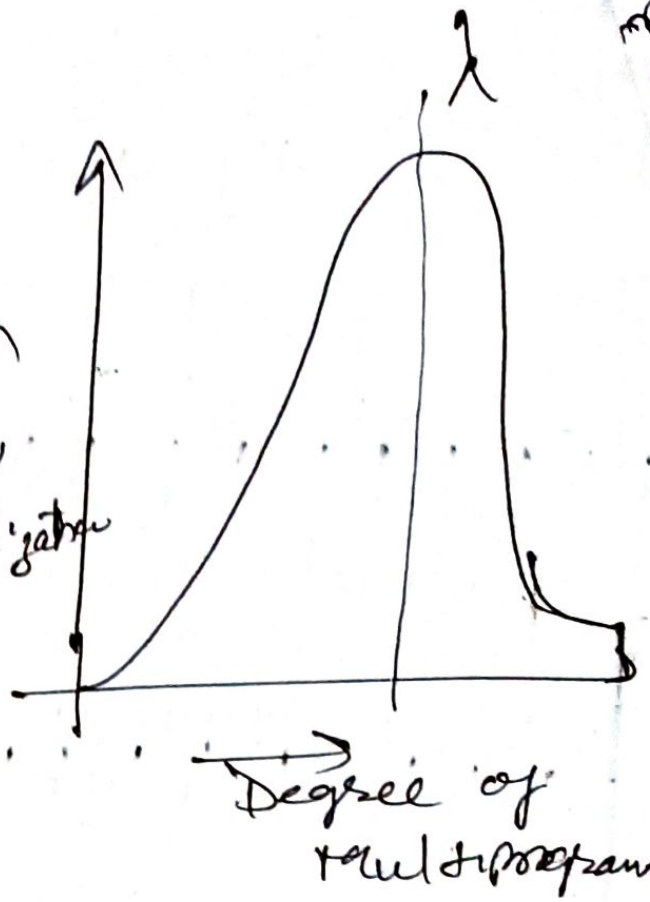
Pf = 10

Thrashing :-

How to Control Thrashing?

(1) Increase Memory Size

(2) Long term Scheduler
↓
to decrease the degree of Multiprogramming



* In case, if Page fault and swapping happens very frequently at a higher rate, then the O.S has to spend more time swapping these Pages. This state in O.S is termed as thrashing. Because of thrashing the CPU utilization is going to be reduced.

* During thrashing, the CPU spends less time on some actual Productive work and spend more time swapping

* The Process Scheduling Mechanism tries to load many Processes into the memory at the same time due to which degree of Multiprogramming can be increased. Now in this situation, there are more

processes in the memory as compared to the available no. of frames in the memory. Here we can say that as soon as memory fills up, the process starts spending a lot of time for the required Pages to be swapped in. In this way CPU utilization becomes low because most of the processes are waiting for Pages.

IoP

Thus a high degree of multiprogramming and lack of frames are two main causes of thrashing in O.S

Techniques Used to handle Thrashing in OS →

(1) Working Set Model →

The goal is to find a no of frames that will be enough to execute the current localities with a limited no of Page faults:

In this model we find the working set window, known as

Δ Delta

* Δ length should be optimal that is neither too small to hold all of the Pages of current locality nor too large to include Pages that don't belong to locality.

Ex- Let a Process P1 the sequence of Page reference be 5, 7, 2, 4, 4, 3, 2, 3, 1

- Let's find the current working set for last Page reference.

(i) If Δ is fairly large [redacted], then working set [redacted] Could be like below -

$$W.S = [5, 7, 2, 4, 3, 1]$$

* Here we have Pages 5, 7 which are hardly referenced

(ii) Again if, Δ = 2 we get WS = [1, 3]
we miss frequently visiting Pages like 4 and 2

(iii) Thus optimal value of Δ can be 5
referencing 4, 3, 2, 3, 1
WS = [2, 4, 3, 1]

So, we can see current WS is different for different Page reference

once the length of W.S is calculated for each process, we do sum to find the total demand, D

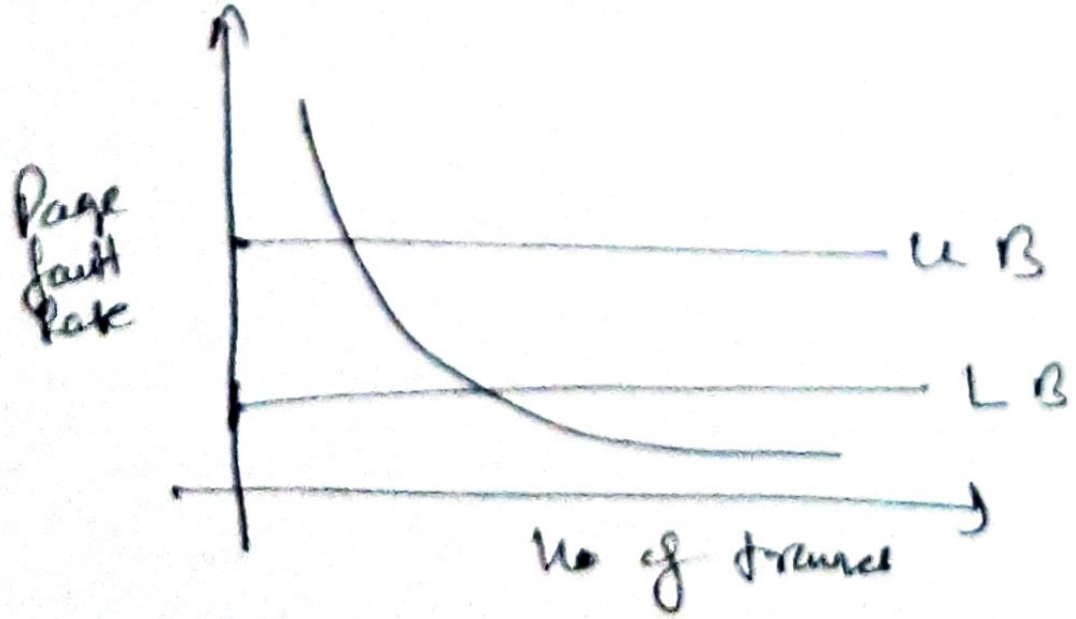
Let 'f' be the total no. of frames available in main memory for allocation then,

⇒ if D > f ⇒ At least one process is thrashing

⇒ if D < f ⇒ No thrashing.

* NB — The challenge in this model is keeping track of the constantly changing current working sets for each process

(2) Page fault frequency :-



In this model we set (UB) and (LB) Lower Bound for Page fault rate(R).

* for each Process we Compare Page fault Rate (R) with UB and LB

(i) If R > UB, then we can Conclude that a Process needs more frames to control this rate. This means we need to allocate more frames to it to avoid thrashing. If no frames available, the Process can be suspended until available

(ii) If R < LB, we have more than a sufficient amount of frames for a Process and Some of it can be allocated to Some other Processes