PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2

Hyunjong Choi, Yecheng Xiang, Hyoseung Kim

UC RIVERSIDE

RTEN Real-time Embedded and Networked Systems Laboratory

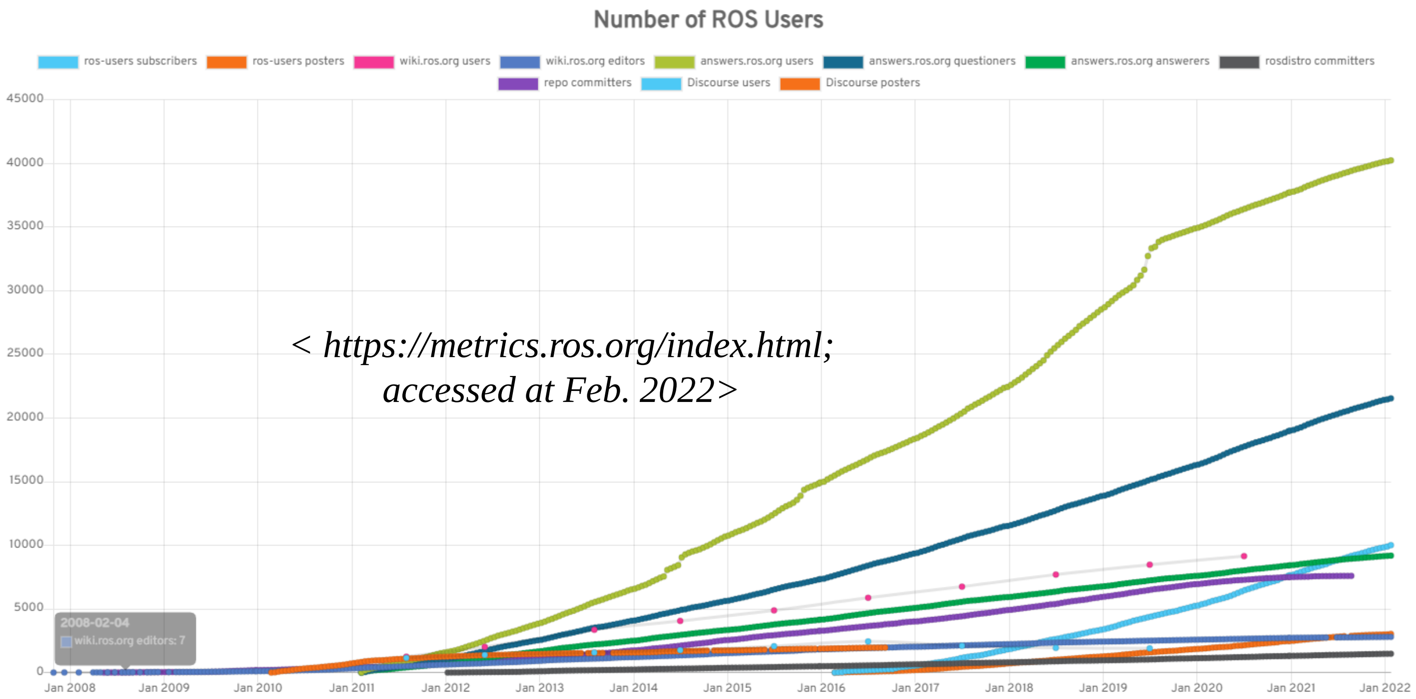
I. Introduction

Robot Operating System (ROS)

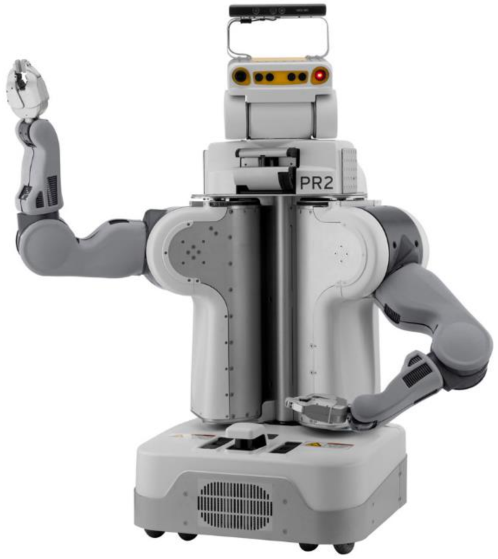
- ROS (since 2007)
  - Popular open-source middleware in academia and industry
  - Provides software tools, robot systems, and best-practices

ROS 1.0
http://ros.org/

➡ Over the decades, it has revealed shortcomings in real-time support for timing- and safety-critical applications



< https://metrics.ros.org/index.html; accessed at Feb. 2022>



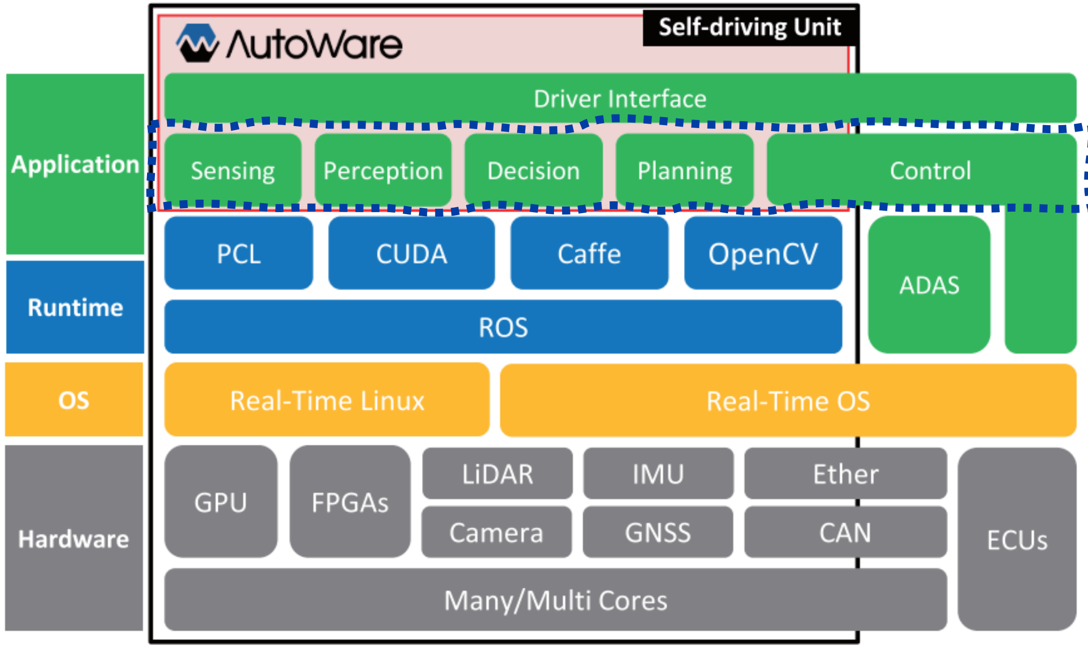
Willow Garage PR2
(original ROS robot)

