**DEADLOCK CHARACTERIZATION**

**Deadlock characterization** describes the distinctive features that are the cause of deadlock occurrence.

**1)Necessary Conditions:**

A deadlock situation can arise if the following four conditions hold simultaneously in a system:

* Mutual Exclusion
* No-preemption
* Hold & wait
* Circular wait

For a deadlock to occur, all four conditions must be met. The impasse is broken if any one of them is averted or resolved.

**1. Mutual exclusion:** At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

**2. Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

**3. No pre-emption:** Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

**4. Circular wait:** A set {P0, P1, ..., Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

**2)Resource-Allocation Graph:**

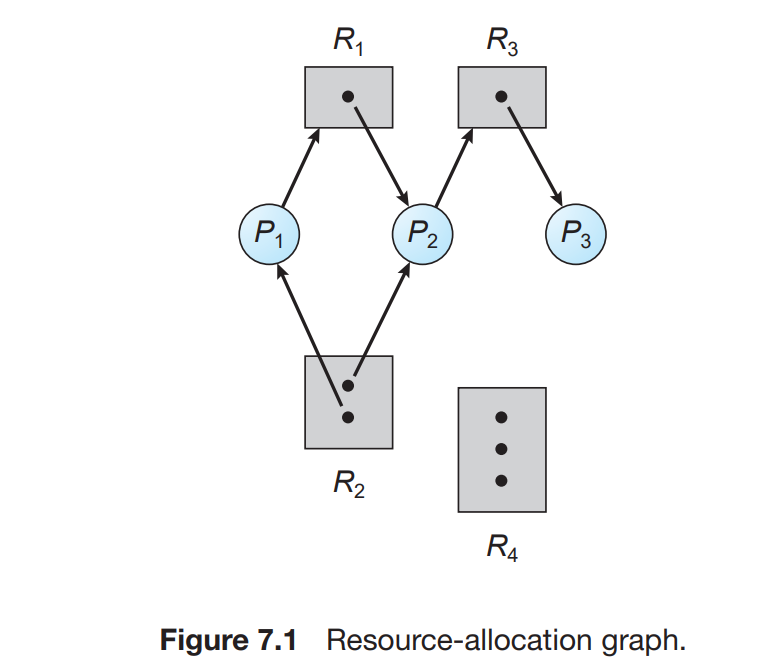
Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes: P = {P1, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {R1, R2, ..., Rm}, the set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj is denoted by Pi → Rj ; it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from resource type Rj to process Pi is denoted by Rj → Pi ; it signifies that an instance of resource type Rj has been allocated to process Pi . A directed edge Pi → Rj is called a request edge; a directed edge Rj → Pi is called an assignment edge.

Pictorially, we represent each process Pi as a circle and each resource type Rj as a rectangle. Since resource type Rj may have more than one instance, we represent each such instance as a dot within the rectangle. Note that a request edge points to only the rectangle Rj , whereas an assignment edge must also designate one of the dots in the rectangle.

When process Pi requests an instance of resource type Rj , a request edge is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource. As a result, the assignment edge is deleted.

The resource-allocation graph shown below depicts the following situation.



• The sets P, R, and E:

◦ P = {P1, P2, P3}

◦ R = {R1, R2, R3, R4}

◦ E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

• Resource instances:

◦ One instance of resource type R1

◦ Two instances of resource type R2

◦ One instance of resource type R3

◦ Three instances of resource type R4

• Process states:

◦ Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.

◦ Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.

