Chapter 6

Deadlock

(Part 2)

Deadlock Avoidance

Detection – "optimistic" approach

- **Allocate resources**
- **"Break" system to fix it**

Deadlock Avoidance

Detection – "optimistic" approach

Allocate resources

"Break" system to fix it

Avoidance – "pessimistic" approach

Don't allocate resource if it may lead to deadlock

If a process requests a resource...

Make it wait until you are sure it's OK.

Deadlock Avoidance

Detection – "optimistic" approach

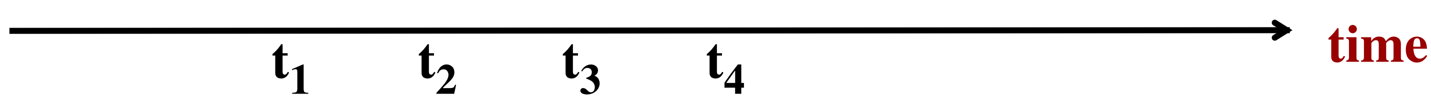
- **Allocate resources**
- **"Break" system to fix it**

Avoidance – "pessimistic" approach

- **Don't allocate resource if it may lead to deadlock**
- **If a process requests a resource...**
  - **Make it wait until you are sure it's OK.**

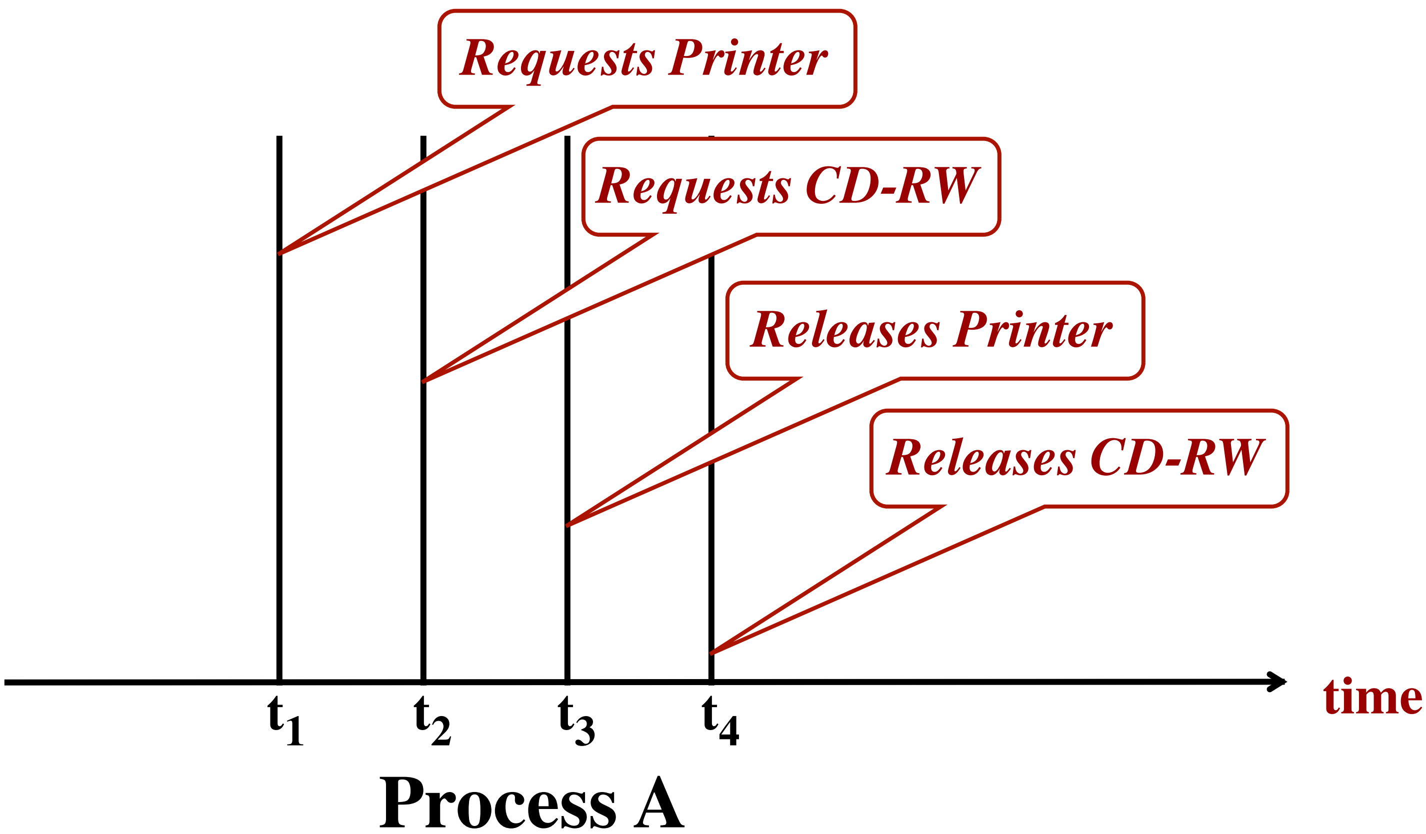
Which one to use depends upon the application!

