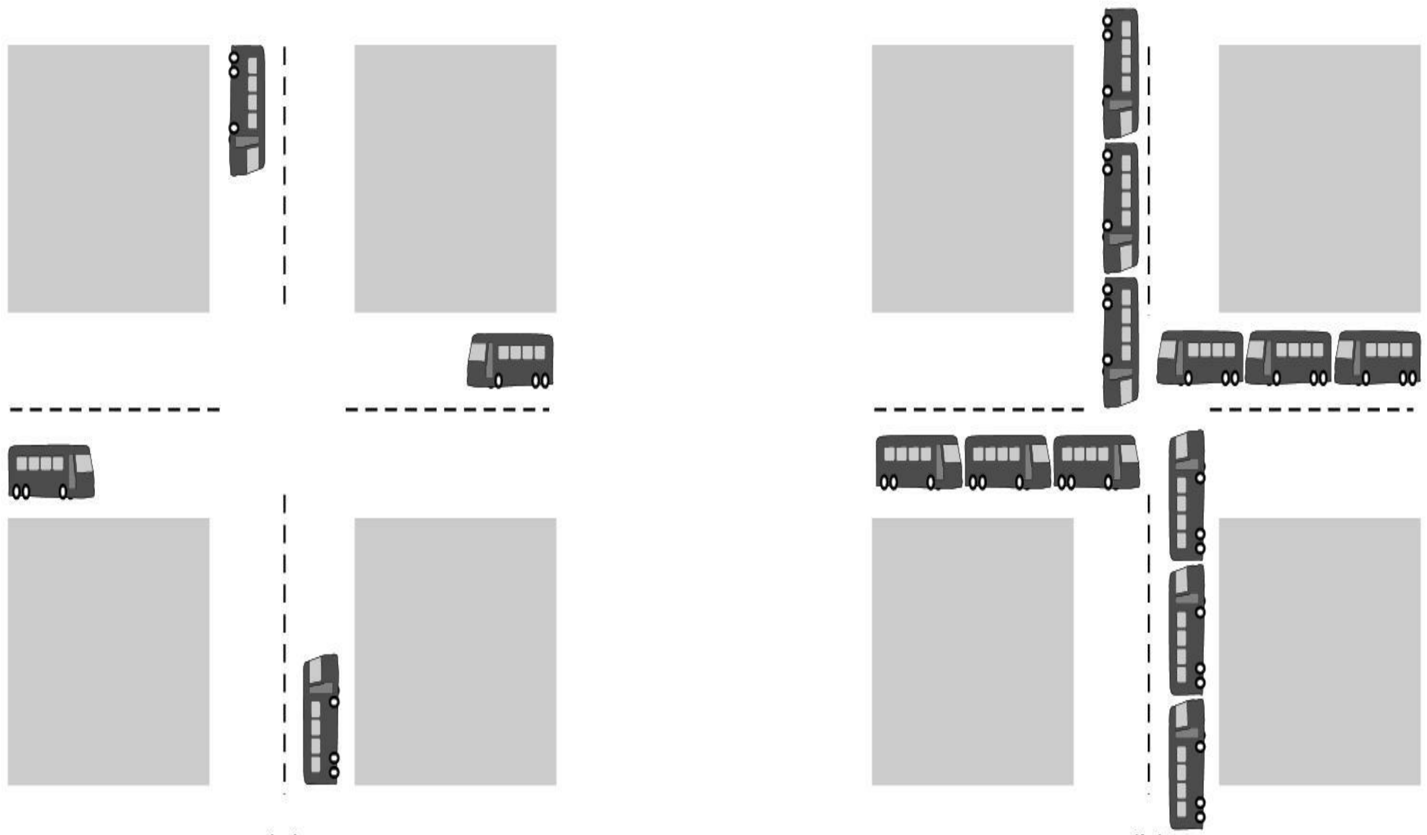
Process Deadlock

Unit 3

Deadlock


(a)
(b)

Resources

- Resources are the passive entities needed by processes to do their work.
- A resource can be:
    - Hardware device (e.g. A tape drive)
    - Piece of information(e.g. A locked record in database)
- Example of resources include CPU time, disk space, memory etc.

Resources

There are two types of resources:

(a) Preemptable resource

- That can be taken away from the process owing it with no ill effects.
- Example: memory

(b) Non-preemptable resource

- That can not be taken away from its current owner without causing the computation to fail.
- Example: CD recorders

The sequence of events required to use a resource is:

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

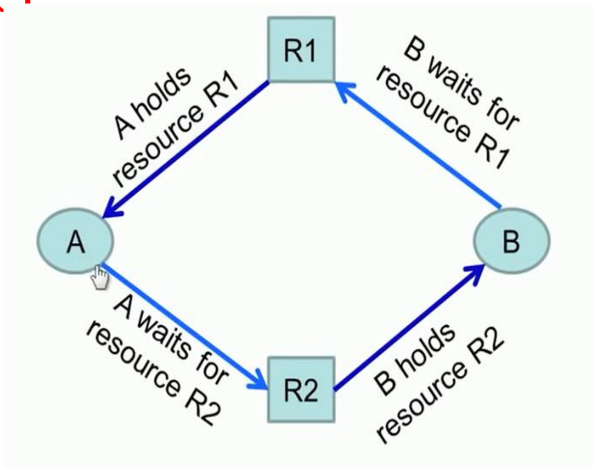
Introduction to Deadlock

- Formal definition :

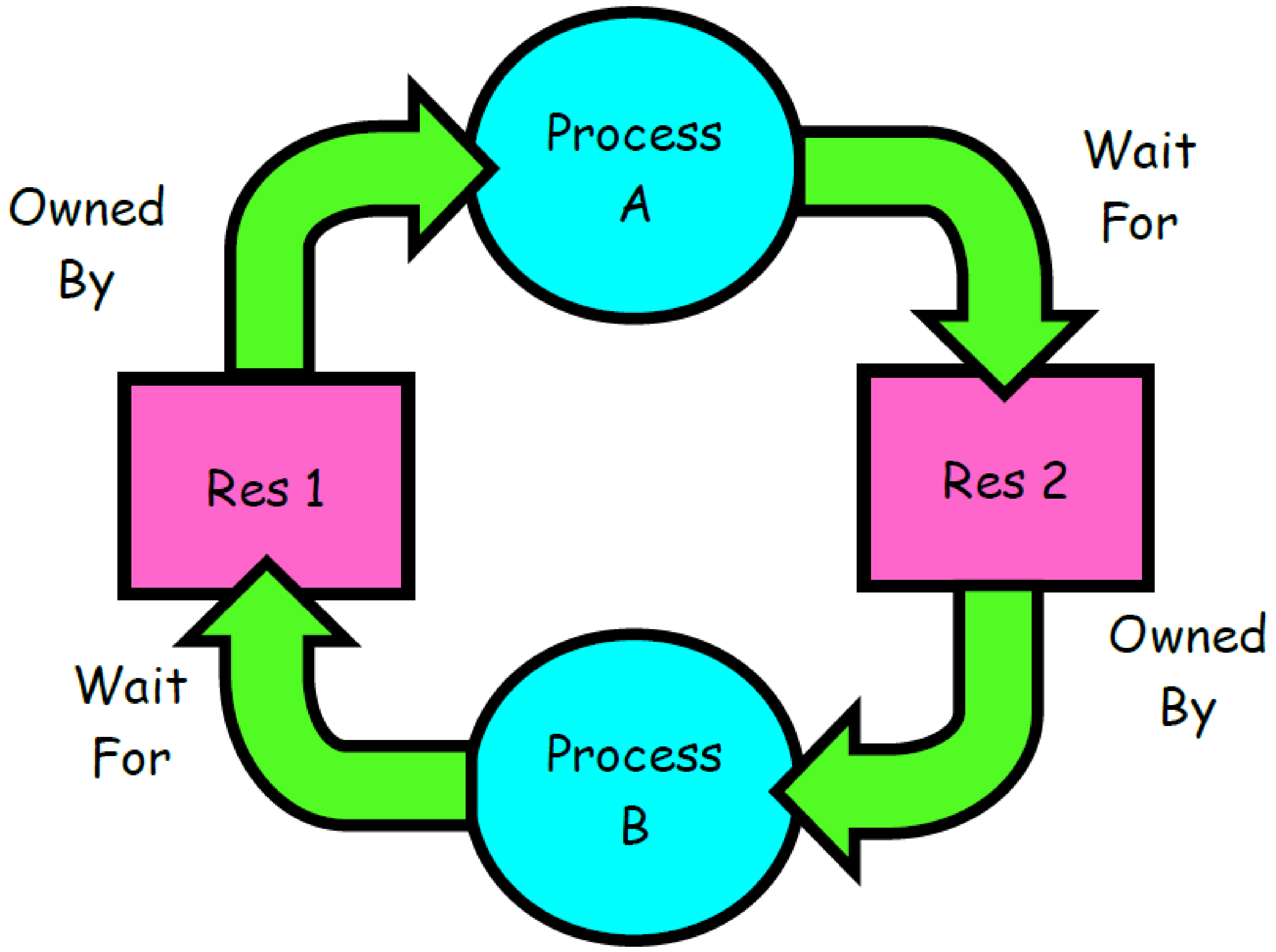
  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

- Because all the processes are waiting, none of them will ever cause any of the events that could wake up (activated) any of the other member in the set, and all the processes continue to wait forever

- That is, none of the processes can-
  - run
  - release resources
  - be awakened



Introduction to Deadlock



Example :

- Two process A and B each want to record a scanned document on a CD.
- A requests permission to use Scanner and is granted.
- B is programmed differently and requests the CD recorder first and is also granted.
- Now, A ask for the CD recorder, but the request is denied until B releases it.
- Unfortunately, instead of releasing the CD recorder B asks for Scanner.
- At this point both processes are blocked and will remain so forever. This situation is called Deadlock.

Starvation vs. Deadlock

Starvation: thread waits indefinitely

- Example, low-priority thread waiting for resources constantly in use by high-priority threads

Deadlock: circular waiting for resources

- Thread A owns Res 1 and is waiting for Res 2 Thread B owns Res 2 and is waiting for Res 1
- Deadlock ==> Starvation but not vice versa
- Starvation can end.
- Deadlock can't end without external intervention

Conditions for Deadlock

1. **Mutual exclusion:** Only one process may use a resource at a time
2. **Hold and wait:** Processes hold resources already allocated to them while waiting for additional resources.
3. **No preemption condition :** Resources cannot be removed from the processes holding them until they are explicitly released by processes holding them.
4. **Circular wait condition :** A circular chain of two or more processes, each of which is waiting for a resource held by the next process in the chain.

   -Potential consequence of the first three.

NOTE: All four of these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible.

Deadlock Modeling

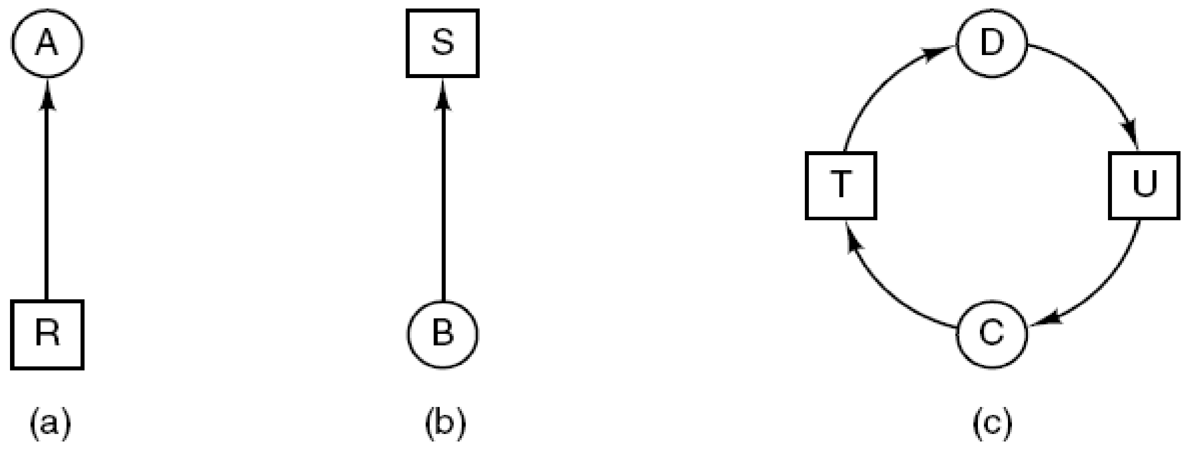


Figure: Resource allocation graphs-

(a) Holding a resource.

(b) Requesting a resource.

(c) Deadlock.

Process - Circle.

Resource - Square.

