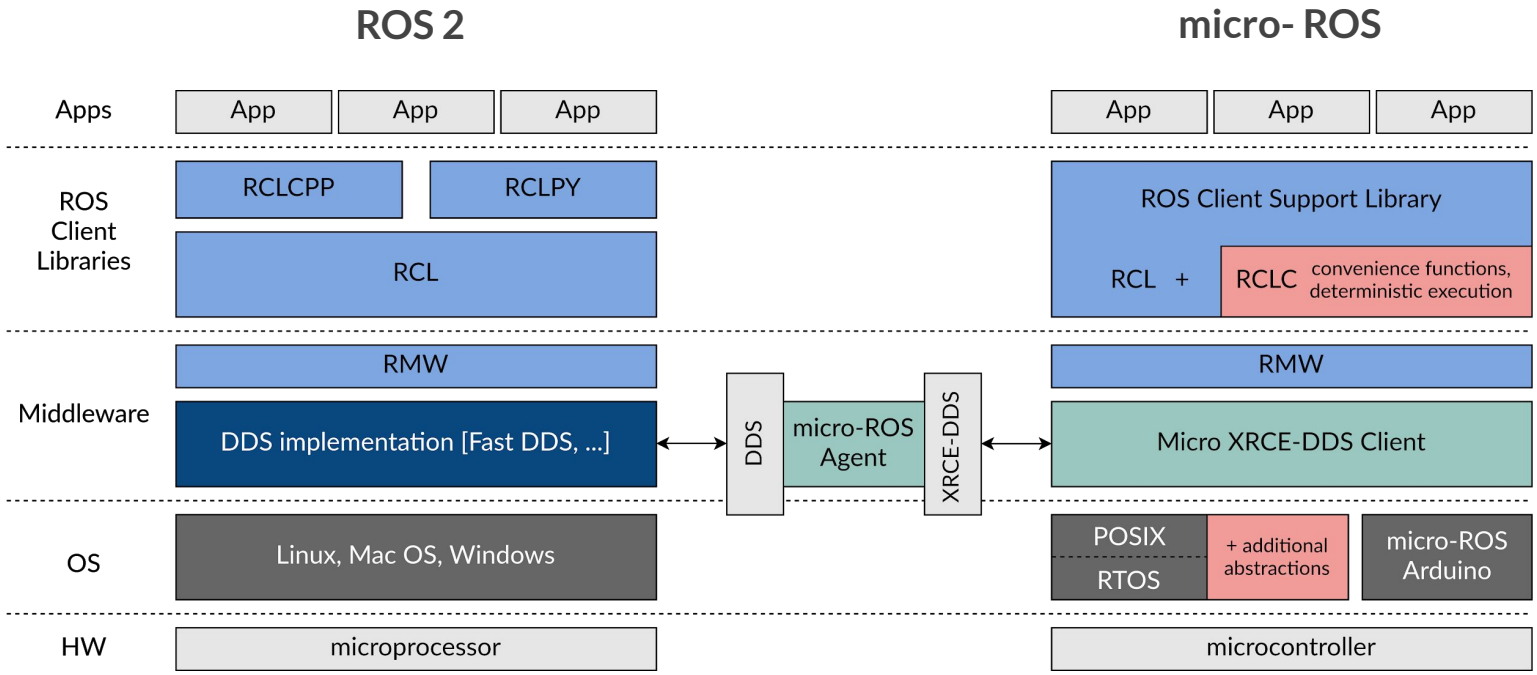
EUROPEAN
ROBOTICS FORUM

# micro-ROS: API and Executor

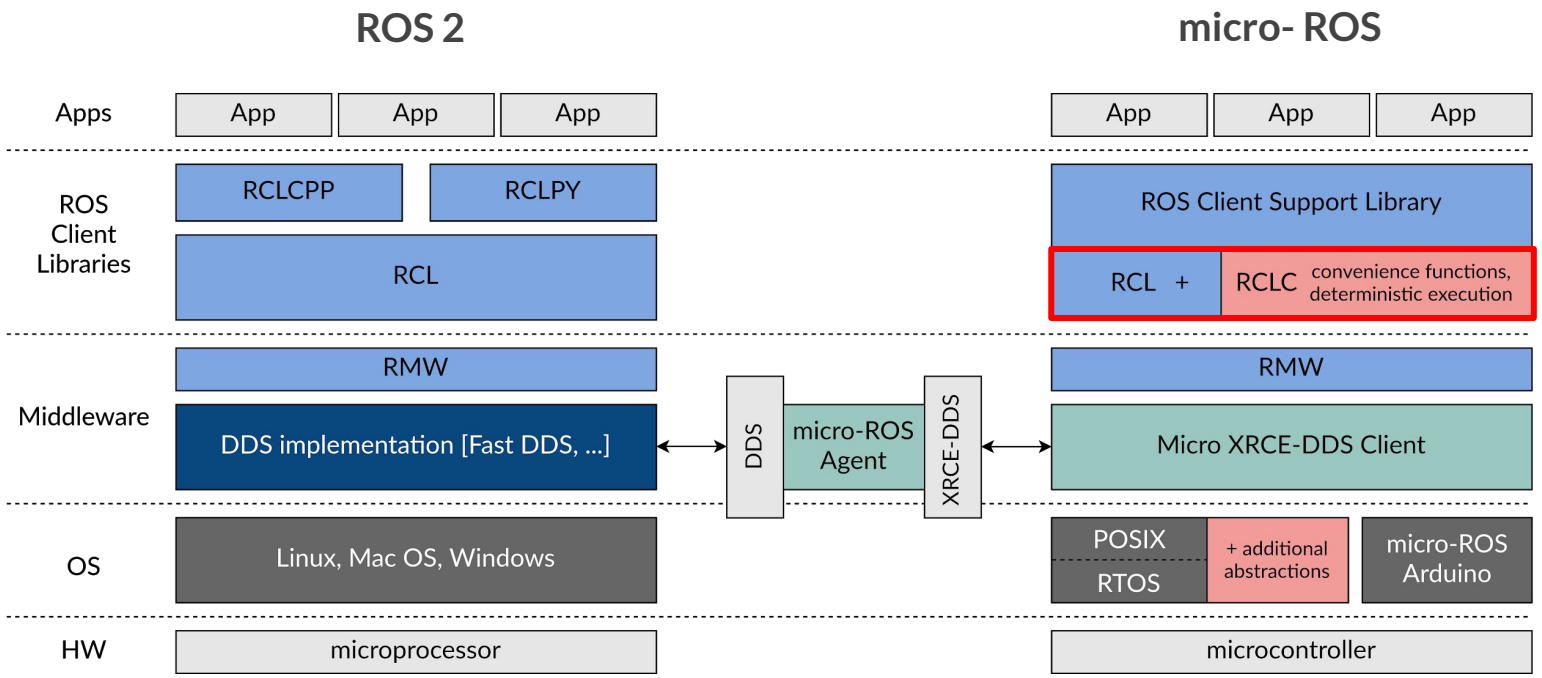
Jan Staschulat - Bosch

April 13th, 2021

# micro-ROS architecture

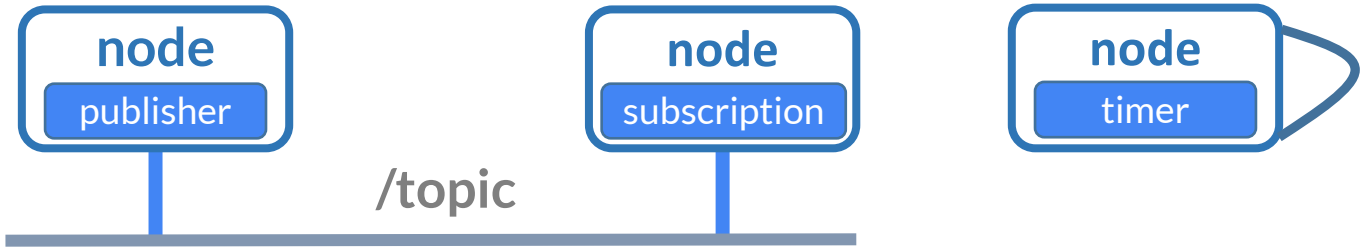


# micro-ROS architecture



# ROS 2 : basic concepts

***Pub-sub communication***




***Executor***


- Checks for new messages
- Executes corresponding callbacks

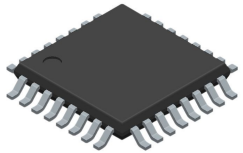
# Why an RCLC API?

