LEC-24 | Memory Management

Page No.
Date: / /

* In Multiprogramming environment, we have multiple process in the main memory (Ready Queue) to keep the CPU utilization high & to make computer responsive to the users.

* To realize this increase in performance, however, we must keep several processes in the memory; that is we must share the main memory. As a result we must manage main memory for all the different processes

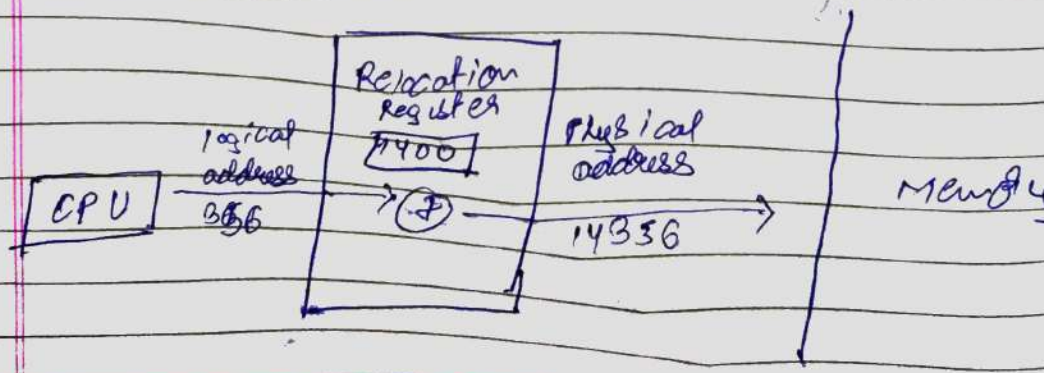
Address Space

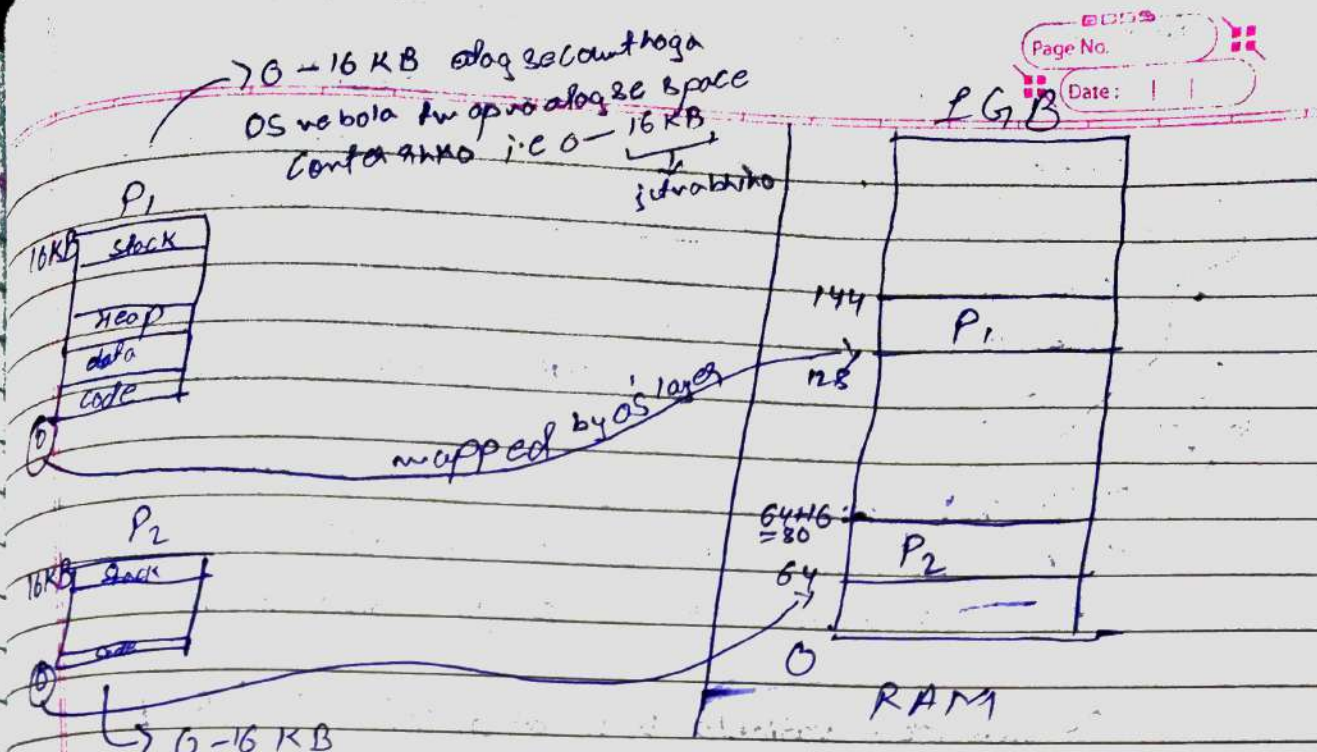
* Logical Address
  - An address generated by CPU
  - The logical address is basically the address of an instruction or data used by a process.
  - User can access logical address of the process.
  - User has indirect access to the physical address through logical address.
  - Logical address does not exist physically. Hence, also virtual address.
  - The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
  - Range 0 to max.

* Physical Address.
- An address loaded into the memory address register of the physical memory.
- User can never access the physical address of the program.
- The physical address is in the memory unit. It's location the main memory physically.
- A physical address can be accessed by a user indirectly but not directly.
- The set of all physical address can be accessed by a user indirectly but not directly, known as Physical address space.
- It is computed by memory Management unit (MMU)
- Range: (R+0) to (R+max), R is a base value.
  ↳ base value

* The run time mapping from virtual to physical address is done by a hardware device called the memory management unit.

- The user program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.





Ye kaise kya milega aur ?

=> P_1 allocate => Base of RAM (Physical address space).

B -> 128

offset -> 16

=

=> P_2 -> B -> 64

offset -> 16

For isolation

let.

P_2 address => 0 + Random value (let 129) (0+129) -> Written in P_2 Code

=> Address => 129

Ab ye 129 jo h, ab ye base wale add hoga kyuki

0 -> 64 zero jo h 64 se mapped h

=> 128 mapped hoga 64 + 129 pe. (193)

actual physical address.

isKa mtlab hua P_2 ka programmer tha wo apne program wale usne (129) likha hoga i.e random value address ki

i.e 0+129 pe koi address or wo mujhe access karna h

ab OS dekhega 0+129 = 129 kya ye P_2 se belong bhi karta h

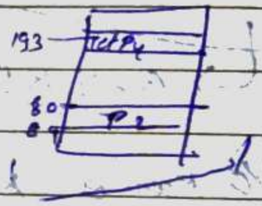
uske liye OS kya karega ye P_2 ke corresponding ek base address nikal ke aaega i.e (64)

done MMU

64 + offset → jo nikalna chaha raha h address
⇒ 193 aaega

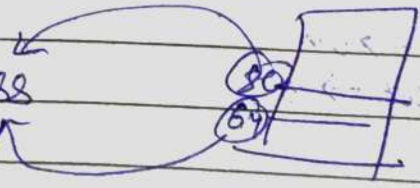
ab OS check karega ki iski max value kitni thi i.e 80 but ye kaha pe access karna chaha raha h 193[th] address location ko access karna chaha raha h P_2 jiski max range thi 16Kb

OS mn doo variable base or offset ko use karke ye dekh leta h ki P_2 program jo h kya wo out of bound i.e Not allowed space pe access karne ki koshish th nhi kr raha h



⇒ OS will throw isolation.

→ Physical address



How OS manages the isolation and protect? (Memory Mapping & Protection)

→ OS provides this virtual address space (VAS) concept.

→ To 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 only these legal addresses.

→ The relocation register contains value of smallest Physical address (Base address [R]); the limit register contains the range of logical addresses (offset) (Eg; relocation = 100040 & limit = 74600).

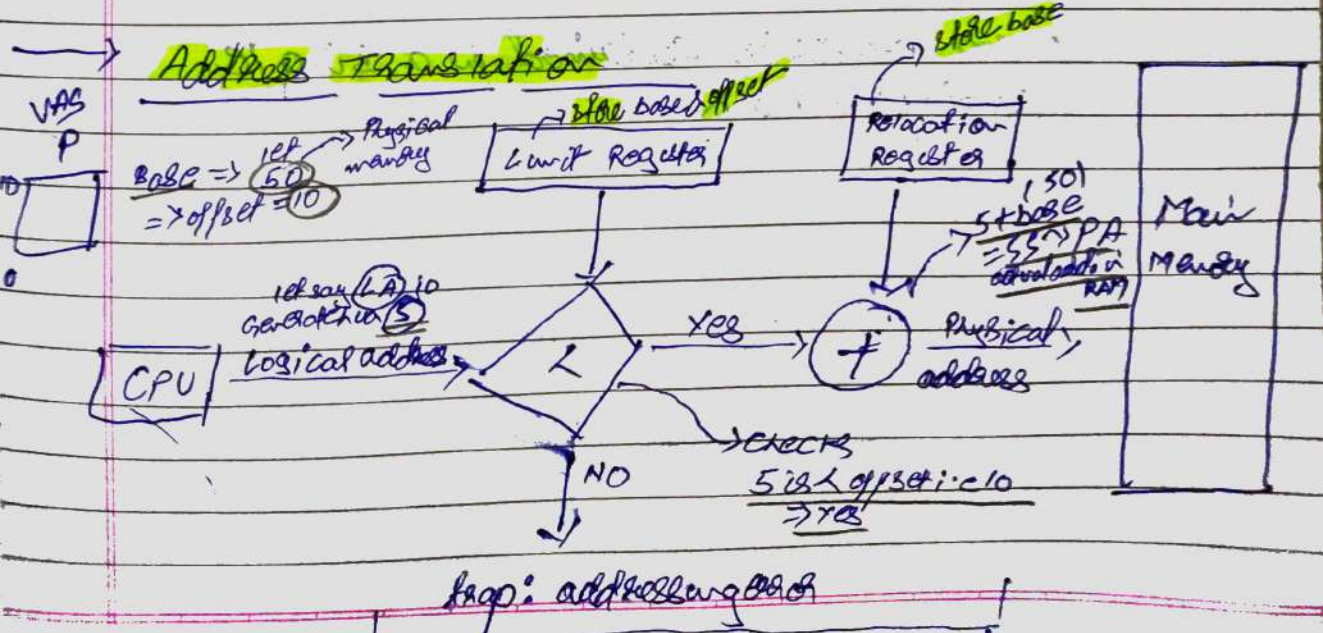
→ Each logical address must be less than the limit register.

→ MMU maps the logical address dynamically by adding the value in the relocation register.

→ When CPU scheduler selects a process for execution, the dispatcher loads the relocation & limit CPU(logical address) is checked against these registers, we can protect both OS & other user's programs & data from being modified by running process.

→ Any attempt by a program executing in user mode to access the OS memory or other users' memory results in a trap to the OS, which treat the attempt as a fatal error.

