TECHNISCHE UNIVERSITÄT DRESDEN

**Faculty of Computer Science** Institute of Systems Architecture, Operating Systems Group

# DISTRIBUTED OPERATING SYSTEMS

## *REAL-TIME SCHEDULING*

**https://tud.de/inf/os/studium/vorlesungen/dos**

**HORST SCHIRMEIER**

# Overview

- Real-Time Systems
- Example: OSEKtime
- Real-Time Scheduling Strategies
  - *Rate Monotonic Scheduling*
  - *Earliest Deadline First Scheduling*
- Summary and Outlook

# Overview

- **Real-Time Systems**
- Example: OSEKtime
- Real-Time Scheduling Strategies
  - *Rate Monotonic Scheduling*
  - *Earliest Deadline First Scheduling*
- Summary and Outlook

# Real-Time Computer Systems

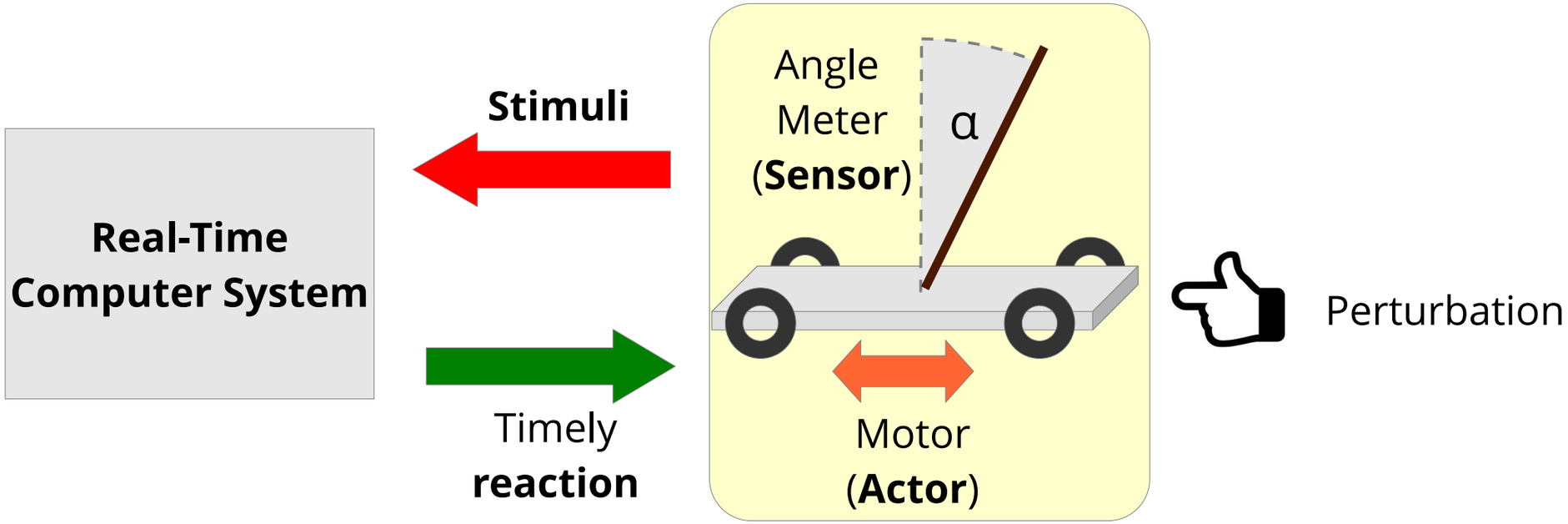
- What's that?

> *„A **real-time computer system** is a computer system in which the **correctness** of the system behavior depends not only on the logical results of the computations, but also on the physical **instant** at which these results are produced."*

Hermann Kopetz [1]

# Example “Inverted Pendulum”

**Goal: α shall be 0°**




***“Controlled Object”***

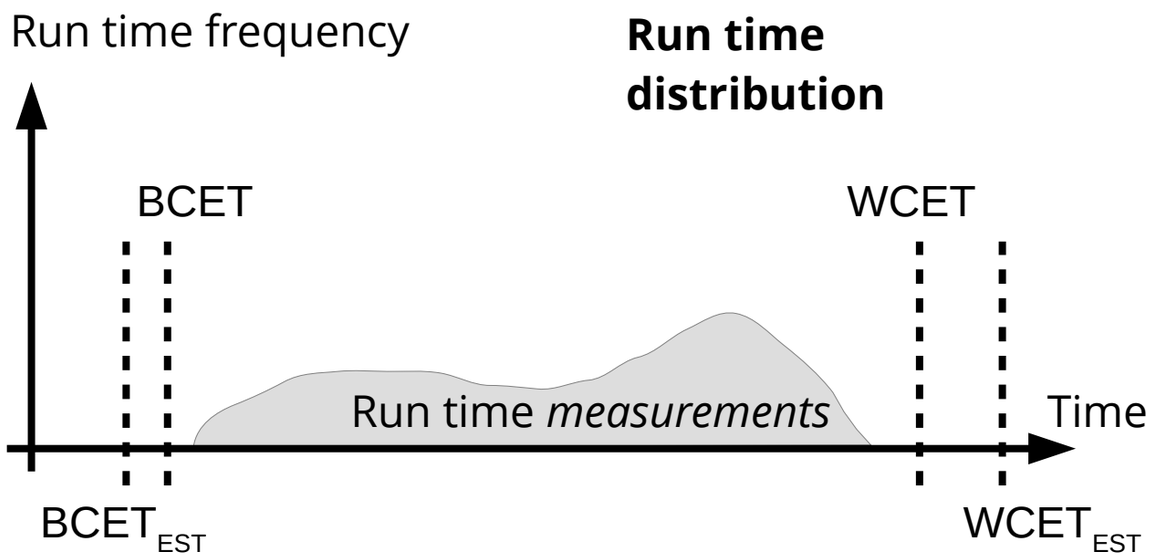
**The reaction time of the computer system (time between the stimulus and the reaction) and the fluctuation (“jitter”) should be minimal.**

# Deadlines

- Defined by the controlled technical/physical system
- Classification:
  - ***soft:*** The result has **utility** even after the deadline has passed – the reaction is still useful.
  - ***firm:*** The result has no utility after the deadline has passed.
  - ***hard:*** A deadline miss is potentially catastrophic.
- A real-time system (with multiple deadlines) is classified as “hard” if at least one deadline is hard. Otherwise, it is classified as “soft”.
  - Hard real-time systems **have to guarantee** meeting their deadlines. This implies different development methods and system structures.

