THE Data Distribution Service

Angelo Corsaro, PhD

CTO, ADLINK Tech. Inc.
Co-Chair, OMG DDS-SIG
Board Director, OMG
angelo.corsaro@adlinktech.com

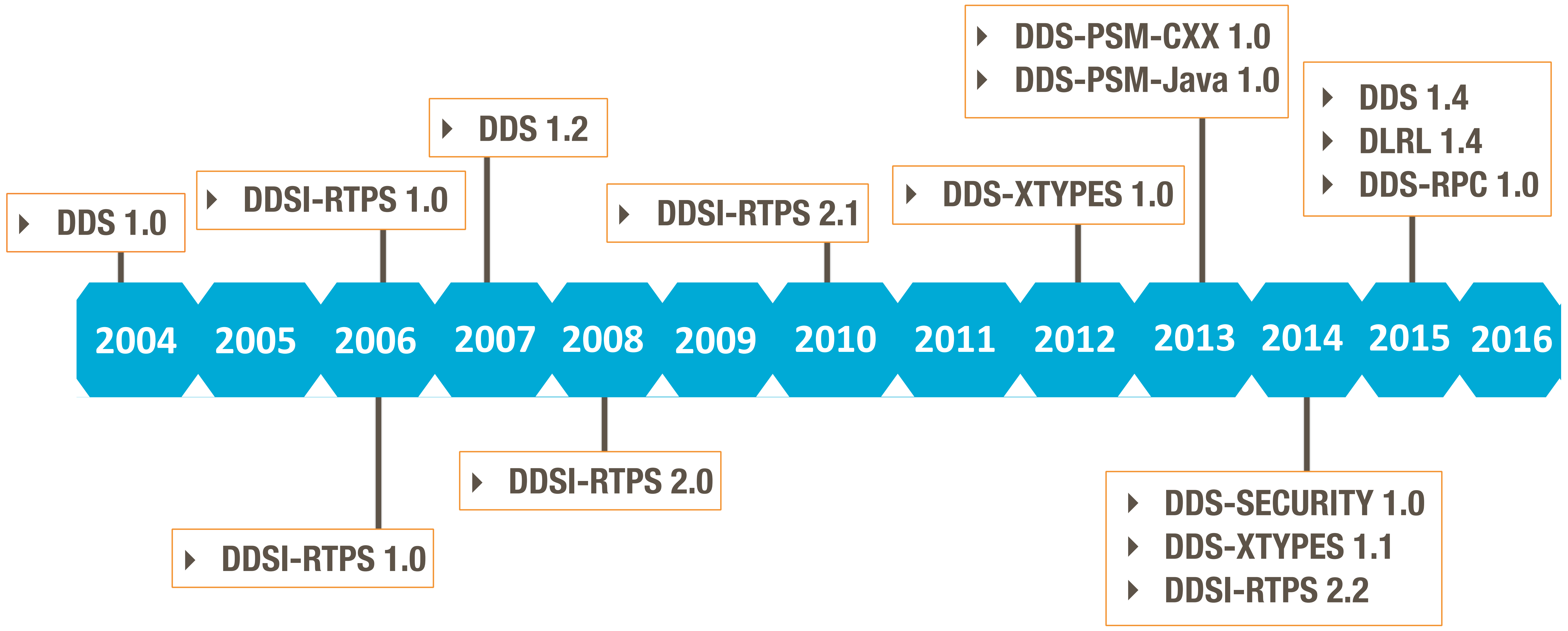
PRISMTECH™ AN ADLINK COMPANY

What is DDS?

DDS

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

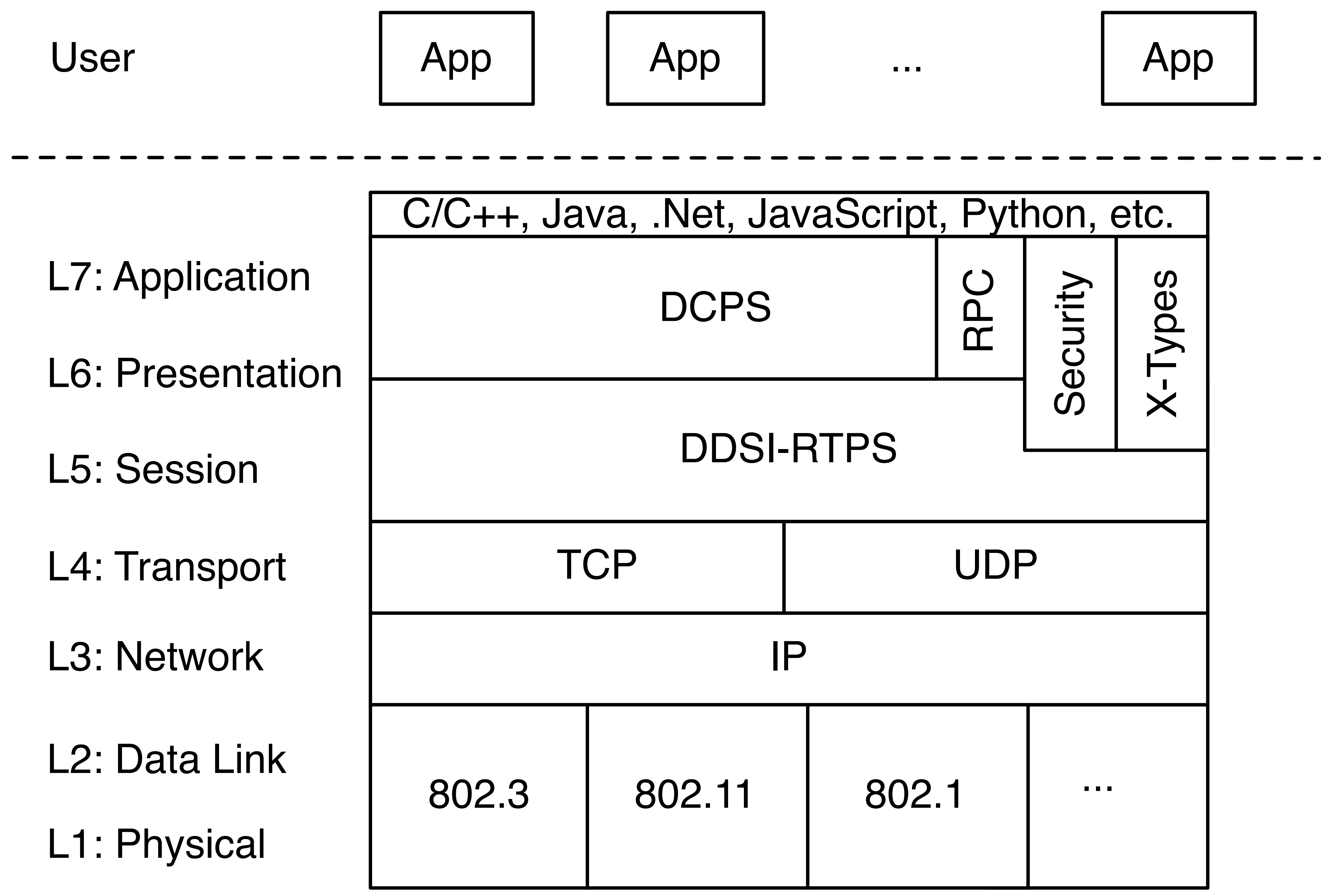
DDS Standard Evolution



DDS Standards

Data Centric Publish Subscribe (DCPS)

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

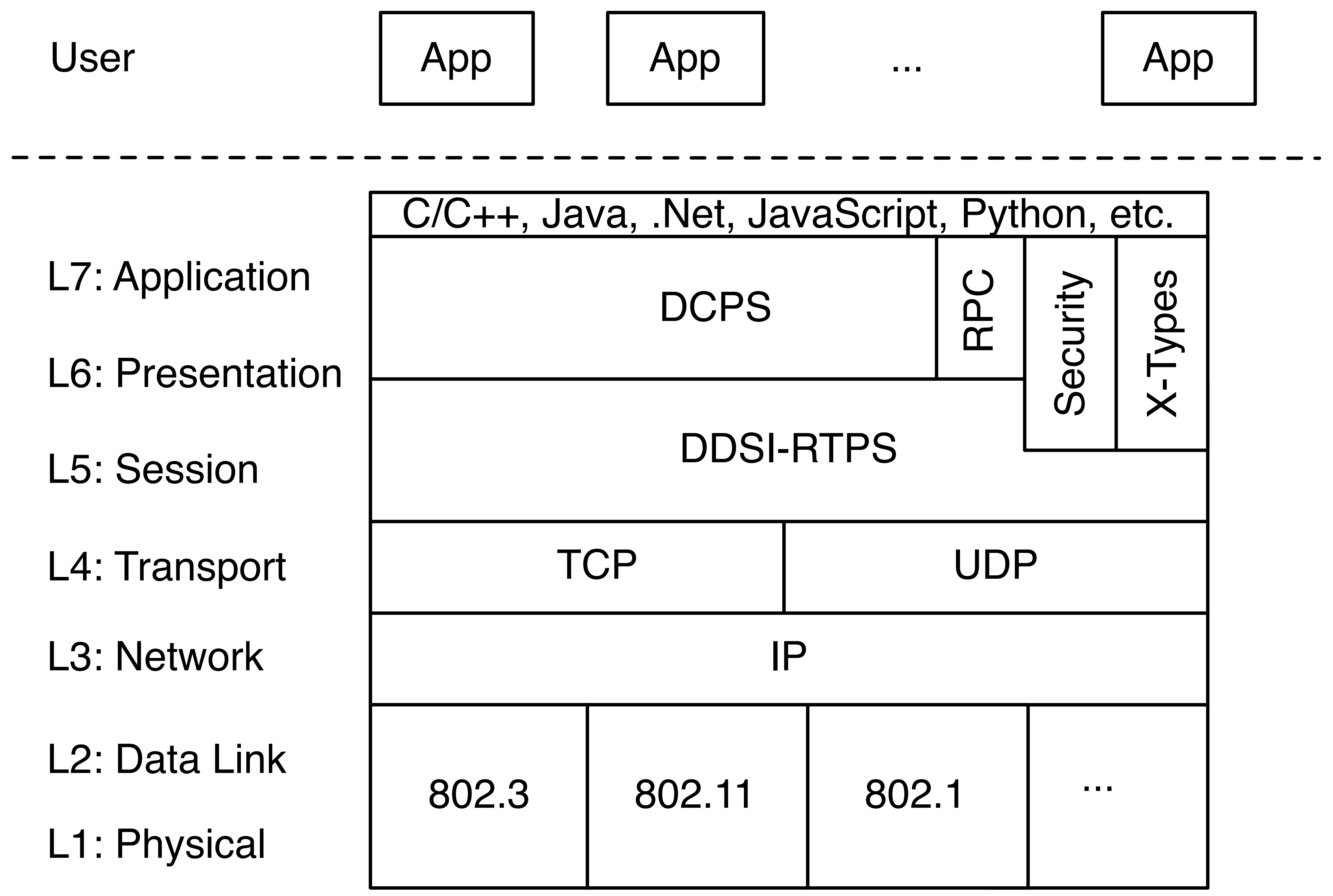


DDS Standards

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.

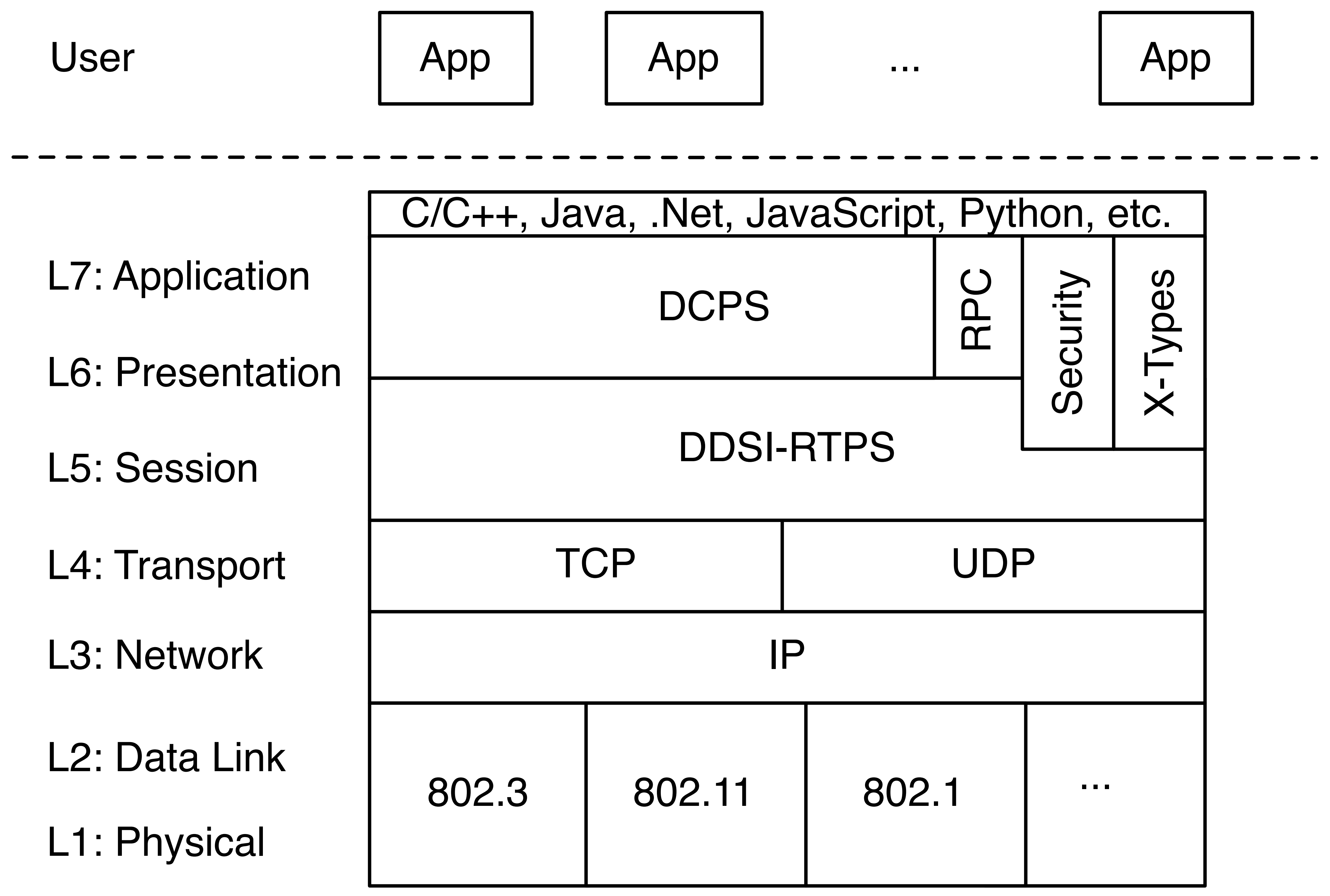


DDS Standards

eXtensible Types (DDS-XTypes)

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

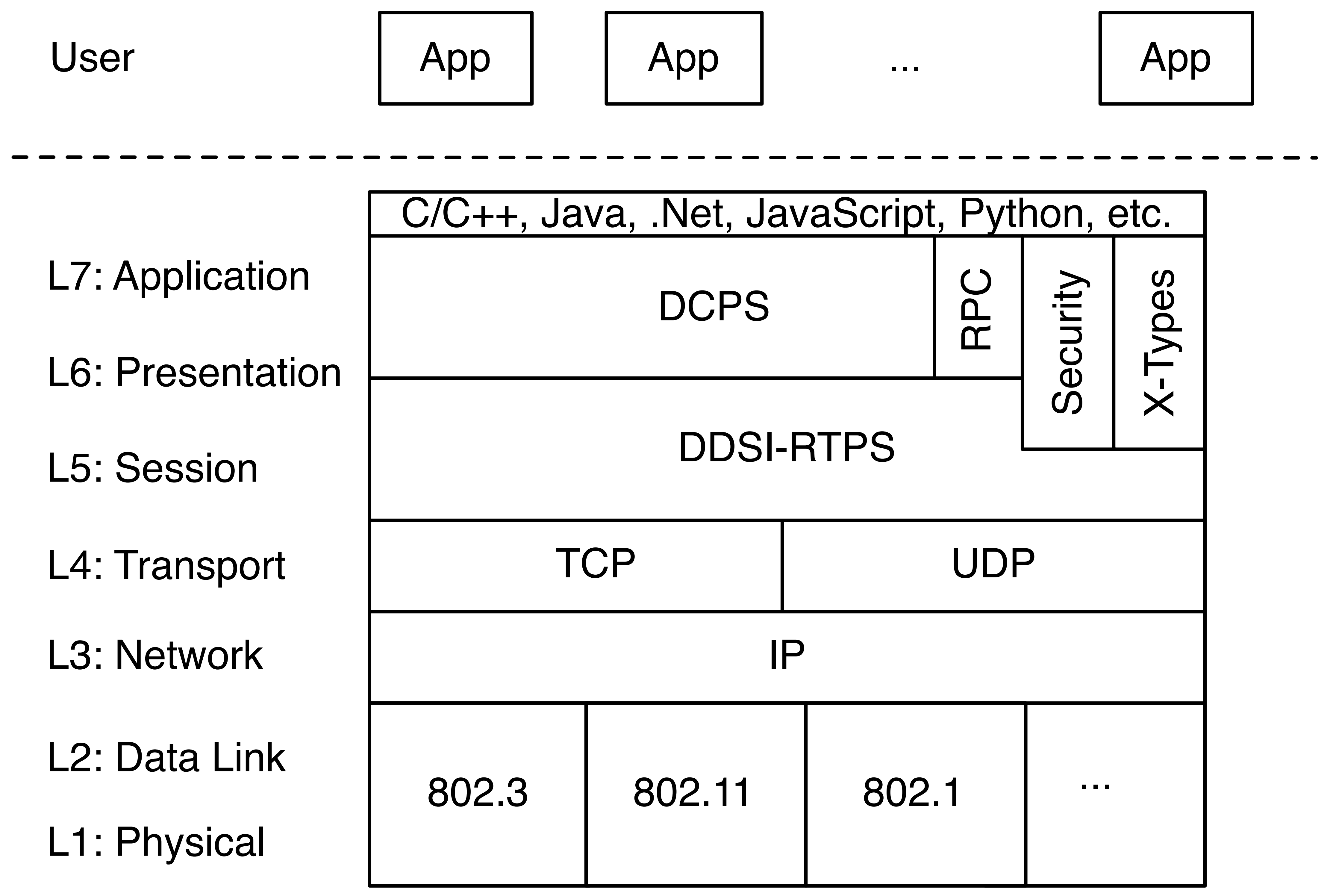
Defines APIs for dynamically defining and operating over DDS types



