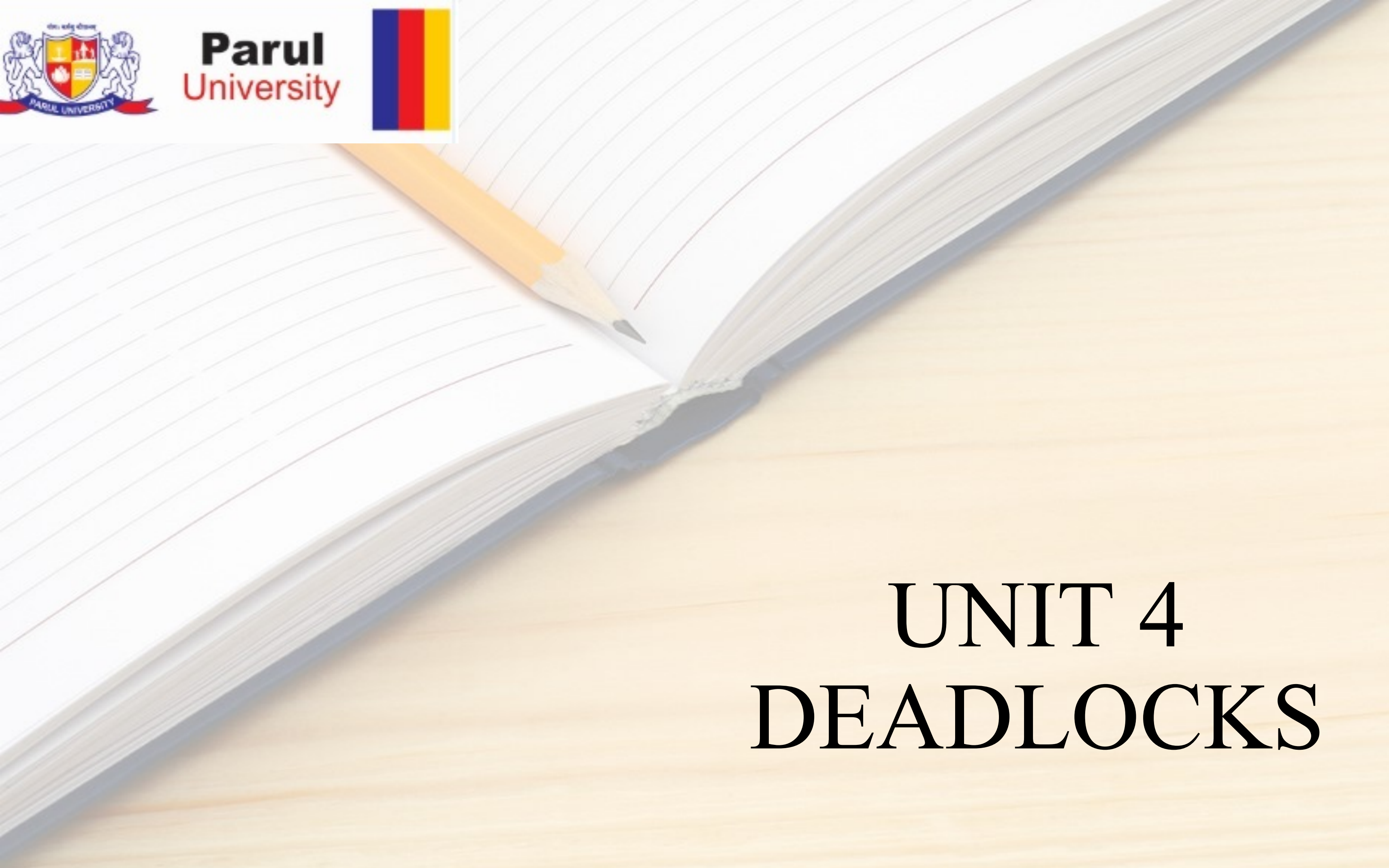
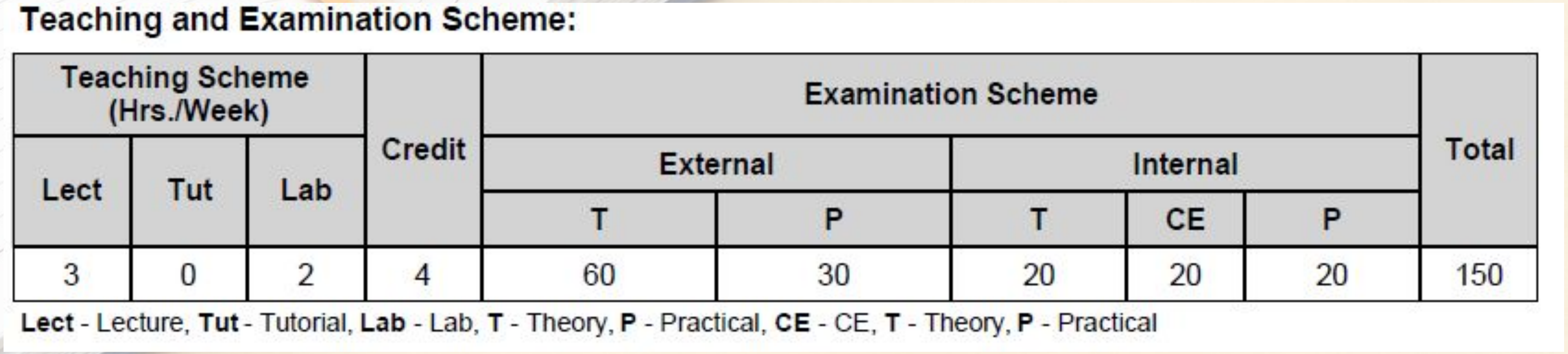


UNIT 4
DEADLOCKS

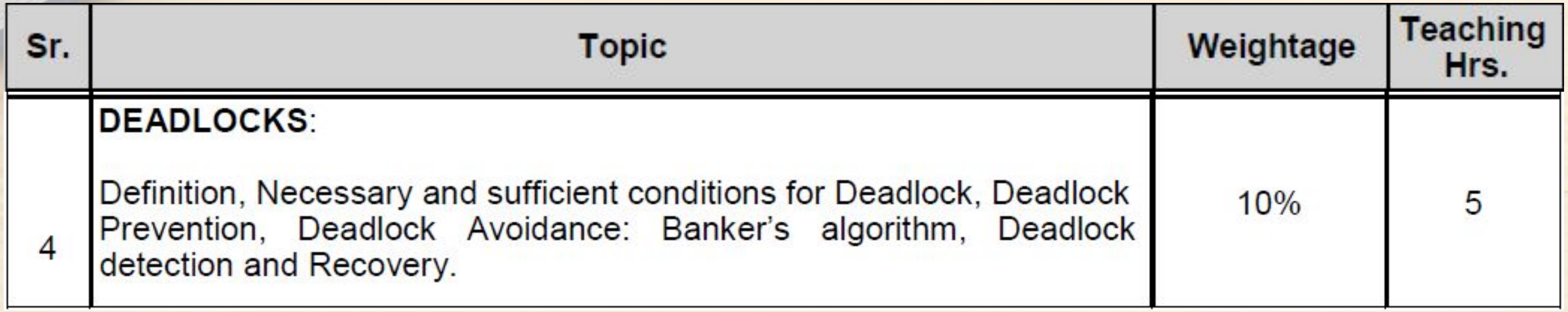
Syllabus



Teaching and Examination Scheme:

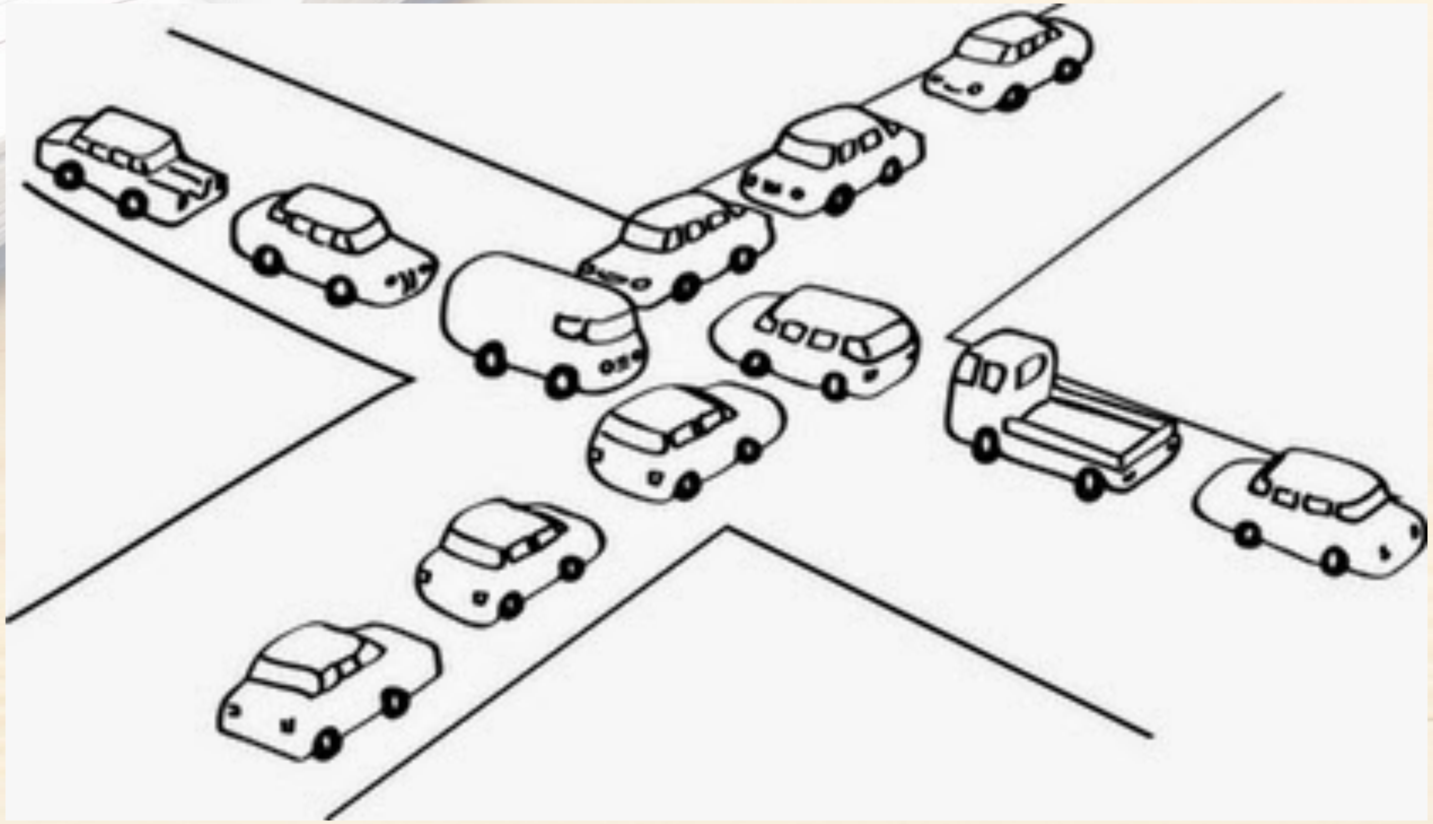
Teaching Scheme (Hrs./Week)			Credit	Examination Scheme					Total
				External		Internal			
Lect	Tut	Lab		T	P	T	CE	P	
3	0	2	4	60	30	20	20	20	150

Lect - Lecture, **Tut** - Tutorial, **Lab** - Lab, **T** - Theory, **P** - Practical, **CE** - CE, **T** - Theory, **P** - Practical



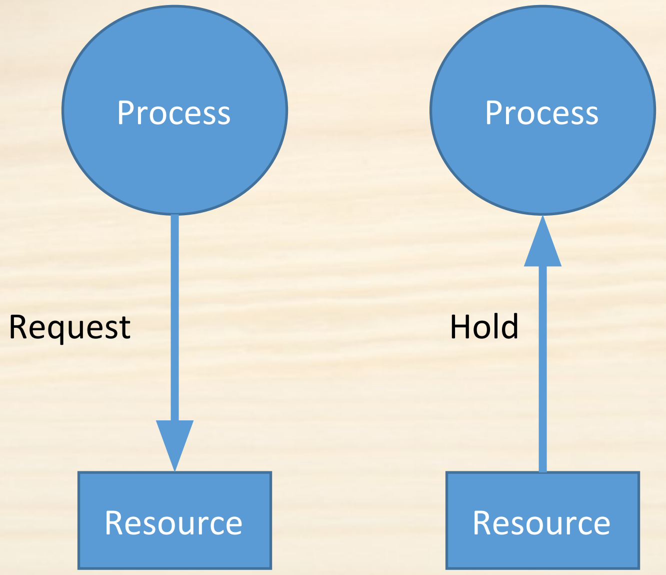
Sr.	Topic	Weightage	Teaching Hrs.
4	**DEADLOCKS**: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.	10%	5

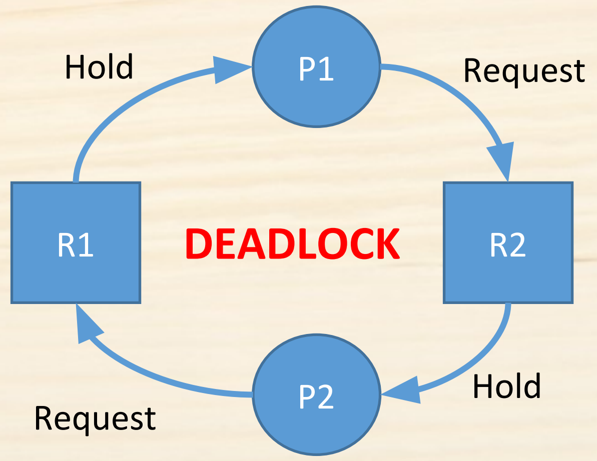
What is Deadlock?



