Resource allocation graph (RAG)

helps predict deadlock is present in system or not

It has nodes, vertices, edges
- process (P)
- Resource (R)

edges:
- request
- assignment

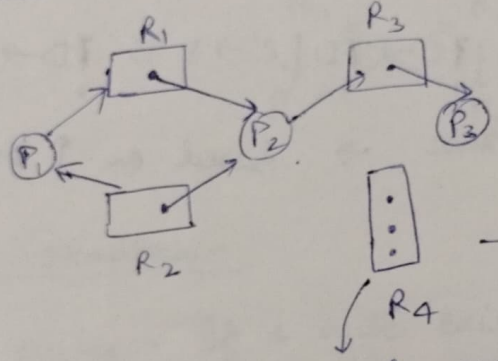
R → P
↓
edge
assignment edge
resources assigned to process

P → R process requesting resource
request edge

○ process node □ resource node

eg: of RAG

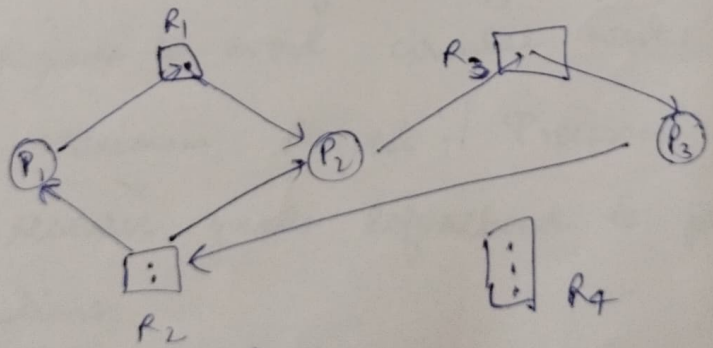
①



(no deadlock occuring)
→ no cycle, no deadlock

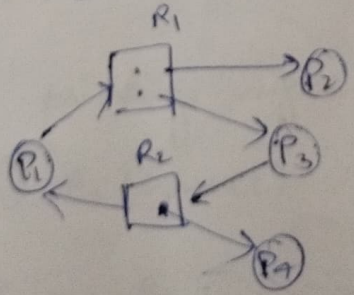
multiple instance of resource

②



cycle occured
∴ deadlock

③



→ no deadlock
cycle detected

If there is a deadlock then there is cycle, if no cycle then no deadlock. If there is cycle, then there may be or may not be a deadlock.

$D \rightarrow C \approx !C \Rightarrow !D$

* Cycle is necessary for deadlock but not sufficient.

note: If all R are of single instance then presence of cycle is necessary & sufficient condition for a deadlock.

$[D \rightleftarrows C \mid D \rightarrow C \mid \approx !C \rightarrow !D \mid C \rightarrow D \approx !D \rightarrow !C]$

for single instance → based on ↑ order of resources invested.

→ Resource Allocation Graph for deadlock avoidance

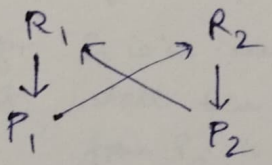
A RAG is a virtual representation of how processes request and hold resources in a system. It helps in detecting and preventing deadlocks, where process gets stuck waiting for resources indefinitely.

example -

Consider 2 processes P_1 & P_2 and 2 resources R_1 & R_2:-

① P_1 holds R_1 & request R_2

P_2 holds R_2 & request R_1



(deadlock)

To prevent deadlock -

① Detect cycles - If a cycle exists, deadlock is possible.

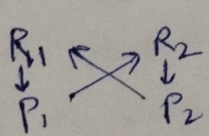
② Use Resource ordering - Assign a fixed order to resource requests to avoid circular wait.

③ Enforce Maximum Request - Processes declare their maximum resource needs beforehand to prevent deadlock-prone situations

Eg: Consider a printing system example:

R_1 (printer) R_2 (scanner) P_1 & P_2

Deadlock :



Deadlock avoidance:

Cycle detection - The system detects a cycle in RAG & prevents the second request.

Resource ordering - This rule could be 'always request scanner before printer to avoid circular waiting.

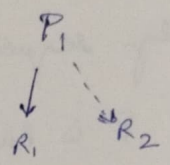
Maximum Request Allocation - Processes must declare in advance which resource they will use together.

→ Mechanism for Handling Deadlocks

① Deadlock Prevention: i) Dissatisfying mutual exclusion i.e. sharing resources, Some resources are inherently non-sharable & so it might be impossible to dissatisfy M.E.

ii) Dissatisfy no premption → snatch resources

P_1 is holding R_1 & waiting for R_2.



