Memory

Tape ⟶ to access sequentially data } Non Volatile

Disk

↓

Performance in Random Access is bad.

Slow & Sequential in Access

Random Access Memory ⟶ Time taken for seq and random memory access is same and comp. better than tape and disk

Very Fast

Registers ⟶ Also random Access memory locations

it takes more time to access any memory because of its size (greater size more time)

Size of RAM < Size of registers

RAM → also called Main Memory or Memory

Cache

Tape, Disk → Secondary storage

Value of variables may or may not be stored in disk

suppose stored in disk then during execution they are brought to RAM (Method 1)

→ Transfer all variables at the beginning in RAM

Method-2 → On Demand Transfer (Demand Paging)

Variable Brought when called first

Peterson Book

Demand Paging

Variables in disk

a: 65	b: 37	c: 84	d: 49	e: 86	f: 74
g: 36	h: 42	i: 49	J: 63	DISK	

In RAM there is space to store variables

a	b	c	d	e	f	g	h	i	j
65			49				42		

print(d) ⟶ print value of d

| 149 Disk ⟶ RAM

---

print(a) | 165 Disk ⟶ RAM

---

print (h) | 142 Disk ⟶ RAM

---

print (a) No transfer (value of a is already available)

---

print (c) | 84 D ⟶ R

	a	b	c	d	e	f	g	h	i	j
h = 36								2 36		

No transfer

---

c = 29 | 2 29 | No. tran.

---

f = 36 | 2 36 | No trans.

Over | 165 | 2 29 | 149 | 236 | 36 R⟶D | 29 R⟶D | 36 R⟶D

h 2 36 | c | f

When work is done updated variables are updated in disk

Paging ⟶ Collection of continuous memory

a, b ⟶ P

c, d ⟶ Q

e, f ⟶ R

g, h ⟶ S

i, j ⟶ T

Always transfers will be in of the form blocks

	P	Q	R	S	T	
8449 D → RAM		8449				print(d)
6537 D → RAM	6537					print(a)
3642 "				3642		print(h)
No transfer						print(a)
No transfer				unchanged		Print(a)
No transfer				36 36		h = 36
No trans.		2949				c = 29
			_36			f = 36

36 36, 2949 RAM → DISK

for value of f → _36 cannot be transfered

So 8674 DISK → RAM

8636 RAM → DISK

	a	b	c	d	e	f	g	h	i	j
Over	65		29	49		36		36		

how it is decided which values needed to be update at the end or not

- Dirty bit → is attached to the memory
  - 1 (unupdated)
  - 2 (updated)

Dirty bit 0 Disk ≠ RAM

1 Disk = RAM

2 → Value is diff in RAM & disk and correct is RAM

0 → 1
0 → 2
1 → 2

Dirty : Paging

0	Disk ≠ RAM	(Disk is correct) Ram wrong
1	Disk = RAM	Ram & Disk has all values correct
2	Ist Correct	Ist value correct in RAM 2nd correct on disk
3	2nd Correct	Ist value in disk & 2nd on RAM is correct
4	Disk ≠ RAM	(RAM is correct) RAM has both values correct)

1 → 2, 3

0 → 1, 2, 3
1 → 4
2 → 4
3 → 4

Always transfers will be in of the form blocks

	P	Q	R	S	T	
84490 → RAM		184449				print(d)
65376 → RAM	16537					print(a)
3642 "				13642		print(h)
No transfer						print(a)
No transfer				unchanged ↑		print(a)
No transfer				436 36		h = 36
No trans.		42949				c = 29
No transf			3 _ 36			f = 36

	a	b	c	d	e	f	g	h	i	j
	094	063	038	079	042	018	021	031	089	079
print(d)				149						

49 D→R

Always transfers will be in of the form blocks

	P	Q	R	S	T	
84490 → RAM	09463	18449	04218	02131	08979	print(d)
6537 D → RAM	16537					print(a)
3642 "	16537	18449	04218	13642	08979	print(h)
No transfer						print(a)
No transfer				unchanged		print(c)
No transfer				43636		h = 36
No trans.		42949				c = 29
			04218			f = 36
			34236			e = 29
			42936			

3636, 2949 RAM → DISK

Virtual Memory (RAM has the capacity to store only 3 variables)

* RAM has less memory capacity than program required

	U	V	W	X	y
	z31	z79	z12	717	z42
print(d)	d49				
print(a)	d49, a65				
print(h)	d49, a65, h42				
print(a)	same				
print(c)	d49, a65, h42, c84				
h = 36	d49, a65, h36, c84				
c = 29	d49, a65, h36, c29				
f = 36	d49, a65, h36, c29, f36				

. ⟶ for updated value

	K	L	M	N
	(4 Pages available in RAM)			
print(d)	Q8449			
print(a)	Q8449, P6537			
print(h)	Q8449, P6537, S3642			
print(a)	Same			
print(c)	Same			
h = 36	Q8449, P6537, S3636			
c = 29	Q2949	"	"	
f = 36	Q2949, P6537, S3636, R8136			

R8136 → garbage

U V W X Y
z31 z79 z12 717 z42

d49

d49, a65

d49, a65, h42

same

d49, a65, h42, c84

d49, a65, h36, c84

d49, a65, h36, c29

d49, a65, h36, c29, f36

Only 3 variables
then

1. first in first out
2. LRU → least recent use
3. Optimal

print(b)

print(h)

print(d)

print(b)

print(R)

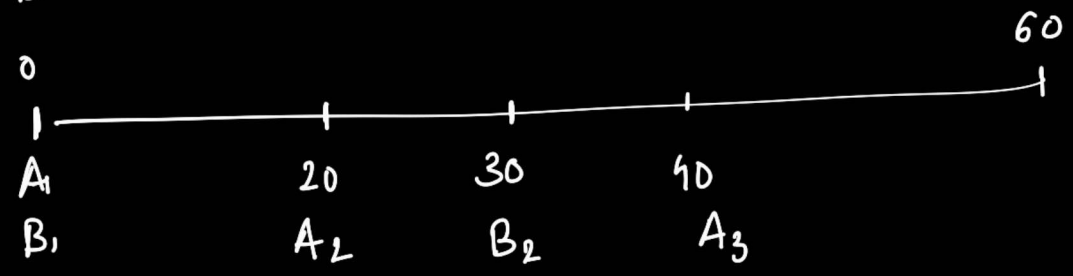
print(h)

print(b)

* Rate Monotonic Scheduling

Periodic Jobs Deadline → A job should be over when next job of same type come.

Job	Period	Service
A	20	5
B	30	12



A_1 0 - 5

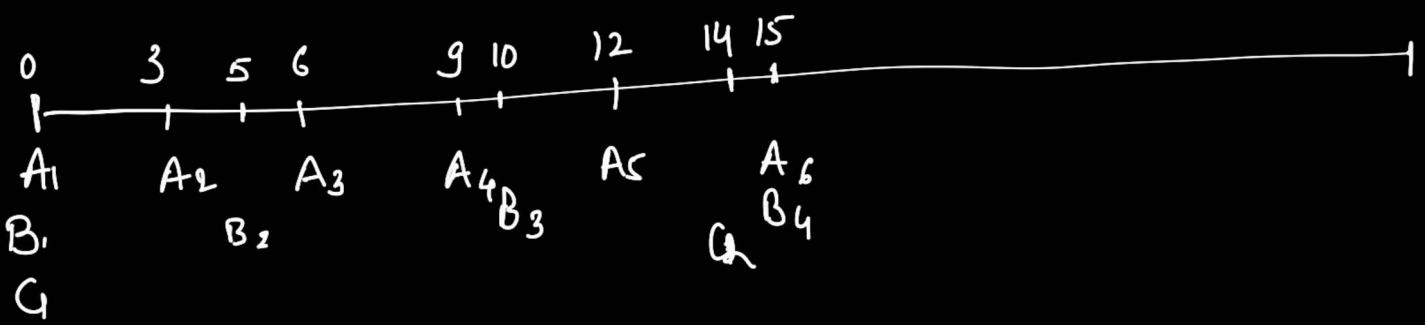
B_1 5 - 17

A_2 20 - 25

B_2 30 - 42

A_3 42 - 77

Job	Period	Service
B	5	1
A	3	1
C	14	3



A_1	0-1	A_2	5-6	A_3	7-8	A_4	9-10
B_1	1-2	B_2	6-7	B_3	10-11	A_5	12-13
C_1	2-5	C_2	14-17				

A_6
B_4

- Job with lesser time period is given priority

B	50	9
A	30	11
C	140	30

At $t = 140$

C_2 140-170

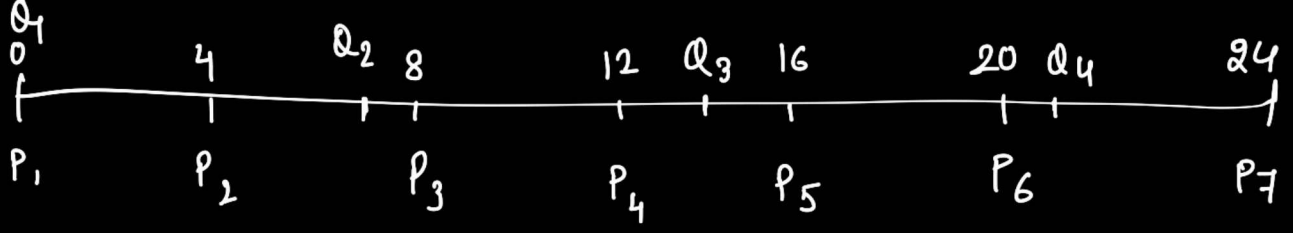
At $t = 170$ A_6
B_4

If considering B_4 due to less service time

If we do B_4 170-179 then

A_6 will miss the deadline
So prioritizing based on less service time is not good.

