Priority-Driven Chain-Aware Scheduling with PiCAS

October 2021

Dr. Hyunjong Choi
Postdoc
University of California, Riverside

R TEN
Real-time Embedded and Networked Systems Laboratory

UC RIVERSIDE

I. Motivation

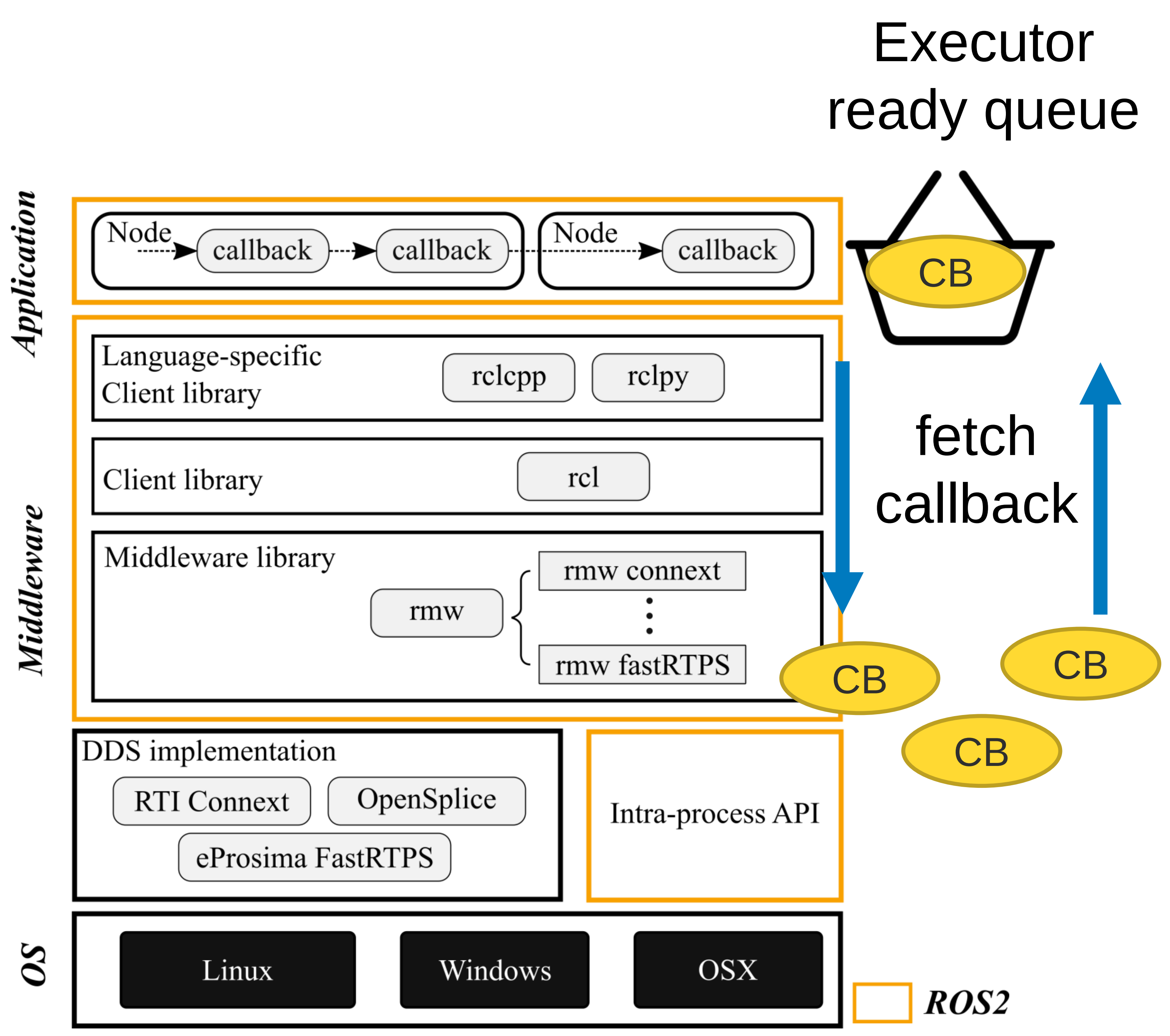
II. PiCAS framework

III. PiCAS on reference system

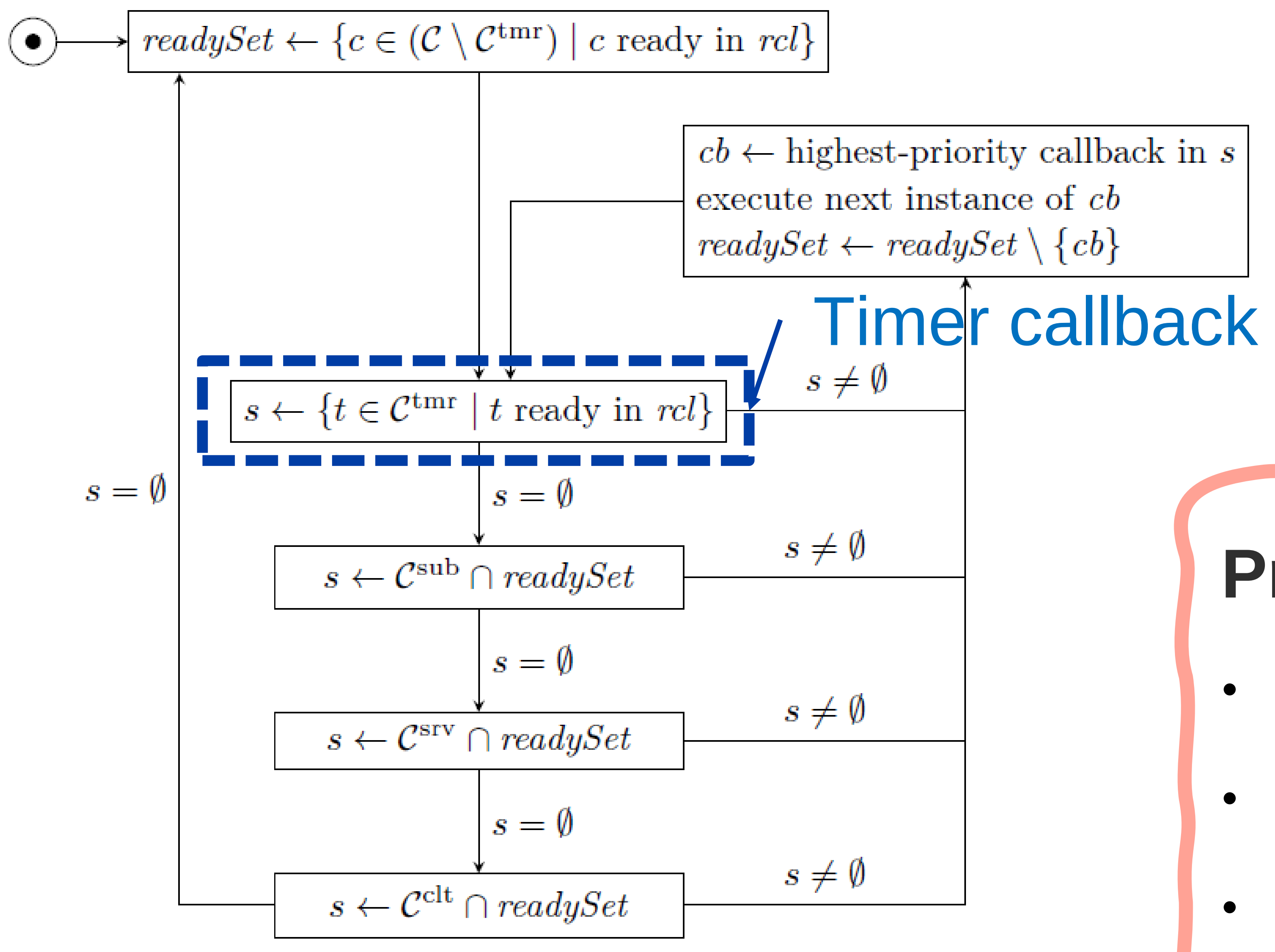
I. Motivation

Motivation

- ROS 2 executor scheduling



< ROS2 architecture >



< Callback scheduling of executor† >

Problems

- Suffers from priority inversion
- No systematic resource allocation methods
- Complex and pessimistic to analyze
- Difficult to prioritize critical chains

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