Deadlock Modeling

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

(d)

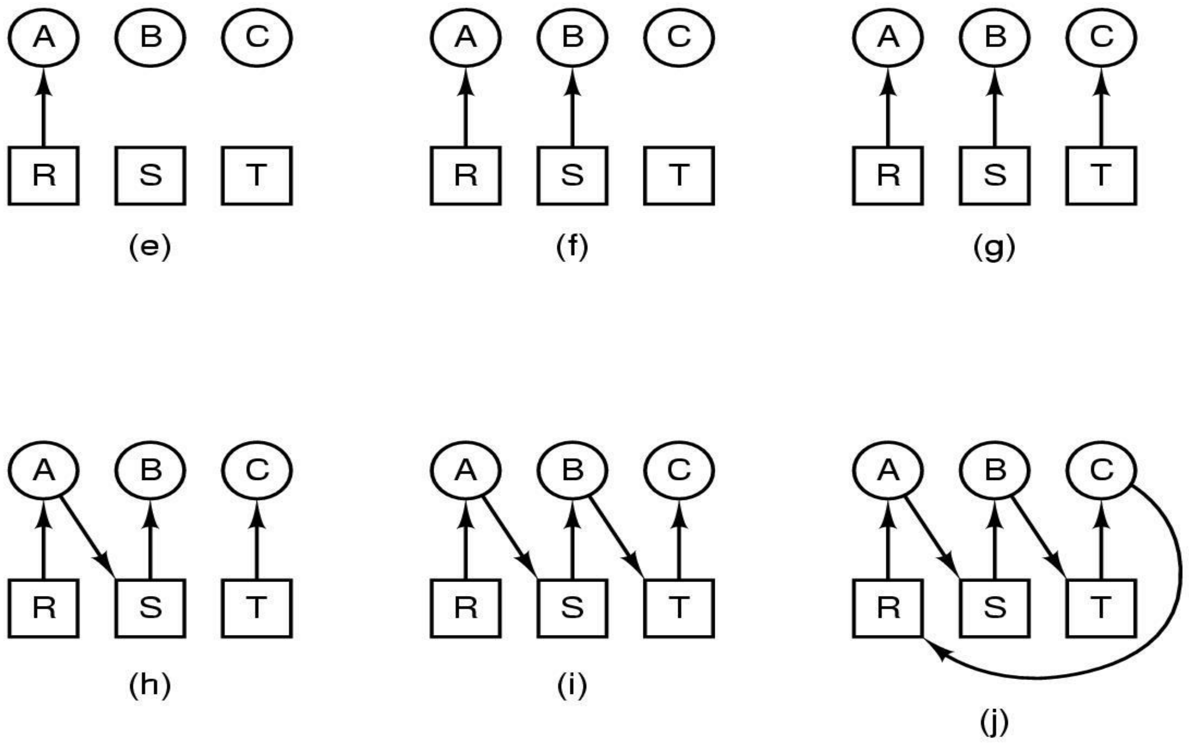


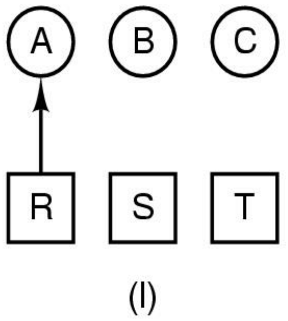
Figure : An example of how deadlock occurs

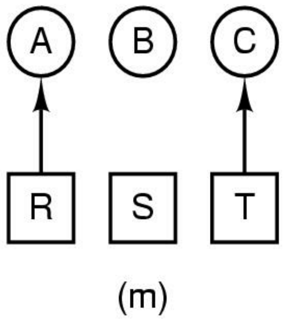
Deadlock Modeling

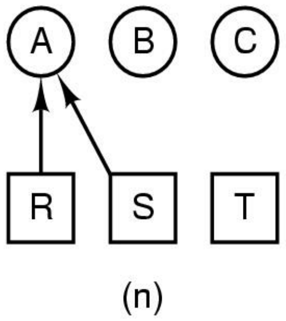
1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S

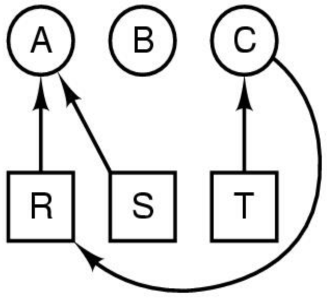
   no deadlock

(k)

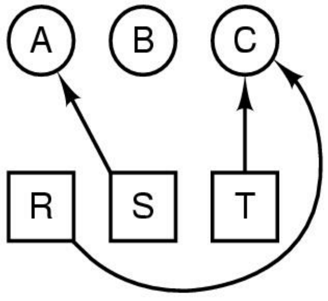




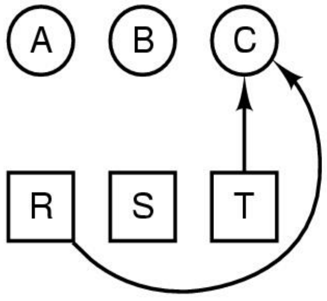




(o)



(p)

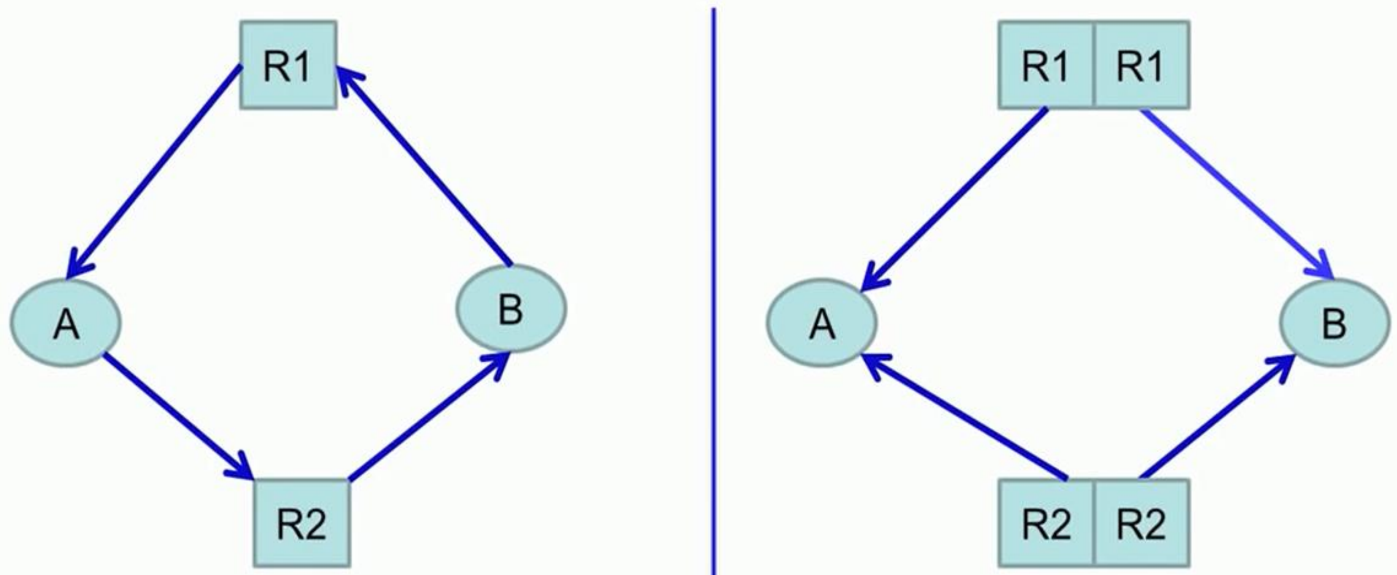


(q)

Figure: No deadlock occurence

Multiple Resources

- Having multiple resources can potentially reduce the chance of having a deadlock



A deadlocked state

No Deadlocks

Should Deadlock be Handled ?

- Preventing / detecting deadlocks could be tedious
- Can we live without detecting / preventing deadlocks?
  - What is the probability of occurrence?
  - What are the consequences of a deadlock? (How critical is a deadlock?)

Deadlock handling strategies :

1. Just Ignore the problem altogether.
2. Detection and recovery. Let deadlocks occur, detect them, and take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four conditions necessary to cause a deadlock.

Deadlock Handling

- Deadlock prevention
- Deadlock avoidance
- Deadlock detection
- Recovery from deadlock

The Ostrich Algorithm (Ignore deadlock)

- "Stick your head in the sand and pretend there is no problem".
- Mathematicians find it totally unacceptable and say that deadlocks must be prevented.
- Engineers ask how often the problem is expected, how often the system crashes for other reasons, and how serious a deadlock is.

The Ostrich Algorithm (Ignore deadlock)

- Most operating systems, including UNIX, MINIX 3, and Windows, just ignore the problem on the assumption that most users would have an occasional deadlock to a rule restricting all users to one process, one open file, and one of everything.
- If deadlocks could be eliminated for free, there would not be much discussion.
- The problem is that the price is high, mostly in terms of putting inconvenient restrictions on processes.
- Thus we are faced with an unpleasant trade-off between convenience and correctness, and a great deal of discussion about which is more important, and to whom.
- Under these conditions, general solutions are hard to find.

Deadlock Prevention

Preventing one of the following condition which leads to deadlock.

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

1. Denying the Mutual Exclusion Condition

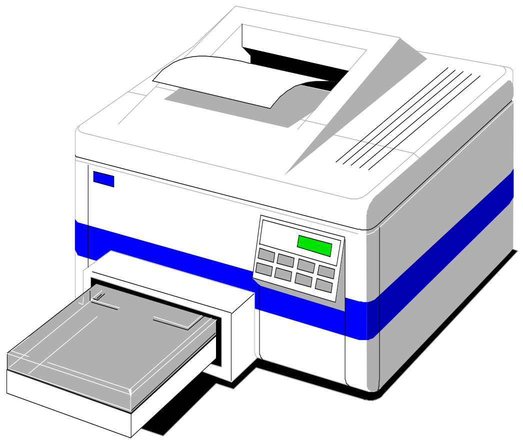
- If no resources were ever assigned exclusively to a single process, we would never have deadlock.
- However, it is equally clear that allowing two processes to write on the printer at the same time will lead to chaos.
- Principle:
  - avoid assigning resource when not absolutely necessary
  - as few processes as possible actually claim the resource

2. Denying the Hold and Wait Condition

- If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlock.
- Require processes to request resources before starting of execution.
  - A process never has to wait for what it needs.
- **Problems**
  - May not know required resources at start of run.
  - Also ties up resources other processes could be using.
  - Resources will not be used optimally with this approach.
- **Variation:**
  - Process must give up all resources whatever it is holding.
  - Then request all immediately needed.

3. Denying the No Preemption Condition

- This is not a viable option.
- Consider a process given the printer
  - halfway through its job, now forcibly take away printer
  - Then !!??



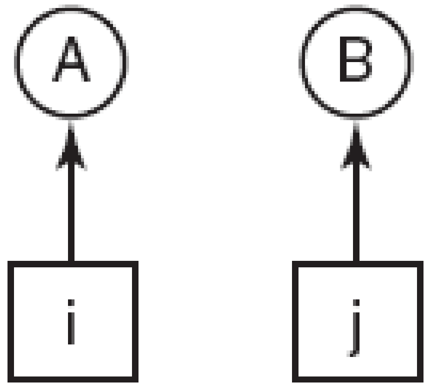
4. Denying the Circular Wait Condition

- <u>One way to achieve this:</u>
  - Process is entitled only to a single resource at any moment.
  - If it needs a second one, it must release the first one.
  - Always acceptable????
- <u>Next way:</u>
  - Provide a global numbering of all the resources.
  - Requests must be made in numerical order.
  - E g., a process may request a printer and then a tape drive, but not in reverse order.
  - **Figure Next page**

4. Denying the Circular Wait Condition

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

(a)

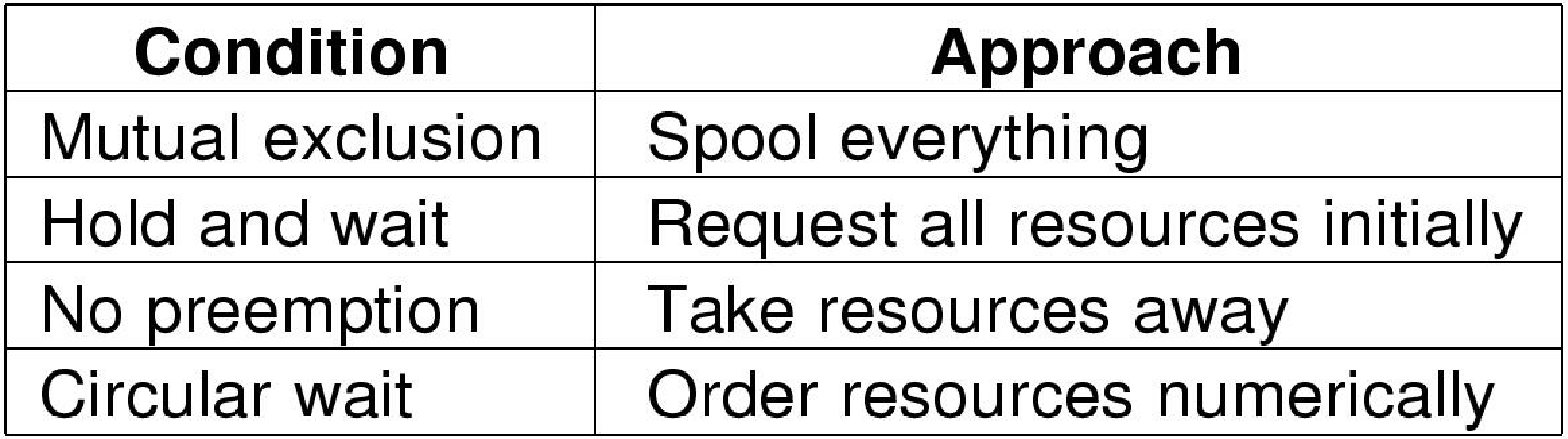


(b)

Figure : (a) Numerically ordered resources (b) A resource graph.