Process-Resource Trajectories

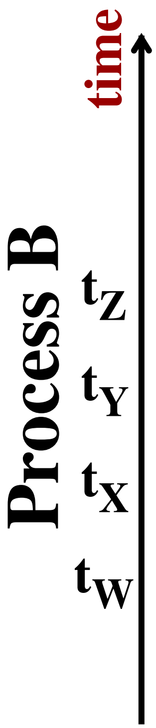


Process A

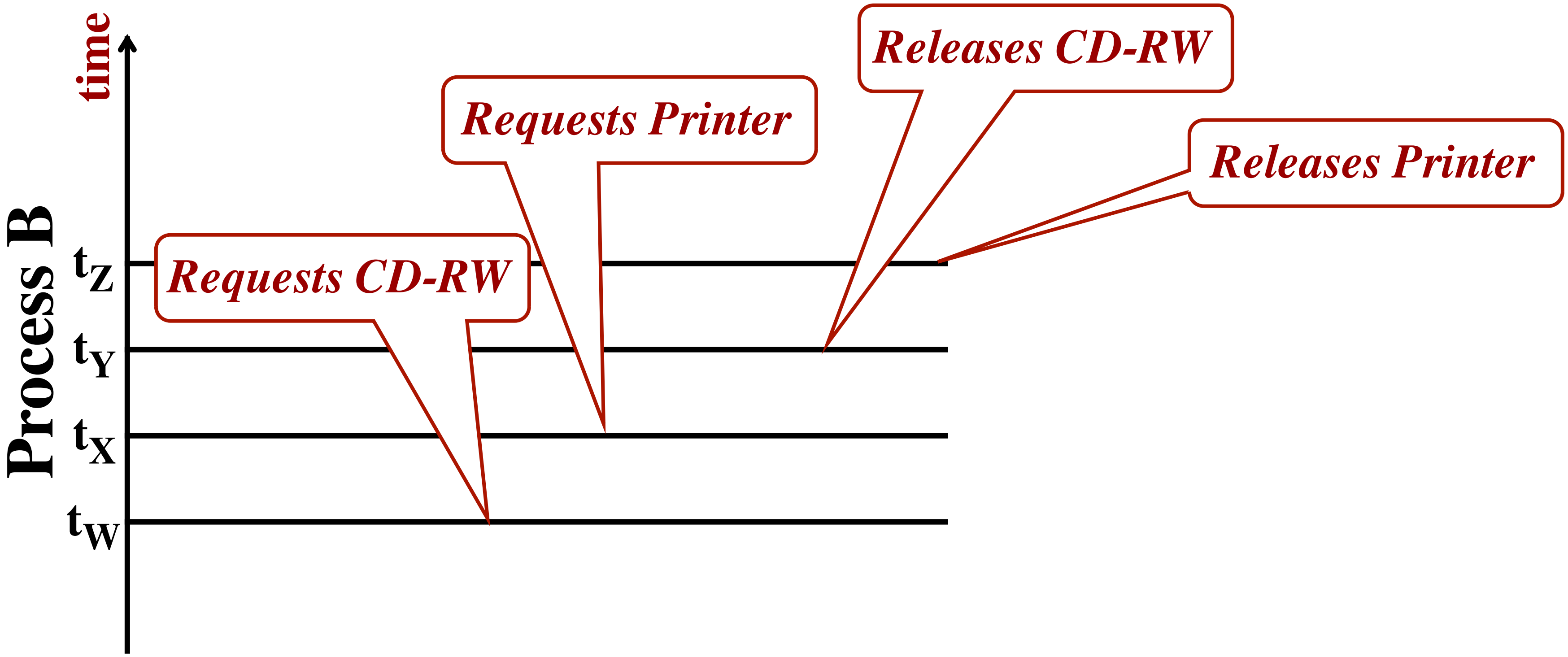
Process-Resource Trajectories



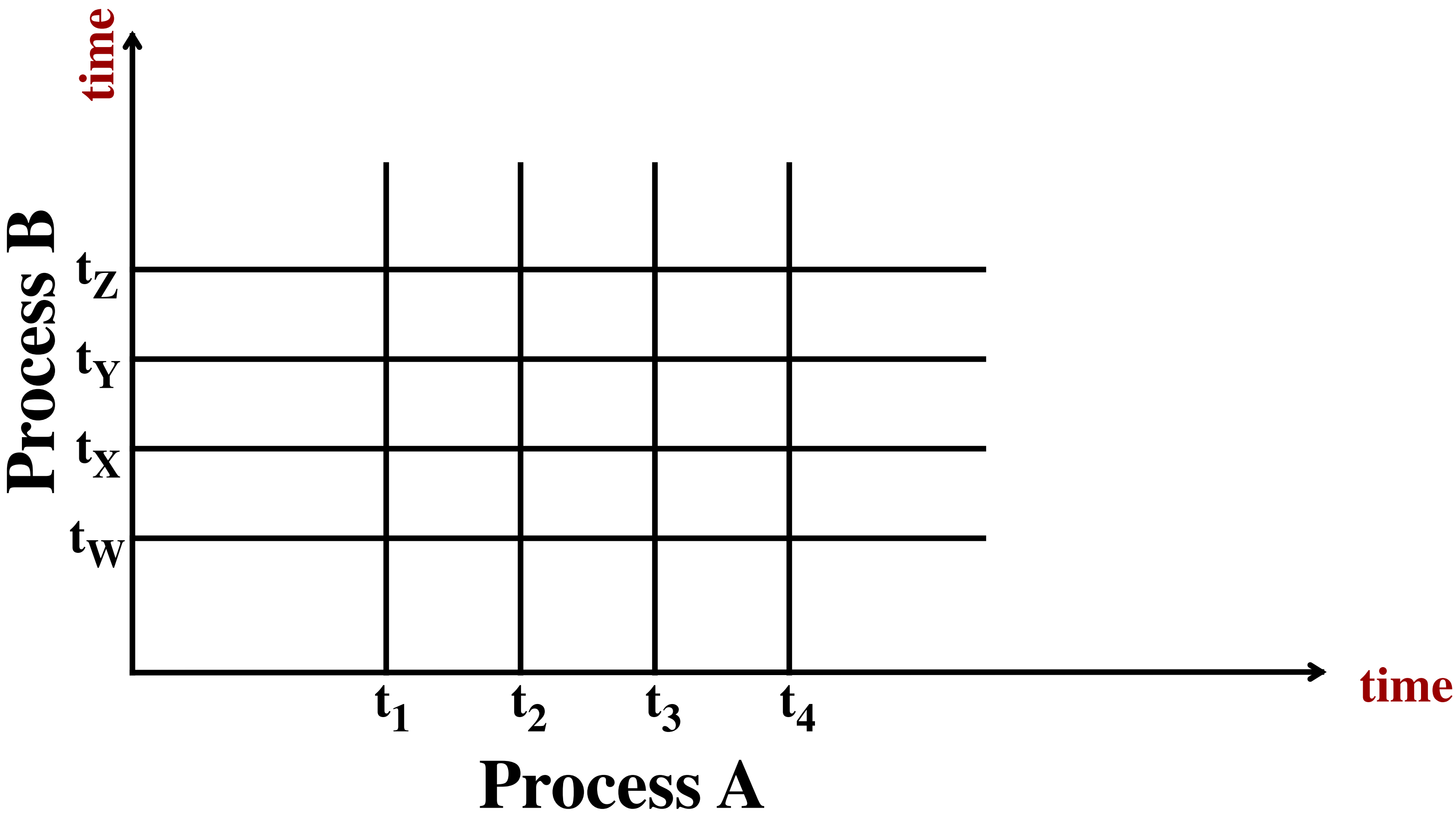
Process-Resource Trajectories



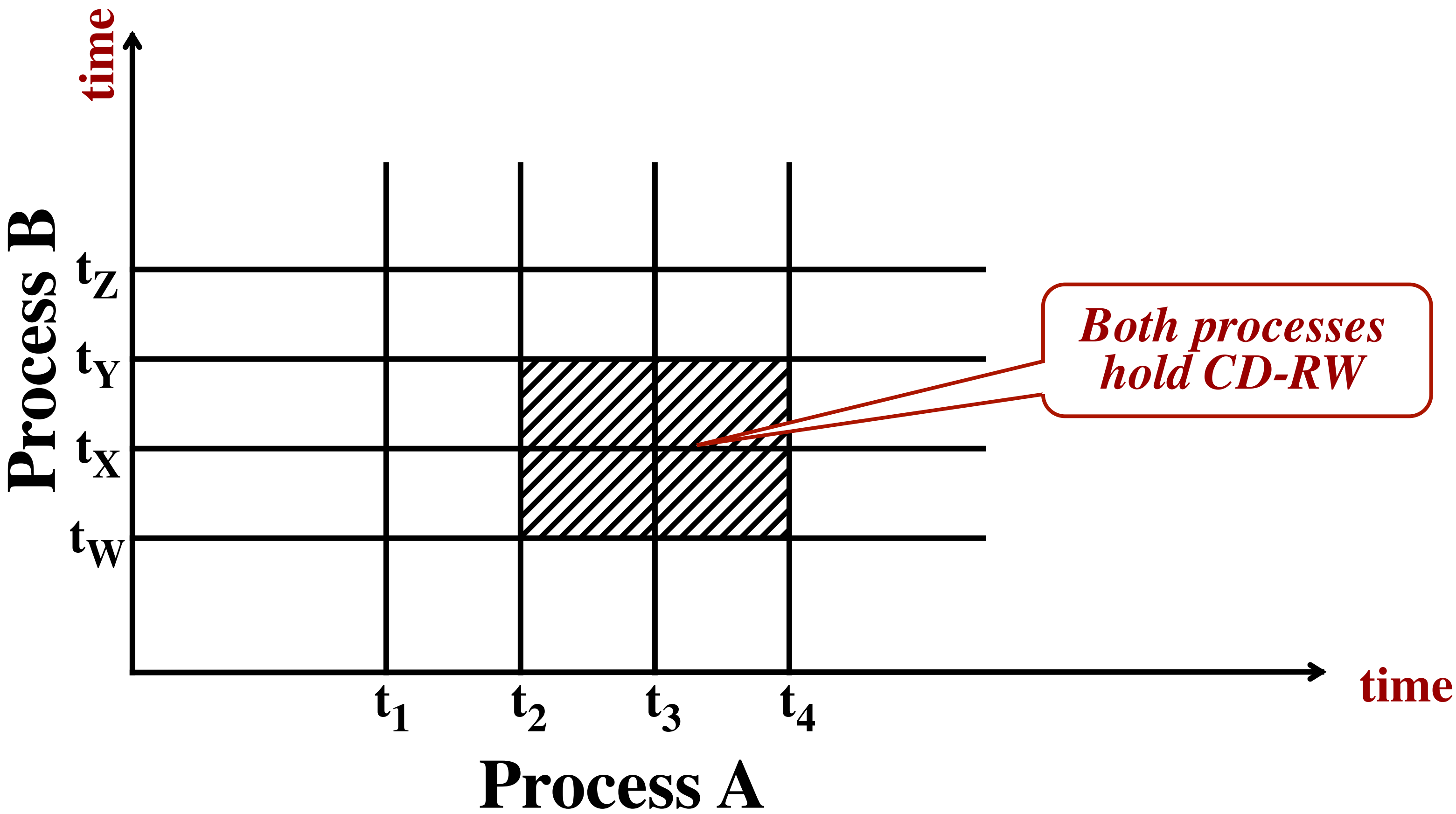
