Autonomous-driving frameworks and predictability: challenges and open problems

Daniel Casini

Assistant Professor, Scuola Superiore Sant'Anna, Pisa, Italy

Invited Talk at Workshop CAPITAL 2022: sCalable And PrecIse Timing AnaLysis for multicore platforms, Friday, June 3th, 2022. Grenoble

Sant'Anna
Scuola Universitaria Superiore Pisa

Retis
Real-Time Systems Laboratory

This Talk

1

Introduction to Frameworks for Autonomous Driving and their challenges for time-predictability

- Apollo – based on Cyber RT
- Autoware – based on ROS 2

2

Apollo – contributions and open problems

3

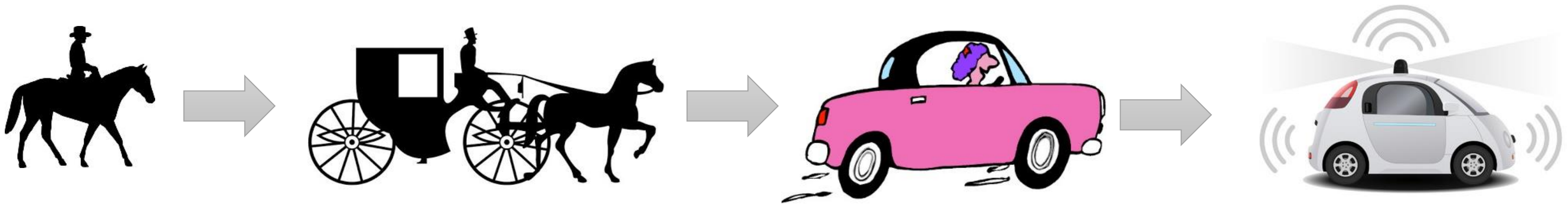
ROS 2 – contributions and open problems

2

Autonomous Driving

Retis
Real-Time Systems Laboratory

Autonomous driving is the next step in the history of human trasportation



Potential future benefits with respect to a human driver:

- ✓ More reliable driving
- ✓ Easier parking
- ✓ No unexperienced driver on the roads

Levels of Driving Automation

The Society of Autonomotive Engineers (SAE) identifies six levels of driving automation:

First 3 levels: The human driver is still the core of the driving system

AUTOMATION

LEVEL 0: **No driving automation**

- Vehicles are manually controlled (steering wheel, throttle, brakes, and everything)

LEVEL 1: **Driver Assistance**

- Cruise Control and Adaptive Cruise Control. The driver is assisted in basic operations such as steering or accelerating

LEVEL 2: **Partial Driving Automation**

- Advanced driver assistance systems (ADAS). Examples are Tesla Autopilot and Cadillac Super Cruise Systems. The vehicle controls steering, deceleration and acceleration. Automatic lane keeping. Some actions may be not supported by the autonomous car. **The driver needs to be ready to take the control at any time.**

Levels of Driving Automation

Real-Time Systems Laboratory

Last 3 levels: Increasing level of driving automation

A U T O M A T I O N

LEVEL 3: **Conditional Automation**

- Vehicles have enviromental detection capabilities, and drives autonomously. **The driver needs to take over when necessary.**

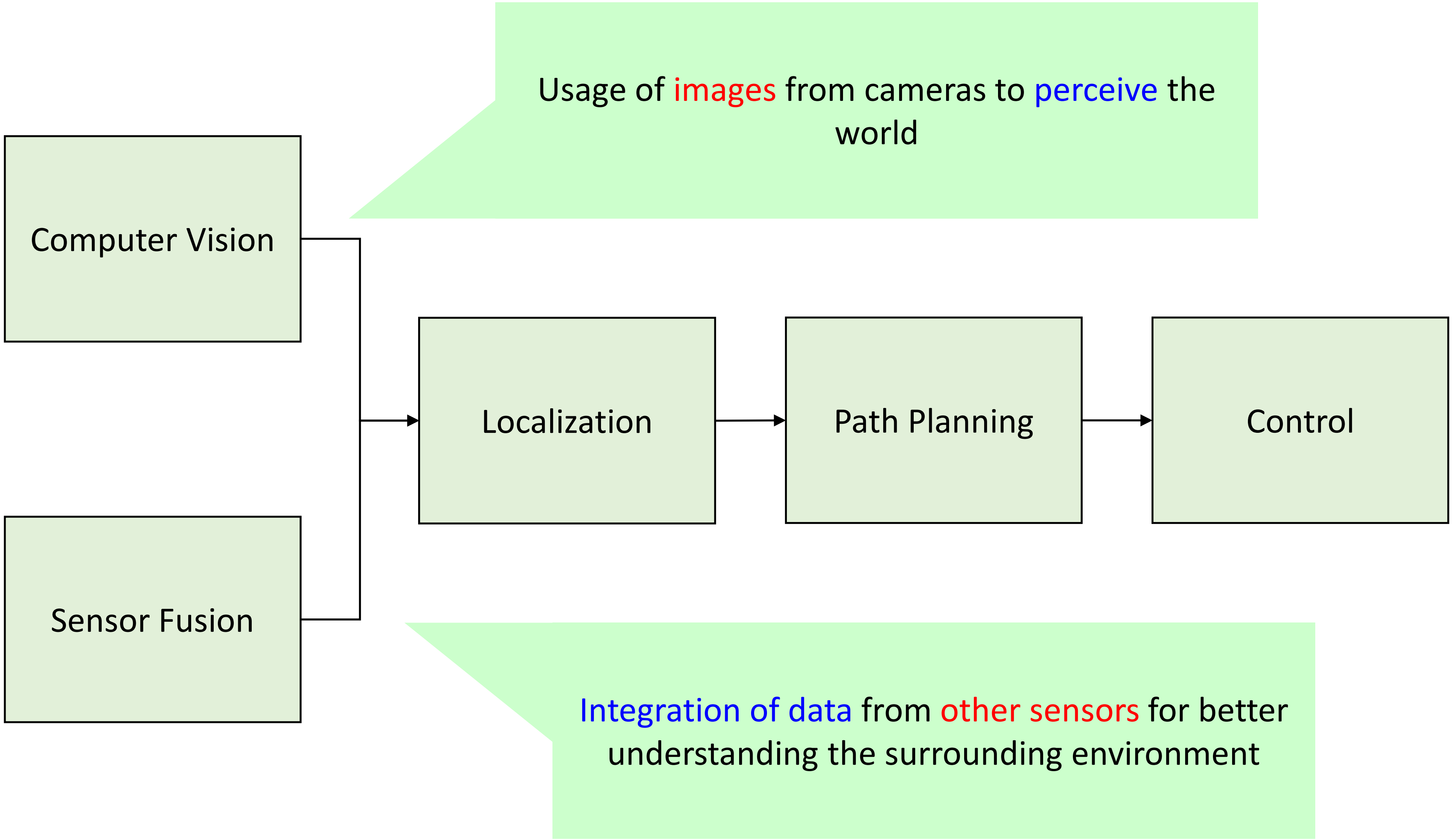
LEVEL 4: **High Automation**

- Vehicles controls every aspect of driving. The vehicle can intervene even if there is a system failure. **No expectation for a human intervention**, but the human has still the option to drive manually. There are legal limitations for the areas in which these vehicles can operate in self-driving (geofencing).

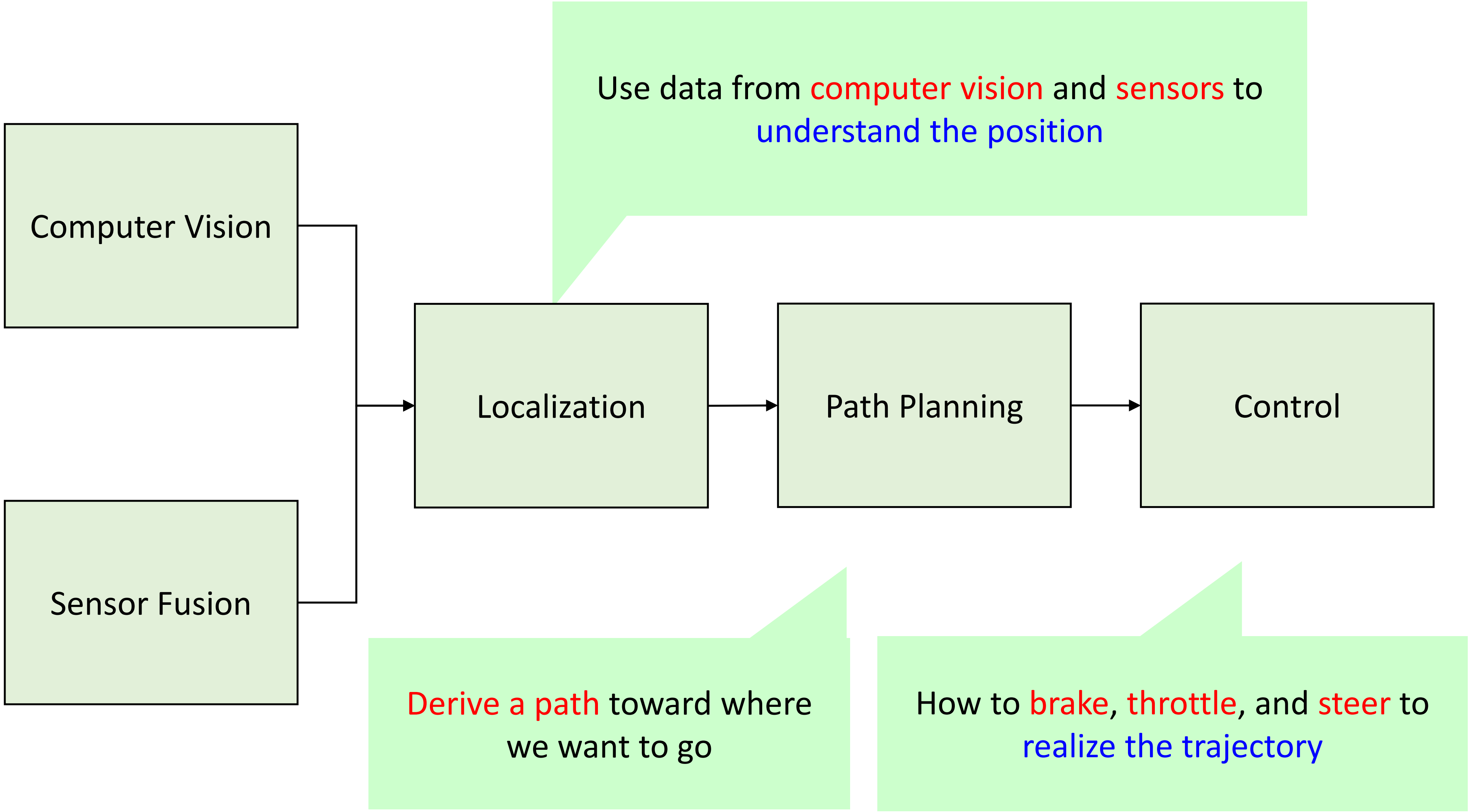
LEVEL 5: **Full Automation**

- The vehicles is autonomous, without any geographic limitation. No steering wheel or driving pedals!

How Self-Driving Car Works



How Self-Driving Car Works



Frameworks for Autonomous Driving

Two main open-source projects: Apollo and Autoware

Autoware

- Started in 2015 by Shinpei Kato at Nagoya University (Japan).
- 2300+ stars on GitHub and 500+ accounts on Slack (10/2018).
- More than 100 companies and runs on more than 30 vehicles in more than 20 different countries.

Baidu 百度 | apollo

- Started in 2017 by Baidu (China).
- The only company to obtain the first batch of road test licenses in China.
- Recently, started the Robotaxi project in China.

Autoware

AUTOWARE.AI

- The first Autoware project based on ROS 1
- Released as a research and development platform for autonomous driving

AUTOWARE.AUTO

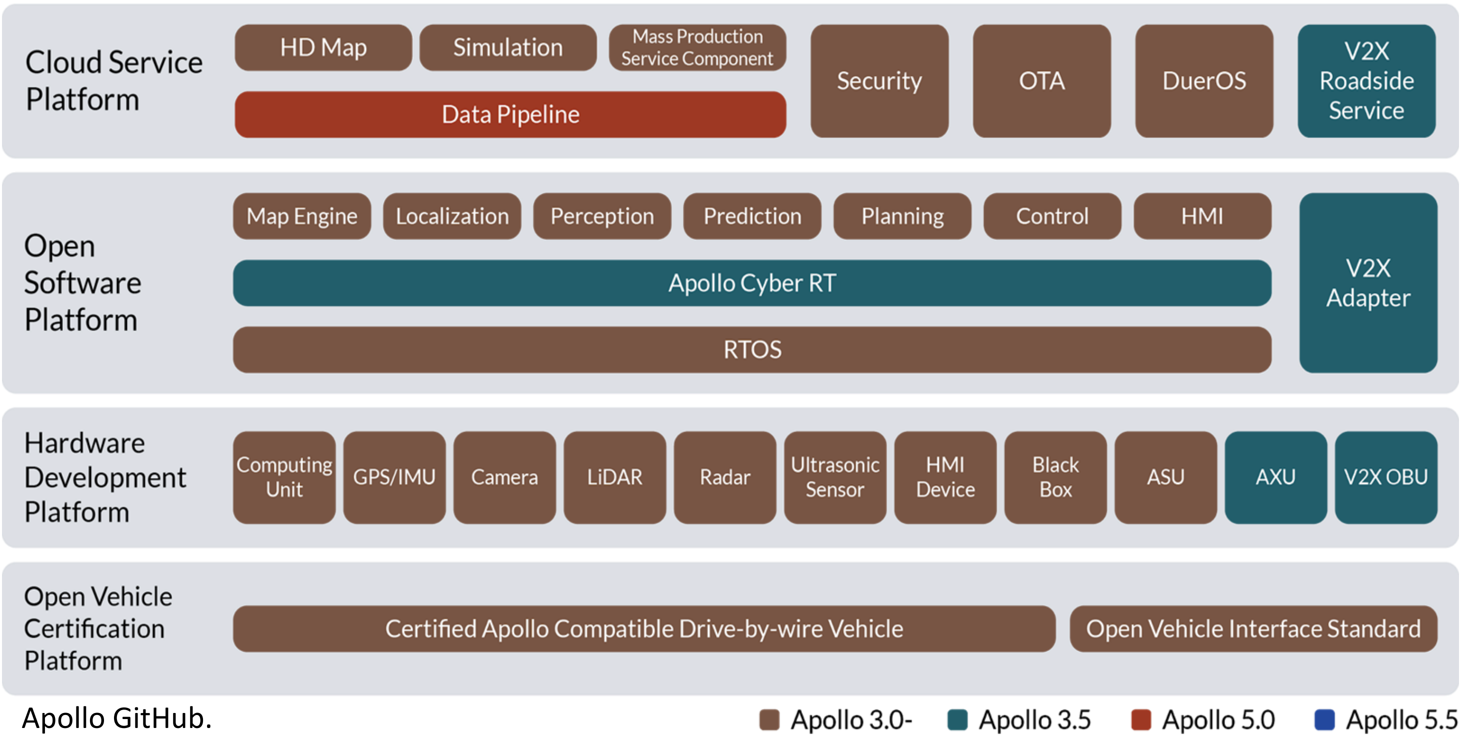
- Second version of Autoware, based on ROS 2
- Based on a redesigned architecture with better software engineering practices

The Apollo Autonomous Driving Framework

The Apollo Stack

➢ Four layers

1. Reference Vehicle Platform
2. Reference Hardware Platform
3. Open Software Platform
4. Cloud Service Platform

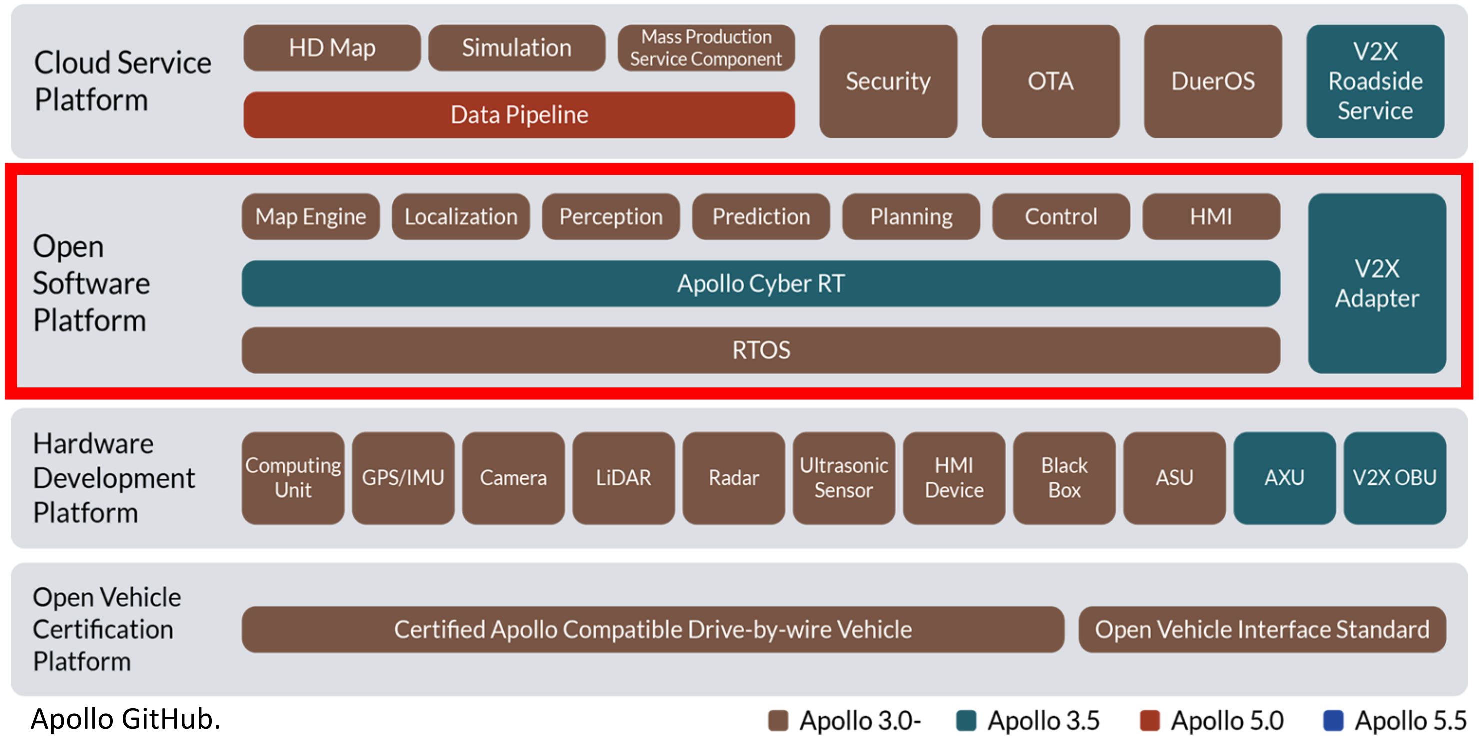


Apollo GitHub.

The Apollo Stack

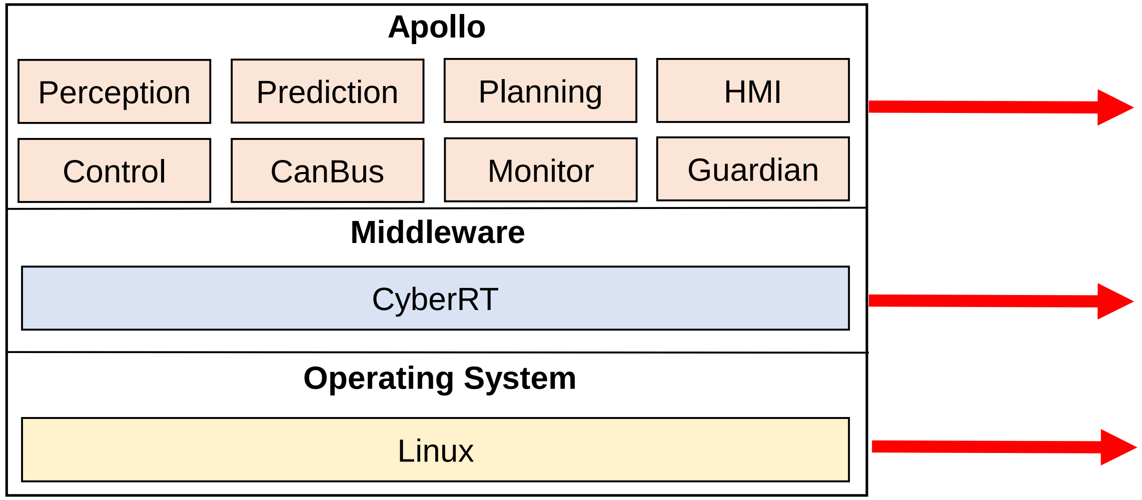
- **Four layers**

  1. Reference Vehicle Platform
  2. Reference Hardware Platform
  3. Open Software Platform
  4. Cloud Service Platform



Apollo GitHub.

