PRISMTECH™
AN ADLINK COMPANY

Introducing DDS

Angelo Corsaro, PhD
Chief Technology Officer
ADLINK Technologies Inc.
angelo.corsaro@adlinktech.com

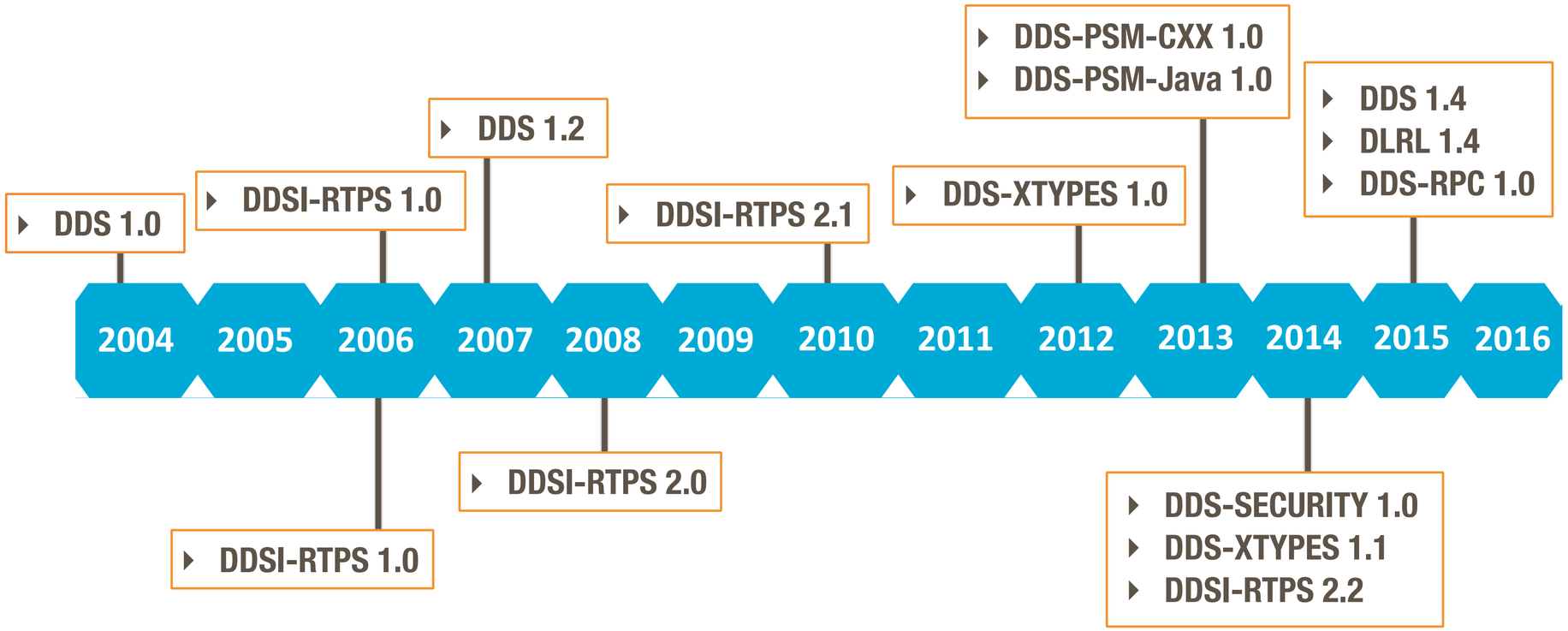
DDS, WHAT?

DDS IN 131 CHARACTERS

DDS is a standard technology for ubiquitous, interoperable, secure, platform independent, and real-time **data sharing** across network connected devices

THE DDS STANDARD

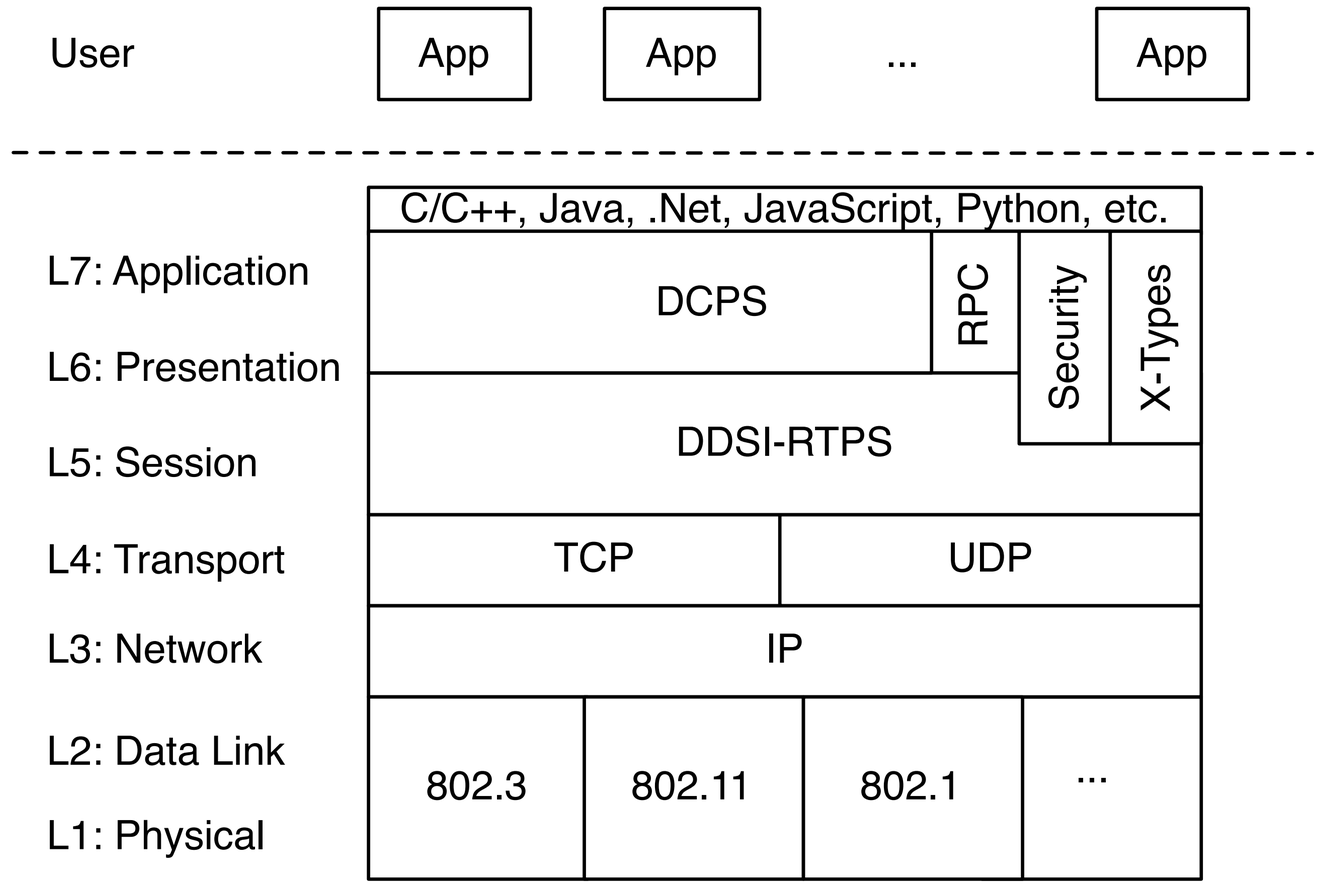
DDS STANDARD EVOLUTION



DDS STANDARDS

Data Centric Publish Subscribe (DCPS)

Defines a high level API for programming language, OS and architecture independent data sharing

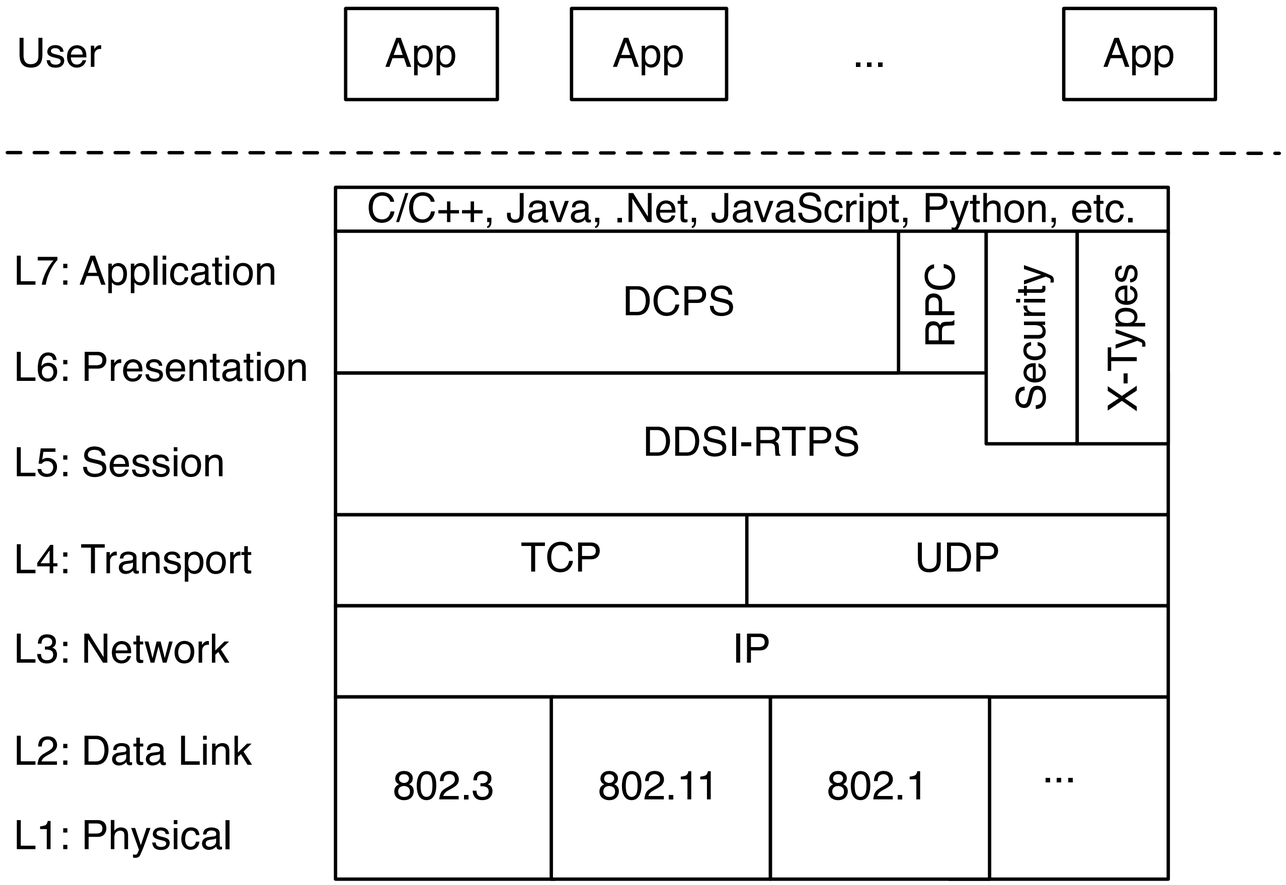


DDS STANDARDS

DDS Interoperability Protocol (DDSI-RTPS)

Defines a wire protocol for interoperable implementation of DCPS abstractions.

This protocol assumes a best-effort transport layer, i.e., reliability is provided by DDSI.

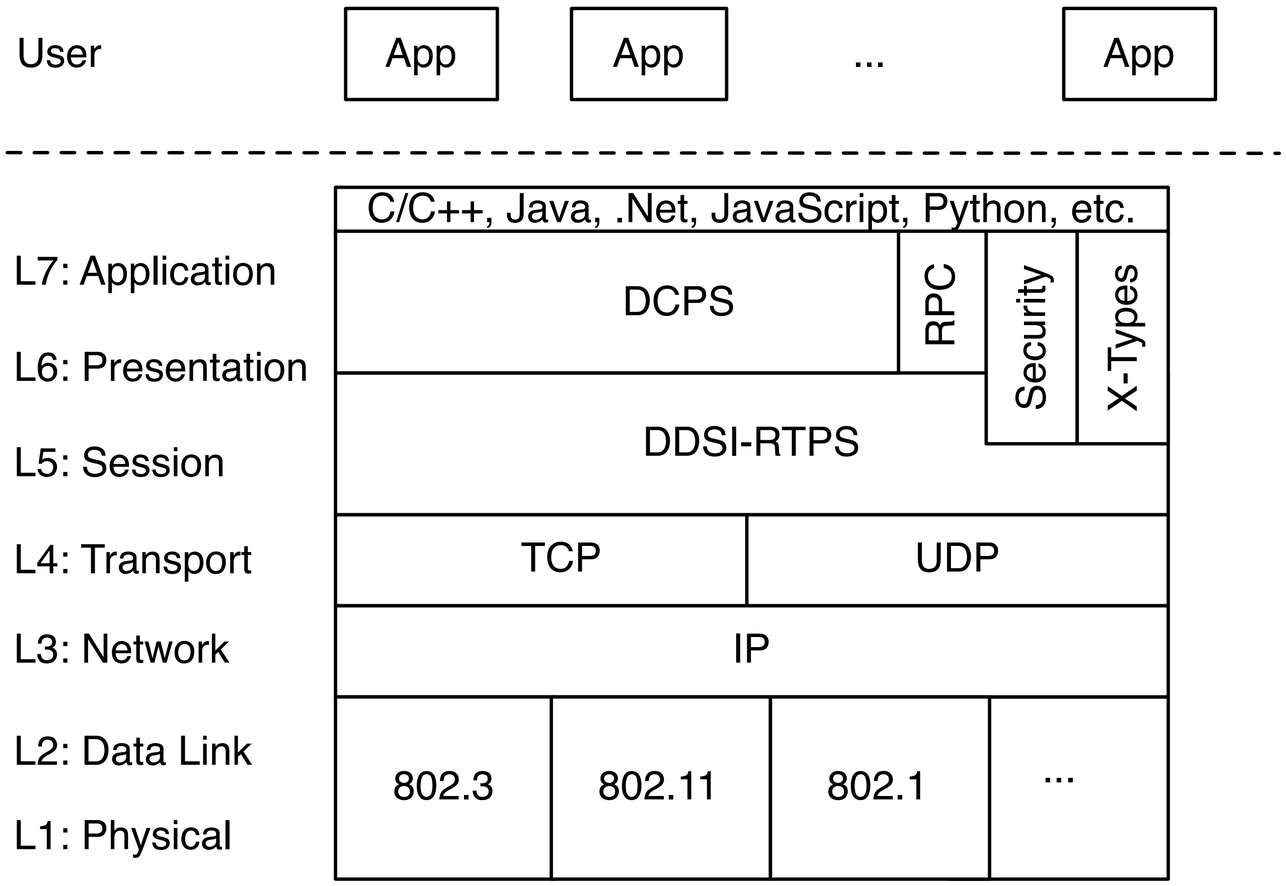


DDS STANDARDS

eXtensible Types (DDS-XTypes)

Extends the DDS type system from nominal to structural, thus providing very good support for evolutions and forward compatibility.

