RTAS 2021

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

Hyunjong Choi, Yecheng Xiang, Hyoseung Kim

UC RIVERSIDE

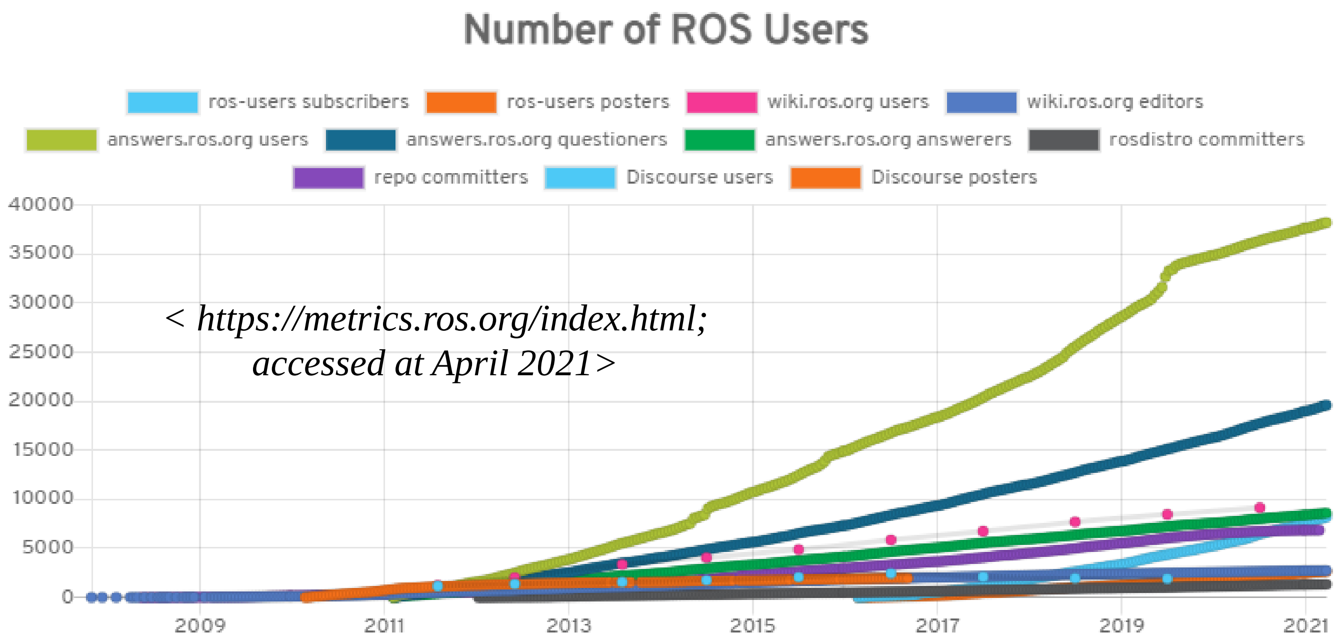
RTEN Real-time Embedded and Networked Systems Laboratory

I. Introduction

# Robot Operating System (ROS)

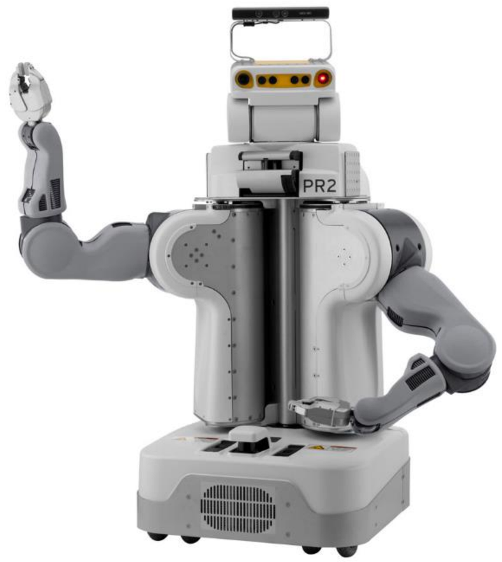
- ROS (since 2007)
  - Popular open-source middleware in academia and industry
  - Provides software tools, robot systems, and best-practices

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 April 2021>*

ROS 1.0
http://ros.org/



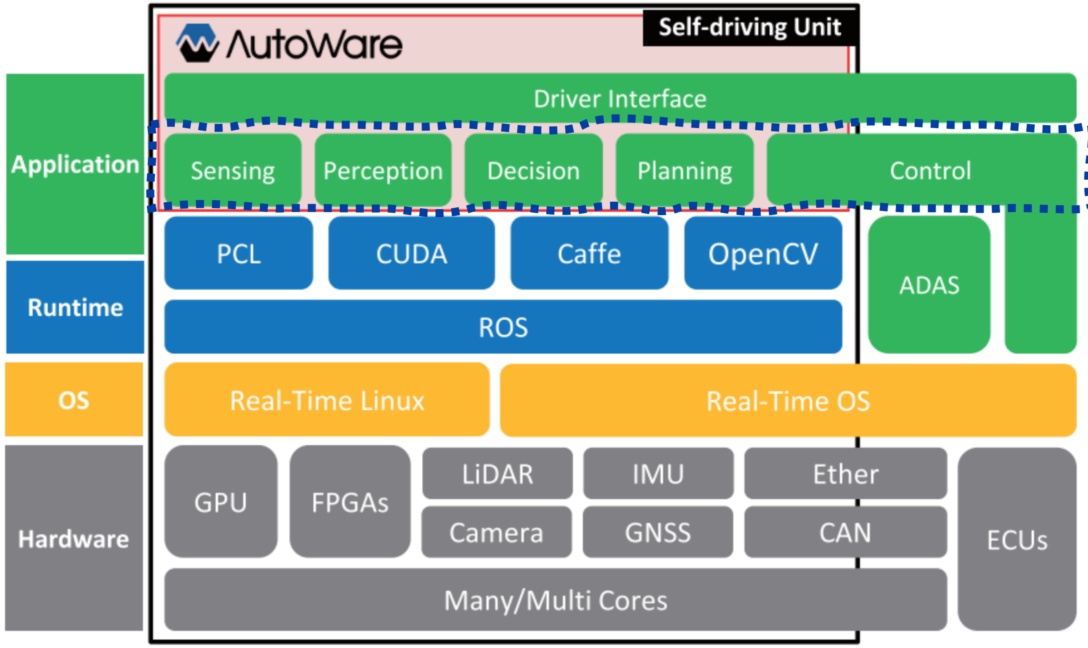

Willow Garage PR2 (original ROS robot)

