Overview

- Today: this lecture explains how to analyze schedulability with pre-period deadlines and blocking times. We also introduce aperiodic task scheduling.

- **To learn more on real-time scheduling**:
- **see chapter 4 on "Hard Real-Time Computing Systems" book from G. Buttazzo** (useful chapters are in the Lab!)

What really happened on Mars?

- **The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface**. Successes included its unconventional "landing" -- bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "**the computer was trying to do too many things at once**".

- **VxWorks provides preemptive priority scheduling of threads**. Tasks on the Pathfinder spacecraft were executed as threads with priorities that were assigned in the usual manner reflecting the relative urgency of these tasks.

- Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft. **A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus**. **Access to the bus was synchronized with mutual exclusion locks (mutexes).**

- You can read more at http://feanor.sssup.it/~giorgio/mars/jones.html

What really happened on Mars?

- **The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex**. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue. **The spacecraft also contained a communications task that ran with medium priority.**

- Most of the time this combination worked fine. However, **very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread**. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. *After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.*

- This scenario is a classic case of priority inversion.

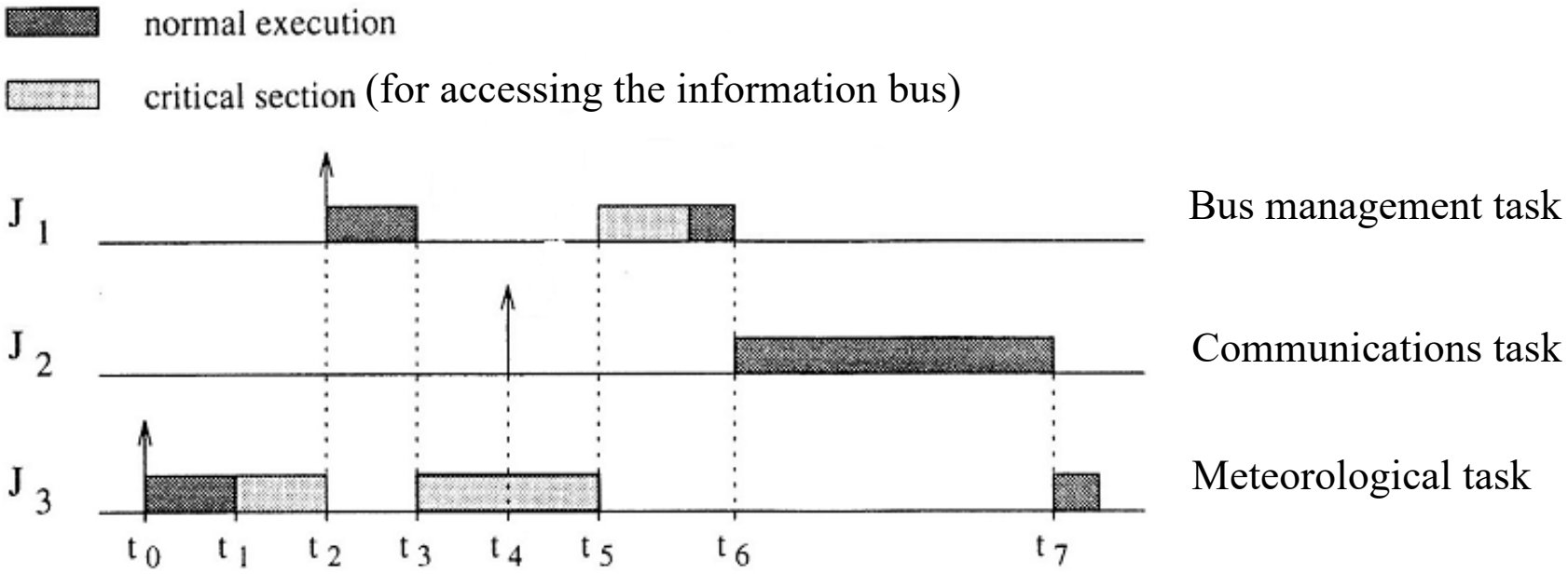
What really happened on Mars?

- **How was this debugged?** *VxWorks can be run in a mode where it records a total trace of all interesting system events, including context switches, uses of synchronization objects, and interrupts*. After the failure, JPL engineers spent hours and hours running the system on the exact spacecraft replica in their lab with tracing turned on, attempting to replicate the precise conditions under which they believed that the reset occurred. **The engineers finally reproduced a system reset on the replica. Analysis of the trace revealed the priority inversion**.

