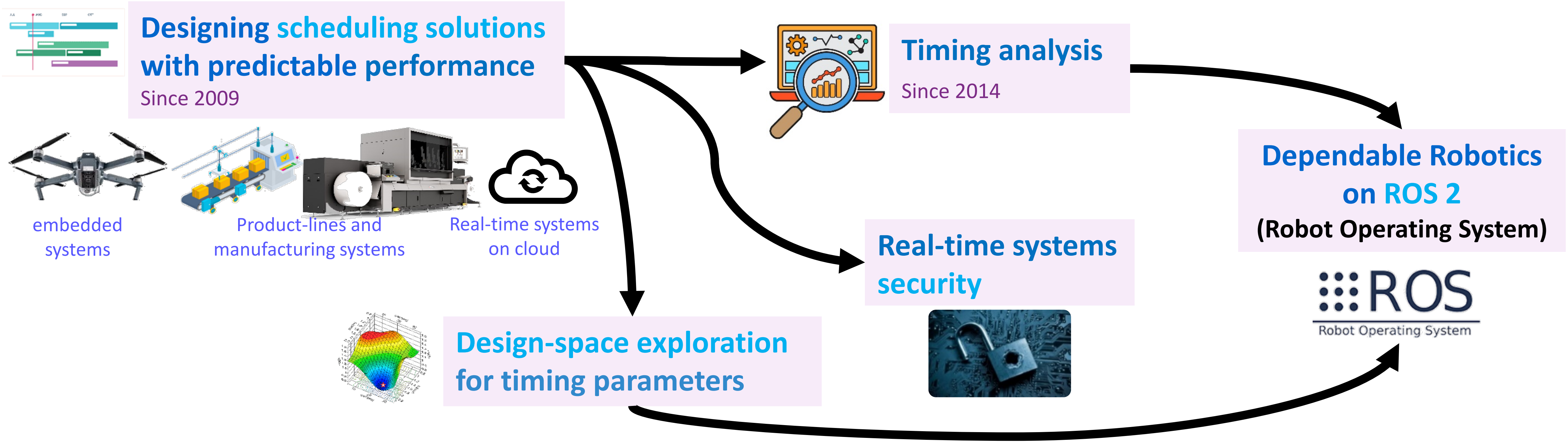
The Right Action at the Right Time:

Past, Present, and Future Trends in Real-Time Systems

Mitra Nasri
m.nasri@tue.nl
Assistant professor
Eindhoven University of Technology (TU/e)

CompSys 2023

My background: designing real-time systems and verifying their correctness



Designing scheduling solutions with predictable performance
Since 2009

embedded systems

Product-lines and manufacturing systems

Real-time systems on cloud

Timing analysis
Since 2014

Real-time systems security

Design-space exploration for timing parameters

Dependable Robotics on ROS 2
(Robot Operating System)

Won a DAAD scholarship in 2013

Won an Alexander von Humboldt Fellowship in 2016

Won Delft Technology Fellowship Award in 2018

Co-PI of an NWO project on scheduling in flexible manufacturing

Co-PI of an EU-project on real-time applications on cloud

University of Tehran
PhD in 2015

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
Postdoc: 2015-2016

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS
Postdoc: 2016-2018

TU Delft
Assistant professor
2018-2020

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY
Assistant professor (tenured)
2020-now

Real-time systems

Real-time systems

Safety

- Human life
- Environment

Time-predictability

- A late (or missed) actuation may cause safety violation
- Example: breaking, air-bag inflation, etc.

Correct response

Timely response

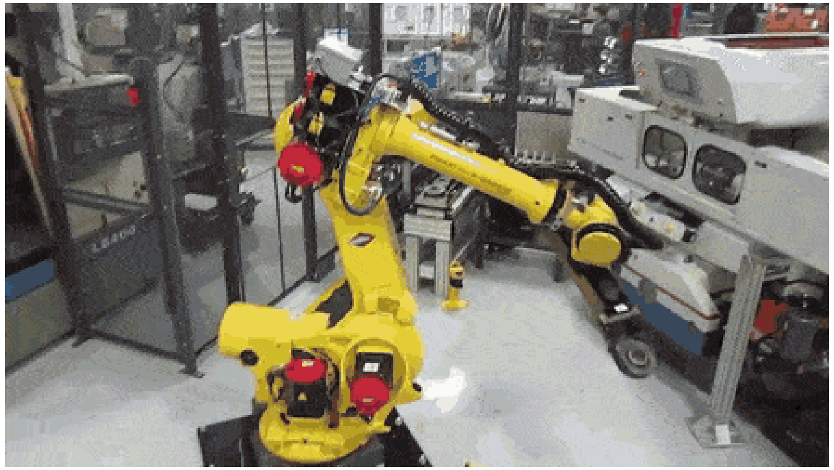
Functional correctness + **Temporal** correctness

Fast ≠ predictable

Real-time systems

Which one(s) is a real-time system?

(A) Deadlines in the order of 10 ms

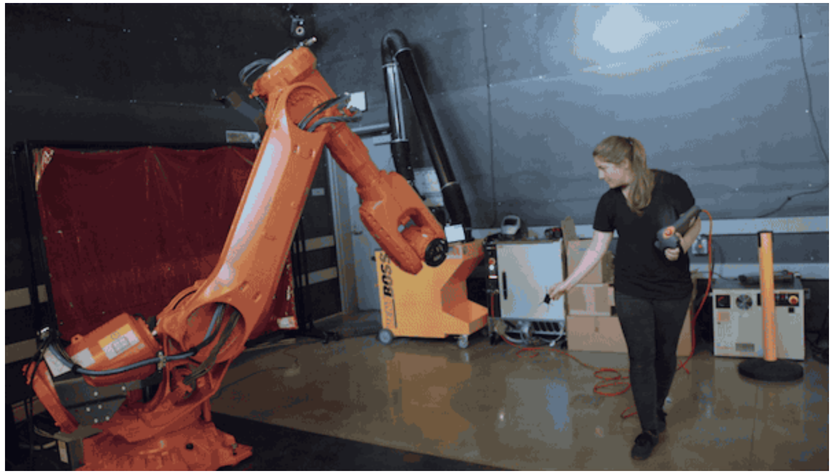


OKUMA (Load and Go Robot)

- Pick and placement
- Path tracking and obstacle avoidance

https://www.digitaljournal.com/pr/industrial-robotics-market-market-size-share-trend-covid-19-impact-and-growth-analysis-report-segmented-by-product-end-user-and-region-analysis-industry-forecast-2022-2027

(B) Deadlines in the order of 100 ms

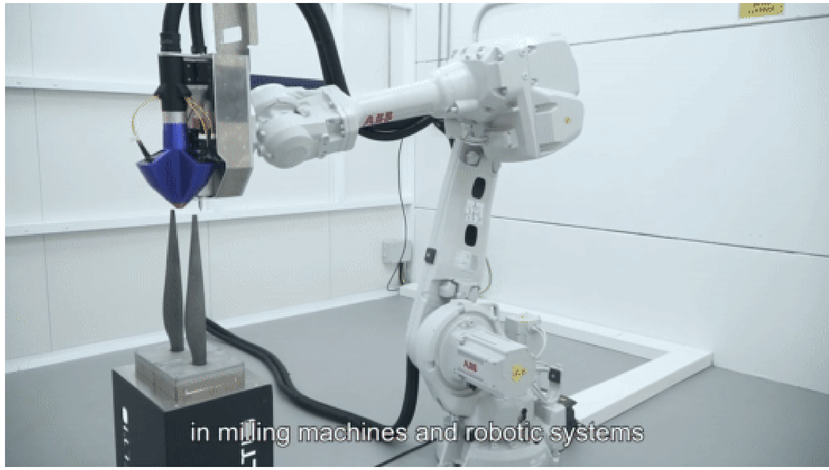


Madlab

- Image processing and object tracking
- Obstacle avoidance

https://www.discovermagazine.com/technology/teaching-robots-to-be-more-than-simple-servants

(C) Deadlines in the order of 300 ms



Aniwaa (Meltio Engine)

- Path planning and path tracking
- Material manipulation/heating

https://www.aniwaa.com/guide/3d-printers/robotic-arm-3d-printing-guide/

Real-time systems

Which one(s) is a real-time system?

All of them!

(A) Deadlines in the order of 10 ms

(B) Deadlines in the order of 100 ms

(C) Deadlines in the order of 300 ms

Real-time systems aren't necessarily "fast"
or have deadlines within few milliseconds!

They are systems that require "**predictable timing behavior**" or shall **satisfy timing constraints**

That are not easy to satisfy

Where do the timing constraints come from?

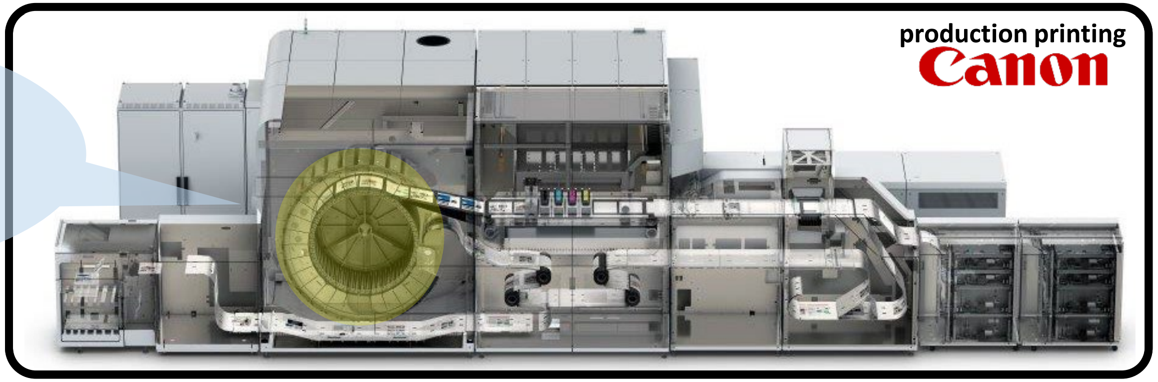
Nature or physics law

Safety requirements

Performance requirements

Quality of service requirements

It takes 45ms for a freshly printed paper to dry enough to be stacked or flipped

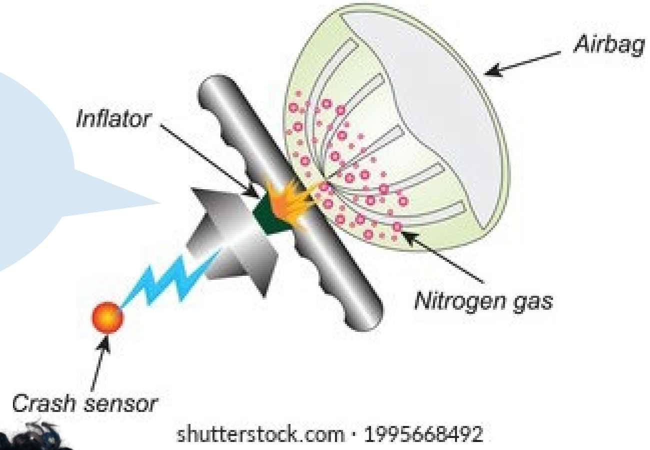


Print 300 pages per minute

Translates to timing constraints of the submodules

Convoy belt controller must execute every 30ms

The chemical reaction that inflates the air bag takes 40ms



Airbag should open from 60 to 100ms after a collision

Refresh rate: 30 frames per second (period = 33ms)

Where do the timing constraints come from?

Nature or physics law

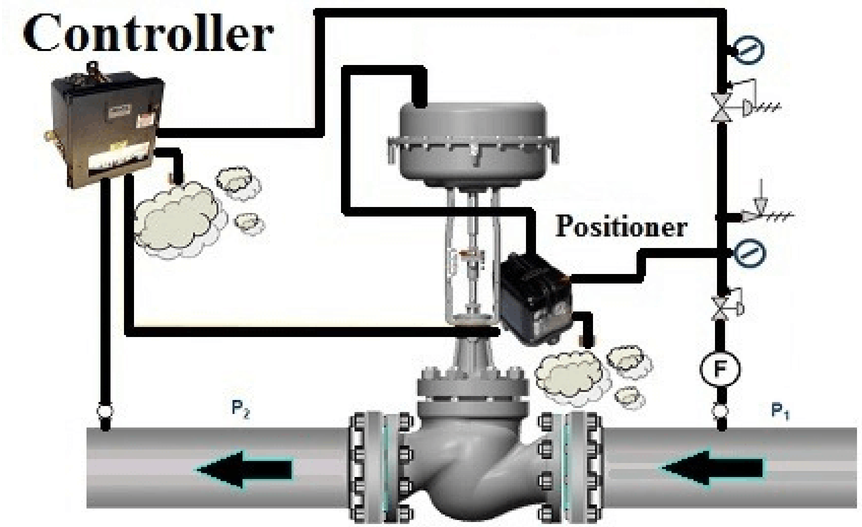
Safety requirements

Performance requirements

Quality of service requirements

Quality of control requirements

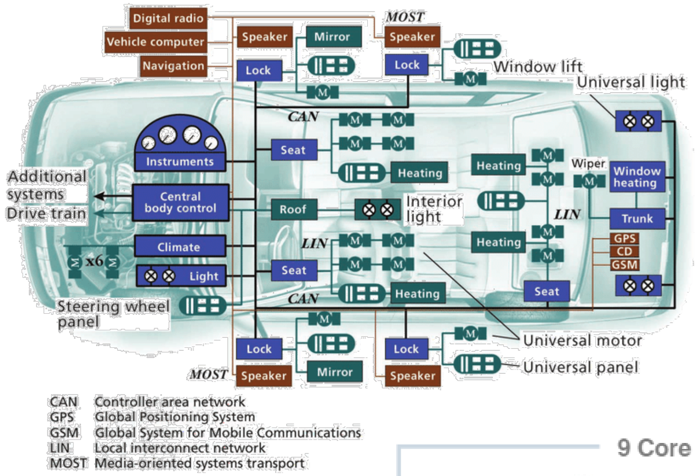
Sampling rate: 2 minutes



Sampling rate: 20ms

Where do the timing constraints come from?

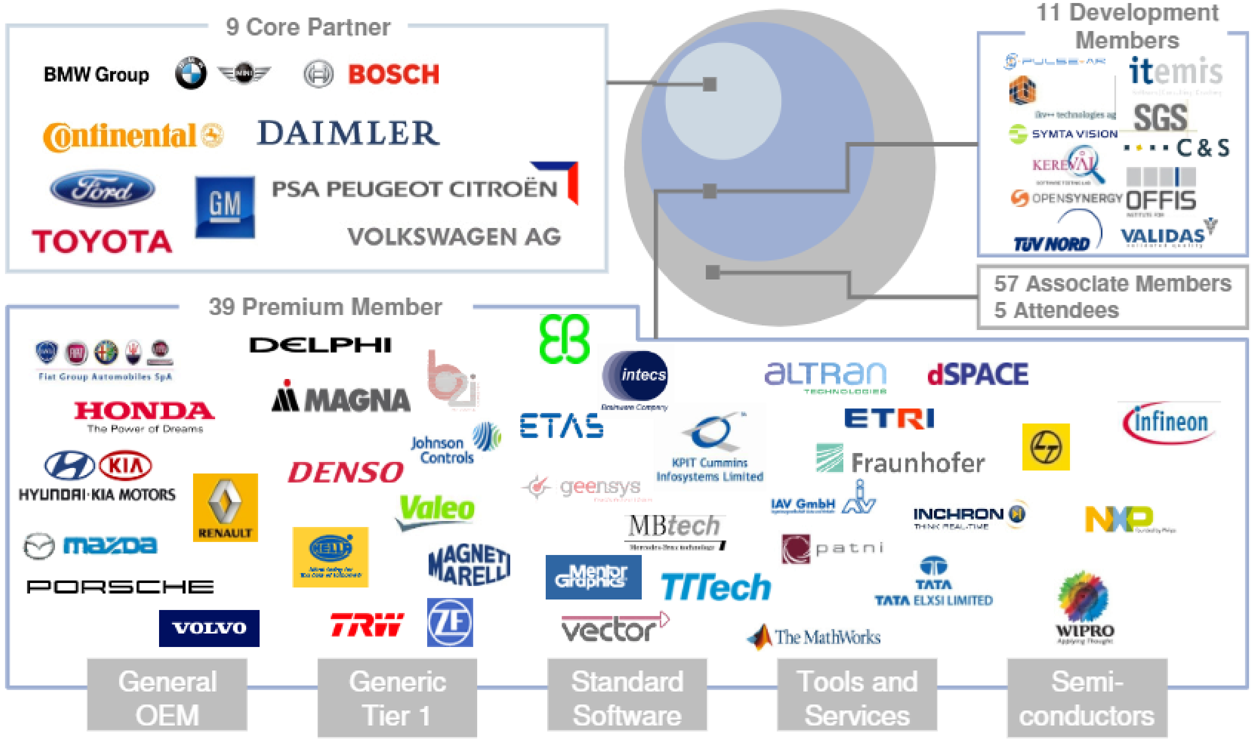
- Nature or physics law
- Safety requirements
- Performance requirements
- Quality of service requirements
- Quality of control requirements
- **Industry standards**



AUTOSAR

Given specifications for timing of runnables and their communication:
{1, 2, 5, 10, 20, 50, 100, 200, 1000}ms

[1] Nicolas Navet, "Automotive Embedded Systems Handbook."
[2] Krammer et al. ,"Automotive benchmark applications for free".



https://www.panonit.com/blog/autosar-%E2%80%93-leading-standard-automotive-industry

Agenda

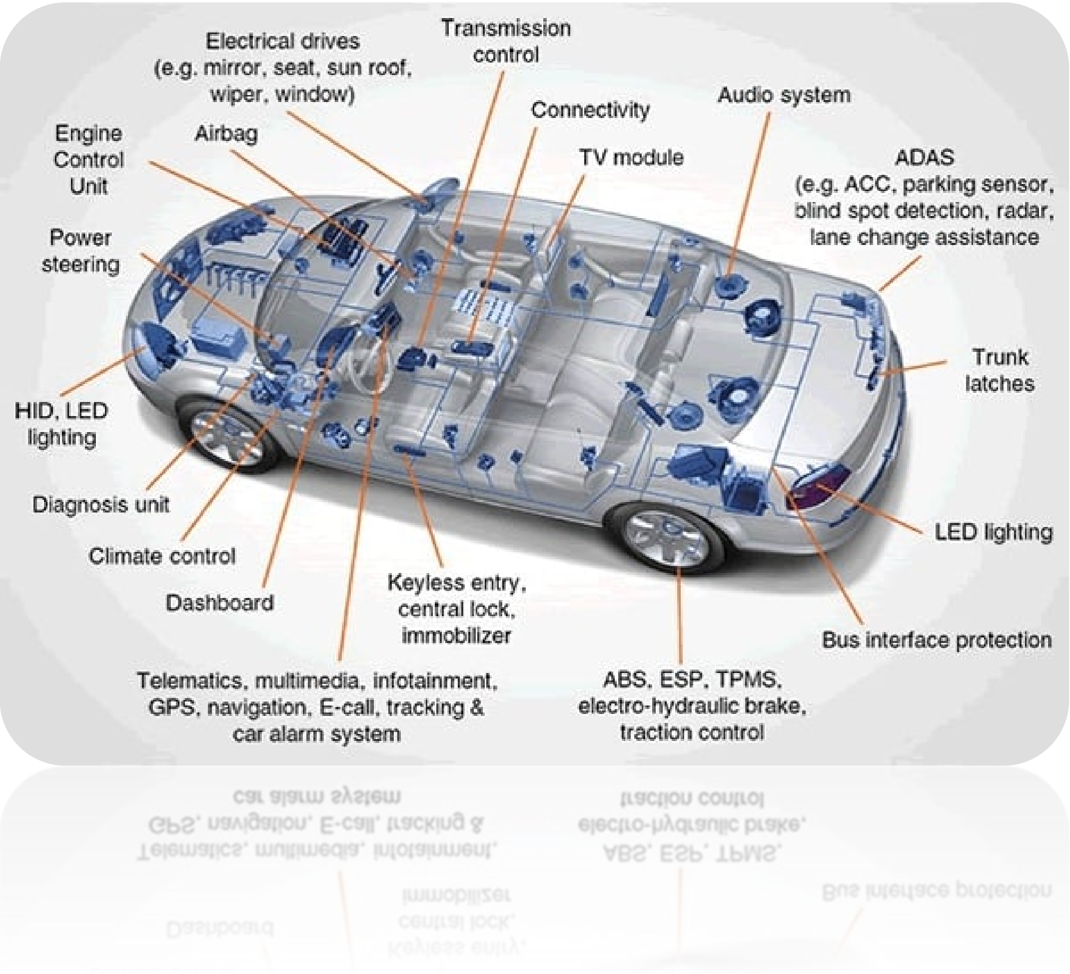
- Where do timing constraints come from?