# How long does a program run?

- Run times vary: Different input parameters, hardware state at the beginning, interrupts, process switches, power management, …




**The estimated WCET$_{EST}$ must be guaranteed to be greater than or equal to the true WCET. However, the gap should be as small as possible (tight bounds).**

- Particularly important: ***Worst Case Execution Time*** **(WCET)**

# Triggers …

… for a **task** can be realized in different ways:

- **Event-triggered Real-Time Systems**
  - A sensor detects a relevant state change – an **event** – of the controlled object.
  - **Task scheduling** takes place **at runtime**.
  - High effort for tests under high load
  - Predictions difficult → **soft real-time systems**

# Triggers …

… for a **task** can be realized in different ways:

- **Time-triggered Real-Time Systems**
  - **Fixed task starting times are planned ahead-of-time** (“offline scheduling”), task are executed periodically.
  - Higher resource requirements: Planning based on **WCET**
  - High energy consumption, continuously active
  - Lower test effort
  - Guarantees possible → **hard real-time systems**

# Overview

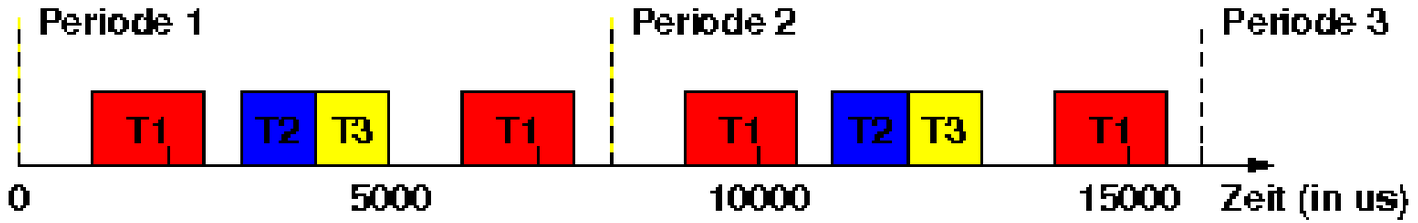
- Real-Time Systems
- **Example: OSEKtime**
- Real-Time Scheduling Strategies
  - *Rate Monotonic Scheduling*
  - *Earliest Deadline First Scheduling*
- Summary and Outlook

# OSEKtime [2]: Goals

- Safe realization of ***„X-By-Wire“*** applications
  (*Steer-by-wire*, *Brake-by-wire*, eGas)
  - Guaranteed, predictable behavior
    - Support for time-triggered applications
    - → OSEKtime OS specification (version 1.0: 2001)
  - Global coordination within the ECU (electrical control unit) network
    - Global time!
    - → FTCom specification
- Compatibility with “classic” OSEK OS tasks
  - Support for event-triggered applications

# OSEKtime: Scheduler

- **Offline Scheduling**: A **dispatcher table** controls periodic task activation:



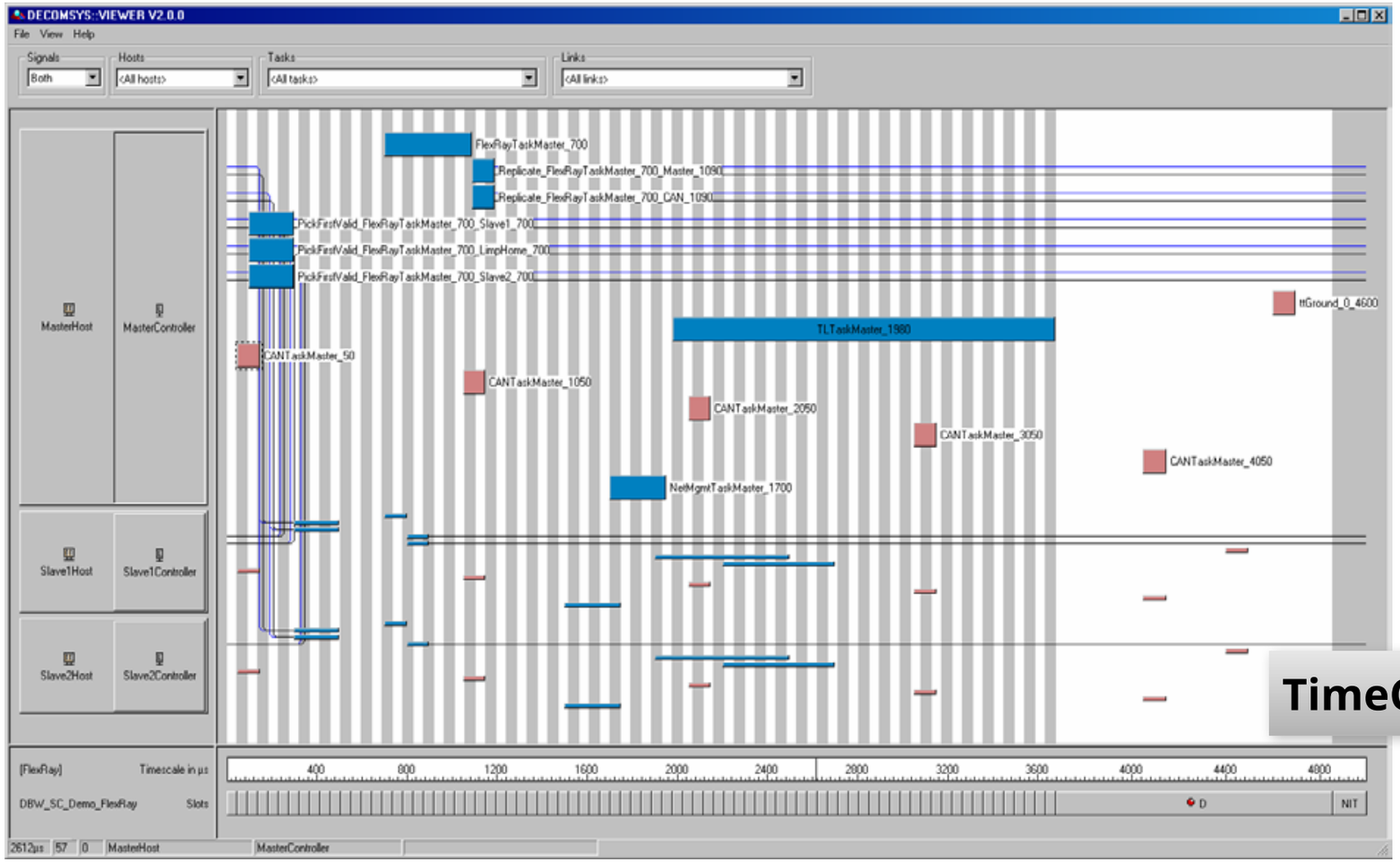

| Task | Start time |
|---|---|
| T1 | 1000 µs |
| T2 | 3000 µs |
| T3 | 4000 µs |
| T1 | 6000 µs |

**A dispatcher table matching the example. One complete cycle is called *„Dispatcher Round“*.**

- Timer interrupt ensures dispatcher activation.
- Only the dispatcher can activate tasks.
- Safety mechanism: ***Deadline Monitoring***

# *Offline Scheduling*

- Tools help developers to arrange and plan tasks.



