ROS-Z

A Rust/Zenoh-native stack, fully ROS 2-compliant

Julien Enoch

julien.e@zettascale.tech

https://github.com/JEnoch

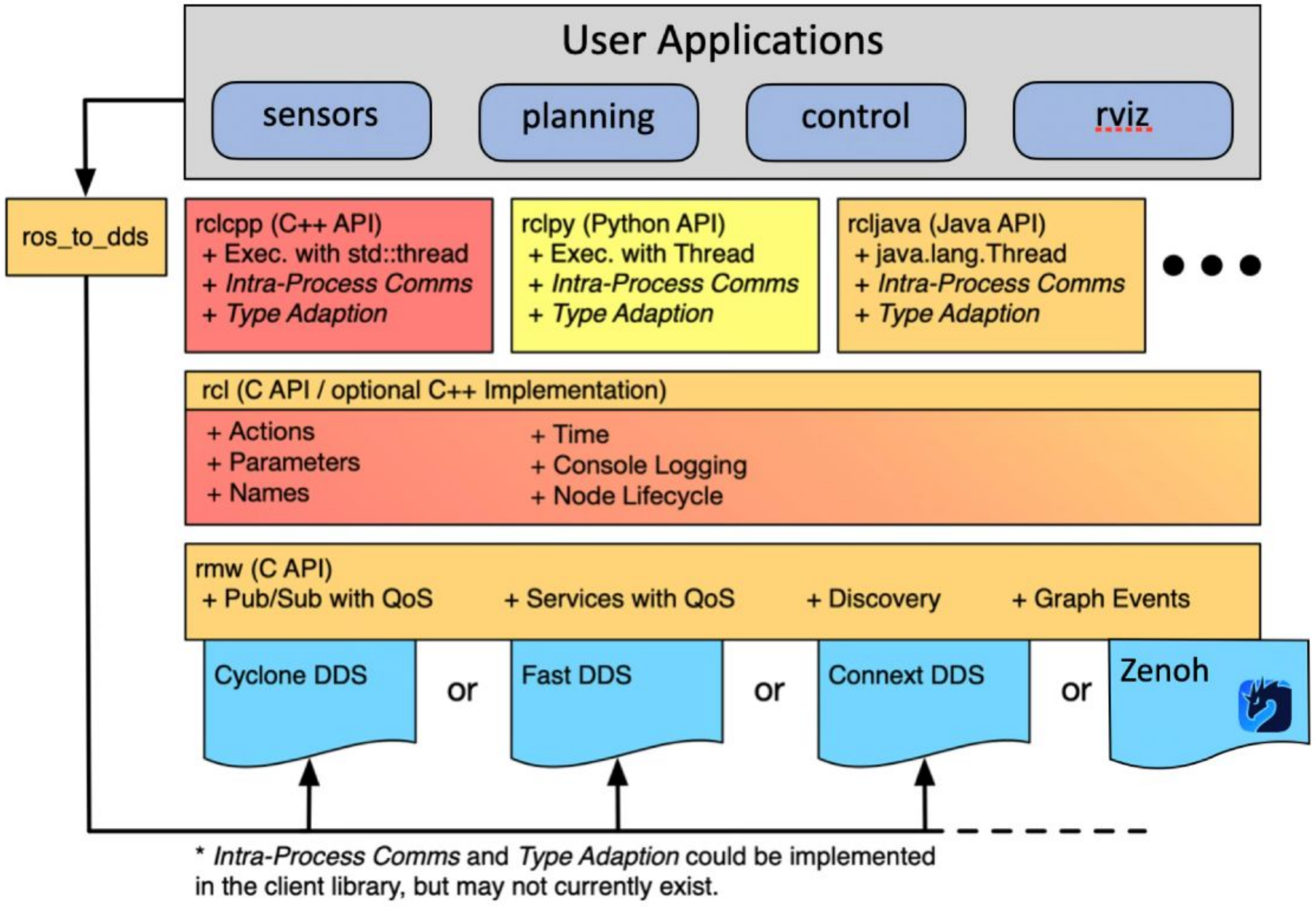
Yuyuan Yuan

yu-yuan.yuan@zettascale.tech

https://github.com/YuanYuYuan

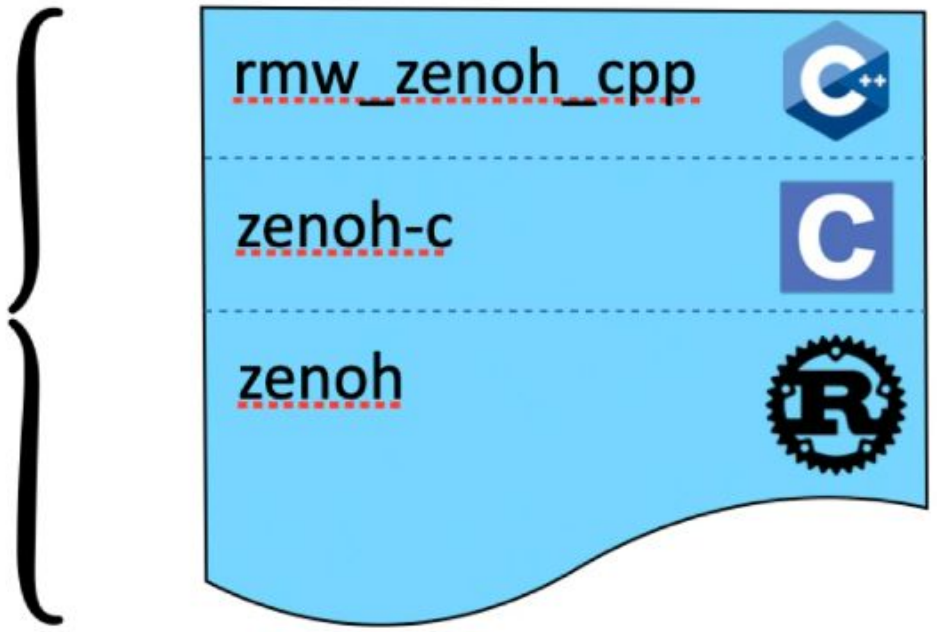
From RMW Zenoh…

RMW Zenoh



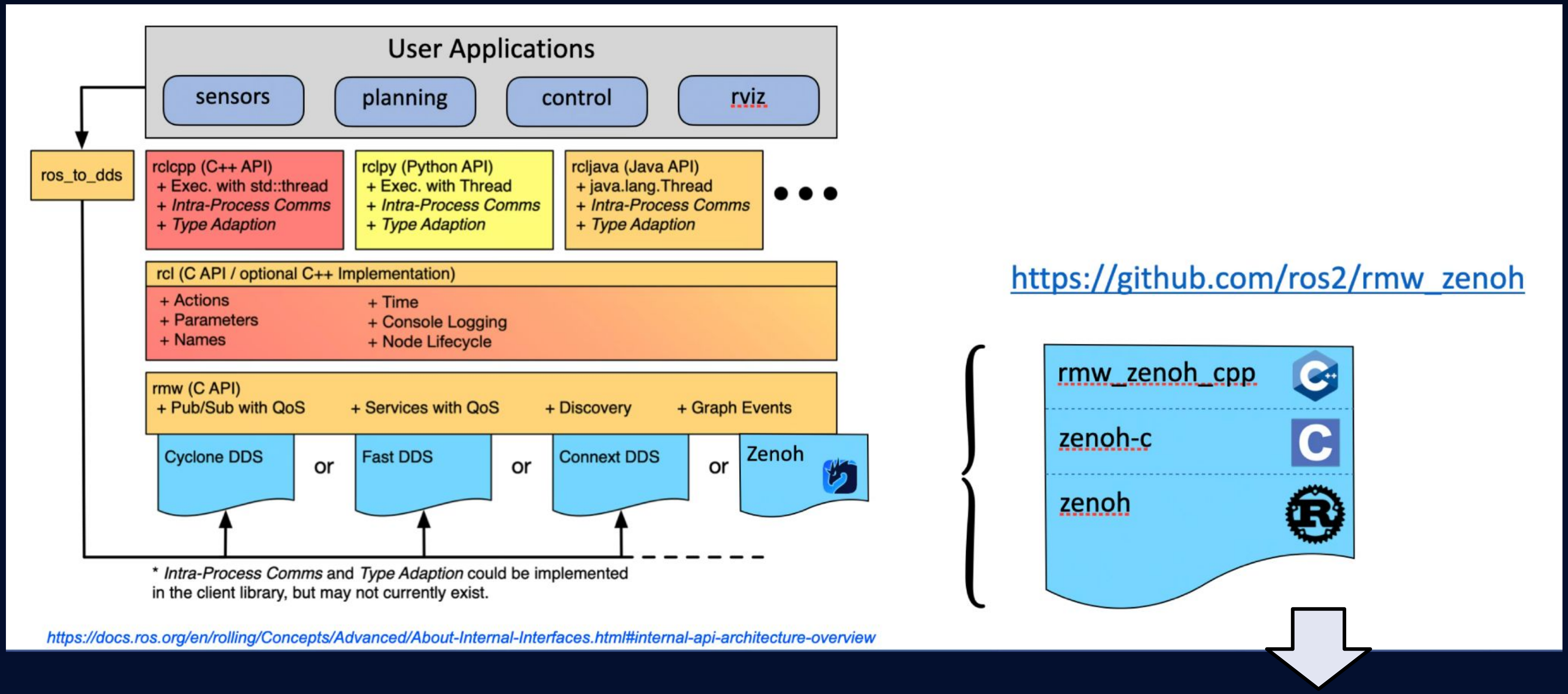
* *Intra-Process Comms* and *Type Adaption* could be implemented in the client library, but may not currently exist.

https://github.com/ros2/rmw_zenoh



https://docs.ros.org/en/rolling/Concepts/Advanced/About-Internal-Interfaces.html#internal-api-architecture-overview

RMW Zenoh - design



* *Intra-Process Comms* and *Type Adaption* could be implemented in the client library, but may not currently exist.

