Robotics and Embedded Systems

Technische Universität München

Analysis (I)

Kai Huang

Google to buy Nest Labs for $3.2bn

- Nest Labs is a home automation company that designs and manufactures sensor-driven, Wi-Fi-enabled, self-learning, programmable thermostats and smoke detectors

- The Nest Thermostat allows interaction with the thermostat via spinning and clicking of its control wheel, which brings up option menus for switching from heating to cooling, access to device settings, energy history, and scheduling.

Outline

- Real-Time Model
- Periodic/Aperiodic Tasks

Basic Terms

- Real-time systems
  - Hard: A real-time task is said to be hard, if missing its deadline may cause catastrophic consequences on the environment under control. Examples are sensory data acquisition, detection of critical conditions, actuator serving.
  - Soft: A real-time task is called soft, if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior. Examples are command interpreter of the user interface, displaying messages on the screen.

Schedule

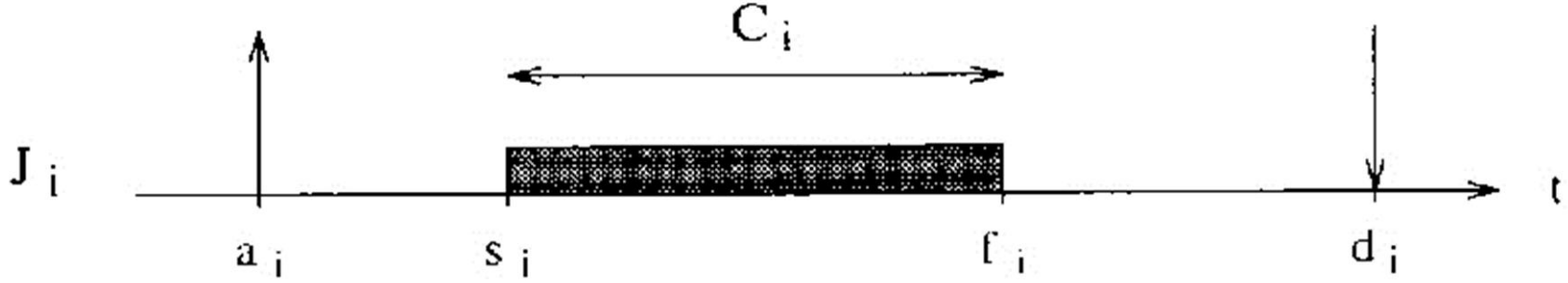
- Given a set of tasks $T = \{T_1, T_2, \dots\}$:
  - A schedule is an assignment of tasks to the processor, such that each task is executed until completion.
  - A schedule can be defined as an integer step function $\sigma: R \rightarrow N$ where $\sigma(t)$ denotes the task which is executed at time t. If $\sigma(t)=0$ then the processor is called idle.
  - If $\sigma(t)$ changes its value at some time, then the processor performs a context switch.
  - Each interval, in which $\sigma(t)$ is constant is called a time slice.
  - A preemptive schedule is a schedule in which the running task can be arbitrarily suspended at any time, to assign the CPU to another task according to a predefined scheduling policy.

Schedule and Timing

- A schedule is said to be feasible, if all task can be completed according to a set of specified constraints.
- A set of tasks is said to be schedulable, if there exists at least one algorithm that can produce a feasible schedule.

- Arrival time a_i or release time r_i is the time at which a task becomes ready for execution.
- Computation time C_i is the time necessary to the processor for executing the task without interruption.
- Deadline d_i is the time at which a task should be completed.
- Start time S_i is the time at which a task starts its execution.
- Finishing time f_i is the time at which a task finishes its execution.

