**Deadlock Handling**

**Methods for Deadlock Handling**

1. Deadlock Prevention.
2. Deadlock Avoidance.
3. Deadlock Detection.

**Deadlock-Prevention**

Do not allow one of the four conditions to occur:

1. Mutual exclusion: M.E. is not required for sharable resources; must hold for non sharable resources. Prevention is not possible for printers and other non-sharable. Shared entities (read only files) don't need mutual exclusion (and aren’t susceptible to deadlock.)
2. Hold and wait: Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none. Utilization is low, starvation possible.
3. Nopreemption**:** If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released, i.e. release any resource already being held if the process can't get an additional resource. Preempted resources are added to the list of resources for which the process is waiting. Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
4. Circular wait:Impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration, i.e. number resources and only request in ascending order.

**Problems associated with Deadlock Prevention:**

Deadlock prevention, it is an attempts to prevent at least one of the four necessary conditions. This may lead to inefficient use of resources as well as inefficient execution of processes. Each of these prevention techniques may cause a decrease in utilization and/or resources, for this reason, prevention technique; it isn't necessarily the best technique. Prevention is generally the easiest to implement.

**Deadlock Avoidance:**

* The simplest and most useful **deadlock avoidance** algorithm requires that each process declare the *maximum number* of resources of each type that it may need.
* The **deadlock-avoidance** algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition (unsafe state).
* **Resource-allocation *state*** is defined by the number of available and allocated resources, and the maximum demands of the processes.
* Given the complete sequence of requests and releases for each process, we can decide for each request whether or not the process should wait.
* For every request, the system considers the resources currently available, the resources currently allocated, and the future requests and releases of each process, and decides whether the current request can be satisfied or must wait to avoid a possible future deadlock.

**Deadlock Avoidance: (Safe state)**

* When a process requests an available resource, the system must decide if immediate allocation leaves the system in a *safe state* or not.
* The system is in ***safe state*** if there exists a safe sequence of all processes. The sequence <*P*1, *P*2, …, *Pn*> is safe if for each *P*i, the resources that *P*i can still request can be satisfied by currently available resources + resources held by all the ***Pj*,** with *j<i*.
* If Pi resource needs are not immediately available, then *Pi* can wait until all *Pj* have finished.
* When *Pj* is finished, *P*i can obtain needed resources, execute, return allocated resources, and terminate.
* When *Pi* terminates, *Pi*+1 can obtain its needed resources, and so on.

**Algorithms for Deadlock Avoidance**

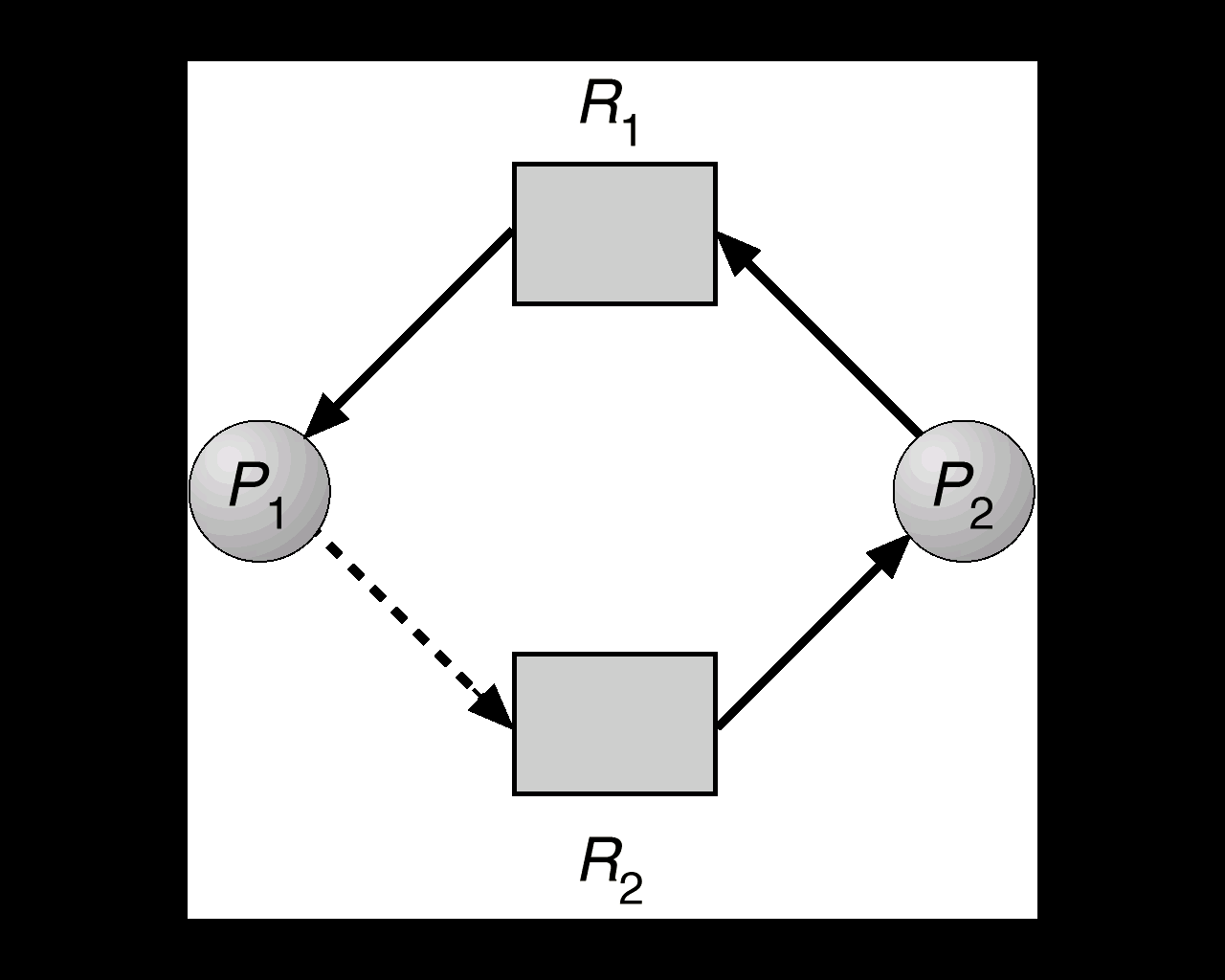
There are two algorithms:

* 1. **Resource allocation graph algorithm**: (Every resource type has a single instance or (1-3)).
  2. **Banker’s algorithm**: (Every type has any number of instances).

**Resource-allocation graph algorithm:**

* *Claim edge* *Pi* → *Rj* indicated that process *Pi* may request resource *Rj*; represented by a dashed line.
* Claim edge converts to request edge *Pi* → *Rj* when a process requests a resource.
* When a resource is released by a process, assignment edge reconverts to a claim edge.

**Resource Allocation Graph Examples**

**  
Figure (1): Unsafe state in resource-allocation graph (RAG)**

**Figure (2): Other examples of** **resource-allocation graph (RAG)**

* Request edge – directed edge *T*1 → *Rj*
* Assignment edge – directed edge *Rj* → *Ti*