

---

# **Solution to The Sample Problem**

Lui Sha

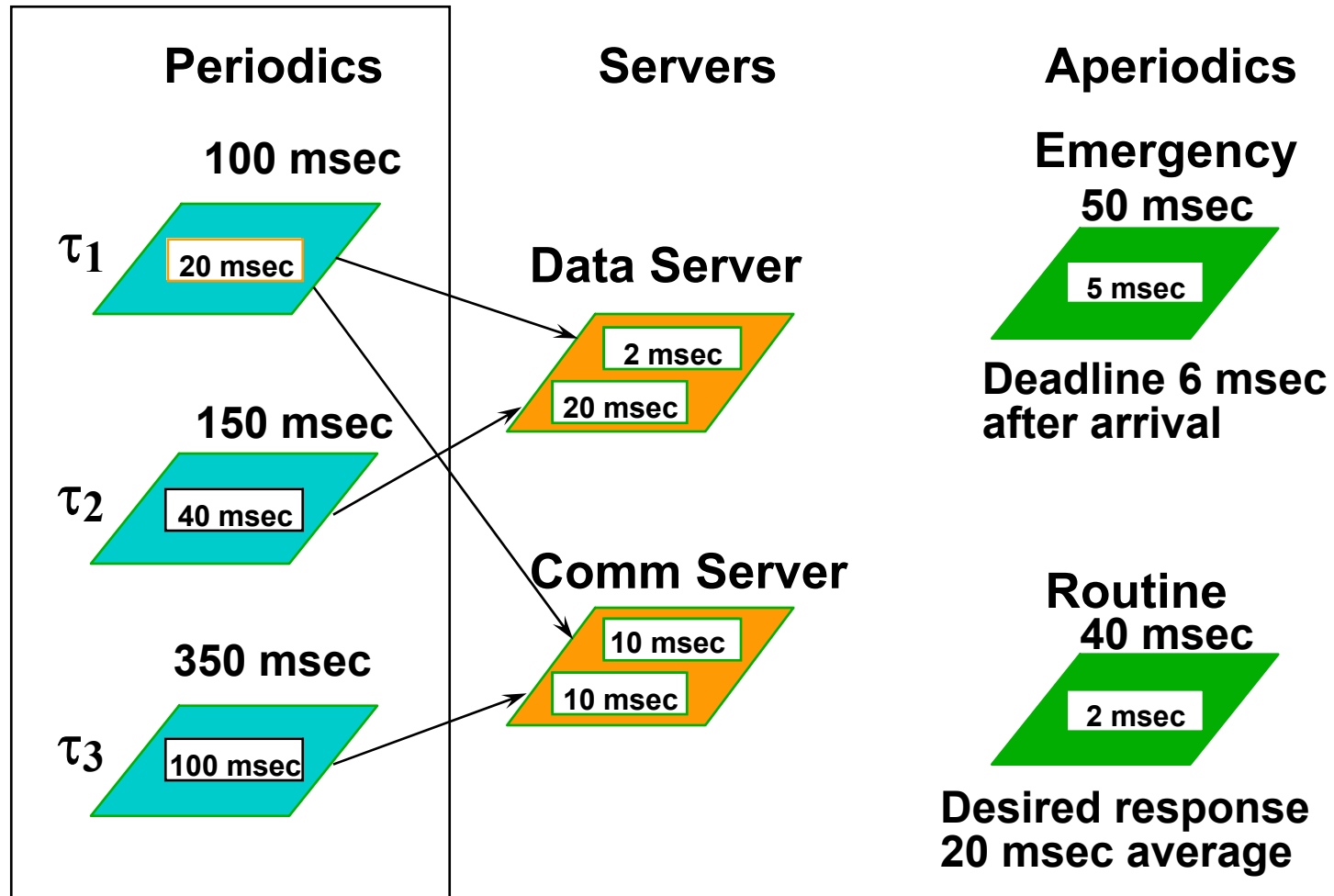
---

# Outlines

We have learned all the major components of generalized Rate Monotonic Scheduling Theory

We now put them together by solving the sample problem

# A Sample Problem



---

# Schedulability: UB Test

Utilization bound(UB) test: a set of  $n$  independent periodic tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasing, if

$$\frac{C_1}{T_1} + \dots + \frac{C_n}{T_n} \leq U(n) = n(2^{1/n} - 1)$$

$U(1) = 1.0$	$U(4) = 0.756$	$U(7) = 0.728$
$U(2) = 0.828$	$U(5) = 0.743$	$U(8) = 0.724$
$U(3) = 0.779$	$U(6) = 0.734$	$U(9) = 0.720$

For large  $n$ , the bound converges to  $\ln 2 \sim 0.69$ .

For harmonic task sets (in which the period of each task is a multiple of all higher frequency tasks), the utilization bound is  $U(n)=1.00$  for all  $n$ .

Conventions, task 1 has shorter period than task 2 and so on.