Software Components



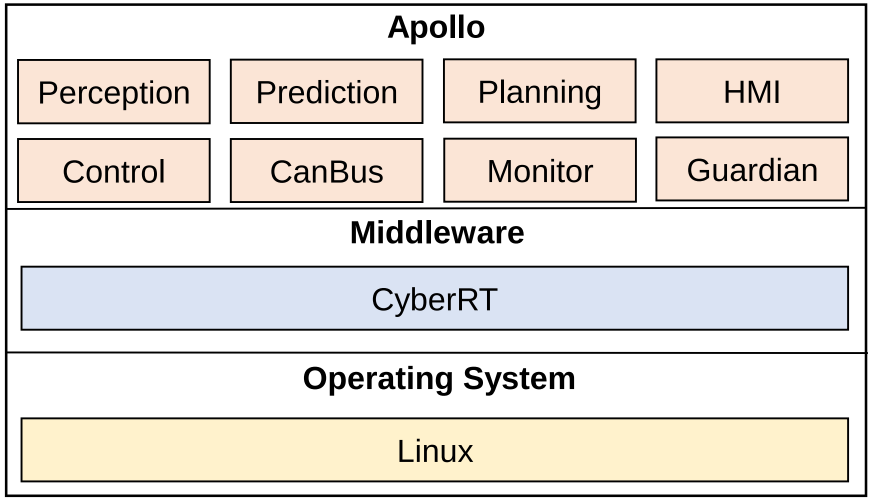
the core of Apollo

handles publish/subscribe communication

Linux equipped with PREEMPT_RT

Modules

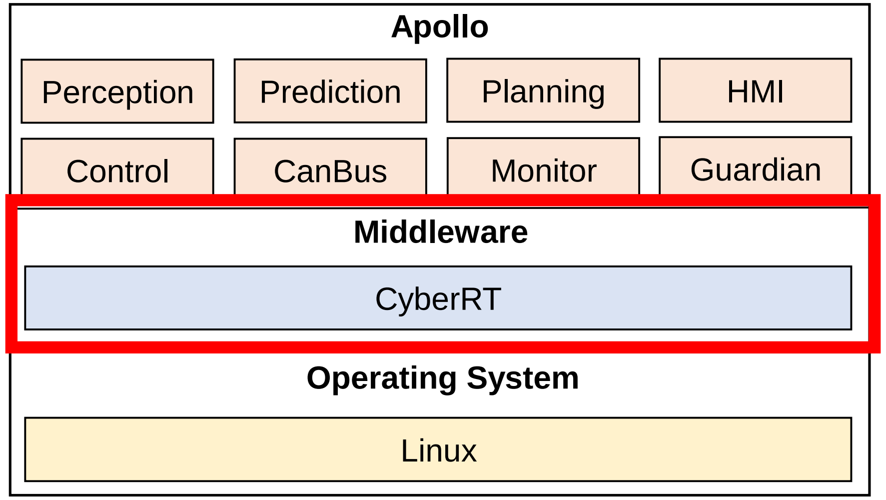
- **Localization:** to understand precisely where the car is in the world
- **Perception:** capability of detecting and classifying obstacles
- **Prediction:** studies and predicts the behavior of all the obstacles
- **Planning:** to plan a path towards the destination
- **Control:** actual commands to drive the car (how to turn the steering wheel, how much to hit the throttle, etc.)



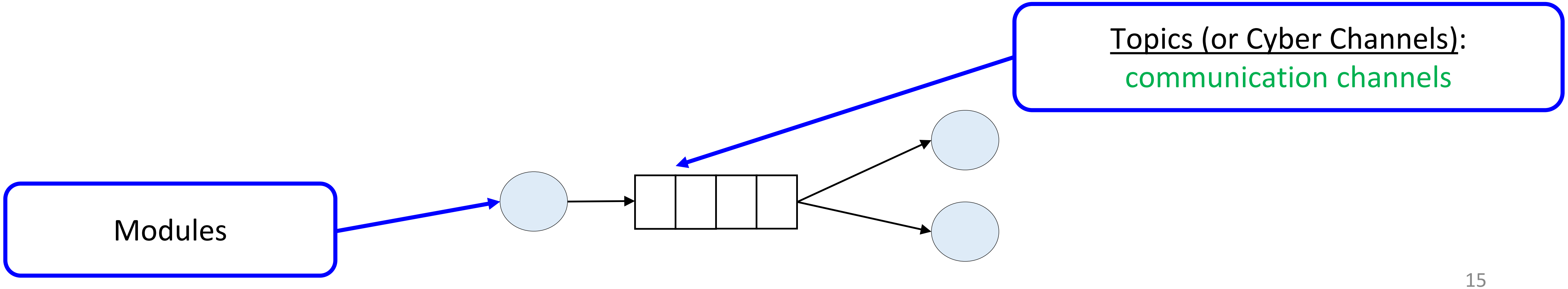
- **HMI:** is a module for viewing the status and controlling the functioning of the vehicle
- **CanBus**: handles communication through CAN
- **Monitor:** checks the system’s status and detects possible fault conditions
- **Guardian:** takes actions based on the monitor’s results

Middleware

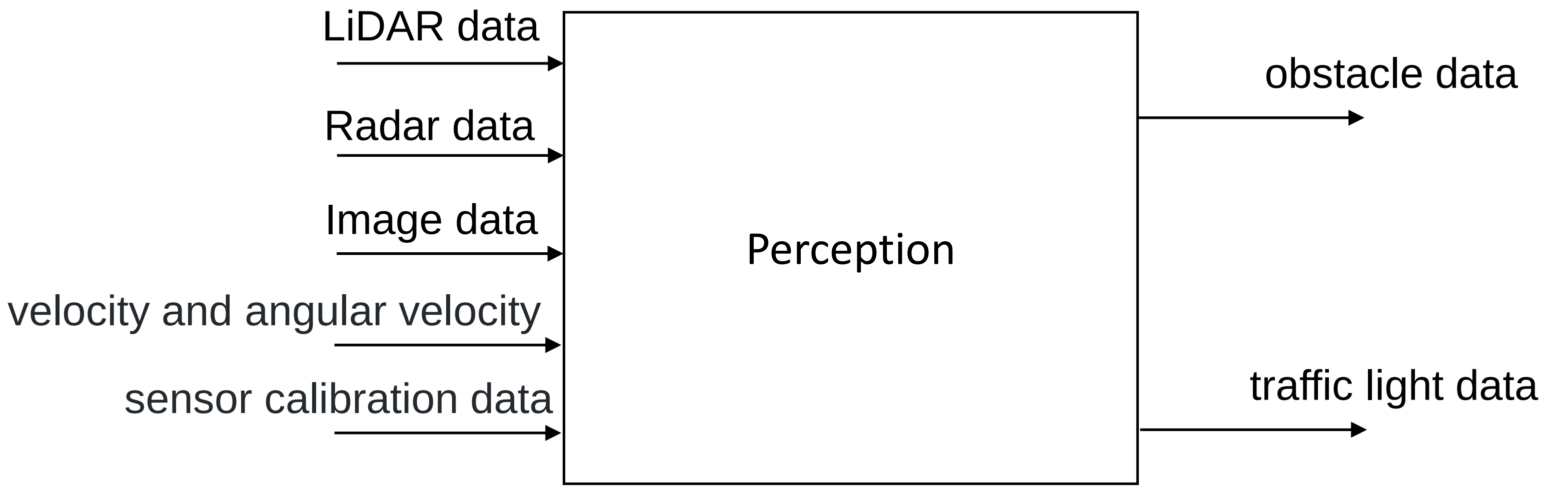
- This is the operating environment of Apollo
- Up to version 3.0, it was a customized version of ROS
  - The changes implemented by Apollo included optimization of shared-memory communication and the usage of protobuf instead of ROS message



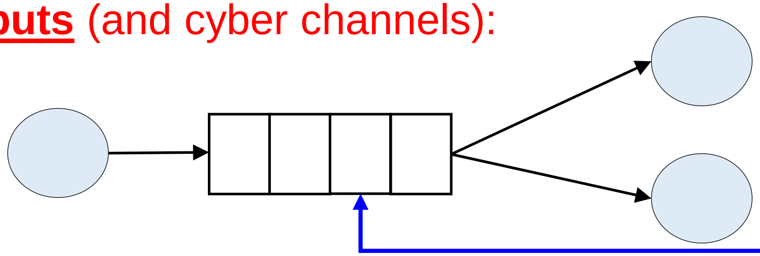
- From version 3.5 on, Apollo is adopting CyberRT
- Communication among modules is performed with a publish/subscribe paradigm



Topics in Apollo: an Example



Inputs (and cyber channels):

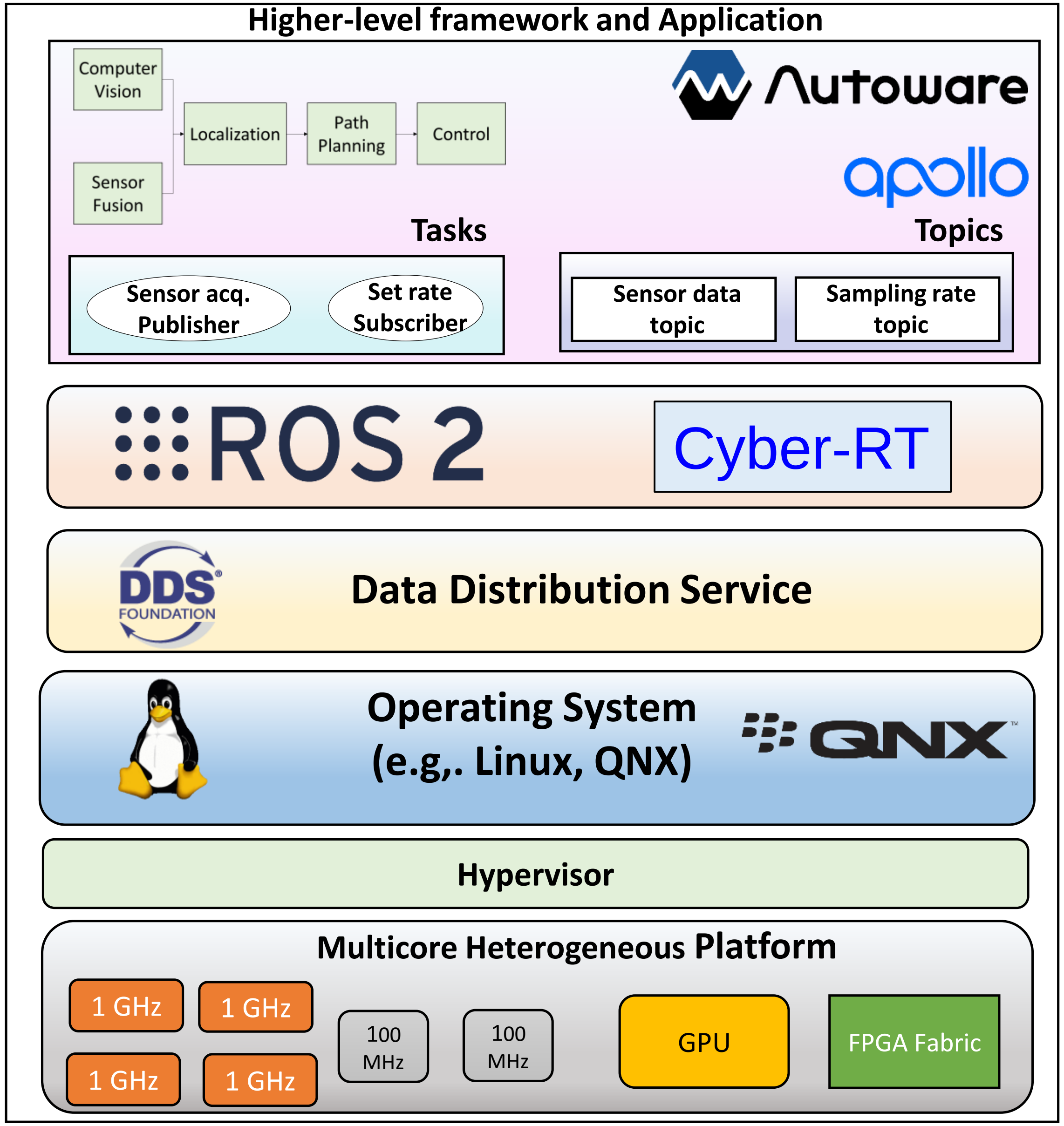


Cyber channel: communication channel

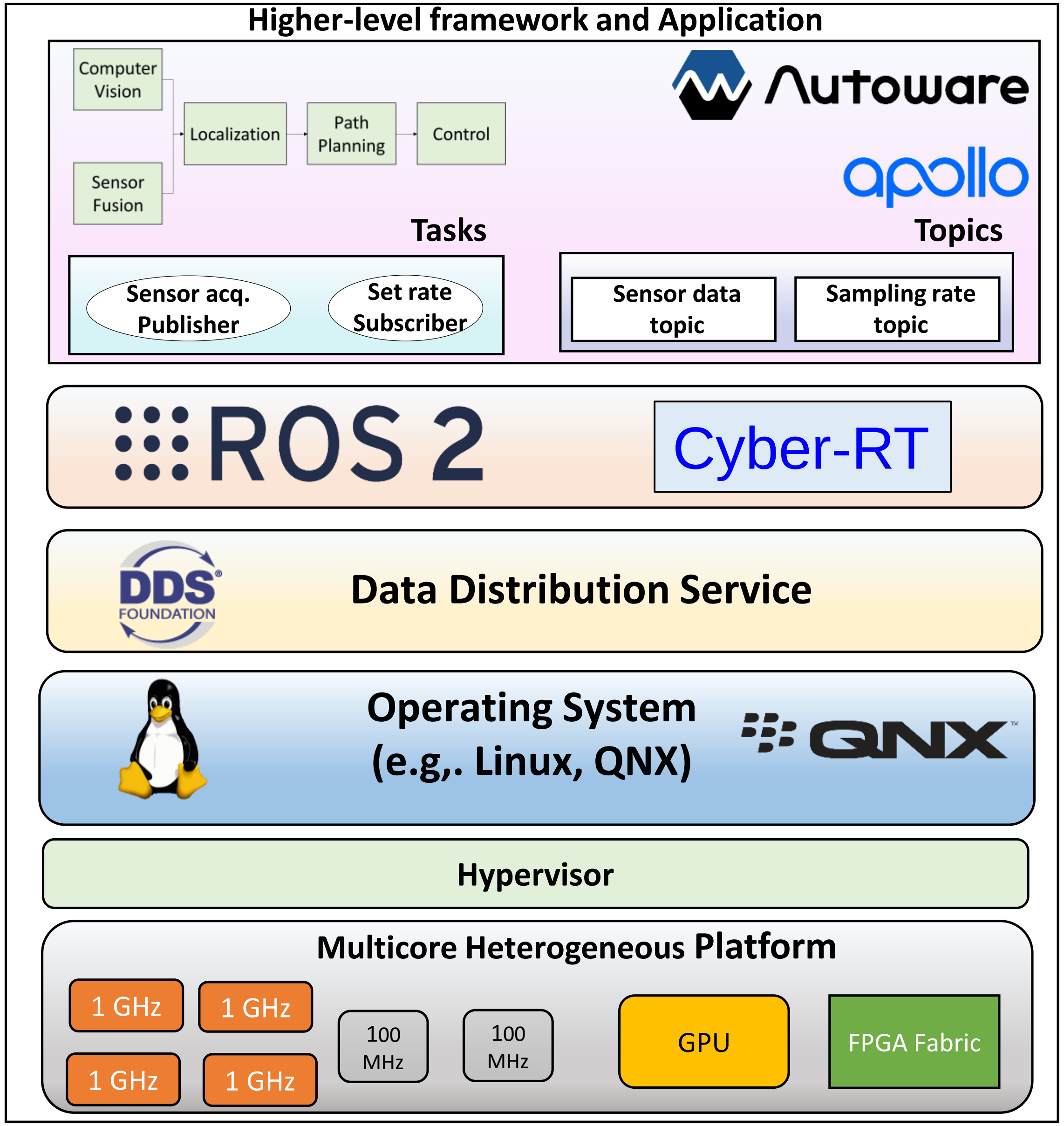
```
components {
  class_name : "RadarDetectionComponent"
  config {
    name : "FrontRadarDetection"
    config_file_path : "…/front_radar_component_conf.pb.txt"
    readers: [{channel: "/apollo/sensor/radar/front"}]
}}
```

Threats to Predictability and Challenges

Threats to Predictability and Challenges

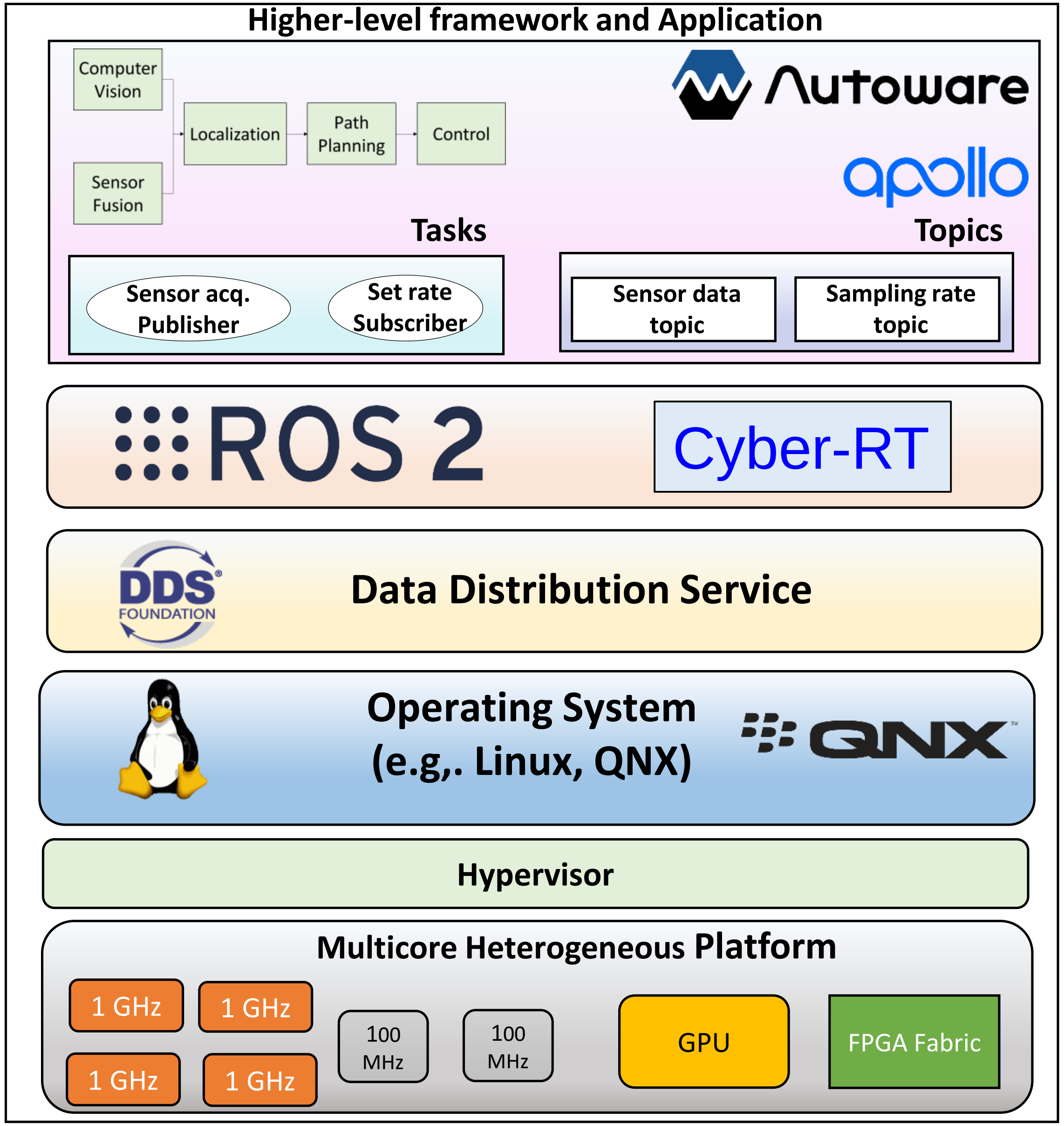


Threats to Predictability and Challenges



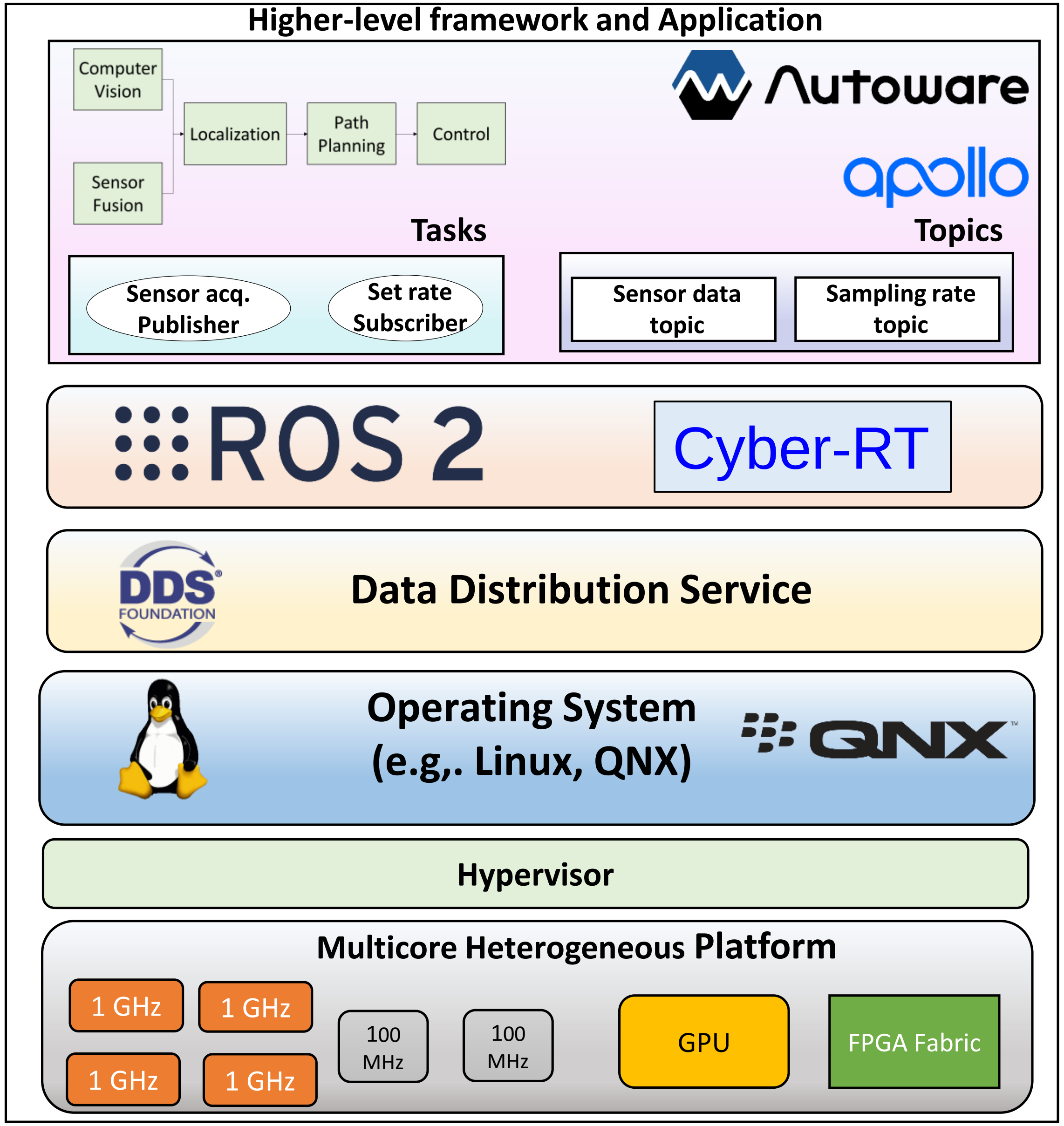
A very complex autonomous driving application that needs to satisfy timing constraints

