Configuring autonomous vehicles

Dr. Silviu S. Craciunas
Principal Scientist

TECHNICAL UNIVERSITY OF DENMARK
20 DECEMBER 2024

tttech auto

Copyright Notice

Project highlights

TTTech Auto safety and robustness in series production

Hardware development until the A-Sample grade

Scalable distributed control system from TTTech Industrial into Vestas 2, 4 and 6 Megawatt turbines

Collins Aerospace Power System and Air Management for the Boeing 787-8, 787-9, 787-10 family

TTTech TTEthernet Switch and End System IP products for the avionics system.

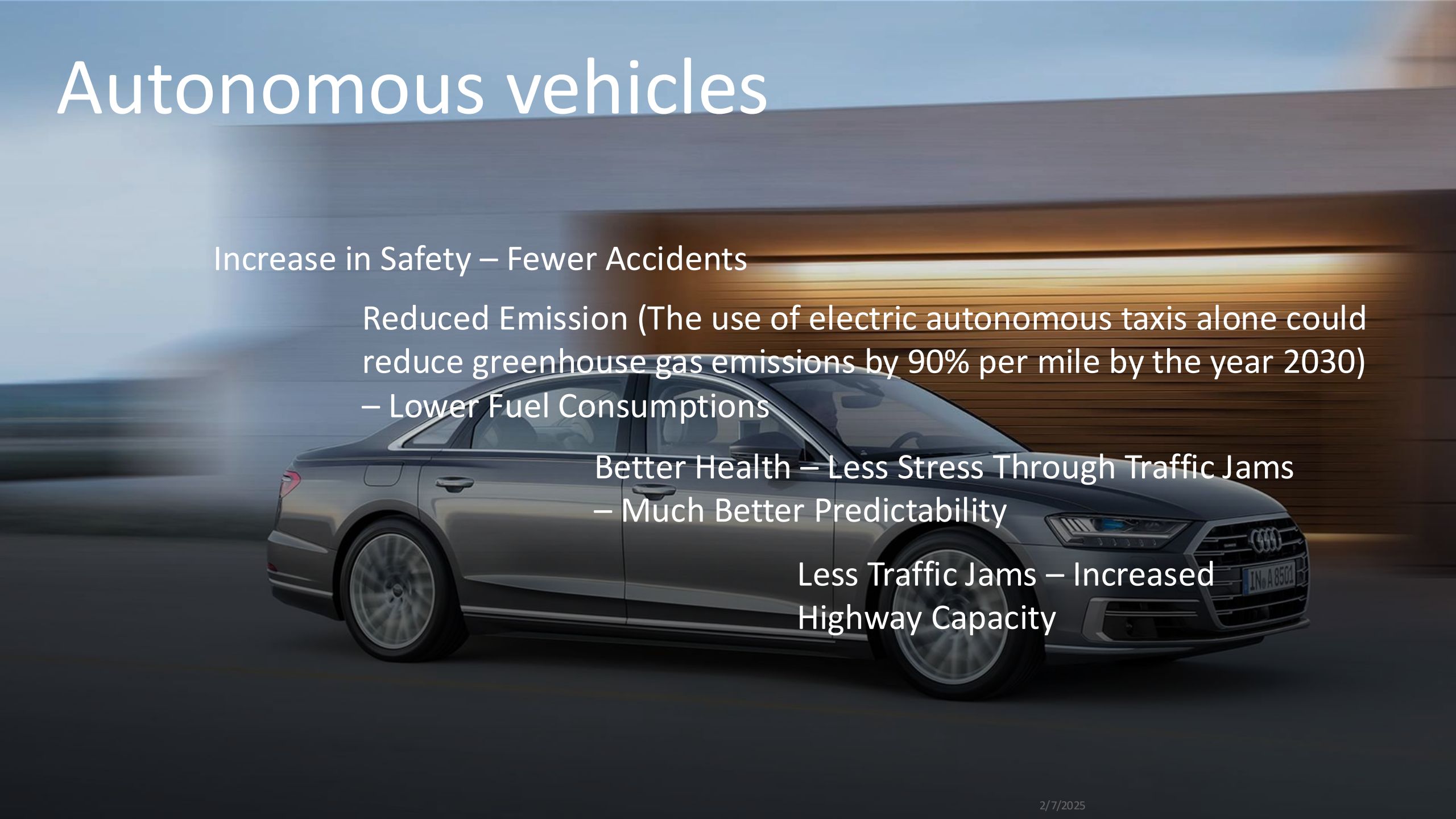
TTEthernet is the “nervous system“ i.e. avionics network platform of the space craft

NXP tttech auto

Accelerating software-defined vehicles together

Introduction

Autonomous vehicles



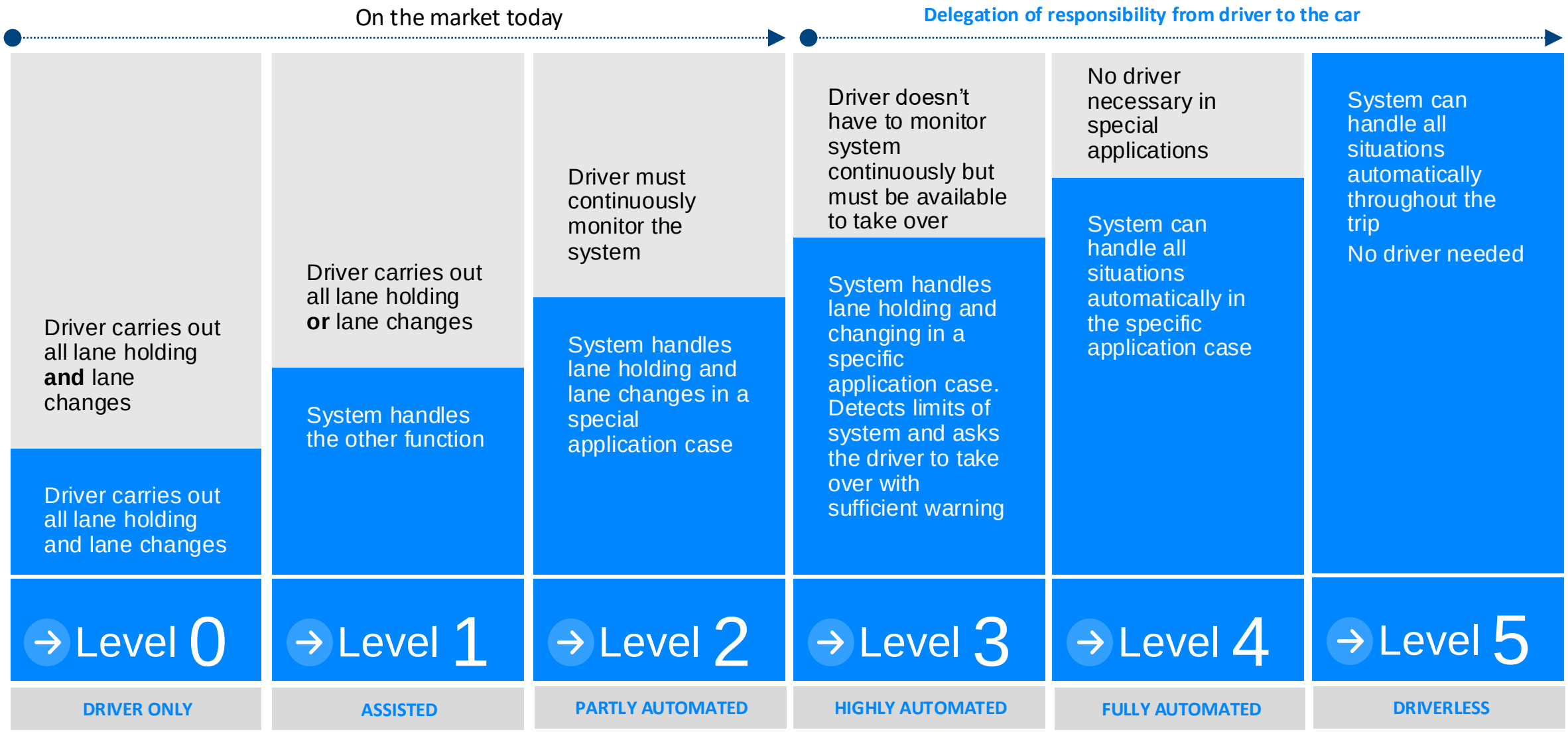
Increase in Safety – Fewer Accidents

Reduced Emission (The use of electric autonomous taxis alone could reduce greenhouse gas emissions by 90% per mile by the year 2030) – Lower Fuel Consumptions

Better Health – Less Stress Through Traffic Jams – Much Better Predictability

Less Traffic Jams – Increased Highway Capacity

Automated Driving - System Classification by VDA



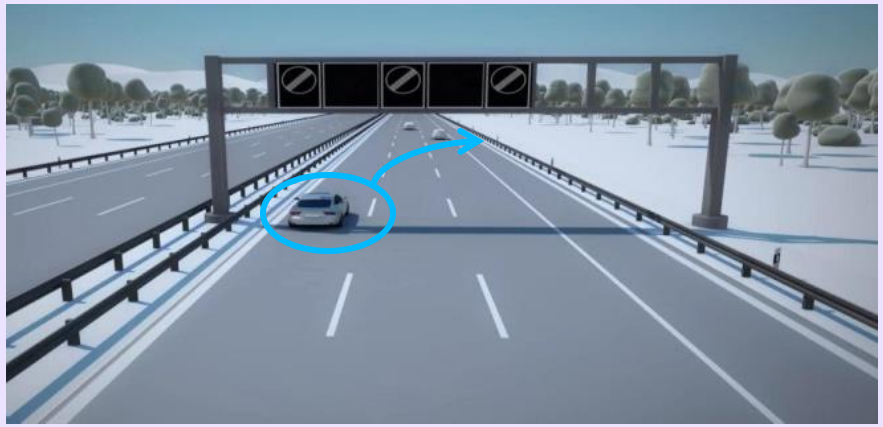
Challenge: The Fail-Operational Requirement

Level 3	“System detects its limits and asks driver to take over in case of faults”
Level 4	“System can handle all situations automatically in the specific application case”

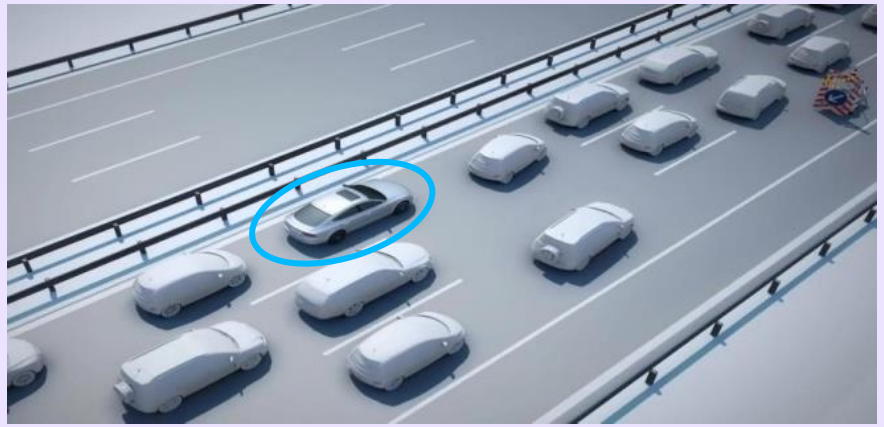
Already from Level 3 onwards there is a requirement for fail-operational behavior.

Component failures must be tolerated!

- Continue operation and request driver to take over (~10s in Level 3 systems)
- If driver does not take over, then situation specific reaction e.g. controlled stop



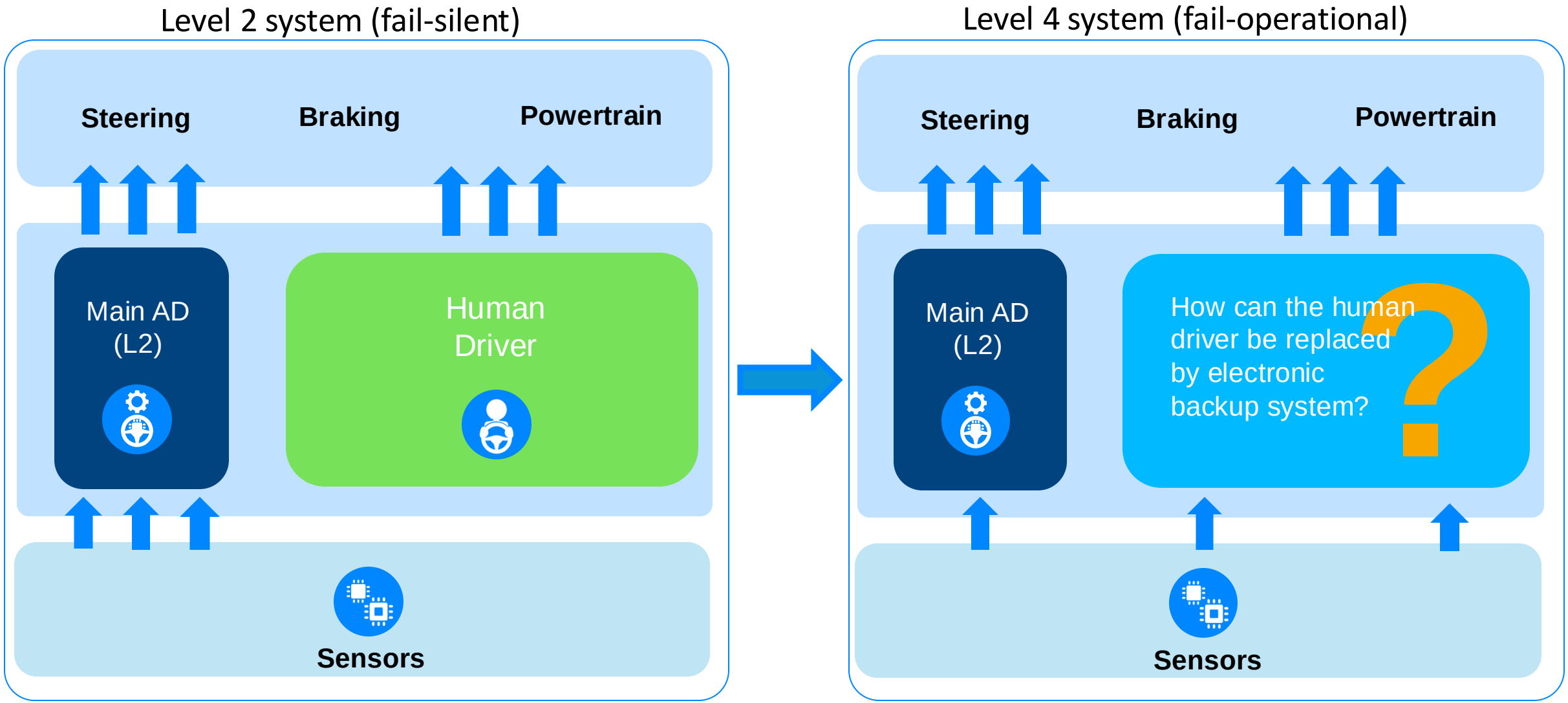
Change lanes and stop the car at safe position



Stop the car in lane (avoiding obstacles)

Level 5	System has to continue driving until planned travel end point

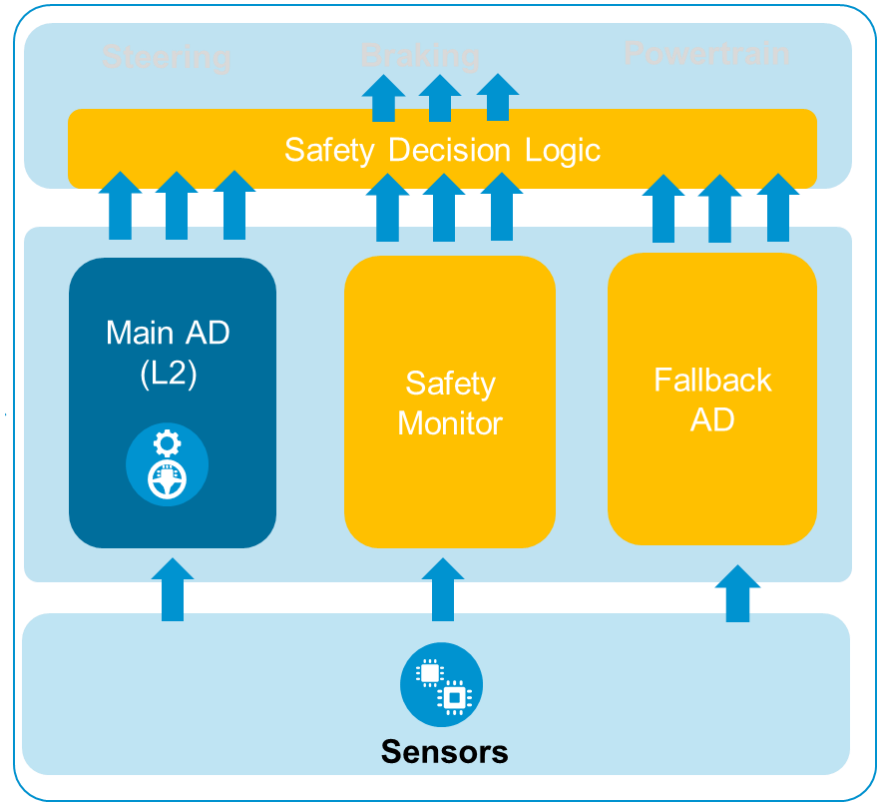
Moving from a Level 2 to a Level 3/4 system



Conceptual Architecture

Systematic partitioning to solve key „impossibilities“:

1. Impossible to **find all design faults** in a large and **complex monolithic software** system
2. Impossible to **avoid single event upsets** (e.g., bit flip) **in non-redundant hardware** during the life-time of an ultra-dependable system
3. Impossible to **establish the ultra-high dependability** of a large monolithic system **by testing and simulation**
4. Impossible to precisely **specify all edge cases** that can be encountered in driving situations
5. Impossible to source AD system modules with full ASIL D qualification: SOCs, OSs, BSW, … contain lower ASIL and QM elements



In order to realize automated driving at higher levels tens to hundreds of functions (applications) need to be integrated with each other to operate as a coordinated whole.

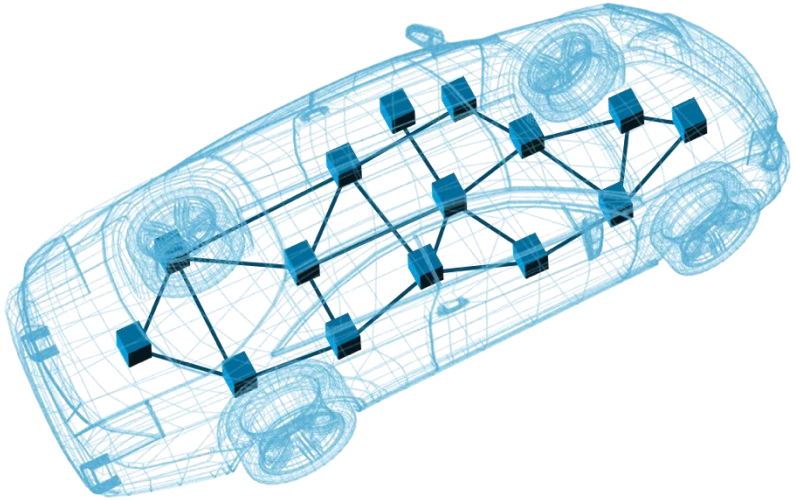
Modern ADAS/AD paradigm

Growing number of complex ADAS functions

- automated/assisted parking, lane changing, emergency brake assistance, autonomous driving.

