# Deadlock


CODE
SECURITY
COMPUTER
UPDATE
CALL
APPLICATION
SERVICES
DEVELOPMENT
REAL-TIME
DISTRIBUTED
THREAD
DRIVER
PROCESSOR
MULTI-USER
RESOURCES
STORAGE
USER
OPERATING
DESIGN
INSTRUCTION
SINGLE-TASKING
HARDWARE
SYSTEM
INSTALLATION
COMPONENT
GUI
PORTABILITY
MEMORY
DEVICE
EMBEDDED
INTERMEDIARY
VIRTUAL
MODERN
MANAGEMENT
INTERFACE
SERVER
TIME
MULTI-TASKING
ARCHITECTURE
SOFTWARE
KERNEL
INTERRUPT
COMMAND LINE
NETWORKING
ALLOCATION
PROCESS

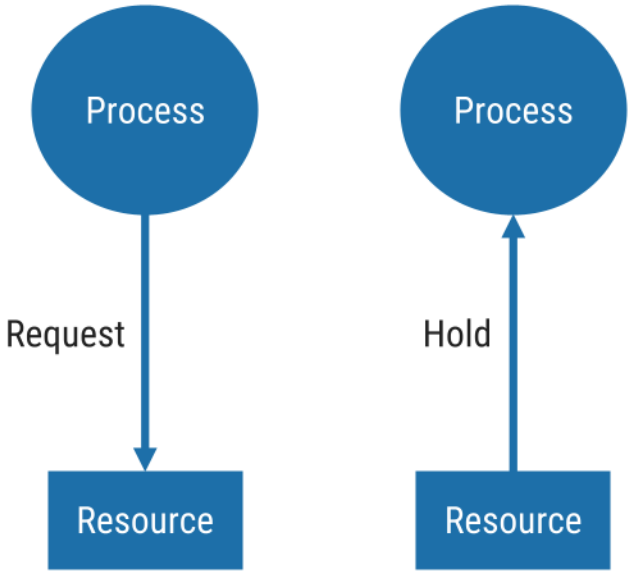
## Outline

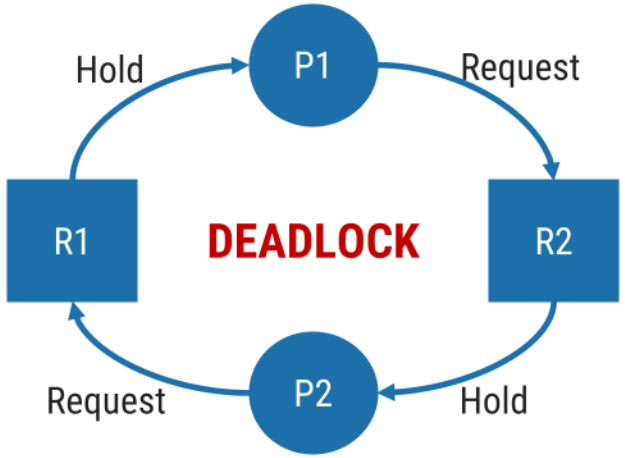
- Basic concepts of Deadlock
- Deadlock characteristics
- Deadlock ignorance
  - Ostrich algorithm
- Deadlock detection and recovery
- Deadlock avoidance
  - Banker's algorithm
- Deadlock prevention

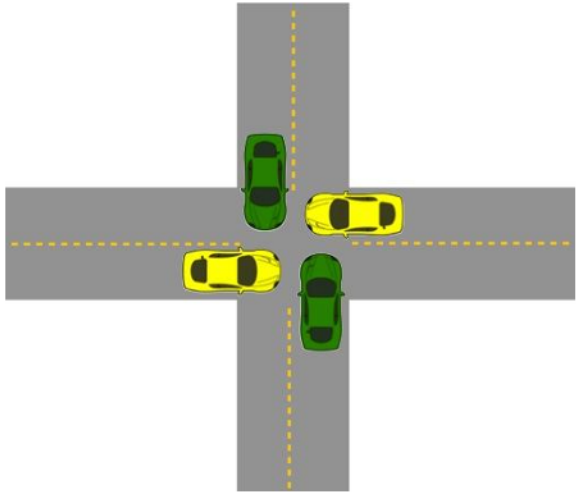
# Basic concepts of Deadlock

# What is Deadlock?

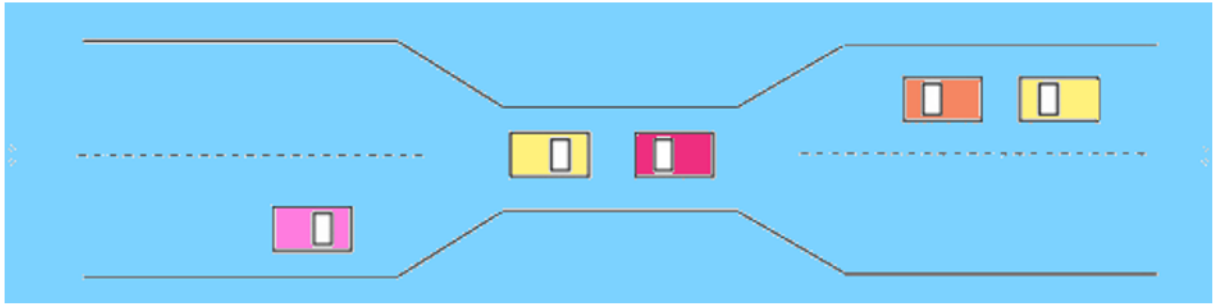
- A set of processes is deadlocked if **each process in the set is waiting for an event that only another process in the set can cause**.
- Deadlocks are a **set of blocked processes each holding a resource and waiting to acquire a resource held by another process**.



