Necessary Conditions

- **Mutual exclusion** - at least one of the resources must require exclusive access.

- **Hold and wait** (also called *wait-for*) - each process must request (and possibly wait for) a resource while holding another resource.

- **Non-preemption** - resources cannot be preempted cheaply

- **Circular wait** - a cycle of hold and wait must exist.
Safe and Unsafe States

Diagram showing the relationship between deadlock states, unsafe states, and safe states.
Banker's Algorithm

• Recall: Global data structures for n processes and m resource types:

```c
int Available[m] = { # of each type in system };
int Max[n][m]; // max possible demand of each P
    // Max[i][j] = k means Pi might
    // request up to k Rj's
int Allocation[n][m]; // current alloc to each P
    // Allocation[i][j] = k
    // means k Rj's alloc to Pi
int Need[n][m]; // remaining needs for each P
    // Need[i][j] =
    //    Max[i][j] - Allocation[i][j]
```
Banker's Algorithm

- Recall: Local data structures for the safety algorithm

```c
int Work[m] = Available;
    // Work replaces Available in simulation
bool Finish[m] = { false, false, ...}
    // Finish[i] set to true if Pi finishes in
    // the simulation
```
Detection Example

- \( P = \{ P_0, \ldots, P_4 \} \)  \( R = \{ A(7), B(2), C(6) \} \)

<table>
<thead>
<tr>
<th>Alloc[n][m]</th>
<th>Requests[n][m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  B  C</td>
<td>A  B  C</td>
</tr>
<tr>
<td>( P_0 )</td>
<td>0  1  0</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>2  0  0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>3  0  3</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>2  1  1</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>0  0  2</td>
</tr>
</tbody>
</table>

- **Available** = \{ 0, 0, 0 \}