

Introduction to Embedded & Real-Time Systems

Real-Time Scheduling 1

Sathish Gopalakrishnan

Electrical and Computer Engineering
The University of British Columbia

Metrics in Real-Time Systems – I

- End-to-end latency:
 - E.g. worst-case, average-case, variance, distribution
 - Can involve multiple hops (across nodes, links, switches and routers)
 - Behavior in the presence or absence of failures
- Jitter
 - Variability in metrics (e.g., variability in throughput)
- Throughput
 - How many requests can be processed in unit time?
 - How many messages can be transmitted in unit time?
- Robustness
 - How many faults can be tolerated before system failures?
 - What functionality gets compromised?



Metrics in Real-Time Systems – II

■ Safety & Certification

- Is the system “safe”?
- Can the system get into an ‘unsafe’ state? Has it been ‘certified’?

■ Modes and reconfiguration

- What happens when the system mission changes?
- What happens under mode changes?
 - What are examples of mode changes?
- What happens when individual elements fail?
- Can the system reconfigure itself dynamically?
- How does the system behave after re-configuration?

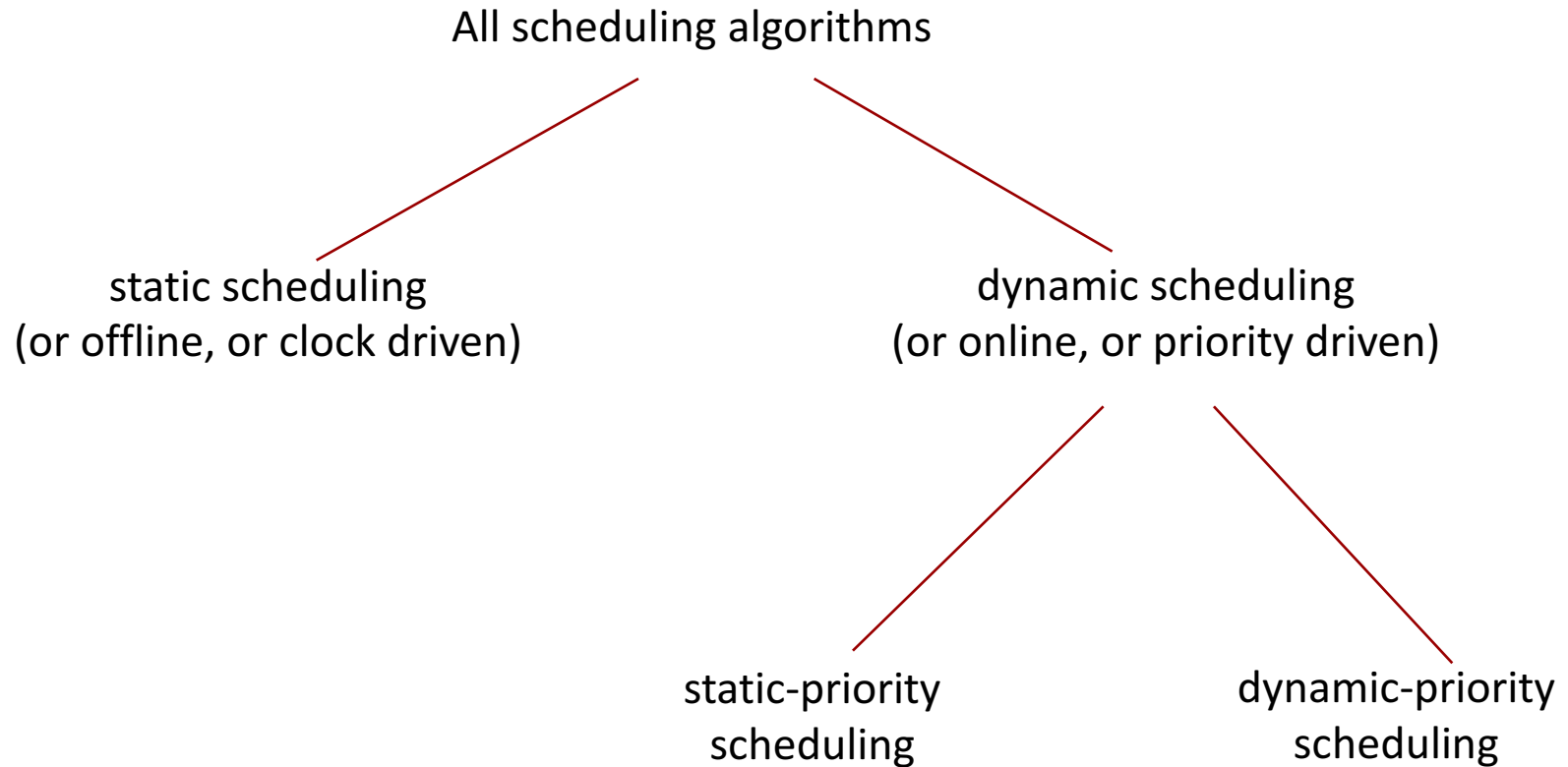
■ Security

- Can the system’s integrity be compromised?
- Can violations be detected?
- Renewed interest in this area with appliances being connected to the Internet

Scheduling

- Scheduler = resource allocator that affects the timing of real-time services
- Offline scheduler = does allocation at design time
- Online scheduler = does allocation at run-time
- Schedulers need to know dependencies and worst-case behavior
- Worst-case execution time (WCET) is often required to be known in real-time systems
- How do you determine worst-case behavior?

Classification of Scheduling Algorithms



Scheduling Algorithms (No Priority)

- Non-priority-based
 - All tasks are created equal
 - There is no way to indicate which tasks are more “important” (more critical) than others
- First-In-First-Out (FIFO) or First-Come-First-Served (FCFS)
 - Ready tasks are inserted into a list
 - Tasks are dispatched from the list in their **order of entry** in the list
 - No preemption, i.e., a task runs to completion before the next task runs
 - No consideration of task priorities
- Round-Robin Preemptive
 - Ready tasks are dispatched **in turn**
 - Each task given its fair share of fixed execution time
 - Preemption of running task at the end of the fixed interval
 - No consideration of task priorities

Scheduling Algorithms (Fixed-Priority)

- Fixed-priority based
 - All tasks are not created equal
 - Some tasks have more importance (higher priority) than others
 - Once a task's priority is assigned, it cannot change during run-time
- **Rate Monotonic Analysis (RMA)**
 - Shorter the period, the higher the priority of the task
 - Assigns fixed priorities in reverse order of period length
 - Tasks requiring frequent attention have higher priority and get scheduled earlier
- Least Compute Time (LCT)
 - Shorter the computation time, the higher the priority of the task
 - Assigns fixed priorities in reverse order of computation length
 - Tasks finishing quickly have higher priority and get scheduled earlier

Scheduling Algorithms (Dynamic-Priority)

- Dynamic-priority based
 - All tasks are not created equal
 - Some tasks have more importance (higher priority) than others
 - Task's priority (and thus, the schedule) may change during execution
- Shortest Completion Time (SCT)
 - Ready task with the **smallest remaining compute time** gets scheduled first
- Earliest Deadline First (EDF)
 - Ready task with the **earliest future deadline** gets scheduled first
- Least Slack Time (LST)
 - Ready task with the **smallest amount of free/slack time** within the cycle gets scheduled first

