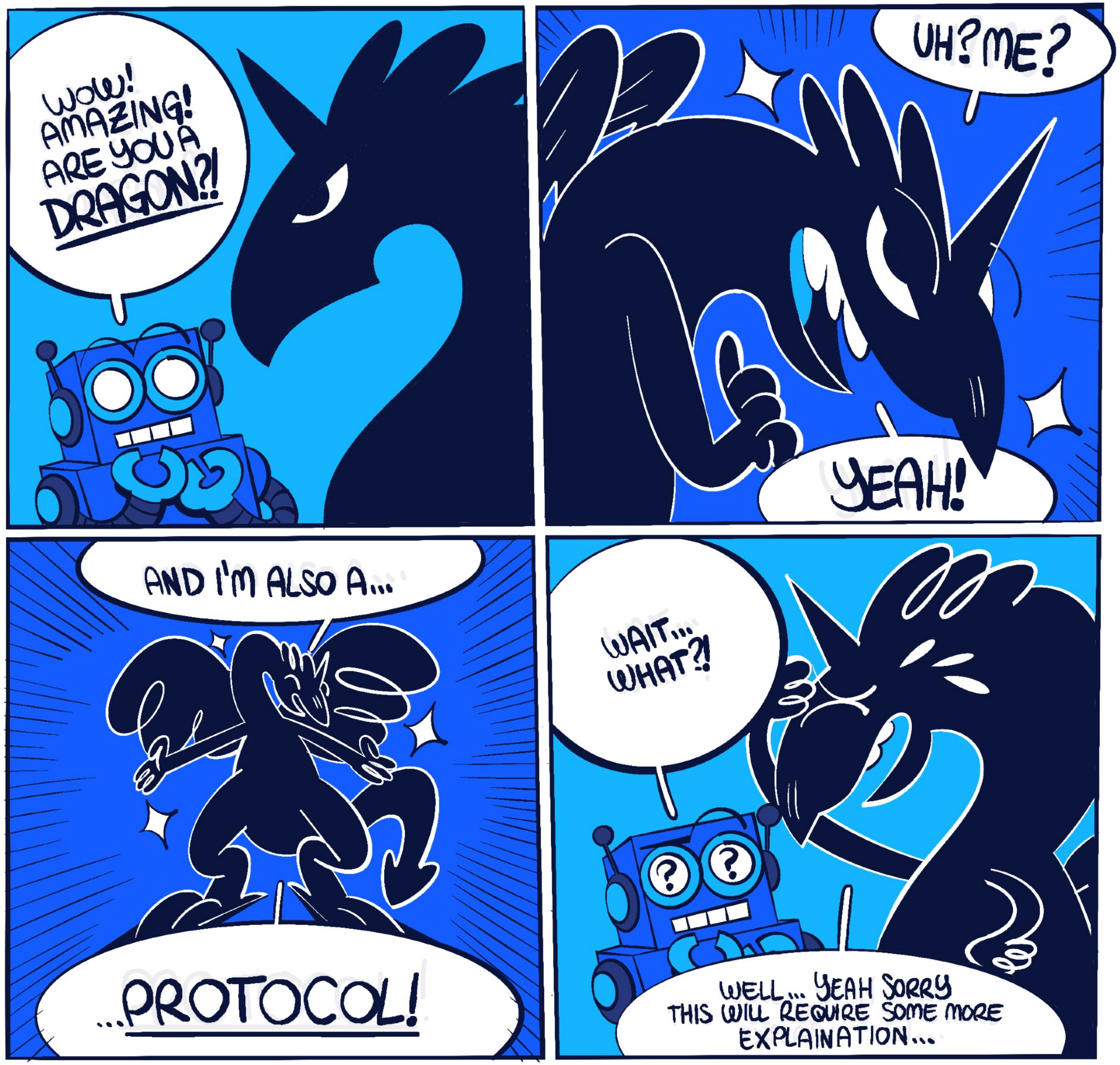
RMW Zenoh Workshop

ROSCon 2024 - Odense

Julien Enoch

Senior Solutions Architect
julien.e@zettascale.tech

Zenoh


WOW! AMAZING! ARE YOU A DRAGON?!
UH? ME?
YEAH!
AND I'M ALSO A...
...PROTOCOL!
WAIT... WHAT?!
WELL... YEAH SORRY THIS WILL REQUIRE SOME MORE EXPLAINATION...

Zenoh

Pub/Sub/Query protocol that **Unifies data** in **motion**, data at **rest** and **computations** from embedded microcontrollers up the data centre

Provides **location-transparent** abstractions for **high performance pub/sub** and **distributed queries** across heterogeneous systems

Built-in support for zero-copy and shared memory

Dragons teach us that if we want to climb high we have to do it against the wind.

