CS & IT ENGINEERING

Operating Systems

Deadlocks

Lecture No.01

By- Dr. Khaleel Khan Sir

TOPICS TO BE COVERED

- Deadlock Handling Strategies
- Deadlock Prevention
- Deadlock Avoidance

Deadlock handling Strategies

(i) Deadlock Prevention

(ii) Deadlock Avoidance [Banker's Algo.]

(iii) Deadlock Detection & Recovery [Doctor's Strategy]

(iv) Deadlock Ignorance [Ostrich Algorithm]

Type 1 : Deadlock never occurs

Vs

Type 2 : Deadlock occurs

I. Deadlock Ignorance [Ostrich Algorithm]

: No - Strategy

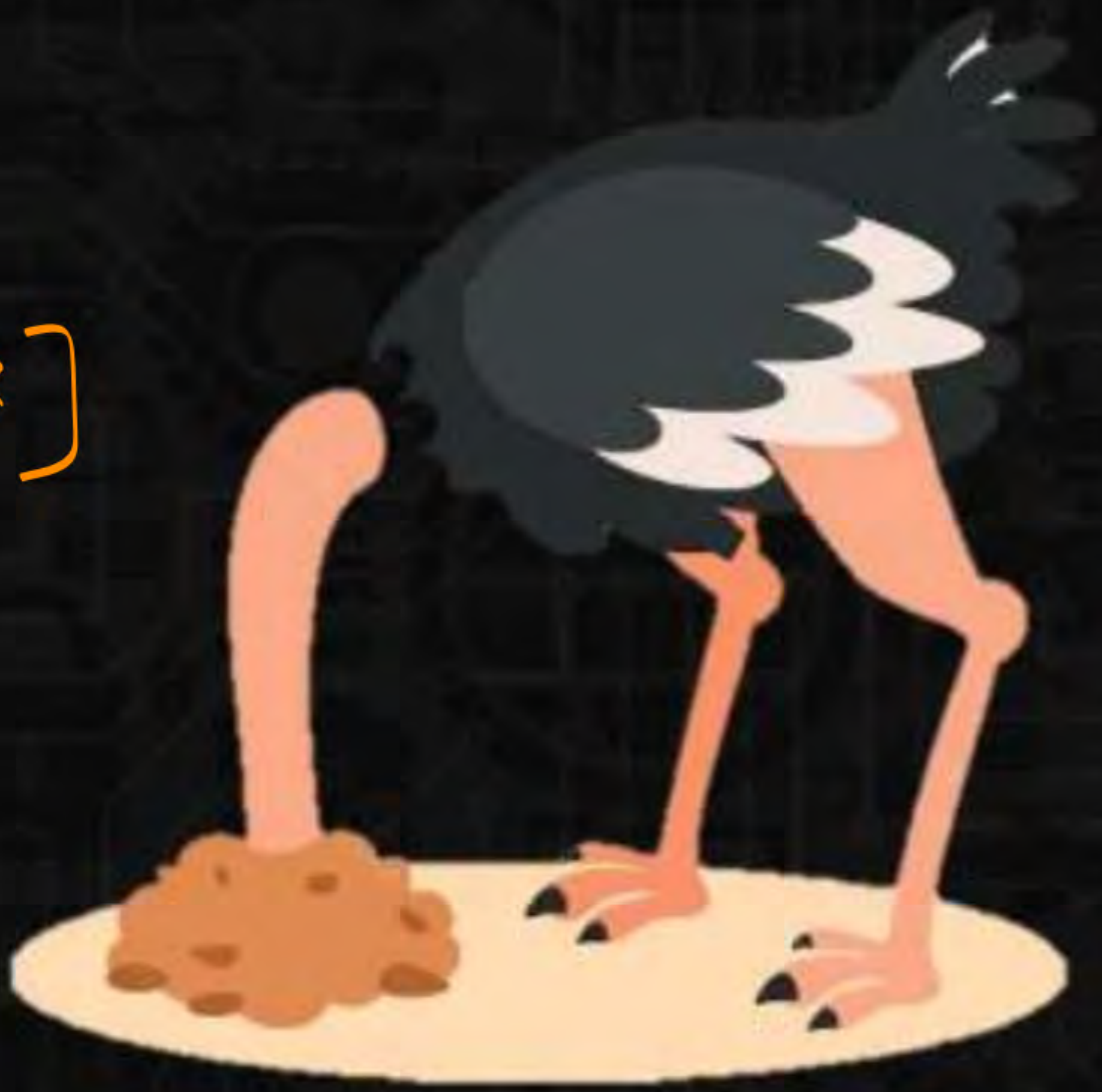
In the Computer System,

applying Ostrich Algorithm means ________;

Restart | Reboot the System;

The Ostrich Algorithm

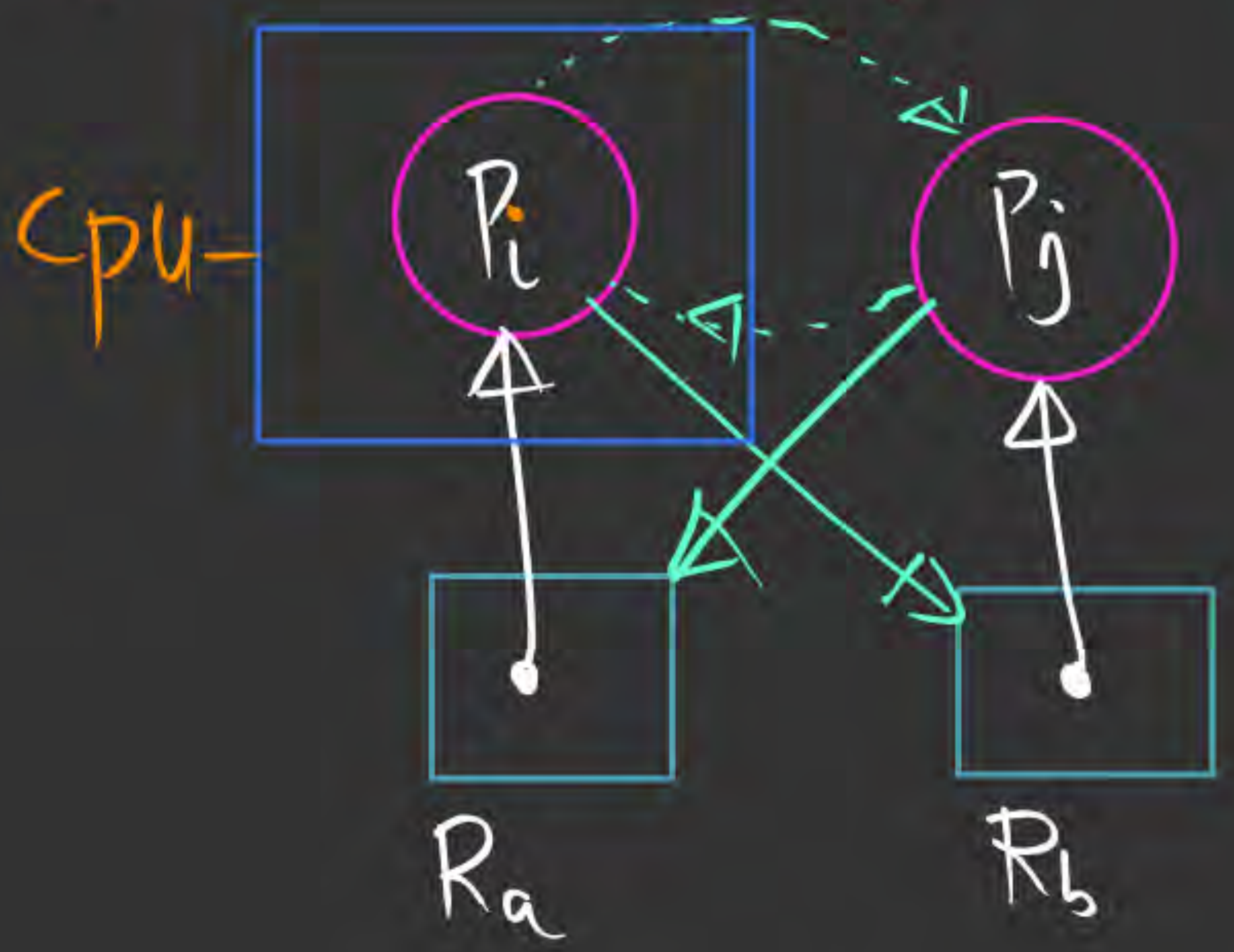
- Pretend there is no problem
- Reasonable if
  - Dead locks occur very rarely [UNIX]
  - Cost of prevention is high
- UNIX and Windows take this approach
- It is a trade-off between
  - Convenience ✓
  - Correctness ✓

