ROS 2 Overview Introduction

This is meant to explain ROS 2 basics, design, distribution and features just in overview. Our centralized mainline documentation site is ROS 2 documentation if we need to know more details.

fujitatomoya@github / tomoyafujita@linkedin

Agenda

- Governance
- Distributions
- Architecture / Design
- Basic Concepts
- Samples / Demos
- Remarkable Features
- 3rd Party Packages

Governance

Open Source Robotics Alliance

is a new initiative of the OSRF to organize and strengthen project governance and community involvement. The OSRA is a mixed membership and meritocratic model, following other successful foundations for open-source projects.

ROS Project Management Committee

The ROS Project Management Committee is responsible for the day-to-day operations of the ROS 2 project. The ROS PMC consists of the Project Leader, the ROS PMC Members (who have full voting rights)

See Current ROS PMC Constituents.

Community Sites

- ROS 2 Github
  - Development, Design, Source code
- ROS Discourse
  - Discussion / Announcement / Release
- Robotics Stack Exchange
  - Centralized QA platform
- Working Group / Specific Interest Group
  - Under construction since OSRA structure transition.

Distributions

Distribution Name	Release Date	End Of Life Date
Kilted Kaiju	May 2025	N.A
Jazzy Jalisco	May 23rd, 2024	May 2029
Iron Irwini	May 23rd, 2023	November 2024
Humble Hawksbill	May 23rd, 2022	May 2027

Distros by Year / Percentage

Distro	October 2023	October 2022	YoY Change
Kinetic	1.14%	2.12%	-0.98
Melodic	5.66%	17.19%	-11.53
Noetic	30.51%	34.8%	-4.29
Foxy	6.37%	13.59%	-7.22
Galactic	2.33%	8.12%	-5.79
Humble	32.79%	10.04%	+22.75
Iron	4.97%	0%	+4.97
Rolling	4.82%	3.38%	+1.44

Package downloads, by distro, as a percentage of all downloads from the ROS servers in October of 2022, and 2023.

Platform & Dependencies

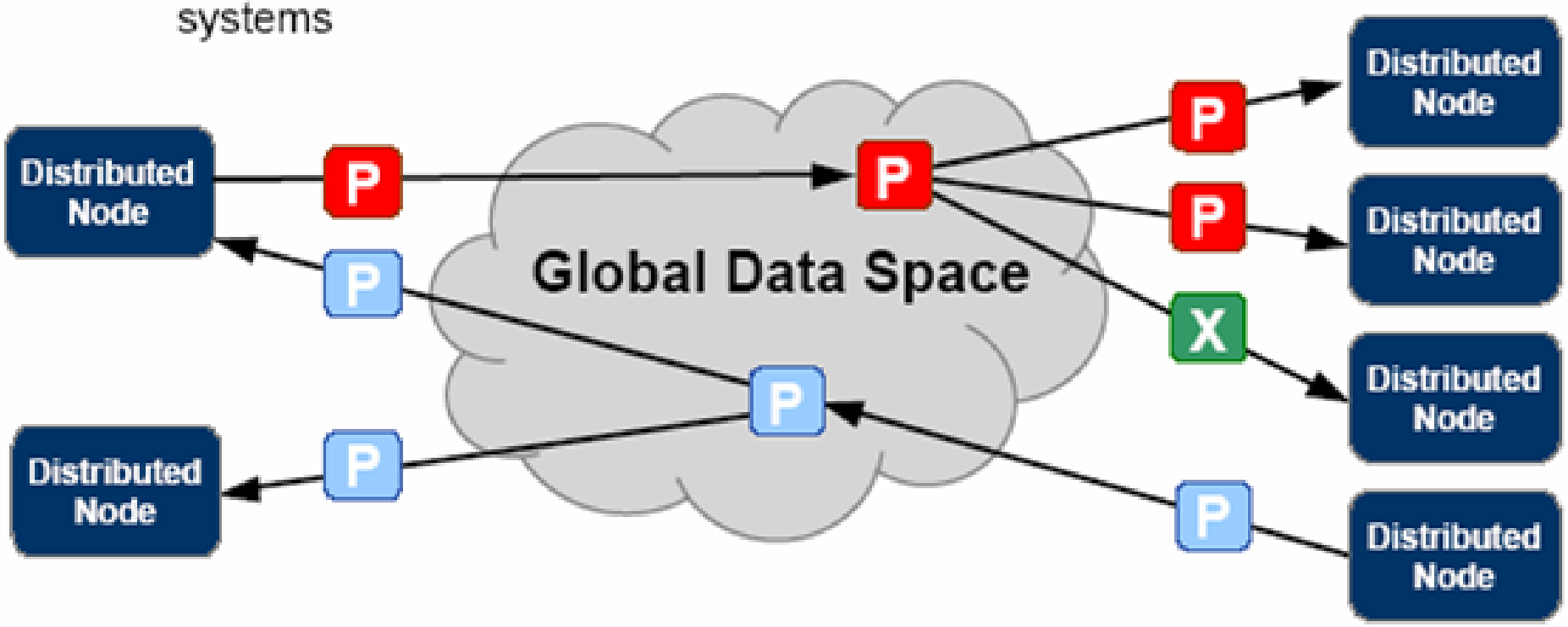
- C++14 / C++17 Standard
- Python version 3.6 or later
- colcon to meta-build system
  - colcon handles ament(ROS 2) and catkin(ROS 1)
- `.msg`, `.srv` and `.action` files
  - Basically compatible, but might need to be updated
- OMG IDL 4.2 Support
- Serializer is ROS 2 MiddleWare's responsibility
- Steady, System and ROS clock available

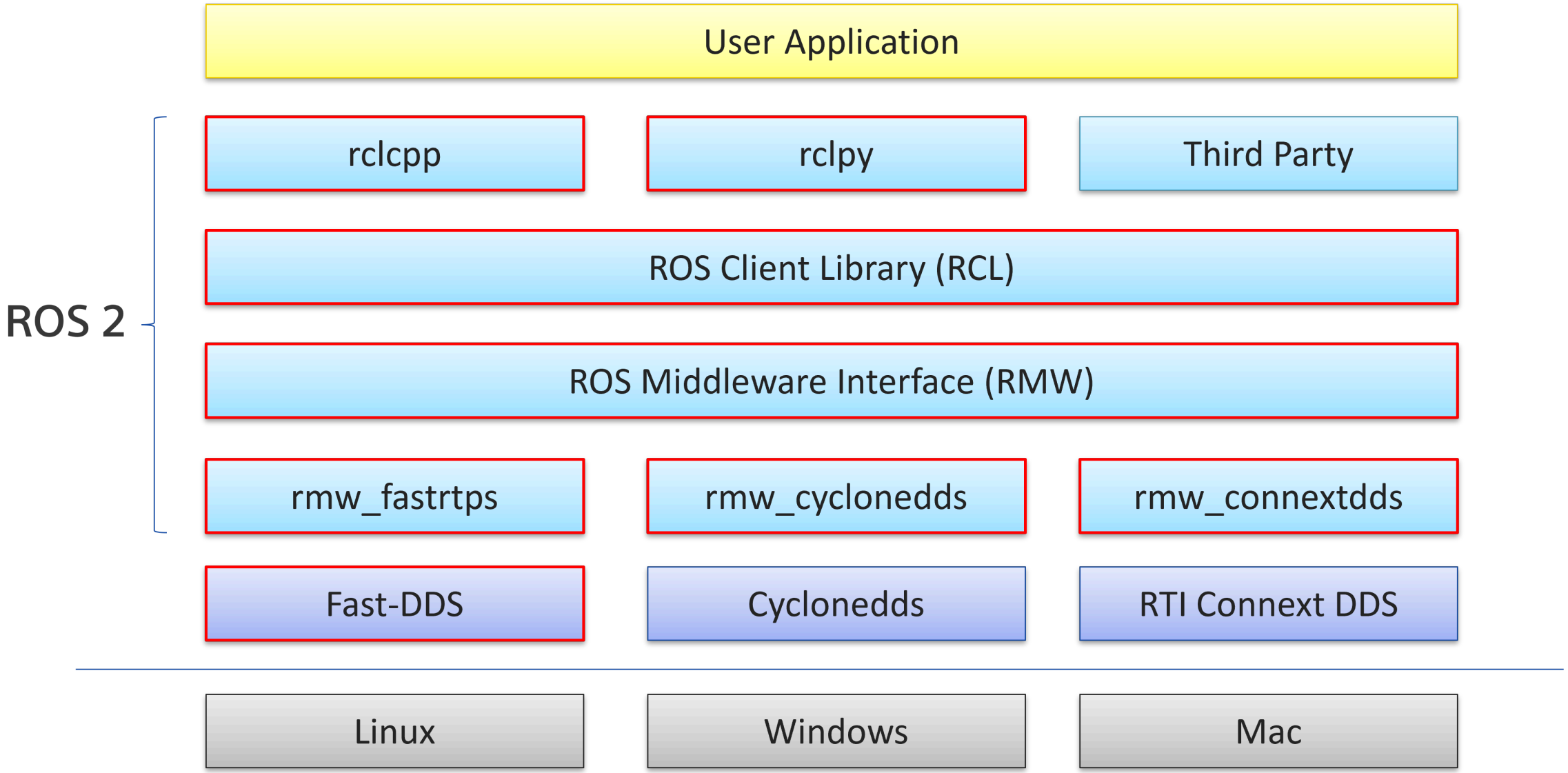
Decoupling

- Location: reduce dependencies
- Redundancy: multiple readers & writers
- Time: data when you want it
- Platform: Connect any set of systems

Benefits

- Modular structure
- Flexibility
- Power




User Application
ROS 2
rclcpp
rclpy
Third Party
ROS Client Library (RCL)
ROS Middleware Interface (RMW)
rmw_fastrtps
rmw_cyclonedds
rmw_connextdds
Fast-DDS
Cyclonedds
RTI Connext DDS
Linux
Windows
Mac

Data Distribution Service

The **OMG Data-Distribution Service for Real-Time Systems® (DDS®)** is the first open international middleware standard directly addressing publish-subscribe communications for real-time and embedded systems.

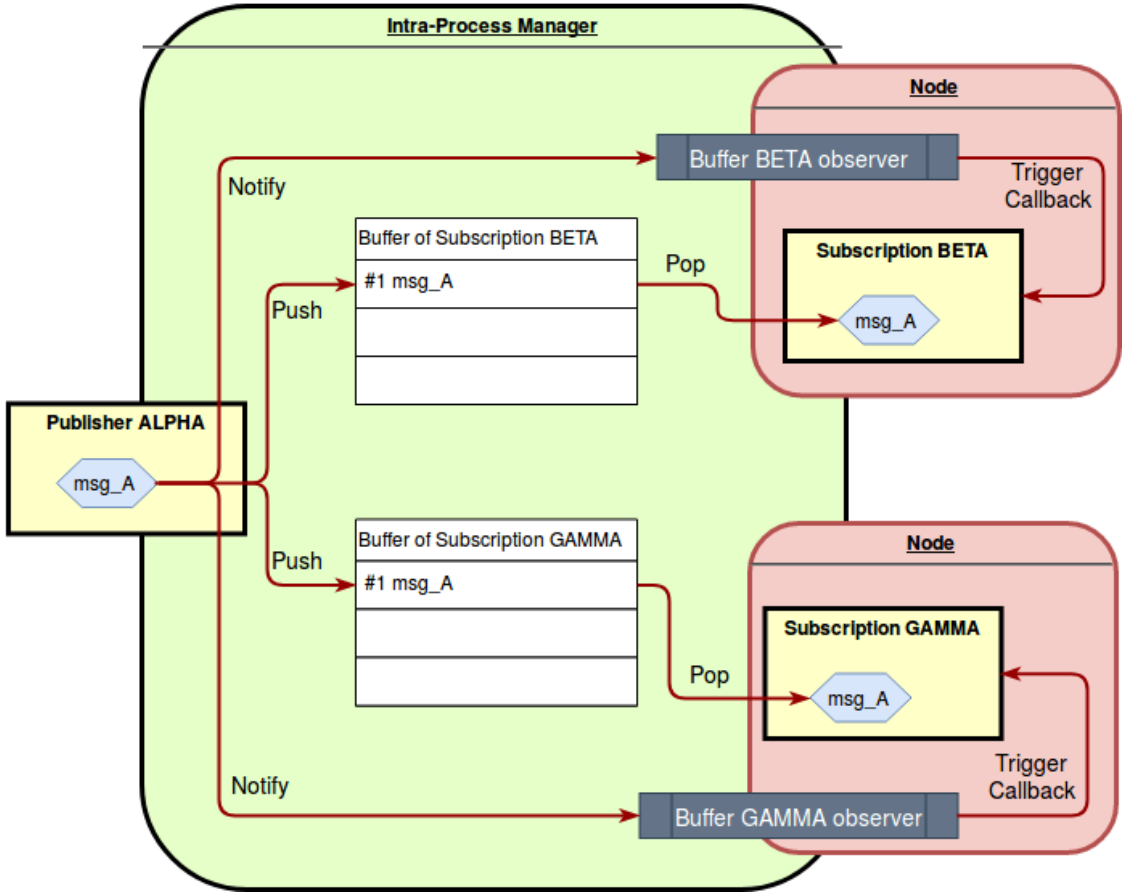
DDS introduces a virtual **Global Data Space** where applications can share information by simply reading and writing **data-objects** addressed by means of an application-defined name (Topic) and a key. DDS features fine and extensive control of **QoS** parameters, including reliability, bandwidth, delivery deadlines, and resource limits. DDS also supports the construction of **local object models** on top of the Global Data Space.

