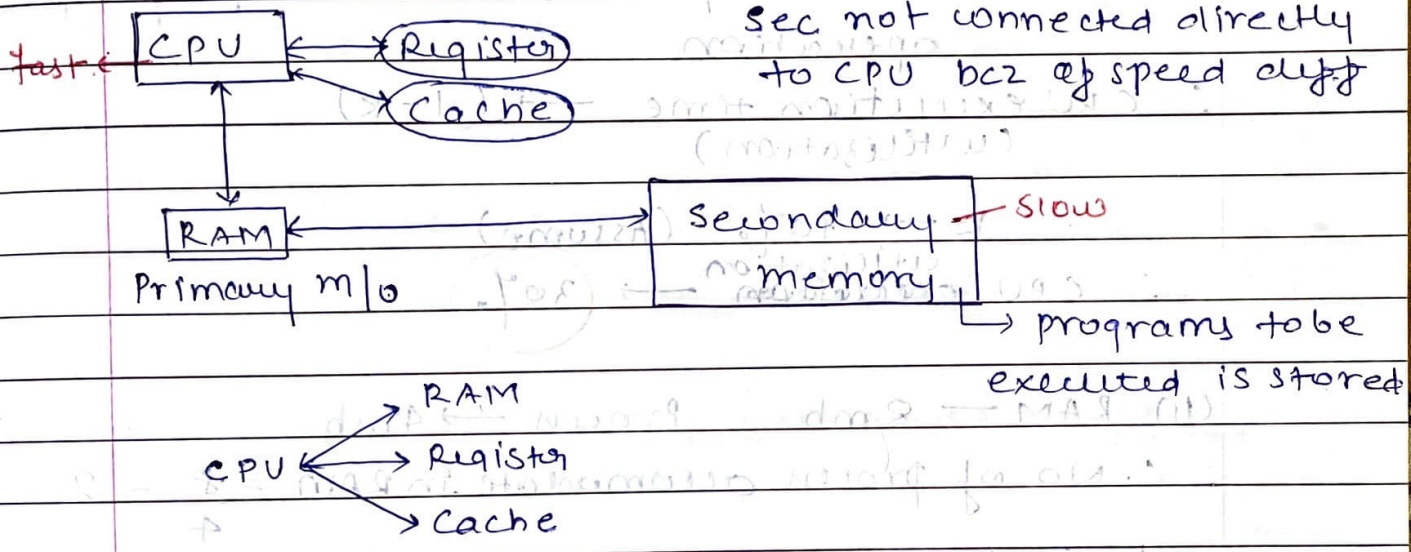
Operating System & System Programming

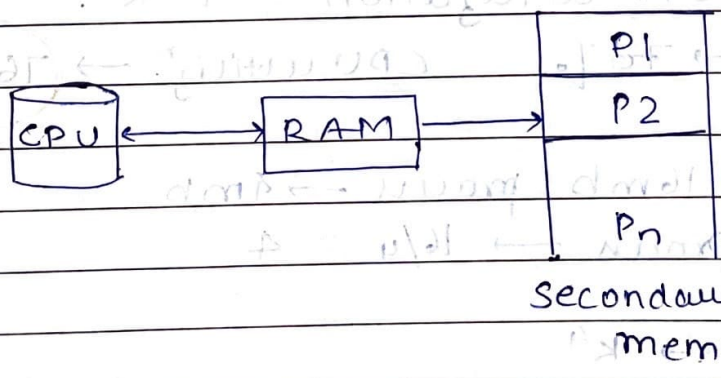
↳ method of managing primary m/o.

- memory management and degree of multi programming →

functionality to manage the various kinds of memories. in efficient manner. (RAM, Hard disks, Registers).



→ Program/process is first brought in primary m/o and then CPU directly interacts with the process. (Reg/cache mein bhi la sakte hai but size kam h)



Degree of multiprogramming

maximum ~~of of~~ no. of processes that a single processor system can accomodate efficiently

CPU utilizatn → $1-k^n$ n → no. of process

→ more the degree of multiprogramming; more the CPU utilization. (Hence system efficiency and performance increases)

(Ex) (i) RAM 4mb Process 4mb

∴ 1 process can come in RAM.

P1 (RAM) let say there is probability 'k' that process will do input/output operation.

∴ CPU execution time → (1-k) (utilization)

k → 70% (Assume)

∴ CPU utilization → 30%.

(ii) RAM — 8mb Process → 4mb

∴ No. of process accomodate in RAM = 8/4 = 2

P1 P2 (RAM) I/O operation → k → for 1, k^2 → for 2

∴ CPU utilization = $1-k^2$.

k → 70% CPU utiliz^n → 76% (approx)

(iii) RAM — 16mb process → 4mb

∴ No. of process → 16/4 = 4

I/O op^n → k^4

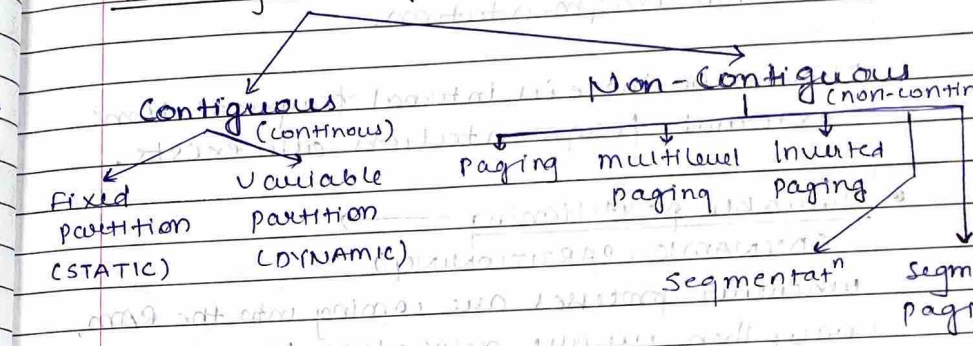
∴ CPU execution = $1-k^4$ = 93%

→ Zyada process ko ready state mein rakhna hai.

→ Degree of multiprogramming increases if
(i) No. of process in RAM increase
(ii) RAM size increase.

∴ RAM memory management is very important.