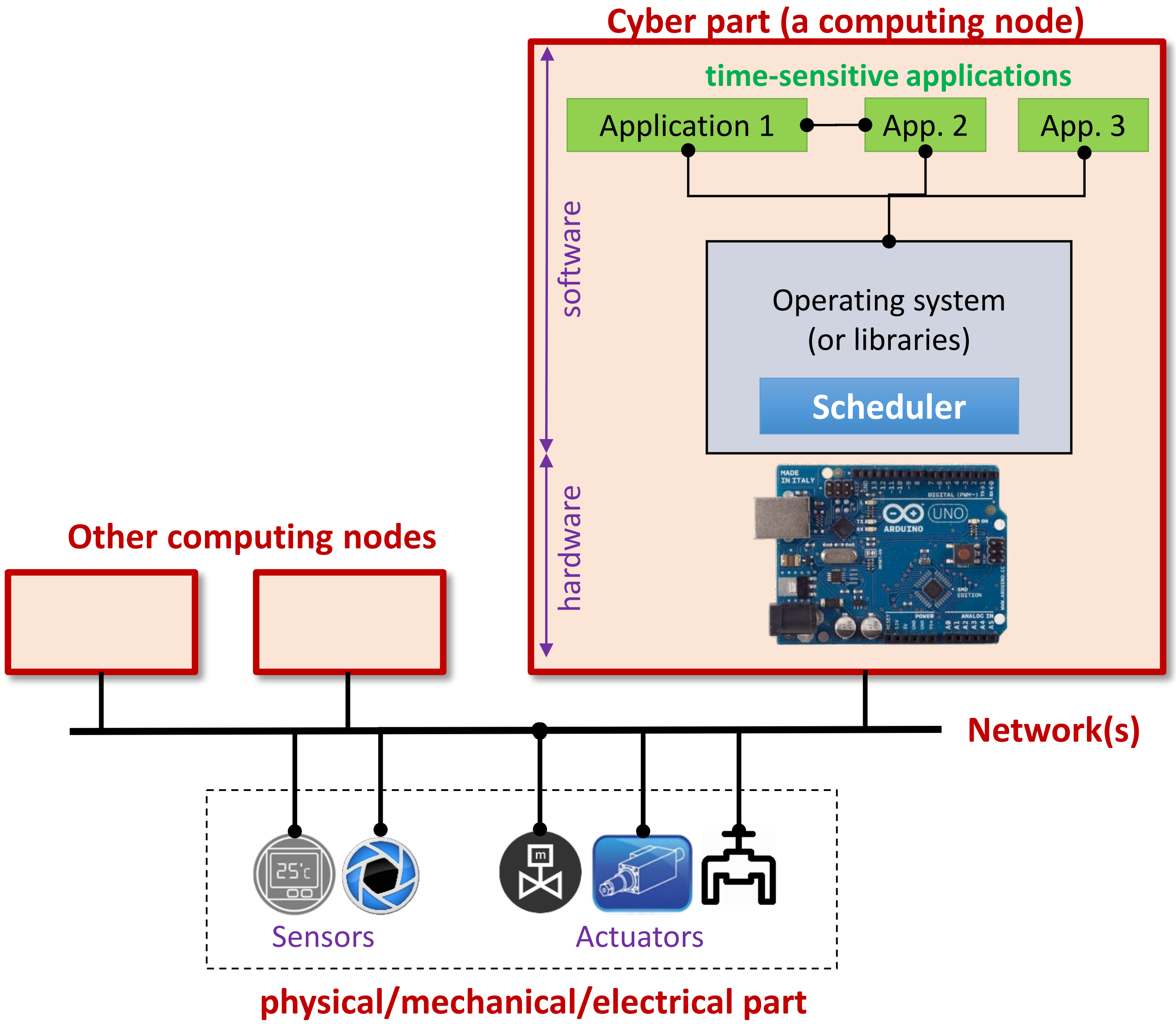
- **What influences the timing behavior of a system?**
  - Why should we care about it?

- **Why the response-time analysis is hard?**
  - What can we do about it?

- **The past, current, and future trends in real-time systems research**

What influences the timing behavior of a system?





Cyber part (a computing node)

time-sensitive applications

Application 1

App. 2

App. 3

software

hardware

Operating system
(or libraries)

Scheduler

Other computing nodes

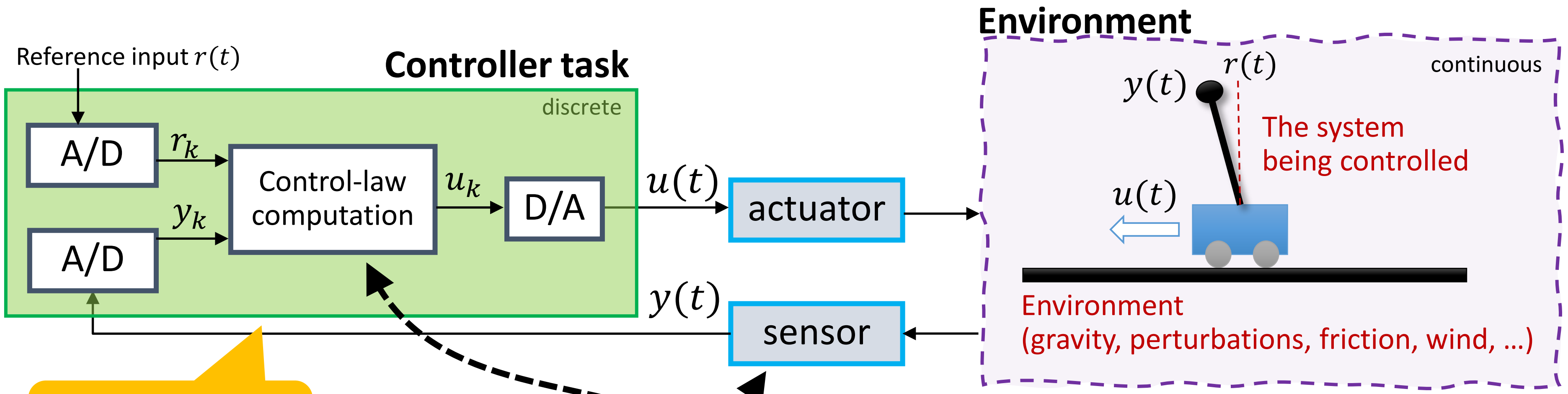
Network(s)

Sensors

Actuators

physical/mechanical/electrical part

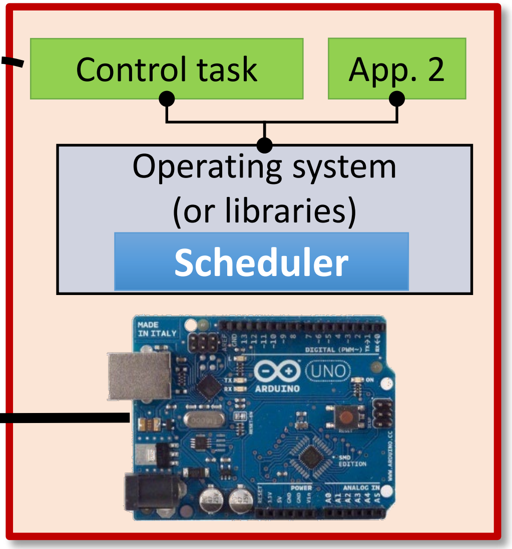
What influences the timing behavior of a system?

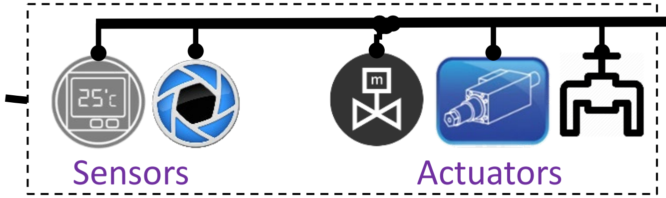


Controller logic
(e.g., PID, MPC, etc.)

Implementation of the control task

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





The control task is released

Control task timeline:

What influences the timing behavior of a system?

Control algorithms often are <u>not prepared to tolerate jitters and delays</u> between the perception and actuation moments

Not bad! Just slightly go right now.

At this moment, the plant is observed

Oh my! There should have been more friction than I expected!

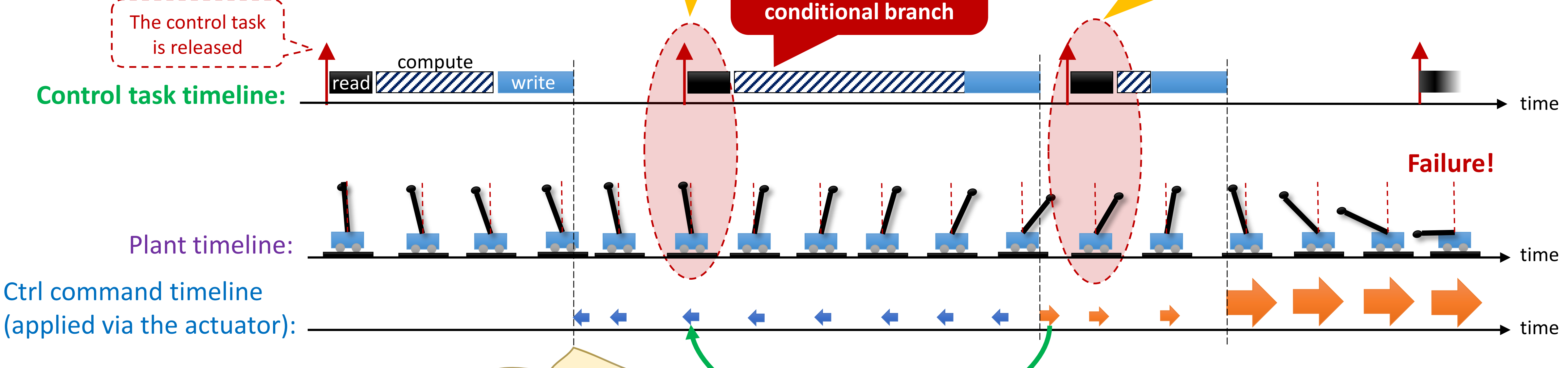
Go RIGHT! **RIGHT**!

The next observation

The controller is unaware of the I/O delay, so it assumes the surface has a lot of friction.

A long delay caused by a cache miss or a conditional branch

The control task is released



Moments at which the control command is updated

Should have been applied long ago

Common timing constraints

Response-time constraints

- Worst-case response time (WCRT) shall be smaller than the deadline

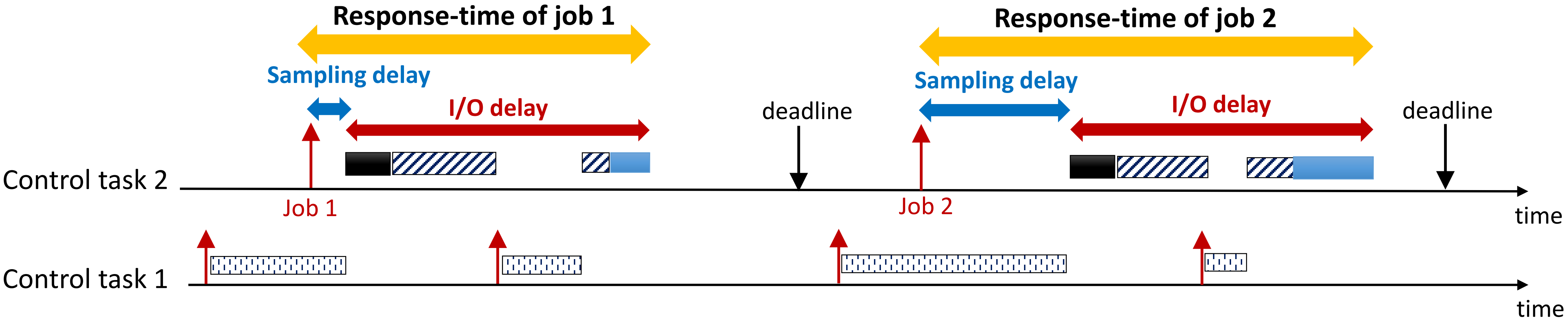
The most common timing constraint

Delay constraints

- Sampling delay
- I/O delay

Jitter constraints

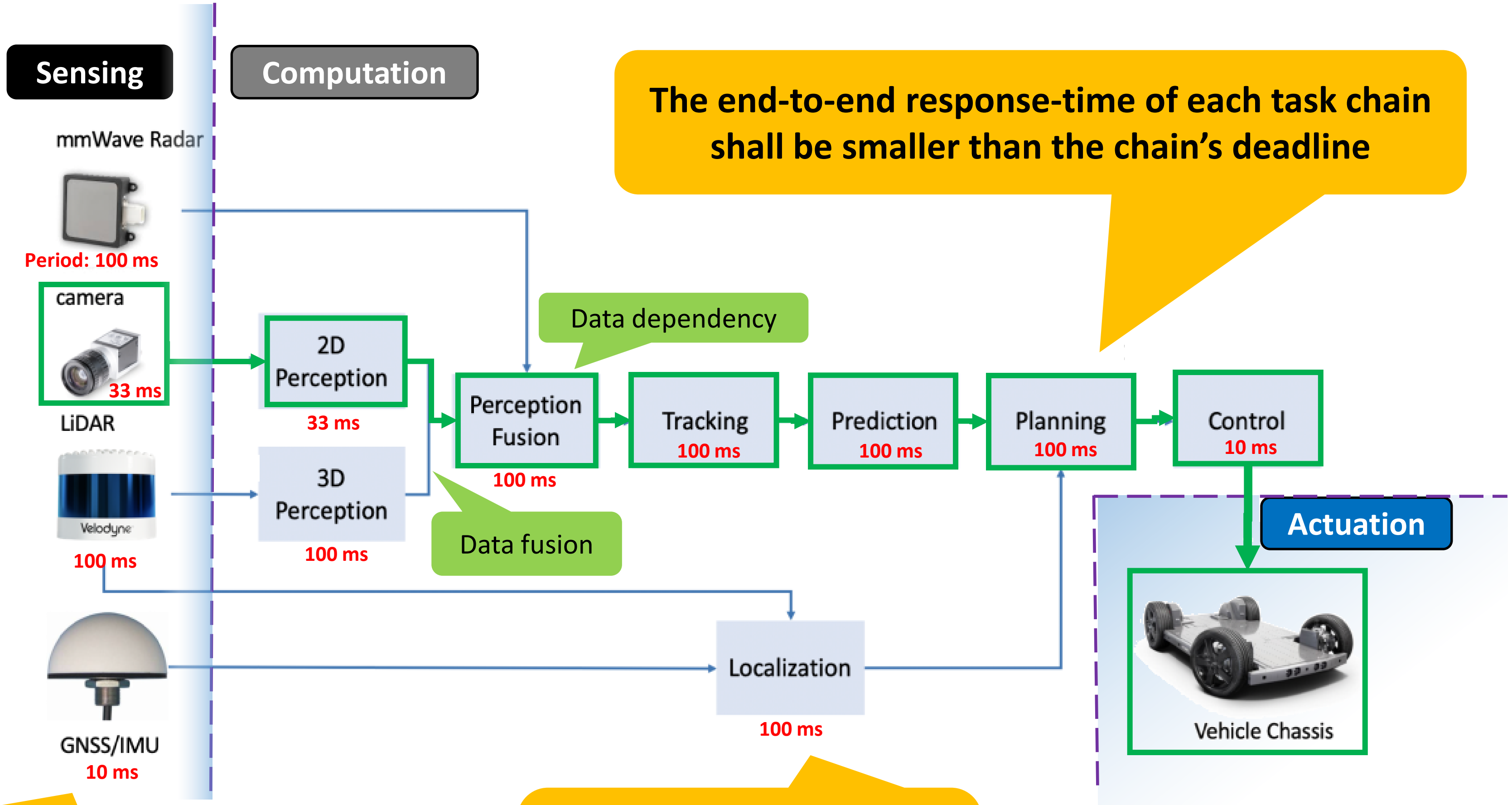
- Response-time jitter
- Sampling jitter
- I/O jitter



- Platform: single-core
- Scheduling policy: fixed-priority policy (task 1 has a higher-priority than task 2)

- **Response time** = completion time - release time
- **Worst-case response-time (WCRT)** = largest response time in the lifetime of a task
- **Sampling delay** = start time – release time
- **I/O delay** = completion time – start time
- **Jitter of X** is the difference between the best and worst values of X.

Today's systems have more complex timing constraints



Sensing

Computation

mmWave Radar
Period: 100 ms

camera
33 ms

LiDAR
100 ms

GNSS/IMU
10 ms

2D Perception
33 ms

3D Perception
100 ms

Perception Fusion
100 ms

Tracking
100 ms

Prediction
100 ms

Planning
100 ms

Control
10 ms

Localization
100 ms

Actuation

Vehicle Chassis

Data dependency

Data fusion

The end-to-end response-time of each task chain shall be smaller than the chain's deadline

Observations should be from 'the same time', otherwise they might be irrelevant/inconsistent.