- **How was the problem corrected?**

What really happened on Mars?

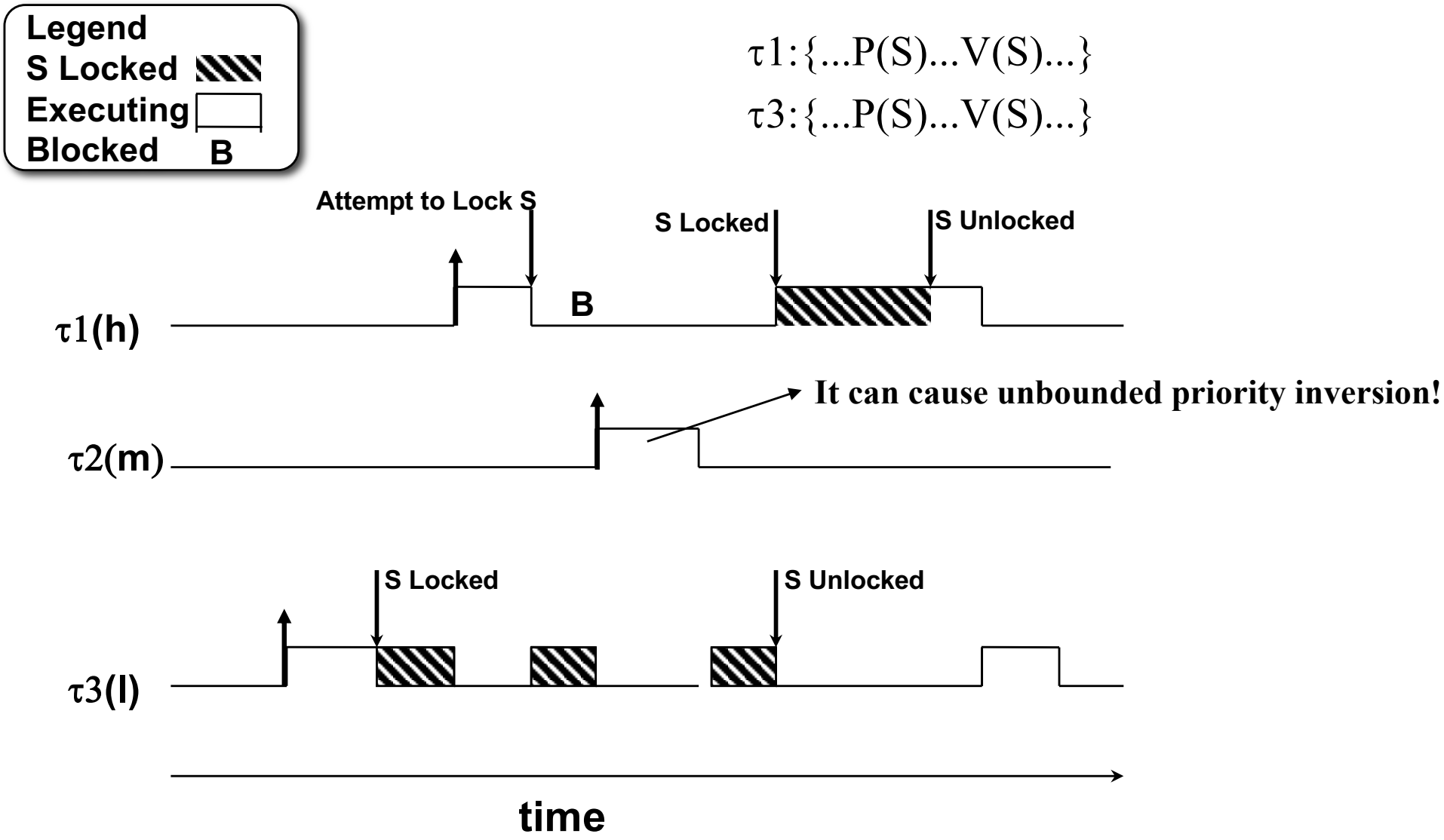
- **How was the problem corrected?** When created, a VxWorks mutex object accepts a boolean parameter that indicates whether **priority inheritance** should be performed by the mutex. The mutex in question had been initialized with the parameter off; *had it been on, the low-priority meteorological thread would have inherited the priority of the high-priority data bus thread blocked on it while it held the mutex, causing it be scheduled with higher priority than the medium-priority communications task, thus preventing the unbounded priority inversion*. **Leaving the "debugging" facilities in the system saved the day. Without the ability to modify the system in the field, the problem could not have been corrected.**



