Inovace tohoto kurzu byla v roce 2011/12 podpořena projektem CZ.2.17/3.1.00/33274 financovaným Evropským sociálním fondem a Magistrátem hl. m. Prahy.

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Department of Distributed and Dependable Systems

Embedded and Real-time Systems

Periodic Task Scheduling

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Tomáš Bureš

<bures@d3s.mff.cuni.cz>

CHARLES UNIVERSITY IN PRAGUE

Faculty of Mathematics and Physics

Periodic tasks

- Periodic tasks – A type of task that consists of a sequence of identical instances, activated at regular intervals.

- Examples
  - Speed regulation
  - Monitoring sensors
  - Audio/video sampling

Periodic tasks scheduling

Periodic tasks scheduling

- **Offline scheduling algorithms**
  - *Timeline (cyclic) scheduling (TS)*
- **Online scheduling algorithms**
  - **Static priorities**
    - *Rate Monotonic (RM)*
    - *Deadline Monotonic (DM)*
  - **Dynamic priorities**
    - *Earliest Deadline First (EDF)*
    - *RM + Priority Inheritance Protocol*

Assumptions

- Tasks assumptions:
  - A1 – instance of a periodic task τ_i are regularly activated
  - A2 – all instances τ_i have the same C_i
  - A3 – all instances τ_i have the same D_i which is equal to T_i
  - A4 – All tasks are independent (no precedence relations, no resource constraints)

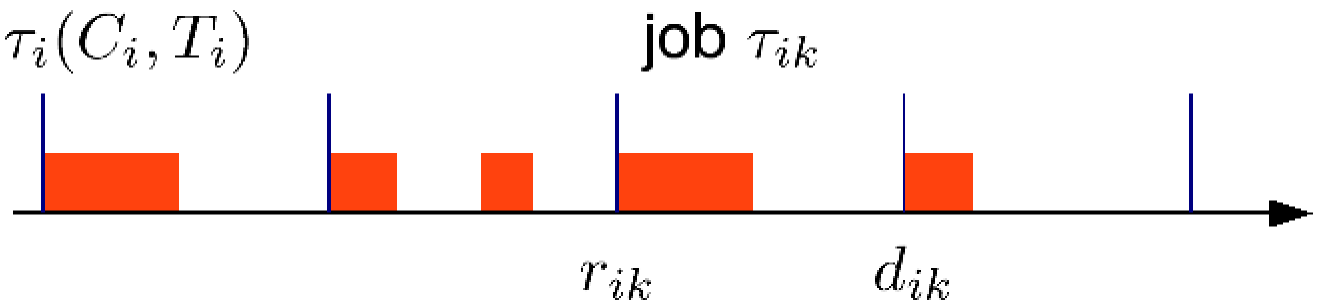
- Implicit assumptions:
  - No task can suspend itself (e.g. on I/O oper.)
  - All tasks are fully preemptable
  - All overheads in kernel are assumed to be zero

- A1, A2 – OK (reflects reality)
- A3, A4 – too tight for practical applications
  - Will be relaxed in future

Notation

- Γ – task set
- τ_i – a generic periodic task
- T_i – period of the task τ_i
- C_i – execution time within a period
- D_i – relative deadline of τ_i
- $\tau_{i,j}$ – j^{th} instance of the task τ_i
- $r_{i,j}$ – release time of $\tau_{i,j}$
- $s_{i,j}$ – start time of $\tau_{i,j}$
- $f_{i,j}$ – finishing (completion) time of $\tau_{i,j}$
- $d_{i,j}$ – absolute deadline of $\tau_{i,j}$

Periodic tasks



- For each periodic task, guarantee that:
  - each job τ_{ik} is activated at $r_{ik} = (k-1)T_i$
  - each job τ_{ik} completes within
$$d_{ik} = r_{ik} + D_i = r_{ik} + T_i = kT_i$$

Timeline Scheduling (Cyclic Scheduling)

- It has been used for 30 years in military systems, navigation, and monitoring systems

- Examples:
  - Air traffic control
  - Space Shuttle
  - Boeing 777

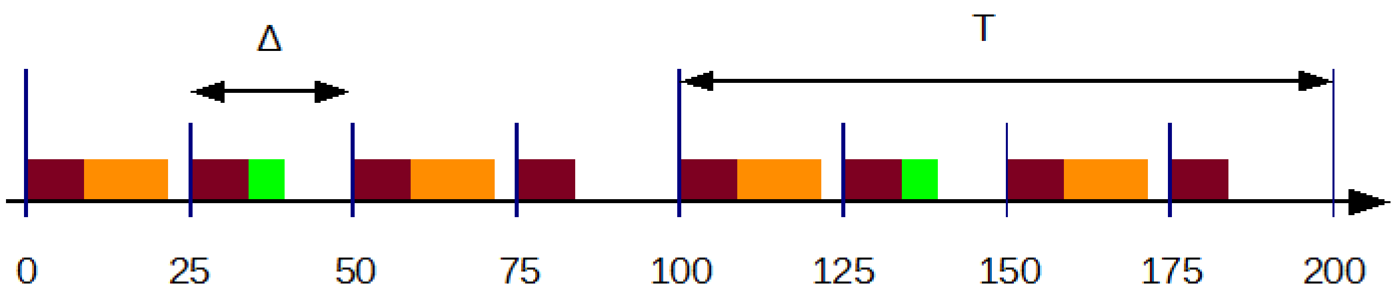
Timeline Scheduling

- Method
  - The time axis is divided in intervals of equal length (time slots)
  - Each task is statically (offline) allocated in a slot in order to meet the desired request rate.
  - The execution in each slot is activated by a timer.
    - Order is determinate in advance
      - Based on major cycle

Example

task	f	T
A	40 Hz	25 ms
B	20 Hz	50 ms
C	10 Hz	100ms

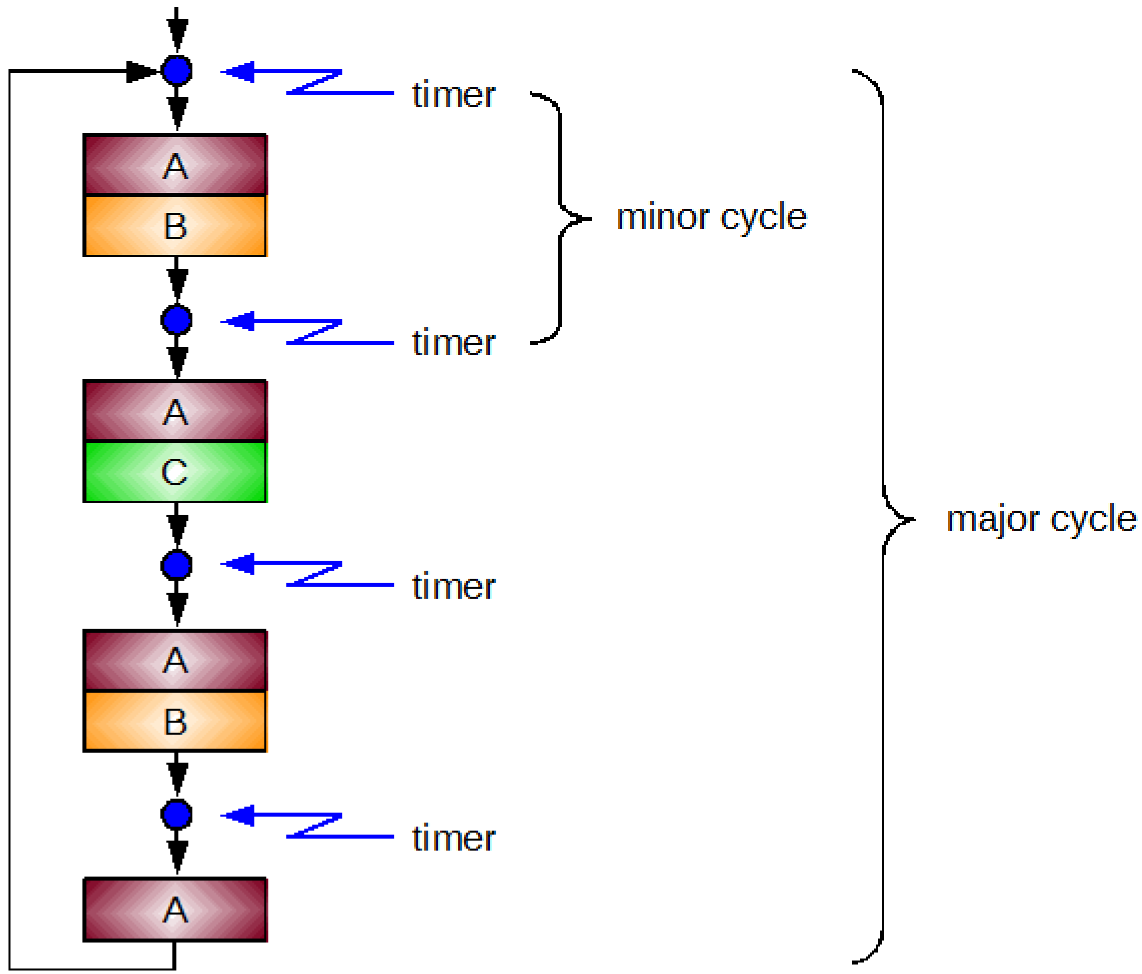
Δ = GCD (minor cycle)
T = lcm (major cycle)



GCD – greatest common divisor
lcm – least common multiple

- Guarantee:
  - $C_a + C_b \leq \Delta$
  - $C_a + C_c \leq \Delta$

Implementation



Timeline Scheduling

- Advantages
  - Simple implementation
  - Low run-time overhead
    - No context switches
  - It allows jitter control
    - Ordering of tasks inside major cycle
- Disadvantages
  - It is not robust during overloads
  - It is difficult to expand the schedule
  - It is not easy to handle aperiodic activities
- But in fact, it suffices in many cases!