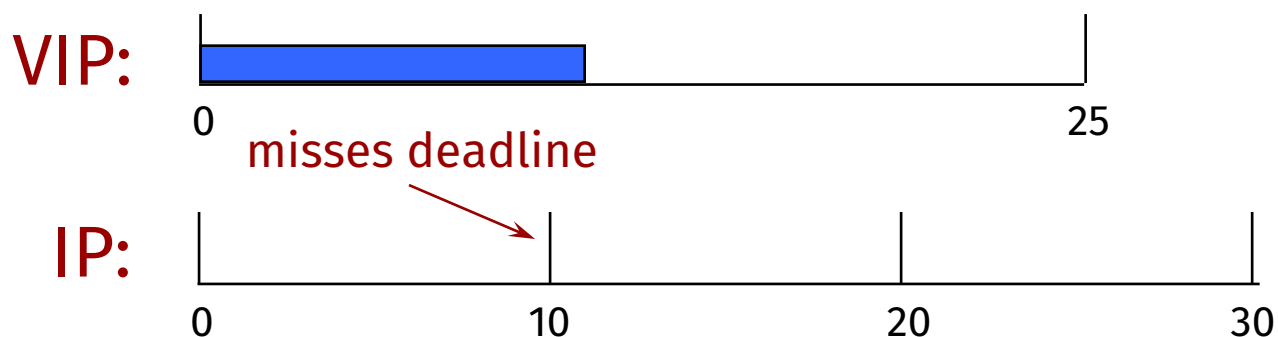
Example of Priority Assignment

Semantics-Based Priority Assignment

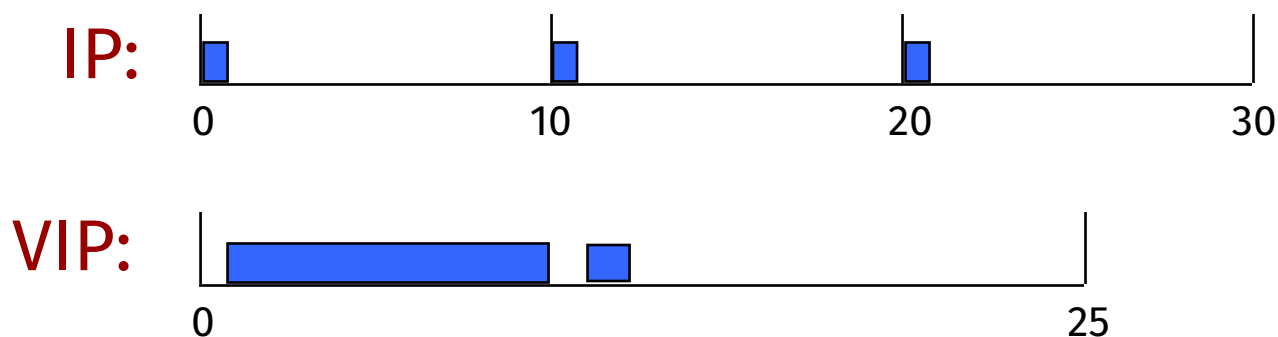
$$\tau_i = \{ C_i, T_i \}$$

$$\text{VIP} = \{ 10, 25 \}$$

$$\text{IP} = \{ 1, 10 \}$$



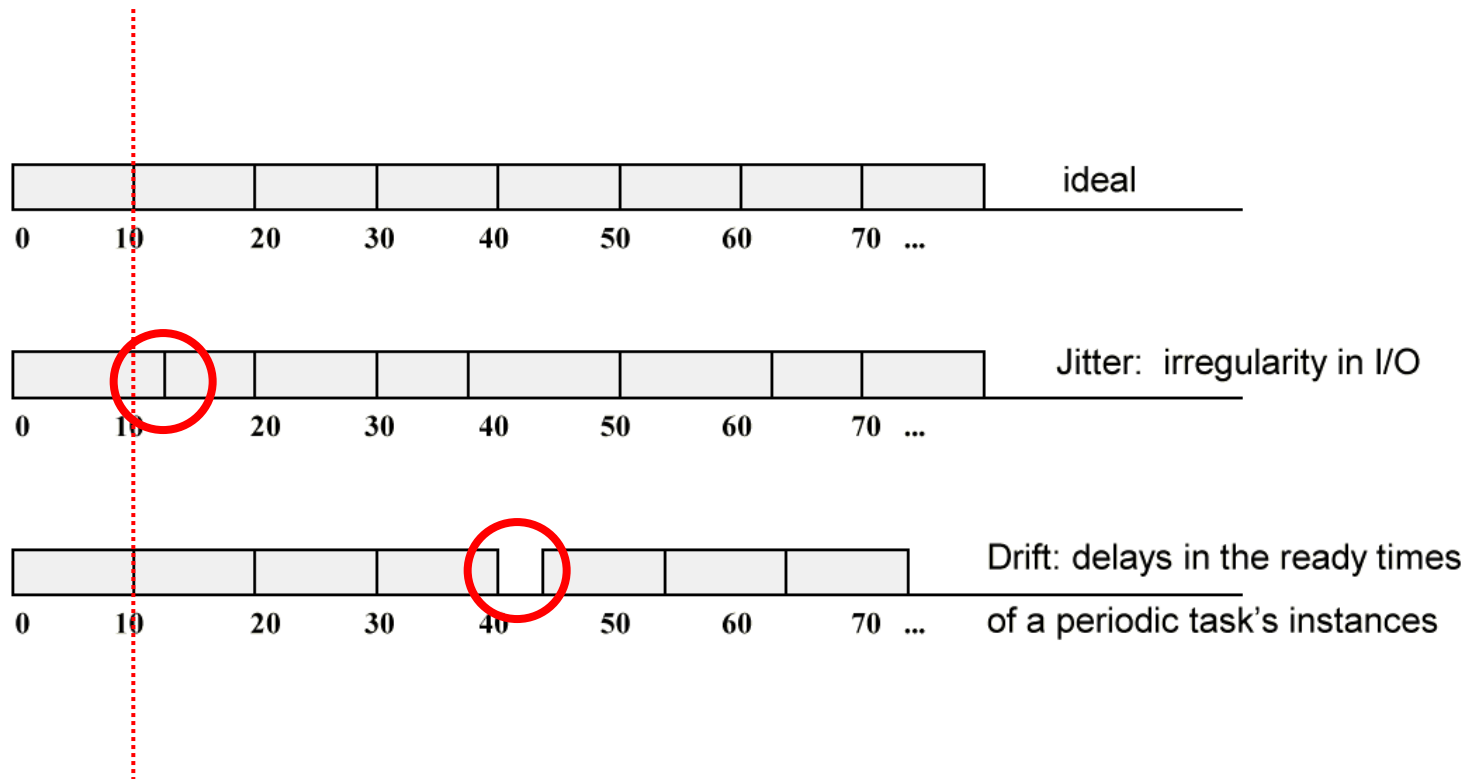
Policy-Based Priority Assignment



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, for schedulability

Drift and Jitter



Drift can be eliminated completely but one can only hope to minimize jitter in general

Sources of Drift and Jitter

- What can cause drift?
 - What can cause jitter?
 - How would you prevent drift?
 - How would you prevent jitter?
-
- If I gave you a piece of code and asked you to spot all of the places where jitter and drift could occur, how you would go about it?

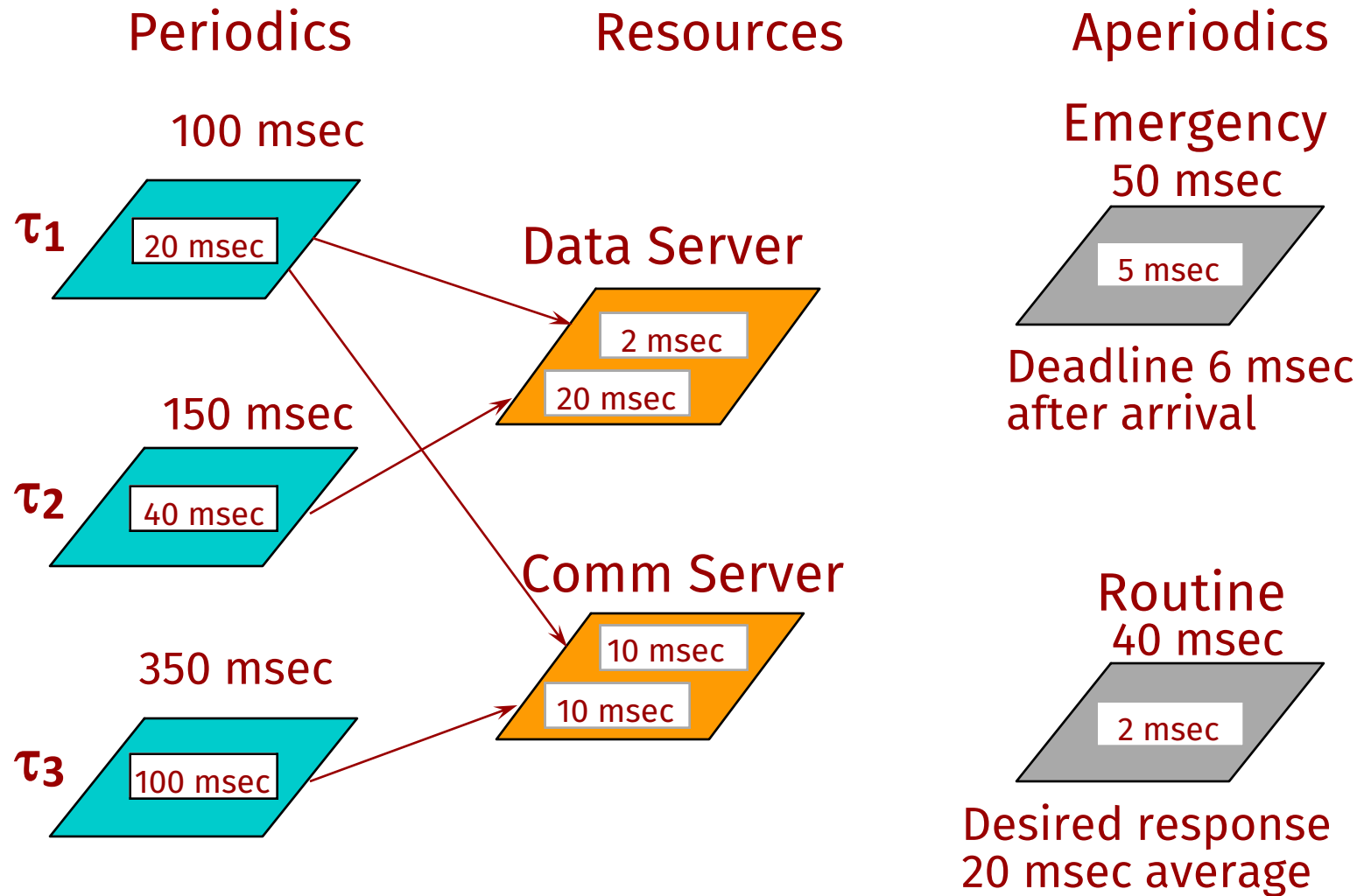
Real-Time Standards

- Real-Time Operating System standards
 - IEEE 1003.1b POSIX Real-Time Extensions
 - OSEK (automotive real-time OS standard)
- Real-Time (and Concurrent) Programming Languages
 - Real-Time Specification for Java
 - Ada 83 and Ada 95
- Real-Time Middleware
 - Real-Time CORBA
- Networks/buses
 - CANbus (Controller Area Network bus)
 - TTA: Time-Triggered Architecture (www.tttech.com)
 - FlexRay (www.flexray.org)

Plan for Lecture(s)

- Present basic theory for periodic task sets
- Extend basic theory to include
 - context switch overhead
 - preperiod deadlines
 - interrupts
- Consider task interactions:
 - priority inversion
 - synchronization protocols (time allowing)
- Extend theory to aperiodic tasks:
 - sporadic servers (time allowing)