zetta scale

Universal Abstractions

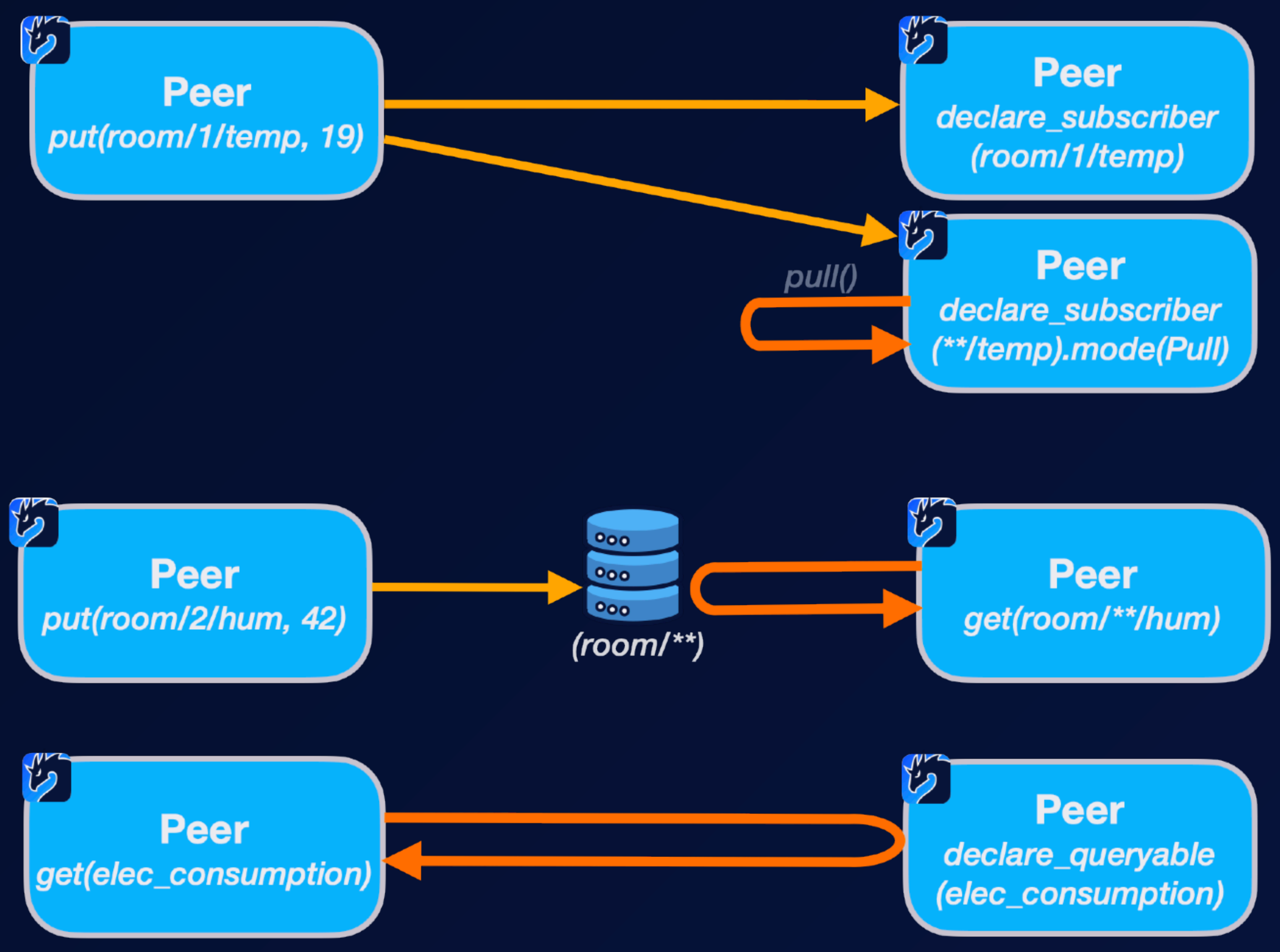
Zenoh's abstraction are **universal** since they allow to express the key patterns in distributed computing, namely:

Publish/Subscribe. Trivially supported by Zenoh's **Publisher** and **Subscriber**

Remote Computation. A **Queryable** represents a generalised computation, since it can transparently deal with replication and partitioning

Storage. Represented by the combination of a **Queryable** and a **Subscriber**

Additionally all these primitives enjoy location transparency by the virtue of being data-centric.



zetta scale

Runs Everywhere

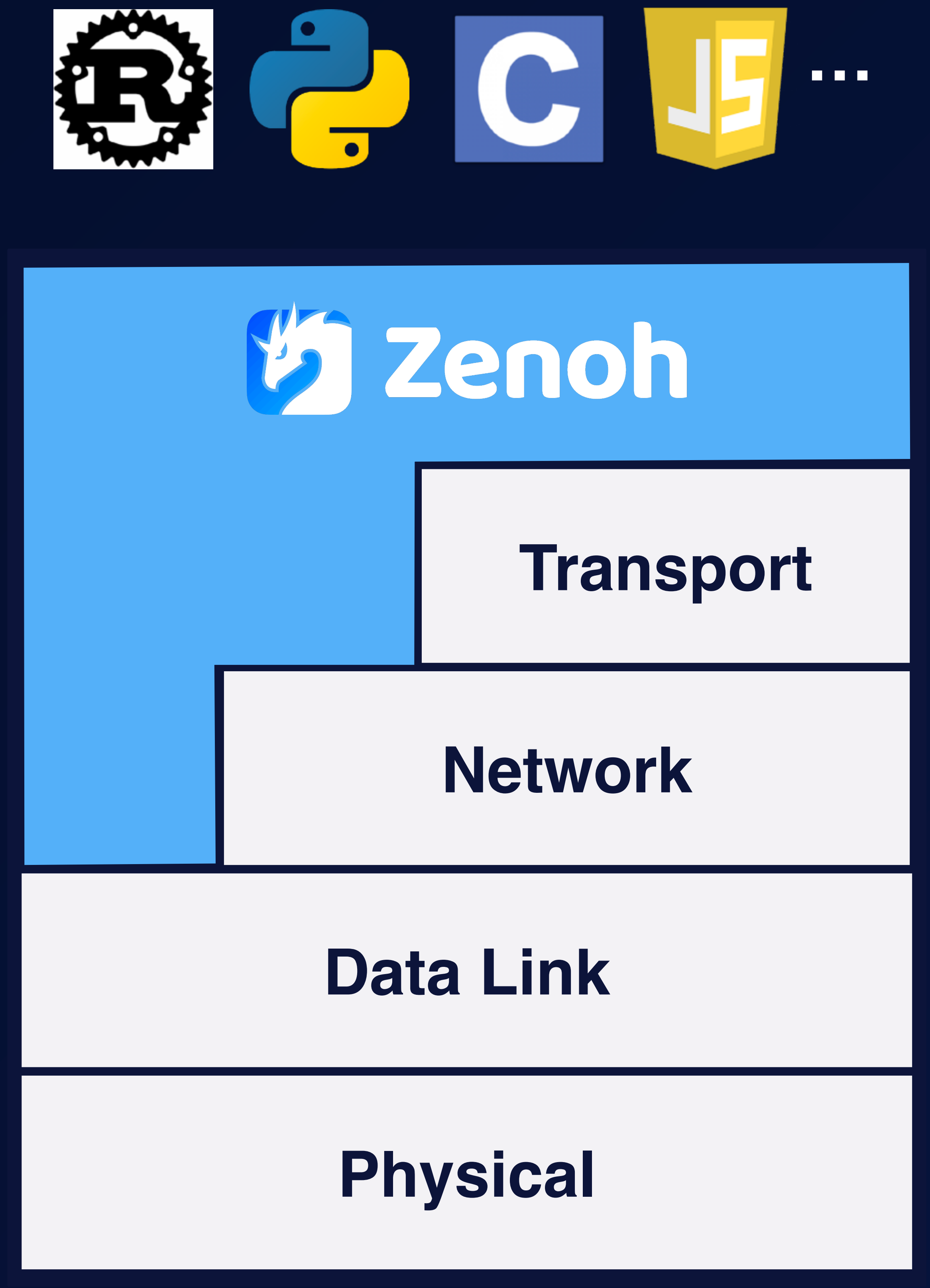
Written in Rust for security, safety and performance

Native libraries and **API bindings** for many **programming languages**, e.g., Rust, C/C++, Python, JS, REST, C#, Kotlin and Java

Built-in support Shared Memory and Zero Copy

Supports **network technologies** from **transport layer down-to** the **data link.** Currently runs on, TCP/IP, UDP/IP, QUIC, Serial, Bluetooth, OpenThreadX, Unix Sockets, Shared Memory

