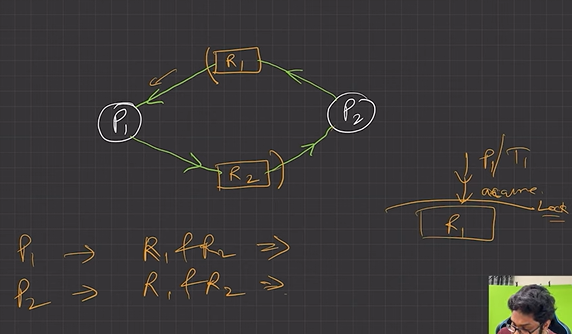
Deadlock Part 1

**Deadlock**

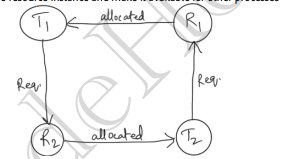
1. CS can be accessed in mutually exclusive way only
2. This can causea problem deadlock
3. There are multiple resources in a system – Memory Space | CPU | Files| locks |I/O devices
4. There are many processes who want to access these resources
5. Fininte processes| multiple processes
6. R1 – R2 | P1 – P2 . P1 acquires R1, P2 acquires R1, Now P1 requires R2 and P2 needs R1. Now this becomes a deadlock
7. Acquires bring lock. Now other process can enter the resource
8. 
9. Process requests a resource (R), if R is not available (taken by other process), process enters in a waiting state. Sometimes that waiting process is never able to change its state because the resource, it has requested is busy (forever), called DEADLOCK (DL)
10. Two or more processes are waiting on some resource’s availability, which will never be available as it is also busy with some other process. The Processes are said to be in Deadlock
11. DL is a bug present in the process/thread synchronization method.

How a process or thread utilize a Resource?

1. Request - check if lock is available or not? If available P1 will lock it
2. Use – P1 will use it
3. Release It will release thelock free

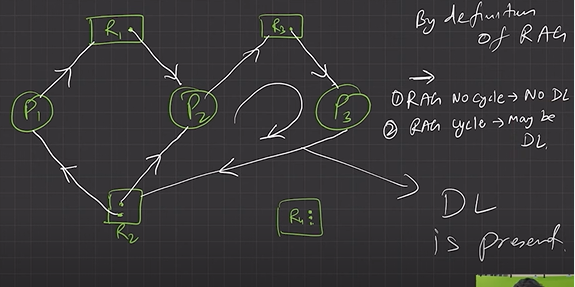
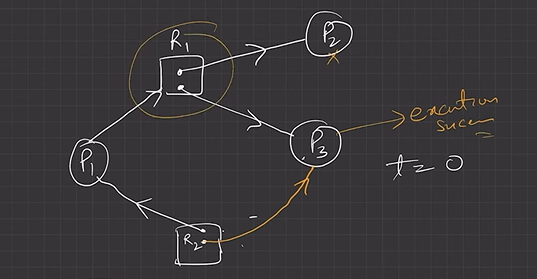
Now some other process can use it

Necessary conditions for Deadlock – 4 Conditions should hold for it to be true

1. Mutual Exclusion: Only 1 process at a time can use the resource, if another process requests that resource, the requesting process must wait until the resource has been released.
2. Hold and wait - A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes
3. No Preemption: Resource must be voluntarily released by the process after completion of execution. (No resource preemption)
4. Circular wait: A set {P0, P1, … ,Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, and so on
   1. 

Now henever these conditions are all true - then deadlock is occurred

\*Resources Allocation Graph(RAG)

1. Vertex
   1. Process Vertex – P
   2. Resources vertex – V
2. Edges
   1. Assign Ri ->- Pi
   2. Request Pi ->- Ri
3. Instances:
   1. ….R for multiple instances
   2. R is for Single Resource
4. Nomenclature:
   1. If there is a cycle, in RAG, there may be a deadlock
      1. Deadlock present
      2. Deadlock not present if P4 gets released
         1. 
   2. No cycle no deadlock

Methods to handle deadlock

* + - 1. Prevent or avoid deadlock
      2. Allow system to go in deadlock,
         1. Detect and then recover
      3. Ostrich Algorithm:
         1. Deadlock ignorance
         2. Ignore the problem altogether and pretend that deadlocks never occur in system. (Ostrich algorithm) aka, Deadlock ignorance.

Deadlock prevention techniques:

: by ensuring at least one of the necessary conditions cannot hold.

1. Mutual Exclusion:
   1. Use locks (mutual exclusion) only for non-sharable resource
   2. Sharable resources like Read-Only files can be accessed by multiple processes/threads
   3. However, we can’t prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable
2. Hold & Wait:
   1. To ensure H&W condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it doesn’t hold any other resource.
   2. Protocol (A) can be, each process has to request and be allocated all its resources before its execution.
   3. Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.
3. No pre-emption:
   1. If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are preempted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (Live Lock may occur).Put sleep between both allocation
   2. If a process requests some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resource from waiting process and allocate them to the requesting process.
4. Circular wait:
   1. To ensure that this condition never holds is to impose a proper ordering of resource allocation.
   2. P1 and P2 both require R1 and R1, locking on these resources should be like, both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.