Process-Resource Trajectories



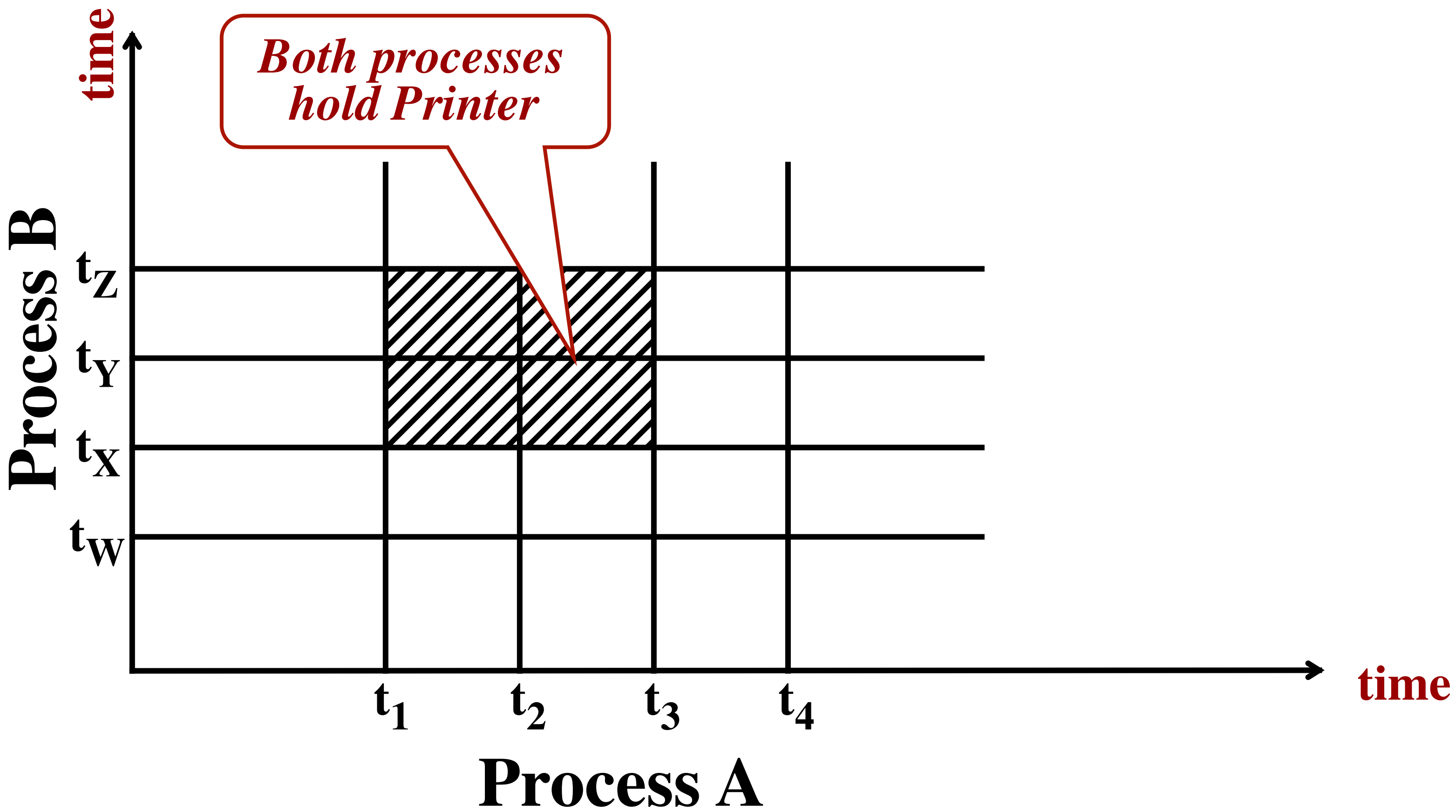
Process-Resource Trajectories



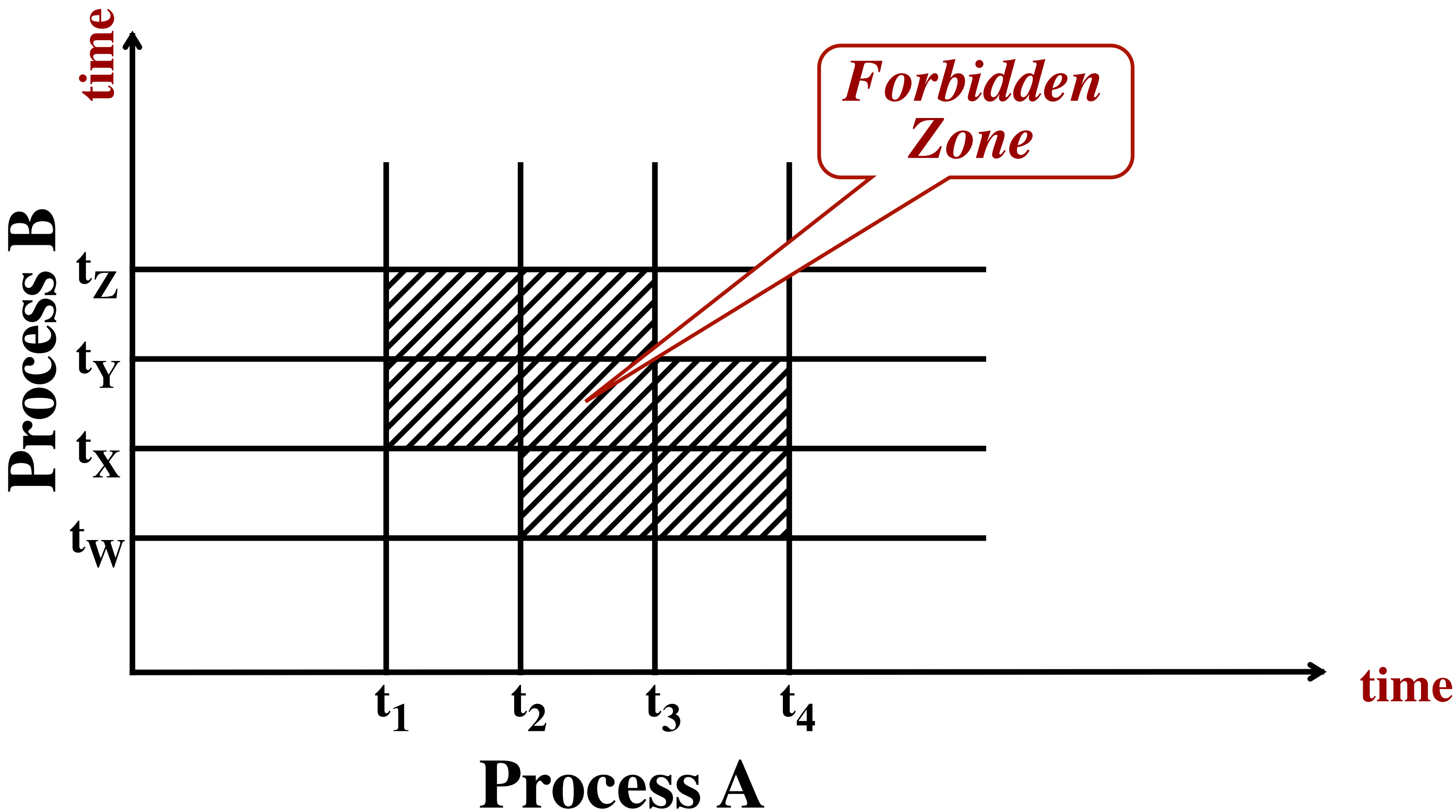
Process-Resource Trajectories



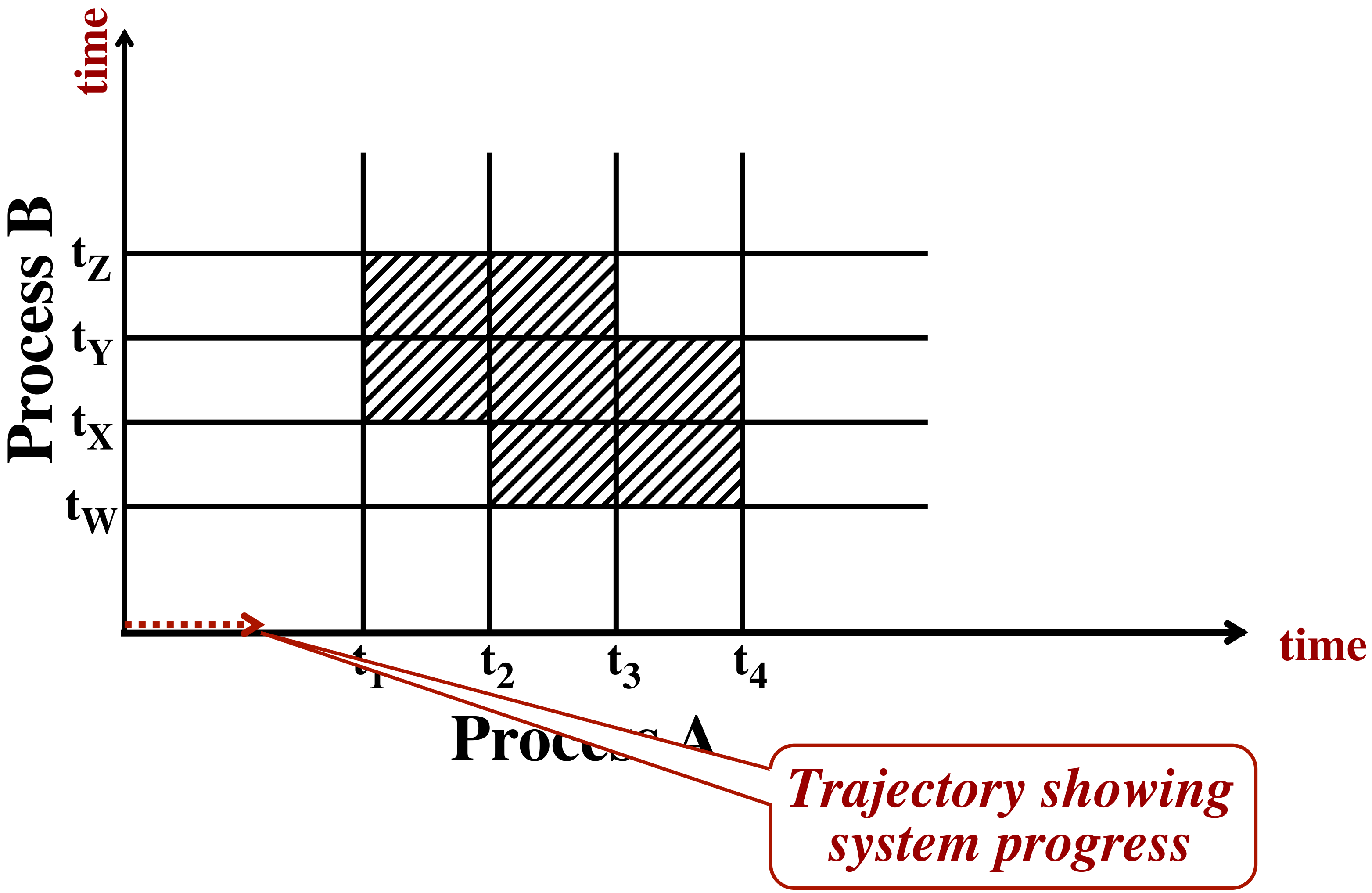
Process-Resource Trajectories



