DDS

Data Distribution Service

Gerardo Pardo-Castellote, Ph.D.

Real-Time Innovations, Inc.

DDS Standard

Data Distribution Service for Real-Time Systems

- Adopted in June 2003
- Finalized in June 2004
- Joint submission (RTI, THALES, MITRE, OIS)
- API specification for Data-Centric Publish-Subscribe communication for distributed real-time systems.

RTI's role

- Member of OMG since 2000
- Co-authors of the original DDS RFP
- Co-authors of the DDS specification adopted in June 2003
- Chair of the DDS Finalization Task Force completed March 2004
- Chair of the DDS Revision Task Force
- Providers of a COTS implementation of the specification (NDDS.4.0)

OMG Middleware standards

CORBA

Distributed object

- Client/server
- Remote method calls
- Reliable transport

Best for

- Remote command processing
- File transfer
- Synchronous transactions

DDS

Distributed data

- Publish/subscribe
- Multicast data
- Configurable QoS

Best for

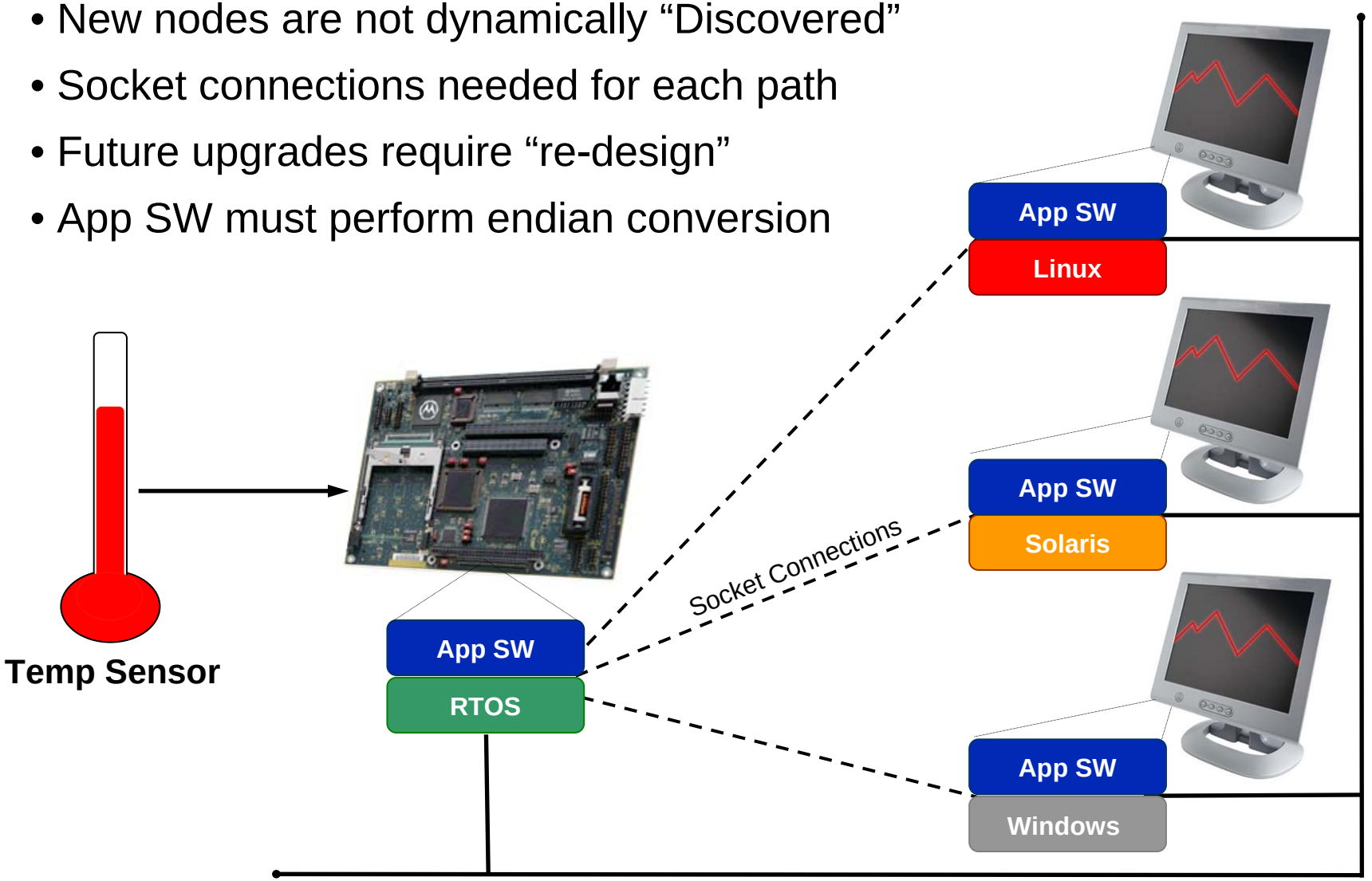
- Quick dissemination to many nodes
- Dynamic nets
- Flexible delivery requirements