- memory management techniques →



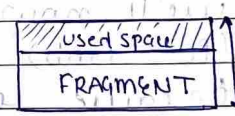
- fixed Size Partitioning (Internal fragmentation)
(STATIC PARTITIONING)

→ No. of partitions are fixed.
→ Size of each partition may or may not same
→ Since contiguous allocation, so spanning is not allowed.

pura process ek partition mein ayega. split nhi hga

LIMITATIONS →

- Internal fragmentation →



- Limit in process size. (Agr partition size se zyada size ki process aa jaye, unhe to store)
- Limitation on degree of programming → (no. of partition se zyada process nhi tab)

- External fragmentation → har partition mein suppose kuchh fragments hai and sare blocks allocate ho gaye. Ab new process comes whose size is less than the sum of fragments but can't be split and stored because contiguous partition. This condition is called external fragmentation.

> Whenever there is internal fragmentation, external fragmentation also exists.

- Variable partitioning → (DYNAMIC PARTITIONING)
whenever processes are coming into the RAM, only then we are allocating space to processes (Runtime ke time)

Advantages
→ No chance for internal fragmentation
→ No limitation on no. of process (Degree of multiprogramming)
→ No limitation of process size

Limitation 1
→ Suppose processes are allocated completely. Now, one process finishes (say P_2 = 4mb) and later (P_4 = 4mb) also finishes. We'll now have 2 holes. A new process (P_5 = 8mb) will come but can't occupy space because contiguous don't support spanning. ∴ This will be case of "External fragmentation"

This can be removed using "compaction"

OS	OS	But here
P1	P1	> we'll need to stop processes
Hole	P3	> cut paste them (it will take lot of time (hence undesirable method).
P3	P5	
Hole		
P5		

Limitation 2
→ Allocation & Deallocation is complex.

- Various memory allocation methods in contiguous memory management →

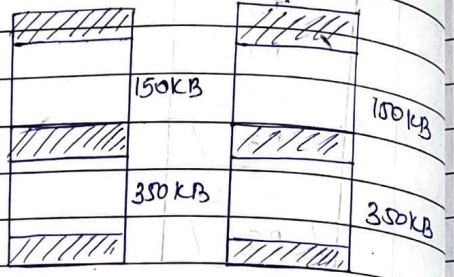
(1) First fit: Allocate the first hole that is big enough. (simple & fast)

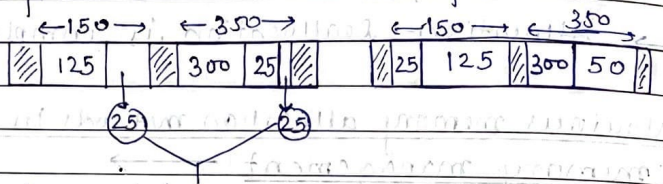
(2) Next fit: Same as first fit but start search from last allocated hole. (pointer)
(Same first fit ke jaise shure se search nahi start hoga har bar)

slow (3) Best fit: Allocate the smallest hole that is big enough (minimum internal fragmentation)

slow (4) Worst fit: Allocate the largest hole. (max int fragment).

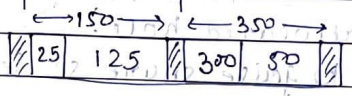
Q. Rquests from processes are 300K, 25K, 125K, 50K respectively
The above could be satisfied with
(a) Best fit but not first fit
(b) first fit but not best fit
(c) Both
(d) none



Best fit →


internal fragmentation

∴ (b) first fit but not best fit

→ using worst fit


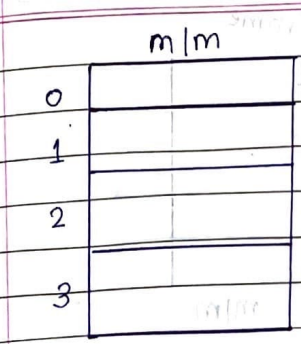
(no fragmentation)

Q

Request no	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Request size	2K	14K	3K	6K	6K	10K	7K	20K
Usage time	4	10	2	8	4	1	8	6
	4	11	3	10				

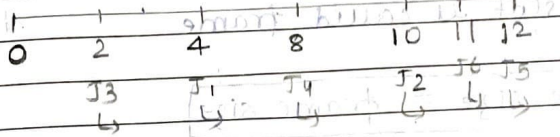
first fit
BEST FIT

Calculate the time at which J_7 will be completed
(a) 17 (c) 26
(b) 19 (d) 37



4K (0) ~~J3~~
8K (1) ~~J4~~ J5
20K (2) ~~J2~~ ~~J6~~ J7
2K (3) ~~J1~~

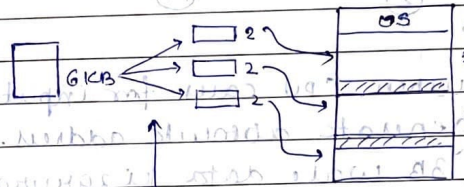
0 → J1
1 → J2
2 → J3
3 → J4
8 → J5
10 → J6



∴ (b) at 19, J_7 will be completed.

- **Need of paging →**

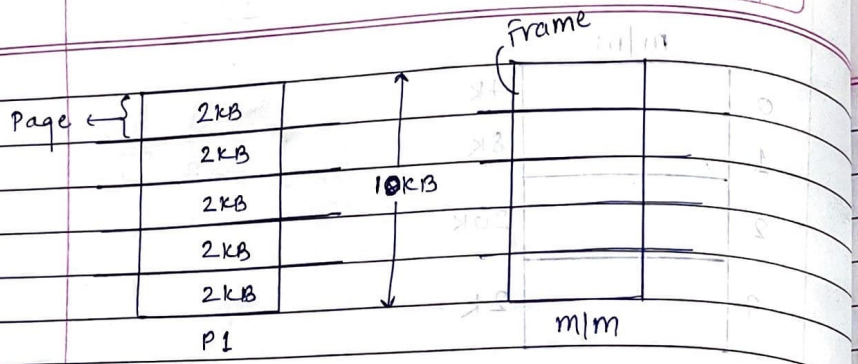
Non contiguous memory allocation →



Helps removing "external fragmentation"

This division is costly.