Job	Period	service
P	4	1
Q	7	5



P_1 0 - 1 P_2 6 - 7 P_3 cannot be done

Q_1 1 - 6 Q_2 7 - 12

Non feasible System

P_1 0 - 1 P_2 6 - 7 P_3 8 - 9 P_4 14 - 15

Q_1 1 - 6 Q_2 9 - 14 Q_3 Not possible

Job	Period	Service
P	40	5
Q	70	50

P_1 0 - 5 P_2 55 - 60 P_3 80 - 85 P_4 135 - 140 P_5 190 - 195

Q_1 5 - 55 Q_2 85 - 135 Q_3 140 - 190

Job	Period	Service	
U	20	12	60%
V	30	15	50%

Non feasible

Job	Period	service	
U	20	9	45%
V	30	14	47%

Main Memory Management

(1) Continuous Allocation
- Static Partition
  - Best fit available
  - Best fit only
- Dynamic Partition
  - first
  - Best
  - Worst

(2) Non-Continuous
- Paging
- Segmentation

Dynamic Partition

Let total memory be 100 (0-99)

K	J	I	H	G	F	E	D	C	B	A	Job
7	5	20	10	20	10	20	10	20	10	20	Sour
S	40	8	35	5	10	10	14	8	3	5	Memo Nee
1017	20-25	0-20	0-10	0-20	0-10	0-20	0-10	0-20	0-10	0-20	service
	0-39	90-97	55-89	50-54	40-49	30-39	16-29	8-15	5-7	0-4	Memory
			35		10		14		3		

Not enough memory to do it from 0-5, 10-15 as 40 memory full continuous memory is not available, its available only after 20 secs

K	L	M	N
7			
8	27	14	10
10-17			
First fit 16-23 Best 40-47 Worst 55-61			

First fit → 3, 6, 10, 35 (Remaining Memory Blocks)

Best fit → 3, 14, 2, 35

Worst fit → 3, 14, 10, 27

For L, M, N
27, 14, 10 → 3, 14, 2, 35

- Best fit → 0 8 No space for 10
- Worst fit → 3, 14, 10, 27 → 0 0 0 → done W.f is Best fit

- 35, 14 → Best fit is best (first fit fail)
- 35, 10, 6 → First fit is best (Best fit fail)

So neither is best, all can fail

- Memory Compaction

at $t=10$ G is shifted from 50-54 to 16-20

Now T is alloted 50-89

K
5
8
10-15
40-47 (let)

at $t=13$ Job K needs 2 more memory 40-47 was there 48-49 also empty then allocated

In case at $t=13$ Job K needs 5 more memory

(1) Terminate

(2) Wait till $t=20$

(3) Transfer

static Partitioning

K	J	I	H	G	F	E	D	C	B	A	Job
											Source
							9	23	10	18	Memory Needed
											service Int
								Q	V	R	Memory allocated
								17	0	2	Internal frag

Wait → .. .. only (0/ total)

P	5
Q	40
R	20
S	8
T	12
U	5
V	10

T → Best fit available

	D	E	F	
	15	35	9	
	Wait in all	wait	Best Avail	Best fully wait
IF	0	0	3	0
EF	30	0	0	5+8+5= 18

Total allotted = 23+10+18 = 51

Total = 100

Avai. = 99

* External Fragmentation → P+S+T+U > 15 But still 15+8+12+5 = 30 < 15 as (each part < 15)

we can't assign to D

* Internal Frag = 12+0+2 = 19

A Job will not feel EF if sum of ↑ Now allotted available parts

is less than the requirement (exp:- Job E)

30 < 35

EF is felt

IF is cost

Fragmentation in Dynamic Partitioning

Holes of size 3, 14, 10, 35

Memory needed & NO IF NO EF

	First	Best	Worst
Internal	0	0	0
External	0	0	0

* D·P rest memory could be used so no Internal fragmentation

Holes of Size : 3, 14, 10, 35

Memory Needed : 70 → no allocation

No IF_1, EF_2 as total 62 < 70

Memory Needed : 40

IF = 0

EF = 62

Job A needed 22 memory →

① Terminal

② Allocate another if available.

③ deny

Process Scheduling

Batch Processing

Ready time = arrival time

Job	Ready Time	Service Time	FCFS First come First serve Interval	Wait	SJF shortest Job first Interval	Wait
A	5	15	5-20	0	5-20	0
B	10	12	20-32	10	29-41	19
C	12	9	32-41	20	20-29	8
				30		27
C	12	4	32-36	20	5-20	0
				30	24-36	14
					20-24	8
						22

Job	Ready Time	Service Time	Shortest Remaining Time Next Interval	Wait	with zero swap time
A	5	15	5 – 12, 16 – 24	0+4	
B	10	12	24 – 36	14	
C	12	4	12 – 16	0	
C	12	4		18	

SRTN ≠

direct Memory Access

Swap-out 2 main memory ⟶ disk

Swap-in 1 disk ⟶ Main memory

Job	Ready Time	Service Time	SRTN with swap Interval	wait	
A	5	15	5–12, 19–27	7	12-14 Swapout A
B	10	12	27-39	17	18-19 Swap in A
C	12	4	14-18	2	
C	12	4		26	

Job	Ready	Service	Interval	wait
P	0	20	0-10, 22-32	12
Q	10	9	19-21	2

SRTN

Job	Ready	Servic	Interval	Wait
R	0	20	0-10, 17-27	7
S	10	4	12-16	2

FCFS

U	0	20	0-20	0
V	10	9	20-29	10
				10

Job A Run 10, needs P, run 15, needs Q, run 10

Job B Run 12, needs Q, run 20, needs R run 5

A [0-10] [10-25 P] [25-37 Wait] [37-47 PQ]

B [0-12] [12-32 Q] [32-37 QR]

PQR → Resource (Disk, Printer)

Job U Run 10, needs G, run 15, needs H, run 10

Job V Run 12, needs H, run 20, needs G, run 5

12-32 → Unsafe 32 → deadlock

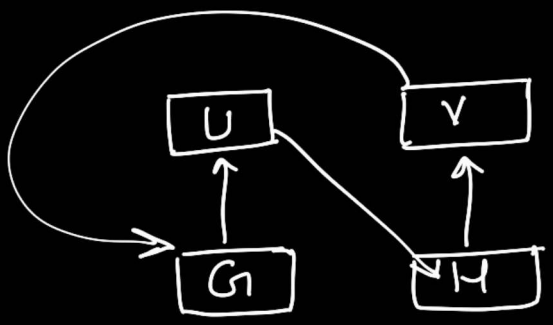
U : [0-10] [10-25 G] [25- Wait for H]

V : [0-12] [12-32 H] [32 - Wait for G]

Resource Allocation Graph * Let's not allow system to go in unsafe state

Job Y waits for Resource Z $Y \rightarrow Z$

Job Y holds resource Z $Z \rightarrow Y$



This is called Deadlock

Deadlock Prevention or Avoidance ((heck Name)

Static Method : Holding higher & Demanding lower not

Job U Run 10, needs G, run 15, needs H, run 10

Job V Run 12, needs G, needs H, run 20+5

G has more priority (higher resource)

U: [0-10] [10-25 G] [25-35 GH] just sort and do

V: [0-12] [12-35 Wait] [35-60 GH]

Dynamic Method (Deadlock Avoidance)

Deadlock Present

Deadlock Avoid

Job A Run 10, need P, run 15, need Q; run 10

Job B Run 12, need Q, run 20, need P. run 5

Job C Run 14, need Q, run 30, need K, run 12

A [0-10] [10-25 P] [25-56 wait for Q] [56-66 PQ]

B [0-12] [wait for Q through available] [66-86 Q] [86-91 PQ]

at 12 resource not given to avoid going to unsafe state

C [0-14] [14-44 Q] [44-56 QK]

Job A Run 10, need P, run 15, need Q; run 10

Job B Run 12, need Q, run 20, need P. run 5

Job C Run 14, need Q, run 30, need K, run 12

Job D Run 16, need K, run 10

↓

Job A Run 10, need P, run 15, need Q, run 10

Job B Run 12, need P, need Q, run 20+5

Job C Run 14, need K, need Q, run 30+12

Job D Run 16, need K, run 10

A: [0-10] [10-25 P] [25-56 wait] [56-66 Q]