Department of Distributed and Dependable Systems

Problems during Overloads

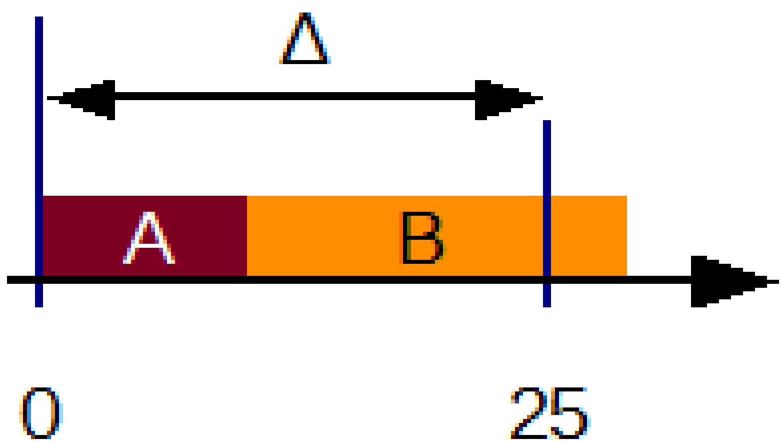
- Problem typical for off -line scheduling
  - Fragility during overload conditions

- What do we do during task overruns?
  - Let the task continue
    - we can have a domino effect on all the other tasks (timeline break)
  - Abort the task
    - the system can remain in an inconsistent state

Problems of Schedule Expandability

- If one or more tasks need to be upgraded (C or T change), we may have to redesign the whole schedule again.

- Example:
  - $C_a + C_b > \Delta$
  - C_B is updated but

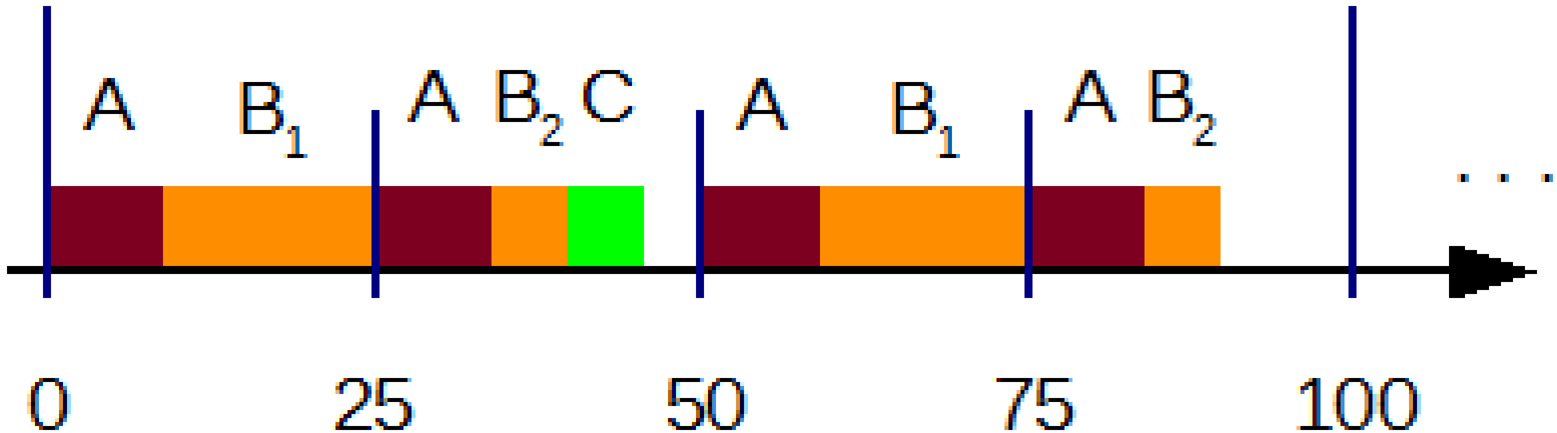


- Requires division of task B into smaller tasks

Department of Distributed and Dependable Systems

Problems of Schedule Expandability

- We have to split task B into two subtasks (B_1, B_2) and rebuild the schedule:



- Guarantee:
  - $C_a + C_{b_1} \leq \Delta$
  - $C_a + C_{b_2} + C_c \leq \Delta$

Problems of Schedule Expandability

- If the frequency of a task is changed, the impact can be even more significant

task	f	T
A	40 Hz	25 ms
B	25 Hz	**40 ms**
C	10 Hz	100ms

	before	after
minor cycle:	$\Delta = 25$	$\Delta = 5$
major cycle:	$T = 100$	$T = 200$

- 40 minor cycles within one major cycle!

Problem with aperiodic tasks

- Difficult to handle aperiodic tasks
  - Requires **on-line** change in task sequence

- **Slot-shifting technique**
  - Spare capacities – how much off-line tasks can be shifted at runtime while still meeting timing constraints
  - At runtime deadline-base algorithm uses spare capacities to schedule aperiodic tasks

- In complex or open systems, it is better to use on-line priority-based scheduling.

Department of Distributed and Dependable Systems

Priority-based Scheduling

- Each task is assigned a priority based on its timing constraints
- We verify the feasibility of the schedule using analytical techniques
- Tasks are executed on a priority-based kernel

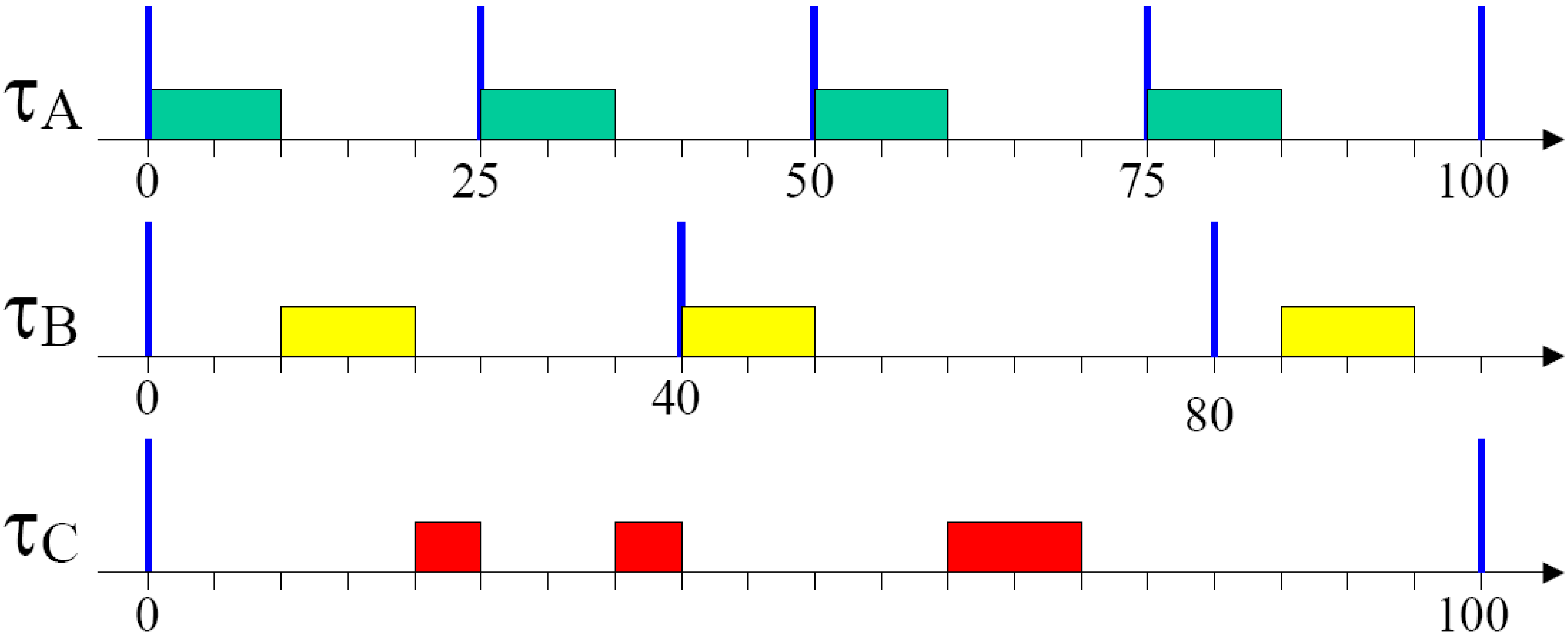
Rate Monotonic Scheduling (RM)

- Each task is assigned a fixed priority proportional to its rate (T)
  - Priorities are assigned before execution (T-based)
  - Preemptive

- Recall of basic assumptions
  - A1 – C_i is constant for every instance of τ_i
  - A2 – T_i is constant for every instance of τ_i
  - A3 – For each task, $D_i = T_i$
  - A4 – Tasks are independent:
    - no precedence relations
    - no resource constraints
    - no blocking I/O operations

Department of Distributed and Dependable Systems

RM Example



How Can We Verify Feasibility?