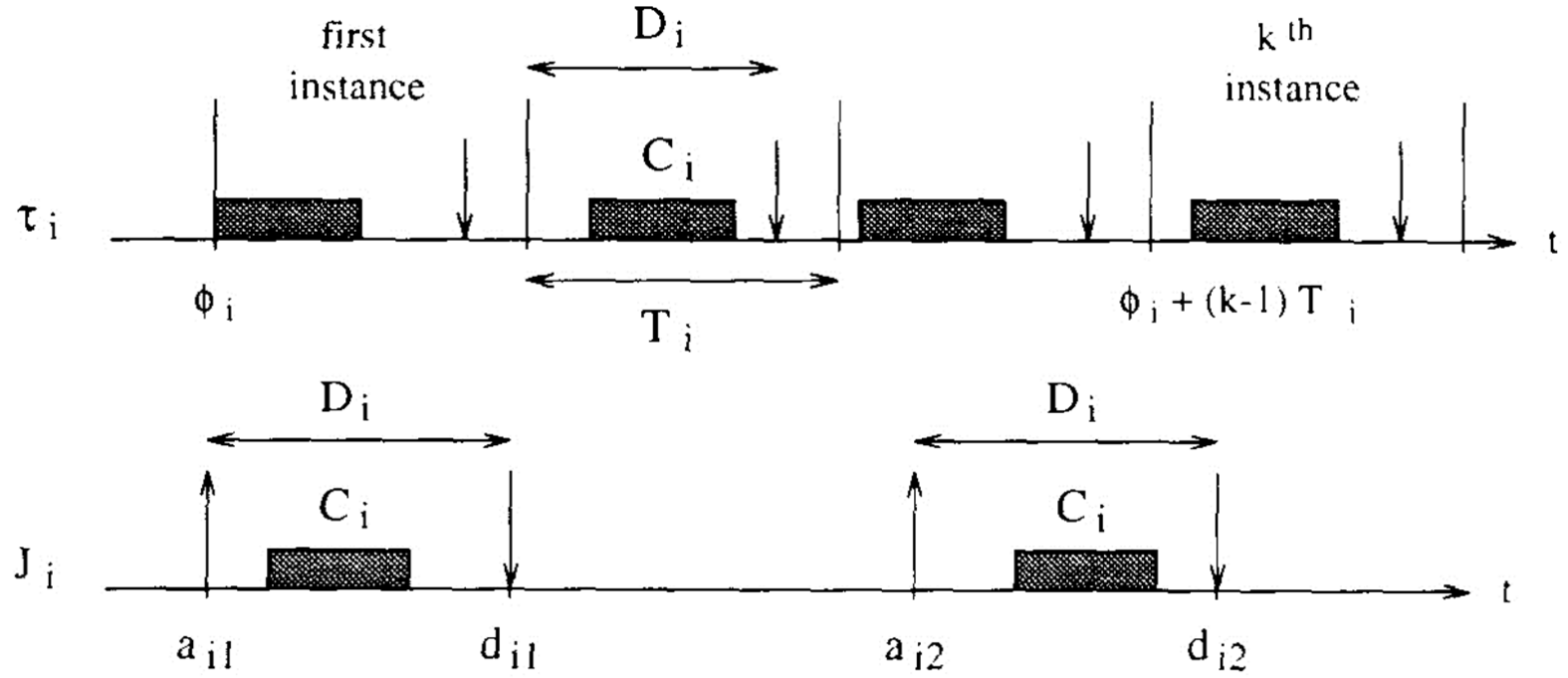
Schedule and Timing



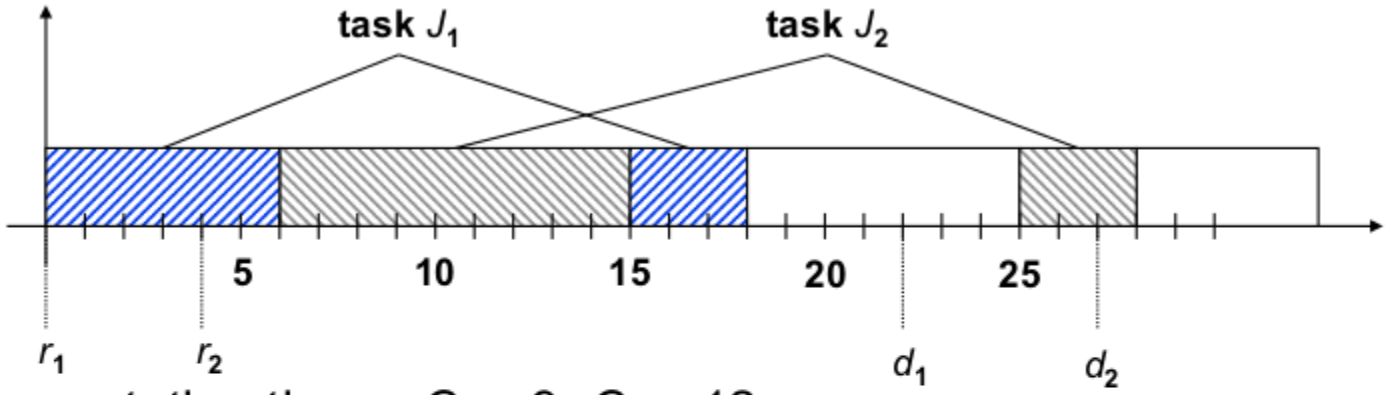
- Using the above definitions, we have $d_i > r_i + C_i$
- Lateness $L_i = f_i - d_i$ represents the delay of a task completion with respect to its deadline; note that if a task completes before the deadline, its lateness is negative.
- Tardiness or exceeding time $E_i = max(0, L_i)$ is the time a task stays active after its deadline.
- Laxity or slack time $X_i = d_i - a_i - C_i$ is the maximum time a task can be delayed on its activation to complete within its deadline.

Schedule and Timing

- Periodic task τ_i : infinite sequence of identical activities, called instances or jobs, that are regularly activated at a constant rate with period T_i . The activation time of the first instance is called phase Φ_i .



Example



Computation times: $C_1 = 9$, $C_2 = 12$

Start times: $s_1 = 0$, $s_2 = 6$

Finishing times: $f_1 = 18$, $f_2 = 28$

Lateness: $L_1 = -4$, $L_2 = 1$

Tardiness: $E_1 = 0$, $E_2 = 1$

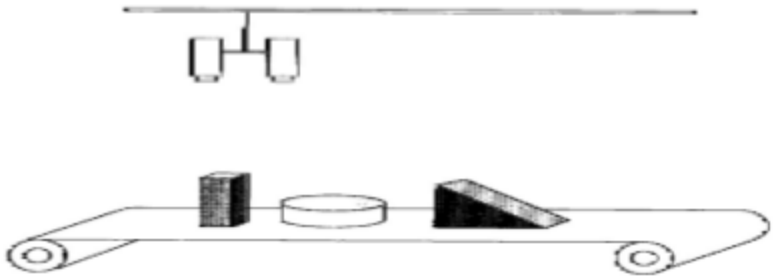
Laxity: $X_1 = 13$, $X_2 = 11$

Precedence Constraints

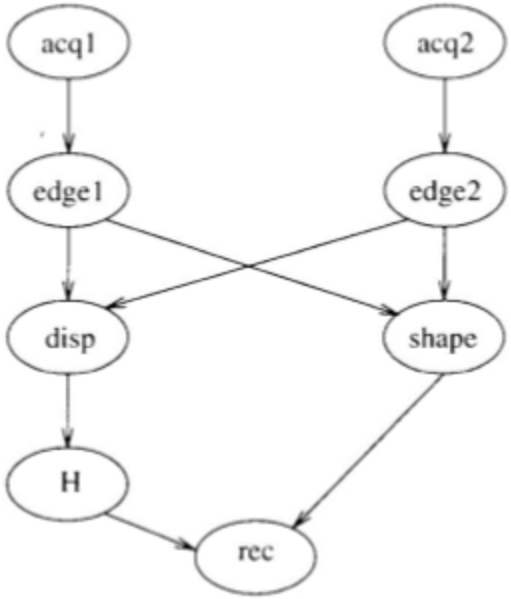
- Precedence relations between graphs can be described through an acyclic directed graph G where tasks are represented by nodes and precedence relations by arrows. G induces a partial order on the task set.
- There are different interpretations possible:
  - All successors of a task are activated (concurrent task execution).
  - One successor of a task is activated (non-deterministic choice).