B: [0-12] [wait for P] [12-66 wait] [66-91 PQ]

C: [0-14] [14-56 KQ]

D: [0-16] [16-56 wait] [56-66 K]

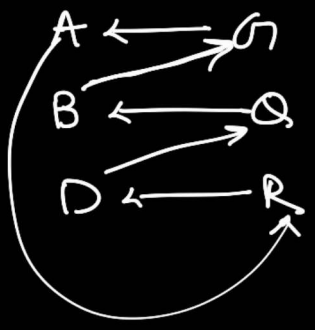
for K

Job	Got	Will need
A	U	R G
B	Q	G H
C	V	M T U
D	R	Q V G
E	M	G T

A ask for G

Let Give

Job	Got	Will need
A	U, G	R
B	Q	G
D	R	Q



Deadlock will take Place if G given to A

This is unsafe condition

D ask for G

Let give

Job	Got	Will need	
A	U	R, G	
B	Q	G, H	
C	V	M T U	C waits for E
D	R G	Q N	E waits for D
E	M	G T	D waits for C

B ask for G

Let give

Job	Got	will need
A	U	R, G
B	Q, G	M
C	V	M T U
D	R G	Q V
E	M	G T

B
E

Deadlock

Give G to B

Job	Got	will need
A	U	R, G
B	Q G	M
C	V	M T U
D	R	Q G
E	M	G T

Done
B
DE
A
C

. give G to A

Job	Got	will need
A	U G	R
B	Q	G M
C	V	M T U
D	R	G Q V
E	M	G T

A waiting for D (R)

D " " G (A)

Deadlock

Bank has 46 rupees

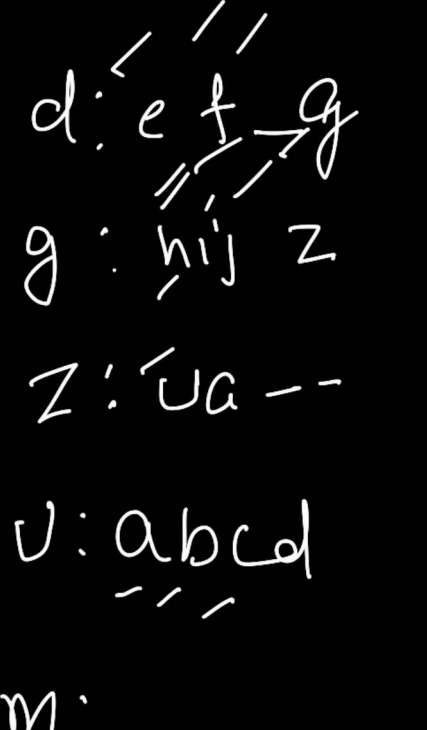
Person	Given	Will need
Hari	80	60
Gyan	40	50
Sunny	5	30

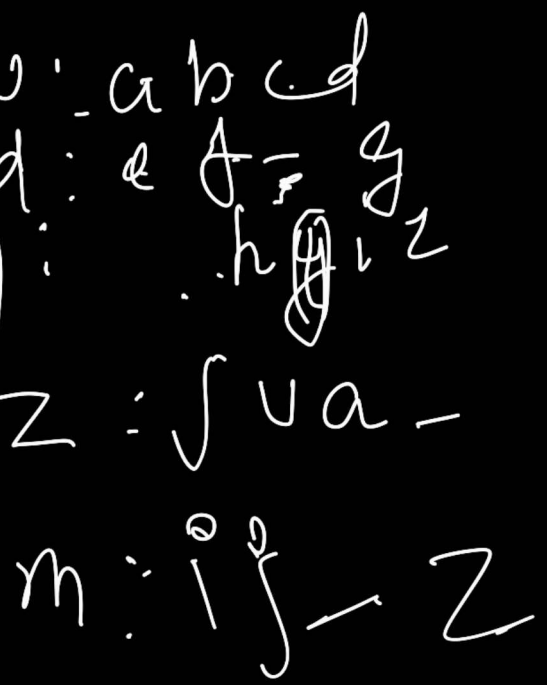
How much maximum money can be given to

Hari ⟶ 1 Max

Gyan ⟶ 16

Sunny ⟶





1 hour lecture missed
One 2 hour lecture not written

Data Compression

Most zeros (0)

Block size 4

data

0101,0000,0011

When complete 0 → 0

else 1<block>

1010 1 , 0 , 1 0011

0101 , 0000 , 0011

1010 1 , 0 , 1 0011

if 1 next 4 are considered

0 0 considered

0 1011 0011 1101

0000 0110 0000 1101

Block size 4

Complete 0 → 0

Single 1 (one) → 10 < addr. of 1 in 2bits >

more 1 → 11 < block >

0 1 0 1 , 0000 , 0 1 0 0 , 0 0 1 1

11 <0101> , 0 , 10 01 , 11 0011

Location of 1

1010 1 1 0 1 1 0 0 1 1 1 0 1

0010 0110 0000

Block size 4

Complete 0 : 00

single 1 : 01 < addr of 1 in 2 bits >

Two 1 : 10 < addr in 3 bits >

More 1 : 11 < block >

1 000 , 0 11 0 , 1001 , 1011

0100 10011 (1,2) i.e 3

address of 11 in combitorial number system

Gautam notes reference

0 ... 3 4places

2places needed

01 → 0
02 → 1
03 → 2
12 → 3
13 → 4
23 → 5

4, 5 both starts at 10x
6, 7 01 11000

11x

Data: 0101, 0001, 0101, 1101

Code: 10100, 0111, 10100, 0101

10101111101

lower one is used

10100111 10101111101

0101 0001 0101 1101

If only 4 is given to 13, then 10101 is not decodable

Block size 5

Complete 0 : 00

single 1 : 01 < addr 1 in 2 bits >

Two 1s : 10 < addr. in 4 bits >

More 1s : 11 < block >

After assigning both, further data compression can be done

01000 00101 00100

0101 101000 101111

01

means 1 at the fourth places

because for 2 1s condition

$^5C_2 = 10$ means 0-9 can only used so 1111

can be used for this purpose

Block size 5

Complete 0 : 00

Singles 1 : 01 < addr of 1 in 3 bits >

Two 1s : 10 < " >

exception
01101 → 00101 01110 → 00011

more 1 : 11 <blocks>

01000 , 00101 , 10000 ⟶ data

01001 , 01101 , 01110 ⟶ Code
or
01111

01	0
02	1
03	2
04	3
12	4
13	5
14	6
23	7
24	8
34	

this system does not use 5

<01 5>
<01 6>

Code

after 01 its 11 we don't need 01 3rd bit to know

01101 | 10111 | 01111 | 10001

00101 | 01001 | 00011 | 10100

For above method

single 1 : 01 <address in 3 or 2 bits>

two 1 : 01 <address in 3bits>

01 5 ⟶ 4th place 1 can be given 5 categ

01 6 ⟶ 8

01 7 ⟶ 9

0000 10
2

01 <10_>

Disk 6 digits

P : 421739 Q : 814263

R : 381436 S : 940629

P : abc Q : def R : ghi S : jkl

RAM 6 digits 4 bits

V x y z p q r s t u

V : 0 (invalid)

: 1 (Disk to RAM transfer)

x : 1 PQ is updated (1st Variable)

y : 1 2nd

z : 1 3rd Variable

O/P	Instruction	P 0000 123456	Q 0000 86 2413	R 0000 814213	S 0000 318613	Disk operation
		↓ Garbage				
81.	print(d)		1000 814263			Q: Disk → Ram 814263 D → R
	h = 83			0010 818313		NIL
	b = 12	0010 121256				
	print(g)			0010 388336		R: D → R 381436 D → R
	f = 69		1001 814269			Q: D → R 814263
	OVER	421739 D → R 421239 R → D	814269 R → D	388336 R → D		

Instruction	P 0000591216	Q 0000612339	R 0000818213	S 0000313814	
f=29		0001 612329			
i=61			0001 818 261		
print(a)	1000421739				P: D→R
print(e)		1001 8142 29			Q: D→R
b=19	1010 421939				
d=61		1101 614229			
g=69			0101698261		
Overall	421939 R→D	614229 R→D	381436 D→R 691461 R→D		

V=0 xyz=000 Nothing needs to be done

V=1 " " " " " "

xyz ≠ 000

V=1 R→D

V=0 & xyz ≠ 111 D→R ... R→D (make changes then)

V=0 & xyz = 111 then we don't need Read D→R we can directly update

0000 : 0 — 0 — invalid
0001 : 1 — 4 — {3}
0010 : 2 — 3 — {2}
0011 : 3 — 7 — {2,3}
0100 : 4 — 2 — {1}
0101 : 5 — 6 — {1,3}
0110 : 6 — 5 — {1,2}
0111 : 7 — 8 — {1,2,3}
1000 : 8 — 1 — valid
1001 : 7
1010 : 7
1101 : 7
⋮ : 7

mod 100

Instruction	P 0123456	Q 0421318	R 0312316	S 0818234
a = d+g	2193456	1814263	1381436	Q: D→R R: D→R
e = a+5		8812463		
h = a+5				3812434
j = (h++)+k			8381536	5381434
k = b−3	8191739			5381434 ↑: D→R
OVER	191739 R → D	812463 R → D	381536 R → D	940629 S: D→R 381429 R → D

