Embedded and Real-Time Systems

Spring 2021

Hamed Farbeh

farbeh@aut.ac.ir

Department of Computer Engineering

Amirkabir University of Technology

Lecture 10

Copyright Notice

This lecture is adopted from

IN4343 Real-Time Systems Course 2018 – 2019, Mitra Nasri, Delft University of Technology

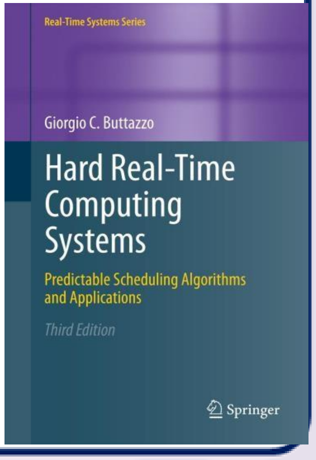
Online scheduling of periodic tasks

R. Devillers and J. Goossens. Liu and layland's schedulability test revisited. *Information Processing Letters*, 73(5):157–161, March 2000.

C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.

Disclaimer: A few slides have been taken from Giorgio Buttazzo's website:

http://retis.sssup.it/~giorgio/rts-MECS.html


Real-Time Systems Series
Giorgio C. Buttazzo
Hard Real-Time Computing Systems
Predictable Scheduling Algorithms and Applications
Third Edition
Springer

Buttazzo's book, chapter 4

Agenda

- Secrets behind L&L and hyperbolic bound tests
- RM schedulability analysis
    - **Response-time analysis**
    - **Park test**
    - **A test for Harmonic task sets**

If you want to pass the course, learn this lecture very well

Assumptions

- For this lecture, we assume

  A1. $\boldsymbol{C_i}$ and $\boldsymbol{T_i}$ are constant for every job of τ_i

  A2. Tasks are <u>fully preemptive</u>

  A3. Context switch, preemption, and scheduling overheads are <u>zero</u>

  A4. Tasks are <u>independent</u>:

  - no precedence relations
  - no resource constraints
  - no blocking on I/O operations
  - no self suspension

Assume that tasks are indexed according to their priority ordering, namely,
$P_1 < P_2 < P_3 < \cdots < P_n$

Example

Is the task set feasible?

Yes $U \leq 1$

Does L&L test accept this task set?

No because $U > 2(2^{1/2} - 1) \sim 0.83$

Does hyperbolic bound test accept this task set?

No because $(0.4 + 1) \times (0.6 + 1) = 2.24 > 2$

Is the task set schedulable by RM? (think)

τ_i	C_i	T_i	D_i	U_i
τ_1	3	5	5	0.6
τ_2	4	10	10	0.4

$U = 1$

$$\sum_{i=1}^{n} U_i \leq 1$$

necessary

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$$

Liu and Layland test

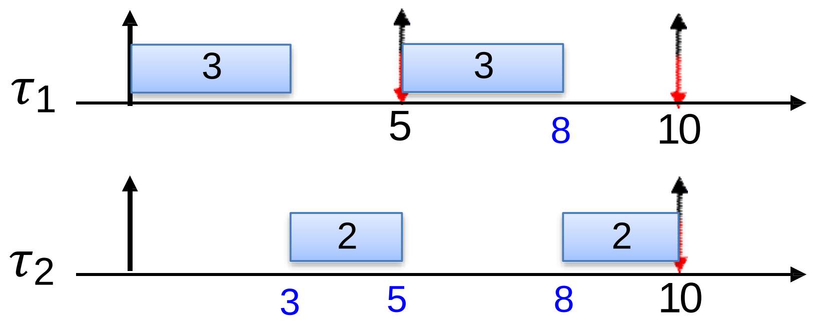
$$\prod_{i=1}^{n} (U_i + 1) \leq 2$$

Hyperbolic bound

Example

Is the task set schedulable by RM? (think)

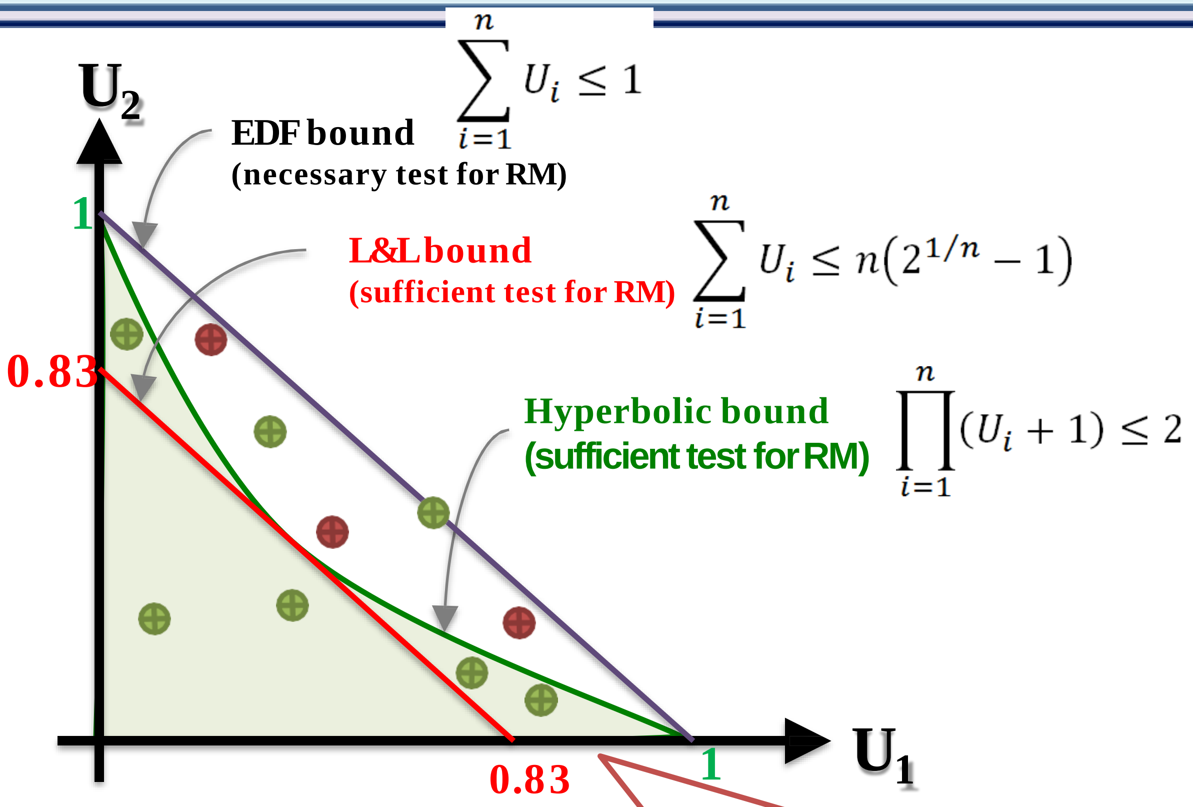
Yes! Here is the schedule:



τ_i	C_i	T_i	D_i	U_i
τ_1	3	5	5	0.6
τ_2	4	10	10	0.4

U =1

Utilization-based tests and RM scheduling



Bad news:
We cannot have a better utilization-based test than the hyperbolic bound!