Precedence Constraints - Example

- Example (concurrent activation):



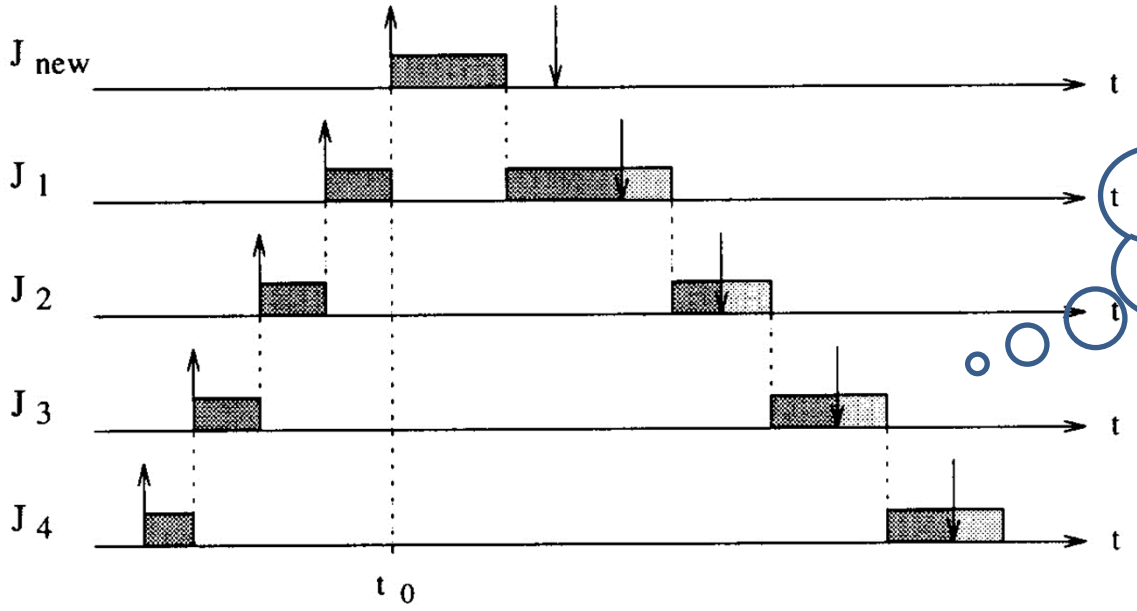
- Image acquisition *acq1* *acq2*
- Low level image processing *edge1* *edge2*
- Feature/contour extraction *shape*
- Pixel disparities *disp*
- Object size *H*
- Object recognition *rec*



Classification of Scheduling Algorithms

- With preemptive algorithms, the running task can be interrupted at any time to assign the processor to another active task, according to a predefined scheduling policy.
- With a non-preemptive algorithm, a task, once started, is executed by the processor until completion.
- Static algorithms are those in which scheduling decisions are based on fixed parameters, assigned to tasks before their activation.
- Dynamic algorithms are those in which scheduling decisions are based on dynamic parameters that may change during system execution.

Classification of Scheduling Algorithms



Domino effect, if acceptance test wrongly accepted a new task.

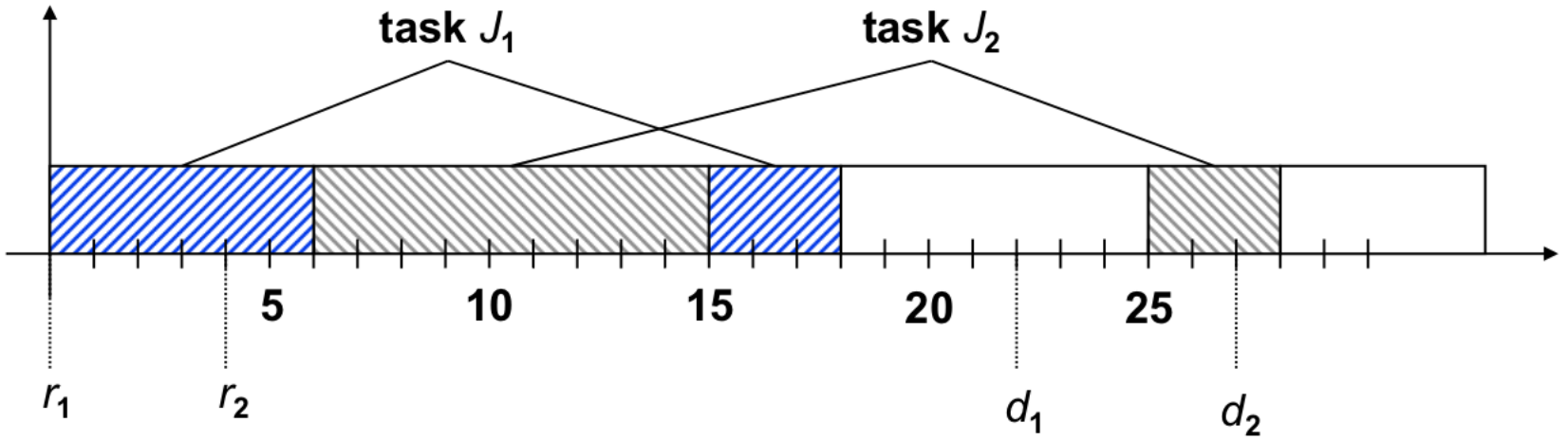
- An algorithm is said optimal if it minimizes some given cost function defined over the task set.
- An algorithm is said to be heuristic if it tends toward but does not guarantee to find the optimal schedule.

Metrics

- Average response time: $\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n}(f_i - r_i)$
- Total completion time: $t_c = \max_i(f_i) - \min_i(r_i)$
- Weighted sum of response time: $t_w = \frac{\sum_{i=1}^{n} w_i(f_i - r_i)}{\sum_{i=1}^{n} w_i}$
- Maximum lateness: $L_{\max} = \max_i(f_i - d_i)$
- Number of late tasks: $N_{\text{late}} = \sum_{i=1}^{n} miss(f_i)$

$$miss(f_i) = \begin{cases} 0 & \text{if } f_i \le d_i \\ 1 & \text{otherwise} \end{cases}$$