- Each task uses the processor for a fraction of time

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

- Hence the total processor utilization is

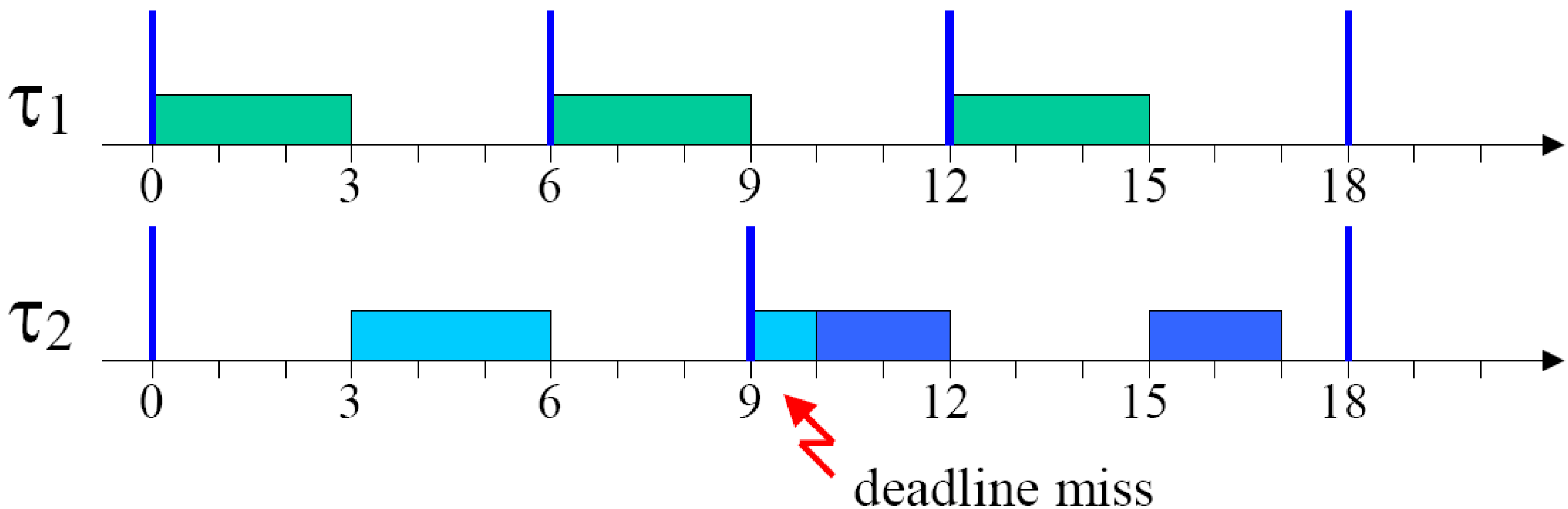
$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Department of Distributed and Dependable Systems

A Necessary Condition

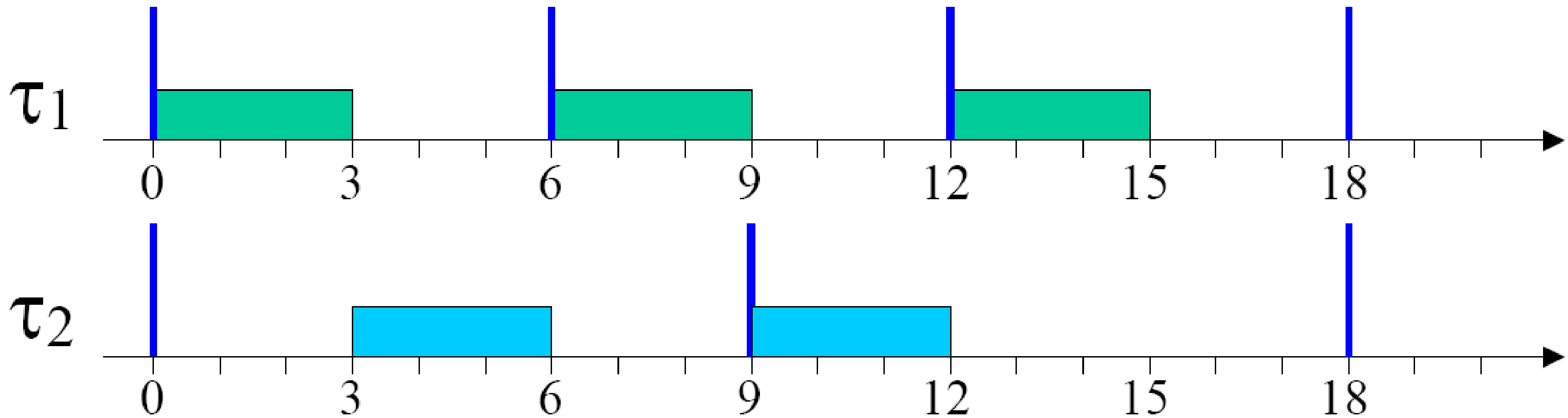
- If $U_p > 1$ the processor is overloaded hence the task set cannot be schedulable
- However, there are cases in which $U_p < 1$ but the task set is not schedulable by RM

- Example: $U_p = \frac{3}{6} + \frac{4}{9} = 0.944$



Utilization Upper Bound

- $U_{ub} = \frac{3}{6} + \frac{3}{9} = 0.833$



- The upper bound U_{ub} depends on the specific task set.

The Least Upper Bound

$$U_{lub} = \min_{\Gamma} U_{ub}(\Gamma, A)$$

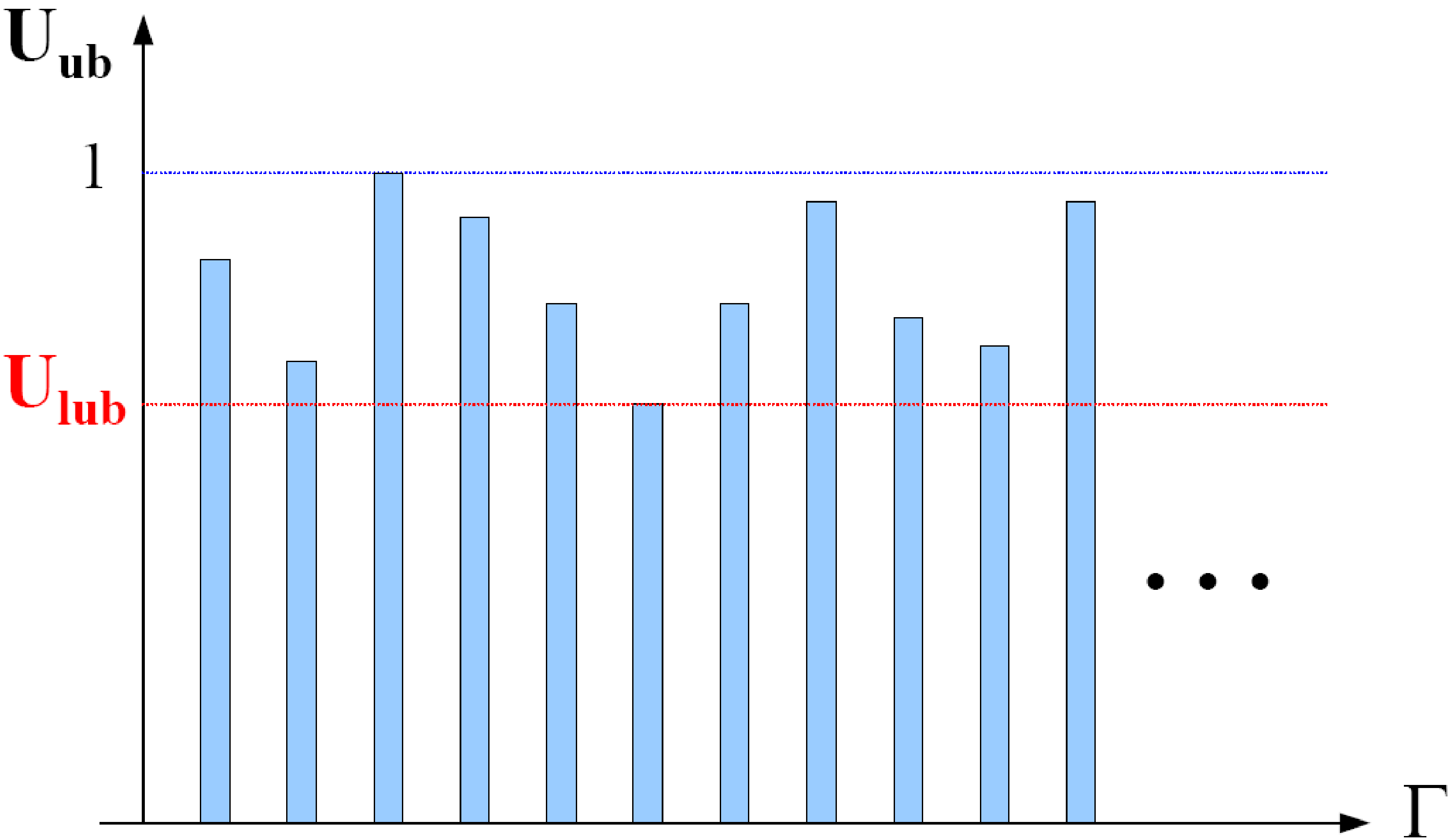


Figure taken from Buttazzo, G.:Task scheduling

A Sufficient Condition

- If $U_p \leq U_{lub}$ the task set is certainly schedulable with the RM algorithm

- **Note:** If $U_{lub} < U_p \leq 1$ we cannot say anything about the feasibility of that task set.