http://willowgarage.com

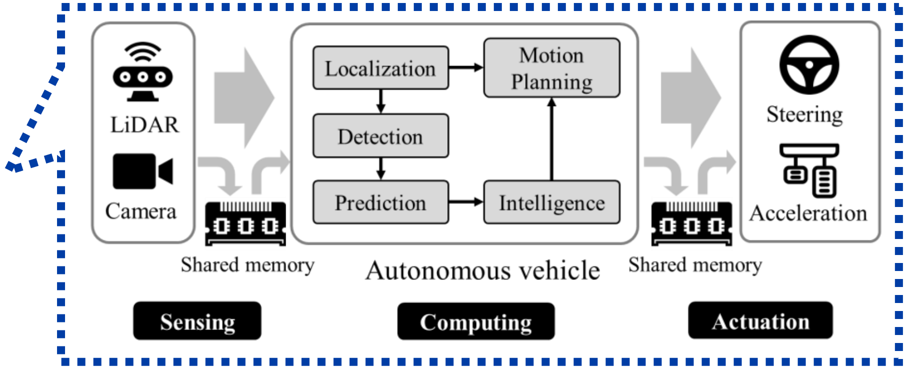
I. Introduction

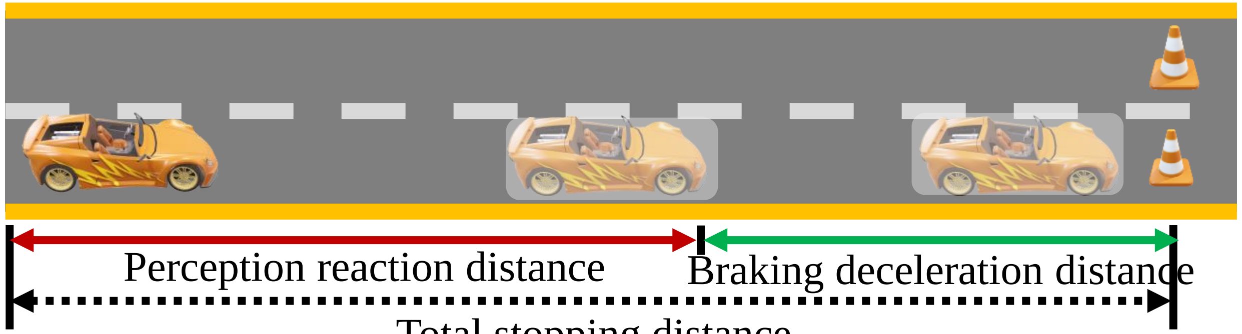
Why real-time in ROS ?

- To develop safety-critical application with ROS
  - Autonomous driving software (e.g., autoware.ai)



< Autoware.ai > †





< Motivational example: chain in self-driving application >

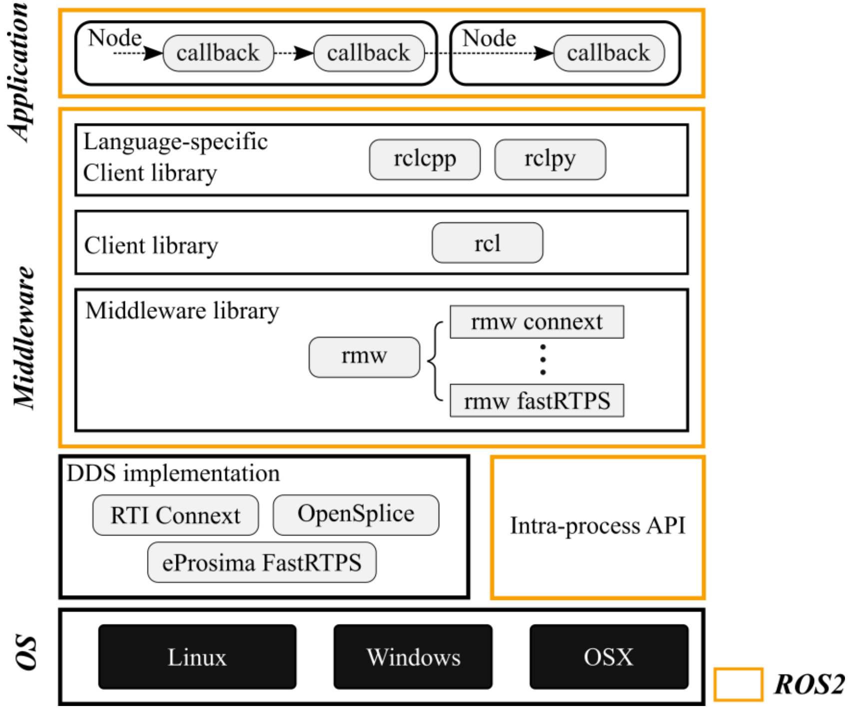
Timing constraint violations (e.g., end-to-end latency) can cause catastrophic accidents

†S. Kato et al. "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems", ICCPS, 2018