Metrics Example



Average response time: $\overline{t_r} = \frac{1}{2}(18+24) = 21$

Total completion time: $t_c = 28 - 0 = 28$

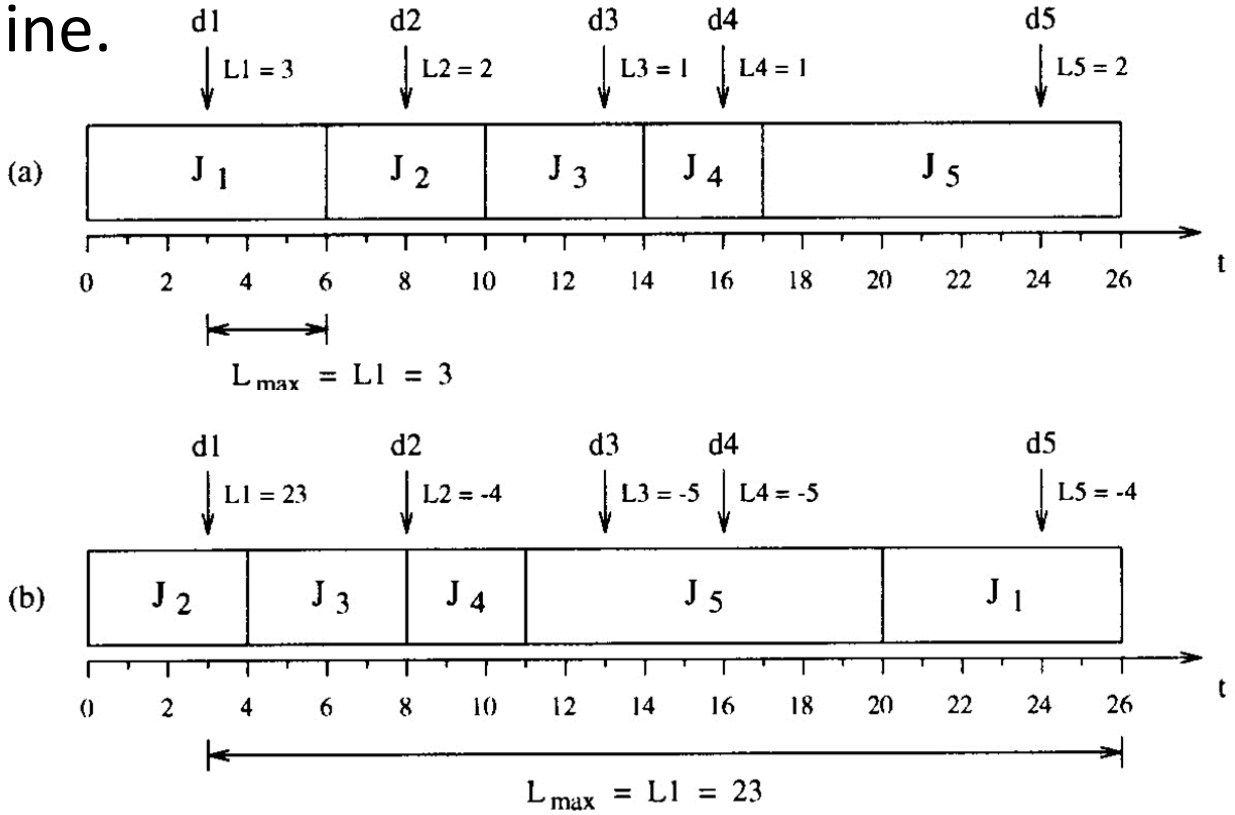
Weighted sum of response times: $w_1 = 2, w_2 = 1: \; t_w = \frac{2 \cdot 18 + 24}{3} = 20$

Number of late tasks: $N_{\text{late}} = 1$

Maximum lateness: $L_{\max} = 1$

Scheduling Example

- In (a), the maximum lateness is minimized, but all tasks miss their deadlines.
- In (b), the maximal lateness is larger, but only one task misses its deadline.



Outline

- Real-Time Model
- Classical real-time scheduling for Periodic Tasks
  - Periodic Rate Monotonic (RM)
  - Earliest Deadline First (EDF)

Model of Periodic Tasks

- Examples: sensory data acquisition, low-level servoing, control loops, action planning and system monitoring. When a control application consists of several concurrent periodic tasks with individual timing constraints, the OS has to guarantee that each periodic instance is regularly activated at its proper rate and is completed within its deadline.

- Definitions:

  T : denotes a set of periodic tasks

  τ_i : denotes a generic periodic task

  $\tau_{i,j}$: denotes the *jth* instance of task i

  $r_{i,j}$, $s_{i,j}$, $f_{i,j}$, $d_{i,j}$:
  denotes the release time, start time, finishing time, absolute deadline of the *jth* instance of task i

  Φ_i : phase of task i (release time of its first instance)

  D_i : relative deadline of task i

Model of Periodic Tasks

- The following hypotheses are assumed on the tasks:
  - The instances of a periodic task are regularly activated at a constant rate. The interval Ti between two consecutive activations is called period. The release times satisfy

$$r_{i,j} = \Phi_i + (j\text{-}1)\, T_i$$

- All instances have the same worst case execution time C_i
- All instances of a periodic task have the same relative deadline D_i. Therefore, the absolute deadlines satisfy

