# Executors in ROS 2

Michael Carroll

**Software Engineer @ Intrinsic**

mjcarroll@intrinsic.ai

**@mjcarroll** on all the things

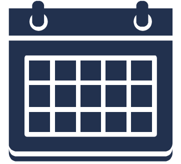
William Woodall

**Software Engineer @ Intrinsic**

wjwwood@intrinsic.ai

**@wjwwood** on all the things

# Outline



- Conceptual Overview of an Executor in ROS 2
- Types of Executors in ROS 2
- Performance Improvements Due to Recent Work
- Picking an Executor for your Application

ROS™

# What are “Entities”

**Primitive Entities**

- Timers
- Guard Conditions
- Subscriptions (inter-process)
- Service Servers/ Clients

**Derived Entities**

- Waitables
- QoS Events
- Intra-process Subscriptions
- Action Servers/ Clients
- etc…

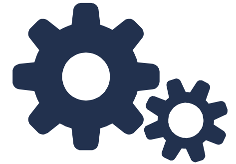
ROS™

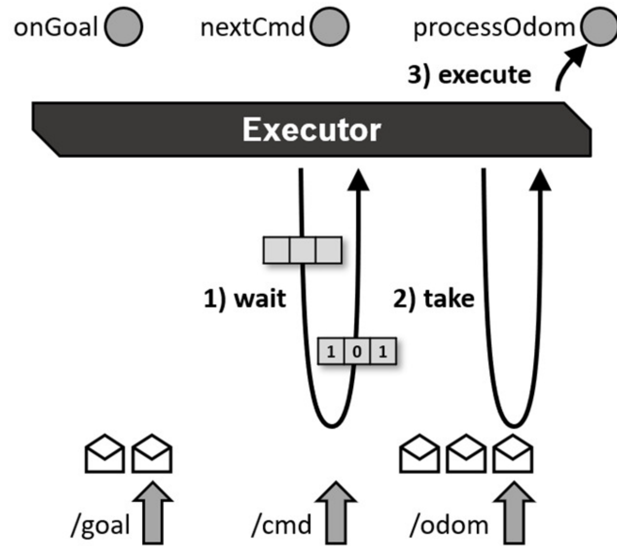
# Executor Responsibilities

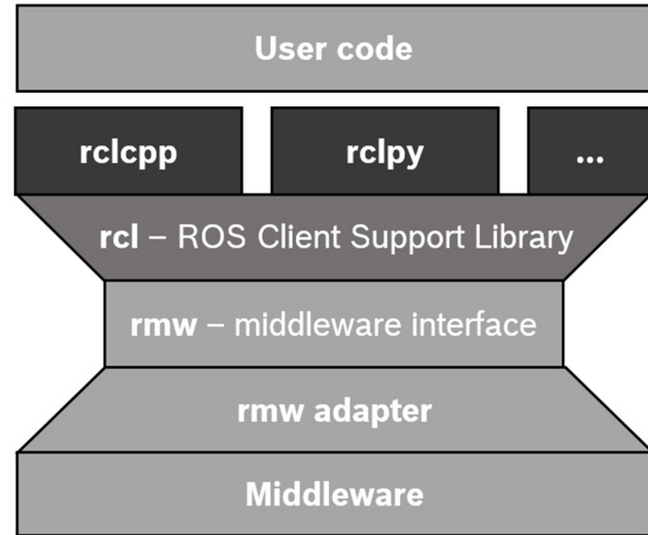
1. Collect entities that should be waited on, and manage their ownership
2. Wait for one or more things to be ready
3. Decide what to execute next (scheduling)
4. Dispatching the entities' task for execution

ROS™

# What does an Executor do






source: https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Executors.html
credit: @ralph-lange, others

ROS™

# More about Waitables

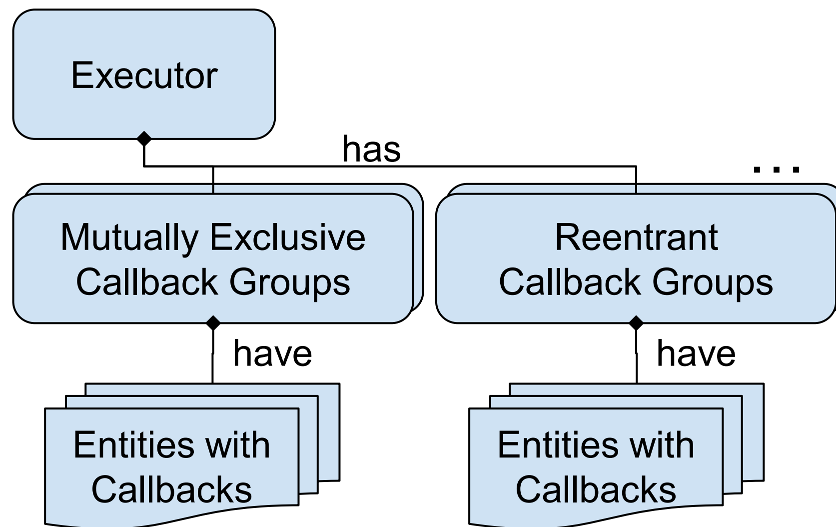
- You can write your own!
- **Waitables** are the base of other complex entities
- **Guard Conditions** can be used to interact with other event systems
- **Waitables** can be used as a **Guard Condition** + **data**

```
class MyWaitable : public rclcpp::Waitable
{
public:
  // ...
  bool
  is_ready(const rcl_wait_set_t & wait_set) override
  { /* ... */ }

  void
  set_on_ready_callback(
    std::function<void(size_t, int)> callback) override
  { /* ... */ }

  std::shared_ptr<void>
  take_data() override
  { /* ... */ }

  void
  execute(const std::shared_ptr<void> & data) override
  { /* ... */ }
  // ...
};
```

