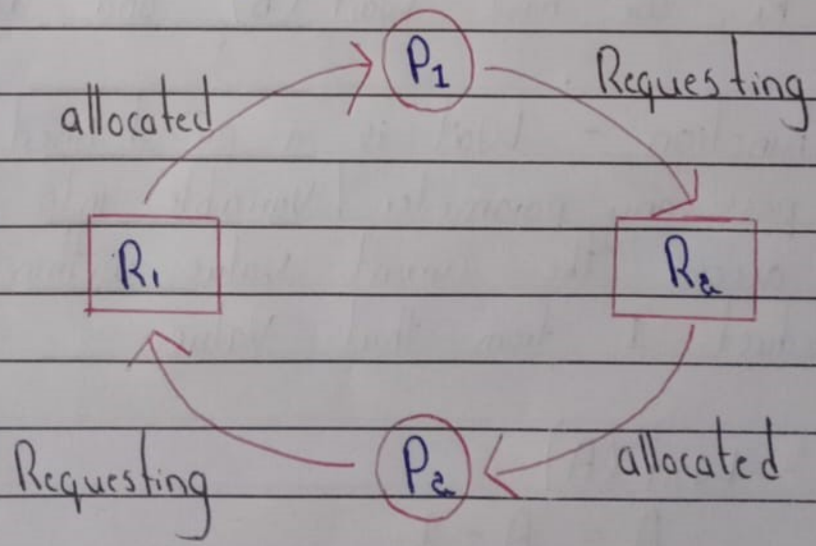
Dead Lock

* In the system we have multiple processes If the processes are blocked indefinitely bcz ~~the~~ of the resources are not available, then we say we have a deadlock Scenario.

* There are limitted no. of resources and the processes are always trying to the access the limitted no. of resources and do the task which are allocated to each and every process.
* So, whenever one process is using a resource, the other process may not be able to access that same resource. bcz already one process is using it.

* While using a resource by a process it can ask for another resource as well. So at that type of a scenario we say that we have a dead lock situation.



Resources are not available bcz resources are already allocated for another process. Once that process execution is over, only that resource will be released from that process.

* Then we ~~at~~ can allocate that resource to a process that is waiting to use that resource.

Example 02

* Semaphores - Semaphores are like ~~t~~ Variables.

* There are two semaphores A and B which are initialized to 1.

$$A = 1 \qquad B = 1$$

* There are two process P_0 and P_1.
* Under P_0, we have Wait (A) and wait (B).
* Under P_1, we have wait (B) and wait (A).

* Wait Function - Wait is a pre-defined function and if we pass any parameter / Variable into this function it will access the current value within the Variable and deduct 1 from that Value.

Ex :- Wait (A)

$$A = A - 1$$
$$A = 1 - 1$$
$$A = 0$$

$A = 1$	$B = 1$
P_0	P_1
Wait (A)	Wait (B)
$A = A - 1$	$B = B - 1$
$A = 1 - 1$	$B = 1 - 1$
$A = 0$	$B = 0$
Wait (B)	Wait (A)

* The second line of P_0 is wait (B). Now when we try to call ~~p~~ wait (B), ~~it~~ the B is equal to 0. So we cannot decrement 1 from 0. Then this process 0 will wait, ~~th~~ until B becomes 1.

* When we consider about P_1, the second line is Wait (A) The semaphore A is also 0. So it cannot call wait on 0, bcz we cannot decrement 1 from 0. ~~7~~ ∴ process 1 is also waiting until A becomes 1.

* In this scenario P_0 and P_1 both are blocked since the semaphores are not available

* There are some real world examples for Dead Lock situations
  (01) Four way Junction
  (02) Narrow Bridge

System Model

* In here we are discussing how we are going to model the deadlocks.

* Identical - In any organization they are 6 printers. ~~and~~. Printer is the resource. Now we have 6 identical resources (Printers). ∴ there are 6 different instances of the same type.

*

* Pre-emptible - At any point we can stop the process and remove the resource allocated to it and that resource can be allocated to another process.

