MICRO-ROS: ROS2 ON MICRO-CONTROLLERS

OFERA project
EU Grant 780785
www.ofera.eu

European Commission

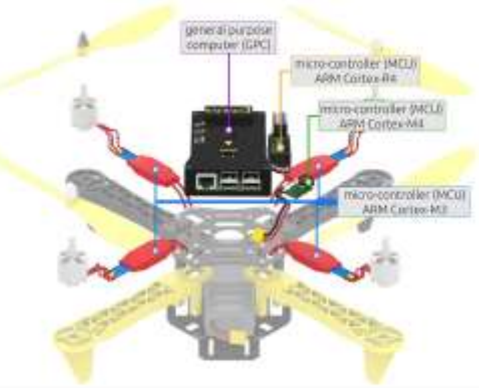
BOSCH

Micro-ROS: ROS2 on microcontrollers
Audience Check

- Disciplines:
  - Computer science
  - Electrical engineering
  - Mechanical engineering
  - Other?
- Who has used an Arduino or similar maker board?
- Who has written hardware drivers for ROS?

Micro-ROS: ROS2 on microcontrollers

Microcontrollers are everywhere


general purpose computer (GPC)
micro-controller (MCU) ARM Cortex-R4
micro-controller (MCU) ARM Cortex-M4
micro-controller (MCU) ARM Cortex-M3

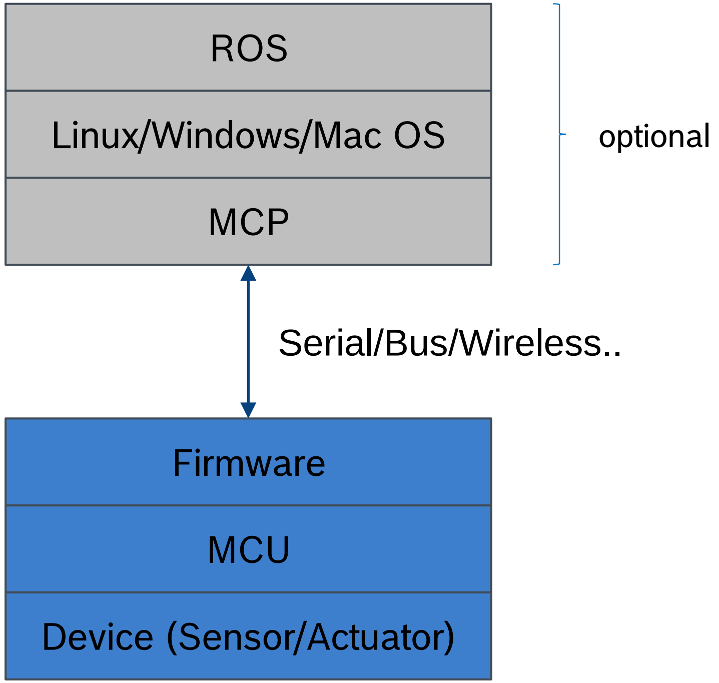
- Typical applications
  - Motor control
  - Sensor interfaces (AD, post-processing)
    - Incl. sensor fusion
  - Driving displays, LEDs, etc.
  - Low-latency real-time control
- Characteristics
  - Low power usage (up to battery operation for years)
  - Very predictable execution times
  - Hardware integration
  - Many integrated I/Os (I²C, SPI, CAN, etc.)
- Sophisticated safety-rated versions available

BUT: Development totally disconnected from ROS-based development

Micro-ROS: ROS2 on microcontrollers

Goals

Typical topology



- Run nodes on microcontroller *seamlessly*
  - Publish/subscribe/services just work
  - Parameters/lifecycle/… just work
- Take advantage of hardware features
  - Power saving
  - Hard real-time scheduling
  - Easy hardware access
- Developer Experience
  - Build using ROS tools
  - Same codebase and APIs wherever possible
- Challenges
  - Resource use (RAM, CPU, Disk)
  - Different build-systems, OS, community expectation