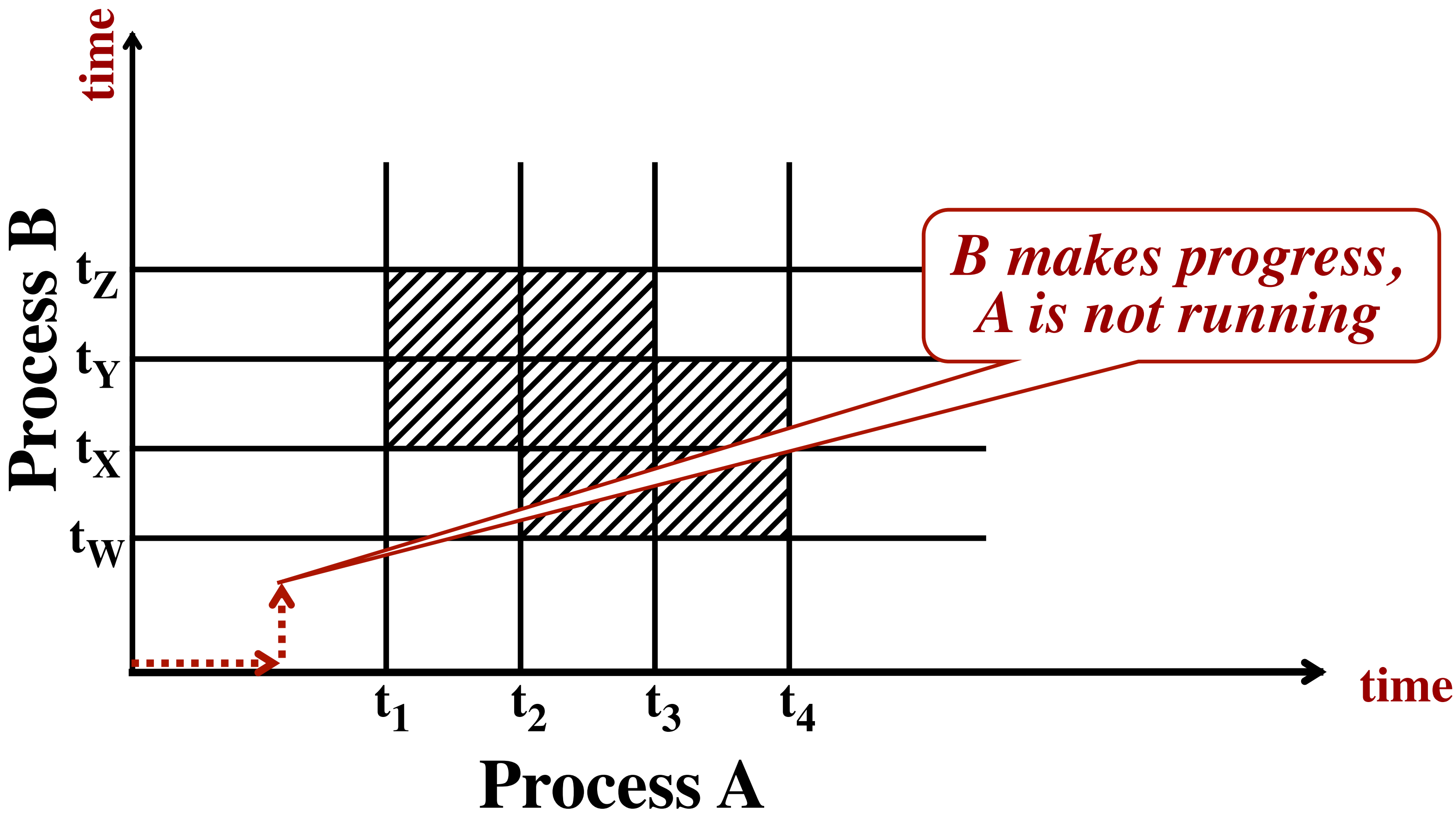
Process-Resource Trajectories



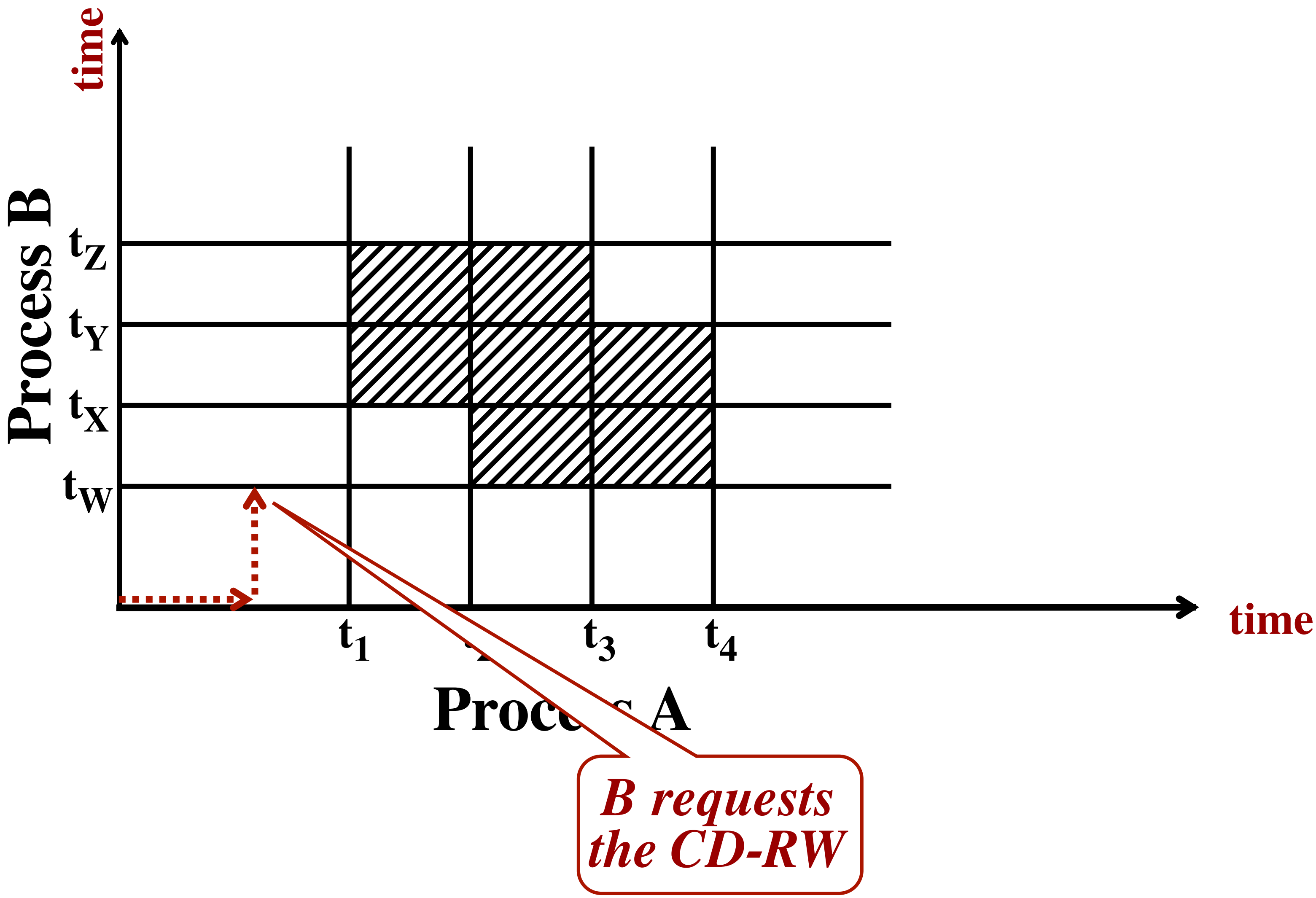
Process-Resource Trajectories



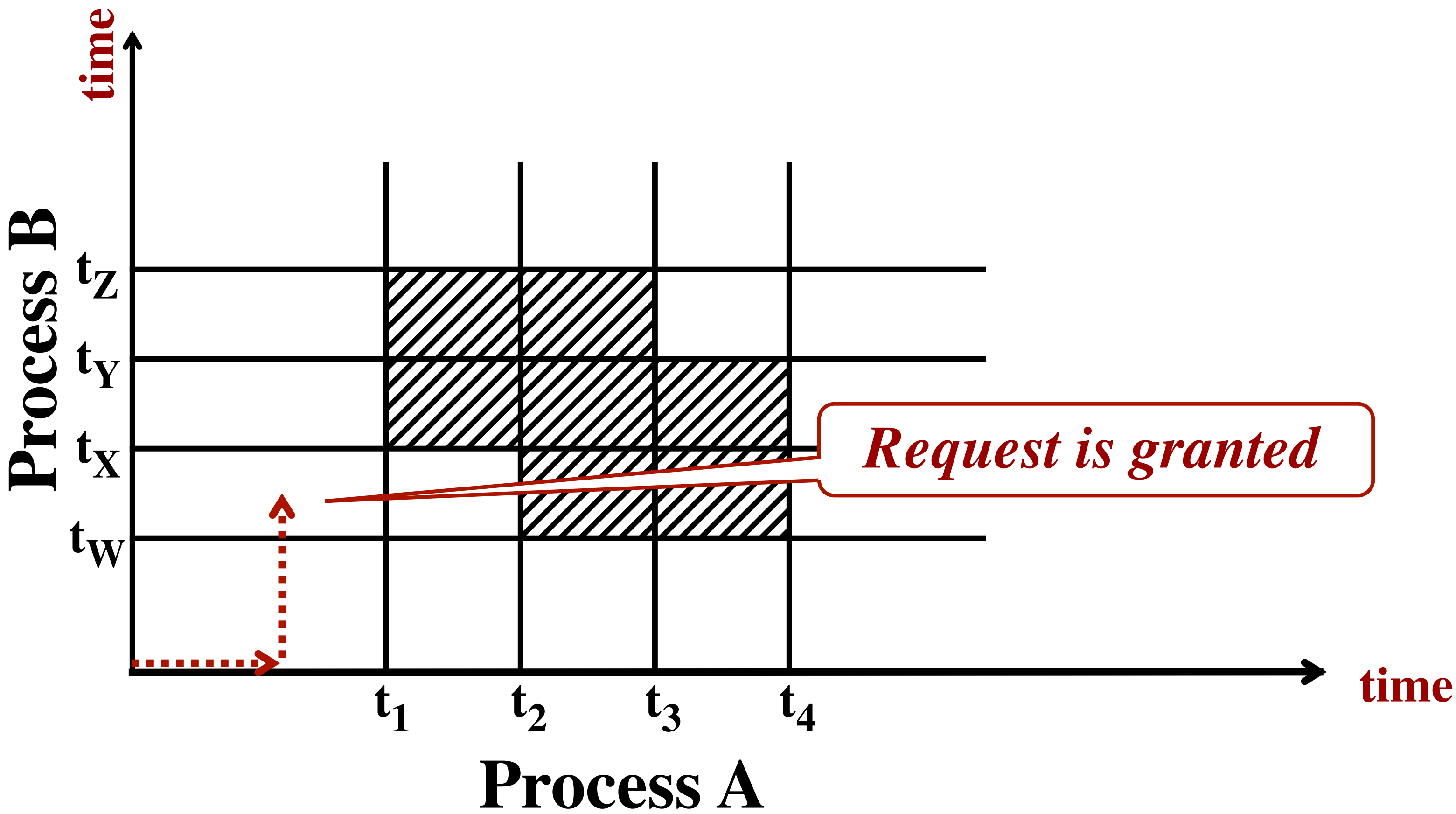
Process-Resource Trajectories



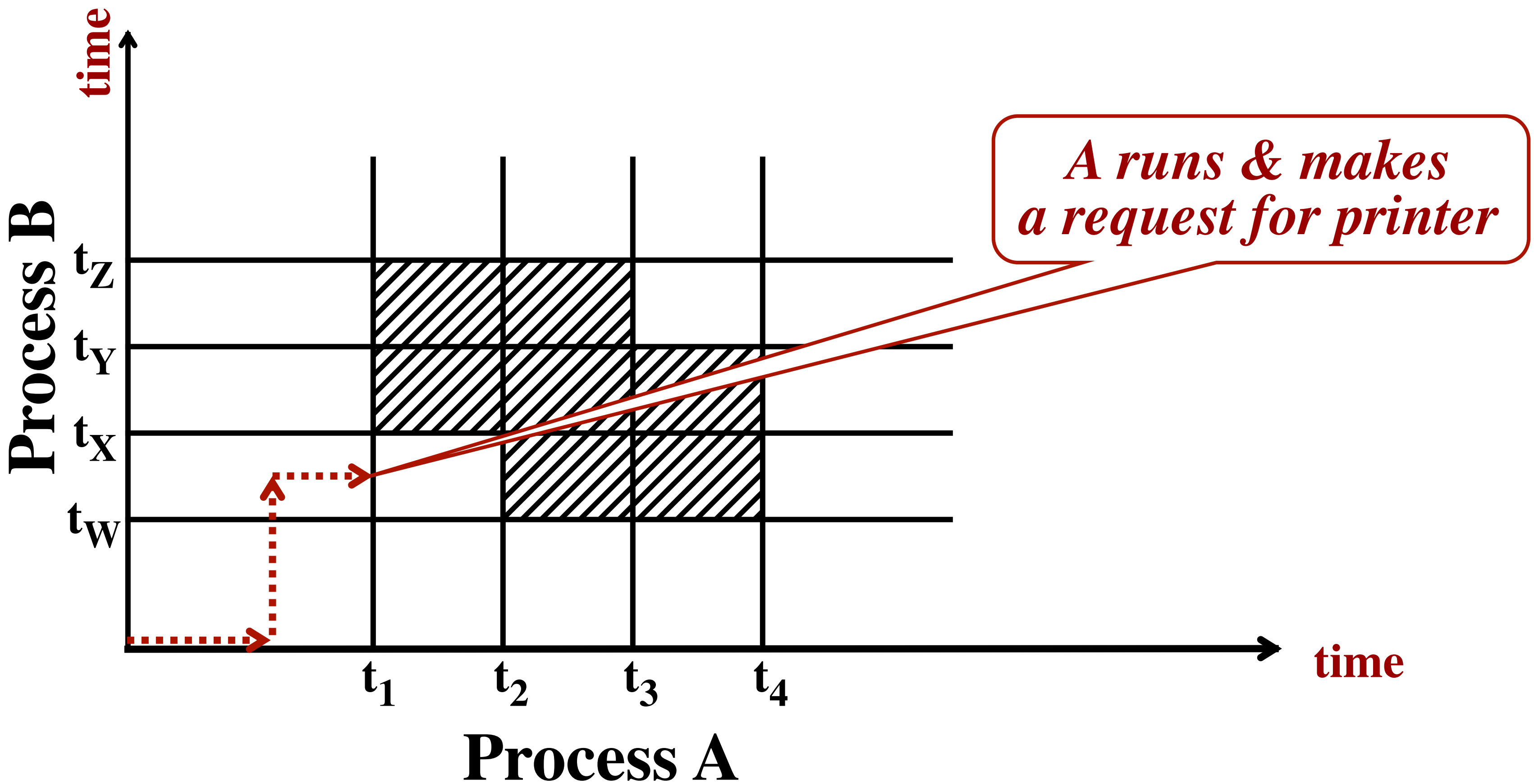