Each task shall finish before its next activation (deadline $\leq$ period)

S. Liu, B. Yu, N. Guan, Z. Dong, and B. Akesson. 2021. RTSS 2021 Industry Session. http://2021.rtss.org/industry-session/

Importance and prevalence of timing constraints in industry

Benny Akesson

ESI

Mitra Nasri

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Geoffrey Nelissen

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

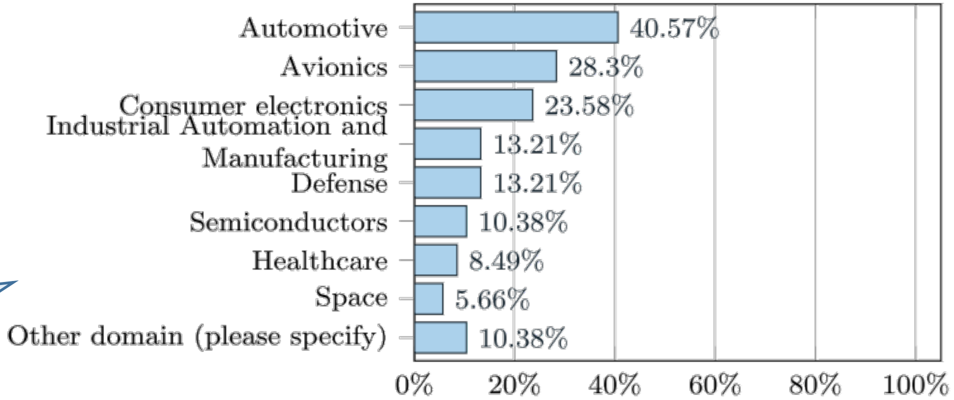
Sebastian Altmeyer

UNI

Rob Davis

UNIVERSITY of York

A Comprehensive Survey of Industry Practice in Real-Time Systems



More than 100 real-time systems practitioners

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

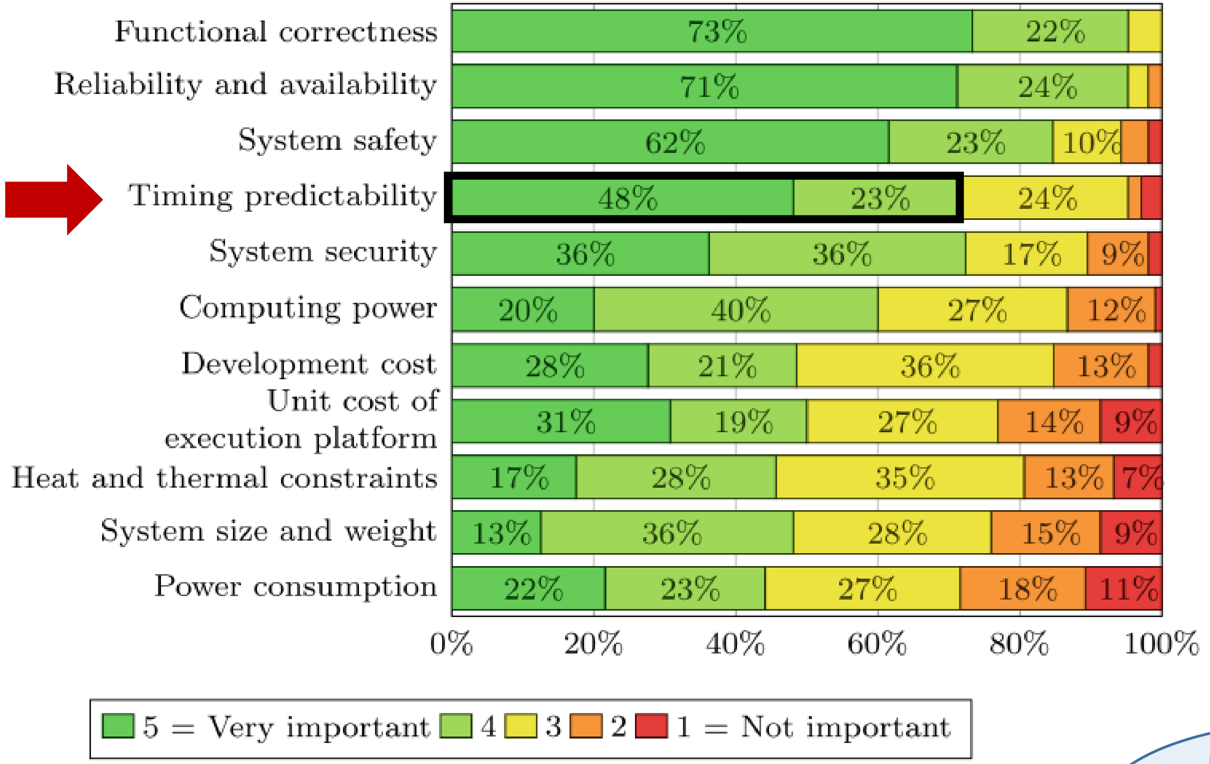
Importance and prevalence of timing constraints in industry

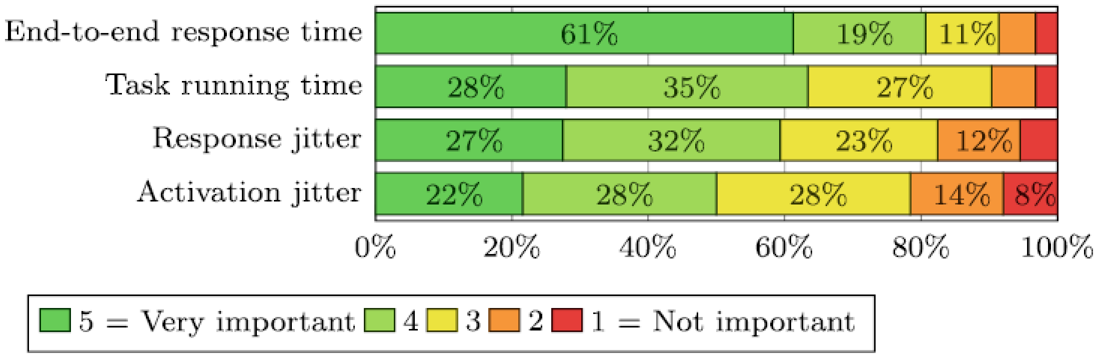
Industry

Timing predictability comes right after system's safety!

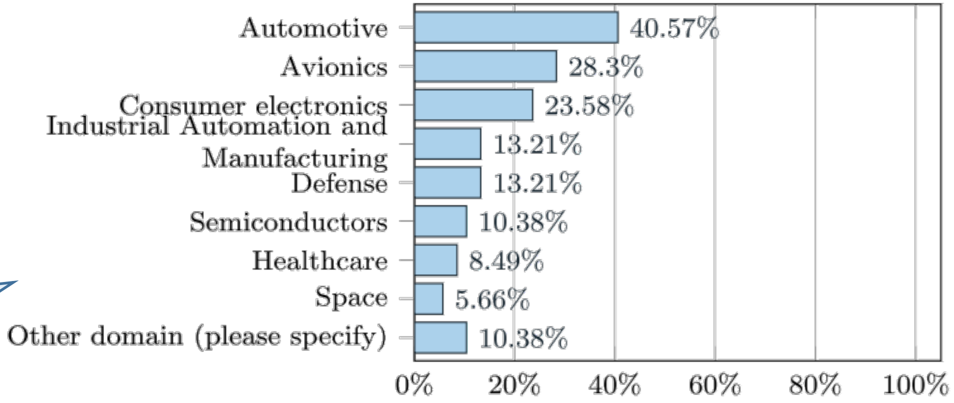
In more than 70% of real-time systems, timing predictability is very important or important.

In 80% of real-time systems, the end-to-end response time is very important or important





More than 100 real-time systems practitioners



Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

What impacts the response time of a task?

The **execution time** of the task

Concurrent execution of <u>other tasks</u> on the **<u>hardware</u>** platform

Related to the **application** and **hardware platform**

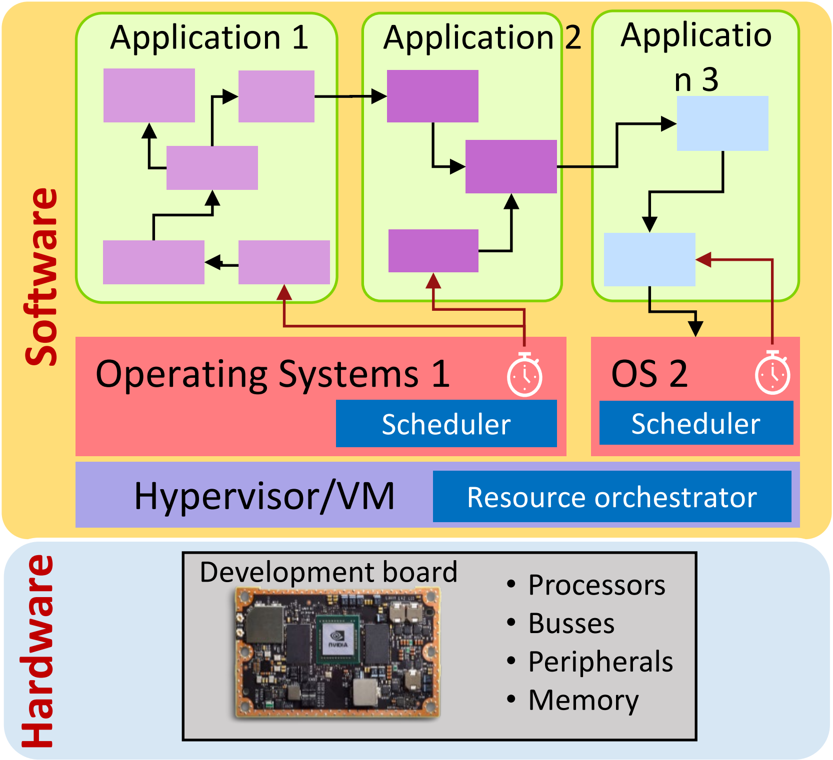
<u>Scheduling</u> policy and **<u>interferences</u>** from **other tasks**

Resource assignment, orchestration, and **management <u>policy</u>**

<u>Data</u> communication (and synchronization) **<u>overheads</u>**

- inside a computing node
- between computing nodes
- over networks

Related to the **operating system, virtualization**, and **communication**



What impacts a task’s execution time?

Task’s code

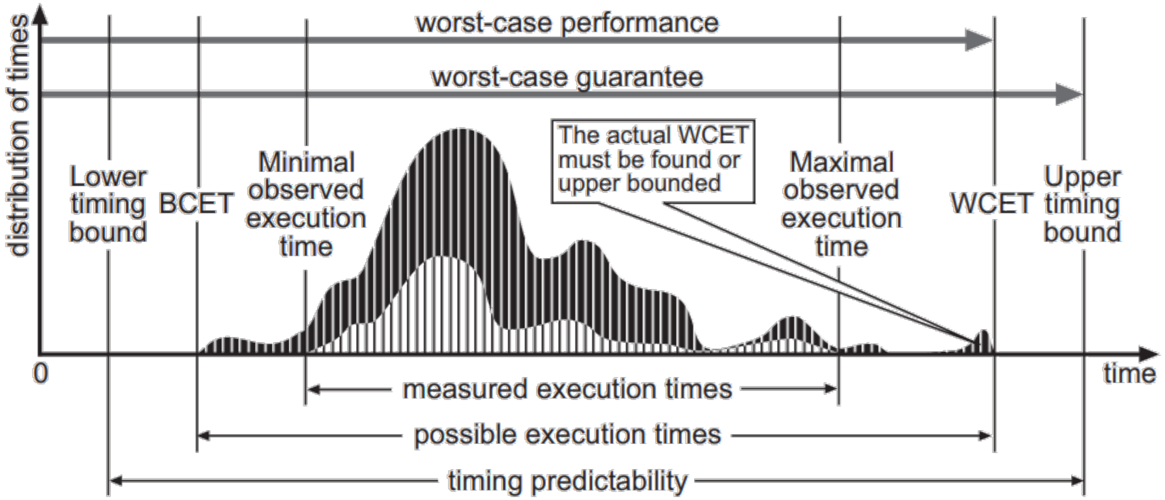
```
While(true)
{
   int temp = readTemperature();
      if (temp > 42)
            send(-1);
      else
      {
            int * array = read10Data();
            int max = -1;
            for (int i=0; i < 10; i++)
                  if (max < 0 || array[i] > max)
                        max = array[i];
            send(max);
      }
      sleep (100, ms);
}
```

Software aspects

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

Hardware aspects

- Cache misses
- Branch predictors
- Out-of-order execution
- Interference on the bus or memory banks
- Resolution of hardware timer
- Context switch overheads



It is somehow addressed for single-core platforms [1]

Finding the **worst-case execution time (WCET)** is a long-lasting **open problem**

There is barely any ‘safe’ solution for multicore platforms

Hardware technologies heavily influence the analysis of WCET ➡ **Difficult to catch up with the advancement of the hardware technology**

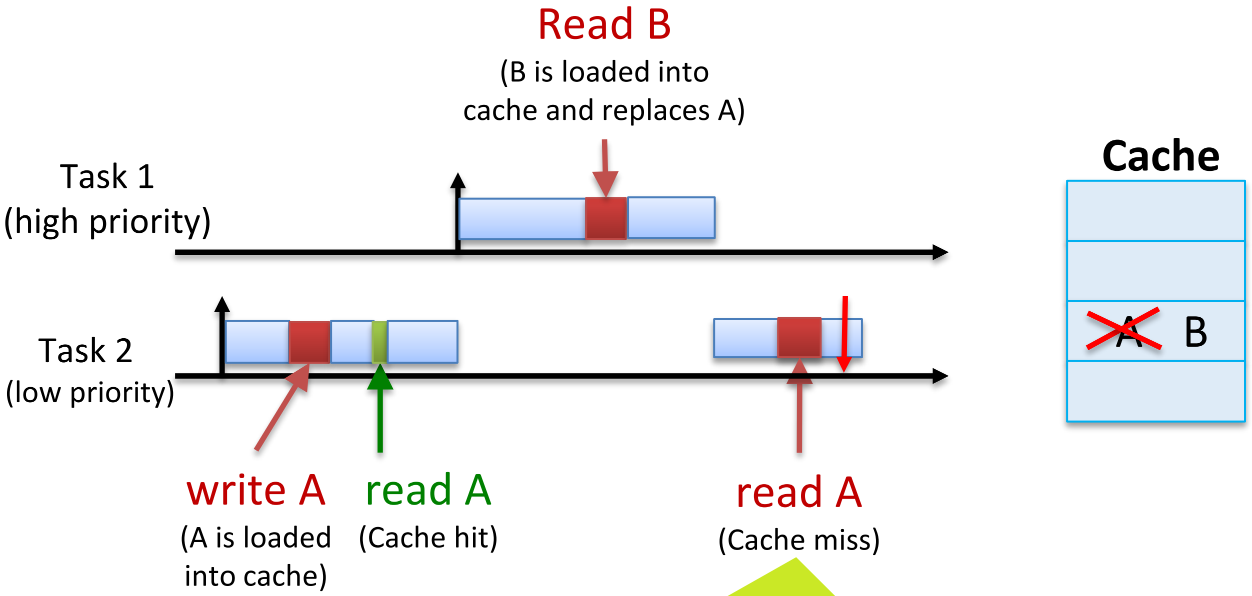
[1] Reinhard Wilhelm, et al., “The worst-case execution-time problem—overview of methods and survey of tools,” ACM Transactions on Embedded Computing Systems 7, 3, Article 36, 2008.

How execution time of one task is affected by co-runners?

**Co-running tasks run concurrently
on a multi-core/multi-processor platform**

Co-runners compete on accessing **shared caches, I/O devices, memory bus, memory banks, and memory controllers**

Cache-related preemption delay (one core):



Read B

(B is loaded into cache and replaces A)

Task 1
(high priority)

Task 2
(low priority)

write A
(A is loaded into cache)

read A
(Cache hit)

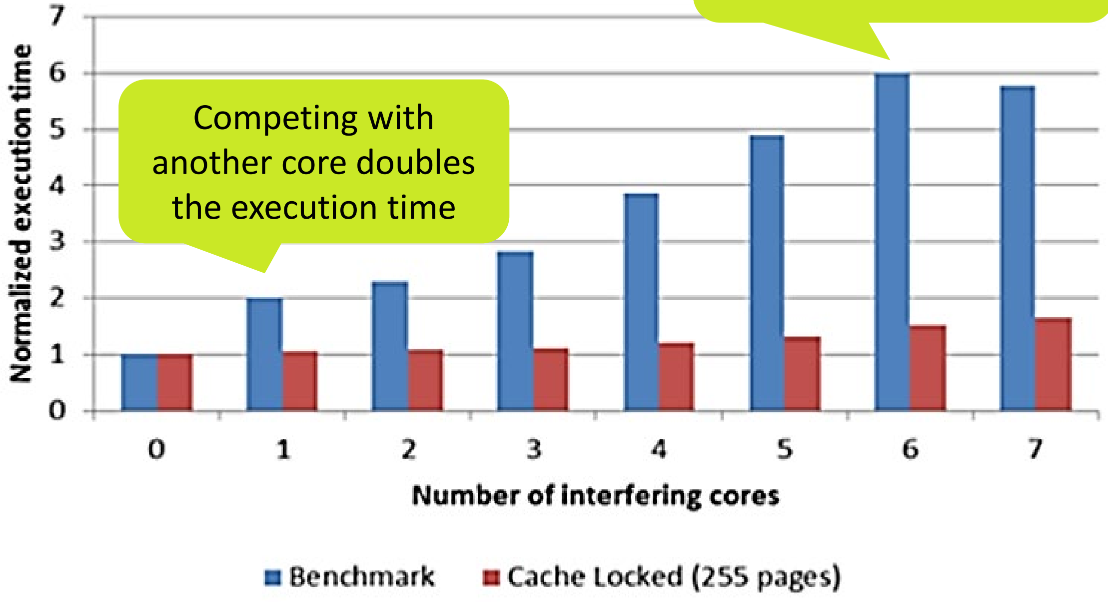
read A
(Cache miss)

Cache

~~A~~ B

If there was no preemption, this was a cache hit

Lockheed Martin Space Systems
Testbed on an 8-core Freescale P4080



Critical software can be slowed by up to 6X if uncontrolled

Competing with another core doubles the execution time

L. Sha, M. Caccamo, R. Mancuso, J. Kim, M. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale, R. Bradford, “Single Core Equivalent Virtual Machines for Hard Real—Time Computing on Multicore Processors,” 2014.

How execution time of one task is affected by co-runners?

Co-running tasks run concurrently on a multi-core/multi-processor platform

Co-runners compete on accessing **shared caches, I/O devices, memory bus, memory banks, and memory controllers**

RTAS'19 best-paper award shacked the state of the art

Co-running tasks can easily slowdown another task by a factor of 300 (on a 4-core platform [Raspberry PI]) just by stressing the memory controller!

Parallelizing applications on multicores may result in slowing the system down
(regardless of the granularity of parallelization)

Michael Garrett Bechtel and Heechul Yun. **Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention**. *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

Finding the worst-case execution time

Static timing analysis (STA)

(to derive safe upper bounds on WCET)

- Obtain the control-flow diagram
- Add the worst-case latency of each instruction

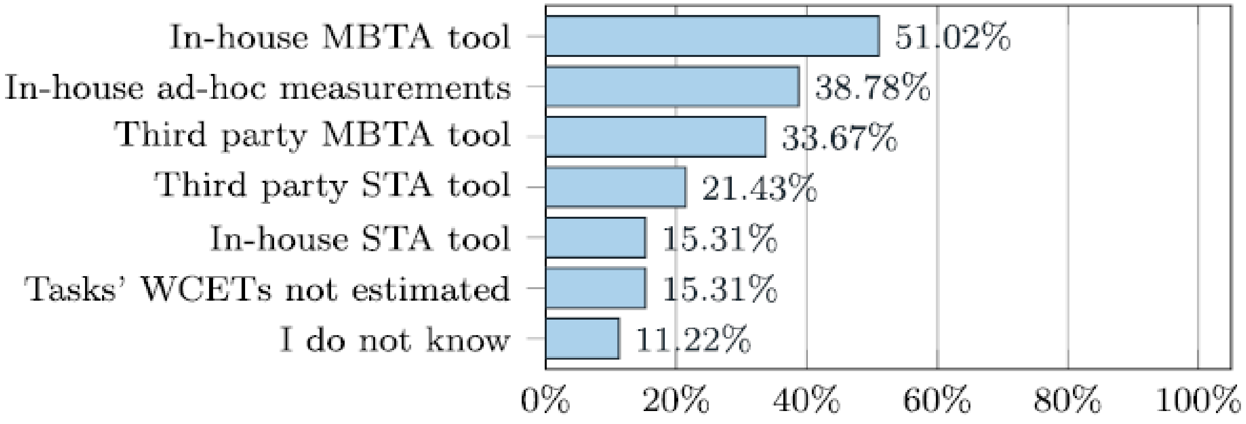
e.g., worst-case cache and memory access latencies, worst-case number of iterations of a loop, …

Measurement-based timing analysis (MBTA)

(measure WCET under <u>normal</u> and <u>stressed</u> scenarios)

- It is often fast
- It does not need knowledge of hardware or code
- It is more representative for actual execution times