Under what circumstances the hyperbolic bound is better than the L&L test?

When the utilization values are far apart! E.g., {0.9, 0.01}

Hyperbolic bound is tight

It is impossible to invent a new utilization-based test A such that A accepts a task set that the hyperbolic bound rejects!

In other words, as long as you only consider task utilizations into account, it is impossible to invent a test that is better than the hyperbolic bound!

Secrets behind
the Liu and Layland and hyperbolic bound tests

They found the

hardest-to-schedule task set with the minimal utilization

The hardest-to-schedule task set with the minimal utilization:

- It is schedulable by RM
- It fully utilizes the processor
  if the execution time of any of the tasks is increased by ϵ, then the task set is not schedulable anymore
- It has the minimum utilization among all task sets that fully utilize the processor

These task sets are on the boundary: by increasing any execution time, they will become unschedulable!

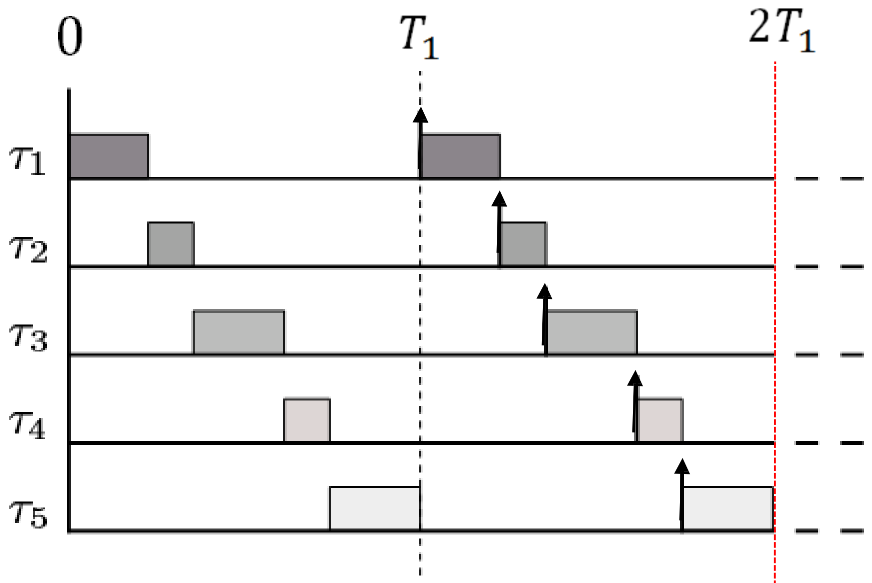
The smallest utilization that a *barely schedulable task set* can have, is our utilization threshold!

$$\sum_{i=1}^{n} U_i \leq n\left(2^{1/n} - 1\right) \qquad \prod_{i=1}^{n} (U_i + 1) \leq 2$$

Exam hint: Pay attention to the definition of the hardest-to-schedule task set.

The hardest-to-schedule task set with the minimal utilization

- All tasks are released at the same time (no release offset)
- **Condition 1**: Periods follow $T_1 < T_2 < \cdots < T_n < 2T_1$
- **Condition 2**: Execution times follow:



$$C_1 = T_2 - T_1$$

$$C_2 = T_3 - T_2$$

$$C_i = T_{i+1} - T_i$$

$$C_n = 2T_1 - T_n$$

Why this generates a fully-utilized task set?

Exam hint: Remember these conditions for the exam

Proof highlights

Goal:

Prove that the hardest-to-schedule task set (defined in the previous slide) has the minimum utilization among all other task sets that fully utilize the processor.

Step1: prove the condition on periods

Claim: In a fully-utilized task set with the minimum utilization, periods must follow

$$T_1 < T_2 < \cdots < T_n < 2T_1$$

Step2: prove the condition on execution times

(assuming that the condition on periods hold)

