

بسم الله الرحمن الرحيم

«سیستم عامل»

۱

جلسه ۹: بن بست (۲)

یادآوری

Resources and deadlocks

- ❑ Processes need access to resources in order to make progress
- ❑ Examples of computer resources
 - ❖ printers
 - ❖ disk drives
 - ❖ kernel data structures (scheduling queues ...)
 - ❖ locks/semaphores to protect critical sections
- ❑ Suppose a process holds resource A and requests resource B
 - ❖ at the same time another process holds B and requests A
 - ❖ both are blocked and remain so ... **this is deadlock**

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Example:

```
var r1_mutex: Mutex
...
r1_mutex.Lock()
Use resource_1
r1_mutex.Unlock()
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Another Example:

```
var r1_sem: Semaphore
r1_sem.Signal()
...
r1_sem.Wait()
Use resource_1
r1_sem.Signal()
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)  
use resource_1  
release (resource_1)
```

Thread B:

```
acquire (resource_2)  
use resource_2  
release (resource_2)
```

Resource acquisition scenarios

Thread A:

```
acquire (resource_1)
use resource_1
release (resource_1)
```

Thread B:

```
acquire (resource_2)
use resource_2
release (resource_2)
```

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

Thread A:

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

Thread B:

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


Resource acquisition scenarios: 2 resources

Thread A:

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

Thread B:

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

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

Thread A:

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

Thread B:

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

Resource acquisition scenarios: 2 resources

Thread A:

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

Thread B:

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

No deadlock can occur here!

Resource acquisition scenarios: 2 resources

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)
```


Resource acquisition scenarios: 2 resources

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)
```

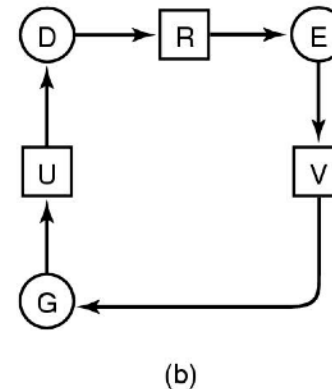
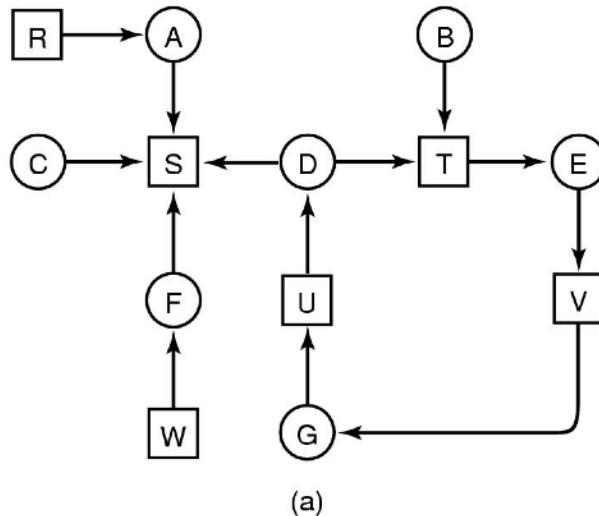
Deadlock is possible!

Dealing with deadlock

- ❑ **Four general strategies**
 - ❖ Ignore the problem
 - Hmm... advantages, disadvantages?
 - ❖ Detection and recovery
 - ❖ Dynamic avoidance via careful resource allocation
 - ❖ Prevention, by structurally negating one of the four necessary conditions

Deadlock detection

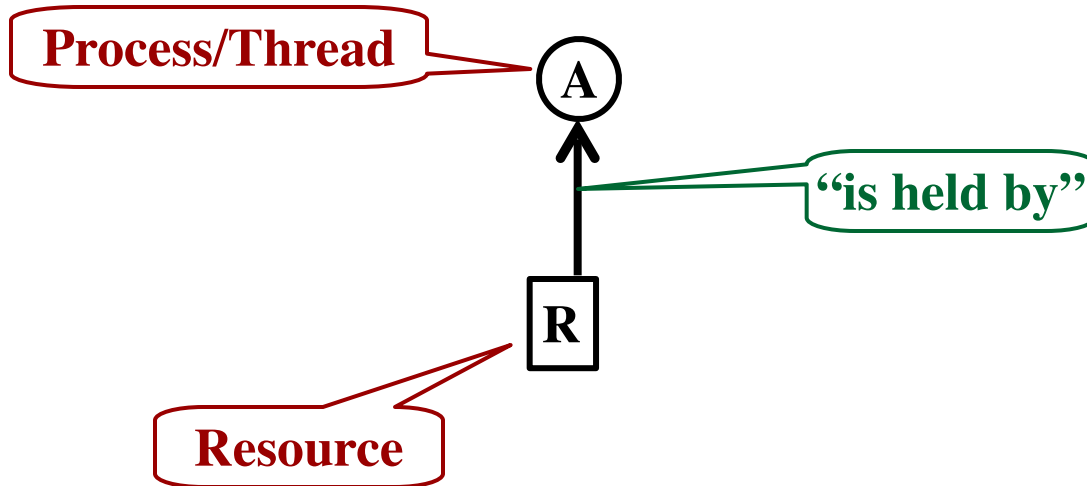
- ❑ Let the problem happen, then recover
- ❑ How do you know it happened?
- ❑ Do a depth-first-search on the resource allocation graph



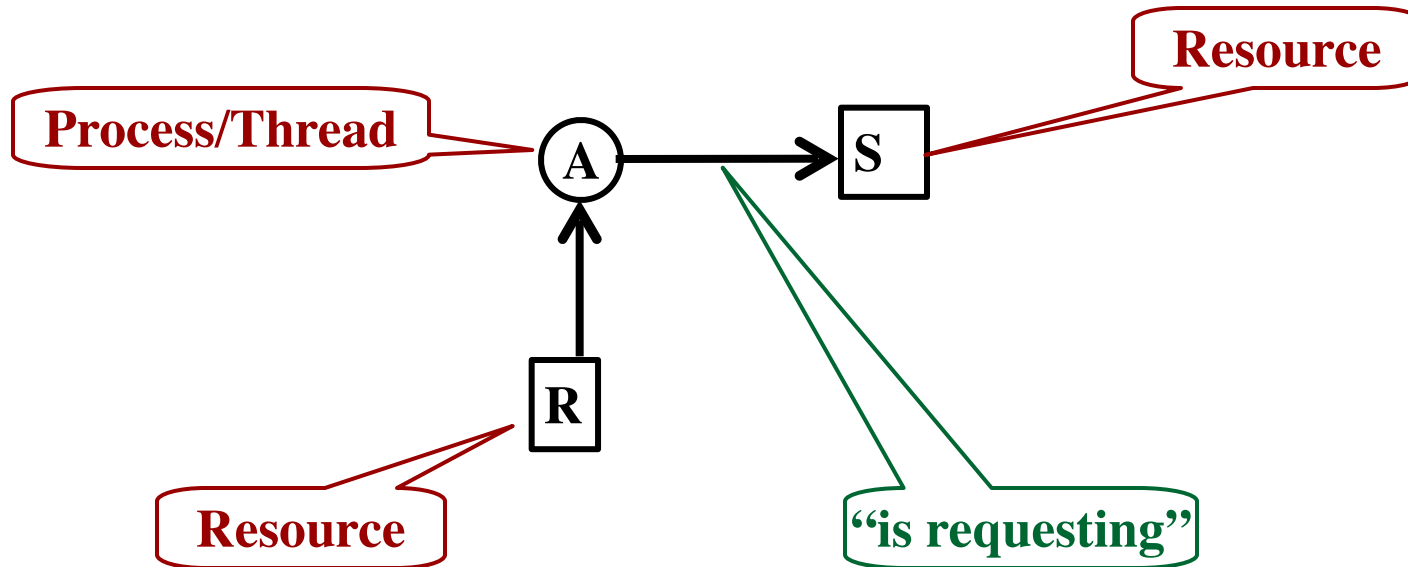
Detection: Resource Allocation Graphs



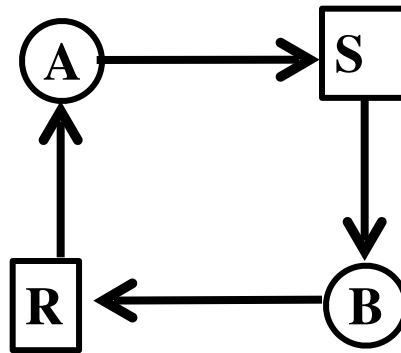
Detection: Resource Allocation Graphs



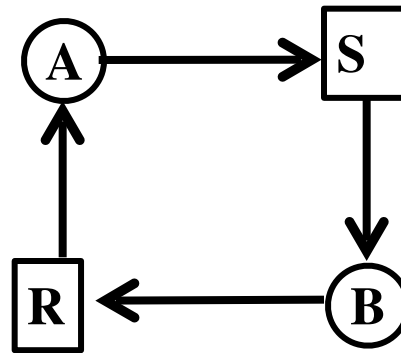
Detection: Resource Allocation Graphs



Detection: Resource Allocation Graphs

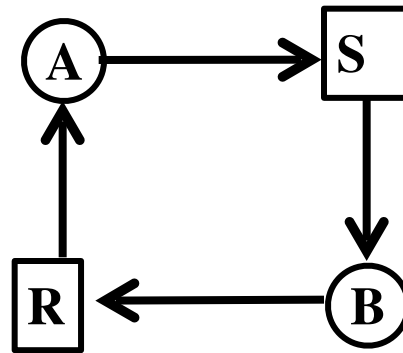