Threats to Predictability and Challenges



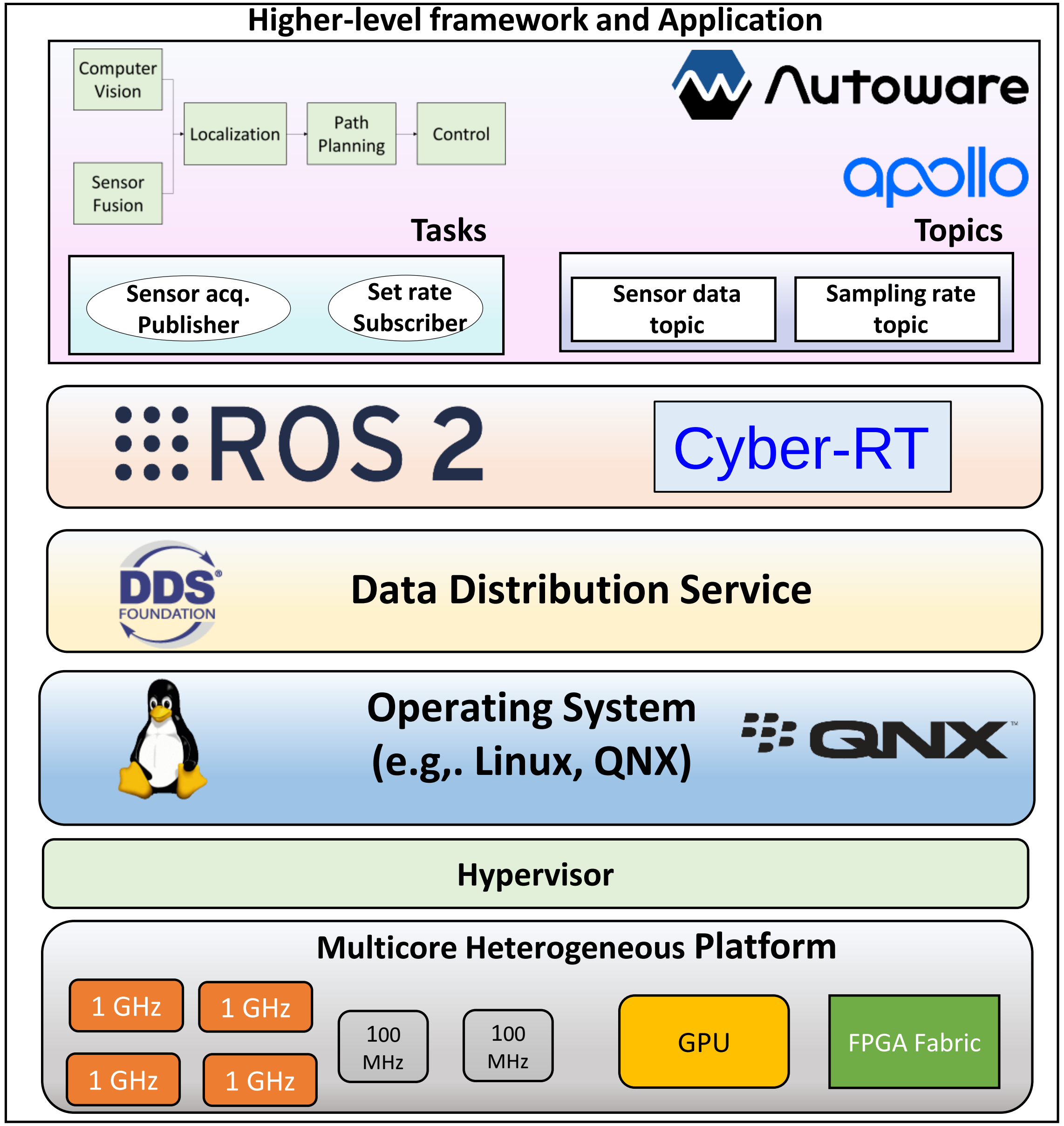
ROS 2 and CyberRT are used to conveniently implement autonomous driving, thanks to the pub/sub infrastructure and the large set of pre-implemented functionalities they provide.

Threats to Predictability and Challenges



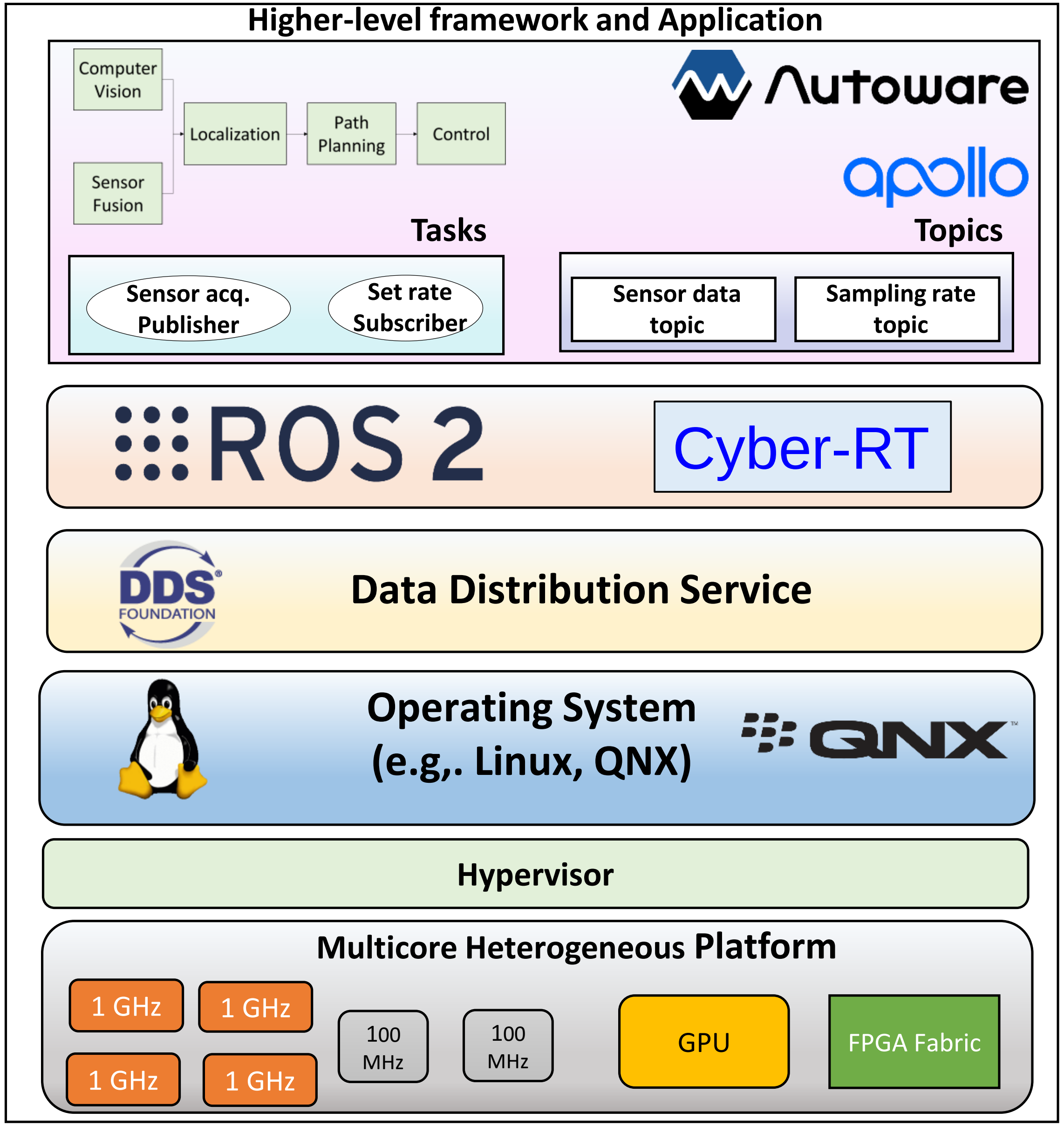
The DDS manages the communication among callbacks, with a complex multi-thread software that needs to be properly scheduled by the OS

Threats to Predictability and Challenges



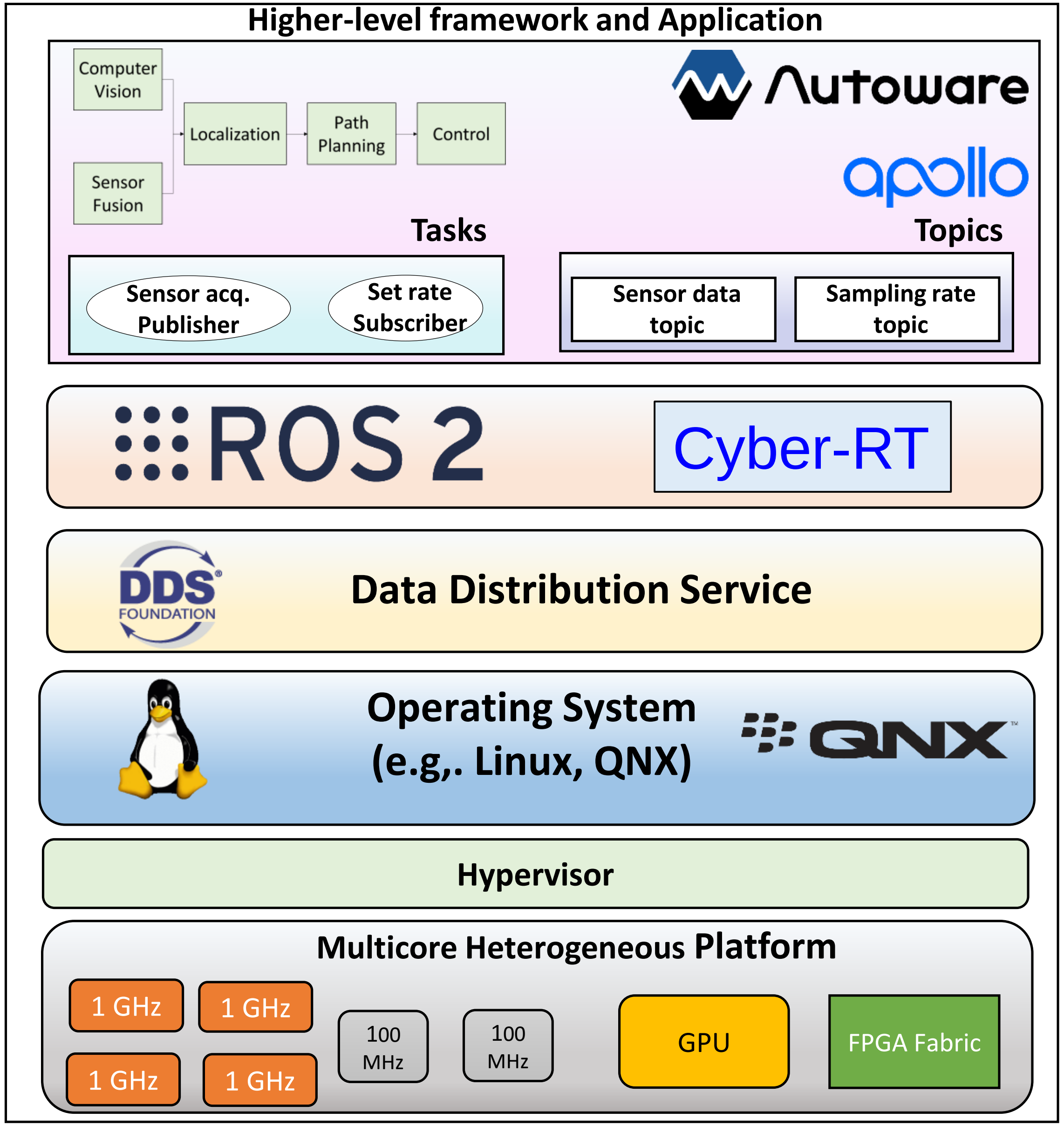
Application-level, ROS2-level, and DDS-level threads needs to be properly scheduled by the OS to meet timing constraints

Threats to Predictability and Challenges



A hypervisor, if present, can greatly help in implementing temporal and spatial isolation between virtual machines (VM), but it needs to be properly configured

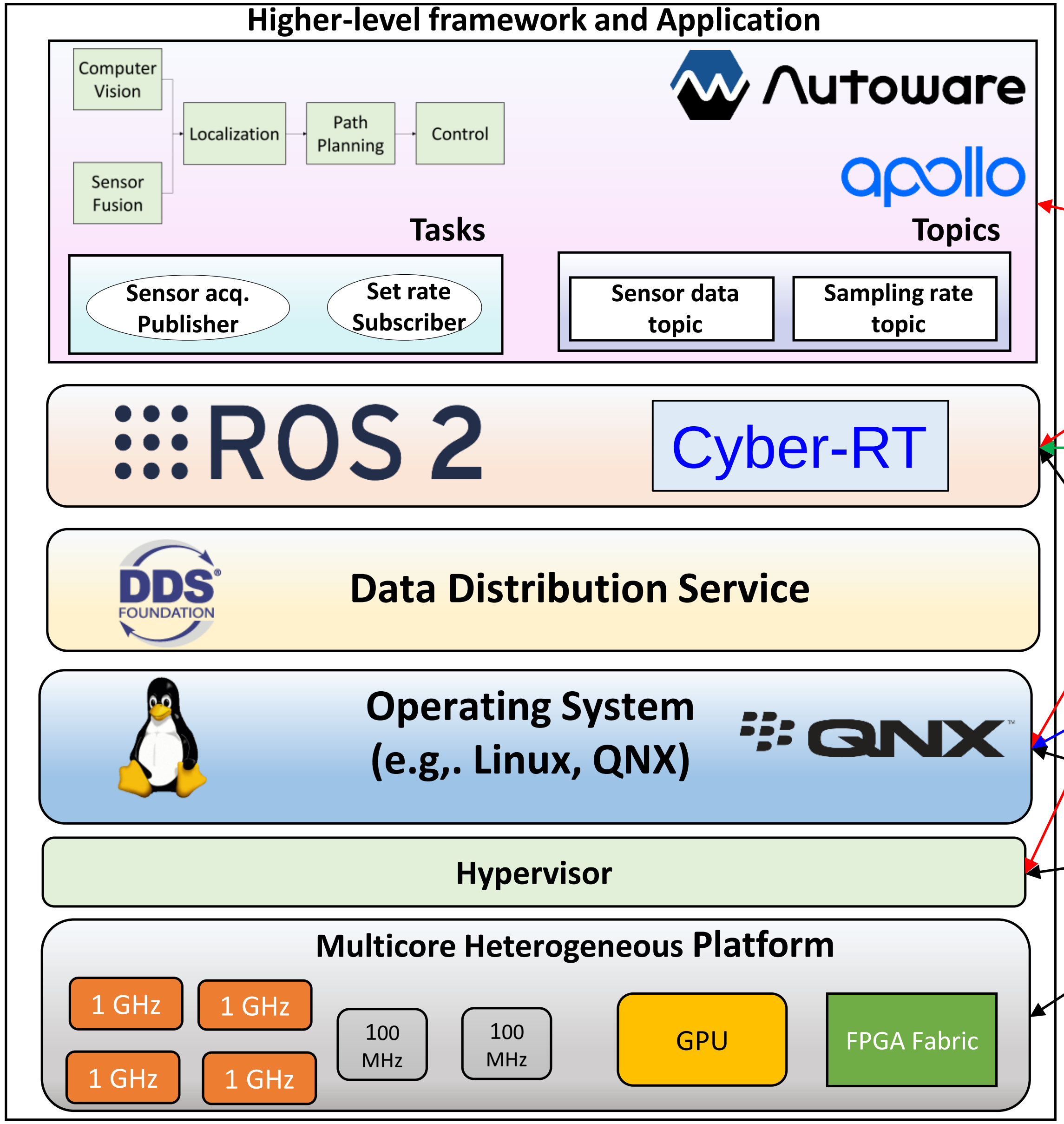
Threats to Predictability and Challenges



The underlying hardware platform can be very complex. Usually is equipped with heterogeneous cores and hardware accelerators and it has complex memory hierarchies, introducing further challenges

What we did to address these threats?

Several contributions:



This Talk

A Multi-Domain Architecture for Autonomous Driving (IEEE ESL, RTCSA 2020)

Response-Time Analysis of ROS 2 (ECRTS 2019, RTSS 2021)

Real-Time Analysis of the QNX Adaptive Partitioning Scheduler (RTAS 2022)

Many others, not covered in the presentation (timing predictability in Tensorflow, QoS regulators for mitigating I/O-related interference, I/O Virtualization, etc.)

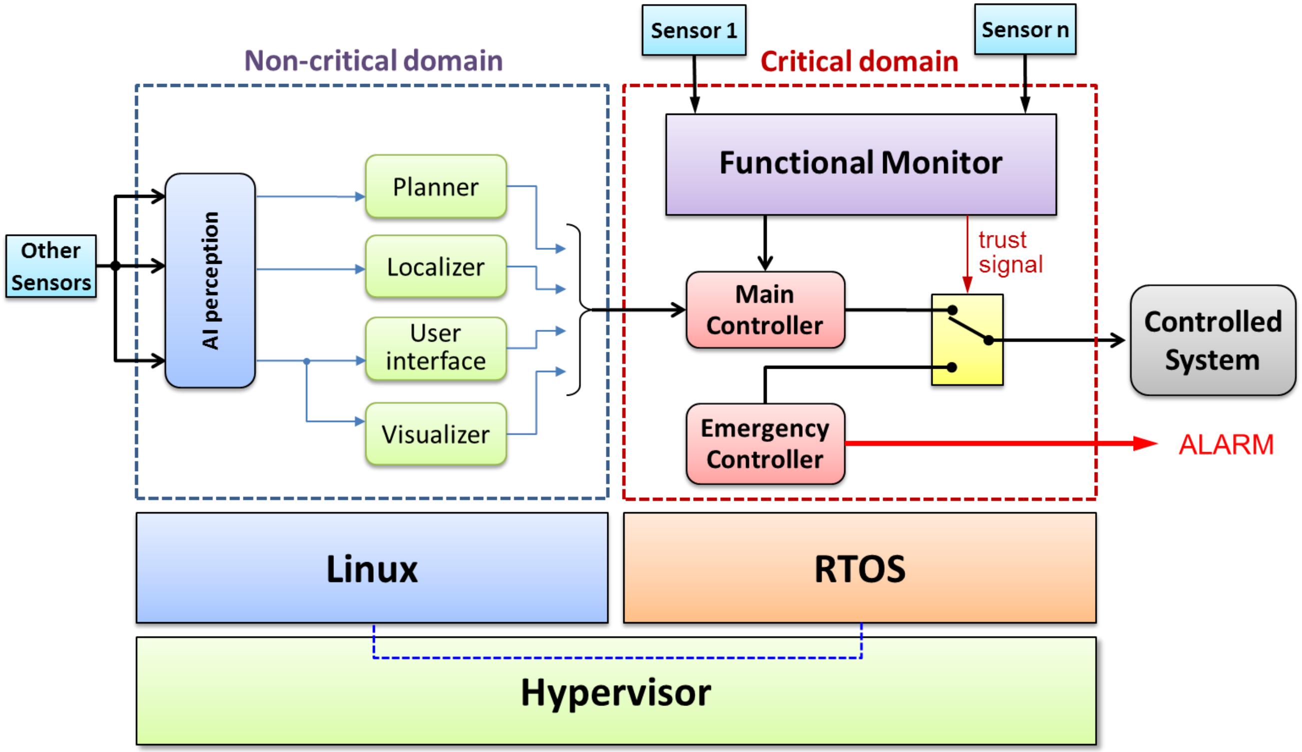
Multi-Domain Architecture

1) Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, **Daniel Casini**, and Giorgio Buttazzo, "A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems", *IEEE Embedded Systems Letters,*, vol. 12, no. 3, pp. 78-82, Sept. 2020.
2) Luca Belluardo, Andrea Stevanato, **Daniel Casini**, Giorgiomaria Cicero, Alessandro Biondi, and Giorgio Buttazzo, "A multi-domain software architecture for safe and secure autonomous driving", *Proc. of the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2021)* , August 18-20, 2021.