Deadlock occurs only if process A requests resource j and B requests resource i.

Summary of approaches to deadlock prevention:



Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

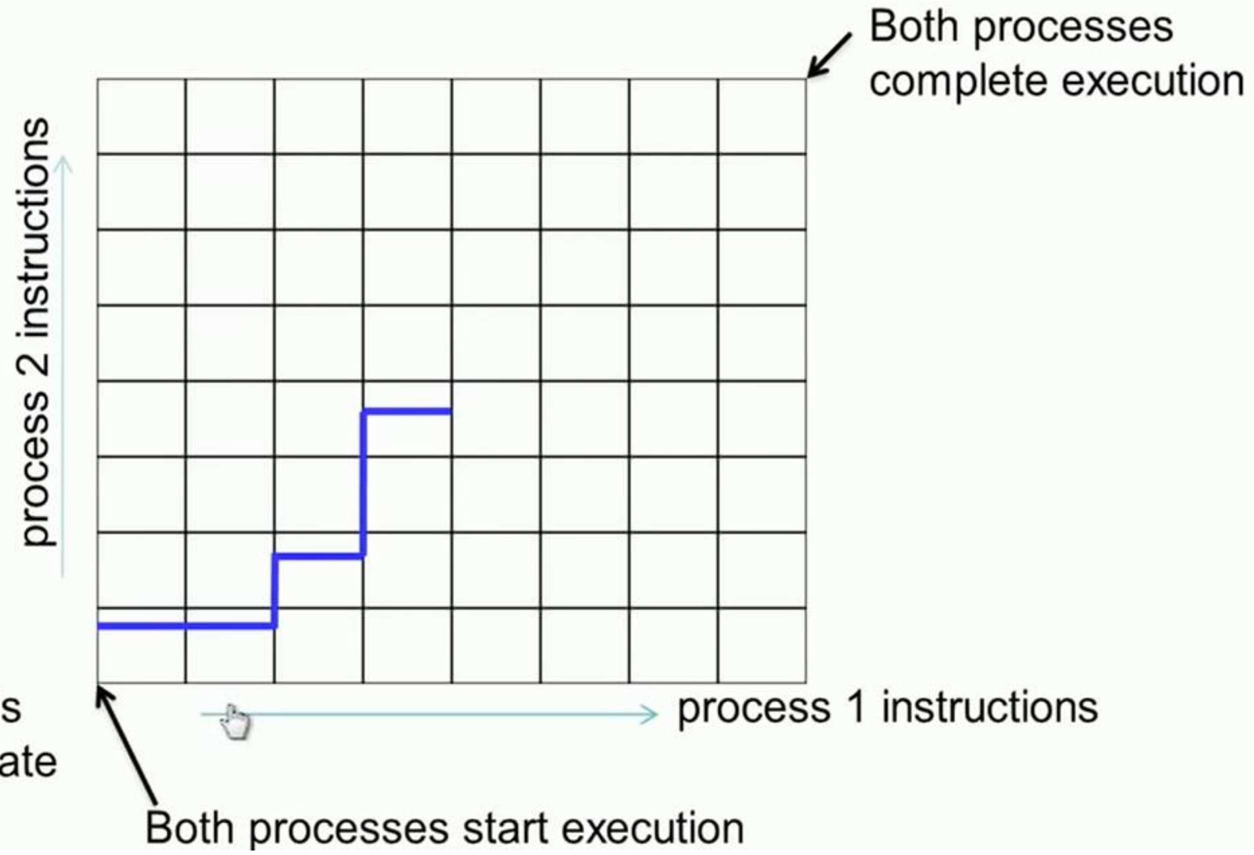
Deadlock Avoidance

- By avoidance, we mean that a system will never go into state which could potentially create a deadlock situation.
- Deadlock avoidance is achieved by being careful at the time of resources allocation.
- The system must be able to decide whether granting a resource is safe or not, and only make the allocation when it is safe.
- Let us observe following **resource trajectory**:

Deadlock Avoidance

- System decides in advance if allocating a resource to a process will lead to a deadlock

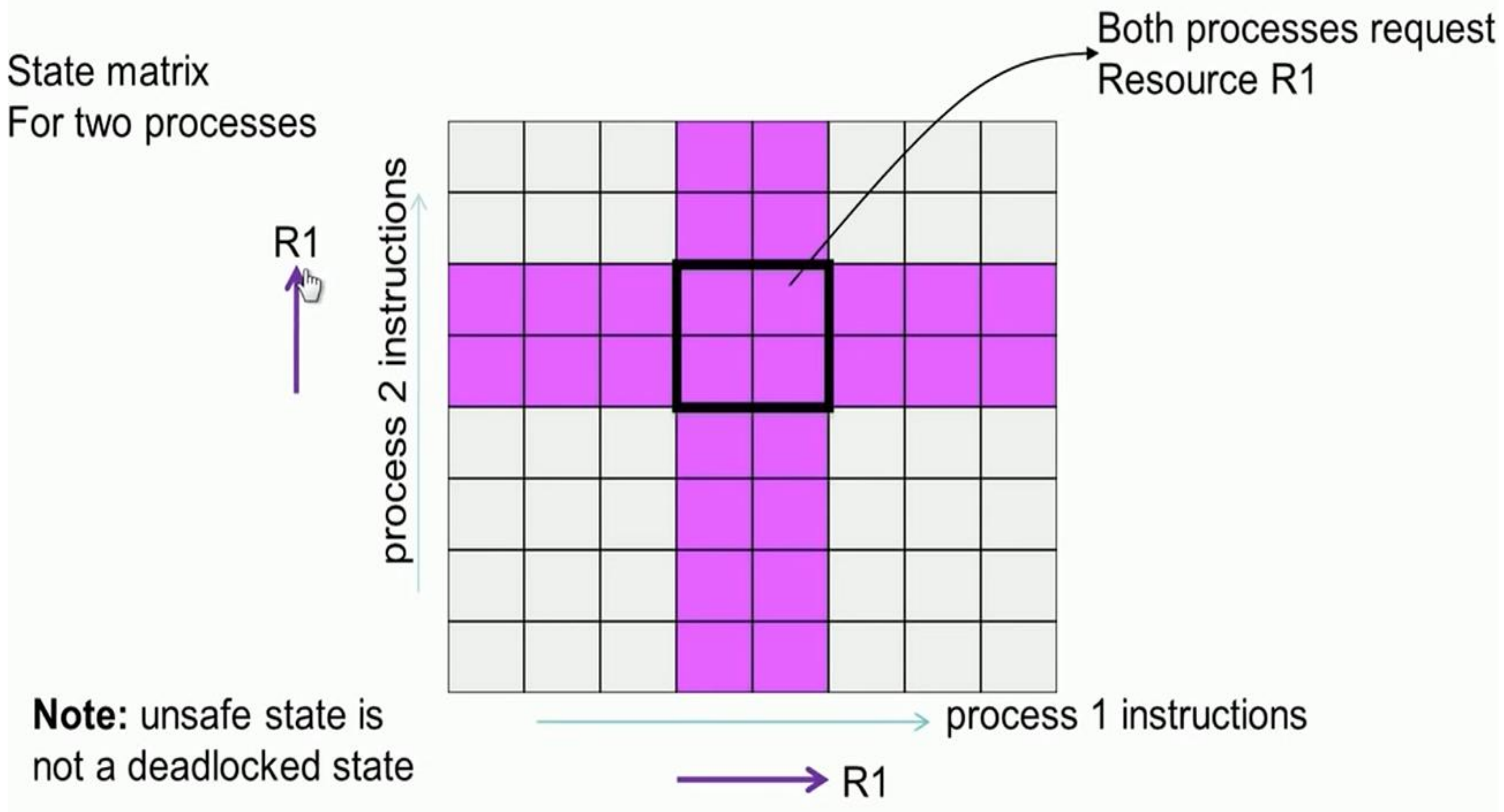
State matrix
For two processes


Both processes complete execution
process 2 instructions
process 1 instructions
Both processes start execution

