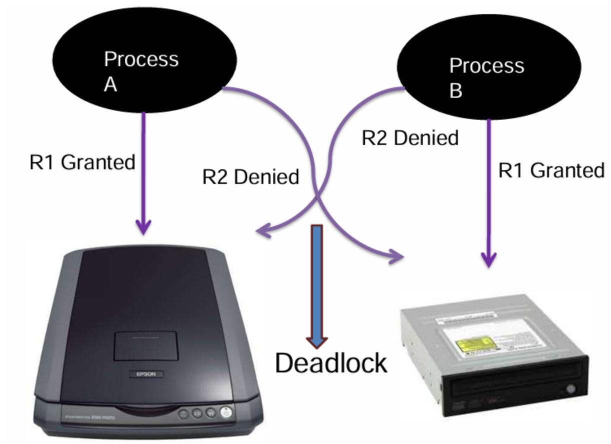
Deadlocks

Deadlock Situation

Two processes each want to write a scanned document on the CD. Process A request permission to use the scanner and is granted it. Process B is request the CD writer and is also granted it. Now A asks for the CD writer, but the request is denied until B releases it. Unfortunately, instead of releasing CD writer, B asks for the scanner. At this stage both processes are blocked. This situation is called **DEADLOCK**.



Resources

- A resource is anything that can be used by a single process at any instant of time.
- A resource can be the hardware device or a piece of information.

Types:

1. **Preemptable Resource:** It is one that can be taken away from the process owing it. Main memory is an example of preemptable resource.
2. **Non-Preemptable Resource:** It is one that cannot be taken away from the process owing it.

Sequence of events required to use a resource:

1. Request the resource
2. Use the resource
3. Release the resource

Conditions for Deadlock

A deadlock can happen due to the following conditions:

1. **Mutual Exclusion Condition**

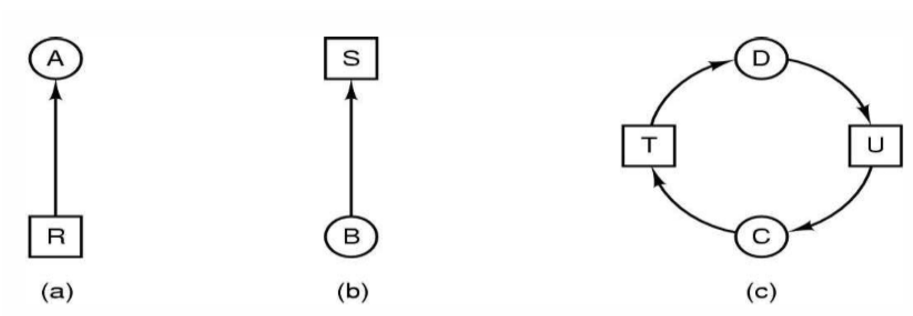
- **Definition**: Only **one process** can use a resource at a time.
- **Example**: If Process A is using a printer, Process B cannot use it until A finishes and releases the printer.
- **Explanation**: The requesting process must wait until the resource is free.

2. **Hold and Wait Condition**
   - **Definition**: A process holding a resource can **request more resources** while waiting for others.
   - **Example**: Process A holds a file and requests access to a database. Meanwhile, Process B holds the database and requests access to the file. Both are now stuck.
3. **Non-Preemptive Condition**
   - **Definition**: Resources allocated to a process **cannot be forcibly taken away**; the process must release them voluntarily.
   - **Example**: Once a process starts printing, the printer cannot be taken away until the job finishes.
4. **Circular Wait Condition**
   - **Definition**: There is a **circular chain** of processes, where each process is waiting for a resource held by the next process in the chain.
   - **Example**:
     - Process A is waiting for a resource held by Process B.
     - Process B is waiting for a resource held by Process C.
     - Process C is waiting for a resource held by Process A.

