# ROSCon 2023

Improving your application's algorithms and optimizing performance using trace data

#### Christophe Bédard October 20, 2023

© 2023 Apex.AI, Inc. | All rights reserved. Apex.AI, Apex.OS, Apex.Grace, Apex.Ida, Apex.Alan, Apex.Ray are trademarks of Apex.AI.







We'll learn how to use trace data to improve performance

- **1. Introduction**
- 2. Tracing
- **3. Instrumentation**
- 4. Trace data analysis
- 5. Examples and use-cases
- 6. Conclusion



### Introduction

- Performance analysis
- Understand what happened during execution
- Extract high-level information
- Find cause of bugs
- Identify misconfigurations
- Optimize performance
- Extract various performance metrics





# Tracing

- Low-level software tracing
  - Fast low-level recording at runtime
  - Low performance impact
  - Payload is usually raw binary data
  - Needs to be processed for it to be useful
- Need to instrument source code in order to collect data
  - Applications
  - Linux kernel (built-in)
  - Drivers
  - Etc.
- ros2 tracing: tracing instrumentation and tools for ROS 2
- This presentation is about trace data processing, not ros2 tracing itself
  - See ROS World 2021 presentation: *Tracing ROS 2 with ros2\_tracing* (vimeo.com/652633418)
  - Or the paper: ros2\_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2 (doi.org/10.1109/LRA.2022.3174346)

- Record low-level execution information
- Trace data needs to be processed





# Why trace? When to trace?

- Tracing is one of many tools in our huge toolbox
- Not always the right tool
  - Optimize a specific function or lines of code
  - Debug a specific function or lines of code
  - What happened at 9:27 pm?
- Tracing examples
  - One instance is taking longer than usual: I/O, kernel scheduling, etc.
  - Anomalies or unexpected behaviour with messages or system: logic bug, executor misconfiguration, etc.
- Especially useful to understand complex systems
  - Like large and/or distributed systems
  - Since complex systems can make debuggers less effective
- Also useful in general to visualize a system
  - Might give hints to optimize your system
  - Even if you're not necessarily looking for performance issues

profiler debugger logs

 $\rightarrow$ 

Tracing is useful to understand the execution of complex systems.





### Instrumentation - ros2 tracing

- Instrumentation is built into ROS 2 by default on Linux starting from ROS 2 Iron Irwini
- Information about the main elements of ROS 2
- Objects
  - Node, publisher, subscription, timer
- Events
  - Callback execution (subscription, timer)
  - Message publication, message taking
  - Internal executor phases, etc.
- Uses the LTTng tracer for instrumentation and recording

- Information about callbacks and messages
- Now included by default on Linux as of ROS 2 Iron Irwini









### Instrumentation – Custom

- 1. Define your tracepoints in a tracepoint provider (shared library)
  - a. Tracepoint names
  - b. Arguments
- 2. Add tracepoints to your code
- 3. Run your application and collect data
  - ros2 trace --ust `ros2:\*' `my app:\*' \$
- This is a simplified version
- For more information, see: Ittng.org/docs/v2.13/#doc-instrumenting

```
// tp.h: tracepoint definition in tp provider header
#include <lttng/tracepoint.h>
LTTNG UST TRACEPOINT EVENT (
 my_app, my_tracepoint,
 LTTNG_UST_TP_ARGS(int, count_arg),
 LTTNG UST TP FIELDS (
    lttng ust field integer(int, count, count arg))
  Insert tracepoint somewhere in your code
#include "tp.h"
lttng_ust_tracepoint(my_app, my_tracepoint, 42);
```





### **Trace data analysis — How to use trace data?**

- Combining trace data from multiple sources helps understand the execution
  - Multiple applications, including ROS 2 + additional application-specific info
  - Linux kernel
  - Distributed systems (with time synchronization)
- Examples
  - ROS 2: callback executions, message publications
  - Linux kernel: scheduling, I/O, system calls, etc.
- Add information specific to your own nodes
  - Instrument and trace your nodes
  - Can provide information about the processing performed by your nodes
- Trace processing tools
  - tracetools analysis: very basic Python API
  - Eclipse Trace Compass: powerful trace viewer and analysis framework
- Trace Compass can display Linux kernel and ROS 2 trace data
  - And more!