Note: unsafe state is not a deadlocked state

Deadlock Avoidance

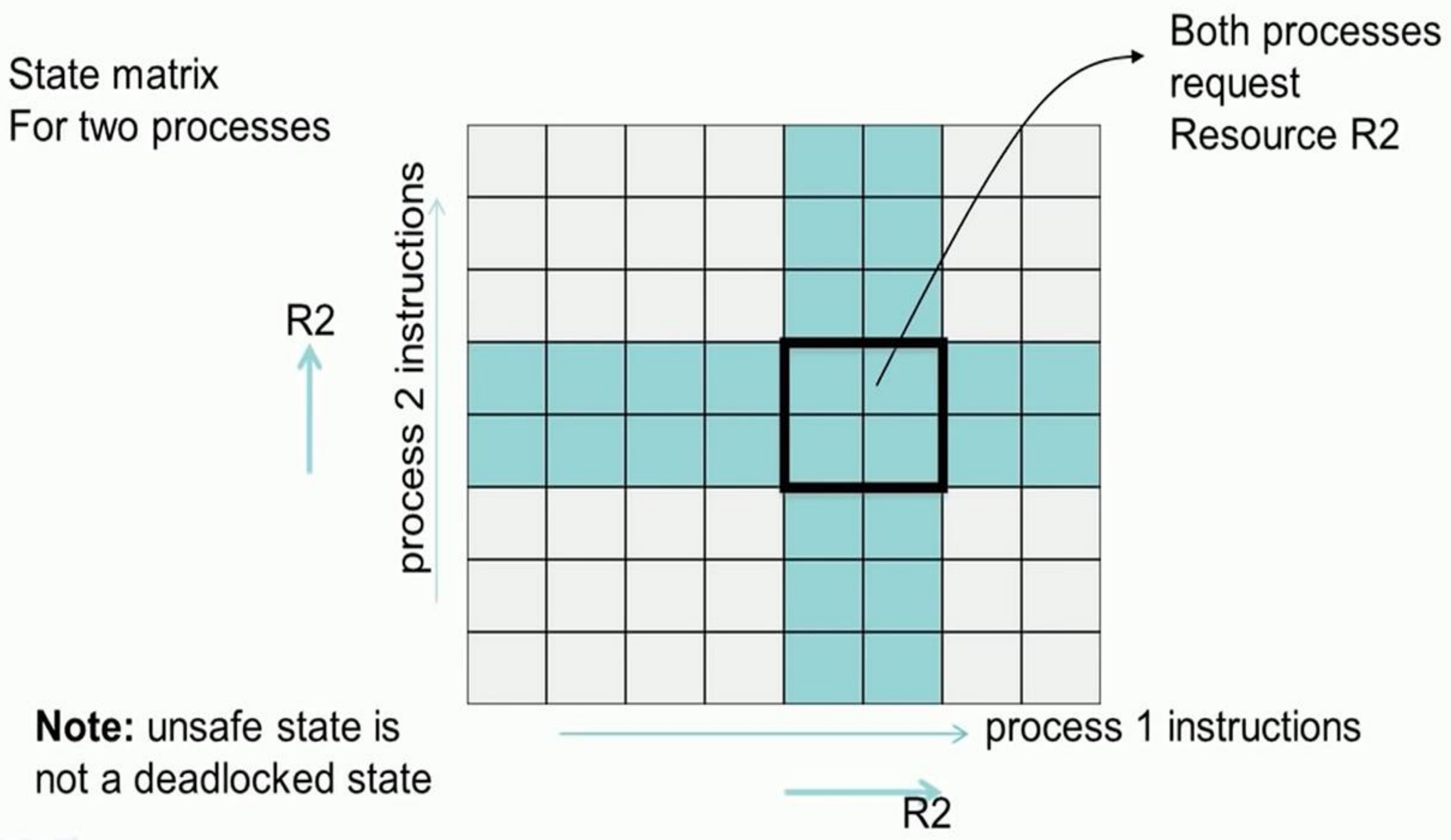
- State matrix for two processes



Note: unsafe state is not a deadlocked state

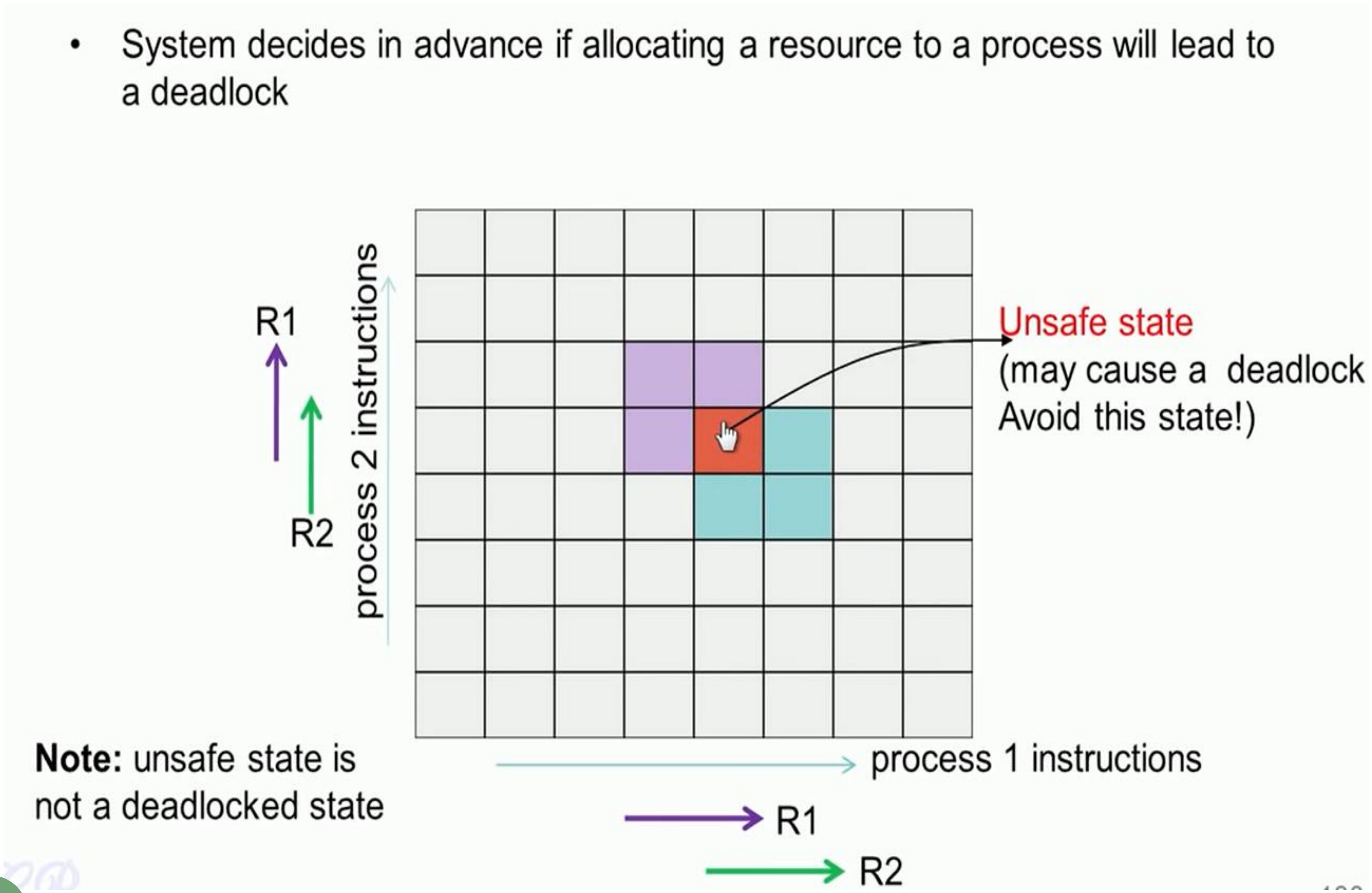
Deadlock Avoidance

- State matrix for two processes



Note: unsafe state is not a deadlocked state

Deadlock Avoidance

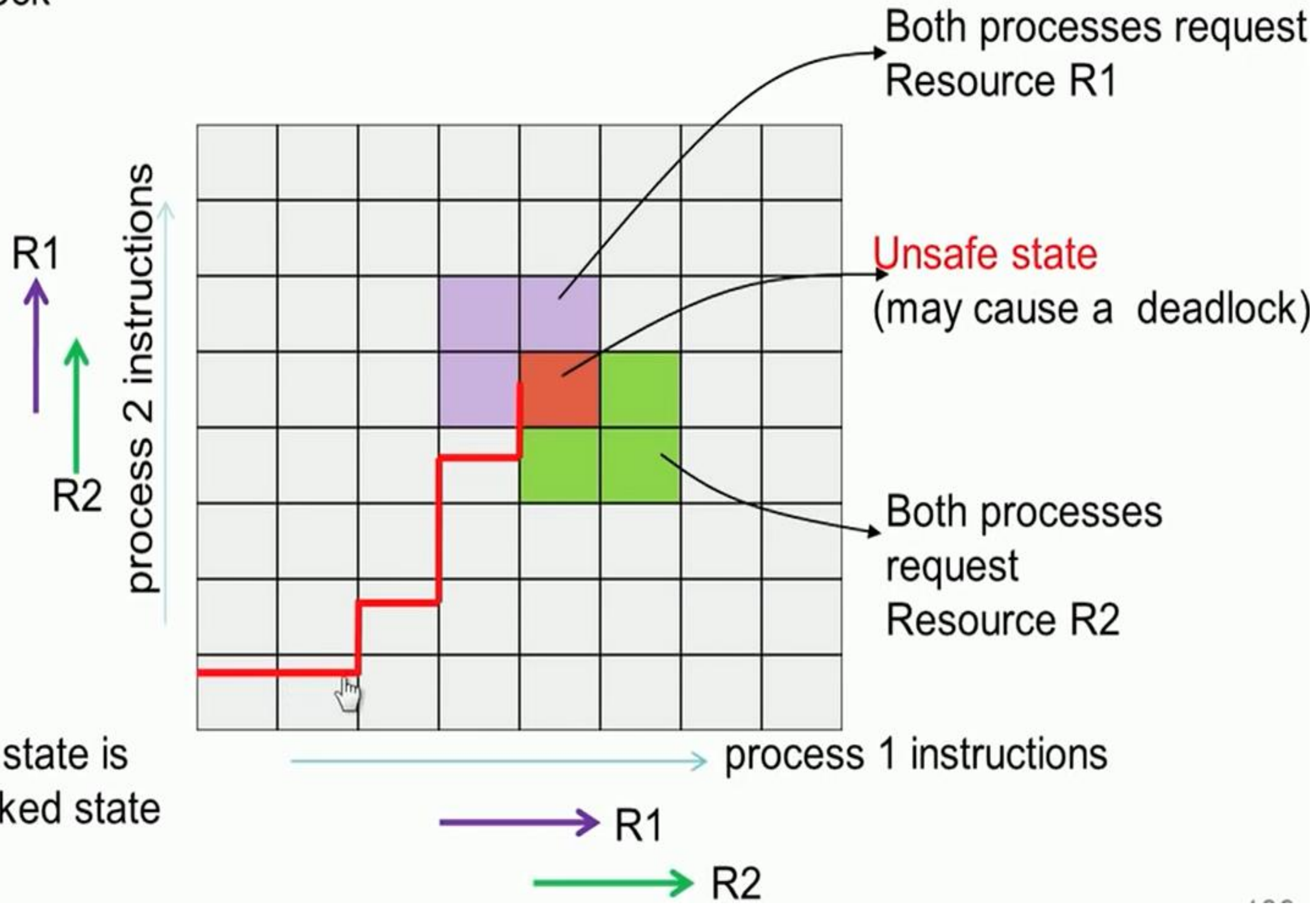


- System decides in advance if allocating a resource to a process will lead to a deadlock

Note: unsafe state is not a deadlocked state

Deadlock Avoidance

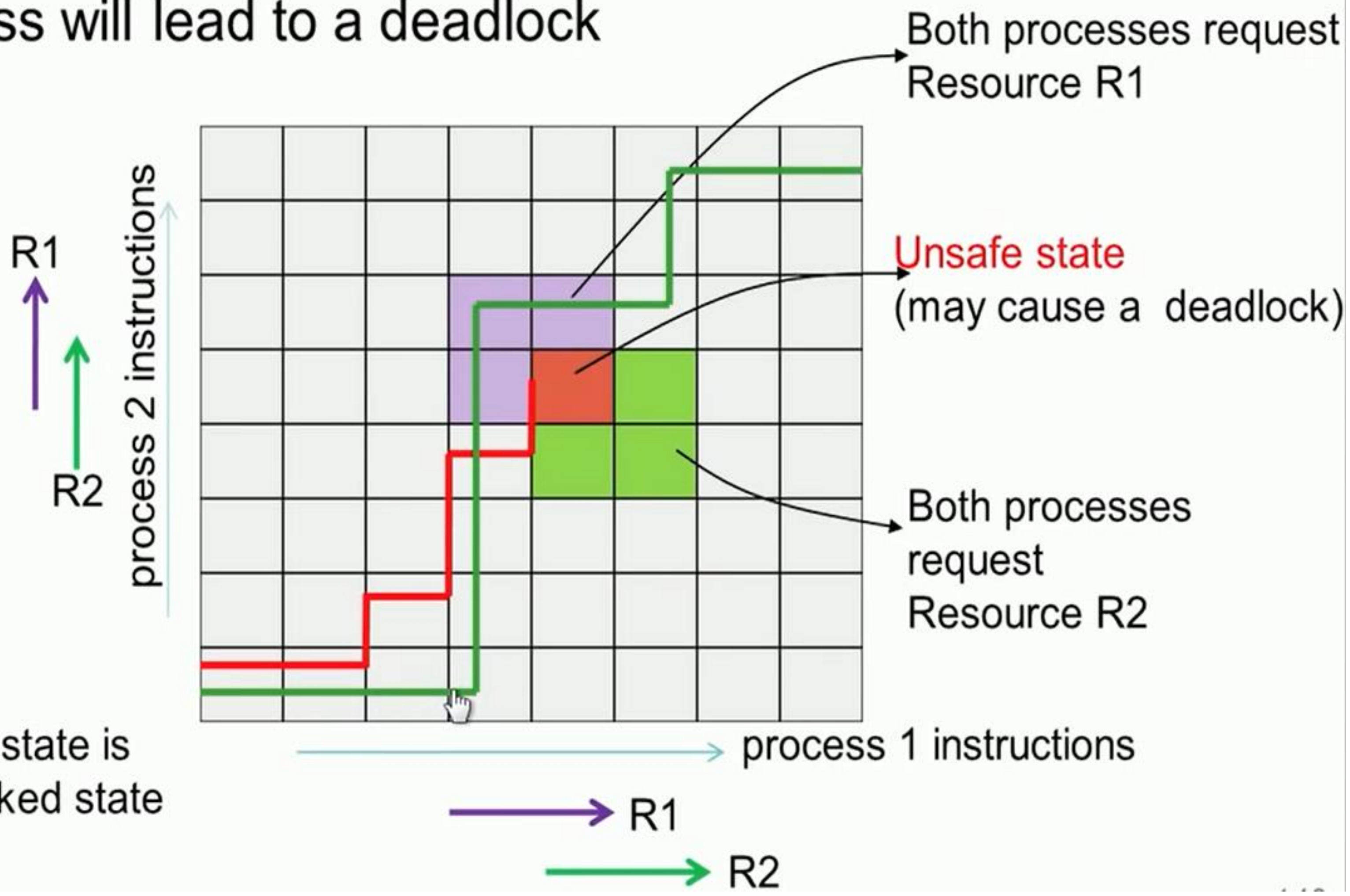
- System decides in advance if allocating a resource to a process will lead to a deadlock



Note: unsafe state is not a deadlocked state

Deadlock Avoidance

- System decides in advance if allocating a resource to a process will lead to a deadlock



Note: unsafe state is not a deadlocked state

Resource Trajectory

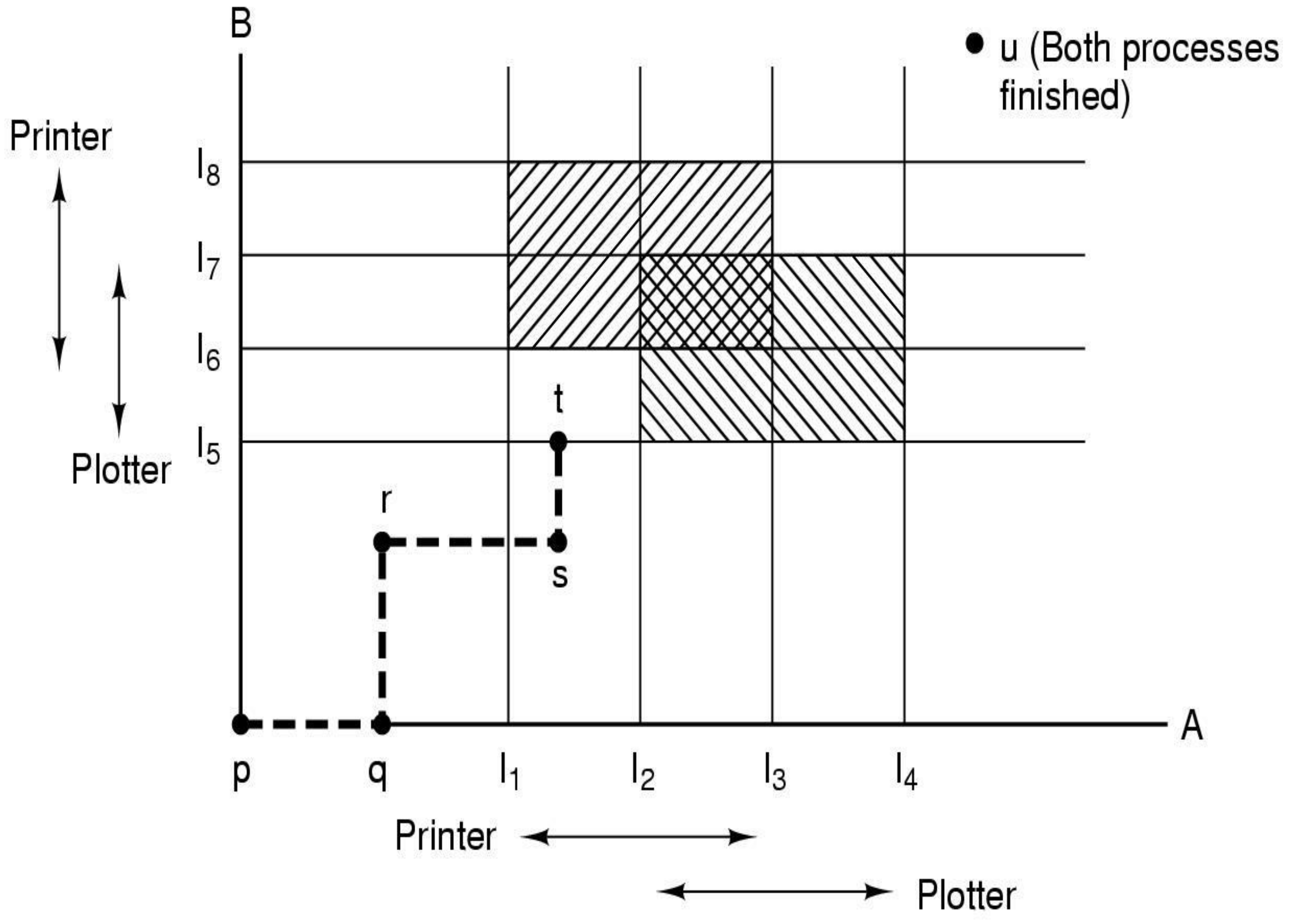


Figure : Two process resource trajectories.