Detection: Resource Allocation Graphs



Deadlock

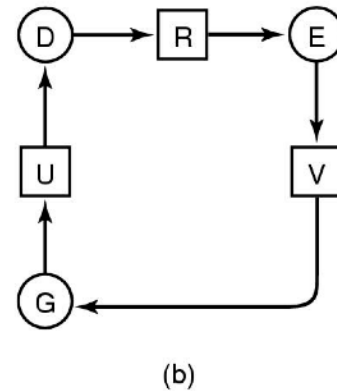
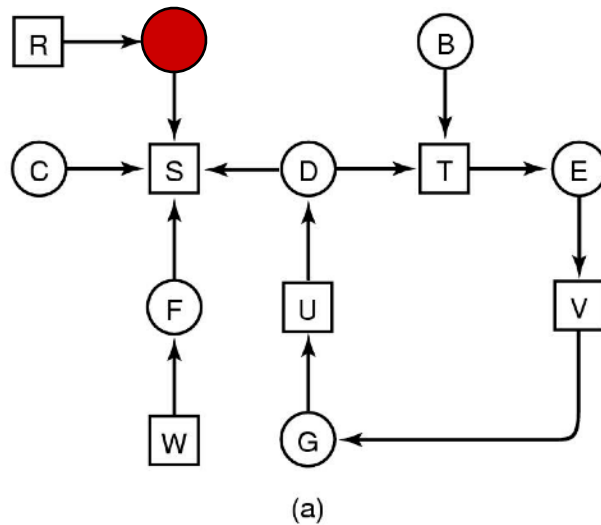
Detection: Resource Allocation Graphs



Deadlock = a cycle in the graph

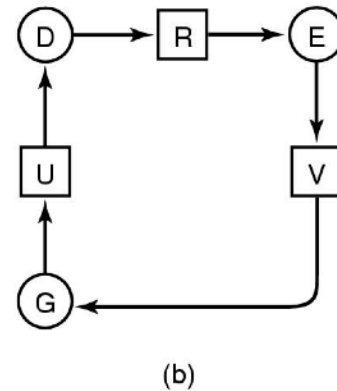
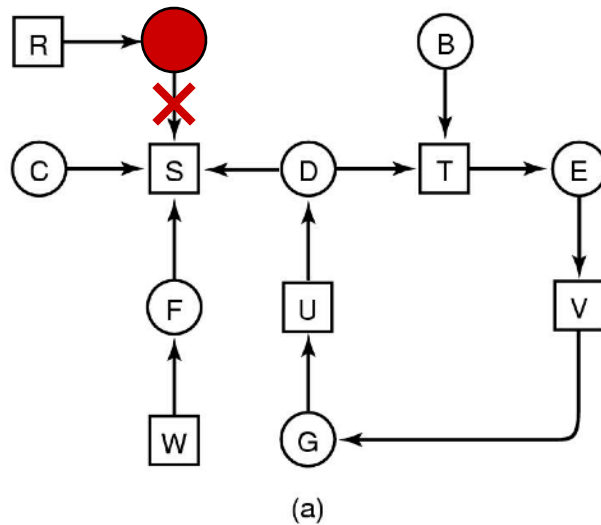
Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph



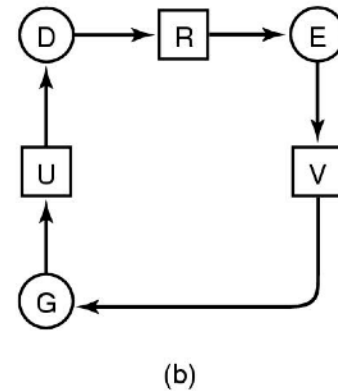
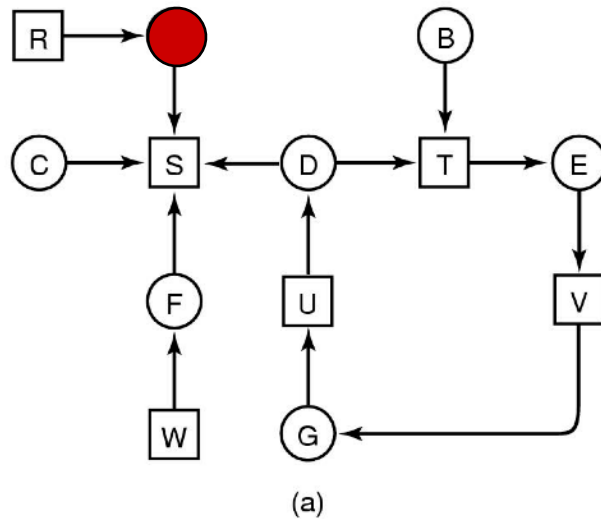
Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph



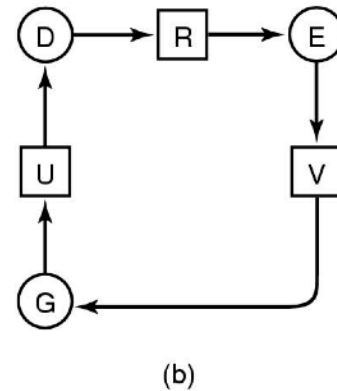
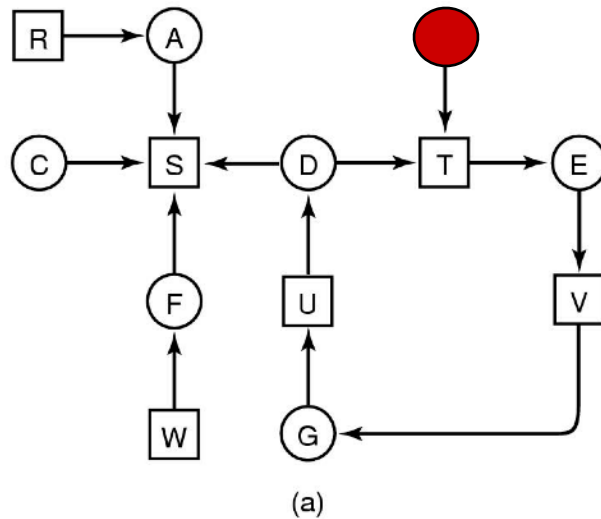
Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph



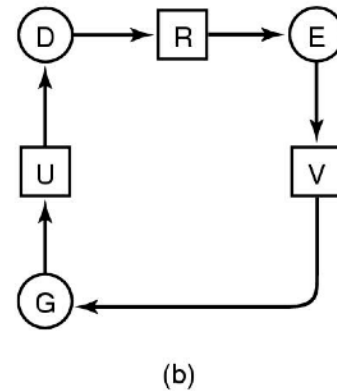
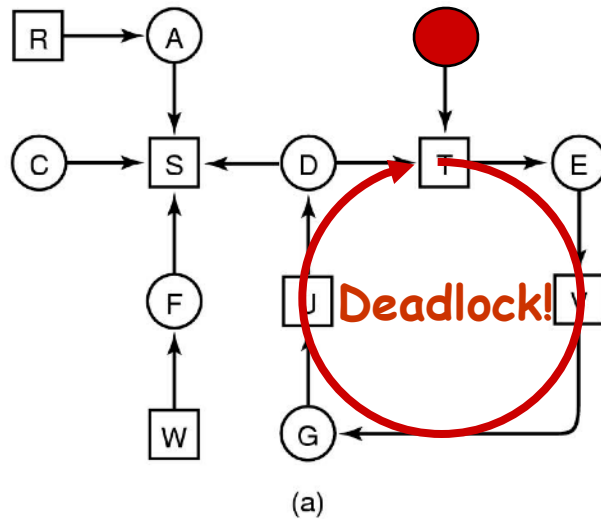
Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph



Deadlock detection (1 resource of each)

- Do a depth-first-search on the resource allocation graph



Multple units/instances of a resource

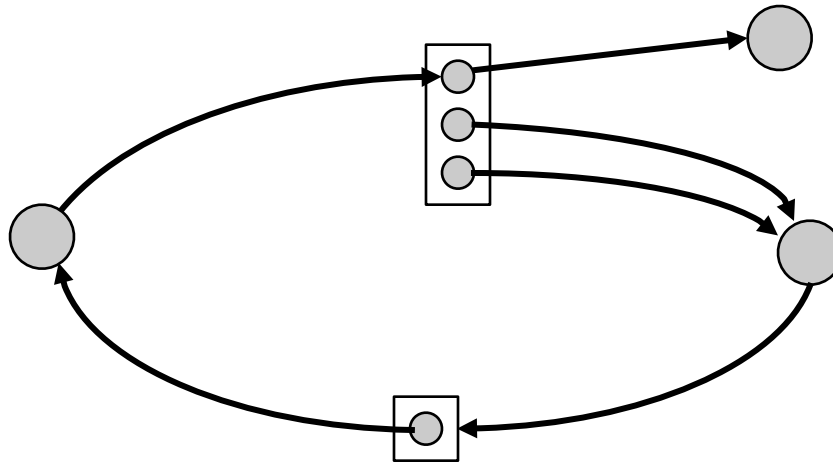
- ❑ **Some resources have only one “unit”.**
 - ❖ Only one thread at a time may hold the resource.
 - Printer
 - Lock on ReadyQueue
- ❑ **Some resources have several units.**
 - ❖ All units are considered equal; any one will do.
 - Page Frames
 - Dice in the Gaming Parlor problem
 - ❖ A thread requests “k” units of the resource.
 - ❖ Several requests may be satisfied simultaneously.

Deadlock modeling with multiple resources

- Theorem: If a graph does not contain a cycle then no processes are deadlocked
 - ❖ A cycle in a RAG is a necessary condition for deadlock
 - ❖ Is it a sufficient condition?

Deadlock modeling with multiple resources

- **Theorem**: If a graph does not contain a cycle then no processes are deadlocked
 - ❖ A cycle in a RAG is a necessary condition for deadlock
 - ❖ Is it a sufficient condition?



Deadlock detection issues

- How often should the algorithm run?
 - ❖ On every resource request?
 - ❖ Periodically?
 - ❖ When CPU utilization is low?
 - ❖ When we suspect deadlock because some thread has been asleep for a long period of time?

Recovery from deadlock

- **If we detect deadlock, what should be done to recover?**
 - ❖ Abort **all** deadlocked processes and reclaim resources
 - ❖ Abort **one** process at a time until deadlock cycle is eliminated

- **Where to start?**
 - ❖ Lowest priority process?
 - ❖ Shortest running process?
 - ❖ Process with fewest resources held?
 - ❖ Batch processes before interactive processes?
 - ❖ Minimize number of processes to be terminated?

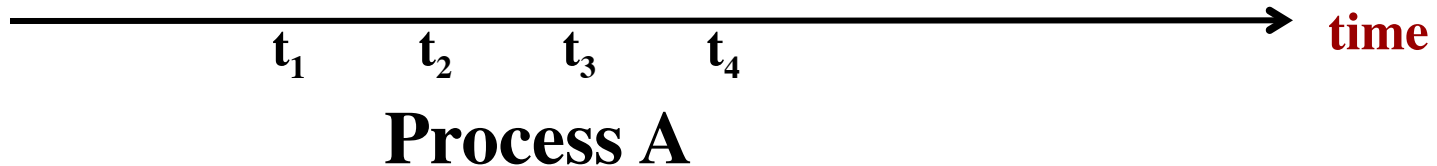
Other deadlock recovery techniques

- ❑ **How do we prevent the resource becoming corrupted**
 - ❖ For example, shared variables protected by a lock?
- ❑ **Recovery through preemption and rollback**
 - ❖ Save state periodically (at start of critical section)
 - take a checkpoint of memory
 - start computation again from checkpoint
 - Checkpoint must be prior to resource acquisition!
 - ❖ Useful for long-lived computation systems

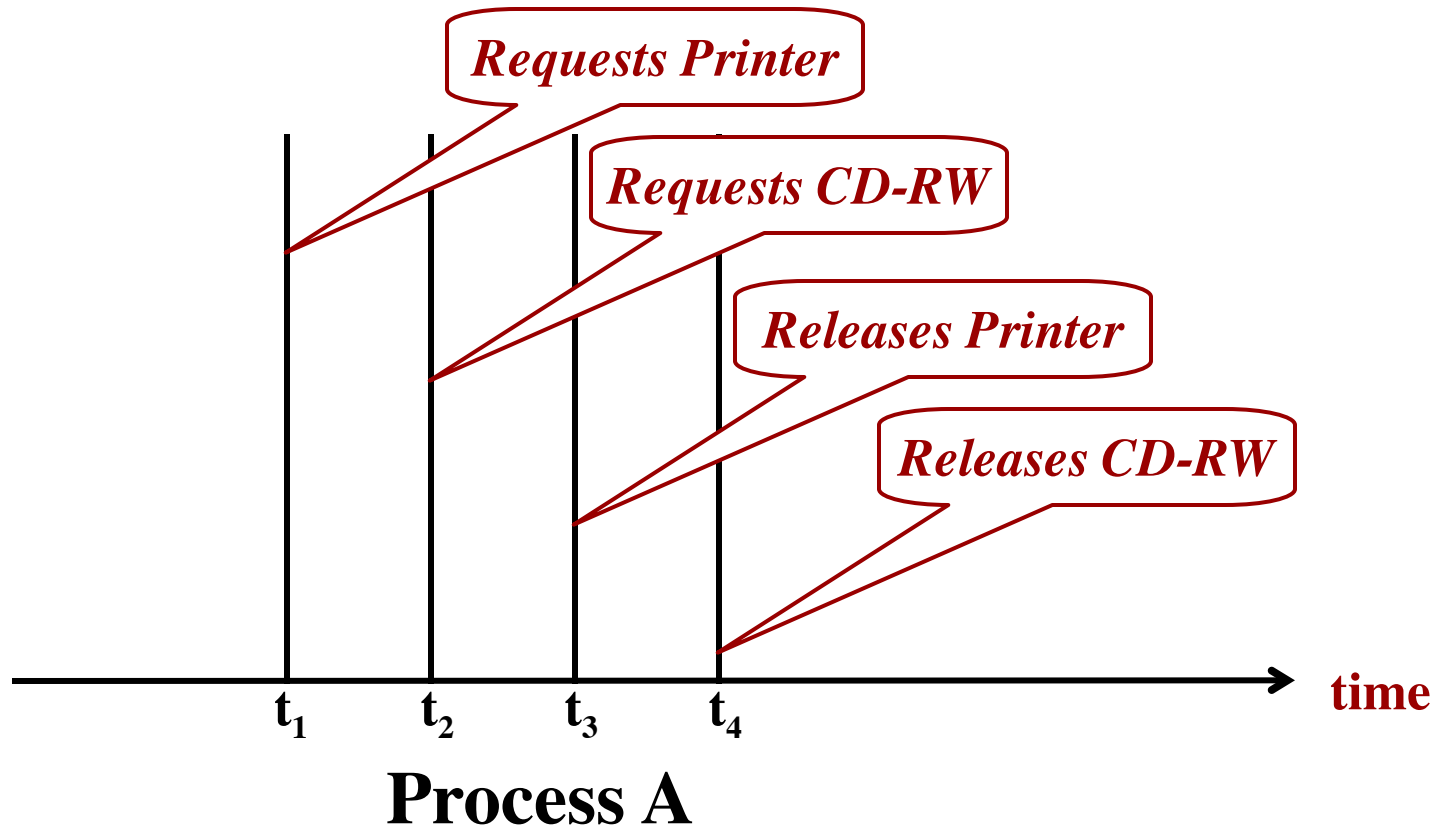
Deadlock avoidance

- Detection vs. avoidance...
