Scheduling in real-time systems

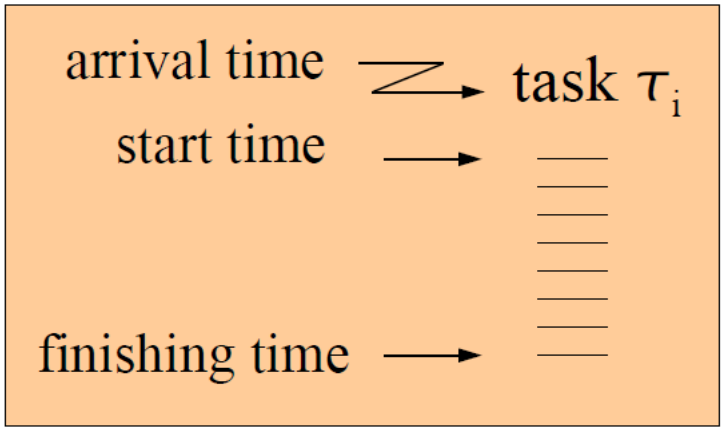
What is a task

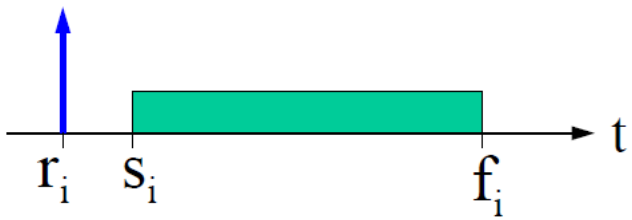
Task

A sequence of instructions that in absence of other activities is continuously executed by the processor until completion.

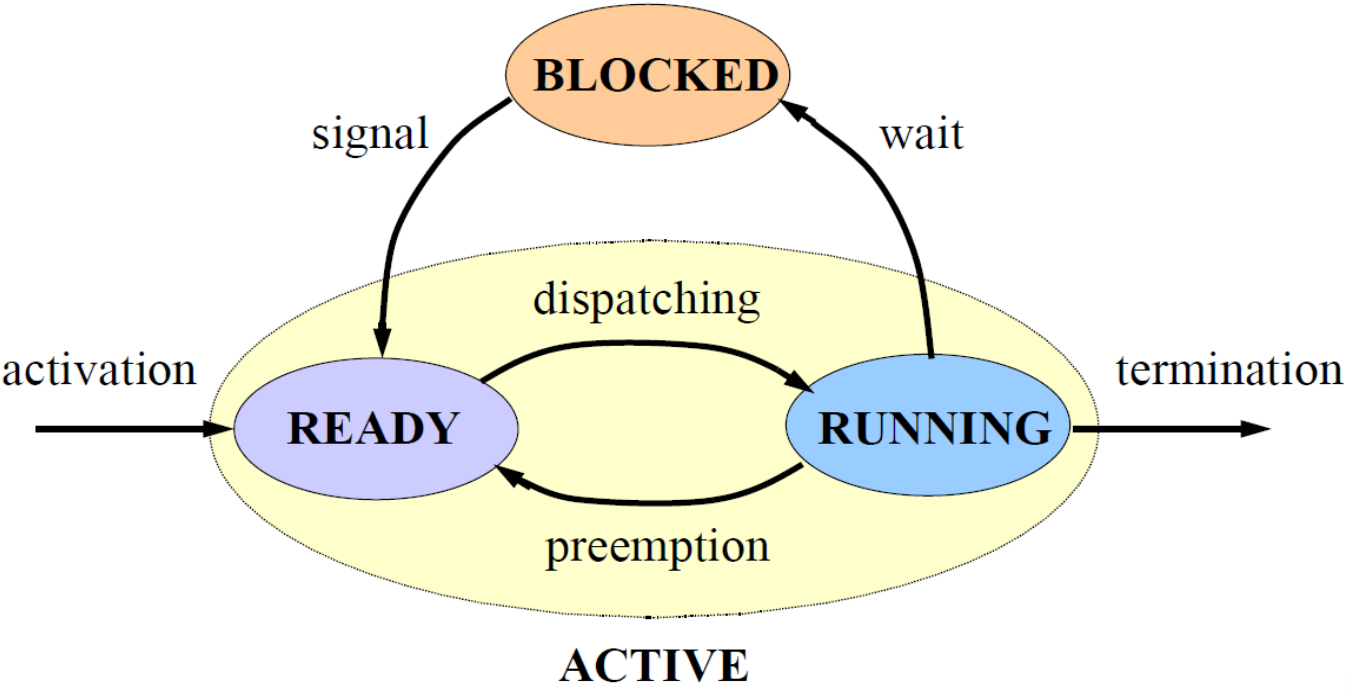
It can be a process or a thread depending on the operating system

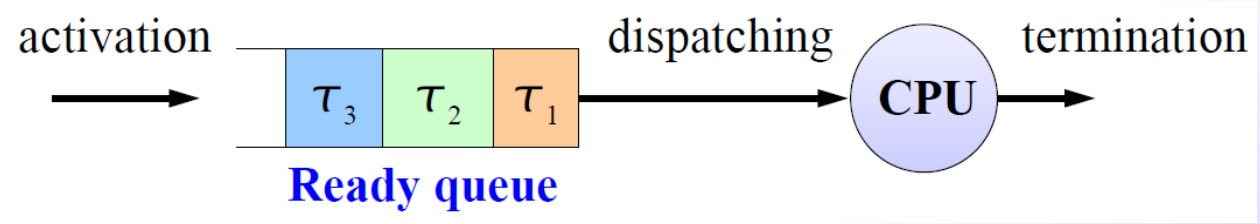
A task is an (infinite?) sequence of instances (jobs)





Task: state transition and ready queue





The ready tasks are kept in a waiting queue, called the ready queue

The strategy for choosing the ready task to be executed on the CPU is the scheduling algorithm

Scheduling

A scheduling algorithm is said to be:

- **preemptive**: if the running task can be temporarily suspended to execute a more important task
- **non preemptive**: if the running task cannot be suspended until completion

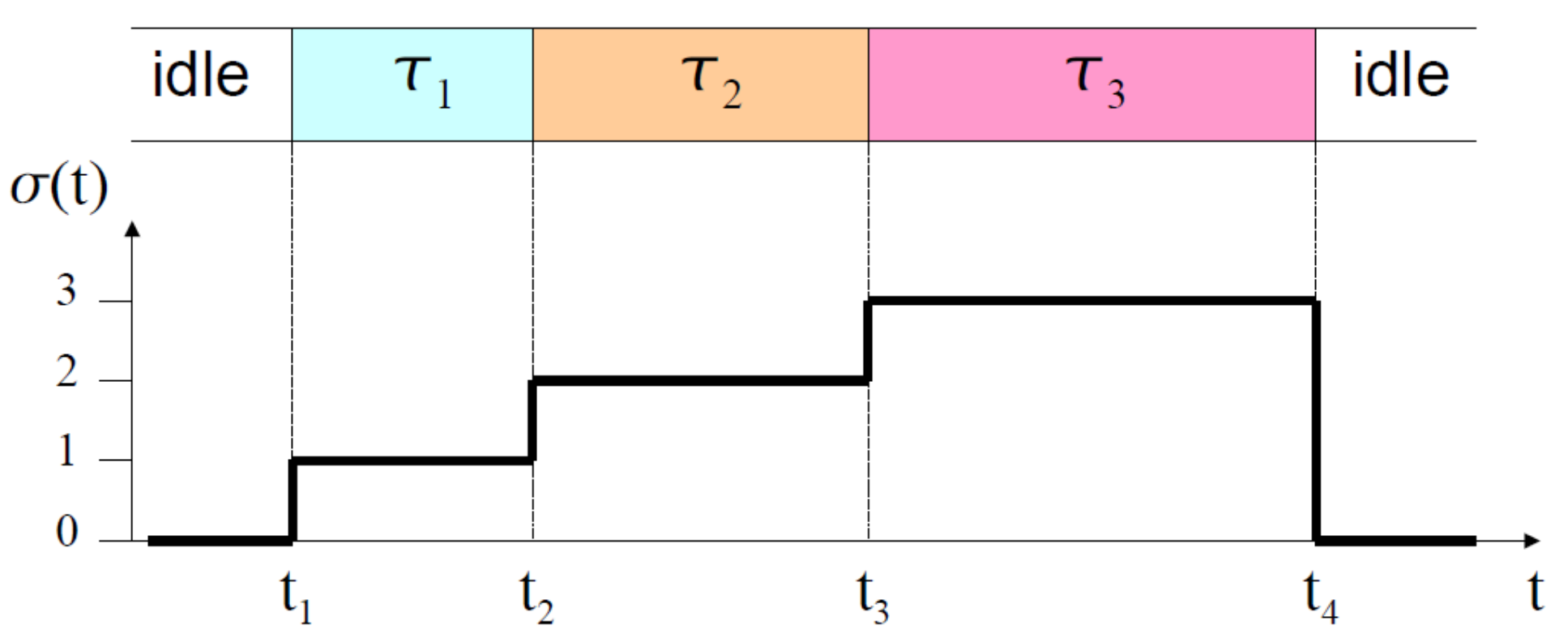
A schedule is a particular assignment of tasks to the processor

given a task set $\Gamma=\{ \tau_1, \tau_2, \tau_3, \dots , \tau_n\}$, a schedule is a mapping $\sigma : R^+ \rightarrow N$ such that $\forall t \in R^+$

$$\sigma(t) = \begin{cases} k > 0 & \text{if } \tau_k \text{ is running} \\ 0 & \text{if the processor is idle} \end{cases}$$