---

## Sample Problem: Applying UB Test

	C	T	U
Task $\tau_1$ :	20	100	0.200
Task $\tau_2$ :	40	150	0.267
Task $\tau_3$ :	100	350	0.286

Total utilization is  $.200 + .267 + .286 = .753 < U(3) = .779$

The periodic tasks in the sample problem are schedulable according to the UB test.

---

# Toward a More Precise Test

UB test has three possible outcomes:

- $0 \leq U \leq U(n) \implies \textit{Success}$
- $U(n) < U \leq 1.00 \implies \textit{Inconclusive}$
- $1.00 < U \implies \textit{Overload}$

UB test is conservative.

---

## Example: Applying Exact Test

Is the task set still schedulable if we increase the computation time of  $\tau_1$  from 20 to 40?

Utilization of first two tasks:  $0.667 < U(2) = 0.828$

- The first two tasks are schedulable by UB test

Utilization of all three tasks:  $0.953 > U(3) = 0.779$

- The UB test is inconclusive
- We need to apply exact test

# Schedulability: Exact Test

Intuition: let  $t = a_0$  be the instance at which task  $\tau_i$  and all higher priority task execute once.

If there is no new arrival from higher priority tasks during  $a_0$ ,  $\tau_i$  actually completes its execution at  $t = a_0$ . If there is new arrivals, the compute  $a_1$  and check if there is new arrivals...

The arrivals are counted by the ceiling function.

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \quad \text{where } a_0 = \sum_{j=1}^i C_j$$

Test terminates when  $a_{n+1} > T_i$  (not schedulable)  
or when  $a_{n+1} = a_n \leq T_i$  (schedulable).



## Example: Applying Exact Test -2

**Use exact test to determine if  $\tau_3$  meets its first deadline:**

$$a_0 = \sum_{j=1}^3 c_j = c_1 + c_2 + c_3 = 40 + 40 + 100 = 180$$

$$\begin{aligned} a_1 &= c_3 + \sum_{j=1}^2 \left\lceil \frac{a_0}{T_j} \right\rceil c_j \\ &= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260 \end{aligned}$$

## Example: Applying the Exact Test -3

$$a_2 = C_3 + \sum_{j=1}^2 \left\lceil \frac{a_1}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

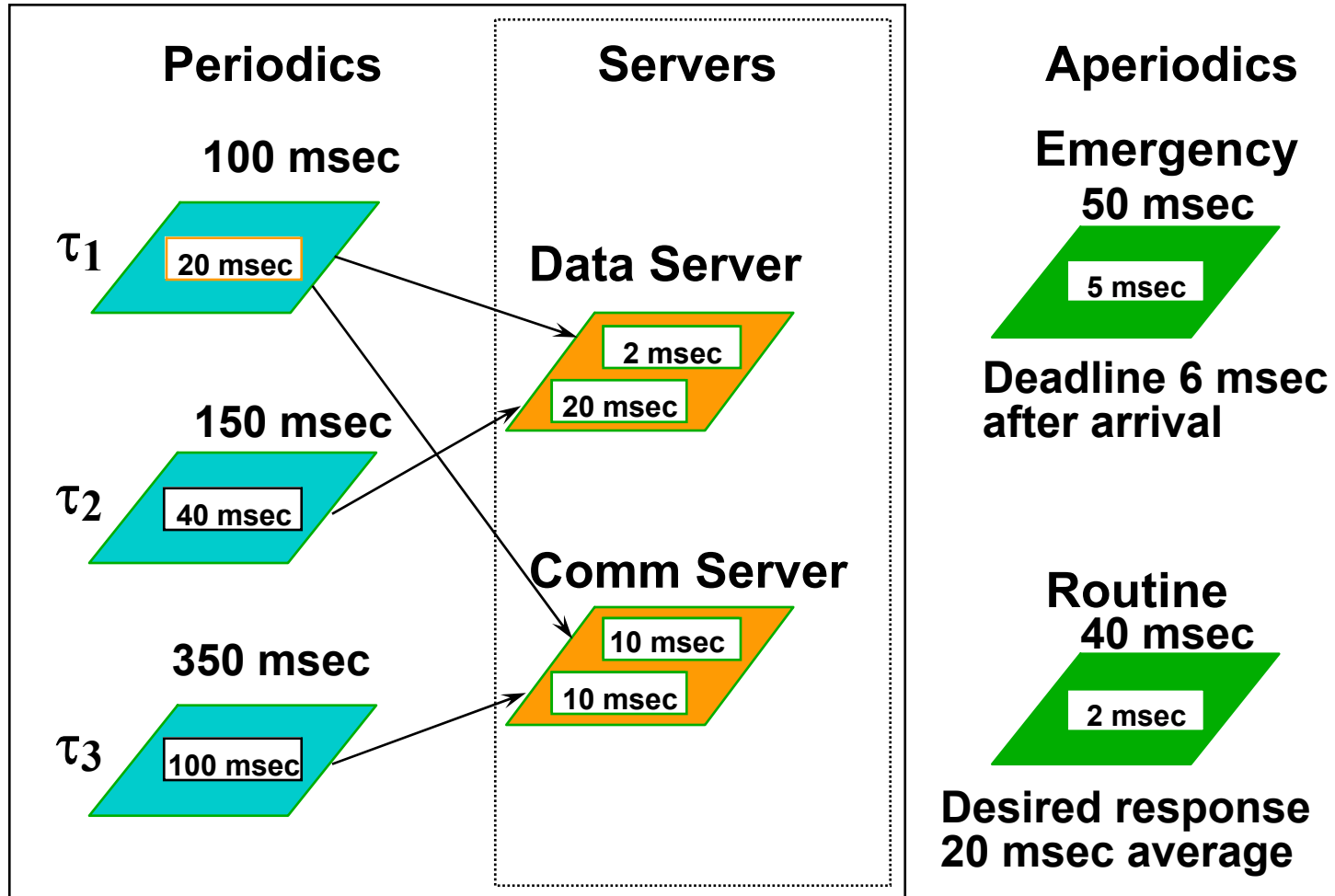
$$a_3 = C_3 + \sum_{j=1}^2 \left\lceil \frac{a_2}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$$a_3 = a_2 = 300 \quad \text{Done!}$$

