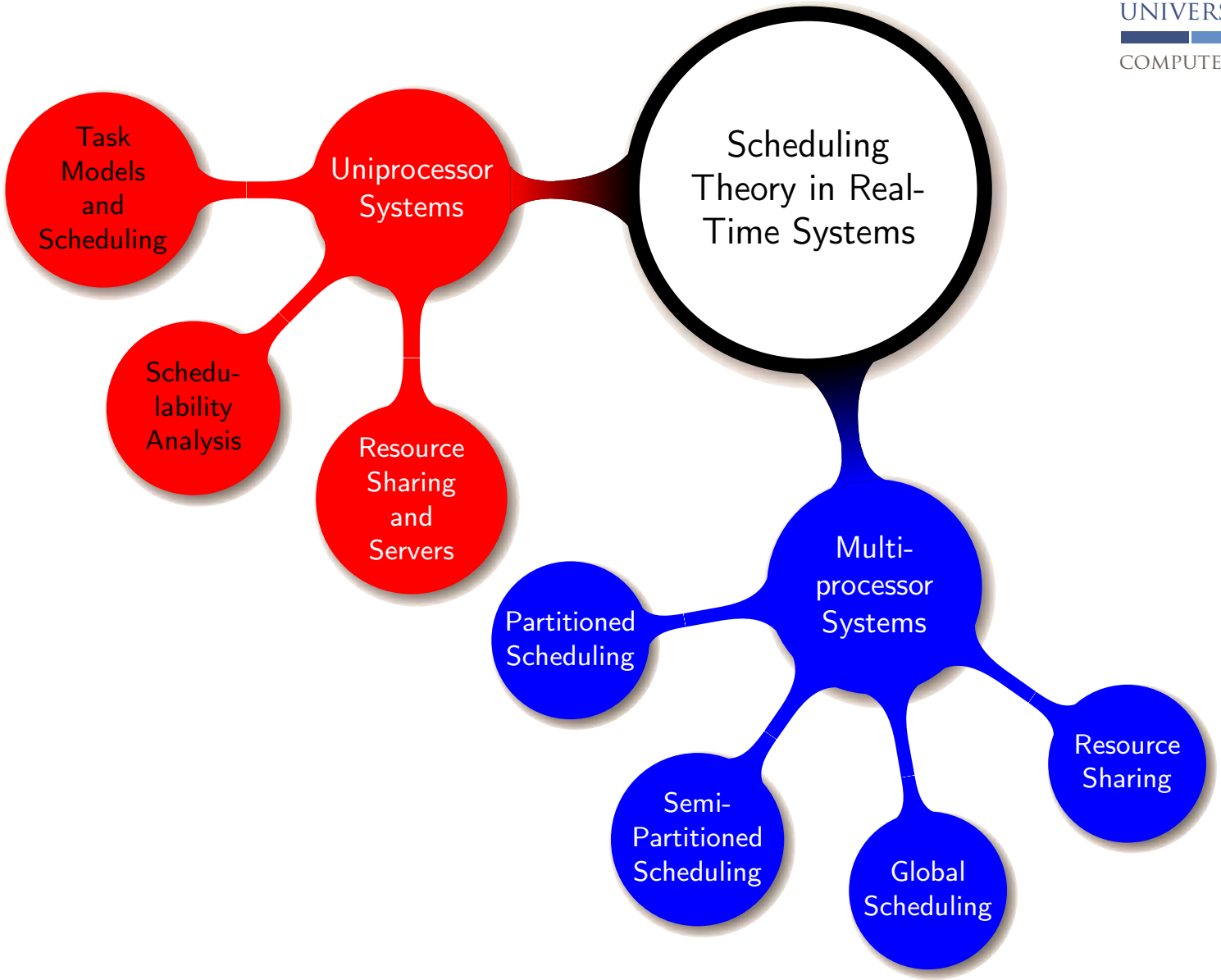
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Schedulability of Periodic and Sporadic Task Sets on Uniprocessor Systems

Jan Reineke

Saarland University

July 16, 2015


Scheduling Theory in Real-Time Systems
Uniprocessor Systems
Task Models and Scheduling
Schedu-lability Analysis
Resource Sharing and Servers
Multi-processor Systems
Partitioned Scheduling
Semi-Partitioned Scheduling
Global Scheduling
Resource Sharing

Recurrent Task Models (revisited)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume ϕ_i is 0.
- Sporadic Task τ_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i, T_i, D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time.

Recurrent Task Models (revisited)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume ϕ_i is 0.
- Sporadic Task τ_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i, T_i, D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time.

Relative Deadline vs Period (revisited)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

For a task set, we say that the task set is with

- *implicit deadline* when the relative deadline D_i is equal to the period T_i, i.e., $D_i = T_i$, for every task τ_i,
- *constrained deadline* when the relative deadline D_i is no more than the period T_i, i.e., $D_i \leq T_i$, for every task τ_i, or
- *arbitrary deadline* when the relative deadline D_i could be larger than the period T_i for some task τ_i.

Some Definitions for Sporadic/Periodic Tasks

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- The jobs of task τ_i are denoted $J_{i,1}, J_{i,2}, \ldots\ldots$
- Periodic Tasks:
  - Synchronous system: Each task has a phase of 0.
  - Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of T_i.
- Task utilization of task τ_i: $u_i := \frac{C_i}{T_i}$.
- System (total) utilization: $U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} u_i$.

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Static-Priority Scheduling

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Different jobs of a task are assigned the same priority.
  - π_i is the priority of task τ_i.
  - HP_i is the subset of tasks with higher priority than τ_i.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, i.e., τ_i has higher priority than τ_k if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?

Static-Priority Scheduling

SAARLAND UNIVERSITY
COMPUTER SCIENCE

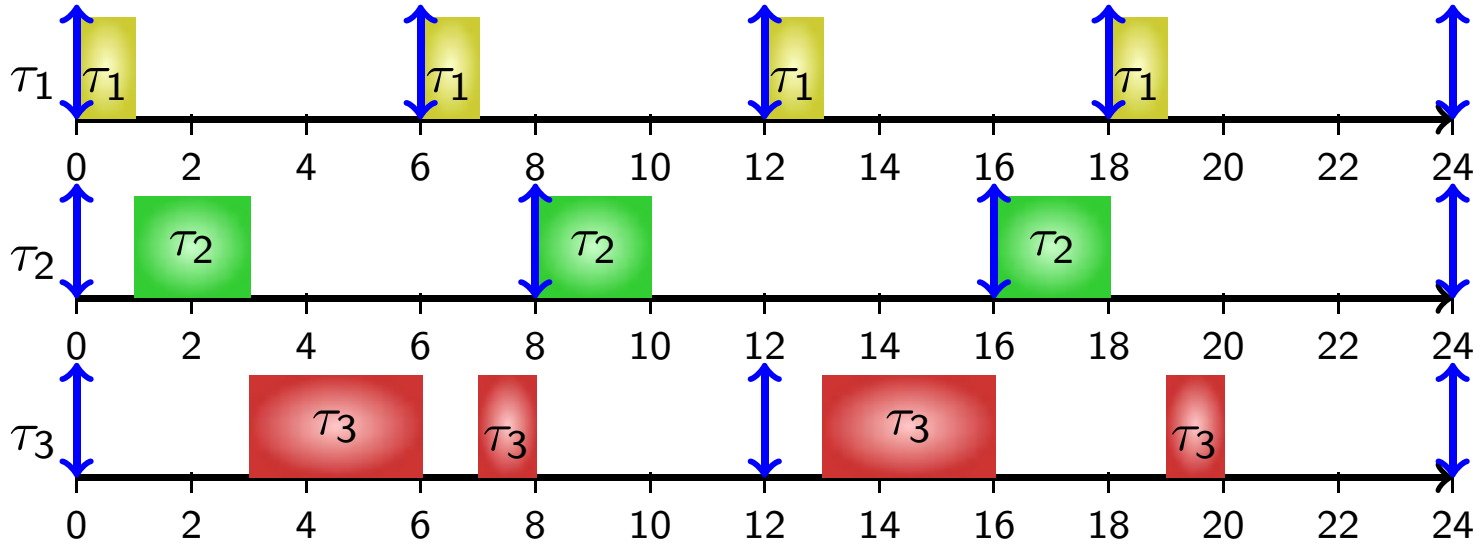
- Different jobs of a task are assigned the same priority.
  - π_i is the priority of task τ_i.
  - HP_i is the subset of tasks with higher priority than τ_i.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, i.e., τ_i has higher priority than τ_k if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?