- It requires the system to be built
- Measurements may not be representative

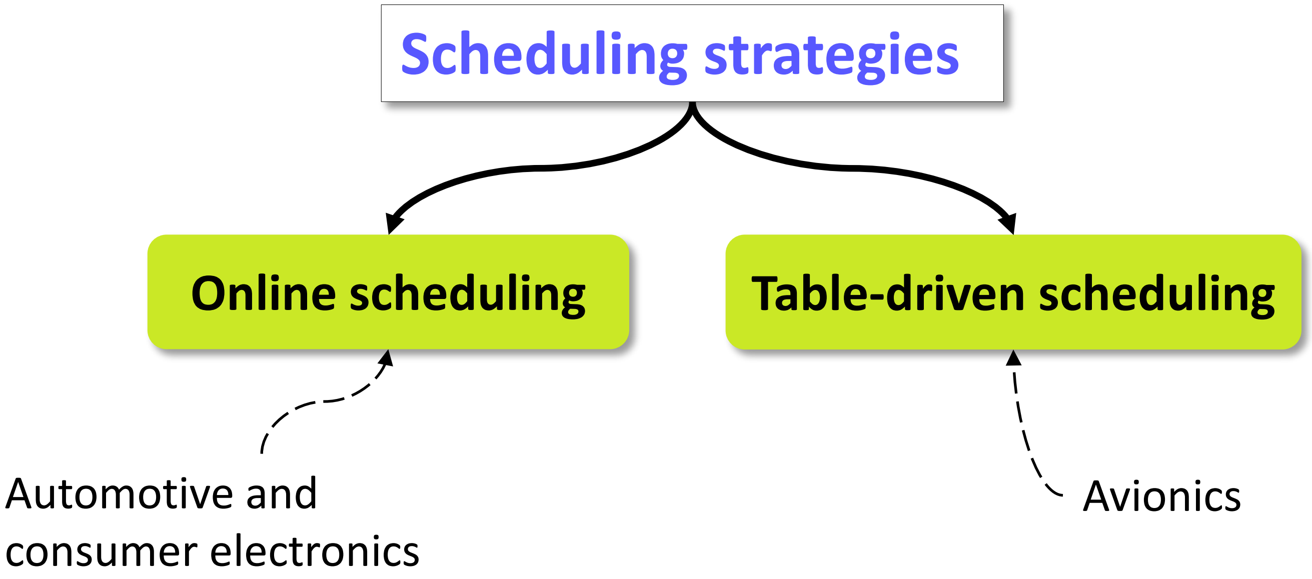


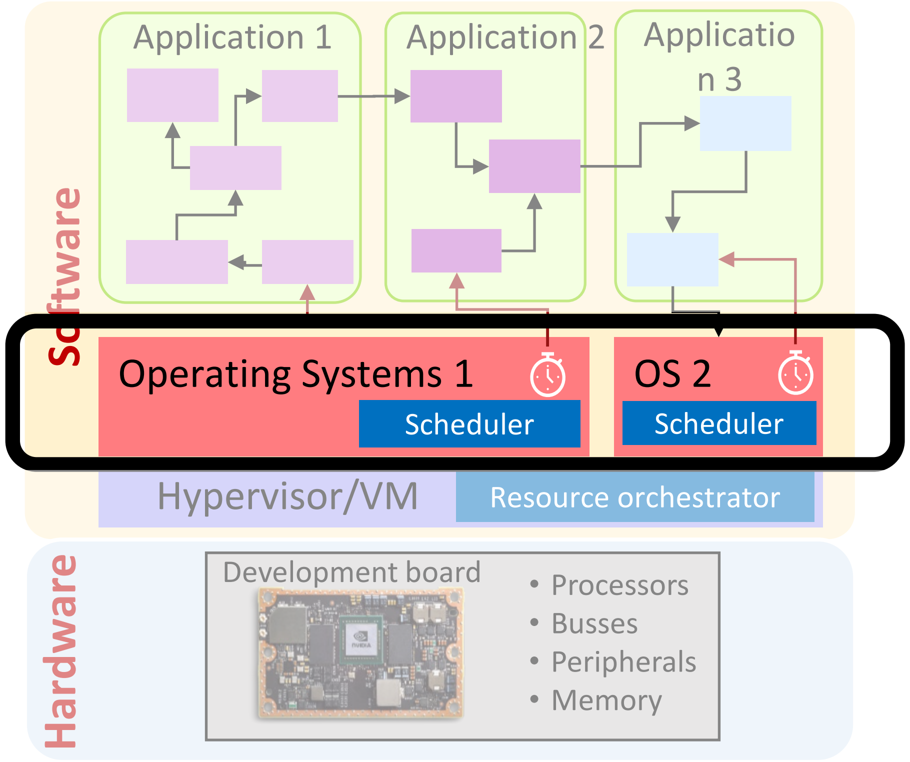
Industry

Measurement-based timing analysis using in-house tools or ad-hoc measurements is the **<u>common way</u>** of obtaining WCET estimates in industry

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

Impact of scheduling policy on a task’s response time





How does scheduling impact a task’s response time?

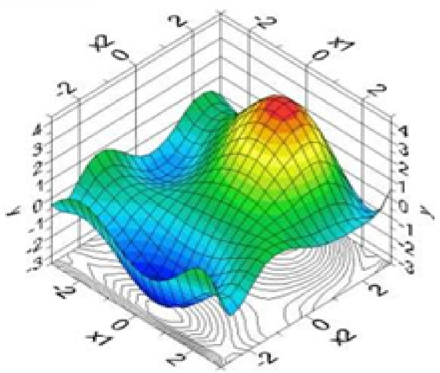
Table-driven scheduling

Stores the entire schedule of the system in a table in memory to be repeatedly followed during the system’s life-time.

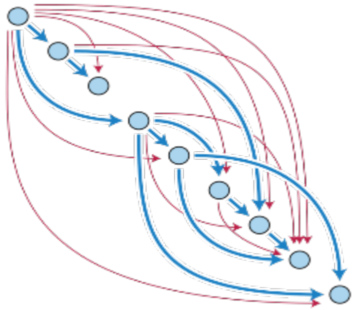
- **Easy to respect the timing constraints** (correct by construction)
- **Allows further optimization** of the schedule
- **Low runtime overhead**

- **Requires a lot of memory**
- **Often not robust** against unexpected deviations
- **Does not use system resources efficiently**

Optimization objectives

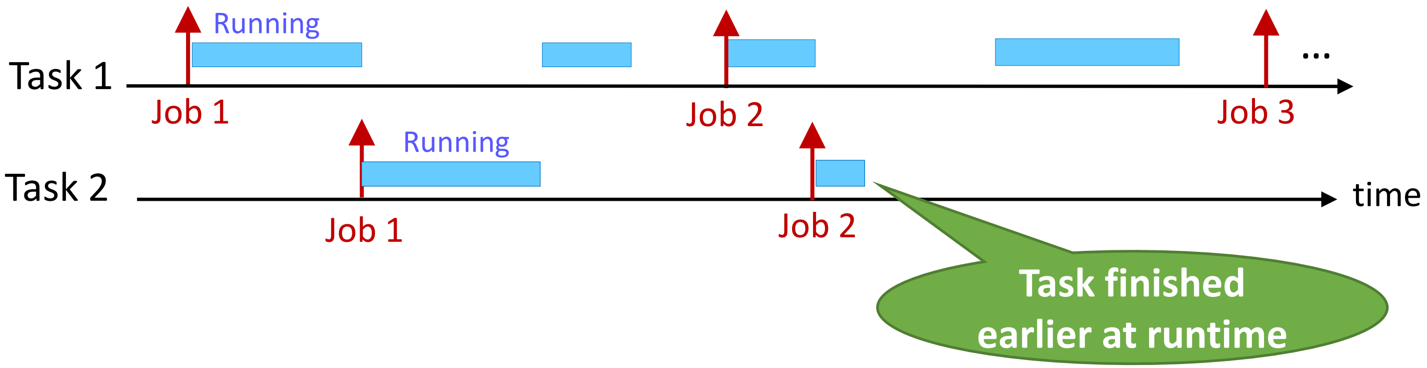


Constraints to respect



Schedule

slot	1	2	3	4	5	6	7	8	9	10	11	12
task	1	1	2	2	1		1	2	2	1	1	

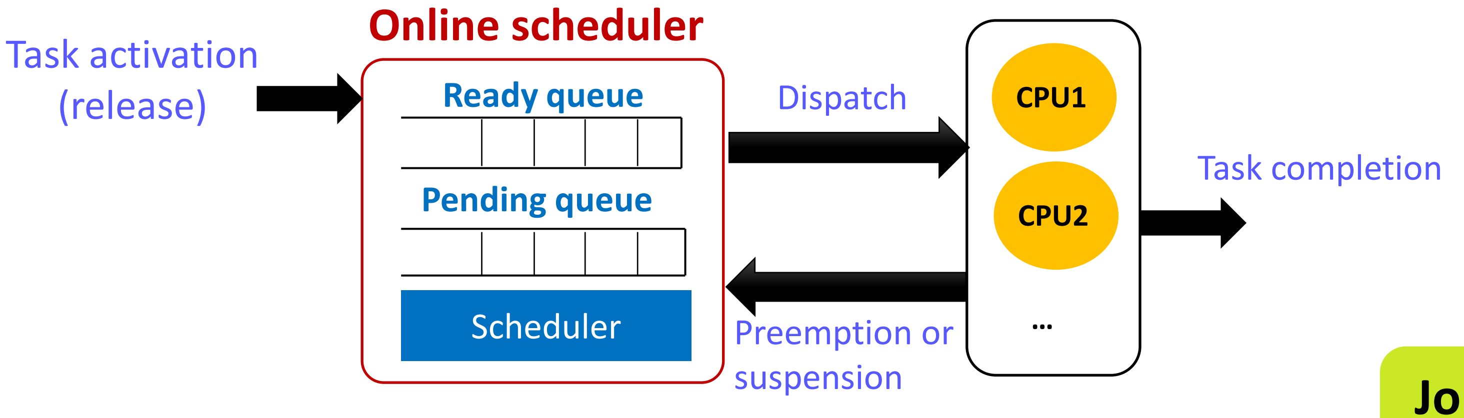


Some solutions to improve memory consumption of table-driven scheduling:

- Mitra Nasri and Björn B. Brandenburg, "Offline Equivalence: A Non-Preemptive Scheduling Technique for Resource-Constrained Embedded Real-Time Systems”, RTAS, 2017, Outstanding Paper Award. [paper | slides | companion page]
- Mitra Nasri, Robert I. Davis, and Björn B. Brandenburg, "FIFO with Offsets: High Schedulability with Low Overheads,” RTAS, 2018.

How does scheduling impact a task’s response time?

Online scheduling



Task activation (release)

Online scheduler

Ready queue

Pending queue

Scheduler

Dispatch

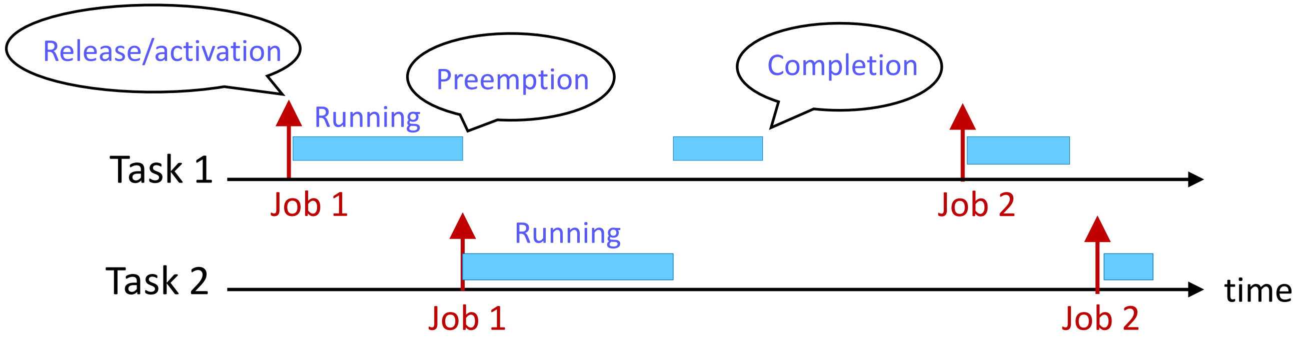
CPU1

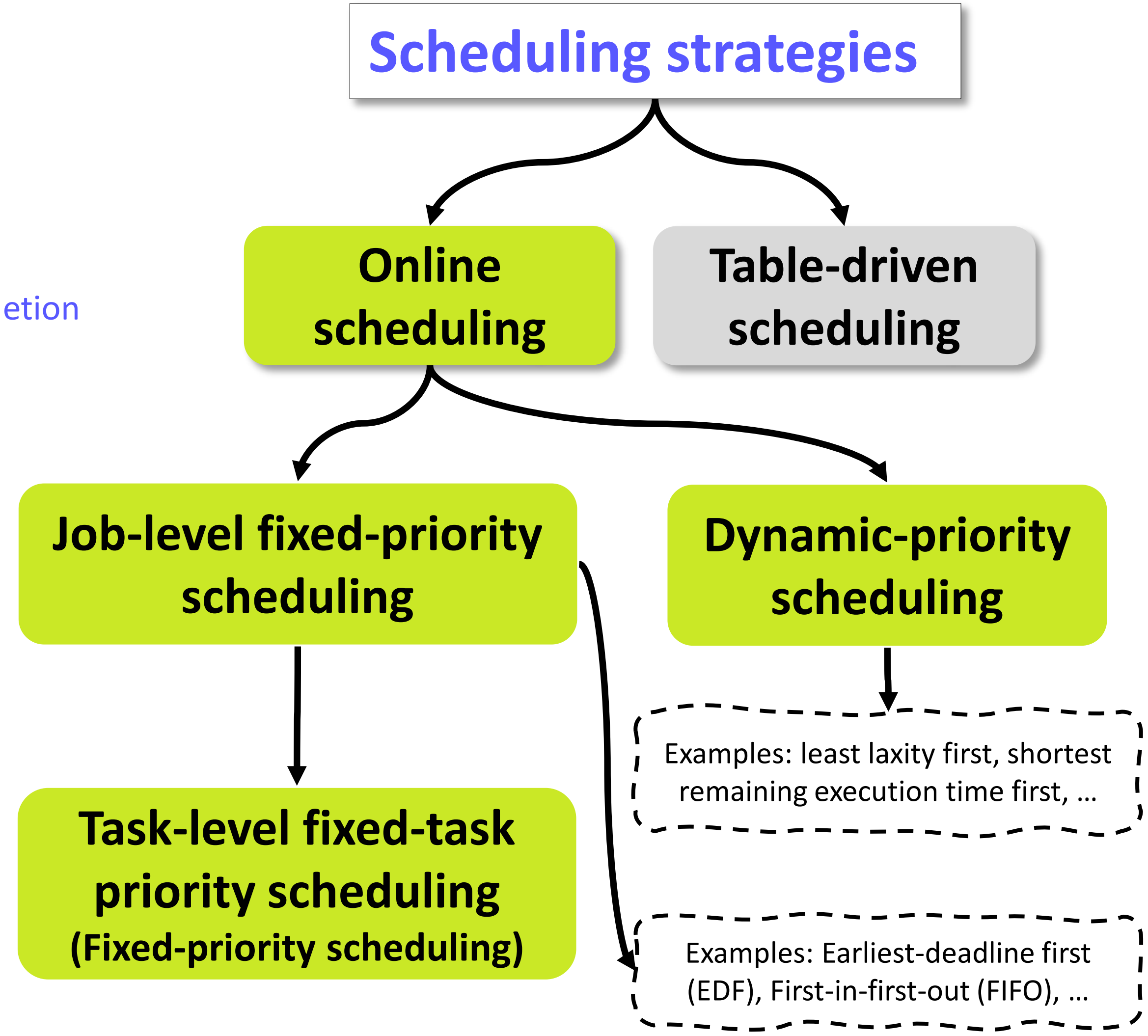
CPU2

…

Task completion

Preemption or suspension





Scheduling strategies

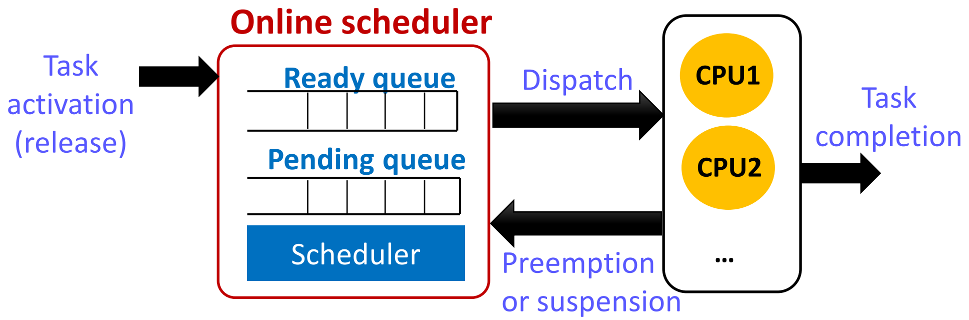
- **Online scheduling**
  - **Job-level fixed-priority scheduling**
    - **Task-level fixed-task priority scheduling (Fixed-priority scheduling)**
    - Examples: Earliest-deadline first (EDF), First-in-first-out (FIFO), …
  - **Dynamic-priority scheduling**
    - Examples: least laxity first, shortest remaining execution time first, …
- **Table-driven scheduling**

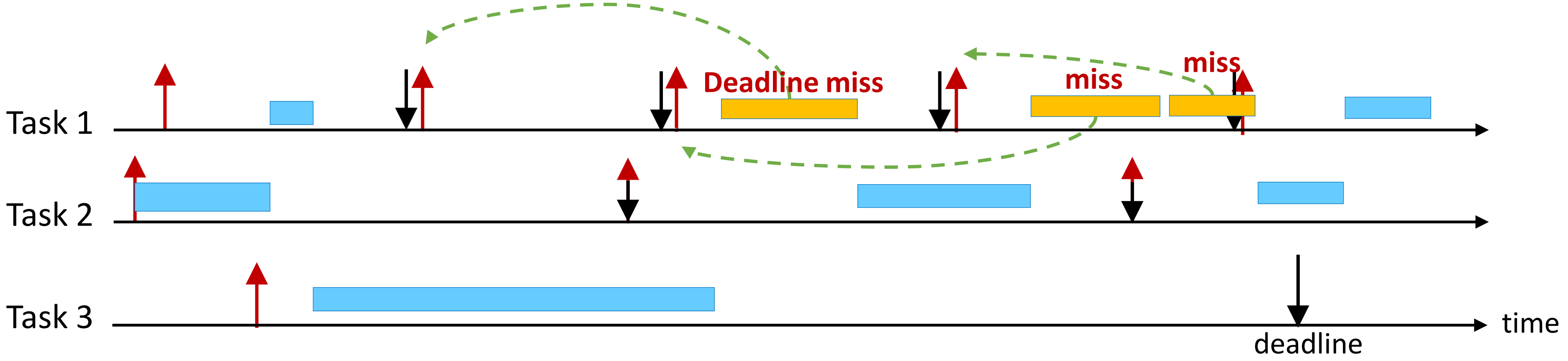
🟦 Execution ↑Release time ↓Deadline

Impact of scheduling policy on a task's response time

Well-known online scheduling policies:

- First-in-first-out (FIFO or FCFS)





Low runtime overhead

Minimizes the I/O delay
(via non-preemptive execution)

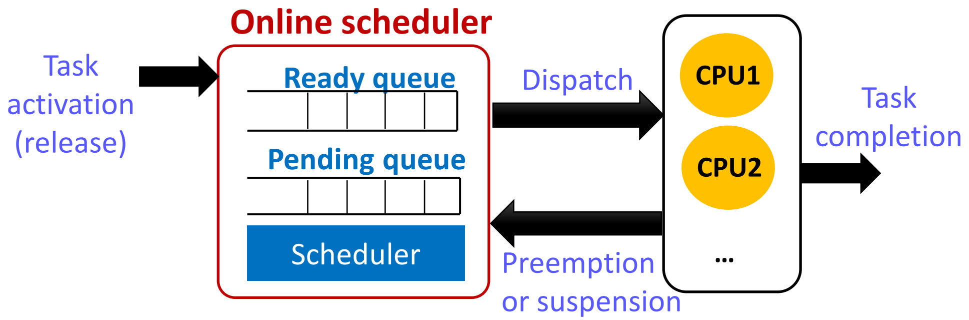
Low success in respecting timing constraints
(has no notion of deadline or priority)

🟦 Execution ↑ Release time ↓ Deadline

Impact of scheduling policy on a task's response time

Well-known online scheduling policies:

- First-in-first-out (FIFO or FCFS)
- Fixed-priority scheduling



Online scheduler

Task activation (release)

