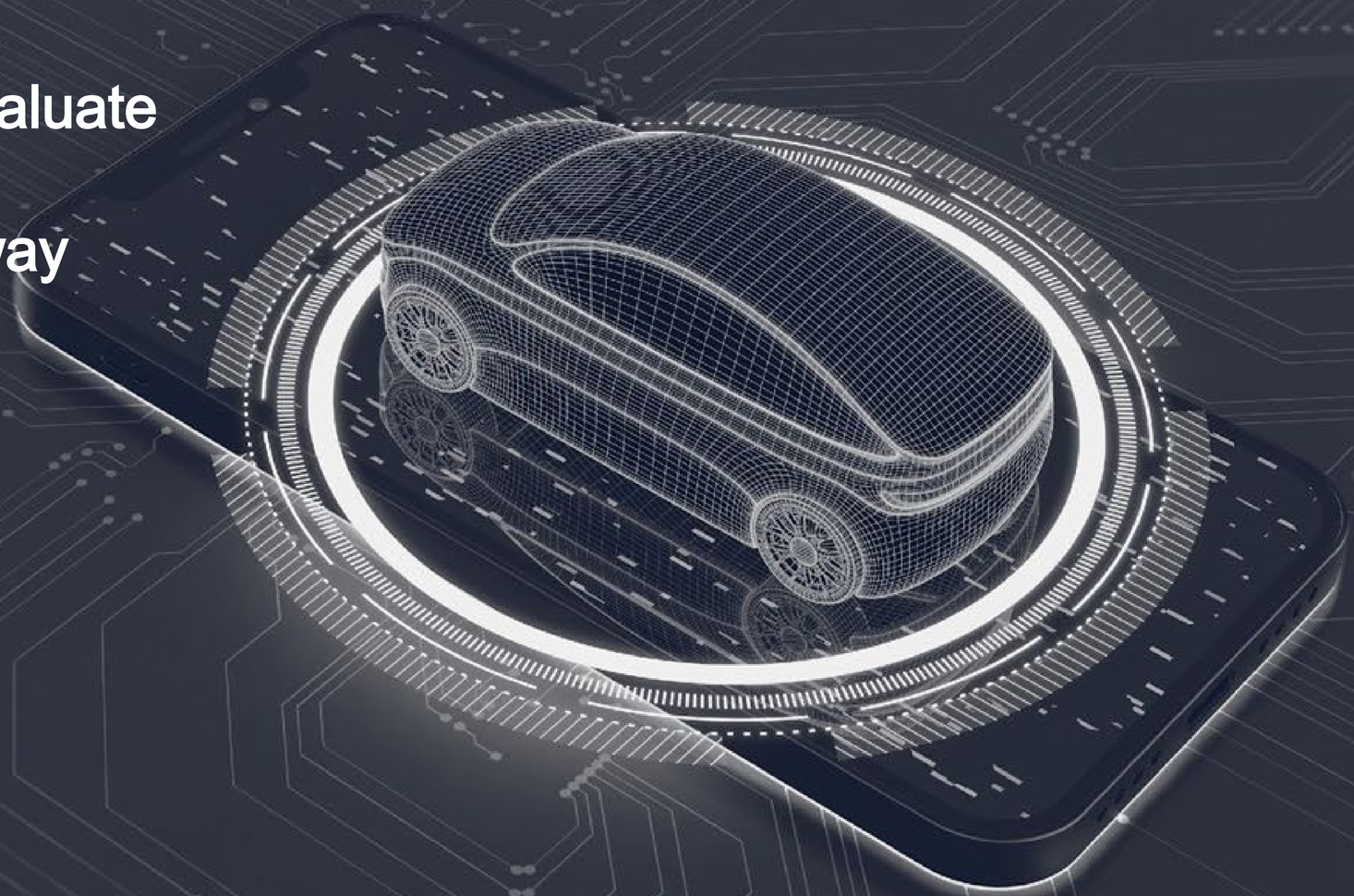
ROSCon 2023

Apex.AI

Using Reference System to evaluate features and performance in a standardized and repeatable way



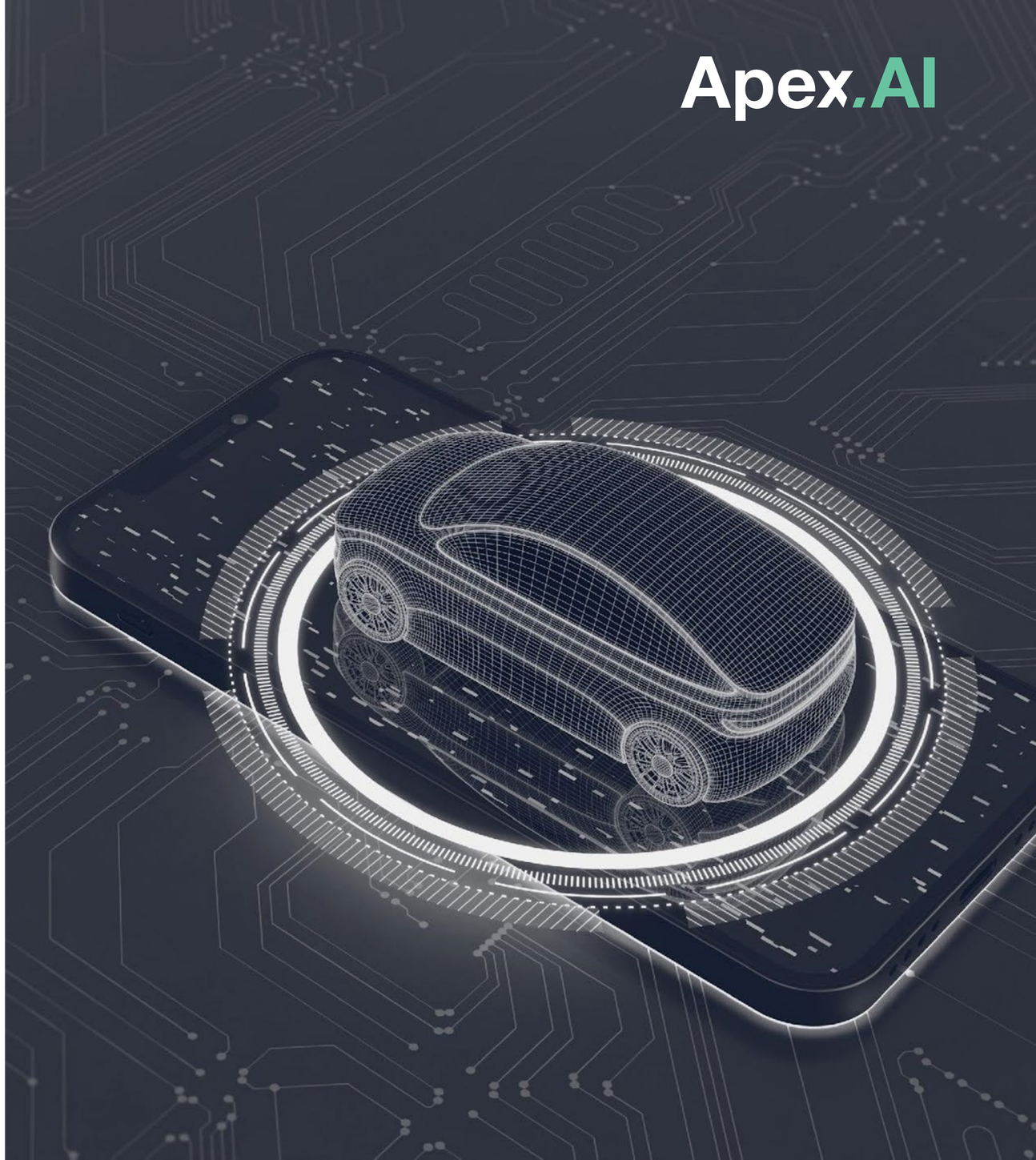
Lyle Johnson
on behalf of Evan Flynn
October 20, 2023

Agenda

1. Filling a tooling gap
2. Execution management configuration
3. Saving time in automotive development

Filling a tooling gap

A Big Bang of executor modes led to the creation of the Reference System tool.