Rate-Monotonic (RM) Scheduling (Liu and Layland, 1973)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

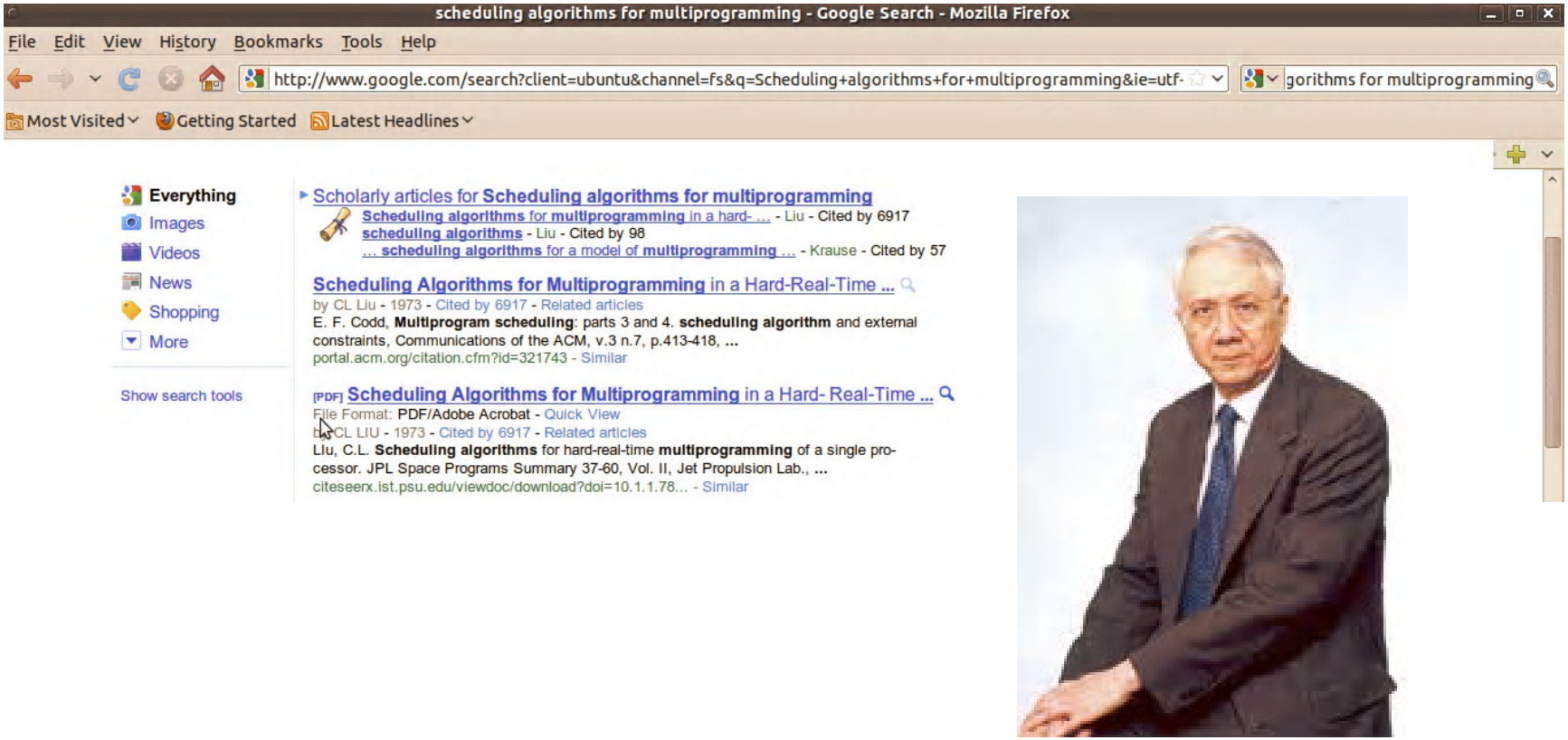
Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.

Example Schedule: $\tau_1 = (1, 6, 6)$, $\tau_2 = (2, 8, 8)$, $\tau_3 = (4, 12, 12)$.
$[(C_i, T_i, D_i)]$



Liu and Layland (Journal of the ACM, 1973)

SAARLAND UNIVERSITY
COMPUTER SCIENCE



Liu and Layland (Journal of the ACM, 1973)

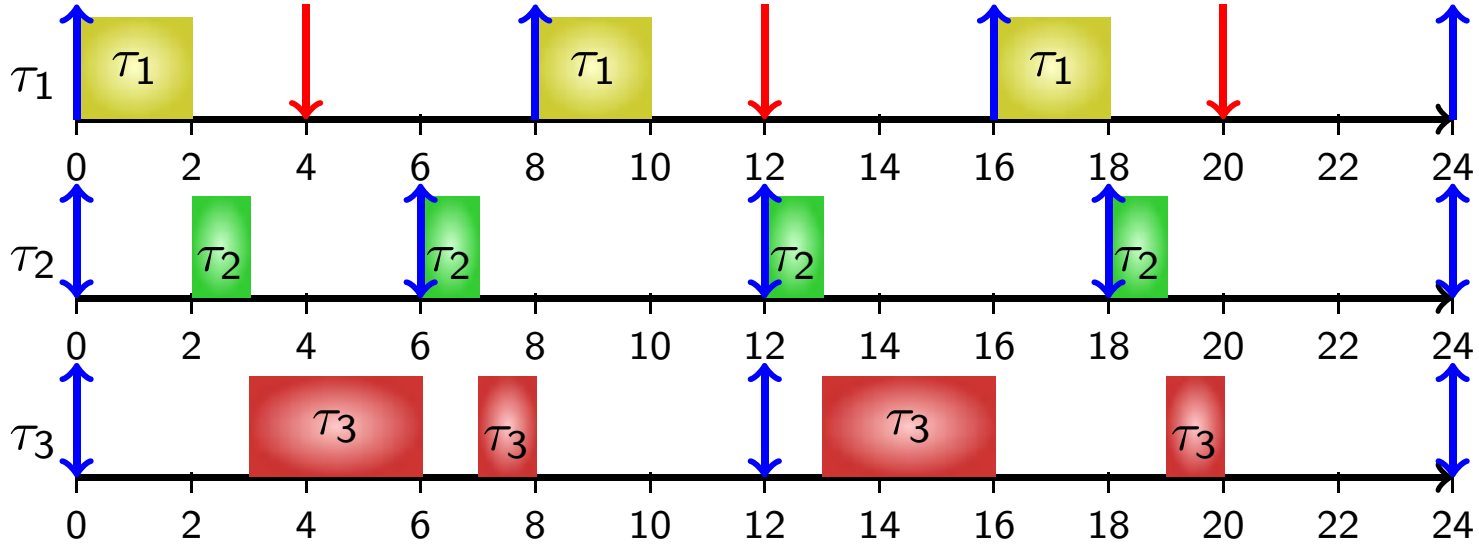
SAARLAND UNIVERSITY

Deadline-Monotonic (DM) Scheduling (Leung and Whitehead)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.

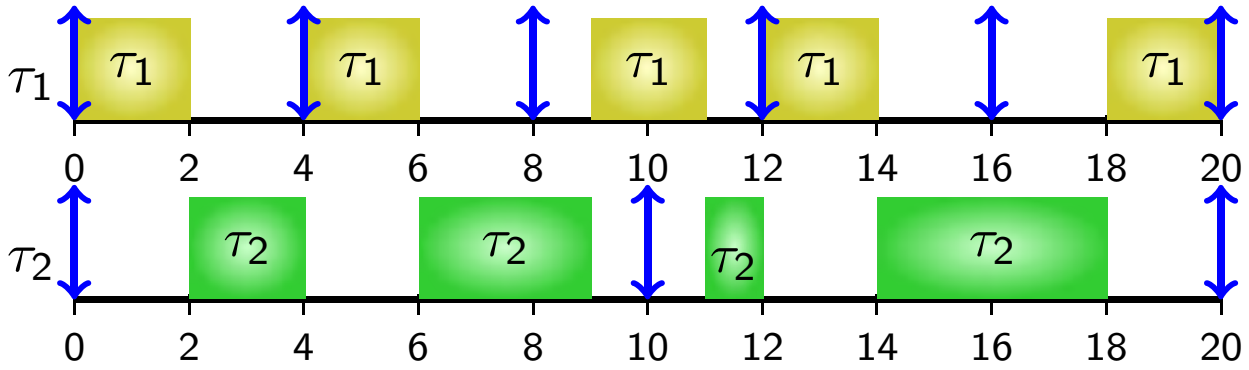
Example Schedule: $\tau_1 = (2, 8, 4)$, $\tau_2 = (1, 6, 6)$, $\tau_3 = (4, 12, 12)$. $[(C_i, T_i, D_i)]$



Optimality (or not) of RM and DM

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



The above system is schedulable.

No static-priority scheme is optimal for scheduling periodic tasks: A deadline will be missed, regardless of how we choose to (statically) prioritize τ_1 and τ_2.

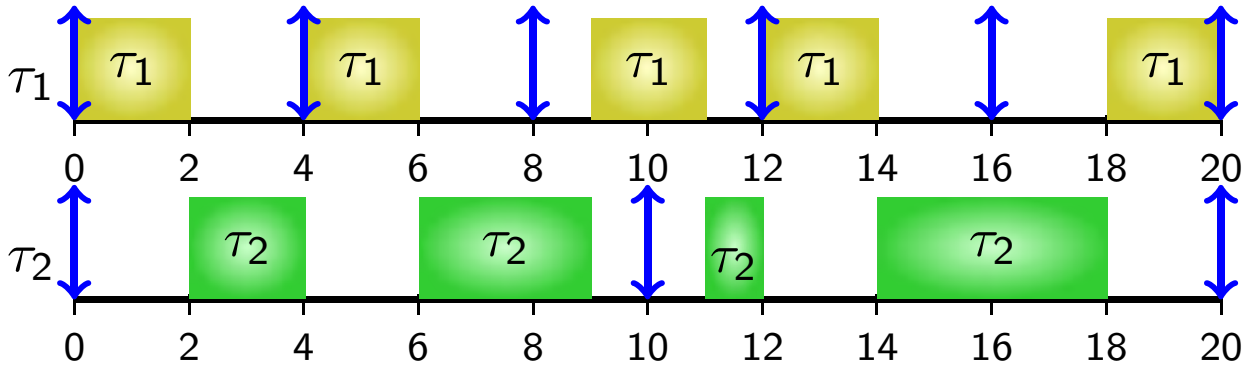
Corollary

Neither RM nor DM is optimal.

Optimality (or not) of RM and DM

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



The above system is schedulable.