→ Address Translation

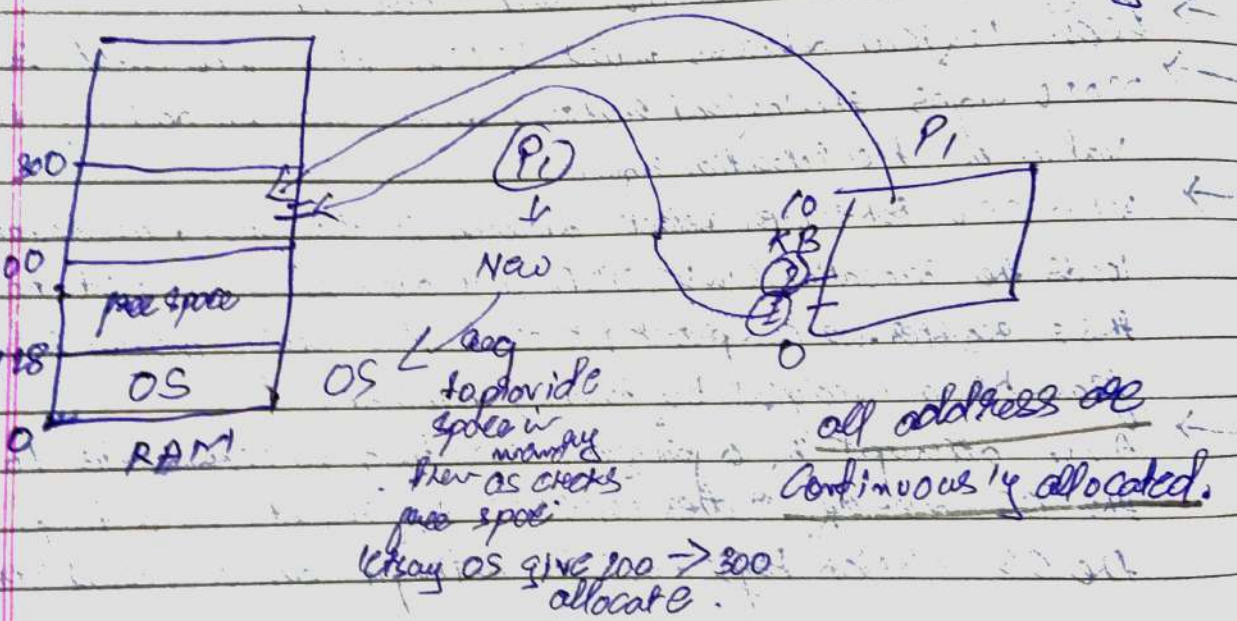


Allocation Methods of Physical Memory

1. Contiguous memory allocation
2. Non-contiguous memory allocation

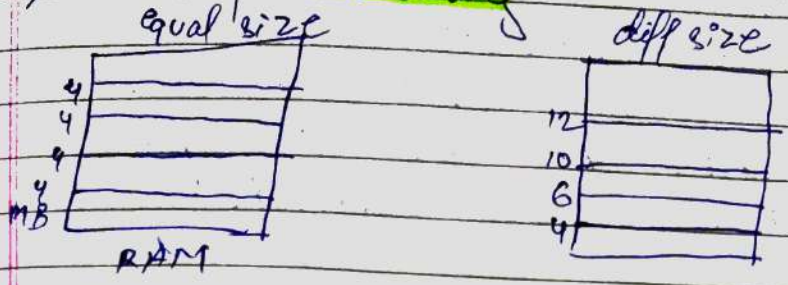
Contiguous Memory allocation

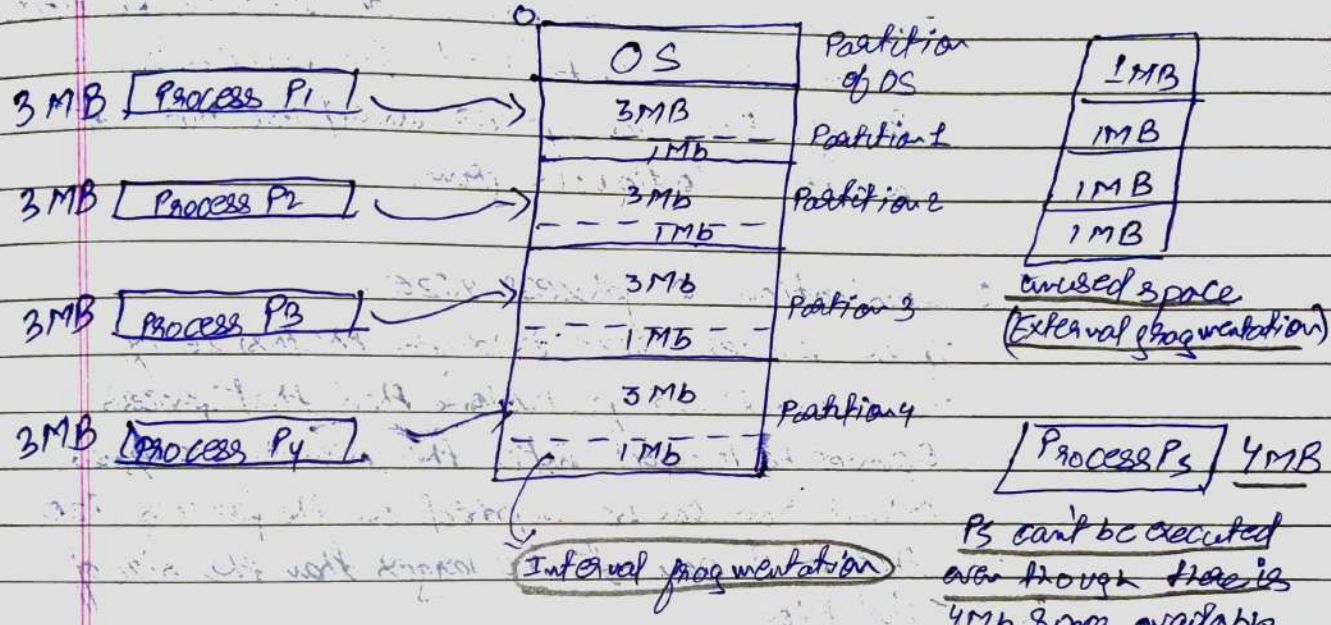
- In this scheme, each process is contained in a single contiguous block of memory.



- each process is contained in a single contiguous block.

→ Fixed partitioning





- Process allocate kewe se pde hi fixed partition bava skkin
- ek partition wae ek hi process aa skta h
  ⇒ Itne no. of partition honge utne hi no. of process honge.

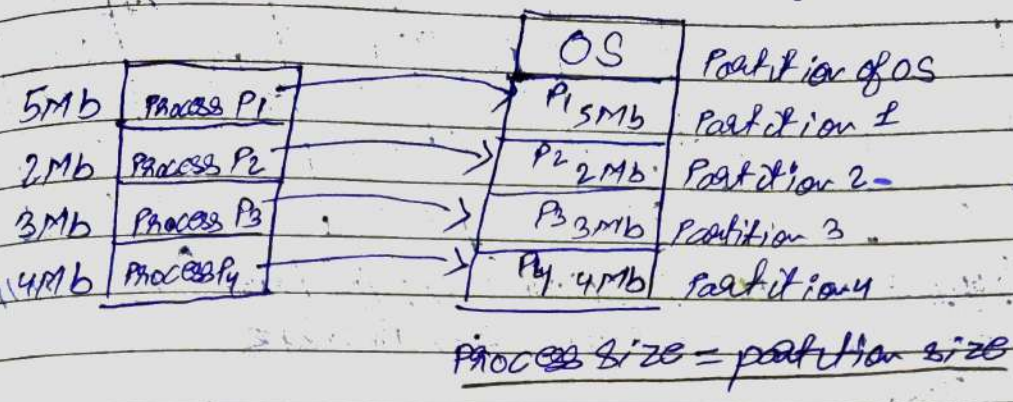
let say:
4Mb Partition

P_6 → 5Mb → will not get allocated.
partition wae 4Mb se < 4Mb process size hi allocate ho skta h.

- Limitations.
  - Internal Fragmentation
    If the size of process is lesser than the total size of partition then some size of the partition gets wasted and remain unused.
  - External Fragmentation
    The total unused space of various partition cannot be used to load the process even though there is space available but not in the contiguous form.
  - Limitation on process size.
    If the process size is larger than size of maximum sized partition then that process cannot be loaded into the memory. Therefore a limitation can be imposed on the process size that is it cannot be larger than the size of largest Partition
  - Low degree of multiprogramming.
    In fixed partitioning the degree of multiprogramming is fixed & very low bcz the size of partition cannot be varied according to the size of process.

Dynamic partitioning

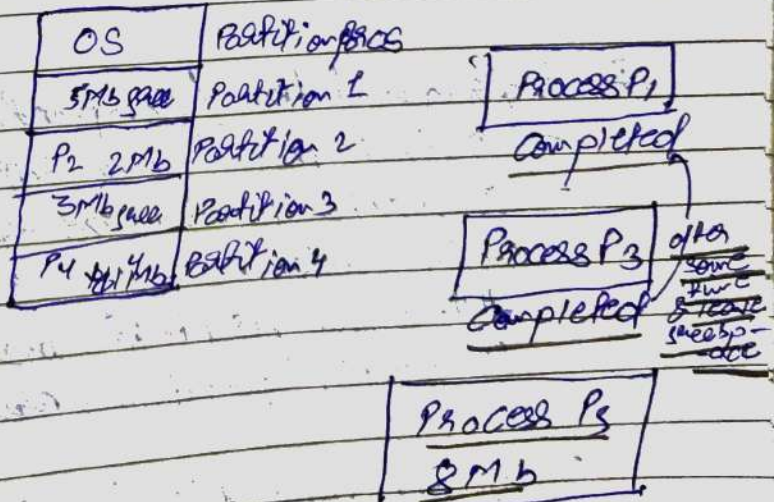
- In this technique, the partition size is not declared initially. It is declared at the time of process loading.



Process size = partition size

- Advantage
  - No internal fragmentation
  - No limit on size of process
  - Better degree of multiprogramming

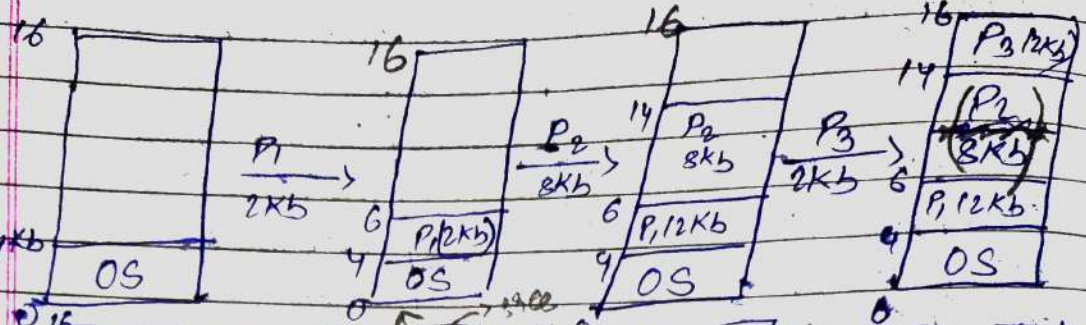
- Limitation
  - External fragmentation



P5 can't be loaded into memory even though there is 8Mb space available but not contiguous.

LEC-25 | Free space Management

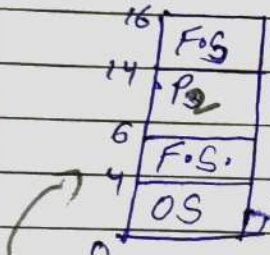
* RAM initial



Initial: FL: 12Kb → NULL FL: 8Kb → NULL FL: 12Kb → NULL FL: NULL

after some time P_1 exist then P_3 exists.

Then.