What led to creating Reference System

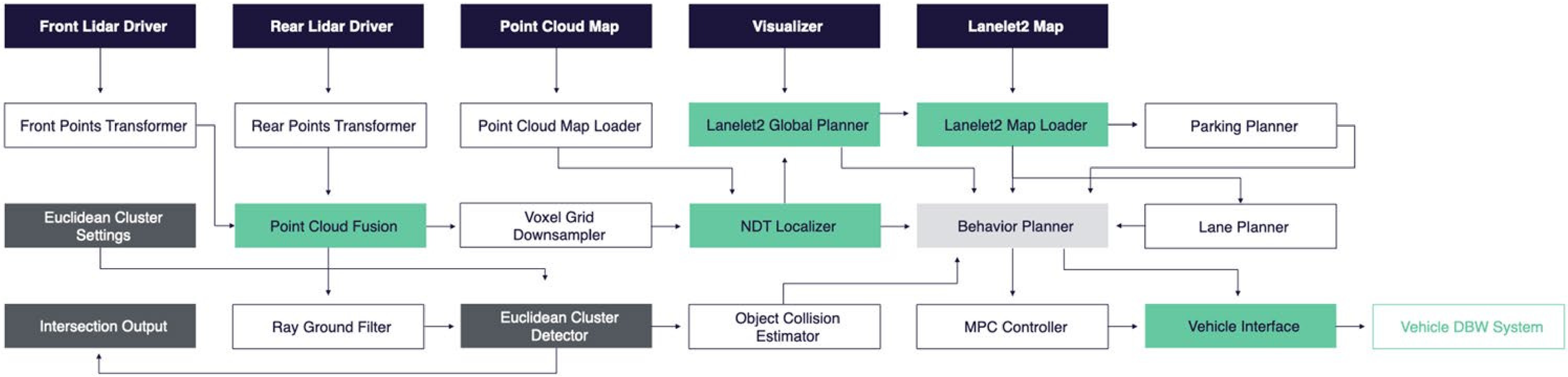
1. Apex.AI developed a custom executor. ROS 2 was adding a few executor modes.

2. Needed a fair and standard way to measure the executor's performance, both against itself and other implementations

3. Xyvr wsyxq i ewyvmk$t i vjsvq er gi $ m$ em}$ evh$H xss$q er } $w} wxi q $ gsr jmkyvexmr w0$ | i gyxsv$ gsr jmkyvexmr w0$ er h$OTM2

4. Need a standardized, repeatable test fixture that can deliver three things:
   A. **Measure performance KPIs** like latency, jitter, CPU, and memory usage
   B. **Flexibility** to allow for **different executors** and various **executor configurations** while still keeping the overall system the same
   C. And **tooling** to automate running experiments and coordinating test campaigns

Creating the Reference System



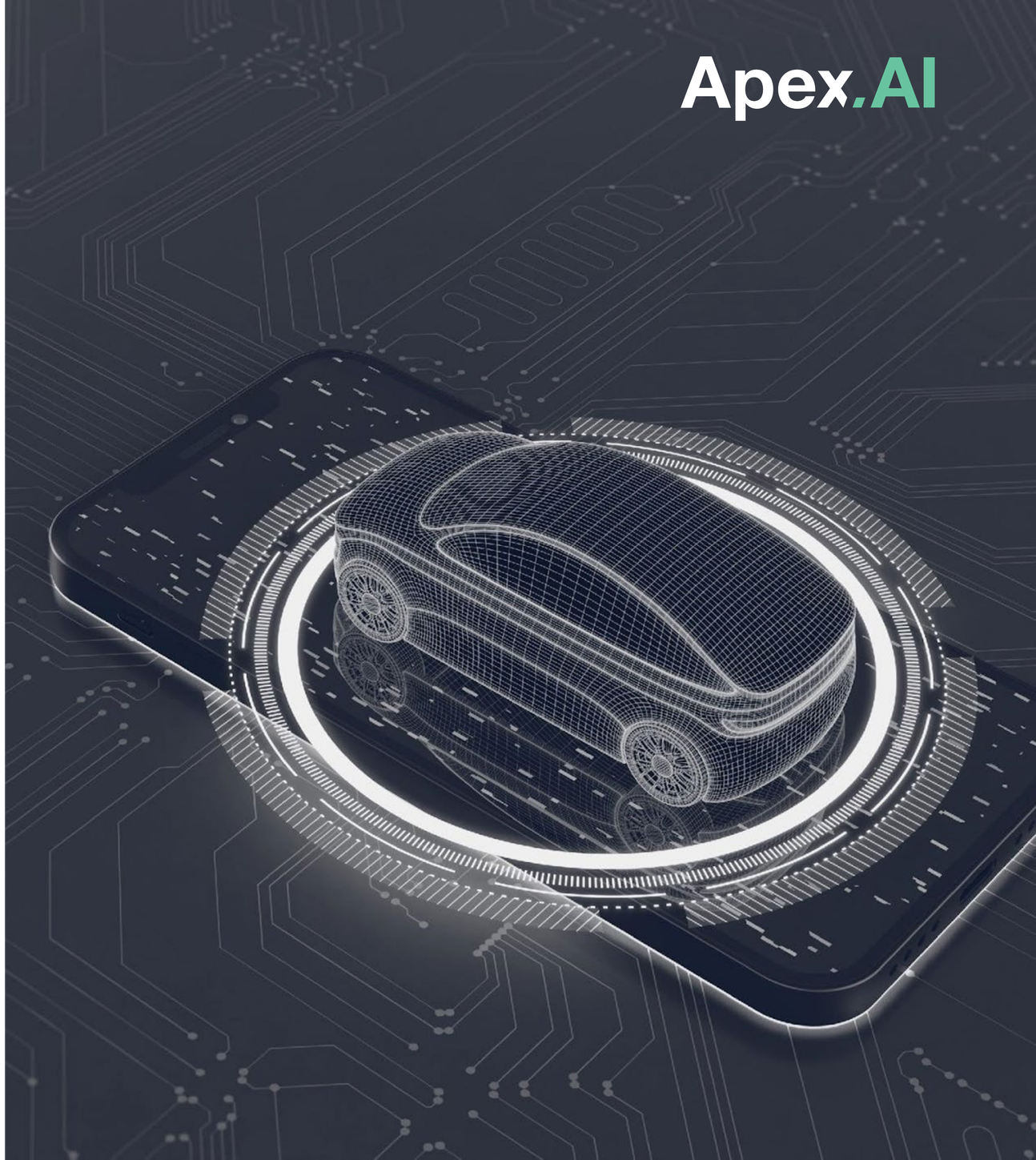
https://ros-realtime.github.io/reference-system/main/

Extending Reference System

1. After ROS World 2021, Apex.AI took the Reference System and brought it in house
2. Integrated it into a **nightly test pipeline** running on a Linux RT Raspberry Pi 4B with fixed CPU frequencies and isolated cores
3. By having this report available nightly, our developers could immediately see key performance improvements or regressions that took place, and address them as needed
4. We also added another Reference System implementation to test new features of the Apex.Grace Executor

Execution management configuration

Knobs to turn and KPIs to track



Execution management configuration

We will isolate specific variables so that we may evaluate their effect on the system.

