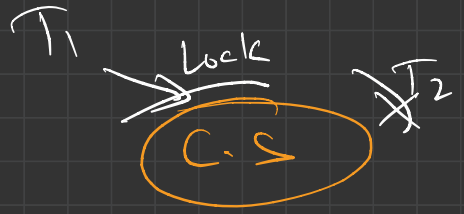
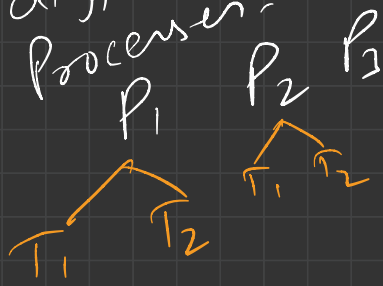
Lec-21

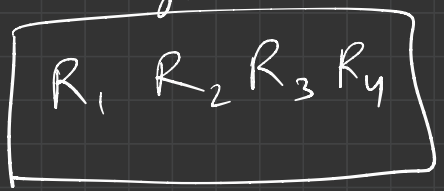
Deadlock



different Processes.



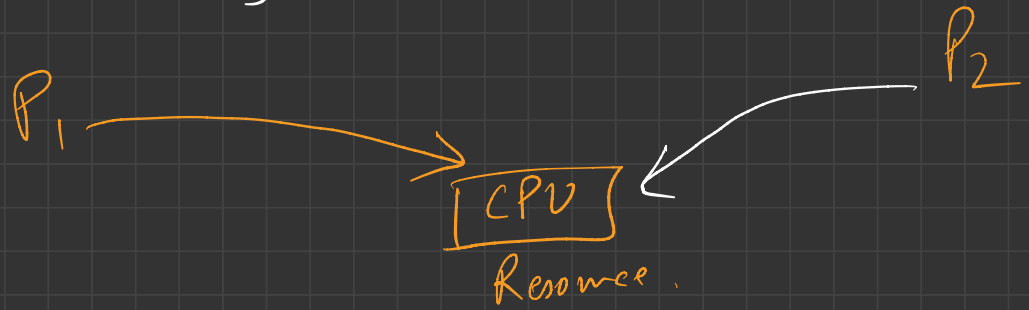
system.



⇒ Finite no. of Resource.

⇒ Multiple Process

→ Memory space, CPU, files, locks, I/O devices etc.



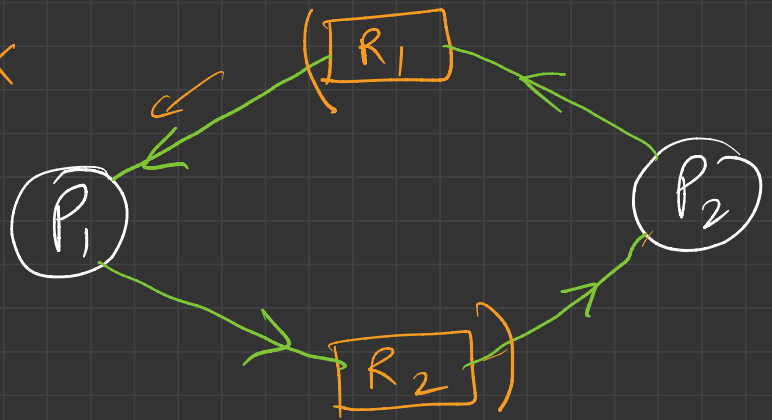
System.

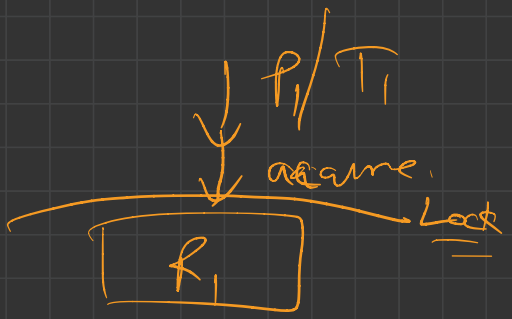
↓

Deadlock

P_1 → Program X

P_2 → Program X.



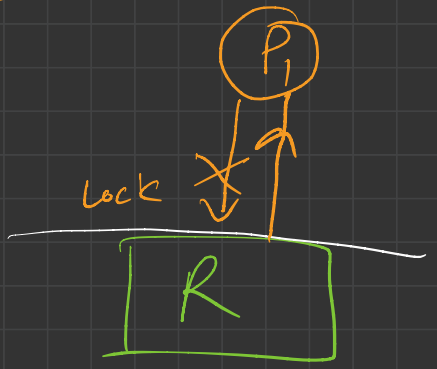


$P_1 \rightarrow R_1 \& R_2 \Rightarrow$

$P_2 \Rightarrow R_1 \& R_2 \Rightarrow$

How a Process/Thread utilize a R ?