Department of Distributed and Dependable Systems

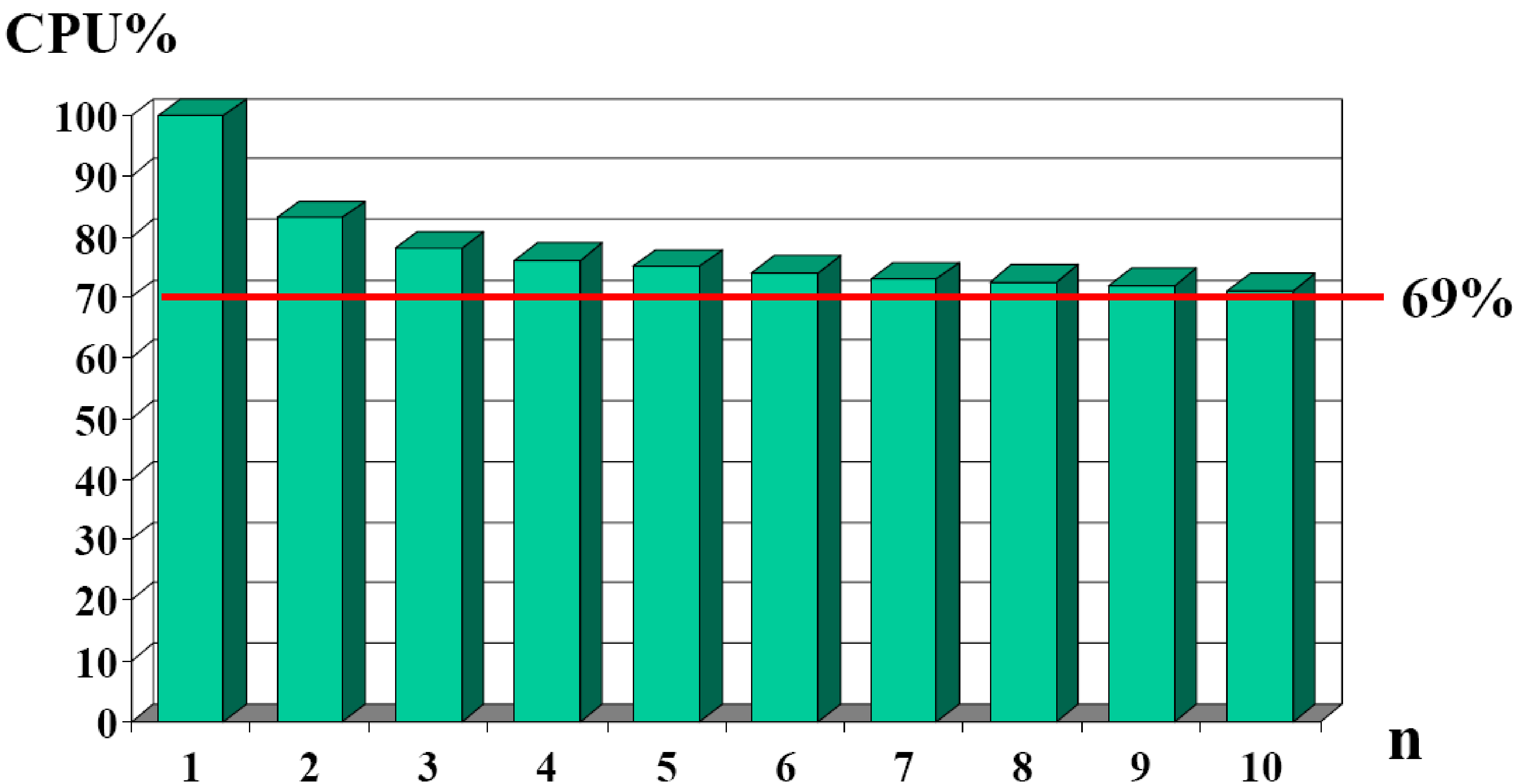
U_{lub} for RM

- In 1973, Liu and Leyland proved that for a set of n periodic tasks scheduled by RM:

$$\boxed{U_{lub} = n(2^{\frac{1}{n}} - 1)}$$

- for $n \to \infty$: $U_{lub} \to \ln 2$

RM Schedulability



RM Guarantee Test

- We compute the processor utilization factor as

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

- Guarantee Test (only sufficient)

$$U_p \leq n(2^{\frac{1}{n}} - 1)$$

Department of Distributed and Dependable Systems

RM Optimality

- RM is optimal among all fixed priority algorithms:
  - If there exists a fixed priority assignment which leads to a feasible schedule for Γ, then RM assignment is feasible for Γ
  - If Γ is not schedulable by RM, then it cannot be scheduled by any fixed priority assignment

Department of Distributed and Dependable Systems

Proof of RM Optimality

- Periodic tasks vocabulary:
  - **Response time of an instance** – a time (measured from the release time) at which the instance is terminated
    - $R_{i,k} = f_{i,k} - r_{i,k}$

- **Critical instant of a task** – a time at which the release of a task will produce the largest response time

Department of Distributed and Dependable Systems

Proof of RM Optimality

- First, we show that a critical instant for any task occurs whenever the task is released simultaneously with all higher-priority tasks.

- Let $\Gamma = \{\tau_1, \dots, \tau_n\}$ be the set of periodic tasks ordered by increasing periods, with τ_n being the task with the longest period. According to RM, τ_n will be the task with the lowest priority.