 - ❖ Detection - "optimistic" approach
 - Allocate resources
 - "Break" system to fix the problem if necessary
 - ❖ 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
 - And how easy is it to recover from deadlock!

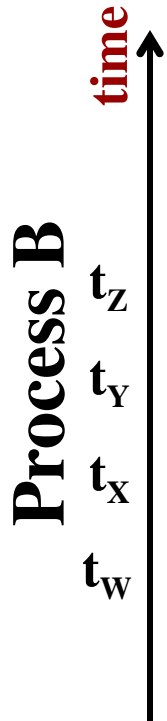
Avoidance using process-resource trajectories



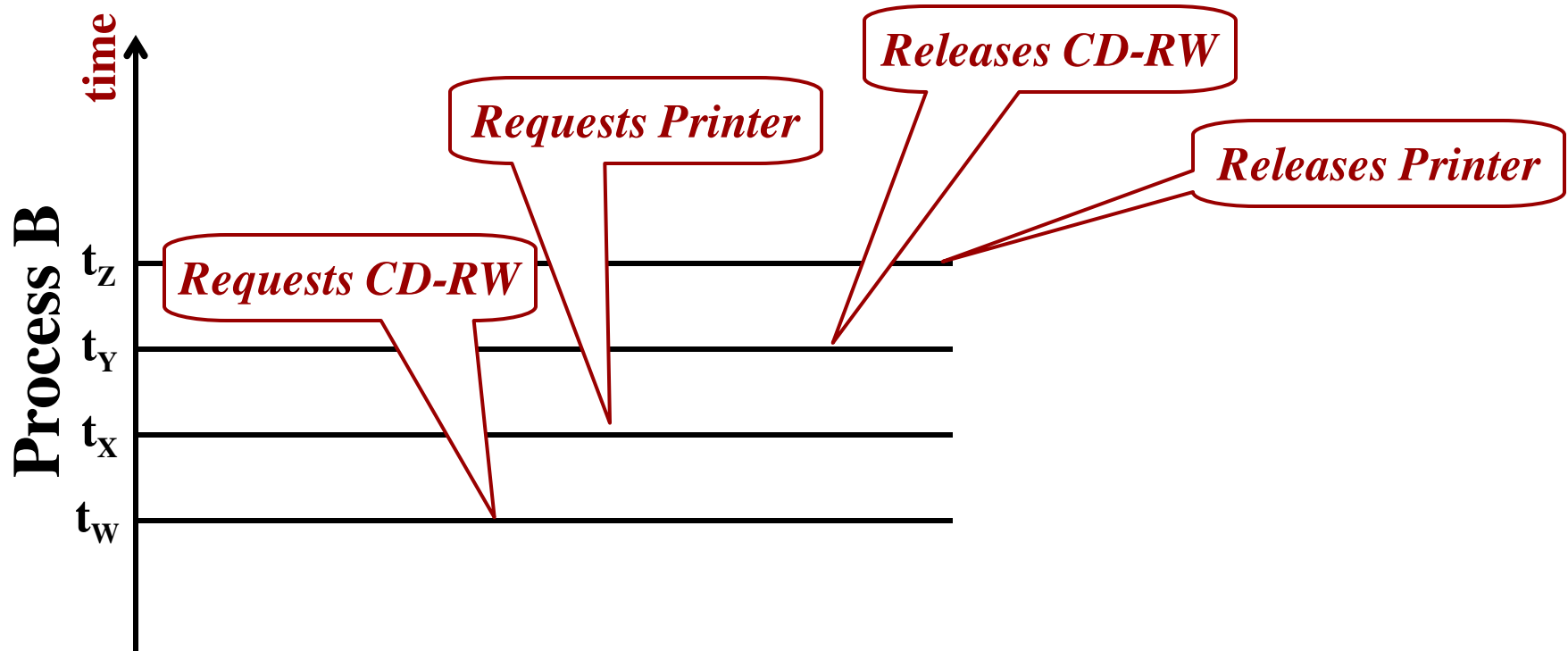
Avoidance using process-resource trajectories

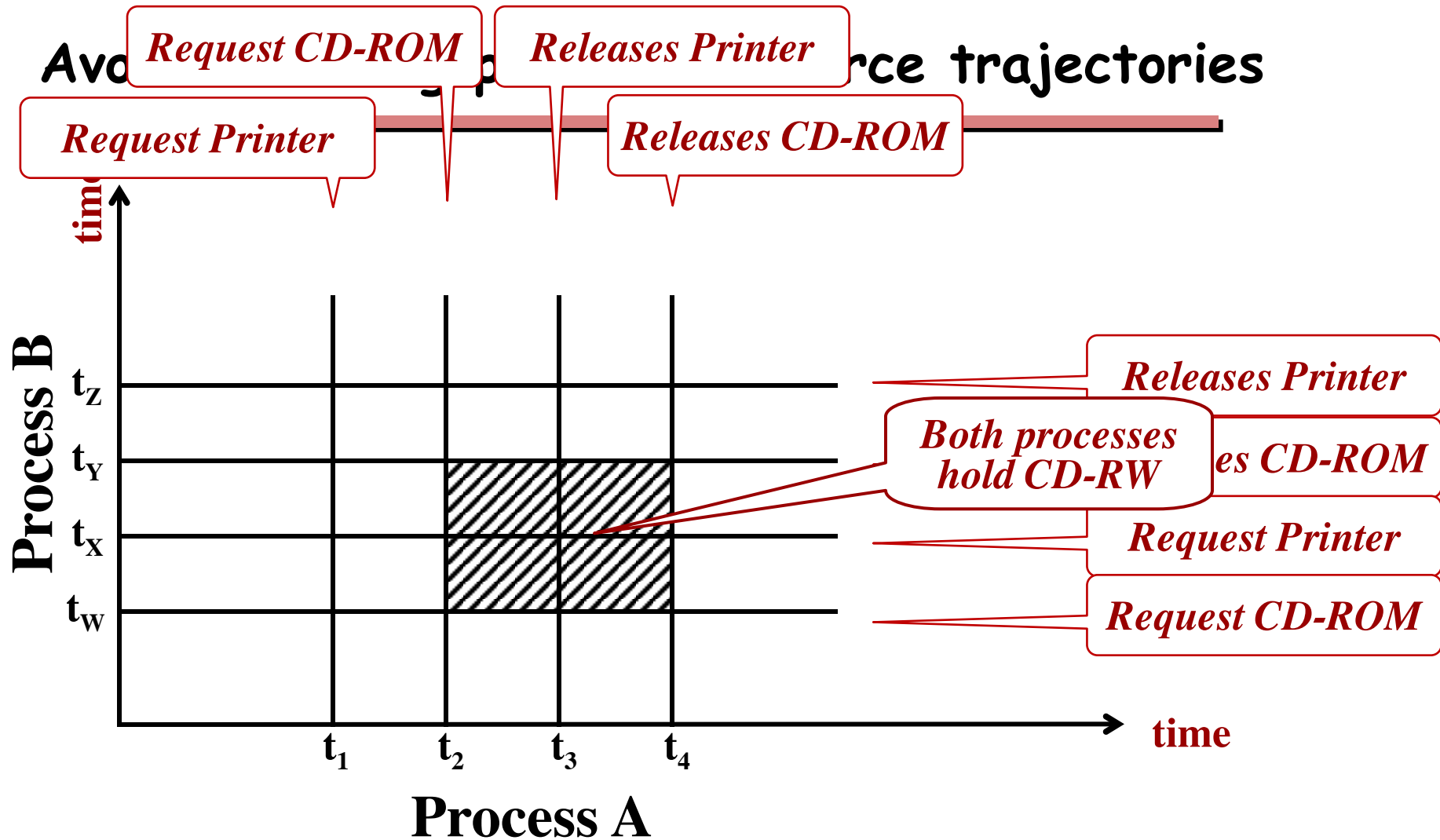


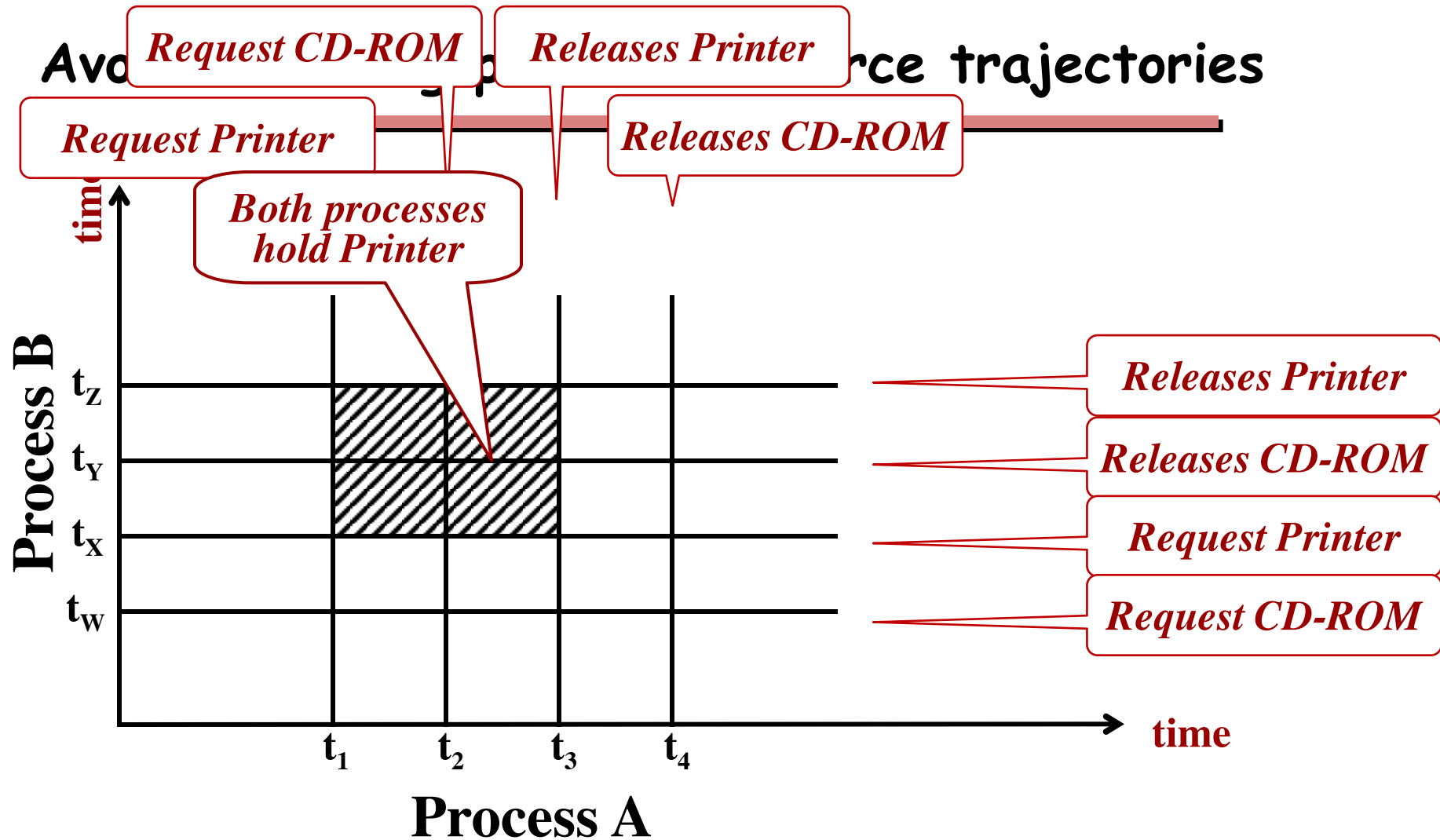
Avoidance using process-resource trajectories

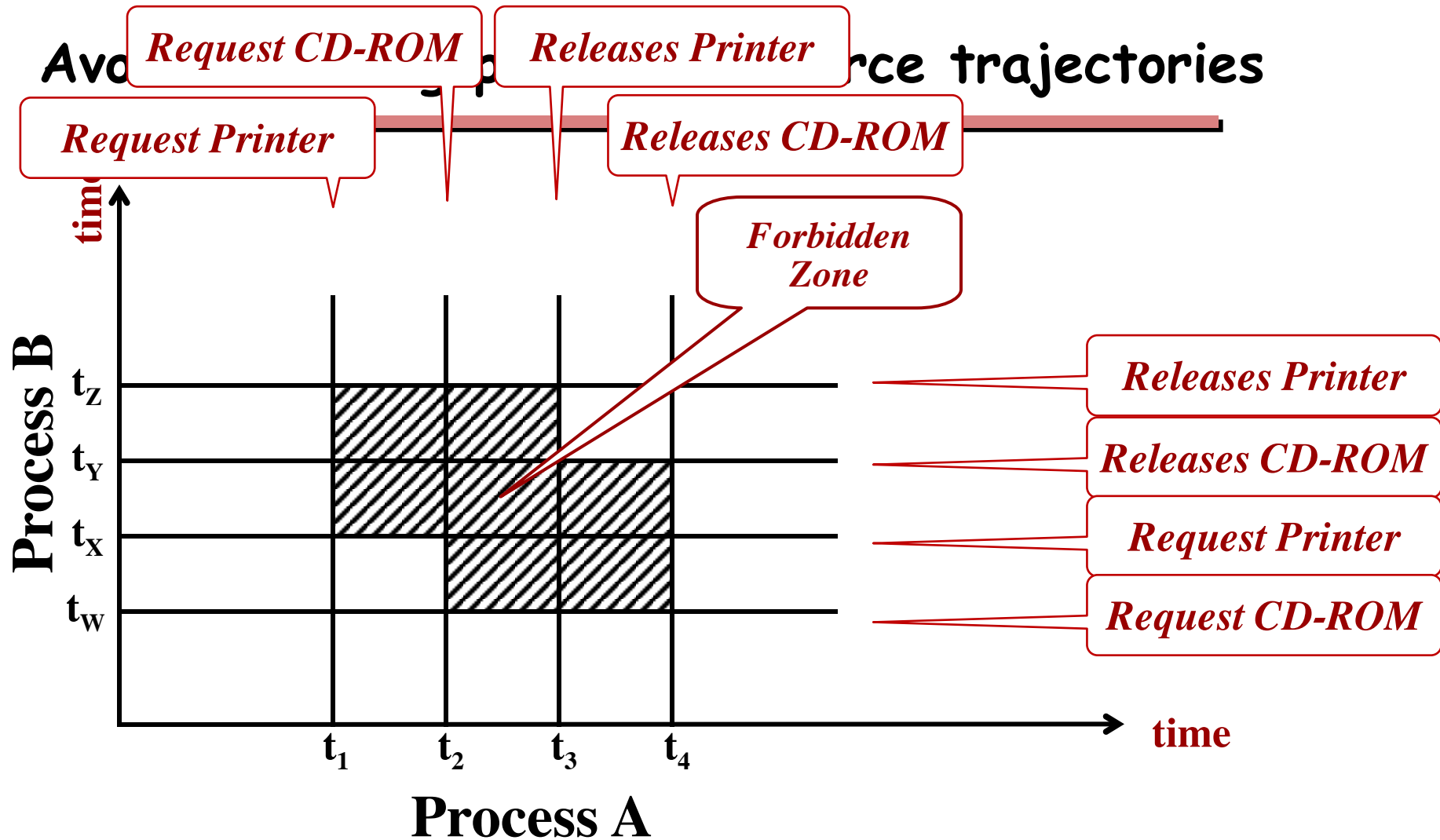


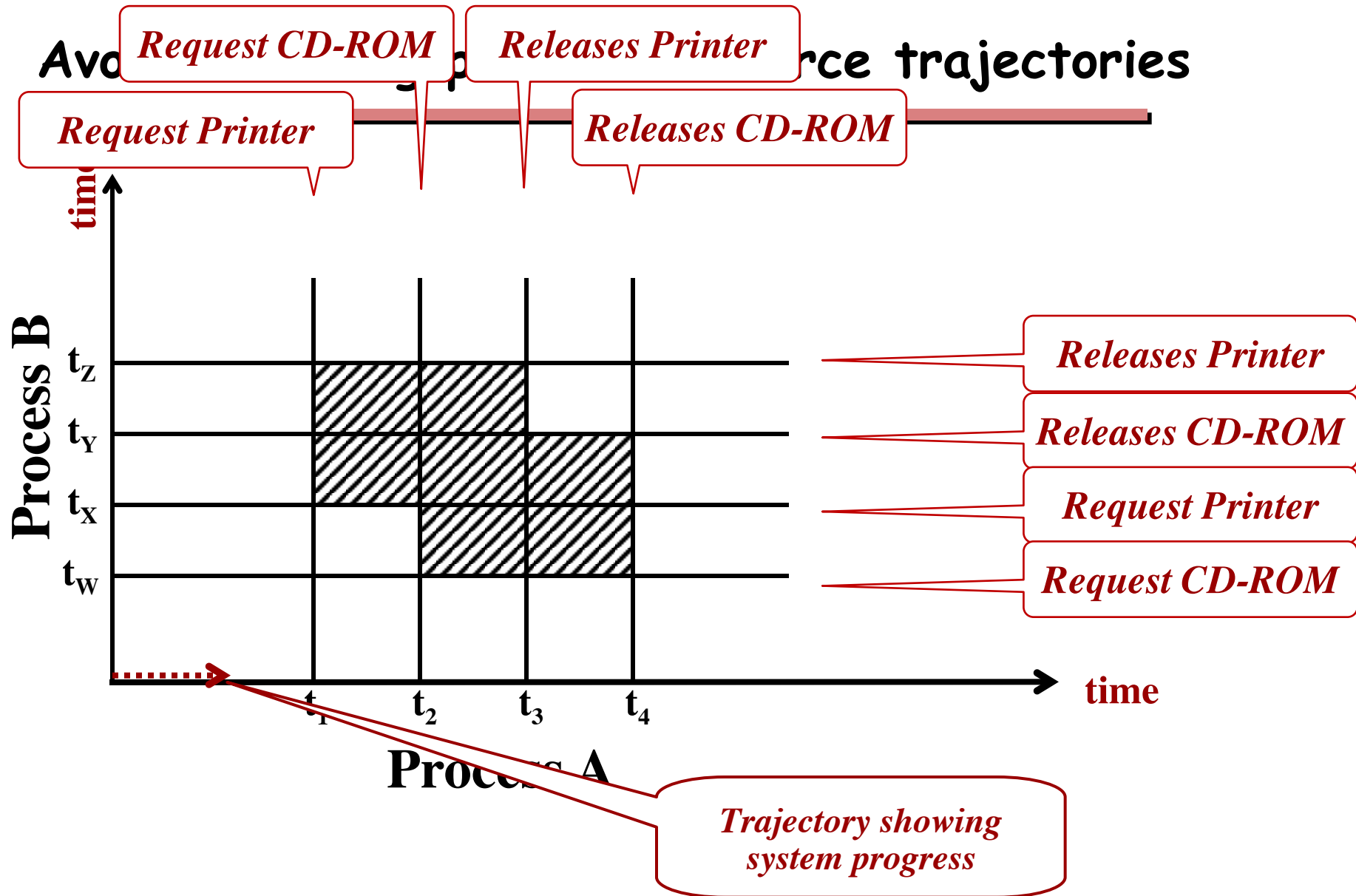
Avoidance using process-resource trajectories

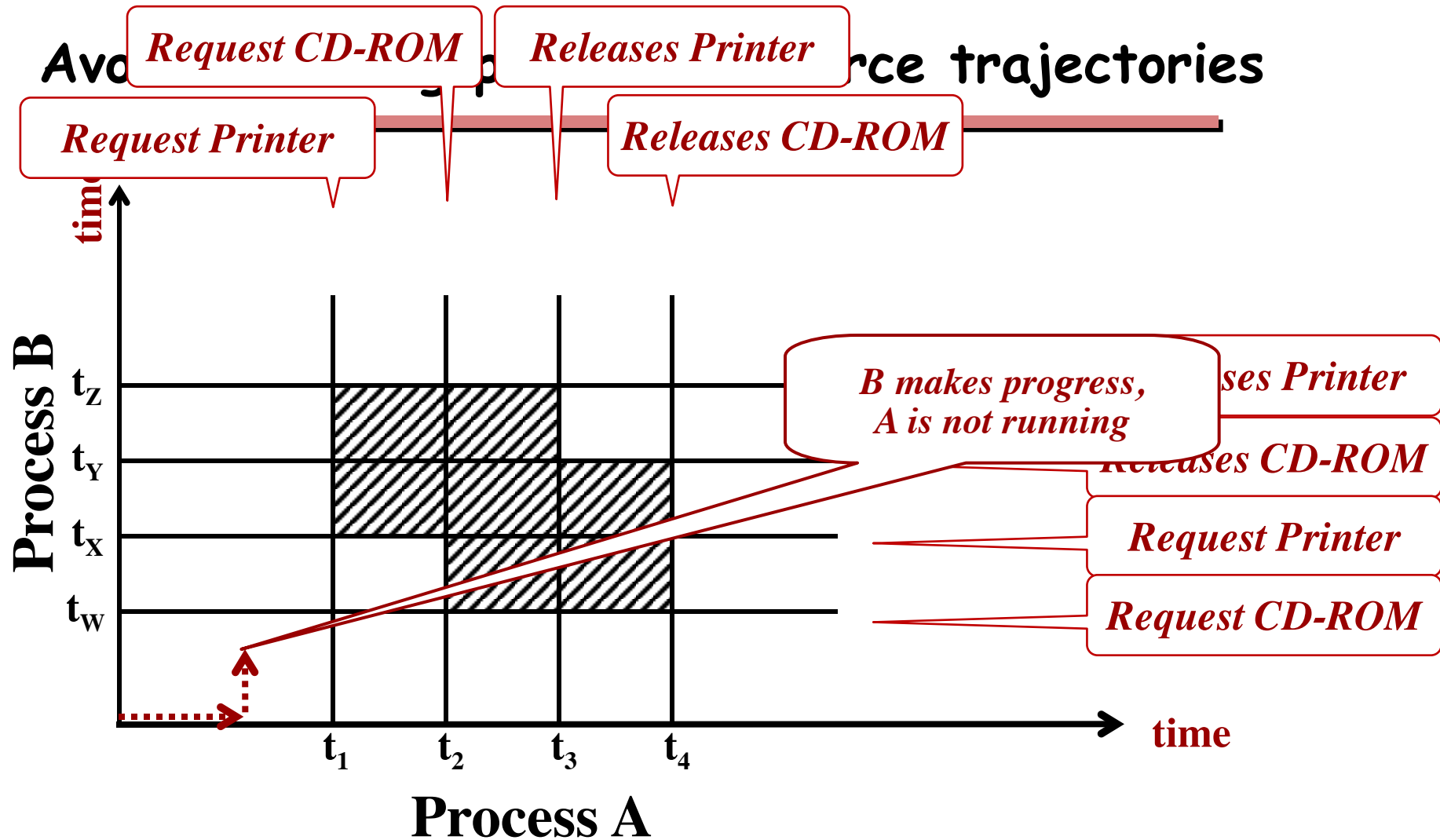


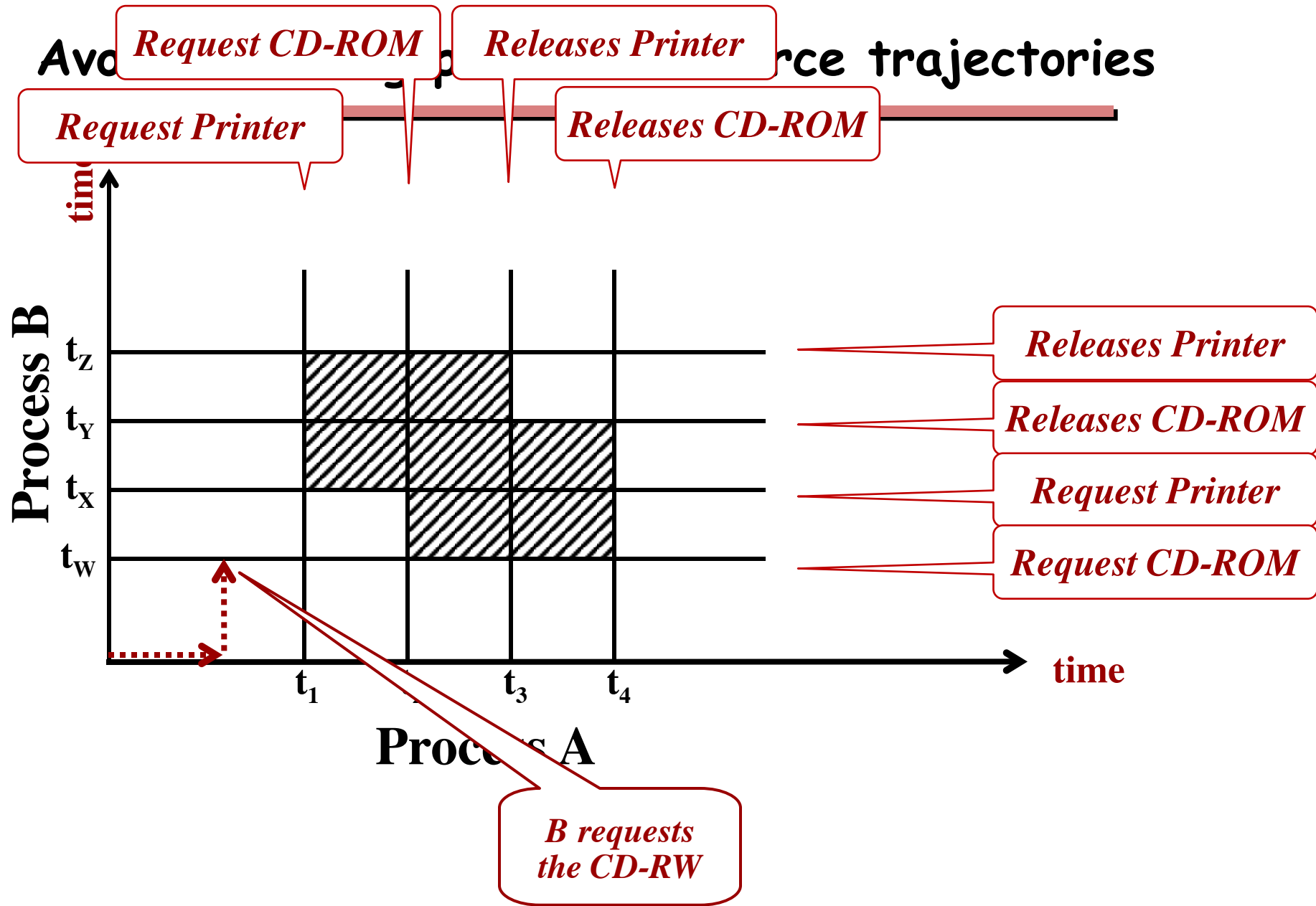


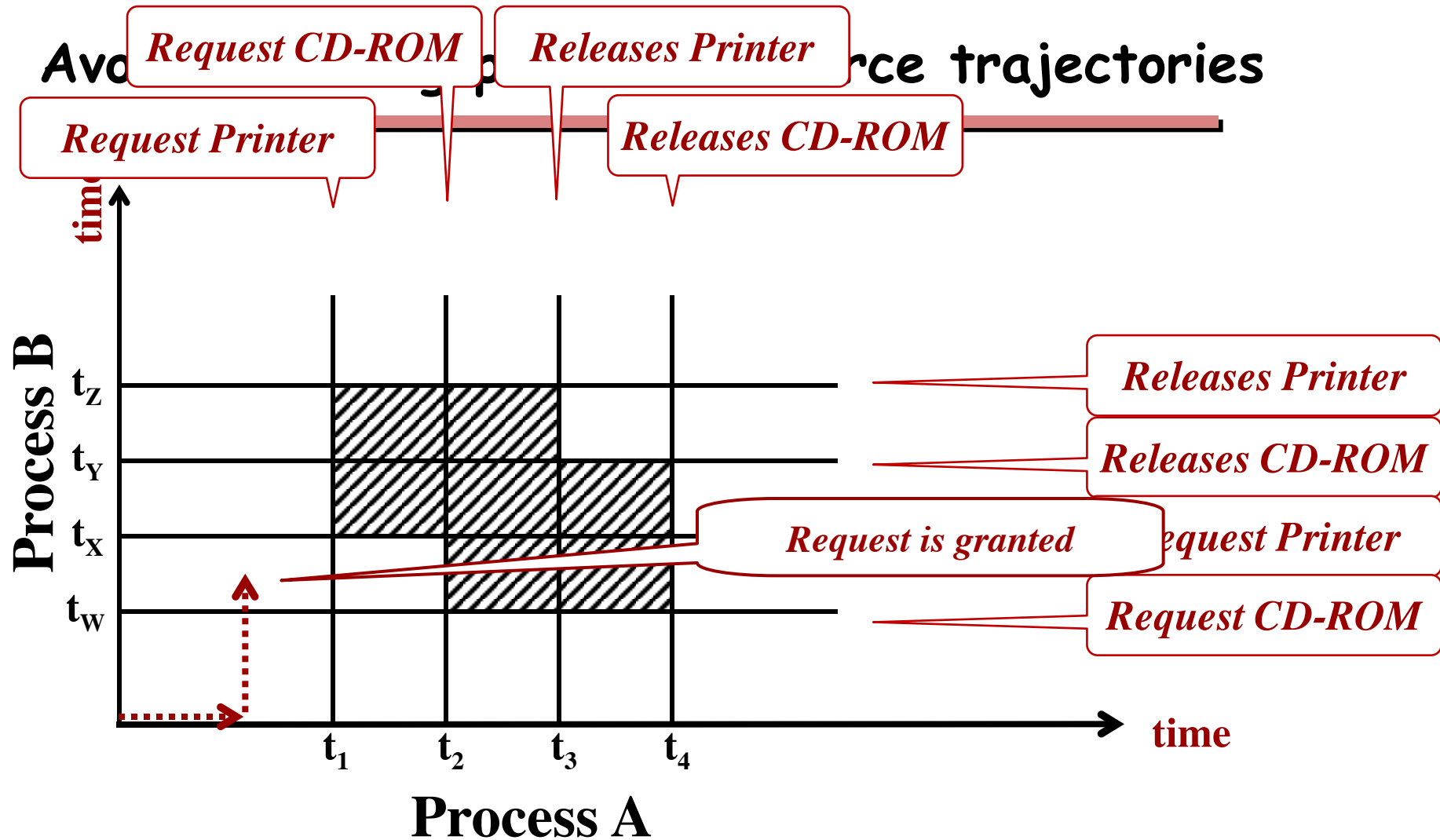