→ Holes are created dynamically (change during runtime)
→ If we break the process ~~in~~ after bringing in main memory, it will be inefficient.
→ Better approach is to break it before bringing to the main memory. i.e secondary memory itself. Each slot is called Page



→ main memory ko bhi pehle hidivide krenge
Q each slot ko called frame

* Page size = frame size

• Paging →

Process size = 4B
Page size = 2B
no. of pages/process
= 4B / 2B = 2

Pgno		Bytes
0	0	1
1	2	3

(P1)

mapping → when CPU calls for input, it doesn't generate absolute address. (Say CPU ko 3B wale data ki zaruurat hai but uss paging ke wajah se m/m mein different location pe stored hai). Hence we need to map the actual address.

• memory management unit →
The address generated by CPU is converted to actual adress using page table.

Page table → It contains frame no. where the page is stored in m/m.
→ Each process has its own page table.

Page table of P1 →

0	↑2
1	↑4

ji's bhi frame mein stored h

∴ 3B is in page no 1 which is stored in ↑4

→ CPU always works on logical address
(Pg no + Page offset)
1 1 (bit)

Physical address
3
frame offset size
4 bits → total phy address
frame no.
frame no ka binary in 3 bit.

frame offset = Page offset

Q. Logical add space = 4GB
PAS → 64mb
Page size → 4KB
No of pages = ?
No of frames = ?
No of entries in pg table = ?
Size of page table = ?

$\frac{64 \times 10^3 \text{ KB}}{4 \text{ KB}}$

∴ No of page = 16×10^3

*memory is byte addressable.

Page No.
Date

LA = $2^{2+30} = 2^{32}$

PA = $2^{6+20} = 2^{26}$

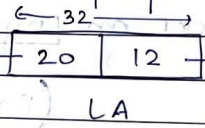
Pg size = $2^{2+10} = 2^{12}$

No of pages = $\frac{2^{26}}{2^{12}} = 2^{14}$

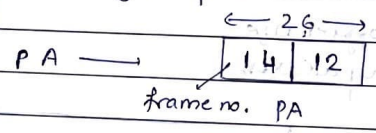
$2^1 = 2$	$2^7 = 128$		
$2^2 = 4$	$2^8 = 256$		
$2^3 = 8$	$2^9 = 512$		
$2^4 = 16$	$2^{10} = 1K$		
$2^5 = 32$	$2^{20} = 1M$		
$2^6 = 64$	$2^{30} = 1G$		
	$2^{40} = 1T$		

Logical add → 32 bits

Page size → page offset → 12 bits



(i) ∴ No of pages = 2^{20}



(ii) ∴ No of frame = 2^{14}

(iii) No of entries in page table = no. of pages = 2^{20}

(iv) Size of page table → No. of blocks × Size of each frame

(14 → frame no. calculate)

= $2^{20} \times 2$

= 2^{34}

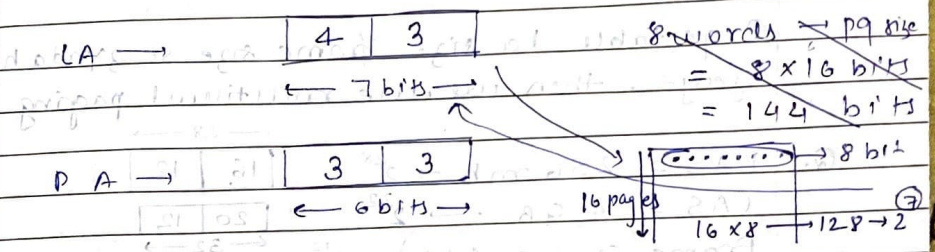
$2^{20} \times 14$ bits

(1 word = 1 byte)

In 8-bit

Page No. 75
Date (12-8)

Q. Consider a system which has LA = 7 bits, PA = 6 bits
Pg size = 8 words.
Calculate no. of pages and no. of frames.



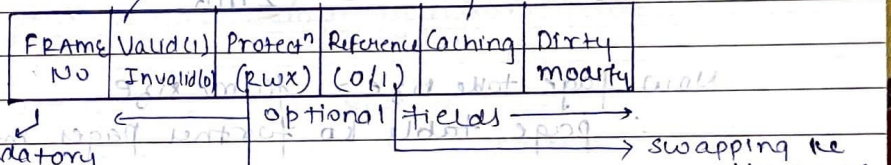
8 word ko represent krne ke liye → 3 bits

no. of pages = $2^4 = 16$

no. of frames = $2^3 = 8$

no of entries in pg table → 16

- **Page table entries** →



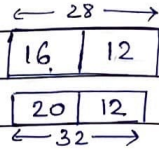
Ek page to part mein kabhi laye hai to mention kr sakte hai.

- Level of paging in operating system →

multilevel paging →

Page table ka size frame size se zyada ho jaye, then we use multilevel paging

Q. PAS = 256 mb → 2^{28}
LAS = 4 GB → 2^{32}
Frame size = 4 KB → 2^{12}
page table entry = 2 B



frame no → 16 bits
frame offset → 12 bit
page offset → 12 bit
page no → 20 bits.

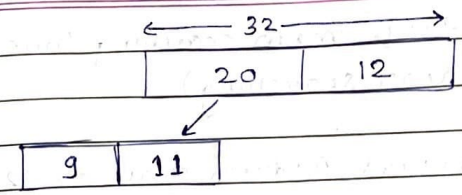
Page table size = 2^{20} × 2 Bytes
= 2 mB

Now page table size > frame size
∴ page table ko further pages mein divide

$\frac{2mB}{4KB} = \frac{2^{21}}{2^{12}} = 2^9$ pages. = 512 pages

Ab we'll make outer page table and use inner page table ke liye entries karenge jisme each block size will be same as inner one → 2B → page table entry