R_2 is free. & can be given to P_1

R_2 is not available & can't be given

P_2 is running, then P_1 must wait for R_2

P_2 is in waiting state then R_2 from P_2 is taken & given to P_1

iii) Dissatisfy Hold & wait

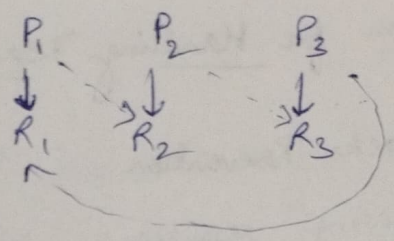
dissatisfy hold
here, we release all held resources.

Dissatisfy wait → not possible or efficient, but ek bounded wait lage sakte, iske baad Reso milne wi honge.

In one case, dissatisfy wait & in another case, we dissatisfy hold. In dissatisfy wait, process is allowed to start the execution only if it has been allocated the resources which ideally is not implemented as it isn't possible to know as we don't know how many R are needed by P beforehand. Also, it may result in low utilization of resources.

Dissatisfy hold - processes must release the held R b4 making any new R request. Next time, either it gets all the R after releasing or may not get many which may lead to starvation.

iv) Dissatisfy Circular Wait →



f: Resource set → natural nos.

eg:
f(tape drives) → 2
f(hard disks) → 4

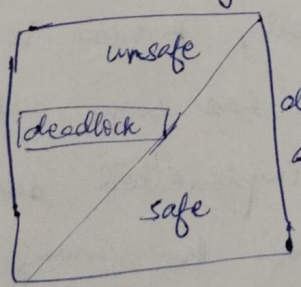
only request resources if:- $f(R_2) < f(R_1)$

If P is having R_i & requesting for R_j, R_j will only be given when $f(R_i) < f(R_j)$, which means the R must be given to the processes in ↑ order of enumeration. will dissatisfy circular wait.

② Deadlock Avoidance:

Safe state: A state is safe only if there exists a safe sequence. A sequence of processes $<P_1, P_2 \dots P_n>$ is a safe seq. for the current allocation state if for each P_i, the resource request that P_i can still make can be satisfied by the currently available resources + resources held by all P_j where $(j < i)$

If no such seq. exists, the system state is said to be unsafe state.



A safe state is not a deadlock state. Conversely, a deadlock is an unsafe state but not all unsafe states are deadlock.

Assignment: Resource Allocatn graph for deadlock avoidance

Eg. 12 R available (tape drives) → 3P → P_0 P_1 P_2

	max.	allocated	Need
P_0 →	10	5	5
P_1 →	4	2	2
P_2 →	9	2	7
		9	

12 − 9 = 3 available

CA = 3 + 2 + 5 + 2 ⇒ 12.

Safe seq :- $<P_1, P_0, P_2>$ at to

→ Banker's Algorithm

Safety Algorithm Resource Request Algorithm

① Total matrix

It tells how many R are available & in what amount.

Total = $[R_1\ R_2\ R_3\ \ldots\ R_m]_{1\times m}$

② Maximum Matrix

How many R are required by the processes in the system.

	R_1	R_2	R_3	R_m
P_1				
P_2				
P_3				
P_n				

$n \times m$

③ Allocation Matrix

How many R the processes have been allocated & of which type.

	R_1	...	R_m
P_1			
⋮			
P_n			

$n \times m$

④ Need Matrix

How many R are further needed by each process.

(need = max − allocation) matrix.

	R_1	...	R_m
P_1			
⋮			
P_n			

$n \times m$

Q: max matrix:

	a	b	c
P_0	6	5	4
P_1	3	4	2
P_2	1	0	4
P_3	3	2	5

total matrix:

a	b	c
7	9	10

allocation matrix:

	a	b	c	
P_0	0	3	4	⇒ 7
P_1	2	1	2	⇒ 5
P_2	0	0	2	⇒ 2
P_3	1	2	1	⇒ 3
	3	6	9	

avl:

a	b	c
4	3	1

Soln. need matrix:

	a	b	c
P_0	6	2	0
P_1	1	3	0
P_2	1	0	2
P_3	2	0	4

4	3	3
6	4	5
7	6	6
7	9	10

6	4	4
6	7	8
6	7	9
7	9	10

Safe sequences:

~~$P_2\ P_1\ P_3\ P_0$~~

$\langle P_1\ P_0\ P_2\ P_3 \rangle$ $\langle P_1\ P_0\ P_3\ P_2 \rangle$

$\langle P_1\ P_2\ P_3\ P_0 \rangle$ $\langle P_1\ P_2\ P_0\ P_3 \rangle$

→ Safety Algorithm

① let work & finish vectors of length m & n respectively.

Initialize work = avl & finish[i] = false for i = 0, 1 .... n-1

② Find index 'i' such that both

a) finish[i] = false

b) need ≤ work

if no such 'i' exists go to 4

③ work = work + allocation;

finish[i] = true

goto 2

④ If finish[i] == true for all 'i' system is in safe state.

Steps:

① calculate need matrix

② ~~three~~ avl total a, b, c ③ CA: Total − avl. total.

④ Need$_i$ ≤ C.A.

