DEADLOCK DETECTION

Deadlock Detection

Two Algorithms:

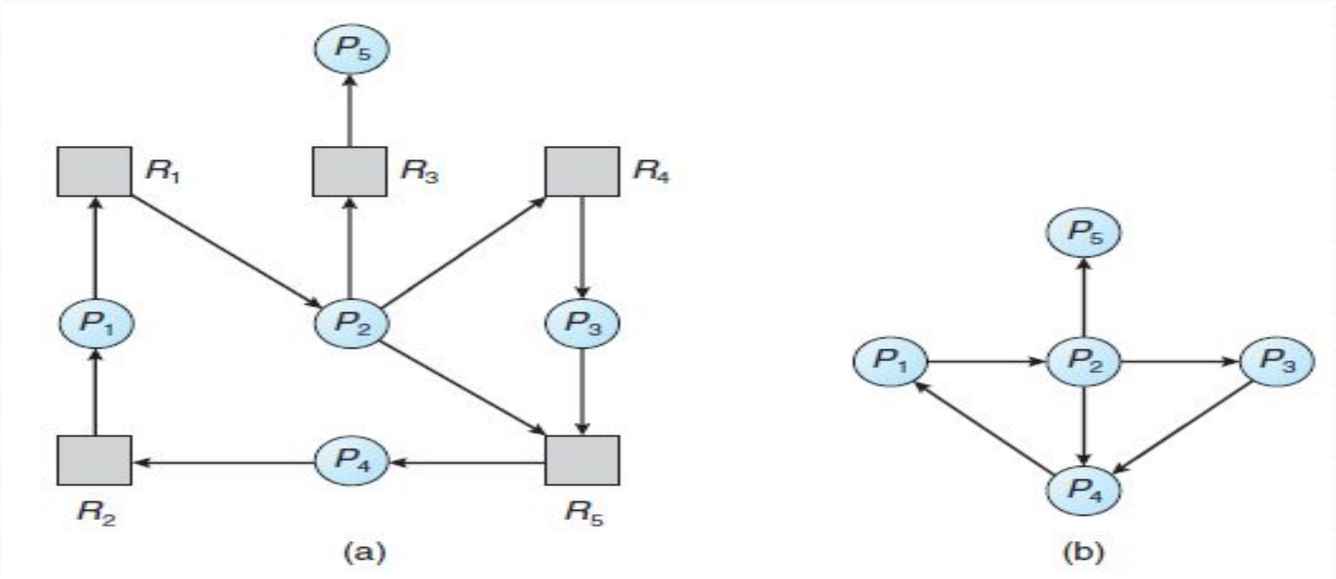
1)Algorithm for Single Instance of Each Resource Type

2) Algorithm for Several Instances of a Resource Type

Deadlock Detection-Single Instance of each Resource Type

- **Deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph.**
- **Wait for graph can be obtained from the resource-allocation graph by removing the resource nodes and collapsing the 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* .**