Assumption: given a set of periods $T_1 < T_2 < \cdots < T_n < 2T_1$

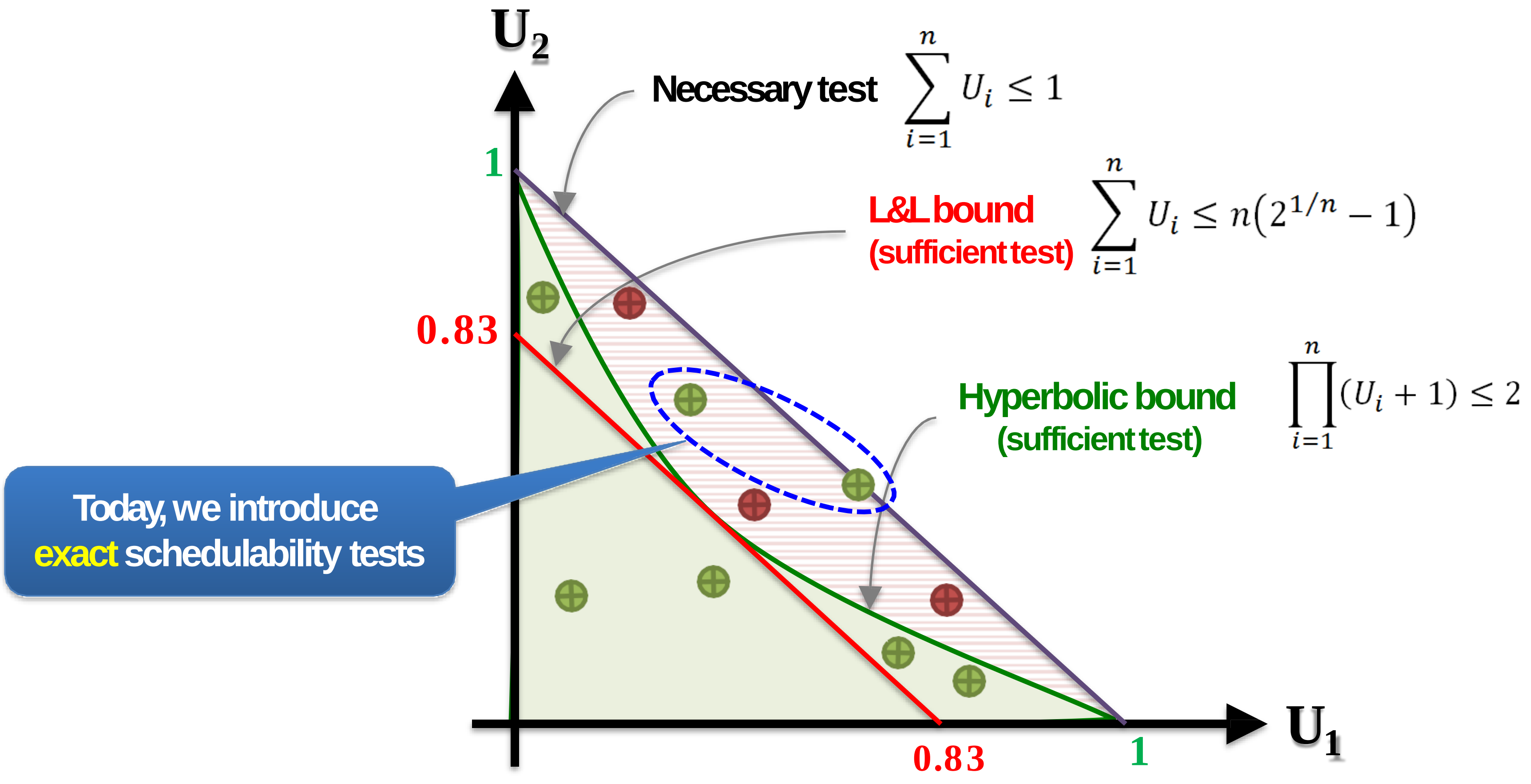
Claim: the utilization of a task set that fully utilizes the processor will be minimum only if $C_i = T_{i+1} - T_i$, $C_n = 2T_1 - T_n$

This slide is just for your information (not in the exam). The interested students can follow the whole proof in the extra slides.

Agenda

- Secrets behind L&L and hyperbolic bound tests
- RM schedulability analysis
  - **Response-time analysis**
  - **Park test**
  - **A test for Harmonic task sets**

Previously on Real-Time Systems



Necessary test $\sum_{i=1}^{n} U_i \leq 1$

L&L bound (sufficient test) $\sum_{i=1}^{n} U_i \leq n\left(2^{1/n} - 1\right)$

Hyperbolic bound (sufficient test) $\prod_{i=1}^{n} (U_i + 1) \leq 2$

Today, we introduce exact schedulability tests

Exact schedulability test

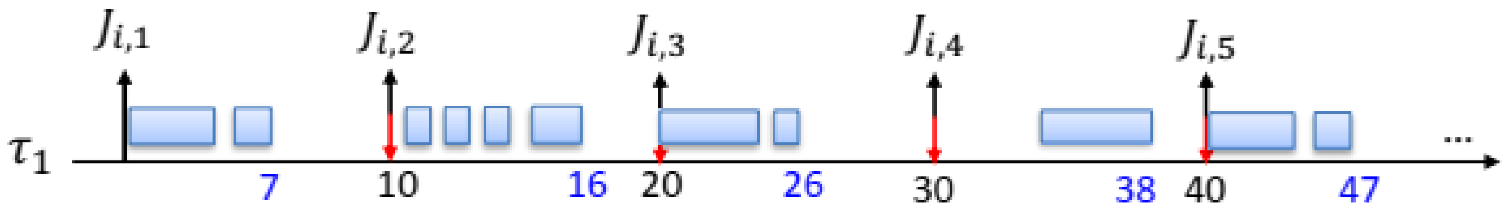
If an **exact schedulability test** for **scheduling algorithm A** **accepts** a task set, then the task set is **certainly schedulable** by the algorithm,
and if the test **rejects** the task set, then the task set is **certainly NOT schedulable** by the algorithm.

A task set is schedulable by algorithm A **if and only if** it is accepted by an exact schedulability test for algorithm A

Response-time analysis (RTA)

Unlike utilization-based tests, response-time analysis takes task's period and worst-case execution time (**WCET**) into account!

Visualization:



Worst-case response time (WCRT):