Basic Concepts

- Node / LifecycleNode / Components
- Executor
- Topics, Services, Parameters and Actions
- Logging
- Launch
- Bagging

Node

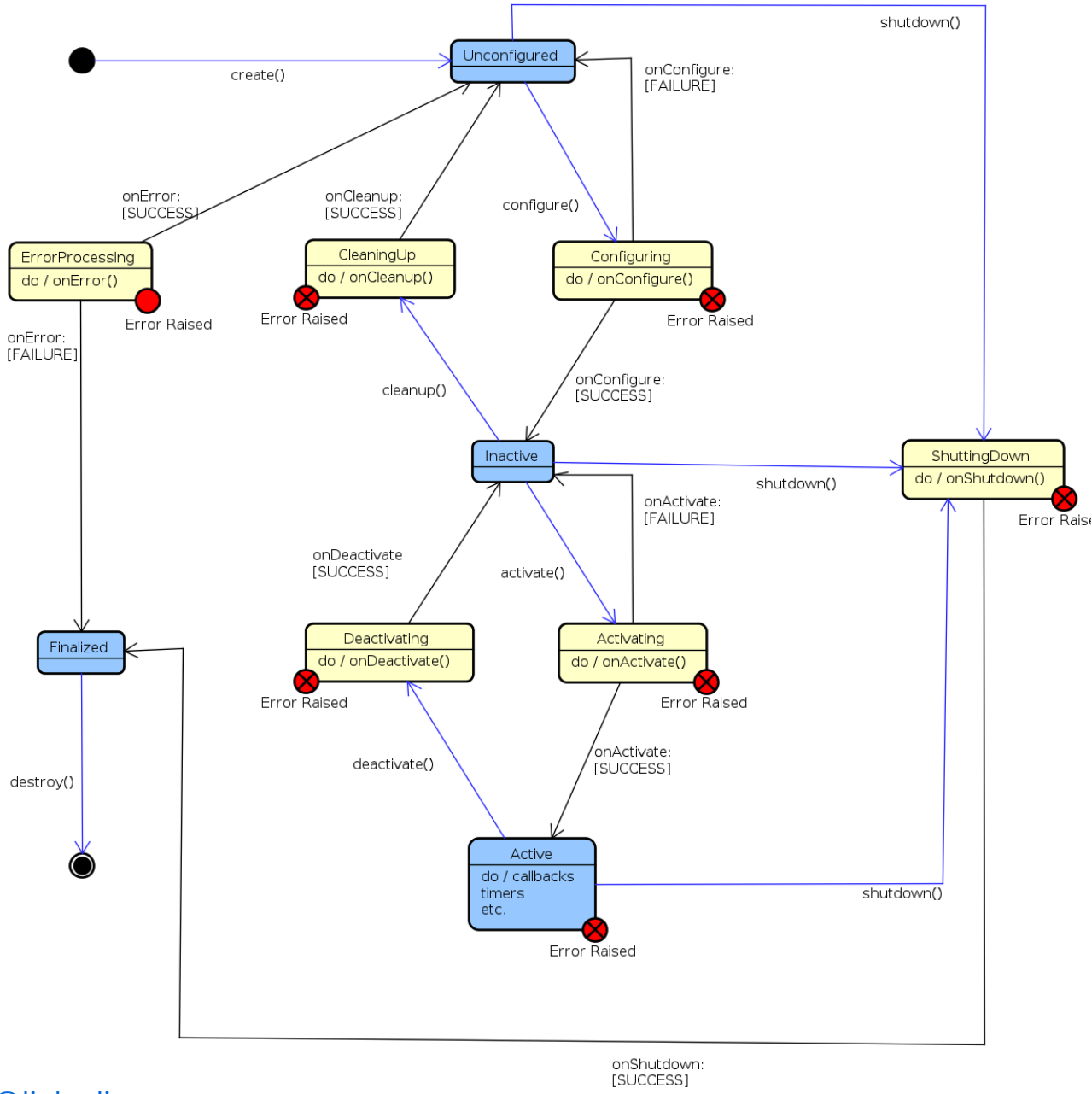
- A node can be a component
- Single process can be constructed with multi-components
- Component can be loaded during runtime via component container
- Intra-Process Manager



Node LifeCycle

A managed life cycle for nodes allows greater control over the state of ROS system. It allows `roslaunch` to ensure that all components have been instantiated correctly before it allows any component to begin executing its behavior. It will also allow nodes to be restarted or replaced on-line.

- Primary State
  - `Unconfigure`, `Inactive`, `Active` and `Finalized`
- Transition State
  - `Configuring`, `ClearningUp`, `ShuttingDown`, `Activating`, `Deactivating` and `ErrorProcessing`


shutdown()
Unconfigured
create()
onConfigure:
[FAILURE]
onError:
[SUCCESS]
onCleanup:
[SUCCESS]
configure()
ErrorProcessing
do / onError()
Error Raised
CleaningUp
do / onCleanup()
Error Raised
Configuring
do / onConfigure()
Error Raised
onError:
[FAILURE]
cleanup()
onConfigure:
[SUCCESS]
Inactive
shutdown()
ShuttingDown
do / onShutdown()
Error Raise
onActivate:
[FAILURE]
onDeactivate
[SUCCESS]
activate()
Finalized
Deactivating
do / onDeactivate()
Error Raised
Activating
do / onActivate()
Error Raised
destroy()
deactivate()
onActivate:
[SUCCESS]
Active
do / callbacks
timers
etc.
Error Raised
shutdown()
onShutdown:
[SUCCESS]

Components

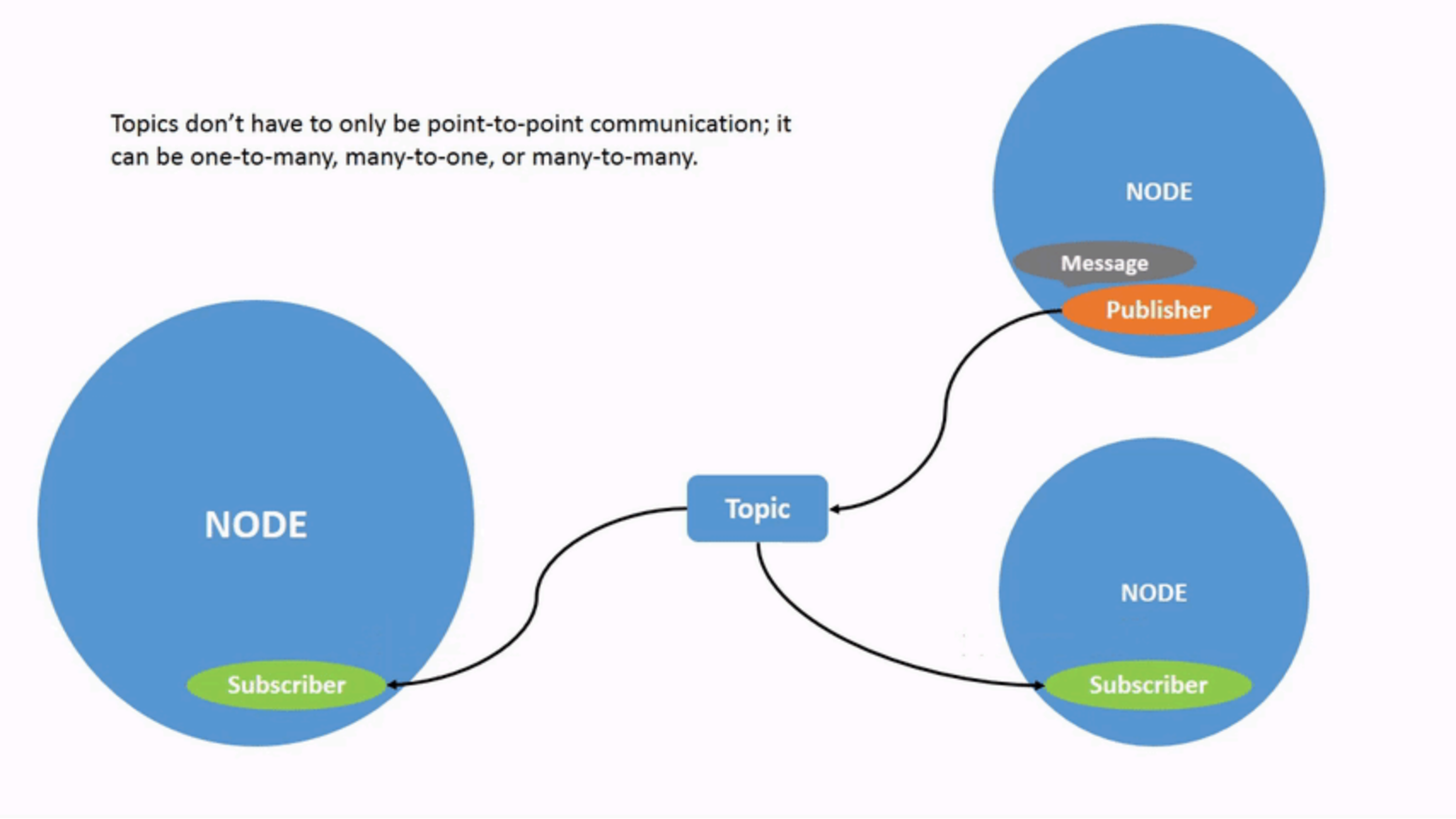
- Just like ROS 1 Nodelet
- Multiple nodes in a single process
- Can load and unload the composable node
- Composable nodes as shared libraries