DDS Standards

Security (DDS-Security)

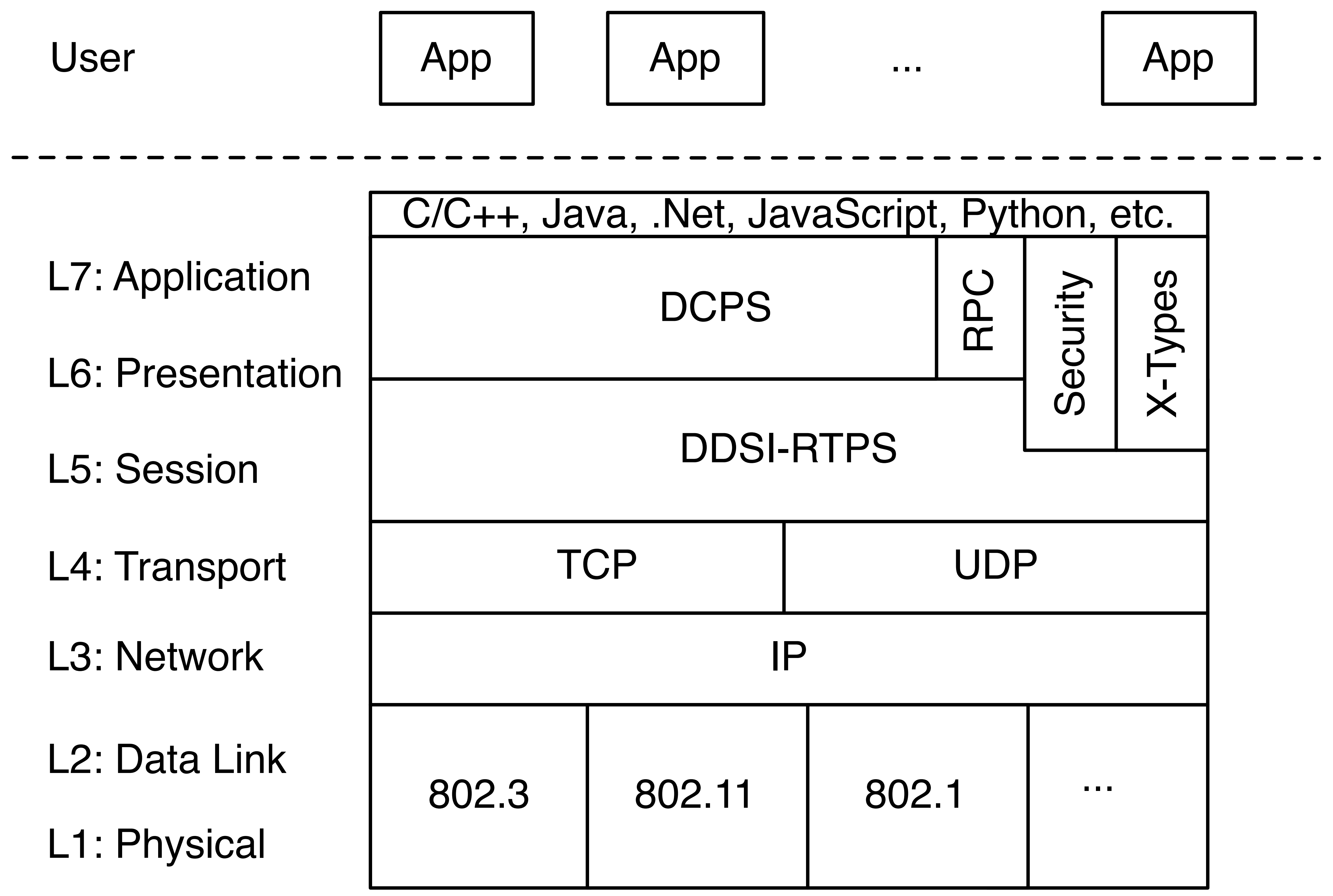
Defines a data-centric security architecture with pluggable Authentication, Access Control, Crypto and Logging.



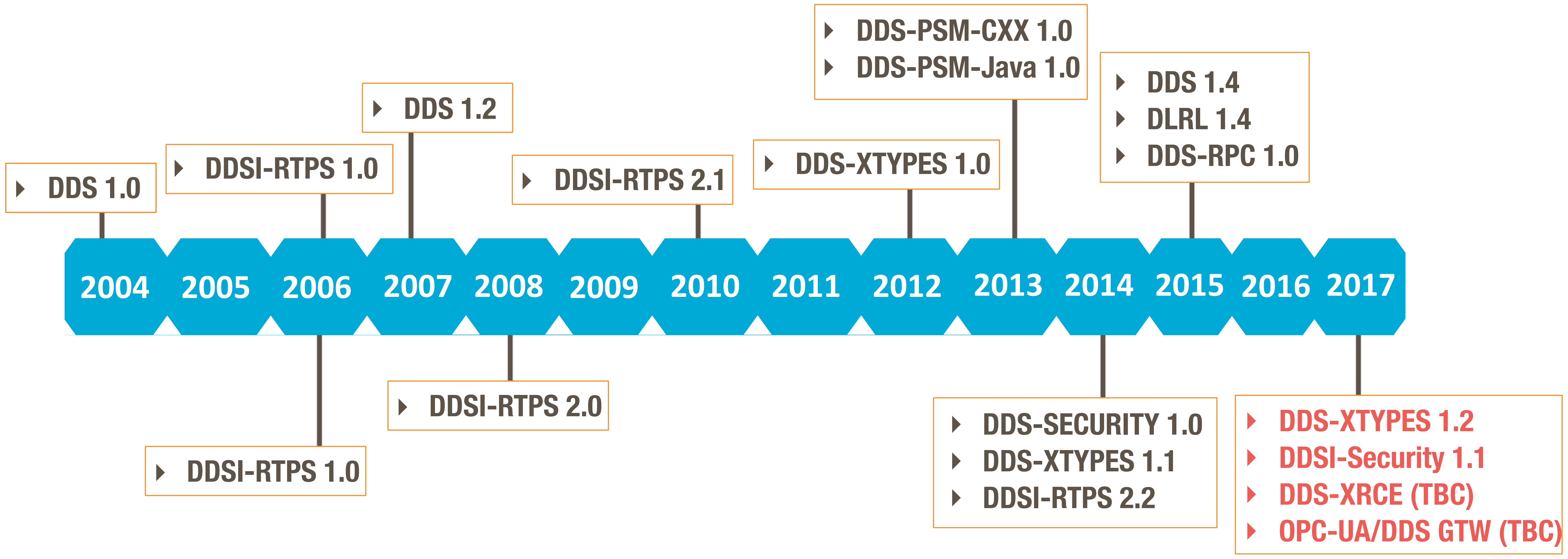
DDS Standards

Remote Procedure Calls (DDS-RPC)

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



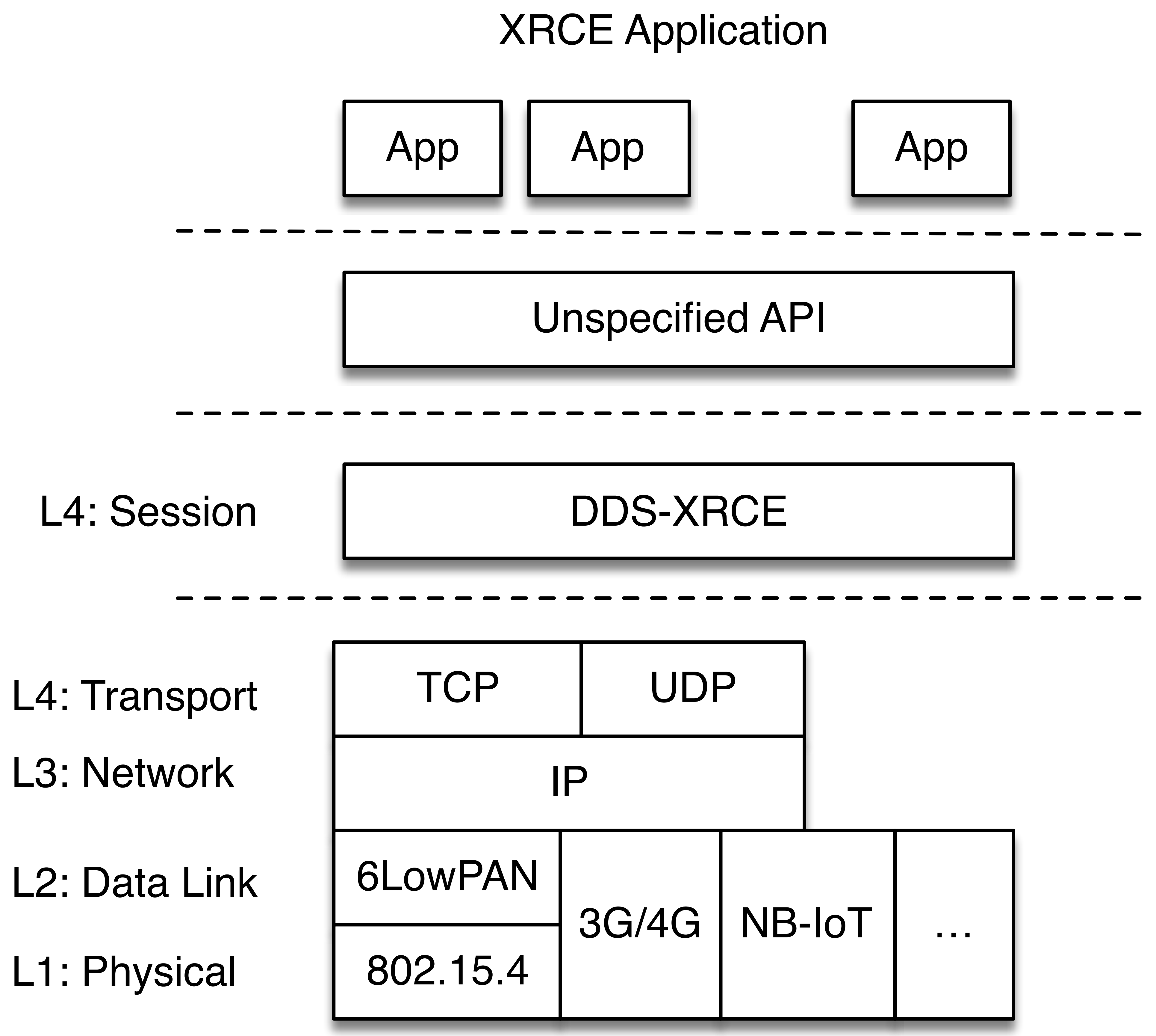
Upcoming Standards



DDS-XRCE

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**

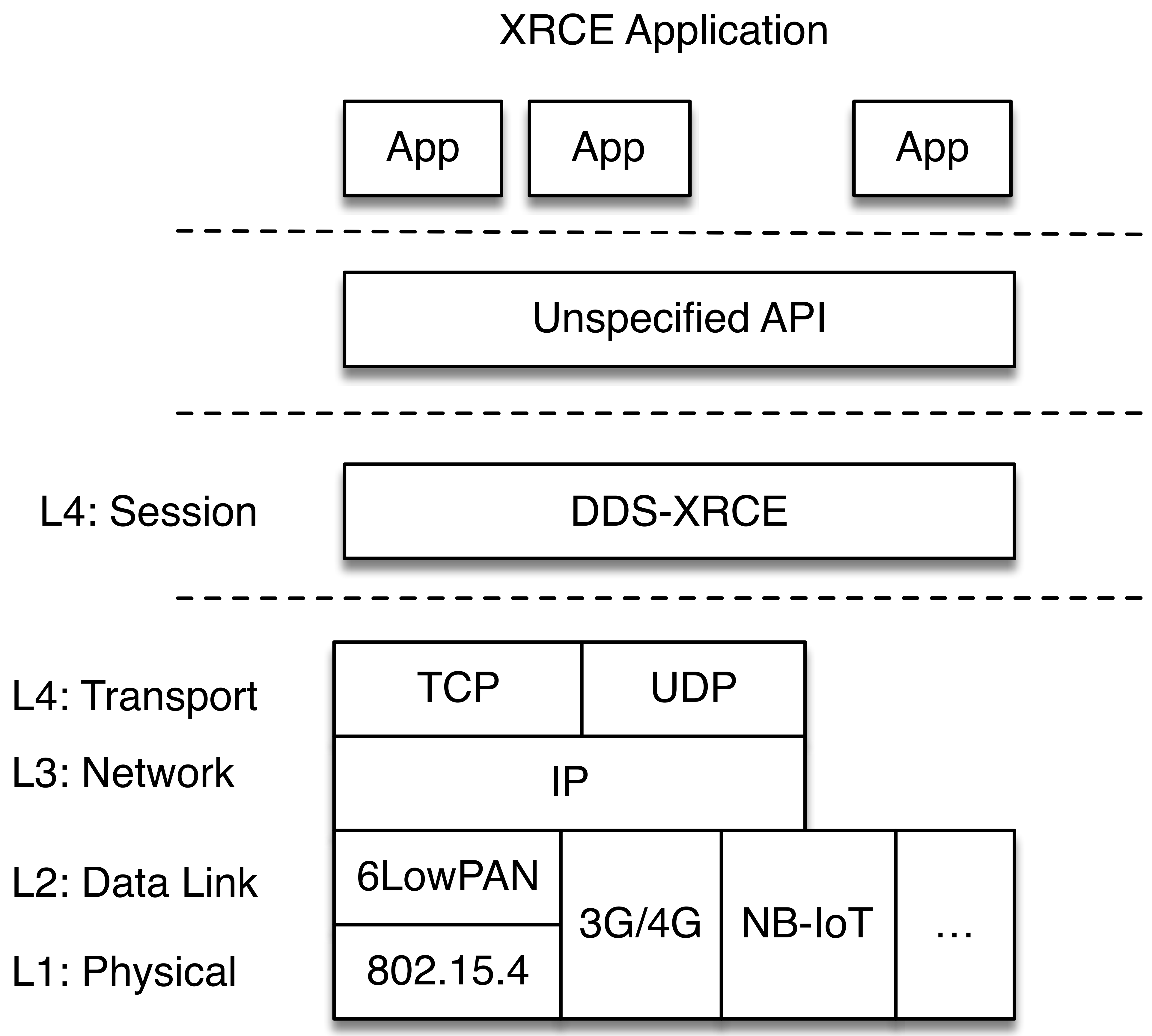


DDS-XRCE

Supports peer-to-peer as well as broker-based communication

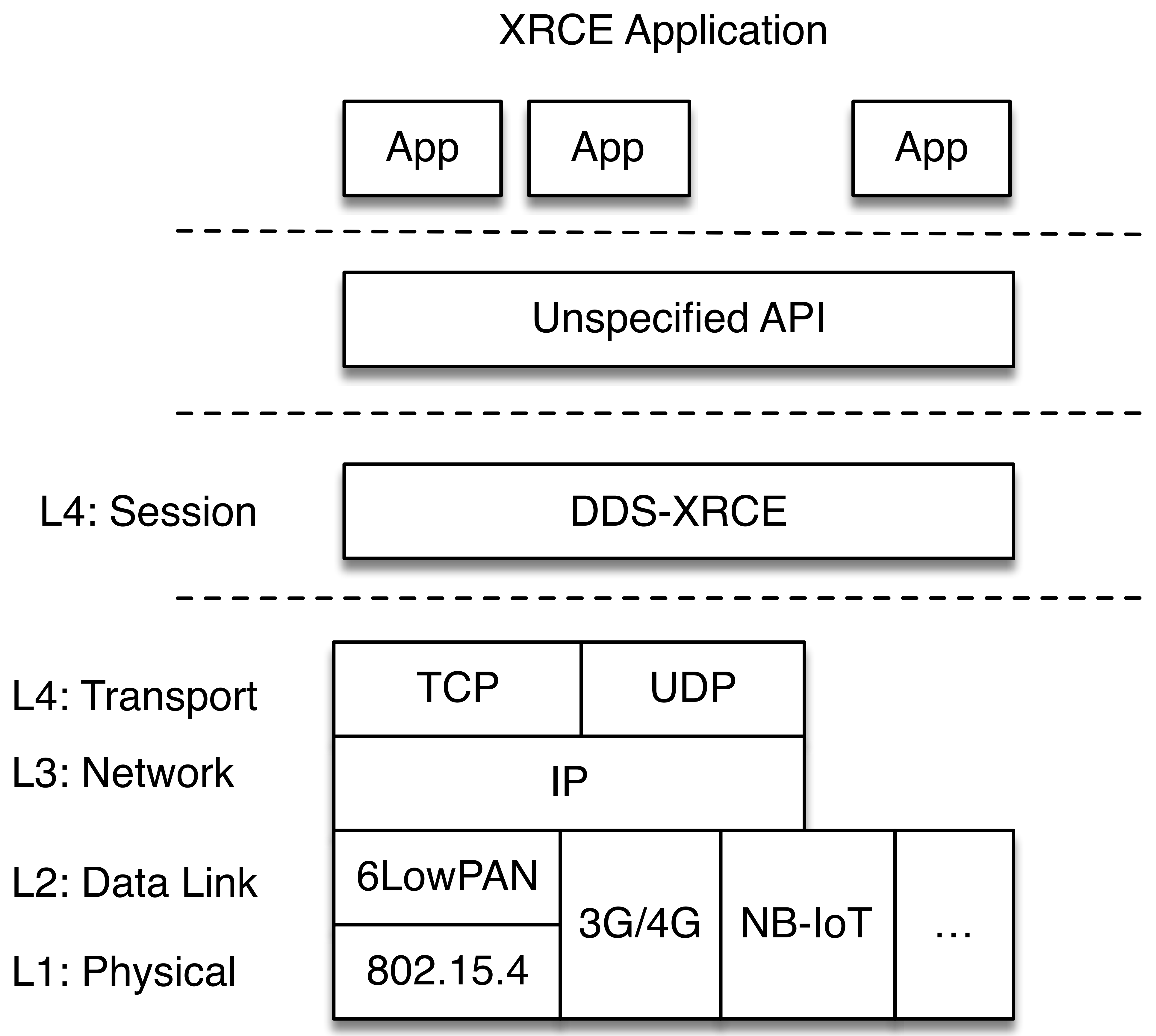
Provides reliability and fragmentation above packet-oriented best-effort transports

