Theory-Guided Adaptive Scheduling for ROS 2

Daniel Enright, Hoora Sobhani, and Hyoseung Kim

University of California, Riverside

UC RIVERSIDE

RTEN Real-time Embedded and Networked Systems Laboratory

Why ROS 2?

ROS 2 TSC

ROS Industrial

78 Members

bit.ly/ROSIMembers

Government

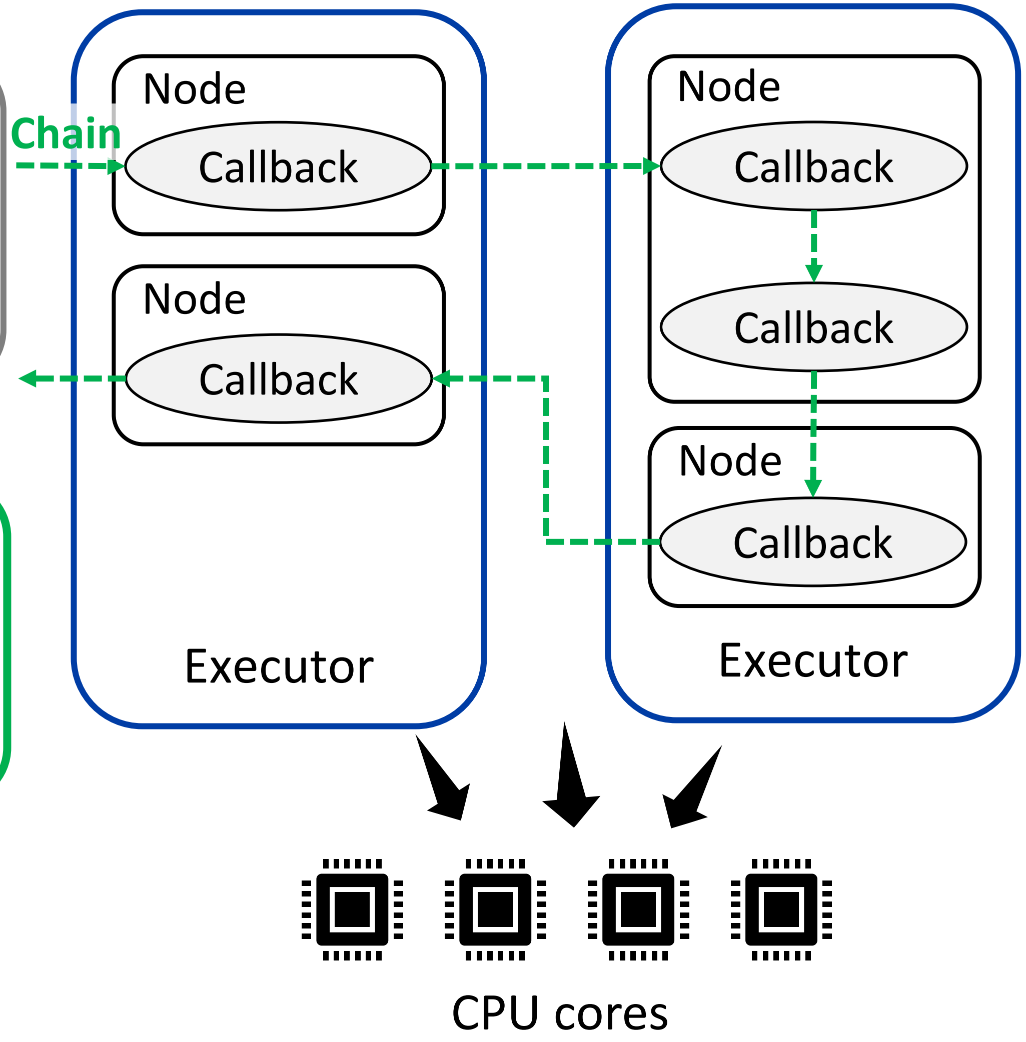
- NASA
- NHSTA / USDOT
- DARPA
- Army / Navy / AF
- NIST
- Dozens of Universities
- Singapore Hospital System

Credit to Katherine Scott, Open Robotics.
https://opencv.org/wp-content/uploads/2021/06/Katherine-Scott-OpenCVWebinar-6242021.pdf

ROS 2 Background

Callback
- ✓ Smallest schedulable unit
- ✓ Triggered by timer, message arrival, etc.
- ✓ Non-preemptable

Processing Chain
- ✓ A sequence of callbacks
- ✓ Must execute in a specific order due to data dependencies



Node
- ✓ Group of callbacks
- ✓ Not a schedulable unit

Executor
- ✓ Executes callbacks
- ✓ Either a single-thread or multi-thread process scheduled by OS

UC RIVERSIDE

Real-Time Research on ROS 2

- Timing analysis
  - Chain response time and schedulability on single-thread executors [1-3] and multi-thread executors [4-5]
- Framework improvements
  - Priority-based scheduling [3]
  - Accelerator support [6]
  - Starvation prevention [7]

[1] Casini, D., Blaß, T., Lütkebohle, I., Brandenburg, B.: Response-time analysis of ROS 2 processing chains under reservation-based scheduling. In: Euromicro Conference on Real-Time Systems (ECRTS) (2019)
[2] Tang, Y., Feng, Z., Guan, N., Jiang, X., Lv, M., Deng, Q., Yi, W.: Response time analysis and priority assignment of processing chains on ROS2 executors. In: IEEE Real-Time Systems Symposium (RTSS) (2020)
[3] Choi, H., Xiang, Y., Kim, H.: PiCAS: New design of priority-driven chain-aware scheduling for ROS2. In: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2021)
[4] Jiang, X., Ji, D., Guan, N., Li, R., Tang, Y., Wang, Y.: Real-Time Scheduling and Analysis of Processing Chains on Multi-threaded Executor in ROS 2. In: RTSS (2022)
[5] Sobhani, H., Choi, H., Kim, H.: Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2023)
[6] Enright, D., Xiang, Y., Choi, H., Kim, H.: PAAM: A Framework for Coordinated and Priority-Driven Accelerator Management in ROS 2. In: 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2024)
[7] Teper, H., et al.: Thread Carefully: Preventing Starvation in the ROS 2 Multi-Threaded Executor. In: 2024 International Conference on Embedded Software (EMSOFT) (2024)

How is ROS 2 timing analysis done?