Ready queue

Pending queue

Scheduler

Dispatch

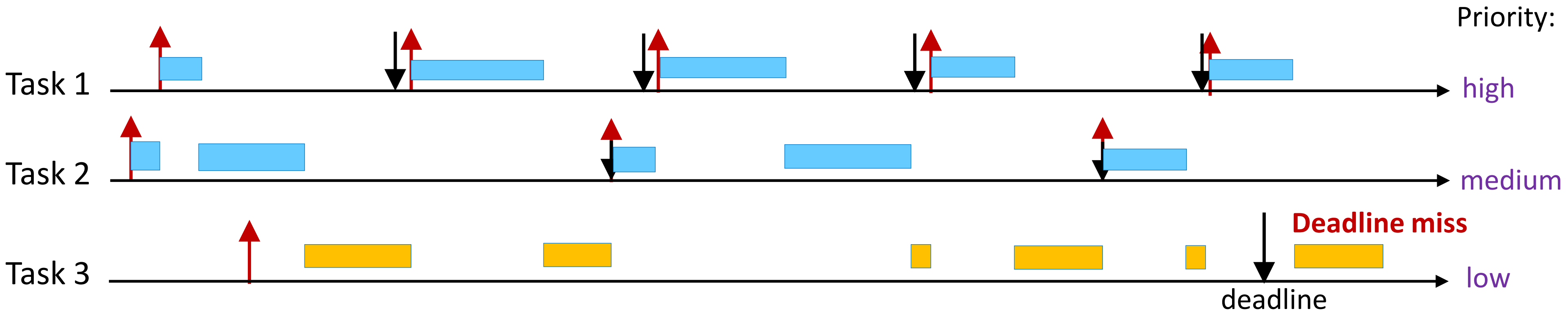
Preemption or suspension

CPU1

CPU2

…

Task completion



Relatively low overhead

Minimizes the sampling and I/O delay of the highest-priority task

Imposes preemptions and hence context switch overheads

Poor support for timing constraints of low-priority tasks

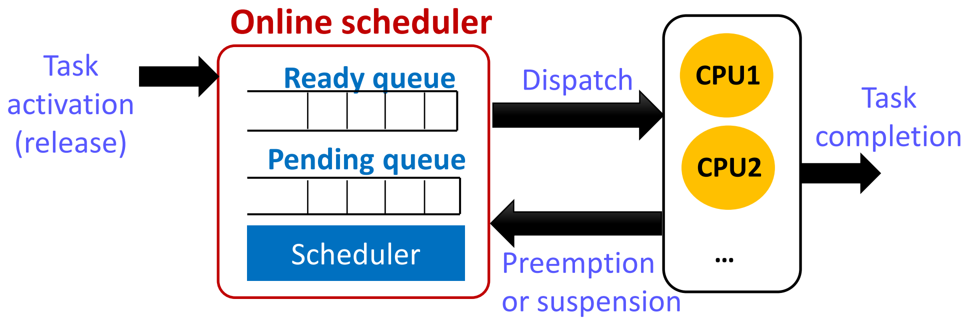
Its effectiveness highly depends on task periods and execution times

If a high-priority task is very long, other low-priority but frequent tasks may miss their deadlines

Impact of scheduling policy on a task’s response time

Well-known online scheduling policies:

- First-in-first-out (FIFO or FCFS)
- Fixed-priority scheduling
- Earliest-deadline first (EDF)



Online scheduler

Task activation (release)

Ready queue

Pending queue

Scheduler

Dispatch

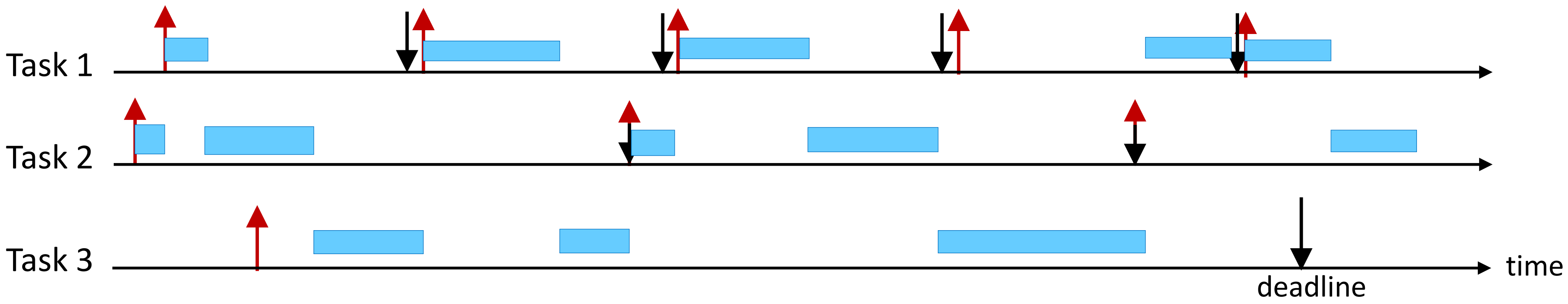
Preemption or suspension

CPU1

CPU2

...

Task completion



Rather high runtime overhead (needs a sorted queue)

Imposes preemptions and hence context switch overheads

Does not minimize I/O or sampling delays

Optimal w.r.t. meeting deadlines (only on single-core platforms, …)

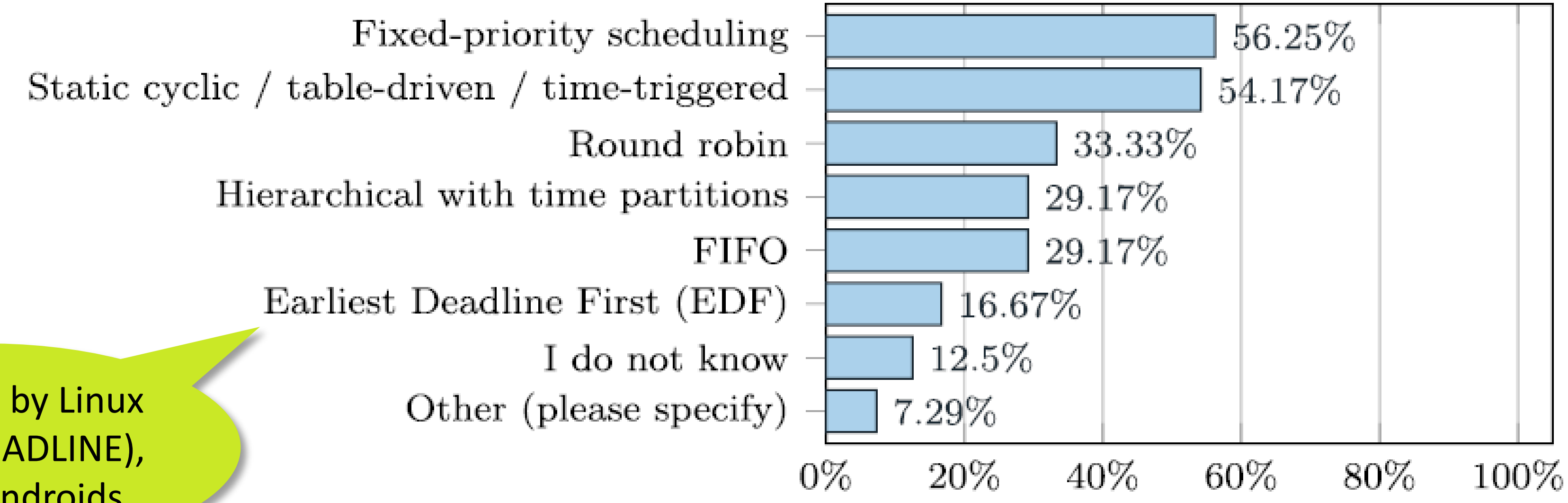
Not optimal on multi-core platforms or in the presence of context switch and precedence constraints

Scheduling policies used in industrial real-time systems

Industry

Fixed-priority scheduling and table-driven scheduling are common in industry.

Systems may use different scheduling policies in different parts/nodes

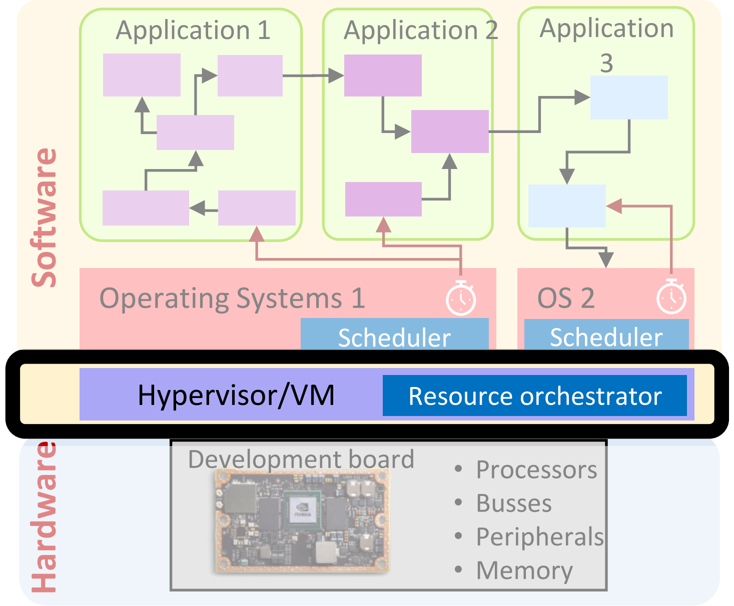


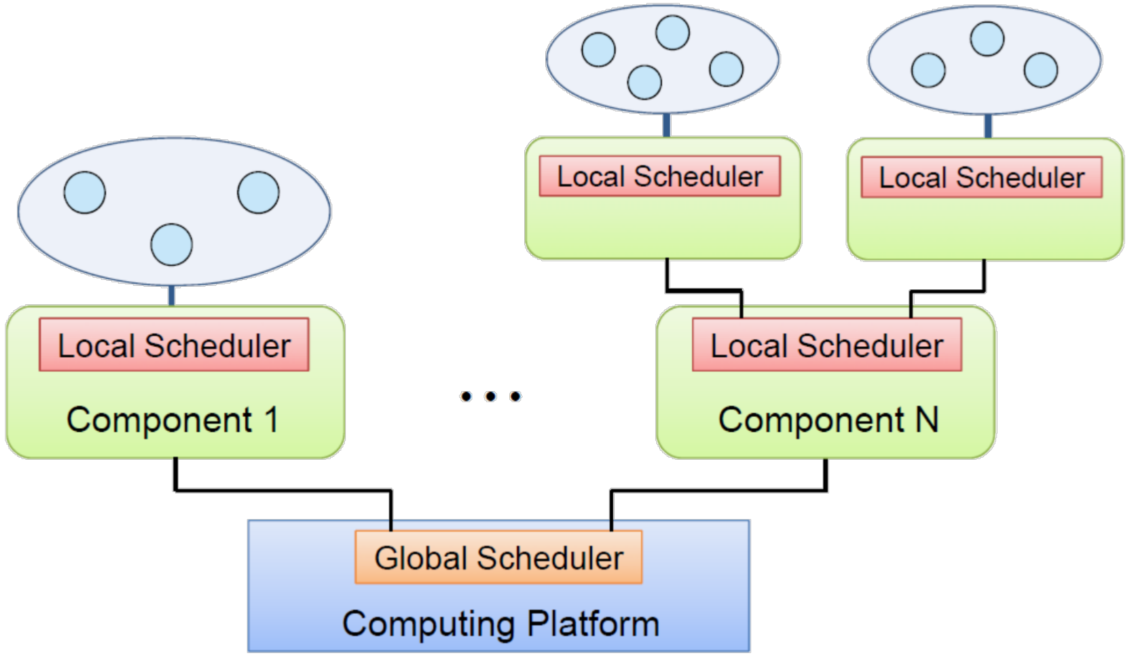
Supported by Linux (Sched_DEADLINE), used in Androids

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

Impacts of the virtualization platform on a task's response time

- **Identifying available resources (cores, memory, …)**
  - In embedded systems, **resources** are **static** and known in advance
- **Mapping**
  - Mapping of tasks to components (reservation servers)
  - Mapping of reservation servers to [hardware] resources
  - Dynamic mapping v.s. static mapping
- **Configurations**
  - Server's type, period, budget, budget-update function





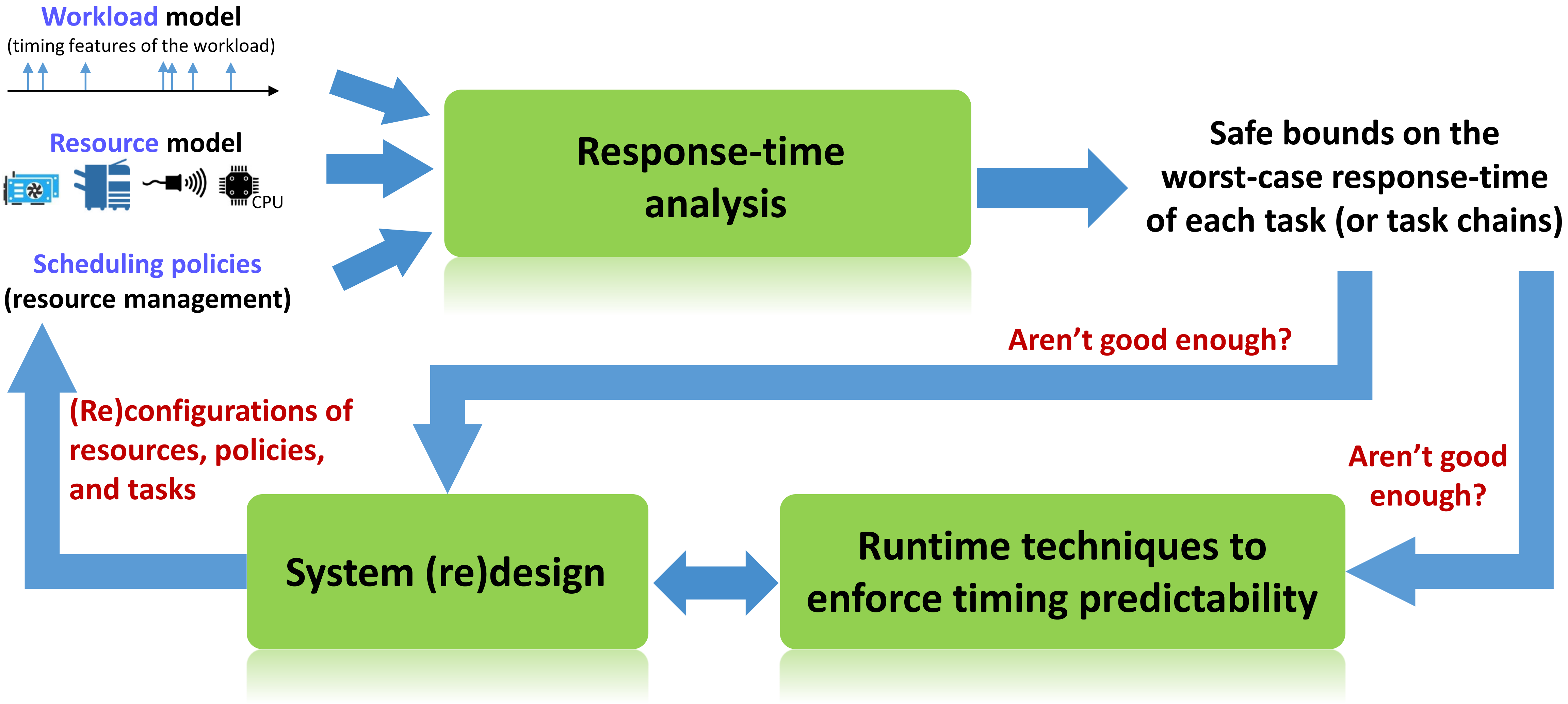
In cloud platforms, resources (CPUs and memory) can be **dynamic**

- **Runtime monitoring** is needed
  - **Checking resource availability**
  - **Resource scaling** (trade-off between performance, timing constraints, and costs)

- Model-based design
- Model-based timing analysis
- Runtime and design-time techniques for timing predictability

The cornerstones of real-time systems' design

How to design/develop time-predictable systems?



Workload model
(timing features of the workload)

Resource model
CPU

Scheduling policies
(resource management)

Response-time analysis

Safe bounds on the worst-case response-time of each task (or task chains)

Aren't good enough?

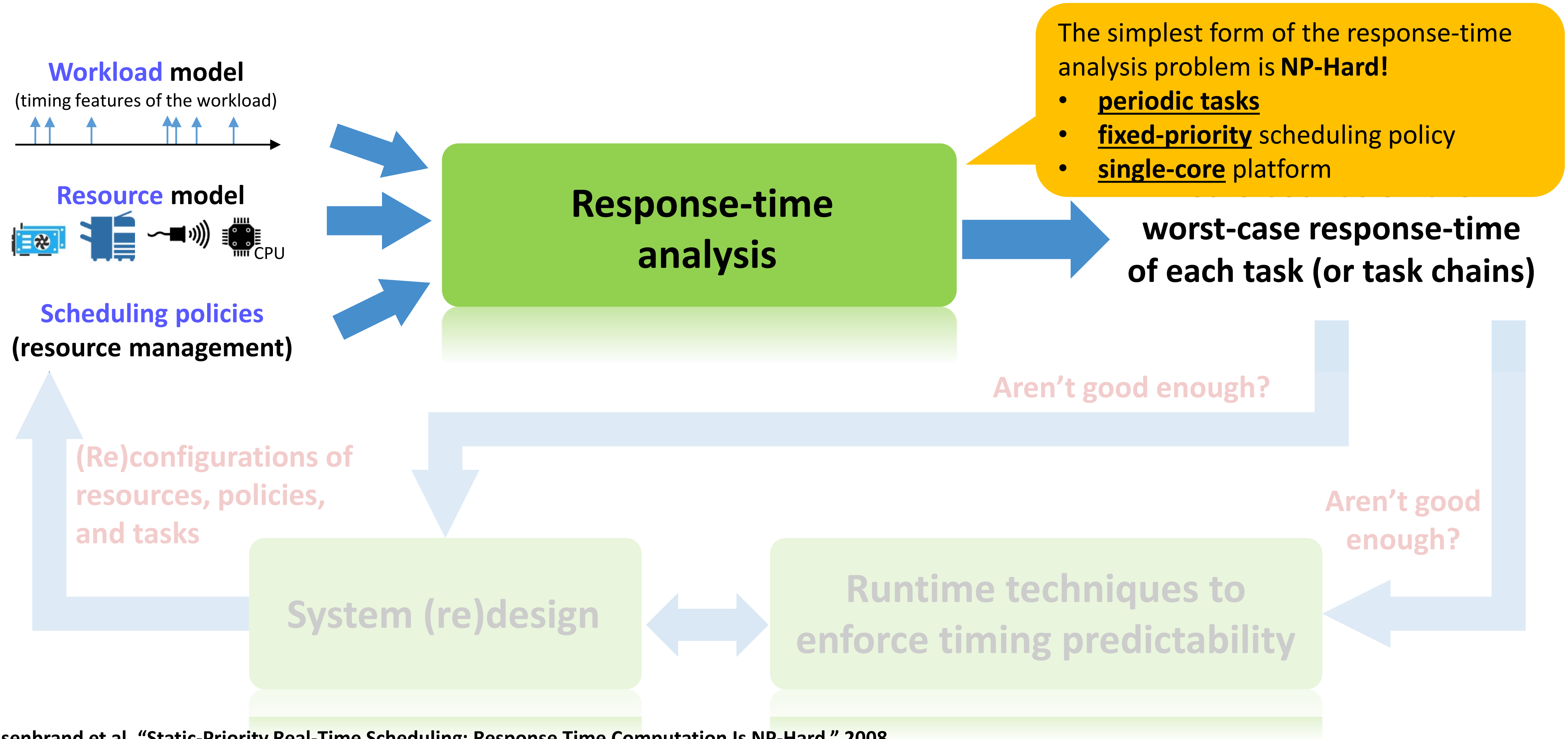
(Re)configurations of resources, policies, and tasks

Aren't good enough?

System (re)design

Runtime techniques to enforce timing predictability

How to assess if a system meets its timing constraints?