$$d_{i,j} = \Phi_i + (j\text{-}1)T_i + D_i$$

Model of Periodic Tasks

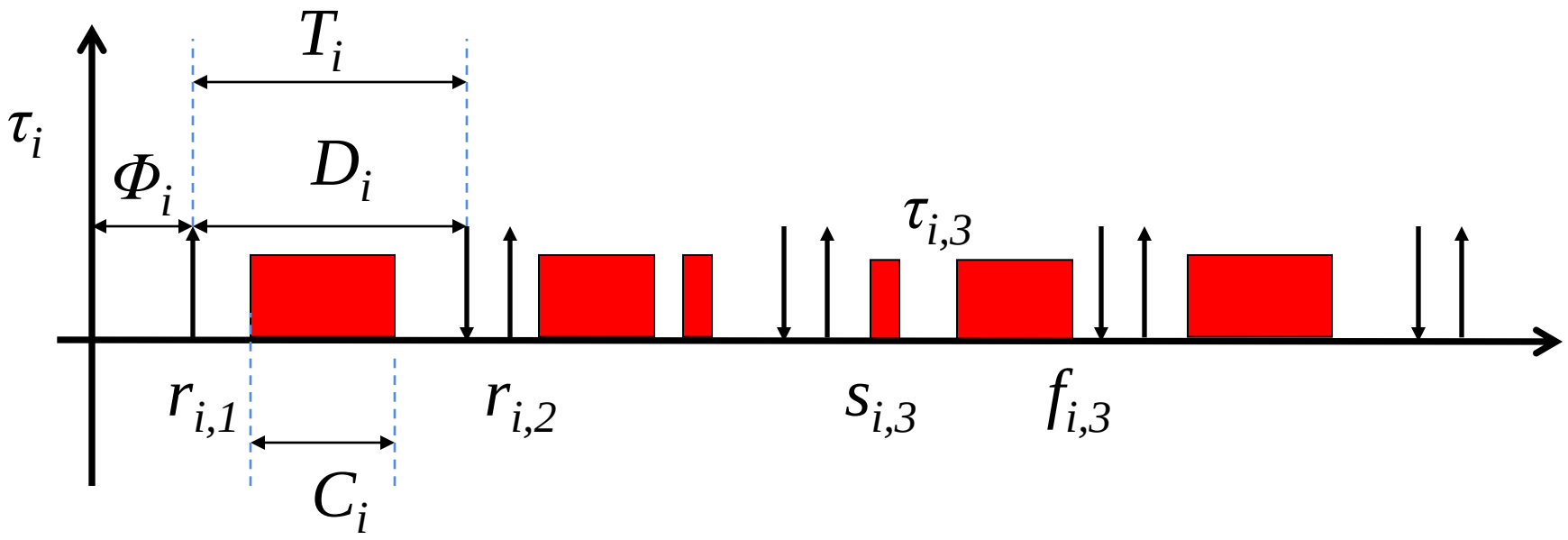
- The following hypotheses are assumed on the tasks cont':
  - Often, the relative deadline equals the period $T_i + D_i$ and therefore

$$d_{i,j} = \Phi_i + jT_i$$

  - All periodic tasks are independent; that is, there are no precedence relations and no resource constraints.
  - No task can suspend itself, for example on I/O operations.
  - All tasks are released as soon as they arrive.
  - All overheads in the OS kernel are assumed to be zero.

Model of Periodic Tasks

- Example:



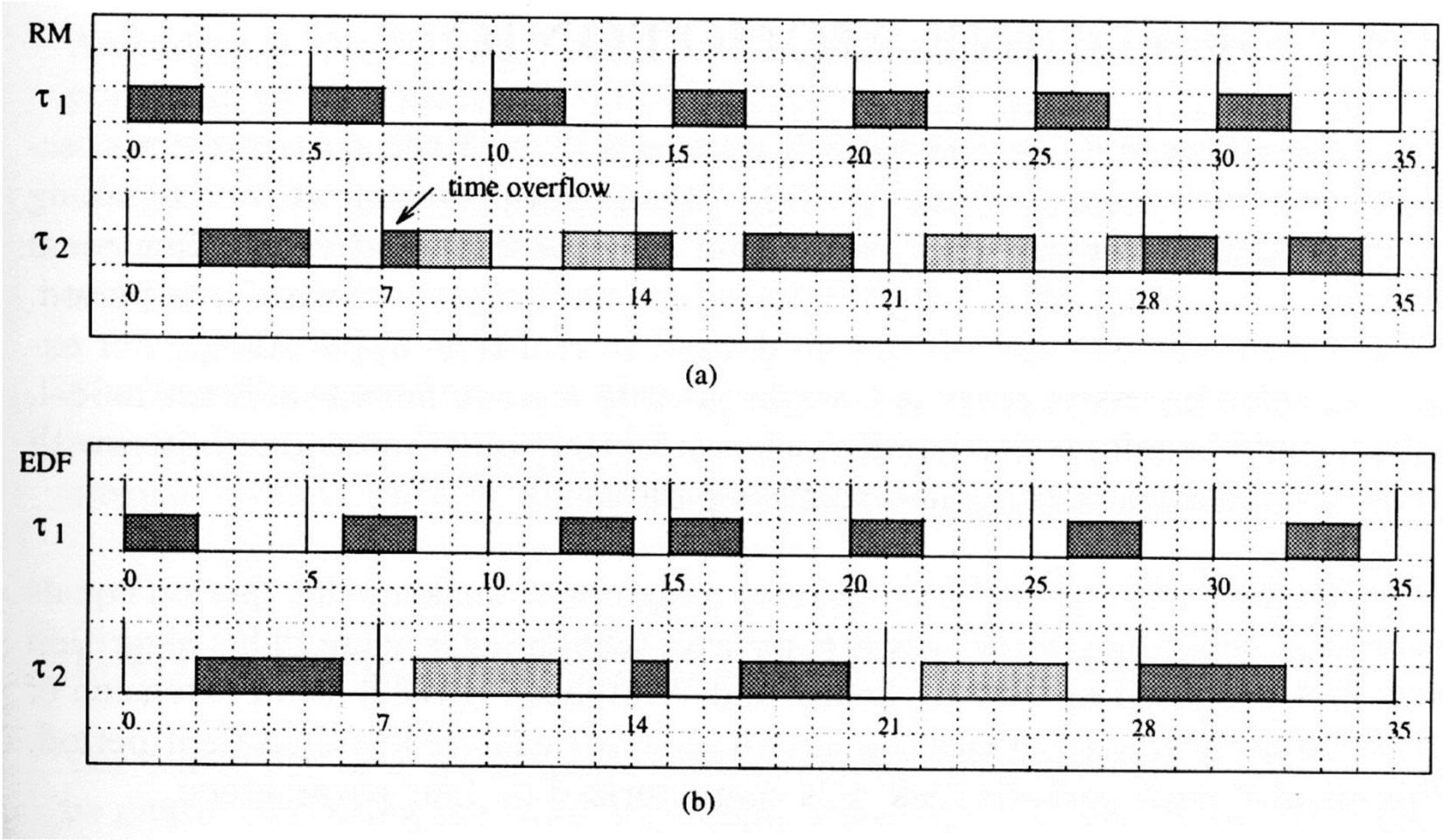
Rate Monotonic Scheduling (RM)

- Assumptions:
  - Task priorities are assigned to tasks before execution and do not change over time (static priority assignment).
  - RM is intrinsically preemptive: the currently executing task is preempted by a task with higher priority.
  - Deadlines equal the periods $T_i = D_i$

- Algorithm:
  - Each task is assigned a priority. Tasks with higher request rates (that is with shorter periods) will have higher priorities. Tasks with higher priority interrupt tasks with lower priority.