Multi-Domain Designs

- A multi-domain architecture used to switch the system to a ‘safe controller’ running on Erika that uses minimal perception features based on legacy sensors (e.g., radars), extending what already done by the Apollo Guardian component

- An alarm signal can be sent to the driver to take back the control, as mandated by **SAE's Level 3 specifications** (Conditional Driving Automation, a.k.a., "eyes off")



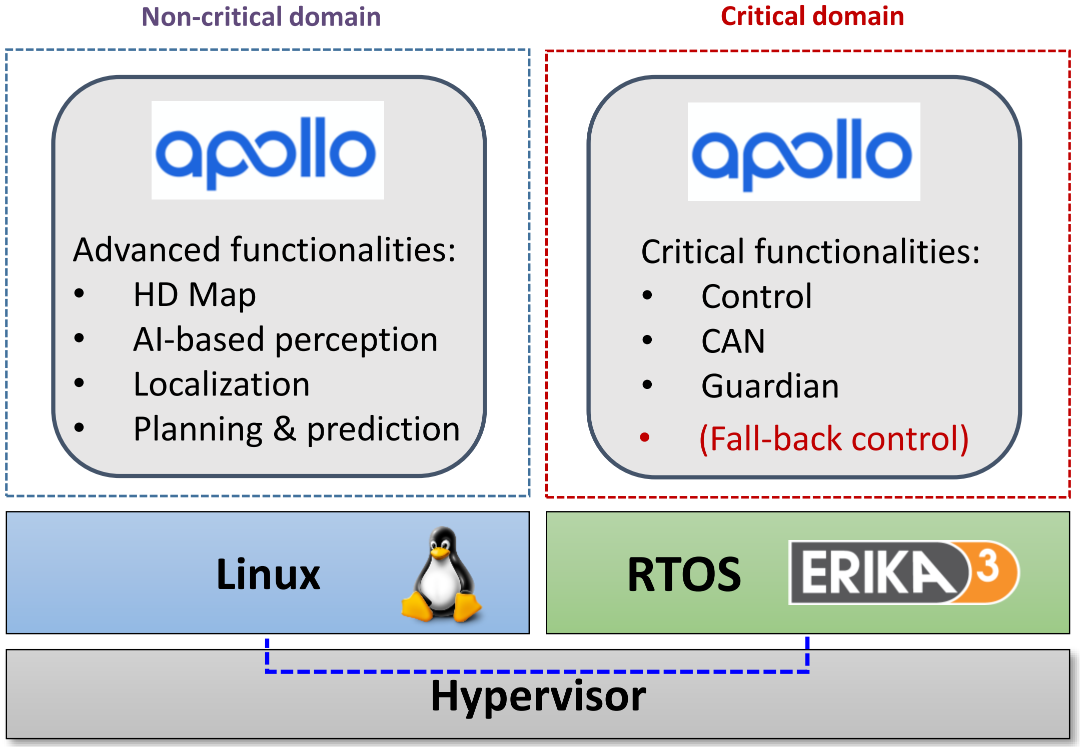
Multi-Domain Architecture for Apollo

OBJECTIVE

Making Apollo safer and more secure by designing a multi-domain architecture

HOW?

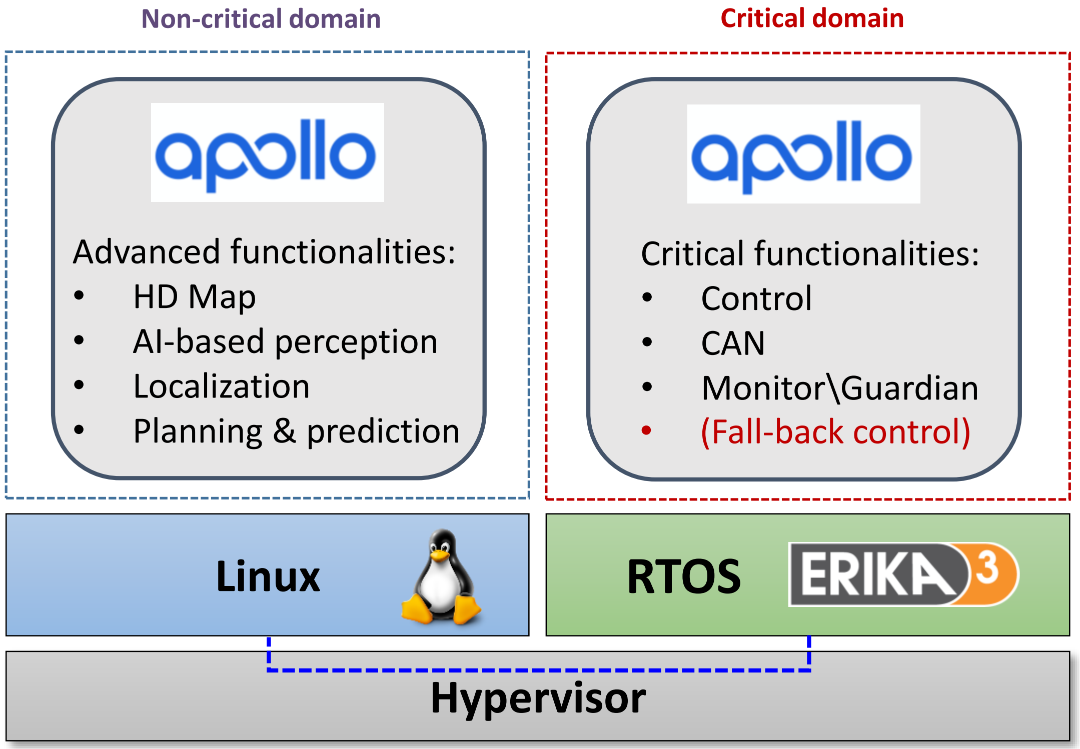
1. Separating Apollo’s modules between a non-critical and a critical domain, running different OSes
2. Using a hypervisor to separate them
3. Restoring the pub/sub CyberRT-based communication



Luca Belluardo, Andrea Stevanato, **Daniel Casini**, Giorgiomaria Cicero, Alessandro Biondi, and Giorgio Buttazzo, "A multi-domain software architecture for safe and secure autonomous driving", *Proc. of the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2021)* , August 18-20, 2021.

Target multi-domain design

- The multi-domain design provides all critical functionalities of Apollo in a <u>separate domain</u> running Erika3.
- Advanced Apollo functionalities (e.g., those requiring drivers of software stacks that not available in a RTOS) on a Linux domain and the CyberRT-based communication (based on the pub/sub paradigm) with those moved to the **Erika3** domain must be restored.
- The components moved to the Erika3 domain are replaced by “bridging” components in the Linux domain, i.e., they act as a bridge to control inter-domain communication
- A fall-back controller (not present in Apollo) may be realized in the future to perform minimal safe control without the advanced Apollo functionalities



Prototype Implementation

HW-in-the-loop Simulation

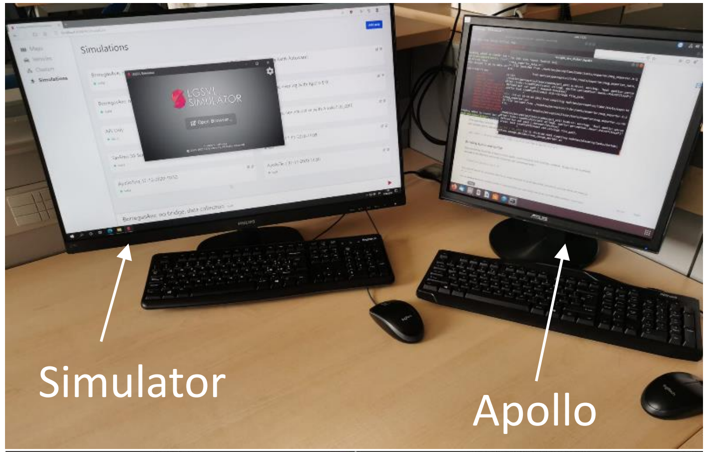
- **HW-in-the-loop simulation environment** to work with Apollo

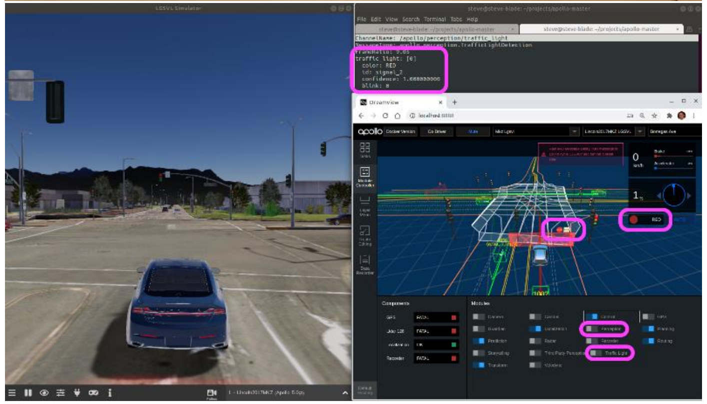
X86 machine with Linux to run Apollo

Gigabit Ethernet

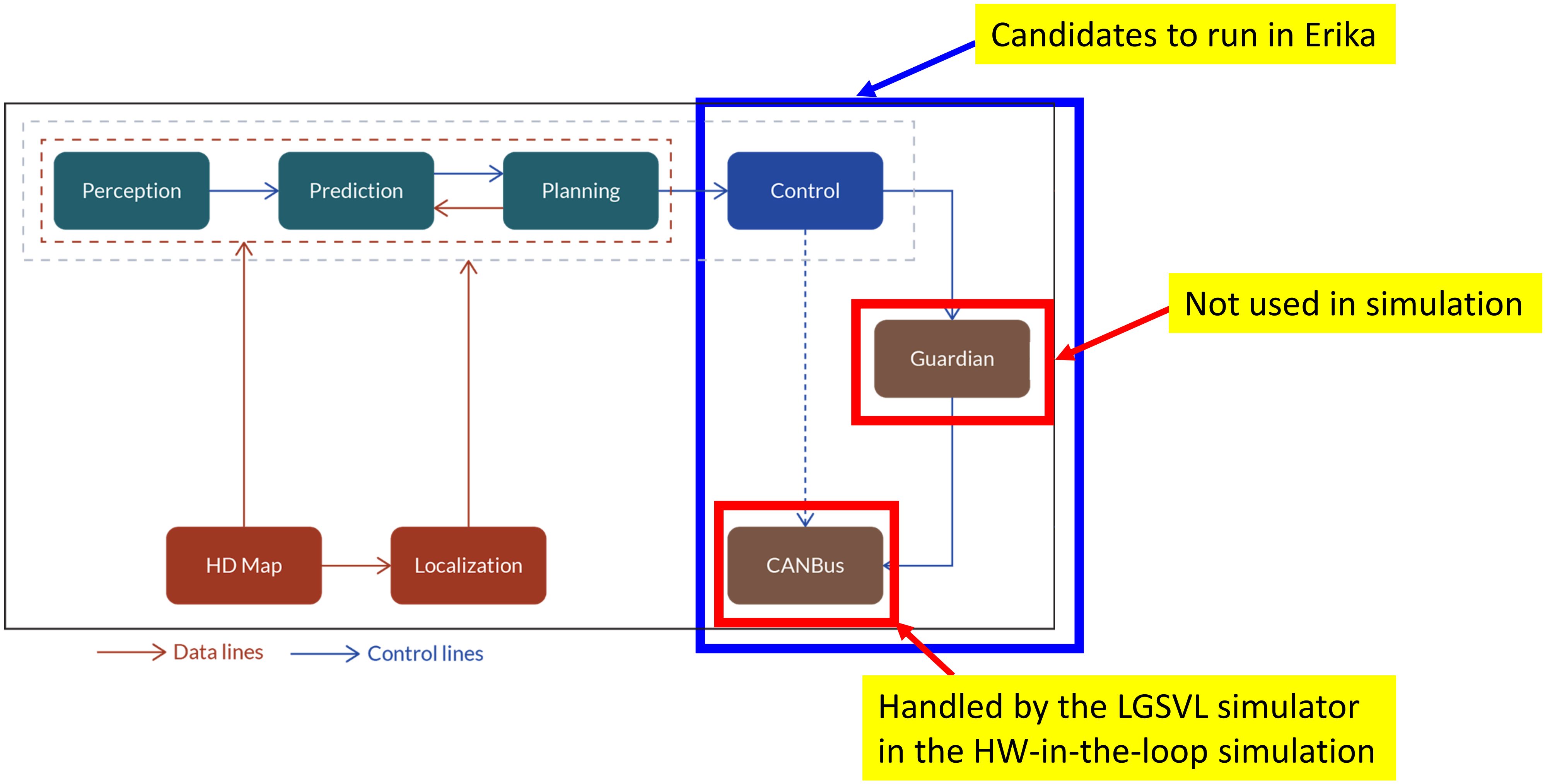
LGSVL Simulator
(https://www.lgsvlsimulator.com/)
on a powerful desktop computer

4 GHz Quad-core CPU, 32GB of RAM,
Nvidia GTX 2080 16GB, Win 10 64-bit





How to split components among VMs?



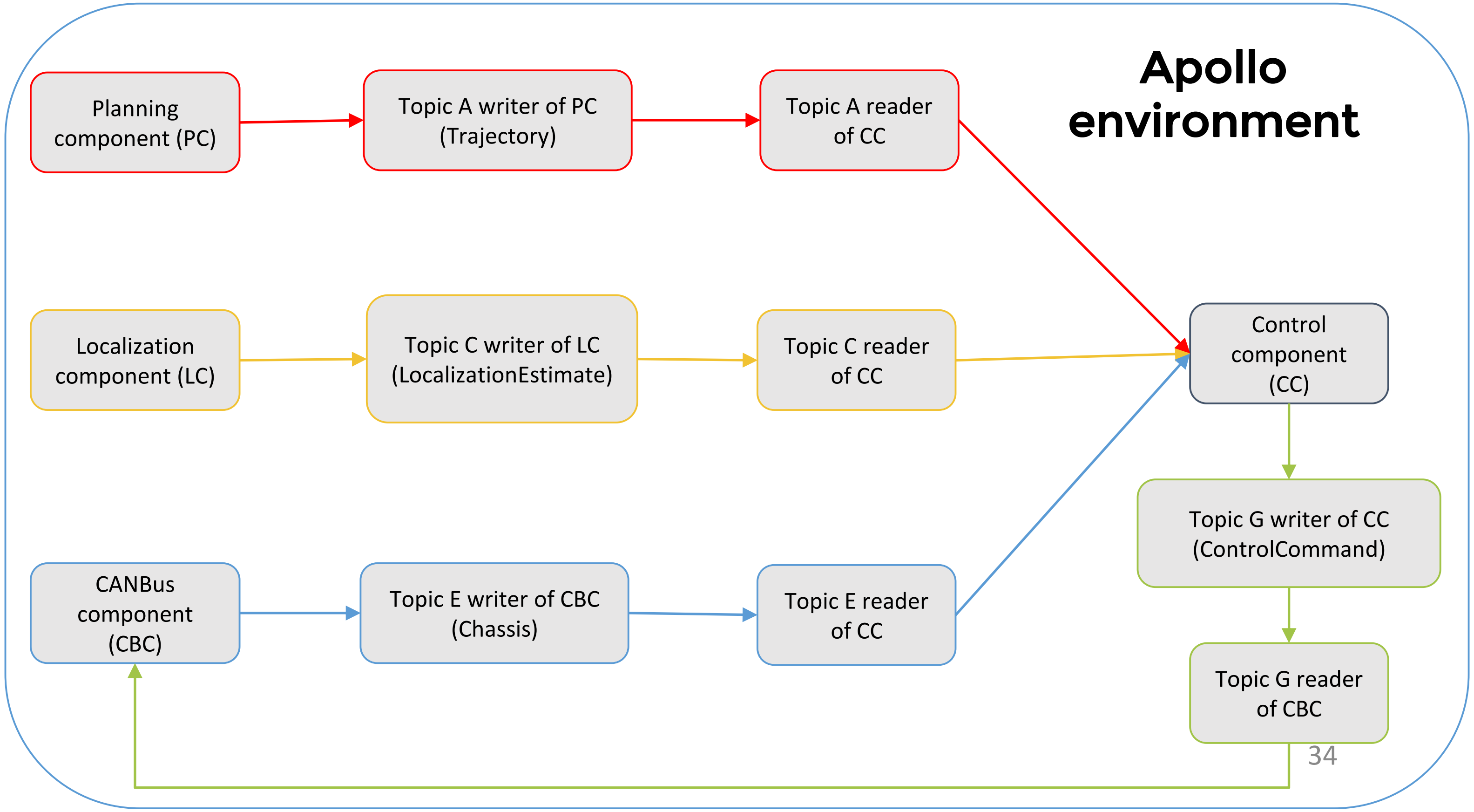
- In our implementation, we moved the **control component** from Linux to Erika

Why the Control Component?

- It is a highly safety-critical component;

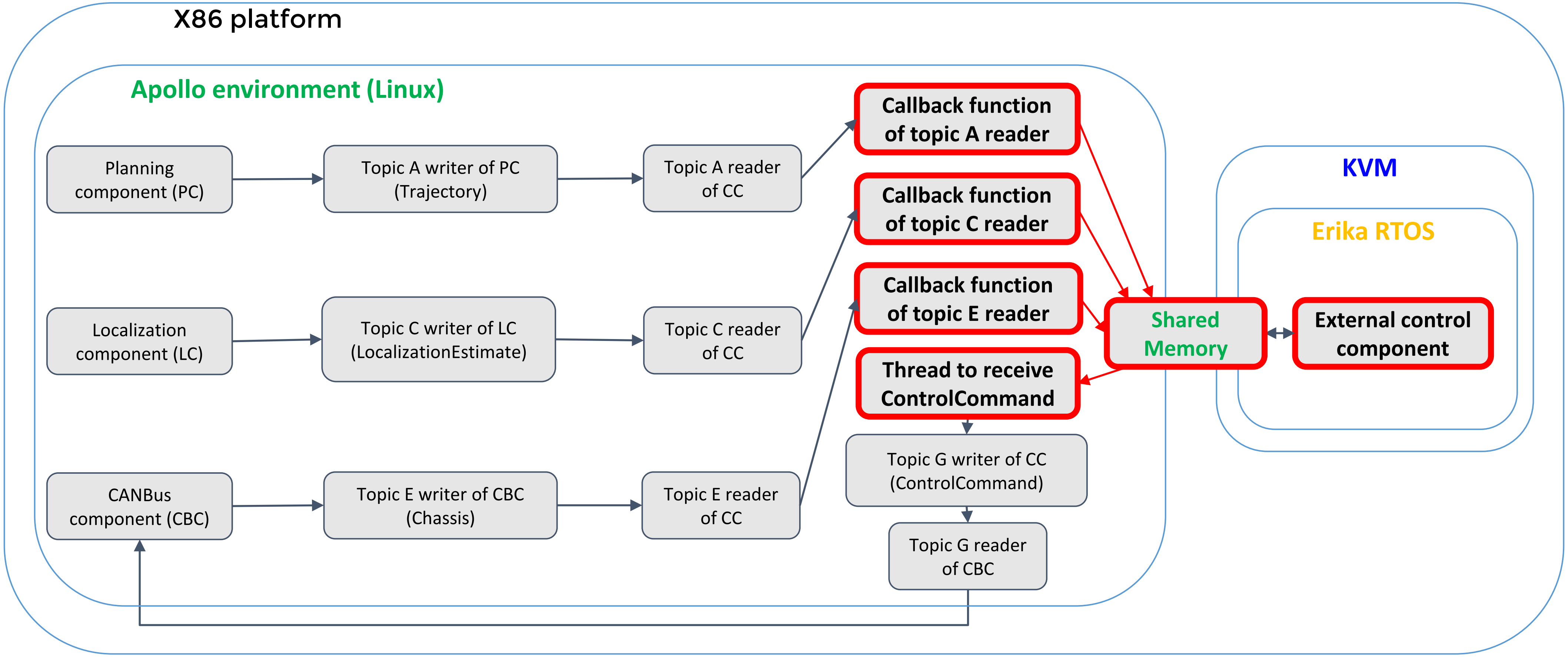
- It <u>does not</u> require complex software stacks or device drivers that are <u>not available</u> on Erika (e.g., for interacting with NVIDIA GPUs, as the perception component does)

- The Control Component is characterized by the following communication relationships (showed in the picture and extracted during our in-depth analysis of the Apollo code).



The Control Component in Erika

- Apollo officially supports **Intel x86** platforms only.
- Realized a multi-domain design on Intel x86 using **KVM** and Erika3 for x86.



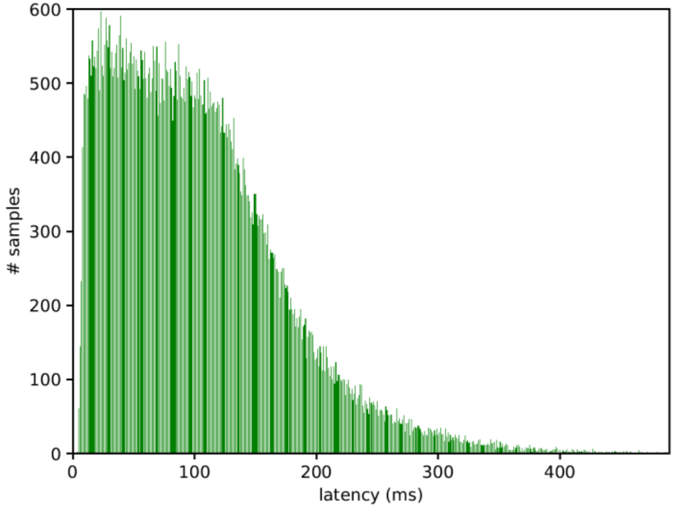
Control Component Dependencies

- The following libraries have been ported to Erika:
  1. **qpOASES**: it is an open-source C++ implementation of the Online Active Set Strategy using the quadratic programming. It is used by the controller to compute the control commands; (see https://github.com/coin-or/qpOASES)
  2. **gflags**: it is a library to manage command-line flags in applications with multiple files. (see https://gflags.github.io/gflags/#intro)
  3. **protobuf:** the library to standardize the exchange messages between components. (see https://developers.google.com/protocol-buffers)
  4. A **POSIX-FatFS wrapper** to avoid modifying the Apollo code for accessing the file system.