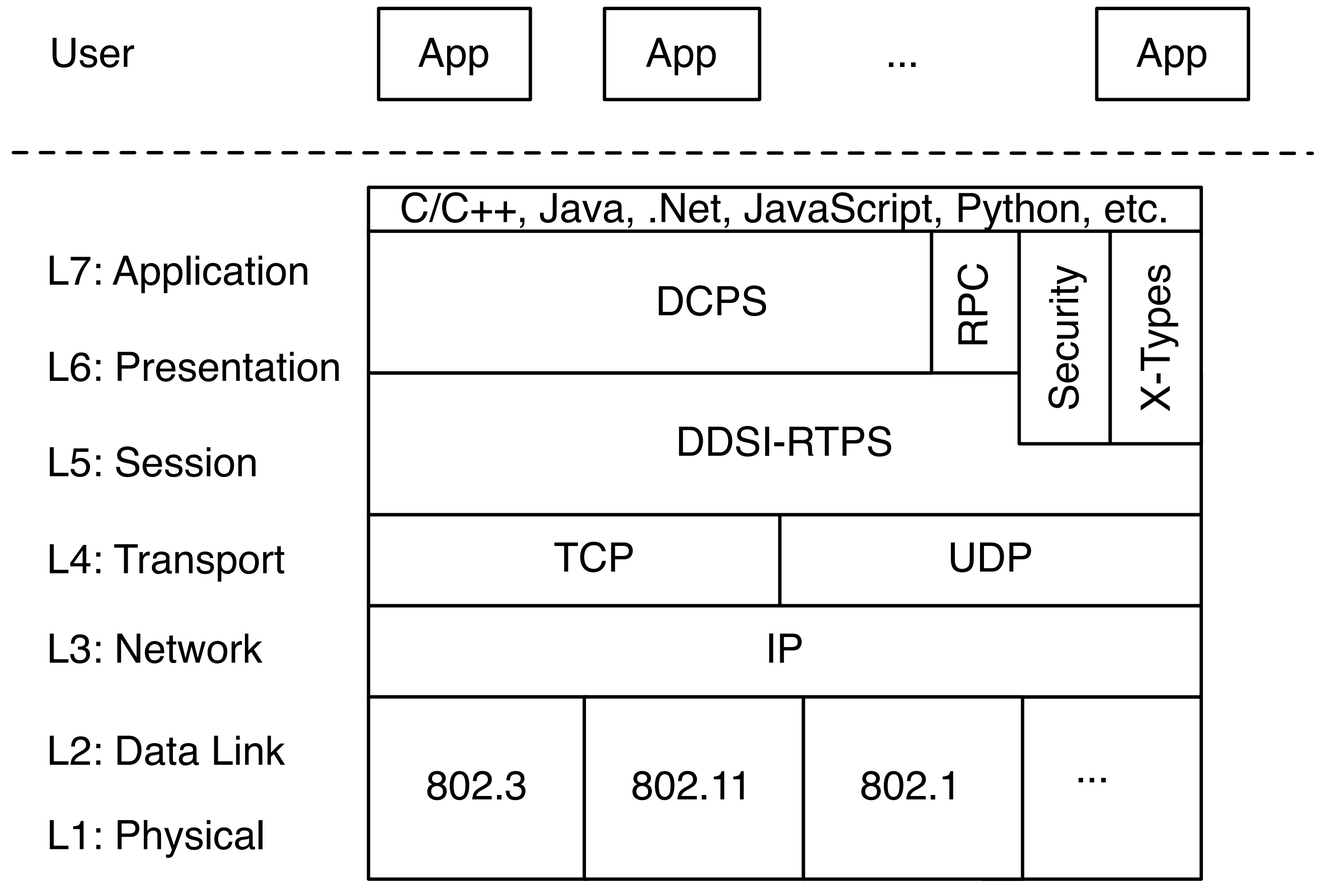
Defines APIs for dynamically defining and operating over DDS types



DDS STANDARDS

Security (DDS-Security)

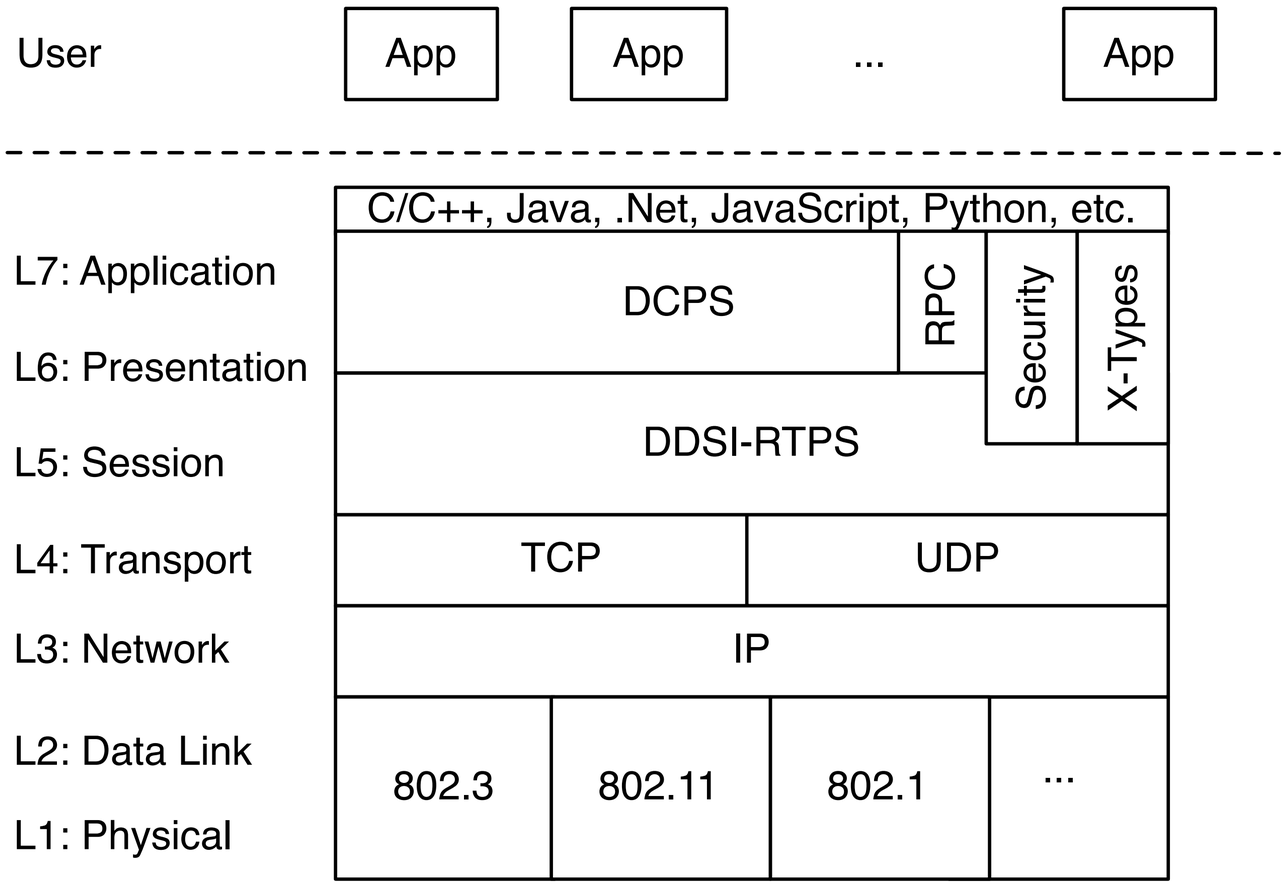
Defines a data-centric security architecture with pluggable Authentication, Access Control, Crypto and Logging.



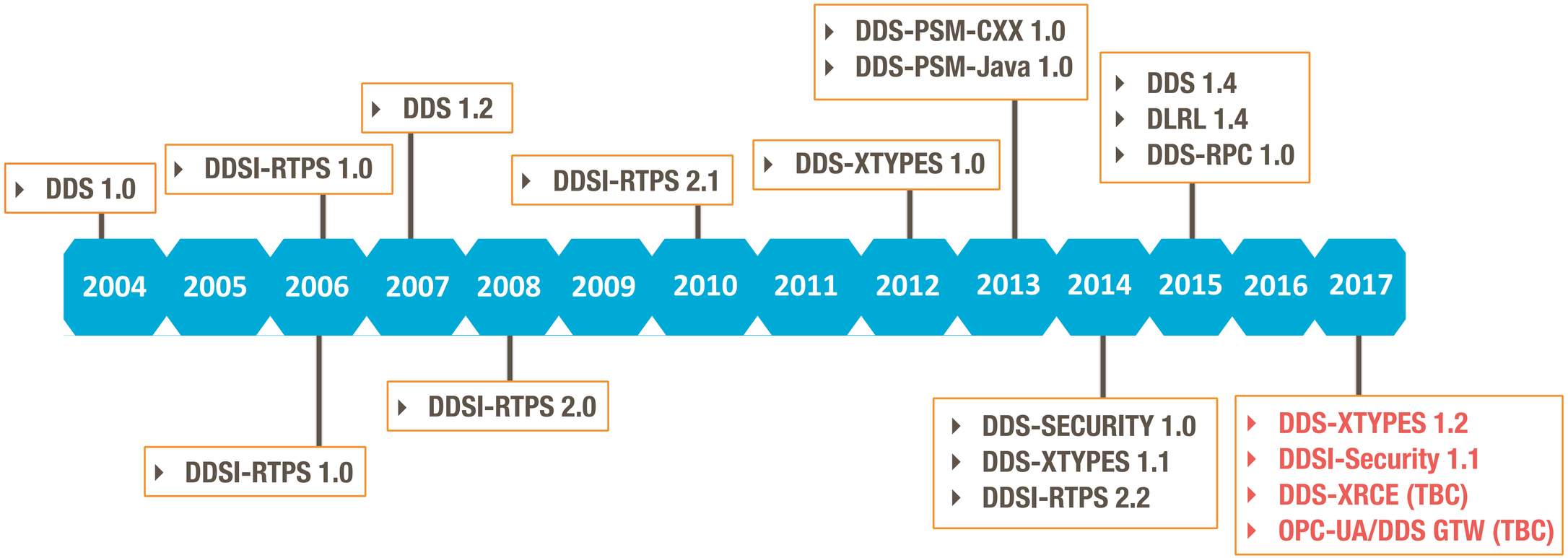
DDS STANDARDS

Remote Procedure Calls (DDS-RPC)

Extends DDS abstractions to support distributed service definition and remote operation invocations.



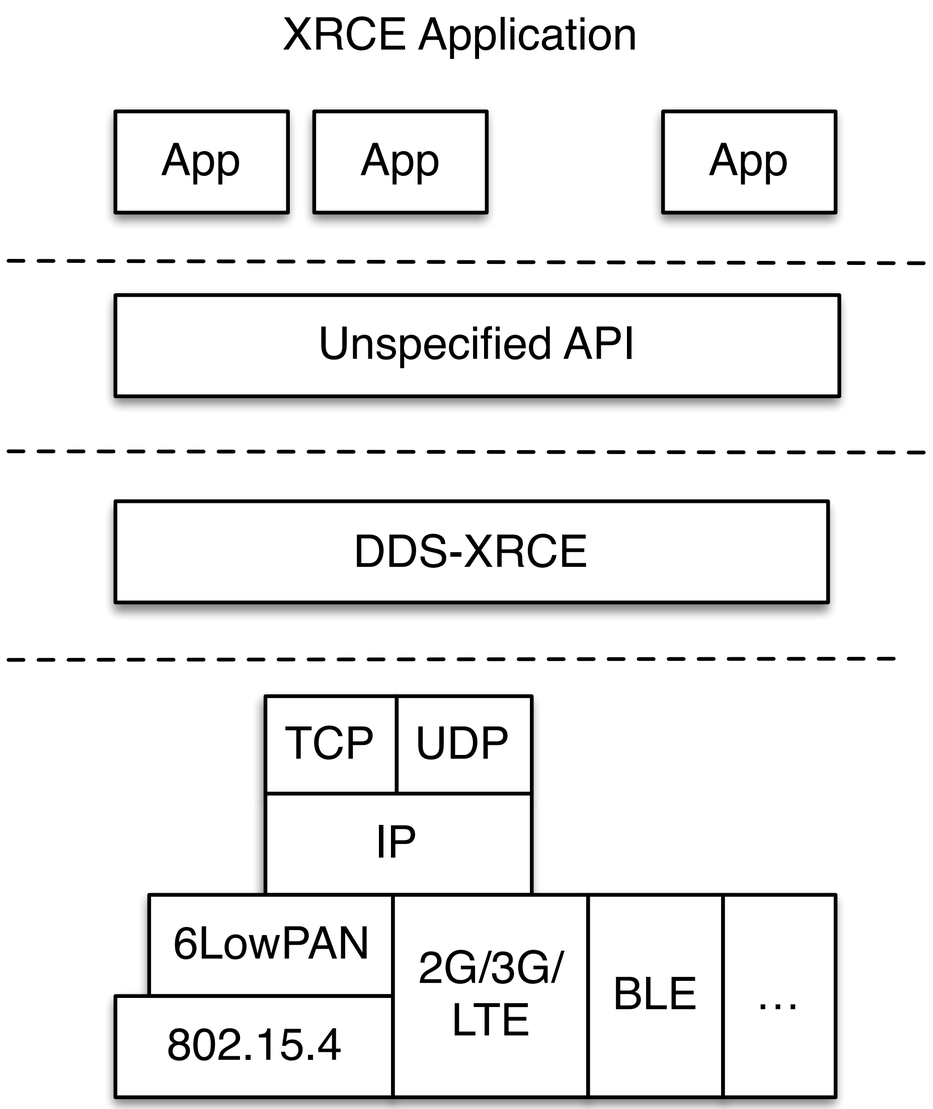
UPCOMING STANDARDS



DDS STANDARDS

eXtremely Resource Constrained Environments DDS (DDS-XRCE)

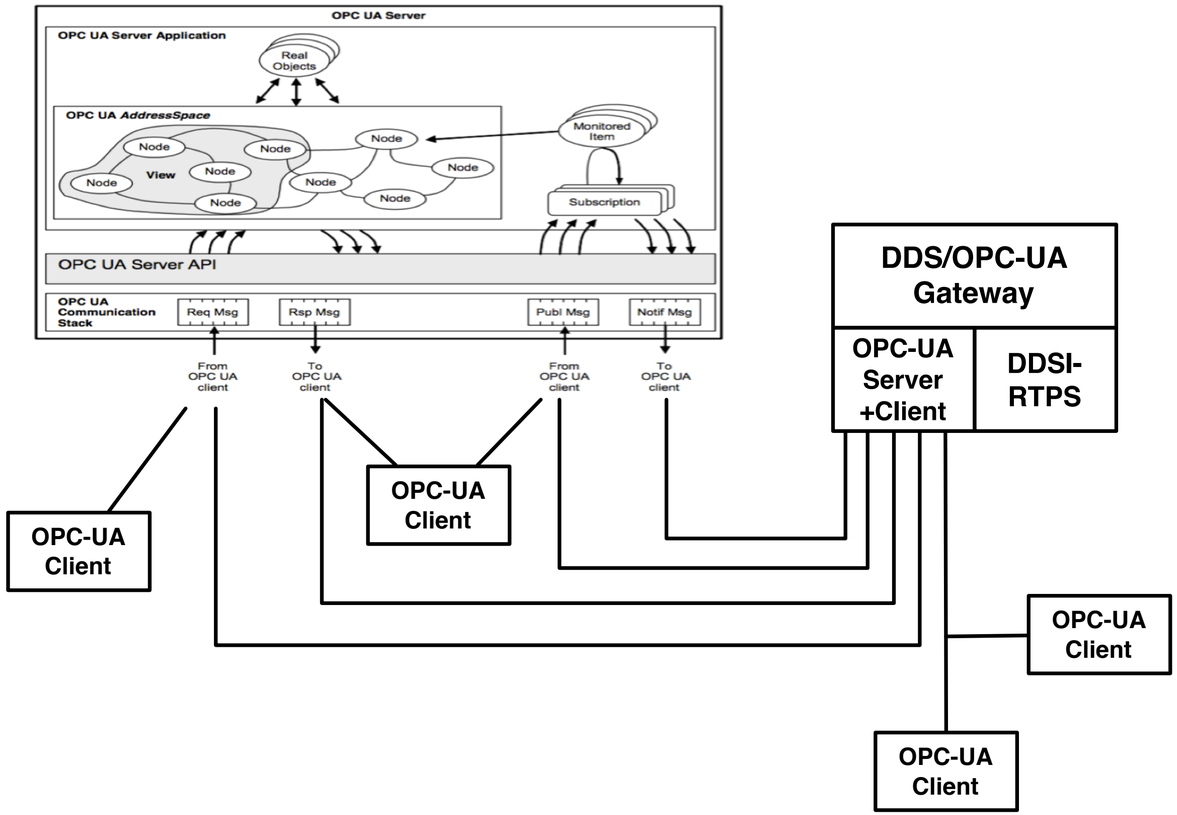
Defines the most wire/power/memory efficient protocol in the market to provide DDS connectivity to extremely constrained targets, such as battery powered devices with less than 100 Kbytes of memory and connected through constrained networks



DDS STANDARDS

OPC-UA/DDS Gateway (DDS-OPCUA-GTW)

Defines a standard Gateway to simplify the bridging between from OPC-UA and DDS



WHO IS USING IT?

EXTREMELY LARGE TELESCOPE (ELT)

DDS used to control the 100.000 mirrors that make up ELT's optics.

These mirrors are adjusted 100 times per second

NASA LAUNCH SYSTEMS

The NASA Kennedy Space Centre uses Vortex to collect the Shuttle Launch System Telemetry.

Vortex streams over 400.000 Msgs/sec

AUTONOMOUS VEHICLES

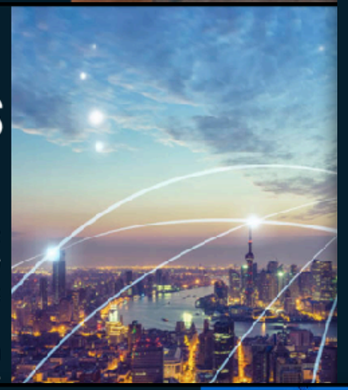
Vortex is used for data sharing within and across the vehicles.

he environment is highly heterogeneous and

SMART CITIES

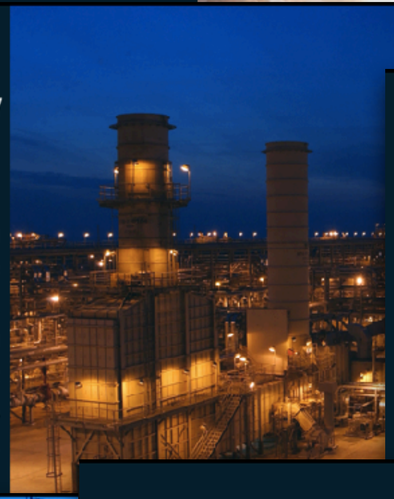
Vortex is used as the integration technology for data sources and sinks

Vortex is also used as a control plane for equipment



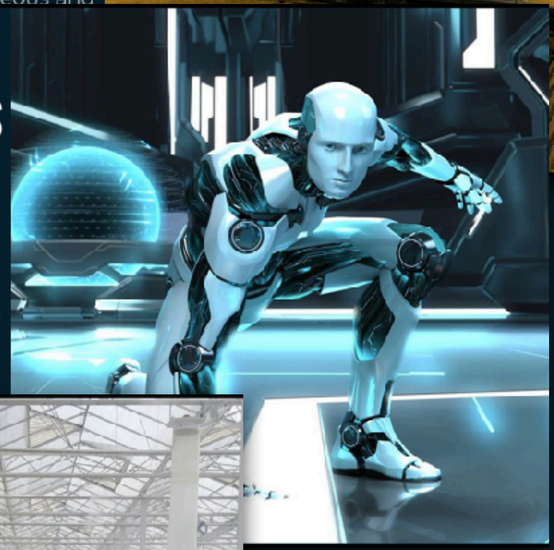
SMART FACTORY

Vortex is used in Smart Factories to provide horizontal and vertical data integration across the traditional SCADA layers.