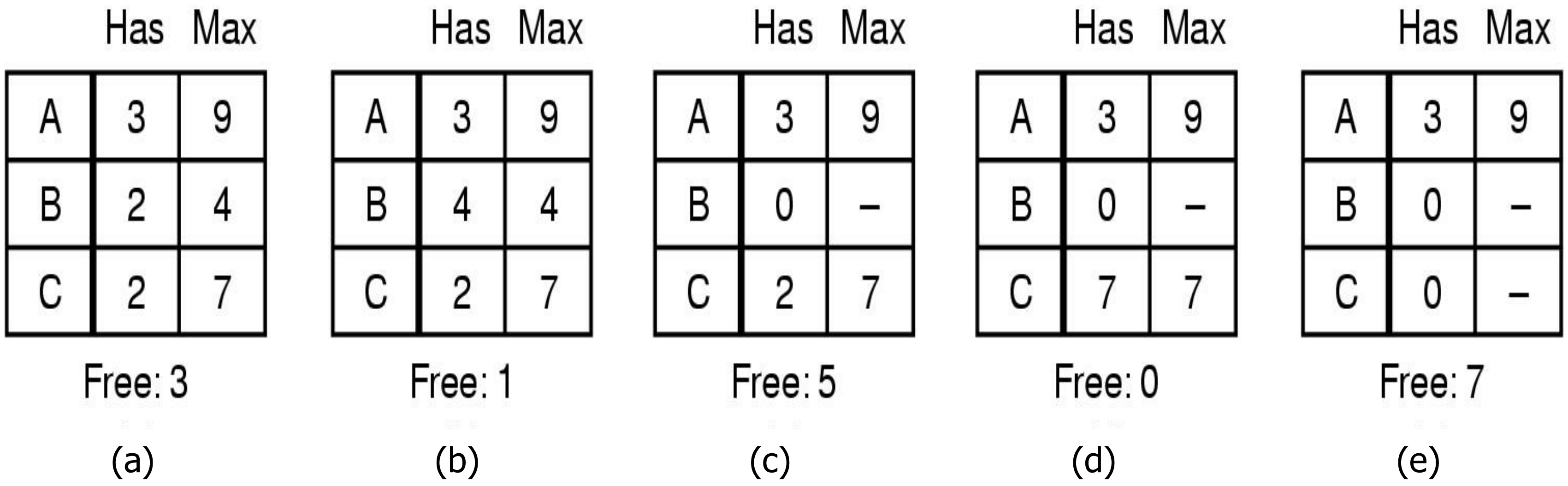
Resource Trajectory

- The horizontal axis represents the number of instructions executed by process A and vertical by B.
- p, q, r, s, t, u are few states.
- When A crosses the I_1 line on the path from r to s, it requests and is granted the printer. When B reaches point t, it requests the plotter.
- At intersection point of I_2 and I_6 , process A is requesting the plotter and B is requesting the printer, and both are already assigned.
- So this region is region of deadlock.(See in earlier figure)

Resource Trajectory

- Important thing to be noted:
  - At point t, process B is requesting a resource(plotter). The system must decide whether to grant it or not. If the grant is made, the system will enter an unsafe region and eventually deadlock.
  - To avoided the deadlock, process B should be suspended until process A has requested and released the plotter.

Safe and Unsafe States



(a)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(b)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(c)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(d)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(e)

	Has	Max
A	3	9
B	0	–
C	0	–

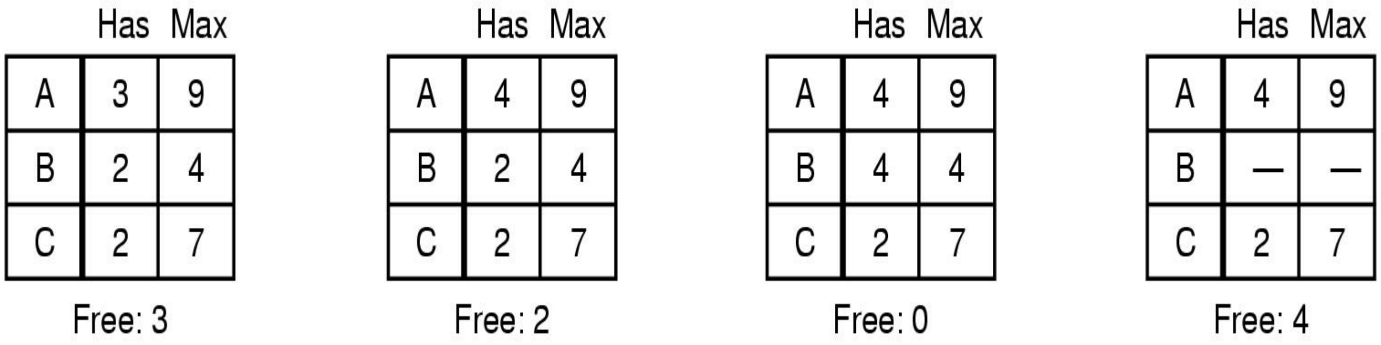
Free: 7

Figure: Demonstration that the state in (a) is safe

Safe and Unsafe States

- Sequence of run:
  - Process B leads to figure (b).
  - Process B completes, obtain (c).
  - Scheduler run process C, leading to (d).
  - When process C completes, get (e).
  - Now process A can get 6 instances.
  - Thus (a) is safe state.

Safe and Unsafe States



	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)

Figure: Demonstration that the sate in (b) is not safe

Note that: *An unsafe state is not a deadlocked state*. System can run for a while. From a safe state, the system can guarantee that all processes will finish; from an unsafe state, no such guarantee can be given.

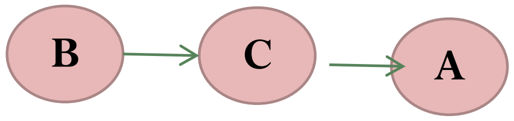
Example :

Process	Allocation	Max
A	3	2
B	2	4
C	2	7

The total number of instance available for resource is 10

Sol:

Process	Allocation	Max	required
A	3	2	6
B	2	4	2
C	2	7	5



1. If we start from A:
   then A need 6 but we have only 3, so not possible (deadlock possible)
2. If B,
   then B need 2 so we use our 3 resource on B. and remaining 1.
   After B is finished ,Total = 1 + 4 = 5
3. If c,
   the c need 5 and we have 5 so remaining is 0.
   After C is finished ,Total =2 +5 =7
4. Finally A,
   A need 6 and remaining 1 , After finished total = 3+7 = 10

The Banker’s Algorithm for a Single Resource

- Dijkstra's(1965).
- Model on the small-town banker might deal with a group of customers.
- Algorithm checks to see if granting the request leads to an unsafe state.
- If granting leads to an unsafe state, it is denied otherwise granting is carried out.
- **Analogy:** Banker-Operating system and Customers-processes.

The Banker’s Algorithm for a Single Resource

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

Free: 10

(a)

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

Free: 2

(b)

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

Free: 1

(c)

Figure : Three resource allocation states:
(a) Safe. (b) Safe. (c) Unsafe.

What would happen if a request from B for one more unit were granted in (b)? =>leads to (c), which is unsafe state.

Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

- ✓ *E= The existing resource, P= Possessed(allocated or assigned), A= Available Resource.*
- ✓ *Here, 1st is allocation matrix and 2nd one is need matrix.*
- ✓ *We can grant resources only when <u>Need≤ Available</u> (for a row).*

Banker's Algorithm for Multiple Resources

Algorithm for checking to see if a state is safe:

1. Look for row, R, whose unmet resource needs all ≤ A. If no such row exists, system will eventually deadlock since no process can run to completion
2. Assume process of the row chosen requests all resources it needs(which is guaranteed to be possible) and finishes. Mark process as terminated, add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes marked terminated (initial state was safe) or no process left whose resource needs can be met (there is a deadlock).

Assigned resources

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources still needed

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

a). Current Allocation Matrix *b). Request Matrix*

E (Existing Resources):	(6 3 4 2)
P (Possessed Resources):	(5 3 2 2)
A (Available Resources):	(1 0 2 0)

Soln:

- Process A, B & C can't run to completion since for each process, Request is greater than Available Resources.
- Now process D can complete since its requests row is less than that of Available resources.

Step 1:

When D completes running, the total available resources is:

==> A = (1, 0, 2, 0) + (1, 1, 0 , 1)= (2, 1, 2, 1)

Now Process E can run to completion

Step 2:

Now process E can also run to completion & return back all of its resources.

==> A = (0 , 0, 0, 0) + (2, 1, 2, 1) = (2, 1, 2, 1)

Step 3:

Now process A can also run to completion leading A to

(3, 0, 1, 1) + (2, 1, 2, 1) = (5, 1, 3, 2)

Step 4:

Now process C can also run to completion leading A to

(5, 1, 3, 2) + (1, 1, 1, 0) = (6, 2, 4, 2)

Step 5:

Now process B can run to completion leading A to

(6, 2, 4, 2) + (0, 1, 0 , 0) = (6, 3, 4, 2)

This implies the state is safe and Dead lock free

Questions:

Q1. A system has three processes and four allocable resources. The total four resource types exist in the amount as E= (4, 2, 3, 1). The current allocation matrix and request matrix are as follows: Using Bankers algorithm, explain if this state is deadlock safe or unsafe.

Current Allocation Matrix				
Process	Ro	R1	R2	R3
P0	0	0	1	0
P1	2	0	0	1
P2	0	1	2	0

Allocation Request Matrix				
Process	Ro	R1	R2	R3
P0	2	0	0	1
P1	1	0	1	0
P2	2	1	0	0

Q2. Consider a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

1) What will be the content of the need Matrix?
2) Is the system in safe state? If yes, then what is the safe sequence?

soln

Need [i,j]= Max [i,j] – Allocation[i,j]

content of Need Matrix is:

	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

1. applying Safety algorithms

For P_i if $Need_i <=$ Available, then p_i is in Safe sequence,

Available = Available + $Allocation_i$

For P0,

need0=7,4,3

Available = 3,3,2

==> Condition is false, So P0 must wait.

For P1 ,

need1= 1,2,2

Available=3,3,2

need1< Available

So P1 will be kept in safe sequence. & Available will be updated as:

Available= 3,3,2 + 2,0,0 = 5,3,2

For P2,

need2= 6,0,0

Available = 5,3,2

==> condition is again false, so P2 also have to wait.

For P3,

need3= 0,1,1

Available= 5,3,2

==> condition is true , P3 will be in safe sequence.

Available = 5,3,2 + 2,1,1 = 7,4,3

For P4,

need4= 4,3,1

Available = 7,4,3

==> condition Needi<= Available is true, so P4 will be in safe sequence

Available = 7,4,3 + 0,0,2 = 7,4,5

Now we have two processes P0 and P2 in waiting state. Either P0 or P1 can be chosen.

Let us take P2

whose need = 6,0,0

Available = 7,4,5

Since condition is true, P2 now comes in safe state leaving the

Available = 7,4,5 + 3,0,2 = 10, 4,7

Next P0

whose need = 7, 4, 3

Available = 10,4,7

since condition is true P0 also can be kept in safe state.

So system is in safe state & the safe sequence is <P1, P3, P4, P2, P0>

Deadlock Detection

- We try to detect when deadlock happens, and then take action to recover after the fact.