⑤ Pick the process & repeat.

C.A. = [4 3 1]

P_0 : 6 2 0

6 ≰ 4 ∴ P_1

P_1 : 1 ≤ 4 ✓ 3 ≤ 3 ✓ 0 ≤ 1 ✓

pick P_1

update CA: 4 3 1
2 1 2
6 4 3

∴ write down all steps briefly & you can write only 1 safe sequence. (not all needed).

Ques.

allocate matrix

	x	y	z
P_0	1	2	1
P_1	2	0	1
P_2	2	2	1
	5	4	3

max. matrix.

	x	y	z
P_0	2	2	4
P_1	2	1	3
P_2	3	4	1
	x	y	z
	5	5	5

Total

<P_1 P_0 P_2>

safe sequence.

① need matrix

	x	y	z
P_0	1	0	3
P_1	0	1	2
P_2	1	2	0

② avl. x y z

4 3 4

③ C.A. = [0 1 2]

P_0 ⇒ 1 < 0 ×

P_1 0 ≤ 0 ✓ 1 ≤ 1 ✓ 2 ≤ 2 ✓

④ pick P_1

⑤ CA: [2 1 3]

P_0: 1 < 2 ✓ 0 < 1 ✓ 3 ≤ 3 ✓

pick P_0

⑥ CA: [3 3 4]

pick P2 ✓

(RRA)

Let Request$_i$ be the request vector for process P_i. If Request$_i$[j] == k, then process P_i wants 'k' instances of resources type R_j. When a request for resources is made by P_i, following steps are followed:

Resource Request Algorithm

i) if Request$_i \leq$ Need$_i$ goto 2. Else raise an error, since process has exceeded maximum claim.

ii) if Request$_i \leq$ Available, goto 3. Else P_i must wait as resources aren't available.

iii) Pretend to have allocated the requested resources to P_i & modify as follows:

Available = available − Request$_i$;

Allocation$_i$ = Allocation$_i$ + Request$_i$;

Need$_i$ = Need$_i$ − Request$_i$;

If resulting resource allocation state is safe, transaction is completed & Process P_i is allocated the resources. But if new state is unsafe, P_i waits for Request$_i$ & old resource allocation state is restored.

Ques. Take 3 resources A, B, C. At t0, allocatn matrix

instances → 10 5 7

$P_0 \rightarrow P_4$

max matrix

	A	B	C
P_0	7	5	3
P_1	3	2	2
P_2	9	0	2
P_3	2	2	2
P_4	4	3	3

total matrix [10 5 7]

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	2
P_3	2	1	1
P_4	0	0	2
	7	2	5

need

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Soln

CA: [3 3 2]

5 4 3

5 4 5

10 4 7 10 5 7

<P_3 P_4 P_1 P_2 P_0> → safe sequence

Req: $P_1 \langle 1\ 0\ 2 \rangle$

① $R_i \le N_i$

$1 \le 1$ ✓ $\quad 0 \le 2$ ✓ $\quad 2 \le 2$ ✓

② $R_i \le A_i$

$1 \le 3$ ✓ $\quad 0 < 3$ ✓ $\quad 2 \le 2$ ✓

④ avl : [2 3 0]

All P_1 : [3 0 2]

need P_1 : [0 2 0]

	alloc			need			avail		
	x	y	z	x	y	z	x	y	z
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0	5	3	2
P_3	2	1	1	0	1	1	5	3	4
P_4	0	0	2	4	3	1	7	4	5
	8	2	7				7	5	5
							10	5	7

$\langle P_1\ P_4\ P_3\ P_0\ P_2 \rangle$

Assignment:

A request $\langle 3,3,0 \rangle$ in this scenario is made by P_4, it can be granted or not. Can a request by process P_0 $\langle 0\ 0\ 2 \rangle$ be granted or not after this.

avl: [0, 0, 2]

P_4 $\langle 3\ 3\ 0 \rangle$

① $R_i \le N_i$ $\quad 3 \le 4$ ✓ $\quad 3 \le 3$ ✓ $\quad 0 \le 1$ ✓

② $R_i \le A_i$ $\quad 3 \le 3$ $\quad 3 \le 3$ $\quad 0 \le 2$ ✓

need won't get completed.

Request can't be granted.

④ avl: [0, 0, 2]

all P_4 : [3 3 2]

need: P_4 : [1, 0, 1]

① $0 \leq 7$ $0 \leq 4$ $2 \leq 3$ ✓

②

$0 \leq 3$ $0 \leq 3$ $2 \leq 2$ ✓

③ all : P_0 [0 1 2]

CA : [3 3 0]

need : [7 4 1]

Req. can't be granted.