Process-Resource Trajectories



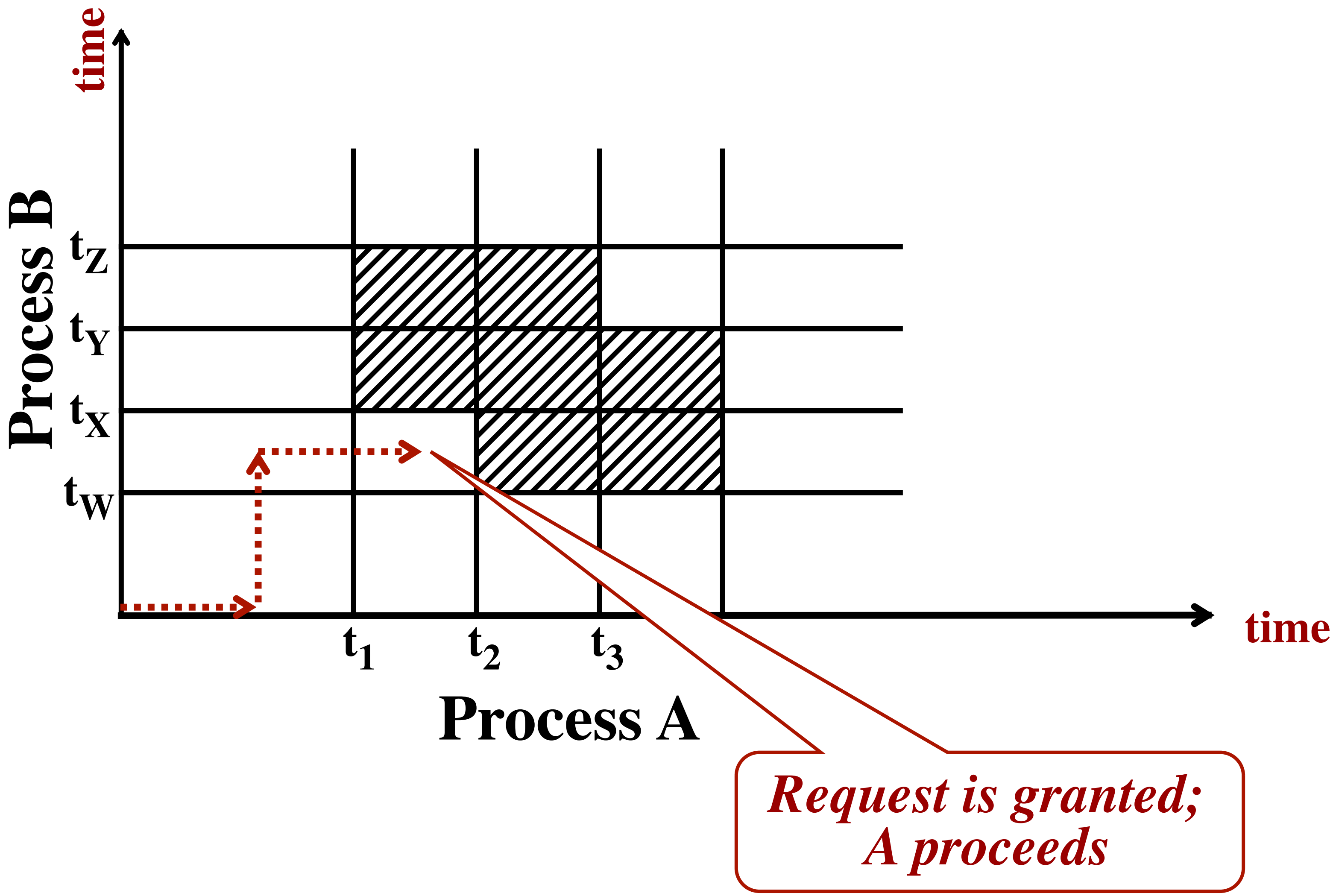
Process-Resource Trajectories



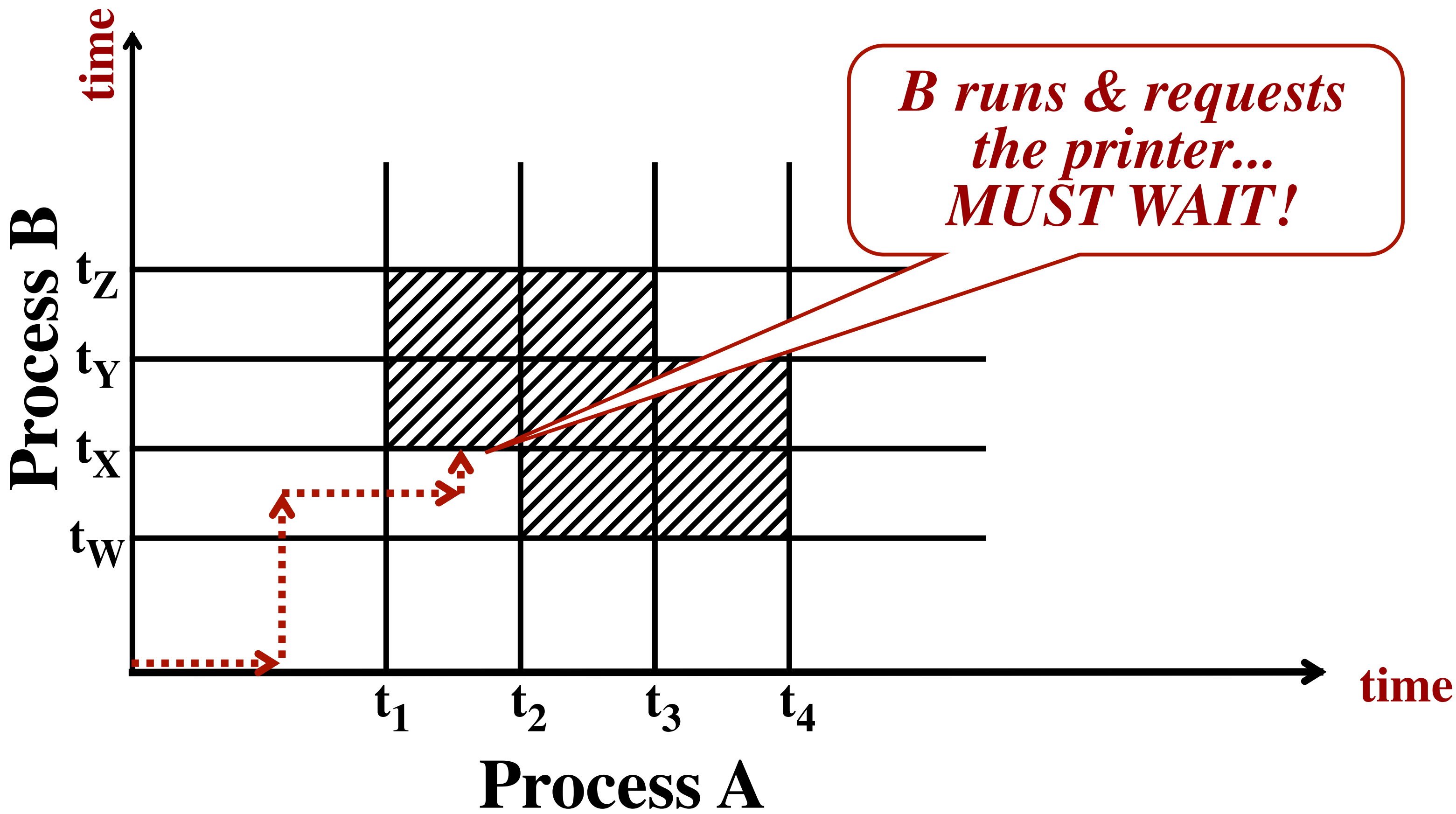
Process-Resource Trajectories



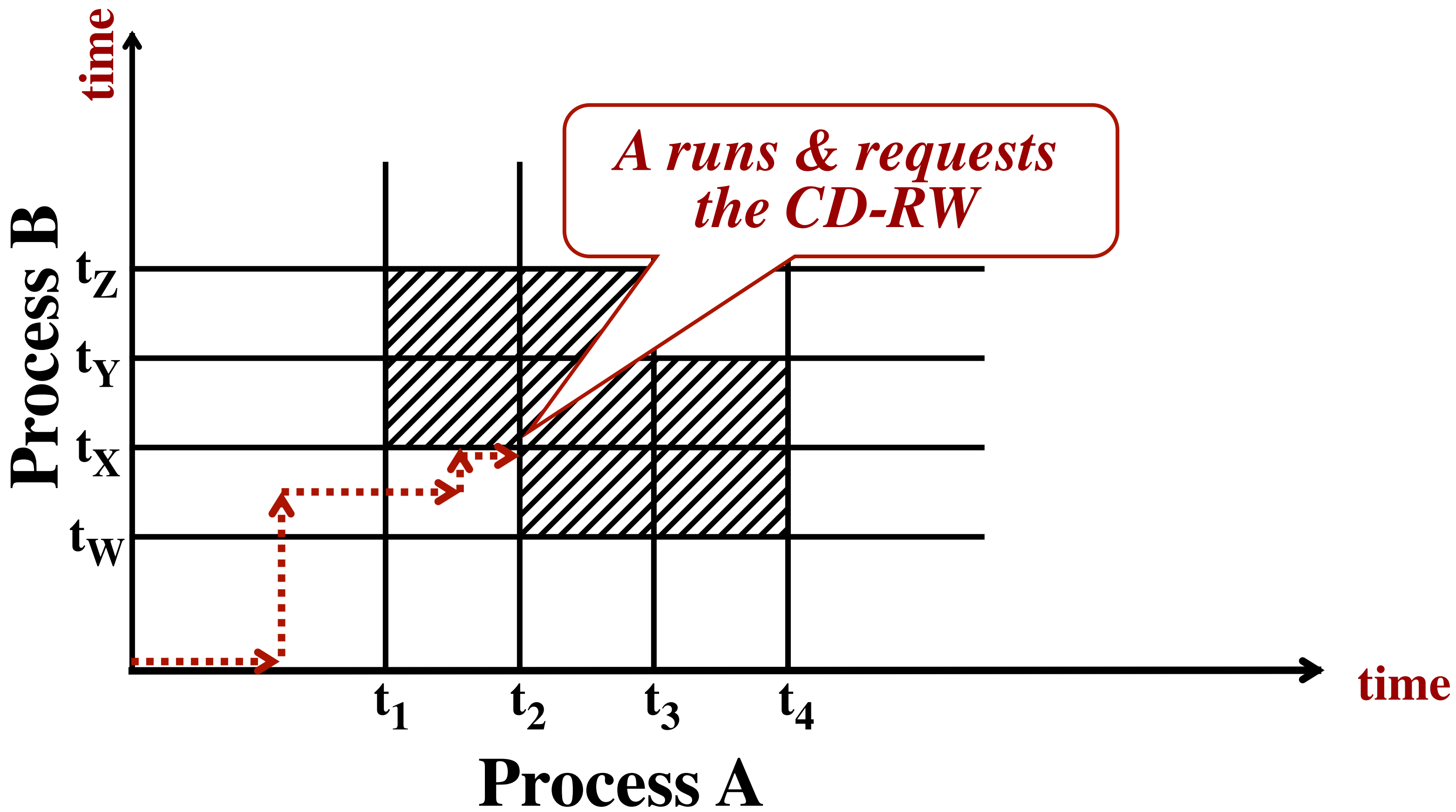
Process-Resource Trajectories