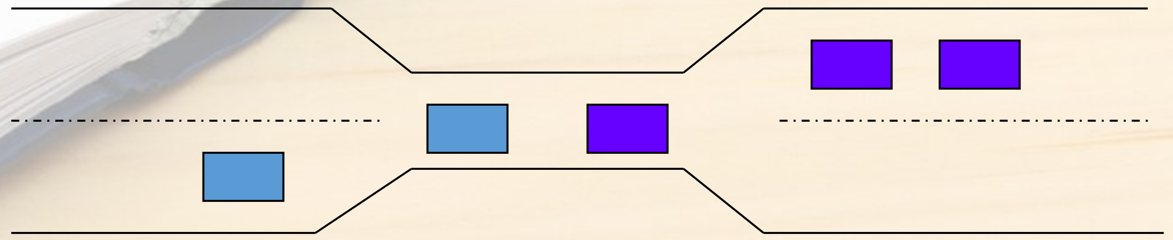
Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set is known as a **Deadlock.**
- A process is *deadlocked* if it is waiting for an event that will never occur.





Example - Bridge Crossing



- Assume traffic in one direction.
  - ❑ Each section of the bridge is viewed as a resource.
- If a deadlock occurs, it can be resolved only if one car backs up (preempt resources and rollback).
  - ❑ Several cars may have to be backed up if a deadlock occurs.
  - ❑ Starvation is possible

Types of Resources

- **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.
  - Example:- CD-ROM (CD recorder), Printer.

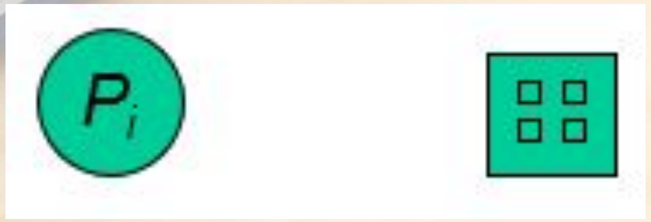
Conditions for Deadlock

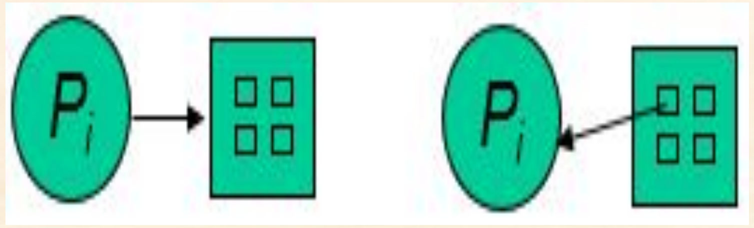
1. **Mutual exclusion**
   - Each resource is either currently assigned to exactly one process or is available.
2. **Hold and wait**
   - Process currently holding resources granted earlier can request more resources.
3. **No preemption**
   - Previously granted resources cannot be forcibly taken away 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 4 of these conditions are necessary and sufficient for deadlock (must hold simultaneously)

Resource Allocation Graph

- A visual way to determine if a deadlock has, or may occur.
- Process is a circle, Resource type is square; dots represent number of instances of resource in type.



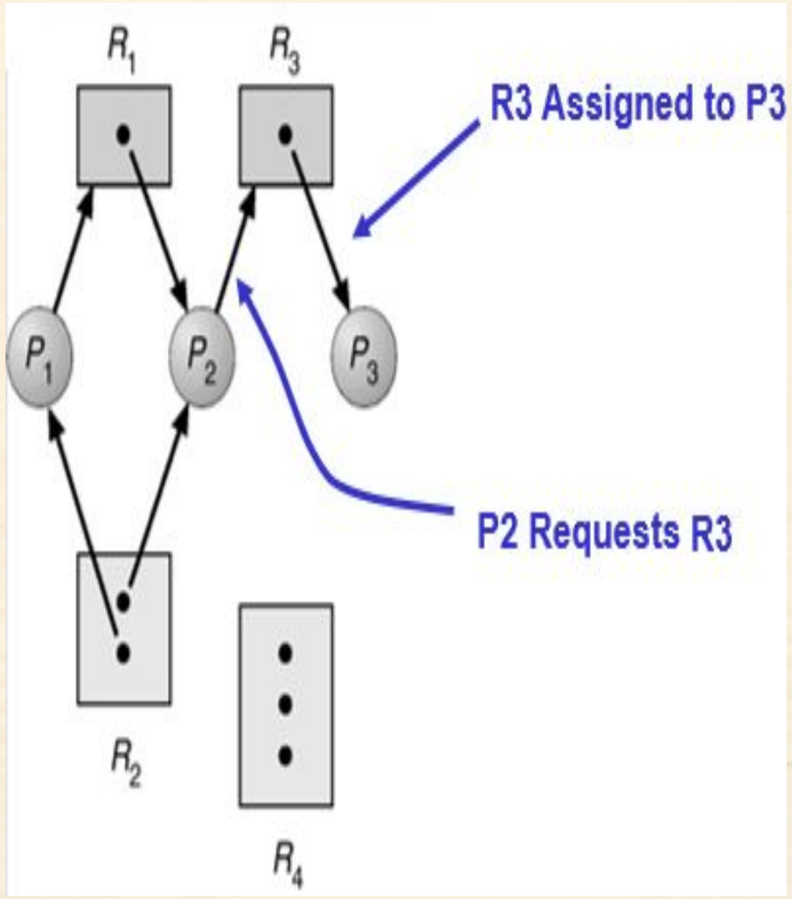


- An arrow from the **process** to **resource** indicates the process is **requesting** the resource. a An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