Micro-ROS: ROS2 on microcontrollers
Device Classes

	ROS2	Micro-ROS
Hardware	X86, ARM Cortex-A, ...	ARM Cortex-M, ...
Resources	>512 MB RAM, >8 GB Disk	>100 KB RAM, >1 MB Flash
Communications	Ethernet, 802.11 WiFi	Serial, WPAN – 250 KBit - 1 MBit/s
Operating System	Linux, Windows, MacOS	RTOS (NuttX by default)
Middleware	DDS variant (by default)	XRCE-DDS (by default)
Middleware Abstraction	RMW	RMW
Client Support Library	RCL	RCL
Execution Layer	RCLCPP / RCLPY / ...	RCL + RCLCPP
Executors	Generic	Micro-ROS custom

Micro-ROS: ROS2 on microcontrollers
Ingredients

Ease of use

- Build system integration
- Default configurations
- Default hardware
  - Looking for collaborators!
- Tutorials
- Community demos
- Slack channel
- Ready-to-use docker containers

MCU-targeted capabilities

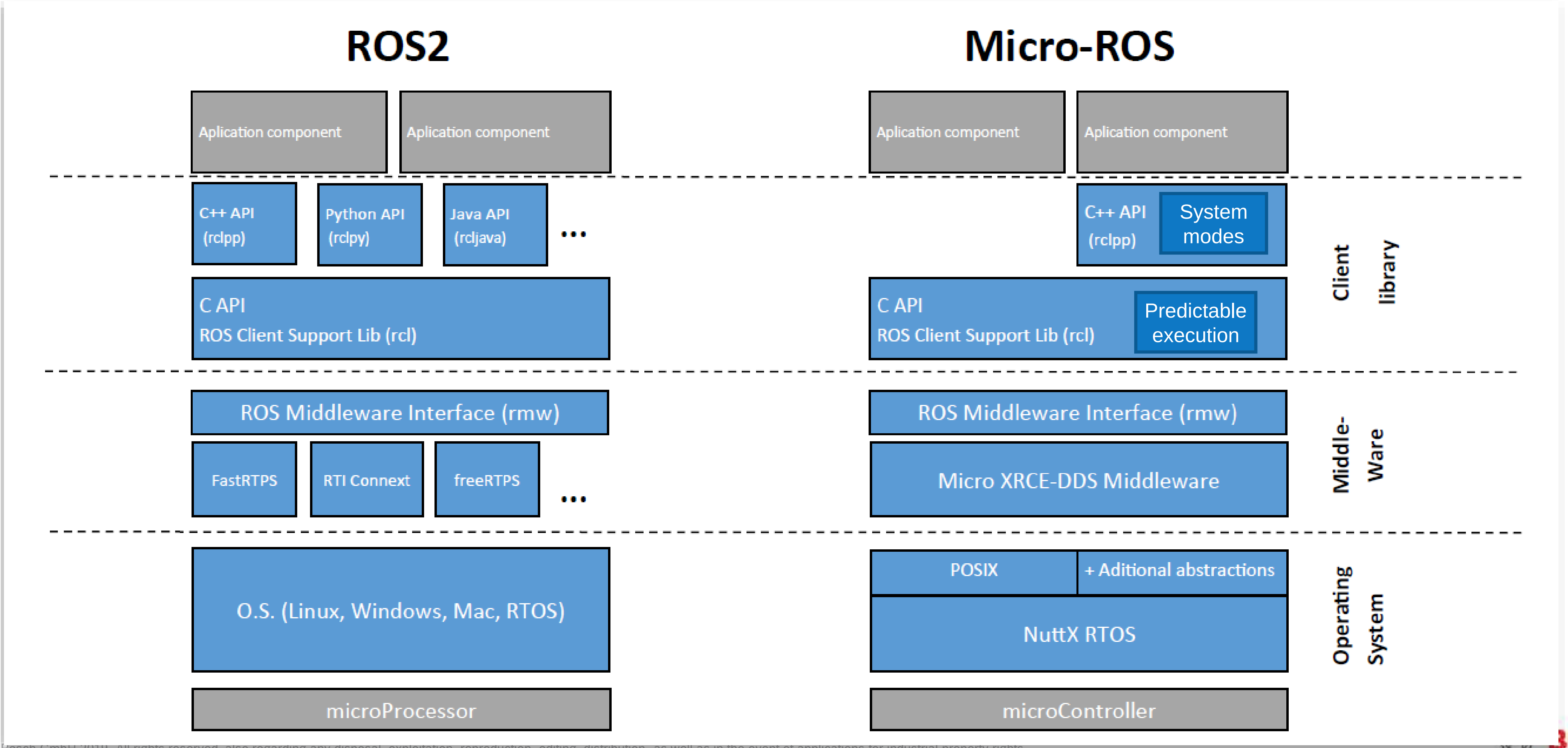
- Middleware XRCE-DDS
- Custom executors
  - E.g., static ordering
- MCU tracing and debugging
- Portability
  - Transports extensible