A sample …

At time t_1, t_2, t_3, and t_4 a **context switch** is performed.
Each interval $[t_i, t_{i+1})$ is called a time slice.



Real-time tasks

r_i request time (arrival time a_i)

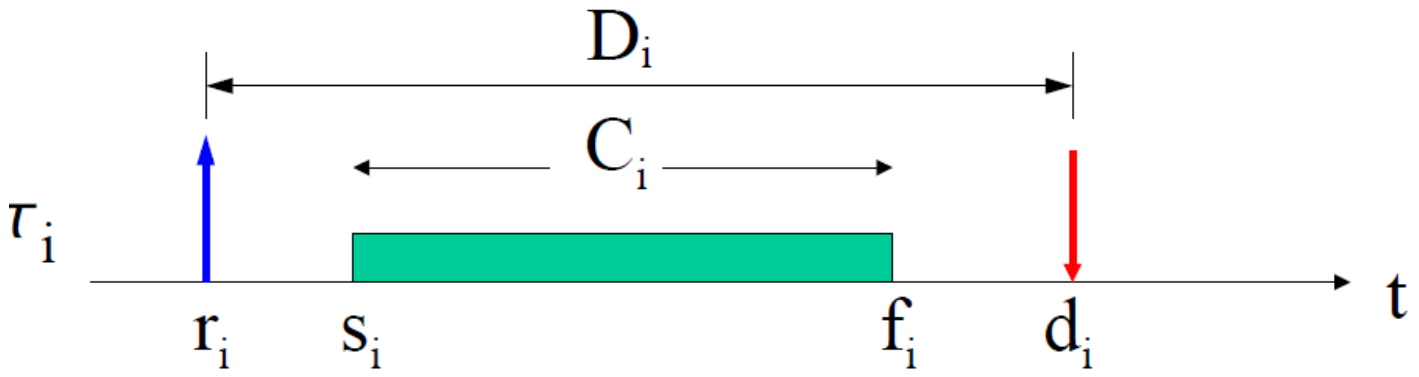
s_i start time

C_i wcet

d_i absolute deadline

D_i relative deadline

f_i finishing time



Task criticality

HARD tasks: all jobs must meet their deadlines: missing a deadline may have serious consequences.

FIRM tasks: only some jobs are allowed to miss their deadline.

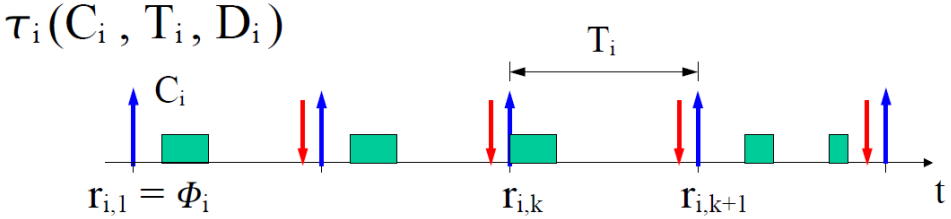
SOFT tasks: jobs may miss deadlines. The goal is to minimize responsiveness.

Tasks activation modes

time driven → periodic tasks

the task is automatically activated by the kernel at regular intervals.

$$\begin{cases} r_{i1} = \Phi_i \\ r_{i,k+1} = r_{i,k} + T_i \end{cases}$$



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

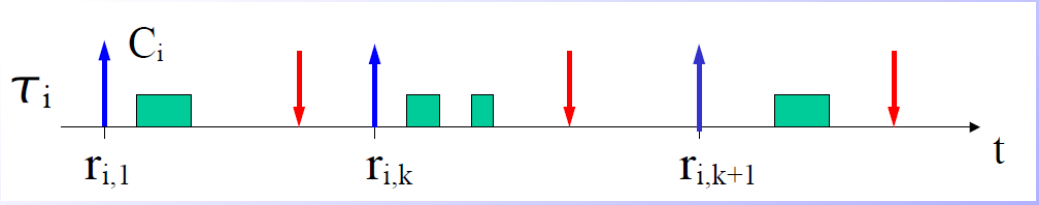
often $D_i = T_i$

event driven → aperiodic tasks

the task is activated upon the arrival of an event or through an explicit invocation of the activation primitive.

aperiodic: $r_{i,k+1} > r_{i,k}$

sporadic: $r_{i,k+1} \geq r_{i,k} + T_i$



Types of constraints

timing constraints

- deadline, activation, completion, jitter
- can be explicit (are included in the specification of the system activities) or implicit (do not appear in the system specification, but must be respected to meet the requirements).

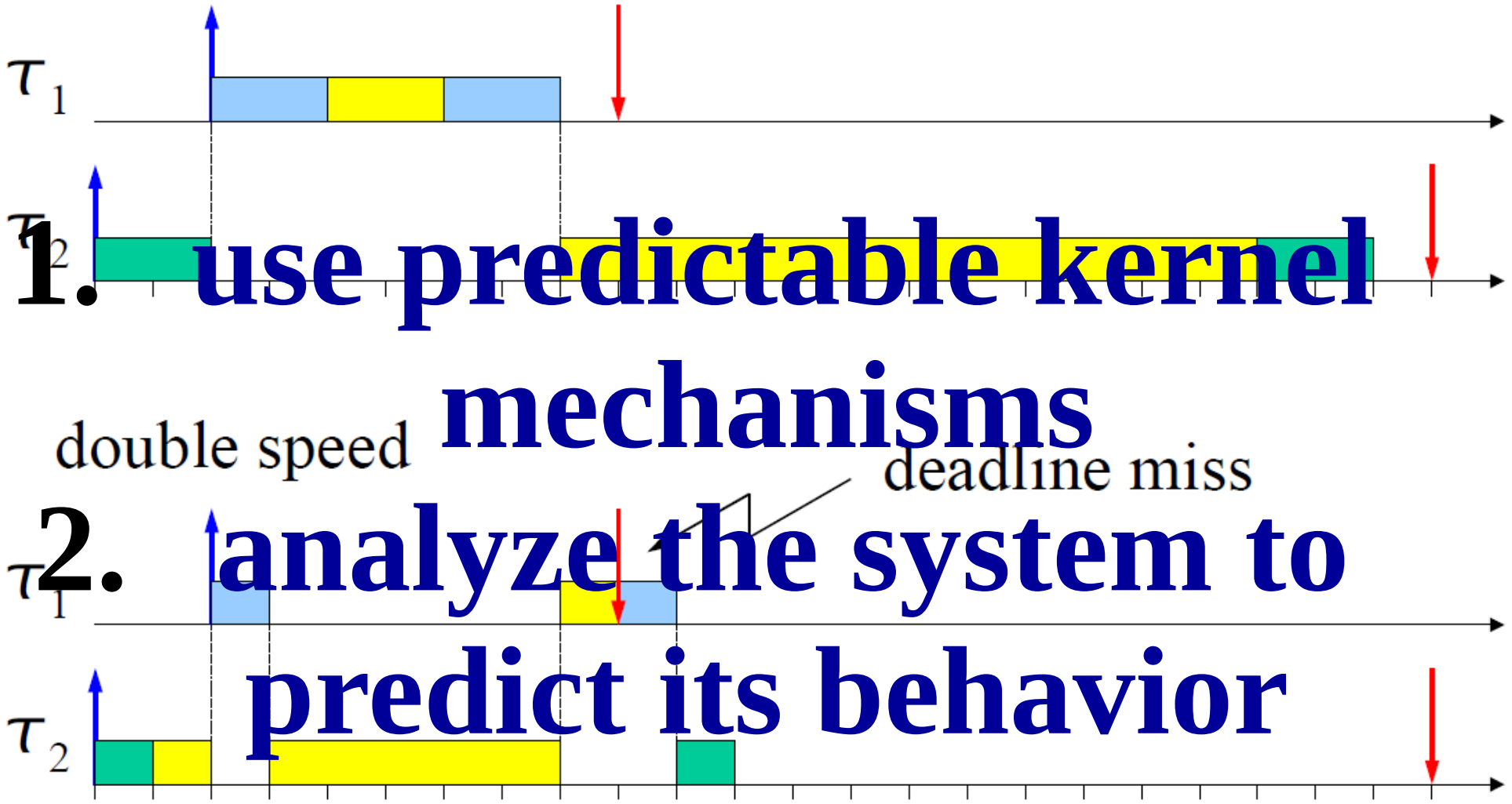
precedence constraints

- they impose an ordering in the execution (sometimes tasks must be executed with specific precedence relations, specified by a Directed Acyclic Graph)

resource constraints

- they enforce a synchronization in the access of mutually exclusive resources.

