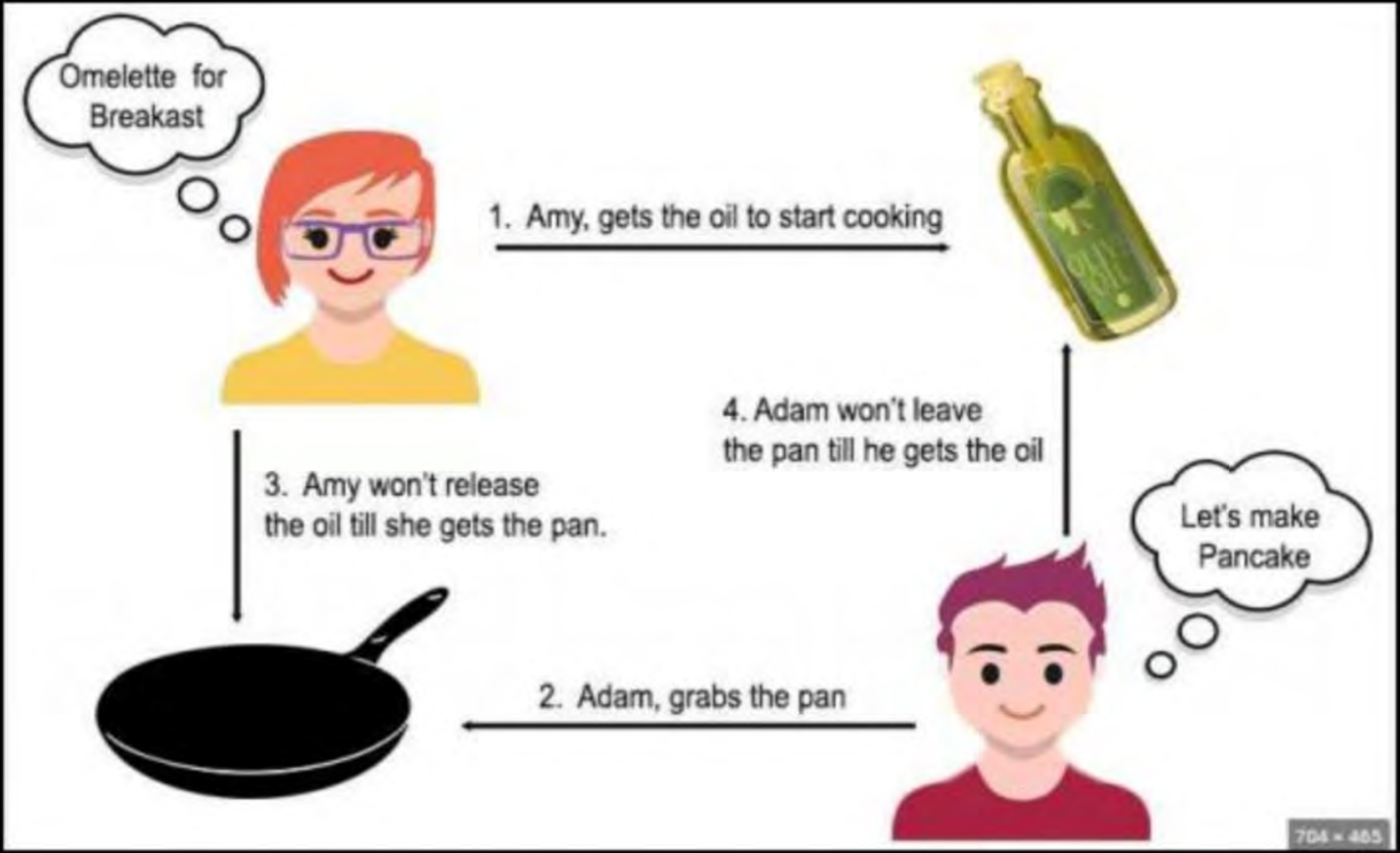
COMPUTER SCIENCE

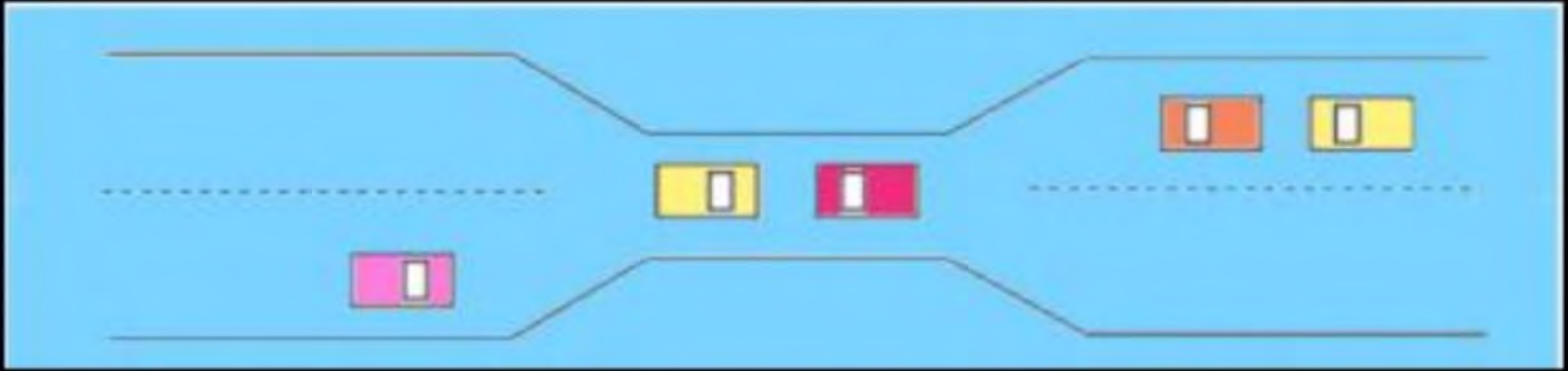
Deadlocks 01

Dr. KHALEEL KHAN SIR

TOPICS TO BE COVERED

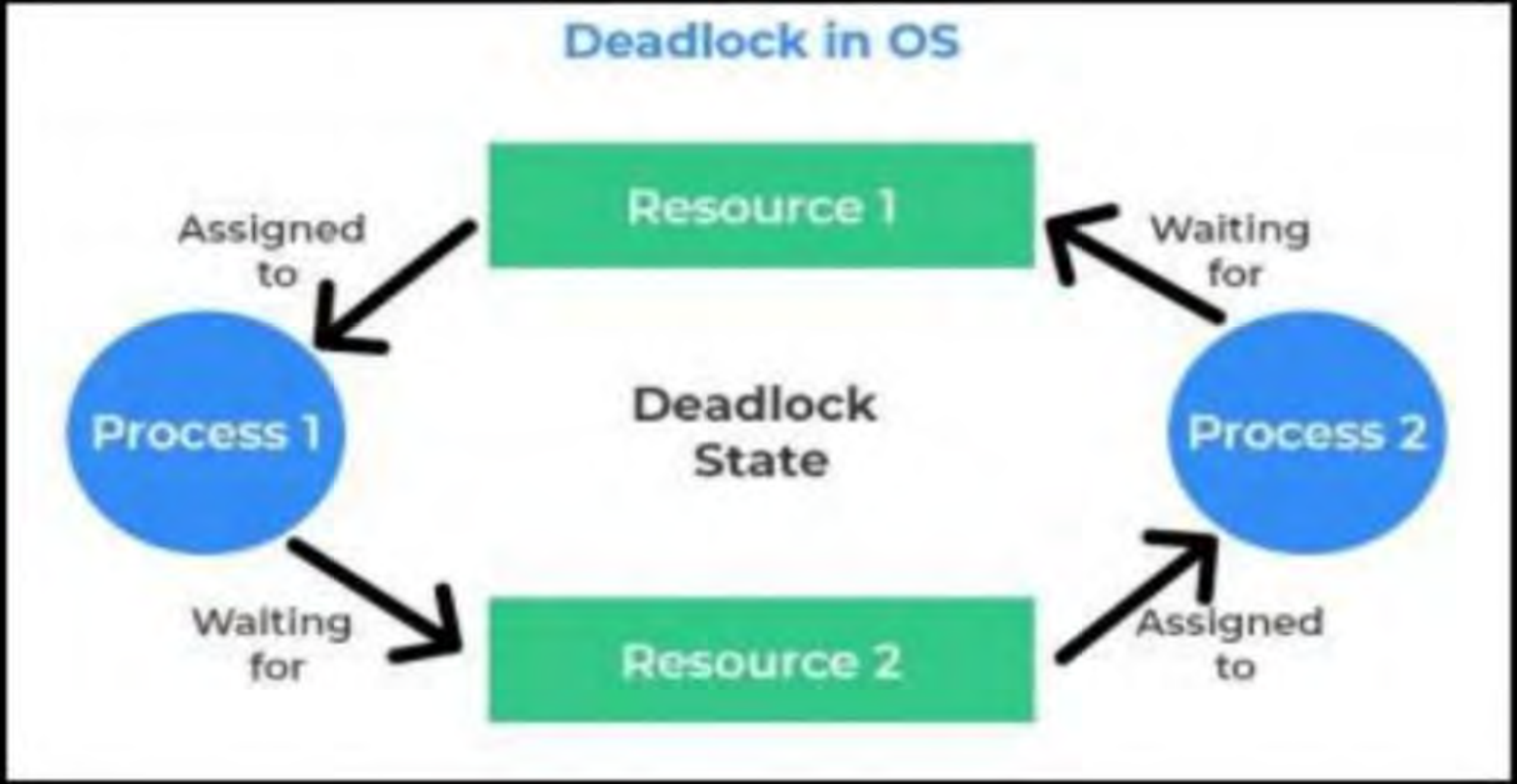
1. Concepts of deadlocks


Omelette for Breakast
1. Amy, gets the oil to start cooking
4. Adam won't leave the pan till he gets the oil
3. Amy won't release the oil till she gets the pan.
Let's make Pancake
2. Adam, grabs the pan




You release the lock first
Once I have finished
my task, you can continue.
Why should I?
You release the lock first
