CS304 Operating Systems

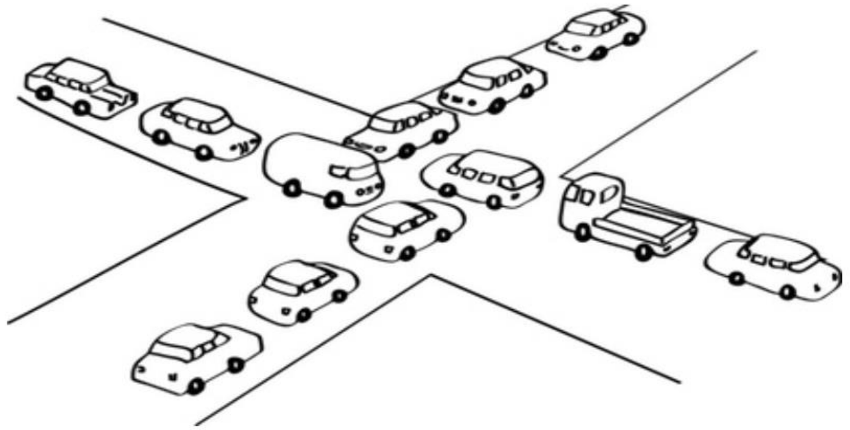
DR GAYATHRI ANANTHANARAYANAN

gayathri@iitdh.ac.in

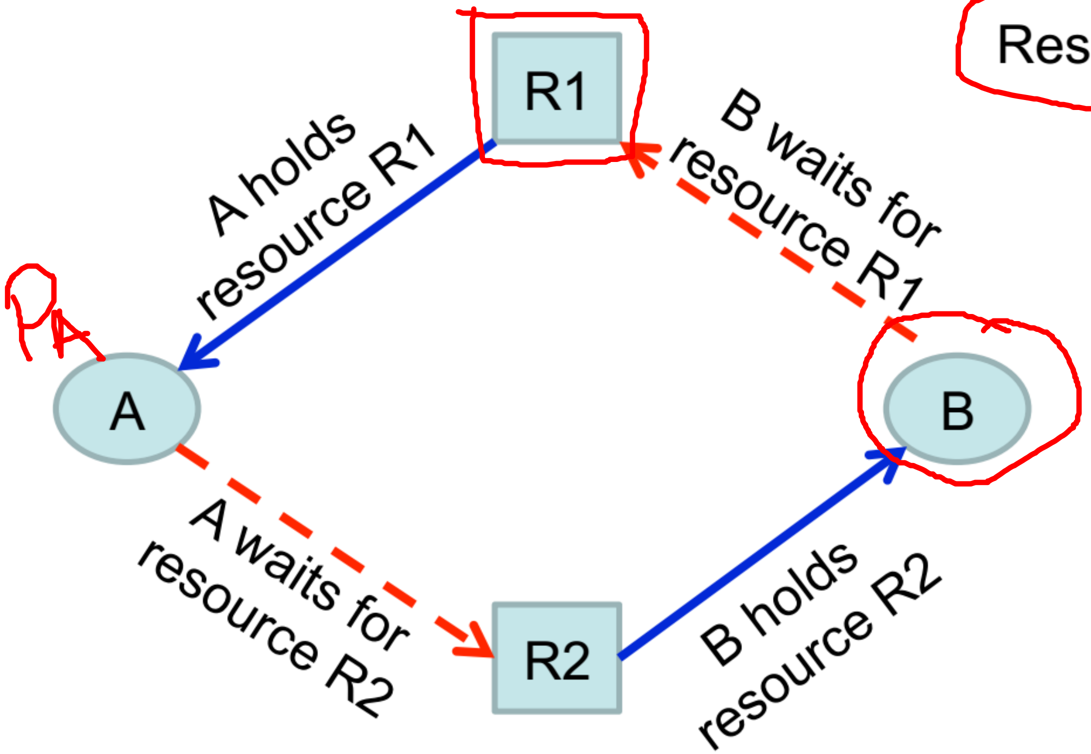
Materials in these slides have been borrowed from textbooks and existing operating systems courses

17 Feb 2021, Deadlocks

- A situation where programs continue to run indefinitely without making any progress
- Each program is waiting for an event that another process can cause



Deadlocks



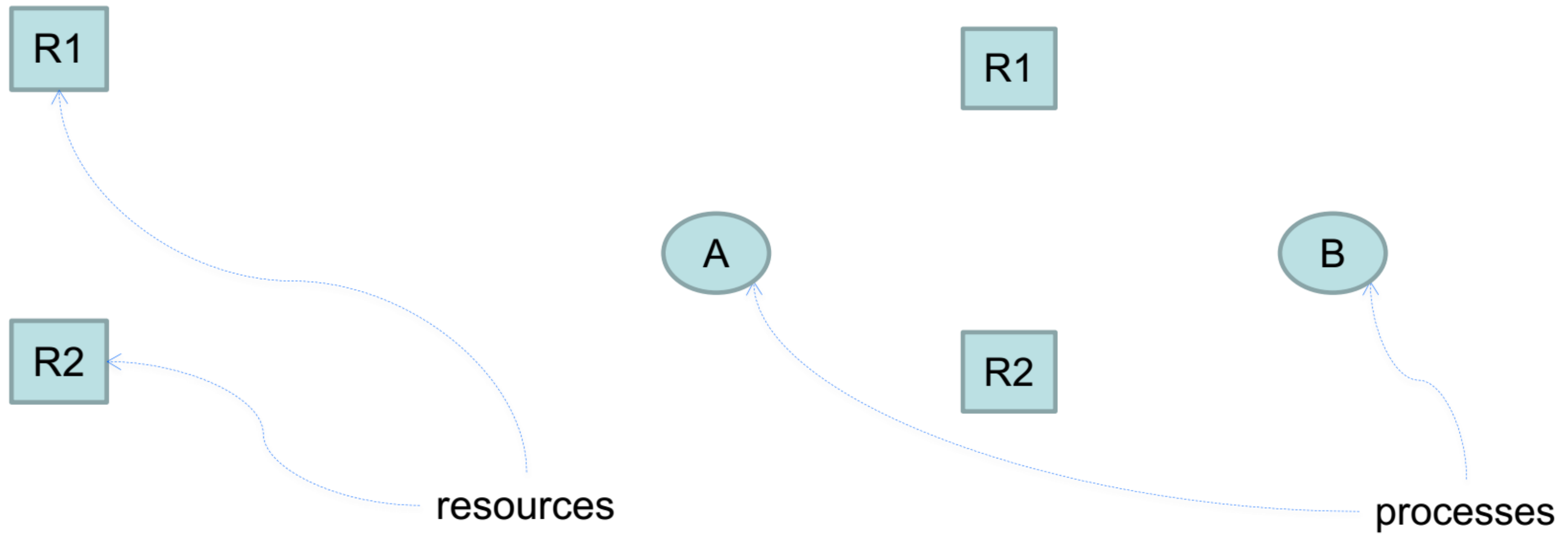
Resource Allocation Graph

A Deadlock Arises:

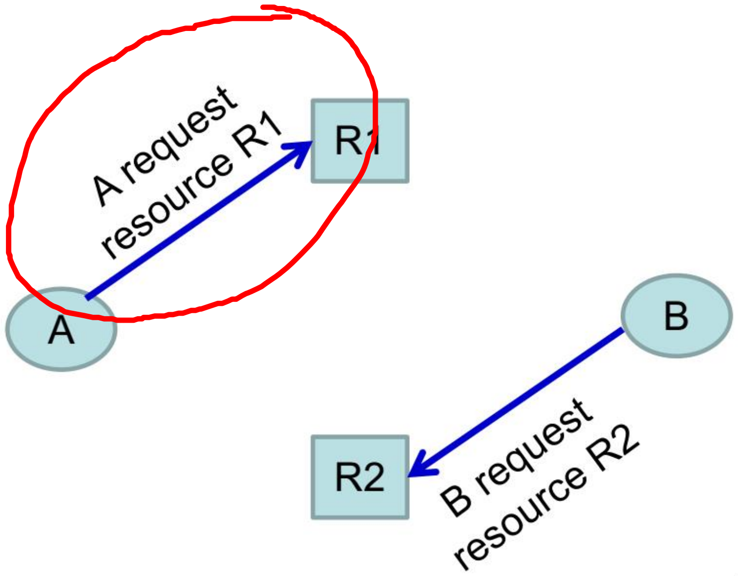
Deadlock : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

Resource Allocation Graph

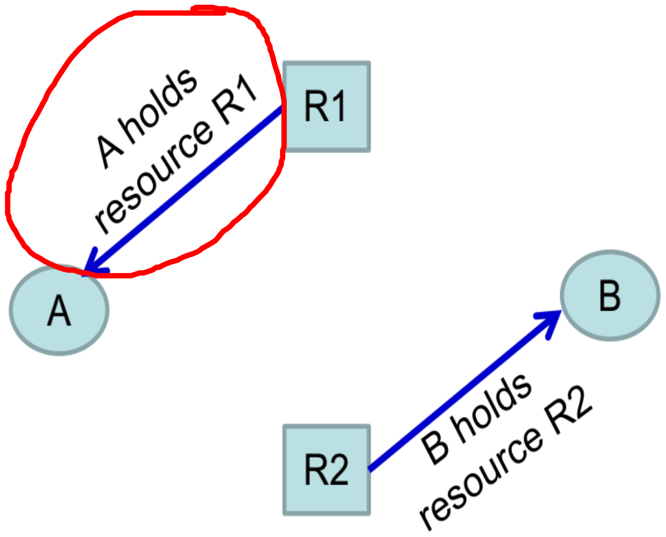
- It is a directed graph
- Used to model resource allocations



