Analyzing the reference system

Christophe Bédard

ROS 2 Executor Workshop
ROS World 2021
October 19, 2021

DORSAL Laboratory
Polytechnique Montréal 🇨🇦

POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY

Plan

1. Tracing
2. `ros2_tracing`
3. Tracing the reference system
4. Analyzing the reference system

Tracing

- Goal: gather runtime execution information
  - Low-level information
- Useful when issues are hard to reproduce
- Workflow (static instrumentation)
  - Instrument an application with trace points
  - Configure tracer, run the application
  - Trace points generate events (information)
  - Events make up a trace
  - Analyze the trace
- Other instrumentation
  - Linux kernel instrumentation (e.g., sched_switch, net_dev_queue)
  - Instrumentation in `LD_PRELOAD`ed libraries

ros2_tracing

- gitlab.com/ros-tracing/ros2_tracing
- Collection of tools closely integrated into ROS 2
  - Since ROS 2 Eloquent (2019)
  - Many improvements and additions since then
- Tools to instrument the core of ROS 2 with LTTng
  - `rclcpp`, `rcl`, `rmw` (`rmw_cyclonedds*`)
- Tools to configure tracing with LTTng
  - Command: `ros2 trace`
  - Action for ROS 2 launch: `Trace`
- Instrumentation
  - Object instances: node, publisher, subscription, timer
  - Events
    - Callback execution (subscription, timer)
    - Message publication
  - etc.

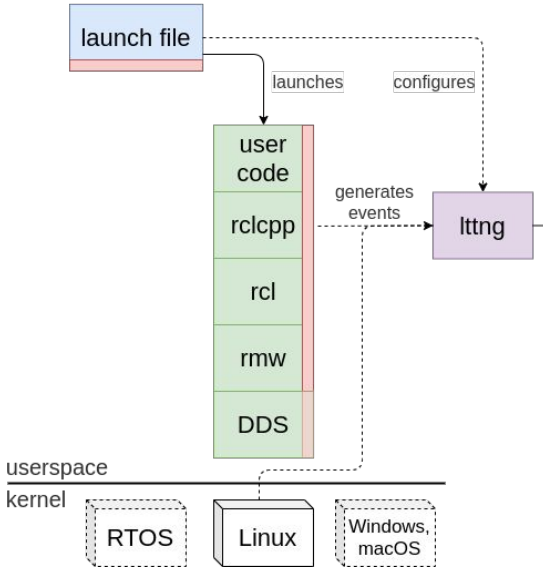
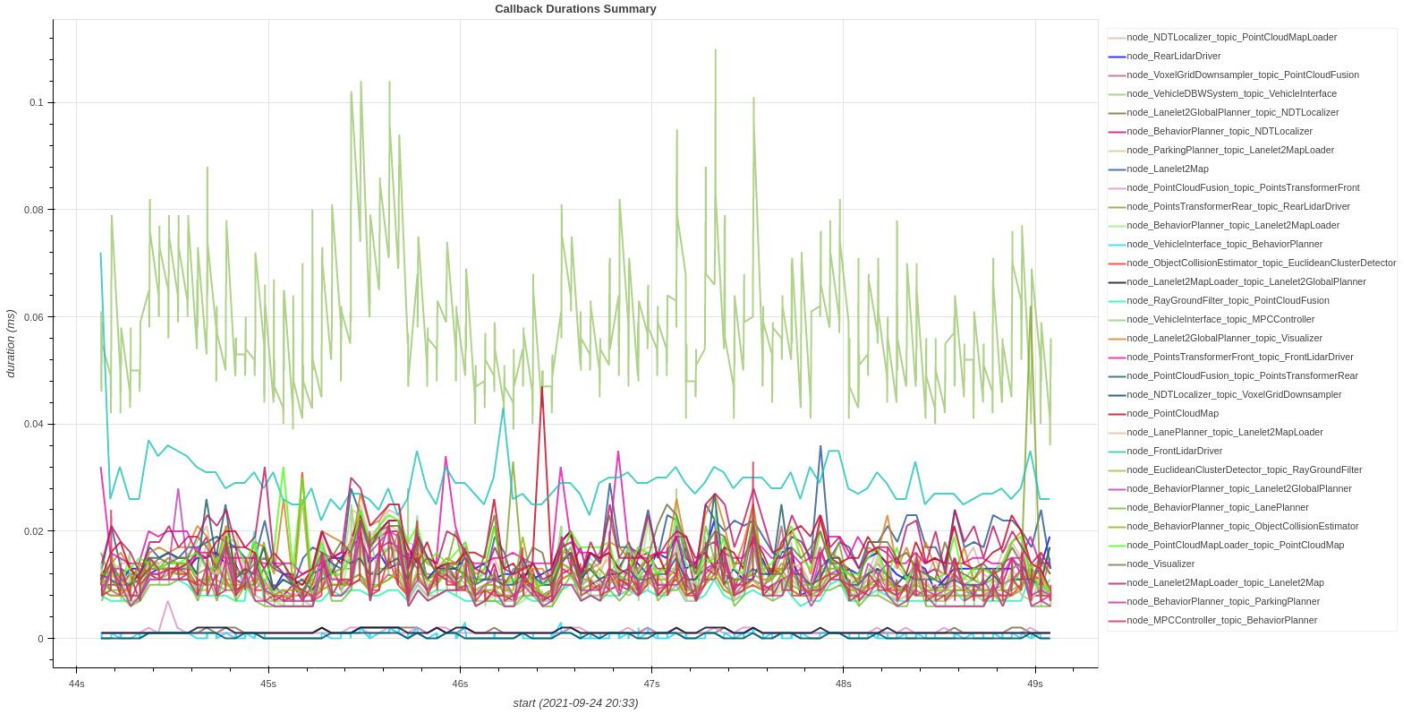