Scheduling anomalies and lesson learnt



1. use predictable kernel mechanisms
2. analyze the system to predict its behavior

We show some scheduling algorithms…

Aperiodic scheduling

- Earliest Due Date
- Earliest Deadline First

Periodic scheduling

- Cycle Executive (Timeline scheduling)
- Rate Monotonic

Note that several others exist but are not considered here.

Earliest Due Date (EDD)

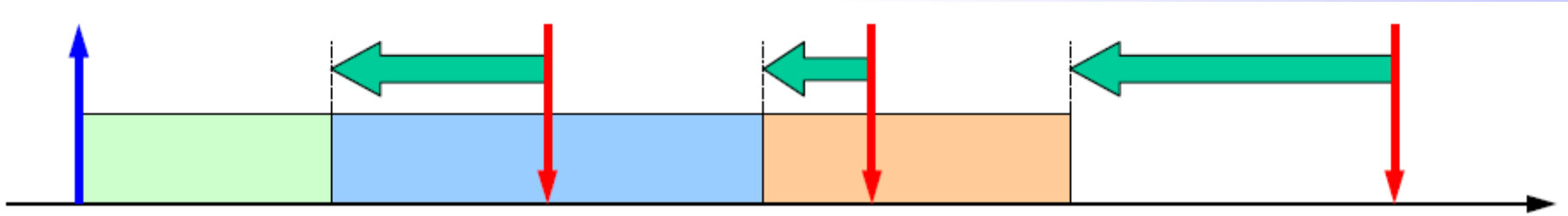
It selects the task with the earliest relative deadline.

Assumptions:

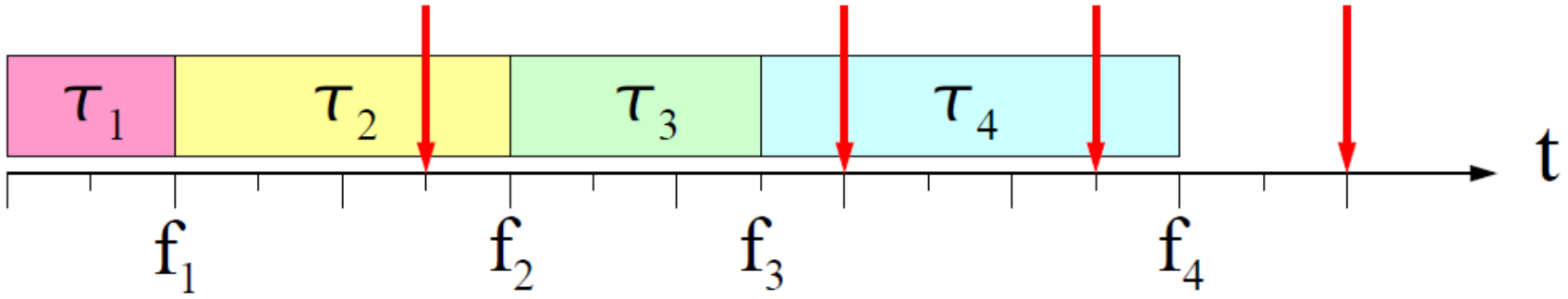
- all tasks arrive simultaneously
- fixed priority (D_i is known in advance)
- preemption is not an issue

It minimizes the maximum lateness (L_{max})

- Lateness L_i = finishing time f_i - absolute deadline d_i
- $L_{max} = \max_i (L_i)$
- If $L_{max} < 0$ then no task miss its deadline



EDD - guarantee test (off line)



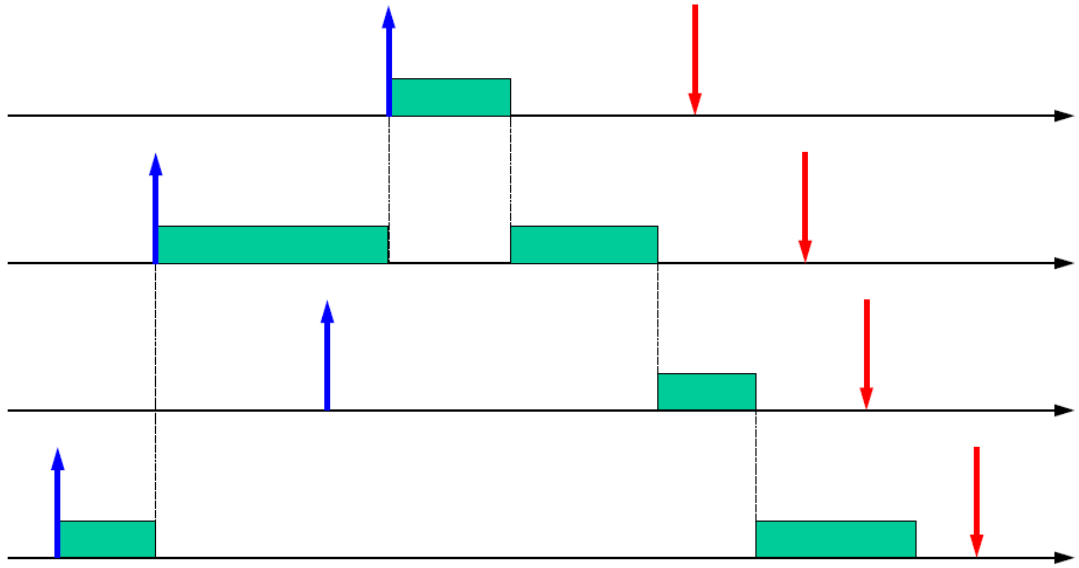
a task set Γ is feasible if $\forall i \quad f_i \leqslant d_i$

$$f_i = \sum_{k=1}^{i} C_k$$

$$\forall i \quad \sum_{k=1}^{i} C_k \leqslant D_i$$

Earliest Deadline First (EDF)

