2IMN20 - Real-Time Systems

Geoffrey Nelissen

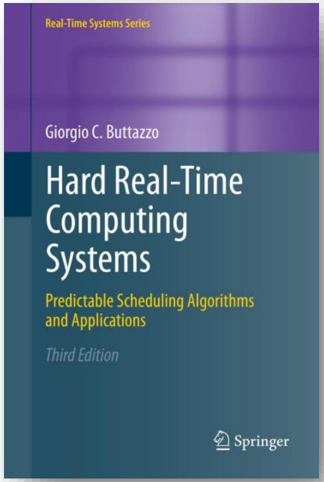
2023-2024

Contact lectures
Geoffrey Nelissen
g.r.r.j.p.nelissen@tue.nl

Contact instruction
Nidhi Srinivasan
s.srinivasan1@tue.nl

Course organization

- **Lectures**
  - Tuesday 8:45 – 10:30
  - Friday 13:30 – 15:15

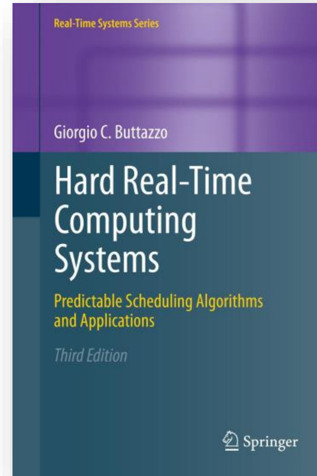


Giorgio Buttazzo

Available as e-book via the library

Course organization

- **Lectures**
  - Tuesday 8:45 – 10:30
  - Friday 13:30 – 15:15



Giorgio Buttazzo

Available as e-book via the library

- **Practical assignments:**
  - One every week (released on Friday, deadline next Friday)
  - Done in groups of 3
- **Practical sessions**
  - Tuesday 10:45 – 12:30 (starts from Nov. 21)
  - Friday 15:30 – 17:15

Note: Your presence in the practical sessions is <u>not mandatory, but it is the place to ask questions about assignments</u>

People involved

- **Responsible lecturer**
  - Geoffrey Nelissen

- **Co-lecturers**
  - Dr. Mitra Nasri
  - Nidhi Srinivasan

- **Assignments**
  - Nidhi Srinivasan (Lead TA)
  - Benjamin van Seggelen
  - Ahmed Ghanem

Who to contact

- **Questions on the course**
  - Contact: Geoffrey Nelissen (g.r.r.j.p.nelissen@tue.nl)

- **Questions on the assignements**
  - Contact : Nidhi Srinivasan (s.srinivasan1@tue.nl)

Evaluation

- **Final written exam**
  - 70% final grade
  - Minimum grade ≥ 5.0 to pass

- ***Mandatory*** **practical training**
  - 30% final grade
  - **weekly exercises** with **strict deadlines**
  - Released on Friday, due next Friday
  - groups of 3 students
  - register via CANVAS
  - Exercises available via CANVAS

Exemptions:
If you passed the assignment last year and want to carry over your grade, send an email to Nidhi

Note: **Students who already attended** the course in a previous year **cannot join a group** with **students attending for the first time** in 2023

Where to find information

Everything will be on Canvas

- Tentative schedule
- Assignments
- Group registration
- Lectures content
- Quizzes
- Extra materials
- Example exams

Course's main objectives

You are able to explain and apply the fundamental concepts of real-time systems

You are able to analyze the response time and prove the schedulability of real-time systems

Here, the analysis refers to the derivation of a theoretically sound (correct) upper bound on the worst-case response time.

You are able to design (at least partially) a real-time system that respects a given set of timing constraints

Course content

WARNING
The course **content has changed**
since last year!

- If you followed the course last year, you should have a **solid basis** for this year, **but** some **content was added**
- ➔You are **highly encouraged to attend the lectures**

Fraud is not permitted!

- You have signed the scientific code of conduct document and you are LEGALLY bound to it. Any form of fraud and/or suspicion of fraud will be reported to the exam committee and will have (severe) consequences. In particular, but not exclusively, you should be aware of:
  - **Anything you hand in MUST be your own work** and MUST contain a **proper citation** of external resources (if and when they are used).
  - You are NOT allowed to **copy/use text/figures/pictures** from publications or the Internet and **work/code** found on repositories (e.g. github, gitlab, stack overflow, studeersnel) **without a proper citation**.
  - You are NOT allowed to **copy work of others**, e.g., your fellow students, or hand in work of others as your own.
  - You are NOT allowed to **make your work available** (in any form) to fellow students.
  - You **MUST keep** your course related Internet **repositories PRIVATE**. This holds **during the course and after the course** (even after your graduation!).
- Must read references: (i) https://www.tue.nl/en/our-university/about-the-university/integrity/

  (ii) https://educationguide.tue.nl/practical-info/regulations-codes-of-conduct-and-guidelines/

Beware of breach of copyright!

- Course learning materials are Eindhoven University of Technology's copyright and you are NOT allowed to **publish any copyrighted material** of third parties in any form on Internet. In particular, but not exclusively, you should be aware of:
  - You are NOT allowed to make any **part of the course** (including exams, answers to exam questions, practical/lab assignments or their solutions) **available on Internet and/or repositories** (e.g. github, gitlab, stack overflow, studeersnel). This holds during the course and after the course (even after your graduation!).

- Must read reference: https://www.tue.nl/en/our-university/about-the-university/integrity/

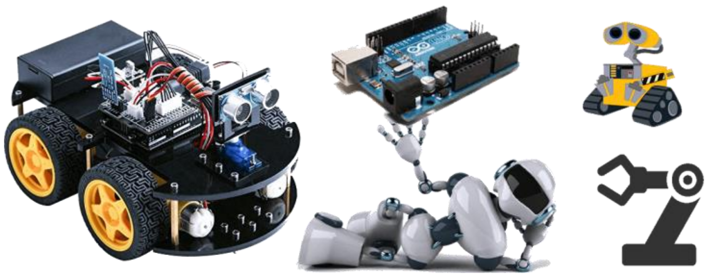
Agenda

- Course outline
- **Introduction to real-time systems**
- Modeling real-time activities

Disclaimer:

Most slides were provided by Dr. Mitra Nasri

What is an embedded system?



Embedded system:

- A computer system that is “embedded” in a physical/mechanical system and usually performs a set of specific functions.
- It often interacts with its environment.

Image source: https://www.maxfizz.com/embedded-systems-training-in-jaipur/ https://www.embedded.fm/blog/2016/12/6/an-introduction-to-the-tricky-parts-of-embedded-systems-1

IRiS

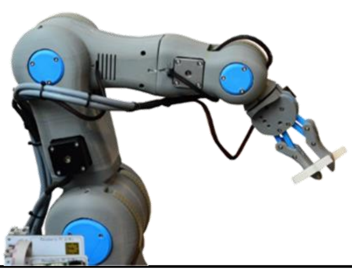
What is a real-time system?

Real-time system

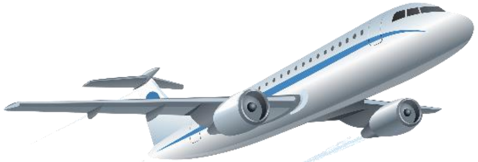
A system whose **<u>correctness</u>** depends not only on the correctness of logical results (e.g., decisions), but also on the <u>time at which the results are produced</u>.

If the system does not react "on time", something can go wrong!






IRiS

Embedded vs. real-time systems

Examples?

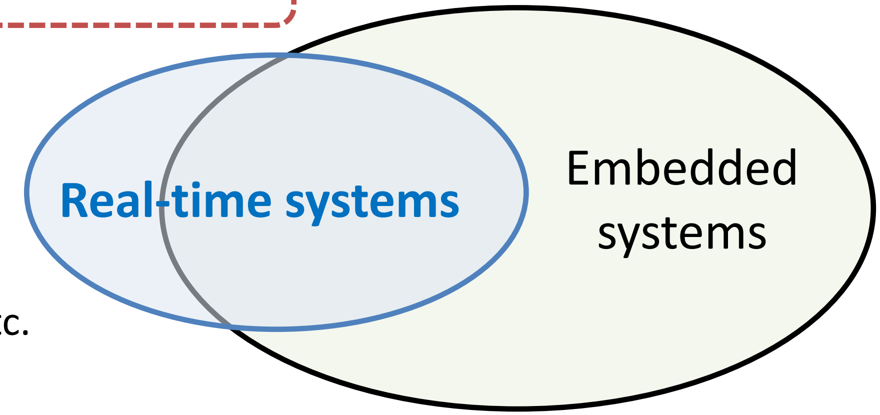
Real-time embedded:
- Nuclear reactor control
- Flight control
- Mobile phone
- Drone, Robot, Car braking system, …

Examples?

Real-time, but not embedded:
- Stock trading system
- Online advertisement selection
- Video streaming/processing

Embedded, but not real-time:
- Home temperature control
- Many home appliances: refrigerator, etc.
- Blood pressure meter

Examples?



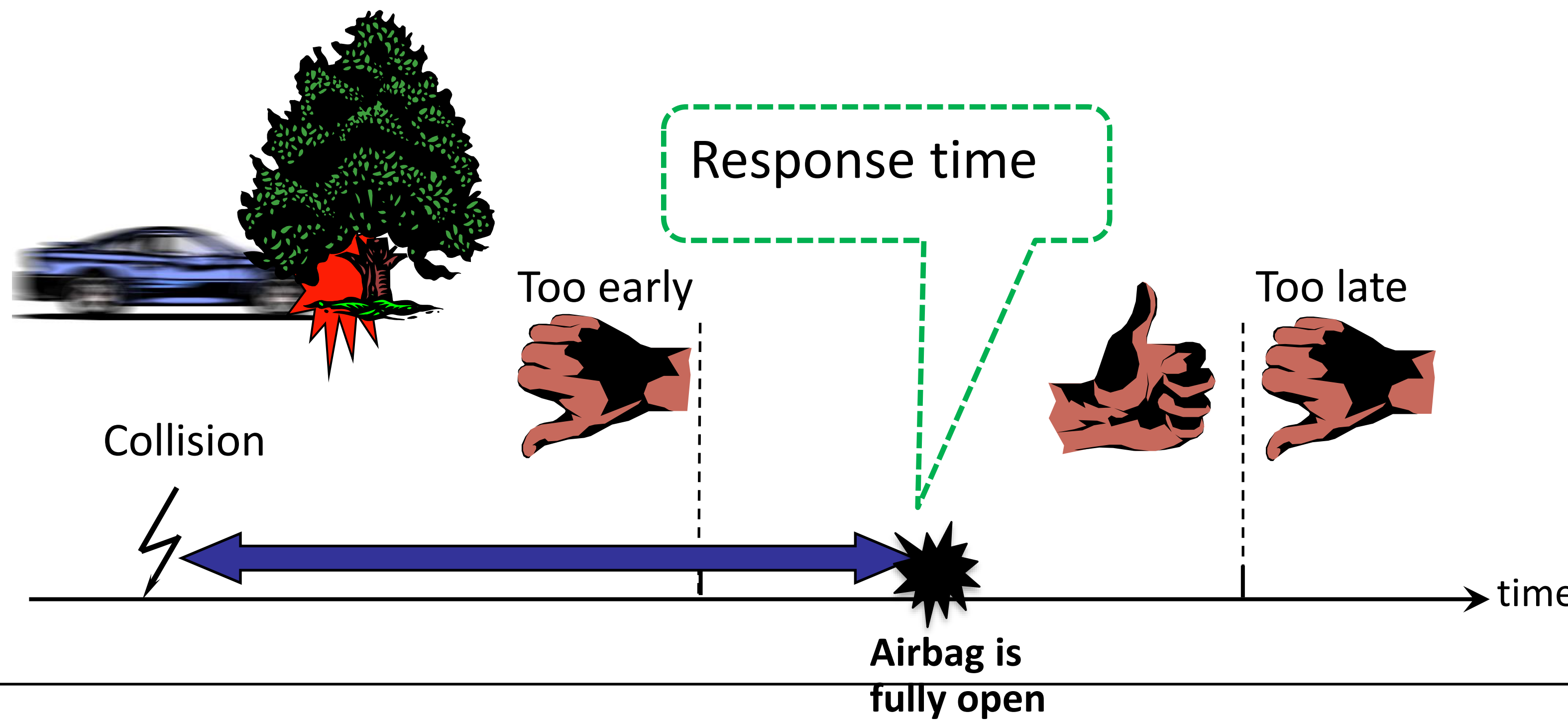
Real-time embedded systems usually interact with their environment. Since the environment is changing, this interaction must happen in a timely way. Otherwise, the actuations might happen be too late (or too early).

Characteristics of real-time systems

- **Reactive**
- **High cost of failure (safety critical)**

We care about their timeliness
(and timing predictability)

Example: airbag system



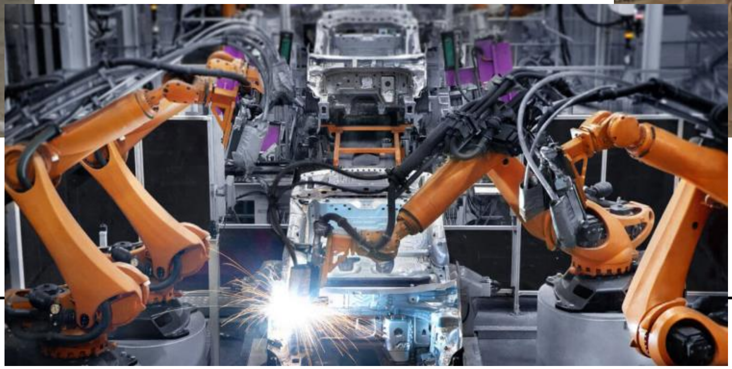
Classes of real-time systems

- **Hard** real-time

Examples?