Resource Allocation Graph



Resource requested by process

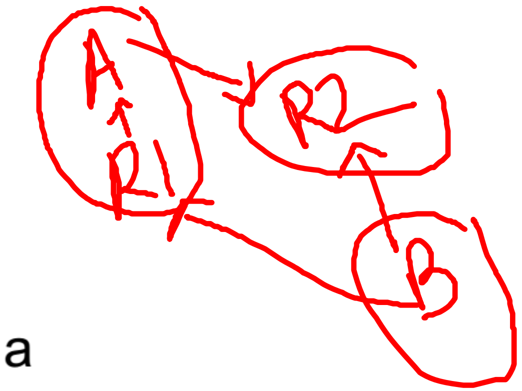


Resource held by process

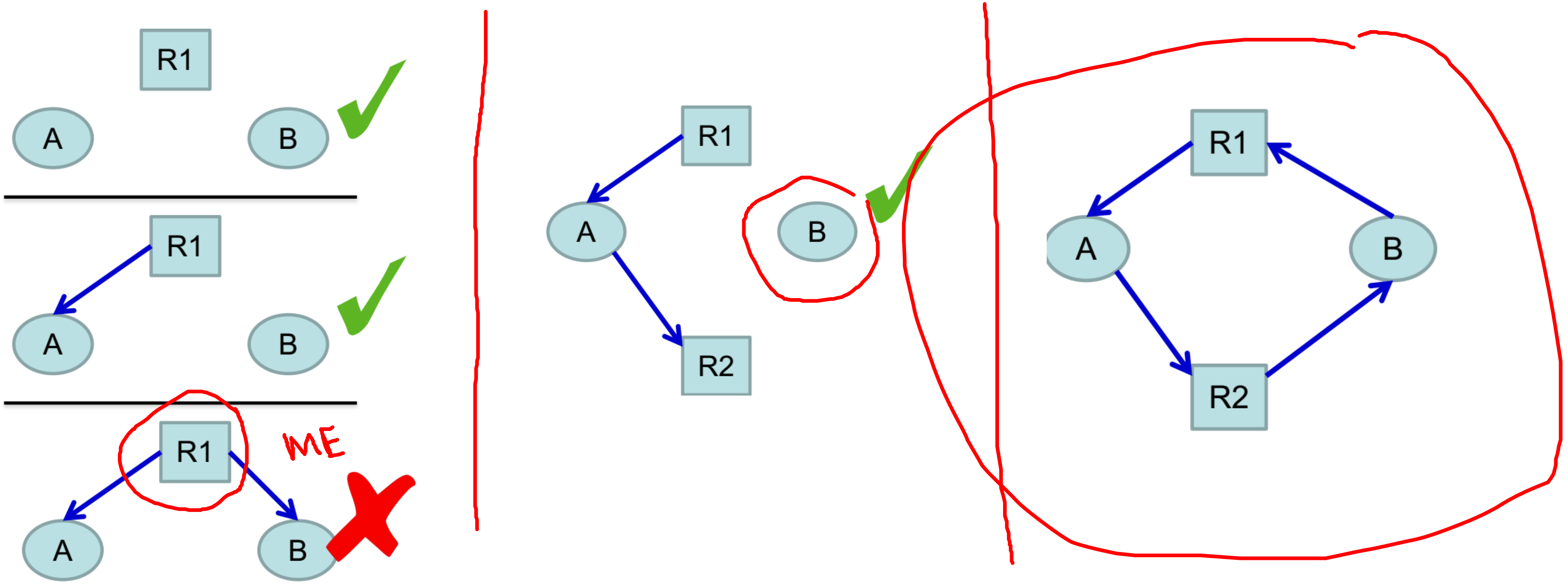
Resource Deadlocks

All four of these conditions must be present for a resource deadlock to occur!!

1. **Mutual Exclusion**
   - Each resource is either available or currently assigned to exactly one process
2. **Hold and wait**
   - A process holding a resource, can request another resource
3. **No preemption**
   - Resources previously granted cannot be forcibly taken away from a process
4. **Circular wait**
   - There must be a circular chain of two or more processes, each of which is waiting for a resouce held by the next member of the chain



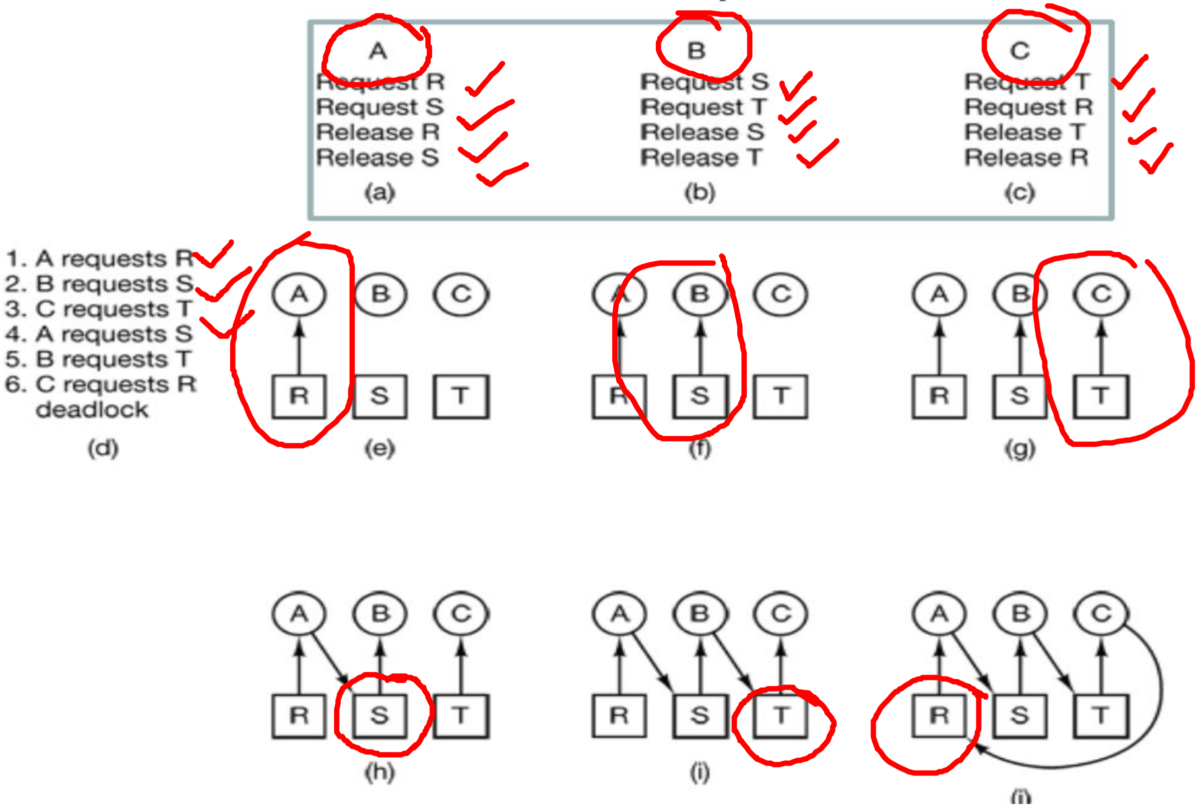
Resource Deadlocks



Deadlocks – Chanced Event

Ordering of resource requests and allocations are probabilistic, thus deadlock occurrence is also probabilistic

R S T

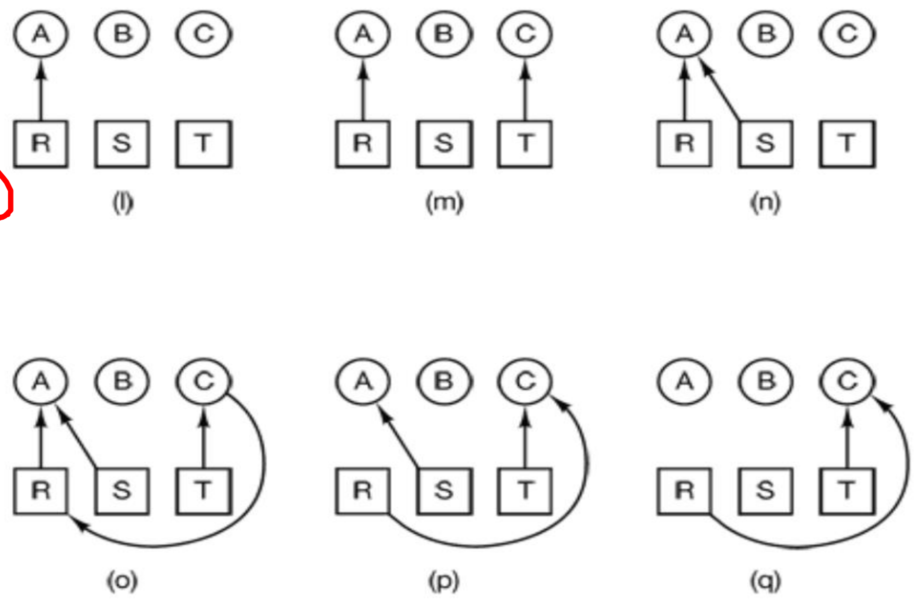


Deadlock occurs

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. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