It selects the task with the earliest absolute deadline

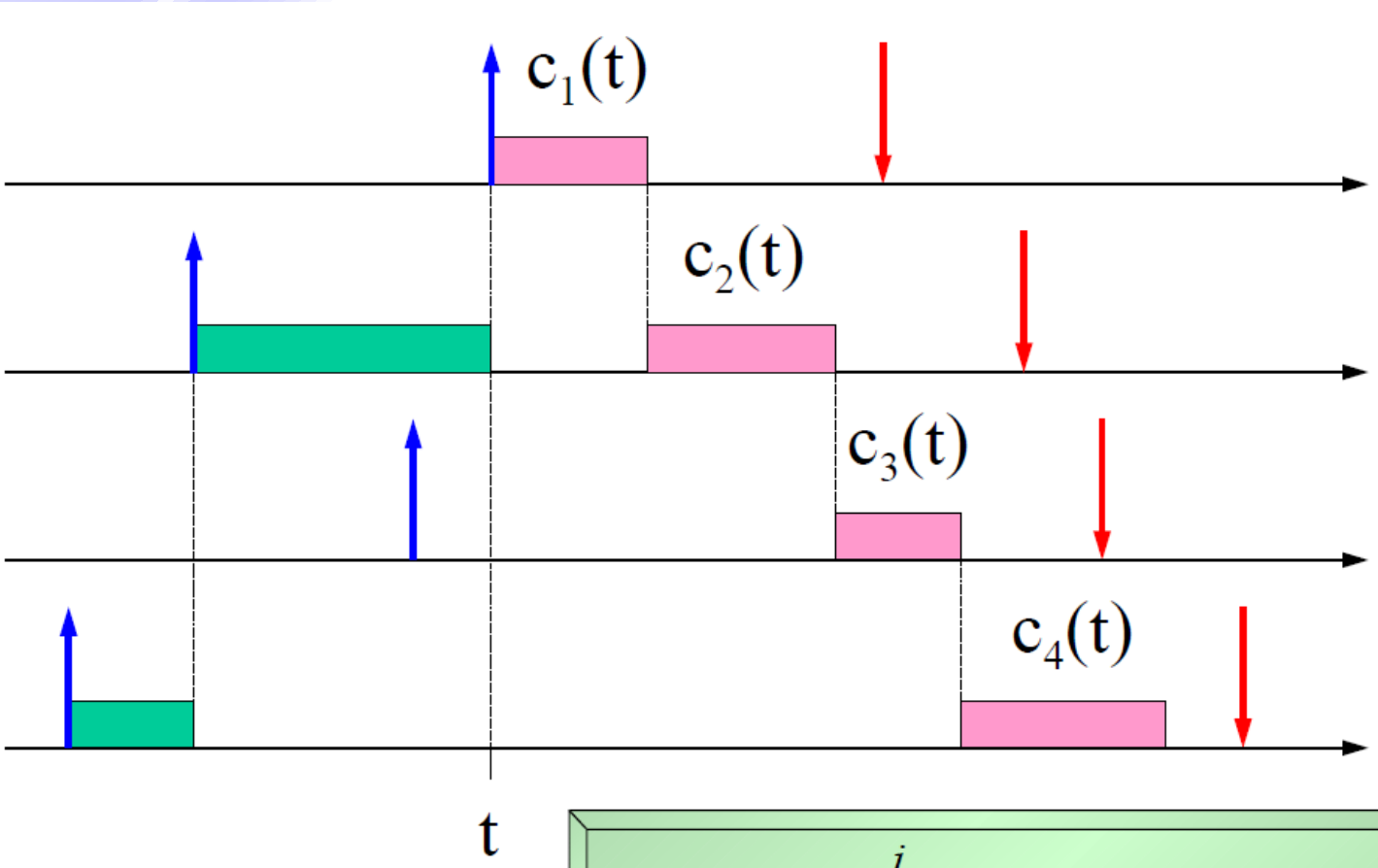


Assumptions:

- tasks may arrive at any time
- dynamic priority (d_i depends on arrival)
- fully preemptive tasks

It minimizes the maximum lateness (L_{max})

EDF - guarantee test (on line)



$$\forall\, i \qquad \sum_{k=1}^{i} c_k(t) \leqslant d_i - t$$

Complexity issues

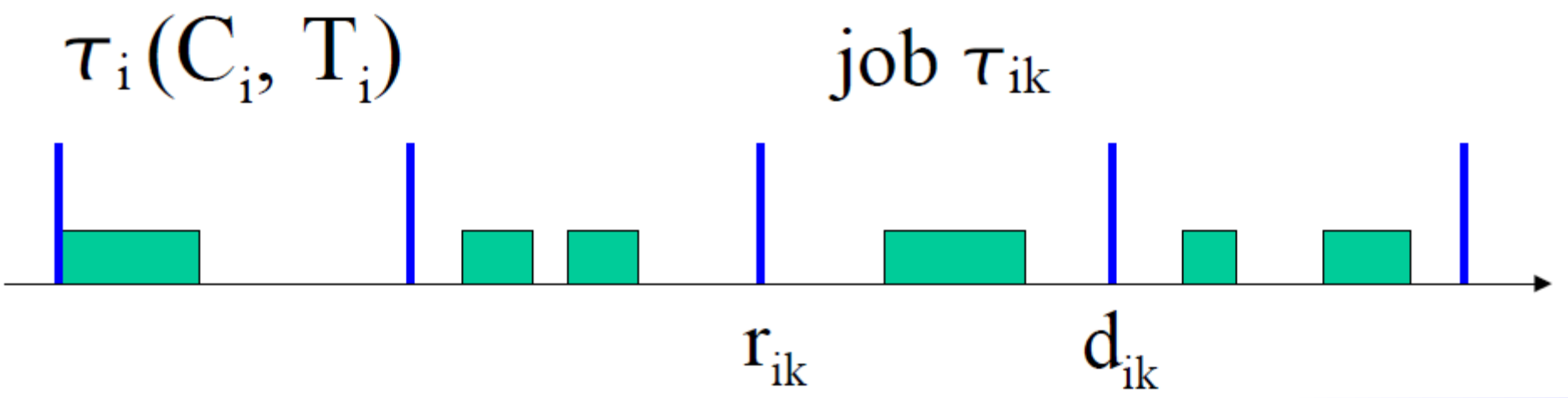
EDD

- O(n log n) to order the task set
- O(n) to guarantee the whole task set

EDF

- O(n) to insert a new task in the queue
- O(n) to guarantee a new task

Periodic task scheduling - objectives



For each periodic task, the objective is to guarantee that:

- each job τ_{ik} is activated at $r_{ik} = (k-1)T_i$ (periodic)
- each job τ_{ik} completes within $d_{ik} = r_{ik} + D_i$

Timeline scheduling

Also called cycle executive or cyclic scheduling

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

- Examples are space shuttle and Boeing 777

Method

- the time axis is divided in intervals of equal length (time slots)
- each task is statically allocated in a slot in order to meet the desired request rate
- the execution in each slot is activated by a timer

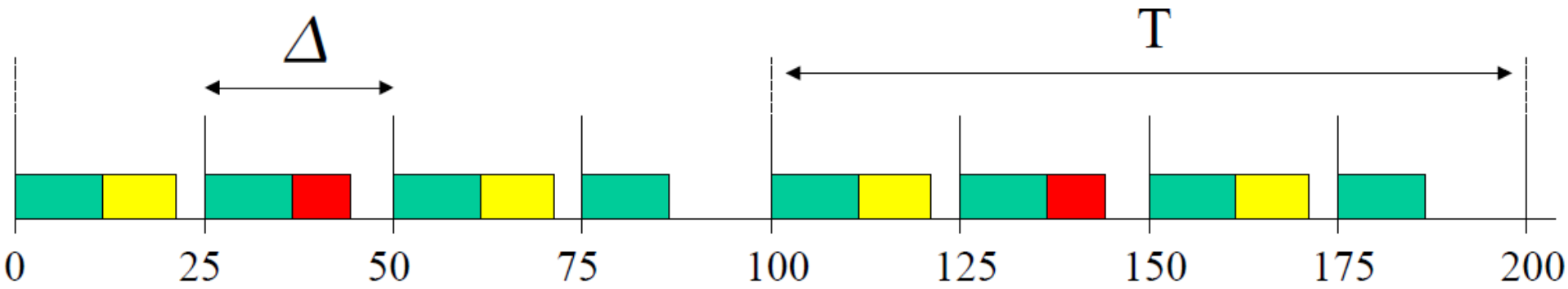
It is an algorithm for offline scheduling