A Sample Problem

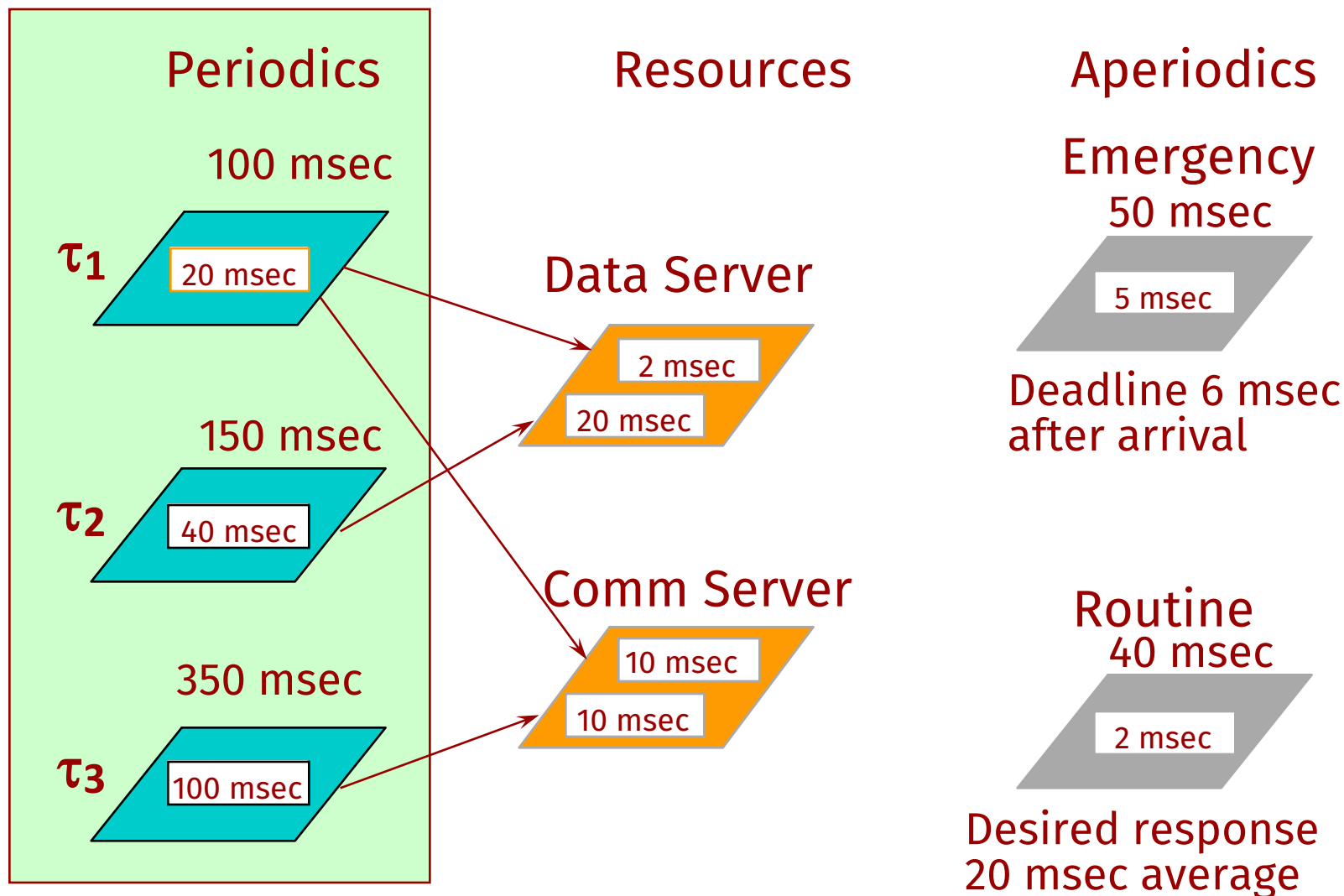


τ_2 's deadline is 20 msec before the end of each period

Rate Monotonic Analysis

- Introduction
- Periodic tasks
- Extending basic theory
- Synchronization and priority inversion
- Aperiodic servers

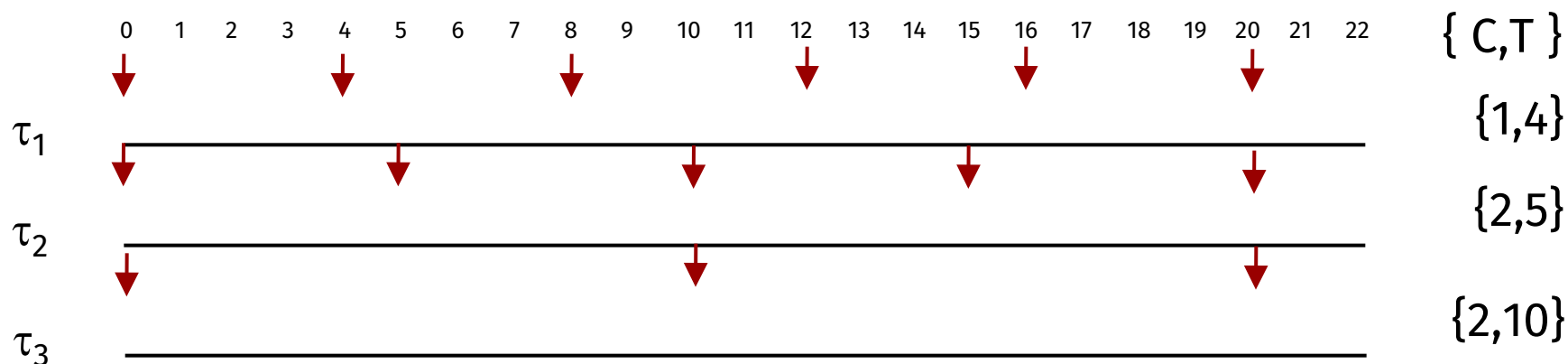
A Sample Problem - Periodics



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

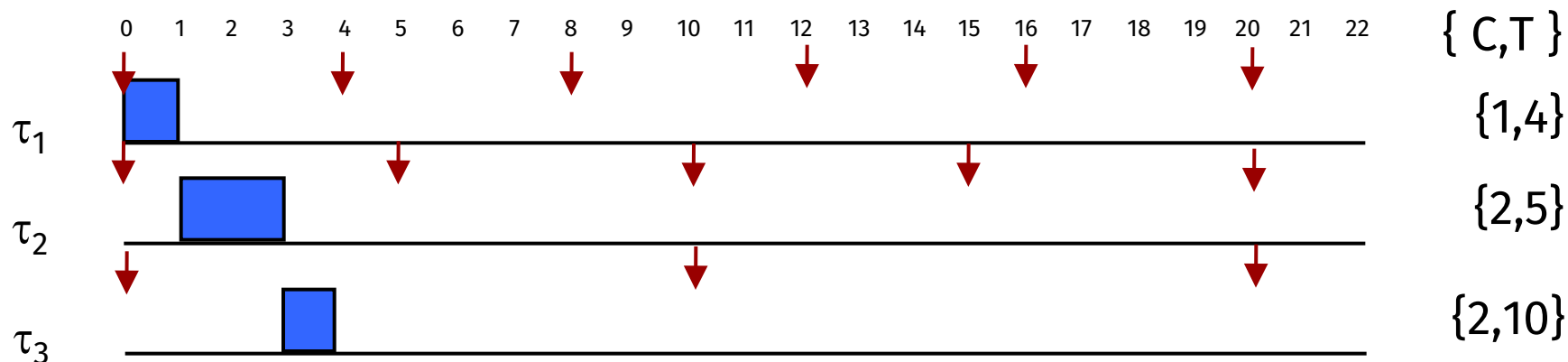
Rate Monotonic Scheduling

- Optimal Static Priority Scheduling
- A task with a shorter period has a higher deadline
 - Shorter Period \rightarrow Higher priority
- Assumption: Period is equal to Deadline



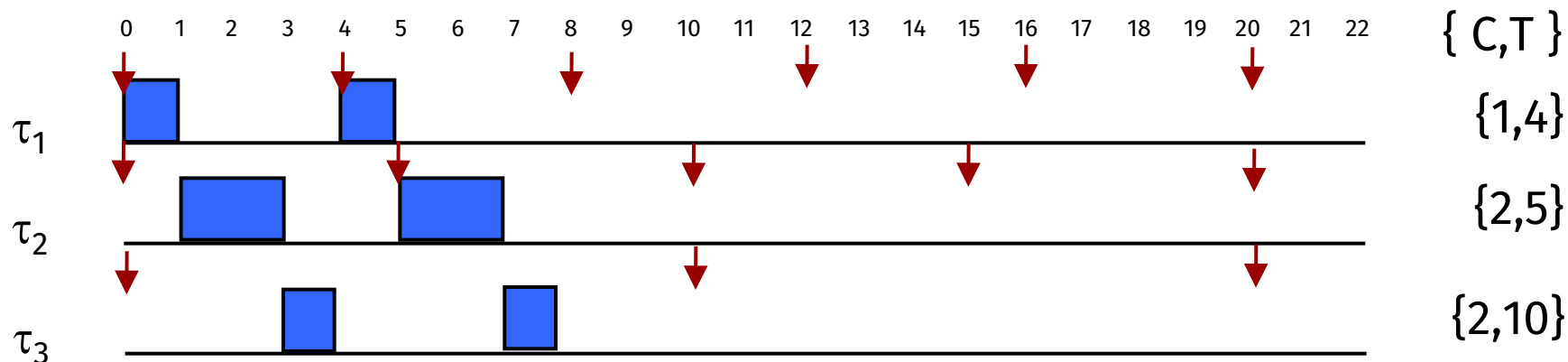
Rate Monotonic Scheduling

- Optimal Static Priority Scheduling
- A task with a shorter period has a higher deadline
 - Shorter Period \rightarrow Higher priority
- Assumption: Period is equal to Deadline