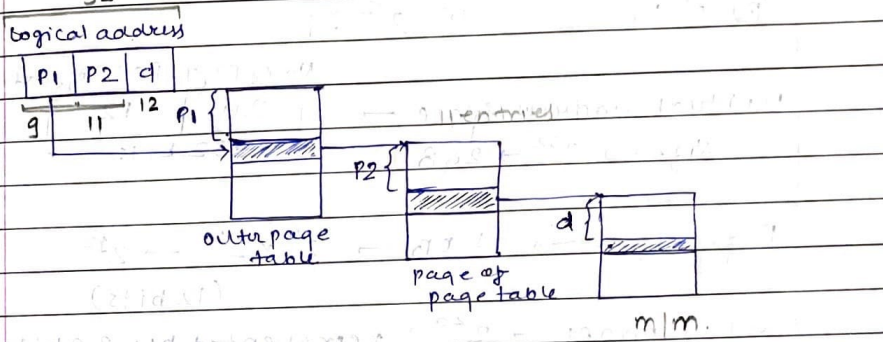
∴ size of outer page table = 512 × 2B = 1 kB



No of entries in outer page table

$\frac{4KB}{2B} = 2^{11}$

Address translation scheme →



- Inverted paging (memory management) →

normal page table
(1) Each process has its own page table
(2) Page table will be in main memory

Inverted paging → Inverted page table is the global page table which is maintained by the operating system for all the processes

Fr no	Page no	Process ID	
0	(P0)	P1	→ Page no 0 of Process 1
1	P1	P2	
2	(P0)	P3	Page no 0 of process 3.

But Inverted table mein searching time Increase ho gaya. (Linear searching)

Q. Consider a virtual Address space of 32 bits and page size of 4KB. System is having a RAM of 128 KB. Then what will be the ratio of page table and inverted page table size if each entry in both is of size 4B?

(A) $2^{15} : 1$ (C) $2^{10} : 1$

(B) $2^{20} : 1$ (D) $2^{12} : 1$

No. of pages	Page offset
20	12

← 32 bits →

virtual address space → size → 2^{32} → 2GB

Page size → 4 KB → $2^2 \times 2^{10}$ → 2^{12} (12 bits)

No of pages → 2^{20} represented by 20 bits.

Size of page table = $2^{20} \times 4B$

→ Phy add → size of m/m → 128 KB → $2^7 \times 2^{10} \rightarrow 2^{17}$ (17 bits)

5	12

← 17 →

No of frames in main memory → 2^5

Size of frame → $2^5 \times 4B$

∴ Ratio : $\frac{2^{20} \times 4B}{2^5 \times 4B} = \frac{2^{15}}{1}$ → (A) $2^{15} : 1$

Page No. 79

To solve Thrashing —
- m/m size increase
- long term scheduler

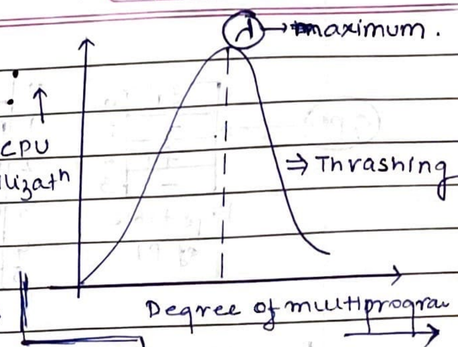
- **Thrashing** →

(bohot page fault → hence time consumption)

RAM → Limited size.

↓

Hence we need paging



> How to maximize DOMP →

(i) Sare processes ka atleast one page RAM mein layenge. — CPU, service time mein idle baitha rhega.

Discussing the worst case here →

CPU ⟷ P1 p1, P2 p1, :, P100 p1

What if CPU demands page 2 of any process? It will be "page fault"

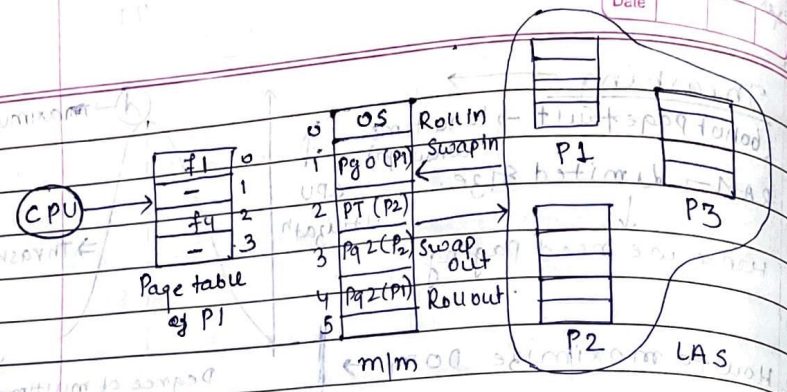
To rectify we use "page fault service time"

> Hard disk se page m/m mein layenge, which takes a lot of time

- **Page fault** →

Virtual memory → section of volatile memory created temporarily on the storage drive.

It is created when a computer is running many processes at once and RAM is running low.



Provides illusion for
(a) process size
(b) process no.

The page if CPU requests and it isn't present in main memory, it is called page fault i.e invalid bit.

page fault service →

The missing page wala block will generate a trap and control shifts from user to operating system (context switching)

Ek process jo chal rha hai, us ko ready state mein bhej dete hai and dusre process ko load kar dete hai running state mein).

Then OS will check authentication for user. Then OS will allow further process of fetching page from LAS.
Then that page will be brought from LAS to m/m. and the address of page will be stored in Page table.
And then control will be given back to the user.

Effective memory access time

no page fault

EMAT = P (Page fault service time) + (1-P) (m/m access)
(in sec) (milli sec) (MA) in (n sec)

P → probability of page fault occurs.

EMAT = p (page fault service time) + (1-p) (m/m access)
(milisec) (nano sec)

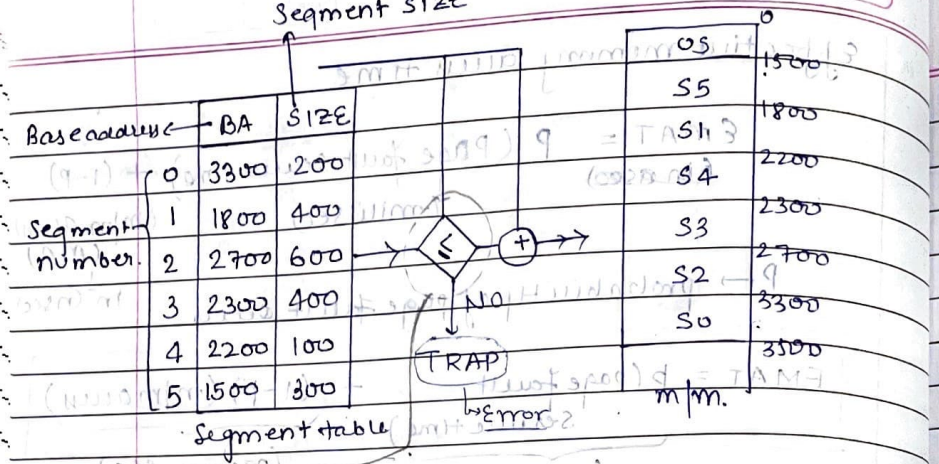
very negligible. in case page fault isliye usme add nahi kiya