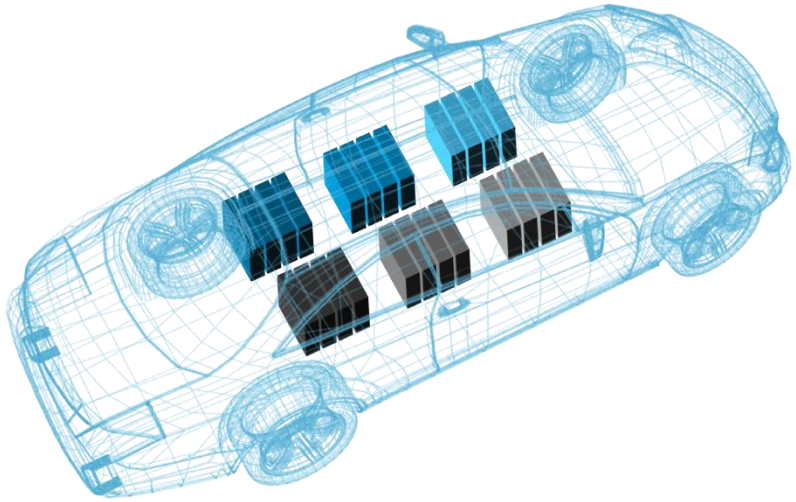
Fusion of multiple software functions into the same hardware platform

- reusability and portability
- integration of mixed-criticality functions with guaranteed temporal and spatial isolation



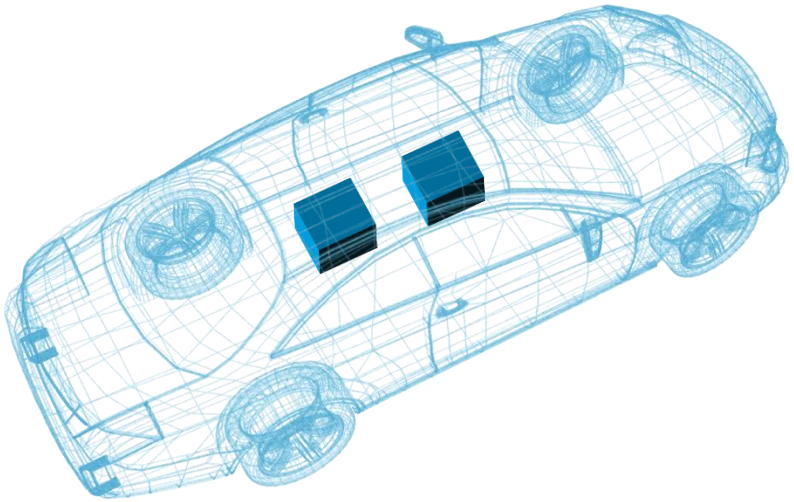
n ECU: n functions

DISTRIBUTED
E/E ARCHITECTURE



5-7 domains: n functions

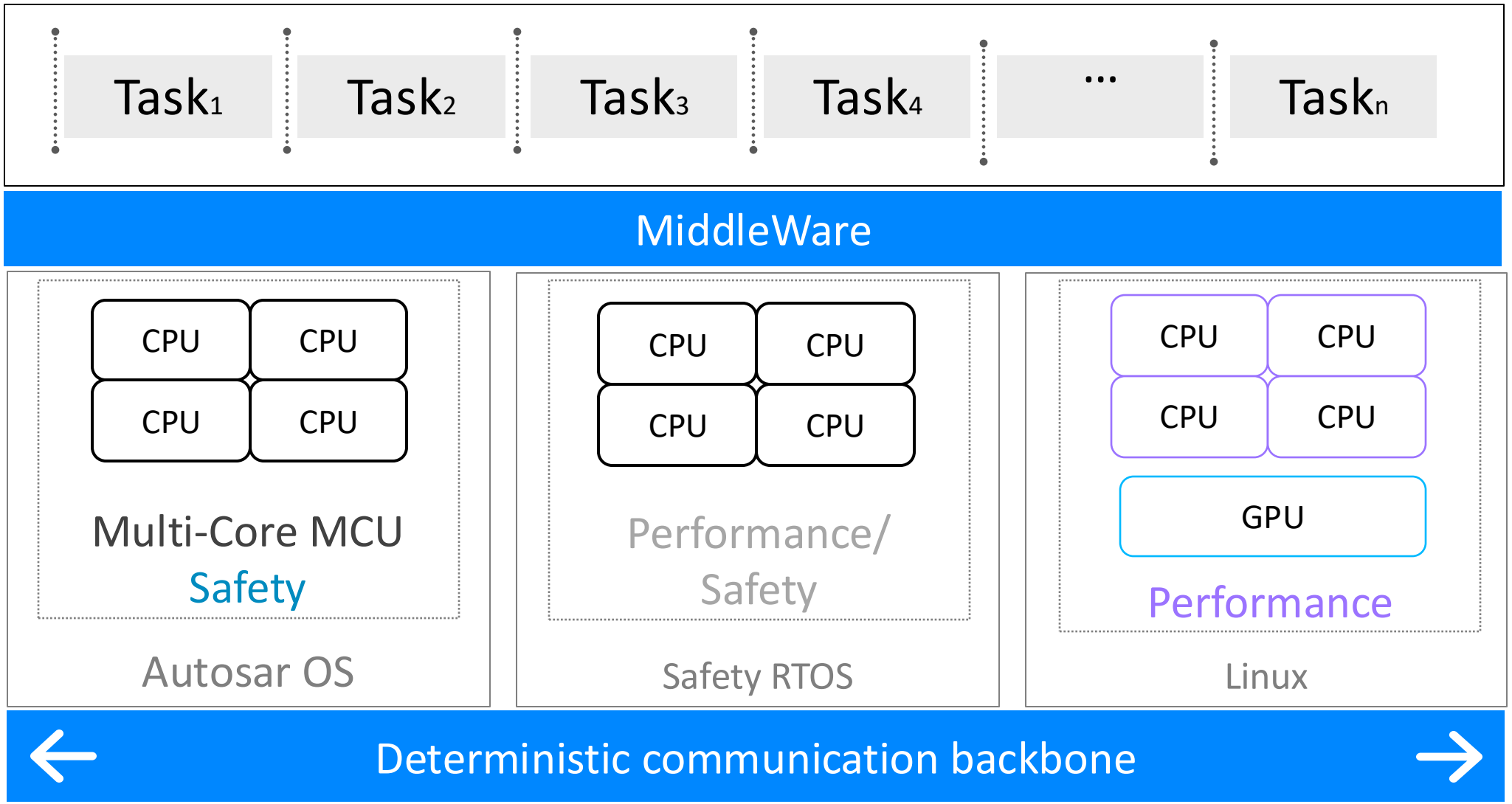
DOMAIN
E/E ARCHITECTURE



2 ECUs: n functions

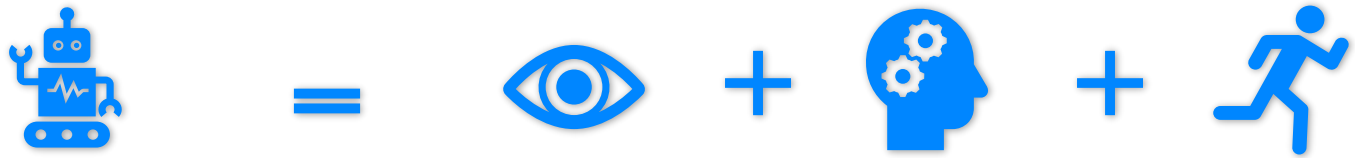
CENTRALISED
E/E ARCHITECTURE

ADAS/AD platform



ADAS/AD systems are real-time systems

AD systems (and real-time systems in general) are reactive systems.



Given an event, the system must produce a reaction or response within a specified time (deadline).

Periodic events occur at predictable (regular) time intervals
- deterministic – all relevant information is available at design time
- complex requirements - deadlines, end-to-end latency, jitter, …

Sporadic events occur with a certain maximum rate
- usually defined by minimum interarrival time

Aperiodic events occur at unpredictable times
- variance – not everything is completely known, especially arrival times
- limited predictability: only simple requirements can be guaranteed in the best case

Most real systems contain periodic, sporadic and aperiodic events

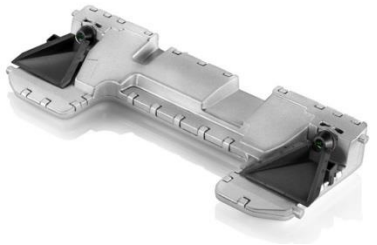
Sensing, computing, actuating

Sensor fusion + scene segmentation based on deep neural networks

Environment camera

Laser scanner

Stereo camera



Ultrasound sensors

Lidar

Radar

Front camera

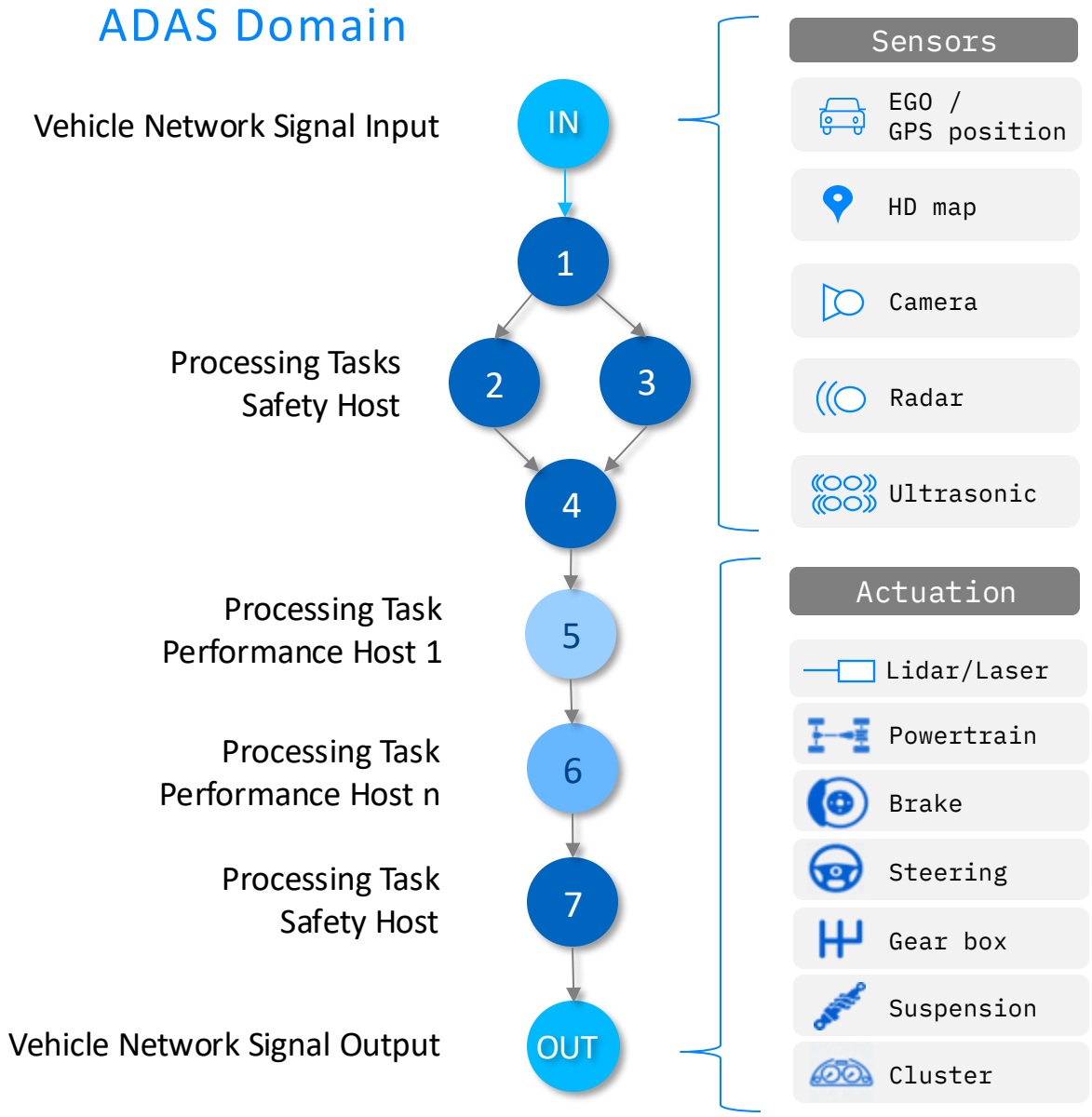
Cause-effect chains

Cause-effect chains are a characteristic of automotive software:

- provide additional timing and dependency requirements on the execution of tasks
- can span across multiple activation patterns
- include multiple tasks, even the same task multiple times
- have priorities and end-to-end latencies
- include communication latencies

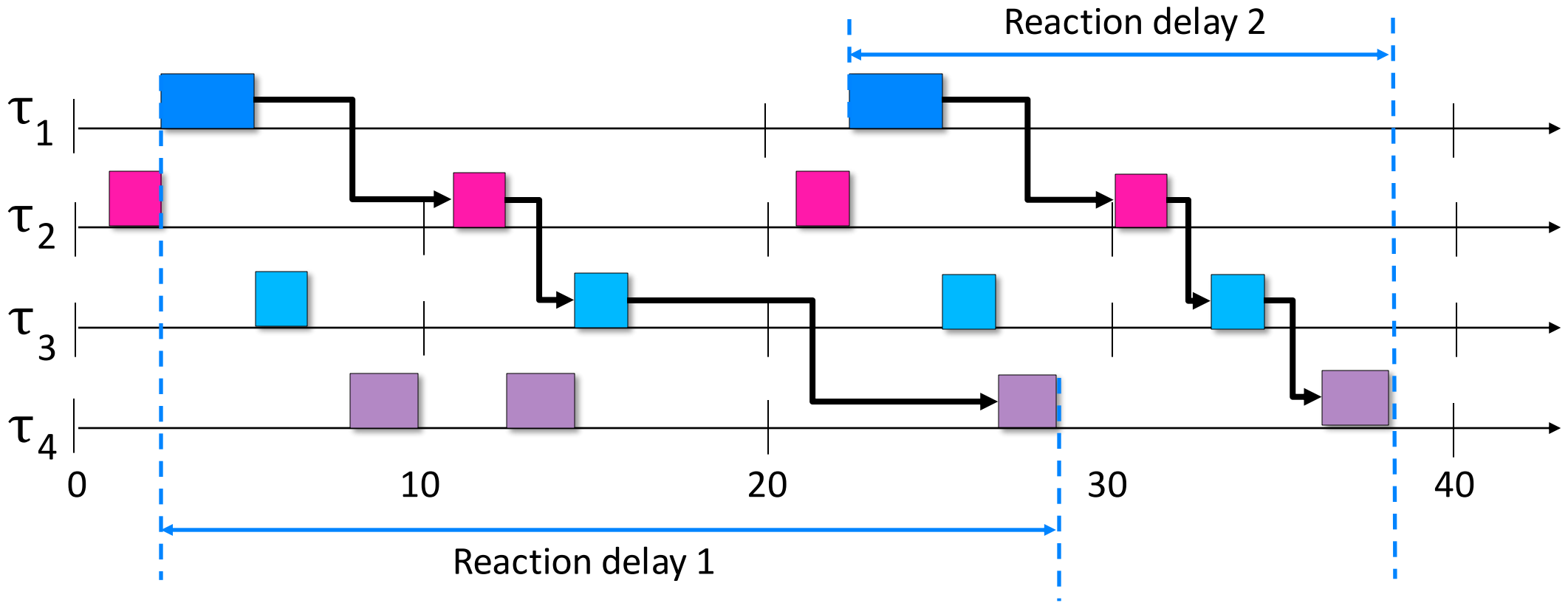
Semantics:

- Reaction delay
- Data age
- Order



Cause-effect chains: reaction delay

After each event, there must be a reaction within the maximum allowed time



Correct-by-design approach

Correct-by-design approach

Correct-by-testing

- Manual resource allocation and priority
- Extensive testing
- Empirically driven
- No guarantees for safe and reliable runtime execution in corner cases

No longer address AD central compute complexity

Correct-by-design

- ✓ Automates manual process
- ✓ Hardware resources are modeled
- ✓ Mathematically proven approach
- ✓ Addresses all corner cases
- ✓ Based on very advanced heuristics
- ✓ Scales with complexity

Future-proof and safe solution

"Design approaches not based on mathematically proven real time scheduling properties are prone to missing deadlines during unusual operational conditions"

UL4600 Standard for Safety for the Evaluation of Autonomous Products

Scheduling (dispatching) algorithms

- FCFS
- (Weighted) Fair queueing
- Priority scheduling
- (Weighted) Round-robin
- …

General-purpose computing

Main objectives: avoid starvation, responsiveness, fairness, QoS

- Rate monotonic
- Deadline monotonic
- Earliest deadline first
- Time-triggered
- …

Real-time/multimedia computing

Main objectives: meeting deadlines & other real-time constraints

Fixed-priority scheduling

- The static priorities of tasks that are in the ready queue determine the execution order of tasks
- Priorities are usually derived from temporal requirements (mostly period and deadlines)

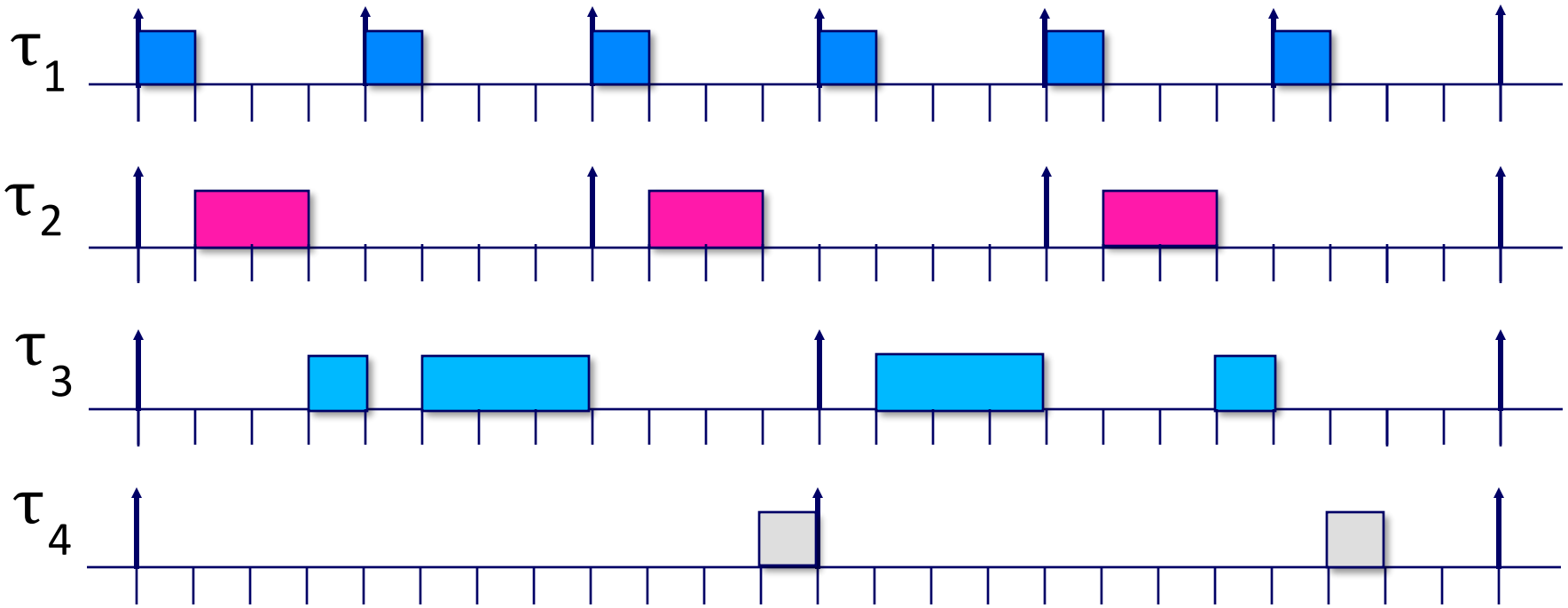
FP

$T_1(1,4)$

$T_2(2,8)$

$T_3(4,12)$

$T_4(1,12)$

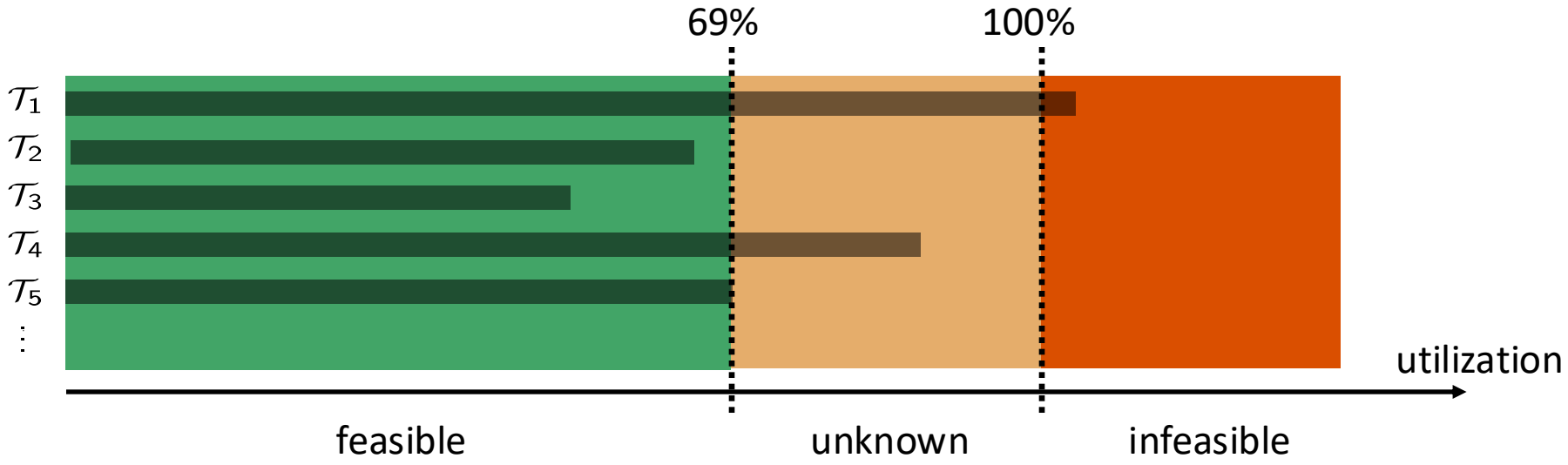


Utilization-based test

[Liu1973] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled under the RM algorithm if its total utilization U is at most

$$\sum_{i=0}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \qquad n \to \infty \qquad \lim_{n\to\infty} n(2^{\frac{1}{n}} - 1) = ln(2) \approx 0.69$$

In practice the bound is on average 88%. [Davis2016]
When the periods are harmonic, the bound is 100%. [Mok1991]