- Most existing analyses rely on supply-bound functions to characterize executor's guaranteed resource availability
  - e.g., executor thread $r_k = (C_k^r, T_k^r)$
    - C_k^r = budget, T_k^r= period
- In Linux, we can use SCHED_DEADLINE
  - OS-level resource reservation and enforcement

```
/* This creates a 200ms / 1s reservation */
attr.sched_policy   = SCHED_DEADLINE;
attr.sched_runtime  =  200000000;
attr.sched_deadline = attr.sched_period = 1000000000;
```

https://lwn.net/Articles/743946/

Problems

- Little attention on resource allocation
  - Existing analysis work assumes resource allocation is *predetermined* by system designers
    - How many executor threads to use?
    - How much budget should be allocated to individual threads?
    - Which chains/callbacks should be assigned to which executors?

- Guarantees may break when deployed in dynamic environments
  - For example:
    - CPU frequency throttling due to thermal/power constraints
    - Changes in sensor data arrival rate

Prior Work

PiCAS [1]

- Explicit **priority-based** scheduling
  - Best-effort (BE) chains may starve
- **Static** resource allocation
  - Node to executor assignment
  - Executor priority assignment (SCHED_FIFO)

ROS-Llama [2]

- Standard ROS 2 scheduling (**fair share**)
  - Real-time (RT) and best-effort (BE) chains may interfere with each other
- **Dynamic** resource allocation
  - SCHED_DEADLINE budget adjustment

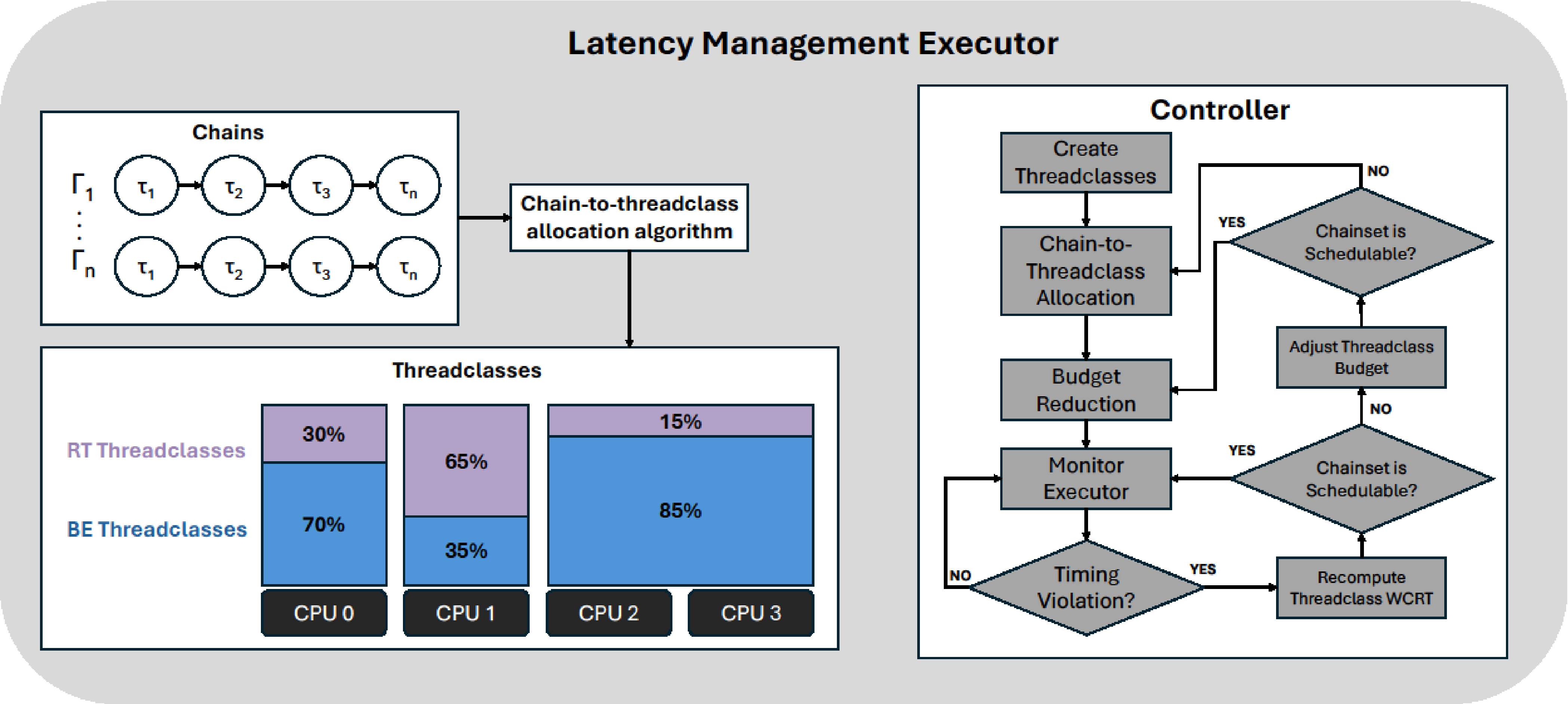
Limitations

1. **Single-threaded executors only**
   - Cannot run chains with utilization > 1.0 (or user must split them manually)
   - Earlier questions remain unanswered (e.g., # of threads, thread budget, etc.)
2. **No isolation** between RT and BE chains