1. Deadlock Detection with one resource of each type:

Example:

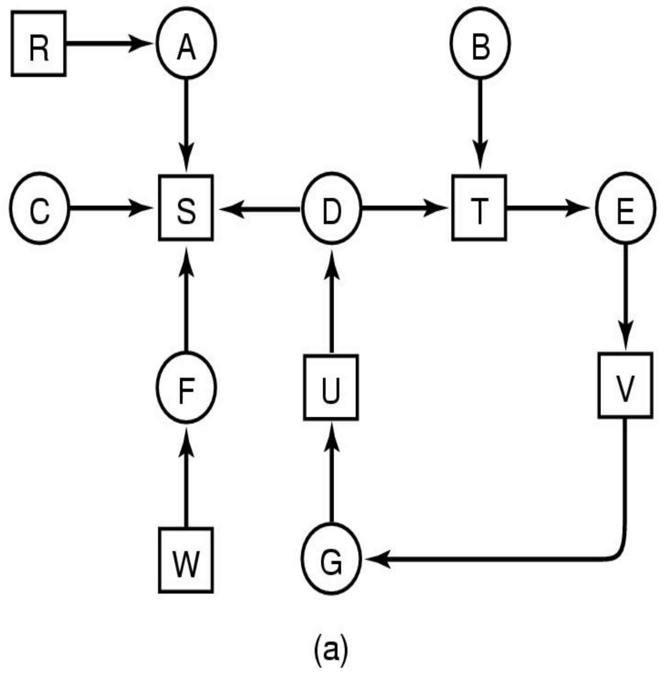
- Consider a complex system with 7 processes, A through G, and 6 resources, R through W.
- The state of which resources are currently owned and which ones are currently being requested is as follows-

Deadlock Detection

1. Process A holds R and wants S.
2. Process B holds nothing but wants T.
3. Process C holds nothing but wants S.
4. Process E holds T and wants V.
5. Process F holds W and wants S.
6. Process G holds V and wants U.

From this, We can construct resource graph as follows:

Deadlock Detection



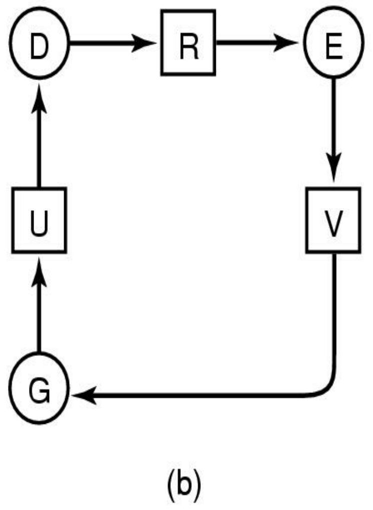
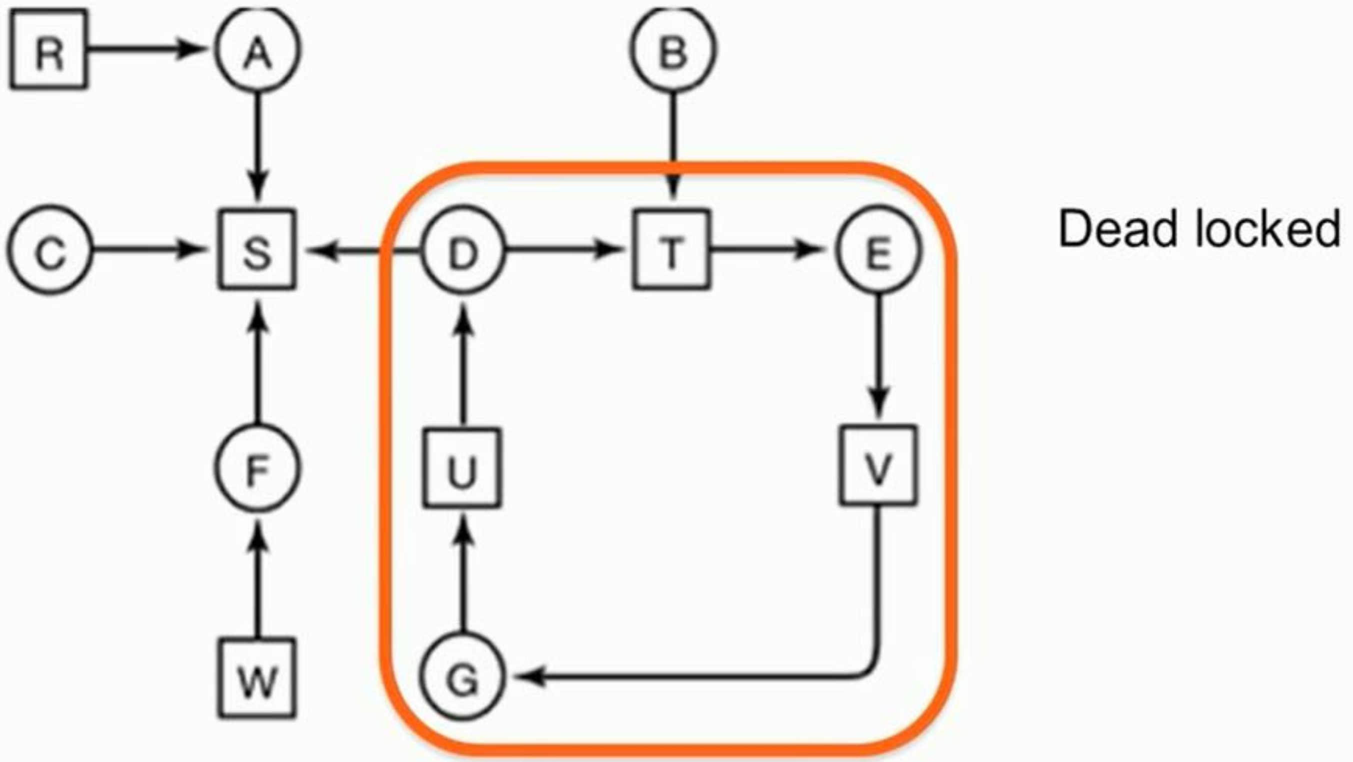


Figure: (a) A resource graph. (b) A cycle extracted from (a).

- *Processes D, E and G are all deadlocked.*
- *Processes A,C and F are not deadlocked because S can be allocated to any one of them, which then finishes and take by other two processes in turn.*

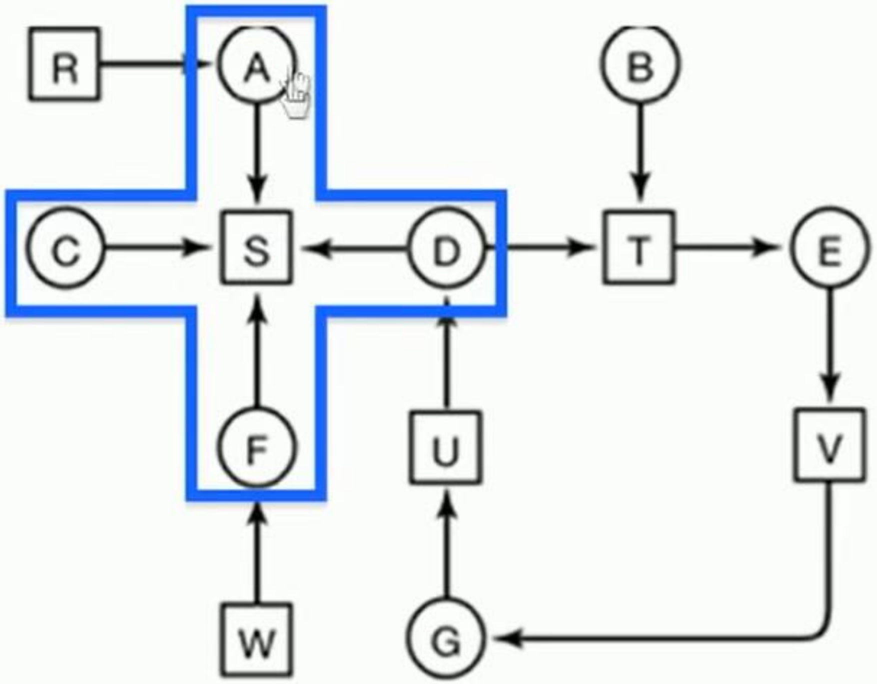
Deadlock Detection

- Deadlock detection with **one resource of each type**
- Find cycles in resource graph



Deadlock Detection

- Deadlock detection with **one resource of each type**
- There should be atleast one sequence of resource allocation, to avoid a deadlock



Not a Dead lock as an allocation sequence possible

Sample Allocation Sequence

S allocated to A, after A completes S allocated to C then, S allocated to F, and finally S allocated to D

Deadlock Detection with One Resource of Each Type

Algorithm for detecting deadlock:

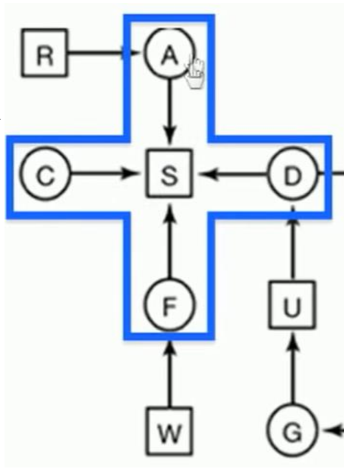
1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates. …

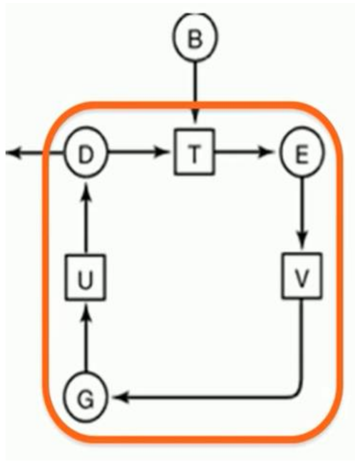
Deadlock Detection with One Resource of Each Type

4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. 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 3.

Deadlock Detection with One Resource of Each Type

- **Start at R** and initialize L to empty list.
- Then add R to the list and move to only possibility, A and add it to L, giving L=[R, A].
- From A, go to S, giving L=[R, A, S].
- S has no outgoing arcs, so it is dead end; forcing us to backtrack to A.
- Since A has no unmarked outgoing arcs, we backtrack to R, completing our inspection of R.
- **Starting at A** also quickly terminates.
- ***But*, starting from B**, we get L=[B,T,E,V,G,U,D] until we get D and finally L=[B,T,E,V,G,U,D,T], we get cycle and stop. (S is a dead end, so we backtrack from S to D)





Deadlock Detection with Multiple Resources

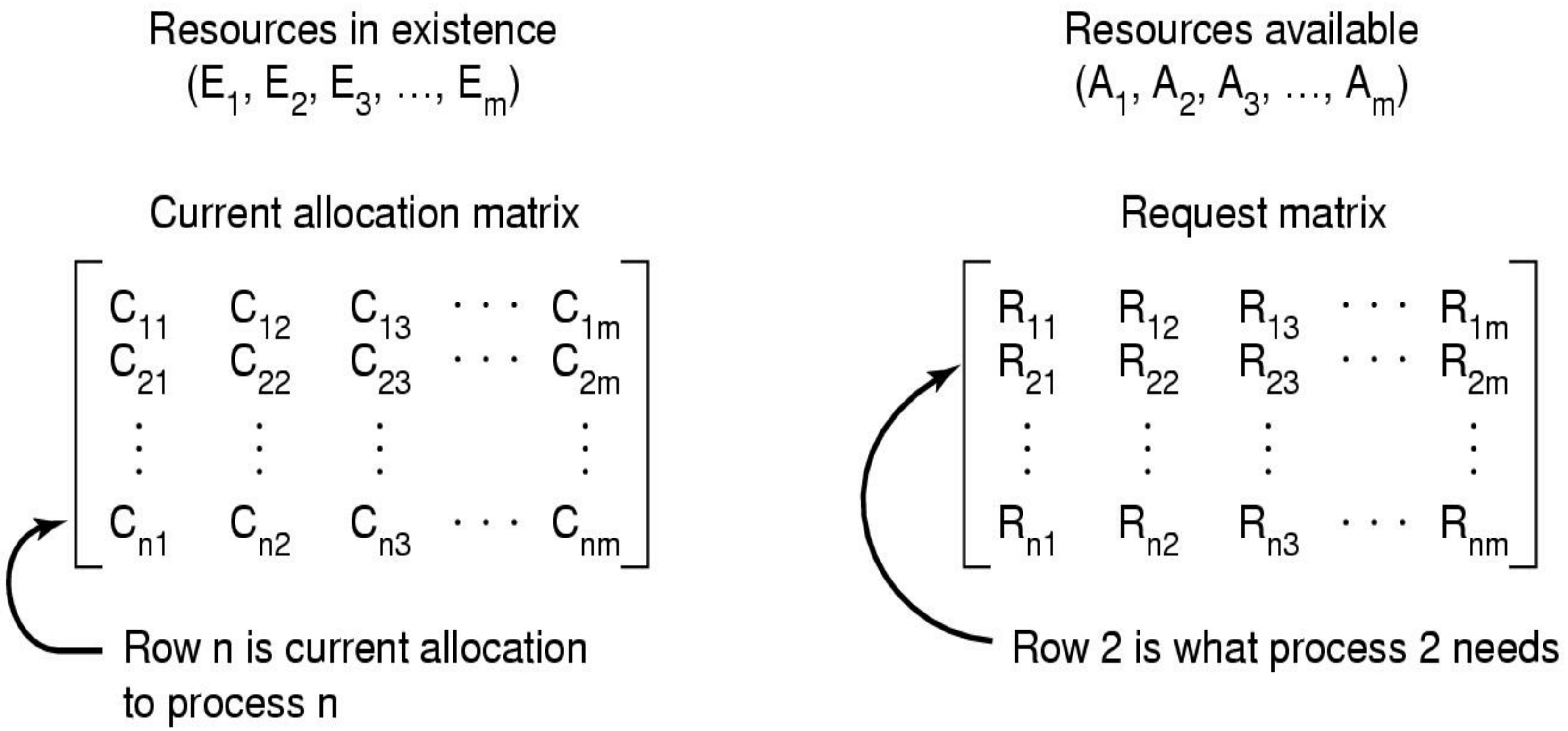


Figure: The four data structures needed by the deadlock detection algorithm.