ROBOTICS

Vortex is heavily used for data sharing in Robotics and is today at the heart of the Robot Operating



SMART GRID

Vortex is used to integrate and normalise data sharing among the various elements of a smart grid at scale

Duke's Energy COW showed how only with Vortex it was possible to distribute the phase alignment signal at scale wit hthe required 20ms periodicity

SMART GREEN HOUSES

Vortex is used to virtualise I/O and provide better decoupling between I/O, Control and Management functions of the system

WHY ARE THESE APPLICATIONS USING DDS?

DDS PROVIDES AN EXTREMELY HIGH LEVEL AND POWERFUL ABSTRACTIONS ALONG WITH A ROCK SOLID INFRASTRUCTURE TO BUILD HIGHLY MODULAR AND DISTRIBUTED SYSTEMS

DDS MAKES IT MUCH EASIER TO SOLVE SOME VERY HARD DISTRIBUTED SYSTEM PROBLEMS, SUCH AS FAULT-TOLERANCE, SCALABILITY AND ASYMMETRY

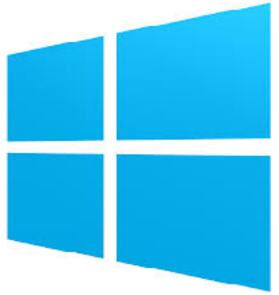
DDS IS LIKE A UNIVERSAL GLUE THAT ALLOWS TO SEAL TOGETHER HIGHLY HETEROGENEOUS ENVIRONMENTS WHILE MAINTAINING A SINGLE, ELEGANT AND EFFICIENT ABSTRACTION

PLATFORM INDEPENDENT

DDS implementation are available for an incredible numbers of platforms, including enterprise, desktop, embedded, real-time, mobile, and web

OS X

iOS



VxWorks®

WIND RIVER

POLYGLOT

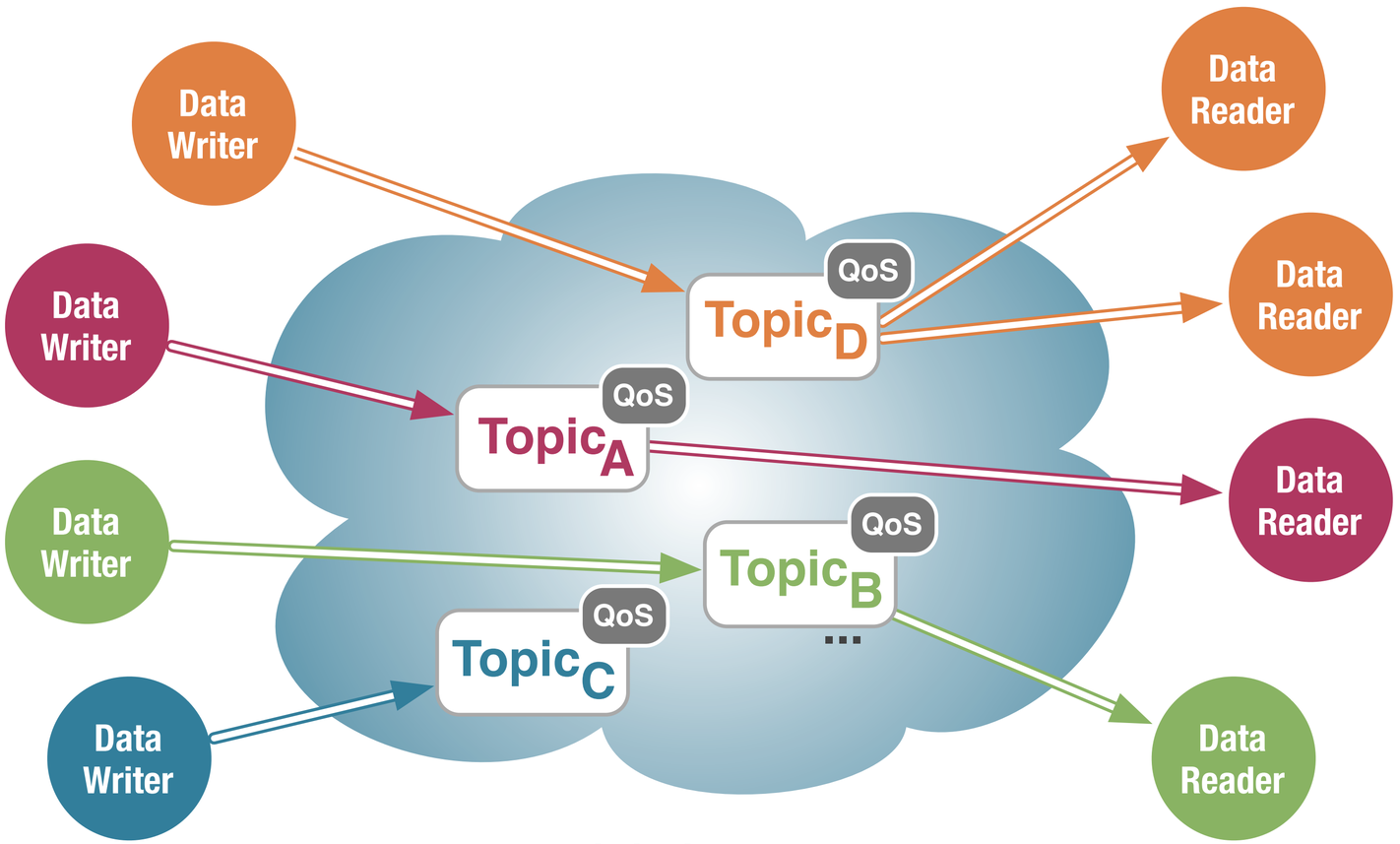
DDS applications can be written in your favourite programming language.

Interoperability across languages is taken care by DDS

DDS: THE MAIN IDEA

VIRTUALISED DATA SPACE

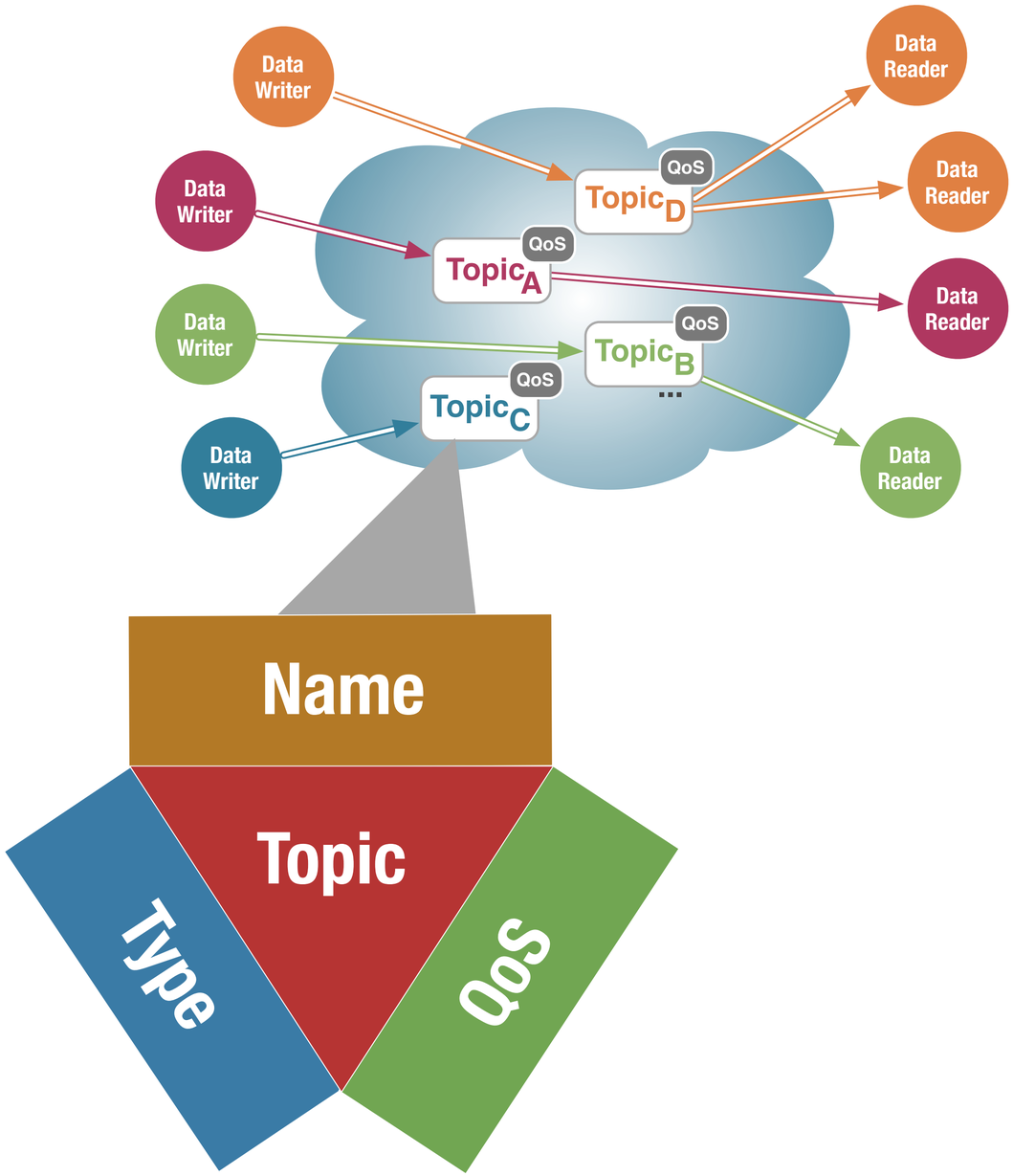
Applications can **autonomously** and **asynchronously read** and **write** data enjoying **spatial** and **temporal decoupling**



DDS Global Data Space

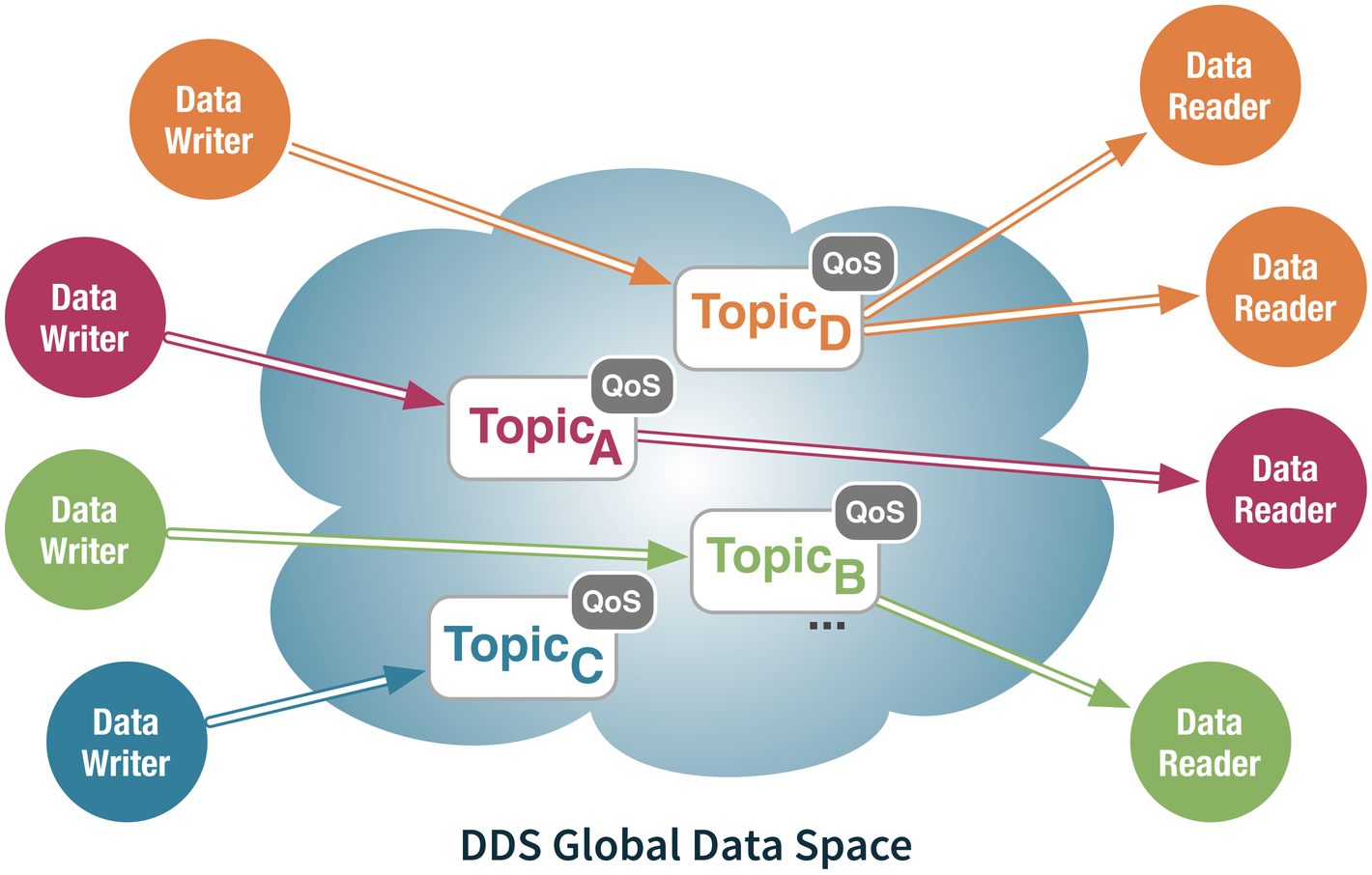
TOPIC

A domain-wide information's class A **Topic** defined by means of a <name, type, qos>



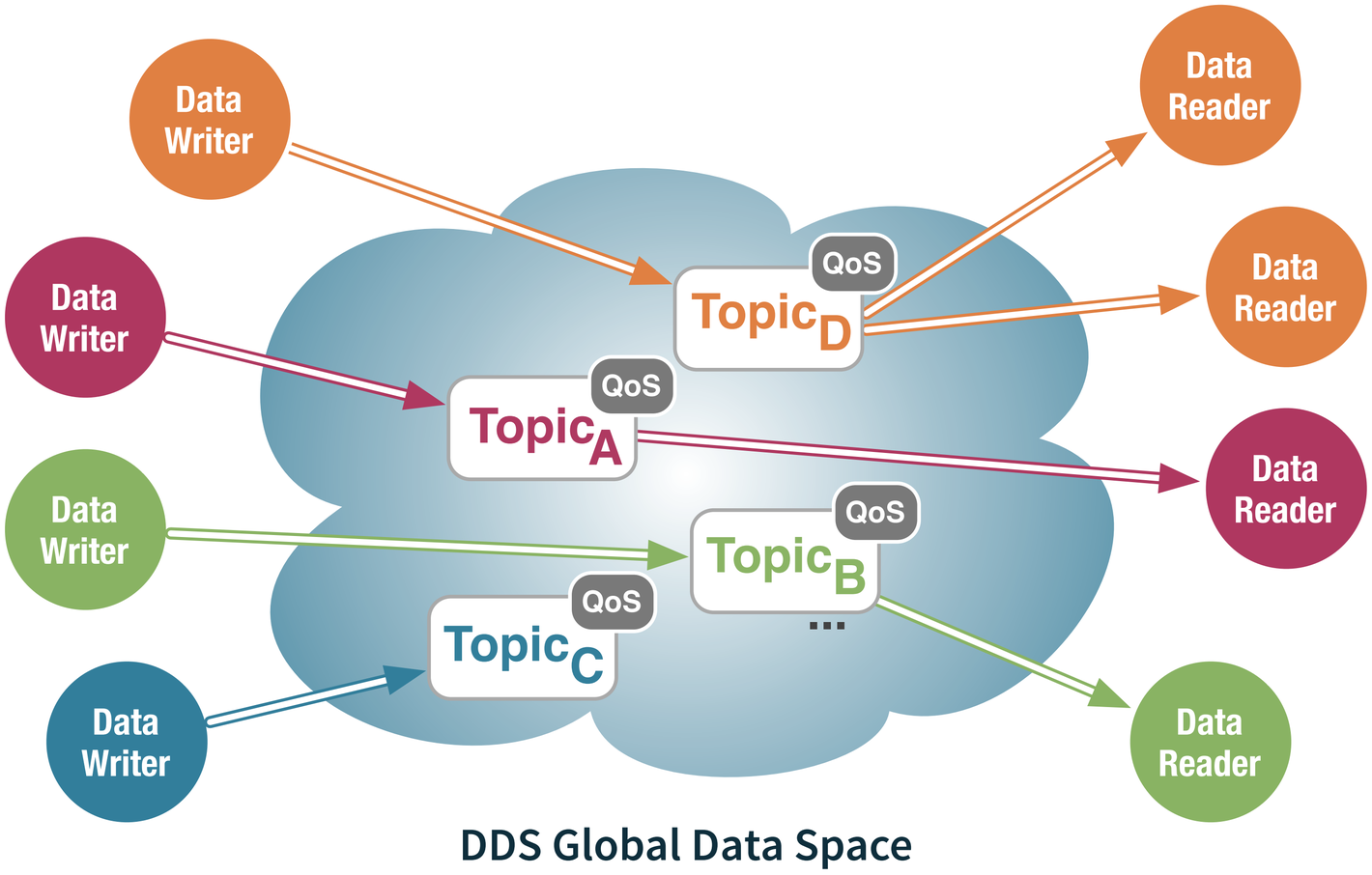
QOS ENABLED

QoS policies allow to express **temporal** and **availability constraints** for data