I. Introduction

ROS 2 (since 2017)

- Most concepts are inherited from the original ROS design (e.g., pub-sub)
- Aims to improve **real-time capability, QoS, and security**
- Supports Data Distribution Service (DDS)



< ROS2 architecture >

Ardent Apalone, released Dec 2017

Eloquent Elusor, released Nov 2019

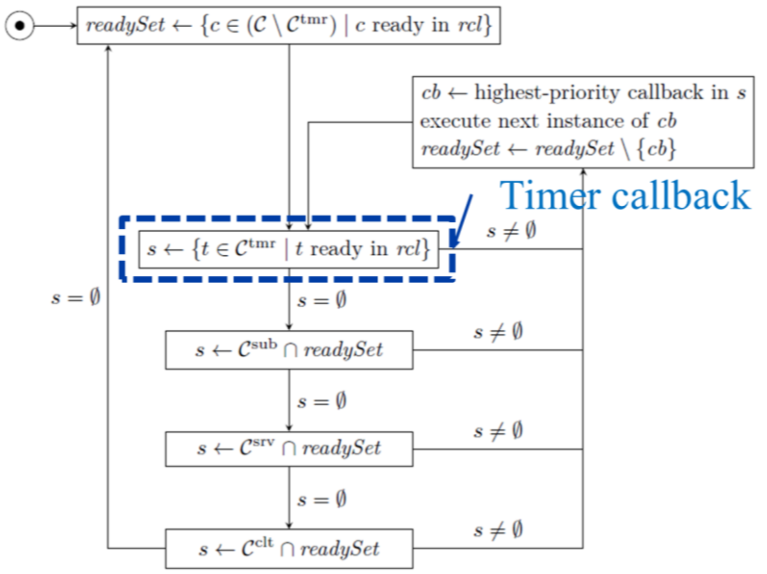
Galactic Geochelone, released May 2021

I. Introduction

Limitation of current ROS2

- Priority-unaware complex layers of abstractions
  - Unique callback scheduling behavior
  - *Ignores criticality or urgency of task chains*

➡ **Suffers from priority inversion**



< Callback scheduling in executor† >

- Lack of systematic support for resource allocation and analysis
  - Poor resource utilization
  - Nondeterministic end-to-end behavior

➡ **Needs a new RT scheduling framework for ROS2 !**

† D. Casini et al. "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", ECRTS, 2019

II. ROS2 Background & System Model

Scheduling-related abstractions in ROS2

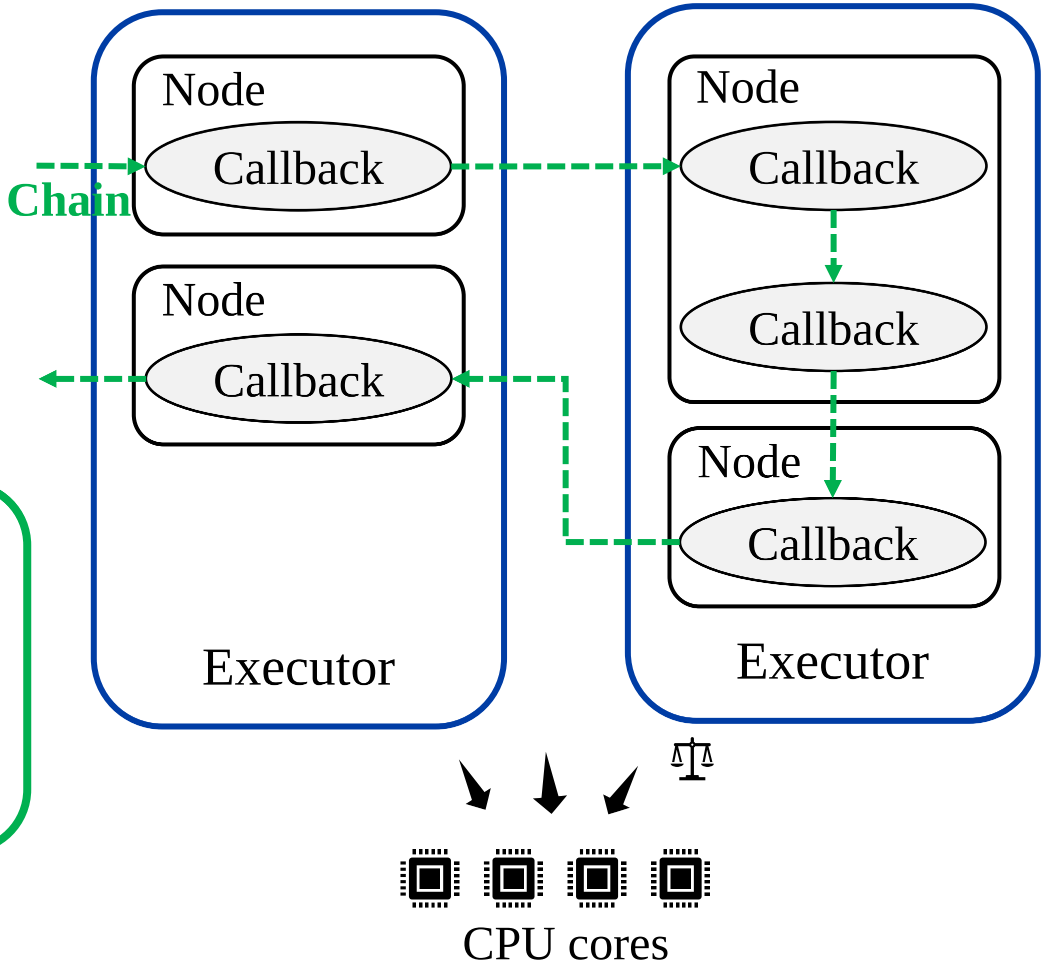
- *Callbacks*, *nodes*, and *executors*

Callback model

- ✓ Timer / regular callbacks
- ✓ Non-preemptive
- ✓ $\tau_i \coloneqq (C_i, D_i, T_i, \pi_i)$