Executor

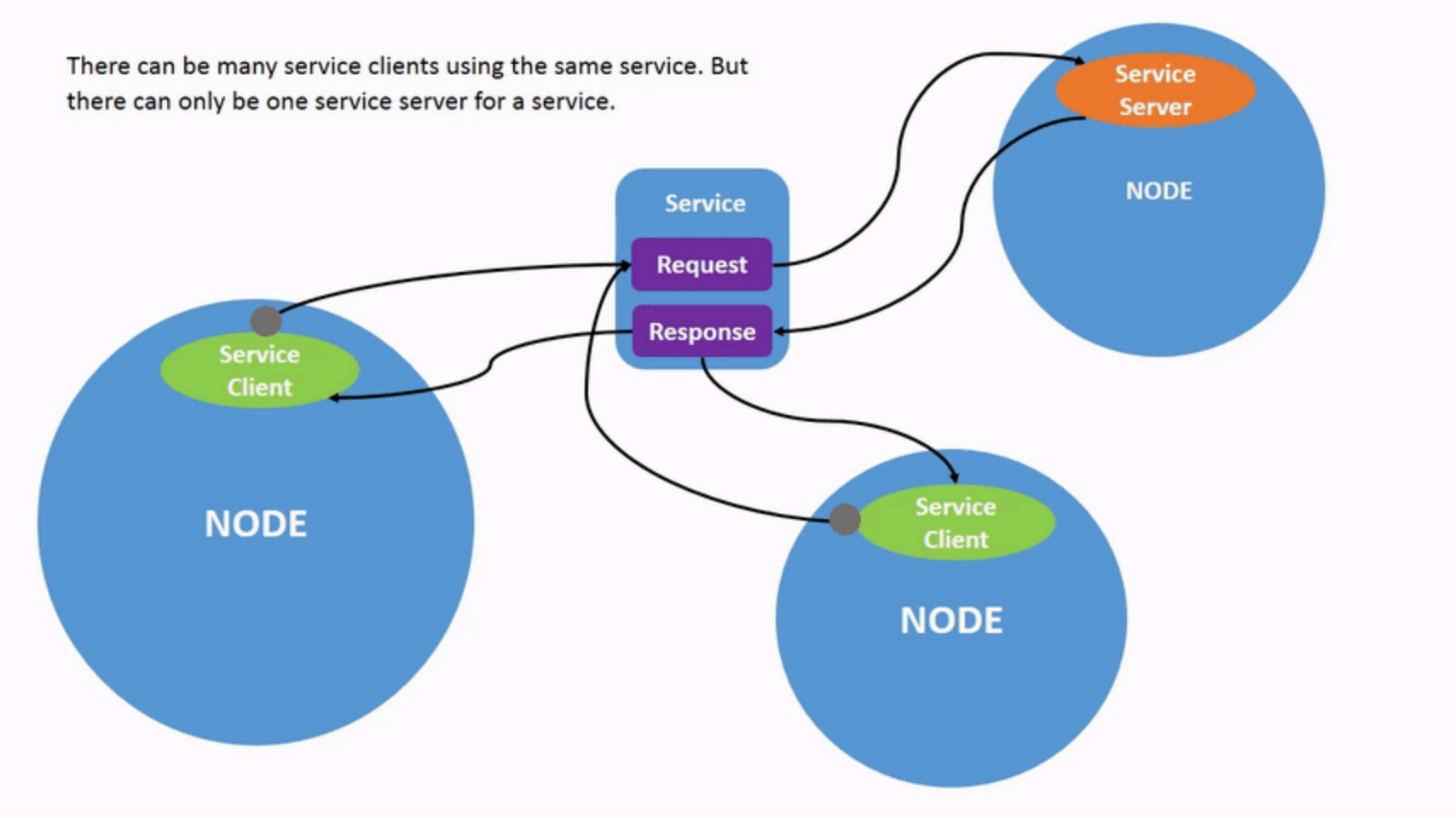
ROS 2 provides executor class aside from Node class, which can be used to dispatch the task to execute the event such as subscription, timer, service and any waitable so on. application has flexibility to use executor as it likes.

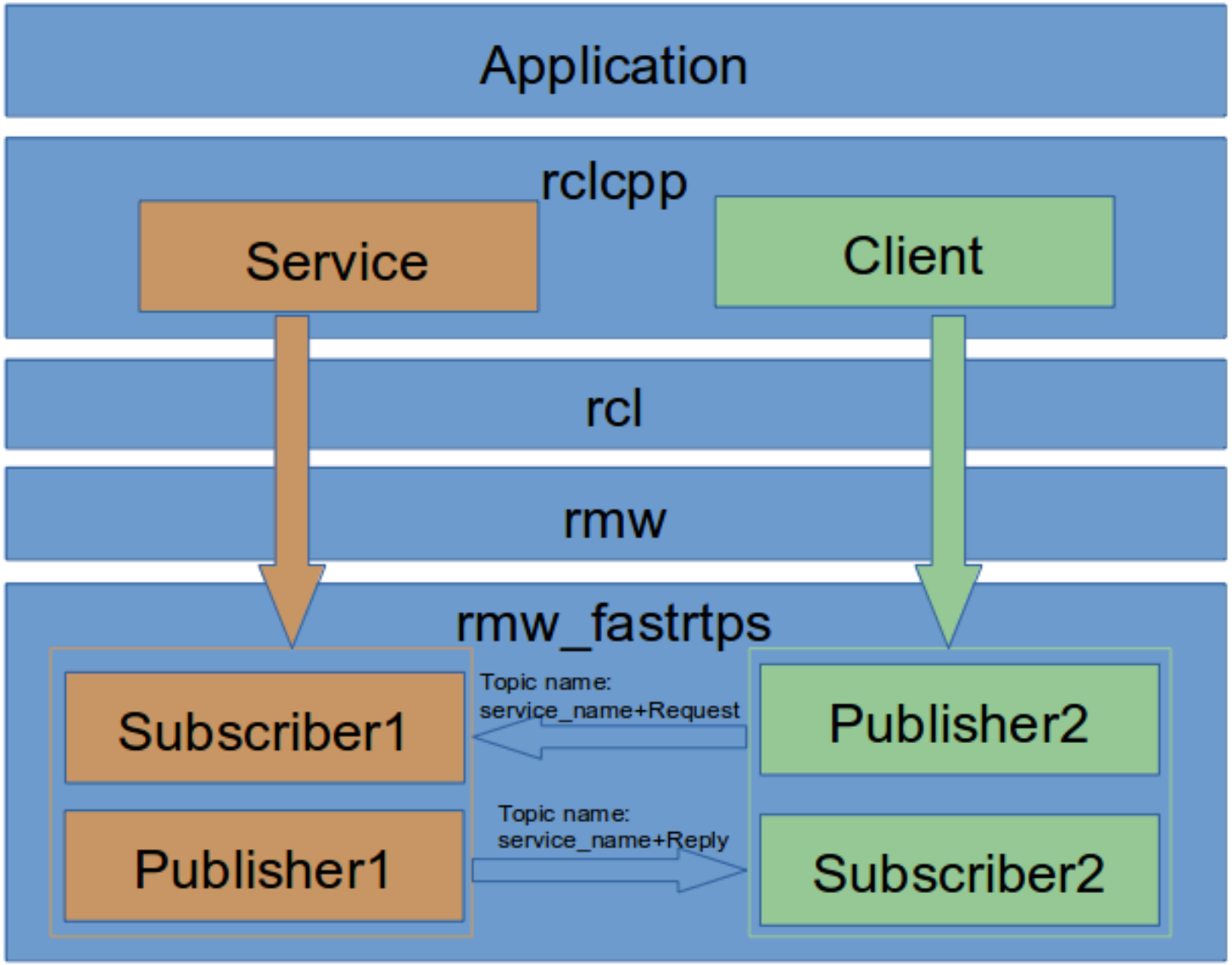
- SingleThreadedExecutor: single thread to dispatch task and execute.
- MultiThreadedExecutor: mulith-thread to dispatch task and execute. (via `std::thread::hardware_concurrency`)
- ~~StaticSingleThreadedExecutor~~ (deprecated): statically collect entities for task. that said once spins, it will not collect or update any entities.
- EventExecutor: Uses events queue in the Executor to execute the entities from associated nodes.

Topics


Topics don't have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.
NODE
Message
Publisher
NODE
Subscriber
Topic
NODE
Subscriber

Service


There can be many service clients using the same service. But there can only be one service server for a service.
Service
Request
Response
Service Server
NODE
Service Client
NODE
Service Client
NODE

