Challenges of Using AI in Automotive CPS

85th Meeting IFIP WG 10.4 - February 2024 - Industrial Panel

Ramon Serna Oliver

Principal Scientist @ TTTech Labs

TTTech fields of operation in CPS

From fail-safe to fail-operational

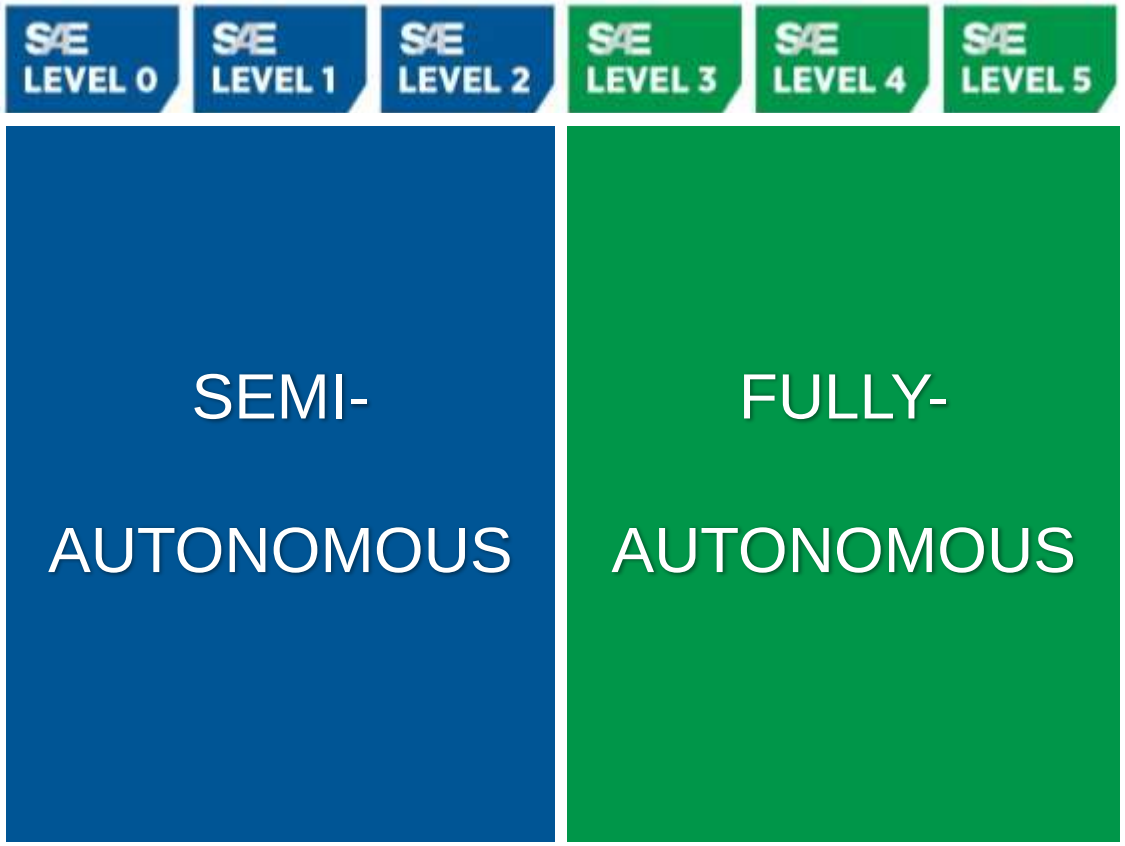
Fail-safe

Fail-operational

Safety by design according to highest safety standards in multiple industries

- ISO 13849
- ISO 25119
- ISO 19014
- ISO 26262
- IEC 61508
- DO 178C/254

Levels of Driving Automation



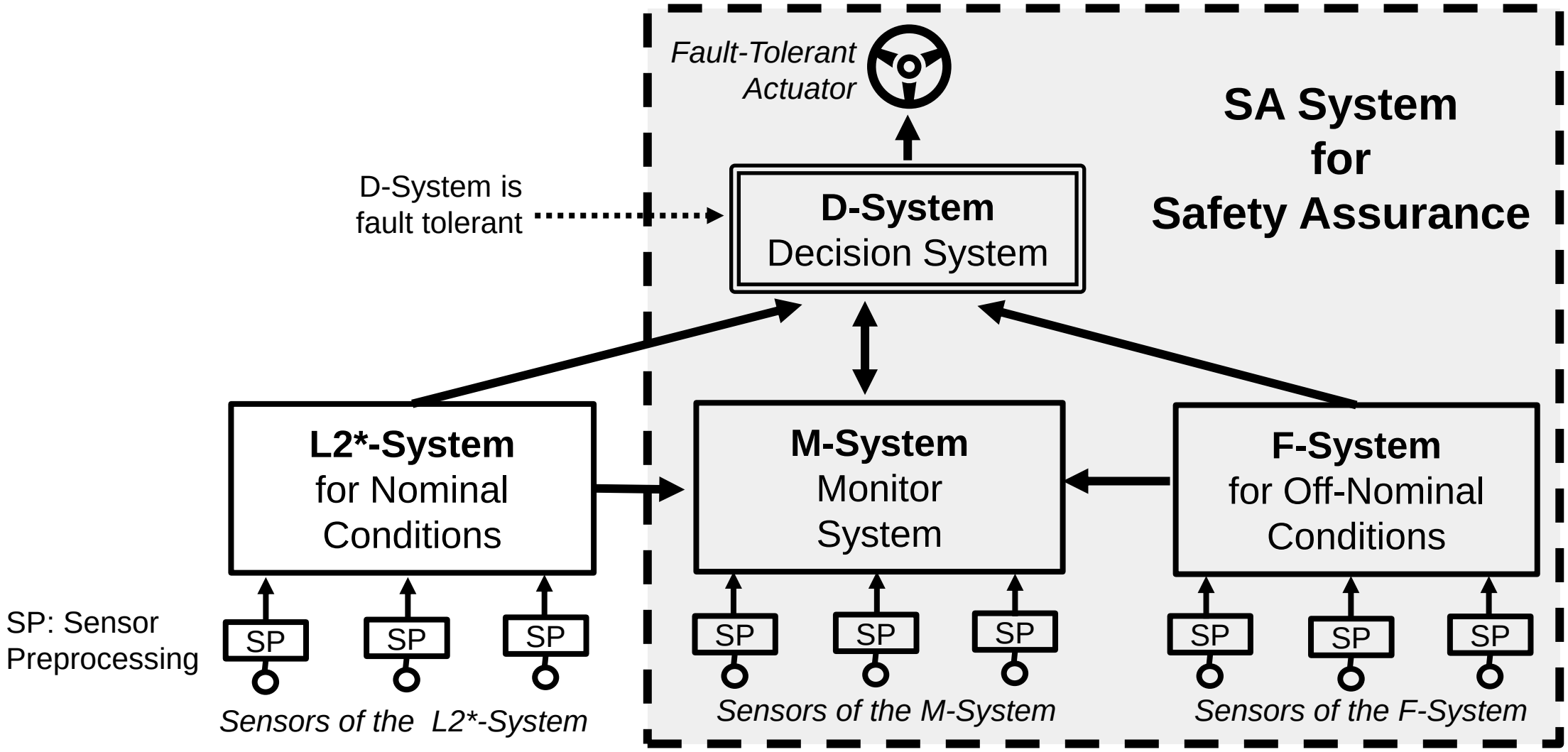
Source: https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic