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Deadlock Detection with Multiple Resources

- Total n processes: P_1 through P_n
- Total m resource classes: E_1 through E_m
- E_i : existing resource vector and
- A_i : available resource vector. $(1 \leq i \leq m)$
- C: current allocation matrix.
- R: request or need matrix.

Deadlock Detection with Multiple Resources

Deadlock detection algorithm:

1. Look for an unmarked process, P_i , for which the i-th row of R is less than or equal to A.
2. If such a process is found, add the *i-th* row of C to A, mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.

Deadlock Detection with Multiple Resources

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (4 & 2 & 3 & 1) \end{array}$$

$$A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (2 & 1 & 0 & 0) \end{array}$$

Current allocation matrix

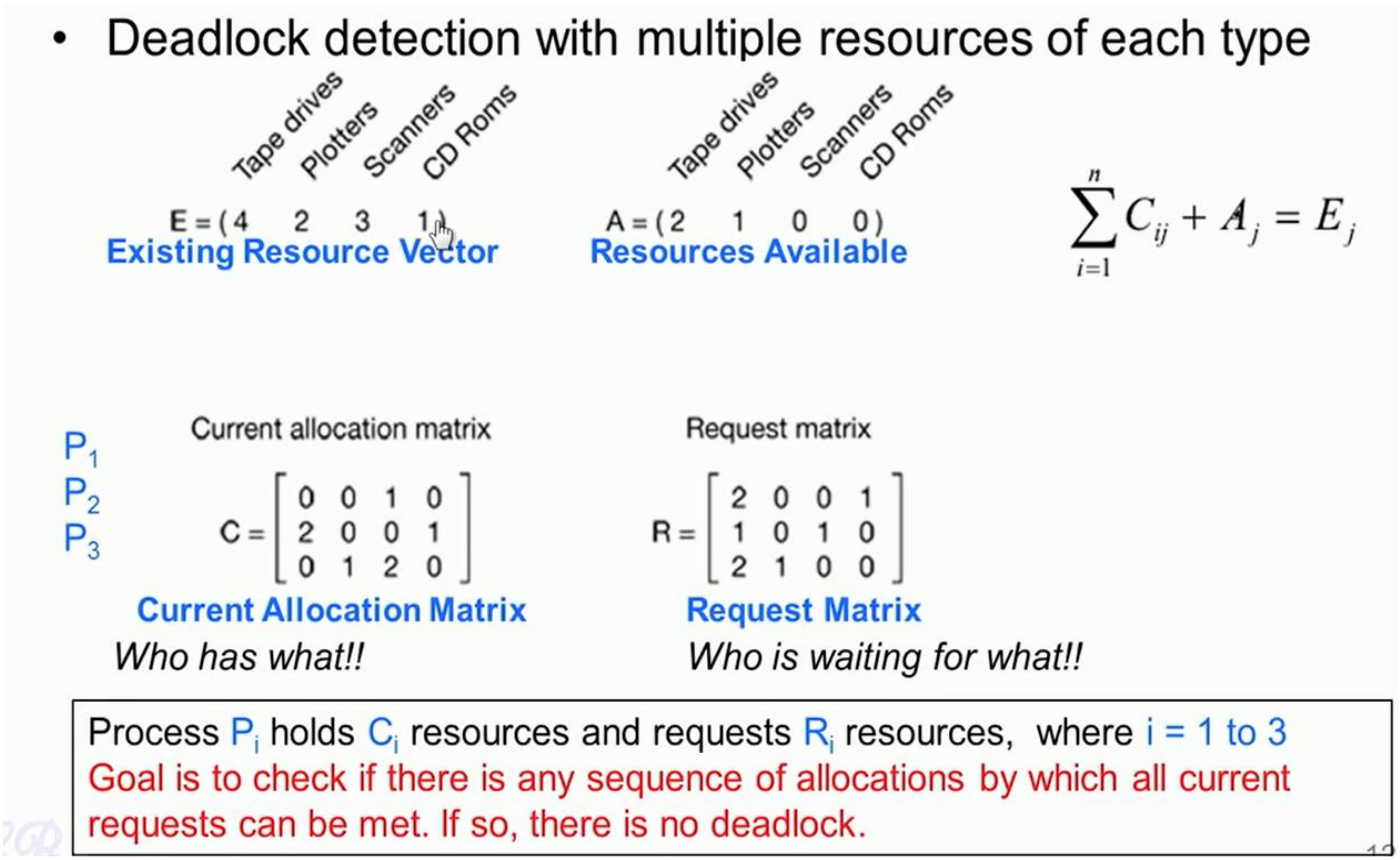
$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Figure : An example for the deadlock detection algorithm.

Deadlock Detection



- Deadlock detection with multiple resources of each type

$$\begin{array}{cccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} & \\ E = (& 4 & 2 & 3 & 1 &) \end{array}$$

Existing Resource Vector

$$\begin{array}{cccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} & \\ A = (& 2 & 1 & 0 & 0 &) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{l} P_1 \\ P_2 \\ P_3 \end{array} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Who has what!!

Request matrix

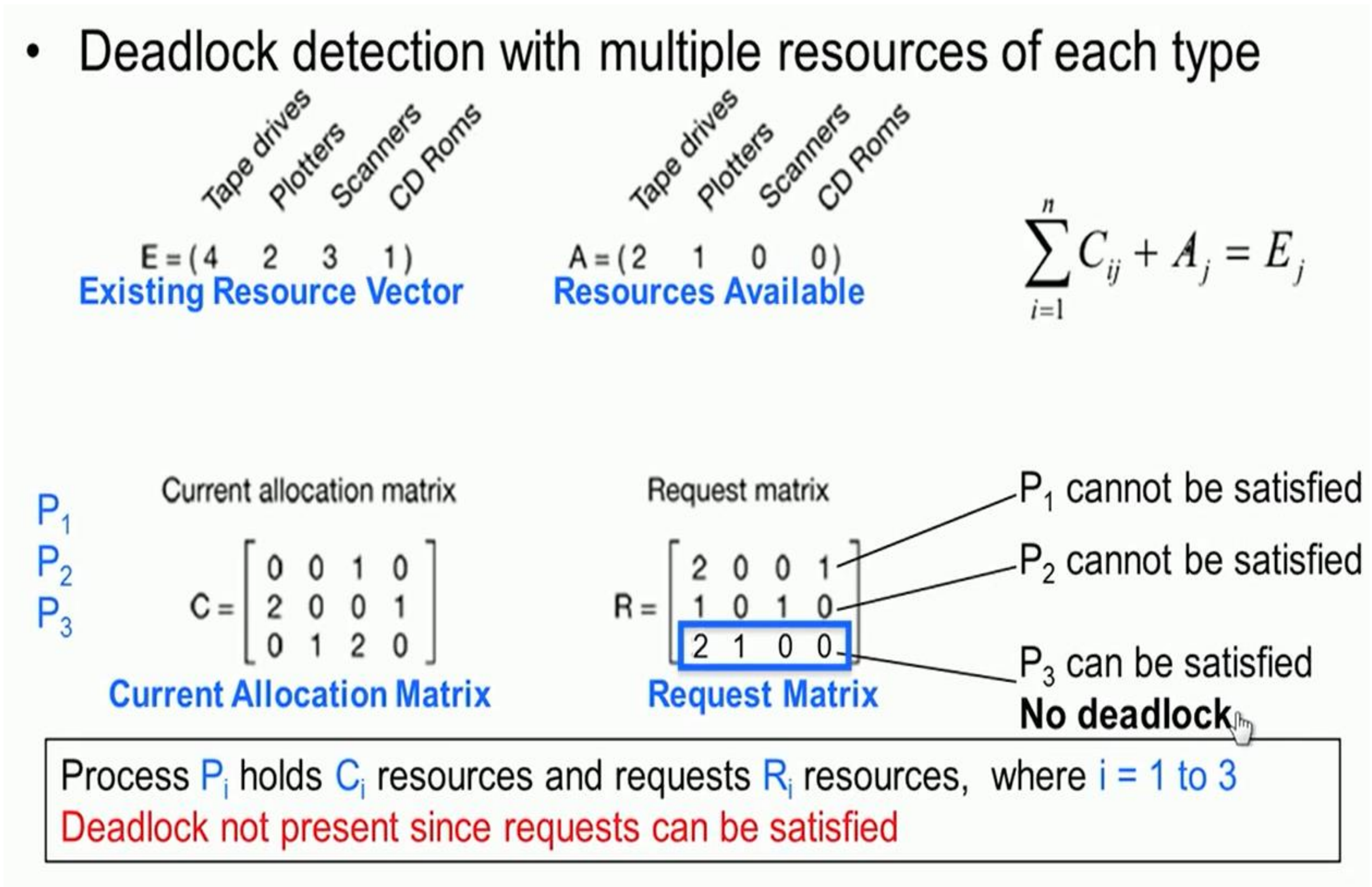
$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Request Matrix

Who is waiting for what!!

Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3
Goal is to check if there is any sequence of allocations by which all current requests can be met. If so, there is no deadlock.

Deadlock Detection



- Deadlock detection with multiple resources of each type

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (4 & 2 & 3 & 1) \end{array}$$

Existing Resource Vector

$$A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (2 & 1 & 0 & 0) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$\begin{array}{l} P_1 \\ P_2 \\ P_3 \end{array} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \boxed{2 \;\; 1 \;\; 0 \;\; 0} \end{bmatrix}$$

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 can be satisfied

No deadlock

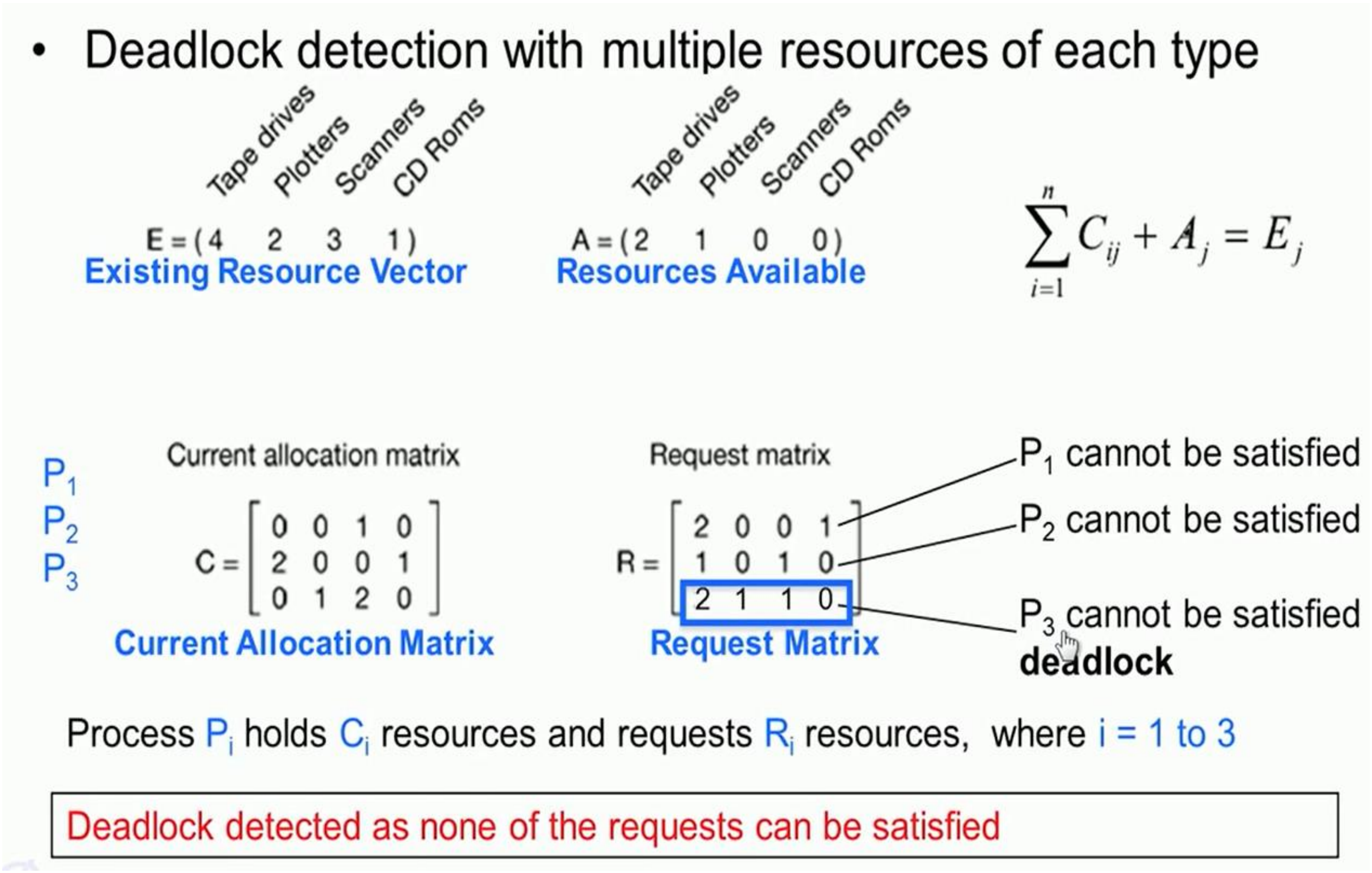
Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3

Deadlock not present since requests can be satisfied

Deadlock Detection with Multiple Resources

- Here, requests represented by 1st and 2nd rows of matrix R can not be satisfied (Compare A with each Rs) . But 3rd one can be satisfied.
- So process 3 runs and eventually returns A=(2 2 2 0).
- At this point process 2 can run and return A=(4 2 2 1).
- Now process 1 can run and there is no deadlock.
- **What happen** if process 2 needs 1 CD-ROM drive, 2 Tape drives and 1 Plotter (i.e. 3rd row of R becomes (2 1 1 0))?
- Entire system gets deadlock or not???

