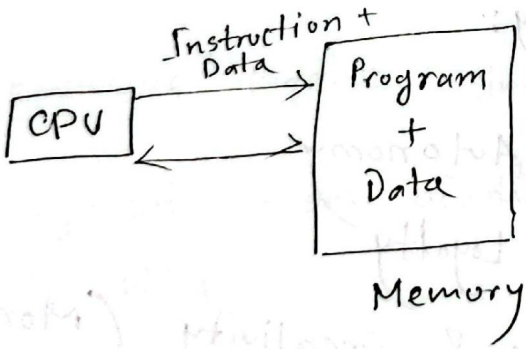
20/11/2024

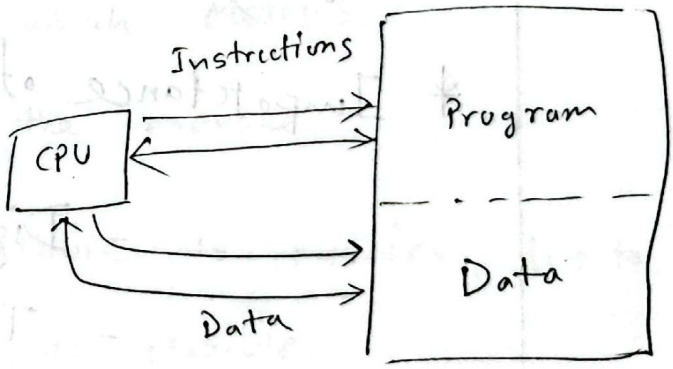
ESD

~~* How to start research~~

* Microprocessor vs Microcontroller
* CISC vs RISC
* Von Neumann vs Harvard
* Serial Peripherals I2C, UART, SPI



Von



Harvard

17/02/24

ESD

* To be Real-Time is ....

  services must provide upon request prior to a deadline.

  -A deadline is a constraint for processing

* Impact of a missed deadline as specified by design constraint.

  - loss of property, life, financial or system itself. (Hard Real time)

  - Soft Real time.

* Examples of Real Time systems.

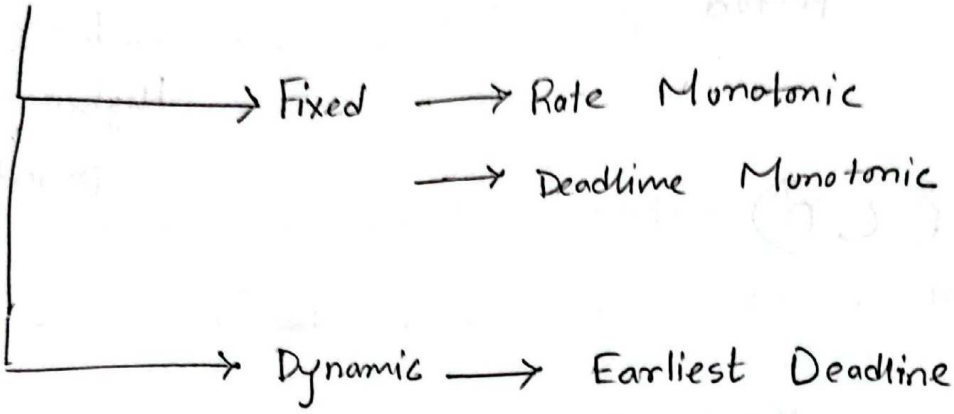
* Examples of RT embedded systems. RT must have deadline

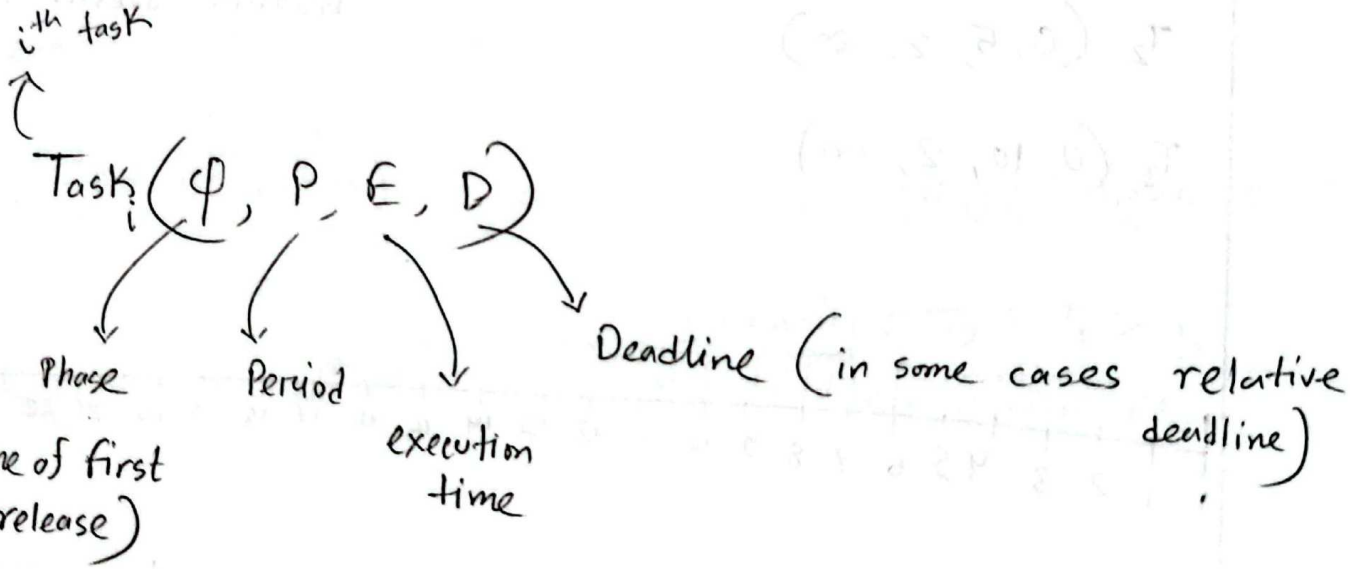
  ↓ Main goal

* Rate Monotonic Concepts.

  RMA saved the landing of Apollo 11

Priority Driven





T_3 $\phi = 13$ → it means that Task 3 starts at the 13^{th} time step.

* Rate Monotonic:

$$\text{Rate} = \frac{1}{\text{period}}$$

Higher the Rate or lower the period
↓
Higher the priority

$T_i\ (\Phi, P, E, D)$

$T_1\ (0, 20, 3, \infty)$

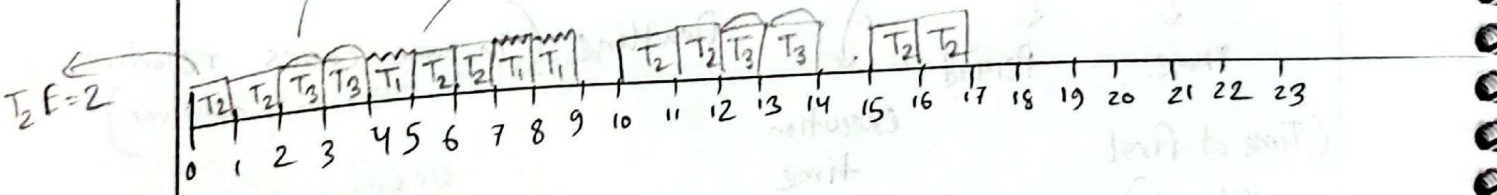
$T_2\ (0, 5, 2, \infty)$

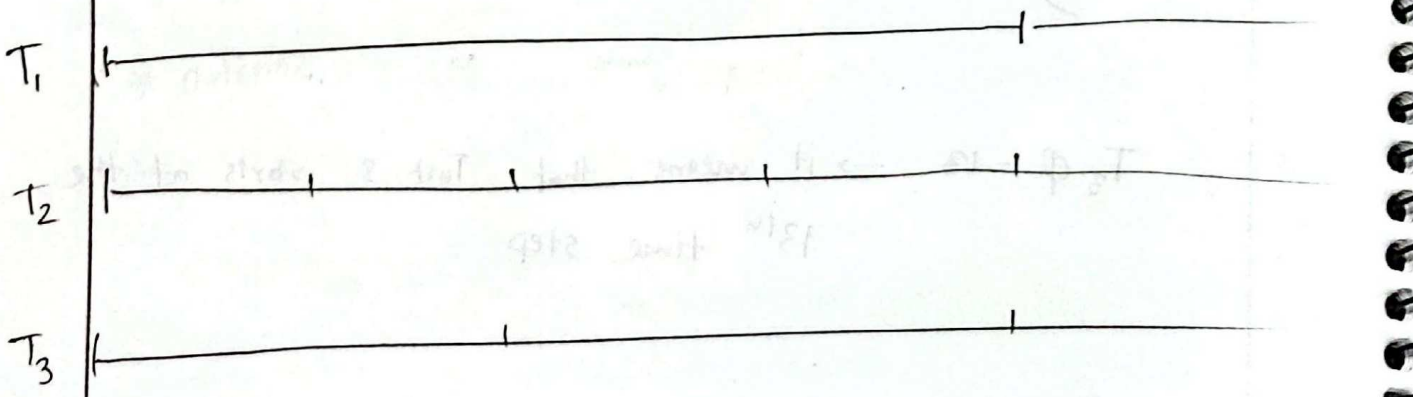
$T_3\ (0, 10, 2, \infty)$

in Rate Monotonic, Deadline doesn't matter.

priority: $T_2 > T_1$ So T_2 will overwrite T_1 Then T_1 will continue

At 5, T_2 appears again and





5 | 20, 5, 10
2 | 4, 1, 2
2, 1, 1

LCM (20, 5, 10) = 20

Execution time will always be lower than the deadline.

→ So, priority : $T_2 > T_3 > T_1$

* Execution time can be greatere than period time but that would not result into errors in the system.

It will fill the memories and spaces.

Deadline Monotonic

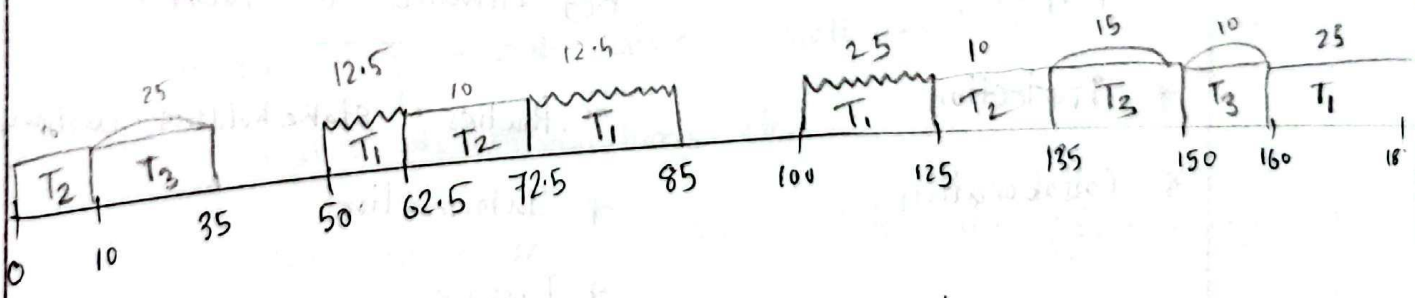
T_1 (50, 50, 25, 100) — from 50 to 150, it has to execute once at least

T_2 (0, 62.5, 10, 20)

T_3 (0, 125, 25, 50)

priority: ∄ D ↓ higher priority.