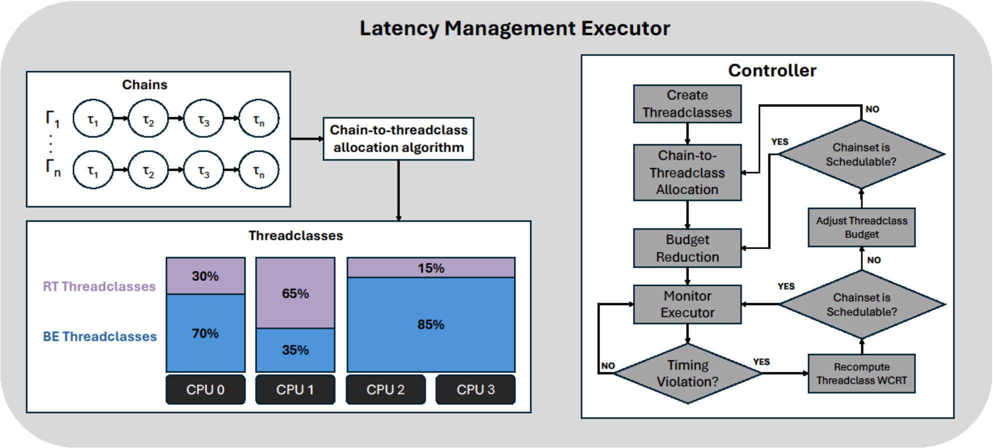
[1] Choi, H., Xiang, Y., Kim, H.: PiCAS: New design of priority-driven chain-aware scheduling for ROS2. In: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) (2021)
[2] Blass, T., Hamann, A., Lange, R., Ziegenbein, D., Brandenburg, B.B.: Automatic latency management for ROS 2: Benefits, challenges, and open problems. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2021)

Our Contributions

- **LaME: La**tency **M**anagement **E**xecutor



Our Contributions



- **LaME: La**tency **M**anagement **E**xecutor
  - Drop-in replacement for the existing multi-thread executor
- Threadclasses***:*** New abstraction for managing executor threads
- RT and BE workload isolation
  - RT chains: <u>priority-based</u> scheduling with deadline constraints
  - BE chains: <u>fair-share</u> scheduling (with starvation freedom when possible)
  - By leveraging existing ROS 2 multi-thread timing analysis*
- Adaptive resource controller
  - Dynamically adjusts resource and chain allocations

* Sobhani, H., Choi, H., Kim, H.: Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2023)

System Model

Callback $\boldsymbol{\tau_i} = (E_i, \pi_i)$

- E_i: WCET of callback τ_i
- π_i: Priority of τ_i within the executor

Chain $\boldsymbol{\Gamma_c} = ([\tau_{c1}, \tau_{c2}, \dots, \tau_{cn}], E_c, T_c, D_c, \zeta_c)$

- $[\tau_{c1} \dots]$: Sequence of callbacks
- E_c: Cumulative WCET of chain Γ_c
- T_c: Period of chain Γ_c
- D_c: Relative deadline of Γ_c ($D_c < T_c$)
- ζ_c: Criticality of chain Γ_c

Executor thread $\boldsymbol{r_k} = (C_k^r, T_k^r)$

- C_k^r: budget
- T_k^r: replenishment period

A set of executor threads
$\boldsymbol{\Pi} = (r_1, r_2, \dots, r_k)$

Existing executor and analysis:
- Executor has a single $\boldsymbol{\Pi}$
- All threads within $\boldsymbol{\Pi}$ use the same budget and period params

Threadclasses

- **Challenge:** Existing multi-thread executor creates one thread per core and treats all threads equally. Is it the best approach?

- **Our solution**
  - Create two threads per CPU core: RT and BE threads
  - Group subsets of threads and allow them to behave as *individual executors*