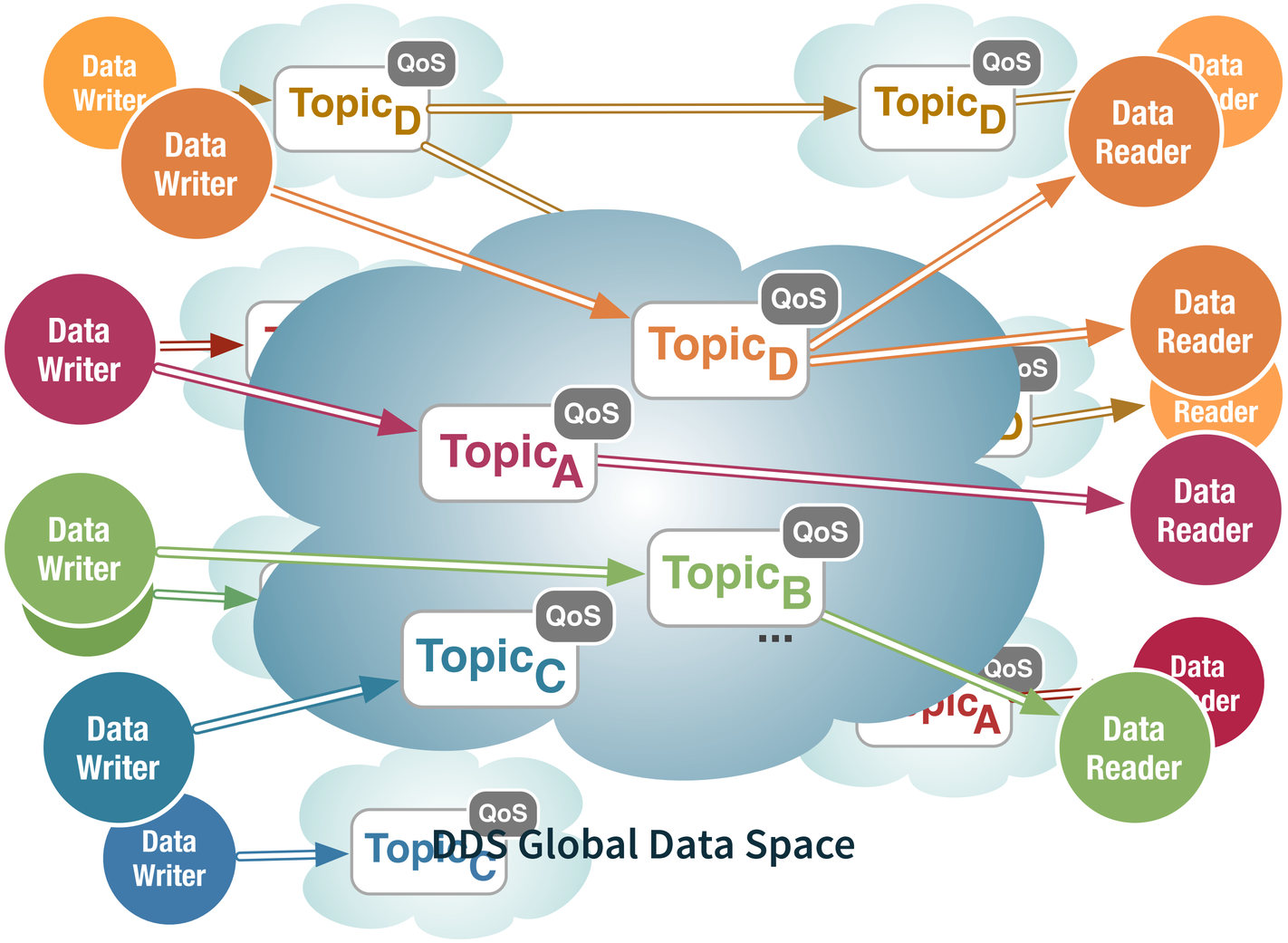
DYNAMIC DISCOVERY

Built-in dynamic discovery **isolates applications from network topology** and **connectivity** details



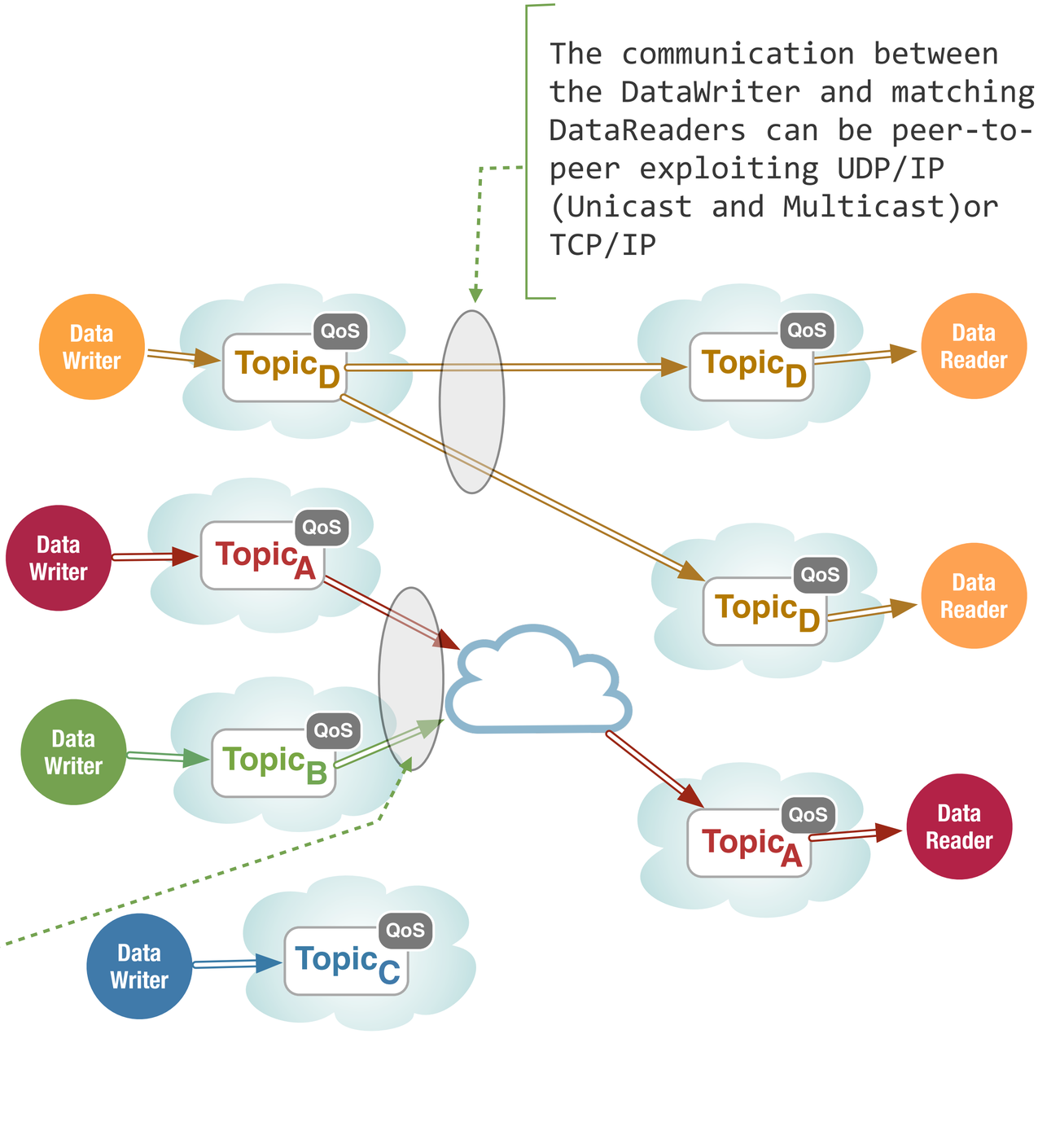
DECENTRALISED DATA-SPACE

No single point of failure
or bottleneck



ADAPTIVE CONNECTIVITY

Connectivity is **dynamically adapted** to chose the most effective way of sharing data



MOST FREQUENTLY HEARD MISCONCEPTIONS

MISCONCEPTION #1

DDS CAN ONLY WORK ON A LAN AS IT REQUIRES MULTICAST

RECTIFICATION #1

The only one thing that DDS assumes from the underlying transport is best effort one-to-one communication.

DDS can and will exploit multicast when available, but does not require it in order to work!

MISCONCEPTION #2

USE DDS ONLY IF YOU NEED REAL-TIME

RECTIFICATION #2

While DDS support for predictable, real-time, communication is unparalleled, in a sense this is just a tiny detail.

DDS main value is simplifying the design, development, deployment and maintenance of distributed systems

MISCONCEPTION #3

DDS IS NOT WIDELY ADOPTED

RECTIFICATION #3

DDS is not widely hyped as its main domain of applications has been thus far the mission and business critical applications as opposed to Internet applications.

That said, it has far more deployments than many of the technologies currently hyped in IoT/IIoT.

MISCONCEPTION #4

THERE ARE NO OPEN SOURCE DDS IMPLEMENTATIONS

RECTIFICATION #4

There are at least two mainstream Open Source implementations of DDS. One of them, from PrismTech, is OpenSplice DDS

MISCONCEPTION #5

DDS CANNOT GO THROUGH FIREWALL OR NATS NOR IT CAN SUPPORT TO INTERNET SCALE APPLICATIONS

RECTIFICATION #5

DDS vendors such as PrismTech have shown not only how DDS can be used to build Internet Scale systems but more importantly how some of the DDS features make it easier!

DECOMPOSING DDS

INFORMATION ORGANISATION

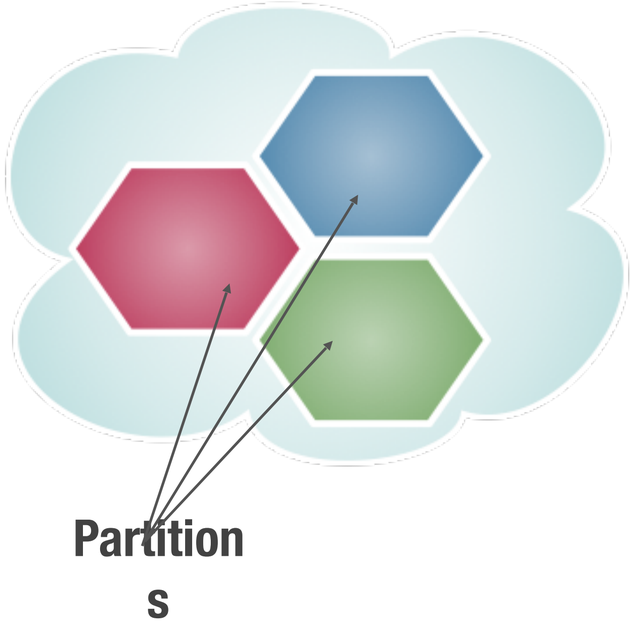
DOMAIN

- DDS data lives within a **domain**
- A **domain** is identified with a non negative integer, such as 1, 3, 31
- The number **0** identifies the default domain
- A domain represent an impassable communication plane

DDS Domain

PARTITIONS

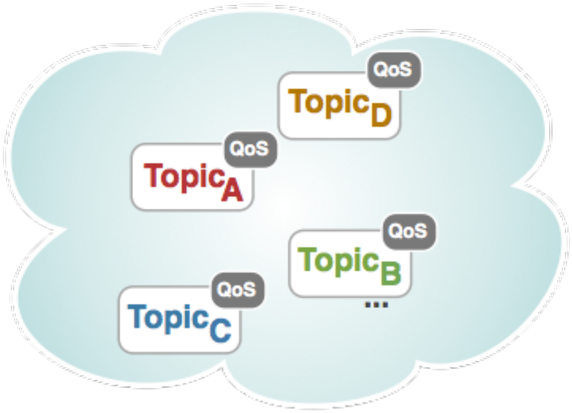
- Partitions are the mechanism provided by DDS to organise information within a domain
- Access to partitions is controlled through QoS Policies
- Partitions are defined as strings:
  - `“system:telemetry”`
  - `“system:log”`
  - `“data:row-2:col-3”`
- Partitions addressed by name or regular expressions:
  - `”system:telemetry”`
  - `“data:row-2:col-*”`

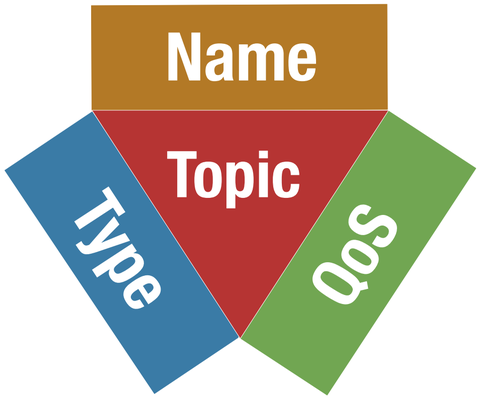


INFORMATION DEFINITION

TOPIC

- A Topic defines a **domain-wide** information's class
- A **Topic** is defined by means of a (name, type, qos) tuple, where
  - **name:** identifies the topic within the domain
  - **type:** is the programming language type associated with the topic. Types are extensible and evolvable
  - **qos**: is a collection of policies that express the non-functional properties of this topic, e.g. reliability, persistence, etc.





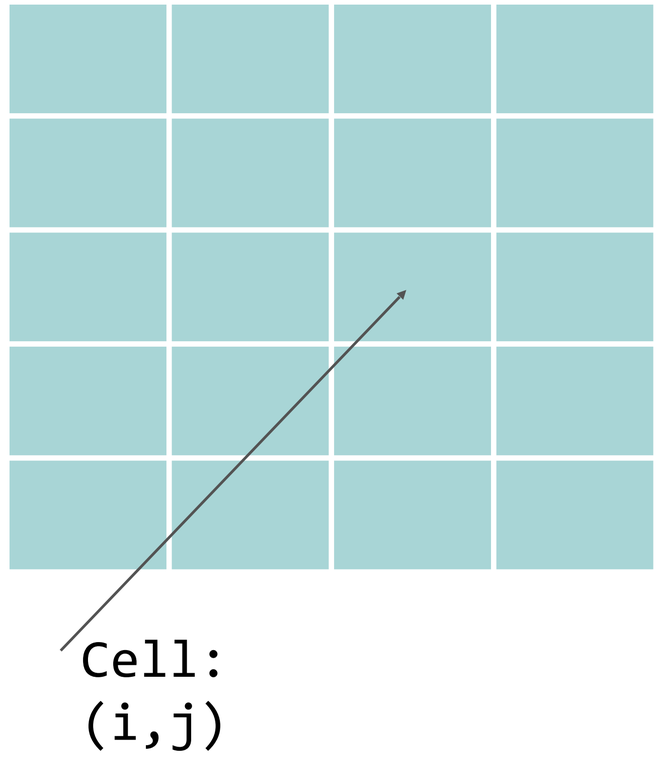
TOPIC AND INSTANCES

- As explained in the previous slide **a topic defines a class/type of information**
- Topics can be defined as **Singleton** or can have multiple **Instances**
- **Topic Instances** are identified by means of the topic key
- A Topic **Key** is identified by a **tuple of attributes** -- like in databases
- Remarks:
  - A **Singleton** topic has a single domain-wide instance
  - A “regular” Topic can have as many instances as the number of different key values, *e.g., if the key is an 8-bit character then the topic can have 256 different instances*

EXAMPLE

ACTIVE FLOOR

- Assume we are building an active floor
- This active floor is made by a matrix of pressure sensors used to detects position, and indirectly movement
- This information is leveraged by the application that uses the active floor for positioning or entertainment