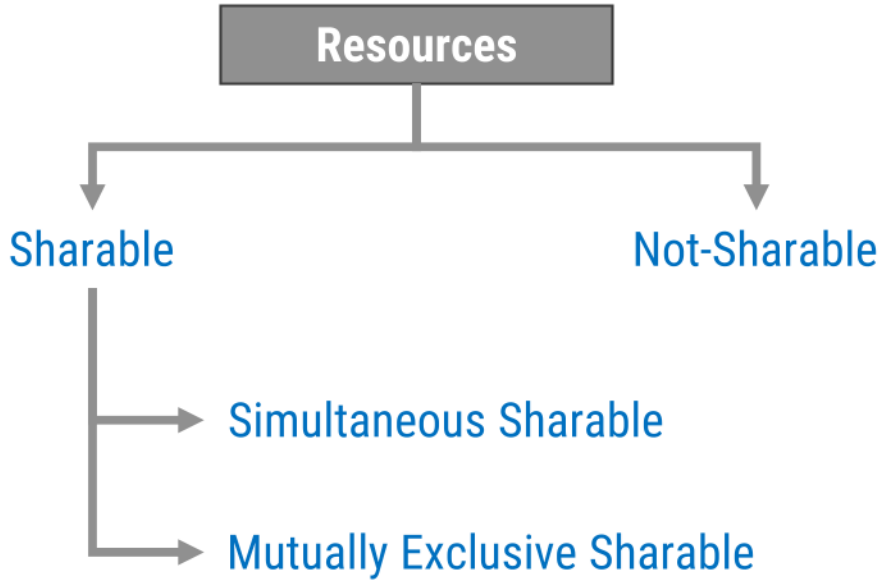



**Exercise** Give an real life example of deadlock?



# Shrable and Not Sharable Resources

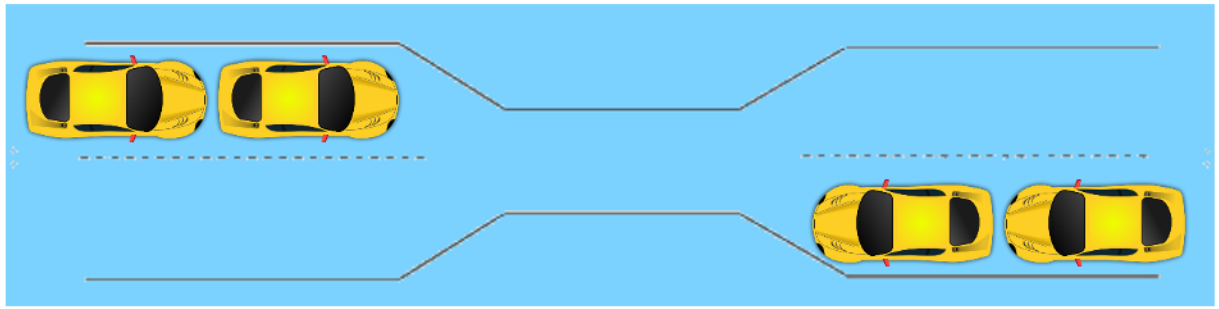
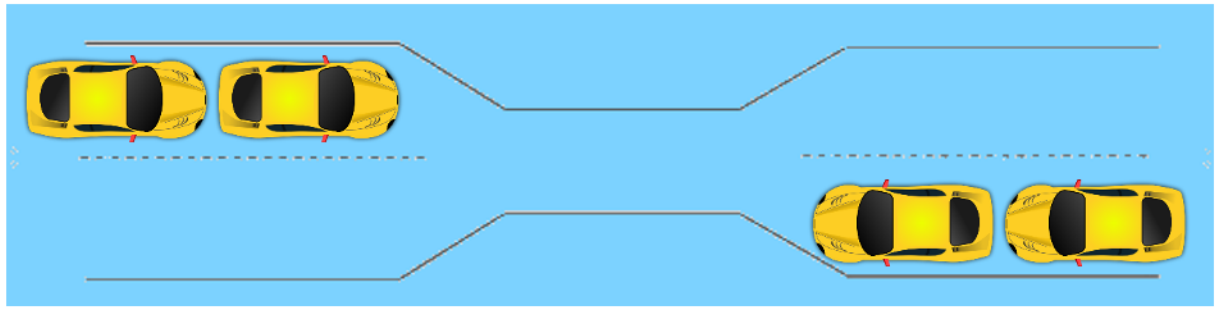
- Not-Shareable: Can not be shared with other processes.
  - **E.g.,** Fence Register
- Shareable: Can be shared with other processes.
  - ***E.g.,*** Printer, Processor
- Simultaneous Sharable: At a time, two or more processes can share.
  - **E.g.,** Read-only Page or files.
- Mutually Exclusive Sharable: Only one process can share at a time.
  - **E.g.,** Printer, Processor.



# Preemptable and non-preemptable resource

- Preemptable:- Preemptive resources are those **which can be taken away from a process without causing any ill effects** to the process.
  - Example:- Memory.
- Non-preemptable:- Non-pre-emptive resources are those **which cannot be taken away from the process without causing any ill effects** to the process.
  - Example:- CD-ROM (CD recorder), Printer.

# Deadlock v/s Starvation

| Deadlock | Starvation |
| --- | --- |
|  |  |
| Infinite Wait | Indefinite Wait |

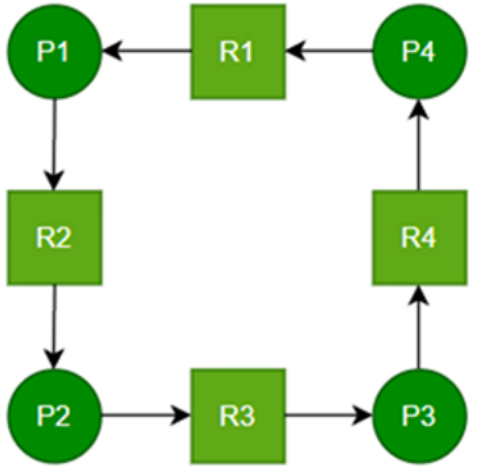
# Deadlock v/s Starvation

| Deadlock | Starvation |
| --- | --- |
| All processes keep waiting for each other to complete and none get executed. | High priority process keep executing and low priority process are blocked. |
| Resources are blocked by the process. | Resources are continuously utilized by the higher priority process. |
| Necessary conditions are mutual exclusion, hold and wait, no preemption, circular wait. | Priorities are assigned to the process. |
| Also known as circular wait. | Also known as lived lock. |
| It can be prevented by avoiding the necessary conditions for deadlock. | It can be prevented by Aging. |

# Necessary Conditions for Deadlock (Deadlock characteristics)

# Necessary Conditions for Deadlock

- **Mutual Exclusion:** Resources must be non-shareable; only one process can use a resource at a time (e.g., a printer).
- **Hold and Wait:** A process holds at least one resource while waiting for additional resources held by other processes.
- **No Preemption:** Resources cannot be forcibly taken from a process; they must be released voluntarily after the process completes.
- **Circular Wait:** A closed chain of processes exists where each process holds a resource needed by the next process in the chain.
- **Example:** Imagine four processes—**P1**, **P2**, **P3**, and **P4**—and four resources—R1, R2, R3, and R4.
  - P1 is holding R1 and waiting for R2 (which is held by P2).
  - P2 is holding R2 and waiting for R3 (which is held by P3).
  - P3 is holding R3 and waiting for R4 (which is held by P4).
  - P4 is holding R4 and waiting for R1 (which is held by P1).



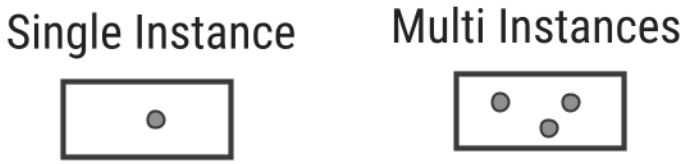
# Conditions that lead to deadlock (Deadlock characteristics)

1. Mutual exclusion
   - **Each resource is either** currently **assigned to exactly one process** or **is available**.
   - Only **one process at a time can use a resource.**
2. Hold and wait
   - Process currently holding resources granted earlier can **request more resources**.
3. No preemption
   - Previously granted resources **cannot be forcibly taken away** from process.
4. Circular wait
   - There must be a **circular chain of 2 or more processes**. Each process is waiting for resource that is held by next member of the chain.