  - Systems/tasks where it is **imperative** that responses occur within the required deadline. Otherwise, there may be **harsh consequences**.

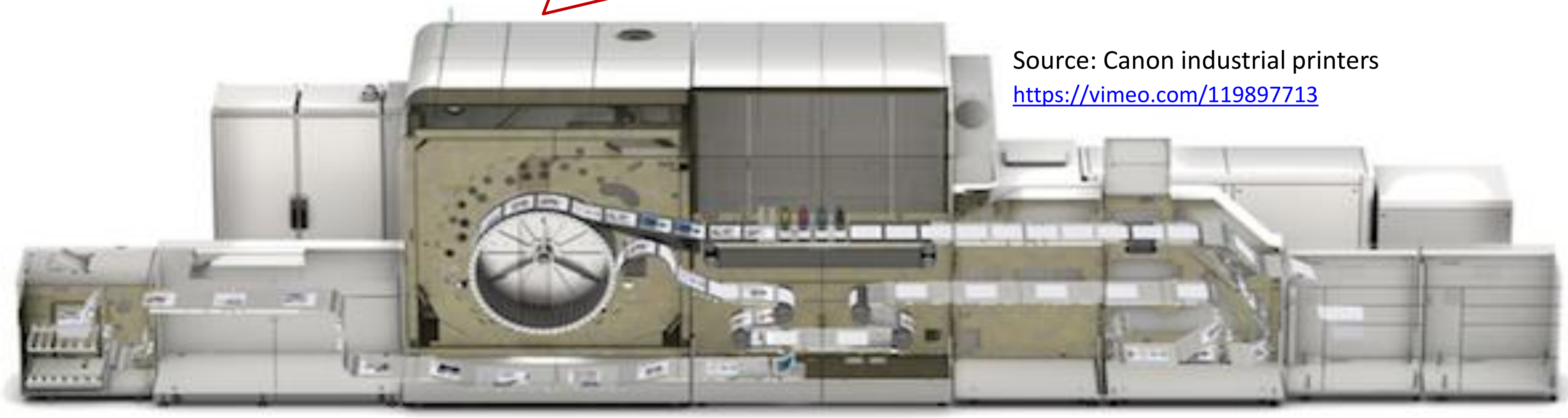
Safety-critical systems
(wrong/late decisions endanger human life or the environment's safety)



Classes of real-time systems

- **Hard** real-time
  - Systems/tasks where it is **imperative** that responses occur within the required deadline. Otherwise, there may be **harsh consequences**.

Business-critical systems
(wrong/late decisions lead to shutdowns or financial losses)



Source: Canon industrial printers
https://vimeo.com/119897713

Classes of real-time systems

- **Hard** real-time
  - Systems/tasks where it is **imperative** that responses occur within the required deadline. Otherwise, there may be **harsh consequences**.
- **Soft** real-time
  - Systems/tasks where deadlines are important. However, the system/task will still function correctly if deadlines are occasionally missed.

Examples?

wrong/late decisions reduce user satisfaction or quality of service

Gaming

Virtual reality

Video streaming

Classes of real-time systems

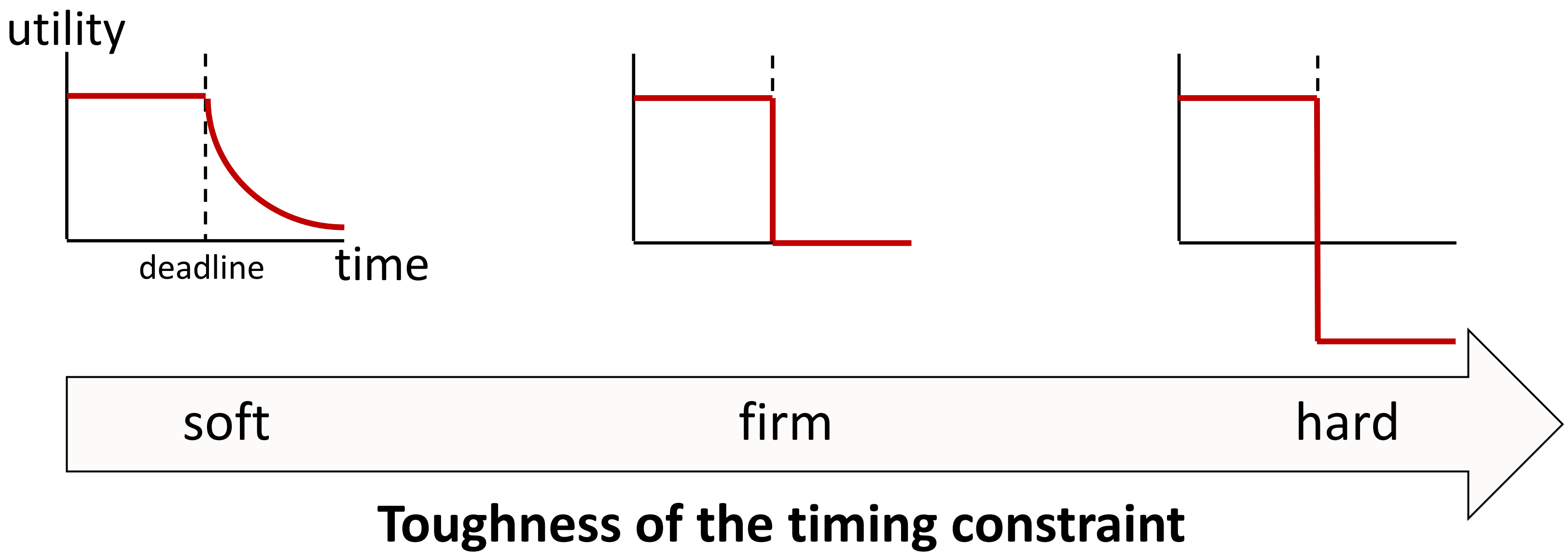
- **Hard** real-time
  - Systems/tasks where it is **imperative** that responses occur within the required deadline. Otherwise, there may be **harsh consequences**.
- **Soft** real-time
  - Systems/tasks where deadlines are important. However, the system/task will still function correctly if deadlines are occasionally missed.
- **Firm** real-time
  - Systems/tasks that are soft real-time but for which there is no benefit from late delivery of service.

Examples?

Forecasting systems, e.g., financial forecast, next action forecast in production lines or user intention forecast for QoS of service improvement

Classes of real-time systems

Utility functions



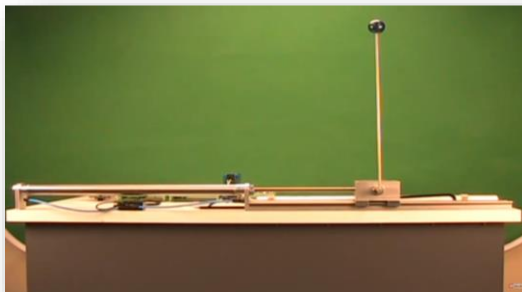
- **Soft** real-time: A response is still valuable after the deadline, but **value decreases** steadily after that.
- **Firm** real-time: Systems/tasks for which the response has **no value** after the deadline.
- **Hard** real-time: Damage is done if a response does not come in time (➔ **negative utility value**).

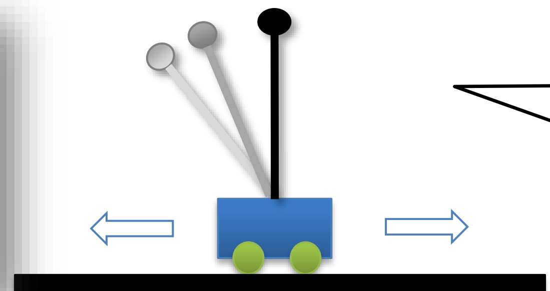
Motivation through an example

An example: a self-balancing device

Inverted pendulum:

https://www.youtube.com/watch?v=a4c7AwHFkT8

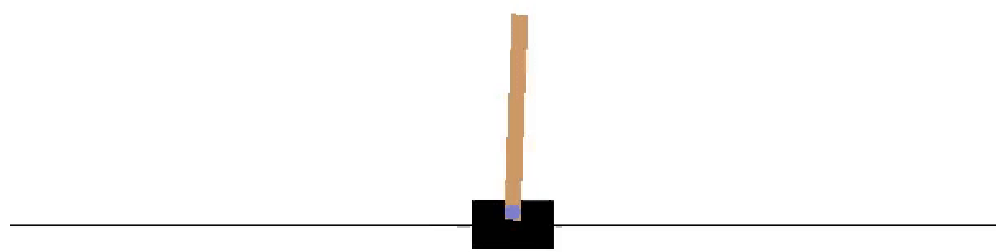




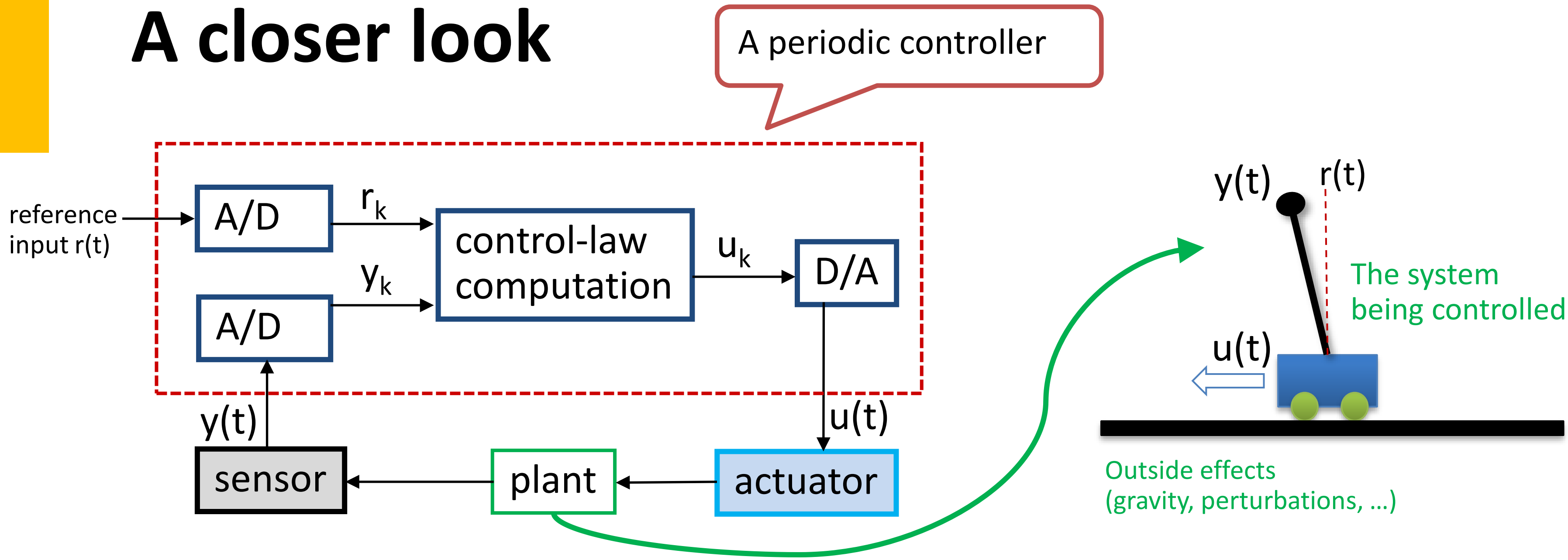
Timeliness is important for stability

Segway Ninebot One S2

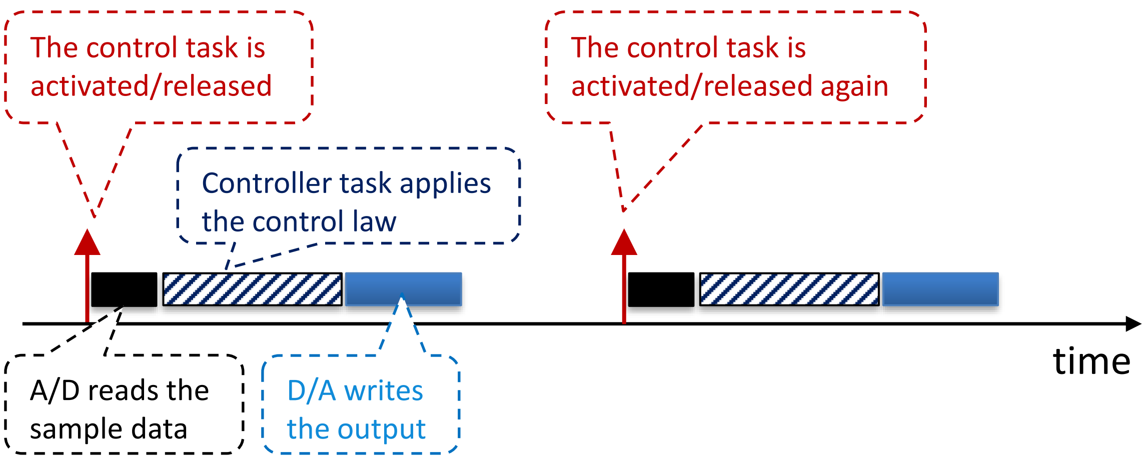
Segway X2 SE



A closer look



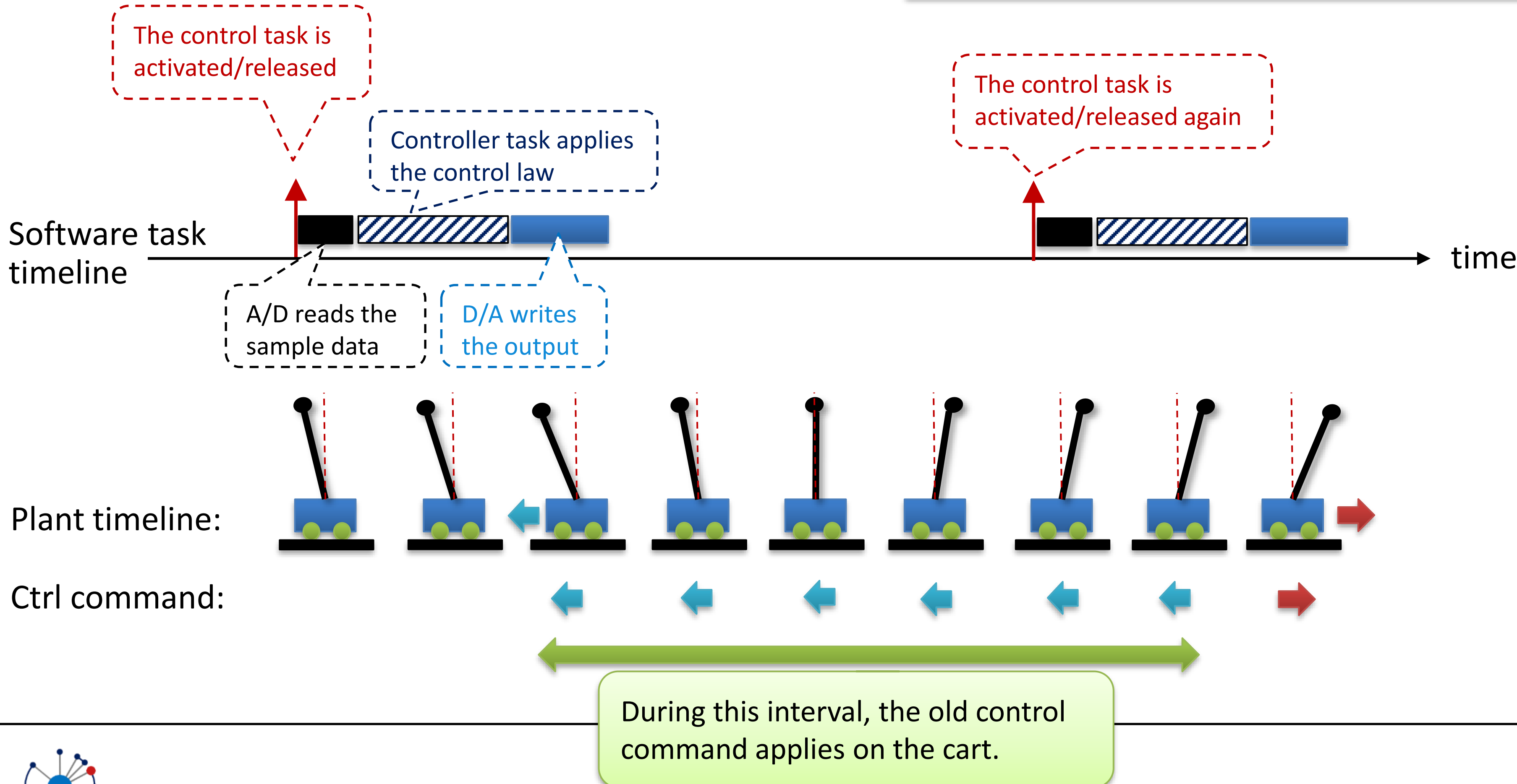
How is it implemented?



While(true)
{
Read input;
Compute control command;
Write output;
Sleep (until the next sampling period);
}

A closer look

```
While(true)
{
    Read input;
    Compute control command;
    Write output;
    Sleep (until the next sampling period);
}
```



A closer look

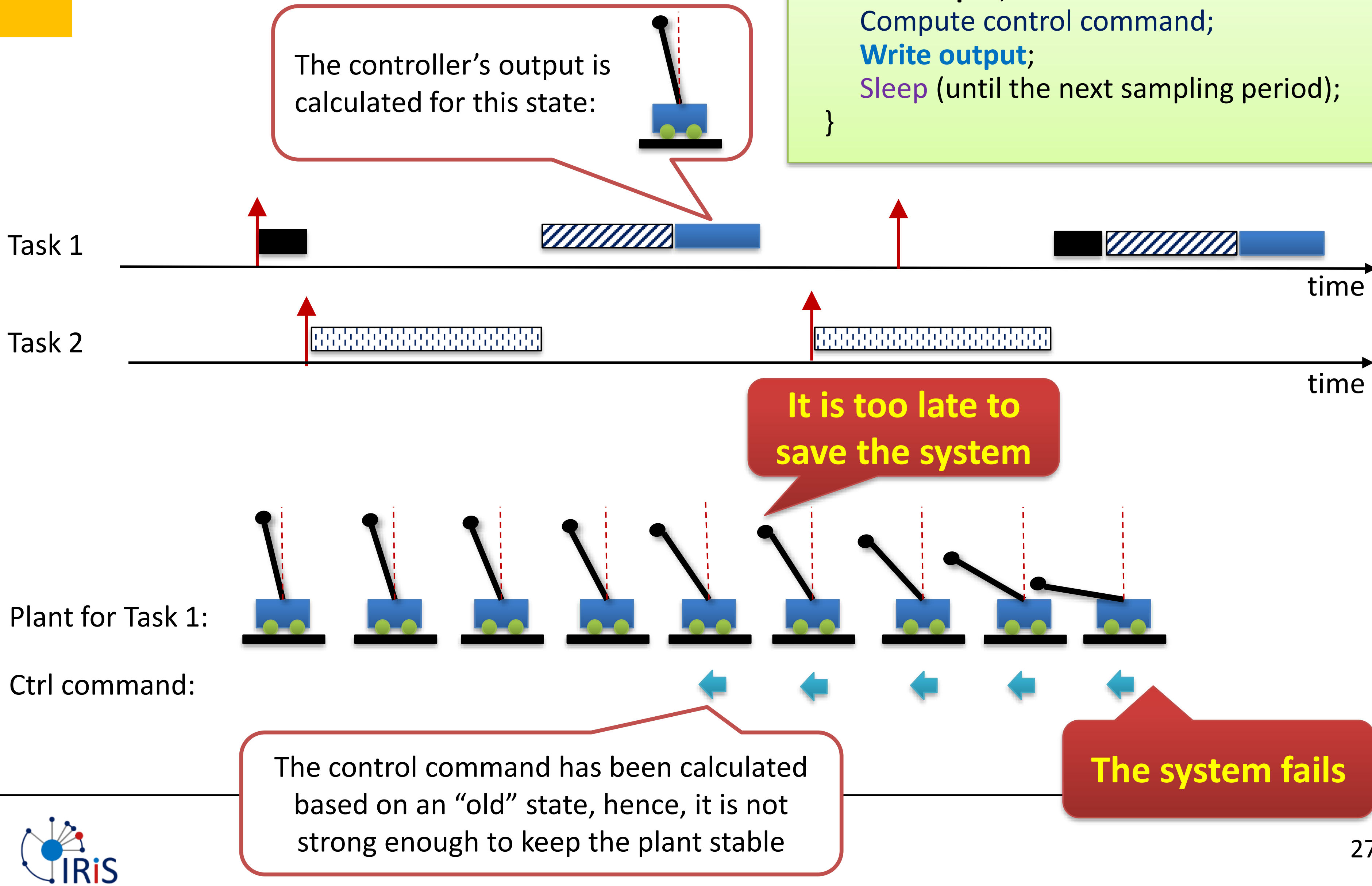
```
While(true)
{
    Read input;
    Compute control command;
    Write output;
    Sleep (until the next sampling period);
}
```

Looks predictable, right?

But what if the system contains more tasks?

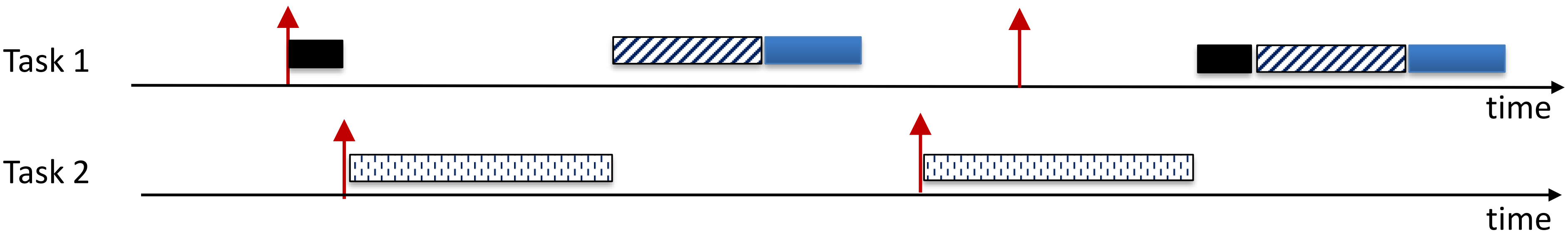
A closer look

```
While(true)
{
    Read input;
    Compute control command;
    Write output;
    Sleep (until the next sampling period);
}
```



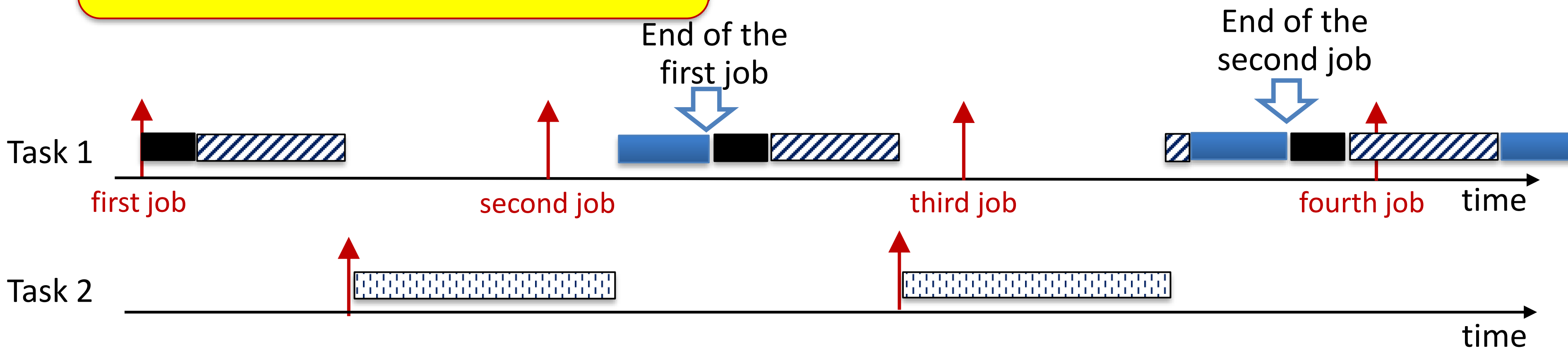
Why is this not a trivial problem?

Question: Does increasing the sampling frequency of task 1 help here?



Why is this not a trivial problem?

Question: Does increasing the sampling frequency of task 1 help here?



Why increasing sampling frequency might not help?

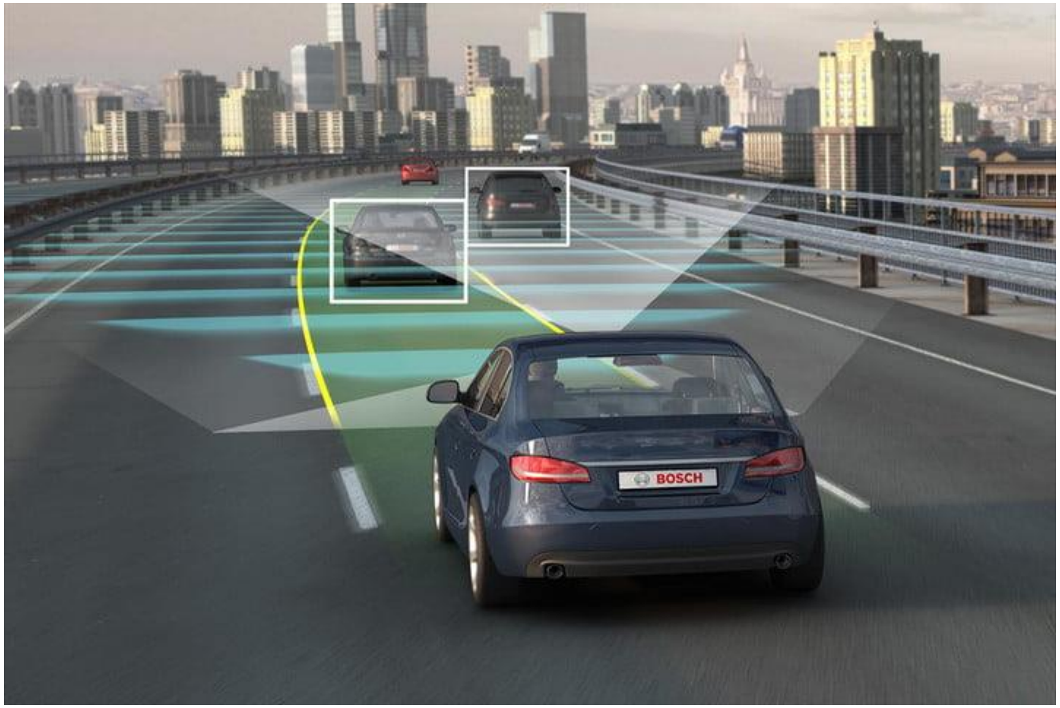
- It increases the workload on the processor
- It may result in the accumulation of unfinished work and eventually makes the problem worse

Question: Does changing priority solve the problem?