DDS and CORBA address different needs

Complex systems often need both…

More Complex Distributed Application

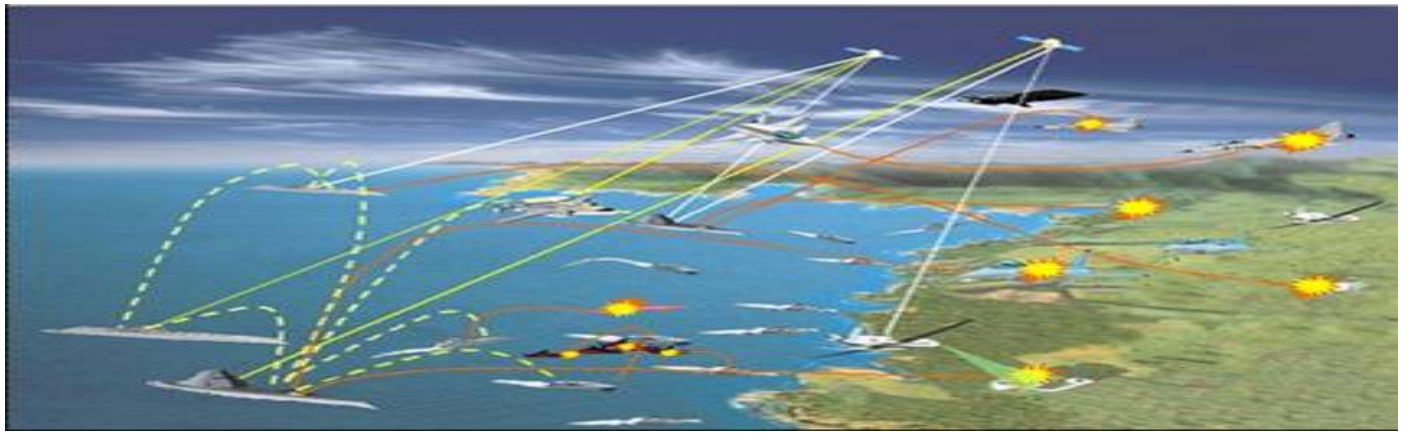
- New nodes are not dynamically “Discovered”
- Socket connections needed for each path
- Future upgrades require “re-design”
- App SW must perform endian conversion



The net-centric vision

- *Vision for "net-centric applications"*
- *Total access to information for real-time applications*
- *This vision is enabled by the internet and related network technologies*
- *Challenge:*

  "Provide the right information at the right place at the right time… no matter what."



Challenges: Factors driving DDS

Need for speed

- Large networks, multicast
- High data rates
- Natural asynchrony
- Tight latency requirements
- Continuously-refreshed data

Complex data flows

- Controlled QoS: rates, reliability, bandwidth
- Per-node, or per-stream differences
- Varied transports (incl. Unreliable e.g. wireless)