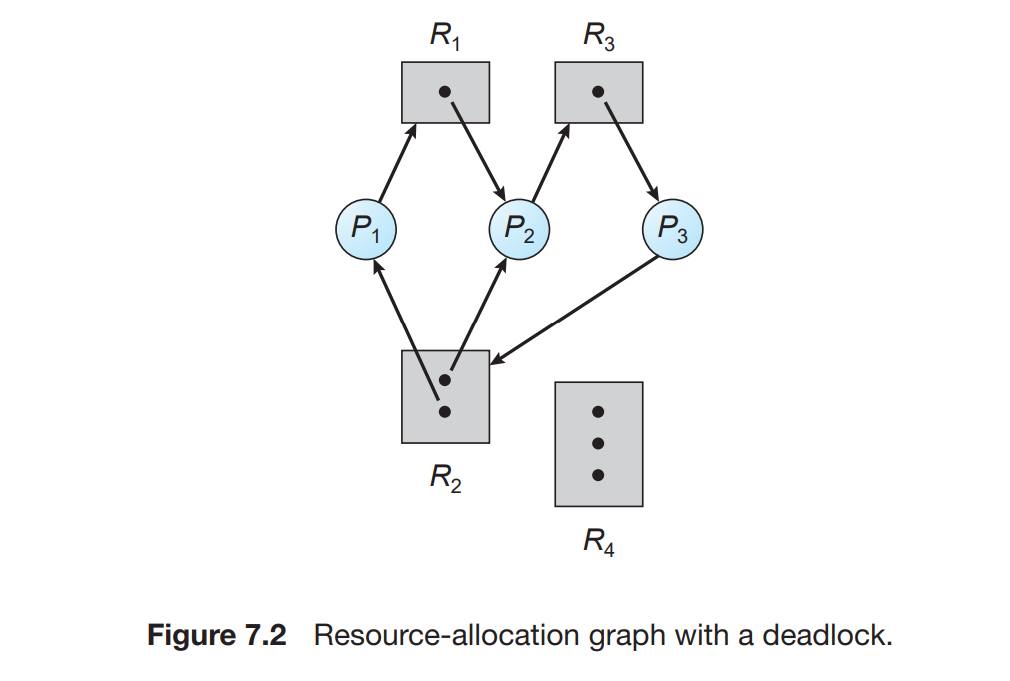
◦ Process P3 is holding an instance of R3.

Given the definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

**Example:** The resource-allocation graph depicted in below



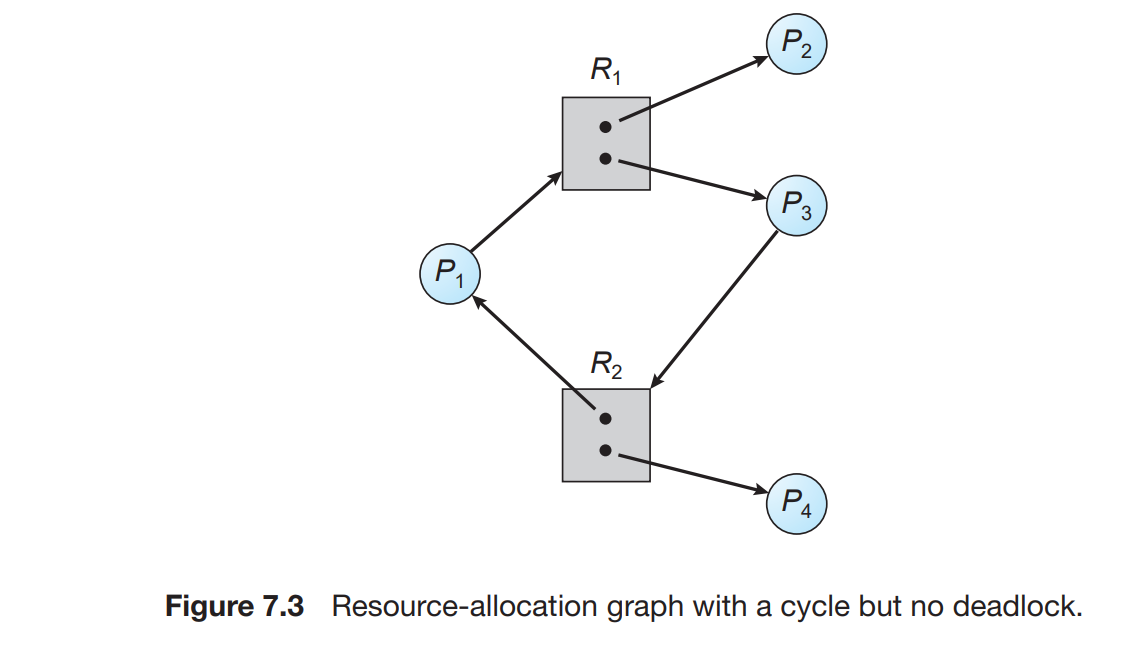
Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, we add a request edge P3 → R2 to the graph. At this point, two minimal cycles exist in the system:

P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

Processes P1, P2, and P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3. Process P3 is waiting for either process P1 or process P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.

**Example:** The resource-allocation graph is



In this example, we also have a cycle: P1 → R1 → P3 → R2 → P1

However, there is no deadlock. Observe that process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.