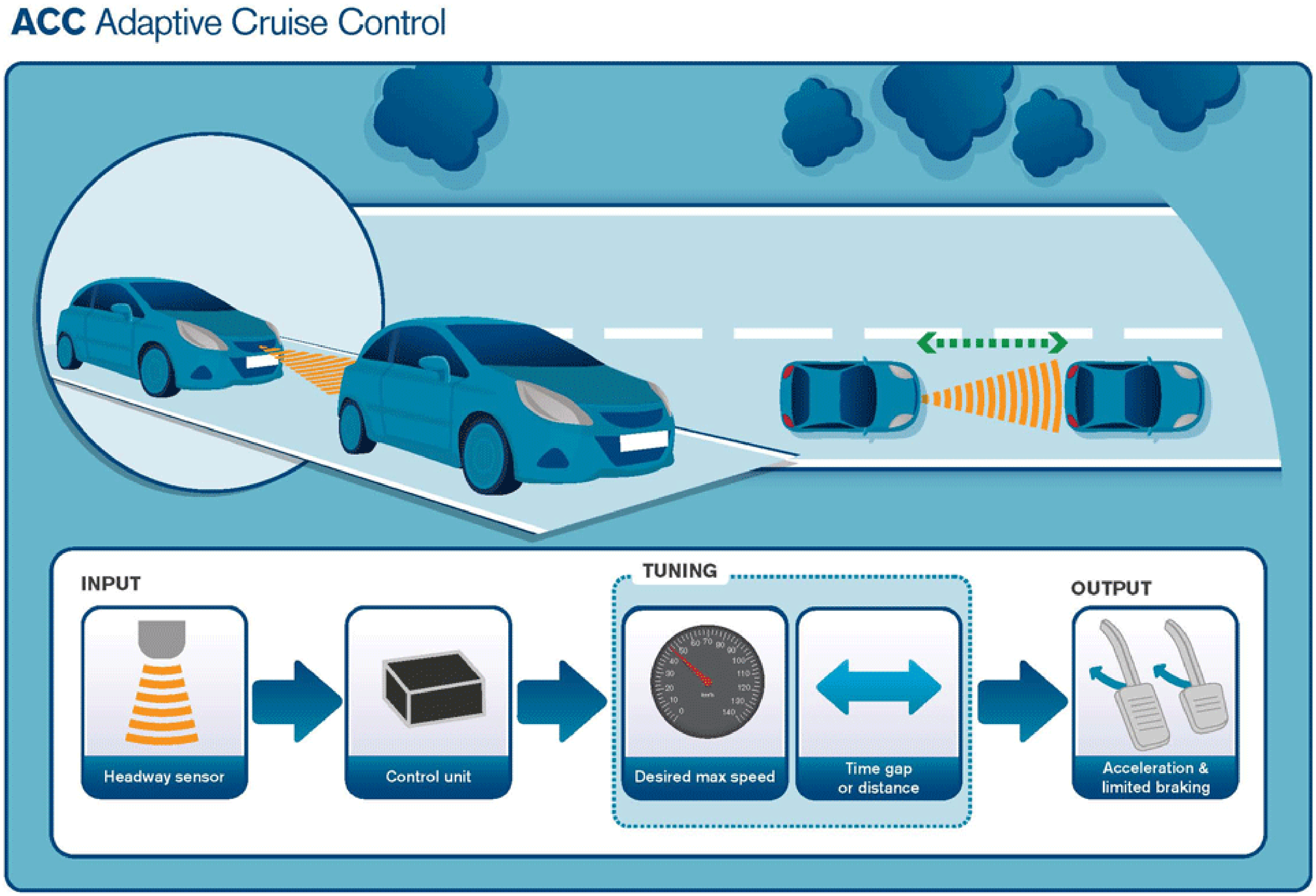
In this case, yes (for Task 1), but what about the other tasks? And what if there are more controlling task with similar requirements?

A more complicated system

Adaptive cruise control (ACC)

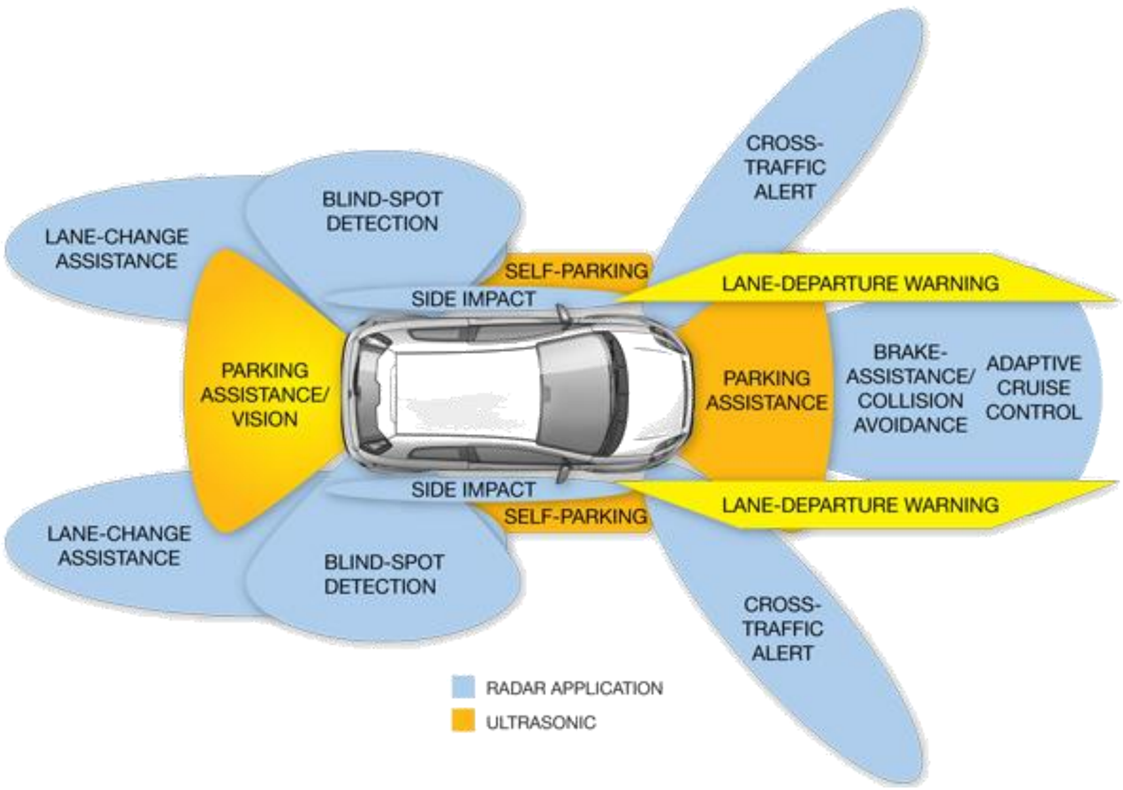


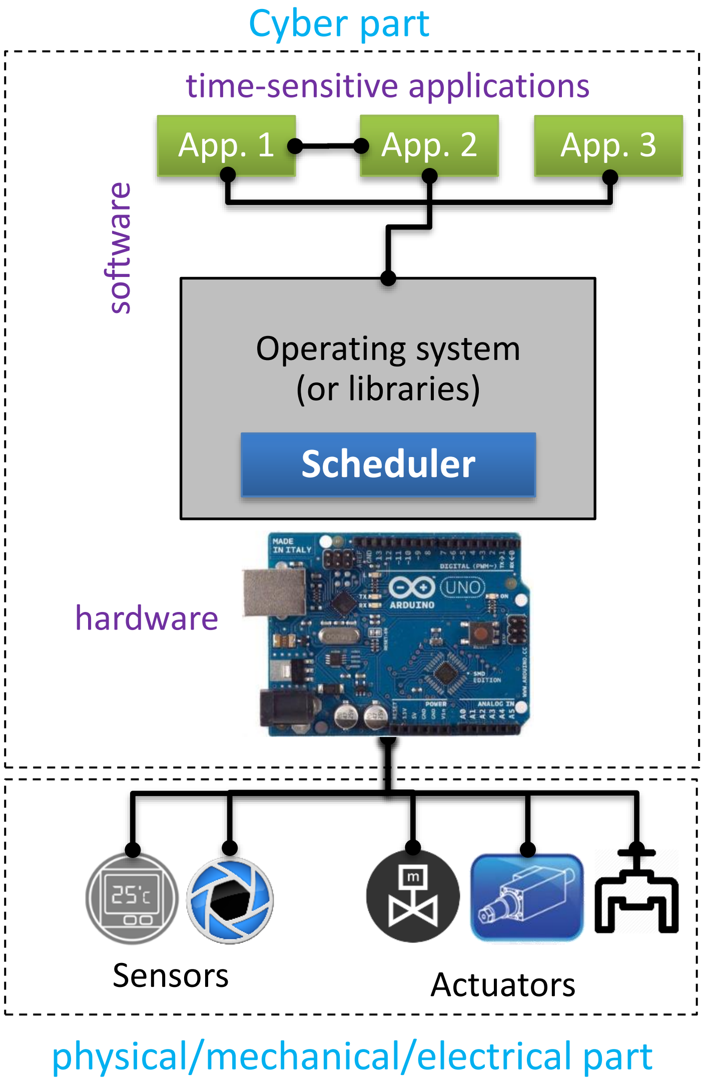
A more complicated system



Note: this is just an example. You do not need to understand these details for the exam.

Multi-task system





Murphy’s law

Anything that can go wrong, will go wrong

Accidents due to software

- 1202 alarm during LEM lunar landing (unschedulable system)
- First flight of the Space Shuttle (task wrong phasing due to time synch issue)
- Ariane 5 (overflow)
- Airbus 320
- Pathfinder (system reset caused by priority inversion)

Course focus

- **Designing predictable real-time systems**
  - fast **!=** real-time

- We will focus on
  - **Proving** (instead of testing)
  - **Worst-case/best-case response time** (instead of average-case)

10 minutes break

Summary so far

- We talked about
  - What are real-time systems
  - What makes them different from embedded systems

Agenda

- Course outline
- Introduction to real-time systems
- **Modeling real-time activities**
  - Modeling task execution time
  - Modeling job releases

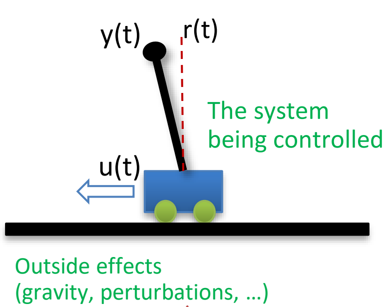
Modeling real-time activities

A **model** is a representation of something. It does not capture all attributes of the represented thing, but rather those that are relevant for a specific purpose.