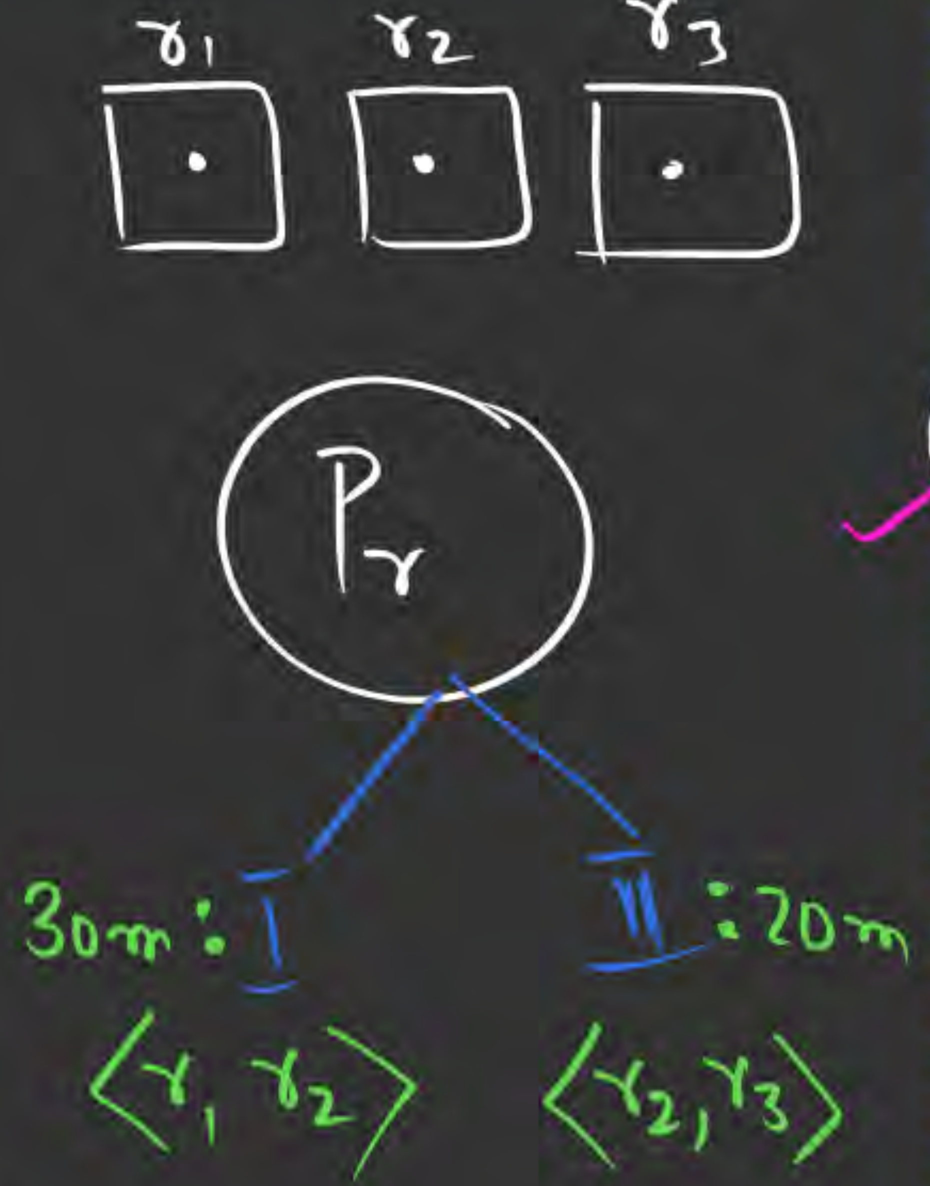


* II) Deadlock Prevention : by Dissatisfying/Negating one/more of the Necessary Conditions;

a) Mutual Exclusion : is non Compromizable/Non dissatisfiable ;

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





(i) Process must Request ALL resources Prior to its execution;

a) Starvation
b) resource Inefficiency

(ii) Process must release ALL its resources, whenever a new request is made;

a) Starvation

3) No-PreEmption : of Resources by/from Processes;

PreEmption of resources,

Forceful (Selfish)

Self (Selfless)

Starvation

4) Circular-wait : Linear / Total order Relation to be implemented