**Task  $\tau_3$  is schedulable using exact test**

$$a_3 = 300 < T = 350$$

# A Sample Problem



---

# Priority Inversion

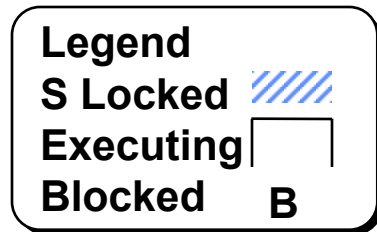
Ideally, under prioritized preemptive scheduling, higher priority tasks should immediately preempt lower priority tasks.

When lower priority tasks causing higher priority tasks to wait due to the locking of shared data, a priority inversion is said to occur.

It seems reasonable to expect the duration of a priority inversion (also called blocking time) to be a function of the duration of the critical sections.

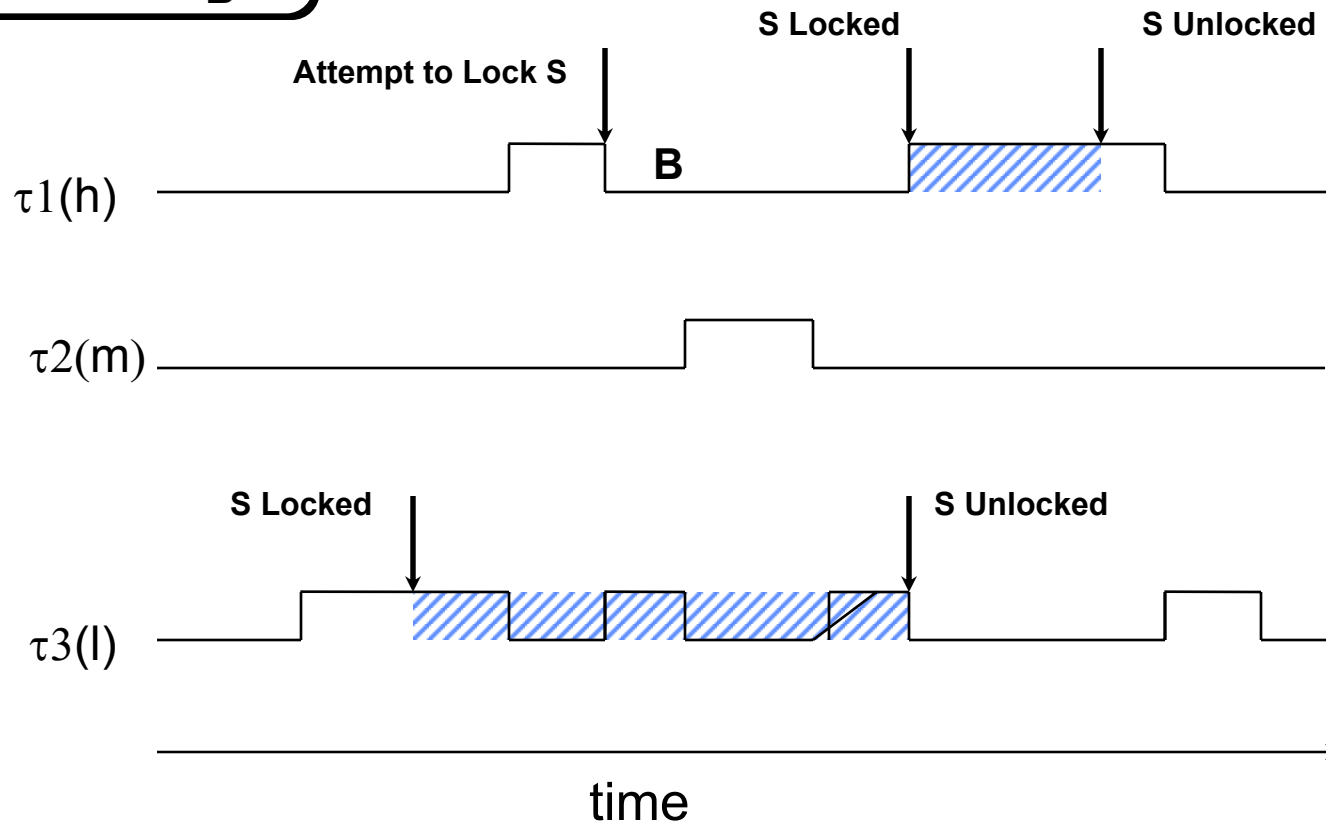
Critical section: the duration of a task using shared resource.