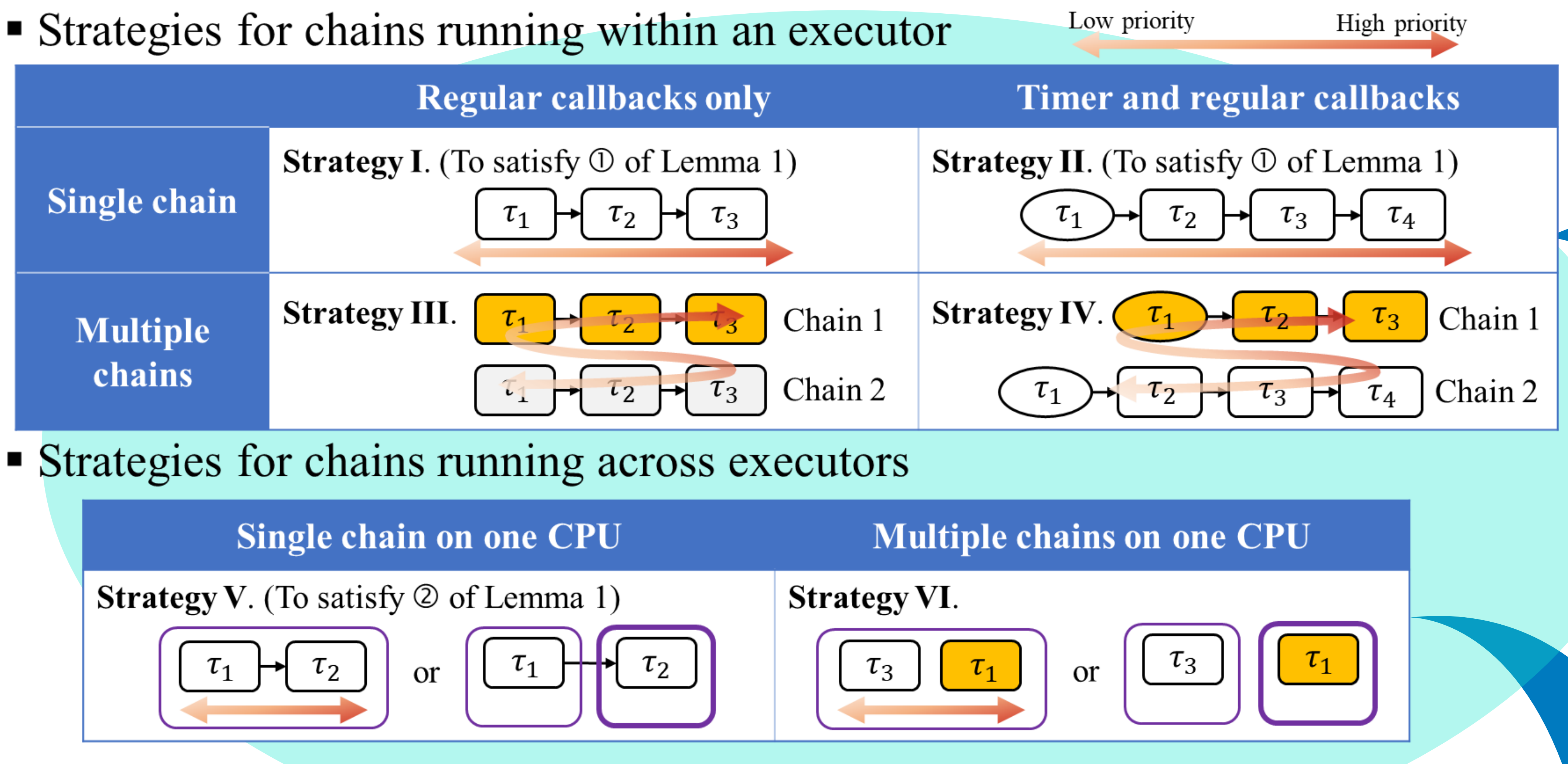
II. PiCAS Framework

PiCAS: Priority-driven Chain-Aware Scheduling framework for ROS2

- **Key idea**: enables ***prioritization of mission-critical chains*** across complex abstraction layers of ROS 2
  - To minimize end-to-end latency
  - To ensure predictability even when the system is overloaded

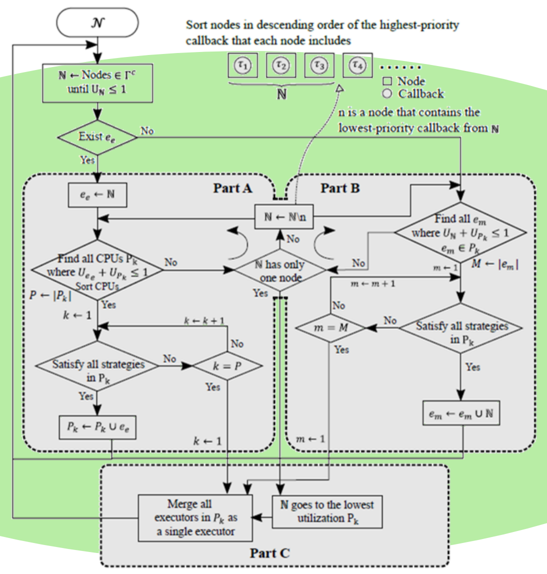
- **PiCAS**: Executor + Resource Allocation Algorithms + Timing Analysis
  - **PiCAS executor:** priority-driven callback scheduling
  - **Resource allocation algorithms**
    - Callback Priority Assignment
    - Chain-Aware Node-to-Executor Allocation
    - Executor Priority Assignment
  - Backed by **formal end-to-end latency analysis**