Combine and visualize trace data from multiple sources or layers.

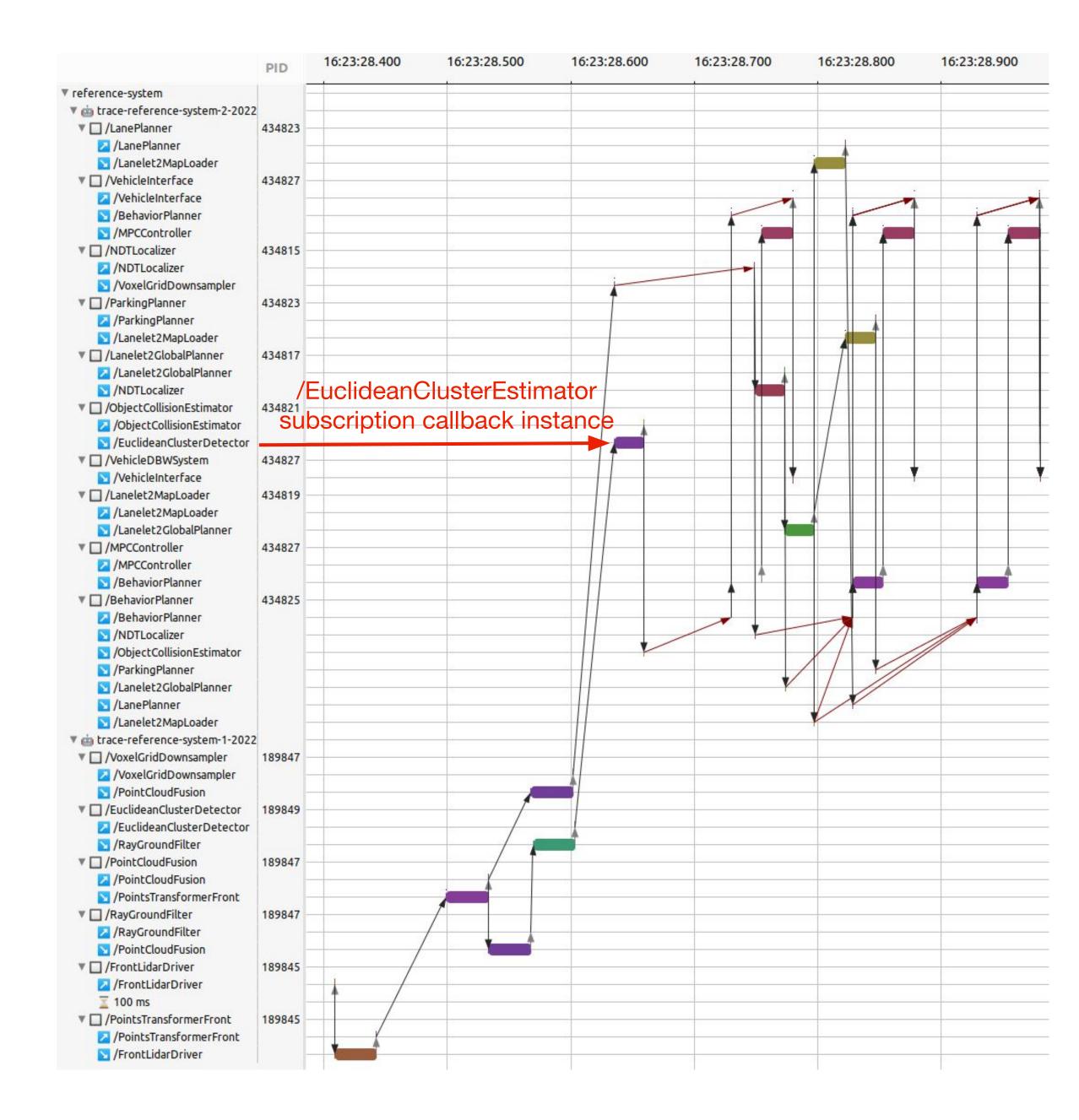




#### Example

- Viewing ROS 2 trace data with Trace Compass
- Horizontal axis is time
- Rows
  - is system (1 trace/system)
    - node
      - 🕨 🔀 timer
      - Subscription
      - 🔁 publisher
- Shows
  - Rectangles: callback executions
  - Arrows: message publications
- Autoware reference\_system
- For a single end-to-end process pipeline instance starting from one LiDAR message

Figure 1. Callbacks and messages over time in Trace Compass.