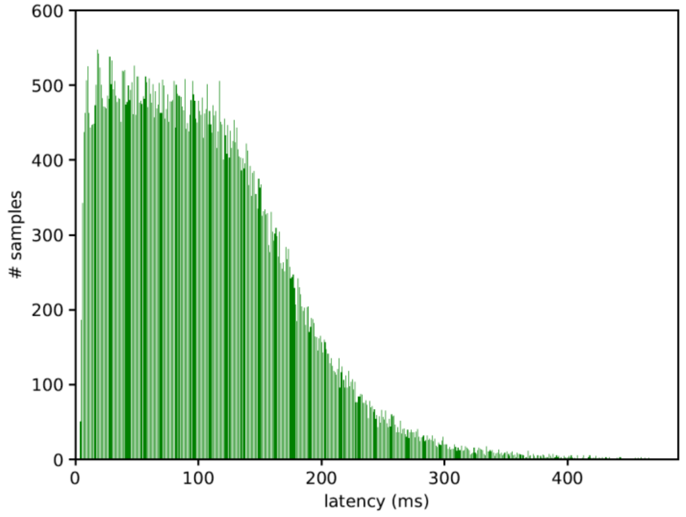
Evaluation

Latency of the planning messages

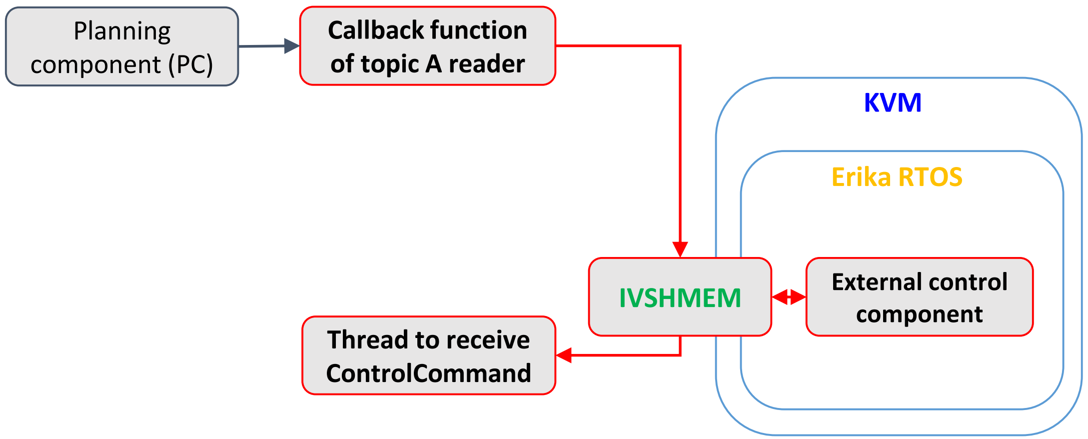
Standard version



Multi-domain version

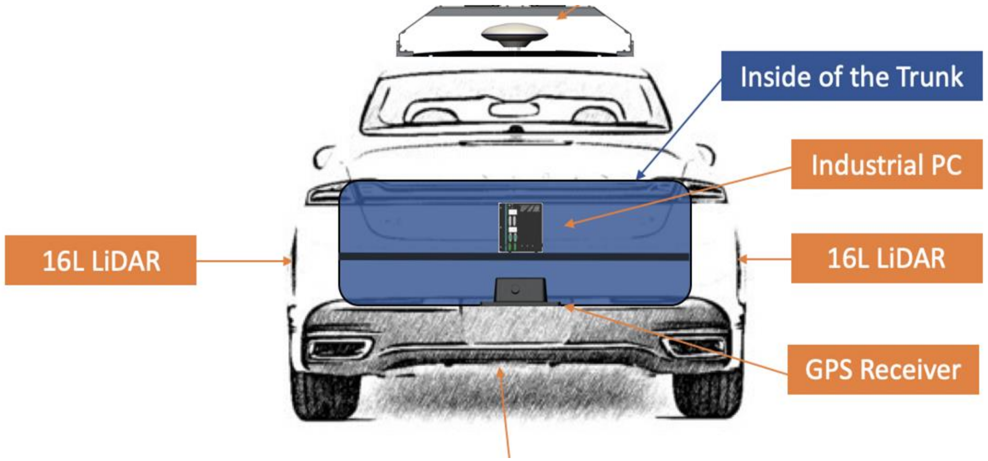


Similar latencies have been observed



Other works with Apollo at the RETIS lab

Porting Apollo on an Embedded Platform



Rear view. Apollo GitHub.

4 GHz Quad-core CPU,
32GB of RAM,
Nvidia GTX 2080 16GB,
Win 10 64-bit

X86 machine with Linux to run Apollo

- How to replace it with one (or more) embedded platforms?

- **Target**: Xilinx Ultrascale+ FPGA-based SoC, high predictability, low power consumption

apollo::planning::PlanningComponent::Proc

	zcu_mod_testing	pc_mod_testing	Ratio
Average	906.26	105.42	8.6
Max	3900	216	18.06

Very challenging problem due to huge slow down due to slower cores

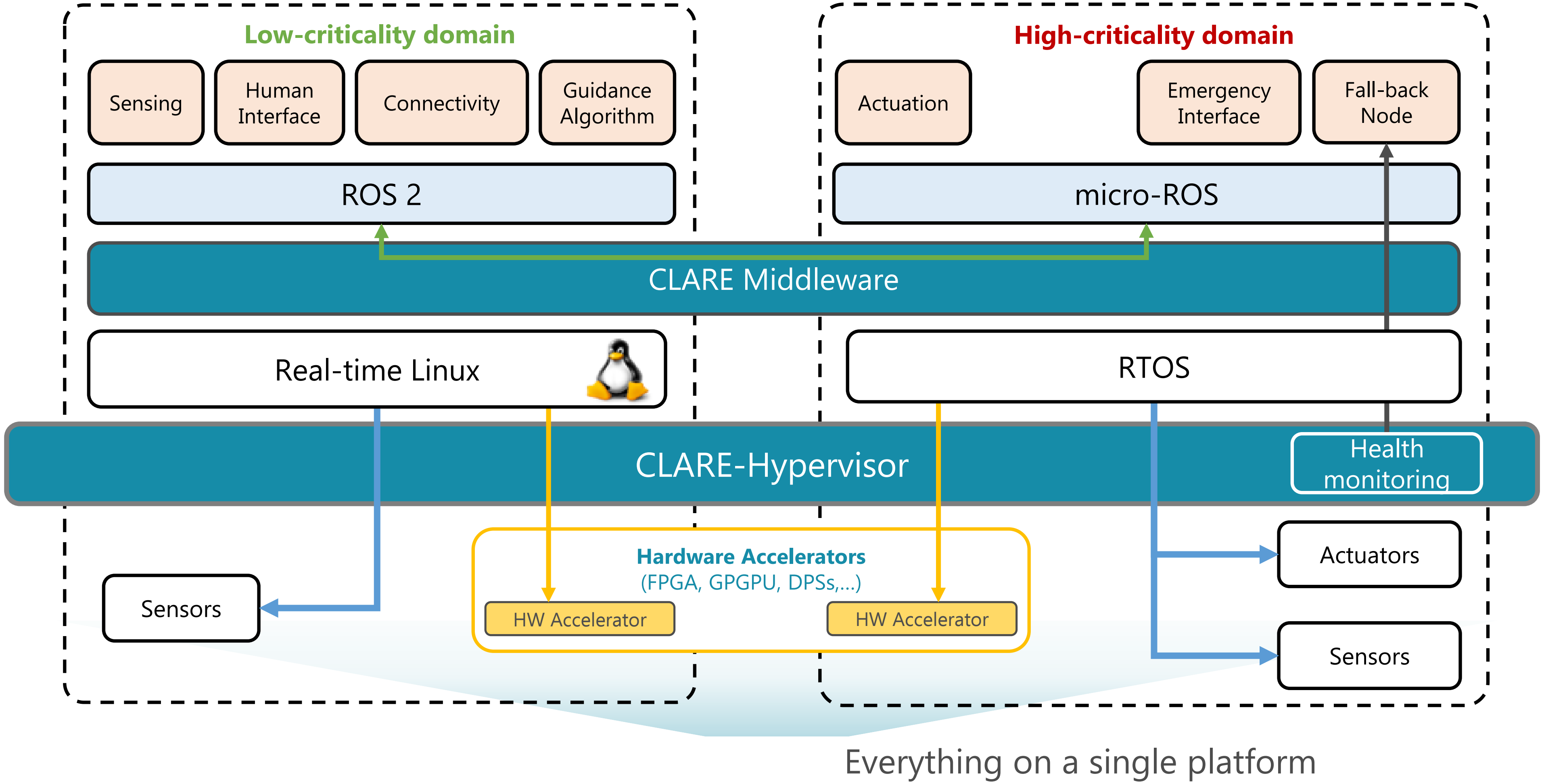
Optimized Acceleration of DNNs

- **Optimized acceleration** of the Apollo Deep Neural Networks for deployment on embedded platforms
- Quantization & network pruning
  - **Current target**: Xilinx Ultrascale+ FPGA-based SoC, high predictability, low power consumption

Apollo Deep Neural Networks
Denseline lane detector
DarkSCNN
Traffic Light Detection
Traffic Light Recognition
Yolo - Object Detection
Lidar (Velodyne 16)

Inter-domain communication

- Transparent, shared-memory communication between ROS 2, running in a Linux-based VM and micro-ROS, running on FreeRTOS
  - Based on the CLARE hypervisor, developed by Accelerat, a spin-off company of the Scuola Superiore Sant'Anna



Real-Time Systems Laboratory

Apollo – Open Problems

- Analysis of processing chains under CyberRT
- Fail-safe controller and safe perception module
- Multi-domain architecture with a type-1 hypervisor on an embedded platform
- And many more, e.g., on the AI and hardware acceleration side

Response-Time Analysis of Processing Chains in ROS 2

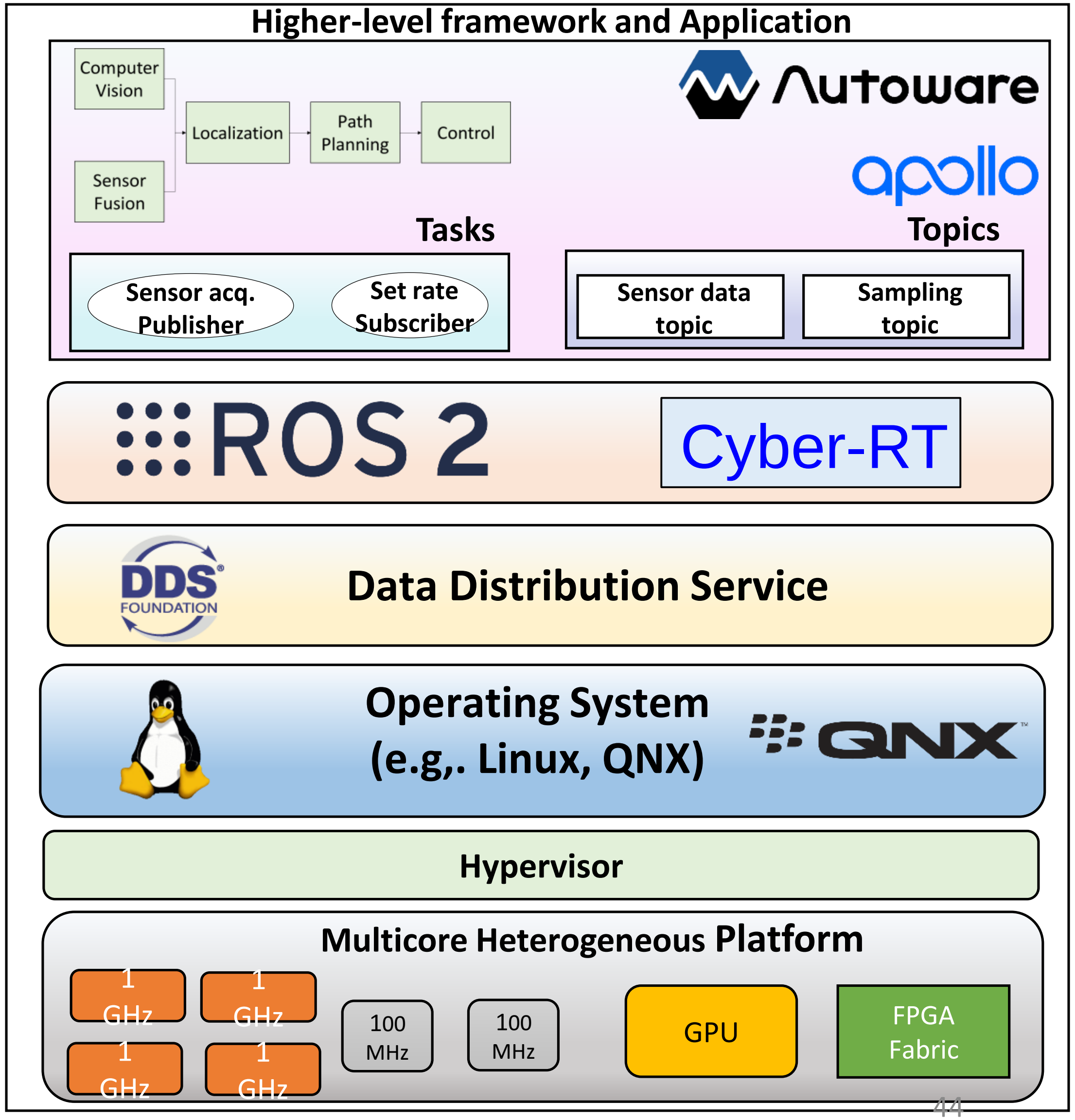
1) **Daniel Casini**, Tobias Blaß, Ingo Lütkebohle, and Björn B. Brandenburg, "Response-Time Analysis of ROS 2 Processing Chains under Reservation-Based Scheduling", In Proceedings of the 31th Euromicro Conference on Real-Time Systems (ECRTS 2019), Stuttgart, Germany, July 9-12, 2019.
2) Tobias Blaß, **Daniel Casini**, Sergey Bozhko, and Björn B. Brandenburg, "A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance", In Proceedings of the 42nd IEEE Real-Time Systems Symposium (RTSS 2021), Dortmund, Germany, December 7-10, 2021.

Why the scheduling in ROS is complex?

Two levels of scheduling:

- ROS processes are scheduled by the Linux operating system
- Callbacks (e.g., C++ functions) are in turn scheduled by the ROS executor implemented by ROS

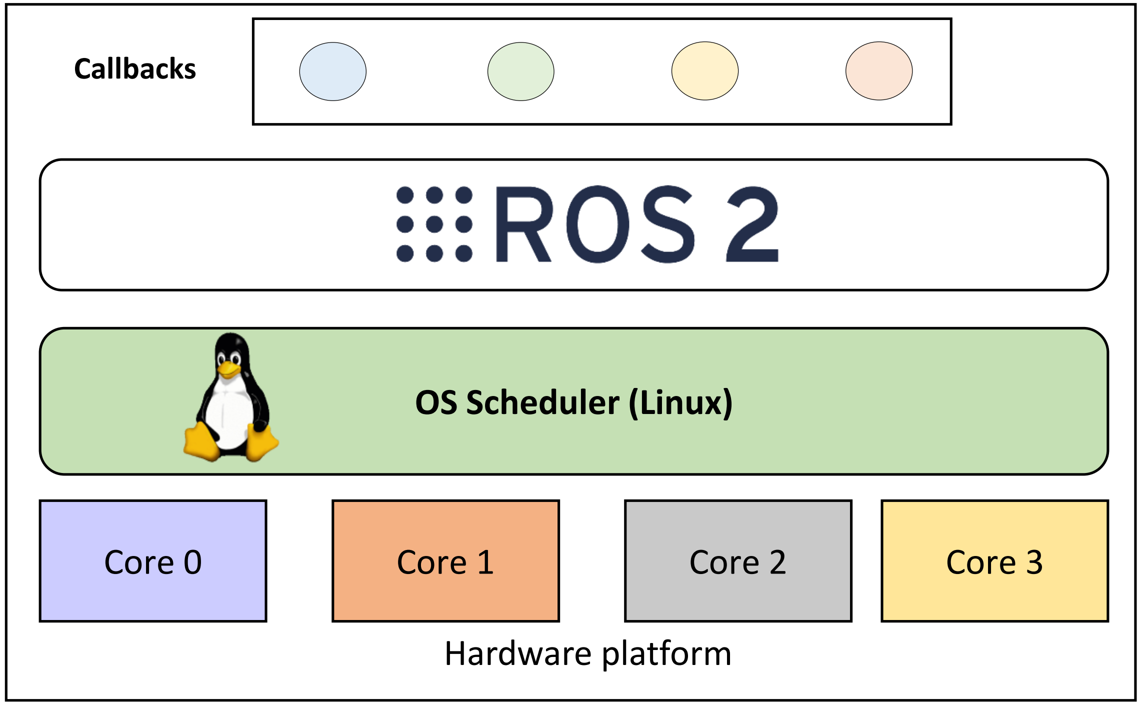
Even more complex when thinking to the complete software stack



Why the scheduling in ROS is complex?

Two levels of scheduling:

- ROS processes are scheduled by the Linux operating system
- Callbacks (e.g., C++ functions) are in turn scheduled by the ROS executor implemented by ROS



ROS-level analysis accounting for the ROS's quirks

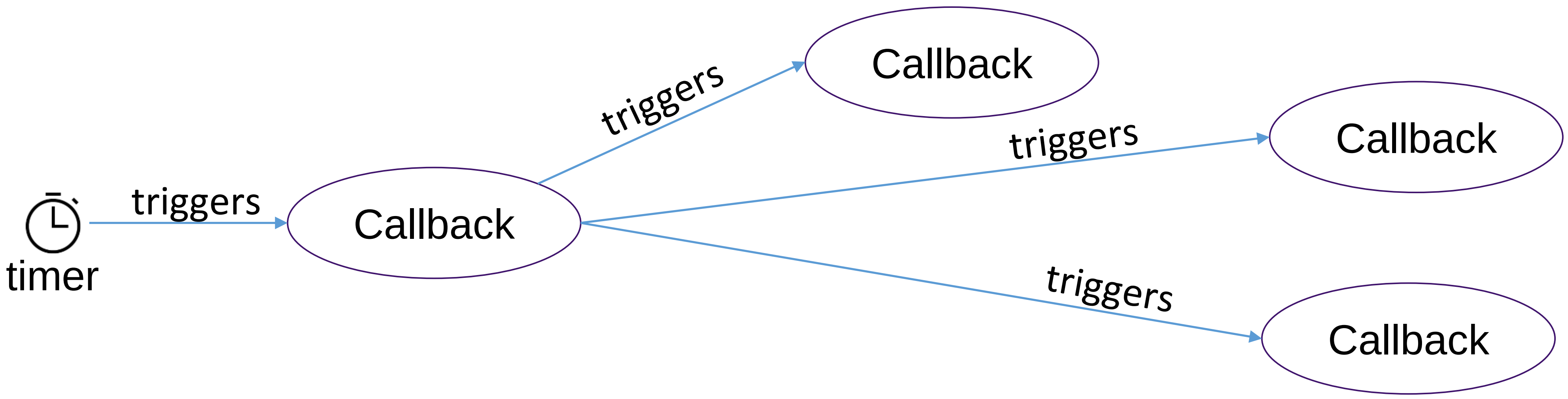
Based on CPA [1]
(Compositional Performance Analysis)

Linux-level scheduling using reservations (e.g., SCHED_DEADLINE)

[1] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," IEEE Proceedings - Computers and Digital Techniques, March 2005.