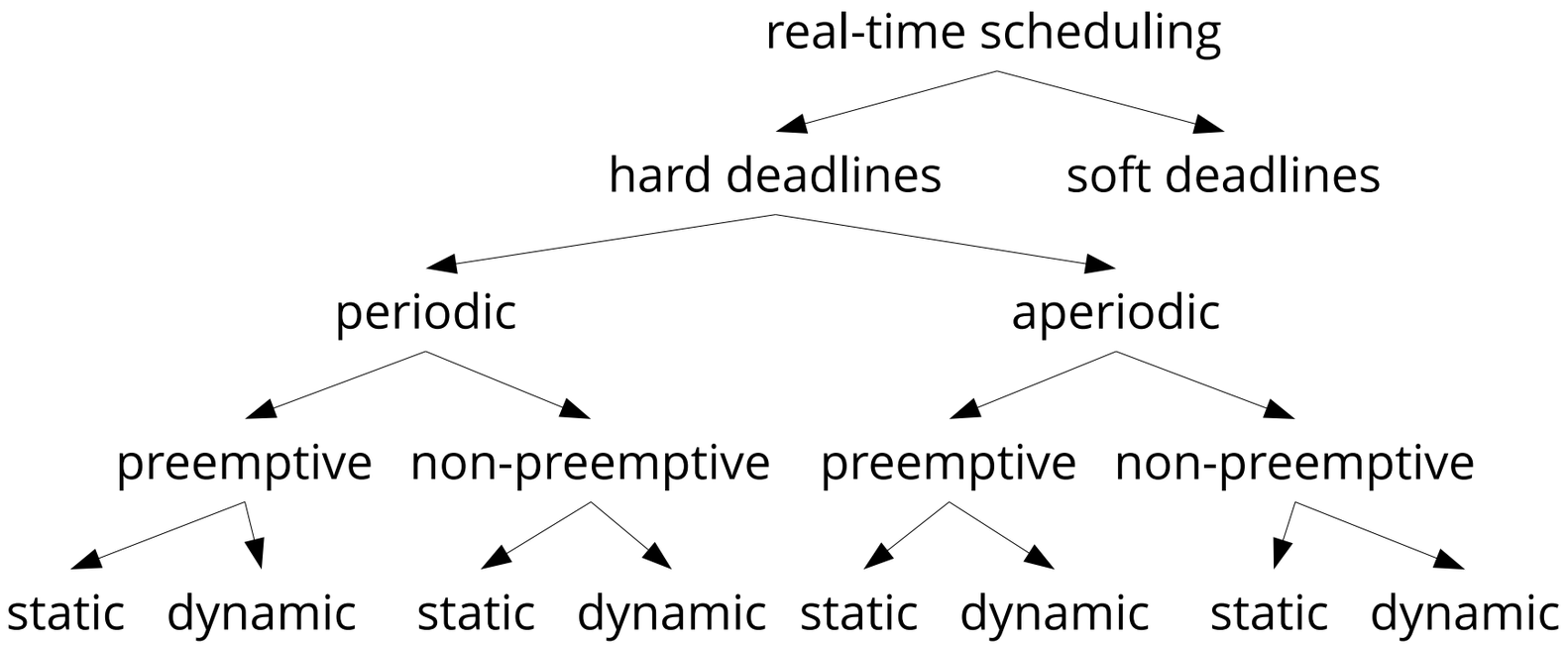

**TimeCore**

# Overview

- Real-Time Systems
- Example: OSEKtime
- **Real-Time Scheduling Strategies**
  - *Rate Monotonic Scheduling*
  - *Earliest Deadline First Scheduling*
- Summary and Outlook

# Real-Time Scheduling

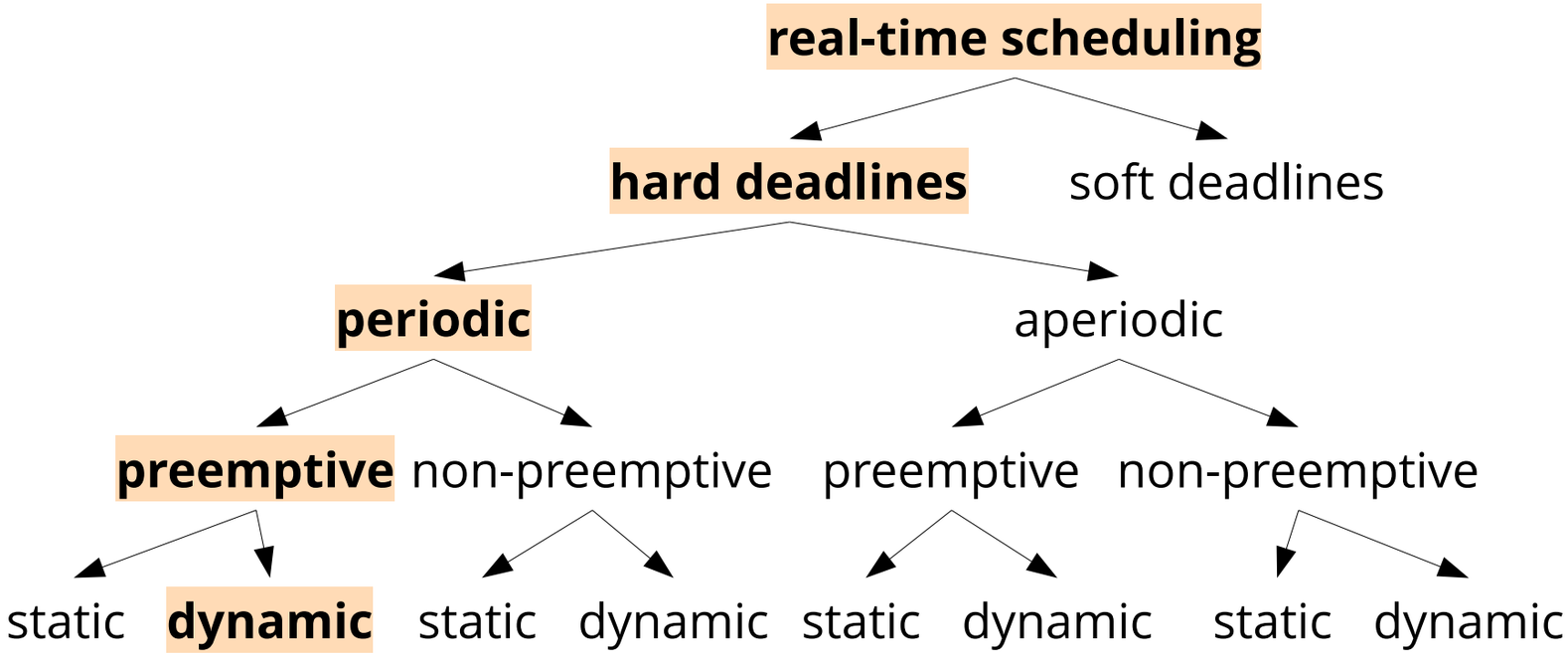
- Aims at giving mathematical guarantees for meeting hard deadlines.
- Taxonomy [3, p. 239]