Periodic Tasks

- Example: 2 tasks, deadline = periods, U = 97%

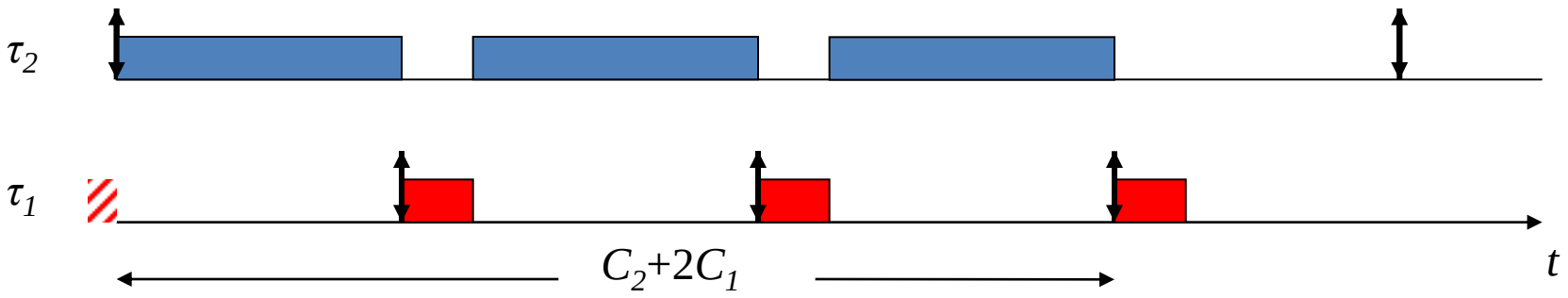


Rate Monotonic Scheduling (RM)

- Optimality: RM is optimal among all fixed-priority assignments in the sense that not other fixed-priority algorithm can schedule a task set that cannot be scheduled by RM.
- The proof is done by considering several cases that may occur, but the main ideas are as follows:
  - A critical instant for any task occurs whenever the task is released simultaneously with all higher priority tasks. The tasks schedulability can easily be checked at their critical instances. If all tasks are feasible at their critical instants, then the task set is schedulable in any other condition.
  - Show that, given two periodic tasks, if the schedule is feasible by an arbitrary priority assignment, then it is also feasible by RM.
- Extend the result to a set of n periodic tasks.

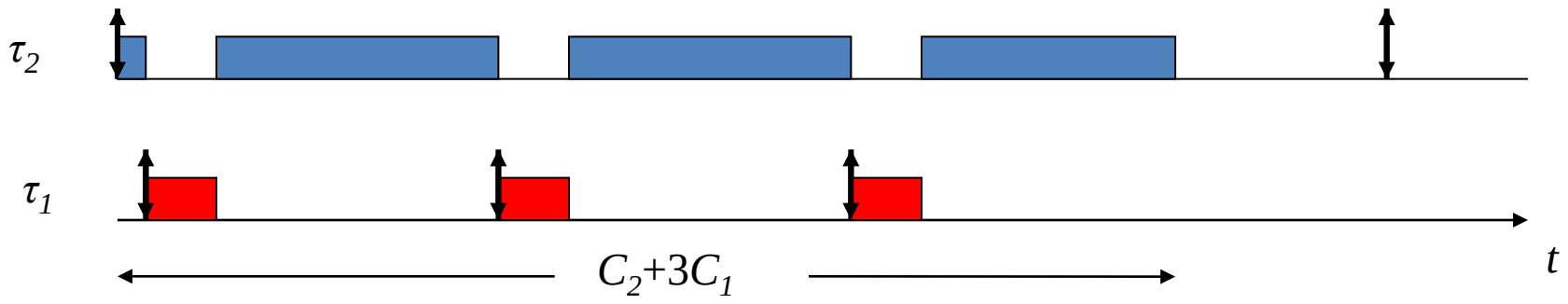
Proof of Critical Instance

- Definition: A critical instant of a task is the time at which the release of a task will produce the largest response time.
- Lemma: For any task, the critical instant occurs if that task is simultaneously released with all higher priority tasks.
- Proof sketch: Start with 2 tasks τ_1 and τ_2. Response time of τ_2 is delayed by tasks τ_1 of higher priority:



Proof of Critical Instance

- Delay may increase if τ_1 starts earlier:

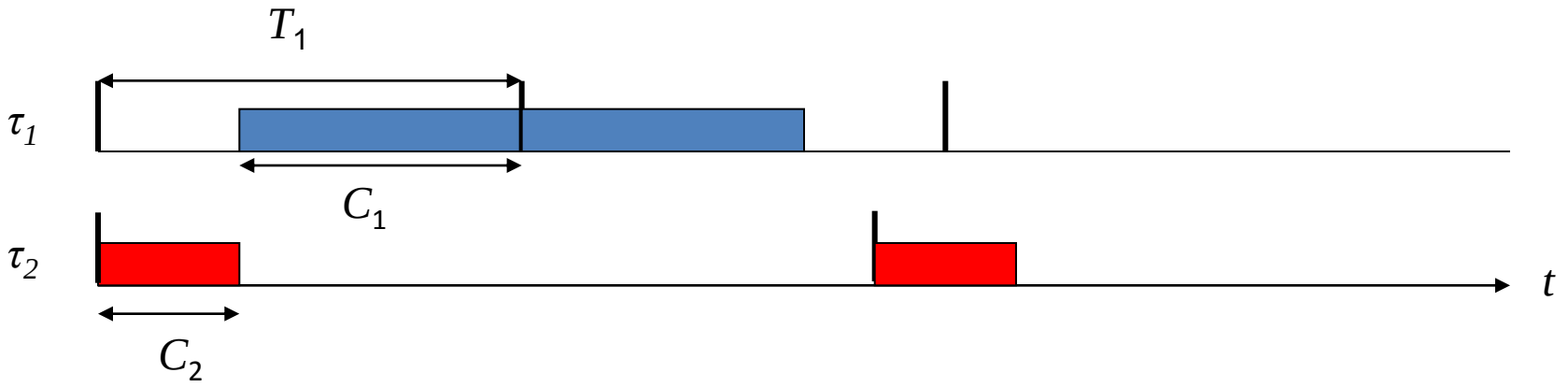


- Maximum delay achieved if τ_1 and τ_1 start simultaneously.
- Repeating the argument for all higher priority tasks of some task τ_2 :

> The worst case response time of a task occurs when it is released simultaneously with all higher-priority tasks.