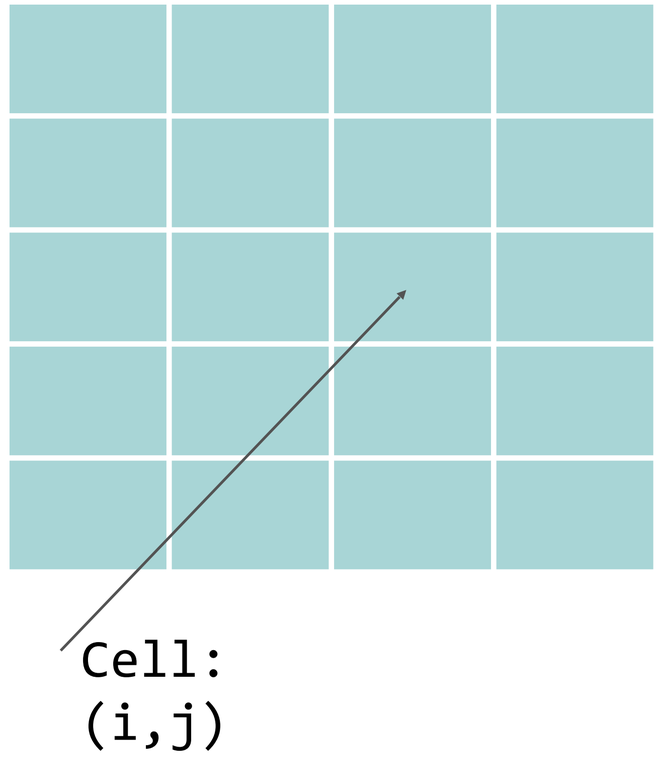


ACTIVE FLOOR

- The generic active cell can be modelled with a topic that has an instance for each value of (i,j). The topic type can be defined as:

```
struct TCell {
   short row;
   short column;
   float pressure; // in kPa
};
#pragma keylist TCell row column
```

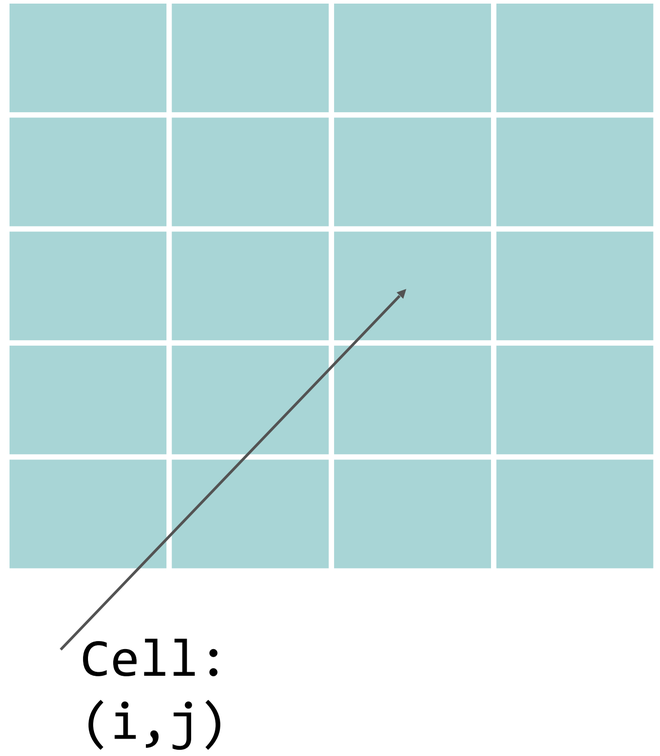
- Each cell is now distinguishable and associated with a topic instance



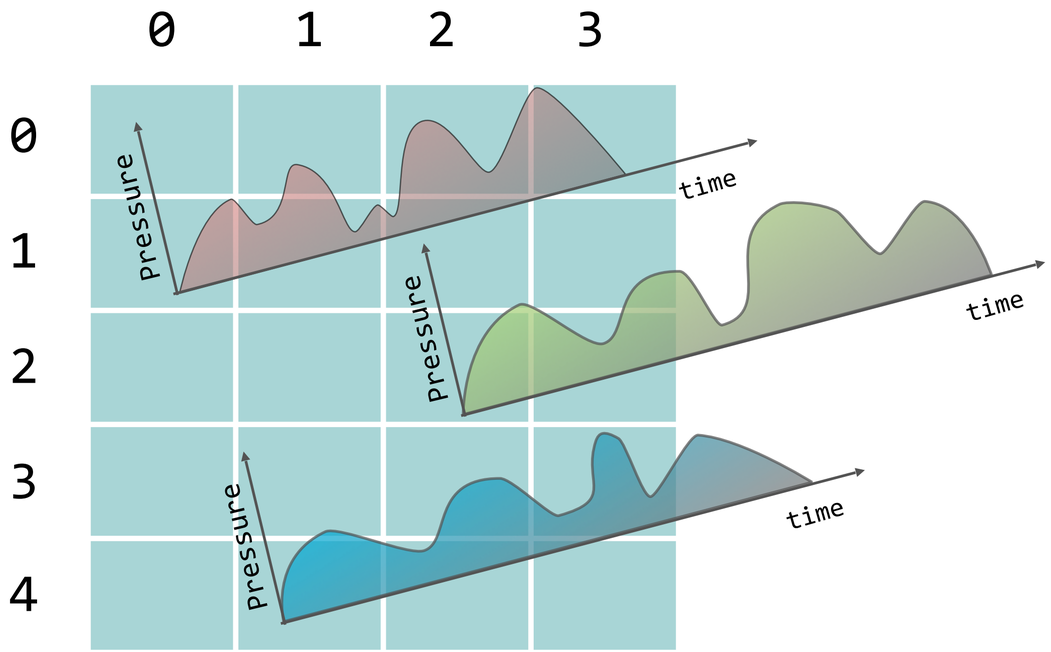
ACTIVE FLOOR

- How can we know when something is on the cell?
- The detection can be based on the difference between the atmospheric pressure, say P0, and the pressure sensed by the cell
- We can model this as a Singleton Topic ReferencePressure defined by the type:

```
struct TReferencePressure {
   float pressure; // in kPa
   float precision;
};
#pragma keylist TReferencePressure
```



ACTIVE FLOOR

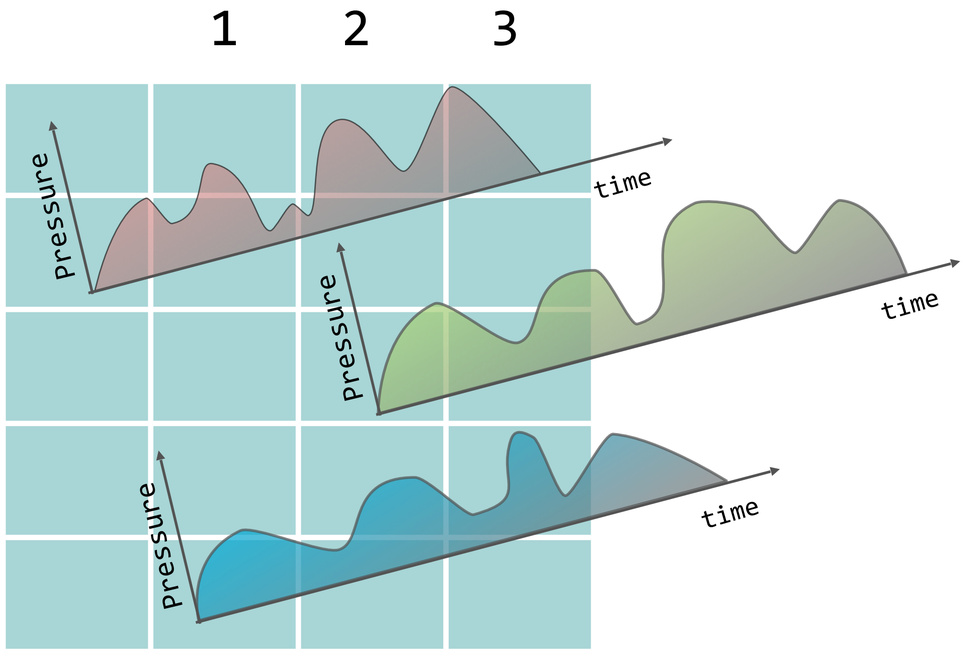


- Each sensor has associated a topic instance identified by the (row,column) coordinate -- the instance key

```
struct TCell {
   short row;
   short column;
   float pressure; // in kPa
};
#pragma keylist Cell row column
```

- Each instance produces a stream of pressure values that in DDS terms are called samples

EXERCISE



- What if we want to extend our model to deal with floors on different levels?
- How would you extend the data model?
- How would you use partitions?

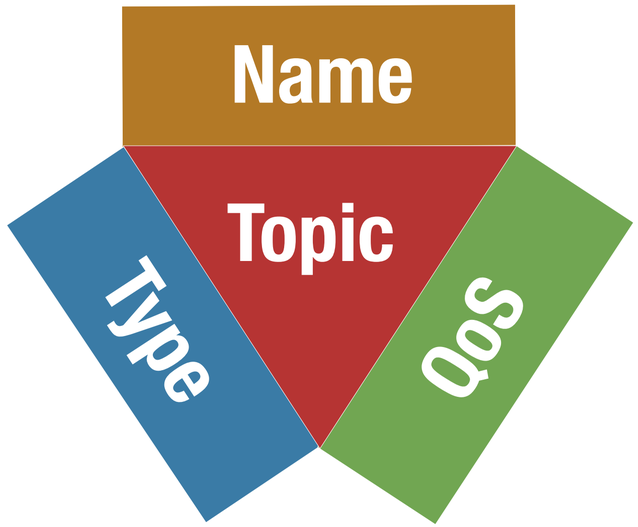
PRODUCING INFORMATION

DATA WRITER

- A **DataWriter (DW)** is a strongly typed entity used to produce **samples** for one or more **instances** of a Topic, with a given **QoS**

- Conceptually, the DataWriter QoS should be the same as the Topic QoS or more stringent

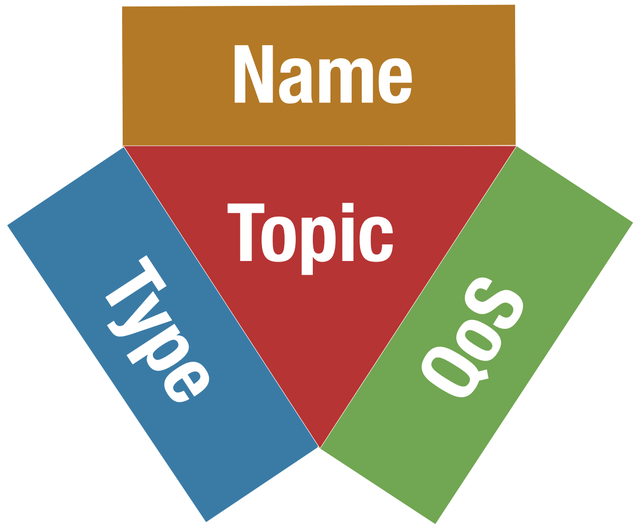
- However, DDS does enforce a specific relationship between the Topic and DataWriter QoS



DATA WRITER

The DataWriter controls the **life-cycle** of Topic Instances and allows to:

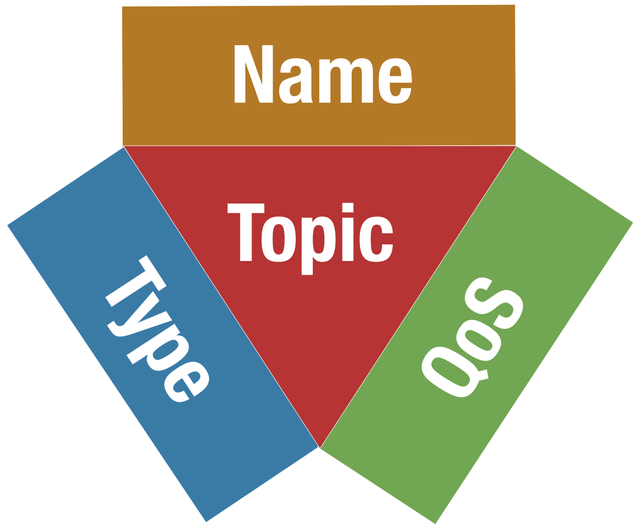
- Define a new topic instance
- Write samples for a topic instance
- Dispose the topic instance



DATA WRITER

The DataWriter controls the **life-cycle** of Topic Instances and allows to:

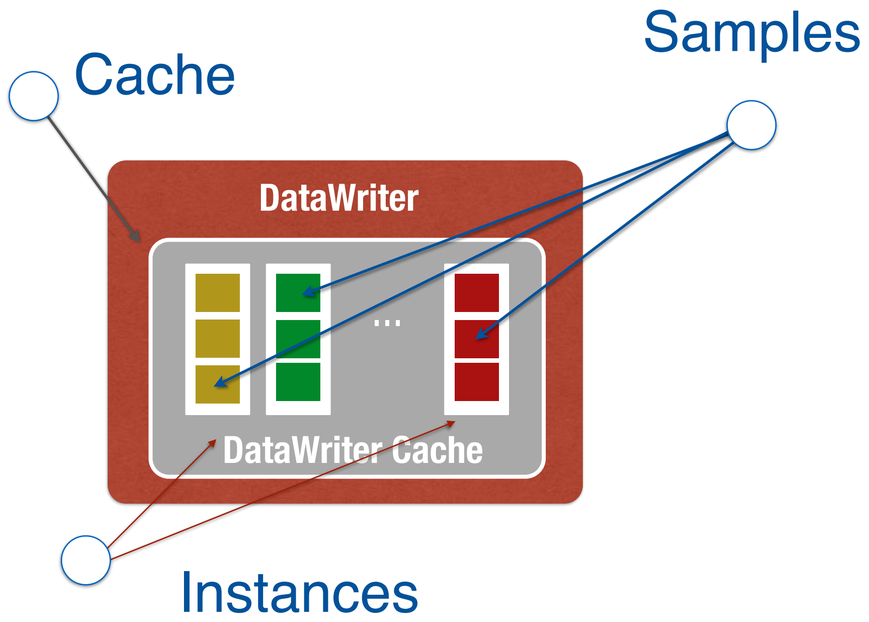
- Define a new topic instance
- Write samples for a topic instance
- Dispose the topic instance



WRITER CACHE

Each DDS DataWriter has an associated cache

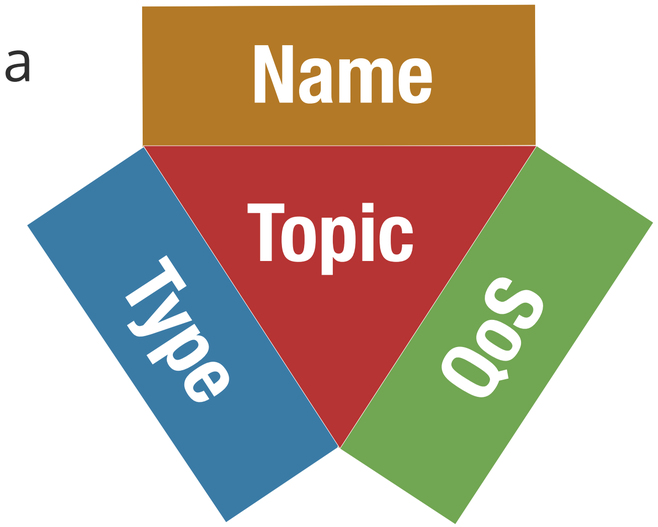
- Writes are always local to the cache
- This cache provides two degrees of temporal decoupling between writers and readers. One w.r.t. processing speed the other w.r.t. temporal coupling
- The writer cache along with DDS QoS Policies provides built-in support for the **Circuit-Breaker** pattern



CONSUMING INFORMATION

DATA READER

- A **DataReader** (DR) is a strongly typed entity used to **access** and/or **consume samples** for a Topic, with a given QoS
- Conceptually, the DataReader QoS should be the same as the Topic QoS or less stringent
- However, DDS does enforce a specific relationship between the Topic and DataReader QoS