## *ROS 2 – RCLCPP drawbacks*

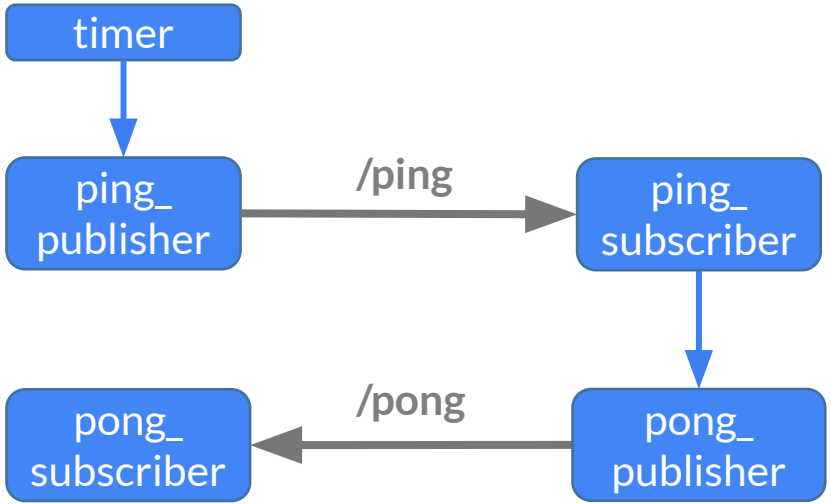
- API in C++ uses dynamic memory allocation
- Executor is not deterministic nor does it  support real-time

## *Micro-ROS – RCLC benefits*

- Thin layer on top of RCL (no additional data structures) feature-complete (publishers, subscriptions, timers, services/clients, guard conditions)
- Executor uses dynamic memory allocation only at startup
- Deterministic Executor with additional features to support real-time applications



# ROS 2 : API example



# RCLC – API: node

```
#include <rclc/rclc.h>
#include <rclc/executor.h>

void main()
{
...
rcl_allocator_t allocator = rcl_get_default_allocator();
rclc_support_t support;
rclc_support_init(&support, 0, NULL, &allocator);

rclc_node_init_default(&node, "pingpong_node", "", &support);
```

# RCLC – API: timer

```
rcl_timer_t timer = rcl_get_zero_initialized_timer();

rclc_timer_init_default(&timer,
      &support,
      RCL_MS_TO_NS(2000),
      ping_timer_callback);
```

# RCLC – API: timer-cb

```
void ping_timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
      (void) last_call_time;
      if (timer != NULL) {
          seq_no = rand();
          sprintf(outcoming_ping.frame_id.data, "%d_%d", seq_no, device_id);
          outcoming_ping.frame_id.size = strlen(outcoming_ping.frame_id.data);
          struct timespec ts;
          clock_gettime(CLOCK_REALTIME, &ts);
          outcoming_ping.stamp.sec = ts.tv_sec;
          outcoming_ping.stamp.nanosec = ts.tv_nsec;
          pong_count = 0;
          rcl_publish(&ping_publisher, (const void*)&outcoming_ping, NULL);
      }
}
```

# RCLC – API: publisher

```
rcl_publisher_t ping_publisher;

rclc_publisher_init_default(&ping_publisher,
      &node,
      ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Header),
      "/microROS/ping");
```

# RCLC – API: subscription

```
rcl_subscription_t ping_subscriber;

rclc_subscription_init_default(&ping_subscriber,
      &node,
      ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Header),
      "/microROS/ping");
```

# RCLC – API: subscription cb

```
void ping_subscription_callback(const void * msgin)
{
      const std_msgs__msg__Header * msg = (const std_msgs__msg__Header *)msgin;
      rcl_publish(&pong_publisher, (const void*)msg, NULL);
      }
}
```

# RCLC – API: executor

```
rclc_executor_t executor = rclc_executor_get_zero_initialized_executor();
rclc_executor_init(&executor, &support.context, 3, &allocator));


rclc_executor_add_timer(&executor, &timer));
rclc_executor_add_subscription(&executor, &ping_subscriber, &incoming_ping,
&ping_subscription_callback, ON_NEW_DATA));

rclc_executor_spin(&executor);
```