Timeline scheduling - example

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

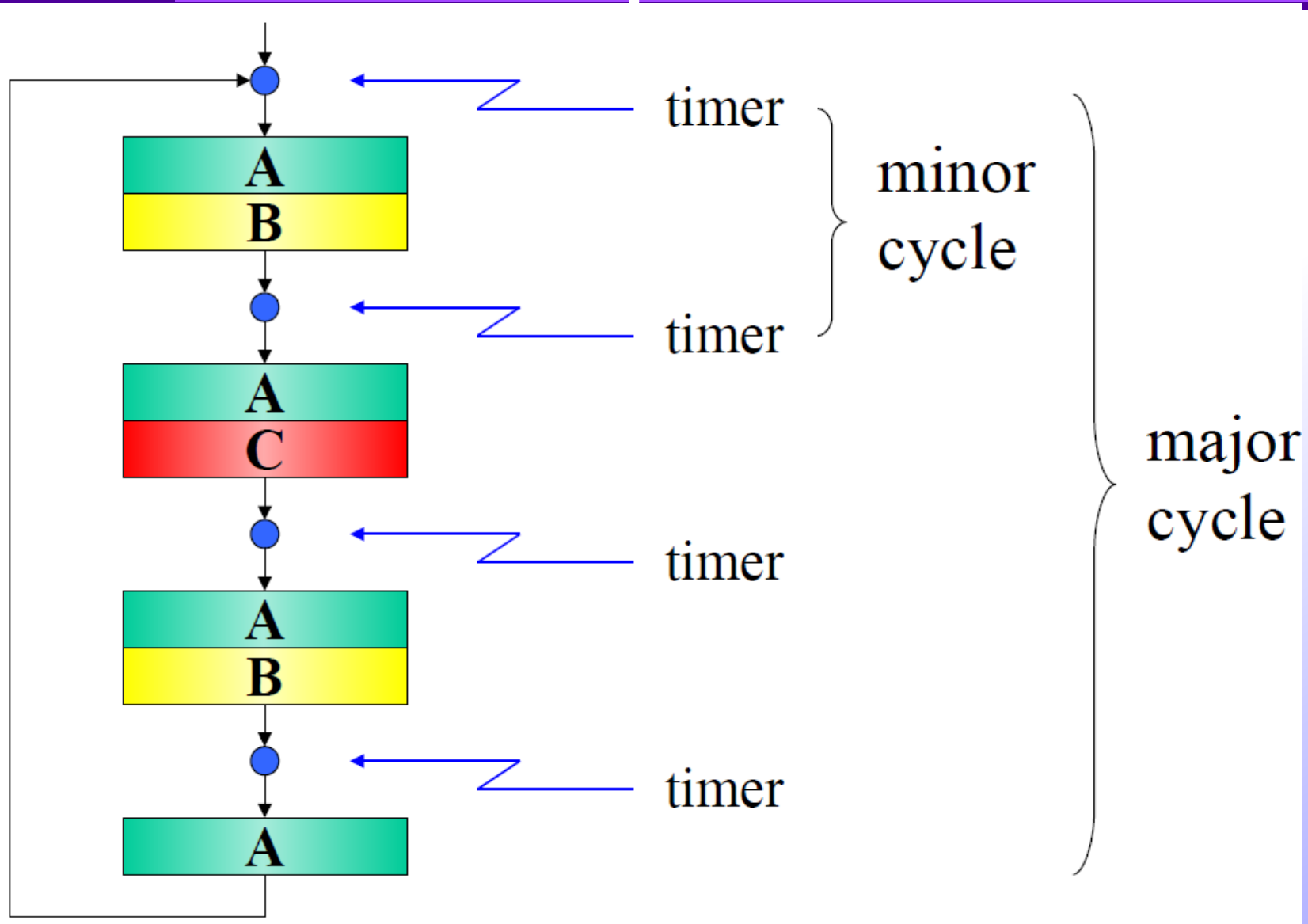
Δ = gcd (minor cycle)

T = lcm (major cycle)



guarantee: $\begin{cases} C_A + C_B \leq \Delta \\ C_A + C_C \leq \Delta \end{cases}$

Timeline scheduling - implementation



Advantages ...

- simple implementation (no real-time operating system is required)
- each procedure shares a common address space
- low run-time overhead
- it allows jitter control

...and disadvantages

- not robust during overloads
- difficult to expand the schedule
- not easy to handle aperiodic activities
- process periods must be a multiple of the minor cycle time
- difficult to incorporate processes with long periods
- difficult to construct and difficult to maintain
- any process with a variable computation time will need to be split into a fixed number of fixed sized procedures
- determinism is not required, but predictability is

Problems during overloads

What do we do during task overruns?

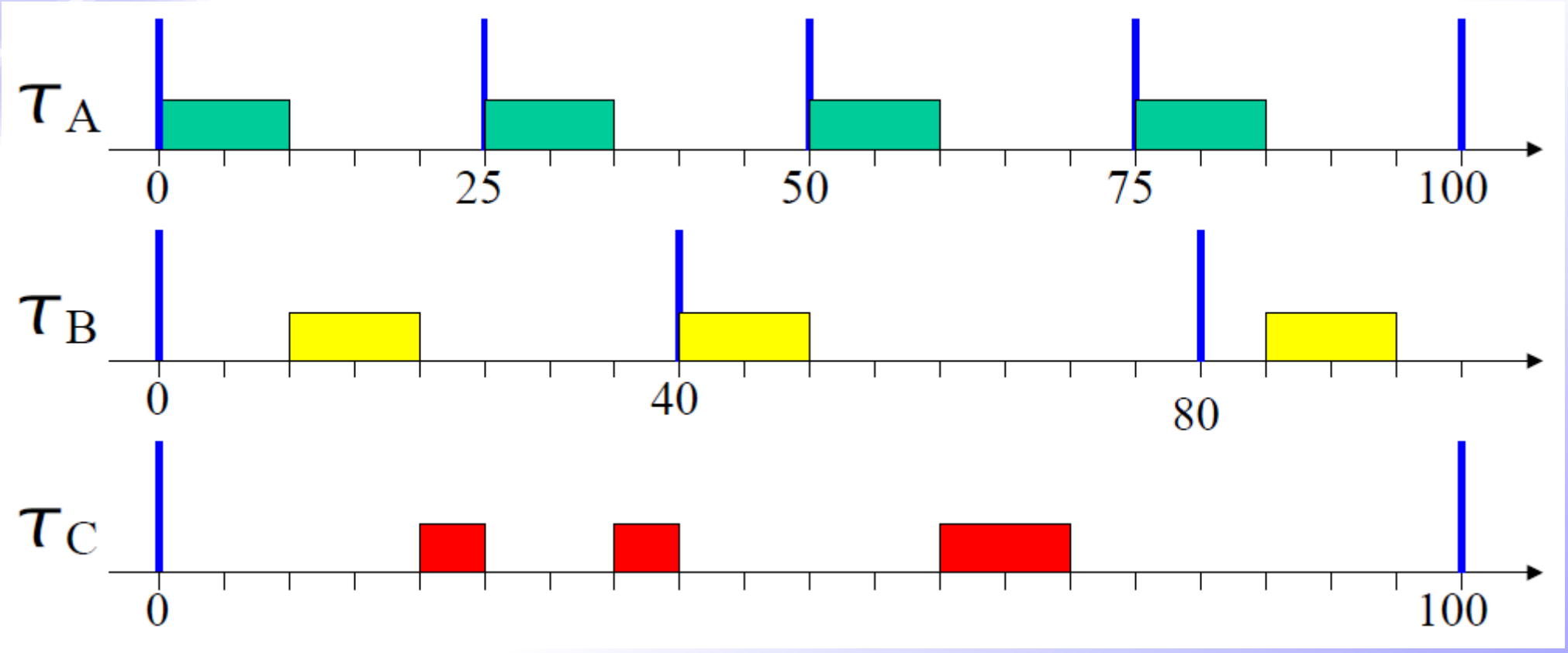
Problems during overloads

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 inconsistent states

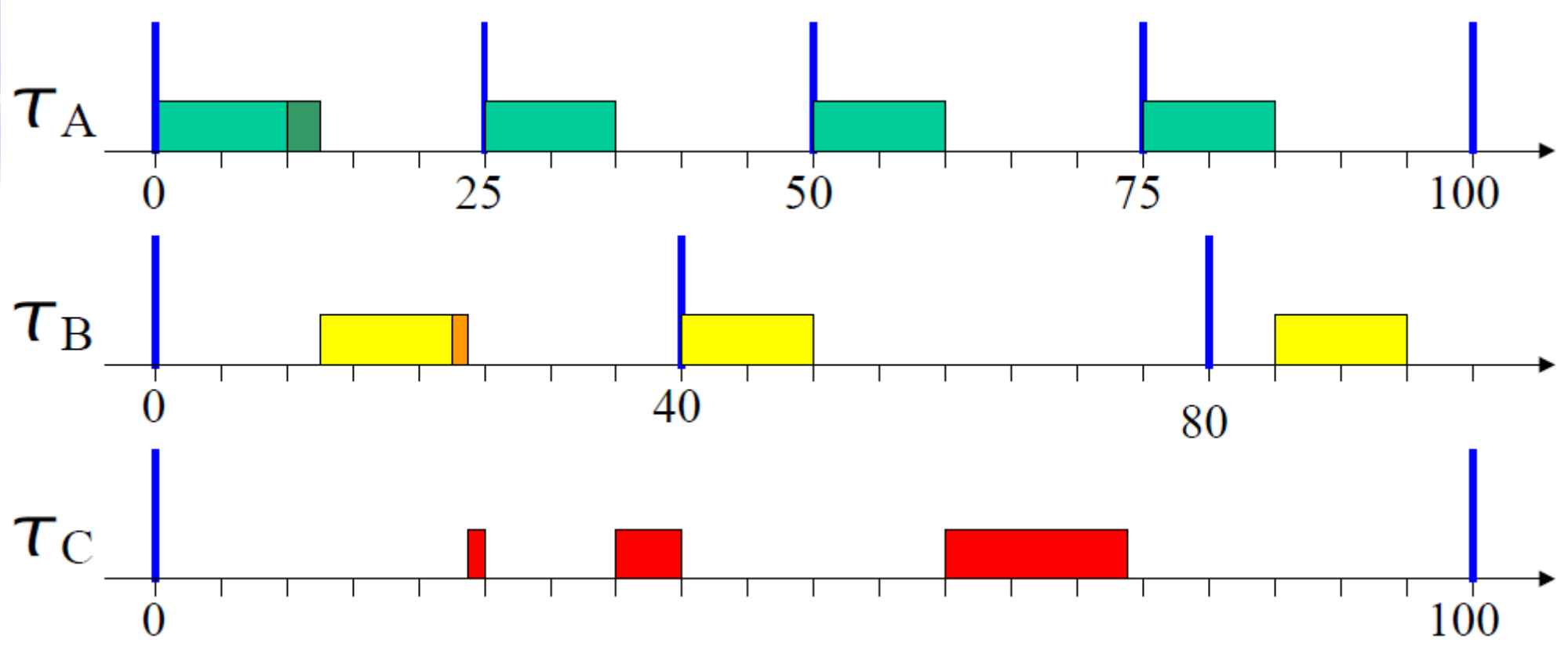
Rate Monotonic (RM)