With the transition from L2 to higher levels the aim is to transfer responsibility from the human to the machine.

Problem Statement

- Hardware and software automotive platforms experience an increasing complexity.
- AI currently enables functionalities otherwise not possible (e.g. perception).
- However, integration of AI in a CPS is not trivial neither on the hardware nor software sides:
  - The failure rate of a single AI subsystem is rather high.
  - Besides functional correctness, the timeliness of the system operations is crucial to guarantee that the system remains fail-operational.
  - Methods to achieve a sufficient validation of correctness and timeliness of AI-based systems are unclear.
- A strategy to address these challenges is to decompose the system into multiple independent subsystems, wherein ideally each subsystem forms a fault-containment unit (FCU).

A proposed Decomposition of an AD L4 System



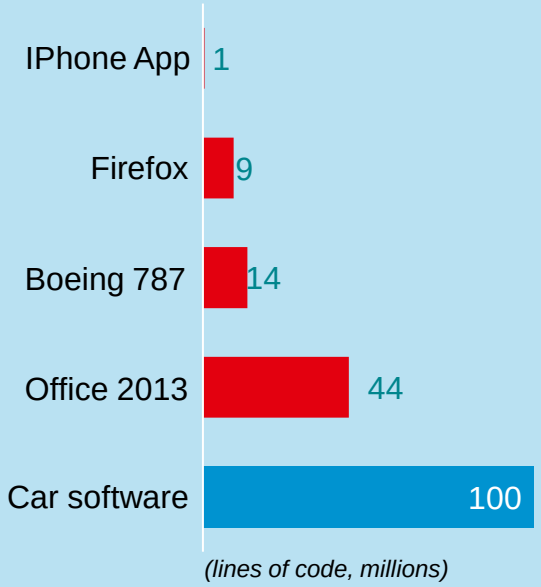
https://www.the-autonomous.com/news/an-architecture-for-driving-automation/

Problem Statement

- Hardware and software automotive platforms experience an increasing complexity.
- AI currently enables functionalities otherwise not possible (e.g. perception).
- However, integration of AI in a CPS is not trivial neither on the hardware nor software sides:
  - The failure rate of a single AI subsystem is rather high.
  - **Besides functional correctness, the timeliness of the system operations is crucial to guarantee that the system remains fail-operational.**
  - Methods to achieve a sufficient validation of correctness and timeliness of AI-based systems are unclear.
- A strategy to address these challenges is to decompose the system into multiple independent subsystems, wherein ideally each subsystem forms a fault-containment unit (FCU).

Growing Complexity of Automotive Functions

Increasing complexity of car software



+

Increasing complexity of timing requirements

- Critical event/data-driven chains
- Hard real-time and determinism requirements
- Orchestration needs of heterogeneously distributed applications
- Mixed-criticality and Freedom From Interference requirements
- Complex task dependencies
- Non-determinism of AI/ML algorithms

=

Design-time guarantees

- Testing-based approaches are not sufficient anymore.
- Correct-by-design approach with mathematically-proven guarantees are needed.
- This implies more knowledge (e.g., timing budgets) and solutions to complex planning problems.

"*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.