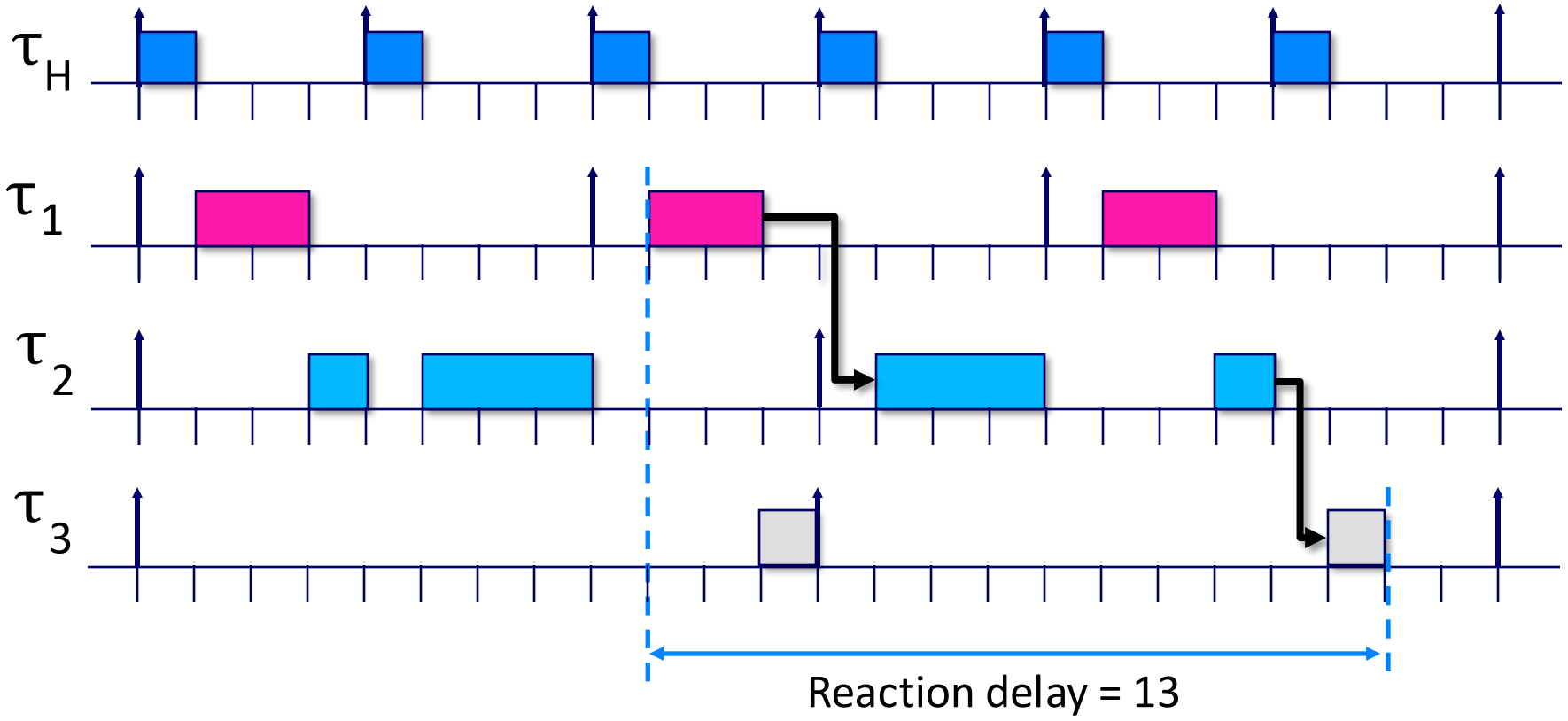


Fixed-priority scheduling wrt. dependency chains

High-priority task: $T_H(1,4)$

Chain: $T_1(2,8)$, $T_2(4,12)$, $T_3(2,12)$

RM

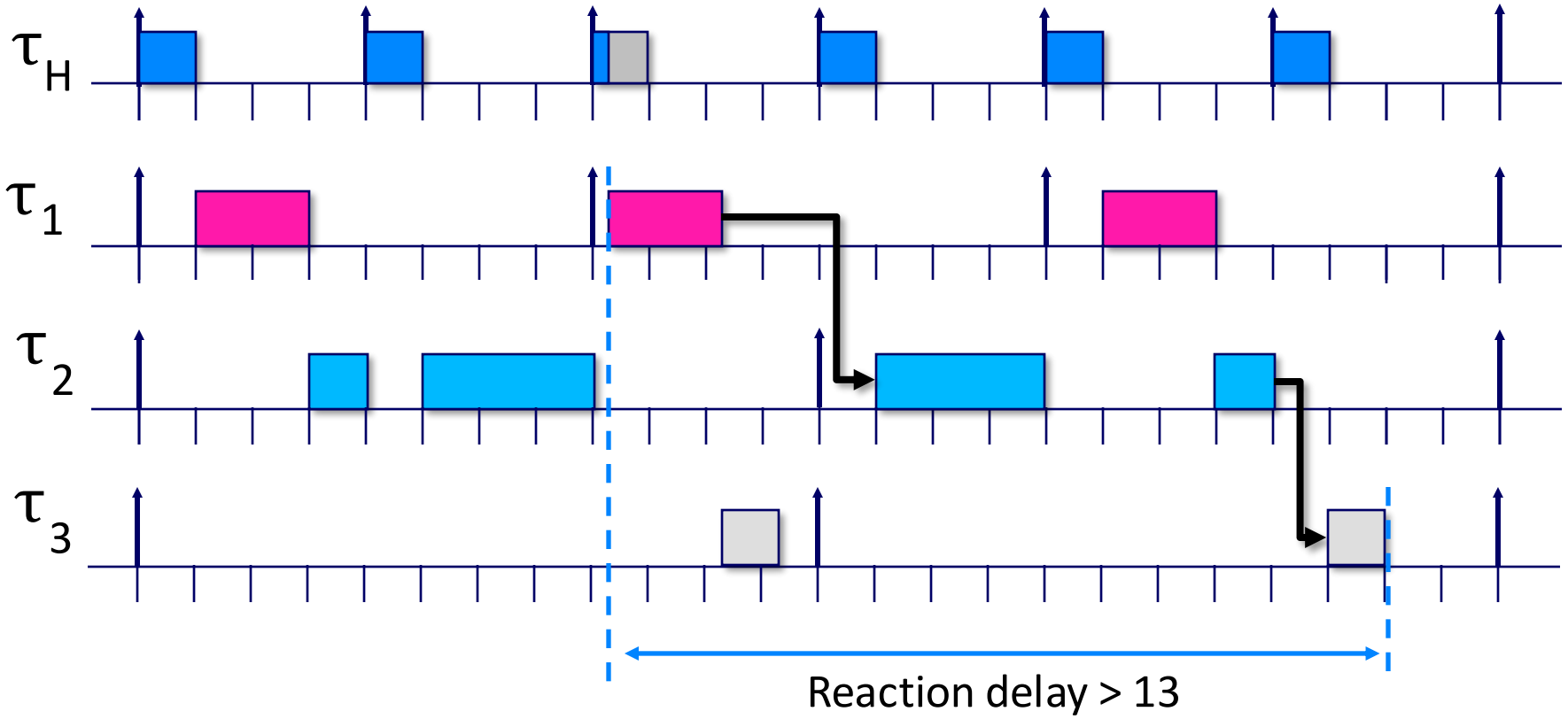


Fixed-priority scheduling wrt. dependency chains

High-priority task: $T_H(1,4)$

Chain: $T_1(2,8)$, $T_2(4,12)$, $T_3(2,12)$

RM



Fixed-priority scheduling

Generally, Fixed-Priority Scheduling (FPS) have the following properties:

- **Static priority**: cannot change at runtime, priorities determine the execution order of tasks
- **Responsiveness**: high-priority tasks run whenever they become active
- **Reduced latency at runtime**: if a task finishes earlier than the WCET assumption then the next lower priority task runs immediately, thereby reducing its latency
- **Scalability**: Easy algorithm that scales well with the number of tasks, proven in use
- **Non-harmonic periods**: non-harmonic periods do not influence the applicability
- **Overload scenarios**: only lower priority tasks are impacted
- **Availability**: FPS are implemented in most Operating Systems

Downside of FPS:

- **Jitter**: tasks have a high run-time variance, c.f. 3.2 in [Locke1992]
- **Starvation**: tasks can starve if the highest priority tasks misbehaves and always stays in the ready queue
- **Dependencies**: complex task dependencies and constraints cannot be easily supported, c.f. 4.1 [Xu2000]
- **Utilization Bound**: in general, the utilization bound for FPS is lower than EDF and TT, c.f. 4.2 [Xu2000]
- **Determinism**: the run-time variance of task execution is high leading to difficult analysis, c.f 4.3 [Xu2000]
- **Compositionality**: adding/modifying tasks leads to changes in lower-priority tasks, c.f 4.4 [Xu2000]

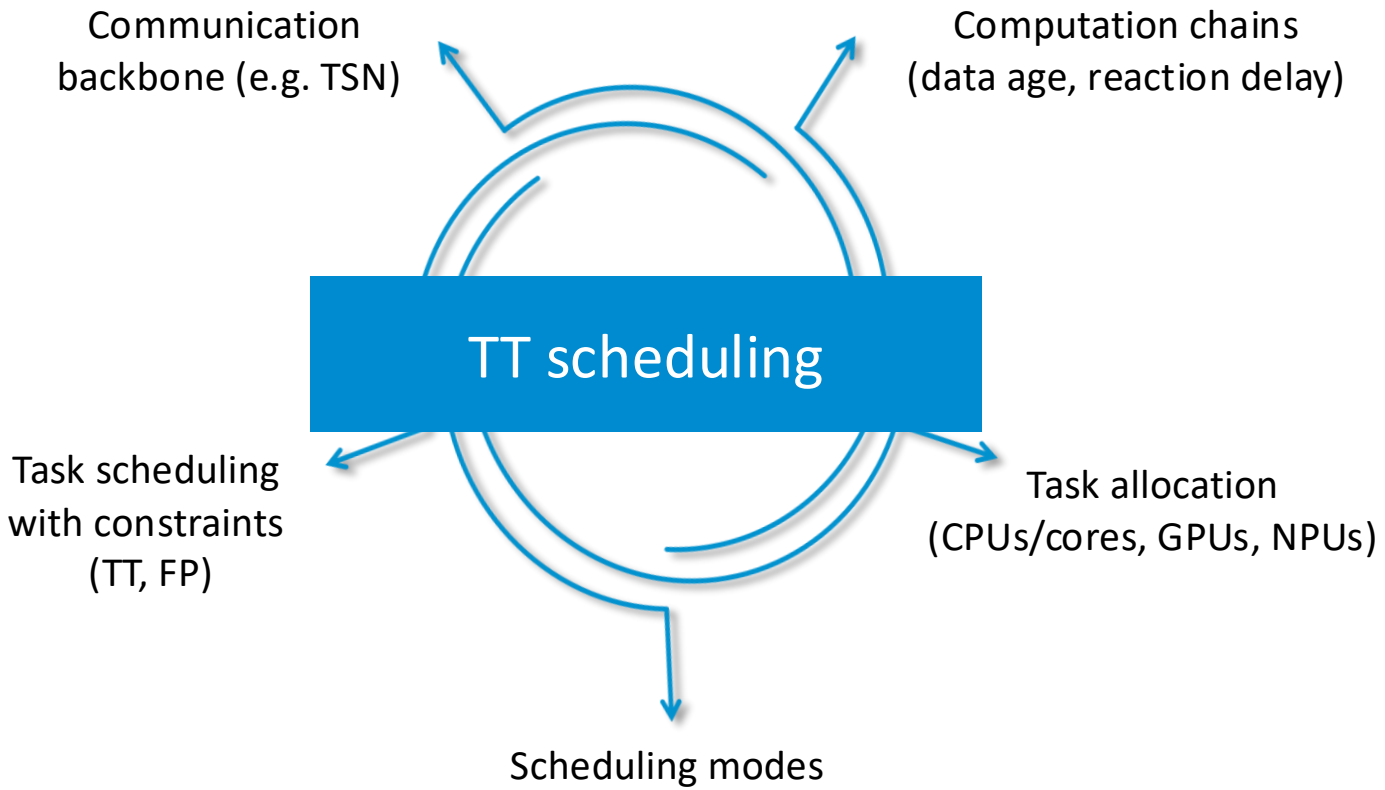
For a more in-depth comparison between FPS and TT please see [Xu2000, Locke1992]

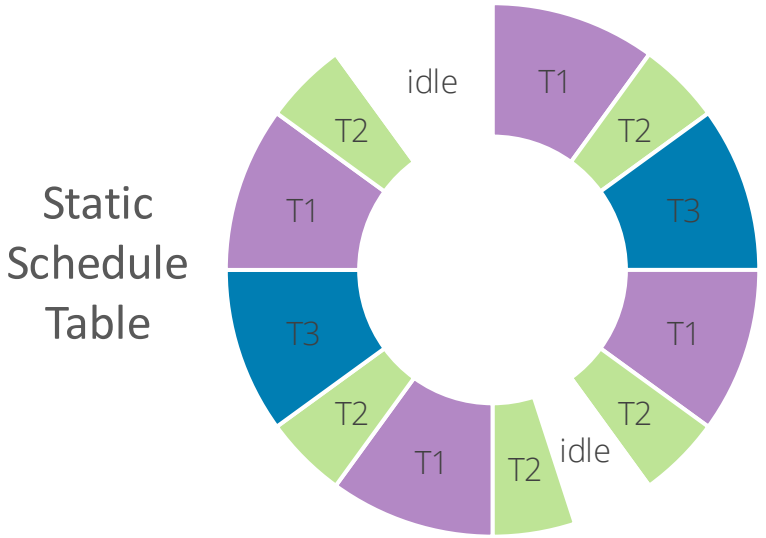
Time-triggered vs. event-triggered

Event-based architectures have poor behavior characteristics when critical situations occur simultaneously.

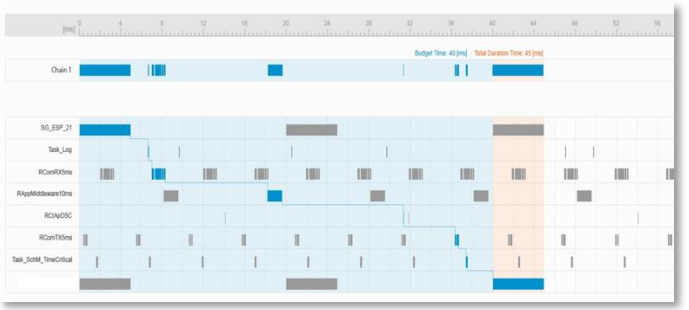
Time-triggered task scheduling

- Tasks are executed based on a **static schedule table** that has been computed at design time (offline)
- All scheduling decisions are taken at **compile time**, WCET needs to be known.
- The complete scheduling timeline for TT tasks is **fixed**.
- Simple (e.g., deadlines) and complex (e.g., chains) **constraints** are satisfied by the **schedule creation**.
- One solution is sufficient and any solution is a sufficient schedulability test.





TT scheduling



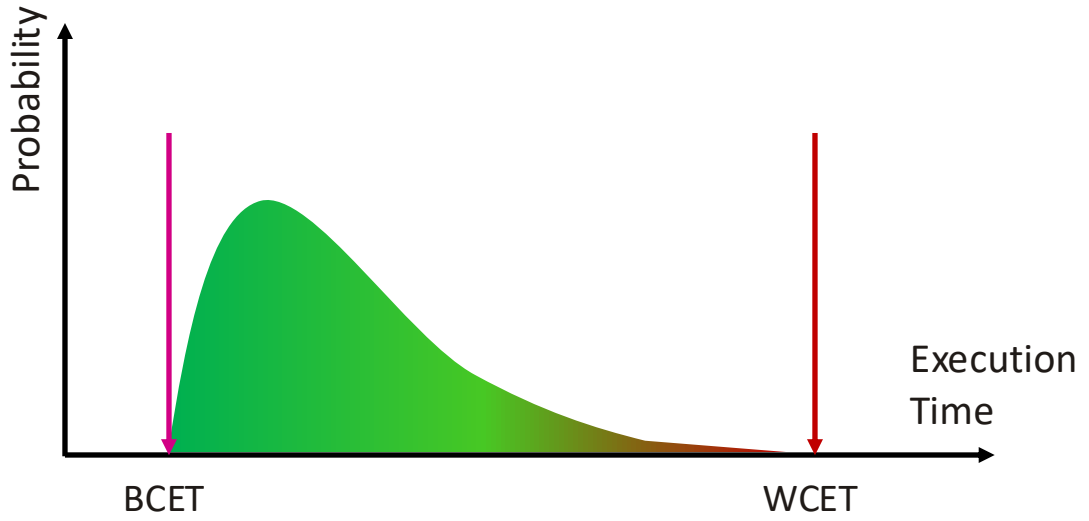
Time-Triggered provides correct-by-design real-time guarantees:

- **Complex timing requirements**: cause-effect chains, different types of jitter
- **Temporal isolation**: no starvation possible, temporal isolation between all tasks
- **Determinism/Predictability**: increased stability and testability, no unwanted run-time effects, many system properties become predictable, e.g., locks, task pre-emption
- **Re-simulation:** Equivalent behaviour between cloud and embedded target
- **Synchronization to communication**: stable real-time behaviour of cause-effect chains
- **Compositionality/Integration**: adding or modifying tasks can be done incrementally, system integration of SWCs from different suppliers is easier
- **Schedulability**: more correct configurations can be realized
- "**What you see is what you get**" execution
- All critical execution and communication is deterministic and easily **traceable**.
- **Coordinated** and efficient task **execution** is enabled even beyond boundaries of single SoCs.
- Safety **certification** is simplified.

Stability to dynamic changes at runtime

WCET is used for generating schedules and testing schedule correctness

At runtime, tasks execute for less than the WCET



Example:

BCET = 1ms

WCET = 2ms

Microtick = 1us

1000 possible execution patterns

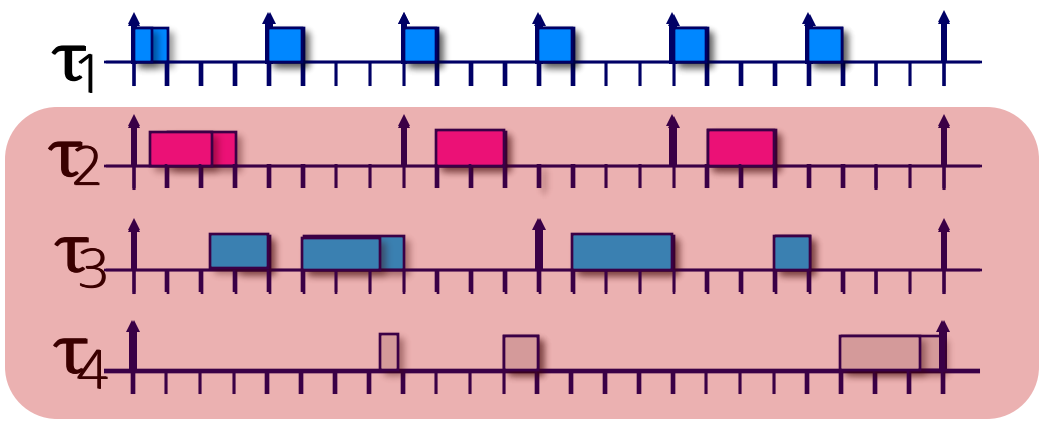
What happens if one task executes for less than the computed WCET?

TT vs. FP/EDF

What happens if at some point in time, one task executes for less than the computed WCET?

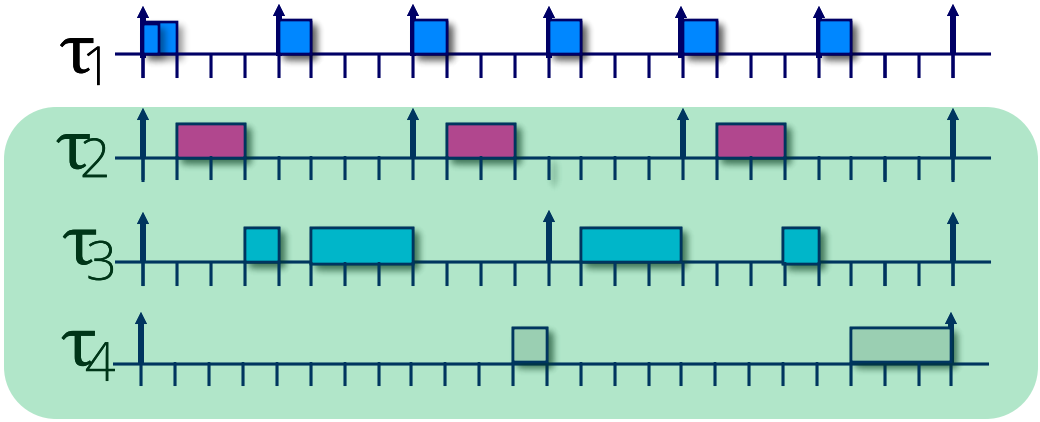
Non-TT (FP/EDF)

- changes the execution of all other tasks
- state change for the entire system



Time-Triggered

- execution of all other tasks remains the same
- no state change



State space explosion

#tasks that run for less than the WCET

Non-TT (FP/EDF)

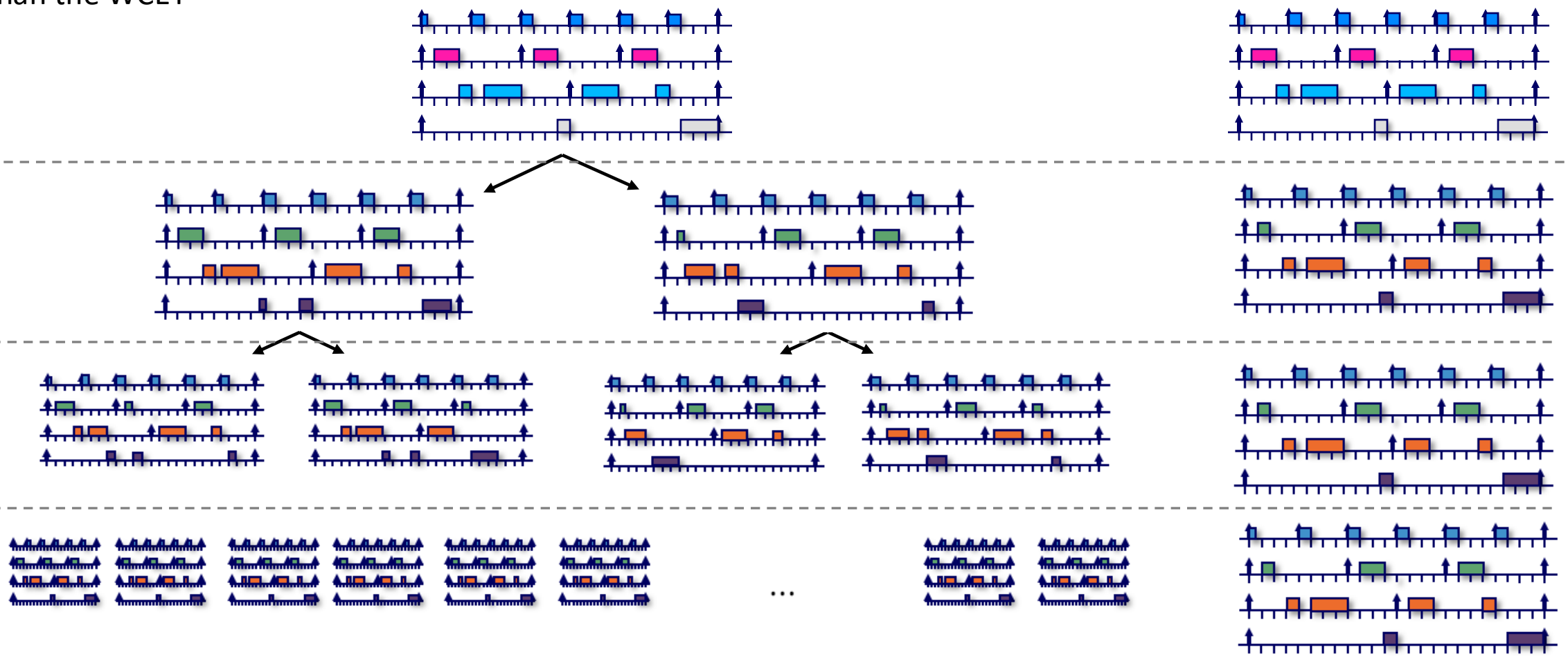
TT

1

2

3

4

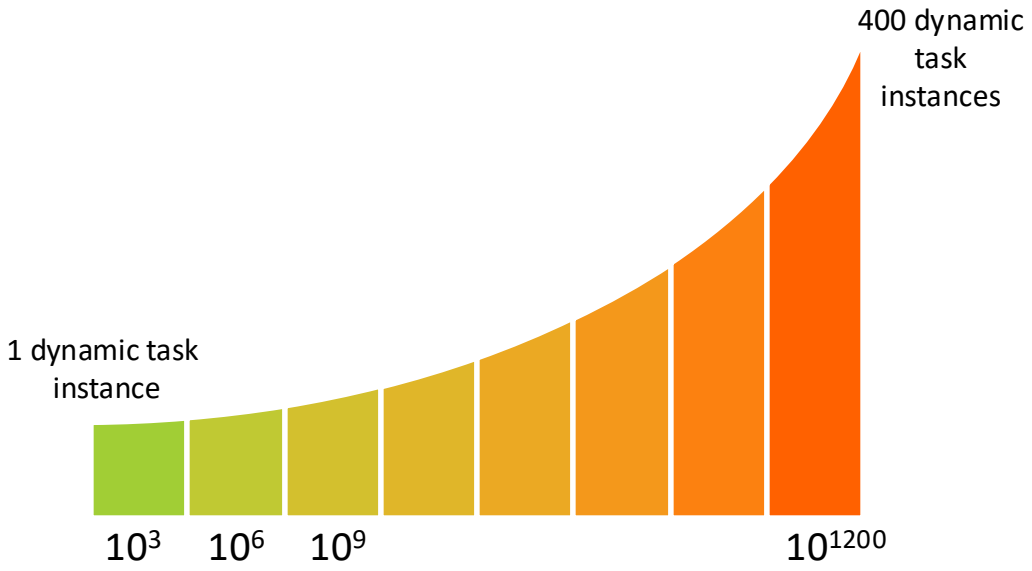


State space explosion

How many system states do I have to potentially check/look at to give design-time guarantees?

