Achieving Eventual Consistency in Zenoh

Sreeja S. Nair, PhD

Senior Technologist
ZettaScale Technology

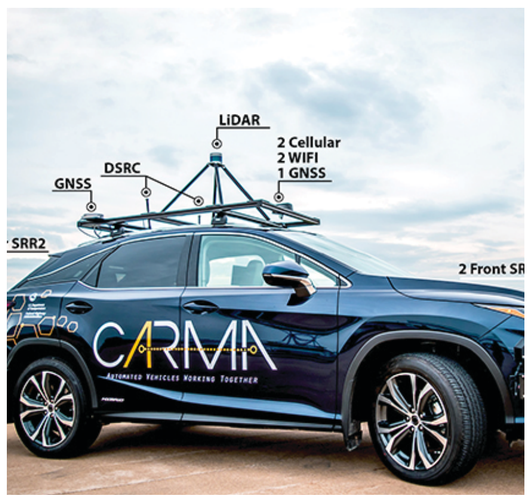
RainbowFS workshop
28/03/2022

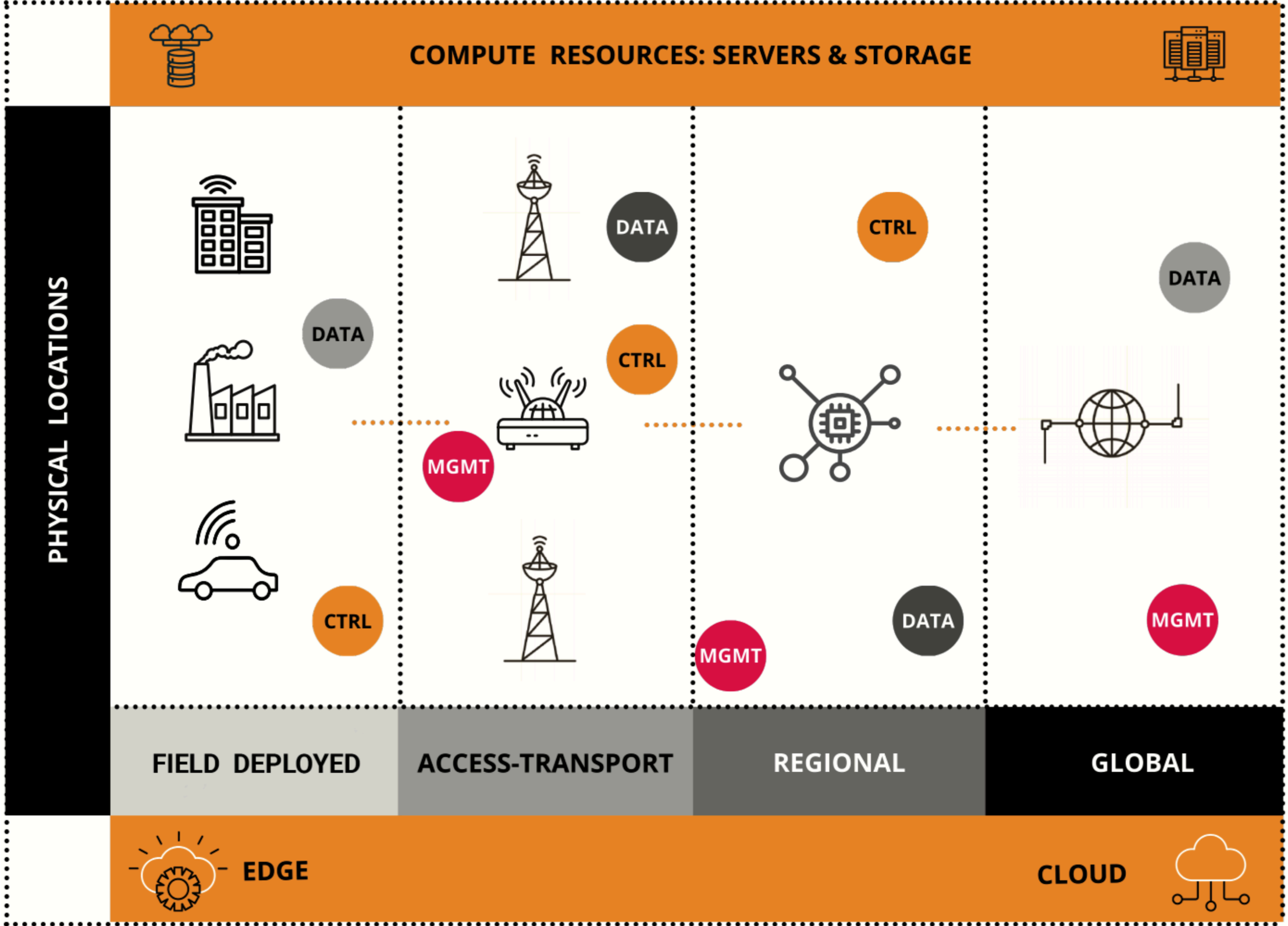
ZettaScale

Mission

Bring to every connected human and machine the
unconstrained freedom to communicate, compute and store
— anywhere, at any scale, efficiently and securely

Powering





Eclipse Foundation, *"From DevOps to EdgeOps: A Vision for Edge Computing"*, 2021

COMPUTE RESOURCES: SERVERS & STORAGE

PHYSICAL LOCATIONS

Produce

Distribute

Compute

Store

Push

Push

Push

Push

DATA

CTRL

DATA

CTRL

Push

MGMT

Retrieve

Pull

Push

Push

Pull

Push

MGMT

DATA

MGMT

Consume

Compute

Store

Compute

EDGE

CLOUD

Eclipse Foundation, *"From DevOps to EdgeOps: A Vision for Edge Computing"*, 2021

zetta scale

Zenoh

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

Provides a **location-transparent API** for high performance **pub/sub** and **distributed queries**

Facilitates data representation **transcoding**, **geo-distributed storage** and **distributed computed values**

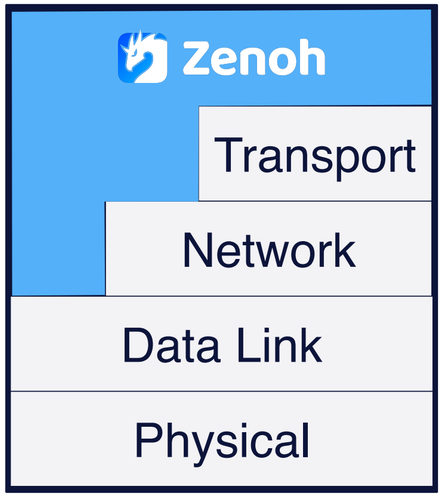
“Some people want it to happen, some wish it would happen, others make it happen.” – Michael Jordan

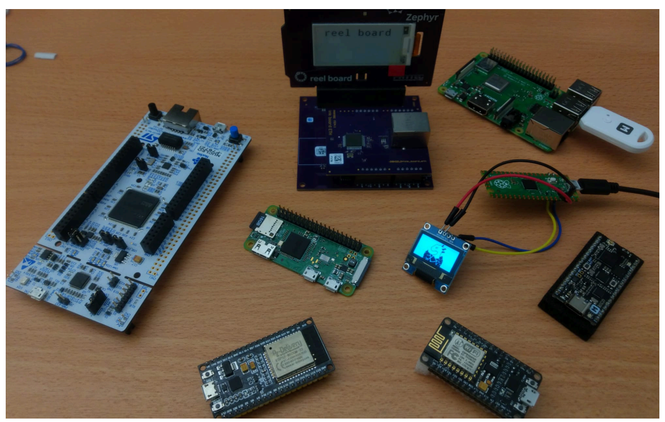
Platform

Native libraries and **API bindings** for many programming languages

Over various **network technologies**: from **transport layer** to **data link**

On **embedded** and **constrained devices**





Extensions

Storage plugin

RocksDB

InfluxDB

MariaDB

Filesystem

Memory

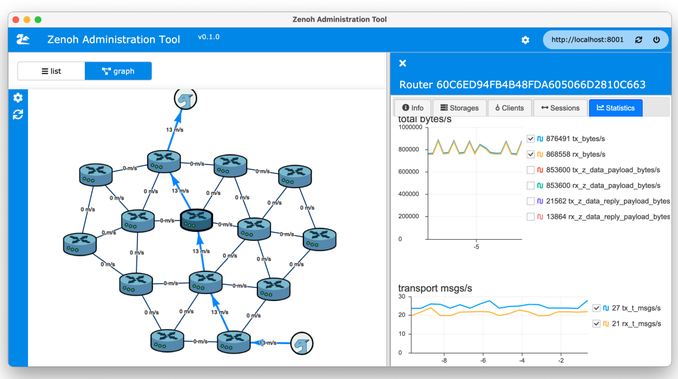
Pluggable backends ...

{ REST } plugin

HTML5 Server Sent Events

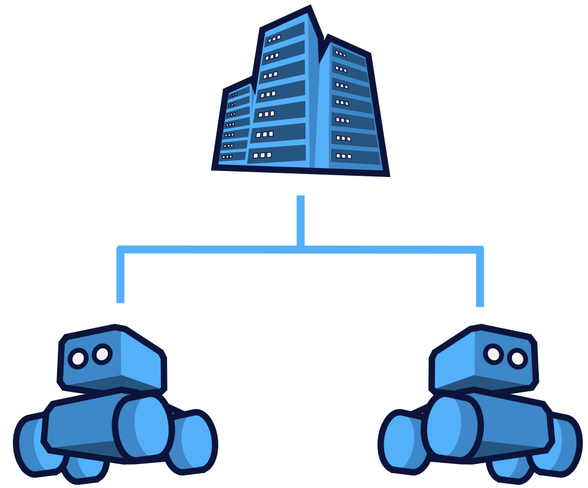
HTML SSE

Administration & monitoring



DDS plugin

V2X/R2X connectivity



ROS2

Topologies

Peer-to-peer

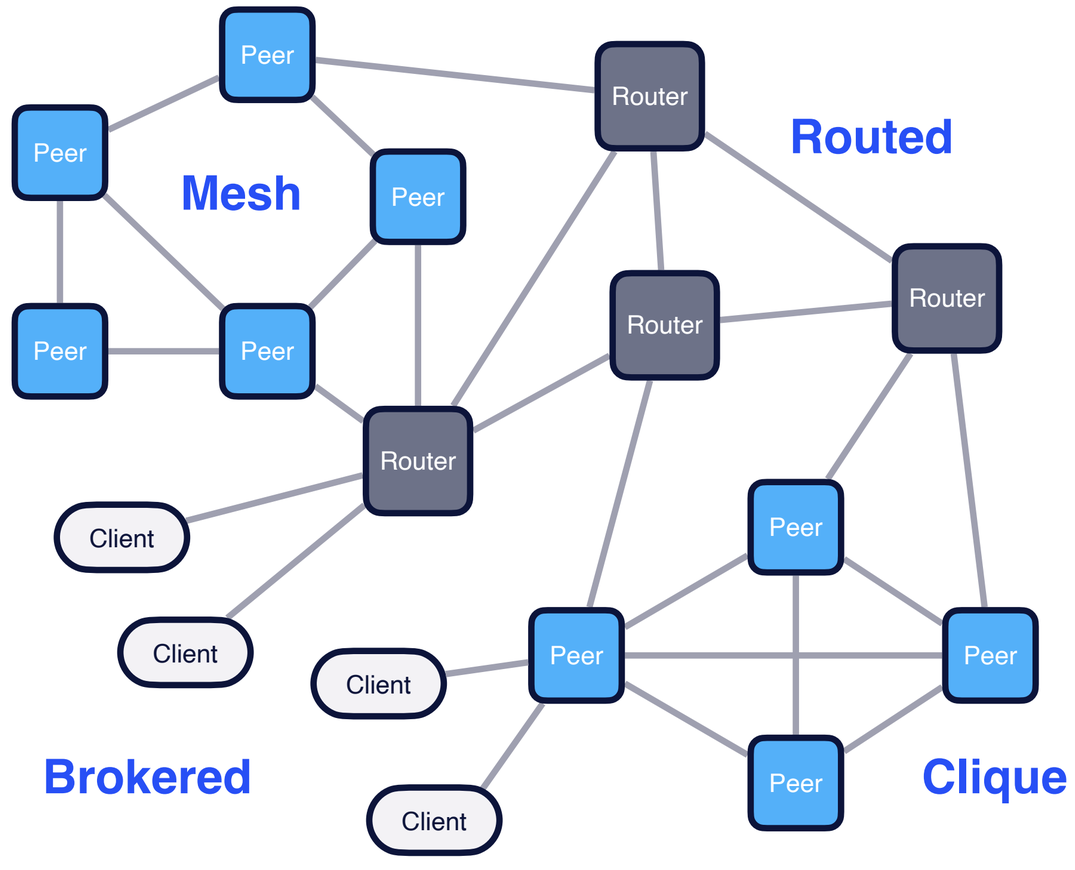
Clique and mesh topologies

Brokered

Clients communicate through a router or a peer

Routed

Routers forward data to and from peers and clients



Abstractions

Resource : A **named data**, in other terms a (key, value)

/home/kitchen/sensor/temp, 21.5
/home/bedroom/sensor/temp, 19

Key expression : An **expression** identifying a set of keys

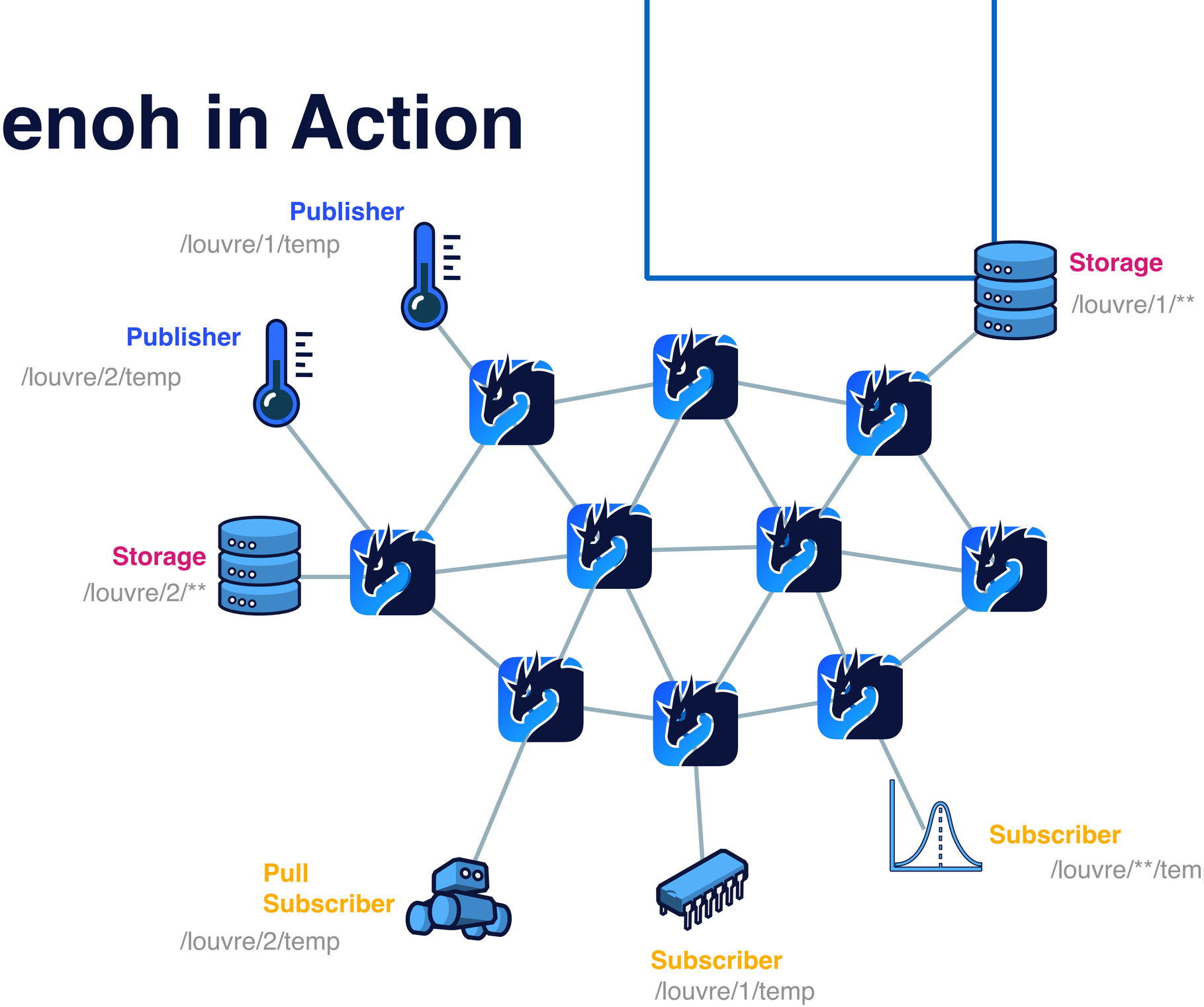
/home/kitchen/sensor/temp
/home/kitchen/**

Publisher : A **spring** of values for a key expression

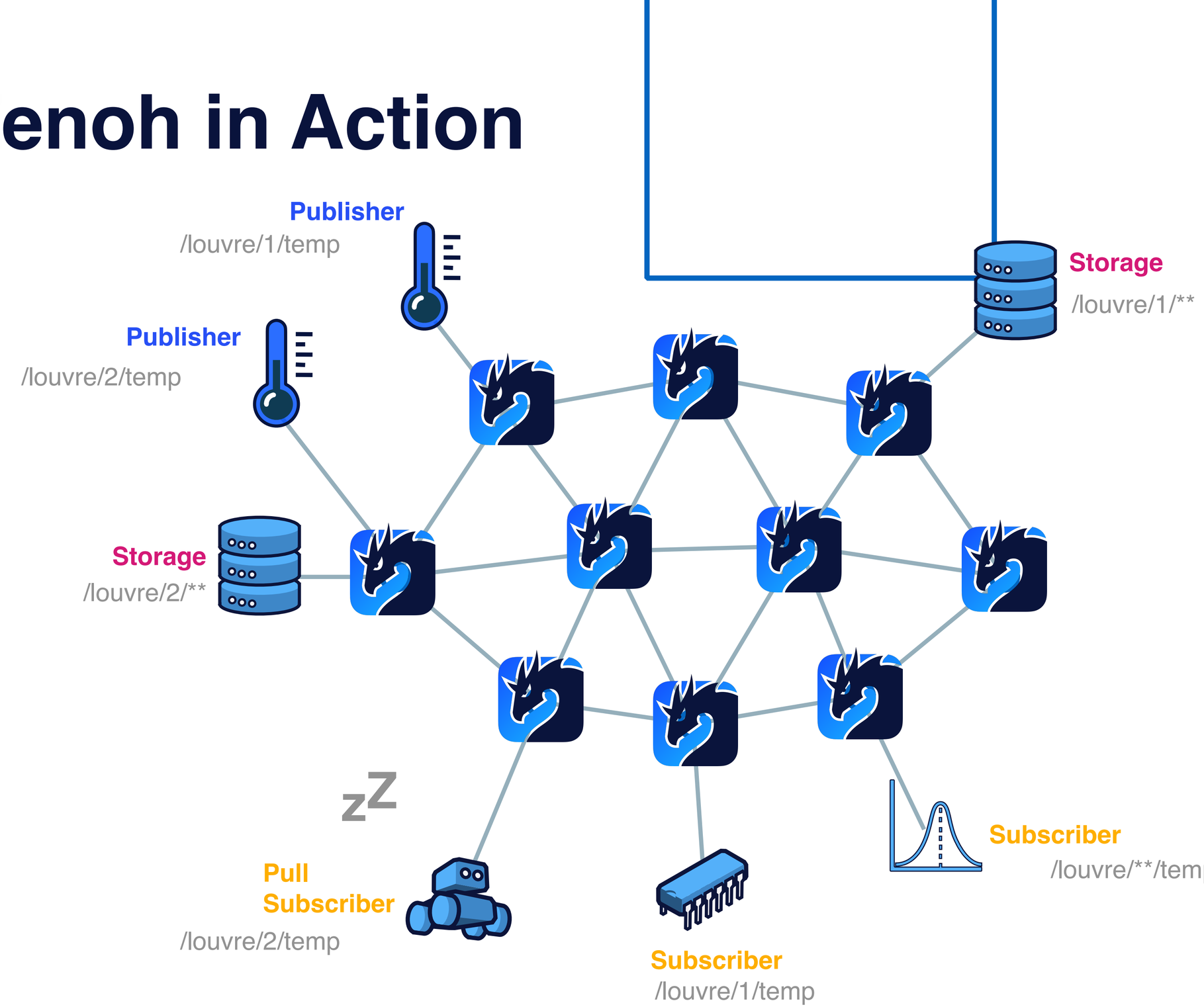
Subscriber : A **sink** of values for a key expression