# Overview

- Real-Time Systems
- Example: OSEKtime
- **Real-Time Scheduling Strategies**
  - ***Rate Monotonic Scheduling***
  - *Earliest Deadline First Scheduling*
- Summary and Outlook

# Example: *Rate-Monotonic* Scheduling




- ***Rate-Monotonic (RM)*** *Scheduling* is a scheduling strategy for preemptive, periodic tasks with hard deadlines. The scheduler works at runtime (with fixed priorities).

# RM Assumptions (Liu & Layland 1973 [4])

A1. All tasks are **preemptible** at any time.
Preemption costs (duration) are **negligible**.

A2. Only required **processing power** is relevant.
Memory, I/O etc. requirements are negligible.

A3. All tasks are **independent**. There are no ordering dependencies.

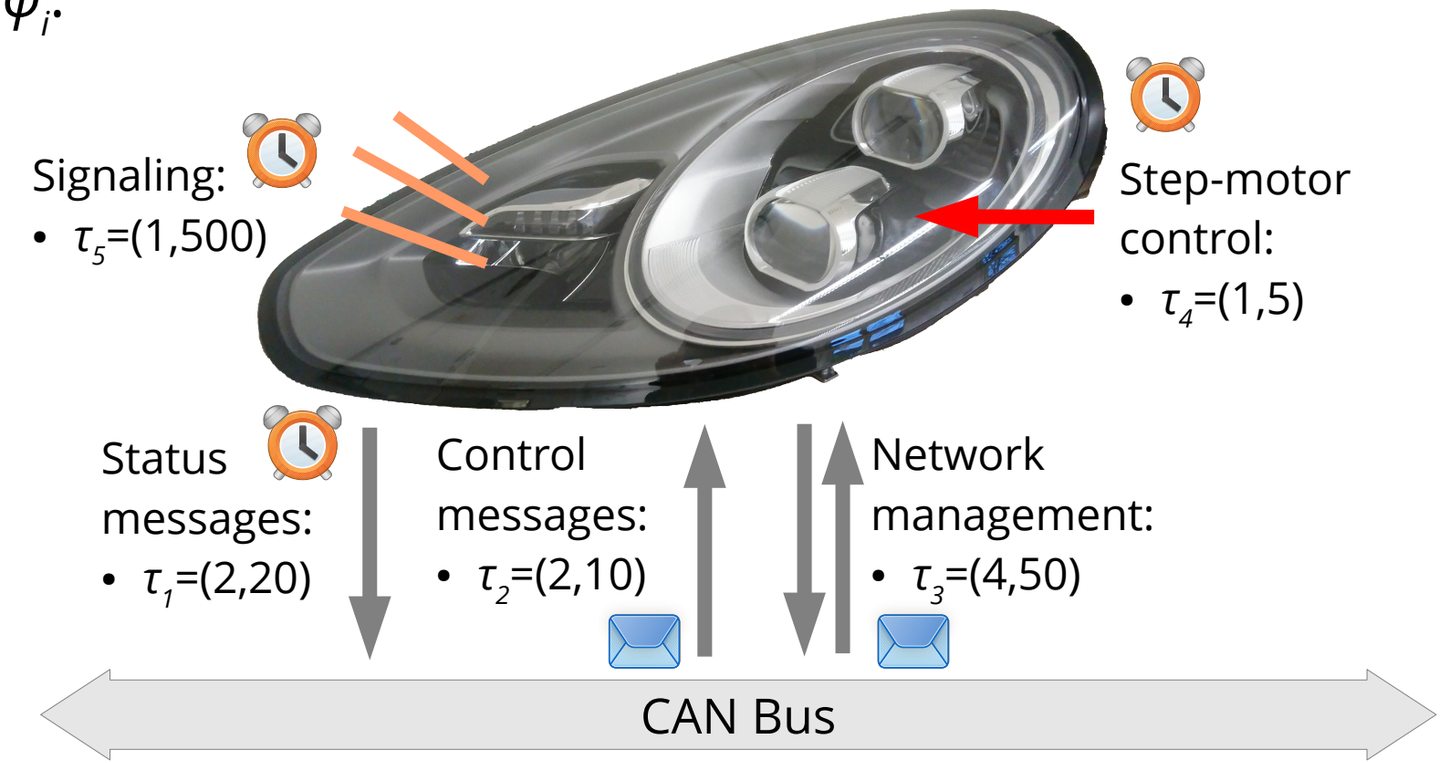
A4. All tasks are **periodic**.

A5. A task's **relative deadline** is identical to its **period**.

# Example: Automotive Headlight ECU

... everything is periodic!