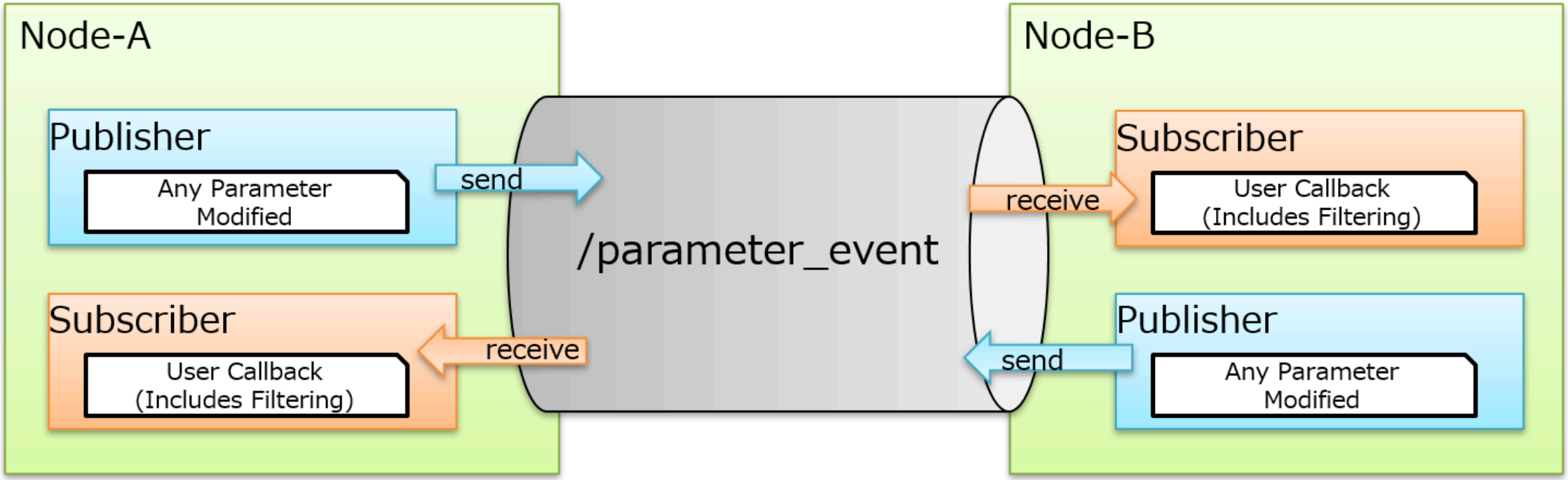


Service architecture

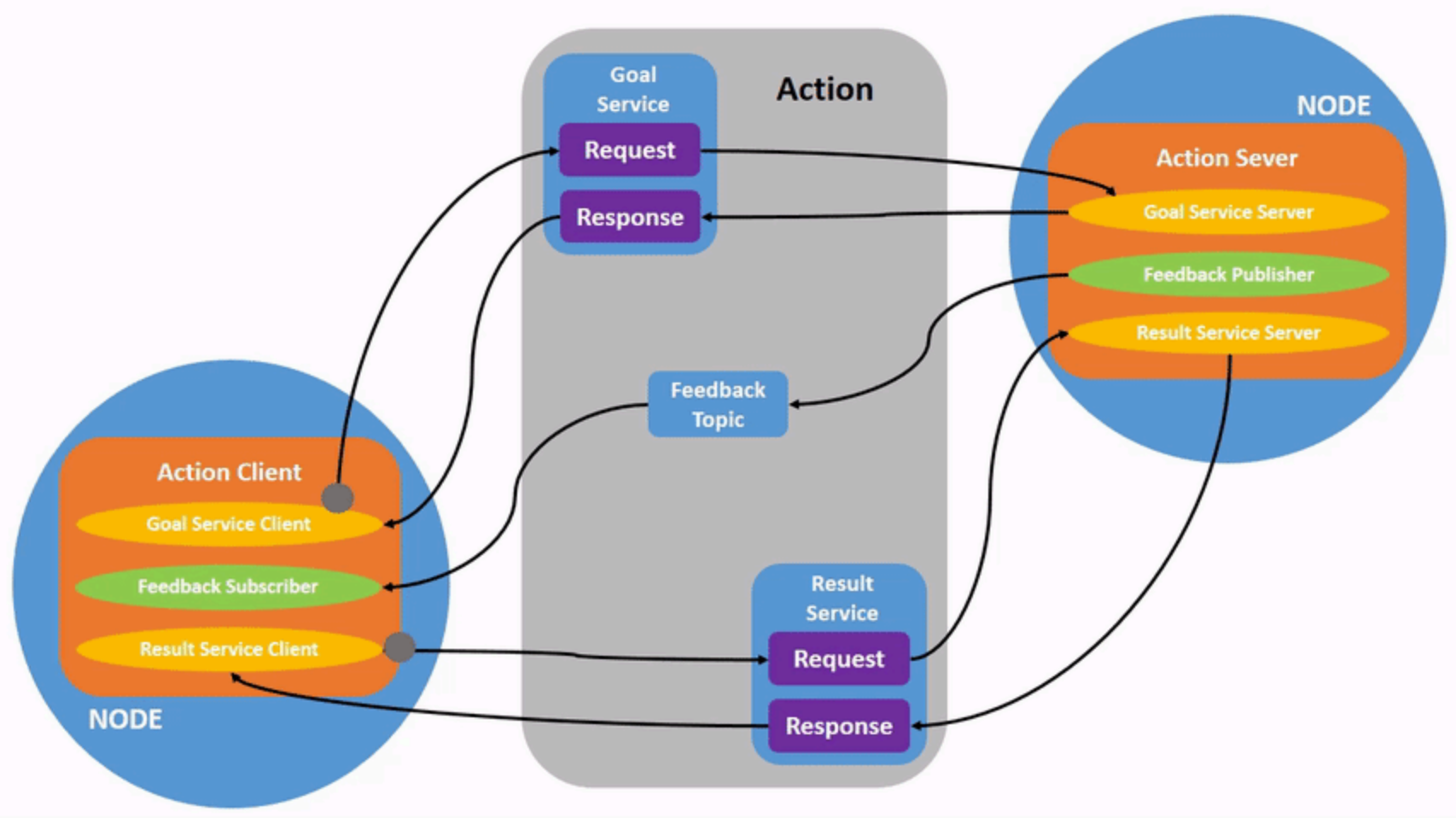
Parameter

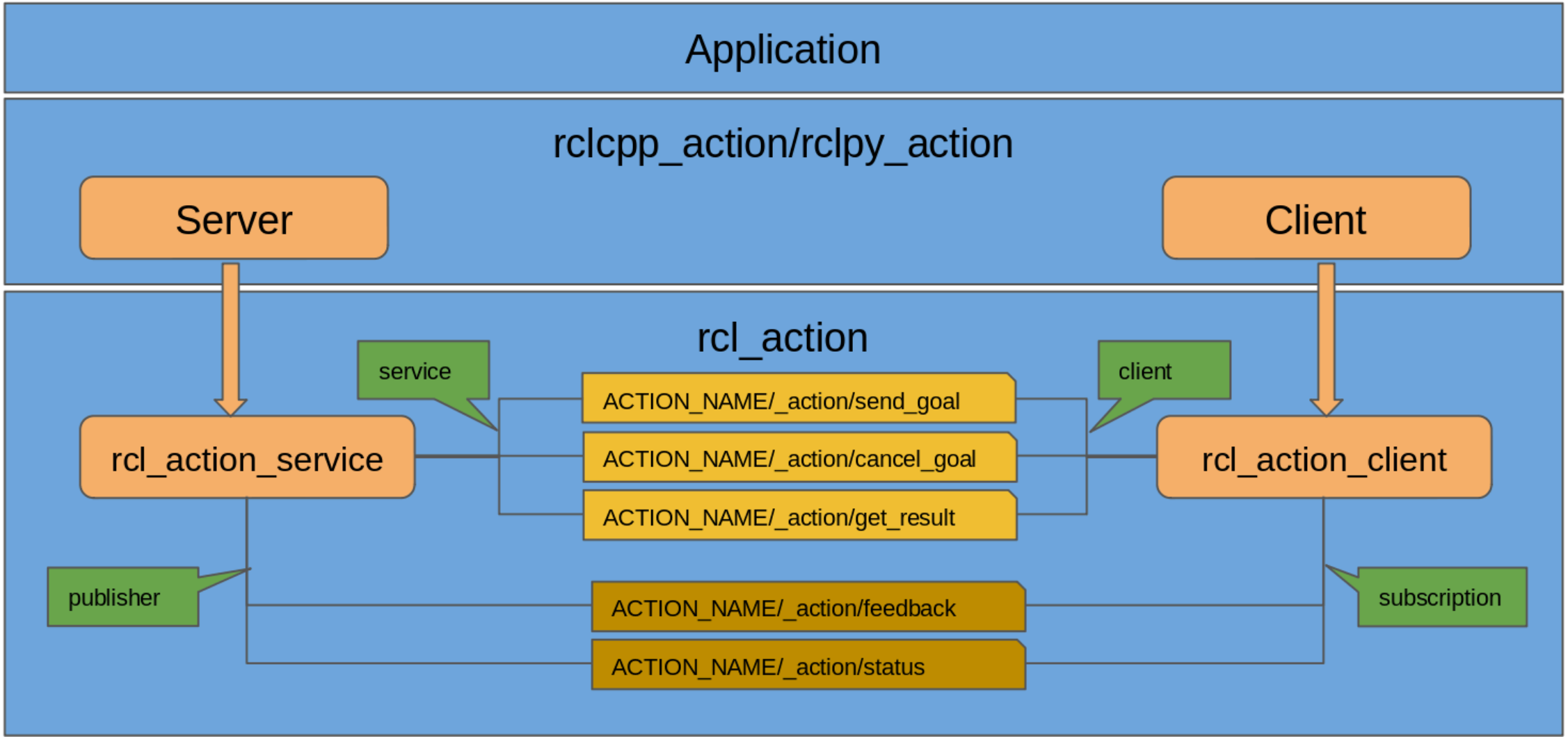
- No parameter server
- Parameters are hosted in Node
- Async / Sync Parameter Client
- `get`, `set`, `list`, `delete` and `describe` methods
- user callback to validate parameter change

Parameter Events

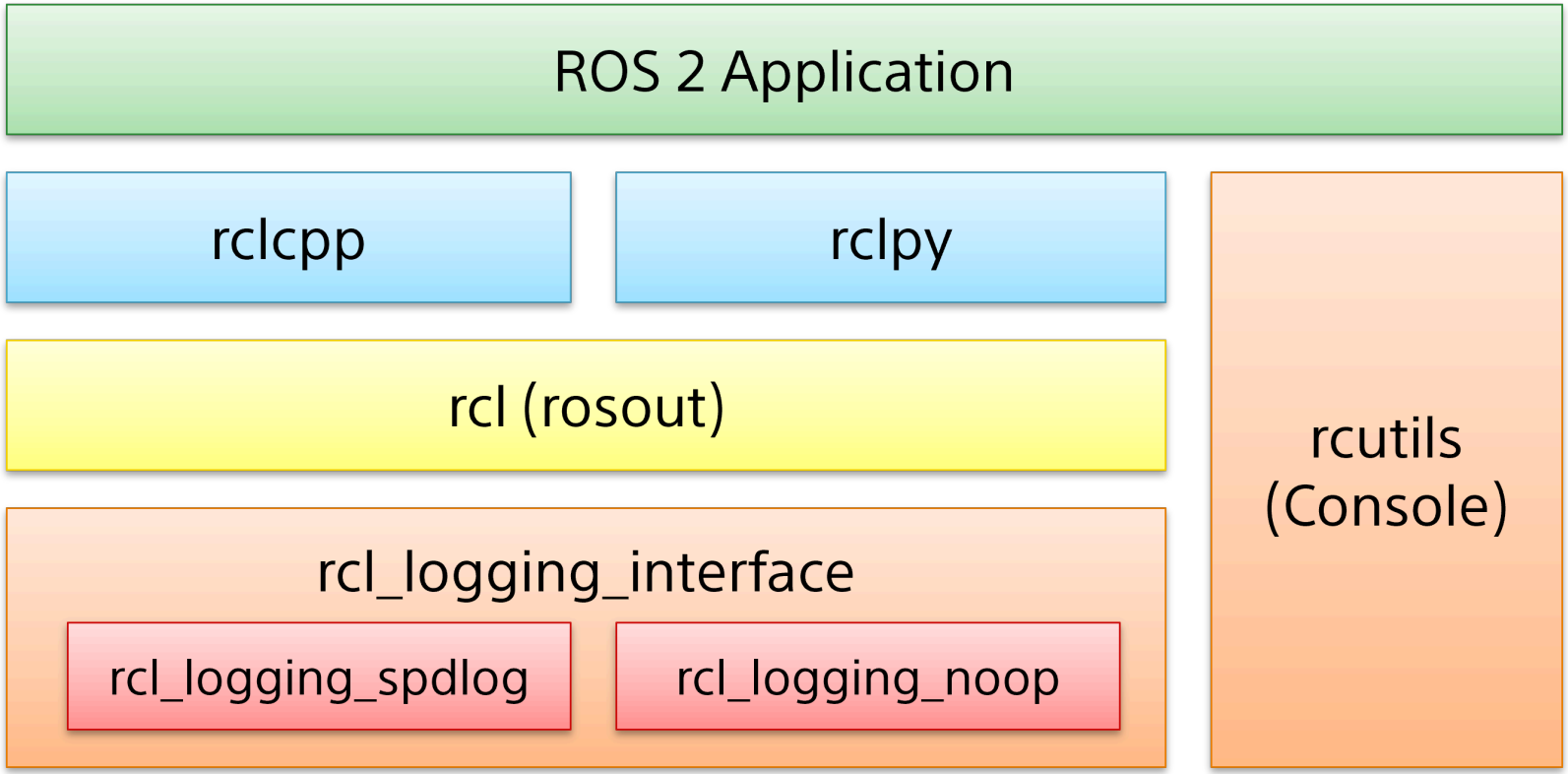


Actions


Goal
Service
Request
Response
Action
NODE
Action Sever
Goal Service Server
Feedback Publisher
Result Service Server
Feedback
Topic
Action Client
Goal Service Client
Feedback Subscriber
Result Service Client
NODE
Result
Service
Request
Response


Application
rclcpp_action/rclpy_action
Server
Client
rcl_action
service
client
ACTION_NAME/_action/send_goal
ACTION_NAME/_action/cancel_goal
ACTION_NAME/_action/get_result
rcl_action_service
rcl_action_client
publisher
subscription
ACTION_NAME/_action/feedback
ACTION_NAME/_action/status

Logging


ROS 2 Application
rclcpp
rclpy
rcl (rosout)
rcl_logging_interface
rcl_logging_spdlog
rcl_logging_noop
rcutils
(Console)

- logging subsystem can output the log to console, files and `/rosout` topic.
- `/rosout` topic is to publish and subscribe all logging data in ROS 2 system.
- Multiple severity levels are supported.
- Environmental variables to support log configuration and format.

Launch

- Similar with ROS 1 `roslaunch`.
- The launch system in ROS 2 is meant to automate the running of many nodes with a single command.
- ROS 2 launch files can be written in Python, XML, or YAML.
- Parameters, remapping rules arguments and process related events are supported.

see more details for https://design.ros2.org/articles/roslaunch.html.

rosbag2

- `ros2 bag` is a command line tool for recording data published on topics and services in your ROS 2 system.
- backend data base can be configured either MCAP (default from Iron) or SQLite3 (default until humble)
- Not only for topics, but services can be also recorded and playback.

Demos / Examples

- Topic
- Service
- Parameter
- Action
- Lifecycle
- Composition

Remarkable Features

- Quality of Service
- Physical Domain / Discovery Range Control
- ROS 2 Security
- True Zero Copy / Loaned Message
- Sync / Async Publisher Mode
- Wait for Acknowledgement
- Content Filtered Topics
- Network Identifier (ToS, DSCP)
- Micro-Controller Support (Micro-ROS)
- service echo / rosbag2 service support

Quality of Service

- QoS policies such as history, depth, reliability, durability, deadline so on.
- QoS preset profiles, e.g) system_default, sensor_data and service.
- QoS events, callback to be fired based on the event, i.e) deadline missed, liveliness missed.
- QoS incompatibility can happen between publisher and subscriber.
- MachedEvent can be generated when any publisher and subscription establishes or drops the connection between them.

`ROS_DOMAIN_ID` Partition

- ROS_DOMAIN_ID, primary mechanism for having different logical networks share a physical network is known as the Domain ID.

Improved Dynamic Discovery

- Discovery protocol initiated in the same subnet By default.
- User can configure the discovery range with environmental variable, such as only localhost, specific peers.

