Deadlock Avoidance

- Unlike **deadlock prevention**, which limits resource requests, **deadlock avoidance** makes runtime decisions to ensure a safe state.
- Avoidance allows more concurrency than prevention.

Strategies

1. **Process Initiation Denial**: A process is started only if its resource needs can be safely met.
2. **Resource Allocation Denial**: A process's request is granted only if it maintains a **safe state**.

Key Matrices in Deadlock Avoidance

- **Resource (R):** Total resources in the system.
- **Available (V):** Resources not yet allocated.
- **Claim (C):** Maximum resource demand of each process.
- **Allocation (A):** Currently allocated resources.

Important Conditions

- $R_j = V_j + \sum_{i=1}^{n} A_{ij}$
- $C_{ij} \leq R_j$ (Max claim ≤ total resources)
- $A_{ij} \leq C_{ij}$ (Allocated resources ≤ claimed resources)

Banker's Algorithm

Safety Algorithm

1. **Initialize:** `Work = Available`, `Finish[i] = false` for all `i`.
2. Find an `i` such that:
   - `Finish[i] == false`
   - `Need[i] <= Work`
3. **Update:**

    - `Work = Work + Allocation[i]`
    - `Finish[i] = true`
    - Repeat step 2 until all processes finish.
4. If all `Finish[i] == true`, system is **safe**.

Resource Request Algorithm

1. **Check Need:** If `Request[i] > Need[i]`, **error**.
2. **Check Availability:** If `Request[i] > Available`, process **waits**.
3. **Pretend Allocation:**
    - `Available = Available - Request[i]`
    - `Allocation[i] = Allocation[i] + Request[i]`
    - `Need[i] = Need[i] - Request[i]`
    - Run **Safety Algorithm** to verify if system remains safe.

Expected Input & Output

- **Input:** No. of processes, max resource need, allocated resources.
- **Output:** System **Safe** or **Unsafe**.