Performance & Predictability

- Executor performance
- Deterministic execution
- System Modes
- Benchmarking tools

micro-ROS: ROS 2 on microcontrollers

Differences between ROS 2 and Micro-ROS



ROS2

Aplication component

Aplication component

C++ API (rclpp)

Python API (rclpy)

Java API (rcljava)

…

C API
ROS Client Support Lib (rcl)

ROS Middleware Interface (rmw)

FastRTPS

RTI Connext

freeRTPS

…

O.S. (Linux, Windows, Mac, RTOS)

microProcessor

Micro-ROS

Aplication component

Aplication component

C++ API (rclpp)

System modes

C API
ROS Client Support Lib (rcl)

Predictable execution

ROS Middleware Interface (rmw)

Micro XRCE-DDS Middleware

POSIX

+ Aditional abstractions

NuttX RTOS

microController

Client library

Middle-Ware

Operating System

micro-ROS: ROS 2 on microcontrollers

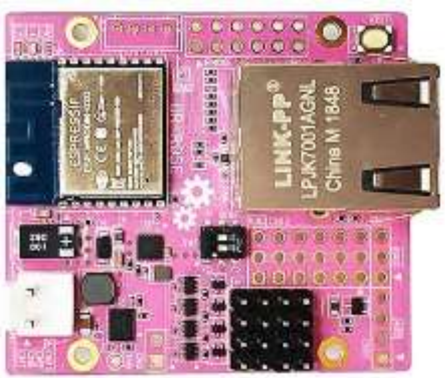
Target Devices

- Reference HW platform
  - Cortex-M4 devices with ~100KB RAM
    - Olimex STM32-E407
  - Cortex-M0 investigated but no longer pursued
- 3rd Party platforms
  - Renesas is on track to support GR-ROSE boards
  - Sony has expressed interest in supporting their SPRESENSE board
- RTOS
  - Default RTOS is NuttX
  - Intel has expressed interest in working on Zephyr support

STM32L1-DISCO

Olimex STM32-E407

GR-ROSE



micro-ROS: ROS 2 on microcontrollers
Build System: Background

- RTOS´s are complete packages including
  - Scheduling (of course)
  - Networking
  - Standard libraries (libc, libm, libstdc++) etc
  - Tools, and many more things
- RTOS´s are highly configurable
  - Most things are turned off by default to save resources
  - Every change can affect system headers

