CS428 Embedded Systems: Dealing with Real Time

Slides from: Intel Higher Education Forum, Embedded Systems Course

http://pixel01.cps.intel.com/education/highered/Embedded/Embedded.htm

Lecture 15

Plan for Lectures

- **Introduction to Real-Time Systems**
  - **Examples**
  - **Terminology, Metrics**
  - **Scheduling Policies**
- Rate-Monotonic Analysis (RMA)
  - Fundamental concepts
  - An Introduction to Rate-Monotonic Analysis: independent tasks
  - Present basic theory for periodic task sets
- Extend basic theory to include
  - Context switch overhead, Interrupts
  - Preperiod deadlines
- Consider task interactions
  - Priority inversion
  - Synchronization protocols (time allowing)
- Extend theory to aperiodic tasks
  - Sporadic servers (time allowing)

Real-time System

- A **real-time system** is a system whose specification includes both logical and temporal correctness requirements.
  - **Logical Correctness:** Produces correct outputs.
    - Can be checked, for example, by Hoare logic.
  - **Temporal Correctness:** Produces outputs at the right time.
    - It is not enough to say that "brakes were applied"
    - You want to be able to say "brakes were applied at the right time"
      - In this course, we spend much time on techniques for checking temporal correctness.
      - The question of how to specify temporal requirements, though enormously important, is shortchanged in this course.

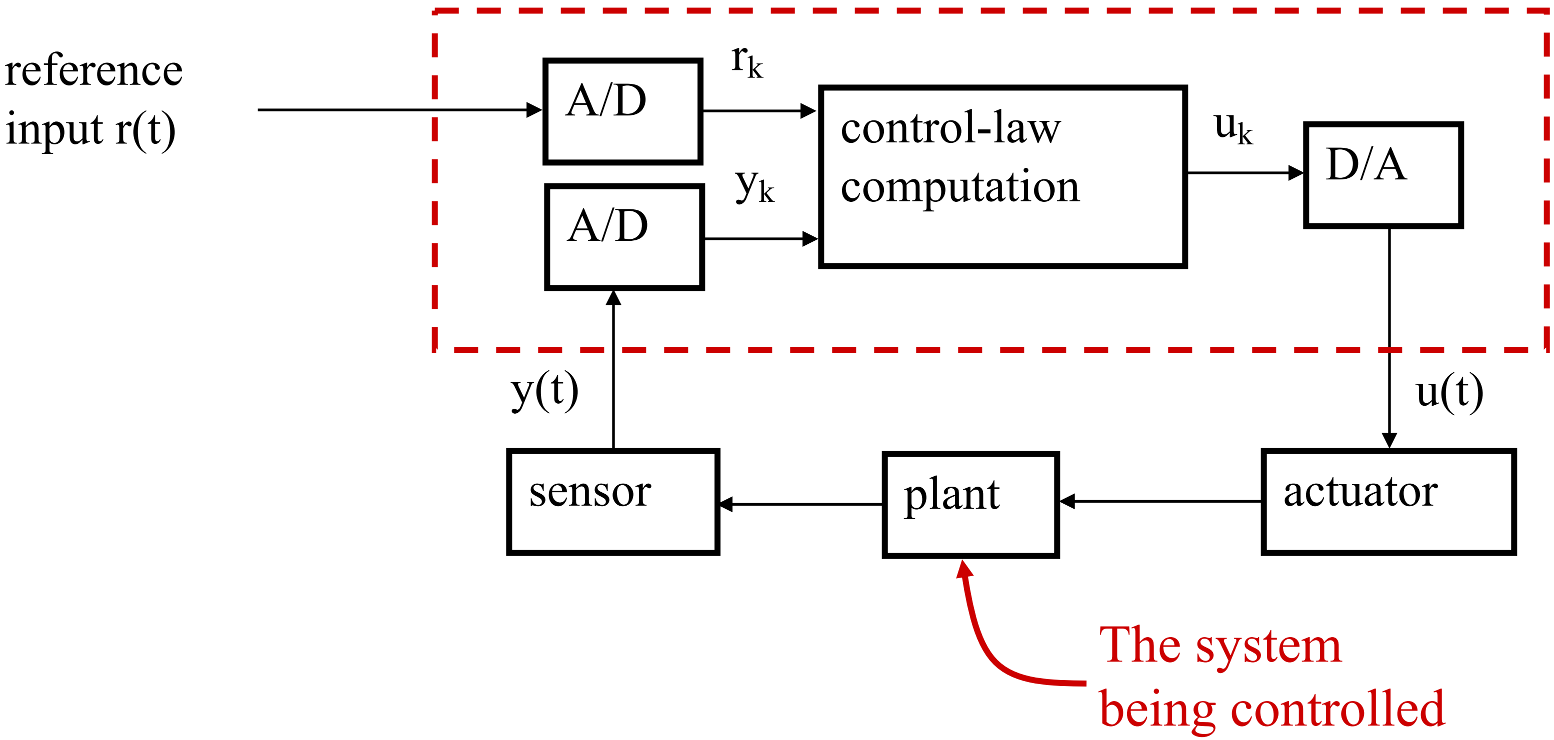
Characteristics of Real-Time Systems

- Event-driven, reactive.
- Concurrency/multiprogramming.
- Stand-alone/continuous operation.
- Reliability/fault-tolerance requirements.
- **Predictable behavior.**

Example Real-Time Applications

Many real-time systems are **control systems**.

Example 1: A simple one-sensor, one-actuator control system.



Simple Control System (cont'd)

Pseudo-code for this system:

```
set timer to interrupt periodically with period T;
at each timer interrupt do
        do analog-to-digital conversion to get y;
        compute control output u;
        output u and do digital-to-analog conversion;
end do
```

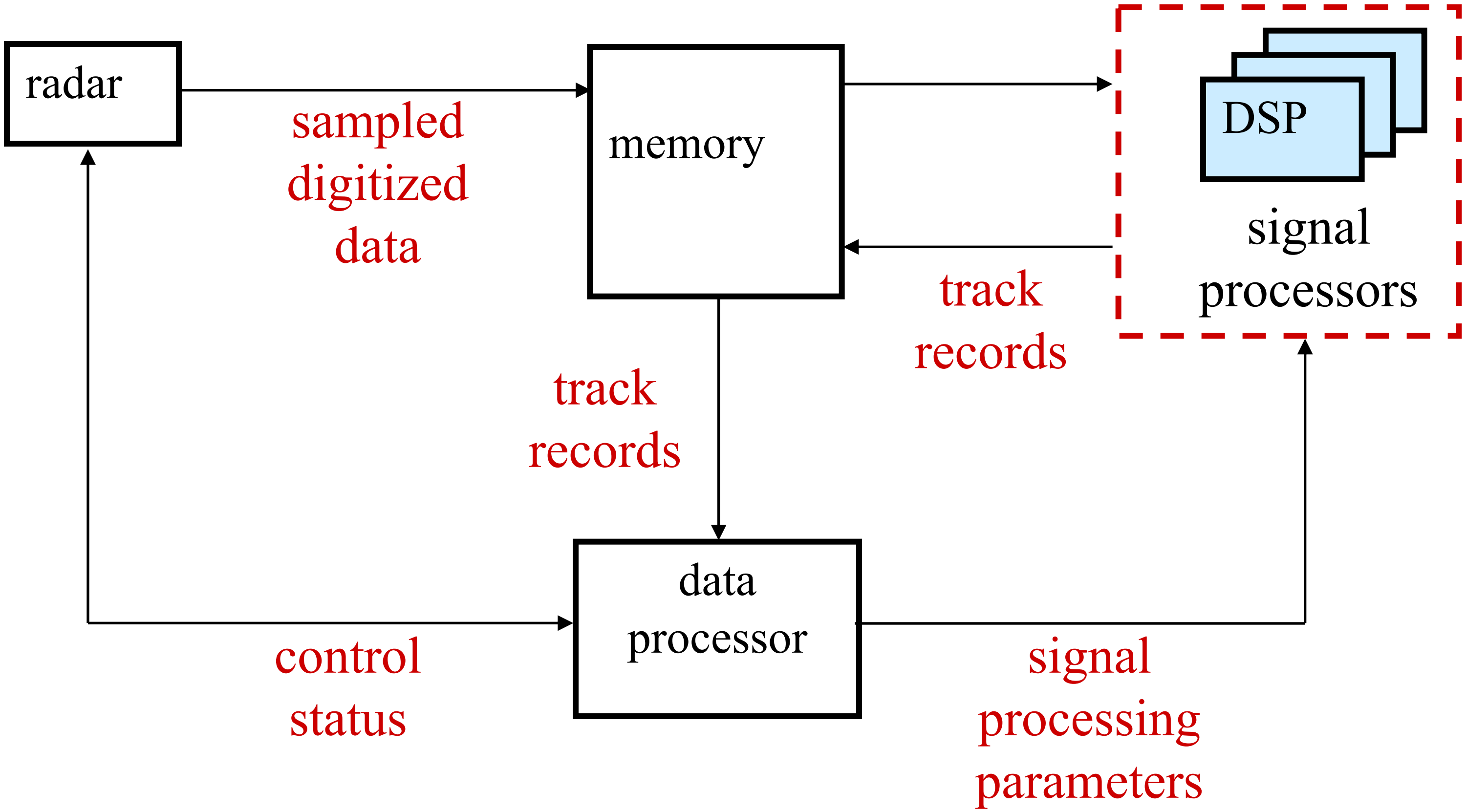
T is called the **sampling period**. *T* is a key design choice. *T*ypical range for *T*: seconds to milliseconds.

Signal-Processing Systems

Signal-processing systems transform data from one form to another.

- **Examples:**
  - Digital filtering.
  - Video and voice compression/decompression.
  - Radar signal processing.
- Response times range from a few milliseconds to a few seconds.

Example: Radar System



Other Real-Time Applications

- **Real-time databases.**
  - Transactions must complete by deadlines.
  - **Main dilemma**: Transaction scheduling algorithms and real-time scheduling algorithms often have conflicting goals.
  - Data may be subject to **absolute** and **relative temporal consistency** requirements.
- **Multimedia.**
  - Want to process audio and video frames at steady rates.
    - TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
    - Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec.
  - **Other requirements**: Lip synchronization, low jitter, low end-to-end response times (if interactive).

Are *All* Systems Real-Time Systems?

- **Question:** Is a payroll processing system a real-time system?
  - It has a time constraint: Print the pay checks every two weeks.

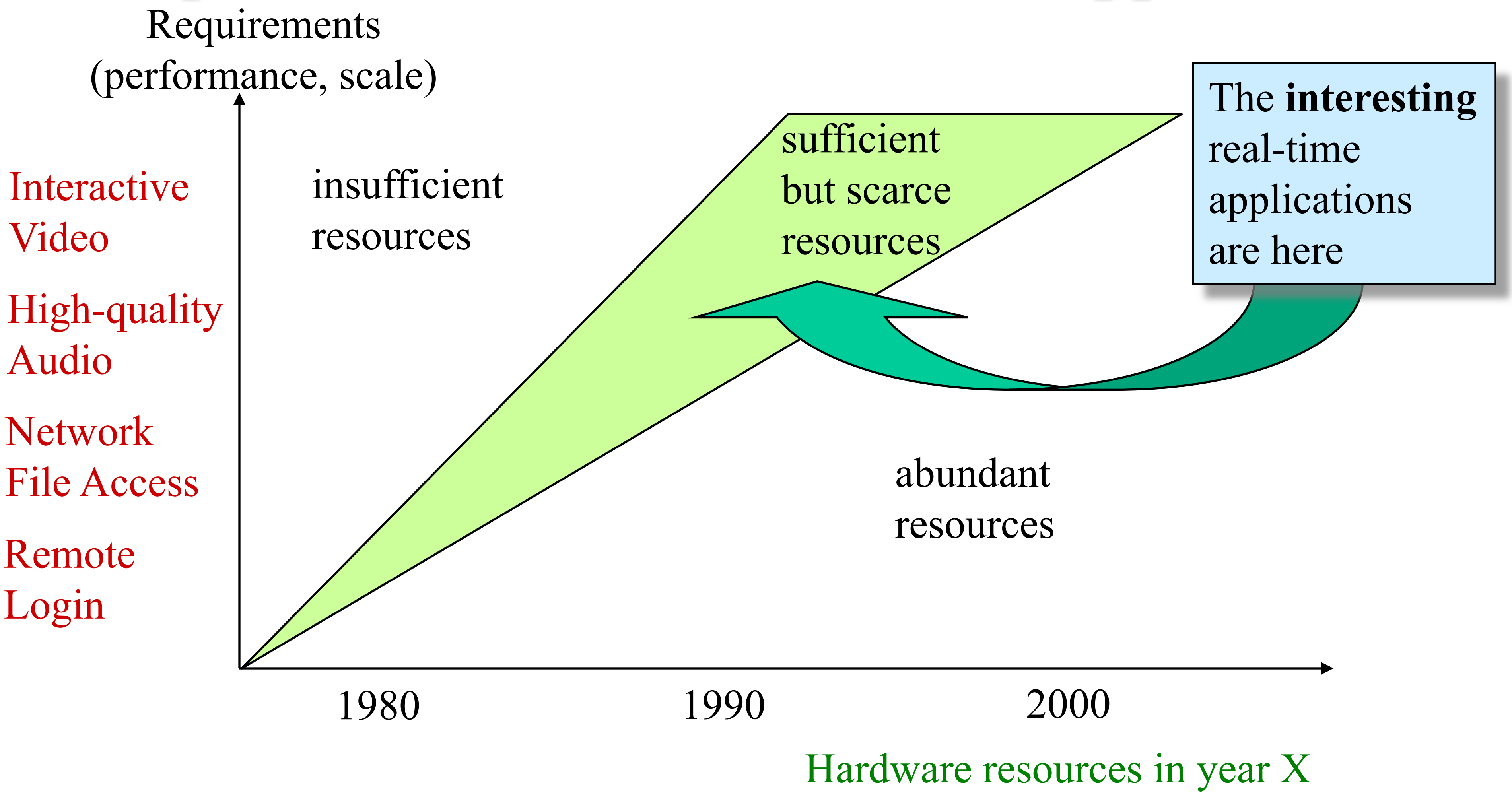
- Perhaps it is a real-time system in a definitional sense, but it doesn't pay us to view it as such.

- We are interested in systems for which it is not *a priori* obvious how to meet timing constraints.