Available on **embedded** and **extremely constrained devices** and **networks** – **5-6** bytes minimal overhead



Any Topology

Peer-to-peer

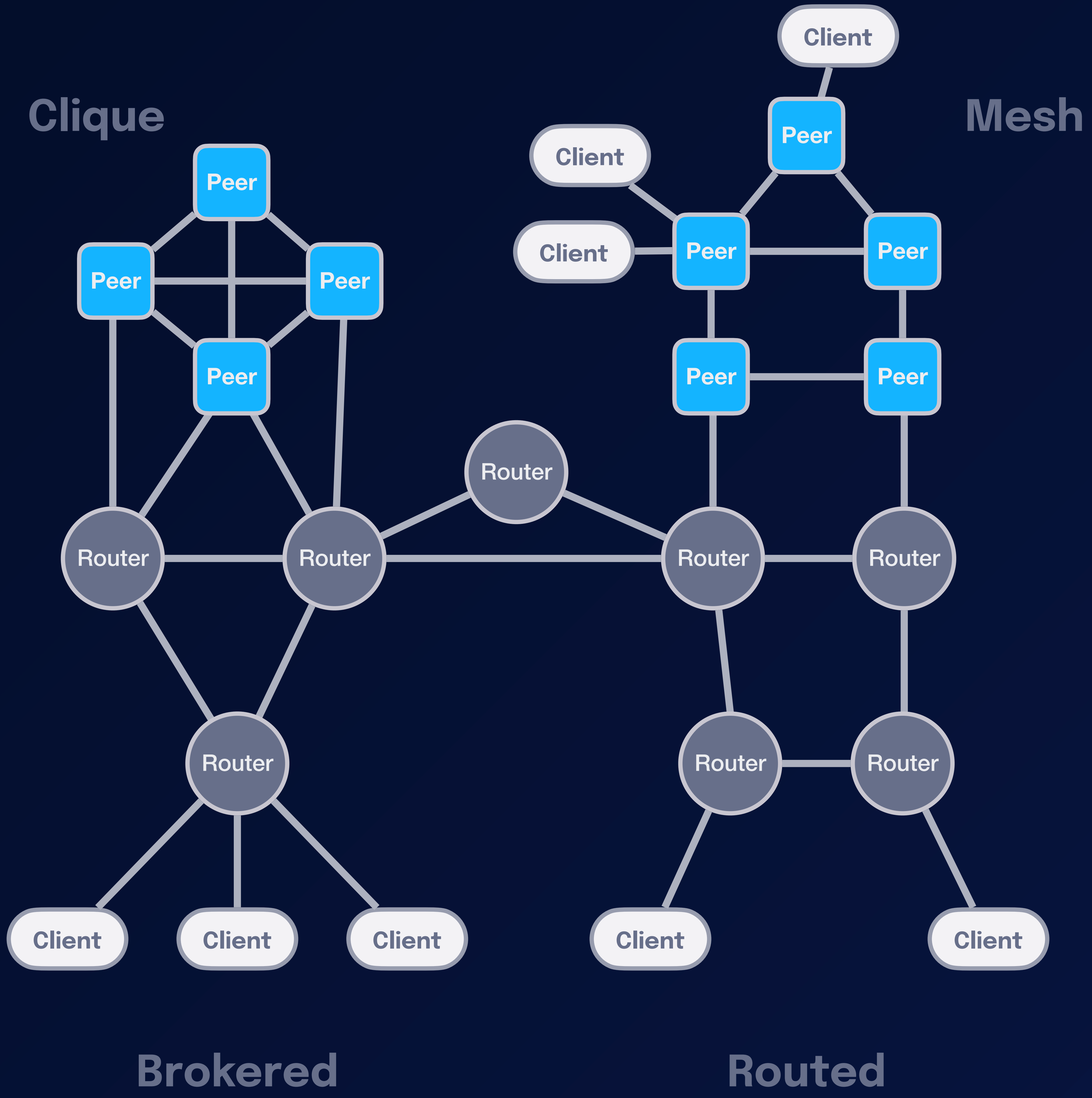
Clique and mesh topologies

Brokered

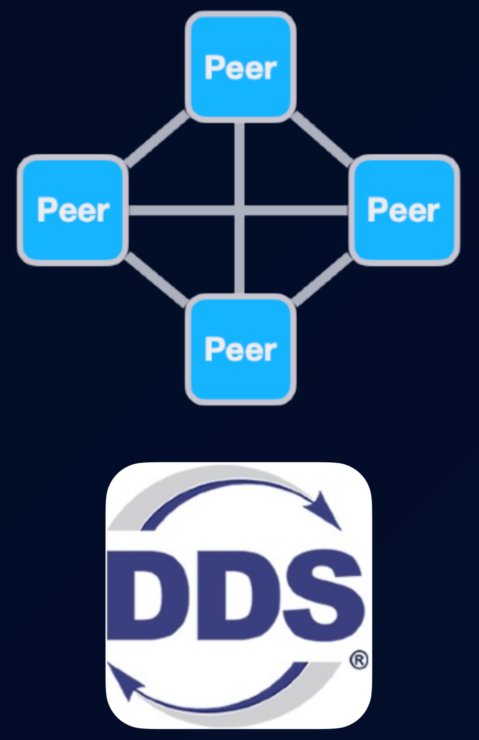
Clients communicate through a router or a peer