→RTOS‘s have relatively sophisticated, diverse and *complex* build systems

- Microcontrollers often build operating system and application into a single firmware image
  - This includes all dependencies, e.g. ROS 2
  - Everything is cross-compiled

micro-ROS: ROS 2 on microcontrollers

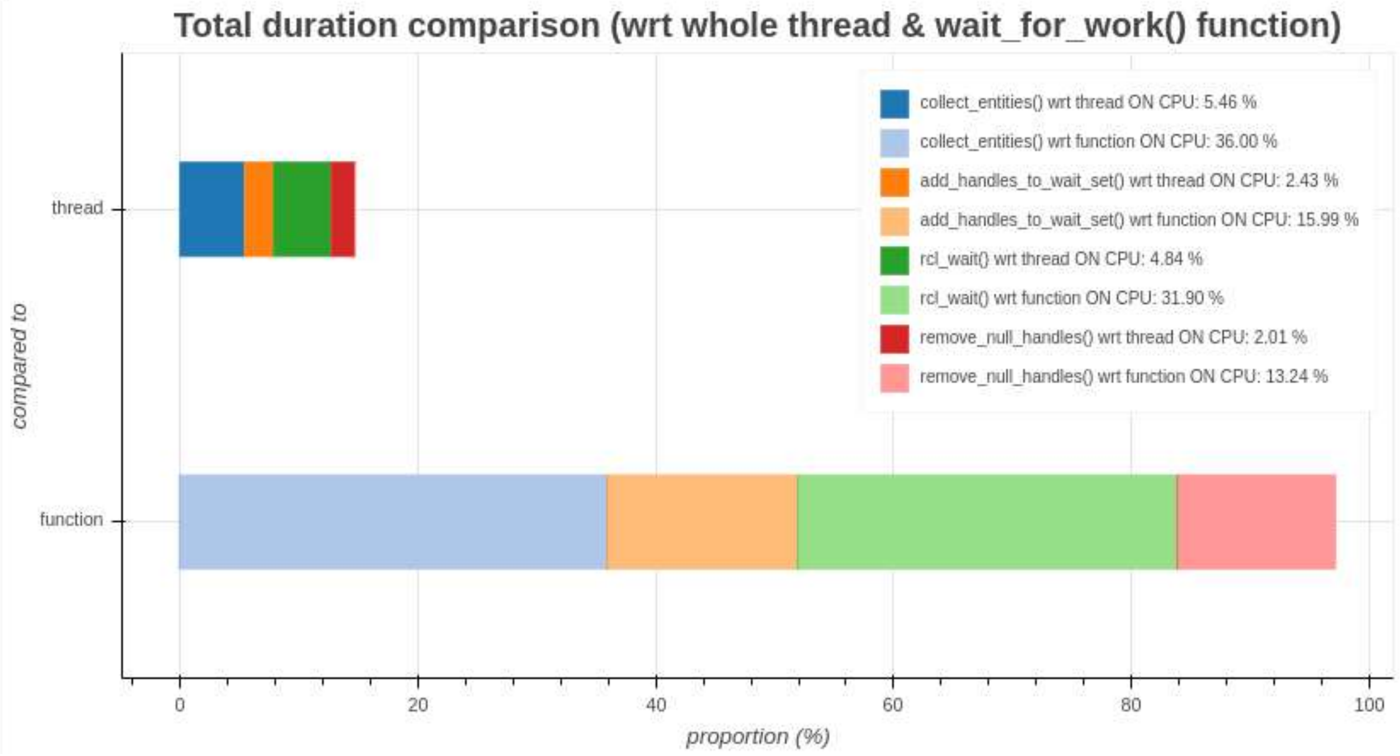
Micro-ROS Build Support

- See https://github.com/micro-ros/micro-ros-build/micro_ros_setup/
- Features
  - Creates the firmware workspace for you
    - RTOS
    - Apps
    - Necessary ROS 2 packages
    - Cross-compilation setup that avoids interference from already source ROS 2 host workspace
  - Creates agent workspace for you
- Example: „ros2 run micro_ros_setup build_firmware.sh“