Instruction	P 0123456	Q 0421318	R 0312316	S 0818234
a = d + g	2193456	1814263	1381436	Q: D→R R: D→R
e = a + 5		8812463		
k = a + 5				3812434
J = (h++) + k			8381536	8382429 S: D→R (Many updates are happ. so bcoz the second
k = k − 3	8191739		time to maintain consistency	5381434 P: D→R

3 Control bits

what is page size

000 : garbage

[001 : 101] partial update

110 : valid

111 : dirty

In this case only

0 0000 : ✓

4 0001 : ✓

3 0010 : ✓

2 0100 : ✓

8 0111 : ✓

1 : 1000 : ✓

P: abc Q: def R: ghi S: jkl

U: abcde V: fghij W: klmno

41361 81419 34579

	U 000 81429	V 000 31461	W 000 84423	Disk op.
print(g)		110 81419		V: D → R
m = 6 q = 9	001 91429		011 84623	
k: 7			111 74679	W: D → R
OVER	41361 D: R 91429 R → D	NIL	74679 W: R → D	

Round Robin Scheduling

A job is given a time. If not over then another job is given a time then previous job is put at the end of queue

RR Quantum 10

Job	Ready	Service	Interval
A	5	22	5-15, 25-35, 39-41
B	7	14	15-25, 35-39

Wait (A) 10+4=14

Wait (B) 8+10 : 18

Wait = Finish - ready - service

				Should be
				44-46
A	5	22	5 - 15	30 - 40
B	7	14	15 - 25	40 - 44
C	19	5	25 - 30	

at t:14 B7

t=15 A_2

t=19

MLFQ Multi level Feedback Queue

$A_1 B_1 A_2 C_1 B_2 A_3$

Interval Under RR

5 - 15, 25 - 35, 44 - 46

15 - 25, 40 - 44

35 - 40

MLFQ

A job is given a time. If not over then quantum job is given a time and previous job is put at the end of low priority queue.

RR disadvantage

Job	ready	service	FCFS	wait	RR	wait
U	0	50	0-50	0	0-10...80-90	40
V	0	50	50-100	50	10-20...90-100	50
				50		90

No advan.

(1) Low & High Priority Queue

(2) Less Quantum in High Priority

Quantum 5/20

Job	Arrival	Service		
A	0	60	0-5	10-30, 40-60, 85-100
B	3	5	5-10	
C	14	30	30-35, 60-80, 100-105	
D	17	10	35-40, 80-85	

Wt: 100-0-60

10-3-5

105-14-30

85-17-10

40+2+61+58 = 191-30 = 161

(3) High Priority job can preempt low priority Job

Job	Arrival	Quantum Service
A	0	60
B	3	5
C	14	30
D	17	10

5/20

0-5, 10-14, 24-44,

5-10

14-19, 44-64, 89-94

19-24, 64-69

94-106

69-89

105 - 0 - 60 = 45

10 - 3 - 5 = 0

94 - 14 - 30 = 50

69 - 17 - 10 = 42

139

time left

Switching time = 5

Job	Service	Ready	FCFS	Wait	SRTN	Context Switch
A	50	0	0-50	0	0-10 20-60	5 10
B	5	10	50-55	40	10-15	0

large diff in wait time

			W	W
C	100	55	0	5
			0	5
D	100	155		5
			0	5
E	100	255		
		⋮		

Static Partition

Parts of size — U V W X
50, 40, 30, 20

Total 140

Job	Need	Given	IF	EF
A	26	W	04	0
B	48	U	02	0
C	46	Nil		40+20 = 60
D	25	← Wait	15	0
E	20	X	0	0
F	49		0	0

26+48 = 74 allocated

04+02 : Wasted

Best fit only Available

Job	Need	IF	EF	
P	205	60	0	→ Static Partition Some part 265
Q	240	100	0	Best fit available 240 < 265

Job	Need	IF	EF	
P	205	60	0	→ Static Partition Some part 265
Q	240	100	0	Best fit available 240 < 265
R	2000	10	0	some part 2010
S	1800	0	2100	free parts sum 2100 each free < 1800
T	400	50	0	free part sum = 2100 - 450 = 1650
U	1700	0	0	→ free part sum < 1700
V	1400	0	1650	or IF ≤ 250 EF 0

Job	Ready	Service	Interval No preemption	Preemption
A	6	60	6-66	6-20, 45-91 Work Overtime Increased by 15
B	20	10	66-76	30-40
C				
D				

Preemption increase in wait = Service time of new job + Swap out + Swap In

No preempt

Increase in wait of new Job = Remaining Service time − Swap out time

Increase in Work over time = Swap in + Swap out → leads to → Increase in waiting time of C.D (other jobs)

Work Over time has increased

SCAN	Look	wait	FCFS	Job	Ready	Location	Shortest seek time first	wait
40	45	40	45	A	5	40	45	40
166		11	51	B	40	34	6	27
42	47	19	59	C	40	42	47	07
48	53	85	65	D	40	48	53	13
95	100	95+72	112	E	40	95	128	87

FAIR

Starvation Mostly decreases wait

CSCAN

40

134

42

48

95

No starvation

Relatively fair as compared to Scan

Round Robin

Time Quantum 100

Find Finish Time of Job and Wait

Job	Ready	Service
A	42	400000
B	61	400000

gf 400

A: 42-142, 242-342, 442-542, 642-742

B: 142-242, 342-442, 542-642, 742-842

FT = 400×2+42−100 = 742

W(A) = 800000 + 42 − 100

W(B) = 80000 + 42

FT = 400×2+42 / 842

Job	Given	Needed	Service	
A	Q	P	8	10-18 22-30
B	P		10	0-10 0-10
C	R	S	9	30-33 22-25
D	T	Q	6	18-24 30-36
E	S	P	12	18-30 10-22
F		R,T	20	33-53 36-56

Job	Given	Needed	Service	
G	U,V		5	0-5 begin
H	Y	U	10	5-15 parbegin Cobegin
I	Z	V	20	5-25 I
J		Y,Z	8	25-33 parend Coend

Using Join

```
    Join
    D=2
    G
    Create K
    H
    goto M
K:  I
M:  Join (D)
    J
```

↳ timings not known

```
G
Create K
H
stop
K: I
   J
   stop
```

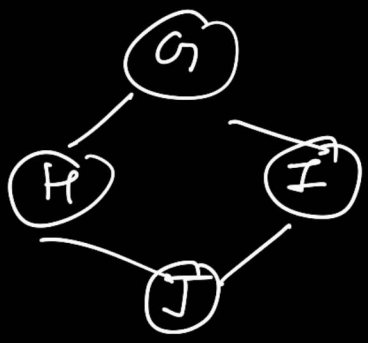
→ Timings know

Let's say H service time is 100

then J will be reached first and it won't work

Join k ⟶ k = k-1

if k ≠ 0 kill yourself

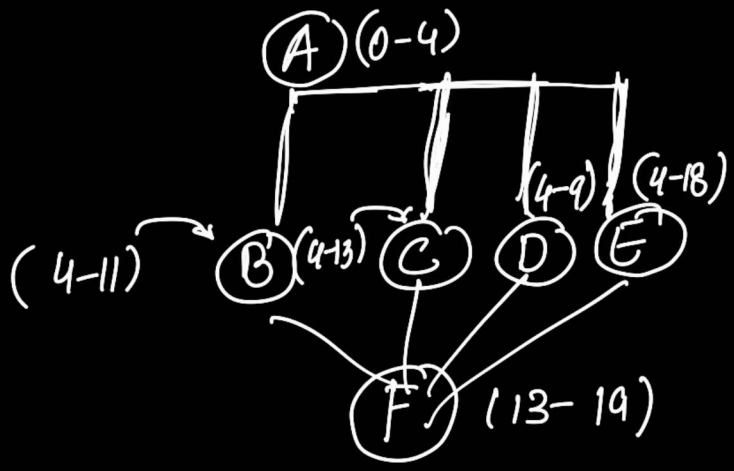



```
U = 3
A
Create P, Q, R
   B
   goto S
P: C
   goto S
Q: D
   goto S
R: E
S: Join(U)
   F
```