OS manages F.S by making → Free list → type of linked list

not contiguous that's why not merged

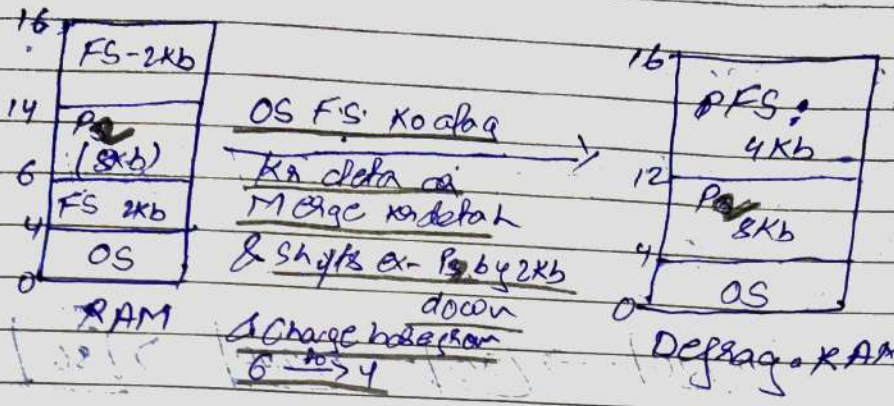
FL: 2Kb → 2Kb → NULL

Problem fragmentation here

② Defragmentation / Compaction

- Dynamic programming suffers from external fragmentation
- Compaction to minimize the probability of external fragmentation.
- All the free partitions are made contiguous, and all the loaded partitions are brought together.
- By applying this technique, we can store the bigger process in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is called defragmentation.

- The efficiency of the system is decreasing in the case of compaction since all the free spaces will be transferred from several places to a single place.



NOW

$P_4 \rightarrow 3Kb$ then FL: [4Kb] → NULL

OS CHECK then NOW P_4 can be allocated.

Limitation: Defragmentation is time consuming

(#) How to satisfy a req of a n size from a list of free holes?

- various algo which are implemented by the OS in order to find out the holes in the linked list & allocate them to the processes.

(1) First fit

- Allocate the first hole that is big enough
- Fast / less time complexity.

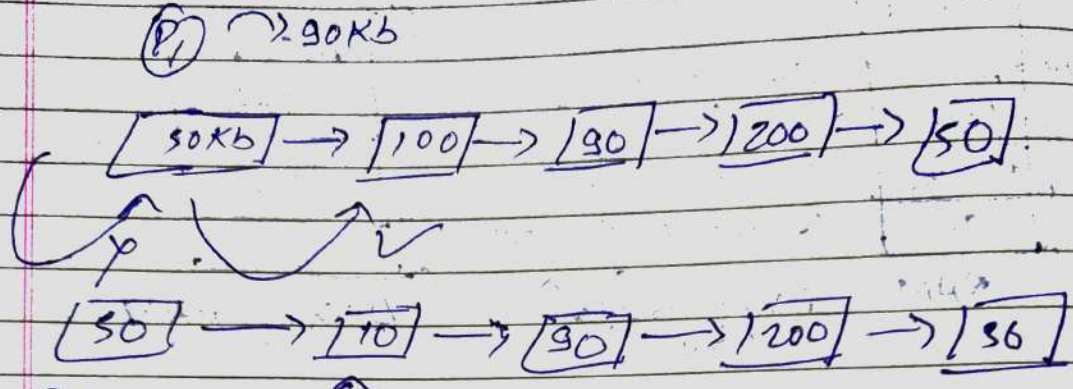
P_1 req → 90Kb

FL: [50] → [100] → [90] → [200] → [50] → NULL

FL: [ ] → [ ] → [ ] → [ ] → [ ] → NULL

② Next fit

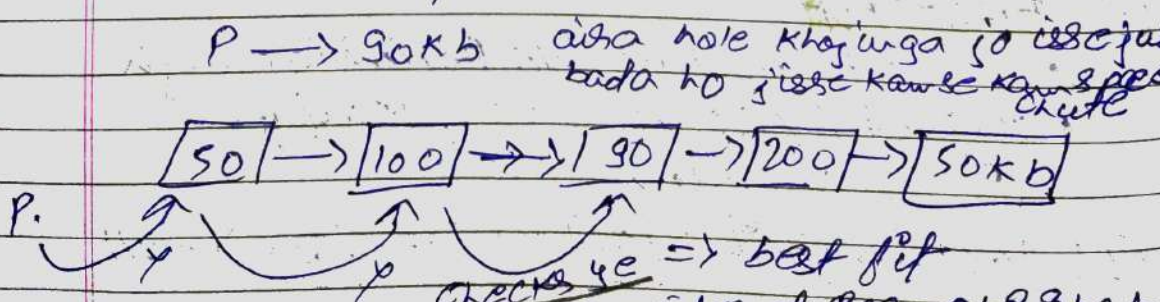
- Enhancement on first fit but starts search always from last allocated hole.
- Fast / less time complexity.



Now P2 comes & req 80Kb then first wale se iterate nhi hoga yr sidha 100-90 wale node se hoga

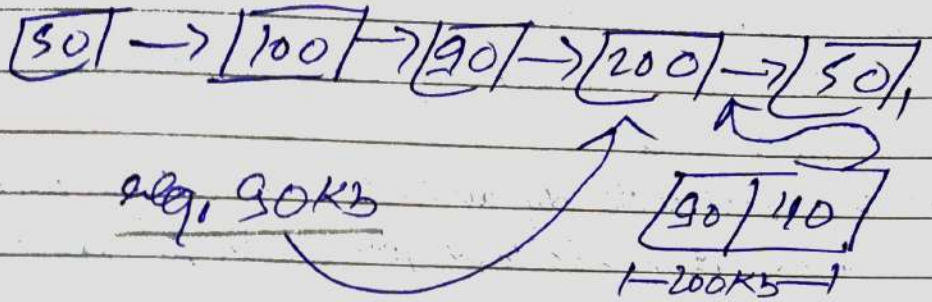
③ Best fit

- Allocate smallest hole that is big enough.
- Lesser internal fragmentation.
- May create many small holes because major external fragmentation.
- Slow, as required to iterate free holes list.



④ worst fit

- allocate the largest hole that is big enough
- Slow, as required to traverse whole free holes list.
- leave bigger holes that may accommodate other processes.



LEC-26 | Non Contiguous Memory Allocation | Paging.

Non Contiguous Memory Allocation.

* The main disadvantage of Dynamic partitioning is External fragmentation.
  - Can be removed by compaction but with overhead.
  - We need more dynamic / flexible / optimal mechanism, to load processes in partitions.

→ Idea behind paging.
- If we have only two small non contiguous free holes in the memory, say 1Kb each
- If OS wants to allocate RAM to a process of 2Kb, in contiguous allocation, it is not possible, as we must have contiguous memory space available of 2Kb (external fragmentation)
- What if we divide the process into 1Kb-1Kb blocks

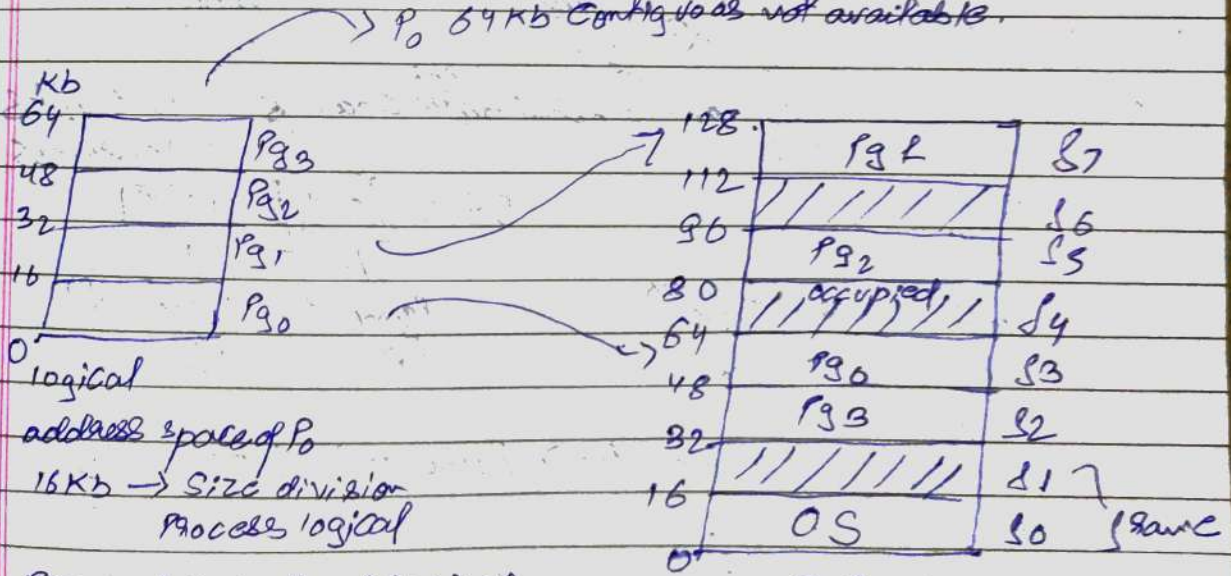
Paging

- Paging is a memory management scheme that permits the physical address space of a process to be non contiguous.
- It avoids external fragmentation & need of compaction.
- Idea is to divide physical memory into sized blocks called frames, along with divide logical memory into blocks of same size called pages (# page size = frame size)

- Page size is usually determined by the processor architecture. Traditionally, pages in a system had uniform size such as 4096 bytes, however processor designs often allow two or more, sometimes simultaneous, page size due to its benefits.

- Page Table
  - A Data structure stores which is mapped to which frame.
  - The page table contains the base address of each page in the physical memory.

- Every address generated by CPU (logical address) is divided into two parts: a page number (p) and a page offset (d). The p is used as an index into a page table to get base address the corresponding frame in physical memory.



Page → logical add. fixed size division

Page size = frame size.

This scheme is paging.

→ logical add. space ko physical add. space me convert karne ke liye MMU table Data structure ko use karta hai

* Page table

	logical pg. no.		Physical frame no.
Each process has its own Page table	00	Pg. 0	frame 3
	01	1	7 (111)
	10	3	2
	11		

- Page table is stored in main memory at the time of process creation & its base address is stored in process control block (PCB)
- A page table base register (PTBR) is present in the system that points to the current page table. Changing page table requires only this one register, at the time of context switching.

(A) Address Translation using Pg. table

logical add. space P_0 → 64 bytes

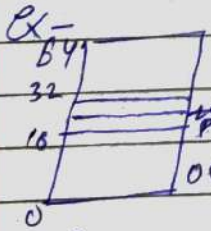
agar iss 64 bytes ko unique identify karna hoga $2^6 = 64$ (6 bits)

CK byte ko th logi. add. space me kitni bytes chahiye; i.e 6 bits

→ binary of 25

$25 = \boxed{01}\boxed{100}$

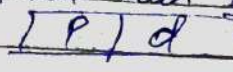
↓ Pg. no. 1 (P) ↓ offset (d) i.e 9



Pg. no. 1 → Base → 16 + 9 = 25

25 → Pg 1 → S_7 → 112 → 112 + 9 = 121

logical add. space.



branch

Physical address

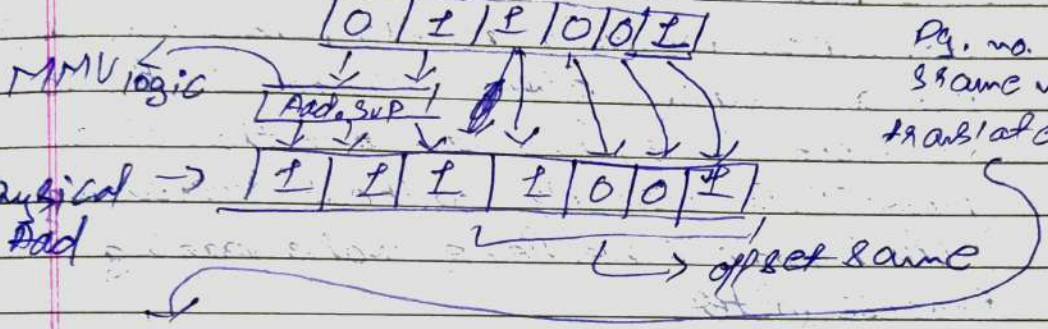
121 => 111 001

→ d (offset same hogya)

3 bit => 111 => frame 7

logical add. ⟷ Phy. add.

logical add.