Unbounded Priority Inversion

- When a high priority task is delayed by lower priority tasks, it is said that priority inversion has occurred and the high priority task is blocked by the lower priority task.
- Priority inversion occurs during synchronization.
- When tasks synchronize, we expect delays due to the use of mutual exclusion.
- And we expect that the delay due to mutual exclusion is a function of the duration of the critical sections.
- When the duration of priority inversion is not bounded by a function of the duration of critical sections, unbounded priority inversion is said to occur.

Unbounded Priority Inversion



The importance of good theory/algorithms

- **Finally, the engineer's initial analysis that "the data bus task executes very frequently and is time-critical -- we shouldn't spend the extra time in it to perform priority inheritance" was exactly wrong. It is precisely in such time critical and important situations where correctness is essential, even at some additional performance cost.**

- **The paper that first identified the priority inversion problem and proposed the solution was:**

- L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. In IEEE Transactions on Computers, vol. 39, pp. 1175-1185, Sep. 1990.

Exact Schedulability Test (Exact Analysis)

$$r_i^{k+1} = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{r_i^k}{p_j} \right\rceil c_j, \quad \text{where } r_i^0 = \sum_{j=1}^{i} c_j$$

Test terminates when $r_i^{k+1} > p_i$ (not schedulable)
or when $r_i^{k+1} = r_i^k \leq p_i$ (schedulable).

Tasks are ordered according to their priority: T_1 is the highest priority task.

The superscript *k* indicates the number of iterations in the calculation.
The index *i* indicates it is the *ith* task being checked.

The index *j* runs from 1 to i-1, i.e. all the higher priority tasks. Recall from the convention - task 1 has a higher priority than task 2 and so on.

<u>We check the schedulability of a single task at the time!!!</u>

Class Exercise 1

Suppose that we have two tasks

- $c_1 = 3$, $p_1 = 5$
- $c_2 = 5$, $p_2 = 14$

- Use the exact test to check the schedulability of task 2. Draw a timeline to confirm that.

Class Exercise 1

Suppose that we have two tasks

- $c_1 = 3, p_1 = 5$
- $c_2 = 5, p_2 = 14$

- Use the exact test to check the schedulability of task 2. Draw a timeline to confirm that.

- $r_2^0 = c_1 + c_2 = 3 + 5 = 8$

$$r_2^1 = c_2 + \left\lceil \frac{r_2^0}{p_1} \right\rceil c_1 = 5 + \left\lceil \frac{8}{5} \right\rceil 3 = 11$$

$$r_2^2 = c_2 + \left\lceil \frac{r_2^1}{p_1} \right\rceil c_1 = 5 + \left\lceil \frac{11}{5} \right\rceil 3 = 14$$

$$r_2^3 = c_2 + \left\lceil \frac{r_2^2}{p_1} \right\rceil c_1 = 5 + \left\lceil \frac{14}{5} \right\rceil 3 = 14$$