Chain model

- ✓ $\Gamma^c \coloneqq [\tau_s, \tau_{m1}, \dots, \tau_e]$
- ✓ τ_s : the start callback of Γ^c
- ✓ τ_{m*} : the intermediate callback of Γ^c
- ✓ τ_e : the end callback of Γ^c



Node model

- ✓ $\mathcal{N} =: \{n_1, \dots, n_j, \dots, n_N\}$
- ✓ Not schedulable entities

Executor model

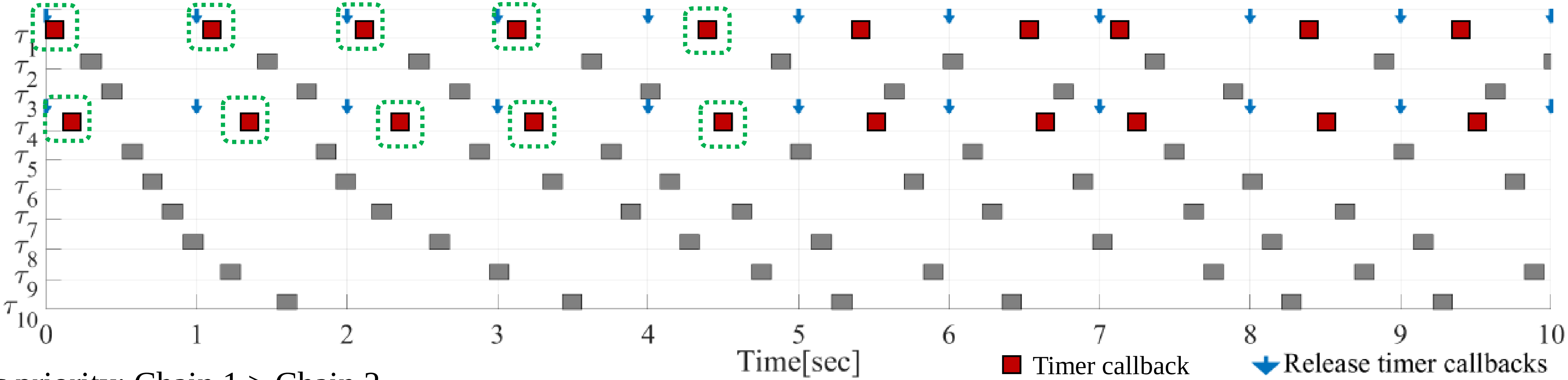
- ✓ $\mathcal{E}_i =: \{e_1, , \dots, e_j, \dots, e_E\}$
- ✓ Preemptive
- ✓ Schedule with `SCHED_FIFO`

Challenges (1/2)

- Challenge I: Fairness-based scheduling ***within executors***

Chain 1

Chain 2



Semantic priority: Chain 1 > Chain 2

Single executor	Mean	Max	Min	STD
Chain 1	36.865	72.752	0.505	21.223
Chain 2	36.730	73.149	0.773	21.154

< End-to-end latency results [sec] >

O1. Prioritizes timer callbacks regardless of chain priority [†]

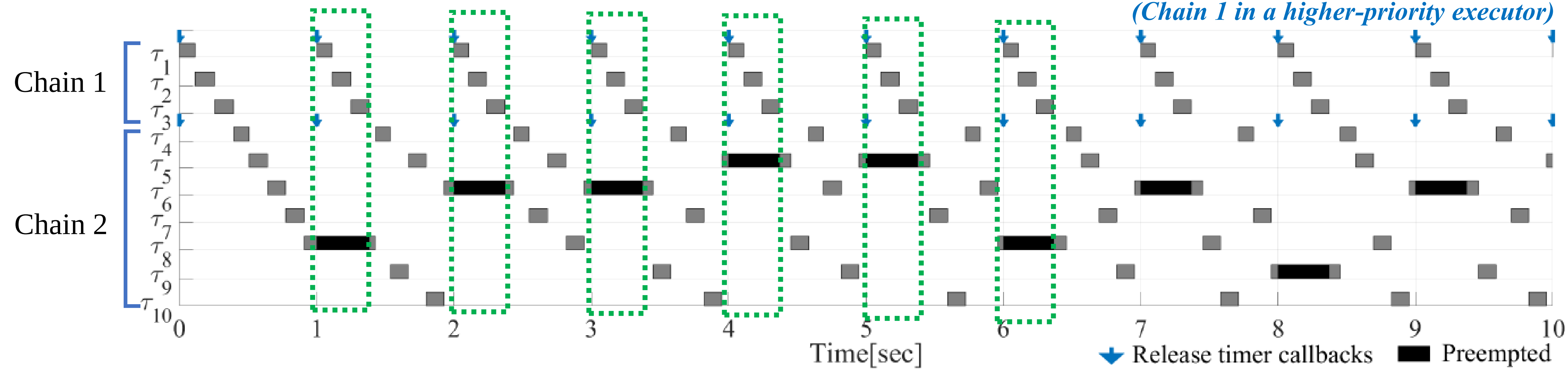
O2. Does not distinguish callbacks by their origin chains

Jeopardizes the timeliness of safety-critical chains

[†] D. Casini et al. "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", ECRTS, 2019

Challenges (2/2)

- Challenge II : Priority assignment for executors



Semantic priority: Chain 1 > Chain 2

Single executor	Mean	Max	Min	STD
Chain 1	0.370	0.392	0.366	0.004
Chain 2	48.795	97.783	0.772	28.304

< End-to-end latency results [sec] >

O3. High penalty due to self-interference
O4. No guidelines on executor priority assignment

Default ROS2 causes unacceptably high latency for chain 2

PiCAS: new real-time scheduling and analysis framework for ROS2

- Key idea: enables ***prioritization of critical computation chains*** across complex abstraction layers of ROS2
  - To minimize end-to-end latency
  - To ensure predictability even when the system is overloaded

- Re-design ROS2 scheduling architecture to achieve

  (1) ***Higher-semantic priority chain*** executes ***first***.