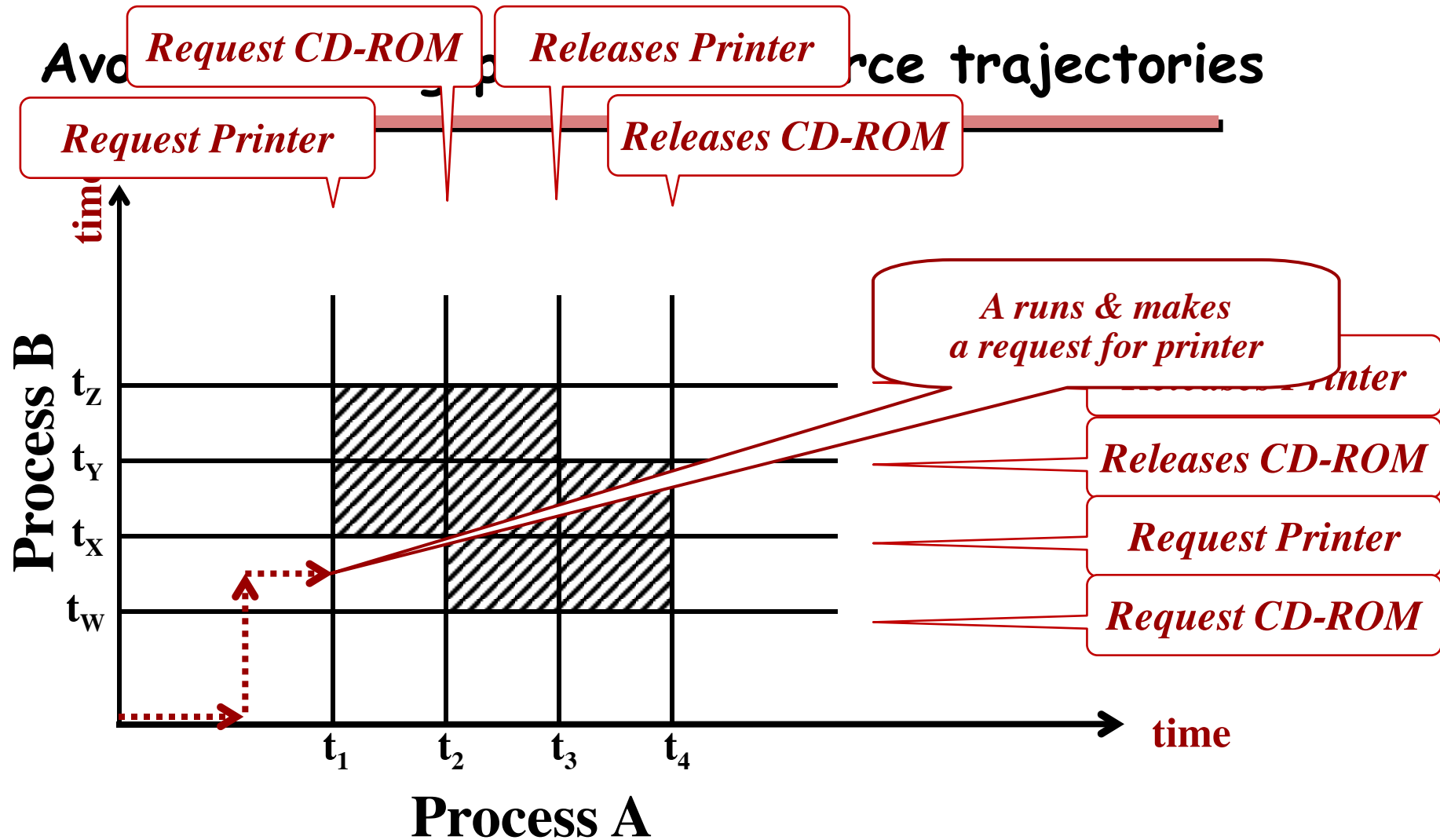


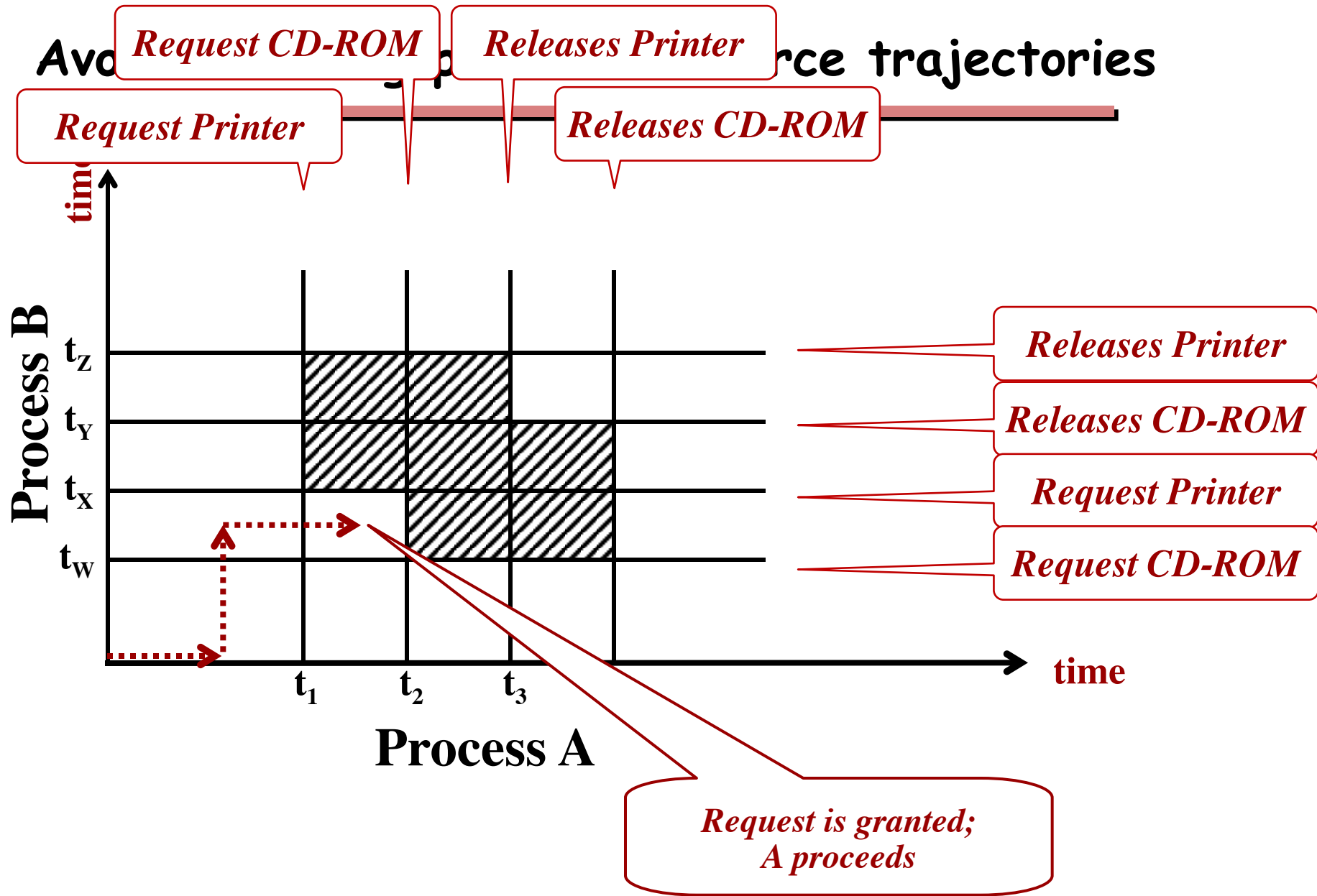


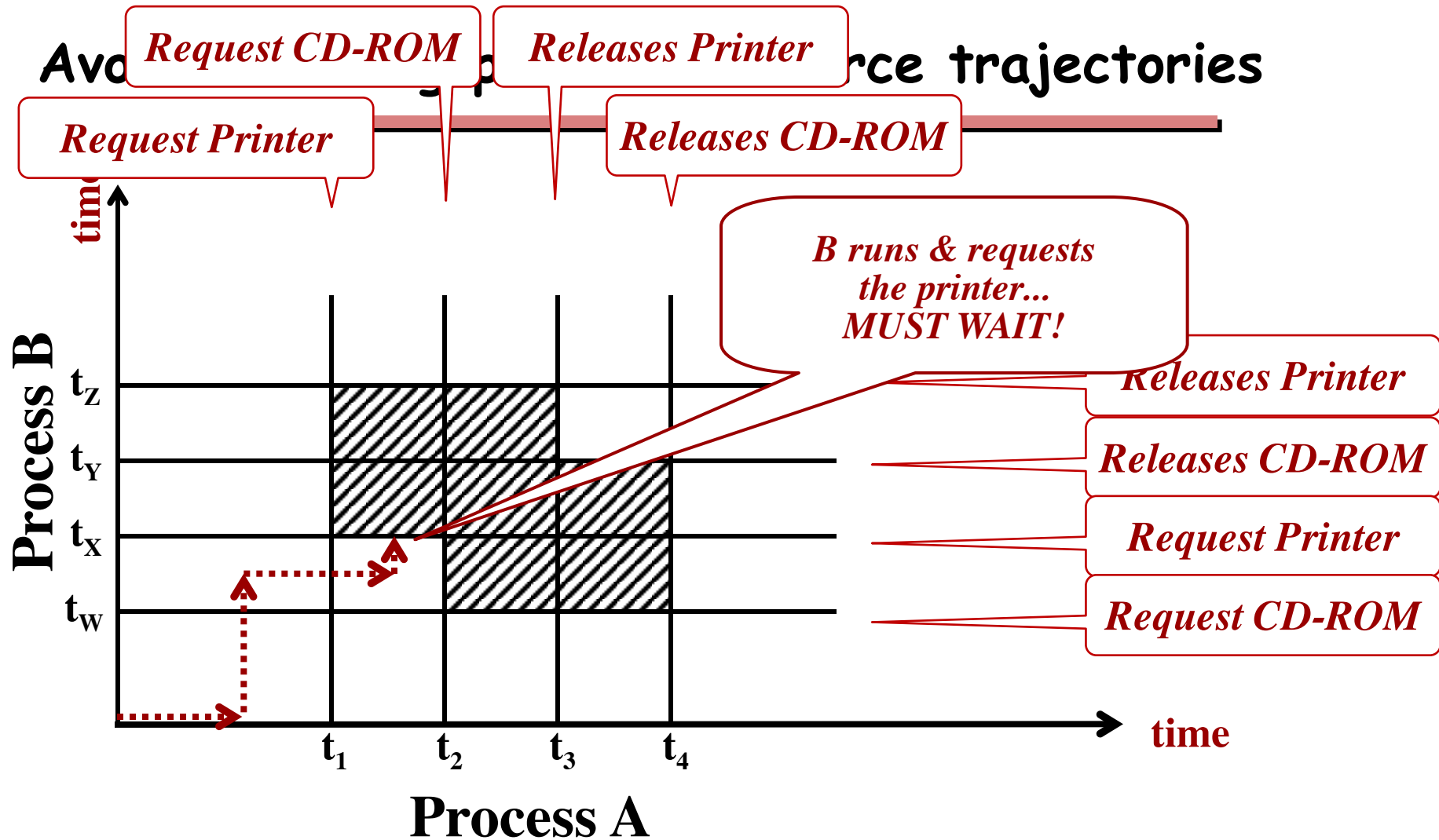


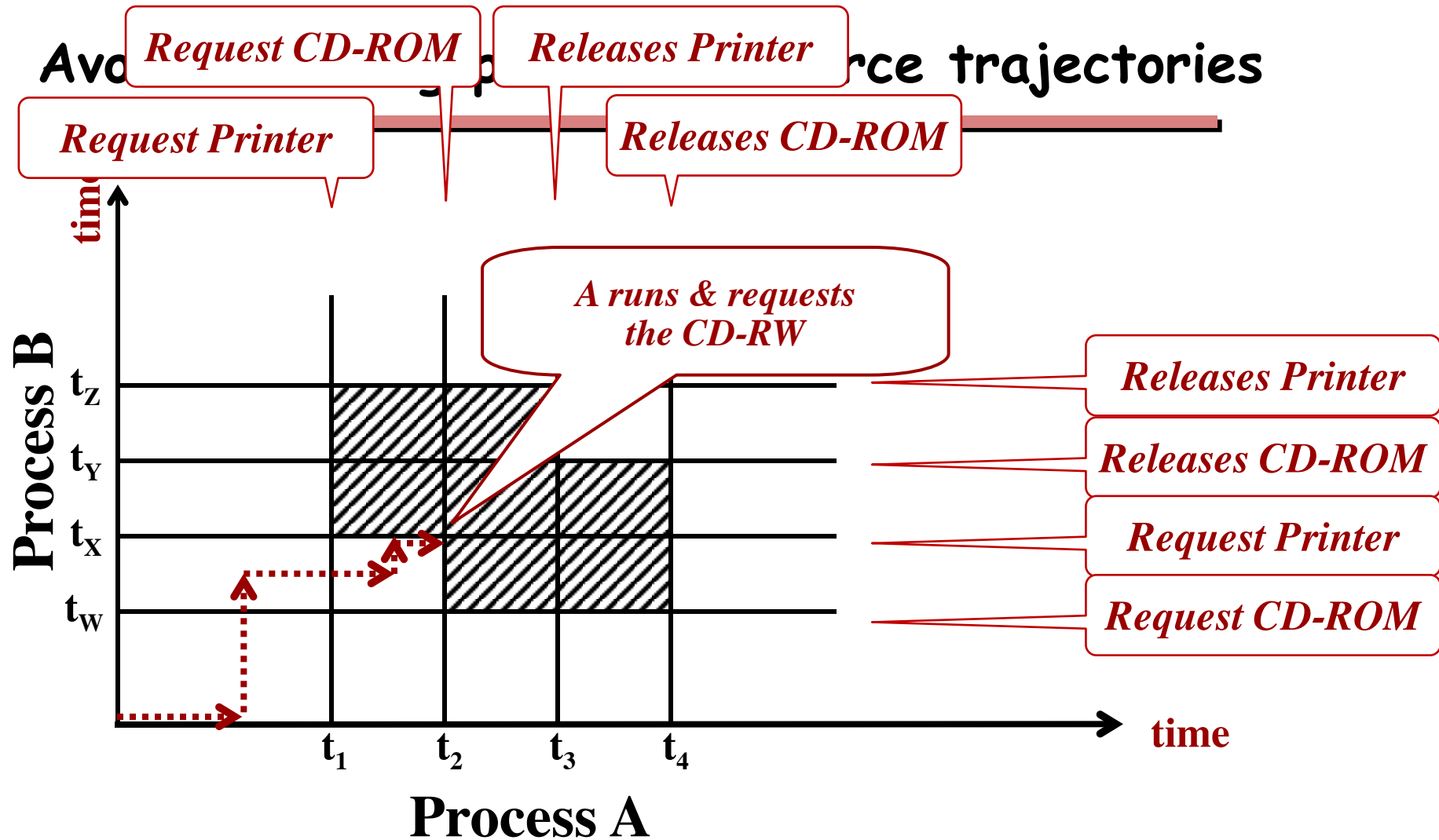


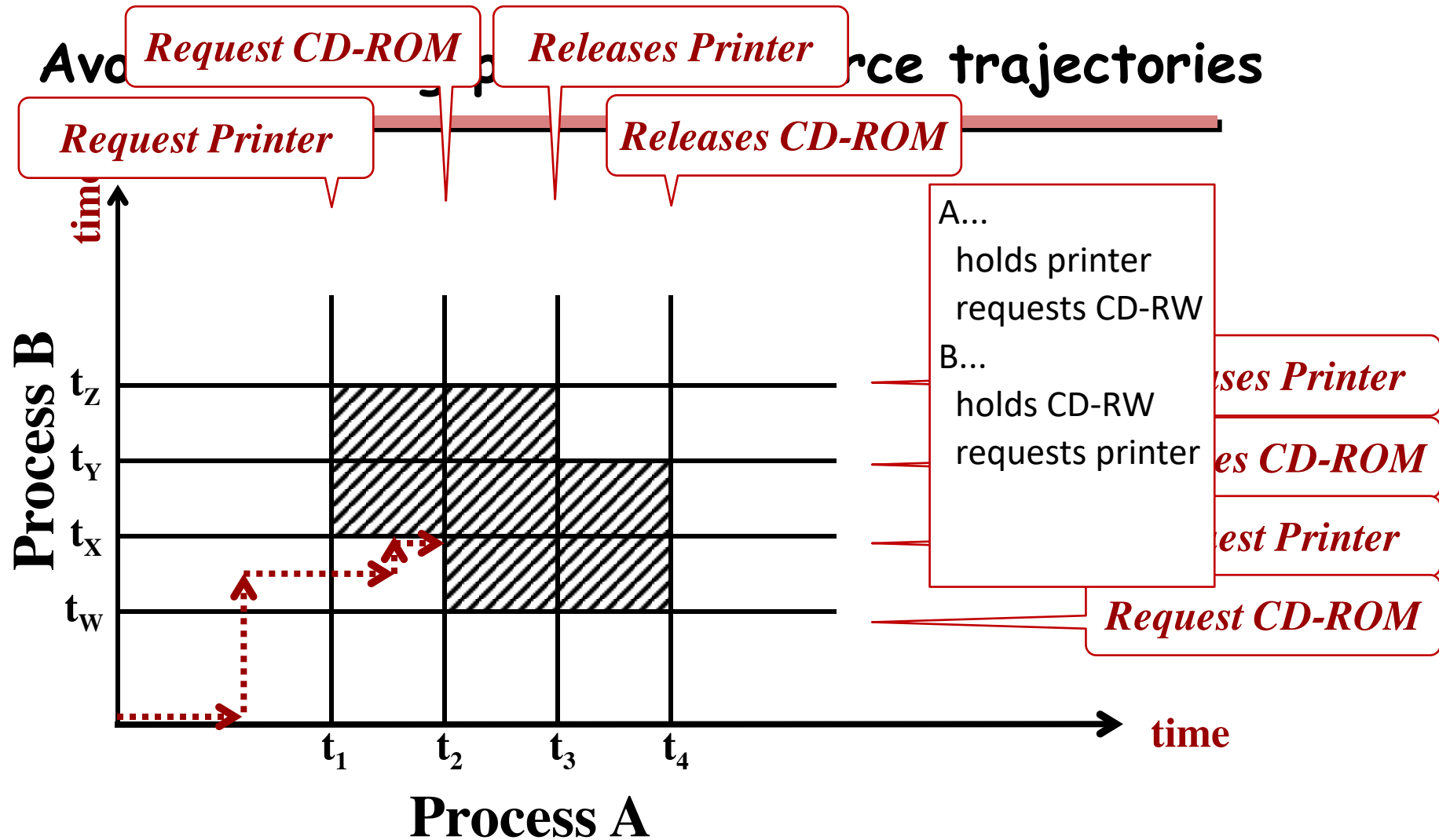




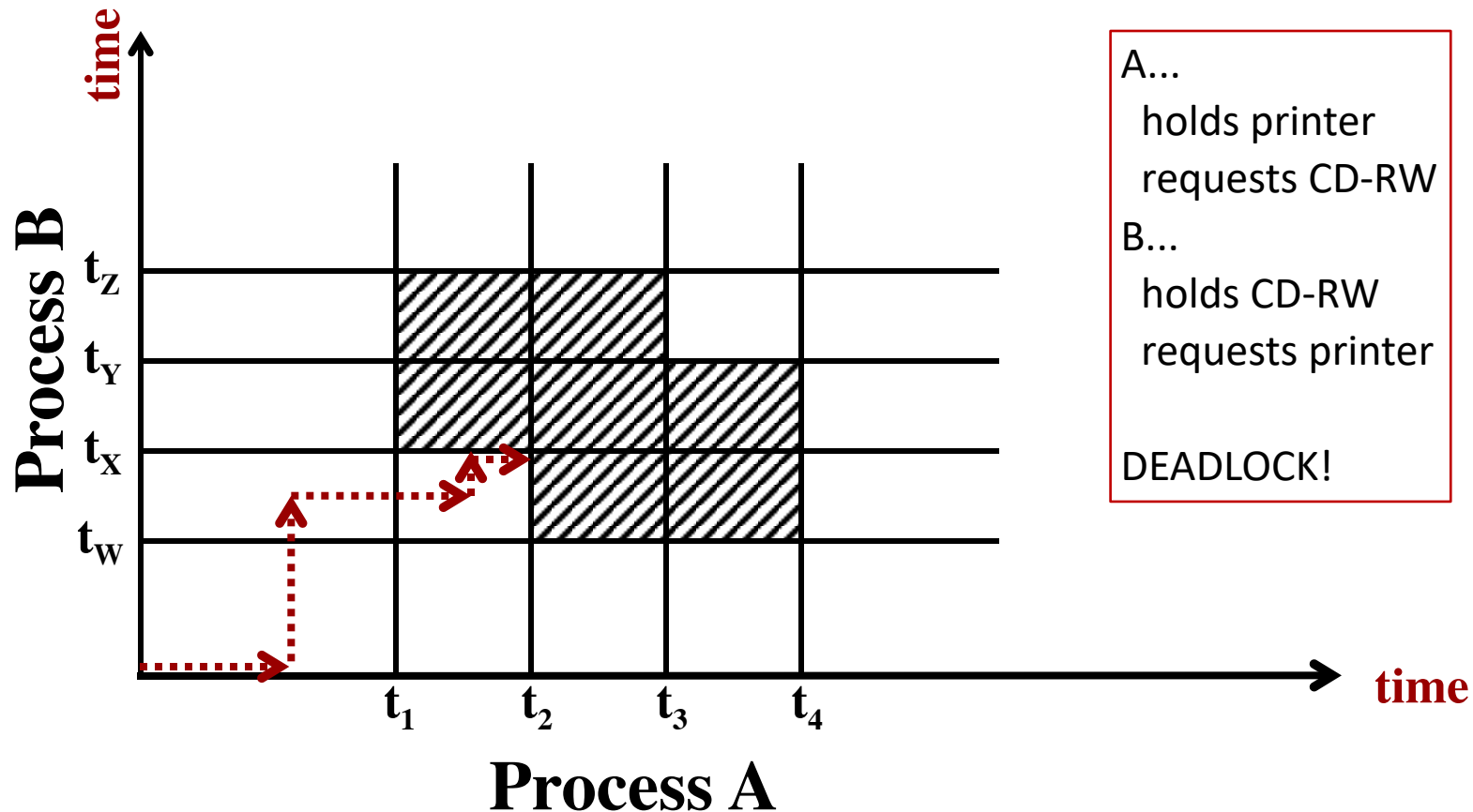




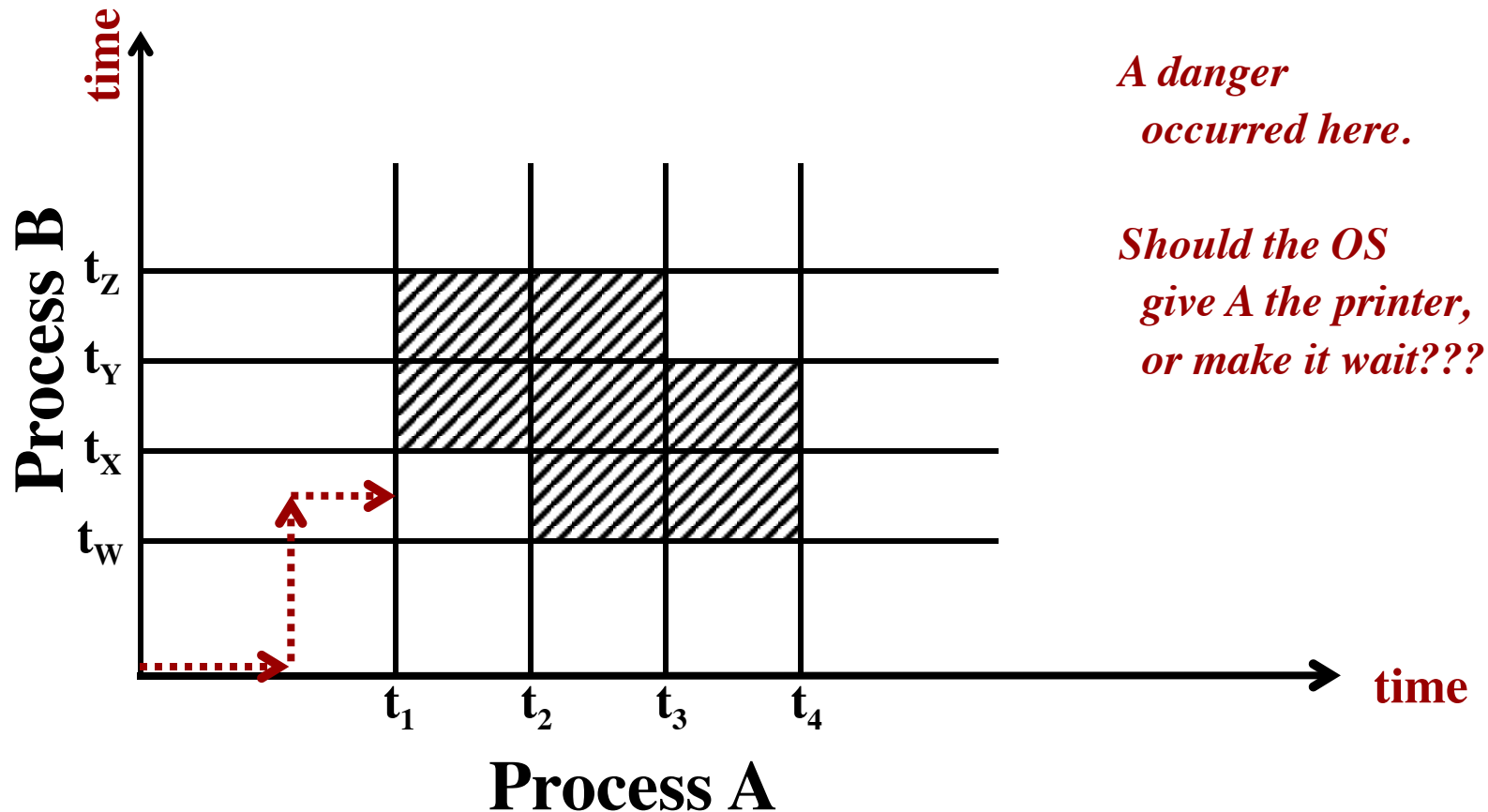




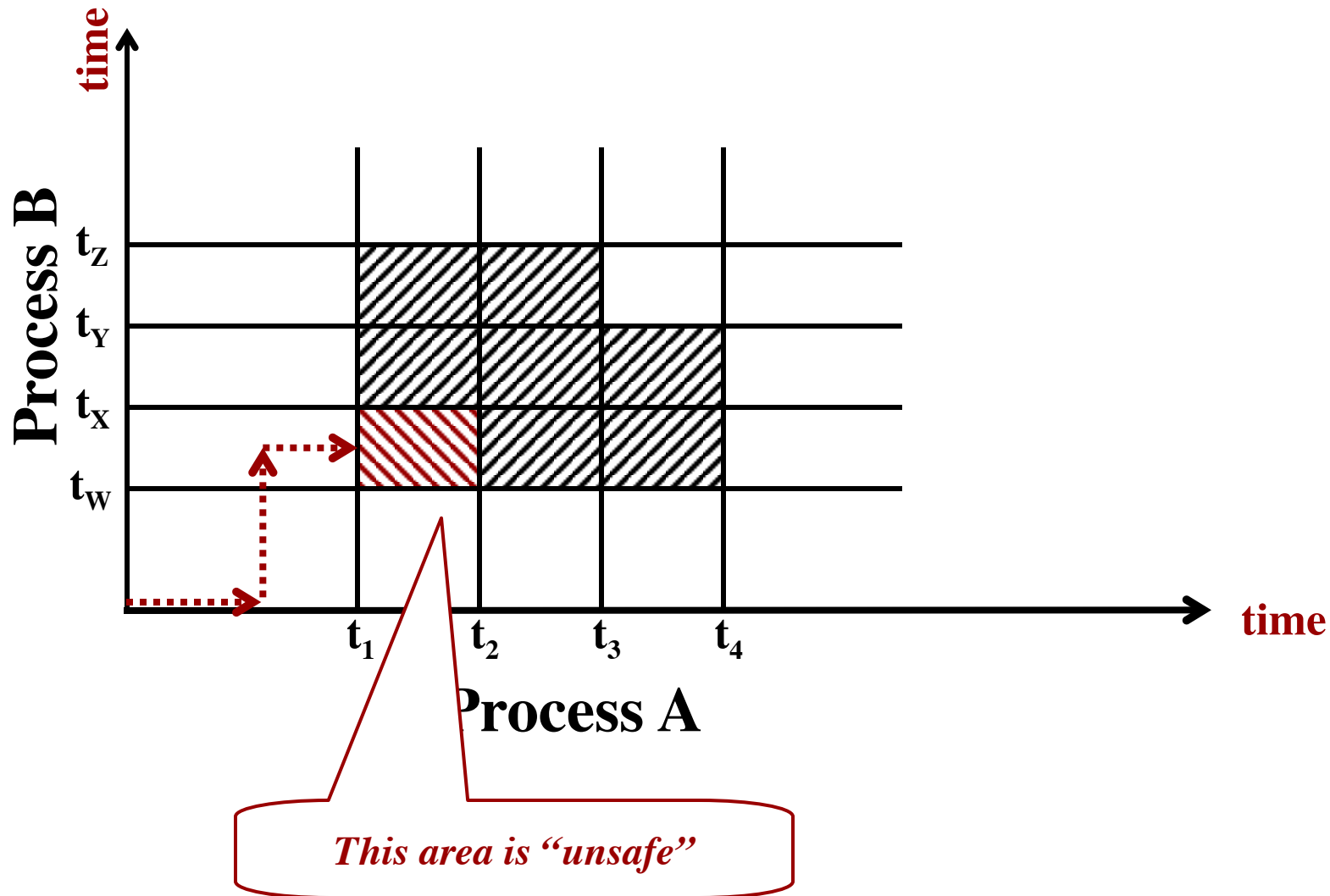
Avoidance using process-resource trajectories



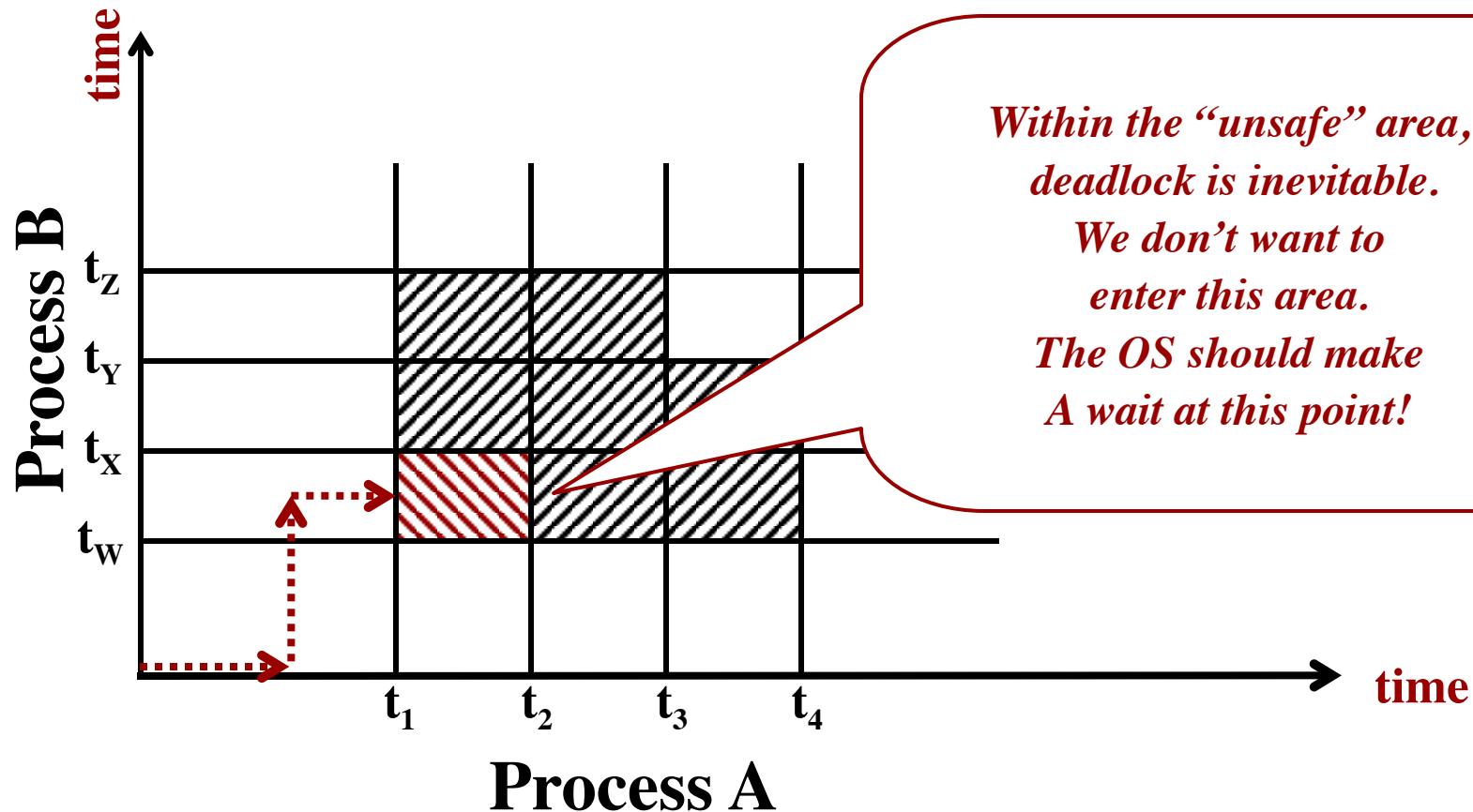
Avoidance using process-resource trajectories



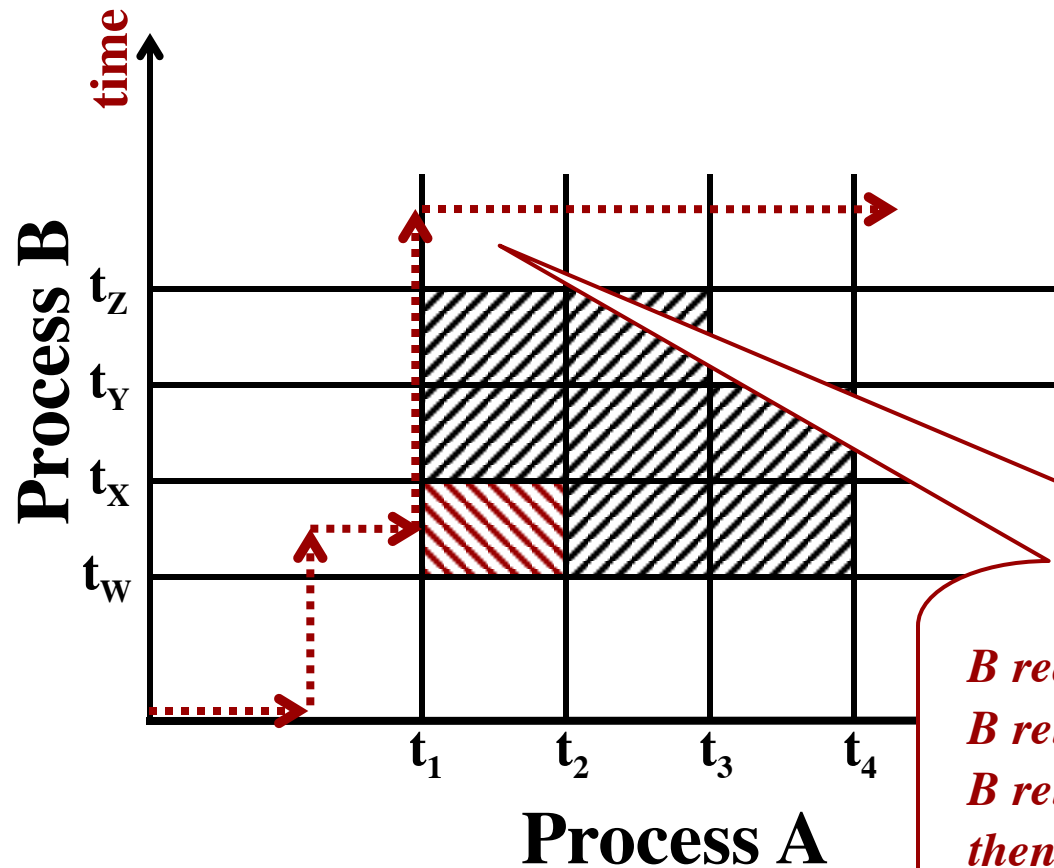
Avoidance using process-resource trajectories



Avoidance using process-resource trajectories



Avoidance using process-resource trajectories



*B requests the printer,
B releases CD-RW,
B releases printer,
then A runs to completion!*

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 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?