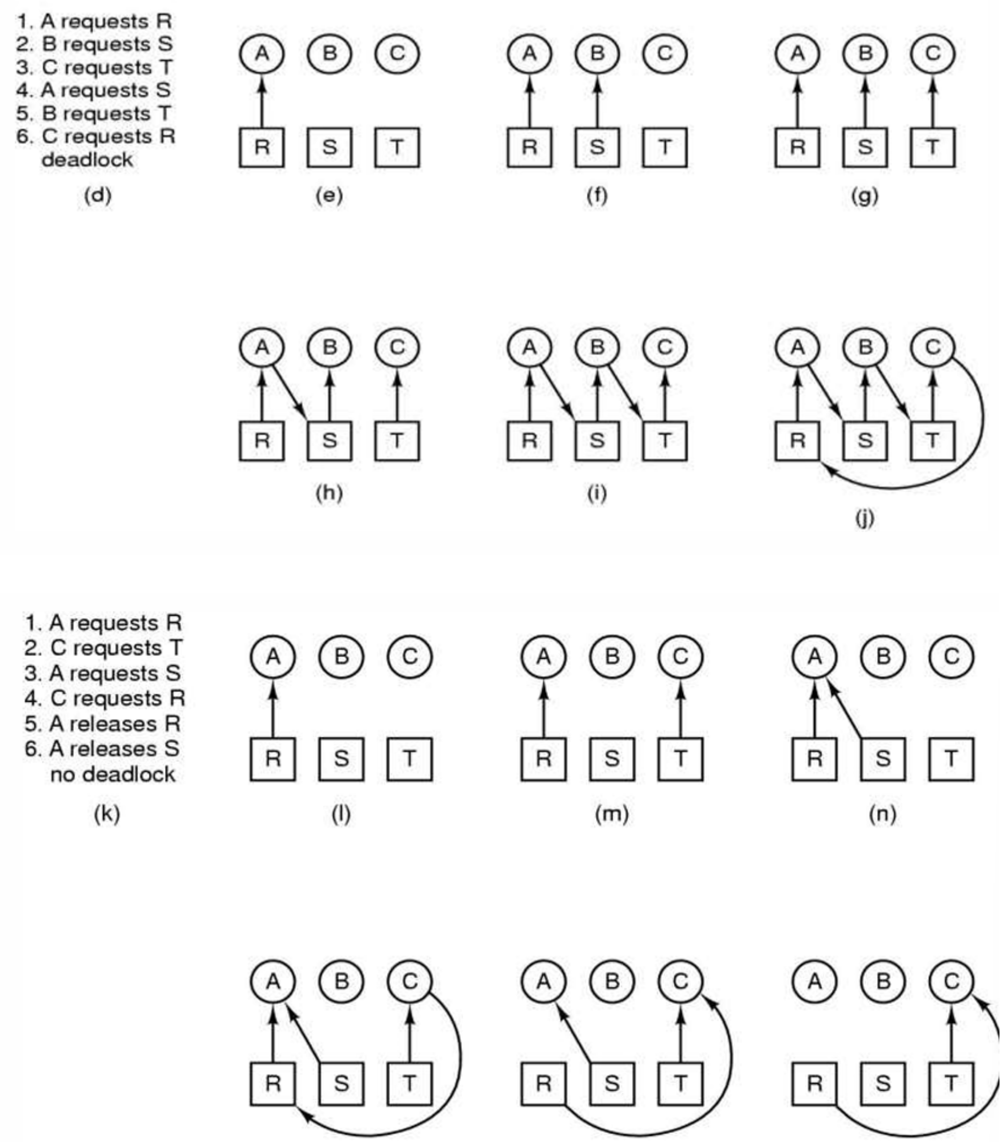
Resource Allocation Graph

Concept: A **graph** helps visualize the relationship between processes and resources.

Example:



- Resource **R** is assigned to **Process A**.
- **Process B** is waiting for Resource **S**.
- **Processes C and D** are in deadlock, both holding and requesting resources **T** and **U**.



