THE EDGE DATA FABRIC

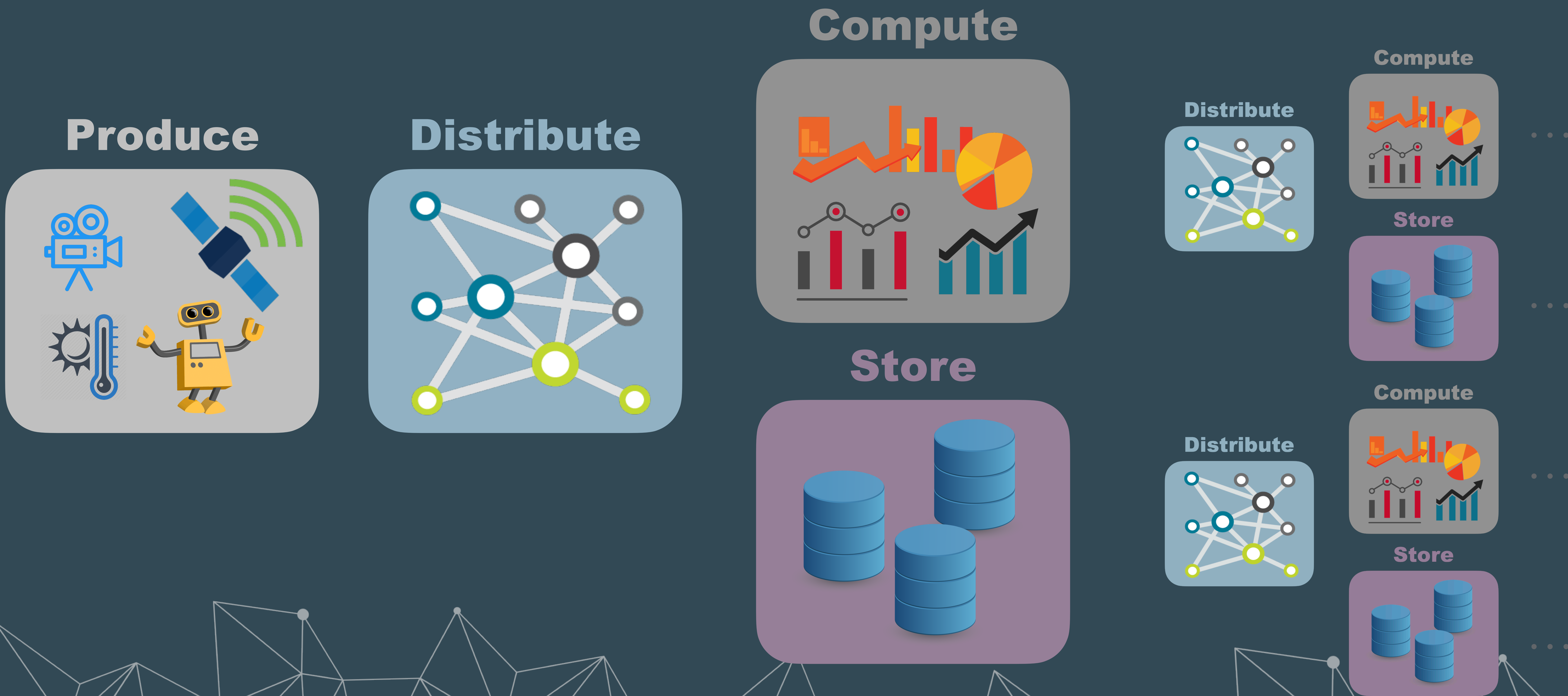
zenoh

ADLINK
Leading EDGE COMPUTING
Advanced Technology Office

Carlos Guimarães, PhD
Senior Technologist
carlos@adlink-labs.tech

Context

The Data Journey

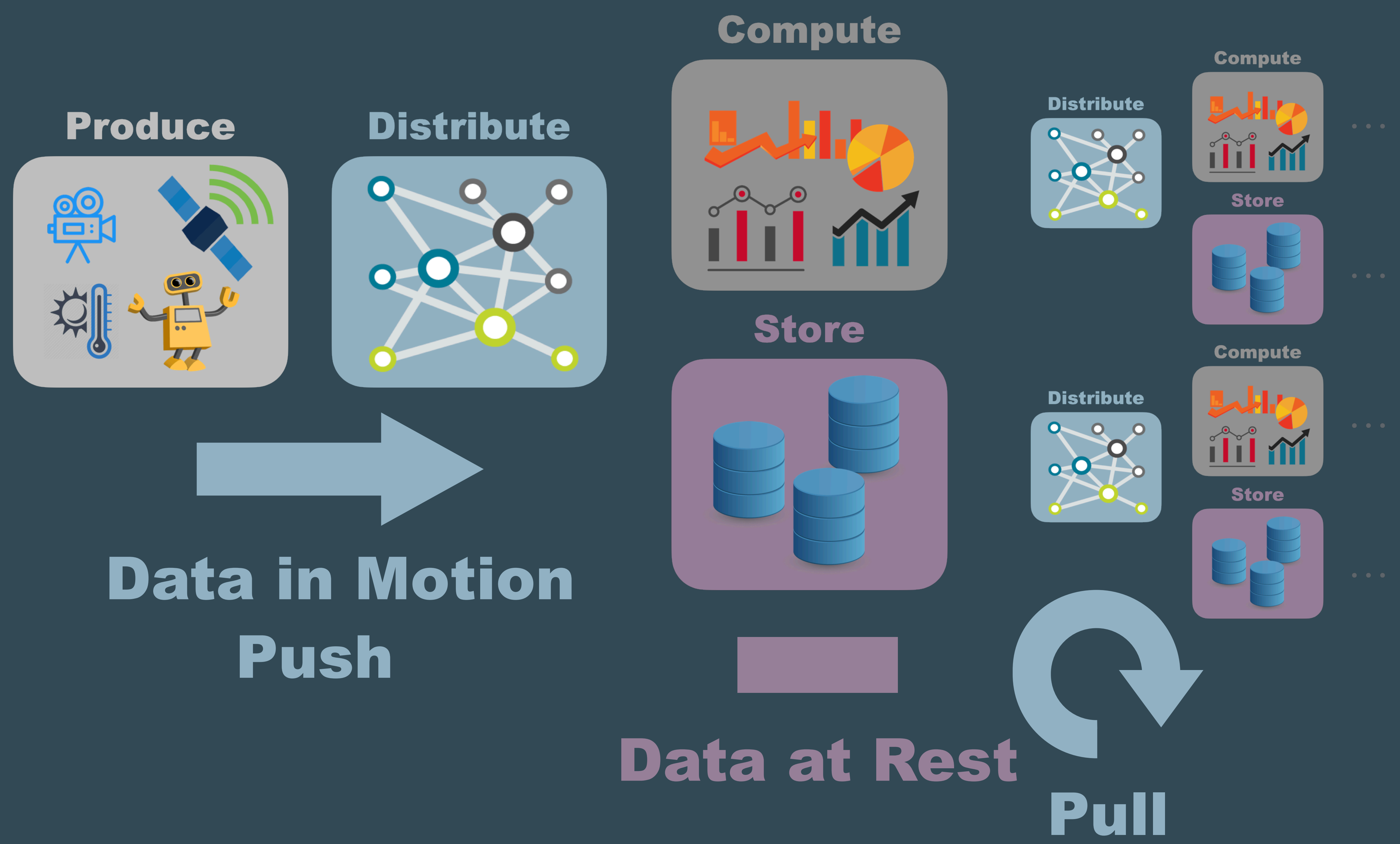


Moving and Resting

Technologies for dealing with data in motion and data at rest have belonged historically to different families

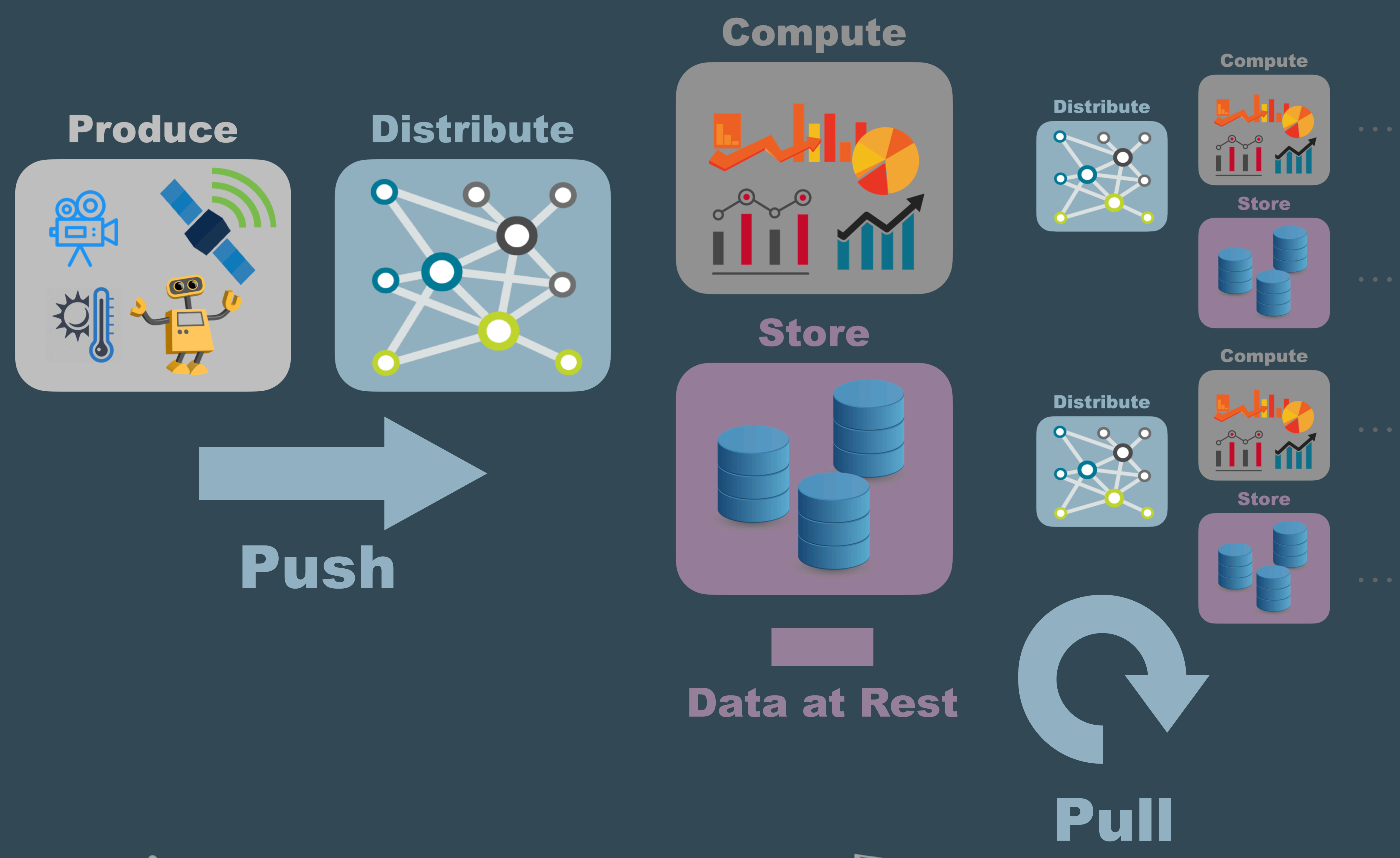
Data in motion is **Push**ed to interested parties

Data at rest is **Pull**ed when needed



Technological Fragmentation

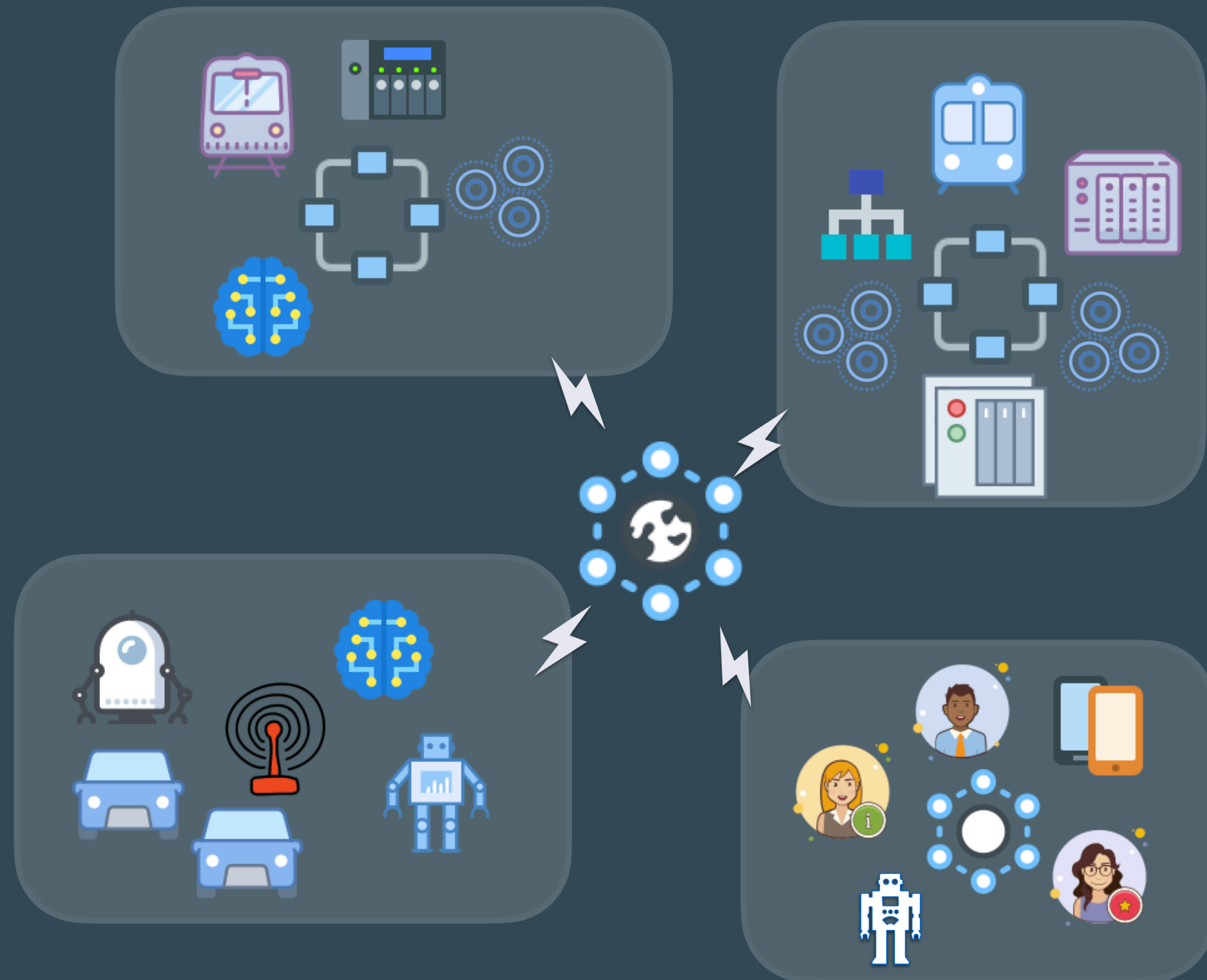
The technological fragmentation exist in several **Data Distribution, Data Storage** and the integration of the two



Decentralisation

The increasing **availability** of and **storage**, **compute** capabilities on **devices** is creating new **opportunities** for **computing** and **storing** and data much **closer** its **production**

Existing technologies for data in motion and data at rest **fall short** in **supporting** this **scenario**. More importantly fail to provide a **unified** and **location transparent data management**.