  (2) ***For each chain***, its instances on the same CPU ***execute in arrival order to prevent self-interference***.

Scheduling strategies

Timer callback
Regular callback
High priority chain
High priority executor
Low priority → High priority

- Strategies for chains running within an executor

	Regular callbacks only	Timer and regular callbacks
Single chain	**Strategy I.** (To satisfy ① of Lemma 1) $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$	**Strategy II.** (To satisfy ① of Lemma 1) $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4$
Multiple chains	**Strategy III.** $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ Chain 1 $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ Chain 2	**Strategy IV.** $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ Chain 1 $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4$ Chain 2

- Strategies for chains running across executors

Single chain on one CPU	Multiple chains on one CPU
Strategy V. (To satisfy ② of Lemma 1) $\tau_1 \rightarrow \tau_2$ or $\tau_1 \rightarrow \tau_2$	**Strategy VI.** τ_3 τ_1 or τ_3 τ_1

Priority assignment

- *Realization* **of scheduling strategies** in two aspects
  - **Callback priority assignment**
  - **Chain-aware node allocation algorithm**

Algorithm 1 Callback priority assignment

Input: Γ: chains

1: $\Gamma \leftarrow$ sort in ascending order of semantic priority π_Γ
2: $p \leftarrow 1$ ▷ Initialize current priority
3: **for all** $\Gamma^c \in \Gamma$ **do**
4:     **for all** $\tau_i \in \Gamma^c$ **do**
5:         $\tau_i \leftarrow p$
6:         $p \leftarrow p + 1$
7:     **end for**
8: **end for**

Chain-aware node allocation

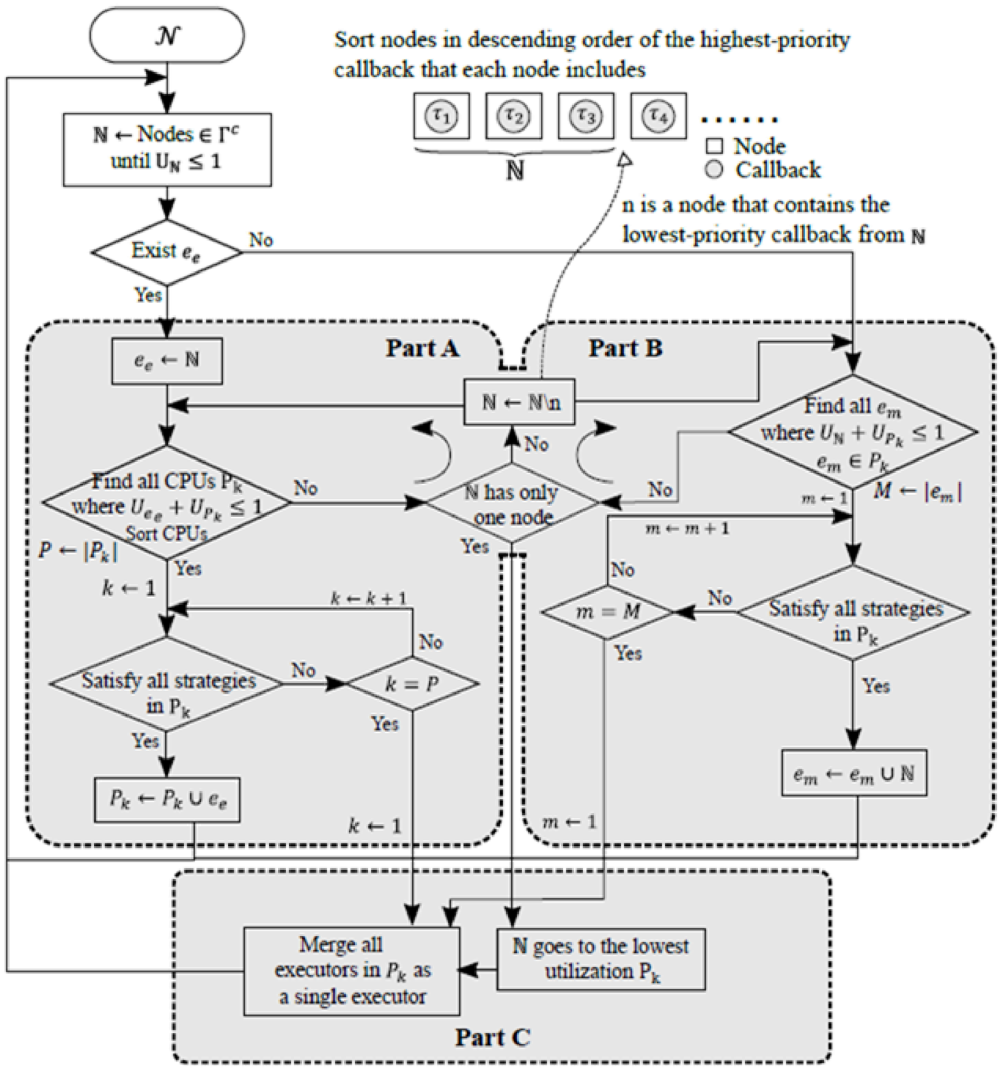
- Purpose: **minimize interference between chains**