and wait until
I complete my task.


INTERVIEWER: EXPLAIN DEADLOCK AND I'LL HIRE YOU
ROGRAMMER: HIRE ME AND I'LL EXPLAIN IT TO YOU



Two/more Processes are in deadlock, iff they wait for the happening of an event which would never happen; < Infinite Blocking >

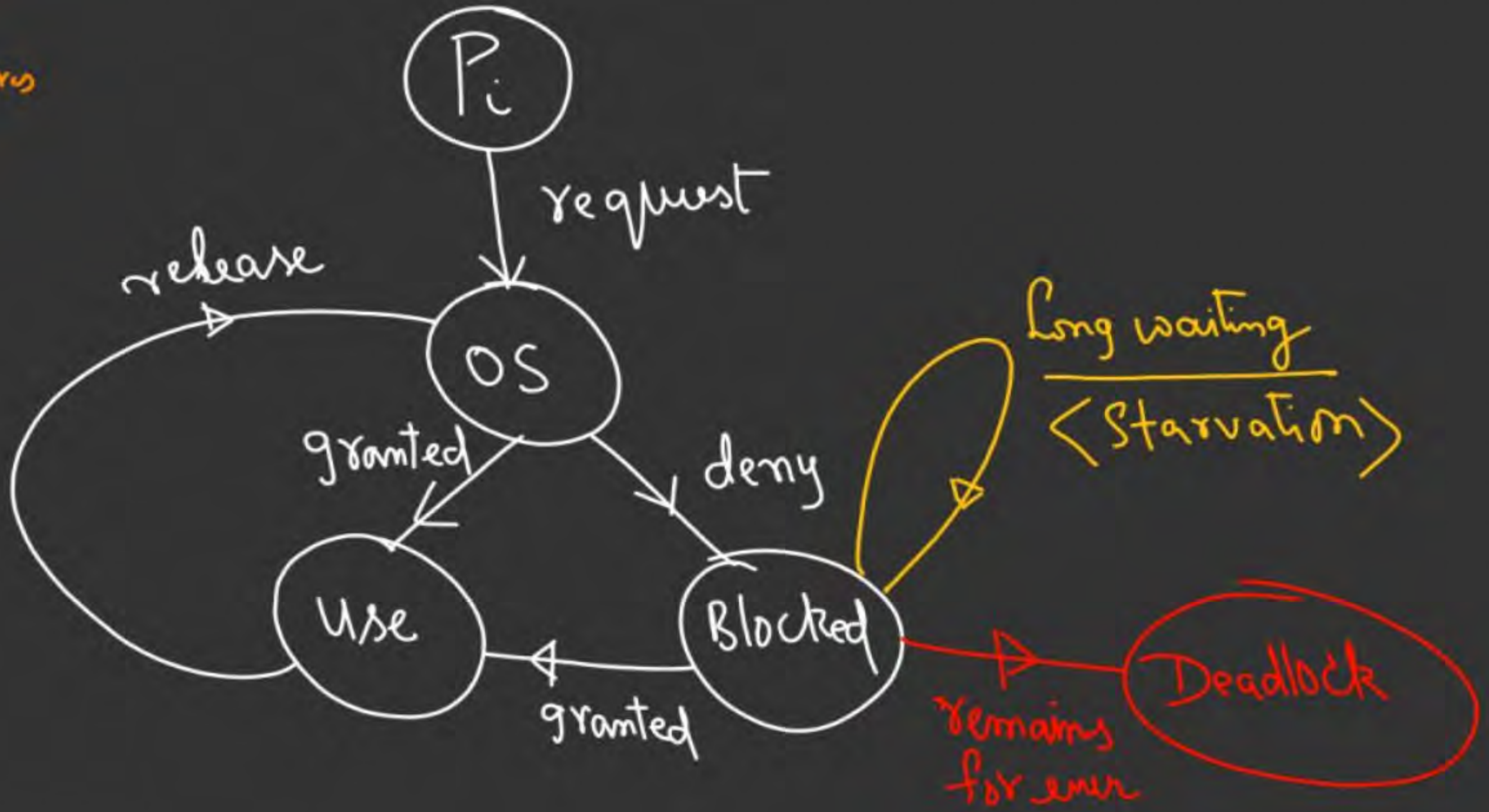
Deadlock is undesirable,

→ Low cpu utilization

→ Ineffective utiliz of resources;

1. System Model:

- → 'n' - processes
- → 'm' - Resources
  - Single Instance — CPU
  - Multi Instance — Register
  - H/W <CPU + Mem + IO ..>
  - S/W <Files, Semaphores ...>

