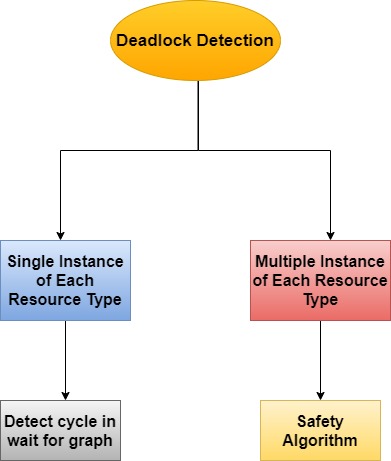
**DEADLOCK DETECTION**

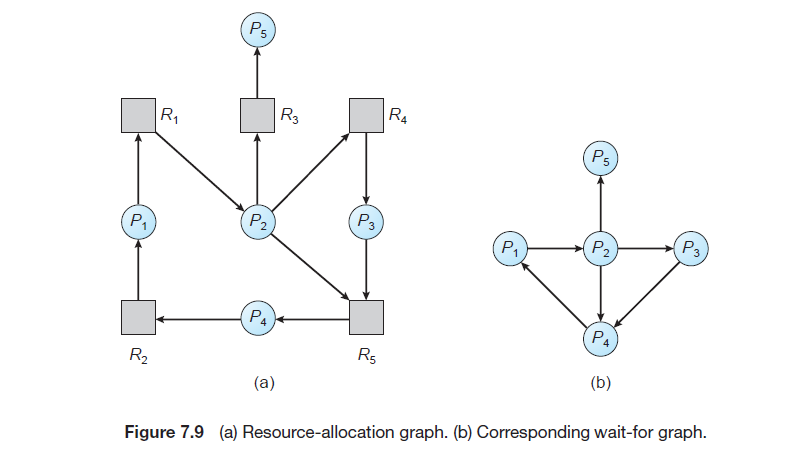
If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In order to get rid of deadlocks, the OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.



**1)Single Instance of Each Resource Type:**

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph. This wait-for graph is obtained from the resource-allocation graph by removing its resource nodes and collapsing its appropriate edges.

An edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi needs. An edge Pi → Pj exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges Pi → Rq and Rq → Pj for some resource Rq . In below Figure, there is a resource-allocation graph and the corresponding wait-for graph.



A deadlock exists in the system if and only if there is a cycle in the **wait-for**graph. In order to detect the deadlock, the system needs to maintain the wait-for graph and periodically system invokes an algorithm that searches for the cycle in the **wait-for** graph.

**2) Several Instances of a Resource Type:**

The wait-for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type. Now we will move towards a deadlock detection algorithm that is applicable for such systems. This algorithm mainly uses several time-varying data structures that are similar to those used in Banker's Algorithm and these are as follows:

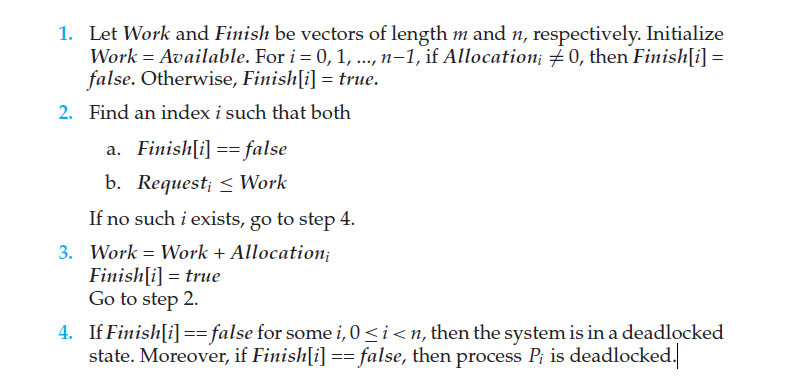
• **Available:** A vector of length m indicates the number of available resources of each type.

• **Allocation:** It is An n × m matrix defines the number of resources of each type currently allocated to each process.

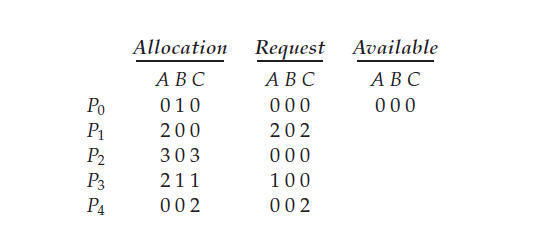
**• Request:** It is An n × m matrix indicates the current request of each process. If Request[i][j] equals k, then process Pi is requesting k more instances of

resource type Rj.

Allocation and Request as vectors; we refer to them as Allocationi and Requesti. The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.



**Example:**

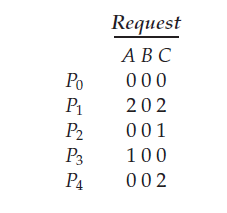


To check whether the system is in deadlock or not?

**Solution:**

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence <P0, P2, P3, P1, P4> results in Finish[i] == true for all i.

**Example:** Suppose now that process P2 makes one additional request for an instance of type C. The Request matrix is modified as follows:



**Solution:**

We claim that the system is now deadlocked. Although we can reclaim the resources held by process P0, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

**3) Detection-Algorithm Usage:**

When should we invoke the detection algorithm? The answer depends on two factors:

1. How often is a deadlock likely to occur?

2. How many processes will be affected by deadlock when it happens?