∴ priority : $T_2 > T_3 > T_1$



19/02/2024

EDD

* Dynamic ⟶ Earliest Deadline first

$T(\varphi, D, ERD)$

Absolute deadline = Relative deadline (RD) + (φ) Release time

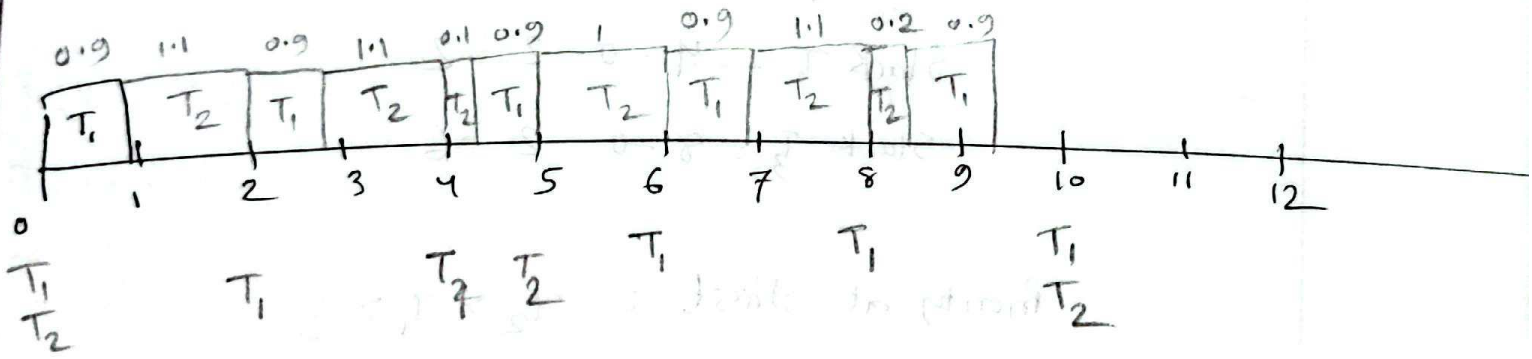
$T_1\ (0, 2, 0.9, 2)$

$T_2\ (0, 5, 2.3, 5)$

* whichever was running — should continue running when possible

$2.3 - 1.1 = 1.2$ $\quad 1.2 - 1.1 = 0.1$ $\quad 2.3 - 1 = 1.3$

$1.3 - 1.1 = 0.2$



T_1 T_2 $\quad T_1$ $\quad T_2$ T_2 $\quad T_1$ $\quad T_1$ $\quad T_1$ T_2

Priority: $T_1 > T_2$

But deadline must be finished before starting a new one

* LST:

Slack time ↓ Priority ↑

Slack time = AD − t − e′

t: current time step; e′: Remaining execution time

$T(\Phi, P, E, AD)$

$T_1(0, 20, 3, 7)$ $T_1(AD) = 7+0 = 7$ $t_1 = 0$ $e_1' = 3$

$T_2(0, 5, 2, 4)$ $T_2(AD) = 4+0 = 4$ $t_2 = 0$ $e_2' = 2$

$T_3(0, 10, 2, 8)$ $T_3(AD) = 8+0 = 8$ $t_3 = 0$ $e_3' = 2$

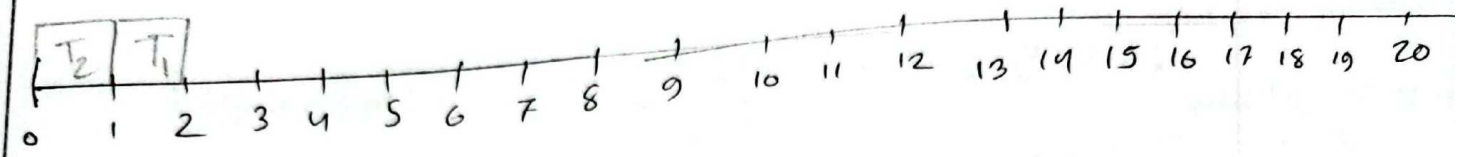
$\therefore$ Slack$_{T_1} = 7 - 0 - 3 = 4$

Slack $T_2 = 4 - 0 - 2 = 2$

Slack $T_3 = 8 - 0 - 2 = 6$

Priority at first : $T_2 > T_1 > T_3$

(Time step 0)



* if another instance of a task arrives before the first one arrives then

ie., T_{21} (some values) → first complete then T_{22}
T_{22} (some values)

The chart: Time step →

	0	1
T_1 (0, 20, 3, ~~6~~)	3	2
T_2 (0, 5, 2, ~~5~~)	3	3
T_3 (0, 10, 2, 8)	6	5

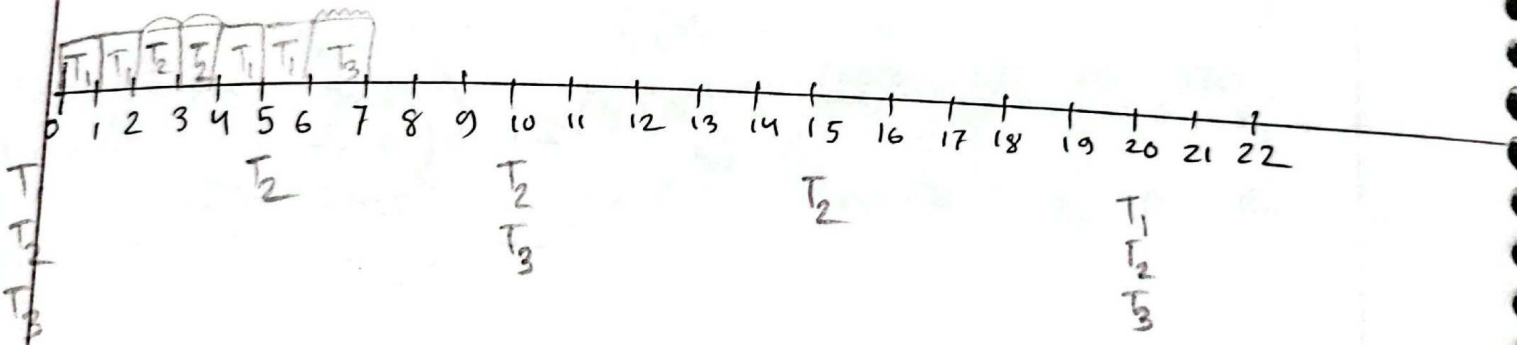
- we could've chose Either T_1 or T_2 at 0.
- But tiebreaker → Run the task that was running

Slack time ↓ priority ↑

Slack time

	0	1	2	3	4	5	6
$T_1(0, 20, 4, 6)$	2	2	2	1	0	(-1)	
$T_2(0, 5, 2, 5)$	3	2	1	(1)		3	2
$T_3(0, 10, 2, 8)$	6	5	4	3	2	1	0

↳ Done with execution



At 5, $T_2(AD) = 5 + 5 = 10$

$Slack = 10 - 5 - 2 = 3$

* if there's starvation is significant, discard the task.

It works for all of the algorithms.

24/02/2024

ESD Priority Inversion

critical section

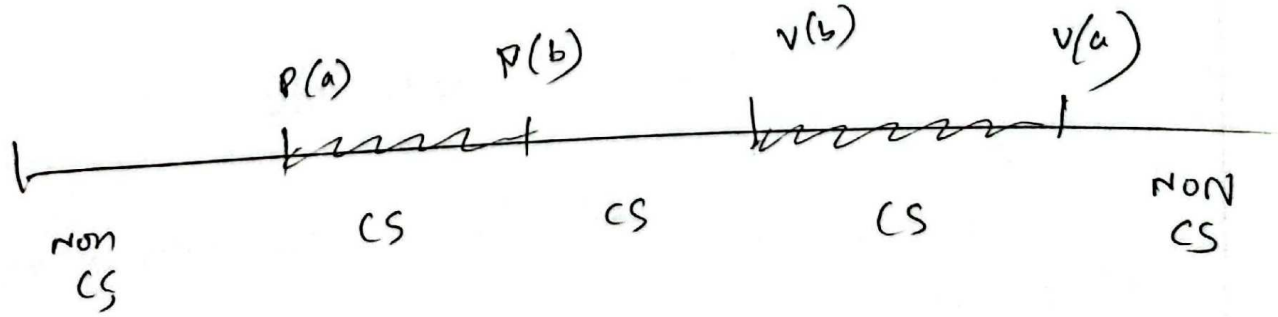
- Accessing the same resources. Then manipulate the ≠ same resource.

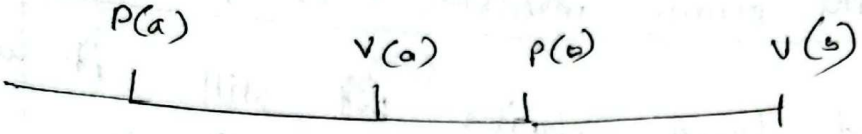
```
P0
{
  Read(A)
  A=A+1;
  write(A);
}

P1
{
  Read(A)
  A=A-1;
  write(A);
}
```

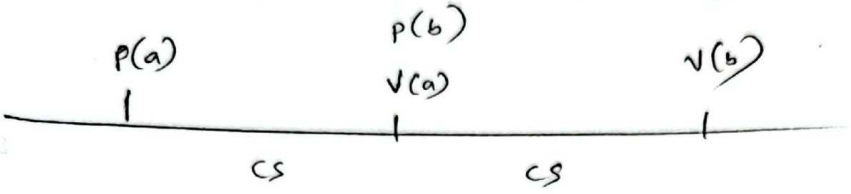
critical section

P = locking

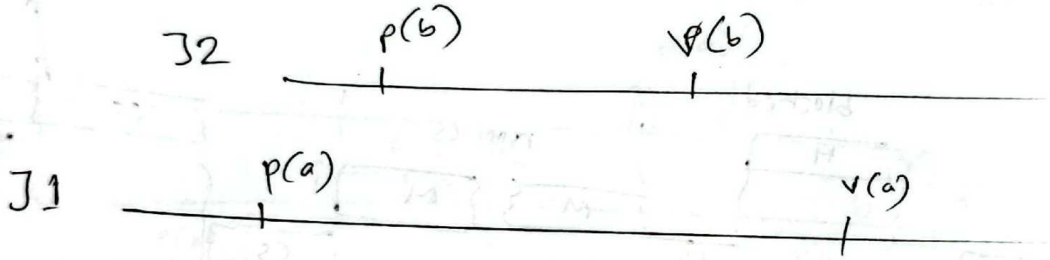
V = unlocking





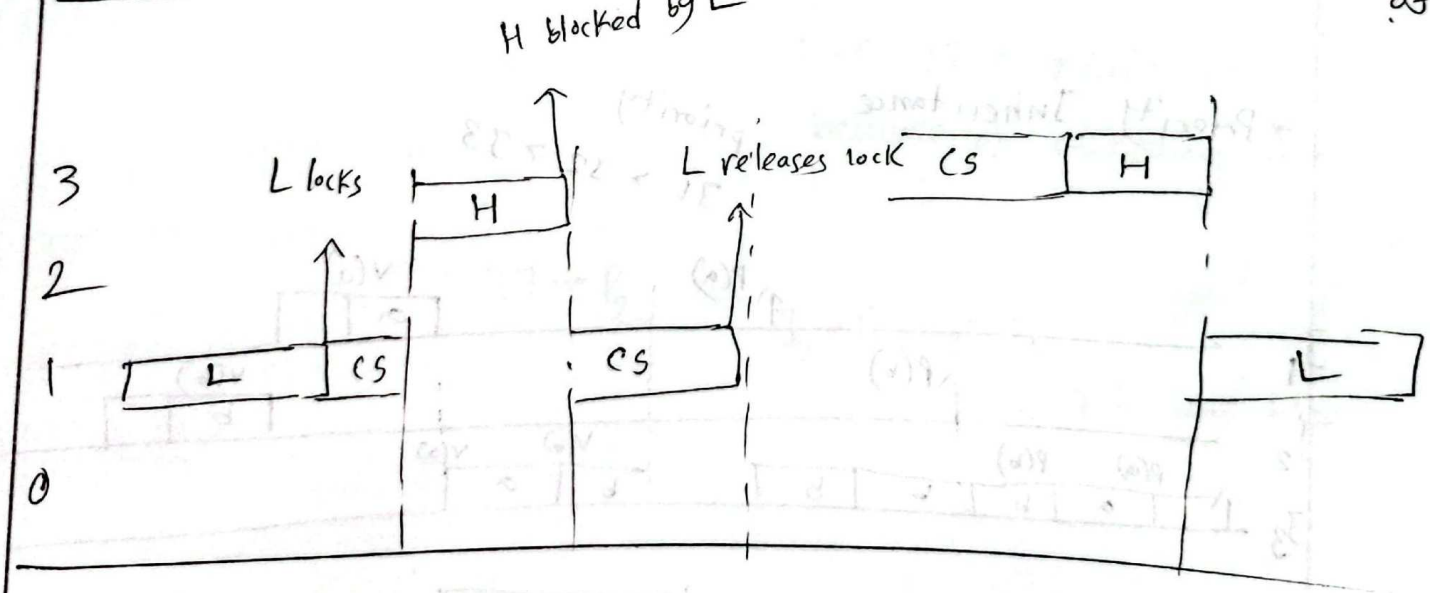


priority J2 > J1



if J2 needs (a) it can't get it beacuse J1 is in execution of (a).