Resource Allocation Graph (RAG) and Deadlock Detection

Components of a Resource Allocation Graph

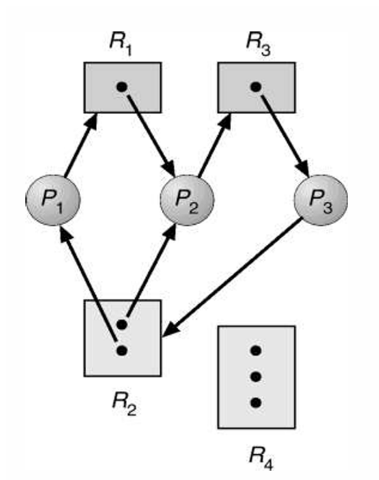
- **Nodes (V):**
  - **Processes**: Represented by circles (e.g., P1, P2, P3).
  - **Resources**: Represented by squares (e.g., R1, R2) with dots indicating available instances.
- **Edges (E):**
  - **Request Edges**: From a process to a resource (e.g., P1 → R2) indicating the process is requesting a resource.
  - **Assignment Edges**: From a resource to a process (e.g., R1 → P2) showing that the resource has been allocated to the process.

Rules for Deadlock Detection

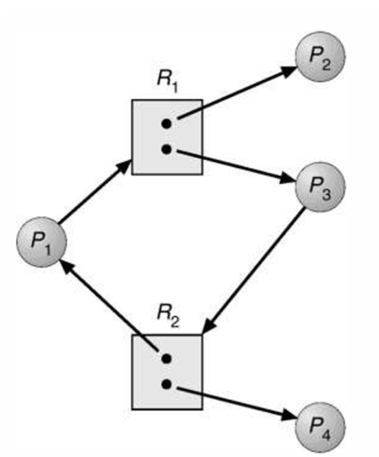
1. **No Cycles**:
   - If the graph **contains no cycles**, no process is in a deadlock state.
2. **Cycle Detection**:
   - If a **cycle exists**, two scenarios can occur:
     a)
     Multiple Resource Instances: Deadlock **may** exist.
     b)
     Single Resource Instance: Deadlock **has occurred**.

Examples

1. Deadlock Scenario



2. Cycle Without Deadlock



Deadlock Avoidance Algorithm: Safe and Unsafe States

1. Safe State

- A state is considered **safe** if:
  - **No deadlock** occurs.
  - There exists a **scheduling order** in which all processes can complete successfully.
  - The system can **guarantee** that every process will finish its tasks.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1
(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5
(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

2. Unsafe State

- A state is considered **unsafe** if:
  - **Deadlock may occur**, meaning some processes might get stuck waiting indefinitely.
  - There is **no scheduling order** that guarantees all processes can complete.
  - The system **cannot ensure** that all processes will finish.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2
(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0
(c)

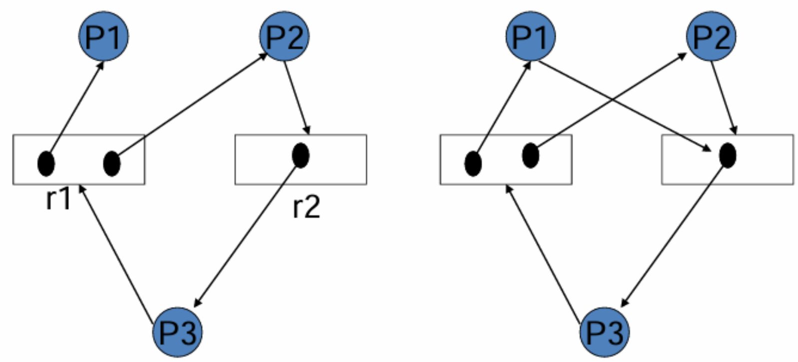
	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