$$R_i = \max\{R_{i,j} \mid \forall j, 1 \le j \le \infty\}$$

It is not practical as it is!
We need a smarter solution

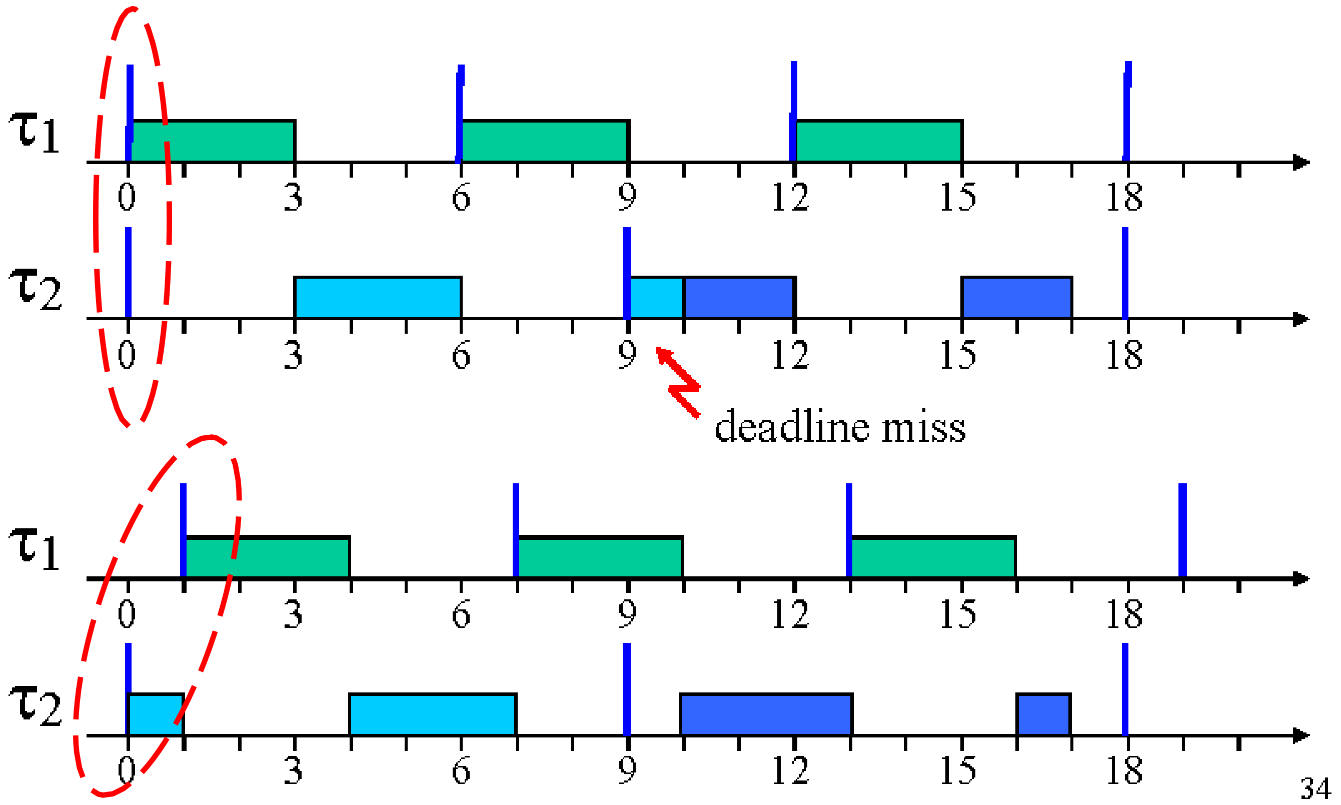
The test:

$$\forall \tau_i \in \tau, \qquad R_i \le D_i$$

For fixed-priority scheduling, the WCRT depends on the alignment of arrival times

Task set A

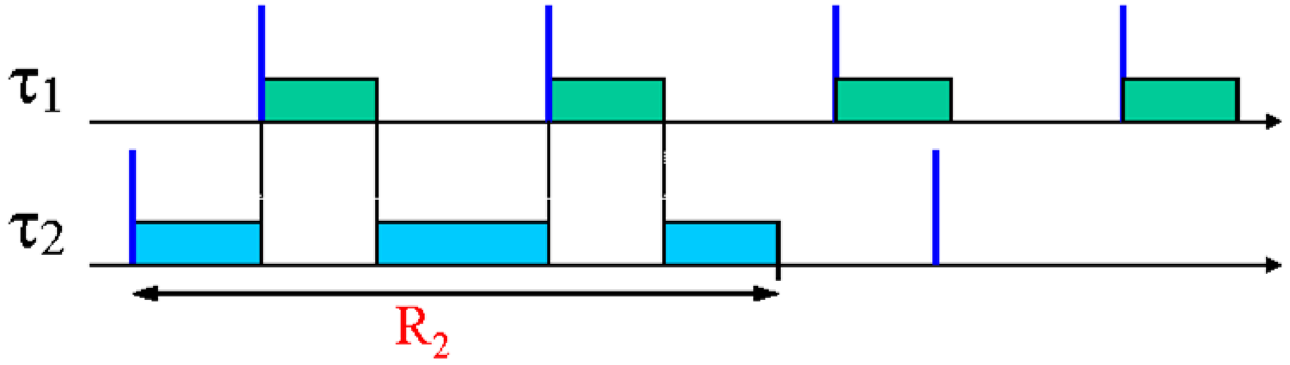
Task set B



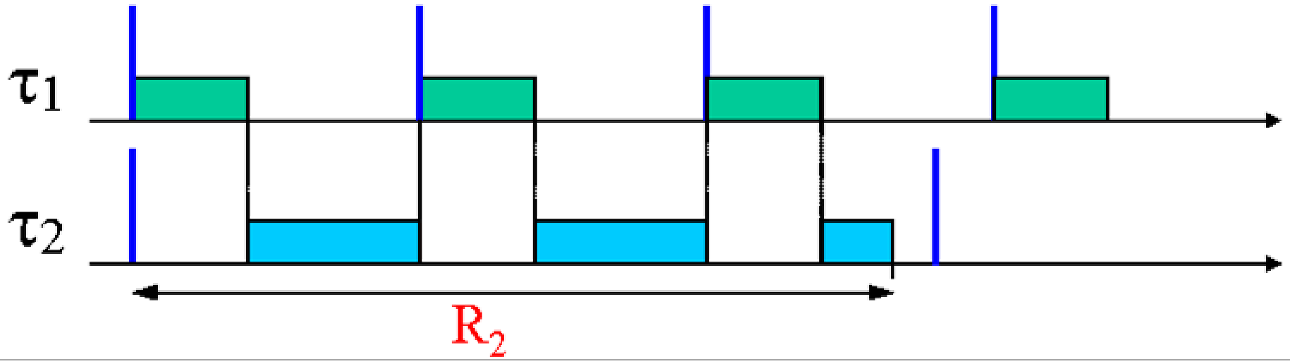
Critical instant

For any task τ_i, **the longest response time** occurs when it arrives together with **all higher-priority tasks.**