Non-time-triggered (FP/EDF/etc.):

- changes the execution of all other tasks
- state change for the entire system



Number of possible assignments of tasks to cores

Number of priority orderings

Number of system states

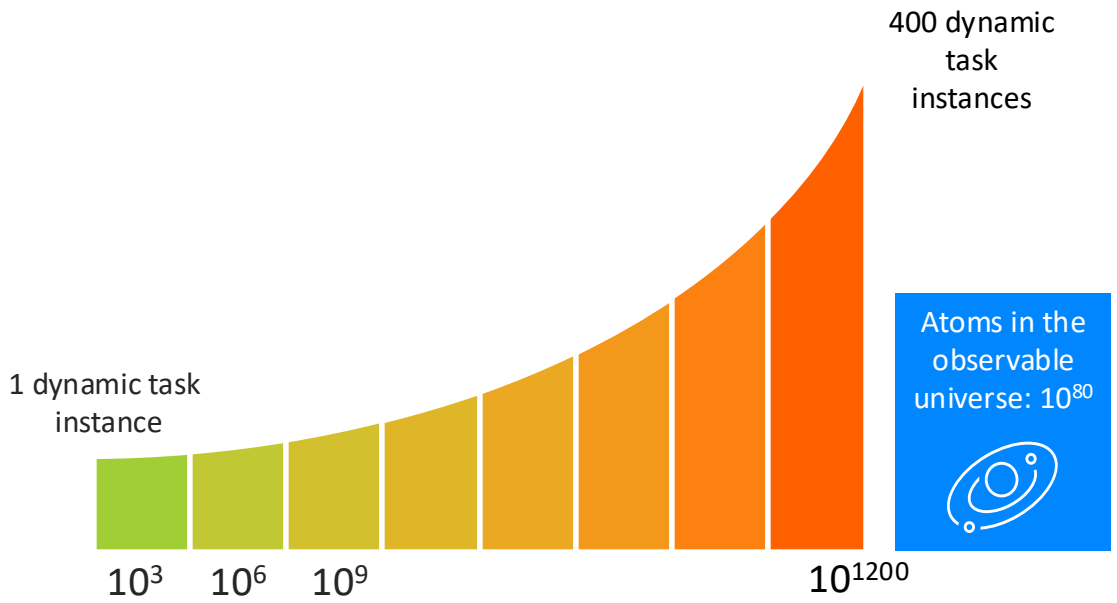
$52^{400} \times 400! \times 10^{1200} = 10^{2755}$

State space explosion

How many system states do I have to potentially test/look at?

Non-time-triggered (FP/EDF/etc.):

- changes the execution of all other tasks
- state change for the entire system



Time-triggered:

- execution of all other tasks remains the same
- no state change

BUT: Finding the static schedule table is still a very complex problem to solve

Decidable vs. Computable

- **Fundamental question**: David Hilbert asked: is mathematics decidable, i.e., is there a way (algorithm) to prove a statement given a set of axioms and rules?
- **Alan Turing** answered the question and, in the process, invented the foundation of all modern computers: the universal Turing machine

undecidable

Problems without an algorithm

decidable

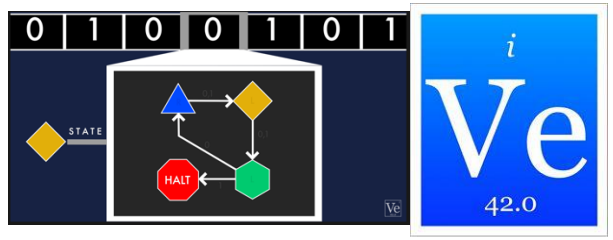
Problems that have an algorithm

tractable

Problems with an efficient algorithm

intractable

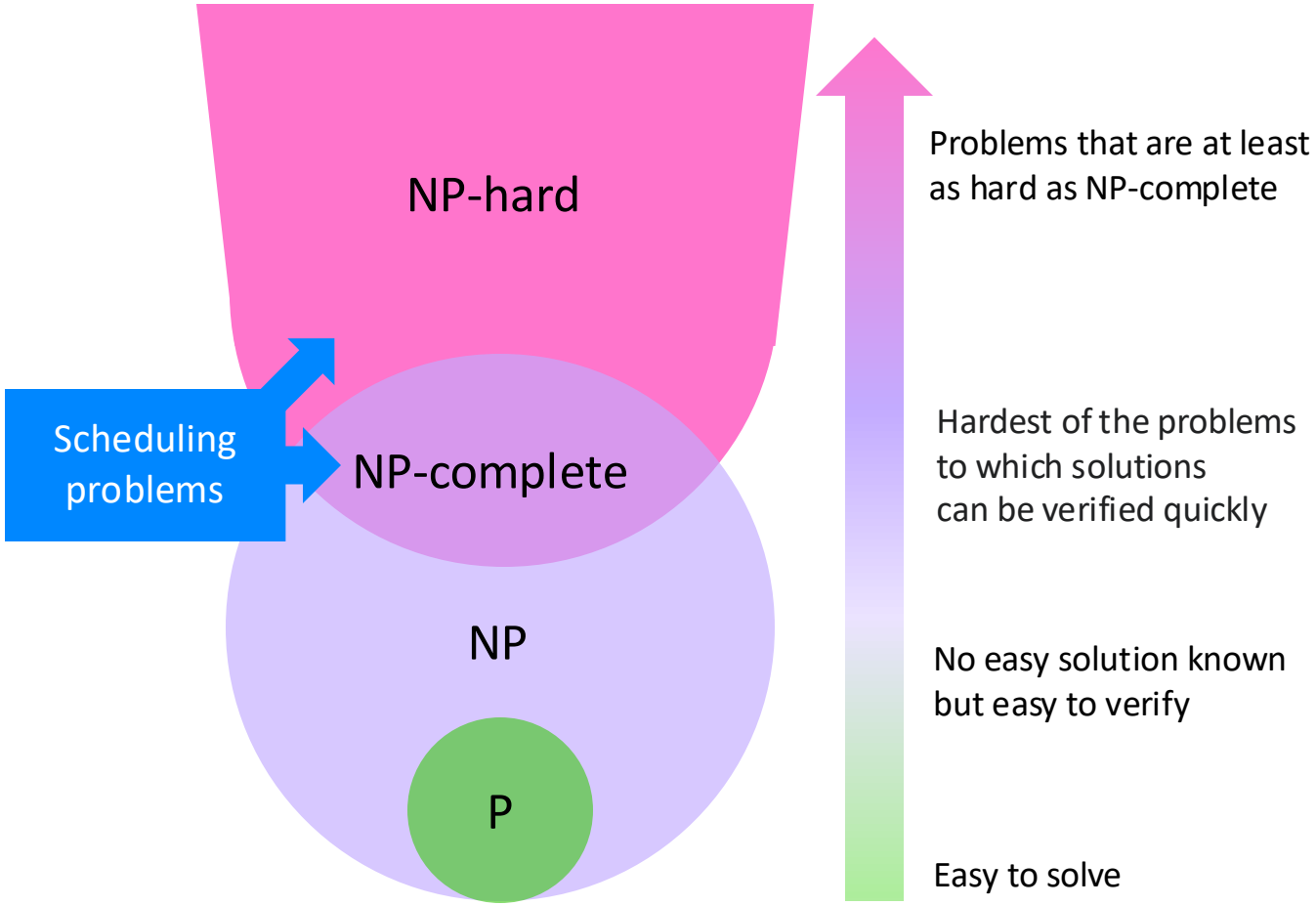
Problems without an efficient algorithm

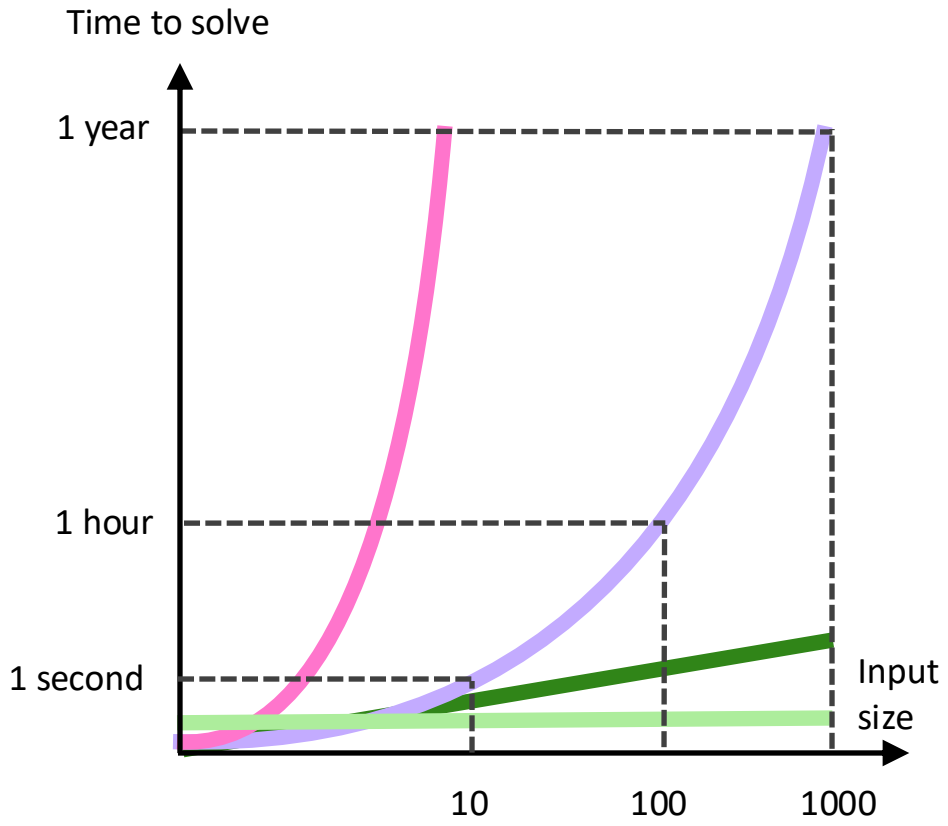


https://www.youtube.com/watch?v=HeQX2HjkcNo

The time-triggered scheduling/configuration problem is decidable.
The main question is, is it tractable?

Complexity classes

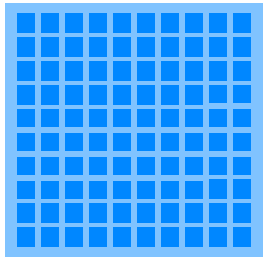




⚠ Complexity is not always equal to algorithm efficiency: compare 1.00001^n vs n^{50}

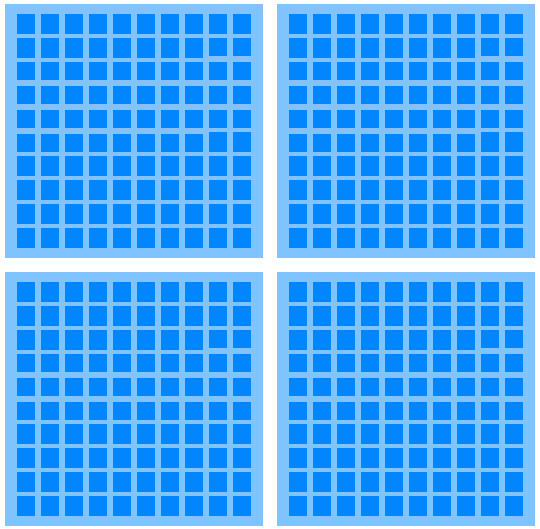
Complexity in the real-world – ADAS L2 example

Software demands



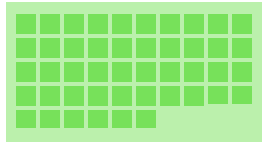
100 Applications

400 Tasks

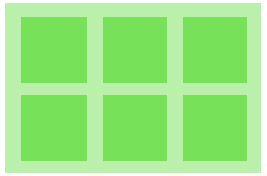


Computation unit resources

46 CPU cores



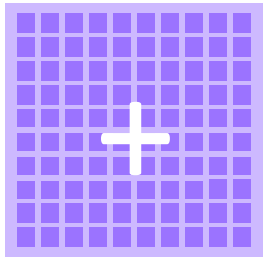
6 Hardware accelerators



2 TSN switches

1024 streams

System constraints



100+
time, latency, precedence constraints etc.

Solution space

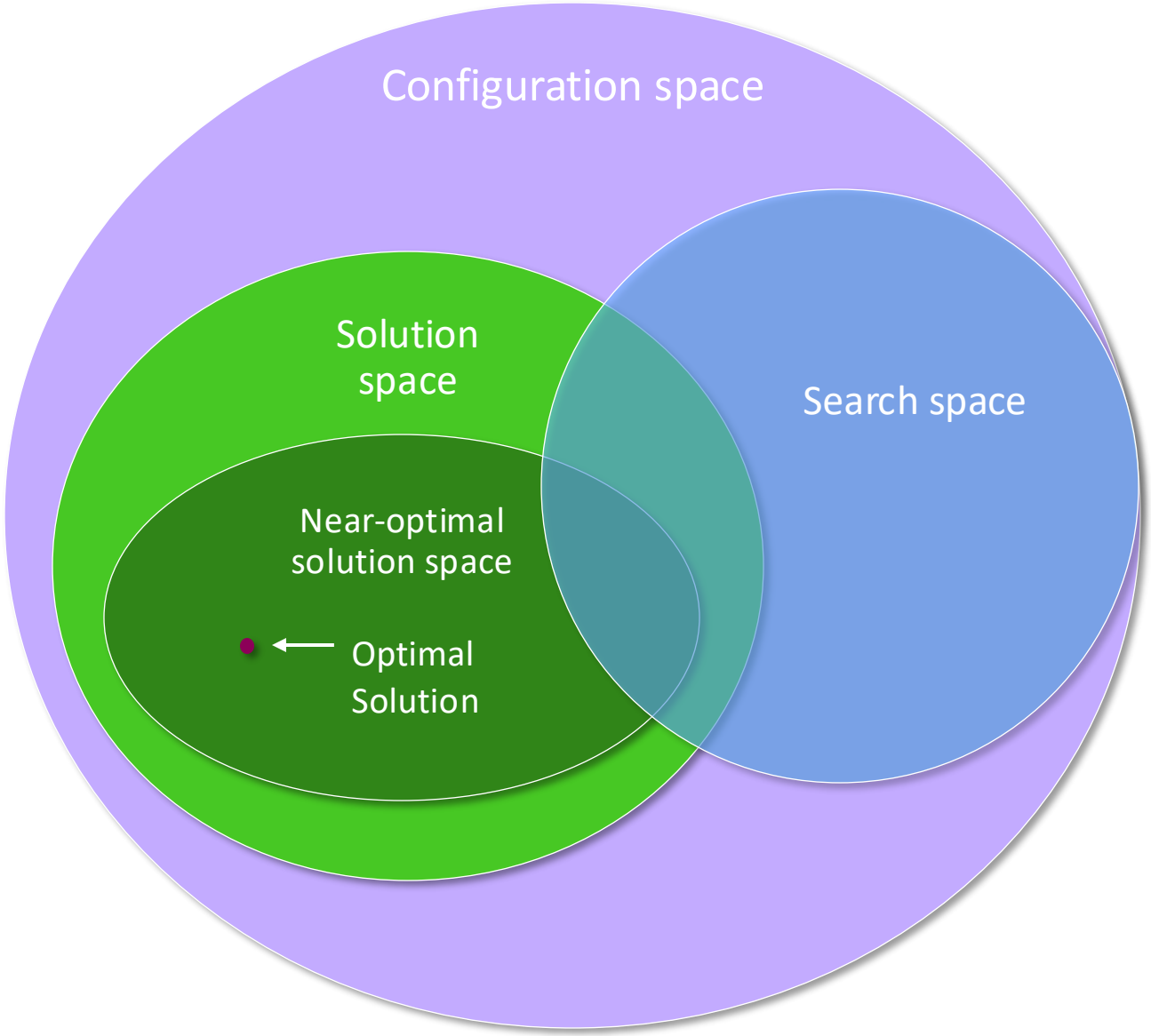
Theoretical vs Valid configurations

10^{5000} possible configurations

Atoms in the universe 10^{80}

10^{5} correct configurations

Optimality



Optimal algorithm:

- Looks at entire configuration space
- Finds a solution if there is one (might take a long time)
- If no solution is found, no other algorithm can find a solution
- Typically, NP-complete

Heuristic algorithm:

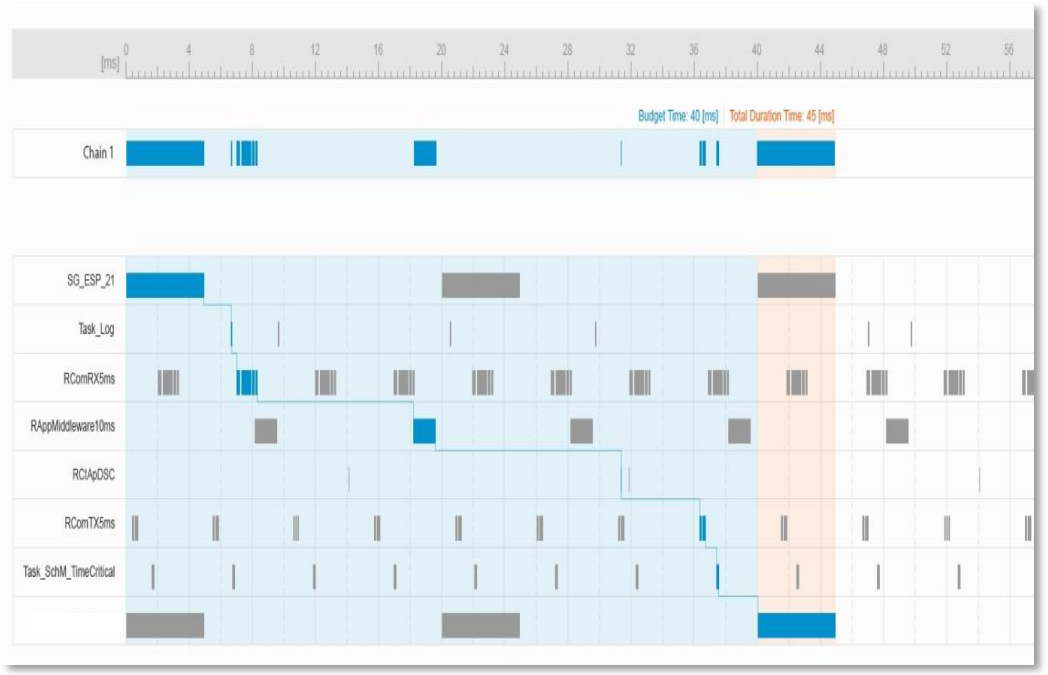
- Looks at a fraction of the configuration space (search space)
- Trade-off performance vs optimality
- Not guaranteed to find a solution
- Designed to be fast
- Tries to find near-optimal solution

Time-triggered and event-triggered integration

Downsides of Time-triggered

- **Inflexible**: no (big) adaptations at runtime, task execution is fixed
  - event-triggered tasks (sporadic activation)
- **Overprovisioning**: Slots must be reserved for the worst-case
  - WCET analysis is hard and very pessimistic!
- **Precomputed**: task properties must be known at design time
  - this is a development process issue!
- **System limitations**: macrotick, hyperperiod

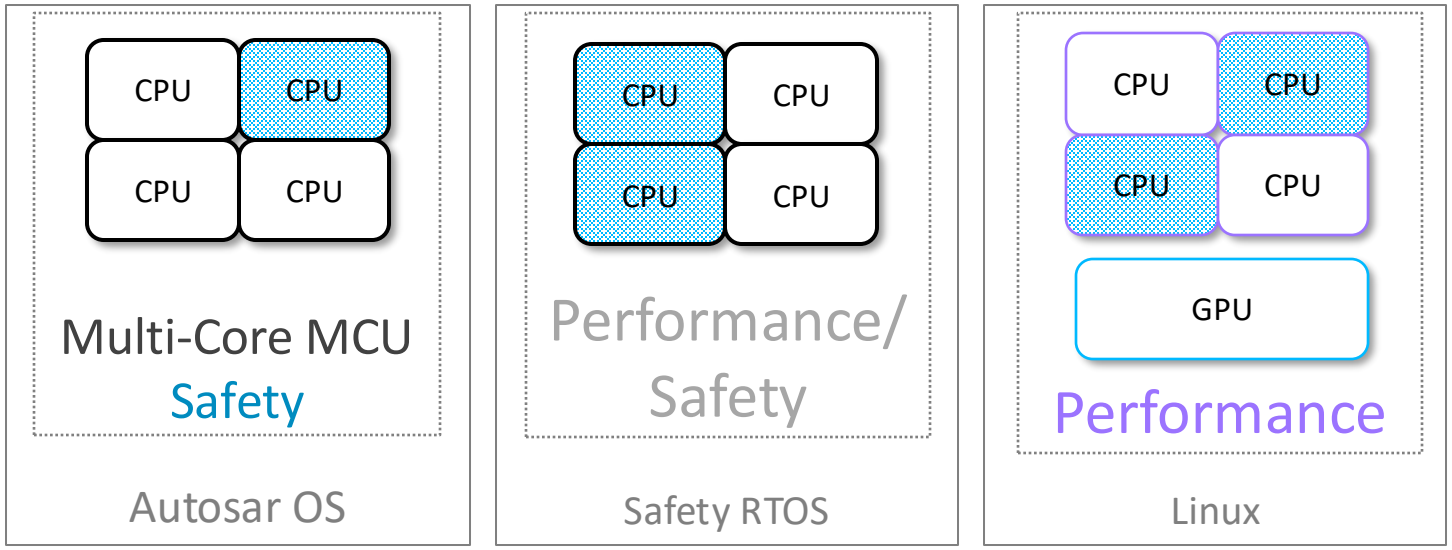
- **Event Driven tasks:**
  - sporadic activation
  - WCET known
  - constrained or arbitrary deadlines