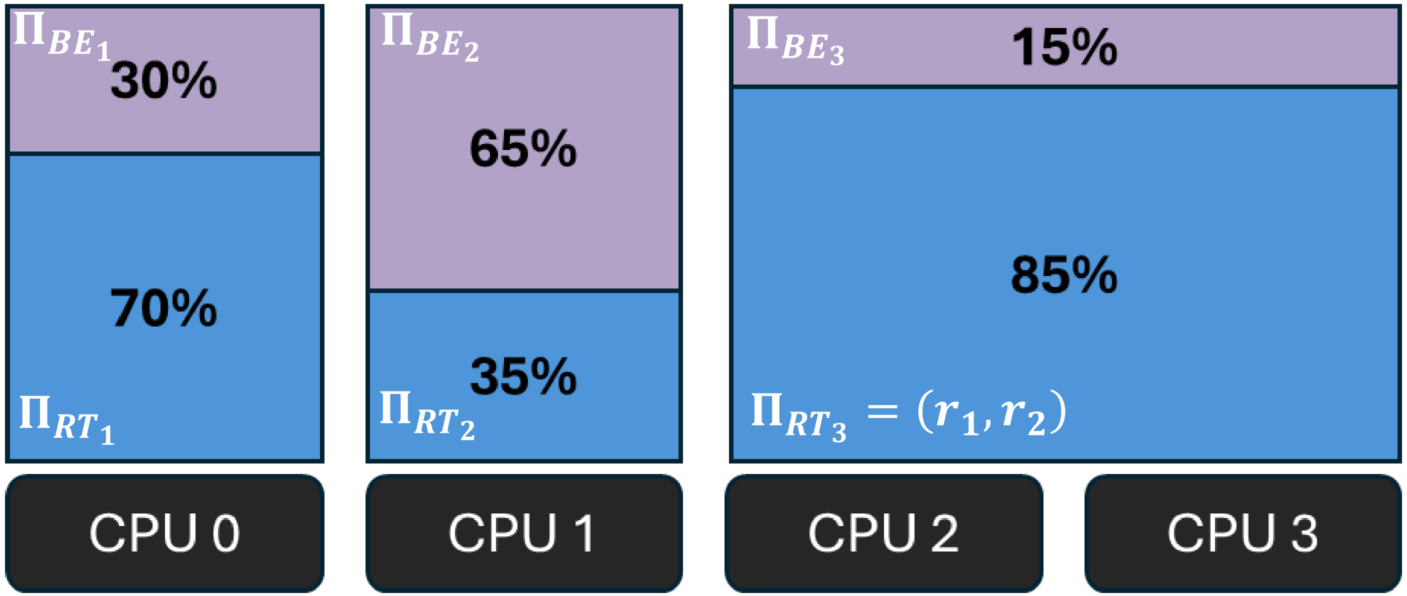
    Threadclasses

- Existing executor: $\{\ \Pi\ \}$, where $\Pi = (r_1, r_2, \ldots, r_k)$

- LaME executor: $\underbrace{\{\Pi_{RT_1}, \Pi_{RT_2}, \ldots\}}_{\text{RT threadclasses}} \cup \underbrace{\{\Pi_{BE_1}, \Pi_{BE_2}, \Pi_{BE_3}\}}_{\text{BE threadclasses}}$

RT threadclasses BE threadclasses

Threadclasses



BE Threadclasses:

- Executes callbacks in a **fairness-oriented** manner

RT Threadclasses:

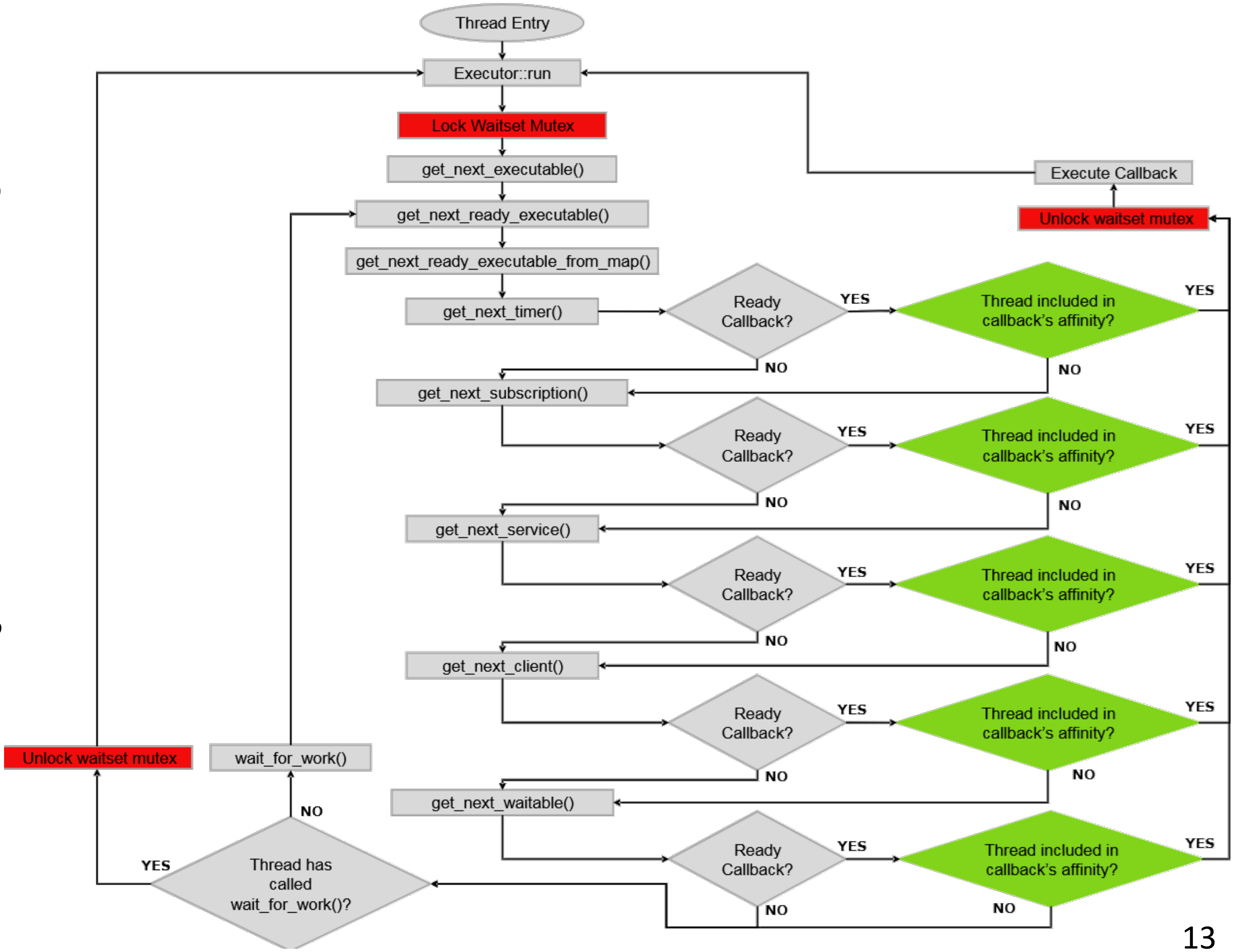
- Executes callbacks in a **priority-driven** manner
- Callback priority assignment by chain criticality

- No interference between threadclasses, thanks to SCHED_DEADLINE
- Allows both partitioned and constrained global callback scheduling within each threadclass

Callback-to-Threadclass Mapping

- **Challenge:** How can we assign callbacks or chains to specific threadclasses?