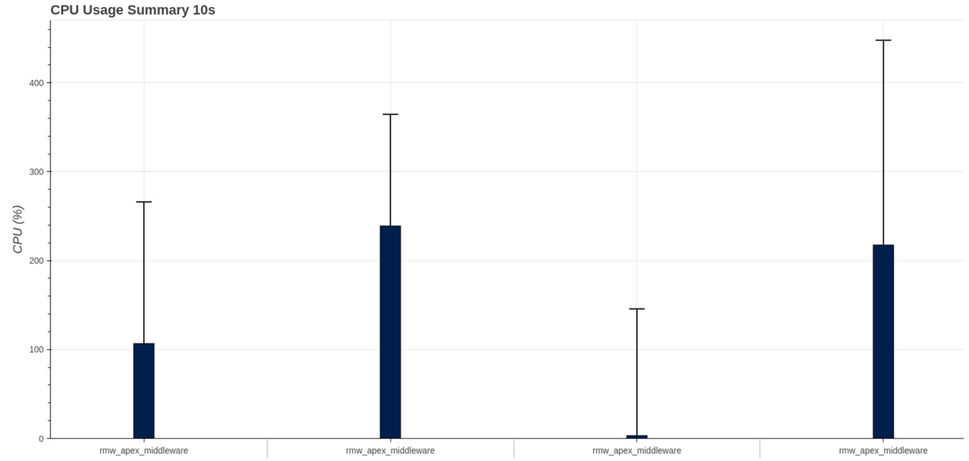
The KPIs are valuable for application developers, architects, and system integrators to better reason about the desired deployment architecture.

A handful of configuration knobs are turned between experiment runs:

- Message size
- Message publishing frequency
- QoS configuration
- Processing time (budget) per node
- Operating mode of the executor

These metrics are measured and used to understand the system behavior:

- CPU utilization
- Memory utilization
- Latency of message delivery
- Dropped messages
- Jitter of cyclic events



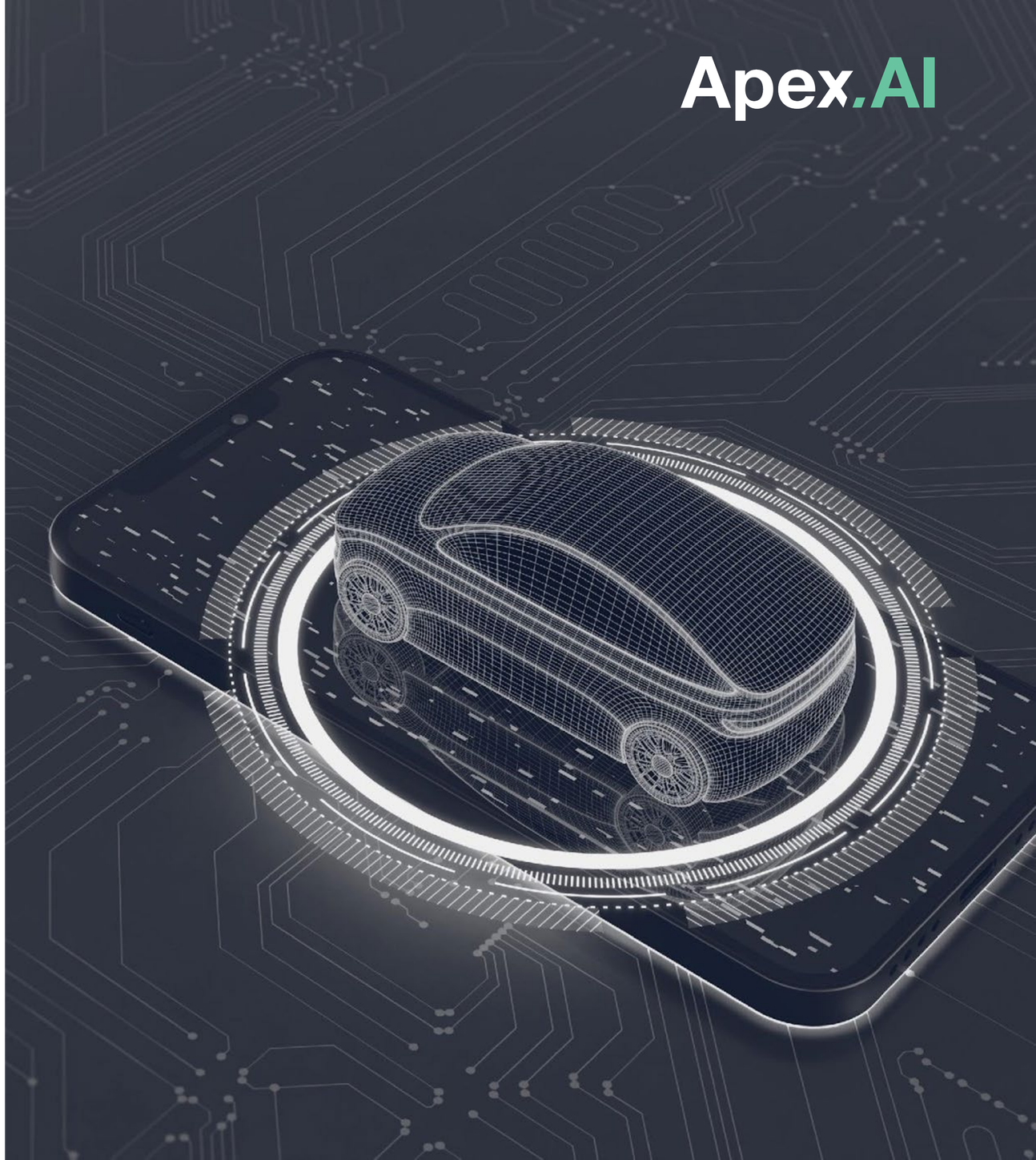
Operating modes for the executor

Overview of the ROS 2 Executor modes that exist with the Iron distro.

- Multi-threaded
- Prioritized
  - Setting the thread priority of the *hot pat h* (the processing pipeline we care about) processes to the highest and set the rest of the threads to lower priorities
- Single-threaded
- Static single-threaded
  - Optimizes the runtime costs by scanning the structure of a node to determine subscriptions, timers, service servers, action servers, etc that the node has
  - It performs this scan only once when the node is added, while the other two executors regularly scan for such changes

Saving time in automotive development

Using the Reference System to detect regressions and improve stability within Apex.Grace