The "Window of Scarcity"

- Resources may be categorized as:
  - **Abundant:** Virtually any system design methodology can be used to realize the timing requirements of the application.
  - **Insufficient:** The application is ahead of the technology curve; no design methodology can be used to realize the timing requirements of the application.
  - **Sufficient but scarce:** It is possible to realize the timing requirements of the application, but careful resource allocation is required.

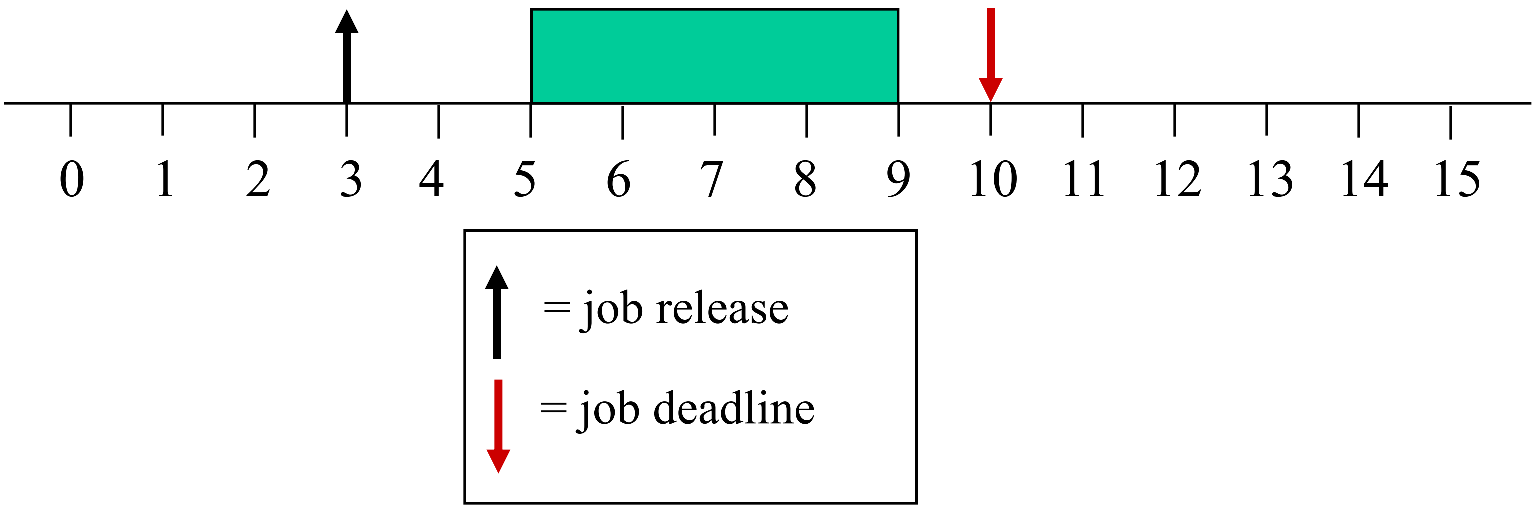
Example: Interactive/Multimedia Applications



Hard vs. *Soft* Real Time

- **Task:** A sequential piece of code.
- **Job:** Instance of a task.
- Jobs require **resources** to execute.
  - **Example resources:** CPU, network, disk, critical section.
  - We will simply call all hardware resources "processors".
- **Release time of a job:** The time instant the job becomes ready to execute.
- **Absolute Deadline of a job:** The time instant by which the job must complete execution.
- **Relative deadline of a job:** "Deadline – Release time".
- **Response time of a job:** "Completion time – Release time".

Example



- Job is released at time 3.
- Its (absolute) deadline is at time 10.
- Its relative deadline is 7.
- Its response time is 6.

Hard Real-Time Systems

- A **hard deadline** *must* be met.
  - If *any* hard deadline is *ever* missed, then the system is **incorrect**.
  - Requires a means for **validating** that deadlines are met.
- **Hard real-time system:** A real-time system in which all deadlines are hard.
  - We mostly consider hard real-time systems in this course.
- **Examples:** Nuclear power plant control, flight control.

Soft Real-Time Systems

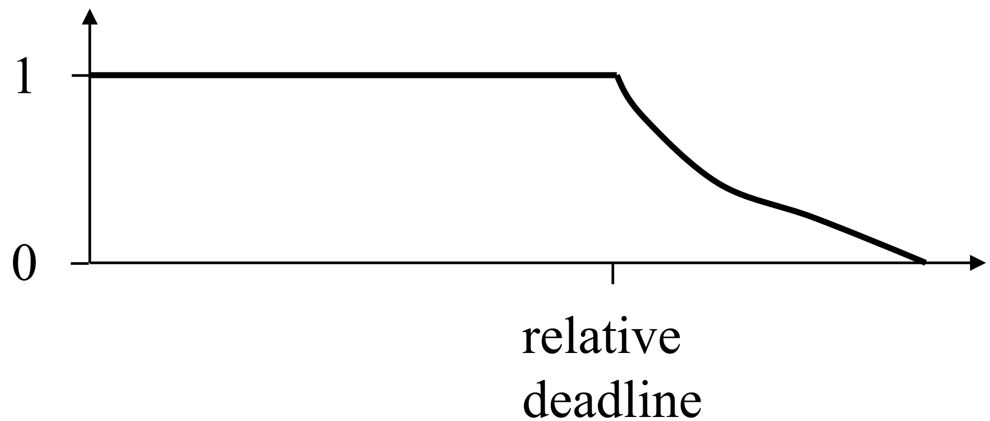
- A **soft deadline** may *occasionally* be missed.
  - **Question:** How to define "occasionally"?

- **Soft real-time system:** A real-time system in which some deadlines are soft.

- **Examples:** Telephone switches, multimedia applications.

Defining “Occasionally”

- **One Approach:** Use probabilistic requirements.
  - For example, 99% of deadlines will be met.
- **Another Approach:** Define a “usefulness” function for each job:



- **Note:** Validation is trickier here.

Reference Model

- Each job J_i is characterized by its **<u>release time</u>** r_i, **<u>absolute deadline</u>** d_i, **<u>relative deadline</u>** D_i, and **<u>execution time</u>** e_i.
  - Sometimes a range of release times is specified: $[r_i^-, r_i^+]$. This range is called **<u>release-time jitter</u>**.
- Likewise, sometimes instead of e_i, execution time is specified to range over $[e_i^-, e_i^+]$.
  - **<u>Note:</u>** It can be difficult to get a precise estimate of e_i (more on this later).

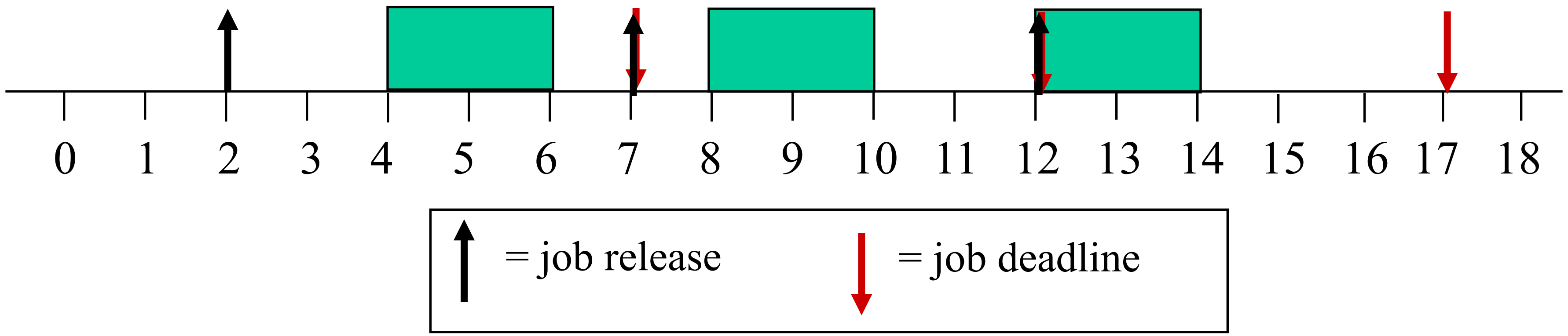
Periodic, Sporadic, Aperiodic Tasks

- **Periodic task:**
  - We associate a **period p_i** with each task T_i.
  - p_i is the interval between job releases.

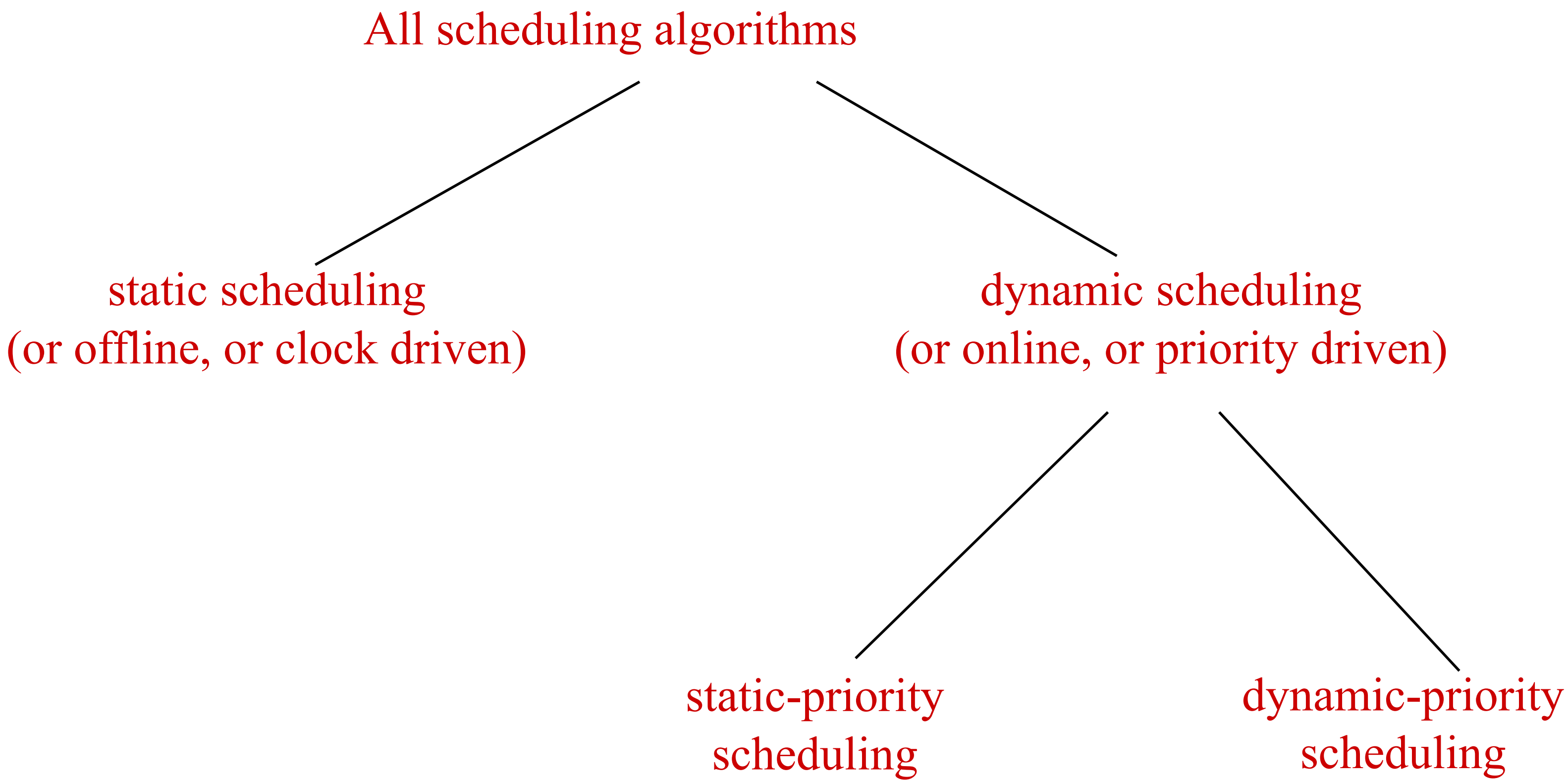
- **Sporadic and Aperiodic tasks:** Released at arbitrary times.
  - **Sporadic:** Has a hard deadline.
  - **Aperiodic:** Has no deadline or a soft deadline.

Examples

A periodic task T_i with $r_i = 2$, $p_i = 5$, $e_i = 2$, $D_i = 5$ executes like this:



Classification of Scheduling Algorithms



Summary of Lecture So Far

- Real-time Systems
  - characteristics and mis-conceptions
  - the "window of scarcity"
- Example real-time systems
  - simple control systems
  - multi-rate control systems
  - hierarchical control systems
  - signal processing systems
- Terminology
- Scheduling algorithms

Plan for Lectures

- Introduction to Real-Time Systems
  - Examples
  - Terminology, Metrics
  - Scheduling Policies
- **Rate-Monotonic Analysis (RMA)**
  - **Fundamental concepts**
  - **An Introduction to Rate-Monotonic Analysis: independent tasks**
  - **Present basic theory for periodic task sets**
- Extend basic theory to include
  - Context switch overhead, Interrupts
  - Preperiod deadlines
- Consider task interactions
  - Priority inversion
  - Synchronization protocols (time allowing)
- Extend theory to aperiodic tasks
  - Sporadic servers (time allowing)

What's Important in Real-Time

Metrics for real-time systems differ from that for time-sharing systems.