• Segmentation →

Paging divide the program without knowing, what is written in the program. (Ek part ka size equal hota hai)

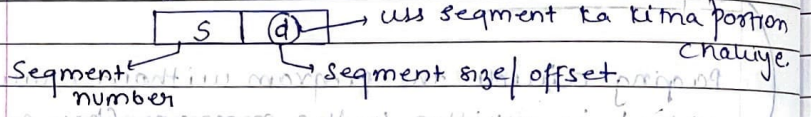
→ function ke acc segments bana diya (acc to user's point of view).
→ size of segment may vary.

Segments are stored in main memory without dividing equally.
It can be of various size.



CPU → generates logical address
↓ (using mem management unit)
we calculate physical address
(using segment table)

CPU → Logical address



→ uss segment ka kitna portion chahiye.

→ yaha, we are checking if $d \leq size$.

- **Overlay (memory management)** →

It is a method by which a large size process can be put to a main memory (from user end)
→ Reduce memory management
→ Reduce time requirement

(used in embedded system)

Q. consider a two pass assembler
pass 1: 80KB, pass 2: 90 KB

Symbol table : 30 KB
common routine: 20 KB

At a time only 1 pass is in use, what is min partition size required if overlay driver is 10 KB size.

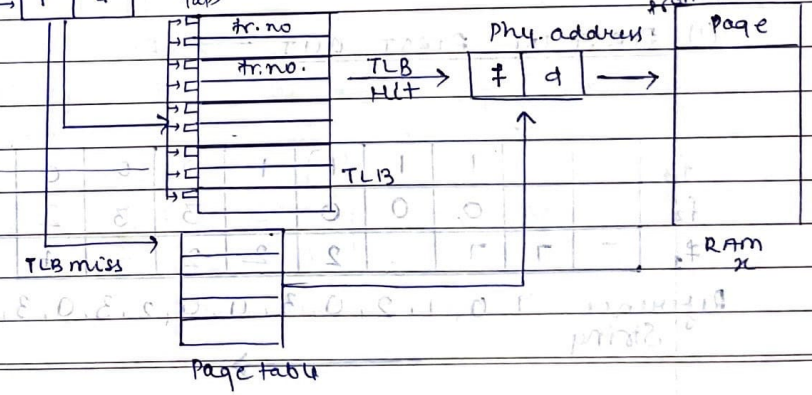
→ when m/m size > prog

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

max among both pass were sufficient size for all.

∴ (min partition size → 150 KB).

- **Translation lookaside buffer (TLB)** → (Cache)



- Cache is faster than RAM.

TLB Hit → Page fault nahi hai. $EMAT = \overset{Hit}{(TLB + x)} + \overset{miss}{(TLB + x + x)}$

with page fault, PF service time bhi add hoga

Q. A paging scheme using TLB. 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 fault.

$EMAT = 0.9\,(10 + 50) + (10 + 100)$

$= 0.9 \times 60 + 110$

$= 54 + 110 = 650$

$\therefore EMAT = 65$ ns

- **Page Replacement.** →

Page replacement algo →
(1) FIFO
(2) Optimal page replacement
(3) Least Recently used (LRU)

(1) FIRST IN FIRST OUT → (FIFO)

7⇌2 3⇌0 0⇌1 4⇌2 2⇌3 0⇌3

f_3			1	1	1	1	0	0	0	3	3	3	3	2	2
f_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f_1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0

Reference String → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

CPU demands byte stored at Page 7 — It is not present in frame. (Case of page fault).
Page service time → fetch and give pg. 7

Page Hit → 3
Page fault/miss → 12

Hit ratio $= \frac{3}{15} \times 100 = 20\%$

miss → $\frac{12}{15} \times 100 = 80\%$

- **Belady's Anomaly in FIFO** →

f_3			3	3	3	2	2	2	2	2	4	4
f_2		2	2	2	1	1	1	1	1	3	3	3
f_1	1	1	1	4	4	4	5	5	5	5	5	5

Ref string → 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 — Hits → 3, Page fault → 9

f_4				4	4	4	4	4	4	3	3	3	Hit: 2
f_3			3	3	3	3	3	3	2	2	2	2	miss: 10
f_2		2	2	2	2	2	2	1	1	1	1	5	
f_1	1	1	1	1	1	1	5	5	5	5	4	4	

No of frames badhane se no. of page fault bhi badh gaye. This is belady's anomaly.

Ye problem FIFO mein aati hai.

FIFO is suffering from belady's anomaly.

- **Optimal page Replacement Algorithm →**

Replace the page which is not used in longest dimension of time in future.

f4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
f2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7

Ref string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

(12) Hit ratio: $\frac{12}{20} \times 100 = 60\%$

(8) miss → 40%

7 sabse bad mein aa raha hai
compared to others
here first replacement use kiya

- **Least recently used → (LRU)**

speed issue (searching algo) compared to FIFO

Replace the least recently used page in past.
(jiski demand list mein sabse pehle huyi thi)

Ref string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

f4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
f3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
f2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	7	7	7	7

(12) Hit ratio → $\frac{12}{20} \times 100 = 60\%$

(8) miss → 40%

- **Most recently used page → (MRU)**

Replace the most recently used page in past.

f4				2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	0	0
f3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
f2		0	0	0	0	3	0	4	4	4	4	4	4	3_1	3_1	3_1	3_1	3_1	3_1	3_1
f1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7