Done! ➔ the task set is schedulable

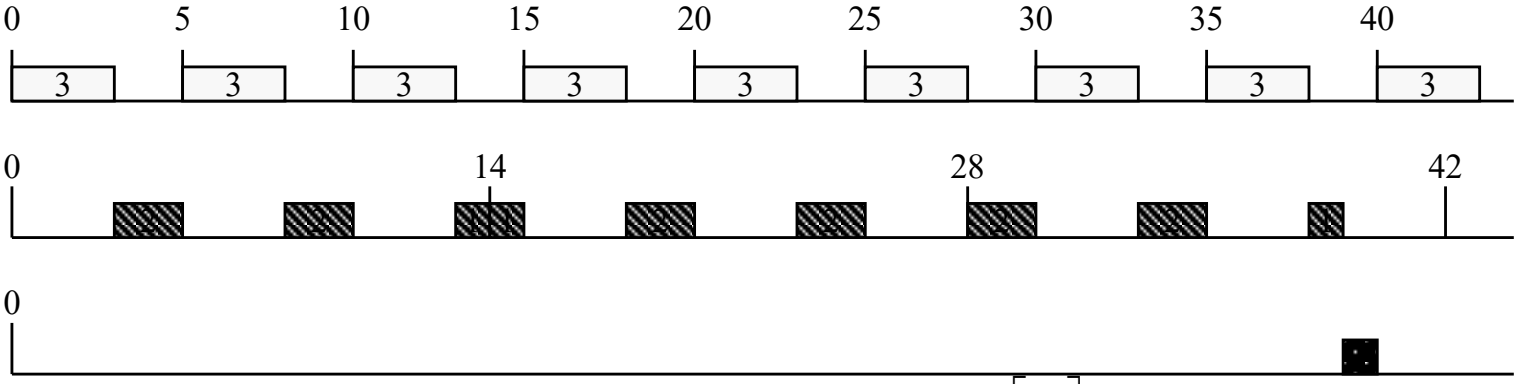
Class Exercise 1

Suppose that we have two tasks

- $c_1 = 3$, $p_1 = 5$
- $c_2 = 5$, $p_2 = 14$

- Quiz: Can we add a task T3 with $c_3 = 1$ and $p_3 = 50$?
- Quiz: What would be the shortest period p_3 such that T3 can still meet its deadlines? (Apply the exact test formulation to confirm that)

Class Exercise 1 (continued)



$$r_3^0 = \sum_{j=1}^{3} C_j = 3+5+1 = 9$$

$$r_3^1 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^0}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{9}{5} \right\rceil 3 + \left\lceil \frac{9}{14} \right\rceil 5 = 12$$

$$r_3^2 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^1}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{12}{5} \right\rceil 3 + \left\lceil \frac{12}{14} \right\rceil 5 = 15$$

$$r_3^3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^2}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{15}{5} \right\rceil 3 + \left\lceil \frac{15}{14} \right\rceil 5 = 20$$

$$r_3^4 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^3}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{20}{5} \right\rceil 3 + \left\lceil \frac{20}{14} \right\rceil 5 = 23$$

$$r_3^5 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^4}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{23}{5} \right\rceil 3 + \left\lceil \frac{23}{14} \right\rceil 5 = 26$$

$$r_3^6 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^5}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{26}{5} \right\rceil 3 + \left\lceil \frac{26}{14} \right\rceil 5 = 29$$

$$r_3^7 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^6}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{29}{5} \right\rceil 3 + \left\lceil \frac{29}{14} \right\rceil 5 = 34$$

$$r_3^8 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^7}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{34}{5} \right\rceil 3 + \left\lceil \frac{34}{14} \right\rceil 5 = 37$$

$$r_3^9 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^8}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{37}{5} \right\rceil 3 + \left\lceil \frac{37}{14} \right\rceil 5 = 40$$

$$r_3^{10} = C_3 + \sum_{j=1}^{2} \left\lceil \frac{r_3^9}{T_j} \right\rceil C_j = 1 + \left\lceil \frac{40}{5} \right\rceil 3 + \left\lceil \frac{40}{14} \right\rceil 5 = 40$$

Class Exercise 1

Suppose that we have two tasks

- $c_1 = 3$, $p_1 = 5$
- $c_2 = 5$, $p_2 = 14$

- Quiz: Can we add a task T3 with $c_3 = 1$ and $p_3 = 50$?
- **Answer**: Yes!

- Quiz: What would be the shortest period p_3 such that T3 can still meet its deadlines? (Apply the exact test formulation to confirm that)
- **Answer**: the shortest period is $p_3 = 40$.

Class Exercise 2

Suppose that three tasks are scheduled under RMS:

- T1 ➔ $c_1 = 4$, $p_1 = 10$
- T2 ➔ $c_2 = 6.1$, $p_2 = 14$
- T3 ➔ $c_3 = 1$, $p_3 = 70$

- Is task T2 schedulable?
- How about the schedulability of task T3?

Class Exercise 2: Task 2

- $c_1 = 4, \quad p_1 = 10$
- $c_2 = 6.1, \quad p_2 = 14$
- $c_3 = 1, \quad p_3 = 70$

$$r_2^0 = 4 + 6.1 = 10.1$$

$$r_2^1 = \left\lceil \frac{10.1}{10} \right\rceil \cdot 4 + 6.1 = 14.1 > 14$$

Task T2 is not schedulable!