Each task is assigned a fixed priority proportional to its rate.



RM - advantages

Transient overruns are better tolerated.



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

- U_p is a measure of the processor load

A necessary condition:

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

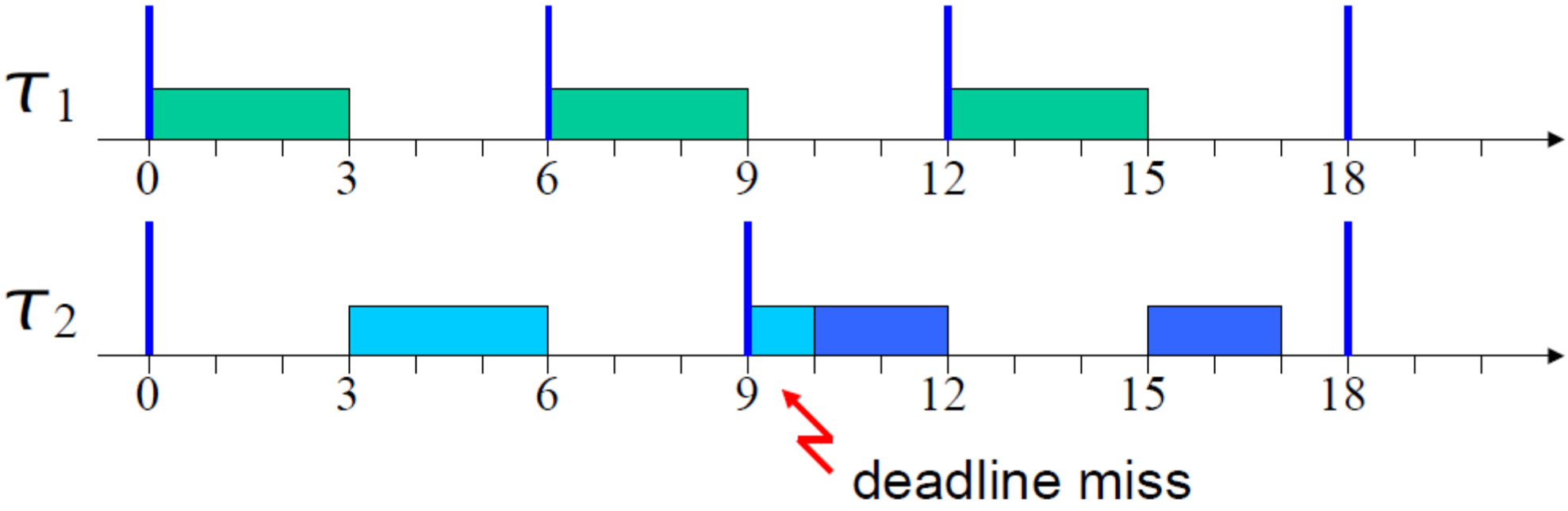
A sufficient condition (n tasks):