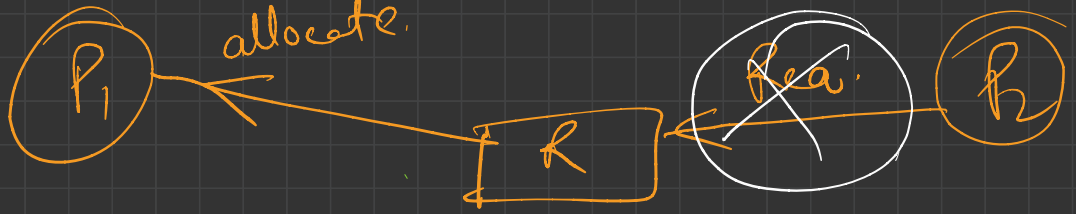
1. Request
2. Use
3. Release.



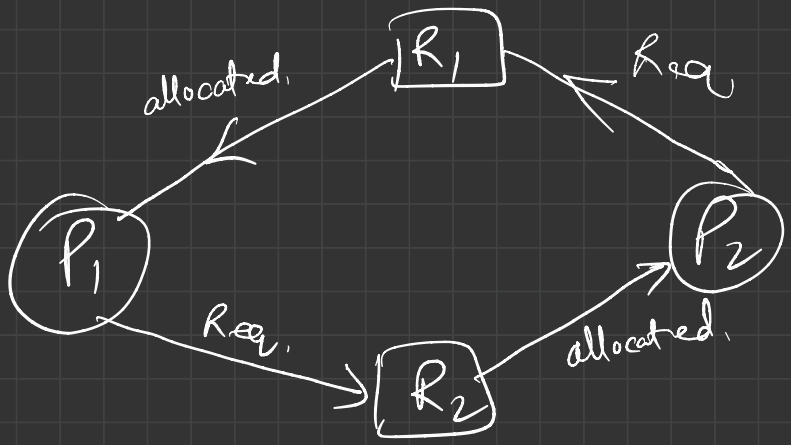
Necessary conditions for DL (4)

(1) Mutual exclusion :-

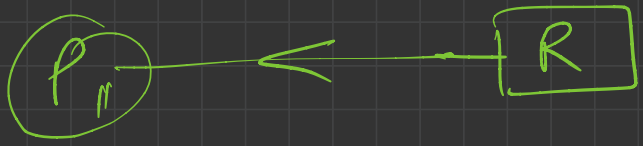
1 R → 1 Process / thread.



② Hold & wait

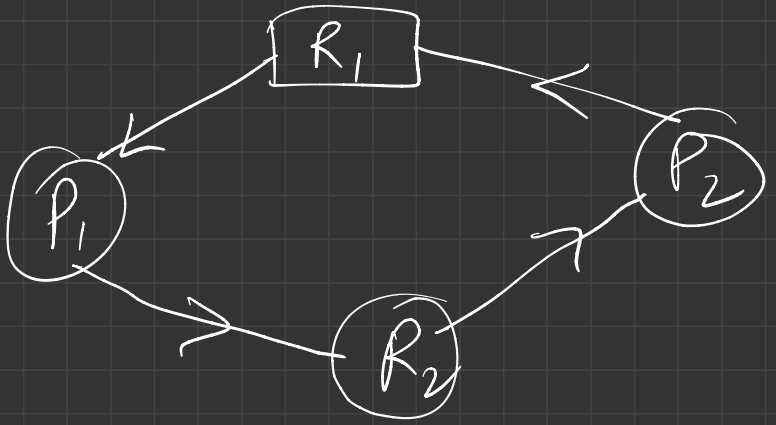


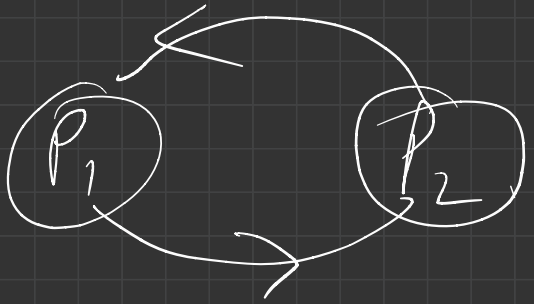
(3) No preemption :-



P_I → execution complete.