	Time-Sharing Systems	Real-Time Systems
Capacity	High throughput	Schedulability
Responsiveness	Fast average response	Ensured worst-case response
Overload	Fairness	Stability

- schedulability is the ability of tasks to meet all hard deadlines
- latency is the worst-case system response time to events
- stability in overload means the system meets critical deadlines even if all deadlines cannot be met

Scheduling Policies

- CPU scheduling policy: a rule to select task to run next
  - cyclic executive
  - rate monotonic/deadline monotonic
  - earliest deadline first
  - least laxity first
- Assume preemptive, priority scheduling of tasks
  - Analyze effects of non-preemption later
- Rate monotonic analysis
  - based on rate monotonic scheduling theory
  - analytic formulas to determine schedulability
  - framework for reasoning about system timing behavior
  - separation of timing and functional concerns
- Provides an engineering basis for designing real-time systems

Rate Monotonic Scheduling (RMS)

- Priorities of periodic tasks are based on their rates: highest rate gets highest priority.
- Theoretical basis
  - optimal fixed scheduling policy (when deadlines are at end of period)
  - analytic formulas to check schedulability
- Must distinguish between scheduling and analysis
  - rate monotonic scheduling forms the basis for rate monotonic analysis
  - however, we consider later how to analyze systems in which rate monotonic scheduling is not used
  - any scheduling approach may be used, but all real-time systems should be analyzed for timing

Rate Monotonic Analysis (RMA)

- Rate-monotonic analysis is a set of mathematical techniques for analyzing sets of real-time tasks.
- Basic theory applies only to independent, periodic tasks, but has been extended to address
  - priority inversion
  - task interactions
  - aperiodic tasks
- Focus is on RM**A**, not RM**S**

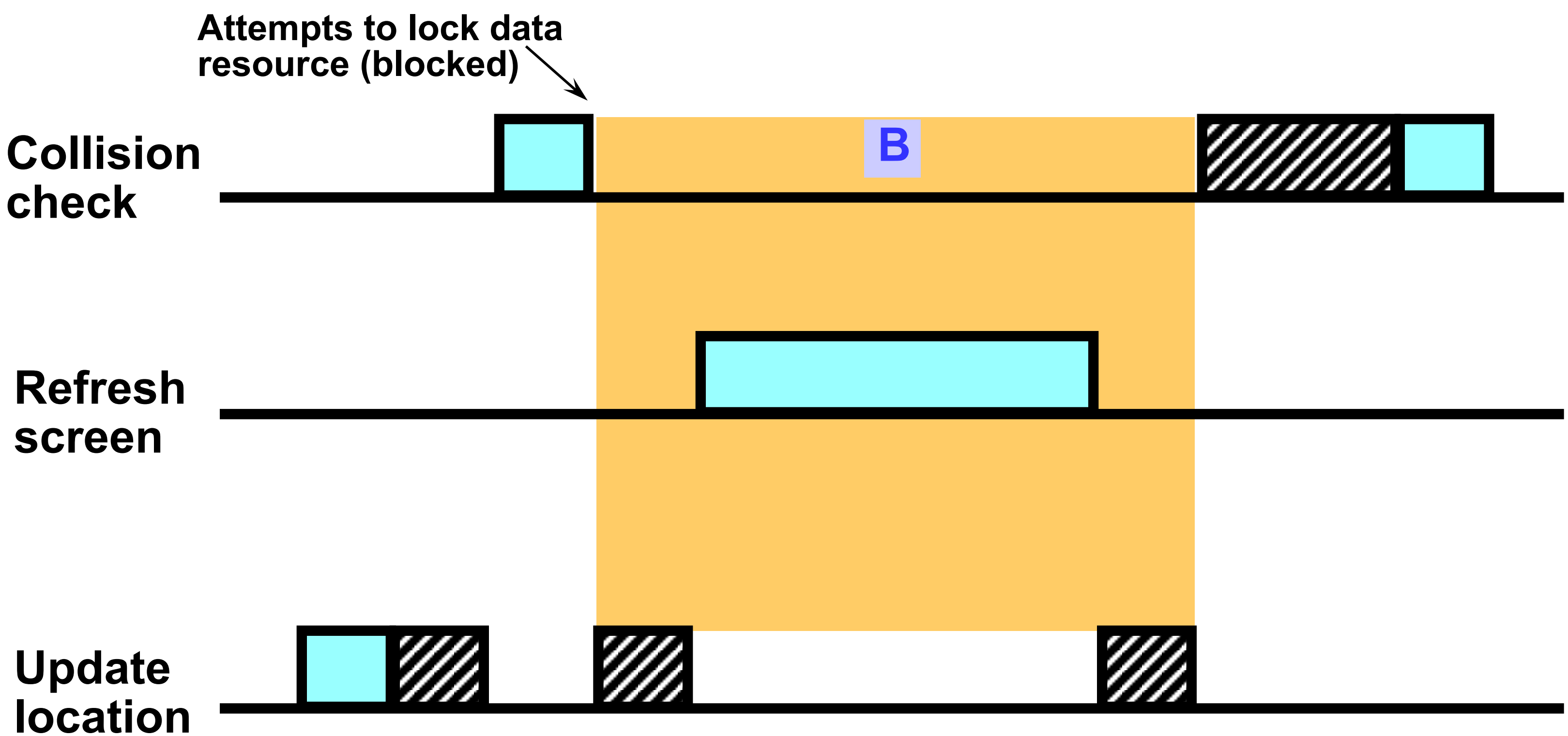
Why Are Deadlines Missed?

- For a given task, consider
  - **preemption**: time waiting for higher priority tasks
  - **execution**: time to do its own work
  - **blocking**: time delayed by lower priority tasks
- The task is schedulable if the sum of its preemption, execution, and blocking is less than its deadline.
- **Focus**: identify the biggest hits among the three and reduce, as needed, to achieve schedulability

Example of Priority Inversion

Collision check: {… P ( ) … V ( ) …}

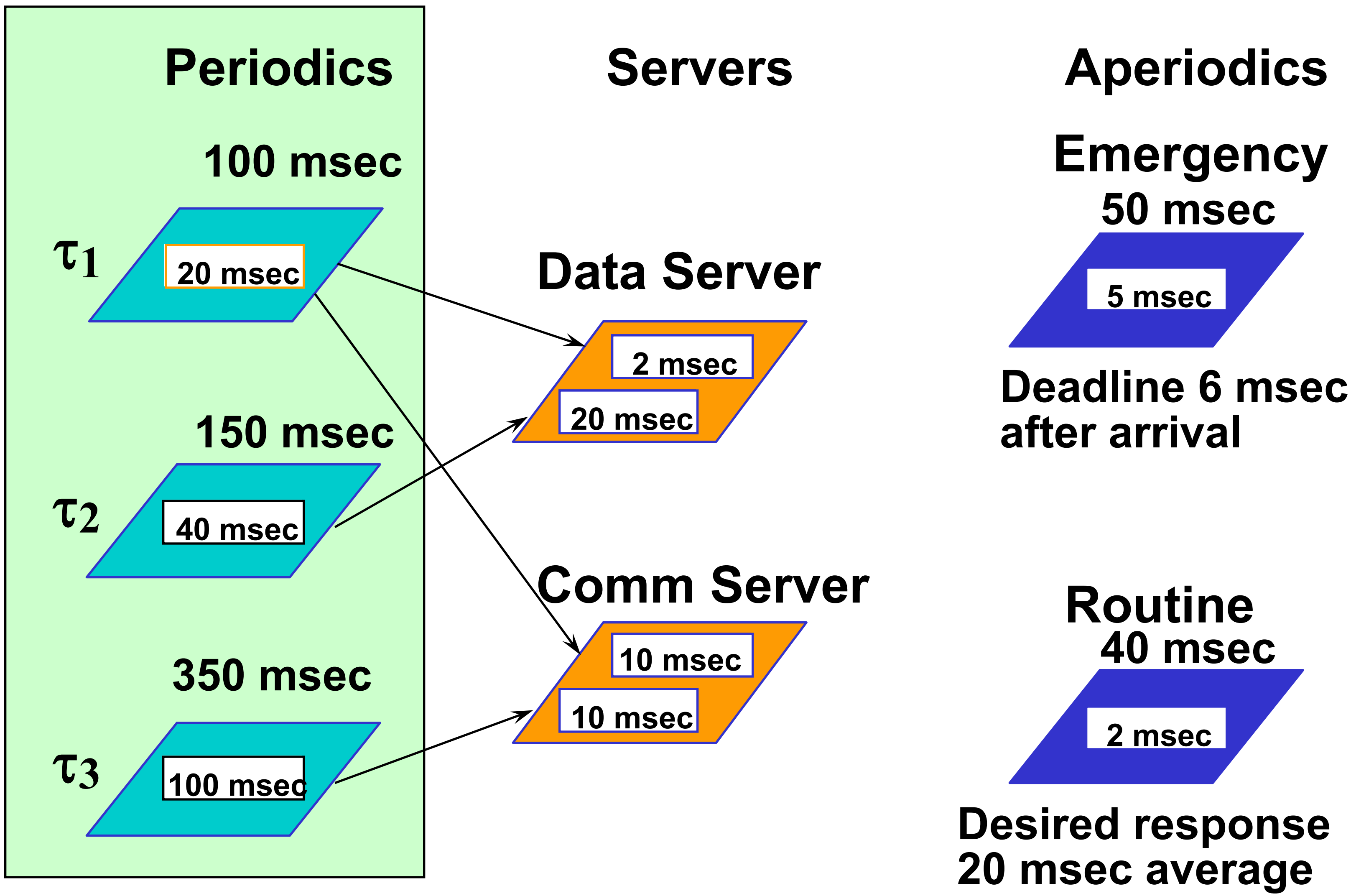
Update location: {… P ( ) … V ( ) …}



Rate Monotonic Theory - Experience

- Supported by several standards
  - POSIX Real-time Extensions
    - Various real-time versions of Linux
  - Java (Real-Time Specification for Java and Distributed Real-Time Specification for Java)
  - Real-Time CORBA
  - Real-Time UML
  - Ada 83 and Ada 95
  - Windows 95/98
  - …

A Sample Problem - Periodics



τ_2's deadline is 20 msec before the end of each period

Concepts and Definitions - Periodics

- Periodic task
  - initiated at fixed intervals
  - must finish before start of next cycle

- Task's CPU utilization:

$$U_i = \frac{C_i}{T_i}$$

  - C_i = worst-case compute time (execution time) for task τ_i
  - T_i = period of task τ_i

- CPU utilization for a set of tasks

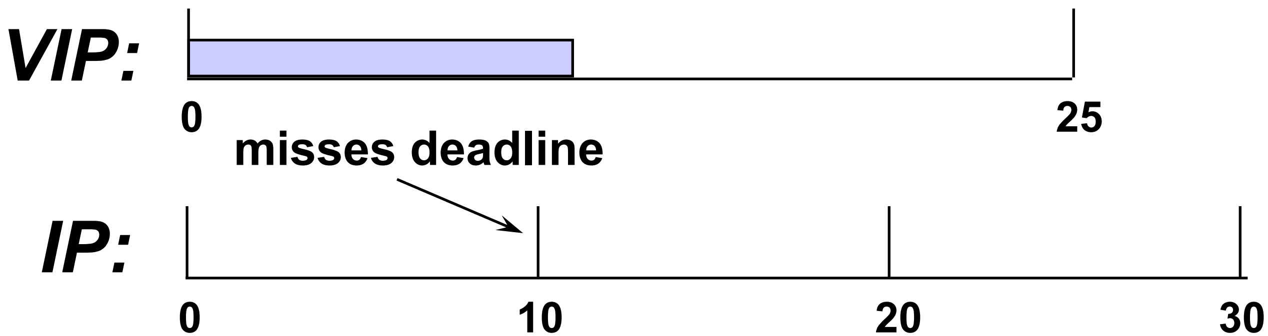
$$U = U_1 + U_2 + \ldots + U_n$$

Example of Priority Assignment

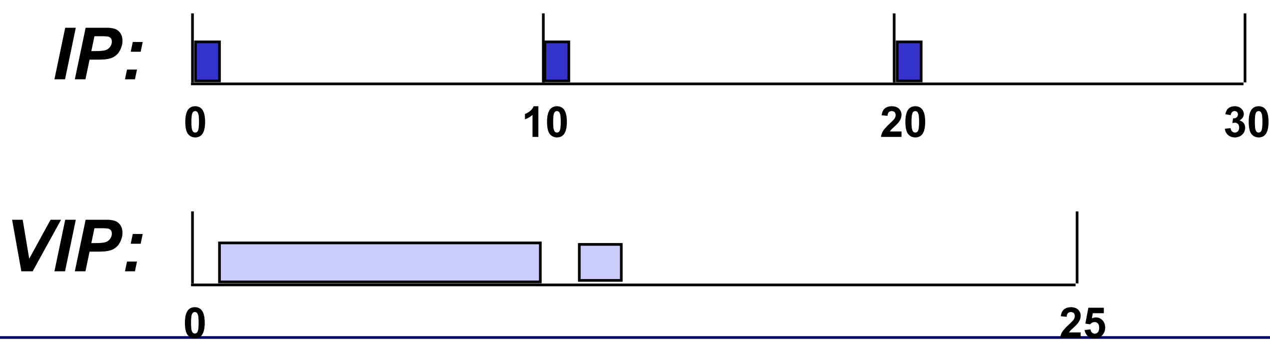
IP: $U_{IP} = \frac{1}{10} = 0.10$

VIP: $U_{VIP} = \frac{11}{25} = 0.44$

Semantics-Based Priority Assignment



Policy-Based Priority Assignment



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 phasings, if

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

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

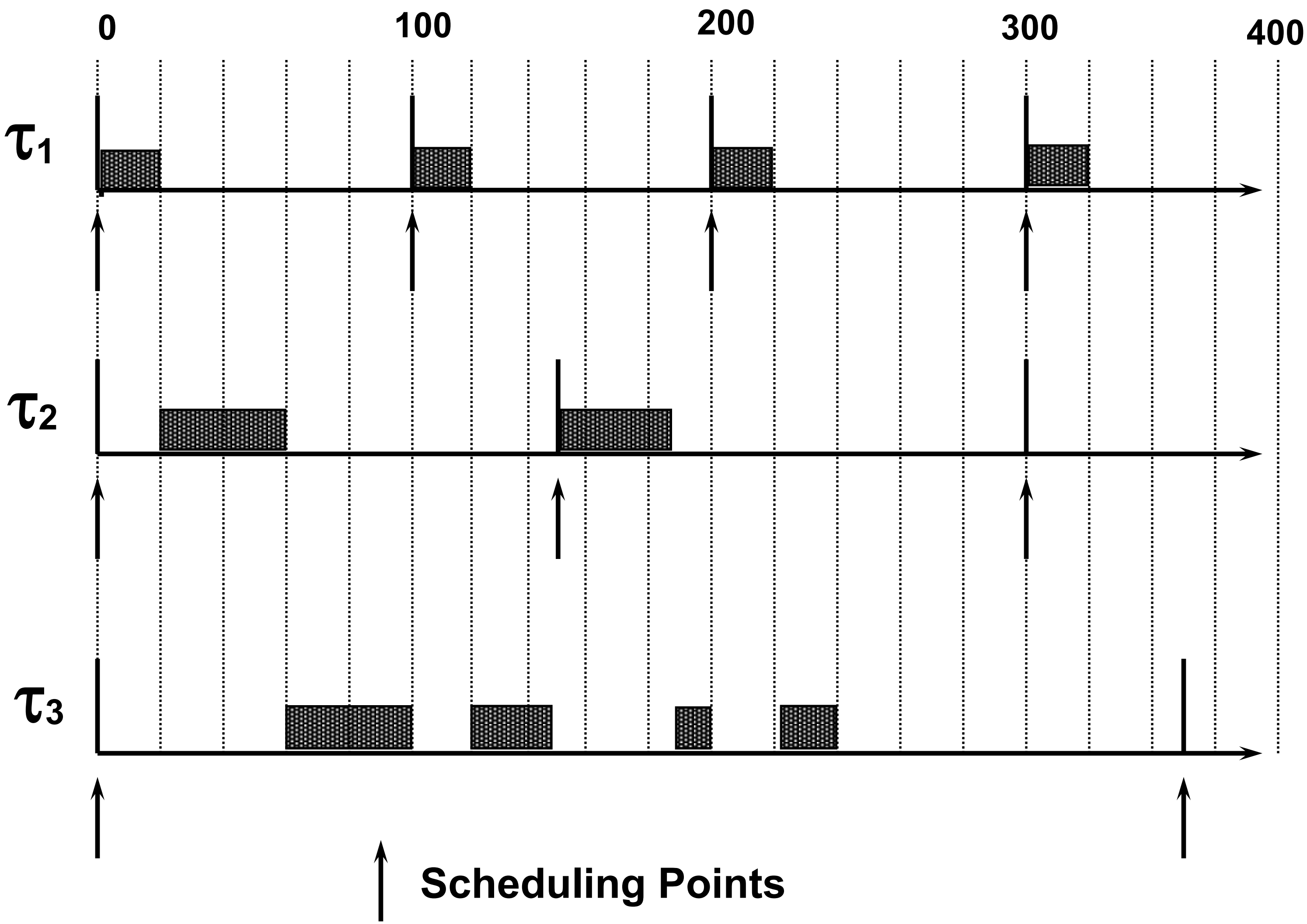
For ***harmonic*** task sets, the utilization bound is U(n)=1.00 for all n.