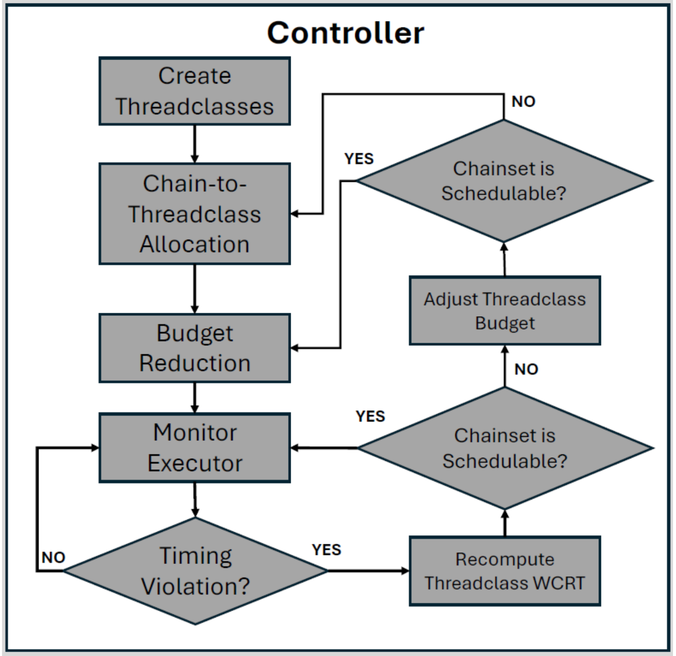
- **Our solution:** Affinity-based callback scheduling that restricts the threads that can execute specific callbacks



UC RIVERSIDE

Adaptive Resource Controller

- Runtime monitoring
  - Callback execution time: using thread-local timers (CLOCK_THREAD_CPUTIME_ID)
  - End-to-end chain response time: using timestamps
    - Both are monitored during chain execution paths
- Adaptive Resource Controller
  - Runs as a standalone thread with non real-time priority in Linux
  - By default, it activates periodically (e.g., 10s)
    - Performs resource adjustment when needed (e.g., changes in max. callback execution time)
    - Also triggered immediately when RT chain deadline is violated



Adaptive Resource Controller

- **Challenge:** Difficult to find optimal resource allocation at runtime
  - Due to inter-dependent choices: # of threadclasses, # of threads for each threadclass, budget for RT & BE threadclasses, chain assignment

- **Our solution:** Two-step approach

Chain-to-Threadclass Allocation:

- Assigns chains to Threadclasses based on their criticality class and a WFD heuristic.
- Merges Threadclasses to make otherwise unschedulable chains, schedulable.
- Ensures that all RT chains will be schedulable.

Dynamic Budget Reduction:

- Reduces the resources allocated to servicing RT chains.
- Remaining budget is allocated to BE Threadclasses on the same CPU cores.
- Monitors executor for timing violations and adjusts budgets accordingly at runtime.

Online Timing Analysis

- Performed by the resource controller

- For RT chains:
  - Check if WCRT $R_c \leq D_c$
    → Schedulability

Theorem 1 (from [19]). *The response time of a chain $\Gamma_c = [\tau_{c_1}, \tau_{c_2}, ..., \tau_{c_n}]$ with a* ***constrained deadline*** *on a* ***priority-driven*** *ROS 2 executor with $|\Pi|$ threads is upper-bounded by $R_c = \Delta + \overline{sbf}_k(E_{c_n} - 1)$, if $dbf(\Delta) < sbf_\Pi(\Delta)$ holds for the following demand bound function $dbf(\Delta)$:*

$$dbf(\Delta) = |\Pi| \cdot (E_c - E_{c_n}) + \sum_{\substack{\Gamma_x \in S_{RT} \setminus \{\Gamma_c\} \\ \wedge \pi_x > \pi_c}} W_x(\Delta, D_x - E_x) + \sum_{\forall \tau_l \in mlp(\tau_{c_1})} \min(E_l - 1, \Delta)$$

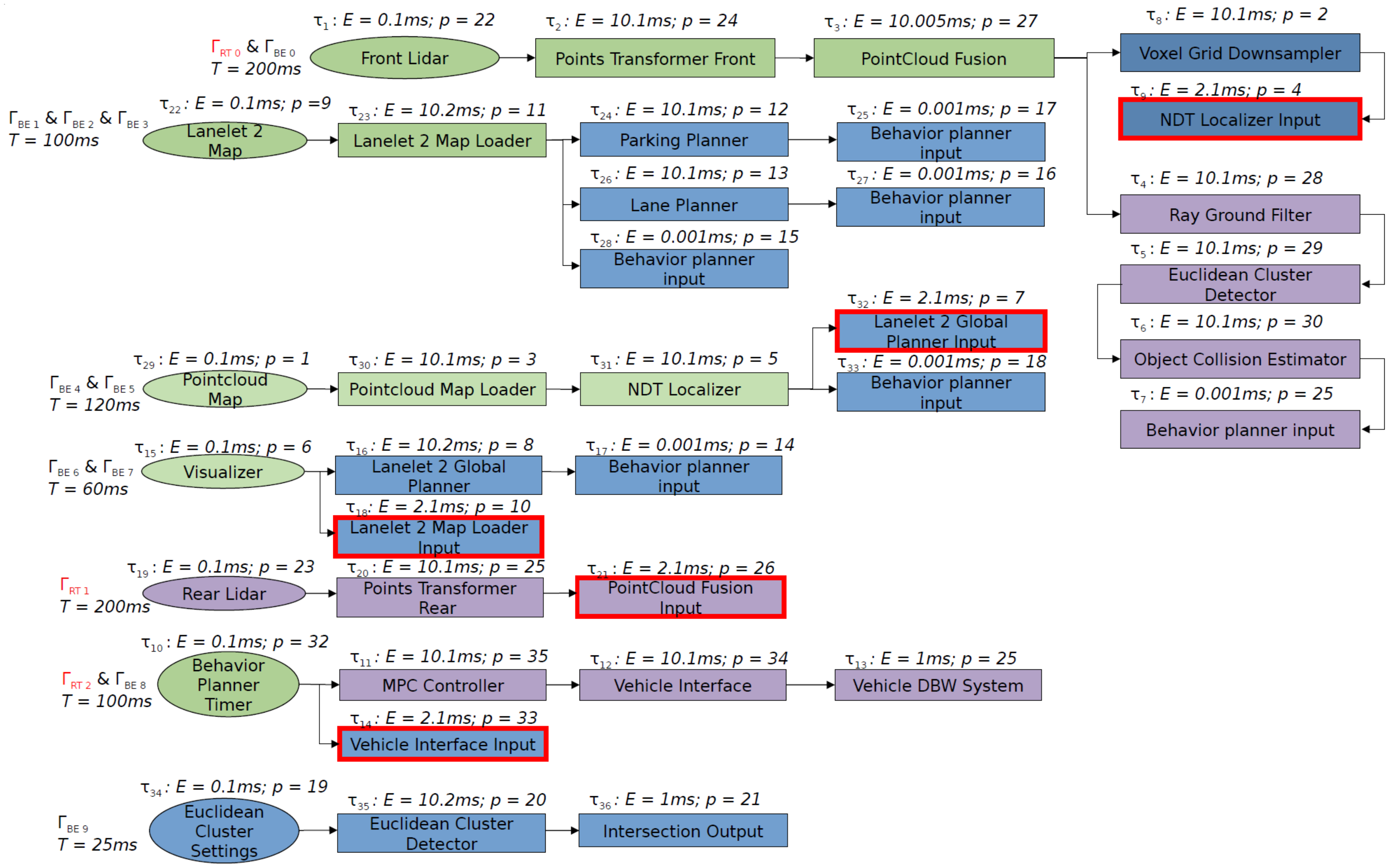
- For BE chains:
  - Check if WCRT R_c is bounded and $R_c \leq$ controller period
  - If so, it executes at least once
    → Starvation freedom