As much flexibility as possible, as much time-triggered as necessary

Modern safety-critical systems (e.g., ADAS)

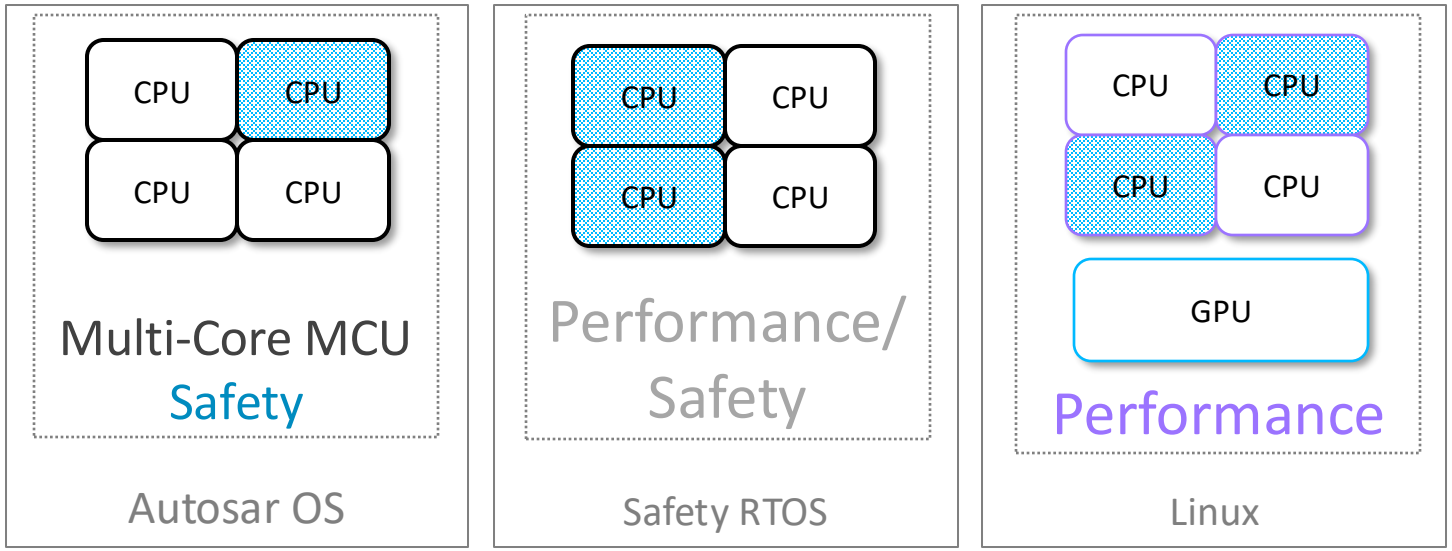
- **Heterogeneous** multi-core multi-SoC platforms featuring a variety of CPUs and GPUs
- Time-Triggered (**TT**) tasks are **periodic** and **statically scheduled**
- Event-triggered (**ET**) are **sporadic** but also require **timing guarantees**
- **Partitioned** scheduling approach without runtime migration of tasks



Modern safety-critical systems benefit most from combining TT and ET, allowing a system to be flexible enough to respond to sporadic events when needed, i.e., **best of both worlds**

Dedicated core

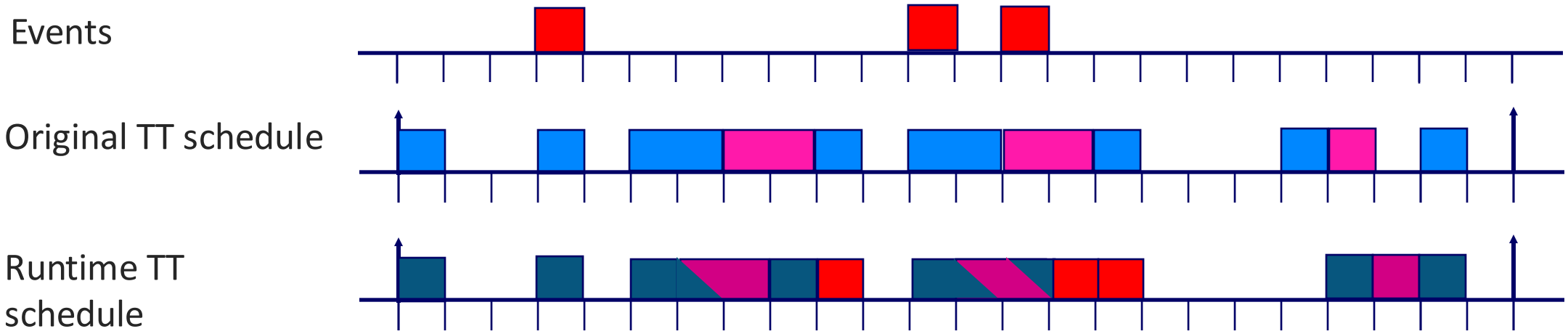
- Dedicate cores to process event-based tasks
- Can be done either using OS scheduler (usually fixed-priority) or via a custom scheduler if needed
- Can also be almost dedicated core : have a few TT slots if needed
- Best for long (non-preemptable) chains that are data-driven and triggered by events



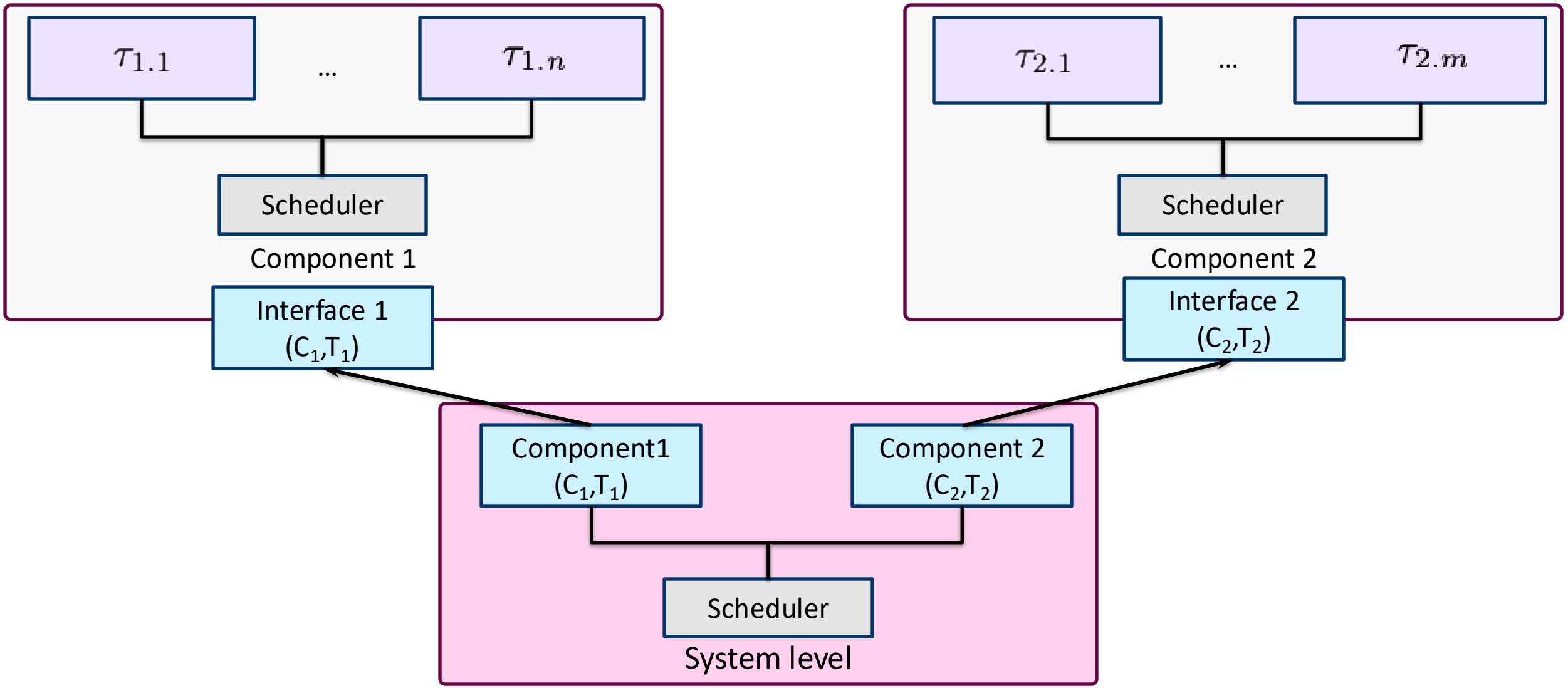
Slack stealing

Idea: Use available slack to process event-based tasks as soon as possible

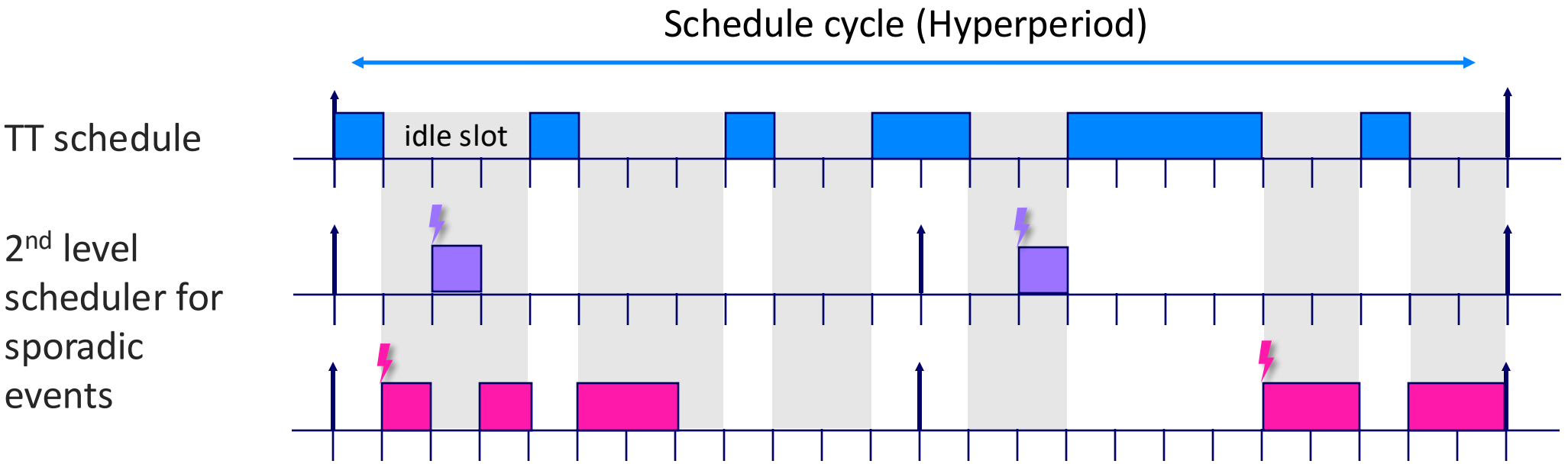
- Keep track of the system slack (i.e. when tasks finish early), and detect when & if this slack can be used
- Does not modify the TT schedule
- No real guarantees can be given for events
- Ideal for: best-effort event task handling



Hierarchical scheduling framework



TT and ET integration



- Time-triggered schedule ensures timing behavior of TT tasks and is calculated offline
- Second-level fixed-priority scheduler that services event-triggered tasks in the idle slots at runtime

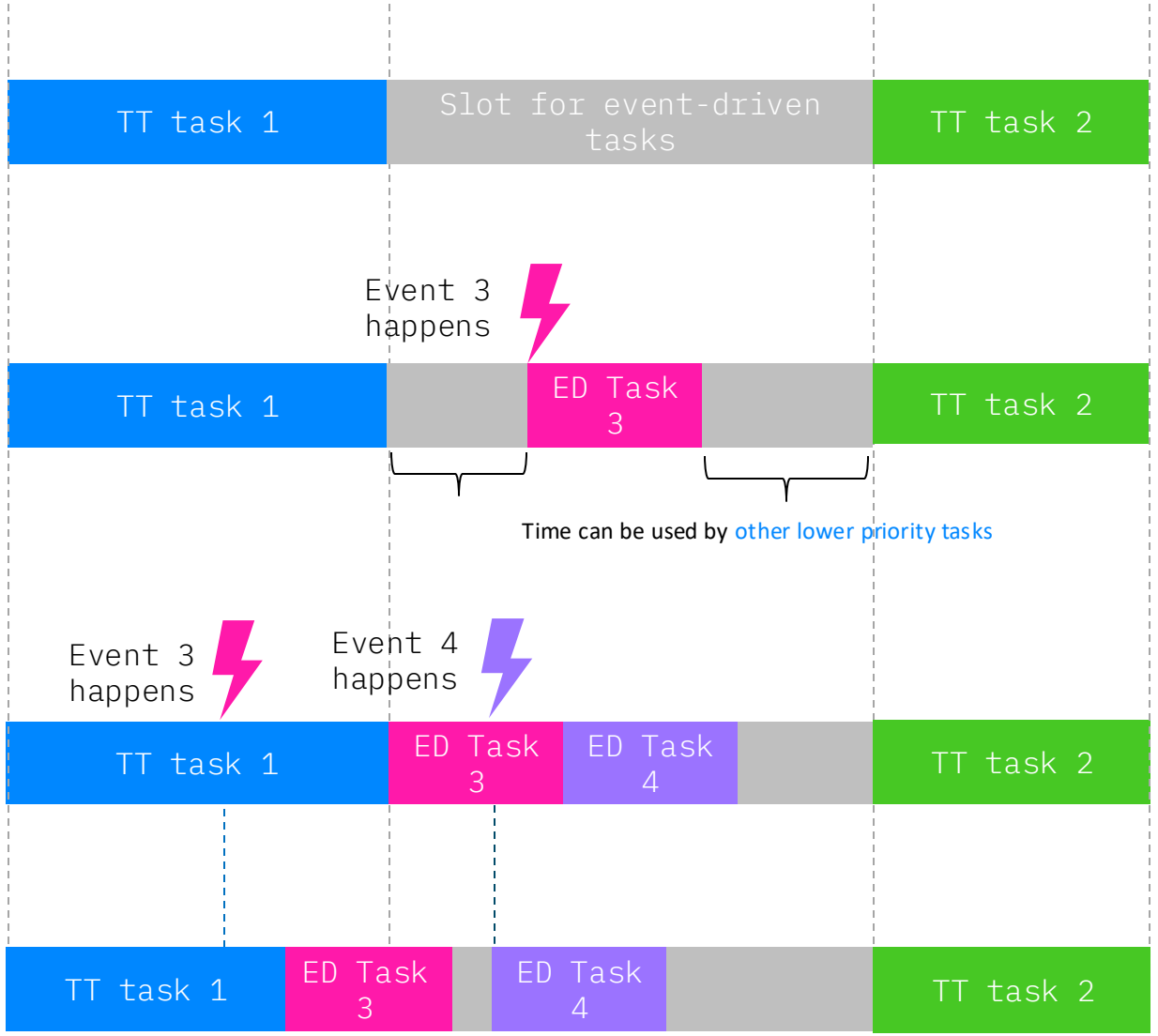
Example Use Cases

Generated schedule table

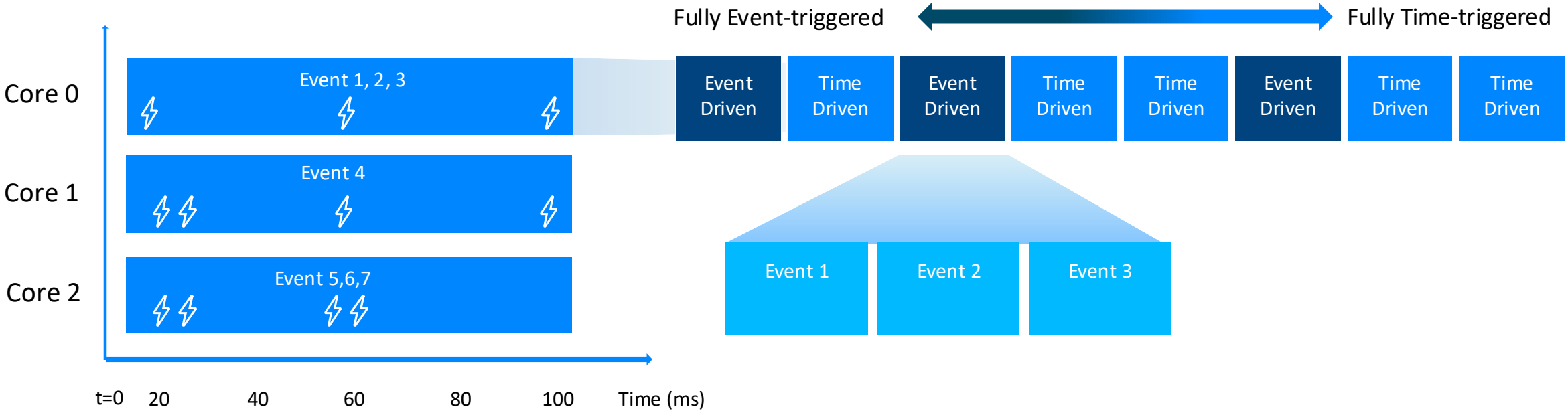
An event arrives during the slot where event-driven tasks can be executed

An event arrives during execution of a TT task

The event-driven task **will start in the TT slot** if the TT task finishes earlier !



TT and ET task allocation and interaction



- The system is partitioned, i.e., TT and ET tasks need to be allocated to cores at design-time
- The TT and ET task allocation problem is crucial to increase schedulability of the system
- The problem is NP-complete, i.e., optimal algorithms are infeasible in the general case

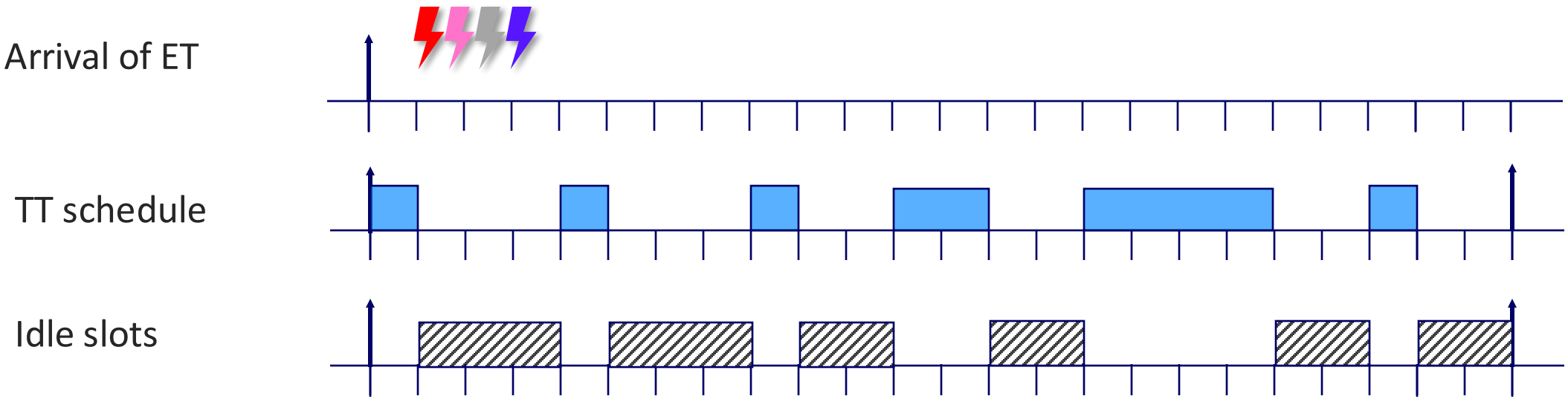
Problem dimension 1: How to place idle slots such that both TT and ET tasks respect their deadlines?

Problem dimension 2: Find a task to core allocation such that problem dimension 1 has a solution (optimization).

How can we check ET task schedulability?

Simulate Fixed-Priority and see if deadlines are met

- We need to do this for every potential correct TT schedule candidate (there are a lot of them)
- …and for every priority ordering (remember the combinatorial explosion of n!)
- …and for every arrival pattern of events (also leads to combinatorial explosion)



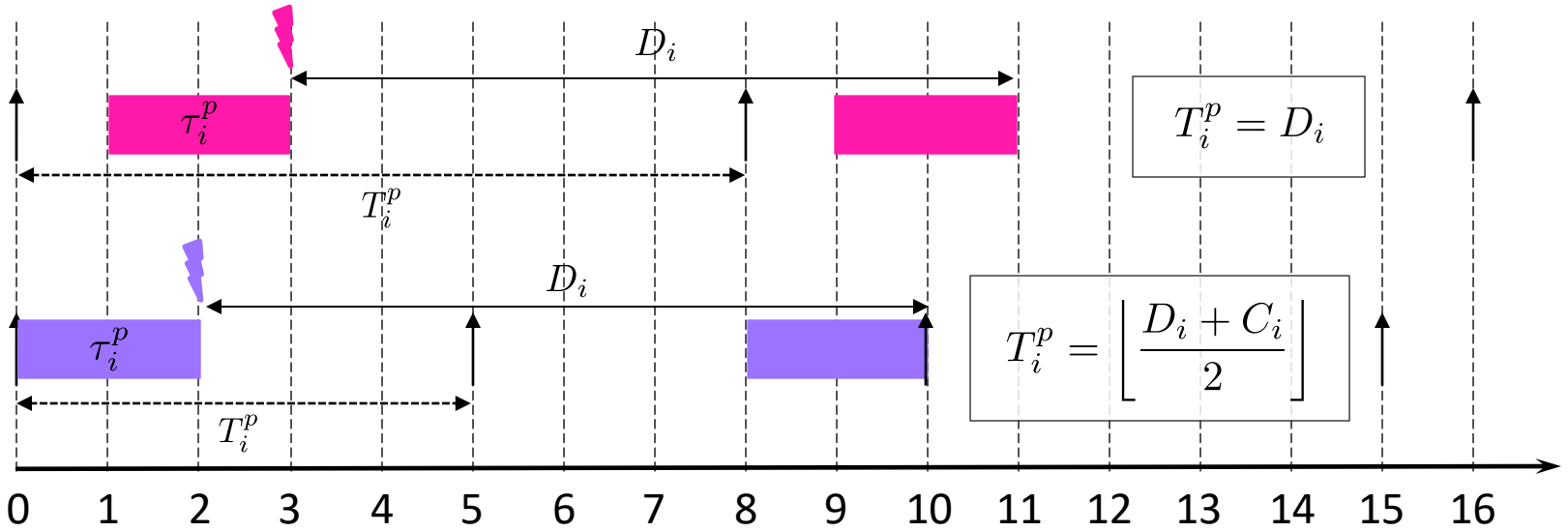
Highly infeasible!