Sample Problem: Applying UB Test

	C	T	U
Task τ_1	20	100	**0.200**
Task τ_2	40	150	**0.267**
Task τ_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

Timeline for Sample Problem



Exercise: Applying the UB Test

Given:

Task	*C*	*T*	*U*
τ_1	1	4	
τ_2	2	6	
τ_3	1	10	

a. What is the total utilization?

b. Is the task set schedulable?

c. Draw the timeline.

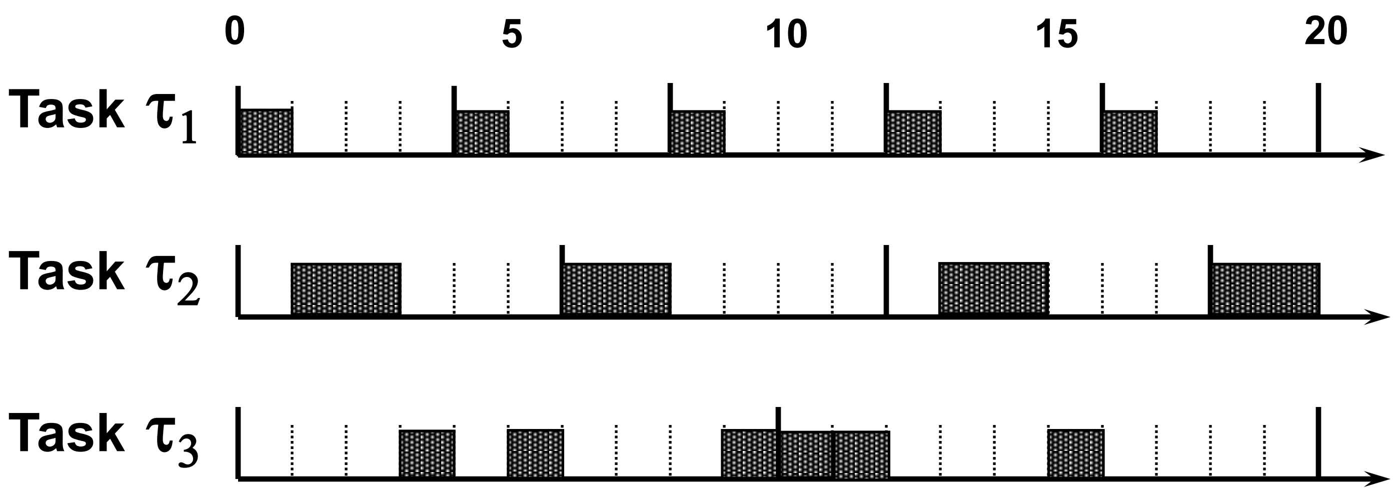
d. What is the total utilization if $C_3 = 2$?

Solution: Applying the UB Test

a. *What is the total utilization?* .25 + .34 + .10 = .69

b. *Is the task set schedulable?* Yes: .69 < U(3) = .779

c. *Draw the timeline.*



d. *What is the total utilization if $C_3 = 2$?*

.25 + .34 + .20 = .79 > U(3) = .779

Lecture 16

Toward a More Precise Test

- UB test has three possible outcomes:

$0 < U < U(n)$ → Success

$U(n) < U < 1.00$ → Inconclusive

$1.00 < U$ → Overload

- UB test is conservative.
- A more precise test can be applied.

Schedulability: RT Test

- Theorem: The worst-case phasing of a task occurs when it arrives simultaneously with all its higher priority tasks.
- Theorem: for a set of independent, periodic tasks, if each task meets its first deadline, with worst-case task phasing, the deadline will always be met.
- **Response time** (RT) or **Completion Time** test: let $\mathbf{a_n}$ **= response time of task** $\boldsymbol{i}$. a_n of task I may be computed by the following iterative formula:

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

- Test terminates when $a_{n+1} = a_n$.
- Task i is schedulable if its response time is before its deadline: $a_n < T_i$
- The above must be repeated for every task i from scratch

- This test must be **repeated for every task τ_i if required**
  - i.e. the value of *i* will change depending upon the task you are looking at
- Stop test once current iteration yields a value of a_{n+1} beyond the deadline (else, you may never terminate).
- The 'square bracketish' thingies represent the 'ceiling' function, NOT brackets

Example: Applying RT Test -1

• Taking the sample problem, we increase the compute time of τ_1 from 20 to 40; is the task set still schedulable?

	C	T	U
Task τ_1:	~~20~~ 40	100	~~0.200~~ 0.4
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

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

– first two tasks are schedulable by UB test

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

– UB test is inconclusive

– need to apply RT test

Example: Applying RT Test -2

•Use RT test to determine if τ_3 meets its first deadline: $\boxed{i = 3}$

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_0}{T_j} \right\rceil C_j = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260$$

Example: Applying the RT Test -3

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

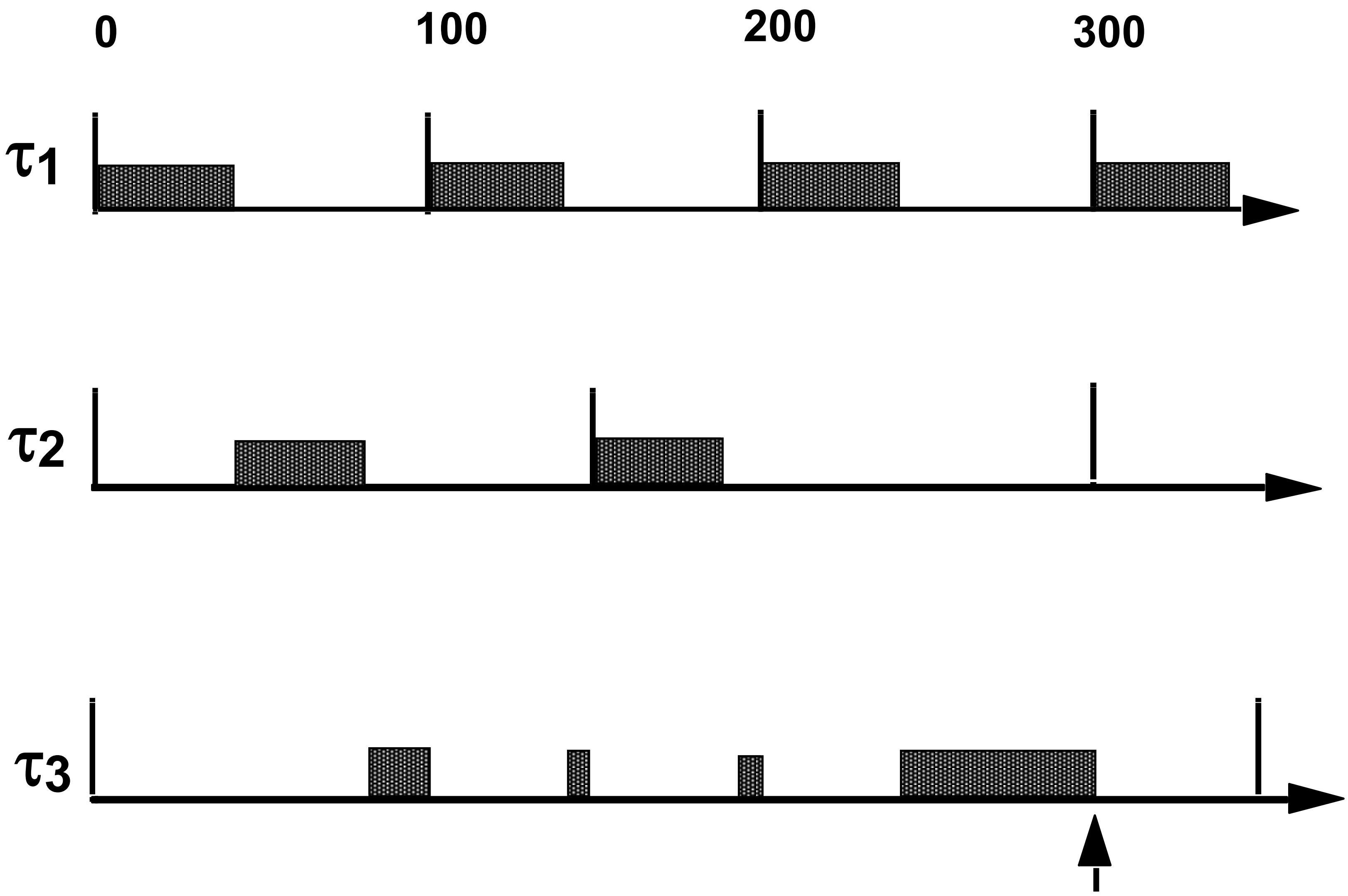
$$a_3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_2}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$a_3 = a_2 = 300$ **Done!**

- Task τ_3 is schedulable using RT test

$$a_3 = 300 < T = 350$$

Timeline for Example



τ_3 completes its work at t = 300

Exercise: Applying RT Test

Task τ_1: $C_1 = 1$ $T_1 = 4$

Task τ_2: $C_2 = 2$ $T_2 = 6$

Task τ_3: $C_3 = 2$ $T_3 = 10$

a) Apply the UB test

b) Draw timeline

c) Apply RT test

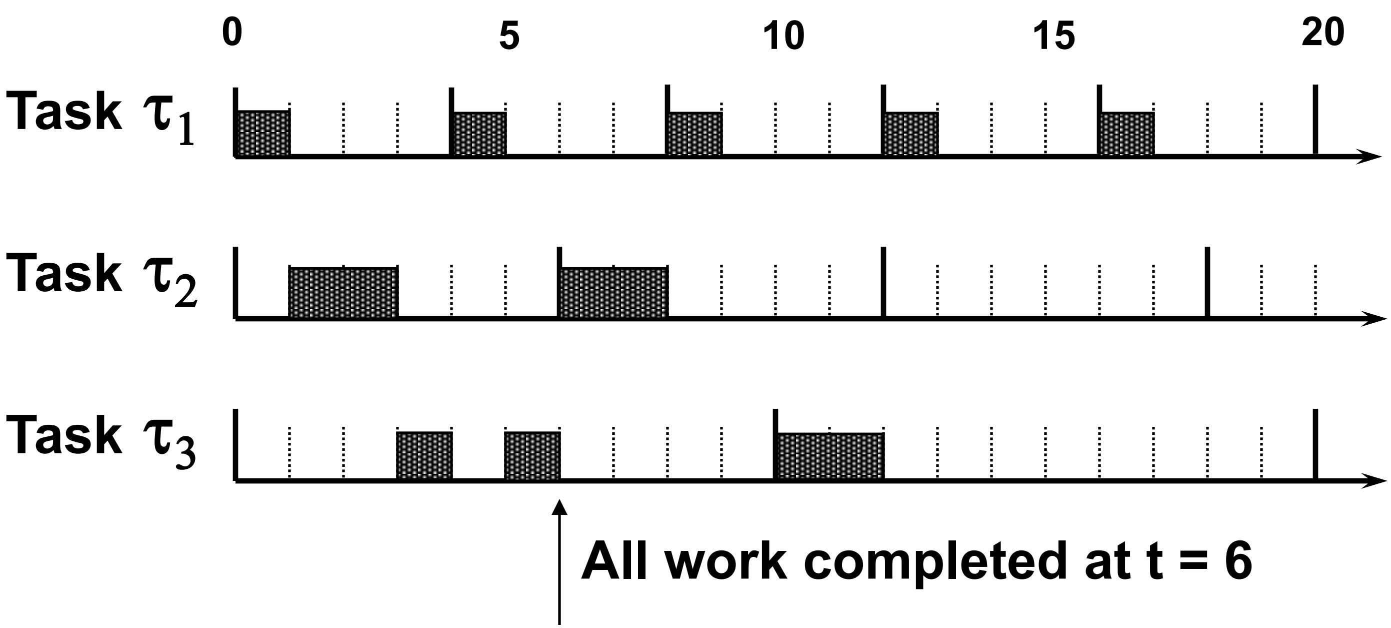
Solution: Applying RT Test

a) UB test

τ_1 and τ_2 OK -- no change from previous exercise

.25 + .34 + .20 = .79 > .779 ==> Test inconclusive for τ_3

b) RT test and timeline



Solution: Applying RT Test (cont.)

c) RT test

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 1 + 2 + 2 = 5$$

$$a_1 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0}{T_j} \right\rceil C_j = 2 + \left\lceil \frac{5}{4} \right\rceil 1 + \left\lceil \frac{5}{6} \right\rceil 2 = 2 + 2 + 2 = 6$$

$$a_2 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_1}{T_j} \right\rceil C_j = 2 + \left\lceil \frac{6}{4} \right\rceil 1 + \left\lceil \frac{6}{6} \right\rceil 2 = 2 + 2 + 2 = 6$$