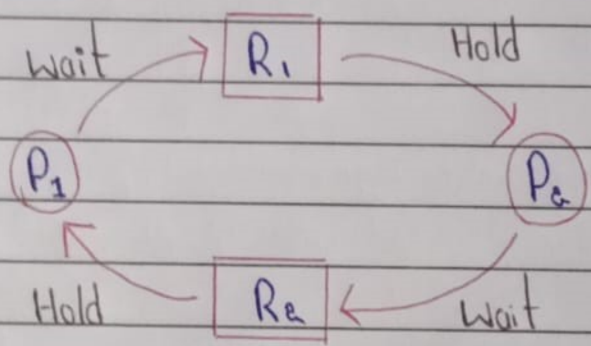
* Non-preemptible - Once you have & start to use the resource by a process, the operation must be completed inorder to release that resource from the process. The resource cannot be released in the middle of the execution.

Ex :- Printer

Necessary Conditions for deadlock

(01) Mutual exclusion Condition - At a time only one process can use the resource. We cannot have resources that can be shared. ∴ the resources cannot be shared.

(02) Hold and Wait Condition - This condition says, One process should hold a resource and wait for another resource. The process currently using a resource and requesting another resource.



- P_1 is holding the R_2 at the moment and it is waiting for R_1 as well.
- P_2 is holding the R_1 at the moment and it is waiting for R_2 as well.

(03) No pre-emption Condition - When we consider about resources that cannot be released immediately ~~Once it is~~ and once it is needed by another process, that process should wait until the other process releases the resource.

(04) Circular Wait Condition - This condition inclined with Hold and wait condition. When we have hold and wait condition we will have this Circular weight.

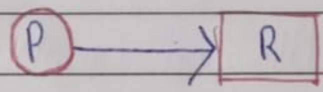
Dead Lock

Dead Lock Modeling

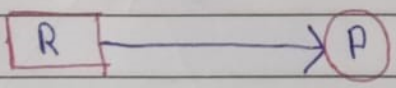
* Vertices → under Vertices we can have,
  1. processes
  2. Resoureces

* Edges → There can be two types of edges.

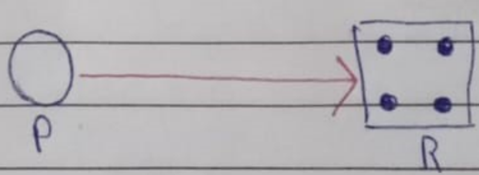
1. Request edge - If you have an arrow from the process to the resource that is a request edge



2. Assignment edge - If you have an arrow from the resource to the process, it is an assignment edge.



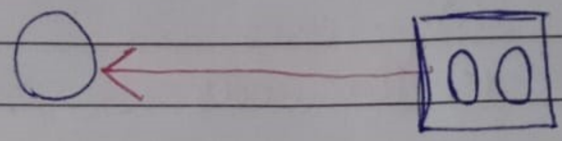
Model Symbols



* From the process, there is an arrow to the resource.

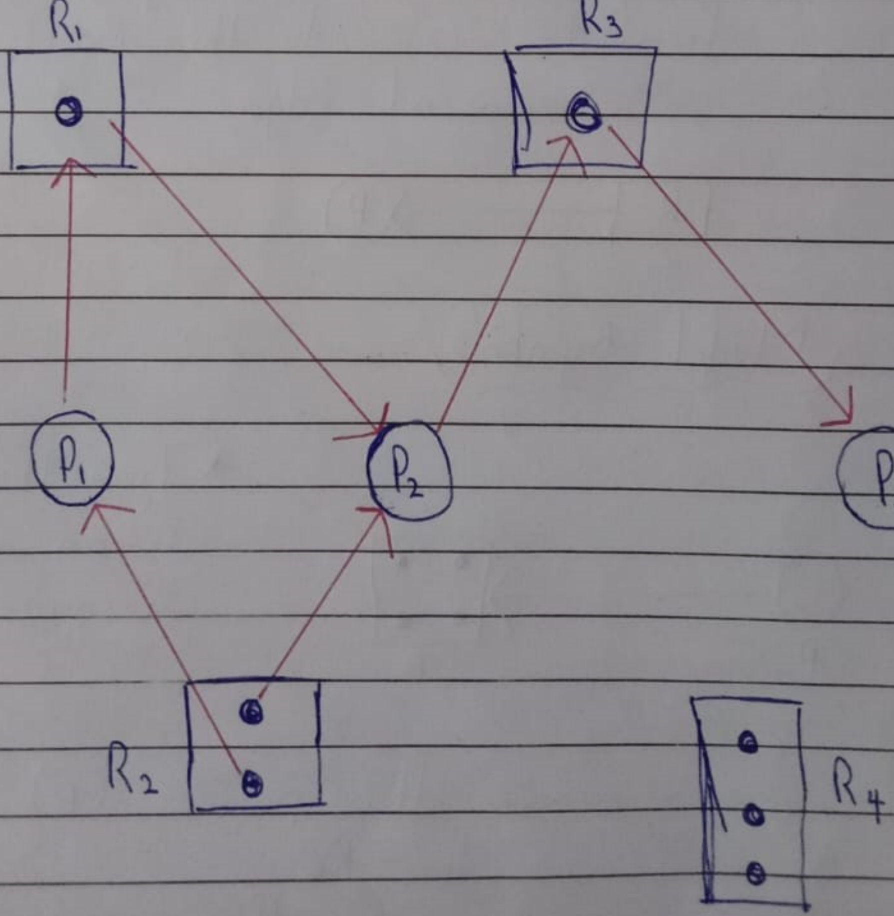
* It means ~~a process~~ the process (P) is requesting a resource from the R.
* There are 4 instances in the resource. ~~When~~
th