Can leverage multicast



DDS-XRCE

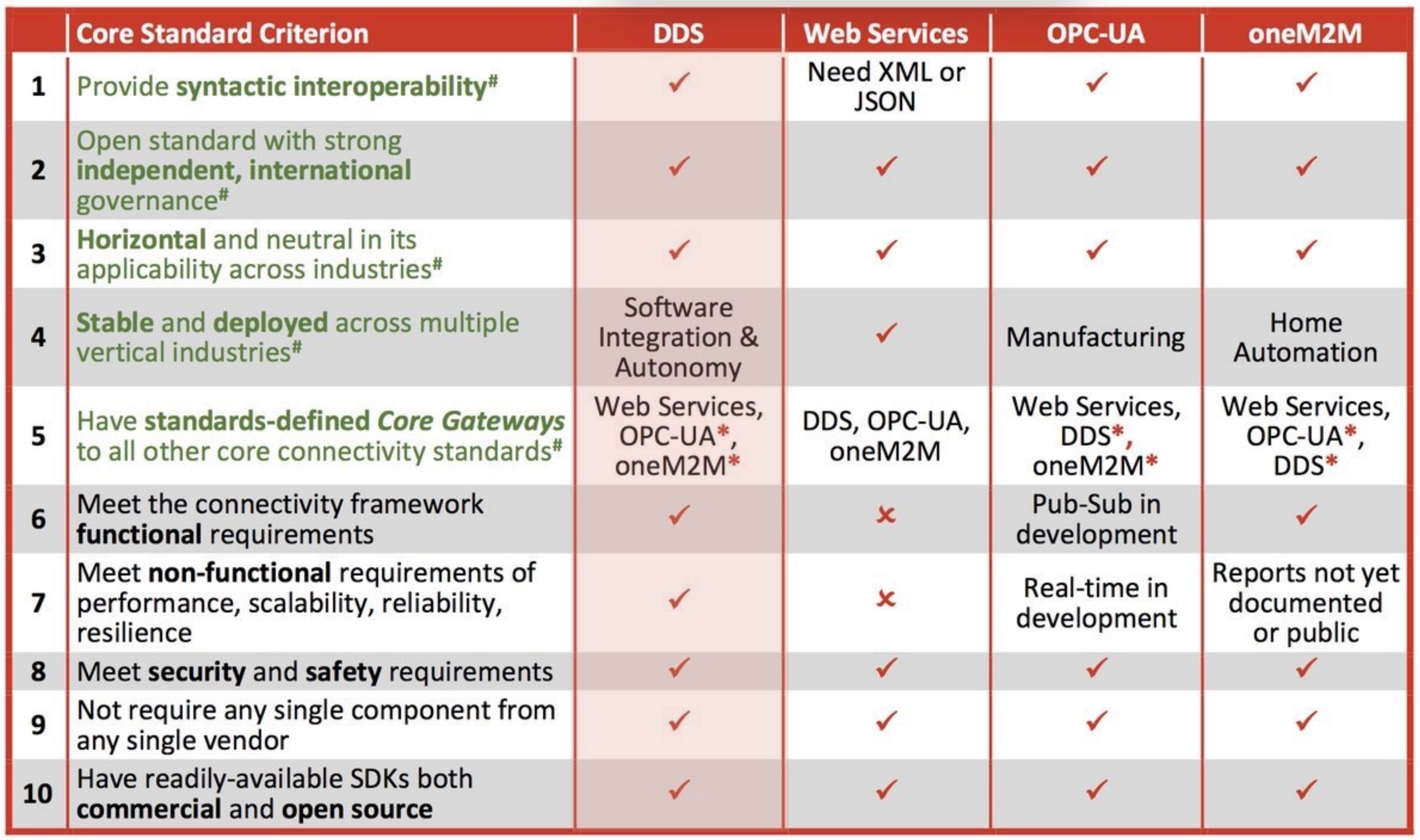
Current prototype runs on **8-bit micro-controllers** and takes in **1 KByte of RAM** and has **wire-overhead** of **3-4 bytes** for data samples



DDS Standard in IIoT

IIC Connectivity

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



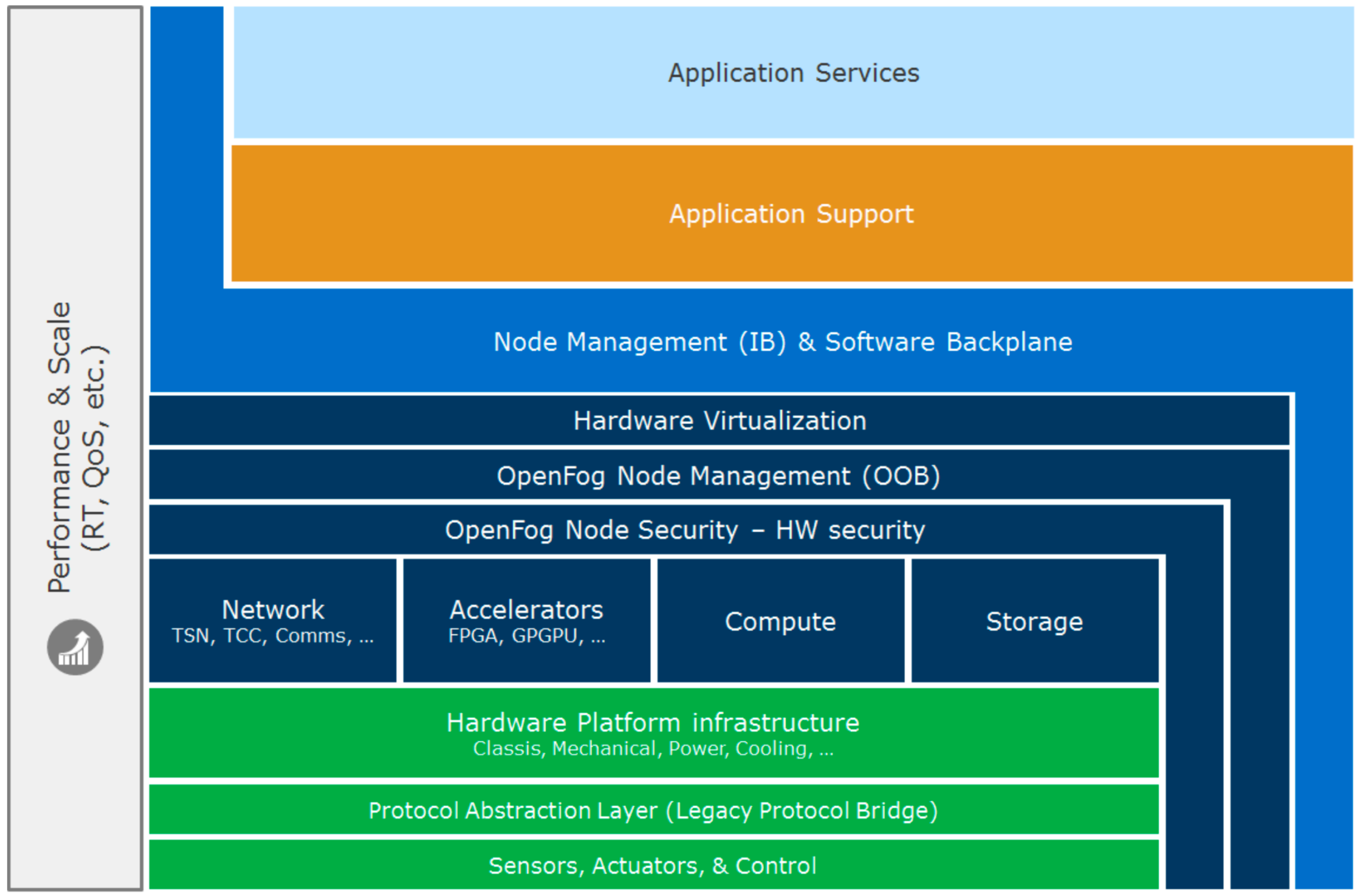
	Core Standard Criterion	DDS	Web Services	OPC-UA	oneM2M
1	Provide **syntactic interoperability**#	✓	Need XML or JSON	✓	✓
2	Open standard with strong **independent, international** governance#	✓	✓	✓	✓
3	**Horizontal** and neutral in its applicability across industries#	✓	✓	✓	✓
4	**Stable** and **deployed** 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.

Open Fog Reference Architecture

The **OpenFog Consortium Reference Architecture** identifies the **OMG DDS** as one for the **key Connectivity** and **Data Management** standards

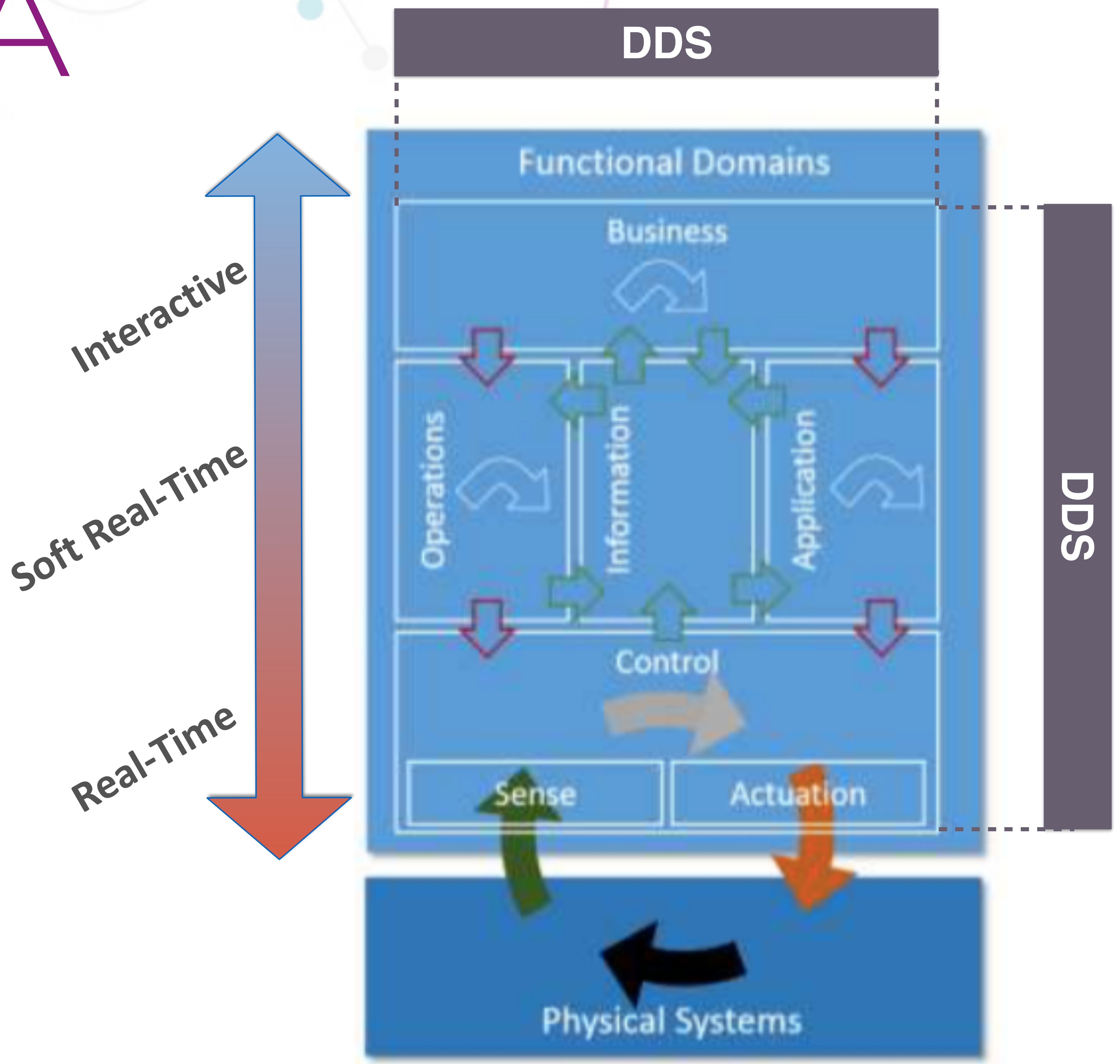


Performance & Scale (RT, QoS, etc.)
Application Services
Application Support
Node Management (IB) & Software Backplane
Hardware Virtualization
OpenFog Node Management (OOB)
OpenFog Node Security – HW security
Network TSN, TCC, Comms, ...
Accelerators FPGA, GPGPU, ...
Compute
Storage
Hardware Platform infrastructure Classis, Mechanical, Power, Cooling, ...
Protocol Abstraction Layer (Legacy Protocol Bridge)
Sensors, Actuators, & Control

DDS in IIRA

DDS is widely used for horizontal (east-to-west) communication on the Control and Information Layers

But it is applicable for horizontal across any view



Who is using DDS?

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

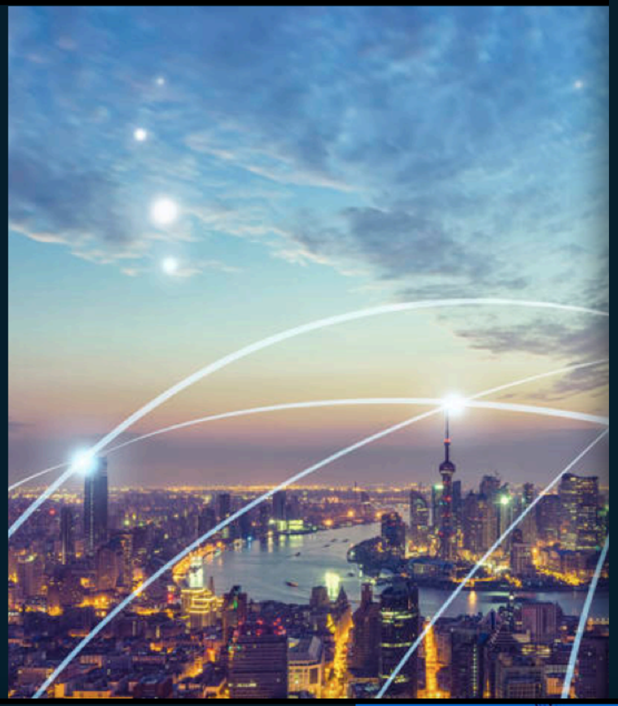
Vortex is used for data sharing within and across the vehicles.