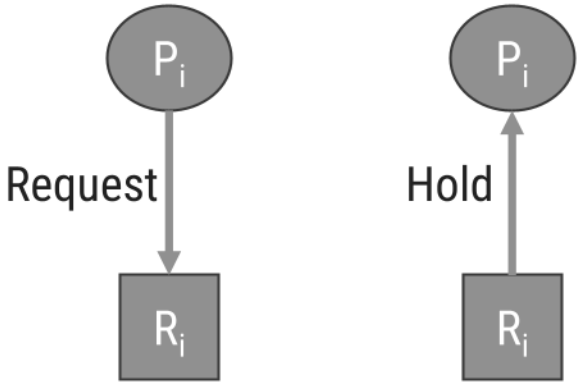
**All four of these conditions must be present for a deadlock to occur.**

# Resource Allocation Graph

- Let it be a graph G = {V, E}.
- **Vertices (V):** Process ($P_i$) and Resources ($R_i$).
  - **Resources are of two types:**
    - **Single Instance:** Only one process can access.
    - **Multi-instances:** may be allocated to multiple processes.

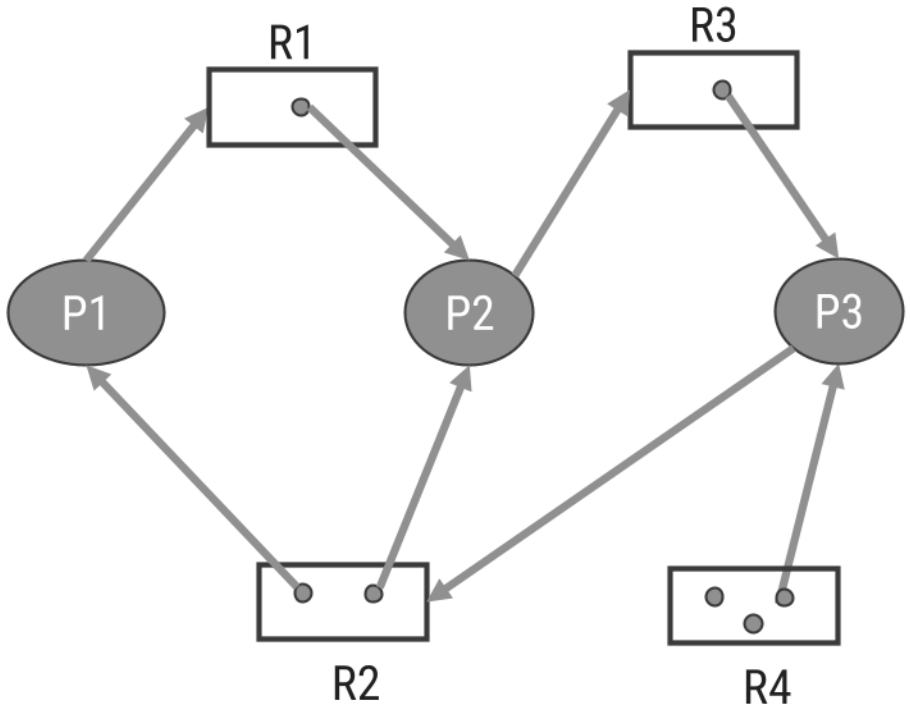



- **Edge (E):** Directed edge from resource to process or process to resource.
  - $\mathbf{P_i \rightarrow R_i}$ - Process-to-resource denotes that the process is requesting the allocation of resources.
  - $\mathbf{R_i \rightarrow P_i}$ - Resource to Process denotes that the resource is held by the process.



# Resource Allocation Graph: Example

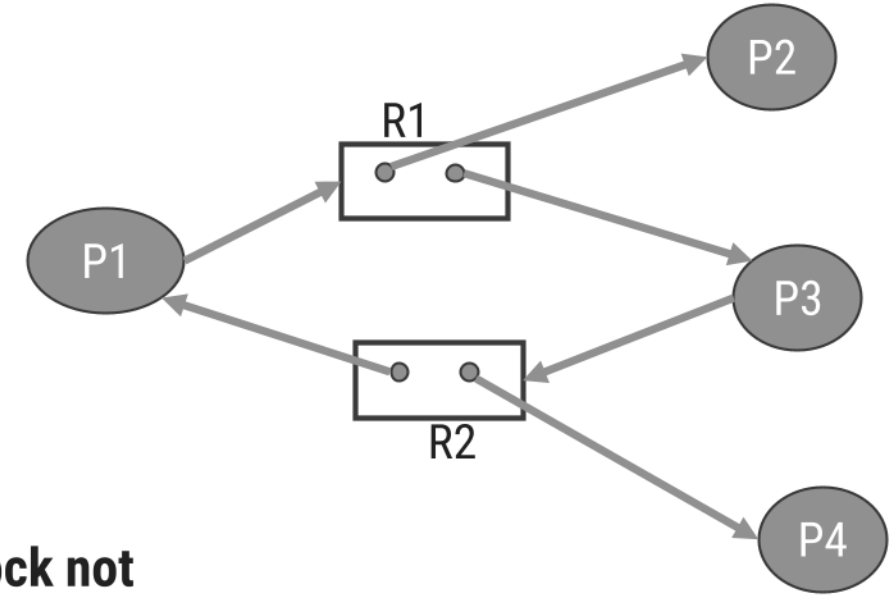
- **Consider the following resource allocation graph. Is the system in a deadlock?**




- **Ans.:**
  - **Deadlock**

# Resource Allocation Graph: Example-2

- **Consider the following resource allocation graph. Is the system in a deadlock?**




- **Ans.:**
  - **Deadlock not**

- **Note Points:**
  - Presence of a cycle in a resource allocation graph is a necessary condition for a deadlock, but not sufficient.
  - If the resource allocation graph has one instance, then a cycle is a necessary and sufficient condition for a deadlock.

# How to Deal with Deadlock

# Strategies for dealing with deadlock

1. Just **ignore** the problem

2. **Deadlock Prevention:** Write from the beginning (at OS configuration time) the steps that are taken so that the deadlock will never occur.
   - **Example: Prevention** by structurally negating (**killing**) one of the four required conditions.
   - It cannot be practically possible.

3. **Deadlock Avoidance:** Monitor the situation continuously, whenever we like to, as deadlock may occur, then take some steps so that deadlock will not occur.
   - Dynamic **avoidance** by careful resource allocation.

4. **Deadlock Detection and Recovery**: Steps taken after deadlock occurrence.
   - Let deadlocks occur, detect them, and take action.
   - It is mostly used.