Class Exercise 2: Task 3

$$a_0 = 4 + 6.1 + 1 = 11.1$$

$$a_1 = \left\lceil \frac{11.1}{10} \right\rceil 4 + \left\lceil \frac{11.1}{14} \right\rceil 6.1 + 1 = 15.1$$

$$a_2 = \left\lceil \frac{15.1}{10} \right\rceil 4 + \left\lceil \frac{15.1}{14} \right\rceil 6.1 + 1 = 21.2$$

$$a_3 = \left\lceil \frac{21.2}{10} \right\rceil 4 + \left\lceil \frac{21.2}{14} \right\rceil 6.1 + 1 = 25.2$$

$$a_4 = \left\lceil \frac{25.2}{10} \right\rceil 4 + \left\lceil \frac{25.2}{14} \right\rceil 6.1 + 1 = 25.2 < 70$$

Even if task T2 is not schedulable, task T3 is schedulable.

It is a common mistake to assume that if a higher priority task is not schedulable so are the lower priority tasks. Don't make this mistake!

Context Switch Time

- So far, we have analyzed the schedulability of a periodic task set assuming that the scheduler does not introduce any overhead. This is just an approximation; to be more precise, we should include scheduling overhead in our analysis

- The context switching time will be identified by **S**: amount of time to deschedule the current executing task and schedule a new one.

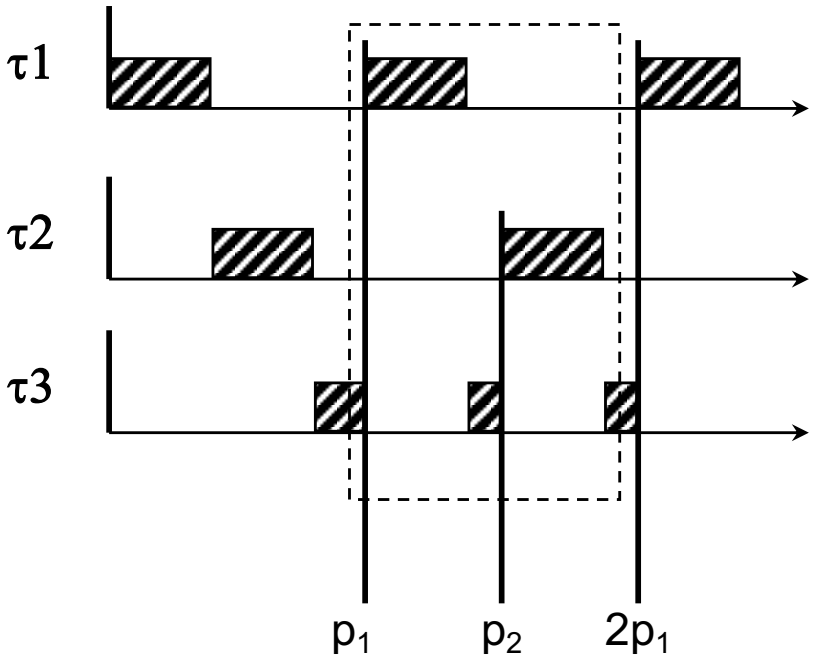
- **When the context switching overhead is not negligible, add 2S to execution time of each task and perform the analysis as if there is no context switching time.**