(1) allocate given ***nodes to executors***, and then

(2) maps ***executors to available CPU cores***

*** Allocate all nodes associated to one chain to the same CPU core whenever it is possible**

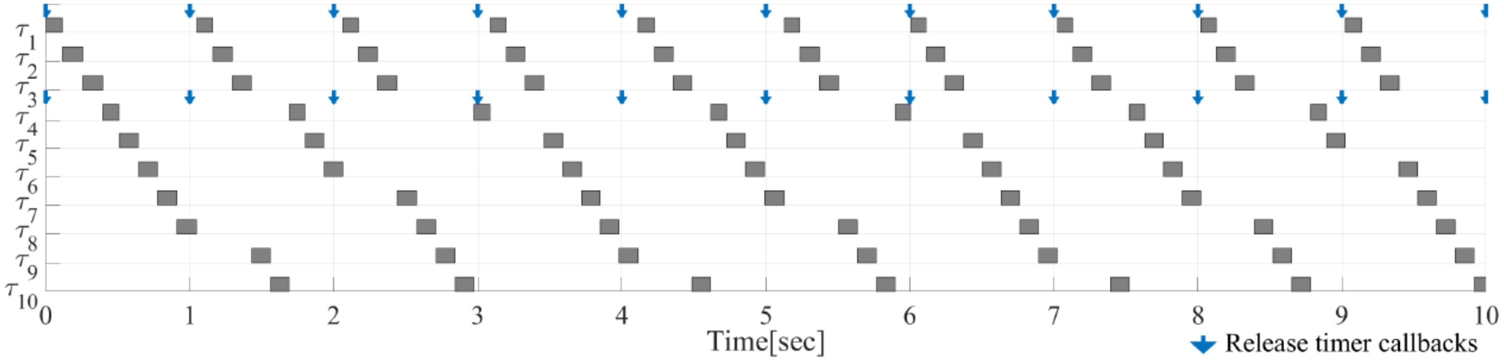
➡ **Reduce end-to-end latency of critical chains!**



	Parameters		
$\mathcal{N}$	Nodes	e_m	Non-empty executors
$\mathbb{N}$	A node set consists of callbacks of a chain Γ^c ($U_{\mathbb{N}} \le 1$)	M	The number of e_m
e_e	Empty executor	P	The number of P_k
U_{P_k}	Utilization of CPU core P_k		
n	A node that has the lowest priority callback of Γ^c in $\mathbb{N}$		

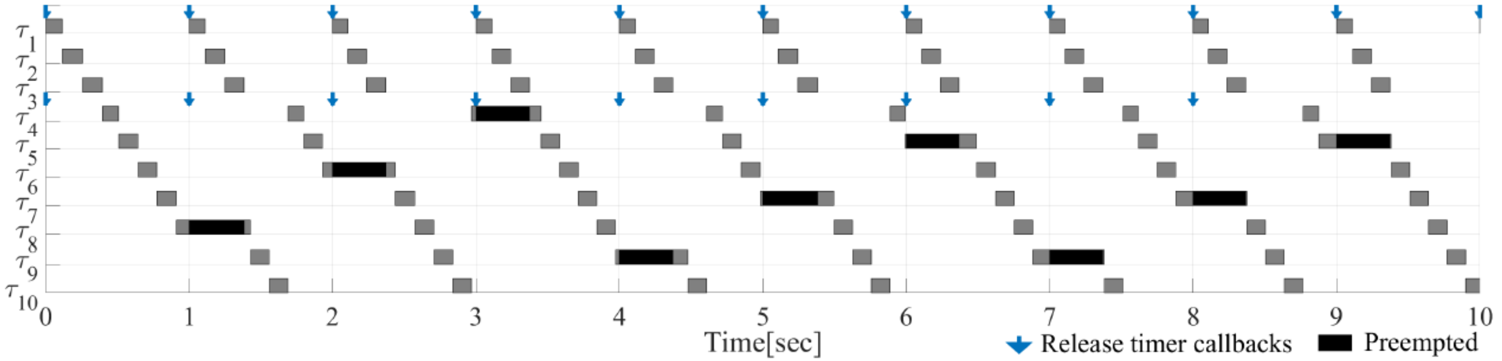
Examples of chain-aware scheduling

- With the same workload at challenges slide.



< All callbacks in a single executor >

Single executor	Mean	Max	Min	STD
Chain 1	0.436	0.506	0.368	0.038
Chain 2	1.196	1.738	0.741	0.348



< One executor per chain >

Executor per chain	Mean	Max	Min	STD
Chain 1	0.369	0.394	0.366	0.004
Chain 2	1.255	1.731	0.737	0.352

Significantly improved end-to-end latency under PiCAS

Analysis of end-to-end latency

- Latency analysis in a multi-core system
  - Segment Φ_i: a subset of a chain on one CPU core
  - Multiple segments if a chain executes over multiple CPU cores

Step 1: Computing the WCRT of each segment of a chain

WCRT of a segment Φ_i, $R^n_{c,i}$

Step 2: Adding the WCRT of all segments of the chain

End-to-end latency of a chain, L_{Γ^c}

< Latency analysis of a chain in a multi-core system >

Execution time of callbacks for the segment

$$R^{n+1}_{c,i} \leftarrow \boxed{B_i} + \boxed{\sum_{\forall j:\tau_j \in \Phi_i} C_j} + \boxed{\sum_{\substack{\forall k:\tau_k \in e(\Phi_i) \vee \\ \tau_k \in e_{HP}}} \eta_i\left(R^n_{c,i}, \tau_k\right) \times C_k}$$

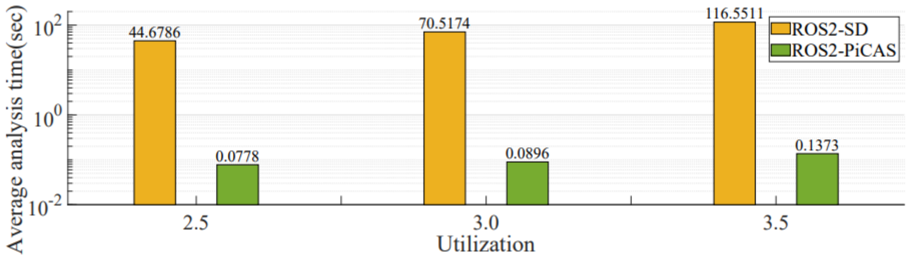
Blocking time from lower priority callback

Interference from higher semantic priority chains

$$L_{\Gamma^c} = \sum_{\Phi_i \subset \Gamma^c} R^n_{c,i} + \boxed{S(\Gamma^c)}$$

Blocking delay by prior instance

Reduce analysis running time!