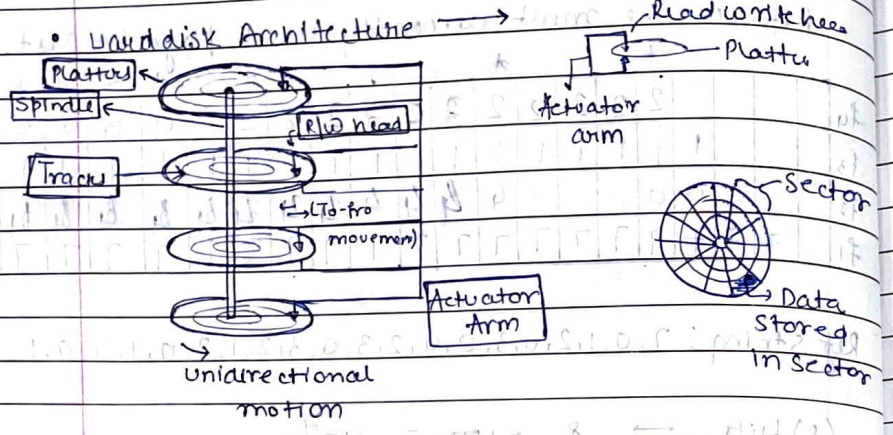
Ref String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

(8) Hits → $\frac{8}{20} \times 100 = 40\%$

(12) Fault / miss → 60%

• DISK ARCHITECTURE →

• Hard disk Architecture →



Platter → Surface → track → Sectors → Data

Ex: 8 × 2 × 256 × 512 × 512kB ~~= Total no~~

8 → Total Surfaces

Total tracks

Total Sectors

Total size of data stored

∴ Disk size = P × S × T × S × D

$= 2^{3+1+8+9+19}$

$= 2^{40}$ B

= 1 TB

No of bits req to represent disk size → 40 bits

Page No. 89
Date

Numericals

• Disk access time (Seek time, Rotational time, Transfer time) →

(1) Seek time → Time taken by R/W head to reach desired track.

(2) Rotation time → Time taken for one full rotation (360°)

(3) Rotation latency → Time taken to reach to desired sector (half of Rotation time)

(4) Transfer time : $\frac{\text{Data to be transfered}}{\text{Transfer Rate}}$

Transfer rate : (Data Rate) { No. of heads (surface) × Cap. of one track × No. of rotn in one sec }

Disk total access time → Seek time* + Rotn time* + Transfer time* + (Controller time + Queue Time) optional.

• Disk Scheduling Algorithm →

Goal : To minimize the seek time.

Seek time → Time taken to reach up to desired track.

① FCFS

② SSTF

③ SCAN

④ LOOK

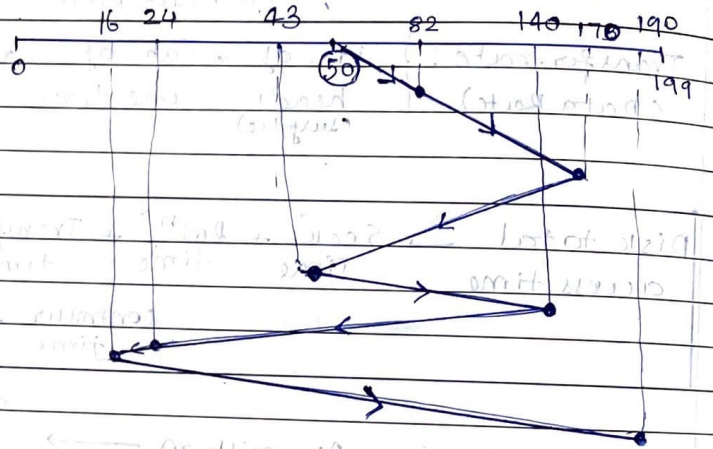
⑤ CSCAN

⑥ CLOOK

(1) First Come First Serve (FCFS) in Disk Scheduling Algorithm ⟶

Q] A disk contains 200 tracks (0-199)
Request queue contains track no.
82, 170, 43, 140, 24, 16, 190 respectively
Current position of R/W head = 50
Calculate total no. of tracks movement by R/W head.

⟶



(82 - 50) + (170 - 82) + (170 - 43) + (140 - 43) + (140 - 24) + (24 - 16) + (190 - 16)

— — — (OR) — — —

(170 - 50) + (170 - 43) + (140 - 43) + (140 - 16) + (190 - 16)

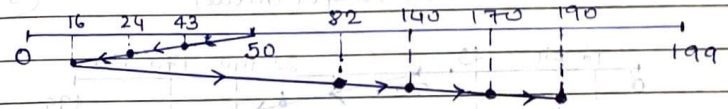
= 120 + 127 + 107 + 124 + 174

= 642

⟶ No starvation
⟶ But it acts without looking data (How performance issue)

(2) Shortest seek time first (SSTF) ⟶

Q] Data same as prev. que.
~~Calculate position of R~~
⟶ Calculate total no. of tracks movement by R/W head using shortest seek time first?
⟶ If R/W head takes 1ms to move from one track to another, then total time taken — ?



(50 - 43) + (43 - 24) + (24 - 16) + (82 - 16) + (140 - 82) + (170 - 140) + (190 - 170)

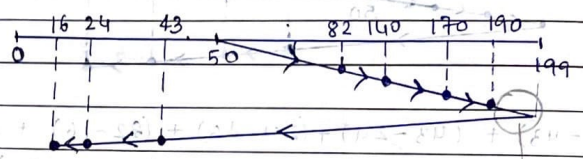
(OR)

(50 - 16) + (190 - 16) = 34 + 174 = 208

⟶ Tries to give optimum result.
⟶ Avg response time is good
⟶ There is starvation
⟶ Although, it gives optimum result, yaha overhead generate hoga (har step ke baad nearest point calculate karna padta hai)

(3) SCAN Algorithm → (elevator)

Q] 200 tracks (0-199)
Req track : 82, 170, 43, 140, 24, 16, 190
Current position of R/W head = 50
→ Calculate total no. of track movements by R/W head using SCAN algorithm.
→ If R/W head takes 1 ms to move from one track to another, then total time taken ? (Take direction towards larger value) (332 ms)



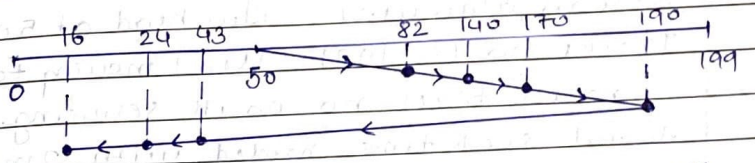
(82-50) + (140-82) + (170-140) + (190-170)
+(199-190) + (199-43) + (43-24) + (24-16)
(OR)
(199-50) + (199-16) = (332) movements

Dynamically, there is a possibility that it receives request in that range. So to avoid it, the head move till end and reverts

Disadvantage
Ek bar direction change kar di, and fir us side request aa gayi, it can't revert. Ab agla side se request hi par vapas aana padega.