Using Reference System at Apex.AI

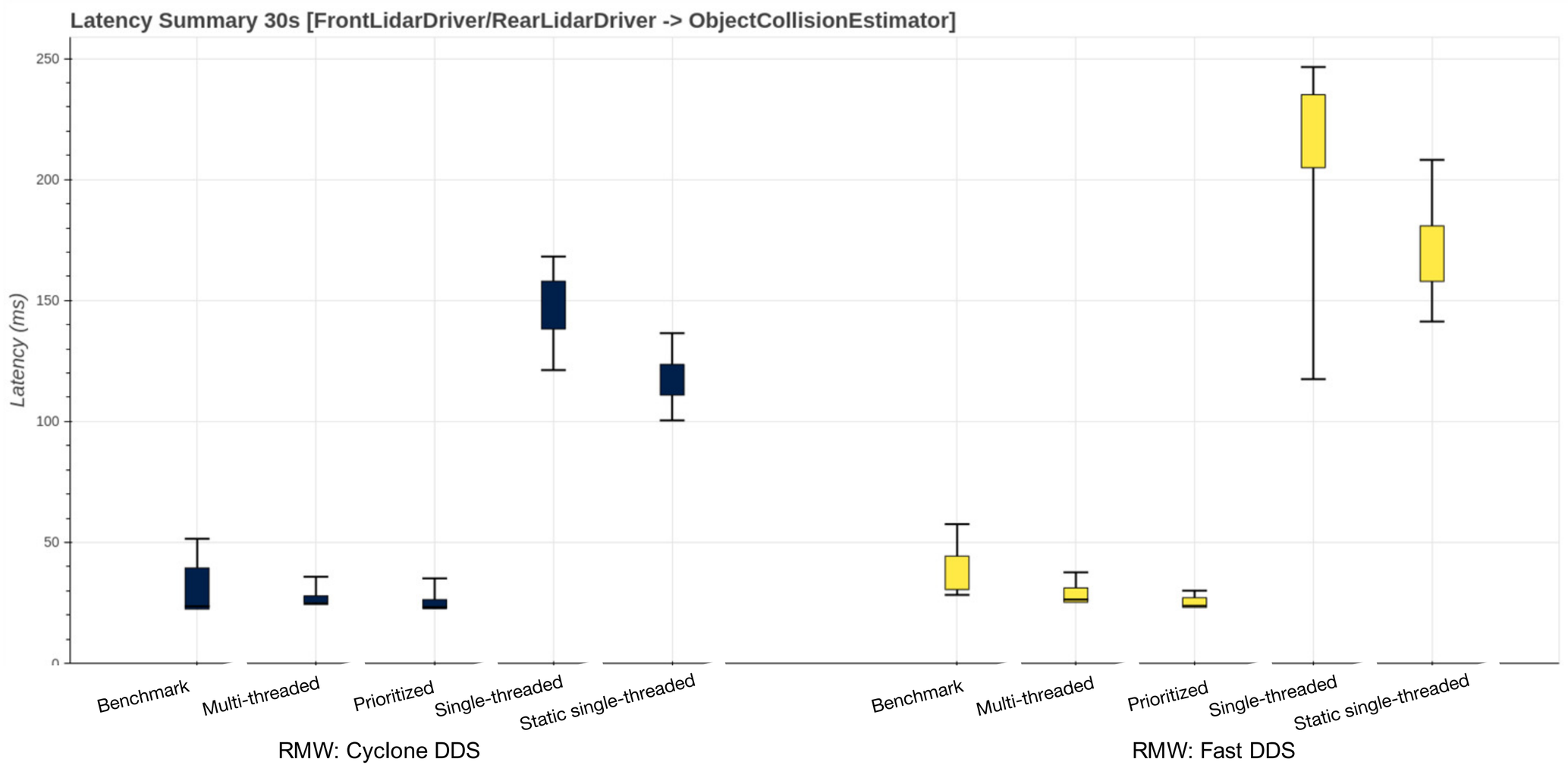
We use the Reference System as part of our CI system for different types of regression testing:

- **Topology**
  - Ensure specific and complex processing chains remain within desired **latency bounds**
  - Generate **empirical evidence** that shows the benefits of one solution over another, both for internal needs as well as in discussions with customers
- **Execution management**
  - **Continuous comparisons** to the ROS 2 executor
    - We care about the Rolling branch
    - But any distro can be tested
  - Perfect companion to the (non-functional) Performance Testing framework
    - `gitlab.com/ApexAI/performance_test`

Let's look at the numbers

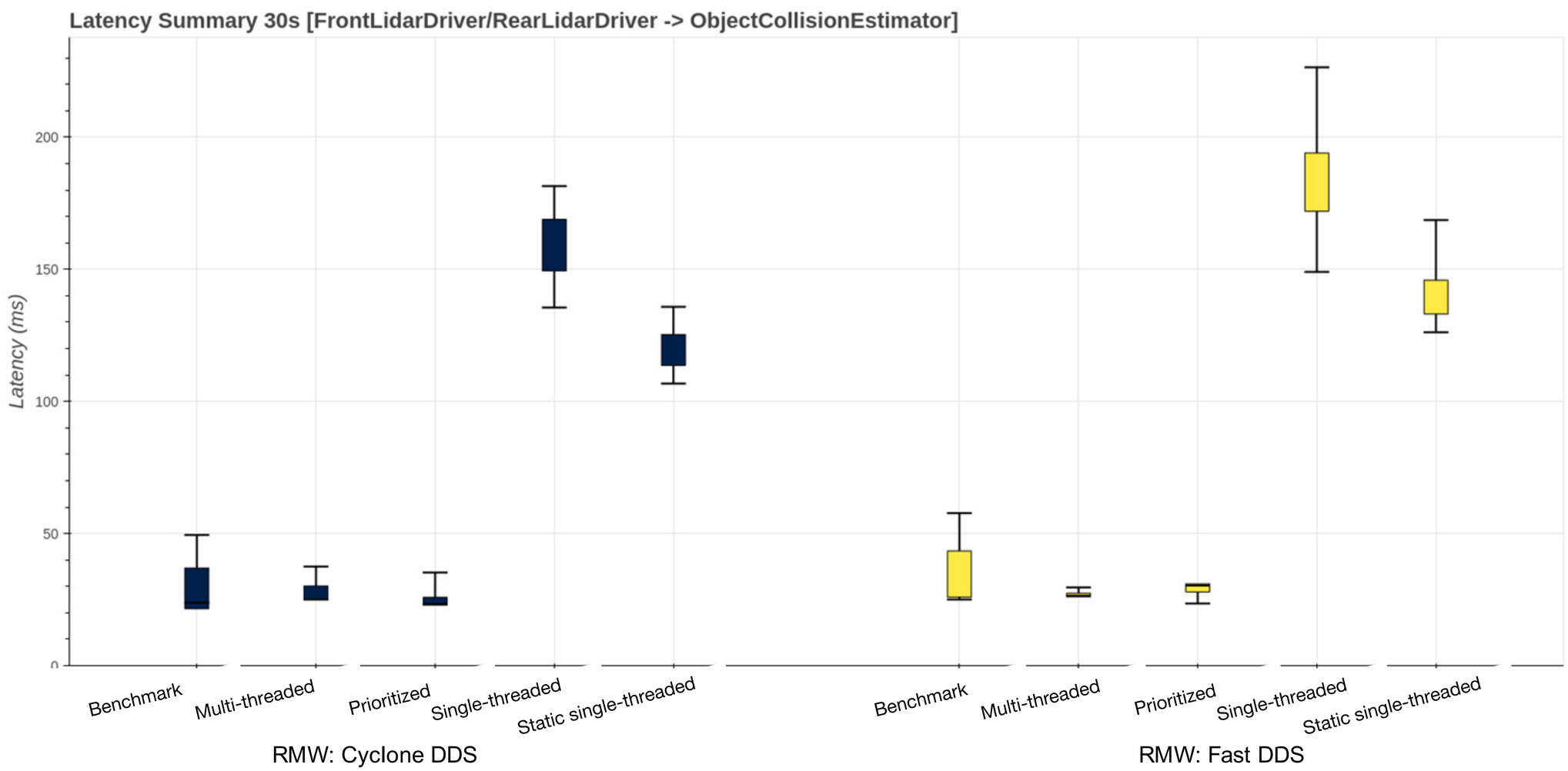
- Latency0svxxlixmqi$mxeoiw$jsv$e$qiwwekixsks$xlvsykl$xli$Ilsx$texl
  - M$xlmw$gewimxglswirxsfi$jvsq$xli$phev$rshixsxli$sfnigx$gspprwsr$iwxmqexsv$wmqypexmrk$e$viep${svph$wgirevms
- Jitter sj$e$xmqi1xvmkkivih$rshi
  - Jsv$i|eqtpi0$e$rshi$wixxsfi$xvmkkivih$iziv}$544q w$qe}fixvmkkivih$wpmklxp}$fijsvisvejxiv$544q w$ew$ksri$f}
- Memory and CPU usage
  - Xli$qiqsv}$erhGTYywekigerzev}$hitirhmrk$srxlii|igyxsv$ywih
  - Qsvi$qiqsv}$erhGTYywekimwrsx$rigiwwevmp}$efehxlmrk0$wmrgi$mx$qmklx$ywih$qier$xli$i|igyxsv$mw$tvsgiwwmrk$ qsvi$hexe