Service time

A	B	C	D	E	F
4	7	9	5	14	6

when is each done

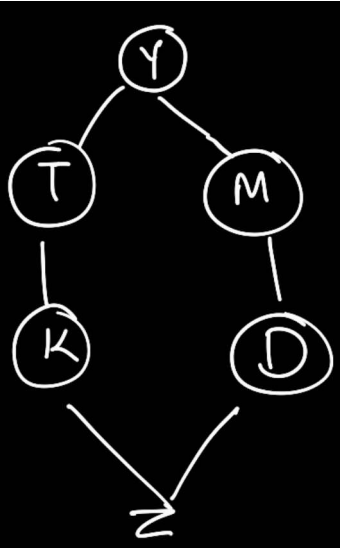


(4-9) D → U = 2

(4-11) B → U = 1

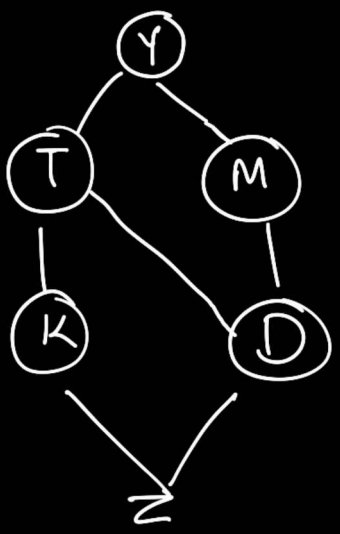
(4-13) C → U = 0 → F (13-19)

if U = 4 (18-24)




```
U:=2
Y
Create P
T
K
goto S
P: M
   D
S: Join(U)
Z
```

```
begin
  Y
  parbegin
    begin
      T
      K
    end
    begin
      M
      D
    end
  parend
  Z
end.
```



28 Feb. 2023

UNIX File System

Small file — directly stored in RAM

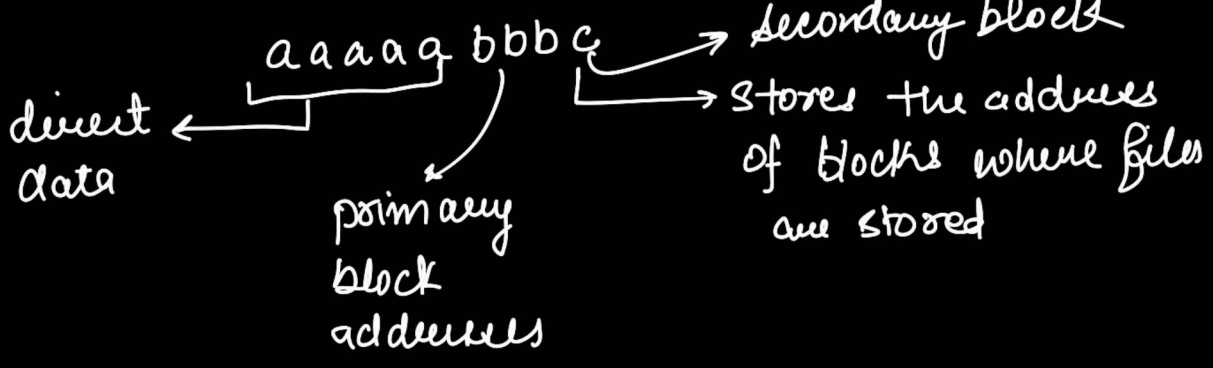
Medium — Disk → RAM

Very big file — Contents: Disk; Addresses (very large): Disk

Addresses of Addresses

RAM

i-node index Node



every block can store 4 letters

a.c g m u t z p e

g m u t z t k y g h m

5 + 3 × 4 + 1 × 4 × 4 = 33

a	i	q vlzt
b	j	r
c	K fkcd	s yzhi
d	L mRhi	t Rh
e hm	m	U
f	n	V glqt
g abde	o hmvlk	W
h	p tkyg	X