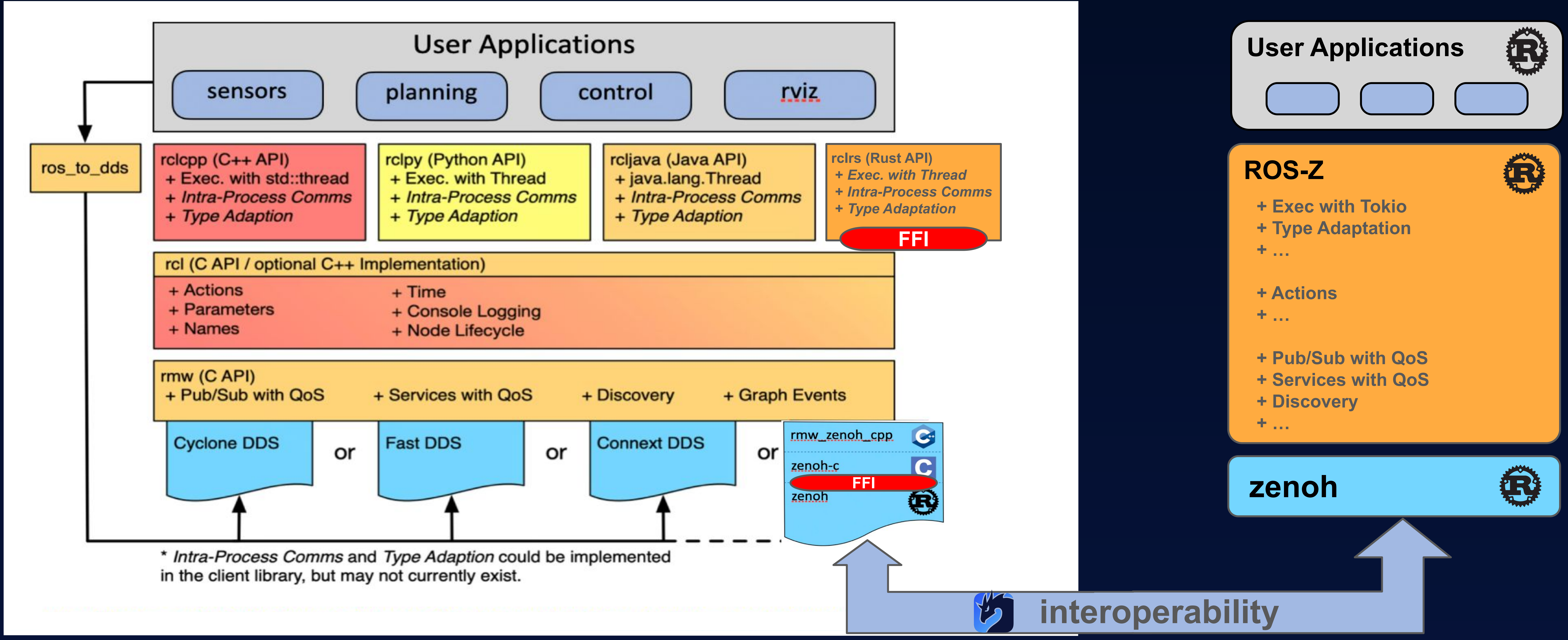
https://docs.ros.org/en/rolling/Concepts/Advanced/About-Internal-Interfaces.html#internal-api-architecture-overview

https://github.com/ros2/rmw_zenoh

ROS	Zenoh
Topics, services names	Key expression `0/chatter/std_msgs::msg::dds_::String_/RIHS01_df668...`
Graph declaration	Liveliness Token key expression `@ros2_lv/0/123/0/11/MP/%/%/talker/%chatter/std_msgs::msg::dds_::String_/RIHS01_df668…`
Messages	CDR-encoded payload + attachment (sequence number, attachment, GID)

… to ROS-Z

What about a pure-Rust stack ?



User Applications

sensors

planning

control

rviz

ros_to_dds

rclcpp (C++ API)
+ Exec. with std::thread
+ *Intra-Process Comms*
+ *Type Adaption*

rclpy (Python API)
+ Exec. with Thread
+ *Intra-Process Comms*
+ *Type Adaption*

rcljava (Java API)
+ java.lang.Thread
+ *Intra-Process Comms*
+ *Type Adaption*

rclrs (Rust API)
+ *Exec. with Thread*
+ *Intra-Process Comms*
+ *Type Adaption*

FFI

rcl (C API / optional C++ Implementation)
+ Actions
+ Parameters
+ Names
+ Time
+ Console Logging
+ Node Lifecycle

rmw (C API)
+ Pub/Sub with QoS
+ Services with QoS
+ Discovery
+ Graph Events

Cyclone DDS

or

Fast DDS

or

Connext DDS

or

rmw_zenoh_cpp

zenoh-c

FFI

zenoh

* *Intra-Process Comms* and *Type Adaption* could be implemented
in the client library, but may not currently exist.

User Applications

ROS-Z
+ Exec with Tokio
+ Type Adaptation
+ …

+ Actions
+ …

+ Pub/Sub with QoS
+ Services with QoS
+ Discovery
+ …

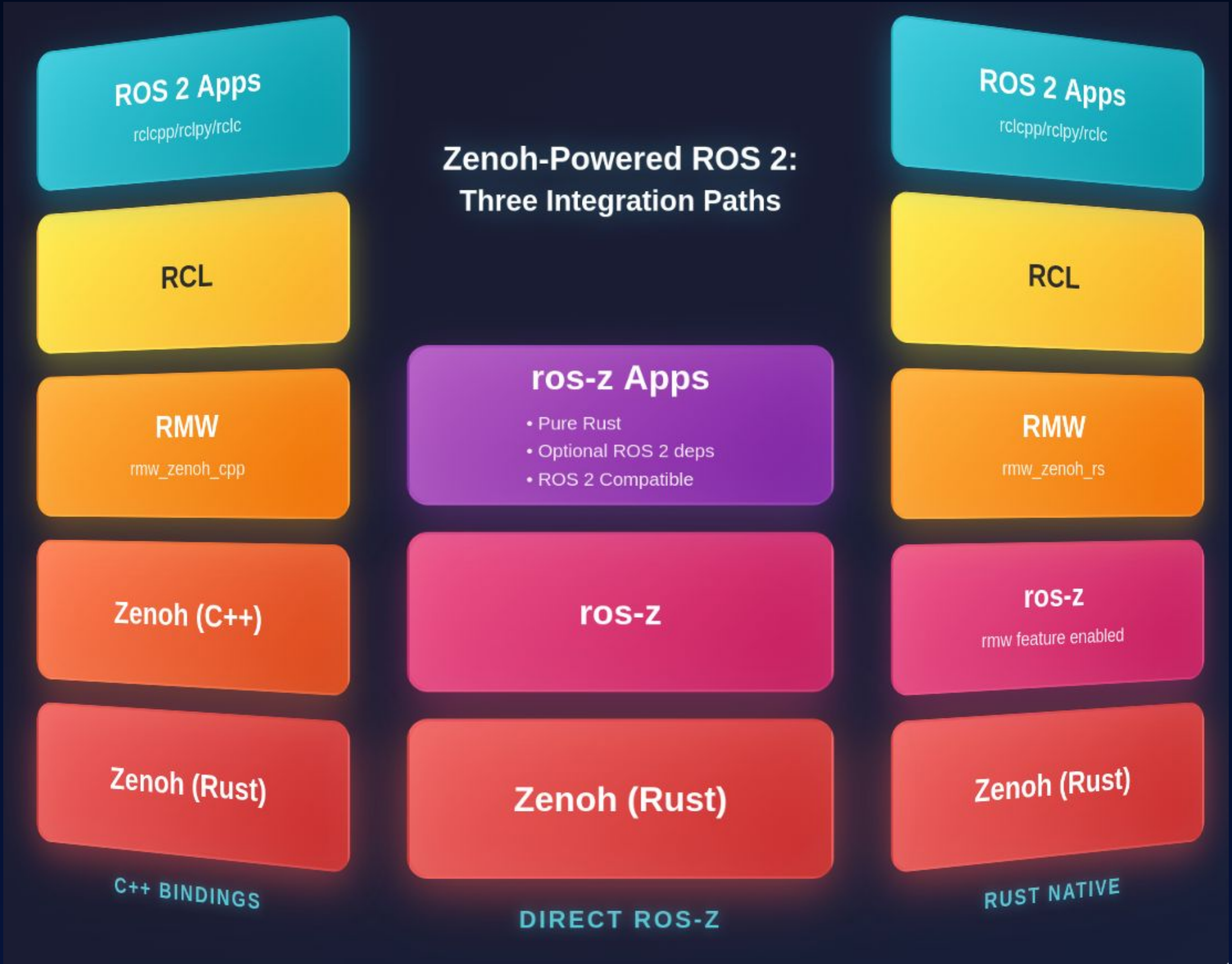
zenoh

interoperability

ROS-Z main goals

- **100% Rust**
  - Builder patterns for evolutivity with backward compatibility
  - Cargo build
- **Rust-native ROS 2 abstractions**
  - Pub/Sub, Services, Actions
  - Graph discovery
- **Battery include: Zero ROS 2 dependencies required**
  - Universal compatibility across Rust platforms
- **Interoperability with RMW Zenoh**
  - But also supporting alternative encodings (protobuf…)