# Deadlock ignorance (Ostrich Algorithm)

# Deadlock ignorance (Ostrich Algorithm)

- When **storm approaches**, an **ostrich puts his head in the sand (ground)** and **pretend (imagine) that there is no problem at all**.

- **Ignore** the **deadlock** and **pretend** that **deadlock never occur**.

- If a deadlock occurs in the system, then the OS will just **ignore** the deadlock **and reboot the system** in order to function well.

- **It is reasonable if**
  - deadlocks occur very rarely
  - difficult to detect
  - cost of prevention is high

- **UNIX** and **Windows** takes this approach

# Deadlock prevention

# Deadlock prevention

- Deadlock can be prevented by **attacking the one of the four conditions** that leads to deadlock.
  1. Attacking the **Mutual Exclusion Condition**
     - No deadlock **if each resource can be assigned to more than one process**.
     - We **can not assign some resources to more than one process at a time** such as **CD-Recorder, Printer** etc…
     - So this solution is **not feasible**.
  2. Attacking the **Hold and Wait Condition**
     - Require processes to **request all their resources before starting execution**.
     - A process is **allowed to run if all resources it needed is available**. Otherwise nothing will be allocated and it will just wait.
     - Problem with this strategy is that a **process may not know required resources at start of run**.
     - Resource will **not be used optimally**.
  3. Attacking the **No Preemption Condition**
     - When a process P0 request some resource R which is held by another process P1 then resource R is **forcibly taken away from the process P1 and allocated to P0**.
     - Consider a process holds the **printer**, halfway through its job; **taking the printer away from this process without having any ill effect is not possible**.
     - This is not a **possible option**.

# Deadlock prevention

- Deadlock can be prevented by **attacking the one of the four conditions** that leads to deadlock.
  4. Attacking the **Circular Wait Condition**
     - Provide a **global numbering of all the resources**.
       1) Printer
       2) Scanner
       3) Plotter
       4) Tape Drive
       5) CD Rom
     - Now the rule is that: **processes can request resources whenever they want to, but all requests must be made in numerical order**.
     - A process **need not acquire them all at once**.
     - Circular wait is prevented if a **process holding resource n cannot wait for resource m, if m > n**.
     - A process **may request 1st a CD ROM, then tape drive**. But it **may not request 1st a tape drive, then CD ROM.**
     - **Resource graph can never have cycle.**

# Deadlock avoidance (Banker's algorithm)

# Safe and unsafe states

- A state is said to be safe **if it is not deadlocked** and **there is some scheduling order in which every process can run to completion** even if all of them suddenly request their maximum number of resources immediately.

- Total resources are 10
- 7 resources already allocated
- So there are 3 still free

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |
| Free : 3 | | |

- A need 6 resources more to complete it.
- B need 2 resources more to complete it.
- C need 5 resources more to complete it.

# Safe state

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |
| Free : 3 | | |

2

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |
| Free : 1 | | |

4

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 0 | - |
| C | 2 | 7 |
| Free : 5 | | |

5

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 0 | - |
| C | 7 | 7 |
| Free : 0 | | |

7

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 0 | - |
| C | 0 | - |
| Free : 7 | | |

6

| Process | Has / Hold | Max required |
|---|---|---|
| A | 9 | 9 |
| B | 0 | - |
| C | 0 | - |
| Free : 1 | | |

# Unsafe state

| Process | Has / Hold | Max required |
|---|---|---|
| A | 3 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |
| Free : 3 | | |

1

| Process | Has / Hold | Max required |
|---|---|---|
| A | 4 | 9 |
| B | 2 | 4 |
| C | 2 | 7 |
| Free : 2 | | |

2

| Process | Has / Hold | Max required |
|---|---|---|
| A | 4 | 9 |
| B | 4 | 4 |
| C | 2 | 7 |
| Free : 0 | | |

4

| Process | Has / Hold | Max required |
|---|---|---|
| A | 4 | 9 |
| B | 0 | - |
| C | 2 | 7 |
| Free : 4 | | |

# Deadlock avoidance

- Deadlock can be avoided by **allocating resources carefully**.
- Carefully **analyze each resource request** to see **if it can be safely granted**.

- Need an algorithm that can always avoid deadlock by making right choice all the time **(Banker's algorithm).**

- Banker's algorithm for single resource
- Banker's algorithm for multiple resource

# Deadlock Avoidance (Bankers Algorithm): Data structures

- Let there be n processes & m varieties or types (printer, disk, etc.) of resources.
- **Data structures used in Bankers Algorithm:**
  - **Allocation:** It is a 2-D array of size *nxm*. In this array, O.S. keeps a record of different resources that are already allocated to various processes.
  - **MAX:** It is a 2-D array of size *nxm*. Here, O.S., keep a record of the maximum possible demand for each resource by process. Unless the OS fulfills these demands, the process cannot be terminated.
  - **Need:** It is a 2-D array of size *nxm*.

$$[Need]_{nxm} = [MAX]_{nxm} - [Allocation]_{nxm}$$

- Available: It is a one-dimensional array of size **n**, where O.S. records the number of resources available.

**Allocation**

| | Resources | |
|---|---|---|
| **Process** | **A** | **B** |
| P0 | 2 | 1 |
| P1 | 0 | 1 |
| P2 | 1 | 0 |

**MAX**

| | Resources | |
|---|---|---|
| **Process** | **A** | **B** |
| P0 | 4 | 2 |
| P1 | 1 | 3 |
| P2 | 2 | 1 |

**Need**

| | Resources | |
|---|---|---|
| **Process** | **A** | **B** |
| P0 | 2 | 1 |
| P1 | 1 | 2 |
| P2 | 1 | 1 |

| | Resources | |
|---|---|---|
| **Available** | **A** | **B** |
| | 2 | 1 |

# Bankers Algorithm: Steps

- **Step-1:** If ***request***$_i$ (request generated by the process $P_i$) is less than or equal to ***Need***$_i$, then go to **Step-2;** otherwise, **error**, because the process has requested more than its maximum demand.

$$request_i \leq Need_i$$

- **Step-2:** If ***request***$_i$ is less than or equal to the **Available**, then go to step 3. Otherwise, process $P_i$ should wait.

$$request_i \leq Available$$

- **Step-3:** Save one copy of the present data structure & create a new copy of the data structure as follows.

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Needi - Requesti$$

$$Available = Available - Request_i$$

- **Step 4:** Now, these modified data structures are given to the Safety algorithm**. If** the safety algorithm **returns a safe state**, then **resources are actually allocated to the process**, and the **old** copies of the data structures are **deleted**. **Else if** the **state is unsafe**, then process $P_i$ should **wait** & **new** data structures are **deleted**.

# Safety Algorithm: Steps

- **Step-1:** Let $Work$ and $Finish$ are one-dimensional arrays of sizes **m** and **n**, respectively. Initialize
  - $Work = Available$
  - $Finish = False$ (all entries of the Finish array to false)

- **Step-2:** Find an $i$ such that

  $Finish[i] = False$ **and** $Need_i \leq Work$

  If no such $i$, then go to step 4.

