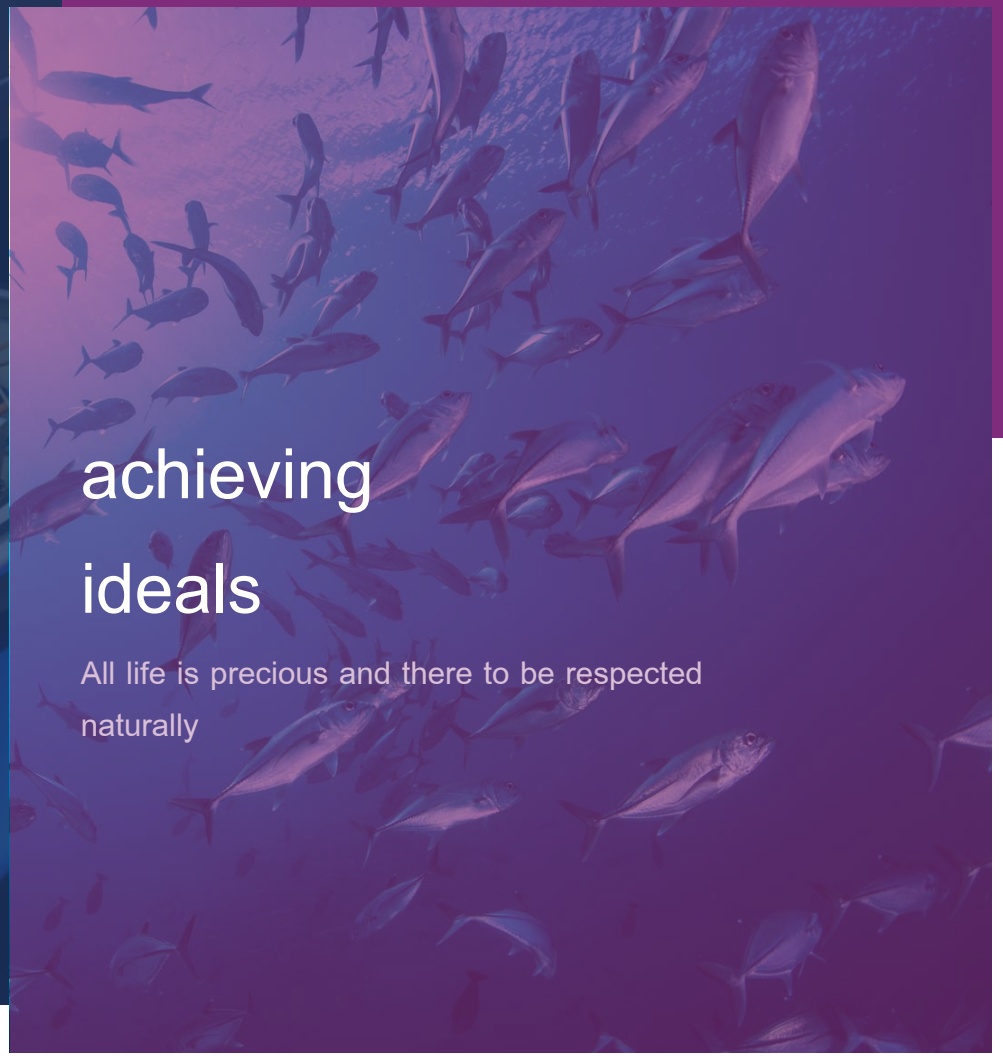
NOBLEO TECHNOLOGY

autonomous
intelligent
systems

creating ideas

Fully autonomous WasteShark to protect our marine life

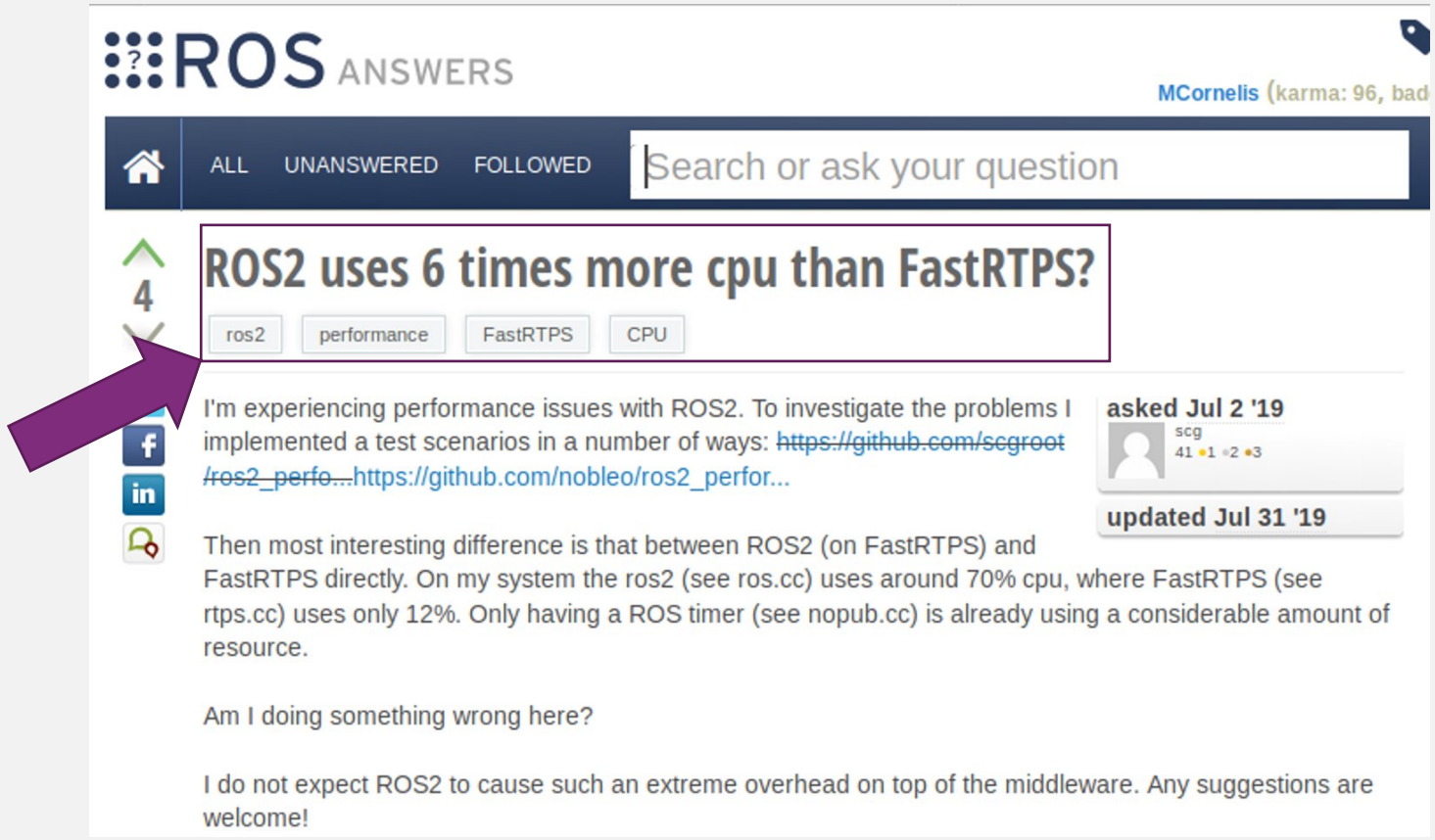


achieving ideals

All life is precious and there to be respected naturally

ROS 2 Static Executor

Problem Statement



Recreating the issue & finding the culprit(s)

	Executors	Pubs	Subs	ROS	ROS nodes	ROS timers	DDS participants	CPU %	Mem Usage	PIDs
ROS 2	1	20	200	yes	10	10	10	105	90 MiB	43
ROS 2 (1 node)	1	20	200	yes	1	1	1	45	24 Mib	7
FastRTPS	NA	20	200	no	0	0	1	4	20 Mib	6

ROS 2 vs ROS 2 (1 node):