ROS™

# Callback Groups

- **Group**ing of Entities with **Callback**s
- Informs the Executor about what can be executed and when
- Designed to solve multi-threading issues




ROS™

# Collecting Entities to Wait

- **Executor** operates on **Callback Groups**, not **Nodes**
- Executor has *weak references* to Callback Groups, which have *weak references* to Entities, until you wait…
- Executor will also extend the *lifetime* of Entities while executing

**Waiting is interrupted when…**

- Executor is explicitly interrupted
  - e.g. ctrl-c, executor.cancel(), ROS shutdown, etc.
- Entities are added to, or removed from, a Callback Group
- Callback Groups are added to, or removed from, an Executor
- Or when one or more of the Entities are ready

ROS™

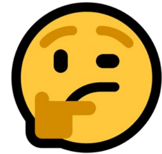
# Waiting

Multiple approaches:

- WaitSet
  - Collect “handles” to entities, pass them to rmw to block
- Callbacks
  - Set up rmw to call our function when the entity is ready

ROS™

# Why not <event library>?

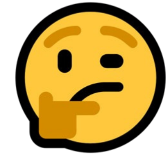


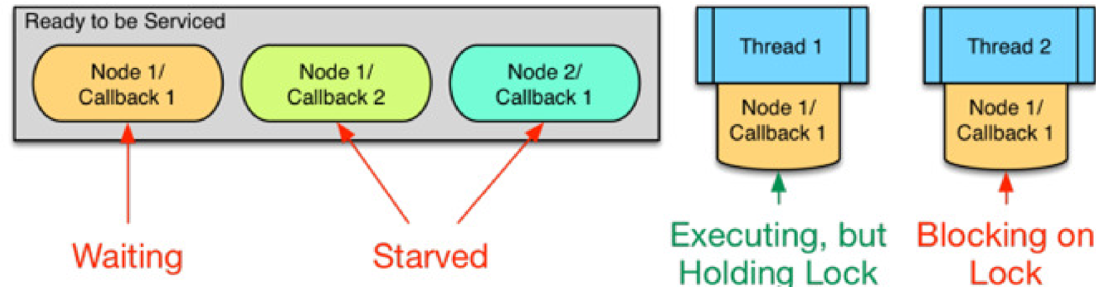
The executor is just yet another implementation of the *proactor* design pattern just like libasio, libuv, etc., but:

- Waiting on the middleware isn't easy to do, unless you …
- Write your own "io loop/ context" for the event libraries, but this isn't easy, it's the hard part where all the dragons are
- Hard to replace callback groups with existing concepts in the event libraries, e.g. *strands* in libasio are similar but insufficient

ROS™

# Why not <event library>?



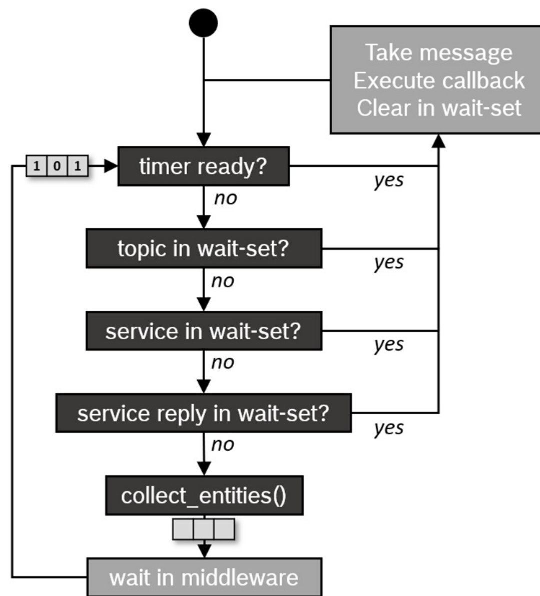



source: https://roscon.ros.org/2014/wp-content/uploads/2014/07/ROSCON-2014-Why-you-want-to-use-ROS-2.pdf