Modeling

- ROS Systems are distributed networks of callbacks

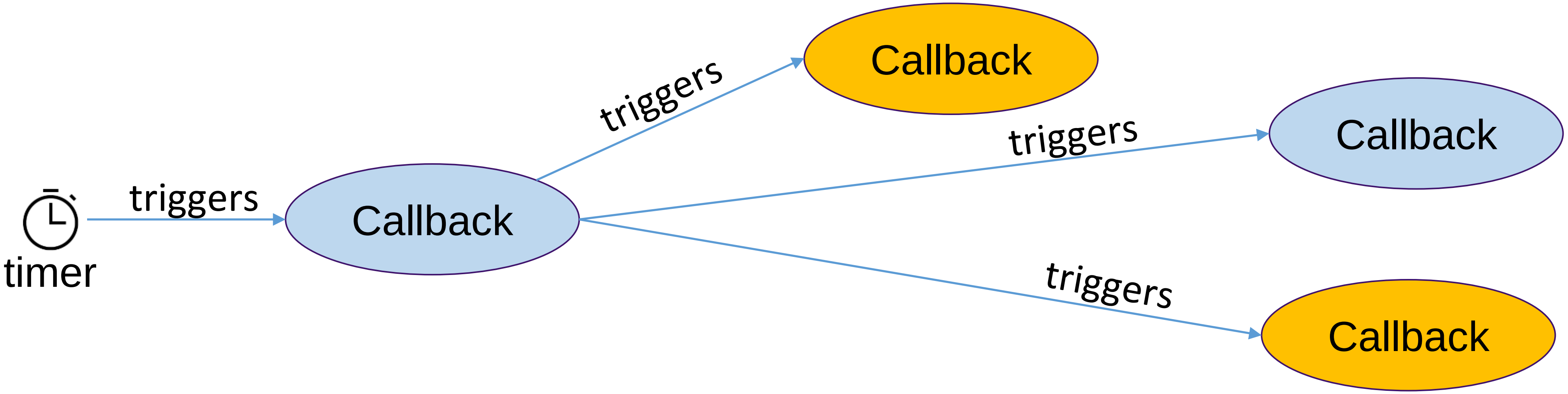


* Slides adapted from the original presentation by Tobias Blass

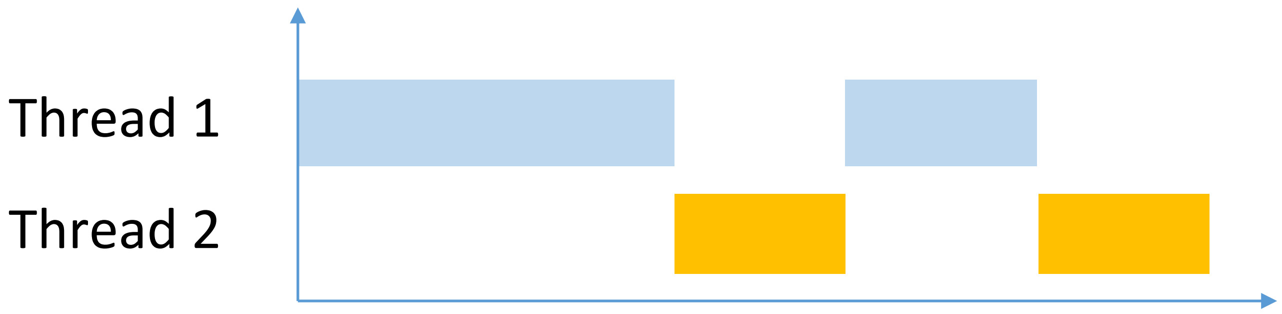
Modeling

- Callbacks are assigned to <u>executor threads</u>

ROS-Level Scheduling

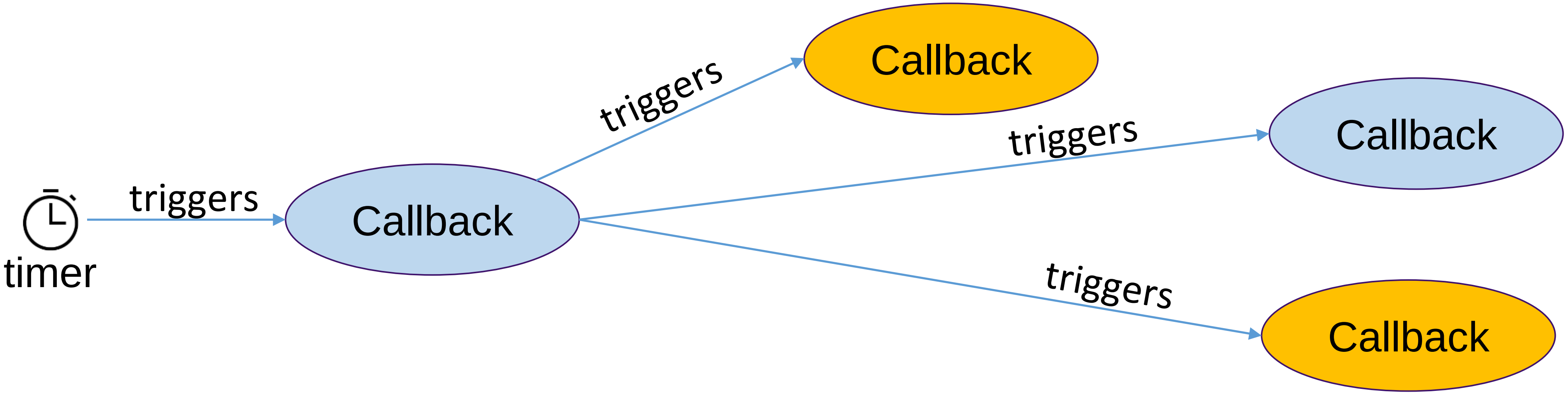


Linux-Level Scheduling



Modeling

ROS-Level Scheduling



The ROS documentation **does not specify the execution order** of callbacks

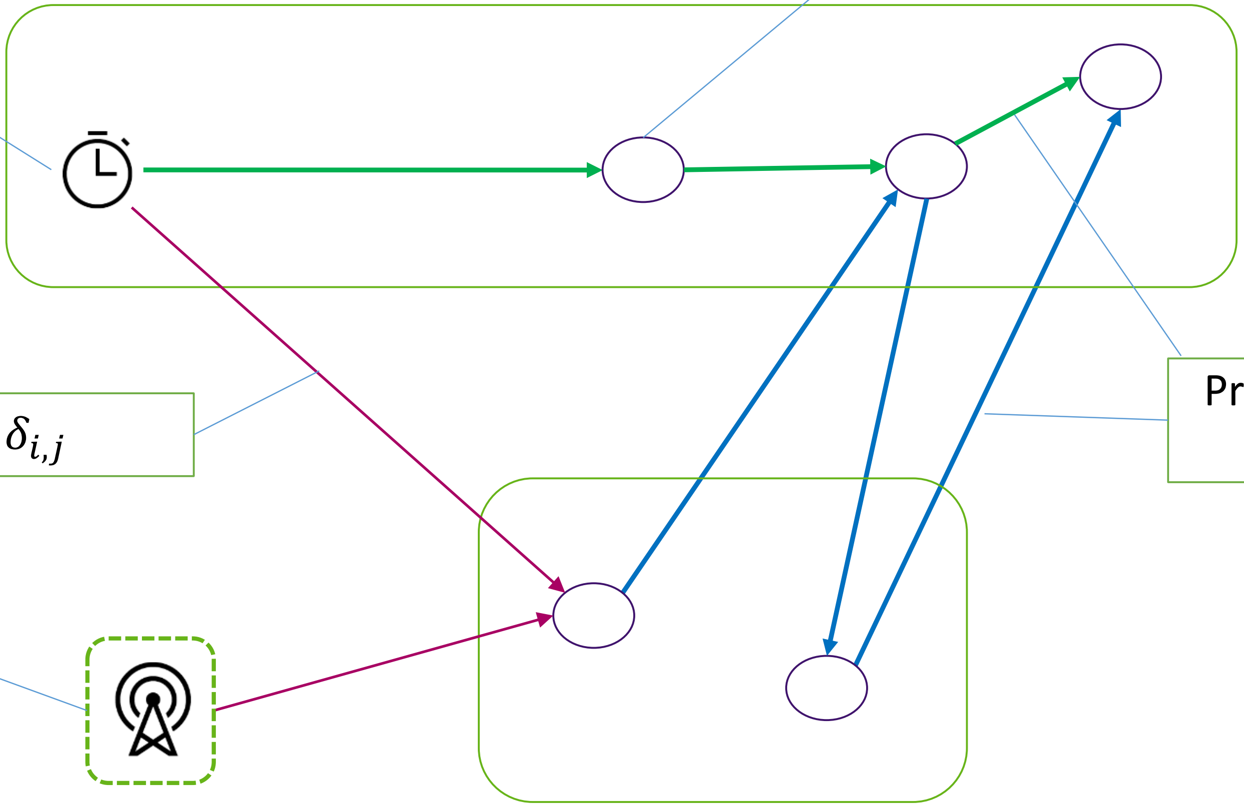
A Real-Time Model for ROS 2

Timer callback

- WCET bound e_i
- Priority π_i
- Event arrival curve η_i^a

Subscription callback

- WCET bound e_i
- Priority π_i



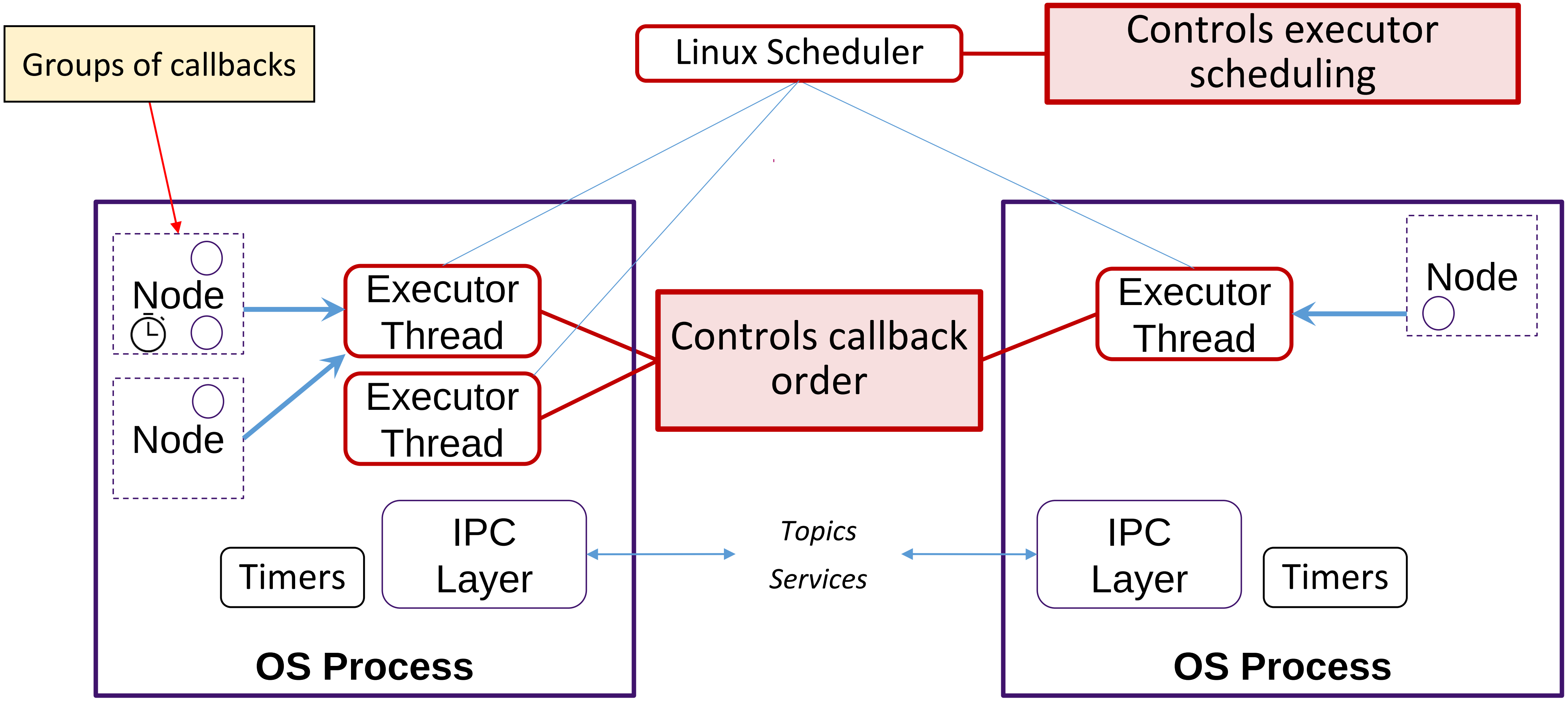
Communication Delay $\delta_{i,j}$

Processing chains of interest

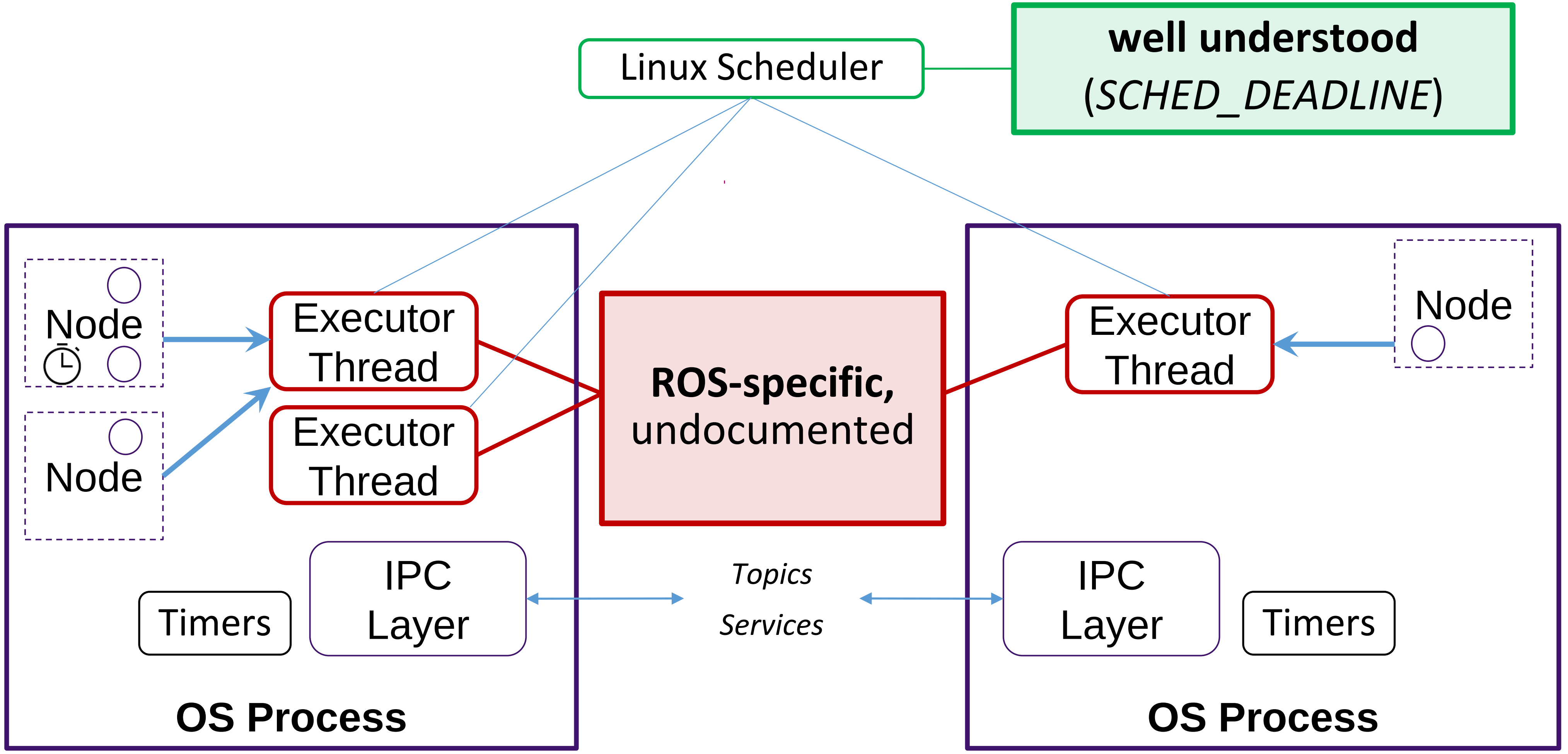
Event source external to ROS

- Event arrival curve η_i^a

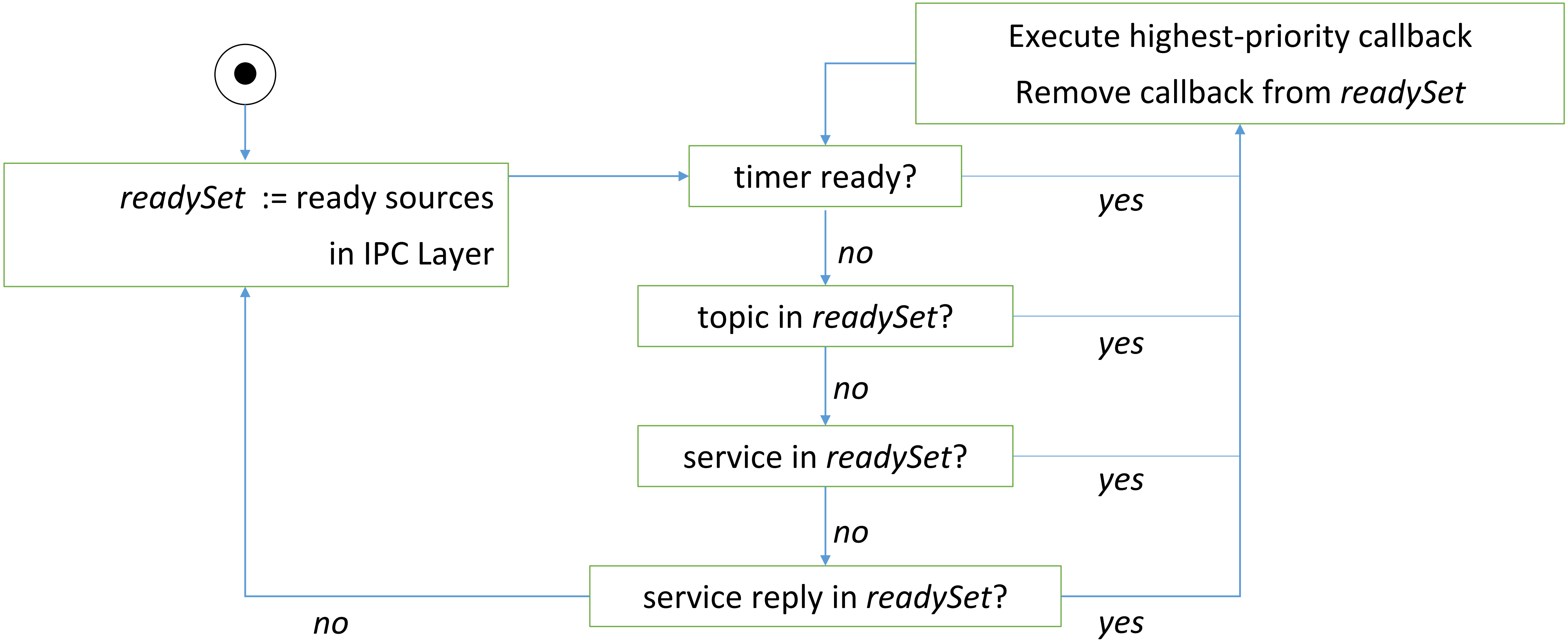
The operating system's view



The operating system's view



The Executor's Algorithm

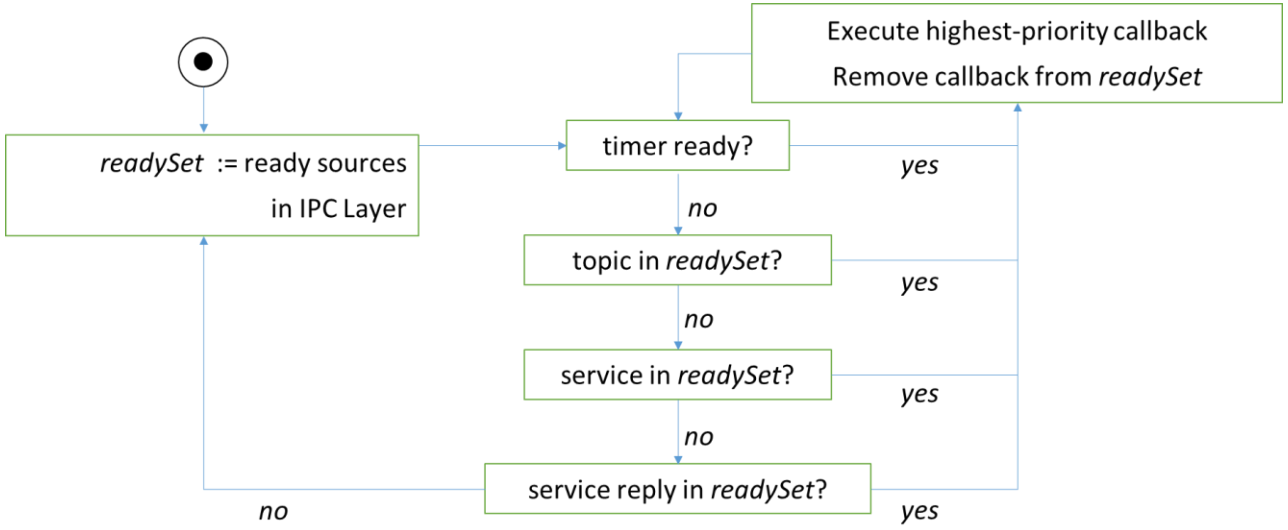


The Executor's Algorithm

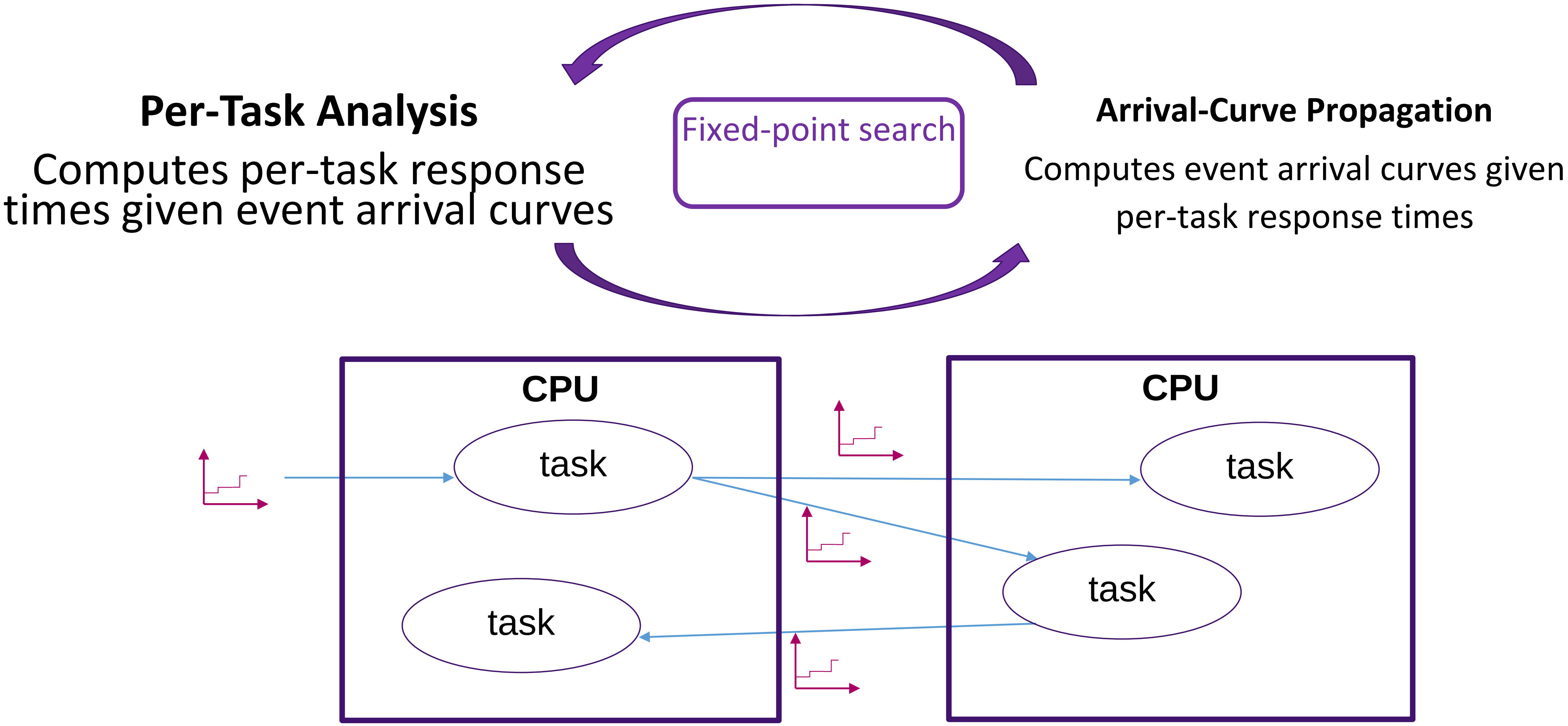
- It significantly differs from usual schedulers
- ROS maintains a **set (i.e., at most one instance per time)** of ready callbacks
- When the set is empty (polling point), it queries the IPC layer for new activations

