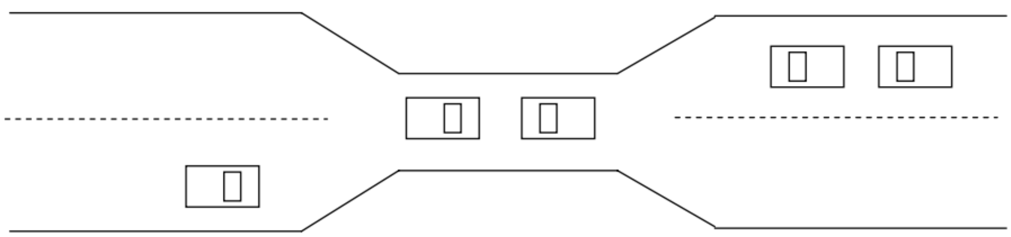
❖ Definition

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set. E.g., a system has 2 disk drives *P*1 and *P*2 each hold one disk drive and each needs another one.

Bridge Crossing Example

- Traffic only in one direction
- Each section of a bridge can be viewed as a resource
- Deadlock can be resolved if one car backs up (preempt resources and rollback)
- Several cars may have to be backed up if a deadlock occurs
- Starvation is possible



❖ System Model

- A System consists of resources. Resource types R_1 , R_2 , ... , R_m : *CPU cycles, memory space, I/O devices*
- Each resource type R_i has W_i instances
- Each process utilizes a resource as follows:

1. **Request**. The process requests the resource. If the resource is being used by another process, then the requesting process must wait until it can acquire the resource.
2. **Use**. The process can operate on the resource (for example, if the resource is a printer, the process can print).
3. **Release**. The process releases the resource.
   - **request** and **release** may be system calls. A system table records if a resource is free or allocated. For each allocated resource, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it is added to a queue of waiting processes for this resource.

❖ Deadlock Characterization

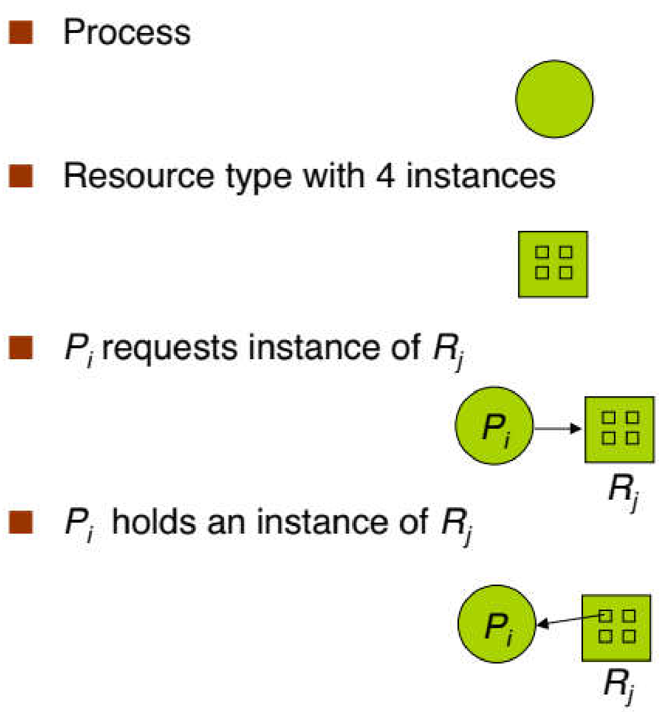
Deadlock can arise if all **four** conditions hold **simultaneously**:

1. **Mutual exclusion:** only one process at a time can use a resource; other processes requesting this resource must be delayed
2. **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

3. **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
4. **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0

❖ **Resource-Allocation Graph**

- Directed graph: a set of vertices V and a set of edges E
- V is partitioned into two sets:
  - $P = \{ P_1, P_2, \ldots, P_n \}$, set of all the processes in the system
  - $R = \{ R_1, R_2, \ldots, R_m \}$, set of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$



❖ Resource-Allocation Graph Example

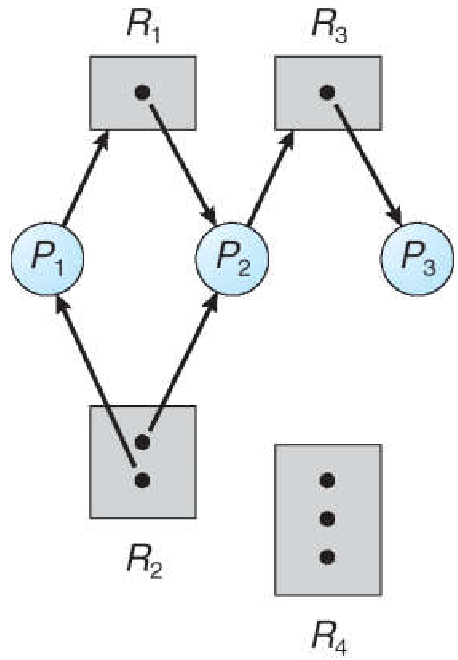
The following resource-allocation graph depicts the following:

The sets P, R, and E:

$P = \{P_1, P_2, P_3\}$

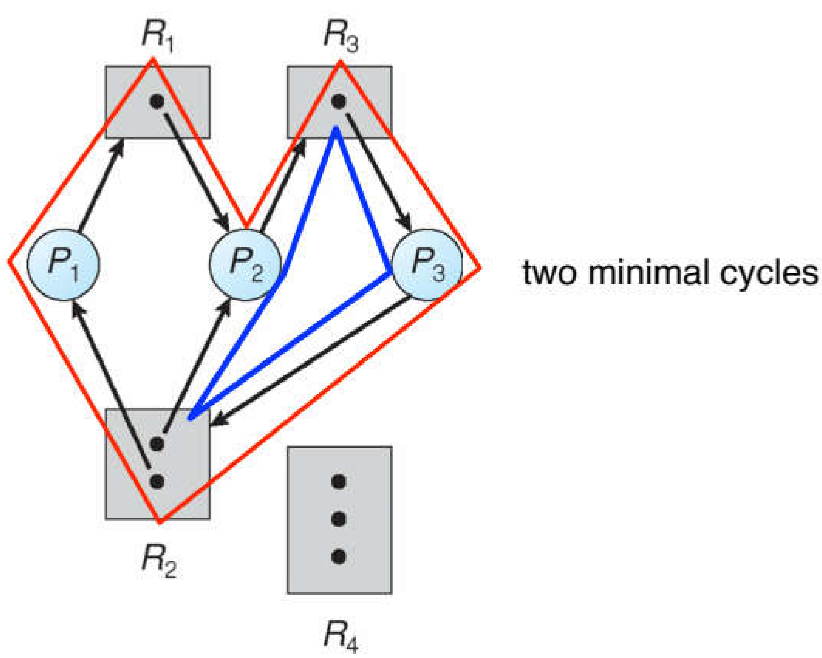
$R = \{R_1, R_2, R_3, R_4\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$



- **Process states:**
- P_1 is holding an instance of resource R_2 and is waiting for an instance of resource type R_1.
- P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance of R_3.
- P_3 is holding an instance of R_3

❖ Resource Allocation Graph with a Deadlock



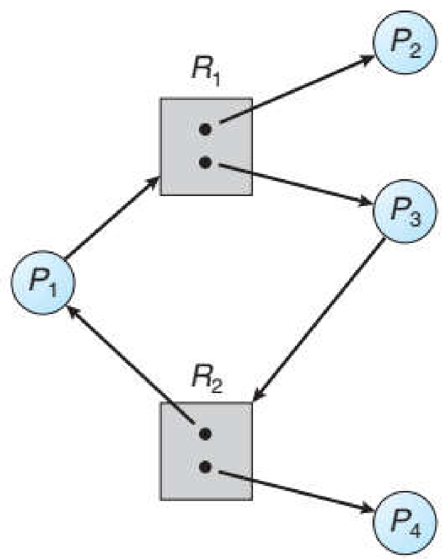
Suppose that process P_3 requests an instance of resource type R_2. Since no resource instance is currently available, we add a request edge $P_3 \rightarrow R_2$ to the graph. Two minimal cycles exist in the system:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

Processes P_1, P_2, and P_3 are ***deadlocked***.

❖ Resource-Allocation Graph with a Cycle but no Deadlock

In this example, we also have a cycle: $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



No deadlock. P_4 may release its instance of R_2. R_2 can be allocated to P_3, breaking the cycle.

In summary:

- If graph contains no cycles ⇒no deadlocks
- If graph contains a cycle ⇒a deadlock *may* or *may not* exist
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

❖ Minimum Resources Required

In general, minimum number of resources required so that system must be in safe state:

$\{(R_1-1) + (R_2-1) + (R_3-1) + \ldots + (R_n-1)\}+1 = (\Sigma R_i - n) +1$; where i is from 1 to n

Example: Suppose there are three processes P_1, P_2 and P_3. P_1 requires 3 resources, P_2 requires 5 resources and P_3 requires 7 resources. What are the minimum number of resources required so the system will never enter in deadlock?

The minimum resources required = 2 + 4 + 6 + 1 = 13