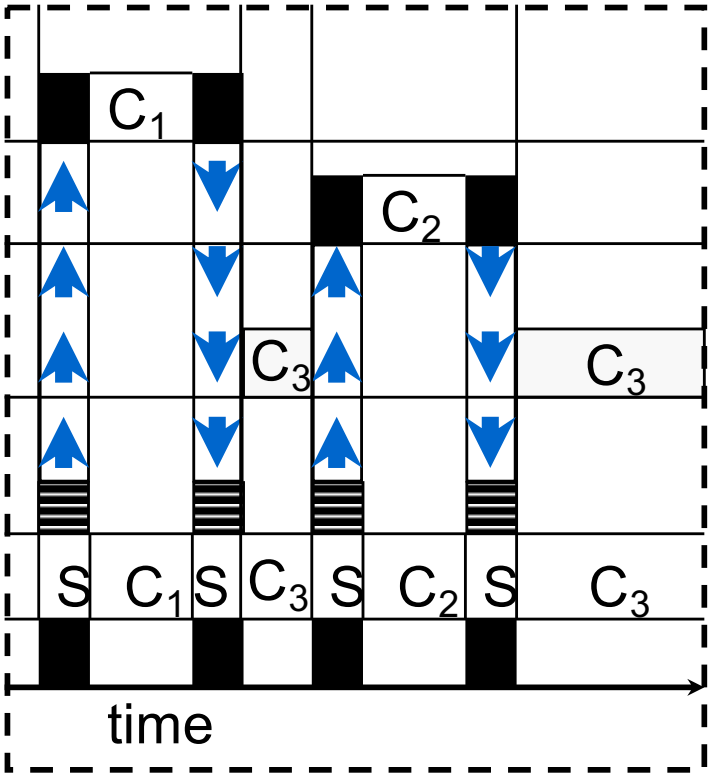
- Why do we model the context switch by adding 2S to execution time of each task?

  (see next slide!)

Modeling Task Switching as Execution Time

- An instance of a periodic task can cause at most (worst case) two context switches. To simplify the analysis, we shall assume that this is always the case.





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

CS

Analyzing Task Switching

- The effect of context switching can be directly added to the computation time, that is, replace $C_i = (C_i + 2S)$ in BOTH the UB test and exact test. For example:

$$\tau_1 \qquad \frac{(C_1 + 2S)}{P_1} \le U(1)$$

$$\tau_2 \qquad \frac{(C_1 + 2S)}{P_1} + \frac{(C_2 + 2S)}{P_2} \le U(2)$$

$$\tau_3 \qquad \frac{(C_1 + 2S)}{P_1} + \frac{(C_2 + 2S)}{P_2} + \frac{(C_3 + 2S)}{P_3} \le U(3)$$

Check schedulability of τ_1 and τ_2

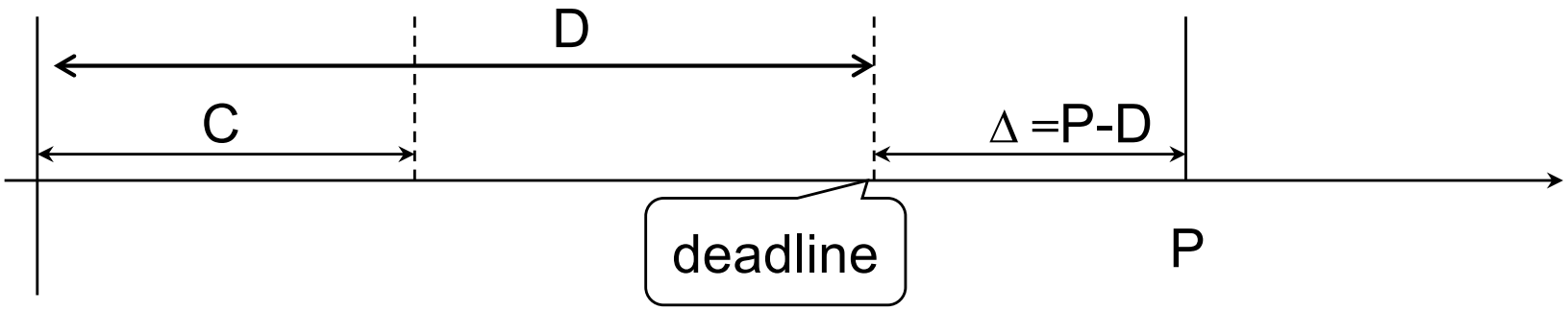
Check schedulability of τ_1, τ_2 and τ_3

Modeling Pre-period Deadlines

Suppose task τ, with computation time C and period P, has a relative deadline D < P.

- In UB tests, pre-period deadline can be modeled as if the task has a longer execution time (C+ Δ), because if the task has execution time (C+ Δ) and can finish before time P, then we know it must finish Δ units before P if it has only execution time C.

- In exact schedulability test, just move the deadline from P to D.