Filling the Gap

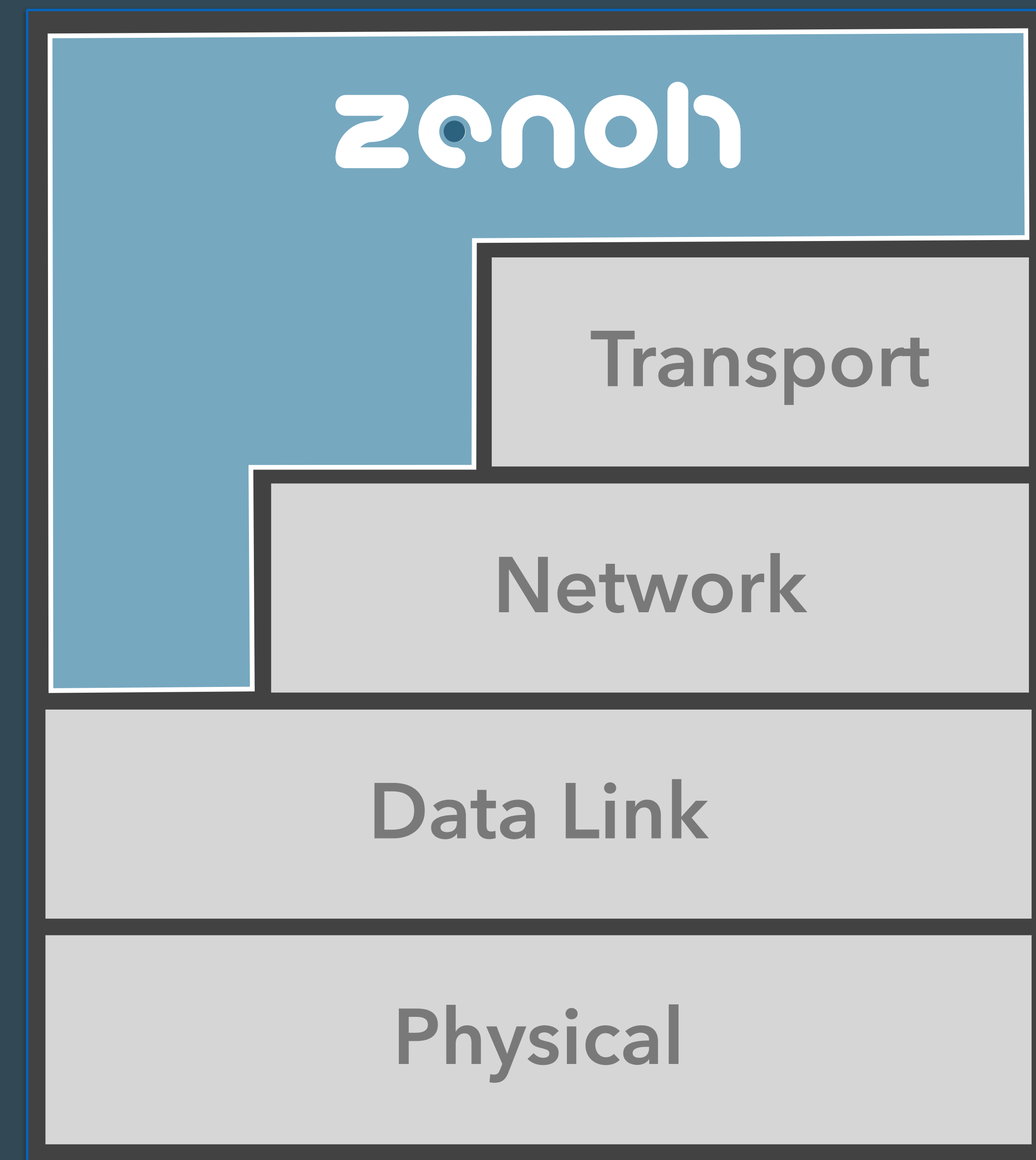
Unifies data in **motion**, data **in-use,** data at **rest** and **computations**.

It carefully **blends** traditional **pub/sub** with **distributed queries**, while retaining a level of **time and space efficiency** that is well beyond any of the mainstream stacks.

It provides built-in support for **geo-distributed storages** and **distributed computations**

zenoh

Provides a **high level API** for **high performance pub/sub** and **distributed queries**, **data** representation **transcoding**, an implementation of **geo-distributed storage** and **distributed computed values**



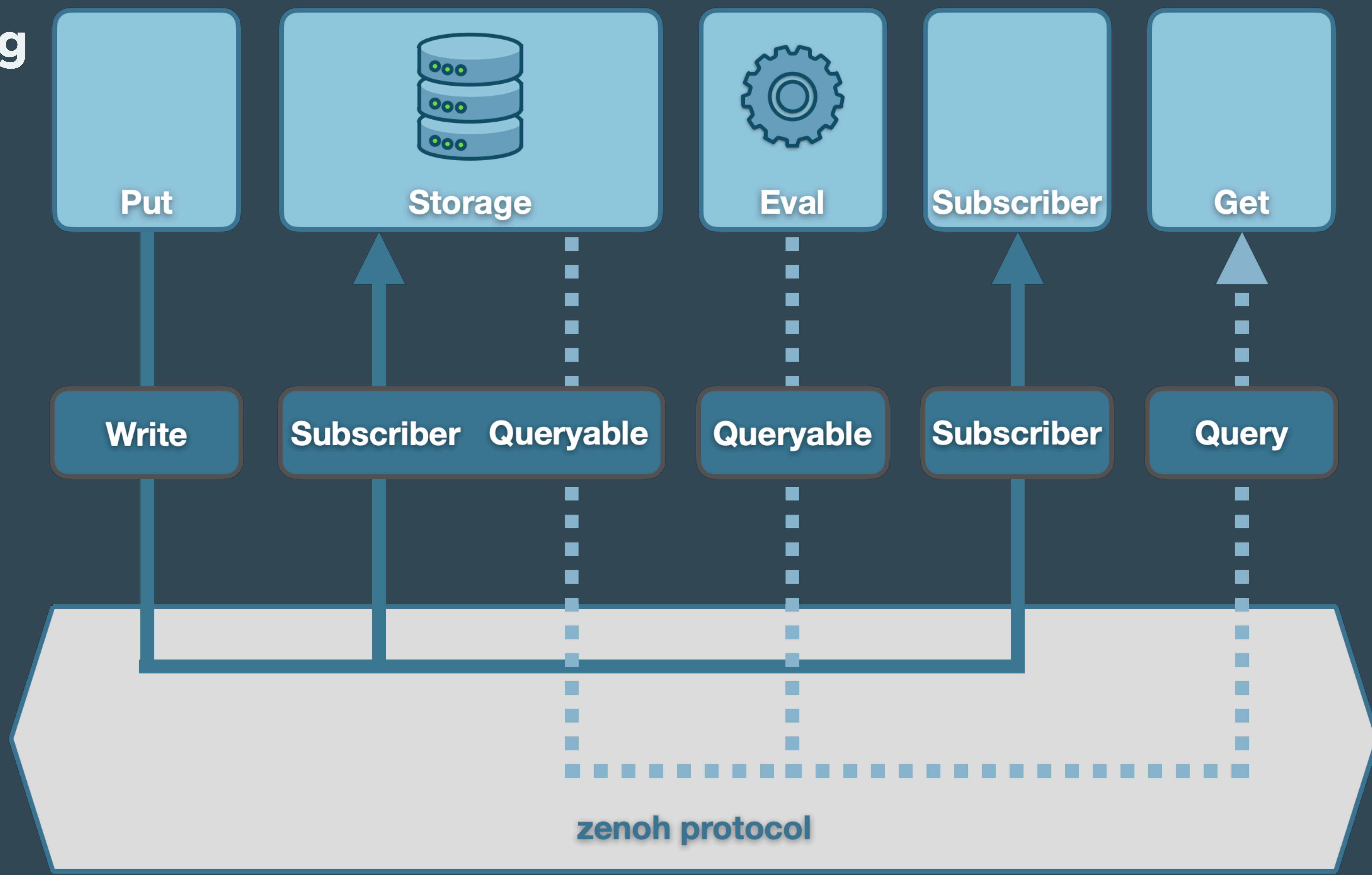
Data-oriented abstractions, supporting distributed storage, content-based filtering and transcoding.

This layer is content aware.

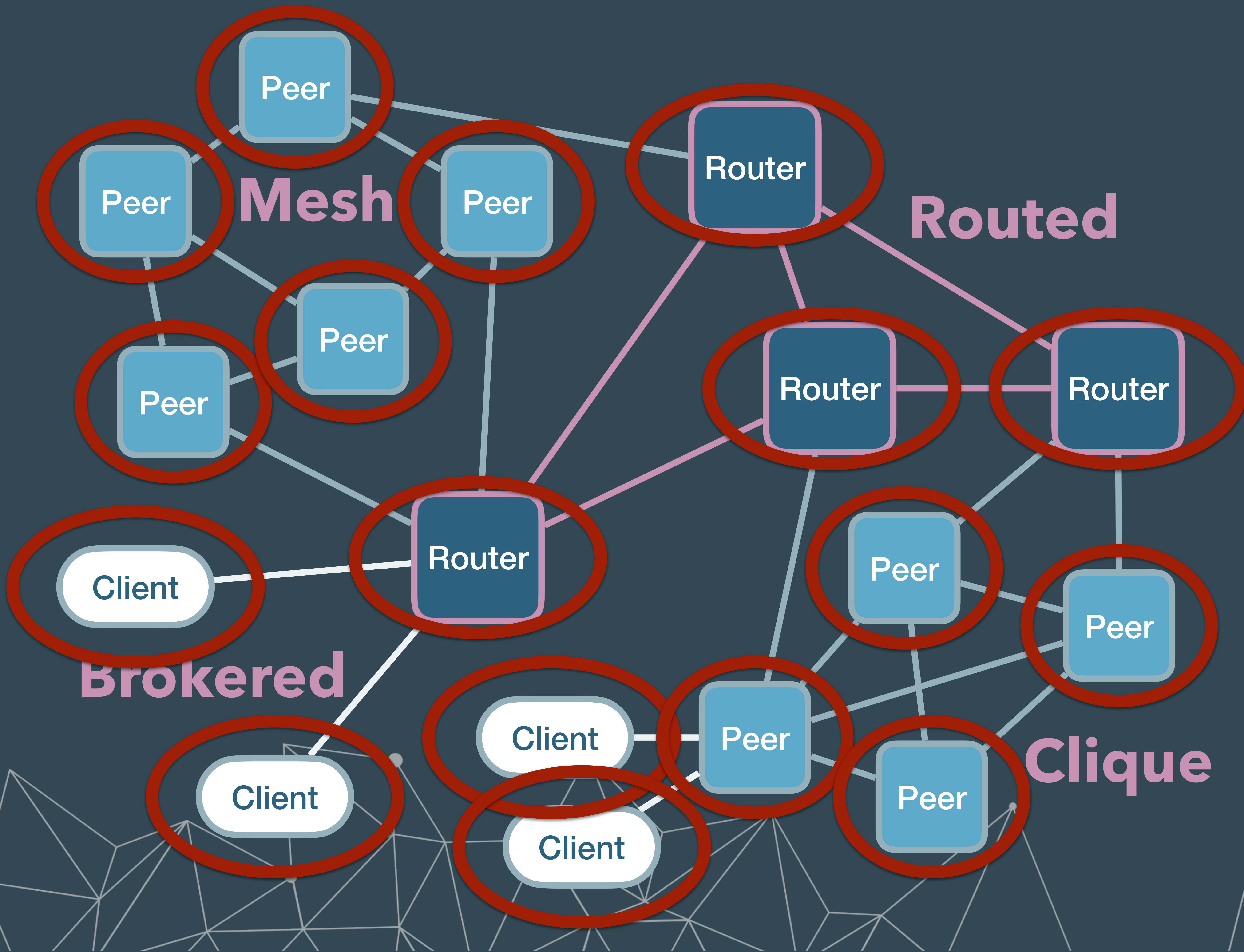
Data transportation primitives, supporting pub/sub (push) and query/reply (pull) communications.

This layer is content agnostic.

Networking layer capable of running above a Data Link, Network or Transport Layer.



Communication models



Peer-to-peer
Clique and *Mesh* topologies

Brokered
Clients communicate through a *router* or a *peer*

Routed
Multiple *routers* cooperate to forward data *to* and *from* *peers* and *clients*

Under the Hood

Naming Data

Following the tradition of Named Data Networking protocols, **data** is **named by a sequence of byte arrays** – called **key** – such as:

```
/home/kitchen/sensors/temp
/home/kitchen/sensors/C202
```

Data interest and **intents** are **expressed** by means of **keys regular expressions**, such as:

```
/home/*/sensors/temp
/home/**/C202
```

Selecting Data

Uses **selector** to **defines data sets**. A selector is composed by a **key expression**, and optionally a **predicate**, a **projection** and a set of **properties**

```
/myhome/*/sensor/temp?value>25
/mycar/dynamics?speed>25#acceleration
```

The **key-expression** is used to **route**, while **predicate, properties, projection,** etc., are interpreted only by the entity that executes the query. It also provide different **policies** to control **query consolidation** and **completeness** and potentially **quorums**

Primitives: Entities

Resource. A **named data**, in other term a (key,value)
(e.g. /home/kitchen/sensor/temp, 21.5
/home/kitchen/sensor/hum, 0.67)

Publisher. A **spring** of values for a key expression
(e.g. /home/kitchen/sensor/temp
/home/kitchen/sensor/*)

Subscriber. A **sink** of values for a key expression
(e.g. /home/kitchen/sensor/temp
/home/kitchen/sensor/*)

Queryable. A **well** of values for a key expression
(e.g. /home/**)

Primitives: Operations

scout – Looks for zenoh entities, the kinds of relevant nodes, either peers or router as specified by a bit-mask.

init & open / close – Establishes / Closes a **zenoh** session.

declare / undeclare – Declare / Undeclare resource, publisher, subscriber and queryable. Declarations are used for discovery and various optimisations.

put – Writes data for a key expression.

pull – Pulls data for a pull subscriber.

get – Issues a distributed query and returns a stream of results. The query target, coverage and consolidation depends on policies.

Consolation Strategies

Allows to correlate, combine and integrate data from different queryables in a single response.

Based on timestamped data.

Applied at three different stages:

- First router (default: Lazy)
- Last router (default: Lazy)
- Reception (default: Full)

None – All responses are forwarded back to the querier.