Queryable : A **well** of values for a key expression

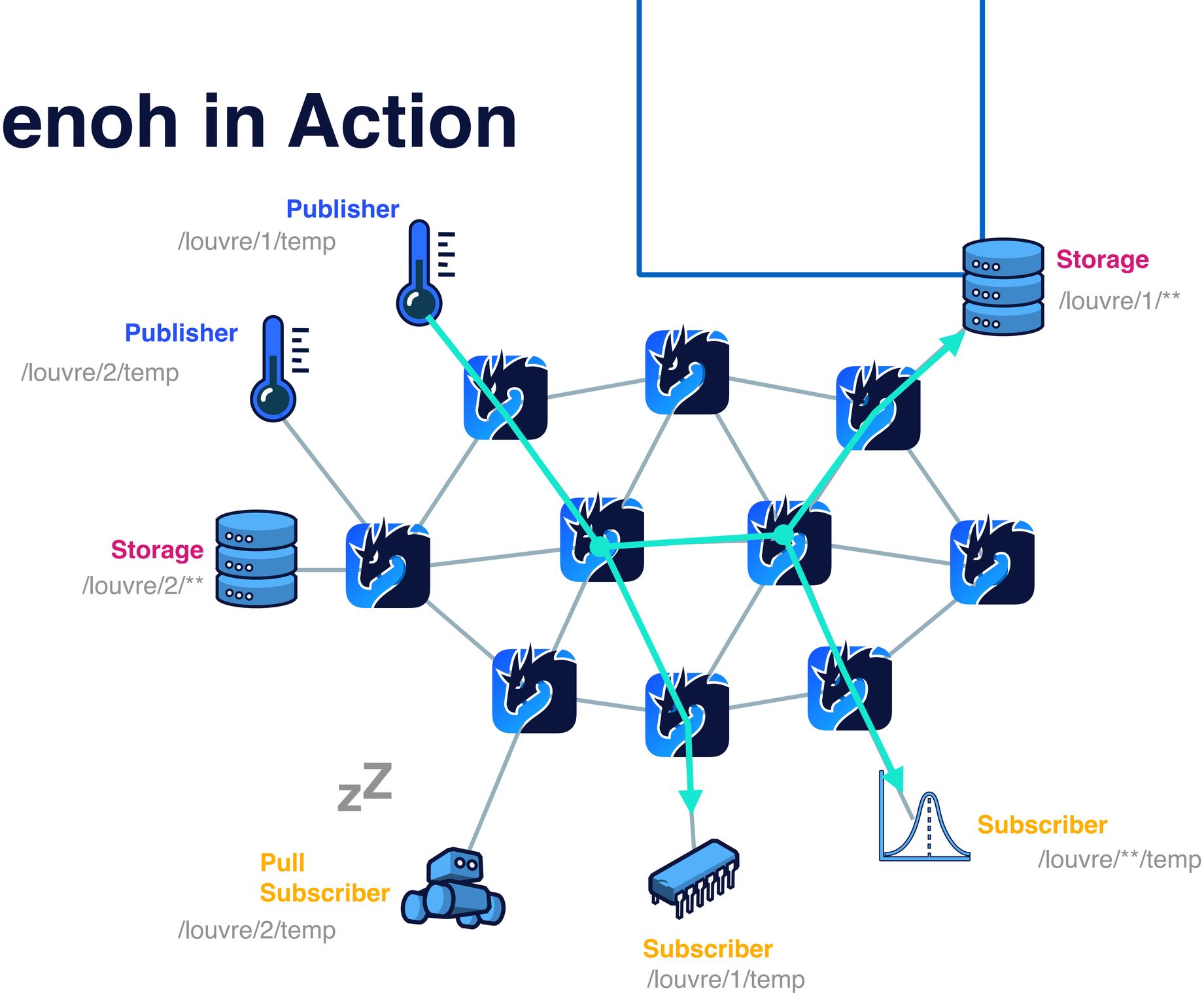
Zenoh in Action



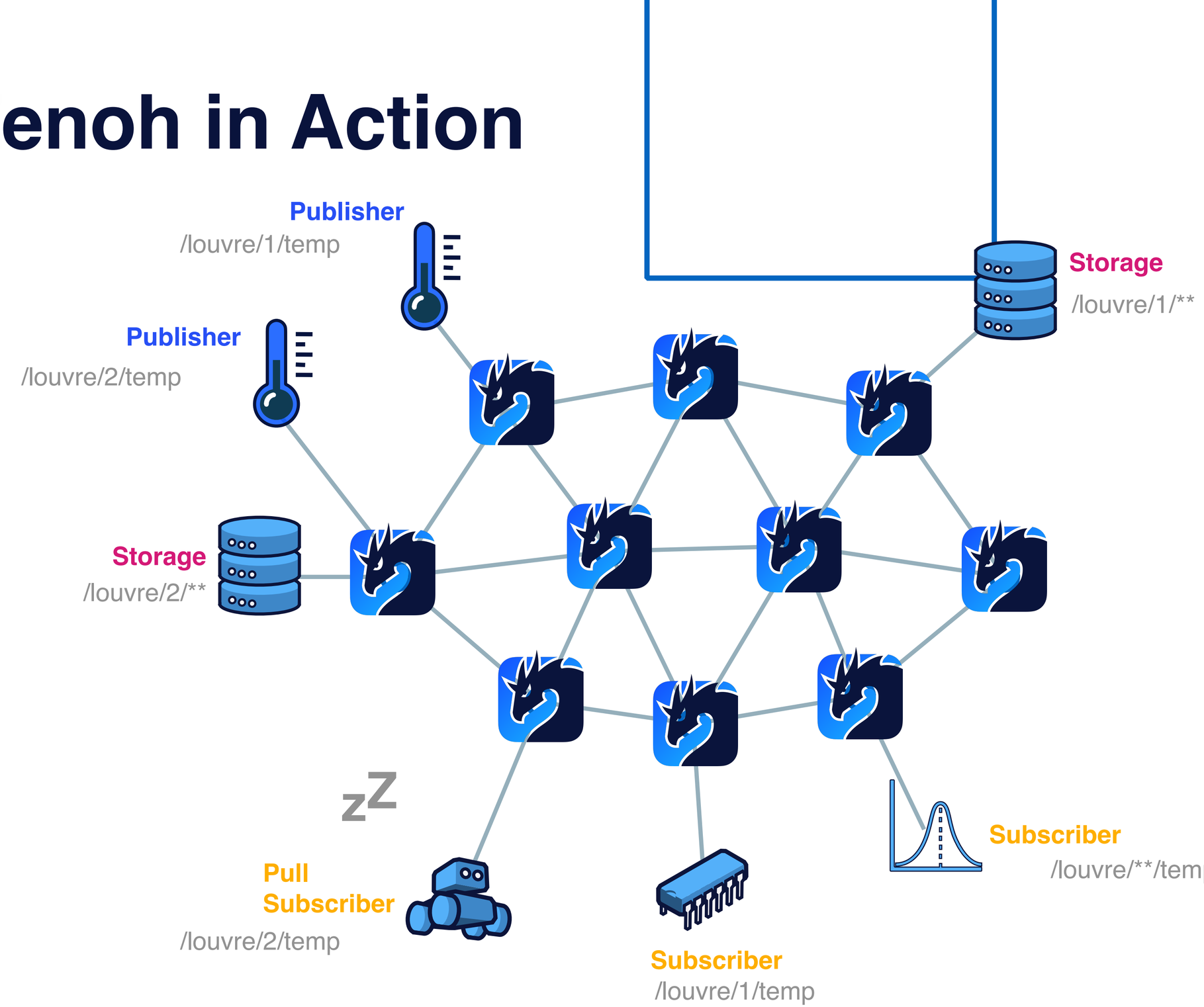
Zenoh in Action



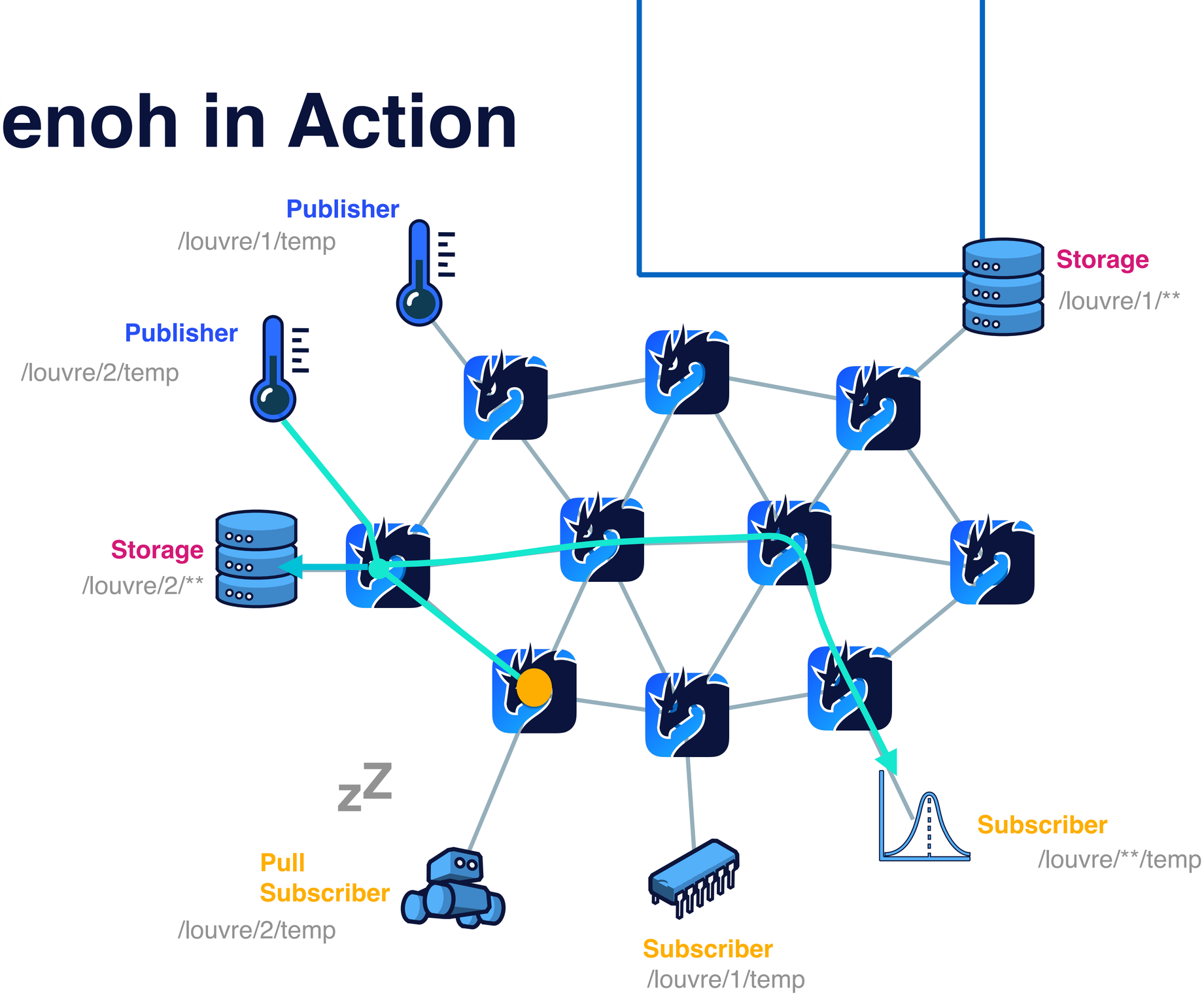
Zenoh in Action



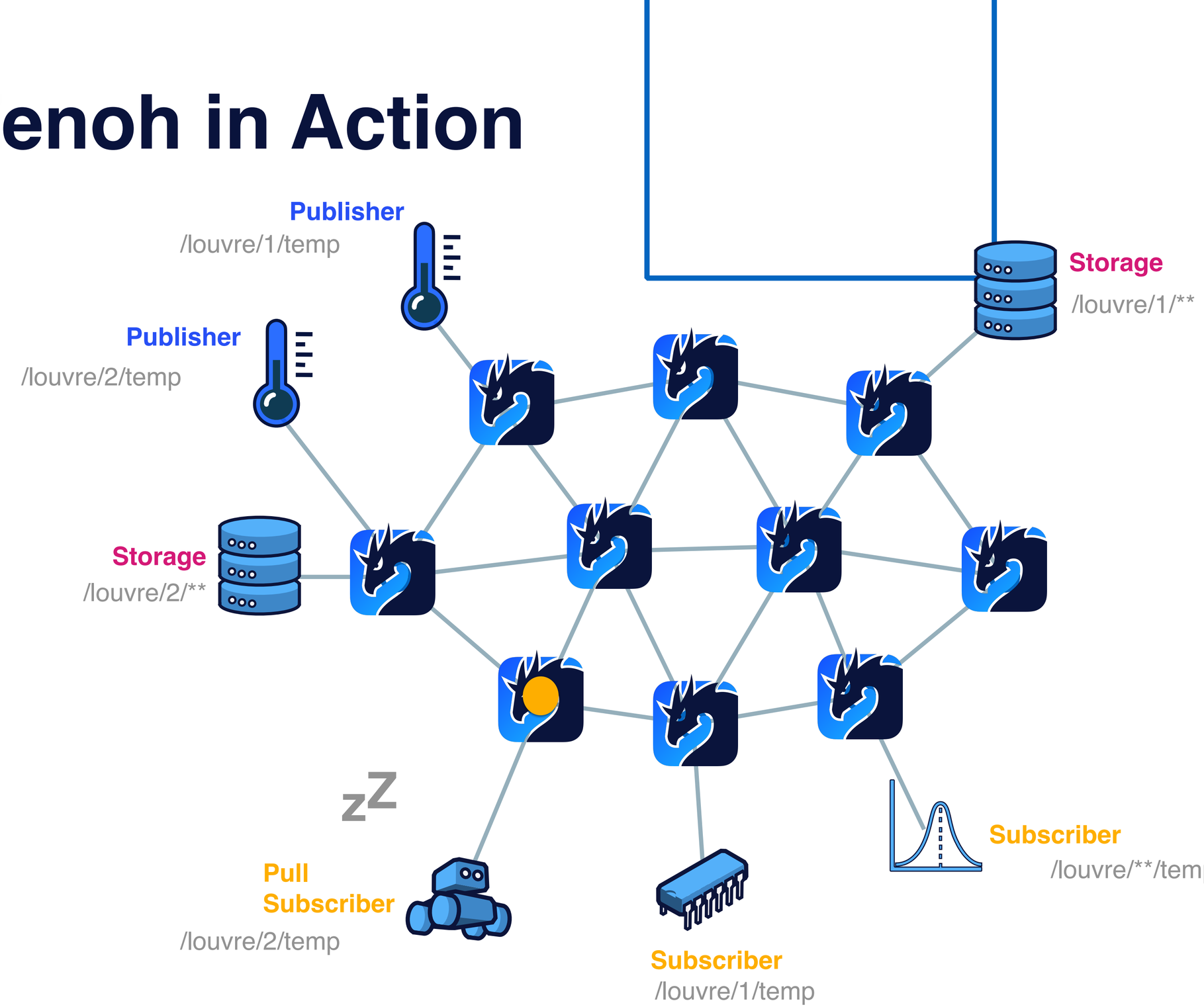
Zenoh in Action



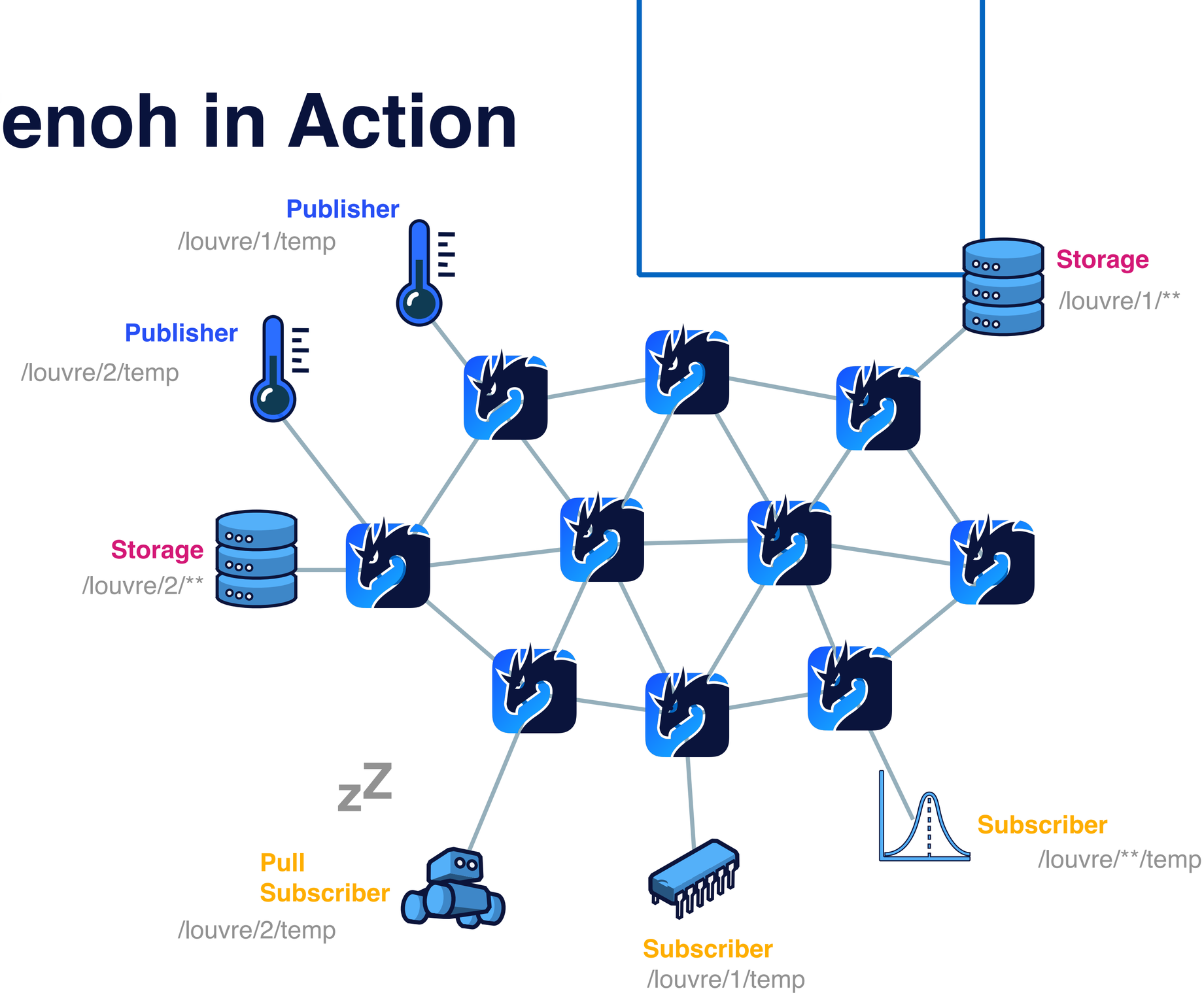
Zenoh in Action



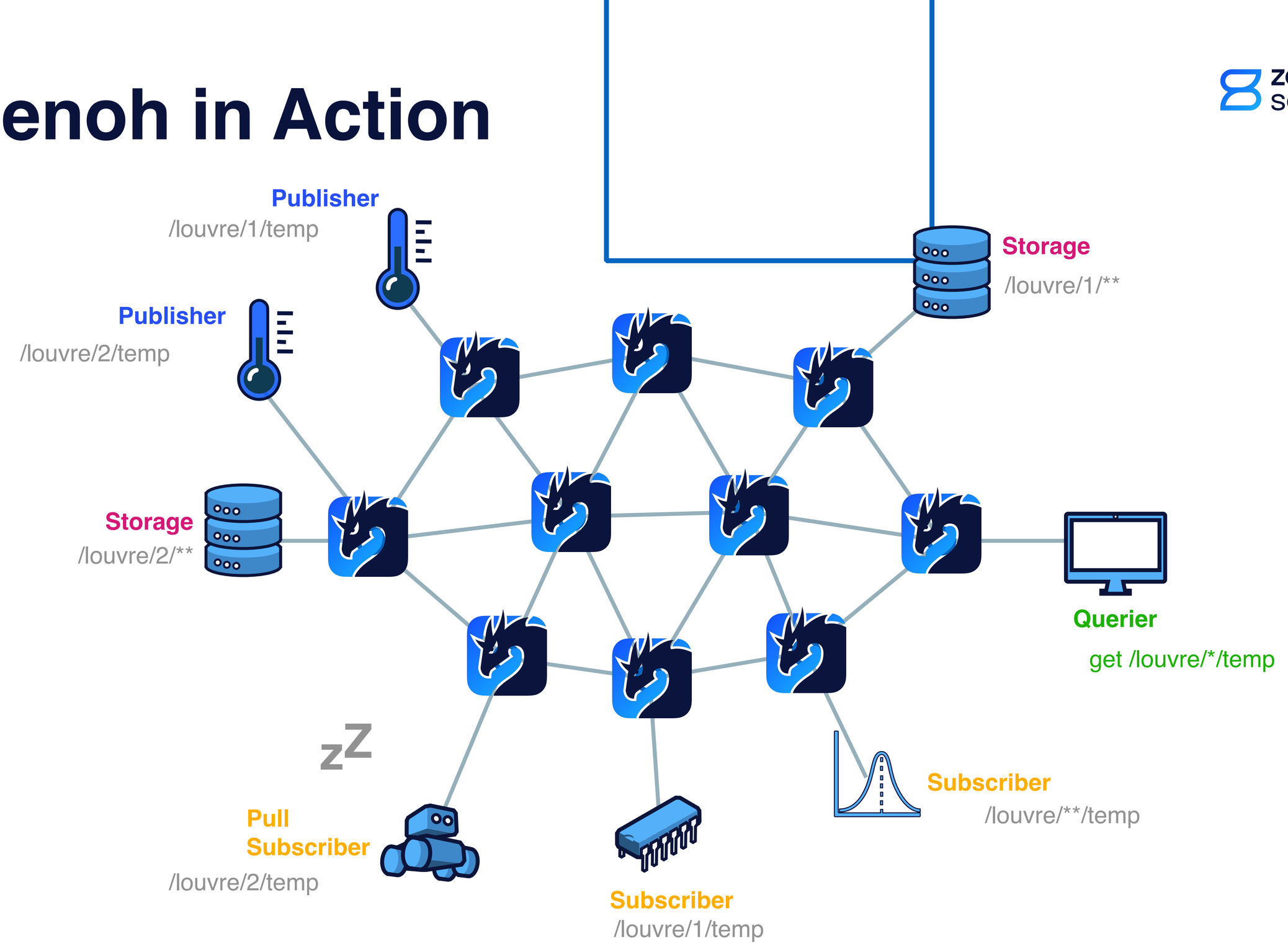
Zenoh in Action



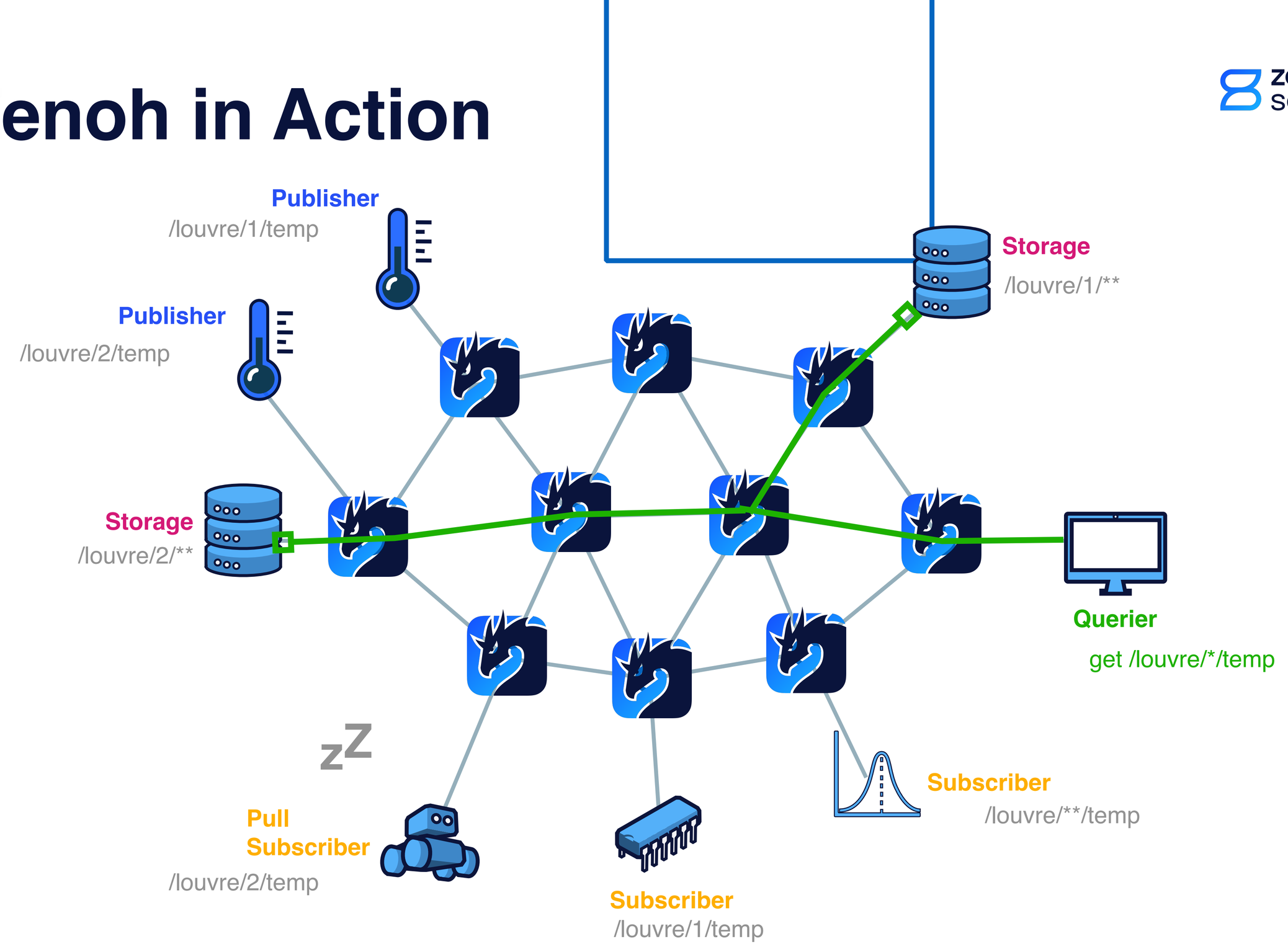
Zenoh in Action



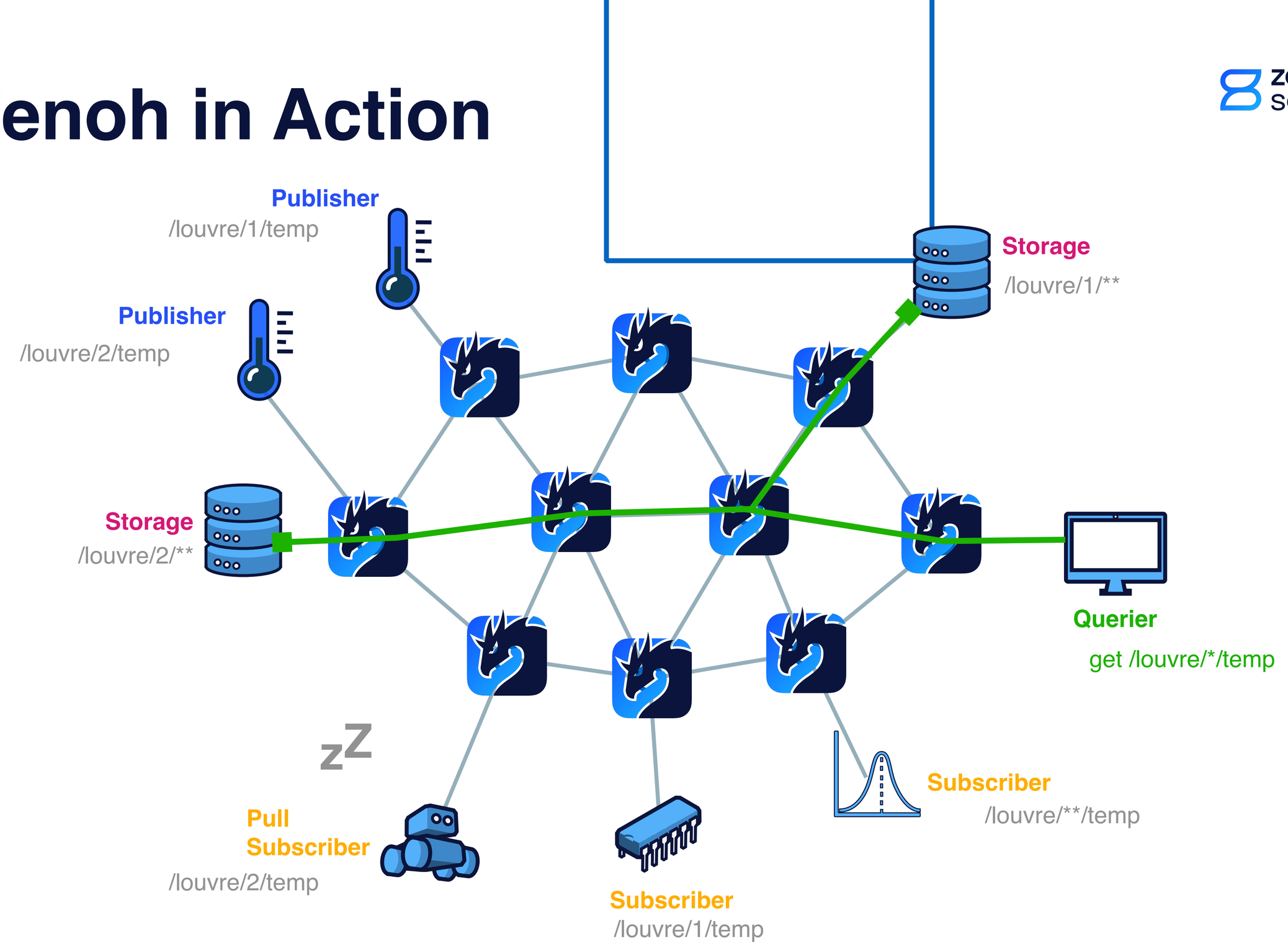
Zenoh in Action



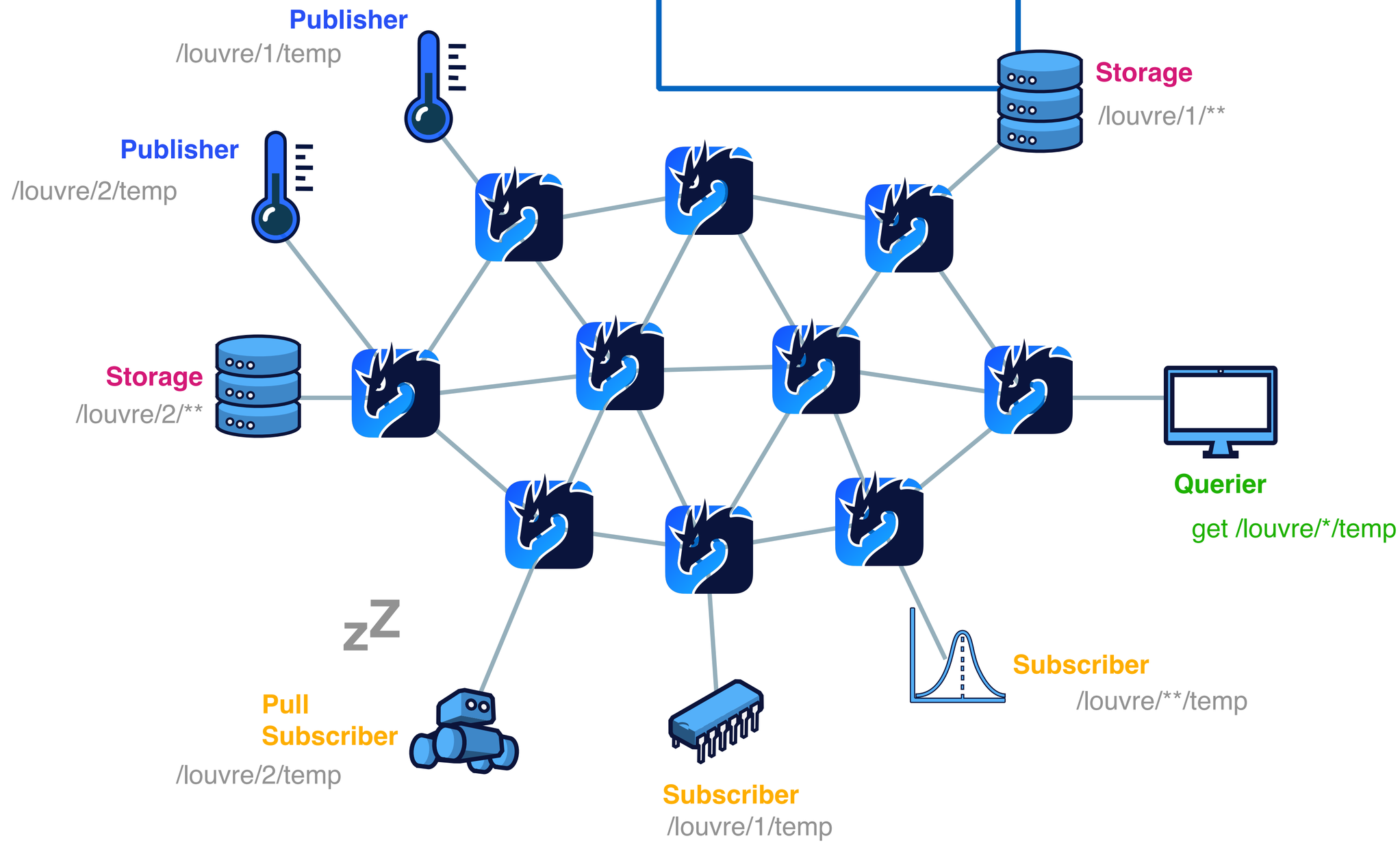
Zenoh in Action



Zenoh in Action

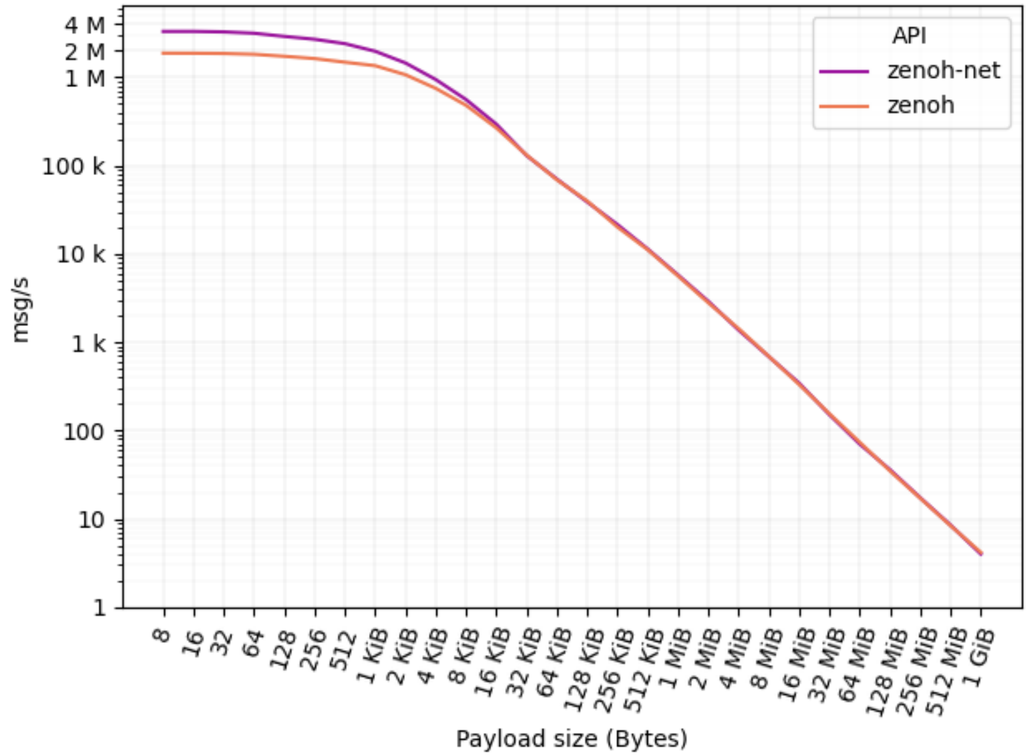


Zenoh in Action



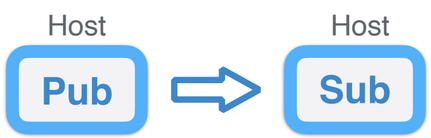
Performance

High throughput (4M msg/s — +40Gb/s)