Resource Allocation Graph

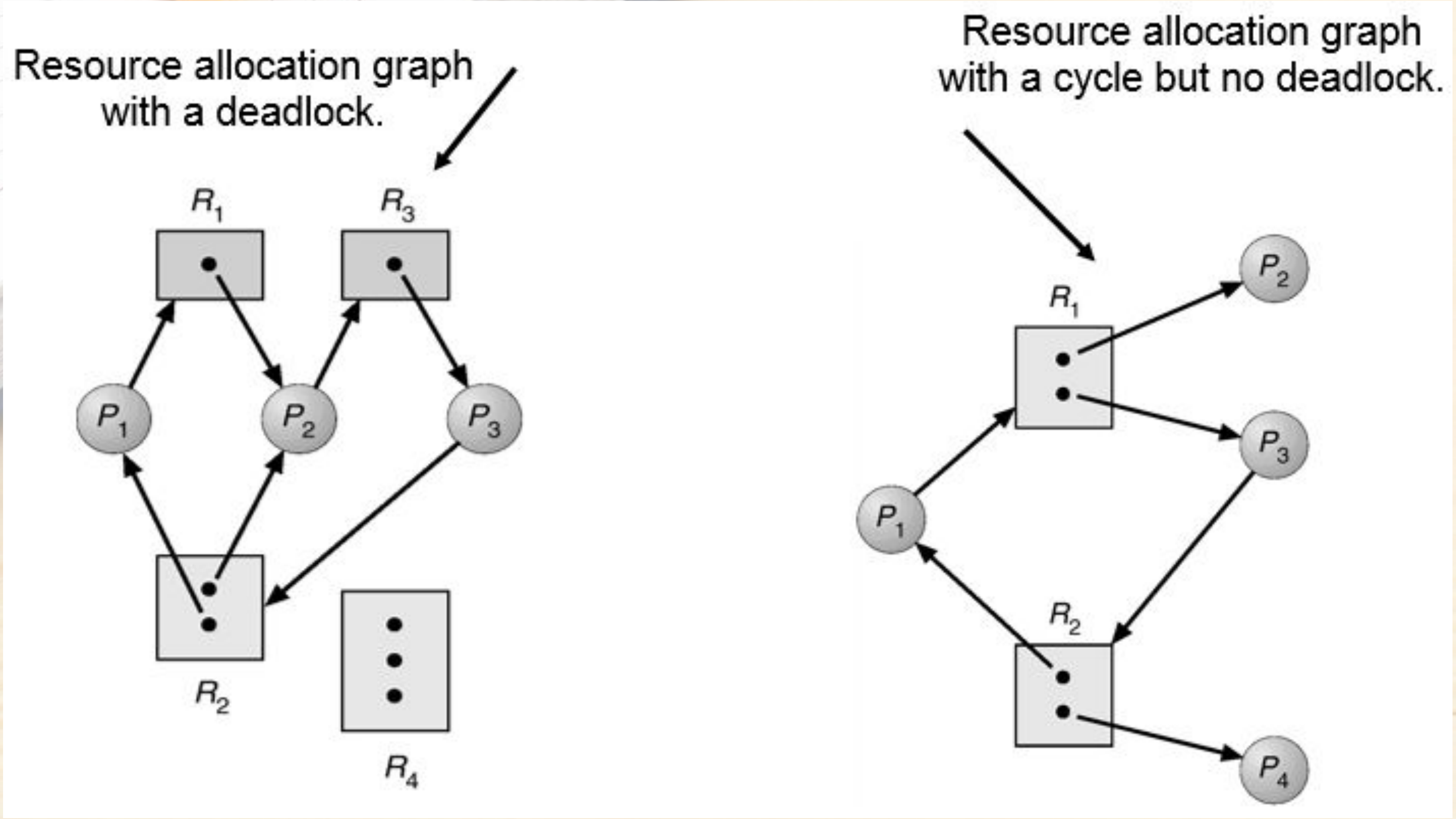
- If the graph contains no cycles, then no process is deadlocked.
- If there is a cycle, then:
  a) If resource types have multiple instances, then deadlock MAY exist.

  b) If each resource type has 1 instance, then deadlock has occurred.



Resource Allocation Graph

Resource Allocation Graph



Resource allocation graph with a deadlock.
R_1
R_3
P_1
P_2
P_3
R_2
R_4
Resource allocation graph with a cycle but no deadlock.
R_1
P_2
P_3
P_1
R_2
P_4

Methods for handling deadlocks

1. Just ignore the problem.
2. Detection and recovery.
    - Let deadlocks occur, detect them and take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating (killing) one of the four required conditions.

Deadlock Management

- **Prevention**
  - Design the system in such a way that deadlocks can never occur
- **Avoidance**
  - Impose less stringent (tight) conditions than for prevention, allowing the possibility of deadlock but sidestepping it as it occurs.
- **Detection**
  - Allow possibility of deadlock, determine if deadlock has occurred and which processes and resources are involved.
- **Recovery**
  - After detection, clear the problem, allow processes to complete and resources to be reused. May involve destroying and restarting processes.

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.
- Reasonable if
  - deadlocks occur very rarely
  - difficult to detect
  - cost of prevention is high
- UNIX and Windows takes this approach

Deadlock Prevention

- Deadlock prevention algorithms ensure that at least one of the necessary conditions (Mutual exclusion, hold and wait, no preemption and circular wait) does not hold true.

- However most prevention algorithms have poor resource utilization, and hence result in reduced throughputs.

❖ **Mutual Exclusion:**

- 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.

❖ Hold and Wait

1. A process can get all required resources before it start execution. This will avoid deadlock, but will result in reduced throughputs as resources are held by processes even when they are not needed. They could have been used by other processes during this time.

2. Second approach is to request for a resource only when it is not holding any other resource. This may result in a starvation as all required resources might not be available freely always.

❖ No preemption

- If a process that is holding some resources requests another resource that cannot be immediately allocated to it, the process releases the resources currently being held.
- Pre-empted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources as well as the new ones that it is requesting.

❖ Circular Wait

- To avoid circular wait, resources may be ordered and we can ensure that each process can request resources only in an increasing order of these numbers. The algorithm may itself increase complexity and may also lead to poor resource utilization.

Deadlock Avoidance

- If we have prior knowledge of how resources will be requested, it's possible to determine if we are entering an "unsafe" state.
  - Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
  - The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
  - Resource allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

❖ **Safe State:**

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence <P1, P2, …Pn> is safe, if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by Pj with j<i.

Cont.