This can create priority inversion effects

- Except timers, up to ROS 2 "Dashing"
- It creates two levels of priority:
  - The **first level of priority** is given by the callback type (timer, subscription, service, service reply, in this order)
  - The **second level of priority** is given by the callback registration order in the ROS program

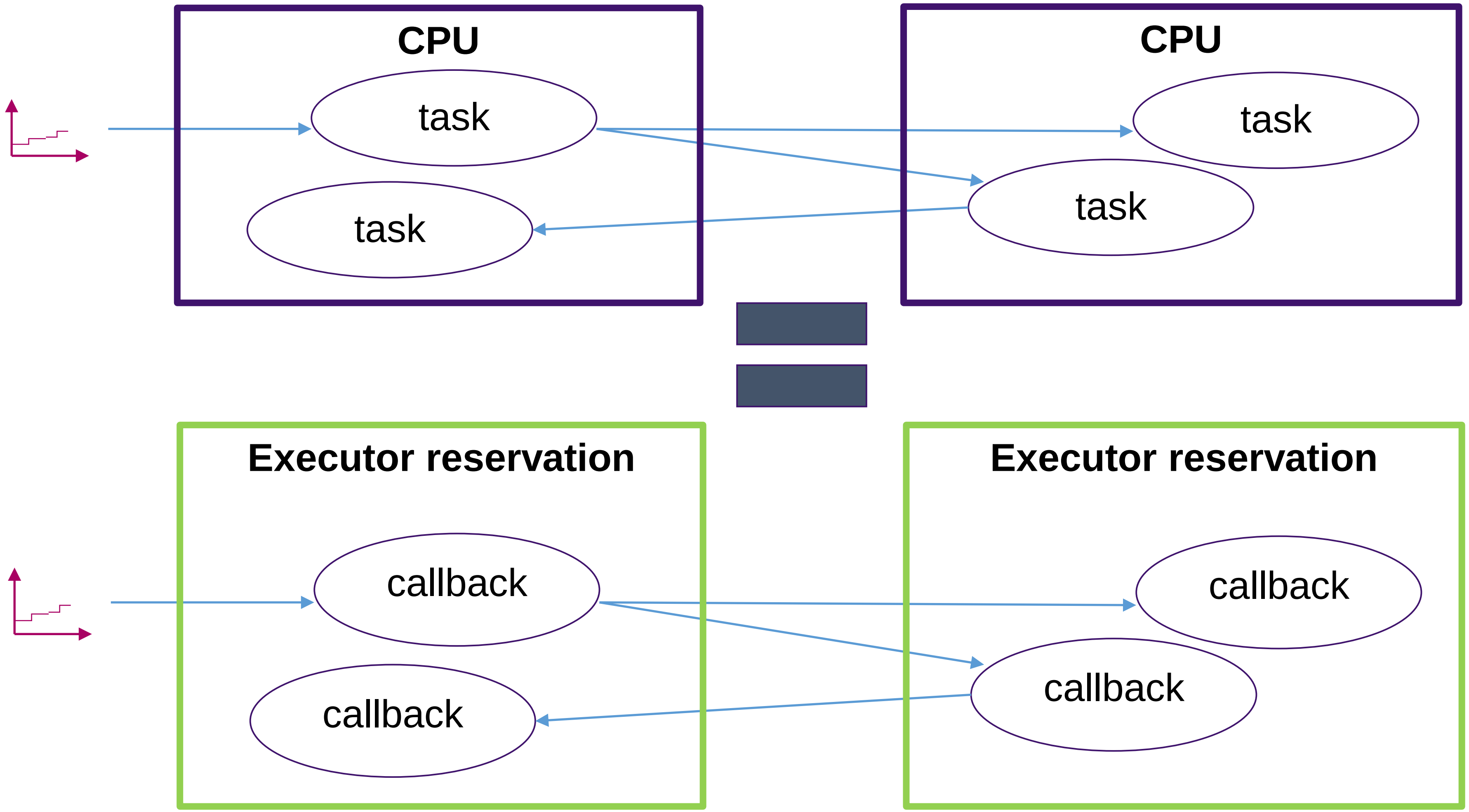


Compositional Performance Analysis

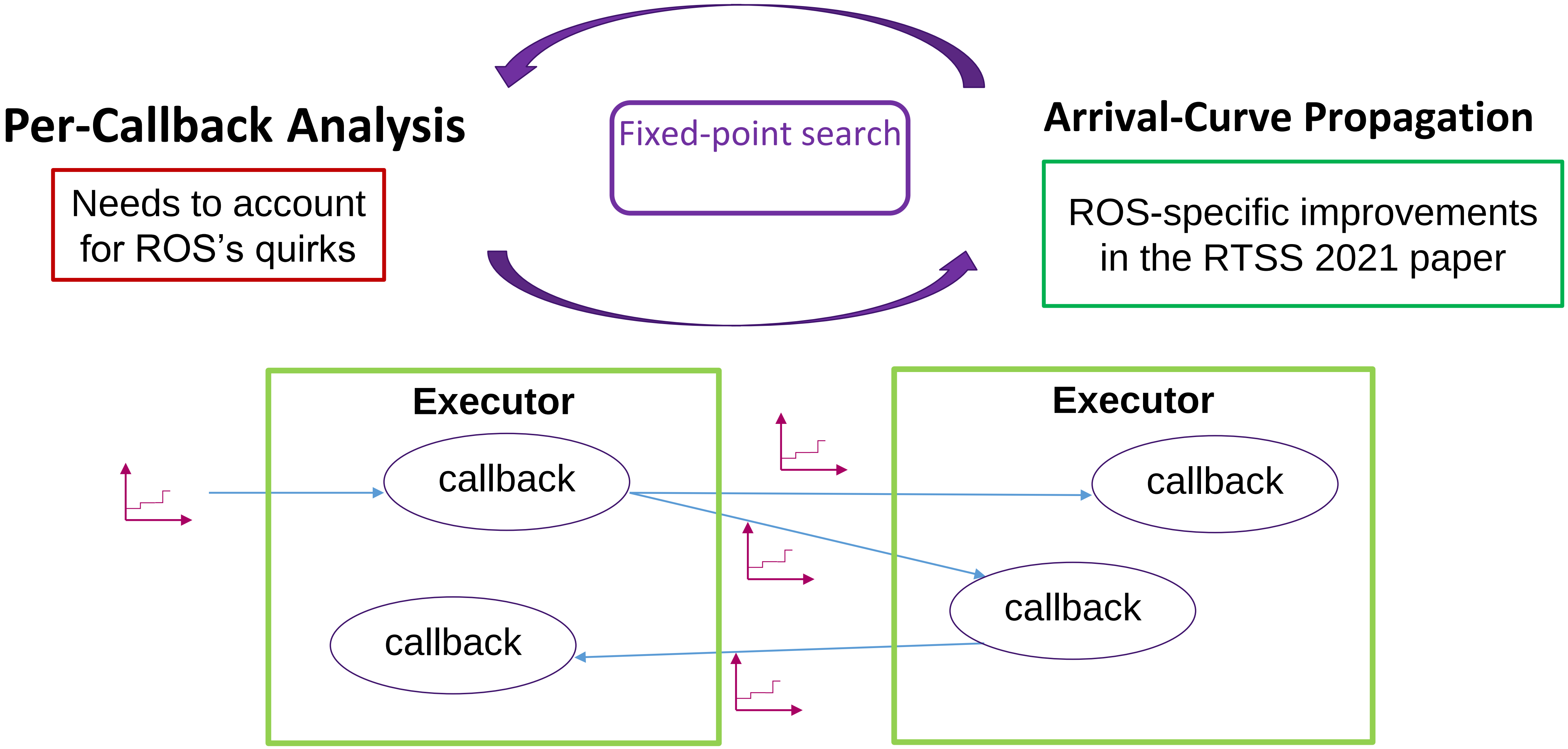


[1] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis - the SymTA/S approach,” IEEE Proceedings - Computers and Digital Techniques, March 2005.

The CPA approach fits ROS well



The CPA approach fits ROS well

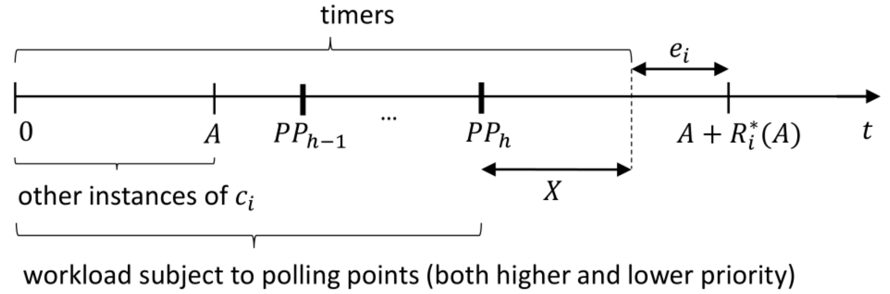


Response-Time Analysis – ECRTS 2019

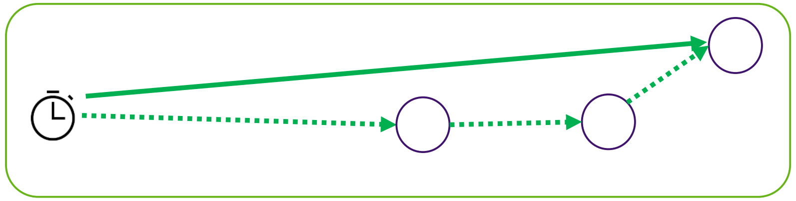
$$sbf_k(A + R_i^*(A)) = rbf_i(A+1) + RBF(hp_k(c_i), A + R_i^*(A) - e_i + 1) + B_i$$

$$sbf_k(A + R_i^*(A)) = rbf_i(A+1) + RBF(\{\mathcal{C}_k \setminus c_i\}, A + R_i^*(A) - e_i + 1)$$

Dedicated analyses for **timers and polling-point-based callbacks**



Response-Time Analysis that accounts for the ROS's peculiarities



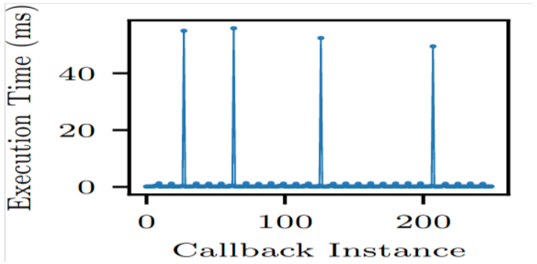
Optimization for **intra-executor** chains

Response-Time Analysis – RTSS 2021

Retis
Real-Time Systems Laboratory

We **improve** upon existing response-time analyses with three techniques.

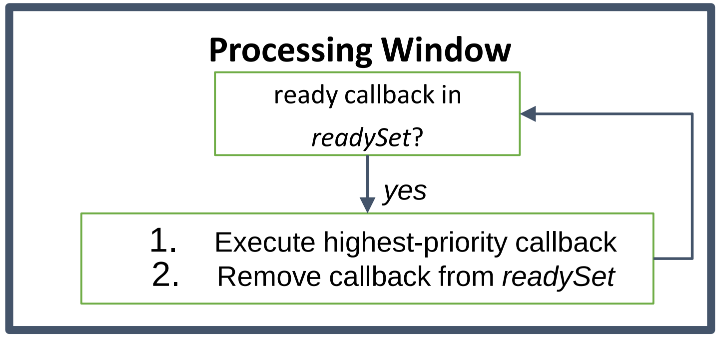
1. Address large **execution-time variance** over time



AMCL /tf callback in the navigation 2 package

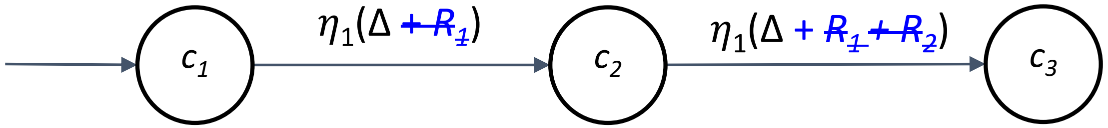
2. Exploit **starvation-freedom** in the callback scheduler

At most one instance of each callback can run in a processing window



3. Improve activation-curve **propagation within executors**

Activation curve $\eta_1(\Delta)$

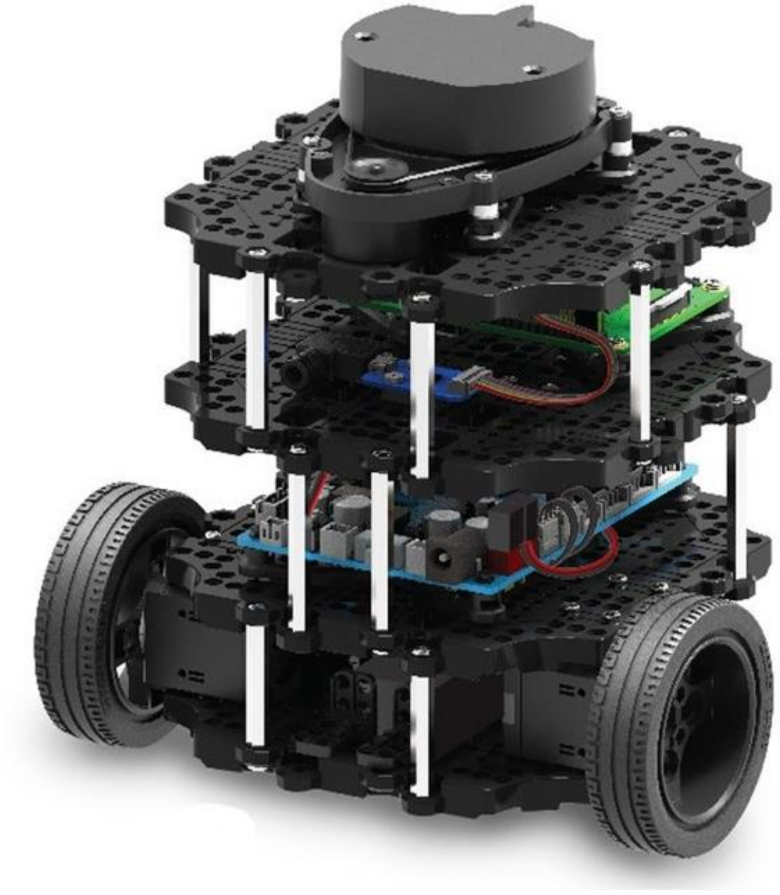


Evaluation

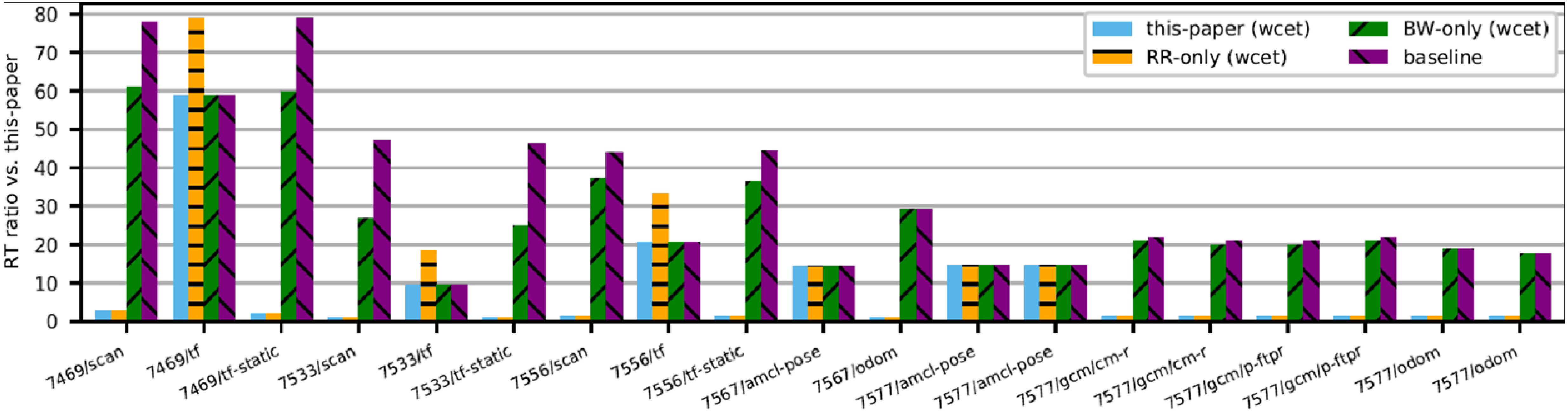
- **Turtlebot 3** "Burger" controlled by a **Raspberry Pi 4B**

- Running various **ROS packages**
  - Navigation 2 packages
  - Turtlebot 3 drivers

- Callback graph extracted from measurements
  - See Blass et al., "Automatic Latency Management for ROS 2: Benefits, Challenges, and Open Problems", RTAS *2021*

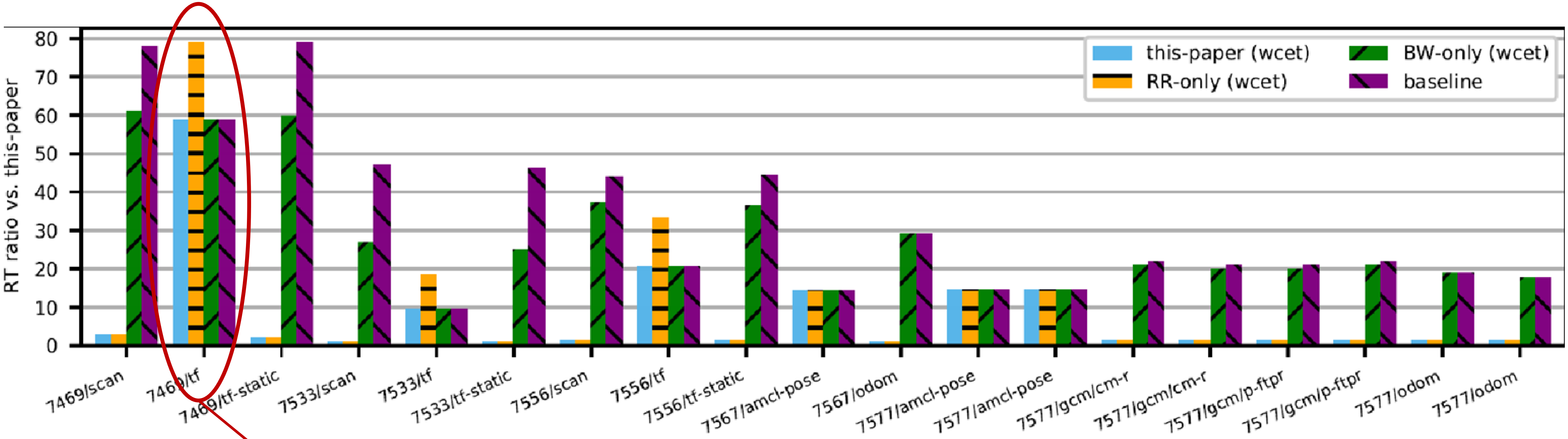


Evaluation



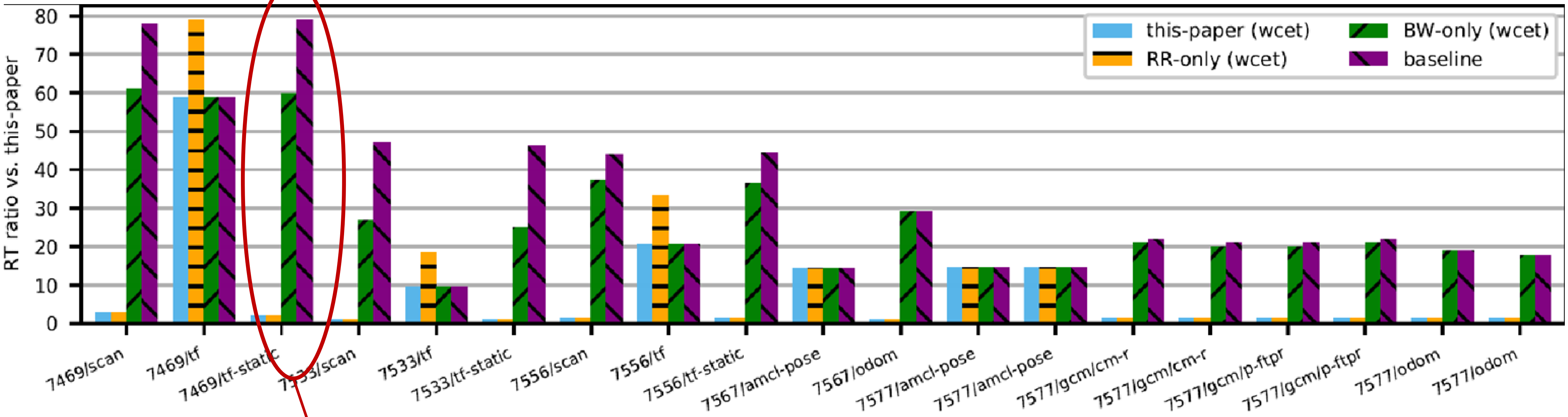
RT ratio vs. this-paper
0
10
20
30
40
50
60
70
80
this-paper (wcet)
RR-only (wcet)
BW-only (wcet)
baseline
7469/scan
7469/tf
7469/tf-static
7533/scan
7533/tf
7533/tf-static
7556/scan
7556/tf
7556/tf-static
7567/amcl-pose
7567/odom
7577/amcl-pose
7577/amcl-pose
7577/gcm/cm-r
7577/gcm/cm-r
7577/gcm/p-ftpr
7577/gcm/p-ftpr
7577/odom
7577/odom