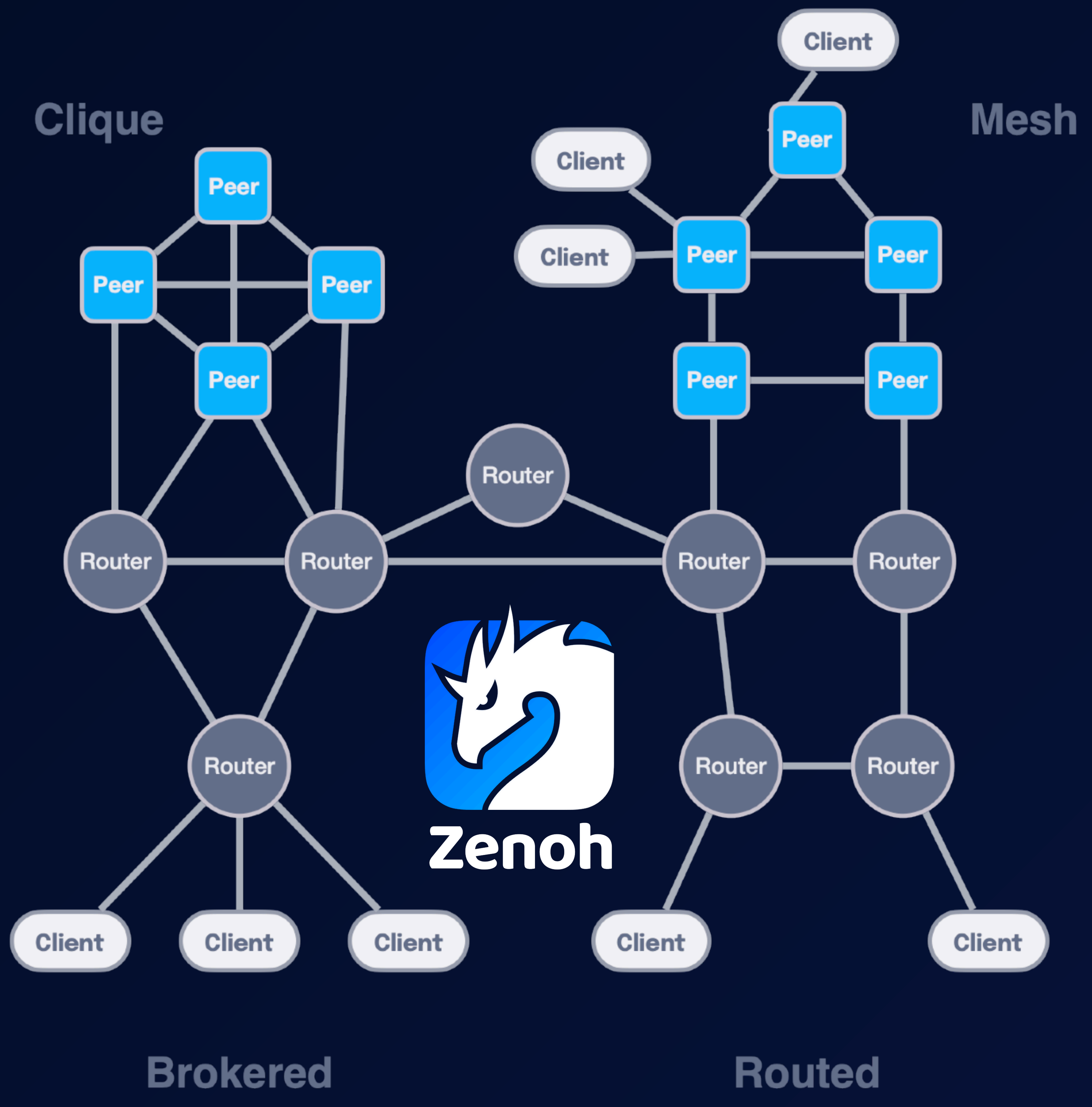
Routed

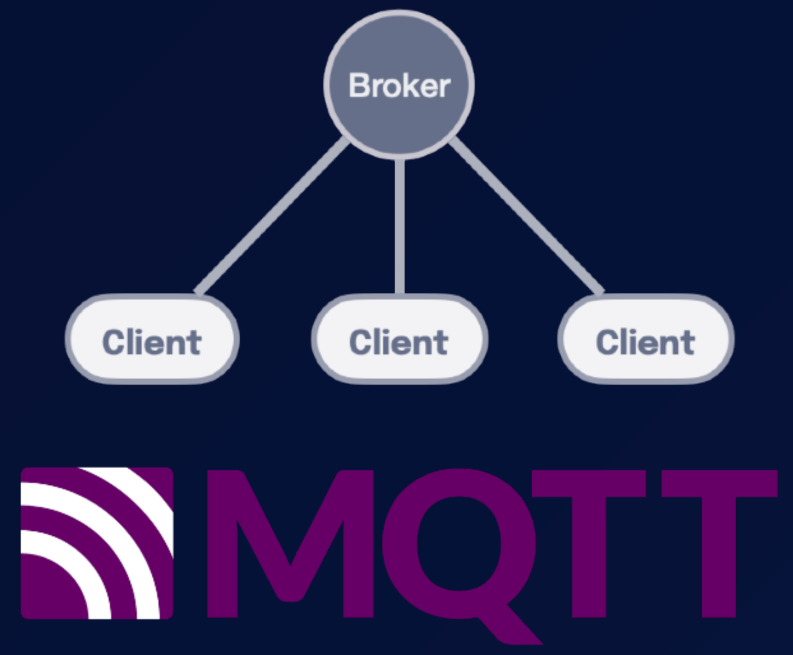
Routers forward data to and from peers and clients