No static-priority scheme is optimal for scheduling periodic tasks: A deadline will be missed, regardless of how we choose to (statically) prioritize τ_1 and τ_2.

Corollary

Neither RM nor DM is optimal.

Aside: Critical Instants

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Definition

A critical instant of a task τ_i is a time instant such that:

1. the job of τ_i released at this instant has the maximum response time of all jobs in τ_i, if the response time of every job of τ_i is at most D_i, the relative deadline of τ_i, and
2. the response time of the job released at this instant is greater than D_i if the response time of some job in τ_i exceeds D_i.

Informally, a critical instant of τ_i represents a worst-case scenario from τ_i's standpoint.

Critical Instants in Static-Priority Systems

SAARLAND UNIVERSITY
COMPUTER SCIENCE

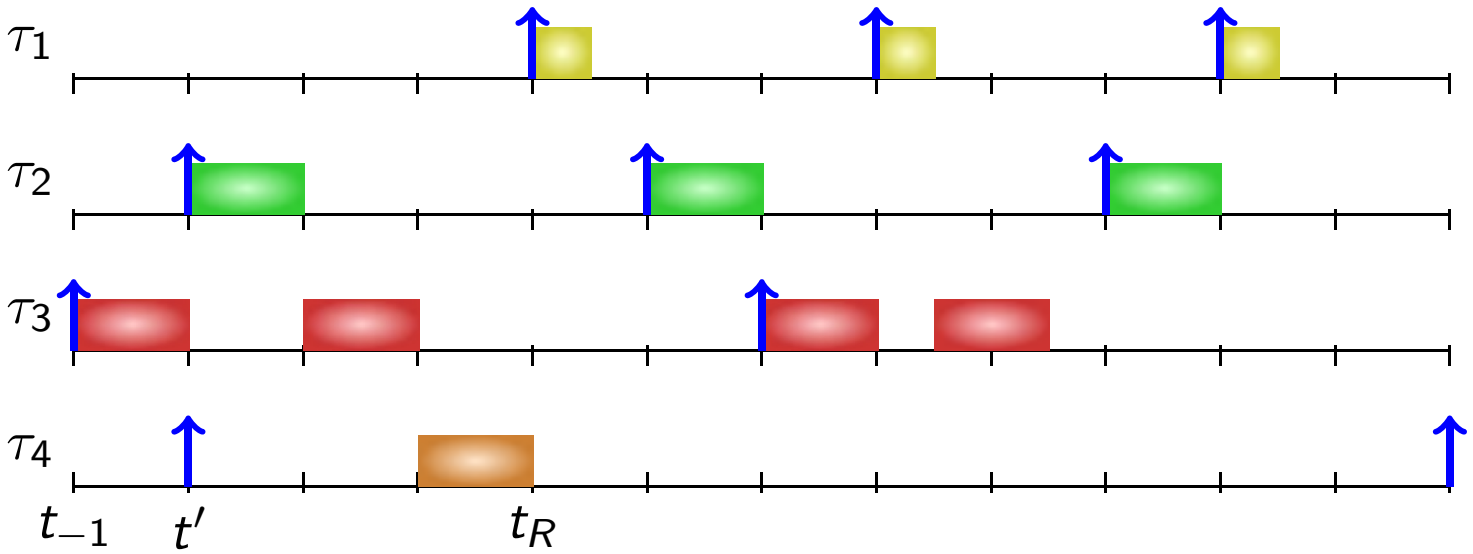
Theorem

[Liu and Layland, JACM 1973] A critical instant of task τ_i for a set of independent, preemptable periodic tasks with relative deadlines equal to their respective periods is to release the first jobs of all the higher-priority tasks at the same time.

We are not saying that $\tau_1, \ldots, \tau_i$ will all necessarily release their first jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for task τ_i.

Critical Instants: Informal Proof

SAARLAND UNIVERSITY
COMPUTER SCIENCE

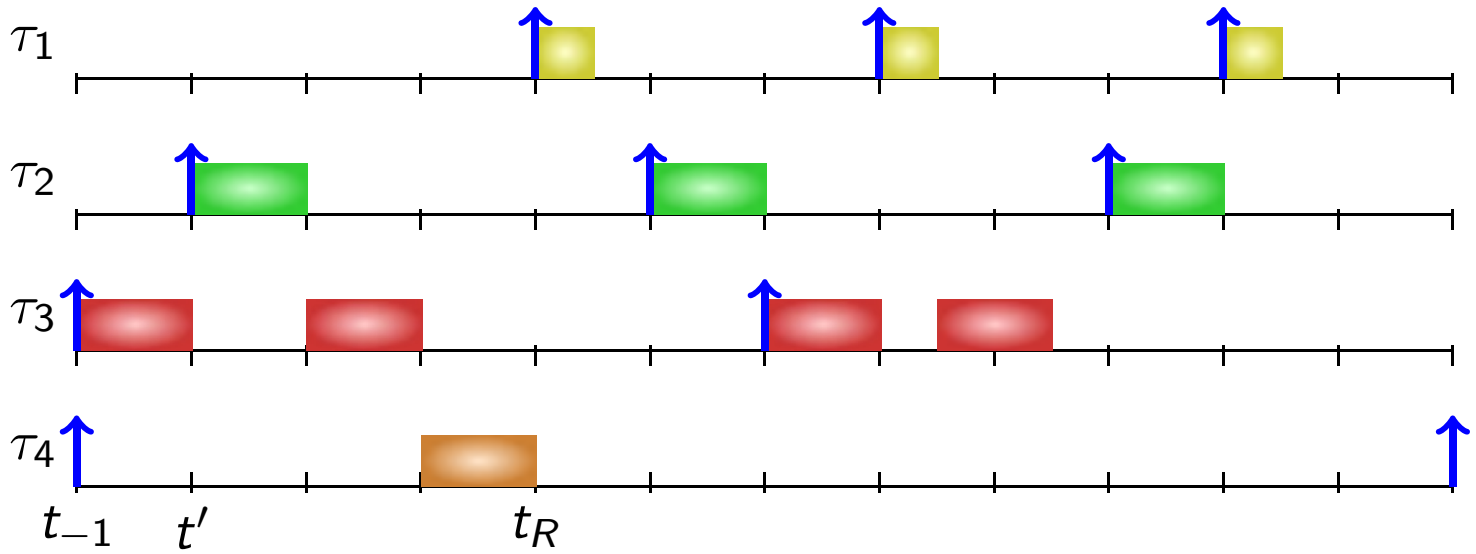


We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Consider a job of τ_i, released at time t', with completion time t_R.
- Let t_{-1} be the latest *idle instant* for $\tau_1, \ldots, \tau_{i-1}$ at or before t_R.
- Let J be τ_i's job released at t'.

Critical Instants: Informal Proof

SAARLAND UNIVERSITY
COMPUTER SCIENCE

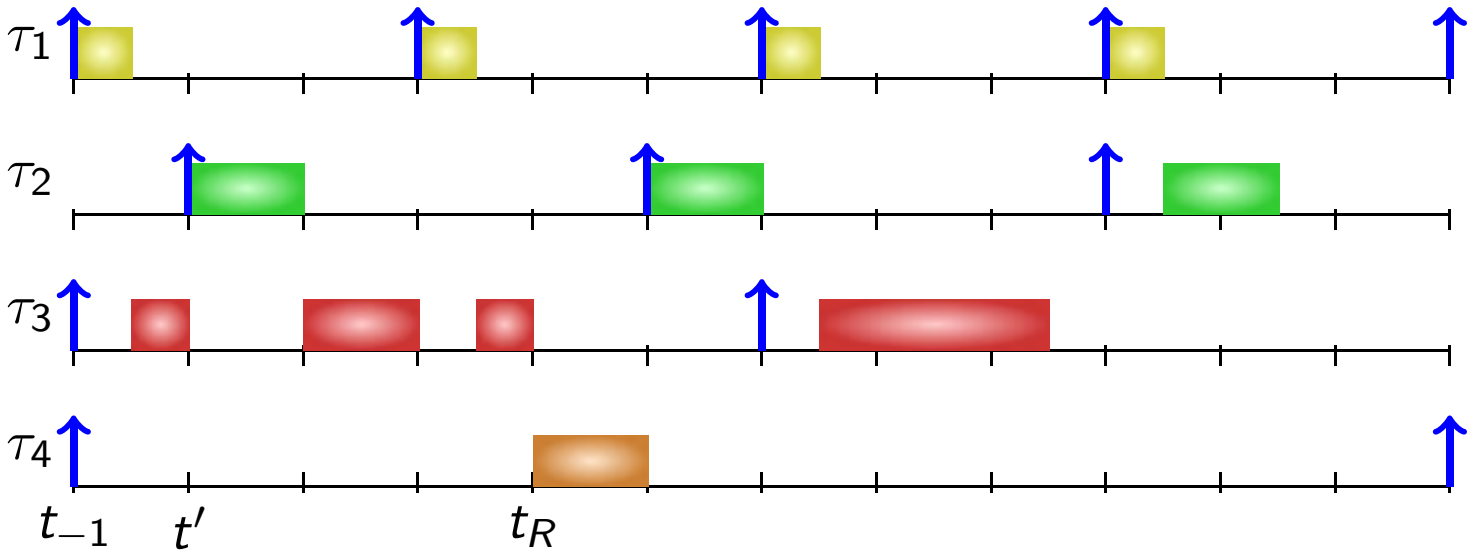


We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Moving J from t' to t_{-1} does not decrease the completion time of J.