Figure 1. Instrumentation and general workflow.

Tracing the reference system - example



Tracing the reference system

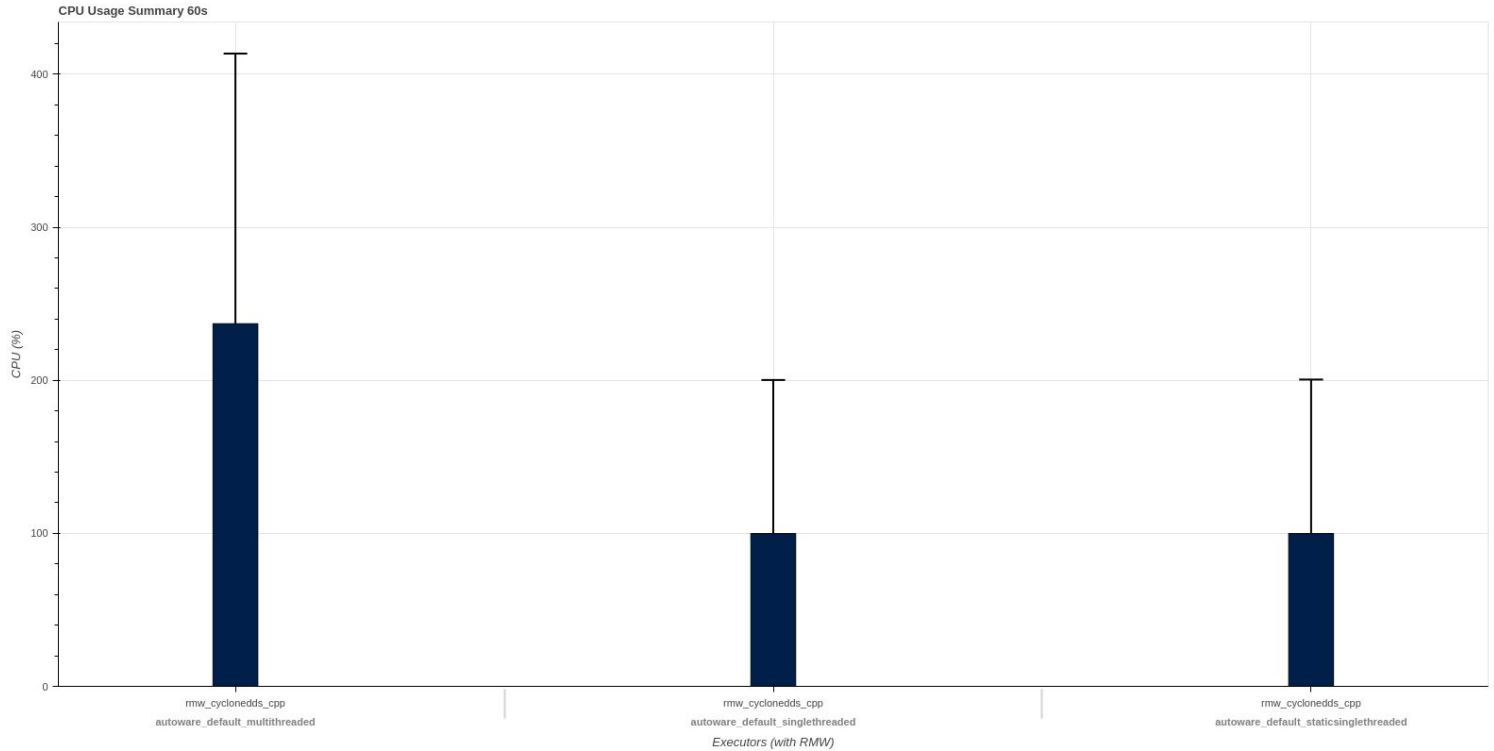
- We can extract a lot of information and compute many different metrics
  - Some of these are quite trivial
  - Others require additional instrumentation or more analysis work
- Timer & subscription callback durations
- Frequency & jitter of publications and callbacks
- Internal executor behaviour (executor-specific)
- Time between message arrival and callback execution
- Time between timer/subscription/service readiness and execution
- Message queue sizes over time
- Latency of a specific path in a processing pipeline
- … and more!

