Banker’s Algorithm (Deadlock Avoidance Scheme):

Data Structures:
- **Available**:
  - Vector of length \( m \)
  - # instances of each resource type available in system
  - If available \([j] = k\), there are \( k \) instances of resource type \( R_j \) available

- **Max**:
  - \( n \times m \) matrix.
  - Maximum # of instances of each resource each process can request
  - If \( Max[i,j] = k \), then process \( P_i \) may request at most \( k \) instances of resource type \( R_j \)

- **Allocation**:
  - \( n \times m \) matrix
  - # instances of each resource type allocated to each process
  - If \( Allocation[i,j] = k \) then \( P_i \) is currently allocated \( k \) instances of \( R_j \).

- **Need**:
  - \( n \times m \) matrix
  - # instances of each resource type each process may need more of
  - If \( Need[i,j] = k \), then \( P_i \) may need \( k \) more instances of \( R_j \) to complete its task

Safety Algorithm:
1. Let \( Work \) and \( Finish \) be vectors of length \( m \) and \( n \), respectively. Initialize:
   \[
   Work = Available \\
   Finish[i] = false \text{ for } i = 0, 1, 2, \ldots, n-1.
   \]
2. Find an \( i \) such that both:
   (a) \( Finish[i] = false \)
   (b) \( Need[i] \leq Work \)
   If no such \( i \) exists, go to step 4.
3. \( Work = Work + Allocation[i] \)
   \( Finish[i] = true \)
   go to step 2.
4. If \( Finish[i] = true \) for all \( i \), then the system is in a safe state.

Resource-Request Algorithm for Process \( P_i \):
- \( Request_i \) = request vector for process \( P_i \).
If \( Request[i] = k \) then process \( P_i \) wants \( k \) instances of resource type \( R_i \).
When resource request made by \( P_i \), the following occurs:
1. If \( Request_i \leq Need[i] \) go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If \( Request_i \leq Available \), go to step 3. Otherwise \( P_i \) must wait, since resources are not available.
3. Pretend to allocate requested resources to \( P_i \) by modifying the state as follows:
   \[
   Available = Available - Request_i; \\
   Allocation_i = Allocation_i + Request_i; \\
   Need[i] = Need[i] - Request_i;
   \]
   - If safe \( \Rightarrow \) the resources are allocated to \( P_i \)
   - If unsafe \( \Rightarrow P_i \) must wait, and the old resource-allocation state is restored
Example of Banker's Algorithm:

- 5 processes $P_1$ through $P_5$
- 3 resource types:
  - A: 10 instances
  - B: 5 instances
  - C: 7 instances

Resource-allocation state at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0 1 0</td>
<td>7 5 3</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2 0 0</td>
<td>3 2 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3 0 2</td>
<td>9 0 2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2 1 1</td>
<td>2 2 2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 0 2</td>
<td>4 3 3</td>
</tr>
</tbody>
</table>

The content of the matrix, Need is defined to be Max – Allocation.

<table>
<thead>
<tr>
<th>Need</th>
<th>A B C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>7 4 3</td>
</tr>
<tr>
<td>$P_1$</td>
<td>1 2 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>6 0 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0 1 1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4 3 1</td>
</tr>
</tbody>
</table>

The system is in a safe state since the sequence $<P_1, P_3, P_0, P_2, P_4>$ satisfies safety criteria.

Example (cont.):

Now $P_1$ requests 1 instance of A and 2 instances of C:

Request $1 (1,0,2)$

1.) Check that Request $\leq $ Available

$(1,0,2) \leq (3,3,2) \Rightarrow true$ (can immediately grant request)

2.) Now see if system is in safe state:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Need</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0 1 0</td>
<td>7 4 3</td>
</tr>
<tr>
<td>$P_1$</td>
<td>3 0 2</td>
<td>0 2 0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3 0 1</td>
<td>6 0 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 0 2</td>
<td>4 3 1</td>
</tr>
</tbody>
</table>

Executing safety algorithm shows that sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies safety requirement.

Can a request for (3,3,0) by $P_4$ be granted from this safe state?

Can a request for (0,2,0) by $P_0$ be granted from this safe state?
Deadlock Detection Algorithm:

Data Structures:
- **Available**:
  - vector of length $m$
  - # of available resources of each type

- **Allocation**:
  - $n \times m$ matrix
  - # of resources of each type currently allocated to each process

- **Request**:
  - $n \times m$ matrix
  - Current request of each process
  - If $Request[i][j] = k$, then process $P_i$ is requesting $k$ more instances of resource type $R_j$

Detection Algorithm
1. Let $Work$ and $Finish$ be vectors of length $m$ and $n$, respectively Initialize:
   (a) $Work = Available$
   (b) For $i = 0, 1, 2, \ldots, n-1$, 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 $i$, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then $P_i$ is deadlocked.
Example of Detection Algorithm

- Five processes $P_0$ through $P_4$
- Three resource types:
  - A: 7 instances
  - B: 2 instances
  - C: 6 instances

- Resource Allocation state at time $T_0$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2 0 0</td>
<td>2 0 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3 0 3</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 0 2</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>

- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in $Finish[i] = true$ for all $i$.

Example (cont.)

- $P_2$ now requests an additional instance of type $C$:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>A B C</td>
<td>A B C</td>
</tr>
<tr>
<td>$P_0$</td>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2 0 0</td>
<td>2 0 2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3 0 3</td>
<td>0 0 1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0 0 2</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>

- What is the state of the system?
  - Can reclaim resources held by process $P_0$, but insufficient resources to fulfill other processes’ requests. System is deadlocked.
  - Deadlock consists of processes $P_1$, $P_2$, $P_3$, and $P_4$. 