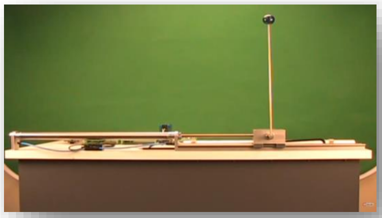
It should be **expressive**
(an accurate representation of reality)

It should be **tractable**
(provide results in a bounded time)

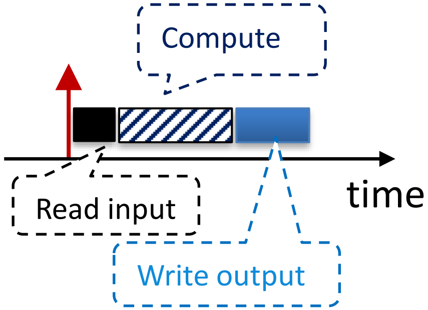
Example:



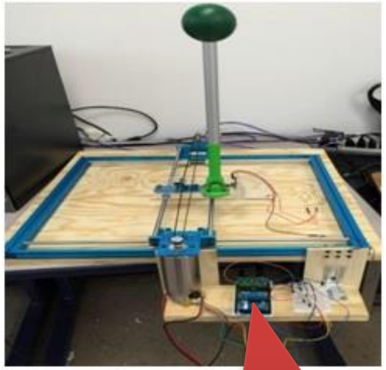
A model



The actual system

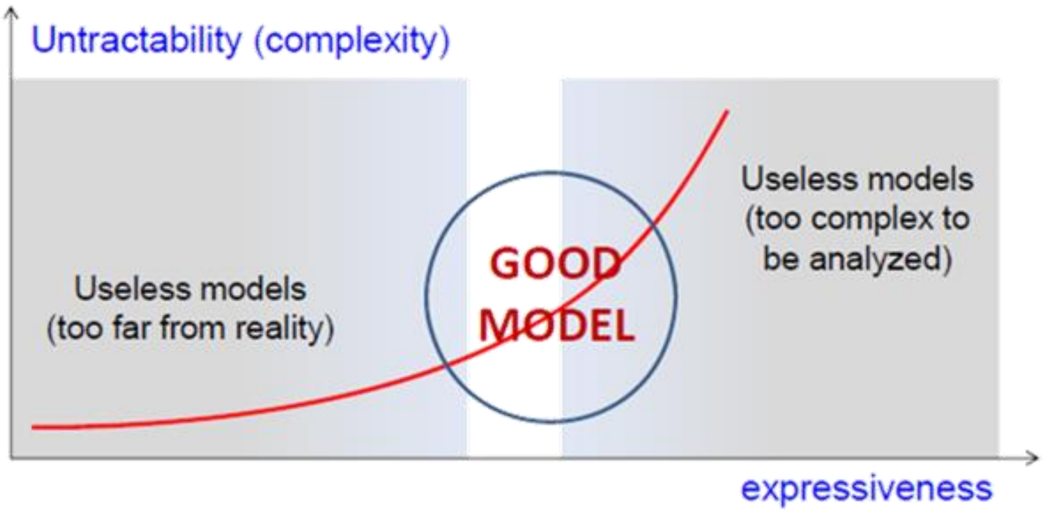


A model



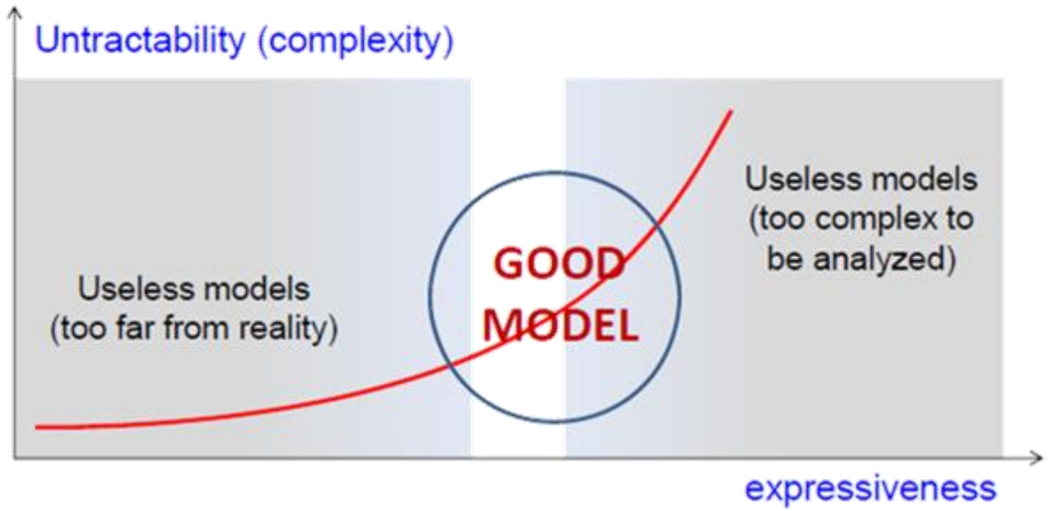
The actual control system

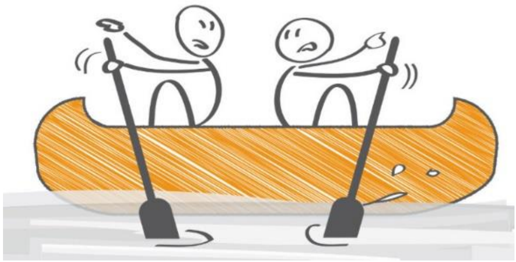
Modeling real-time activities



- If we try to include a lot of details in the model, we will not be able to analyze it.
- If we give up on details of the model, our results may either become too pessimistic or useless (far from reality)

Modeling real-time activities





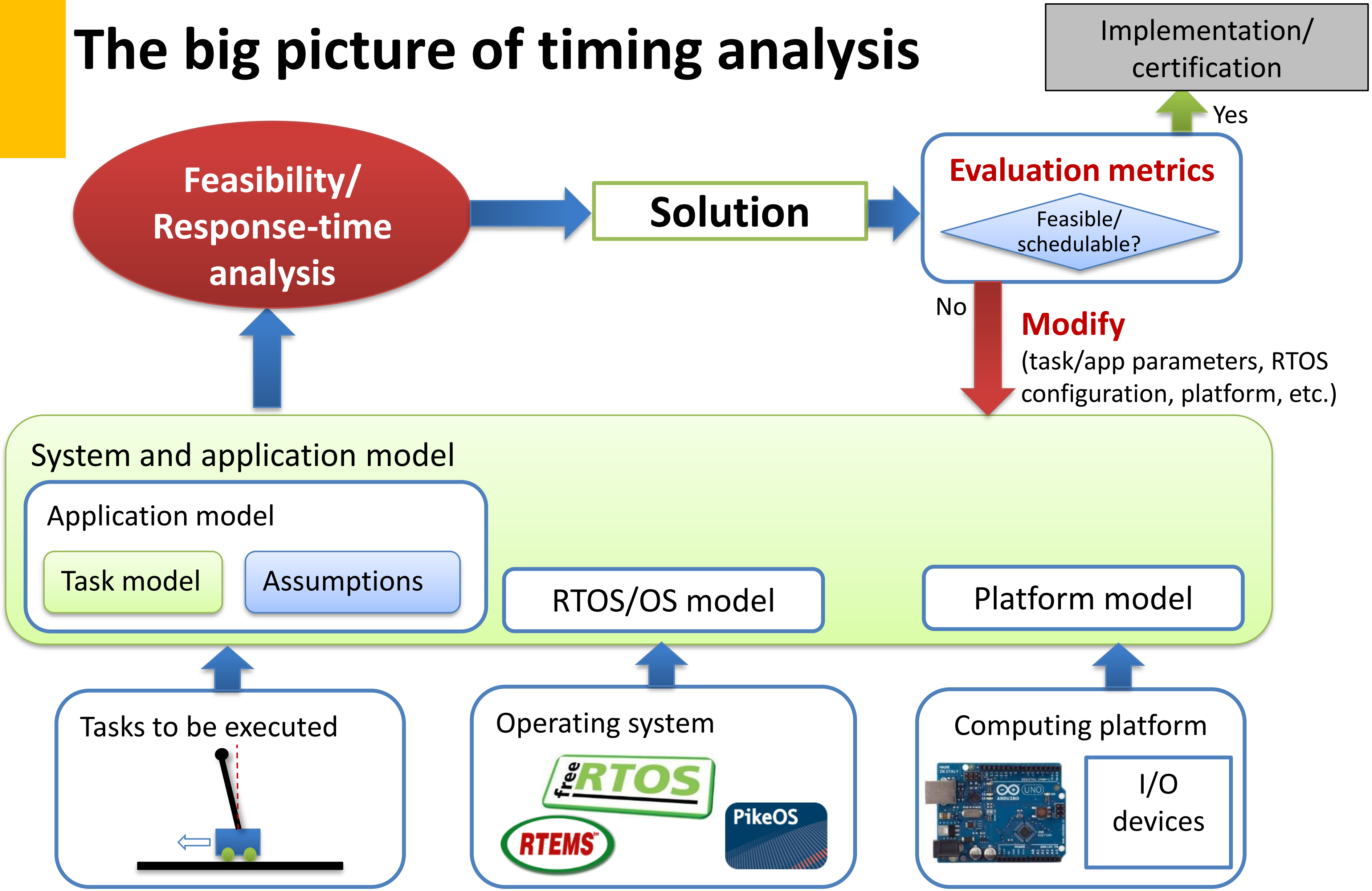
"All models are WRONG!
But some of them are useful!"

George Box

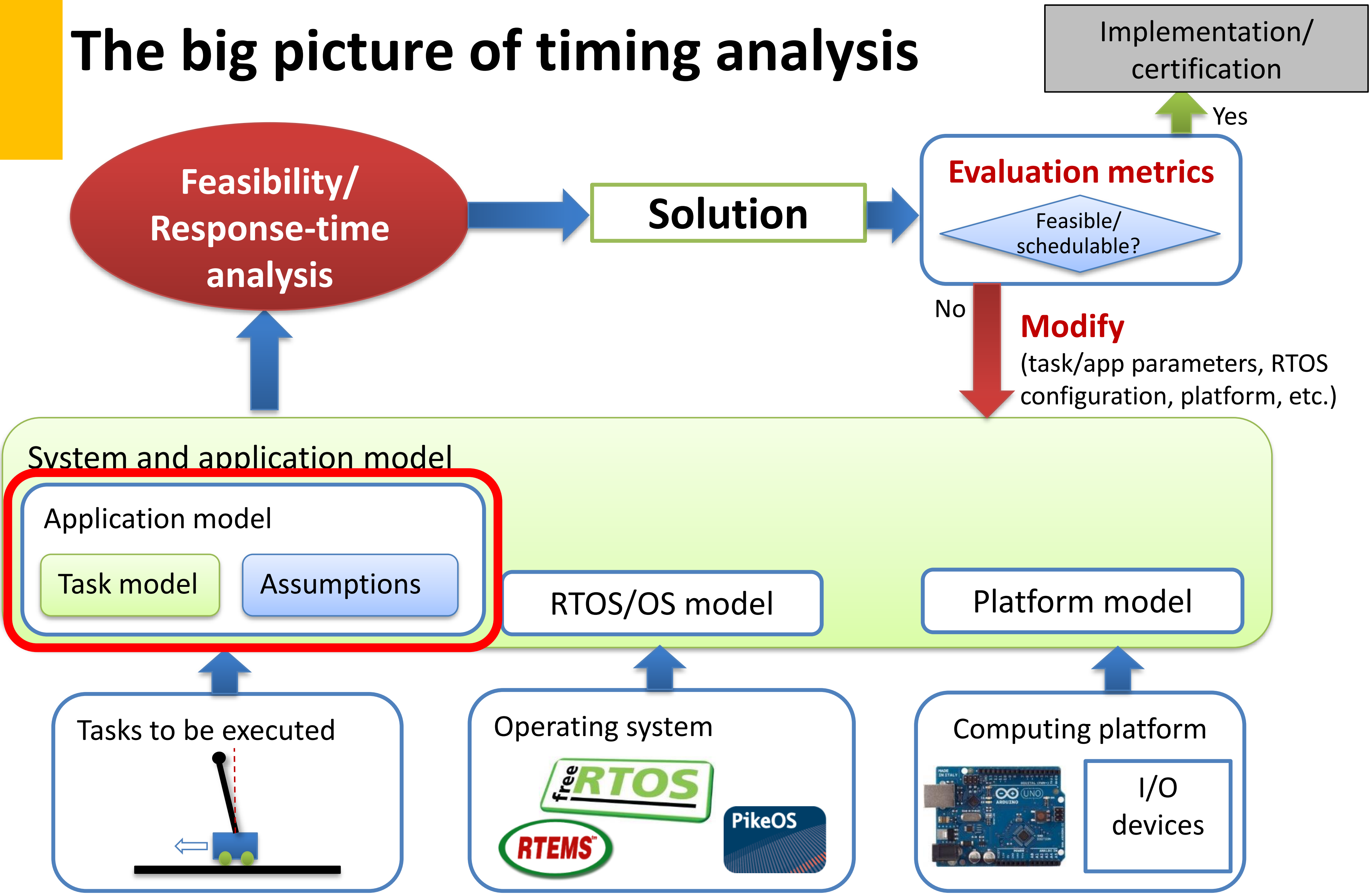
Important aspects in building a model

- Clearly identifying the assumptions you need to simplify reality (but don't simplify too much);
- Defining the variables that characterize the model.
- Defining the metrics for evaluating the outputs of your system and its performance.