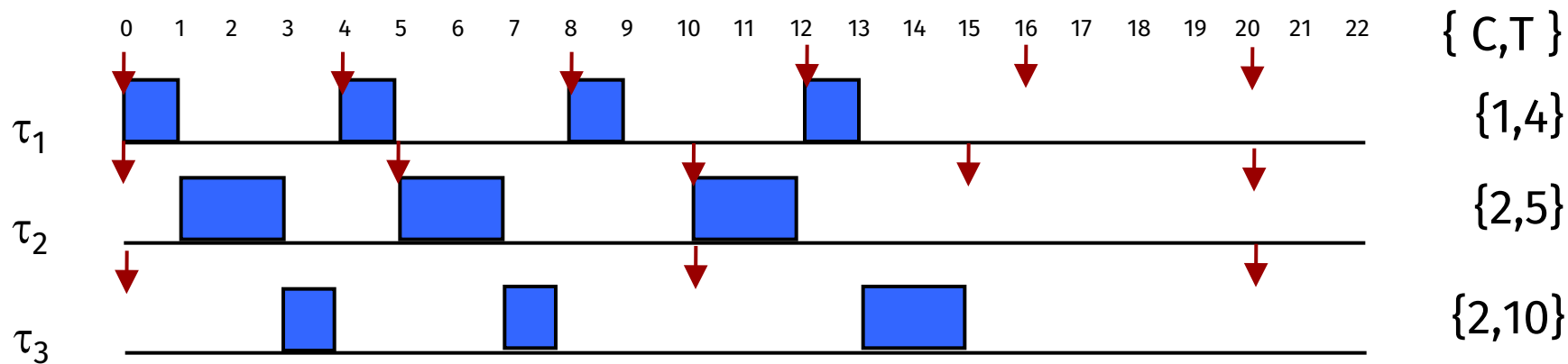
Rate Monotonic Scheduling

- Optimal Static Priority Scheduling
- A task with a shorter period has a higher deadline
 - Shorter Period \rightarrow Higher priority
- Assumption: Period is equal to Deadline



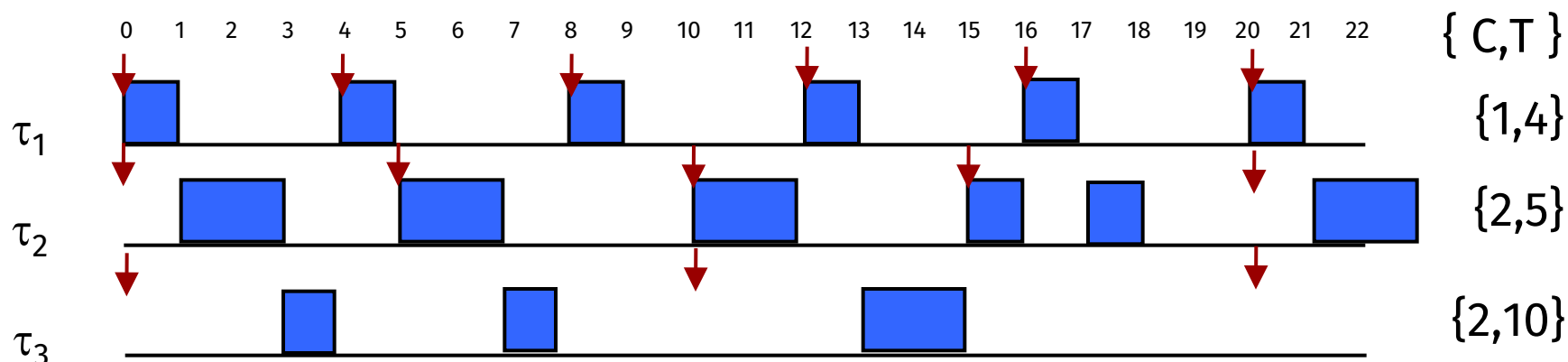
Rate Monotonic Scheduling

- Optimal Static Priority Scheduling
- A task with a shorter period has a higher deadline
 - Shorter Period \rightarrow Higher priority
- Assumption: Period is equal to Deadline



Rate Monotonic Scheduling

- Optimal Static Priority Scheduling
- A task with a shorter period has a higher deadline
 - Shorter Period \rightarrow Higher priority
- Assumption: Period is equal to Deadline



Rate Monotonic Scheduling

- What does “Optimal” mean?
 - **If any possible fixed (static) priority combination could work, then RMS will work**
- Rate Monotonic Analysis (RMA)
 - The proof that an RMS scheme is schedulable
 - Liu and Leyland (“*Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*”, JACM, 1973)

Concepts and Definitions - Periodics

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

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

- Task's CPU utilization:
 - 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

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

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

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

$$U(1) = 1.0 \quad U(4) = 0.756 \quad U(7) = 0.728$$

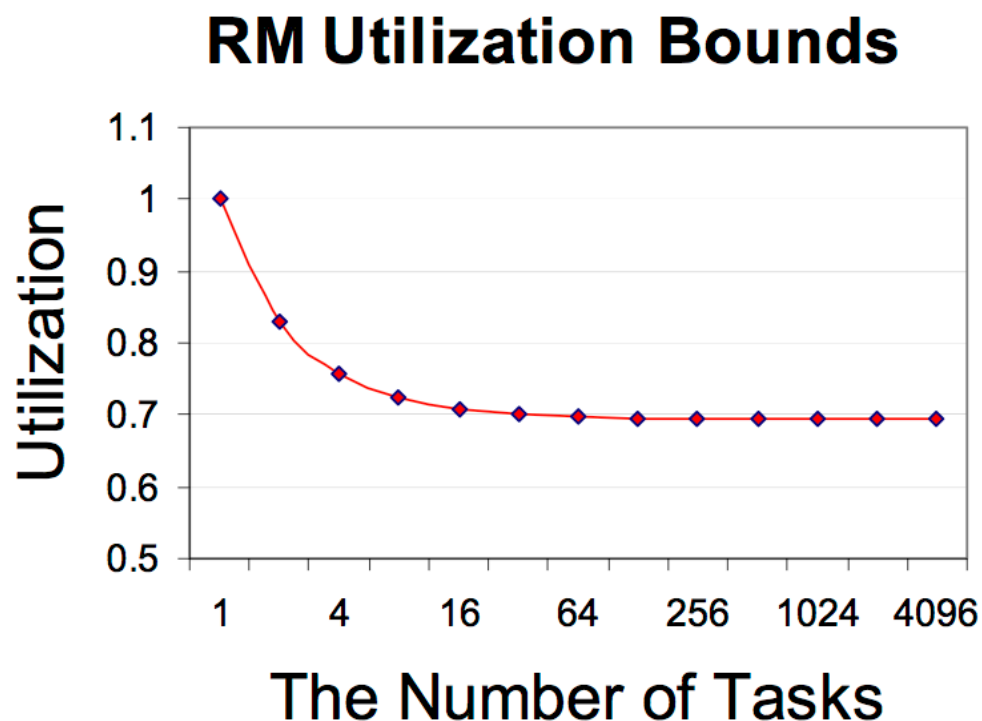
$$U(2) = 0.828 \quad U(5) = 0.743 \quad U(8) = 0.724$$

$$U(3) = 0.779 \quad U(6) = 0.734 \quad U(9) = 0.720$$

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

RMS Utilization Bound

- Real-time system is schedulable under RM if
Limit of $n(2^{1/n}-1)$ as $n \rightarrow \infty = 0.69$