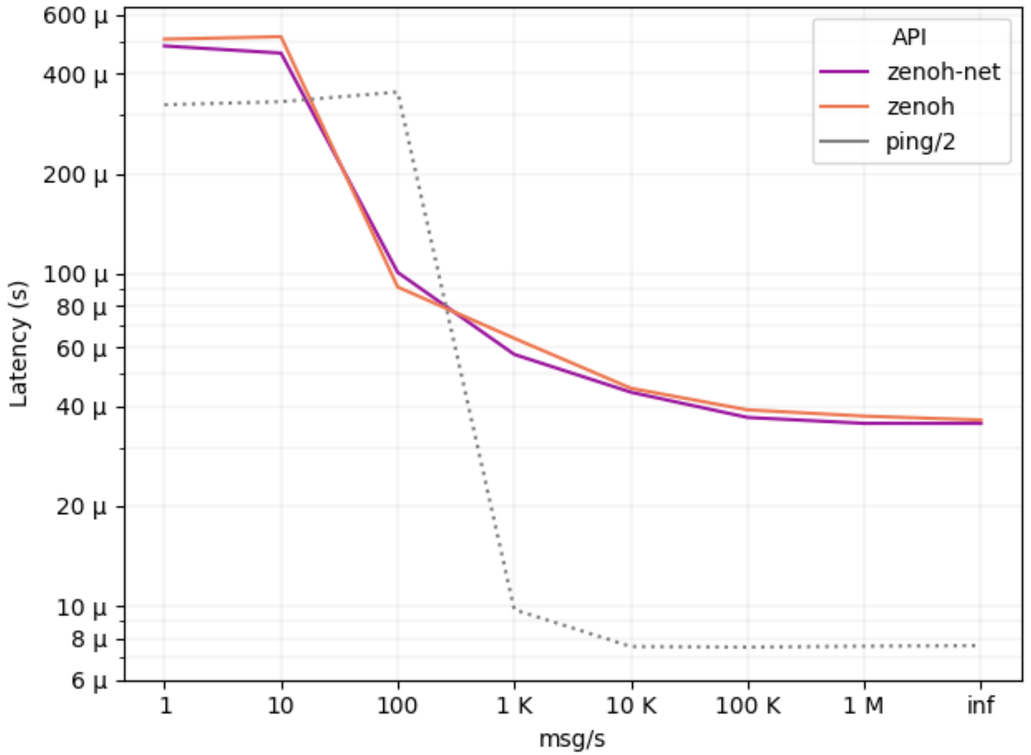


Low latency (35 us)

Minimal **wire overhead** of **4-6 bytes**



Test ran on 10/07/2021 on
Ubuntu 20.04
AMD Ryzen
32GB RAM
100Gbps ETH



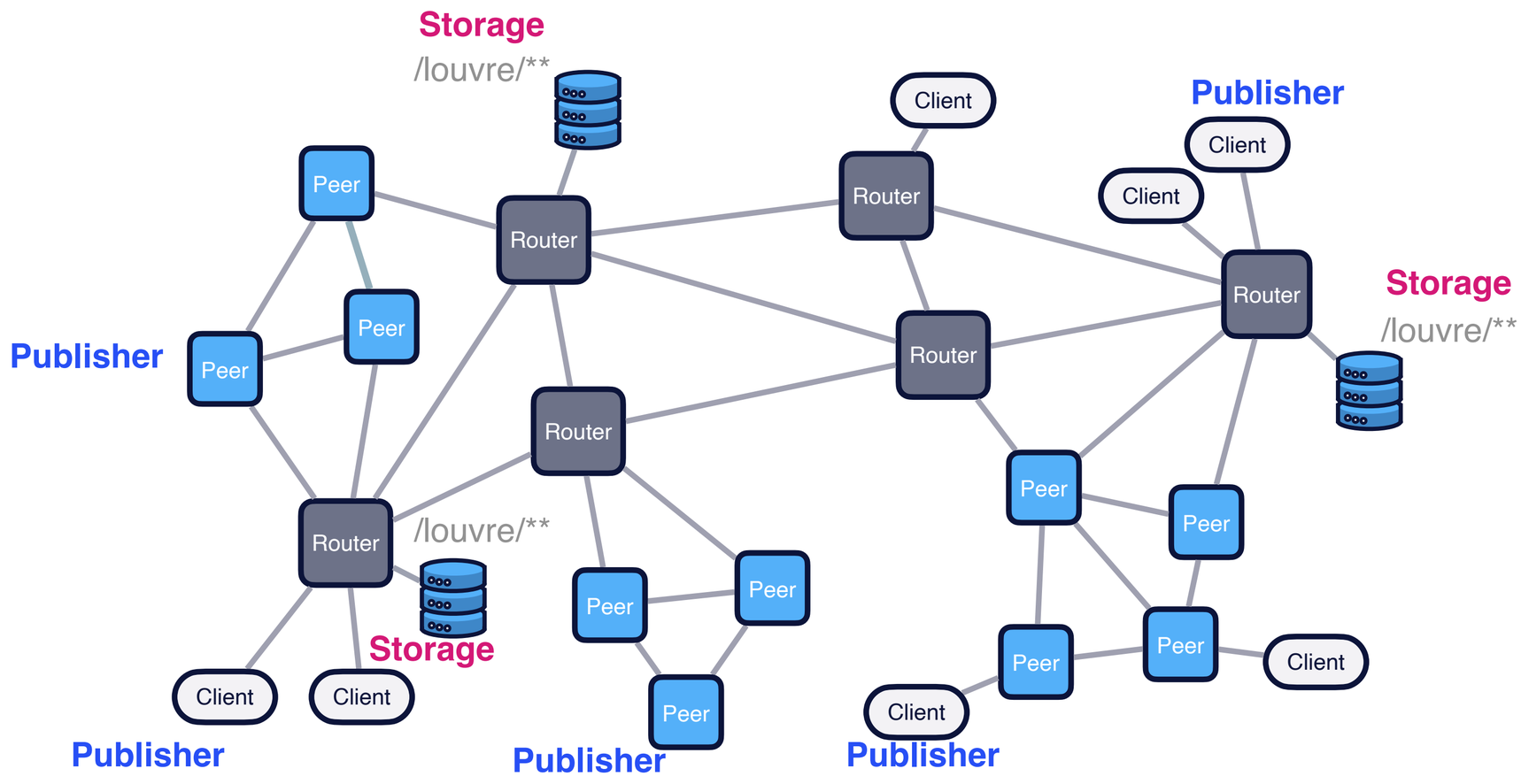
“One of the things I love about music is live performance.” - Yo-Yo Ma

Replication in Zenoh

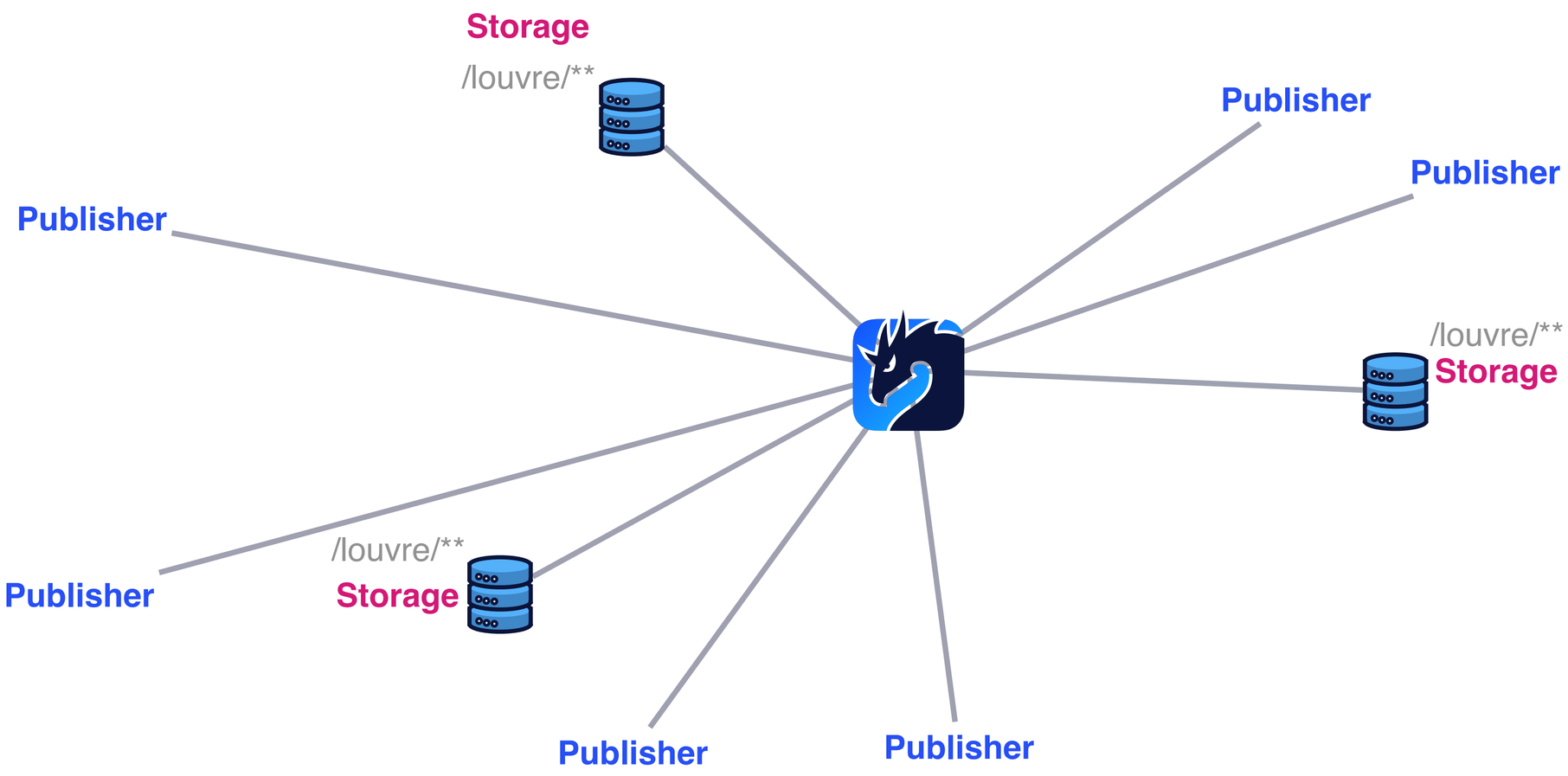
Fault tolerance

Load balancing

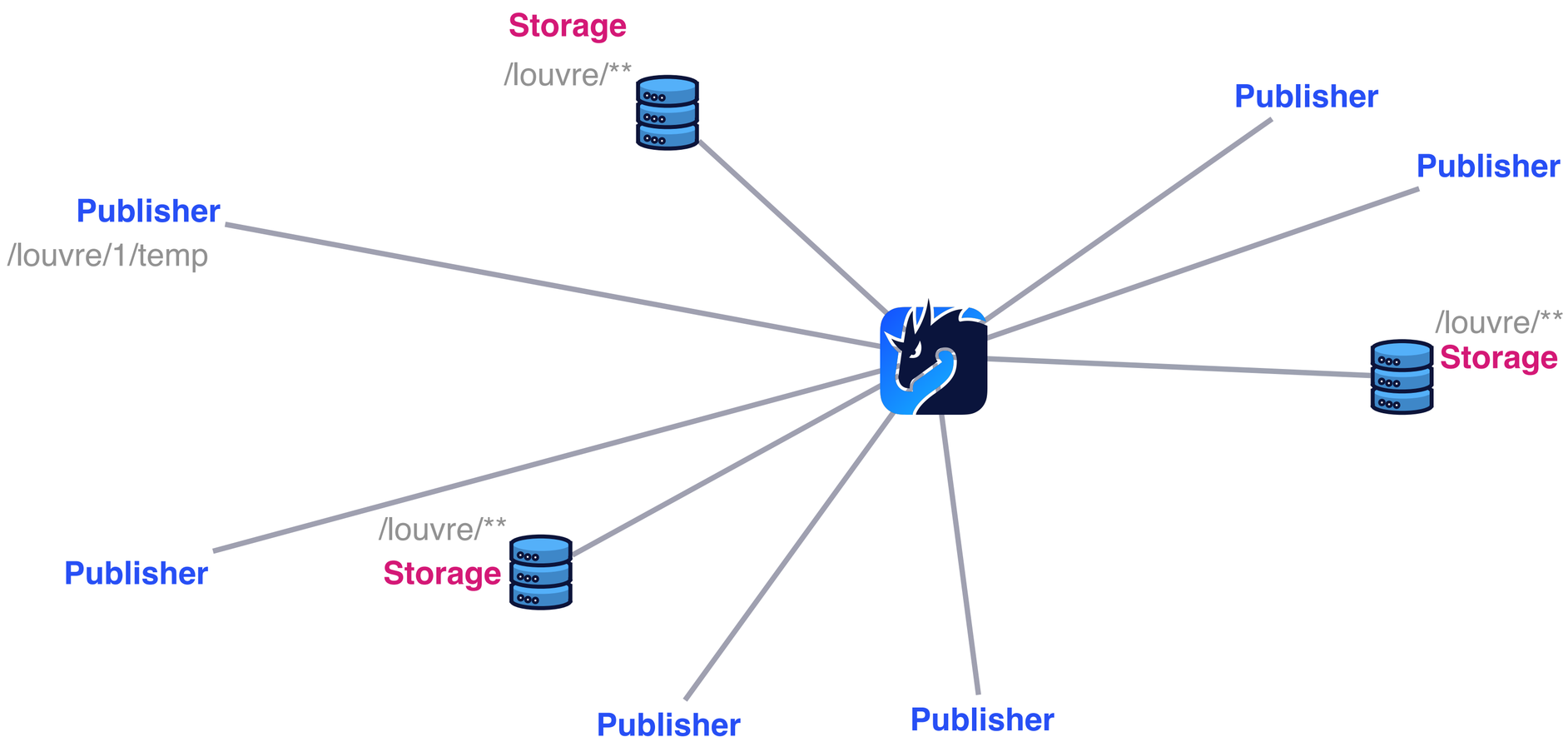
Multi-master replication



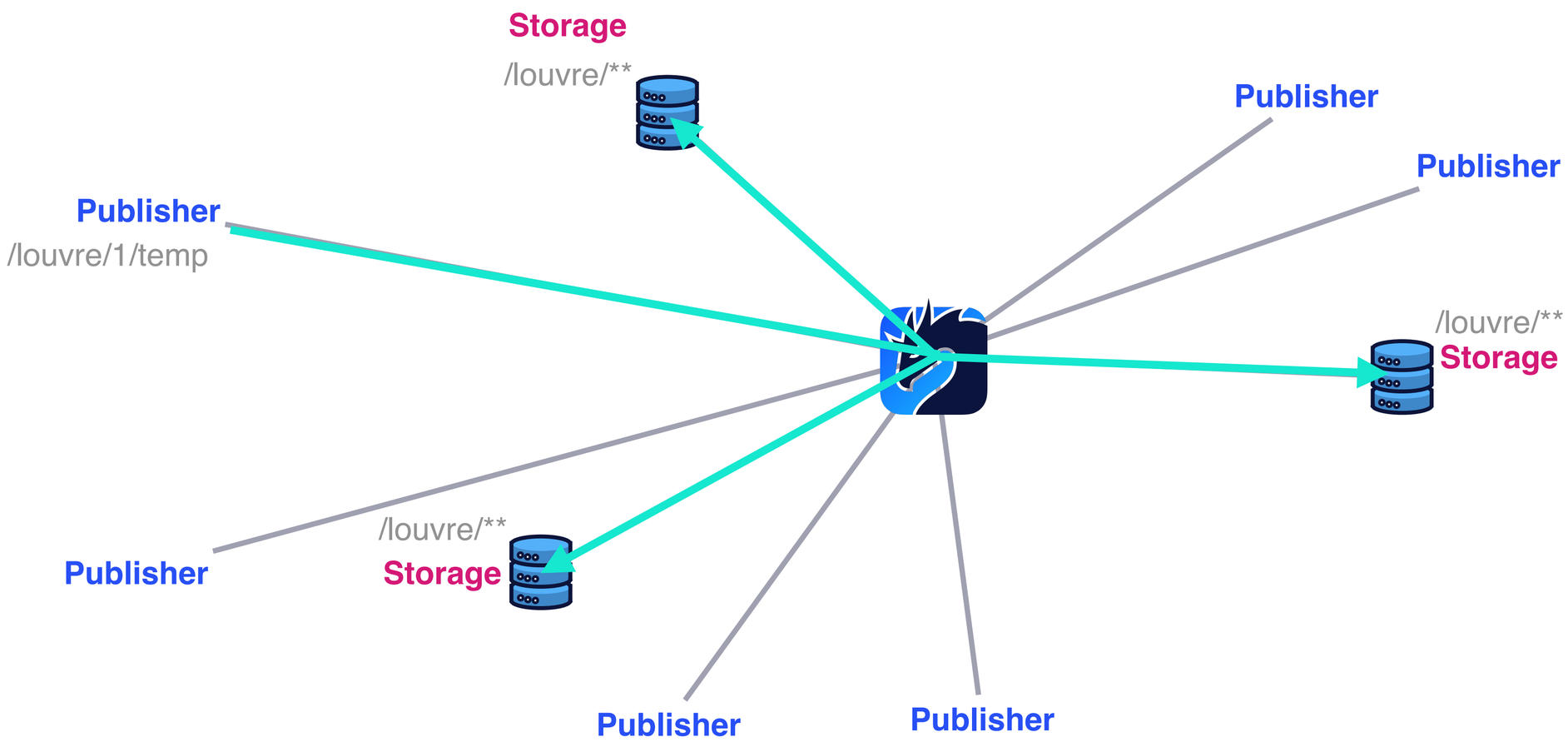
Replicated storages Abstract View



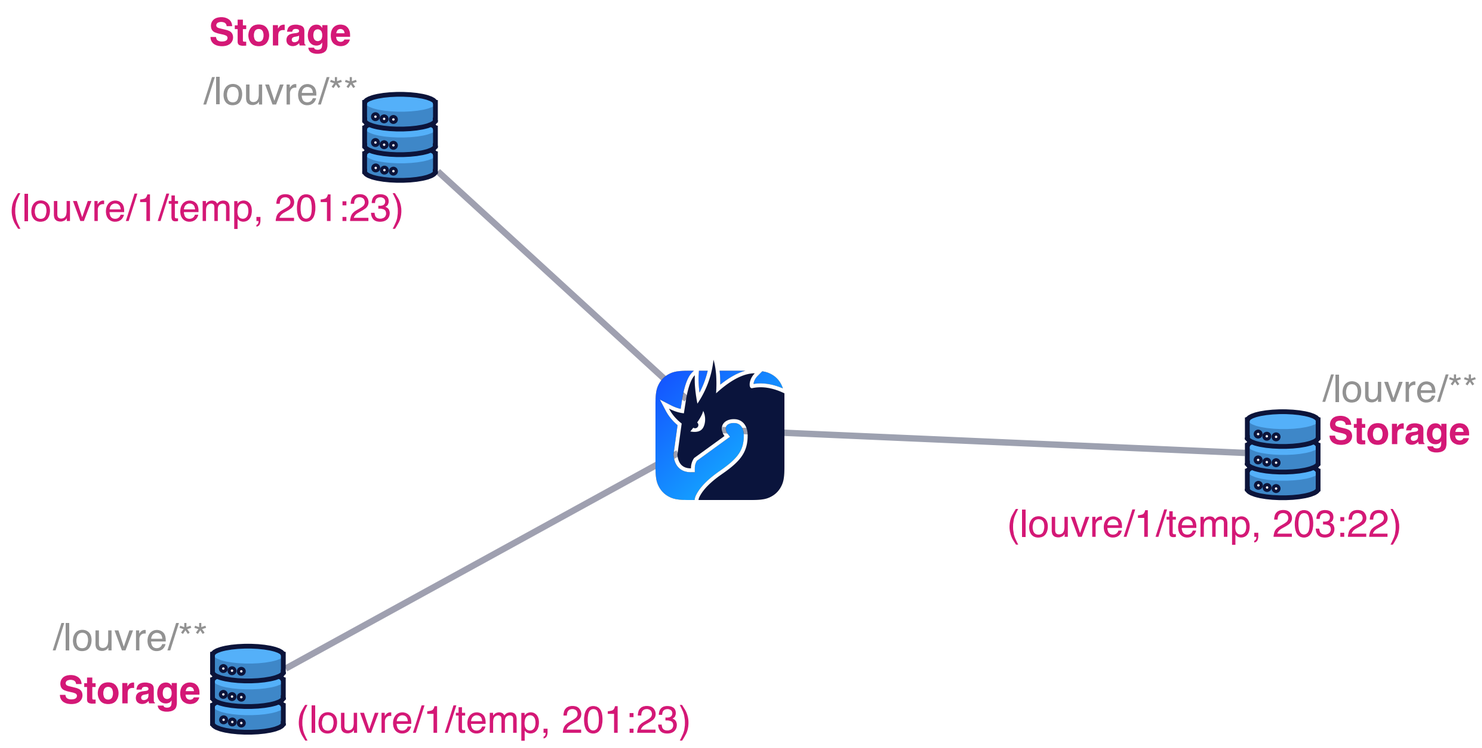
Replicated storages Abstract View



Replicated storages Abstract View

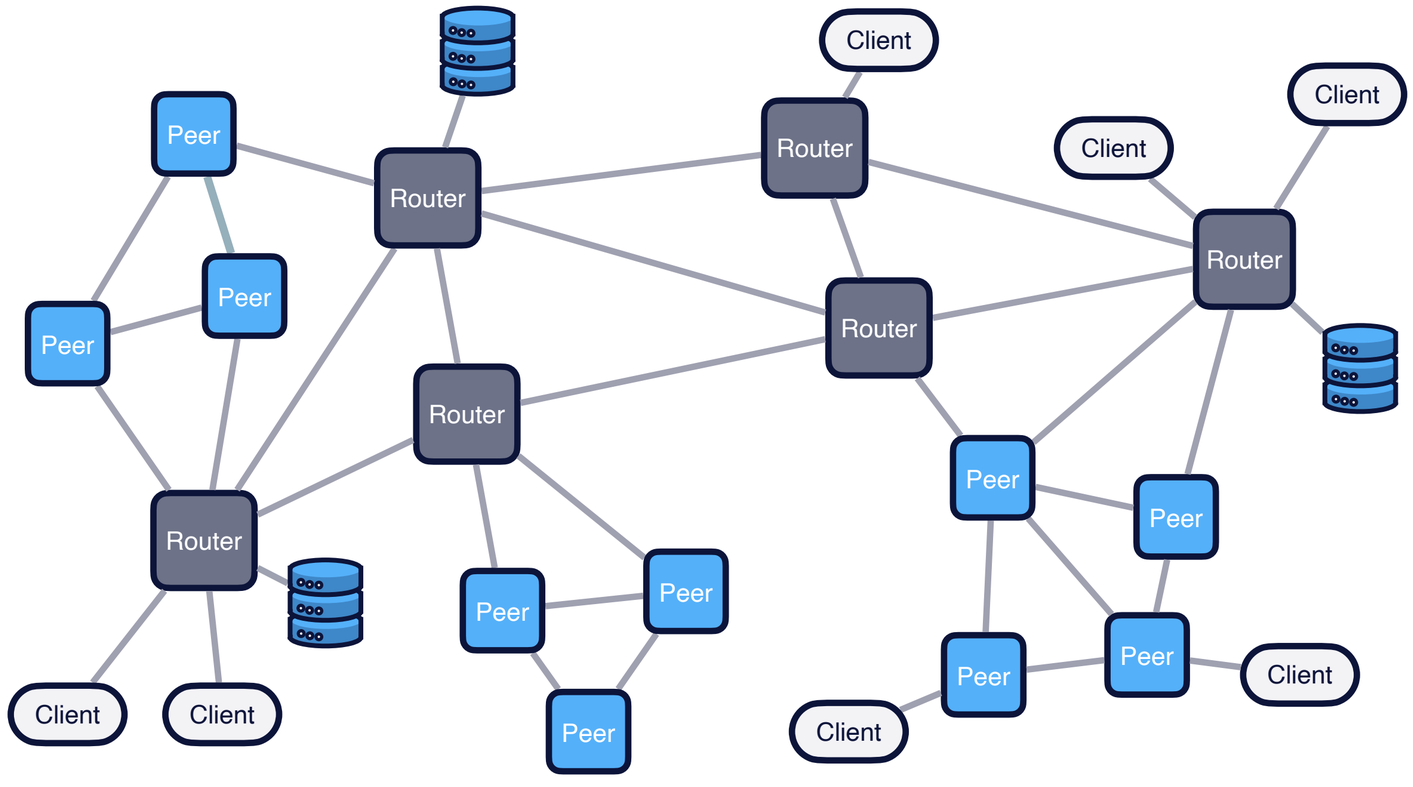


Why consistency?