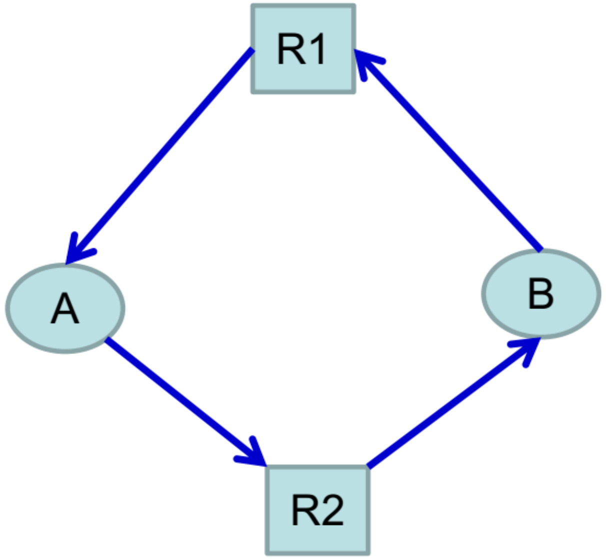
(k)



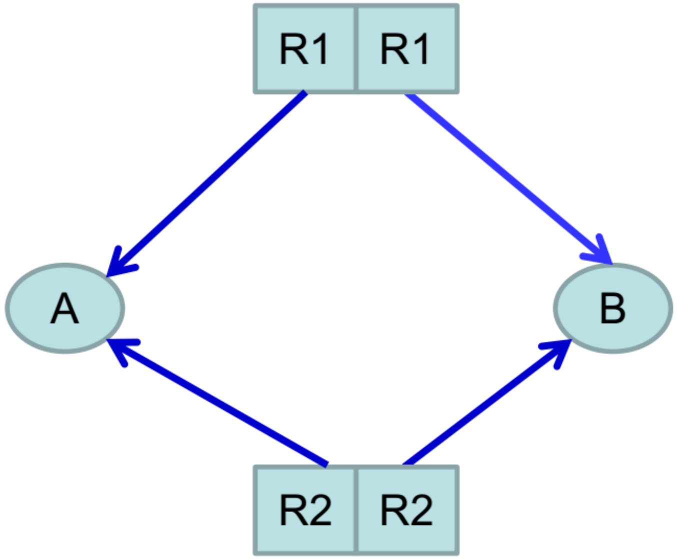
No dead lock occurrence
(B can be granted S
after step q)

Multiple resources

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



A deadlocked state



No Deadlocks

Should deadlocks be handled ?

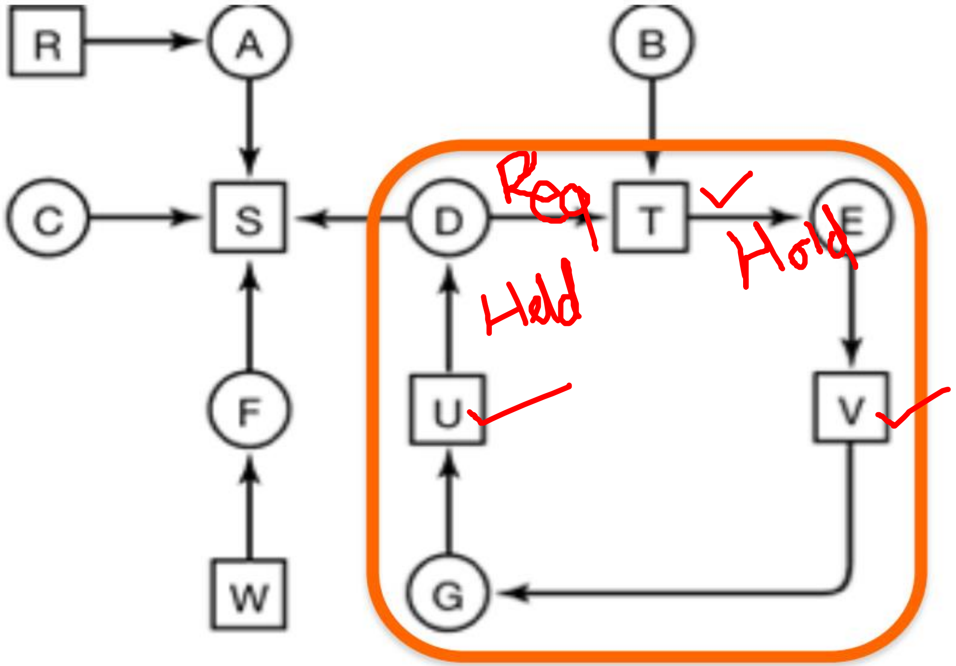
- Preventing / detecting deadlocks could be tedious
- Can we live without detecting / preventing deadlocks?
  - What is the probability of occurrence? ??
  - What are the consequences of a deadlock? (How critical is a deadlock?) ??
- Detection and Recovery
- Avoidance
- Prevention

Deadlock Detection

- How can an OS detect when there is a deadlock?
- OS needs to keep track of
  - Current resource allocation
    - Which process has which resource
  - Current request allocation
    - Which process is waiting for which resource
- Use this informaiton to detect deadlocks