Oversampling – SPoll

Simple polling: Use one periodic polling task per event – oversampling
How long is the sampling period?

a) Strictly periodic

b) Non-strictly-periodic



⚠ Can be quite inefficient!

Think of an event with C = 2 ms, D = 20 ms, and T = 100ms. (2% utilization)
We have to reserve a slot every 9 ms, consuming 22,2 % of CPU bandwidth.

Solution using response-time analysis

Holistic scheduling [Pop2003]

1. Generate TT schedule such that TT tasks are fulfilled.
2. Check schedulability over resulting idle slots using the response-time method over partitioned resources for every event-based task.
3. Recompute TT schedule if ET tasks not feasible

Slot-shifting [Isovic2009] is a similar method with a different schedulability test that does not assume FP scheduling.

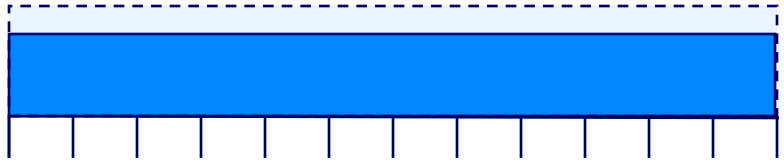
Can be very time-consuming!
No guidance on how to place idle slots.

Can we do better?

Periodic resource abstraction

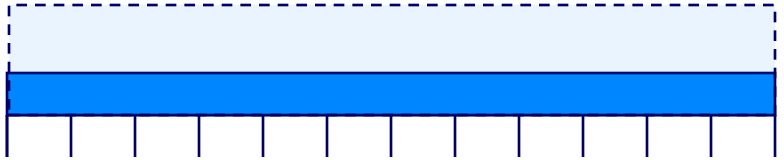
Dedicated resource:

- available all the time at full capacity



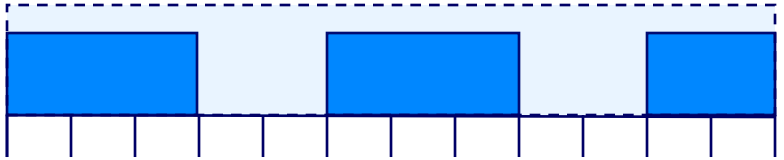
Fractional resource:

- available all the time at reduced capacity



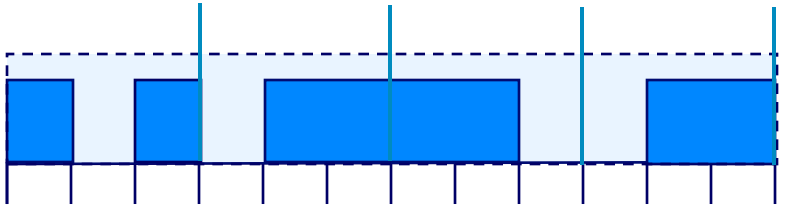
Partitioned resource:

- available some of the time at full capacity



Periodic resource:

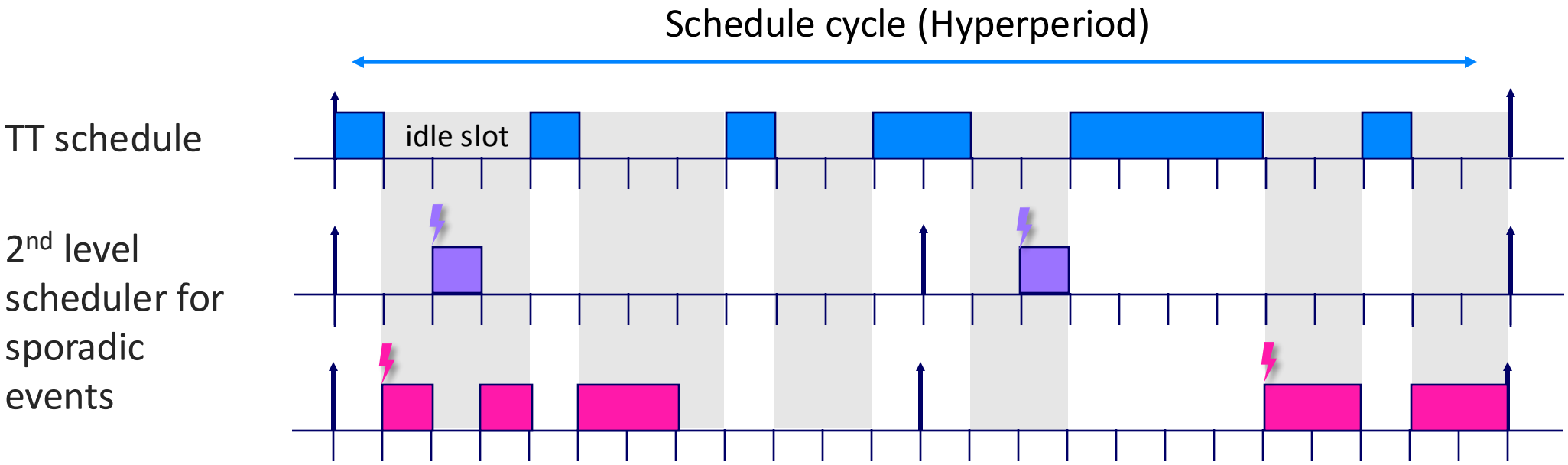
- available periodically at full capacity (e.g., R(2,3,3))



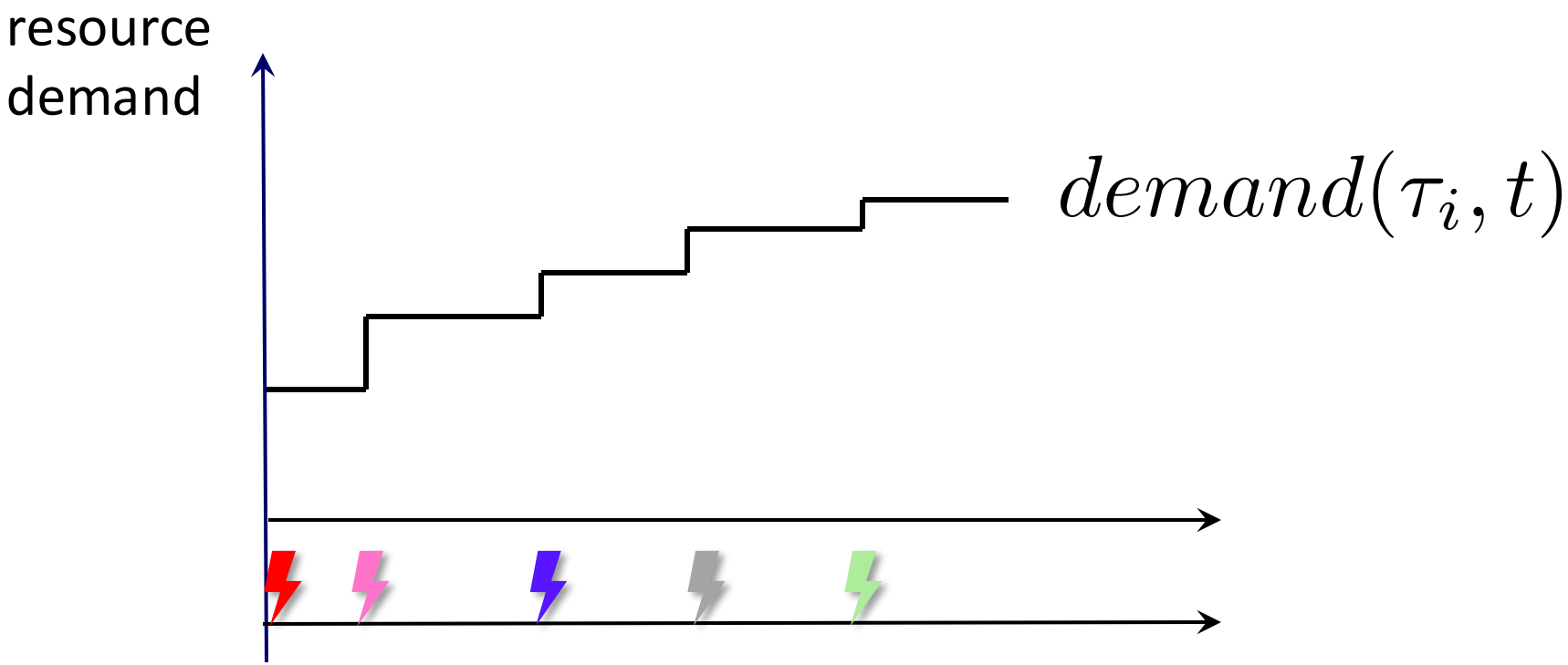
Periodic resource abstraction

Explicit Deadline Periodic: R(C,D,T) [Shin2008]

- We can do the schedulability analysis for ET independently of the TT schedule
- We then treat the polling tasks (periodic resources) as normal TT tasks and create the TT schedule for all
- We still have to find a feasible TT schedule for both TT and polling tasks



Response-time analysis



$$demand(\tau_i, t) = C_i + \sum_{\forall \tau_j \in HP(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j.$$

Response-time analysis

To verify the schedulability of constrained-deadline task τ_i under fixed-priority scheduling in uniprocessor systems, the time-demand analysis (TDA) [Lehoczky89] can be adopted.

Task τ_i is schedulable under DM scheduling, where $HP(\tau_i)$ is the set of tasks with higher priority than task τ_i since, if and only if

$$\exists t \text{ with } 0 < t \leq D_i \text{ and } C_i + \sum_{\forall \tau_j \in HP(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

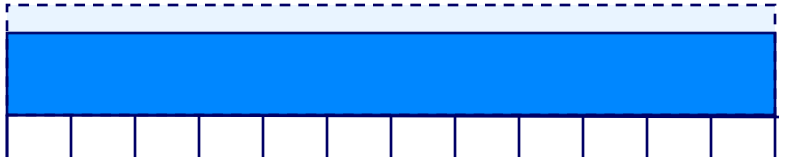
- C_i: Computation time of τ_i
- $\left\lceil \frac{t}{T_j} \right\rceil$: The number of times task τ_j is executed within an interval of length t in the worst case
- C_j: Computation time of τ_j

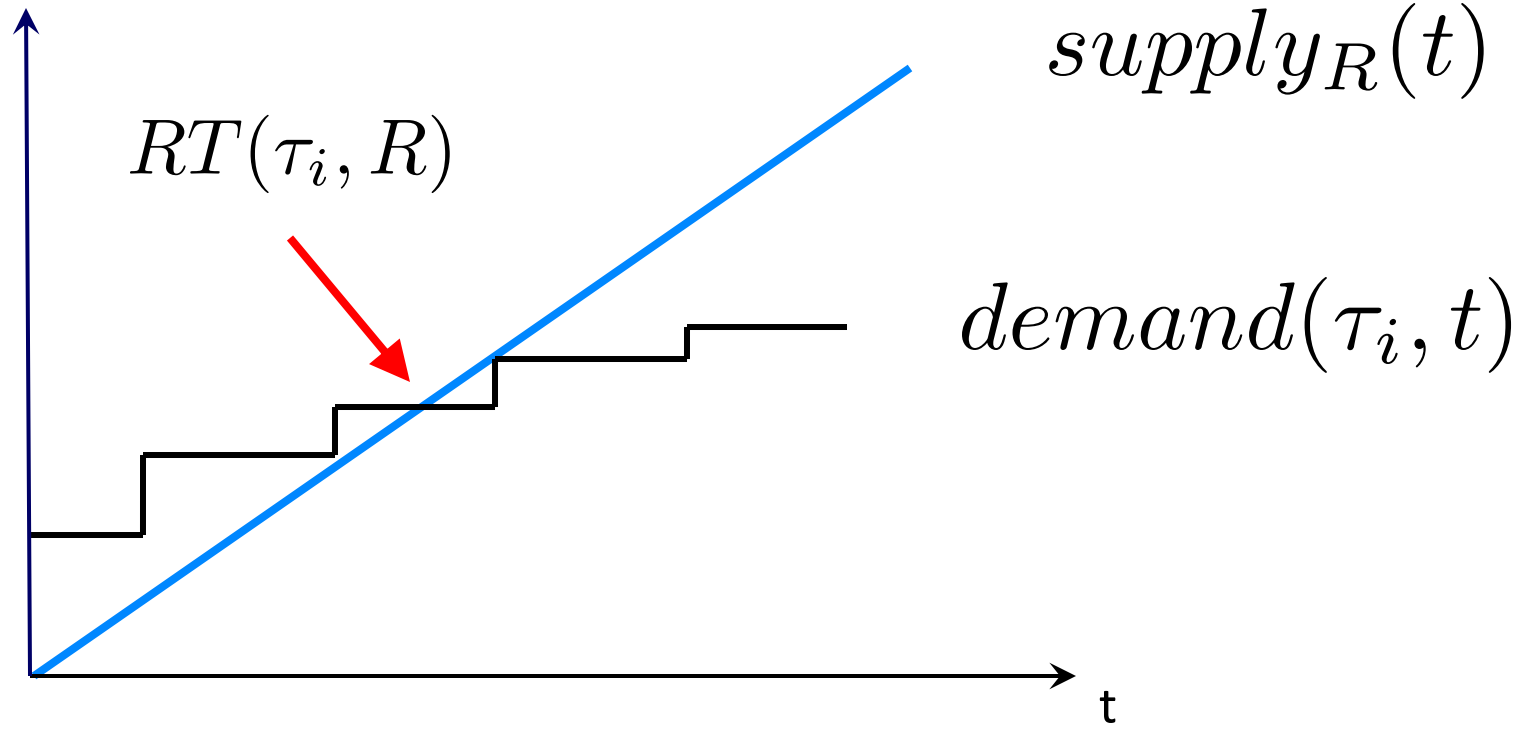
J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In IEEE Real-Time Systems Symposium, 1989.

Response-time analysis on a dedicated resource

Dedicated resource:

- available all the time at full capacity



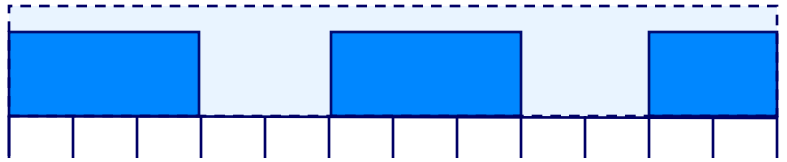


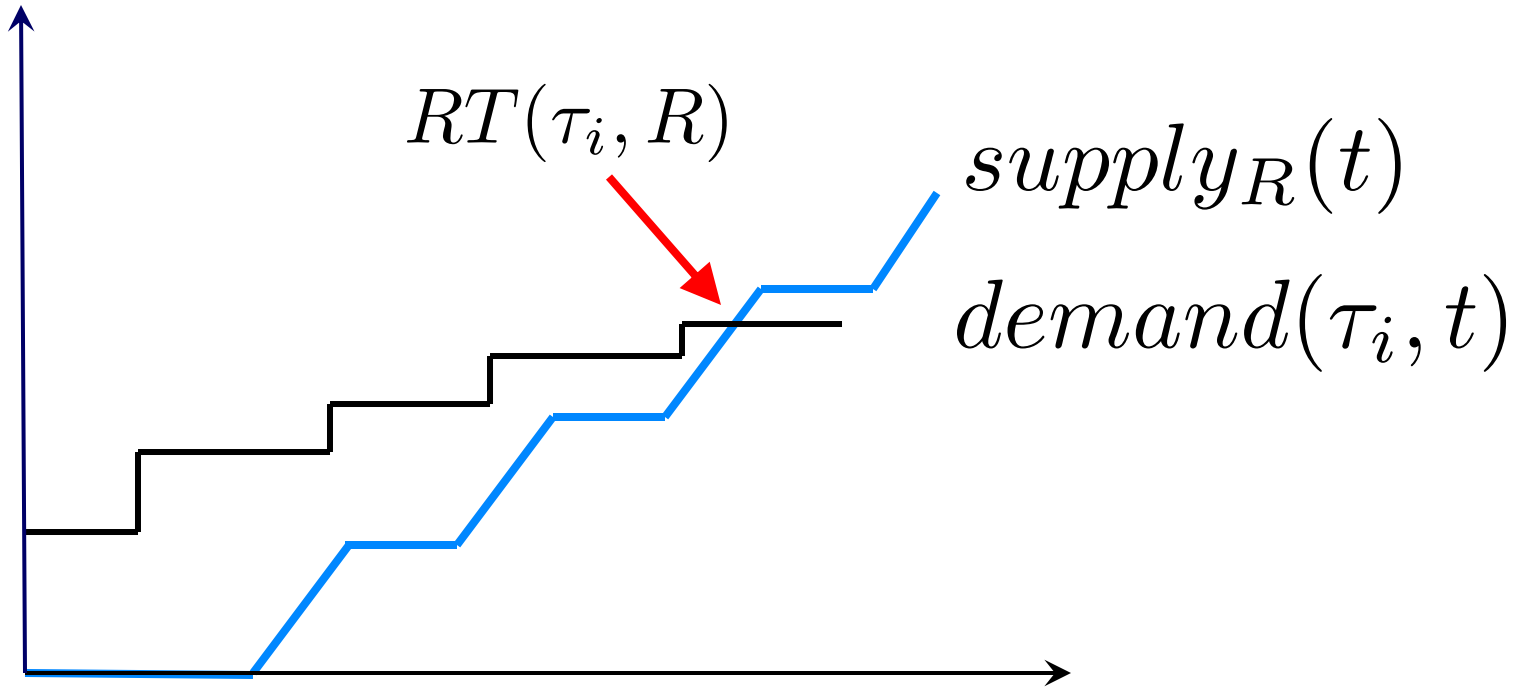
$$RT(\tau_i, R) = earliest\ t : supply_R(t) \geq demand(\tau_i, t)$$

Response-time analysis on a partitioned resource

Partitioned resource:

- available some of the time at full capacity



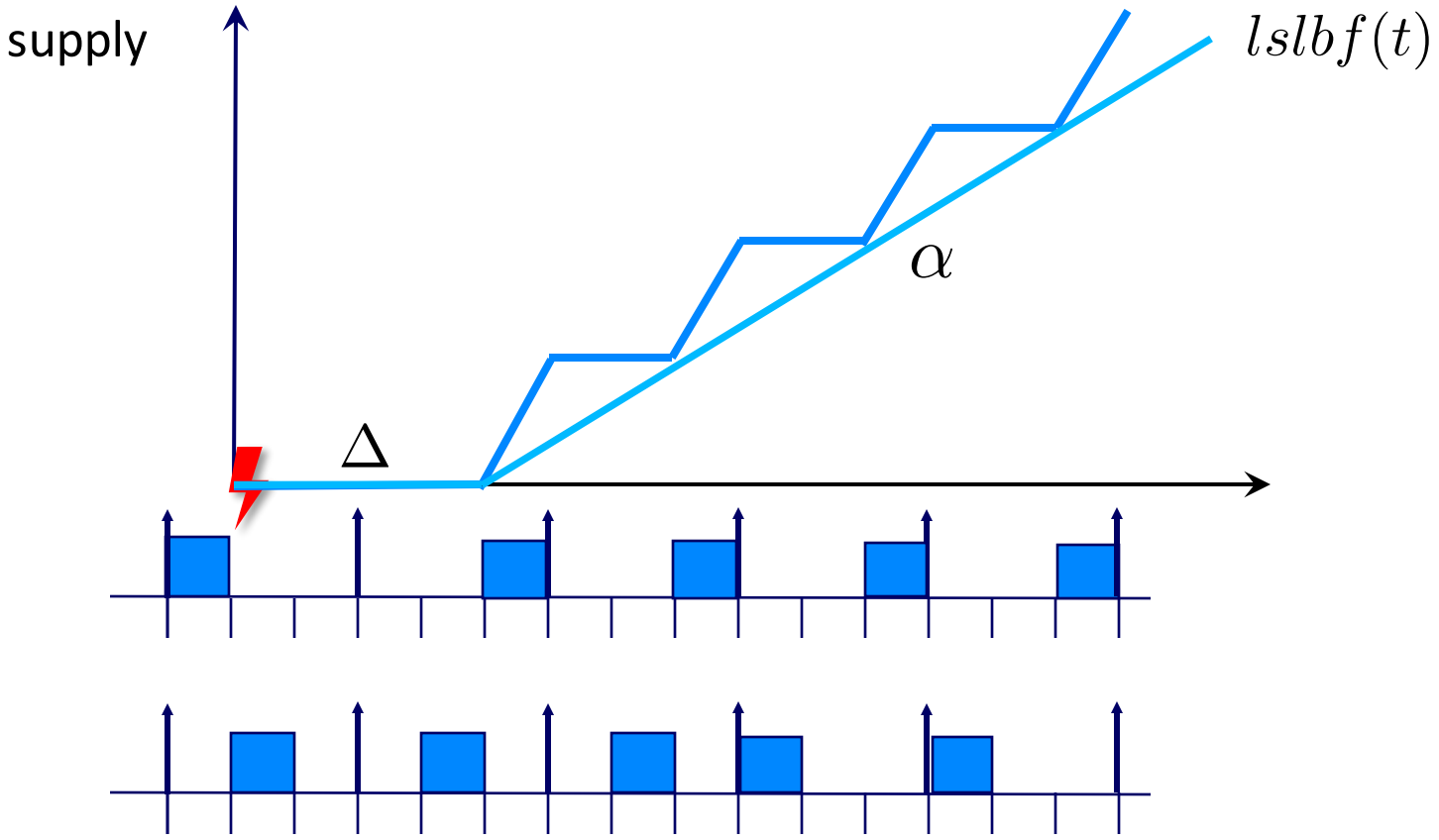


$$RT(\tau_i, R) = earliest\ t : supply_R(t) \geq demand(\tau_i, t)$$

Periodic resource abstraction - AdvPoll

Explicit Deadline Periodic: R(C,D,T) [Shin2008]

- we do not care when the supply C is given, as long as it is given in every period T until the deadline D
- we can use the worst-case linear supply bound and schedulability test defined below [Almeida2004]



$$\alpha = \frac{C}{T}$$

$$\Delta = T + D - 2C$$

$$lslbf(t) = max\{0, (t - \Delta) \cdot \alpha\}$$

$$RT(\tau_i, R) = earliest\ t :$$

$$t \geq \Delta + demand(\tau_i, t)/\alpha$$

$$RT(\tau_i, R) = earliest\ t : t \geq \Delta +$$

$$\frac{1}{\alpha}\left(C_i + \sum_{\forall \tau_j \in HP(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j\right)$$