Possible states are:

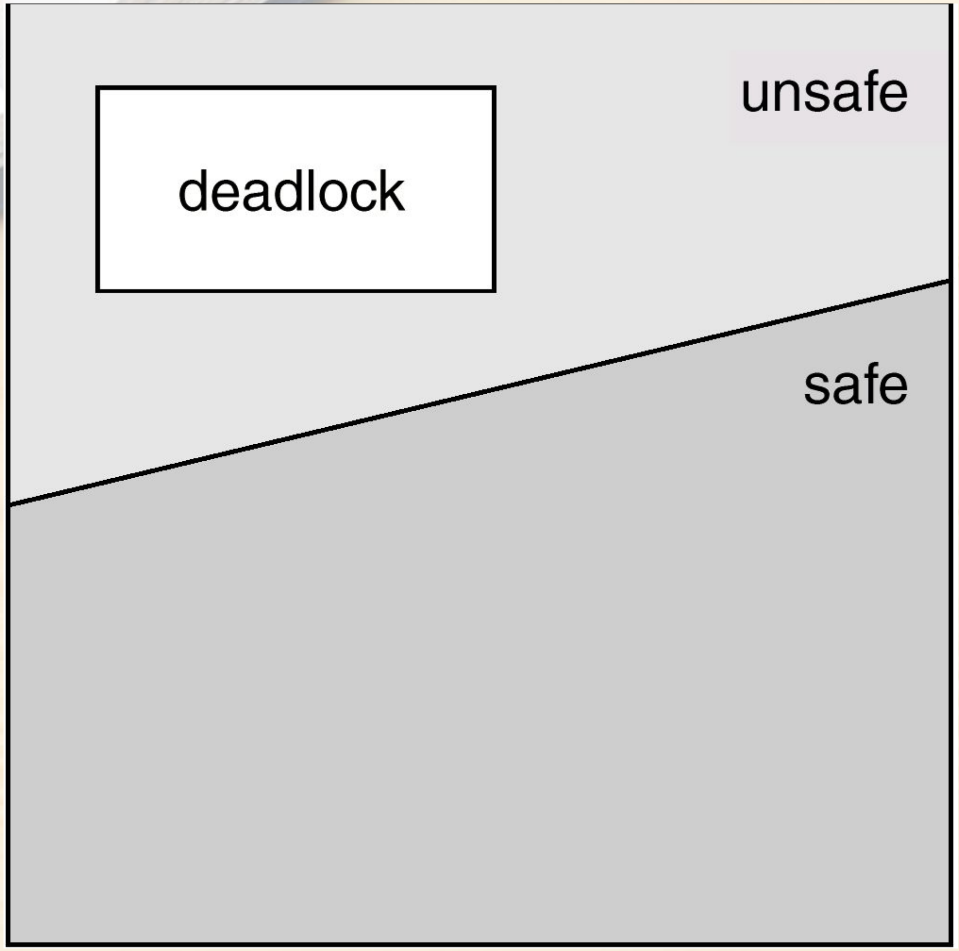
Deadlock No forward progress can be made.

Unsafe state ***Possibility*** of deadlock.

Safe state A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to finish.

The rule is simple: **If a request allocation would cause an unsafe state, do not honor that request.**

NOTE: All deadlocks are unsafe, but all unsafe are NOT deadlocks.


unsafe
deadlock
safe

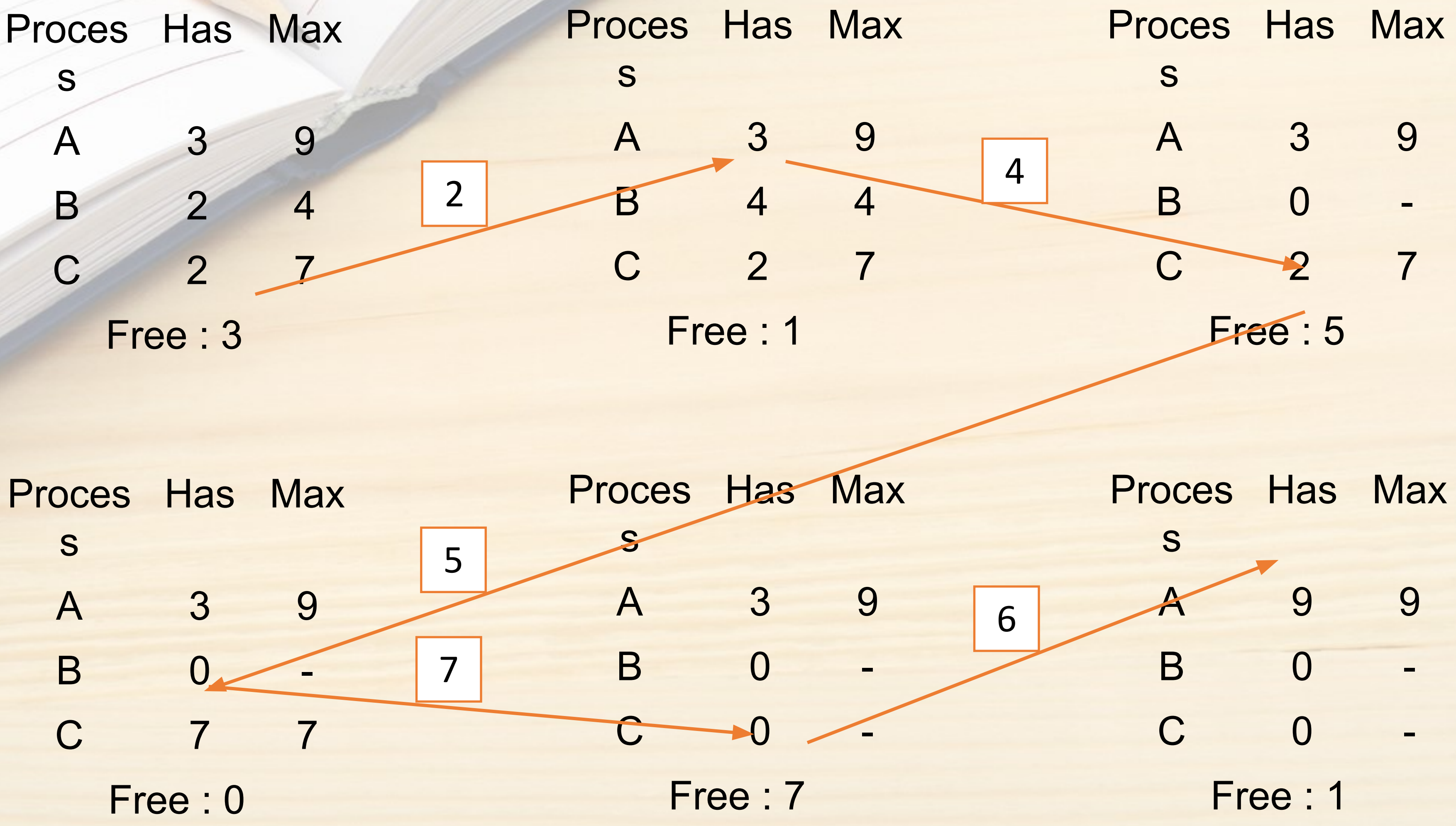
Safe and unsafe states

- Total resources are 10
- 7 resources already allocated
- So there are 3 still free
- A need 6 resources more to complete it.
- B need 2 resources more to complete it.
- C need 5 resources more to complete it.