③ Deadlock Detection & Recovery –

[only for single instances : wait for graph (for dlk detection)]

↳ For multiple instances :

1) Let work & finish be vectors of length 'm' and 'n' respectively. Initialize work = available for $i = 0, 1 \ldots n-1$
if allocation $\neq 0$ then finish[i] = false else finish[i] = true

2) Find an index 'i' such that
   a) Finish[i] == false
   b) Request$_i \leq$ work
If no such 'i' exist goto 4.

3) work = work + allocation$_i$
   finish[i] = true
   goto 2

4) If finish[i] == false for some 'i', $0 \leq i \leq n$, then system is deadlocked. Also if finish[i] == false, then process P_i is deadlocked.

Given:-

Allocation matrix at time to

	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2
	7	2	6

total instances

A	B	C
7	2	6

Request matrix

	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

CA: [0 0 0]

Soln.

work = 0 0 0 P_0 P_2 P_3 P_4 P_1

0 1 0
3 1 3
5 2 4

∴ safe sequence

at t_0 : P_2 : 0 0 1

then only P_0 can be fulfilled

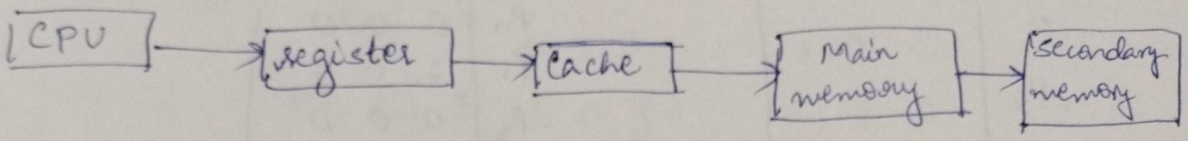
& P_1 P_2 P_3 P_4 are in deadlock & not finished.

Deadlock Recovery

- Process Termination
  1. abort all deadlocked processes
  2. abort processes one by one
     - based on priority
     - types of process
     - amt. of resources holding
     - time of wait & execution
  4. Ignorance
- Resource Preemption
  1. select a victim
     - HP will evade R from low prior. P.
  2. Rollback
  3. Starvation

→ Memory Management

size ↑ per unit cost ↑ access time ↓



Locality of Reference – S.M. is larger & has less per unit cost but more access time ∴ instructions from SM are preferred & kept in the main memory for less access time.

Ques. 100 ms to access data from SM
10 ms to access data from MM
hit ratio = 90% Find total access time.

Soln. first check in MM → 10 × 0.9
then in SM → (10+100) 0.1

Ans. 9 + 11 = 20 ms

OS will translate the logical address generated by CPU to the physical address of main memory.

OS decides which data in SM gets which area in the main memory.

→ Memory Allocation Policies

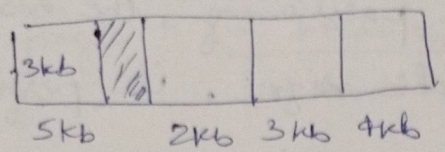
Contigous Non-Contigous

A. Contigous — static / fixed
— variable / dynamic

When bringing a process from SM to MM allocate it in MM consecutively together & in continous partition which also has a past further fixed size partioning &

Variable size partioning.

eg:



Internal fragmentation – In fixed size, if any process's size is less than the size of the partition then when the process gets inside the memory, it creates a hole which is called as internal fragmentation. No problem in variable size.

External fragmentation – When the memory is sufficiently available to store a process but not in continuous form then we cannot accomodate that process & this is called external fragmentation.

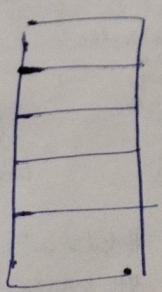
Solution: memory compaction: when memory is reallocated to the processes so that all holes get combined to form one bigger hole & we end on partition allocation algorithms.

① Best fit ② Worst fit ③ First fit ④ Next fit

Question: Consider 5 memory partitions of size 100kb, 500kb, 200kb, 300kb, 600kb. If seq. of request for block sizes 212kb, 417kb, 112kb, 426kb arrive in the same order, which of the following makes efficient use of memory?

① Best ② First ③ Next ④ Worst

Soln.



	Best	worst	first	next
100kb				
500kb	417	417	212	212
200kb	112		112	112
300kb	212	112		
600kb	426	212	417	417
	433 left	~~659~~ 659	659	659

(However, keep any process in the left space).

Prove:

Claim 1: If it is fixed size partioning, the best fit performs best but in variable size, it performs worst.

Claim 2: In fixed size partioning, worst fit performs worst but in variable size, performs best.

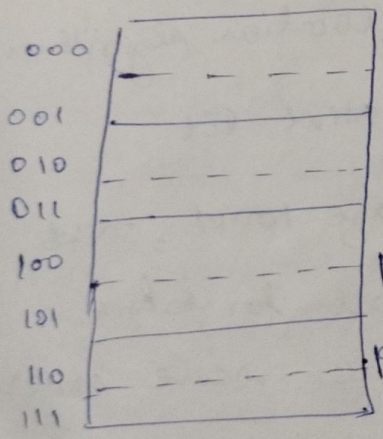
If they are right, prove it with examples.