(4) Circular wait.





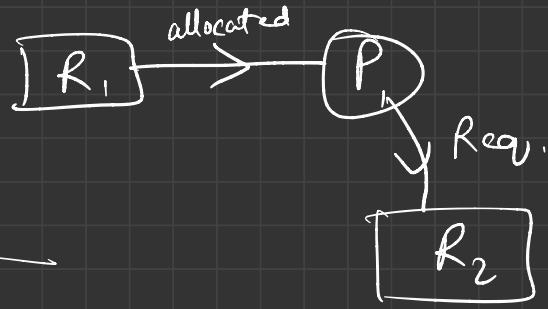
* Resource allocation graph (RAG)

① Vertex → ① Process vertex (P)

② Resource vertex (R)

② Edges → ① Assign

② Request



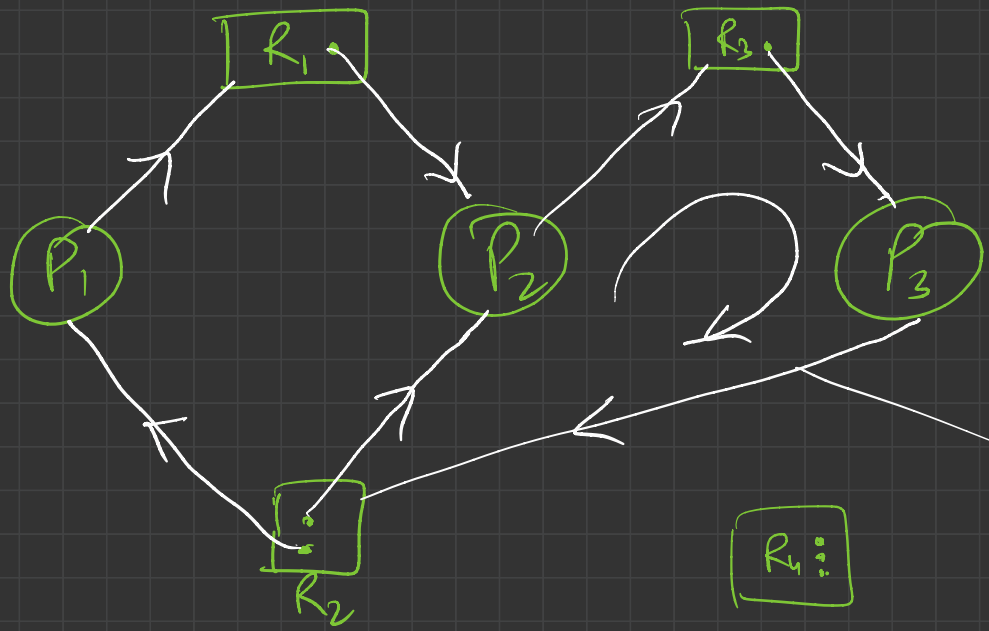
Multiple instance → R (...)

CPU → 4 (CPU)

R

→ System Representation

(1)