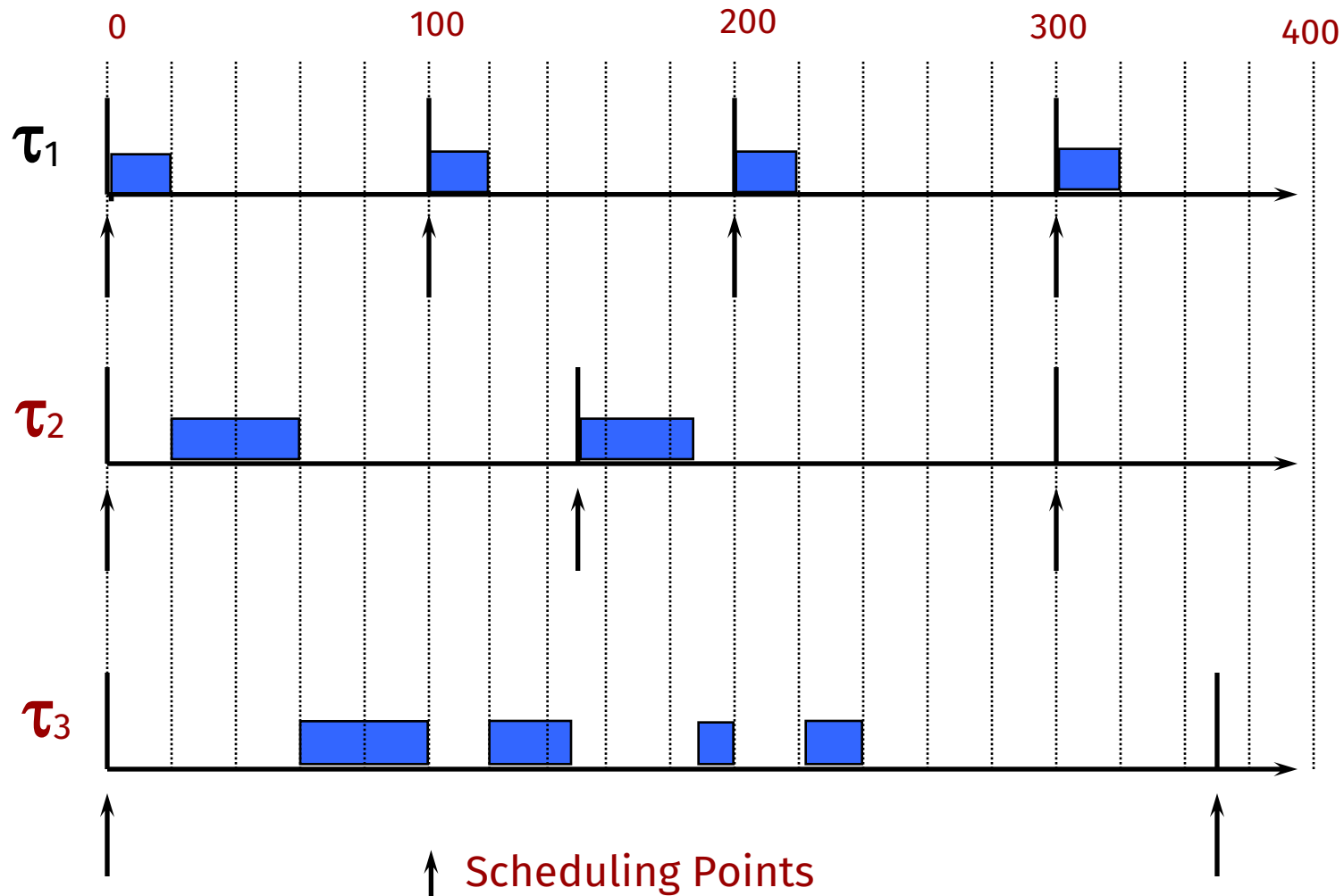


Sample Problem: Applying UB Test

	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

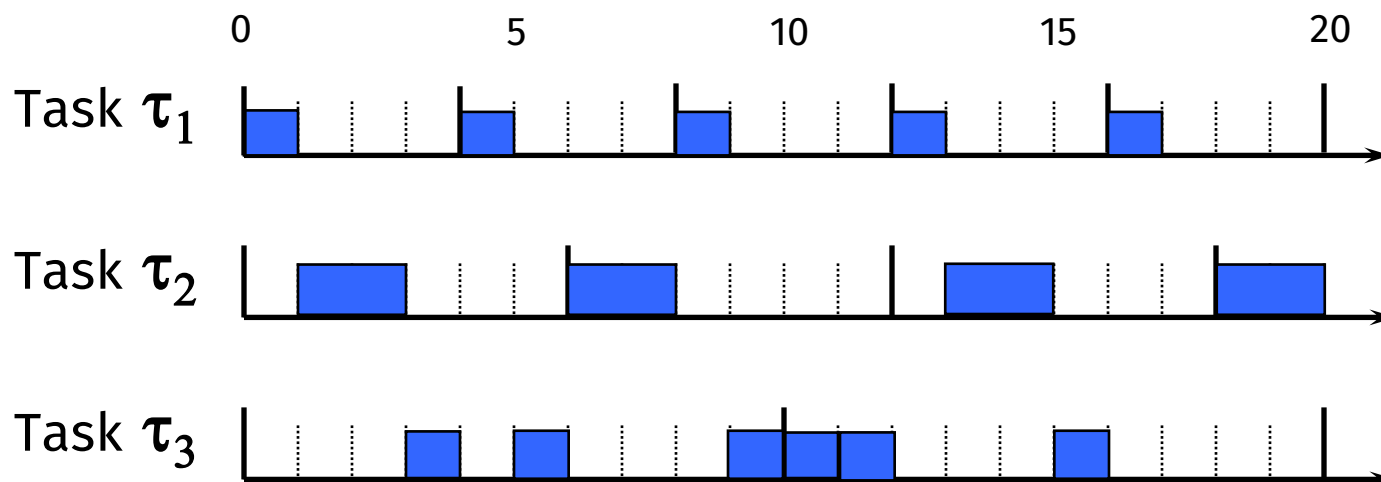
Given:

	C	T	U
Task $\tau_1: 1$	4		
Task $\tau_2: 2$	6		
Task $\tau_3: 1$	10		

- What is the total utilization?
- Is the task set schedulable?
- Draw the timeline.
- 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 $C_3 = 2$?
- $.25 + .34 + .20 = .79 > U(3) = .779$

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 **a_n = response time of task i** . a_n of task i may be computed by the following iterative formula:



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

- Test terminates when $a_{n+1} = a_n$.

- 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: $i = 3$

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

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

Example: Applying the RT Test -3

$$a_2 = c_3 + \sum_{j=1}^2 \left\lceil \frac{a_1}{T_j} \right\rceil c_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

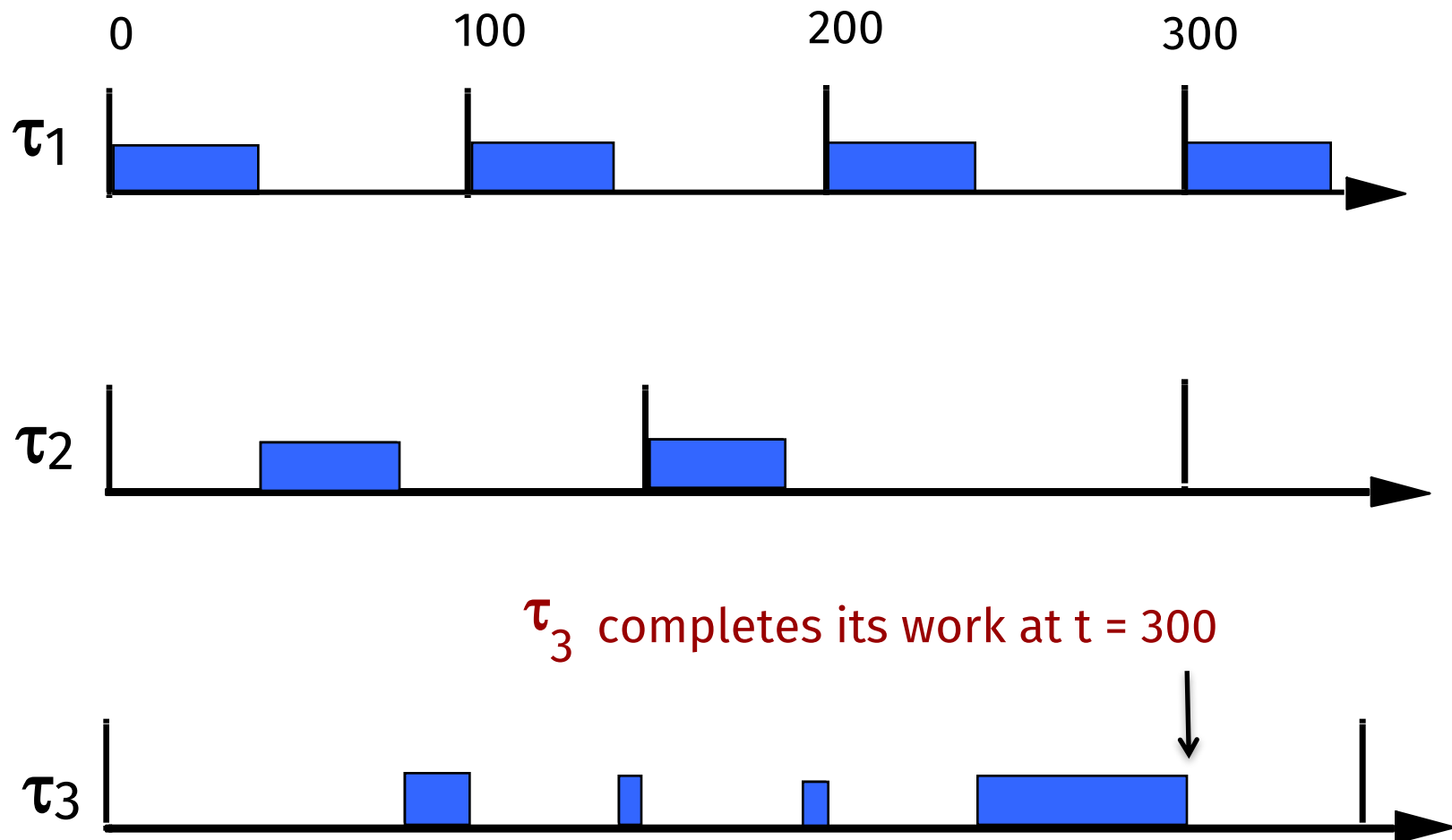
$$a_3 = c_3 + \sum_{j=1}^2 \left\lceil \frac{a_2}{T_j} \right\rceil c_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

