Operating Systems
Course Code: **71203002004**

Deadlock Handling Techniques

by -
Minal Rajwar

Deadlock Prevention

Deadlock prevention is a method to avoid processes from getting stuck waiting for each other, like cars avoiding traffic jams at intersections.

Deadlock occurs only if these four conditions hold together:

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

To prevent deadlock, at least one of these conditions must be eliminated.

1. Eliminate Mutual Exclusion

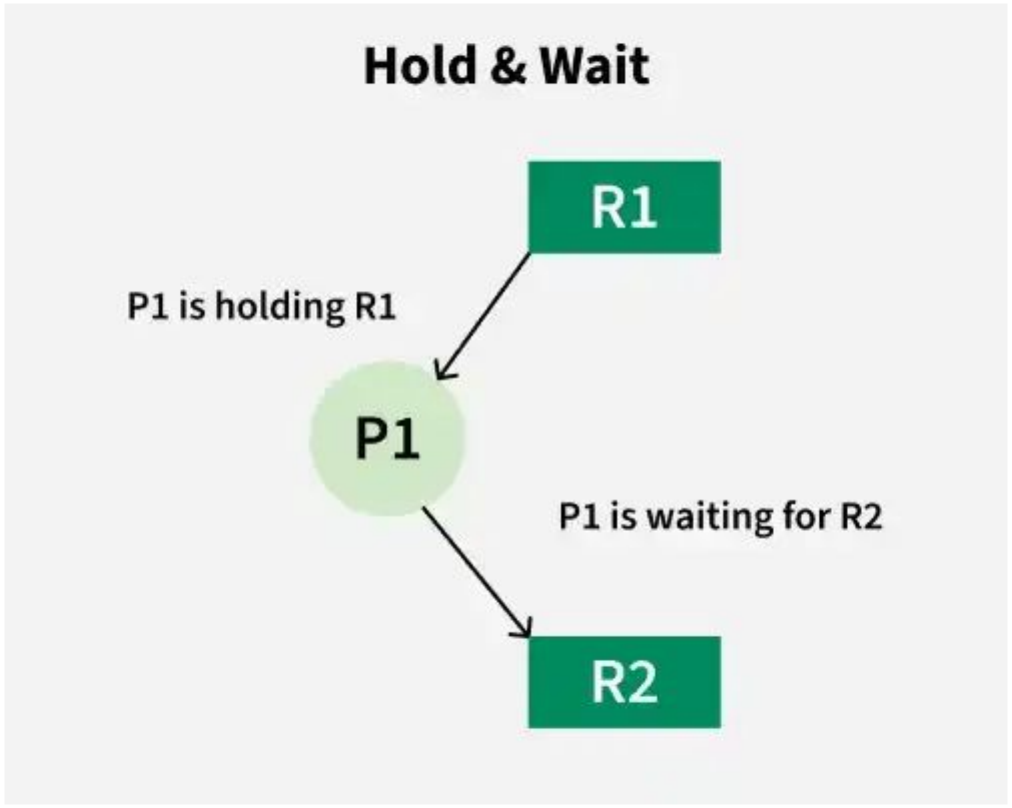
- Some resources (like printers) can’t be shared, so this condition usually can’t be removed.

- Sharable resources (like read-only files) can be used by many processes at once.

2. Eliminate Hold and Wait

A process should not hold one resource while waiting for another.
Ways to prevent it:

- **Eliminate wait:** Process requests all needed resources in advance.
- **Eliminate hold:** Process releases held resources before requesting new ones.



3. Eliminate No Preemption

Normally, resources can’t be taken back once allocated.

Prevention:

- A process releases resources after use.
- If resources are unavailable, the process releases what it holds and waits until all are free.

4. Eliminate Circular Wait

Circular wait occurs when processes form a chain, each waiting for another.

Prevention:

- Give each resource a unique number.
- Processes request resources only in increasing order.
- This avoids cycles.

Feasibility of Deadlock Prevention

- **Mutual Exclusion:** Can’t be fully removed; some resources are non-shareable.
- **Hold and Wait:** Hard to eliminate; releasing all resources while requesting new ones is inefficient.
- **No Preemption:** Risky; preempting can cause inconsistencies and inefficiency.
- **Circular Wait:** Eliminating it is the most practical way to prevent deadlock.

Deadlock Avoidance

The OS prevents deadlocks by tracking the maximum resources each process may need.

Before granting a request, the OS checks if the system will remain **safe**.

- **Safe state:** Resources can be allocated without causing deadlock.
- **Unsafe state:** Granting resources may lead to deadlock.

Essentially, the OS avoids entering a circular wait.