Done

Read more: A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems

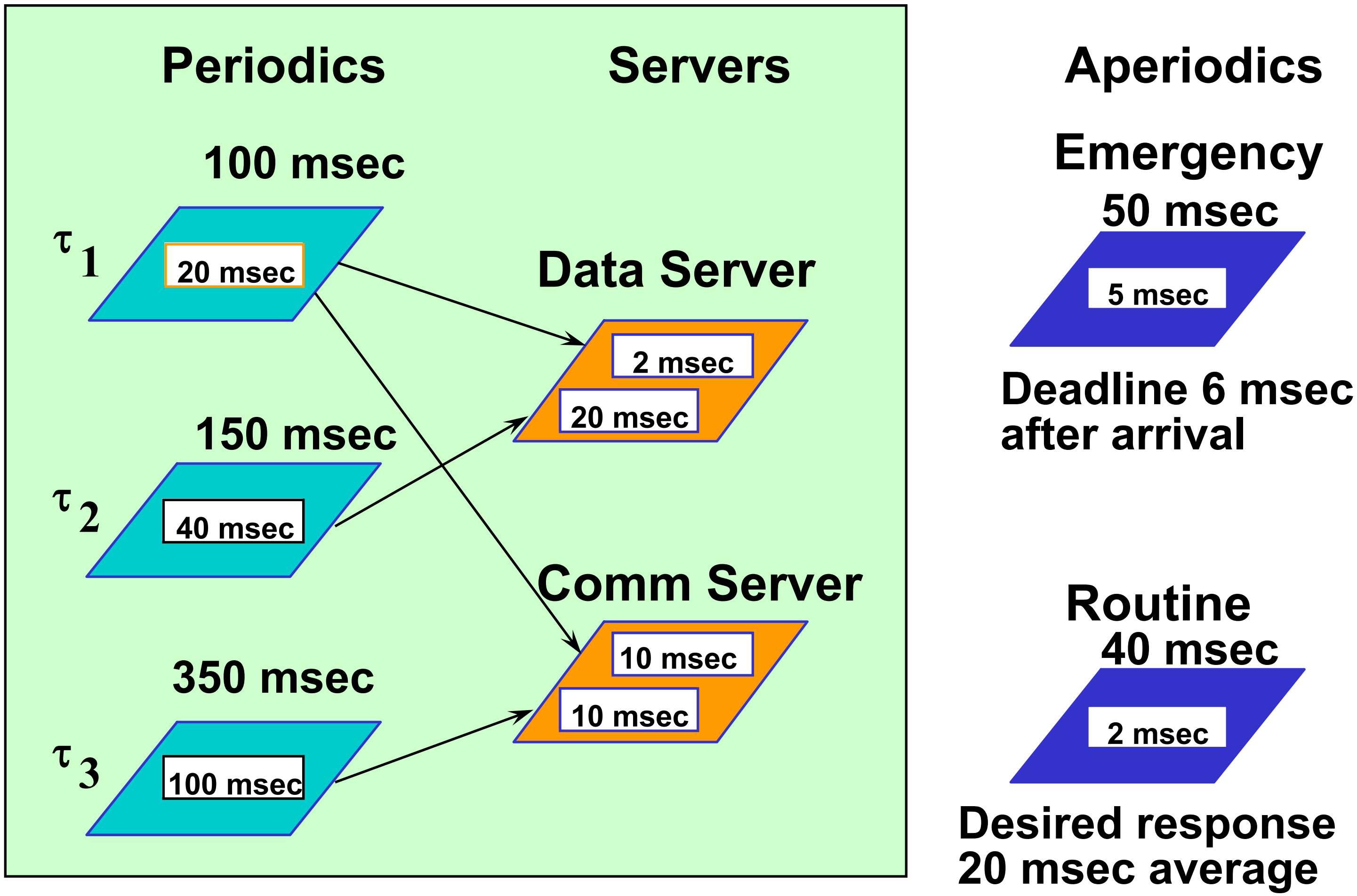
Summary

- Real-time goals are
  - fast response, guaranteed deadlines, and stability in overload
  - any scheduling may be used, but all real-time systems should be analyzed for timing
- Rate monotonic analysis
  - based on rate monotonic theory - analytic formulas to determine schedulability
  - framework for reasoning about system timing behavior
  - Provides an engineering basis for designing real-time systems
- RMS basic concepts
  - UB test is simple but conservative
  - RT test is more exact but also more complicated.
- To this point, UB and RT tests share the same limitations:
  - all tasks run on a single processor
  - there is zero context switch overhead
  - deadlines are always at the end of the period
  - there are no interrupts
  - rate-monotonic priorities are assigned
  - all tasks are periodic and noninteracting
  - tasks do not suspend themselves

Plan for Lectures

- Introduction to Real-Time Systems
  - Examples, Terminology, Metrics, Scheduling Policies
- Rate-Monotonic Analysis (RMA)
  - Fundamental concepts
  - An Introduction to Rate-Monotonic Analysis: independent tasks
- Present basic theory for periodic task sets
- **Extend basic theory to include**
  - **Context switch overhead, Interrupts**
  - **Preperiod deadlines**
- Consider task interactions
  - Priority inversion
  - Synchronization protocols (time allowing)
- Extend theory to aperiodic tasks
  - Sporadic servers (time allowing)

A Sample Problem



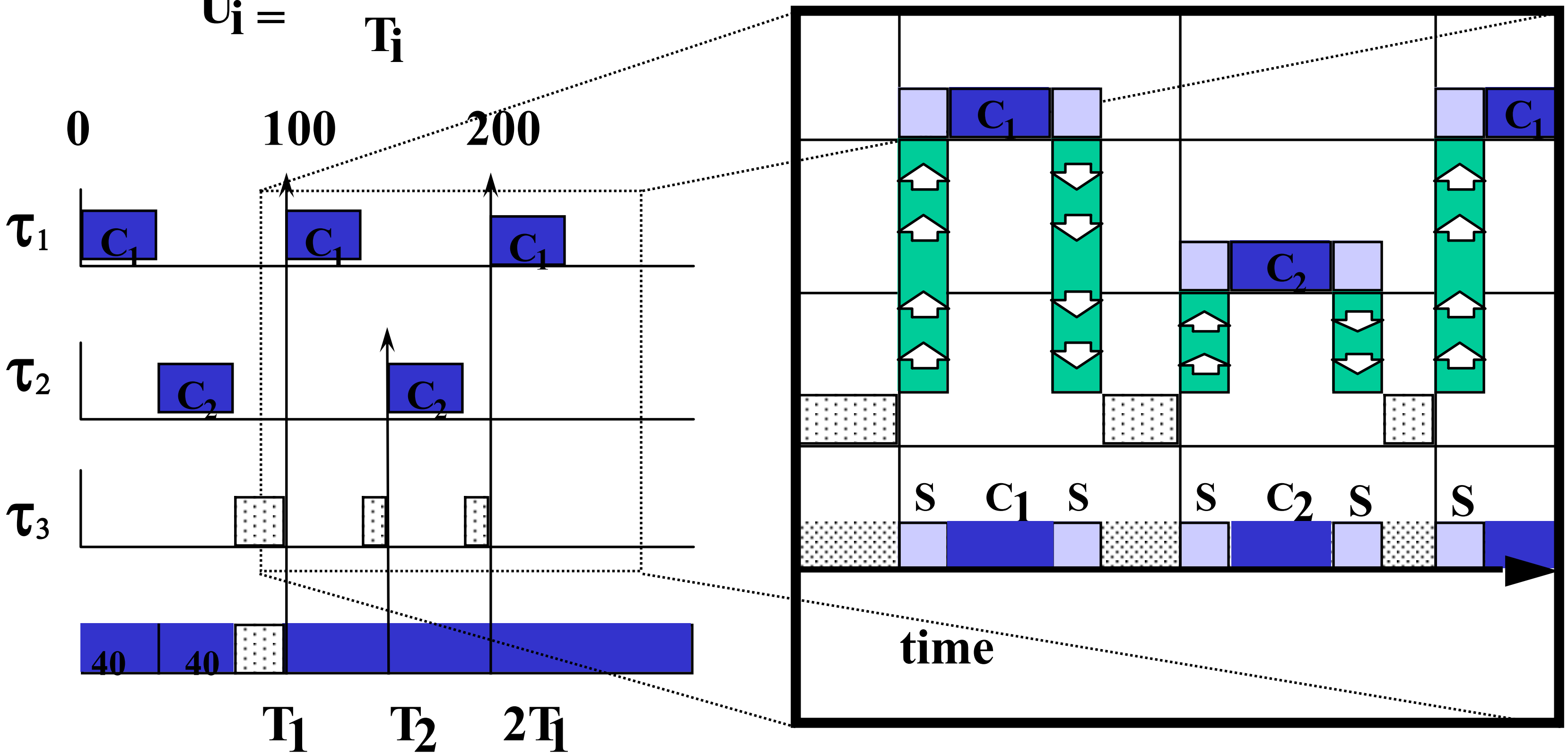
τ_2's deadline is 20 msec before the end of each period

Extensions to Basic Theory

- This section extends the schedulability tests to address
  - nonzero task switching times
  - preperiod deadlines
  - interrupts and non-rate-monotonic priorities

Modeling Task Switching as Execution Time

$$U_i = \frac{C_i + 2S}{T_i}$$



Two scheduling actions per task
(start of period and end of period)

Modeling Preperiod Deadlines

- Suppose task τ, with compute time *C* and period *T*, has a preperiod deadline *D* (i.e. $D < T$).
- Compare total utilization to modified bound:

$$U_{total} = \frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n, \Delta_i)$$

where Δ_i is the ratio (D_i / T_i).

$$U(n, \Delta_i) = \begin{cases} n\left((2\Delta_i)^{1/n} - 1\right) + 1 - \Delta_i, & \frac{1}{2} < \Delta_i \leq 1.0 \\ \Delta_i, & \Delta_i \leq \frac{1}{2} \end{cases}$$

Schedulability with Interrupts

- Interrupt processing can be inconsistent with rate-monotonic priority assignment.
  - interrupt handler executes with high priority despite its period
  - interrupt processing may delay execution of tasks with shorter periods
- Effects of interrupt processing must be taken into account in schedulability model.

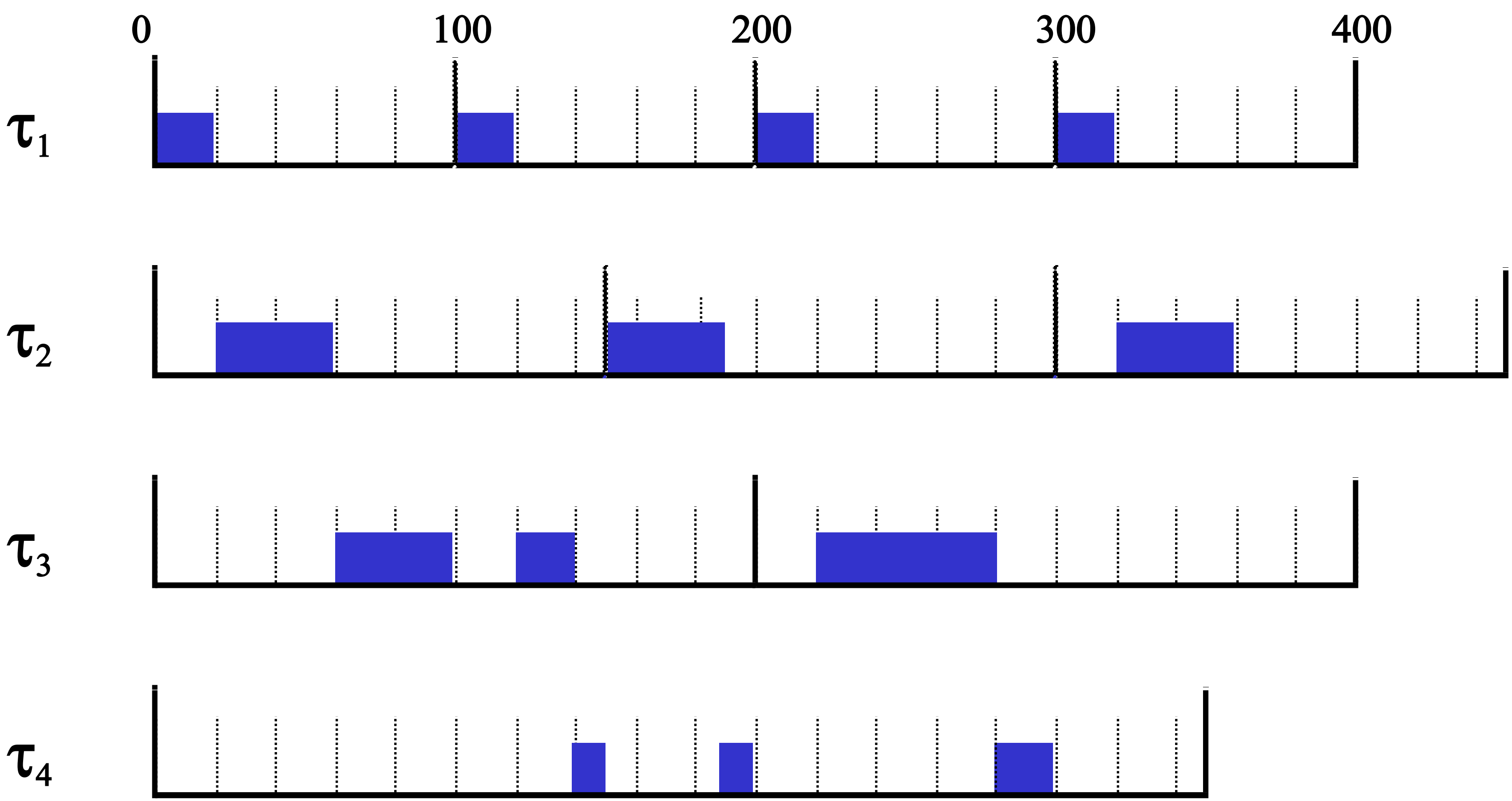
- Question is: how to do that?