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

- Task τ_3 is schedulable using RT test

$$a_3 = 300 < T = 350$$

Timeline for Example



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

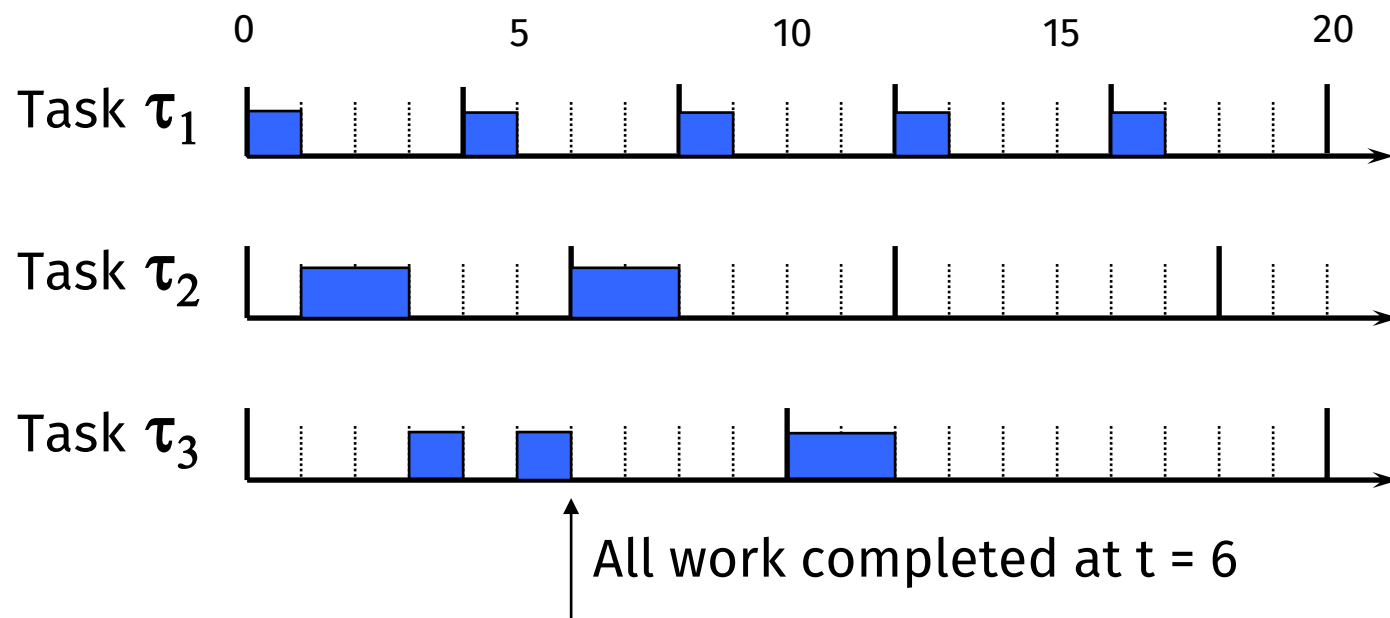
Solution: Applying RT Test

a) UB test

τ_1 and τ_2 OK -- no change from previous exercise

$.25 + .34 + .20 = .79 > .779 \implies$ 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

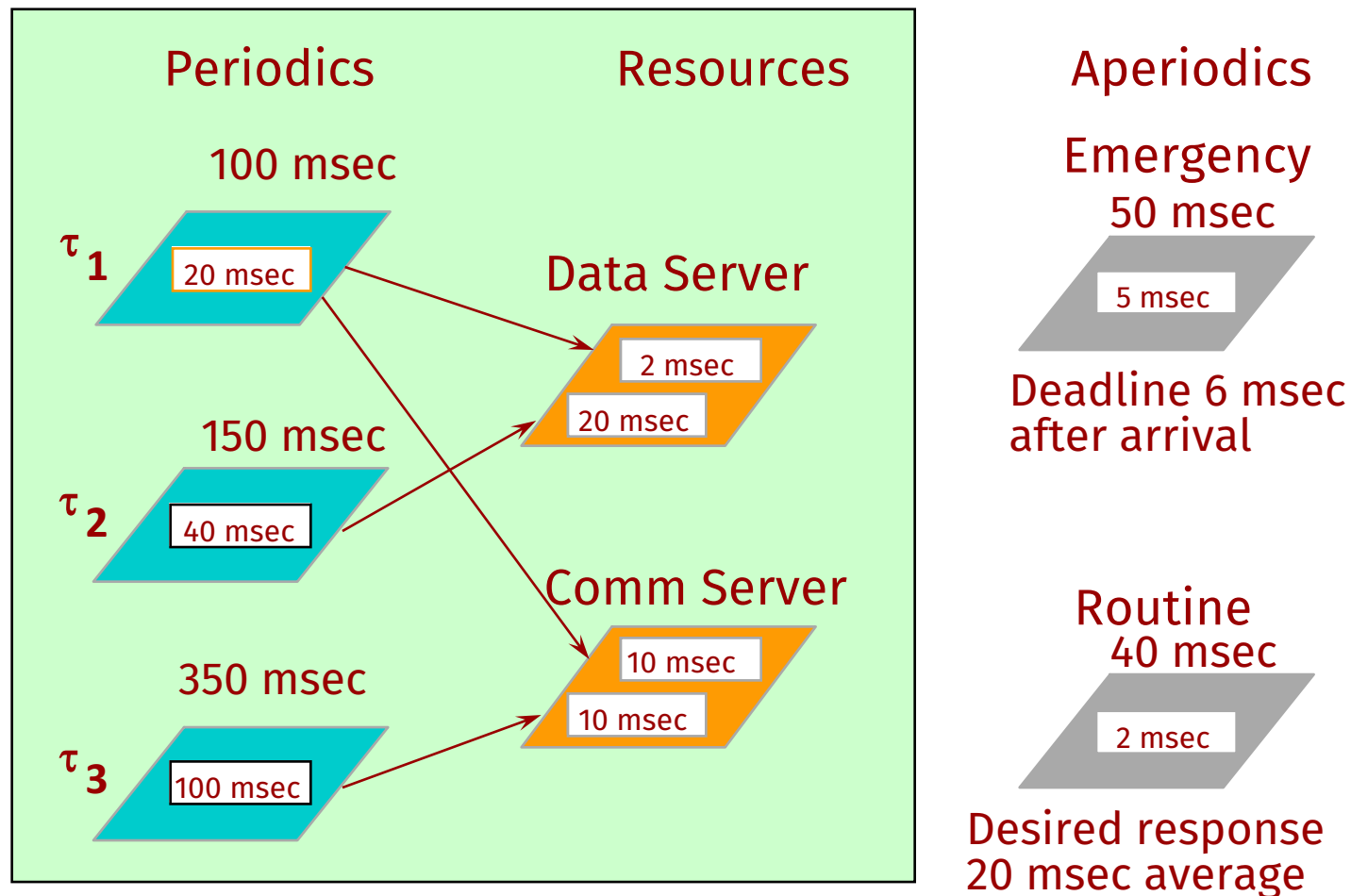
Summary

- 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
 - all tasks are periodic and noninteracting
 - deadlines are always at the end of the period
 - there are no interrupts
 - Rate-monotonic priorities are assigned
 - there is zero context switch overhead
 - tasks do not suspend themselves

Rate Monotonic Analysis

- Introduction
- Periodic tasks
- **Extending the basic theory**
- Synchronization and Priority Inversion
- Aperiodic servers

A Sample Problem

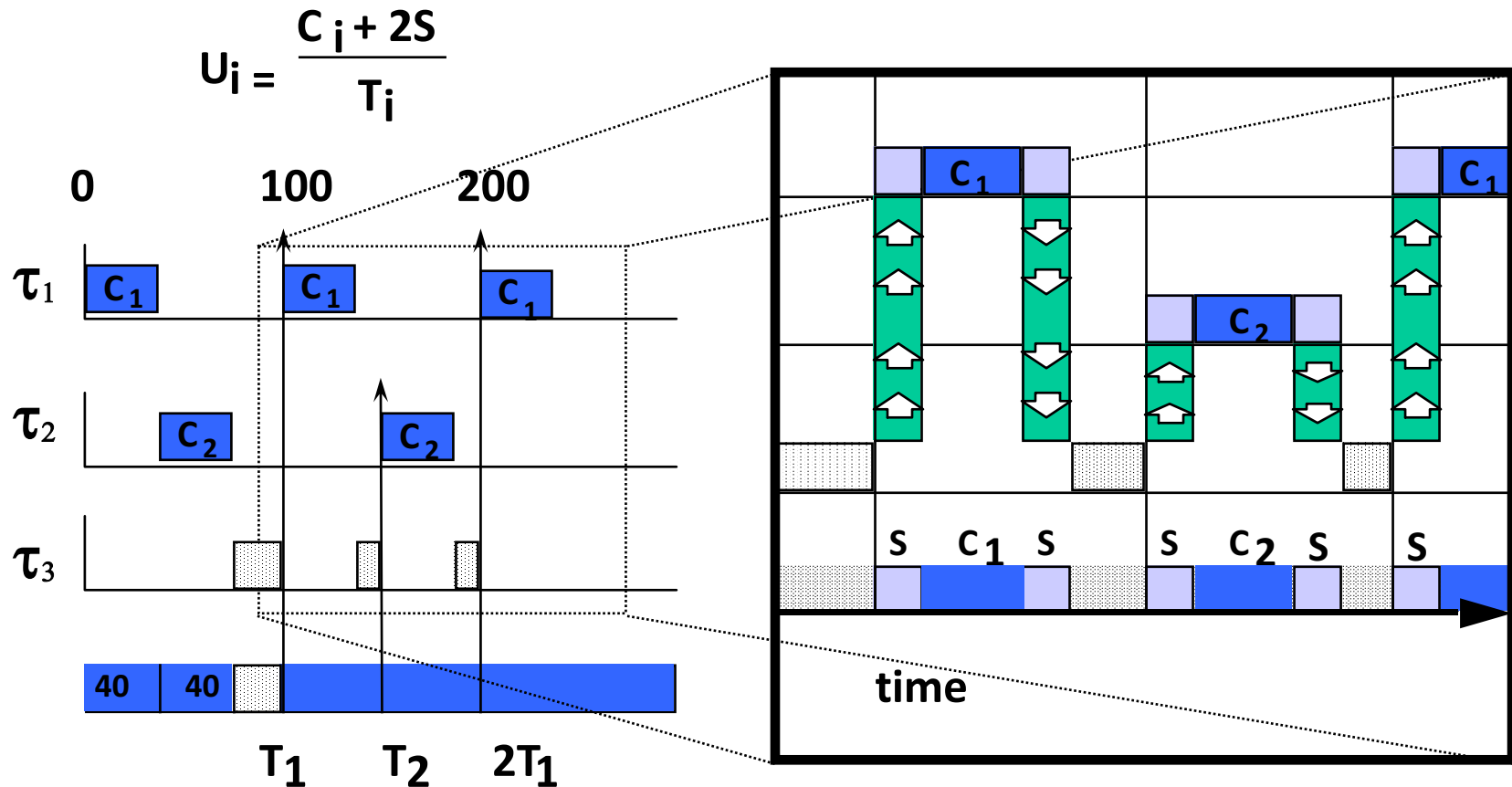


τ_2 's deadline is 20 msec before the end of each period

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



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:

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

where Δ_i is the ratio (D_i / T_i).