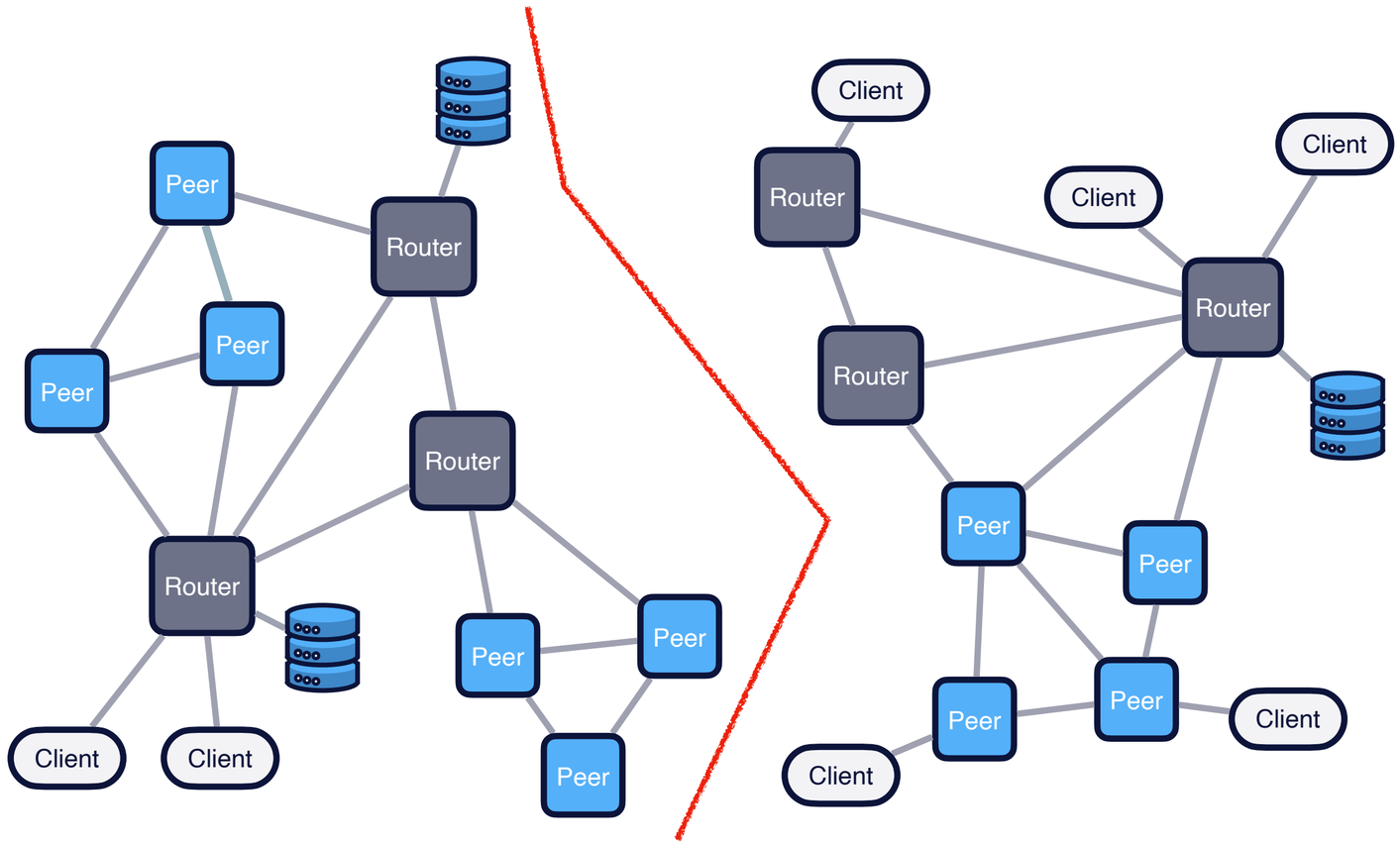
Diverging scenario

- Network might have partitions



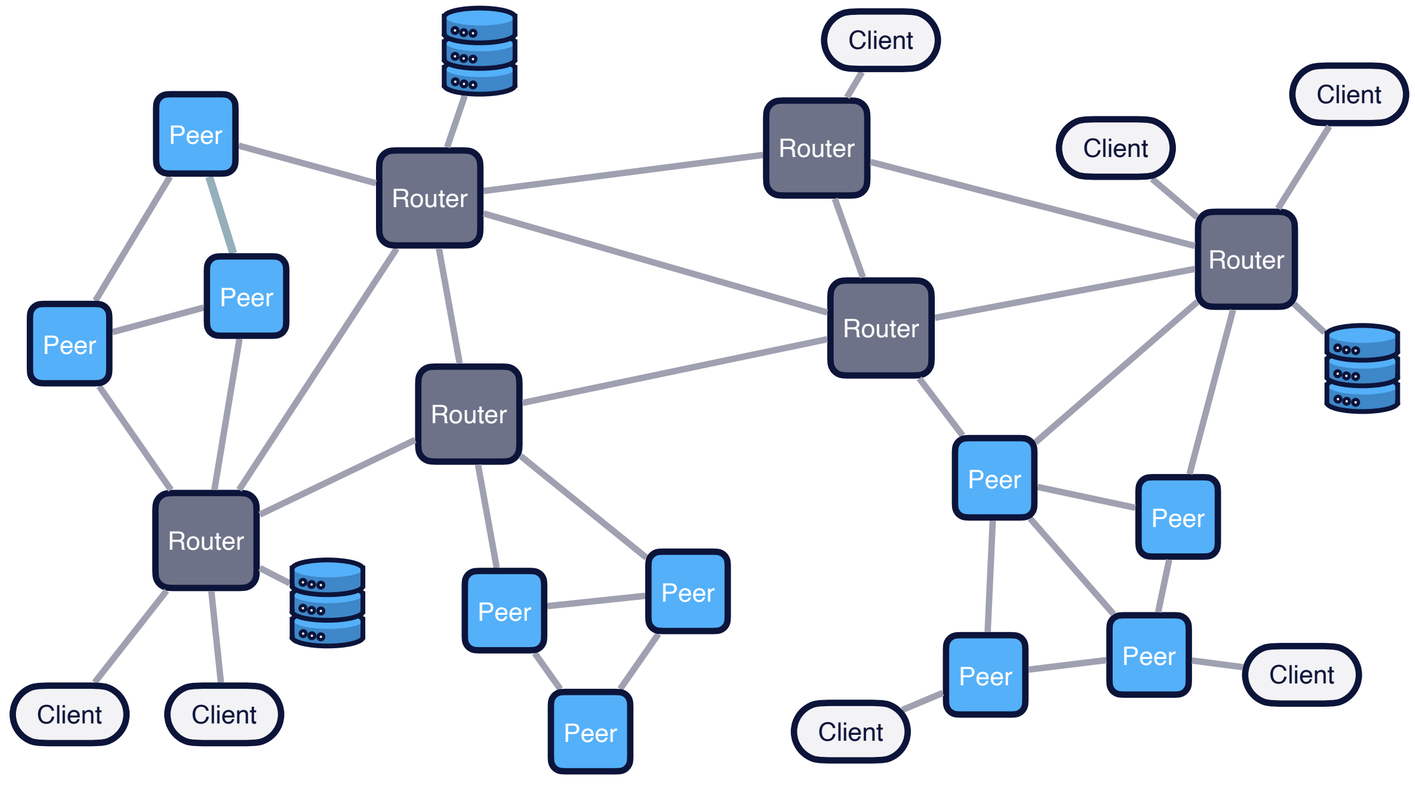
Diverging scenario

- Network might have partitions



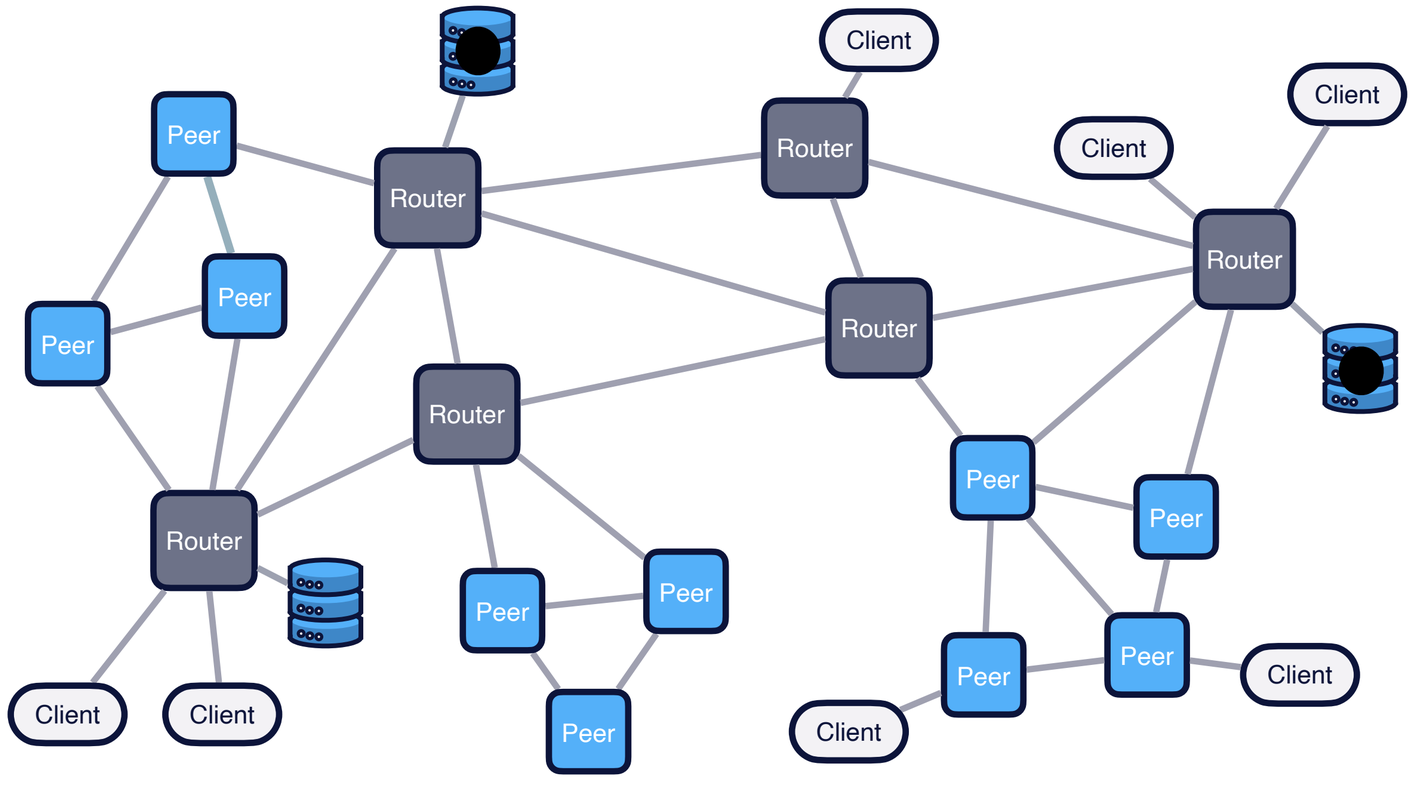
Diverging scenario

- Network might have partitions
- Some messages might be lost before reaching some storages



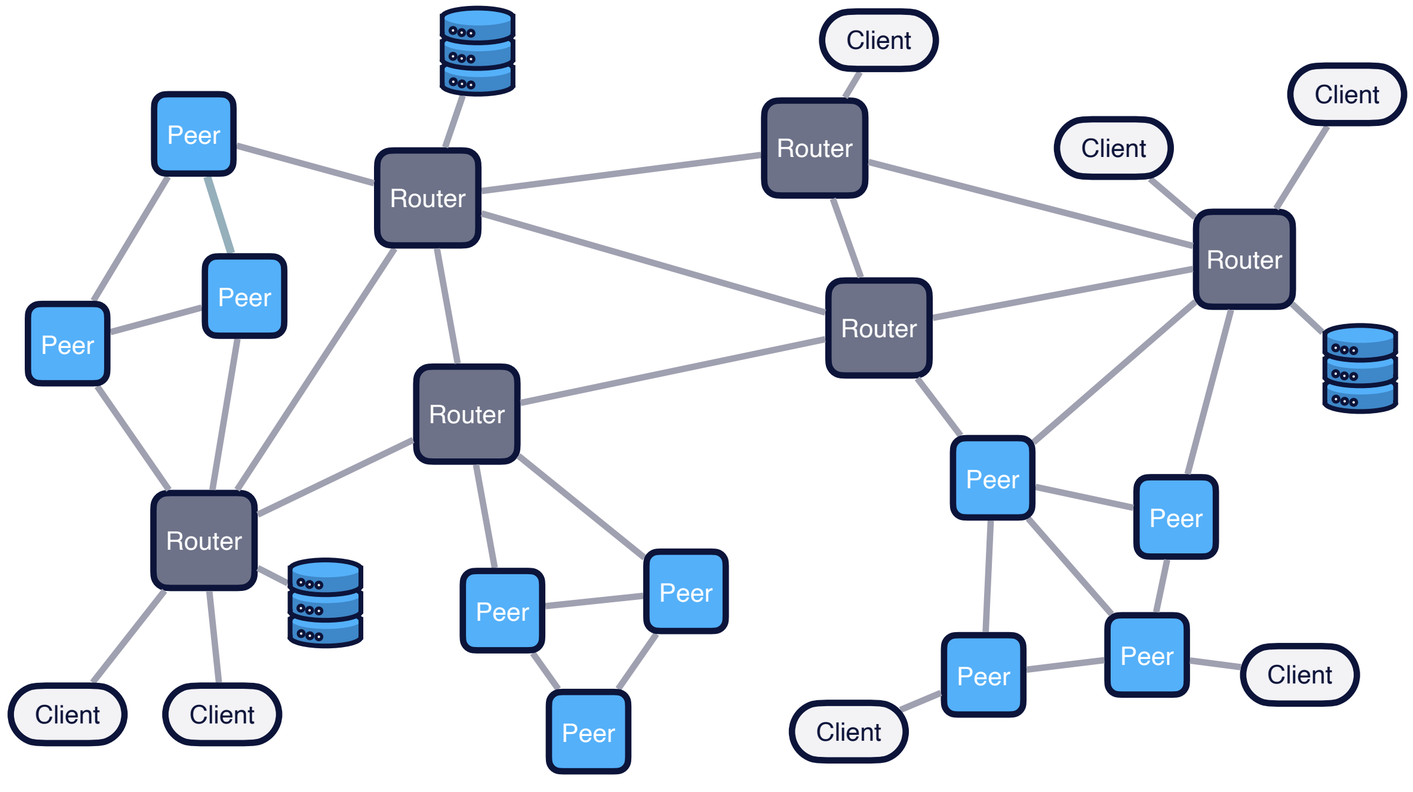
Diverging scenario

- Network might have partitions
- Some messages might be lost before reaching some storages



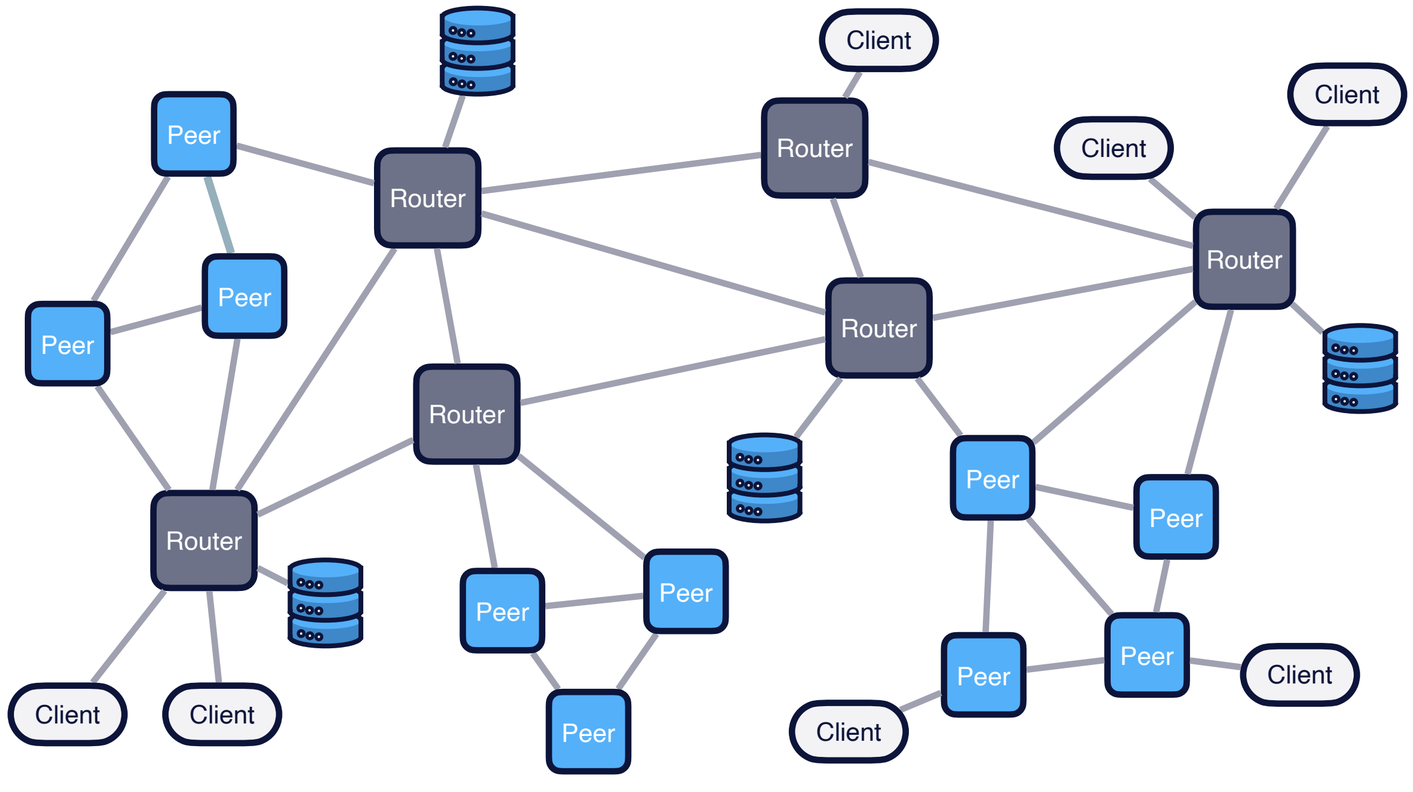
Diverging scenario

- Network might have partitions
- Some messages might be lost before reaching some storages
- A new storage is configured on a running system

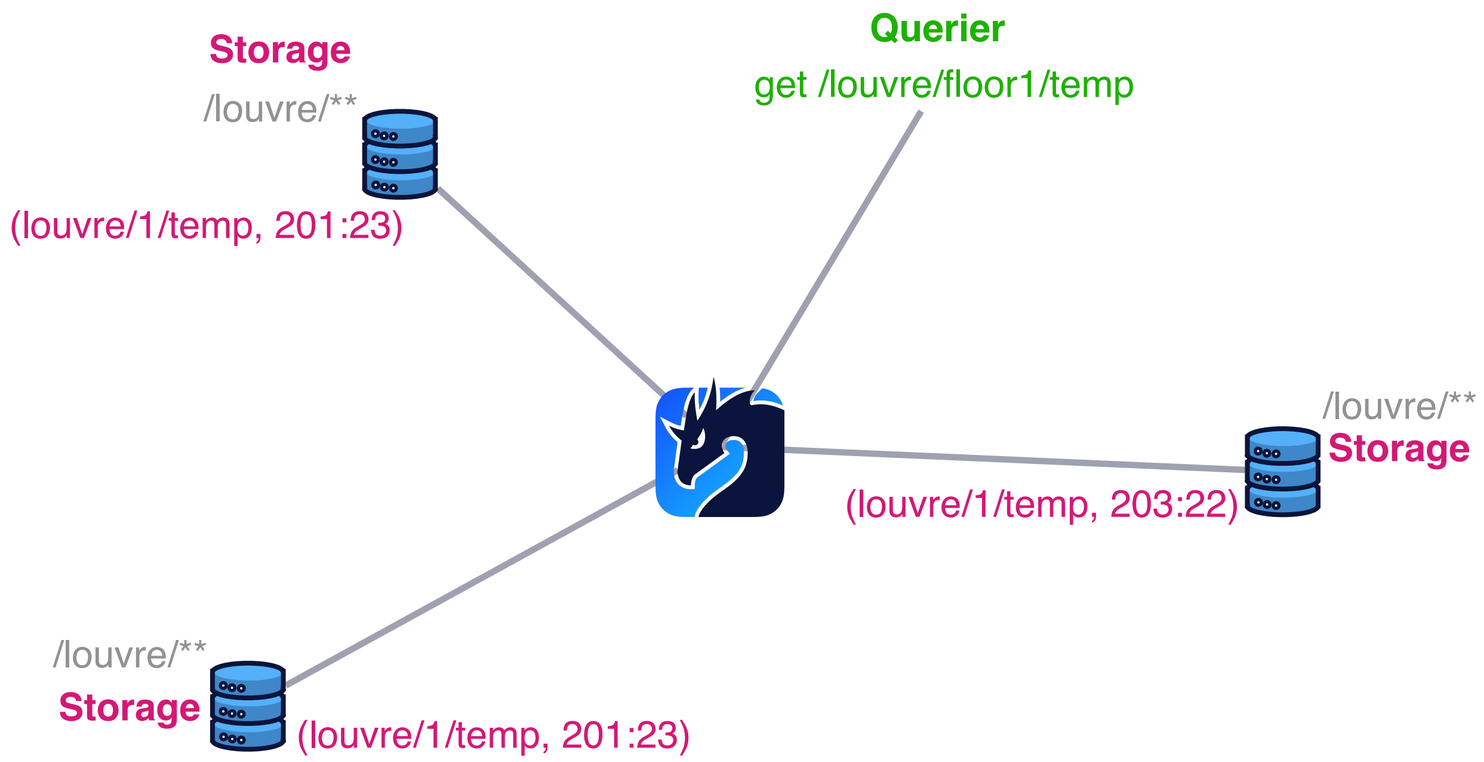


Diverging scenario

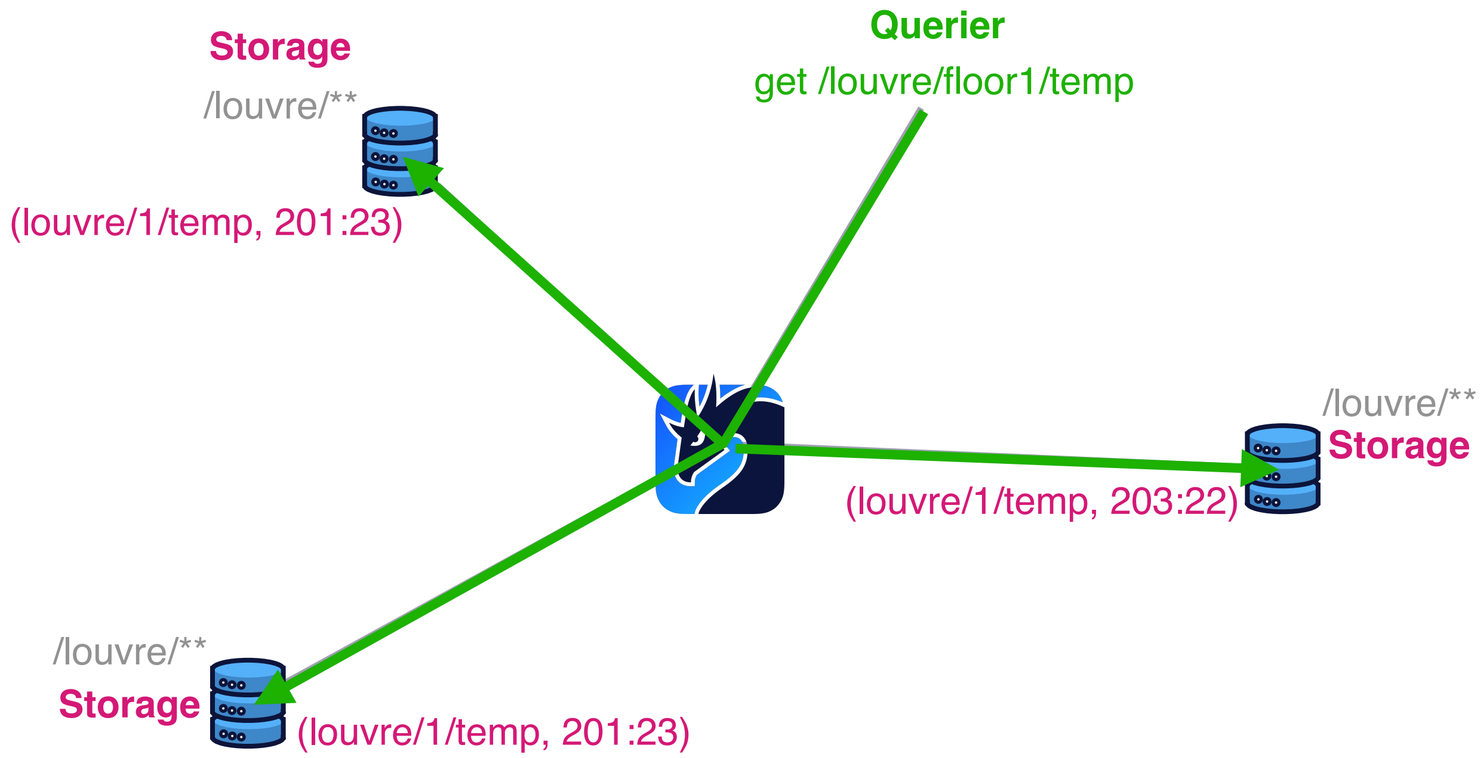
- Network might have partitions
- Some messages might be lost before reaching some storages
- A new storage is configured on a running system