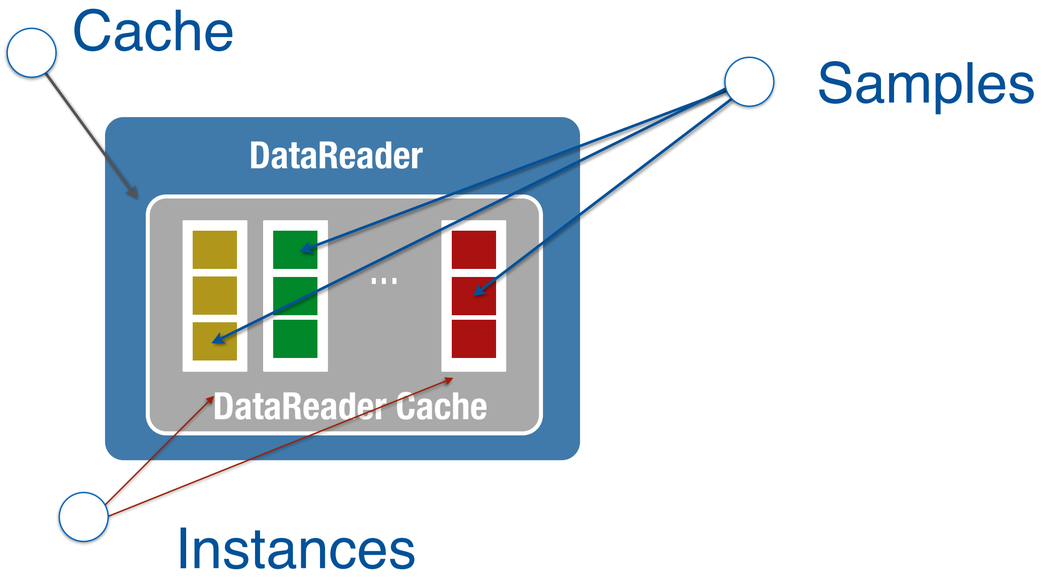


READER CACHE

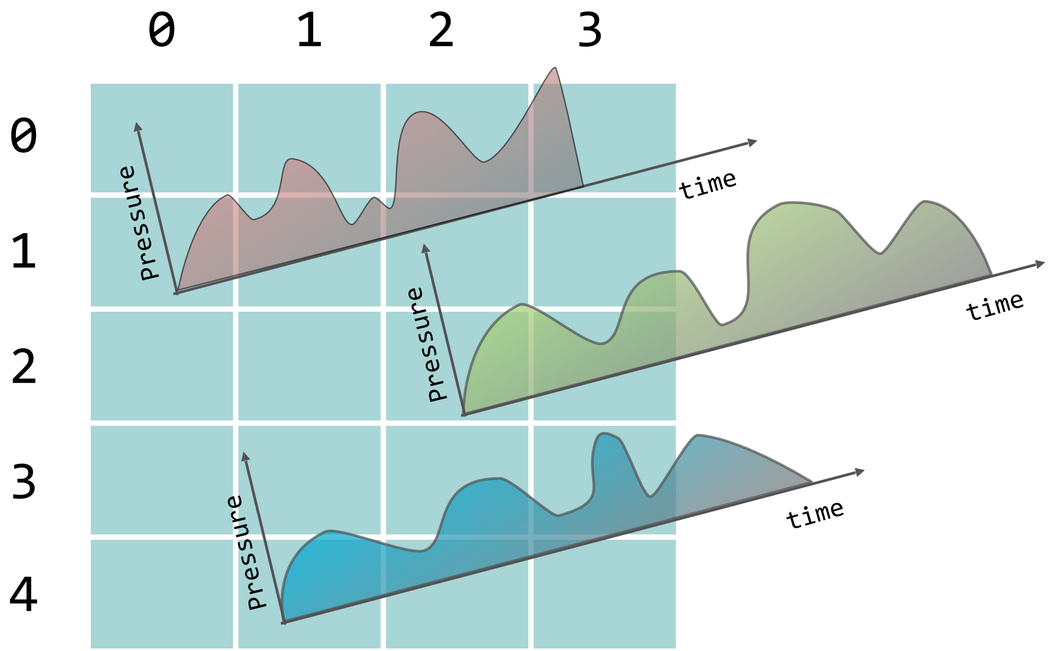
The Reader Cache stores the last $n \in \boldsymbol{N}\infty$ samples

for each relevant instance

Where: $\boldsymbol{N}^{\infty} = \boldsymbol{N} \cup \{\infty\}$

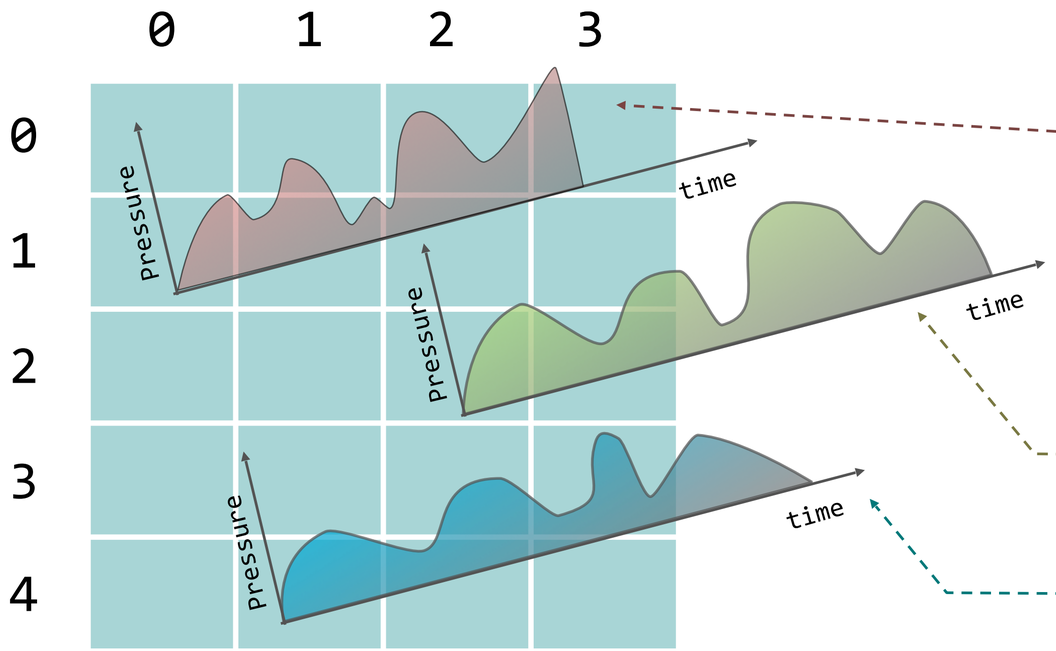


DATA READER



Depending on its QoS a DataReader may provide access to:

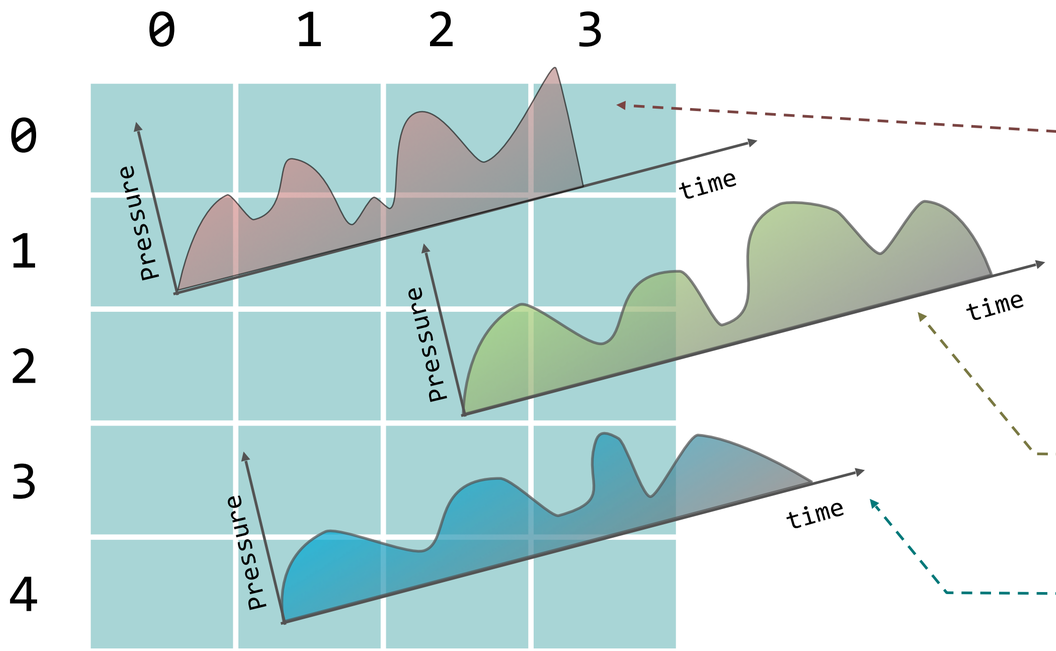
- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

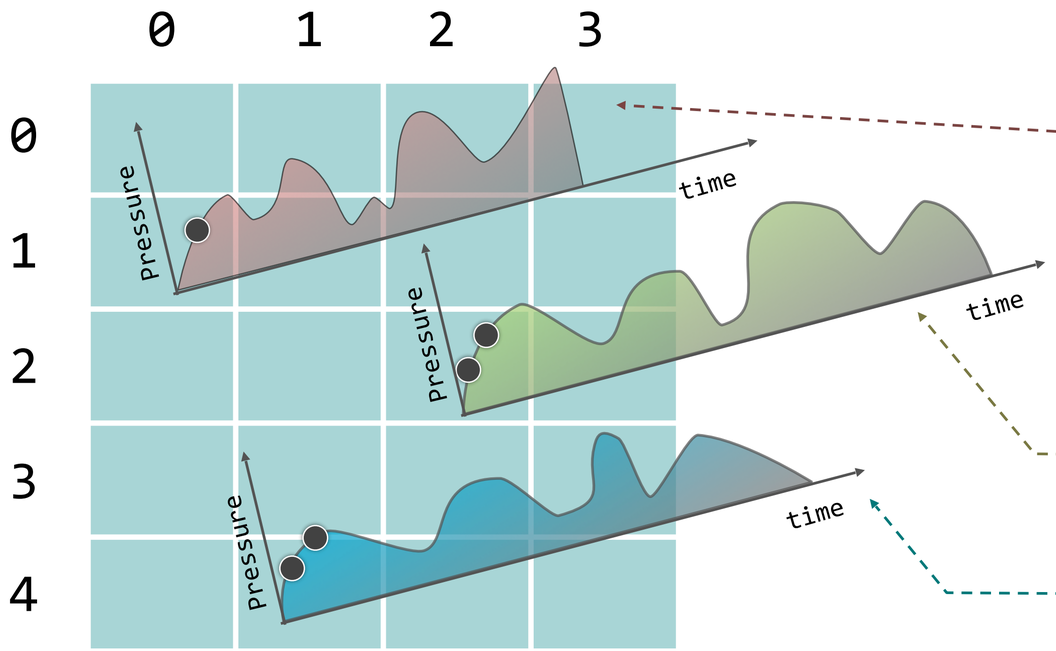
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

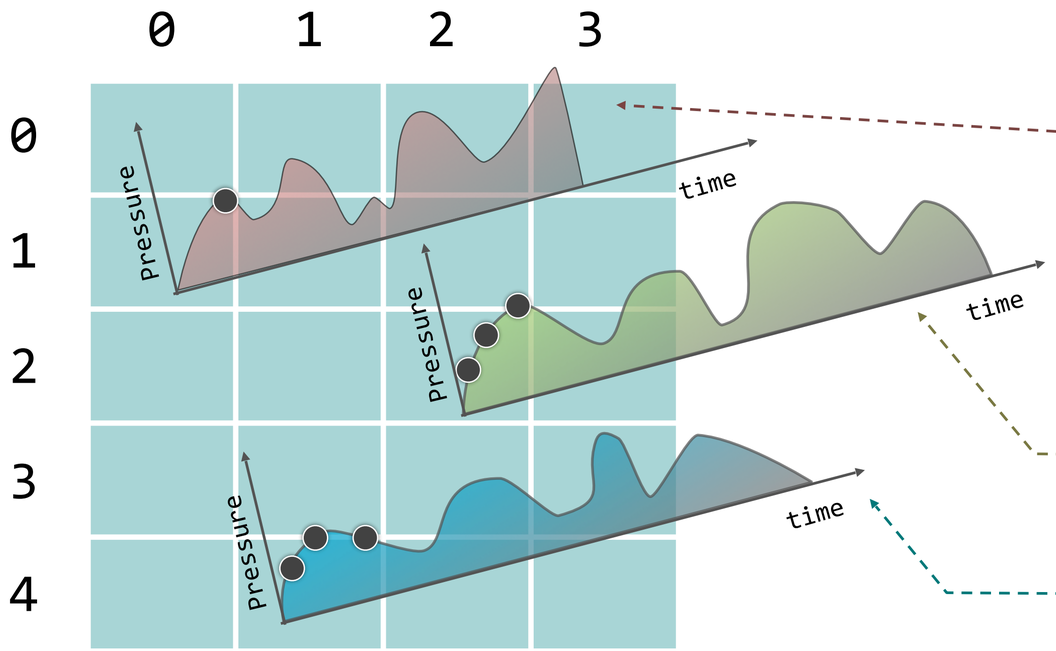
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

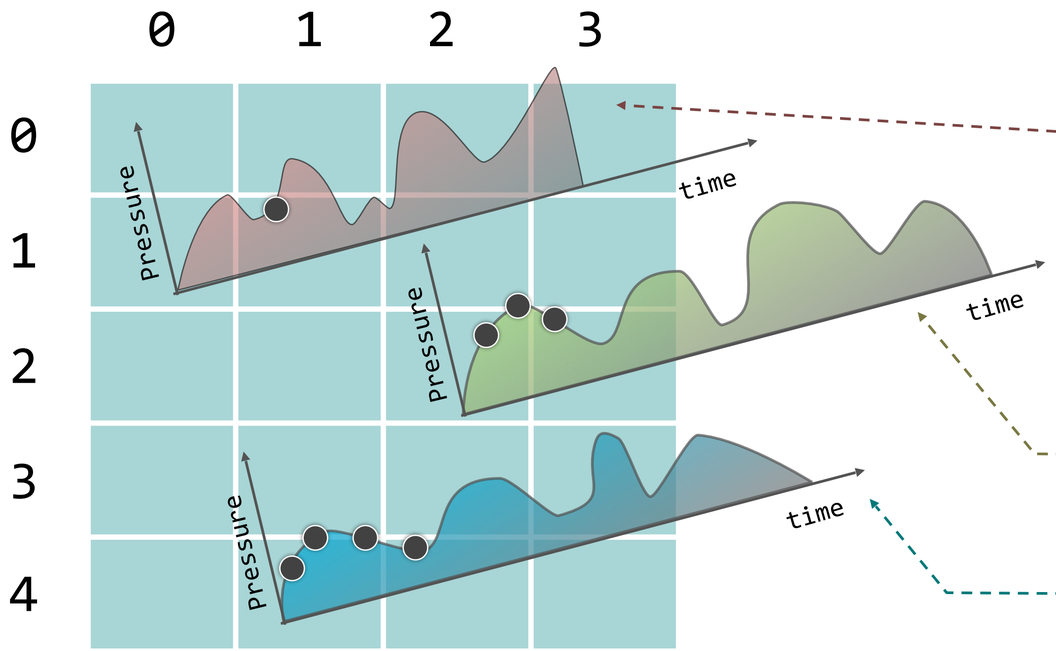
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

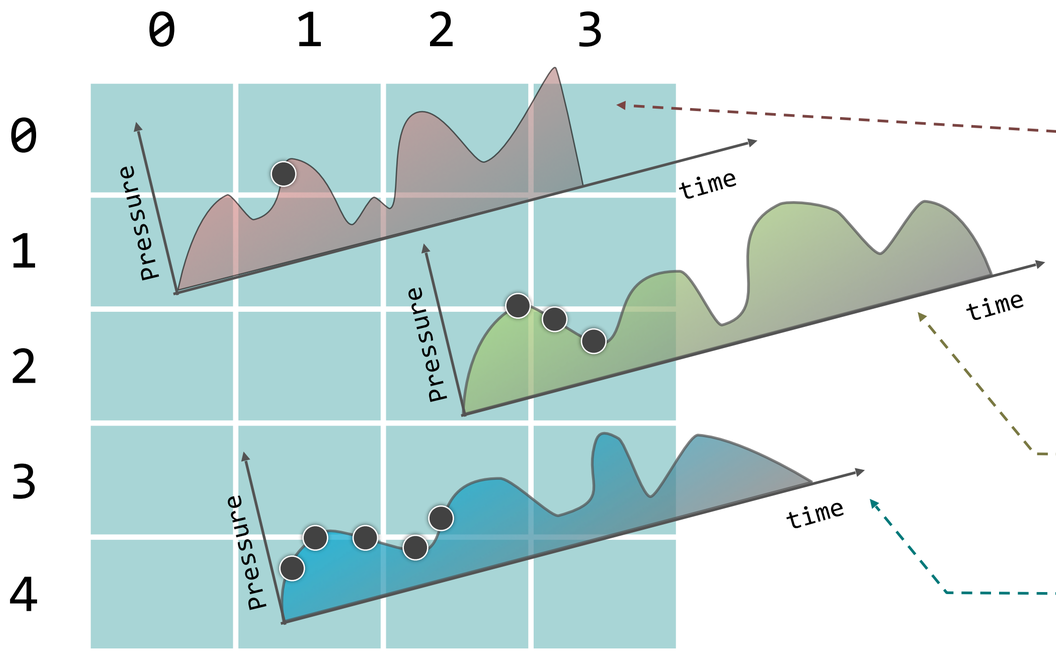
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