*http://willowgarage.com*

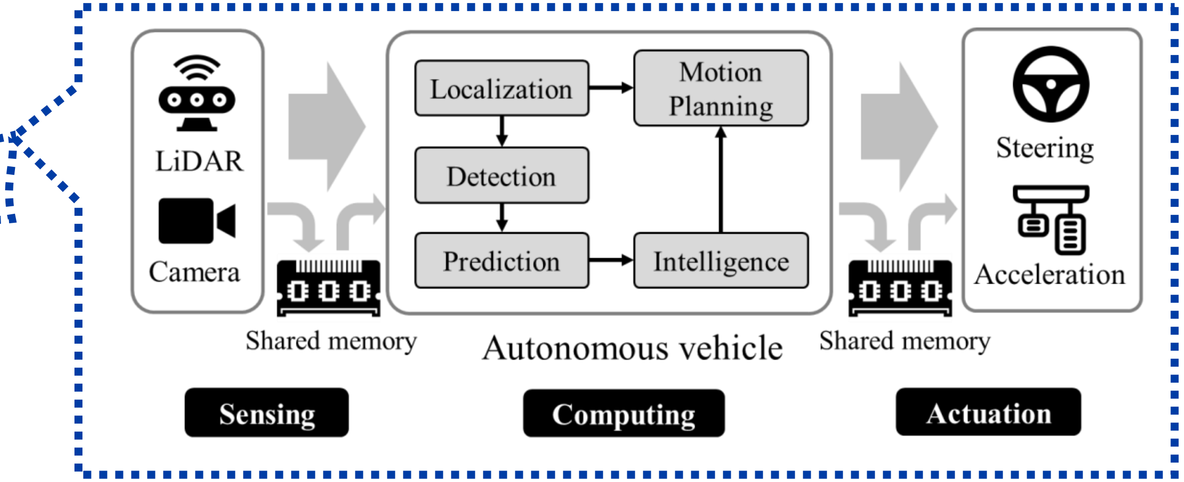
I. Introduction

# Why real-time in ROS ?

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




< Autoware.ai > †




< Chain in self-driving application >

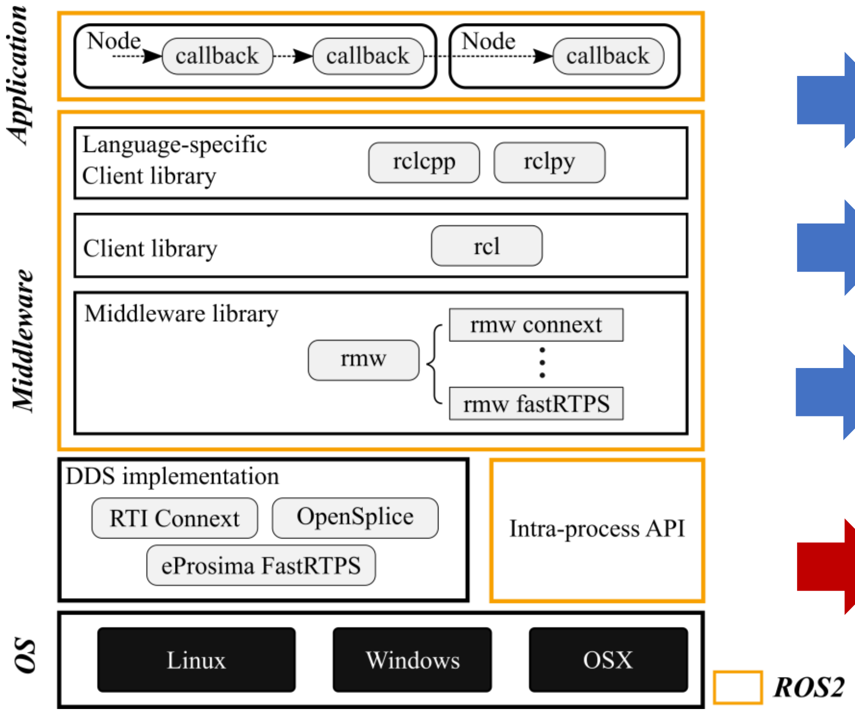
**Timing constraint violations (e.g., end-to-end latency) can cause catastrophic accidents**

†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 >

**Suffers from priority inversion**

**Too complex and pessimistic to analyze**

**No systematic resource allocation policy**

**Needs a new RT scheduler for ROS2 !**

Ardent Apalone, released Dec 2017

Eloquent Elusor, released Nov 2019

# Contributions

- We propose **a new priority-driven chain-aware scheduler** for ROS2 in a multi-core environment (PiCAS)
- We develop **analysis to upper-bound the end-to-end latency of chains** under the proposed PiCAS framework
- We **implement PiCAS** in the Eloquent Elusor version of ROS2 on an **embedded platform** (NVIDIA Xavier NX)
- PiCAS outperforms the default ROS2 scheduler and the latest analysis work in terms of **end-to-end latency**