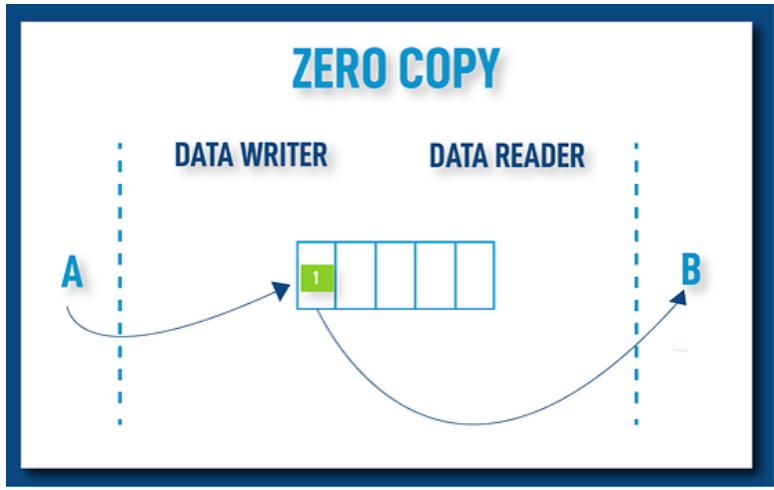
ROS 2 Security

The ability to secure communications among nodes within the ROS 2 computational graph.
Security files enable encryption and authentication, and define policies both for individual nodes and for the overall ROS graph.

see Setting up security for more details.

Zero Copy Loaned Message

- True Zero Copy / Write Just Once
- No encapsulation / No Serialization
- Copy-Less communication channel using shared memory.
- Application borrows memory from RMW Implementation.
- Fast-DDS v2.2.0 (ros:galactic or later)

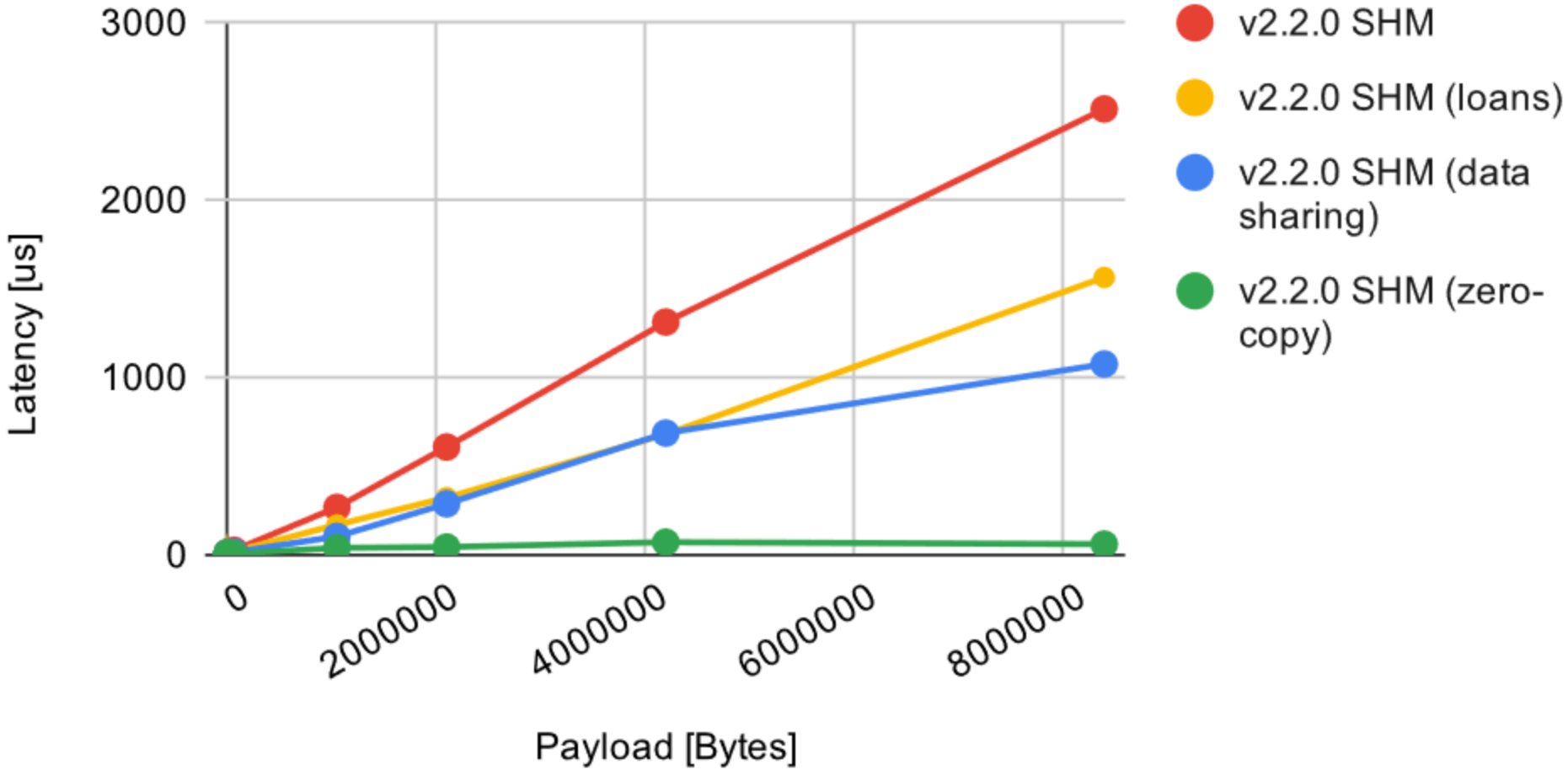


- `LoanedMessage`
  - This class needs to be used by application to borrow memory from middleware during publication.
  - LoanedMessage Talker Demo
- Constraints
  - bound data type only supported. (plain data type)
- Configuration
  - Suitable for `PREALLOCATED_MEMORY_MODE` and `PREALLOCATED_WITH_REALLOC_MEMORY_MODE` memory configurations only
  - see MemoryManagementPolicy

DDS	Tier	Description
Fast-DDS	1	Zero Copy supported. Even with shared memory transport, DDS feature full supported.
Cyclonedds	1	DDS with shared memory bypass by iceoryx which requires daemon. When using shared memory, DDS feature cannot be supported such as QoS.
RTI Connext DDS	1	Commercial only, but most featured DDS implementation
Eclipse Iceoryx	2	Only shared memory w/o network, daemon process required.

Fast DDS v2.2.0 SHM Reliable

Median comparison



Sync / Async Publication

in general with ROS 2, we recommend async send not to block the application threads, so that application can be available for tasks in application perspective. on the other hand, async causes context switches, this could affect the latency.

- eProsima Fast-DDS can be configured (default async)
  - see Publication Mode to change this behavior.
- Eclipse Cyclonedds only supports sync send.

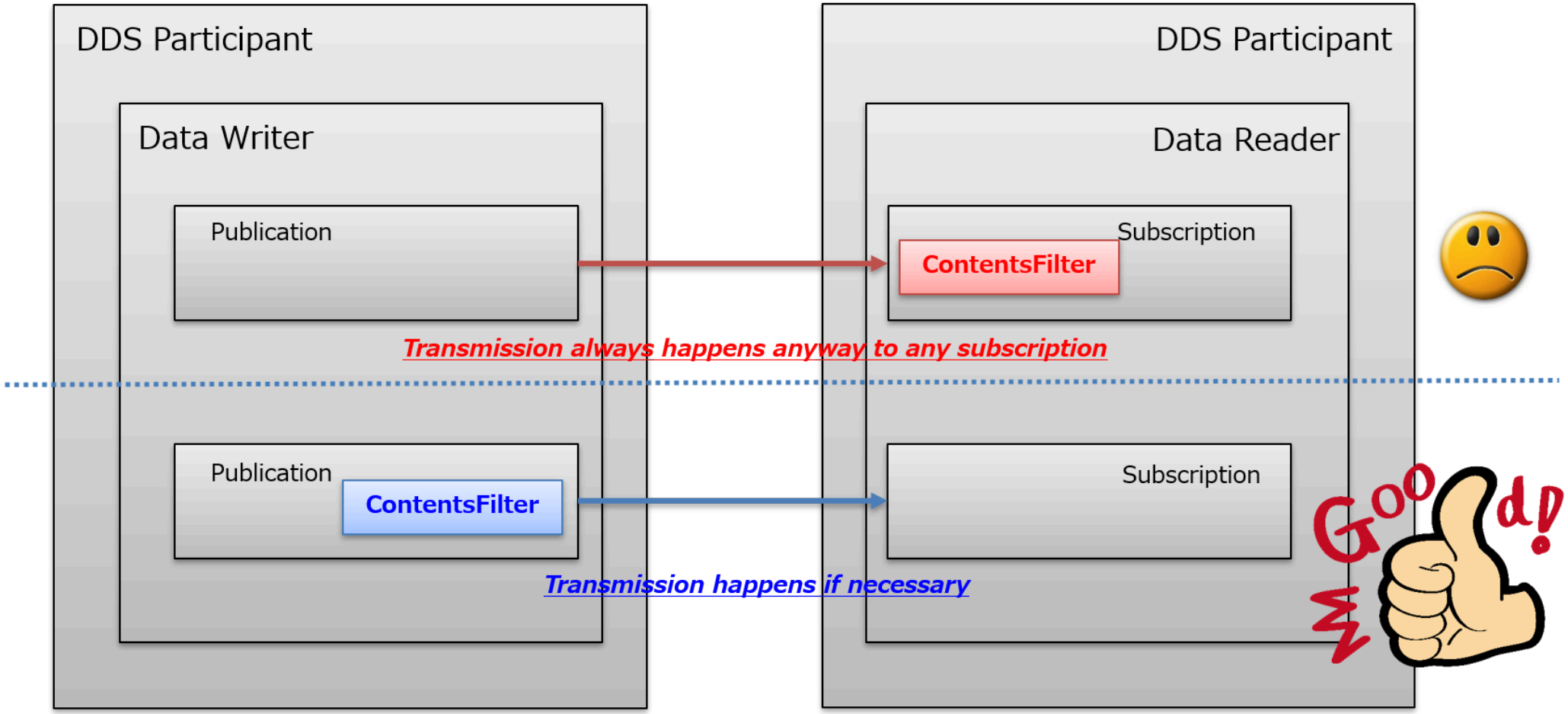
Wait for Acknowledgements

This operation blocks the calling thread until either all data written by the reliable DataWriter entities is acknowledged by all matched reliable DataReader entities, or else the duration specified by the max_wait parameter elapses, whichever happens first. A return value of OK indicates that all the samples written have been acknowledged by all reliable matched data readers; a return value of TIMEOUT indicates that max_wait elapsed before all the data was acknowledged.