ROS-Z dive in



Zenoh-Powered ROS 2: Three Integration Paths

C++ BINDINGS

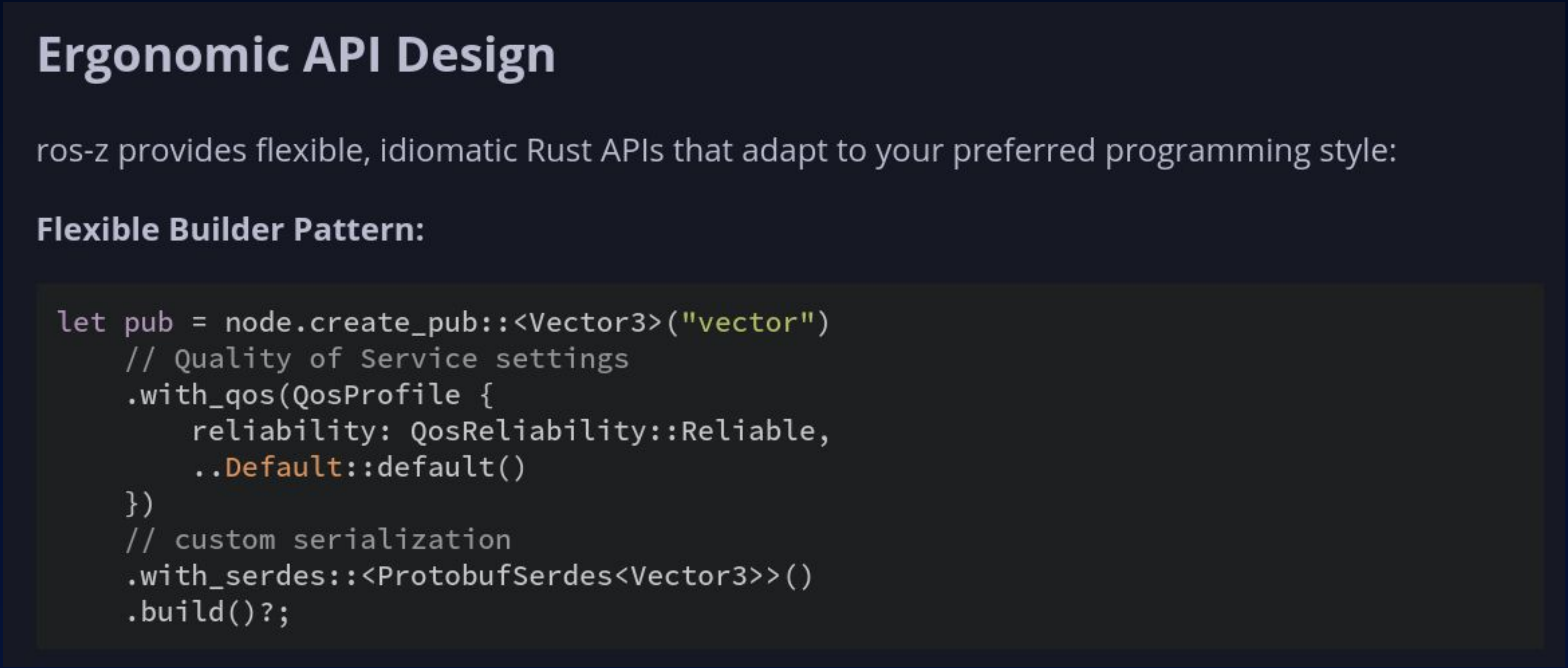
- ROS 2 Apps — rclcpp/rclpy/rclc
- RCL
- RMW — rmw_zenoh_cpp
- Zenoh (C++)
- Zenoh (Rust)

DIRECT ROS-Z

- ros-z Apps
  - Pure Rust
  - Optional ROS 2 deps
  - ROS 2 Compatible
- ros-z
- Zenoh (Rust)

RUST NATIVE

- ROS 2 Apps — rclcpp/rclpy/rclc
- RCL
- RMW — rmw_zenoh_rs
- ros-z — rmw feature enabled
- Zenoh (Rust)

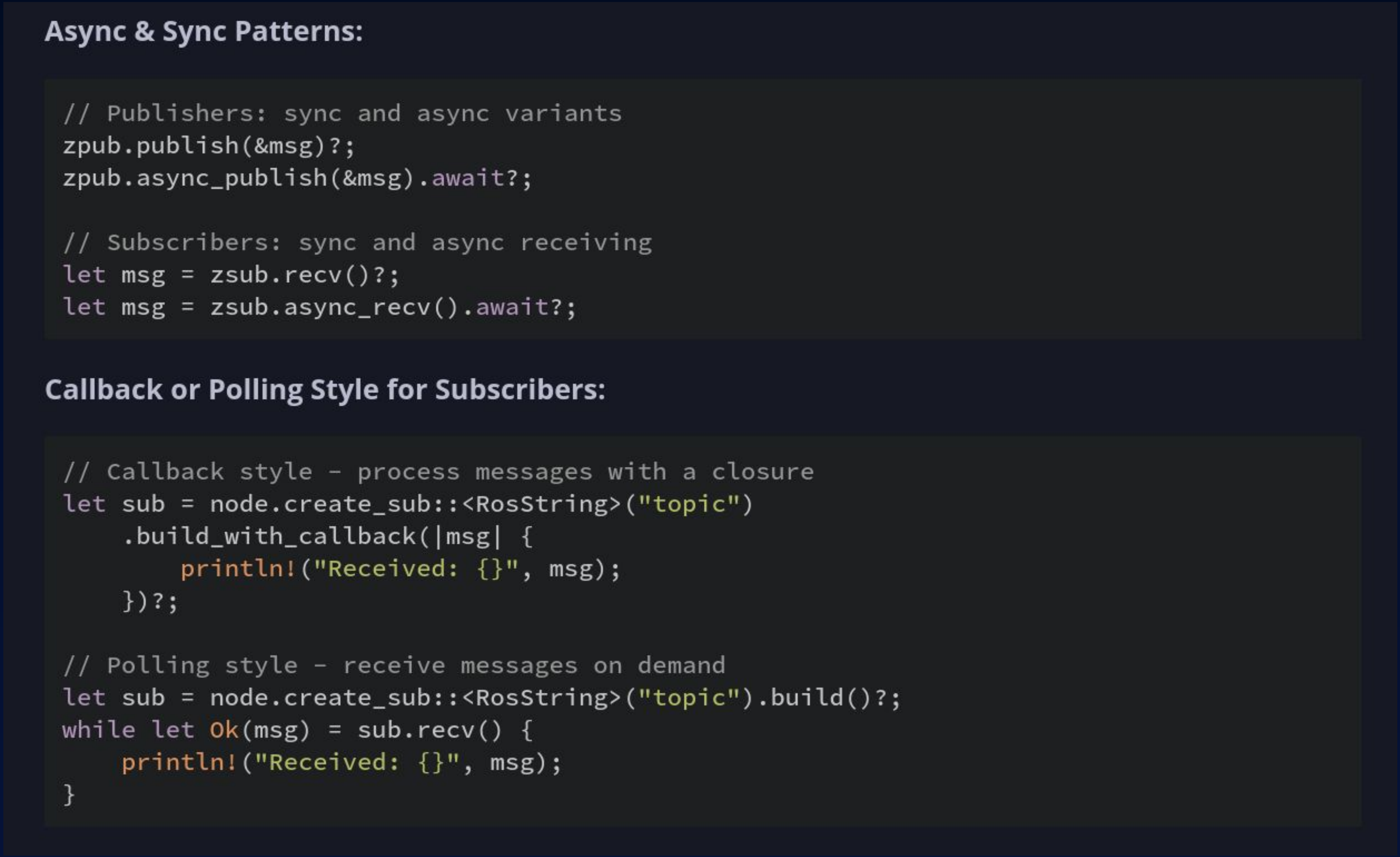


Ergonomic API Design

ros-z provides flexible, idiomatic Rust APIs that adapt to your preferred programming style:

Flexible Builder Pattern:

```rust
let pub = node.create_pub::<Vector3>("vector")
    // Quality of Service settings
    .with_qos(QosProfile {
        reliability: QosReliability::Reliable,
        ..Default::default()
    })
    // custom serialization
    .with_serdes::<ProtobufSerdes<Vector3>>()
    .build()?;
```

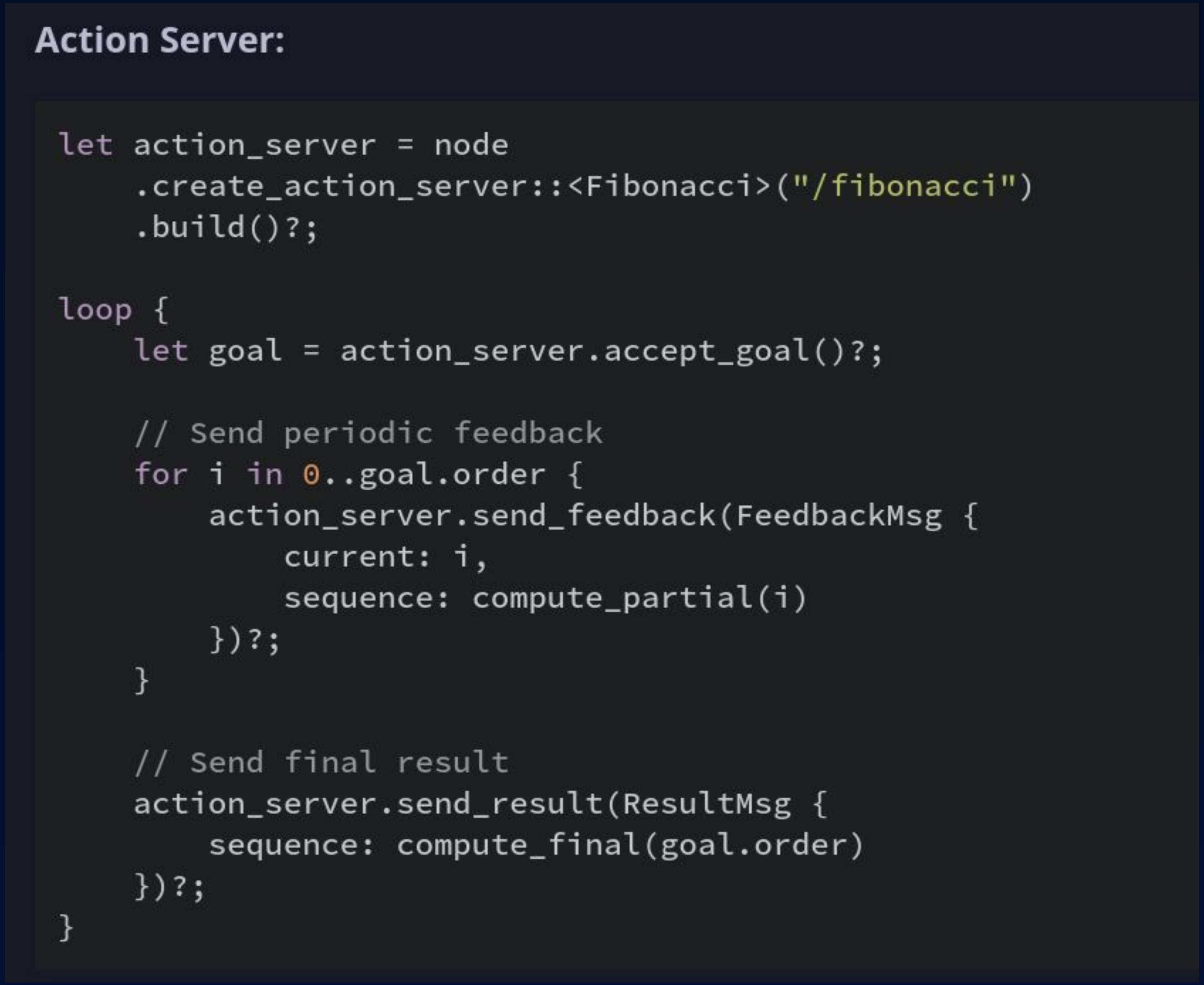


Async & Sync Patterns:

```
// Publishers: sync and async variants
zpub.publish(&msg)?;
zpub.async_publish(&msg).await?;

// Subscribers: sync and async receiving
let msg = zsub.recv()?;
let msg = zsub.async_recv().await?;
```