# Scheduling-related abstractions in ROS2

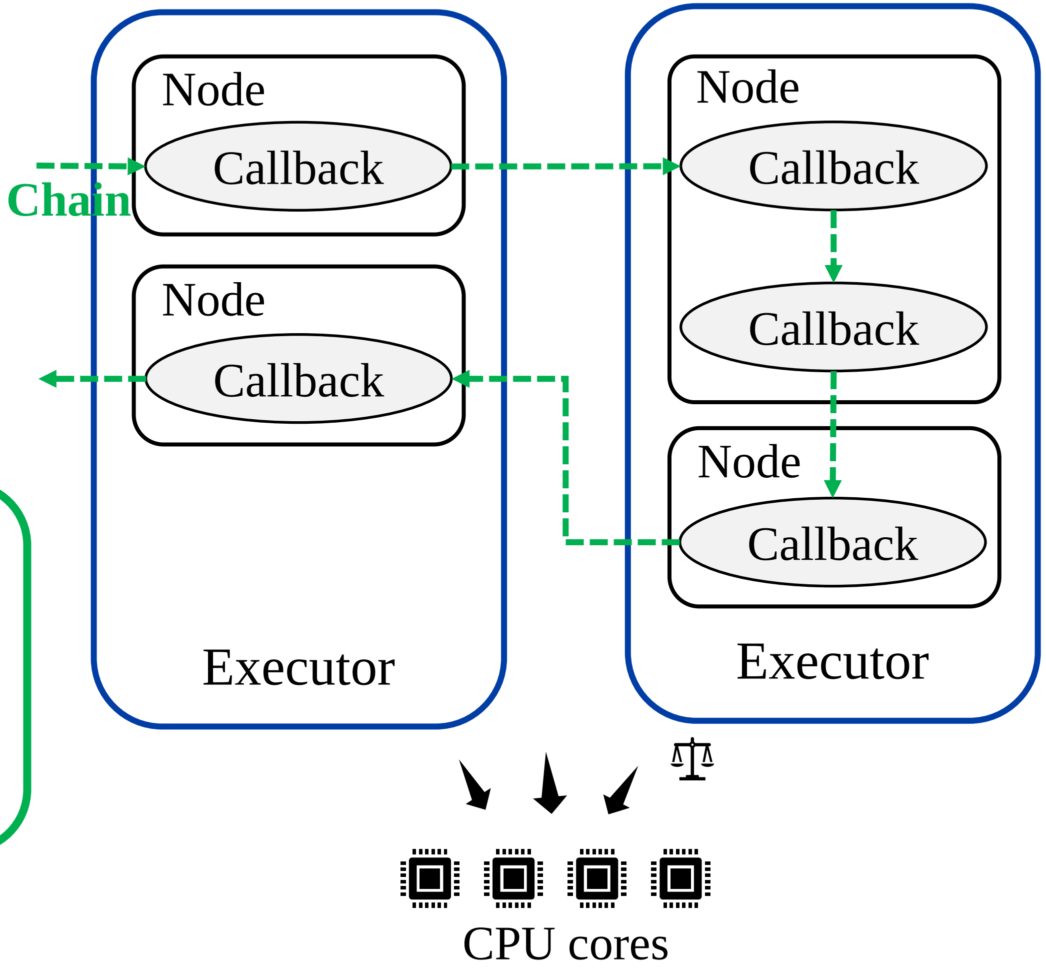
- *Callbacks*, *nodes*, and *executors*

## Callback model

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

## Chain model

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




## Node model

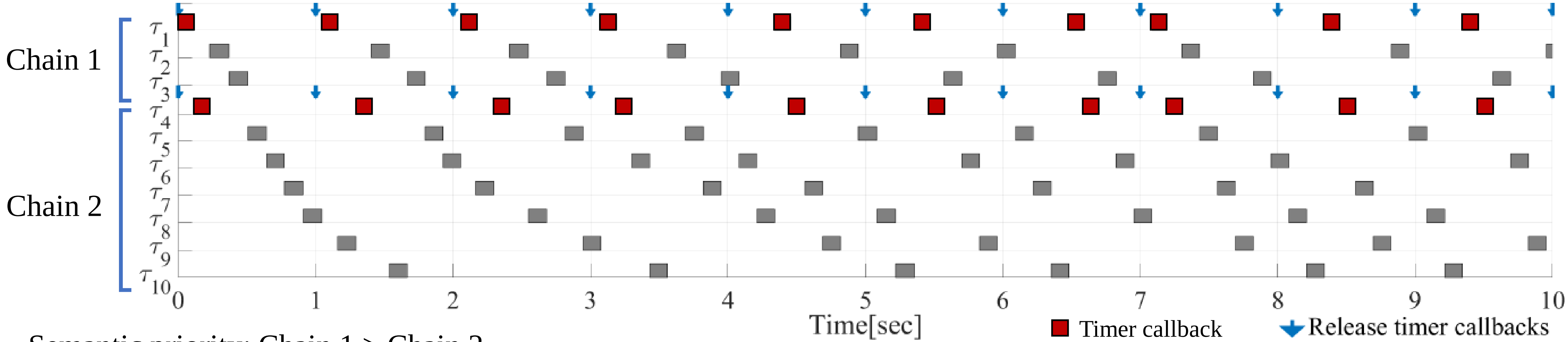
- ✓ $\mathcal{N} =: \{n_1, \dots, n_j, \dots, n_N\}$
- ✓ Not schedulable entities