Content Filtered Topics

- RMW Content Filtering completed. (Fast-DDS / RTI supported)
- Optimization to conserve network resource.
- ROS2 Design Overview

DDS	Content Filter Supported?
Fast-DDS	Fully(Reader/Writer) Supported
Cyclonedds	Not Supported
RTI Connext DDS	Fully(Reader/Writer) Supported

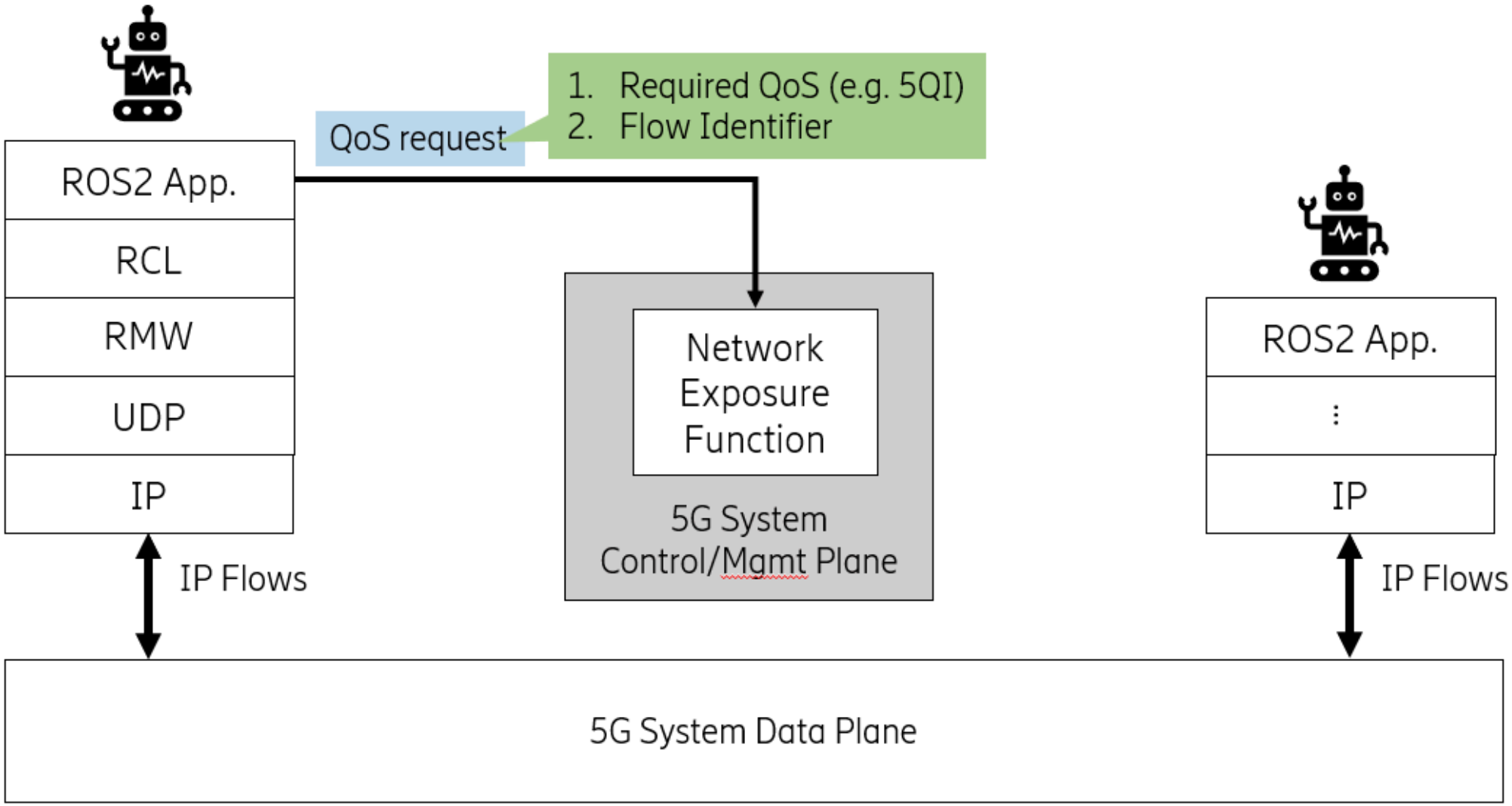

DDS Participant
Data Writer
Publication
ContentsFilter
Transmission always happens anyway to any subscription
Publication
ContentsFilter
Transmission happens if necessary
DDS Participant
Data Reader
Subscription
ContentsFilter
Subscription
Good!

ContentFilteredTopic provides the improvement for CPU consumption and network traffic.

- Reader can get what they need to receive.
- Much less network consumption and user callbacks.
- Application can be agnostic from filtering.
- Filtering expression and parameter is really flexible.
- Filtering can be reconfigured at runtime.

Network Identifier (ToS, DSCP)

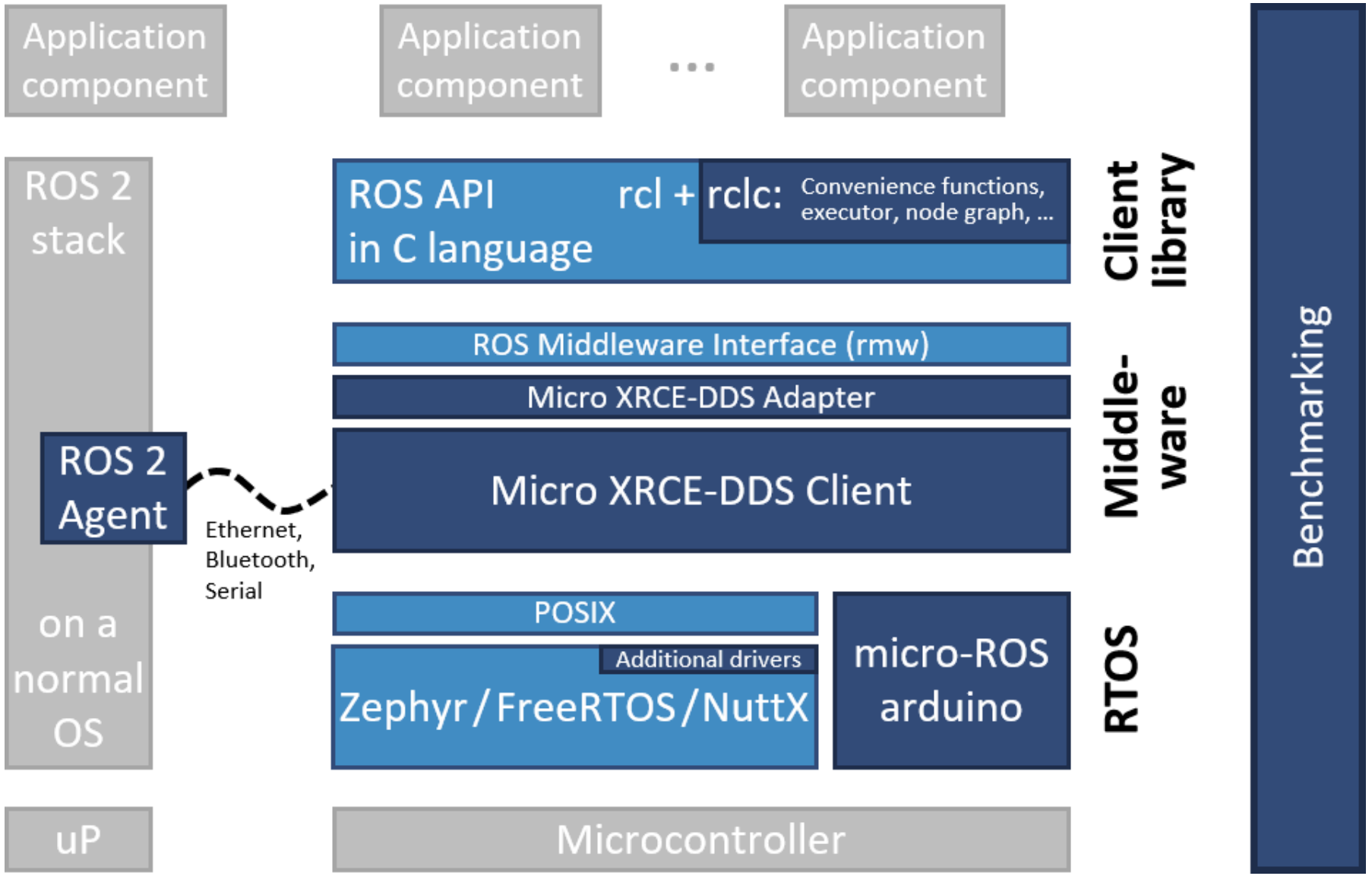
- Differentiated Services is a widely-used QoS architecture for IP networks. The required DS-based QoS is set by the application in the 6-bit DS Code Point (DSCP) sub-field of the 8-bit DS field in the IP packet header.
- 5G network 5QI: The Network Exposure Function (NEF) in the 5G core network provides robust and secure API for QoS specification. This API enables applications to programmatically (HTTP-JSON) specify required QoS by associating 5G QoS Identifiers (5QIs) to flow identifers, as shown in the figure next.

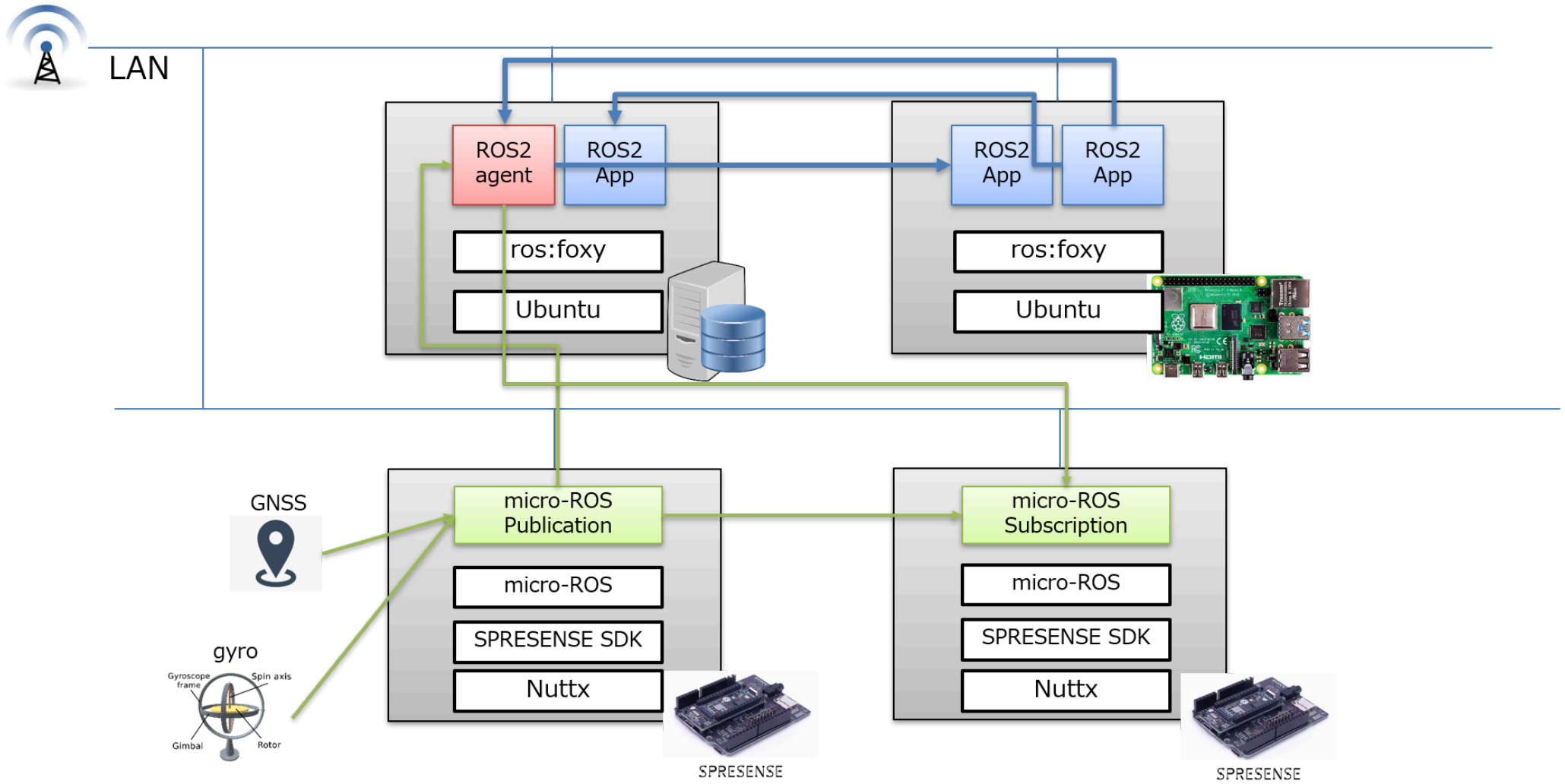

1. Required QoS (e.g. 5QI)
2. Flow Identifier
QoS request
ROS2 App.
RCL
RMW
UDP
IP
IP Flows
Network Exposure Function
5G System Control/Mgmt Plane
ROS2 App.
⋮
IP
IP Flows
5G System Data Plane

Micro-Controller Support

puts ROS 2 onto microcontrollers!