Topology in Perspective

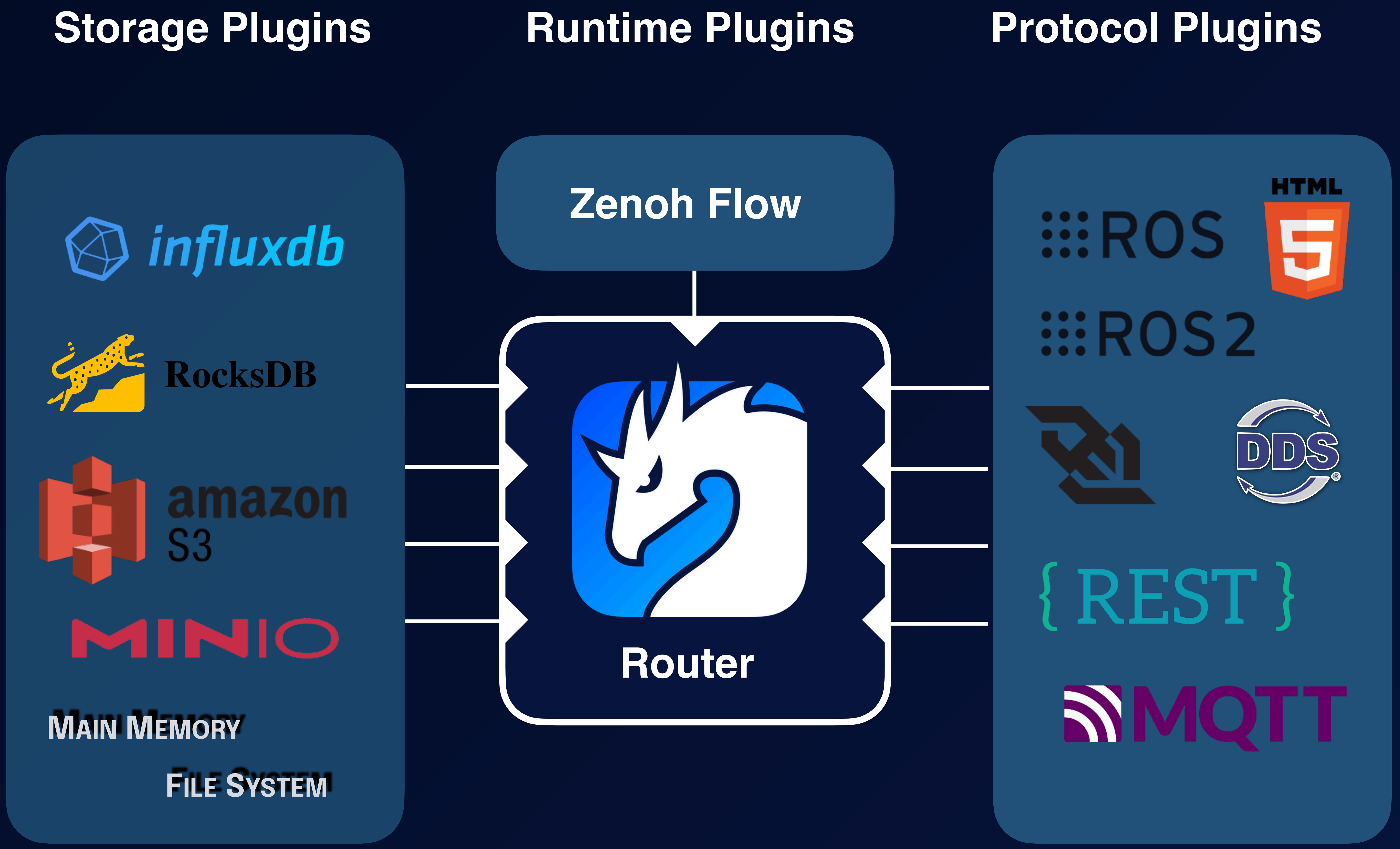






zetta scale

Plug-Ins



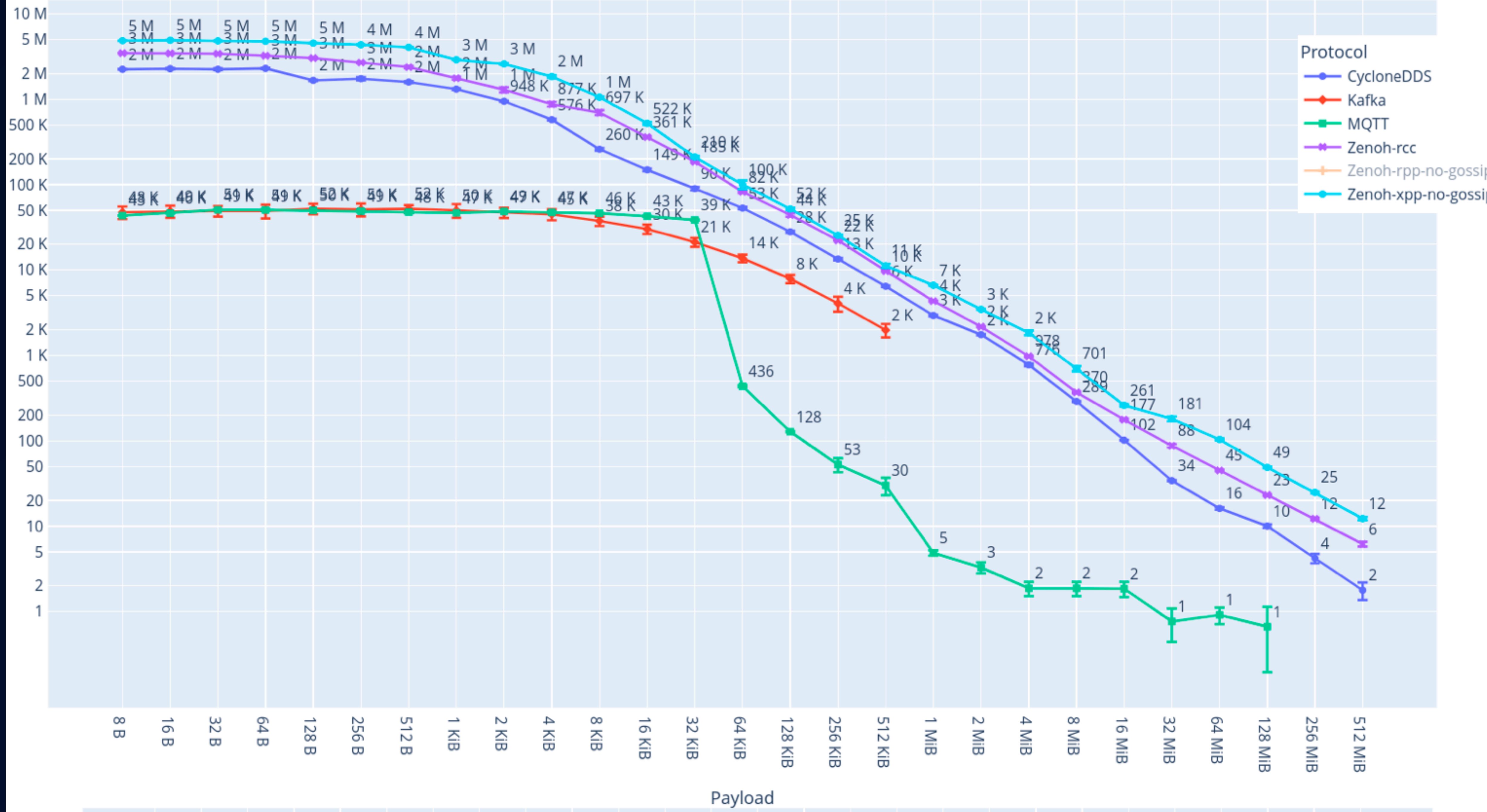
Zenoh vs DDS, MQTT & Kafka

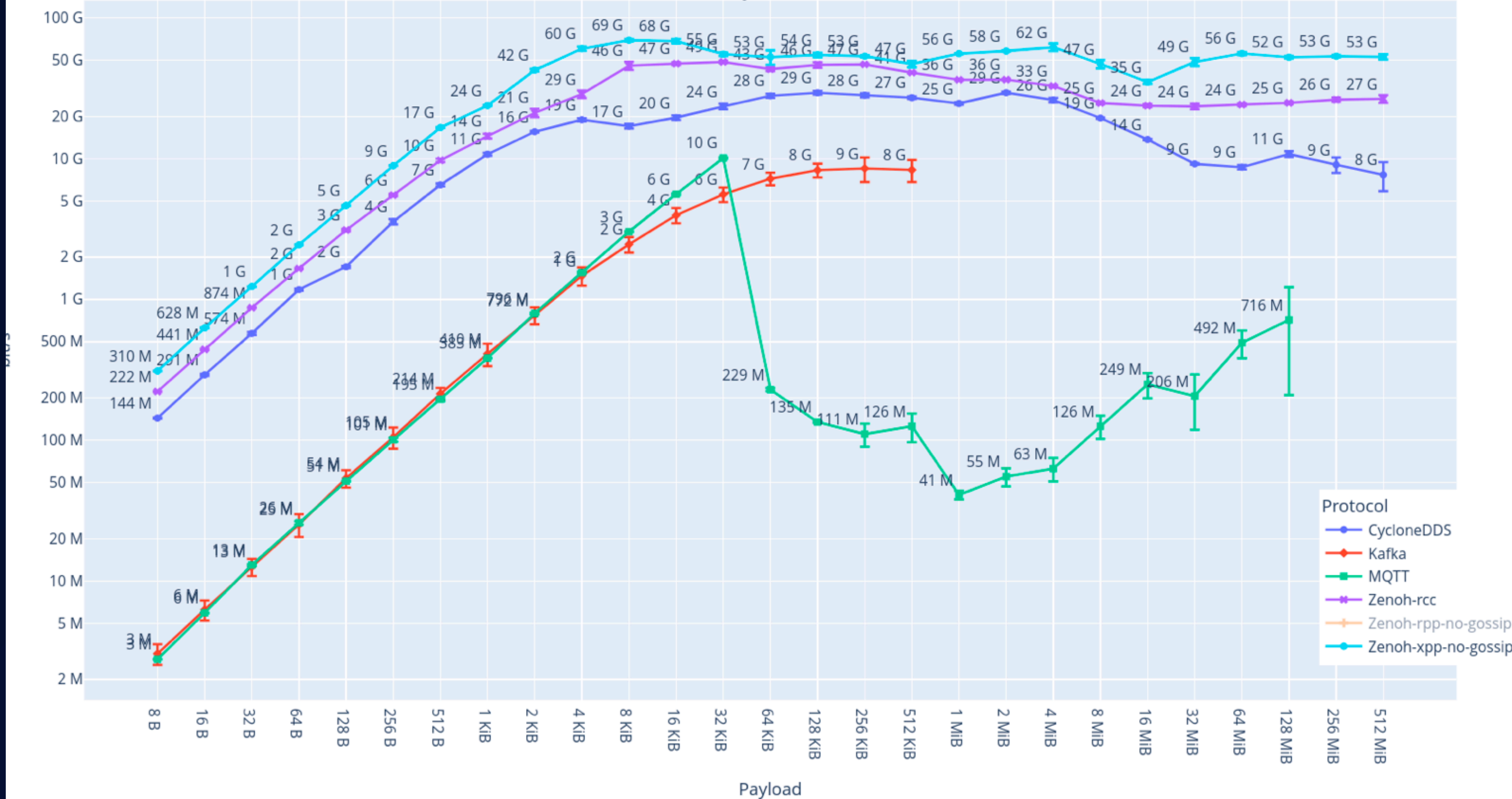
Independent Benchmark from NTU