## 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***




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**

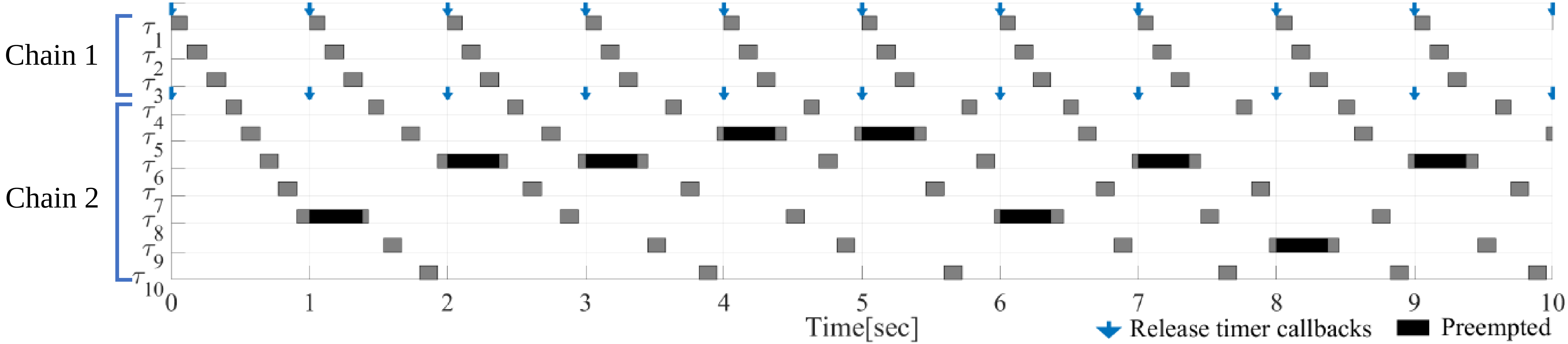
**Jeopardizes the timeliness of safety-critical chains**

[†] 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

*(Chain 1 in a higher-priority executor)*




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**

# Priority-driven chain-aware scheduling

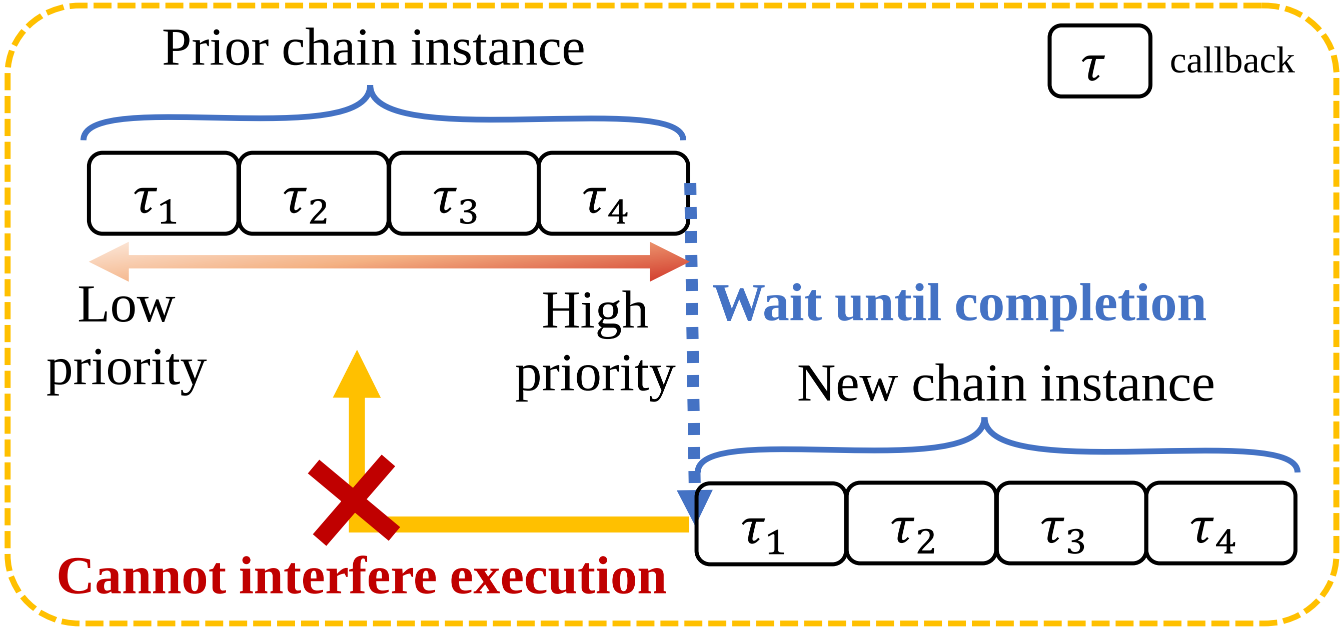
- Re-design ROS2 default scheduling architecture
  - (1) Higher-semantic priority chain executes first (from challenge I)
  - (2) For each chain, its instances on the same CPU execute in arrival order to prevent self-interference (from challenge II)

**Lemma 1**

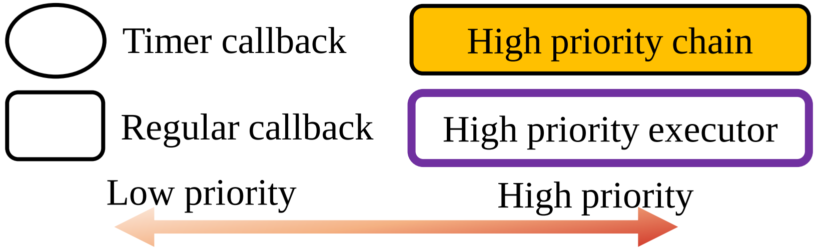
For $\Gamma^c := [\tau_1, .., \tau_i, \dots, \tau_j, \dots, \tau_N]$ whose callbacks are on the same CPU, ***a prior chain instance is guaranteed to complete,*** if the following conditions are met:

① $\boldsymbol{\tau_j}$ **has a higher callback priority** than $\tau_i$,

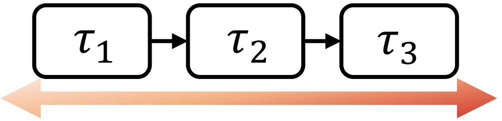
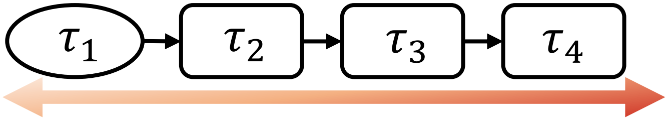
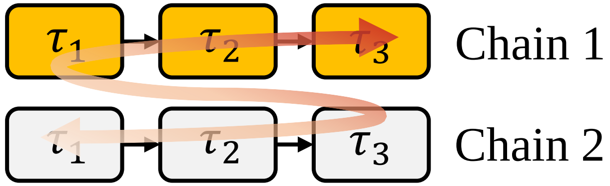
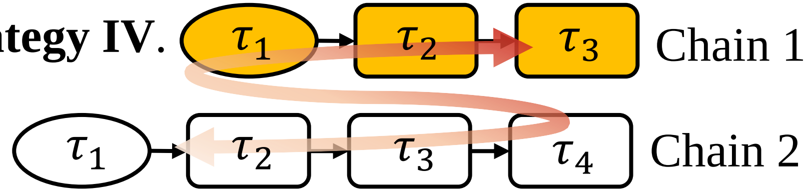
② $\boldsymbol{\tau_j}$ **runs on an executor with the same or higher priority** than $\tau_i$'s executor.



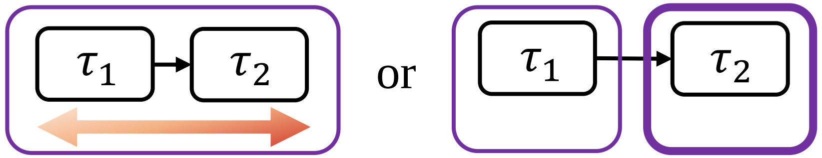
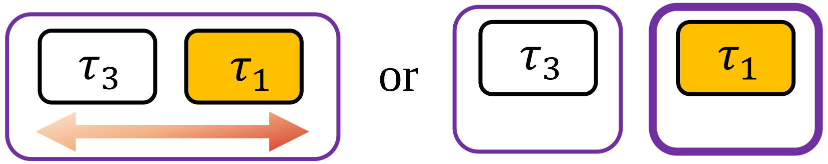
# Scheduling strategies




- Strategies for chains running within an executor

| | Regular callbacks only | Timer and regular callbacks |
|---|---|---|
| **Single chain** | **Strategy I.** (To satisfy ① of Lemma 1)   | **Strategy II.** (To satisfy ① of Lemma 1)   |
| **Multiple chains** | **Strategy III.**   | **Strategy IV.**   |

- Strategies for chains running across executors

| Single chain on one CPU | Multiple chains on one CPU |
|---|---|
| **Strategy V.** (To satisfy ② of Lemma 1)   | **Strategy VI.**   |

# Priority assignment

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

**Algorithm 1** Callback priority assignment

**Input:** $\Gamma$: chains

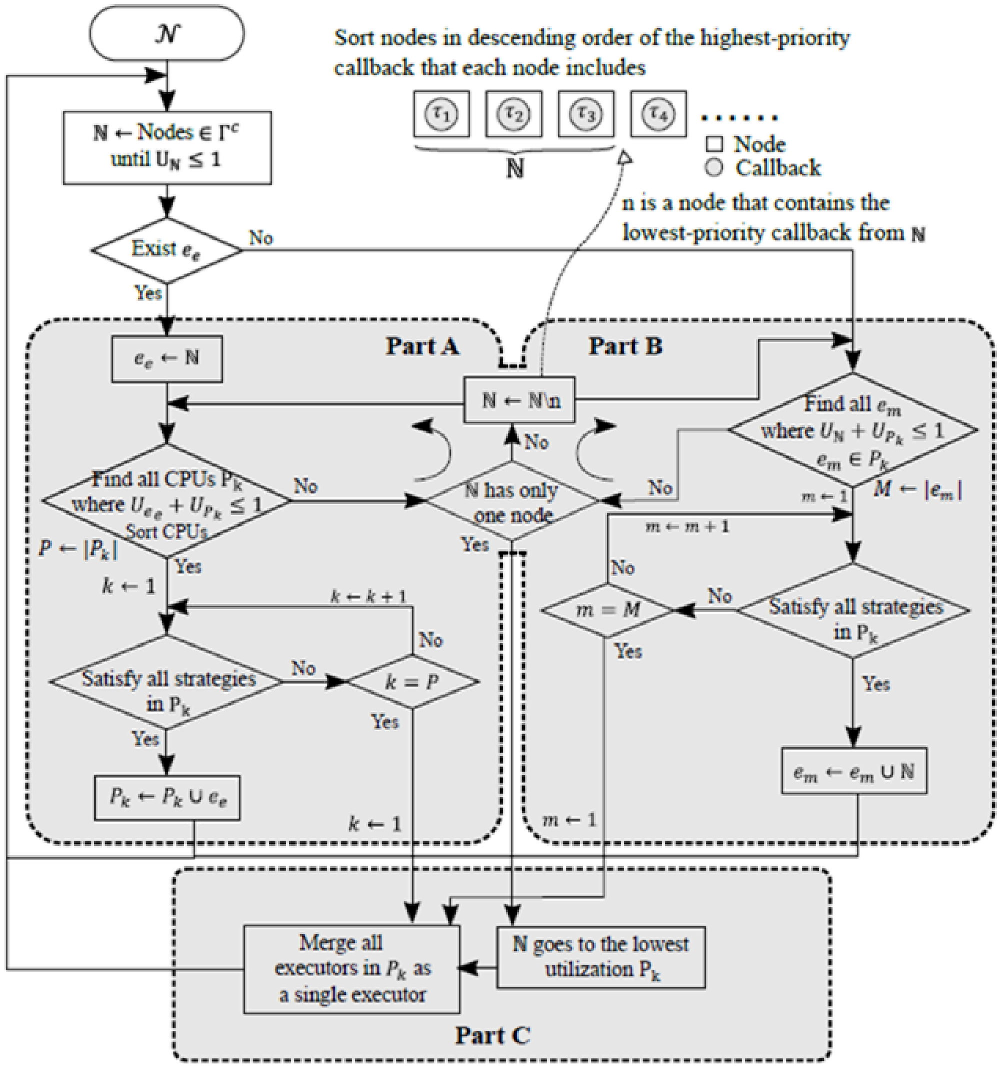
1: $\Gamma \leftarrow$ sort in ascending order of semantic priority $\pi_\Gamma$
2: $p \leftarrow 1$ ▷ Initialize current priority
3: **for all** $\Gamma^c \in \Gamma$ **do**
4:     **for all** $\tau_i \in \Gamma^c$ **do**
5:         $\tau_i \leftarrow p$
6:         $p \leftarrow p + 1$
7:     **end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