Non-Contiguous Memory Allocation

A. Paging

Segmentation

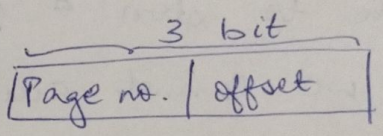
The logical address space or secondary memory is divided into equal sized partitions & each partition is called a page.



Process A (8 byte)

$$\#Pages = \frac{\text{logical add. space (LAS)}}{\text{page size}}$$

$$= \frac{8 \text{ bytes}}{2 \text{ bytes}}$$

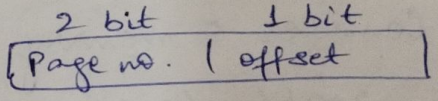


L.A.

no. of bits in page no. = $\log_2$(# pages)

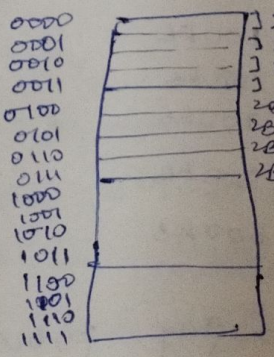
eg: $= \log_2 4$

$= 2$



Ques. LAS is 32 bytes & memory is word addressable. word size = 2by. Find out no. of words in LAS.

Soln. # words = 32 / 2 = 16



$\log_2$ # words

$\log_2 16$

bits in # words = 4 bits

page size = 8 bytes = 4 words

$$\# \text{ pages} = \frac{2^4 \text{ words size}}{4 \text{ words size}} = \frac{32}{8} = 4$$

page no.	offset
2 bits	2 bits

4.

Ques. [Frame size is just like page size but its in Main memory]. F. size = 2bytes Here, main memory is Physical address space [PAS] = 8 bytes. Find no. of frames in MM & PA bits & bits division for frame no. & offset.

Soln $\frac{8}{2} = 4 =$ no. of frames in MM.

$\log_2 4 = 2$ $\qquad 8 = 2^3 \rightarrow$ bits.

frame no.	offset
2	1

3

Ans.

Ques. If LAS = 32 bits, PAS = 8 bits, Page size = 8 bytes. Find LAS, PAS, no. of pages & no. of frames.

Soln $\frac{8}{8}$ $\qquad \frac{2^{32}}{8} = 2^{29} = 2^{29} \Rightarrow$ no. of pages.

PAS:

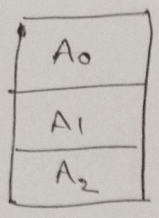
page no.	offset
2^{29}	2^3

2^{32}

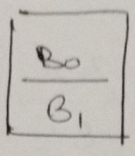
frame no.	offset
2^5	2^3

2^8

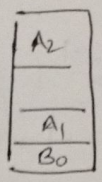
Ans. Here, frame size = page size.



Process A (6B)



Process B (4B)



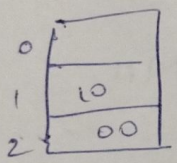
frame 0 = 00
frame 1 = 01
frame 2 = 10
frame 3 = 11

more than one page of more than one process.

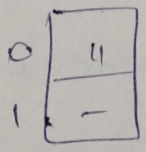
Page Table

To know, that which page of which process is running at / which frame no. in the main memory, page table can be used.

More than 1 page of more than 1 processes can be placed anywhere in the main memory



Page table for process A



Page table for process B

Page table is created for each process that stores the info of frames nos. of respective pages.

Translation Lookaside Buffer (TLB): cache — some portion of pagetable will be here

In simple paging mechanism, main memory has to be accessed twice, one for accessing the page table through the page no. & then for actually accessing the main memory — PAS (frame + offset). to reduce the effective main memory access time, a cache memory that contains some part of the page table is used which is called translation

lookaside buffer.

H → hit ratio of TLB cache.

M → main memory access time

C → cache memory access time

Effective main memory access time (EMMAT) ⇒ $H(M+C)+(1-H)(2M+C)$

Ques. Considering a paging mechanism if main memory access time is 100ns & TLB having 80% hit ratio & access time as 20ns.

Soln. $EMMAT = 0.8(100+20) + 0.2(200+20)$

$= 140ns.$

Page Table uses:

i) Which page no. is running on which frame

ii) Protection Bit

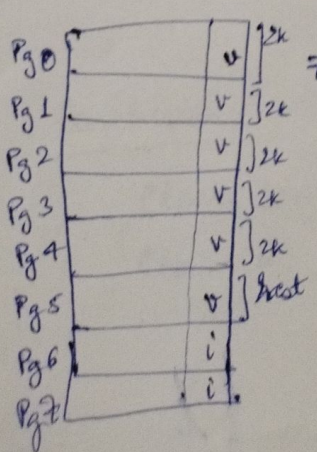
iii) Page valid / invalid

Page is in process's LAS ⇒ valid

Ques. LA = 14 bit Page size = 2kb Process size = 10469 bytes.