PiCAS Algorithms



< Chain-aware scheduling strategies >

< Node-to-Executor allocation >

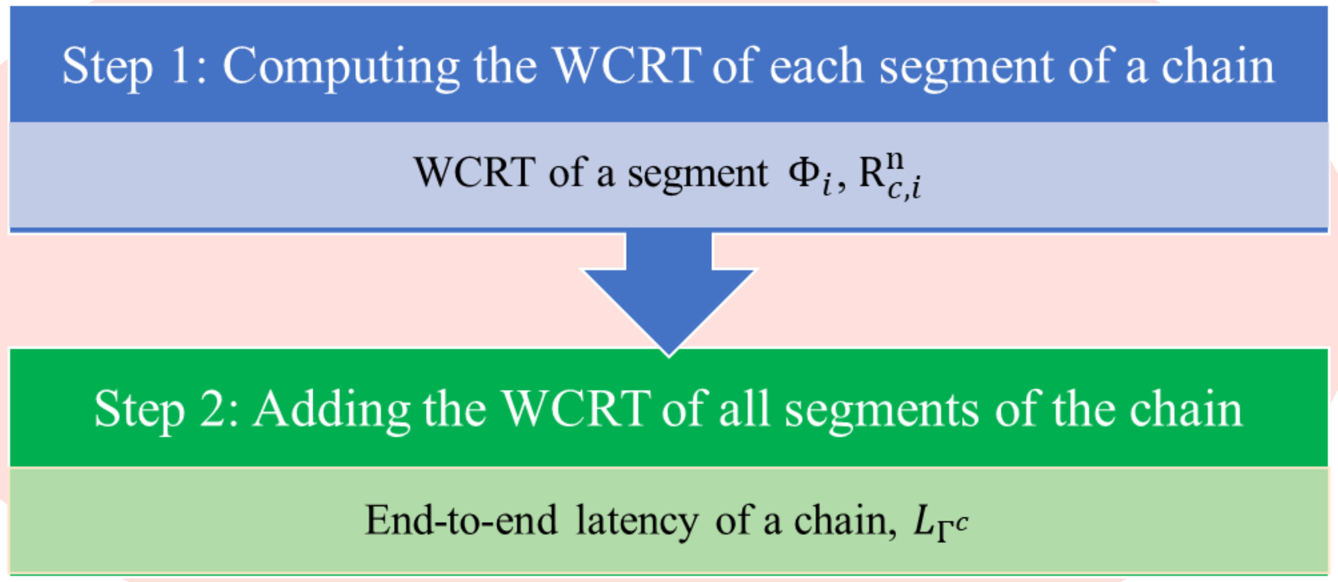


Algorithm 1 Callback priority assignment

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**

< Priority assignment >



< End-to-end timing analysis >

For details, please see our paper:

Hyunjong Choi, Yecheng Xiang, and Hyoseung Kim, **PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2.** In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021. [Paper | Slides | Video]

PiCAS Executor (1/2)

- Implemented as an extension to the rclcpp wait-set executor
- PiCAS executor API

```
// Set RT priority and CPU affinity of executor instance
void Executor::set_executor_priority_cpu(int priority, int cpu);

// Enable/Disable priority-based callback scheduling
void Executor::enable_callback_priority();
void Executor::disable_callback_priority();

// Set callback priority
void Executor::set_callback_priority(rclcpp::TimerBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::SubscriptionBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::ServiceBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::ClientBase::SharedPtr ptr, int priority);
void Executor::set_callback_priority(rclcpp::WaitableBase::SharedPtr ptr, int priority);

// Spin for PiCAS (RT executor priority & CPU affinity)
void SingleThreadedExecutor::spin_rt();
```

Default parameters

```
class Executor
{ ...
#ifdef PICAS
  bool callback_priority_enabled = false;
  int executor_priority = 0;
  int executor_cpu = 0;
```

executor.hpp

```
...
#ifdef PICAS
  int callback_priority = 0;
#endif
...
```

client.hpp, service.hpp, timer.hpp, subscription_base.hpp, waitable.hpp

PiCAS Executor (2/2)

- Implementation details

```
Bool Executor::get_next_executable
{

bool success = false;
if (!success) {
     wait_for_work(timeout);
}

success = get_next_ready_executable(any_executable);
...
}
```

Update wait-set whenever each callback completes

get_next_executable of executor.cpp (PiCAS)

Select the highest priority callback among all ready callbacks

```
Bool Executor::get_next_ready_executable
{
...
memory_strategy_->get_next_waitable(any_exe,
weak_nodes);
    if (any_exe.cb && highest_priority <
any_exe.waitable->callback_priority) {
      highest_priority = any_executable.waitable-
>callback_priority;
      any_executable.timer = nullptr;
      any_executable.subscription = nullptr;
      any_executable.service = nullptr;
      any_executable.client = nullptr;
    }
    else any_executable.waitable = nullptr;
...
}
```

get_next_ready_executable of executor.cpp (PiCAS)

➡ Callbacks can be scheduled based on their priorities

- Pro: waiting time for high-priority callback can be minimized
- Con: overhead; not good for high throughput of short, same-priority callbacks

III.PiCAS on reference system

PiCAS on reference system

- Clone our forked repository

```
git clone https://github.com/rtenlab/reference-system.git
```

- Build with PiCAS executor
  - Use PICAS CMake variable

```
colcon build --cmake-args -DRUN_BENCHMARK=TRUE -DTEST_PLATFORM=TRUE -DPICAS=TRUE
```

- Configuration change for Linux RT priority
  - Modify /etc/security/limits.conf

```
<userid>  hard rtprio    99
<userid>  soft  rtprio    99
```

- Notes
  - PiCAS is implemented as an extension to *rclcpp,* located in *reference-system/rclcpp.* This local *rclcpp* overrides the default ROS2 rclcpp.
  - If -DPICAS=FALSE, *reference-system/rclcpp* is exactly the same as the ROS2 Galactic version.

How to use PiCAS executor

Declaration of executor

Enable callback priority (PiCAS executor)

callback_priority_enabled = true

Set executor's real-time priority and assign CPU

Executor (process) scheduled under SCHED_FIFO policy that has a sched_priority value in the range 1 (low) to 99 (high).