Latency — Humble



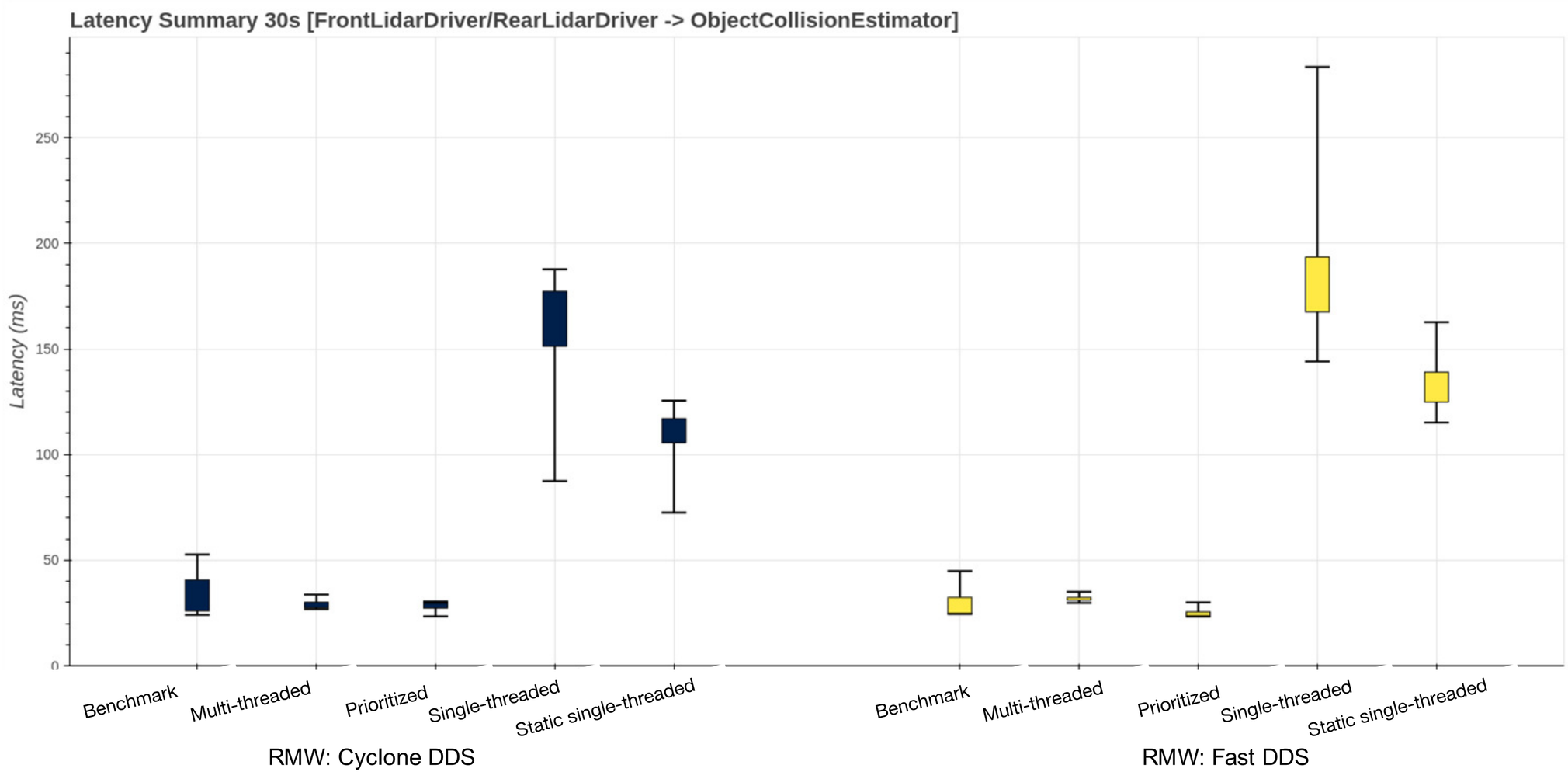
Latency variance of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

Latency — Iron



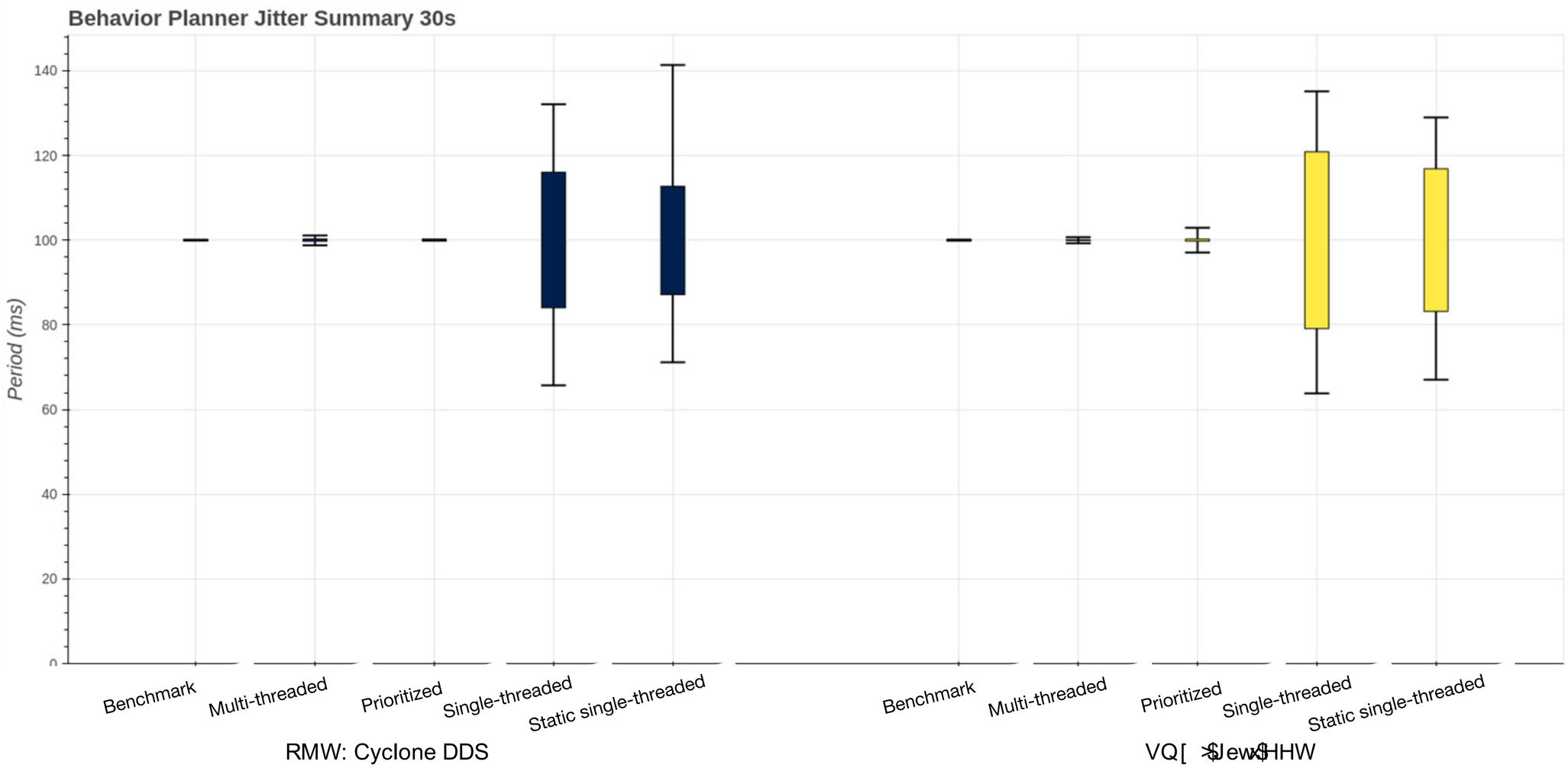
Latency variance of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