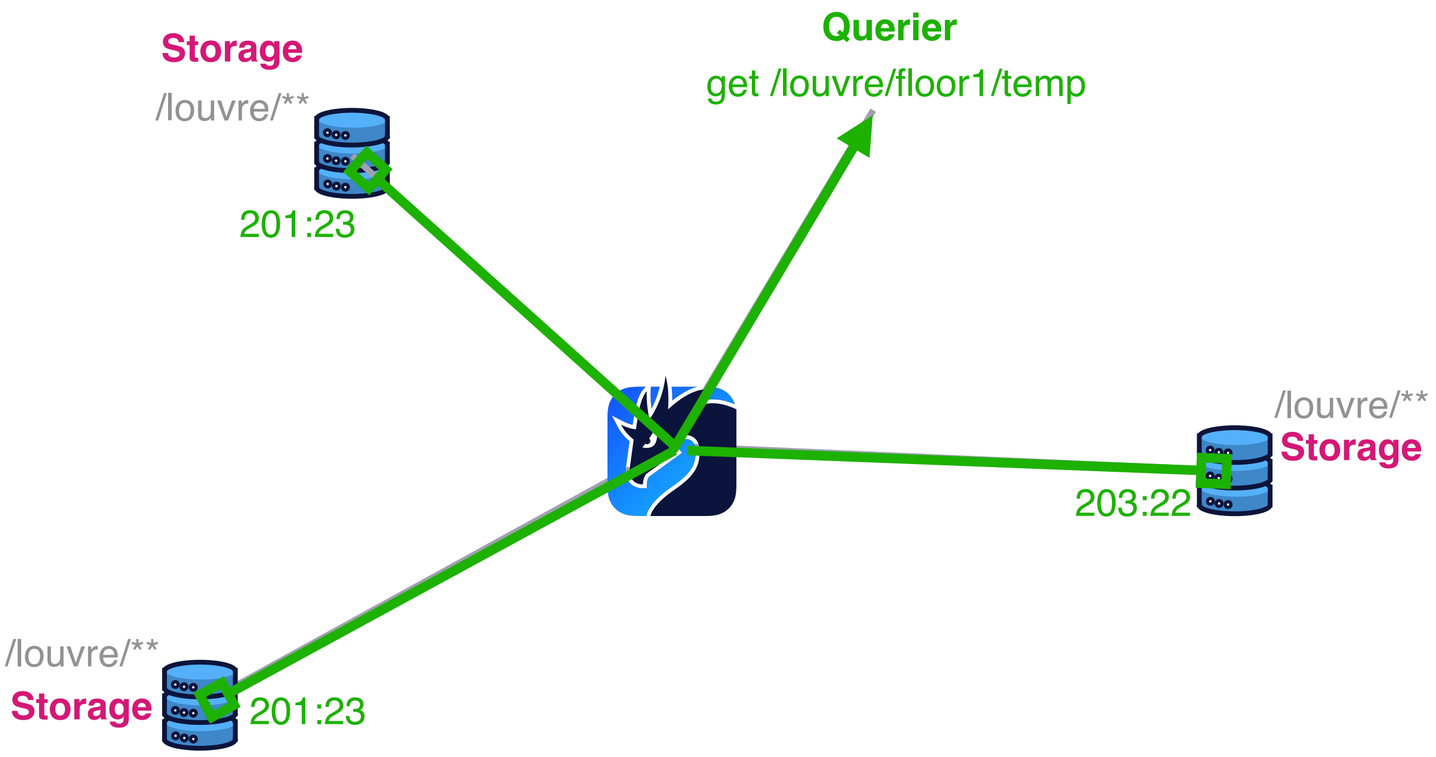
Replication now



Replication now



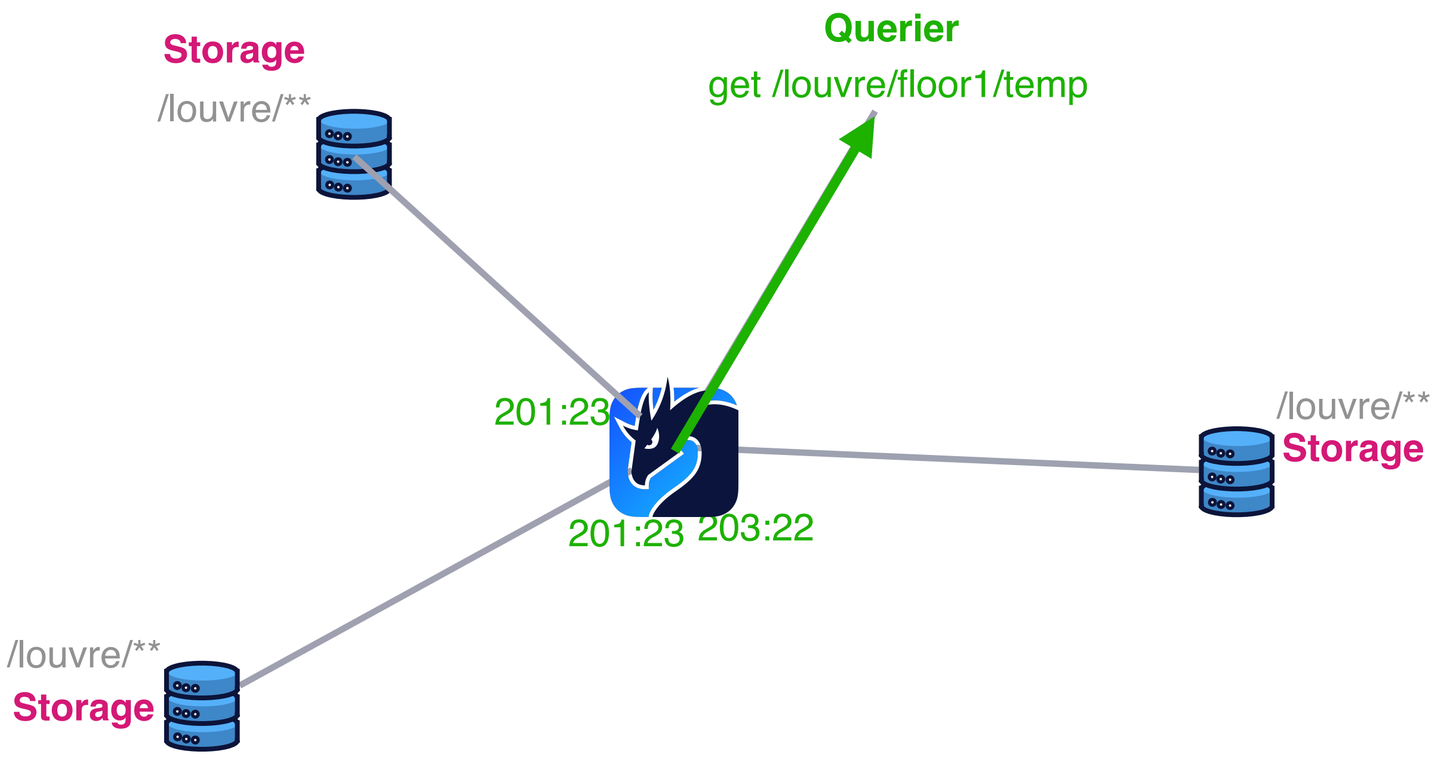
Result Consolidation



Results from the storage are shown as timestamp: value

Consolidation modes : NONE, LAZY, **FULL**

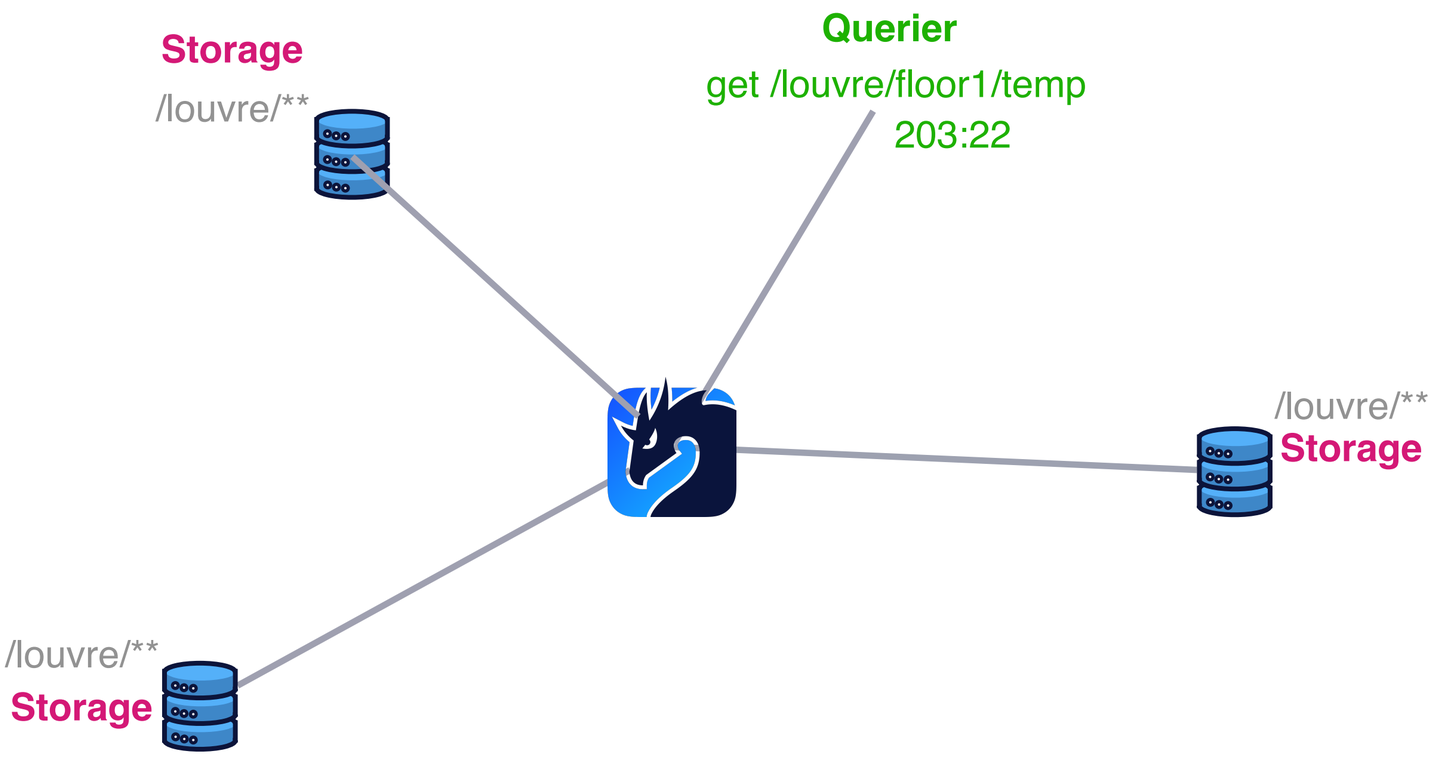
Result Consolidation



Results from the storage are shown as timestamp: value

Consolidation modes : NONE, LAZY, **FULL**

Result Consolidation



Results from the storage are shown as timestamp: value

Consolidation modes : NONE, LAZY, **FULL**

Why a better replication?

The state of the storages remain diverged

Need to query all storages

- Impacts performance

Guarantees for proper replication

Eventual Consistency between different storages subscribing to the same key expression

First step: support for only full replication

System Model

- Multiple publishers and storages for the same data
- Atomic operations (no transactions)
- Each operation has a unique timestamp (using Hybrid Logical Clock)
- Keyspace of storage is defined at storage startup
- Storages don't **explicitly** talk to each other
  - Instead subscribe or query
- Messages may be lost, duplicated or reordered

All models are wrong, but some are useful - George E. P. Box

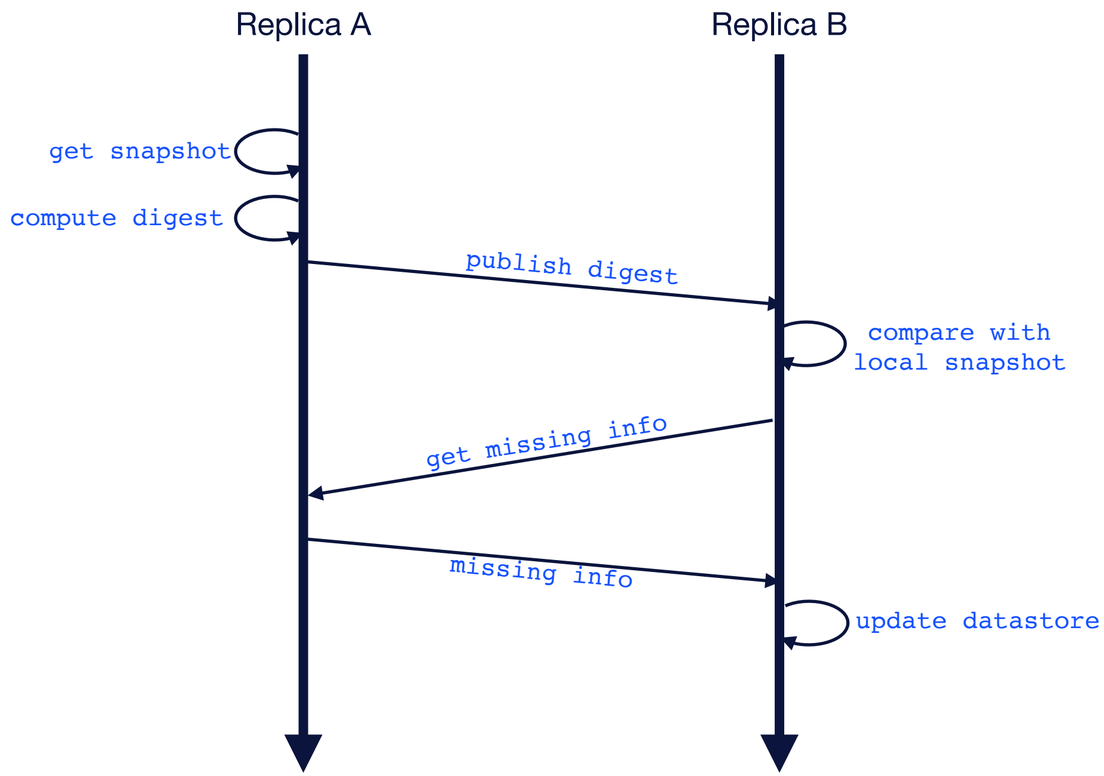
Anti-Entropy Protocol

- Guarantee
  - (Strong ?) Eventual Consistency
- Two phases:
  - Detect misalignment
  - Perform alignment
- Performed by publish/subscribe on /@-digest/**

Anti-Entropy workflow

At pre-defined intervals, publish a digest of the data store snapshot

On receiving a digest, check the inconsistencies with own data store and request missing data



Properties for snapshots

$$snapshot\_time < current\_time - propagation\_delay - possible\_skew$$

$$propagation\_delay \approx network\_diameter$$

$$possible\_skew \approx UHLC\_MAX\_DELTA\_MS$$

$$interval_k = (ORIGIN + \Delta \times (k-1), ORIGIN + \Delta \times k]$$

$$snapshot\_time = k * \Delta$$

Properties for snapshots

$$snapshot\_time < current\_time - propagation\_delay - possible\_skew$$

$$propagation\_delay \approx network\_diameter$$

$$possible\_skew \approx UHLC\_MAX\_DELTA\_MS$$

$$interval_k = (ORIGIN + \Delta \times (k-1), ORIGIN + \Delta \times k]$$

$$snapshot\_time = k * \Delta$$

Sample settings:

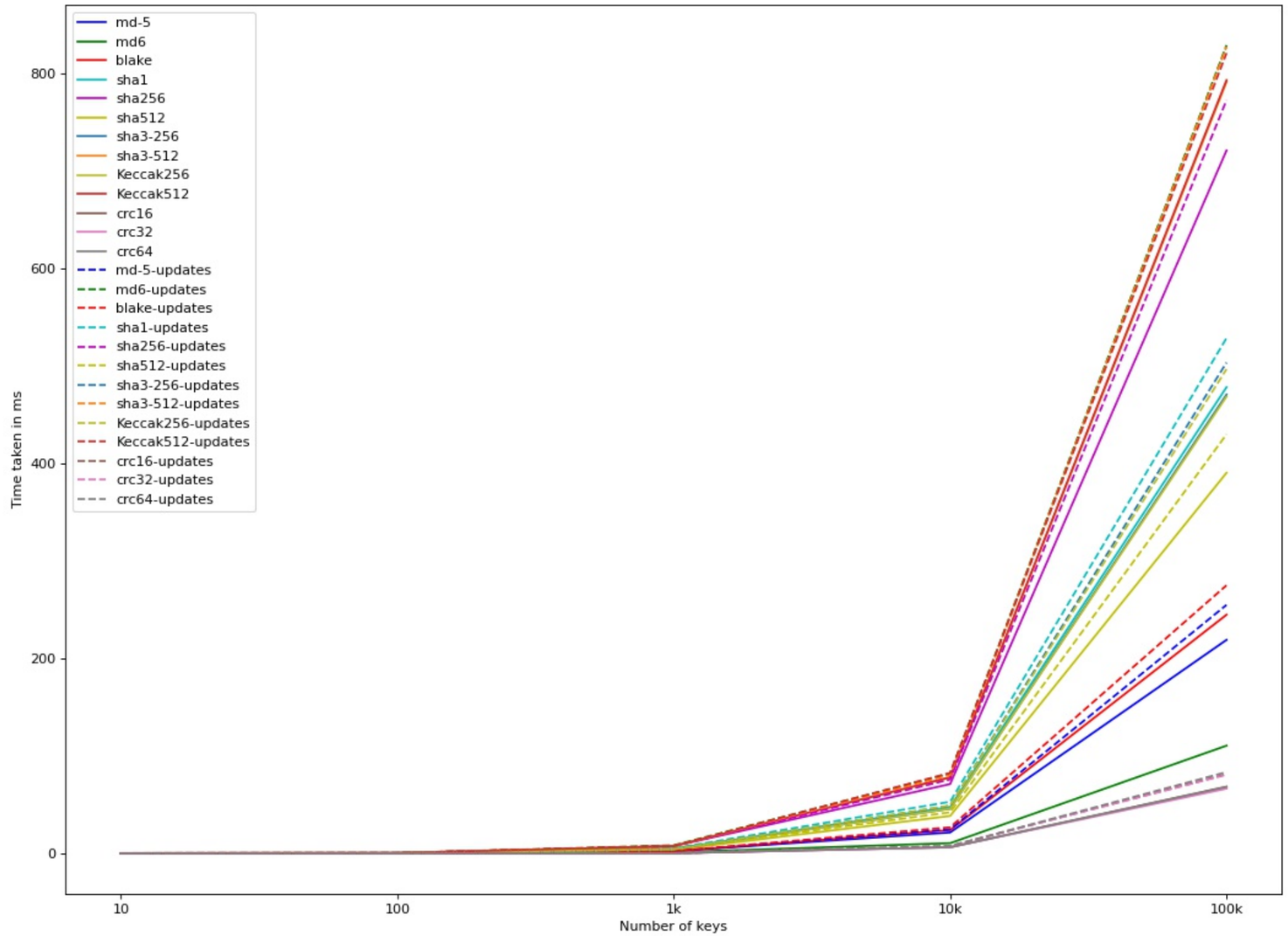
```
const EPOCH_START: SystemTime = SystemTime::UNIX_EPOCH;

const PROPAGATION_DELAY: Duration = Duration::from_millis(200);

const DELTA: Duration = Duration::from_millis(1000);
```

Efficient Hashing algorithm

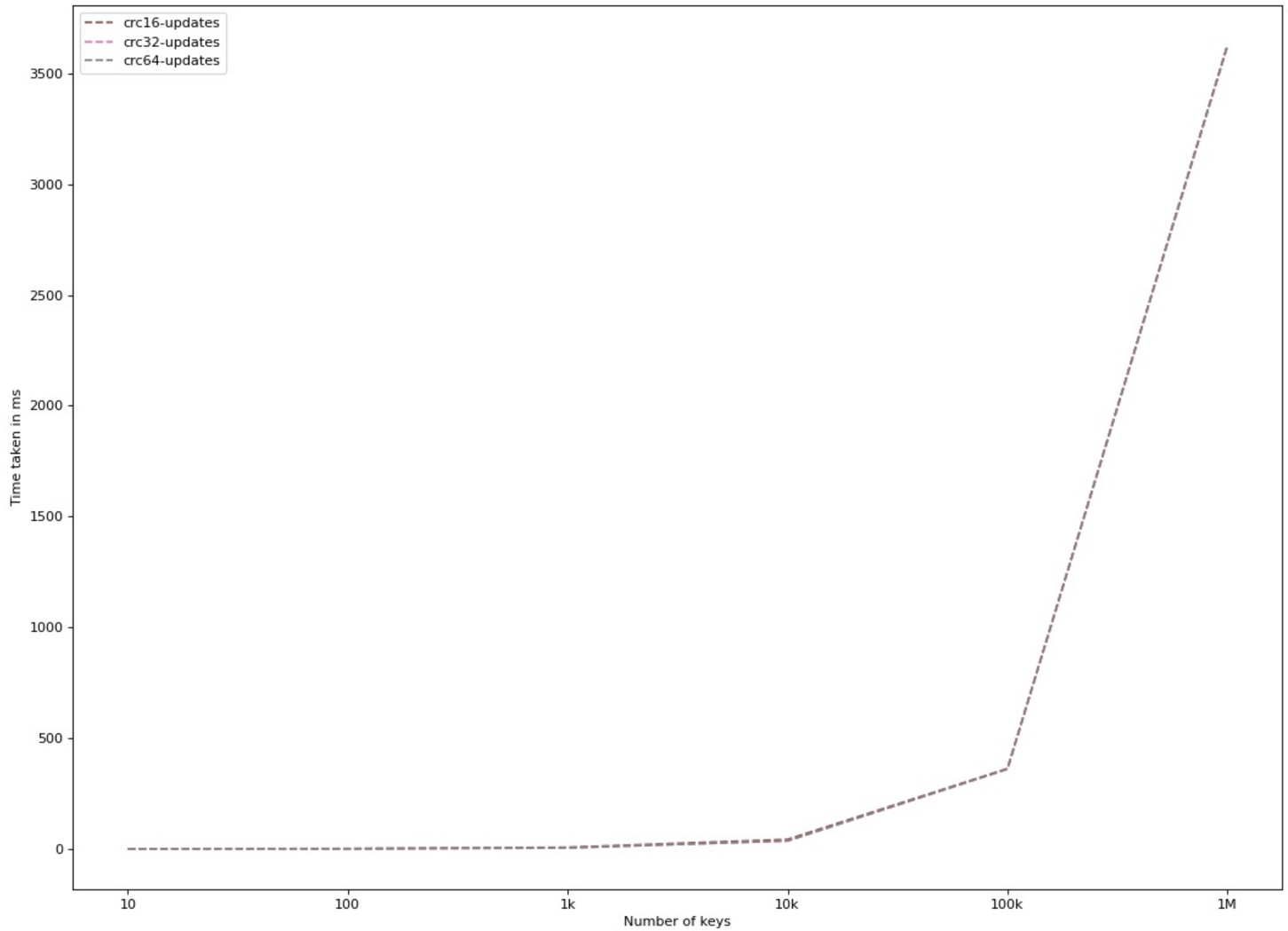
- Use CRC hashes
- Sort data before hashing



Run on Apple M1 processor with 8GB RAM

Performance of CRC hash

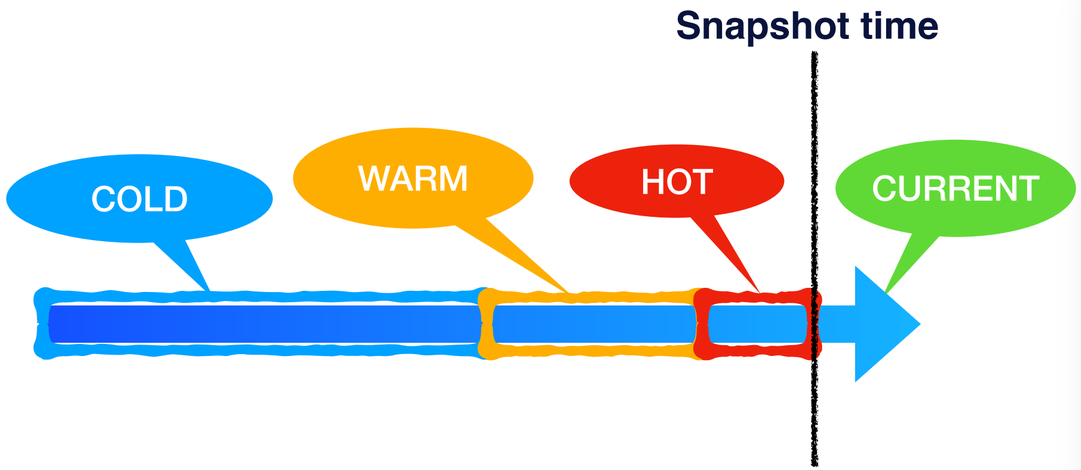
- Fast performance upto 10k records
- 64 bits CRC preferred
- Fingerprinting in a tree structure with a hash per interval/subinterval



Run on Apple M1 processor with 8GB RAM

Slicing time

Slicing time into eras



Cold

- Historical stable data
- Required usually for alignment at startup/after a long disconnect

Warm

- Usually stable
- Required when a storage is taken off for maintenance and brought up back again

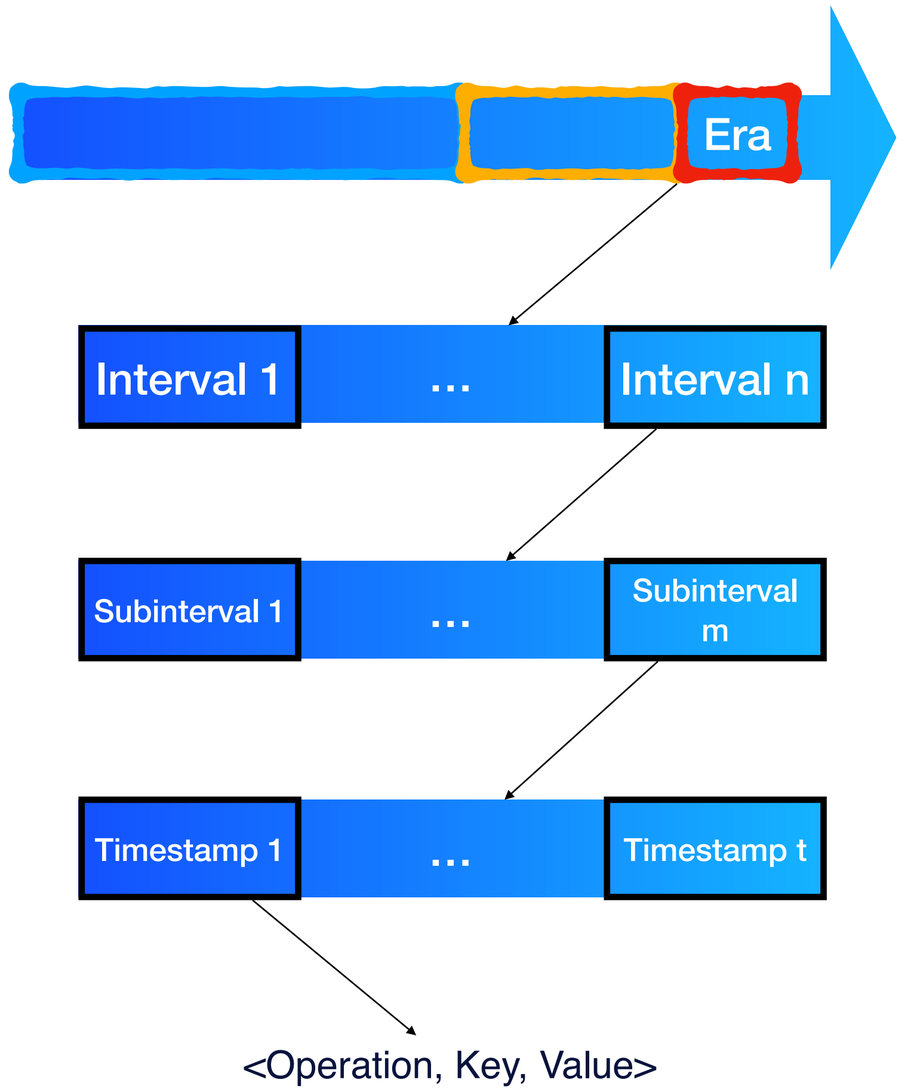
Hot

- Fast-changing data, almost sure all messages have been delivered
- To be aligned in case of temporary network disruptions

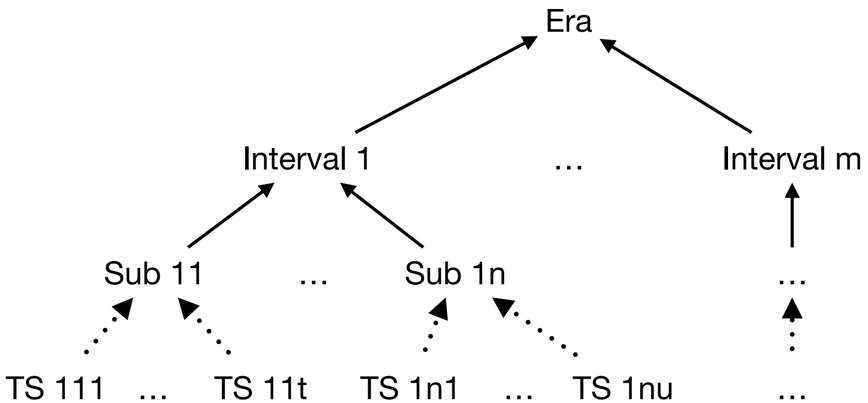
Current

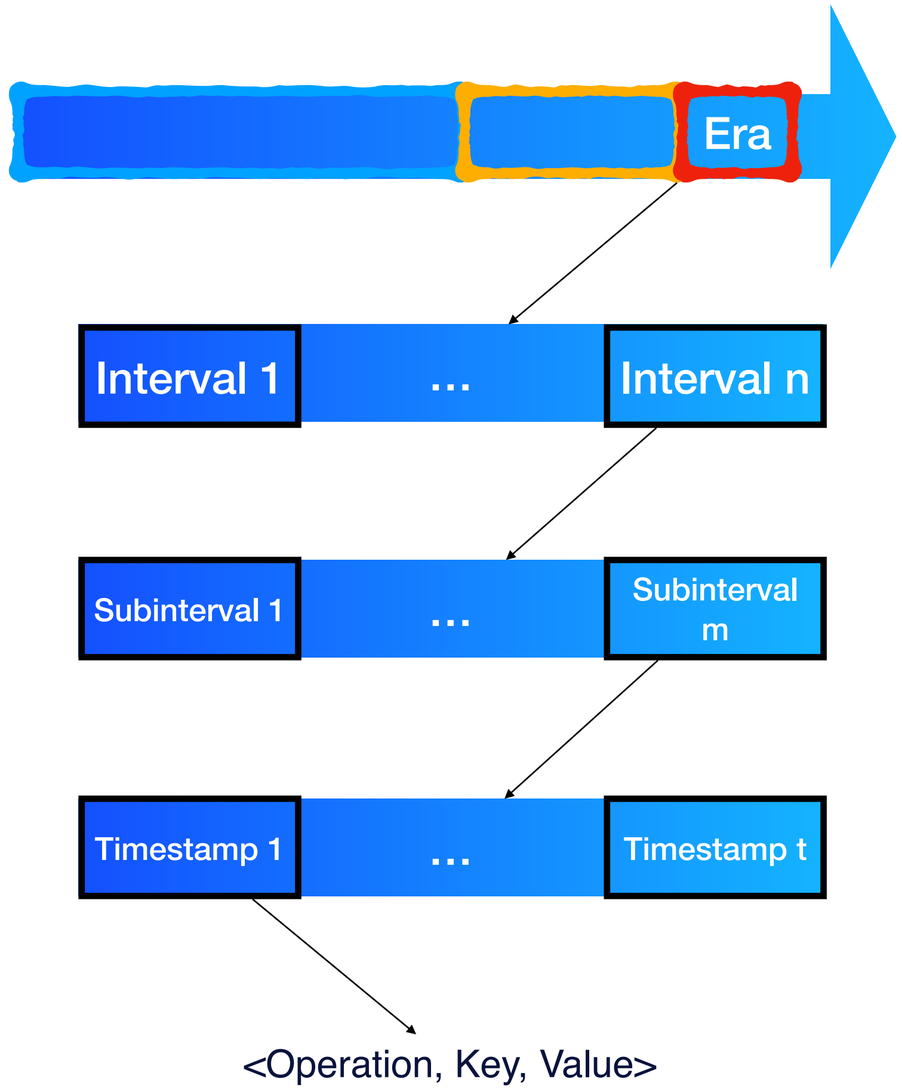
- Fast changing data, messages might still be in transit
- Not meant to be aligned