Theorem 3 (from [19]). *The response time of a chain $\Gamma_c = [\tau_{c_1}, \tau_{c_2}, ..., \tau_{c_n}]$ with an* ***arbitrary deadline*** *on a* ***standard*** *ROS 2 executor with $|\Pi|$ threads is upper-bounded by $R_c = \Delta + \overline{sbf}_k(E_{c_n} - 1)$, if $dbf(\Delta) < sbf_\Pi(\Delta)$ holds for the following $dbf(\Delta)$:*

$$dbf(\Delta) = |\Pi| \cdot (E_c - E_{c_n}) + (\sum_{\Gamma_x \in S_{BE}} W_x^*(\Delta, D_x - E_x)) - E_c$$

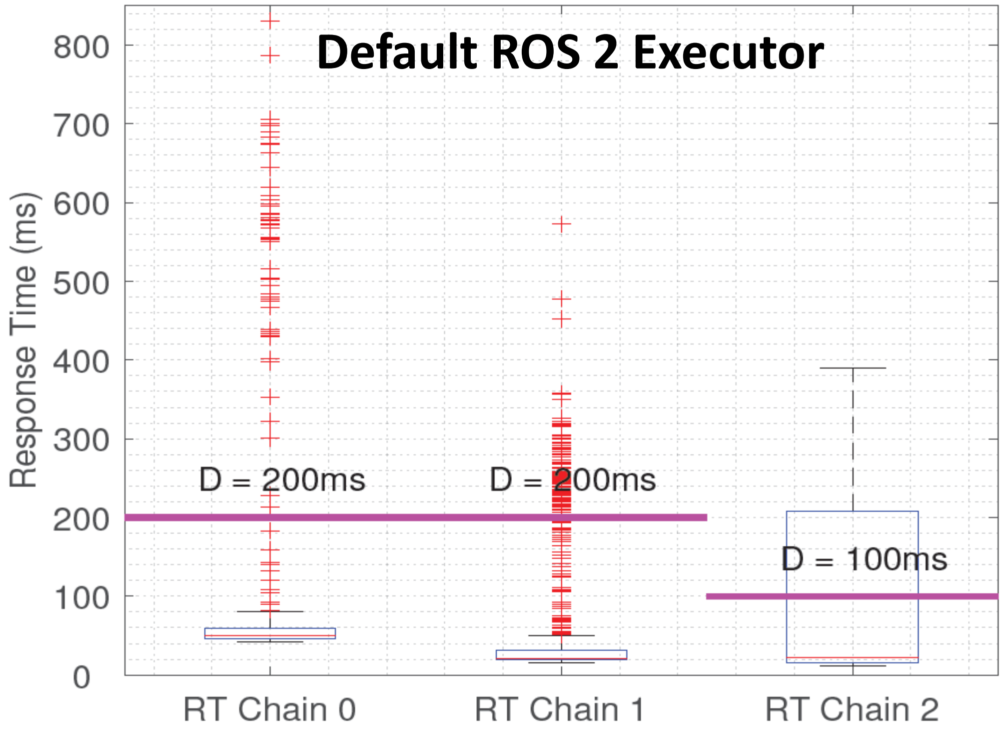
Case Study 1: Autoware Reference System

- Nvidia Jetson AGX Xavier platform
- 15 chains executed by 3 threads
- Arbitrary overloads: BE chains 4 and 5 were duplicated with each callback running for 100ms