# Unbounded Priority Inversion



$\tau1:\{\dots P(S)\dots V(S)\dots\}$

$\tau3:\{\dots P(S)\dots V(S)\dots\}$



---

# Basic Priority Inheritance Protocol

Let the lower priority task use the priority of the blocked higher priority task.

In this way, medium priority tasks can no longer preempt the low priority task (which the high priority task is waiting on).

Priority inheritance is transitive.

# Basic Priority Inheritance Protocol

## Legend

S Locked



Executing

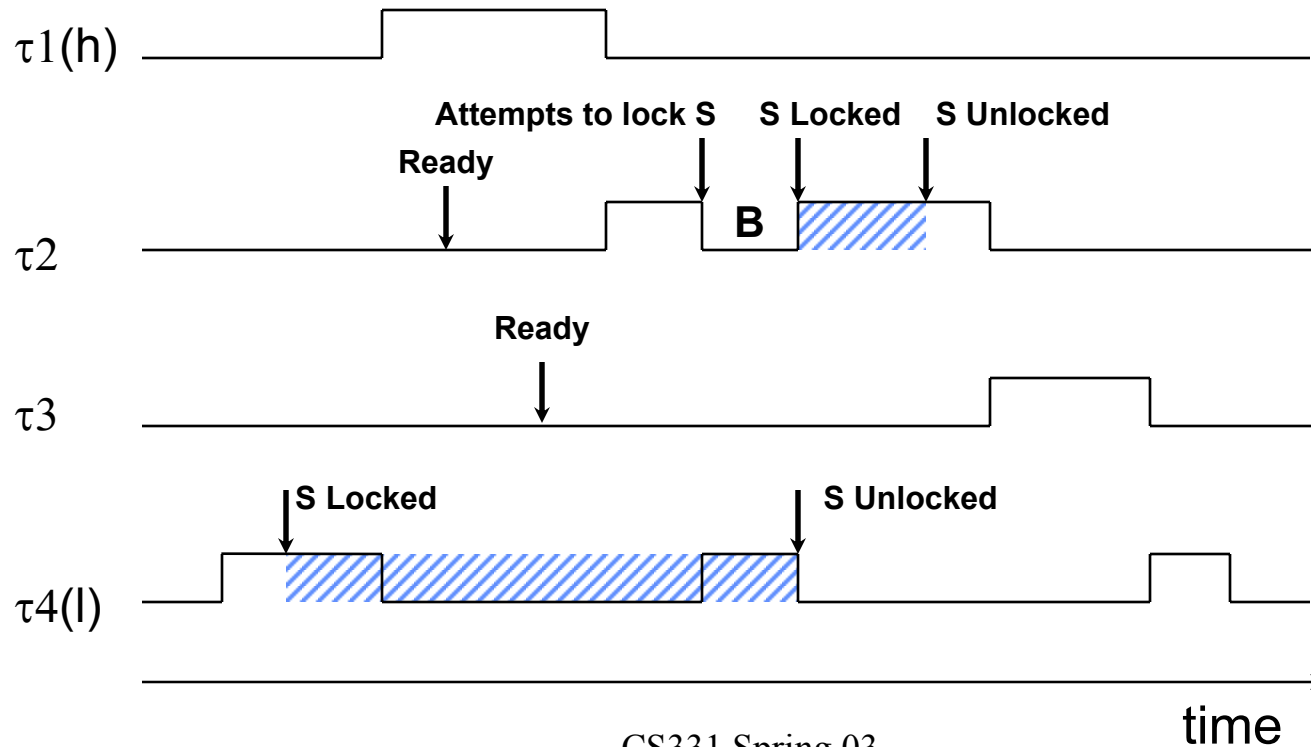


Blocked

B

$\tau_2: \{ \dots P(S) \dots V(S) \dots \}$

$\tau_4: \{ \dots P(S) \dots V(S) \dots \}$



---

# Property of Basic Priority Inheritance

OS developers can support it without knowing application priorities.

There will be no deadlock if there are no nested locks, or an application-level deadlock avoidance scheme (such the ordering of resource) is used.

Chained priority is fact of life. But a task is blocked by at most  $n$  lower priority tasks sharing resources with it, when there is no deadlock.

Priority inheritance protocol is supported by almost all of the real time OS and is part of the POSIX real-time extensions.



---

# Priority Ceiling Protocol

A priority ceiling is assigned to each semaphore, which is equal to the highest priority task that may use this semaphore.