The big picture of timing analysis



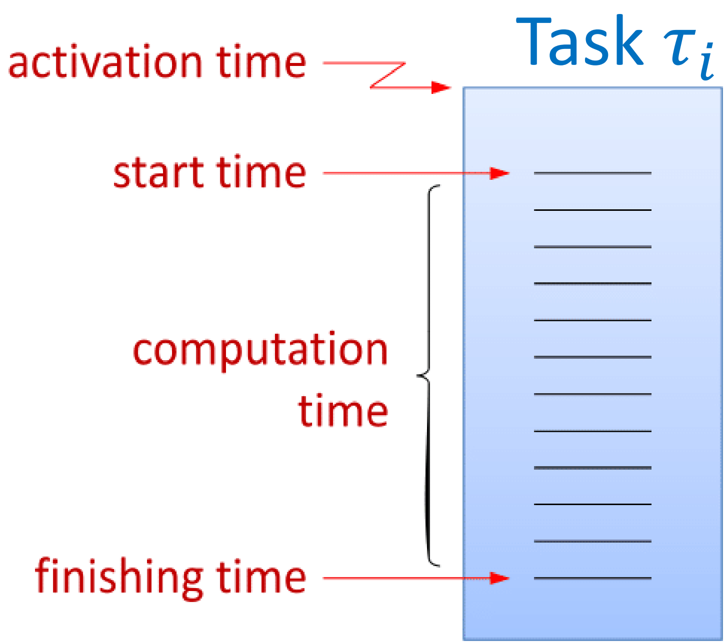
The big picture of timing analysis



Task's timing model

Task: a **sequence of actions** that must be carried out to, e.g., implement a functionality or respond to an event

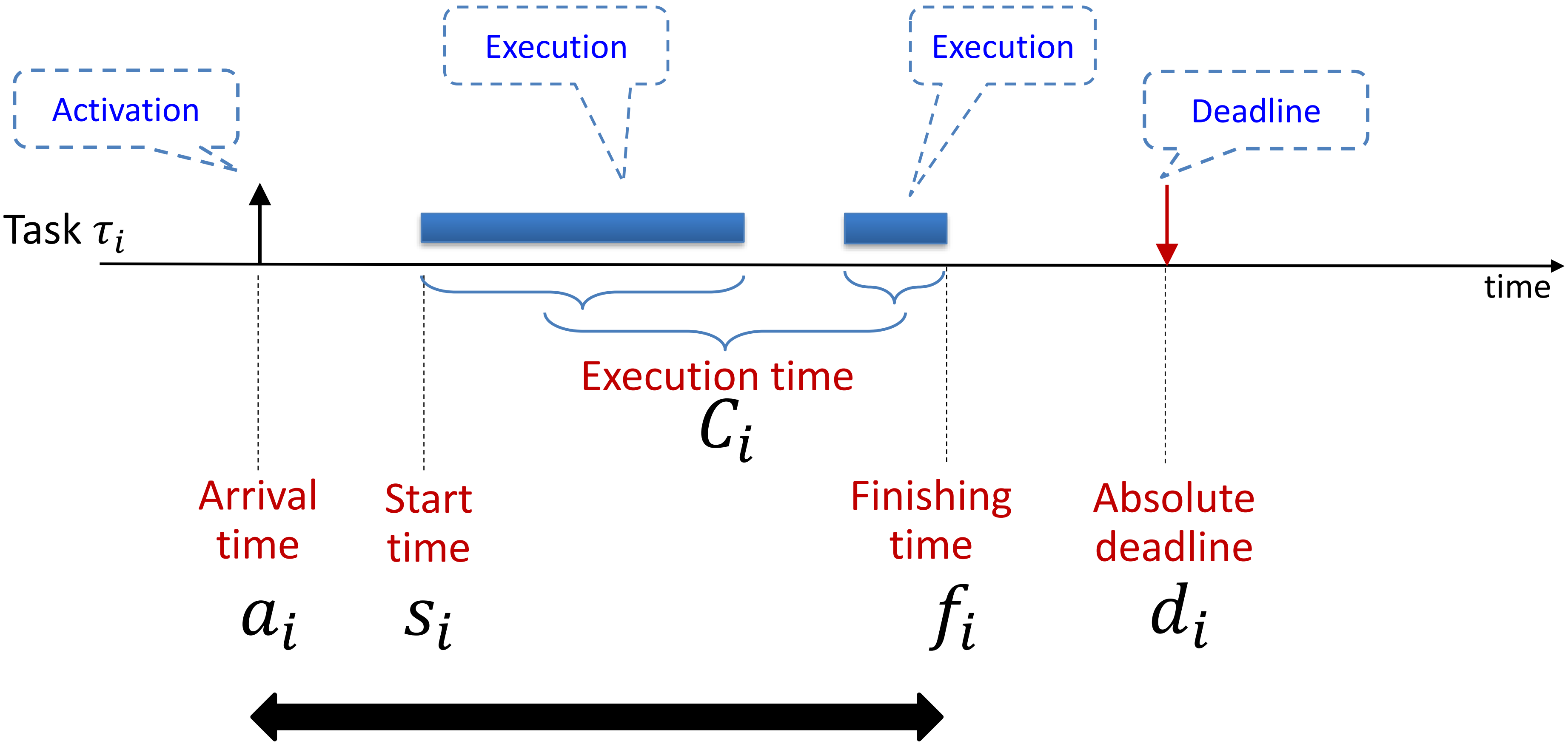
In **this course**, we primarily **assume tasks** are implemented as a **piece of code** that must be **executed on a processor**




```
While(true)
{
    int temp = readTemperature();
    if (temp > 42)
        sleep (100, ms);
    else
    {
        int * array = readData();
        sortData(array);
        sendLargestData(array);
    }
    sleep (100, ms);
}
```

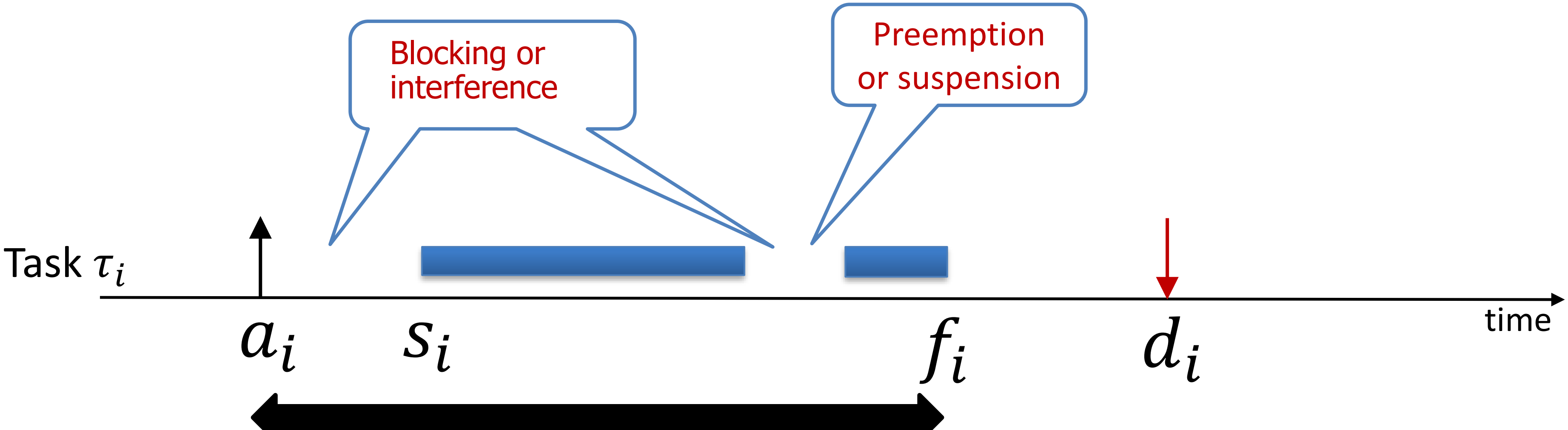
Delay until **next activation**

Task's timing model



Response time $R_i = f_i - a_i$

What does impact the response time?



Response time $R_i = f_i - a_i$

- A task may get preempted (paused and removed from the processor) by the **scheduler**
- A task may be interfered with or blocked by other tasks in the system
- The execution time of the task largely impacts its response time

Question: how to calculate the execution time?

Modeling execution time

```
While(true)
{
   int temp = readTemperature();
   if (temp > 42)
         send(-1);
   else
   {
         int * array = read10Data();
         int max = -1;
         for (int i=1; i < array[0]; i++)
               if (max < 0 || array[i] > max)
                     max = array[i];
         send(max);
   }
   sleep (100, ms);
}
```

Question: What influences the execution time of a task?

Software aspects

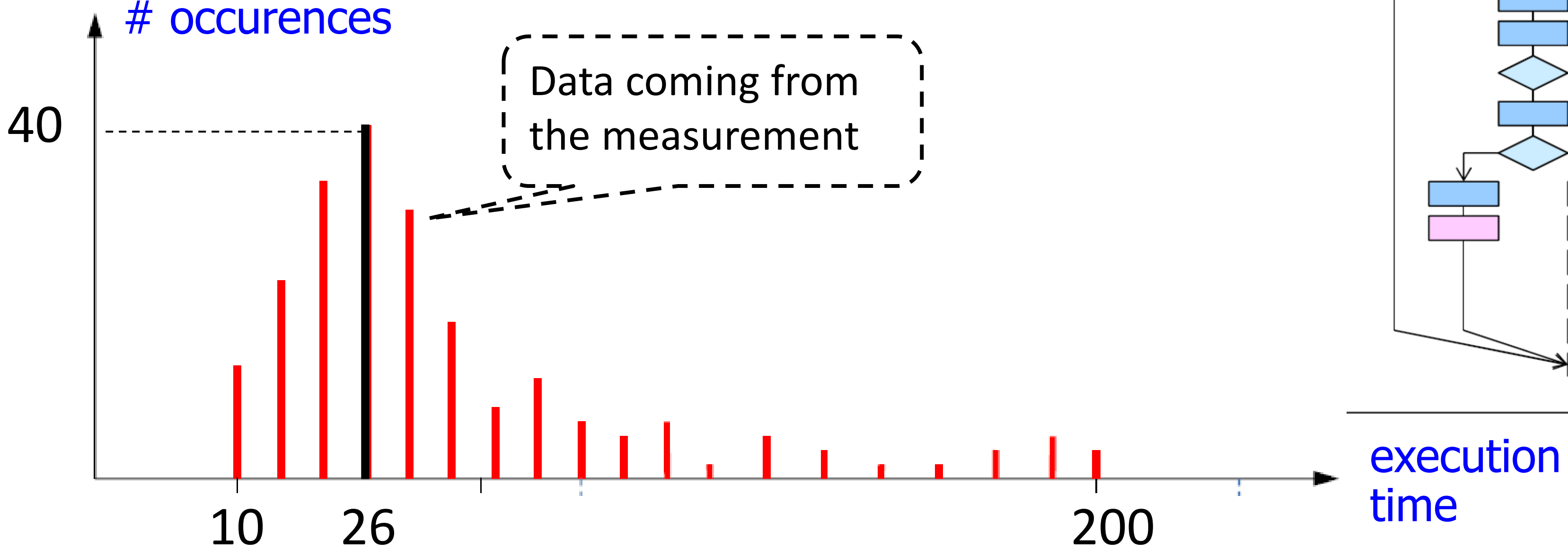
- Input value
- Program path (branches)
- Number of iterations in loops

Hardware aspects

- Processor architecture
- Branch predictors
- Out-of-order pipeline execution
- Cache misses
- Contention on memory bus, …

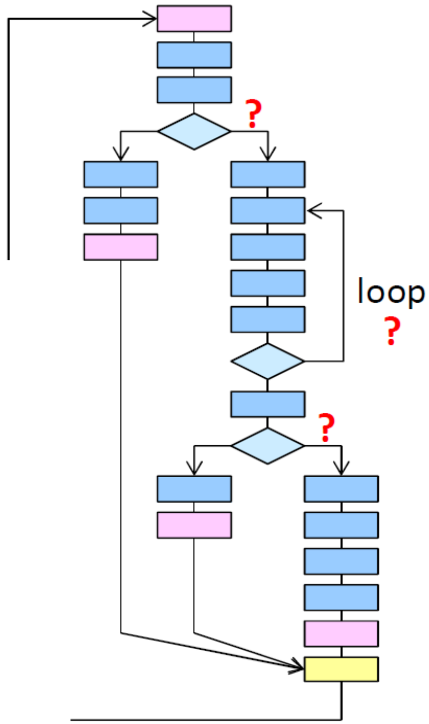
How to find (a safe) C_i?

We can "measure"..... Can we?



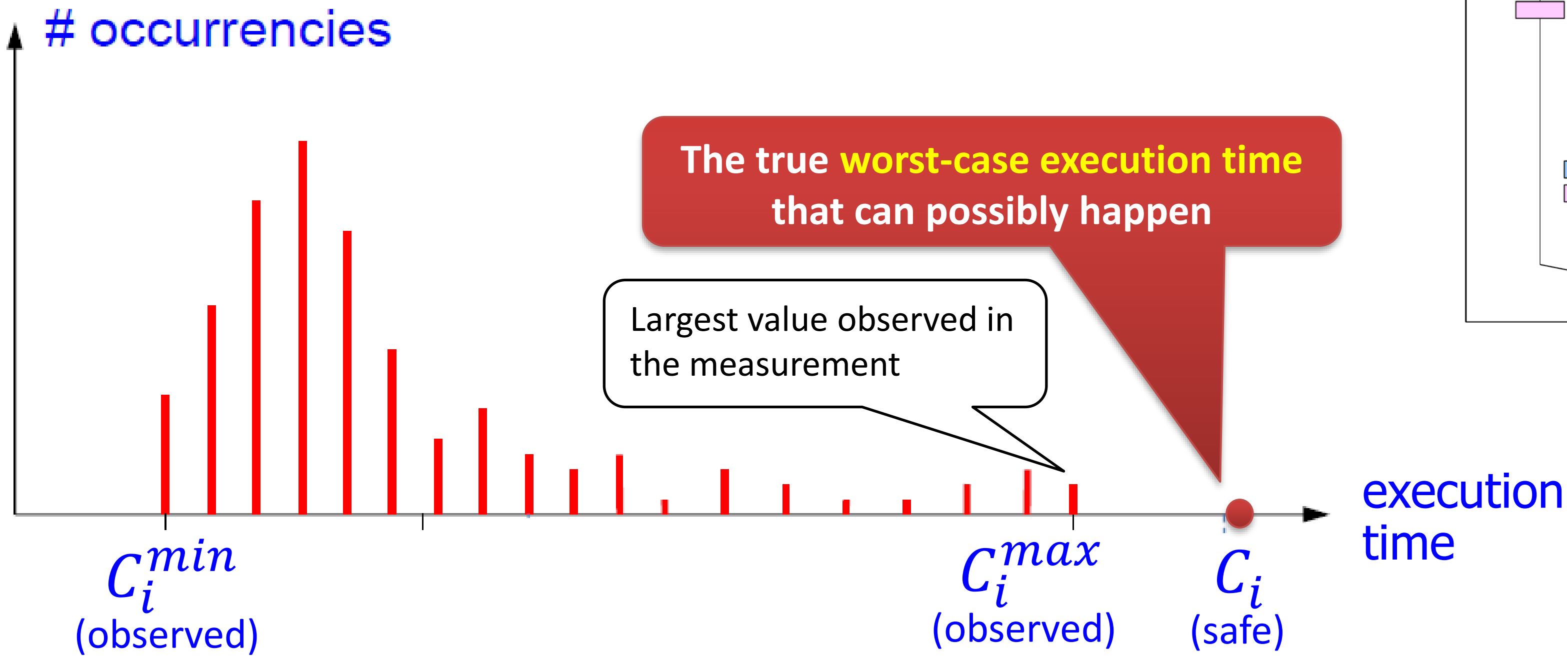
Data coming from the measurement

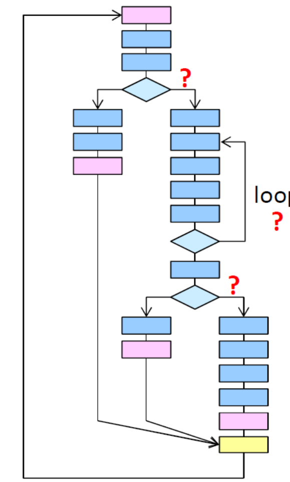
40 times out of 100, the measured execution time was 26



Question: what is the worst-case execution time and why?