- Upcoming work: Integrate as CMake macros for ease of use

micro-ROS: ROS 2 on microcontrollers
Executor Performance

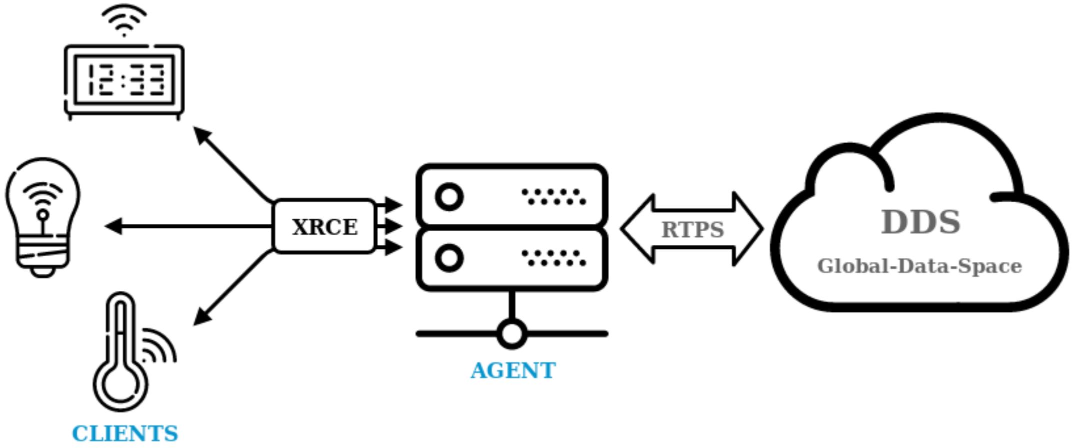
- The current SingleThreadedExecutor adds measurable overhead
  - For some use-cases, just *polling* the middleware already consumes 20% of CPU
- Nobleo has addresses this in rclcpp PR 873
- We‘ve also identified more overhead in the rmw implementations, this is current work.



HAND-OVER TO EPROSIMA FOR REMAINDER OF TALK

micro-ROS: ROS 2 on microcontrollers

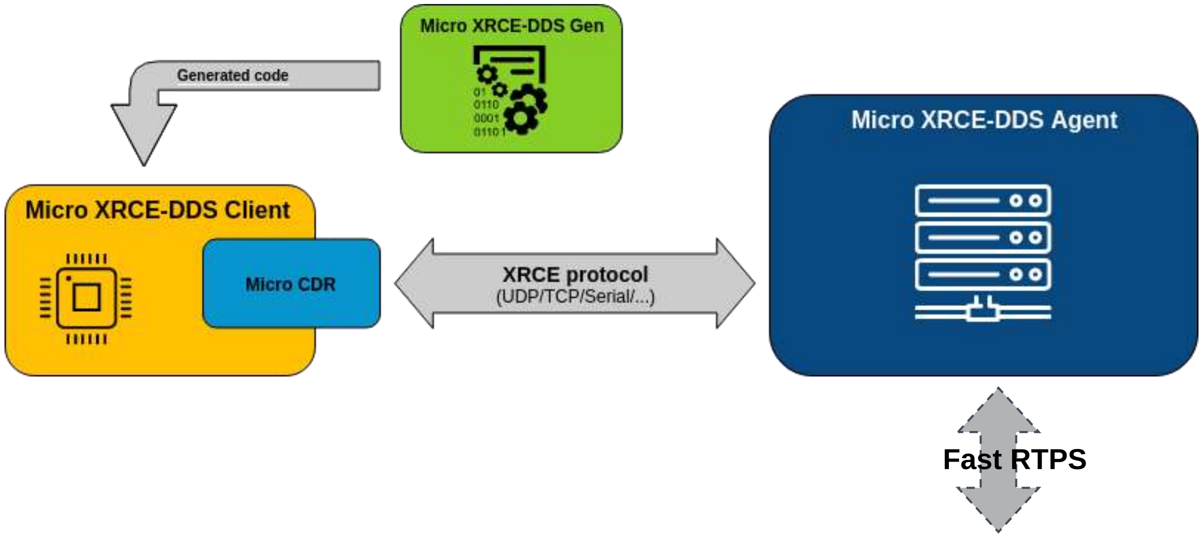
DDS meets MCUs: DDS-XRCE



- OMG's DDS-XRCE (DDS for eXtremely Resource Constrained Environments) brings DDS on MCUs
- Based on Client-Server architecture
  - Power-Saving
  - Stateless
- Agent acts on behalf of Clients (Low resource devices) on the DDS global data space.