**Part A**. Allocate sorted nodes ℕ to $e_e$ and $e_e$ to a feasible CPU

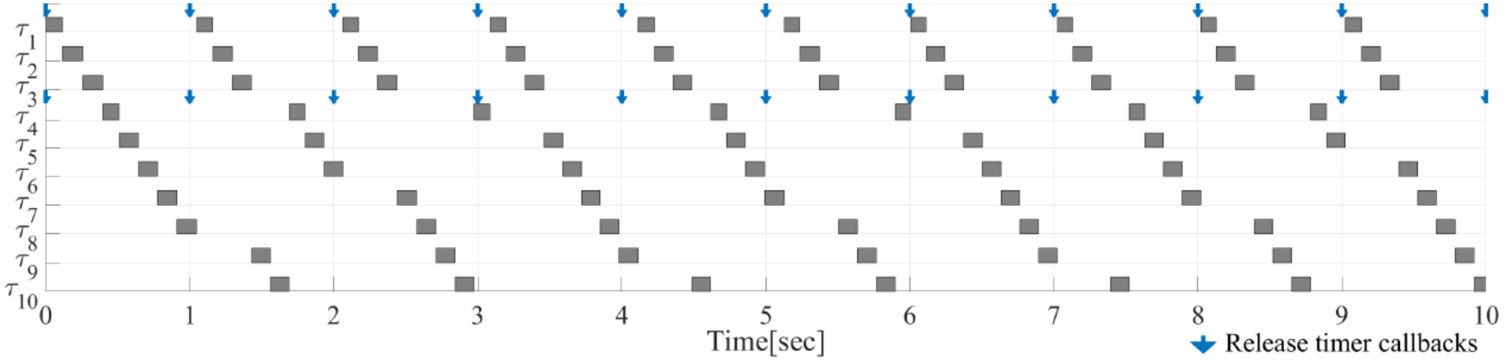
**Part B**. Allocate sorted nodes ℕ to feasible $e_m$ when $e_e$ does not exist

**Part C**. Handle all leftover nodes that were not allocated to executors by Part A & B

| | Parameters | | |
|---|---|---|---|
| $\mathcal{N}$ | Nodes | $e_m$ | Non-empty executors |
| ℕ | A node set consists of callbacks of a chain $\Gamma^c$ $(U_ℕ \leq 1)$ | 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 $\Gamma^c$ in ℕ | | |

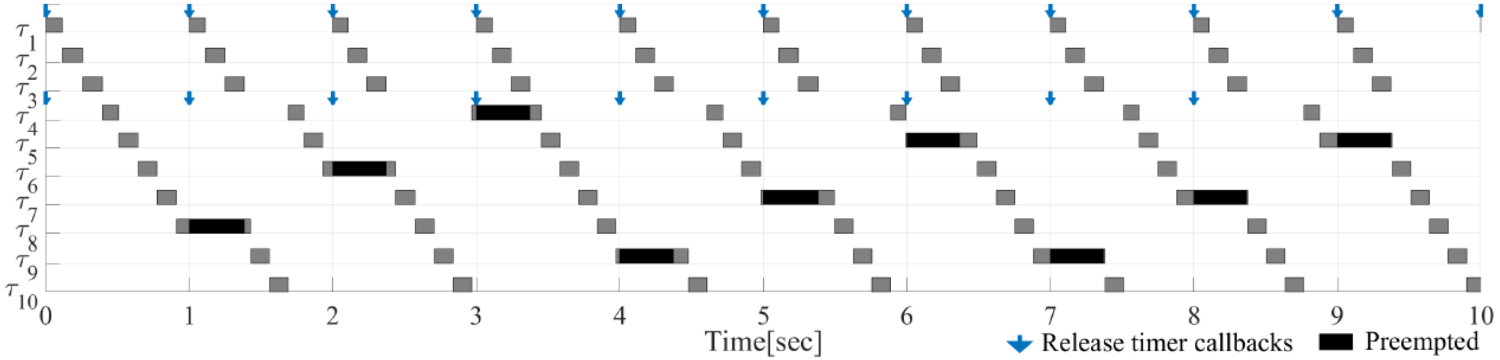
# Examples of chain-aware scheduling

- With the same workload at page 7.