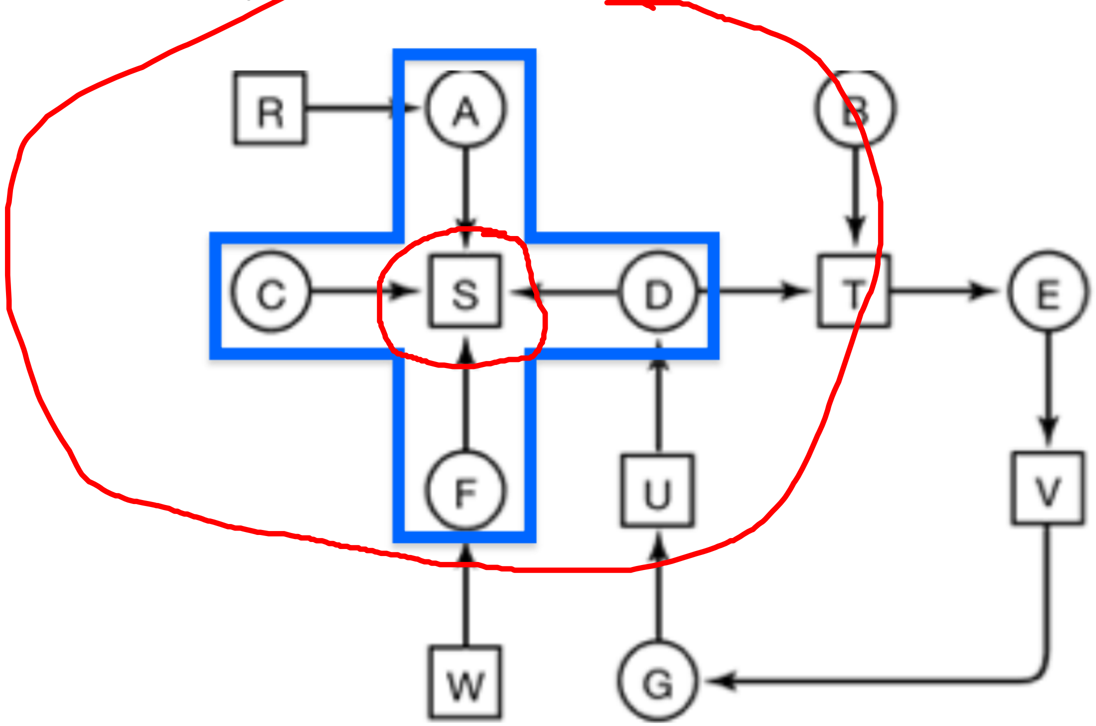
Deadlock Detection

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



Dead locked

- 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

- 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

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

Current Allocation Matrix

Who has what!!

Request matrix

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

P1 ?
P2 ?
P3 ?

X P1
X P2

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 with multiple resources of each type

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \end{array}$$

$$E = (4 \quad 2 \quad 3 \quad 1)$$

Existing Resource Vector

$$\begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \end{array}$$

$$A = (2 \quad 1 \quad 0 \quad 0)$$

Resources Available

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

Current allocation matrix

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

Current Allocation Matrix

Request matrix

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

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 can be satisfied

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

- Deadlock detection with multiple resources of each type

$$
\begin{array}{cccc}
\text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms}
\end{array}
$$

$$E = (4 \quad 2 \quad 3 \quad 1)$$

Existing Resource Vector

$$
\begin{array}{cccc}
\text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms}
\end{array}
$$

$$A = (2 \quad 1 \quad 0 \quad 0)$$

Resources Available

Current allocation matrix

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

Current Allocation Matrix

Request matrix

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

Request Matrix

P_3 runs and its allocation is (2, 2, 2, 0)
On completion it returns the available resources are A = (4 2 2 1)
Either P_1 or P_2 can now run.
NO Deadlock!!!

- Deadlock detection with multiple resources of each type

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

Existing Resource Vector

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

Resources Available

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

P_1
P_2
P_3

Current allocation matrix

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

Current Allocation Matrix

Request matrix

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

Request Matrix

P_1 cannot be satisfied

P_2 cannot be satisfied

P_3 cannot be satisfied

deadlock

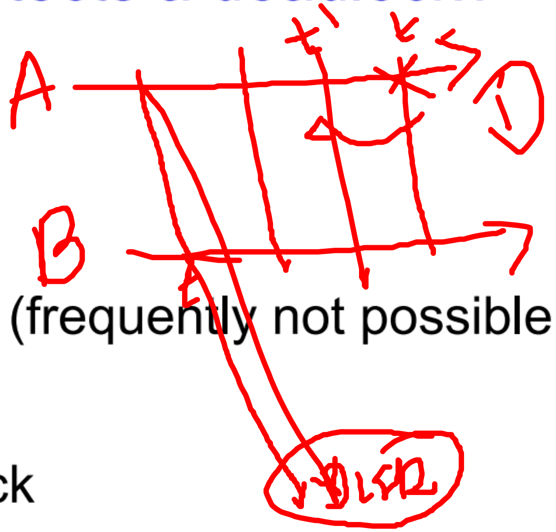
Process P_i holds C_i resources and requests R_i resources, where $i = 1$ to 3
Deadlock detected as none of the requests can be satisfied

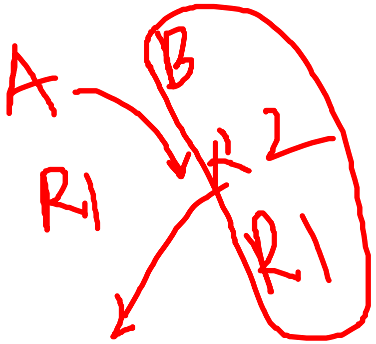
Deadlock Recovery

checkpointing

What should the OS do when it detects a deadlock?

- Raise an alarm
  - Tell users and administrator
- Preemption
  - Take away a resource temporarily (frequently not possible)
- Rollback
  - Checkpoint states and then rollback
- Kill low priority process
  - Keep killing processes until deadlock is broken
  - (or reset the entire system)