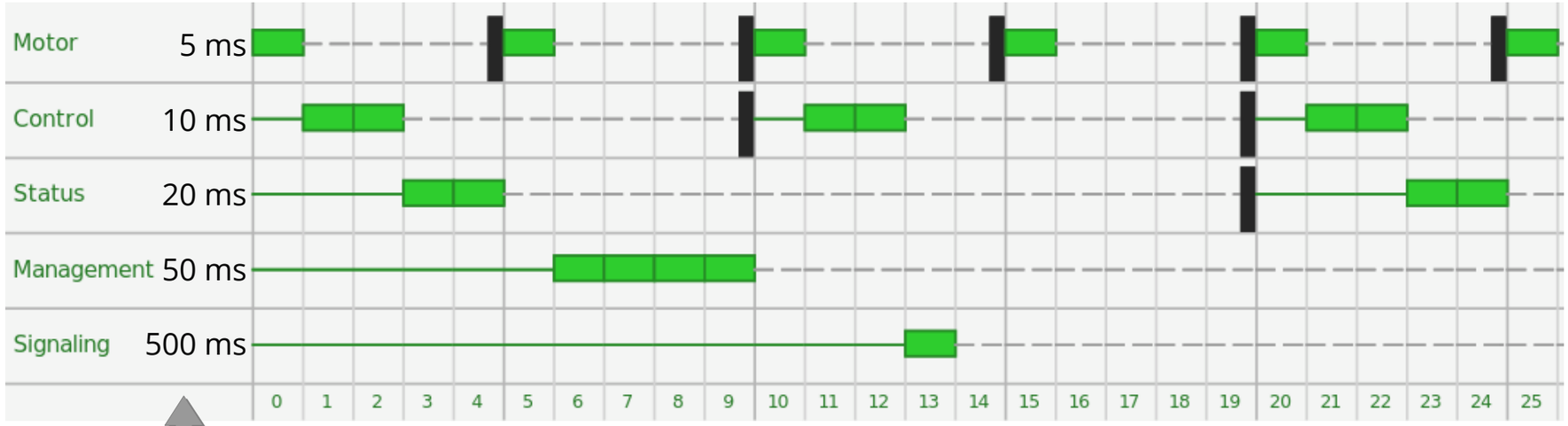
- For each task $\tau_i=(C_i,T_i)$, WCET $C_i$ and periode $T_i$ are known, but not the time offset (the "phase") $\phi_i$.



# Rate Monotonic Algorithm

- Priority increases monotonically with the event rate (=frequency)
  - i.e., small period → high priority
- High-priority tasks preempt low-priority tasks
- Example:

Gantt diagram for $\phi_i = 0$




To implement RMS in practice, all you need is an operating system with a preemptive “fixed priority” scheduler.

# Schedulability Analysis

- Question to be answered: **Do all tasks meet their deadlines?**
  - We can only calculate a schedule if all tasks are completely time triggered. In our example, the phases are arbitrary.

- Must hold: The system's utilization *U* is less than or equal to 1.

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

*U*: System utilization
*m*: Number of tasks

**Assumption: Uniprocessor**

- Example: $\tau_1$=(1,5), $\tau_2$=(2,20), $\tau_3$=(2,10), $\tau_4$=(4,50), $\tau_5$=(1,500)

$$U = \sum_{i=1}^{m} \frac{C_i}{T_i} = \frac{1}{5} + \frac{2}{20} + \frac{2}{10} + \frac{4}{50} + \frac{1}{500} = 0.582 \leq 1$$

**OK, U ≤ 1, but is this sufficient?**

# The “70% Rule” [4]

- Claim: **No deadline miss**, if the following condition holds:

$$U \leq m \cdot \left(2^{1/m} - 1\right)$$

*U*: System utilization
*m*: Number of tasks

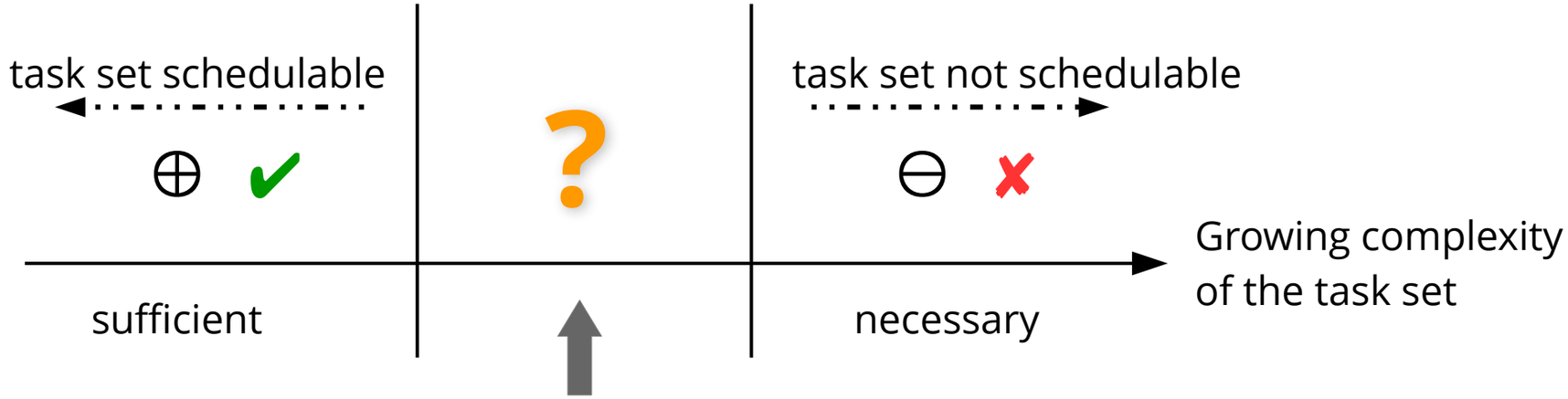
- For large *m*, the limit converges towards ln(2) ≈ 0,6931, i.e. about 70%.