* when the process is requesting a resource one of these instances will be allocated to the process.



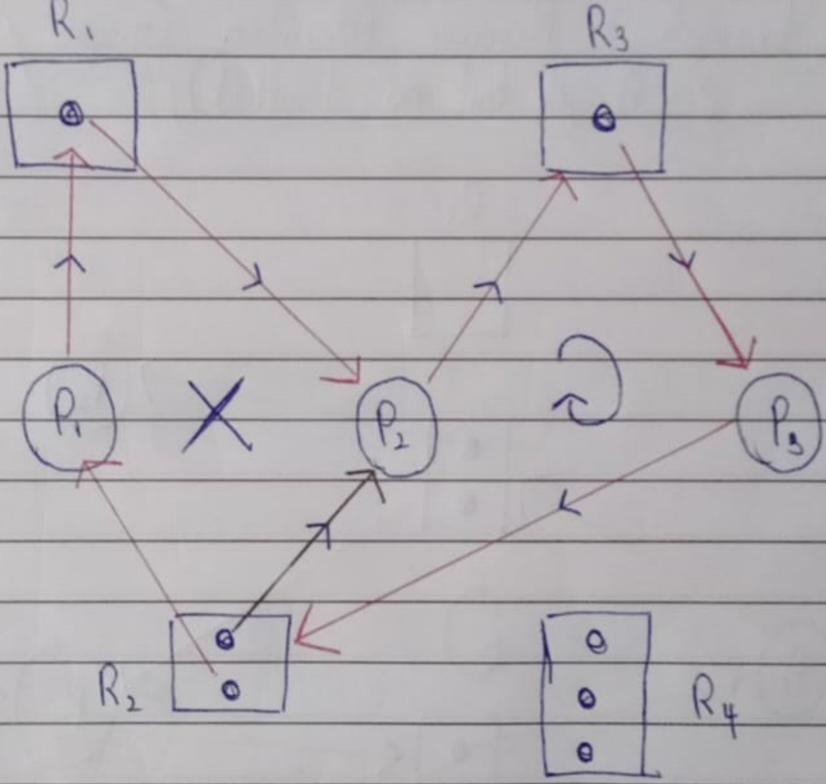
* Here there is an arrow head from the resource to the process. ~~That~~ (Assignment edge)
* It means, one instance of this resource is allocated to this process.