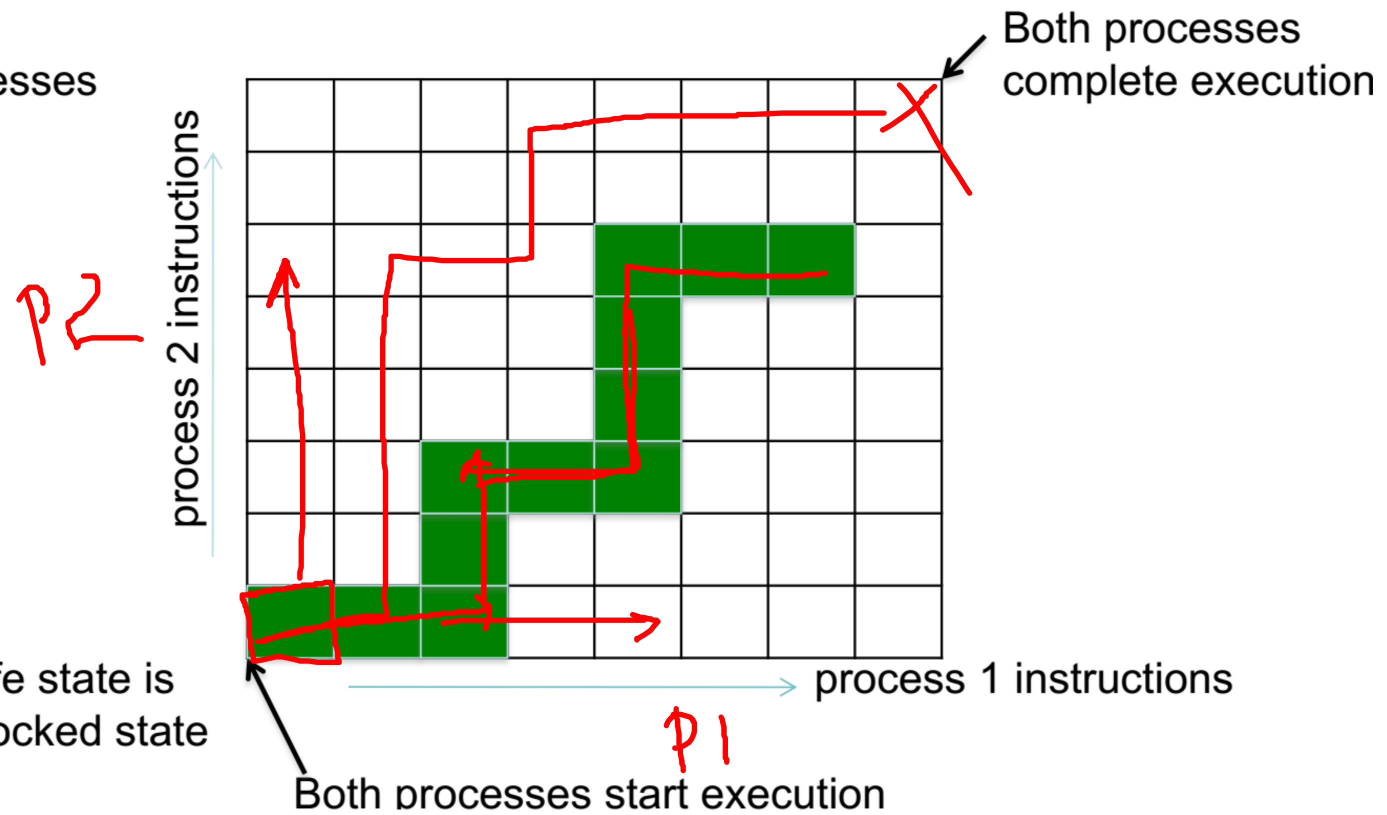


OSTRICH

Deadlock Avoidance

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

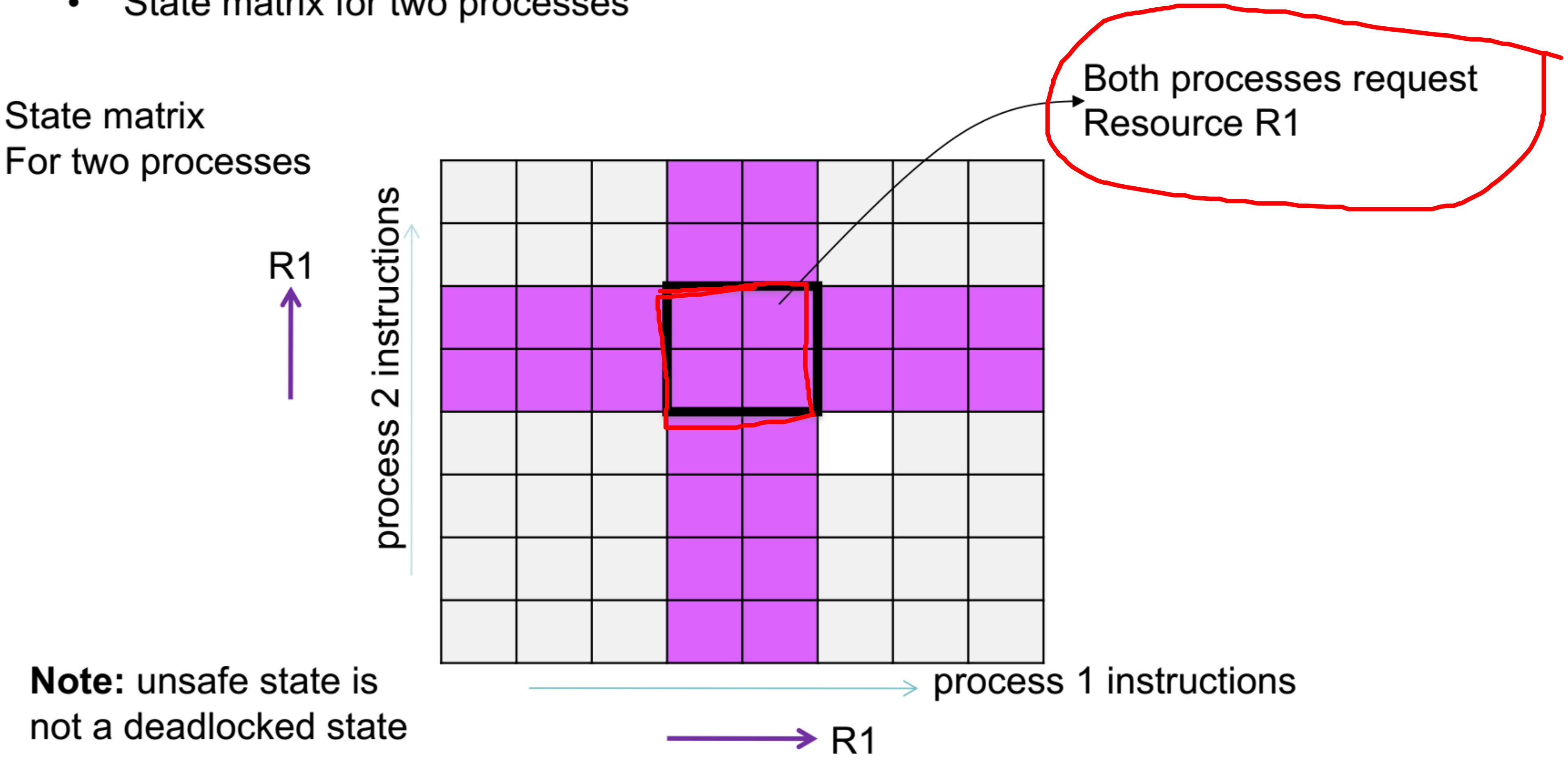
State matrix
For two processes



Note: unsafe state is not a deadlocked state

Deadlock Avoidance

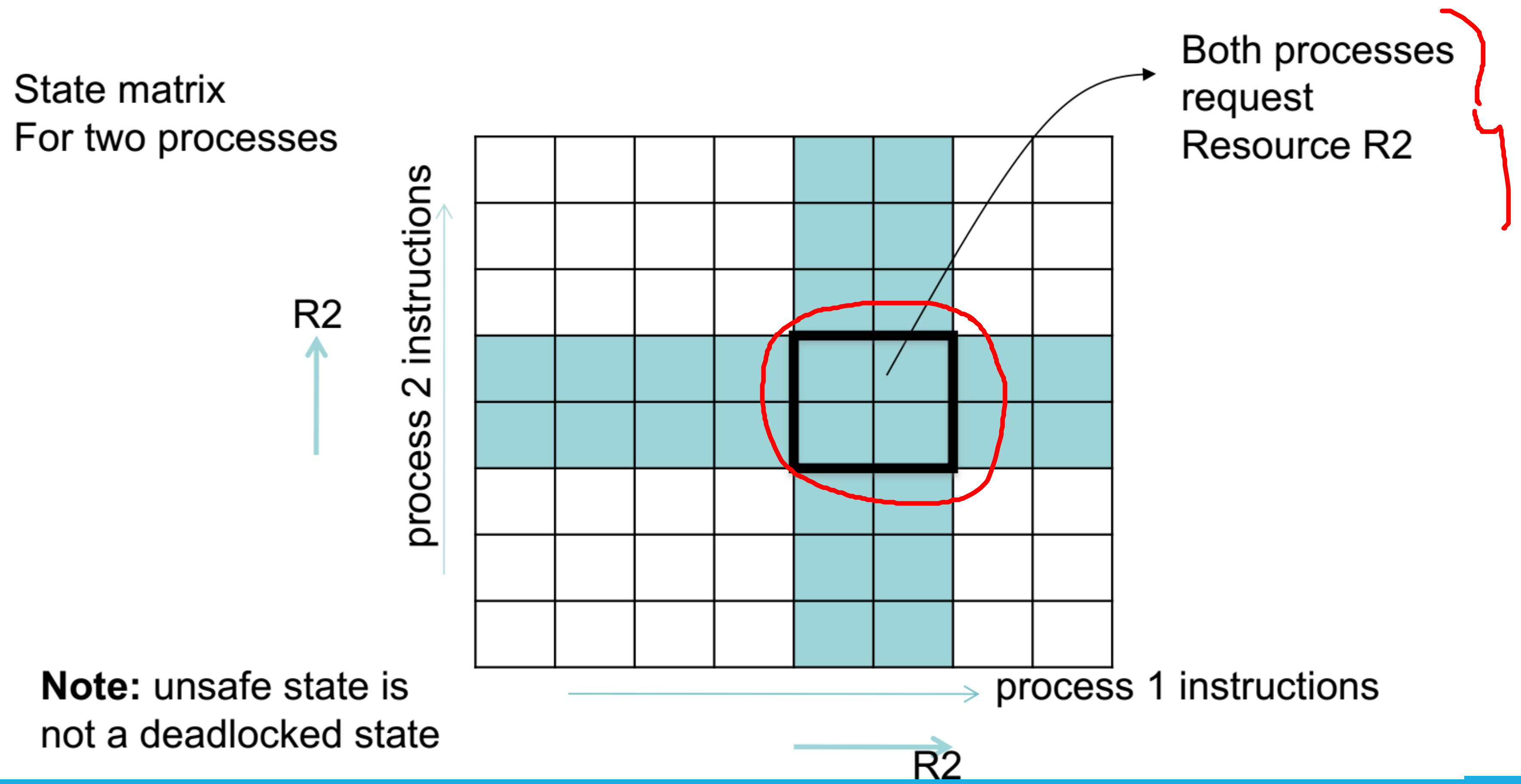
- State matrix for two processes



Note: unsafe state is not a deadlocked state

Deadlock Avoidance

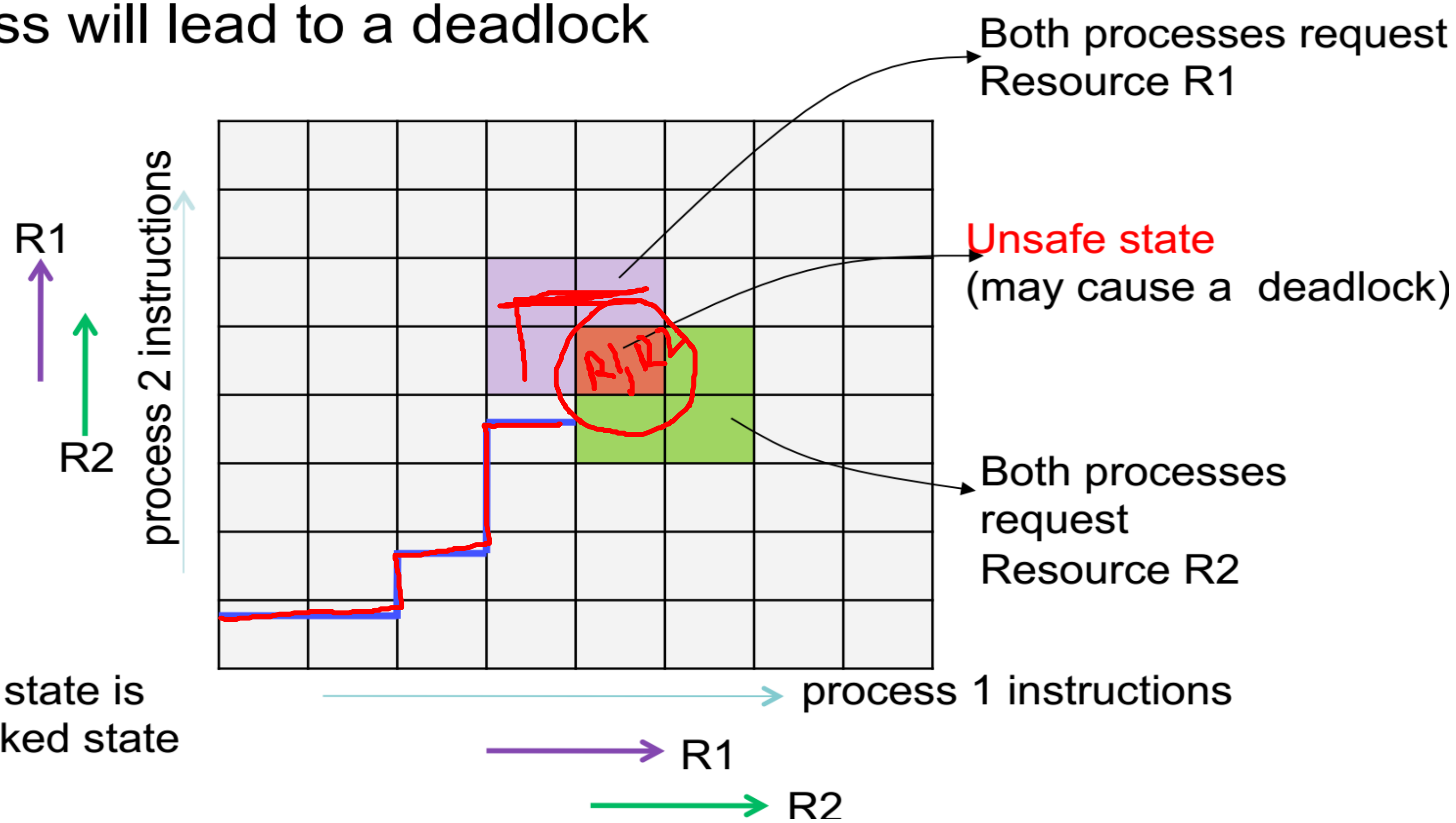
- State matrix for two processes



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

Is there an algorithm that can always avoid deadlocks by conservatively make the right choice.

- Ensures system never reaches an unsafe state

- Safe state : A state is said to be safe, if there is some scheduling order in which every process can run to completion even if all of them suddenly requests their maximum number of resources immediately
- An unsafe state does not have to lead to a deadlock; *it* ***could*** *lead to a deadlock*

Example with a Banker

5

- Consider a banker with 4 clients (P_1, P_2, P_3, P_4).
  - Each client has certain credit limits (totaling 20 units)
  - The banker knows that max credits will not be used at once, so he keeps only 10 units

	Has	Max
A	3	9
B	2	4
C	2	7

Total : 10 units