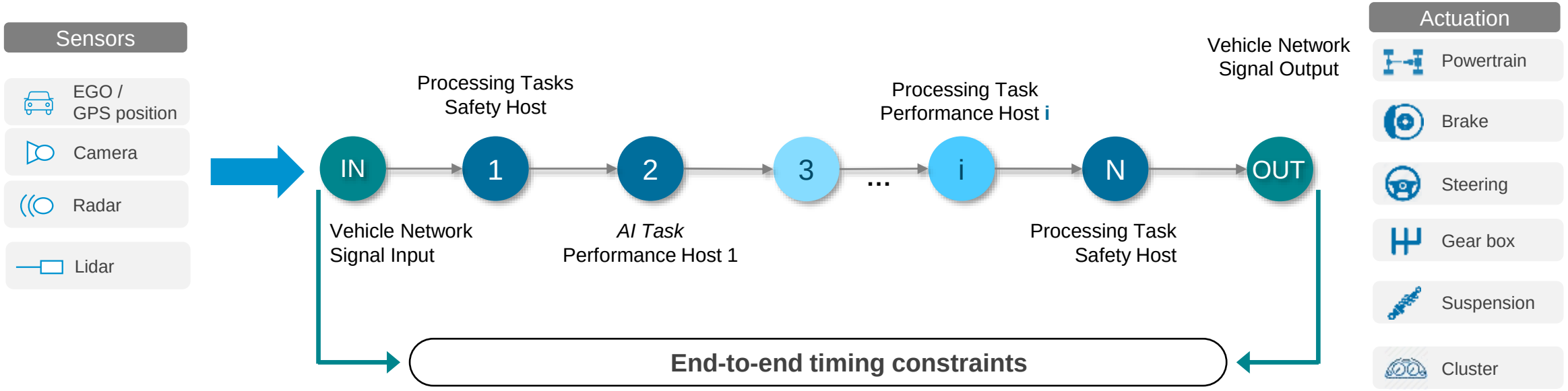
Design-time guarantees require solving a system-wide planning problem.

Functional Complexity of Automotive Functions

Event Chains

- **Event Chains** (EC), or *computation chains*, define maximum end-to-end latency between sequences of distributed software components.
- EC are modelled as data flows through the system, i.e. distributed among multiple nodes.
- The local resource configurations are arranged to guarantee that EC always meet their required latency (i.e. worst-case).
- The use of AI increases even further the system complexity due to non-determinism of algorithms and performance-oriented hardware.

Event Chains Example



Global System Planning

1

System Requirements

- System definition
- Functional constraints
- Temporal constraints
- Platform definition

2

System Model

- Hardware resources
- Workload
- Communication
- Dependencies
- Low-level constraints

3

Global Configuration (Planning)

4

Visualization, Simulation Validation & Verification

- CPU Cores
- Hardware Accelerators
- Network
- Middleware stacks
- Hypervisors
- …

5

A **complete system model** is essential to provide design-time guarantees.

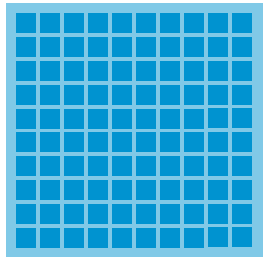
- Including functional dependencies and timing constraints.

Global planning requires configurability (i.e. control) of the artifacts, but…

- it is not always available “out-of-the-box” for AI hardware & software, due to e.g.
- the “black-box” nature of AI algorithms and the performance-oriented hardware management.

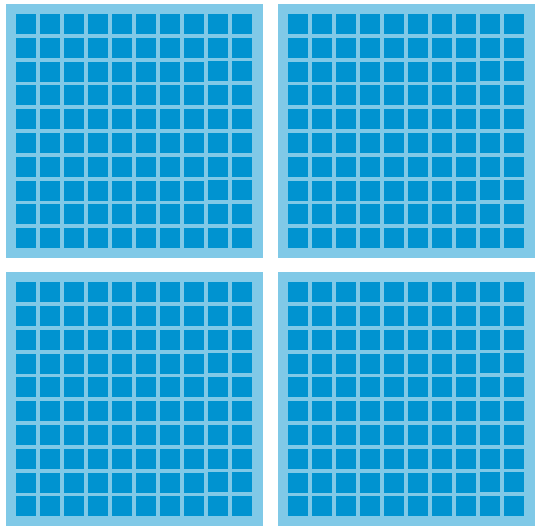
Configuration Challenge in Numbers

Software demands



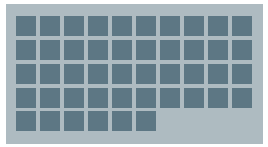
100
Functions

400 Tasks

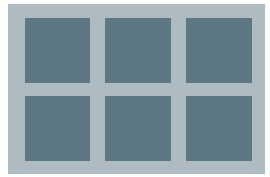


Computation unit resources

46 CPU cores



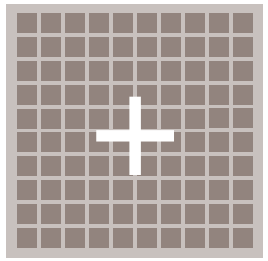
6 Hardware accelerators



2 TSN switches

1024 Streams

System constraints



100+
Time, latency, precedence constraints, etc.