Task Switching and Pre-period Deadline under RM

- Suppose that task 2 has D_2 as relative deadline, we just add $\Delta_2 = p_2 - D_2$ to task 2's execution time ***LOCALLY*** (Why?)

$$\tau_1 \quad \frac{(C_1 + 2S)}{p_1} \leq U(1)$$

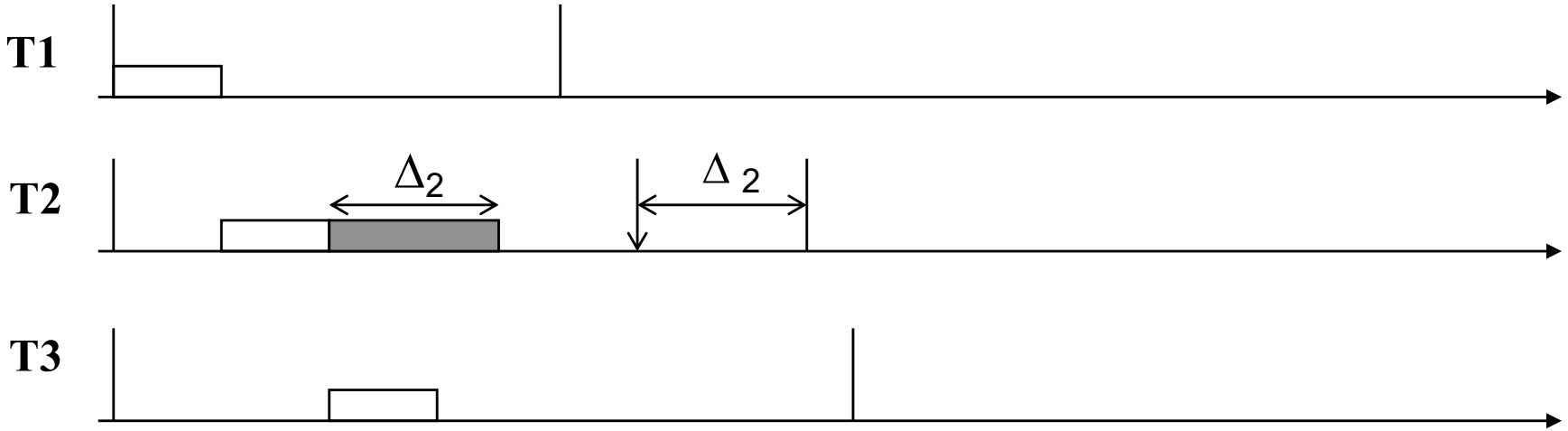
$$\tau_2 \quad \frac{(C_1 + 2S)}{p_1} + \frac{(C_2 + 2S + \Delta_2)}{p_2} \leq U(2)$$

$$\tau_3 \quad \frac{(C_1 + 2S)}{p_1} + \frac{(C_2 + 2S)}{p_2} + \frac{(C_3 + 2S)}{p_3} \leq U(3)$$

- Notice that tasks have to be ordered according to RM priorities: τ_1 is the highest priority task in the system!

Task Switching and Pre-period Deadline under RM

- Suppose that task 2 has D2 as relative deadline, we just add $\Delta 2 = p2 - D2$ to task 2's execution time ***LOCALLY*** (Why?)



- We inflate the execution time of task T2 only when we check its schedulability because the pre-period deadline of a task does not affect the schedulability of other tasks!
- Utilization Bound with pre-preriod deadlines becomes a task by task test: if you have N tasks, you need to check N equations to verify the schedulability of the entire task set (The original UB studied in lecture 9 allowed to check the schedulability of the entire task set by checking a single equation!!!).

Quiz: UB test and pre-period deadline

- Consider the Utilization Bound test:

- We have modeled the effect of pre-period deadline by adding (P-D) to execution time C.