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

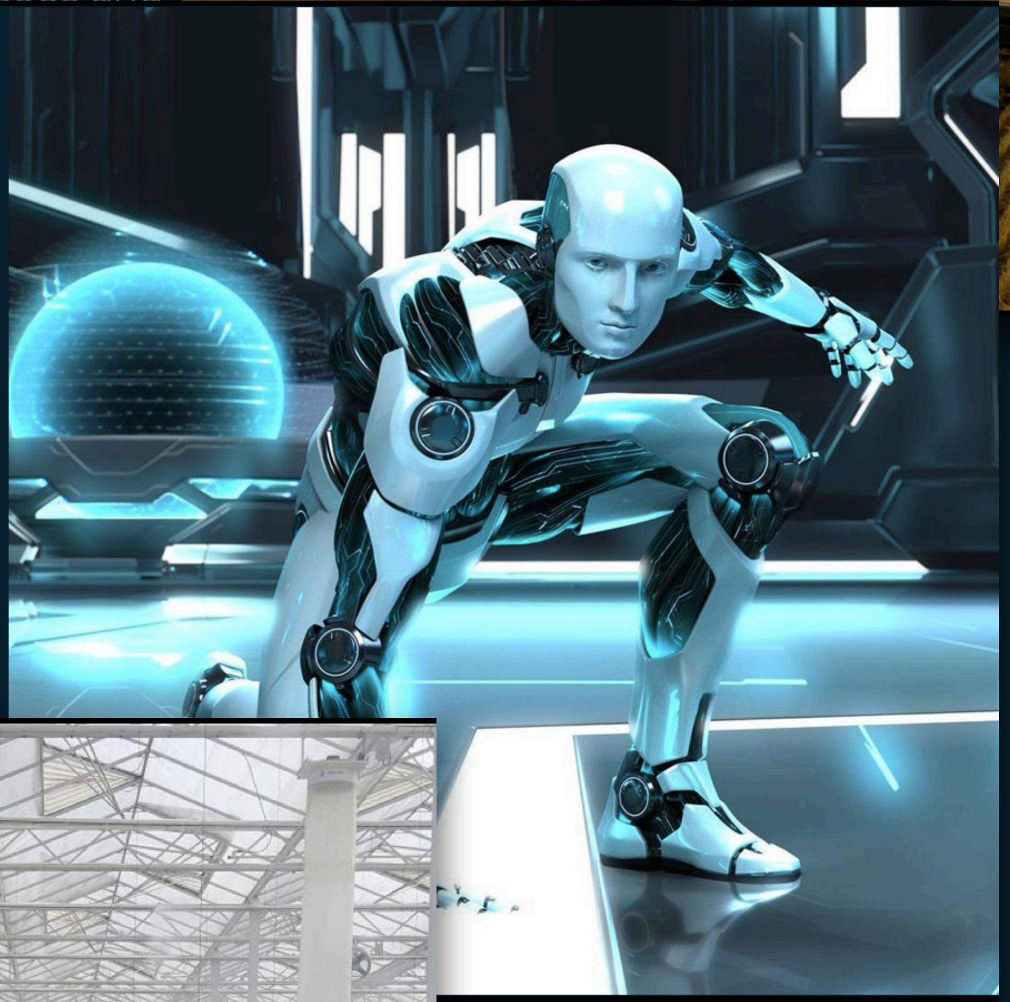


SMART FACTORY

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

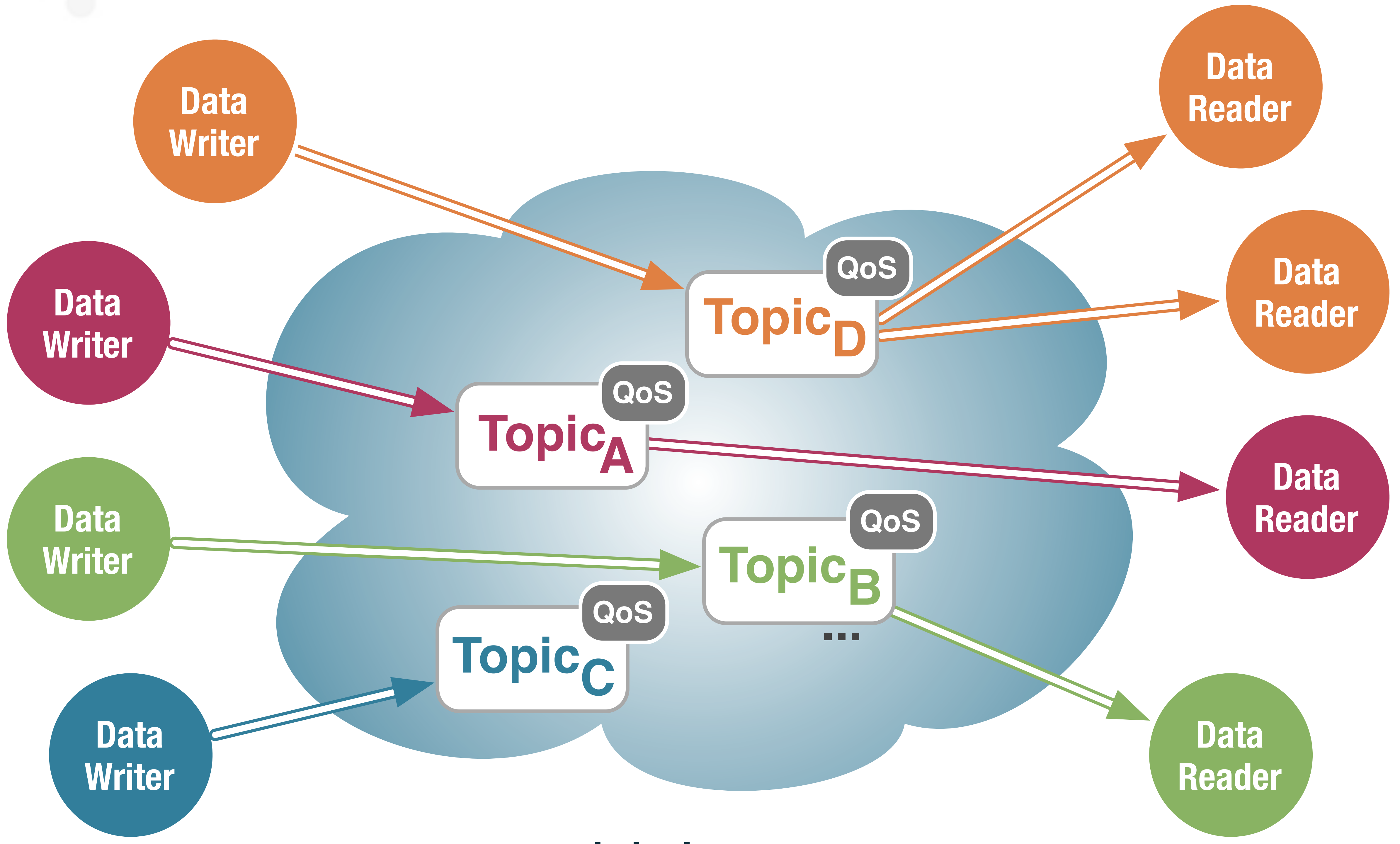
SMART GREEN HOUSES

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

Abstractions

DDS Abstraction

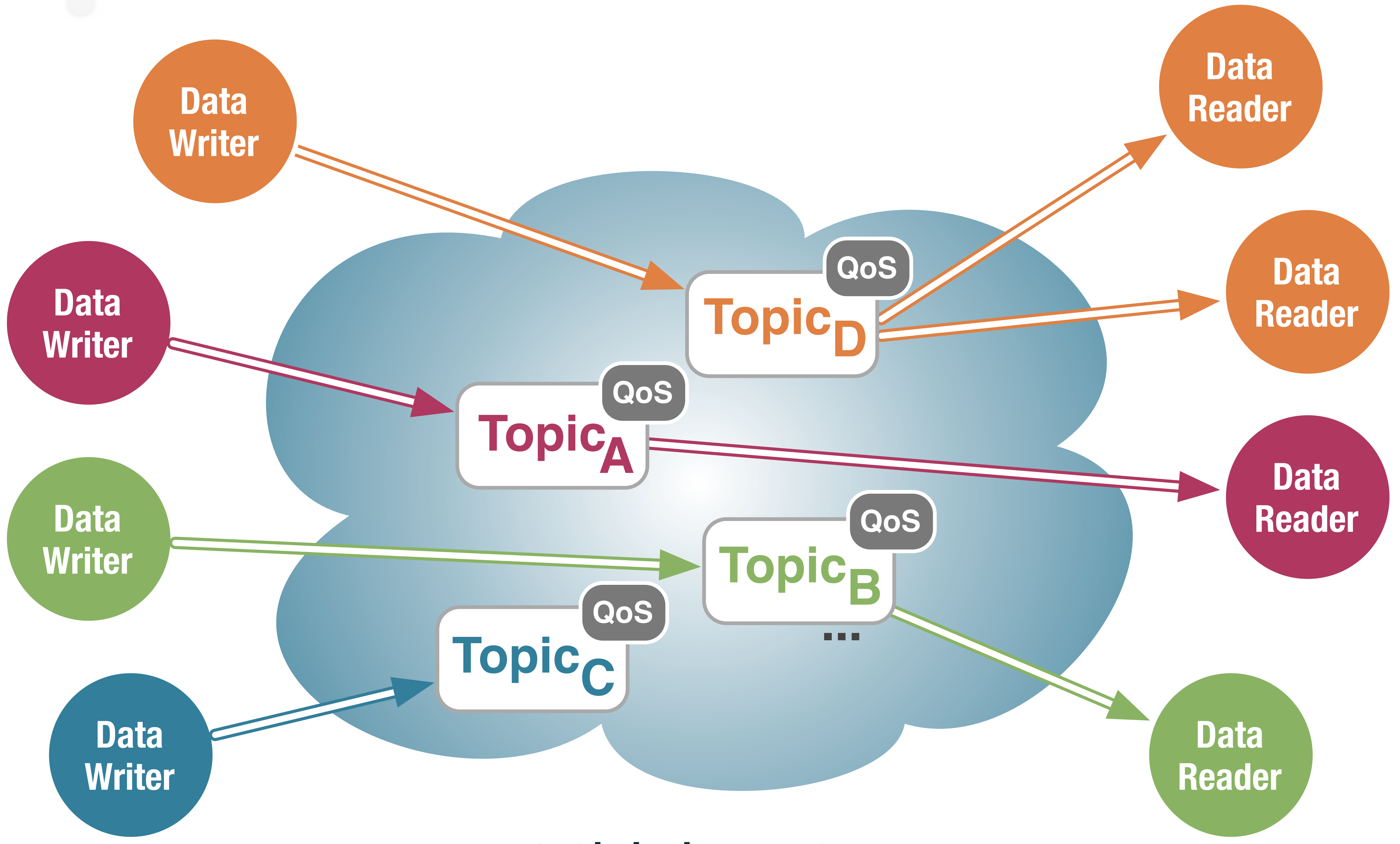
DDS provides applications with a Virtual **Global Data Space** abstraction



DDS Global Data Space

DDS Abstraction

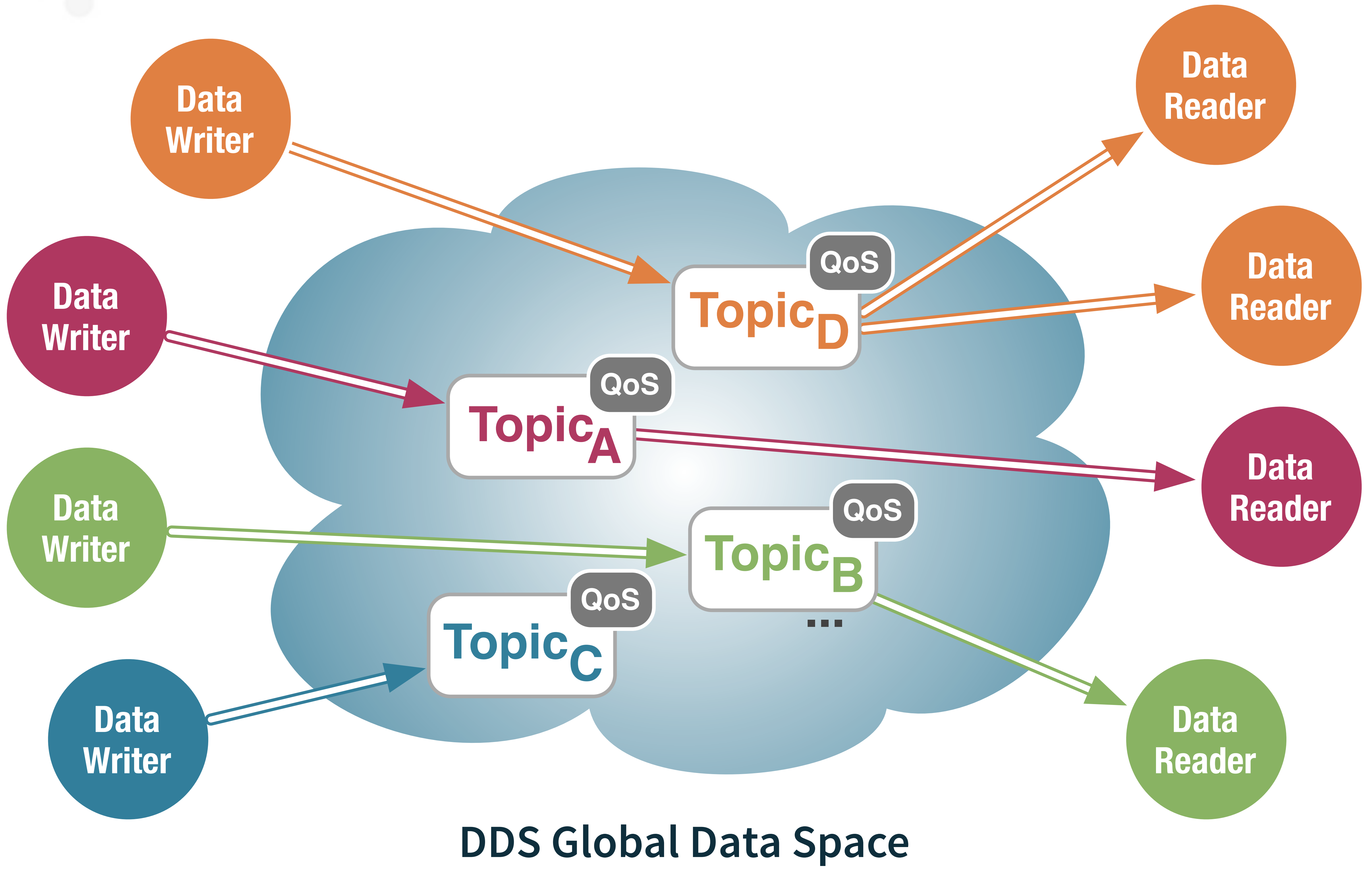
Applications coordinate by **autonomously** and **asynchronously reading** and **writing** data in the Data Space enjoying **spatial** and **temporal decoupling**



DDS Global Data Space

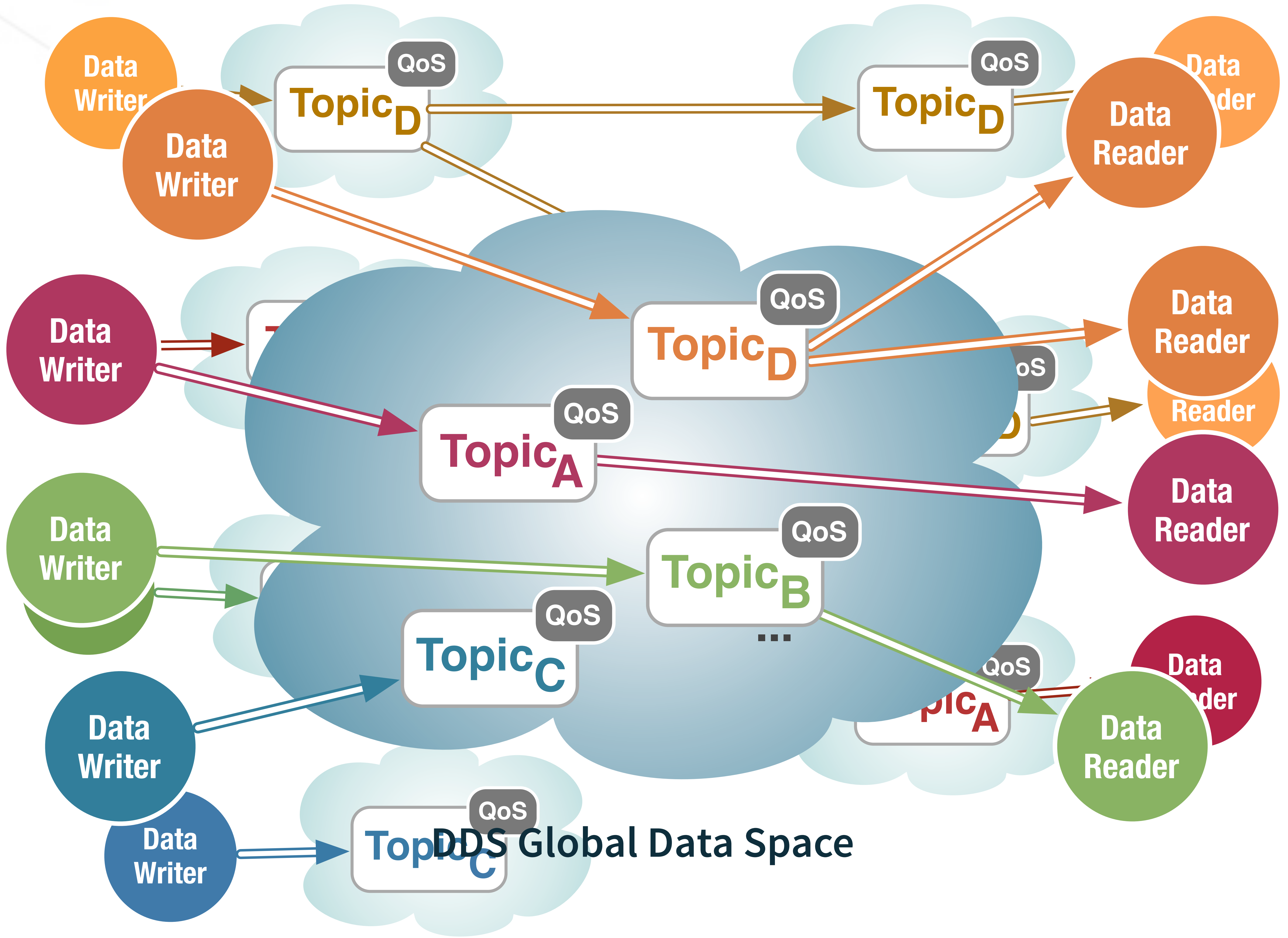
Dynamic Discovery

DDS has built-in dynamic discovery that **automatically matches interest** and **establishes data path isolating applications from network topology** and **connectivity** details