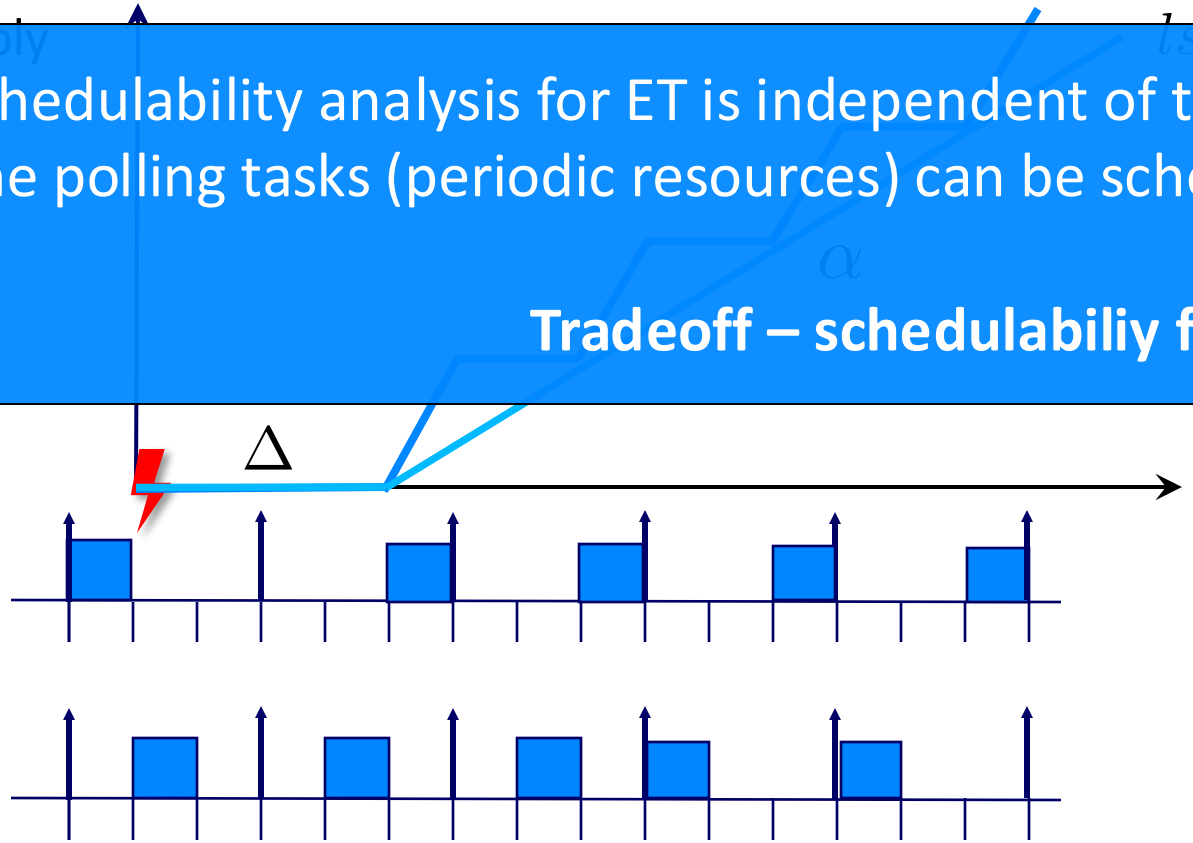
Periodic resource abstraction - AdvPoll

Explicit Deadline Periodic: R(C,D,T) [Shin2008]

- we do not care when the supply C is given, as long as it is given in every period T until the deadline D
- we can use the worst-case linear supply bound and schedulability test defined below [Almeida2004]

- Schedulability analysis for ET is independent of the TT schedule
- The polling tasks (periodic resources) can be scheduled as normal TT tasks

Tradeoff – schedulabiliy for runtime



$$t \geq \Delta + demand(\tau_i, t)/\alpha$$

$$RT(\tau_i, R) = earliest\ t : t \geq \Delta + \frac{1}{\alpha}\left(C_i + \sum_{\forall \tau_j \in HP(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j\right)$$

Another way to express this schedulability condition

Task τ_i is schedulable under DM scheduling, where $HP(\tau_i)$ is the set of tasks with higher priority than task τ_i since, if and only if

$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(T_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t$$

$$\mathcal{P}_i(t) \rightarrow \begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i \right) \cup \mathcal{P}_{i-1}(t). \end{cases}$$

Has to hold for all τ_i

And at least for one of the values in the set of points of interest

Lipari G, Bini E. A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing. 2005;1(2):257-269. doi:10.3233/EMC-2005-00019

Another way to express this schedulability condition

Earliest point of $P_{i-1}(D_i)$ for which the workload at D_i is smaller than D_i

$$C_i + \min_{t \in P_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \leq D_i.$$

I am getting only a fraction of the CPU, so scale all execution times

$$\bigwedge_{i=1 \ldots n} \bigvee_{t \in P_{i-1}(T_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil \frac{C_j}{\alpha} \leq t$$

The tasks are not receiving any execution time from the global scheduler during the interval $\Delta > 0$

$$\Delta + \frac{C_i}{\alpha} + \min_{t \in P_{i-1}(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \frac{C_j}{\alpha} + (D_i - t) \leq D_i$$

The schedulability condition then becomes:

$$\bigwedge_{i=1 \ldots n} \bigvee_{t \in P_{i-1}(D_i)} \Delta \leq t - \frac{1}{\alpha} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Lipari G, Bini E. A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing. 2005;1(2):257-269. doi:10.3233/EMC-2005-00019

Another way to express this schedulability condition

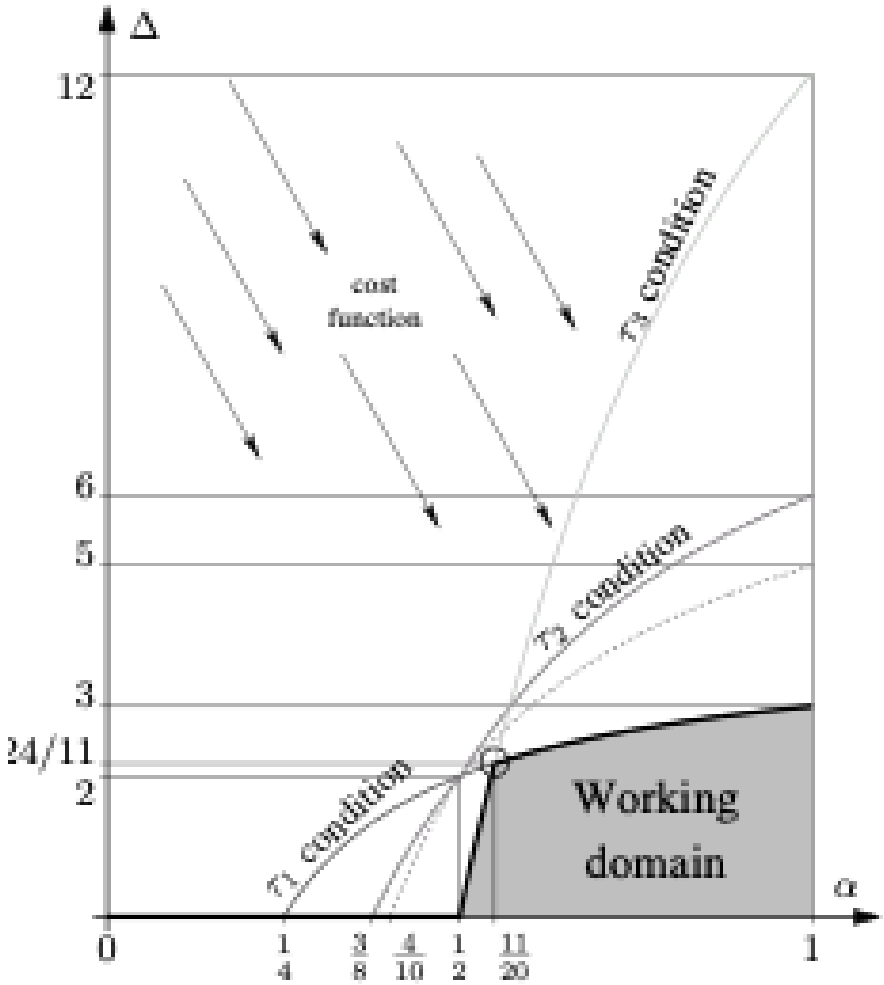
$$\bigwedge_{i=1\ldots n} \bigvee_{t \in \mathcal{P}_{i-1}(D_i)} \Delta \leq t - \frac{1}{\alpha} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Ca be put in terms of optimization problem with e.g. swithing cost

$$c_1 \frac{T_{\text{Overhead}}}{P} + c_2 \alpha$$

When choosing the server parameters, we must balance two opposite needs:

- the required bandwidth should be small, to not waste the total processor capacity;
- the server period should be large, otherwise the time wasted in context switches performed by the global scheduler will be too high.



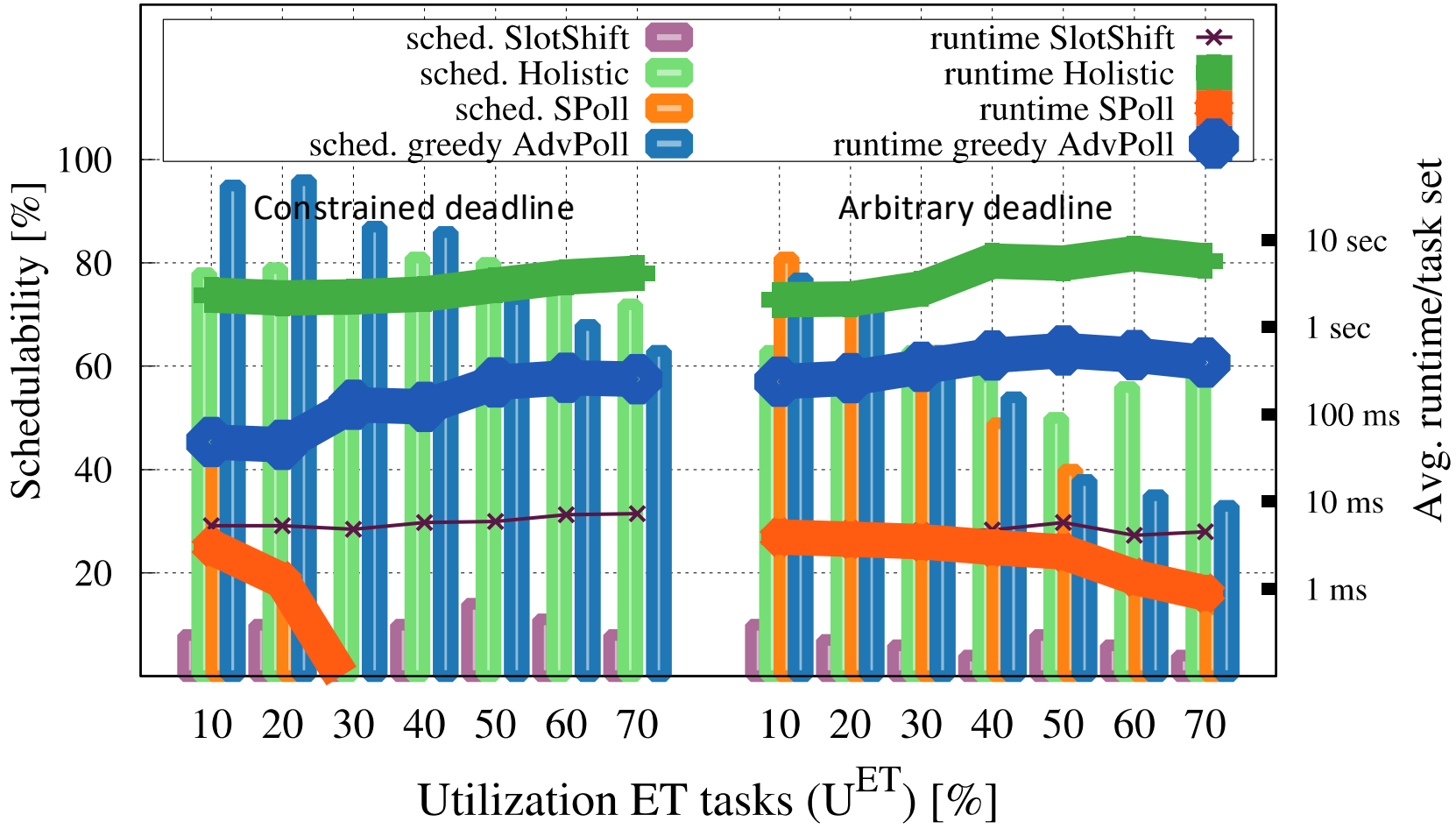
Lipari G, Bini E. A methodology for designing hierarchical scheduling systems. Journal of Embedded Computing. 2005;1(2):257-269. doi:10.3233/EMC-2005-00019

Single-core experiments

- 30 TT and 20 ET tasks per task set
- Periods ∈ {5, 10, 20, 40, 80} ms
- Microtick of 250µs

- Constrained Deadline: Di is uniformly selected in upper half of [Ci,Ti]
- Arbitrary Deadline: Di ∈ [Ci,5 · Ti]

- 20% TT task utilization
- Increasing ET utilization
- 100 task sets per configuration

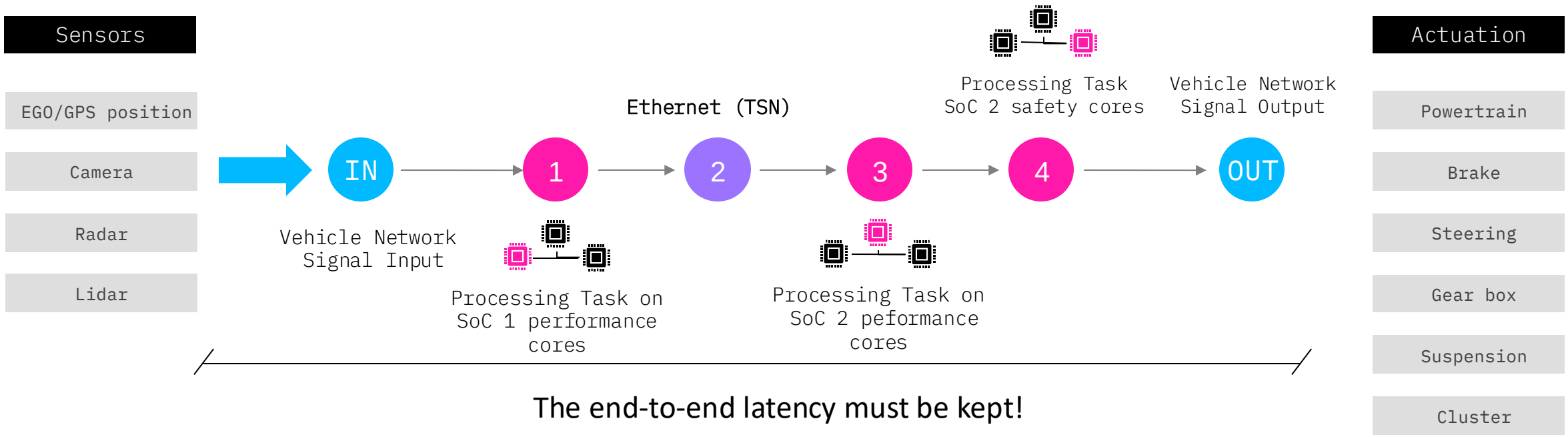


SPoll is quickest and can be used for low utilization
Holistic has better schedulability at high utilization but takes much longer time
AdvPoll has the best overall trade-off between runtime and schedulability

Real-time networks

Computation Chains Spanning across a Network of Chips

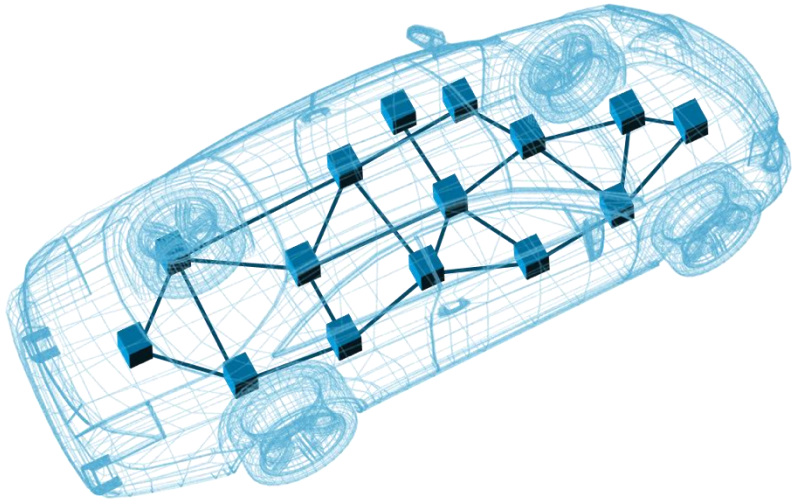
Example: ADAS/AD workload

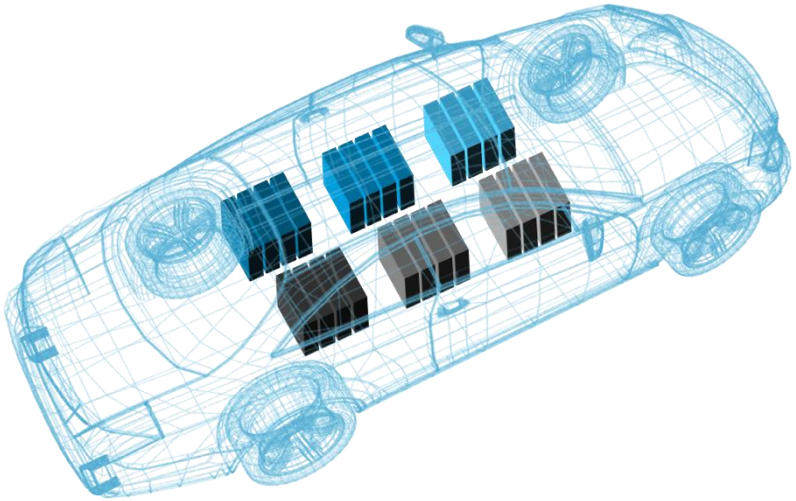


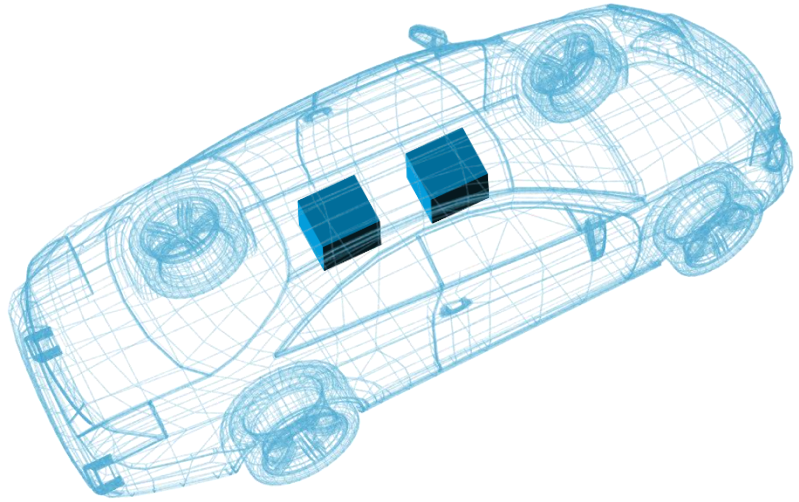
The end-to-end latency must be kept!

> Task Scheduling and Network Communication must be aligned **across SoCs and CPUs**

Automotive networks







n ECU: n functions

DISTRIBUTED
E/E ARCHITECTURE

5-7 domains: n functions

DOMAIN
E/E ARCHITECTURE

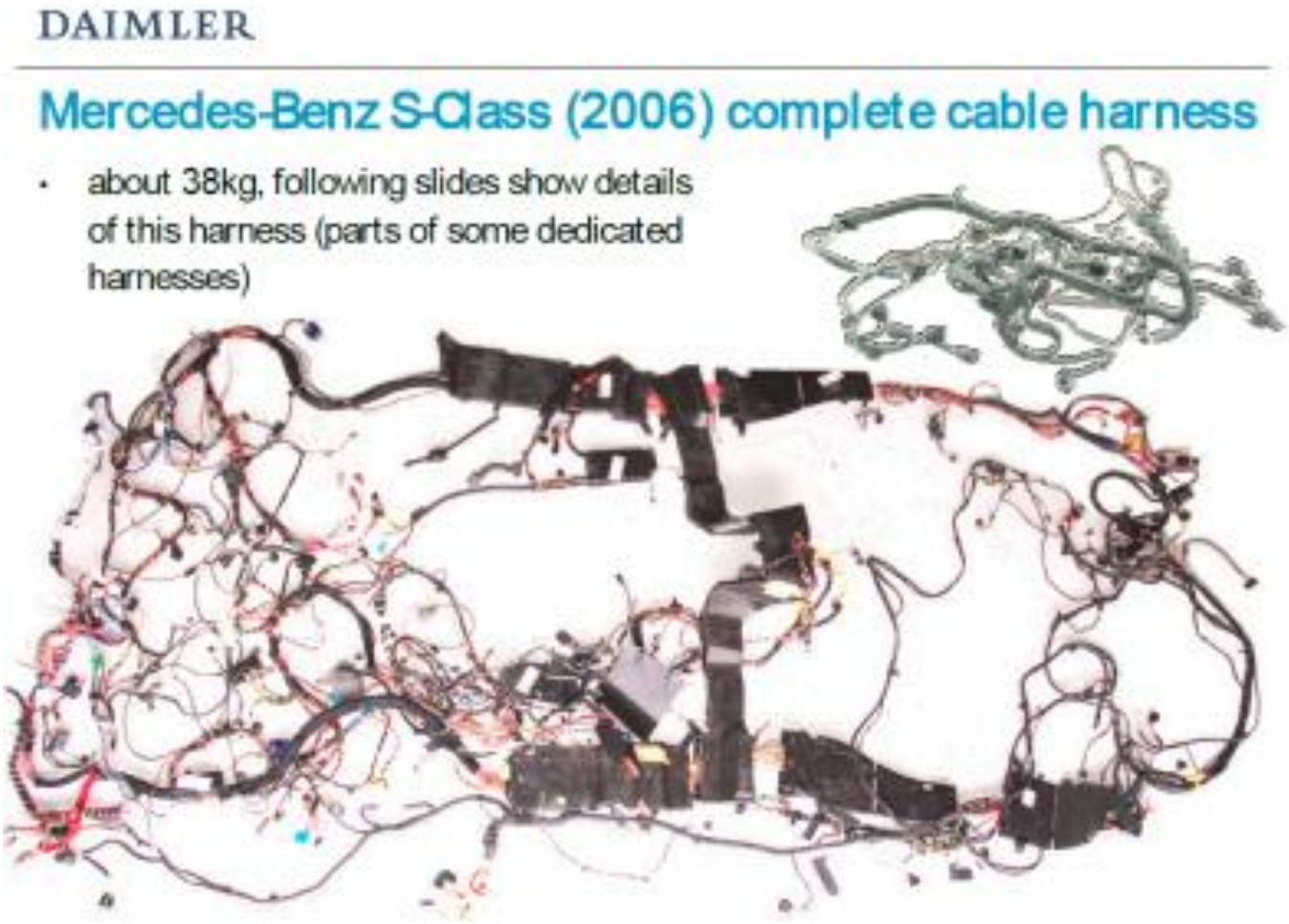
2 ECUs: n functions

CENTRALISED
E/E ARCHITECTURE

YESTERDAY & TODAY | 2020 – 2025 | 2025 +

Automotive networks

- **Where do we come from?**
- Increased number of ECUs led to lots of wiring.