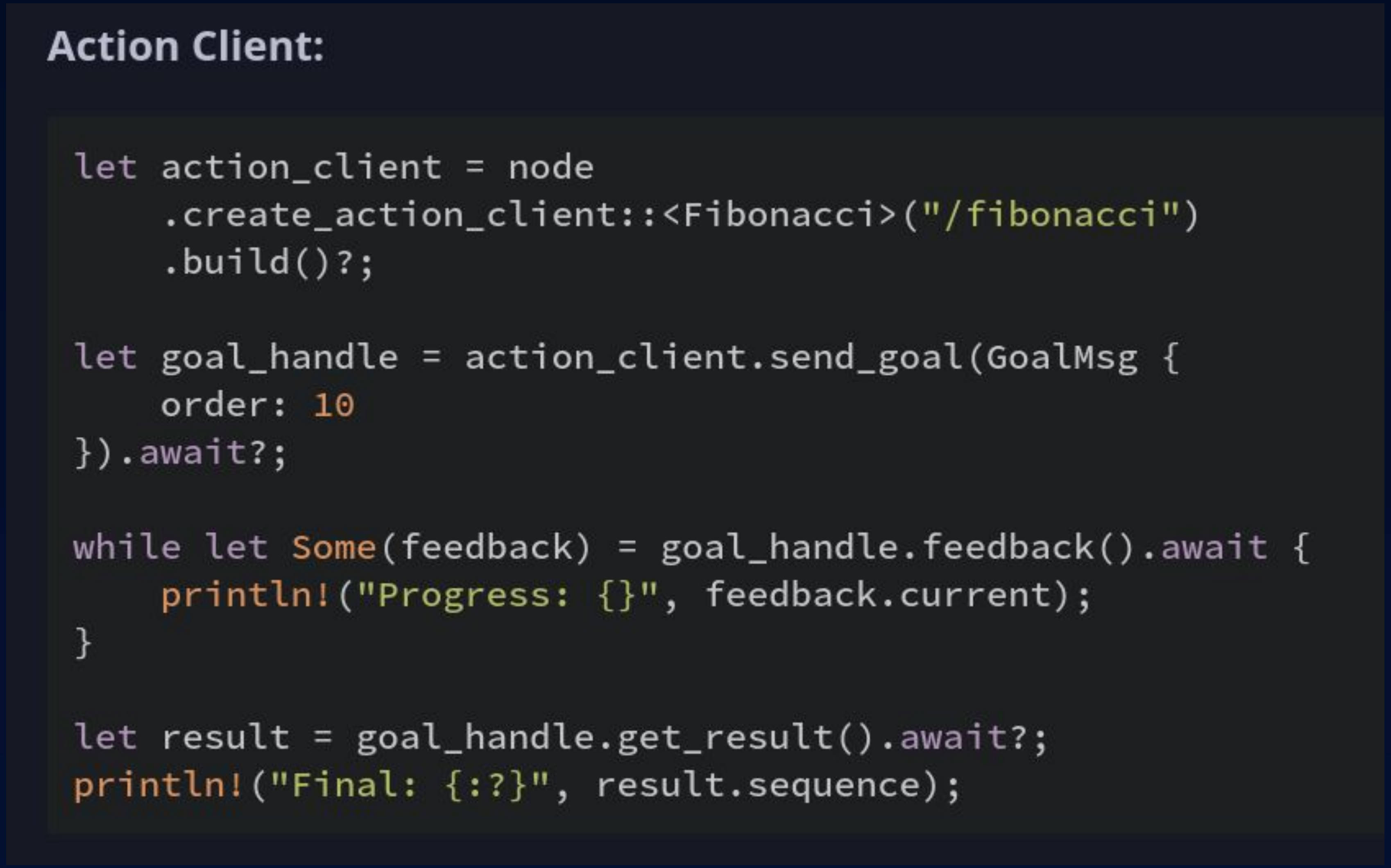
Callback or Polling Style for Subscribers:

```
// Callback style - process messages with a closure
let sub = node.create_sub::<RosString>("topic")
    .build_with_callback(|msg| {
        println!("Received: {}", msg);
    })?;

// Polling style - receive messages on demand
let sub = node.create_sub::<RosString>("topic").build()?;
while let Ok(msg) = sub.recv() {
    println!("Received: {}", msg);
}
```



Action Server:

```
let action_server = node
    .create_action_server::<Fibonacci>("/fibonacci")
    .build()?;

loop {
    let goal = action_server.accept_goal()?;

    // Send periodic feedback
    for i in 0..goal.order {
        action_server.send_feedback(FeedbackMsg {
            current: i,
            sequence: compute_partial(i)
        })?;
    }

    // Send final result
    action_server.send_result(ResultMsg {
        sequence: compute_final(goal.order)
    })?;
}
```

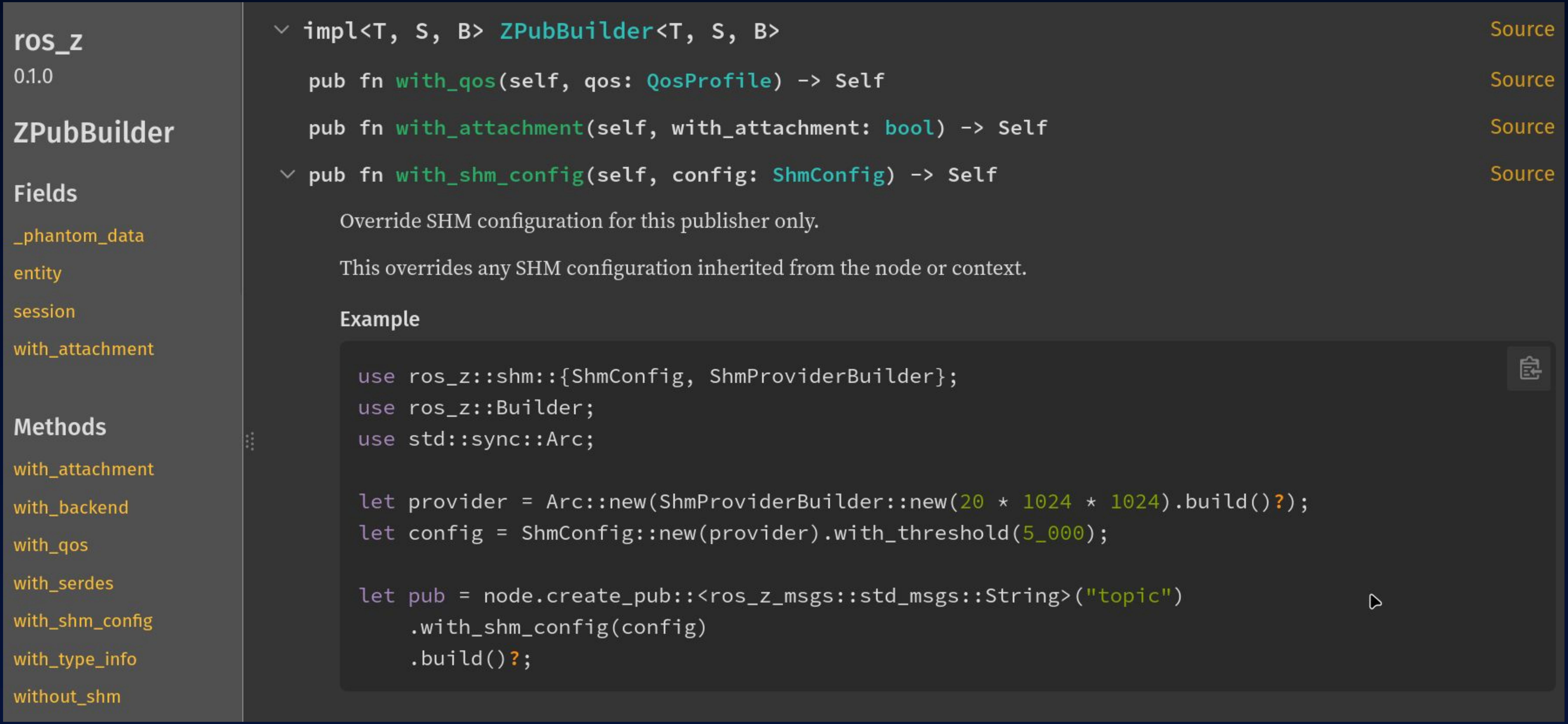


Action Client:

```
let action_client = node
    .create_action_client::<Fibonacci>("/fibonacci")
    .build()?;

let goal_handle = action_client.send_goal(GoalMsg {
    order: 10
}).await?;

while let Some(feedback) = goal_handle.feedback().await {
    println!("Progress: {}", feedback.current);
}

let result = goal_handle.get_result().await?;
println!("Final: {:?}", result.sequence);
```

- **Easy to use**
- **Compared to RCLCPP API**

Cargo Doc



ros_z
0.1.0

ZPubBuilder

Fields

_phantom_data
entity
session
with_attachment

Methods

with_attachment
with_backend
with_qos
with_serdes
with_shm_config
with_type_info
without_shm