Zenoh can deliver at peak performance of ~70Gbps at 8Kb payload:

- 3.3x higher than DDS
- 23x higher than Kafka
- 35x than MQTT (higher for larger payload)

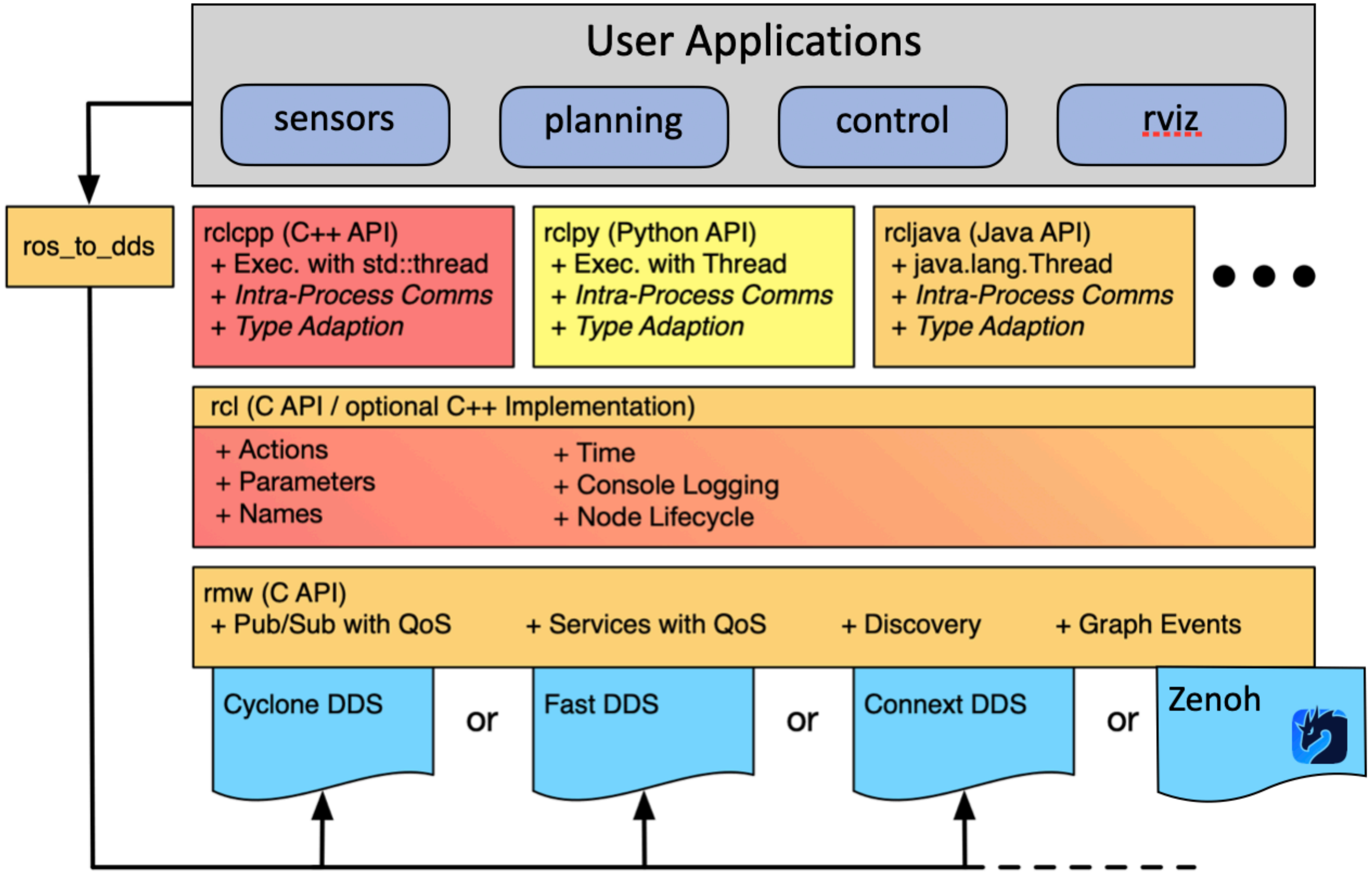
Zenoh’s latency 10us (7us for pico)
- 25us for MQTT
- 75us for Kafka
- 8us DDS