example:

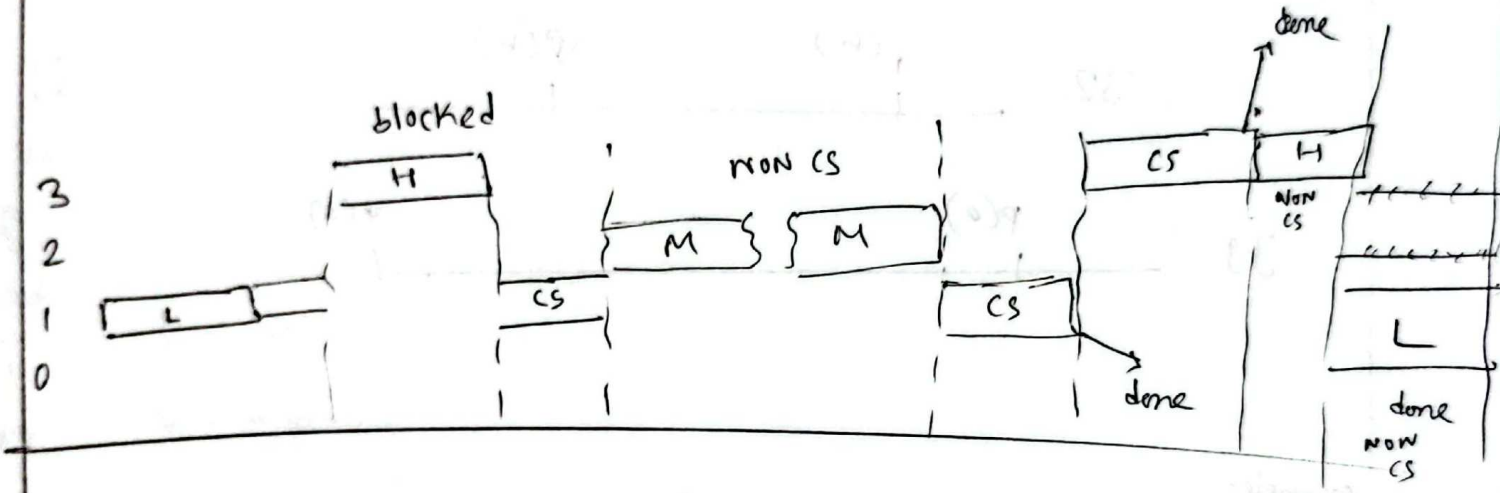


Here, in CS, H and L are trying to use the same resource.

where did the priority inversion happen?

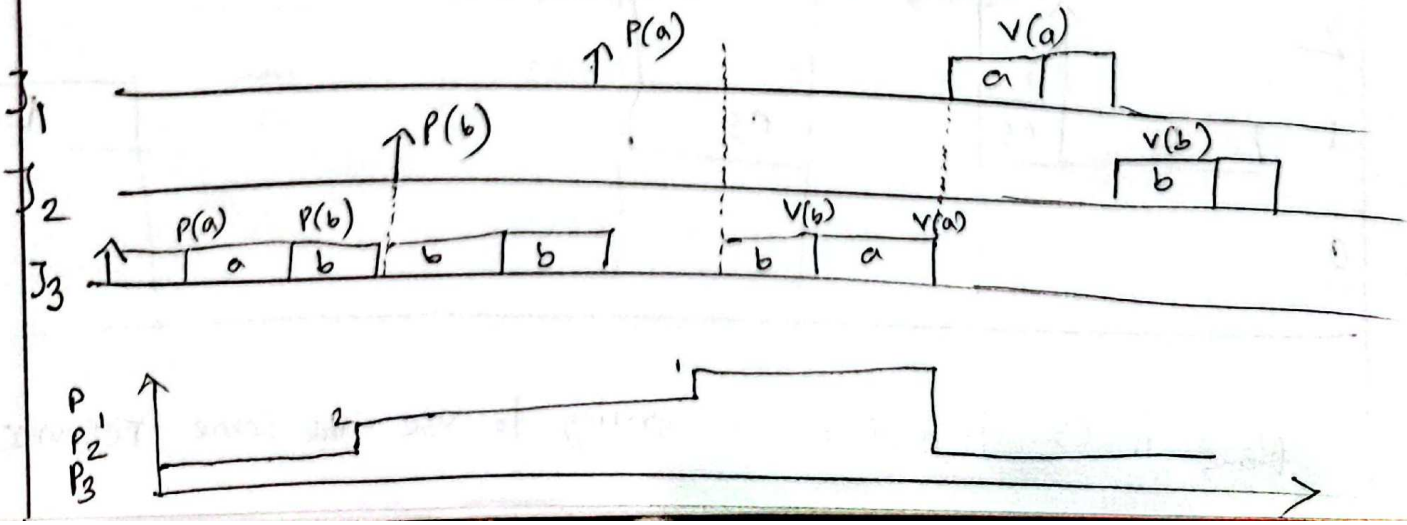
L had lower priority but still it was running its cs. even though H arrived.

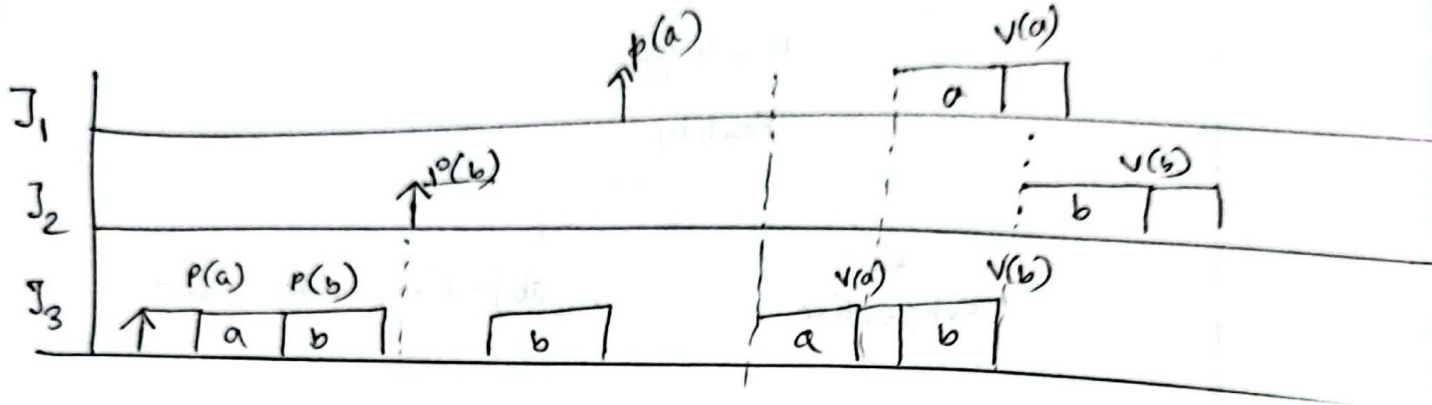
* priority: H > M > L



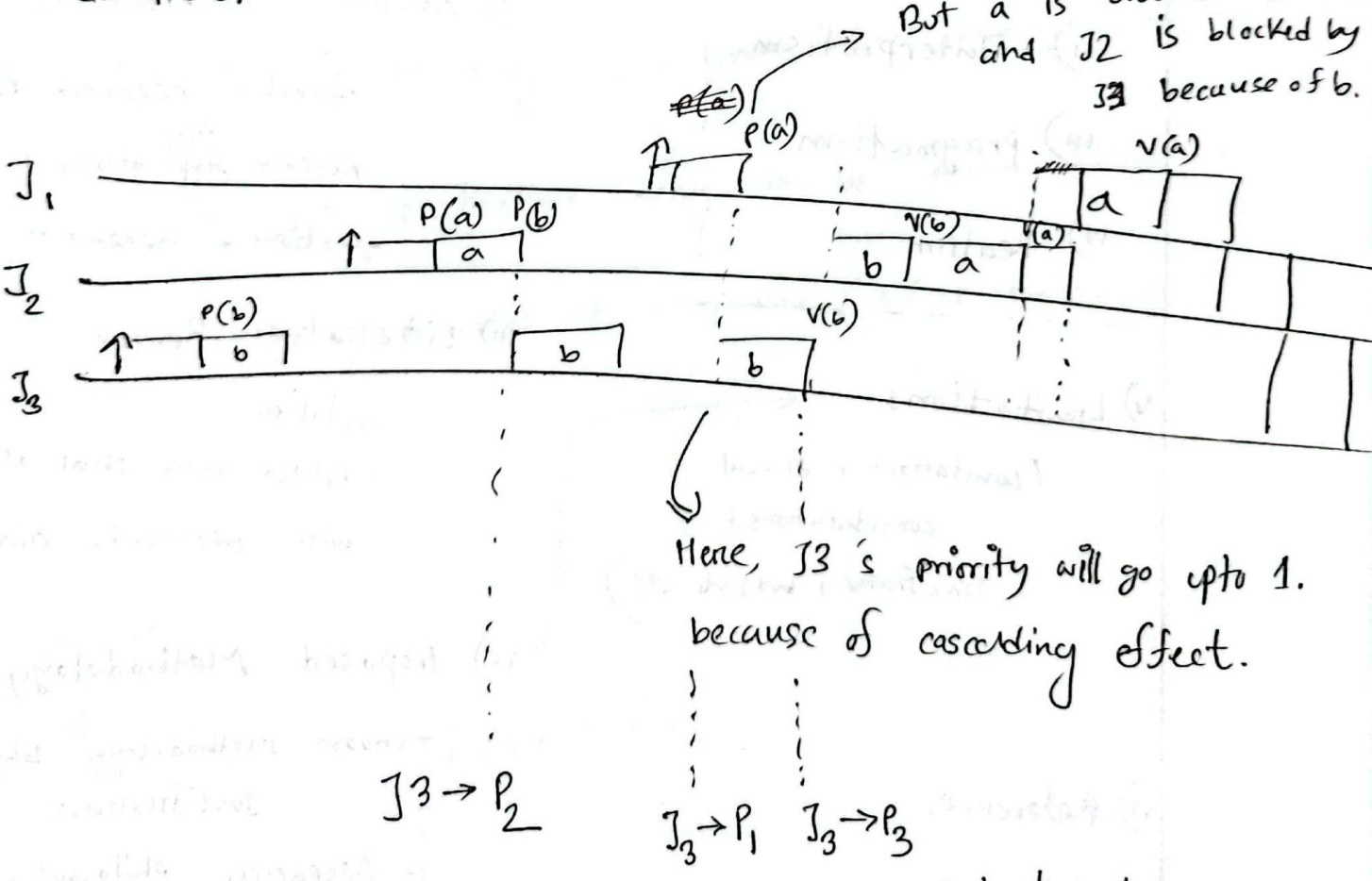
* Priority Inheritance

priority
J1 > J2 > J3




J1
J2
J3
P(a)
V(a)
a
P(b)
V(b)
b
P(a)
P(b)
a
b
V(a)
V(b)
a
b

example 2:

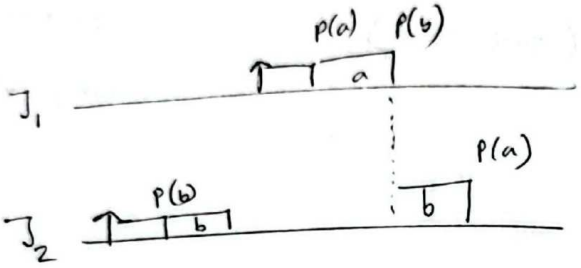

But a is blocked by J2 and J2 is blocked by J3 because of b.
P(a)
P(a)
V(a)
a
J1
P(a)
P(b)
a
V(b)
V(a)
b
a
J2
P(b)
b
b
V(b)
b
J3
Here, J3's priority will go upto 1. because of cascading effect.
J3 → P2
J3 → P1
J3 → P3
Since b is done here.

