Deadlocks detection and recovery
If a system does not employ either a deadlock-prevention or a deadlock-avoidance algorithm, then a deadlock situation may occur. In such environment, two algorithms are used:

- **Deadlock-detection algorithm**: Examines the state of the system to determine whether a deadlock has occurred or not.
  - The frequency of invoking this algorithm depends on the likelihood of a deadlock and how many processes will be affected by a deadlock.
  - **Static rules**: when the CPU utilization drops below 40%, or execute algorithm each 1h.
  - **Maybe after each resource request** (Overhead in computational time).

- **Deadlock-recovery algorithm**: To drive back the system into a safe state.
Deadlocks Detection

If a system contains resources that have only one instance, then we can use an algorithm that exploits a variant of the RAG, k.a., wait-for-graph.

Definition

**Wait-for-graph.** Is a directed graph in which the nodes represent processes and the edges represent a process waiting for another process.

- The meaning of an edge $[P_i \rightarrow P_j]$: Process $P_i$ is waiting for a resource that is held by process $P_j$.
- The WFG is obtained from the RAG by removing the resource nodes and collapsing the appropriate edges.

Let $\mathcal{R}$ by a resource-allocation graph and let $\mathcal{W}$ be its corresponding wait-for-graph then:

$$\forall (P_i, P_j, R_r) : (P_i \rightarrow R_r) \in \mathcal{R} \land (R_r \rightarrow P_j) \in \mathcal{R} \implies (P_i \rightarrow P_j) \in \mathcal{W}$$
Deadlocks Detection

If a system contains resources that have only one instance, then we can use an algorithm that exploits a variant of the RAG, k.a., **wait-for-graph**.
Deadlocks Detection

If a system contains resources that have only one instance, then we can use an algorithm that exploits a variant of the RAG, k.a., \textit{wait-for-graph}.

If $\exists$ cycle in $\mathcal{W}$ $\iff$ deadlock
Deadlocks Detection

If a system contains resources that have multiple instances, then the system can apply a modified version of the Banker’s algorithm to detect deadlocks.

Deadlock Detection Algorithm.

1. Let Work and Finish be vectors of length m and n respectively such that Work = Available and ∀i, if Allocation; ≠ 0: Finish[i] = false. Else, Finish[i] = true.

2. Find an index i:
   - Finish[i] == false
   - Request; ≤ Work
     Else goto 4.

3. Work = Work + Allocation;i;
   Finish[i] = true;
   goto 2;

4. If ∃i: Finish[i] == false then the system is deadlocked.
Deadlocks Detection Algorithm

Allocation

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Request

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Deadlocks Detection Algorithm

Available
R1  R2  R3
0    0    0

Allocation
R1  R2  R3
P0  0  1  0
P1  2  0  0
P2  3  0  3
P3  2  1  1
P4  0  0  2

Request
R1  R2  R3
P0  0  0  0
P1  2  0  2
P2  0  0  0
P3  1  0  0
P4  0  0  2

Process P0
Process P1
Process P2
Process P3
Process P4
## Deadlocks Detection Algorithm

### Available

- Resource 1: 0
- Resource 2: 0
- Resource 3: 0

### Allocation

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Request

<table>
<thead>
<tr>
<th>Process</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### We claim that the system is not deadlocked
Deadlocks Detection Algorithm

We claim that the system is not deadlocked
A safe sequence exists <P0, P2, P3, P1, P4>
## Deadlocks Detection Algorithm

### Available

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Allocation

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Request

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Suppose that Process P2 requests one additional instance of resource type 3.
Deadlocks Detection Algorithm

Available
R1  R2  R3
0    0    0

Allocation
R1  R2  R3
P0  0  1  0
P1  2  0  0
P2  3  0  3
P3  2  1  1
P4  0  0  2

Request
R1  R2  R3
P0  0  0  0
P1  2  0  2
P2  0  0  1
P3  1  0  0
P4  0  0  2

Process P0  Process P1  Process P2  Process P3  Process P4
Deadlocks Detection Algorithm

Resource allocation and request matrices:

**Allocation**

- Resource 1: P0: 0, P1: 2, P2: 3, P3: 2, P4: 0
- Resource 2: P0: 1, P1: 0, P2: 3, P3: 1, P4: 0
- Resource 3: P0: 0, P1: 0, P2: 0, P3: 0, P4: 0

**Request**

- Resource 1: P0: 0, P1: 2, P2: 0, P3: 1, P4: 0
- Resource 2: P0: 0, P1: 0, P2: 0, P3: 0, P4: 0
- Resource 3: P0: 0, P1: 2, P2: 0, P3: 1, P4: 0

**Available**

- Resource 1: 0
- Resource 2: 0
- Resource 3: 0

**Deadlock**

(Not enough resources)

Even if we reclaim resource from P0

- Process P0
- Process P1
- Process P2
- Process P3
- Process P4
Deadlocks Recovery Algorithm

When a system determines that a deadlock is taking place. A deadlock recovery routine is executed:

- **Process termination.** Abort processes involved in a deadlock, all at a time or one by one till deadlock is gone.
  - Which process to terminate (priority, number of resources, how much left to terminate, dependency, ...)
  
  Processes which termination will not incur an expensive cost should be aborted

- **Resource preemption.** Taking away resources from certain process and allocating them to others.
  - Victim, selecting a resource to be preempted + cost should be minimum.
  - Rollback, hard to determine a safe state, processes are aborted (total rollback).
  - Starvation, a process may not get a given resource again.