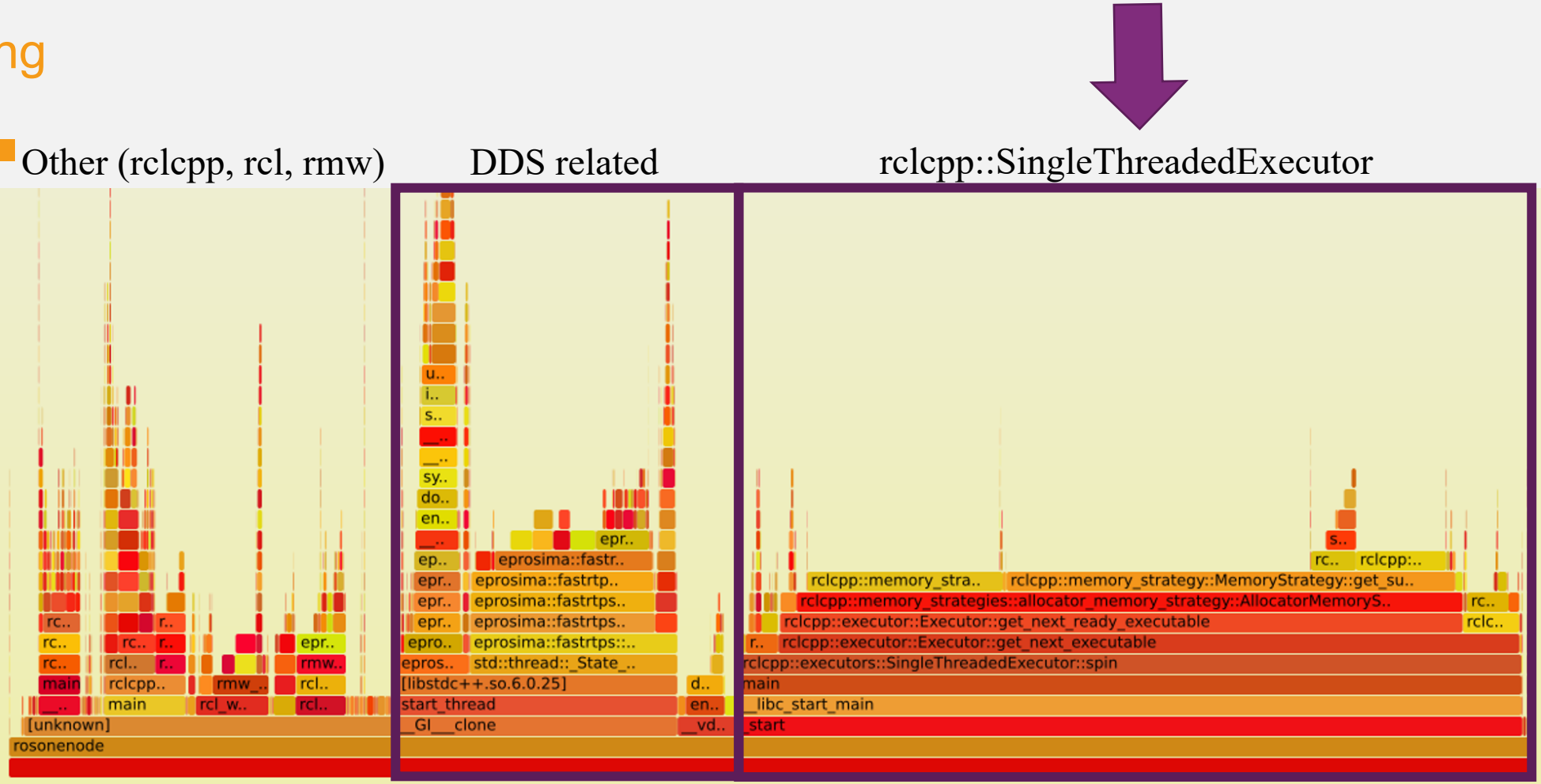
- 1-to-1 mapping of Nodes to DDS participants (major)
- extra nodes + timers = extra ROS 2 overhead (minor)

ROS 2 (1 node) vs FastRTPS:

- ROS 2 overhead (rmw-layer and up)