Lazy – Only replies more recent that the previously sent are forwarded

Full – Only the most recent reply for each resource is sent back to the querier

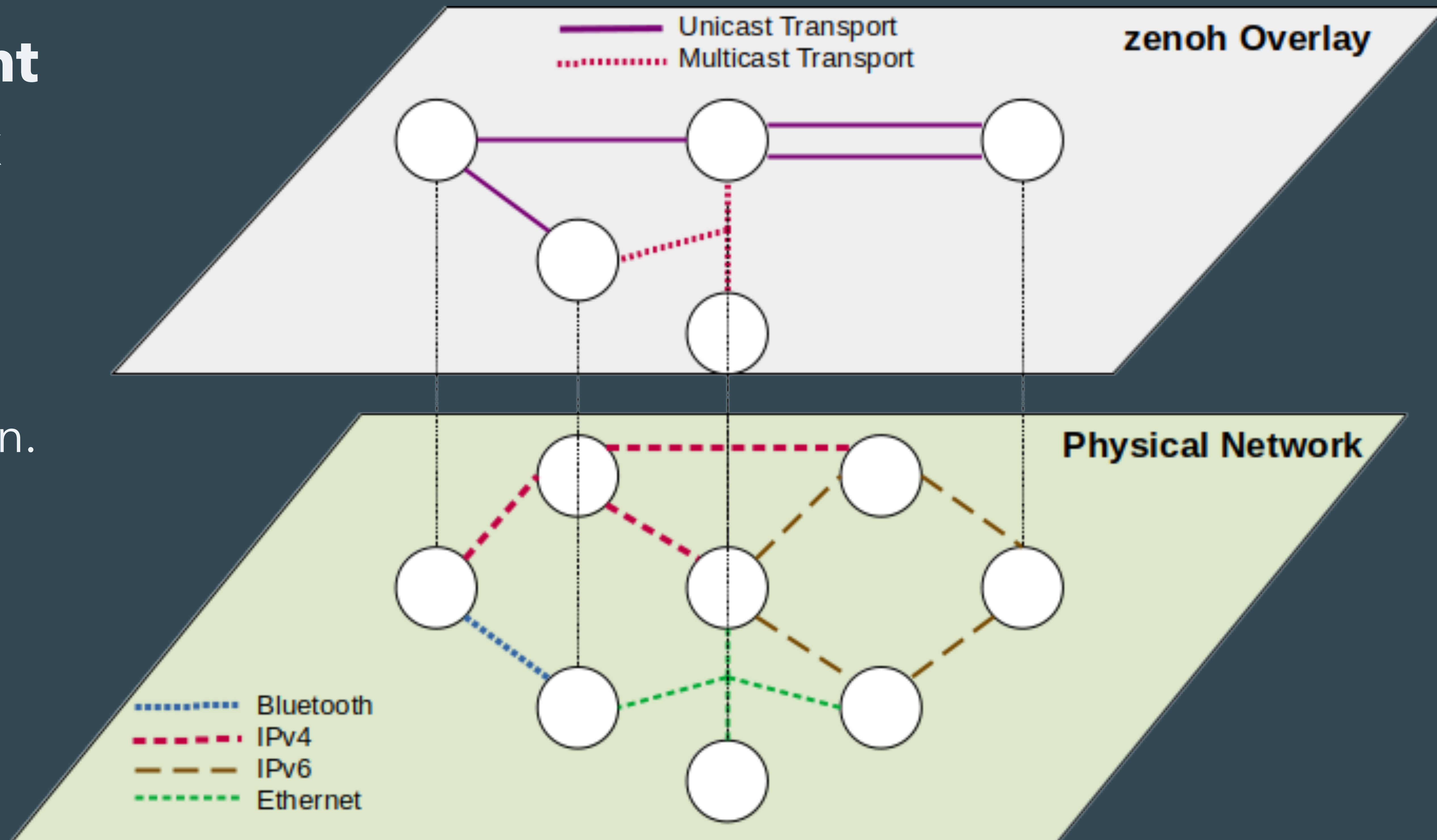
Overlay Infrastructure

zenoh sessions are **transparent** from the **underlying network protocols.**

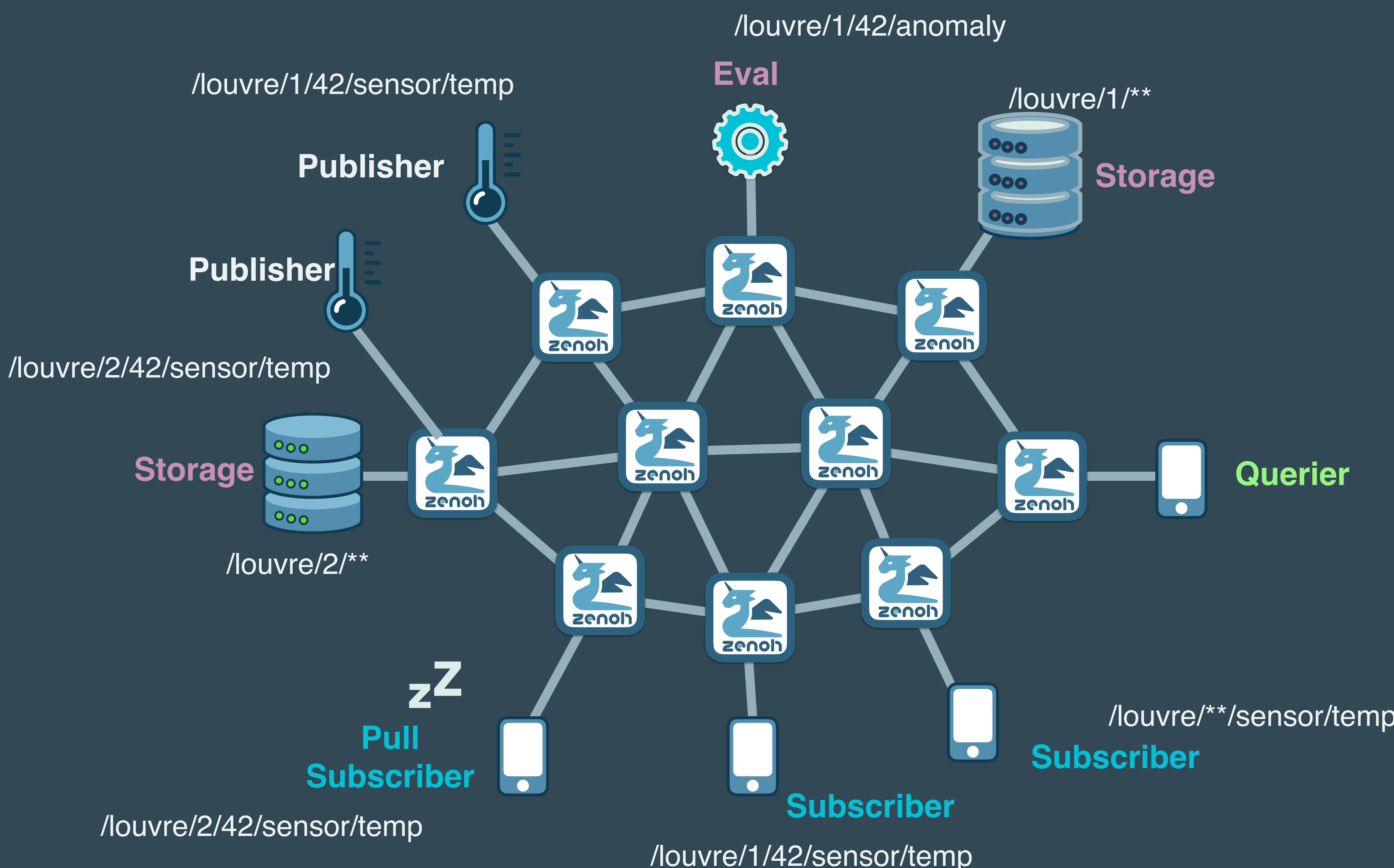
- Currently tested over: UDP, TCP, TLS, QUIC, Ethernet, Thread, Bluetooth, and more coming soon.

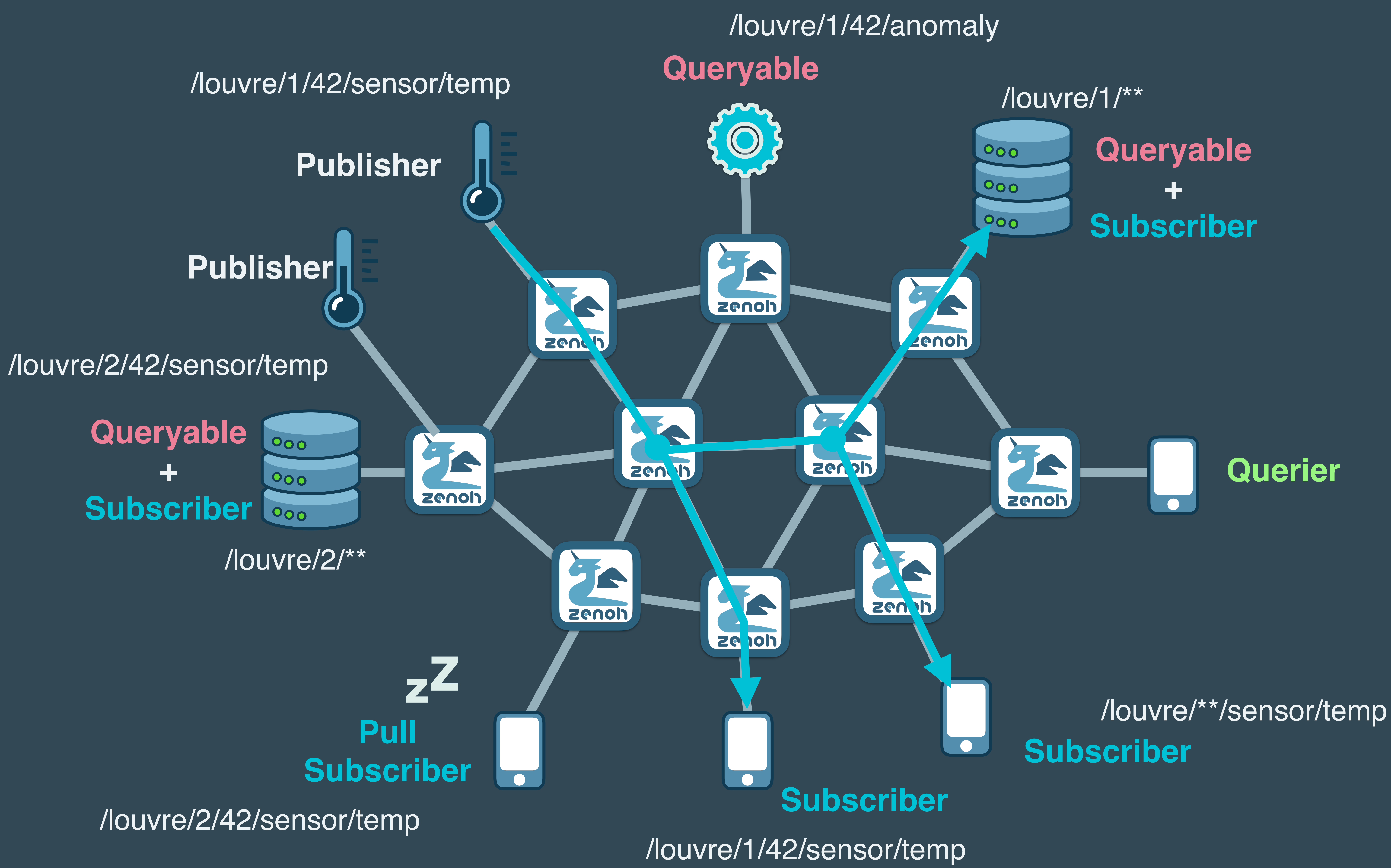
Support both **unicast** and **multicast communication**.

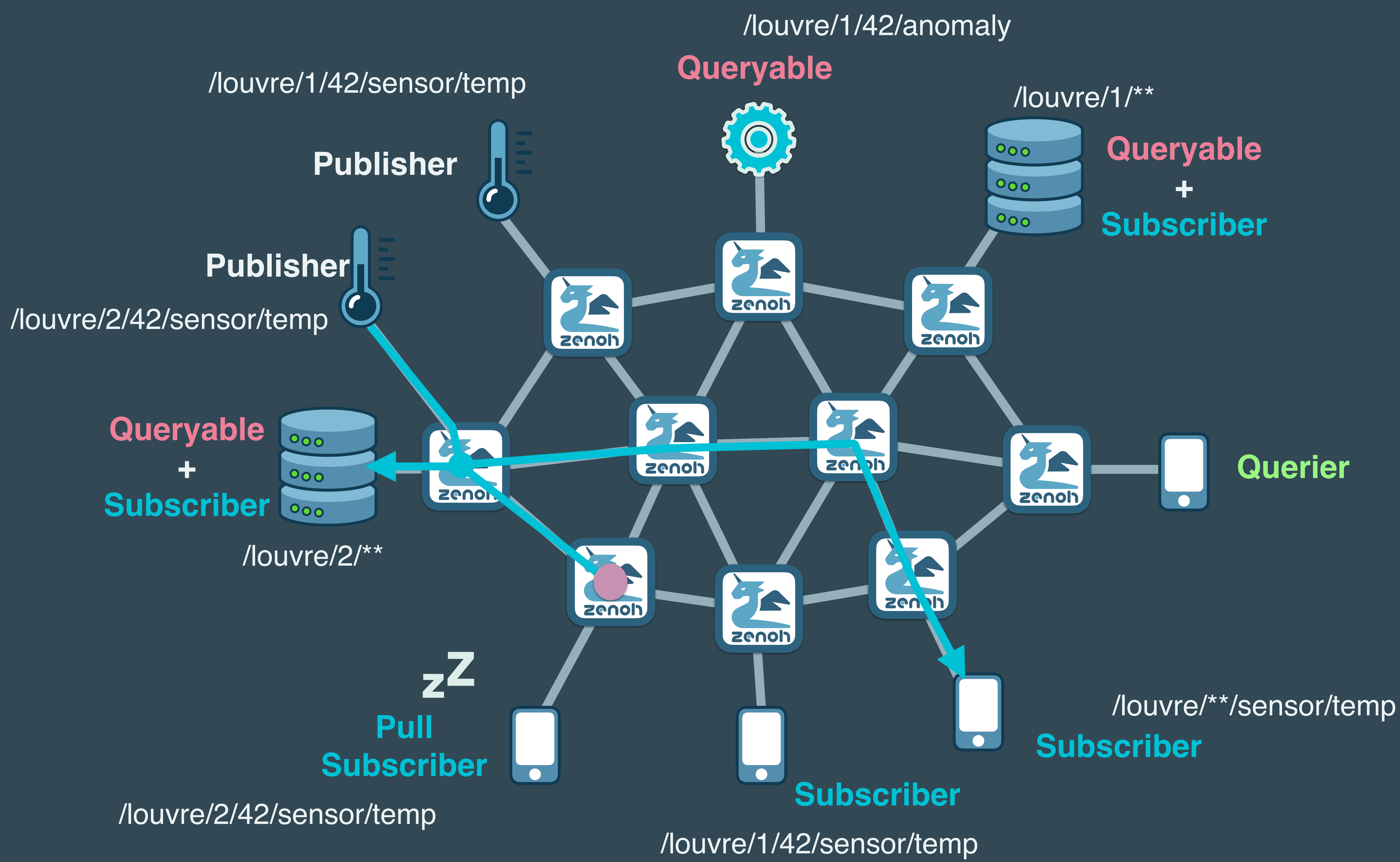
Zenoh session can be single-link or multi-link.