Allocate nodes to executor

`std::thread spinThread(&rclcpp::executors::SingleThreadedExecutor::spin_rt, &executor);`

Run executor (*spin_rt()*)

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  using TimeConfig = nodes::timing::Default;
  // uncomment for benchmarking
  //using TimeConfig = nodes::timing::BenchmarkCPUUsage;
  // set_benchmark_mode(true);

  auto nodes = create_autoware_nodes<RclcppSystem, TimeConfig>();

  rclcpp::executors::SingleThreadedExecutor executor;
  executor.enable_callback_priority();
  RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "PiCAS priority-based cal

  executor.set_executor_priority_cpu(90, 0);
  RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "PiCAS executor 1's rt-pr

  for (auto & node : nodes) {
    executor.add_node(node);
  }
  executor.spin_rt();

  nodes.clear();
  rclcpp::shutdown();

  return 0;
}
```

autoware_default_singlethreaded_picas_single_executor.cpp

How to assign callback priority

- Callback priority assignment on reference system

Set unique priority to callbacks

```cpp
namespace callback
{
namespace priority
{
  struct Default
  {
    // The higher number, more critical callback
    static constexpr int FRONT_LIDAR_DRIVER_CALLBACK = 51;
    static constexpr int REAR_LIDAR_DRIVER_CALLBACK = 50;
    static constexpr int POINT_CLOUD_MAP_CALLBACK = 22;
    static constexpr int LANELET_2_MAP_CALLBACK = 30;
    static constexpr int VISUALIZER_CALLBACK = 27;
    static constexpr int POINTS_TRANSFORMER_REAR_CALLBACK = 52;
    static constexpr int POINTS_TRANSFORMER_FRONT_CALLBACK = 53;
    static constexpr int POINT_CLOUD_FUSION_CALLBACK_1 = 55;
    static constexpr int POINT_CLOUD_FUSION_CALLBACK_2 = 54;
    static constexpr int POINT_CLOUD_MAP_LOADER_CALLBACK = 24;
    static constexpr int VOXEL_GRID_DOWNSAMPLER_CALLBACK = 23;
    static constexpr int RAY_GROUND_FILTER_CALLBACK = 56;
    static constexpr int NDT_LOCALIZER_CALLBACK_1 = 26;
    static constexpr int NDT_LOCALIZER_CALLBACK_2 = 25;
    static constexpr int EUCLIDEAN_CLUSTER_SETTINGS_CALLBACK = 47;
    static constexpr int INTERSECTION_OUTPUT_CALLBACK = 49;
    static constexpr int EUCLIDEAN_CLUSTER_DETECTOR_CALLBACK = 57;
```

autoware_reference_system/system/priority/default.hpp