micro-ROS: ROS 2 on microcontrollers

Micro XRCE-DDS



- eProsima C99 (Client) / C++11 (Agent) implementation of XRCE protocol
- Multiple and extensible transport support: UDP/IP, TCP/IP, Serial ... or create your own!
- Low memory usage (Client library):
  - Stack: ~2 KB
  - Heap: 0 KB (**only static memory**)
  - .Text (code in Flash): core: 64 KB +/- TCP profile: 2 KB +/- UDP profile: 1 KB +/- Serial profile: 5.5 KB ...
- Agent library API: micro-ROS-Agent
- ROS 2
  - Several success stories: Robotis, Renesas.
  - Crystal and Dashing enabled.

Open-source: https://github.com/eProsima/Micro-XRCE-DDS/

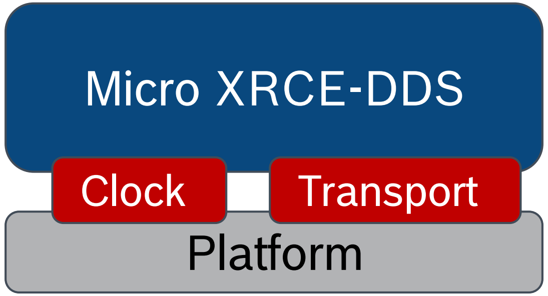
https://micro-xrce-dds.readthedocs.io/en/latest/

micro-ROS: ROS 2 on microcontrollers

Micro XRCE-DDS Client portability

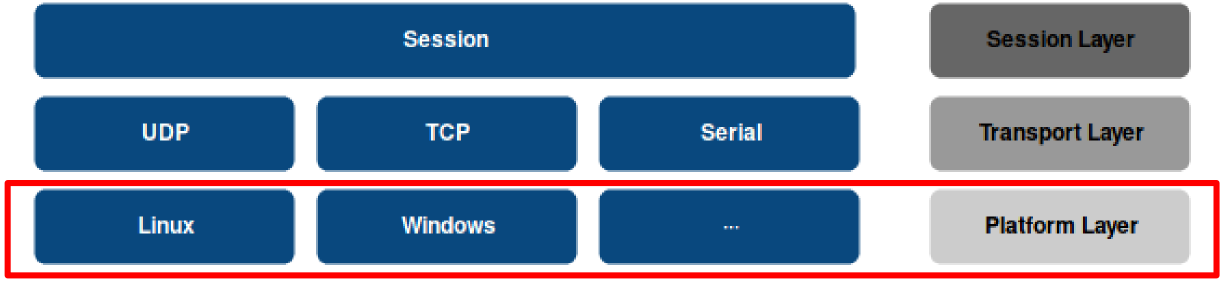
A) Clock dependency.

- Relative clock measurement. Crazyflie timer registers e.g.



B) Platform transports.

- Simple pairs of functions required:
  - Init/Close.
  - Write/Read.
- Common platforms implementation provided.