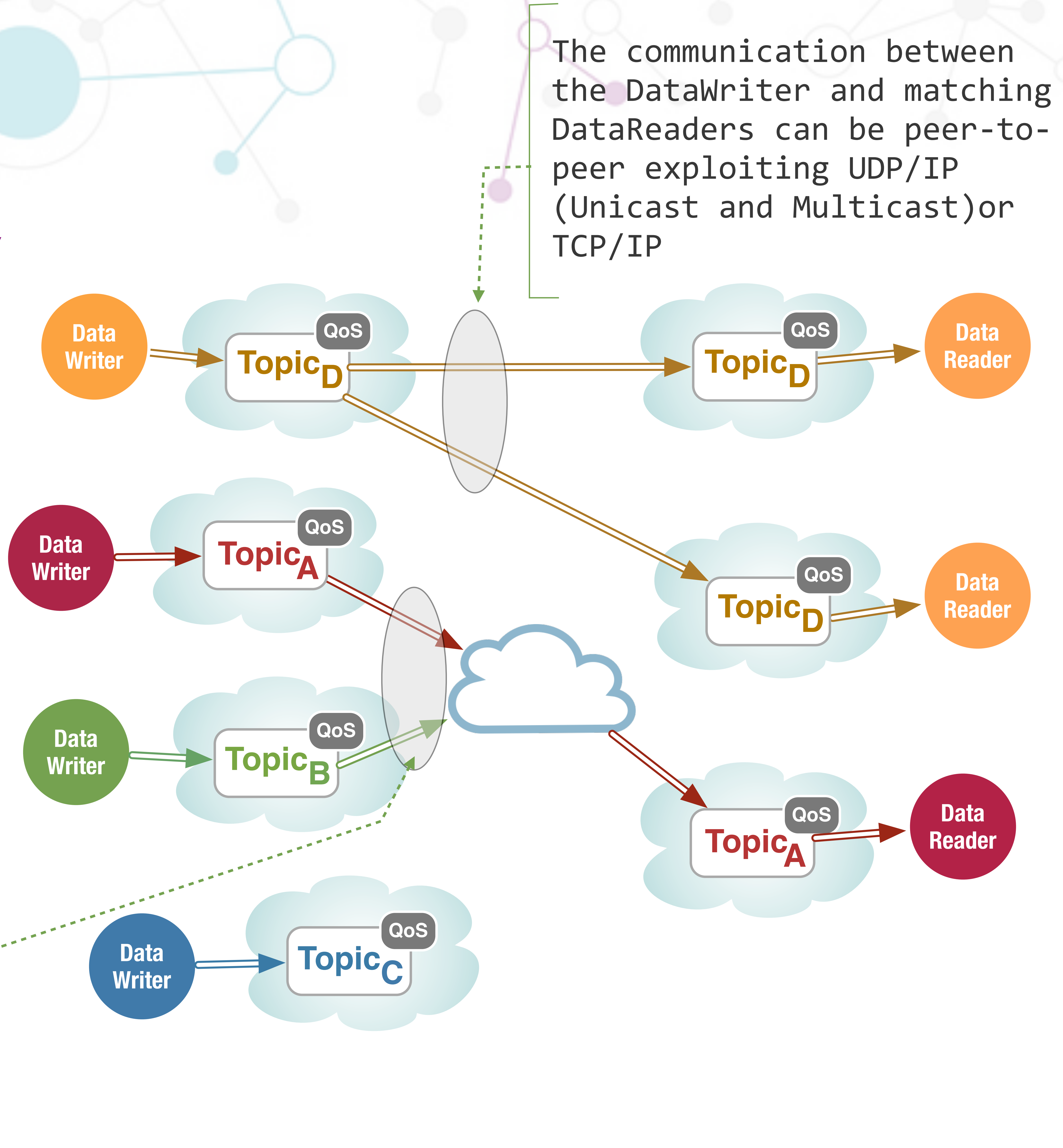
Decentralised Data-Space

DDS global data space implementation is **decentralised** and does not suffer of single point of failure or bottleneck



Adaptive Connectivity

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



The communication between the DataWriter and matching DataReaders can be peer-to-peer exploiting UDP/IP (Unicast and Multicast)or TCP/IP

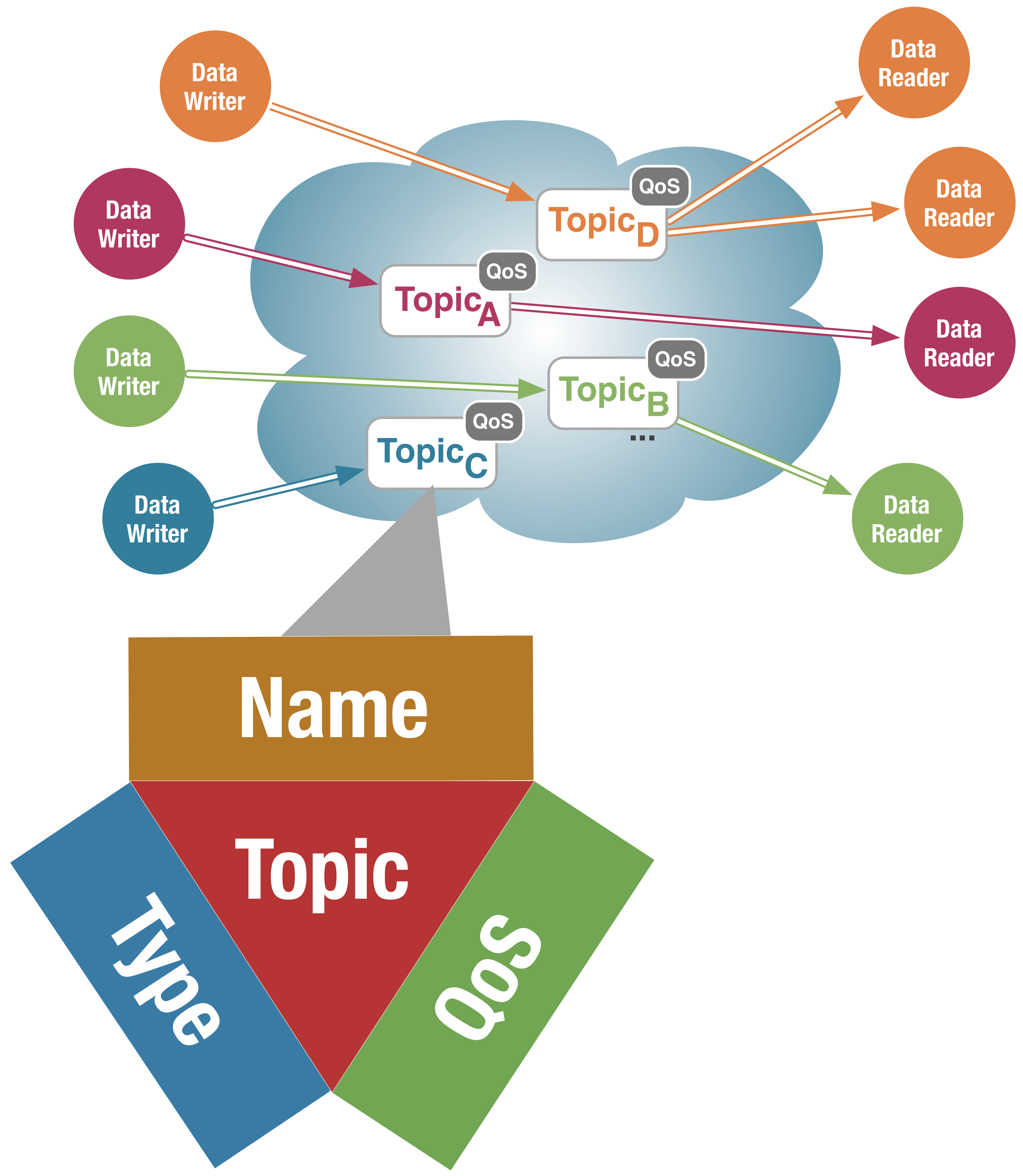
The communication between the DataWriter and matching DataReaders can be “brokered” but still exploiting UDP/IP (Unicast and Multicast)or TCP/IP

Information Organisation

Topics

DDS data streams are defined by means of **Topics**

A **Topic** represented is by means of a <name, type, qos>

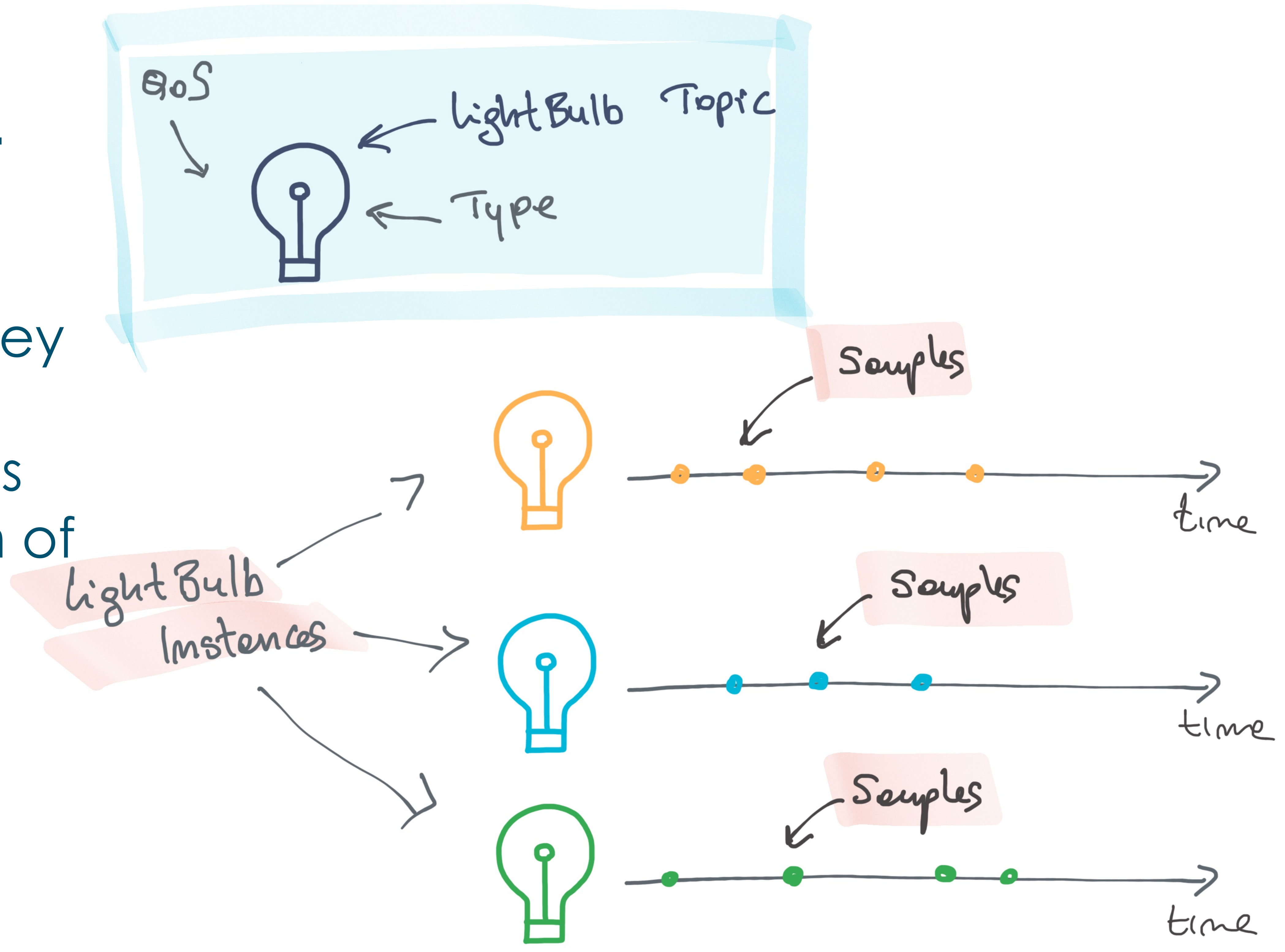


Topic Instances

Topic may **mark some** of their associated type **attributes** as **key-fields**

Each unique key value (tuple of key attributes) identifies a Topic Instance. Each Topic Instance has associated a FIFO ordered stream of samples

DDS provides useful **instance life-cycle management** and samples demultiplexing

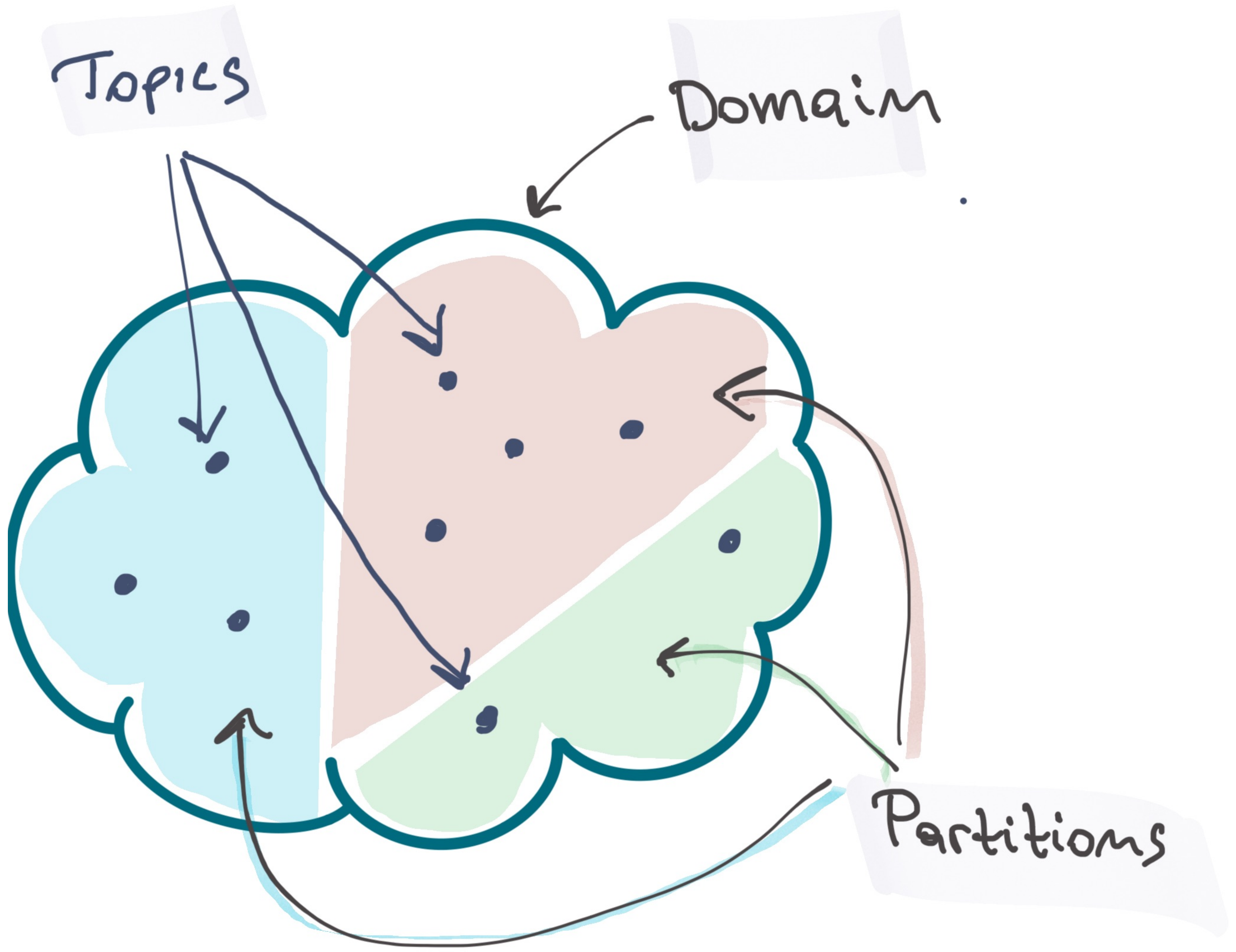


Information Scopes

DDS information lives within a **domain**

A domain can be thought as organised in **partitions**

Samples belonging to a given **Topic Instance** are read/written from/in one or more **partitions**