Pg. no. ko kaise frame no. me translate karu

ye frame ke liye Page table ka use kiya i.e 01 ke corresponding frame kaun sa 111

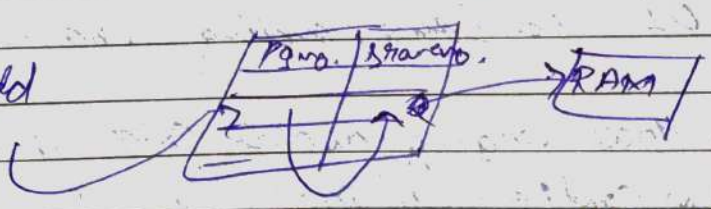
Q. How paging avoids external fragmentation?

=> Non contiguous allocation of the pages of the process is allowed in the random free frame of the physical memory.

Q. Why paging is slow & how do we make it fast?

=> There are too many memory references to access the desired location in Physical memory.

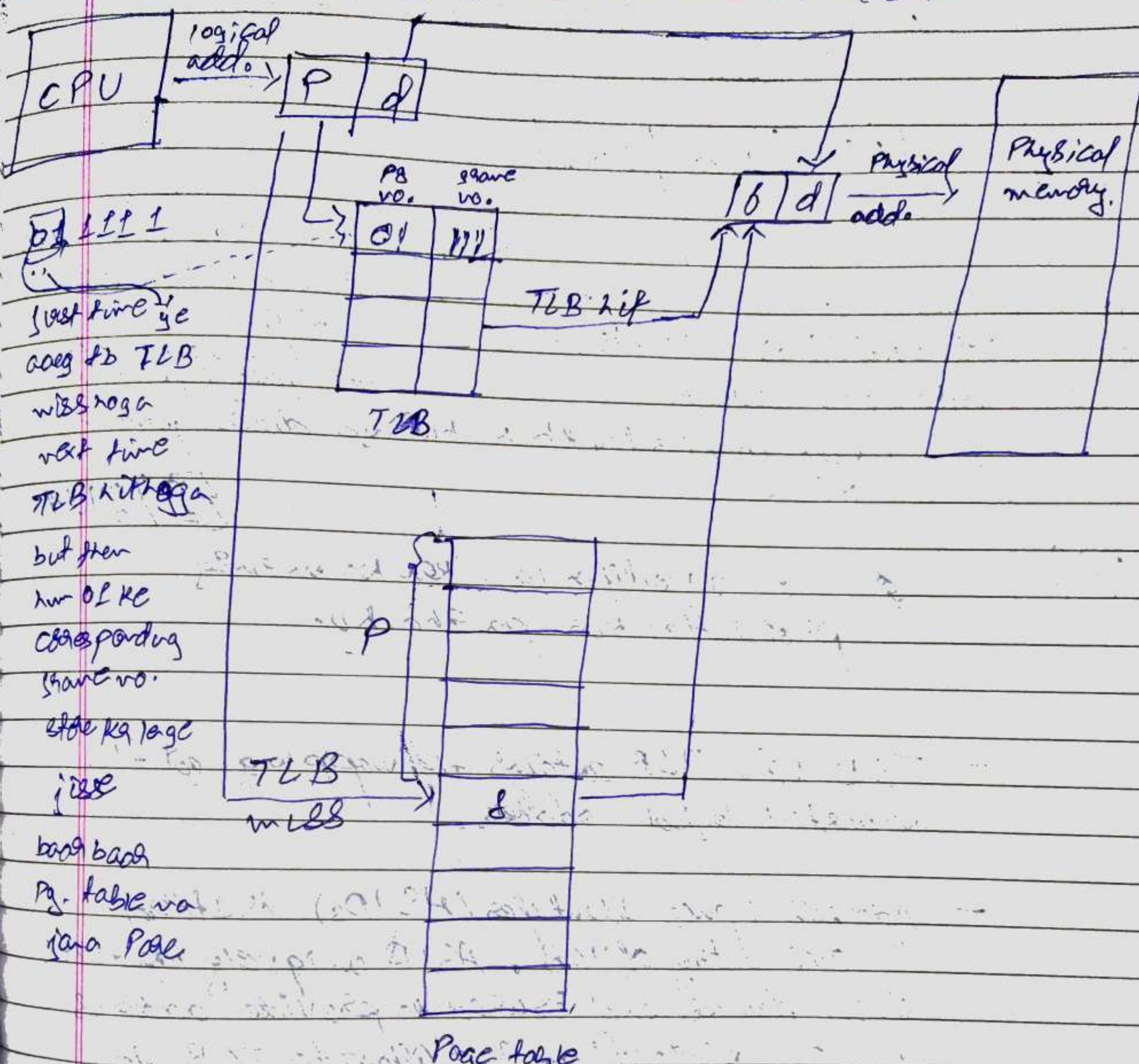
* Translation Look aside buffer (TLB)
- A hardware support to speed up paging process
- It's a hardware cache, high speed memory.
- TBL has key & value
- Page table is stored in main memory & b/c of this when the memory reference is made the translation is slow.
- When we are retrieving physical address using page table, after getting frame address corresponding to the page no., we put an entry of the into the TLB. So that next time, we can get the values from TLB directly without referencing actual page table. Hence, make paging process faster.



Iss overhead ko bachne ke liye ek hardware support ka use kiya h

TLB access time memory se bhi kam hota h

Paging hardware with TLB



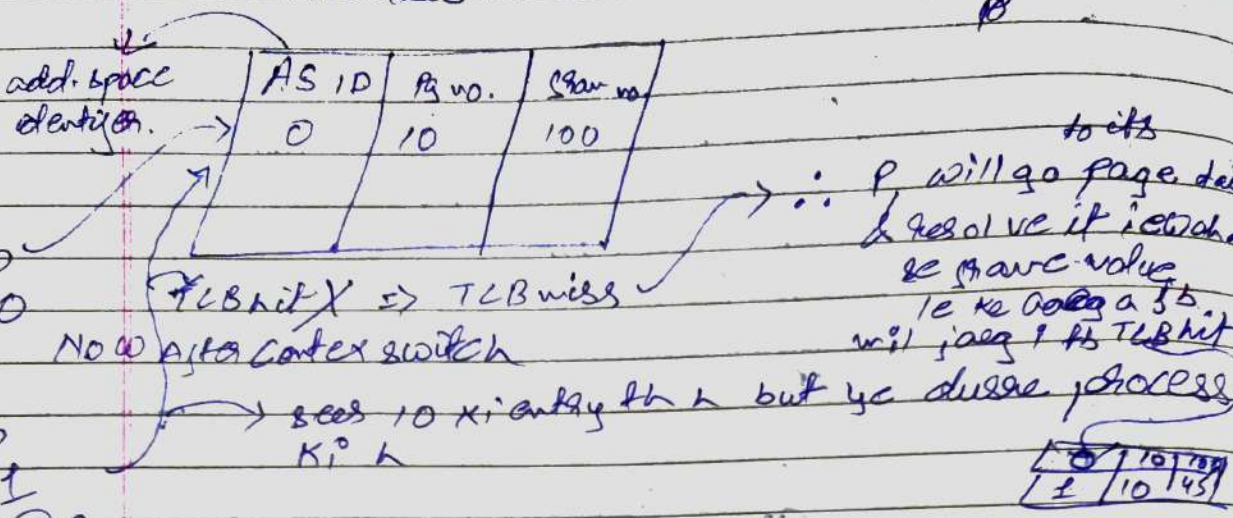
Jb bhi context switch hoga P_0 se P_1 aaega tb page table or TLB bhi change karni hogi dono mae info the same hi h (

i.e TLB mae jo entries h unko delete karna hogi

~~① i.e TLB reset~~

① TLB flush (reset) → is a cost kaam

② unique identifiers that will identify unique processes



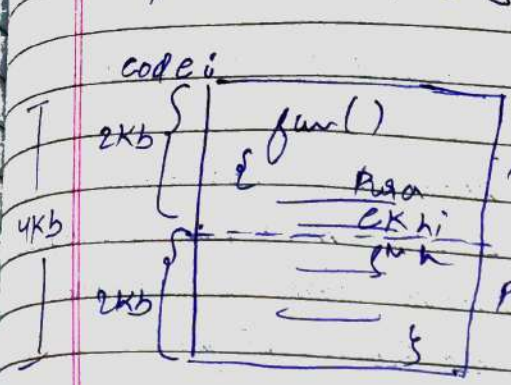
③ Yc IDentifier use kra ke memory protection kar pa rha hu.

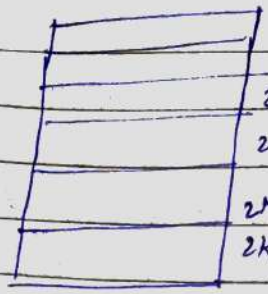
- TLB hit, TLB contains the mapping for the requested logical address.

- Address space Identifiers (ASIDs) is stored in each entry of TLB. ASID uniquely identifies each process and is used to provide address space protection and allow to TLB to contain entries for several different processes. When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match the attempt is treated as TLB miss

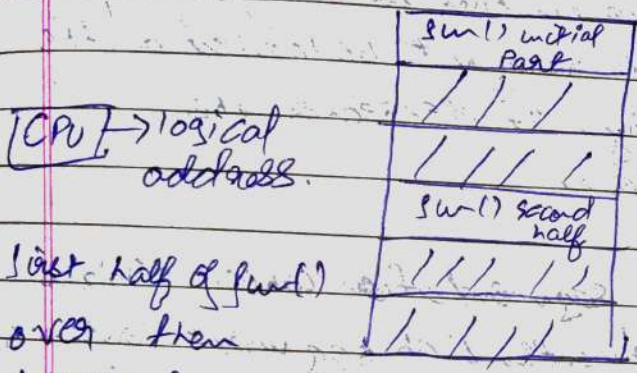
LEC-27 | Segmentation | Non-contiguous memory allocation

* problem in paging





user program logical address space.



MMU dusre frame pe shift hogaega
same frame pe chl rhe hanga th INEFFICIENT kaam kata h

This problem is solved by technique segmentation.

* Segmentation is memory management technique that supports the user view of memory.

* A logical address space is a "coll" of segments these segments are based on user view of logical memory.

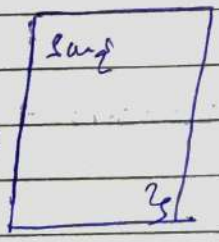
* Each segment has segment no. & offset, degrading segment.
<segment-number, offset> {S, d}

* Process is divided into variable segments based on user view.

* Paging is closer to OS rather than the user. It divides all the process into the form of pages although a process can have some relative parts of function which need to be loaded on the same pages.

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

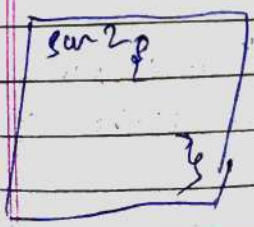
* It is better to have segmentation which divides the process into the segments. Each segment contains the same type of function such as the main function can be included in one segment and the library functions can be included in the other segment.