- **Step-3:**
  - $Work = Work + Allocation_i$
  - $Finish[i] = True$ *or* $Finish[i] = T$

  **Go to step 2.**

- **Step 4:** If all the entries of array Finish are True, then state is declared as Safe otherwise it is unsafe.

# Banker's algorithm for single resource: Example

- What the algorithm does is **check to see if granting the request leads to an unsafe state**. If it does, the **request is denied**.

- If **granting the request leads to a safe state**, it is **carried out**.

- If we have situation as per figure
  - then it is safe state
  - because with 10 free units
  - one by one all customers can be served.

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | 6 |
| B | 0 | 5 |
| C | 0 | 4 |
| D | 0 | 7 |
| Free : 10 | | |

# Banker's algorithm for single resource (safe state): Example

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |
| Free : 2 | | |

2

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 4 | 4 |
| D | 4 | 7 |
| Free : 0 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 0 | - |
| D | 4 | 7 |
| Free : 4 | | |

3

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 0 | - |
| D | 7 | 7 |
| Free : 1 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 1 | 5 |
| C | 0 | - |
| D | 0 | - |
| Free : 8 | | |

4

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 5 | 5 |
| C | 0 | - |
| D | 0 | - |
| Free : 4 | | |

# Banker's algorithm for single resource (safe state): Example

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 0 | - |
| C | 0 | - |
| D | 0 | - |
| Free : 9 | | |

5

| Process | Has / Hold | Max required |
|---|---|---|
| A | 6 | 6 |
| B | 0 | - |
| C | 0 | - |
| D | 0 | - |
| Free : 4 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | - |
| B | 0 | - |
| C | 0 | - |
| D | 0 | - |
| Free : 10 | | |

- The order of execution is C, D, B, A. So if we can find proper order of execution then there is no deadlock.

# Banker's algorithm for single resource (unsafe state): Example

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 6 |
| B | 2 | 5 |
| C | 2 | 4 |
| D | 4 | 7 |
| Free : 1 | | |

# Bankers Algorithm (Multiple Resources): Example

- **Consider the following snapshot of the system, where the processes P0, P1, P3, and P4 are executing.**

**Allocation**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

**MAX**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 7 | 5 | 0 |
| P2 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 5 | 2 |
| P4 | 0 | 6 | 5 | 6 |

| | Resources | | | |
|---|---|---|---|---|
| **Available** | **A** | **B** | **C** | **D** |
| | 1 | 5 | 2 | 0 |

- **Answer the following:**
  1. What is the content of Array Need?
  2. Is the system in a safe state?, if Yes, then what is the safe sequence.
  3. If a request of the process P1 arrives for resources (0, 4, 2, 0), can the request be immediately granted?

# Bankers Algorithm (Multiple Resources): Example

- **Consider the following snapshot of the system, where the processes P0, P1, P3, and P4 are executing.**

**Allocation**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

**MAX**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 7 | 5 | 0 |
| P2 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 5 | 2 |
| P4 | 0 | 6 | 5 | 6 |

| | Resources | | | |
|---|---|---|---|---|
| **Available** | **A** | **B** | **C** | **D** |
| | 1 | 5 | 2 | 0 |

- **Sol.**
  1. Content of Need Array = Max- Allocation

**Need**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 2) **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | 2 | 0 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | F | F | F | F | F |

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | 2 | 0 |

➢ Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$

- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$

➢ Found, i=0, (for P0)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | **3** | **2** |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | **T** | F | F | F | F |

➢ Found, i=2, (for P2)

| Work | A | B | C | D |
|---|---|---|---|---|
| | **2** | **8** | **8** | **6** |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | **T** | F | **T** | F | F |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 2) **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 2 | 8 | 8 | 6 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | F | T | F | F |

➢ Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$

- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$

➢ Found, i=1, (for P1)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 8 | 8 | 6 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | F | F |

➢ Found, i=3, (for P3)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 11 | 8 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | F |

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | 2 | 0 |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 2) **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 11 | 8 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | F |

- Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$
- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$
- Found, i=4, (for P4)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 12 | 12 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | T |

- All the processes are completed. No deadlock.
- **System is in Safe State**
- **Safe Sequence: P0, P2, P1, P3, P4.**

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | 2 | 0 |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 3) If a request of the process P1 arrives for resources (0, 4, 2, 0), can the request be immediately granted?

➢ Check $request_i \leq Need_i$ → [P1 (0, 4, 2, 0) <= (0, 7, 5, 0) ]

➢ If so, $request_i \leq Available$, → [P1 (0, 4, 2, 0) <= (1, 5, 2, 0) ]

➢ No update as follows:

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Needi - Requesti$$

$$Available = Available - Request_i$$

➢ Apply the Safety Algorithm to check whether the system is in a safe state or an unsafe state.

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 5 | 2 | 0 |

Allocation

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

Need

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 1 | 0 | 0 |

Need

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 3 | 3 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

Allocation

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 4 | 2 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 3) P1 request (0, 4, 2, 0). **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 1 | 1 | 0 | 0 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | F | F | F | F | F |

- Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$
- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$
- Found, i=0, (for P0)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 1 | 1 | **1** | **2** |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | **T** | F | F | F | F |

- Found, i=2, (for P2)

| Work | A | B | C | D |
|---|---|---|---|---|
| | **2** | **4** | **6** | **6** |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | **T** | F | **T** | F | F |

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | **4** | **2** | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | **1** | **0** | 0 |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | **3** | **3** | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 3) P1 request (0, 4, 2, 0). **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 2 | 4 | 6 | 6 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | F | T | F | F |

- Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$
- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$
- Found, i=3, (for P3)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 2 | 10 | 9 | 8 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | F | T | T | F |

- Found, i=1, (for P1)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 11 | 8 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | F |

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 4 | 2 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 1 | 0 | 0 |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 3 | 3 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Sol.** 3) P1 request (0, 4, 2, 0). **Is the system in a safe state?**

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 11 | 8 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | F |

- Find *i*, where $Finish[i] = False$ and $Need_i \leq Work$
- If so, then $Work = Work + Allocation_i$ and $Finish[i] = T$
- Found, i=4, (for P4)

| Work | A | B | C | D |
|---|---|---|---|---|
| | 3 | 14 | 12 | 12 |

| Finish | P0 | P1 | P2 | P3 | P4 |
|---|---|---|---|---|---|
| | T | T | T | T | T |

- All the processes are completed. No deadlock.
- **System is in Safe State**
- **Safe Sequence: P0, P2, P1, P3, P4 or P0, P2, P3, P1, P4**

**Allocation**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 4 | 2 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

| Available | A | B | C | D |
|---|---|---|---|---|
| | 1 | 1 | 0 | 0 |

**Need**

| Process | A | B | C | D |
|---|---|---|---|---|
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 3 | 3 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

# Bankers Algorithm (Multiple Resources): Example

- **Consider the following snapshot of the system, where the processes P0, P1, P3, and P4 are executing.**

**Allocation**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 |
| P2 | 1 | 3 | 5 | 4 |
| P3 | 0 | 6 | 3 | 2 |
| P4 | 0 | 0 | 1 | 4 |

**MAX**

| | Resources | | | |
|---|---|---|---|---|
| **Process** | **A** | **B** | **C** | **D** |
| P0 | 0 | 0 | 1 | 2 |
| P1 | 1 | 7 | 5 | 0 |
| P2 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 5 | 2 |
| P4 | 0 | 6 | 5 | 6 |

| | Resources | | | |
|---|---|---|---|---|
| **Available** | **A** | **B** | **C** | **D** |
| | 1 | 5 | 2 | 0 |

- **Answer the following:**
  1. What is the content of Array Need?
  2. Is the system in a safe state?, if Yes, then what is the safe sequence.
  3. If a request of the process P1 arrives for resources (0, 4, 2, 0), can the request be immediately granted?