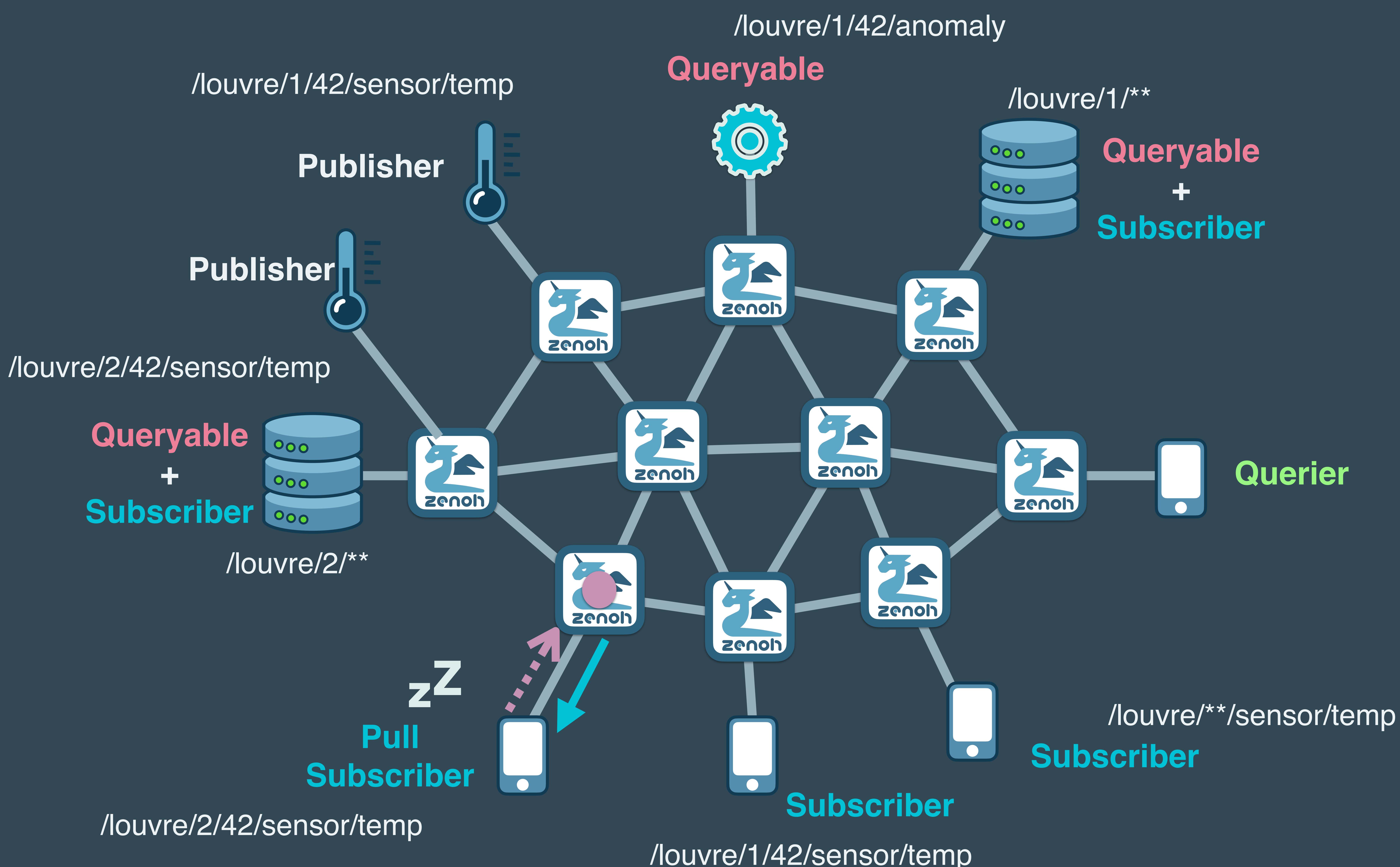


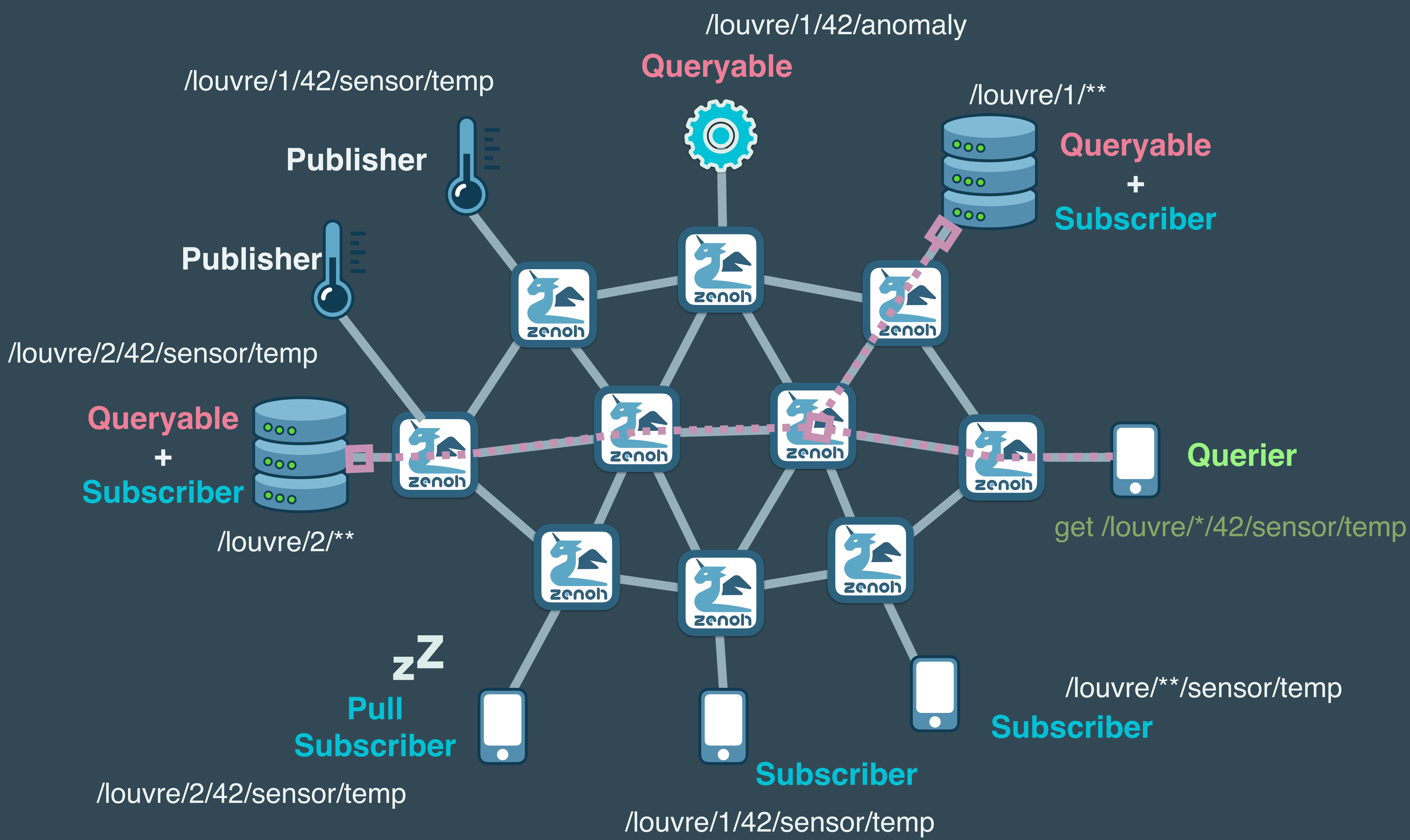
Putting it all Together

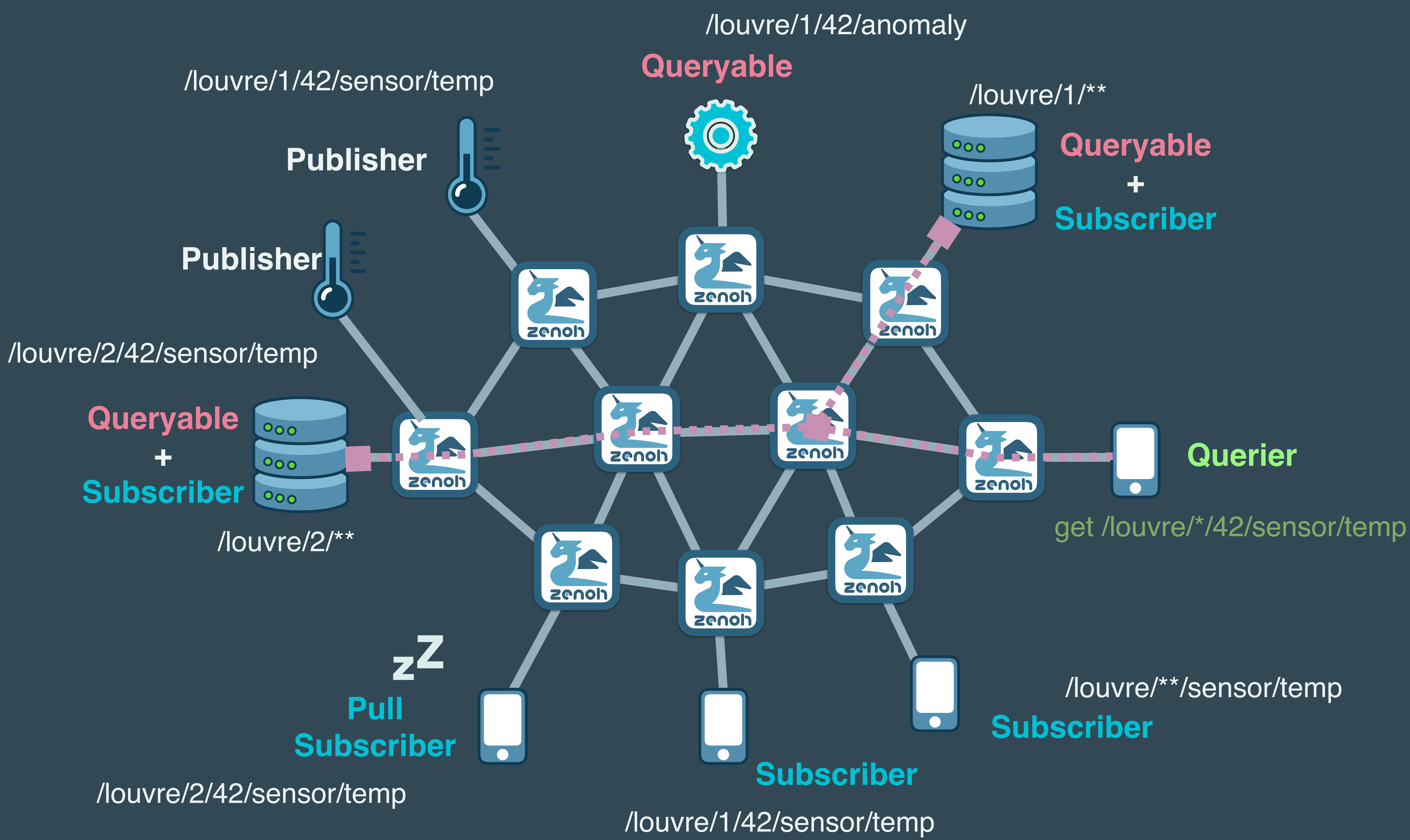

/louvre/1/42/anomaly
Eval
/louvre/1/42/sensor/temp
/louvre/1/**
Publisher
Storage
Publisher
/louvre/2/42/sensor/temp
Storage
Querier
/louvre/2/**
zZ
Pull
Subscriber
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/2/42/sensor/temp
/louvre/1/42/sensor/temp


/louvre/1/42/anomaly
Queryable
/louvre/1/42/sensor/temp
Publisher
/louvre/1/**
Queryable
+
Subscriber
Publisher
/louvre/2/42/sensor/temp
Queryable
+
Subscriber
Querier
/louvre/2/**
zZ
Pull
Subscriber
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/2/42/sensor/temp
/louvre/1/42/sensor/temp


/louvre/1/42/anomaly
Queryable
/louvre/1/42/sensor/temp
/louvre/1/**
Publisher
Queryable
+
Subscriber
Publisher
/louvre/2/42/sensor/temp
Queryable
+
Subscriber
Querier
/louvre/2/**
zZ
Pull
Subscriber
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/2/42/sensor/temp
/louvre/1/42/sensor/temp


/louvre/1/42/anomaly
Queryable
/louvre/1/42/sensor/temp
/louvre/1/**
Publisher
Queryable
+
Subscriber
Publisher
/louvre/2/42/sensor/temp
Queryable
+
Subscriber
Querier
/louvre/2/**
zZ
Pull
Subscriber
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/2/42/sensor/temp
/louvre/1/42/sensor/temp


/louvre/1/42/anomaly
Queryable
/louvre/1/42/sensor/temp
Publisher
/louvre/1/**
Queryable
+
Subscriber
Publisher
/louvre/2/42/sensor/temp
Queryable
+
Subscriber
/louvre/2/**
Querier
get /louvre/*/42/sensor/temp
zZ
Pull
Subscriber
/louvre/2/42/sensor/temp
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/1/42/sensor/temp