$$U_i = \frac{C_i + (P_i - D_i)}{P_i}$$

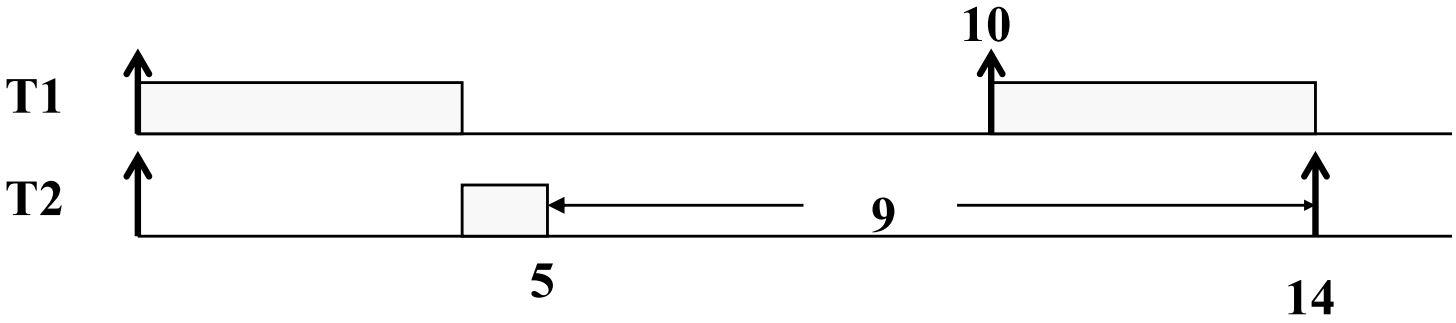
Task utilization when modeling pre-period deadline by inflating execution time!

- **Do you think it is correct to subtract (P-D) from the period P?**
- (Hint: are the 2 methods equivalent? Is there a counter example?)

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

Task utilization when modeling pre-period deadline by reducing period!

Quiz: UB test and pre-period deadline



- **<u>Answer</u>:** we can find a counter example showing that the second method is wrong.

  ➔ If we reduce the period, the UB test might give the wrong result because changing the period affects the priority of a task too. (see below!)

- Consider the case of two tasks, { (C_1 = 4, P_1 = 10), (C_2 = 1, P_2 = 14)}. If D_2 is less than 5, then Task T2 will not be schedulable.

- Assume D_2=4, 4/10 + 1/4 = 0.4 + 0.25 = 0.65 < U(2). That is, it tells us that even if the relative deadline is at D_2=4, it is still schedulable. **<u>This is obviously wrong!!!</u>**

Quiz: UB test and pre-period deadline

- Consider the Utilization Bound test:

- We have modeled the effect of pre-period deadline by adding (P-D) to execution time C.

$$U_i = \frac{C_i + (P_i - D_i)}{P_i}$$

Task utilization when modeling pre-period deadline by inflating execution time!

- **Do you think it is correct to subtract (P-D) from the period P?**
- (Hint: are the 2 methods equivalent? Is there a counter example?)

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

Task utilization when modeling pre-period deadline by reducing period!

Example: Schedulability with Task Switching and Pre-period Deadline under RM

Given the following tasks:

	C	p	D
Task τ1	1	4	
Task τ2	2	6	5
Task τ3	2	10	

Assume S = 0.05, are these 3 tasks schedulable?

Example: Schedulability with Task Switching and Pre-period Deadline under RM

	C	p	D
Task τ1	1	4	
Task τ2	2	6	5
Task τ3	2	10	

Assume S = 0.05, are these 3 tasks schedulable?

Solution: Schedulability with Task Switching and Pre-period Deadline

τ_1 $\quad \frac{(1+2(0.05))}{4} = 0.275 \leq U(1) = 1.00$

τ_2 $\quad \frac{(1+2(0.05))}{4} + \frac{(2+2(0.05)+1)}{6} = 0.791 \leq U(2) = 0.828$

τ_3 $\quad 0.275 + 0.35 + \frac{(2+2(0.05))}{10} = 0.835 > U(3) = 0.779$

Exact analysis:

$r_3^0 = 1.1 + 2.1 + 2.1 = 5.3$

$r_3^1 = 2.1 + \text{ceil}(5.3/4) * 1.1 + \text{ceil}(5.3/6) * 2.1 = 6.4$

$r_3^2 = 2.1 + \text{ceil}(6.4/4) * 1.1 + \text{ceil}(6.4/6) * 2.1 = 8.5$

$r_3^3 = 2.1 + \text{ceil}(8.5/4) * 1.1 + \text{ceil}(8.5/6) * 2.1 = 9.6$

$r_3^4 = 2.1 + \text{ceil}(9.6/4) * 1.1 + \text{ceil}(9.6/6) * 2.1 = 9.6$

$$r_i^{k+1} = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{r_i^k}{p_j} \right\rceil \cdot c_j$$

Task 3 is schedulable, it completes by 10!