Tracing

Other (rclcpp, rcl, rmw) DDS related rclcpp::SingleThreadedExecutor



Flamegraph image of ROS 2 (1 node), generated with perf

Proposed optimization of executor

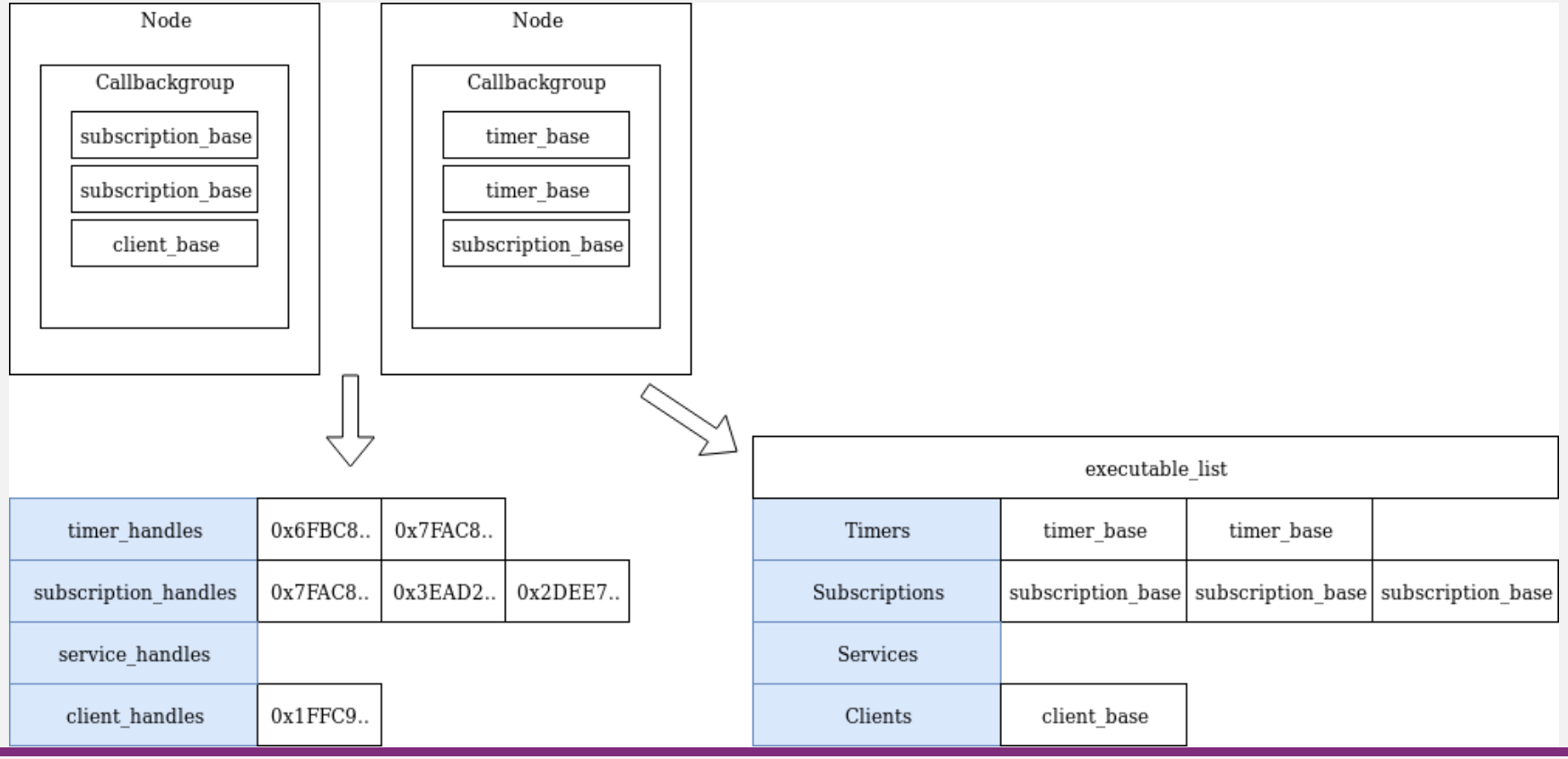
Static assumption

- allows:
  - Collect entities **once**
  - Map wait-set to struct of executables
  - Build wait-set **once**
- disallows:
  - Adding things after spin()