Solution space

Theoretical vs Valid configurations

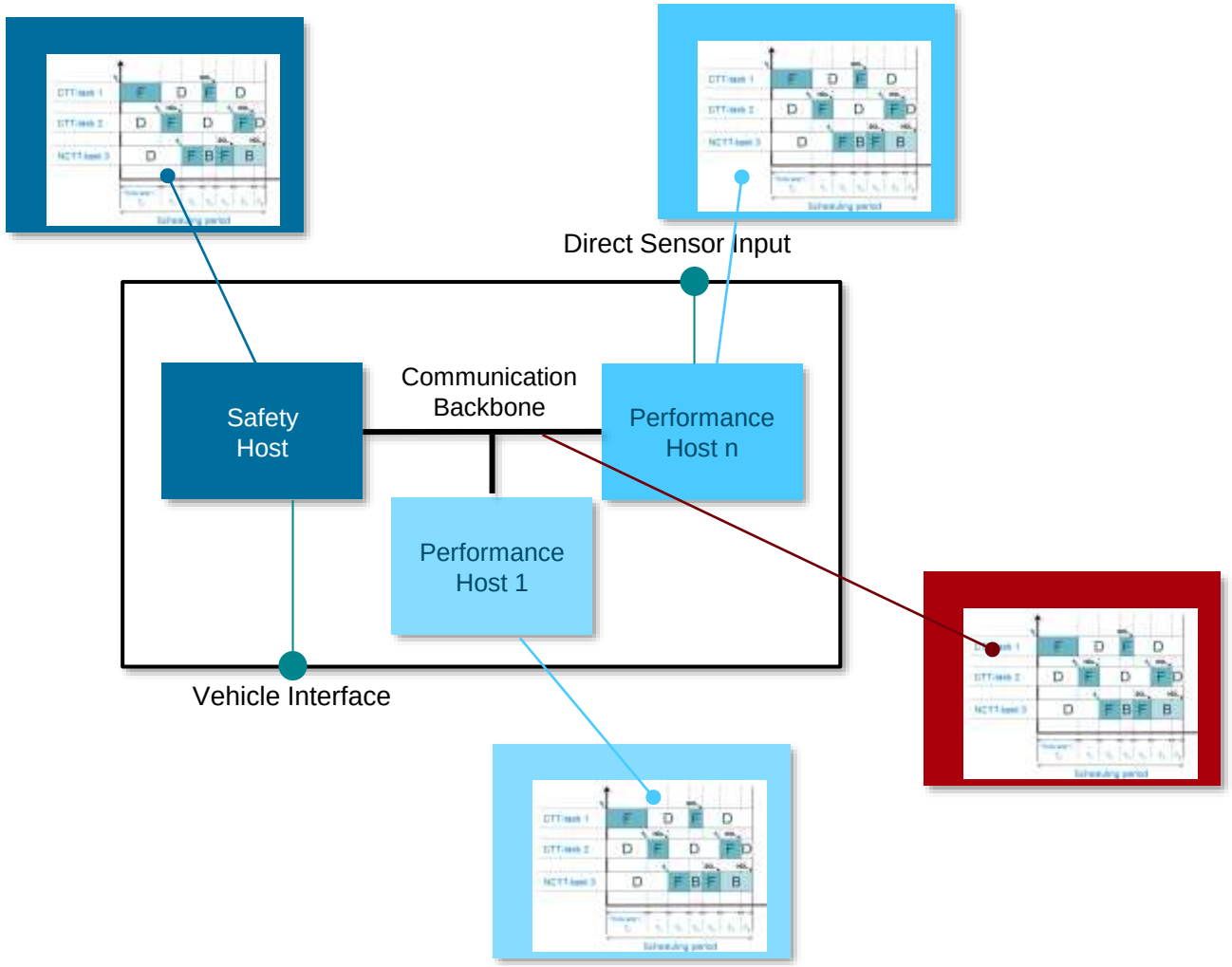
10^{5000} Possible configurations

Atoms in the universe 10^{80}

10^{5} Correct configurations

With growing system size, it becomes a challenge to plan when and where these software components are executed, and message are transmitted.

Global Planning with Time-Triggered Architecture



Software components are executed based on computed **timetables** with correct-by-design properties, e.g.:

- **Complex timing requirements**: cause-effect chains, jitter;
- **Temporal isolation**: no starvation possible, temporal isolation between all tasks, freedom-from-interference (FFI);
- **Determinism**: increased stability and testability, no unwanted run-time effects;
- **Re-simulation:** equivalent behaviour between cloud and embedded targets;
- **Synchronization to communication**: stable real-time behaviour of cause-effect chains;
- **Compositionality**: incrementally adding or modifying tasks;
- **Predictability**: many system properties become predictable, e.g., locks, task pre-emption;
- **Integration**: simplifies integration of functions from different suppliers;
- **Stability**: fewer system states, less testing / higher stability;
- **Schedulability**: more correct configurations can be realized.

Challenges Integrating AI Accelerators

Fact

- AI algorithms require complex computations exceeding CPU capabilities.
- GPUs and other *AI-accelerators* are used to leverage workload.
- The global planning of system resources allow correct-by-design guarantees.

Problem

- Accelerators are engineered to deliver high-throughput (best-effort performance), often working behind proprietary APIs.
- Freedom-from-interference (FFI) and real-time guarantees are not in scope.
- The integration of accelerators without harming timeliness of the system is not trivial.

Solution

- Overrule the native (proprietary) management of accelerators to enable launching jobs according to a global system planning.
- Ensure safety is preserved by maintaining end-to-end timeliness properties and enforce run-time compliance to the computed schedule.