# Banker's algorithm for multiple resource: Example-2

Total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

Allocated Resources

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

**Available**

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 1 | 0 | 2 | 0 |

**Is the system in a safe state?, if Yes, then what is the safe sequence.**

**Allocation**

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 3 | 0 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | 1 | 1 | 0 | 1 |
| P5 | 0 | 0 | 0 | 0 |

**Need**

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 1 | 0 |
| P5 | 2 | 1 | 1 | 0 |

# Banker's algorithm for multiple resource: Example-2

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 1 | 0 | 2 | 0 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 3 | 0 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | 1 | 1 | 0 | 1 |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 1 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 1 | 0 | 1 | 0 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 3 | 0 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | 1 | 1 | 1 | 1 |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 2 | 1 | 2 | 1 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 3 | 0 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 1 | 0 | 2 | 1 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 4 | 1 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 1 | 3 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | - | - | - | - |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 0 | 2 | 0 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms | no of resources held by each process |
|---|---|---|---|---|---|
| P1 | - | - | - | - | |
| P2 | 0 | 2 | 1 | 2 | |
| P3 | 1 | 1 | 1 | 0 | |
| P4 | - | - | - | - | |
| P5 | 0 | 0 | 0 | 0 | |

| Process | Tape Drives | Plotters | Scanners | CD Roms | no of resources still needed by each process to proceed |
|---|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 | |
| P2 | 0 | 0 | 0 | 0 | |
| P3 | 3 | 1 | 0 | 0 | |
| P4 | 0 | 0 | 0 | 0 | |
| P5 | 2 | 1 | 1 | 0 | |

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| (5) | (2) | 3 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | - | - | - | - |
| P2 | - | - | - | - |
| P3 | (1) | (1) | 1 | 0 |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 | 0 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 2 | 1 | 3 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | - | - | - | - |
| P2 | - | - | - | - |
| P3 | 4 | 2 | 1 | 0 |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | - | - | - | - |
| P2 | - | - | - | - |
| P3 | - | - | - | - |
| P4 | - | - | - | - |
| P5 | 0 | 0 | 0 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 4 | 2 | 3 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | - | - | - | - |
| P2 | - | - | - | - |
| P3 | - | - | - | - |
| P4 | - | - | - | - |
| P5 | 2 | 1 | 1 | 0 |

no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 | 0 |
| P3 | 0 | 0 | 0 | 0 |
| P4 | 0 | 0 | 0 | 0 |
| P5 | 0 | 0 | 0 | 0 |

no of resources still needed by each process to proceed

# Banker's algorithm for multiple resource: Example

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms | no of resources held by each process |
|---|---|---|---|---|---|
| P1 | - | - | - | - | |
| P2 | - | - | - | - | |
| P3 | - | - | - | - | |
| P4 | - | - | - | - | |
| P5 | - | - | - | - | |

| Process | Tape Drives | Plotters | Scanners | CD Roms | no of resources still needed by each process to proceed |
|---|---|---|---|---|---|
| P1 | 0 | 0 | 0 | 0 | |
| P2 | 0 | 0 | 0 | 0 | |
| P3 | 0 | 0 | 0 | 0 | |
| P4 | 0 | 0 | 0 | 0 | |
| P5 | 0 | 0 | 0 | 0 | |

# Banker's algorithm for multiple resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 6 | 3 | 4 | 2 |

total no of each resource

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 5 | 3 | 2 | 2 |

resources hold

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 1 | 0 | 2 | 0 |

available (free) resources

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 3 | 0 | 1 | 1 |
| P2 | 0 | 1 | 0 | 0 |
| P3 | 1 | 1 | 1 | 0 |
| P4 | 1 | 1 | 0 | 1 |
| P5 | 0 | 0 | 0 | 0 |

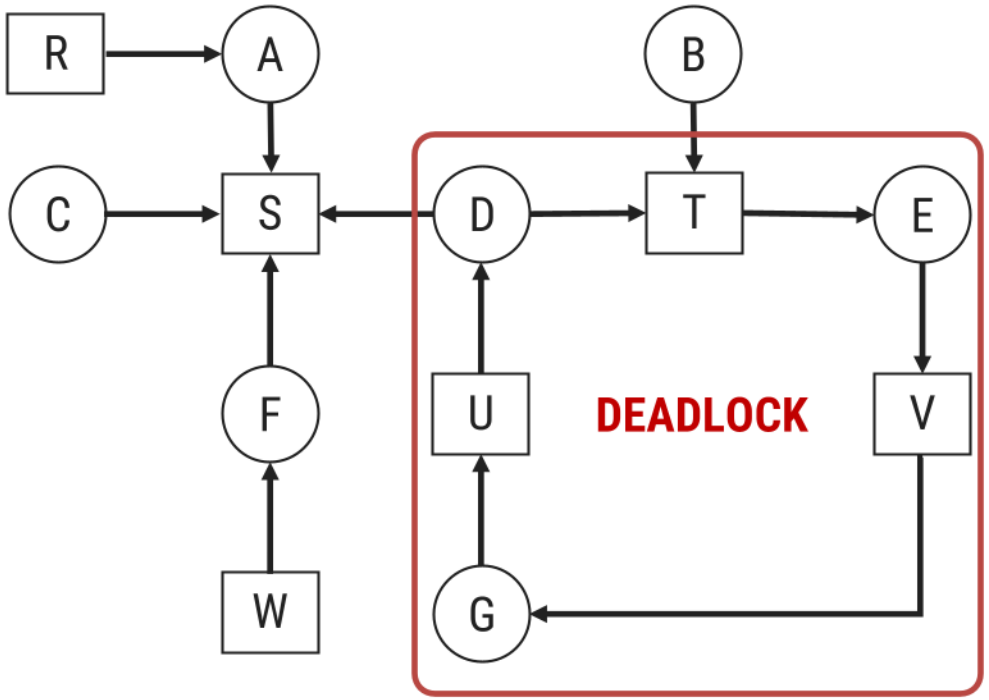
no of resources held by each process

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 1 | 1 | 0 | 0 |
| P2 | 0 | 1 | 1 | 2 |
| P3 | 3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 1 | (1) |
| P5 | 2 | 1 | 1 | 0 |

no of resources still needed by each process to proceed

# Deadlock detection and recovery

# Deadlock detection for single resource (RAG - Resource Allocation Graph)




- We are starting from node D.
- Empty list L = ()
- Add current node so Empty list = (D).