Rebuild on trigger (node guard_condition)

- allows:
  - Adding things after spin()

Static Executor - Implementation



Static Executor - Implementation

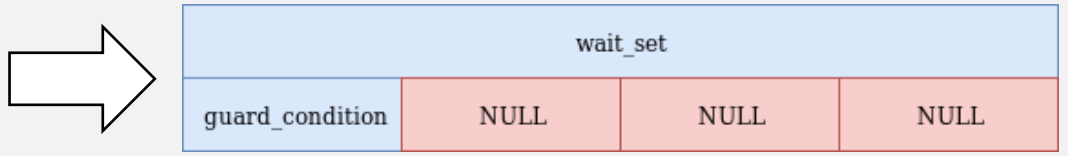
rcl_wait()				
wait_set				
Timers	0x6FBC8..	NULL		2
Subscribers	0x7FAC8..	NULL	NULL	3
Services				0
Clients	NULL			1
Waitable				0

executable_list			
Timers	timer_base	timer_base	
Subscriptions	subscription_base	subscription_base	subscription_base
Services			
Clients	client_base		

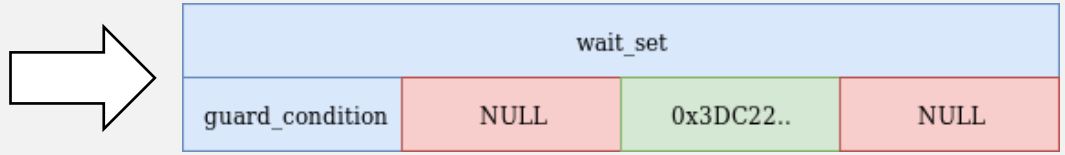
Static Executor - Implementation

During this process check node guard_conditions

Example: system remains static



Example: subscription is added to node2



When static executor is updated

- Run collect entities again
- Rebuild wait-set
- Create executable_list again

Comparison

	Executors	Pubs	Subs	ROS	ROS nodes	ROS timers	DDS participants	CPU %	Mem Usage	PIDs
ROS 2	1	20	200	yes	10	10	10	105	90 MiB	43
ROS 2 (1 node)	1	20	200	yes	1	1	1	45	24 MiB	7
ROS 2 (1 node)	**1 (static)**	**20**	**200**	**yes**	**1**	**1**	**1**	**8**	**24 MiB**	**7**
FastRTPS	NA	20	200	no	0	0	1	4	20 MiB	6

Conclusion:

If both **participant mapping** and the **executor** are addressed, the overhead of ROS 2 reduces to acceptable levels

Improvements to ROS 2

Executor:

- Static / Semi-dynamic version
  - SingleThreaded (under review!) Nobleo
  - MultiThreaded (currently out of scope)
- Improved intra-process communication (merged!) Alberto "Alsora" Soragna/Community
- Scheduling: FiFo / deadline / priority / callback-group (example package!) Bosch
- RT safe (needs to be looked at)

RMW

- 1-to-1 mapping of DDS participants fix (design doc in the making!) Community
- Improve RMW API (in progress!) Bosch

Status

Contributor(s)

Thank you

Links

Nobleo:

Participant mapping

- Discourse: https://discourse.ros.org/t/reconsidering-1-to-1-mapping-of-ros-nodes-to-dds-participants/10062
- Design Document: https://github.com/ros2/design/pull/250

Intra-process

- Fix: https://github.com/ros2/rclcpp/pull/778

Static executor

- Original ROS answers post: https://answers.ros.org/question/327477/ros2-uses-6-times-more-cpu-than-fastrtps/
- Discourse: https://discourse.ros.org/t/singlethreadedexecutor-creates-a-high-cpu-overhead-in-ros-2/10077
- Original research: https://github.com/nobleo/ros2_performance
- Pull Request (rclcpp master fork version): https://github.com/ros2/rclcpp/pull/873
- Static Executor explanation (rclcpp dashing version + detailed README): https://github.com/nobleo/rclcpp-static-executor

Links

Bosch:

Meta-executor: https://github.com/boschresearch/ros2_examples

rmw Design doc: https://github.com/ros2/design/issues/259