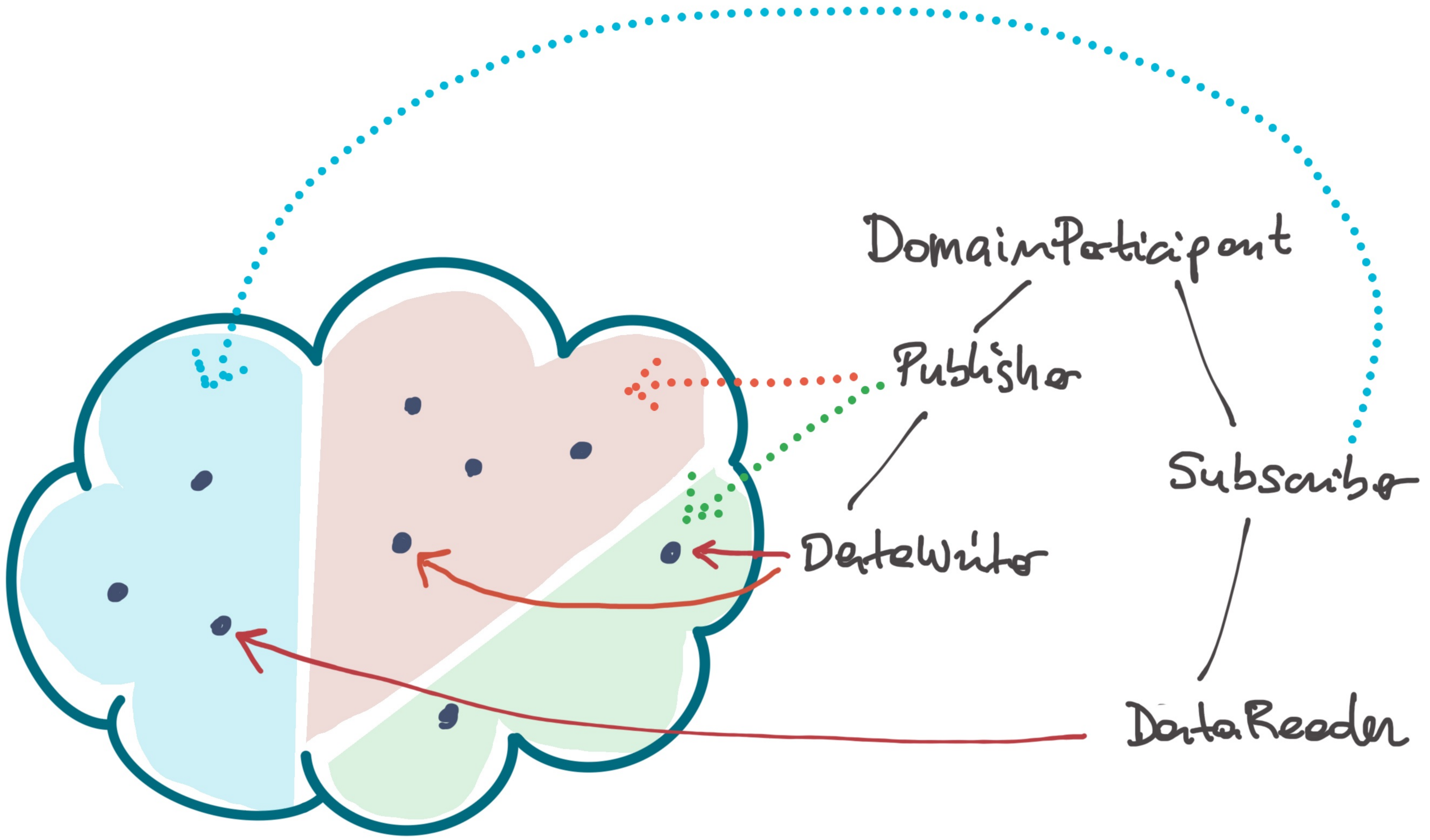


Writing / Reading Data

Domain Participants provide **access** to a DDS **domain**

Publisher and **Subscribers** provide access to **partitions**

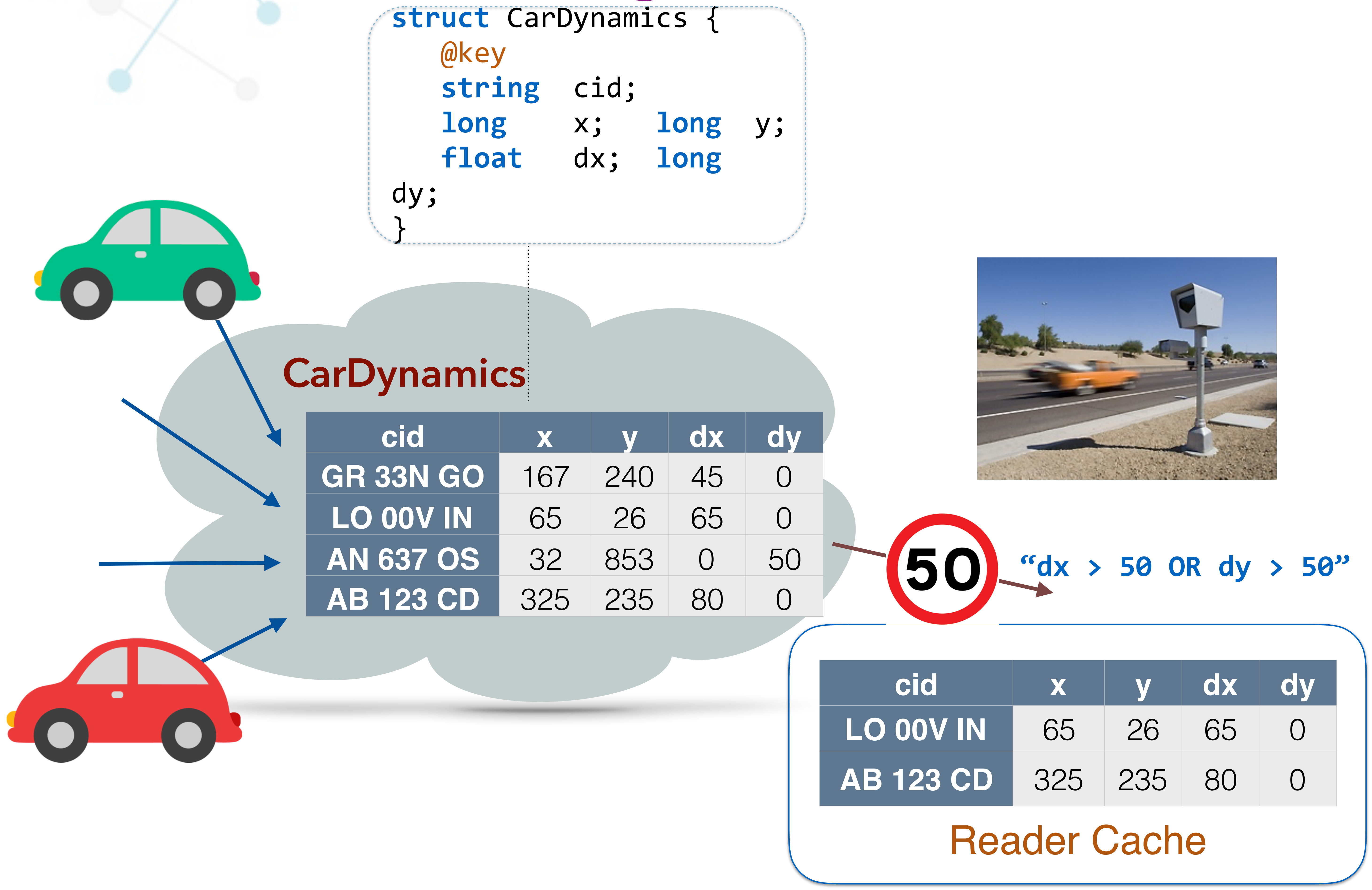
DataWriter/DataReaders write/read data to/from the set of partitions associated with their Publisher/ Subscriber