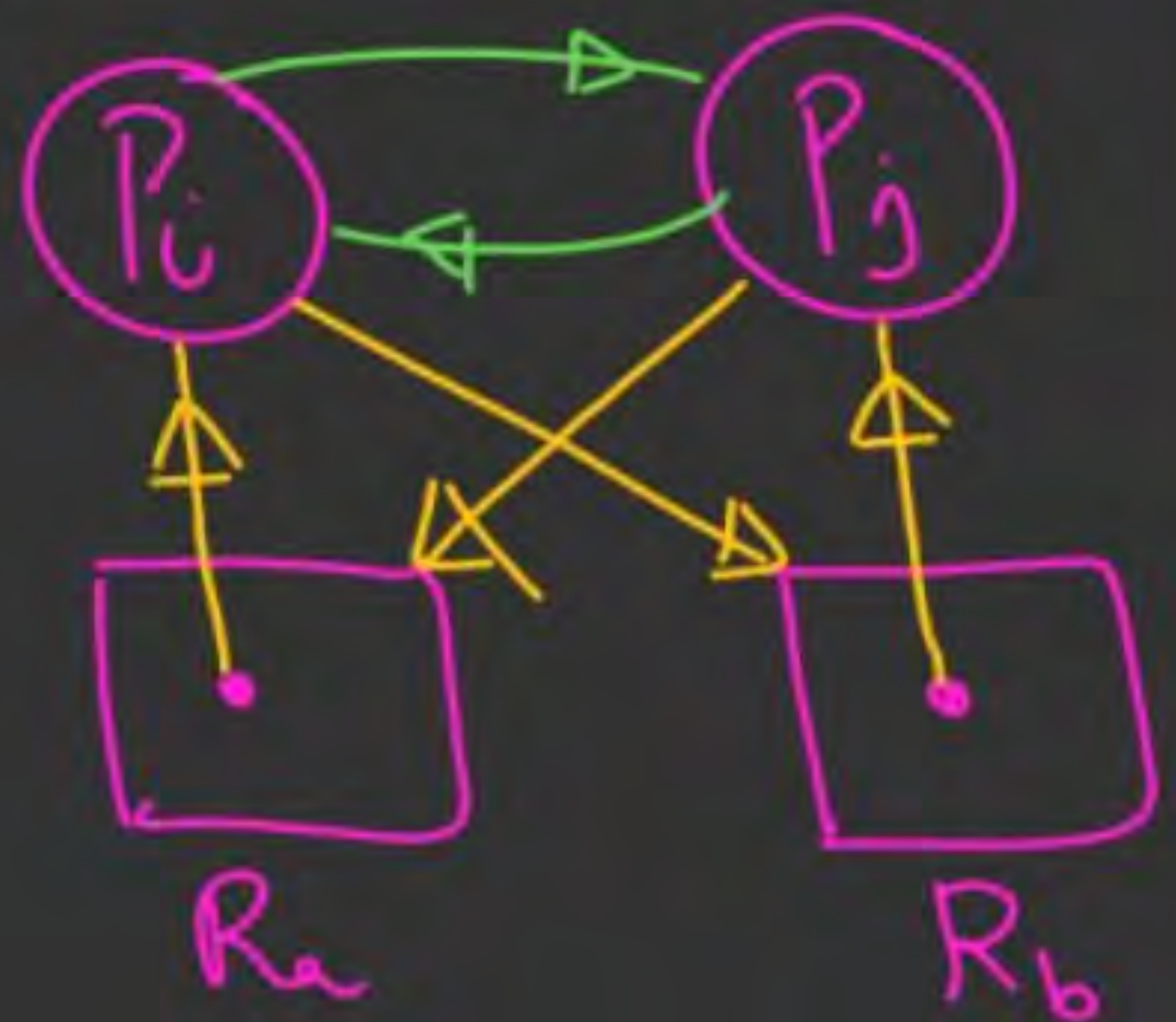


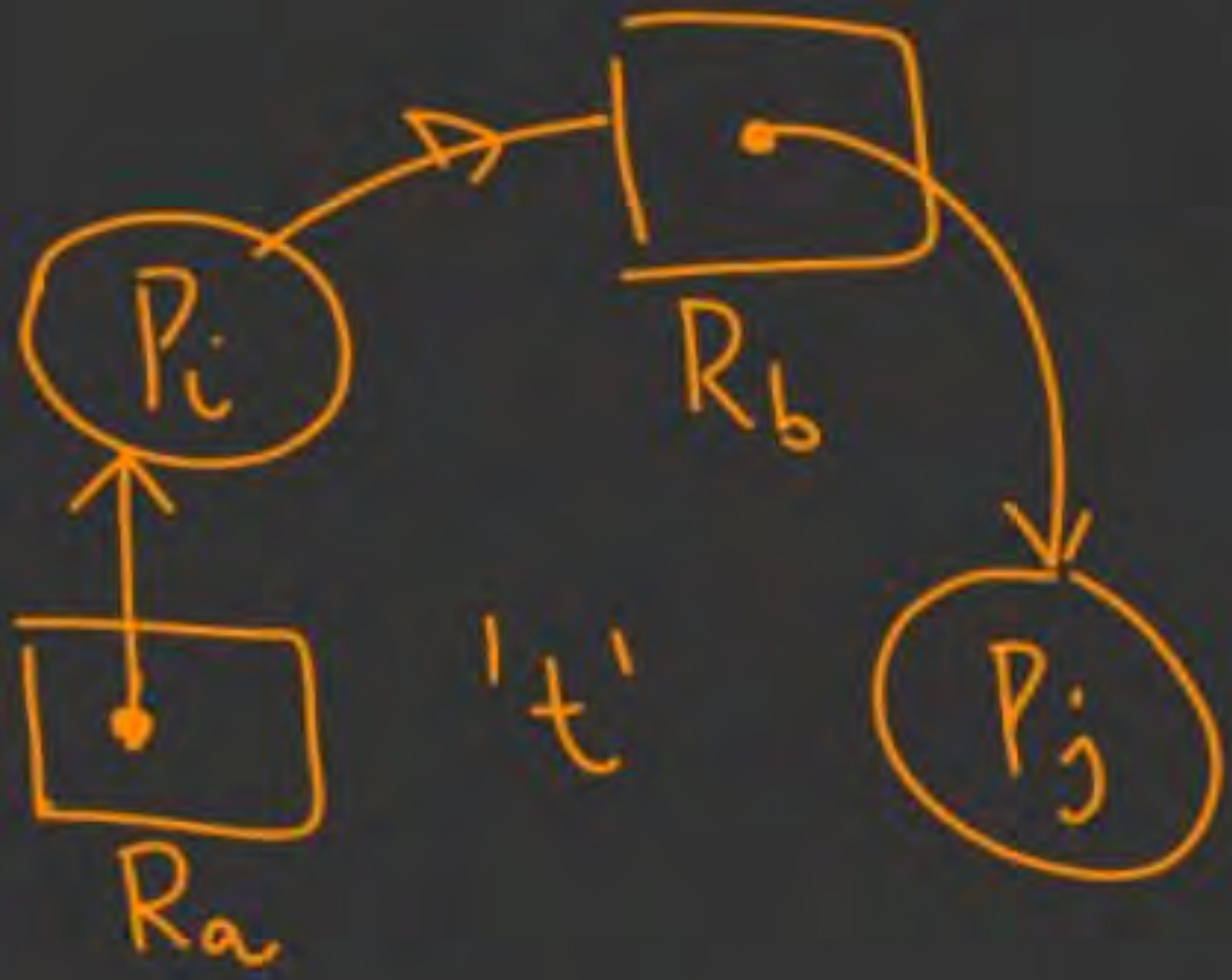
Characterization

1. **Necessary Conditions:**

(i) Mutual Exclusion: <CS> → Shared resources