Analyzing the reference system

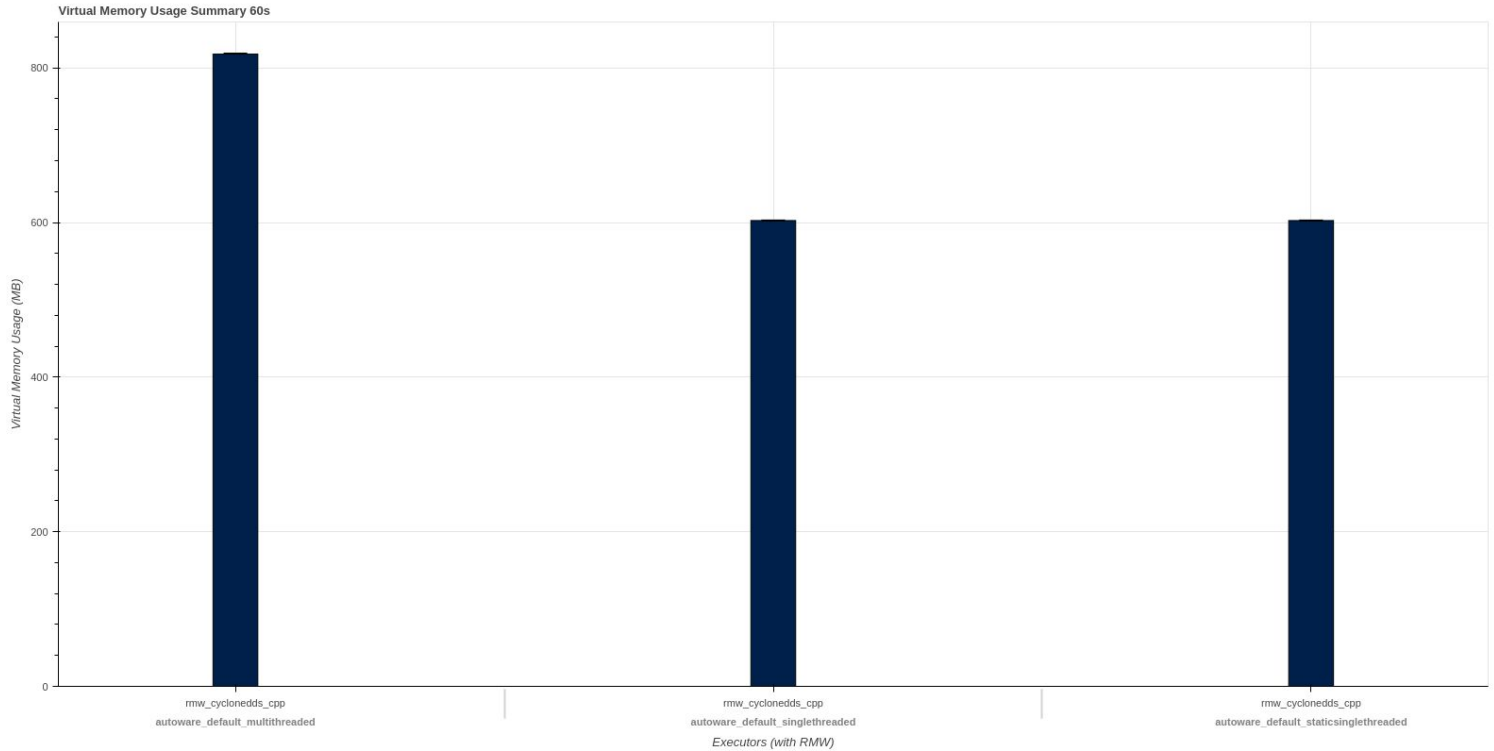
KPIs:

1. CPU usage
   - Using `psrecord`
2. Memory usage
   - Using `psrecord`
3. Latency from Front LiDAR to Object Collision Estimator
   - Using timestamps collected in nodes along the path
4. Dropped samples from Front LiDAR to Object Collision Estimator
   - Using timestamps collected in nodes along the path
5. Jitter of cyclic Behavior Planner callback
   - Using callback start timestamps

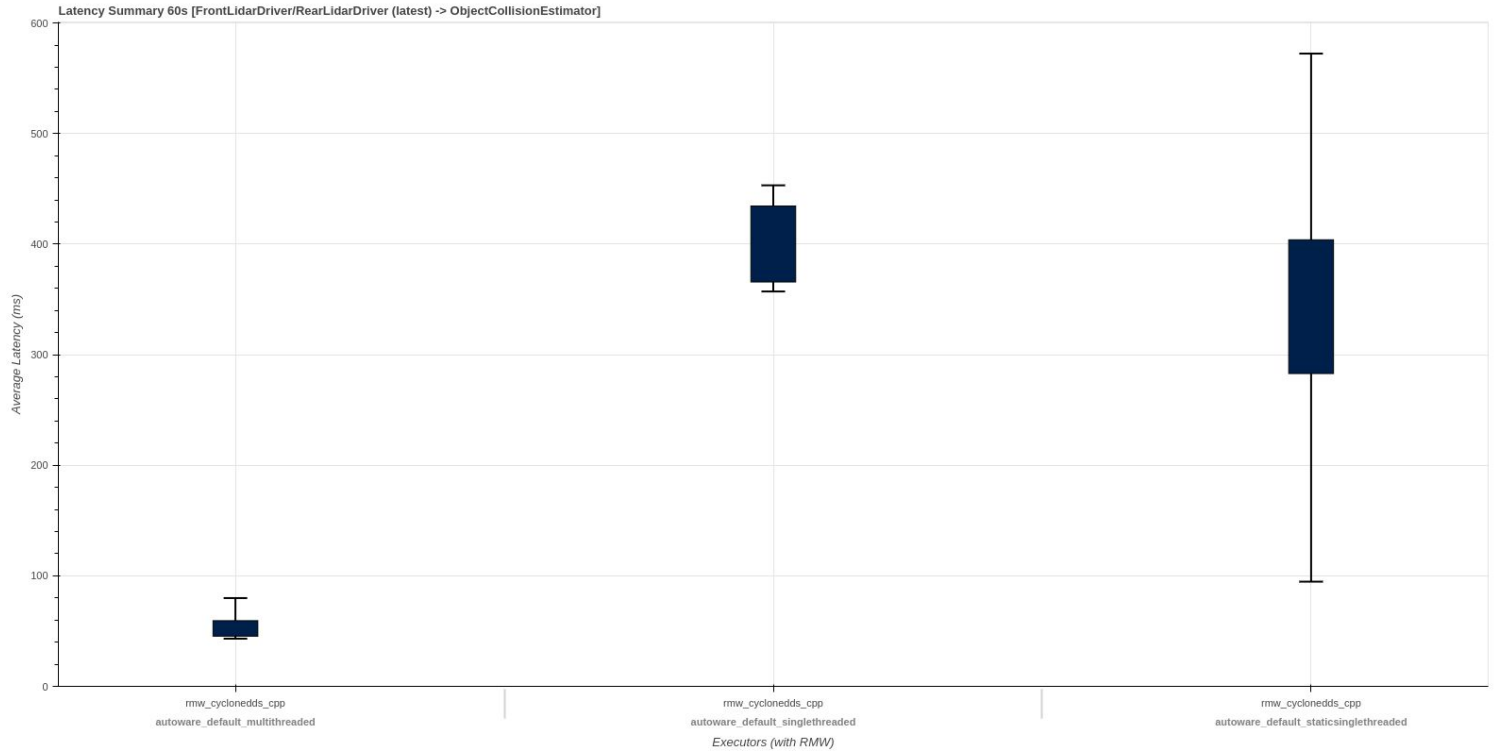
Baseline results - CPU usage



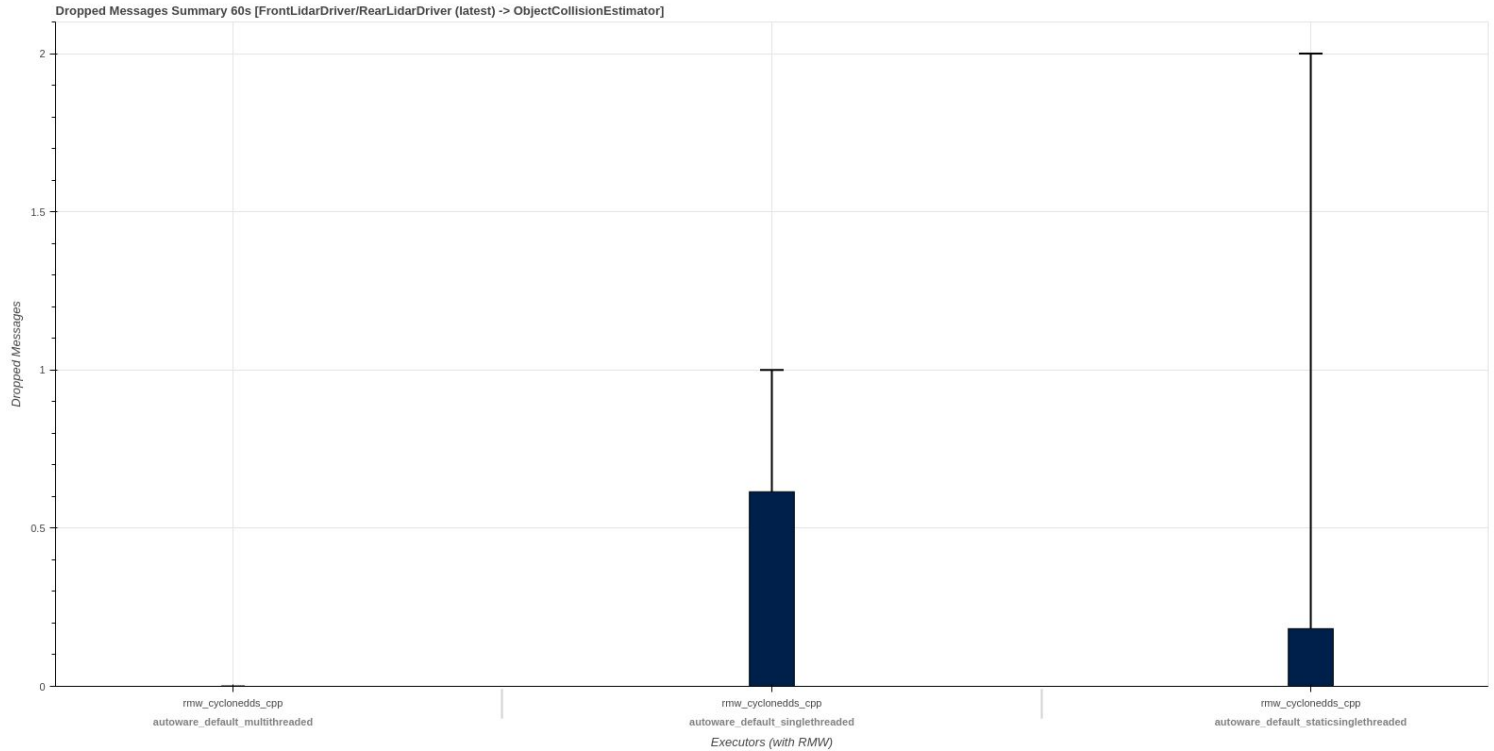
Baseline results - memory usage



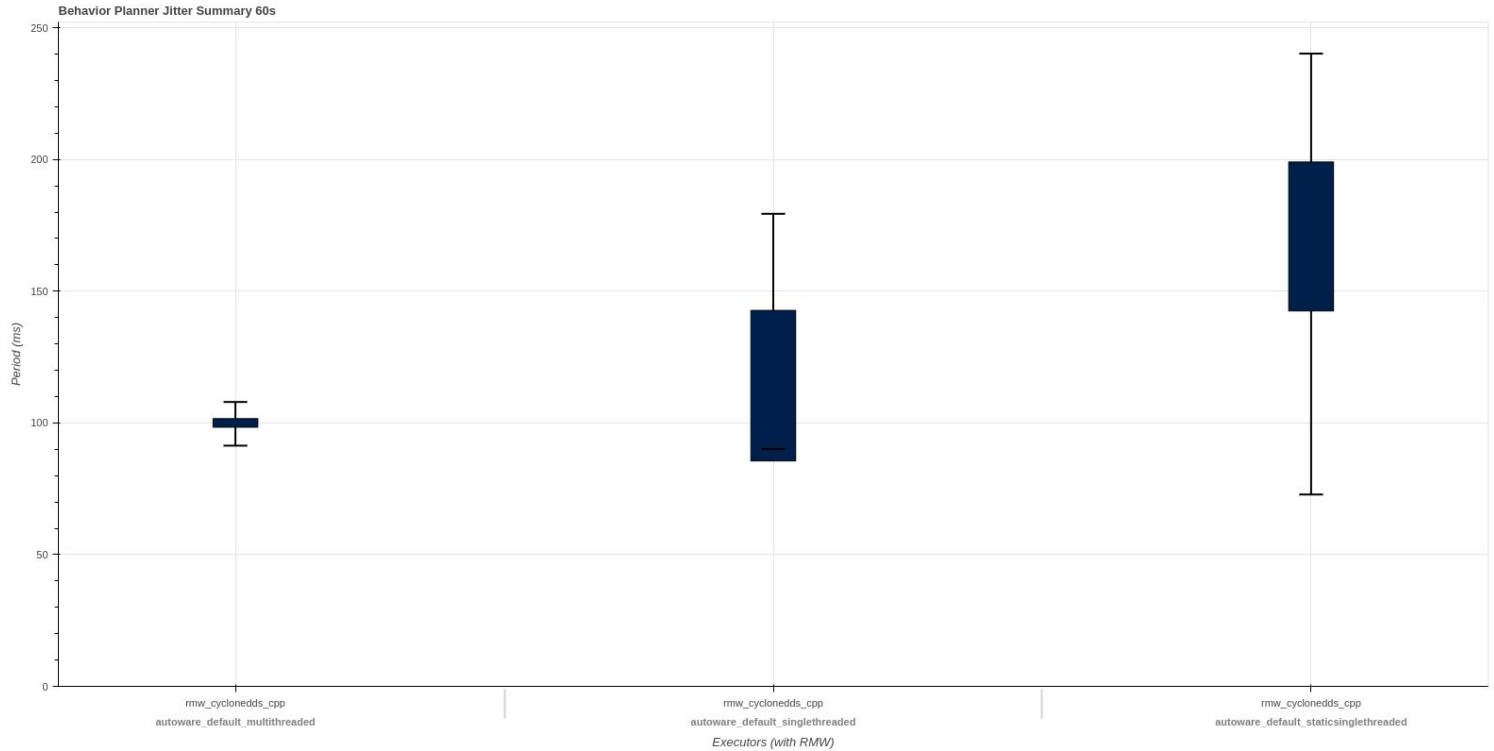
Baseline results - latency



Baseline results - dropped samples



Baseline results - jitter



Baseline results - summary

1. CPU usage
   - *Lower for single-threaded executors