Critical Instant

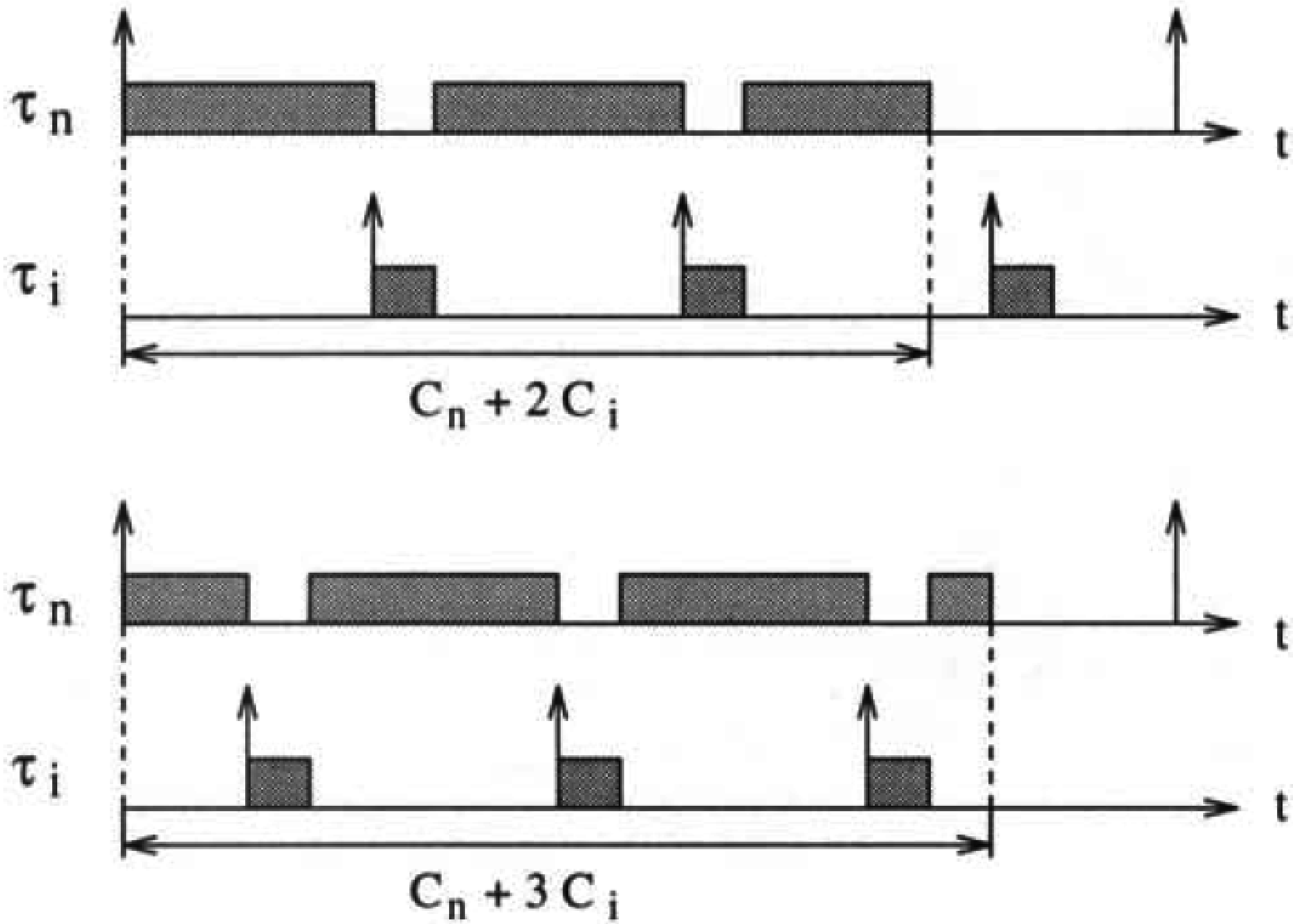


Figure from Buttazzo, G.:Hard RT Comp. Systems

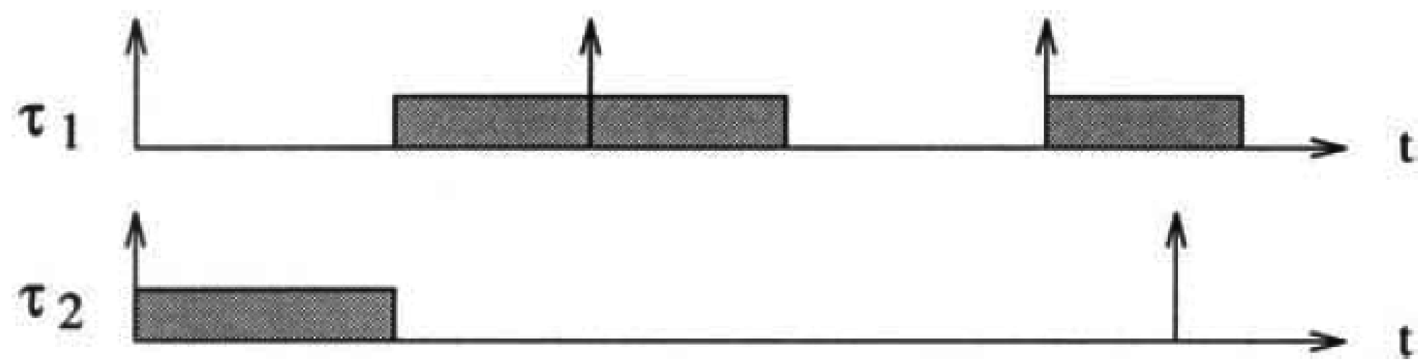
Proof of RM Optimality

- Schedulability of a task can be checked at the critical instant
  - If each of the tasks is schedulable at its critical instant, the whole task set is schedulable

- RM optimality is justified by showing that if a task set is schedulable by an arbitrary priority assignment, then it is also schedulable by RM.

Proof of RM Optimality

- Consider a set of two periodic tasks τ_1 and τ_2, with $T_1 < T_2$. If the priorities are not according to RM, then task τ_2 will receive greater priority.



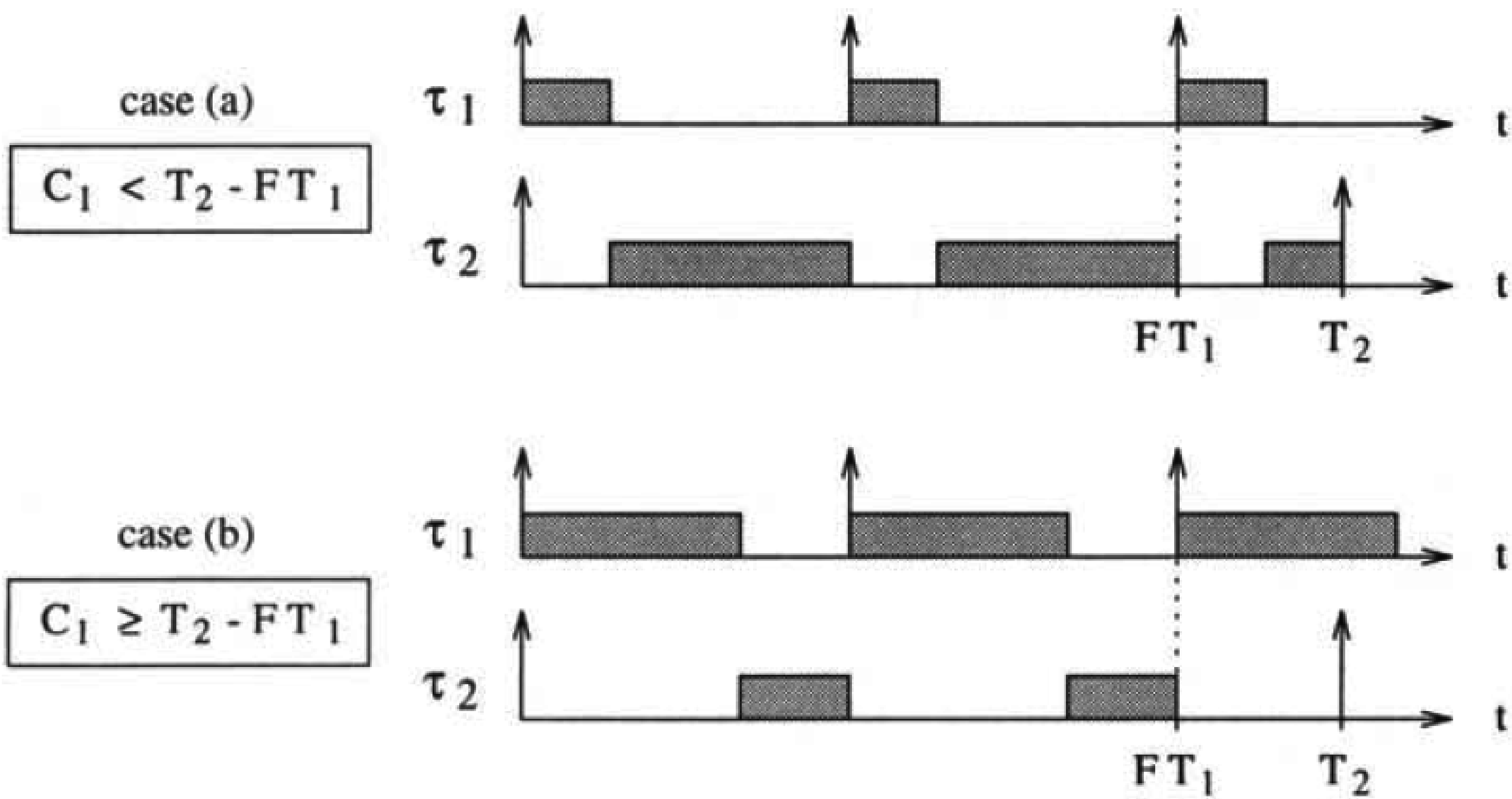
- The schedule is feasible if:

$$C_1 + C_2 \leq T_1 \qquad \textbf{\#1}$$

Department of Distributed and Dependable Systems

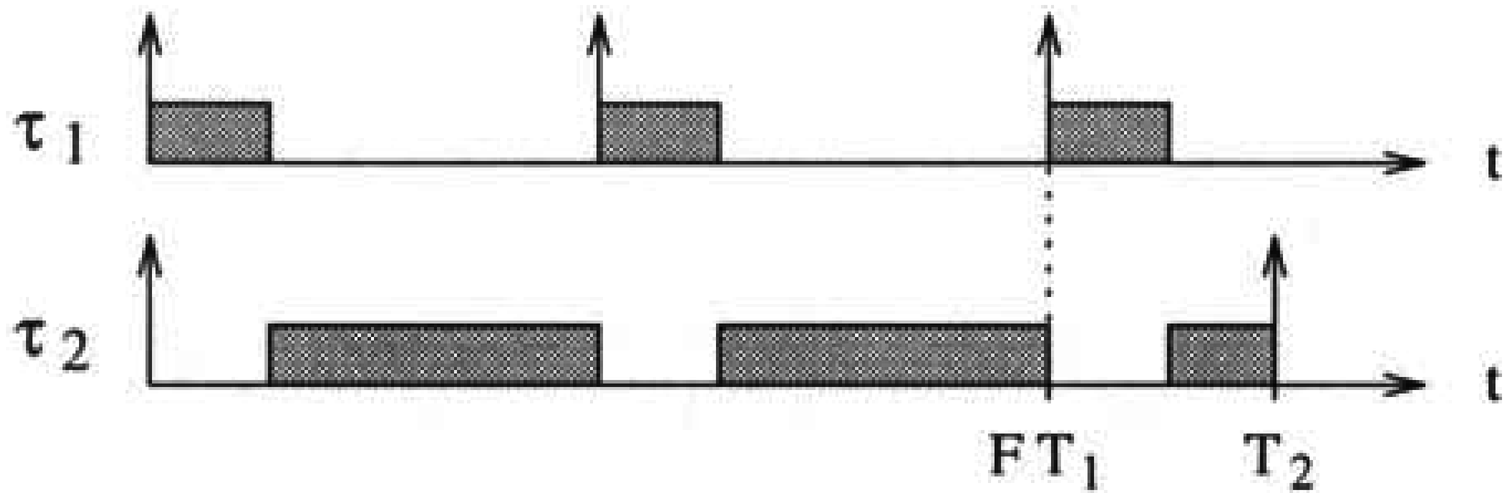
Proof of RM Optimality

- Let $F = \lfloor T_2/T_1 \rfloor$ be the number of periods of τ_1 entirely contained in T_2. We distinguish two cases