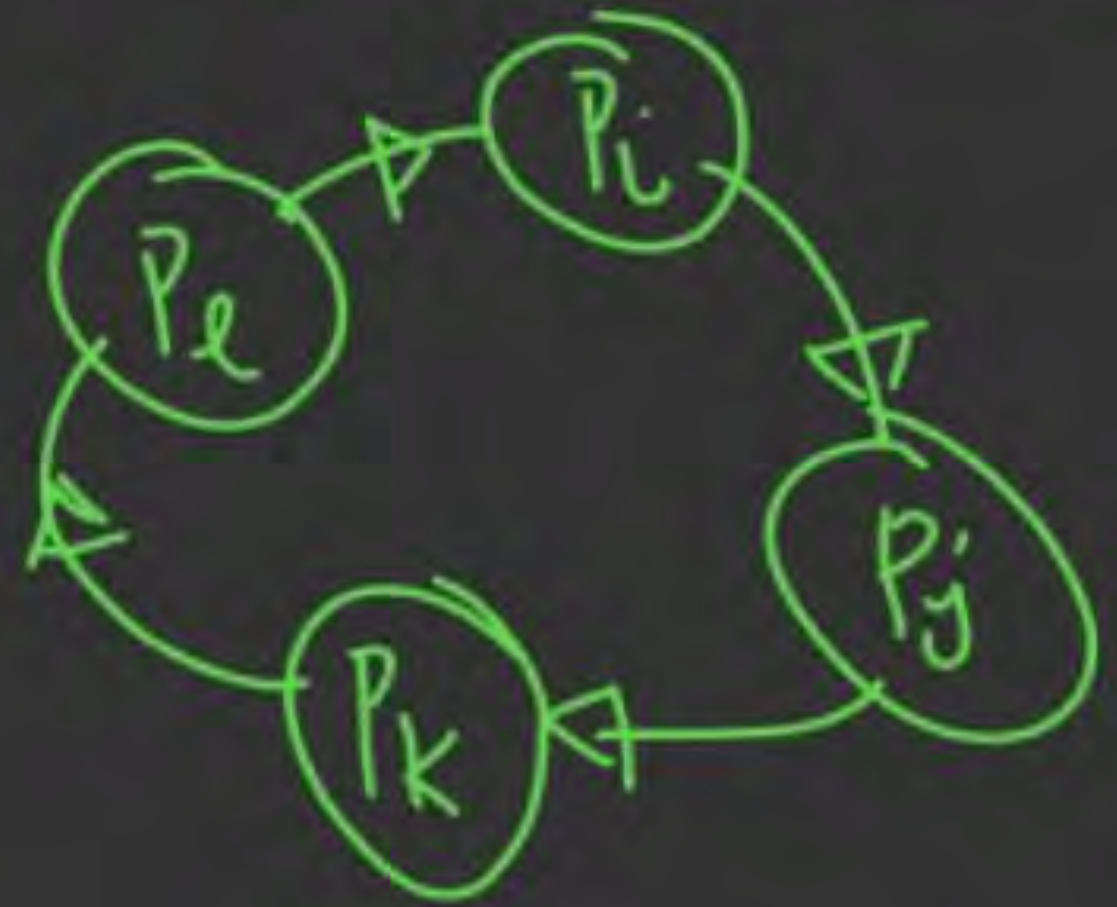
(ii) Hold and wait:





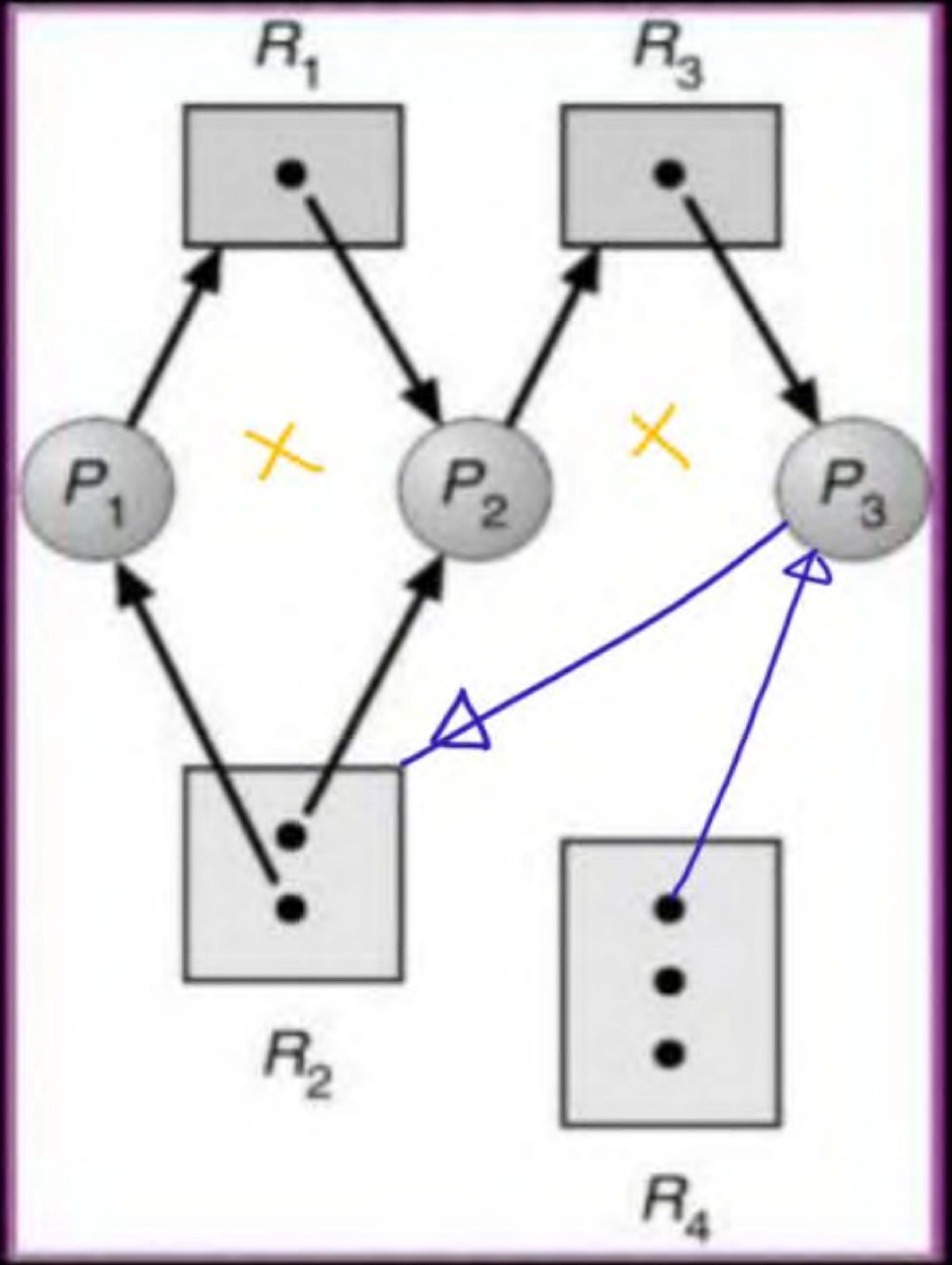
(iii) No-PreEmption of resources

(iv) Circular wait:



Resource Allocation Graph (R.A.G)

$G = (V, E)$



G1

P1 & P2 are Blocked

P3 also gets Blocked

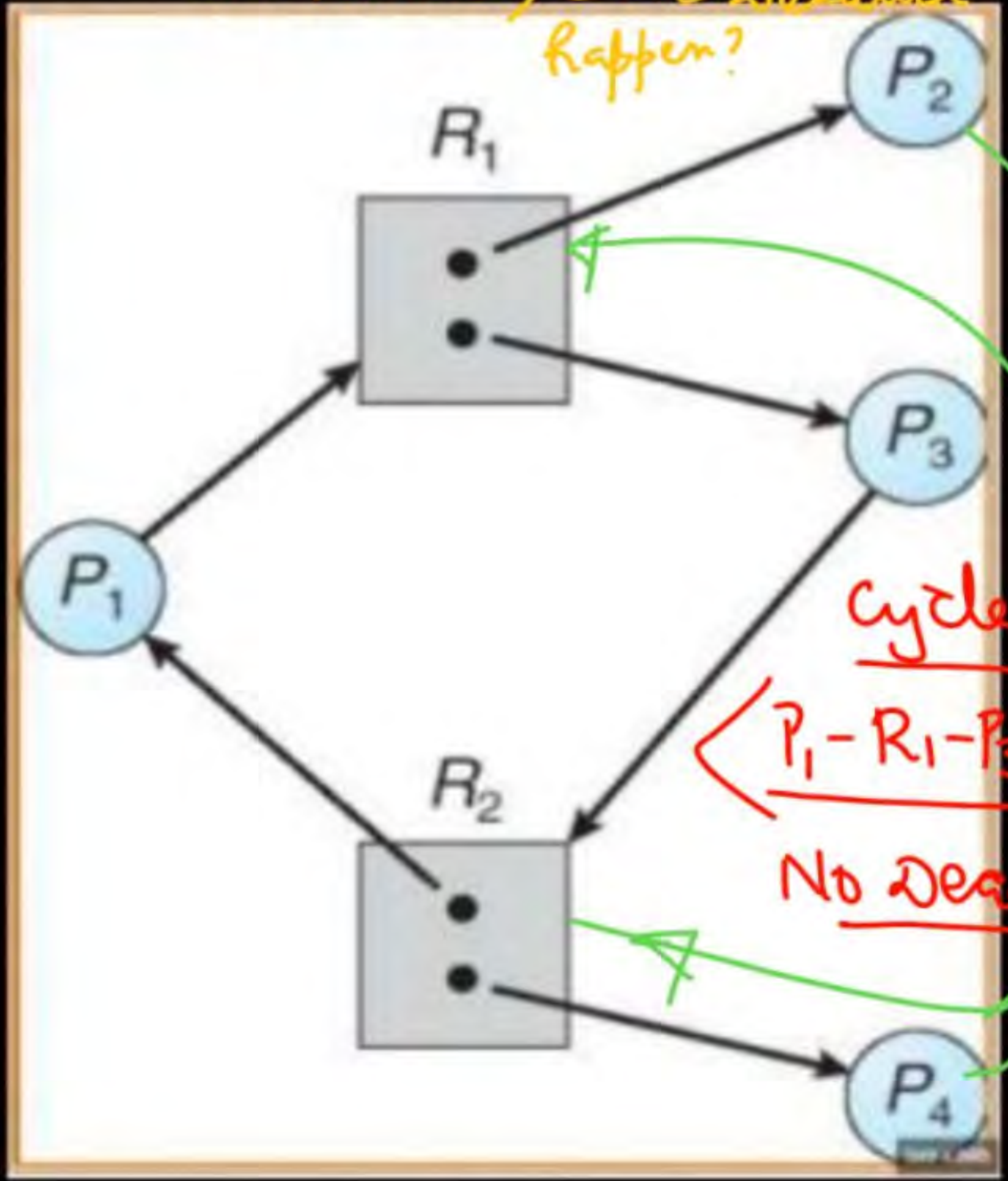
Cycle

$P_1 - R_1 - P_2 - R_3 - P_3 - R_2 - P_1$

$\Downarrow$

Deadlock

In future, can deadlock happen?



G2

t_1'

Cycle ?

$P_1 - R_1 - P_3 - R_2 - P_1$

No Deadlock

Note 1: If the R.A.G has multi-Instance Resource, then cycle is only necessary condition

Note 2: If R.A.G has only Single Instance Resource then cycle is necessary & sufficient condition for deadlock;

