



WORKSHOP

**ROS 2 Executor: How to make
it efficient, real-time and
deterministic?**

October 2021

The background is a light blue gradient with faint, white line-art illustrations. On the left, a person is sitting at a desk with a laptop. In the center, there are two hands, one palm up and one palm down. On the right, there is a robot-like figure. The overall theme is a workshop or collaborative work environment.

Why have a workshop about executors?



Hasn't everything already been said?

- Single-threaded executor
- Multi-threaded executor
- Static single-threaded executor ([Nobleo's talk at ROSCON 2019](#))
- Callback-group-level executor (extension)
- RCLC executor
- We had [Ingo's talk at ROSCon 2019](#) and [Ralph's talk at ROS-Industrial 2020](#)



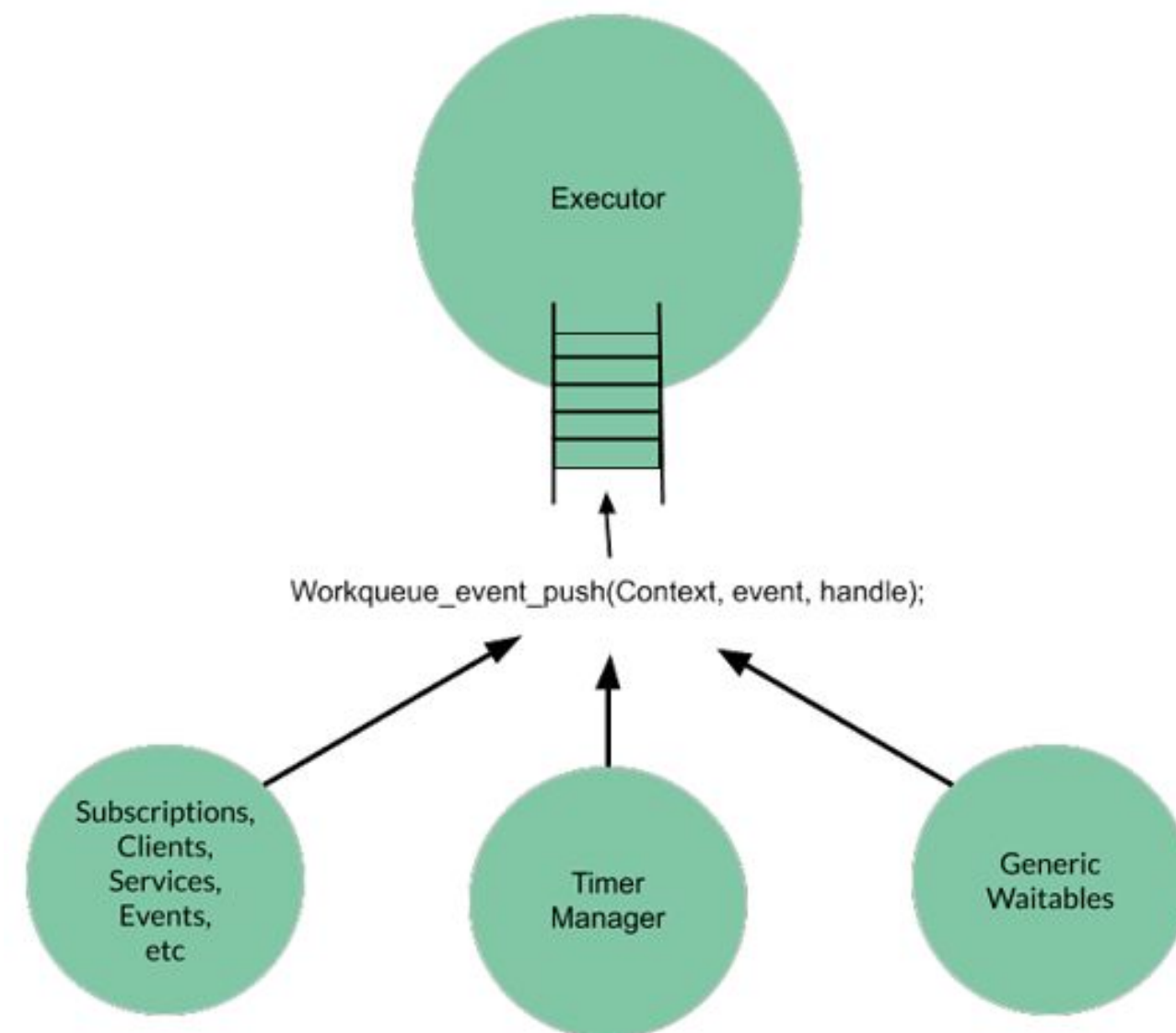


More to come?

<https://discourse.ros.org/t/ros2-middleware-change-proposal/15863>

As I mentioned in the last middleware working group meeting, the iRobot team would like to propose a change to how ROS2 handles incoming events.