**Limit value calculation: de L'Hospital's rule**

- **Advantage:** Simple test, fast calculation
  - Example 1: *U*=58.2%, *m*=5
    - $m \cdot (2^{1/m} - 1) = 74.35\%$, condition holds → no deadline miss ✔
  - Example 2: $\tau_1$=(2,5) instead of $\tau_1$=(1,5), then *U*=78.2%, *m*=5
    - $m \cdot (2^{1/m} - 1) = 74.35\%$, condition does **not** hold → **maybe** deadline miss
- **Disadvantage:** No conclusion if the condition does not hold

# Sufficient and Necessary Conditions

- **Sufficient** condition **true**
  - e.g. $U \leq m \cdot \left(2^{1/m} - 1\right)$
  - → Task set is **schedulable**
- **Necessary** condition **false**
  - e.g. $U \leq 1$ does not hold
  - → Task set is **not schedulable**



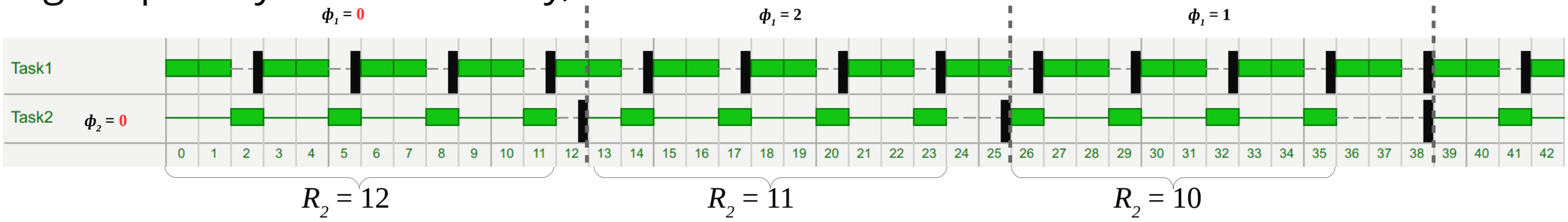

**Ideally a "precise test": Sufficient *and* necessary condition**

# Exact Test: Response-Time Analysis [5]

Condition (necessary *and* sufficient)

$$\forall i \in \{1, \ldots, m\} : R_i \leq T_i$$

- Iff response time $R_i$ is less than or equal to period $T_i$ for *all* tasks, all deadlines are met.
- Highest-possible start delay with $\phi_i = 0$: Already at the beginning of the period, all higher-priority tasks are ready, too.




- Calculating $R_i$:

$$R_i = C_i + I_i = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

$I_x$: "Interference" – Delay caused by higher-priority tasks

$hp_x$: Indexes of tasks with higher priority than $x$

$\lceil x \rceil$: Integer round-up (*ceiling* function)

# Exact Test: Iterative Solution

- Calculate $R_i$ by fixed-point iteration:
  - Abort as soon as $R_i^{n+1} = R_i^n$ or $R_i^{n+1} > T_i$

$$R_i^{n+1} = C_i + \sum_{j \in hp_i} \left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j$$

  - Test pseudocode for all tasks:

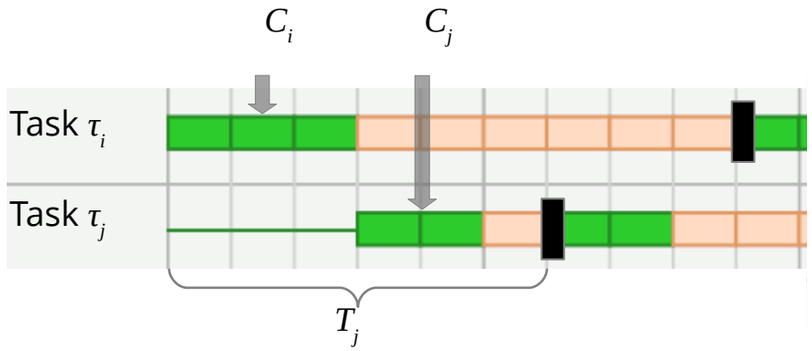
```
for (each task τ_i) {
    I = 0
    do {
        R = I + C_i
        if (R > T_i) return false // deadline is missed
        I = Σ_{j∈hp_i} ⌈R / T_j⌉ · C_j
    } while (I + C_i > R)
}
return true // all deadlines are met
```

# *Rate-Monotonic Scheduling* is "optimal"

- To prove:

RM is an *optimal* scheduling algorithm for *fixed* priorities. That means, if *any* algorithm provides a valid schedule, then RM does so, too.

- Direct proof: Assume, algorithm A provides a valid schedule but prioritizes **long** periods:
  - In A's schedule: $prio(\tau_i) = prio(\tau_j) + 1$ and $T_i > T_j$ (different to RM)
    - $C_i + C_j \leq T_j$ is true, because the schedule is valid and $\tau_i$ has a higher priority

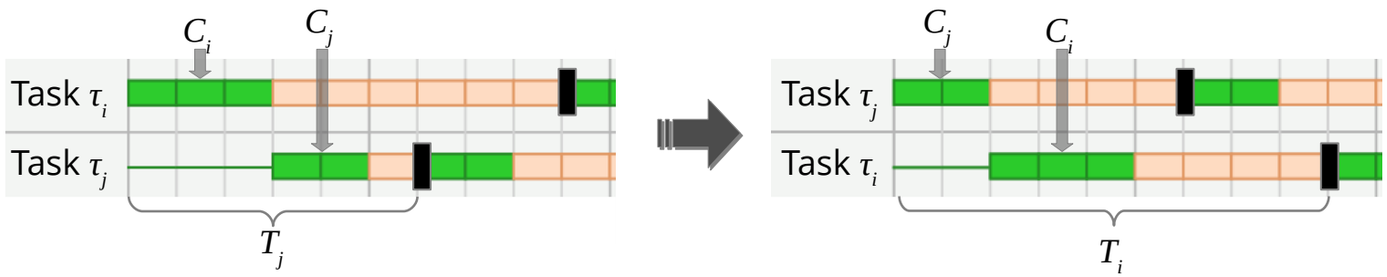


# *Rate-Monotonic Scheduling* is "optimal"

- To prove:

RM is an *optimal* scheduling algorithm for *fixed* priorities. That means, if *any* algorithm provides a valid schedule, then RM does so, too.

- Direct proof: Assume, algorithm A provides a valid schedule but prioritizes **long** periods:
  - In A's schedule: prio($\tau_i$) = prio($\tau_j$) + 1 and $T_i > T_j$ (different to RM)
    - $C_i+C_j \leq T_j$ is true, because the schedule is valid and $\tau_i$ has a higher priority
  - What's the effect of exchanging the priorities of (only) these two tasks?
    - $\tau_j$ schedulable, because now with higher priority; $\tau_i$ also schedulable because $C_i+C_j \leq T_j < T_i$



# *Rate-Monotonic Scheduling* is "optimal"

- To prove:

RM is an *optimal* scheduling algorithm for *fixed* priorities. That means, if *any* algorithm provides a valid schedule, then RM does so, too.

