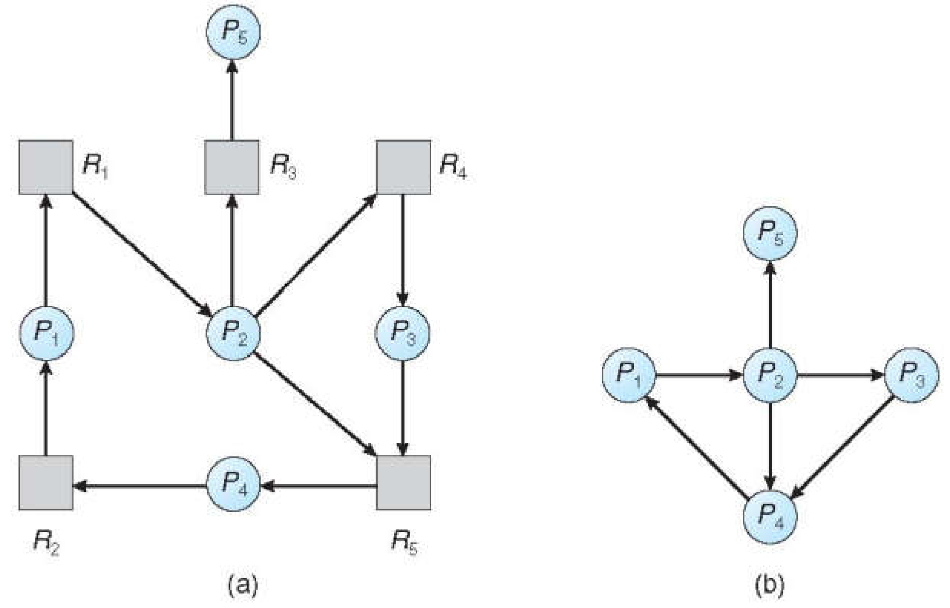
❖ Deadlock Detection

If a system does not employ either a ***deadlock-prevention*** or a ***deadlock avoidance*** algorithm, then a deadlock situation may occur. The system may provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock.

1. Single Instance of Each Resource Type

- Maintain **wait-for** graph: we obtain this graph from the resource-allocation graph by removing the resource nodes.
- Nodes are processes.
- $P_i \rightarrow P_j$ if P_i is waiting for P_j (to release a requested resource).
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.

❖ Resource-Allocation Graph and Wait-for Graph



Resource-allocation graph Corresponding wait-for graph

2. Several Instances of a Resource Type

The detection algorithm employs several time-varying data structures:

- **Available**: A vector of length ***m*** indicates the number of available resources of each type
- **Allocation**: An ***n*** **x** ***m*** matrix defines the number of resources of each type currently allocated to each process
- **Request**: An ***n*** **x** ***m*** matrix indicates the current request of each process. If ***Request*[*i*, *j*]** = ***k***, then process P_i is requesting ***k*** more instances of resource type R_j

❖ Detection Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively.
Initialize:
(a) ***Work = Available***
(b) For ***i*** = 1, 2, …, *n*, if $\boldsymbol{Allocation_i} \neq \mathbf{0}$, then
Finish*[i] = *false; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:
(a) ***Finish[i] == false***
(b) $\boldsymbol{Request_i} \leq \boldsymbol{Work}$
If no such ***i*** exists, go to step 4

3. $\boldsymbol{Work = Work + Allocation_i}$
Finish[i] = true
go to step 2

4. If ***Finish[i] == false***, for some ***i***, $1 \leq \boldsymbol{i} \leq \boldsymbol{n}$, then the system is in deadlock state. Moreover, if ***Finish[i] == false***, then $\boldsymbol{P_i}$ is deadlocked

❖ Example of Detection Algorithm

Suppose that a system has five processes $\boldsymbol{P_0}$ through $\boldsymbol{P_4}$; three resource types A(7instances), *B*(2 instances), and *C*(6 instances).
Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

The sequence < $\boldsymbol{P_0 , P_2 , P_3 , P_1 , P_4}$ > will result in ***Finish[i] == true*** for all ***i***

Suppose now that process $\boldsymbol{P_2}$ makes one additional request for an instance of type *C*. The ***Request*** matrix is modified as follows:

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

The system is now deadlocked. Although we can reclaim the resources held by process P_0, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

❖ Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, there are two options for breaking a deadlock. One is simply to abort one or more processes to break the circular wait. The other is to preempt some resources from one or more of the deadlocked processes.

1. Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

2. Resource Preemption

- **Selecting a victim** – minimize cost.
- **Rollback** – return to some safe state, restart process for that state.
- **Starvation** – same process may always be picked as victim.