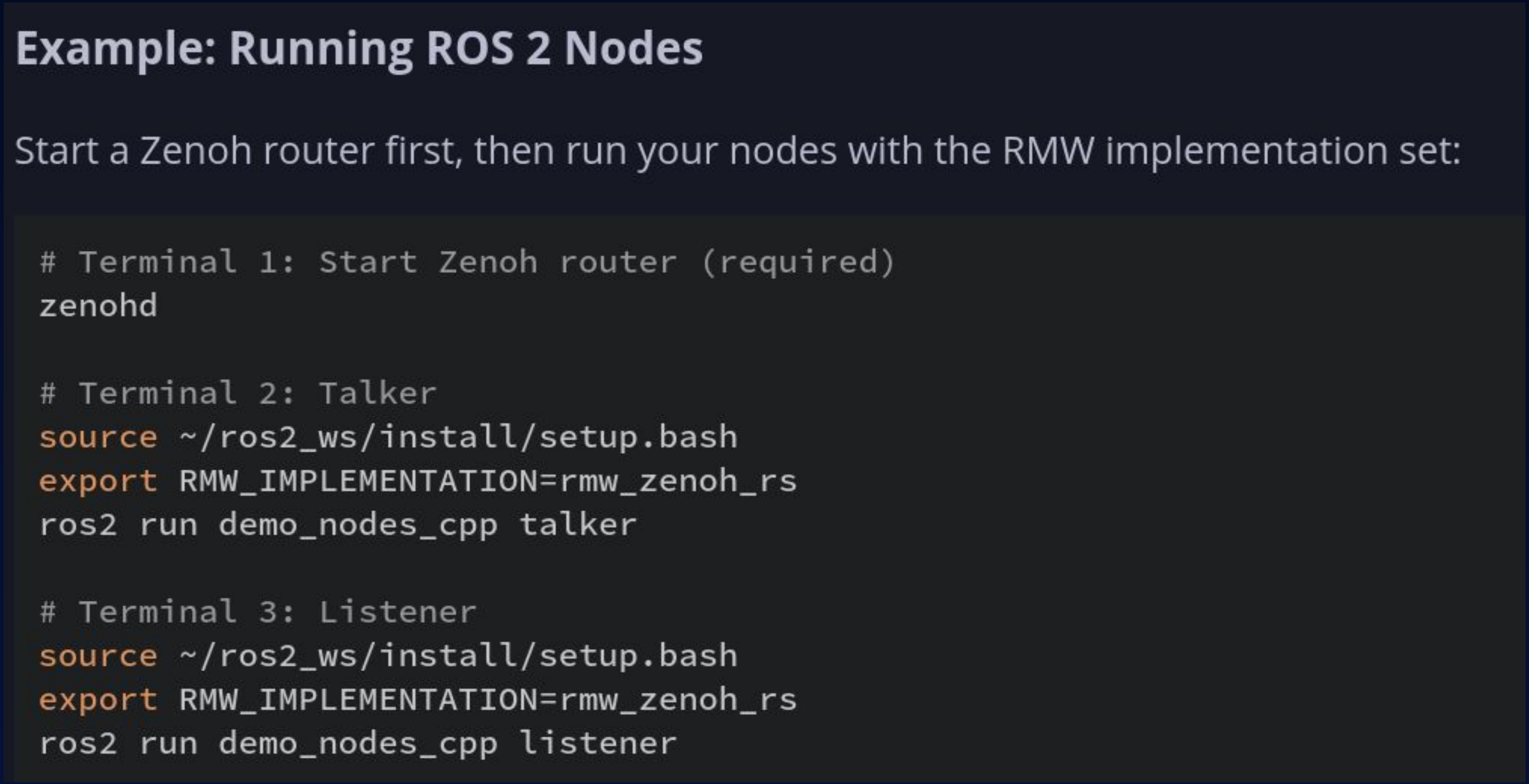
```
impl<T, S, B> ZPubBuilder<T, S, B>                                      Source

    pub fn with_qos(self, qos: QosProfile) -> Self                      Source

    pub fn with_attachment(self, with_attachment: bool) -> Self         Source

    pub fn with_shm_config(self, config: ShmConfig) -> Self             Source
```

Override SHM configuration for this publisher only.

This overrides any SHM configuration inherited from the node or context.

Example

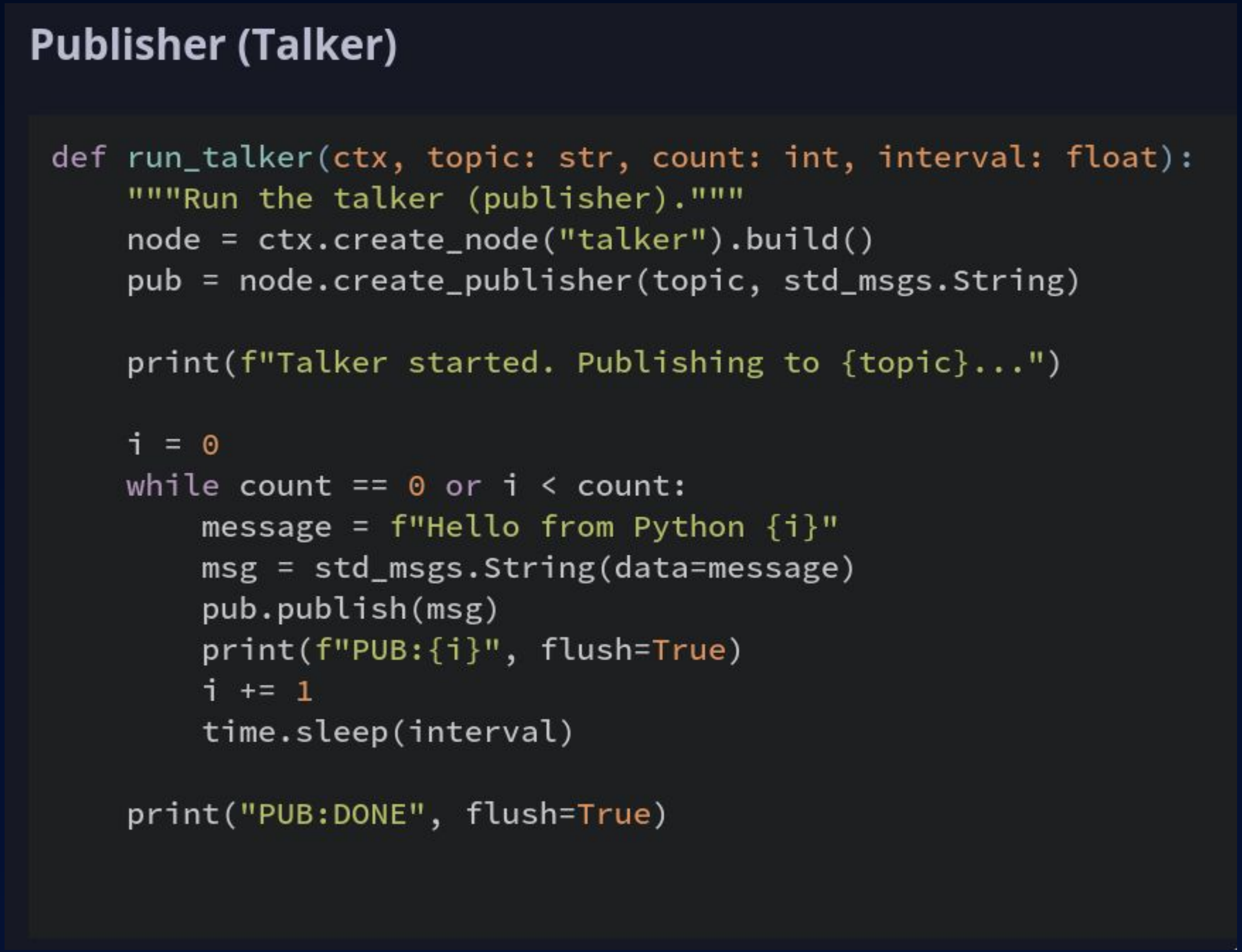
```rust
use ros_z::shm::{ShmConfig, ShmProviderBuilder};
use ros_z::Builder;
use std::sync::Arc;

let provider = Arc::new(ShmProviderBuilder::new(20 * 1024 * 1024).build()?);
let config = ShmConfig::new(provider).with_threshold(5_000);

let pub = node.create_pub::<ros_z_msgs::std_msgs::String>("topic")
    .with_shm_config(config)
    .build()?;
```



Example: Running ROS 2 Nodes

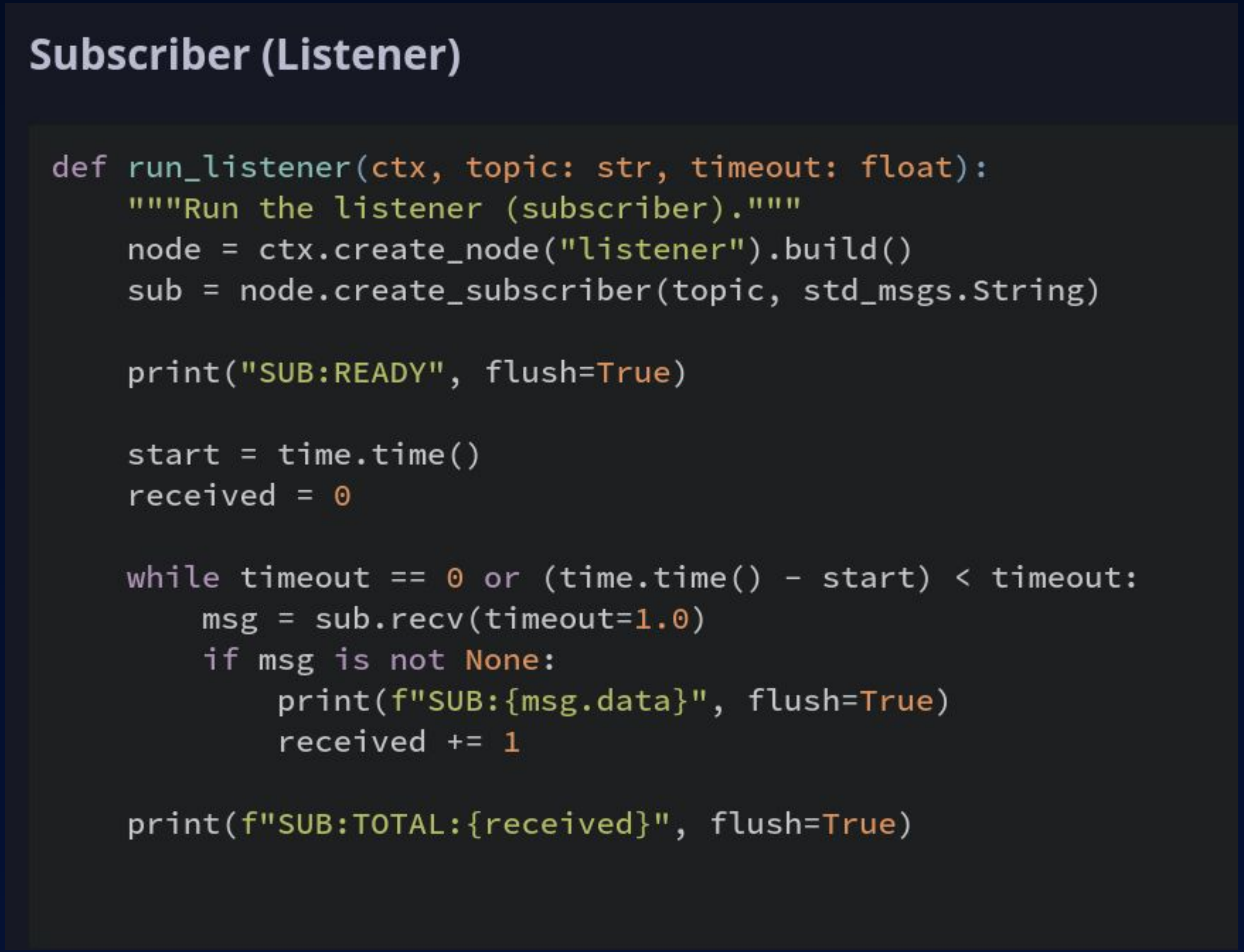
Start a Zenoh router first, then run your nodes with the RMW implementation set:

```bash
# Terminal 1: Start Zenoh router (required)
zenohd

# Terminal 2: Talker
source ~/ros2_ws/install/setup.bash
export RMW_IMPLEMENTATION=rmw_zenoh_rs
ros2 run demo_nodes_cpp talker

# Terminal 3: Listener
source ~/ros2_ws/install/setup.bash
export RMW_IMPLEMENTATION=rmw_zenoh_rs
ros2 run demo_nodes_cpp listener
```