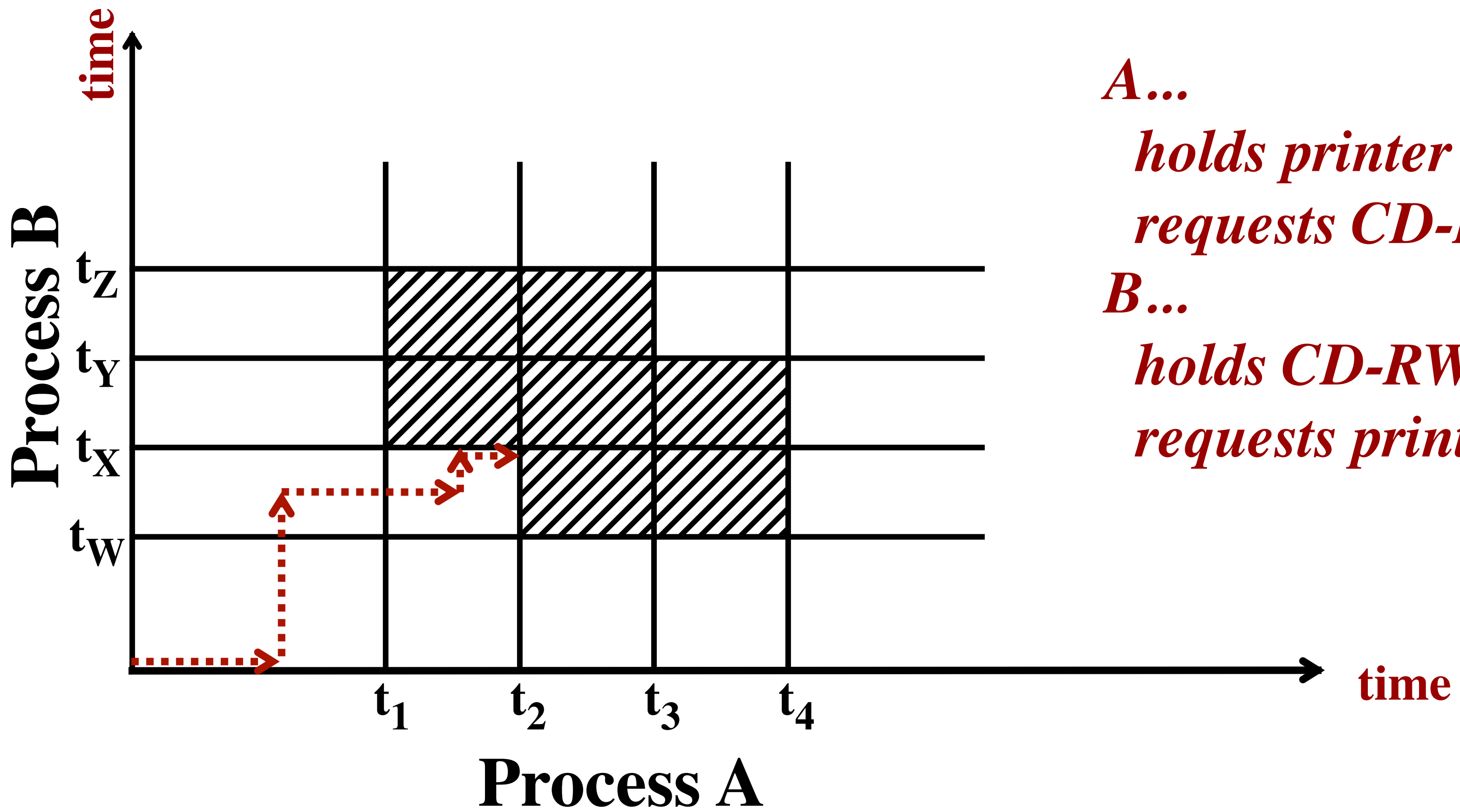
Process-Resource Trajectories



Process-Resource Trajectories



Process-Resource Trajectories



A…

holds printer

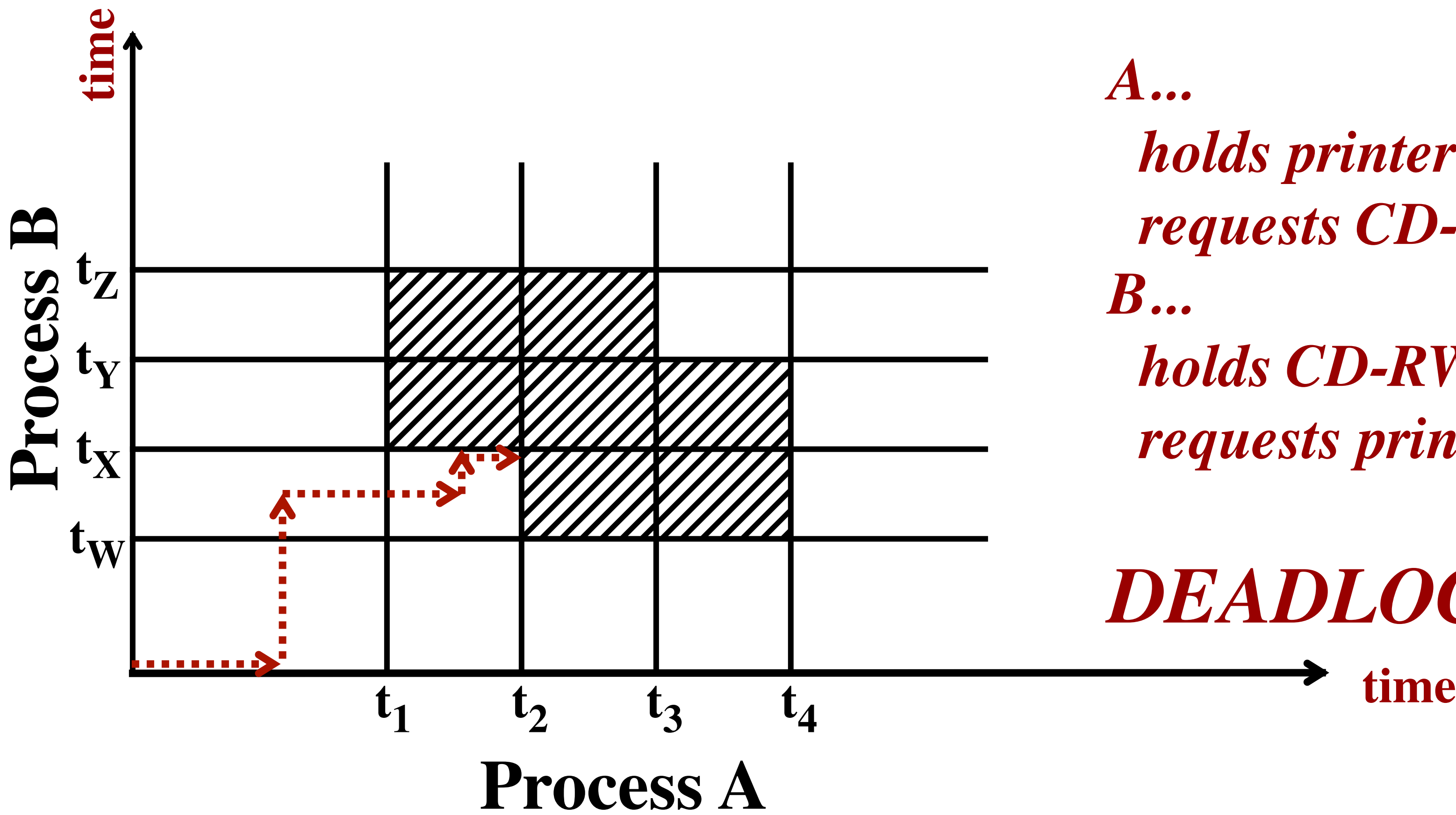
requests CD-RW

B…

holds CD-RW

requests printer

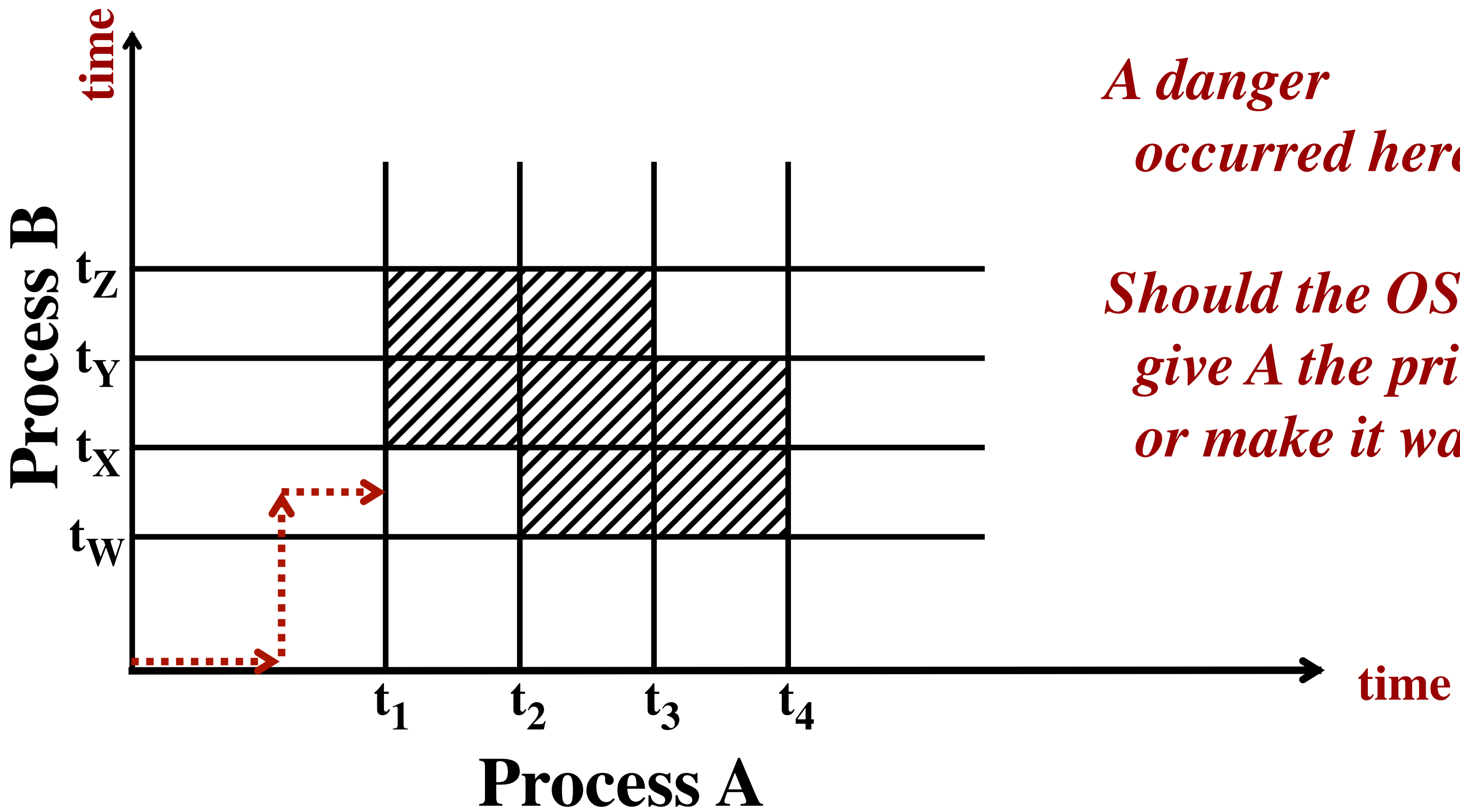