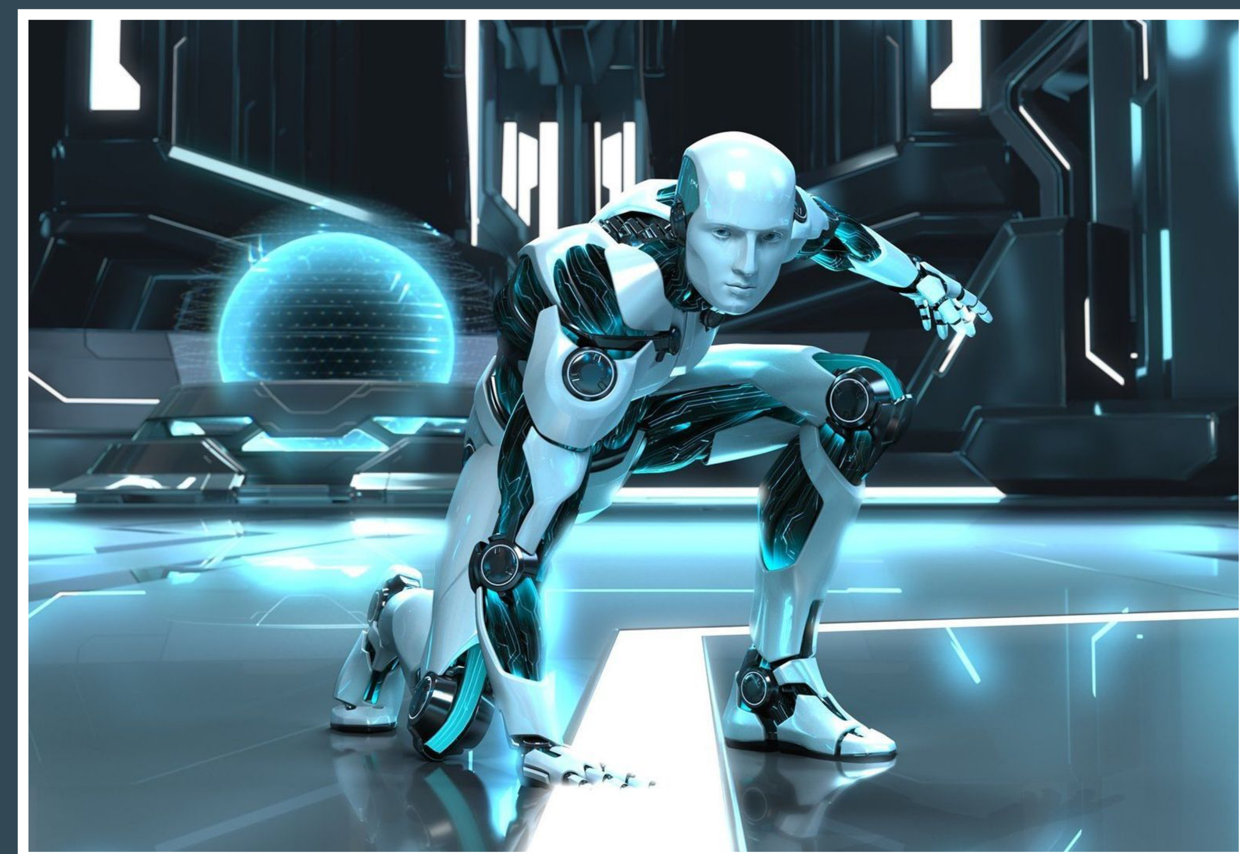


/louvre/1/42/anomaly
Queryable
/louvre/1/42/sensor/temp
Publisher
/louvre/1/**
Queryable
+
Subscriber
Publisher
/louvre/2/42/sensor/temp
Queryable
+
Subscriber
/louvre/2/**
Querier
get /louvre/*/42/sensor/temp
zZ
Pull
Subscriber
/louvre/2/42/sensor/temp
/louvre/**/sensor/temp
Subscriber
Subscriber
/louvre/1/42/sensor/temp