Find no of pages in LAS.

Soln. LAS: ⇒ 2^{14} bytes. Page size = $2^{1} \cdot 2^{10} = 2^{11}$



pages = $\frac{2^{14}}{2^{11}} = 2^3 = 8$ pages

v → valid

i → invalid

Inverted Paging Exists

The memory is being wasted by storing the info about the invalid pages in the page table. The total # valid pages in the system for all processes cannot exceed the no. of frames in ~~each~~ MM to avoid this ~~working~~ wastage. ~~the~~.

Here, we create the frame table or IPT where no. of entries is same as no. of frames in MM.

Ques: LA = 32 bits PA = 16 bits.

If paging is applied with page size 1024 bytes & each page table entry contain 4 bytes info. Calculate.

A. ① no. of pages ② # ~~of~~ frame.

B. Page table size in ① simple paging & ② inverted paging

Soln: LAS = 2^{32} bytes PAS = 2^{16} bytes

Page size = 2^{10}

A. # pages = $\frac{2^{32}}{2^{10}} = 2^{22}$

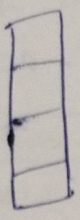
~~fra~~ frames = $\frac{2^{16}}{2^{10}} = 2^6$

B. ① $4 \times 2^{22} = 2^{24}$ ② $2^6 \times 2^{22} = 2^{28}$

B. Segmentation

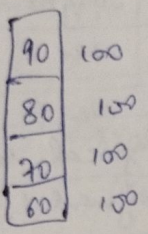
Example 1: Fixed-Size Partitioning

mem. blocks



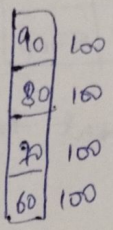
Processes: 90KB
80KB
70KB
60KB

Best Fit



Performs well with minimal internal fragmentation

Worst Fit



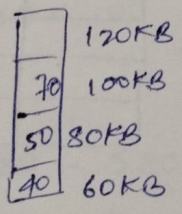
no optimization here

Example 2: Variable-size Partitioning

mem. blocks

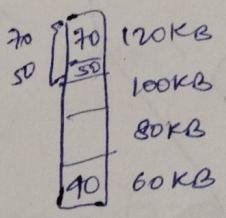
70, 50, 40

Best



higher fragmentation
worst performance

Worst



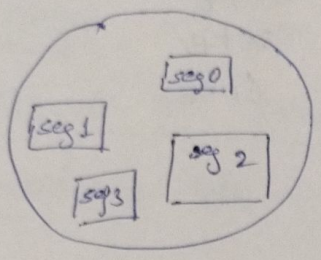
efficient usage of largest memory
∴ Performs best in this case

Conclusion:

Claim 1 is TRue → Best fit ✓ in fixed size
Claim 2 is True → worst fit ✓ in variable size

B. Segmentation

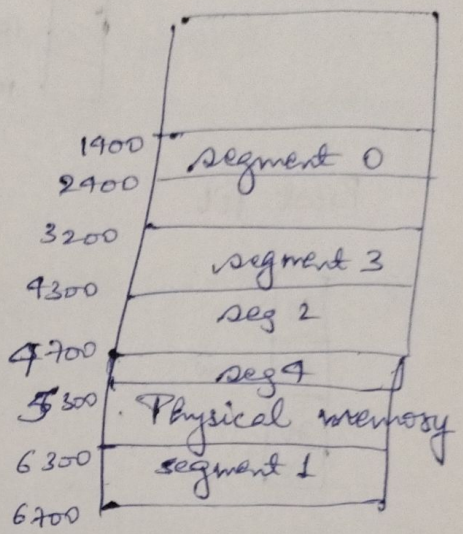
divided it unequal sizes. accessed through MM.