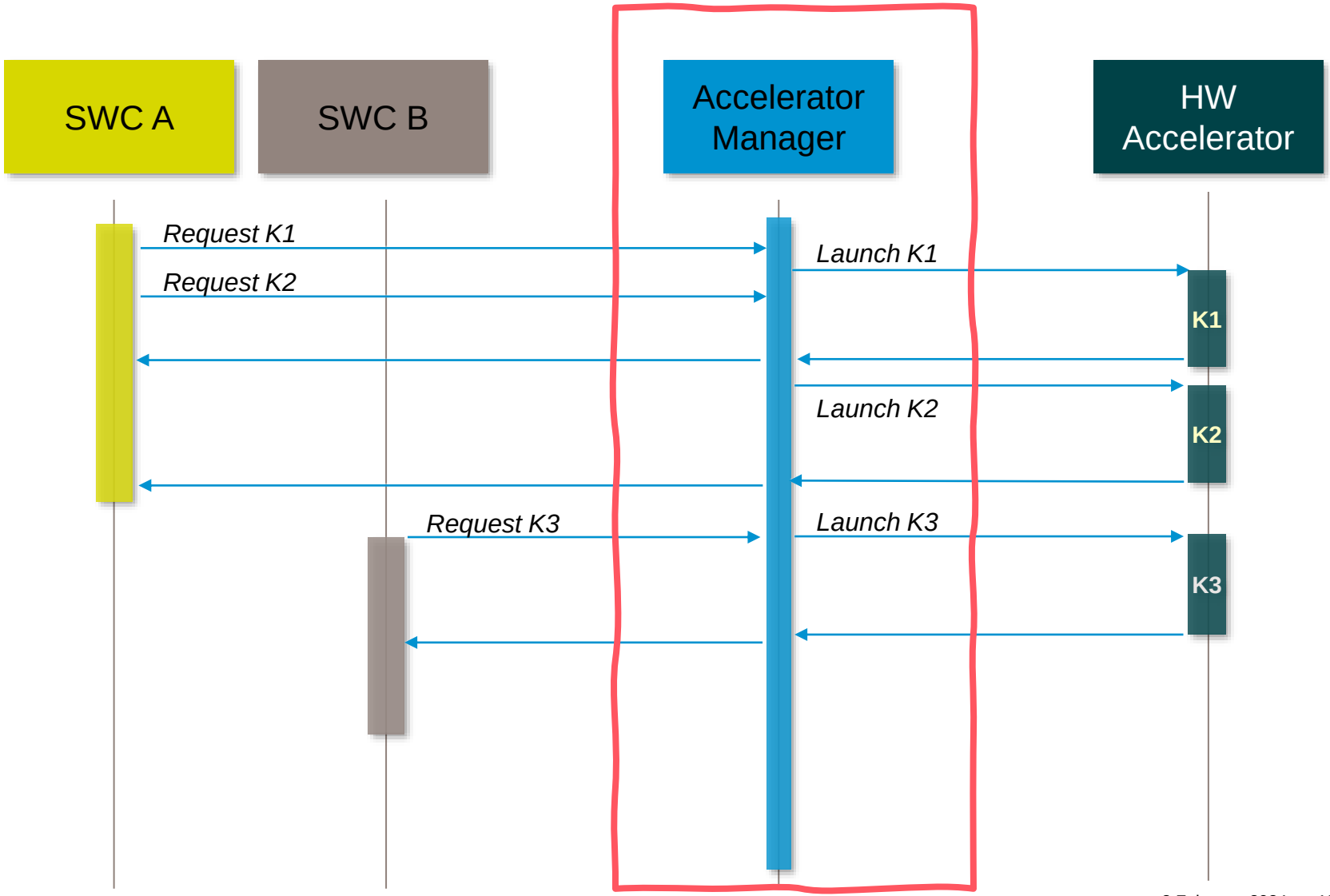
Simplified Sequence Diagram

The default direct access of software components to hardware accelerators delegates the control flow to the vendor.

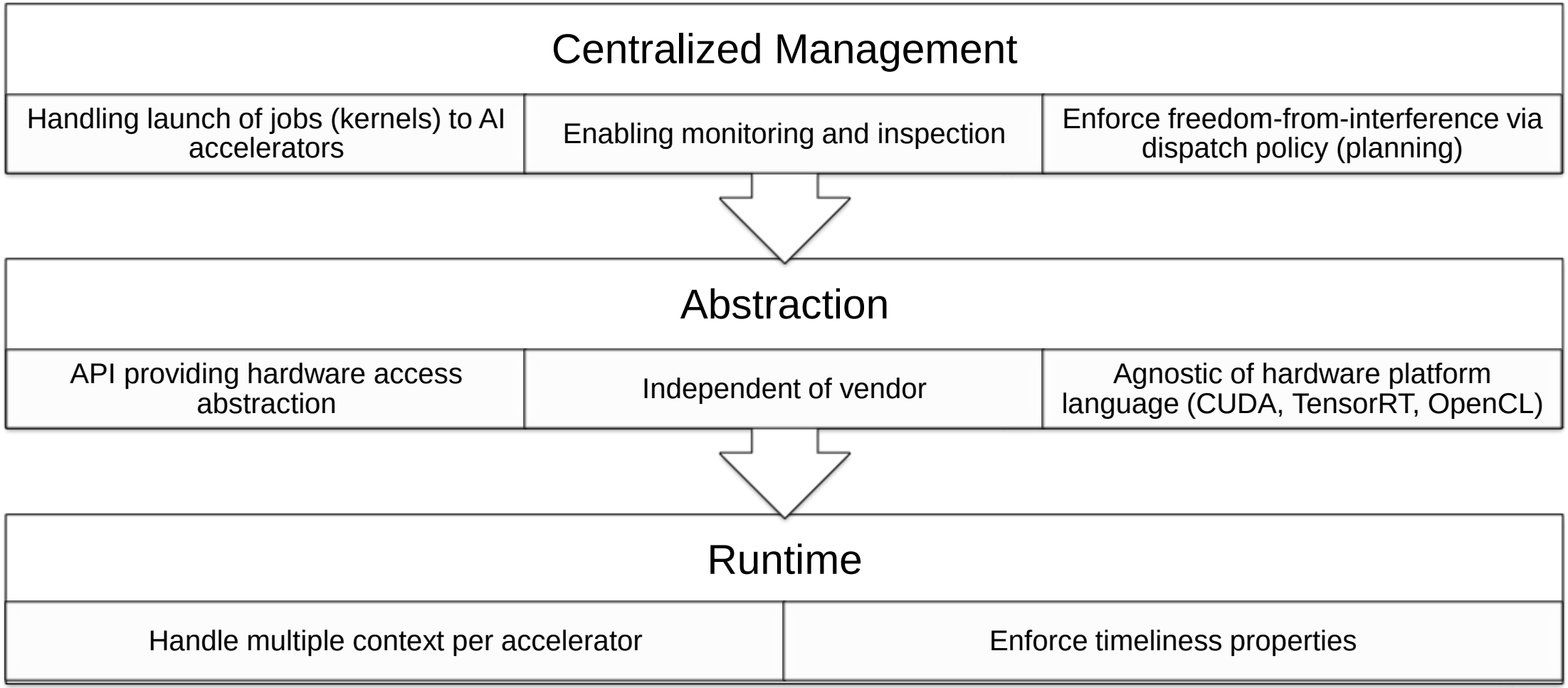
Accelerators are typically engineered to deliver performance and throughput but may not behave in a deterministic / predictable manner, wrt. e.g.:

- choice of parallelization,
- re-ordering of queued jobs,
- trading run-time vs throughput

Adding an accelerator manager acting as a broker allows fine-level control of the jobs executed by the hardware.