We believe that rather than using user level waitsets, that an executor event queue design will allow events to propagate faster. When the executor thread is waiting on events to arrive, it simply blocks allowing the CPU to perform other work. When awakened, the events are processed in the order they are received. Each event contains the a type enumeration and a unique handle to process the event.



Hi Michael and Dejan,

As we progress with the development of our own ROS2 executor, I would like to talk to you 15-30 mins about the current executors and APEX's view on it? There are many approaches at the moment and we would like to get your opinion on which API to follow, at least initially.

Hallo Michael,

you might want to invite the authors of this paper to present their approach at the Executor Workshop as well:

- PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS 2:
- https://intra.ece.ucr.edu/~hyoseung/pdf/rtas21_picas.pdf

Apex.OS 1.3 Release

and real-time manner. To achieve that, Apex.OS introduces a deterministic executor with this release to abstract the complex scheduling policies of various Linux / RTOS flavors and simplifies the development of a safety certifiable software stack.

- Still a lot of people are working on improving performance, determinism and real-time capabilities
- This shows that the executor is one of the most crucial components for ROS 2 in product use





Motivation

- Quo vadis ROS 2 executor?
 - At ROSCon 2019 we had 2 executors and 2 new were introduced
 - Today, at ROSWorld 2021, we have 4 executors and 4 new are introduced
 - Should we avoid having 8 executors at ROSCon 2023 and introducing 8 more?
- Goals for the workshop
 - Understand how crucial the executor is for a ROS 2 system
 - Get an overview of the existing executors and their capabilities
 - Introduce an infrastructure that enables executor approaches to be examined / compared
 - Trigger the discussion in the community
 - What are the pain points, what are the needed features?
 - Is a consolidate to 2-3 execution models possible?
 - Will this topic be addressed in an existing working group or a new one?





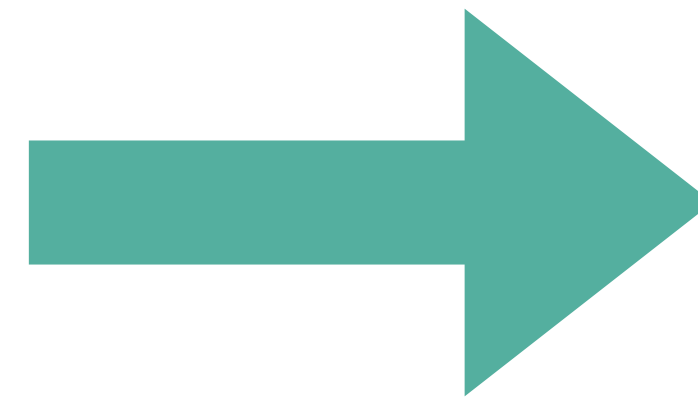
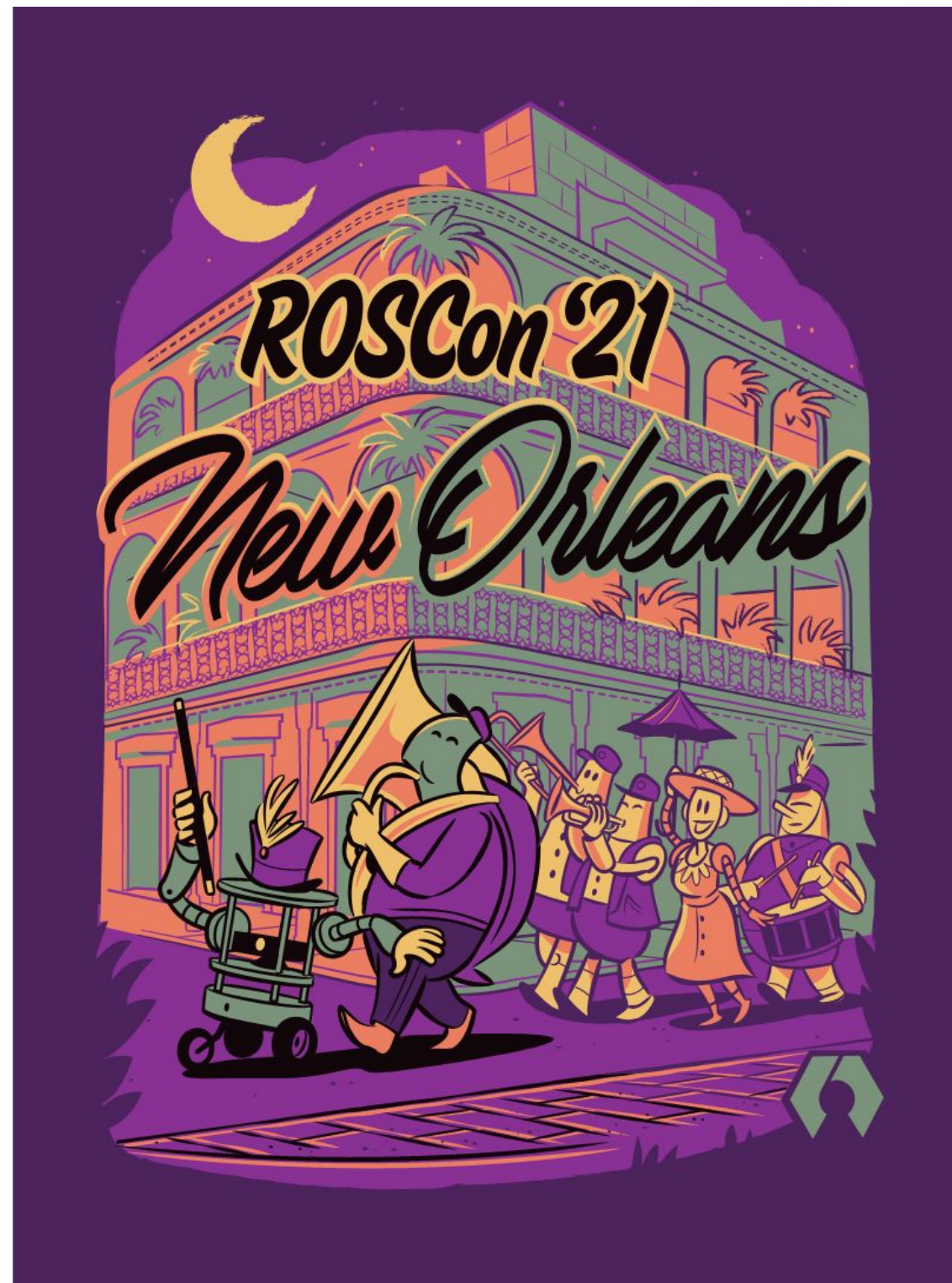
Planned Agenda