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

Schedulability with Interrupts

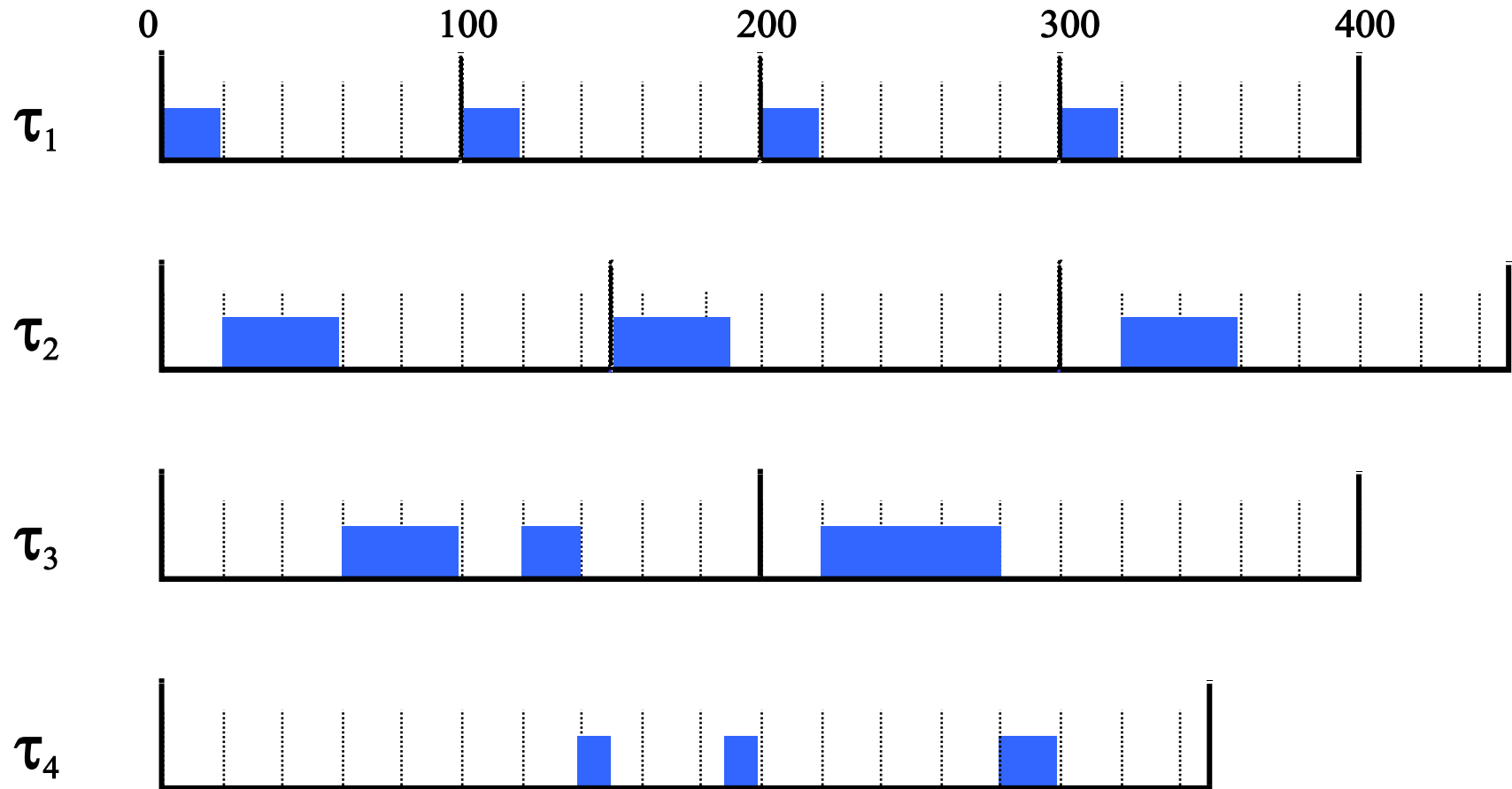
- Interrupt processing can be inconsistent with rate-monotonic priority assignment.
 - interrupt handler executes with high priority despite its period
 - interrupt processing may delay execution of tasks with shorter periods
- Effects of interrupt processing must be taken into account in schedulability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

Example: Determining Schedulability with Interrupts

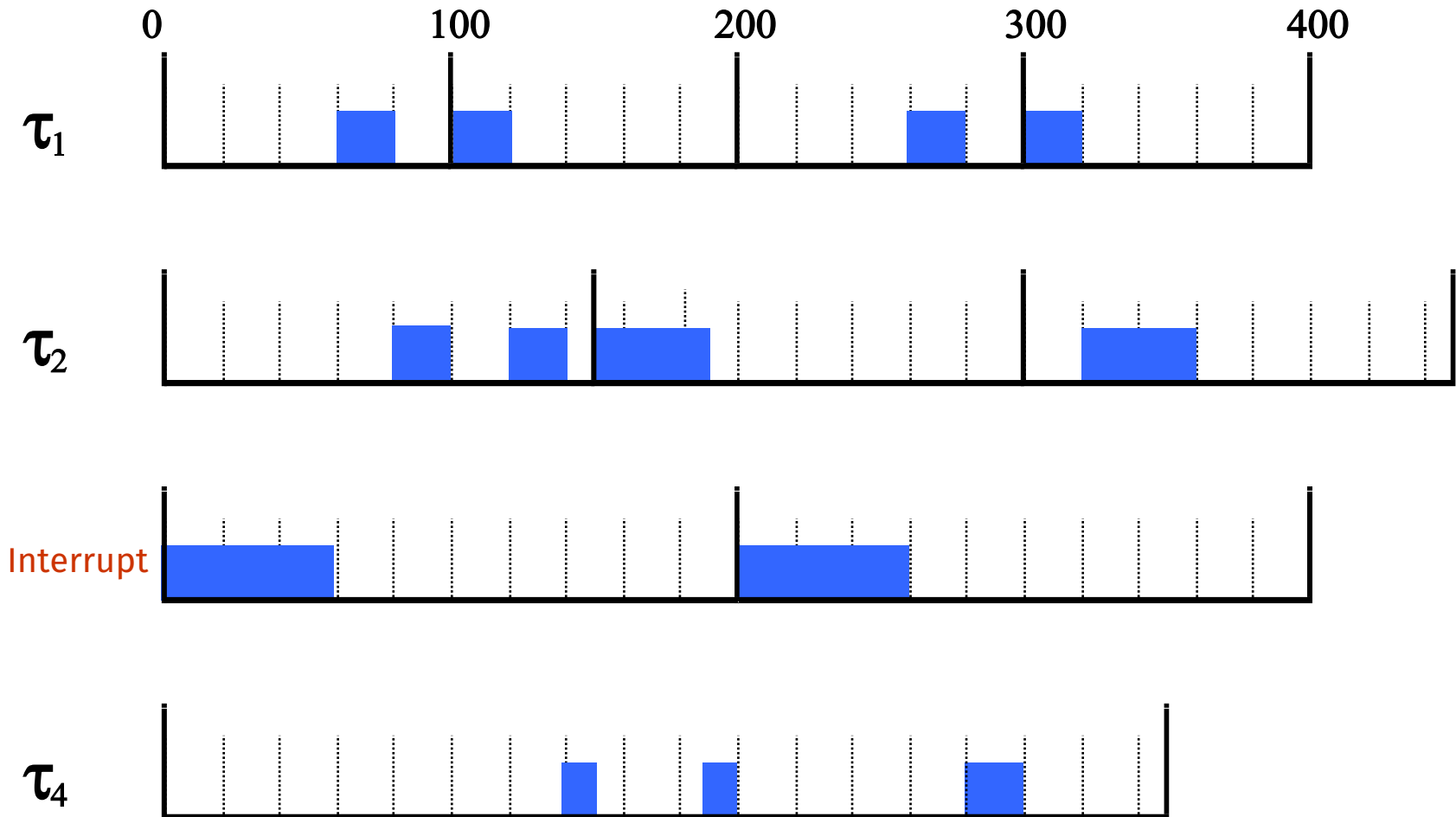
	C	T	U
Task τ_1 :	20	100	0.200
Task τ_2 :	40	150	0.267
Task τ_3 :	60	200	0.300
Task τ_4 :	40	350	0.115

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 $U(n)$.