Avoidance with multiple resources

Total resource vector

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

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

Available resource vector

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

Maximum Request Vector

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

Row 2 is what process 2 might need

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

Banker's algorithm for multiple resources

- ❑ Look for a row, R , whose unmet resource needs are all smaller than or equal to A . If no such row exists, the system will eventually deadlock since no process can run to completion
- ❑ Assume the process of the row chosen requests all the resources that it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all its resources to A vector
- ❑ Repeat steps 1 and 2, until either all process are marked terminated, in which case the initial state was safe, or until deadlock occurs, in which case it was not

Avoidance with multiple resources

Total resource vector

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

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

Available resource vector

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

Maximum Request Vector

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

Row 2 is what process 2 might need

Run algorithm on every resource request!

Avoidance with multiple resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

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 with multiple resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

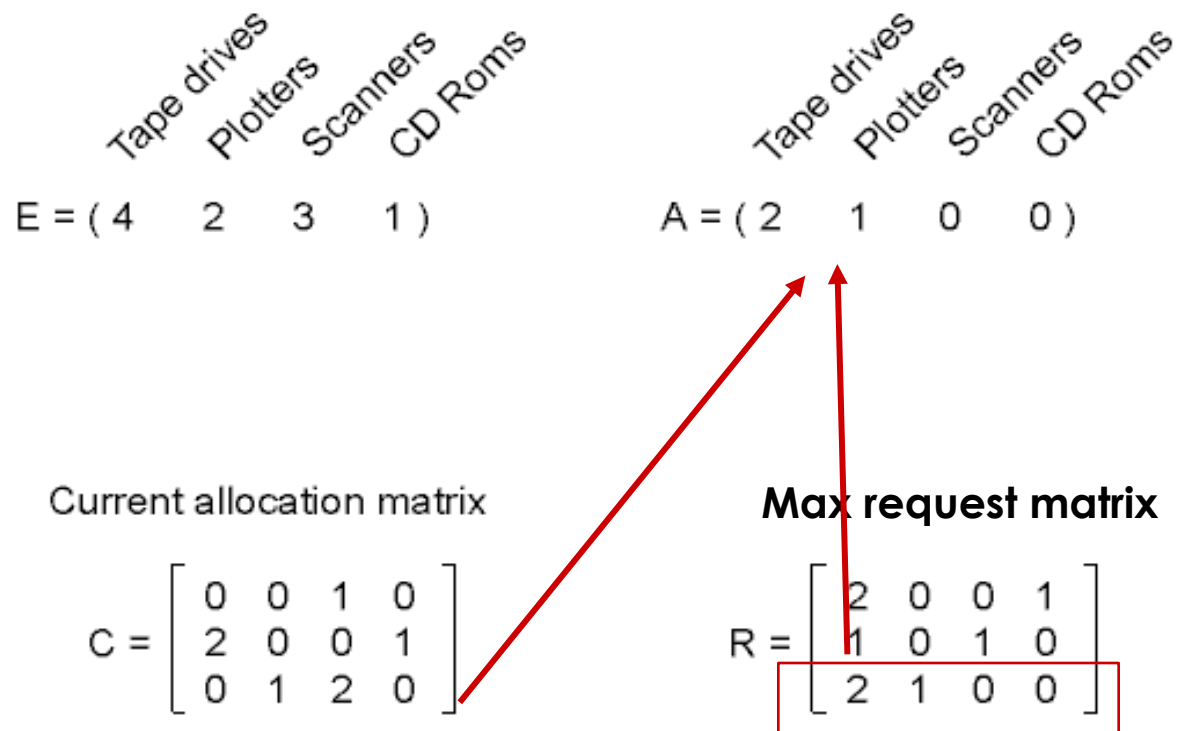
Current allocation matrix

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

Max request matrix

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

Avoidance with multiple resources



Avoidance with multiple resources

$$E = \begin{matrix} & \begin{matrix} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{matrix} \\ \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix} \end{matrix}$$

$$A = \begin{matrix} & \begin{matrix} \text{Tape drives} \\ \text{Plotters} \\ \text{Scanners} \\ \text{CD Roms} \end{matrix} \\ \begin{pmatrix} 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 \end{pmatrix} \end{matrix}$$

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 with multiple resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

Current allocation matrix

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

Max request matrix

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

Avoidance with multiple resources

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 \\ 4 & 2 & 2 & 1 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

Current allocation matrix

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

Max request matrix

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

Problems with deadlock avoidance

- ❑ **Deadlock avoidance is often impossible**
 - ❖ because you don't know in advance what resources a process will need!

- ❑ **Alternative approach “deadlock prevention”**
 - ❖ Make deadlock impossible!
 - ❖ Attack one of the four conditions that are necessary for deadlock to be possible