- Full day in person workshop
- Switching between talks and hands-on sessions
- Step-by-step improving a system running on a Raspberry Pi
 - Using different ROS 2 executors
 - Tweaking the underlying system for real-time
 - Hands-on introduction to the tools for analyzing the system





Plans change





Agenda Part I

- Current status of executor in ROS 2 Galactic (William)
- Introduction to the reference system (Evan)
- Analyzing the reference system (Christophe)
- Tune the system for real-time (Andrei)
- Callback-group-level executor (Ralph)
- Events executor (Alberto)





Agenda Part II

- PiCAS executor (Hyunjong)
- RCLC executor (Jan)
- Executor with wait-set and polling subscription (Michael)
- Lock-free ROS 2 executor: A ring-buffer to rule them all (Pablo)
- Panel discussion (all)





Q&A

- Please use the Q&A box for questions
- After each talk 2 questions will be answered live
- The other questions will be answered in the chat after the talk
- If you already have questions for the panel discussion, start the question with “Panel:”



The background features a light blue gradient with faint, white line-art illustrations. On the left, a person is shown from the chest up, wearing a hoodie and working on a laptop. In the center, a globe is depicted with latitude and longitude lines. On the right, there is a coffee cup on a saucer. The overall theme is productivity and taking a break.

**Have a break -
10 min**

The background features a light blue gradient with faint, white line-art illustrations. A large globe is centered in the upper half. To the left, a person is shown sitting at a desk with a laptop, their hand raised as if speaking. To the right, another person is shown from the chest up, also with a hand raised. The overall theme is global communication and discussion.

Panel Discussion



Executor Overview

Executor	Presented by	Characteristics	Availability
Single-threaded	William	<ul style="list-style-type: none">- Based on wait-set- Dynamic wait-set reconfiguration	Always
Multi-threaded	William	<ul style="list-style-type: none">- Based on wait-set- Using multiple threads	Always
Static single-threaded	William	<ul style="list-style-type: none">- Static wait-set configuration for lower CPU usage	since Foxy
Callback-group-level	Ralph	<ul style="list-style-type: none">- Callback groups of a node can be assigned to different executors- Available for all executors that are based on the rclcpp base class	since Galactic
Events	Alberto	<ul style="list-style-type: none">- Listener API as alternative to the wait-set- Event queue with chronological order	planned for Humble
PiCAS	Hyunjong	<ul style="list-style-type: none">- Priority-driven chain-aware scheduling- Resource allocation algorithms and timing analysis	On fork only
RCLC	Jan	<ul style="list-style-type: none">- User-defined processing sequence for callbacks- Custom trigger conditions, static memory implementation	C Client Library
Wait-set + polling subscription	Michael	<ul style="list-style-type: none">- Executing node callbacks with optional execution conditions- Optimized wait-set usage with polling subscriptions	On private fork only
Ring-buffer	Pablo	<ul style="list-style-type: none">- Lock-free ring buffer and single threaded executors- Optimised for large throughput and low CPU usage	On private fork only