Composition of era

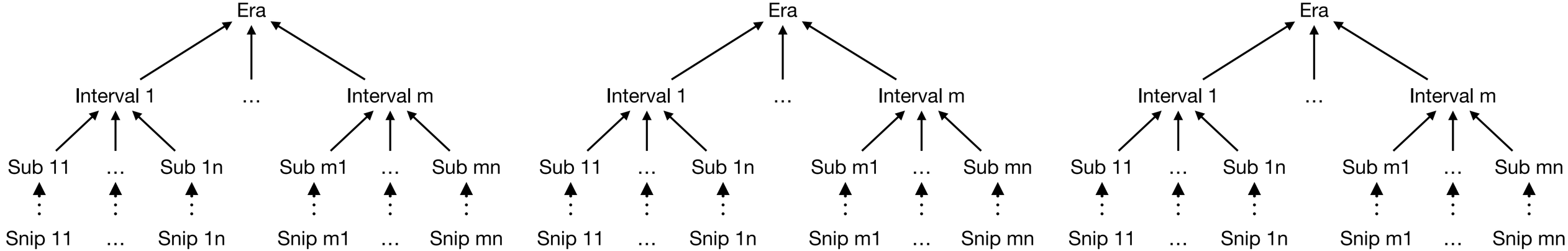


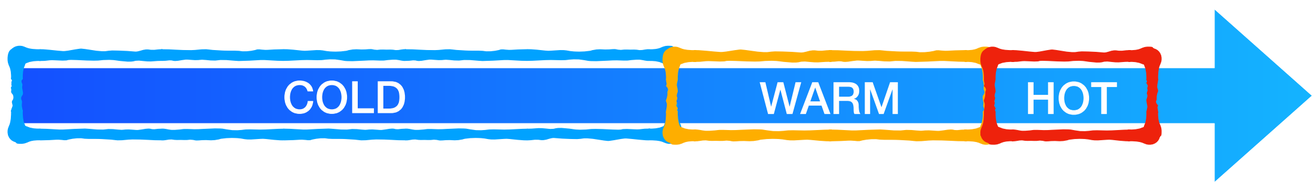
Composition of era



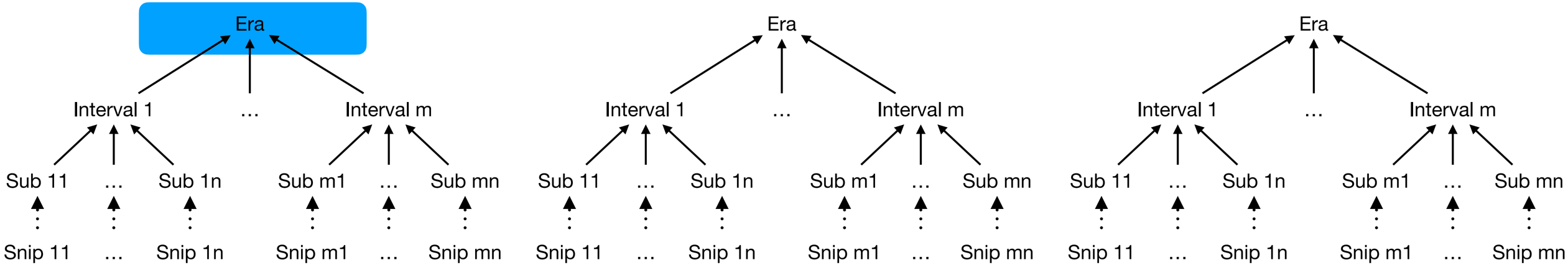


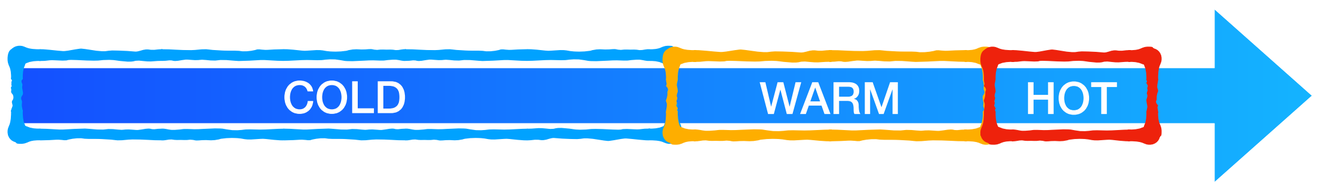
Compressing the digest



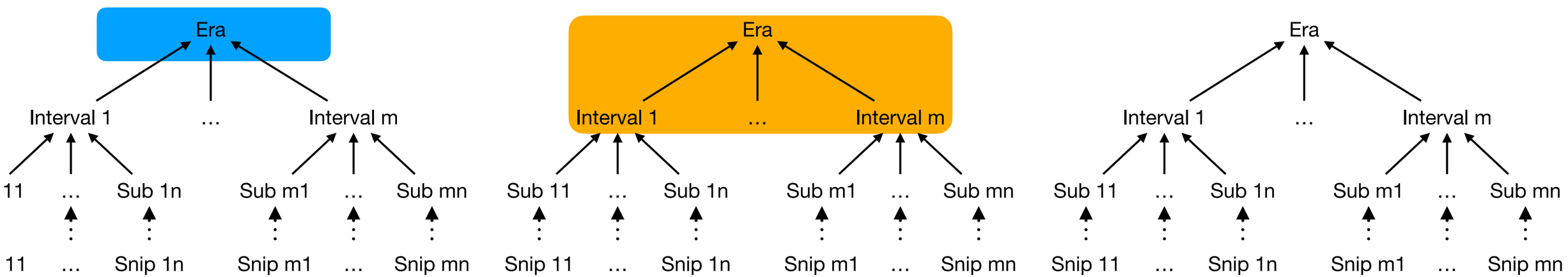


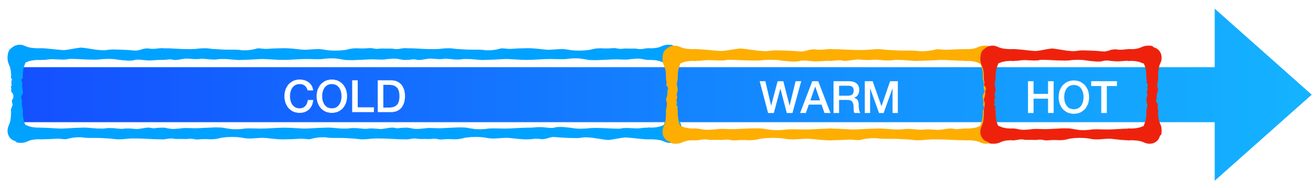
Compressing the digest



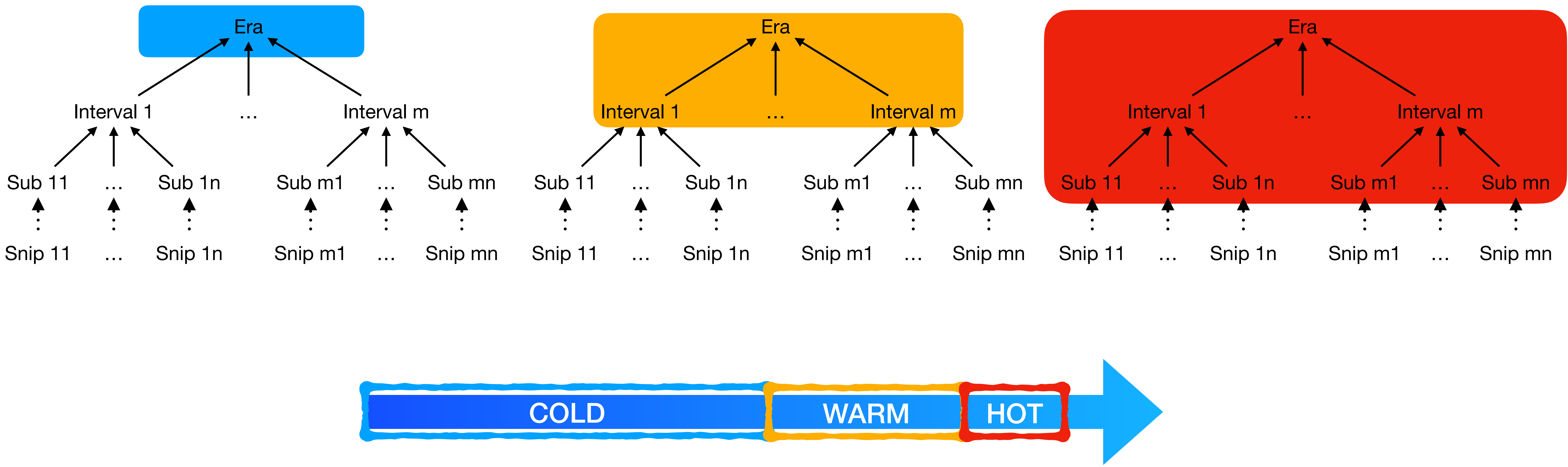


Compressing the digest

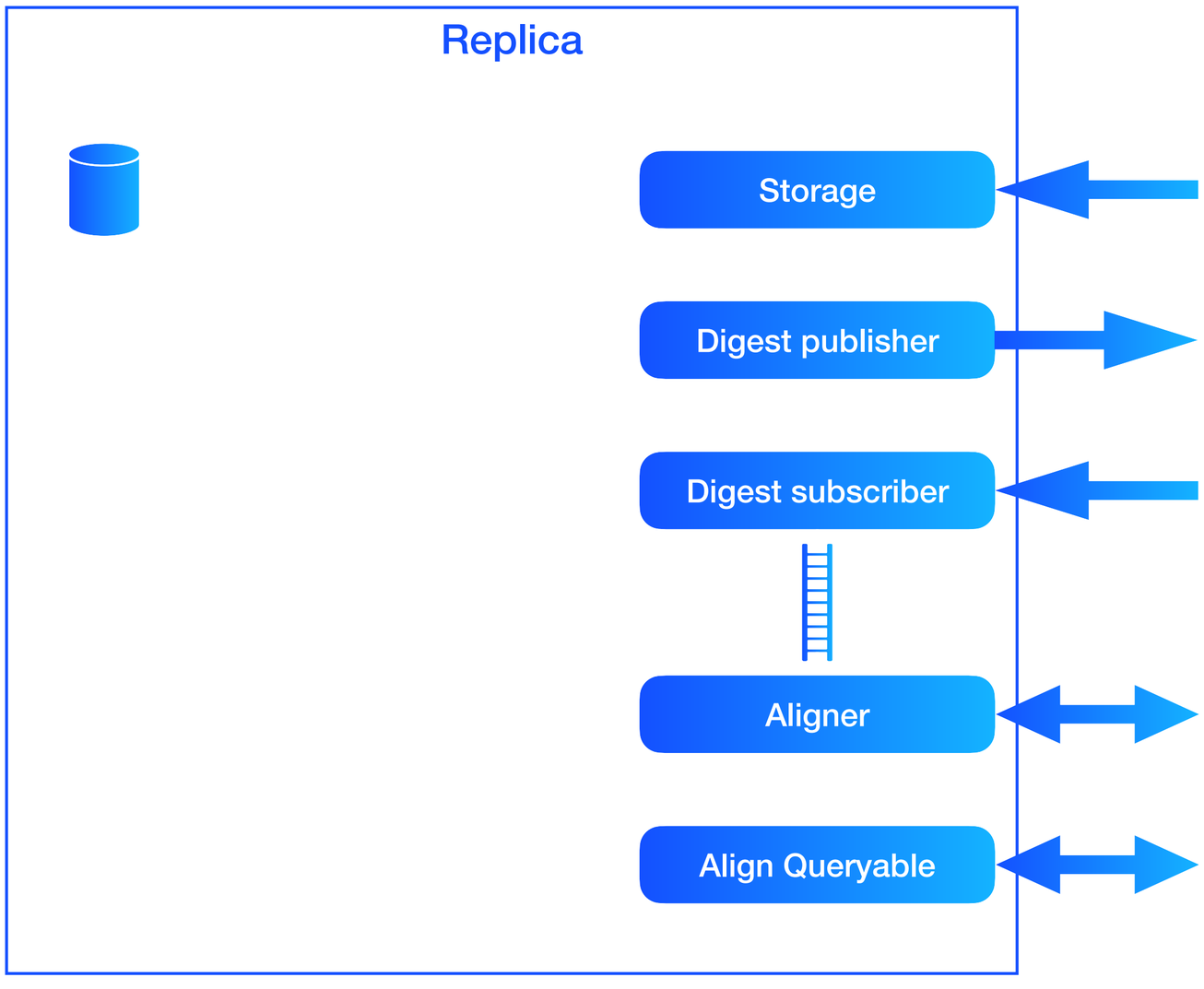




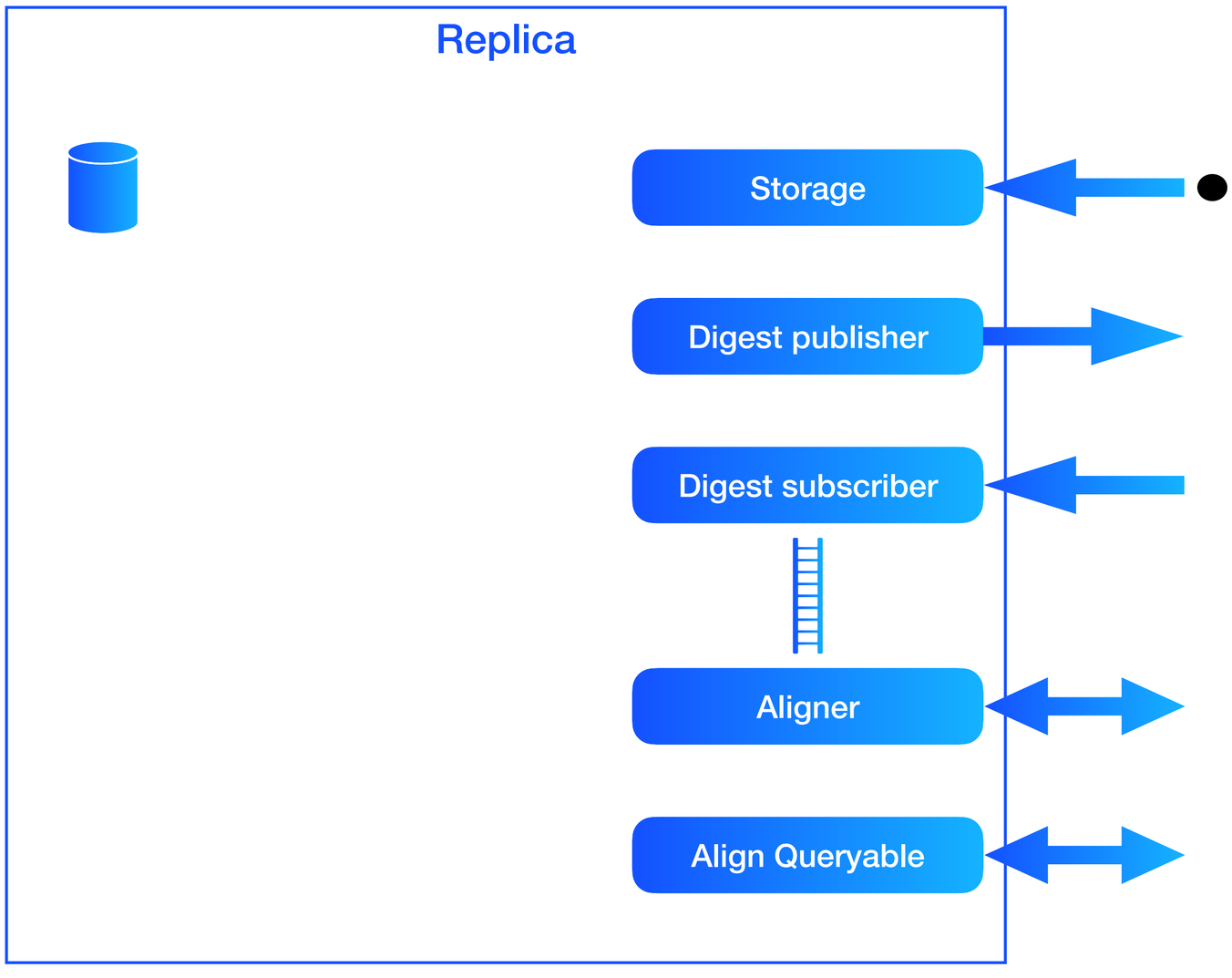
Compressing the digest



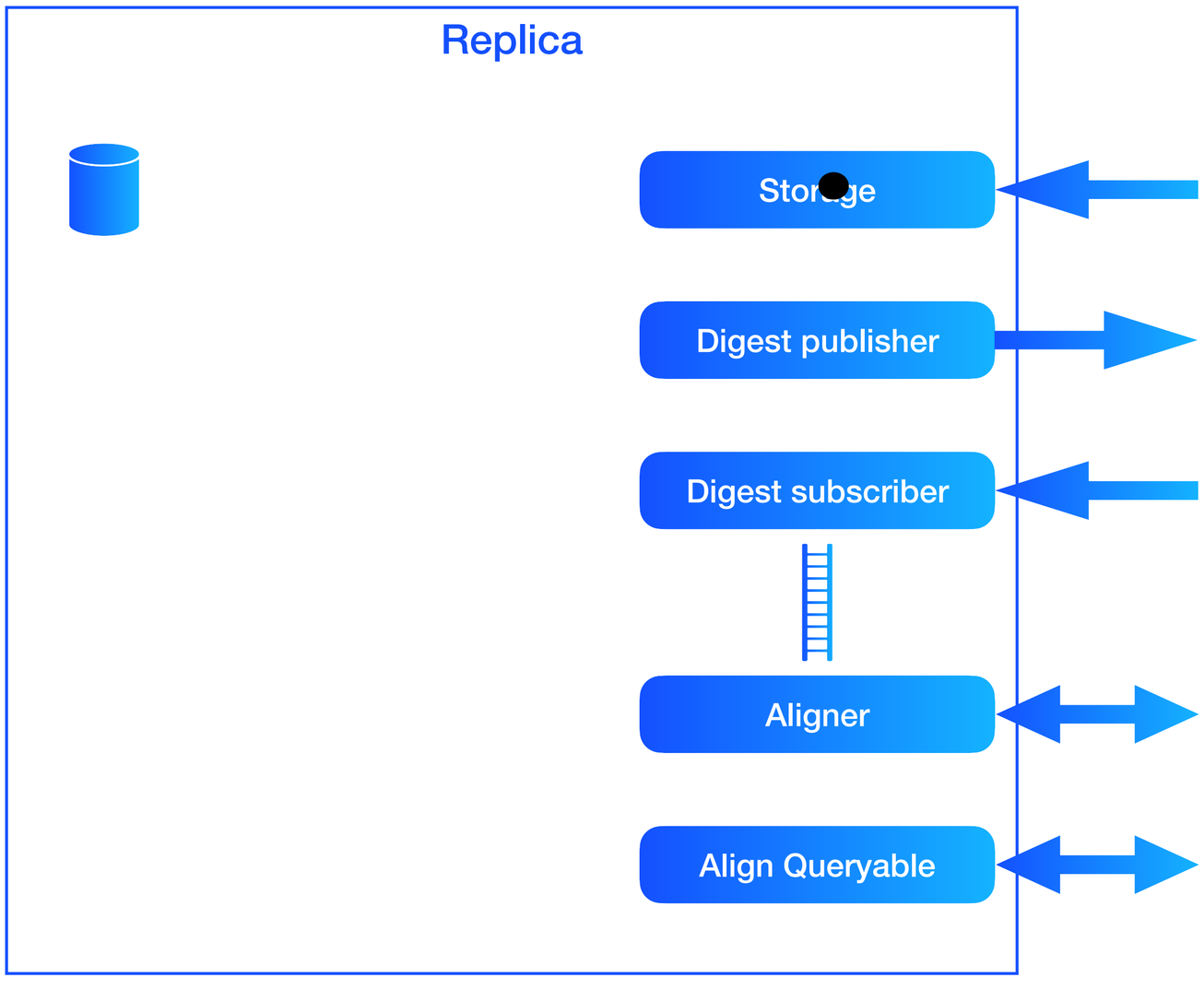
Zenoh Replica



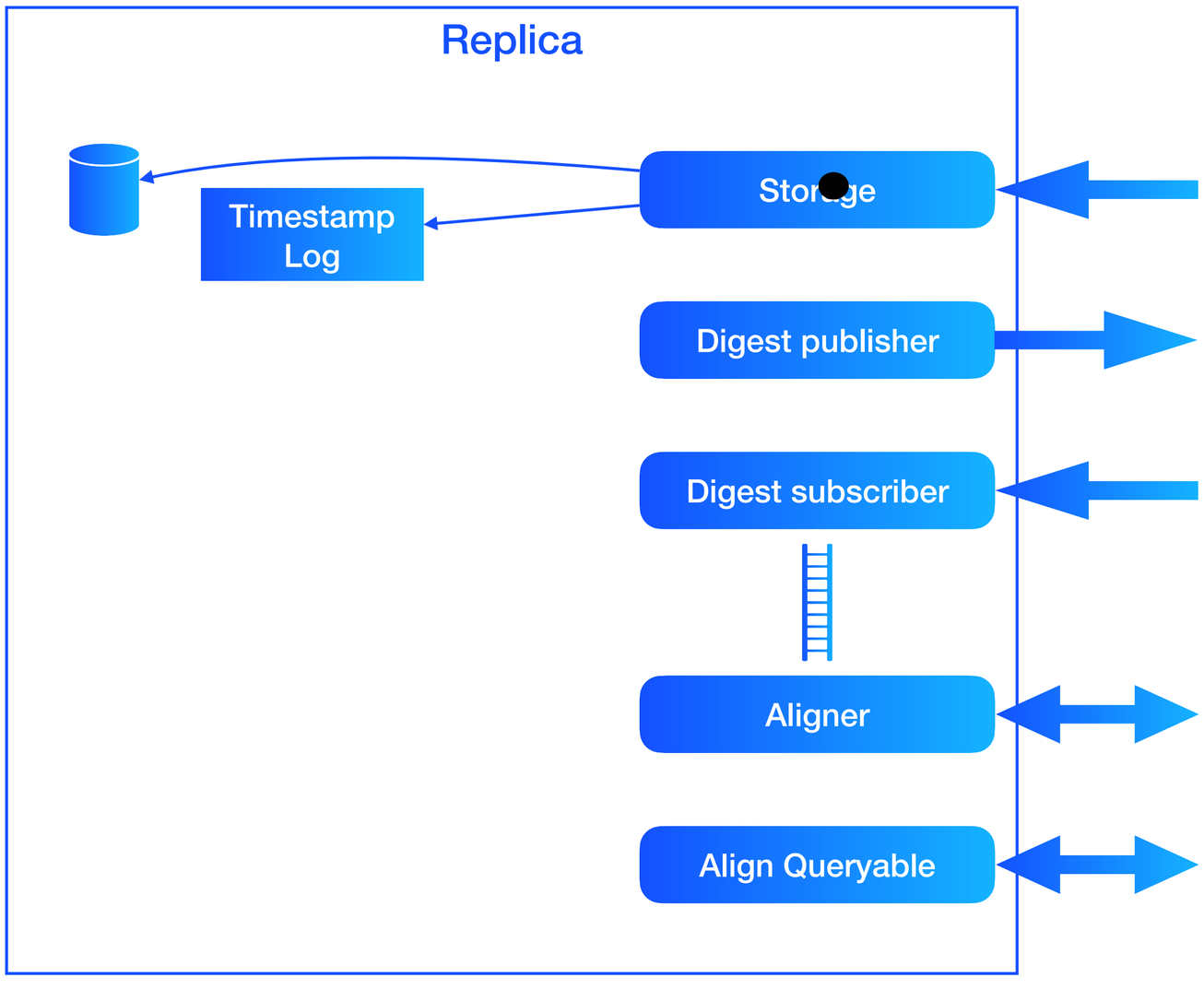
Zenoh Replica



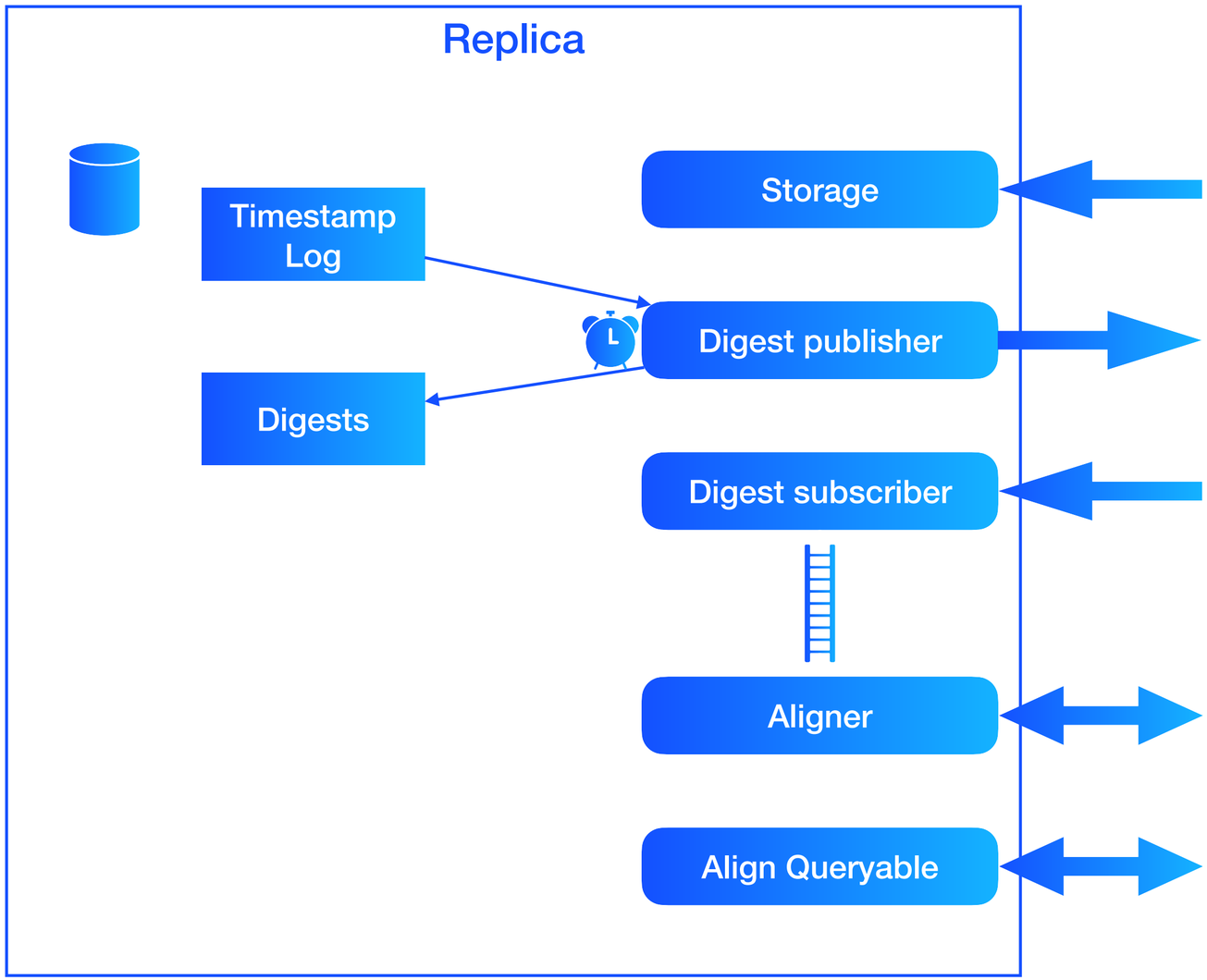
Zenoh Replica



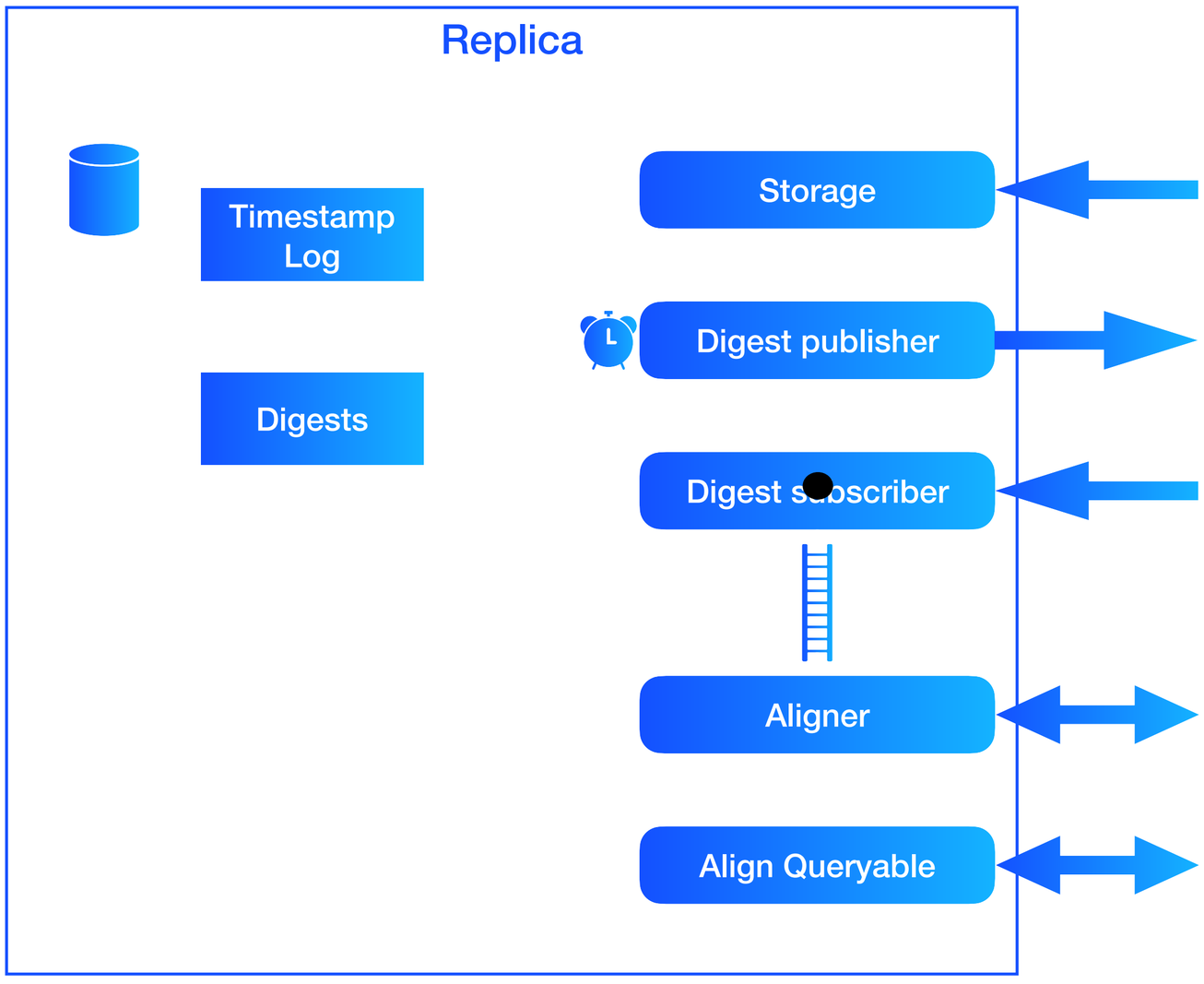
Zenoh Replica



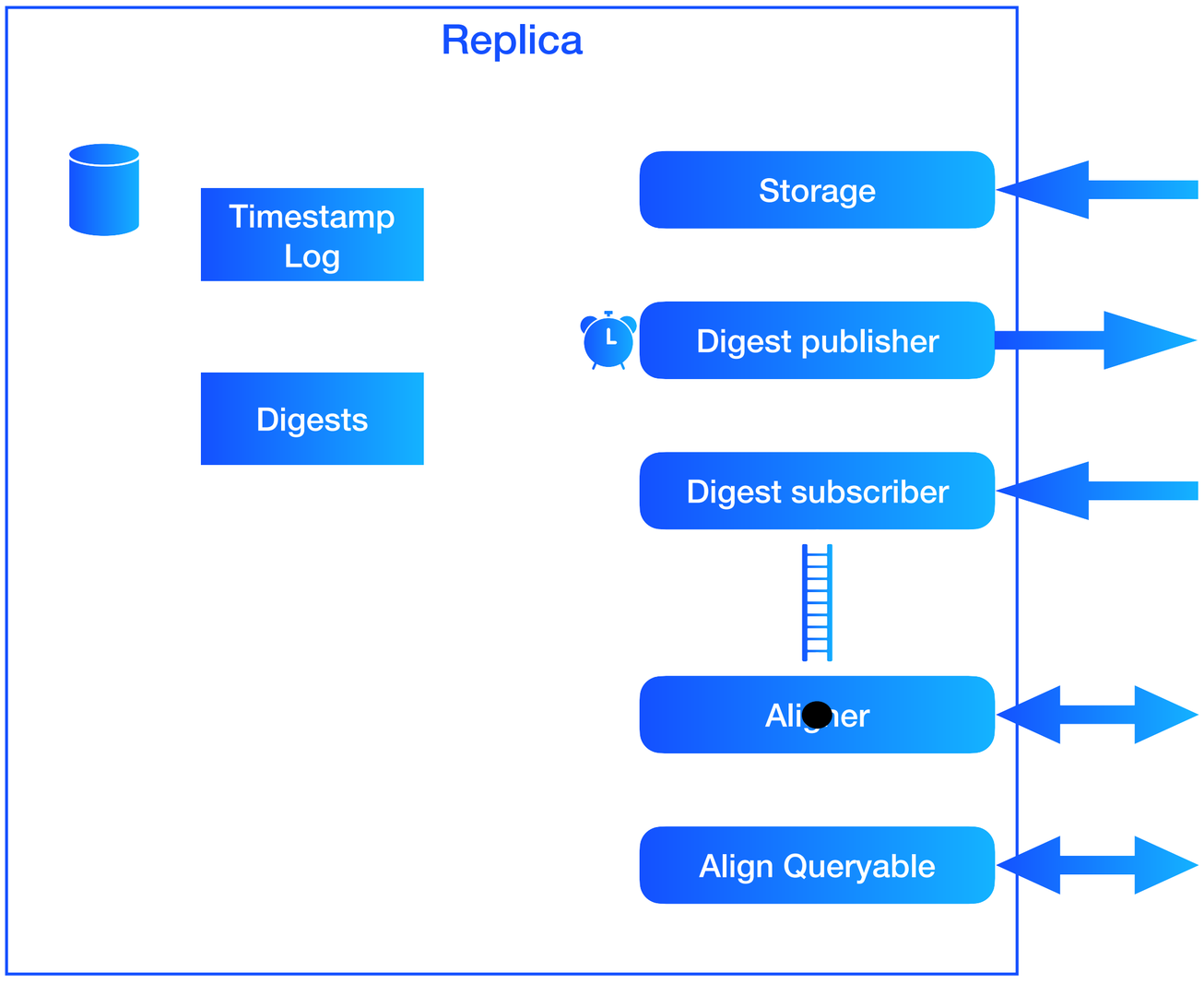
Zenoh Replica



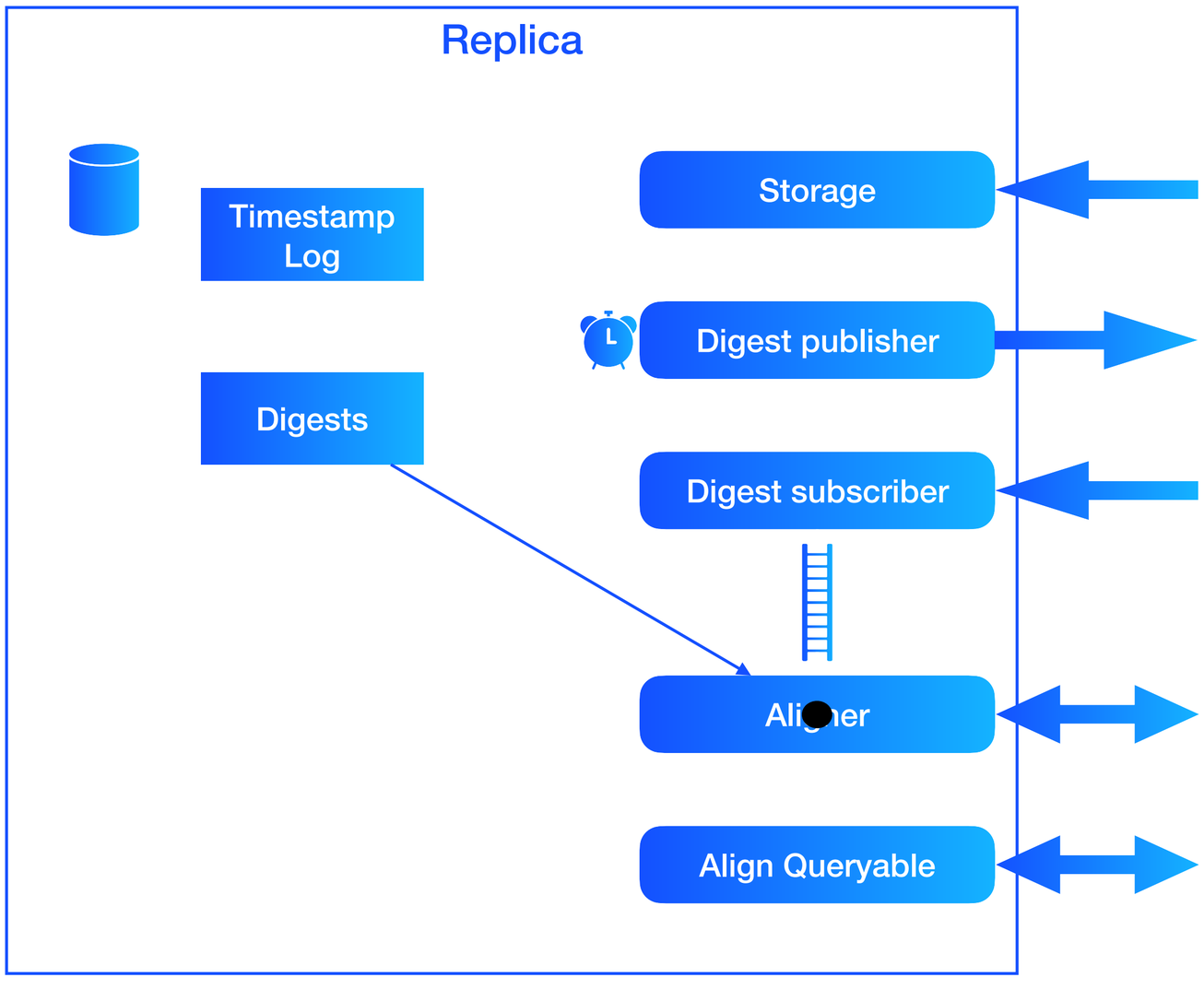
Zenoh Replica



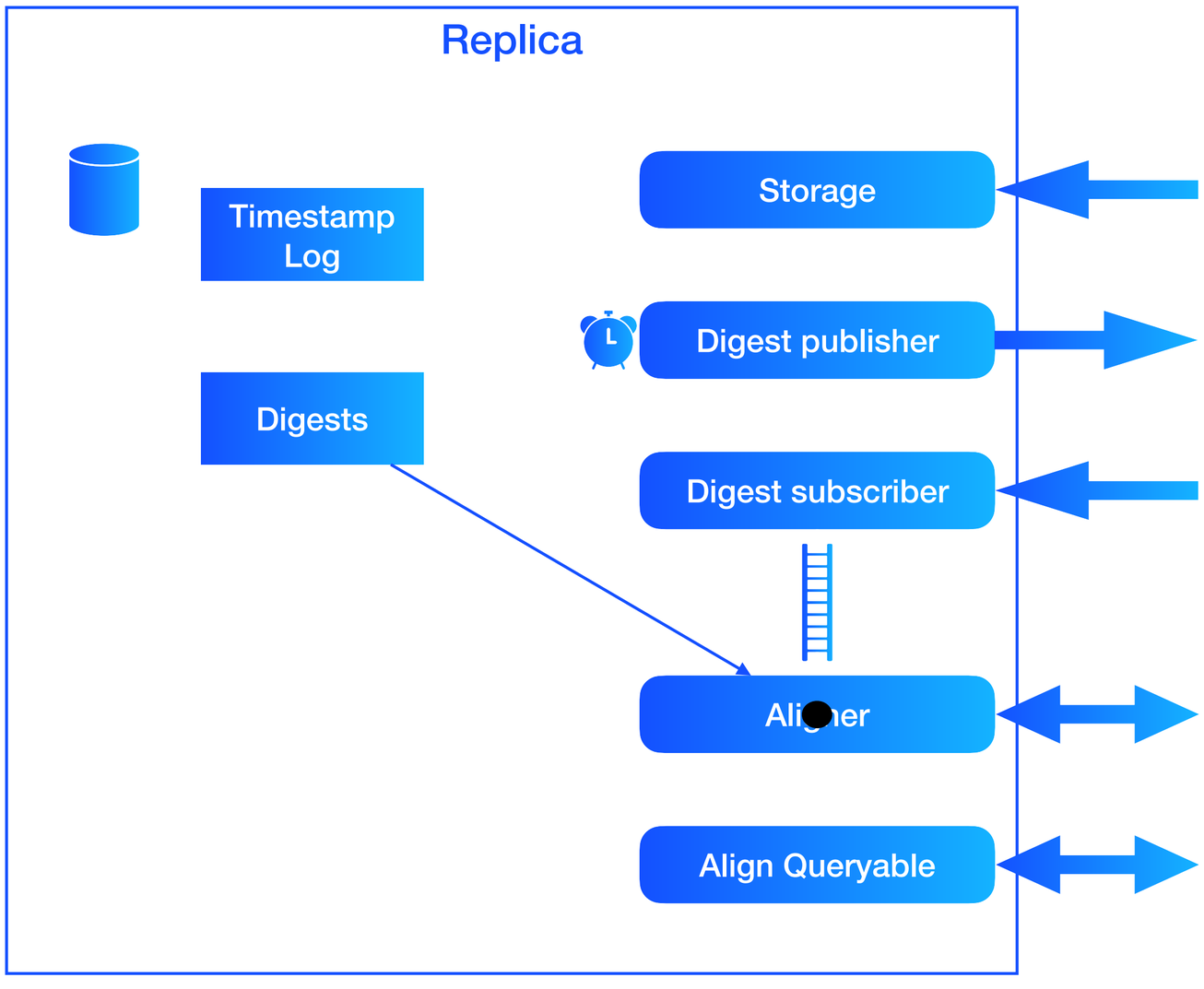
Zenoh Replica



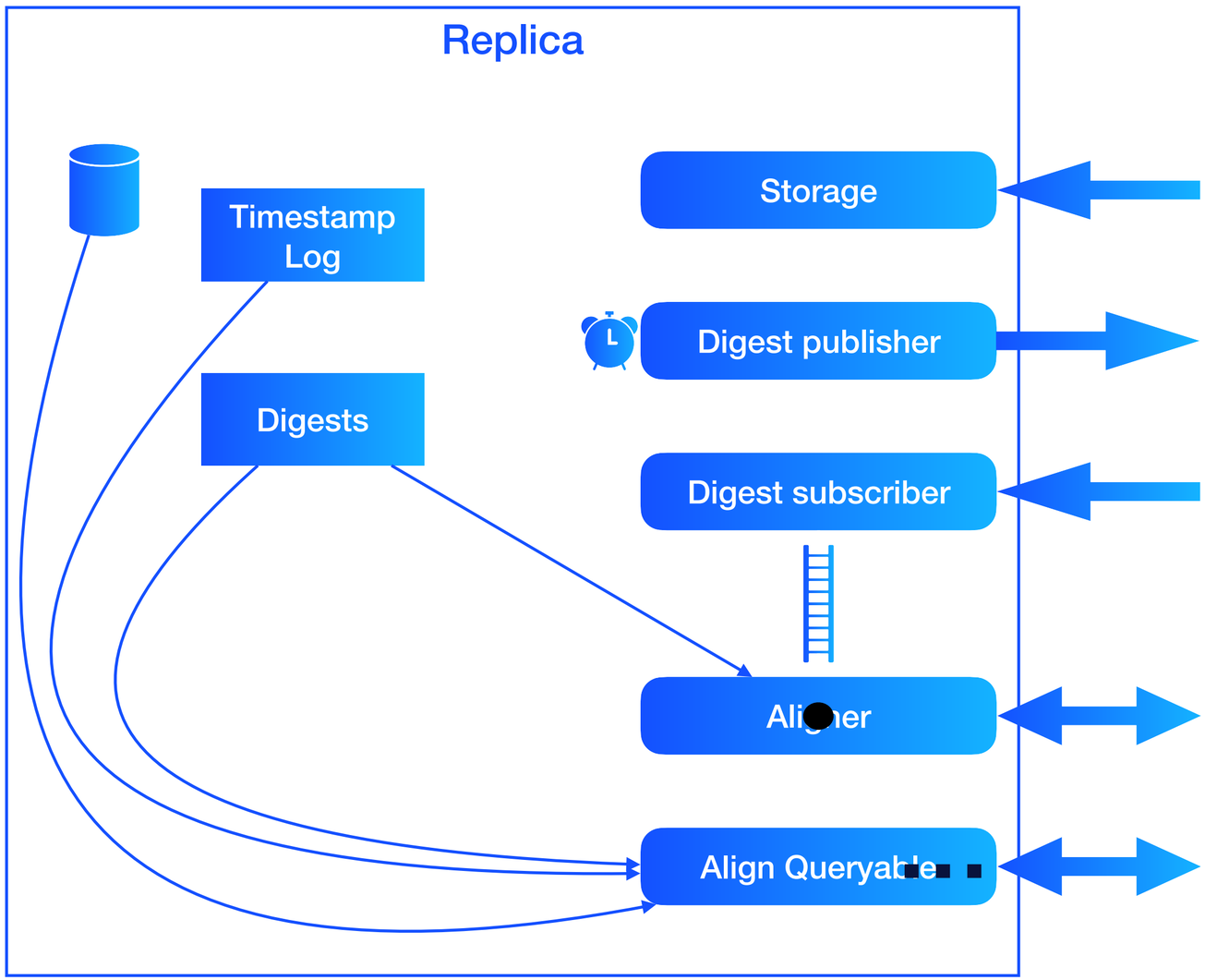
Zenoh Replica



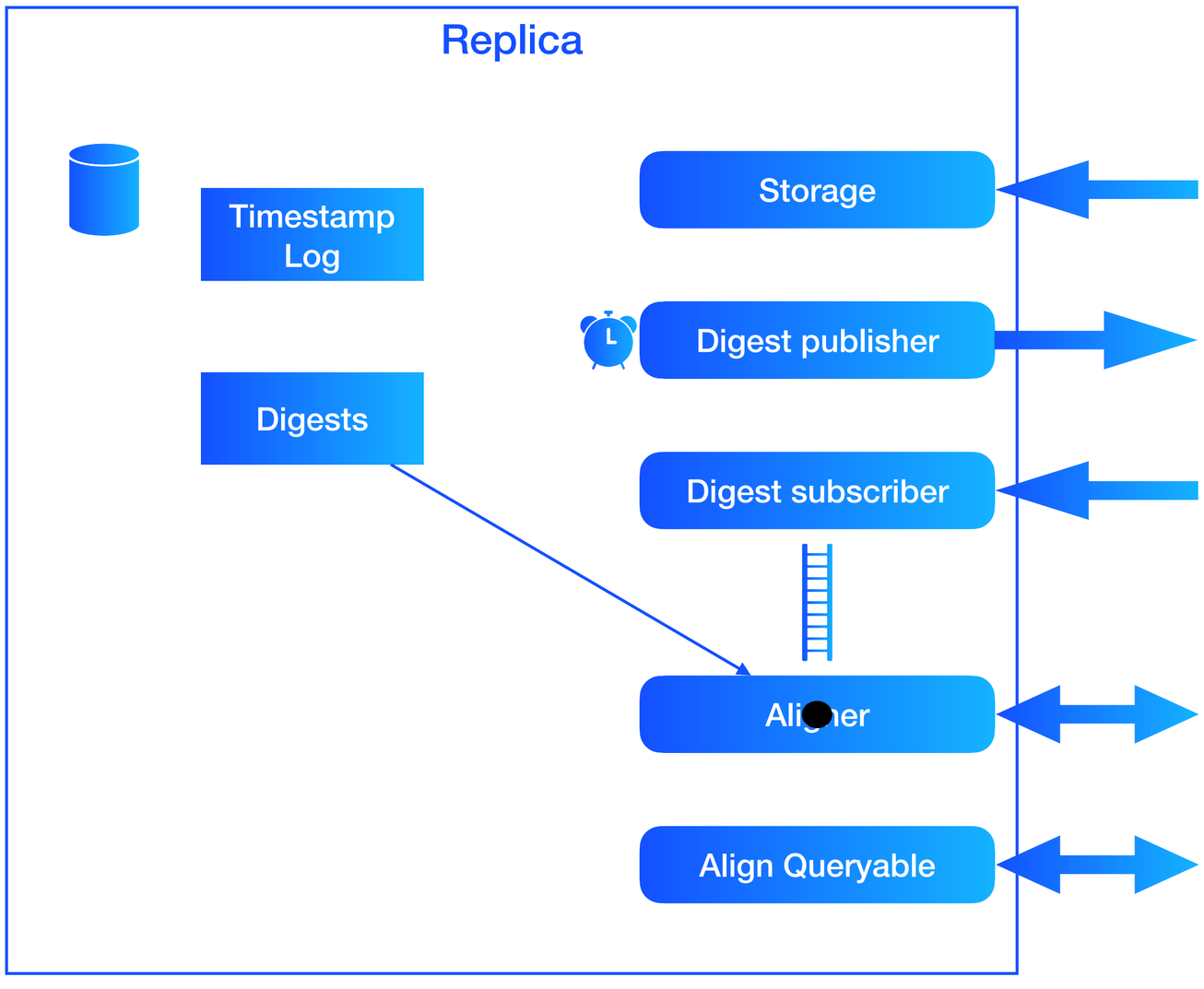
Zenoh Replica



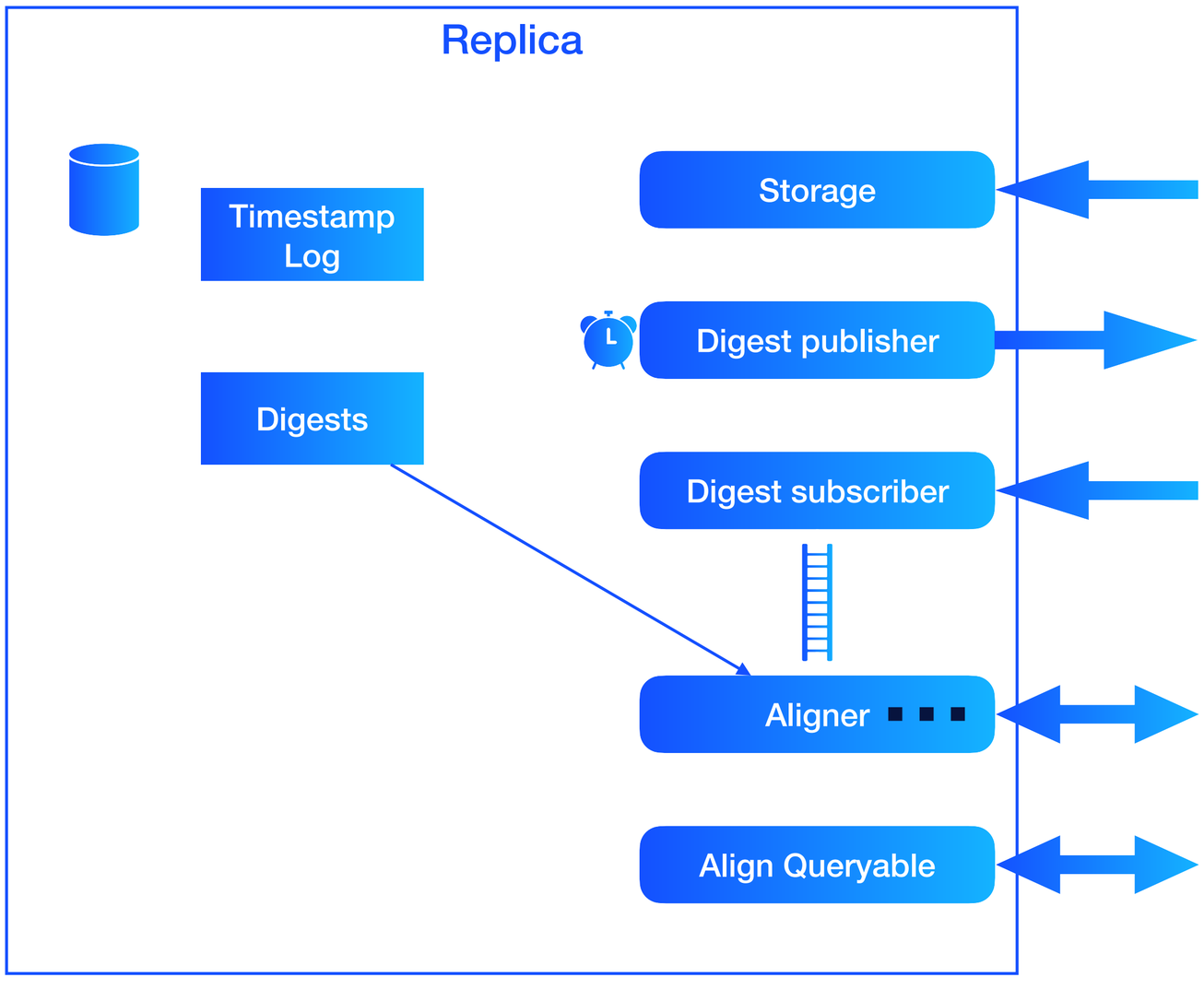
Zenoh Replica



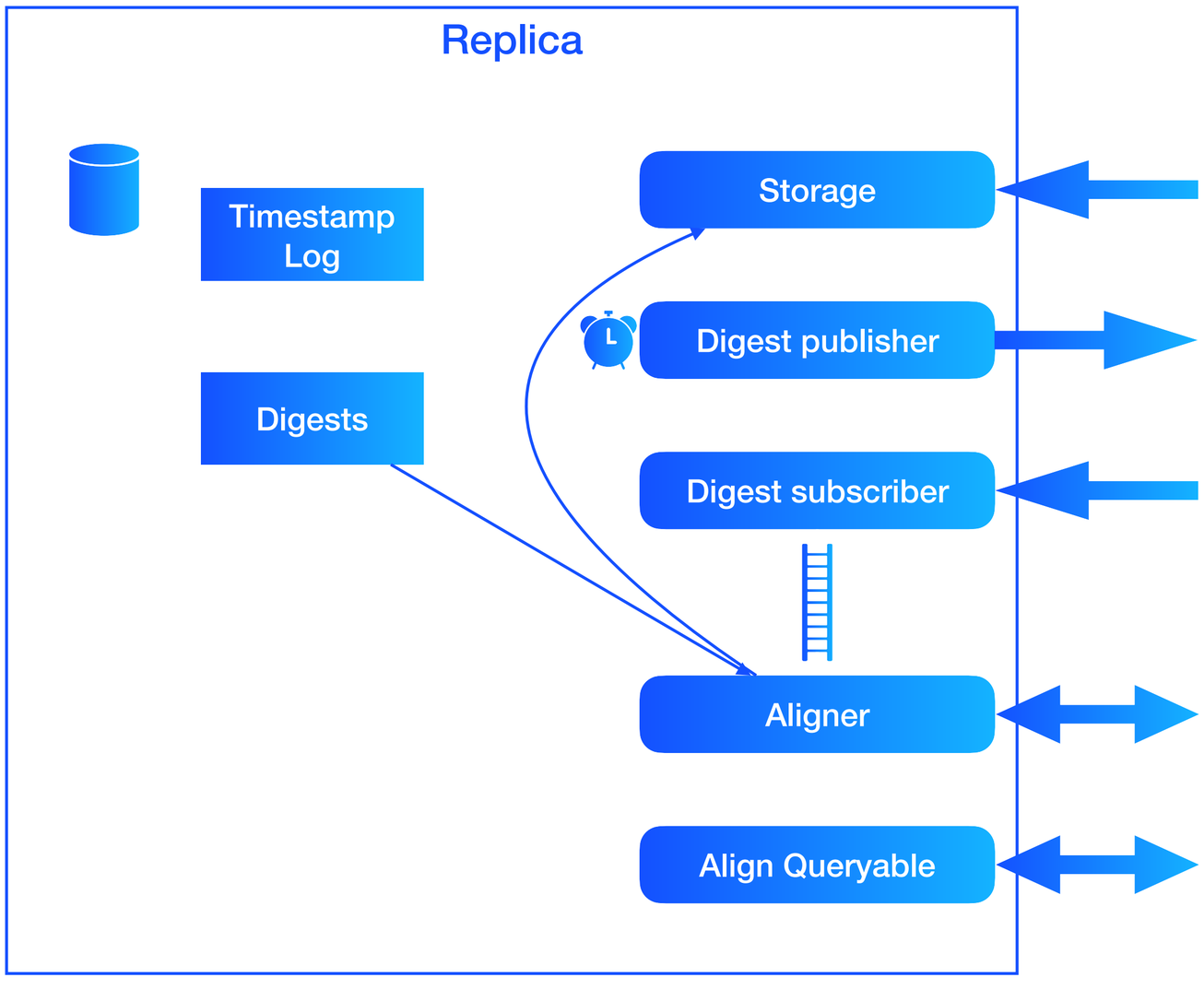
Zenoh Replica



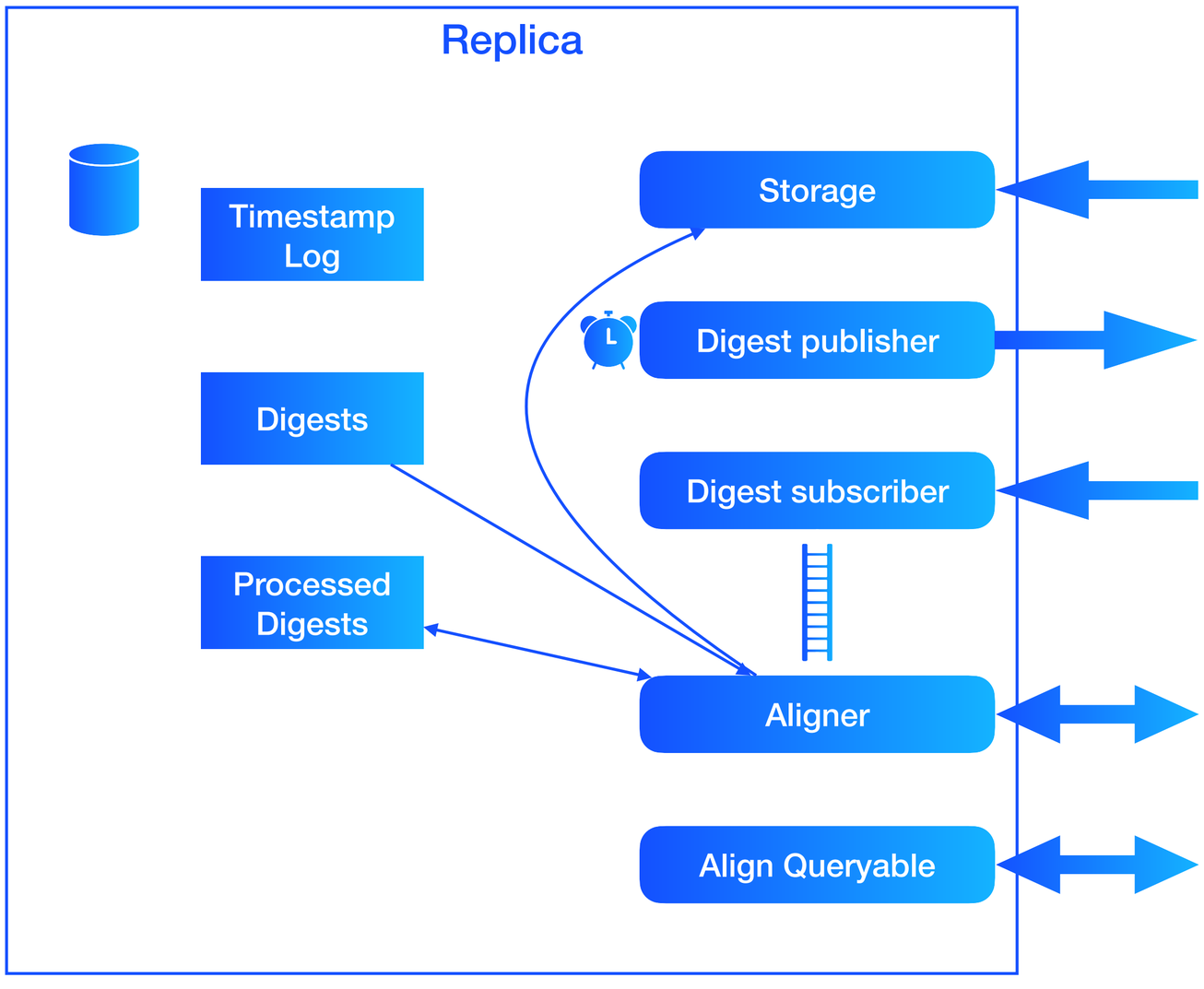
Zenoh Replica



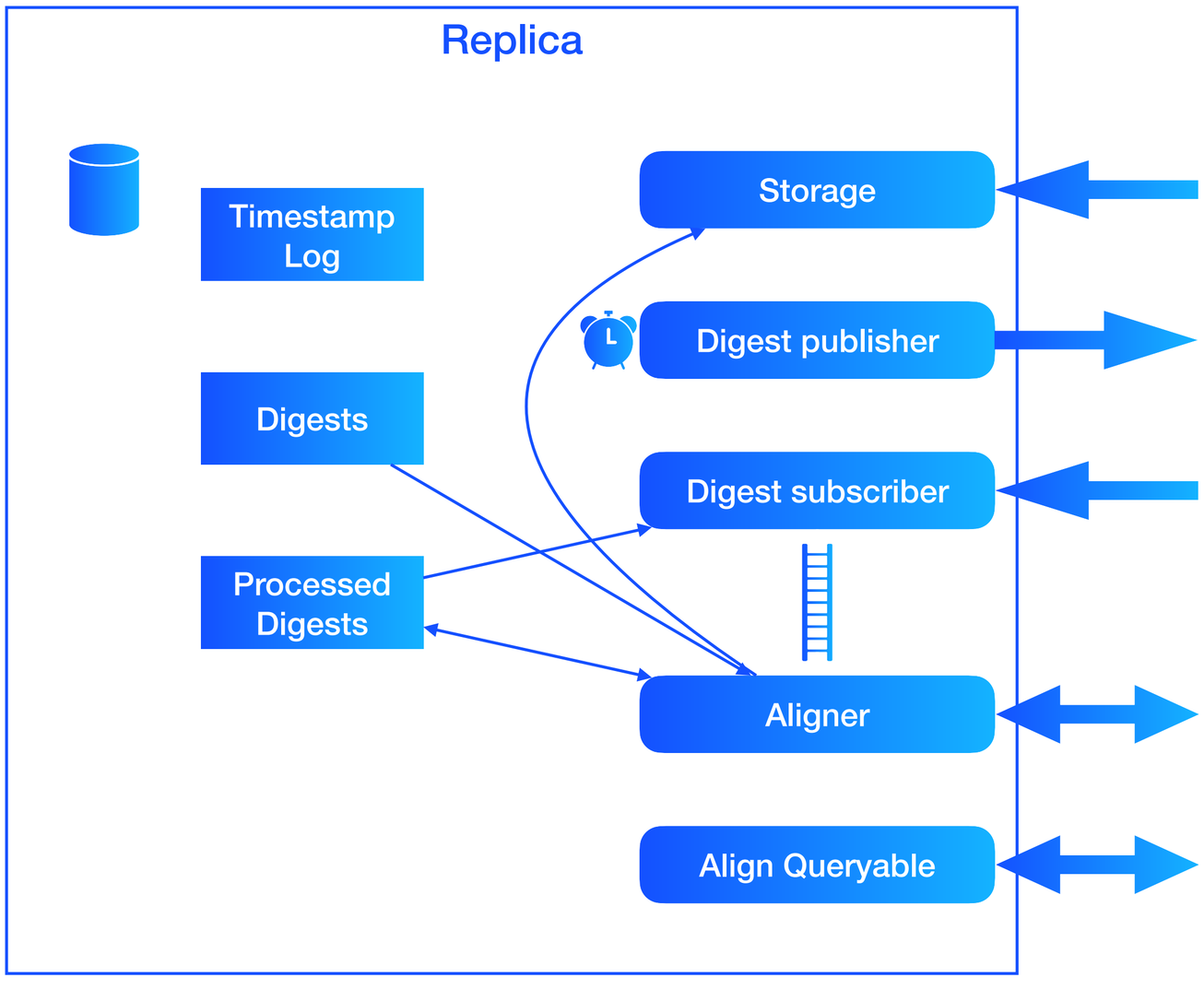
Zenoh Replica



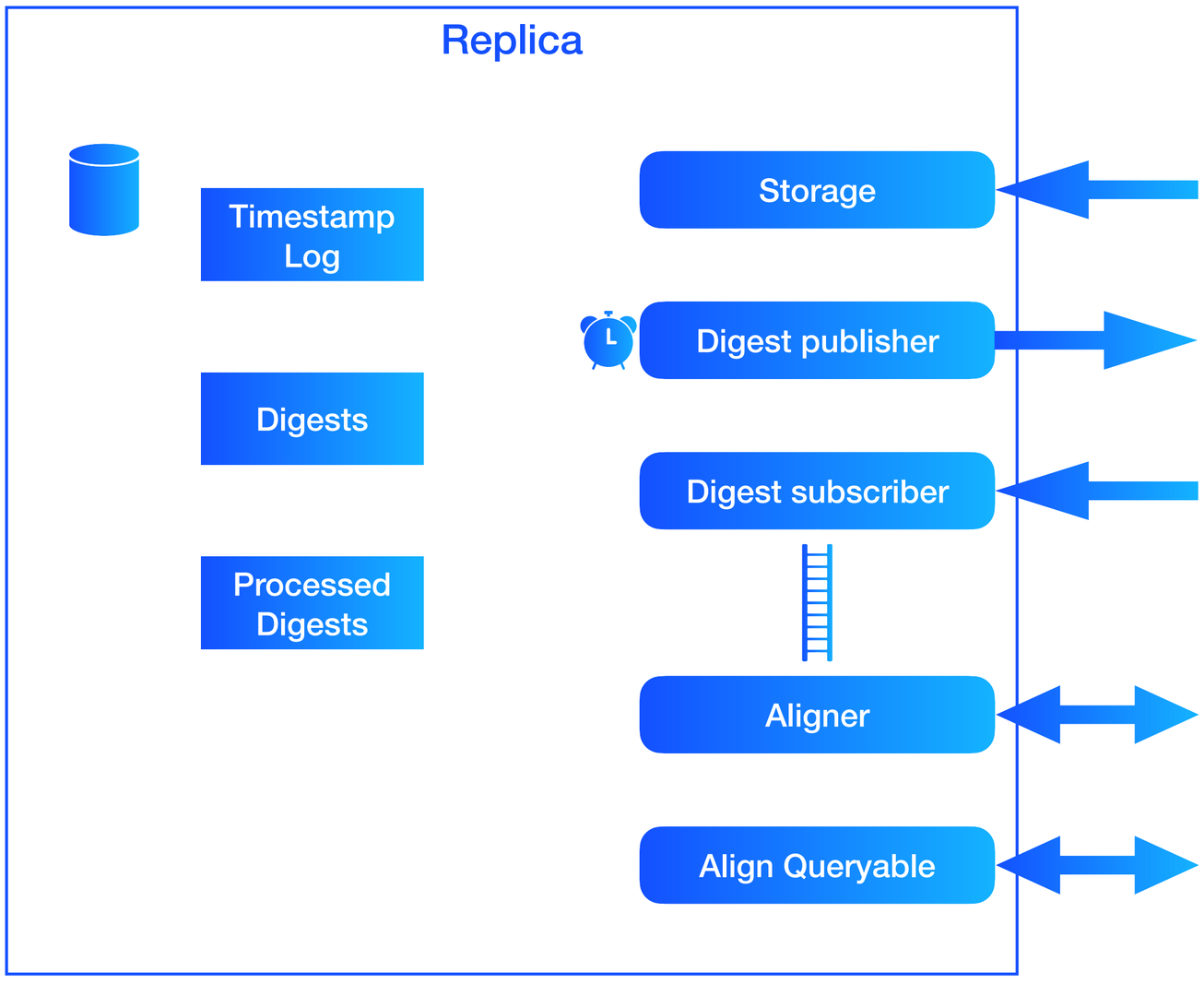
Zenoh Replica



Zenoh Replica



Zenoh Replica



Steps towards Eventual Consistency

1. Key value store, full replication support