micro-ROS: ROS 2 on microcontrollers

Drone demo

Crazyflie 2.1

- DDS-XRCE Client

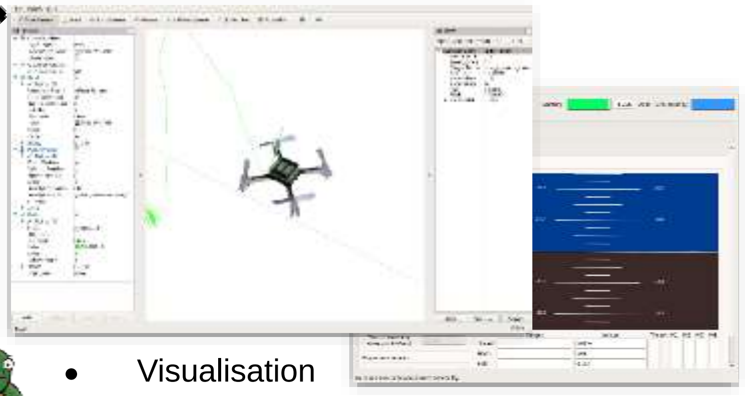
DDS-XRCE

micro-ROS

- thin_kobuki _driver
- DDS-XRCE Client

< 100 KB RAM

DDS-XRCE



- Visualisation
- Control application
- Drone application
- DDS-XRCE Agent

DDS

- DDS-XRCE Agent

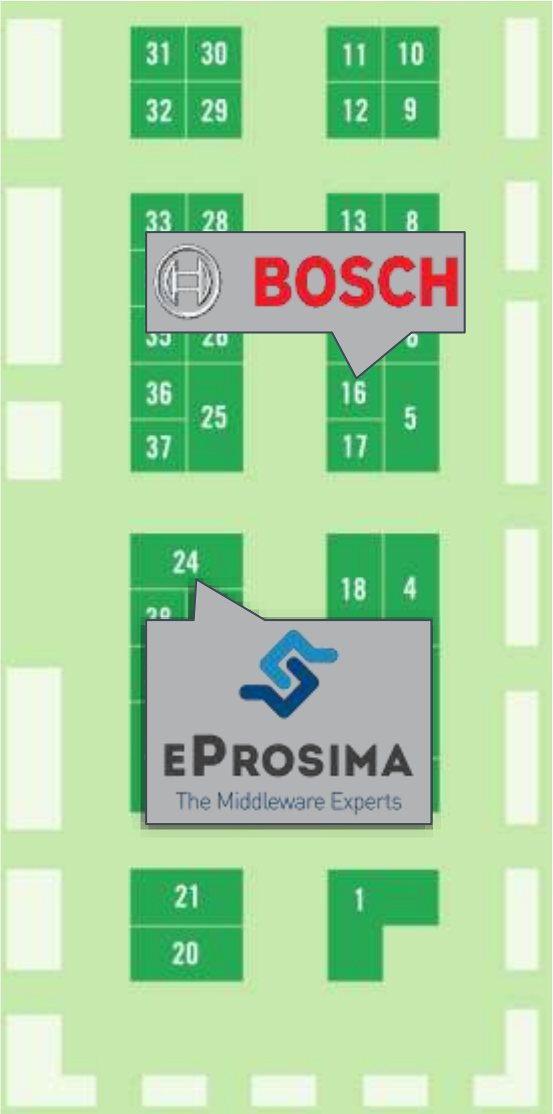
Status

Crystal	Dashing	Next
No executor	LET executor	
RMW Simple communication mechanisms.	RMW configuration.	Complete RMW Implementation.
Basic and nested type support, no arrays.	Full type support.	
NuttX firmware incorporated in build system.		Incorporate new platforms to build system. e.g. FreeRTOS
Plain C API support (RCL).		CPP API support (RCLPP) in some platforms.
Demos.		Demos and more tutorials.
Ready to use dockers.		ROS2 Packages.

micro-ROS: ROS 2 on microcontrollers

Further information

- All open-sourced code at GitHub
  - https://github.com/micro-ROS
- Web-site: micro-ros.github.io
- Slack micro-ros.slack.com
- ROS 2 Embedded Working Group
- ROS Discourse in Embedded category
- ROS 2 Embedded Design Page
  - https://github.com/ros2/design/pull/197


31
30
11
10
32
29
12
9
33
28
13
8
BOSCH
36
25
16
5
37
17
24
18
4
EPROSIMA
The Middleware Experts
21
1
20