```cpp
// setup communication graph
// sensor nodes
nodes.emplace_back(
  std::make_shared<typename SystemType::Sensor>(
    nodes::SensorSettings{.node_name = "FrontLidarDriver",
      .topic_name = "FrontLidarDriver",
      .cycle_time = TimingConfig::FRONT_LIDAR_DRIVER,
#ifdef PICAS
      .callback_priority = CallbackPriority::FRONT_LIDAR_DRIVER_CALLBACK
#endif
      }));
```

autoware_reference_system/include/autoware_reference_system/autoware_system_builder.hpp

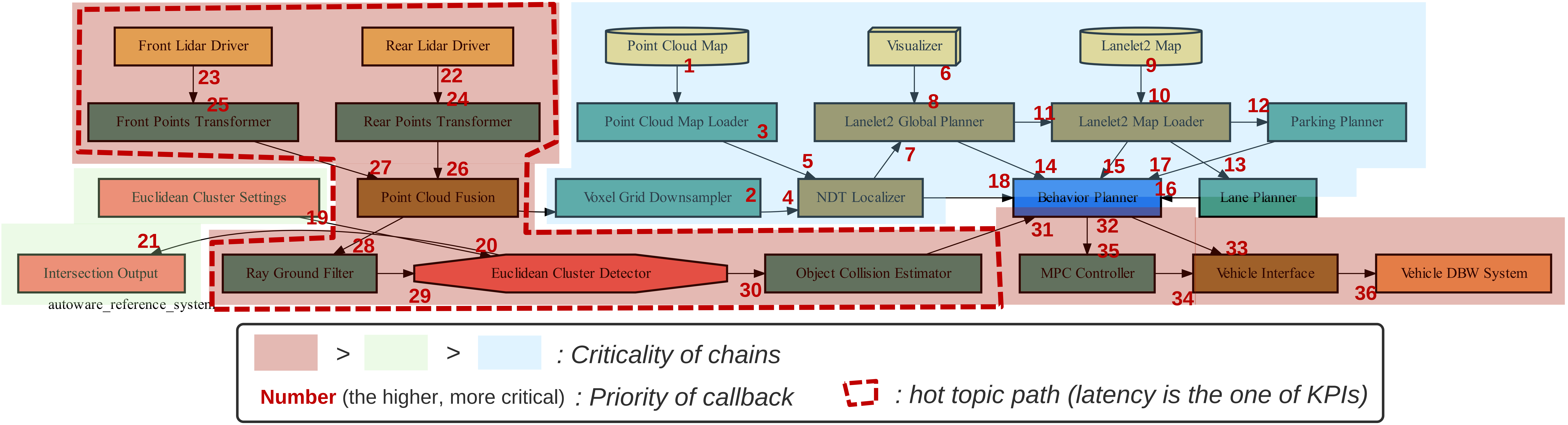
```cpp
class Sensor : public rclcpp::Node
{
public:
  explicit Sensor(const SensorSettings & settings)
  : Node(settings.node_name)
  {
    publisher_ = this->create_publisher<message_t>(settings.topic_name, 1);
    timer_ = this->create_wall_timer(
      settings.cycle_time,
      [this] {timer_callback();});
#ifdef PICAS
    timer_->callback_priority = settings.callback_priority;
#endif
  }
```

reference_system/include/reference_system/nodes/rclcpp/nodes.hpp

- Or, use API, e.g., `executor.set_callback_priority(node->callback, priority)`

Evaluation

- Autoware model



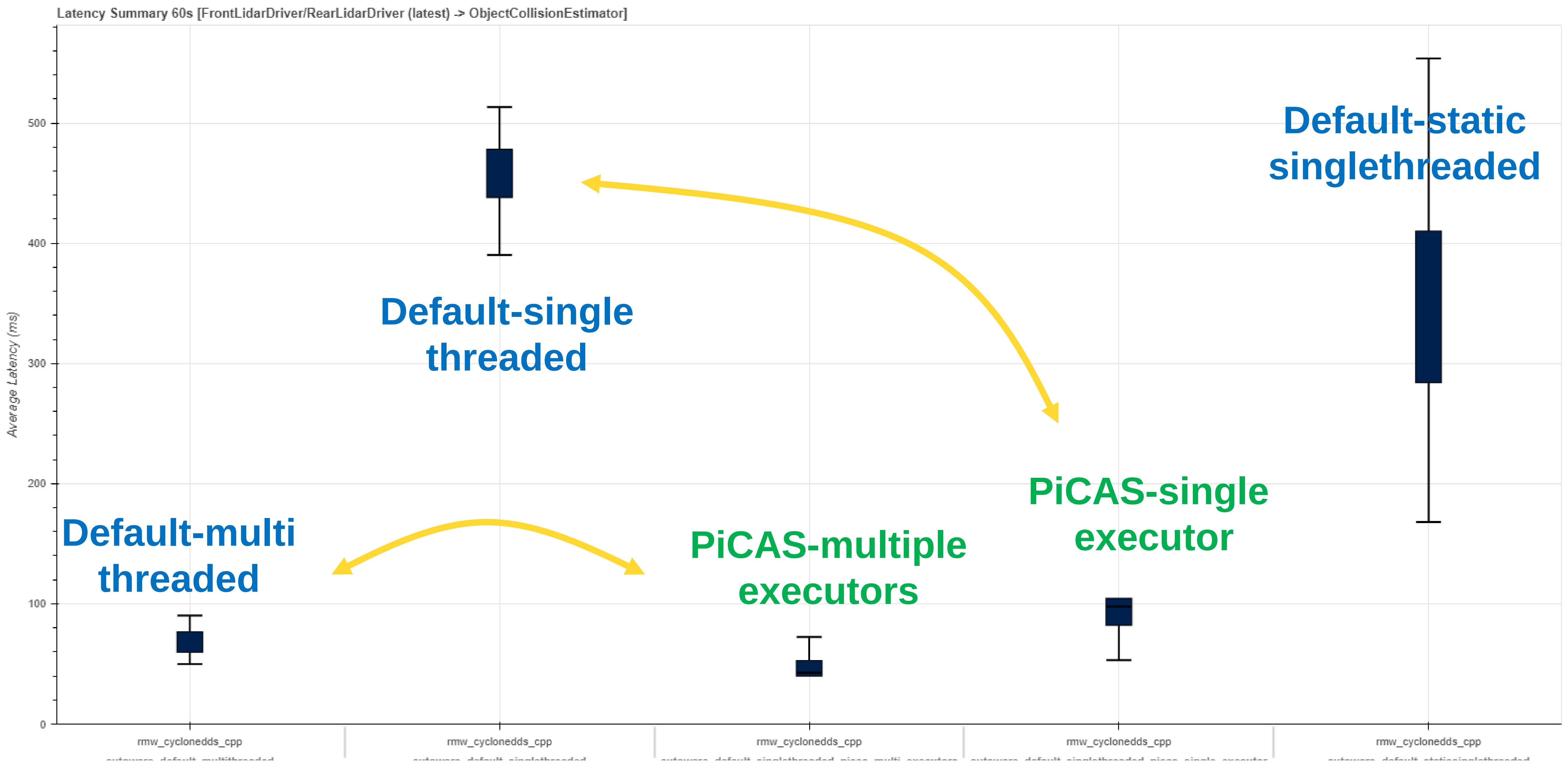
- Single executor instance & multiple executor instances
  - Based on the PiCAS priority assignment and node-to-executor allocation algorithms
  - Algorithm implementation: https://github.com/rtenlab/ros2-picas

Evaluation

- Experiment environment
  - Raspberry Pi 4 with fixed CPU frequency of 1.5GHz
  - 4 CPU cores for multiple executors (PiCAS) and multithreaded executor (ROS2 default)