Critical Instants: Informal Proof

SAARLAND UNIVERSITY
COMPUTER SCIENCE

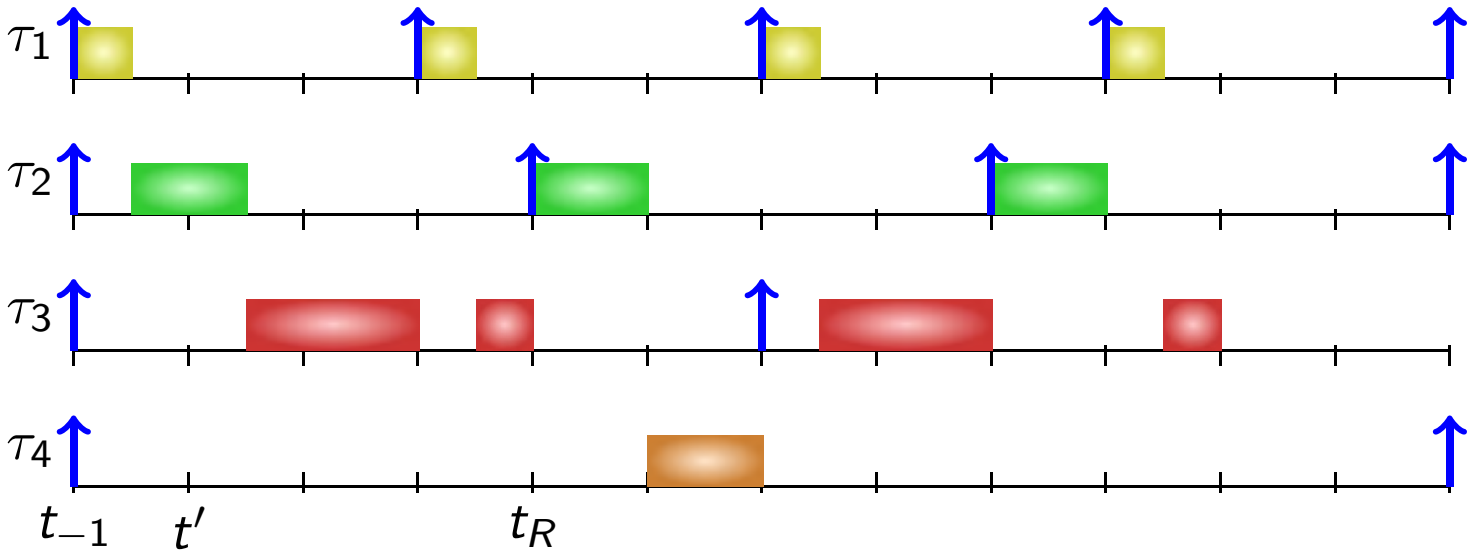


We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Releasing τ_1 at t_{-1} does not decrease the completion time of J.

Critical Instants: Informal Proof

SAARLAND UNIVERSITY
COMPUTER SCIENCE



We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Releasing τ_2 at t_{-1} does not decrease the completion time of J.
- Repeating the above movement and proves the criticality of the critical instant

Harmonic Real-Time Systems

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Definition

A system of periodic tasks *harmonic* (also: *simply periodic*) if for every pair of tasks τ_i and τ_k in the system where $T_i < T_k$, T_k is an integer multiple of T_i.

For example: Periods are $2, 6, 12, 24$.

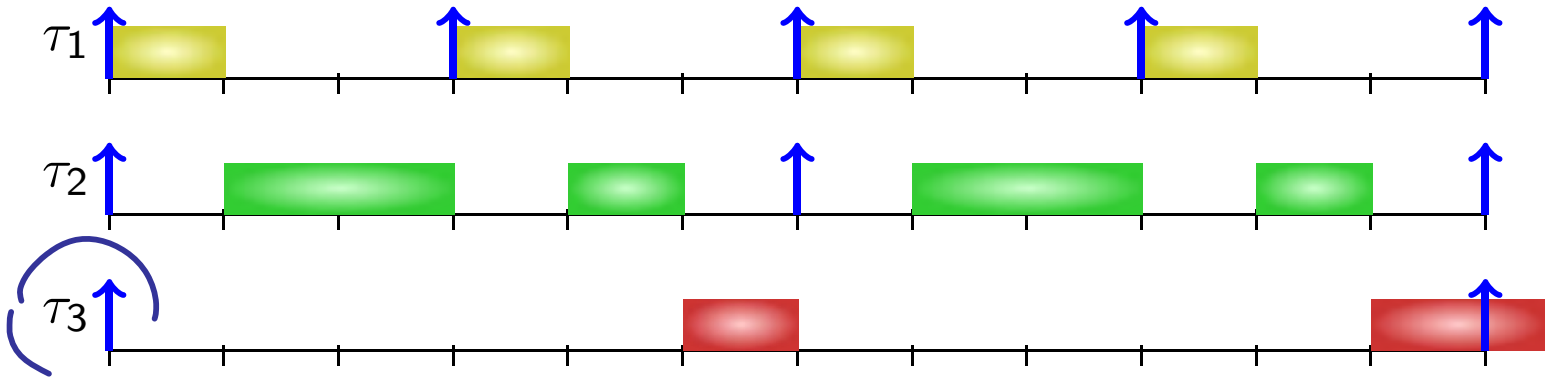
Theorem

[Kuo and Mok]: A system $\mathcal{T}$ of harmonic, independent, preemptable, and implicit-deadline tasks is schedulable on one processor according to the RM algorithm if and only if its total utilization $U = \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}$ is less than or equal to one.

Proof for Harmonic Systems ("if" part)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

The case for the "only-if" part is similar and left as an exercise.



Suppose for a contradiction that $\mathcal{T}$ is not schedulable and that τ_i misses its deadline.

- The response time of τ_i is larger than D_i.
- By critical instants, releasing all the tasks $\tau_1, \tau_2, \ldots, \tau_i$ at time 0 will lead to a response time of τ_i larger than D_i.

Proof for Harmonic Systems (cont.)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

As the schedule is work-conserving, we know that from time 0 to time D_i, the whole system is executing jobs. Therefore,

$$
\begin{aligned}
D_i &< \text{the workload released in time interval } [0, D_i) \\
&= \sum_{j=1}^{i} C_j \cdot (\text{ the number of job releases of } \tau_j \text{ in time interval } [0, D_i)) \\
&= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{D_i}{T_j} \right\rceil =^* \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},
\end{aligned}
$$

where $=^*$ is because *$D_i = T_i$ is an integer multiple of T_j when $j \leq i$*.

By canceling D_i, we reach the contradiction by having

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j} \leq 1.$$

Proof for Harmonic Systems (cont.)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

As the schedule is work-conserving, we know that from time 0 to time D_i, the whole system is executing jobs. Therefore,

$$
\begin{aligned}
D_i &< \text{the workload released in time interval } [0, D_i) \\
&= \sum_{j=1}^{i} C_j \cdot (\text{ the number of job releases of } \tau_j \text{ in time interval } [0, D_i)) \\
&= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{D_i}{T_j} \right\rceil =^* \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},
\end{aligned}
$$

where $=^*$ is because *$D_i = T_i$ is an integer multiple of T_j when $j \leq i$*.

By canceling D_i, we reach the contradiction by having

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j} \leq 1.$$

Optimality Among Static-Priority Algorithms

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

A system $\mathcal{T}$ of independent, preemptable, synchronous periodic tasks that have relative deadlines equal to their respective periods can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.

We will only discuss systems with 2 tasks, and the generalization is left as an exercise.

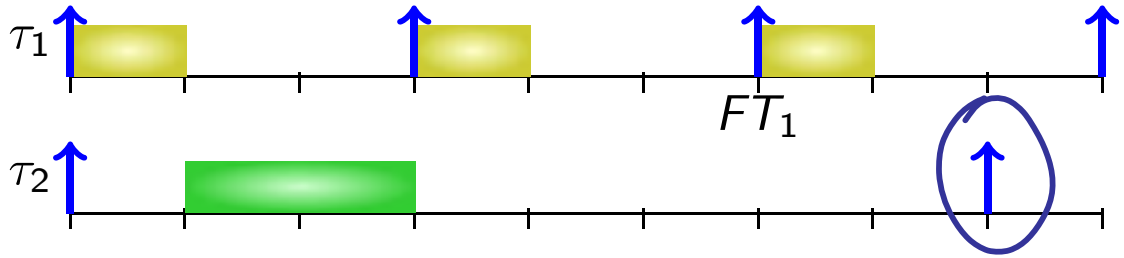
- Suppose that $T_1 = D_1 < D_2 = T_2$ and τ_2 is with higher priority.
- We would like to swap the priorities of τ_1 and τ_2.
- Without loss of generality, the response time of τ_1 after priority swapping is always equal to (or no more than) C_1.
- By the critical instant theorem, we only need to check response time of the first job of τ_2 during a critical instant.
- Assuming that non-RM priority ordering is schedulable, the critical instant theorem also implies that $C_1 + C_2 \leq T_1$.

Optimality Among Static-Priority Algorithms (cont.)

After swapping (τ_1 has higher priority), there are two cases:

Case 1

There is sufficient time to complete all F jobs of τ_1 before the second job arrival of τ_2, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$. In other words, $C_1 + F \cdot T_1 < T_2$.



To be schedulable

$(F+1)C_1 + C_2 \leq T_2$ must hold.