- Direct proof: Assume, algorithm A provides a valid schedule but prioritizes **long** periods:
  - In A's schedule: $\text{prio}(\tau_i) = \text{prio}(\tau_j) + 1$ and $T_i > T_j$ (different to RM)
    - $C_i + C_j \leq T_j$ is true, because the schedule is valid and $\tau_i$ has a higher priority
  - What's the effect of exchanging the priorities of (only) these two tasks?
    - $\tau_j$ schedulable, because now with higher priority; $\tau_i$ also schedulable because $C_i + C_j \leq T_j < T_i$
    - **Also** a valid schedule → **RM is optimal!**

# RM Scheduling: Conclusion

- RM is **easy to use** and **optimal** for fixed priorities
  - Operating system must only provide a fixed-priority scheduler
- Response-time analysis enables exact schedulability test
  - Important for hard real-time systems: Mathematical **guarantee**!
- In most cases, the “70% rule” suffices.

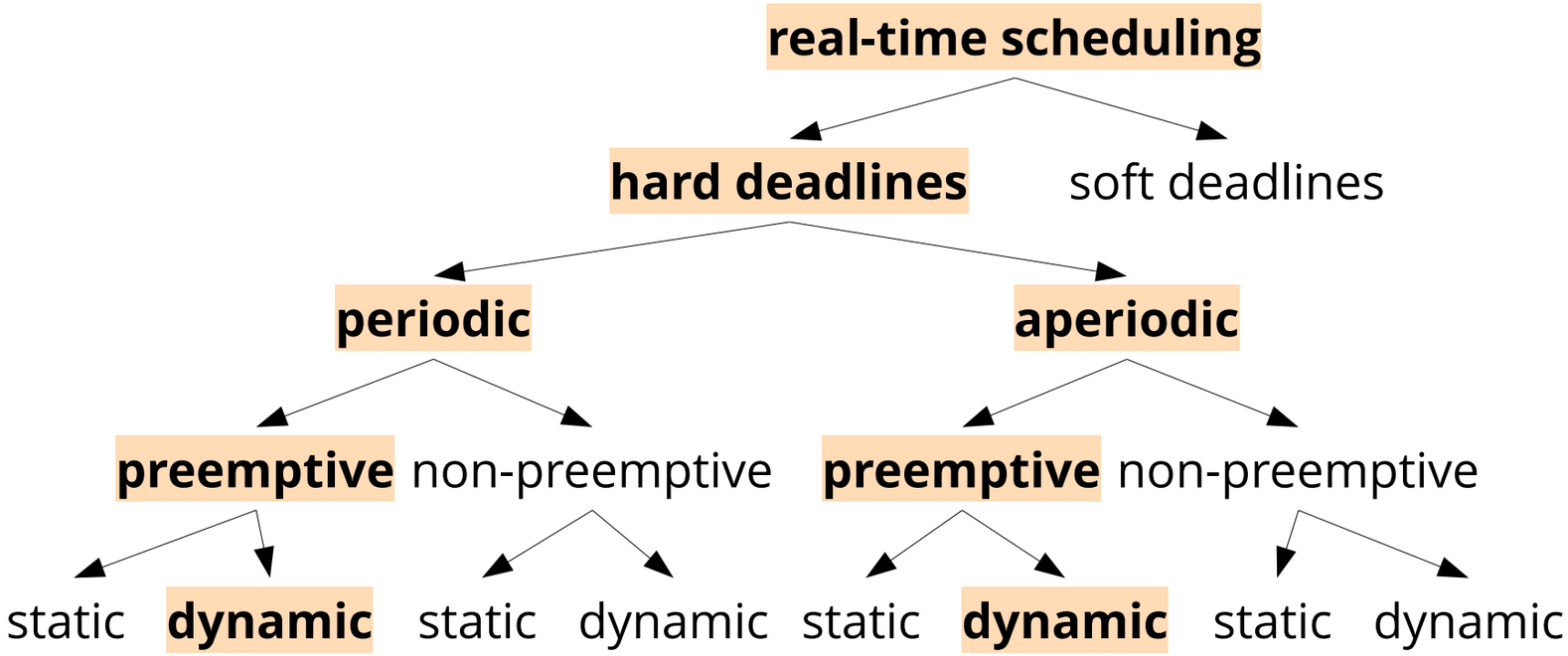
⚠ Careful:

- Assumptions A.1–5 must hold!
  - Uniprozessor, no task dependencies, …
- WCET determination problematic on modern processors
  - Memory hierarchy, out-of-order execution, DRAM access times, …
- Always consider the *whole* system.

# Overview

- Real-Time Systems
- Example: OSEKtime
- **Real-Time Scheduling Strategies**
  - *Rate Monotonic Scheduling*
  - ***Earliest Deadline First Scheduling***
- Summary and Outlook

# Example: *Earliest Deadline First* Scheduling




- ***Earliest Deadline First* (EDF)** scheduling is a scheduling strategy for preemptive, periodic and aperiodic tasks with hard deadlines. Priorities are assigned dynamically (at runtime).

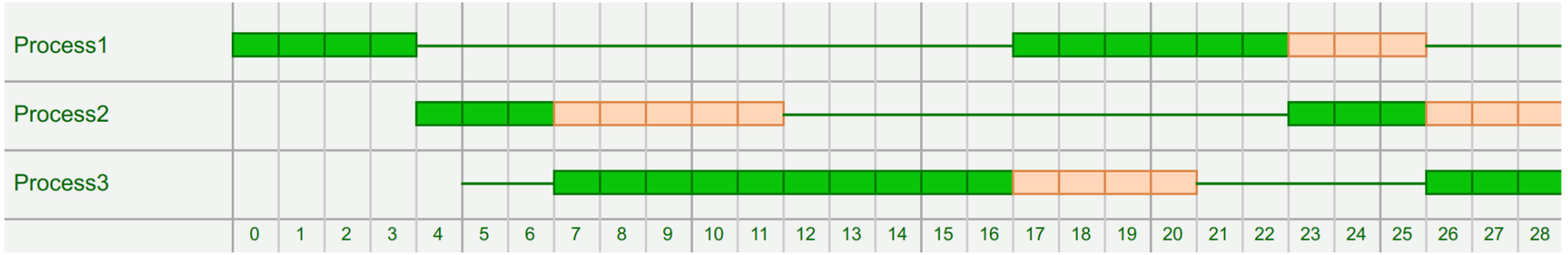
# EDF Algorithm

- EDF sorts runnable tasks by their **absolute** deadlines.
- If the first-listed task has an earlier deadline than the currently running task, it **instantly** preempts it.

However, deadlines are usually specified **relative** to the task arrival.