Process	Has	Max
A	3	9
B	2	4
C	2	7

Free : 3

Safe states



Process	Has	Max
A	3	9
B	2	4
C	2	7

Free : 3

Process	Has	Max
A	3	9
B	4	4
C	2	7

Free : 1

Process	Has	Max
A	3	9
B	0	-
C	2	7

Free : 5

Process	Has	Max
A	3	9
B	0	-
C	7	7

Free : 0

Process	Has	Max
A	3	9
B	0	-
C	0	-

Free : 7

Process	Has	Max
A	9	9
B	0	-
C	0	-

Free : 1

Unsafe states

Process	Has	Max
A	3	9
B	2	4
C	2	7

Free : 3

$\boxed{1}$

Process	Has	Max
A	4	9
B	2	4
C	2	7

Free : 2

$\boxed{2}$

Process	Has	Max
A	4	9
B	4	4
C	2	7

Free : 0

$\boxed{4}$

Process	Has	Max
A	4	9
B	0	-
C	2	7

Free : 4

Banker’s Algorithm

- Used for multiple instances of each resource type.
- Each process must a priori claim maximum use of each resource type.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

- Let n = number of processes and m = number of resource types.
  - *Available*: Vector of length m. If *Available*[j] = k, there are k instances of resource type Rj available.
  - *Max*: $n \times m$ matrix. If *Max*[i,j] = k, then process Pi may request at most k instances of resource type Rj.
  - *Allocation*: $n \times m$ matrix. If *Allocation*[i,j] = k, then process Pi is currently allocated k instances of resource type Rj.
  - *Need*: $n \times m$ matrix. If *Need*[i,j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

$$\mathit{Need}[i,j] = \mathit{Max}[i,j] - \mathit{Allocation}[i,j]$$

Safety Algorithm

Step 1:Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize

- *Work* := *Available*
- *Finish*[*i*] := *false* for *i* = 1,2,…,n.

Step 2:Find an *i* (i.e. process *Pi*) such that both:

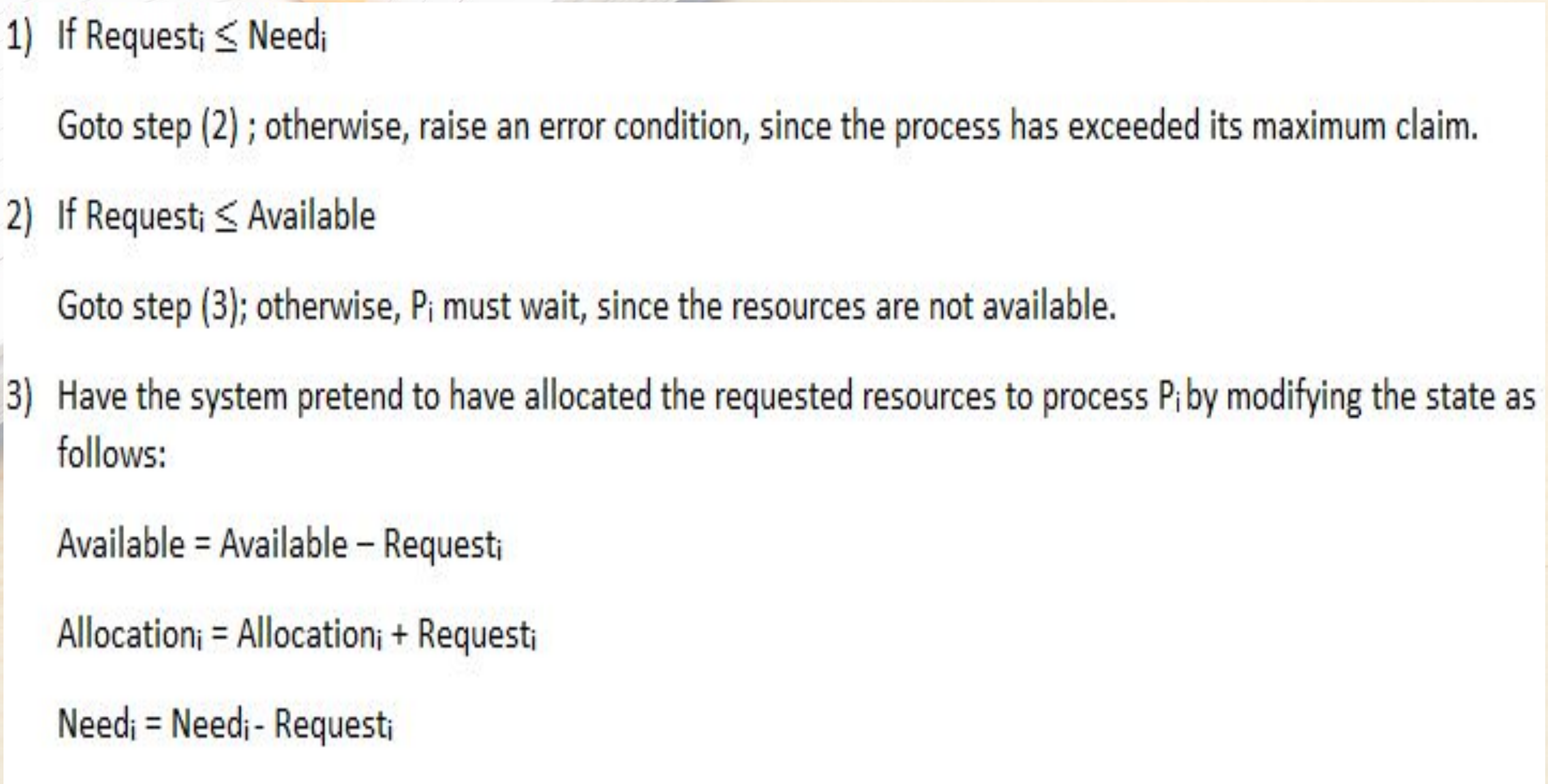
- *Finish*[*i*] = *false*
- *Need_i* <= *Work*
- If no such *i* exists, go to step 4.