2. Memory usage
   - *Lower for single-threaded executors
   - More relevant when comparing multiple RMW implementations
3. Latency from Front LiDAR to Object Collision Estimator
   - Lowest latency for the multi-threaded executor
   - Static single-threaded executor is better than non-static
4. Dropped samples from Front LiDAR to Object Collision Estimator
   - No dropped samples for the multi-threaded executor
   - Static single-threaded executor is better than non-static
5. Jitter of cyclic Behavior Planner callback
   - Most stable for the multi-threaded executor

For more info about `ros2_tracing`

- Watch my talk tomorrow!
  - Day 1, October 20th, 12:40 PM CDT (UTC-5)
- gitlab.com/ros-tracing/ros2_tracing
- Tutorial
  - real-time-working-group.readthedocs.io/en/latest/Guides/ros2_tracing_trace_and_analyze.html

- github.com/christophebedard
- christophe.bedard@polymtl.ca

Tracing ROS 2 with ros2_tracing

Embedded

12:40 - 13:10 CDT

Christophe Bédard

This talk will introduce tracing, a way to extract execution information, and LTTng, a low-overhead kernel & user space tracer. Then we'll present ros2_tracing, a collection of tools that includes instrumentation directly in the ROS 2 core as well as tools to easily configure tracing through a launch action and a `ros2 trace` command. We'll explain how the instrumentation was designed and show how to use the tools in your own project. Finally, we'll talk about analyzing the trace data and present a demo.