  - Run `colcon test` with *RUN_TIMES* option of 60 seconds
  - Evaluation criteria : Key Performance Indicators (KPIs) of reference system

Evaluation

- Latency summary

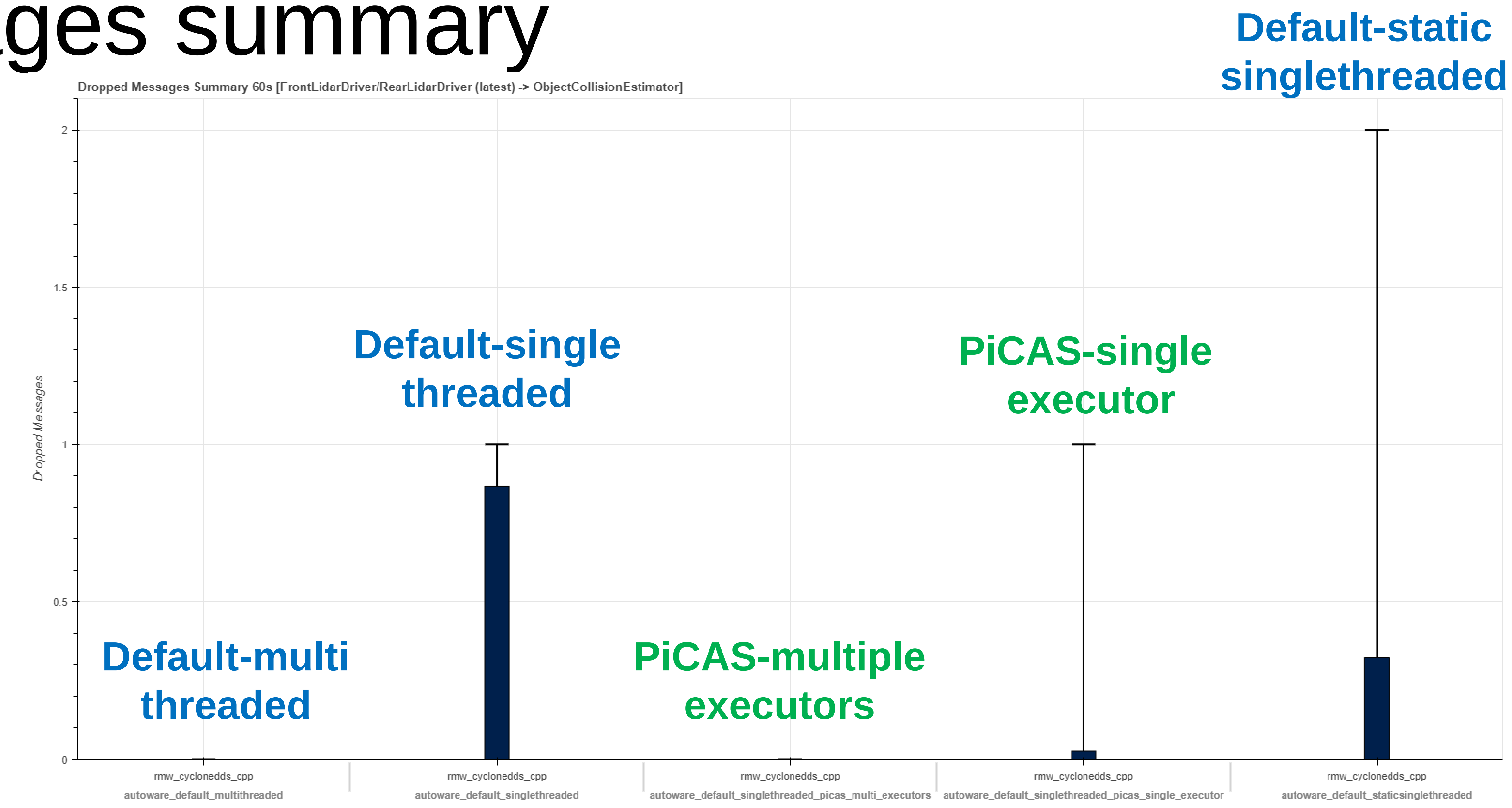


Latency Summary Table 60s [FrontLidarDriver/RearLidarDriver (latest) -> ObjectCollisionEstimator]

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_multithreaded	rmw_cyclonedds_cpp	latency	49.8478	68.1878	90.1849	76.76550999	59.61009	8.57771
1	autoware_default_singlethreaded	rmw_cyclonedds_cpp	latency	390.14	458.074	513.353	478.1154	438.0326	20.0414
2	autoware_default_singlethreaded_picas_multi_executors	rmw_cyclonedds_cpp	latency	42.6901	46.3615	72.4256	52.83746	39.88554	6.47596
3	autoware_default_singlethreaded_picas_single_executor	rmw_cyclonedds_cpp	latency	53.0758	93.319	97.5617	104.5243	82.1137000	11.2053
4	autoware_default_staticsinglethreaded	rmw_cyclonedds_cpp	latency	168.066	347.027	553.69	410.0831	283.9709	63.0561

Evaluation

- Dropped messages summary

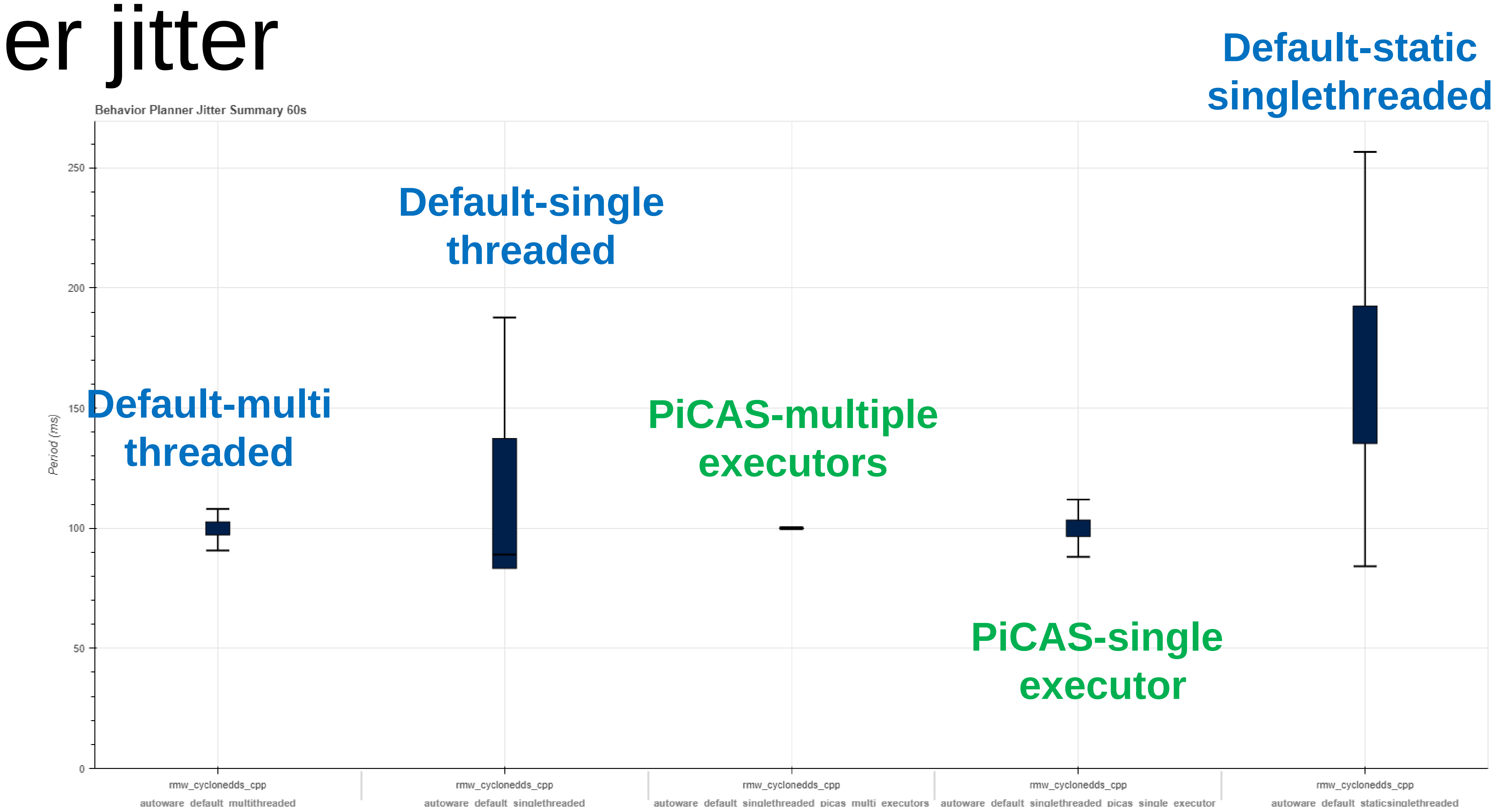


Dropped Messages Summary Table 60s [FrontLidarDriver/RearLidarDriver (latest) -> ObjectCollisionEstimator]

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_multithreaded	rmw_cyclonedds_cpp	dropped	0	0	0	0	0	0
1	autoware_default_singlethreaded	rmw_cyclonedds_cpp	dropped	0	0.868132	1	1.202931	0.53333	0.334799
2	autoware_default_singlethreaded_picas_multi_executors	rmw_cyclonedds_cpp	dropped	0	0	0	0	0	0
3	autoware_default_singlethreaded_picas_single_executor	rmw_cyclonedds_cpp	dropped	0	0.0282776	1	0.1934025999	0	0.165125