### **Example (2)**

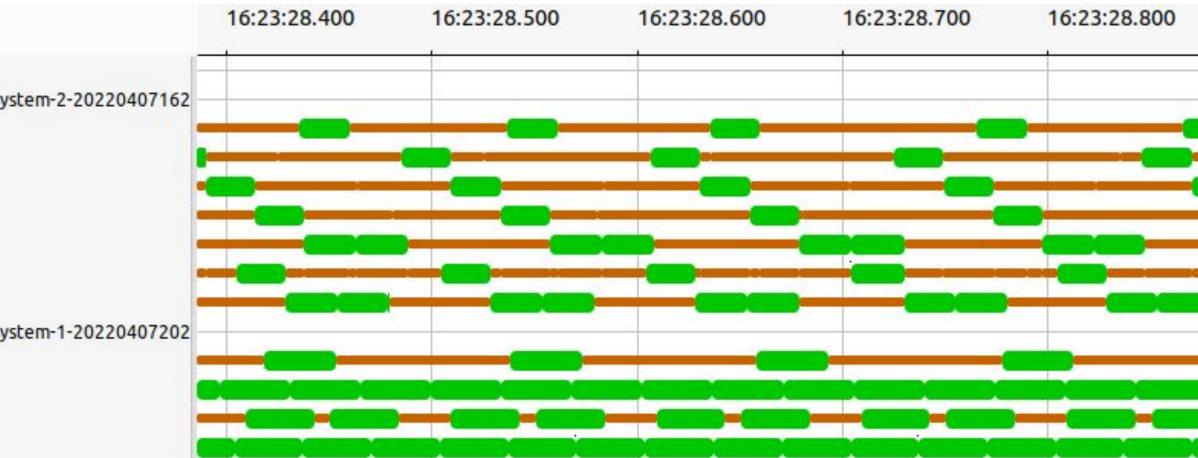
- State of executors over time
  - Green: executing (callback)
  - Orange: waiting for work
- 1 executor/process
- Some executors are busier than others
- Could explain message processing delays
  - Message reception  $\rightarrow$  callback

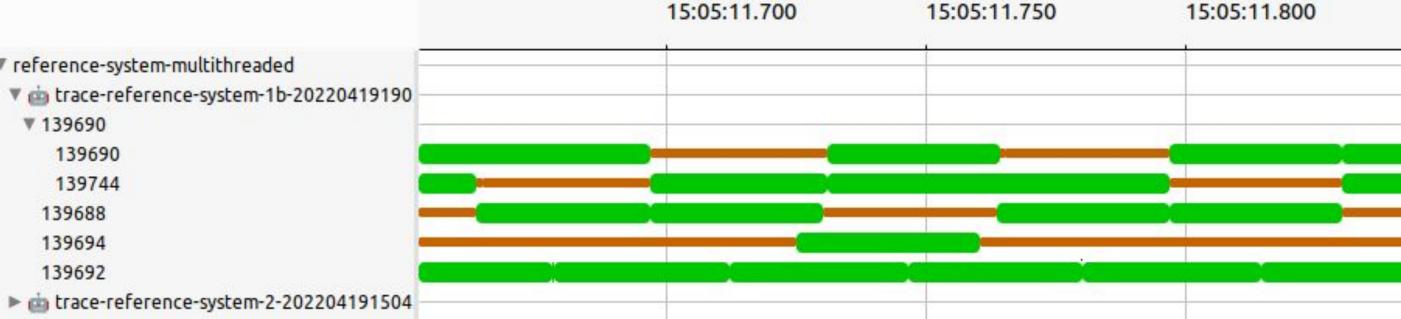
▼ reference-system
🔻 🤖 trace-reference-sy
434821
434815
434817
4 <mark>34</mark> 819
434823
434825
434827
🔻 🤖 trace-reference-sy
189855
189847
189845
189849

▼ reference-system-multithreaded ▼ 139690 139690 139744 139688 139694 139692

Figure 2. State of single-threaded executors over time.

Figure 3. State of multi-threaded and single-threaded executors over time.