- From this node there is one outgoing arc to T so add T to list.

- So list become L = (D, T).
- Continue this step….so we get list as below

  L = (D, T, E)…… L = (**D**, T, E, V, G, U, **D**)

- In the above step in the list, node **D appears twice**, **so deadlock**.

# Deadlock detection for single resource (RAG - Resource Allocation Graph)

## Algorithm for detecting deadlock for single resource

**For each node, N in the graph, perform the following five steps with N as the starting node.**

1) Initialize L to the empty list, designate all arcs as unmarked.

2) Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.

3) From given node, see if any unmarked outgoing arcs. If so, go to step 4; if not, go to step 5.

4) Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 2.

5) If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 2.

# Deadlock detection for multiple resources: Example

T =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 4 | 2 | 2 | 1 |

total no of each resource

A =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 2 | 1 | 0 | 0 |

no of resources that are available (free)

C =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 1 | 0 |
| P2 | 2 | 0 | 0 | 1 |
| P3 | 0 | 1 | 1 | 0 |

no of resources held by each process

R =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 2 | 0 | 0 | 1 |
| P2 | 1 | 0 | 2 | 0 |
| P3 | 2 | 1 | 0 | 0 |

no of resources still needed by each process to proceed

# Deadlock detection for multiple resources: Example

T =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 4 | 2 | 2 | 1 |

total no of each resource

A =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 0 | 0 | 0 | 0 |

no of resources that are available (free)

C =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 1 | 0 |
| P2 | 2 | 0 | 0 | 1 |
| P3 | 2 | 2 | 1 | 0 |

no of resources held by each process

R =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 2 | 0 | 0 | 1 |
| P2 | 1 | 0 | 2 | 0 |
| P3 | 2 | 1 | 0 | 0 |

no of resources still needed by each process to proceed

# Deadlock detection for multiple resources: Example

T =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 4 | 2 | 2 | 1 |

total no of each resource

A =

| Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|
| 2 | 2 | 1 | 0 |

no of resources that are available (free)

C =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 0 | 0 | 1 | 0 |
| P2 | 2 | 0 | 0 | 1 |
| ~~P3~~ | ~~0~~ | ~~0~~ | ~~0~~ | ~~0~~ |

no of resources held by each process

DEADLOCK

R =

| Process | Tape Drives | Plotters | Scanners | CD Roms |
|---|---|---|---|---|
| P1 | 2 | 0 | 0 | 1 |
| P2 | 1 | 0 | 2 | 0 |
| ~~P3~~ | ~~2~~ | ~~1~~ | ~~0~~ | ~~0~~ |

no of resources still needed by each process to proceed

# Deadlock Detection & Recovery (Multiple Resources)

- Deadlock recovery is the most commonly used scheme to handle deadlock.

- **Deadlock Detection Algorithm:** It is the same as the safety algorithm.
- Let Request is a two-dimensional array of size $n \times m$, where request generated by different processes for various resources at any instant of time is recorded.
- **Step-1:** Let work and Finish be 1-D arrays of size $n$ & $m$, respectively. Initialize
  - $Work = Available$
  - $Finish = False$, (for all processes $P_i$ whose allocation is not zero, otherwise it is **True**.
- **Step-2:** Find process $P_i$ such that

  If no such $i$, then exit & go to **Step-4.**
- **Step-3:** Update the Work and Finish as follows
  - $\boldsymbol{Work = Work + Allocation_i}$ & $\boldsymbol{Finish\,[i] = True}$, and then go to **Step-2.**
- **Step-4:** If all the entries of Finish are True, then there is no deadlock in the system. Otherwise there is deadlock.

The process corresponding to which Finish entries are false are in deadlock.

**Request**

| | Resources | | |
|---|---|---|---|
| **Process** | **A** | **B** | **C** |
| P0 | 0 | 0 | 1 |
| P1 | 1 | 0 | 1 |
| P2 | 1 | 2 | 0 |

# Deadlock Detection & Recovery (Multiple Resources): Exercise

- **Ques. 1) Consider the following snapshot:**

**Allocation**

| | Resources | | |
|---|---|---|---|
| **Process** | **A** | **B** | **C** |
| P0 | 0 | 1 | 0 |
| P1 | 2 | 0 | 0 |
| P2 | 3 | 0 | 3 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

**Request**

| | Resources | | |
|---|---|---|---|
| **Process** | **A** | **B** | **C** |
| P0 | 0 | 0 | 0 |
| P1 | 2 | 0 | 2 |
| P2 | 0 | 0 | 0 |
| P3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 2 |

| | Resources | | |
|---|---|---|---|
| **Available** | **A** | **B** | **C** |
| | 0 | 0 | 0 |

- **Is the System in a Deadlock?**

- **Ans.** **No deadlock, Sequence: P0, P2, P1, P3, P4.**

# Deadlock Detection & Recovery (Multiple Resources): Exercise

- **Ques. 1) Consider the following snapshot:**

- **Is the System in a Deadlock? If process P2 requests one copy of resource C.**

**Allocation**

| | Resources | | |
|---|---|---|---|
| **Process** | **A** | **B** | **C** |
| P0 | 0 | 1 | 0 |
| P1 | 2 | 0 | 0 |
| P2 | 3 | 0 | 3 |
| P3 | 2 | 1 | 1 |
| P4 | 0 | 0 | 2 |

**Request**

| | Resources | | |
|---|---|---|---|
| **Process** | **A** | **B** | **C** |
| P0 | 0 | 0 | 0 |
| P1 | 2 | 0 | 2 |
| P2 | 0 | 0 | 1 |
| P3 | 1 | 0 | 0 |
| P4 | 0 | 0 | 2 |

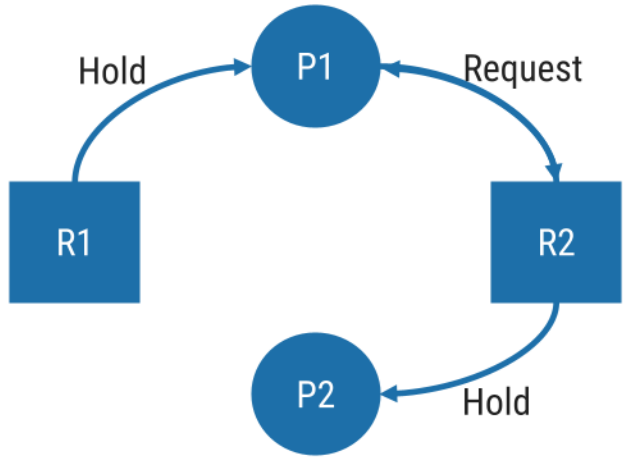
| | Resources | | |
|---|---|---|---|
| **Available** | **A** | **B** | **C** |
| | 0 | 0 | 0 |