Application Domains

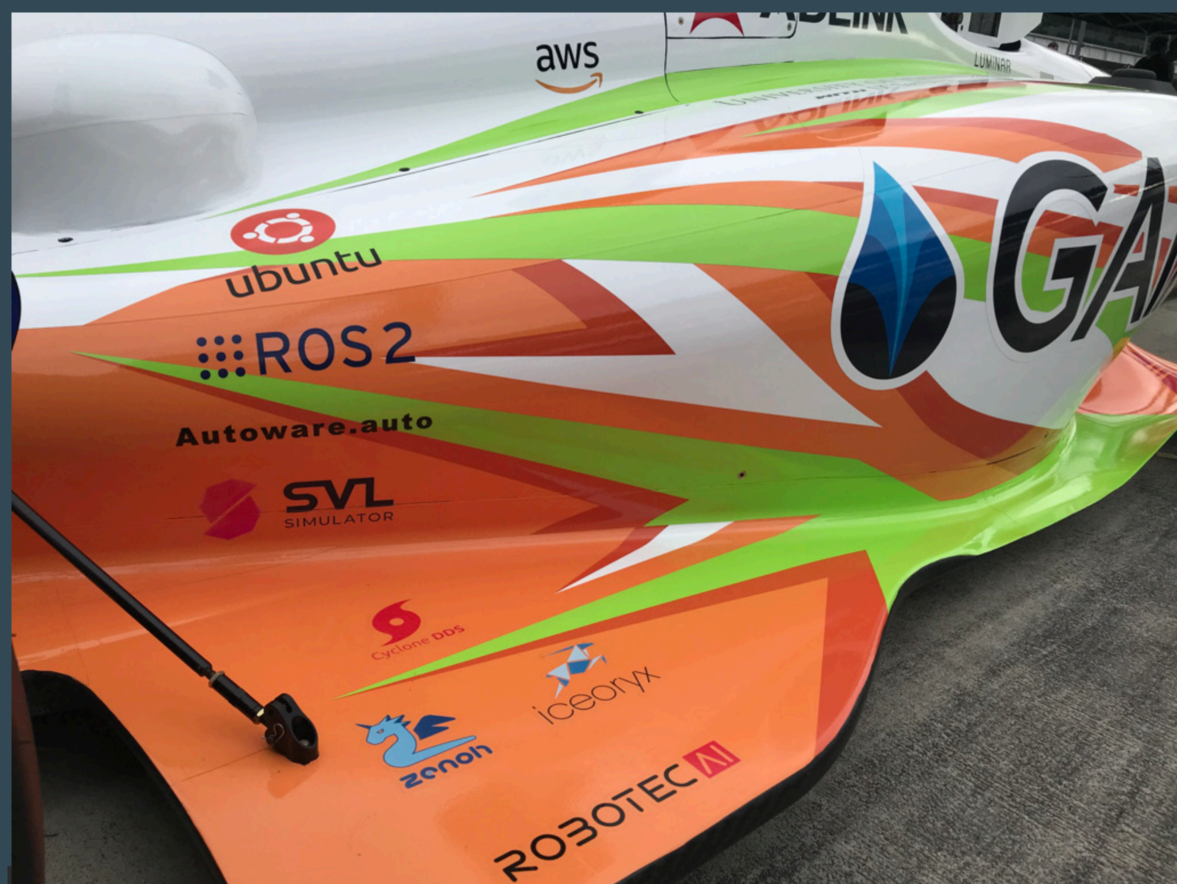




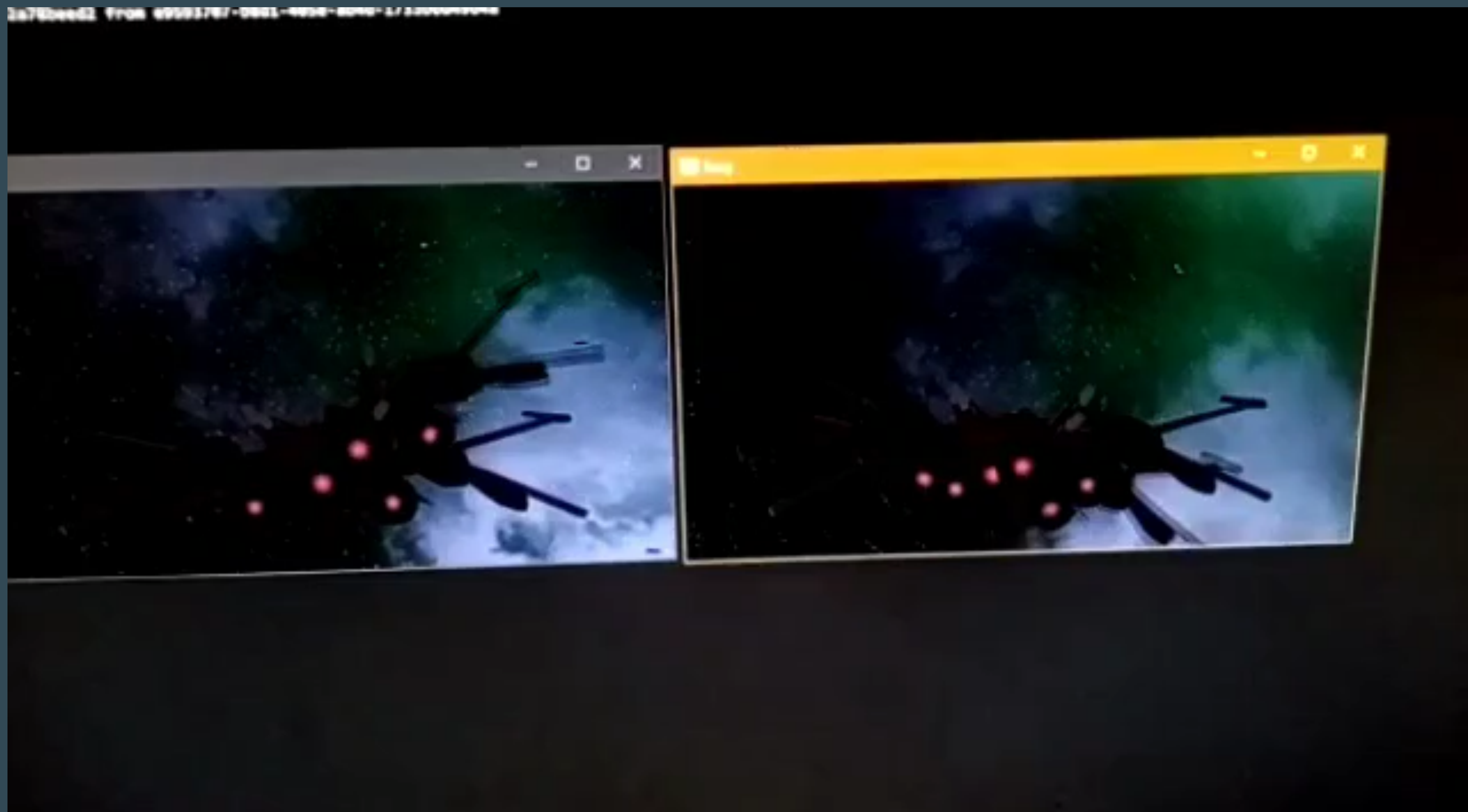




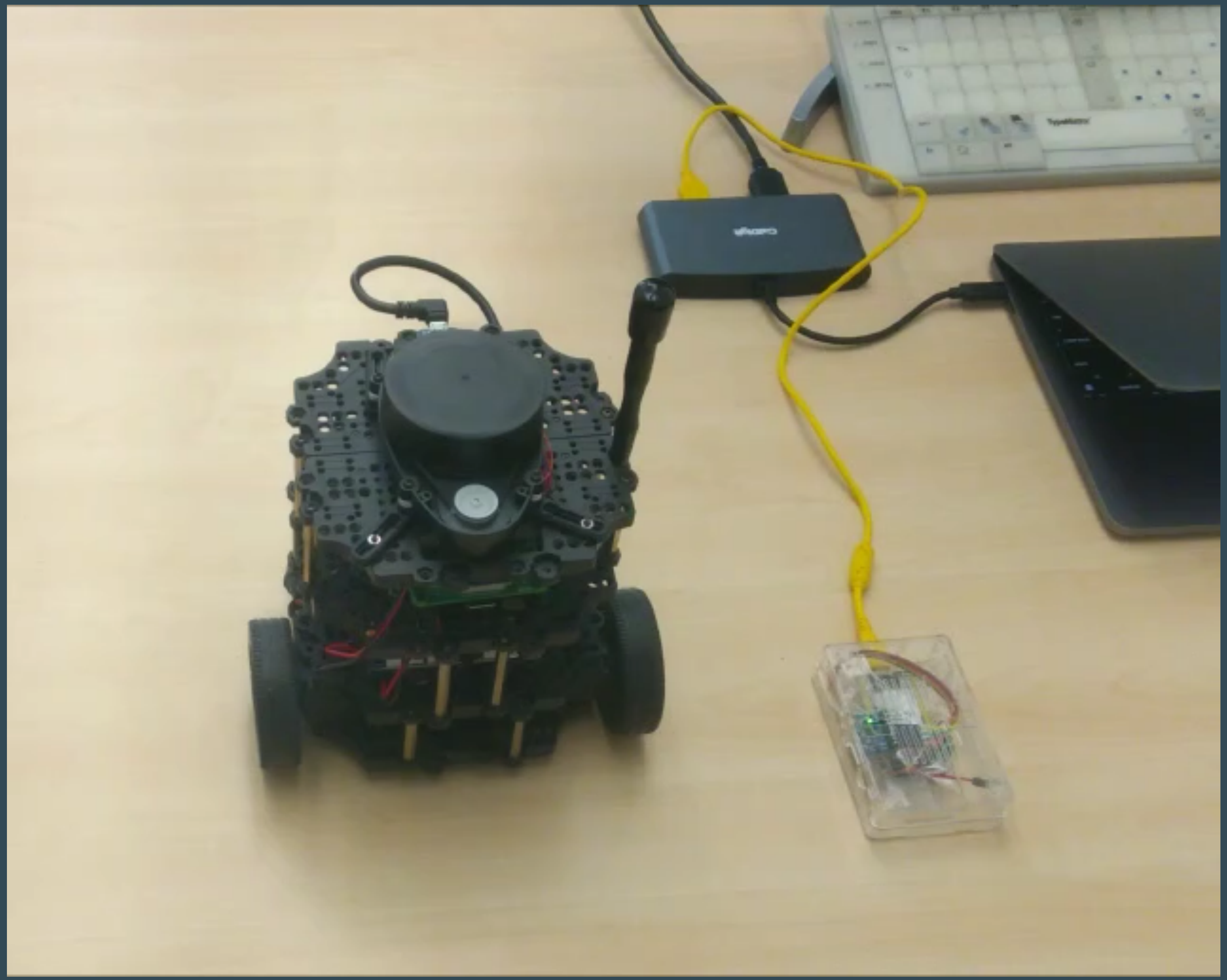
Indie Autonomous Challenge



Online Gaming



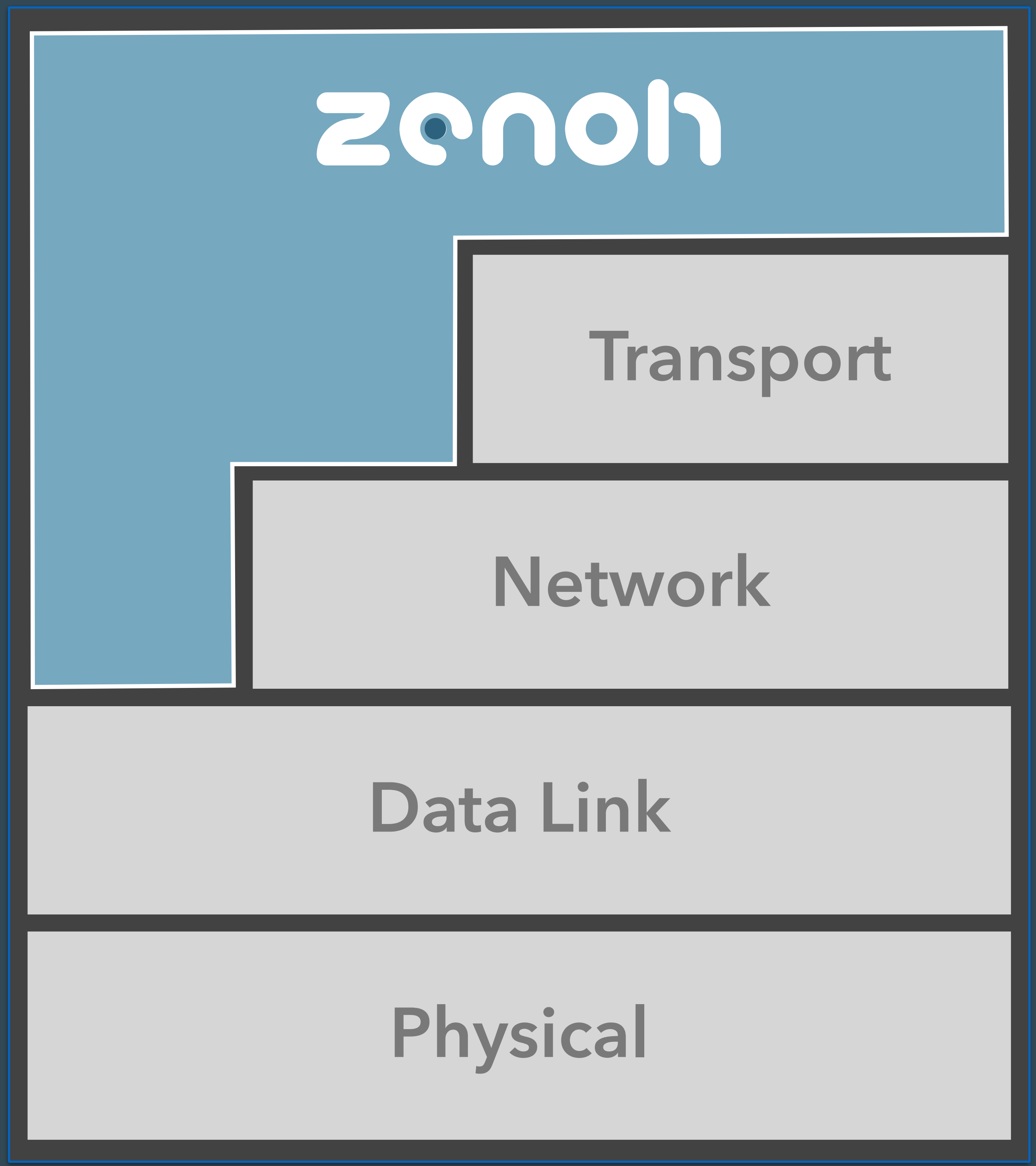
Robotic Systems



Final Thoughts

Protocol Summary Highlights

- **Most wire/power/memory efficient protocol** in the market.
  - Supported by the whole Cloud-to-things continuum (i.e., from cloud servers to microcontrollers).
- Supports **push** and **pull pub/sub** along with **distributed queries** and **remote procedure calls.**
- **Resource keys** are **represented as integers** on the wire, these integer are **local to a session** => good for wire efficiency.
- Supports for **peer-to-peer** and **routed communication.**
- Support for **zero-copy.**
- **Ordered reliable data delivery, fragmentation** and **batching.**
- Minimal **wire overhead** for user data is **4-6 bytes**



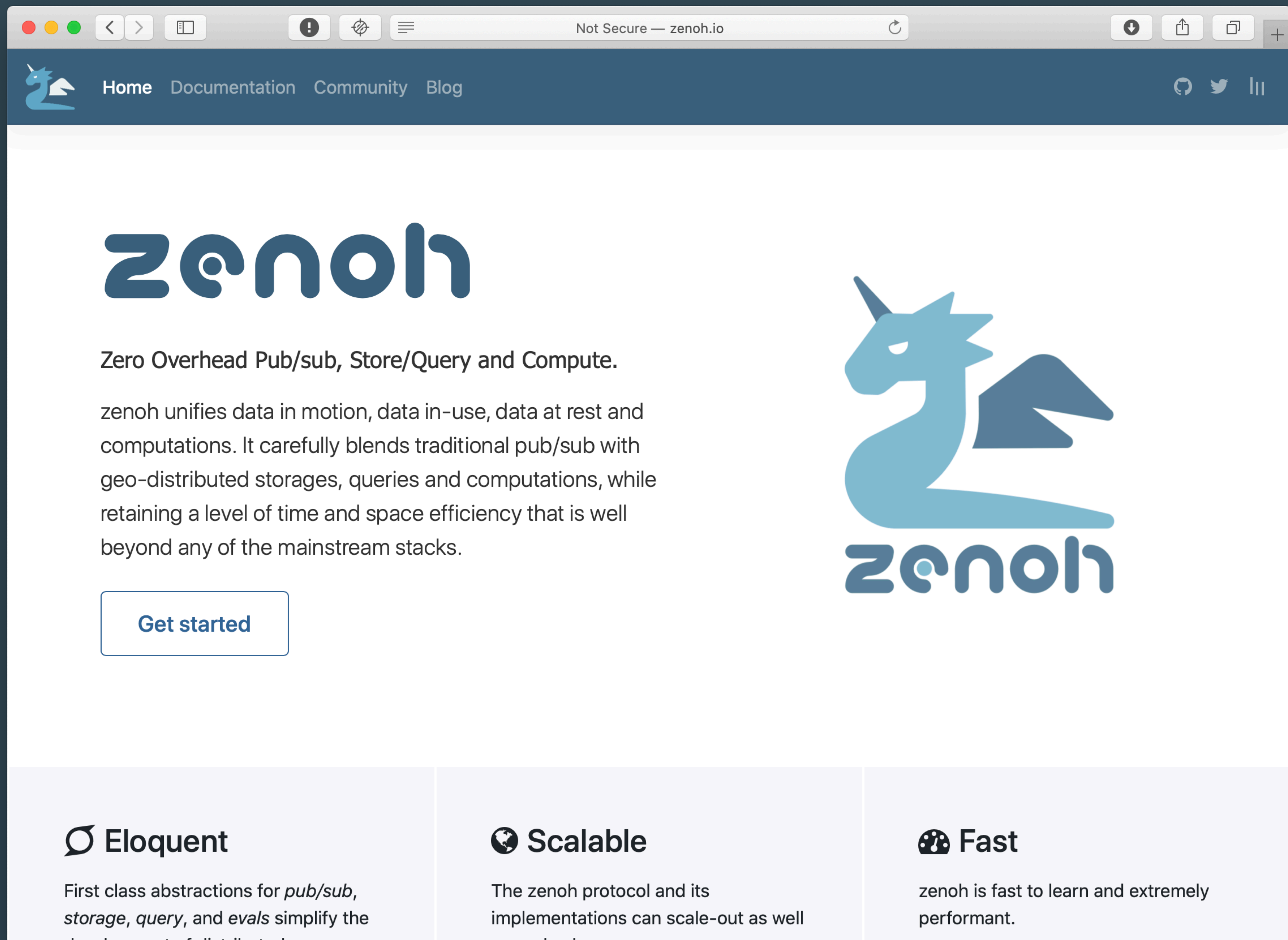
zenoh is an innovative and performant protocol that solves some of they problems at the very core of IoT and Edge Computing

Its open architecture enables to easily expand both storage back-ends as well as protocols that are routed and integrated into the zenoh world

If you like zenoh, **star our repo** and start hacking some code!

References

ECLIPSE FOUNDATION
EDGE | NATIVE



https://zenoh.io/

https://github.com/eclipse-zenoh/

https://gitter.im/atolab/zenoh

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

Appendix

Code Lab: v0.5.0-beta.9

APIs

REST API

zenoh runs on any RUST supported platform plus a few embedded targets such as Zephyr and Arduino. Zenoh also offers a REST API for programming and administration.

zenoh·rust

zenoh·c

zenoh·pico

zenoh·py

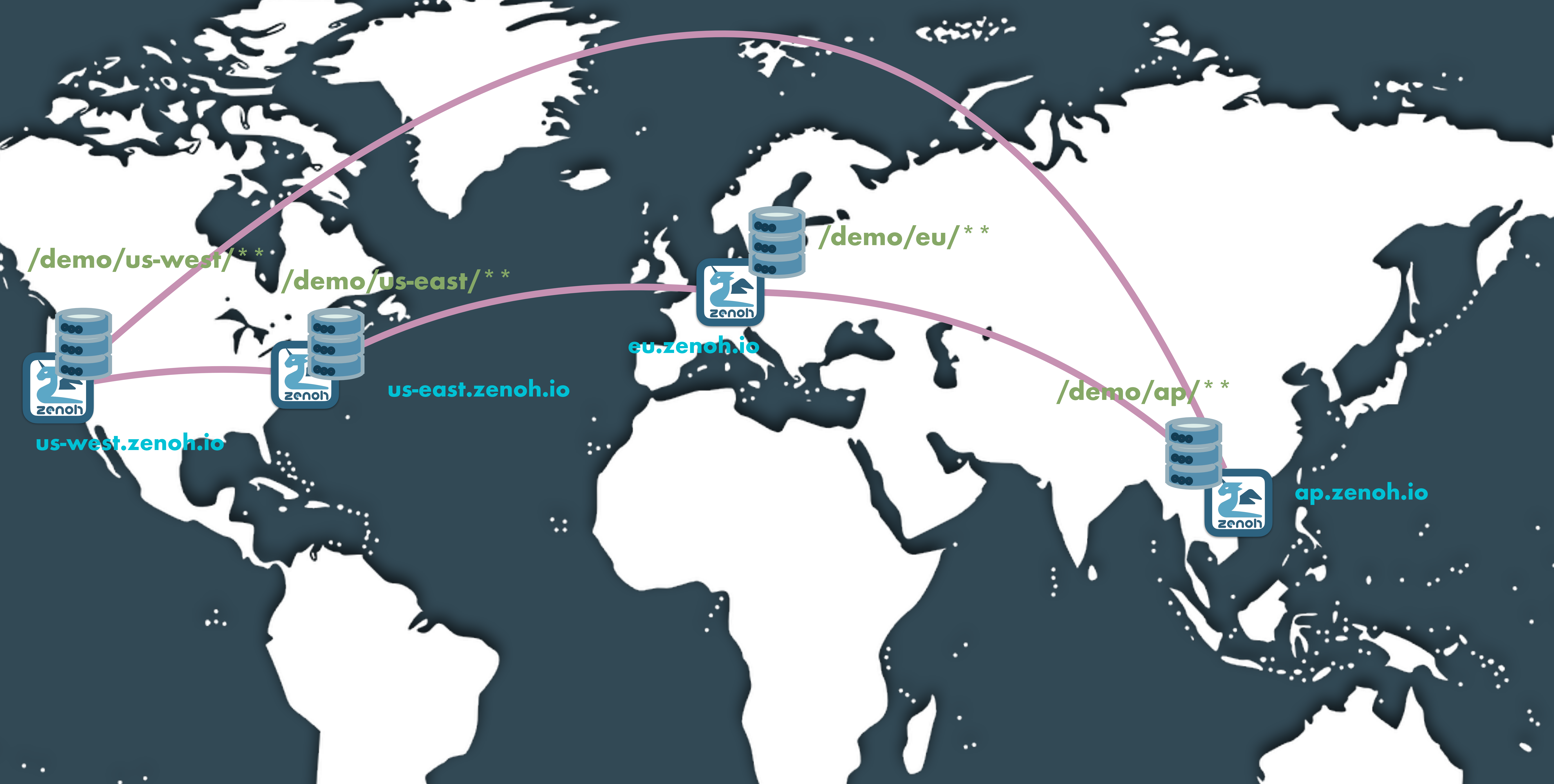
In progress

zenoh·c#

Coming Up

zenoh·java

zenoh·go



Example:

- Put data: `curl -X PUT -d 'Hello World!' http://us-west.zenoh.io:8000/demo/eu/test`
- Get data: `curl http://ap.zenoh.io:8000/demo/*/test`

Greetings

```
from zenoh import Zenoh

# Get a zenoh session
zs = Zenoh({‘peer’: ‘tcp/eu.zenoh.io:7447’})
z = zs.workspace()

# play around
z.put(“/demo/eu/greet/italian”, “Ciao!”)
```

More Greetings...

```
z.put(“/demo/us-east/greet/american”, “Hi!")
z.put(“/demo/us-west/greet/american”, “What’s Up!”)
z.put(“/demo/ap/greet/japanese”, “Aisatsu!”)
z.put(“/demo/eu/greet/portuguese”, “Viva!”)
```