Task set A

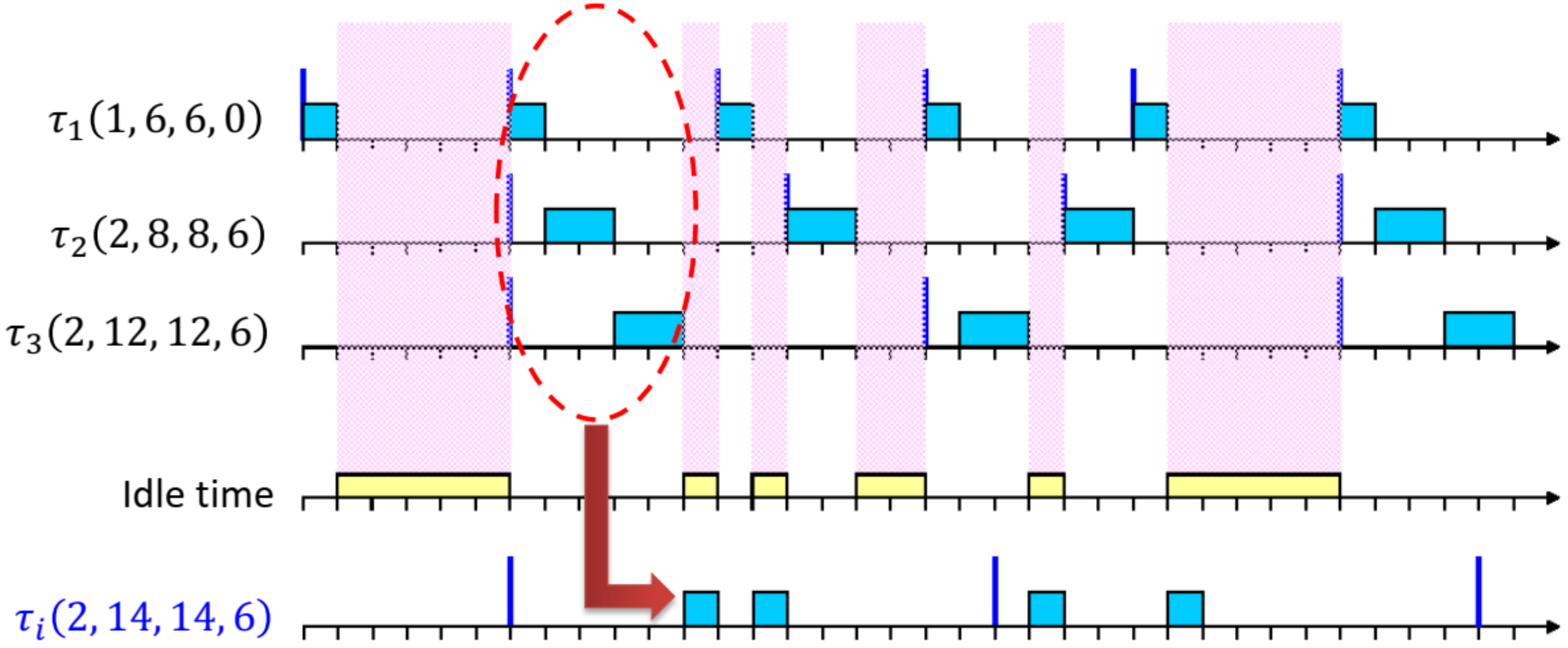


Task set B



Critical instant

For independent preemptive tasks under fixed priorities, the critical instant of τ_i occurs when it arrives together with all higher priority tasks.



$P_1 < P_2 < P_3 < \cdots < P_i$

Notations: $\tau_i(C_i, T_i, D_i, \phi_i)$

Response-time analysis (RTA) (Audsley '90)

$$R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k$$

The solution is based on fixed-point iterations:

Starting point:
$R_i^{(0)} = C_i$

Iterate until:
$R_i^{(n)} \leq R_i^{(n-1)}$

Usage: Use $R_i^{(0)}$ to calculate $R_i^{(1)}$, then use $R_i^{(1)}$ to obtain $R_i^{(2)}$, …., continue until $R_i^{(n)} = R_i^{(n-1)}$

Ceiling operator: $\lceil x \rceil = \min\{m \in \mathbb{Z} \mid m \geq x\}$
Example: $\lceil 2.4 \rceil = 3$, $\lceil 2.7 \rceil = 3$, $\lceil -2.4 \rceil = -2$

Exam hint!

Understanding the terms

Make sure that the execution of τ_i finishes

For all higher-priority tasks

Multiply that by the WCET of τ_k

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

Count the maximum number of jobs released by task τ_k in the interval $[0, R_i^{(n-1)})$

Exercise (5min)

- Exam question: find the WCRT of tasks τ_1 and τ_2 when they are scheduled by rate monotonic priorities

τ_i	C_i	T_i	D_i	U_i
τ_1	2	5	5	0.4
τ_2	4	10	10	0.4
τ_3	1	25	25	0.04

$U = 0.84$

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

Exercise (5min)

- Exam question: find the WCRT of tasks τ_1 and τ_2 when they are scheduled by rate monotonic priorities

τ_i	C_i	T_i	D_i	U_i
τ_1	2	5	5	0.4
τ_2	4	10	10	0.4
τ_3	1	25	25	0.04

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

$$R_1^{(0)} = 2$$

$$R_1^{(1)} = 2 + \sum_{k=1}^{1-1} \left\lceil \frac{R_1^{(0)}}{T_k} \right\rceil \cdot C_k = 2$$

We stop here since $R_1^{(n)} \leq R_1^{(n-1)}$ $\Rightarrow$ **WCRT of τ_1 is 2**

Exercise (5min)

- Exam question: find the WCRT of tasks τ_1 and τ_2 when they are scheduled by rate monotonic priorities

τ_i	C_i	T_i	D_i	U_i
τ_1	2	5	5	0.4
τ_2	4	10	10	0.4
τ_3	1	25	25	0.04

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

$$R_2^{(0)} = 4$$

$$R_2^{(1)} = 4 + \sum_{k=1}^{2-1} \left\lceil \frac{R_2^{(0)}}{T_k} \right\rceil \cdot C_k = 4 + \left\lceil \frac{4}{5} \right\rceil \cdot 2 = 6$$

$R_2^{(1)} \leq R_2^{(0)}$? **NO, so continue**

$$R_2^{(2)} = 4 + \sum_{k=1}^{2-1} \left\lceil \frac{R_2^{(1)}}{T_k} \right\rceil \cdot C_k = 4 + \left\lceil \frac{6}{5} \right\rceil \cdot 2 = 8$$

$R_2^{(2)} \leq R_2^{(1)}$? **NO, so continue**

$$R_2^{(3)} = 4 + \sum_{k=1}^{2-1} \left\lceil \frac{R_2^{(2)}}{T_k} \right\rceil \cdot C_k = 4 + \left\lceil \frac{8}{5} \right\rceil \cdot 2 = 8$$

$R_2^{(3)} \leq R_2^{(2)}$? **Yes, so stop**

$\Rightarrow$ **WCRT of τ_2 is 8**

Exercise (5min)

- Exam question: find the WCRT of tasks τ_1 and τ_2 when they are scheduled by rate monotonic priorities