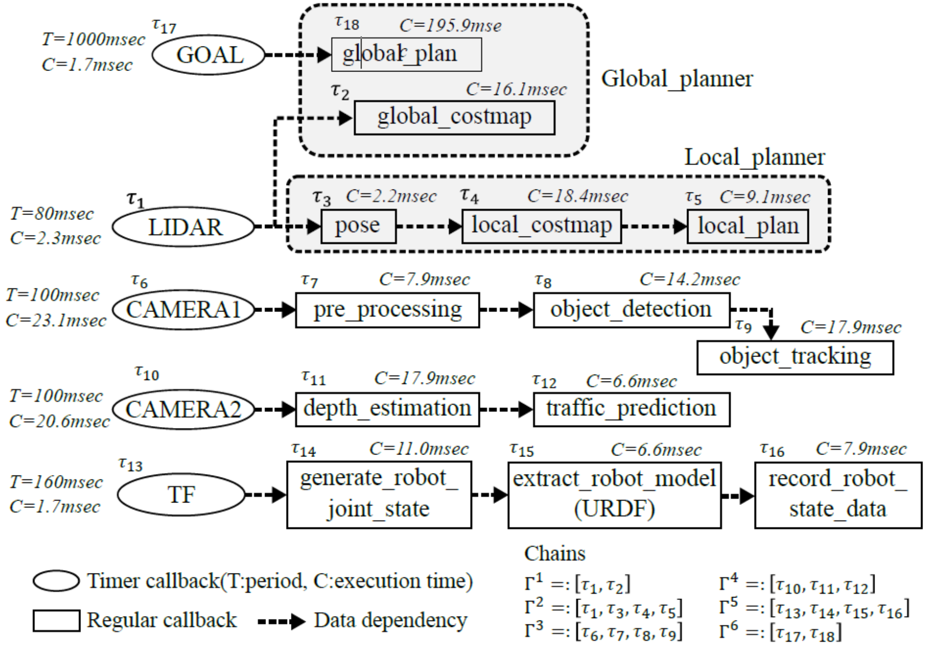


V. Evaluation

Evaluation

- Experimental setup for case study
  - Implemented in the Eloquent/Foxy/Galactic versions on Ubuntu18.04 on NVIDIA Xavier NX and Raspberry Pi 4.
  - Comparison of approaches
    - ✓ **ROS2-SD** † : **ROS2 default scheduler** with resource reservation and **WCRT analysis**
    - ✓ **ROS2-PiCAS** : proposed scheduler with end-to-end latency analysis
  - Case study in a multi-core system
    - ✓ Inspired by the indoor self-driving stack of F1/10 vehicle
    - ✓ 6 real-time chains (18 callbacks) and 6 best-efforts chains in a 4-core system
    - ✓ Low-indexed chains are more critical chains

< F1TENTH >

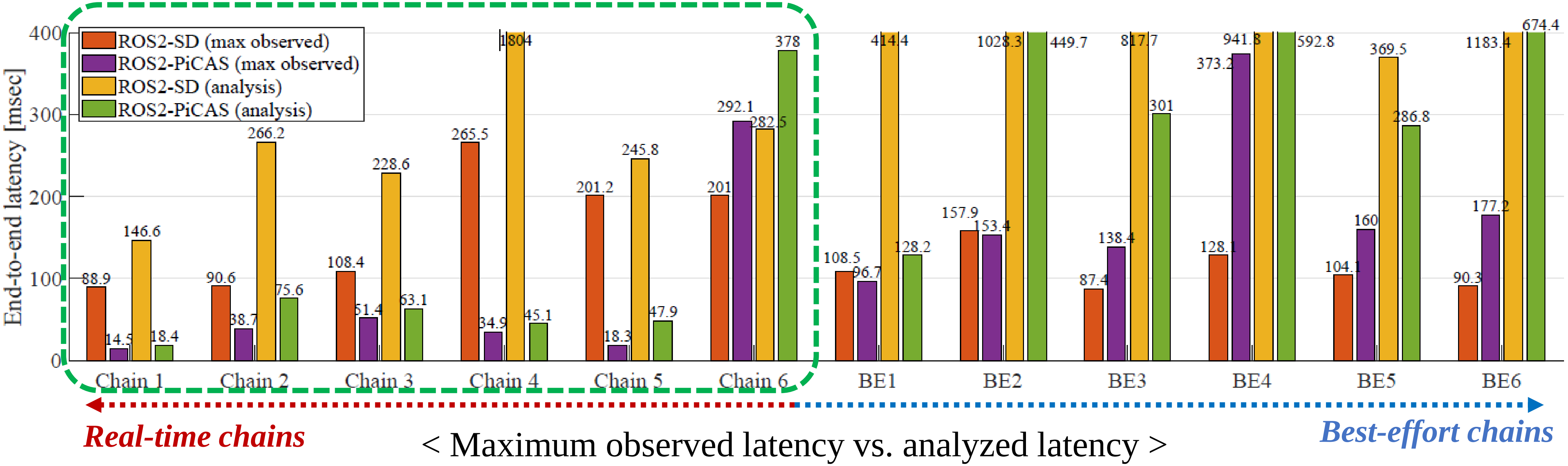


< Case study >

† D. Casini et al. "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", ECRTS, 2019