Example :- Resource Allocation Graph (with no cycles)



* ~~When we have resource allocation~~
* Step 01 – Findout whether there is a cycle or not.
    * If all the arrow heads are going on a same direction, we can come up with a cycle.
    * In this example all the arrows are not in same direction. ∴ no deadlock is available

Ex: Example – Resource Allocation Graph (A cycle and deadlock)



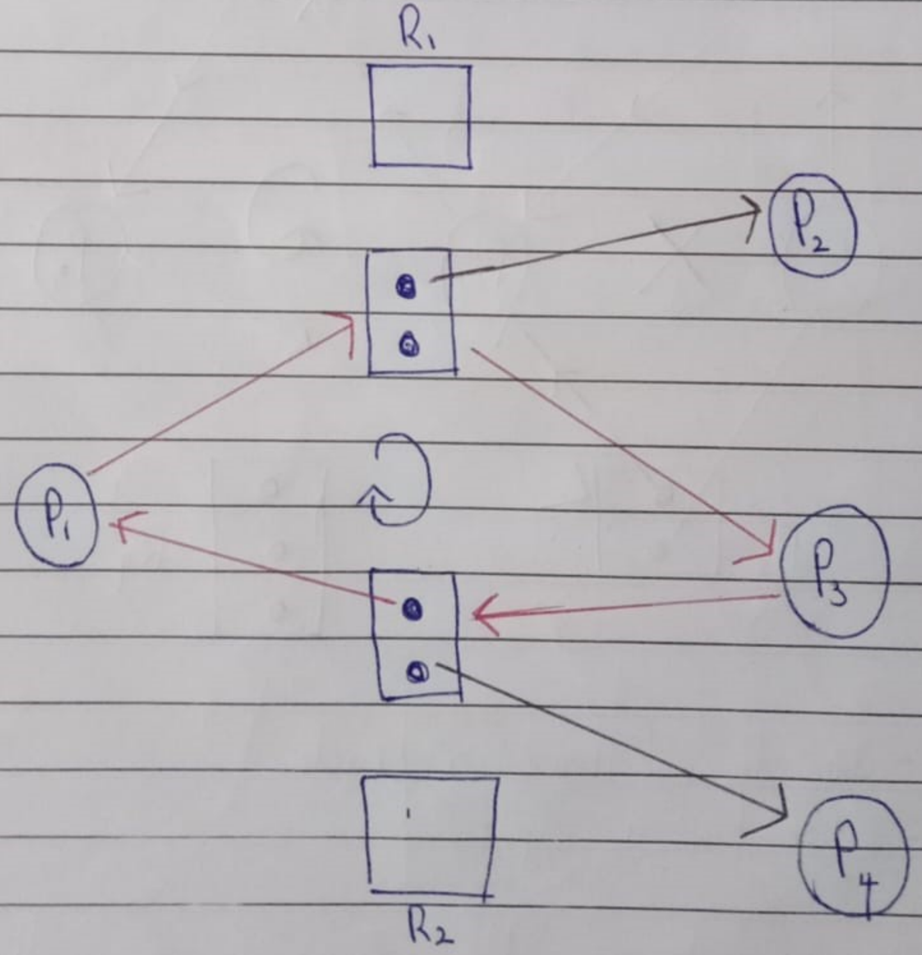
* X – we do not have a cycle.
* ↺ – we have a cycle.

* In this example we have a cycle as well as a deadlock

* R_2 and R_3 are the resources involved in the cycle and P_2 and P_3 are processes that are involved in the cycle.

* R_3 instance is allocated to P_3 and P_3 is requesting an instance from R_2. But R_2 instances are all allocated to P_1 and P_2 processes. So P_3 needs to wait. P_2 is requesting for R_3.
* Here we have a cycle and a deadlock both

Example :- Resource Allocation Graph (A cycle but no deadlock)



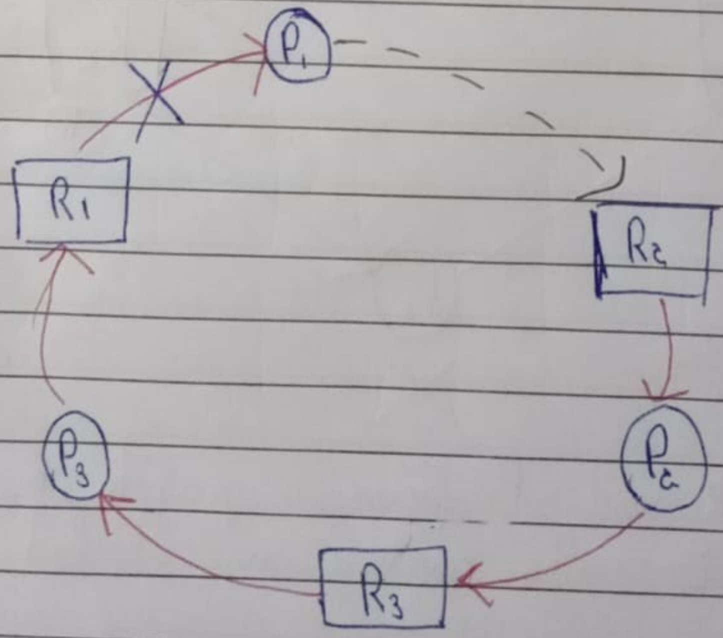
* In this example, there is a cycle.
* In R_1, there are multiple instances and one is allocated P_2 and other one is allocated to P_3. P_2 is not waiting for any other processes.