Workload model
(timing features of the workload)

Resource model

Scheduling policies
(resource management)

Response-time analysis

worst-case response-time of each task (or task chains)

The simplest form of the response-time analysis problem is **NP-Hard!**

- **periodic tasks**
- **fixed-priority** scheduling policy
- **single-core** platform

Aren't good enough?

(Re)configurations of resources, policies, and tasks

Aren't good enough?

System (re)design

Runtime techniques to enforce timing predictability

[1] . Eisenbrand et al. "Static-Priority Real-Time Scheduling: Response Time Computation Is NP-Hard," 2008.

[2] F. Eisenbrand et al. "EDF-schedulability of synchronous periodic task systems is coNP-hard," 2010.

A closer look at the response-time analysis problem

One of the simplest forms of the problem:

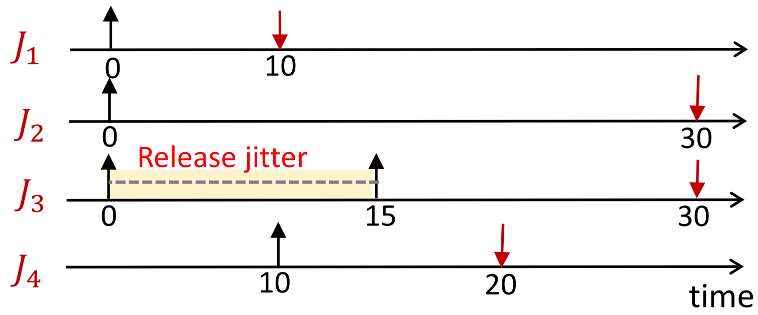
Response-time analysis problem

Given

- **a set of non-preemptive tasks/jobs** (with a given arrival interval, execution time, and deadline)
- **scheduled by a fixed-priority scheduling policy**
- **on a single-core platform,**

Determine

the worst-case response time of each job



Priorities are decided by the scheduling policy

Job	Release time Min	 Max	Deadline	Execution time Min	 Max	Priority
J_1	0	0	10	1	2	high
J_2	0	0	30	7	8	medium
J_3	0	15	30	3	13	low
J_4	10	10	20	1	2	high

Release jitter

Earliest release time

Latest release time

BCET

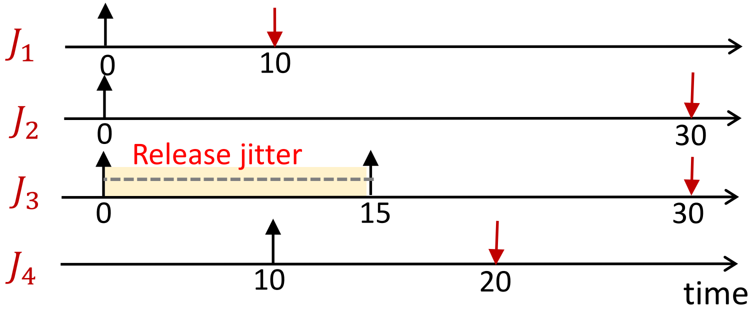
WCET

Uncertainty in execution time

A closer look at the response-time analysis problem

Goal: find the worst-case response time of each job
(for any imaginable schedule that is generated by a fixed-priority scheduling policy on one core)

Q: Why can't we "simulate" one schedule using a discrete-event simulator and see if there will be a deadline miss?



Job	Release time		Deadline	Execution time		Priority
	Min	Max		Min	Max	
J_1	0	0	10	1	2	high
J_2	0	0	30	7	8	medium
J_3	0	15	30	3	13	low
J_4	10	10	20	1	2	high

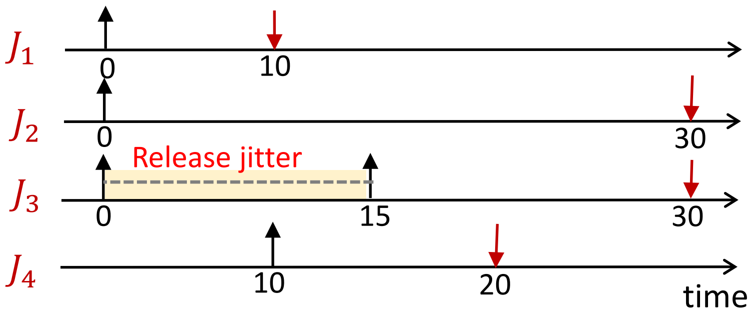
Earliest release time

Latest release time

BCET

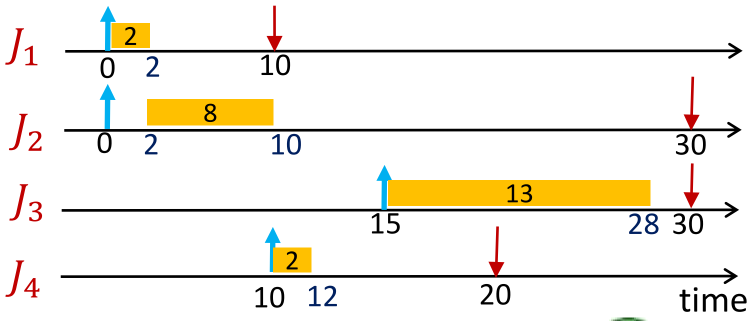
WCET

A closer look at the response-time analysis problem



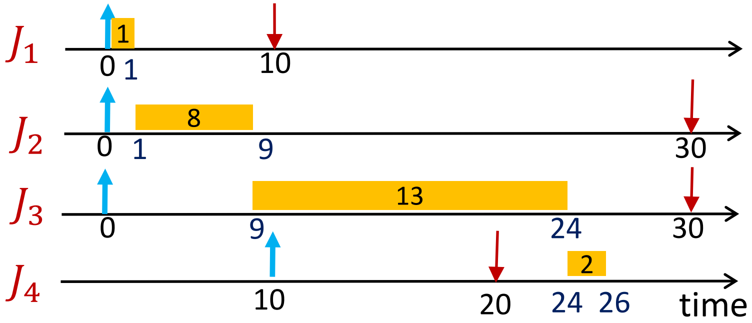
Job	Release time		Deadline	Execution time		Priority
	Min	Max		Min	Max	
J_1	0	0	10	1	2	high
J_2	0	0	30	7	8	medium
J_3	0	15	30	3	13	low
J_4	10	10	20	1	2	high

Execution scenario 1: jobs are released very **late** and have their largest execution time.



✓ **No deadline miss**

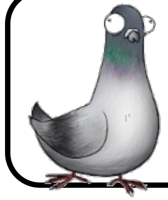
Execution scenario 2: jobs are released very **early** and have their largest execution time except for J_1.



Deadline miss for J_4

How should we find such a worst-case scenario?

A closer look at the response-time analysis problem



Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) **is not practical**

1200 different combinations for release times and execution times for a job set with 4 jobs!

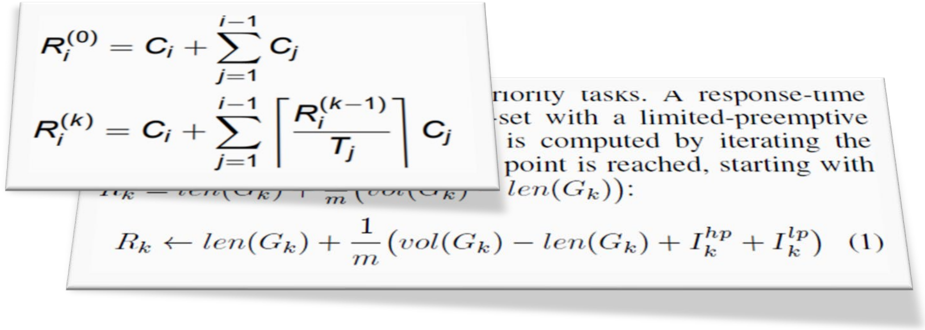
State of the art on response-time analysis

State of the art on response-time analysis

Fixed-point iteration-based analyses

✓
- Fast

✗
- Pessimistic
- Limited to periodic/sporadic arrival patterns
- Hard to extend

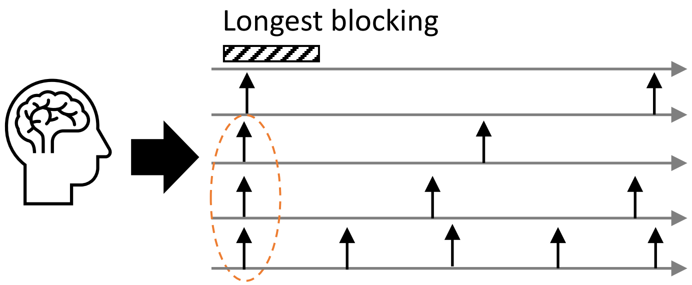


$$R_i^{(0)} = C_i + \sum_{j=1}^{i-1} C_j$$

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

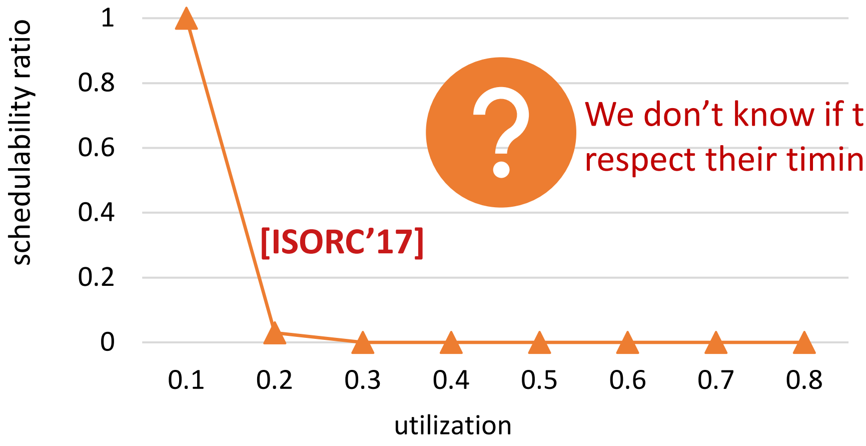
riority tasks. A response-time -set with a limited-preemptive is computed by iterating the point is reached, starting with $len(G_k)$):

$$R_k \leftarrow len(G_k) + \frac{1}{m}(vol(G_k) - len(G_k) + I_k^{hp} + I_k^{lp}) \quad (1)$$



Where has it taken us?

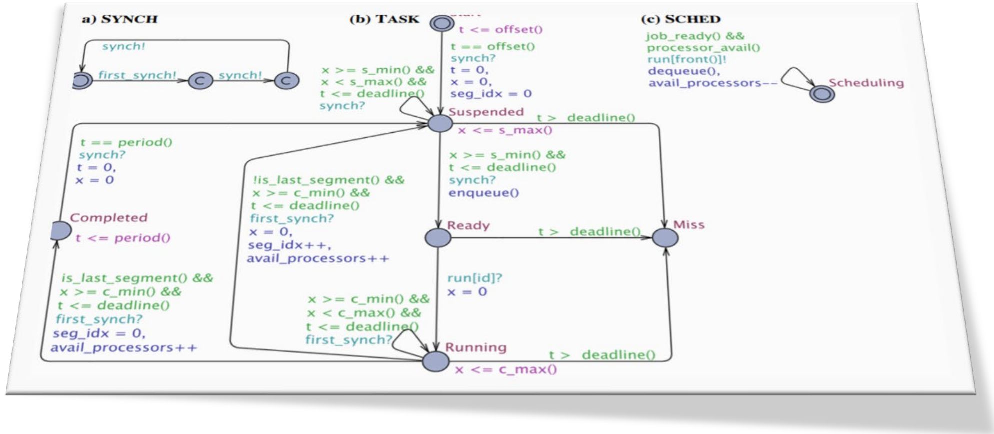
Experiment: limited-preemptive scheduling of parallel DAG tasks
Setup: 16 cores, 10 periodic DAG tasks



We don't know if these task sets respect their timing constraints

[ISORC'17]: Serrano et al., "An Analysis of Lazy and Eager Limited Preemption Approaches under DAG Based Global Fixed Priority Scheduling", ISORC, 2017.
Schedulability ratio = success ratio of an analysis to detect task sets that respect their timing constraints

State of the art on response-time analysis



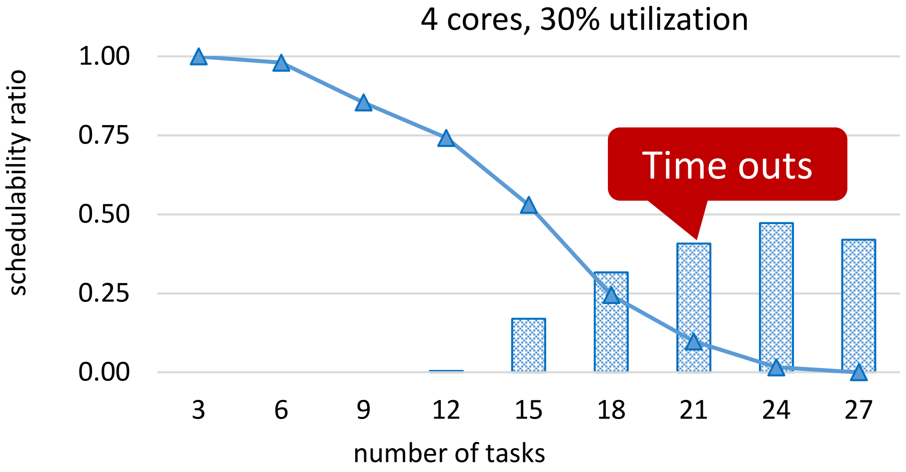
Exact analyses in generic formal verification tools (e.g., UPPAAL)

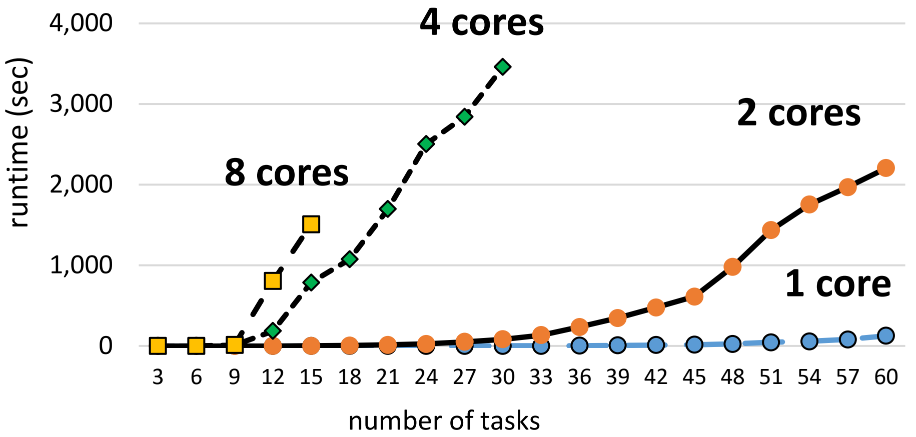
- **Accurate**
- **Easy to extend**

- **Not scalable**
- **Prone to model infidelity** (modeling mistakes)

There is a need for **generalizable, accurate,** and **scalable** response-time analysis

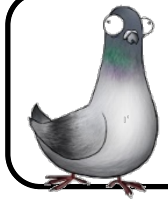
Where has it taken us?





Setup: sequential non-preemptive periodic tasks scheduled by global fixed-priority scheduling policy (FP)

A closer look at the response-time analysis problem

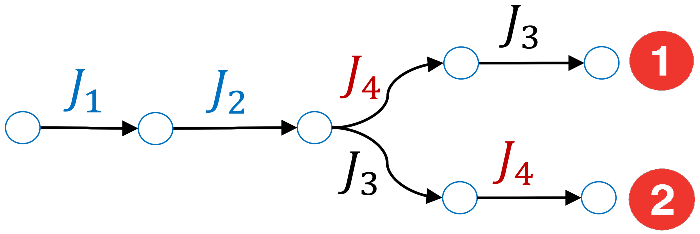


Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) **is not practical**

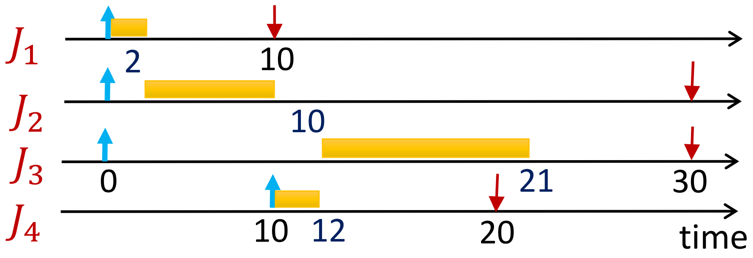
1200 different combinations for release times and execution times for a job set with 4 jobs!

Our observation:

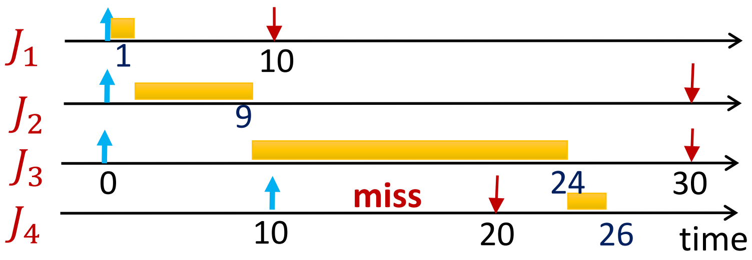
There are fewer permissible job orderings than schedules



Example for path ①

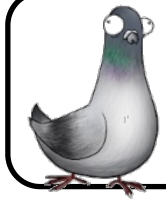


Example for path ②



- **2 possible job ordering**
- **1200 different combinations for release times and execution times**

A closer look at the response-time analysis problem



Naively enumerating all possible combinations of release times and execution times (a.k.a. execution scenarios) **is not practical**

Our observation:

There are fewer permissible job orderings than schedules

Solution idea:

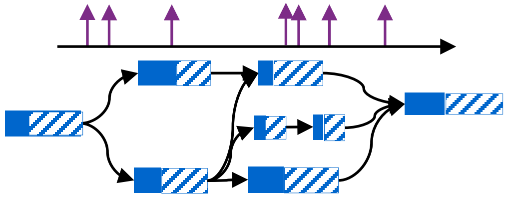
We use job-ordering abstraction to build a graph that abstracts all possible schedules

It is called the “schedule-abstraction graph”

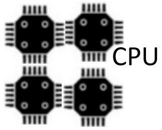
Goal: an *accurate* and *efficient* analysis

Schedule-abstraction graph in a nutshell

Workload model
(timing features of the workload)



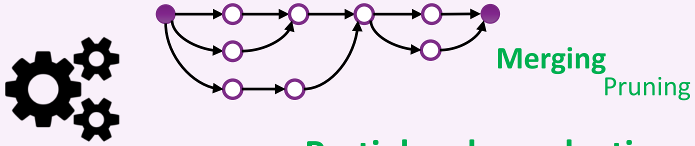
Resource model



Scheduling policy
(job-level fixed-priority policies)

3000 times faster than
generic verification tools (e.g., **UPPAAL**)

Schedule-abstraction graph
(a reachability-based response-time analysis framework)



Partial-order reduction

Response-time bounds

In our RTAS’22 work, we made it
5 orders-of-magnitude faster
using partial-order-reduction

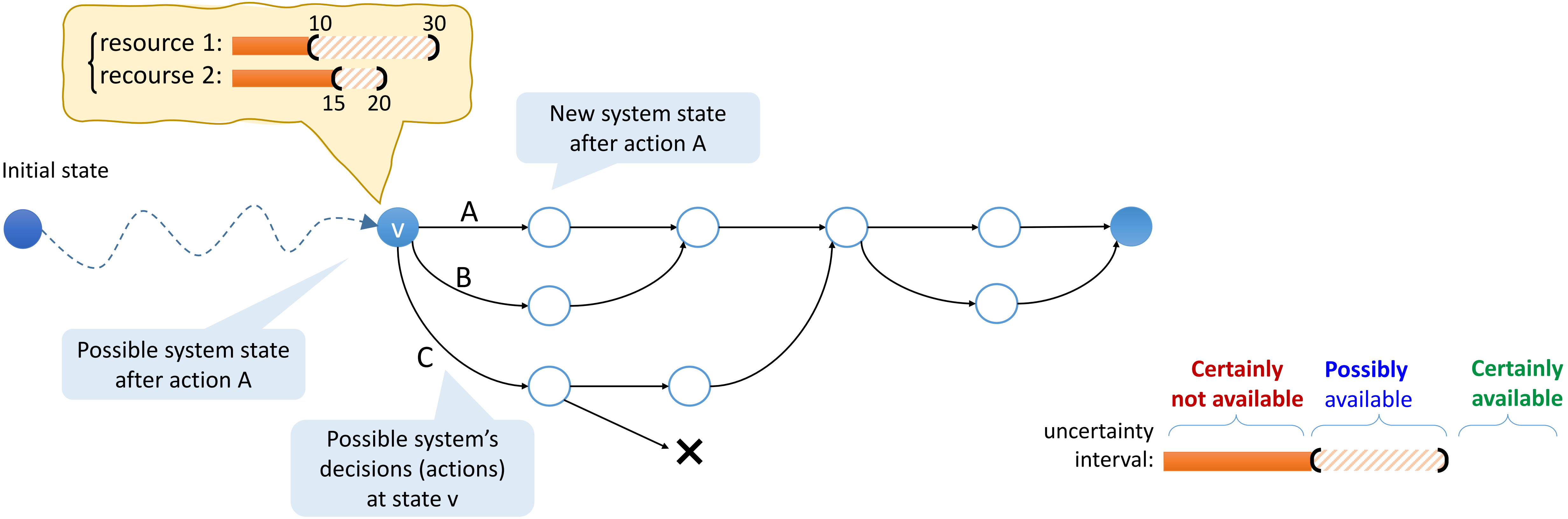
(it is a formal verification engine dedicated to **timing models** and **timing properties)**

Many top-rank conference papers
[RTSS’17, ECRTS’18, ECRTS’19, DATE’19, RTSS’20, RTSS’21, RTAS’22 (best-paper award), RTNS’22, ECRTS’22]
Open-source implementation: https://github.com/gnelissen/np-schedulability-analysis

How does it work?