By $C_1 + C_2 \leq T_1$, we have

$$F(C_1 + C_2) \leq F \cdot T_1$$

$$\overset{F \geq 1}{\Rightarrow} FC_1 + C_2 \leq F \cdot T_1$$

$$(F+1)C_1 + C_2 \leq F \cdot T_1 + C_1 < T_2$$

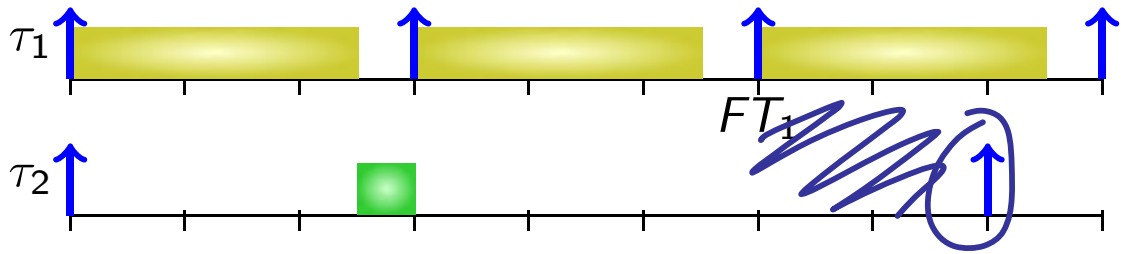
$$\Rightarrow (F+1)C_1 + C_2 < T_2$$

Optimality Among Static-Priority Algorithms (cont.)

After swapping (τ_1 has higher priority), there are two cases:

Case 2

The F-th job of τ_1 does not complete before the arrival of the second job of τ_2.
In other words, $C_1 + F \cdot T_1 \geq T_2$, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$.



By $C_1 + C_2 \leq T_1$, we have

$$F(C_1 + C_2) \leq F \cdot T_1$$

$$\overset{F \geq 1}{\Rightarrow} FC_1 + C_2 \leq F \cdot T_1$$

To be schedulable

$FC_1 + C_2 \leq FT_1$ must hold.

Remarks on the Optimality

SAARLAND UNIVERSITY
COMPUTER SCIENCE

We have shown that if any two-task system with implicit deadlines ($D_i = T_i$) is schedulable according to arbitrary fixed-priority assignment, then it is also schedulable according to RM.

Exercise: Complete proof by extending argument to n periodic tasks.

Note: When $D_i \leq T_i$ for all tasks, DM (Deadline Monotonic) can be shown to be an optimal static-priority algorithm using similar argument. Proof left as an exercise.

Utilization-Based Schedulability Test

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Task utilization:

$$u_i := \frac{C_i}{T_i}.$$

- System (total) utilization:

$$U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i}.$$

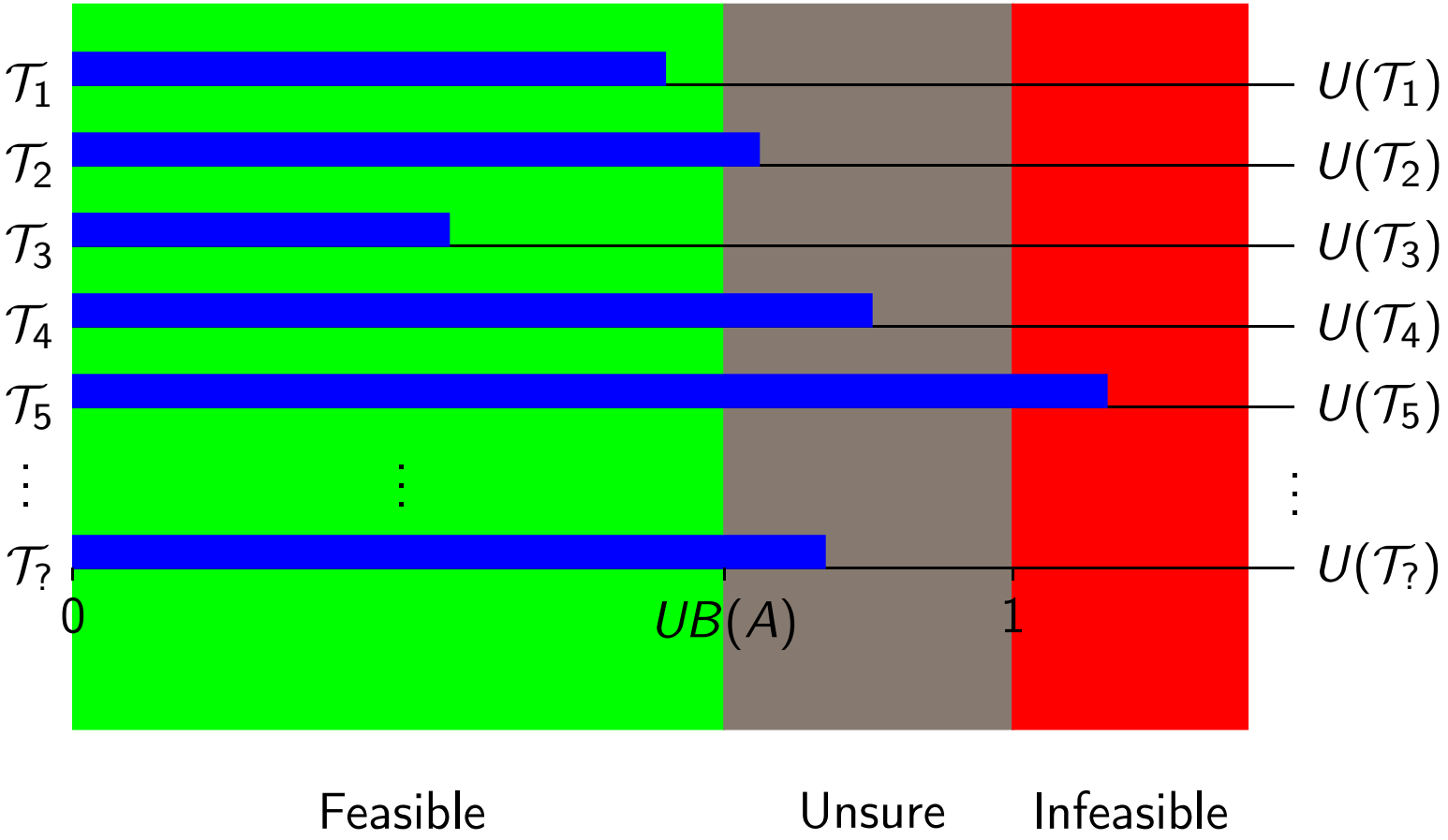
A task system $\mathcal{T}$ fully utilizes the processor under scheduling algorithm A if any increase in execution time (of any task) causes A to miss a deadline.

$UB(A)$ is the utilization bound for algorithm A:

$$UB(A) := \bigsqcup \{U \in \mathbb{R} \mid \forall \mathcal{T}.U(\mathcal{T}) \leq U \Rightarrow \mathcal{T} \textit{ is schedulable under } A\}$$
$$= \bigsqcap \{U(\mathcal{T}) \mid \mathcal{T} \textit{ fully utilizes the processor under } A\}$$

What is $UB(A)$ useful for?

SAARLAND UNIVERSITY
COMPUTER SCIENCE



Liu and Layland Bound

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

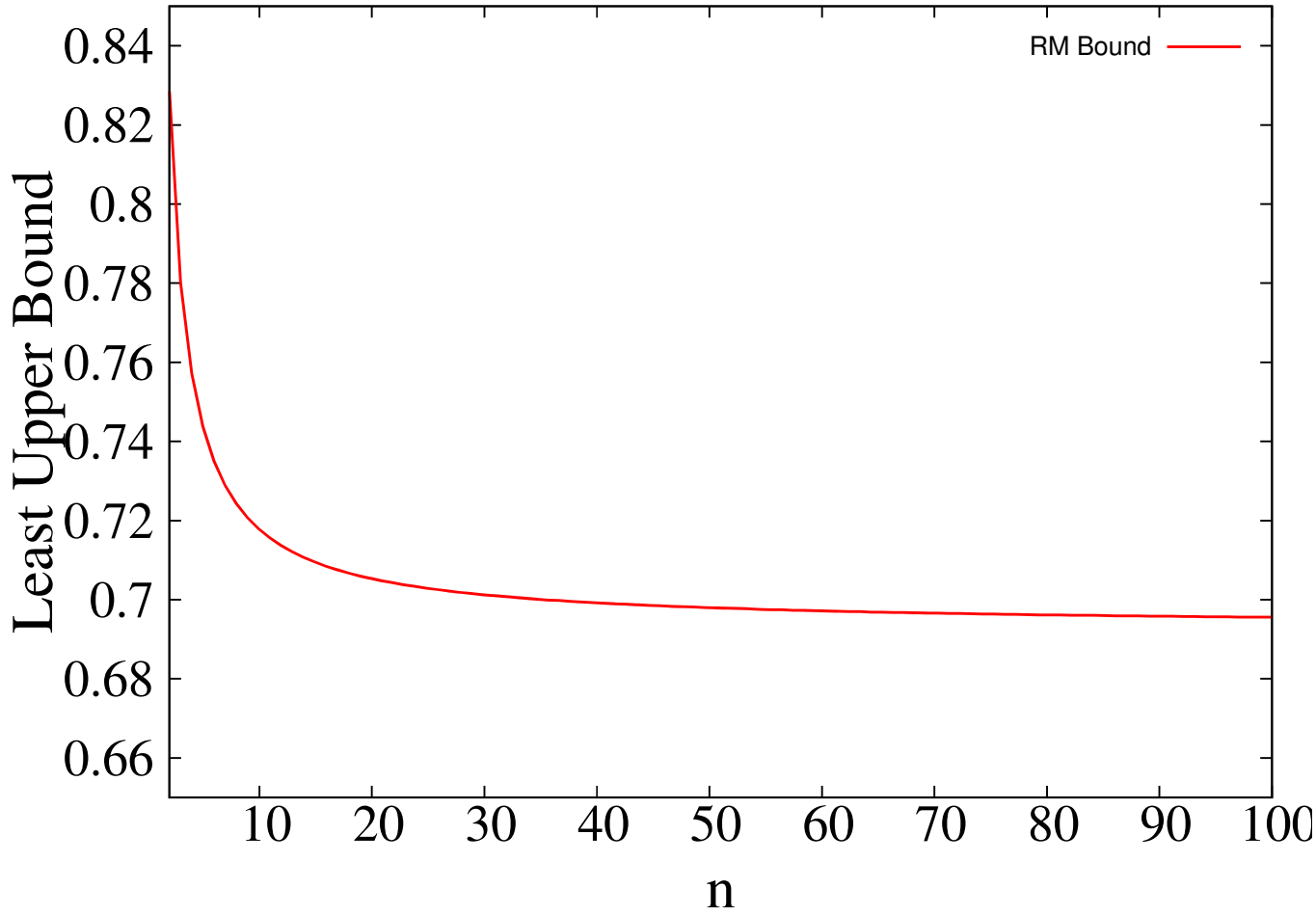
[Liu and Layland] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{\frac{1}{n}} - 1)$. In other words,