Dynamic configurations

- Fast location transparency

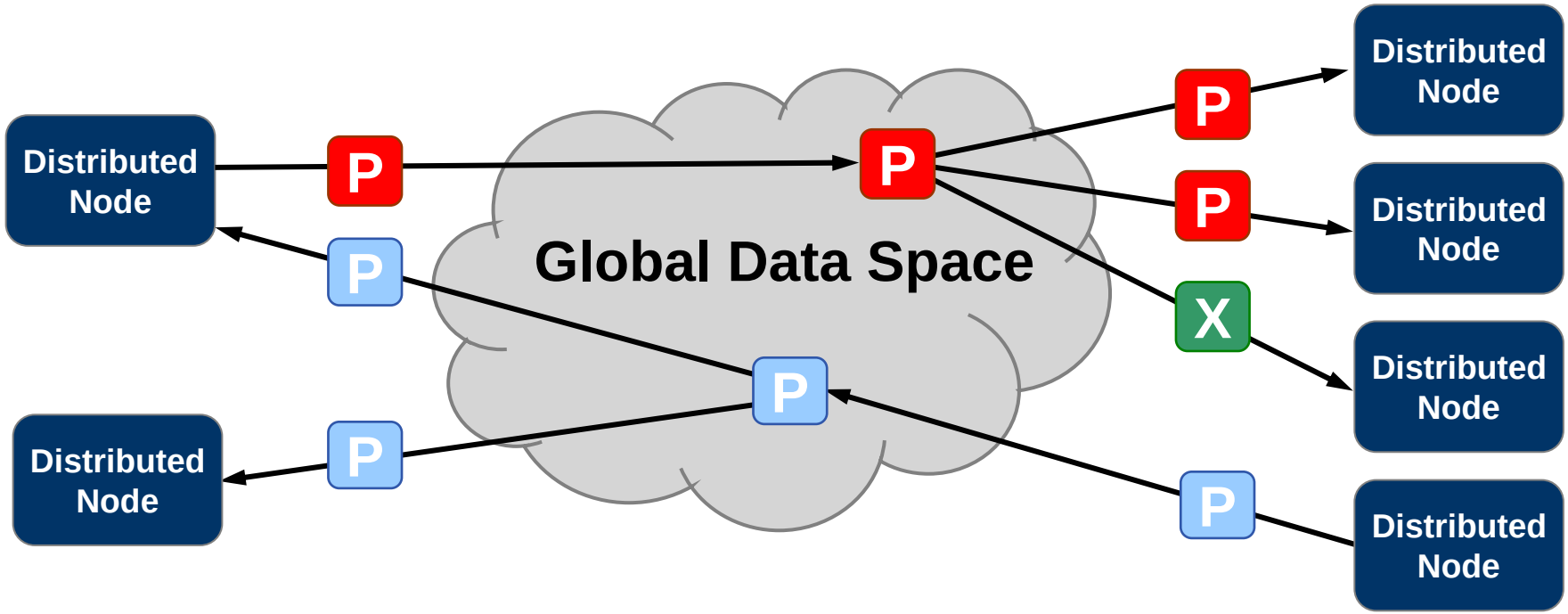
Fault tolerance

- No single-points of failure
- Transparent failover

DDS

Provides a "Global Data Space" that is accessible to all interested applications.

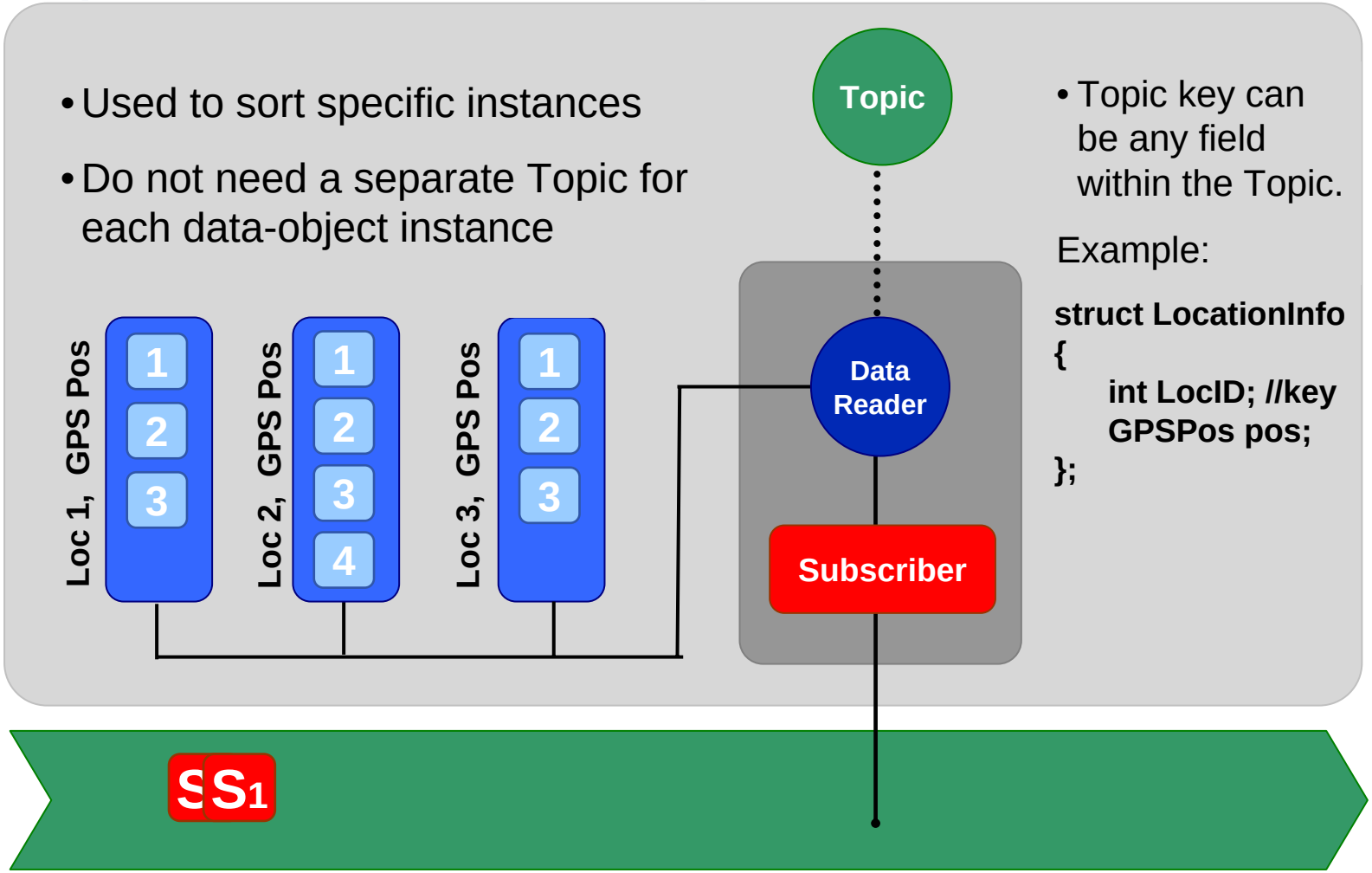
- Data objects addressed by ***Topic*** and ***Key***
- Subscriptions are ***decoupled*** from Publications
- Contracts established by means of ***QoS***
- Automatic ***discovery*** and configuration