Strategies

(i) Deadlock Prevention
(ii) Deadlock Avoidance <Banker's Algo.>

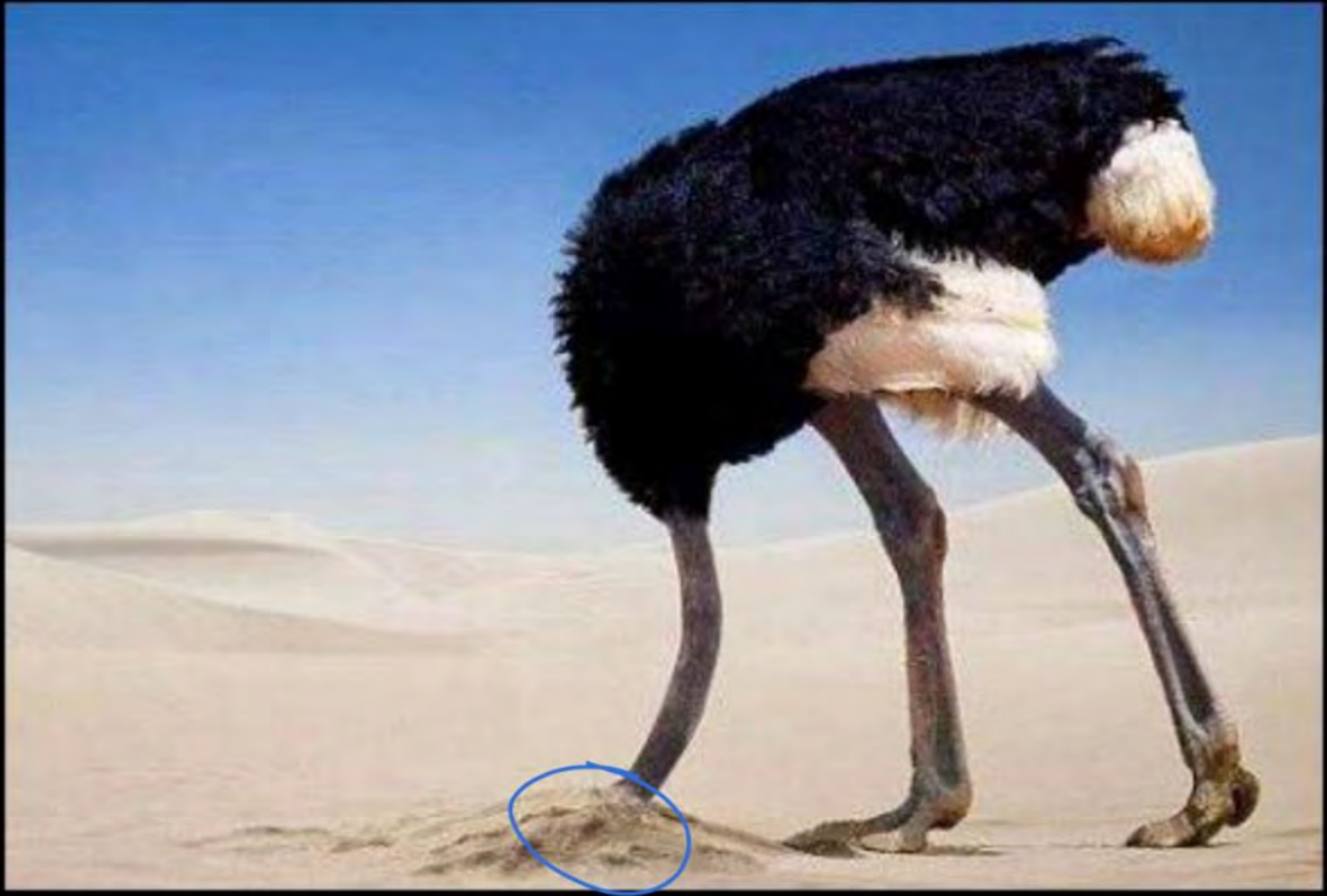
T_1 : Deadlock Never occurs

(iii) Deadlock Detection & Recovery <Doctor's Algo>*

(iv) Deadlock Ignorance; <Ostrich Algorithm>

<NO-Strategy>

T_2 : <Deadlock occurs>



Deadlock Ignorance

The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
  - Deadlocks occur very rarely;
  - Cost of prevention is high;
- UNIX and Windows take this approach
- It is a trade-off between
  - Convenience
  - Correctness

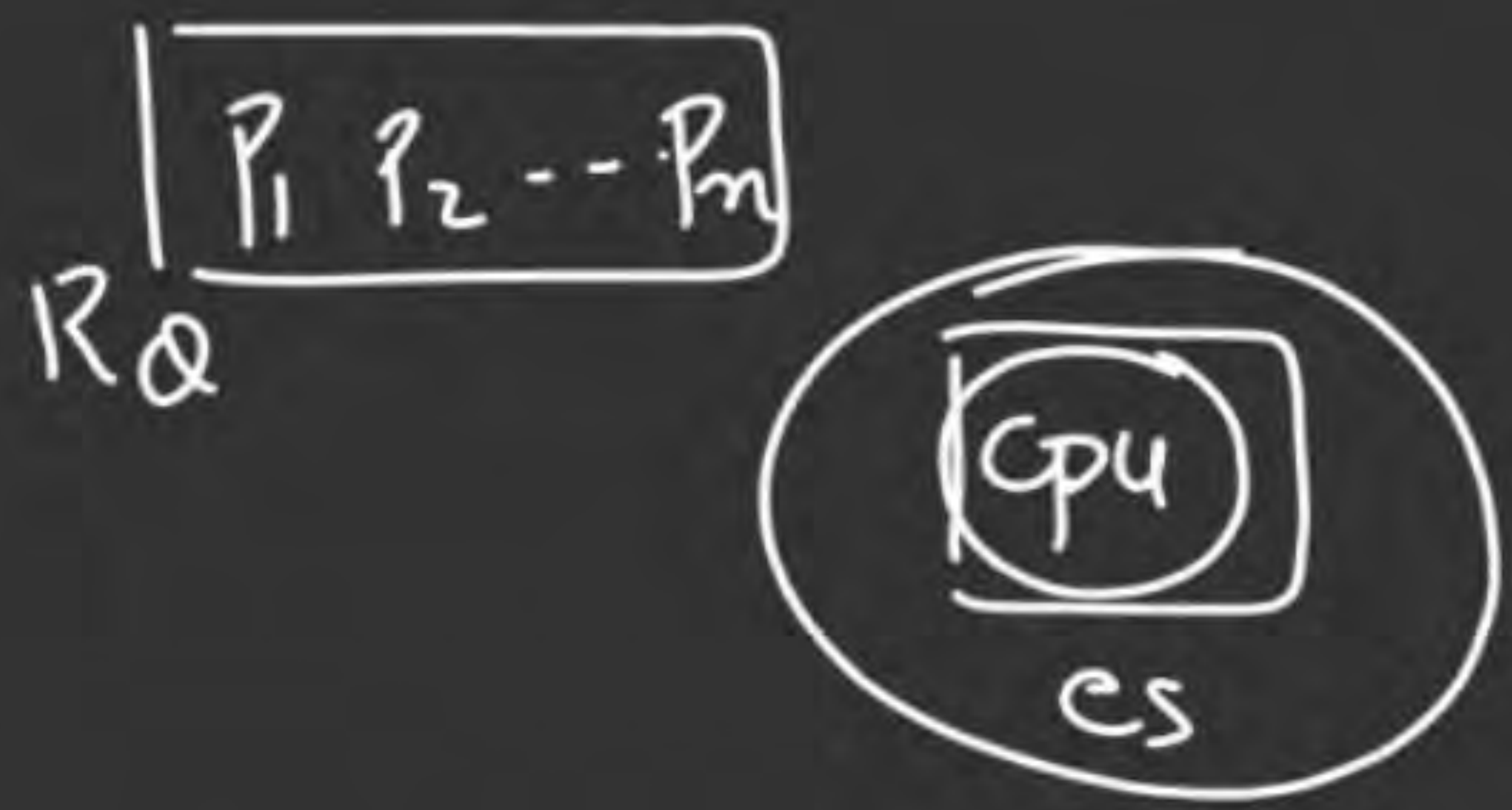
Burying your head in the sand will not make you invisible & neither It Solves the Problem It will only Increase your frustration;

No Strategy

Hanged

⊛ II. Deadlock Prevention: < by dissatisfying / Negating one/more of the necessary condition;

1) Mutual Exclusion:



Can we Imaging a System without a Shared Resource?

NO

Sys. must/will always have a Shared resource (CS)

"Mut. Exclusion is non dissatisfying"

(ii) ! (Hold and wait) : (Hold or wait)