Publisher (Talker)

```python
def run_talker(ctx, topic: str, count: int, interval: float):
    """Run the talker (publisher)."""
    node = ctx.create_node("talker").build()
    pub = node.create_publisher(topic, std_msgs.String)

    print(f"Talker started. Publishing to {topic}...")

    i = 0
    while count == 0 or i < count:
        message = f"Hello from Python {i}"
        msg = std_msgs.String(data=message)
        pub.publish(msg)
        print(f"PUB:{i}", flush=True)
        i += 1
        time.sleep(interval)

    print("PUB:DONE", flush=True)
```

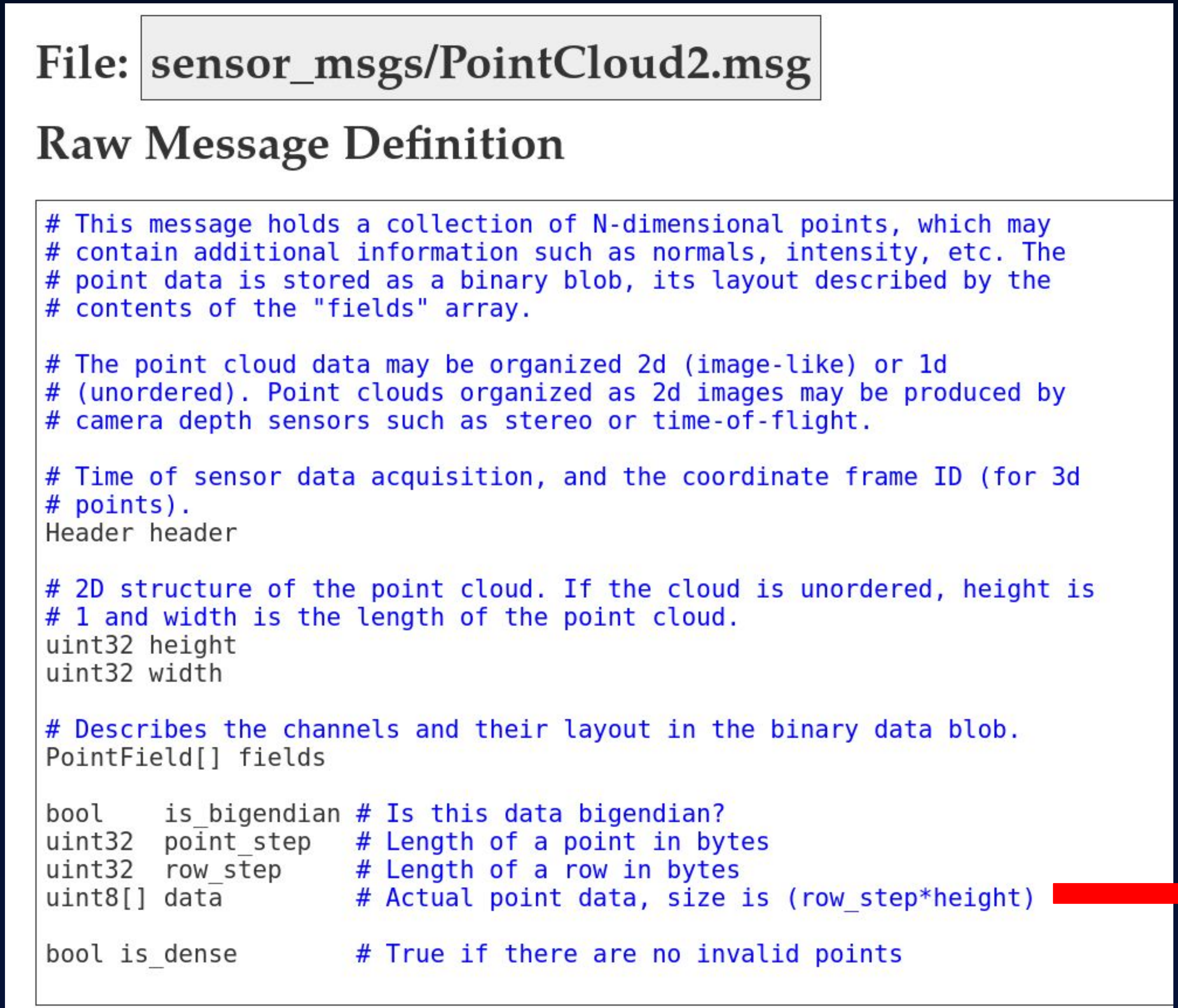


Subscriber (Listener)

```python
def run_listener(ctx, topic: str, timeout: float):
    """Run the listener (subscriber)."""
    node = ctx.create_node("listener").build()
    sub = node.create_subscriber(topic, std_msgs.String)

    print("SUB:READY", flush=True)

    start = time.time()
    received = 0

    while timeout == 0 or (time.time() - start) < timeout:
        msg = sub.recv(timeout=1.0)
        if msg is not None:
            print(f"SUB:{msg.data}", flush=True)
            received += 1

    print(f"SUB:TOTAL:{received}", flush=True)
```