Measurement-based WCET estimation is never fully safe!





Estimated using a multiplying safety factor, a **probabilistic analysis** or using a **static timing analysis** of the task flow diagram

Agenda

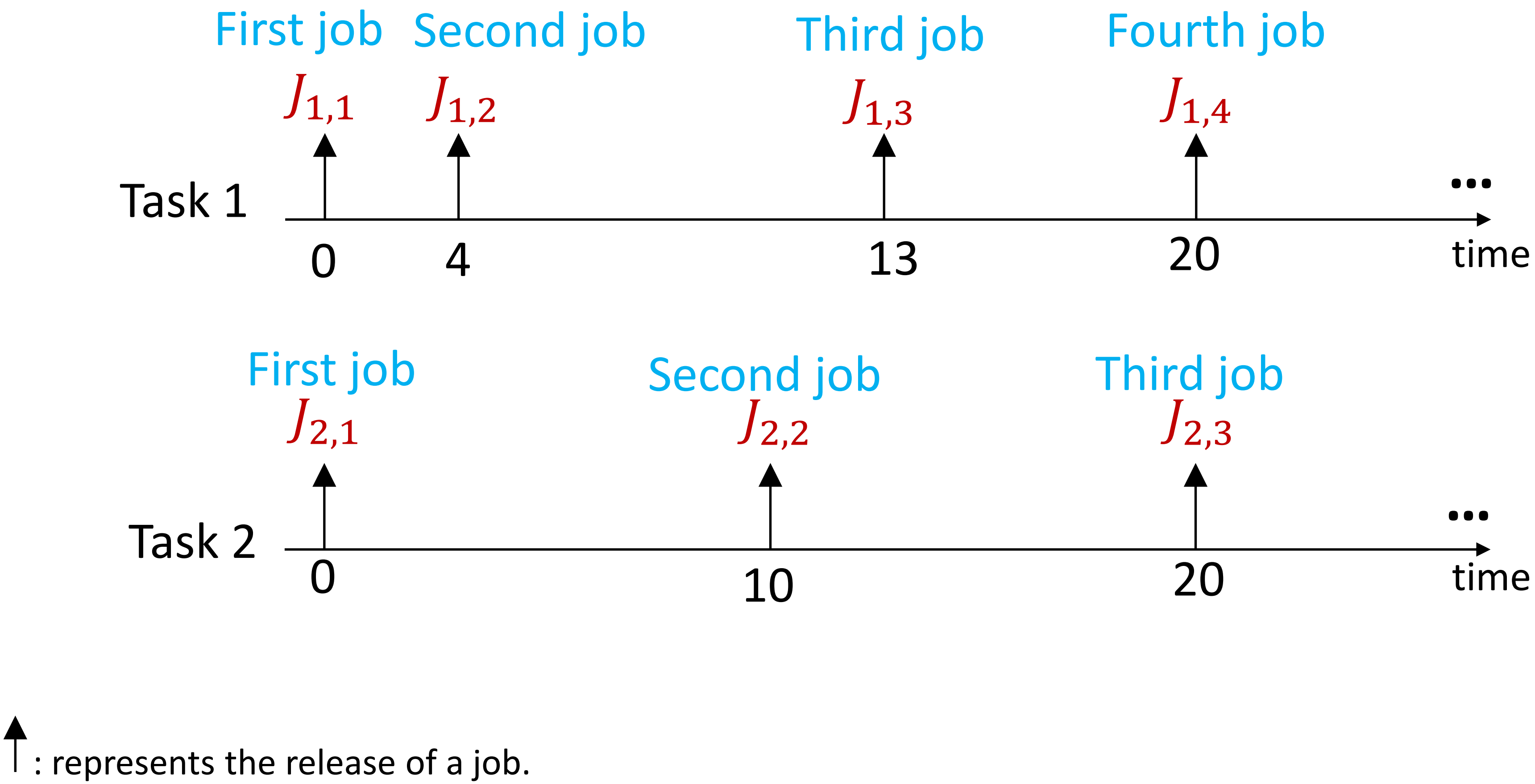
- Course outline
- Introduction to real-time systems
- **Modeling real-time activities**
  - Modeling computation
  - Modeling task execution time
  - Modeling job releases

Arrival model of a task

- **A task releases a sequence of jobs to be executed.**

Job

- A **Job** is an instance of a task (a task releases jobs)



Arrival model of a task

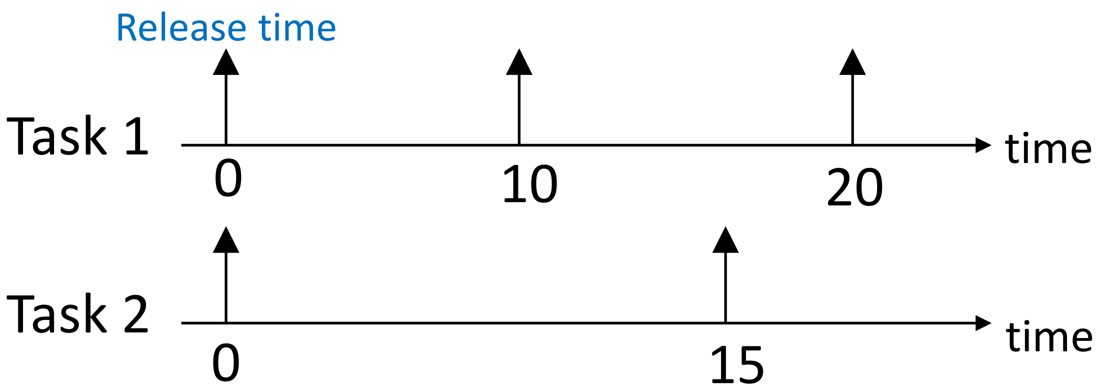
- **A task releases a sequence of jobs to be executed. Job may be released**
- **Periodically**
  - Fixed inter-arrival time between jobs
- **Sporadically**
  - There is a *minimum-inter-arrival time* between consecutive jobs of a task
  - These tasks have lower-bounded inter-arrival time
- **Aperiodically**
  - Unbounded (no lower or upper-bound on) inter-arrival time (**event-based activation**)

Time-triggered activation

Event-triggered activation

Periodic tasks

- Each task (or group of tasks) executes repeatedly with a particular period
- Periodic tasks are widely used, in particular, in **control systems**



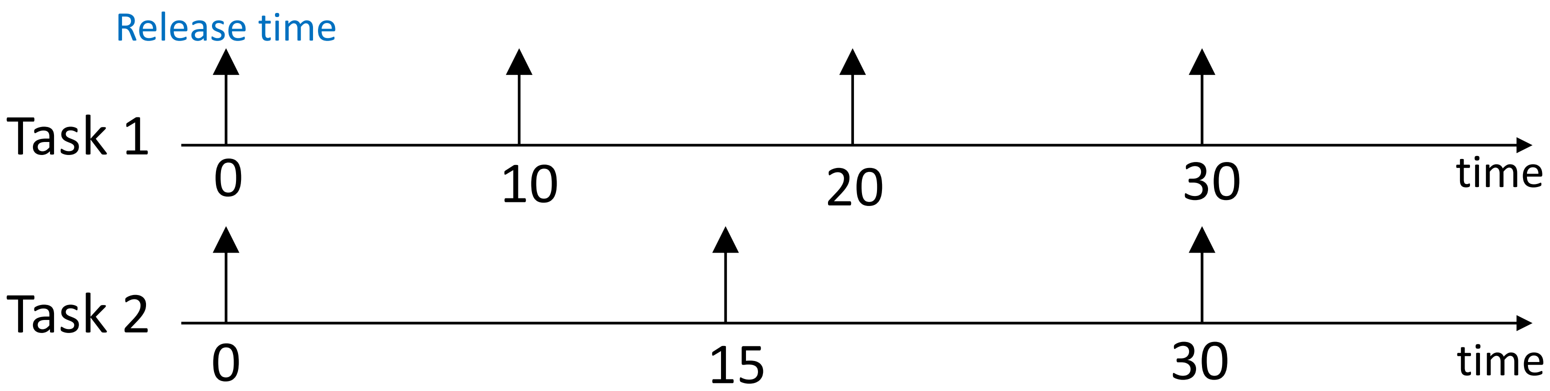
Question: What are the periods?

The period of Task 1 is 10 and Task 2 is 15.

↑ : represents the arrival of an instance of a task (a.k.a. **job**).

Periodic tasks

- Each task (or group of tasks) executes repeatedly with a particular period
- Periodic tasks are widely used, in particular, in control systems



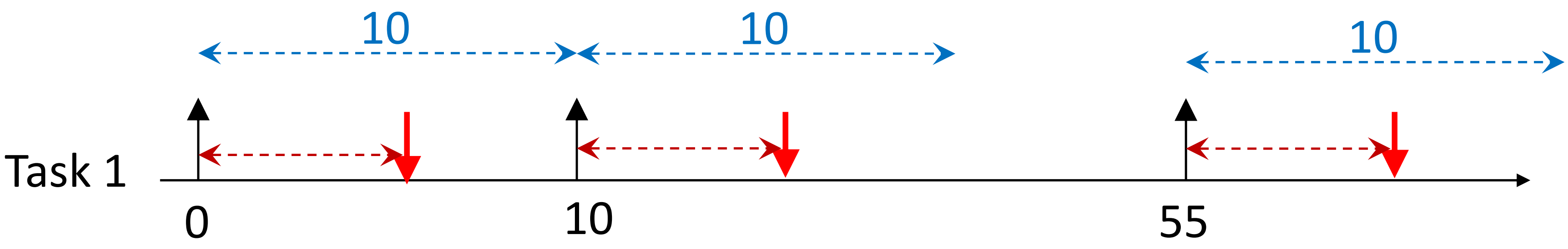
Question: is there any moment in time from which the arrival pattern of both tasks repeats?

Yes. At time 30. It is called the **"hyperperiod"** of the tasks set.

: represents the arrival of an instance of a task (a.k.a. **job**).

Sporadic tasks

- A sporadic task is a task whose next release time is **lower bounded** by a value called **"minimum inter-arrival time"**

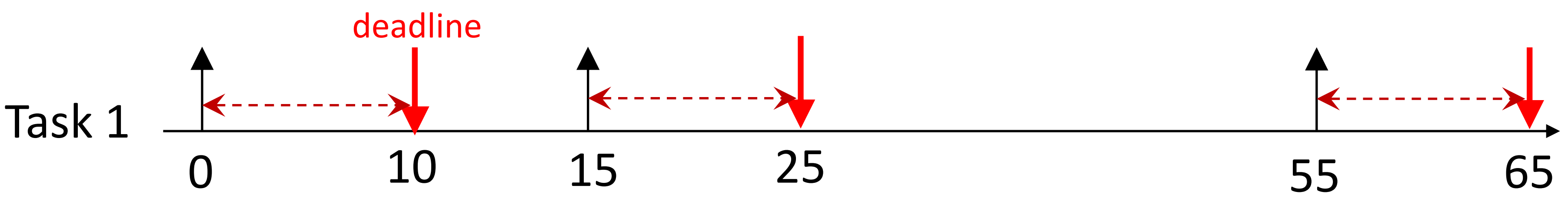


Minimum inter-arrival time = 10

↑: represents the release of a job. ↓: represents the deadline of a job.