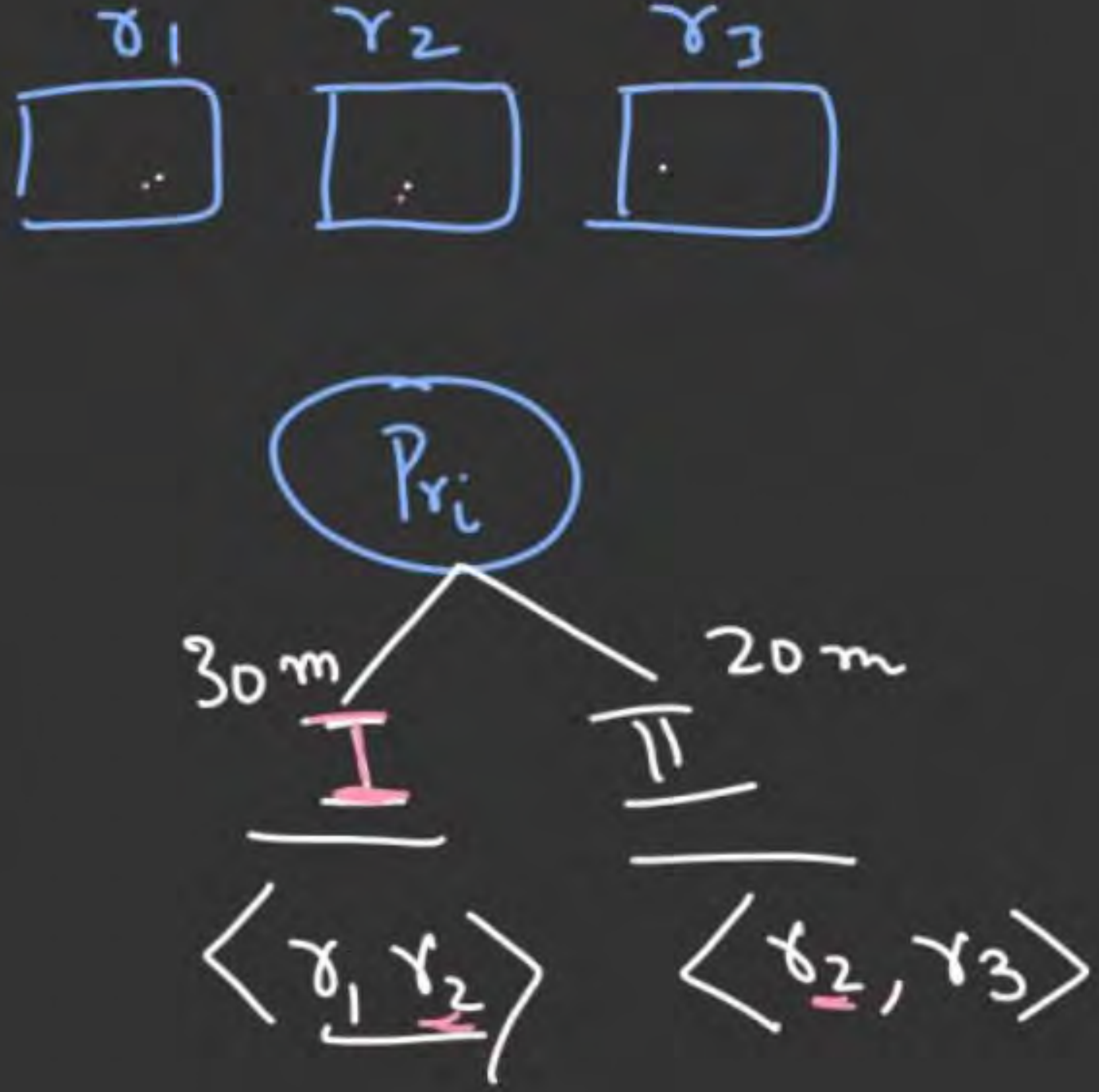
a) Request & be allocated all the resources Prior to its execution

(i) Starvation

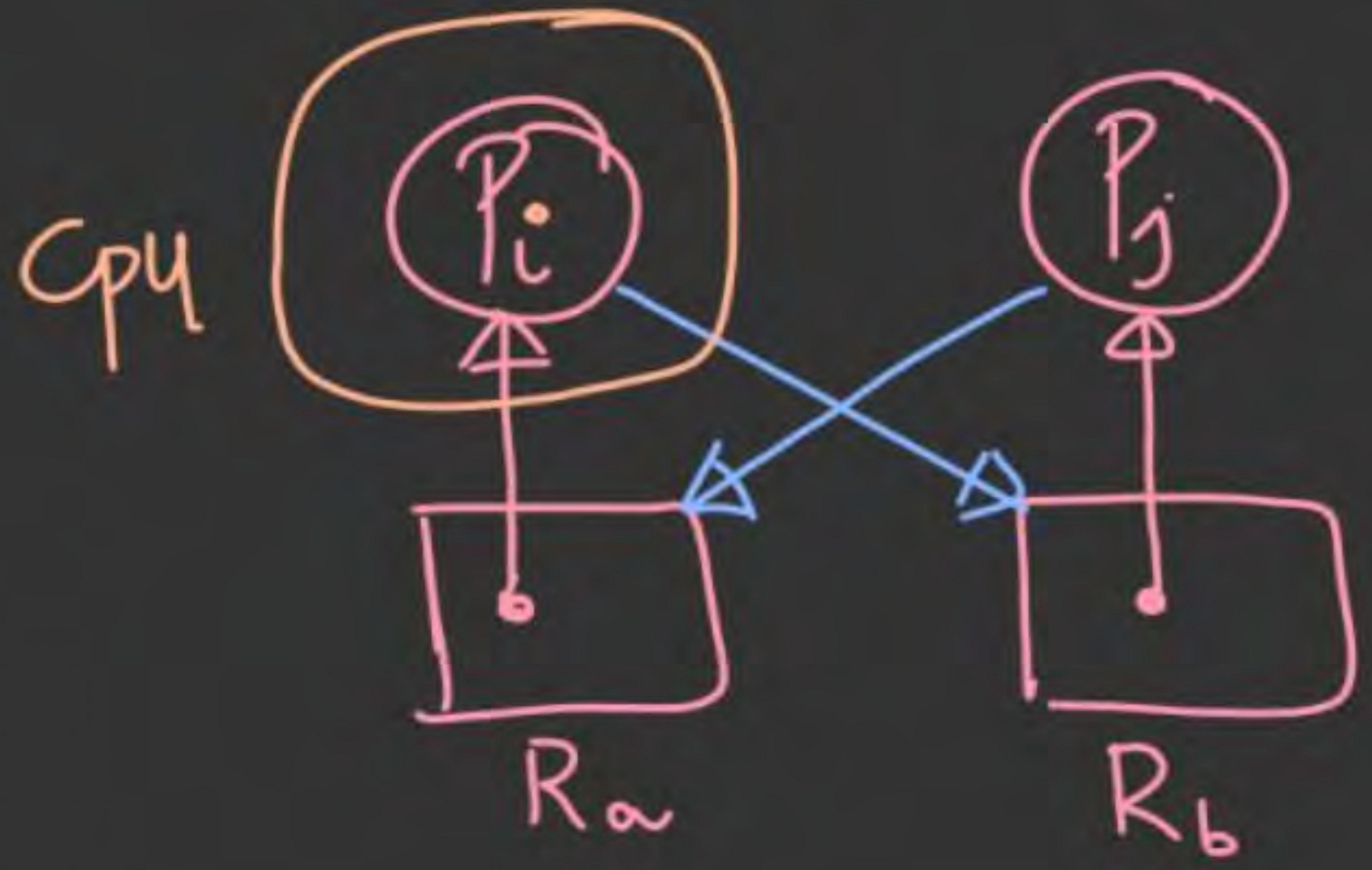
(ii) Inefficiency;

b) Process must release all the resources b/f making a fresh request;

– Starvation



(iii) !(NO-PreEmption):



Pre Emption

- forcible (Selfish)
- Self (Selfless)

Starvation

(10) Circular-wait:

a) Number all resources uniquely

b) Never allow a process to request a lower numbered resource than the last one requested.