free block list ~~p~~ ~~p~~ k o g l q t v b f j m r c a h m u x d i w

U.txt file

h i y d j f k c d h m u l k y z h i a b d e m R h i v l z t R h

h i y d j k o s v

k o s g l q t

v

gmotzekyghm

delete y (8th letter) from a.c
imp

a.c gmutzpe

5 → 6-9 → 10-13

in table
↑
p → tk~~yg~~ tkg

Now if we delete 10th but it doesnot keep track of deleted letters

* Every block should be atleast 50% filled

gmotzekyghm

delete y (8th letter, 9,10) from a.c
imp

a.c gmutzpe

5 → 6-9 → 10-13

in table
↑
p → tk~~yg~~ tk~~g~~

e → ~~k~~m
↓
less than 50% full

transfer it to next overflow

a.c gmutzp~~e~~

p → tkm

e → will be free

e ~~p~~ ~~k~~ ~~m~~ ~~o~~ ~~g~~ ~~l~~ ~~q~~ ~~t~~ ~~y~~ bfjm rc ahn uxdiw

in u.txt insert 10th letter a m

free block list ~~p e r o g l q t~~ y b f j m r c a h n u x d i w

u.txt file h i y d j f k c d h m u l l y z h i a b d e m k h i v l z t k h

h i y d j k o s v

k o g l q t

v

5 + 3×4 = 33 → very close to 33 so m cannot be inserted
+1×4×4

Now do

5 + 3×4 + 2×4×4 =

0: n h m u k → 0: n h m
e: u k

h i y d j k o e v b

v: s g e

b: q t

free block list ~~p e r o g l q t~~ y b f j m r c a h n u x d i w

free block list ~~x p e r o g l q t~~ y b f j m r c a h n u x d i w

h . p y u d k _ _ f j _ m _

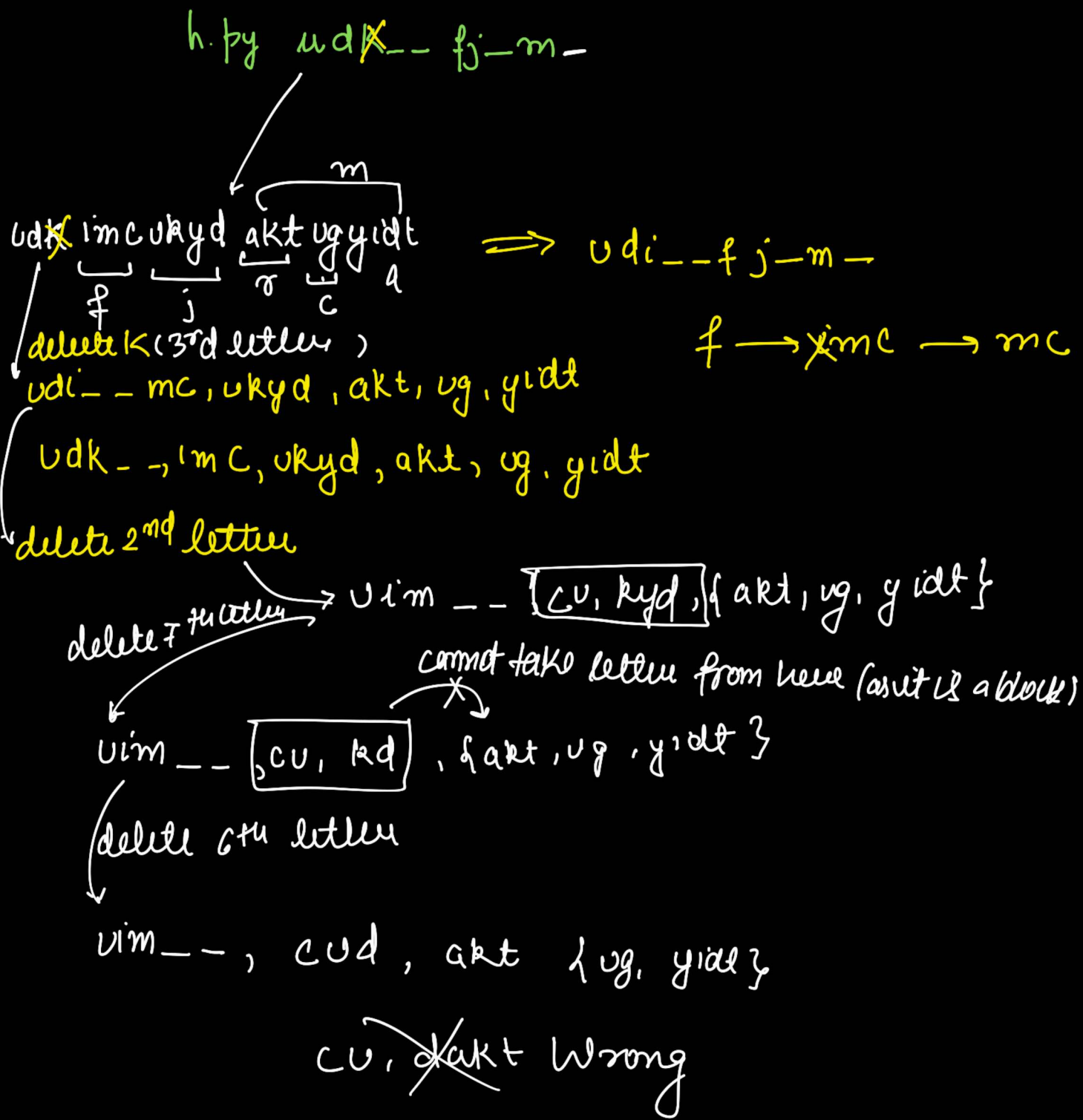
u d k i m c u k y d a k t u g y i d t

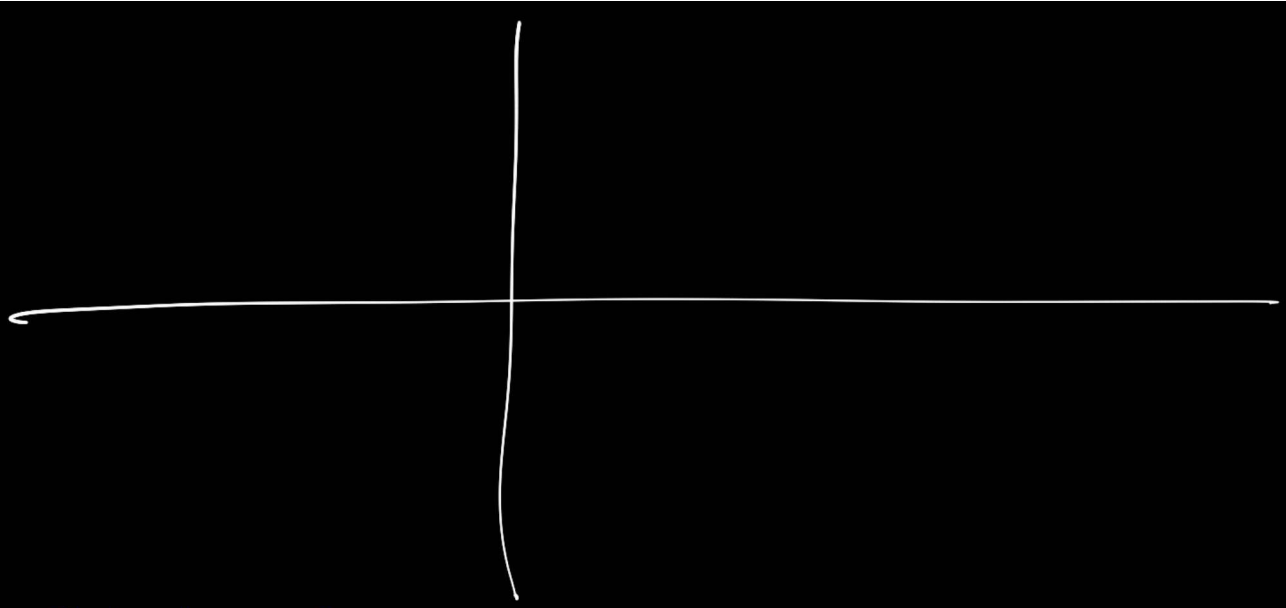
f j r c a

m

it should also be >50%.
So if we delete k

a	y i d t	i		q	v l z t
b		j	u k y d	u	a k t
c	u g	K	f k c d	s	y z h i
d		L	m k h i	t	k h
e	~~k~~ m	m	r c a	U	
f	i m c	n		V	g l q t
g	a b d e	o	h m u l k	w	
h		p	~~t~~ k y g r m	X	


h.py udk__fj_m_
m
udk imcukyd akt ugyidt
f j r c a
udi__fj_m_
delete k (3rd letter)
udi__mc, ukyd, akt, ug, yidt
f → imc → mc
udk__, imc, ukyd, akt, ug, yidt
delete 2nd letter
delete 7th letter
uim__ {cu, kyd}, {akt, ug, yidt}
cannot take letter from here (as it is a block)
uim__ {cu, kd}, {akt, ug, yidt}
delete 6th letter
uim__, cud, akt {ug, yidt}
cu, dakt Wrong



6 March, 2023 Assuming arrival time = 0

Job	Service	Interval		
		Direct Memory Access Interval	State	Programmed I/O
A	(40 run) (80 I/O) (90 run)			CPU 0 - 210 (0-40 Run) (40-120 Block) (120-210 run) 40-120 Disk State
B	(60 run)		210 - 270	CPU (0 - 210 ready) State (210-270 run)

When Program is under execution then other program can do its I/O → (Input / Output, Disk operation)

Program Driven

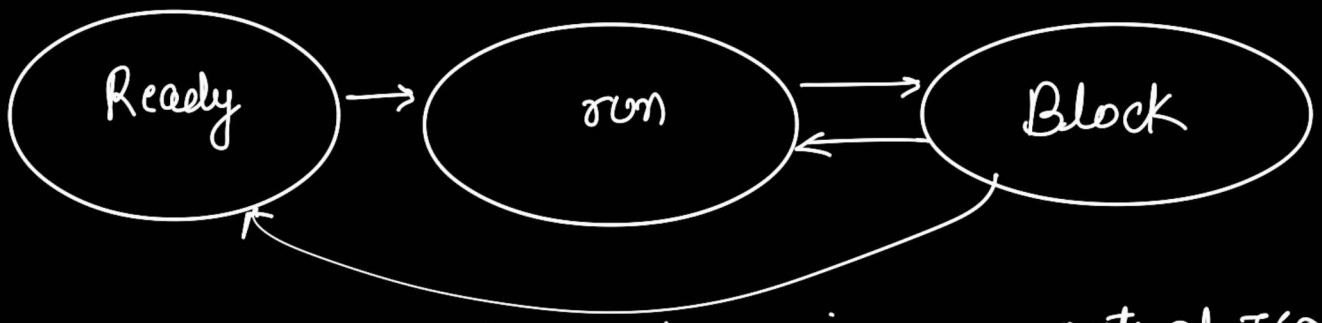
I/O : Input / Output on Disk

→ change the state from blocked to ready

Interrupt driven I/O (30 units to send request to disk
50 units to do the operation

Interval	State	Job	Service
CPU (0-(40+30))	(0-40 run)	A	(40 run) (I/O 80) (90 run)
Disk (70-120) Disk	(40-120 block)		
(130-220) CPU	(120-130 ready) (130-220 run)		
(70-130 CPU)	(0-70 ready) (70-130 run)		

State Transition Diagram



ID I/O → CPU is used during some parts of I/O (here for 30 secs)

whereas in PD I/O CPU is unused during complete I/O

Direct Memory Access (DMA)

Disk and CPU are independent units and they can do their work in parallel if needed

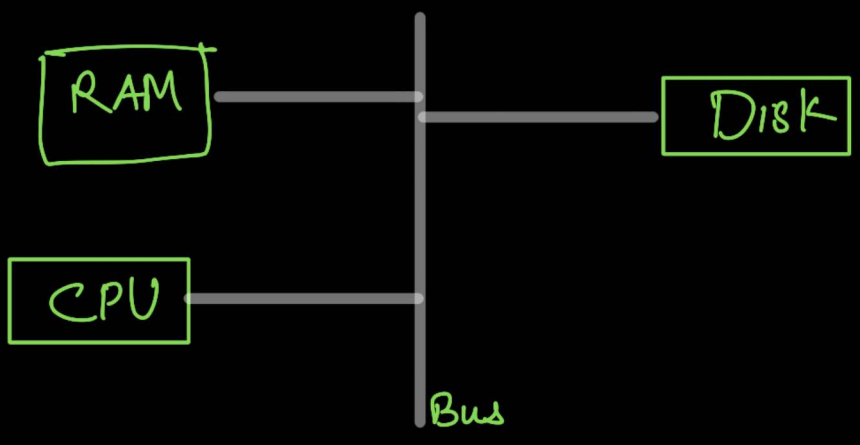
Job	Service	Interval		
		Direct Memory Access Interval	State	Programmed I/O
A	(40 run) (80 I/O) (90 run)	(0-40 CPU) (40-120) disk (120-210) CPU	(0-40 run) (40-120 block) (120-210 run)	CPU 0-210 40-120 state Disk (0-40 Run) (40-120 Block) (120-210 run)
B	(60 run)	(40-100 run)	(0-40 ready) (40-100 run)	210-270 CPU CPU (0-210 ready) state (210-270 run)

CPU (40-120) Pooling

Busy Waiting

Pooling → CPU constantly checking whether the work is over or not

ID I/O → CPU is waiting but it can do other work (and signal is send to CPU when work is done)



Cycle Stealing Mode → In DMA CPU operation and disk operations occur concurrently and CPU is given permission to use the bus, when CPU is not using it so Disk can use bus to do operations (robbery free)

In example of DMA (40-70) Disk is using Bus and CPU working (but not using Bus)

Swap in → 5 Swapout → 7
Total Memory 100 Shortest remaining time next

D)

Job	Arrival	Memory Need	Service	Interval
P	0	40	50	0-10, 30-70
Q	10	40	15	10-18, 23-30
R	18	40	5	18-23

As Memory is only 100

Swap Out	In
0-10	
30-70	
10-25	
25-30	

C

Job	Arrival	Memory	Service	Interval	Swap In	Swap Out
P	0	40	50	0-10 (M0-M39), (35-75)	18-25, (31-70)	30-35
Q	10	40	16	10-25 (M40-M79) (20-31)	10-26	
R	18	40	5	25-30 (M0-M39)	26-31	

as t = 25
RST(Q) = 1 <
RST(R) = 5
So No Need for swap

A

Job	Arrival	Memory	Service	Interval	Swap In	Swap Out
P	0	40	50	(0-10) M(0-39) (18-25) M(0-39) (30-35) (43-71)	Wait 21	
Q	10	60	16 / 10-18	10-18 M(40-99) (35-43) M(40-99)	(18-25) Wait 17	(30-35)
R	18	60	5	(25-30) M(40-99)	Wait 7	
					45	