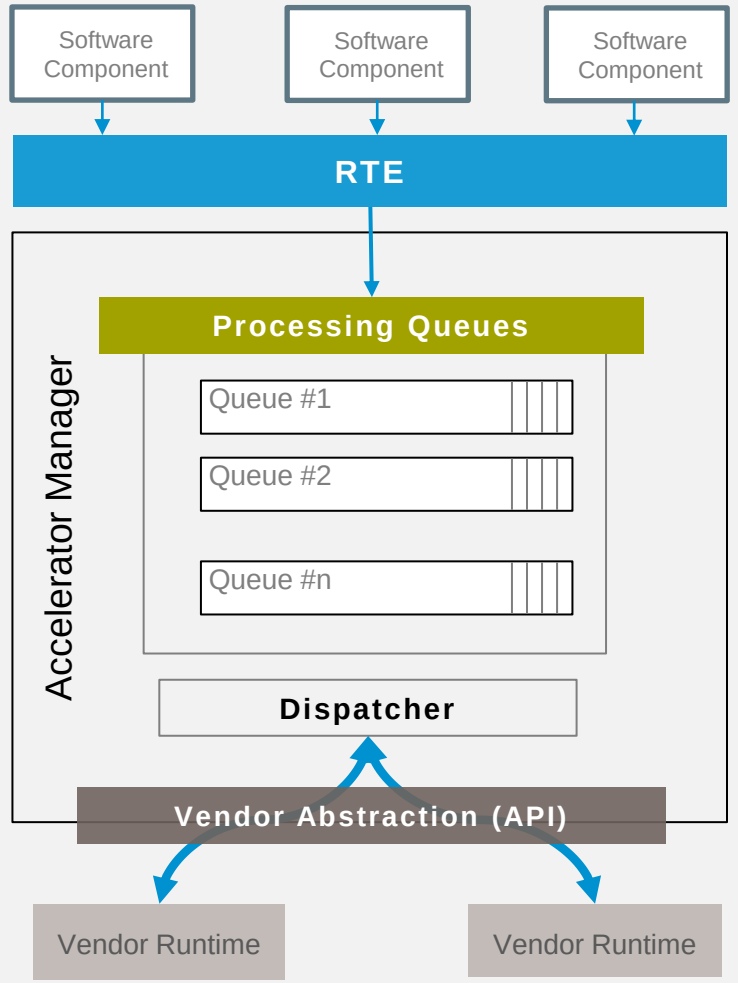


Concept Overview



Example Accelerator Manager

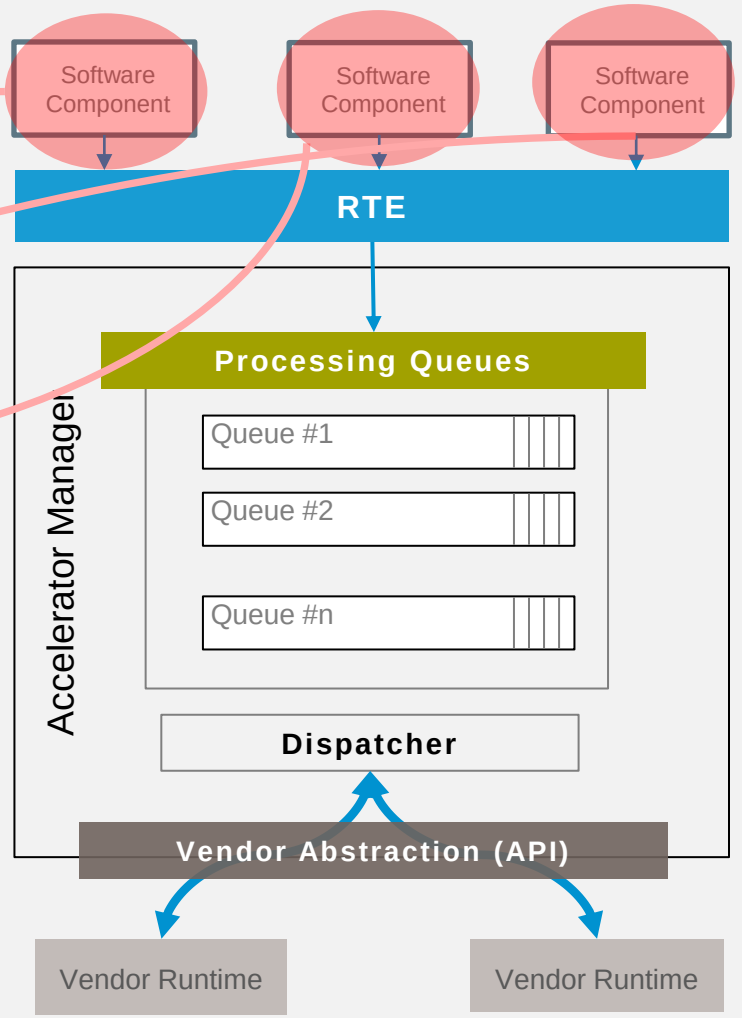
- **Accelerator Manager** provides SWC the functionality of requesting the execution of kernels in accelerators (e.g. GPU, DLA, …).
- **Dispatcher** selects jobs (kernels) from the queue of requests and launches them in the accelerators.
- **Vendor Abstraction** (device specific) abstracts the vendor runtime (drivers) to interact with the silicon.
- *Accelerator control* (requests and notifications) is handled by the **vendor runtime** via the vendor abstraction API.