25/02/2024

ESD

priority Inheritance Deadlocks:

- circular dependency



Solution: Priority ceiliengg (it works before priority inheritance)

- The job specifications needs to be known before design time
- Assigning resources at design time. (Before simulation)

Resources

S	J (s) > S', S''
S'	J'
S''	J''

Conditions: i) assign priority ceiling ii) a resources has to be free
iii) then others jobs has to exceed the priority ceiling of that resource

priority: $J_1 > J_2 > J_3$

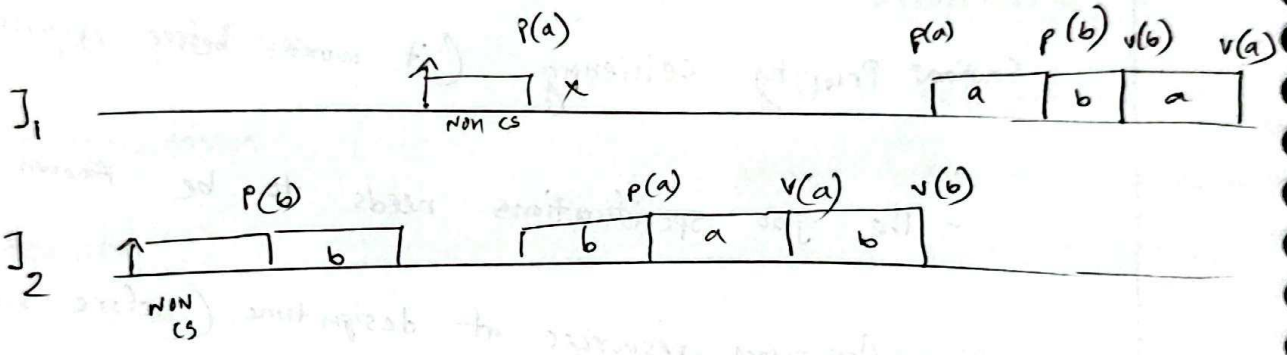
1) Assign priority ceiling.

$a \to 1$ (J_1, J_2 both want a, J_1 has higher priority. That's why value is 1)

$b \to 1$ (same as a)

$\approx \begin{bmatrix} a \to P_1 \\ b \to P_2 \end{bmatrix}$

Solution?



J_2

When J_1 was trying to access $\neq a$, even though it was free, other resources' jobs ceiling doesn't exceed "1".

↳ that resource being "b". since J_1 has to work with both a and b.

since $a \not> b$. so that's why the third condition doesn't satisfy

Examples:
one

$a \rightarrow 1$ (J_1, J_3 will access a) $J_1 > J_3$

$b \rightarrow 1$ (J_1, J_3 will access b) $J_1 > J_3$

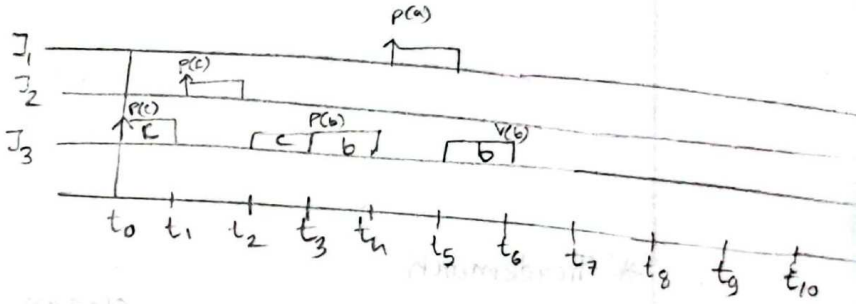
$c \rightarrow 2$ (J_2 will access c)

$\approx$ [$a \rightarrow P_1$, $b \rightarrow P_1$, $c \rightarrow P_2$]

$t_0 \rightarrow J_3$ (c) then b

$t_1 \rightarrow J_2$ (c)

$t_4 \rightarrow J_1$ (a) then b



Exam:

1) A scenario will be given. find the problem in the scenario and then fix it.

2)

26/02/2024

ESD

* types of times

  elapsed time
  specific delay
  timed out ——→ if deadline is not met
  deadline

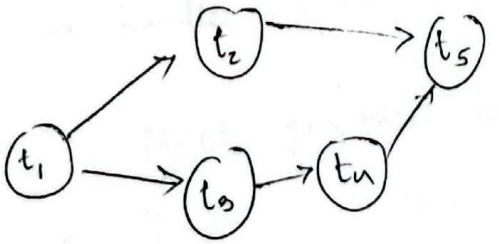
* States

  - State Behaviors
  - Events
  - Exceptions

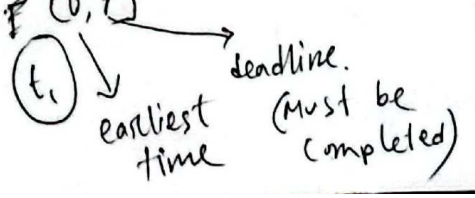
* Computational Model

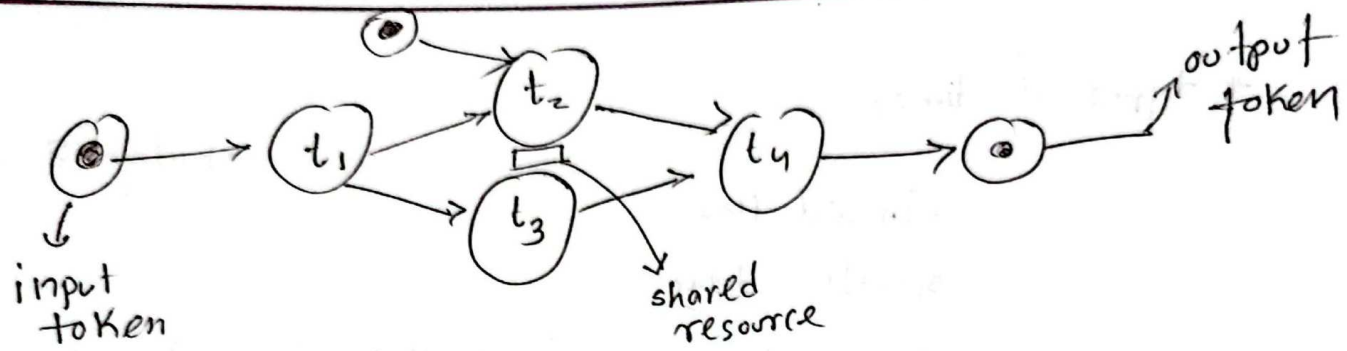
  - There's a formal way of expressing the model.

* Dependency graph



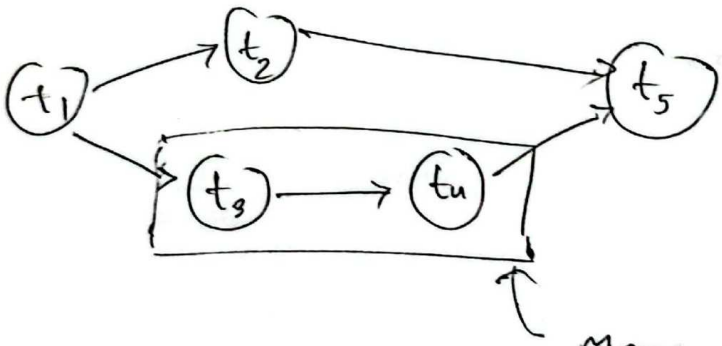
t_2, t_3 can work only if t_1 is done.





(t_1 needs an input token to start)

(t_2 needs an input token and t_1 to finish to start)



Means that t_3 and t_4 are on the same level.

28/02/2024

ESD

Modeling and Specifications

- Model & system
  - Structural
  - Behavioral / Functional

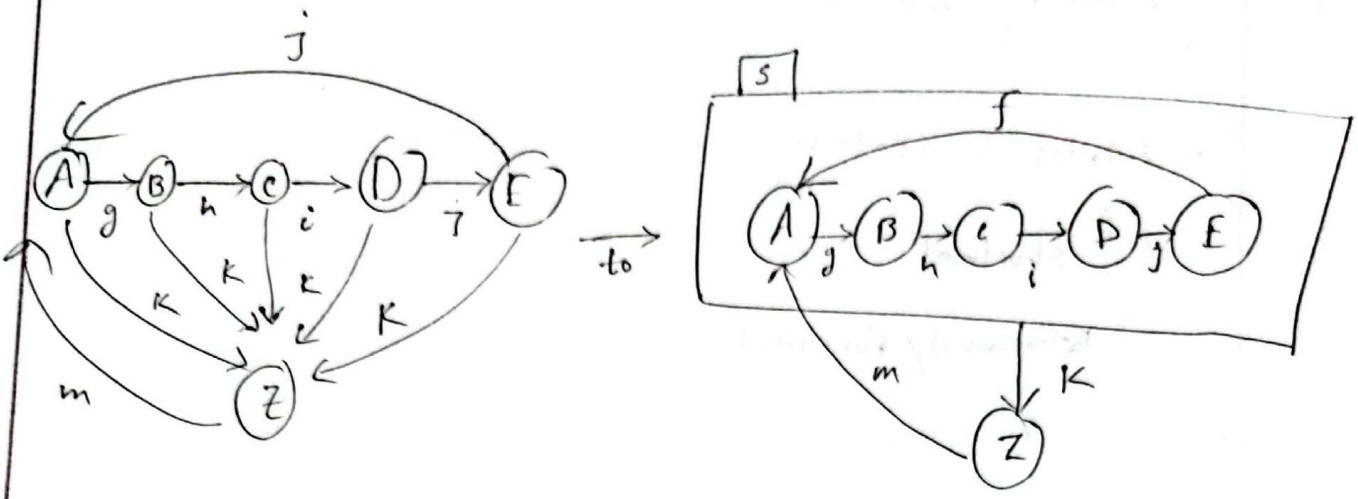
* Types of Communication
  - Based on Synchronization
    - Sync
    - async
  - Based on method
    - Shared memory
    - Message Passing

* FSM

* State Machine — Elevator example question.

* FSM - Statechart Hierarchy

How states can change based on events.



FSM

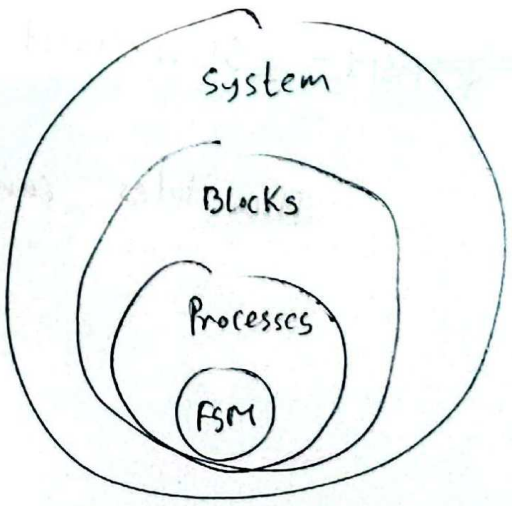
* Statechart (concurrency)

* ~~FSM~~ - Timed (Answering Machine)

{Timed Automata} <Answering Machine>

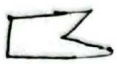
*SDL - Heirarchy

FSM are building Blocks of FSM




initial state


state


input


output

* SDL representation of FSM

- Declarations & Assignments of Variables in SDL

- SDL process interaction.

Block view

Alternative view
with corresponding queues

- SDL subsystems [Blocks inside Blocks]

- SDL timer

setting the timer: SET(now+T, Z)

if timesout ? then go to A right away.

if not, then reset(Z) then go to A.