different sizes

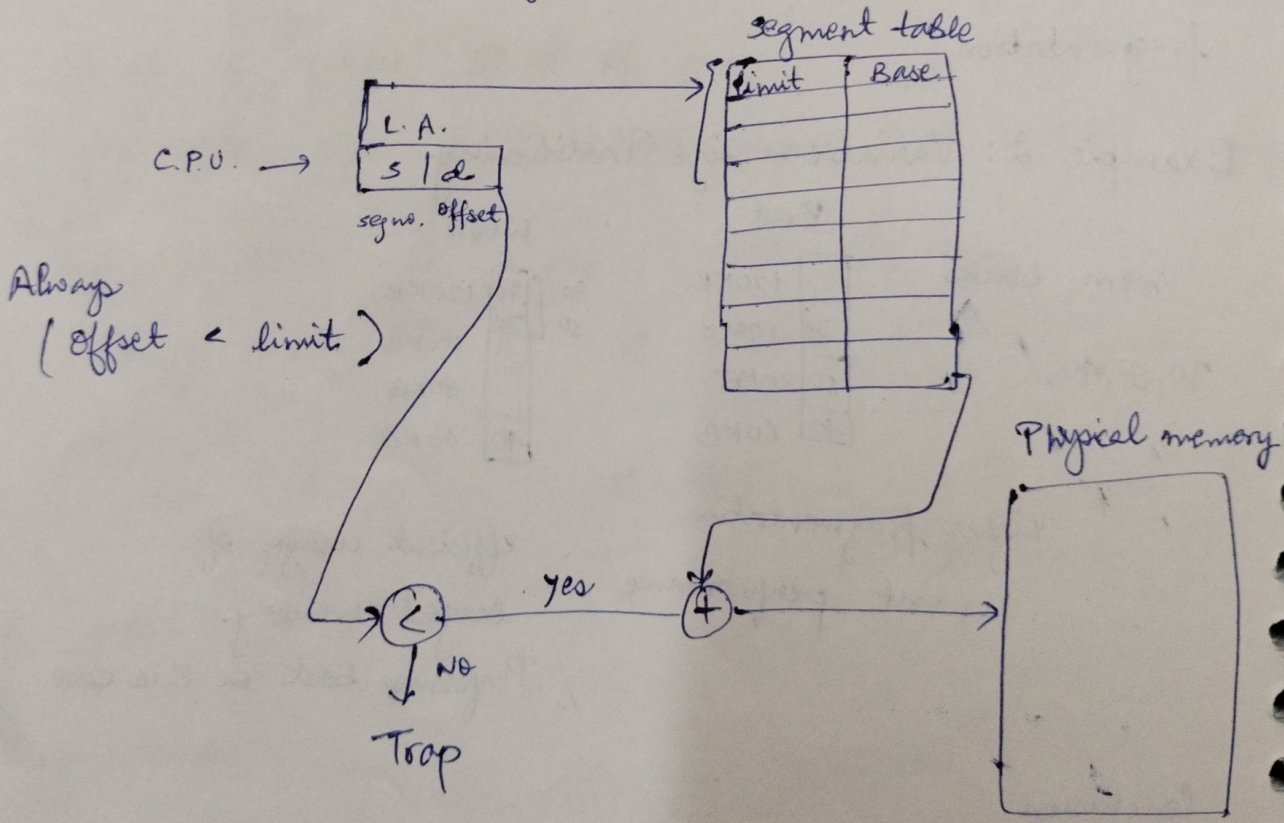
	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



limit is ⇒ length

base is ⇒ starting address



L.A. is divided into segment no. & offset. Seg. No is used to index into the segment table to obtain the limit & base for that segment.

Offset value is compared ~ the limit. If it is smaller then

it will be added to the Base value to obtain PA. else it results into a trap.

Ques.

Seg.	Base	length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Segment table

What are the PA for the given LA. ?

① 0, 430
② 1, 11
③ 2, 500
④ 3, 400.
⑤ 4, 112

Soln.

① 0, 430

$430 < 600$ ✓

$430 + 219 = 649$

② $11 < 14$ ✓

$11 + 2300 = 2311$

③ $500 < 100$ ×

∴ Trap (error)

④ $400 < 580$ ✓

$400 + 1327 = 1727$

⑤ $112 < 96$ ×

∴ Trap (error)

→ Page Replacement Algorithms

Page fault / Miss → Page not found in MM, so we check in SM.

If all pages are filled, we replace it using algos.

Performance of Demand Paging – $p \times$ (page fault time) + $(1-p)$ main memory access time

① FIFO Policy –

Reference string: < 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0 >

frames:

F3			1	1	1	1	0	0	0	3	3	3	3	2	2
F2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	PF	PF	PF	PF	H	PF	PF	PF	PF	PF	PF	H	PF	PF	H

Hit = 3
PF = 12

$\frac{12}{15}$ $\frac{3}{15}$

PF Ratio = $\frac{4}{5}$ $\frac{1}{5}$ = Hit ratio

80% 20%.

Ans.

Ques.

RS. < 1 2 3 4 1 2 5 1 2 3 4 5 >

frames:

3			3	3	3	2	2	2	2	2	4	4
2		2	2	2	1	1	1	1	1	3	3	3
1	1	1	1	4	4	4	5	5	5	5	5	5
	F	F	F	F	F	F	F	H	H	F	F	H

H = 3/12 = 1/4

frames:

4				4	4	4	4	4	4	3	3	3
3			3	3	3	3	3	3	2	2	2	2
2		2	2	2	2	2	2	1	1	1	1	5
1	1	1	1	1	1	1	5	5	5	5	4	4
	F	F	F	F	H	H	F	F	F	F	F	F