Deadlock Detection



- Deadlock detection with multiple resources of each type

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1\ ) \end{array}$$

Existing Resource Vector

$$\begin{array}{ccccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0\ ) \end{array}$$

Resources Available

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Current allocation matrix

$$C = \begin{array}{c} P_1 \\ P_2 \\ P_3 \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Current Allocation Matrix

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ \boxed{2 \quad 1 \quad 1 \quad 0} \end{bmatrix}$$

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 cannot be satisfied

deadlock

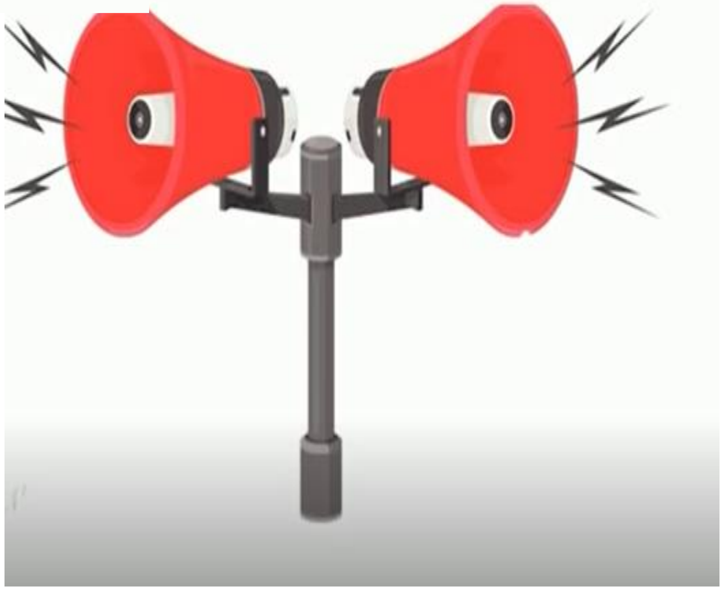
Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3

Deadlock detected as none of the requests can be satisfied

Deadlock Recovery

What should the OS do when it detects a deadlock?

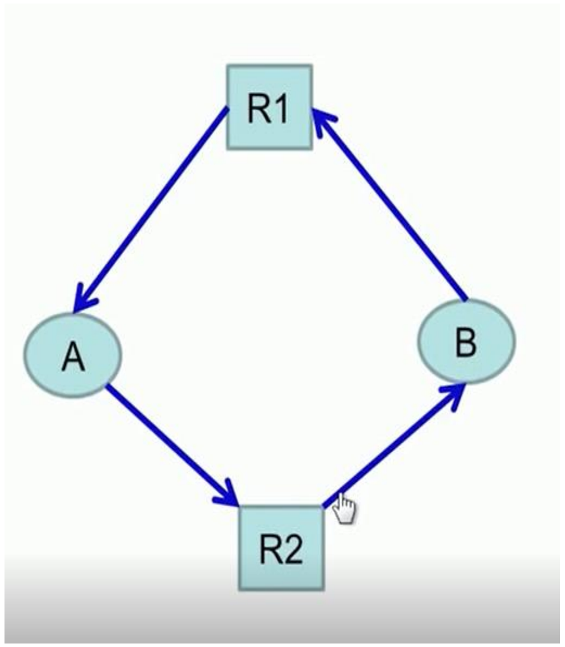
- Raise an alarm
  - Tell users and administrator

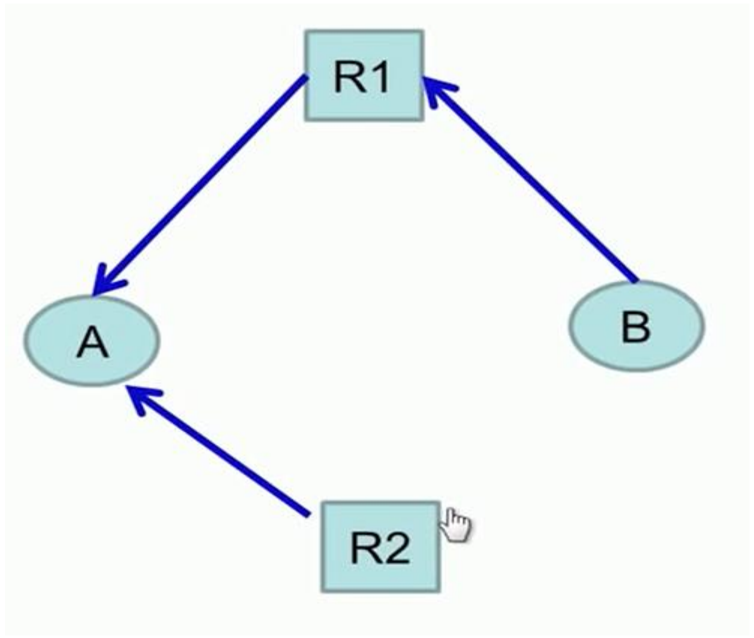


Recovery from Deadlock

(a) Recovery through preemption

- Temporarily take a resource away from its current owner and give it to anther process.
- It depends on nature of the resources.

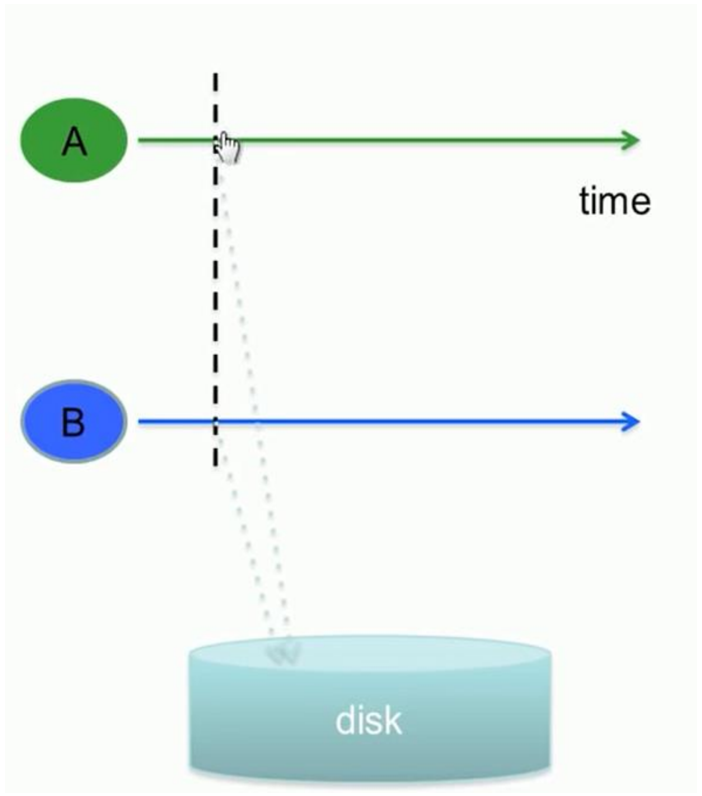


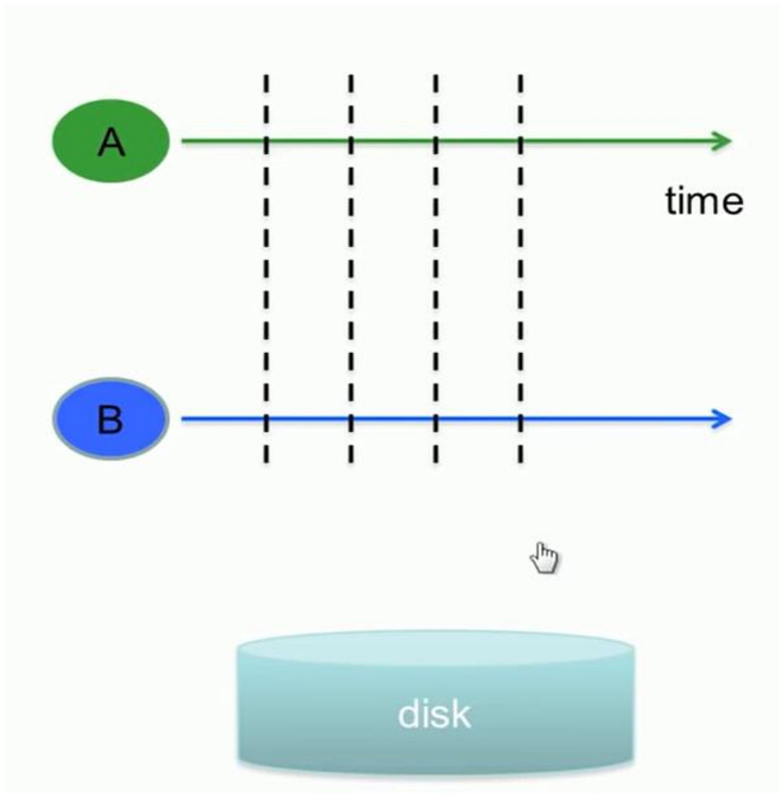


Recovery from Deadlock

(b) Recovery through rollback

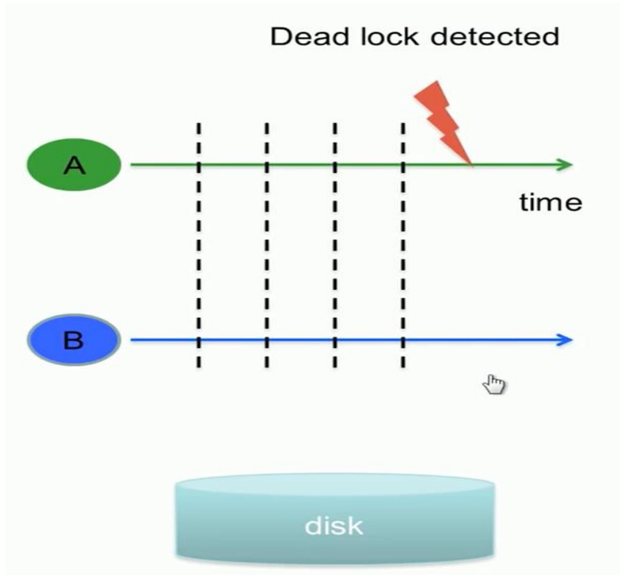
- Checkpoint a process periodically *(checkpointing a process means that its state is written to a file so that it can be restarted later).*

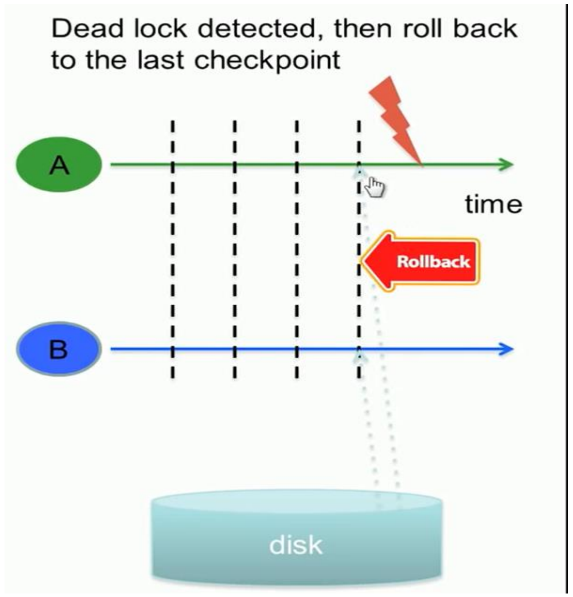




Recovery from Deadlock

- To do the recovery, a process is rolled back to a point in time before it acquired some other resource by starting one of its earlier check-points or to the last known state prior to deadlock.





- That is, restart the process if it is found deadlocked.

Recovery from Deadlock

(c) Recovery through killing processes

- Crudest but simplest way to break a deadlock.
- Kill one of the processes in the deadlock cycle.
- The other processes get its resources.
- Choose process to kill, that will yield no ill effect to the entire system.-

Recovery from Deadlock

- There are three basic approaches to recovery from deadlock:Inform the system operator, and allow him/her to take manual intervention.
- Terminate one or more processes involved in the deadlock
- Preempt resources.

Recovery from Deadlock

Process Termination

- Two basic approaches, both of which recover resources allocated to terminated processes:
  - Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
  - Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.
- In the latter case there are many factors that can go into deciding which processes to terminate next:
  - Process priorities.
  - How long the process has been running, and how close it is to finishing.
  - How many and what type of resources is the process holding. (Are they easy to preempt and restore?)
  - How many more resources does the process need to complete.
  - How many processes will need to be terminated
  - Whether the process is interactive or batch.
  - (Whether or not the process has made non-restorable changes to any resource.)

Recovery from Deadlock

Resource Preemption

- When preempting resources to relieve deadlock, there are three important issues to be addressed:
  - **Selecting a victim** - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
  - **Rollback** - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (I.e. abort the process and make it start over.)
  - **Starvation** - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Finished Unit 3