2. Time series data store, full replication support
3. Partial replication support
4. Support out-of-zenoh data

Conclusion

A replication algorithm that supports storing data anywhere in the device to cloud continuum

Supports different storage technologies

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

zetta scale

Thank You

www.zettascale.tech
sreeja@zettascale.tech

Backup

Design decisions

- **Log of timestamps**
  - Have only two versions - stable and transient
    - Require formal arguments for safety
- **Snapshots**
  - Run a cleanup task for cleaning old snapshots
- **Generating timestamp log**
  - HYBRID - PULL at startup and INTERCEPT while running
    - Out-of-zenoh data might need PULL again periodically

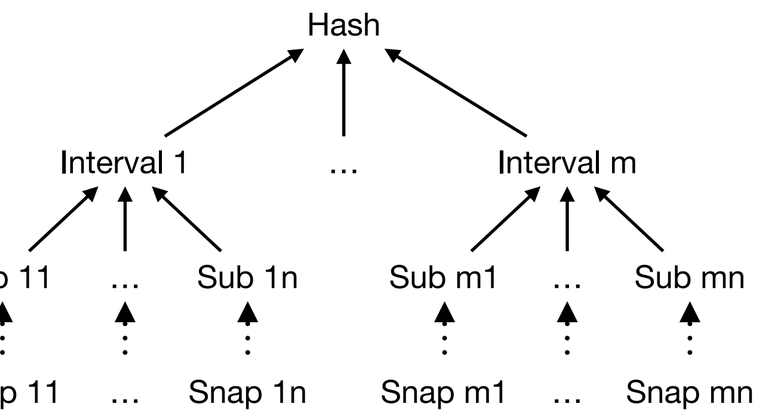
Versions

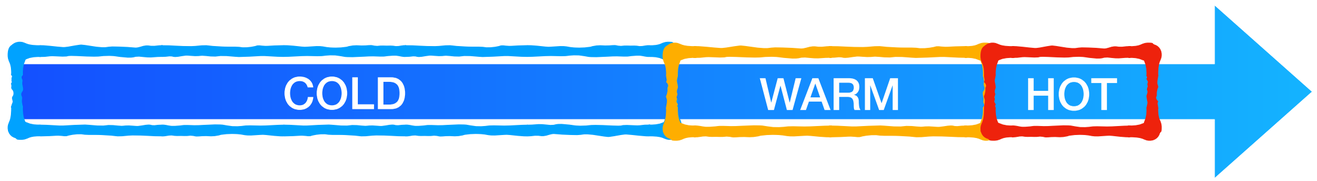
1. Key value store, full replication support
2. **Time series data store support**
3. Partial replication support
4. Support out-of-zenoh data

zetta scale

Digest for a time-series data store

Compressing digest

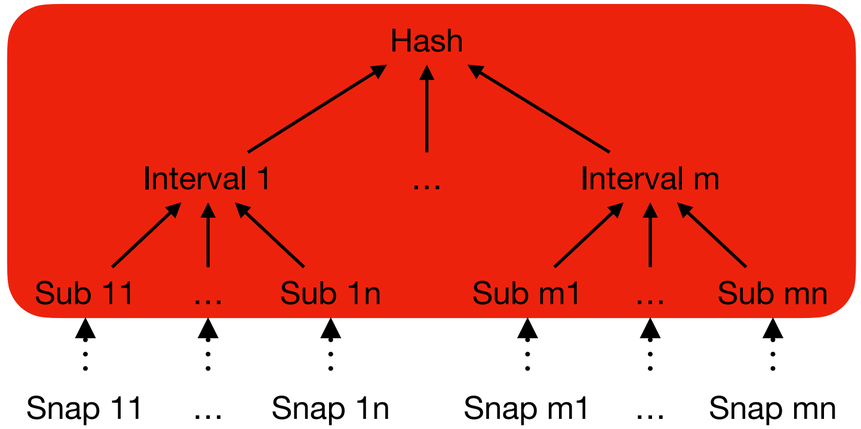


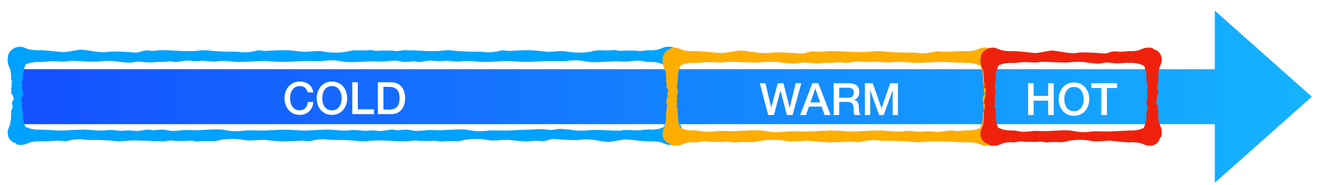


zetta scale

Digest for a time-series data store

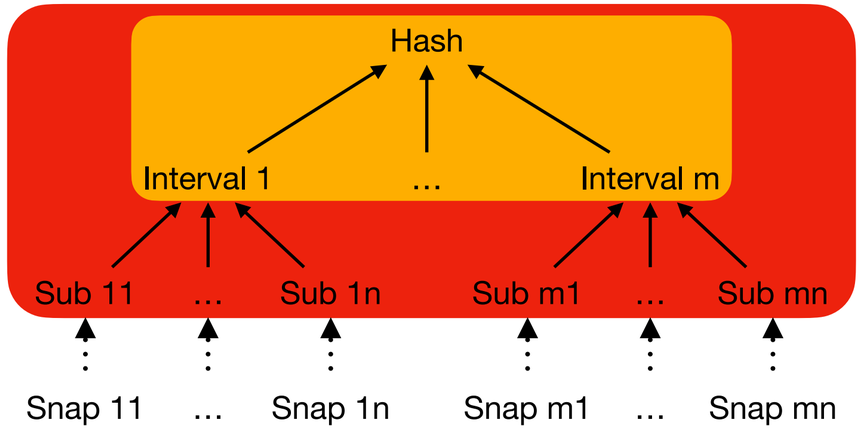
Compressing digest

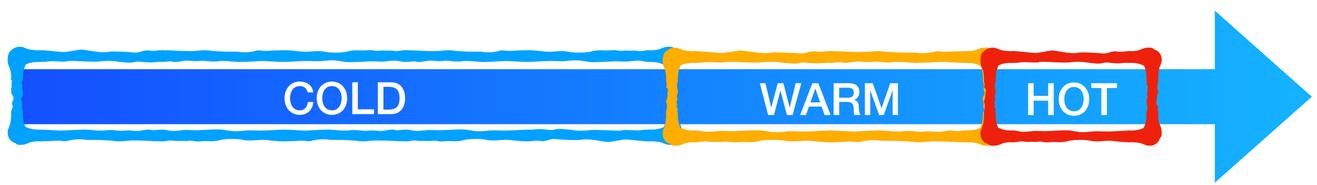




Digest for a time-series data store

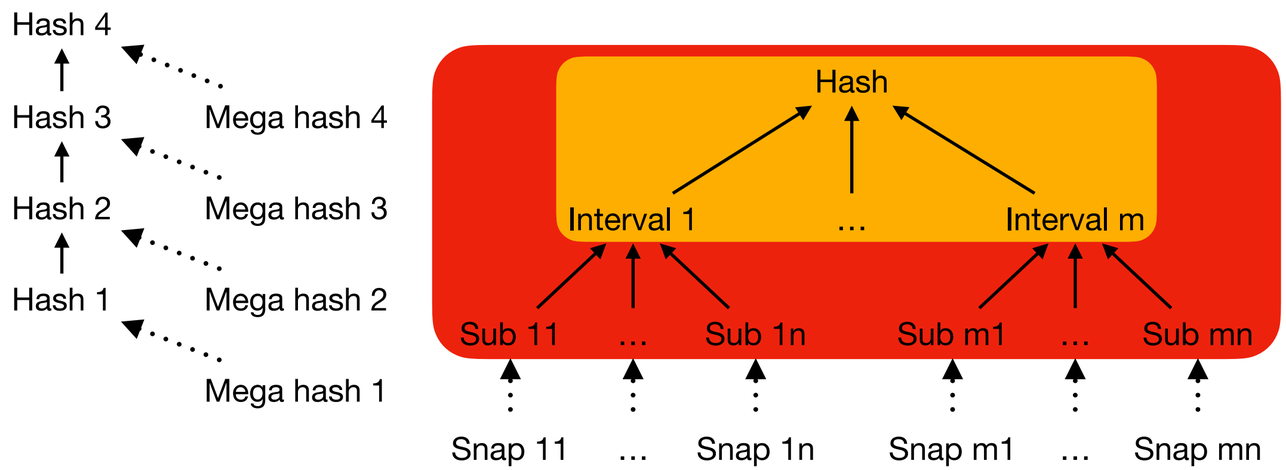
Compressing digest

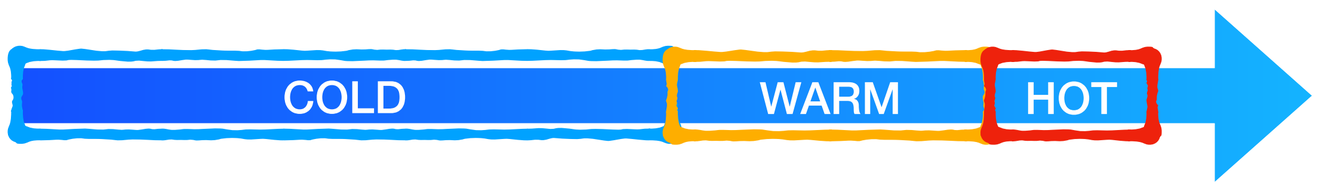




Digest for a time-series data store