Example: Determining Schedulability with Interrupts

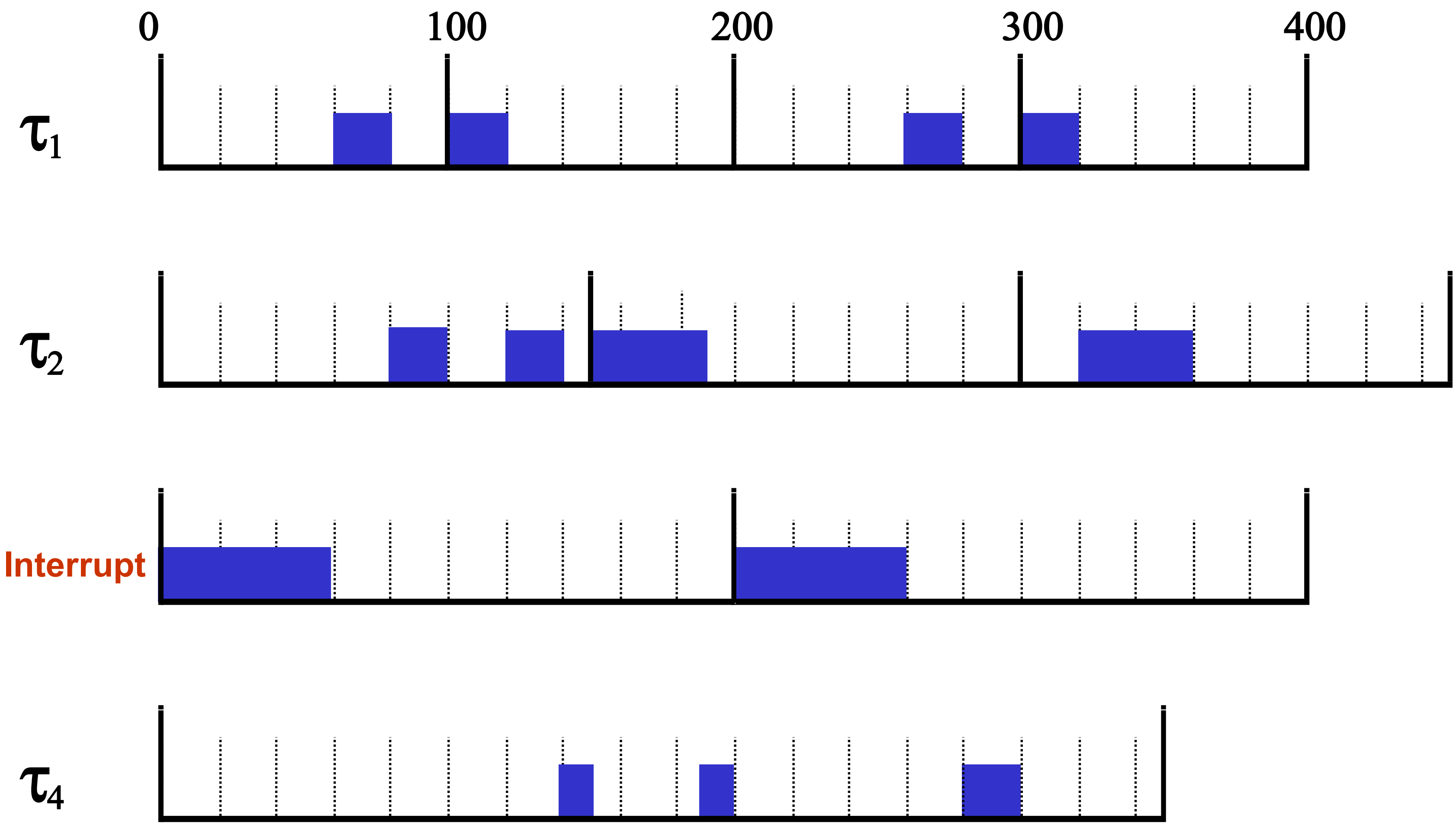
	C	T	U
Task τ_1:	20	100	0.200
Task τ_2:	40	150	0.267
Task τ_3:	60	200	0.300
Task τ_4:	40	350	0.115

τ_3 is an interrupt handler

Example: Execution with Rate-Monotonic Priorities



Example: Execution with an Interrupt Priority



Resulting Table for Example

Task (i)	**Period (T)**	**Execution Time (C)**	**Priority (P)**	**Deadline (D)**
τ_3	200	60	Hardware (*highest*)	200
τ_1	100	20	*High*	100
τ_2	150	40	*Medium*	150
τ_4	350	40	*Low*	350

UB Test with Interrupt Priority

- Test is applied to each task.
- Determine effective utilization (f_i) of each task τ_i using

$$f_i = \boxed{\sum_{j \in H_n} \frac{C_j}{T_j}} + \boxed{\frac{C_i}{T_i}} + \boxed{\frac{1}{T_i} \sum_{k \in H_1} C_k}$$

Preemption from tasks that can "hit" more than once (with period less than D_i)

Execution of task under test

Preemption from tasks that can hit only once (with period greater than D_i)

Compare effective utilization against bound U(n).

- **$n = \mathrm{num}(H_n) + 1$**
- **$\mathrm{num}(H_n)$ = the number of tasks in the set H_n**

UB Test with Interrupt Priority: t3

- For τ_3, no tasks have a higher priority:
  - $H = H_n = H_1 = \{ \}$

$$f_3 = \sum 0 + \frac{C_3}{T_3} + \sum 0$$

Note:

num(H_n) = 0; therefore, utilization bound is U(1).

Plugging in the numbers:

$$f_3 = \frac{C_3}{T_3} = \frac{60}{200} = 0.3 < 1.0$$

UB Test with Interrupt Priority: τ_1

To τ_1, τ_3 has higher priority: $H = \{\tau_3\}$; $H_n = \{\ \}$; $H_1 = \{\tau_3\}$

$$f_1 = \sum 0 + \frac{C_1}{T_1} + \frac{1}{T_1}\sum_{k=3} C_k$$

Note:

num(H_n) = 0; therefore, utilization bound is U(1).
Plugging in the numbers:

$$f_1 = \frac{C_1}{T_1} + \frac{C_3}{T_1} = \frac{20}{100} + \frac{60}{100} = 0.800 < 1.0$$

UB Test with Interrupt Priority: τ_2

To τ_2: H = $\{\tau_1, \tau_3\}$; H_n = $\{\tau_1\}$; H_1 = $\{\tau_3\}$.

$$f_2 = \sum_{j=1} \frac{C_j}{T_j} + \frac{C_2}{T_2} + \frac{1}{T_2} \sum_{k=3} C_k$$

Note:

num(H_n) = 1; therefore, utilization bound is U(2).
Plugging in the numbers:

$$f_2 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_2} = \frac{20}{100} + \frac{40}{150} + \frac{60}{150} = 0.867 > 0.828$$

UB Test with Interrupt Priority: τ_4

To τ_4: H = {τ_1, τ_2, τ_3}; H_n = {τ_1, τ_2, τ_3}; H_1 = { }.

$$f_4 = \sum_{j=1,2,3} \frac{C_j}{T_j} + \frac{C_4}{T_4} + \sum 0$$

Note:

num(Hn) = 3; therefore, utilization bound is U(4).

Plugging in the numbers:

$$f_4 = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} + \frac{C_4}{T_4}$$

$$= \frac{20}{100} + \frac{40}{150} + \frac{60}{200} + \frac{40}{350} = 0.882 > 0.756$$

Exercise: Schedulability with Interrupts

- Use the UB test to determine which tasks are schedulable

- Given the following tasks:

Task (i)	Period (T)	Execution Time (C)	Deadline (D)	Priority (P)
τ_{int}	6	2	6	HW
τ_1	4	1	3	High
τ_2	10	1	10	Low

Solution: Schedulability with Interrupts

$$\frac{C_{int}}{T_{int}} \le U\,(1)$$

0.334 < 1.0

{H1}

$$\frac{C_1}{T_1} + \frac{C_{int}}{T_1} \le U\,(1, .75)$$

0.250 + 0.500 = 0.750 = U(1, .75)

{H_n}

$$\frac{C_{int}}{T_{int}} + \frac{C_1}{T_1} + \frac{C_2}{T_2} \le U\,(3)$$

0. 334 + 0. 250 + 0.100 = 0.684 < 0.779

Basic Theory: Where Are We?

- We have shown how to handle
  - task context switching time: include 2S in C
  - Pre-period deadlines: change bound to U(n, Di)
  - non-rate-monotonic priority assignments
- We still must address
  - task interactions
  - aperiodic tasks
- We still assume
  - single processor
  - priority-based scheduling
  - a task does not suspend *itself* voluntarily

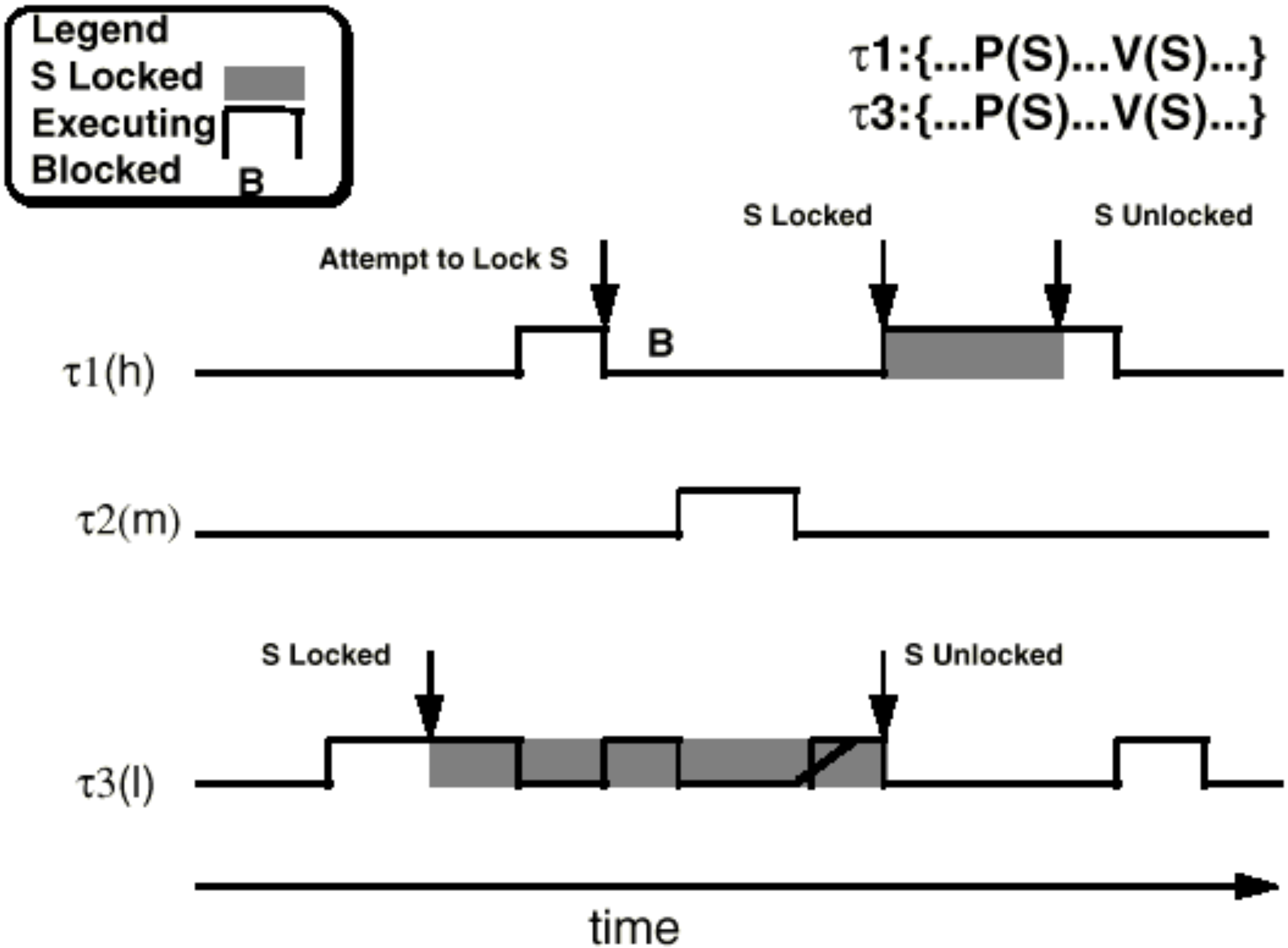
Plan for Lectures

- Introduction to Real-Time Systems
  - Examples, Terminology, Metrics, Scheduling Policies
- Rate-Monotonic Analysis (RMA)
  - Fundamental concepts
  - An Introduction to Rate-Monotonic Analysis: independent tasks
- Present basic theory for periodic task sets
- Extend basic theory to include
  - Context switch overhead, Interrupts
  - Preperiod deadlines
- **Consider task interactions**
  - **Priority inversion**
  - **Synchronization protocols (time allowing)**
- Extend theory to aperiodic tasks
  - Sporadic servers (time allowing)

Priority Inversion

- Ideally, under prioritized preemptive scheduling, higher priority tasks should *immediately* preempt lower priority tasks.
- When lower priority tasks cause higher priority tasks to wait (e.g. the locking of shared data), **priority inversion** is said to occur.
- It seems reasonable to expect that the duration of priority inversion (also called **blocking time**) should be a function of the duration of the critical sections.
- **Critical section**:
  - the duration of a task using a shared resource.