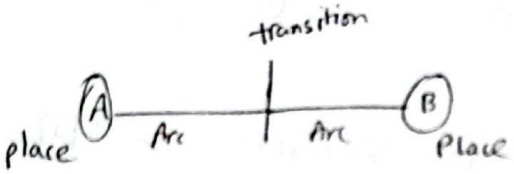
03/03/2024

ESD

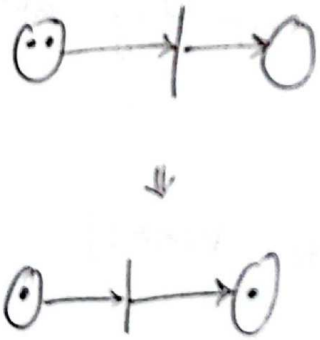
* Patri net

→ definitions

Places to places → transitions, Arcs



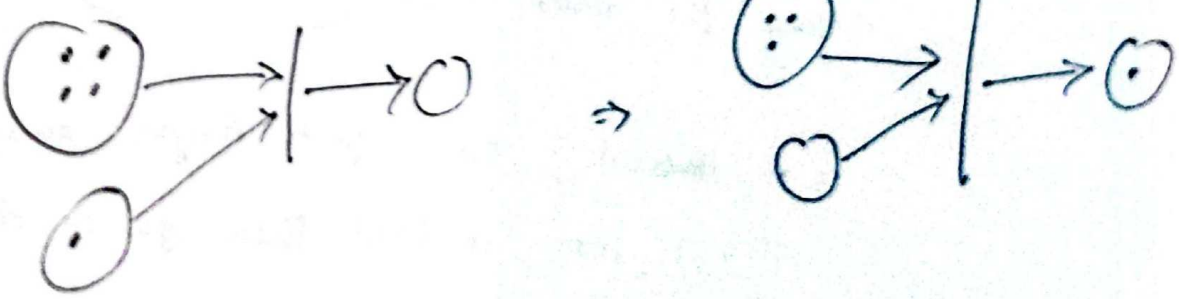
dots inside the places → tokens

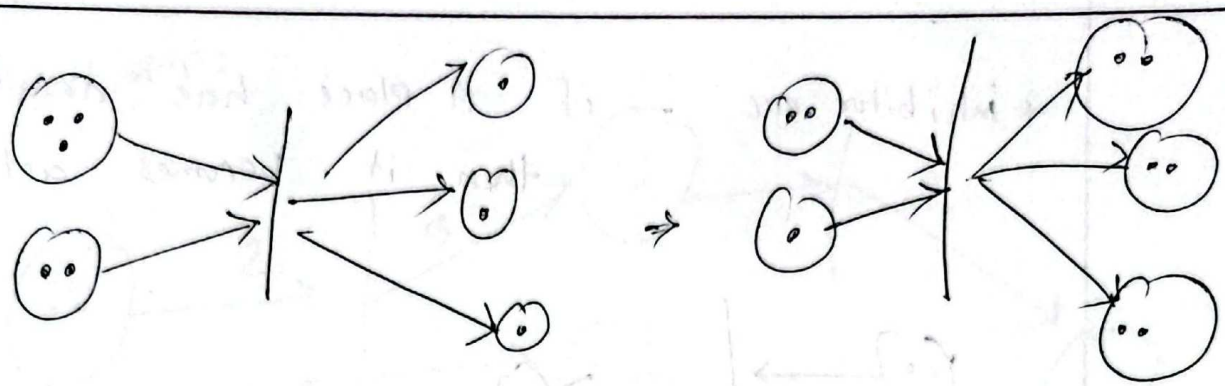


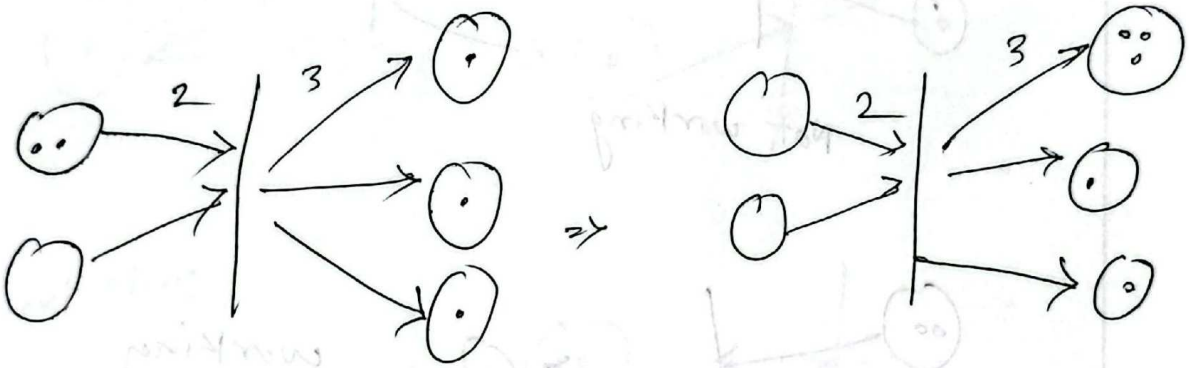
Places = $P_1, P_2, P_3 \dots P_n$

transitions = $t_1, t_2, t_3 \dots t_n$

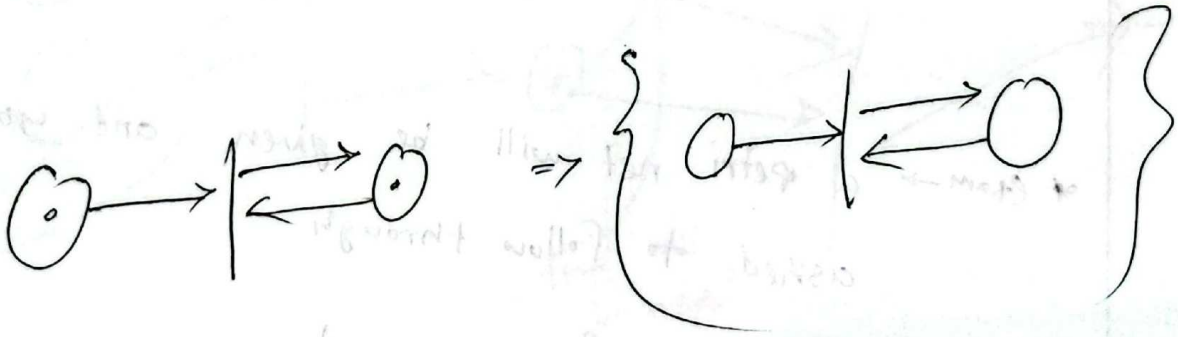
* tokens & firing

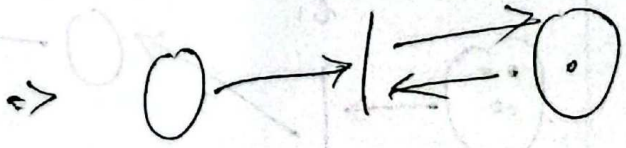




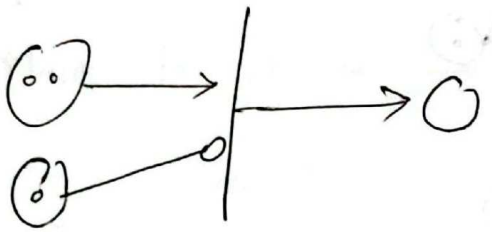


How to
Never run out of tokens?