Inc. order

a) Number ALL resources uniquely

b) Never allow a process to request a lower numbered resource than the last one allocated;

→ Starvation

Res		Res-Id
A	→	8
B	→	4
C	→	6
D	→	12 ✓
E	→	20 ✓
F	→	3
G	→	9

P_i

F - ~~A~~ - ~~G~~ - B ✓ A G

3 - 8 - 9 4

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

a) Resource Allocation Graph Algorithm : Single Instance Resource

b) Banker's Algorithm : Multi-Instance Resource

→ Each Process has to give Apriori Knowledge about the resources

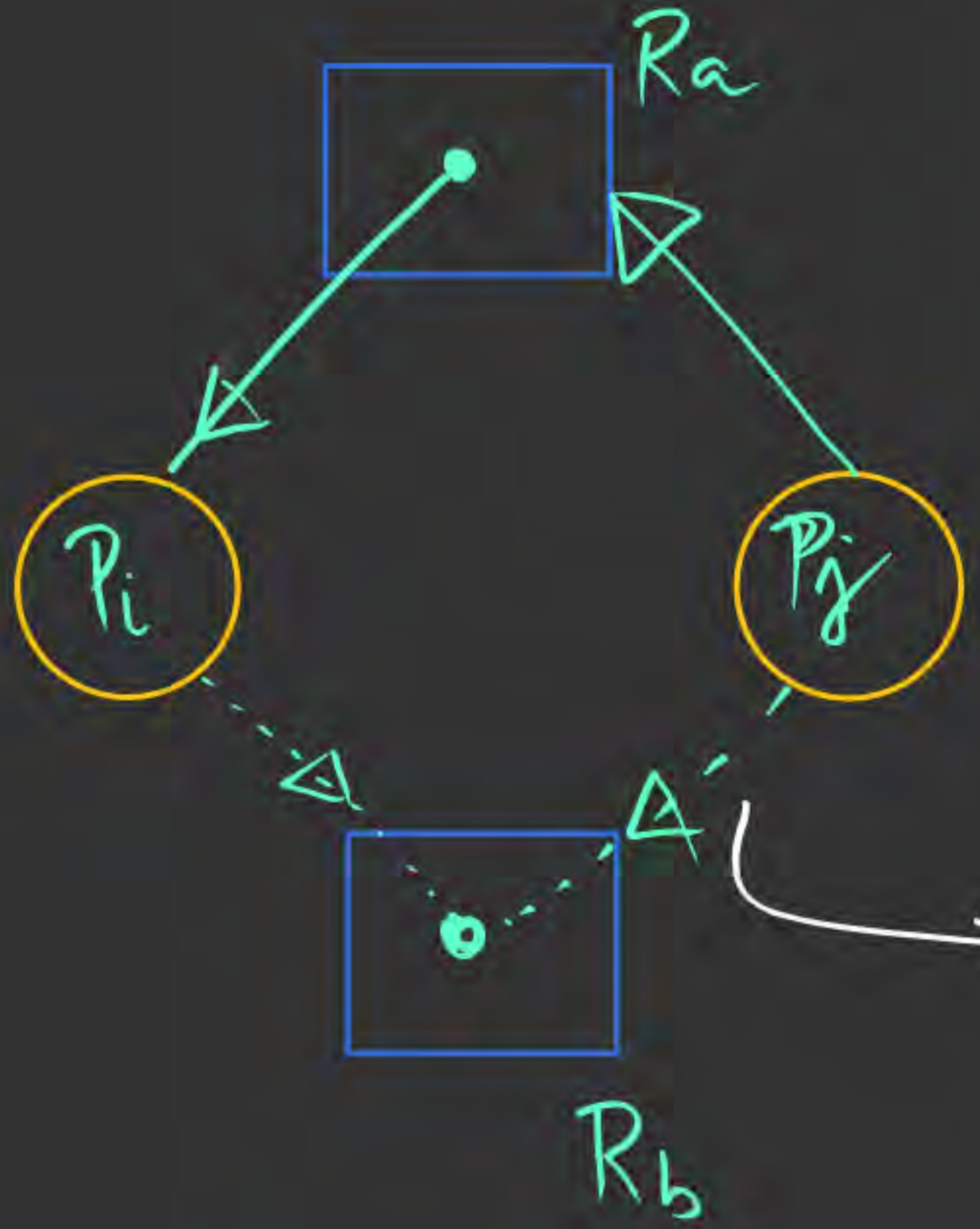
→ O.S → avoids the deadlock by operating the System always in SAFE STATE