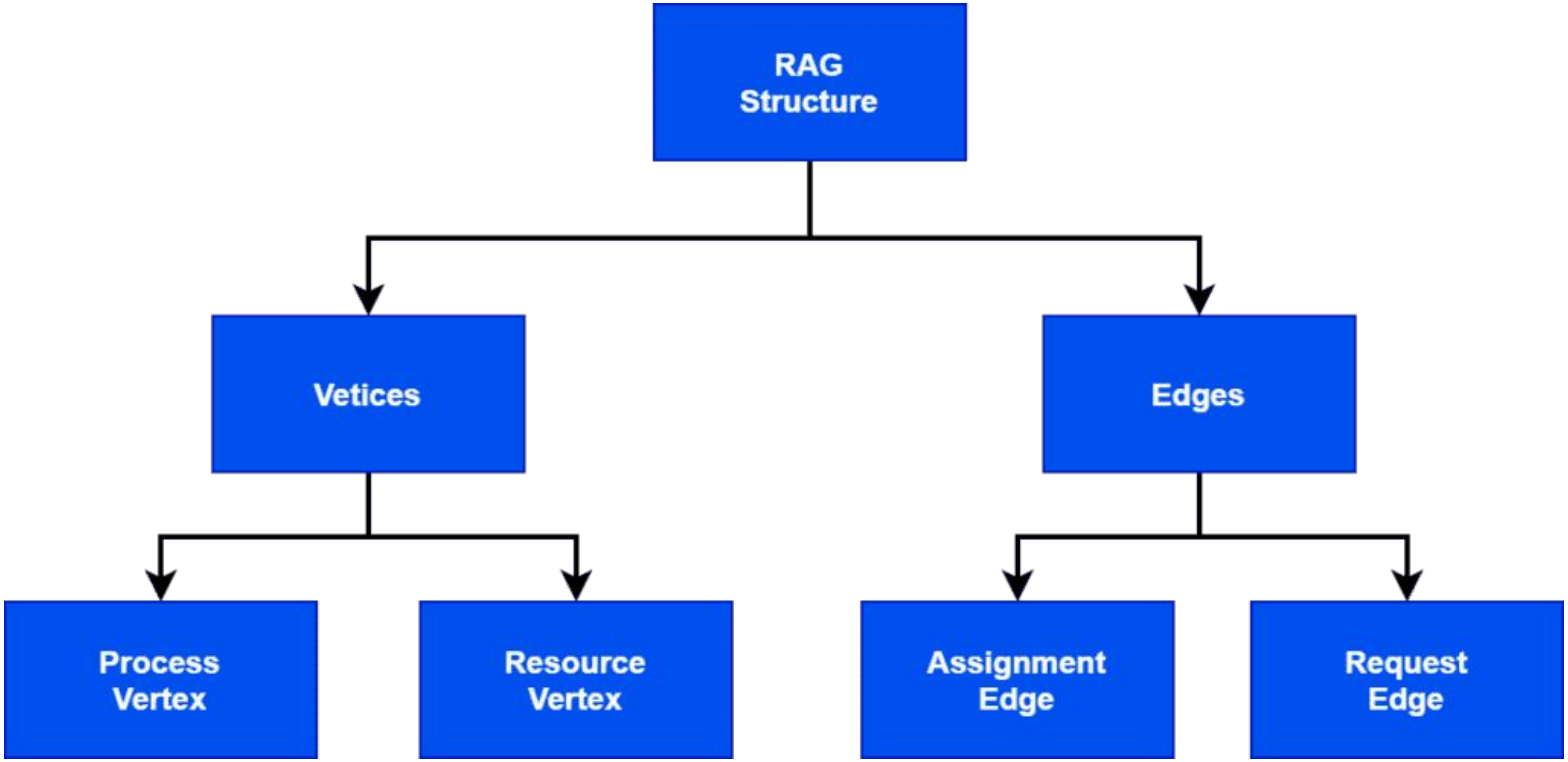
Resource Allocation Graph (RAG) Algorithm

RAG helps predict deadlocks by showing allocated and needed resources for each process.

Components:

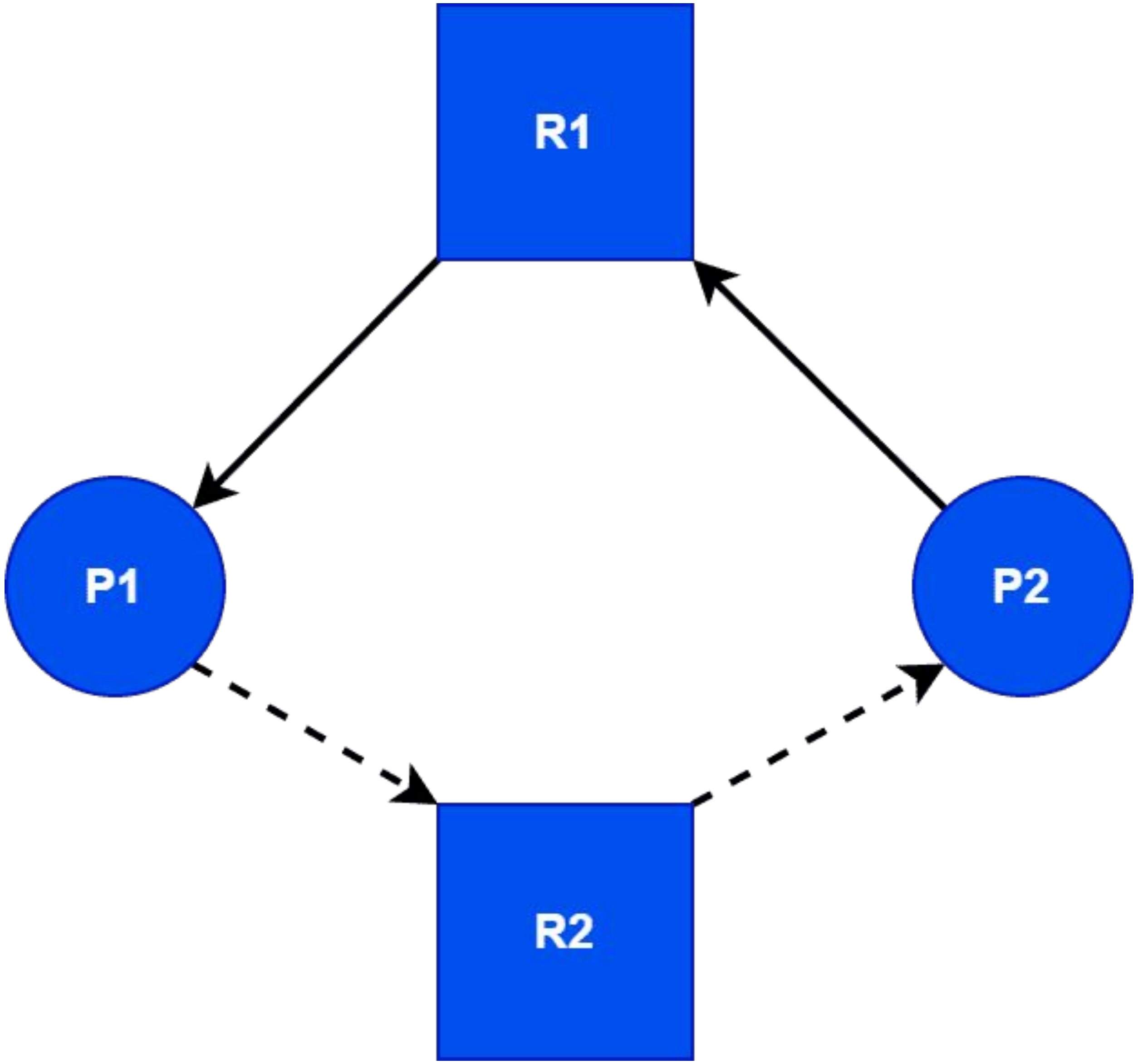
- **Vertices:** Process and resource
- **Edges:** Assignment (resource → process) and Request (process → resource)

Works well when each resource has a **single instance**.


RAG Structure
Vetices
Edges
Process Vertex
Resource Vertex
Assignment Edge
Request Edge

Resource Allocation Graph (RAG) Algorithm …

- If a resource has multiple instances, deadlock prediction becomes uncertain; the **Banker's Algorithm** is used instead.
- Example: If P1 will need R2 later, the OS can delay granting R2 to P2, preventing a circular wait—but this may reduce resource utilization.


R1
P1
P2
R2

Banker’s Algorithm

Used when resources have **multiple instances**.

Named after banks, which ensure they never run out of resources.

The OS needs:

- Maximum resources each process may need
- Total available resources

It calculates a **safe sequence** of process execution to avoid deadlocks.

Banker’s Algorithm…

Maintains matrices for:

- Total available resources
- Maximum resources per process
- Allocated resources per process
- Currently needed resources per process

DISCUSSION & REVISION

1. Which deadlock condition is easiest to prevent practically?
2. What state allows resource allocation without causing deadlock?
3. Which algorithm uses a graph to represent processes and resources?
4. Banker's algorithm is used when resources have single or multiple instances?
5. In RAG, what type of edge represents a process requesting a resource?

REFERENCES

- https://www.geeksforgeeks.org/operating-systems/deadlock-prevention/
- https://www.baeldung.com/cs/os-deadlock
- https://www.scaler.com/topics/operating-system/deadlock-prevention-in-operating-system/