Don't consider swap In/Out time during checking seeking time

B

Job	Arrival	Memory	Service	Interval	Swap In	Swap Out
U	0	60	50	(0-12) M(0-59) (59-97)/(29-67)	12-19	(24-29)
V	10	40	42/142	(12-19) M(60-99) (24-59)/(24-29)(67-97)		
W	12	60	5	(19-24) M(0-59)		

Now Reduce Avg waiting

25+5+7

8+20

Job	Arrival	Memory	Service	Interval	Swap In	Swap Out
P	0	40	50	0-10 (M0-M39) (35-75)	~~18-25~~ (31-70)	~~30-35~~
Q	10	40	16	10-25 (M40-M79)	10-26	
R	18	40	5	(20-31) 25-30 (M0-M39)	26-31	

Reduced Avg wait (As due to SwapIn/Out Increase wait)

Job	Arrival	Memory	Service	Interval	Wait
P	0	40	50	0-10 31-71	21
Q	10	60	16 10-18	10-26	0
R	18	60	5	26-31	8
					29 < 45

So waiting time reduced

Reduce Avg wait

Job	Arrival	Memory	Service	Interval	
P	0	80	50	0-10, 43-83	33
Q	10	80	30	10-18, 23-45	5
R	18	80	5	18-23	
					38

DO this PROP 488

7 March, 2023

Page Replacement Algorithms

1. FIFO
2. LRU
3. Optimal
4. Second chance
5. Approximate LRU

bit not the object
is set, remove ref
but if Reference bit
FIFO used
↑
everything same

Capacity is 4

removal upto first ↑

h removed
r → h

Need	FIFO	LRU	OPTIMAL	Implement LRU	Second chance	Reference at Time Quantum 3
G	G f	G f		G	Gr	Gr
M	GM f	GM f		GM	GrMr	GrMr
T	GMT f	GMT f		GMT	GrMrTr	GrMrTr → GhMhTh
M	GMT			GTM	GrMrTr	GhMrhTh
H	GMTH f	GMTH f		GTMH	GTMH	GrMrTrHr	GhMrhThHr
U	MTHU f	MTHU f	GMTU	TMHU	TMHU	MTHUr	MrhThHrUr
T				MHUT	TrMHU	MTrMUr	GhMrhHrUr
G	THUG f	THUG		HUTG	rHUTG	TrHUrGr	MhTrHhUh
M	HUGM f	TUGM		UTGM	UTGM	HUrGrTMr	MhTrHhUh
K	UGMK f	TKGM		TGMK	TGMK	GrTMrU	MH or U victim
						TMrUGK	

Victim : Removed element

Fault : Bringing click to — ram transfer (new object brought)

GrMrTrHr
MrTrUrG
TrHrGM
UrGMT
GMTHUr

rh > r > h > nil

FIFO 8 faults with capacity 4
6 faults with capacity 5

FIFOs

Optimal, LRU → Capacity Increase fault reduces
In some cases it may remain unchange
i.e. fault non-creasl

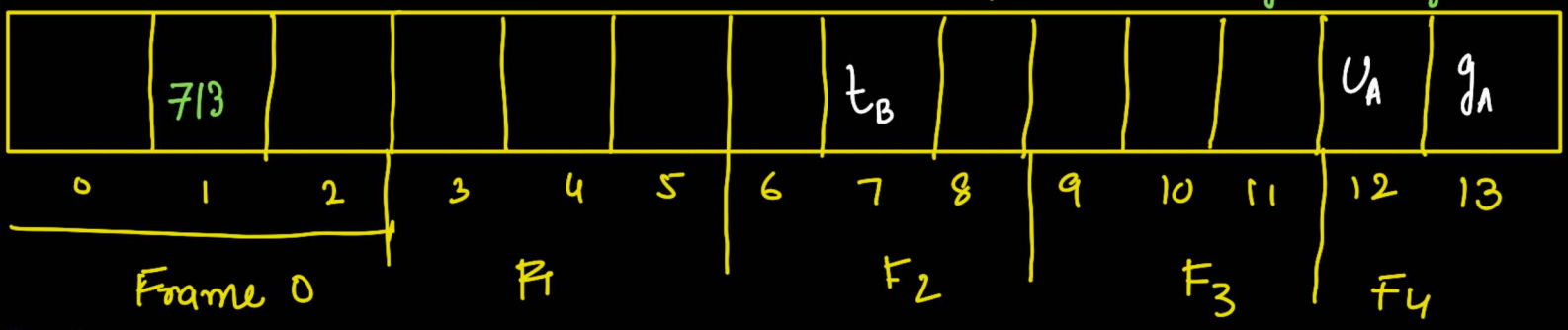
FIFO → Capacity Increase, fault increase
Bradely Anomoly

Q Give a smallest reference sequence so that first victim is
H in FIFO, U in LRU, T in optimal

Ans → H U H T I K H U 3 frames

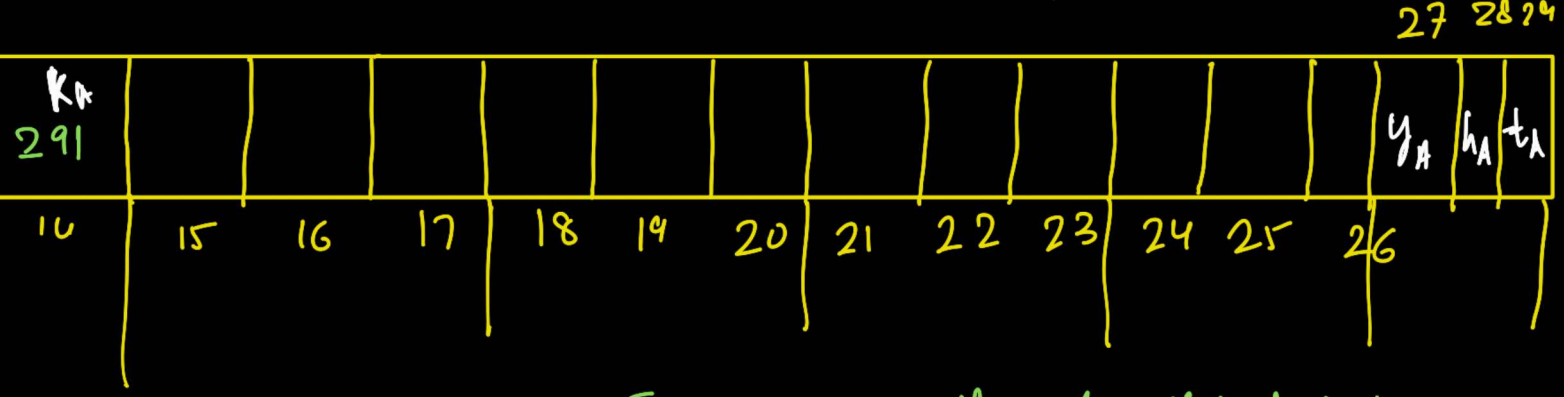
* 13 March, 2023

RAM → Inverted Page Table Maps Frames to (Program, Page)



PageSize : Frame Size → 3

Entire RAM is divided into parts



Frames needed = $\lceil \frac{10}{3} \rceil$

every page alloted a frame = 4

Virtual size = Virtual Address space : 10

main() Program A

{ int u, g, k, y, h, t, m, z, w, v

P_0 P_1 P_2

d / P_3

Page Fragm. = 4 × 3 − 10 = 2

Page table 4906

1 $P_0 \rightarrow F_4$

2 $P_1 \rightarrow F_2$

3 $P_2 \rightarrow F_0$

4 $P_3 \rightarrow F_6$

K = 291 → Variable z

z = 713

P_z = F0

z → 2nd vari

V_n P_r $r = n/3$, q = r^{th} entry of Page table

Physical location of V_n

V_7 $r = 7/3 = 2$ $q = 0$

Physical Address = $q \times 3 + n \bmod 3$ (offset)

A : 0 × 3 + 7 % 3 = 1

Program B Virtual Size = 8 Page Table = T28

int k, i, h, o, t, b, m, g

Frames Needed = $\lceil 8/3 \rceil = 3$

IF = PF = 3×3 − 8 = 1

Physical Address for t (V_4)

For V_4 Page no. = $\frac{4}{3} = 1$ Frame no = 2

Offset = 4 % 3 = 1

PA = 2×3 + 1 = 7

Program C Page Table 51

int i, d, h, y, z, d Page Frag = 0

Inverted Page Table

0	1	2	3	4	5	6	7	8	9
A_2	G	B_1	.	A_0	C_0	A_3	B_0	B_2	A_1

Physical Address 19 Frame no. = 19/3 = 6

d → V_9 r = 9/3 = 3

q = 6 PA = 6×3 + 0

= 18

So 19th is garbage

6th entry of IPT

A_3

Page 3 of Program A

3×3 + 19 % 3

= 10 V_{10} of A

(garbage) Currently in A → Not declared

Frame Size is a Power of 2 $= 2^K$ 2, 4, 8

Frame 0 F0000

00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	m 01010	01011	01100	01101
0	1	2	3	4	5	6	7	8	9	10	11	12	13

F000 F001 F010 P_1

Size 2 F(1111)

01110	01111	10000	10001	10010	10011	.....	28	29	30	31
14	15	16	17	18	19	20 ...				