* In R_2 resource, one instance is allocated to P_1 and other instance is allocated to P_4. P_4 is also not waiting for any other processes.

* Here we have a cycle but we don't have a deadlock.
* Because, at a point P_2 will complete the execution. When P_2 completes its execution the allocated resource to P_2 is released. Then it can be allocated for the process (P_1) waiting for that resource. Then the cycle will break. This is same for P_4 as well.

(1) Deadlock Prevention

(03) Prevent no - premption

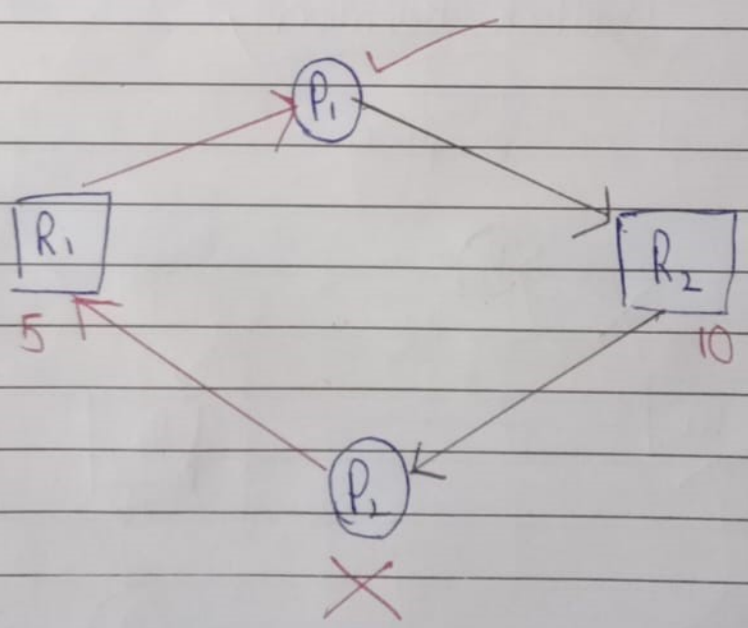


At the moment P_1 is using resouce 01.
R_1 is allocated to P_1.

* If P_1 needs to do a ~~*~~ new request for R_2 it should release the resources which are holding at the moment and wait.

(04) Deny Circular Wait

* A number is assigned to each resource by looking at the resource usage.
* We are always going to check whether the holding resource's resource number is less than the newly requesting resource's resource number.
* That means a process can request a new resource only if the newly requesting resource's resource number is larger than the Currently holding resource's resouce number.



P_1 holds R_1 and P_1 process is going to make a new request for R_2. Now the numbers are checked whether the newly requesting resource's resource number is larger than the Currently holding resource's resource number. ~~The~~ If it is true then this request will be granted

Deadlock Avoidance

Example

	Maximum needs	Allocation	Current need	Available
				3
P_0	10	5	5	+2
				5
P_1	4	2	2	+5
P_2	9	2	7	10
				+2
				12

* As the first step we are going to find out how many resources are ~~the~~ available at the moment.
  1. Get the total of Allocation.
  2. Substract it from the total resources in the system

Available = Total Resources − Current Total Allocation
= 12 − 9
= 3

* Check the current need column and check whether the available resources can be accomadated to the current needs requirement.

* Available resources are 3.
* For the P_0 process 5 resouurcess are needed. So we can't allocate available 3 resources to it. For the P_1 process 2 resouces are needed. Now the available resources can be allocated to this. When you accomadate any request we have to write down the sequence.