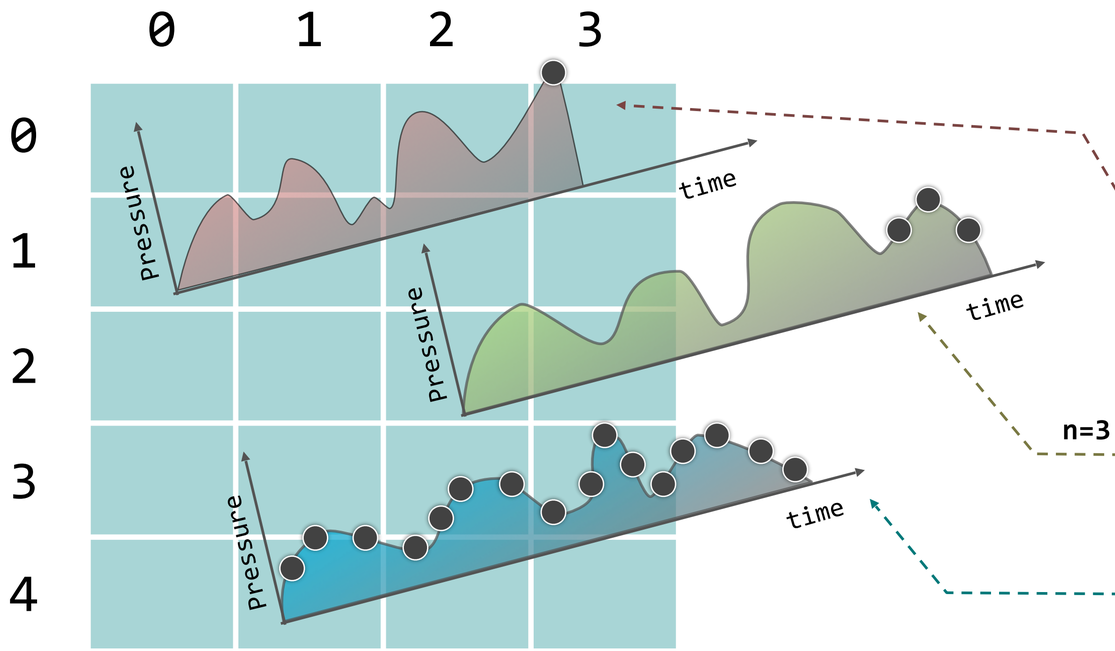
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

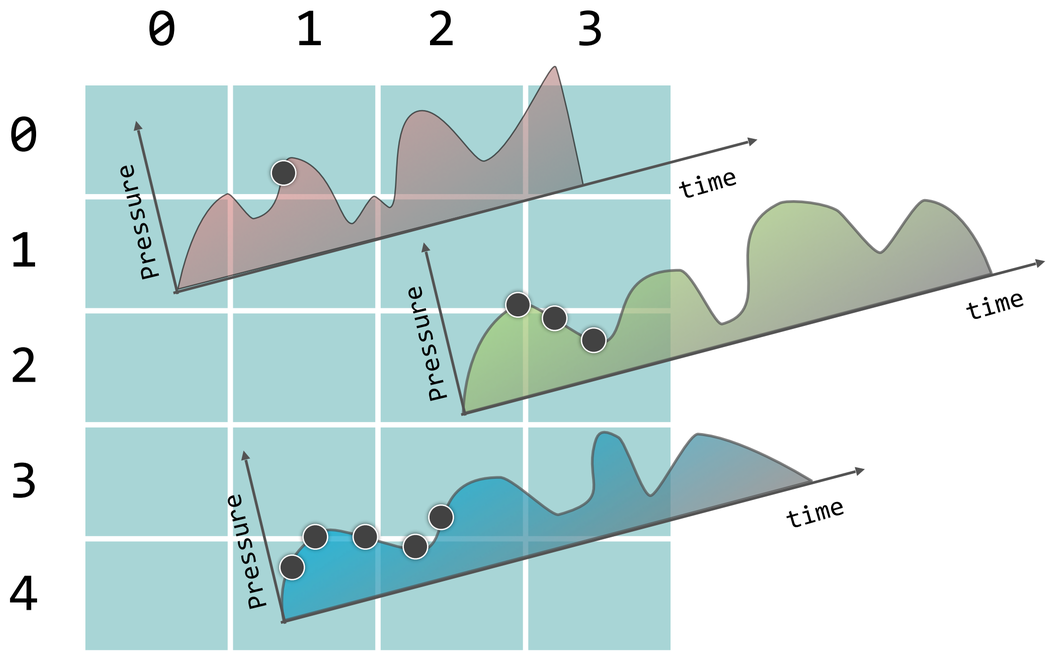
DATA READER



Depending on its QoS a DataReader may provide access to:

- **last** sample
- last **n** samples
- **all** samples produced since the DataReader was created

DATA READER

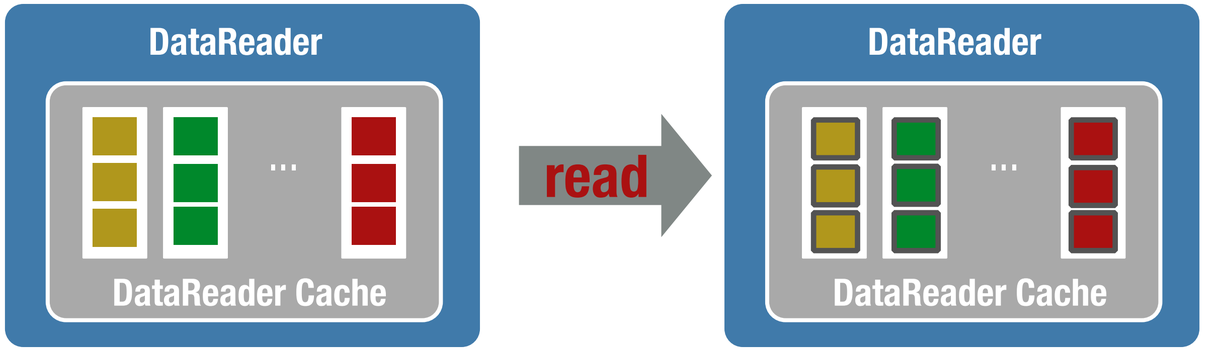


- Samples are stored in the DataReader Cache
- Samples can be read or taken from the cache
  - Samples taken are evicted from the cache
  - Samples read remain in the cache and are simply market as read
- The cache content can be selected based on content or state. More on this later...

DATA SELECTION

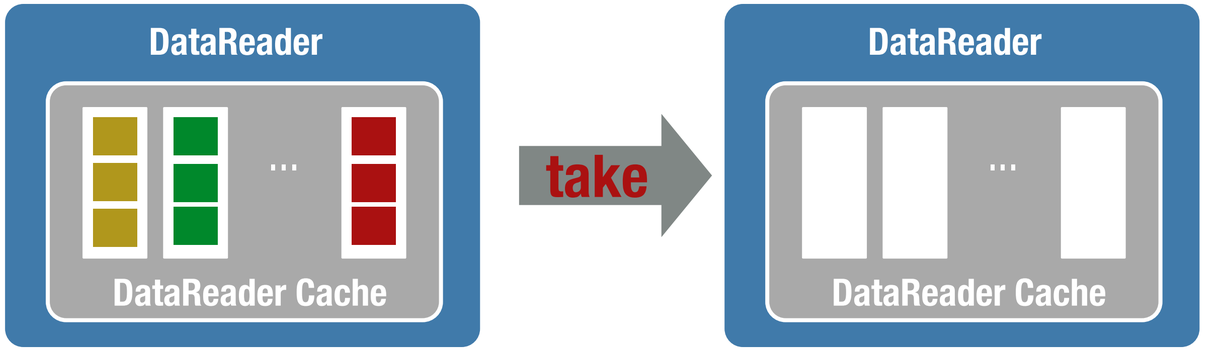
READING DATA

- The action of **reading samples** for a Reader Cache is **non-destructive**.
- **Samples** are **not removed** from the cache



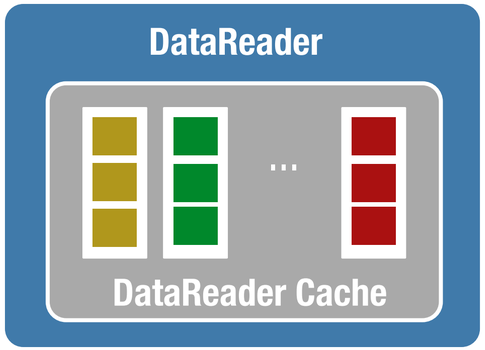
TAKING DATA

- The action of **taking samples** for a Reader Cache is **destructive**.
- **Samples** are **removed** from the cache



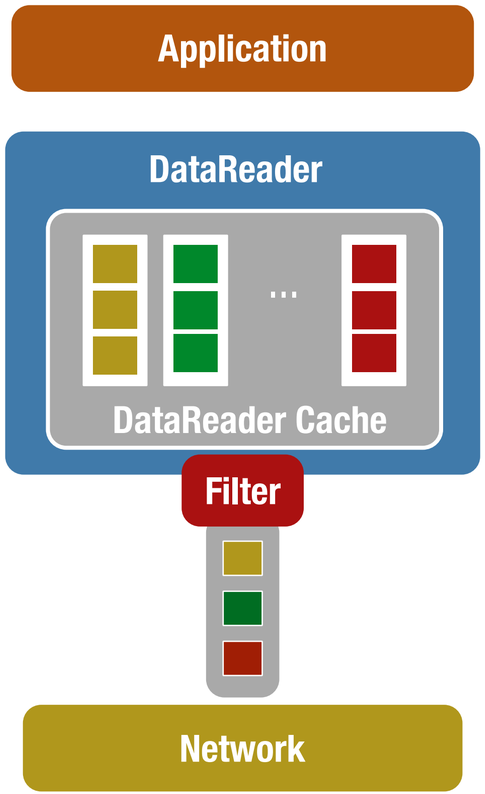
SAMPLES SELECTOR

- Samples can be selected using composable content and status predicates



CONTENT FILTERING

- Filters allow to control what gets into a DataReader cache
- Filters are expressed as SQL where clauses or as Java/C/JavaScript predicates

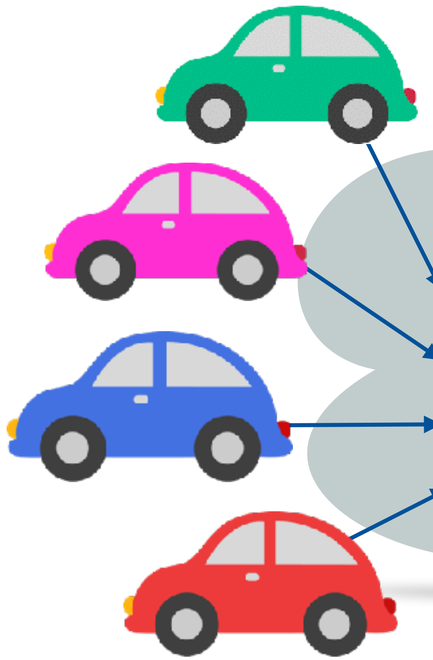


CONTENT FILTERS

Content Filters can be used to **project** on the local cache only the Topic data satisfying a given predicate

Type

```
struct CarDynamics {
   @key
   string  cid;
   long    x;   long  y;
   float   dx;  long  dy;
}
```



CarDynamics

cid	x	y	dx	dy
GR 33N GO	167	240	45	0
LO 00V IN	65	26	65	0
AN 637 OS	32	853	0	50
AB 123 CD	325	235	80	0

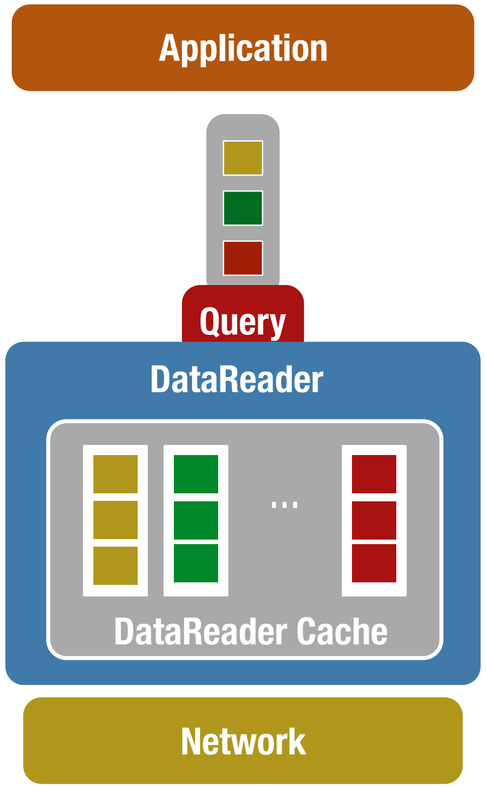
“dx > 50 OR dy > 50”

cid	x	y	dx	dy
LO 00V IN	65	26	65	0
AB 123 CD	325	235	80	0

Reader Cache

CONTENT-BASED SELECTION

- Queries allow to control what gets out of a DataReader Cache
- Queries are expressed as SQL where clauses or as Java/C/JavaScript predicates

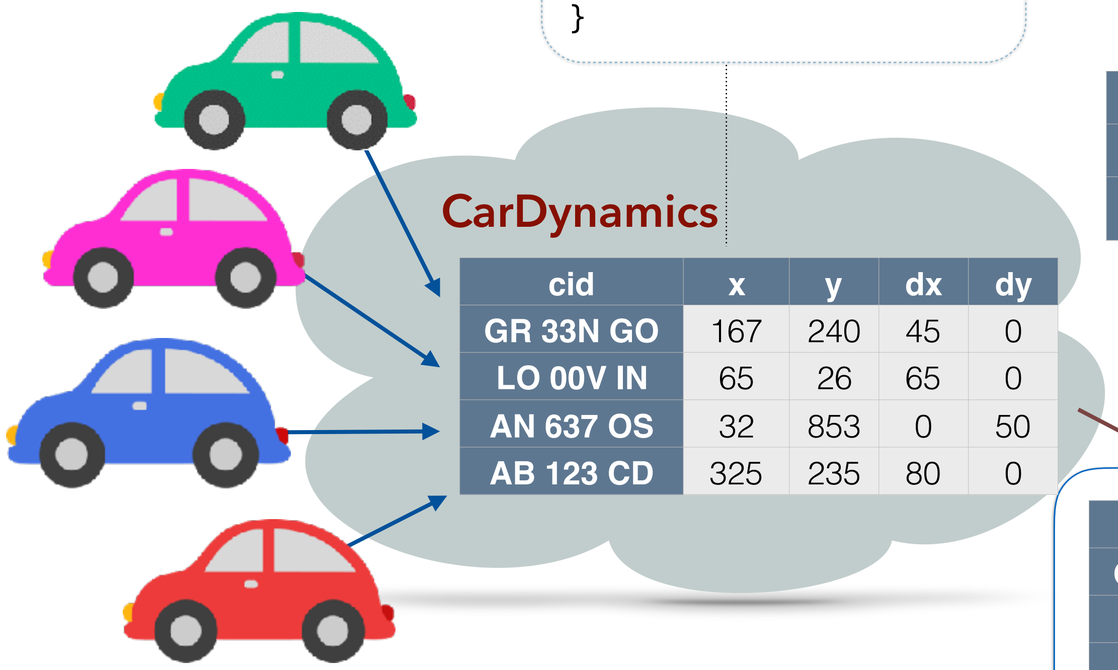


QUERIES

Queries can be used to select out of the local cache the data matching a given predicate

Type

```
struct CarDynamics {
    @key
    string  cid;
    long    x;    long  y;
    float   dx;   long  dy;
}
```



CarDynamics

cid	x	y	dx	dy
GR 33N GO	167	240	45	0
LO 00V IN	65	26	65	0
AN 637 OS	32	853	0	50
AB 123 CD	325	235	80	0

cid	x	y	dx	dy
LO 00V IN	65	26	65	0
AB 123 CD	325	235	80	0

`“dx > 50 OR dy > 50”` query

cid	x	y	dx	dy
GR 33N GO	167	240	45	0
LO 00V IN	65	26	65	0
AN 637 OS	32	853	0	50
AB 123 CD	325	235	80	0

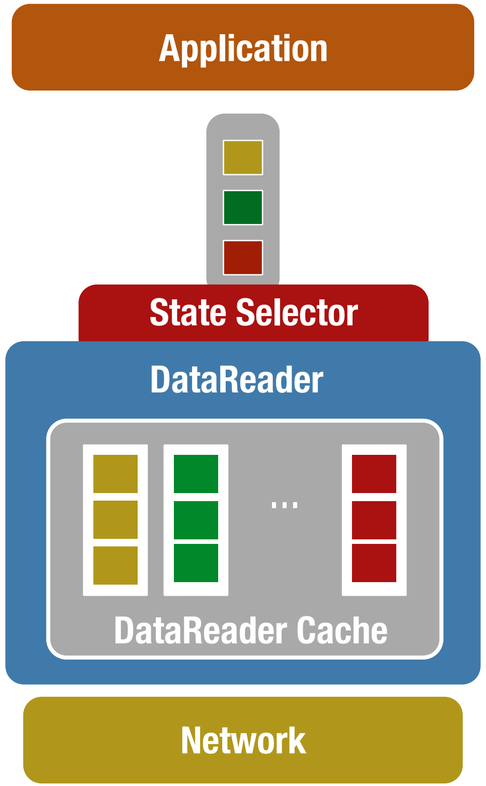
Reader Cache

SAMPLE, INSTANCE, AND VIEW STATE

- The samples included in the DataReader cache have associated some meta-information which, among other things, describes the status of the sample and its associated stream/instance
- The **Sample State** (READ, NOT_READ) allows to distinguish between new samples and samples that have already been read
- The **View State** (NEW, NOT_NEW) allows to distinguish a new instance from an existing one
- The **Instance State** (ALIVE, NOT_ALIVE_DISPOSED, NOT_ALIVE_NO_WRITERS) allows to track the life-cycle transitions of the instance to which a sample belongs

STATE-BASED SELECTION

- State based selection allows to control what gets out of a DataReader Cache
- State base selectors predicate on samples meta-information



INTERACTION MODELS

INTERACTION MODELS

Polling

- The application proactively polls for data availability as well as special events, such as a deadline being missed, etc. Notice that all DDS API calls, exclusion made for *wait* operations, are non-blocking

Synchronous Notification

- The application synchronously waits for some conditions to be verified, e.g., data availability, instance lifecycle change, etc.