Process-Resource Trajectories



A…
holds printer
requests CD-RW
B…
holds CD-RW
requests printer

DEADLOCK!

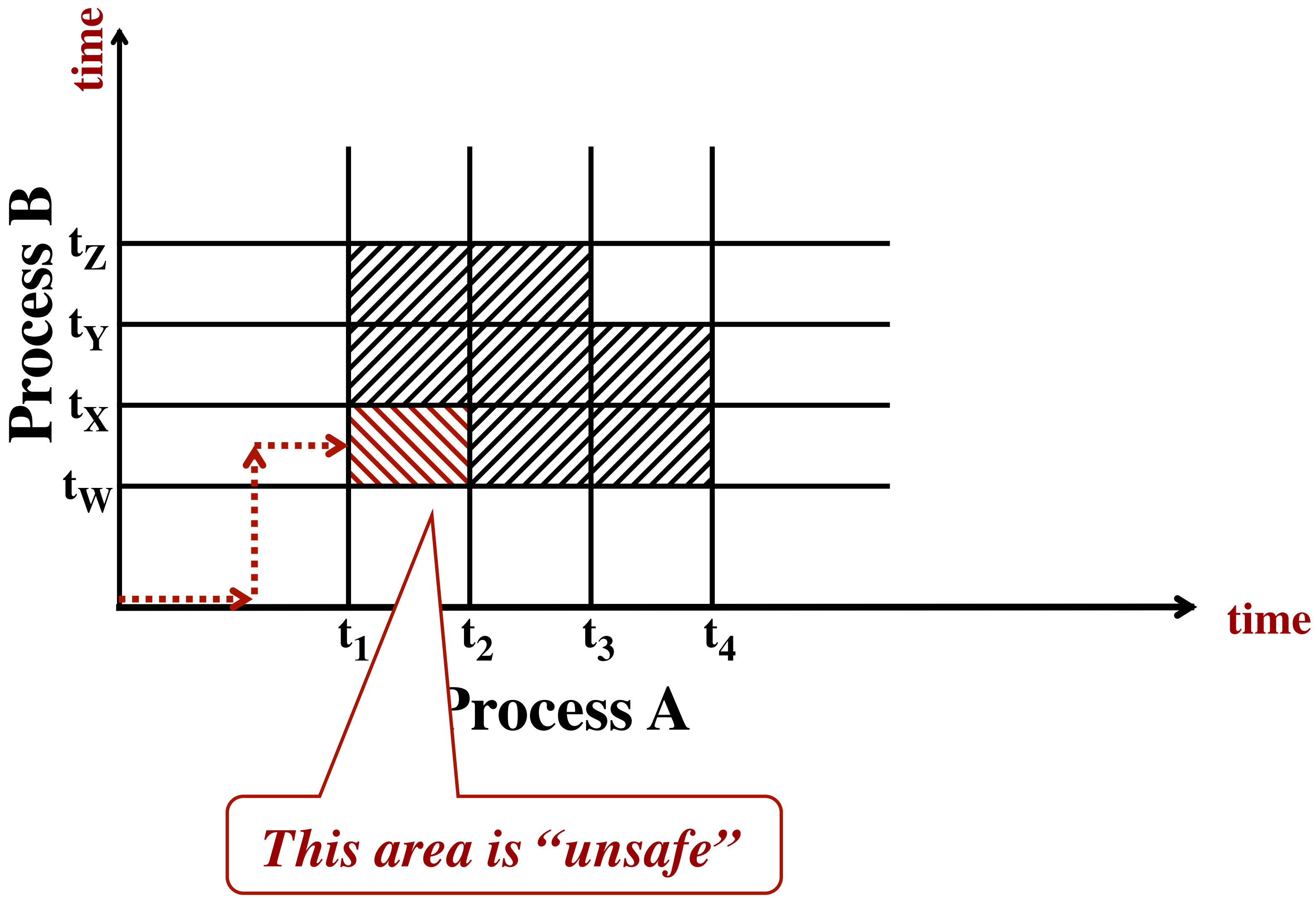
Process-Resource Trajectories



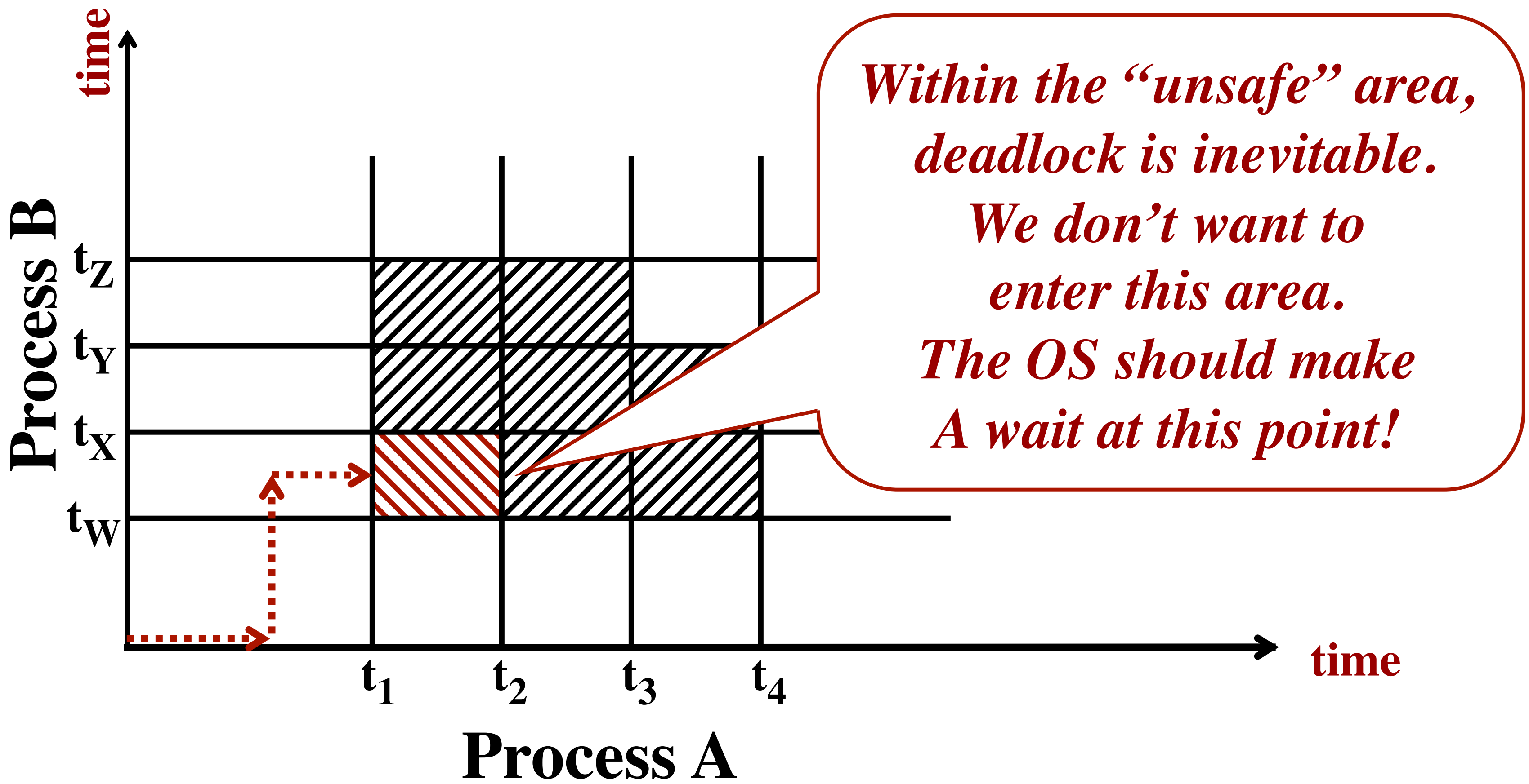
A danger occurred here.