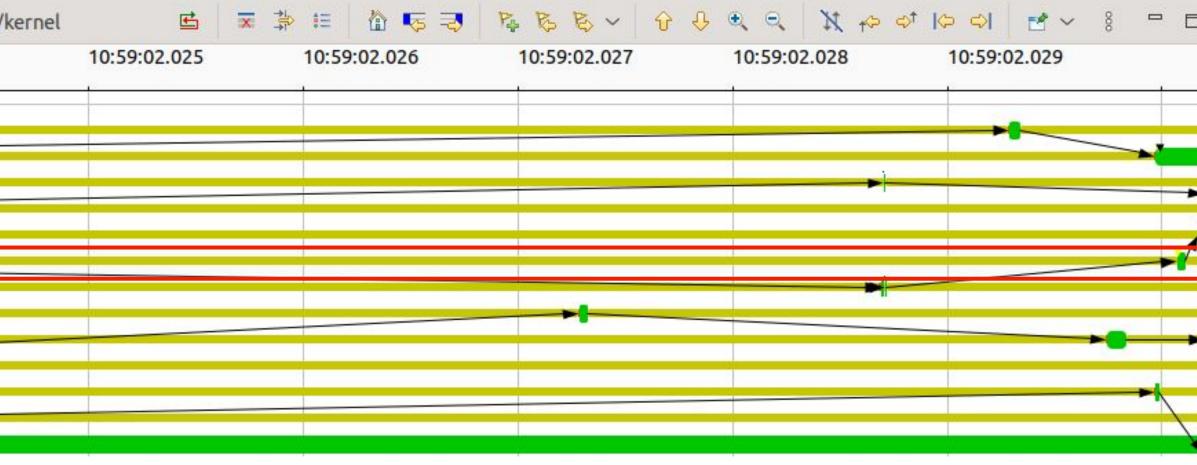




### Example (3)

rocess	TID	PID		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025	10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
trace-reference-syster	m-										
java	<ul> <li>Sector and a sector and a secto</li></ul>	203145									
autoware_defaul	252359	252359 =									
workqueue	2180										
autoware_defaul	252424	252424		-							
Chrome_ChildIOT	88012	88012 -									
autoware_defaul		252409	<u> </u>	/							
wqtimer	1786	1786		/							
Xorg	3497	3497 🗧									
VizCompositorTh	88013	88013									
recvUC	252503	252503		1							
recvUC		252391									
					<u> </u>						
tev	252483	252483		$\checkmark$							
tev autoware_defaul Executor (incubat		252483		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second	<b>№ № №</b> √ <del>0</del> <del>0</del> <del>0</del> <del>0</del>	the second s
autoware_defaul	252418			10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025	10:59:02.026		<b>№ 10:59:02.028</b>	<mark>0,                                    </mark>
autoware_defaul	252418 :or) ×	252418		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat trace-reference-syster i i trace-reference-syster	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat trace-reference-syster @ trace-reference-syster 252361	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat trace-reference-syster @ trace-reference-syster 252361 252363 252359 252359	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat trace-reference-system @ trace-reference-system 252361 252363 252359 252359 252409	252418 :or) ×	252418 • 09081058		10:59:02.022	10:59:02.023	10:59:02.024	10:59:02.025		and the second		the second s
autoware_defaul Executor (incubat trace-reference-syster @ trace-reference-syster 252361 252363 252359 252359 252409 252357	252418 :or) ×	252418 • 09081058		10:59:02.022			10:59:02.025	10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
autoware_defaul Executor (incubat trace-reference-system @ trace-reference-system 252361 252363 252359 252359 252359 252357 252416	252418 :or) ×	252418 • 09081058		10:59:02.022				10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
autoware_defaul Executor (incubat trace-reference-system trace-reference-system 252361 252363 252359 252359 252359 252409 252357 252416 252420	252418 :or) ×	252418 • 09081058		10:59:02.022				10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
autoware_defaul Executor (incubat trace-reference-system trace-reference-system 252361 252363 252359 252359 252359 252409 252357 252416 252420 252418	252418 :or) ×	252418 • 09081058		10:59:02.022				10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
autoware_defaul Executor (incubat trace-reference-system (a) trace-reference-system (b) trace-reference-system (c) trace-reference-system	252418 :or) ×	252418 • 09081058		10:59:02.022				10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029
autoware_defaul Executor (incubat trace-reference-system cip trace-reference-system 252361 252363 252359 252359 252359 252409 252416 252416 252418	252418 :or) ×	252418 • 09081058		10:59:02.022				10:59:02.026	10:59:02.027	10:59:02.028	10:59:02.029

Figure 4. Kernel scheduling view (control flow) vs. executor view: thread scheduling explains some executor delays.