Asynchronous Notification

- The application registers the interest to be asynchronously notified when specific condition are satisfied, e.g. data available, a publication matched, etc.

SYNCHRONOUS NOTIFICATIONS

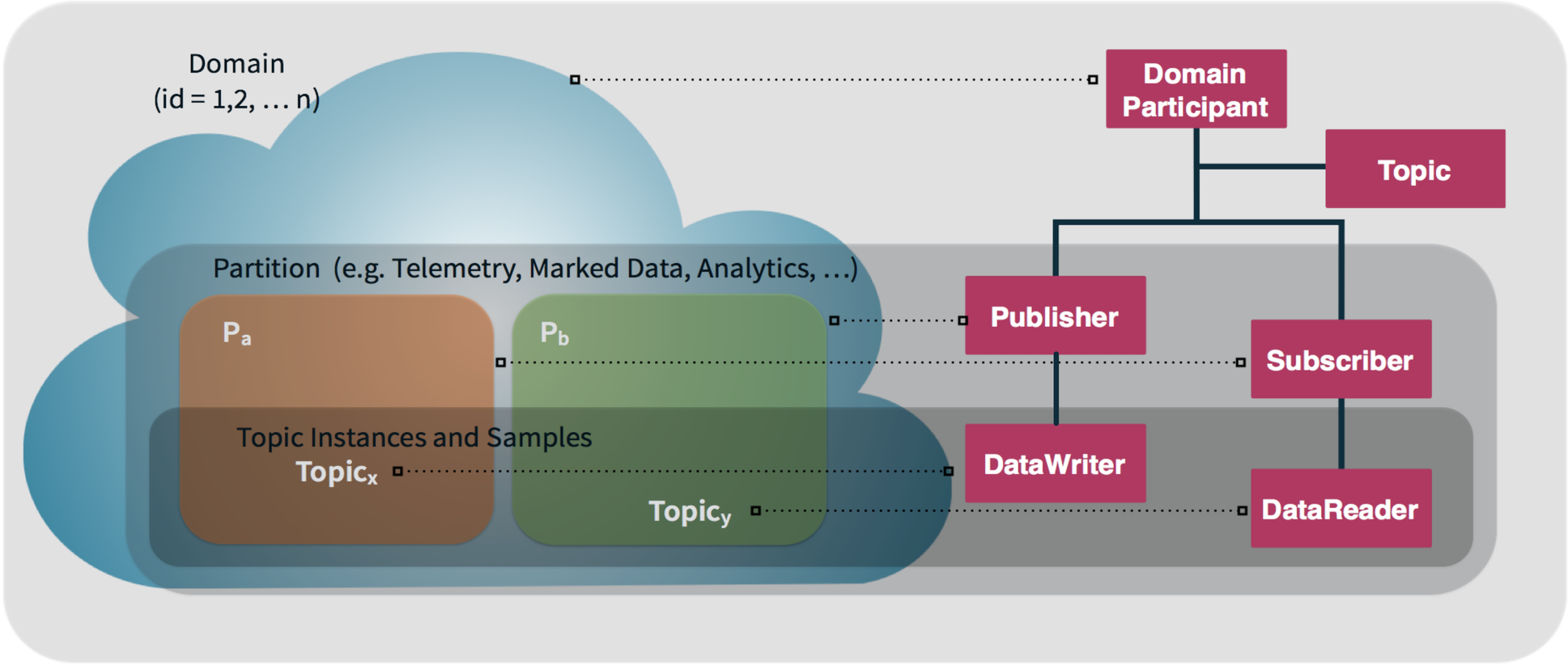
- DDS provides a mechanism known as **WaitSet** to synchronously wait for a condition
- Condition can predicate on:
  - communication statuses
  - data availability
  - data availability with specific content
  - user-triggered conditions

ASYNCHRONOUS NOTIFICATIONS

- DDS provides a mechanism known as **Listeners** for asynchronous notification of a given condition
- Listener interest can predicate on:
  - communication statuses
  - data availability

PUTTING IT ALL TOGETHER

Anatomy of a DDS Application



WRITING DATA IN C++

```
#include <dds.hpp>

int main(int, char**) {

  DomainParticipant dp(0);
  Topic<Meter> topic("SmartMeter");
  Publisher pub(dp);
  DataWriter<Meter> dw(pub, topic);

  while (!done) {
     auto value = readMeter()
     dw.write(value);
     std::this_thread::sleep_for(SAMPLING_PERIOD);
  }

  return 0;
}
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
 };

 struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
 };
 #pragma keylist Meter sn
```

READING DATA IN C++

```
#include <dds.hpp>

int main(int, char**) {

  DomainParticipant dp(0);
  Topic<Meter> topic("SmartMeter");
  Subscriber sub(dp);
  DataReader<Meter> dr(dp, topic);

  LambdaDataReaderListener<DataReader<Meter>> lst;
  lst.data_available = [](DataReader<Meter>& dr) {
       auto samples = data.read();
       std::for_each(samples.begin(), samples.end(), [](Sample<Meter>& sample) {
         std::cout << sample.data() << std::endl;
       }
   }
   dr.listener(lst);
   // Print incoming data up to when the user does a Ctrl-C
   std::this_thread::join();
   return 0;
}
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
 };

 struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
 };
 #pragma keylist Meter sn
```

WRITING DATA IN SCALA

```scala
import dds._
import dds.prelude._
import dds.config.DefaultEntities._

object SmartMeter {

  def main(args: Array[String]): Unit = {
    val topic = Topic[Meter](“SmartMeter”)
    val dw = DataWriter[Meter](topic)
    while (!done) {
      val meter = readMeter()
      dw.write(meter)
      Thread.sleep(SAMPLING_PERIOD)
    }
  }
}
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
 };

 struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
 };
 #pragma keylist Meter sn
```

READING DATA IN SCALA

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```

```
import dds._
import dds.prelude._
import dds.config.DefaultEntities._


object SmartMeterLog {
  def main(args: Array[String]): Unit = {
    val topic = Topic[Meter](“SmartMeter”)
    val dr = DataReader[Meter](topic)
    dr listen {
      case DataAvailable(_) => dr.read.foreach(println)
    }
  }
}
```

WRITING DATA IN PYTHON

```
import dds
import time

if __name__ == '__main__':
    topic = dds.Topic("SmartMeter", "Meter")
    dw = dds.Writer(topic)

    while True:
        m = readMeter()
        dw.write(m)
        time.sleep(0.1)
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```

READING DATA IN PYTHON

```python
import dds
import sys

def readData(dr):
    samples = dds.range(dr.read())
    for s in samples:
        sys.stdout.write(str(s.getData()))

if __name__ == '__main__':
    t = dds.Topic("SmartMeter", "Meter")
    dr = dds.Reader(t)
    dr.onDataAvailable = readData
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```

DDS VS. MQTT ETC.

IIC CONNECTIVITY

The recently released **IIC Connectivity Framework** reveals how the **OMG DDS** is the fittest standard for connectivity in IIoT

	Core Standard Criterion	DDS	Web Services	OPC-UA	oneM2M
1	Provide **syntactic interoperability**#	✓	Need XML or JSON	✓	✓
2	Open standard with strong **independent, international** governance#	✓	✓	✓	✓
3	**Horizontal** and neutral in its applicability across industries#	✓	✓	✓	✓
4	**Stable** and **deployed** across multiple vertical industries#	Software Integration & Autonomy	✓	Manufacturing	Home Automation
5	Have **standards-defined *Core Gateways*** to all other core connectivity standards#	Web Services, OPC-UA*, oneM2M*	DDS, OPC-UA, oneM2M	Web Services, DDS*, oneM2M*	Web Services, OPC-UA*, DDS*
6	Meet the connectivity framework **functional** requirements	✓	✗	Pub-Sub in development	✓
7	Meet **non-functional** requirements of performance, scalability, reliability, resilience	✓	✗	Real-time in development	Reports not yet documented or public
8	Meet **security** and **safety** requirements	✓	✓	✓	✓
9	Not require any single component from any single vendor	✓	✓	✓	✓
10	Have readily-available SDKs both **commercial** and **open source**	✓	✓	✓	✓

#green = Gating Criteria * = work in progress, ✓= supported, ✗ = **not** supported

Table 8-1: IIoT Connectivity Core Standards Criteria applied to key connectivity framework standards.

DDS VS OTHER STANDARDS

	Transport	Paradigm	Scope	Discovery	Content Awareness	Data Centricity	Security	Data Prioritisation	Fault Tolerance
AMQP	TCP/IP	Point-to-Point Message Exchange	D2D D2C C2C	No	None	Encoding	TLS	None	Impl. Specific
CoAP	UDP/IP	Request/ Reply (REST)	D2D	Yes	None	Encoding	DTLS	None	Decentralised
DDS	UDP/IP (unicast + mcast) TCP/IP	Publish/ Subscribe Request/ Reply	D2D D2C C2C	Yes	Content-Based Routing, Queries	Encoding, Declaration	TLS, DTLS, DDS Security	Tranport Priorities	Decentralised
MQTT	TCP/IP	Publish/ Subscribe	D2C	No	None	Undefined	TLS	None	Broker is the SPoF

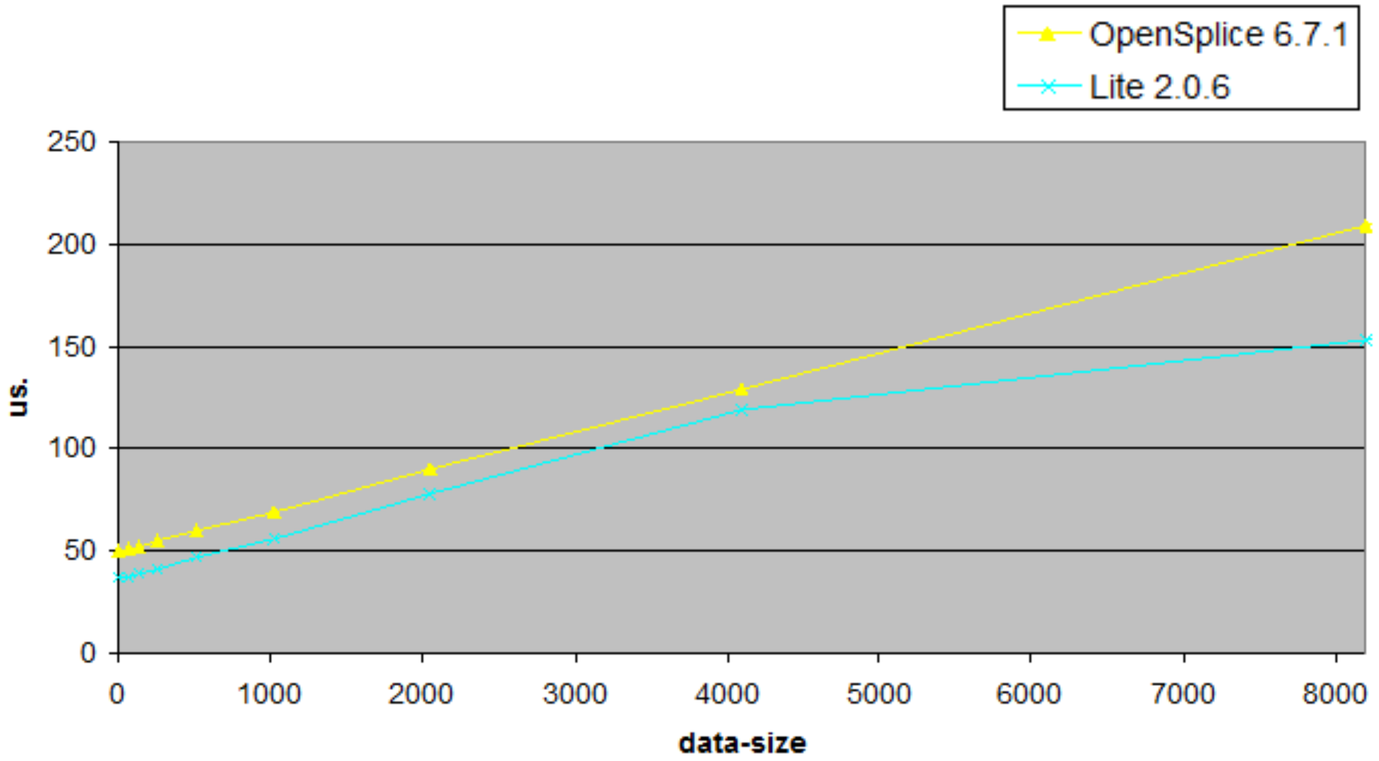
[Ref: A Comparative Study of Data-Sharing Standards for the Internet of Things, Cutter Journal, Dec 2014

PERFORMANCE

LATENCY

Vortex DDS latency can be as low as ~30 usec

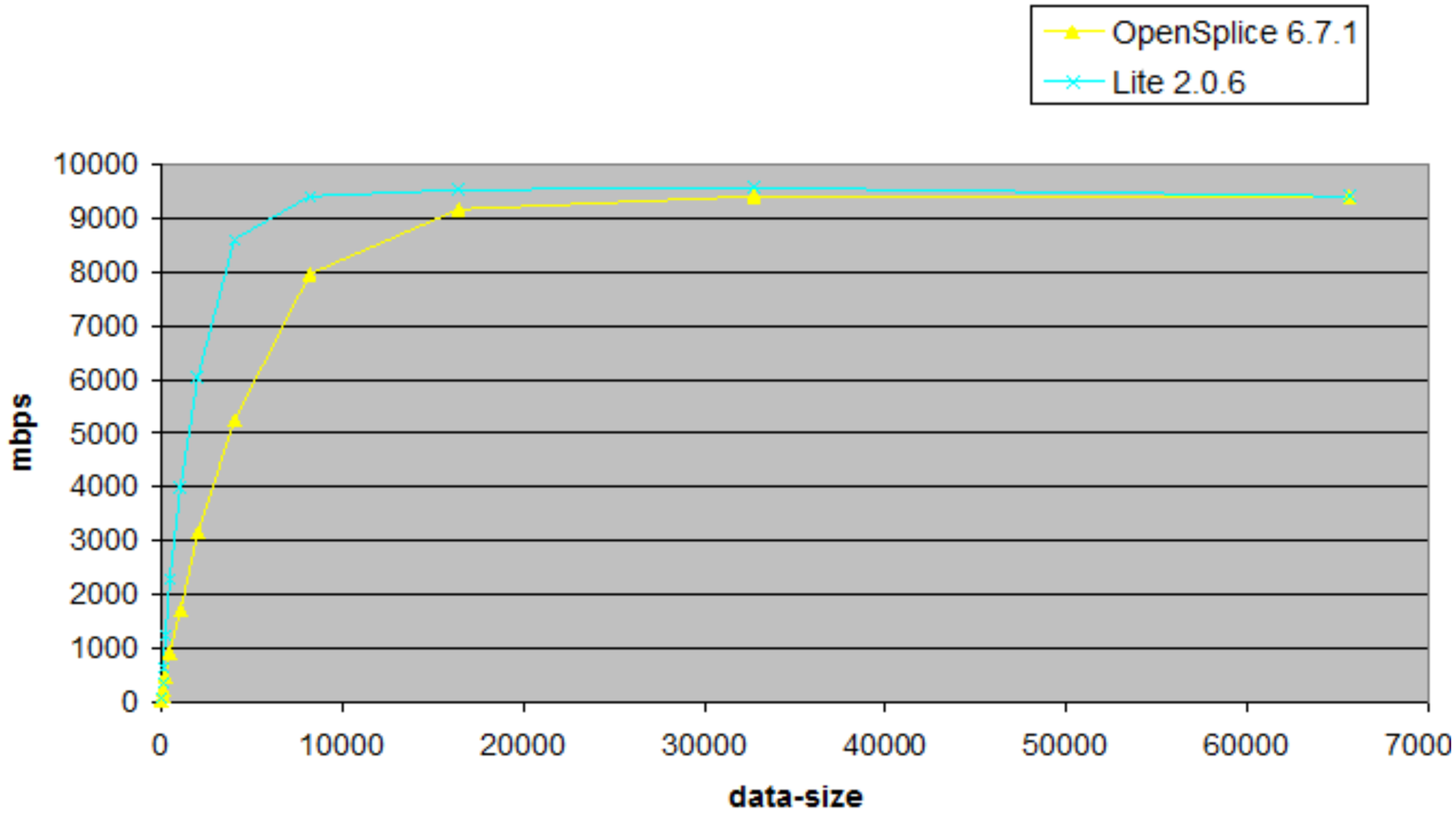
This is the lowest among DDS implementations several times better than what can be obtained with MQTT, AMQP, OPC-UA



THROUGHPUT

Vortex DDS can easily saturate a 10Gbps network

Vortex throughput is 2-3x better than competing technologies



CONCLUDING REMARKS

WRAPPING UP

DDS provides a very high level abstractions to architect and implement distributed systems

DDS has built-in support for several patterns that are essential to keep systems working at scale, such as circuit-breakers and temporal decoupling

DDS can address the most challenging environment w.r.t. volumes, latencies and scale