Latency — Rolling



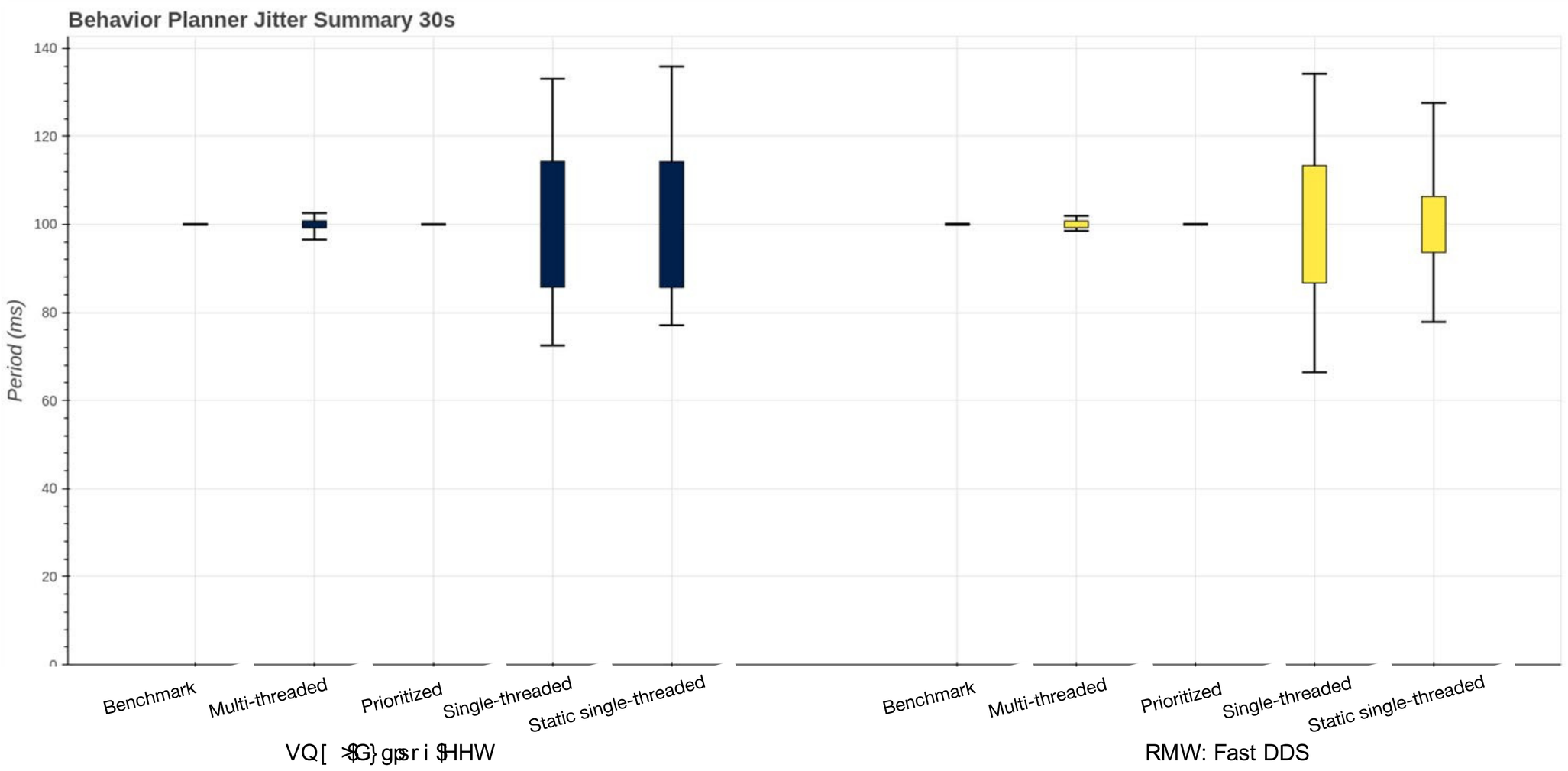
Latency variance of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

Jitter — Humble



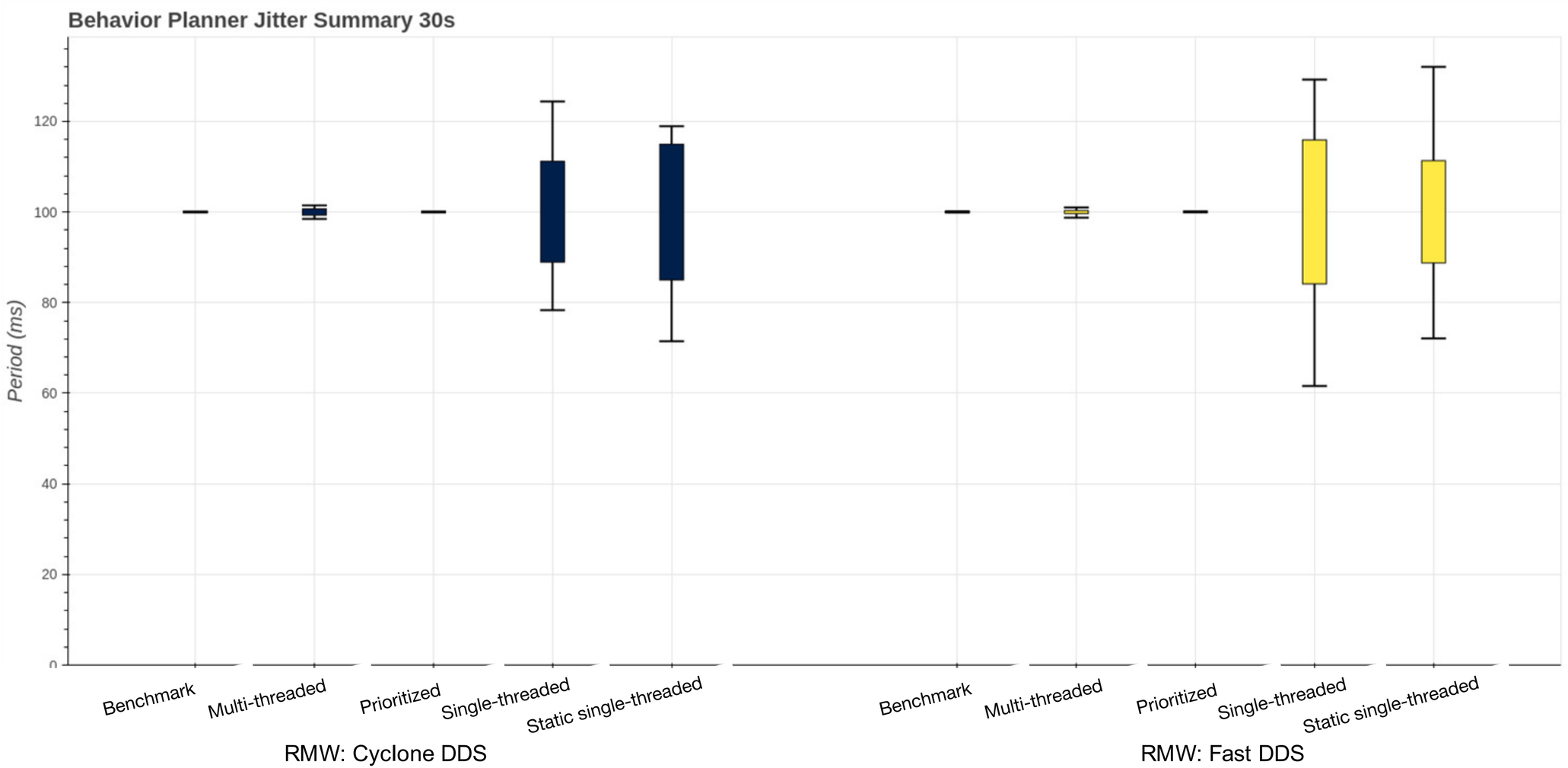
Execution jitter of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