Should the OS give A the printer, or make it wait???

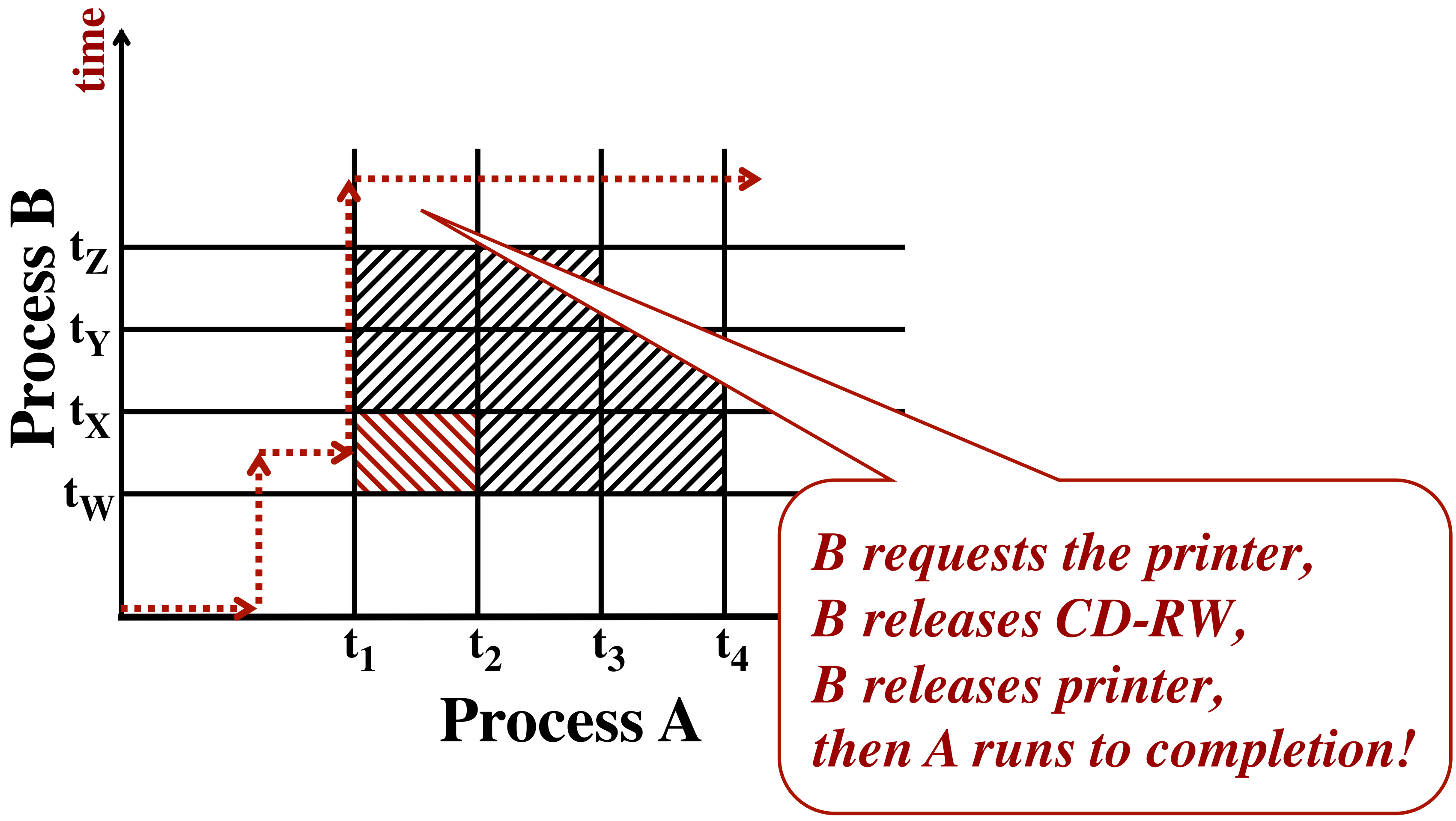
Process-Resource Trajectories



Process-Resource Trajectories



Process-Resource Trajectories



Safe states

The current state:

"which processes hold which resources"

A "safe" state:

- **No deadlock,** ***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 units immediately.**

The Banker's Algorithm:

Goal**: Avoid unsafe states!!!**

When a process requests more units, should the system grant the request or make it wait?

The Banker's Algorithm

Assumptions:

- **Only one type of resource, with multiple units.**
- **Processes declare their maximum potential resource needs ahead of time.**

When a process requests more units should the system make it wait to ensure safety?

Example: One resource type with 10 units

	Has	Max	
A	3	9	**6**
B	2	4	**2**
C	2	7	**5**

Free: 3

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

How many more this process might need

Unsafe states

10 total resource units

	Has	Max	
A	3	9	6
B	2	4	2
C	2	7	5

Free: 3

	Has	Max	
A	4	9	5
B	2	4	2
C	2	7	5

Free: 2

	Has	Max	
A	4	9	5
B	4	4	0
C	2	7	5

Free: 0

	Has	Max	
A	4	9	5
B	—	—	0
C	2	7	5

Free: 4

Unsafe!

Avoidance Modeling - Multiple Resource Types

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

Note: These are the max. possible requests, which we assume are known ahead of time

Banker's Algorithm for Multiple Resources

1) Look for a row, *R*, whose unmet resource needs are all smaller than or equal to A. If such row exists, all the possible needs for this process could be met right now.

2) Assume the process of the row chosen requests all the resources that it needs (which is guaranteed to be possible) and the terminates. Mark that process as "terminated" and add all its resources back to the "A" vector.

Repeat steps 1 and 2, until either all process are marked terminated, in which case the initial state was safe. If some processes remain, then initial state was UNSAFE!

Avoidance algorithm

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

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

Current allocation matrix

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

Max request matrix

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

Avoidance algorithm

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

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

Current allocation matrix

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

Max request matrix

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

Avoidance algorithm

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,4 & 2 & 3 & 1\,) \end{array} \qquad A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (\,2 & 1 & 0 & 0\,) \end{array}$$

Current allocation matrix

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

Max request matrix

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

Avoidance algorithm

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

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

Current allocation matrix

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

Max request matrix

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

Avoidance algorithm

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

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

Current allocation matrix

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

Max request matrix

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

Avoidance algorithm

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

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

2 2 2 0

4 2 2 1

Current allocation matrix

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

Max request matrix

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

Deadlock Avoidance

Deadlock avoidance is usually impractical because you don’t know in advance what resources a process will need!

Deadlock Avoidance

Alternative approach: "deadlock prevention"

Prevent the situation in which deadlock *might* occur for all time!

Attack one of the four conditions that are necessary for deadlock to be possible.

Four conditions necessary for deadlock:

- **Mutual exclusion condition**
- **Hold and wait condition**
- **No preemption condition**
- **Circular wait condition**

Attacking the conditions

Attacking mutual exclusion?

- **Not really an option for some resource types**
- **May work for other types**

Attacking no preemption?

- **Not really an option for some resource types**
- **May work for other types**

Attacking the conditions

Attacking hold and wait?

- **Require processes to request all resources before they begin!**
- **Process must know ahead of time**
- **Process must tell system its "max potential needs"**

If a process decides it wants more than its initial declared needs, it must...

- **Release all resources**
- **Give the system a new "max potential needs"**
- **Resume execution**

Issues:

- **Under-allocation of resources**
- **Resource needs not known in advance**

Attacking the conditions

Attacking circular wait?

- **Number each of the resources**

- **Require each process to acquire lower numbered resources before higher numbered resources.**

Example:

1. **Printer**
2. **Scanner**
3. **CD-Rom**
4. **Plotter**

More precisely: **A process is not allowed to request a resource whose number is lower than the highest numbered resource it currently holds.**

Recall this Example of Deadlock

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

Assume that resources are ordered:

1. **Resource_1**
2. **Resource_2**
3. **...etc...**

Recall this Example of Deadlock

Thread A:

```
acquire (resource_1)
acquire (resource_2)
use resources 1 & 2
release (resource_2)
release (resource_1)
```

Thread B:

```
acquire (resource_2)
acquire (resource_1)
use resources 1 & 2
release (resource_1)
release (resource_2)
```

Assume that resources are ordered:

1. Resource_1
2. Resource_2
3. ...etc...

Thread B violates the ordering!

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
holds X
requests Y

Process B
holds Y
requests Z

Process C
holds Z
requests X

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
  holds X
  requests Y

X < Y

Process B
  holds Y
  requests Z

Process C
  holds Z
  requests X

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
holds X
requests Y

$\boxed{X < Y}$

Process B
holds Y
requests Z

$\boxed{Y < Z}$

Process C
holds Z
requests X

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
    holds X
    requests Y

$X < Y$

Process B
    holds Y
    requests Z

$Y < Z$

Process C
    holds Z
    requests X

$Z < X$

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
holds X
requests Y $X < Y$

This is impossible!

Process B
holds Y
requests Z $Y < Z$

Process C
holds Z
requests X $Z < X$

Why Does Resource Ordering Work?

Assume deadlock has occurred.

Process A
holds X
requests Y $\boxed{X < Y}$

This is impossible!

Process B
holds Y
requests Z $\boxed{Y < Z}$

Conclusion:
The assumption must have been incorrect

Process C
holds Z
requests X $\boxed{Z < X}$

Resource Ordering

The chief problem:

It is hard to come up with an ordering of the resources that everyone finds acceptable!

Still, I believe this is particularly useful within an OS.

1. **ProcessControlBlock**
2. **FileControlBlock**
3. **Page Frames**

Also, the problem of resources with multiple units is not addressed.