A task can lock a semaphore if and only if its priority is higher than the priority ceilings of all locked semaphores that are already locked by other tasks.

If a task is blocked by lower priority tasks, the lower priority task inherits its priority.

PCP is now an option in the POSIX real-time extensions.

# Blocked at Most Once (PCP)

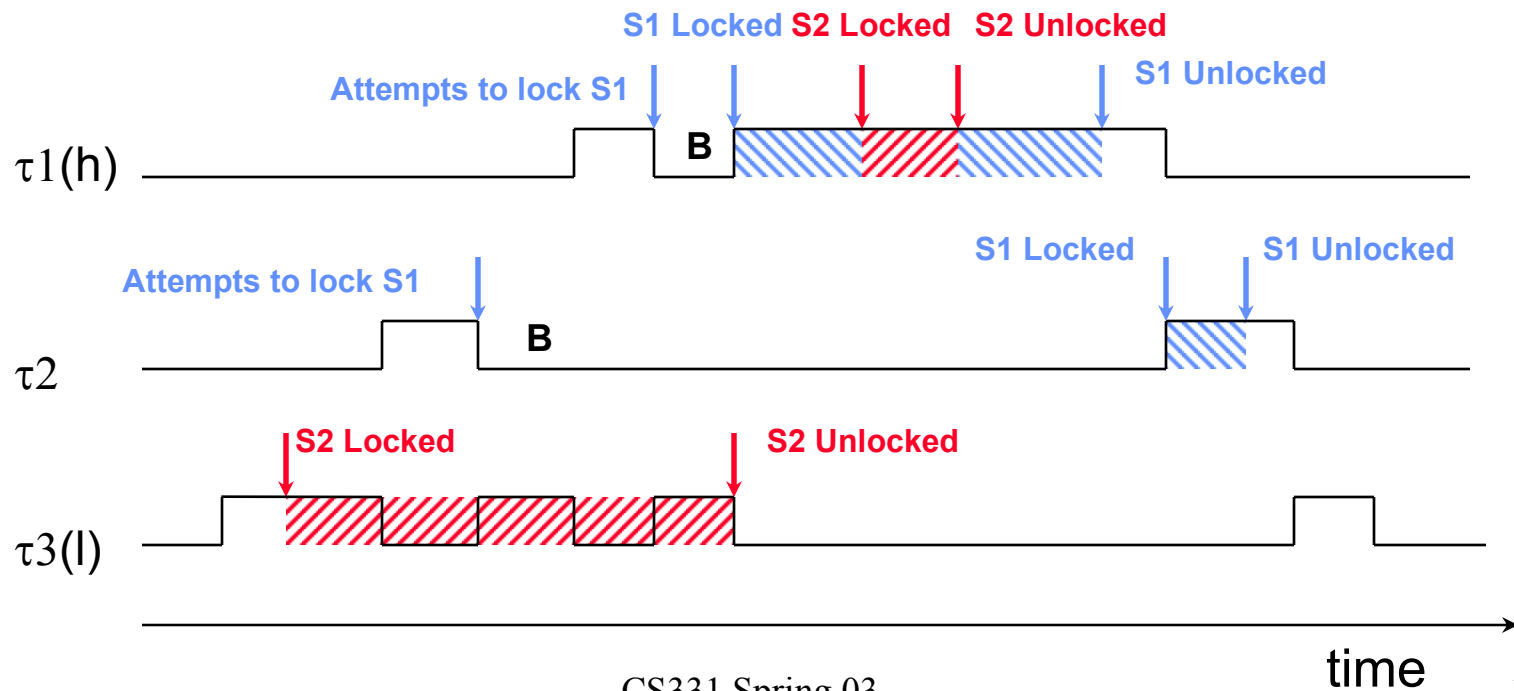
## Legend

**S1 Locked**   
**S2 Locked**   
**Executing**   
**Blocked** **B**

$\tau_1: \{ \dots P(S1) \dots P(S2) \dots V(S2) \dots V(S1) \dots \}$

$\tau_2: \{ \dots P(S1) \dots V(S1) \dots \}$

$\tau_3: \{ \dots P(S2) \dots V(S2) \dots \}$



# Deadlock Avoidance: Using PCP

## Legend

S1 Locked 

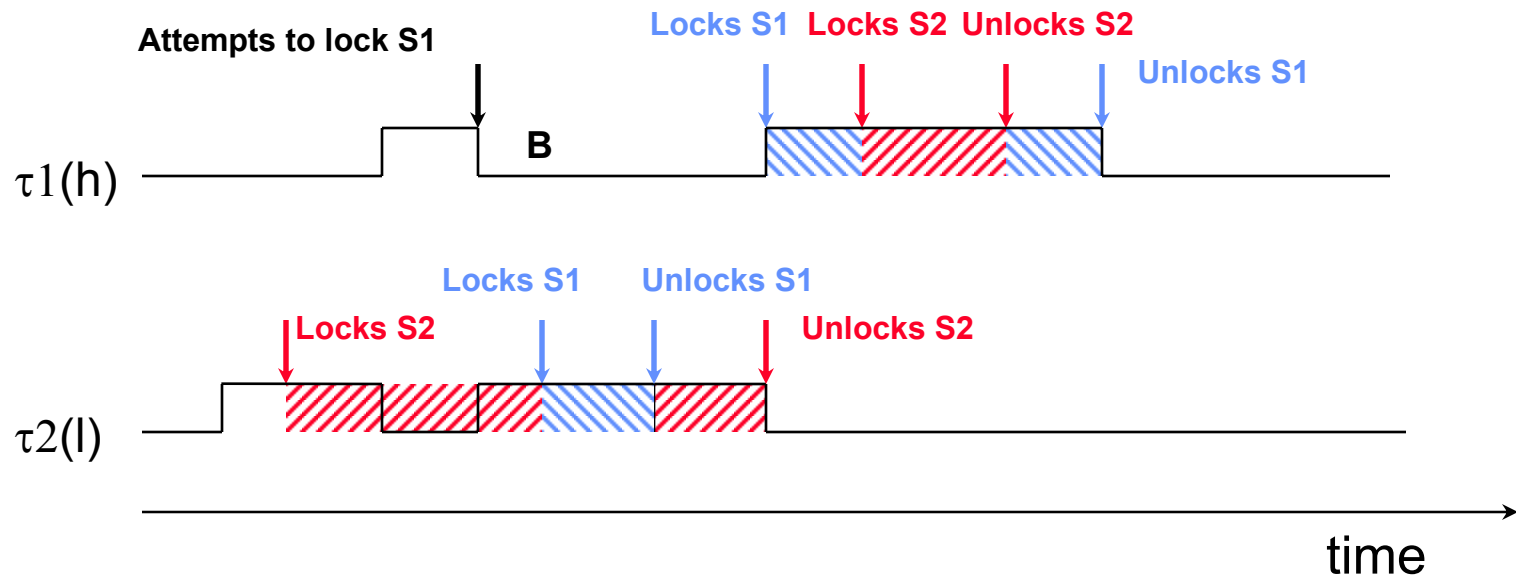
S2 Locked 

Executing 

Blocked 

$\tau1: \{ \dots P(S1) \dots P(S2) \dots V(S2) \dots V(S1) \dots \}$

$\tau2: \{ \dots P(S2) \dots P(S1) \dots V(S1) \dots V(S2) \dots \}$



# Schedulability Analysis

## A uni-processor equation using BIP

preemption      execution      blocking

$$\forall i, 1 \leq i \leq n, \sum_{j=1}^{i-1} \frac{c_j}{T_j} + \frac{c_i + (b_{i+1} + \dots + b_n)}{T_i} \leq i(2^{1/i} - 1)$$

## A uni-processor equation using PCP

preemption      execution      blocking

$$\forall i, 1 \leq i \leq n, \sum_{j=1}^{i-1} \frac{c_j}{T_j} + \frac{c_i + \max(b_{i+1}, \dots, b_n)}{T_i} \leq i(2^{1/i} - 1)$$