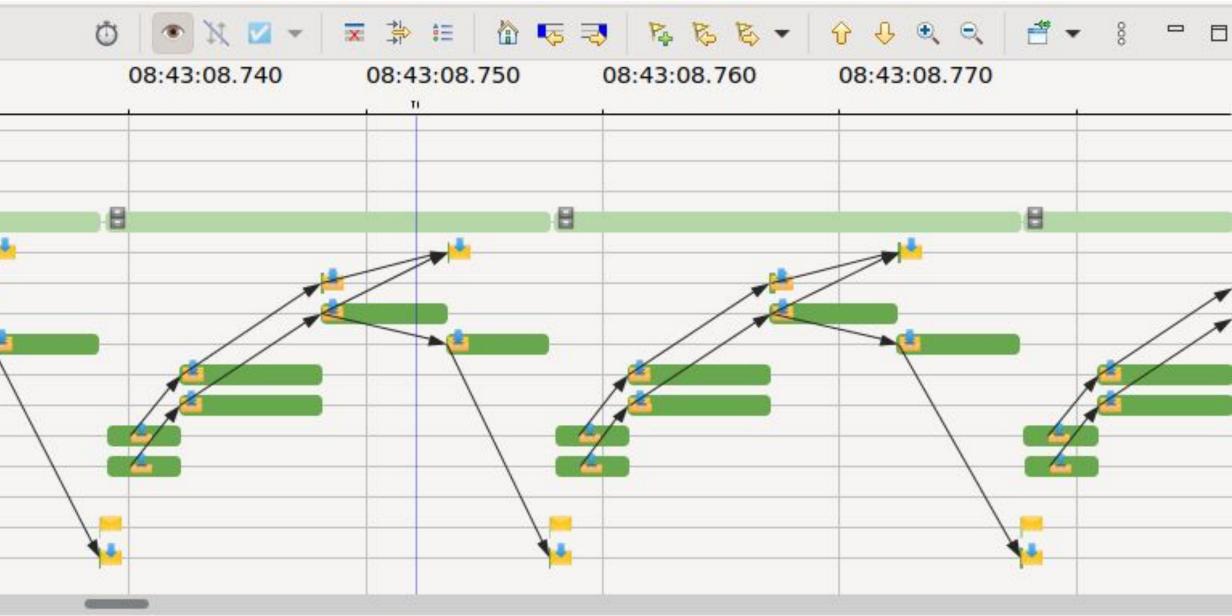


### **Example — Taking it to the next level**

- We've implemented a Trace Compass plugin for Apex.Grace, our fork of ROS 2
- This is somewhat specific to our custom executor
  - Executor-centric view (below) vs. node-centric (previously)
  - See presentation from ROS World 2021 executor workshop: *Executor with wait-set and polling subscription*

	ID	PID	Host
2023-05-04_lidar_perception_stack_lidar_pe			
v is 2023-05-04_lidar_perception_stack_lid			ade
▼ executor 1	1	109494	
▼task 1	1		
item /::block_fusion_node#1	/::block_fusion_node#1		
item /::filter_transform_vlp16_front#1	/::filter_transform_vlp16_front#1		
item /::filter_transform_vlp16_rear#1	/::filter_transform_vlp16_rear#1		
item /::ray_ground_classifier_block_node#1	/::ray_ground_classifier_block_node#1		
item /::vlp16_front#1	/::vlp16_front#1		
item /::vlp16_rear#1	/::vlp16_rear#1		
item /::vlp16_udp_driver_node_front#1	/::vlp16_udp_driver_node_front#1		
item /::vlp16_udp_driver_node_rear#1	/::vlp16_udp_driver_node_rear#1		
▼ executor 2	2	109494	
▼task 1	1		
item /::euclidean_cluster_block_node#1	/::euclidean cluster block node#1		

Figure 5. State of Apex.Grace executor and message publications over time.