Step 3: Work := *Work* + Allocation_*i*

- *Finish*[*i*] := *true*
- go to step 2

Step 4: If *Finish*[*i*] = *true* for all *i*, then the system is in a safe state.

Resource-Request Algorithm



1) If $Request_i \leq Need_i$

   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If $Request_i \leq Available$

   Goto step (3); otherwise, P_i must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows:

   $Available = Available – Request_i$

   $Allocation_i = Allocation_i + Request_i$

   $Need_i = Need_i - Request_i$

Banker's algorithm for single resource

- 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	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free : 10

Banker's algorithm for single resource

Process	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free : 2

Process	Has	Max
A	1	6
B	1	5
C	4	4
D	4	7

Free : 0

Process	Has	Max
A	1	6
B	1	5
~~C~~	~~0~~	~~0~~
D	4	7

Free : 4

Process	Has	Max
A	1	6
B	1	5
~~C~~	~~0~~	-
D	7	7

Free : 1

Process	Has	Max
A	1	6
B	1	5
~~C~~	~~0~~	-
~~D~~	~~0~~	-

Free : 8

Process	Has	Max
A	1	6
B	5	5
~~C~~	~~0~~	-
~~D~~	~~0~~	-

Free : 4

Banker's algorithm for single resource

Process	Has	Max
A	1	6
~~B~~	~~0~~	-
~~C~~	~~0~~	-
~~D~~	~~0~~	-

Free : 9

Process	Has	Max
A	6	6
~~B~~	~~0~~	-
~~C~~	~~0~~	-
~~D~~	~~0~~	-

Free : 4

Process	Has	Max
~~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

Proces s	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free : 1

Banker's algorithm for multiple resource

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
1	0	2	0

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
1	0	1	0

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
2	1	2	1

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
1	0	2	1

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
5	1	3	2

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
5	0	2	0

Available (free) resources

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

no of resources held by each process

Process	Tape Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
5	2	3	2

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
2	1	3	2

Available (free) resources

Process	Tape Drive	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 Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

Available (free) resources

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

no of resources held by each process

Process	Tape Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
4	2	3	2

Available (free) resources

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

no of resources held by each process

Process	Tape Drive	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

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

Available (free) resources

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	-	-	-	-
P2	-	-	-	-
P3	-	-	-	-
P4	-	-	-	-
P5	-	-	-	-

ld by each

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

no of resources still needed by each process to proceed

Banker's algorithm for multiple resource

Tape Drive	Plotters	Scanners	CD Roms
6	3	4	2

total no of each resource

Tape Drive	Plotters	Scanners	CD Roms
5	3	2	2

resources hold

Tape Drive	Plotters	Scanners	CD Roms
1	0	2	0

Available (free) resources

Process	Tape Drive	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

ld by each

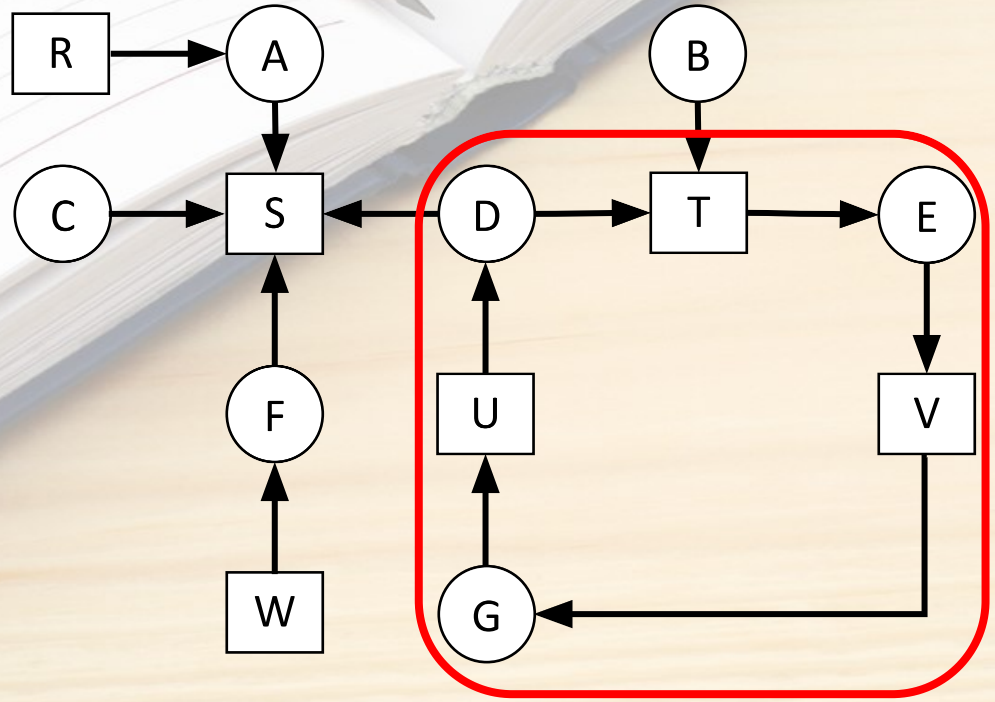
Process	Tape Drive	Plotters	Scanners	CD Roms
P1	1	1	0	0
	0	1	1	2
	3	1	0	0
	0	0	1	1
	2	1	1	0

no of resources still needed by each process to proceed

Deadlock Detection

- If deadlocks are not avoided, then another approach is to detect when they have occurred and recover somehow.
- In addition to the performance hit of constantly checking for deadlocks, a policy / algorithm must be in place for recovering from deadlocks.

Deadlock detection for single resource



- 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 empty list the node D appears twice, so deadlock.

Deadlock detection for single resource

- 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.
  - An algorithm to detect a cycle in a graph requires an order of *n*^2 operations, where *n* is the number of vertices in the graph.

Several instances of a resource type

- Data Structures
  - ***Available***: Vector of length m. If *Available*[j] = k, there are k instances of resource type R_j available.
  - ***Allocation***: $n \times m$ matrix. If *Allocation*[i,j] = k, then process P_i is currently allocated k instances of resource type R_j.
  - ***Request***: An $n \times m$ matrix indicates the current request of each process. If *Request* [i,j] = k, then process P_i is requesting k more instances of resource type R_j .

Deadlock Detection Algorithm

- Step 1: Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize
  - *Work* := *Available*
  - For $i = 1,2,\ldots,n$, if $Allocation(i) \neq 0$, then $Finish[i] := false$, otherwise $Finish[i] := true$.
- Step 2: Find an index i such that both:
  - $Finish[i] = false$
  - $Request\ (i) \leq Work$
  - If no such i exists, go to step 4.