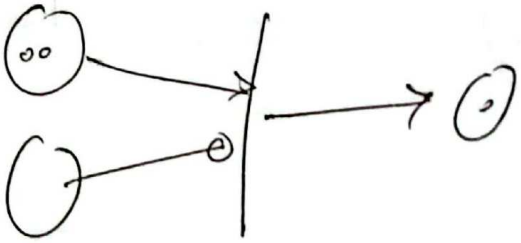




* inhibitor arc → if A place has no token then it becomes active

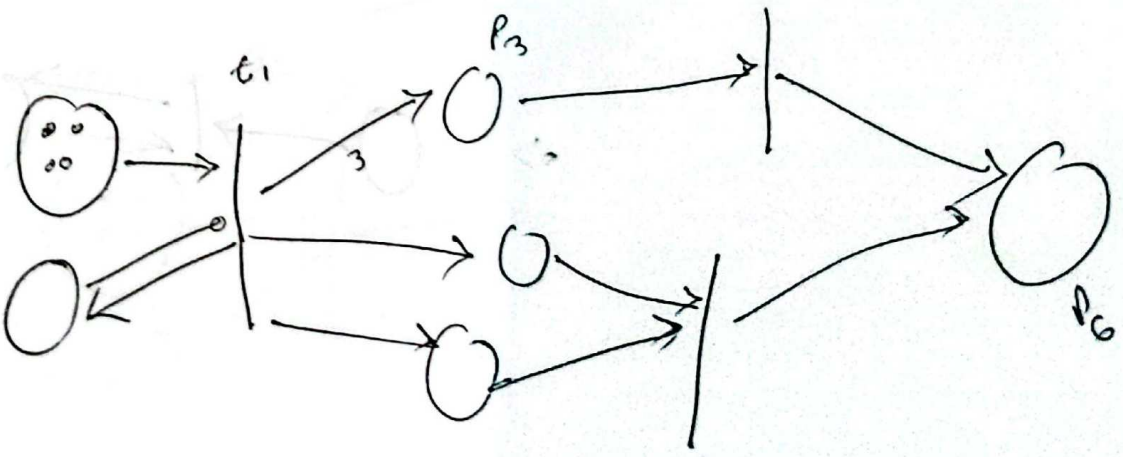


Not working

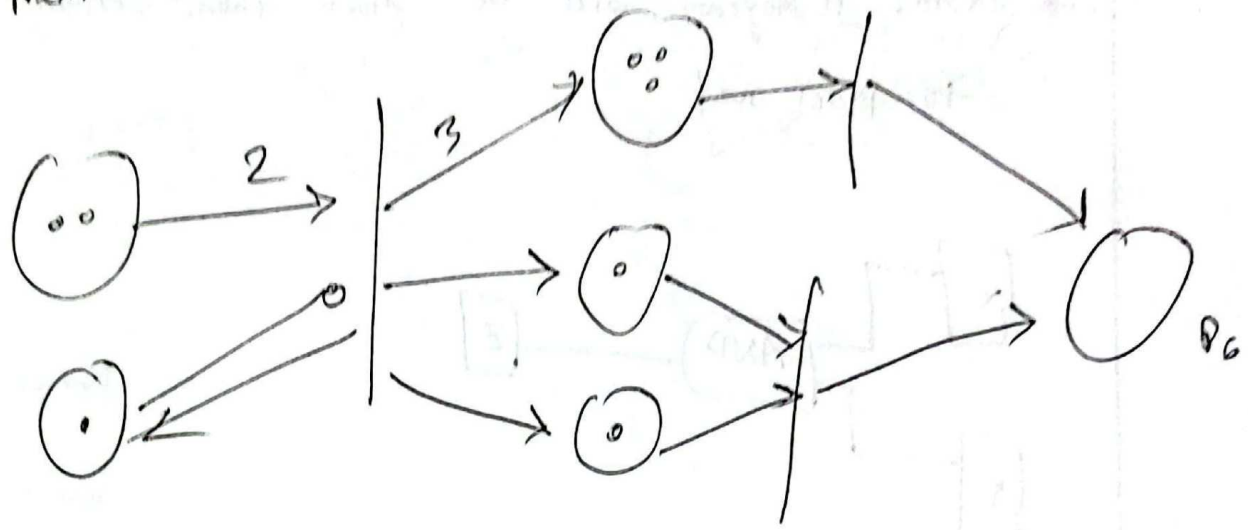


working

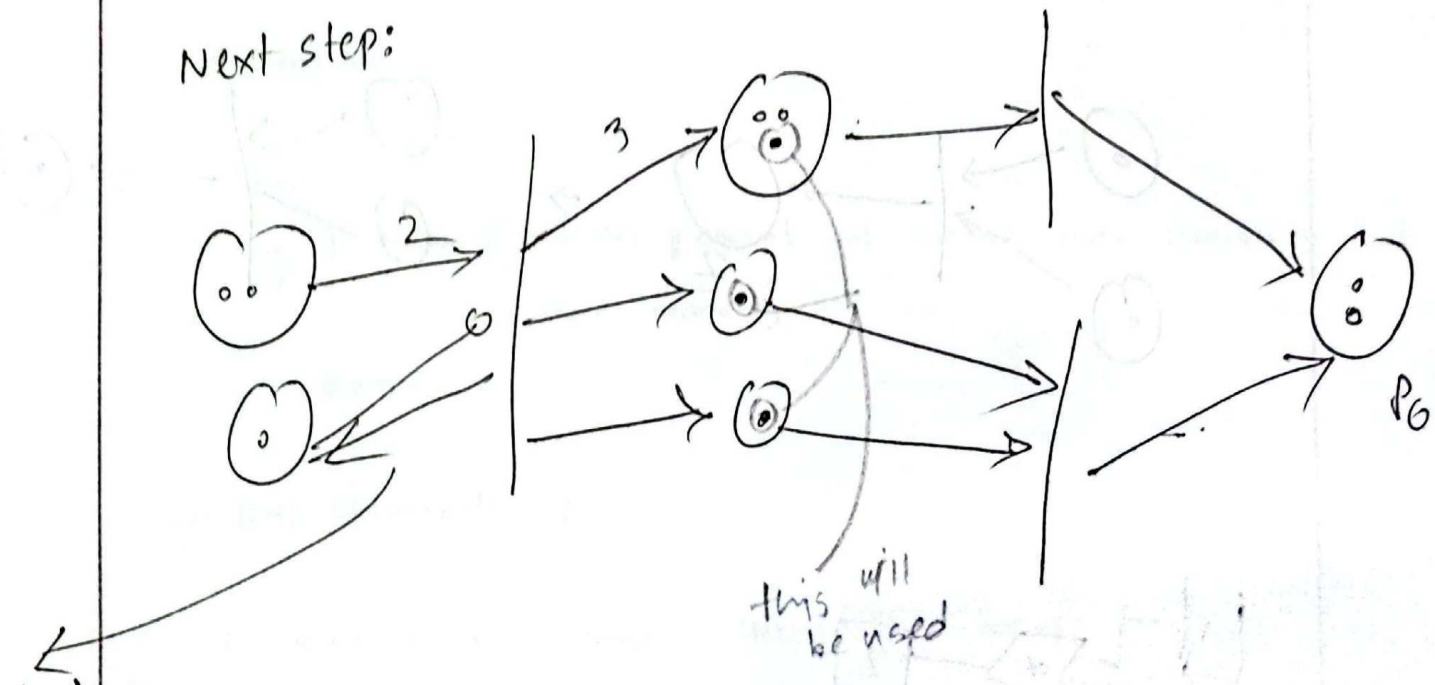
* Exam → a petri net will be given and you'll be asked to follow through.



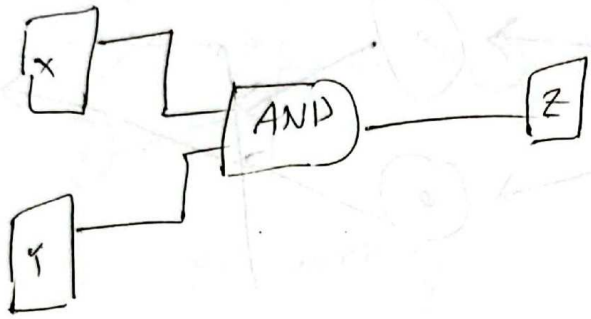
Next step:

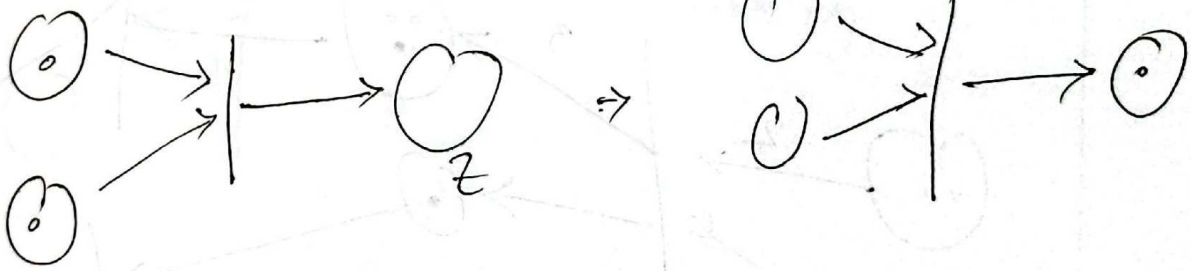


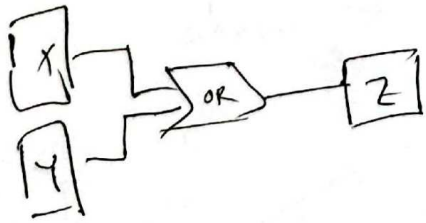
Next step:

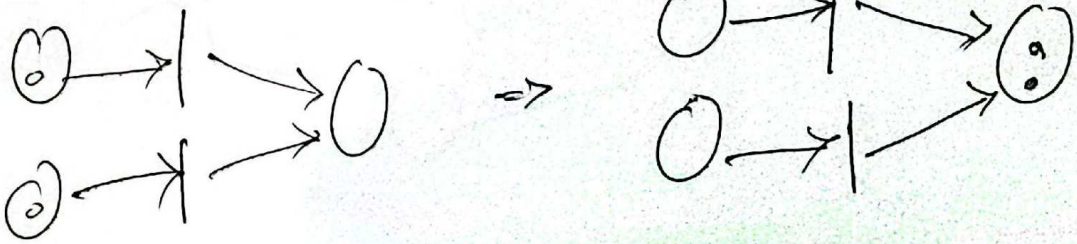


* exam: A diagram will be given, then convert it to petri net









07/03/2024

Optimizations

- Merging of tasks
- splitting of tasks (resources doesn't get blocked, more flexibility but more overhead)

Example

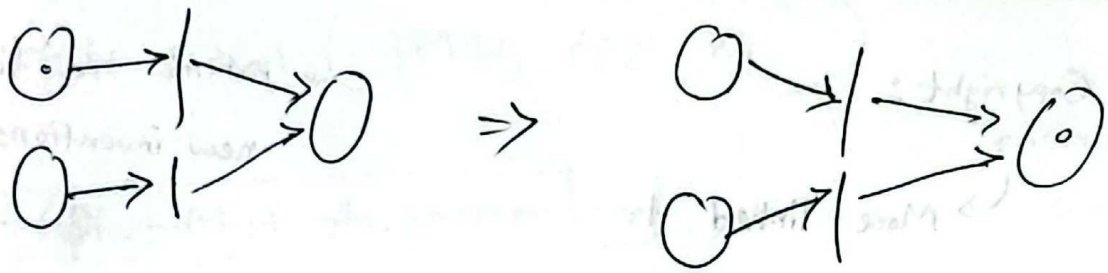
High-level Optimizations:

C works in row major. When the array is called, the blocks of allocated memory gets cached. More Misses than hits → poor cache behavior.

→ when row wise is better or worse than coloumn wise traversal.

for 3D, 2nd approach, K is rowise interms of j and columnwise in terms of i.

OR



Example

image

07/03/2024

Optimizations

- Merging of tasks
- Splitting of tasks (resources g doesn't get blocked, more flexibility but more overhead)

Example

High-level Optimizations:

C works in row major. When the array is called, the blocks of allocated memory gets cached. More Misses than hits → poor cache behavior.

→ When row wise is better or worse than coloumn wise traversal.

For 3D, 2nd approach, k is rowise interms of j and columnwise in terms of i.

07/03/24

FSD

Right side is better in terms of merging.

page - 19 : Top right is good er..

But bottom right is better.

Better locality = Better cache.

↳ Better Performance.

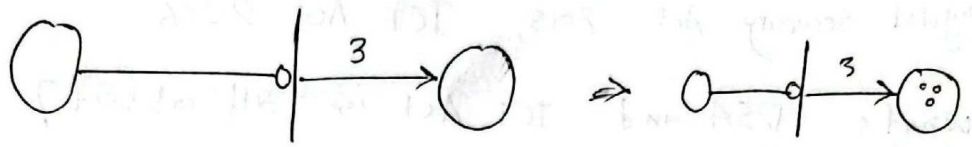
↳ enrolling, assigning etc...

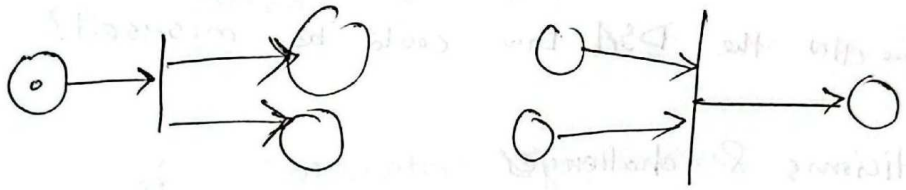
Important

- Array folding
- SOA

10/03/2024

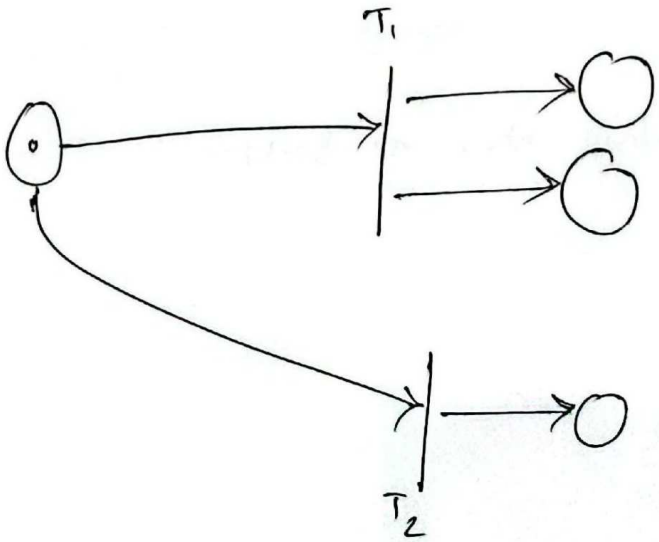
ESD





Concurrency

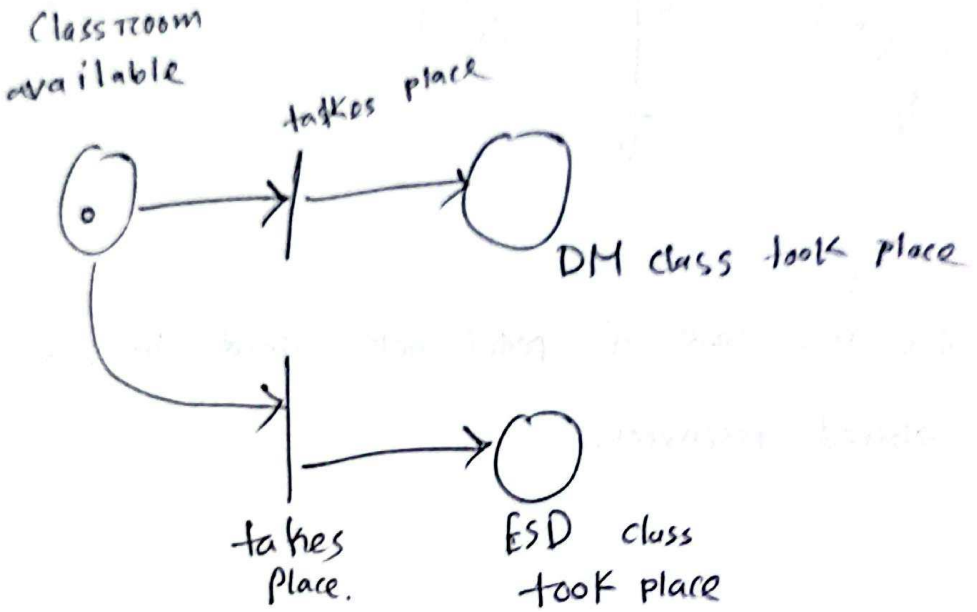
synchronization



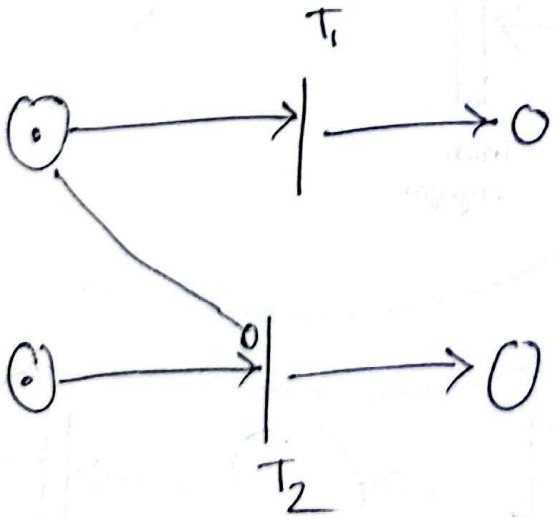
Mutually exclusive

either T_1 or T_2 will be activated. Since there's only one token available. So Non-deterministic.