$$UB(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.$$

n	$UB(RM, n)$	n	$UB(RM, n)$
2	0.828	3	0.779
4	0.756	5	0.743
6	0.734	7	0.728
8	0.724	9	0.720
10	0.717	$\rightarrow \infty$	$\ln 2 \geq 0.693$

Utilization Bound in Terms of n

SAARLAND UNIVERSITY
COMPUTER SCIENCE



Proof Sketch for $UB(RM, n)$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Note: The original proof for this theorem by Liu and Layland is not correct. For a corrected proof, see R. Devillers & J. Goossens at `http://www.ulb.ac.be/di/ssd/goossens/lub.ps`. Note the proof we present is a bit different than the one presented by Buttazzo's textbook. Without loss of generality (why?), we will only consider task sets with distinct periods, i.e., $T_1 < T_2 < \cdots < T_n$. We will present our proof sketch in two parts:

1. First, we consider the special case where $T_n \leq 2T_1$.
2. Second, we show how to relax this constraint.

Proof Sketch: Difficult-To-Schedule

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Definition

A task set $\mathcal{T}_n$ is called *difficult-to-schedule* under scheduling algorithm A if $\mathcal{T}_n$ *fully utilizes* the processor if scheduled under A.

Our Strategy

- We seek the most difficult-to-schedule task set $\mathcal{T}_n$ for n tasks:
  A task set that is difficult-to-schedule with the minimal utilization.
- We derive a tight lower bound on the utilization of the most difficult-to-schedule task set $\mathcal{T}_n$ in terms of n, the $UB(RM, n)$.

Proof Sketch: Three Steps

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. For a task set with given periods, define the execution times for the most difficult-to-schedule task set.
2. Show that any difficult-to-schedule task set whose execution times differ from those in step 1 has a greater utilization than the most difficult-to-schedule task set provided in step 1.
3. Compute a closed-form expression for $UB(RM, n)$.

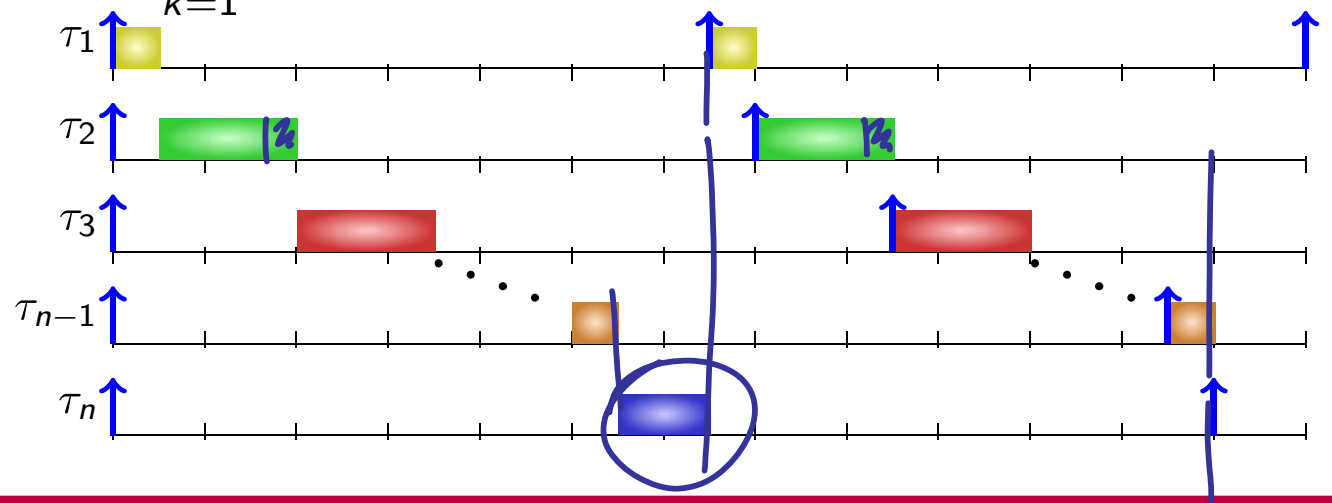
Step 1: Execution Time

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Suppose $\mathcal{T}_n^*$ is a task set with $T_n \leq 2T_1$ and

$$C_k = T_{k+1} - T_k \qquad \text{for } k = 1, 2, \ldots, n-1$$

$$C_n = T_n - 2\sum_{k=1}^{n-1} C_k = 2T_1 - T_n. \qquad \text{(Because: } \textstyle\sum_{k=1}^{n-1} C_k = T_n - T_1\text{)}$$



Theorem

Such a task set $\mathcal{T}_n^*$ is the *most difficult-to-schedule* task set.

Proof of the Most Difficult-to-Schedule Statement

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Proof strategy:
We show that the difficult-to-schedule task set given on the previous slide can be transformed into any difficult-to-schedule task set *without decreasing* the task set's utilization:

Starting with the highest priority task and working our way down to the lowest priority task, we incrementally modify the execution times of $\mathcal{T}_n^*$ to match any other difficult-to-schedule task set, and for each modification, utilization does not decrease.

If we need to increase (decrease) the execution time of task τ_i, then we compensate for this by decreasing (increasing) the execution time of some τ_k, where $k > i$.

Proof of the Most Difficult-to-Schedule: Increase by Δ

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Let's increase the execution of some task τ_i ($i < n$) by Δ ($\Delta > 0$).

$$C_i' = T_{i+1} - T_i + \Delta = C_i + \Delta.$$

To keep the processor busy up to T_n, we can decrease the execution time of a task τ_k ($k > i$) by Δ.

$$C_k' = C_k - \Delta.$$

Since $T_i < T_k$, the utilization of the above task set $\mathcal{T}_n'$ is no less than the original task set $\mathcal{T}_n^*$ by

$$U(\mathcal{T}_n') - U(\mathcal{T}_n^*) = \frac{\Delta}{T_i} - \frac{\Delta}{T_k} \geq 0.$$

Proof of the Most Difficult-to-Schedule: Decrease by Δ

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Let's decrease the execution of some task τ_i ($i < n$) by Δ ($\Delta > 0$).

$$C_i' = T_{i+1} - T_i - \Delta = C_i - \Delta.$$

To keep the processor busy up to T_n, we can increase the execution time of a task τ_k ($k > i$) by 2Δ. (*Why 2Δ?*)

$$C_k' = C_k + 2\Delta.$$

Since $T_k \leq 2T_i$, the utilization of the above task set $\mathcal{T}_n'$ is no less than the original task set $\mathcal{T}_n^*$ by

$$U(\mathcal{T}_n') - U(\mathcal{T}_n^*) = \frac{-\Delta}{T_i} + \frac{2\Delta}{T_k} \geq 0.$$

$$\geq \frac{-\Delta}{T_i} + \frac{2\Delta}{2T_i} = 0$$

Calculating $UB(RM, n) =$ The Utilization of the Most Difficult-to-Schedule Task Set

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Utilization of $\mathcal{T}_n^*$ for given task periods:
  Let x_i be $\frac{T_{i+1}}{T_i}$.

$$U(\mathcal{T}_n^*) = \frac{2T_1 - T_n}{T_n} + \sum_{i=1}^{n-1} \frac{T_{i+1} - T_i}{T_i} = 2\frac{T_1}{T_n} + (\sum_{i=1}^{n-1} x_i) - n$$

$$= \frac{2}{\Pi_{i=1}^{n-1} x_i} + (\sum_{i=1}^{n-1} x_i) - n$$

- Which task periods minimize the utilization?
  By getting partial derivatives of $U(\mathcal{T}_n^*)$ to the variables, we know that $U(\mathcal{T}_n^*)$ is minimized when

$$\frac{\partial U(\mathcal{T}_n^*)}{\partial x_k} = 1 - \frac{2(\Pi_{i=1}^{n-1} x_i)/x_k}{(\Pi_{i=1}^{n-1} x_i)^2} = 1 - \frac{2}{x_k \Pi_{i=1}^{n-1} x_i} = 0, \forall k = 1, 2, \ldots, n-1.$$

- Therefore, all x_i need to be equal for $k = 1, 2, \ldots, n-1$: $x_1 = x_2 = \cdots = x_{n-1}$:

$$x_k = \frac{2}{\Pi_{i=1}^{n-1} x_i} \Rightarrow x_k^n = 2 \Rightarrow x_k = 2^{\frac{1}{n}}.$$