Deadlock Detection Algorithm

- Step 3: *Work* := *Work* + Allocation(*i*)
  - *Finish*[*i*] := *true*
  - go to step 2
- Step 4: If *Finish*[*i*] = *false* for some *i*, $1 \le i \le n$, then the system is in a deadlock state. Moreover, if *Finish*[*i*] = *false*, then *Pi* is deadlocked.

Algorithm requires an order of m × (n^2) operations to detect whether the system is in a deadlocked state.

Deadlock detection for multiple resource

E =

Tape Drive	Plotters	Scanners	CD Roms
4	2	3	1

total no of each resource

A =

Tape Drive	Plotters	Scanners	CD Roms
2	1	0	0

no of resources that are available (free)

C =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

no of resources held by each process

R =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	2	0	0	1
P2	1	0	1	1
P3	2	1	0	0

no of resources still needed by each process to proceed

Deadlock detection for multiple resource

E =

Tape Drive	Plotters	Scanners	CD Roms
4	2	3	1

total no of each resource

A =

Tape Drive	Plotters	Scanners	CD Roms
0	0	0	0

no of resources that are available (free)

C =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	0	0	1	0
P2	2	0	0	1
P3	2	2	2	0

no of resources held by each process

R =

Process	Tape Drive	Plotters	Scanners	CD Roms
P1	2	0	0	1
P2	1	0	1	1
P3	0	0	0	0

no of resources still needed by each process to proceed

Deadlock detection for multiple resource

E =

Tape Drive	Plotters	Scanners	CD Roms
4	2	3	1

total no of each resource

A =

Tape Drive	Plotters	Scanners	CD Roms
2	2	2	0

no of resources that are available (free)

C =

Process	Tape Drive	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 Drive	Plotters	Scanners	CD Roms
P1	2	0	0	1
P2	1	0	1	1
~~P3~~	~~0~~	~~0~~	~~0~~	~~0~~

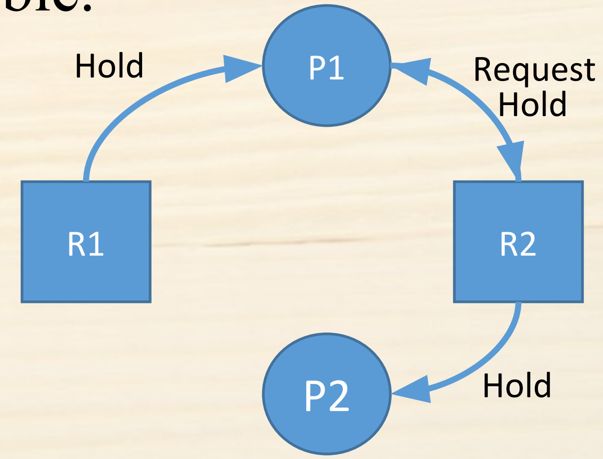
no of resources still needed by each process to proceed

Detection-Algorithm Use

- When, and how often to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - ❑ One for each disjoint cycle
- How often --
  - Every time a request for allocation cannot be granted immediately
    - ❑ Allows us to detect set of deadlocked processes and process that “caused” deadlock. Extra overhead.
    - ❑ Every hour or whenever CPU utilization drops.
  - With arbitrary invocation there may be many cycles in the resource graph and we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

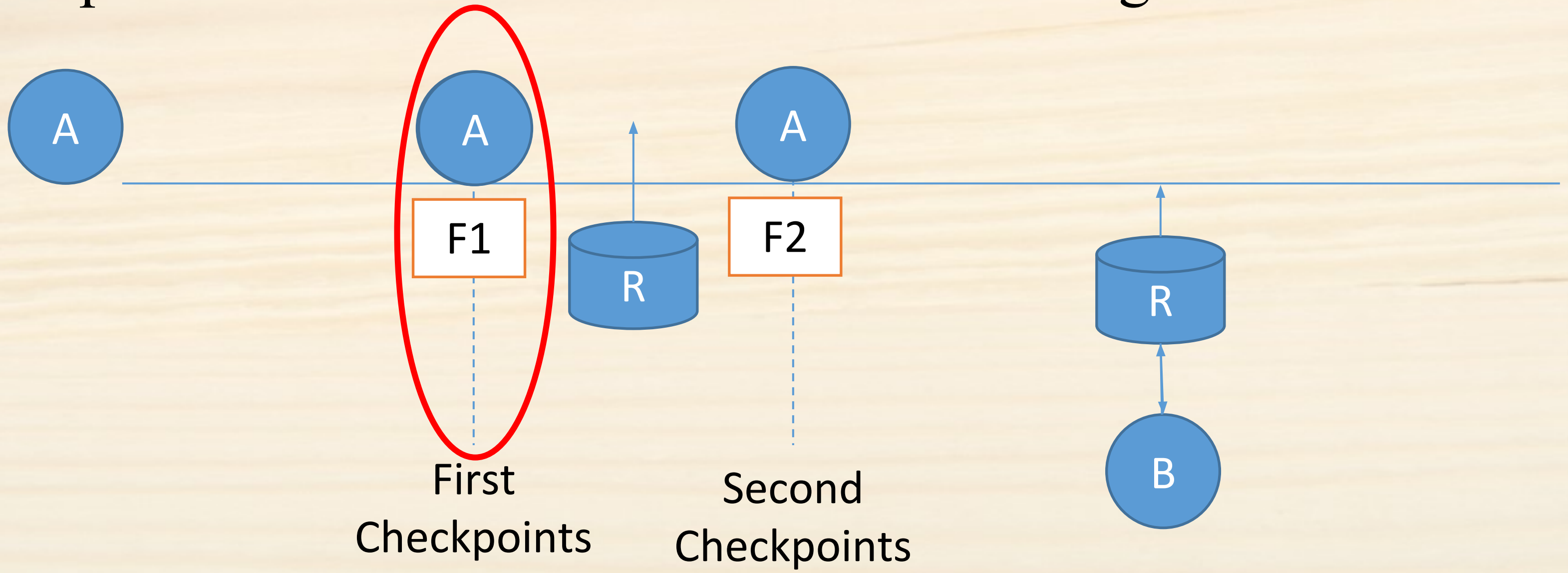
Deadlock recovery

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.



Deadlock recovery (cont…)

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 (cont…)

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.
       - o After killing each process check for deadlock
         - • If deadlock recovered then stop killing more process
         - • Otherwise kill another process

ABHIJITSINH PARMAR