(4) LOOK Algorithm → (same as scan but end tak nahi jaate).

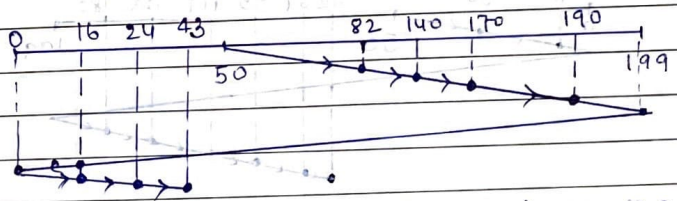
Q] Same as prev (use LOOK algo) (towards larger)



(190-50) + (190-16) = 314 movements.

(5) C-SCAN Algorithm →

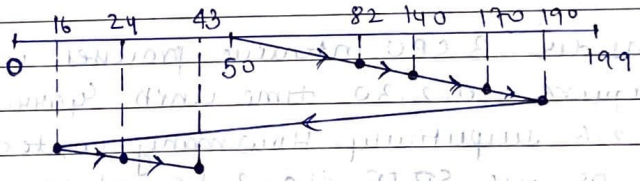
Q] Same as prev. (use C-Scan). (towards large)



(199-50) + (199-0) + (43-0) = 391 movements

(6) C-LOOK Algorithm →

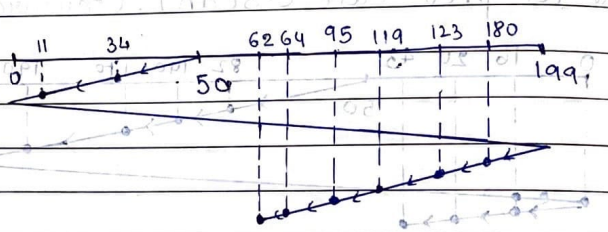
Q] Same as prev (use C-look) (towards large)



(190-50) + (190-16) + (43-16) = 341 movements

- **IMPORTANT QUESTIONS** →

Q. Track requests in disk queue
95, 180, 34, 119, 11, 123, 62, 64
C-scan algo used. R/w head at 50
Tracks no (0-199). Head moving to
smaller track no on its servicing.
total seek time needed with 2msec
time to move from one track to another
while servicing these req. is?
Assume moving one end to another end will
take 10msec.



~~(50-0) + 10 + (199-62)~~

$(50-0) + (199-62)$

$50 + 137$

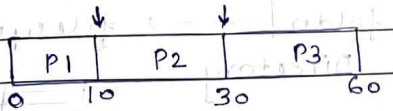
As per data given in qus

$50 \times 2 + 137 \times 2 + 10 = 384$

Q. Consider 3 CPU intensive processes which require 10, 20, 30 time units & arrived at 0, 2, 6 respectively. How many context switches if OS use SRTF algo? Do not count at time zero and at the end.

→

	Arr time	Burst time
P1	0	10
P2	2	20
P3	6	30



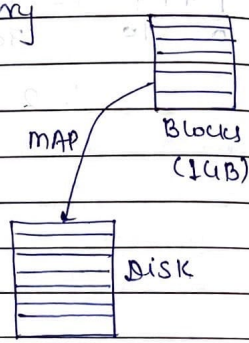
∴ 2 context switching.

* FILE SYSTEM →

Software (it manages files)

How data is stored and fetched

User → file → folder/ → file system
Directory



• file Attributes and operations →

file → collection of data/information

operations on files
(1) Creating
(2) Reading
(3) Writing
(4) Deleting
(5) Truncating
(6) Repositioning

file Attributes (metadata)
(1) Name
(2) Extension (Type)
(3) Identifier (OS POV)
(4) Location
(5) Size
(6) modified date, created date
(7) Protection/Permission
(8) Encryption/compression

Both file data & attribute deleted.

only data deleted

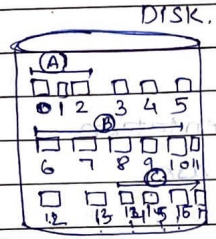
• Allocation methods →

Contiguous Allocat^n

Non contiguous allocat^n.
(1) Linked List Allocat^n
(2) Indexed Allocat^n.

Purpose:
(a) Efficient Disk Allocation.
(b) Access should be faster.

• Contiguous Allocation →



Directory →

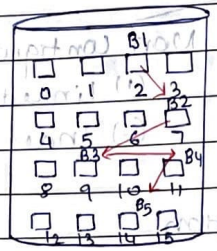
file	start	length
A	0	3
B	6	5
C	14	4

kitna part acquire kiya hai.

Advantages:
(1) Easy to Implement
(2) Excellent Read performance
(Sequential, direct)
↳ so that access time na bache.

Disadvantages:
(1) Disk will become fragmented.
(2) Difficult to grow file.

- **Linked list allocation** ⟶



Directory

file	Start
FA	2

Data	Pointer

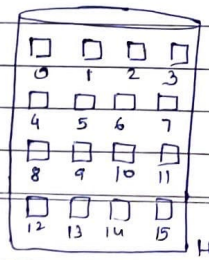
⟶ We use pointer [D|P] → [D|P] → [—]
Data ke sath pointer stored hoga, pointing towards the next block

› Advantages ⟶ :
(1) No external fragmentation
(2) file size can increase

› Disadvantages ⟶
(1) Large seek time
(2) Random Access / Direct access Difficult
(3) overhead of pointers

In case kisi particular file ka 4th block chahiye, uske liye pura search krna padega.