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

UB Test with Interrupt Priority: t3

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

$$f_3 = \sum 0 + \frac{c_3}{T_3} + \sum 0$$

$\text{Notnum}(H_n) = 0$; 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:

$\text{num}(H_n) = 0$; 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:

$\text{num}(H_n) = 1$; 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 : $H = \{\tau_1, \tau_2, \tau_3\}$; $H_n = \{\tau_1, \tau_2, \tau_3\}$; $H_1 = \{ \}$.

$$f_4 = \sum_{j=1,2,3} \frac{c_j}{T_j} + \frac{c_4}{T_4} + \sum 0$$

Note:

$\text{num}(H_n) = 3$; therefore, utilization bound is $U(4)$.

Plugging in the numbers:

$$\begin{aligned} f_4 &= \frac{c_1}{T_1} + \frac{c_2}{T_2} + \frac{c_3}{T_3} + \frac{c_4}{T_4} \\ &= \frac{20}{100} + \frac{40}{150} + \frac{60}{200} + \frac{40}{350} = 0.882 > 0.756 \end{aligned}$$

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	HW	6
τ_2	4	1	High	3
τ_3	10	1	Low	10

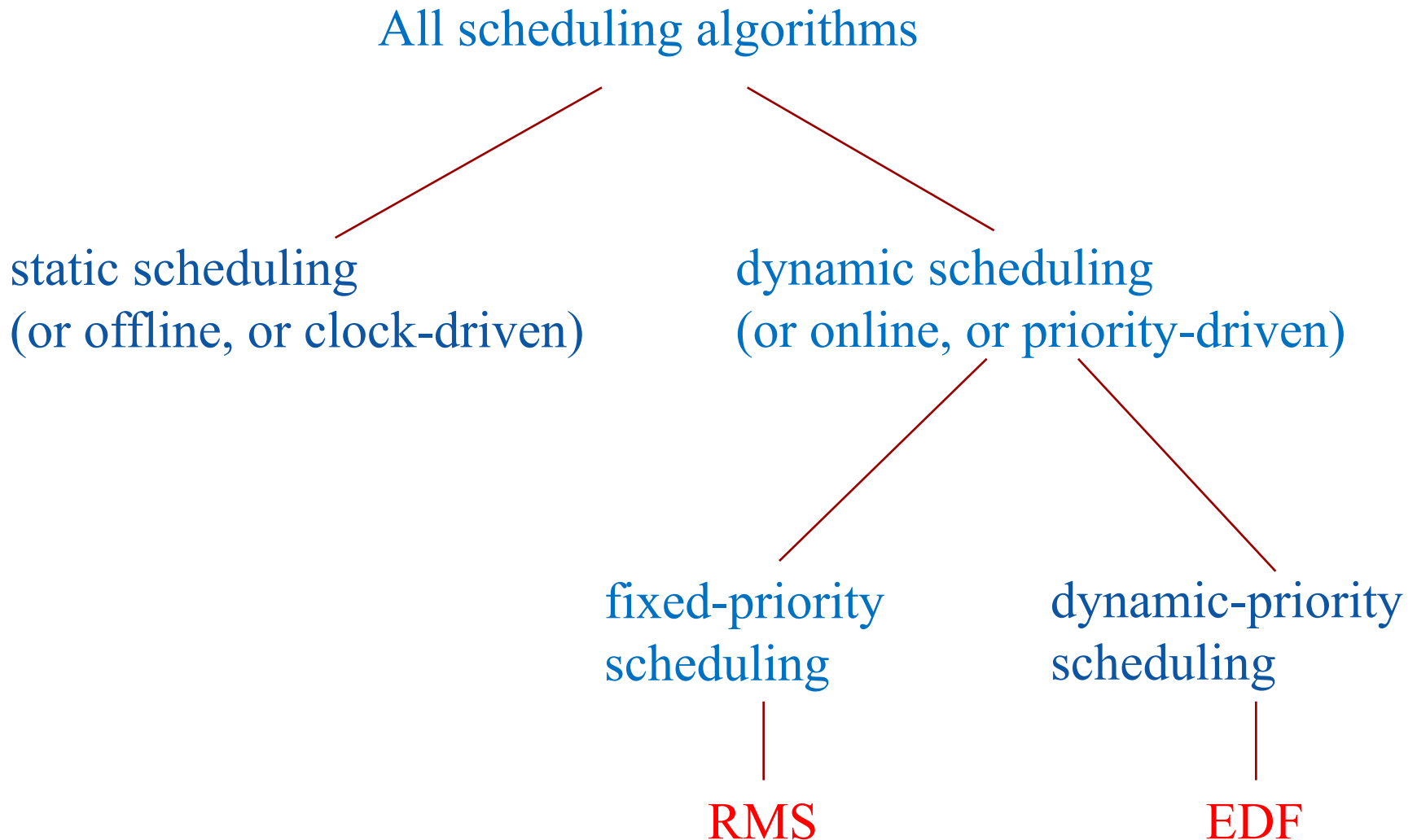
Solution: Schedulability with Interrupts

$$\frac{C_{int}}{T_{int}} \leq U \quad (1) \qquad 0.334 < 1.0$$

$$\frac{C_1}{T_1} + \overset{\{H1\}}{\frac{C_{int}}{T_1}} \leq U(1, .75) \qquad 0.250 + 0.500 = 0.750 = U(1, .75)$$

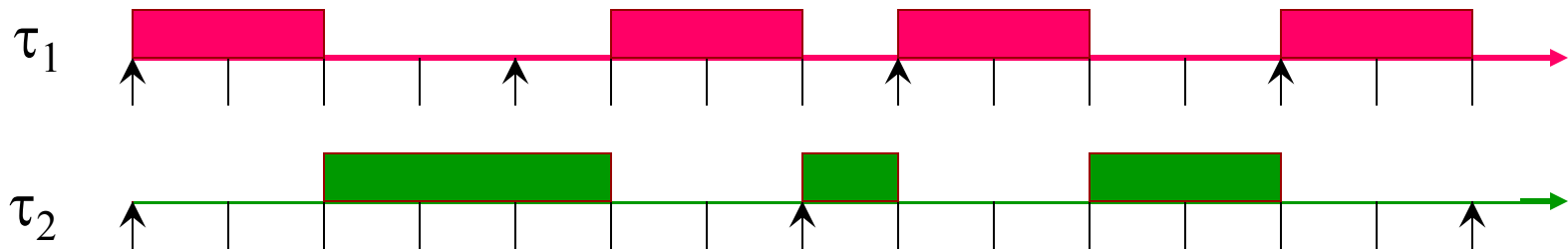
$$\overset{\{H_n\}}{\frac{C_{int}}{T_{int}} + \frac{C_1}{T_1} + \frac{C_2}{T_2}} \leq U(3) \qquad 0.334 + 0.250 + 0.100 = 0.684 < 0.779$$

Classification of Scheduling Algorithms



Earliest Deadline First (EDF) Scheduling

- Can be used to schedule periodic tasks
- Uses dynamic priorities and preemptive scheduling
 - Higher priority to task with earlier deadline
- **Example:** 2-task set where each task is $(C, T=D) \rightarrow \{(2, 4), (3, 7)\}$



- Utilization U of task set $\{\tau_i\}$ for $i = 1, \dots, n$:

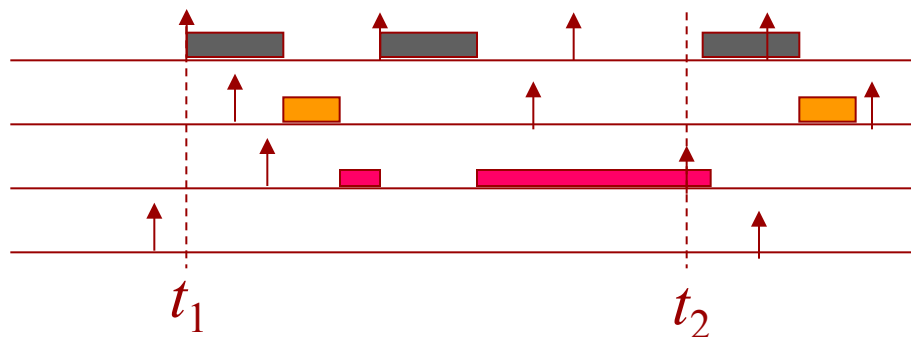
$$U_i = \frac{C_i}{T_i}$$

$$U = \sum_{i=1}^n U_i = \sum_{i=1}^n \frac{C_i}{T_i}$$

EDF Schedulability Condition

Theorem: A task set is schedulable under EDF if and only if $U \leq 1$.

Proof:



- Assume that “overflow” occurs at time t_2 .
- Let t_1 be the latest time before t_2 such that
 - the processor is fully utilized in the interval $[t_1, t_2]$
 - only instances with deadlines before t_2 executes in $[t_1, t_2]$
- If such a t_1 cannot be found, then set $t_1 = 0$.
- Let C_d be the computational demand in $[t_1, t_2]$

$$C_d = \sum_{r_i \geq t_1, d_i \leq t_2} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor * c_i \leq \sum_{i=1}^n \frac{t_2 - t_1}{T_i} * c_i = (t_2 - t_1)U$$

- But an overflow implies that $C_d > (t_2 - t_1)$: a contradiction if $U \leq 1$.

Points to Note

- If deadlines are shorter than periods, the necessary condition for schedulability under EDF is an open problem.
- If $U > 1$, which task will first miss its deadline is unpredictable.

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 $U(n, D_i)$
 - 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

Summary

- Present basic theory for periodic task sets
- Extend basic theory to include
 - context switch overhead
 - preperiod deadlines
 - Interrupts

Next Lecture:

- Consider task interactions:
 - priority inversion
 - synchronization protocols (time allowing)
- Extend theory to aperiodic tasks:
 - sporadic servers (time allowing)