Graph-Theoretic Models for Deadlocks

Graph-theoretic models help visualize and analyze the potential for deadlocks in a system by using **wait-for graphs** and **resource-allocation graphs.**

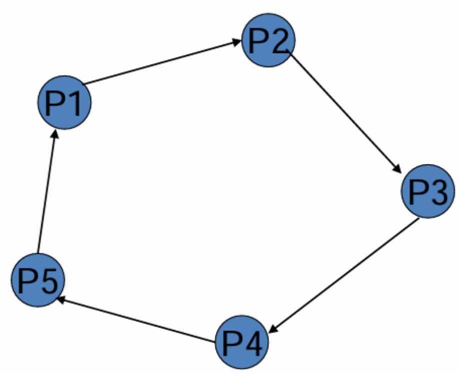
1. Resource-Allocation Graph (RAG)



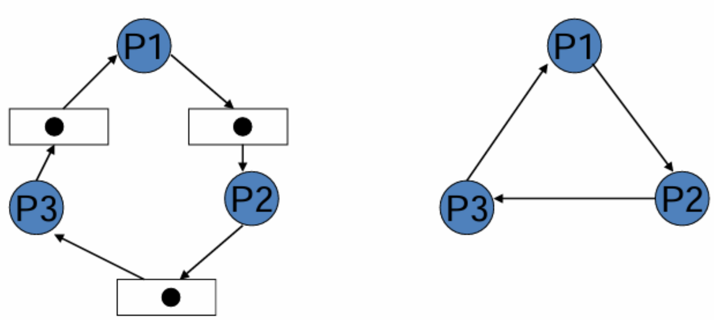
Resource allocation graph
Without deadlock

With deadlock

2. Wait-for Graph (WFG)



Any resource allocation graph with a single copy of resources can be transferred to a wait-for graph.



Deadlock Avoidance Algorithms

1. Single Instance of a Resource Type

- **Technique:** Use a **Resource-Allocation Graph (RAG)**.

2. Multiple Instances of a Resource Type

- **Technique:** Use the **Banker's Algorithm**.

Banker's Algorithm

Example: 1

CLASS EXERCISE-1
Find safe sequence?

Total=[10,5,7]

5 processes P_0 through P_4
3 resource types A (10 units), B (5 units), and C (7 units).

	Allocation	***Max***	***Available***
	A B C	***A B C***	***A B C***
P_0	**0 1 0**	**7 5 3**	**3 3 2**
P_1	**2 0 0**	**3 2 2**	
P_2	**3 0 2**	**9 0 2**	
P_3	**2 1 1**	**2 2 2**	
P_4	**0 0 2**	**4 3 3**	

Need = Max – Allocation

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

P_1, P_3, P_4, P_2, P_0 satisfies safety criteria.

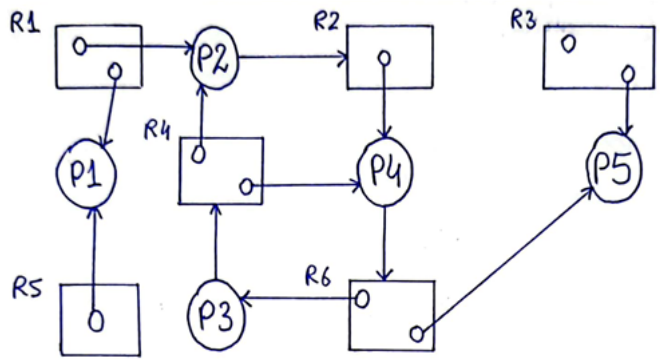
If (Need[i] ≤ Available)
Then Available=Available + Allocation[i];

WORKING: Available

Available= (3,3,2)
+(2,0,0) Allocation[p1]
Now available= (5,3,2)
+(2,1,1) Allocation[p3]
Now available= (7,4,3)
+(0,0,2) Allocation[p4]
Now available= (7,4,5)
+(3,0,2) Allocation[p2]
Now available= (10,4,7)
+(0,1,0) Allocation[p0]
Now available= (10,5,7)