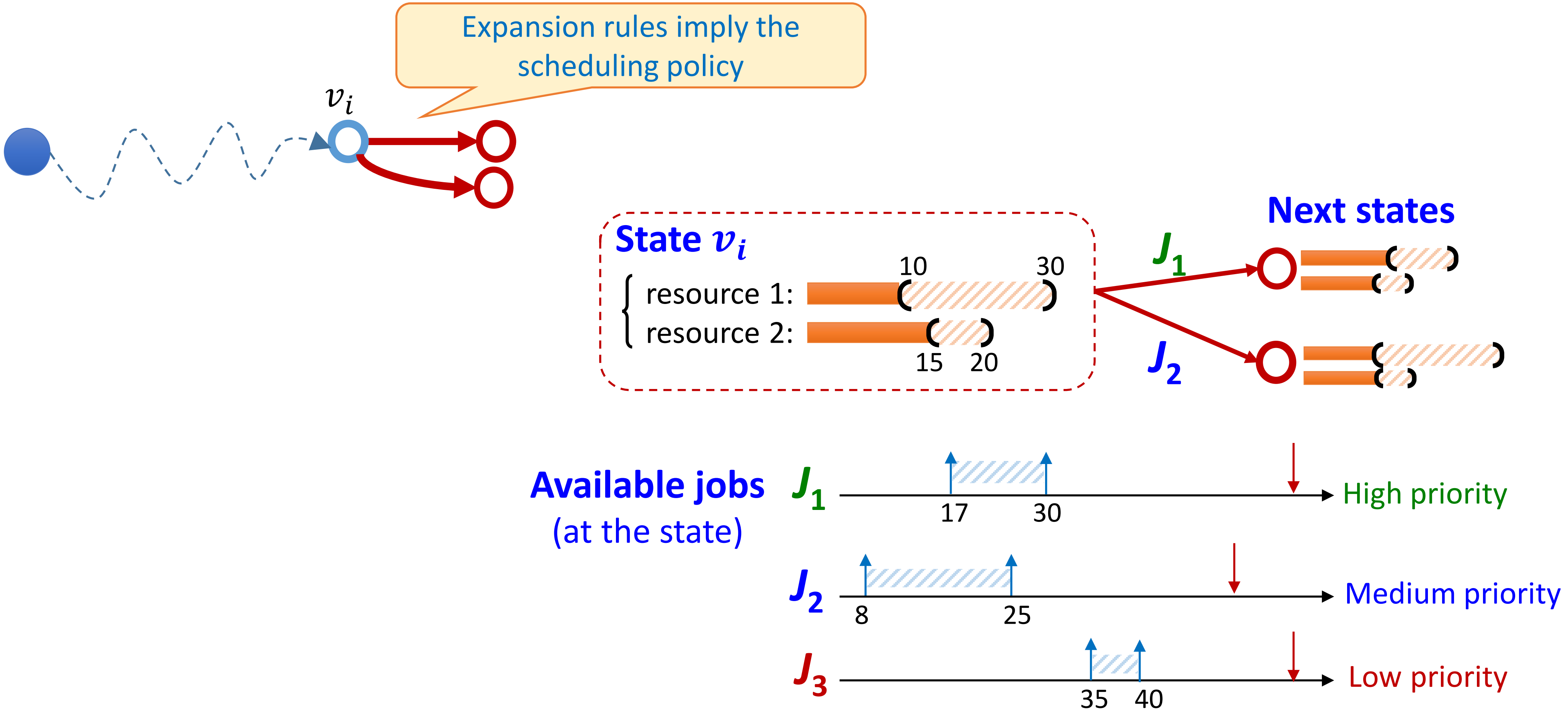
Our solution is a **reachability analysis** that

- Uses **uncertainty intervals** to combine uncertainties in the platform and task activation patterns
- **Merges states whose future is similar**
- **Does not** explore paths that **do not contribute to the worst-case behavior**



This work is in collaboration with the IRIS group (M&CS)

Handling uncertainty



Expansion rules imply the scheduling policy

v_i

State v_i

resource 1: 10 30

resource 2: 15 20

Next states

J_1

J_2

Available jobs

(at the state)

J_1 17 30 High priority

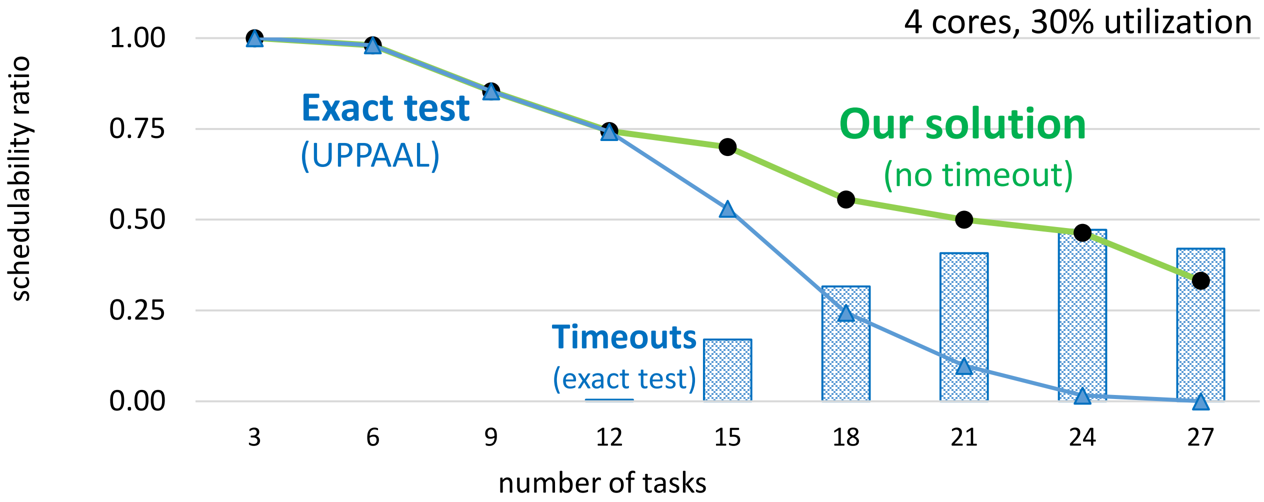
J_2 8 25 Medium priority

J_3 35 40 Low priority

[ECRTS'2018]

Taste of results: sequential tasks (global scheduling)

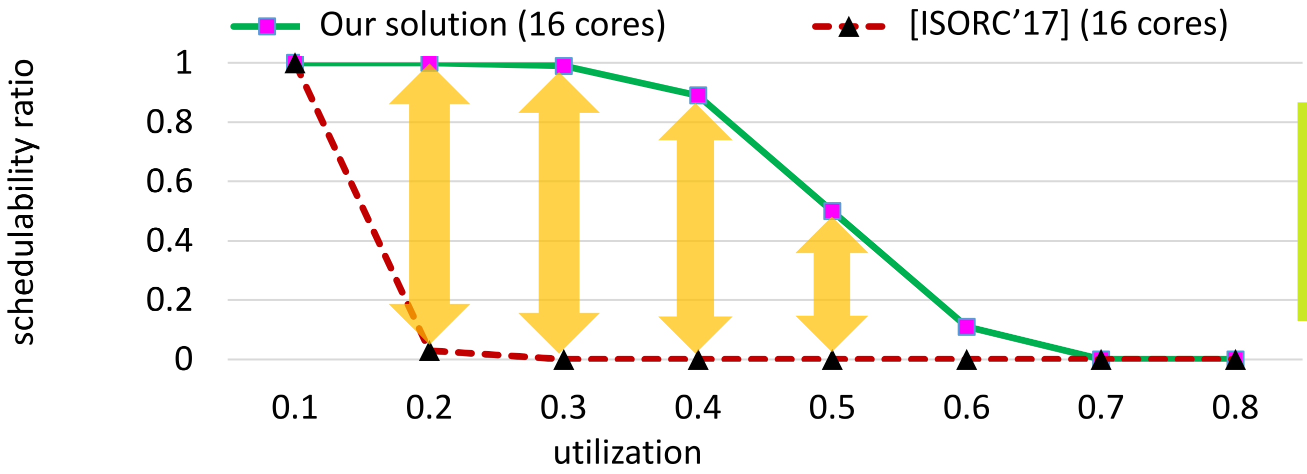
Comparison to UPPAAL



Almost as accurate as the exact test

Yet, 3000 times faster

Comparison to the fixed-point iteration-based methods



3.5 times more successful
(in determining whether a system meets its timing constraints)

[Exact test] Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg, "An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks", DATE, 2019.
[ISORC'17] M. Serrano, et al., "An Analysis of Lazy and Eager Limited Preemption Approaches under DAG-Based Global Fixed Priority Scheduling", ISORC, 2017.
[Our solution] Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg, "A Response-Time Analysis for Non-preemptive Job Sets under Global Scheduling," ECRTS, 2019.

- Model-based design
- Model-based timing analysis
- Runtime and design-time techniques for timing predictability

The cornerstones of real-time systems’ design

Designing for timing predictability

Application oriented

OS oriented

Hardware oriented

Network oriented

Design-time techniques

General trends

- **Analyzing** a given **COTS component** to obtain its **worst-case timing behavior**
- **(Re)configuring** existing [COTS] components for better predictability
- **Building** more **time-predictable** SW/HW components or networks

Runtime techniques

General trends

- **Monitoring** timing behavior
- **Enforcing** time-predictive behavior
- **Runtime verification** (ensuring correct timing behavior)

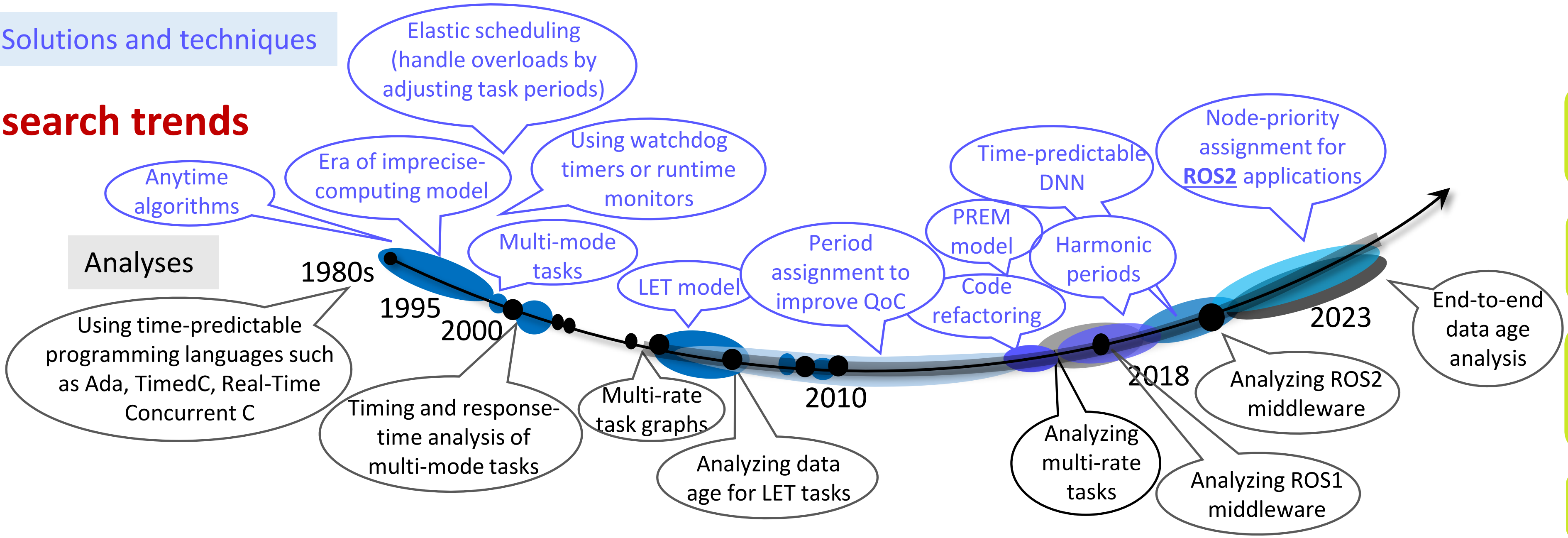
Application-oriented techniques for timing predictability

Developing more time-predictable applications

Trading execution time and quality

Trading period and quality

Multi-rate task graphs



Solutions and techniques

Analyses

Research trends

Elastic scheduling (handle overloads by adjusting task periods)

Anytime algorithms

Era of imprecise-computing model

Using watchdog timers or runtime monitors

Multi-mode tasks

LET model

Period assignment to improve QoC

PREM model

Code refactoring

Time-predictable DNN

Harmonic periods

Node-priority assignment for **ROS2** applications

Using time-predictable programming languages such as Ada, TimedC, Real-Time Concurrent C

Timing and response-time analysis of multi-mode tasks

Multi-rate task graphs

Analyzing data age for LET tasks

Analyzing multi-rate tasks

Analyzing ROS1 middleware

Analyzing ROS2 middleware

End-to-end data age analysis

Future trends

Time-predictable robotics via ROS2

Real-time applications for **edge** and **cloud**

Multi-rate task graphs: **larger** and **complex timing constraints**

Time-predictable AI

Geoffrey Nelissen (TU/e)

Dakshina Dasari (Bosch)

Available during the CompSys industrial panel.

Ask **me** or **them** about ROS and Task Graphs

[1] Björn Brandenburg and Tobias Blaß works on ROS 2 Response-Time Analysis.
[2] Mitra Nasri works on assigning harmonic periods
[3] Enrico Bini, Morteza Mohaqeqi, Anton Cervin, Karl-Erik Arzen, and Mitra Nasri works on assigning period values to improve quality of control.
[4] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeniet al., "Elastic Scheduling for Flexible Workload Management," 2002.
[5] Seminal papers: https://cmte.ieee.org/tcrts/education/seminal-papers/
[6] Cong Liu works on time-predictable DNN
[7] Mathias Becker, Dakshina Dasari, Daniel Cassini, and Mitra Nasri works on the analysis of multi-rate task graphs.

Operating-System-oriented techniques for timing predictability

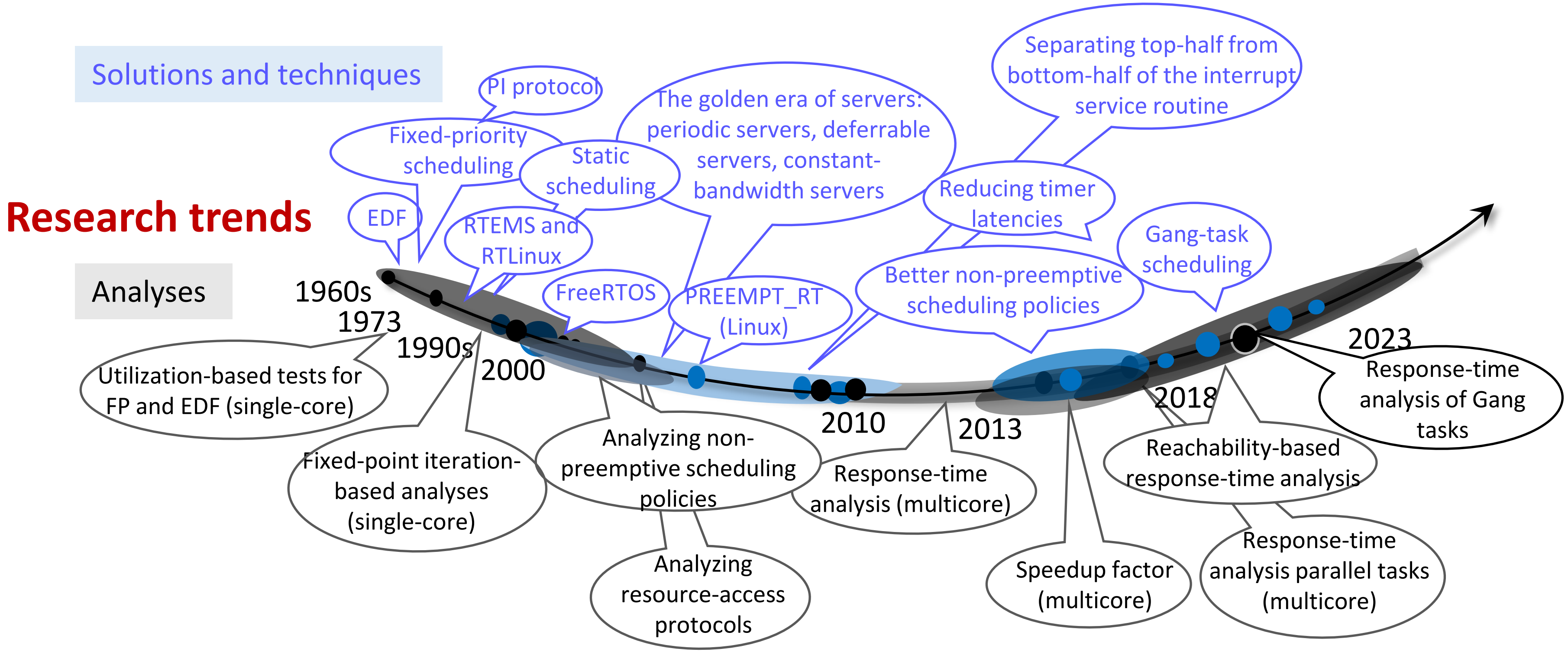
Scheduling policies and RTOSes

Better interrupt service routines and timers

Resource-access protocols (Priority Ceiling, Priority Inheritance, MrsP)