4	autoware_default_staticsinglethreaded	rmw_cyclonedds_cpp	dropped	0	0.325088	2	0.826076	0	0.500988

Evaluation

- Behavior planner jitter

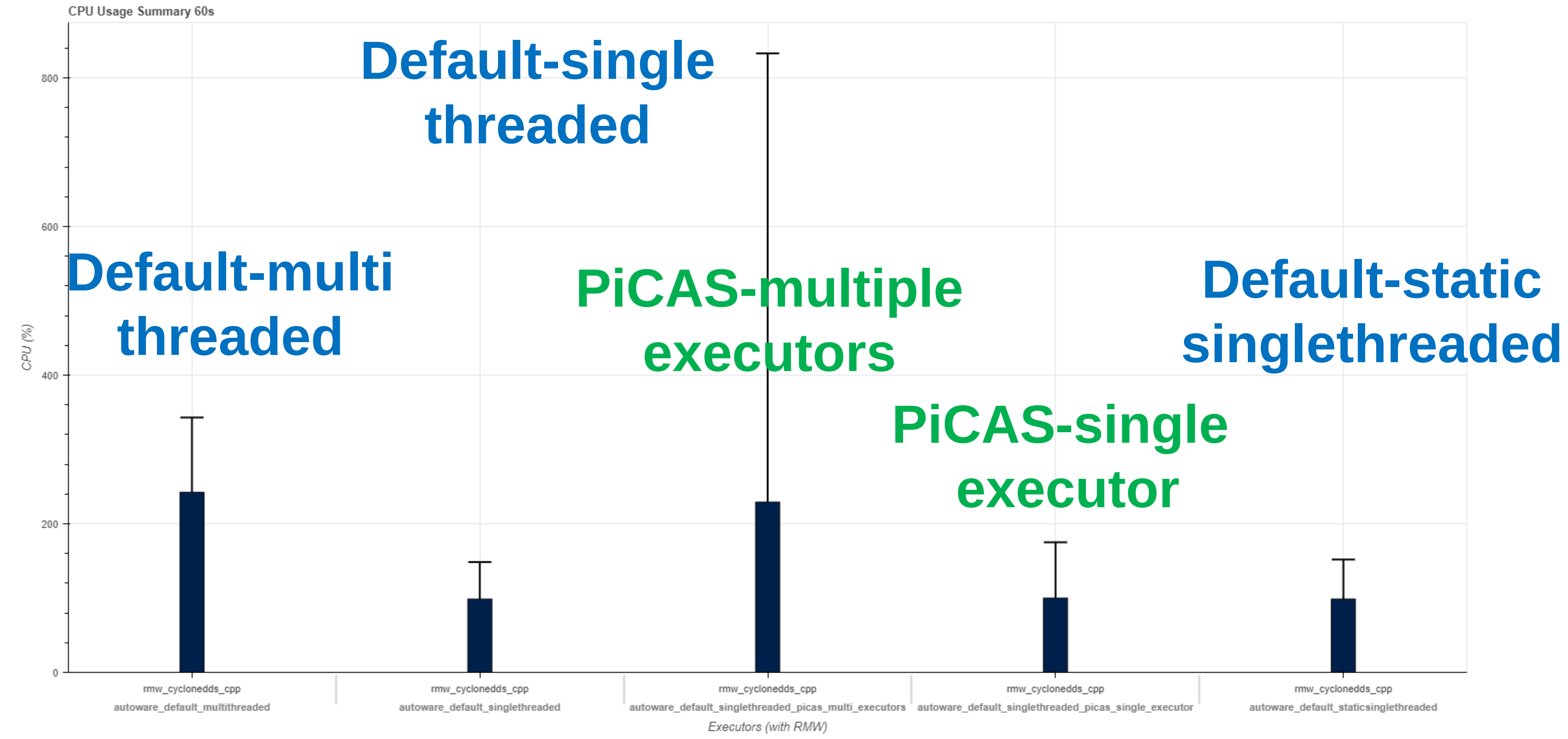


Behavior Planner Jitter Summary Table 60s

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_multithreaded	rmw_cyclonedds_cpp	period	90.7153	99.9909	108.076	102.68037	97.30143	2.68947
1	autoware_default_singlethreaded	rmw_cyclonedds_cpp	period	89.0889	110.359	187.791	137.4093	83.308699999	27.0503
2	autoware_default_singlethreaded_picas_multi_executors	rmw_cyclonedds_cpp	period	99.6448	99.9992	100.403	100.0936087	99.9047913	0.0944087
3	autoware_default_singlethreaded_picas_single_executor	rmw_cyclonedds_cpp	period	88.1241	100.002	111.957	103.45313	96.550869999	3.45113
4	autoware_default_staticsinglethreaded	rmw_cyclonedds_cpp	period	84.1847	163.916	256.623	192.5484	135.2836	28.6324

Evaluation

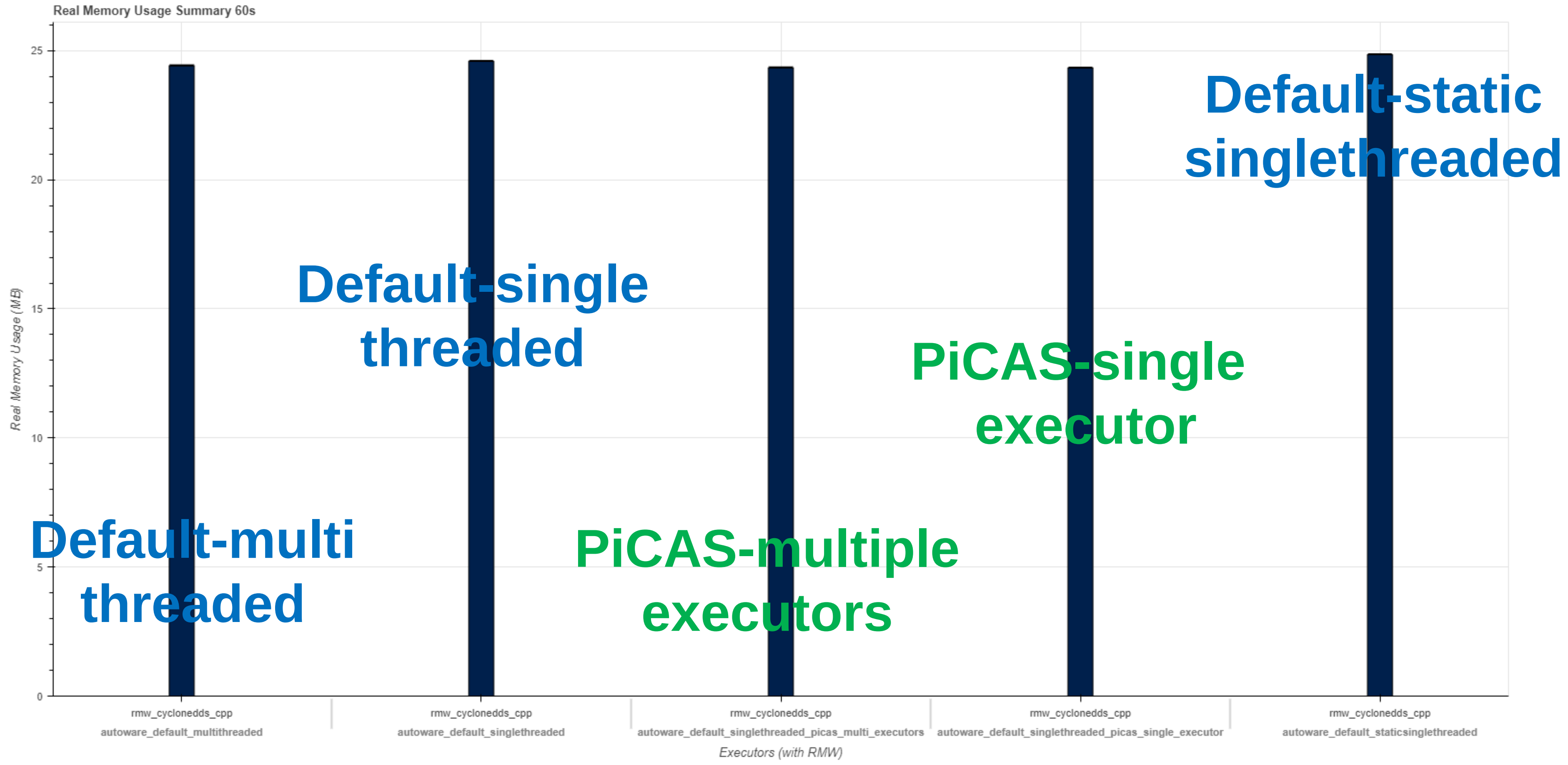
- CPU usage summary



CPU Usage Statistics 60s

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_multithreaded	rmw_cyclonedds_cpp	cpu	0	242.3965	343	274.5	223	35.4647578
1	autoware_default_singlethreaded	rmw_cyclonedds_cpp	cpu	0	98.67198	148.5	100.9	98	11.0350447
2	autoware_default_singlethreaded_picas_multi_executors	rmw_cyclonedds_cpp	cpu	0	229.1575	833	259.5750	198.925	45.2638521
3	autoware_default_singlethreaded_picas_single_executor	rmw_cyclonedds_cpp	cpu	0	99.98559	175.1	102.4	99.6	10.9679357
4	autoware_default_staticsinglethreaded	rmw_cyclonedds_cpp	cpu	0	98.70412	151.9	100.9	98.5	10.2886736