Example Accelerator Manager

- **Accelerator Manager** provides SWC the functionality of requesting the execution of kernels in accelerators (e.g. GPU, DLA, …).
- **Dispatcher** selects jobs (kernels) from the queue of requests and launches them in the accelerators.
- **Vendor Abstraction** (device specific) abstracts the vendor runtime (drivers) to interact with the silicon.
- *Accelerator control* (requests and notifications) is handled by the **vendor runtime** via the vendor abstraction API.

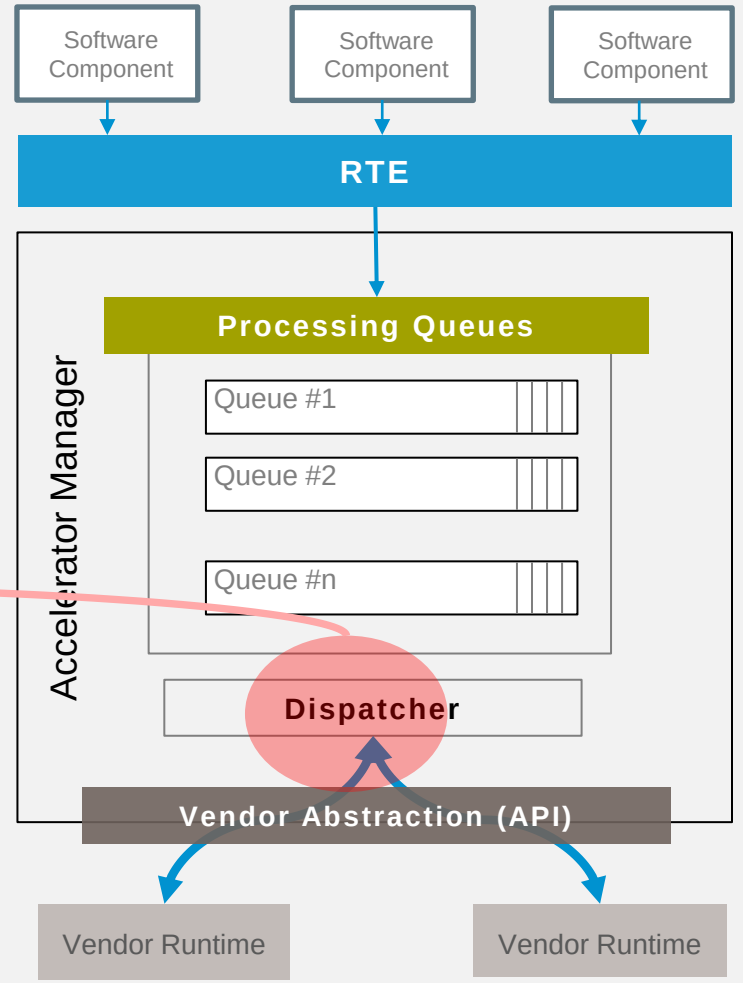
Software components request kernel execution, declare kernel properties, data transfers, etc…
The kernel is coded in the native hardware model language of the accelerator (e.g. CUDA, OpenCL, TensorRT).



Example Accelerator Manager

- **Accelerator Manager** provides SWC the functionality of requesting the execution of kernels in accelerators (e.g. GPU, DLA, …).
- **Dispatcher** selects jobs (kernels) from the queue of requests and launches them in the accelerators.
- **Vendor Abstraction** (device specific) abstracts the vendor runtime (drivers) to interact with the silicon.
- *Accelerator control* (requests and notifications) is handled by the **vendor runtime** via the vendor abstraction API.

At runtime, kernels are launched at their planned time. The global planning of resources guarantees that the required accelerators are available.
The accelerator manager monitors the progression.