τ_i	C_i	T_i	D_i	U_i
τ_1	2	5	5	0.4
τ_2	4	10	10	0.4
τ_3	1	25	25	0.04

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

$R_3^{(0)} = 1$

$$R_3^{(1)} = 1 + \sum_{k=1}^{3-1} \left\lceil \frac{R_3^{(0)}}{T_k} \right\rceil \cdot C_k = 1 + \left\lceil \frac{1}{5} \right\rceil \cdot 2 + \left\lceil \frac{1}{10} \right\rceil \cdot 4 = 7$$

$R_3^{(1)} \leq R_3^{(0)}$? **NO, so continue**

$$R_3^{(2)} = 1 + \sum_{k=1}^{2-1} \left\lceil \frac{R_3^{(1)}}{T_k} \right\rceil \cdot C_k = 1 + \left\lceil \frac{7}{5} \right\rceil \cdot 2 + \left\lceil \frac{7}{10} \right\rceil \cdot 4 = 9$$

$R_3^{(2)} \leq R_3^{(1)}$? **NO, so continue**

$$R_3^{(3)} = 1 + \sum_{k=1}^{2-1} \left\lceil \frac{R_3^{(2)}}{T_k} \right\rceil \cdot C_k = 1 + \left\lceil \frac{9}{5} \right\rceil \cdot 2 + \left\lceil \frac{9}{10} \right\rceil \cdot 4 = 9$$

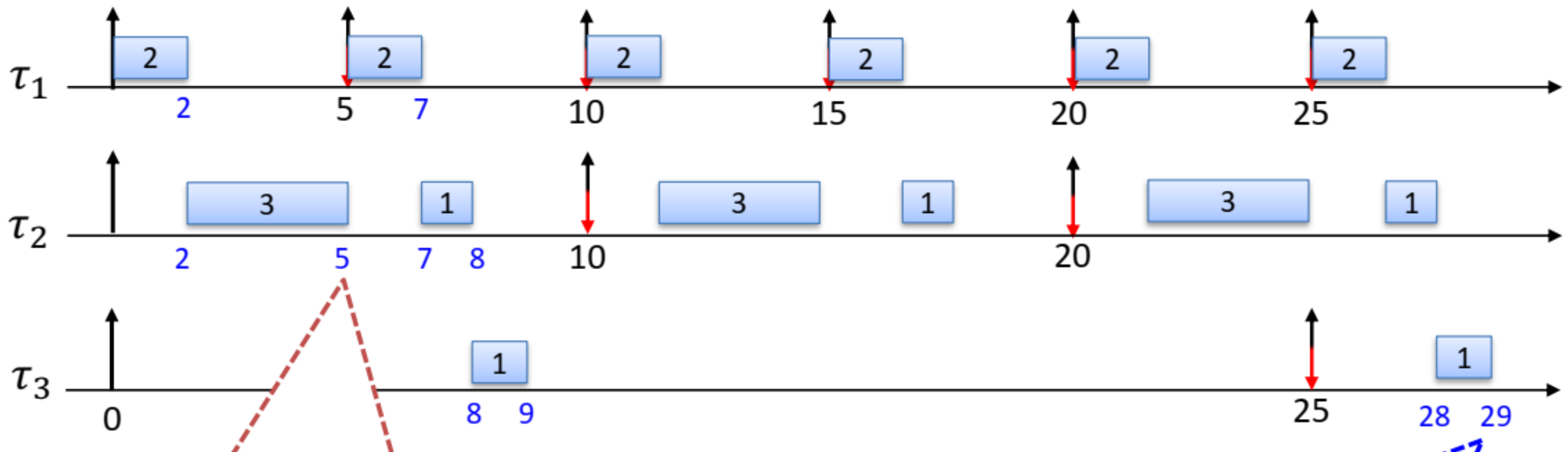
$R_3^{(3)} \leq R_3^{(2)}$? **Yes, so stop**

➡ **WCRT of τ_3 is 9**

Visualizing the WCRT

τ_i	C_i	T_i	D_i	U_i	R_i
τ_1	2	5	5	0.4	2
τ_2	4	10	10	0.4	8
τ_3	1	25	25	0.04	9

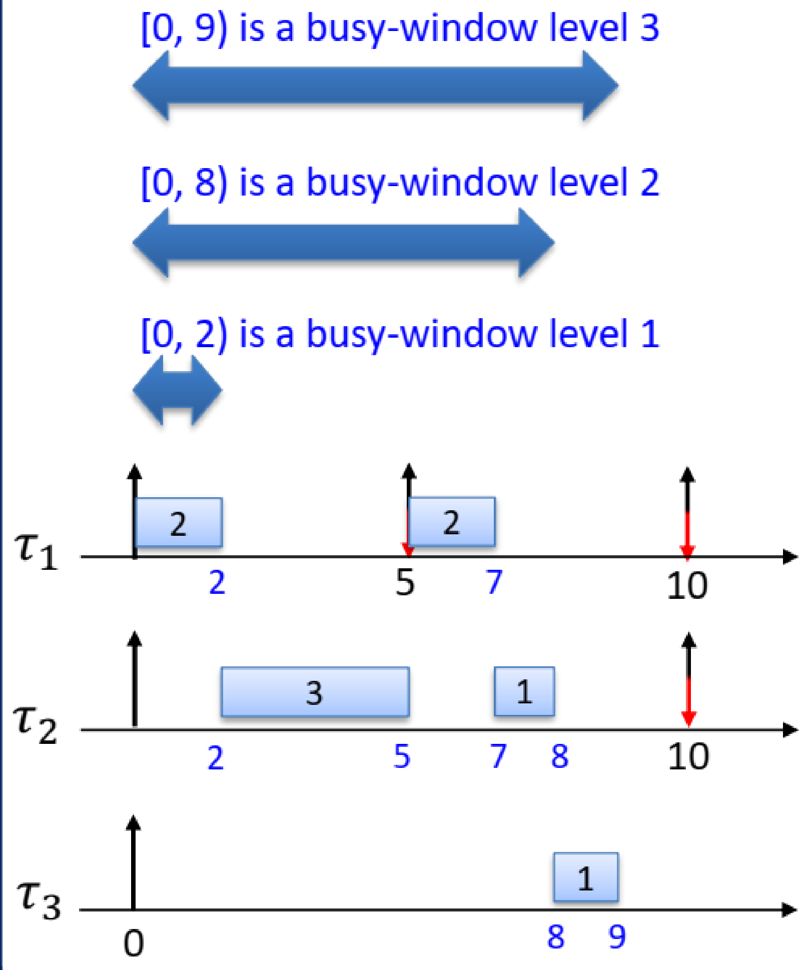
$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$



An iteration is needed whenever R_i^n becomes larger than the arrival time of a new higher-priority job that has <u>not been considered</u> in R_i^{n-1}

What is $R_{3,2}$?
Why is $R_{3,2}$ smaller than R_3?