Jitter — Iron



Execution jitter of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

Jitter ╟ Rolling

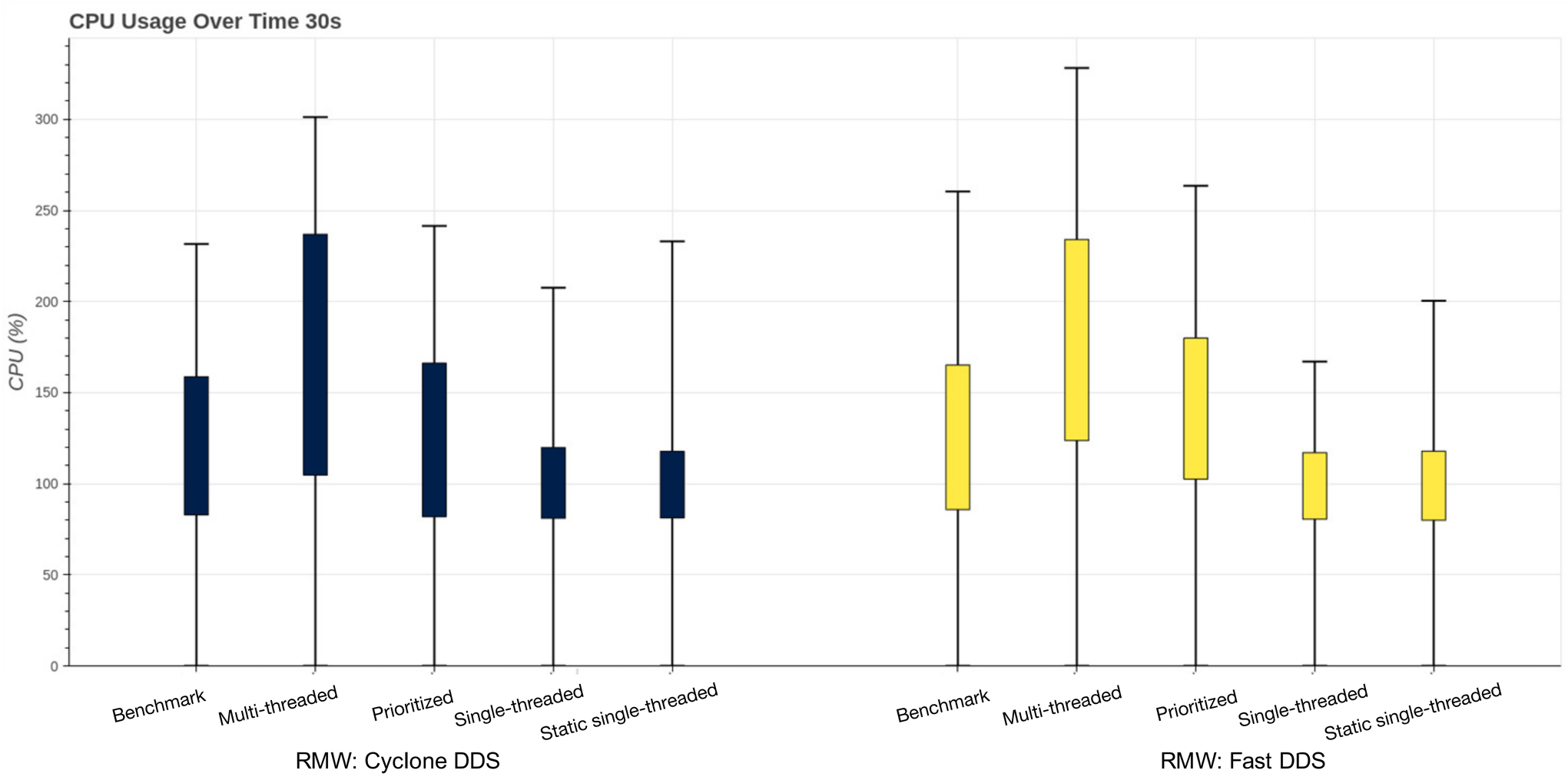


RMW: Cyclone DDS

RMW: Fast DDS

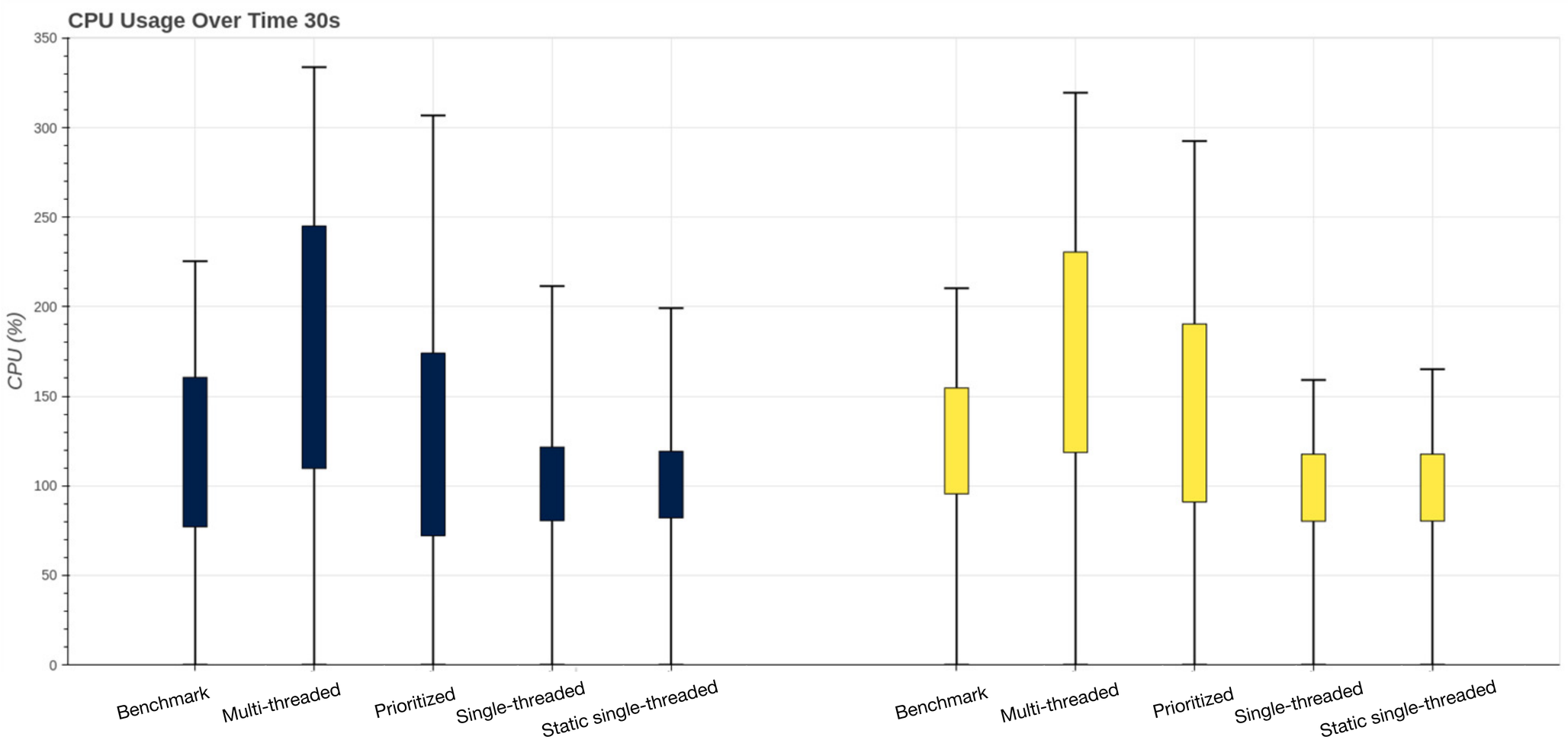
Execution jitter of the different executor modes in ROS 2 Humble to Rolling with two RMW layers