- Extend ROS 2 for Micro-Controllers / Real-Time OS
- ROS 2 agent proxy required
- eXtremely Resource Constrained Environment DDS
- SPRESENSE w/ NuttX Supported

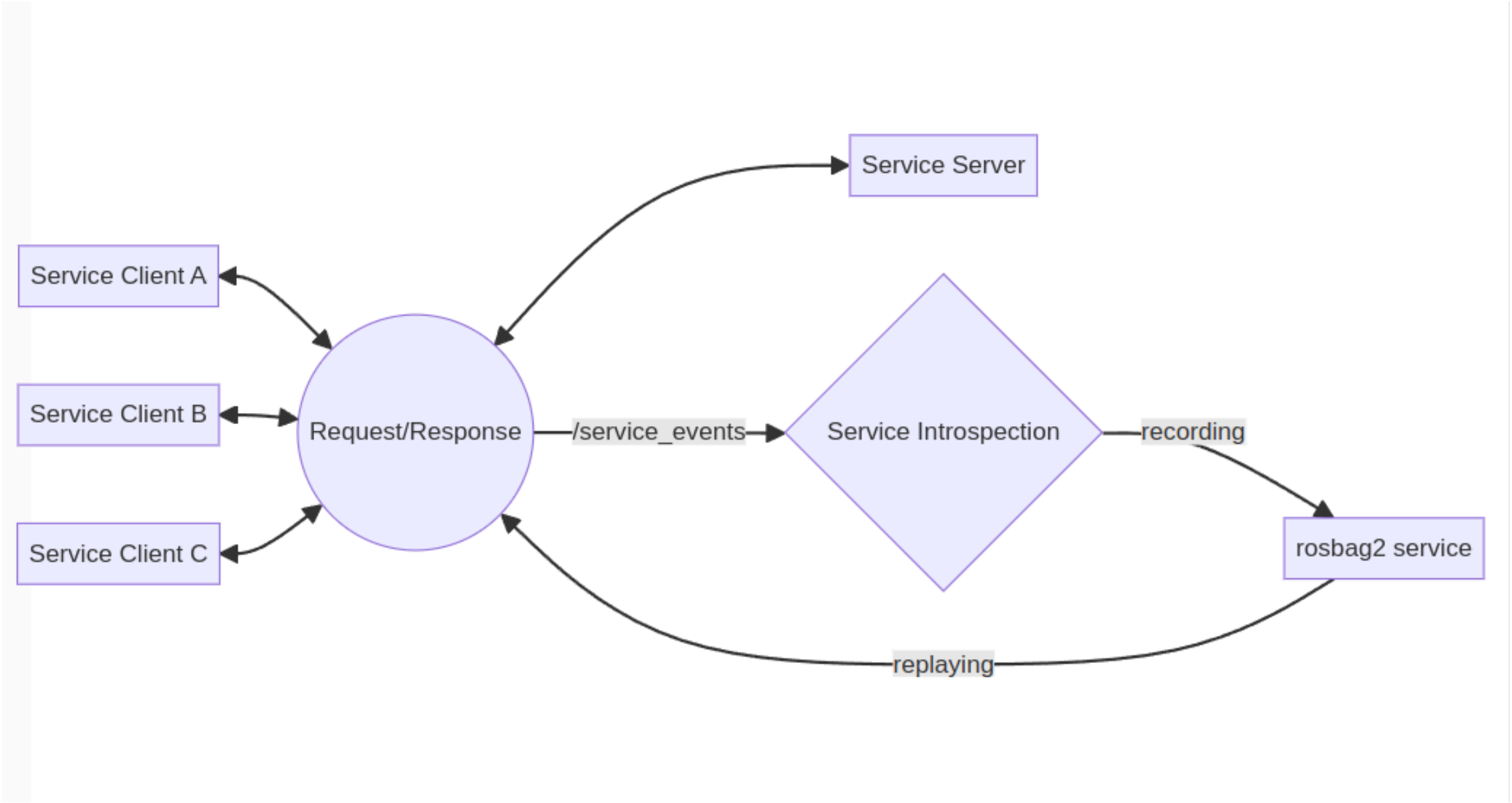

Application component
Application component
...
Application component
ROS 2 stack
ROS API in C language
rcl + rclc:
Convenience functions, executor, node graph, ...
Client library
ROS Middleware Interface (rmw)
Micro XRCE-DDS Adapter
Micro XRCE-DDS Client
Middle-ware
ROS 2 Agent
Ethernet, Bluetooth, Serial
on a normal OS
POSIX
Additional drivers
Zephyr / FreeRTOS / NuttX
micro-ROS arduino
RTOS
Benchmarking
uP
Microcontroller


LAN
ROS2 agent
ROS2 App
ros:foxy
Ubuntu
ROS2 App
ROS2 App
ros:foxy
Ubuntu
GNSS
gyro
Gyroscope frame
Spin axis
Gimbal
Rotor
micro-ROS Publication
micro-ROS
SPRESENSE SDK
Nuttx
SPRESENSE
micro-ROS Subscription
micro-ROS
SPRESENSE SDK
Nuttx
SPRESENSE

rosbag2 service support

This feature depends on Service introspection implemented in Iron. rosbag2 takes advantage of it to record all service requests and responses, and also replays service data from the bag file.

This feature allows you to debug/enhance/test/simulate the ROS 2 services more efficiently.


Service Server
Service Client A
Service Client B
Service Client C
Request/Response
/service_events
Service Introspection
recording
rosbag2 service
replaying

- Recording

```
# All services and all topics
ros2 bag record --all
# All topics
ros2 bag record --all-topics
# All services
ros2 bag record --all-services
```

- Playback

```
ros2 bag play --publish-service-requests bag_path
```

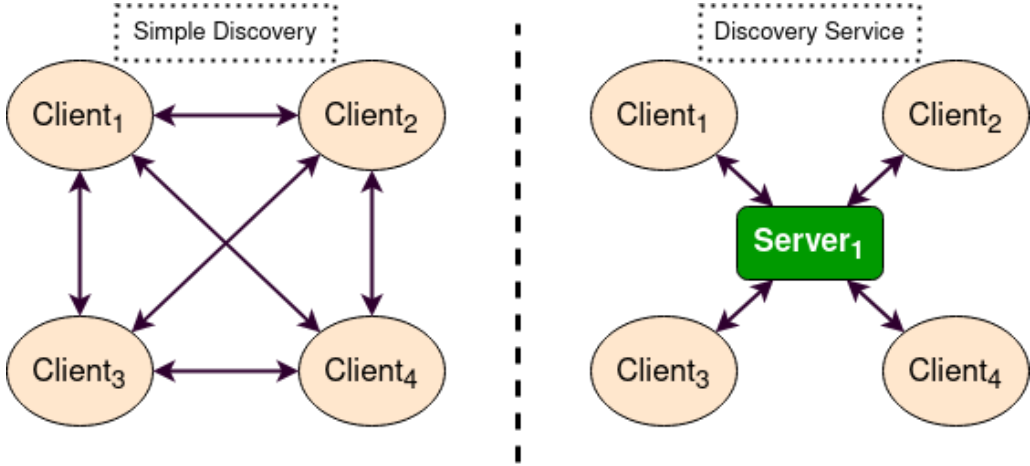
Please check out Managing Service Data Tutorial and Design Document for more information.

System Packages

- Fast-DDS Discovery Server
- rmw_zenoh
- Global Persistent Parameter Server
- rcl_logging_syslog with FluentBit / Fluentd
- ros2ai, next-gen CLI with LLMs.
- Service Load-Balancing (PoC)

Fast-DDS Discovery Server

The Discovery Server provides a Client-Server Architecture that allows nodes to connect with each other using an intermediate server. Each node functions as a discovery client, sharing its info with one or more discovery servers and receiving discovery information from it. This reduces discovery-related network traffic and it does not require multicasting capabilities.



RMW Zenoh

A new ROS MiddleWare (RMW) that integrates Zenoh with ROS 2 and rmw_zenoh is now available. However, it is still a preview because there are some known bugs in it, and we aren't quite ready to commit to it for the long term.

rmw_zenoh is one of `Non-DDS` RMW implementations.

Note that rmw_zenoh requires `Zenoh Router` daemon running.

To use, we need to compile rmw_zenoh from the source.

```
### Start the zenoh router
ros2 run rmw_zenoh_cpp rmw_zenoh

### Talker and  Listener
RMW_IMPLEMENTATION=rmw_zenoh_cpp ros2 run demo_nodes_cpp talker
RMW_IMPLEMENTATION=rmw_zenoh_cpp ros2 run demo_nodes_cpp listener
```

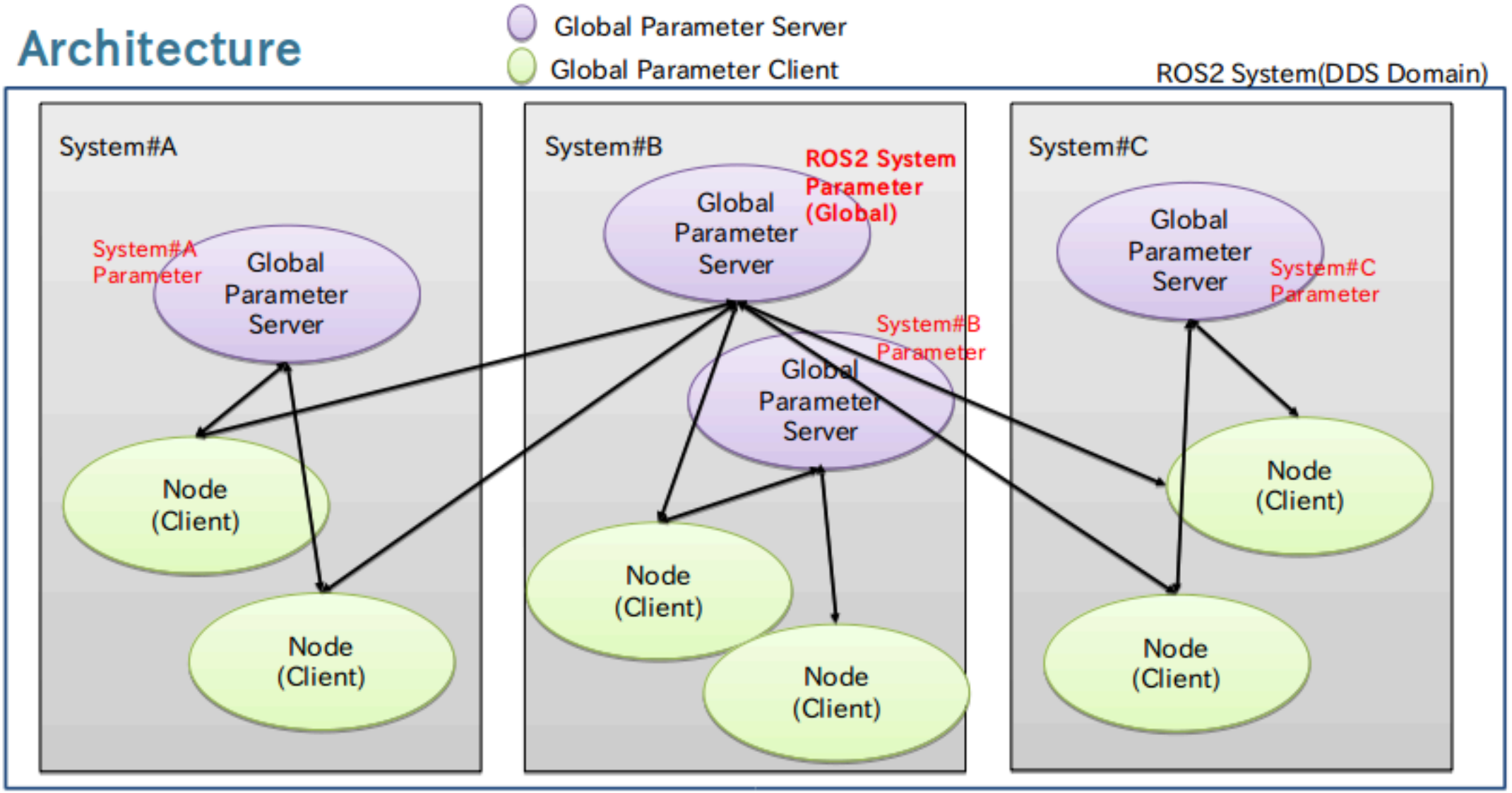
- By default, discovery traffic is only in local host system. If network communication is needed, we need to configure zenoh router and restart.
- uses CDR as the serialization format. (Compatible with DDS based RMWs)