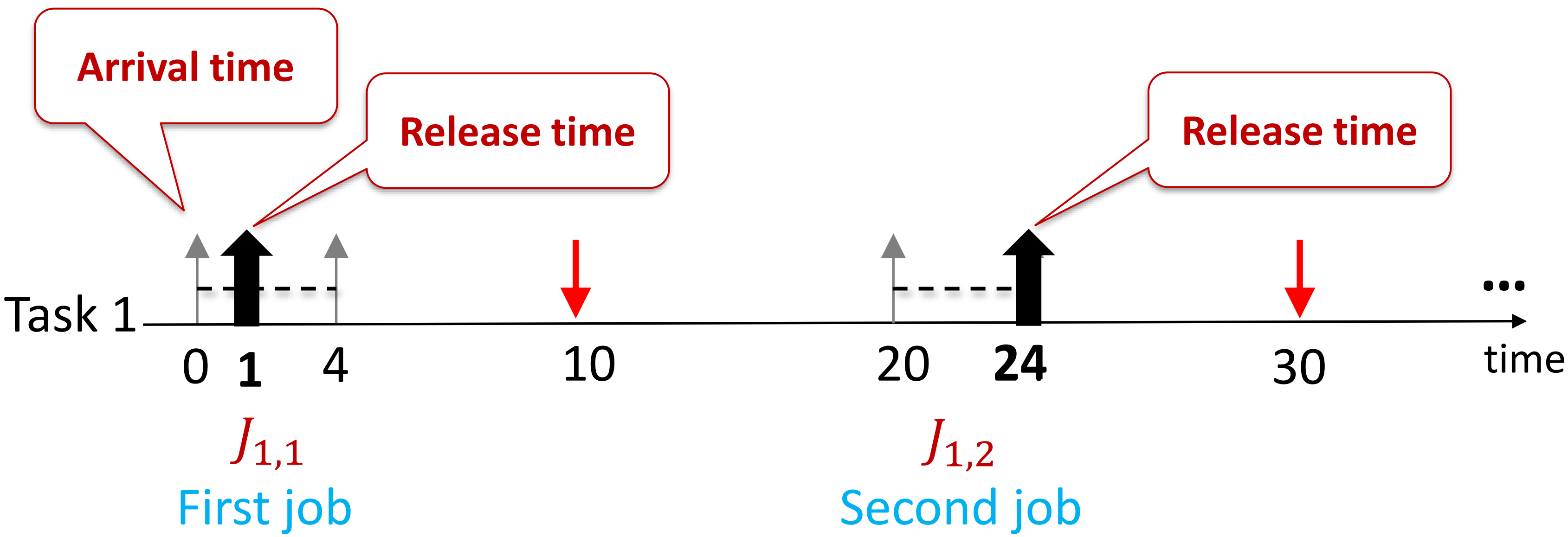
Aperiodic tasks

- Aperiodic tasks are released by **events**
- There is **no lower or upper bound** on the inter-arrive time of their jobs
- Their release time is hence dynamic and may not be predicted



Release jitter

- Periodic tasks can have **release jitter**



- Task 1 has **4 time units of release jitter**.
- Release jitter happens due to interrupt handler overheads, input data availability, functional dependencies (e.g., pre-computation), …
- The actual release time of the first job of Task 1 happens within interval [0, 4] non-deterministically.

Note

- Arrival time refers to the time at which the triggering event happens.
- Release time refers to the actual time a job of the task becomes ready to execute.

- Whenever we refer to tasks with **zero release jitter**, **the arrival time is equal to the release time for every job**. In that case, we use these two terms interchangeably.

- If it is not stated explicitly, **we consider periodic tasks with no release jitter.**

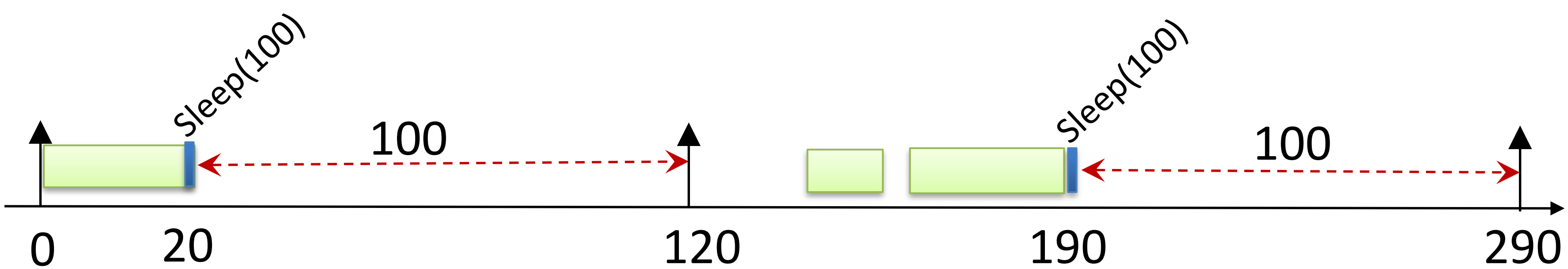
Example

```
While(true)
{
   int temp = readTemperature();
   if (temp > 42)
        send(-1);
   else
   {
        int * array = read10Data();
        send(array);
   }
   sleep (100, ms);
}
```

Question: What type of task is this? Why?

A: Periodic
B: Sporadic
C: Aperiodic

Question: What is the minimum inter-arrival time?

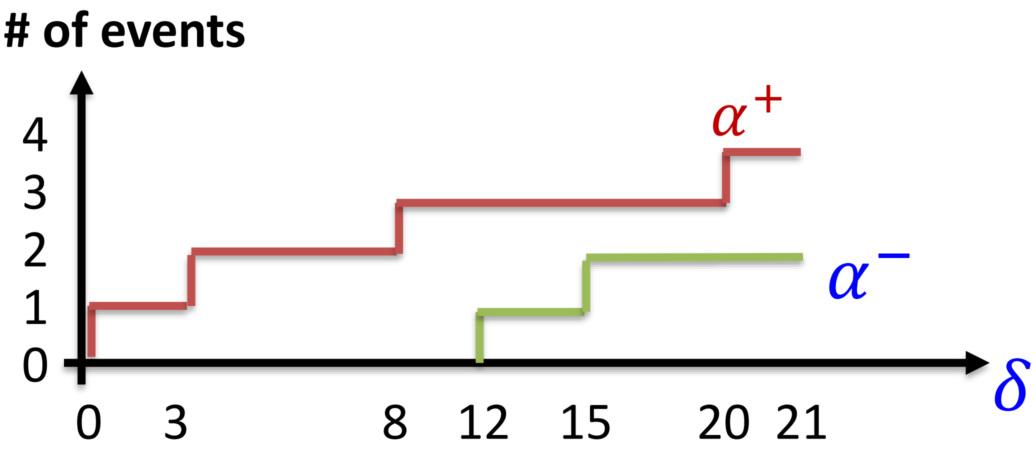


Modelling complex arrival patterns with arrival curves

- An arrival curve represents the **lower bound** and **upper bound** on the **number of events in any time interval**.

α^+ = maximum number of events in any interval of duration δ

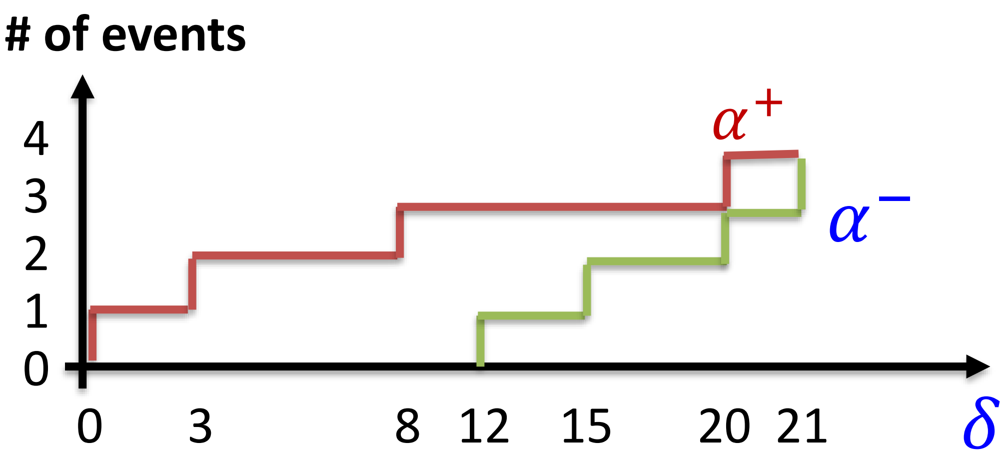
α^- = minimum number of events in any interval of duration δ

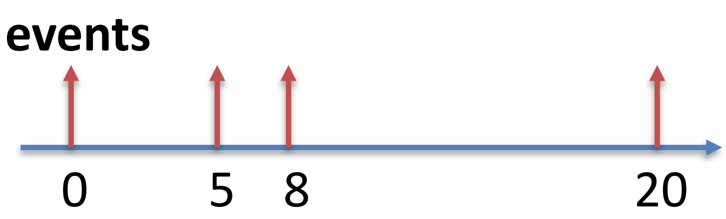


Arrival curves

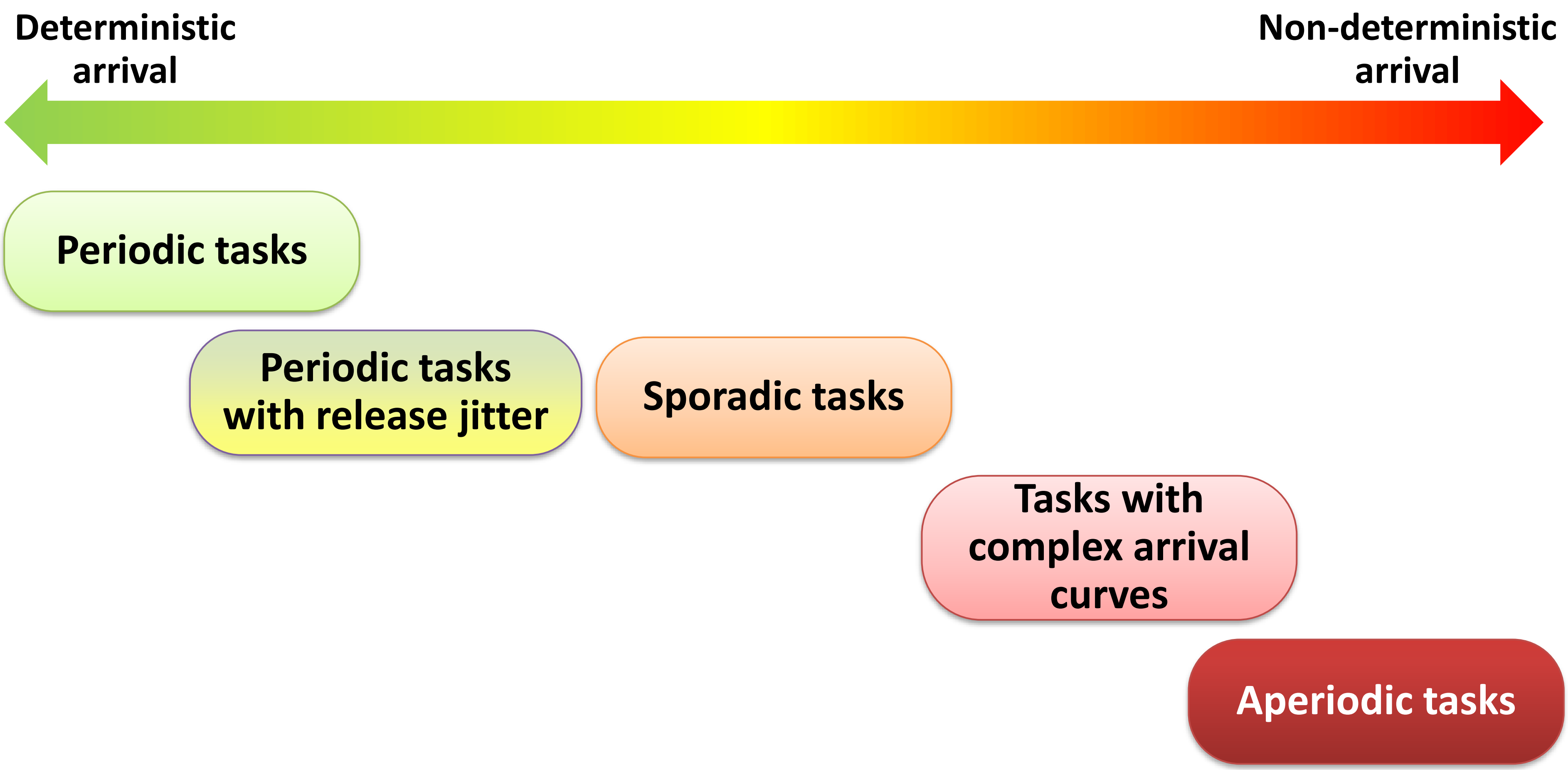
α^+ = maximum number of events in any interval of duration δ

α^- = minimum number of events in any interval of duration δ





From predictability's perspective



Summary

- We discussed
  - the title of the course
  - examples of different **classes of real-time systems**
  - **motivation to analyzing and designing** correct real-time systems
  - How to **model** the timing behavior of a task

Next lecture

- Modeling real-time tasks (Part 2)
- Scheduling 101

Disclaimer

- In some lectures, I have used slides that were obtained from friendly colleagues.

- Thanks to
  - Mitra Nasri
  - Giorgio Buttazzo
  - Reinder Bril
  - Koen Langendoen
  - Akos Ledeczi
  - Gerd Gross
  - Zonghua Gu
  - and many others