Proof of RM Optimality – Case 1

- Case 1
  - The computation time C_1 is short enough that all requests of τ_1 within the critical time zone of τ_2 are completed before the second request of τ_2. That is, $C_1 < T_2 - FT_1$



Proof of RM Optimality – Case 1

- The task set is schedulable if

$$(F+1)C_1 + C_2 \le T_2 \qquad \textbf{\#2}$$

- We show how #1 implies #2

$$FC_1 + FC_2 \le FT_1$$

- Since $F \ge 1$, we can write:

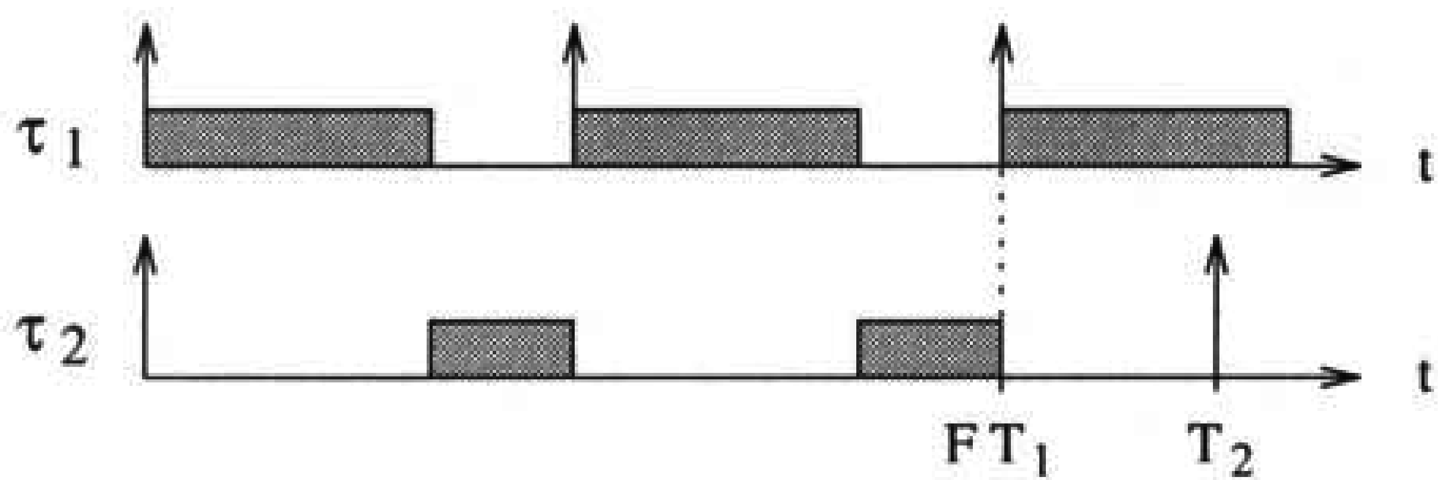
$$FC_1 + C_2 \le FC_1 + FC_2 \le FT_1$$
$$(F+1)C_1 + C_2 \le FT_1 + C_1$$

- Since $C_1 \le T_2 - FT_1$, we have:

$$(F+1)C_1 + C_2 \le FT_1 + C_1 \le T_2$$

Proof of RM Optimality – Case 2

- Case 2
  - The execution of the last request of τ_i in the critical time zone of τ_2 overlaps the second request of τ_2. That is, $C_1 \geq T_2 - FT_1$.



Proof of RM Optimality – Case 2

- The task set is schedulable if $FC_1 + C_2 \leq FT_1$. **#3**

- We show how #1 implies #3

$$FC_1 + FC_2 \leq FT_1$$

- Since $F \geq 1$, we can write:

$$FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1$$

- This can be generalized to n tasks

Department of Distributed and Dependable Systems

Calculation of U_{lub} for two tasks

- Two periodic tasks τ_1 and τ_2 with $T_1 < T_2$
- We have to
  - Assign priorities to tasks according to RM, so that τ_1 is the task with the highest priority;
  - Compute the upper bound U_{ub} for the set by setting tasks' computation times to fully utilize the processor;
  - Minimize the upper bound U_{ub} with respect to all other task parameters.

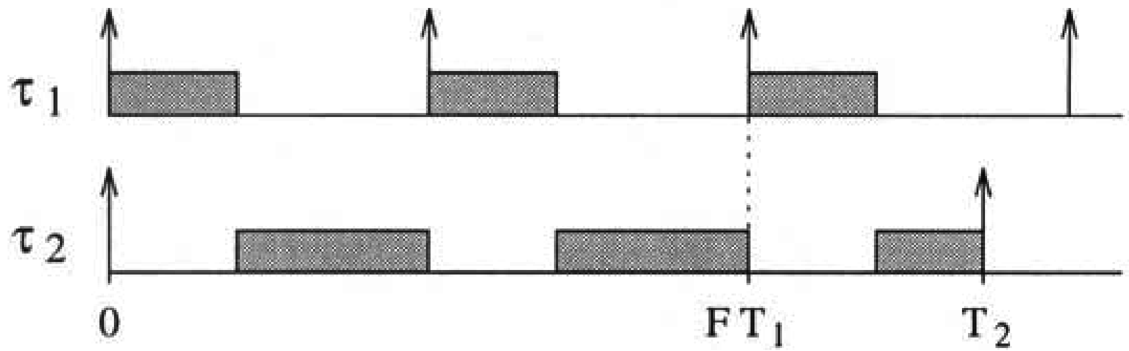
- As before, let $F = \lfloor T_2/T_1 \rfloor$ be the number of periods of τ_1 entirely contained in T_2. Without loss of generality, the computation time C_2 is adjusted to fully utilize the processor.

Department of Distributed and Dependable Systems

Calculation of U_{lub} for two tasks

- Two cases must be considered:

case (a) $\boxed{C_1 < T_2 - FT_1}$



case (b) $\boxed{C_1 \geq T_2 - FT_1}$

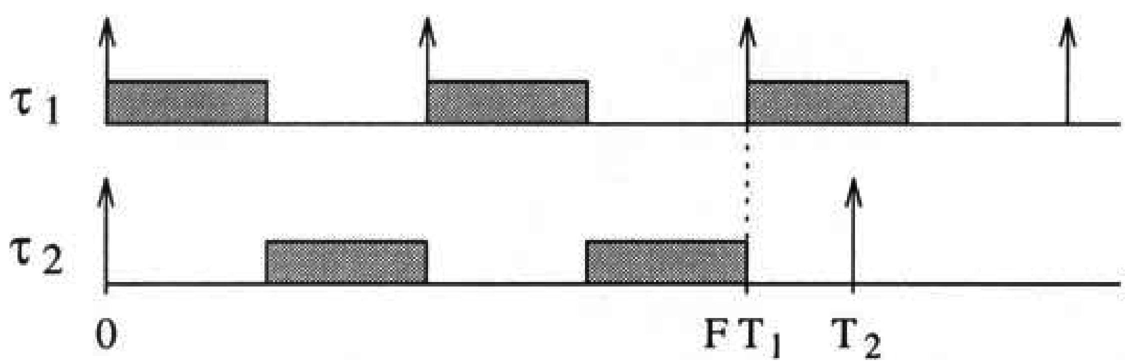


Figure from Buttazzo, G.:Hard RT Comp. Systems

Calculation of U_{lub} for two tasks – Case 1

- The computation time C_1 is short enough that all requests of τ_1 within the critical time zone of τ_2 are completed before the second request of τ_2. That is, $C_1 \leq T_2 - FT_1$.
- In this situation, the largest possible value of C_2 is $C_2 = T_2 - C_1(F + 1)$