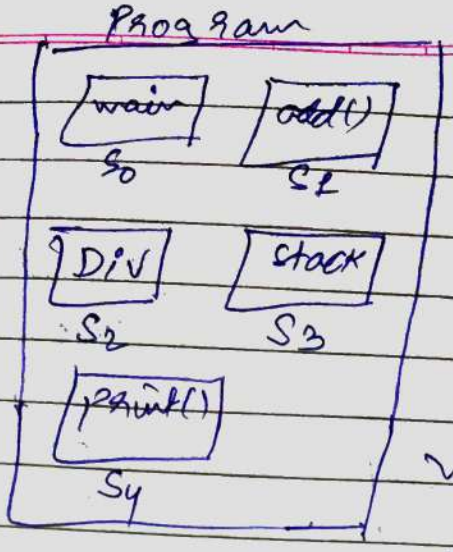
2 pg mae va divide ka ke segment mae divide karaeg.
→ diff segment
↳ of varying size

- user view
user view ke hisab se, OS hme function ko ek segment mae divide karaeg aur usko logical address ke part th contiguous ek ko ek part mae have karaeg

-> Variable partitioning of logical add. space



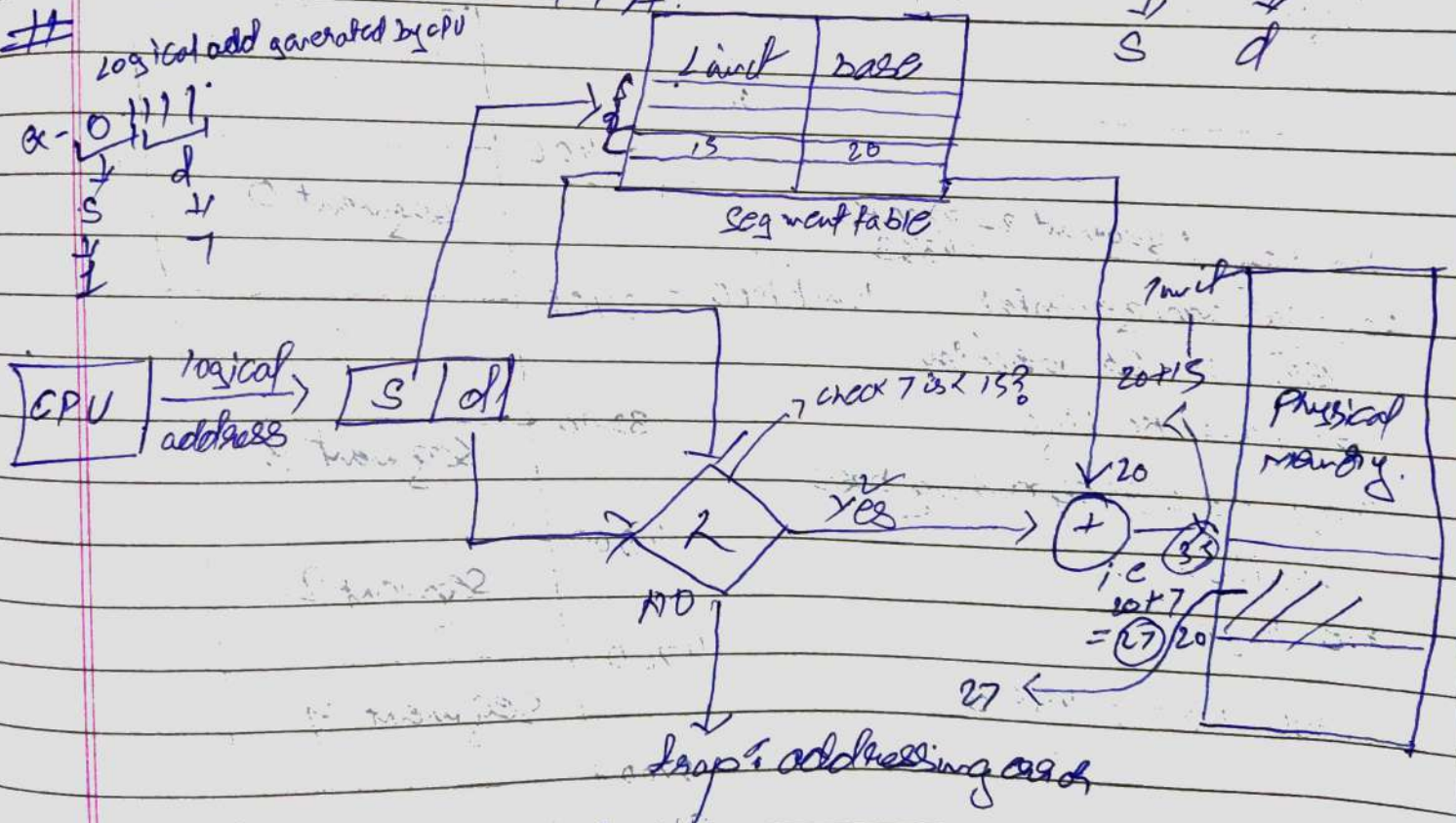
fun or fun2 diff th done ko diff segment mae deal kiya.



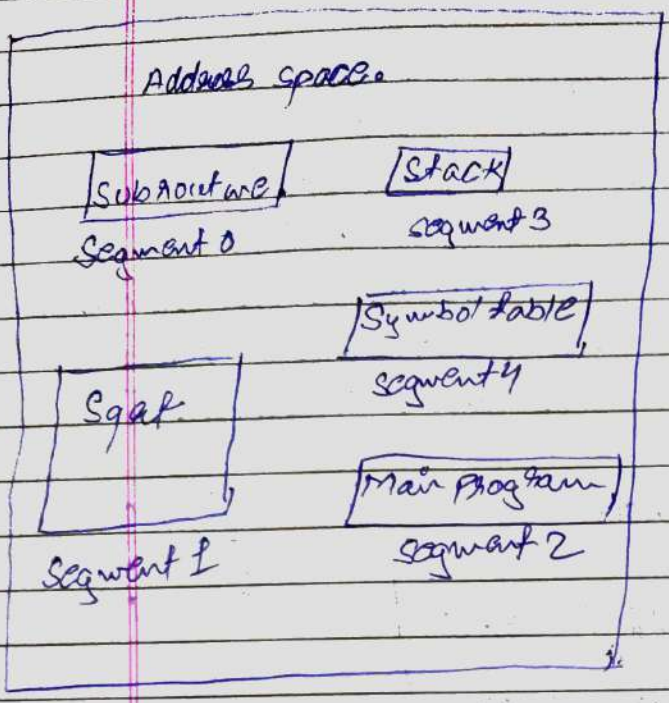
Segment size
diff
segment compiler
decide karta h ki user
view me kisko segment
me aana chahiye.

* MMU → Translation.
LA → PA.

LA → | seg no. | offset |
↓ S ↓ d



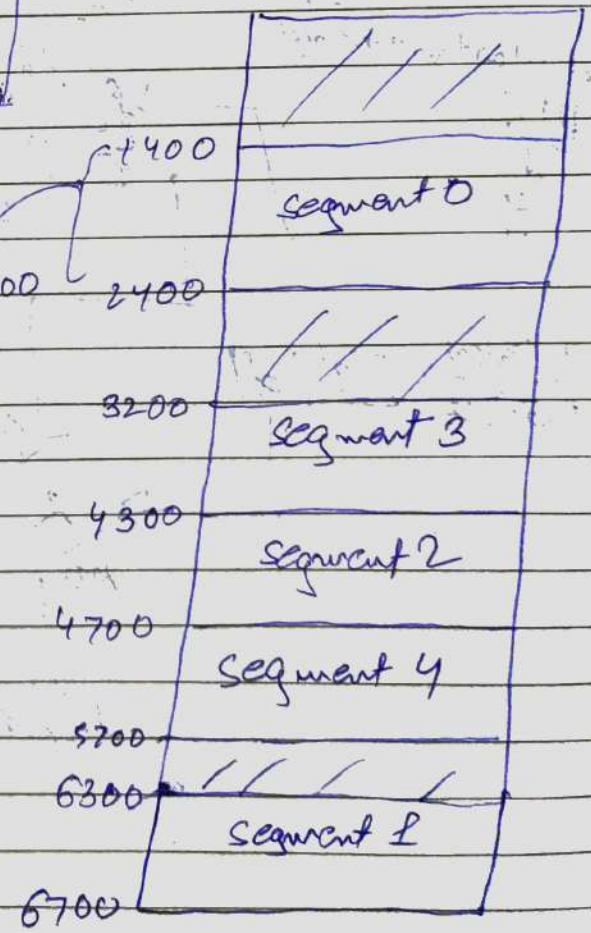
Example:



Segment table

Limit	Base
1000	1400
400	6300
400	4300
1100	3200
1000	4700

Physical Memory.



Let say: segment 2 = 4300 + 53
= 4353

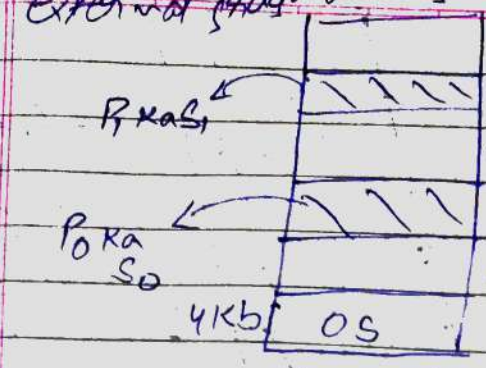
Logical address generated (53) in 53 position
page ke liye
seg 2 ke corresponding base
address
i.e 4300

4300 + 53 = 4353

limit 1000

If has external fragmentation in segmentation.

Ex- External frag in segmentation



Kaun se segment mein pura seg. allocate hoga => No external frag.

$P_1 \rightarrow S_0 \rightarrow 2Kb$

$S_1 \rightarrow 3Kb.$

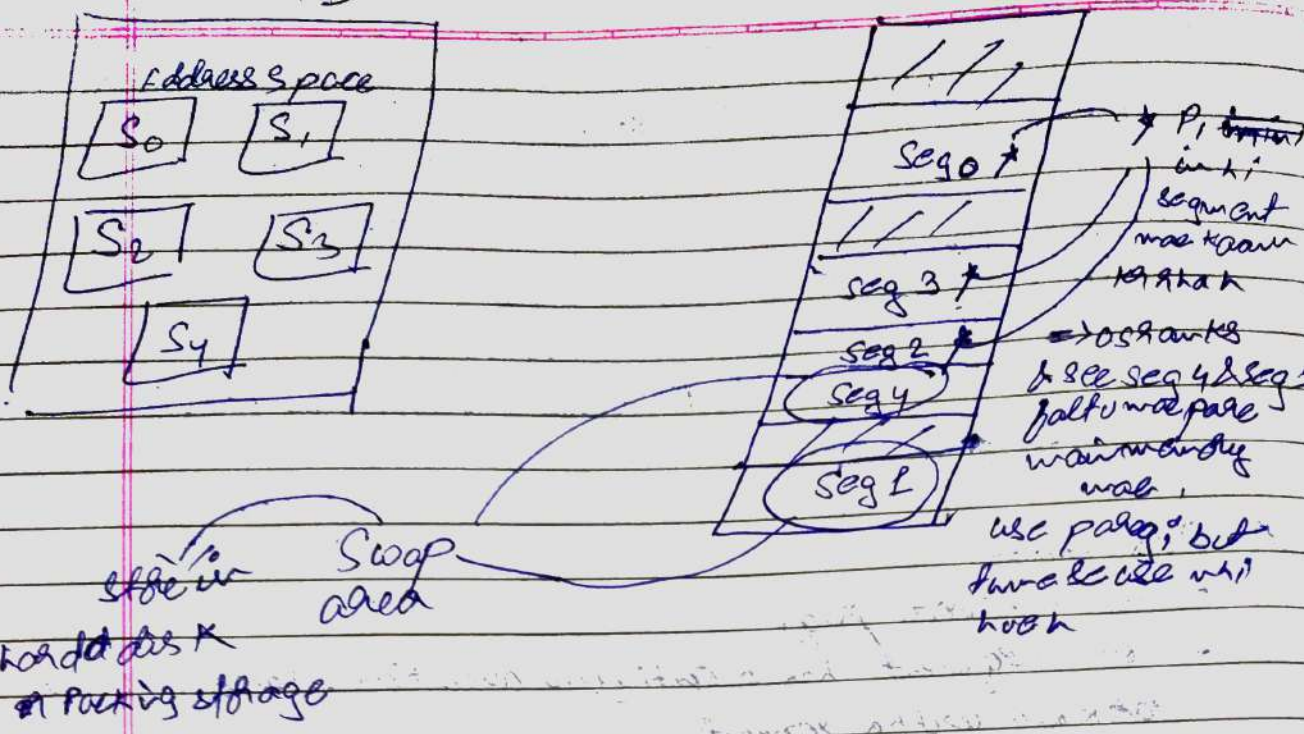
3Kb contiguous chahiye

=> Ith yaha external frag aaya.