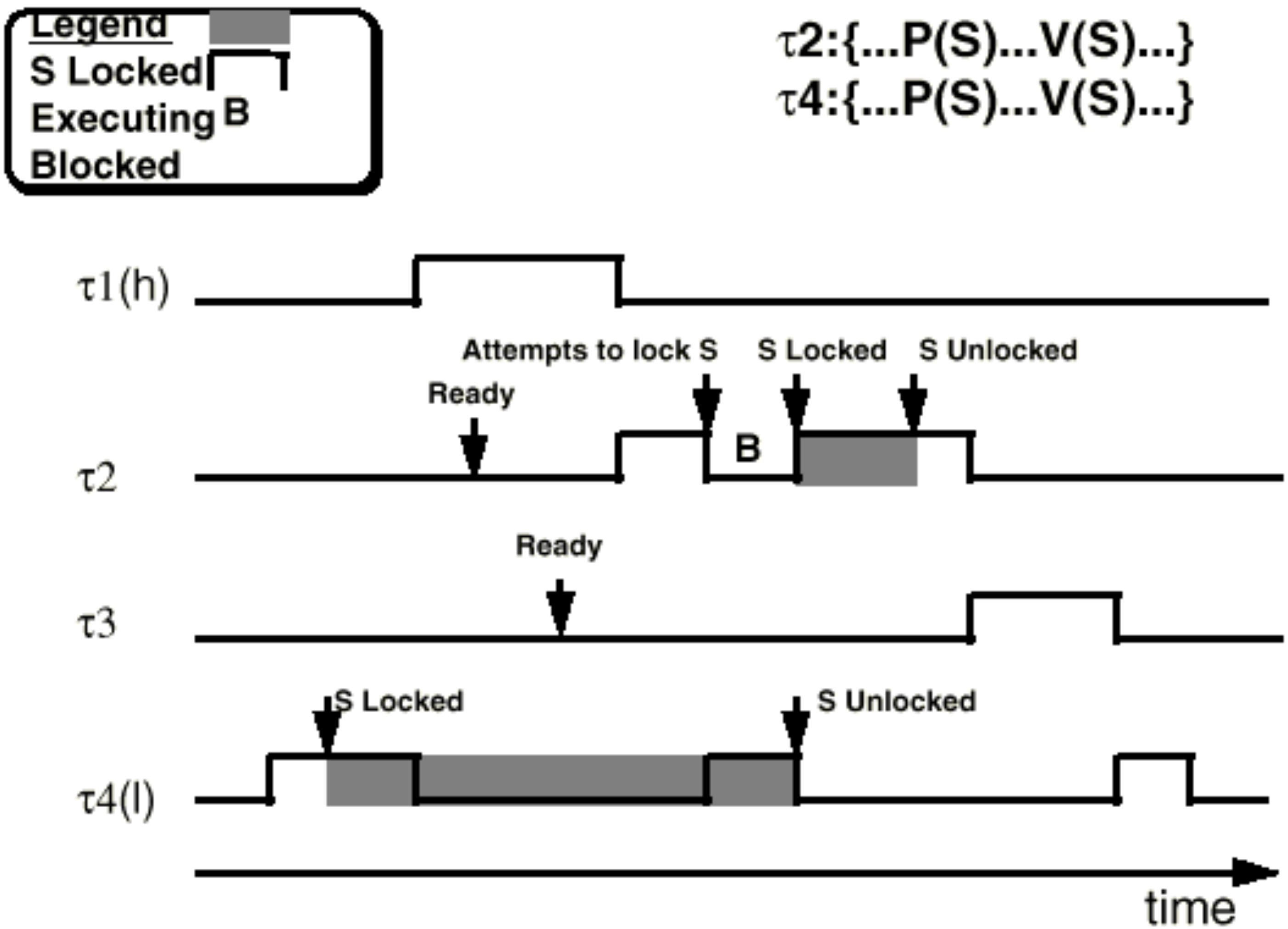
Unbounded Priority Inversion



Basic Priority Inheritance Protocol

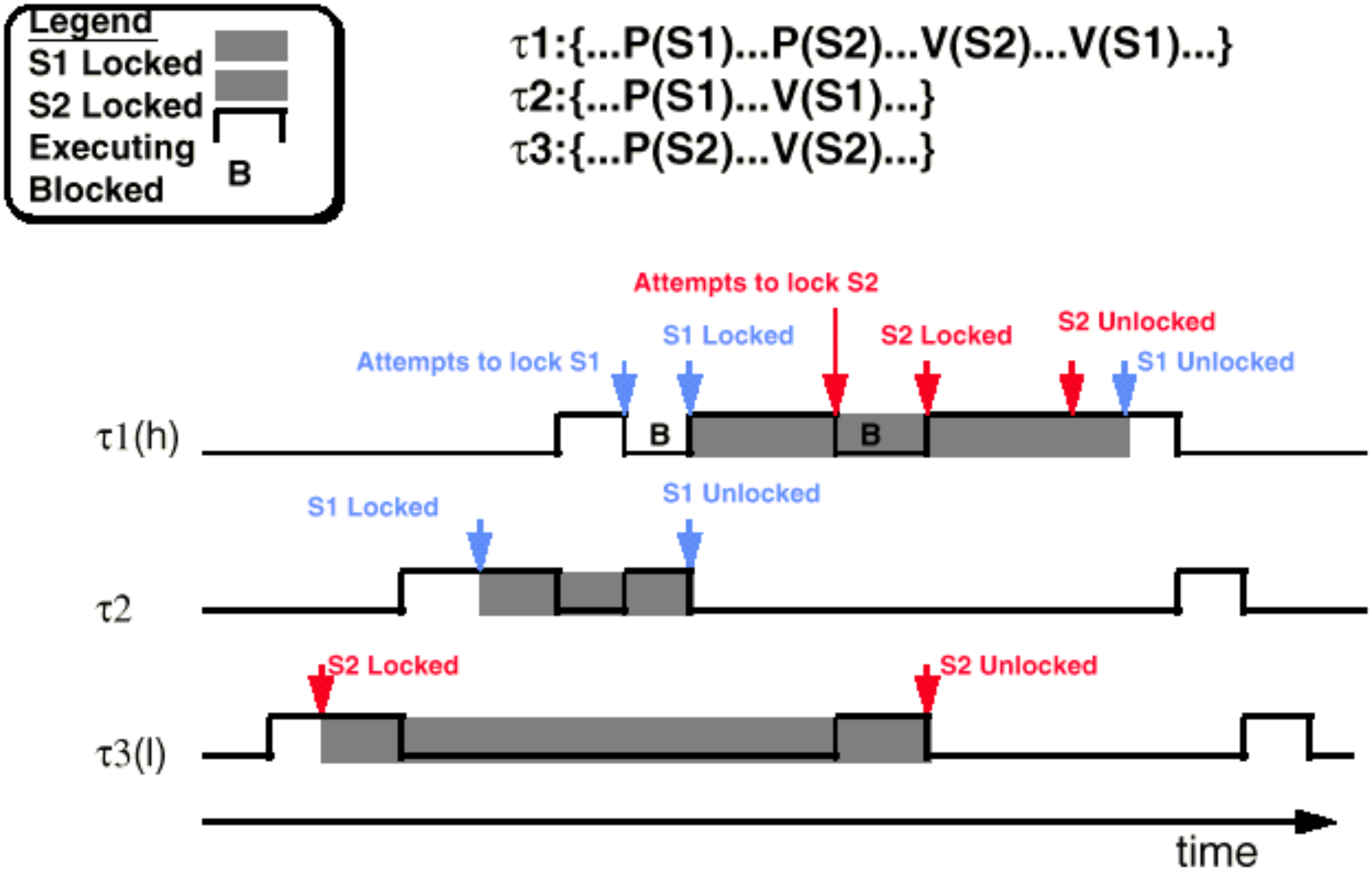
- Let the lower priority task τ_3 use the highest priority of the higher priority tasks it blocks. In this way, the medium priority tasks can no longer preempt low priority task τ_3, which has blocked the higher priority tasks.
- **Priority inheritance is transitive**.
  - If A blocks B and B blocks C, A should execute at the priority of max(B,C).

Basic Priority Inheritance Protocol

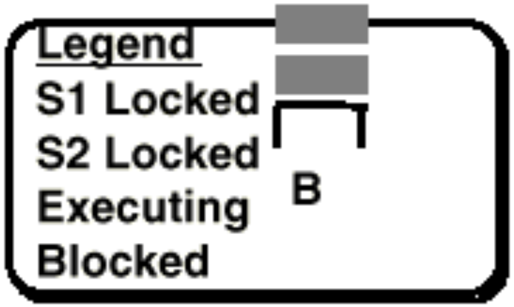


Chained Blocking

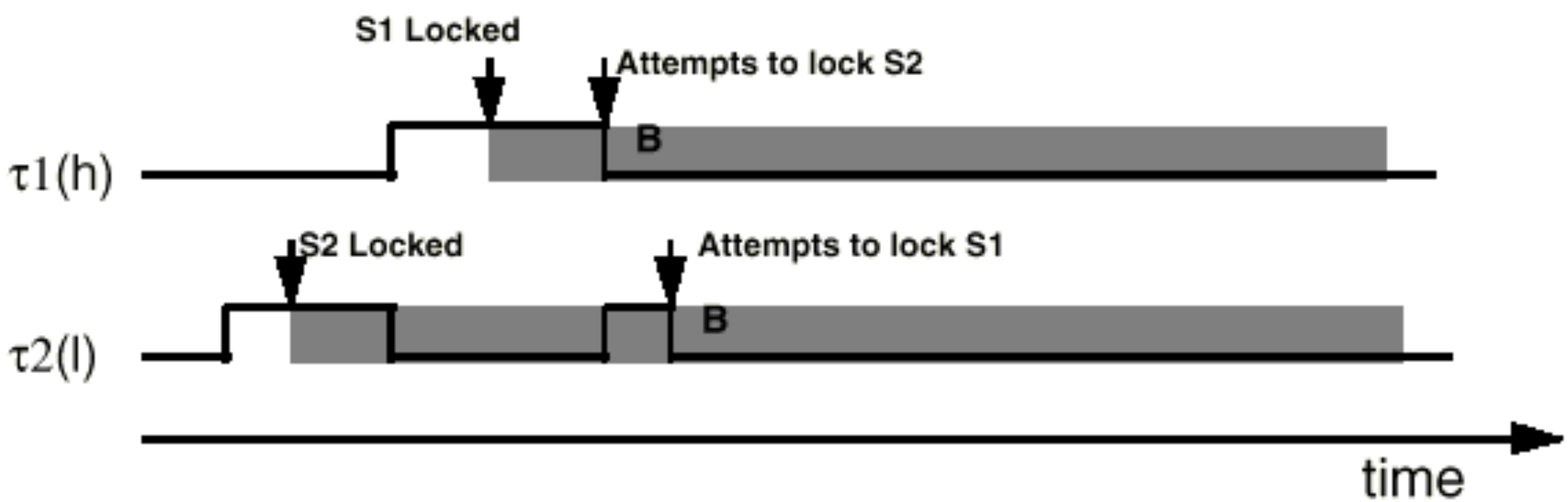
$\tau1$:{...P(S1)...P(S2)...V(S2)...V(S1)...}
$\tau2$:{...P(S1)...V(S1)...}
$\tau3$:{...P(S2)...V(S2)...}



Deadlock Under BIP



$\tau1$:{...P(S1)...P(S2)...V(S2)...V(S1)...}
$\tau2$:{...P(S2)...P(S1)...V(S1)...V(S2)...}



Properties of Basic Priority Inheritance

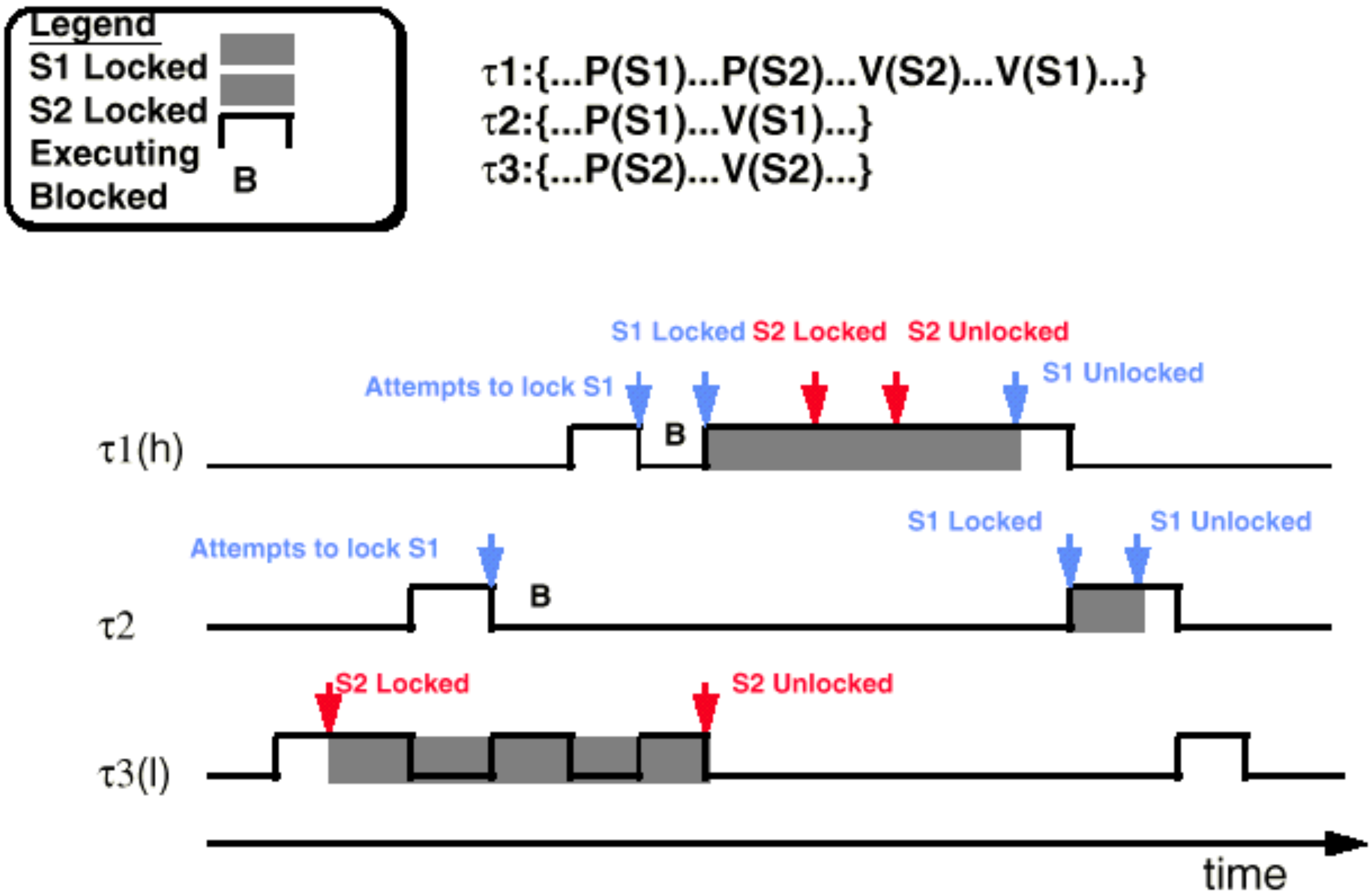
- There will be no deadlock if there is no nested locks, or application level deadlock avoidance scheme such the ordering of resource is used.
- Chained priority is fact of life. But a task is blocked at most by n lower priority tasks sharing resources with it, when there is no deadlock.
- The priority inheritance protocol is supported in POSIX real time extensions.
  - It is easy to implement
  - it is supported by not only most RT OS vendors but also OS/2, Windows 95, Windows CE, AIX, HP/UX and Solaris.