- **Ans.** The system is in a **deadlock** as no process is present whose

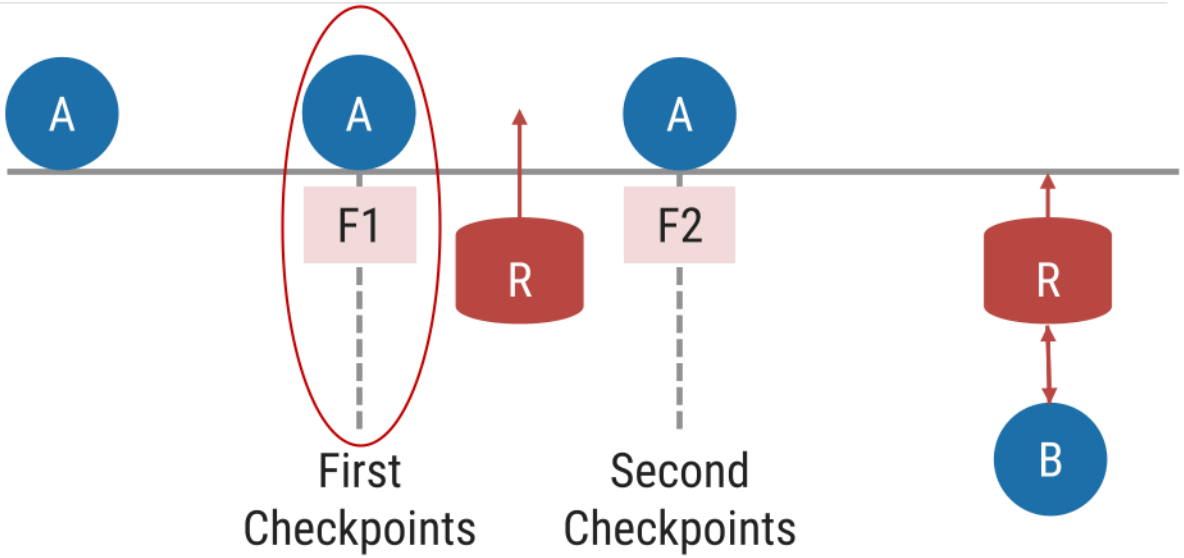
  request <=work.

# Deadlock recovery

1. Recovery through pre-emption
   - In this method **resources are temporarily taken away from its current owner and give it to another process**.
   - The ability to take a resource away from a process, have another process use it, and then give it back without the process noticing it is **highly dependent on the nature of the resource**.
   - Recovering this way is frequently **difficult or impossible**.




2. Recovery through rollback
   - **PCB (Process Control Block)** and **resource state** are **periodically saved** at "**checkpoint**".
   - When **deadlock is detected**, **rollback the preempted process up to the previous safe state** before it acquired that resource.
   - **Discard the resource** manipulation that occurred after that checkpoint.
   - **Start the process after it is determined** it can run again.

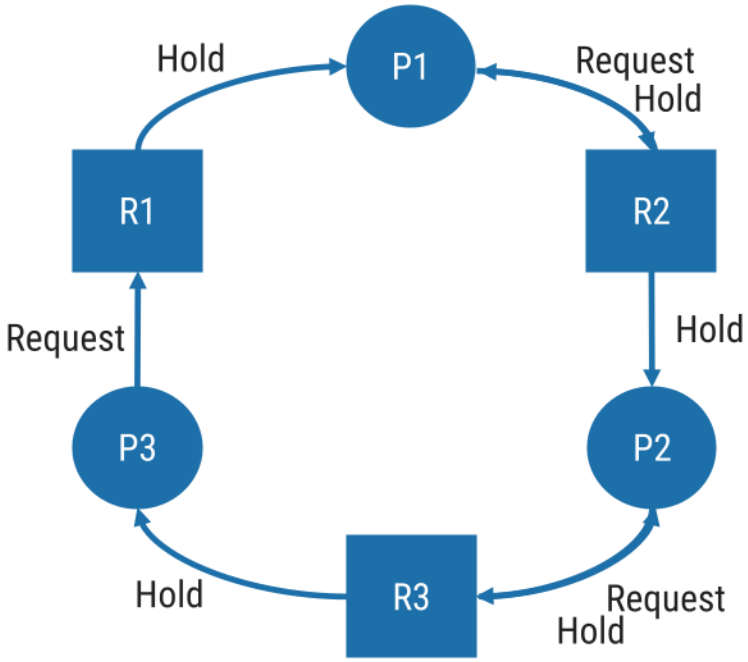


# Deadlock recovery

3. Recovery through killing processes

- The simplest way to break a deadlock is **to kill one or more processes**.
- Kill all the process involved in deadlock

- Kill process one by one.
- After killing each process check for deadlock
  - If deadlock recovered then stop killing more process
  - Otherwise kill another process



# Example for Exercise

- Consider a system consisting of four resources of same type that are shared by three processes, each of which needs at most two resources. Show the system is deadlock free.

| Process | Has / Hold | Max required |
|---|---|---|
| A | 1 | 2 |
| B | 1 | 2 |
| C | 1 | 2 |
| Total : 4 & Free : 1 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 2 | 2 |
| B | 1 | 2 |
| C | 1 | 2 |
| Free : 0 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | - |
| B | 1 | 2 |
| C | 2 | |
| Free : 2 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | - |
| B | 2 | 2 |
| C | 7 | 7 |
| Free : 1 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | - |
| B | 0 | - |
| C | 1 | 2 |
| Free : 3 | | |

| Process | Has / Hold | Max required |
|---|---|---|
| A | 0 | - |
| B | 0 | - |
| C | 2 | 2 |
| Free : 2 | | |

# Practice Questions

1. What is RAG? Explain briefly.
2. What is Deadlock? List the conditions that lead to deadlock. How Deadlock can be prevented?
3. Which are the necessary conditions for Deadlock? Explain Deadlock recovery in brief.
4. Consider the snapshot of the system with Five Processes and Four types of resources A,B,C,D.
   - Currently available set of resources is (1,5,2,0).
   - Answer the following Questions using bankers algorithm.
     1. Find the content of Need Matrix.
     2. Is the System in Safe State?
     3. If request from Process P1 arrives for (0,4,2,0) can the request be granted immediately

| | Allocation | | | | | Max | | | |
|---|---|---|---|---|---|---|---|---|---|
| Process | A | B | C | D | | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 | | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 | | 1 | 7 | 5 | 0 |
| P2 | 1 | 3 | 5 | 4 | | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 3 | 2 | | 0 | 6 | 5 | 2 |
| P4 | 0 | 0 | 1 | 4 | | 0 | 6 | 5 | 6 |

# Thank You