* Advantages.
  - No external frag.
  - one segment has a contiguous allocation, hence efficient working within segment.
  - The size of segment table is generally less than the the size of pg. table
  - It results in a more efficient system b/c the Compiler keeps the same type of function in one segment.

* Disadvantage
  - External fragmentation.
  - The diff size of segment is not good that the time swapping.

* Modern system architecture provides both Segmentation & paging implemented in some hybrid approach



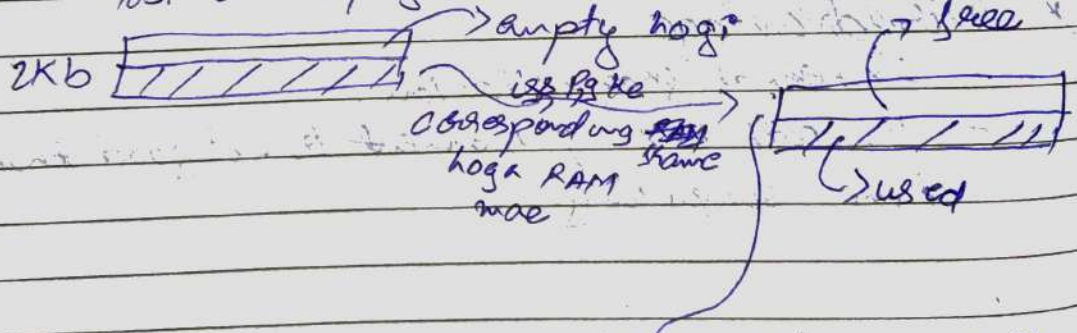
* Internal frag in Paging.

Program → 15 Kb

Pg size → 2 Kb

16 Kb → divide hojata 2 Kb - 2 Kb wale 2

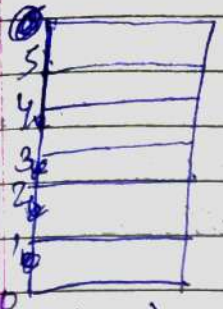
* last wala page



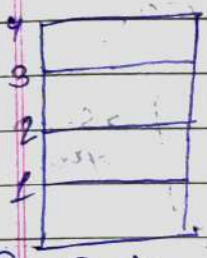
internal fragmentation that can not be avoided.

LEC-28 | Virtual Memory.

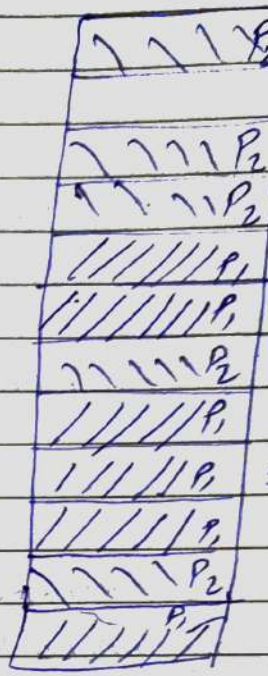
* Virtual memory is a technique that allows the execution of processes that are not by treating a part of secondary memory as the main memory. (swap-space).



P1 (6Kb)

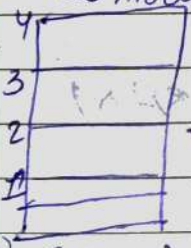


P2 (5Kb)



RAM (12Kb)

Now new Process comes after allocation of P1 & P2



OS ek free frame ko dedega but baki 4 frames ka kya i.e P3 ko execute ni vhi ho paega

P3 (15Kb)

↳ need 5 frames

But modern PC mae aisa nhi hota

Ex - 16 RAM → OS
↳ Programs
↳ GTA 5 → heavy 30-40Gb game

Ab ye game 16Gb RAM mae kaise load hoga?

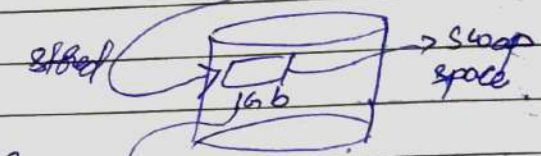
=> Virtual memory concept use kiya.

=> Kuch aisa procedure karo jisse processes ko ready queue state mein aane ke process ke kuch hi pages ko allocate karo → aur pages jinki need h.

Example — P_1 → $P_{g.}$ 0, 1, 2 are needed
&
P_2 → P_{g} 0 is needed
||
Total 5 frames allocated in RAM.

Now when P_3 comes → P_g 1, 4 are needed
=> Total 7 frames have been allocated out of 12 frames.

Baki jo process ki bachi (left) pages h wo →



Jb bhi koi page need hoga swap space hoga OS jidhi se swap in karega

=> 19 Kb total prog. given 12 Kb RAM

RAM + SWAP Area → Virtual

* Advantage of this is, program can be larger than physical memory.

* It is required that instructions must be in Physical memory to be executed. But it limits the size of program to the size of physical memory. In fact, in many cases, the entire program is not needed at the same time. So, we want

an ability to execute a program that is only partially in memory would give many benefits.

(a) A program would no longer be constrained by the amount of physical memory that is available.

(b) b/c each user program could take less physical memory, more programs could be run at same time with a corresponding increase in CPU utilization and throughput

(c) Running a program that is not entirely in memory would benefit both the system and the user.

* Programmer is provided very large virtual memory when only a smaller physical memory is available.

Ye pure scheme apply karne ka popular method h demand paging.

* Demand paging.

— In demand paging, the pages of process which are least used, get stored in the secondary memory.

— A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced.

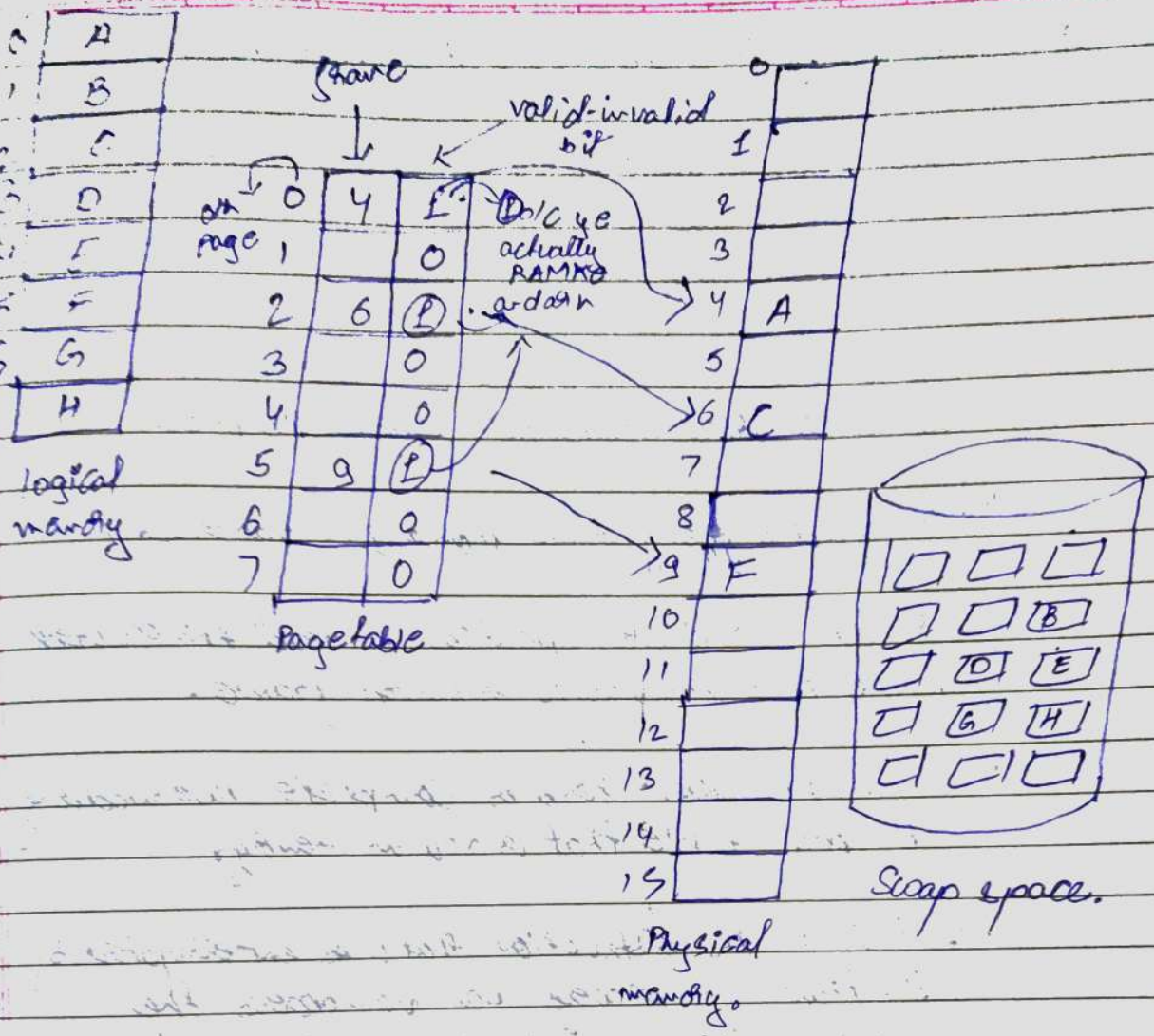
- Rather than swapping the entire process into memory, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.

- We are viewing a process as a sequence of pages rather than one large contiguous address space, using ~~concerned with individual pages~~ of a pages of a process. The term swapper is technically incorrect. A swapper manipulates entire process but pager is concerned with individual pages.

* How Demand paging work?

(a) When a process is to be swapped-in, the pager guesses which pages will be used.
(b) Instead of swapping in a whole process, the pager brings only those pages into memory. This, it avoids reading into memory pages that will not be used anyway.
(c) Above way, OS decreases the swap time & the amount of physical memory needed.
(d) The valid - invalid bit scheme in the page table is used to distinguish b/w pages that are in memory and that are on the disk.

- valid-invalid bit 1 means, the associated page is both legal and in memory.
- valid-invalid bit 0 means, the page either is not valid (not in the LAS of the process) or is valid but is currently on the disk.