---

## Sample Problem: Using BIP

	C	T	D	B
$\tau_1$	20	100		30
$\tau_2$	40	150	20	10
$\tau_3$	100	350		

---

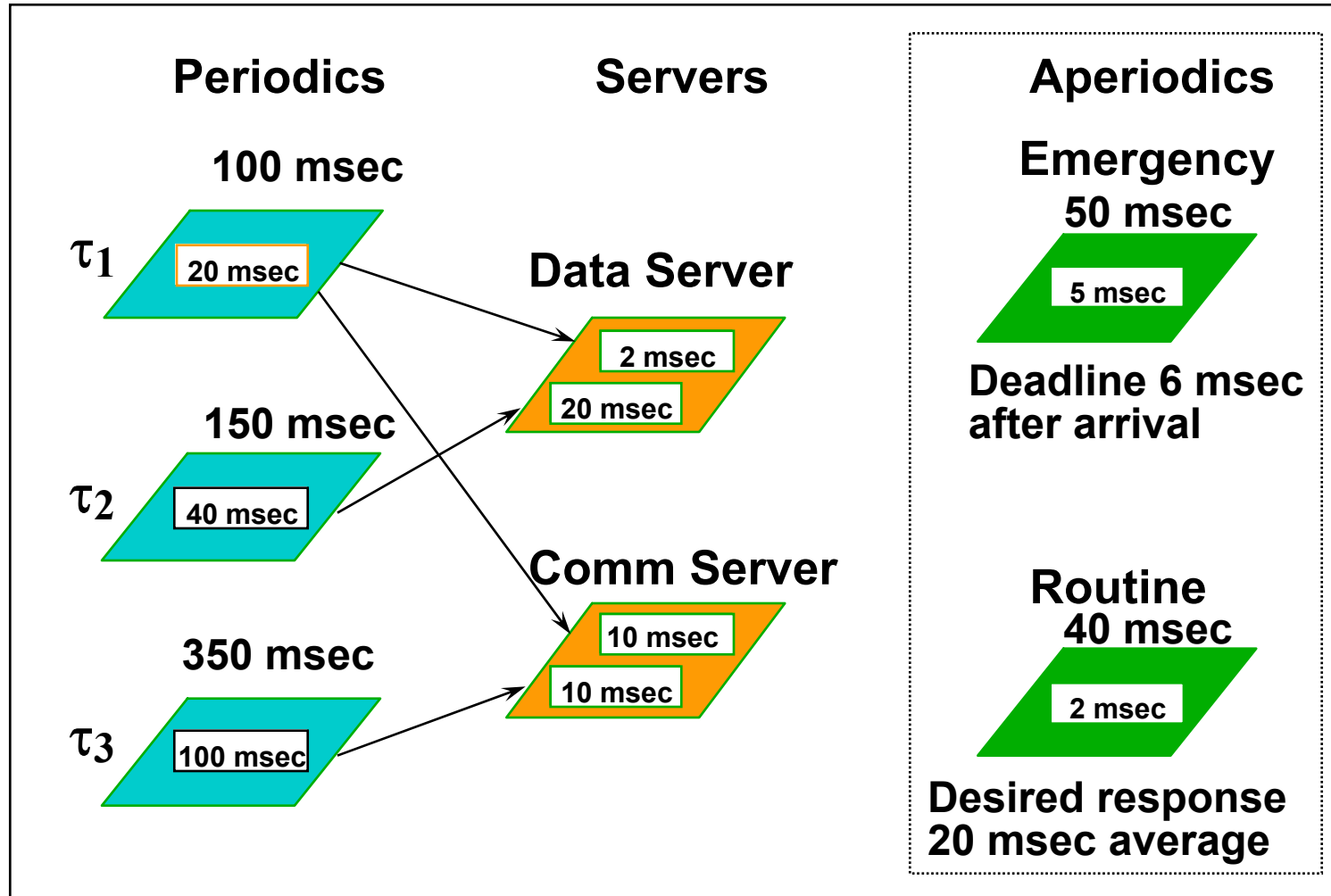
# Schedulability Model Using BIP

$$\frac{C_1}{T_1} + \frac{B_1}{T_1} \leq U(1) \quad \frac{20}{100} + \frac{30}{100} = 0.50 < 1.0$$

$$\frac{C_1}{T_1} + \frac{C_2 + D_2}{T_2} + \frac{B_2}{T_2} \leq U(2) \quad \frac{20}{100} + \frac{40 + 20}{150} + \frac{10}{150} = 0.667 < 0.828$$

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} \leq U(3) \quad \frac{20}{100} + \frac{40}{150} + \frac{100}{350} = 0.753 < 0.779$$

# A Sample Problem



---

# Concepts and Definitions

Aperiodic task: runs at irregular intervals.

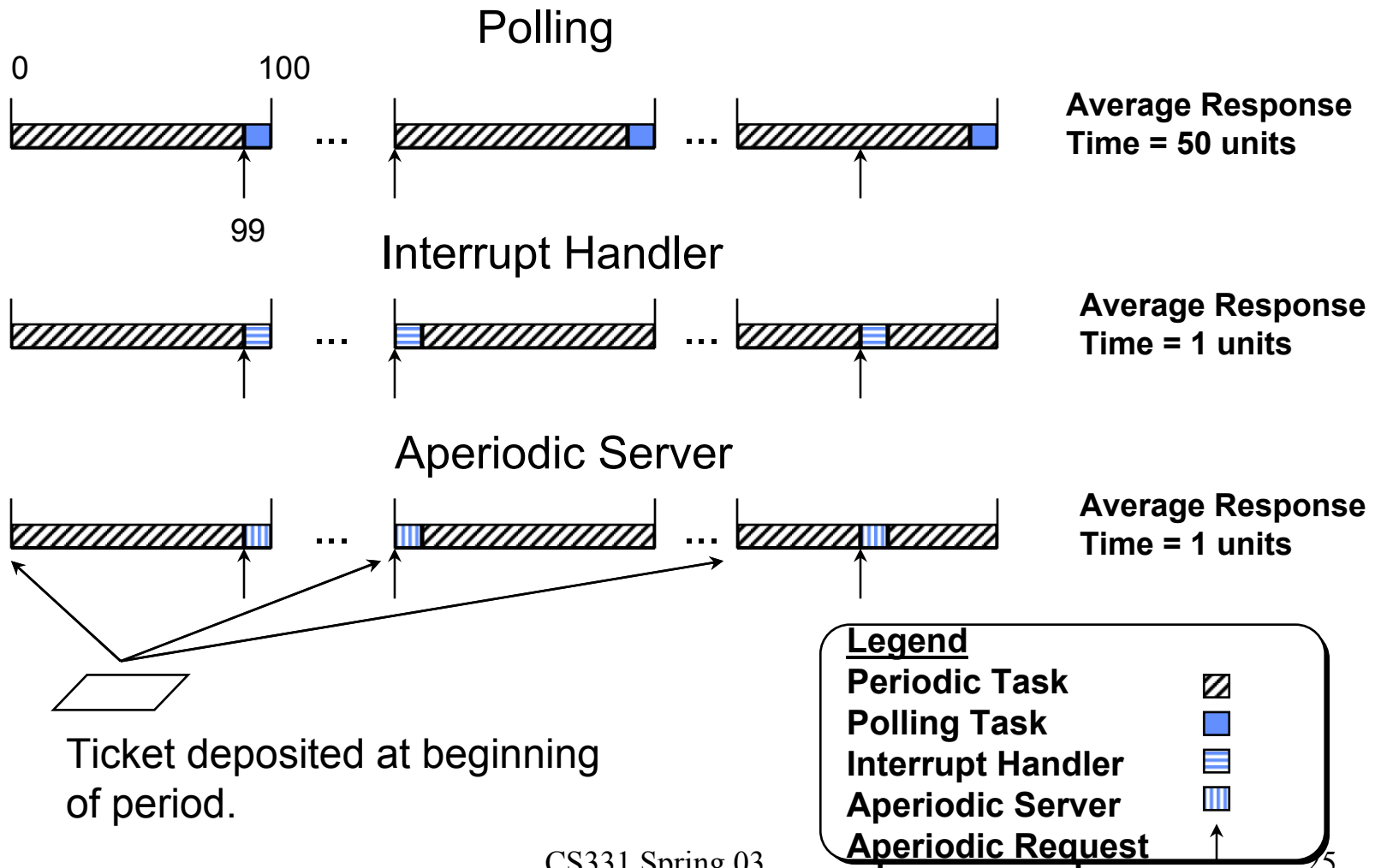
Aperiodic deadline:

Hard: minimum interarrival time

Soft: best average response



# Scheduling Aperiodic Tasks



# Sporadic Server (SS)

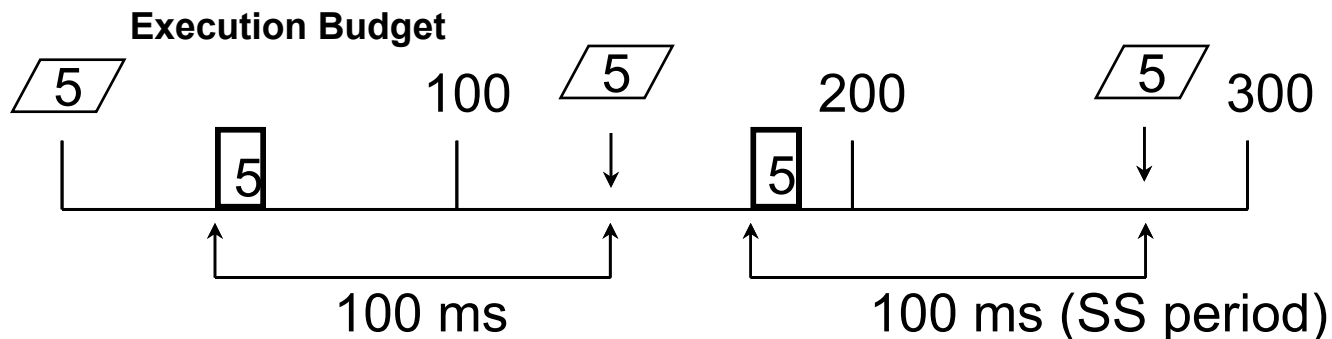
Modeled as periodic tasks

Fixed execution budget (C)

Replenishment interval (T)

Priority is based on T, which can be adjusted to meet the requirements.

Replenishment occurs one “period” after start of use.



---

# Sample Problems: Aperiodic - 1

## Emergency Server (ES)

- Execution Budget:  $C = 5$
- Replenish Interval:  $T = 50$
- Pre-period Deadline:  $D = 44$

## General Aperiodic Server (GS)

Design guideline: Give it as high a priority as possible and as many “tickets” as possible, without causing periodic tasks to miss their deadlines:

- Execution Budget:  $C = 10$
- Replenish Interval:  $T = 100$

---

# Aperiodic - 2

Simulation and queuing theory using M/M/1 approximation indicates that the average response time is 4 msec.

The server budget:  $10/100 = 0.1$

The workload:  $2/40 = 0.05$

Server's workload:  $0.05/0.1 = 0.5$

The average response time:

$$\begin{aligned} & (\text{Average Execution Time}) / (1 - \text{Workload}) \\ & = 2 / (1 - 0.5) = 4 \end{aligned}$$

---

# Summary

We have reviewed

- the basic concepts of real time computing
- the basics of GRMS theory
  - Independent tasks
  - synchronization
  - aperiodic tasks

*"Through the development of [Generalized] Rate Monotonic Scheduling, we now have a system that will allow [Space Station] Freedom's computers to budget their time, to choose between a variety of tasks, and decide not only which one to do first but how much time to spend in the process"*

--- Aaron Cohen, former deputy administrator of NASA, "Charting The Future: Challenges and Promises Ahead of Space Exploration", October, 28, 1992, p. 3.