By definition of RAG

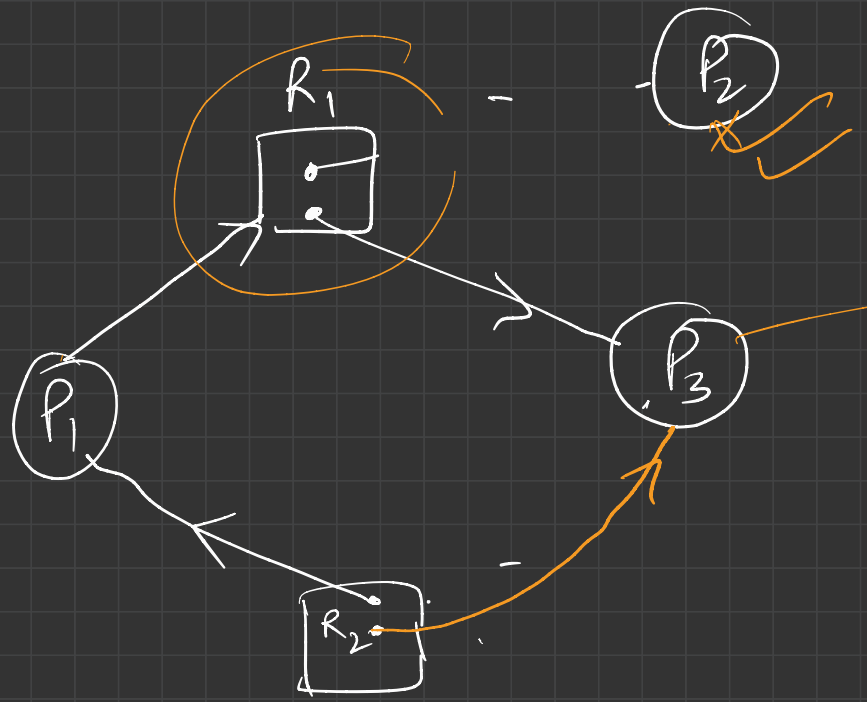
→

① RAG No cycle → No DL

② RAG cycle → may be DL.

DL is present.

(2)



execution succes

t2 0

P4 → R2 → Release

* Methods to handle DL

1. Prevent or avoid DL
2. allow system to go in DL
   - Detect → recover.
3. Ostrich algorithm :- (DL ignorance)
   App. programmers.

* DL Prevention :-

① Mutual exclusion :-

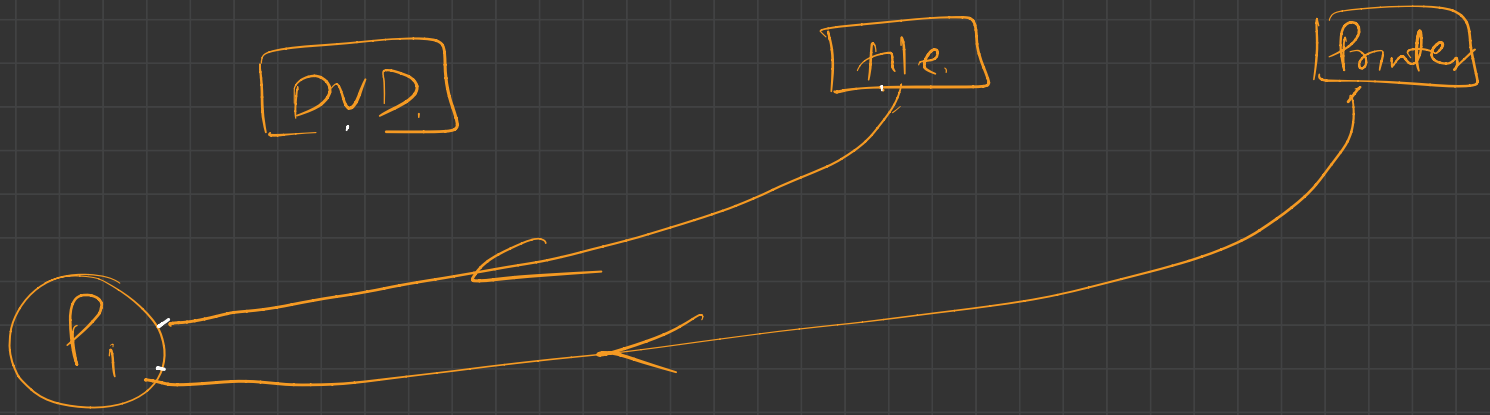
→ non-sharable resources. → mutex.

CS → Memory space

② Read-only file → sharable resource

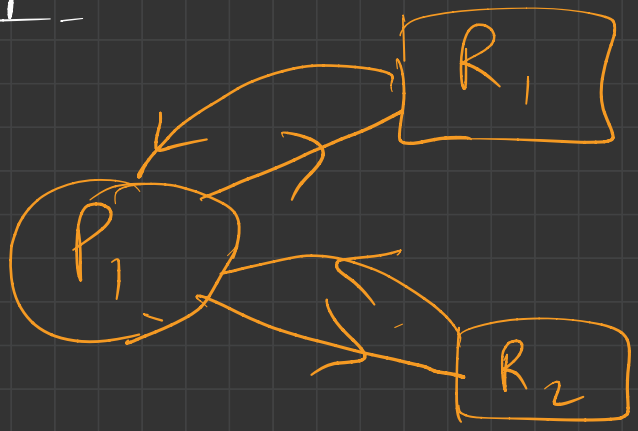
Lock X

(2) Hold & wait :-



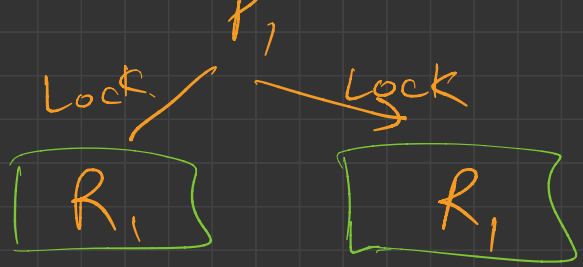
(1) Copy
(2) Sort
(3) Print ⟶ Printer Req.