CPU — Humble



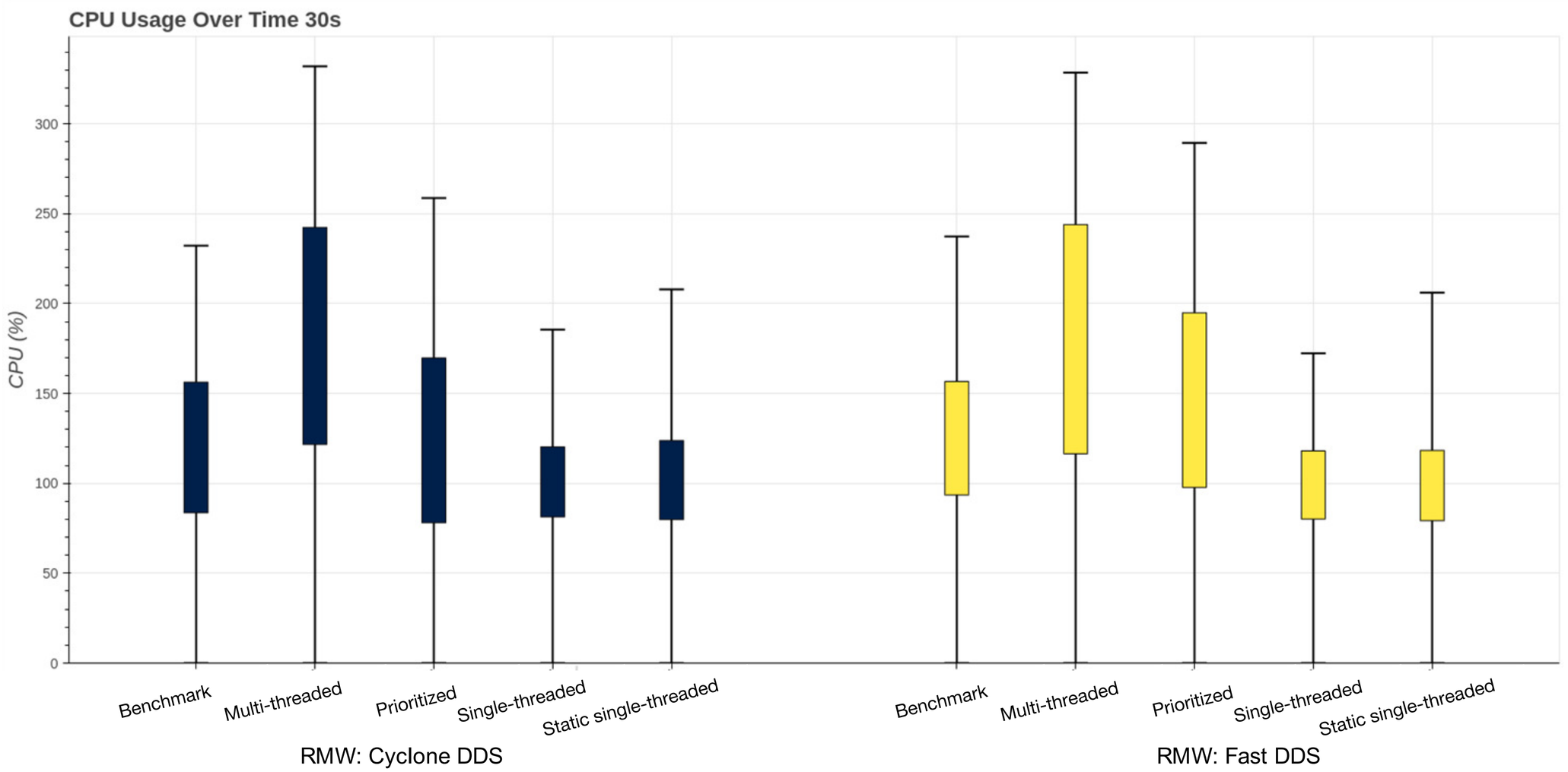
CPU consumption of the different executor modes for ROS 2 Humble to Rolling with two RMW layers

CPU — Iron



CPU consumption of the different executor modes for ROS 2 Humble to Rolling with two RMW layers

CPU — Rolling



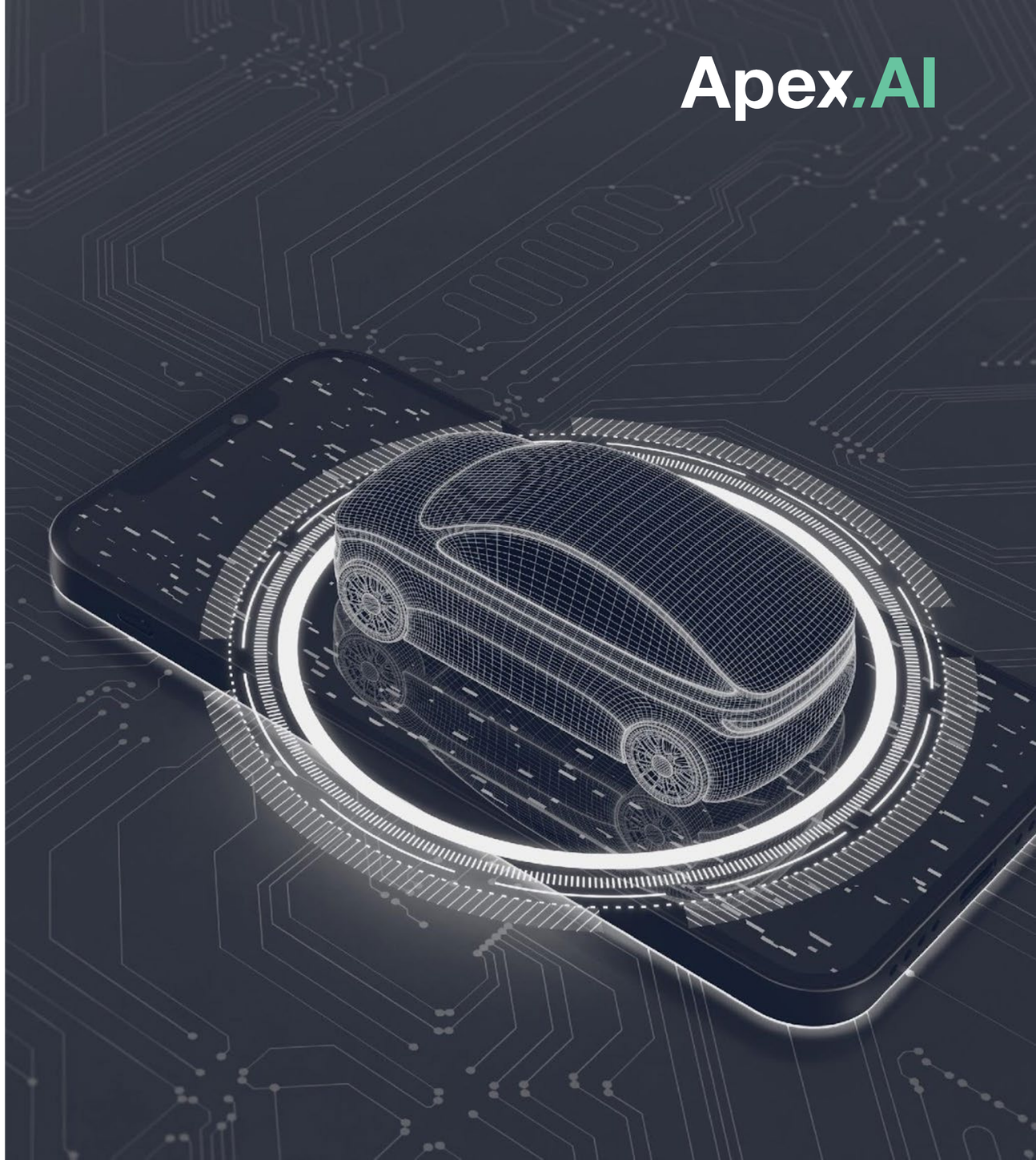
RMW: Cyclone DDS

RMW: Fast DDS

CPU consumption of the different executor modes for ROS 2 Humble to Rolling with two RMW layers

In closing

Prepare your questions!



Use Reference System as part of your prototyping workflow

¡

Thank you!

Apex.AI

evan.flynn@apex.ai
lyle.johnson@apex.ai