- Memory usage summary



Real Memory Usage Statistics 60s

#	exe	rmw	type	low	mean	high	top	bottom	std_dev
0	autoware_default_multithreaded	rmw_cyclonedds_cpp	real	19.309	24.420044	24.434	24.434	24.434	0.21661509
1	autoware_default_singlethreaded	rmw_cyclonedds_cpp	real	19.551	24.590507	24.605	24.605	24.605	0.2114211
2	autoware_default_singlethreaded_picas_multi_executors	rmw_cyclonedds_cpp	real	19.105	24.337465	24.355	24.355	24.355	0.24359012
3	autoware_default_singlethreaded_picas_single_executor	rmw_cyclonedds_cpp	real	19.227	24.331834	24.344	24.344	24.344	0.20236399
4	autoware_default_staticsinglethreaded	rmw_cyclonedds_cpp	real	19.188	24.846127	24.863	24.863	24.863	0.23209066

Thank you

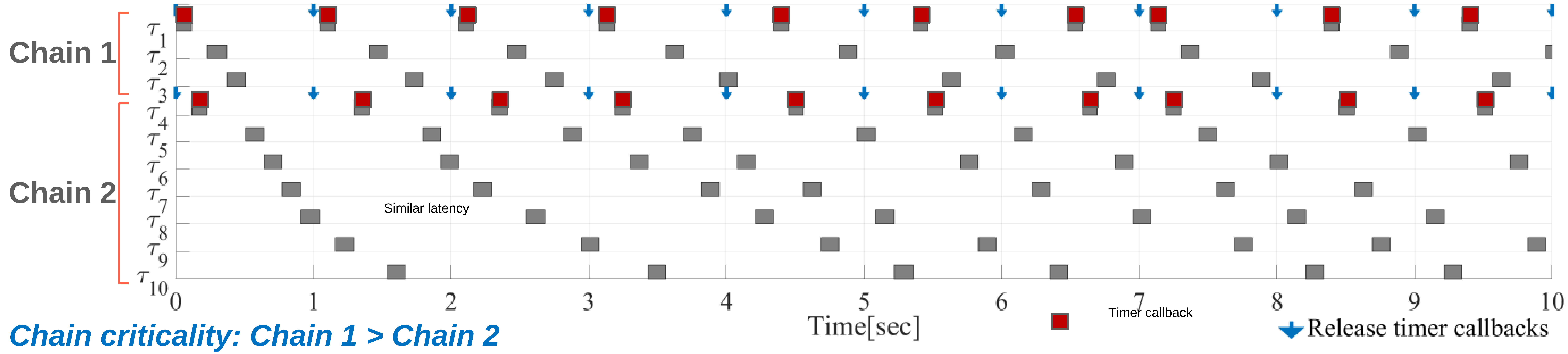
Q & A

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

Challenges (1/2)

❑ Challenge I: Fairness-oriented callback scheduling within executors

(Chains 1 & 2 in the same executor)



Chain criticality: 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 >

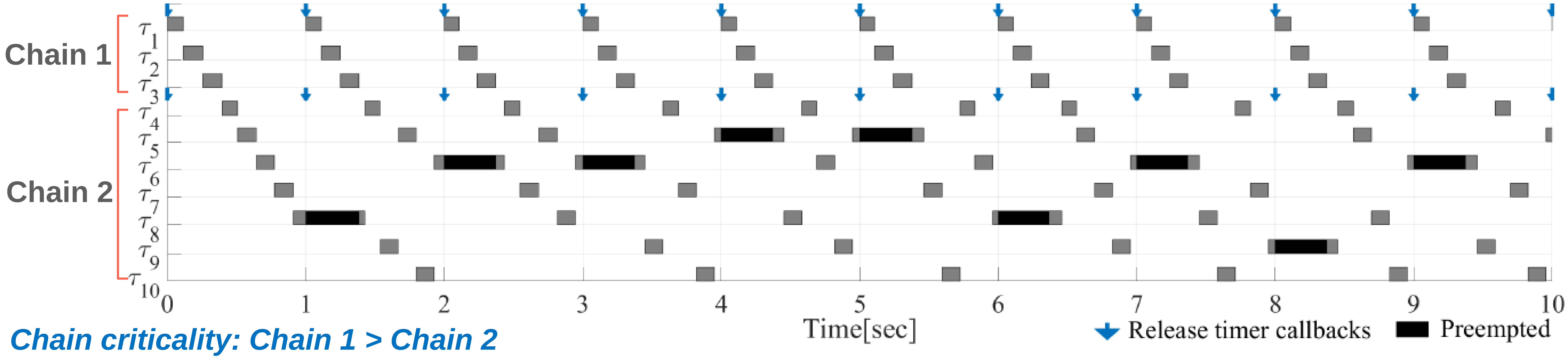
O1. Timer callbacks always get the highest priority
O2. No way to respect chain criticality

➡ **Fairness-oriented scheduling can jeopardize the timeliness of mission-critical chains**

Challenges (2/2)

❑ Challenge II: Priority assignment of executors

(Chain 1 in a higher-priority executor)



Chain criticality: 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 Linux scheduler or naïve priority assignment can cause unacceptably high latency**