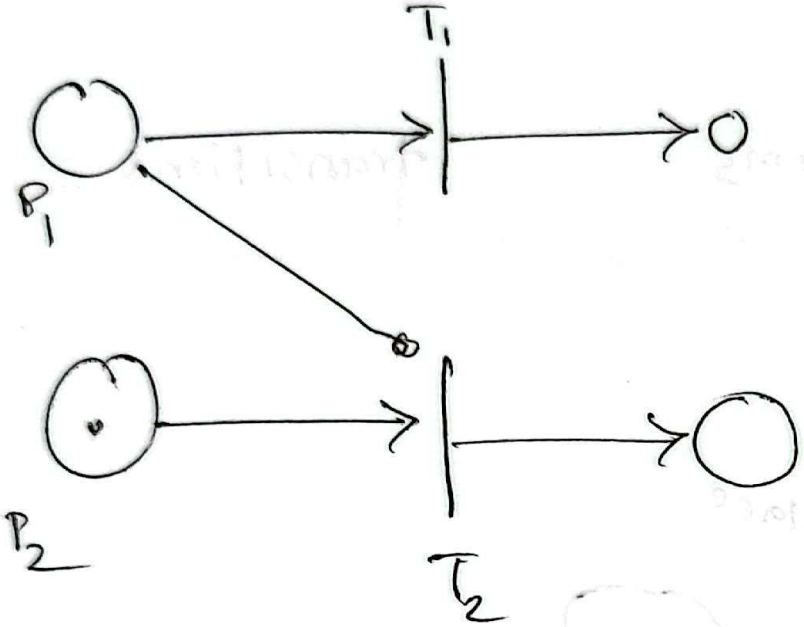
Places = conditions transitions = events



Mutually exclusive → if one takes place then the other one doesn't take place.



Here T_1 can happen, But the conditions don't fulfill in case of T_2.

* 

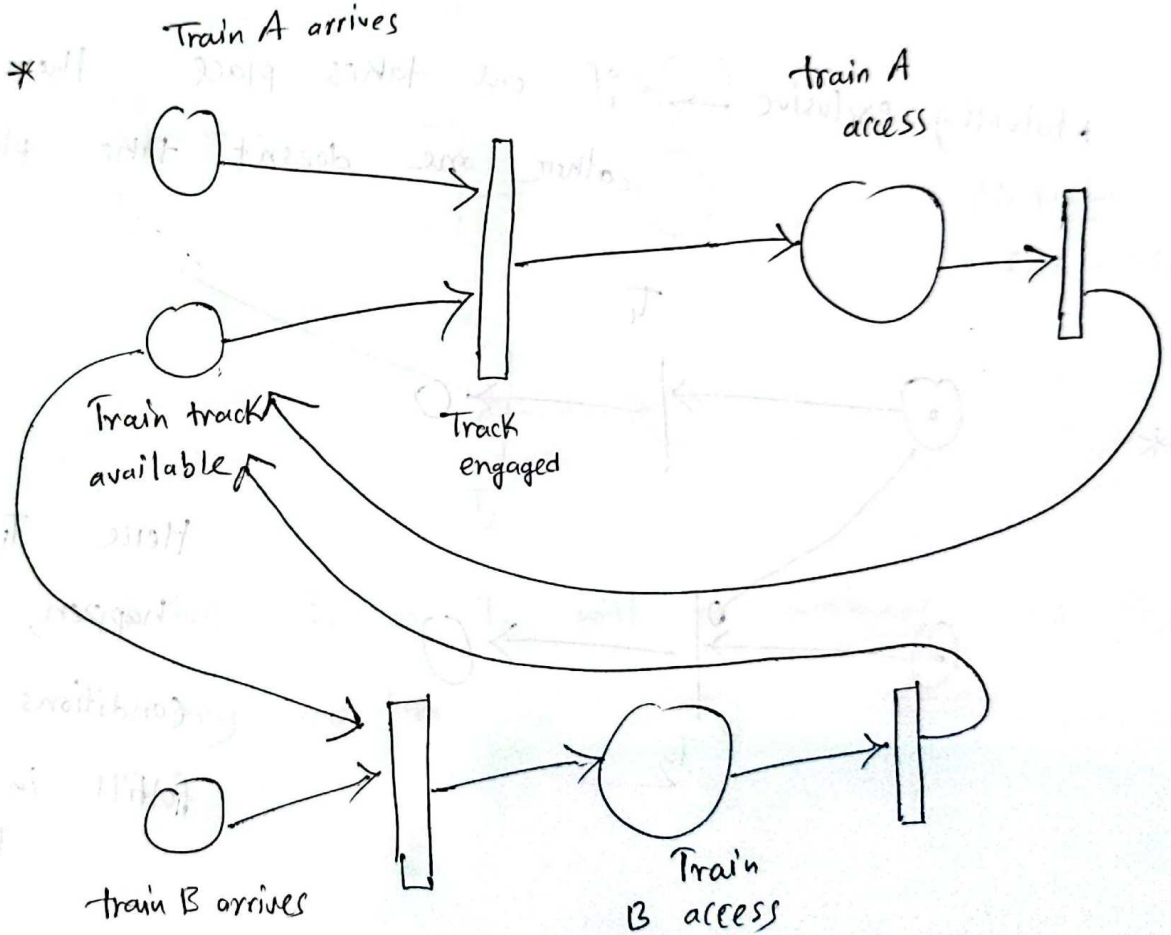
Here, T_1 won't get triggered. T_2 will.

* one use case of petri nets could be; usage of shared resources.

Make sure t_1 and t_2 don't collide

* one use case of petri nets could be ; usage of shared resources.

*



* 3 philosopher problem

3 actions?

- Eating
- Thinking
- Getting forks

they need two forks to eat. But there's only three forks available.

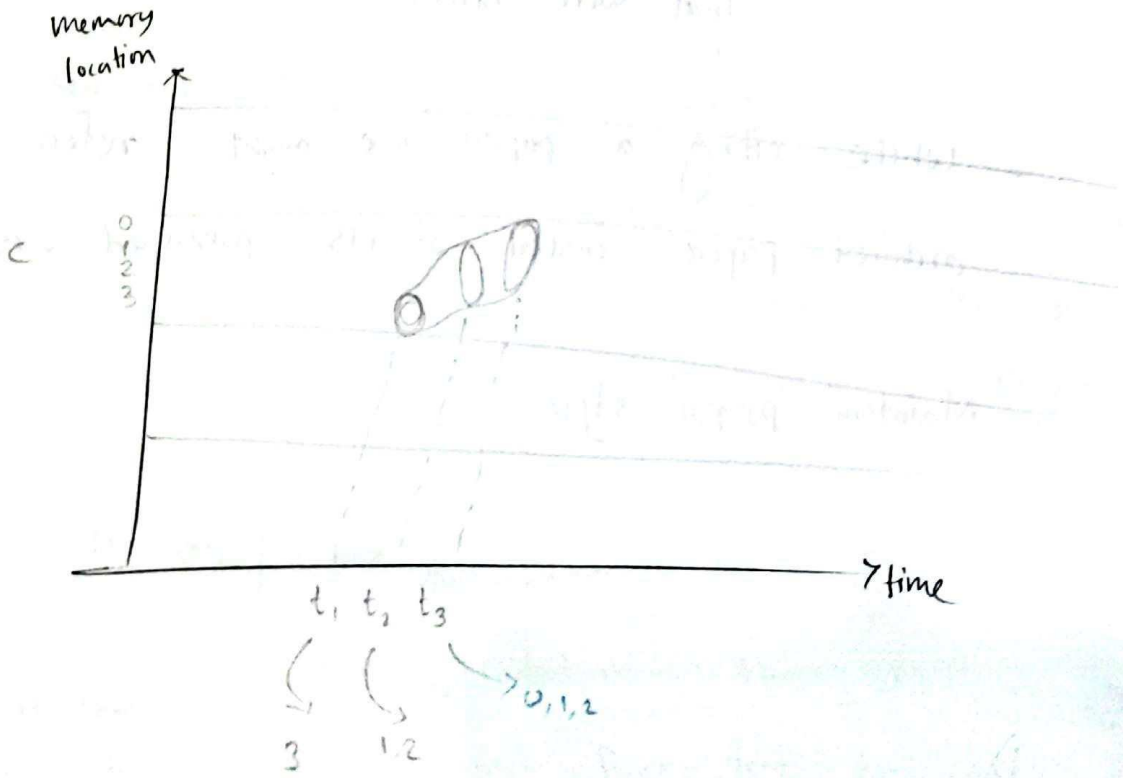
11/03/2024

ESD

Array Folding ← unbounded
inter
intra

↳ point? → Optimization of Memory Allocation.

intra
inter array folding: only allocate memory when needed and the delocate it when its not needed.

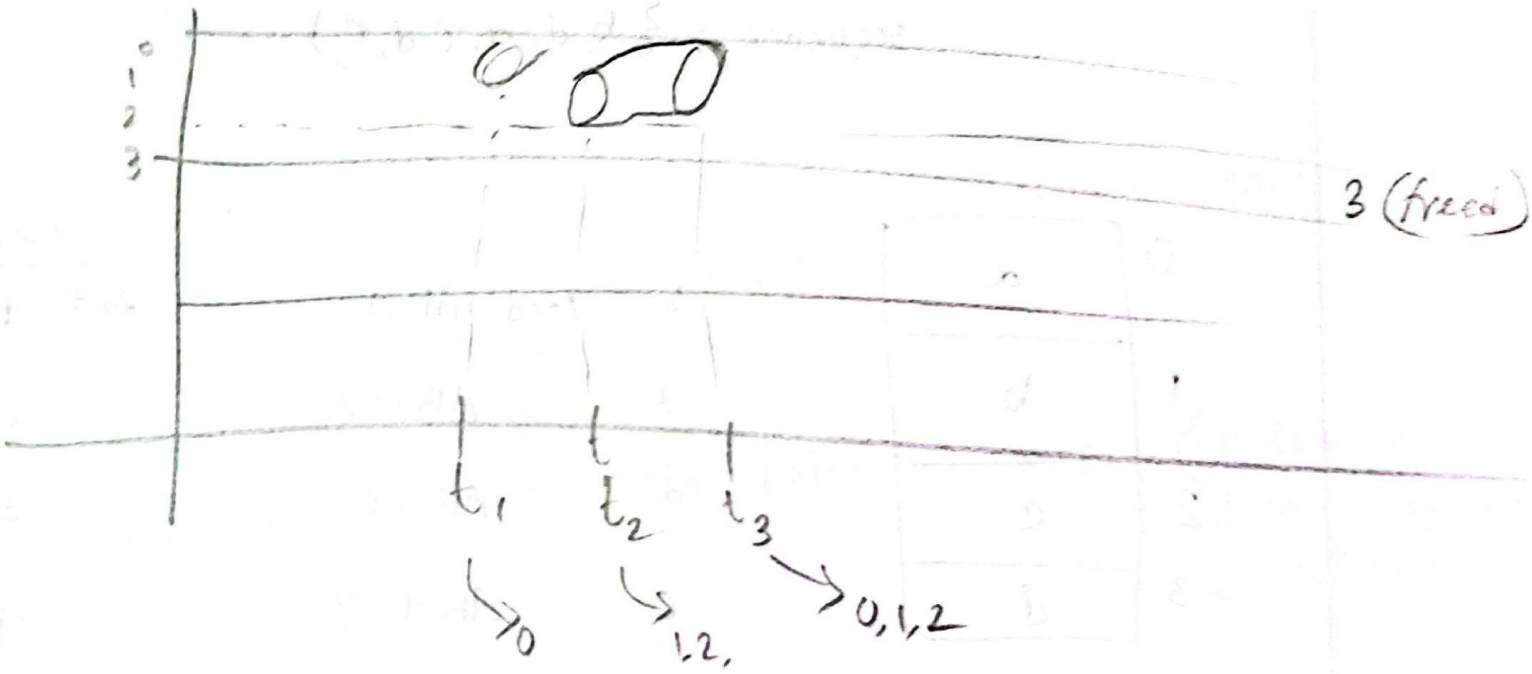


t 3 is only used at t_1, 0 is free until t_3.