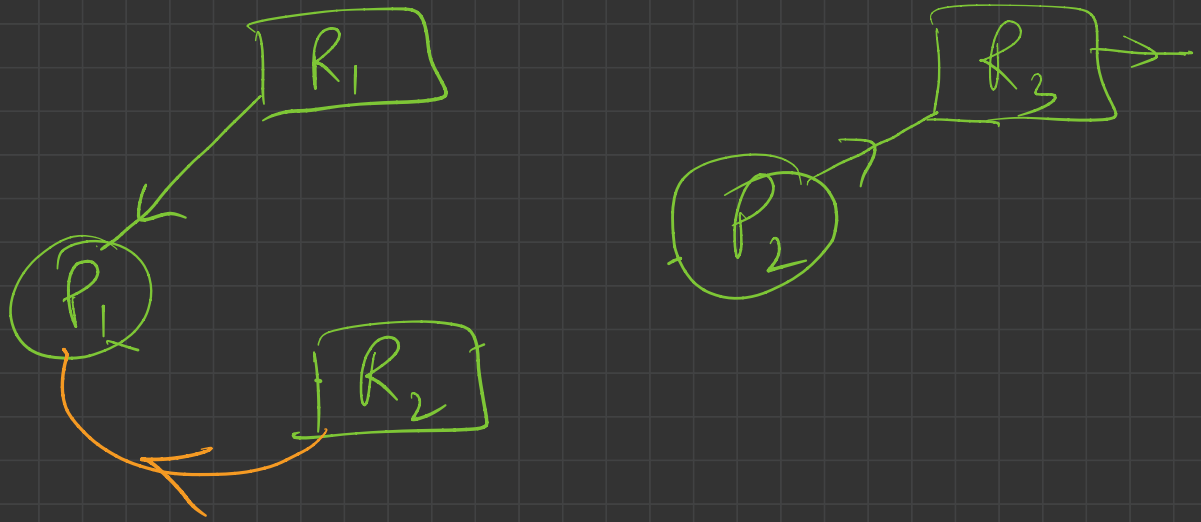
③ No - preemption :-



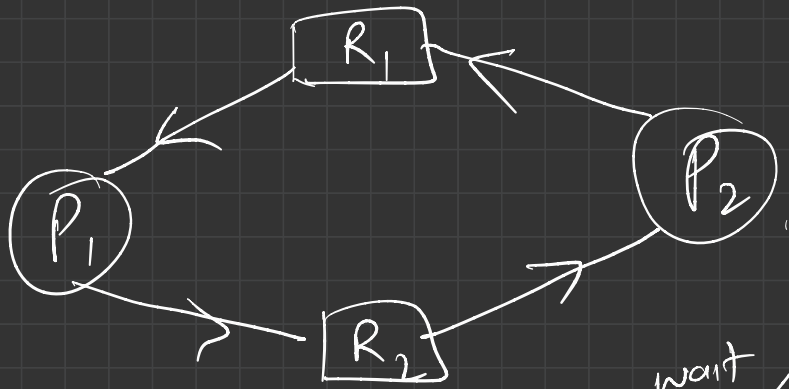
$P_1 \leftarrow R_1$

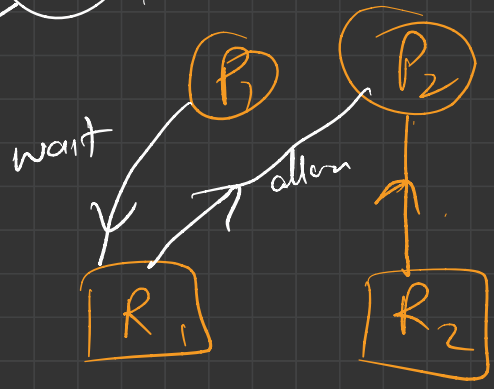
Give Lock




R1
P1
R2
R3
P2

④ Circular wait





Soln ⇒ $P_1 \rightarrow R_1 \& R_2$

$P_2 \rightarrow R_1 \& R$

Order ⇒ $R_1 \xrightarrow{R_1 \text{ already}} R_2$

fixed order