Proof of RM Optimality (2 Tasks)

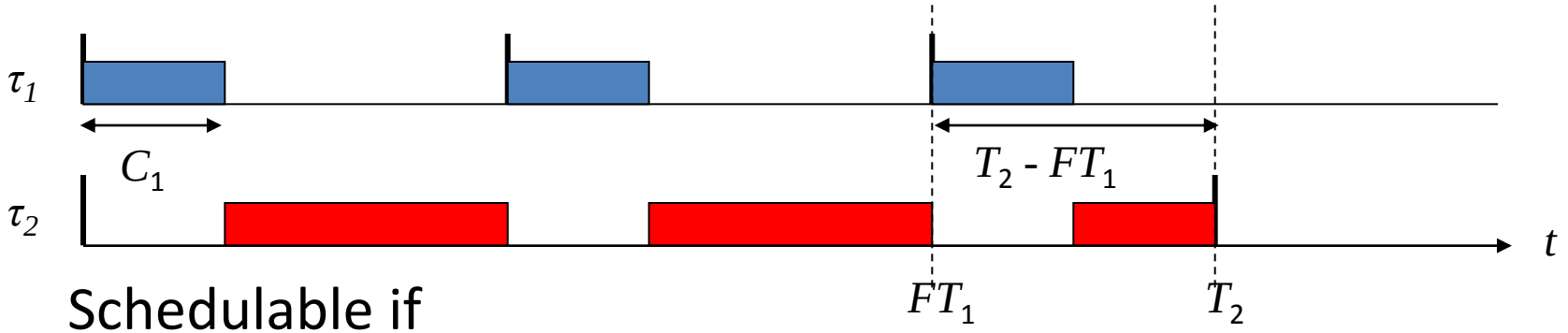
- We have two tasks τ_1, τ_2 with periods $T_1 < T_2$
- Define $F = \lfloor T_2/T_1 \rfloor$: number of periods of τ_1 fully contained in T_2
- Consider tow case A and B:
- A: Assume RM is not used → prio(τ_2) is highest:



- Schedule is feasible if $C_2 + C_1 < T_1$ (A)

Proof of RM Optimality (2 Tasks)

- B: Assume RM is used → prio(τ_1) is highest



Schedulable if

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \le T_2 \text{ and } C_1 \le T_1 \qquad \text{(B)}$$

- We need to show that (A)$\Rightarrow$(B): $C_1+C_2 \le T_1 \Rightarrow C_1 \le T_1$

$$C_1+C_2 \le T_1 \Rightarrow FC_1 + C_2 \le FC_1 + FC_2 \le FT_1 \Rightarrow$$
$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \le FT_1 + \min(T_2 - FT_1, C_1)$$
$$\le \min(T_2 , C_1 + FT_1) \le T_2$$

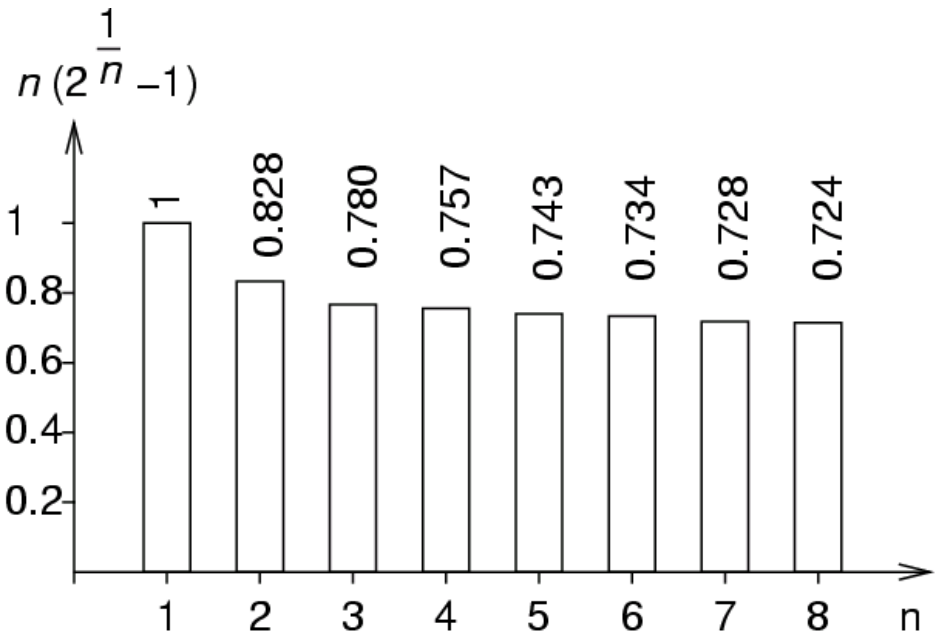
Given tasks τ_1 and τ_2 with $T_1 < T_2$, then if the schedule is feasible by an arbitrary fixed priority assignment, it is also feasible by RM.

Rate Monotonic Scheduling (RM)

- Schedulability analysis: A set of periodic tasks is schedulable with RM if

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

This condition is sufficient but not necessary.



- The term $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$

denotes the processor utilization factor U which is the fraction of processor time spent in the execution of the task set.