Priority Ceiling Protocol

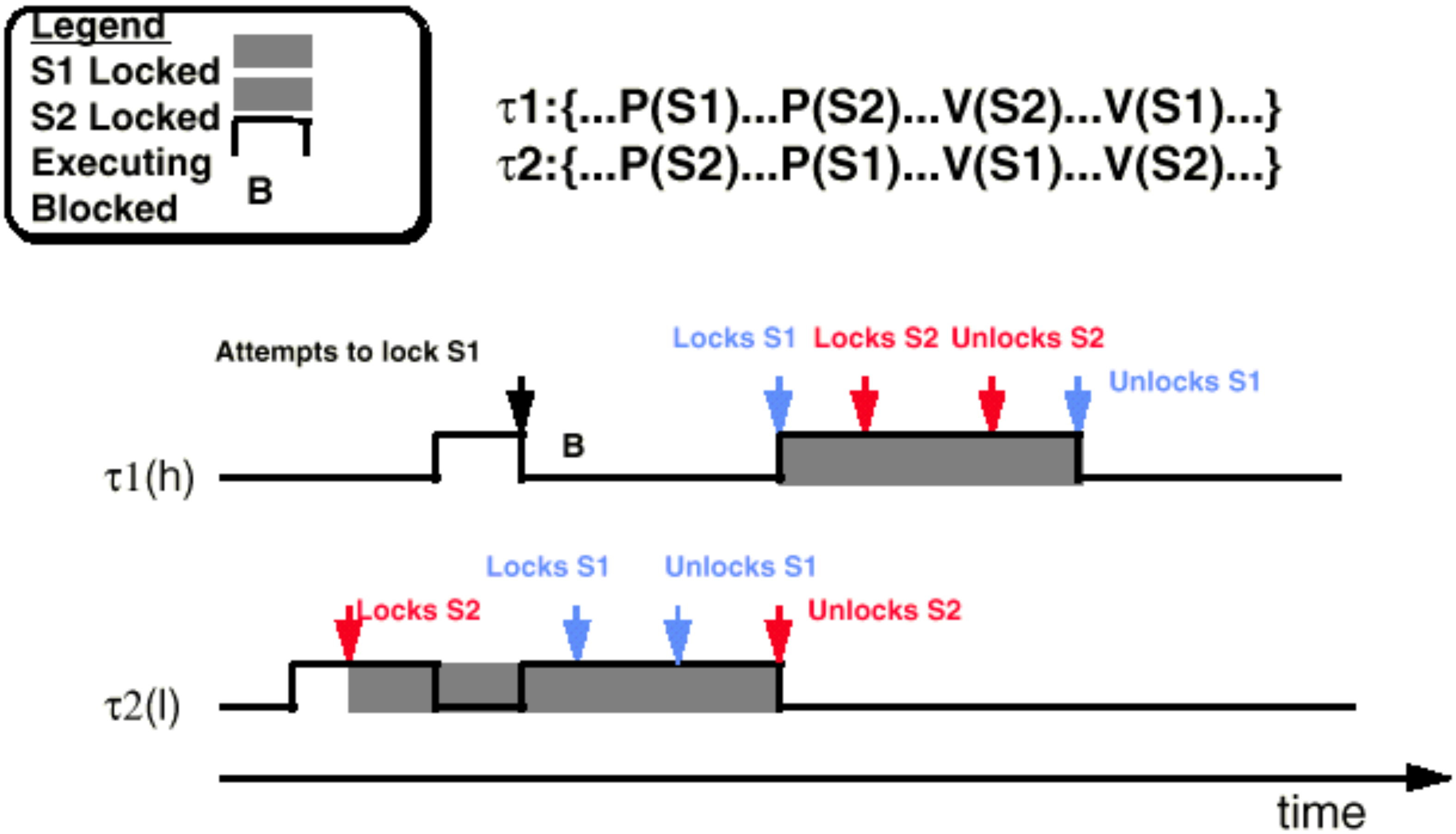
- A **priority ceiling** is statically assigned to each mutex, which is equal to the highest priority task that may use this mutex.
- A task can lock a mutex if and only if its priority is higher than the priority ceilings of all mutexes currently locked by other tasks.
- If a task is blocked by a lower priority task, the lower priority task inherits its priority.

Blocked by At Most One Critical Section (PCP)

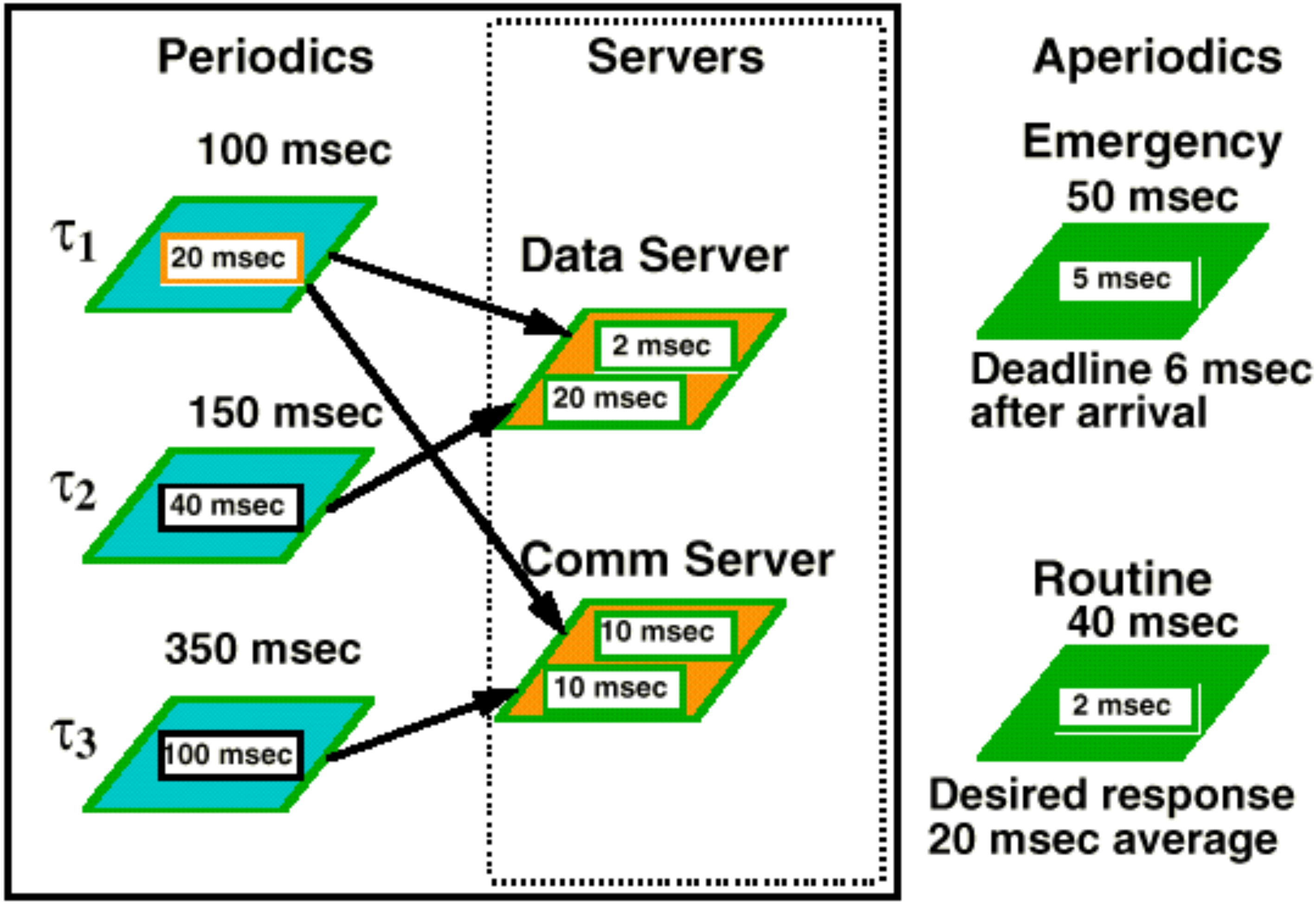
τ1:{...P(S1)...P(S2)...V(S2)...V(S1)...}
τ2:{...P(S1)...V(S1)...}
τ3:{...P(S2)...V(S2)...}



Deadlock Avoidance: Using PCP



A Sample Problem



τ_2's deadline is 20 msec before the end of periods

Sample Problem: Using BIP

	C	T	E	B
τ_1	20	100		(20+10)
τ_2	40	150	20	10
τ_3	100	350		

$$\overset{\text{preemption}}{\frac{C_1+\dots+C_{i-1}}{T_i}} + \overset{\text{execution}}{\frac{C_i}{T_i}} + \overset{\text{early deadline}}{\frac{\mathbf{E}_i}{T_i}} + \overset{\text{blocking}}{\frac{B_i}{T_i}} \le U\ (i)$$

τ_2's deadline is D = 20 msec before the end of period

Schedulability Model Using BIP

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

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

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

Plan for Lectures

- Introduction to Real-Time Systems
  - Examples, Terminology, Metrics, Scheduling Policies
- Rate-Monotonic Analysis (RMA)
  - Fundamental concepts
  - An Introduction to Rate-Monotonic Analysis: independent tasks
- Present basic theory for periodic task sets
- Extend basic theory to include
  - Context switch overhead, Interrupts
  - Preperiod deadlines
- Consider task interactions
  - Priority inversion
  - Synchronization protocols (time allowing)
- **Extend theory to aperiodic tasks**
  - **Sporadic servers (time allowing)**

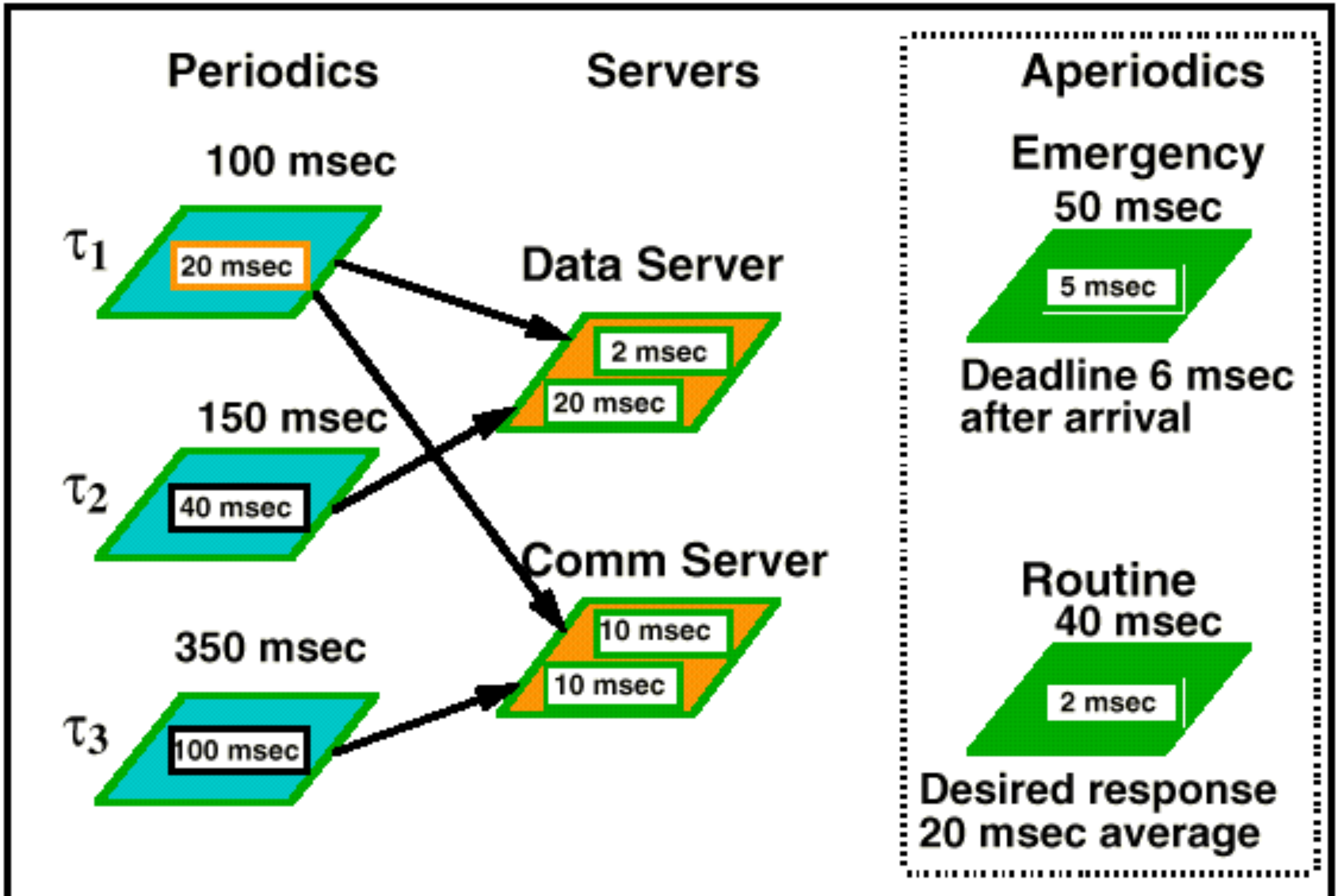
Concepts and Definitions

- Aperiodic task
  - runs at irregular intervals
- Aperiodic deadline
  - hard, minimum interarrival time
  - soft, best average response

Sporadic Server (SS)

- To provide ondemand service to aperiodic events, we can allocate a **budget periodically**. A periodic event can execute as long as there is budget left.
- Modeled as periodic tasks
  - Fixed execution budget (**C**)
  - Replenishment interval (**T**)
- Priority is based on T, just like periodic tasks.
- Replenishment occurs one “period” after **start** of use.

A Sample Problem



τ_2's deadline is 20 msec before the end of periods

Sample Problems: Aperiodic

- Emergency Server (ES)
  - Execution Budget, C = 5
  - Replenish Interval, T= 50
- General Aperiodic Server (GS) Design guideline:
  - Give it as high a priority as possible and as much "tickets" as possible, without causing regular periodic tasks to miss deadlines:
    - Execution Budget, C = 10
    - Replenish Interval, T = 100
- Simulation and queuing theory using M/M/1 approximation indicate that the average response time is ~2 msec.

Summary of Lecture

- Synchronization in real-time systems
  - Priority inversion
  - Unbounded priority inversion
  - Protocols to bound priority inversion
    - basic priority inheritance protocol
    - priority ceiling protocol
- Dealing with Aperiodic tasks
  - sporadic servers
- Solving our example problem completely
  - early deadlines
  - average response time