Res. Name	Rid
A →	8
B →	5
C →	10
D →	2
E →	12
F →	15

<linear order>

<Starvation>

D B — A C

2 — 5 — 8 — 10

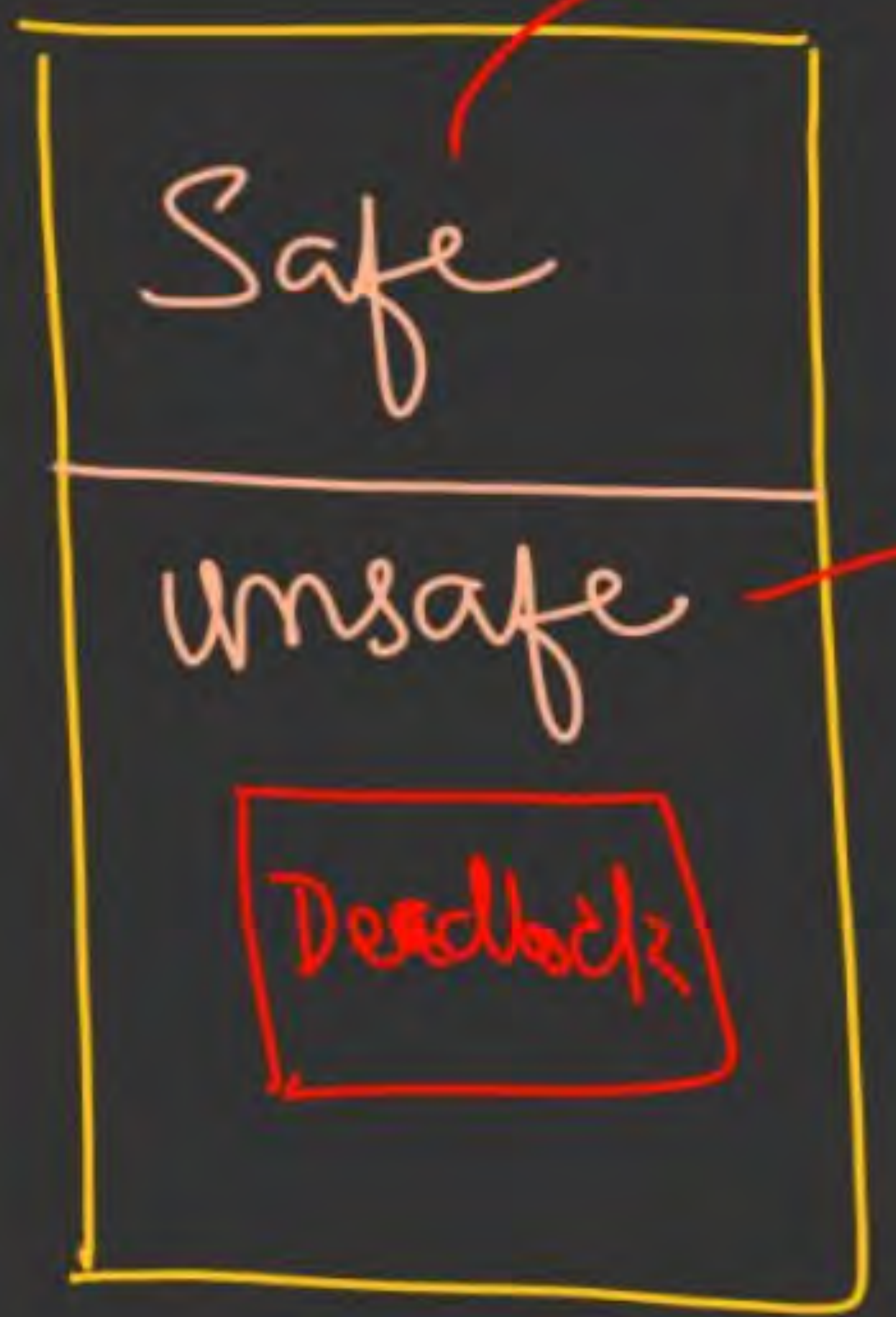
Deadlock prevention

- To design a system in such a way that the possibility of a deadlock is excluded a priori.
- Prevention philosophy: We know what the preconditions are; So prevent one or more these from occurring.
- For example: Circular wait can be prevented by linear ordering of the resource types. If a process holds resources of type Rj , then it can request resources of type Rk, k > j, but not Ri where i <= j. Similarly, any other process holding Ri can request Rj but a process holding Rj cannot request Ri.

III. Deadlock Avoidance: <Banker's Algorithm>

(i) Safety Algorithm (ii) * Resource-Request Algo.

System State:



→ No Deadlock

→ Warning

(Safety Algo)

"The main o
of Banker's A
is to alway
operate the
System i
Safe m

System Parameters that define its State ;

'n' : no. of Processes

m : no. of resources

Maximum $[1..n, 1..m]_{n \times m}$

$Max[i,j] = K$

$P_i \xrightarrow[max]{} K(R_j)$

Allocation $[1..n, 1..m]_{n \times m}$

$Alloc[i,j] = a$

$P_i \xleftarrow[alloc]{} a(R_j)$ $[a \le K]$

(v) $Need[1..n, 1..m]_{n \times m} = \underline{Max - Alloc}$

$Need[i,j] = b$ $\underline{K - a}$

$P_i \xrightarrow[Need]{} b(R_j)$

(vi) $Request[1..n, 1..m]_{n \times m}$

$Req[i,j] = e$;

$P_i \xrightarrow[req]{} e(R_j)$ @ time 't' $[e \le b]$

(vii) $Total[1..m]$

$Total[j] = z$ | There are 'z' copies of R_j in System

(viii) $Available[1..m] = Total - \sum_{i=1}^{n} Alloc$

$Avail[j] = y$;

There are $y(R_j)$ free avail @'t'

$A = \frac{Total}{50}$

$\langle P_1 .. P_n \rangle = 35$

$Avail = 15$

1) $n = 5$; $\langle P_1 \dots P_5 \rangle$

$R = 35$ (Total)

Safety Algo: Sys is said to be Safe iff the Need of all Processes can be satisfied with the 'avail' resources in some order

Avail = Total − $\sum$Alloc

$34 - 32 =$

t_0:

Pid	Max R	Alloc R	Need R	Avail R
P_1	12	6	6	3
P_2	6	3	3	~~6~~
P_3	20	12	8	~~10~~
P_4	15	7	8	~~16~~
P_5	8	4	4	~~28~~
		32		$\langle 35 \rangle$

t_1: $\langle P_2; P_5; P_1; P_3; P$

Safe Sequences

Session II : 20/11 Banker's Algorithm :   Total : <A, B, C> = <10, 5, 7>

R.Q	Allocation A B C		Max A B C	Available A B C	Need A B C	
P0	0 1 0	0 3 0	7 5 3	~~3 3 2~~	7 4 3	7 2 3 ×
P1	~~2 0 0~~	3 0 2	3 2 2	<2 3 0>	0 2 0 ×	
P2	3 0 2		9 0 2	<2 1 0> ✓	6 0 0 ×	
P3	2 1 1		2 2 2		0 1 1 ×	
P4	0 0 2		4 3 3		4 3 1 ×	

t_1 : < P_1 ; P_3 ; P_4 ; P_0 ;

T_0 : System is Safe :

T_1 : P_1 : ⟶ <1, 0, 2> : Req_1
↓
is granted ✓

T_2 : P_4 : ⟶ <3, 3, 0> : Req_4 ?
NOT Grant

T_3 : P_0 : ⟶ <0, 2, 0> : Req_0 ? denie

THANK YOU!