Python API is supported

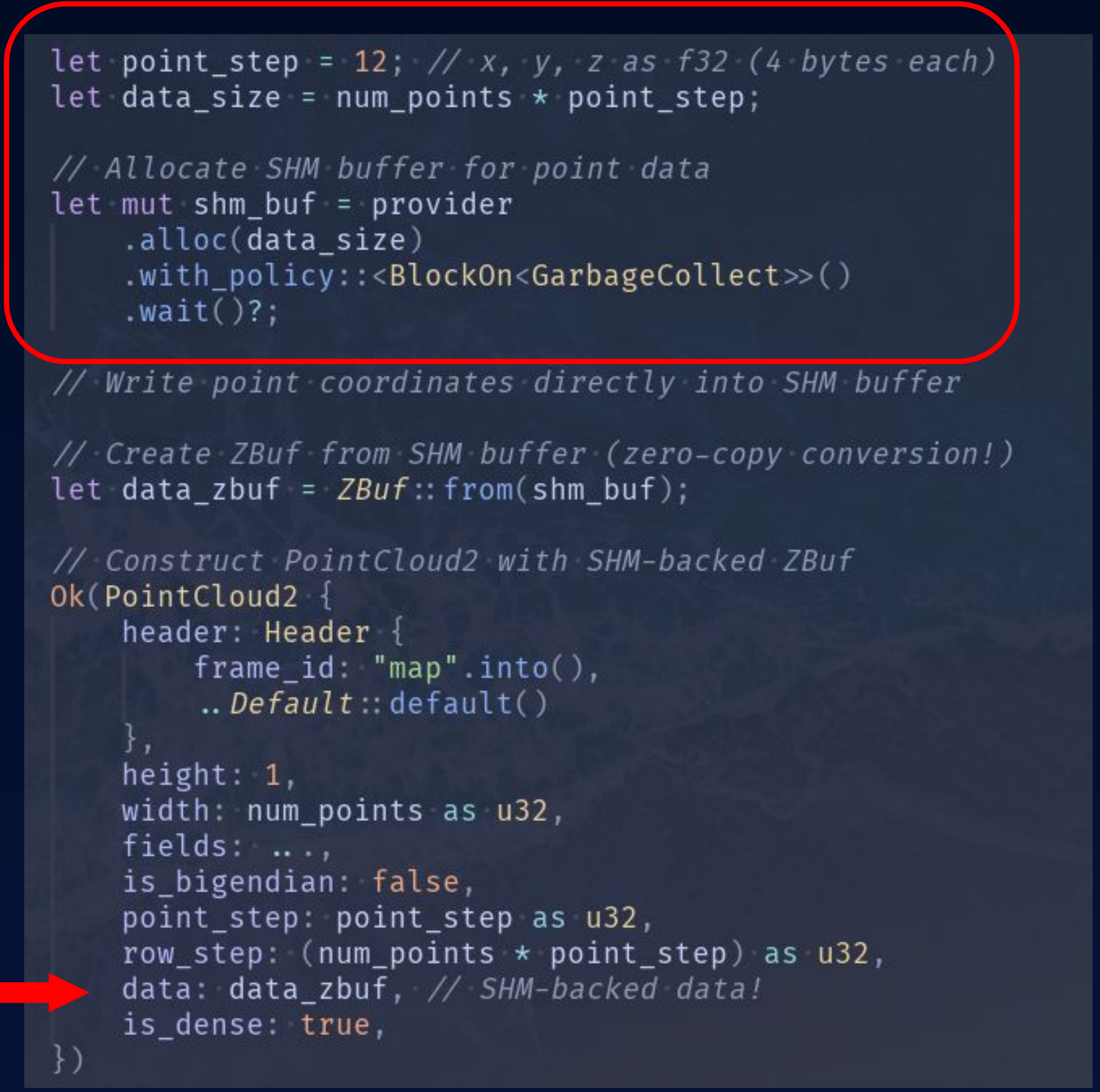
Mixed memory message is possible



File: sensor_msgs/PointCloud2.msg

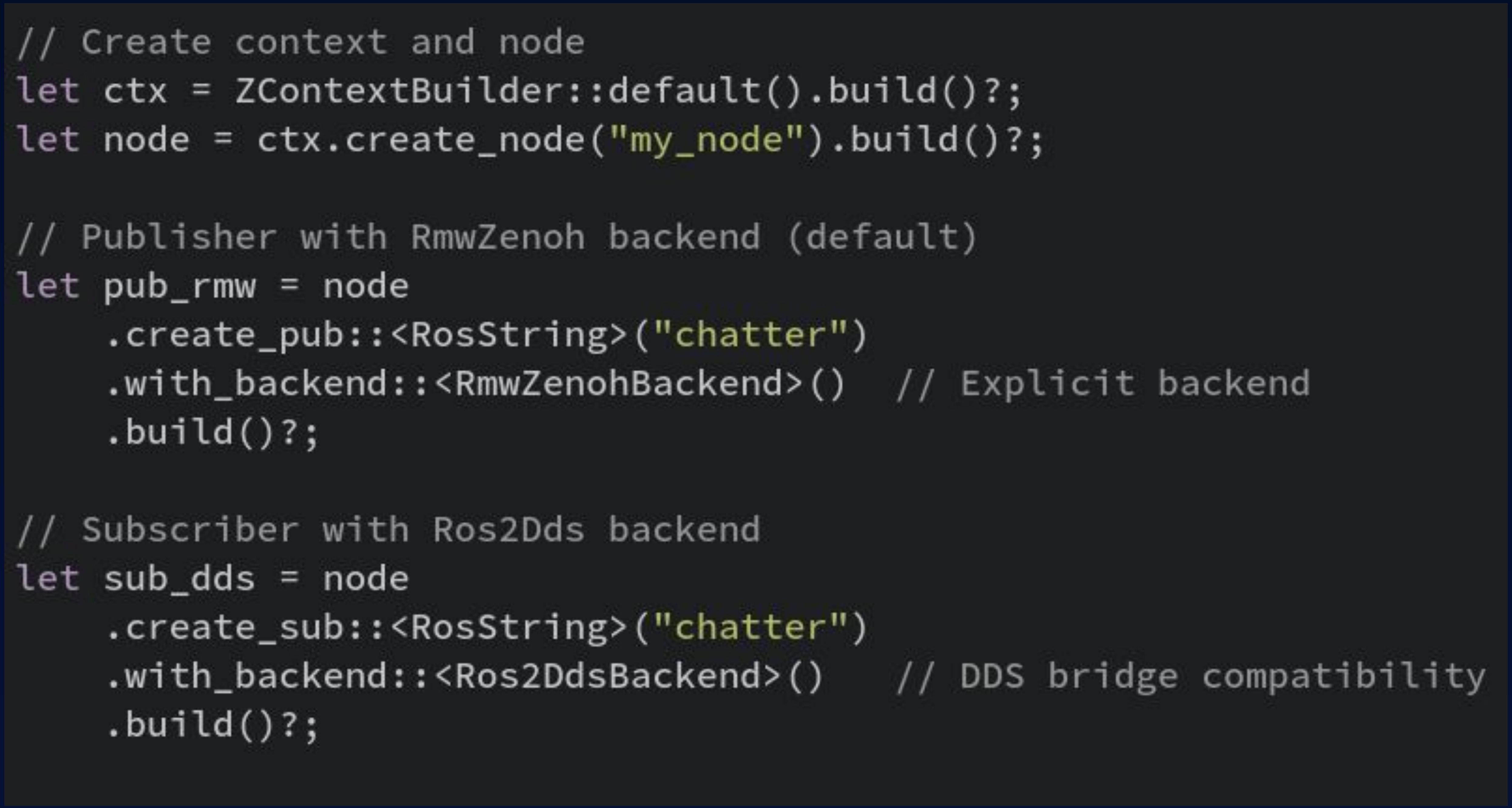
Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian # Is this data bigendian?
uint32  point_step   # Length of a point in bytes
uint32  row_step     # Length of a row in bytes
uint8[] data         # Actual point data, size is (row_step*height)

bool is_dense        # True if there are no invalid points
```



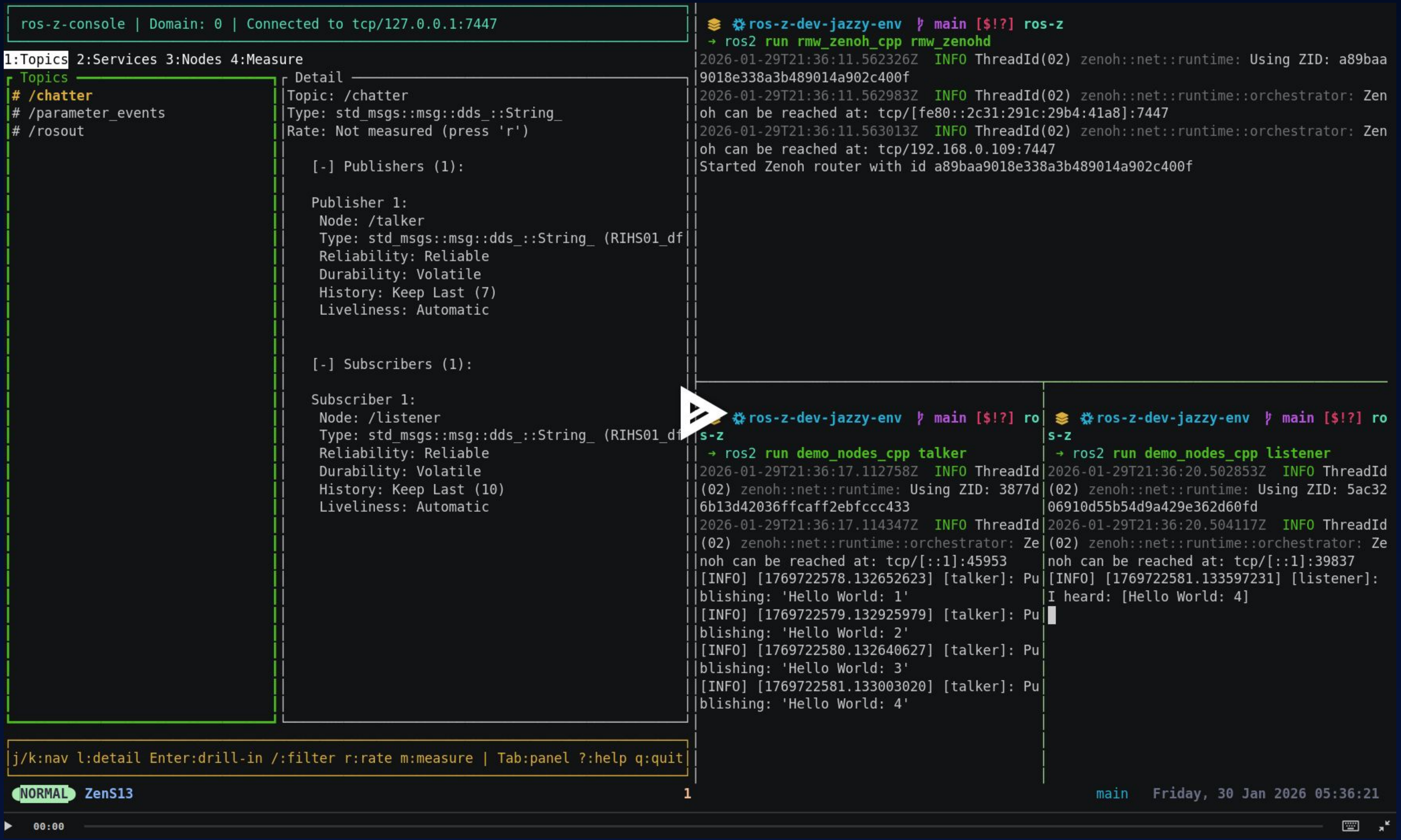
```
let point_step = 12; // x, y, z as f32 (4 bytes each)
let data_size = num_points * point_step;

// Allocate SHM buffer for point data
let mut shm_buf = provider
    .alloc(data_size)
    .with_policy::<BlockOn<GarbageCollect>>()
    .wait()?;
// Write point coordinates directly into SHM buffer

// Create ZBuf from SHM buffer (zero-copy conversion!)
let data_zbuf = ZBuf::from(shm_buf);

// Construct PointCloud2 with SHM-backed ZBuf
Ok(PointCloud2 {
    header: Header {
        frame_id: "map".into(),
        ..Default::default()
    },
    height: 1,
    width: num_points as u32,
    fields: ...,
    is_bigendian: false,
    point_step: point_step as u32,
    row_step: (num_points * point_step) as u32,
    data: data_zbuf, // SHM-backed data!
    is_dense: true,
})
```

Support rmw_zenoh & zenoh-bridge-ros2dds



```
// Create context and node
let ctx = ZContextBuilder::default().build()?;
let node = ctx.create_node("my_node").build()?;

// Publisher with RmwZenoh backend (default)
let pub_rmw = node
    .create_pub::<RosString>("chatter")
    .with_backend::<RmwZenohBackend>()  // Explicit backend
    .build()?;