a) Resource Allocation Graph Algorithm : <Single Instance Resource>

→ whenever process Starts, it puts claim edges to those resources, which the process anticipates that will be used in future (Apriori Knowledge) ; claim edge is indicated by dotted line ;

→ claim will be converted to Assigned edge only if the resulting State is Safe (i.e there should not be formation of a cycle in R.A.G)

→ If there is no cycle, then the System is Said to be Safe ;



t_1 : Safe

t_2 : < Future >

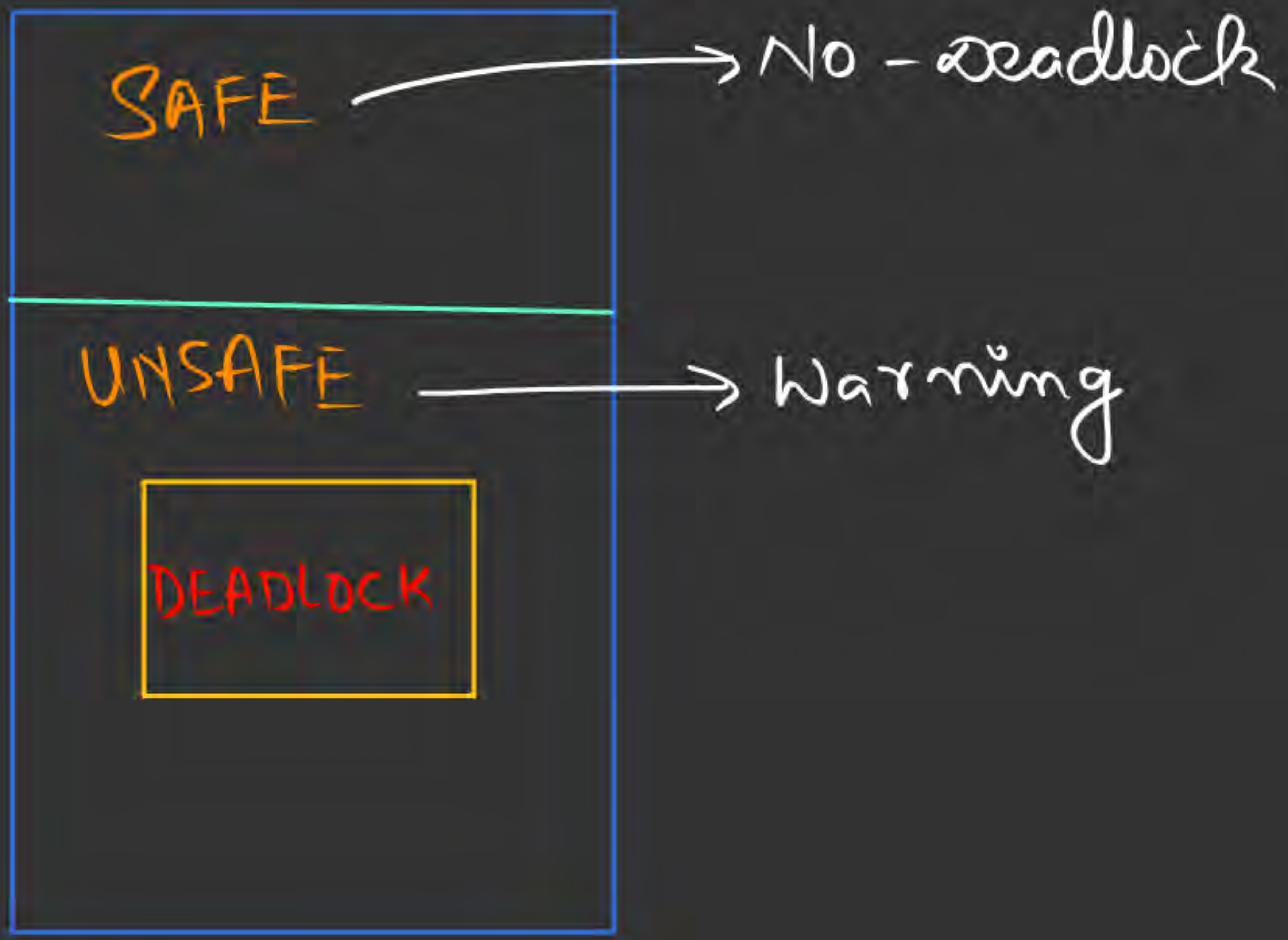
What happens, if Process P_j requests R_b

Converting this claim edge into assigned edge is making the system unsafe

5) Deadlock Avoidance [Banker's Algorithm]

a) Safety Algorithm (b) Resource Request Algorithm :

→ To always operate the System in SAFE Mode/State ;



Data Structures/Parameters for Banker's Algo.

1. 'n' : no. of Processes
2. m : no. of Resources
3. Maximum$[1..n, 1..m]_{n\times m}$;

   $Max[i, j] = \underline{K}$

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

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

   $Alloc[i, j] = \underline{a}$

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

   $[a \leq K]$

5) Need$[1..n, 1..m]_{n\times m}$ = Max − Alloc.

   $Need[i, j] = b$

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

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

   $Req[i, j] = c \qquad [c <= b]$

   $P_i \xrightarrow[req]{} \underline{c(R_j) @ 't'}$

7) Total$[1..m]$

   $Total[j] = z$

   "There are a total of 'z' copies of R_j;

8) Available$[1..m]$

   $Avail[j] = y$

   There are 'y' of R_j free Avail @ time 't'

$$\boxed{Avail = Total - \sum Alloc}$$

Ex 1: $n = 5$; $m = 1$; $R = 27$ (Total)

	Max R	Alloc R	Need R	Avail R
P_1	10	5	5	~~2~~
P_2	15	7	8	~~4~~
P_3	3	2	1	~~8~~
P_4	12	7	5	~~13~~
P_5	8	4	4	~~20~~
		$\Sigma = 25$		27

: Sys is said to be Safe iff the need of all Processes can be satisfied with the Avail resources in some order

t: $\langle P_3; P_5, P_1; P_2; P_4 \rangle$ Safe Sequence

"SAFE"


THANK YOU
GW
SOLDIERS !