Evaluation



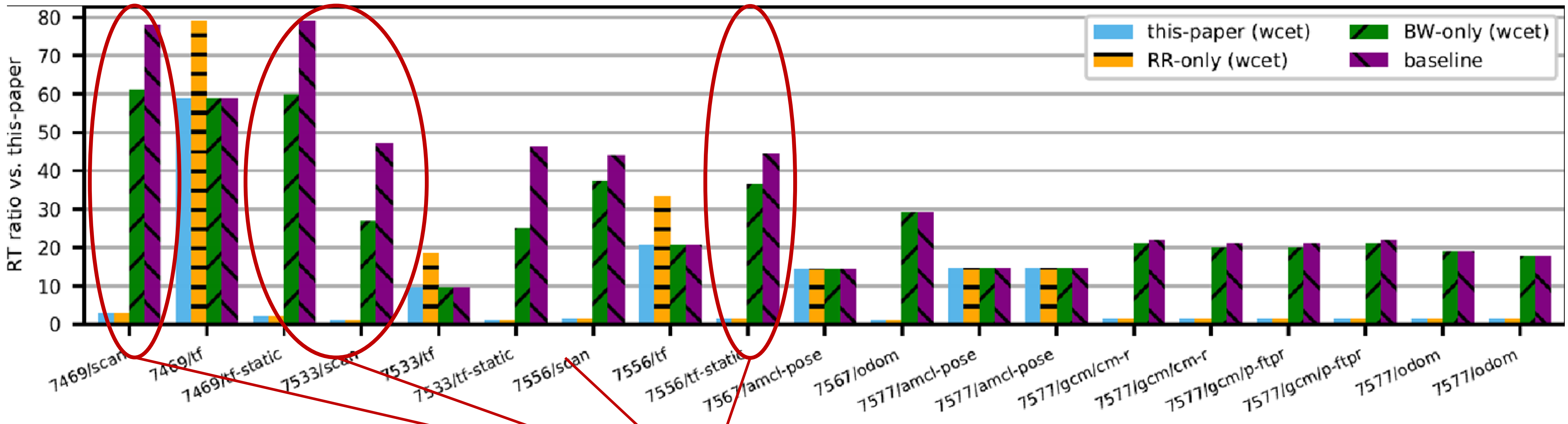
High-variance callback ⇒ Large gains from execution-time curves

Evaluation



Shares executor with bursty callback
⇒ Large gains from exploiting starvation-freeness

Evaluation



Gains over baseline thanks to improved arrival curve propagation

ROS 2 – Open Problems

- Real-Time Analysis of the multi-threaded executor of ROS 2
- Real-Time Analysis of the (many) custom executors that have been developed over the last years (see https://www.apex.ai/roscon-21)
- Scheduling of ROS 2 and DDS threads in a coordinate fashion (e.g., using QNX APS)
- Development of new special-purpose and real-time friendly executors
- And many more…

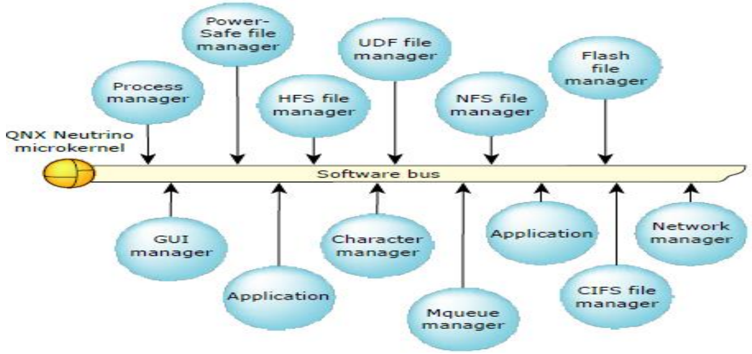
Real-Time Systems Laboratory

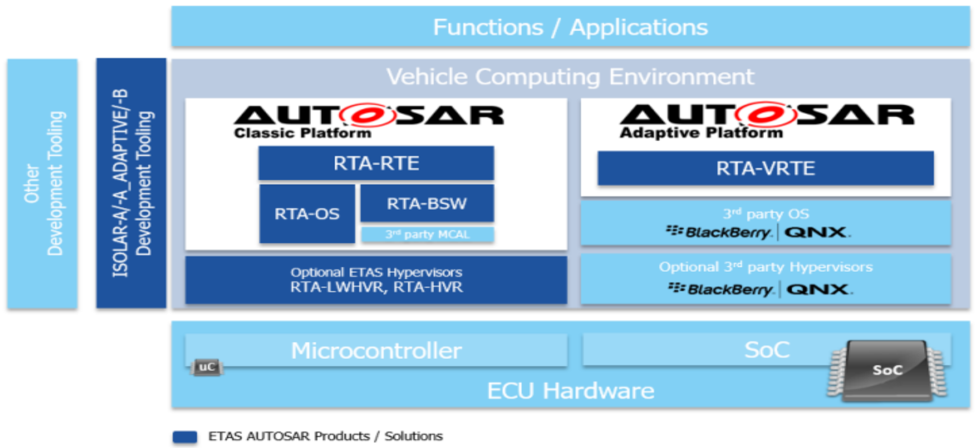
The opportunity of using QNX

- Preferred base operating system by many automotive OEMs
- ISO-26262 certified at the highest level of assurance (ASIL-D)
- POSIX Compliant, commercial, proven: AUTOSAR Adaptive needs POSIX
- Commercial OS supporting CPU reservations!

Interestingly, the QNX reservation-based scheduler supports multiple threads in the same reservation, thus offering opportunities for the coordinate scheduling of ROS and DDS threads

However, it is first necessary to understand the behavior of the **QNX reservation-based scheduling algorithm**





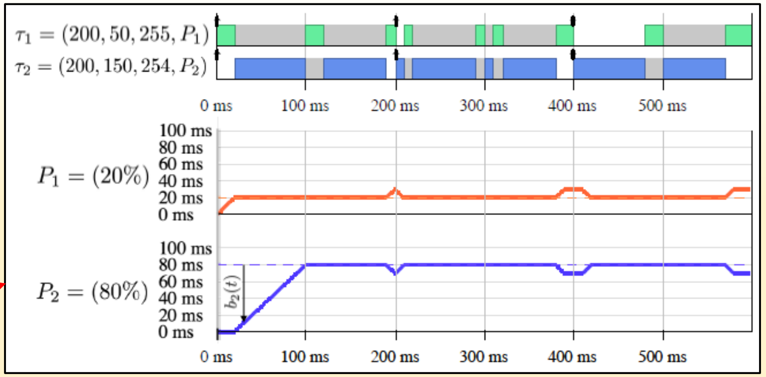
QNX Adaptive Partitioning Scheduler

Dakshina Dasari, Matthias Becker, **Daniel Casini**, and Tobias Blaß, "End-to-End Analysis of Event Chains under the QNX Adaptive Partitioning Scheduler", *In Proceedings of the 28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2022)*, Milan, Italy, May 4-6, 2022.

Retis
Real-Time Systems Laboratory

How does APS work?

- To describe the behavior of APS partitions, we presented a set of **rules**
  - They have been derived by relying both on the QNX documentation, and by performing proper validation experiments to corroborate our findings with empirical evidence (more about this later in the presentation)
  - Validation results have been checked by collecting scheduling traces



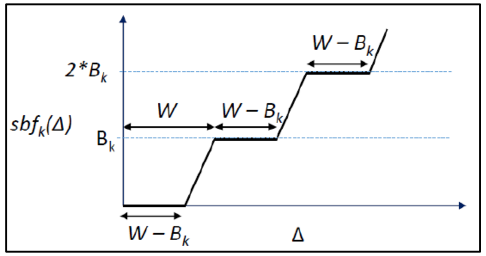
- We considered two cases:
  - Budget reclaiming is disabled, which can be configured with the SCHED_APS_SCHEDPOL_LIMIT_CPU_USAGE
  - Budget reclaiming is enabled and idle time is distributed in priority order, which is the default option in QNX (SCHED_APS_SCHEDPOL_DEFAULT)

- Several rules have been derived, to define how budget is initialized, decremented, incremented, etc.

Analyzing APS

- To analyze APS, we provided three contributions

1. A supply bound function which lower-bounds the minimum service provided by a partition in any time interval of length Δ



2. A response-time analysis for the threads running in each partition

$$sbf_k(A + R_{x,h}(A)) \geq rbf_e(A+1) + RBF(\mathcal{T}_k^{hep}(\gamma_{x,h}) \setminus \{\tau_e\}, A + R_{x,h}(A) + 1)$$

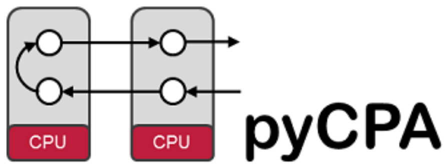
3. A condition to guarantee that each partition can correctly deliver the supply to pending tasks, i.e., to guarantee that the core is not overloaded

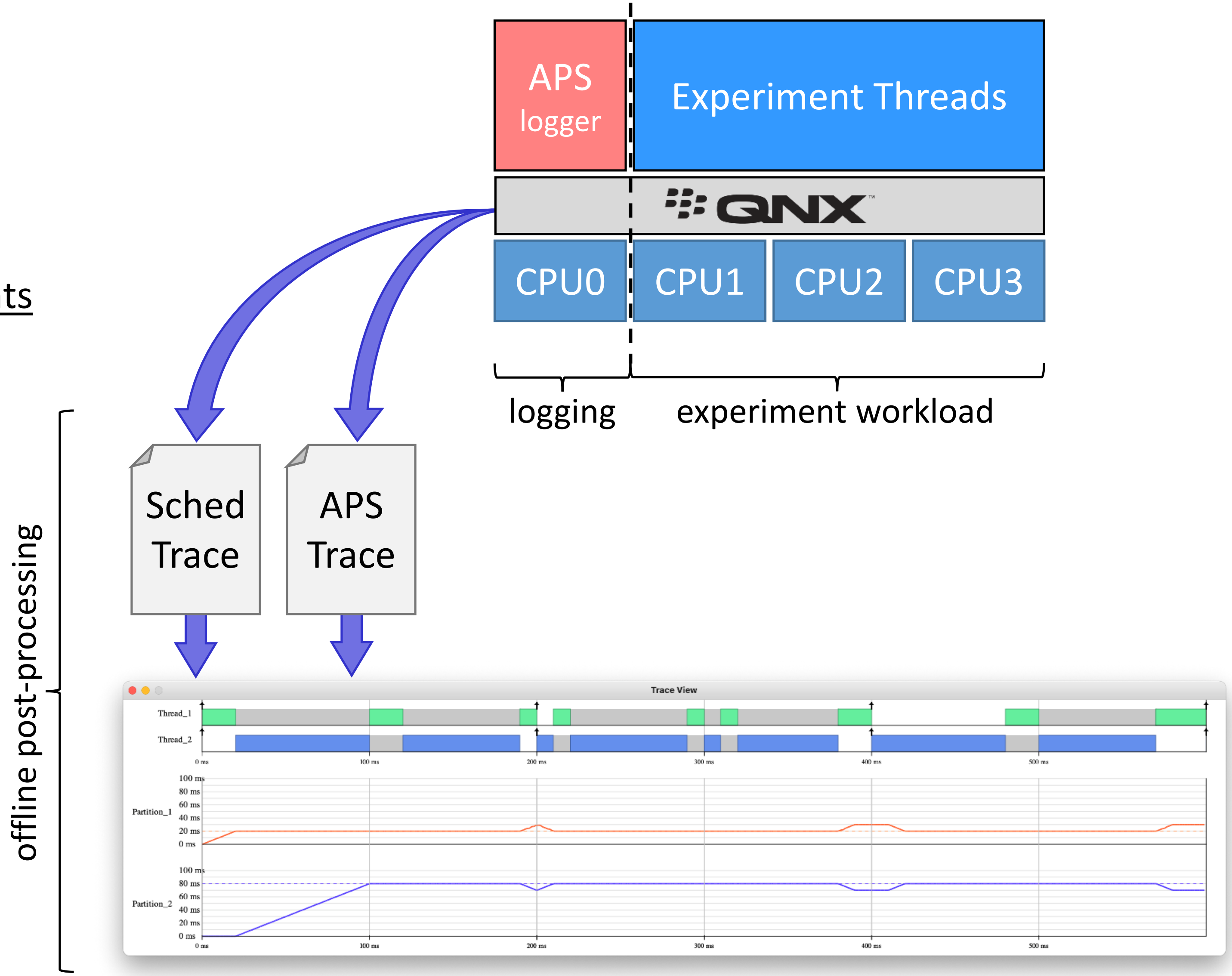
$$\sum_{P_k \in \mathcal{P}_j} B_k \leq W$$

More details in the paper

Experiments

- Setup on the real platform
  - QNX Software Development Platform 7.1
  - Raspberry Pi 4b (4 cores, 4GB RAM)
- Recording Traces
  - QNX Tracelogger to record scheduling events
  - Custom logger to record APS trace
    - Granularity 1ms
    - Assigned to dedicated core
- Custom tool to evaluate the recorded traces
- Analysis implemented on top of PyCPA

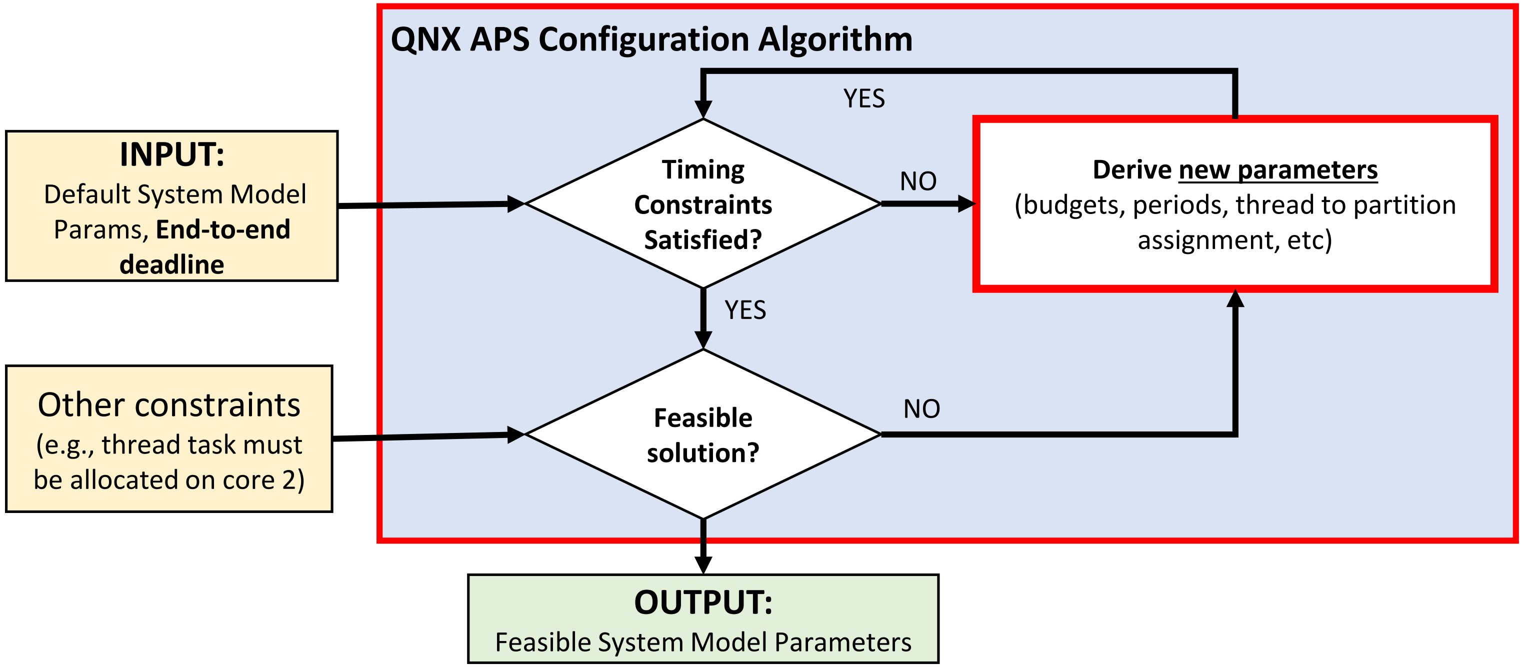




QNX– Open Problems

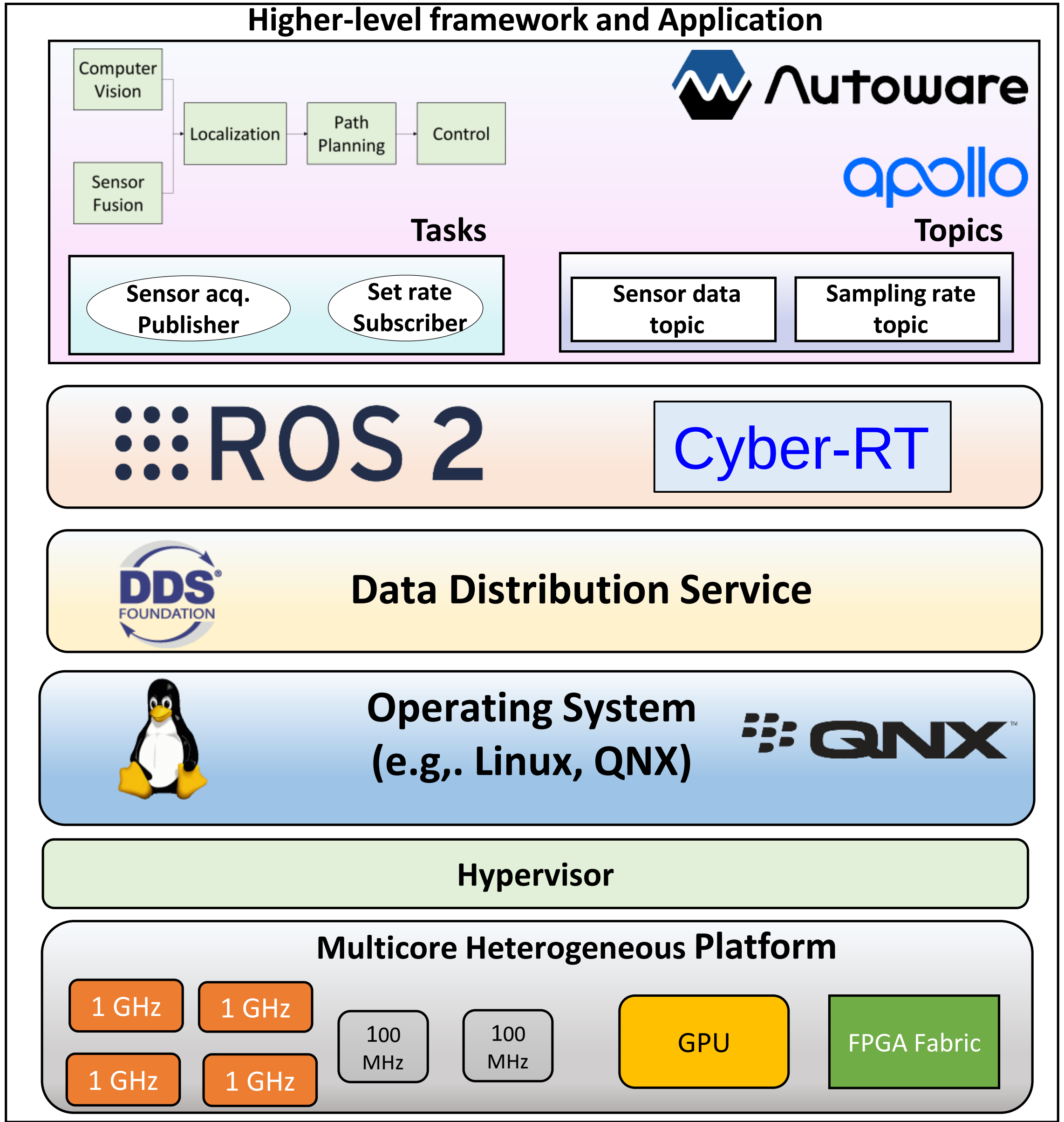
- Many, as usual, but let's discuss a particular one:

The design of tools for the design-space exploration of the system parameters using this analysis.



Conclusions

Conclusions



- Autonomous driving is incredibly complex – much work still need to be done
- This is due to unpredictability issues occurring at multiple levels – from the application to the hardware
- Redesigning such systems to be time-predictable would be the best, but it is not an easily viable option because they are already widely used and provides a large amount of pre-implemented functionalities

We should try our best to improve their time-predictability by acting at different levels

Are you interested in these topics?

- Join the RAGE 2022 workshop at DAC 2022! (see https://rage2022.github.io/)
- Advanced rate registration ends June 10th
- Organized by me, Dakshina Dasari (Bosch), and Matthias Becker (KTH)

Thank you!

Daniel Casini
daniel.casini@santannapisa.it