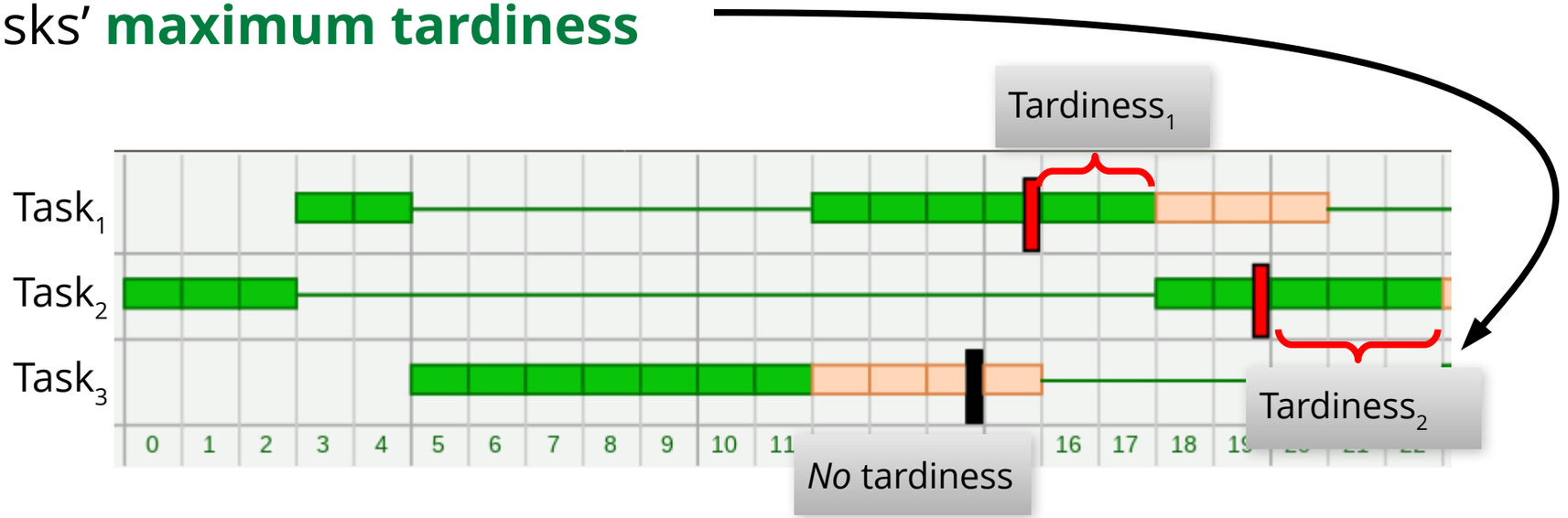
Example:

| *Process name* | Arrival | CPU burst | IO burst | Deadline |
|---|---|---|---|---|
| Process1 | 0 | 10 | 3 | 33 |
| Process2 | 4 | 3 | 5 | 24 |
| Process3 | 5 | 10 | 3-5 | 24 |



# EDF Optimality

- EDF minimizes the tasks' **maximum tardiness**




- If a schedule exists that meets all deadlines, so does EDF
  → **EDF is optimal**
  - … for independent tasks with dynamic priorities
- Holds specifically for **periodic** tasks:
  Iff $U \leq 1$, EDF finds a valid schedule (without missing deadlines).

Proof in [6]

# EDF Scheduling: Conclusion

- **Optimal** for periodic *and* aperiodic task sets
  - Higher utilization than RM scheduling by dynamic priorities

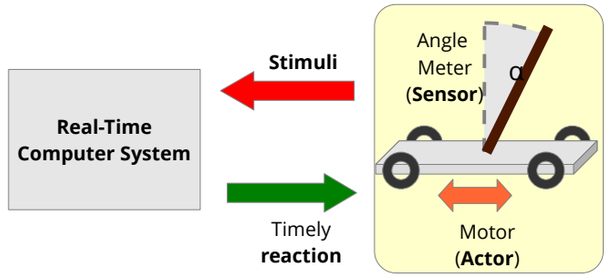
⚠ But:

  - Usually only implemented in special “real-time operating systems”
  - No guarantees regarding number of deadline misses and sum of tardinesses
  - Less predictable than e.g. RM
    - Response times can vary strongly: “jitter”
    - Overload scenarios: “domino effect”

# Overview

- Real-Time Systems
- Example: OSEKtime
- Real-Time Scheduling Strategies
  - *Rate Monotonic Scheduling*
  - *Earliest Deadline First Scheduling*
- **Summary and Outlook**

# Summary




- **Real-time systems**
  - Correctness also depends on the **point in time** a result is produced.
  - Soft, firm, hard deadlines; event- vs. time-triggered RTS
- **Offline scheduling**: Fixed task start times, “dispatcher rounds”
- **Rate-Monotonic** scheduling
  - Optimal for preemptive, periodic tasks w/ hard deadlines, fixed priorities – dynamic (online)
  - Algorithm: Small period → high priority
  - Schedulability tests: 70% rule, response-time analysis
  - Easy to implement – fixed-priority scheduler
- **Earliest Deadline First** scheduling
  - Optimal for preemptive, periodic+aperiodic tasks w/ hard deadlines, dyn. priorities – dynamic (online)
  - Task arrival with sooner absolute deadline → preemption of currently running task
  - Higher utilization, less predictable, only in special real-time OSs

# Outlook: Extending the Strategies

- Handling **sporadic tasks**
  - Limited arrival rate, but no strict period
- Handling task dependencies
- Increasing CPU utilization
  - **Mixed-critical systems**
  - Restriction to **“harmonic tasks”**: any task's period is a multiple of all shorter periods
- **Mode changes**
  - e.g., blinker / step motor gets active
- [Temporary] Overload situations
- Adaptation to [heterogeneous] multi-processor systems

# Literature


[1] Kopetz, Hermann: *Real-Time Systems: Design Principles for Distributed Embedded Applications* (2nd ed.). Springer Publishing Company, Inc., 2011. https://doi.org/10.1093/comjnl/29.5.390

[2] Automotive Open System Architecture – http://www.autosar.org

[3] Peter Marwedel. 2010. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems* (2nd ed.). Springer Publishing Company, Incorporated.

[4] C. L. Liu and J. W. Layland. 1973. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*. J. ACM 20, 1 (January 1973), 46–61. http://dx.doi.org/10.1145/321738.321743

[5] M. Joseph and P. Pandya. 1986. *Finding response times in real-time systems*, BCS Computer Journal, 29 (5): 390–395, https://doi.org/10.1093/comjnl/29.5.390

[6] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, USA, 1997.