V. Evaluation

Case study I



< Maximum observed latency vs. analyzed latency >

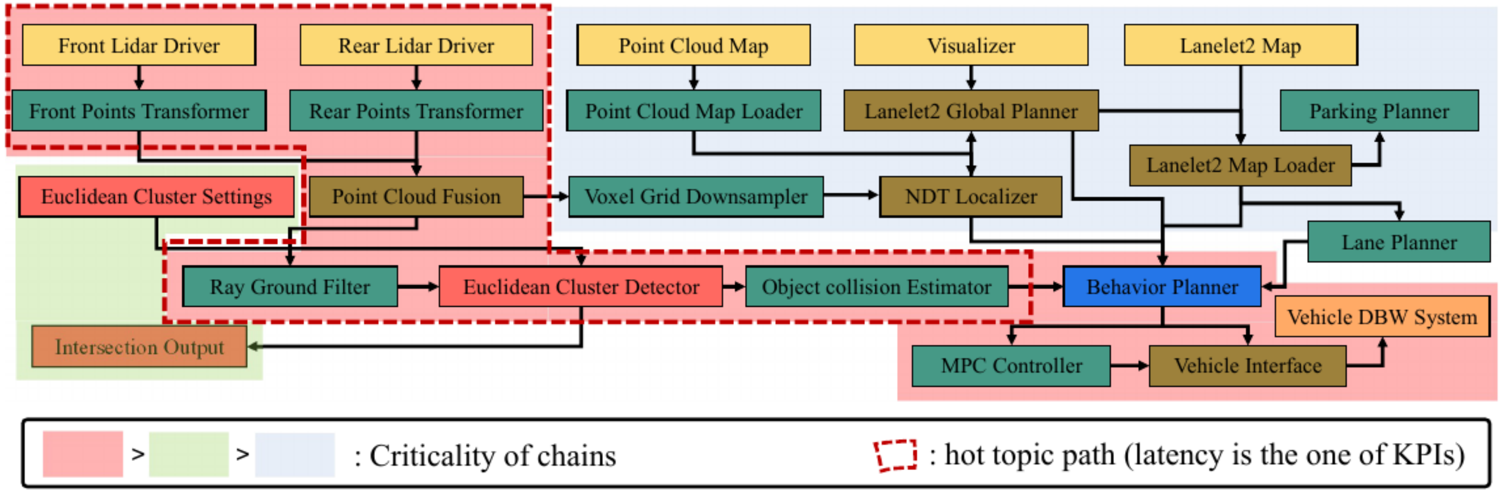
PiCAS schedules chains while respecting their semantic priorities.

Our latency analysis provides tighter upper-bounds for real-time chains.

V. Evaluation

Case study II

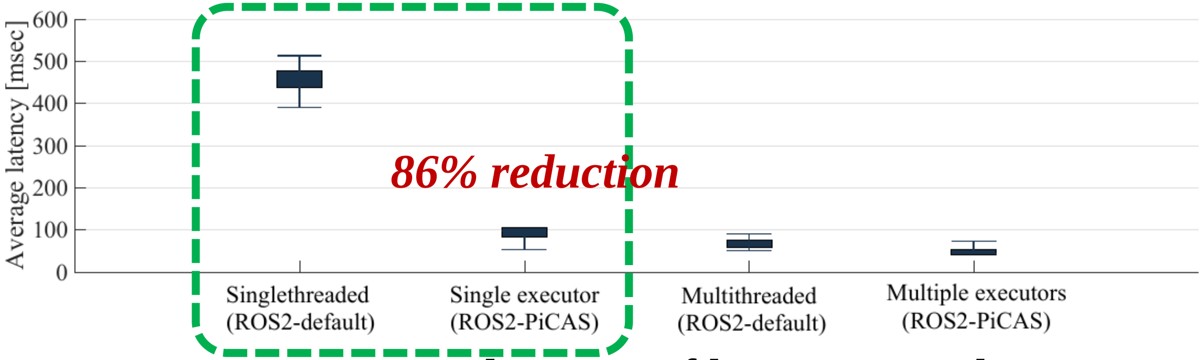
- Integration to *Reference system* †



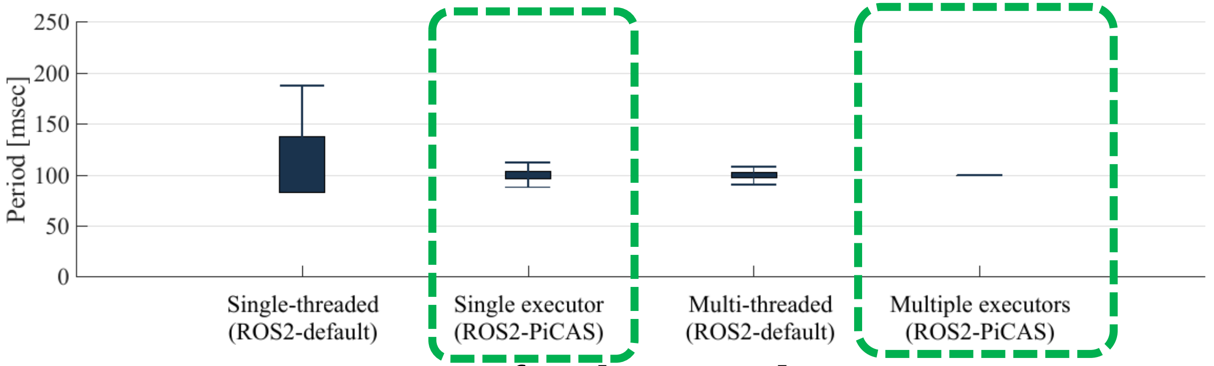
< Chain configuration of Autoware model >

ROSCon 2021, Real-Time Executor Workshop

† https://github.com/rtenlab/reference-system



< Average latency of hot topic path >



< Jitter of Behavior Planner >

Conclusion & Future work

- Conclusion
  - Proposed **a priority-driven chain-aware scheduling** and its **end-to-end latency analysis** framework
  - New design of ROS2 scheduling includes **scheduling strategies**, **priority assignment of callbacks**, and **chain-aware node allocation**
  - ROS2-PiCAS **outperforms** the existing ROS2 scheduling w.r.t. the end-to-end latency **under practical scenarios**
- Future work
  - Evaluate PiCAS to more complex scenario, e.g., autoware.auto (built on ROS2)
  - Support multi-threaded executor of ROS2

Thank you

PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2

- Hyunjong Choi, Yecheng Xiang, Hyoseung Kim

Q & A

PiCAS source: https://github.com/rtenlab/ros2-picas
PiCAS on reference system: https://github.com/rtenlab/reference-system
PiCAS paper: https://intra.ece.ucr.edu/~hyoseung/pdf/rtas21_picas.pdf