① If a process never attempts to access some invalid bit page, the process will be executed successfully without ever the need page present in Swap space

② what happens if the process tries to access a page that was not brought into memory, access to a page marked invalid causes page fault. Paging hardware noticing invalid bit for a demanded page will cause a trap to the OS.

(h) (e) The procedure to handle the page fault:

I. Check an internal table (in PCB of the process) to determine whether the reference was valid or an invalid memory access.

II. If ref. was invalid process throws exception
If ref is valid, pages will swap in the page.

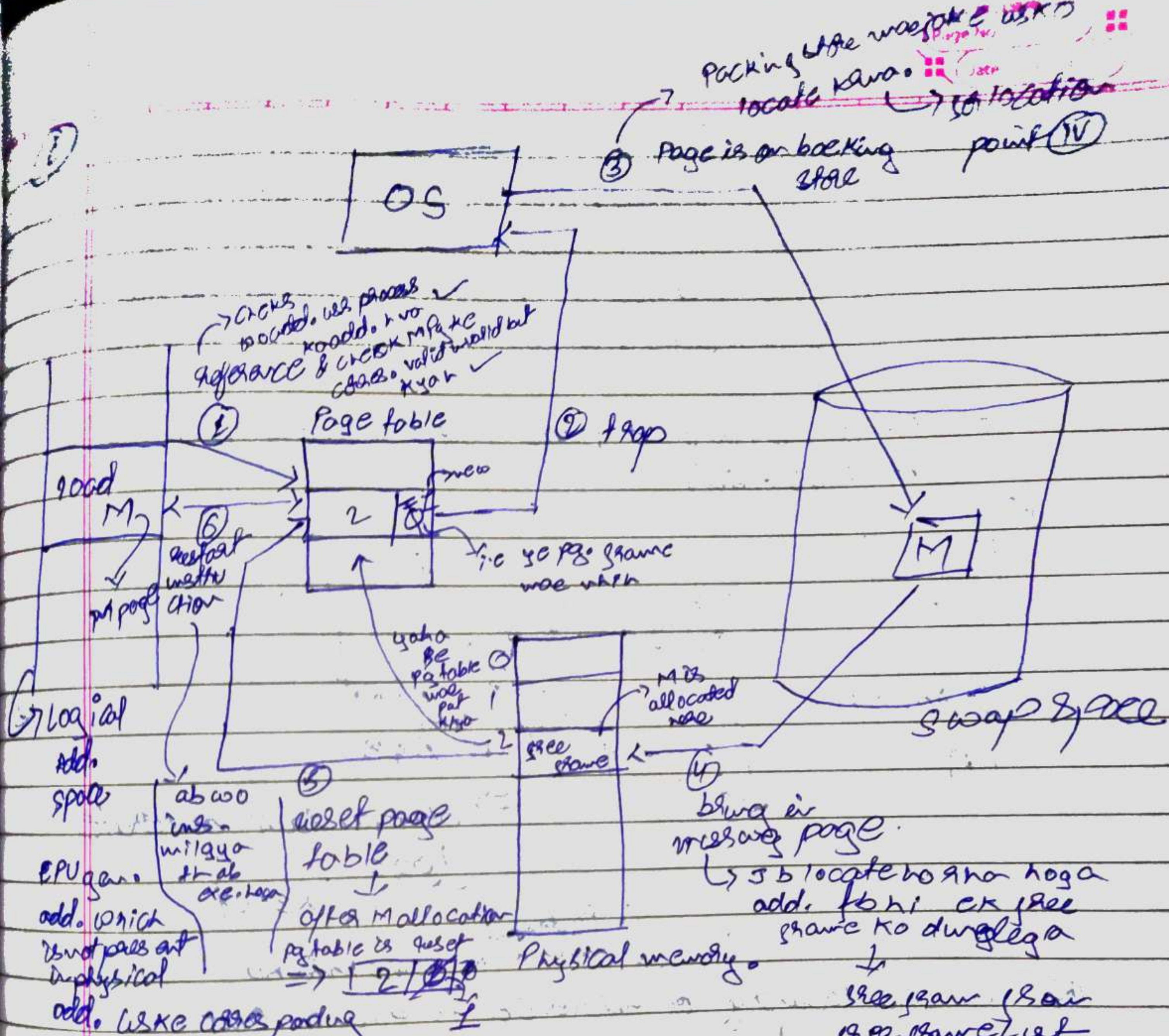
III. We find a free frame (from free frame list)

IV. Schedule a disk operation to read the desired page into the newly allocated frame.

V. When the disk read is complete we modify the page table that is now in memory.

VI. Restart the instruction that was interrupted by the traps. The process can now access the page as though it had always been in memory.

(I)



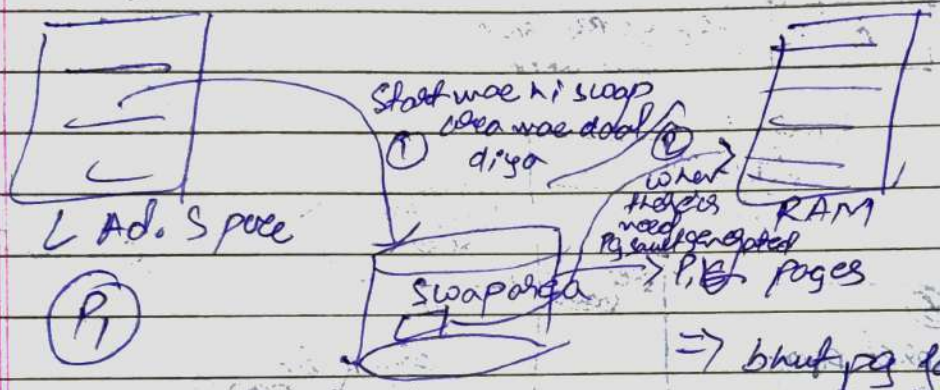
CPU gen. add. which is not present in physical add. uske corresponding swap area mae padaa

Ex - address of M page

(5) Pure Demand Paging

- In extreme case, we can start executing a process with no pages in memory. When OS set the instruction pointer to the first instruction of the process, which is not in the memory,

- Never bring a page into memory until it is required.



⇒ bhut pg fault generate honga
⇒ overhead

∴ Pg demand use wala kaafi kam use karta hai

(6) We use locality of reference to bring out reasonable performance from demand paging.

Matlab wo wale pages jo ki pages ko lga rha h ki need hogi jinki

Page No.
Date : / /

* Advantages of virtual memory.
  - The degree of multi programming will be increased
  - User can run large apps with less real Physical memory.

* Disadvantages of virtual memory
  - The system can become slower as swapping takes time
  - Thrashing may occur

LEC-29 | Page Replacement Algorithm

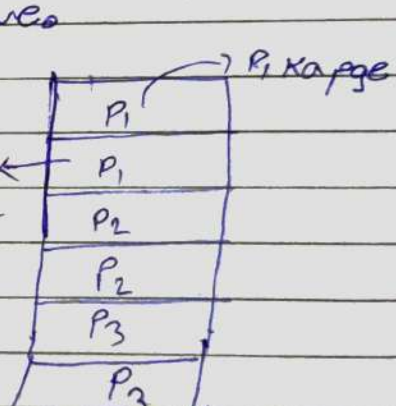
- Whenever page fault occurs, that is, a process tries to access a page which is not currently present in a frame & OS must bring the page from swap space to a frame.

- OS must do page replacement to accomodate new page into a free frame, but there might be a possibility the system is working in high utilization & all the frames are busy, in that case OS must replace one of the pages allocated into some frame with new page.

- The page replacement algorithm decides which memory page is to be replaced. Some allocated page is swapped out from the frame & new page is swapped into the freed frame.

Swap area

when code run
this page was instruction
P1 process ki
Page 3 pe jana
h



Ab mujhe P1 ke corresponding
Page3 ko swap in karna h

RAM

Ab koi frame khali ho ya
Jo iss ex mae nhi h
=> Now, ek aisa frame search krna hoga
jise delocate krke

Ab
OS ⟶ generate page fault

Ye swap in or swap out karne mae time lagta

page fault service time often

(#) Types of page Replacement Algorithm
(Aim to have minimum page faults)

(a) FIFO.

i. Allocate frame to the page as it comes into the memory by replacing the oldest page.

ii. Performance is not always good.
- The page replaced may be an initialization module that was used long time ago (good replacement candidate.
- The page may contain a heavily used variable that was initialized early and is in constant use (will again cause page fault).

Initially 3 frames

Page 7 aaya phir '0' phir '1' ... now 2 comes & replace oldest page (1)

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	4	4	4	0				0	0			7	7	7
	0	0	0	3	3	3	2	2	2				1	1			1	0	0
		1	1	1	0	0	0	3	3				3	2			2	2	1
1	2	3	4	5	6	7	8	9	10				11	12			13	14	15

(1) 7 page fault frame initially empty.

7 aaya tb 1st first page fault

FIFO → Is page fault jit vi jada page fault utni inefficient algorithm

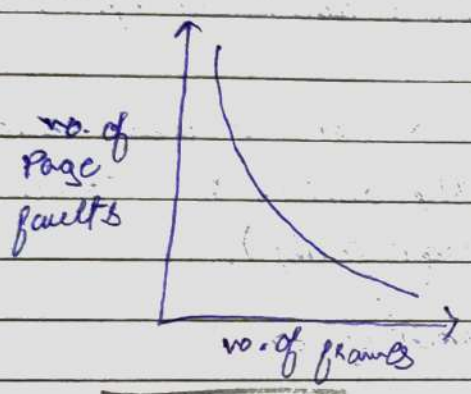
Easy.

iii. Belady's anomaly is present.
- In the case of LRU & optimal page replacement algorithm, it is seen that no. of page fault will be reduced, if we increase number of

frames. However, Belady found that, In FIFO page replacement algo, the no. of page faults will get increased with the increment in no. of frames.

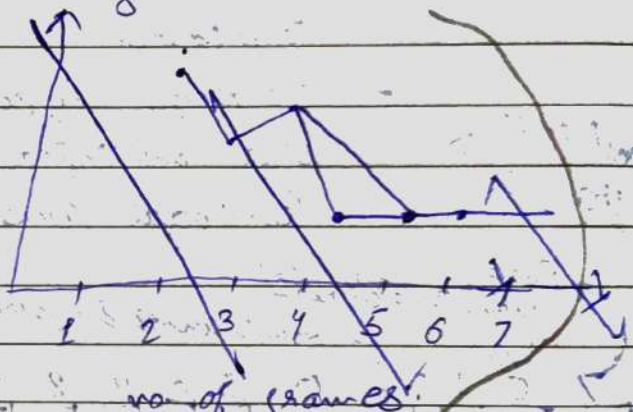
— This is strange behaviour shown by FIFO algo in some of the cases.