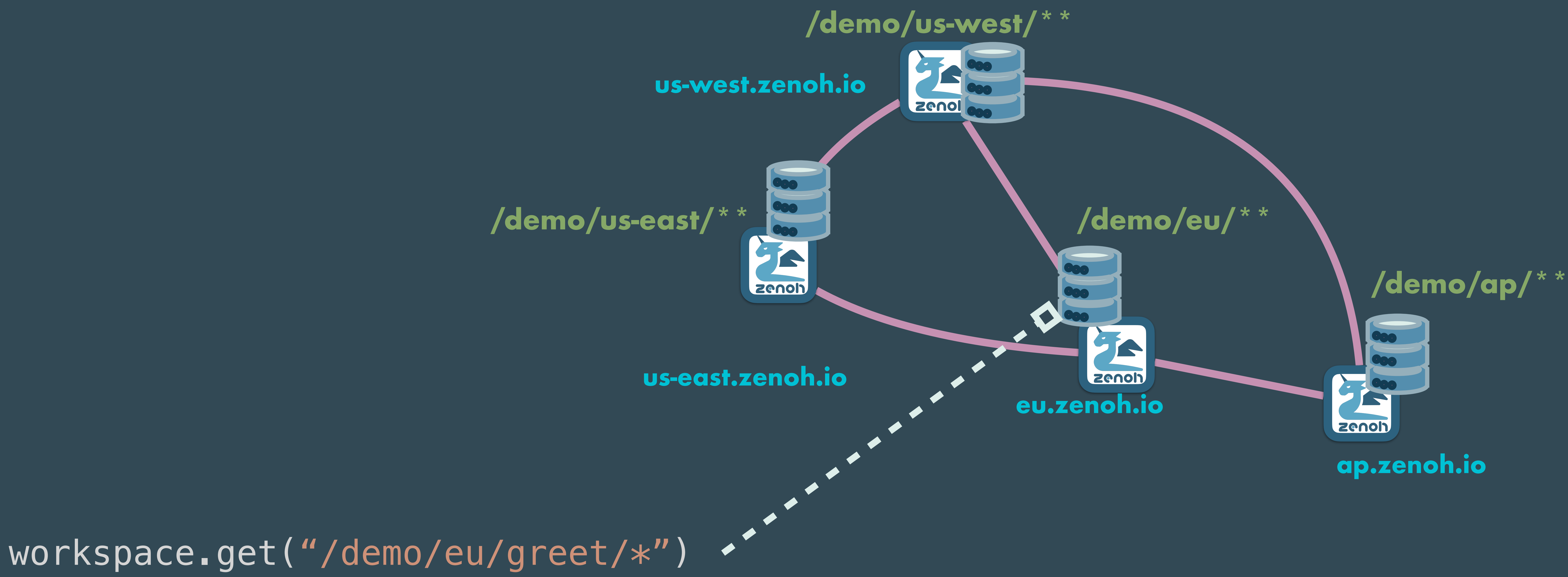
Getting Greetings

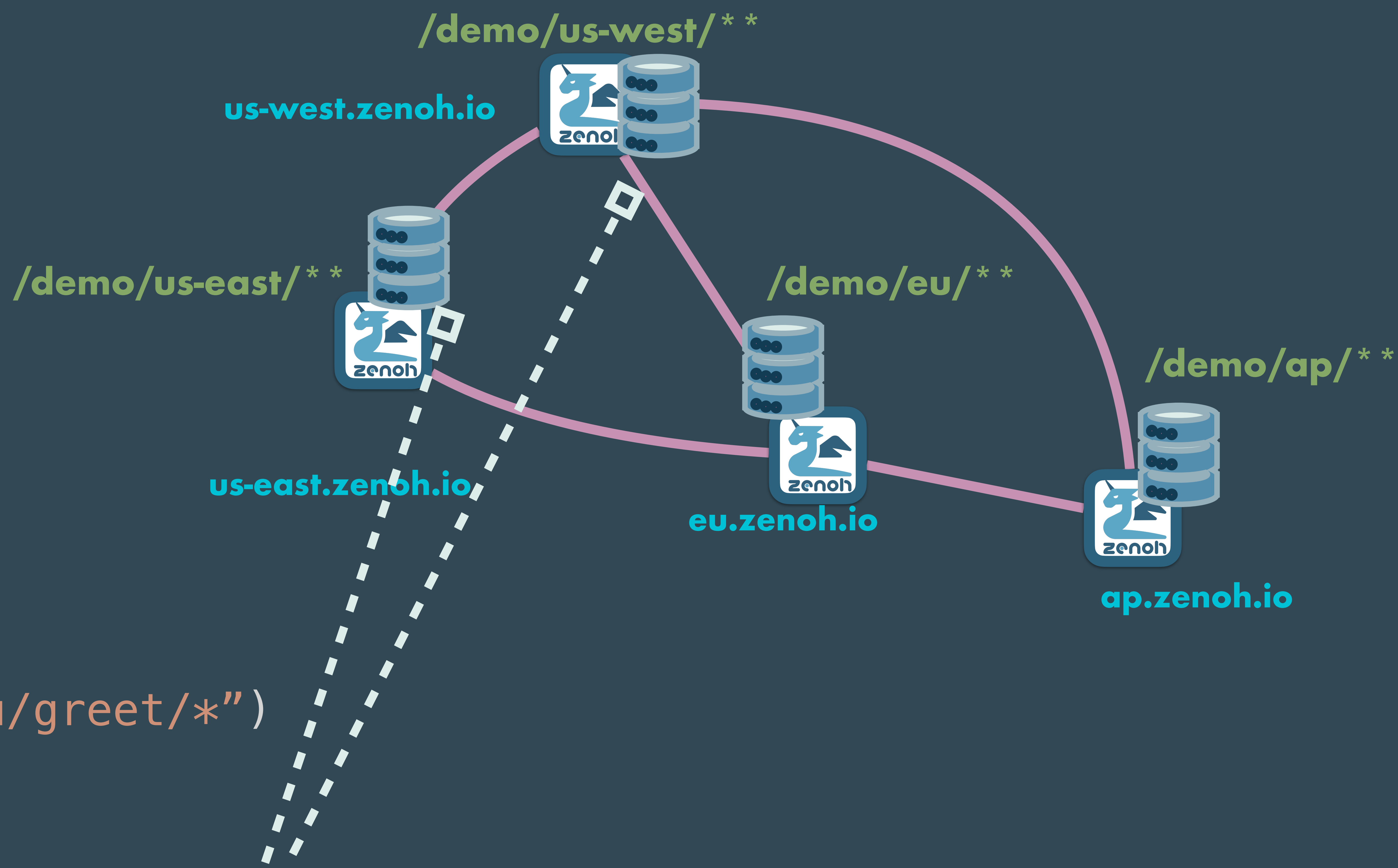
```
from zenoh import Zenoh, ChangeKind

# Get a zenoh session
zs = Zenoh({‘peer’: ‘tcp/eu.zenoh.io:7447’})
z = zs.workspace()

# Define the listener
def listener(change):
    print("{} : {} (encoding: {} , timestamp: {})”
      .format(change.path,
          "DELETED" if change.kind == ChangeKind.DELETE
                    else change.value.get_content(),
          "none"    if change.kind == ChangeKind.DELETE
                    else change.value.encoding_descr(),
          change.timestamp))
```

```
z.subscribe(“/demo/*/greet/*“, listener)
```

Finding out Greetings

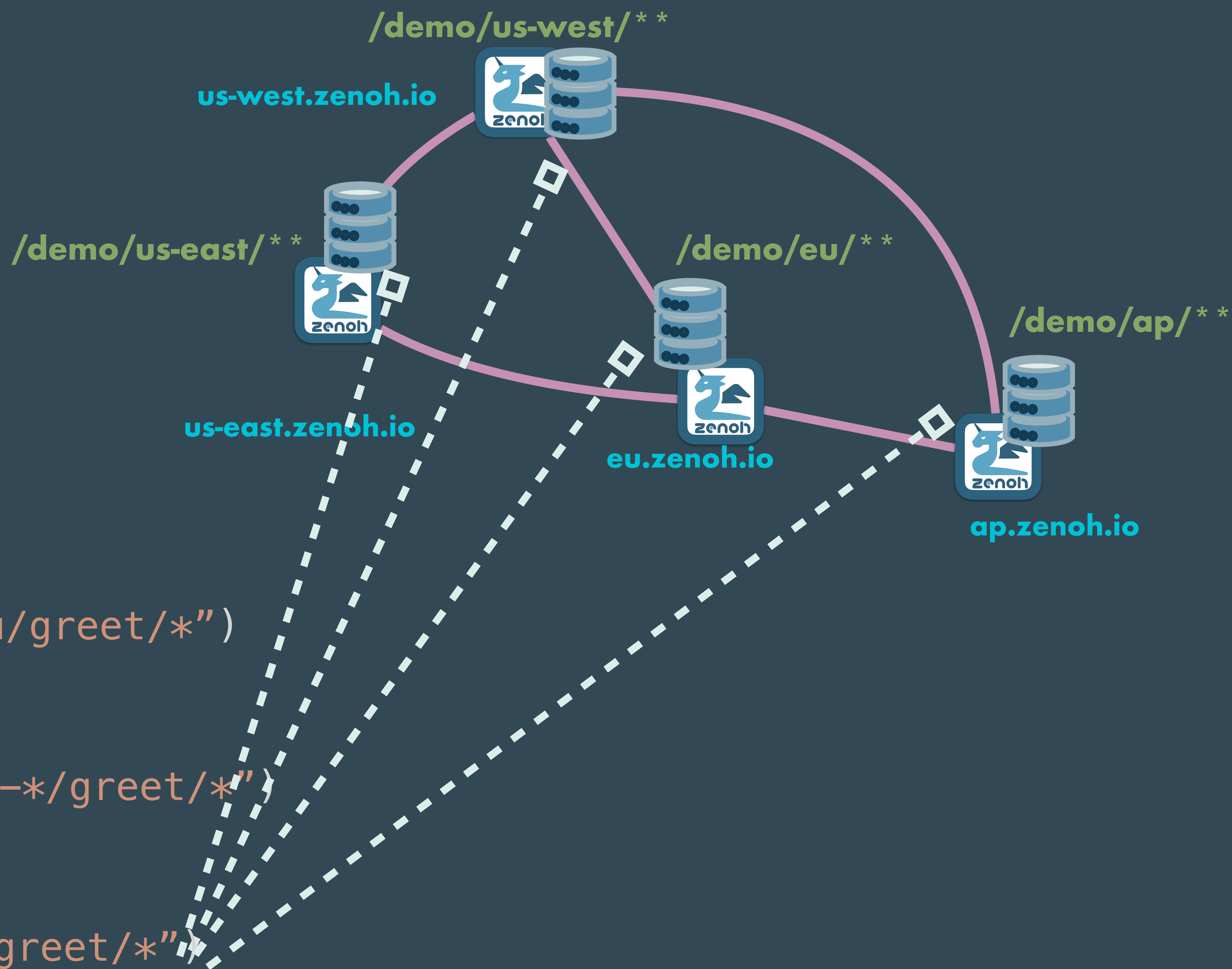
```
# How do people greet in EU?
workspace.get(“/demo/eu/greet/*”)

# How about American?
workspace.get(“/demo/us-*/greet”)

# Just get me all you know about greeting…
workspace.get(“/demo/*/greet/*”)
```


/demo/us-west/* *
us-west.zenoh.io
/demo/us-east/* *
/demo/eu/* *
/demo/ap/* *
us-east.zenoh.io
eu.zenoh.io
ap.zenoh.io
workspace.get(“/demo/eu/greet/*”)




```
workspace.get(“/demo/eu/greet/*”)
```

```
workspace.get(““/demo/us-*/greet/*””)
```



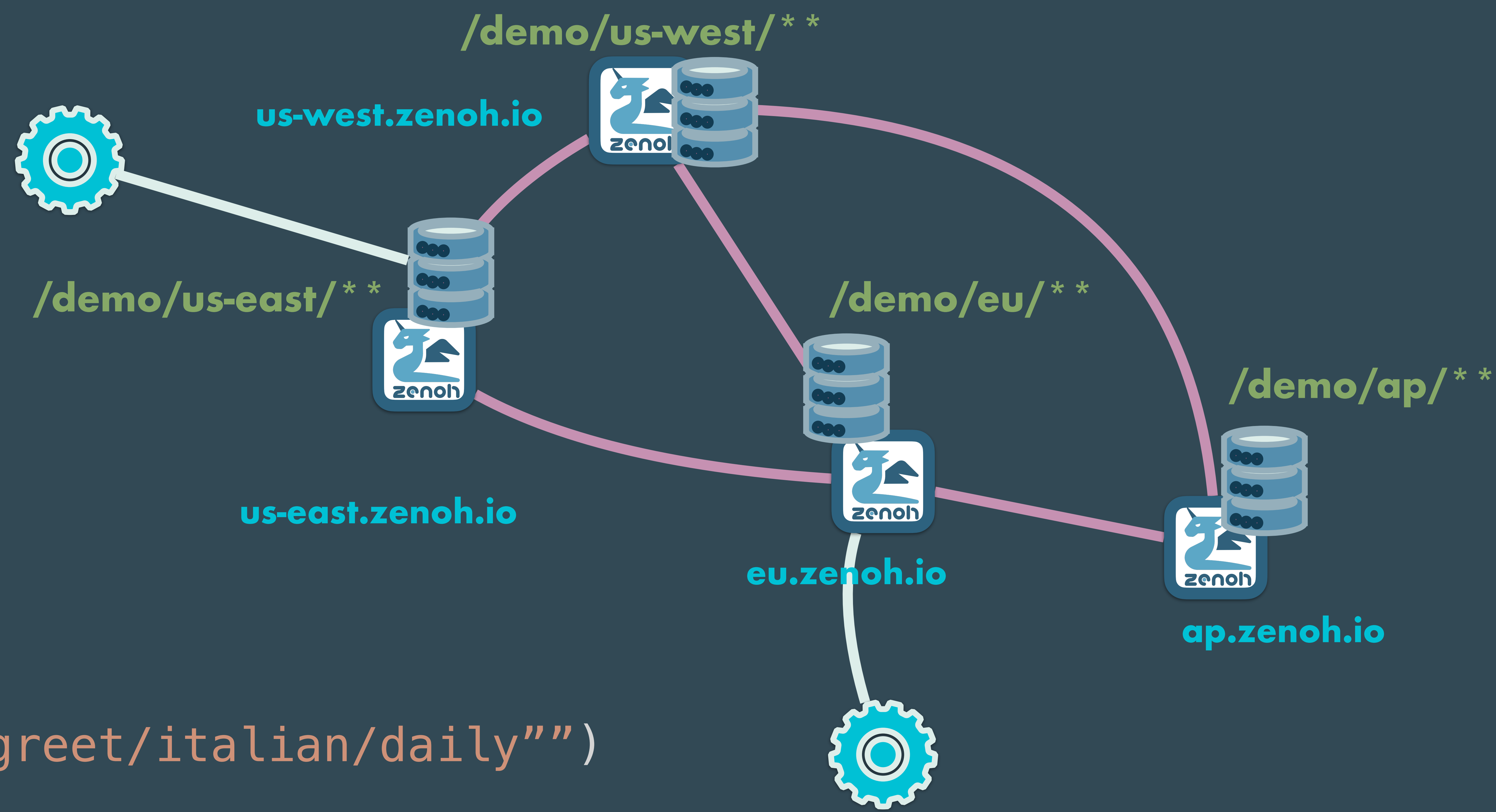

```
workspace.get("/demo/eu/greet/*")

workspace.get("/demo/us-*/greet/*")

workspace.get("/demo/*/greet/*")
```

Greeting of the Day

Imagine you want to do a greeting of the day that each time somebody tries to query it generates a random quote, or a daily quote, etc.

We could do that with an eval, here is how:

```
def quote_eval(request):
    make_a_cute_quote(request)

z.register_eval("/demo/*/greet/*/daily", quote_eval)
```




```
workspace.get(““/demo/*/greet/italian/daily””)
```

```
workspace.get(“/demo/*/greet/american/daily”)
```

To resolve this query zenoh will pick the eval that happens to be “closer” to the querier.

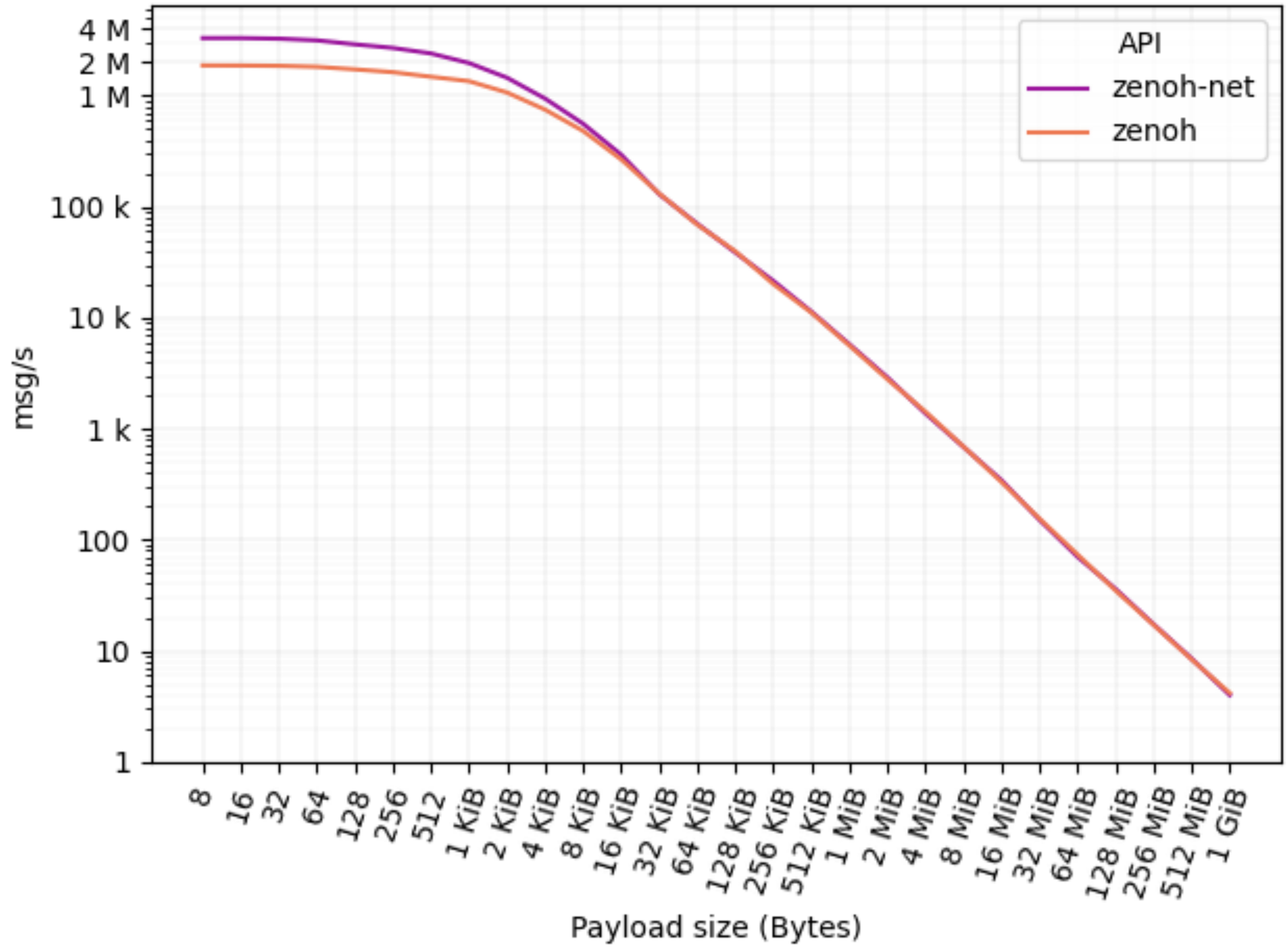
This is true in general as queries can target at the same time evals and storages.