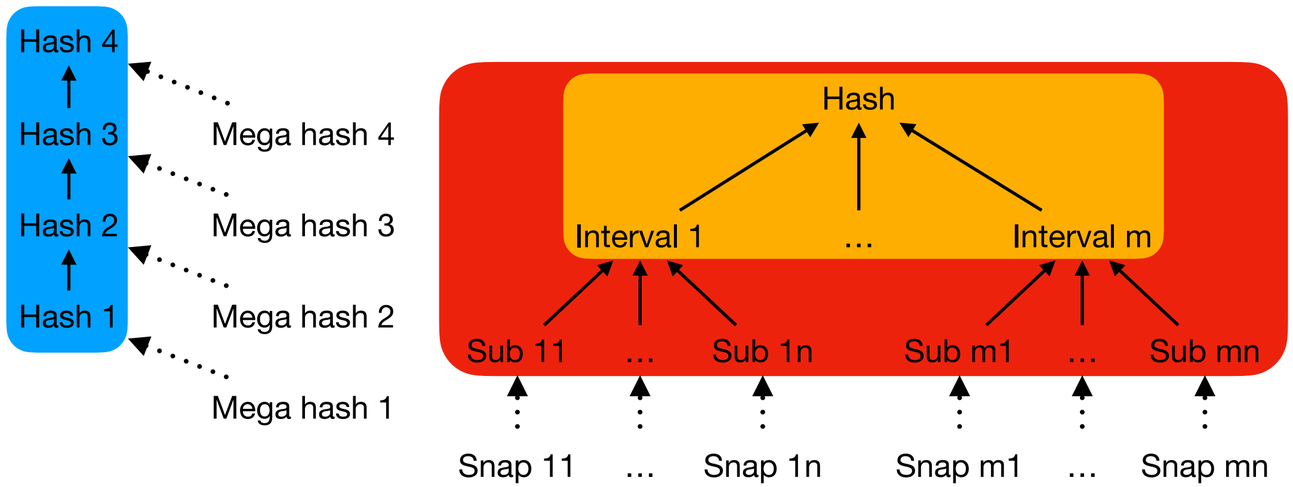
Compressing digest

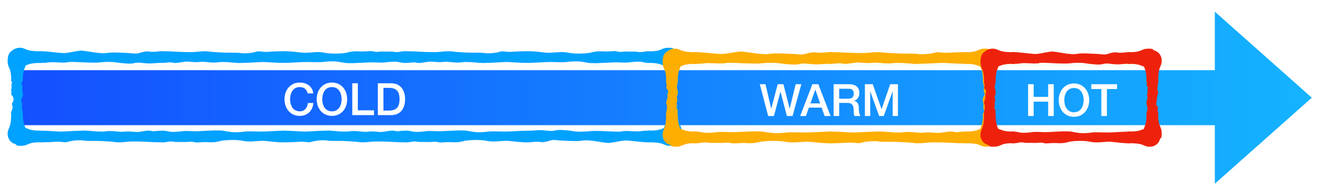




Digest for a time-series data store

Compressing digest





Comparing digest creation

Key value	COLD	WARM	HOT
Hashing methodology	Subinterval based		
Hash Structure	Shallow tree with 3 levels		
Storage	Entire tree stored		
Transmission	Top level only	First two levels	Entire tree
Misalignment detection	1 - era content 2 - interval content 3- subinterval content	1- interval content 2- subinterval content	1- request subinterval content

Time series	COLD	WARM	HOT
Hashing methodology	Heirarchical + incremental	Subinterval based	
Hash Structure	Linked list	Shallow tree with 3 levels	
Storage	Entire list stored	Entire tree stored	
Transmission	Head only	First two levels	Entire tree
Misalignment detection	N- list entry n+1- mega content n+2- interval content n+3-	1- interval content 2- subinterval content	1- request subinterval content

Versions

1. Key value store, full replication support
2. Time series data store support
3. Partial replication support
    - Using trie with the existing hash tree
4. Support out-of-zenoh data
    - Might require consensus to assign HLC timestamp and then run the usual alignment algorithm

Open Questions

- Aligning key-value stores with time-series stores
- Content-dependent optimisations to minimise network overhead/ roundtrip

The important thing is not to stop questioning - Albert Einstein