Content Awareness

Content Filtering

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



```
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”`

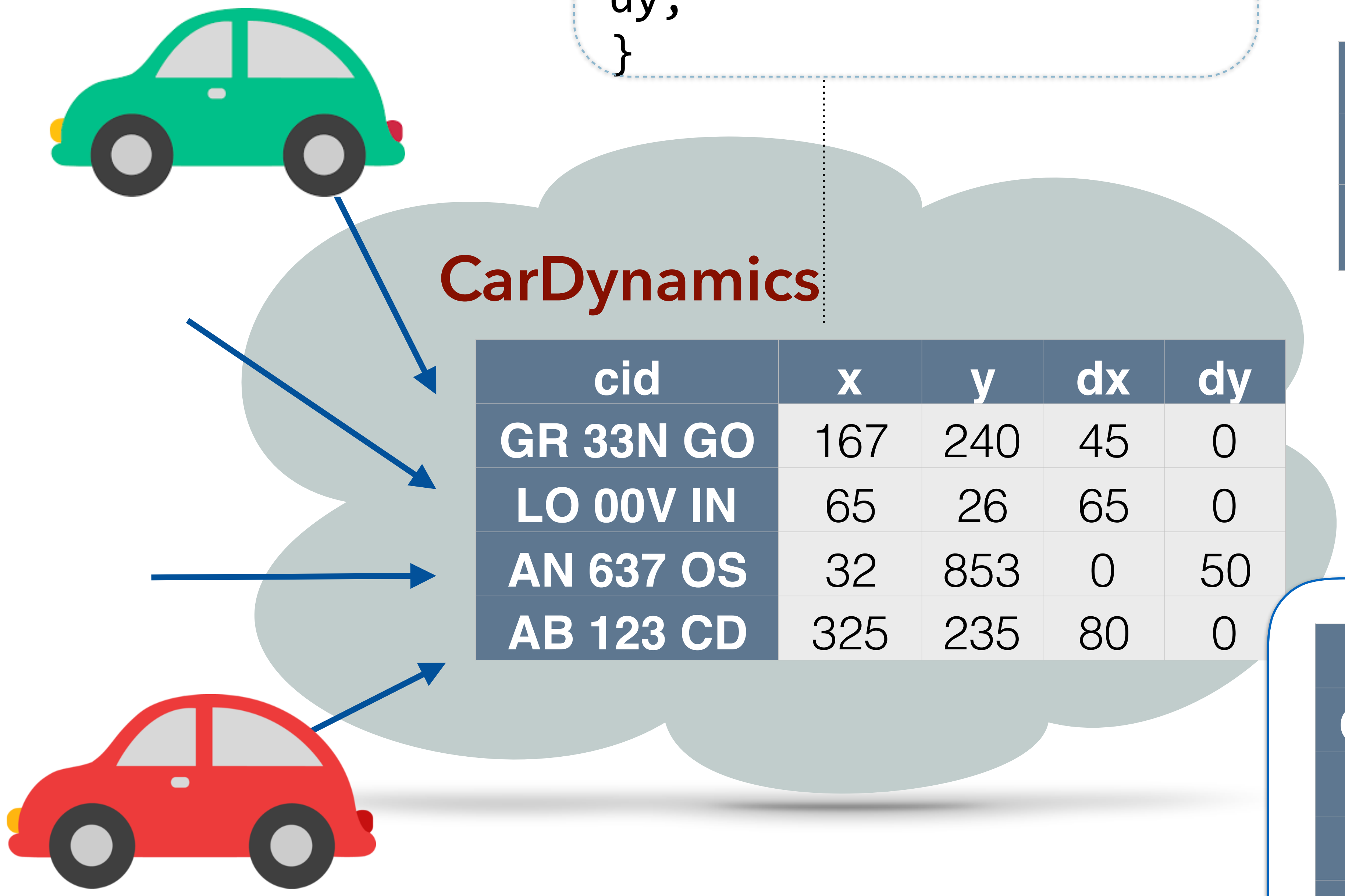
cid	x	y	dx	dy
LO 00V IN	65	26	65	0
AB 123 CD	325	235	80	0

Reader Cache

Queries

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

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



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

50

“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

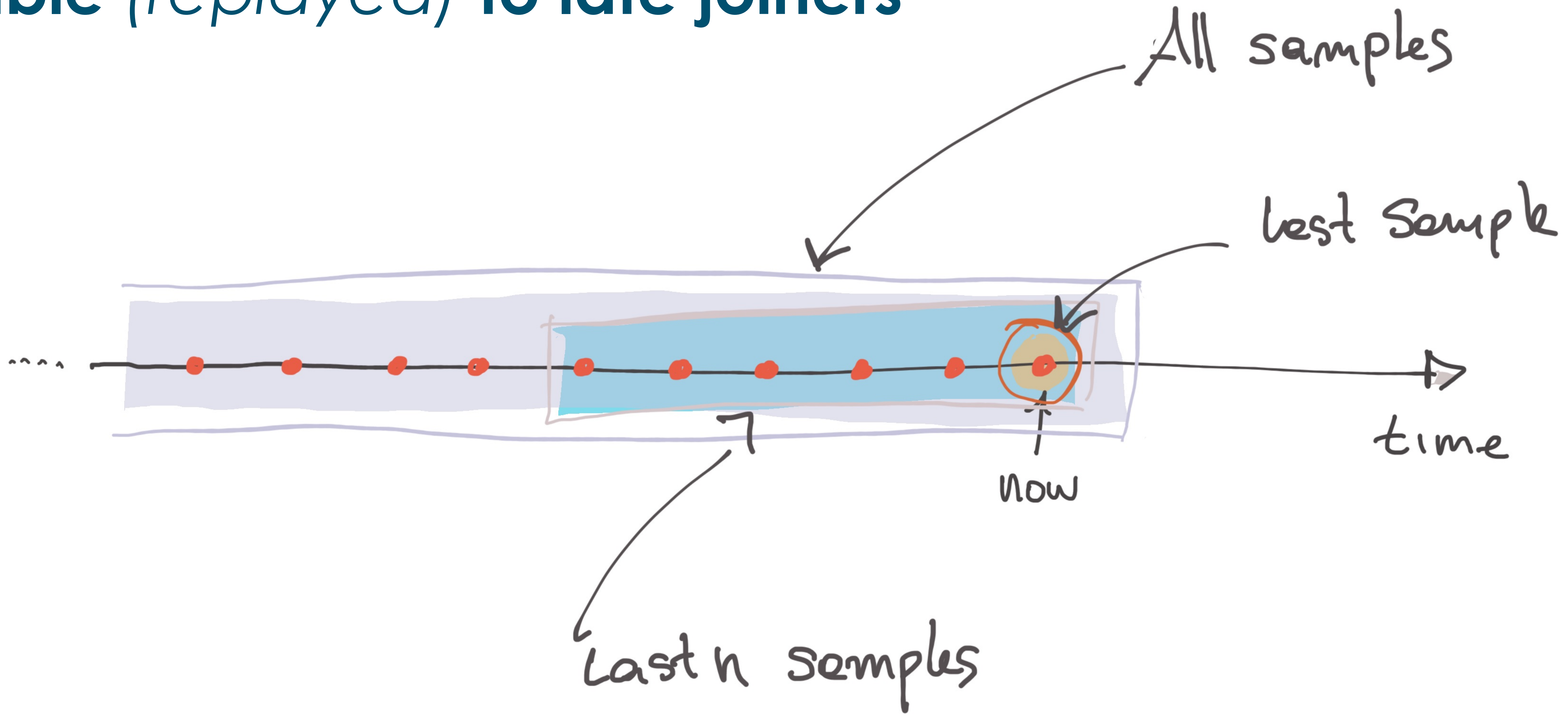
Reader Cache

Stream Durability

Stream Durability

Through QoS settings it is possible to control which subset of the stream data will be **retained** and **made available** *(replayed)* **to late joiners**

DDS can store the last **n** samples (n=1 is a special case) or all the samples written for a topic



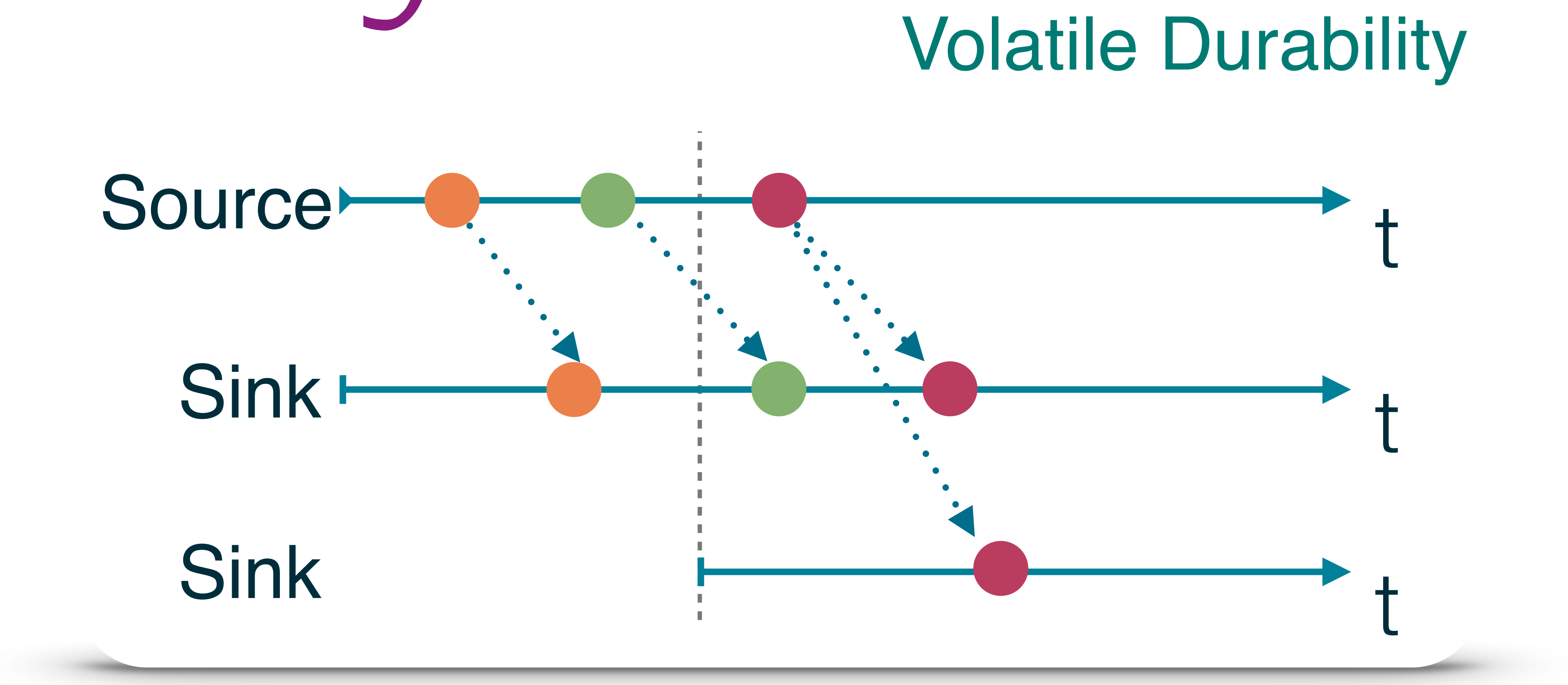
Stream Durability

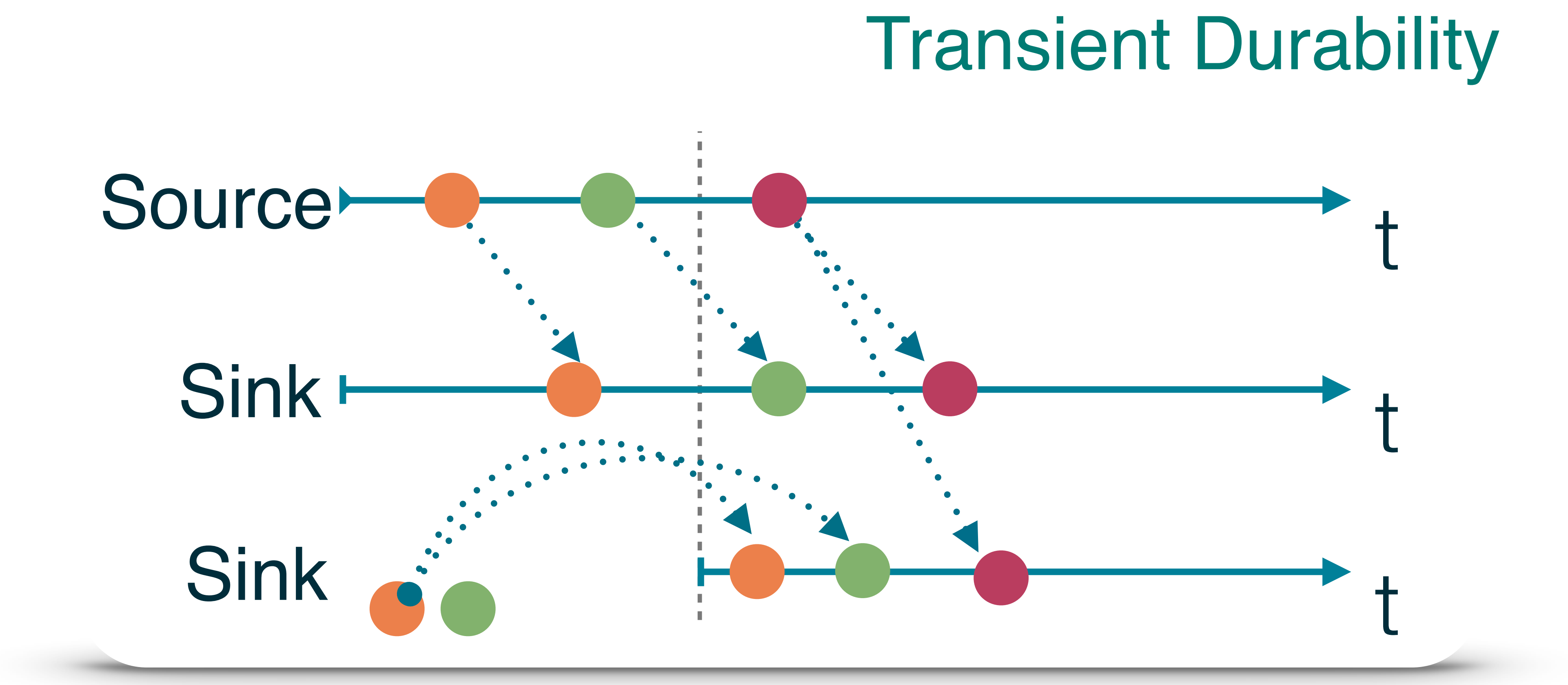
DDS provides three kinds of durability:

Volatile: i.e. no durability

Transient (Local): data is available for late joiners (re-play) as far as the system (data source) is running

Durable: data is available for late joiners (re-play) as far as the system/source is running





Durability Implementation

DDS **Durability is** implemented as **a high-performance distributed service** that provide control over the number of copies of data that should be maintained for availability

Recent data is maintained on memory to reduce access latency

In deployments that support IP multicast, the overhead of durability is practically negligible

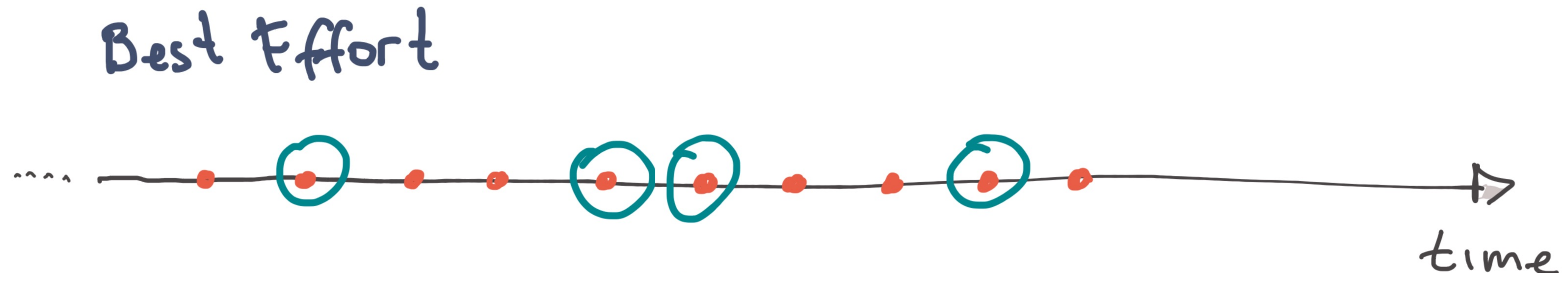
Storage back-ends are pluggable, e.g., File System, RDBMS, etc.

Stream Reliability

Best Effort

DDS will deliver an arbitrary subsequence of the samples written against a Topic Instance

Samples may be dropped because of network loss or because of flow-control

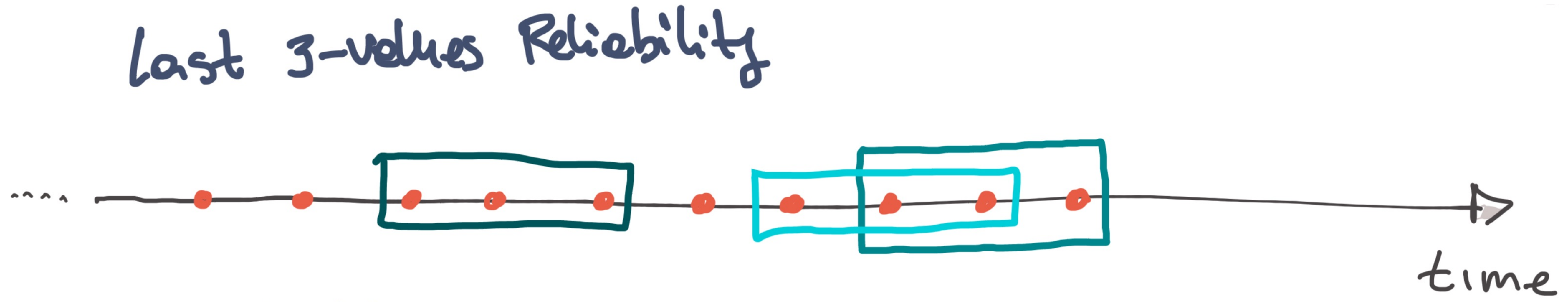


Last n-values Reliability

Under stationary conditions an application is guaranteed to receive the last n-samples written for a Topic Instance

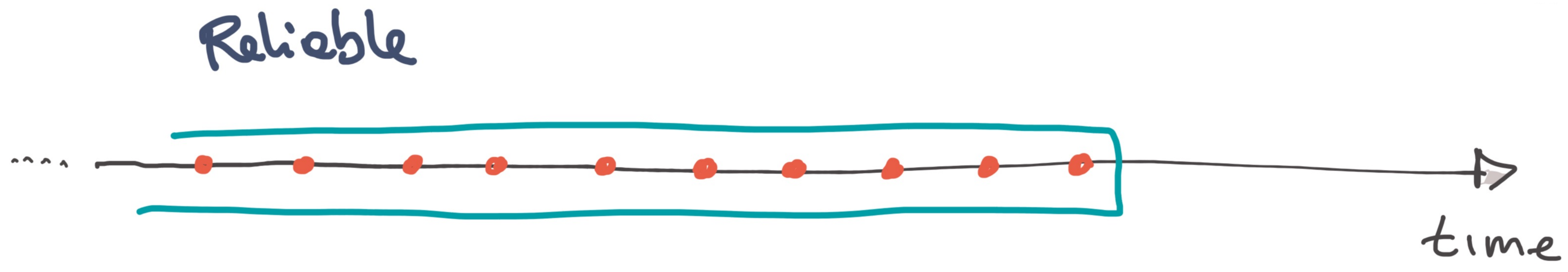
Samples falling outside the history may be dropped at the sending or receiving side for flow/resource control

Notice that this kind of reliability behaves as a circuit breaker for slow consumers



Reliable

All samples written against a Topic Instance are delivered. Since from a theoretical perspective reliability in asynchronous systems either violate progress or requires infinite memory, DDS provides QoS to control both resources as well as blocking time



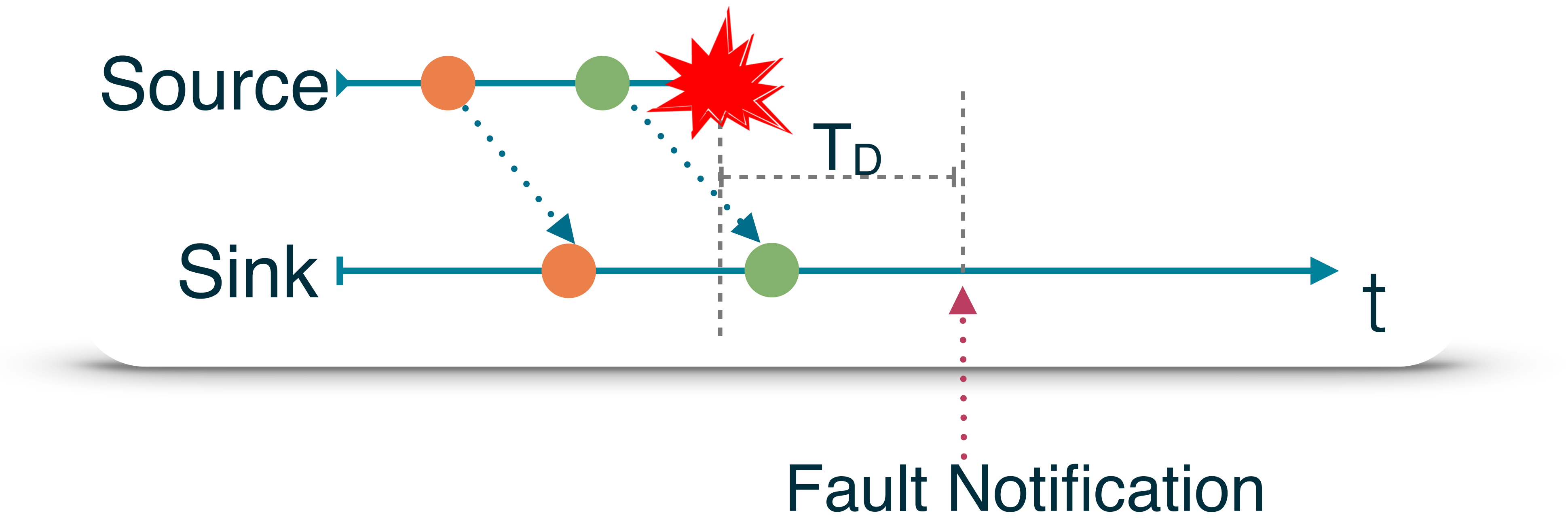
Fault-Tolerance

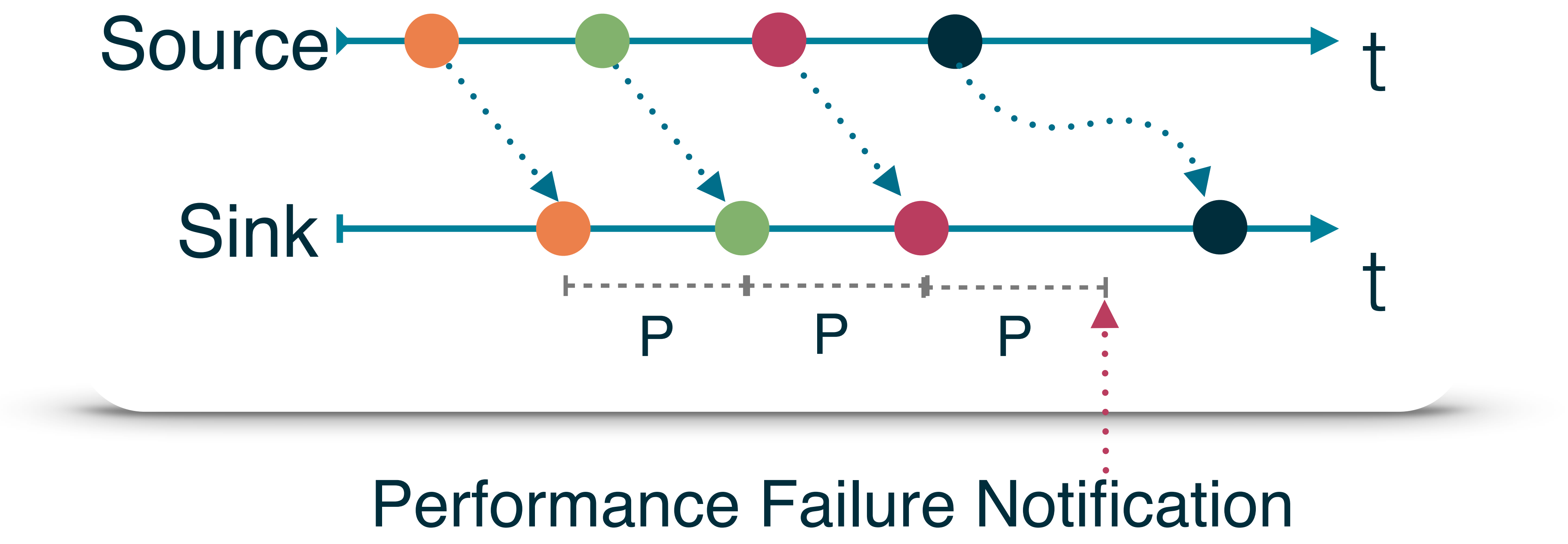
Failure Detection

DDS provides mechanism for detecting traditional faults as well as performance failures

The Fault-Detection mechanism is controlled by means of the DDS Liveliness policy

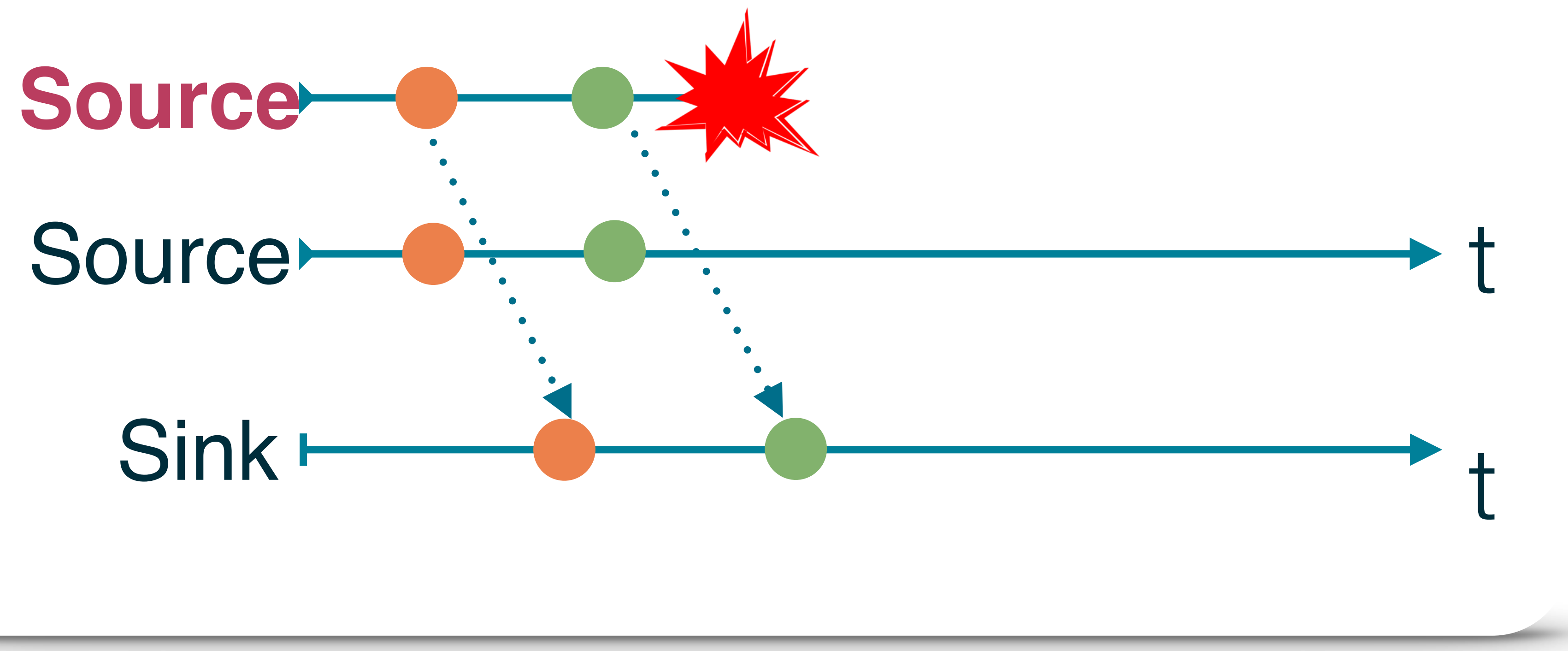
Performance Failures can be detected using the Deadline Policy which allows to receive notification when data is not received within the expected delays





Fault-Masking

DDS provides a built-in fault-masking mechanism that allow to replicate **Sources** and transparently switch over when a failure occurs
At any point in time the "active" source is the one with the highest strength. Where the strength is an integer parameter controller by the user



A Breath of Code

WRITING DATA IN C++