// Subscriber with Ros2Dds backend
let sub_dds = node
    .create_sub::<RosString>("chatter")
    .with_backend::<Ros2DdsBackend>()   // DDS bridge compatibility
    .build()?;
```

ros-z-console



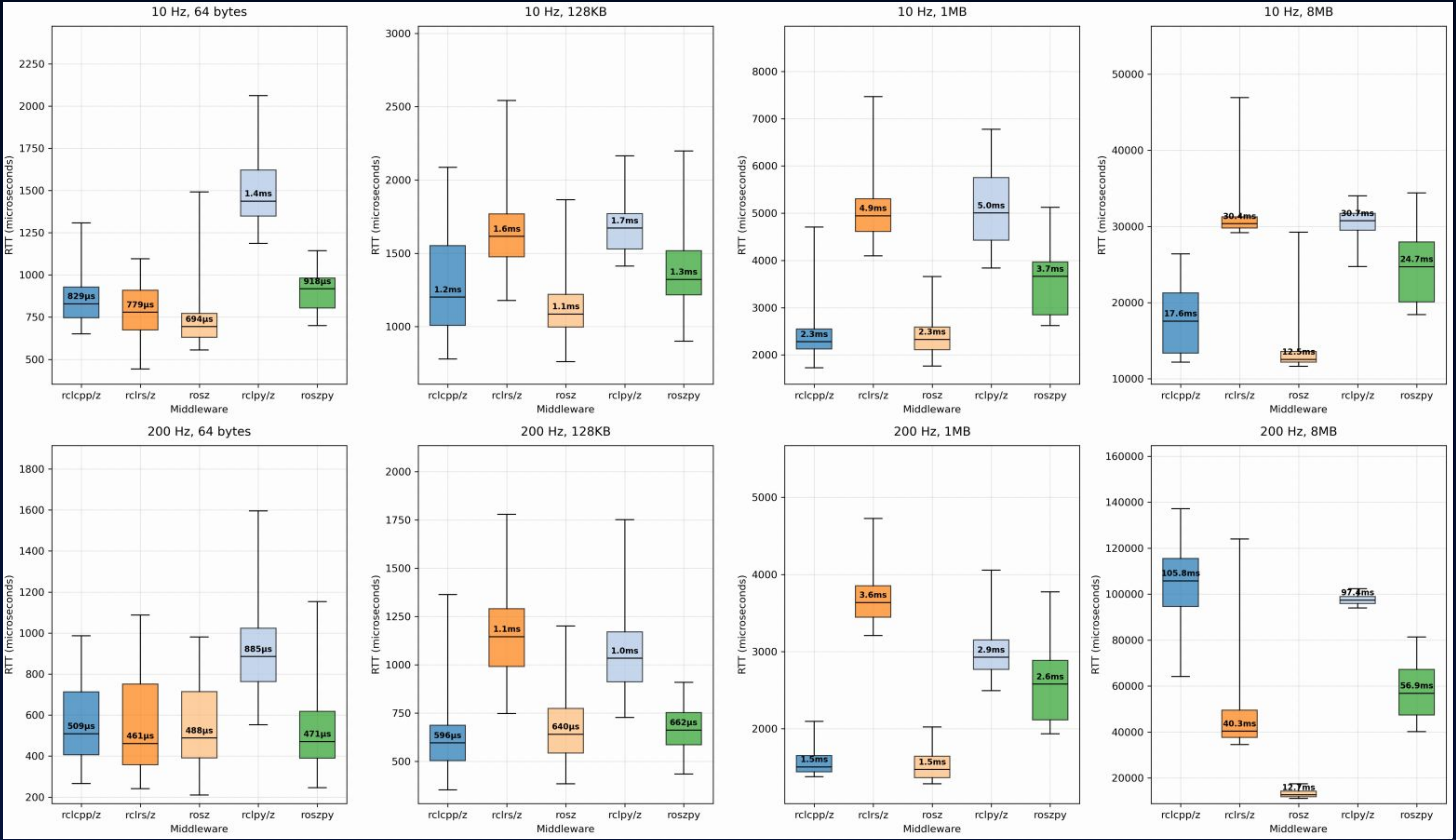
```
ros-z-console | Domain: 0 | Connected to tcp/127.0.0.1:7447

1:Topics 2:Services 3:Nodes 4:Measure
┌ Topics ─────────────┐┌ Detail ──────────────────────────────────────
│# /chatter           ││Topic: /chatter
│# /parameter_events  ││Type: std_msgs::msg::dds_::String_
│# /rosout            ││Rate: Not measured (press 'r')
│                     ││
│                     ││   [-] Publishers (1):
│                     ││
│                     ││   Publisher 1:
│                     ││    Node: /talker
│                     ││    Type: std_msgs::msg::dds_::String_ (RIHS01_df
│                     ││    Reliability: Reliable
│                     ││    Durability: Volatile
│                     ││    History: Keep Last (7)
│                     ││    Liveliness: Automatic
│                     ││
│                     ││
│                     ││   [-] Subscribers (1):
│                     ││
│                     ││   Subscriber 1:
│                     ││    Node: /listener
│                     ││    Type: std_msgs::msg::dds_::String_ (RIHS01_df
│                     ││    Reliability: Reliable
│                     ││    Durability: Volatile
│                     ││    History: Keep Last (10)
│                     ││    Liveliness: Automatic

j/k:nav l:detail Enter:drill-in /:filter r:rate m:measure | Tab:panel ?:help q:quit
```

```
ros-z-dev-jazzy-env  main [$!?] ros-z
→ ros2 run rmw_zenoh_cpp rmw_zenohd
2026-01-29T21:36:11.562326Z  INFO ThreadId(02) zenoh::net::runtime: Using ZID: a89baa
9018e338a3b489014a902c400f
2026-01-29T21:36:11.562983Z  INFO ThreadId(02) zenoh::net::runtime::orchestrator: Zen
oh can be reached at: tcp/[fe80::2c31:291c:29b4:41a8]:7447
2026-01-29T21:36:11.563013Z  INFO ThreadId(02) zenoh::net::runtime::orchestrator: Zen
oh can be reached at: tcp/192.168.0.109:7447
Started Zenoh router with id a89baa9018e338a3b489014a902c400f
```

```
ros-z-dev-jazzy-env  main [$!?] ro
s-z
 → ros2 run demo_nodes_cpp talker
2026-01-29T21:36:17.112758Z  INFO ThreadId
(02) zenoh::net::runtime: Using ZID: 3877d
6b13d42036ffcaff2ebfccc433
2026-01-29T21:36:17.114347Z  INFO ThreadId
(02) zenoh::net::runtime::orchestrator: Ze
noh can be reached at: tcp/[::1]:45953
[INFO] [1769722578.132652623] [talker]: Pu
blishing: 'Hello World: 1'
[INFO] [1769722579.132925979] [talker]: Pu
blishing: 'Hello World: 2'
[INFO] [1769722580.132640627] [talker]: Pu
blishing: 'Hello World: 3'
[INFO] [1769722581.133003020] [talker]: Pu
blishing: 'Hello World: 4'
```

```
ros-z-dev-jazzy-env  main [$!?] ro
s-z
 → ros2 run demo_nodes_cpp listener
2026-01-29T21:36:20.502853Z  INFO ThreadId
(02) zenoh::net::runtime: Using ZID: 5ac32
06910d55b54d9a429e362d60fd
2026-01-29T21:36:20.504117Z  INFO ThreadId
(02) zenoh::net::runtime::orchestrator: Ze
noh can be reached at: tcp/[::1]:39837
[INFO] [1769722581.133597231] [listener]:
I heard: [Hello World: 4]
```

```
NORMAL  ZenS13                                  1                    main   Friday, 30 Jan 2026 05:36:21
```


10 Hz, 64 bytes
RTT (microseconds)
2250
2000
1750
1500
1250
1000
750
500
829µs
779µs
694µs
1.4ms
918µs
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
10 Hz, 128KB
RTT (microseconds)
3000
2500
2000
1500
1000
1.2ms
1.6ms
1.1ms
1.7ms
1.3ms
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
10 Hz, 1MB
RTT (microseconds)
8000
7000
6000
5000
4000
3000
2000
2.3ms
4.9ms
2.3ms
5.0ms
3.7ms
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
10 Hz, 8MB
RTT (microseconds)
50000
40000
30000
20000
10000
17.6ms
30.4ms
12.5ms
30.7ms
24.7ms
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
200 Hz, 64 bytes
RTT (microseconds)
1800
1600
1400
1200
1000
800
600
400
200
509µs
461µs
488µs
885µs
471µs
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
200 Hz, 128KB
RTT (microseconds)
2000
1750
1500
1250
1000
750
500
596µs
1.1ms
640µs
1.0ms
662µs
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
200 Hz, 1MB
RTT (microseconds)
5000
4000
3000
2000
1.5ms
3.6ms
1.5ms
2.9ms
2.6ms
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware
200 Hz, 8MB
RTT (microseconds)
160000
140000
120000
100000
80000
60000
40000
20000
105.8ms
40.3ms
12.7ms
97.4ms
56.9ms
rclcpp/z
rclrs/z
rosz
rclpy/z
roszpy
Middleware

Wish List - looking for sponsors!

- Go API (already sponsored!)
- Native buffers for mixed-memories data (CUDA, Torch Tensor)
- Nostd feature. Run on embedded devices.
- Foxglove formats support
- Launching system

EMPYREAN

Trustworthy, Cognitive and AI-Driven Collaborative associations of IoT devices and edge resources for data processing

EMPYREAN research project has received funding from the European Union's HORIZON Europe under the Grant Agreement n° 101136024.

EcoMobility

Intelligent, Safe & secure connected Electrical Mobility solutions: Towards European Green Deal & Seamless Mobility

EcoMobility has received funding from Chips Joint Undertaking (Chips JU) under Grant Agreement No 101096387. Co-funded by European Union.

UP2DATE4SDV

Enabling safe & secure modular UPdates, UPgrades and DynAmic Task reallocation and Execution for Software-Defined Vehicles

EMPYREAN research project has received funding from the European Union's HORIZON Europe under the Grant Agreement n° No. 101203060.

EdgeAI-Trust

Decentralized Edge Intelligence: Advancing Trust, Safety, and Sustainability in Europe

EdgeAI-Trust "Decentralized Edge Intelligence: Advancing Trust, Safety, and Sustainability in Europe" project has received funding from Chips Joint Undertaking (Chips JU) under Grant Agreement No 101139892-2.

O-CEI

O-CEI Open CloudEdgeIoT Platform Uptake in Large Scale Cross-Domain Pilots

O-CEI project has received funding from the European Union's Horizon Europe Framework Programme under the Grant Agreement N° 101189589.

Funded by the European Union

ChipsJU

Zenoh

Join our discord

ROS-Z

Star ros-z project

COSCUP

Welcome to Taiwan