RMW Zenoh

ROS 2 has a modular architecture



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*

• • •

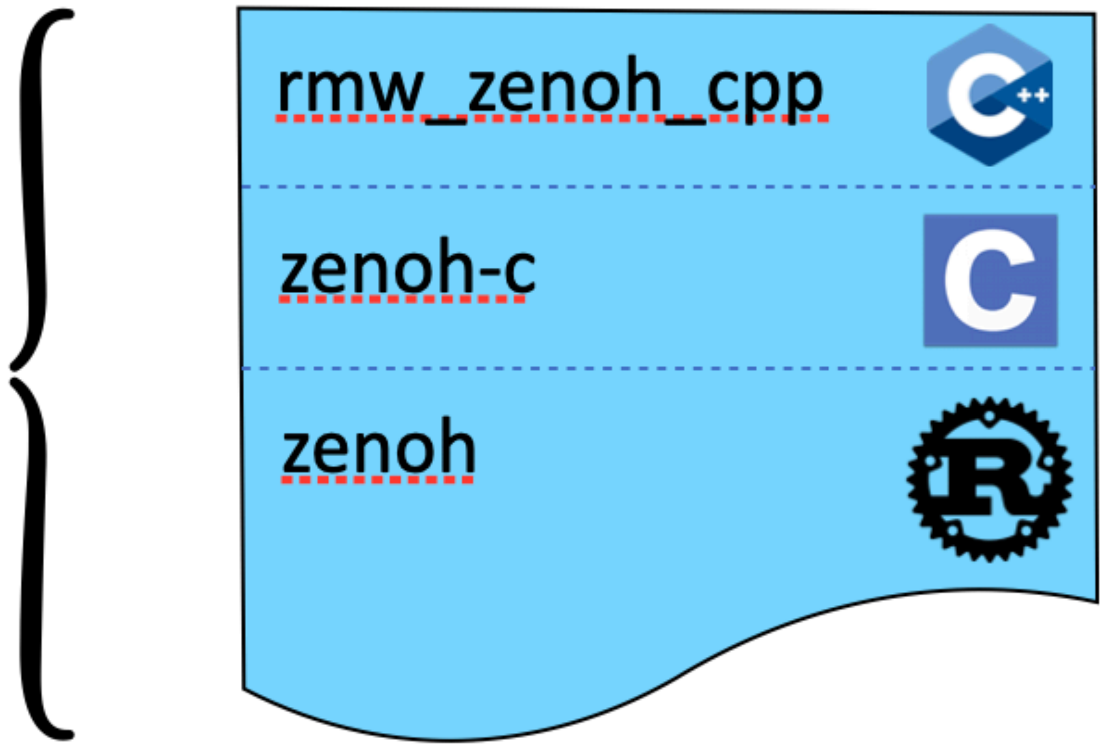
rcl (C API / optional C++ Implementation)
+ Actions
+ Parameters
+ Names
+ Time
+ Console Logging
+ Node Lifecycle

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

Cyclone DDS or Fast DDS or Connext DDS or 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



rmw_zenoh_cpp

zenoh-c

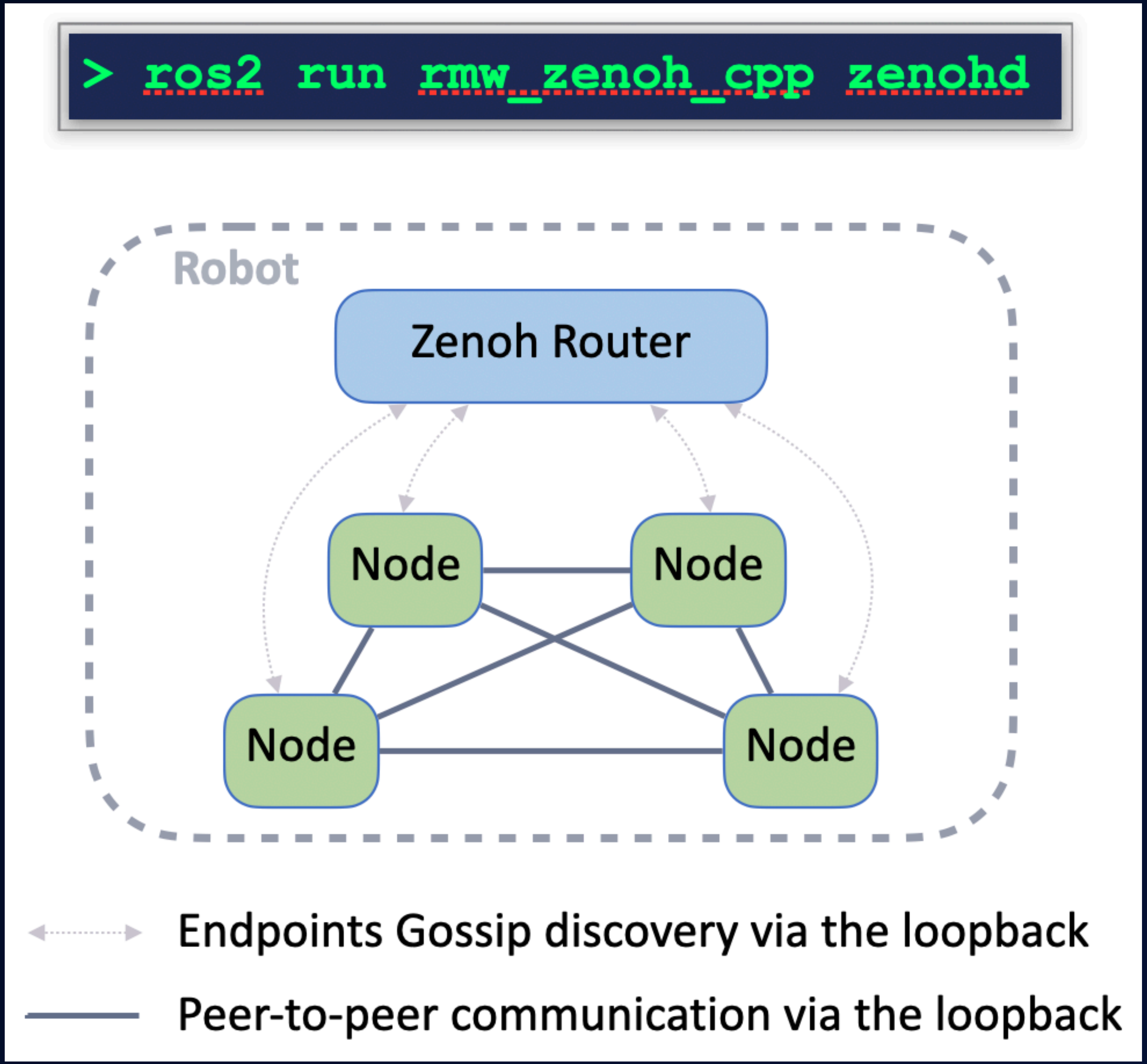
zenoh

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

ROS to Zenoh Mapping

Data encoding	Still DDS **CDR** encoding via code generation: .msg/.srv/.action => .idl => serializer/deserializer code
Publisher / Subscriber	Zenoh Publisher / Subscriber
Service Server / Client	Zenoh Queryable / Querier
Actions	Still mapped by RCL on 3 Services and 2 Topics (thus 3 Queryables and 2 Publishers)
Parameters	Still mapped by RCL on Services (thus Queryables)
Entities discovery and ROS Graph	Zenoh Liveliness tokens (lightweight and reactive)