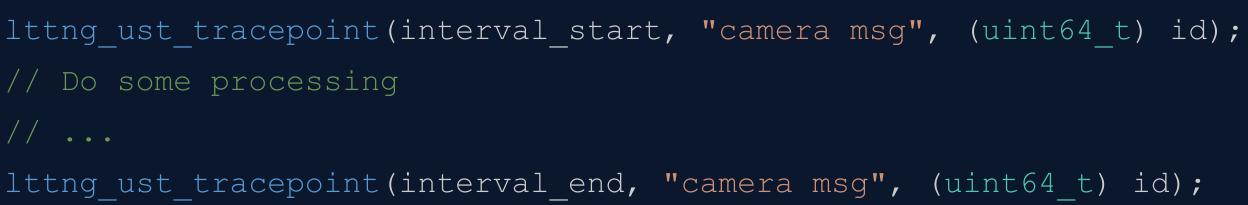


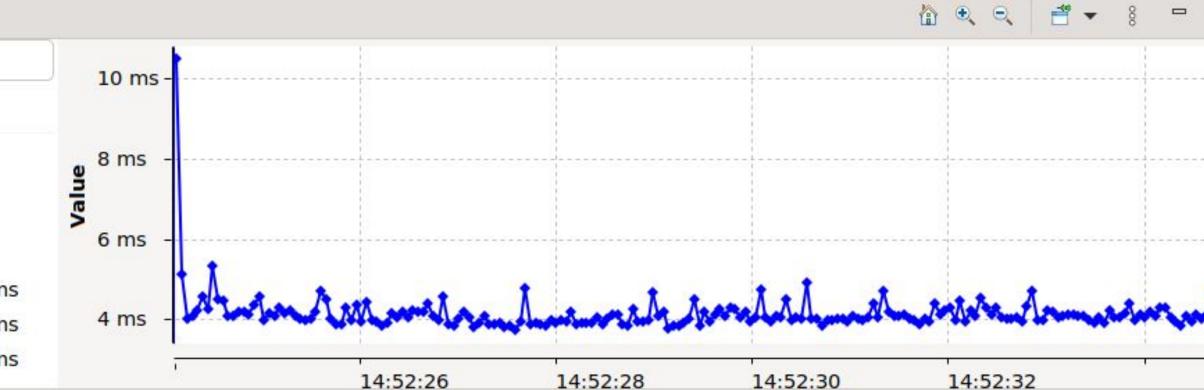
# **Example — Taking it to the next level (2)**

- We can see the callback executions over time
  - What if we want to know more about what happens in our callback?
- Custom tracepoints to provide information about processing done inside a function or callback
- Display durations over time

Apex.Grace Metrics ×					
type filter text					
	Legend	Mean	Std	Min	Max
- a a 2022-12-17_16794-intervals-fusion-worklos	ad				
Executors					
👻 🚍 Intervals					
👻 🚍 camera msg					
delay		47.825 ms	3.433 ms	0	51.056 m
duration		4.16 ms	517. <mark>271 μ</mark> s	3.734 ms	10.662 m
interval		51.966 ms	3.684 ms	0	55.173 m

Figure 6. Duration (interval\_end - interval\_start) values over time.