```cpp
#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

```
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

```
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)
    }
  }
}
```

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

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

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

```
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
```

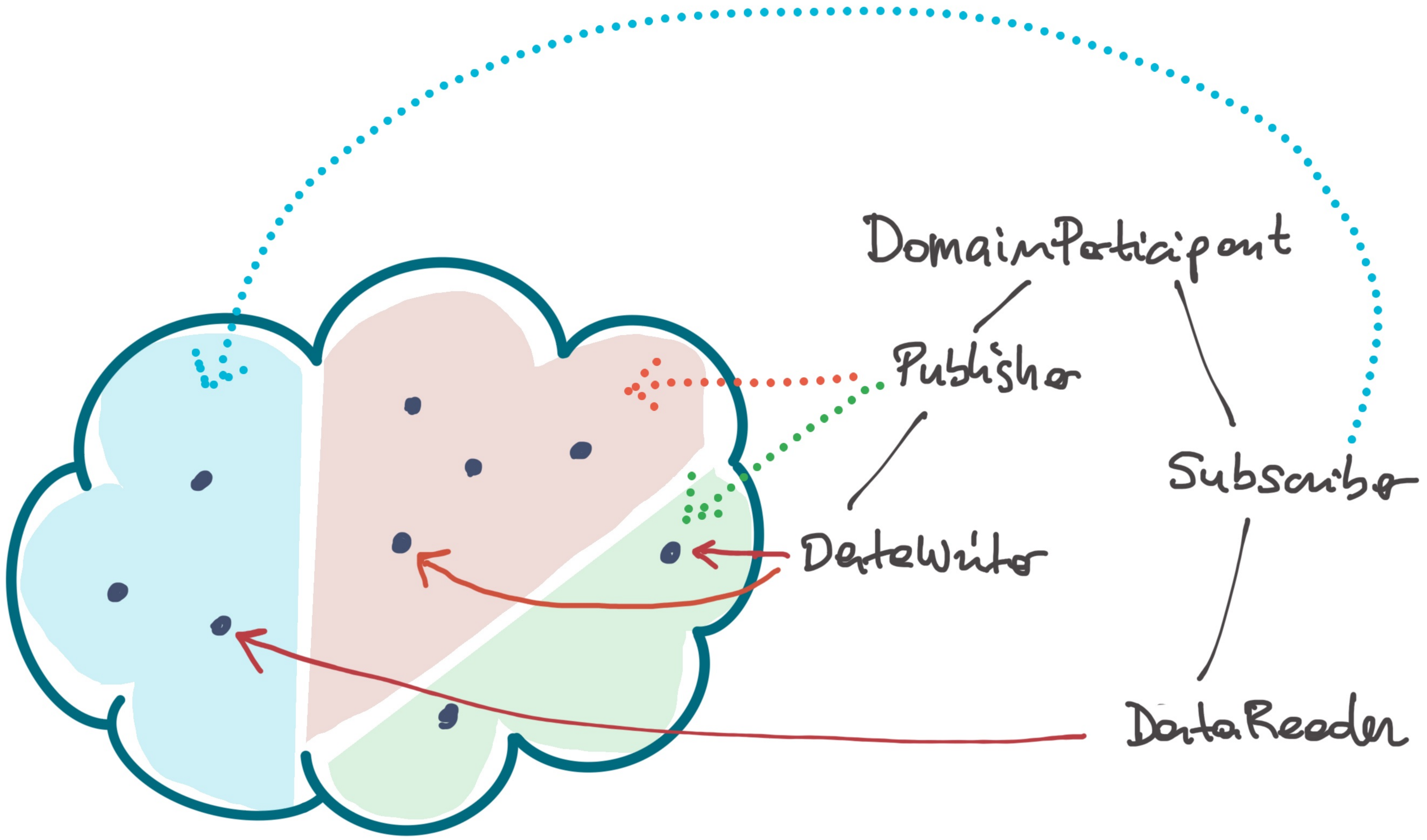
Security

DDS Security Goals

Provide a **data-centric security** that allows to control access to the DDS Global Data Space

Ensure that the security solution is multicast-friendly

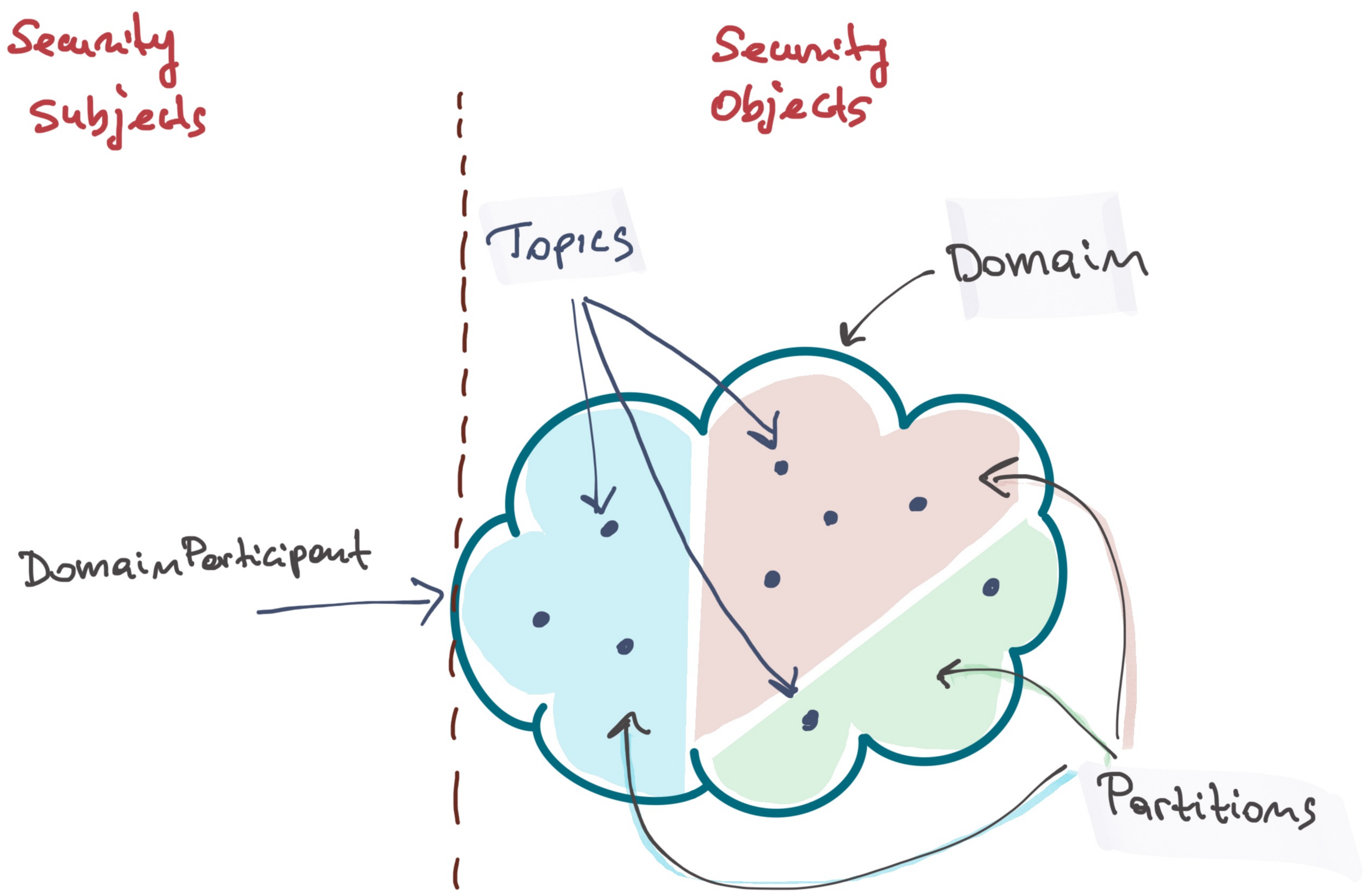
Design for extensibility and customisability



DDS Security Model

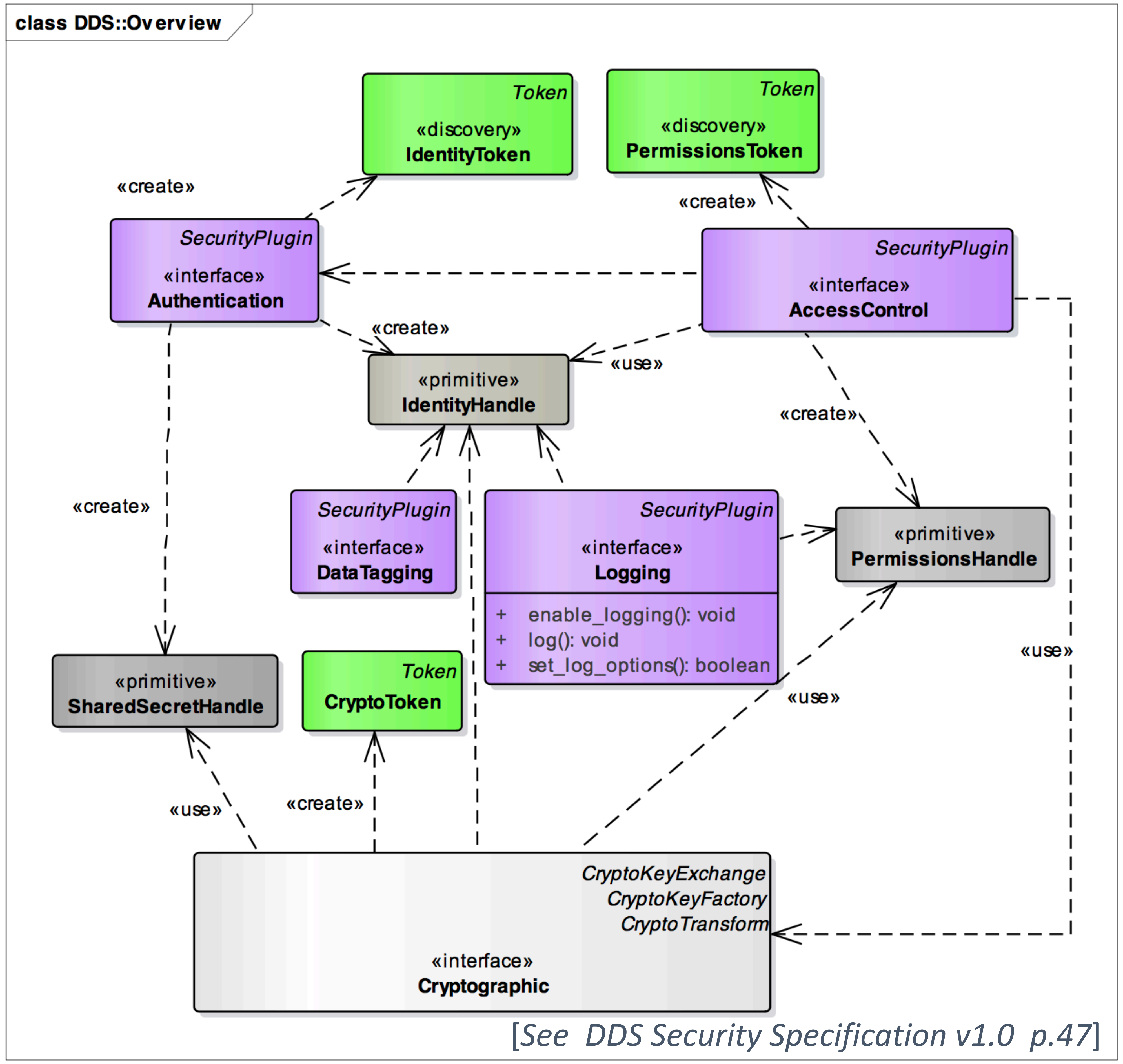
The DDS Security provides

- **Confidentiality** of the data samples
- **Integrity** of the data samples and the messages that contain them
- **Authentication** of DDS writers and readers
- **Authorisation** of DDS writers and readers
- **Non-repudiation** of data



Plug-in Architecture

The DDS Security standard has a modular and **plug-in architecture** that allows for pluggable **Authentication**, **Access Control**, **Logging**, **Cryptography** and **Data Tagging**



[See DDS Security Specification v1.0 p.47]

Default Plugins

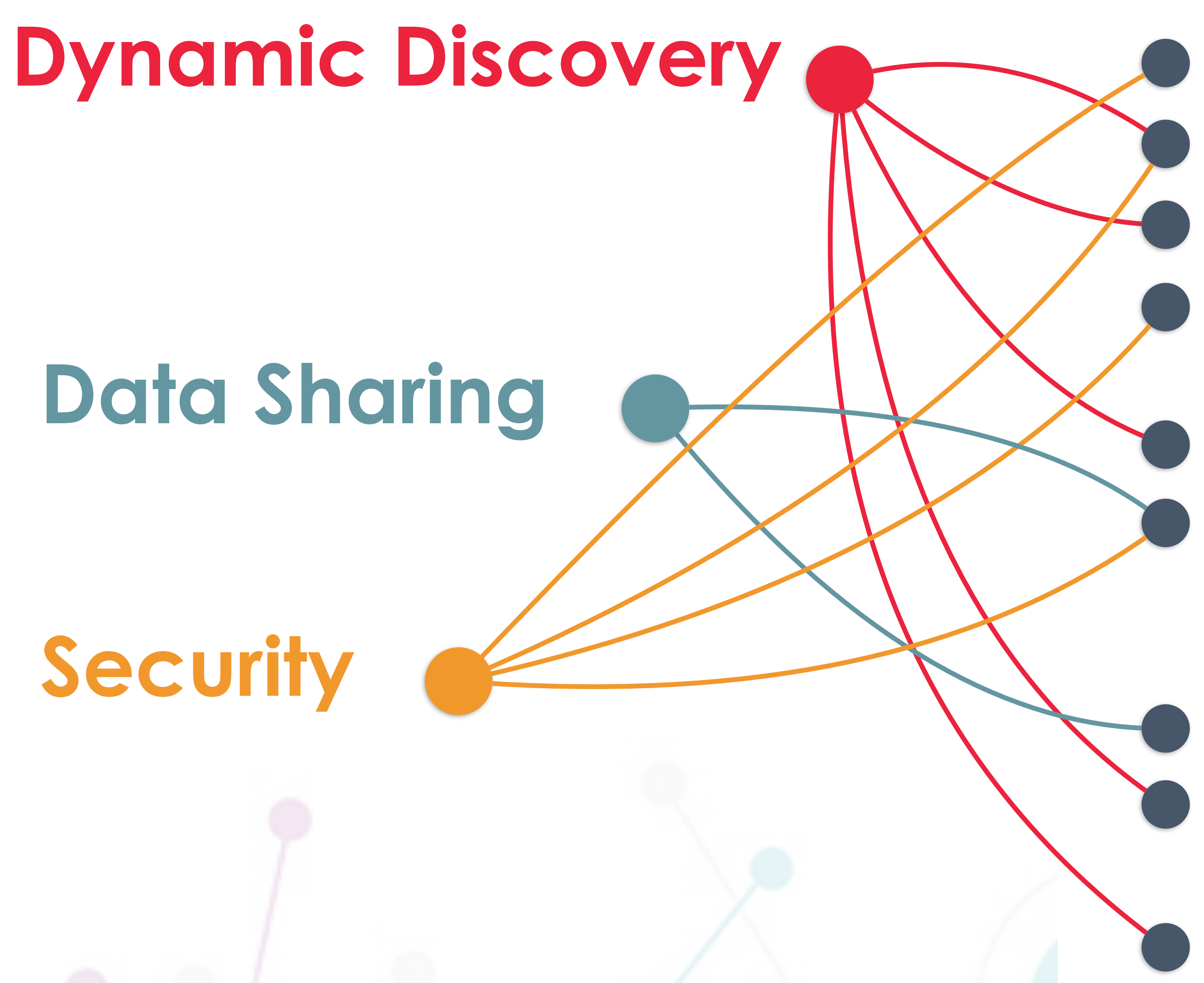
	Name	Description
Authentication	DDS:Auth:PKI-DH	*Uses PKI with a pre- configured shared Certificate Authority. RSA or DSA and Diffie- Hellman for authentication and key exchange.*
Access Control	DDS:Access:Permissions	*Permissions document signed by shared Certificate Authority*
Cryptography	DDS:Crypto:AES-GCM-GMAC	*AES-GCM (AES using Galois Counter Mode) for encryption. AES-GMAC for message authentication*
Data Tagging	DDS:Tagging:DDS_Discovery	*Send Tags via endpoint discovery*
Logging	DDS:Logging:DDS_LogTopic	*Logs security events to a dedicated DDS Log Topic*

DDS & oneM2M

DDS & oneM2M

DDS Features

oneM2M Common Services *(most important core svcs)*



Identification	Identity management of the entities (AEs, CSEs, NSEs, …)
Registration	CSE-CSE Registration, AE-CSE Registration, …
Discovery	Discovery of entities and information/resources
Security	confidentiality, integrity, availability, credential/key management, encryption, privacy, authentication, authorization
Group Management	Management of groups, support of bulk operations and access
Device Management	Firmware updates, configuration settings, topology management, Software installation, logging, monitoring, diagnostics, Reuse of existing DM technologies
Subscribe / Notify	Support of event-related notifications (change of values)
Network Exposure	Abstraction of the underlying network interface, (eg. usage of remote device triggering, location services, …)
Comm. Management	Selection of communications channels, scheduling, Store-and-forward, reachability status awareness
Location	Manages and provides location information services

Summing Up

Concluding Remarks

DDS provides an extremely powerful set of abstractions and mechanism for data sharing in large scale distributed systems

DDS appears to be a natural fit for oneM2M and we would be delighted to help defining the DDS binding for oneM2M