Busy-window level


[0, 9) is a busy-window level 3
[0, 8) is a busy-window level 2
[0, 2) is a busy-window level 1
τ_1
2
2
2
5
7
10
τ_2
3
1
2
5
7
8
10
τ_3
1
0
8
9

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

A **busy-window level *i*** is a window of time during which the processor is busy by executing tasks with priority P_i or higher.

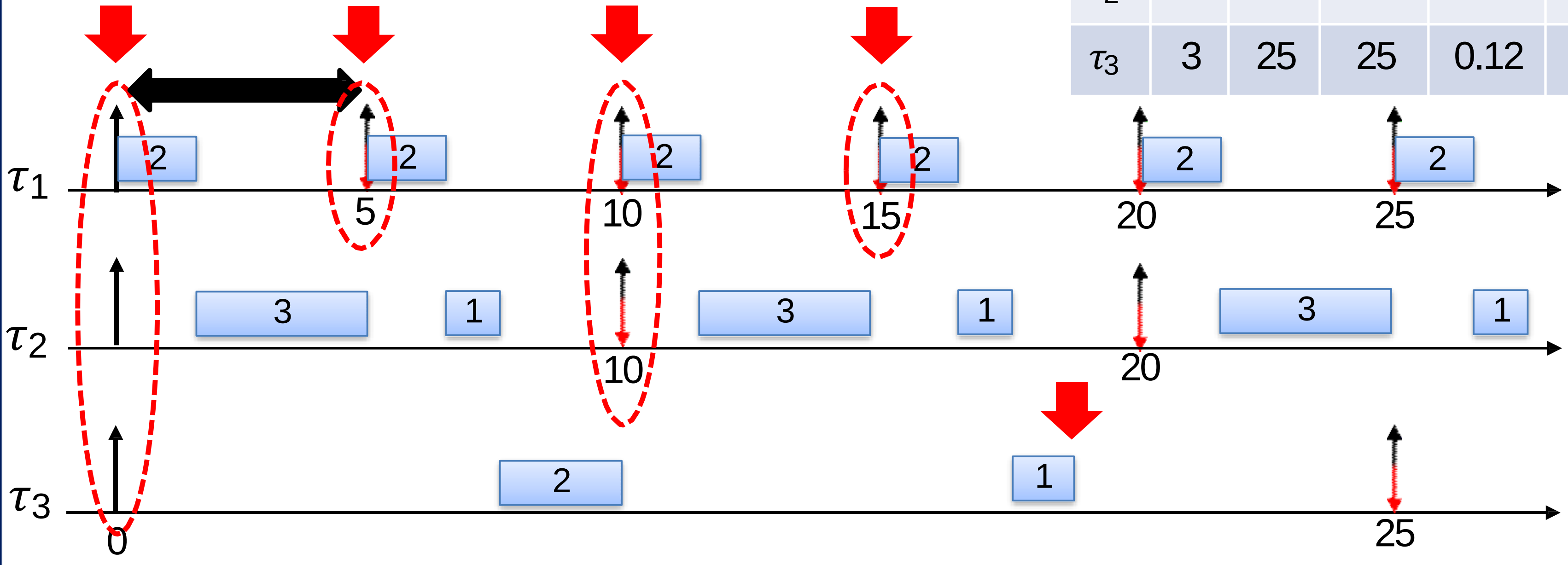
Computational complexity of RTA

How many iterations are needed to calculate R_3?

$R_3^{(0)} = 3$ $R_3^{(1)} = 9$ $R_3^{(2)} = 11$

$R_3^{(3)} = 17$ $R_3^{(4)} = 19$ $R_3^{(5)} = 19$

τ_i	C_i	T_i	D_i	U_i	R_i
τ_1	2	5	5	0.4	2
τ_2	4	10	10	0.4	8
τ_3	3	25	25	0.12	19



The stop condition basically ensures that the workload that must be finished will be finished before a new task arrives

Computational complexity of RTA

- Calculating the WCRT of a task using RTA is a **weakly NP-Hard** problem [Eisenbrand2008]

- **It has a pseudo-polynomial time computational complexity**

What does it mean?

Example: $O(w \cdot n)$
n is the number of tasks
w is the ratio of the largest to the smallest period

$w = 2^n$
$\Rightarrow O(w \cdot n) = O(n \cdot 2^n)$
$\rightarrow$ **exponential**

Computational complexity of RTA

- Calculating the WCRT of a task using RTA is a **weakly NP-Hard** problem [Eisenbrand2008]

- **It has a pseudo-polynomial time computational complexity**

  - The complexity not only depends on the number of tasks n but also depends on the periods and execution times
  - Namely, for some task sets it is **very fast**, and for some other task sets with the same number of tasks, it is **very very slow**

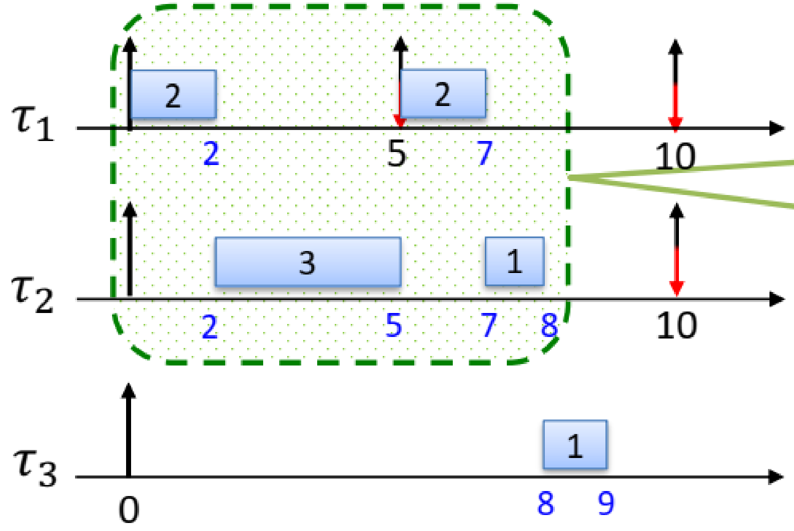
Don’t worry.
In practice, RTA is really fast for periodic tasks :D

How to make the RTA faster?