As available== TOTAL --> STOP with safe sequence

Example: 2



a) Matrix Representation:

	R_1	R_2	R_3	R_4	R_5	R_6	(Allocation)
P_1	1	0	0	0	1	0	
P_2	1	0	0	1	0	0	
P_3	0	0	0	0	0	1	
P_4	0	1	0	1	0	0	
P_5	0	0	1	0	0	1	

	R_1	R_2	R_3	R_4	R_5	R_6	(Request)
P_1	0	0	0	1	0	0	
P_2	0	1	0	0	0	0	
P_3	0	0	0	1	0	0	
P_4	0	0	0	0	0	1	
P_5	0	0	0	0	0	0	

R_1	R_2	R_3	R_4	R_5	R_6
0	0	1	0	0	0

→ The request of P_5 can be entertained with the available resources.

$$[0\,0\,0\,0\,0\,0] \leq [0\,0\,1\,0\,0\,0]$$

$$\text{Available} = \text{Old} + \text{Allocated}[i]$$

$$= [0\,0\,1\,0\,0\,0] + [0\,0\,1\,0\,0\,1]$$

New Allocation:

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	1	0	0	0	1	0
P_2	1	0	0	1	0	0
P_3	0	0	0	0	0	1
P_4	0	1	0	1	0	0
P_5	0	0	0	0	0	0

New Available:

R_1	R_2	R_3	R_4	R_5	R_6
0	0	2	0	0	1

→ The request of P_4 can be entertained with the available resources.

$$[0\,0\,0\,0\,0\,1] \leq [0\,0\,2\,0\,0\,1]$$

$$\text{Available} = \text{Old} + \text{Allocated}[i]$$

$$= [0\,0\,2\,0\,0\,1] + [0\,1\,0\,1\,0\,0]$$

New Allocation:

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	1	0	0	0	1	0
P_2	1	0	0	1	0	0
P_3	0	0	0	0	0	1
P_4	0	0	0	0	0	0
P_5	0	0	0	0	0	0

New Available:

R_1	R_2	R_3	R_4	R_5	R_6
0	1	2	1	0	1

P_1, P_2 and P_3 any processes request can be entertained.

→ We will entertain P_3's request first.

New Allocation:

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	1	0	0	0	1	0
P_2	1	0	0	1	0	0
P_3	0	0	0	0	0	0
P_4	0	0	0	0	0	0
P_5	0	0	0	0	0	0

New Available:

R_1	R_2	R_3	R_4	R_5	R_6
0	1	2	1	0	2

P_1 and P_2 both requests can be entertained

→ We will entertain P_2's request first

New Allocation:

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	1	0	0	0	1	0
P_2	0	0	0	0	0	0
P_3	0	0	0	0	0	0
P_4	0	0	0	0	0	0
P_5	0	0	0	0	0	0

New Available:

R_1	R_2	R_3	R_4	R_5	R_6
1	1	2	2	0	2

Now, For P_1

New Allocation:

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	0	0	0	0	0	0
P_2	0	0	0	0	0	0
P_3	0	0	0	0	0	0
P_4	0	0	0	0	0	0
P_5	0	0	0	0	0	0

New Available.

R_1	R_2	R_3	R_4	R_5	R_6
2	1	2	2	1	2

Hence Safe Sequence: $P_5 \rightarrow P_4 \rightarrow P_3 \rightarrow P_2 \rightarrow 1$

b) There are no dead locks.

The safe sequences will be:

P_5, P_4, P_1, P_2, P_3

P_5, P_4, P_1, P_3, P_2

P_5, P_4, P_2, P_3, P_1

P_5, P_4, P_2, P_1, P_3

P_5, P_4, P_3, P_1, P_2

P_5, P_4, P_3, P_2, P_1