< 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 $\Phi_i$: 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 $\Phi_i$, $R^n_{c,i}$

↓

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

End-to-end latency of a chain, $L_{\Gamma^c}$

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

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

- $B_i$: Blocking time from lower priority callback
- $\sum C_j$: Execution time of callbacks for the segment
- $\sum \eta_i(R^n_{c,i}, \tau_k) \times C_k$: Interference from higher semantic priority chains

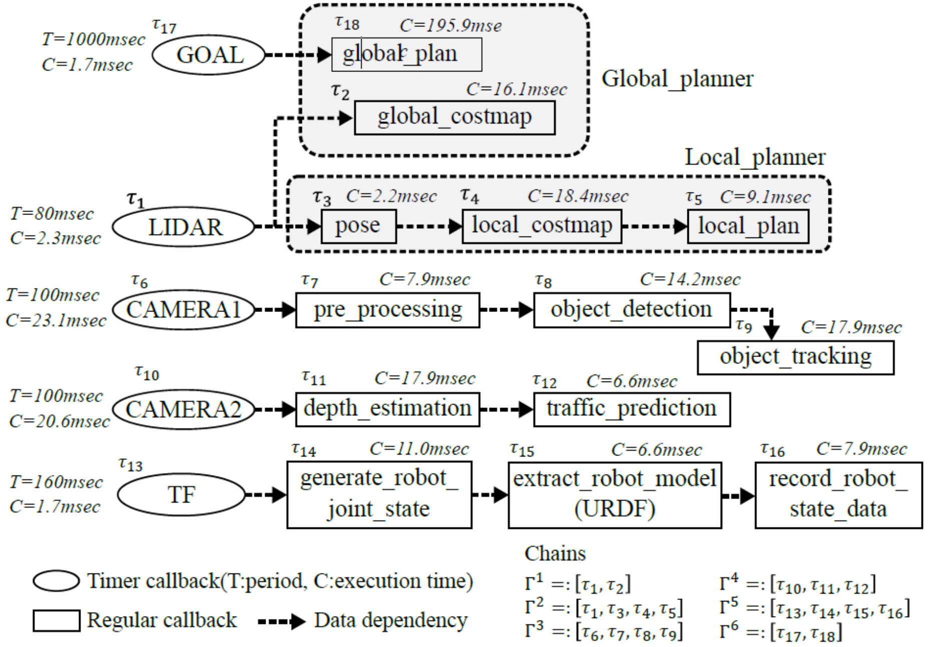
$$L_{\Gamma^c} = \sum_{\Phi_i \subset \Gamma^c} R^n_{c,i} + S(\Gamma^c)$$

- $S(\Gamma^c)$: Blocking delay by prior instance

V. Evaluation

# Evaluation

- **Case studies**, **schedulability analysis**, and **analysis running time**
- Experimental setup for case study
  - Implemented in the Eloquent Elusor of ROS2 on Ubuntu18.04 on NVIDIA Xavier NX
  - Comparison of approaches
    - ✓ **ROS2** : **ROS2 default scheduler** with no analysis
    - ✓ **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

