Principles of Deadlock
A **deadlock** is a state in which a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process in the same set.

1. Reusable Resources
Reusable resources are resources that can be used by one process at a time and must be released after use.
Examples: CPU, memory, printers, database locks.

2. Consumable Resources
Consumable resources are resources that can be created and destroyed. Once consumed, they cannot be reused.
Examples: Messages, signals, or events in an interprocess communication system.

3. Resource Allocation Graphs (RAGs)
A Resource Allocation Graph (RAG) is a directed graph that represents the allocation of resources to processes.
- **Processes (P1, P2, ...)** are represented as circles.
- **Resources (R1, R2, ...)** are represented as squares.
- **Edges (→ and ←)** indicate request and allocation.

Example:
P1 → R1 → P2 → R2 → P1 (Deadlock occurs because each process waits for a resource held by another.)

4. Conditions for Deadlock
A deadlock occurs if these four conditions hold simultaneously:
1. **Mutual Exclusion** – A resource can only be held by one process at a time.
2. **Hold and Wait** – A process holding a resource can request additional resources.
3. **No Preemption** – A resource cannot be forcibly taken from a process; it must be released voluntarily.
4. **Circular Wait** – A closed chain of processes exists, where each process is waiting for a resource held by the next process in the chain.

6.2 Deadlock Prevention
Deadlock prevention aims to ensure that at least one of the four conditions never holds.
1. Mutual Exclusion

- If possible, make resources sharable.
- Example: Read-only files can be shared among multiple processes.

2. Hold and Wait

- Ensure processes request all required resources at the start.
- Example: A process requests all required memory and files before execution.

3. No Preemption

- Allow preemption of resources if necessary.
- Example: If a process holding a resource waits for another, forcibly take its resources and restart it later.

4. Circular Wait

- Impose an ordering of resource requests.
- Example: Always request resources in increasing order (e.g., request printer first, then scanner).

6.3 Deadlock Avoidance

Deadlock avoidance ensures that the system never enters an unsafe state where deadlocks could occur.

1. Process Initiation Denial (Banker's Algorithm)

- A process is only started if its maximum resource needs can be satisfied without causing a deadlock.
- Example: If a system has 10 memory units and Process A needs 6 while Process B needs 5, denying Process B’s request avoids deadlock.

2. Resource Allocation Denial (Safe State Check)

- A process is only allocated a resource if it leaves the system in a safe state.
- Example: The system checks if granting a request leads to a state where all processes can eventually complete.

6.4 Deadlock Detection

1. Deadlock Detection Algorithm

- The system periodically checks for deadlocks using a detection algorithm (similar to the Banker's Algorithm).
- Example: A graph-based algorithm detects cycles in the Resource Allocation Graph.

2. Recovery from Deadlock

Once a deadlock is detected, recovery strategies include:

1. **Process Termination** – Kill one or more processes.
    - Example: The lowest-priority process is terminated to break the deadlock.
2. **Resource Preemption** – Take resources from one process and

allocate them to another.
   - Example: Forcefully freeing up a locked file and reallocating it.

Safe State vs. Unsafe State in Deadlock Avoidance

1. Safe State

A system is in a **safe state** if it can allocate resources to processes in a way that ensures all processes can eventually complete **without leading to a deadlock**.

🔷 **Key Idea:** There exists a **safe sequence** of process execution where all requested resources can be allocated in some order, allowing all processes to finish execution.

Example of a Safe State:

- Assume we have **10 memory units** available.
- Three processes **P1, P2, and P3** exist with the following resource needs:

Process	Maximum Need	Allocated	Remaining Need
P1	7	2	5
P2	4	1	3
P3	5	3	2

Total **allocated** = 2 + 1 + 3 = 6
Total **available** = 10 - 6 = 4
If P2 requests 3 more units, we check:

- If we grant 3 to P2, it finishes execution and releases all resources.
- Then P1 and P3 can also finish.

Since a sequence exists where all processes finish, the system is **safe**.

2. Unsafe State

A system is in an **unsafe state** if it **cannot** guarantee that all processes will complete execution, potentially leading to deadlock.

🔷 **Key Idea:** There is no **safe sequence** ensuring that all processes can get their required resources and finish.

Example of an Unsafe State:

- Suppose we have **10 memory units** available.
- Three processes **P1, P2, and P3** with the following allocations:

Process	Maximum Need	Allocated	Remaining Need
P1	7	2	5

P2	4	2	2
P3	6	3	3

Total **allocated** = 2 + 2 + 3 = 7

Total **available** = 10 - 7 = 3

Now, if P3 requests 3 more units, the system **cannot** grant it immediately, and no process can complete execution and release resources. The system is in an **unsafe state**, which could lead to a **deadlock**.

Differences Between Safe and Unsafe States

Feature	Safe State	Unsafe State
Guarantee of Completion	Yes	No
Potential for Deadlock	No	Yes
Safe Sequence Exists?	Yes	No

⚠️ **Important:**

- **Unsafe ≠ Deadlock**, but an unsafe state **can** lead to a deadlock if not managed properly.
- The **Banker's Algorithm** is used to check for **safe states** before granting resources.