Deadlock Avoidance

— Good, (Optimal), Futuristic

└ Implementation — Difficult

— It assumes that the OS is fully aware all resources that a process will use in its life cycle

↘ If Achievable, then implementable

— Algos — System - Matrix

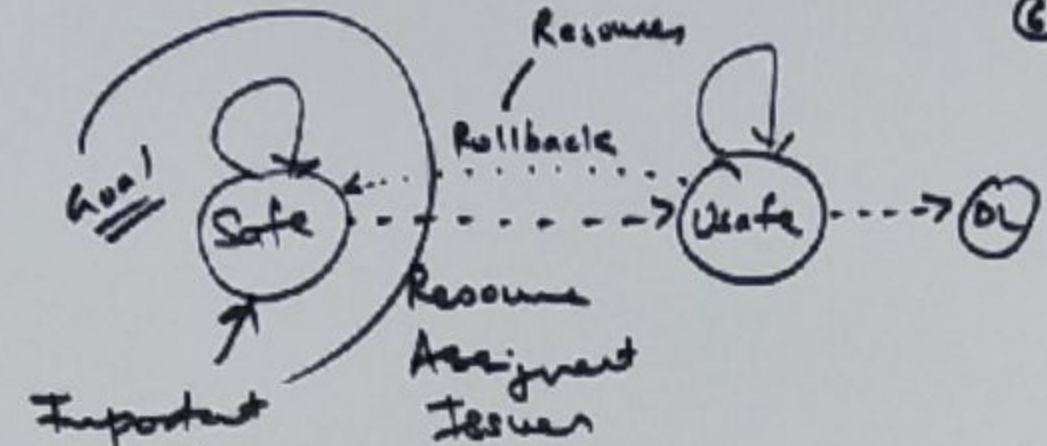
↓

Detect Safety.

→ Safe

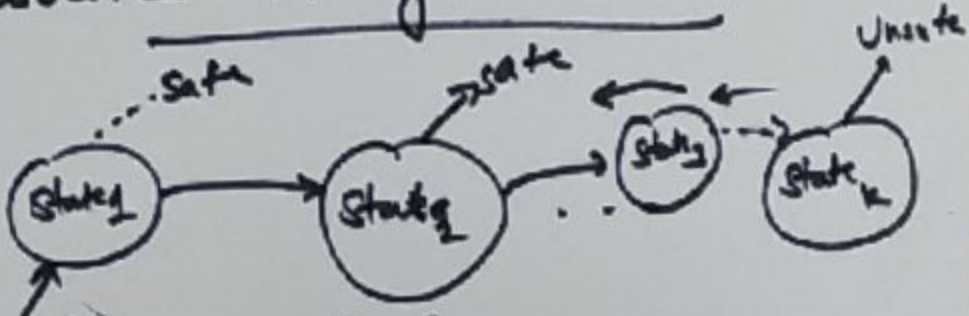
↳ No Deadlock (Avoid)

Unsafe (Any incorrect Resource assigned)



Important aspect of Deadlock Avoidance (Banker's Algo)

Resource Assignment Problem



- Assigning Resources

$P_1 - R_2$

$P_2 - R_3$

- Free Resources P_3

$P_1 - R_1 R_2$ (Free)

$P_2 - R_3 R_4$ (Free)

- Maintains different Matrices

- Resource — System

① R_1 R_2 R_3 ... R_k

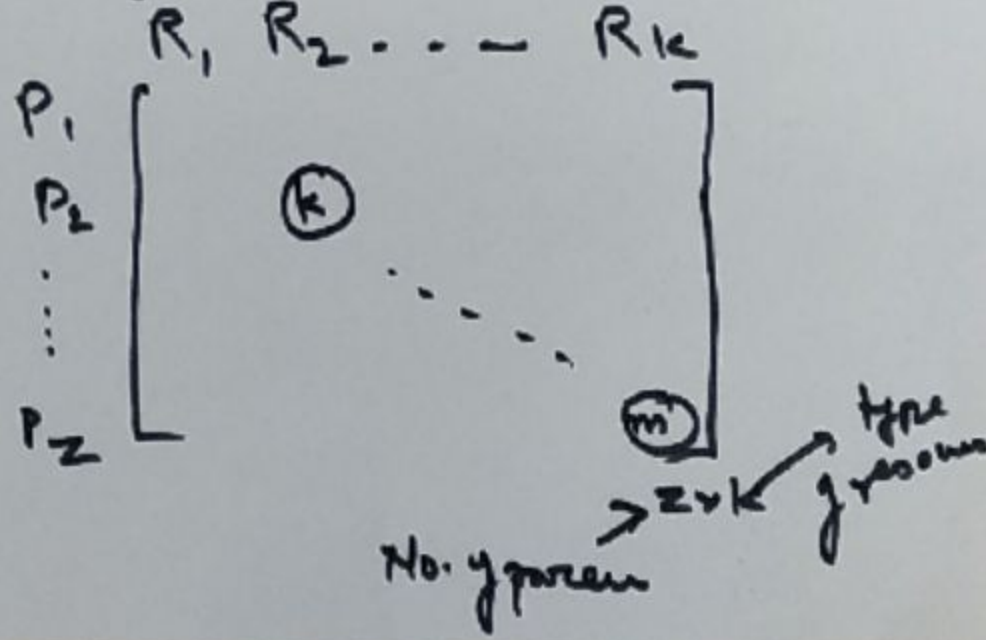
[x y z .... m]

⇓

[1 2 4 ... 1]

1×k → Type resources

② Assignment Resources



③ Need Matrix

	R_1	R_2	...	R_k
P_1				
P_2	0	0	...	0
⋮				
P_2				

- P_2 has received all needed resources
- P_2 can finish and free the assigned resources.