#### Use-cases — Our own experience

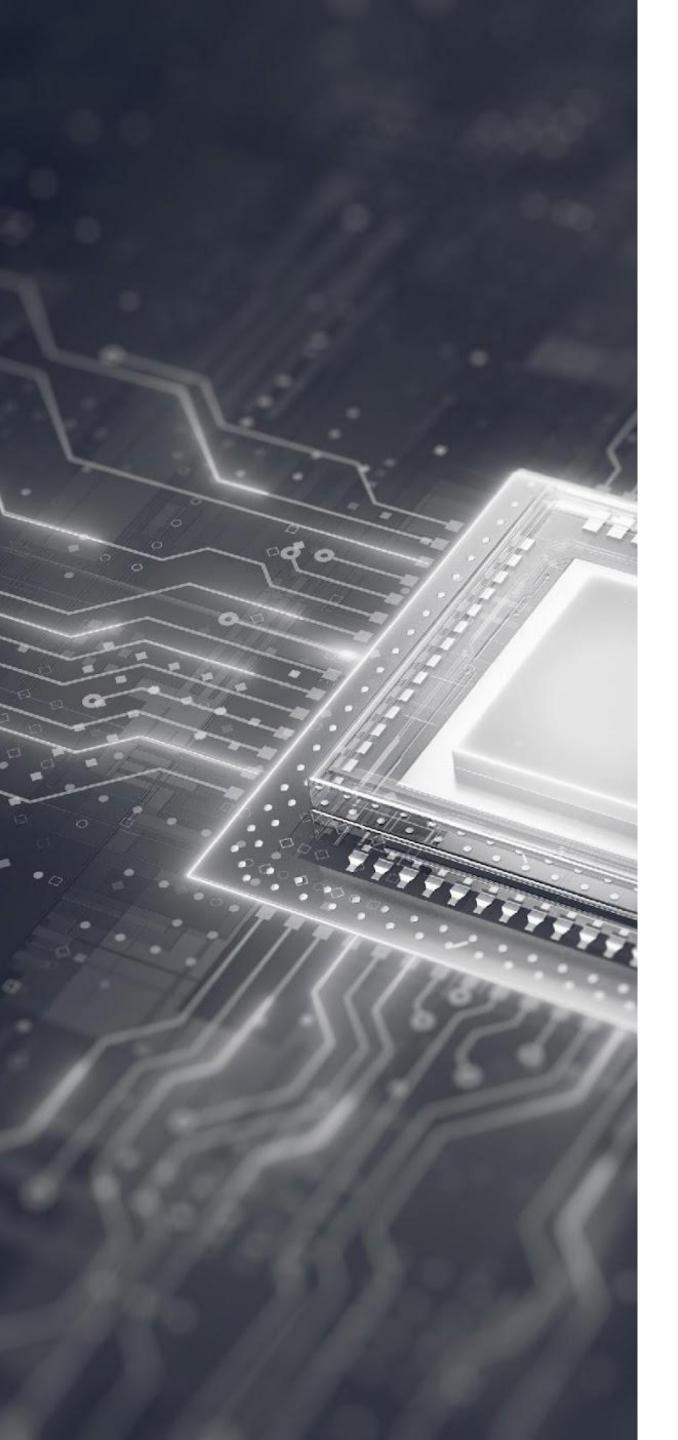
- Using tracing and our Trace Compass-based tool, we've identified and fixed multiple issues
- Executor misconfiguration
  - Very visually obvious, but would've never guessed otherwise
  - I wasn't even looking for an issue
- Performance issue
  - Could clearly see that a node couldn't process messages fast enough: bottleneck
- Performance instability due to bad execution logic
  - Would've been hard to find otherwise
- Execution strategy optimization for our LiDAR stack
  - Compare different execution strategies both visually and using KPIs
  - Make changes and optimize
  - Useful for system integrators, not only for core application developers

Being able to visualize the execution of your system is very powerful, even if you're not really looking for issues.









# Conclusion

- ROS 2 has built-in tracing instrumentation
- - Could use a bit more love, though!
- Implement your own Trace Compass plugin



• Tracing can help understand the execution of an application • Even if you're not looking for performance issues! • Collect trace data from multiple sources and analyze the combined data • Eclipse Trace Compass can display ROS 2 trace data Add custom application-specific instrumentation



# Thank you!

#### christophe.bedard@apex.ai

#### Links

- github.com/ros2/ros2 tracing
- tracecompass.org
- <u>github.com/christophebedard/ros2-message-flow-analysis</u>
- github.com/christophebedard

#### **Relevant papers**

- C. Bédard, I. Lütkebohle, and M. Dagenais, "ros2\_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2," IEEE Robotics and Automation Letters, vol. 7, no. 3, pp. 6511–6518, 2022.
- C. Bédard, P.-Y. Lajoie, G. Beltrame, and M. Dagenais, "Message Flow Analysis with Complex Causal Links for Distributed ROS 2 Systems," Robotics and Autonomous Systems, vol. 161, p. 104361, 2023.

© 2023 Apex.Al, Inc. | All rights reserved. Apex.AI, Apex.OS, Apex.Grace, Apex.Ida, Apex.Alan, Apex.Ray are trademarks of Apex.AI.