$\langle P_1, \ldots, \ldots \rangle \longrightarrow$ Sequence of execution of processes.

* When you accomadate this 2 to P_1 process the current allocation of P_1 process (2) is released That ~~2 of~~ released two should be added to the available Coloumn. Now in the system we have 5 available resources. Now we can accomadate 5 available resources to P_0. Then the current allocation of P_0 will be released. It is again added to the available coloumn.

* Now in the system we have 10 available resources. Now we can accomadate 10 available resources to P_2. The the allocation of P_2 will be released. It is again added to the available coloumn.

* Sequence of process execution should also be updated according ~~th~~ to the accomadating Order

$\langle P_1, P_0, P_2 \rangle$

* With the Available no. of resources we can accomadate all the process requests. ∴ we can say that, no ~~dead~~ deadlock is there and the system is safe.

When One more resource is allocate to P_2

* Allocation coloumn should be edited.
* Also the current need coloumn should be edited.

	Maximum needs	Allocation	Current need	Available
				2
				+2
P_0	10	5	5	4
P_1	4	2	2	
P_2	9	2+1=3	7-1=6	

Available - Total resources - Current total allocation

= 12 - 10

= 2

< P_1 ,

* Now only 4 available resources are there after accomadating for ~~P_0~~ P_1. P_0 and P_2 cannot be ~~allocated with~~ the ~~available~~ accomadated with available resources.

* That means, we cannot find a safe sequence and the system is not safe and there can be a deadlock.

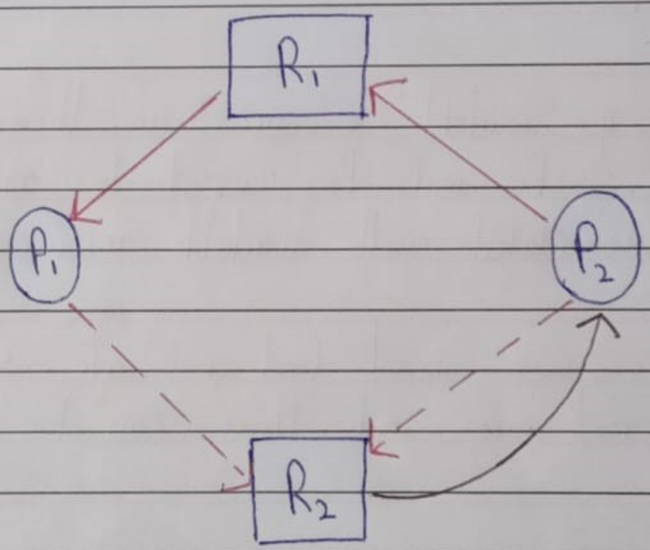
04 SEP 2022

Resource Allocation Graph Algorithm

* Claim edge → Indicates the future requests of the process.
  → Represented by a dashed line.
  → only deal with single instances.

Example

* ~~Suppose P_2 requests R_2.~~



* R_1 is allocated to P_1
* P_2 is requesting from R_1.
* Suppose P_2 is requesting for R_2. Then we have to check what will happen.
  * At the moment R_2 is free.
  * Since it is free it can be allocated to P_2
  * Then cycle detection algorithm should run and check whether there is a cycle or not.

* If there is a cycle P_2 request will be rejected.
* And if there is no cycle P_2 request will be accepted

Banker's Algorithm

* used for multiple instance Resources

Example :- $A = 10$, $B = 5$, $C = 7$

	Allocation A	Allocation B	Allocation C	Max A	Max B	Max C
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3
Total	7	2	5			

Available
(Total Resc - Current total allocation)

	A	B	C
	3	3	2
P_0	+2	0	0
	5	3	2
P_1	+2	+1	+1
	7	4	3
P_2	0	+1	0
	7	5	3
P_3	+3	0	+2
	10	5	5
P_4	0	0	+2
	10	5	7

Need (max - allocation)

	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

$\langle P_1, P_3, P_0, P_2, P_4 \rangle$

* With the Available no. of resources we can accomodate all process requests and we are getting a safe sequence

$\langle P_1, P_3, P_0, P_2, P_4 \rangle$

* ∴ this system is in a safe state and no deadlock will be there.