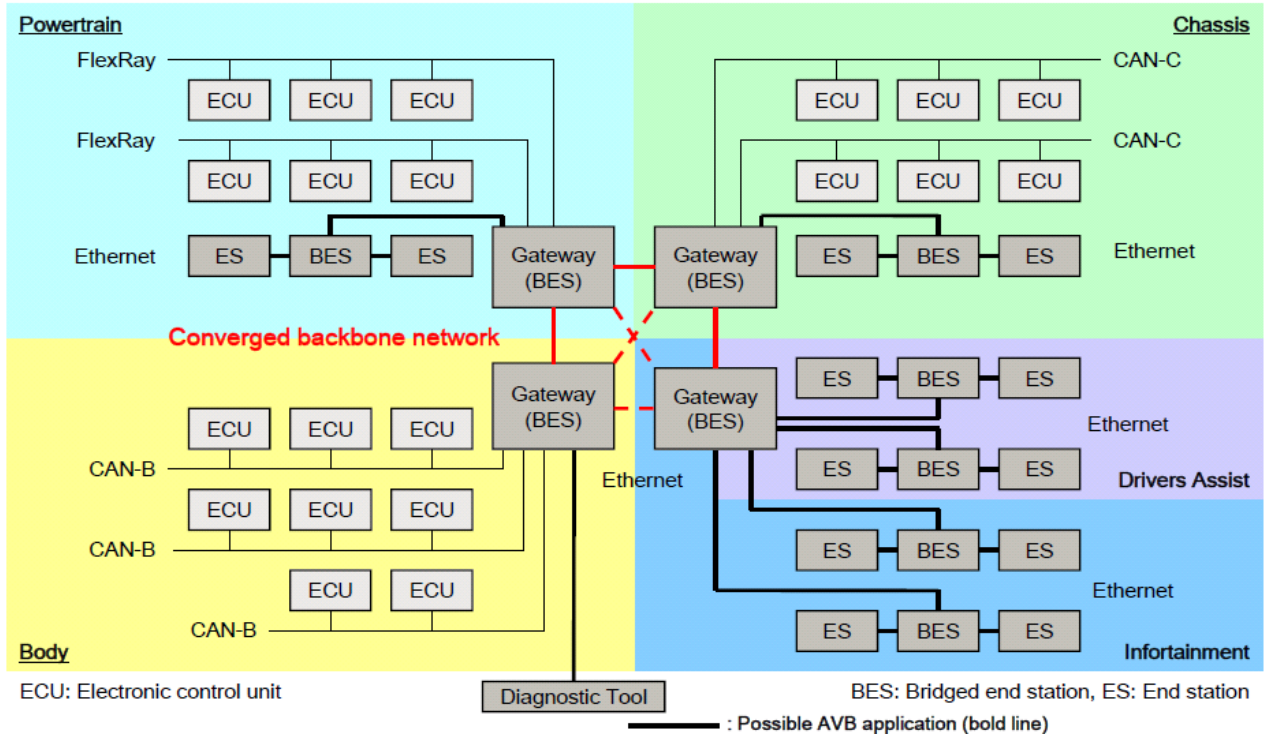


Source: IEEE 802.3 RTPGE SG

Automotive networks

Where do we come from?

- Increased number of ECUs led to lots of wiring.
- Automotive networks were a mixture of domain and zonal architectures interconnected by different protocols and technologies e.g., CAN, FlexRay and

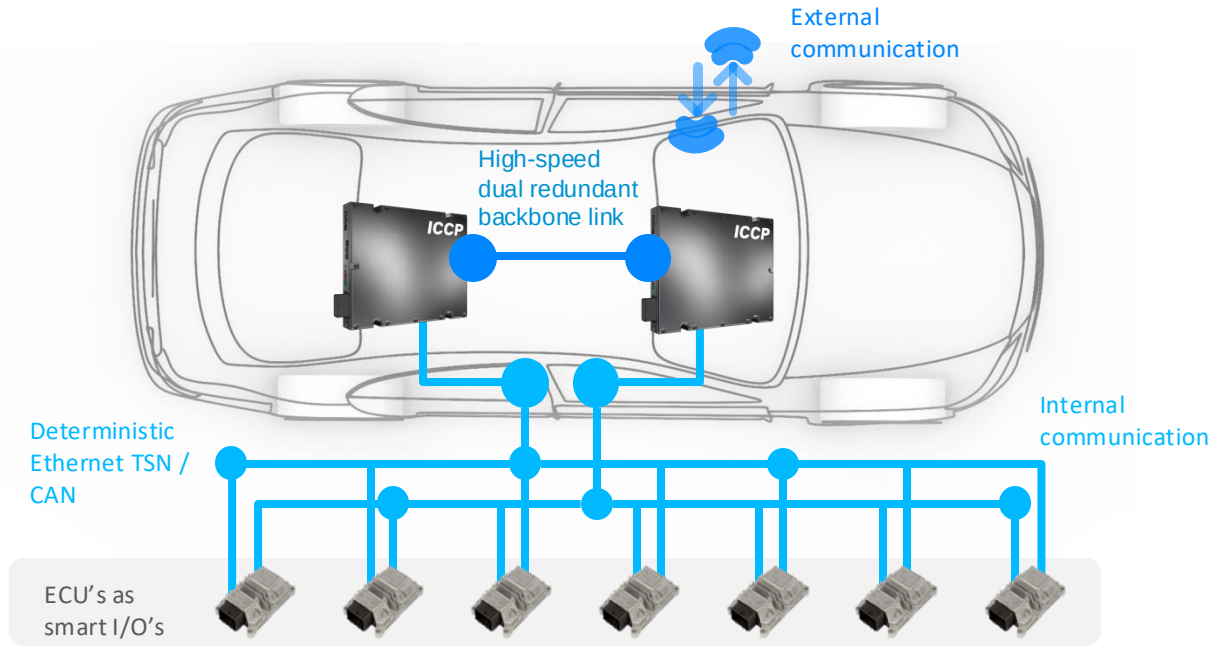


Source: IEEE 802.1 AVB TG presentation

Automotive networks

Where do we come from?

- Increased number of ECUs led to lots of wiring.
- Communication networks were a mixture of zonal, domain with network technologies with several technologies implied. E.g., CAN, FlexRay and LIN.
- Centralized computing platforms like ICCP aim at gathering the formerly distributed computations into a more powerful unit.
  - Centralization might help to harmonize the communication network.
  - But applications are becoming more complex and might have additional communication requirements.



Distributed real-time system

Real-time distributed systems

- Guarantee on timely and predictable delivery of messages across the network
- Task execution (message production and consumption) synchronized with communication

EVENT-DRIVEN SCHEDULING
Event-handlers with predictable response time

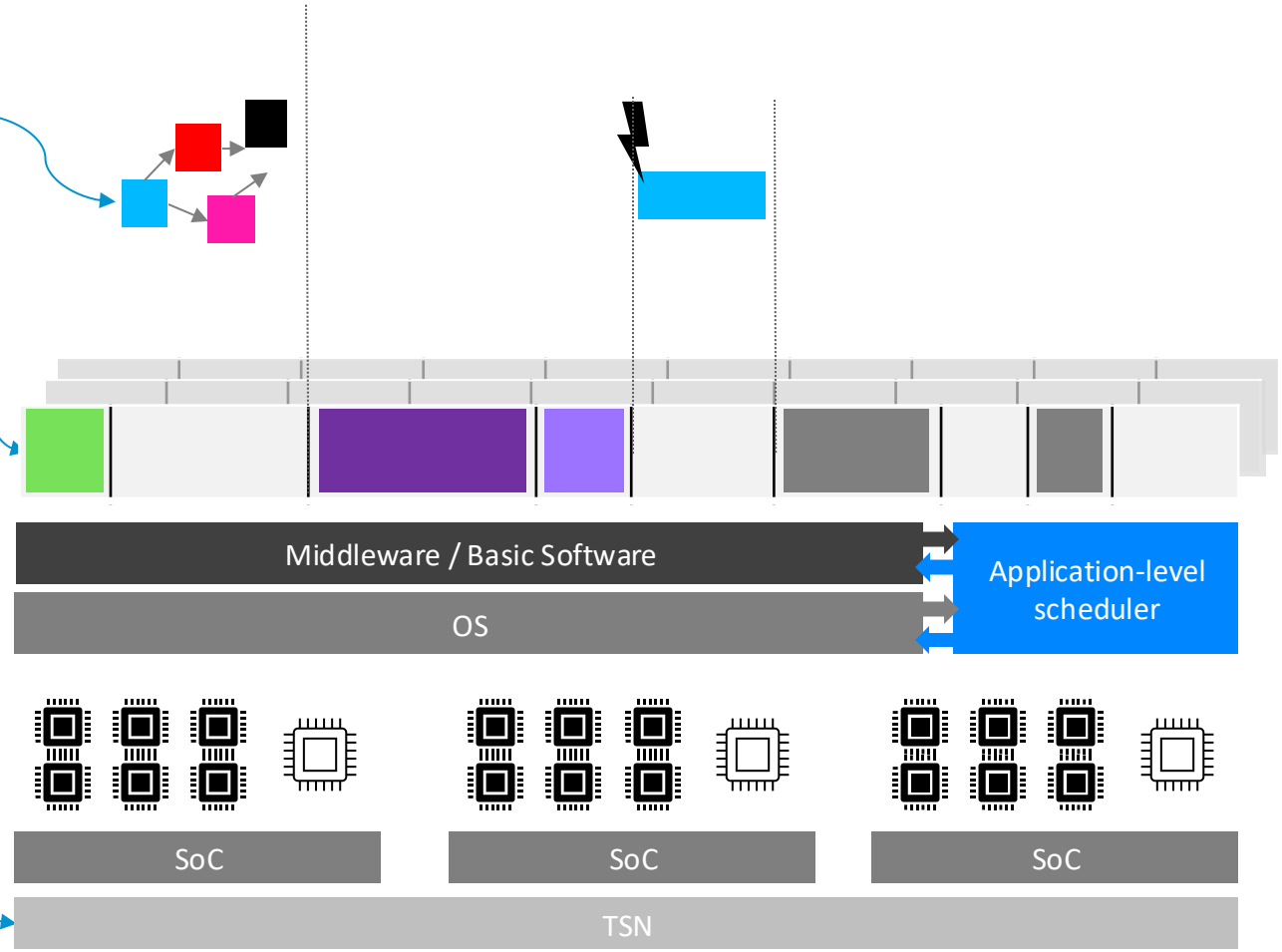
2nd Level Scheduler

TIME-TRIGGERED SCHEDULING
For resource allocation / partitioning, typically done by Software / System Integrator

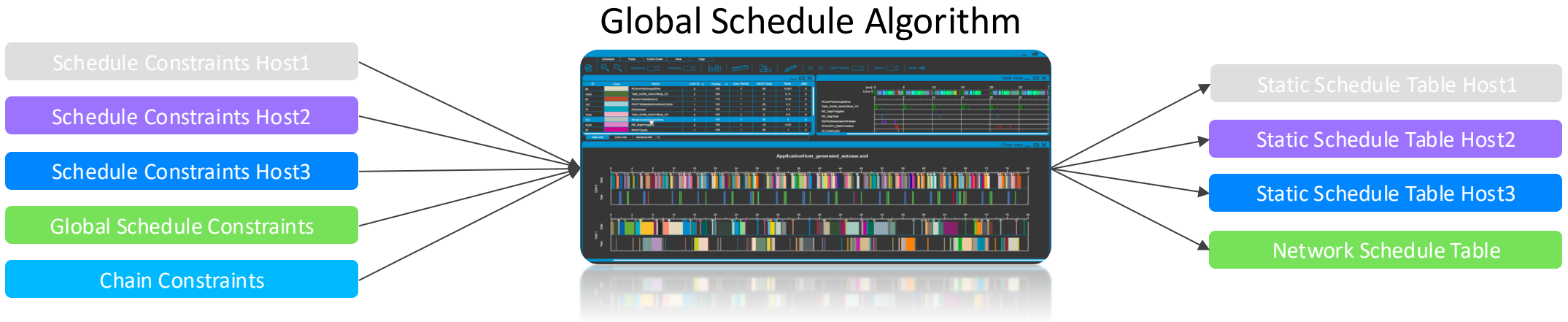
1st Level Schedule

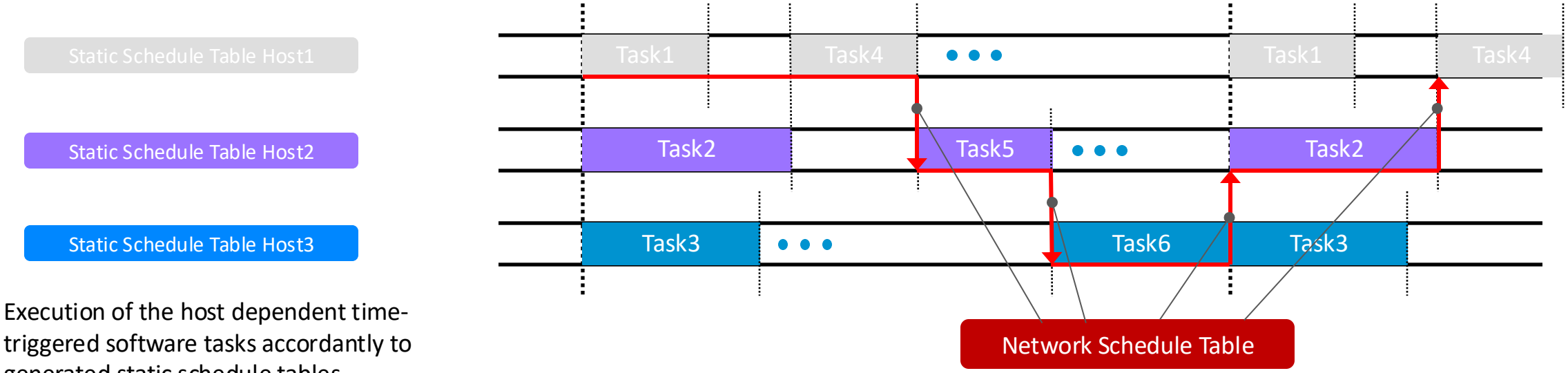
NETWORK SCHEDULING
The network scheduling, enables to ensure end-to-end behaviors across SoCs

TSN Communication



Time-Triggered scheduling

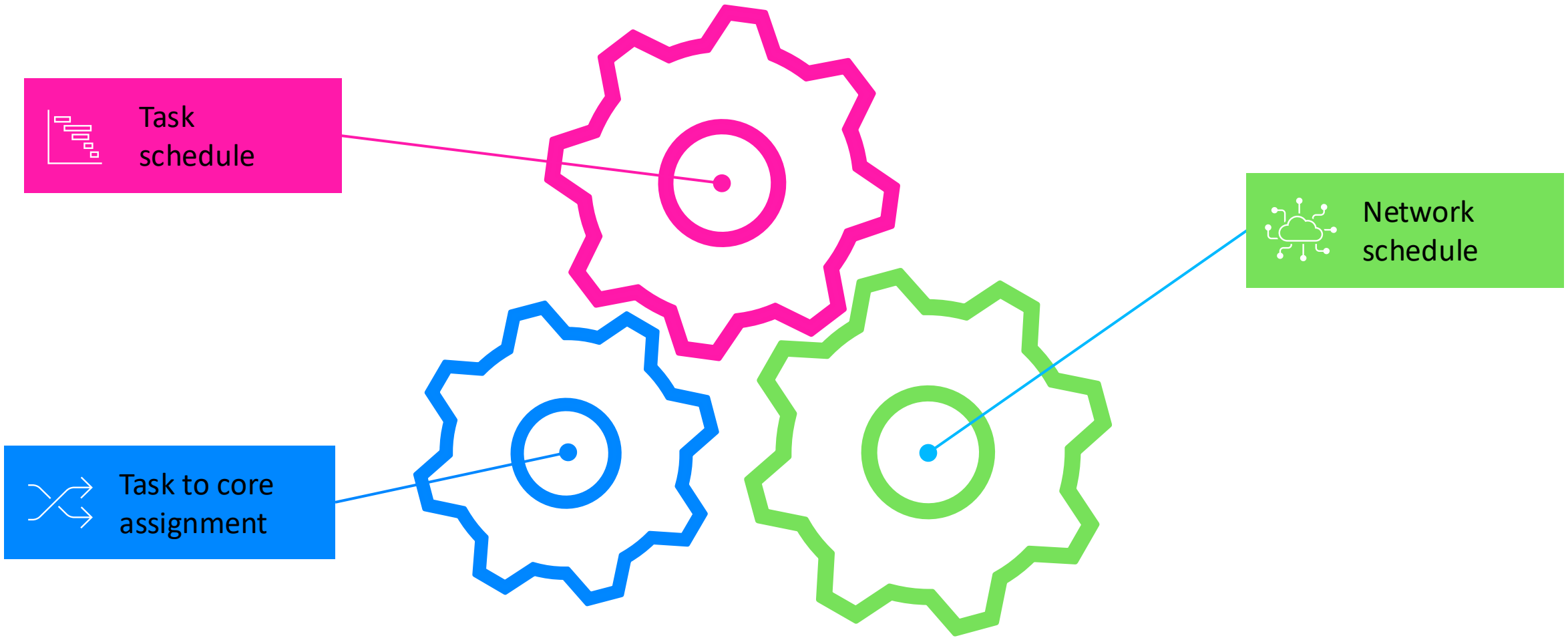




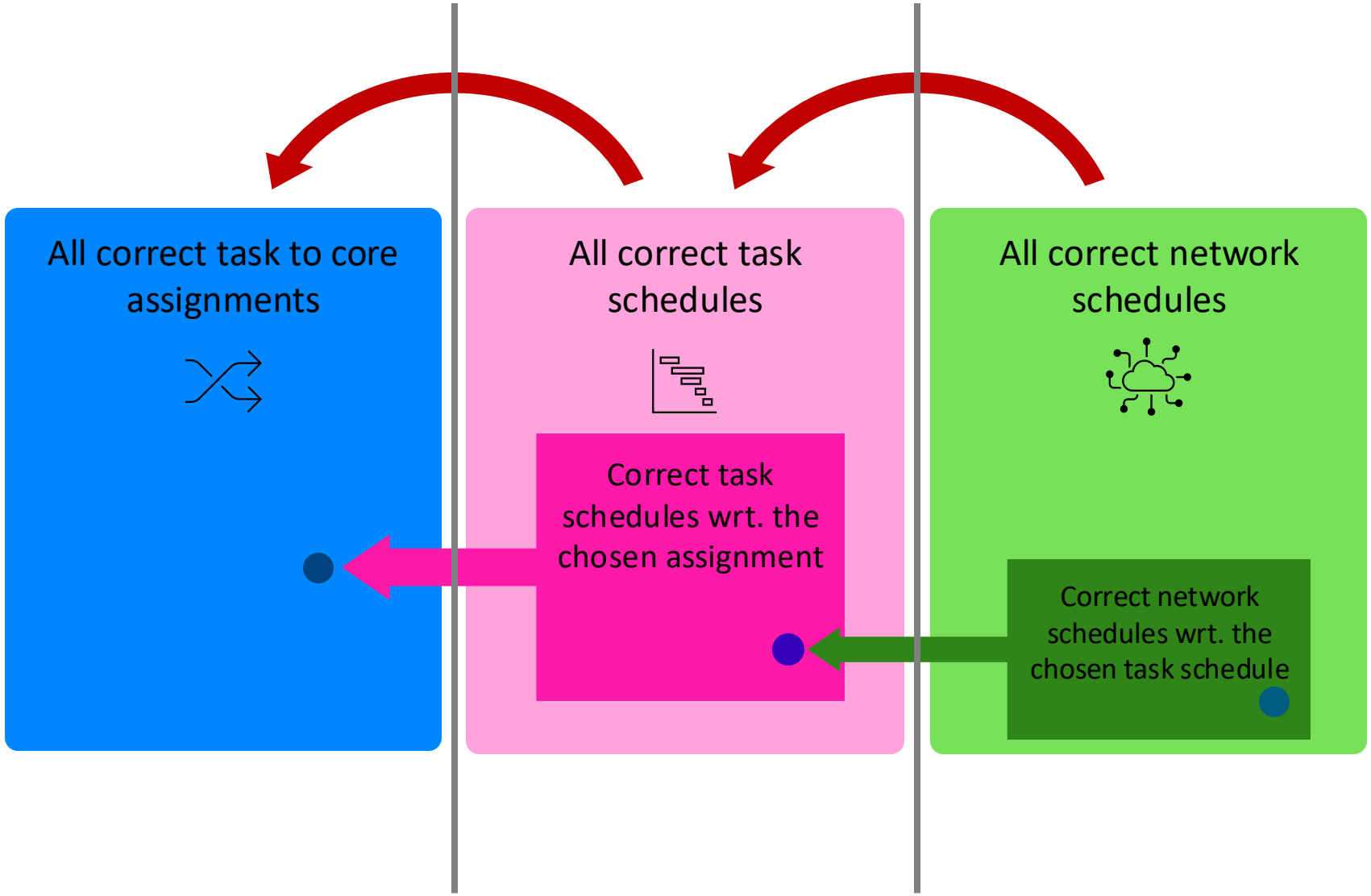
Execution of the host dependent time-triggered software tasks accordantly to generated static schedule tables

Handling of the communication messages accordantly to generated time-triggered timing

Problem dimensions



Problem dimensions



Thank you!


silviu.craciunas@tttech.com


https://scraciunas.github.io/


www.linkedin.com/in/scraciunas