Deadlock Detection-Single Instances of each Resource type

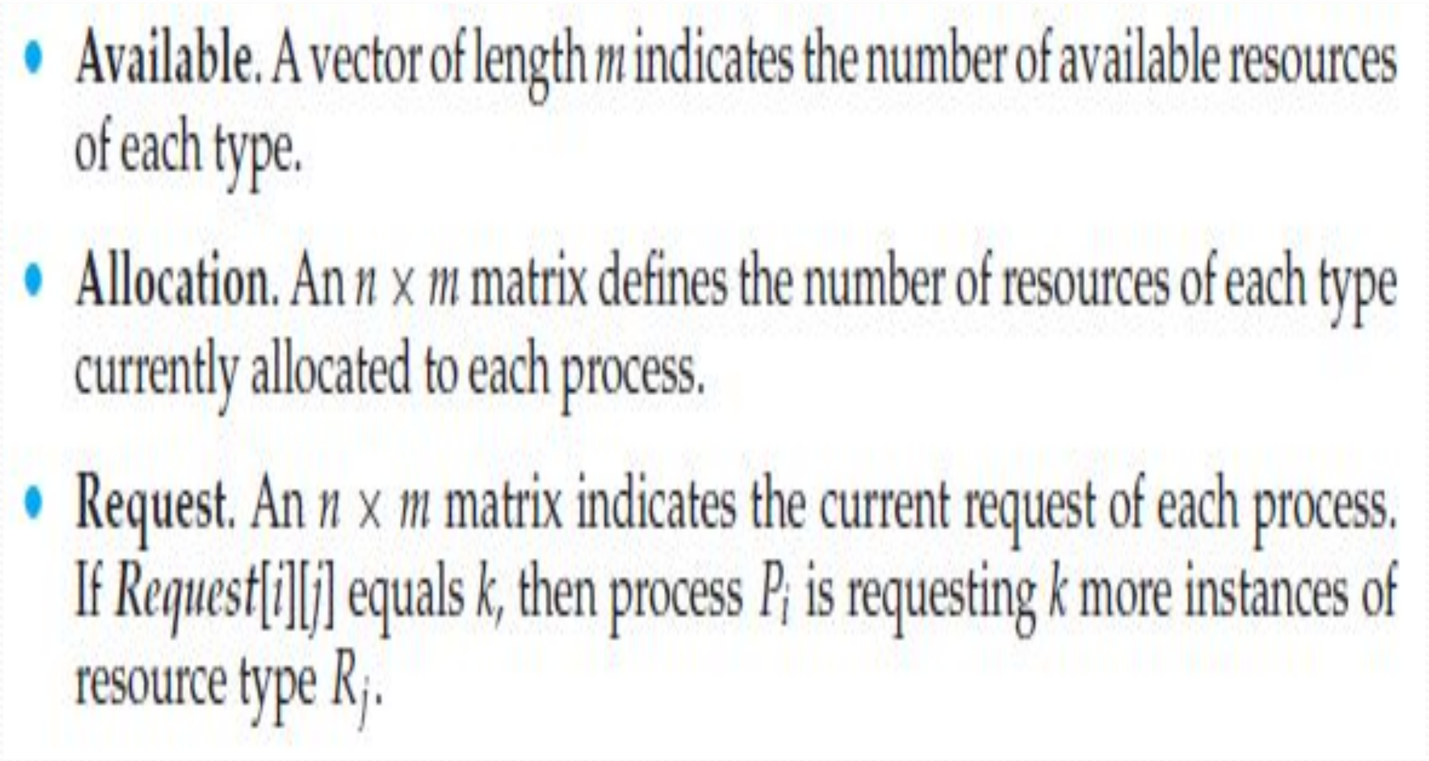


(a) Resource-allocation graph. (b) Corresponding wait-for graph.

- A deadlock exists in the system if and only if the wait-for graph contains a cycle.
- To detect deadlocks, the system needs to ***maintain the wait for*** graph and periodically ***invoke an algorithm that searches for a cycle in*** the graph.