- $U_{low} < 0{,}69$ for a large number of tasks

$$U_{LOW} = n\left(2^{\frac{1}{N}} - 1\right)$$

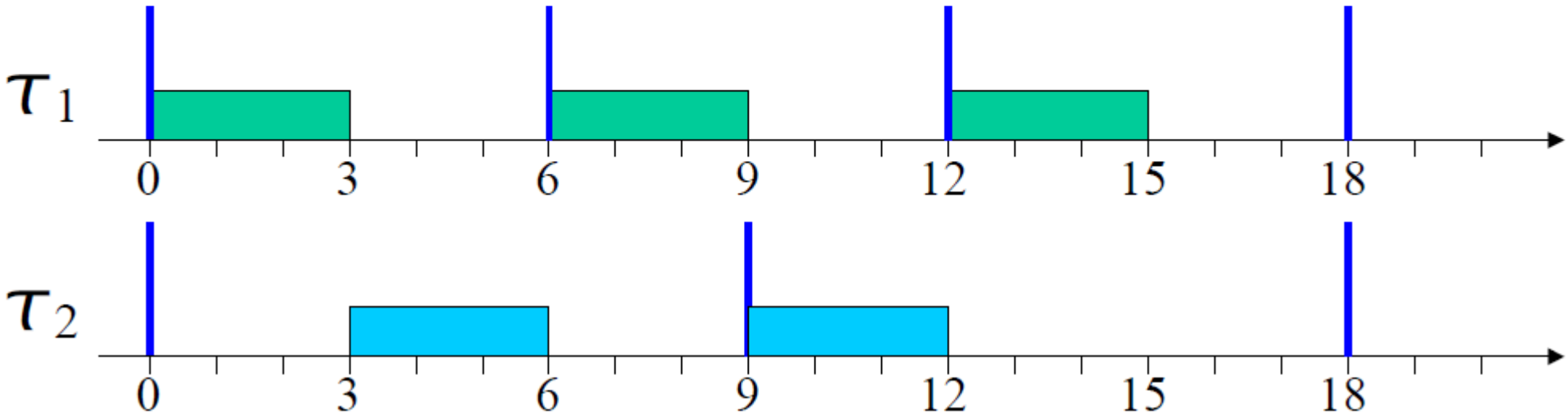
Example - Unfeasible RM schedule

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



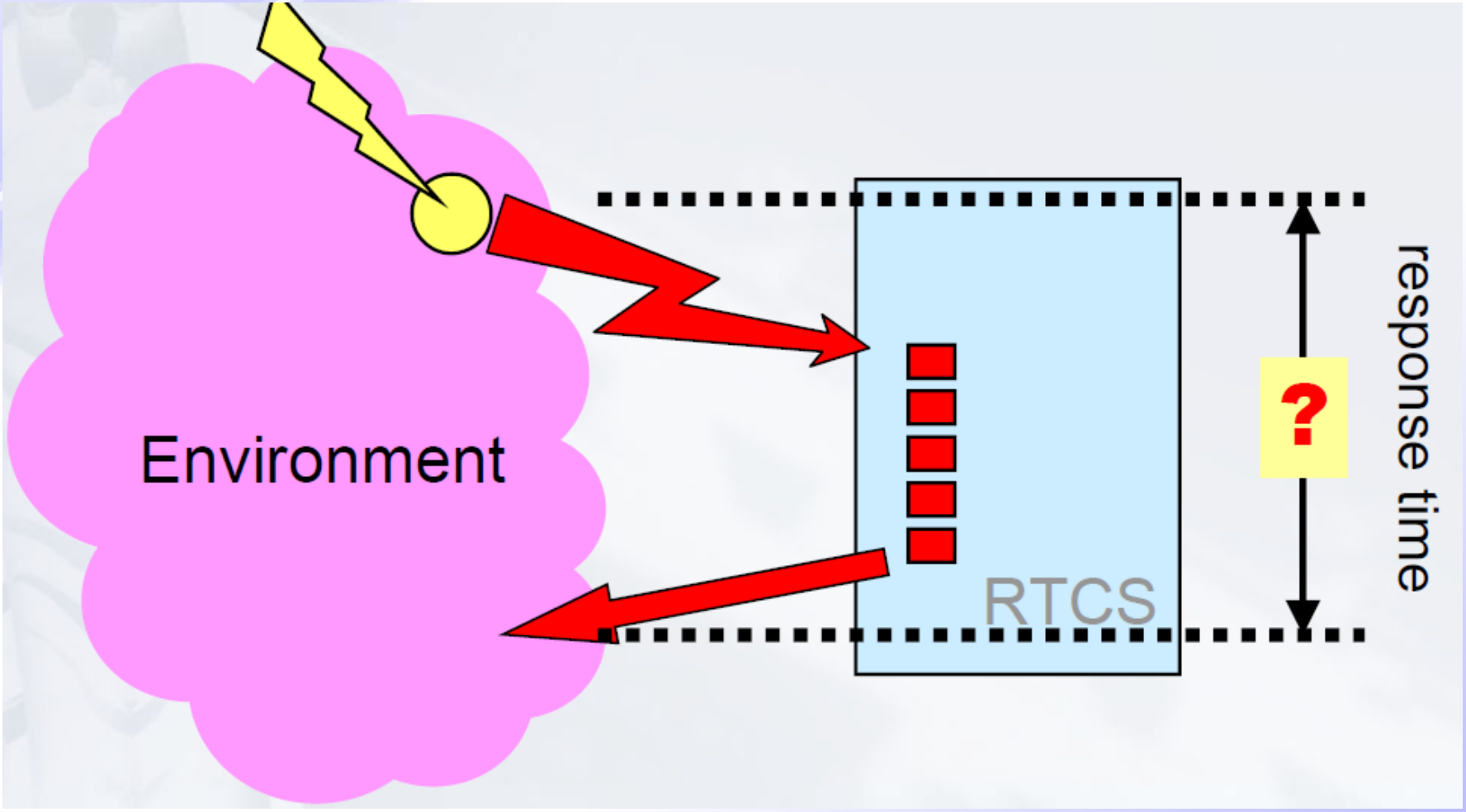
Example - Utilization upper bound

$$U_p = \frac{3}{6} + \frac{3}{9} = 0.833$$



Introduction to Worst-Case Execution-Time Analysis

How do we know the timing is right?



Time in RTS Construction

Design

Architecture, resource planning, schedules

Implementation

Timing Analysis

Schedulability analysis, WCET analysis

Timing-Analysis abstraction

In general it is infeasible to model all possible execution scenarios and combinations of task execution times

Timing analysis to derive or estimate execution time bounds or estimates

- best-case execution time (BCET), a lower bound on execution time.
- worst-case execution time (WCET), an upper bound on the execution time. Knowing worst-case execution times is of prime importance for the schedulability analysis of real-time systems.

Note: execution time is NOT response time.

Basic notions on timing analysis - 1

Bounds on the execution time of a task can be computed only by methods that consider **all possible execution times** (i.e., all possible executions) of the task

These methods use abstraction of the task to make timing analysis. Abstraction loses information, so the **computed WCET bound usually overestimates** the exact WCET (and vice versa for BCET)

- How much is lost depends on the methods used for timing analysis and on overall system properties

Timing-analysis tool **derives bounds or estimates** for the execution times of application tasks

- the WCET bound represents the worst-case guarantee the method or tool can give

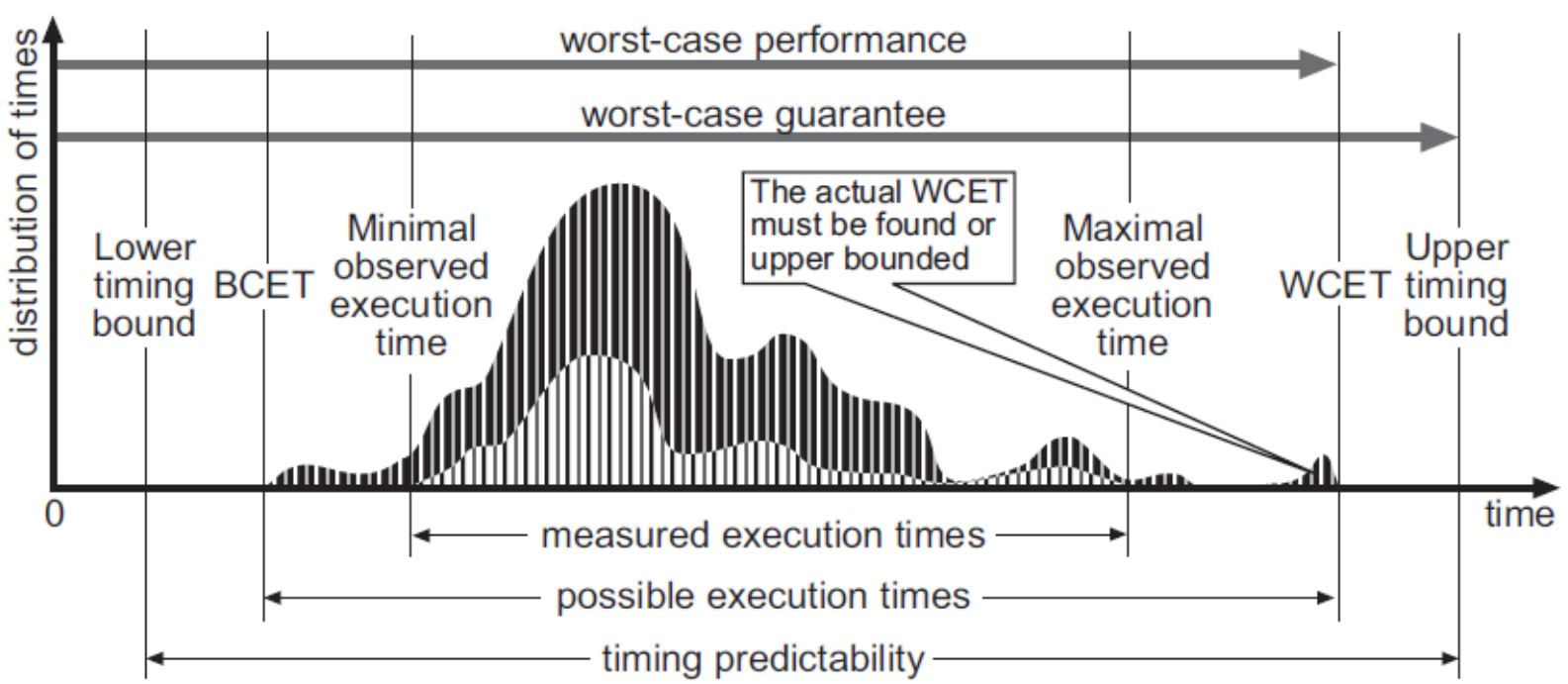
Basic notions on timing analysis - 2

Light gray curve: a subset of measured executions

- Its minimum and maximum are the minimal observed execution times and maximal observed execution times.

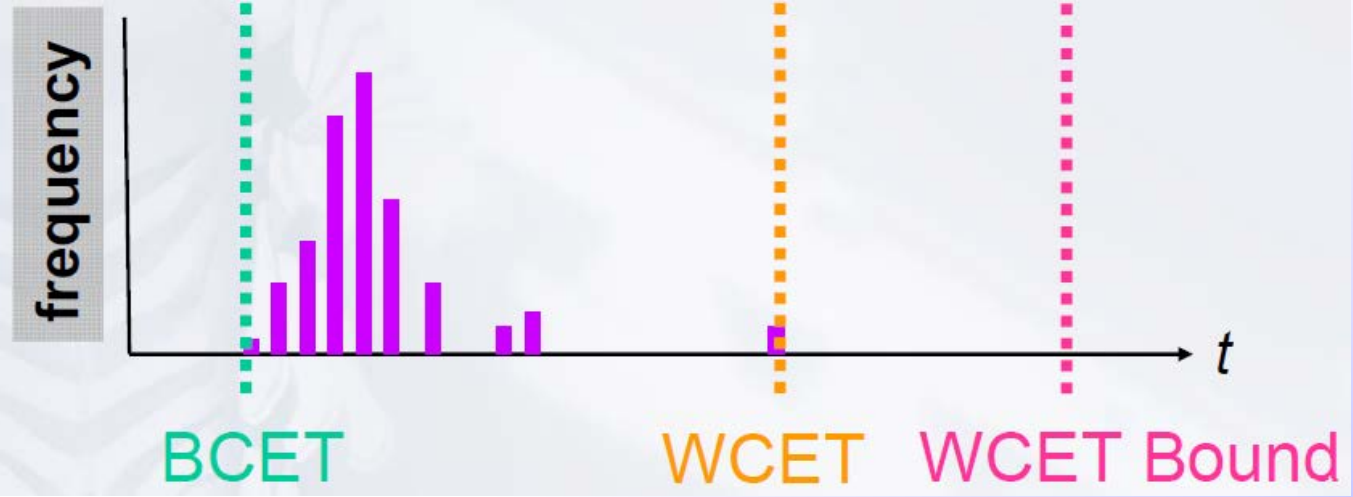
Dark grey curve: an envelope of the former, represents the times of all executions.