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

A better starting point!

What do you suggest?



Instead of starting from

$$R_i^{(0)} = C_i$$

Start from

$$R_i^{(0)} = R_{i-1} + C_i$$

Why is this okay?

Because anyway τ_i cannot start its execution until the busy-window level $i-1$ has finished!

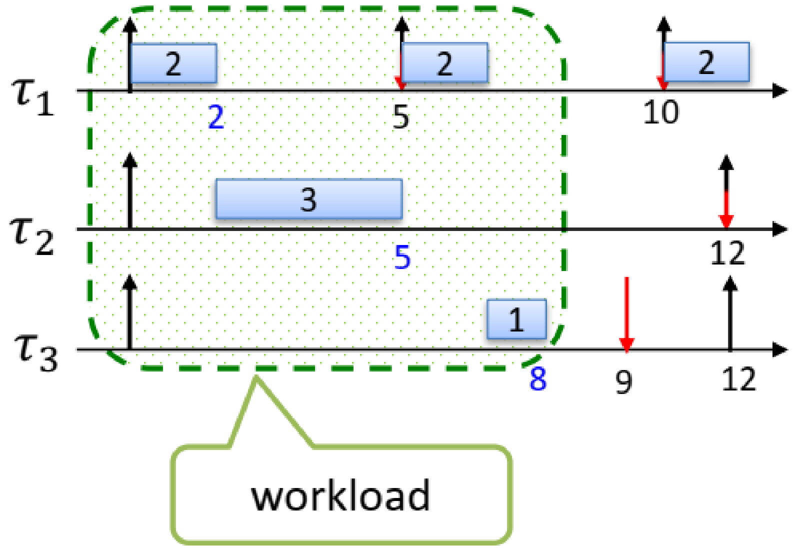
Using RTA to build fast sufficient tests

Park's test

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

Idea:
Instead of searching for the exact WCRT, just check if the workload that must be completed before the deadline is smaller than the deadline of the task

How?



M. Park and H. Park. An efficient test method for rate monotonic schedulability. IEEE Transactions on Computers, 63(5):1309–1315, 2014

Using RTA to build fast sufficient tests

Park's test

$$\begin{cases} R_i^{(n)} = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i^{(n-1)}}{T_k} \right\rceil \cdot C_k \\ R_i^{(0)} = C_i \\ \text{Continue if } R_i^{(n)} > R_i^{(n-1)} \end{cases}$$

Idea:
Instead of searching for the exact WCRT, just check if the workload that must be completed before the deadline is smaller than the deadline of the task

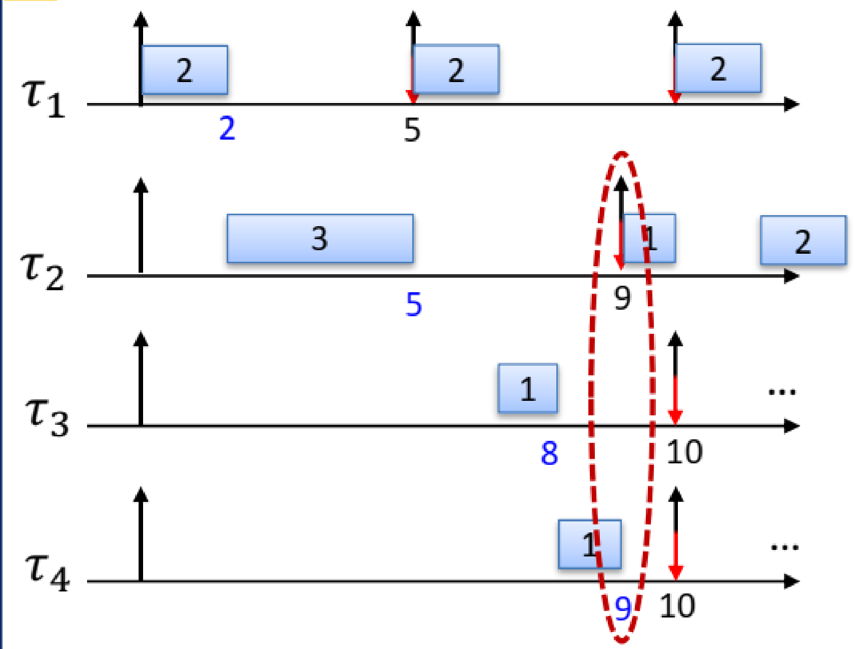
$$\overbrace{C_i + \sum_{k=1}^{i-1} \left\lceil \frac{D_i}{T_k} \right\rceil \cdot C_k}^{\text{workload that must be completed before the deadline}} \le D_i$$

What is the computational complexity per task?

$$O(n)$$

Why the Park's test is just a sufficient test?

Park's test is just a sufficient test



Is this task set schedulable by RM?

Yes! The critical instant is schedulable.
=> The WCRT of each task is smaller than its deadline

What is the WCRT of τ_3 and τ_4?

8 and 9, respectively.

What does Park's test say about τ_4?

It says τ_4 will miss its deadline

Why did it happen?

When a higher-priority task is released after the WCRT of the task under study, Park's test considers that task within the workload that must be finished by the deadline! Namely, it upper-approximate the workload!

$$C_4 + \sum_{k=1}^{4-1} \left\lceil \frac{D_4}{T_k} \right\rceil \cdot C_k \le D_4 \quad \Rightarrow 1 + \left(\left\lceil \frac{10}{5} \right\rceil \cdot 2 \right) + \left(\left\lceil \frac{10}{9} \right\rceil \cdot 3 \right) + \left(\left\lceil \frac{10}{10} \right\rceil \cdot 1 \right)$$

$$= 1 + 4 + 6 + 1 = 12 \not\le 10$$

A test for harmonic tasks

Han et al. [1997] have proven that **rate monotonic** is **optimal** if periods are **harmonic**
(each period divides smaller ones).

Hence, if periods are **harmonic**, the following test is both necessary and sufficient for RM schedulability

$$U \leq 1$$

C.-C. Han and H.-Y. Tyan. A Better Polynomial-time Schedulability Test for Real-time Fixed-priority Scheduling Algorithms. In IEEE Real-Time Systems Symposium (RTSS), pages 36–45, 1997.