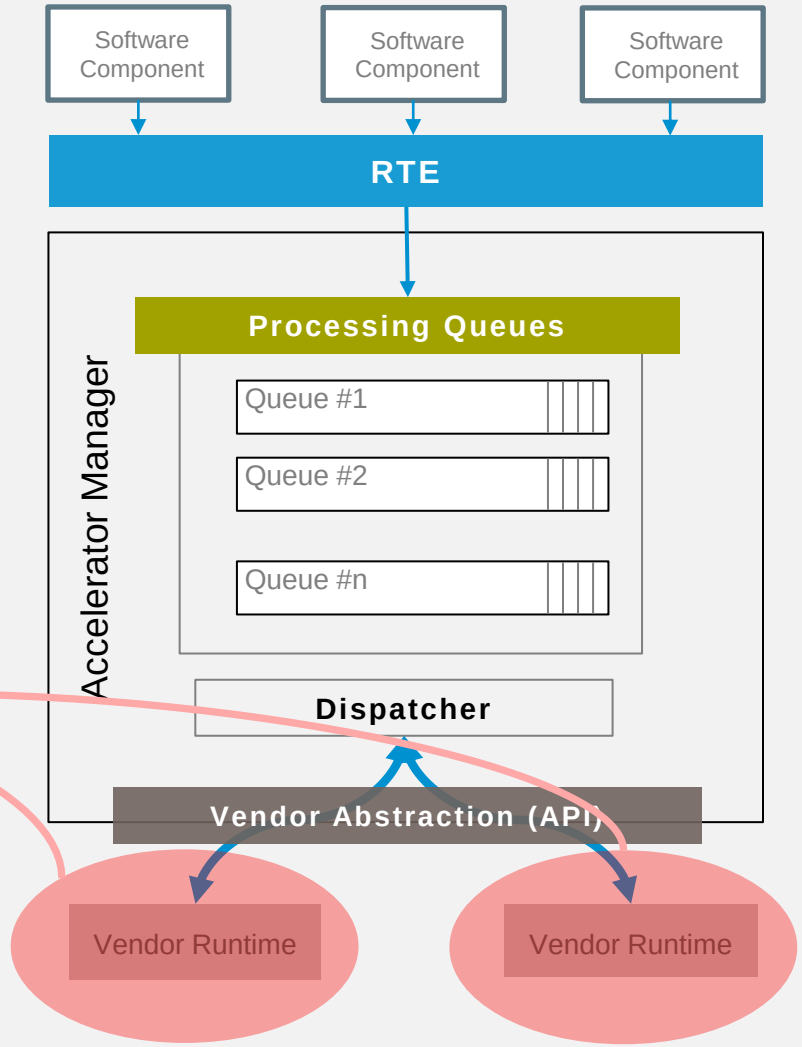


Example Accelerator Manager

- **Accelerator Manager** provides SWC the functionality of requesting the execution of kernels in accelerators (e.g. GPU, DLA, …).
- **Dispatcher** selects jobs (kernels) from the queue of requests and launches them in the accelerators.
- **Vendor Abstraction** (device specific) abstracts the vendor runtime (drivers) to interact with the silicon.
- *Accelerator control* (requests and notifications) is handled by the **vendor runtime** via the vendor abstraction API.

Upon being launched, the execution of kernels is handled by the vendor stack.

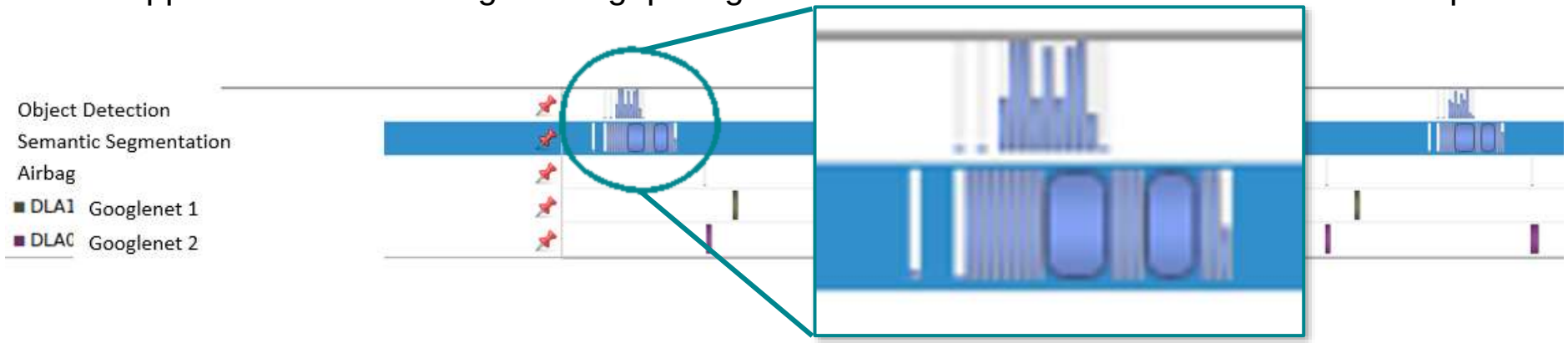
Drivers, API, documentation, etc… remain as provided by vendors.



Example Comparative Traces

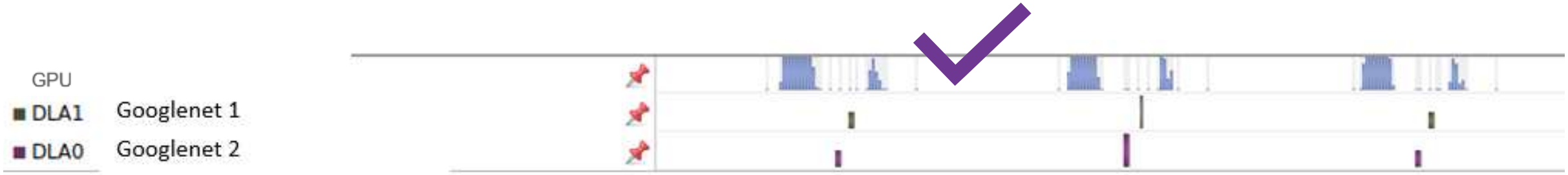
Without Accelerator Manager

- Vendor supplied drivers favor high-throughput against FFI. Interference is out of control and depends on runtime conditions.



With Accelerator Manager

- **Accelerator Manager** guarantees FFI by organizing the launch of kernels in non-interfering manner.



Take Aways

- Automotive systems CPS are growing in complexity.
  - AI enables functionalities otherwise not possible (e.g. perception).
  - AI hardware & software are crucial for L3+ autonomous driving.
  - COTS components focus on performance not dependability.
  - AI components operate as self-managed “black box(es)”.
- High failure-rate of AI requires additional considerations.
  - A system architecture that decomposes the system in nearly-independent subsystems each forming FCUs is necessary.
- Timeliness is a crucial factor to guarantee the system remains fail-operational.
  - The integration of AI components “out-of-the-box” is not suited for real-time.
  - Global system planning and management of accelerators help reduce uncertainty and increase time predictability.

TTTech

Thank you.

Ramon Serna Oliver
Principal Scientist

ramon.serna.oliver@tttech.com

Follow us on

LinkedIn

labs.tttech.com ↗


TTTech

TTTech