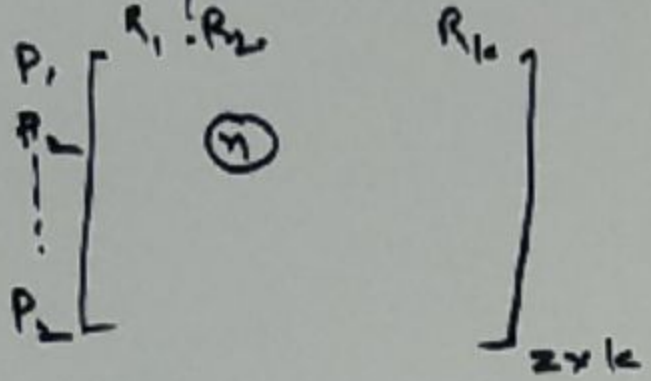
④ Available Resources — Presently Free

Assigned resources

	R_1	R_2	...	R_k
P_1				
P_2				
⋮				
P_2				

— sum up columns

- Algo./System has to be aware of maximum resource requirements for all process.



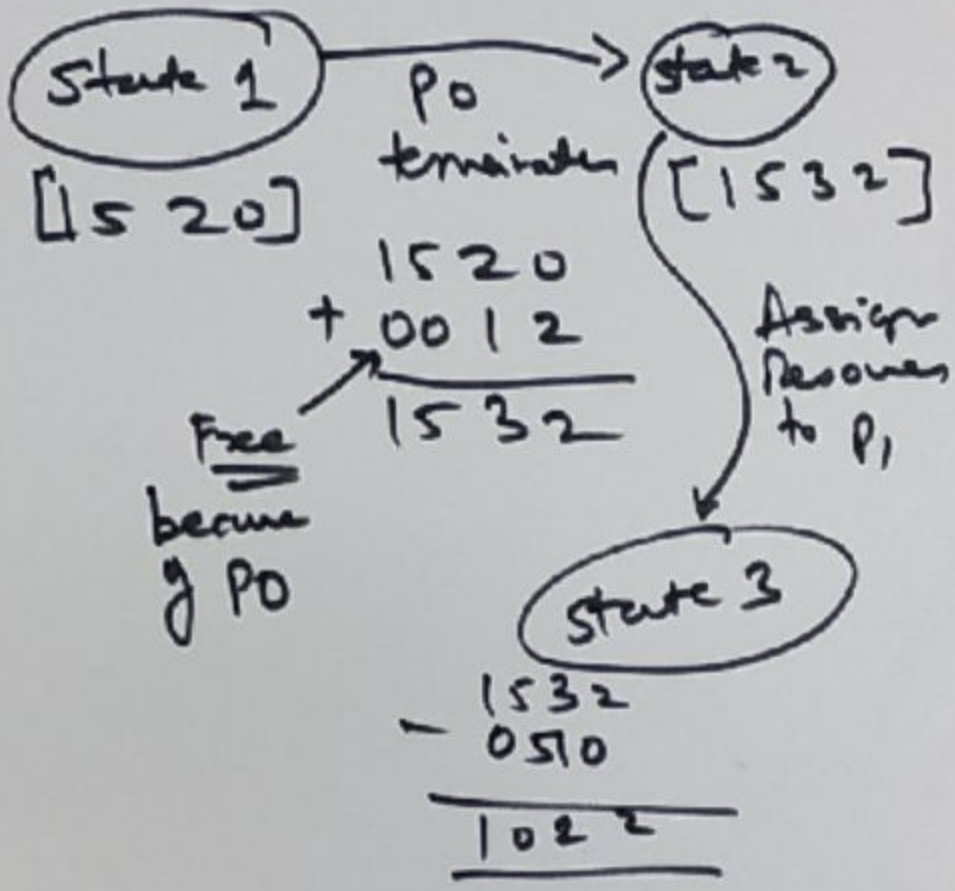
- Every value in the Max. Resource Requirement matrix is the max. res. req. for every process

Need Matrix – represents the resources still needed.

→ Diff - Max Res – Assign. Matrix
2 x 10 2 x 10

For every row in Need matrix,
Compare with currently available
Resource matrix.

↳ atleast one row ⟶



Example: RAG

P_1 P_2 P_3

R_1 R_2 R_3

$P_1 \xrightarrow{\text{Req}} R_1$, $\xrightarrow{\text{Assign}} R_3$

$P_2 \xrightarrow{\text{Req}} R_2, R_3$

$P_3 \xrightarrow{\text{Req}} R_3$ $\xrightarrow{\text{Assign}} R_1, R_2$