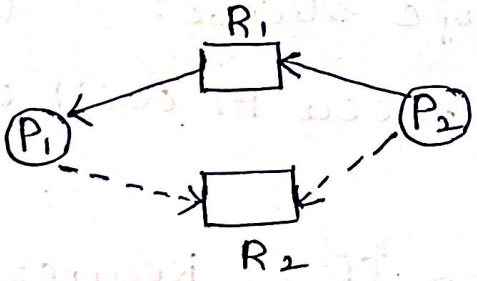
→ Resource-Allocation Graph Algorithm:

- If we have resource allocation system with only one instance of each resource type, we can use the resource allocation graph discussed previously, for deadlock avoidance.
- Along with request and assignment edges it also contain a new type of edge called "claim edge".
- A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in future.
- This edge direction is similar to request edge but is represented by a dashed line.

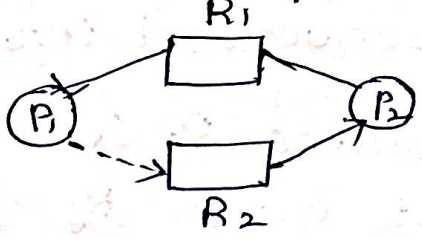


* Resource Allocation Graph.

- When process P_i requests resource R_j, the claim edge is converted into request edge.

→ When Resource R_j is released by P_i, then assignment edge $R_j \rightarrow P_i$ is reconverted to claim edge.

- Now Suppose that Process P_i requests resource R_j. Then request can be granted only if converting the requesting edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in formation of cycle in graph.
- This can be checked by cycle detection algorithm.
- This algorithm requires n^2 operations for detecting cycle, where n is number of process in the system.
- If no cycle exists, then allocation of resource leaves the system in safe state.
- If cycle is found, then allocation leaves the system in an unsafe state.
- In that case, a process P_1 will wait for the request.
- In the above graph if P_2 – Request – R_2 it cann't be as it forms the cycle.



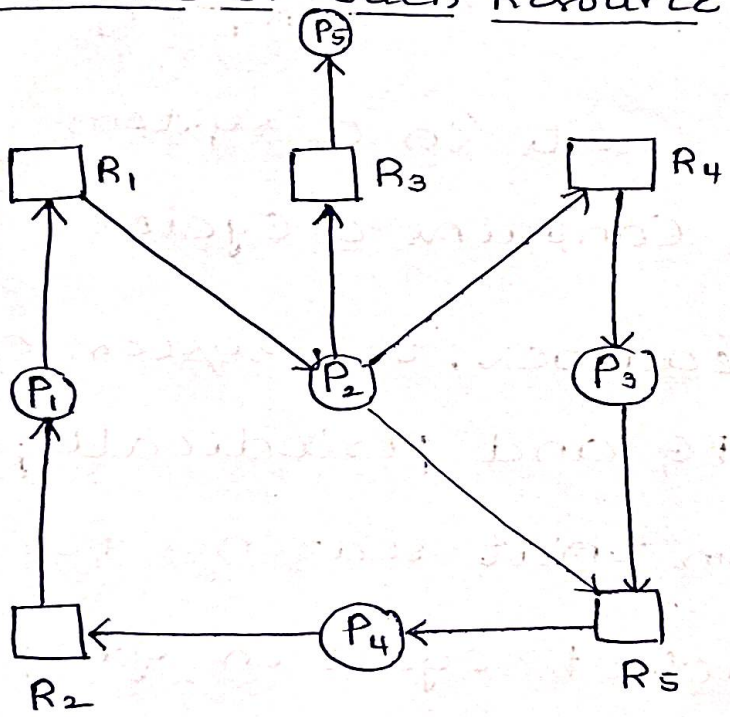
→ Deadlock Detection:

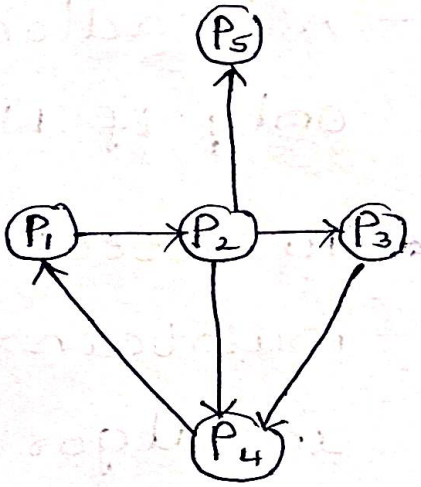
- In deadlock detection, the system may provide:

(1) An algorithm that examines the state of the system to determine whether a deadlock has occured.