Reservation-based scheduling

Response-time analyses



Future trends

Resource orchestration in **edge** and **cloud**

Policing/orchestrating Memory and I/O Bandwidth, and GPU access management

Response-time analyses for generic scheduling problems

Time-predictable resource-access mechanisms

[1] Seminal papers: https://cmte.ieee.org/tcrts/education/seminal-papers/
[2] F. Reghenzani, et al. "The Real-Time Linux Kernel: A Survey on PREEMPT_RT," 2019.
[3] N. C. Audsley, et al., "Fixed Priority Scheduling: A Historical Perspective", 1995.
[4] L. Sha, et al., "Real-Time Scheduling Theory: A Historical Perspective", 2004.
[5] K. Jeffay and D. L. Stone, "Accounting for interrupt handling costs in dynamic priority task systems," 1993.
[6] C. Mercer, S. Savage, and H. Tokuda, "Temporal protection in real-time operating systems" 1994.
[7] L. Abeni and G. C. Buttazzo, "Resource Reservation in Dynamic Real-Time Systems," 2004.
[8] Herman Kopetz, Gerhad Fohler: Time-Triggered Scheduling and slot shifting
[9] L. Sha, R. Rajkumar, and J. P. Lehoczkyet al., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," 1990.
[10] R.I. Davis and A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems," 2011.
[12] M. Nasri works on Schedule-Abstraction Graph (reachability-based response-time analyses)
…

Hardware-oriented solutions for timing predictability

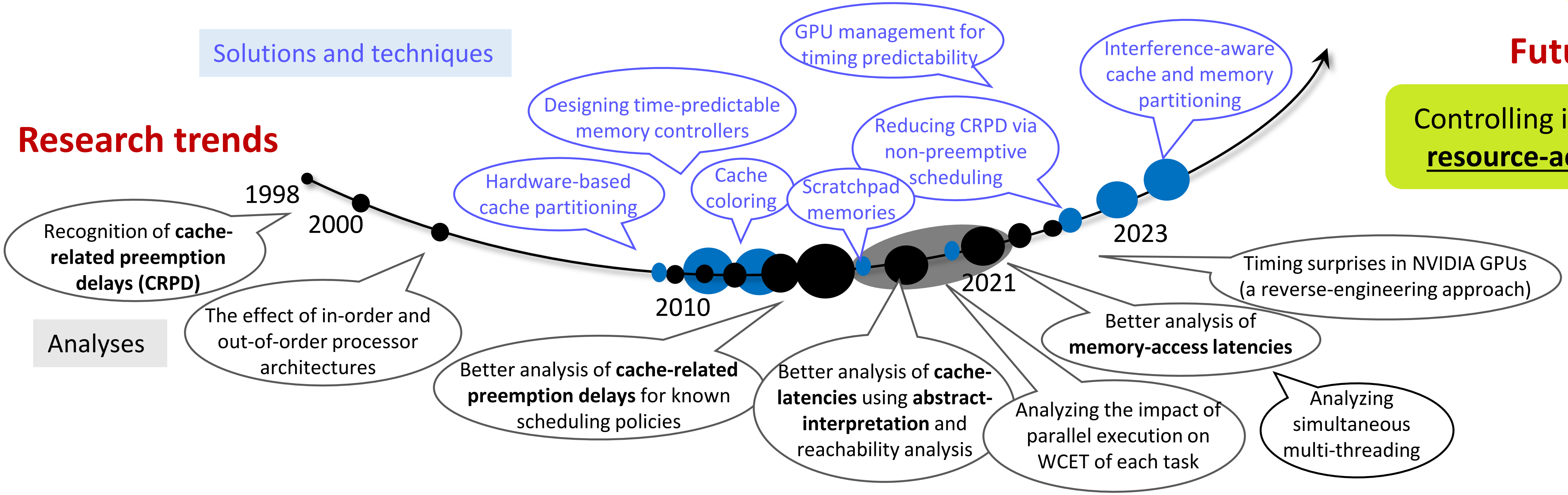
Building more time-predictable **cache, memories, and memory controllers**

More accurate estimation of **cache-** and **DRAM-related latencies** of COTS hardware

ARM, Intel, and NVIDIA are already integrating it in their current platforms

Future trends

Controlling interference through **resource-access orchestration**



Solutions and techniques

Research trends

1998

2000

2010

2021

2023

Analyses

Recognition of **cache-related preemption delays (CRPD)**

The effect of in-order and out-of-order processor architectures

Hardware-based cache partitioning

Designing time-predictable memory controllers

Cache coloring

Scratchpad memories

GPU management for timing predictability

Reducing CRPD via non-preemptive scheduling

Interference-aware cache and memory partitioning

Better analysis of **cache-related preemption delays** for known scheduling policies

Better analysis of **cache-latencies** using **abstract-interpretation** and reachability analysis

Analyzing the impact of parallel execution on WCET of each task

Better analysis of **memory-access latencies**

Analyzing simultaneous multi-threading

Timing surprises in NVIDIA GPUs (a reverse-engineering approach)

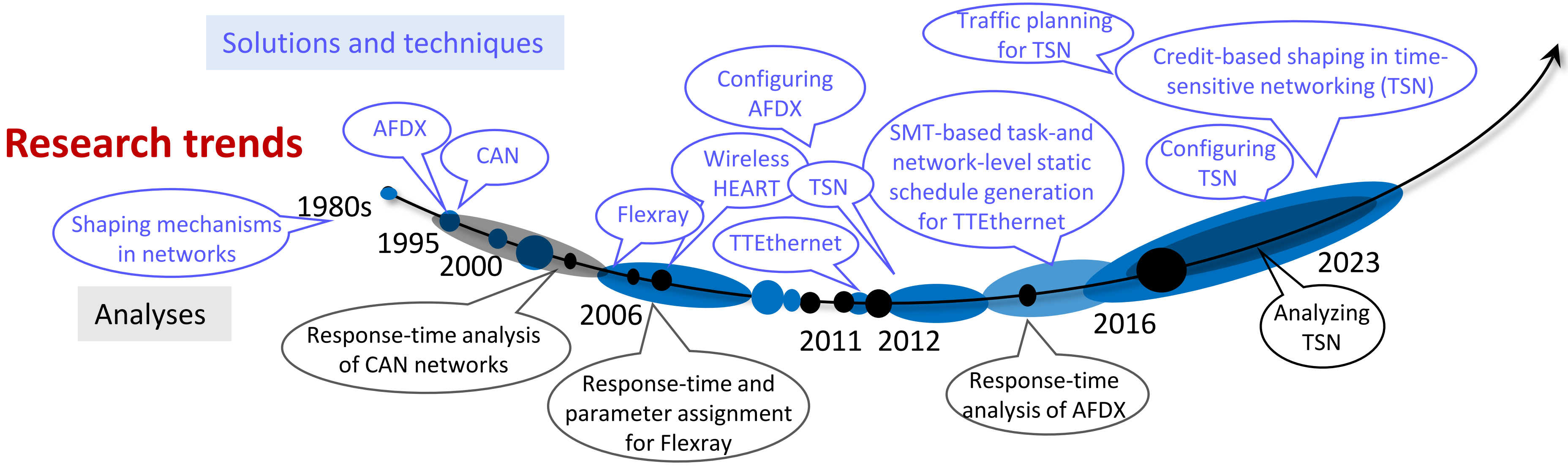
[1] R. Wilhelm, et al., "The worst-case execution-time problem—overview of methods and survey of tools," ACM Transactions on Embedded Computing Systems, 2008.
[2] T. Hiroyuki and N. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," International workshop on Hardware/software codesign, 2000.
[3] S. Altmeyer and C. Maiza, "Cache-related preemption delay via useful cache blocks: Survey and redefinition," Journal of Systems Architecture, 2011.
[4] J. Xiao, Y. Shen, A. Pimentel , "Cache Interference-aware Task Partitioning for Non-preemptive Real-time Multi-core Systems," ACM TECS, 2022.
[5] A. Rashid, G. Nelissen, and E. Tovar, "Tightening the CRPD bound for multilevel non-inclusive caches., Journal of Systems Architecture, 2022.
[6] Works of Marco Caccamo, Rodolfo Pelizzoni, and Renato Mancuso on MemGaurd, works of Kees Goossens on CompSoc, works of Jim Anderson on GPUs, PhD thesis of Mohamed Hassan, Heechul Yun, and Benny Akesson on predictable memory controllers and DRAM, …
[7] M. Hassan, "On the Off-chip Memory Latency of Real-Time Systems: Is DDR DRAM Really the Best Option", RTSS, 2018.
[8] P. Sohal, R. Tabish, U. Drepper, R. Mancuso, "Profile-driven memory bandwidth management for accelerators and CPUs in QoS-enabled platforms," Real-Time Systems, 2022.
[9] M. Bechtel and H. Yun, "Cache Bank-Aware Denial-of-Service Attacks on Multicore ARM Processors," 2023.
[10] S. Osborne, "Simultaneous Multithreading and Hard Real Time: Can it be Safe?" 2020.

Time-predictable networking

CAN, Flexray, Ethernet, TTEthernet, AFDX, TSN

Analyzing end-to-end response time in networked real-time systems (CAN, LIN, Flexray, Ethernet, TTEthernet, TSN)

Generating **optimal routes and static schedules** and **configuring** network components



Solutions and techniques

Research trends

Analyses

Shaping mechanisms in networks

AFDX

CAN

Configuring AFDX

Wireless HEART

TSN

Flexray

TTEthernet

Traffic planning for TSN

SMT-based task-and network-level static schedule generation for TTEthernet

Credit-based shaping in time-sensitive networking (TSN)

Configuring TSN

1980s

1995

2000

2006

2011 2012

2016

2023

Response-time analysis of CAN networks

Response-time and parameter assignment for Flexray

Response-time analysis of AFDX

Analyzing TSN

Future trends

TSN

SDN for **real-time systems**

Time-predictable **IoT**

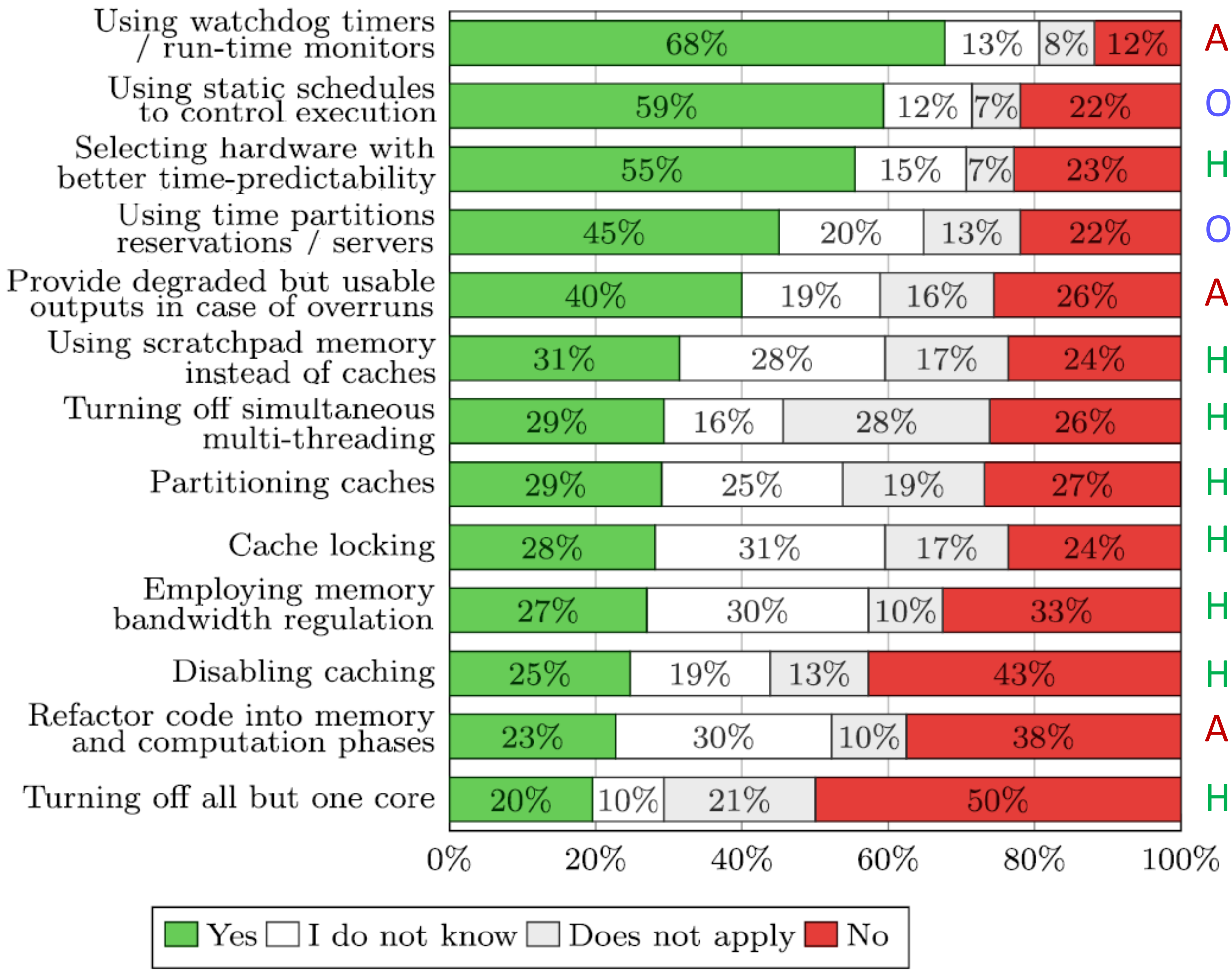
End-to-end analysis of heterogeneous networks

[1] Follow up on:
- Ramon Serna Oliver, Silviu Cracionas, and Hermann Kopetz: TT-Ethernet
- Ramon Serna Oliver, Silviu Cracionas, Mohammad Ashjaei, Luis Almeida, Frank Durr: TSN
- K. Tindell, A. Burns, and A. Wellings, R. Davis, R. Brill: analysis of CAN networks
- Lothar Thiele and Tarek Abdelzaher: real-time wireless sensor networks and wireless HEART
- Paup Pop, Petro Eles: Flexray analysis

[2] S. Craciunas, et al., “Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks,” 2016. (over 400 citations)
[3] S. Craciunas, et al., “An overview of scheduling mechanisms for time-sensitive networks”, 2017.
[4] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou, “Real-time communication and coordination in embedded sensor networks,” 2003.
[5] S. Serna Oliver, et al., “SMT-based task-and network-level static schedule generation for time-triggered networked systems,” 2014.
[6] S. Craciunas, et al., “Optimal static scheduling of real-time tasks on distributed time-triggered networked systems”, 2014.
[7] L Deng et al., “A survey of real-time ethernet modeling and design methodologies: From AVB to TSN,” 2022.
[8] V Gavriluţ et al., “Constructive or optimized: An overview of strategies to design networks for time-critical applications, “ 2022.
[9] T Pop, P Pop, P Eles, Z Peng, A Andrei, ”Timing analysis of the FlexRay communication protocol,” 2008.

Abbreviations
- Full-Duplex Switched Ethernet (AFDX), main target: avionics (Airbus)
  - Bandwidth guarantee for real-time applications + dual redundant channel for reliability
- Controller Area Networks (CAN): main target: automotive and manufacturing
- Flexray: a time-triggered protocol based on TDMA
- Time-Triggered Ethernet (TTEthernet)
- Time-Sensitive Networking (TSN)
- Software-Defined Networking (SDN)

Time-predictability techniques used in industry



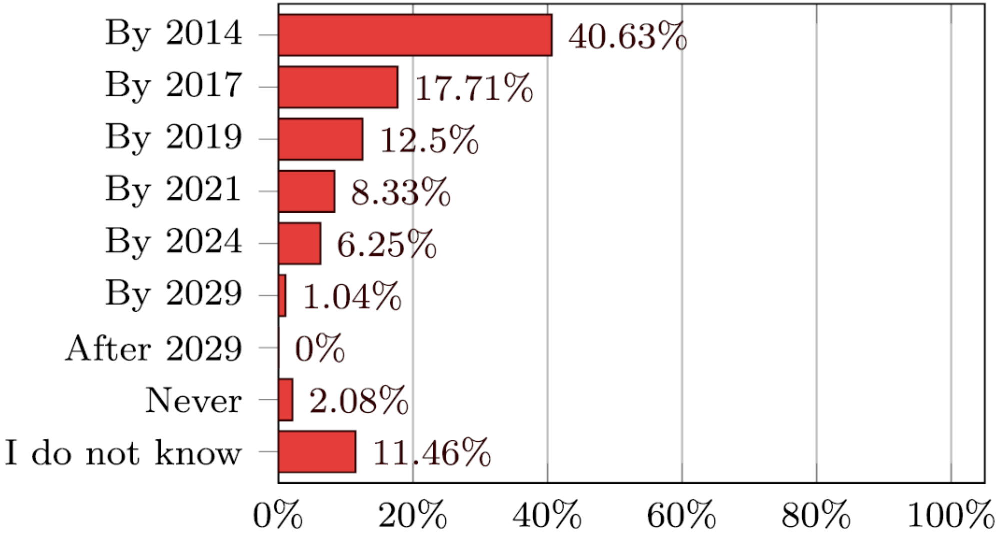
Application
OS-level
Hardware-level
OS-level
Application
Hardware-level
Hardware-level
Hardware-level
Hardware-level
Hardware-level
Hardware-level
Application
Hardware-level

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

Future trends in industry

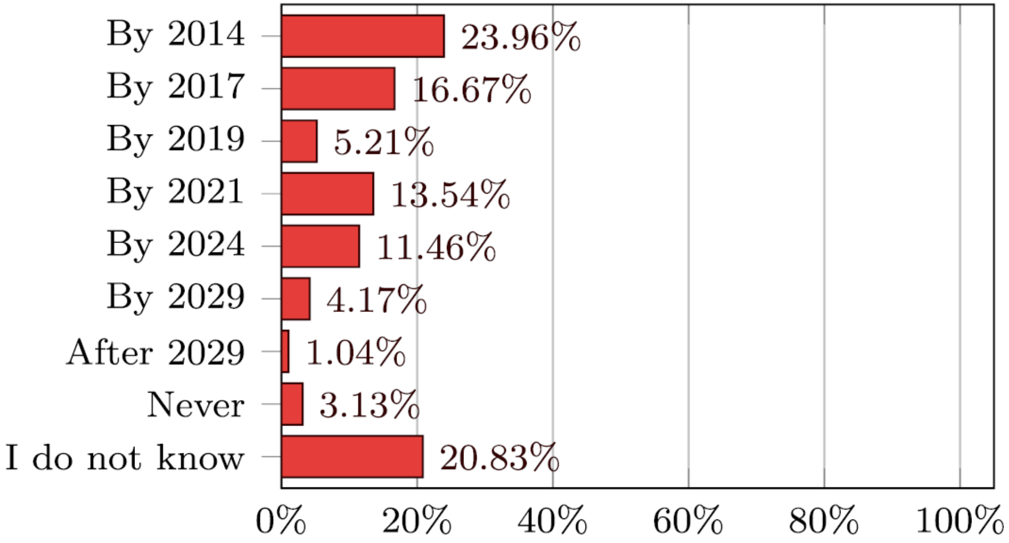
Use of multicore and MPSoC

The use of heterogeneous multi-cores (2 to 16 cores):



+85% of new developments by 2024 will use multicore

The use of heterogeneous multi-cores with different types of CPUs, GPUs, and other accelerators:



70% of new developments will use heterogeneous MPSoC by 2024

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "**A Comprehensive Survey of Industry Practice in Real-Time Systems**," Real-Time Systems Journal (RTS), Springer, 2021.

Mitra Nasri
m.nasri@tue.nl
Assistant professor
Eindhoven University of Technology (TU/e)

Future trends

Application-oriented solutions

- **Time-predictable robotics** via **ROS2** (robotic operating system)
- Real-time applications on **edge** and **cloud**
- Multi-rate task graphs: **larger** and **complex timing constraints**
- **Time-predictable AI**

OS-oriented solutions

- **Policing/orchestrating Memory Bandwidth, I/O, GPU management**
- **Generic frameworks for response-time analyses**
- **Time-predictable resource-access/management**

Hardware-oriented solutions

- Controlling interference through **<u>resource-access orchestration</u>**

Network-oriented solutions

- **TSN**
- **SDN** for **real-time systems**
- Time-predictable **IoT**

Real-time systems community:

- RTSS, RTAS, ECRTS, RTNS, RTCSA, EmSoft, Date (E2 topic), DAC
- **ACM SigBed**
- **IEEE TCRTS**