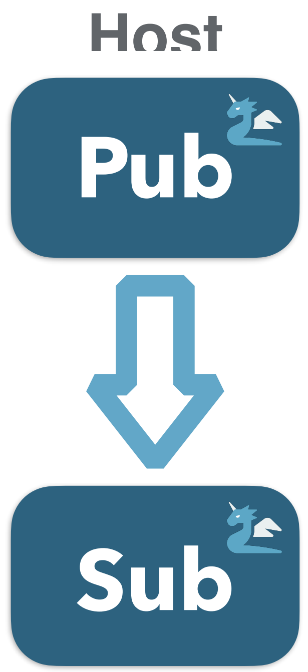
Appendix

Performance

Throughput (msg/s)

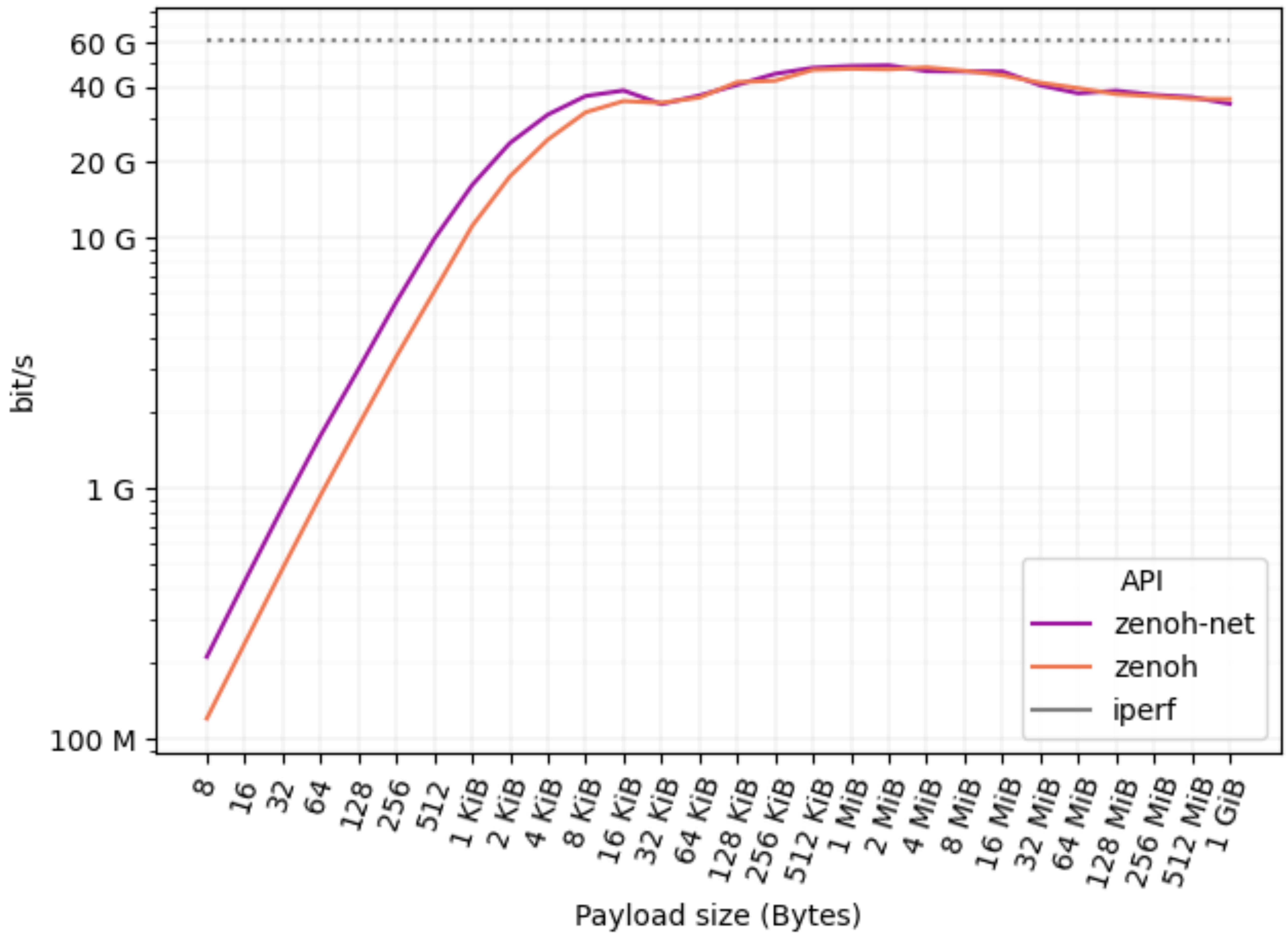
Test ran on 10/07/2021 on
Centos 8
AMD Ryzen
32GB RAM
100Gbps ETH

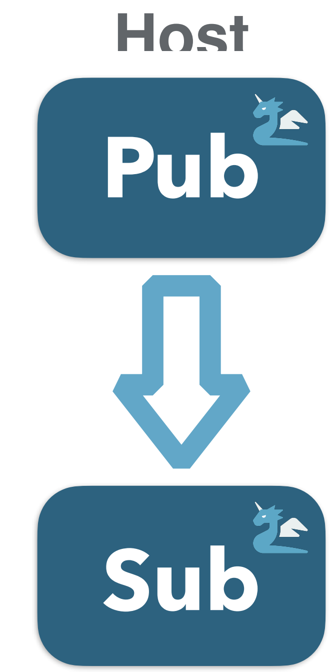




Throughput (Gb/s)

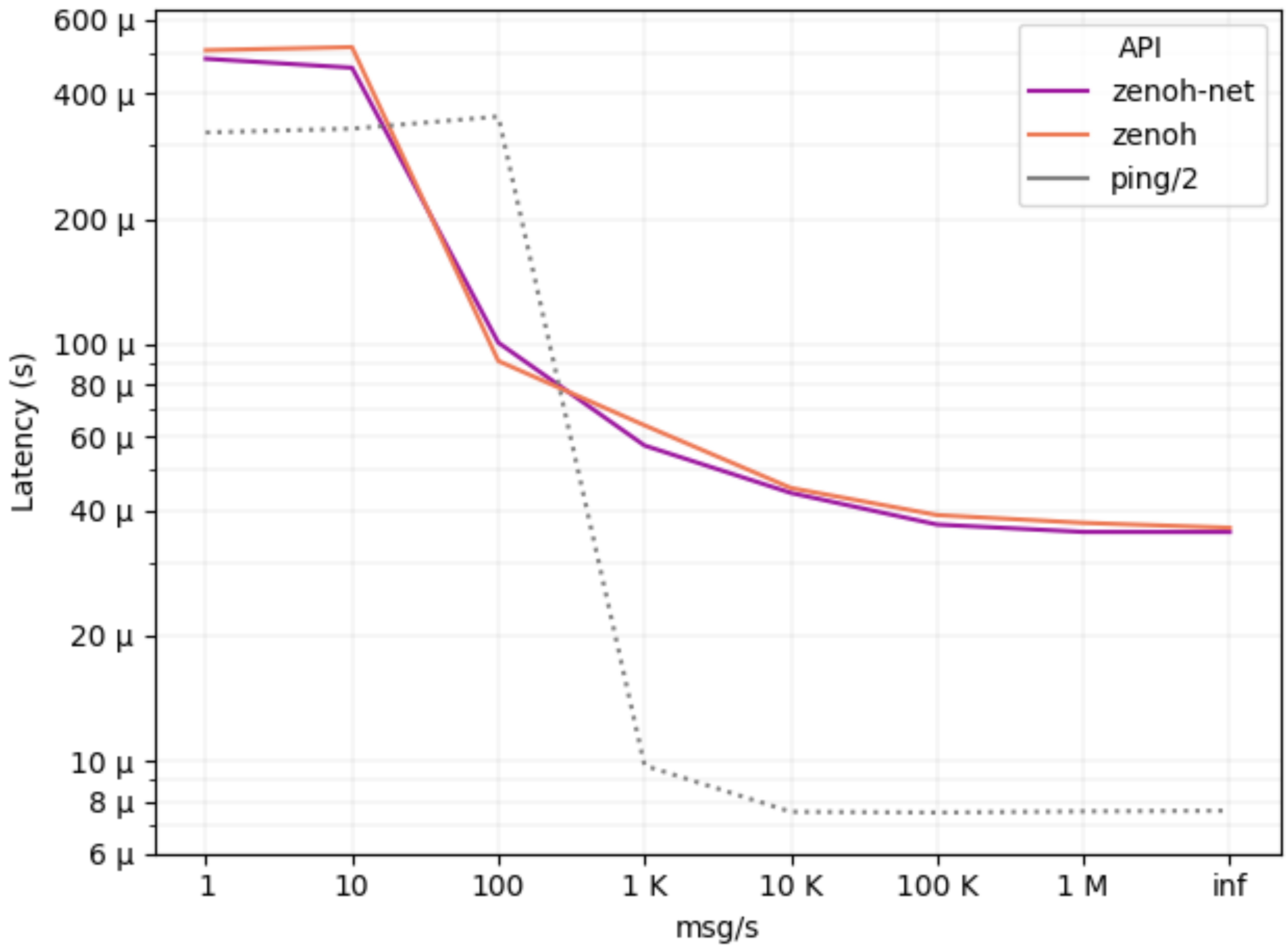
Test ran on 10/07/2021 on
Centos 8
AMD Ryzen
32GB RAM
100Gbps ETH





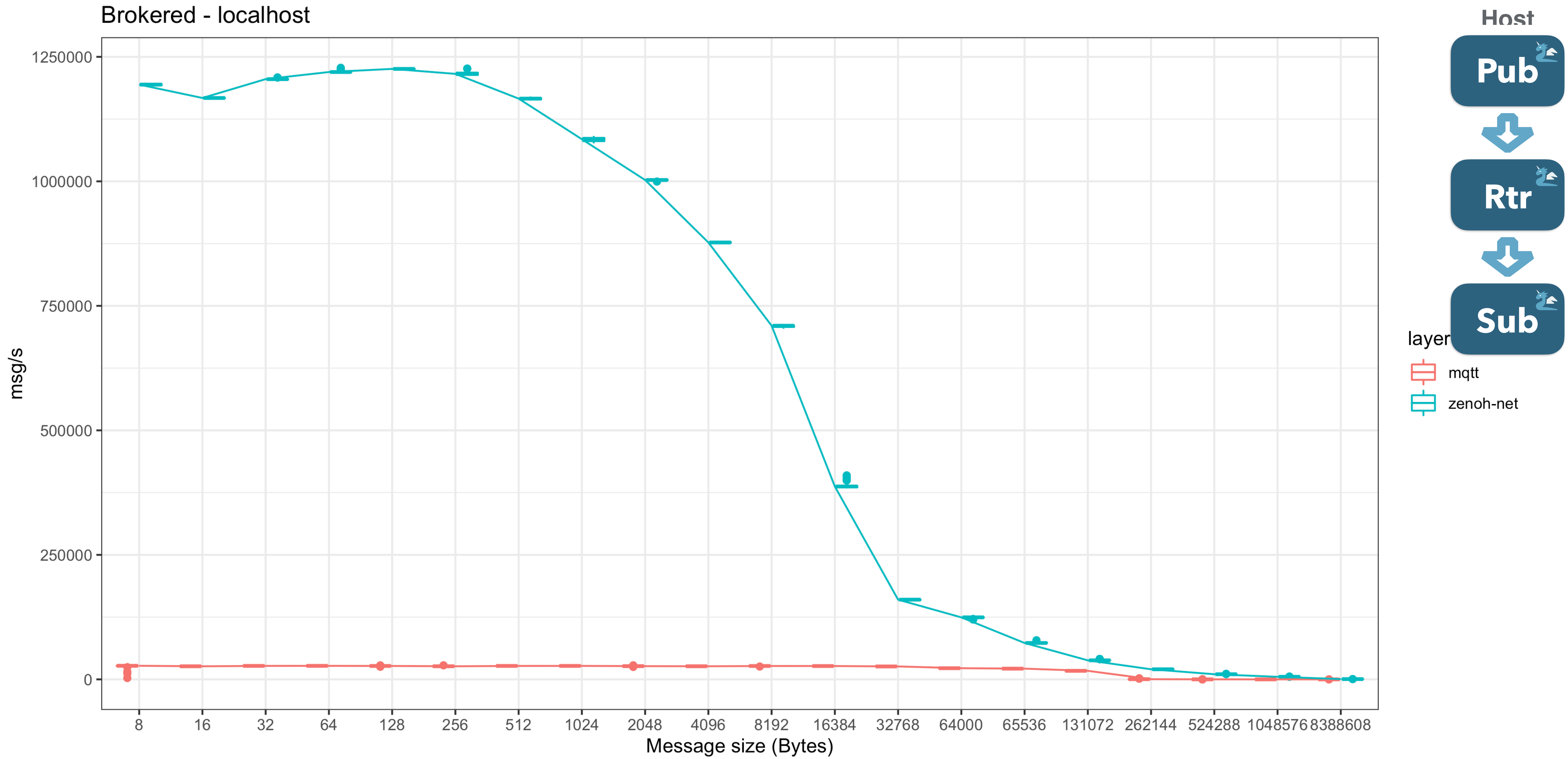
Latency

Test ran on 10/07/2021 on
Centos 8
AMD Ryzen
32GB RAM
100Gbps ETH



Zenoh vs MQTT (msg/s)

Test ran on 11/03/2021 on
Centos 8
AMD Ryzen
32GB RAM



Zenoh vs MQTT (Gb/s)

Test ran on 11/03/2021 on
Centos 8
AMD Ryzen
32GB RAM