(2) An algorithm to recover from deadlock.

Single Instance of Each Resource Type:





Wait for graph.

* Resource allocation graph

- If all the Resources has only one instance, then deadlock detection algorithm uses variant of RAG called wait for graph.

- We obtain this graph by removing resource nodes and collapsing appropriate edges
- In WFG,
  - an edge from $P_i \rightarrow P_j$ means P_i is waiting for P_j to release a resource that P_i needs
  - an edge $P_i \rightarrow P_j$ in WFG exists if

    RAG contains edges $P_i \rightarrow R_q$

    $R_q \rightarrow P_j$ for R_q
- A deadlock exists in a system if and only if WFG contains a cycle.
- To detect deadlock, the system need to maintain WFG and periodically invoke an algorithm that searches for a cycle.
- An alg to detect cycle in graph requires $O(n^2)$ operations, where n = no. of vertices in a graph.

Note: WFG Scheme is not applicable with several instances of resource type.

(b) Several Instances of a Resource Type:

- Algorithm here employs various Data Structures that are similar to those used in "Banker's algorithm".

(1) Available – vector of length m
↓
indicates no. of available resources of each type.

(2) Allocation – n x m matrix
↓
indicates no. of resource of each type allocated to proces.

(3) Request – n x m matrix
↓
indicates current request of each proces

	Allocation			Request			Available		
	A	B	C						
P_0	0	1	0	0	0	0	0	0	0
							0	1	0
P_1	2	0	0	2	0	2	0	1	0
							3	0	3
P_2	3	0	3	0	0	0	3	1	3
							2	1	1
P_3	2	1	1	1	0	0	5	2	4
							0	0	2
P_4	0	0	2	0	0	2	5	2	6

$< P_0, P_2, P_3, P_4, P_1 >$

- Suppose P_2 makes an additional request (0 0 1)

	Allocation	Request	Available
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	0 1 0
P_2	3 0 3	0 0 1	0 1 0
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

$< P_0,$

- Here number of available resources are not sufficient to fulfill requests of other processes: (P_1, P_2, P_3, P_4)
- Thus deadlock exists among P_1, P_2, P_3, P_4.

(C) Detection-Algorithm Usage:

- When should we invoke detection algorithm
- It depends on 2 factors
  (a) How often a deadlock likely to occur?
  (b) How many processes will be affected by deadlock when happened.

Note: Invoking deadlock detection alg will incur overhead in Computational time. Alternative is to invoke / hour or when CPU utilization drops < 40%.

→ Recovery from deadlock:

- When deadlock detection algorithm determines deadlock there are 2 possibilities to deal with deadlocks.

(a) To inform operator to deal it manually.
(b) Let sys recover from it automatically.

- There are two options to break the deadlock.

(a) Abort one or more processes to break Circular wait.

(b) To preempt some resources from one or more deadlock processes

(a) Process Termination:

- 2 methods: In both system reclaims all resources allocated to the terminated processes

(a) Abort all deadlocked processes
- This method break deadlock at greater expense.

- Because deadlock have computed longer time and again they have to compute

(b) Abort one process at a time until deadlock is eliminated.

- Better option.

- Choosing of which process to abort depends on many factors like:

(a) What the priority of process is

(b) How long it has computed and how much time left for computation.

(c) How many and what types of the resources are used.

(d) How many more resources are needed.

(e) Whether the process is interactive/batch.

(b) Resource preemption:

- Here 3 issues need to be addressed.

(1) Selecting a victim – Which resource and which process to be preempted depends

on no. of resources deadlock process is holding.

(2) Rollback: If we preempt a resource what need to be done with the process.

- We must rollback process to some safe state.
- In general, it is difficult to determine what a safe state is. So better to total rollback, abort and then restart the process

(3) Starvation: Resources must not always be preempted from same process

- We must ensure that a process can be rollback for finite number of times