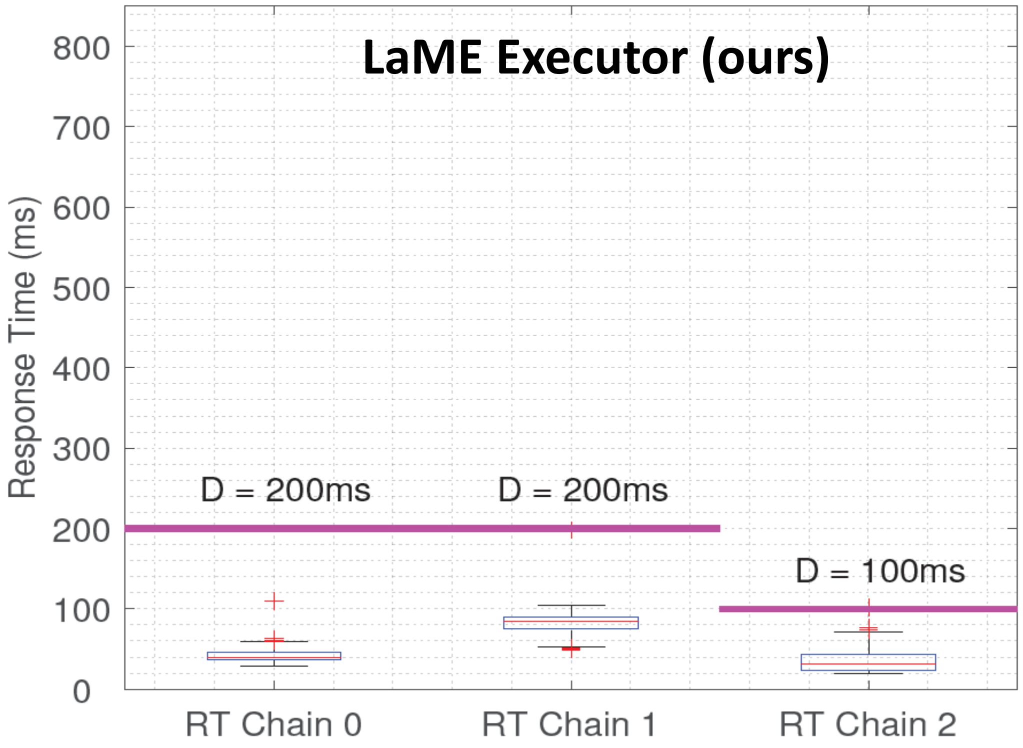


Case Study 1: Autoware Reference System

- RT chains



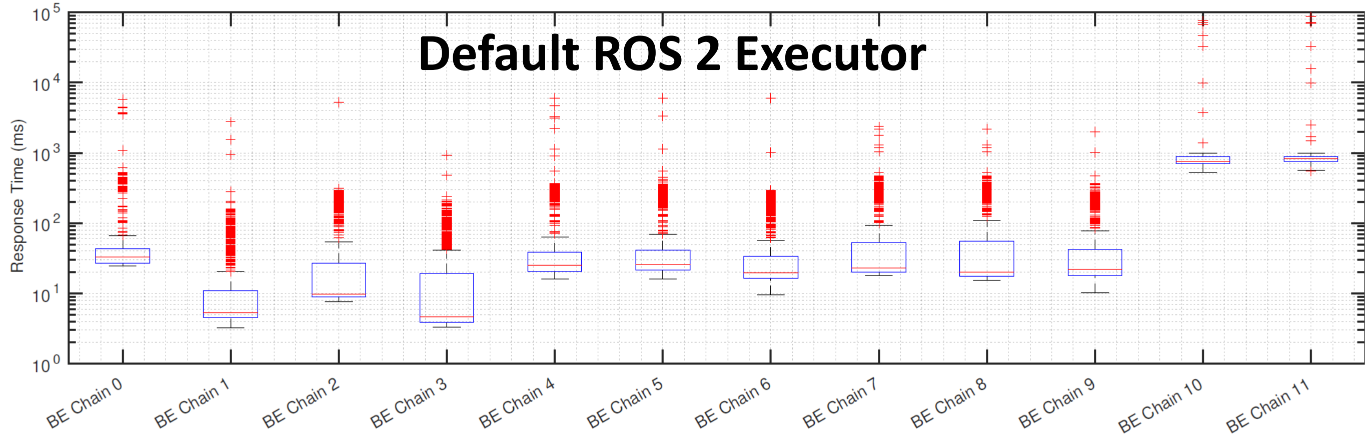
- Deadline misses for all RT chains



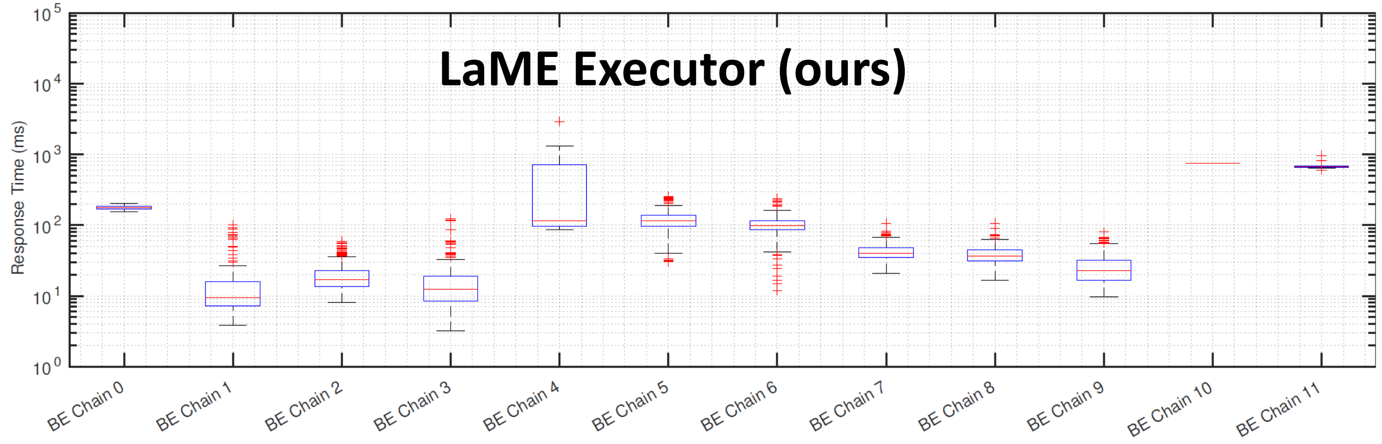
- No deadline misses for RT chains
- Guaranteed WCRT
- Smaller observed response time (up to 4x better)