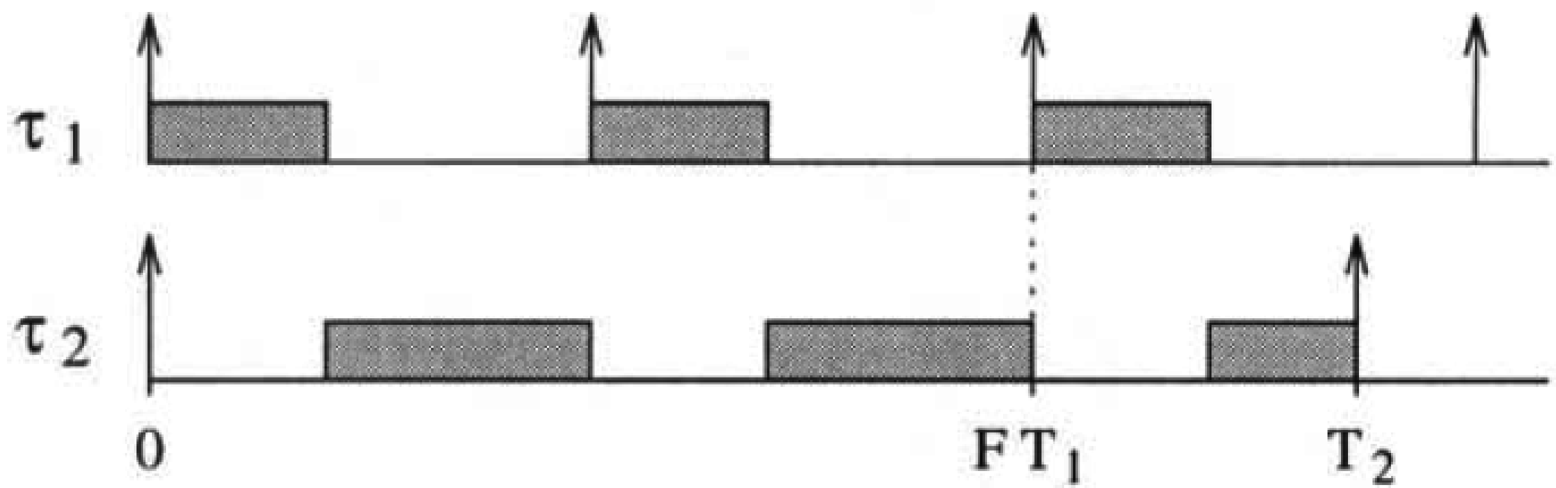


Figure from Buttazzo, G.:Hard RT Comp. Systems

Calculation of U_{lub} for two tasks – Case 1

- The corresponding upper bound U_{ub} is thus

$$U_{ub} = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - C_1(F+1)}{T_2}$$

$$= 1 + \frac{\mathrm{C}_1}{\mathrm{T}_1} - \frac{\mathrm{C}_1}{\mathrm{T}_2}(\mathrm{F}+1) = 1 + \frac{\mathrm{C}_1}{\mathrm{T}_2}\left[\frac{\mathrm{T}_2}{\mathrm{T}_1} - (\mathrm{F}+1)\right]$$

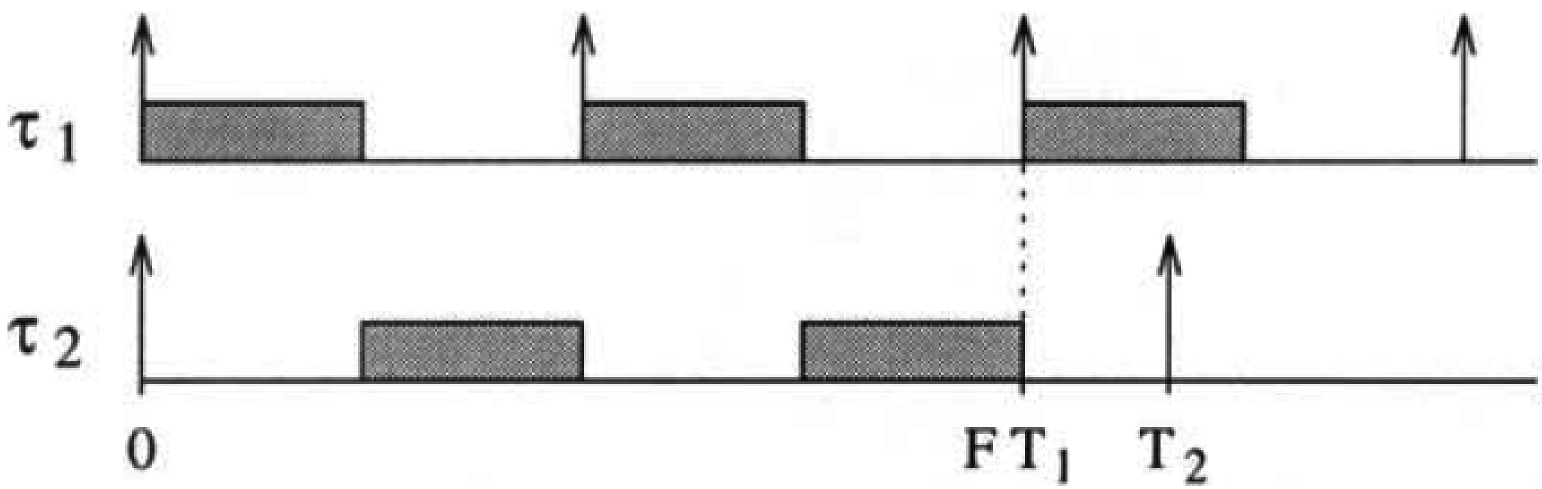
- Since the quantity in the square brackets is negative, U_{ub} is monotonically decreasing in C_1, and, being $C_1 \leq T_2 - FT_1$, the minimum of U_{ub} occurs for

$$C_1 = T_2 - FT_1$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for two tasks – Case 2

- The execution of the last request of τ_1 in the critical time zone of τ_2 overlaps the second request of τ_2. That is, $C_1 \geq T_2 - FT_1$
- In this situation, the largest possible value of C_2 is $C_2 = (\mathrm{T}_1 - \mathrm{C}_1)\mathrm{F}$



Calculation of U_{lub} for two tasks – Case 2

- The corresponding upper bound U_{ub} is thus

$$U_{ub} = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{(T_1 - C_1)F}{T_2} = \frac{T_1}{T_2}F + \frac{C_1}{T_1} - \frac{C_1}{T_2}F$$

$$= \frac{T_1}{T_2}F + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]$$

- Since the quantity in the square brackets is positive, U_{ub} is monotonically increasing in C_1, and, being $C_1 \geq T_2 - FT_1$, the minimum of U_{ub} occurs for

$$C_1 = T_2 - FT_1$$

Department of Distributed and Dependable Systems

Calculation of Ulub for two tasks

- In both cases, the minimum value of U_{ub} occurs for $C_1 = T_2 - T_1 F$
- Using the minimal value of C_1, we have:

$$U = \frac{T_1}{T_2} F + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right) = \frac{T_1}{T_2} F + \frac{(T_2 - T_1 F)}{T_2}\left(\frac{T_2}{T_1} - F\right)$$

$$= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]$$

- To simplify notation, let $G = T_2/T_1 - F$. Thus,

$$U = \frac{T_1}{T_2}(F + G^2) = \frac{(F + G^2)}{T_2/T_1} = \frac{(F + G^2)}{(T_2/T_1 - F) + F} = \frac{F + G^2}{F + G}$$

$$= \frac{(F + G) - (G - G^2)}{F + G} = 1 - \frac{G(1 - G)}{F + G}$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for two tasks

- Since $0 \leq F < 1$, the term $G(1-G)$ is nonnegative. Hence, U is monotonically increasing with F. As a consequence, the minimum of U occurs for the minimum value of F; namely, $F = 1$. Thus,

$$U = \frac{1+G^2}{1+G}$$

- Minimizing U over G, we have

$$\frac{dU}{dG} = \frac{2G(1+G)-(1+G^2)}{(1+G)^2} = \frac{G^2+2G-1}{(1+G)^2}$$

- and $dU/dG = 0$ for $G^2 + 2G - 1 = 0$, which has two solutions:

$$G_1 = -1-\sqrt{2} \qquad \mathrm{G}_2 = -1+\sqrt{2}$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for two tasks

- Since $0 \leq G < 1$, the negative solution $G = G_1$ is discarded. Thus, the least upper bound of U is given for $G = G_2$:

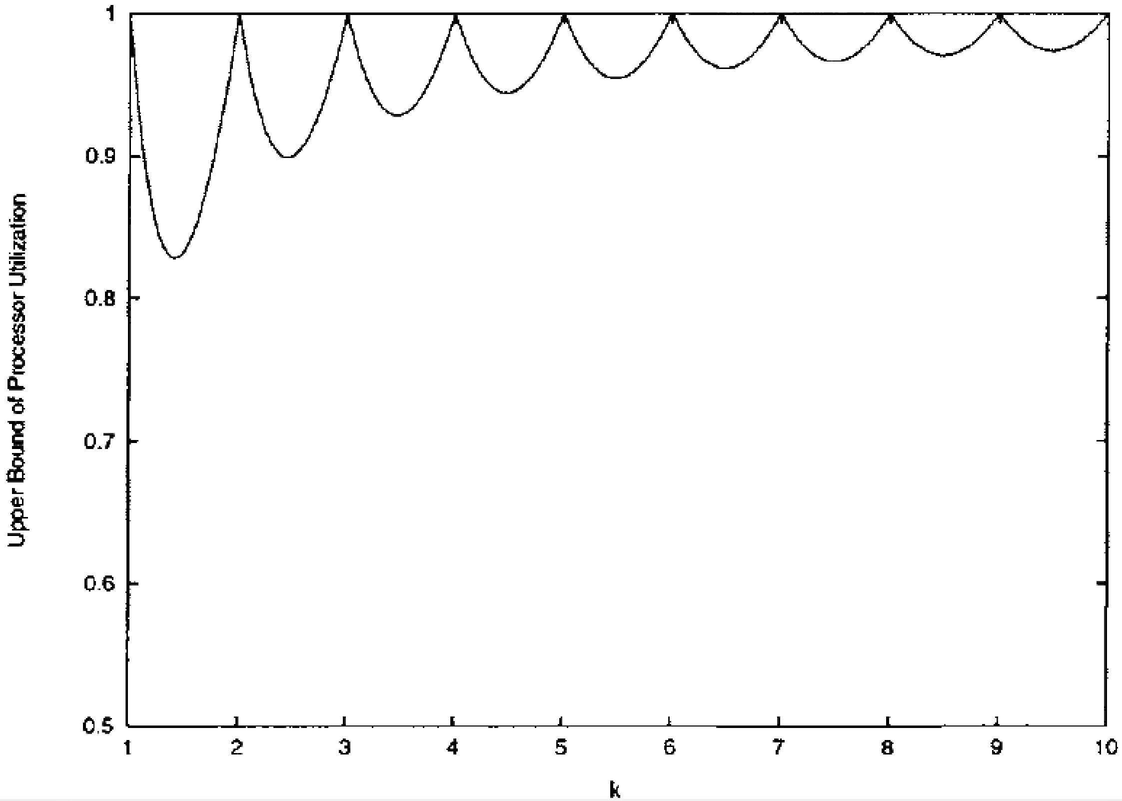
$$U_{lub} = \frac{1 + \left(\sqrt{2} - 1\right)^2}{1 + \left(\sqrt{2} - 1\right)} = \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2\left(\sqrt{2} - 1\right)$$