Zenoh Router



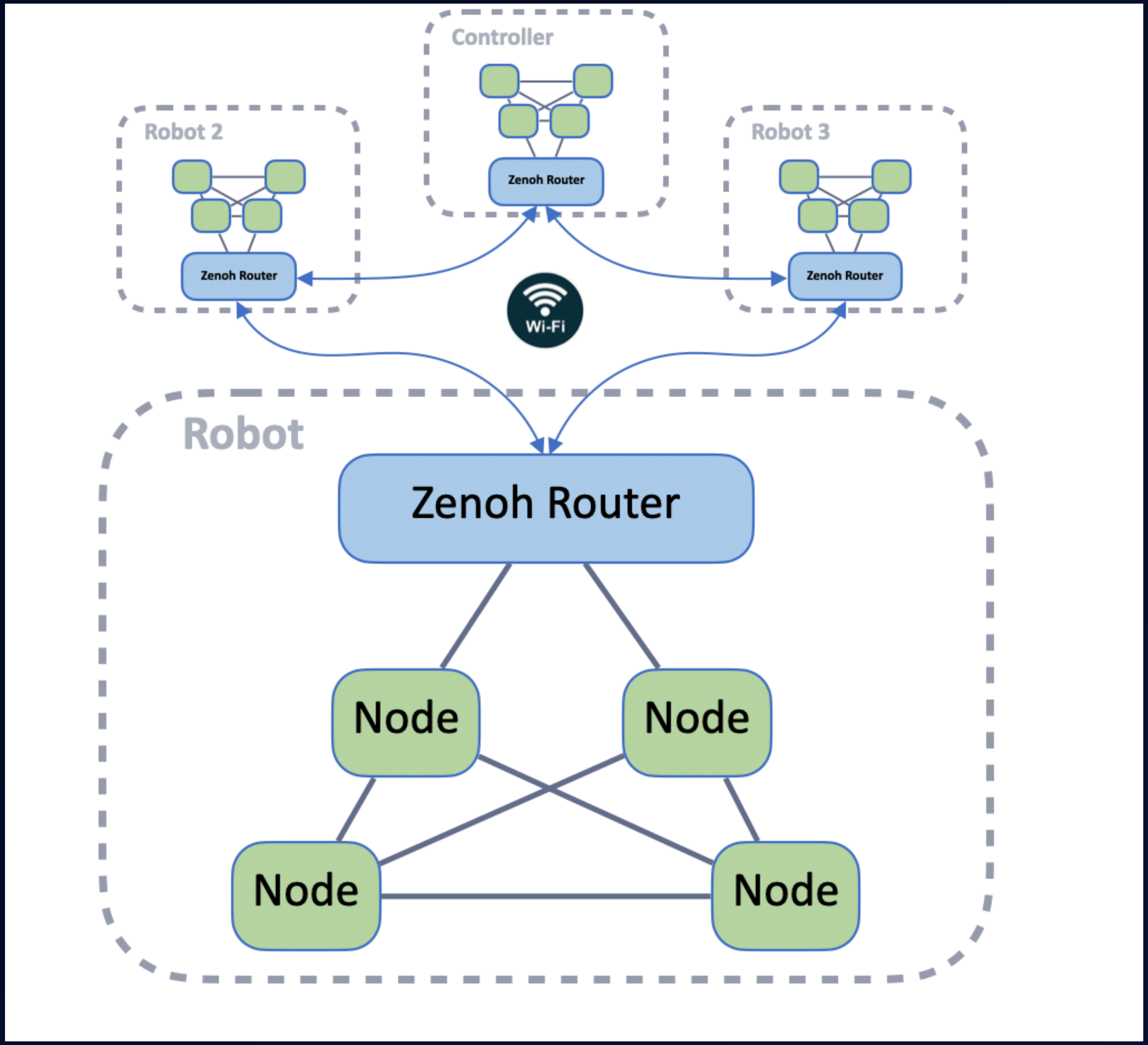
```
> ros2 run rmw_zenoh_cpp zenohd
```

Robot
Zenoh Router
Node
Node
Node
Node
Endpoints Gossip discovery via the loopback
Peer-to-peer communication via the loopback

For Discovery:

- The router listen on tcp/0.0.0.0:7447
- Each Node connect to the router on tcp/127.0.0.1:7447
- The router acts as a broker for endpoints discovery
- Nodes establish peer-to-peer connections via 127.0.0.1

Zenoh Router



For external communications:

- Configure the router to connect to other routers, via TCP, TLS, QUIC…
- Benefits:
  - Less connections, less overheads
  - Batching for better throughput
  - Smaller surface of attack
  - Single point to configure Access Control and Downsampling

Installation

- Only for **Iron**, **Jazzy** or **Rolling**
- Not yet available as Debian package
- Built from sources, as any ROS 2 package:

```
> cd ros_workspace
> git clone https://github.com/ros2/rmw_zenoh src/rmw_zenoh
> rosdep install --from-paths src --ignore-src --rosdistro $ROS_DISTRO -y
> colcon build --cmake-args -DCMAKE_BUILD_TYPE=Release
> export RMW_IMPLEMENTATION=rmw_zenoh_cpp
```

Already done in your image:

zettascaletech/roscon2024_workshop

Configuration

- 1 configuration file (json5, json or yaml) per Node and Router
- Default config files in sources under : `rmw_zenoh/rmw_zenoh_cpp/config/`
- `$ZENOH_SESSION_CONFIG_URI` for Nodes config
  - Most of the time nothing to change here
- `$ZENOH_ROUTER_CONFIG_URI` for Router config
  - External connectivities
  - TLS keys and certificates
  - Access control
  - Downsampling
  - …

zetta
scale

Thank You

Patience, persistence and perspiration make an unbeatable combination for success.