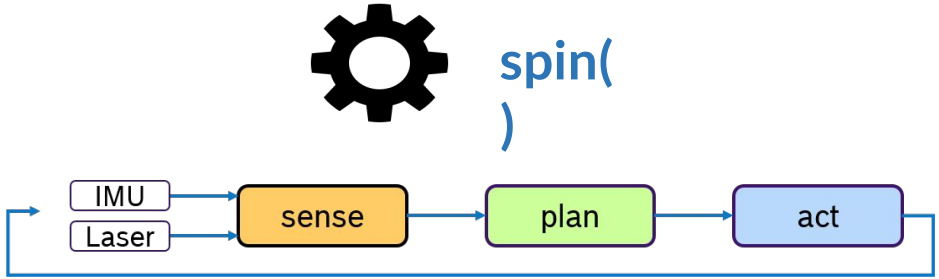
# Deterministic executor

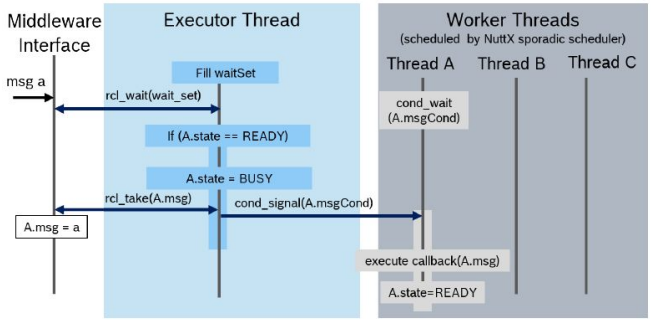
## *Deterministic behavior*

- User-defined order of callback processing
- Priorization possible

## *Additional features for real-time*

- Trigger condition to support domain specific-scheduling (and/or activation semantics)
  => Sense-plan act control loops
  =>Synchronization of messages (sensor fusion)
- Real-time scheduling by using RTOS priority-based scheduling (priorities for threads) in Executor (proof-of-concept)

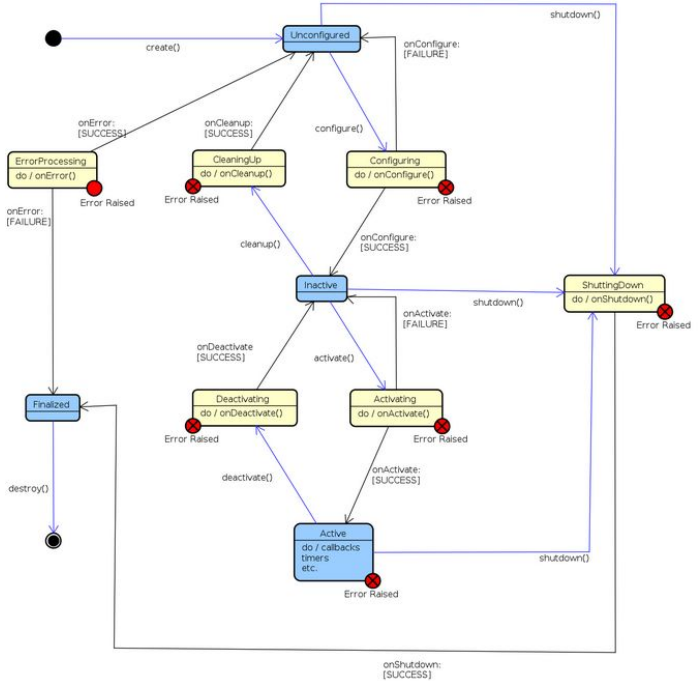


# RCLC Lifecycle

- Convenience function for **ROS 2 Lifecycle Node with rclc**
- rclc lifecycle node bundles an rcl Node and the ROS 2 lifecycle state machine

- Greater control over the state of ROS system
  - ROS 2 standard node life cycle
  - configure, activate, deactivate, cleanup, …
  - integrated with launch, e.g., ensure all components active before any component begins executing its behavior
- Previously only available for C++ (rclcpp_lifecycle)
- Now available for C (rclc):
- Builds upon rcl_lifecycle (as does rclcpp_lifecycle)
  - **Transitions and callbacks** implemented, working, and tested
  - **Lifecycle services** implemented, pull request **pending**
    - Under discussion: Completely avoid dynamic memory allocation.
    - Not yet possible due to strings in lifecycle messages



# RCLC Lifecycle

- Convenience function for **ROS 2 Lifecycle Node with rclc**
- rclc lifecyle node bundles an rcl Node and the ROS 2 lifecycle state machine

**Initialition:**

```
rclc_node_init_default(&my_node, "lifecycle_node", …);
rcl_lifecycle_get_zero_initialized_state_machine();
rclc_make_node_a_lifecycle_node(&lifecycle_node, &my_node, …);
```

**Transitions and Callbacks:**

```
rclc_lifecycle_register_on_configure(&lifecycle_node, &my_on_configure);
rclc_lifecycle_change_state(&lifecycle_node, …TRANSITION_CONFIGURE, …);
```

**Lifecycle services**: *(pull request pending!)*

```
rclc_lifecycle_add_get_state_service(&lifecycle_node, &executor);
rclc_lifecycle_add_get_available_states_service(&lifecycle_node, &executor);
rclc_lifecycle_add_change_state_service(&lifecycle_node, &executor);
```

# Back-up slides

# XXX