Deadlock Detection-Several Instances of a Resource Type

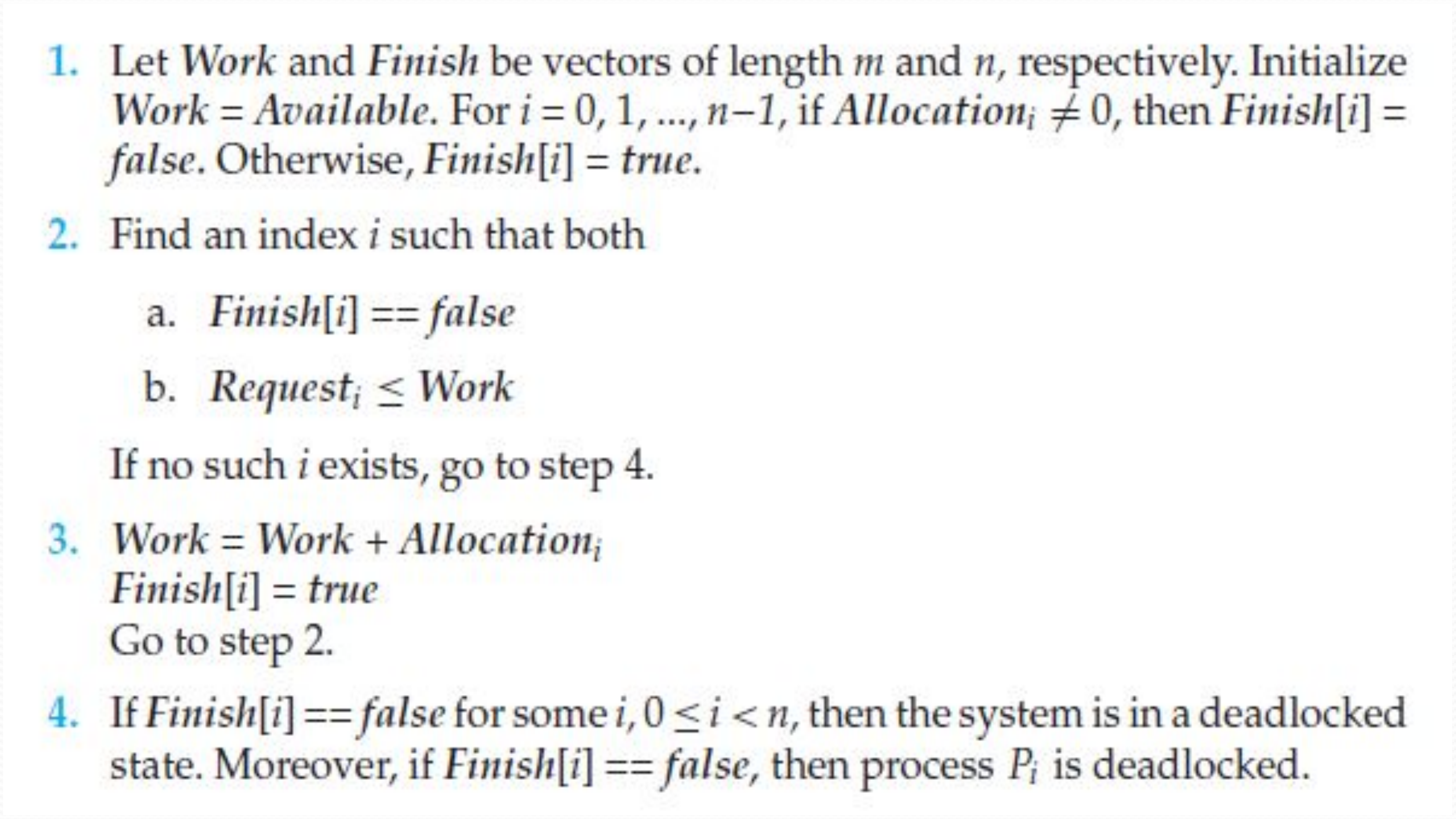
Data structures used



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

Deadlock Detection-Several Instances of a Resource Type

Algorithm

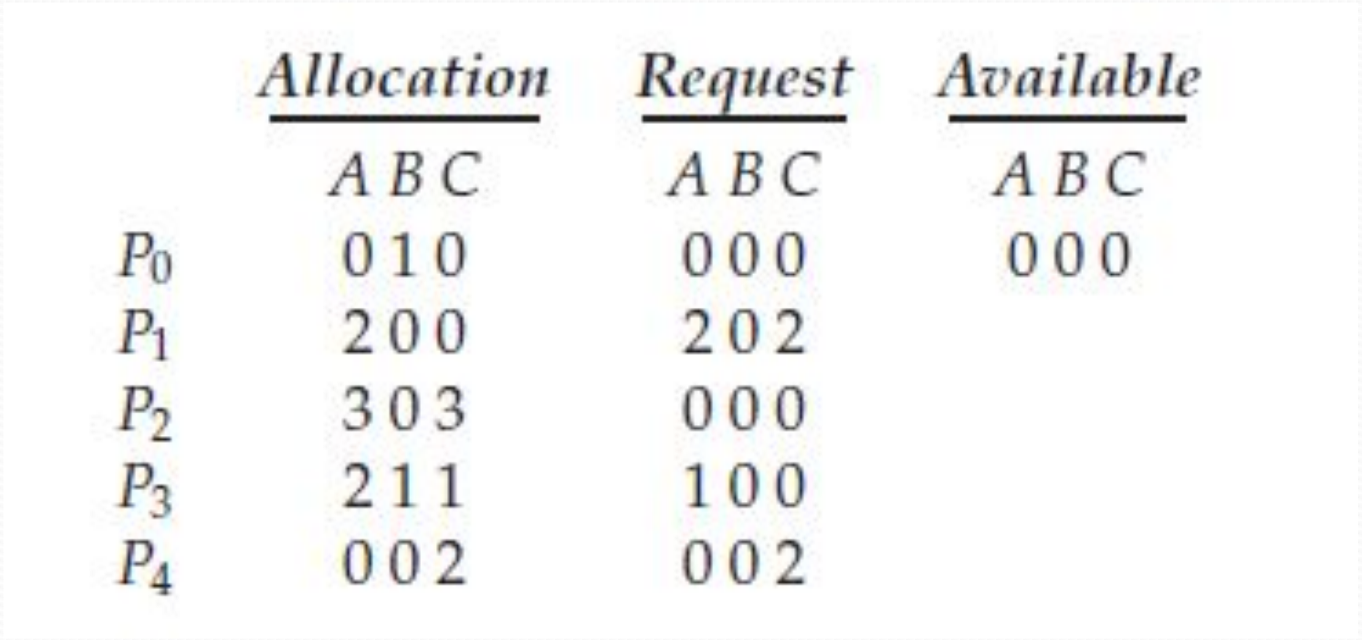


1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize $Work = Available$. For $i = 0, 1, ..., n-1$, if $Allocation_i \neq 0$, then $Finish[i] = false$. Otherwise, $Finish[i] = true$.
2. Find an index i such that both
   a. $Finish[i] == false$
   b. $Request_i \leq Work$

   If no such i exists, go to step 4.
3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   Go to step 2.
4. If $Finish[i] == false$ for some i, $0 \leq i < n$, then the system is in a deadlocked state. Moreover, if $Finish[i] == false$, then process P_i is deadlocked.

Deadlock Detection-Example

- Consider a system with five processes *P0* through *P4 and three resource types A, B, and C. Resource type A has seven* instances, resource type *B has two instances, and resource type C has six* instances. Suppose that, at time *T0, we have the following resource-allocation* state: **Detect deadlock**.



	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	

Deadlock Detection-Example

Suppose now that process *P2 makes one additional request for an instance* of type *C, will the system be in deadlock state?*

DEADLOCK RECOVERY

Recovery from Deadlock

- There are two options for breaking a deadlock.

1. Abort one or more processes to break the circular wait.
2. Preempt some resources from one or more of the deadlocked processes.

Recovery from Deadlock

Process Termination

Abort all deadlocked processes.

- The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and will have to be recomputed later.

Abort one process at a time until the deadlock cycle is eliminated

- After each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked

Recovery from Deadlock

- Processes to be aborted are selected based on following.

1. What the priority of the process is
2. How long the process has computed and how much longer the process will compute before completing its designated task
3. How many and what types of resources the process has used (for example, whether the resources are simple to preempt)
4. How many more resources the process needs in order to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

Recovery from Deadlock

Resource Preemption

- To eliminate deadlocks using resource preemption, successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

Three issues need to be addressed:

1. **Selecting a victim**

- Which resources and which processes are to be preempted? we must determine the order of preemption to minimize cost.

2. Rollback.

- After preemption of resources the process must rollback ie. abort the process and then restart it.

Recovery from Deadlock

3. **Starvation.**

- We must ensure that starvation will not occur. That means resources will not always be preempted from the same process.
- We must ensure that a process can be picked as a victim only a (small) finite number of times.