Calculating $UB(RM, n)$ (cont.)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- By substituting $x_i = 2^{\frac{1}{n}}$ into our utilization formula, we have

$$U(\mathcal{T}_n^*) \leq n(2^{\frac{1}{n}} - 1)$$

Removing the Constraint on $T_n \leq 2T_1$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Strategy

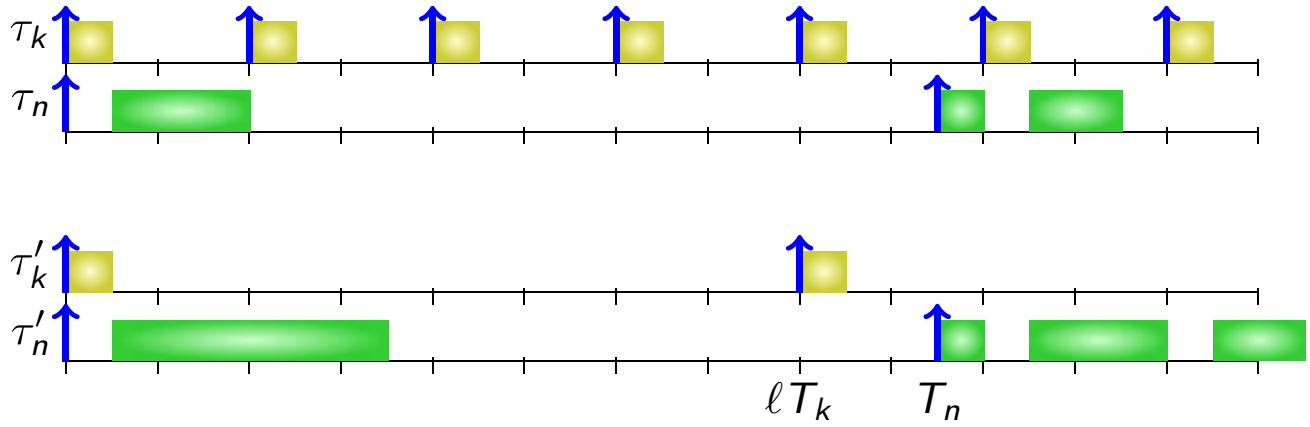
- Suppose that $\mathcal{T}_n$ is a difficult-to-schedule task set with n tasks.
- We are going to transform any difficult-to-schedule task set $\mathcal{T}_n$ with n tasks and $T_n > 2T_i$ for some i to $\mathcal{T}'_n$ such that
  1. the period T_n in $\mathcal{T}'_n$ is no more than twice of T_1 in $\mathcal{T}'_n$, and
  2. $U(\mathcal{T}_n) \geq U(\mathcal{T}'_n)$.

Transform $\mathcal{T}_n$ step-by-step to get $\mathcal{T}'_n$:

- Find a task τ_k in task set $\mathcal{T}_n$ with $\ell T_k < T_n \leq (\ell + 1)T_k$ and ℓ is an integer that is at least 2.
- Create a task τ'_n such that the period is T_n and the execution time C'_n is $C_n + (\ell - 1)C_k$.
- Create a task τ'_k such that the period is ℓT_k and the execution time C'_k is C_k.
- Let $\mathcal{T}'_n$ be $\mathcal{T}_n \setminus \{\tau_k, \tau_n\} \cup \{\tau'_k, \tau'_n\}$

Removing the Constraint on $T_n \leq 2T_1$ (cont.)

SAARLAND UNIVERSITY
COMPUTER SCIENCE



Conditions:

1. $\ell T_k < T_n \leq (\ell + 1) T_k$ with $\ell \geq 2$.
2. $T'_n = T_n, C'_n = C_n + (\ell - 1) C_k$ and $T'_k = \ell T_k$ and $C'_k = C_k$.

Results: since $\ell T_k < T_n$, we know

$$U(\mathcal{T}_n) - U(\mathcal{T}'_n) = \frac{C_n}{T_n} + \frac{C_k}{T_k} - \frac{(\ell - 1) C_k + C_n}{T_n} - \frac{C_k}{\ell T_k}$$

$$= \left(\frac{1}{\ell T_k} - \frac{1}{T_n} \right) (\ell - 1) C_k > 0$$

Concluding the RM Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Moreover $\mathcal{T}'_n$ above is also a difficult-to-schedule task set.
- By repeating the above procedure, we can transform to a task set $\mathcal{T}'_n$ with $T'_n \leq 2T'_1$ without increasing its utilization.

This concludes the proof of the following theorem:

Theorem

[Liu and Layland, JACM 1973] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{\frac{1}{n}} - 1)$. In other words,

$$UB(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.$$

$$\lim_{n\to\infty} UB(RM, n) = \lim_{n\to\infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \geq 0.693$$

Remarks on Harmonic Task Set

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- If the total utilization is larger than 0.693 but less than or equal to 1, the utilization-bound schedulability test cannot provide guarantees for schedulability or unschedulability.
- Sometimes, we can manipulate the periods such that the new task set is a harmonic task set and its schedulability can be used.

Improving Schedulability: Hyperbolic Bound

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

[Bini and Buttazzo, ECRTS 2001] A system of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if

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

Note that this is also only a sufficient schedulability test.

Recall Most Difficult-to-Schedule Task Set

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Suppose $\mathcal{T}_n^*$ is a task set with $T_n \leq 2T_1$ and

$$C_k = T_{k+1} - T_k \qquad \text{for } k = 1, 2, \ldots, n-1$$

$$C_n = T_n - 2\sum_{k=1}^{n-1} C_k = 2T_1 - T_n.$$

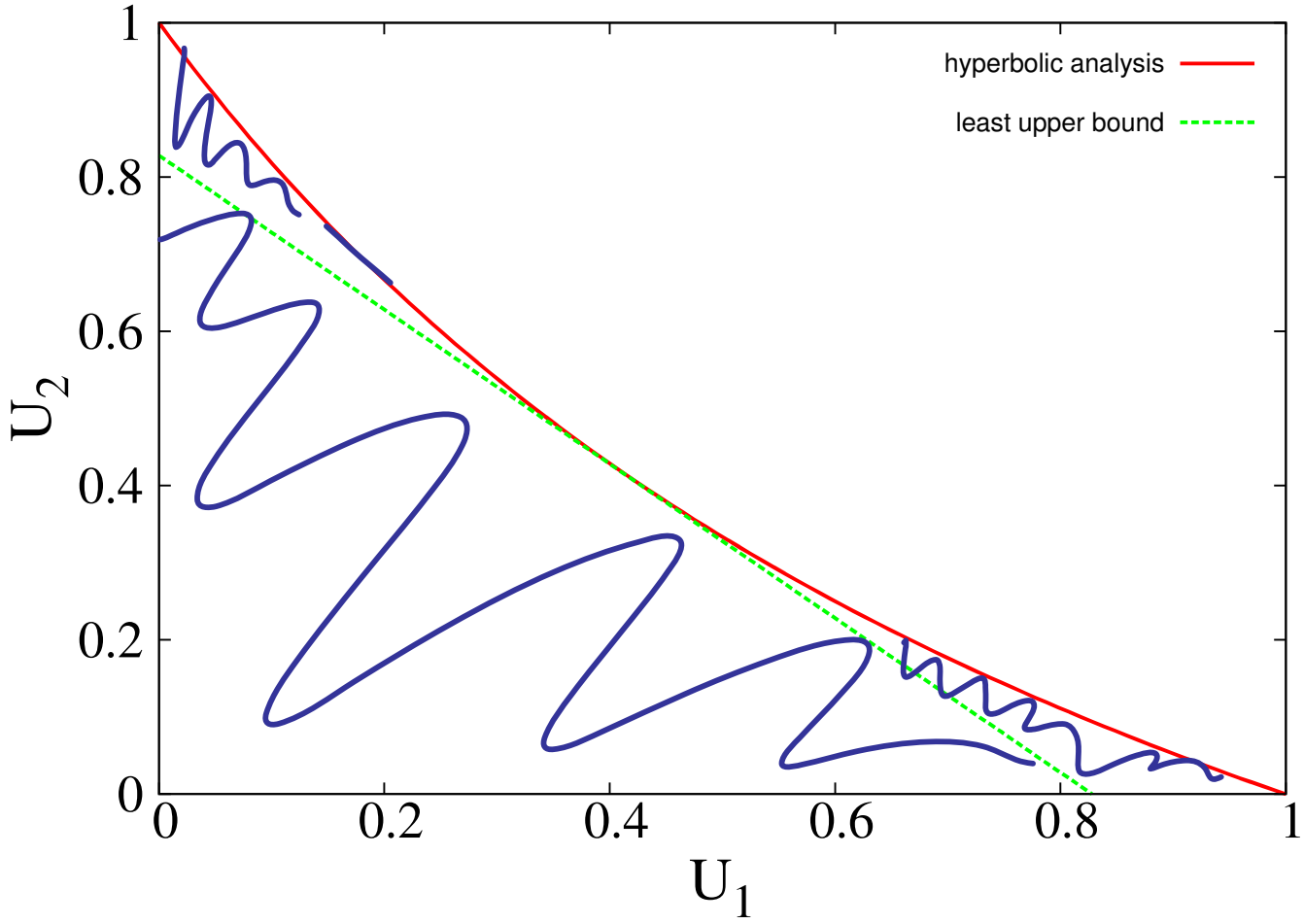
Let x_i be $\frac{T_{i+1}}{T_i}$.