So, we can just replace keep the values of 3 at index 0 at t_1. since 0 is not used.

This allows the 3rd index to be free.

(Freeing the Memory) → (so Memory optimization)

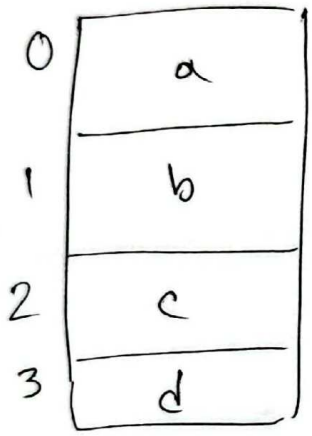


Inter array folding:

* SOA

we make sure execution cycle cost isn't high.

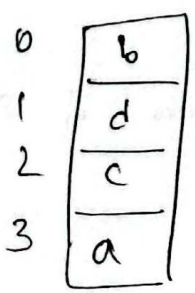
sequence = { b, d, a, c, d, c }



		cost
b	load AR, 1	~~cost~~ 1
d	AR+=2	2
a	AR--3	3
c	AR+=2	4
d	AR++	4
c	AR--	4

No cost

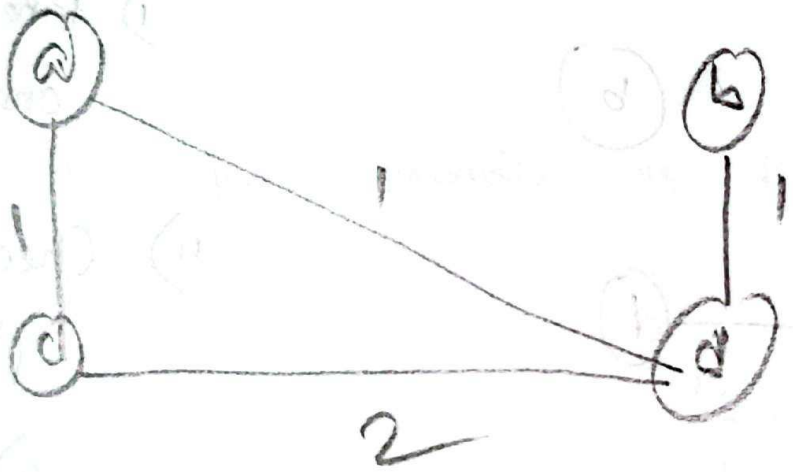
* let's say we figured out the best way to layout ↔ keep the data.



	cost
AR, 0	1
++	1
+2	2
--	2
--	2
++	2

So, How to get the optimal layout?

seq = { b, d, a, c, d, c }



sequence graph

arc value = the times the pattern comes (c to d or d to c happens twice)

purpose → create a graph that helps us to create the optimal layout.

So

Answer: ~~Minimum spanning tree~~

1) High valued arcs should be together.

observation

- theres no cycle
- Nodes will be visited once
- Hamilton path figuring
- It will be a tree
  ↳ one node can be connected with at most two nodes.

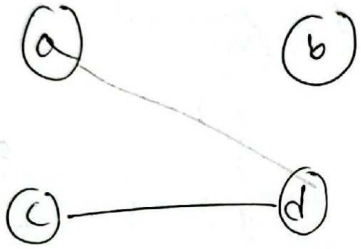
(c,d) 2

(a,c) 1

(a,d) 1

(b,d) 1

sorted via weight-wise.



i) take the highest ones first

ii) Check:

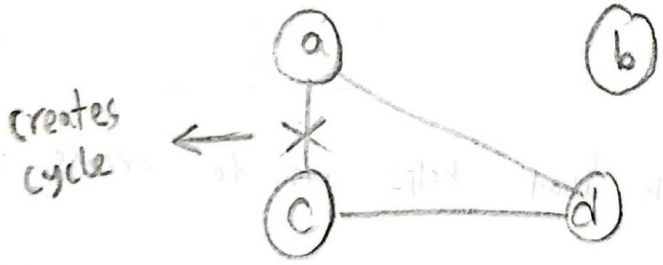
1) degree < 3

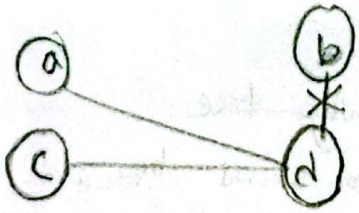
ii) No cycle creation

~~iii) tree must~~

if not then

- backtrack and choose another arc





→ degree of d becomes 3.

So backstrack (we can't choose (a,d))

12/03/2024

ESD

Testbench Coverage - Examples

Simulation - based input : we have to consider all the possible inputs

10, 11 inputs provides all the possible outputs of the models.

	10	11	
XOR	1	0	[1, 0, All possible outputs are there]
NAND	0	1	

* FSM

I = Inputs O = Outputs Q = all sets Q_0 = initial states

f = transition functions H = final states

Here, even though the inputs are the same, all the other things can be different.

* FSM equivalence checking - Algorithm.

i) find all the possible states $(Q_1 \times Q_2)$

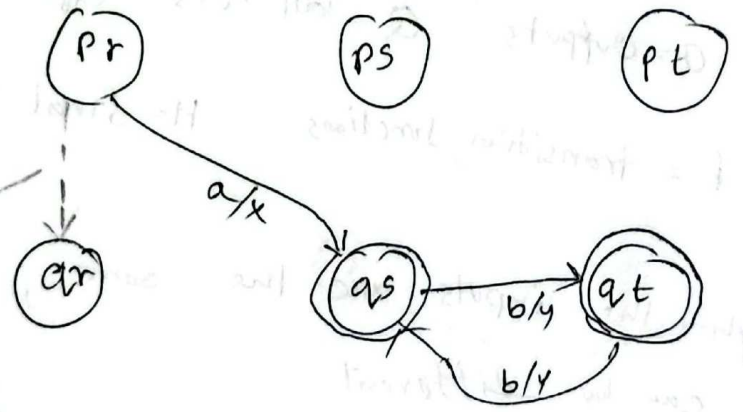
ii) For ~~same~~ every input, from each possible state, find destination states

iii) then check: ~~i) If destination states~~

i)

$Q_1 = \{p, q\}$ $Q_2 = \{r, s, t\}$ $\Sigma = \{a, b\}$

$Q_1 \times Q_2 = \{(p,r), (p,s), (p,t), (q,r), (q,s), (q,t)\}$



if there was this line then the FSMs would've been non-equivalent

Mismatches

qr

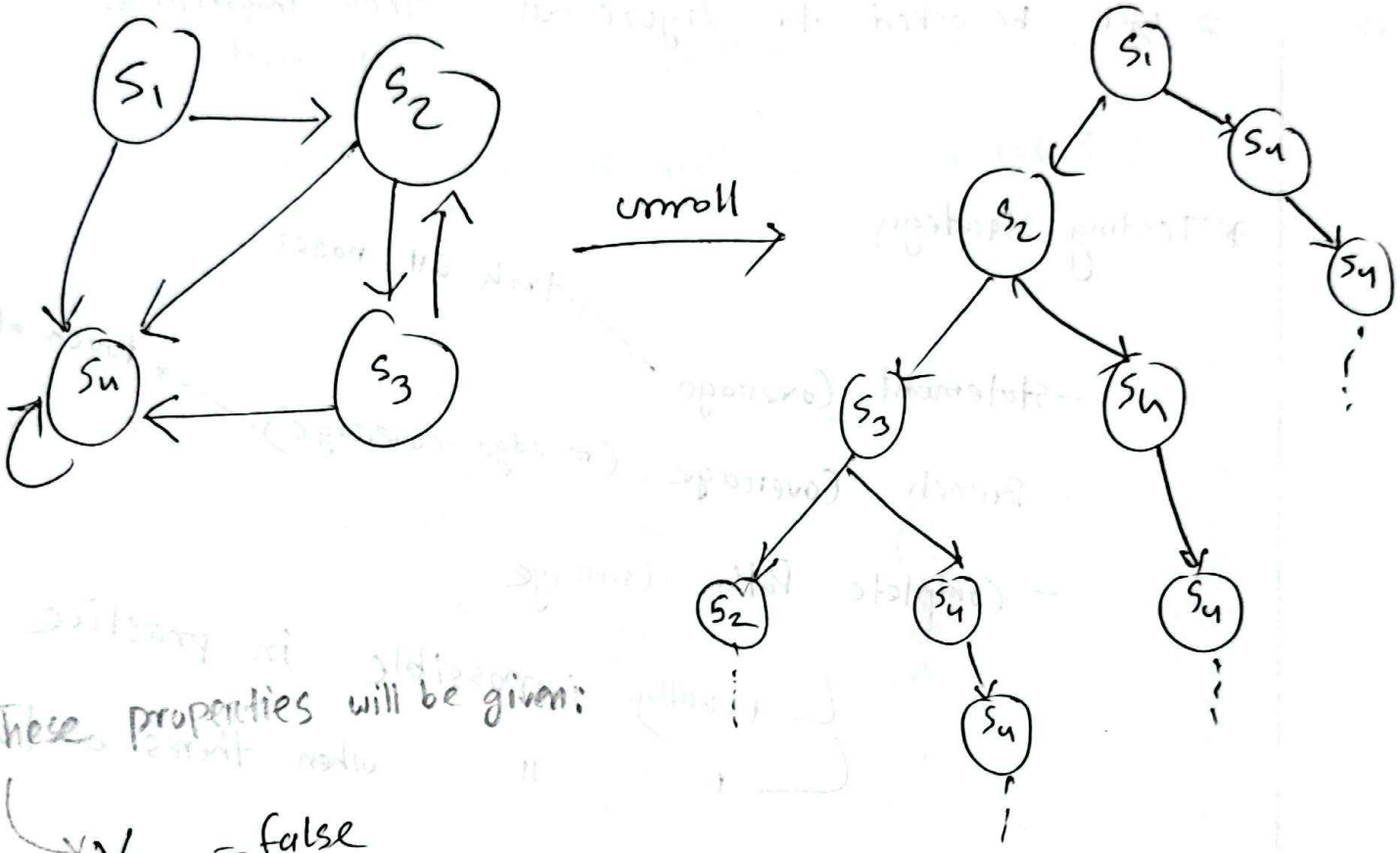
ps

pt

But, there's no lines between them. so green light

– these have to be not mismatches. Here, (qr) is a mismatch. Because q is a final state but r isn't. Same goes for (ps) (pt)

* Model Checking - the one with properties



These properties will be given:

$\forall_{S_2}$ = false

E_{S_2} = true

$\forall_{S_2 \to S_3}$ = true

$\forall_{\text{reach } S_4}$ = False (there's a loop in S_2 and S_3 .... might never go to S_4)

if these are all true

then Model check is successful

19/03/24

ESD

Introductory + Hardware stuff → Mukhosto jnish

, → 1 question

Model specifications → timed automata, SDL

petri nets → Advanced, seenario

Scheduling → 1

priority protocols → 1

optimizations, validations, verifications → 1

implementation Related stuff mainly

MID Related stuff → Group Presentation on Topic

Optimizations → Memory, Execution time

→ How to allocate Memory

inter — array folding → between arrays

intra " " → between elements of the same array

→ instead of allocating different spaces for different arrays, after an array's access time is over, deallocate the memory and replace it with a different array.

* ARM vs x86

→ ARM performance has come a long way. The performance is closer to x86 even though energy cost is very low.