H = 2/12 = 1/6

Belady's Anomaly :-

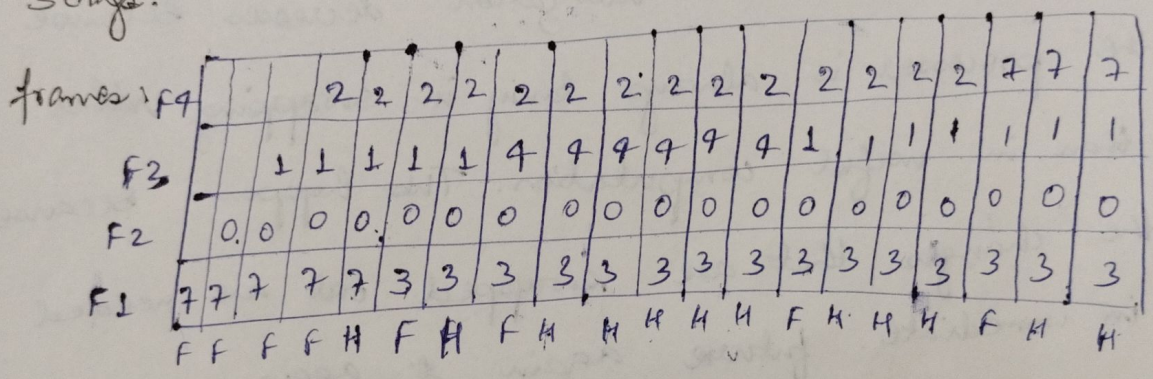
Even after ↑ the ~~p~~ # page frames, page faults ↑ which is unexpected, found (in FIFO policy).

(like in prev. ques.)

② Optimal Page Replacement Algorithm

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

Reference strings: < 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 >

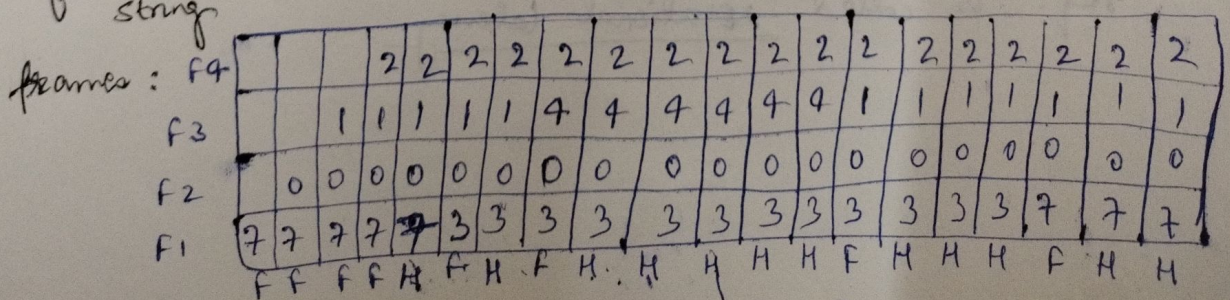


Hits = 12 Page faults = 8

$HR = \frac{12}{20} = \frac{4}{5} \Rightarrow 80\%$

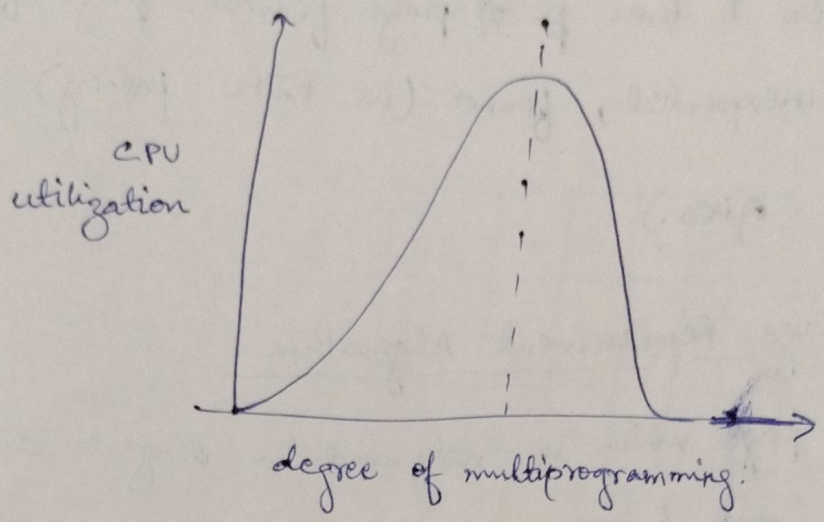
③ Least Recently Used (LRU)

Reference string: < 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 >



Hits = 12 Faults = 8

→ Thrashing —



When we ↑ the degree of multiprogramming beyond a certain level, the CPU utilization decreases because the processor is always busy in swapping rather than in useful computation. This happens because the changes that are swapped out are needed in immediate future again & again.

→ Disk Scheduling —

Seek Time — Time required to move the read-write head on the desired track.

But to take the head on the desired sector, the time req. is called rotational latency.