- **Indexed file Allocation** ⟶



Directory

file	Index Block
A	6

Hard Disk

Advantages :
(1) Support direct access
(2) No External fragmentation

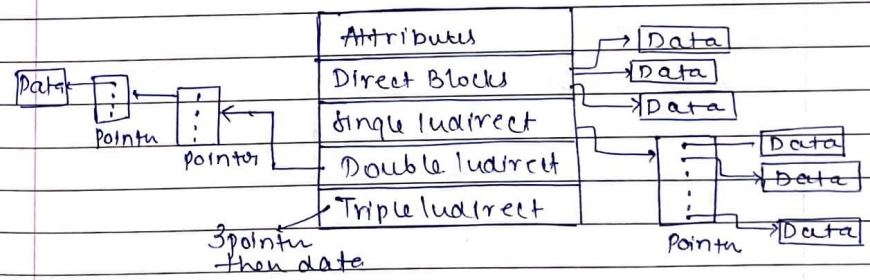
Disadvantages ⟶ OR
(1) Pointer overhead
(2) Multilevel Index (unnecessary in case of small box.)
↓
Andar box very large ; Index file ki size bohot zyada hojayegi

Ex: UNIX ⟶ (I)Node
↳ Index.

- **Unix INode Structure** ⟶

I ⟶ Index & Node

Hybrid approach ⟶ attributes, Direct blocks, Single Indirect, Double Indirect, Triple Indirect



Q. A file system uses Unix inode data structure which contain 8 direct block addresses, 1 indirect block, 1 doubly & 1 triple indirect block. Size of each disk block = 128 B & size of each block address is 8 B. Find the max possible file size?

⟶ 8 + 1 + 1 + 1 = 11 blocks

$128 = 2^7$

Indirect: Total no of pointers in 1 disk block = $\frac{128\,B}{8\,B}$

$= 16$

Double Indirect: Total pointers = 16×16

Triple Indirect: Total pointers = $16 \times 16 \times 16$

$\therefore$ Total addresses = $\left(8 + 16 + 16^2 + 16^3\right) \times 128\,B$

$= 547\,KB$

PROTECTION AND SECURITY ⟶

Unauthorized person bich mein interfere na kare in b/w process by other user or processes.

- Computer ⟶ (object + hardware + software)
- Each object has unique name & accessed through a well-defined set of operations.
- Protection problem → ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Security violation Categories (CIA)

Confidentiality, Availability, Integrity

(1) Breach of confidentiality (Encryption)
⟶ Unauthorised reading of data.

(2) Breach of Integrity
⟶ Unauthorised modification of data.

Encryption → plaintext to ciphertext.

(3) Breach of availability (24*7)
⟶ Unauthorized destruction of data

(4) Theft of service
⟶ Unauthorized used of resources

(5) Denial of service (DOS)
⟶ Prevention of legitimate user.

Principles of protection →

(1) Principle of least privilege
(can be static) (during system life, or during life of process) (could be dynamic) (changed by process as needed) - domain switching, privilage escalation.

Domain Structure →
Domain → Set of access-rights.
Access-right = <object-name, rights-set>

Access matrix →

→ View protection as a matrix (access matrix)
→ Rows represent domains
columns represent objects.

object / domain	F1	F2	Printer
D1	read		
D2			print
D3		read	
D4	read write		

The Security Problem →
- Threat → potential security violation
- Attack → attempt to breach security
  - accidental
  - malicious.

- masquerading (breach authentication)
  ↳ pretending to be an authorized user to escalate privilages.
- Replay attack
  → As is or with message modificat^n
- man-in-the middle attack.
  → Intruder sits in data flows, masquerading as sender to receiver & vice versa
- session hijacking
  → Intercept an already-established session to bypass authentication.

Program threats →

Trojan Horse →

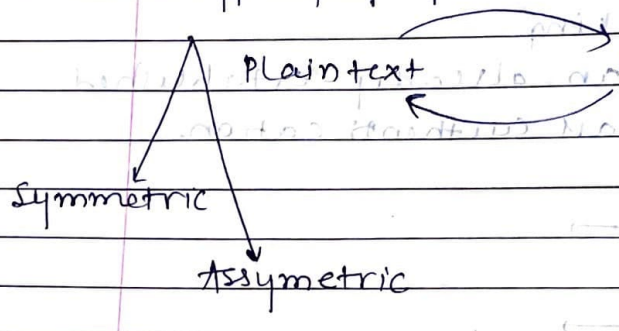
Viruses →

worms →

System & Network Threats

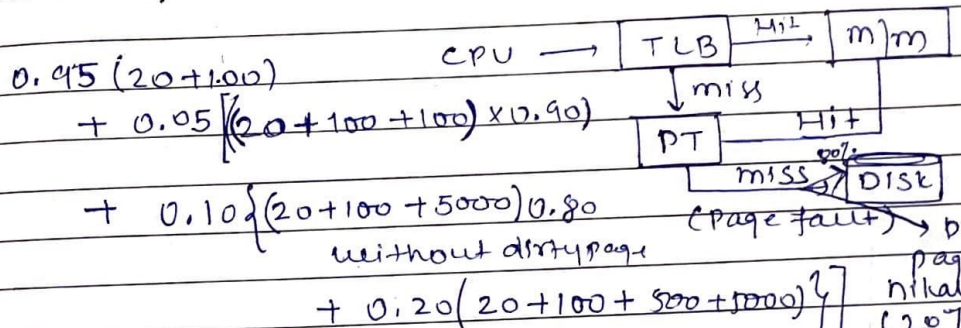
- Port scanning
- Denial of service

Cryptography →



Page No. 105
Date

Q. Consider a paging system uses 1 level page table residing in m/m and a TLB m/m access time 100 ns. TLB ~~load~~ lookup takes 20 ns. Each page transfer takes 5000 ns to/from DISK. TLB Hit ratio 95%. Page fault rate = 10%. Assume that for 20% of total page fault, a dirty page has to be written back to disk before required page is read in from disk. TLB update time is negligible. Find avg memory access time?



$0.95(20+100)$
$+ 0.05[(20+100+100) \times 0.90)$
$+ 0.10\{(20+100+5000)0.80$ (without dirty page)
$+ 0.20(20+100+5000+5000)\}]$

$= 154.5 ns$
$\simeq 155$