2014 ROS™

# Deciding what to Execute

- Existing executors have existing scheduling algorithms
- You might want your own, executor is intended to be extended (work in progress)
- Custom executors allow for different scheduling and things like batching

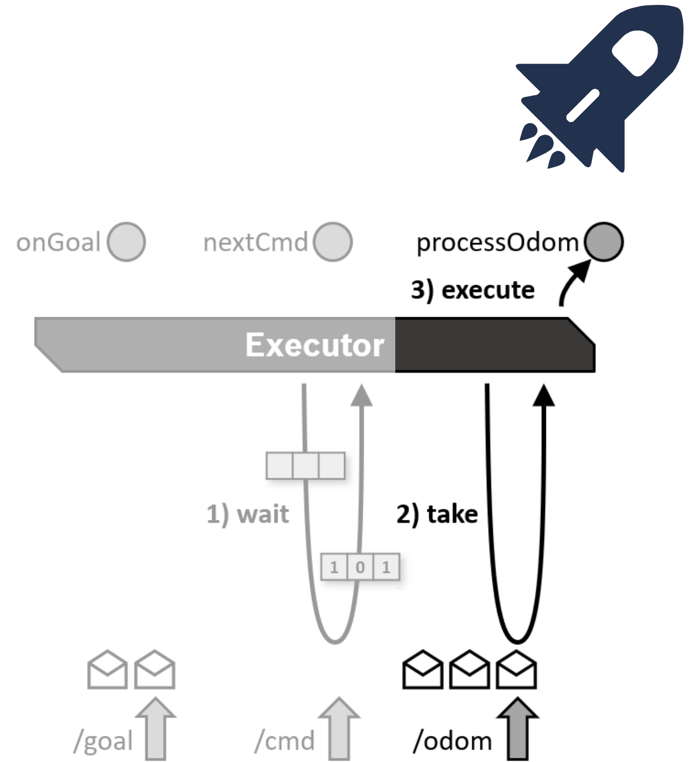



source: https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Executors.html#scheduling-semantics

ROS™

# Execution

1. Once the entity is ready, and the executor is not busy…
2. Take any data associated with the entity for the callback
3. Execute the user-defined callback, passing the taken data



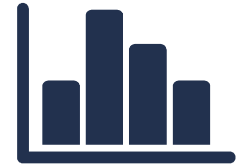

ROS™

# Types of Executors (cpp)

| | Wait Mechanism | Execution | Timers |
|---|---|---|---|
| **SingleThreadedExecutor** | WaitSet | Single Thread | in wait loop |
| **MultiThreadedExecutor** | WaitSet | Multi-Threaded | in wait loop |
| **EventsExecutor** | callbacks | Single Thread | threaded TimerManager |

ROS™

# Events Executor



SingleThreadedExecutor | MultiThreadedExecutor | StaticSingleThreadedExecutor | EventsExecutor

| | SingleThreadedExecutor | MultiThreadedExecutor | StaticSingleThreadedExecutor | EventsExecutor |
|---|---|---|---|---|
| CPU [%] | 12.3 | 20.7 | 13.1 | 2.8 |
| Average Latency [us] | 100 | 559 | 122 | 21 |

source: https://discourse.ros.org/t/the-ros-2-c-executors/38296
credit: Alberto Soragna (@alsora)

~June 2022

" pseudo-random system with **~40 publishers** and **~70 subscriptions** spread across **8 executors**

# Recent Improvements

- Unified SingleThreadedExecutor and StaticSingleThreadedExecutors
- Implemented rclcpp::WaitSet
- Improved performance of "Entity Collection" with double buffering

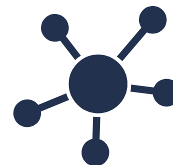
ROS™

# Picking an Executor

- Consider your workload… do you prioritize:
  - Low Latency
  - High Throughput
  - Deterministic Execution
- No "silver bullet"
- EventsExecutor for responsive and performant option
- MultiThreadedExecutor to utilize multi-core systems
- Use multiple executors if needed

ROS™

# Impact of Callback Groups



- **MutuallyExclusiveCallbackGroup**'s cause a lot of work in entity collection when used with the multi-threaded executors
  - Avoid this by using a **ReentrantCallbackGroup** if appropriate or multiple callback groups if not
- The default Callback Group (if you don't specify one) is a single **MutuallyExclusiveCallbackGroup** in order to keep with ROS 1's behavior and for a safe default

ROS™

# Alternative: Use WaitSet

- The introduction of rclcpp::WaitSet grew out of a request to not use Executors at all and handle waiting and execution entirely within user code.
- See examples:
  https://github.com/ros2/examples/blob/rolling/rclcpp/topics/minimal_subscriber/static_wait_set_subscriber.cpp

ROS™

# Thanks!

## Special thanks to…

Janosch Machowinski
**@jmachowinski** on GitHub
@ Cellumation GmbH
Has a talk right after this one!

Alberto Soragna
**@alsora** on GitHub
@ iRobot

## and…