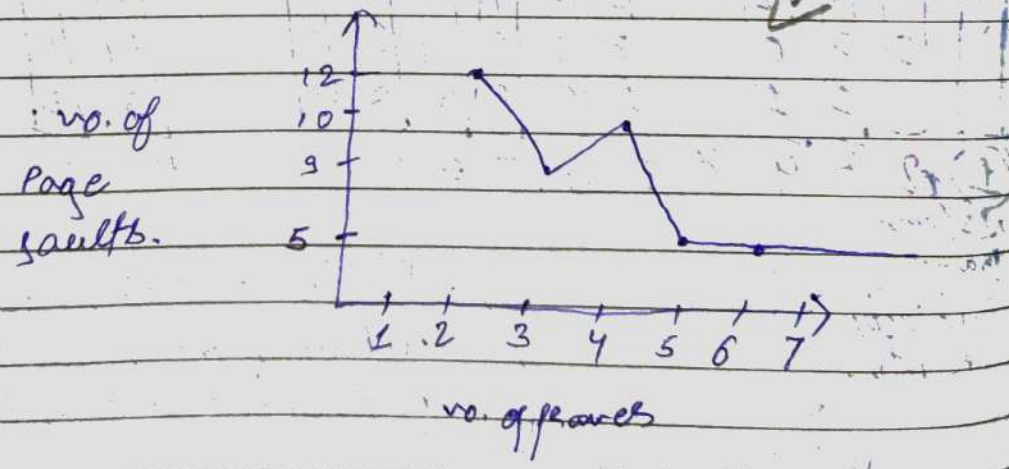
Normally:



but Belady's anomaly in FIFO

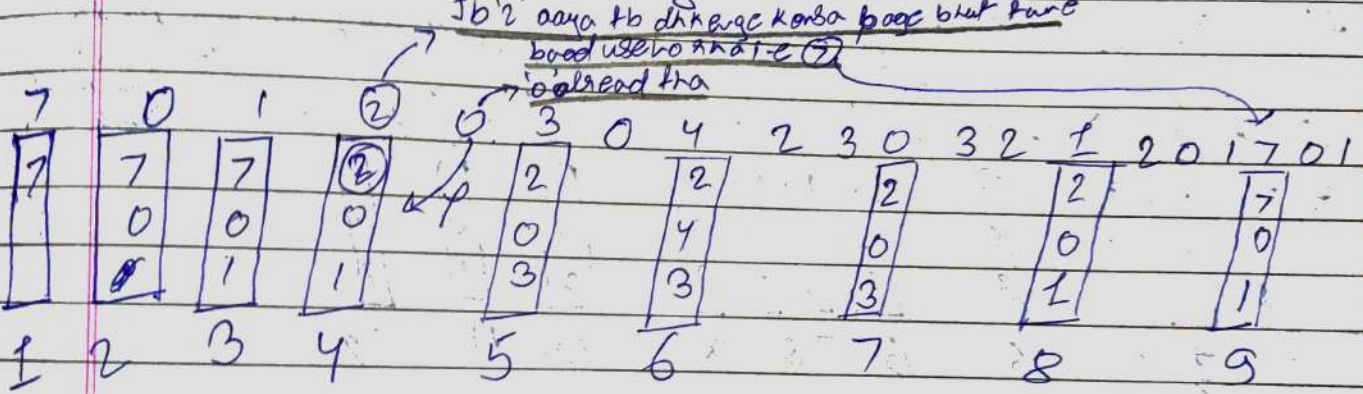
Ref: 1 2 3 4 1 2 5 1 2 3 4 5





(b) Optimal page replacement

- Find if a page is never referenced in future. If such a page exists, replace this page with new page. If no such page exists, find a page that is referenced farthest in future. Replace this page with new page.
- Lowest page fault rate among any algorithm
- Difficult to implement as OS requires future knowledge of reference string to which is kind of impossible (similar to SJF scheduling).



OPR → 9 page faults.

(c) Least-recently used (LRU)

- We can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.
- Can be implemented by two ways.
  1. Counters
     - Associate time field with each page table entry.
     - Replace the page with smallest time value

(ii) **Stack**

- Keep a stack of page number.
- Whenever page is referenced, it is removed from the stack & put on the top.
- By this, most recently used is always on the top & least recently used is always on the bottom.
- As entries might be removed from the middle of the stack, so doubly linked list can be used.

Jb '2' aaya tb ye dkhna ki konsa page sabse oldest h i.e jo bhut time se pdi thi nhi ngi / jt aage use i.e (1) oldest

Jb '5' aaya tb 2,0,1 me (1) oldest h b/c recently th referhua h jisse bagal wala

7 0 1 (2) 0 (3) 0 4 2 3 0 3 2 1 2 0 1 7 0 1

1	2	3	4	5	6	7	8	9	10	11	12
7	7	7	2	2	4	4	4	0	1	1	1
	0	0	0	0	0	0	3	3	3	0	0
	0	1	1	3	3	2	2	2	2	2	7

LRU → 12 page fault

* **LRU implement** (counting)

Jb '2' aaya tb dkhna h sbse smallest count kiska h i.e (7) → (1), count

7 0 1 (2) 0 3 0 4 2 3

7	7	7	2	2
	0	0	0	0
		1	1	3

Page count → universal hota (hoga)

0 → 2 2nd / 5 → when 0 comes after 2nd

1 → 3

2 → 4

3 →

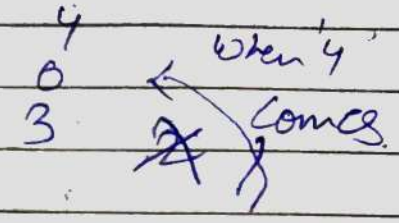
4 →

7 → 1 → 1st

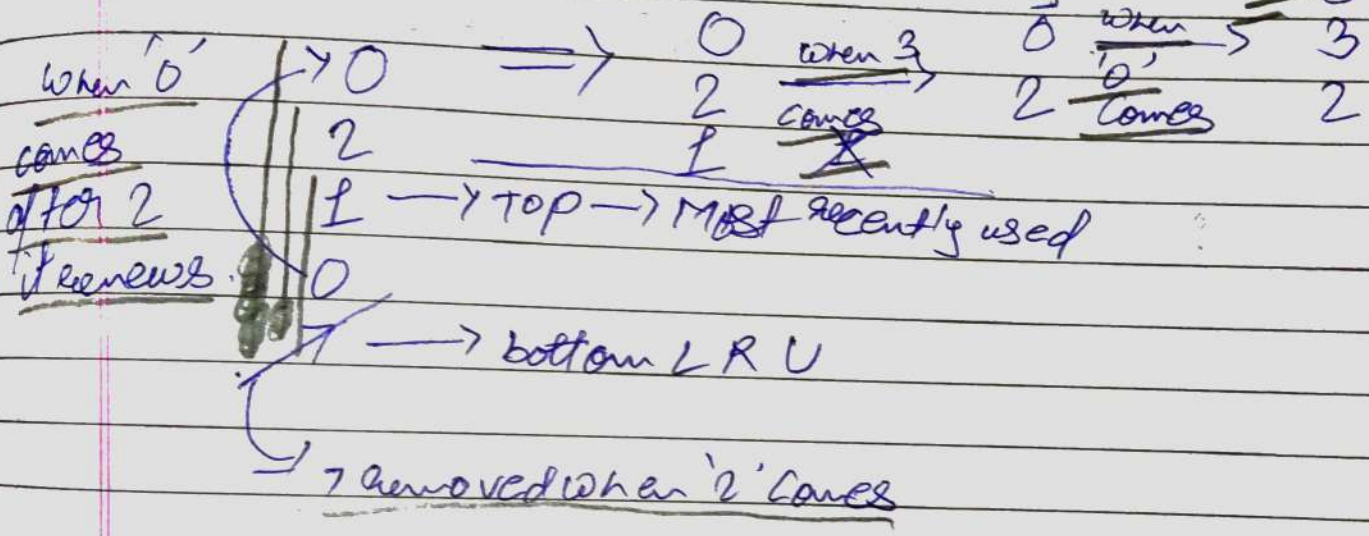
By Global shared count variable LRU can be implemented.

→ Stack based

7 0 1 2 0 3 0 4 2 3 . . .



Stack:



(d) Counting based Page replacement

Keep a counter of the no. of references that have been made to each page.

① Least frequently used (LFU)

- Actively used pages should have a large reference count.
- Replace page with the smallest count.

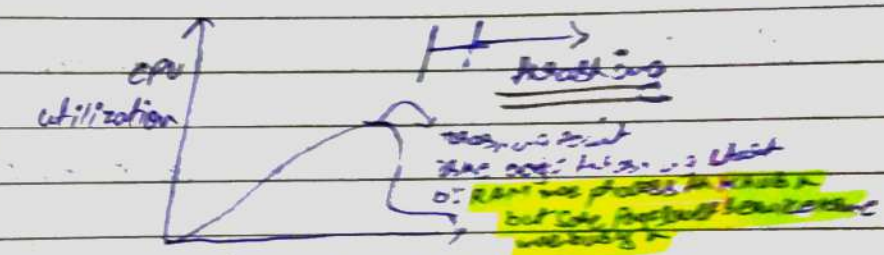
② Most frequently used (MFU)

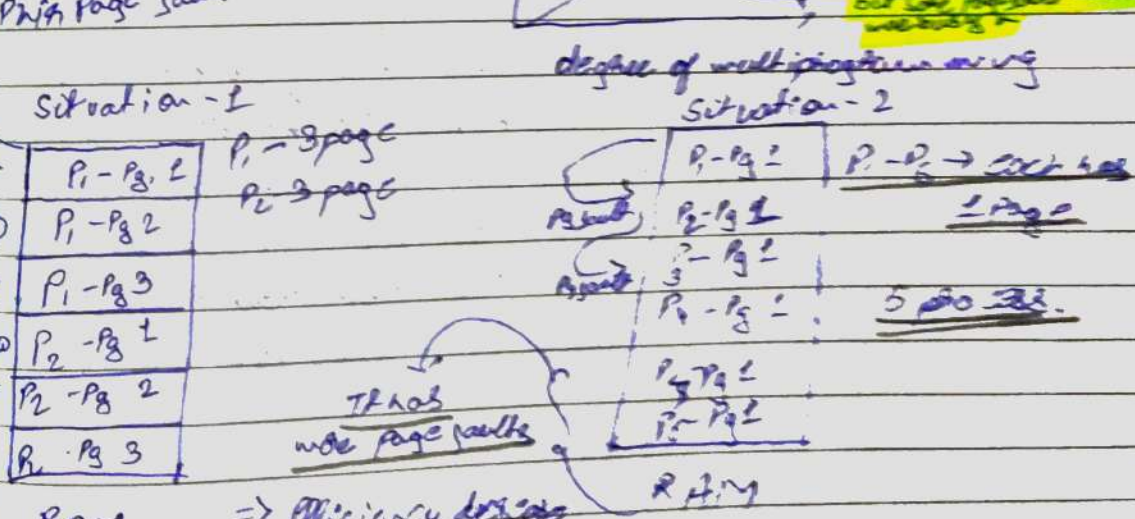
- Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

LEC-30 Thrashing

- If the process does not have the no. 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, replacing pages that must be brought back in immediately.

- This high paging activity is called Thrashing.
- A system is thrashing when it spends more time servicing the page faults than executing processes.





Thrashing more page faults

=> Efficiency decrease

CPU will be more busy servicing page faults

→ Cause of Thrashing.
- Initial low CPU utilization ... ↑ing degree of multiprogramming.
- A global pg replacement algo, replaces pages w/o regard to the process
- A process may need more frames .. cause faults again
- other processes frames are replaced & they may need those soon.
- As a result CPU utilization ↓es
- CPU scheduler now, may increase CPU utilization by increasing degree of multiprogramming.
- ultimately, CPU utilization drops drastically.

(2) Technique to handle Thrashing.

(i) Working set model.
- This model is based on the concept of the locality model.
- This basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it move to some new locality. But if the allocated frames are lesser than the size of current locality the process is bound to thrash.

(ii) Page fault frequency.
- Thrashing has a high page fault rate.
- We want to control the page fault rate.
- When it is too high, the process needs more frames. Conversely, if the page fault rate is too low, then the process may have too many frames.

- We establish upper and lower bounds on the desired page frame rate.
- If P.f. rate exceeds upper limit allocate the process another frame, if P.f. rate falls below the lower limit, remove a frame from the process
- By controlling Pf rate, thrashing can be prevented.