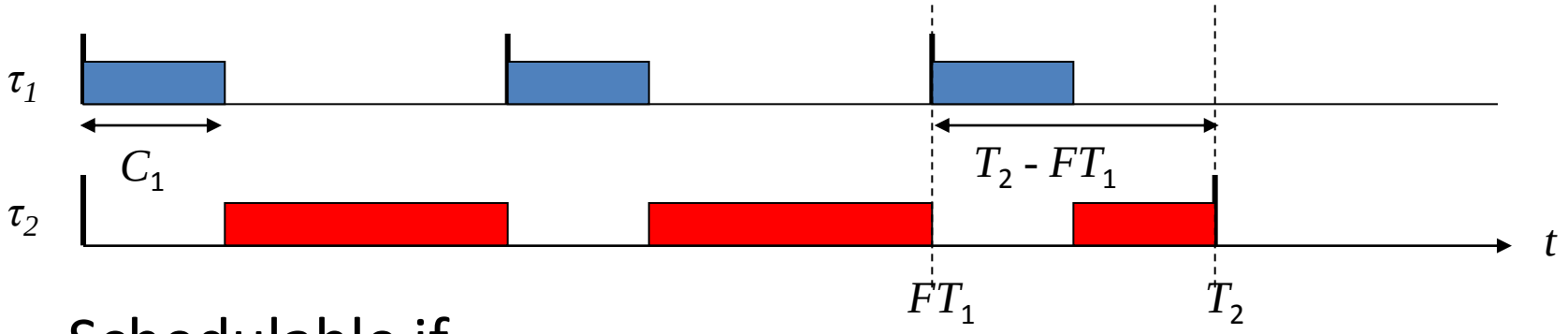
Proof of Utilization Bound (2 Tasks)

- We have two tasks τ_1, τ_2 with periods $T_1 < T_2$. Define $F = \lfloor T_2/T_1 \rfloor$: number of periods of τ_1 fully contained in T_2

- Proof procedure: Compute upper bound on utilization U
  - assign priorities according to RM;
  - compute upper bound U_{up} by setting computation times to fully utilize processor (C_2 adjusted to fully utilize processor);
  - minimize upper bound with respect to other task parameters.

Proof of Utilization Bound (2 Tasks)

- As before:



Schedulable if

$$FC_1 + C_2 + \min(T_2 - FT_1, C_1) \le T_2 \text{ and } C_1 \le T_1$$

- Utilization:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - FC_1 - \min\{T_2 - FT_1, C_1\}}{p_2}$$

$$= 1 + \frac{C_1(T_2 - FT_1) - T_2 \min\{T_2 - FT_1, C_1\}}{T_2 T_1}$$

Proof of Utilization Bound (2 Tasks)

- Minimize utilization bound w.r.t. C_1 :
  - If $C_1 \leq T_2 - FT_1$ then U decreases with increasing C_1
  - If $T_2 - FT_1 \leq C_1$ then U decreases with decreasing C_1
  - Therefore, minimum U is obtained with $C_1 = T_2 - FT_1$:

$$U = 1 + \frac{(T_2 - FT_1)^2 - T_1(T_2 - FT_1)}{T_2 T_1}$$

$$= 1 + \frac{T_1}{T_2}\left(\left(\frac{T_2}{T_1} - F\right)^2 - \left(\frac{T_2}{T_1} - F\right)\right)$$

- We now need to minimize w.r.t. $G = T_2 / T_1$ where $F = \lfloor T_2/T_1 \rfloor$ and $T_1 \leq T_2$. As F is integer, we first suppose that it is independent of $G = T_2 / T_1$. We obtain

Proof of Utilization Bound (2 Tasks)

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

- Minimizing U with respect to G yields

$$2G(G-F) - (G-F)^2 - F = G^2 - (F^2 + F) = 0$$

- If we set F = 1, then we obtain

$$G = \frac{T_2}{T_1} = \sqrt{2}$$

$$U = 2(\sqrt{2} - 1)$$

- It can easily be checked, that all other integer values for F lead to a larger upper bound on the utilization.

Outline

- Real-Time Model
- Classical real-time scheduling for Periodic Tasks
  - Periodic Rate Monotonic (RM)
  - Earliest Deadline First (EDF)

EDF Scheduling (earliest deadline first)

- Assumptions:
  - dynamic priority assignment
  - intrinsically preemptive
  - $D_i \leq T_i$
- Algorithm: The currently executing task is preempted whenever another periodic instance with earlier deadline becomes active.

$$d_{i,j} = \Phi_i + (j-1)T_i + D_i$$

- Optimality: No other algorithm can schedule a set of periodic tasks if the set that can not be scheduled by EDF.
- The proof is simple and follows that of the aperiodic case.

EDF Scheduling

- A necessary and sufficient schedulability test if $D_i = T_i$:
  - A set of periodic tasks is schedulable with EDF if and only if

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

- The term

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

denotes the average processor utilization.

EDF Scheduling

- If the utilization satisfies $U > 1$, then there is no valid schedule: The total demand of computation time in interval $T = T_1 \cdot T_2 \cdot \ldots \cdot T_n$ is

$$\sum_{i=1}^{n} \frac{C_i}{T_i} T = UT > T$$

  and therefore, it exceeds the available processor time.

- If the utilization satisfies $U \leq 1$, then there is a valid schedule.
  - We will proof this by contradiction: Assume that deadline is missed at some time t_2. Then we will show that the utilization was larger than 1.

EDF Scheduling

- If the deadline was missed at t_2 then define t_1 as the maximal time before t_2 where
  - the processor is continuously busy in [t_1, t_2] and
  - the processor only executes tasks that have their arrival time AND deadline in [t_1, t_2] .
- Why does such a time t_1 exist?
  - We find such a t_1 by starting at t_2 and going backwards in time, always ensuring that the processor only executed tasks that have their deadline before or at t_2 :
    - Because of EDF, the processor will be busy shortly before t_2 and it executes on the task that has deadline at t_2.
    - Suppose that we reach a time when the processor gets idle, then we found t_1 : There is a task arrival at t_1 and the task queue is empty shortly before.
    - Suppose that we reach a time such that shortly before the processor works on a task with deadline after t_2, then we also found t_1 : Because of EDF, all tasks the processor processed in [t_1, t_2] arrived at or after t_1 (otherwise, the processor would not have operated before t_1 on a task with deadline after t_2).

EDF Scheduling

- Within the interval $[t_1, t_2]$ the total computation time demanded by the periodic tasks is bounded by

$$C_p(t_1, t_2) = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U$$

number of complete periods of task I in the interval

- Since the deadline at time t2 is missed, we must have:

$$t_2 - t_1 < C_p(t_1, t_2) \leq (t_2 - t_1)U \Rightarrow U > 1$$

Periodic Tasks

- Example: 2 tasks, deadline = periods, U = 97%