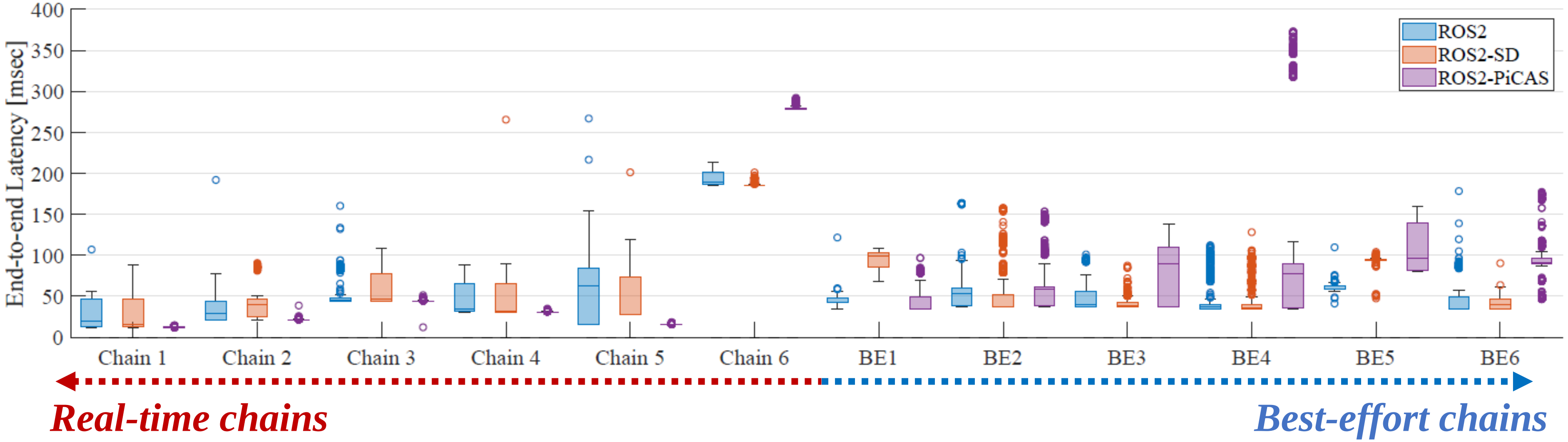



< Case study >

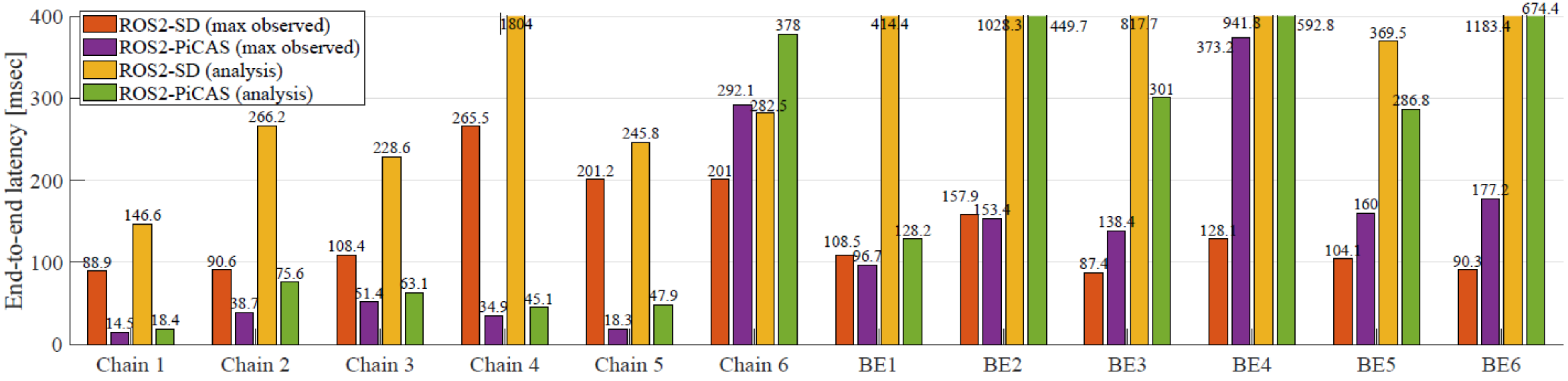
† D. Casini et al. "Response-time analysis of ROS 2 processing chains under reservation-based scheduling", ECRTS, 2019

V. Evaluation

# Case study




< End-to-end latency of chains >




< Maximum observed latency vs. analyzed latency >

PiCAS outperforms on most real-time chains (e.g., 14ms vs 88ms for chain 1)

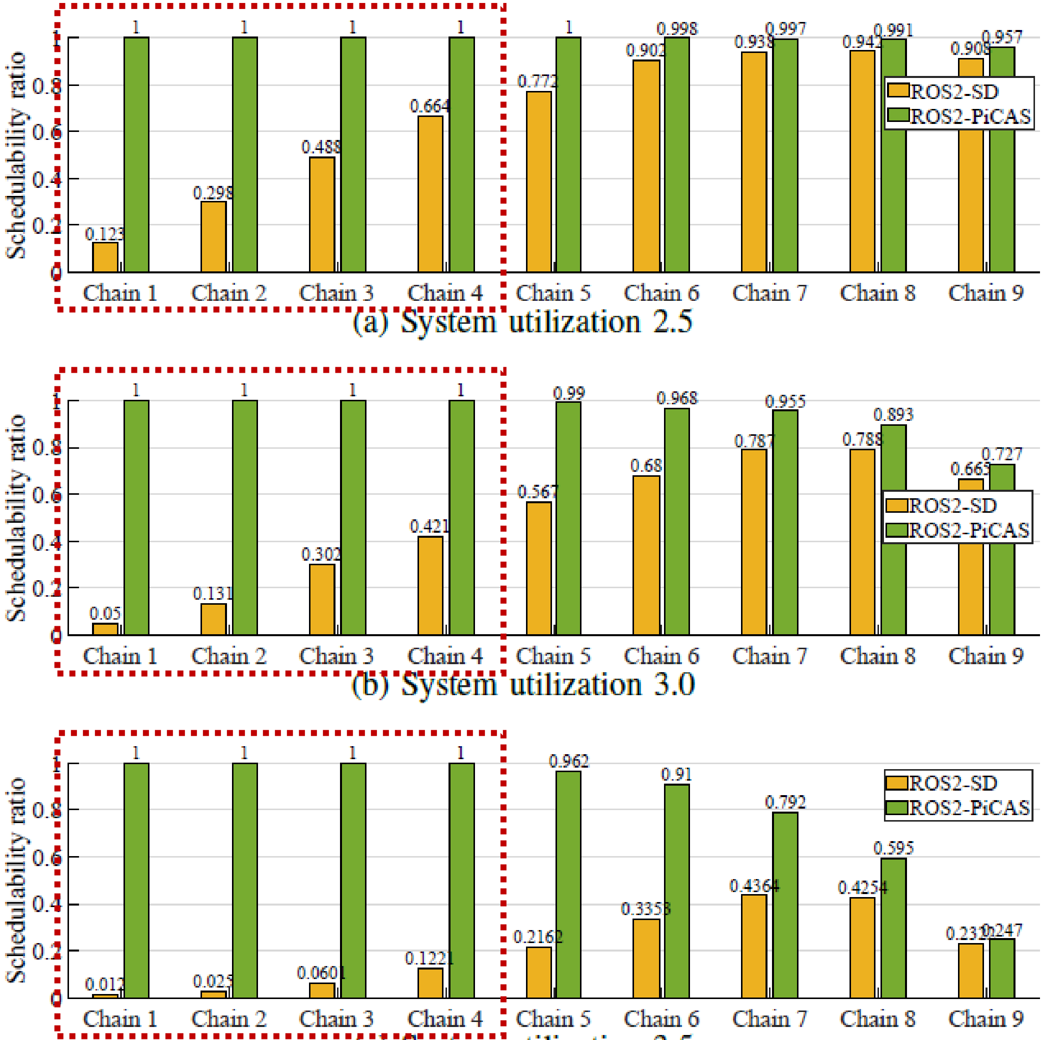
Schedules chains while respecting their semantic priority

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

# Schedulability experiments

- Workload generation
  - 1,000 randomly-generated workload sets of callbacks
  - Utilization from {2.5, 3.0, 3.5} for 4-core environment
  - 45 callbacks that forms 9 chains (i.e., 5 callbacks per chain)
  - Chain’s period (deadline) is chosen in the range [50, 1000] msec

*High-priority real-time chains*



(a) System utilization 2.5
(b) System utilization 3.0
(c) System utilization 3.5

**Schedulability ratio decreases as the utilization increase**

**ROS2-PiCAS outperforms ROS2-SD for all utilization setups.**

**ROS2-PiCAS prioritizes chains based on their semantic priority**

# 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
  - Deploy PiCAS to more complex scenario, e.g., autoware.auto (built on ROS2)

# Thank you

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

- Hyunjong Choi, Yecheng Xiang, Hyoseung Kim

# Q & A