Case Study 1: Autoware Reference System

- BE chains



- Large fluctuation in BE chain response time

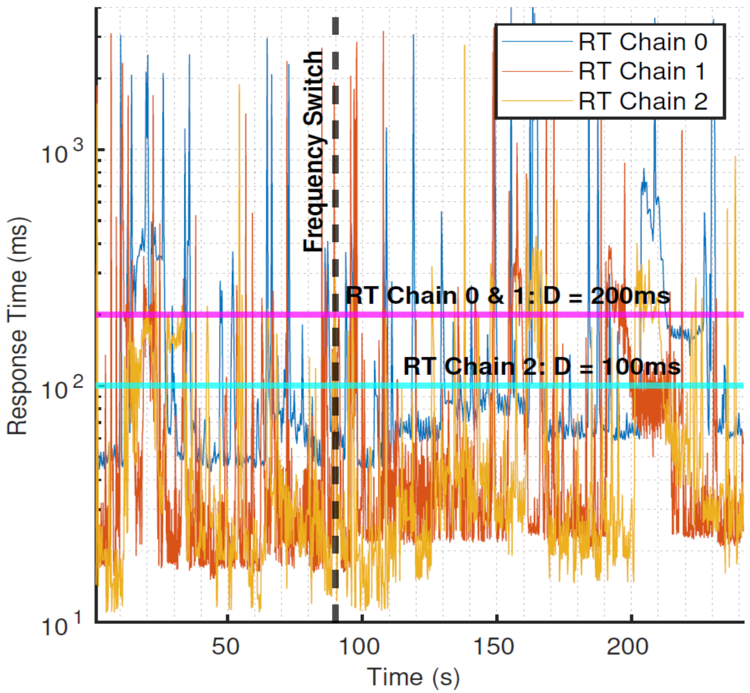


- Much smaller response time for BE chains

Case Study 2: Online Frequency Throttling

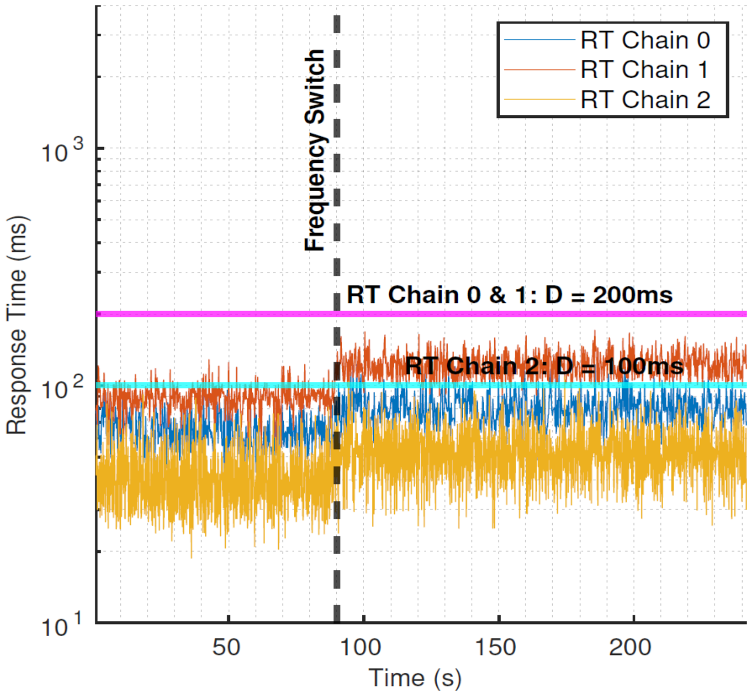
- Autoware reference system with a frequency throttling event at t=90s

Default ROS 2 Executor



- No WCRT guarantees
- More deadline misses after throttling

LaME Executor (ours)

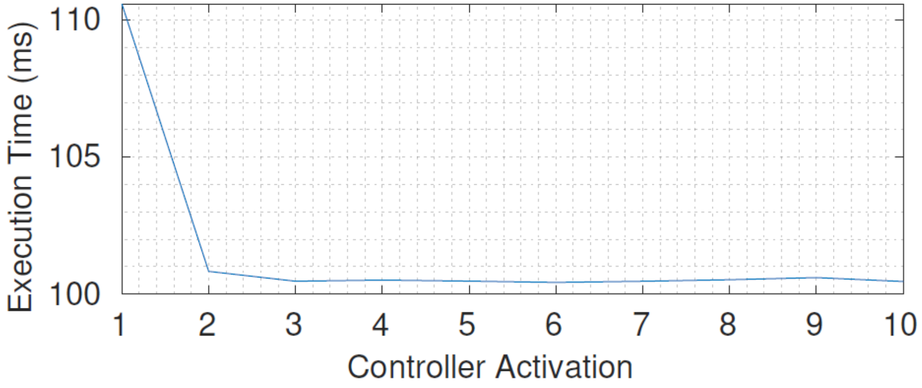


- All chains meet their deadlines even after frequency throttling

UC RIVERSIDE

Controller Runtime Overhead

- Approximately 110ms per activation
  - Mainly due to repeated response-time analysis for budget adjustment
  - However, controller runs as a separate thread with standard priority in Linux – it does not interfere with executor threads
- In stable states, the controller cost becomes negligibly small



(a) Controller Runtime Per Activation

Operation	Time (ms)	Percentage
Thread class creation	0.185	0.17%
Chain-to-thread allocation	0.636	0.58%
Initial budget reduction	109.287	99.25%
Total	110.108	100%

(b) Controller Overhead Breakdown

Conclusions

- LaME: Latency Management Executor Framework
  - Implemented as a redesign of the ROS 2 Multi-threaded Executor
  - Threadclasses for performance isolation and fine-grained resource management
    - Priority-driven scheduling for RT chains
    - Fairness-oriented scheduling for BE chains
  - Adaptive resource controller for dynamic resource and chain assignments
  - Guaranteed worst-case response time for RT chains
    - Up to 4x better maximum observed response time

https://github.com/rtenlab/reference-system-latency-management