F011 F100 P_0 F101 P_2 25 F110 F111

A: int u, g, k, y, h, t, M, z, W, d

P_0 P_1 P_0

P_0	P_1	P_1
F4	F2	F5
100	010	101

PageTable → 100010101

PageFrag → 2

* Physical Address of m V_6 V_{0110}

01 → 1 P_1 → F_2

9 bits: 10 001 010 1

P_0 P_1 P_1

((01)x3 to (01x3)+2)=pqr

0110 / 01010 → 01010

6/4 = 1st page

6%4 = 2 offset

1st Page = 2nd F

0110 Page 2x4+2 Frame 2

V.A abcd

Let pqr

(abx3) to (abx3)+2 → bits in page table

PA = pqrcd

B : e, j, b, G, r, i, p, q, a, l

P_0 P_1 P_2 P_3 P_4

F_{13} F_4 F_1 F_{14} F_5

Page Table : 1101 0100 0001 1110 0101 = 20 bits
0 1 2 3 4 5 6 7 8 9 10 11 (12)

$V_6 = P_3$ → F_{14} 1110 → 28 29 (p q)

Virtual 0110 → Page | → offset (011) × 4 = 12

pqrs = (abc × 4) to (abc × 4) + 3

11100 = 28

4 bits to store frame no.

pqrsd

i : 0101

(010 × 4) = 8

00011 = 3

Conline method

$V_5 \rightarrow \frac{5}{2} = 2^{nd}$ Page q = 1 (1 × 2) + (5 − 1.2)

= 3

Let 512 in RAM. Let a program has 29 Virtual spaces.

Page Size → 8

$\frac{512}{8}$ = 64 frames every frame address in 6 bits

$\lceil 29/8 \rceil$ = 4 Pages page Add in 2 bits

6 × 4 = 24 bits in page table

Page Table : 10011011000 10 10010110100

Physical for Virtual

abcde

01 → 11

01110 → 14 — 14/8 = 1st Page

(1×6) + (14%2) = 6

(1100001) → 49

49×8 + 6 = 398

(ab×6) to (ab×6) + 5 → pqrstu

6th to 13

1100001

pqrstucde

11000 01110 → 398

March 14, 2023

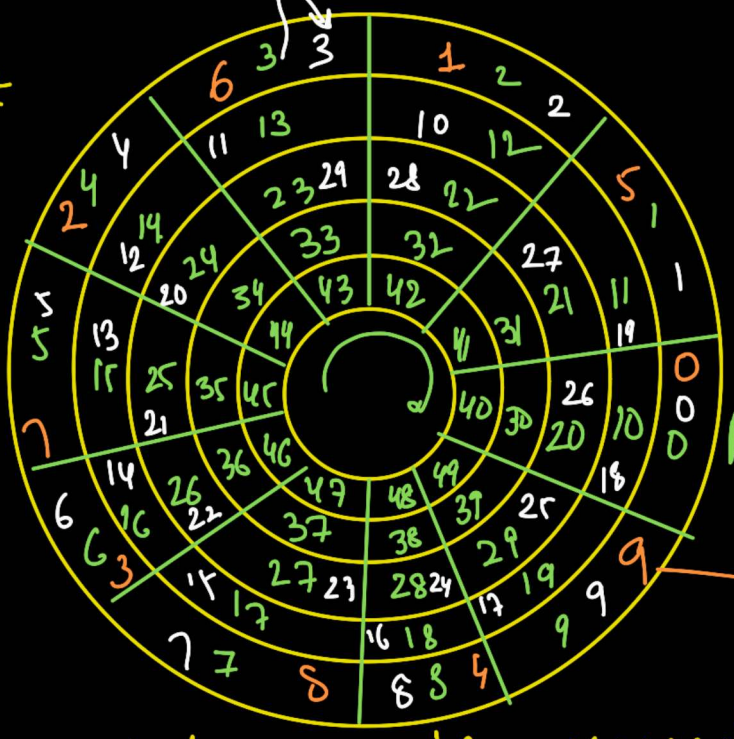
Logical numbering (shifting done by OS)
Hardware Numbering

1 unit / revolution

3 unit track head movement time)

Circular Disk

Always Rotating



At t=0 head is at block 0

How & when block 6 can be accessed?

At t=6 block 6 automatically comes under head

Block 17

Move head in left direction for unit distance

at t=3 head is at 13

Wait for 4 unit time head will be at 17

Total time = 7

(a,b) $\xrightarrow{\text{accessed at time}}$ b+10m

Block = 12 time taken = 12

move head unit left at t=3 head on block 13 wait for 9 secs

Access 29 then 10

9 + 11 = 20

Move head 2 units left

at t = 6 h on 26

wait 3 secs t = 9 h on 29

Move head unit right

at t : 12 h on 12

wait 8 secs

at t = 20 h on 10

Always multiple of 10

Access 10 ⟶ 29

unit left at t = 3 h on 13

wait for 7 secs t = 10 h on 10

unit left at = 13 h on 23

wait for 6 secs t = 19 h on 29

So always go in sorted order but this can be contradicted

example

Access 27, 11

27 ⟶ 6 + 1 + 3 + 1

= 11

Access 11, 27

3 + 8 + 3 + 3

= 17

This all comes under Comp. Architecture

Now Under OS

Access 0, 1, 2, 3, 4 ... 13

0 1 10 → 20

⋮

9 9 13 → (23)

seek time → position the head on correct track

Latency time → on correct block

(21) is also possible

0, 1, 2 ... 9, 12, 13, 14, 15 skipped (10, 11)

0 ... ... 9 12 13 14 15

Full Transfer → 0, 1, 2 → 20-29

0-9 10-19

read at t=0 transferred to ram takes 0.01s

now at t = 1.01s some info of 1 block is missed so we have to wait for one full rotation

Make 100 unit/revolution 2 unit for Disk → RAM

30 unit/track head movement-time

Logical Block No skew

File Access 0, 1, 2

0-9 20-29 40-49

5 units Disk to RAM

7 units RAM to disk

30 units on disk access

Job	Service	
A	(40 run) (h = 42 q) (30 run)	(0-40) CPU (40-54 → Disk) (54-84 Disk op.) (84-119 CPU) (54-89) CPU
B	(35 run)	

send h to disk two number send h & (26) ? → to refresh

Interrupt Driven IO

Job	Service	
P	(40 run) (print (m)) (30 run)	(0-40) CPU (40-47) (47-77 D.O) (82-87 D→RAM) (87-117 run) (47-82 run)
Q	(35 run)	

(Bring m) disk

Job	arrival	Service	
A	0	(40 run) (h = 42 q) (30 run	(0-54) RUN (54-84 block) (84-89 read)
B	15	35 run	(15-54 ready) (54-89 run)

When job P is executing P's critical return
then Job Q should not execute Q's critical return

Critical BC HI

Job P : output ABCD

Q GHIJ

$\frac{8!}{4!\,4!} = 70$

AGBHIC DJ not when critical

AGHBCDJ " "

ABGHIDJ → yes when critical
→ No when atomic

AGHBCJD yes always

UVW
KLM $\frac{6!}{3!\,3!} = 20$

VW, KL atomic (assume two as single) → in between V W nothing should come

VW, KL critical then possible

KL U V M W → something which is not critical can come

V U W ✓

V K W ✗ → critical

Mutual Exclusion (2 people should not be there at the same time)

Time Taken by instr.

↓ Initial F is 0

5 while (F == 1) Wait ; ⟶ someone using (inside)

3 F = 1;

Critical Section

7 F = 0

P: A(22) B(17) C(14) D(9)

Q: G(14) H(40) I(6) J(42)

0-22 A

22-27 while

⋮

72-77 while → Value of F is taken at the beginning

77-82 while

82-85 F=1

85-102 B

0-14 G

14-19 while

19-22 F=1

22-62 H

62-68 I

68-75 F=0 → Work is done at the end of interval

75-117 J

Replace G(14) by G(24)

0-22 A

22-27 while

27-30 F=1

0-24 G

24-29 while

29-32 F=1 → both happening together

Mutual exclusion Violated

One solution to do

F=1 before while loop

Test and set Solution (A special hardware for this was made)

```
F = 1
while (F == 1) wait ;
```

Time 10

U = TAS(F) → Take the value of flag and make it 1 and old value is returned

Critical section

F = 0

G1(24)

0-22 A

22-32 TAS

0-24 G1

~~24-34~~ TAS → Parallel TAS cannot be done

32-42 TAS

Initial G1, H = 0

Gyan	Hari
G1 = 1	H = 1
while (H == 1) wait	while (G1 == 1) wait
CRS	CRS
G1 = 0	H = 0