free : 3 units

  - Clients declare maximum credits in advance. The banker can allocate credits provided no unsafe state is reached.

Safe State

10 units

6 5 7

	Has	Max
A	3	9
B	2	4
C	2	7

free : 3 units

Allocate 2 units to B

	Has	Max
A	3	9
B	4	4
C	2	7

free : 1 units

B completes

	Has	Max
A	3	9
B	0	-
C	2	7

free : 5 units

5

6 5

Allocate 5 to C

	Has	Max
A	3	9
B	0	-
C	7	7

free : 0 units

C completes

	Has	Max
A	3	9
B	0	-
C	0	-

free : 7 units

6

Allocate 6 units to A

	Has	Max
A	9	9
B	0	-
C	0	-

free : 0 units

This is a safe state because there is some scheduling order in which every process executes

Unsafe State

10

	Has	Max
A	4	9
B	2	4
C	2	7

free : 2 units

Allocate 2 units to B

	Has	Max
A	4	9
B	4	4
C	2	7

free : 0 units

B completes

	Has	Max
A	4	9
B	0	-
C	2	7

free : 4 units

< 5

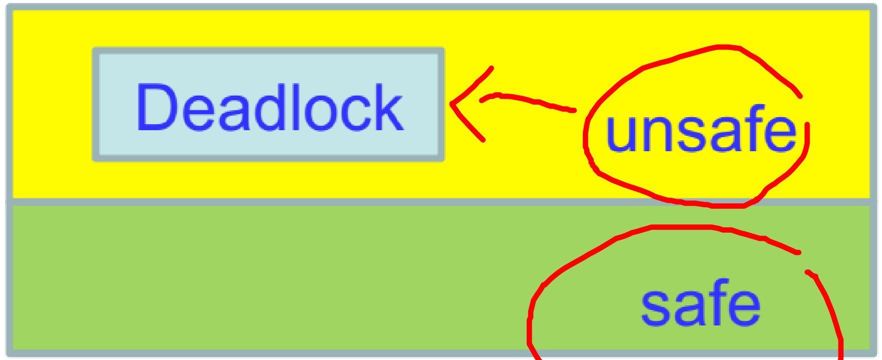
5

This is an unsafe state because there exists NO scheduling order in which every process executes

Banker's Algorithm [Single Resource]

When a request occurs

- If(**is_system_in_a_safe_state**)
  - Grant request
- else
  - postpone until later



Please read Banker's Algorithm with multiple resources from Modern Operating Systems, Tanenbaum

Deadlock Prevention

- Deadlock avoidance not practical, need to know maximum requests of a process
- Deadlock prevention
  - Prevent at-least one of the 4 conditions

1. **Mutual Exclusion**
2. **Hold and wait**
3. **No preemption**
4. **Circular wait**

Prevention

1. **Preventing Mutual Exclusion**
   - ~~Not feasible in practice~~
   - But OS can ensure that resources are optimally allocated
2. **Hold and wait**
   - One way is to achieve this is to require all processes to request resources before starting execution
     - May not lead to optimal usage
     - May not be feasible to know resource requirements
3. **No preemption**
   - Pre-empt the resources, such as by virtualization of resources (eg. Printer spools)
4. **Circular wait**
   - One way, process holding a resource cannot hold a resource and request for another one
   - Ordering requests in a sequential / hierarchical order.

R1

R2

Hierarchical Ordering of Resources

- Group resources into levels
  (i.e. prioritize resources numerically)
- A process may only request resources at higher levels than any resource it currently holds
- Resource may be released in any order
- eg.

L — 1
P — 3

L H

  – Semaphore s1, s2, s3 (with priorities in increasing order)
    down(S1); down(S2); down(S3) ; → allowed
    down(S1); down(S3); down(S2); →not allowed