- Its minimum and maximum are BCET and WCET



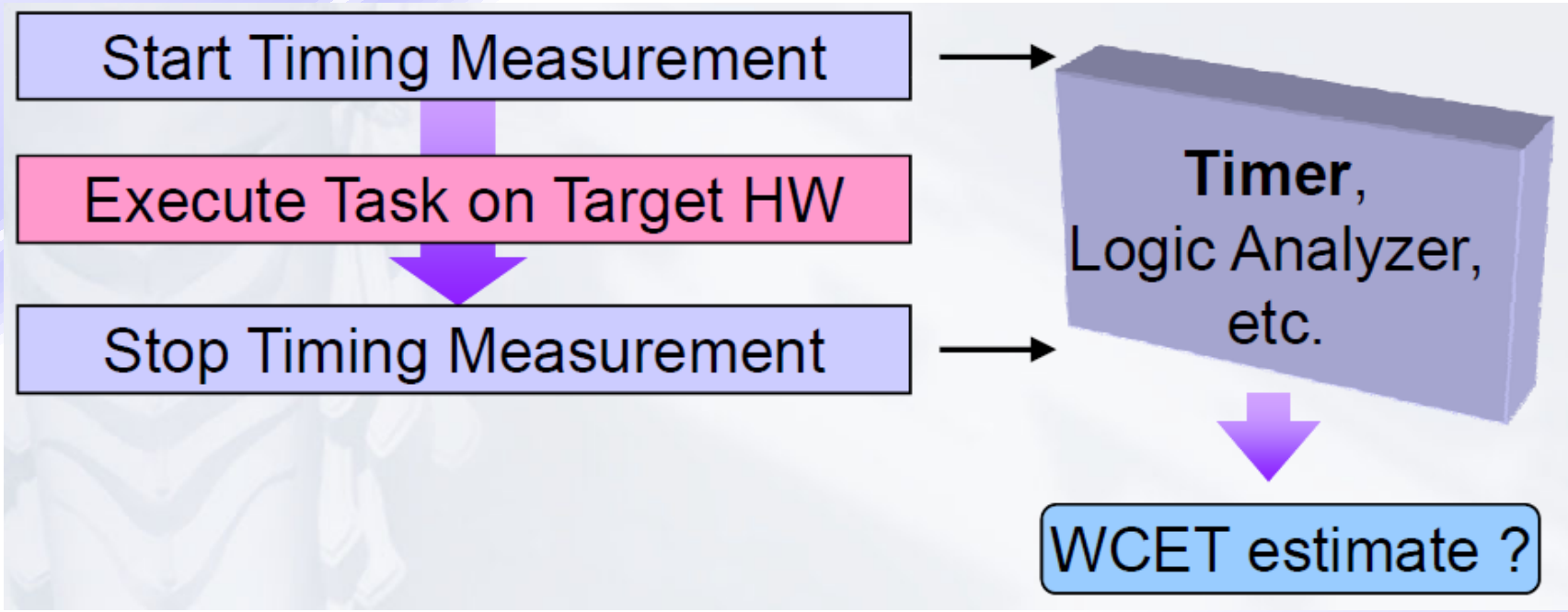
Remarks on WCET Analysis

- Computes upper bounds
- Bounds are application-dependent
- Assesses time that processor is actually executing the code
- WCET result is hardware-dependent
  - WCET bounds must be safe
  - WCET bounds should be tight



Quality of WCET analysis

Measurement-can you measure WCET?



Measuring all different execution traces of a real-size program is intractable in practice (e.g., mid-size task with 1040 different paths)

Selected test data for measurement may fail to trigger the longest execution trace

- (continue on next slide)

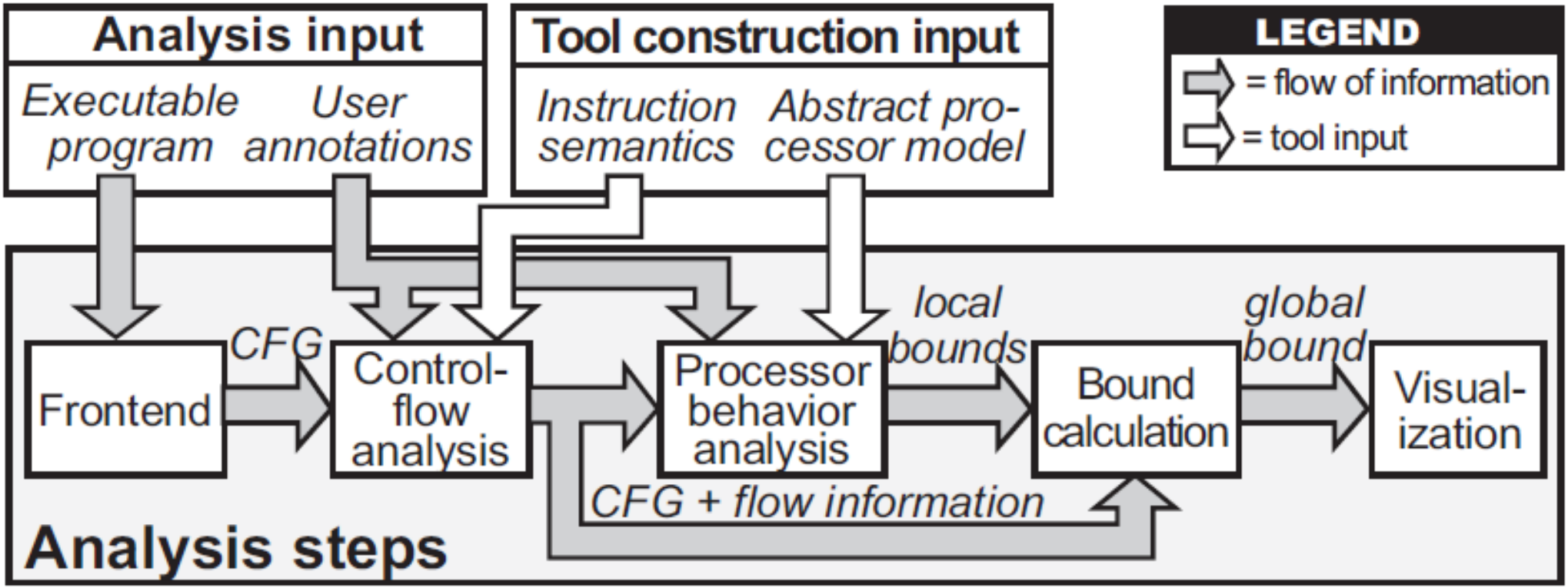
Measurement-can you measure WCET?

- When selecting test data, rare execution scenarios may be missed (e.g., exception handling, …)
- Partitioning: combining WCET of parts does not necessarily yield the global WCET (anomalies)
- Internal processor state may not have been in its worst-case initial setting

Systematic WCET analysis:

- Simple measurements may be useful to get a first rough estimate of the execution time, but more systematic WCET analysis techniques and possibly hardware adaptions are required to obtain a trustworthy WCET bound!

Static timing-analysis tool



Core components of a static timing analysis tool

- The flow of information is shown by brown arrows
- The white arrows represent tool-construction input

Methods for Static WCET Analysis

Methods do not rely on executing code on real hardware or on a simulator, but rather take the task code itself, combines it with some (abstract) model of the system, and obtains upper bounds from this combination

- Value Analysis, control-flow analysis, process-behaviour analysis, symbolic simulation …

Control-Flow Analysis: gather information about possible execution paths

- Input: a task representation (e.g. the call graph) and possibly additional information such as ranges for the input data and iteration bounds of loops
- Results: constraints on the dynamic behavior of the task.

Bibliography

- Giorgio C. Buttazzo, Sistemi in Tempo Reale, Pitagora Editrice Bologna, 2006.
- Lui Sha and John B. Goodenough. 1990. Real-Time Scheduling Theory and Ada. *Computer* 23, 4 (April 1990), 53-62.
- Reinhard Wilhelm et al., The worst-case execution-time problem — overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages.
- http://ti.tuwien.ac.at/rts/teaching/courses/wcet