- That is,

$$U_{lub} = 2\left(2^{1/2} - 1\right) \simeq 0.83$$

Calculation of U_{lub} for two tasks

- Notice that if T_2 is a multiple of T_1, $G = 0$ and the processor utilization factor becomes 1.
  In general, the utilization factor for two tasks can be computed as a function of the ratio $k = T_2/T_1$



Calculation of U_{lub} for n tasks

- From the previous, the conditions for computing the least upper bound were:

$$F = 1$$
$$C_1 = T_2 - FT_1$$
$$C_2 = (T_1 - C_1)F$$

- which can be rewritten as

$$T_1 < T_2 < 2T_1$$
$$C_1 = T_2 - T_1$$
$$C_2 = 2T_1 - T_2$$

Calculation of U_{lub} for n tasks

- Generalizing for an arbitrary set of n tasks, the worst conditions for the schedulability of a task set that fully utilizes the processor are

$$T_1 < T_n < 2_{T_1}$$
$$C_1 = T_2 - T_1$$
$$C_2 = T_3 - T_2$$
$$\ldots$$
$$C_{n-1} = T_n - T_n - 1$$
$$C_n = T_1 - (C_1 + C_2 + \cdots + C_{n-1}) = 2T_1 - T_n$$

Calculation of U_{lub} for n tasks

- Thus, the processor utilization factor becomes
$$U = \frac{T_2 - T_1}{T_1} + \frac{T_3 - T_2}{T_2} + \cdots + \frac{T_n - T_{n-1}}{T_{n-1}} + \frac{2T_1 - T_n}{T_n}$$
- Defining $R_i = T_{i+1}/T_i$,
and noting that $R_1 R_2 \ldots R_{n-1} = T_n/T_1$,
the utilization factor may be re-written as
$$U = \sum_{i=1}^{n-1} R_i + \frac{2}{R_1 R_2 \ldots R_{n-1}} - n$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for n tasks

- To minimize U over $R_i, i = 1, \dots, n-1$, we have

$$\frac{\partial U}{\partial R_k} = 1 - \frac{2}{R_i^2 \left(\prod_{i \neq k}^{n-1} R_i\right)}$$

- Defining $P = R_1 R_2 \dots R_{n-1}$, U is minimum when

$$R_1 P = 2$$
$$R_2 P = 2$$
$$R_{n-1} P = 2$$

- That is, when all R_i have the same value:

$$R_1 = R_2 = \cdots = R_{n-1} = 2^{1/n}$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for n tasks

- Substituting the value of R_i in U we obtain

$$U_{lub} = (n-1)2^{1/n} + \frac{2}{2^{1-1/n}} - n$$
$$= n2^{1/n} - 2^{1/n} + 2^{1/n} - n$$
$$= n\left(2^{1/n} - 1\right)$$

Department of Distributed and Dependable Systems

Calculation of U_{lub} for n tasks

- For high values of n, the least upper bound converges to

$$\boxed{U_{lub} = \ln 2 \simeq 0.69}$$

- This is proved using substitution $y = \left(2^{1/n} - 1\right)$. From that $n = \frac{\ln 2}{\ln(y+1)}$. Hence

$$\lim_{n\to\infty} n\left(2^{1/n} - 1\right) = (\ln 2) \lim_{y\to 0} \frac{y}{\ln(y+1)}$$

- And using l'Hospital's rule:

$$\lim_{y\to 0} \frac{y}{\ln(y+1)} = \lim_{y\to 0} \frac{1}{1/(y+1)} = \lim_{y\to 0}(y+1) = 1$$

Department of Distributed and Dependable Systems

Hyperbolic Bound

- There is a tighter sufficient condition called hyperbolic bound

- **Theorem:** Let $\Gamma = \{\tau_1, \dots, \tau_n\}$ be a set of n periodic tasks, where each task τ_i is characterized by a processor utilization U_i. Then Γ is schedulable with the RM algorithm if

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

Hyperbolic Bound

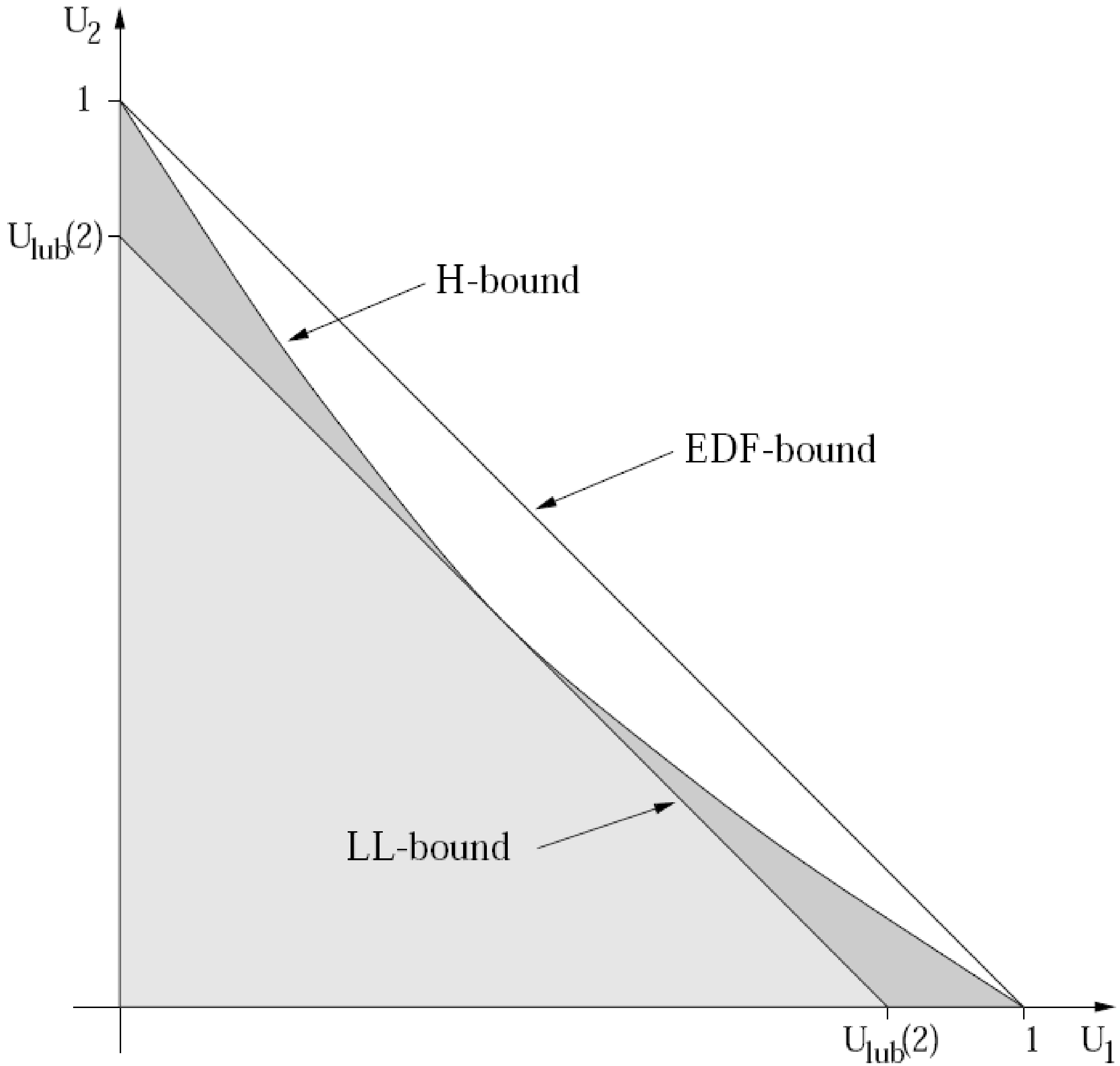


Figure from Buttazzo, G.:Hyperbolic bound for RM