- For $i = 1, 2, \ldots, n-1$, we have $U_i = x_i - 1 \Rightarrow U_i + 1 = x_i$.
- $U_n = 2\frac{T_1}{T_n} - 1 \Rightarrow U_n + 1 = 2\frac{T_1}{T_n}$.

$$\Pi_{i=1}^{n}(U_i + 1) = \left(\frac{T_2}{T_1}\frac{T_3}{T_2}\ldots\frac{T_n}{T_{n-1}}\right) \cdot 2\frac{T_1}{T_n}$$
$$= 2.$$

Hyperbolic Bound vs Utilization Bound

SAARLAND UNIVERSITY
COMPUTER SCIENCE



Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Necessary and Sufficient RM-Schedulability

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Time-demand analysis (TDA) was proposed by Lehoczky, Sha, and Ding.
- TDA can be applied to produce a schedulability test for any fixed-priority algorithm that ensures that each job of every task completes before the next job of that task is released.
- For some important task models and scheduling algorithms, this schedulability test is necessary and sufficient.

Schedulability Condition

The time-demand function $W_i(t)$ of the task τ_i is defined as follows:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Theorem

A system $\mathcal{T}$ of periodic, independent, preemptable tasks is schedulable on one processor by algorithm A if it holds that:

$$\forall \tau_i \in \mathcal{T} \; \exists t \text{ with } 0 < t \leq D_i \text{ and } W_i(t) \leq t$$

This condition is also necessary for synchronous, periodic task sets and also sporadic task sets.

Note that this holds for implicit-deadline and constrained-deadline task sets. The sufficient condition can be proved by contradiction.

How to Use TDA?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

The theorem of TDA might look strong as it requires to check all the times t with $0 < t \leq D_i$ for a given τ_i. There are two ways to avoid this:

- Iterate using $t(k+1) := W_i(t(k))$, starting with $t(0) := \sum_{j=1}^{i} C_j$, and stopping when, for some ℓ, $t(\ell) = W_i(t(\ell))$ or $t(\ell) > D_i$.
- Only consider $t \in \{\ell T_j - \epsilon\}\, 1 \leq j \leq i, \ell \in \mathcal{N}^+$, where ϵ is a constant close to 0. That is, only consider t at which a job of higher-priority tasks arrives.

Complexity of TDA Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

For analyzing whether task τ_i can meet the timing constraint, the complexity is $O(iD_i)$.

- It is polynomial time if the input is in the unary format. That is, when D_i is 6, the input is 111111 instead of 110 in the binary format.
- It is exponential time in the binary format.
- Formally, this is called pseudo-polynomial time complexity.

Theorem

Eisenbrand and Rothvoss [RTSS 2008]: Fixed-Priority Real-Time Scheduling: Response Time Computation Is $\mathcal{NP}$-hard.

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Utilization-Based Test for EDF Scheduling

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

Liu and Layland: A task set $\mathcal{T}$ of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

Proof

- The *only if* part is obvious: If $U > 1$, then some task clearly must miss a deadline. So, we concentrate on the *if* part.
- We prove the contrapositive, i.e., if $\mathcal{T}$ is not schedulable, then $U > 1$.
  - Let $J_{i,k}$ be the first job to miss its deadline.
  - Let t_{-1} be the last idle instant before $d_{i,k}$, the absolute deadline of $J_{i,k}$.
  - t_{-1} could be 0 if there is no idle time.

(cont.)

Proof of Utilization-Bound Test for EDF

Proof.

Because $J_{i,k}$ missed its deadline, we know that

$$d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k}) \text{ by jobs with deadline no more than } d_{i,k}$$

$$= \sum_{a_k \geq t_{-1}, d_k \leq d_{i,k}} C_k$$

$$= \sum_{j=1}^{n} \left\lfloor \frac{d_{i,k} - t_{-1}}{T_j} \right\rfloor C_j$$

$$\leq \sum_{j=1}^{n} \frac{d_{i,k} - t_{-1}}{T_j} C_j$$

This proof is actually also valid if relative deadlines are larger than periods.

By cancelling $d_{i,k} - t_{-1}$, we conclude the proof by

$$1 < \sum_{j=1}^{n} \frac{C_j}{T_j} = U.$$

Proof of Utilization-Bound Test for EDF

Proof.

Because $J_{i,k}$ missed its deadline, we know that

$$d_{i,k} - t_{-1} < \text{demand in } [t_{-1}, d_{i,k}) \text{ by jobs with deadline no more than } d_{i,k}$$

$$= \sum_{a_k \geq t_{-1}, d_k \leq d_{i,k}} C_k$$

$$= \sum_{j=1}^{n} \left\lfloor \frac{d_{i,k} - t_{-1}}{T_j} \right\rfloor C_j$$

$$\leq \sum_{j=1}^{n} \frac{d_{i,k} - t_{-1}}{T_j} C_j$$

This proof is actually also valid if relative deadlines are larger than periods.

By cancelling $d_{i,k} - t_{-1}$, we conclude the proof by

$$1 < \sum_{j=1}^{n} \frac{C_j}{T_j} = U.$$

Relative Deadlines Less than Periods

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

A task set $\mathcal{T}$ of independent, preemptable, periodic tasks with relative deadlines less than or equal to their periods can be feasibly scheduled (under EDF) on one processor if

$$\sum_{k=1}^{n} \frac{C_k}{D_k} \leq 1.$$

Note: This theorem only provides a sufficient condition.

Necessary and Sufficient Conditions

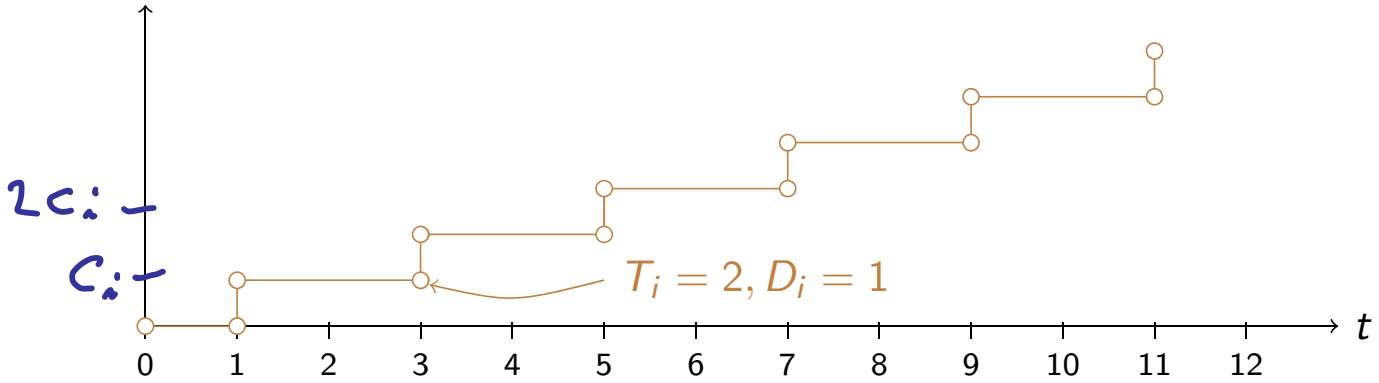
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Theorem

Define demand bound function $dbf(\tau_i, t)$ as

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} \cdot C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} \cdot C_i.$$

A task set $\mathcal{T}$ of independent, preemptable, periodic tasks with relative deadlines equal to or less than their periods can be feasibly scheduled (under EDF) on one processor if and only if $\forall L \geq 0, \sum_{i=1}^{n} dbf(\tau_i, L) \leq L$.



Proof for EDF Schedulability Test

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- The processor demand in time interval $[t_1, t_2]$ is the demand that must be finished in interval $[t_1, t_2]$. That is, only jobs that arrive no earlier than t_1 and have absolute deadline no more than t_2 are considered.
- The processor demand $g_i([t_1, t_2])$ contributed by task τ_i is

$$g_i([t_1, t_2]) = C_i \cdot \max\left\{0, \left\lfloor \frac{t_2 + T_i - D_i - \phi_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil\right\}$$

$\#$ of jobs with deadline no more than t_2

$\#$ of jobs with arrival time less than t_1

- The feasibility is guaranteed if and only if in any interval $[t_1, t_2]$, the processor demand is no more than the available time, i.e.,

$$t_2 - t_1 \geq \sum_{i=1}^{n} g_i(t_1, t_2) \geq \sum_{i=1}^{n} C_i \cdot \left\lfloor \frac{t_2 + T_i - D_i - t_1}{T_i} \right\rfloor$$

- Replacing $t_2 - t_1$ by L, we conclude the proof.

Complexity of the Exact Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

For analyzing whether a task set can be schedulable by EDF, the time complexity is $O(nL_{\max})$, where $L_{\max}$ is the hyper-period $LCM(T_1, T_2, \ldots, T_n)$.

- It takes pseudo-polynomial time.

Theorem

Eisenbrand and Rothvoss [SODA 2010]: testing EDF schedulability of such a task set is (weakly) co$\mathcal{NP}$-hard. That is, deciding whether a task set is not schedulable by EDF is (weakly) $\mathcal{NP}$-hard.

Comparison between RM and EDF (Implicit Deadlines)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

RM

- Low run-time overhead: $O(1)$ with priority sorting in advance
- Optimal for static priority
- Schedulability test is $\mathcal{NP}$-hard (even if the relative deadline = period)
- Utilization bound: 0.693
- In general, more preemptions

EDF

- High run-time overhead: $O(\log n)$ with balanced binary tree
- Optimal for dynamic priority
- Schedulability test is easy (when the relative deadline = period)
- Utilization bound: 1
- In general, fewer preemptions