Global Persistent Parameter Server

ROS 2 Persistent Parameter Server, that resides in the ROS 2 system to serve the parameter daemon. The other nodes(e.g the client demo provided in the code) can write/read the parameter in Parameter Server, and **Parameter Server is able to store the parameter into the persistent storage which user can specify such as tmpfs, nfs, or disk**.

see more details for https://github.com/fujitatomoya/ros2_persist_parameter_server

Architecture

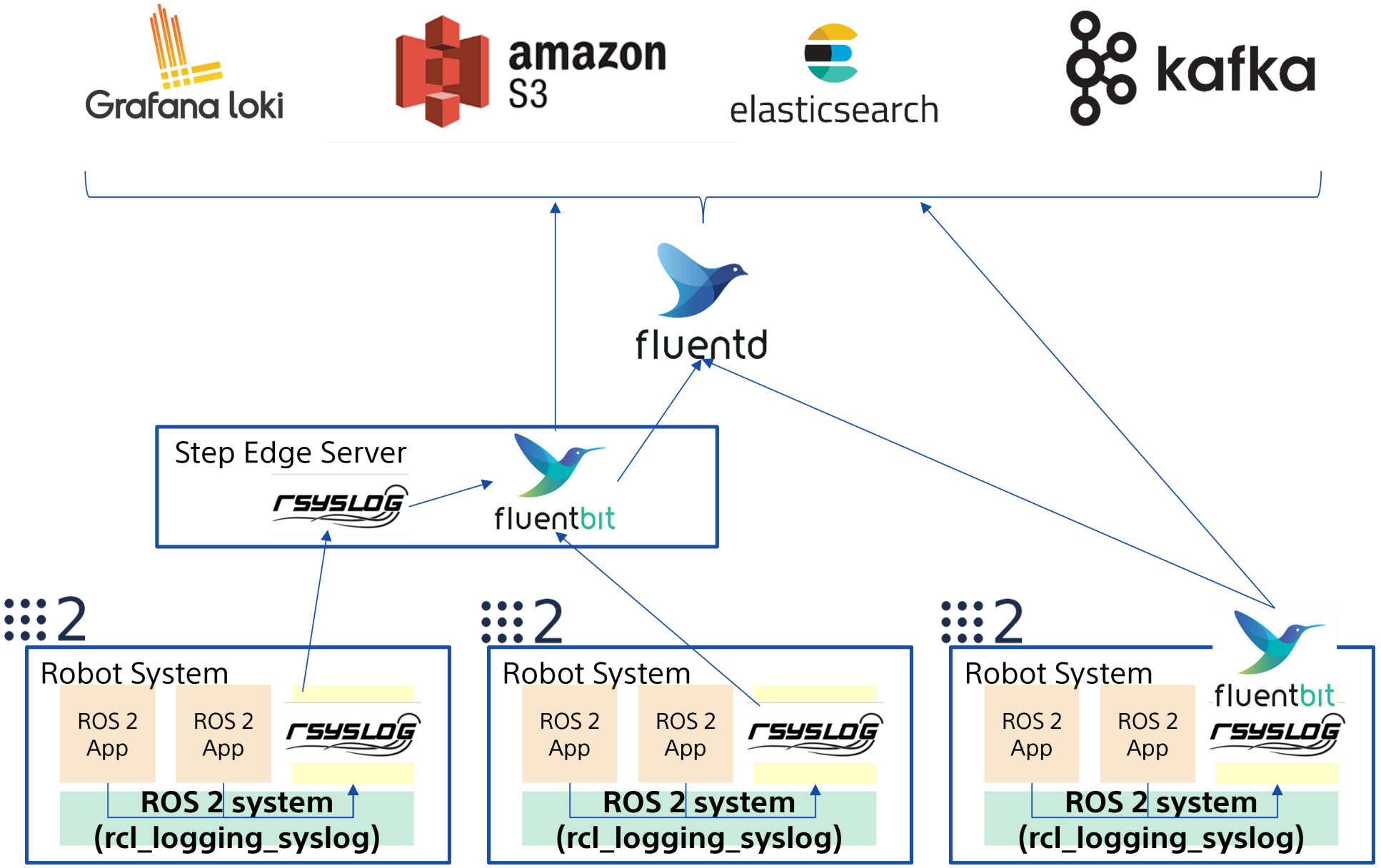


rcl_logging_syslog with FluentBit / Fluentd

rcl_logging_syslog uses SYSLOG(3) to send the log data to rsyslog a.k.a **rocket-fast system for log processing** 🚀.

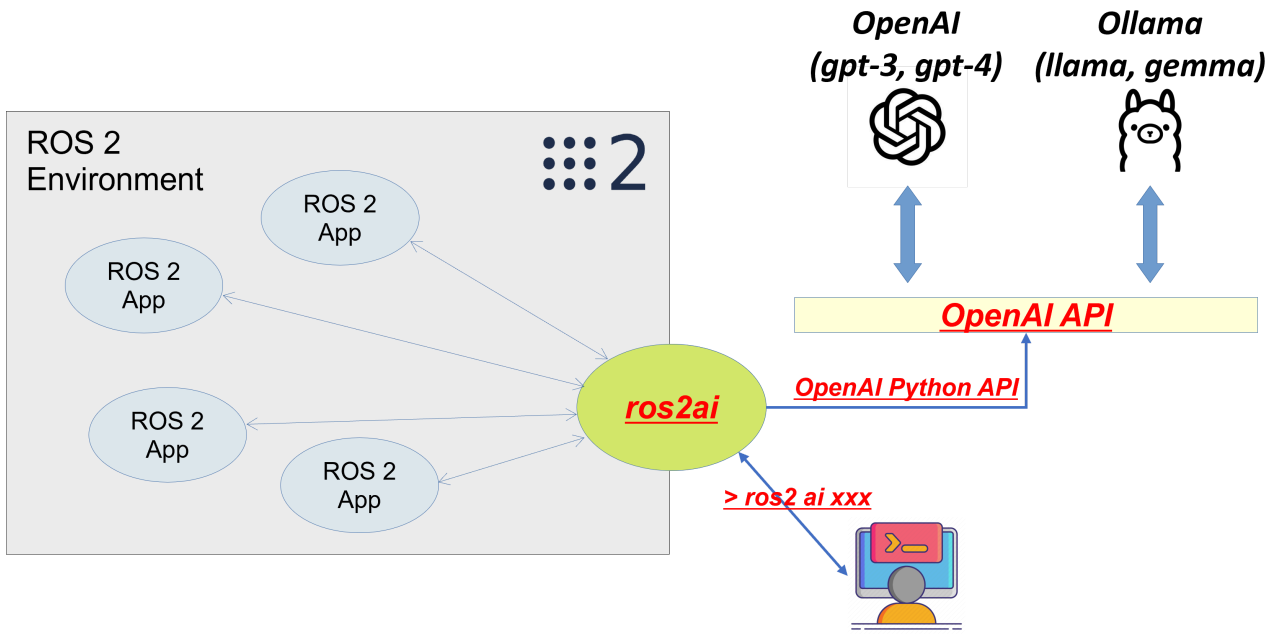
The main objective is that **Enabling ROS 2 logging system with Cloud-Native Log Management and Observability**.

see more details for https://github.com/fujitatomoya/rcl_logging_syslog


Grafana loki
amazon S3
elasticsearch
kafka
fluentd
Step Edge Server
rsyslog
fluentbit
Robot System
ROS 2 App
ROS 2 App
rsyslog
ROS 2 system
(rcl_logging_syslog)
Robot System
ROS 2 App
ROS 2 App
rsyslog
ROS 2 system
(rcl_logging_syslog)
Robot System
fluentbit
ROS 2 App
ROS 2 App
rsyslog
ROS 2 system
(rcl_logging_syslog)

ros2ai

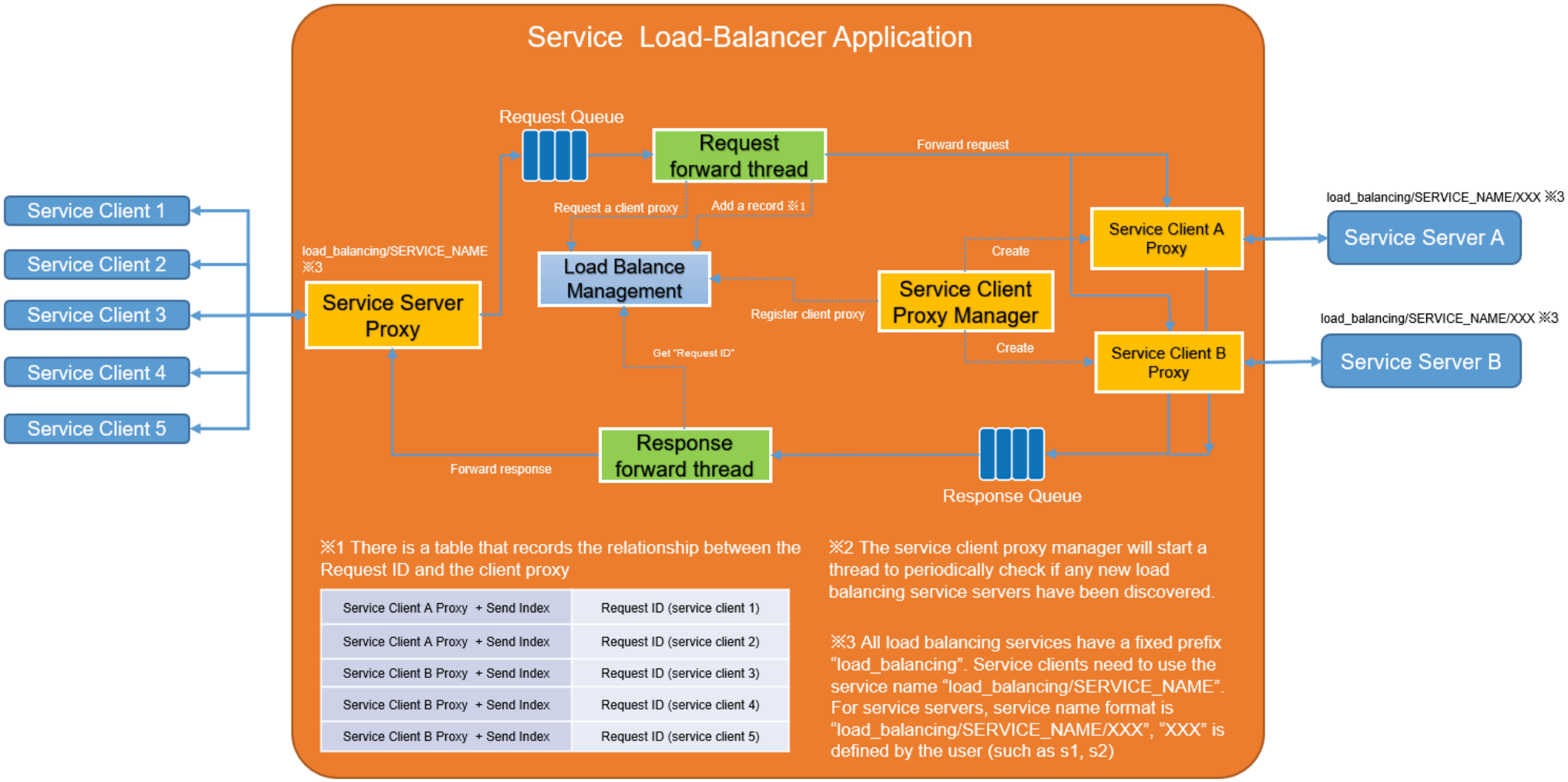
ros2ai is a next-generation ROS 2 command line interface extension with OpenAI and Ollama.



See how it works 🔥

Service Load-Balancing (PoC)

- ROS 2 service load-balancing in application layer without protocol change.
- Support multiple service servers on the same service path to have robustness and load-balancing mechanism.
- Scale / Offload ROS 2 service server/client application with remapping but code modification.



- Service Server Proxy is implemented based on rclcpp::GenericService.
- Service Client Proxy is implemented based on rclcpp::GenericClient.

Reference

- ROS 2 Documentation Top
- Governance
- Why ROS 2?
- ROS 2 Design
- Micro-ROS