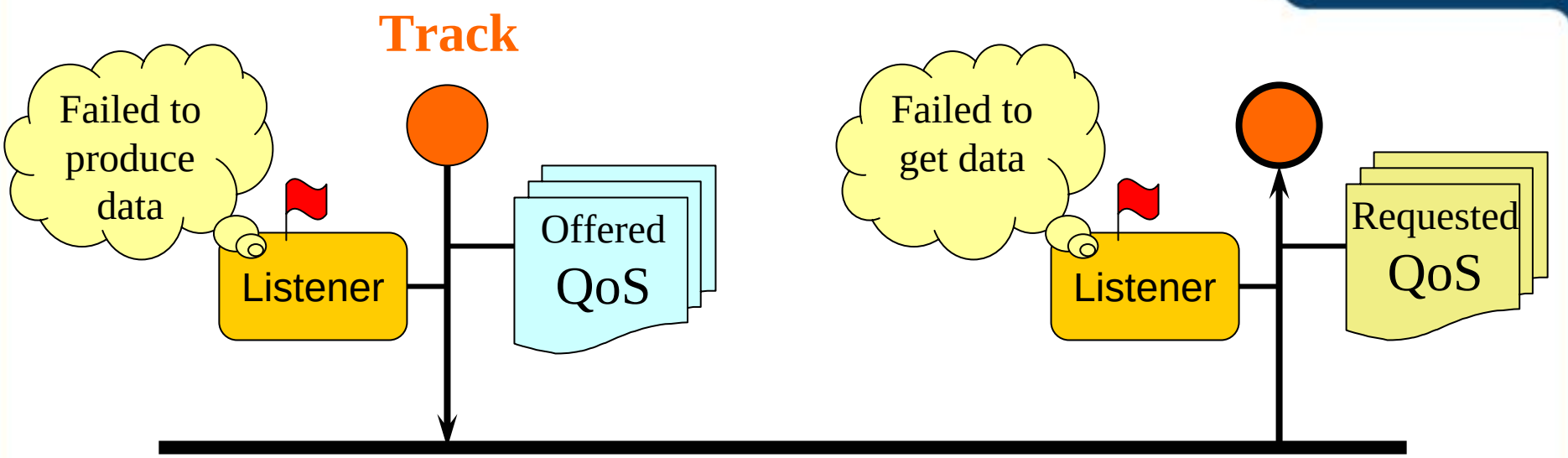
Data object addressing: Keys

Address in Global Data Space = (Topic, Key)

Multiple instances of the same topic



DDS communications model



Publisher declares information it has and specifies the Topic

- … and the offered QoS contract
- … and an associated listener to be alerted of any significant status changes

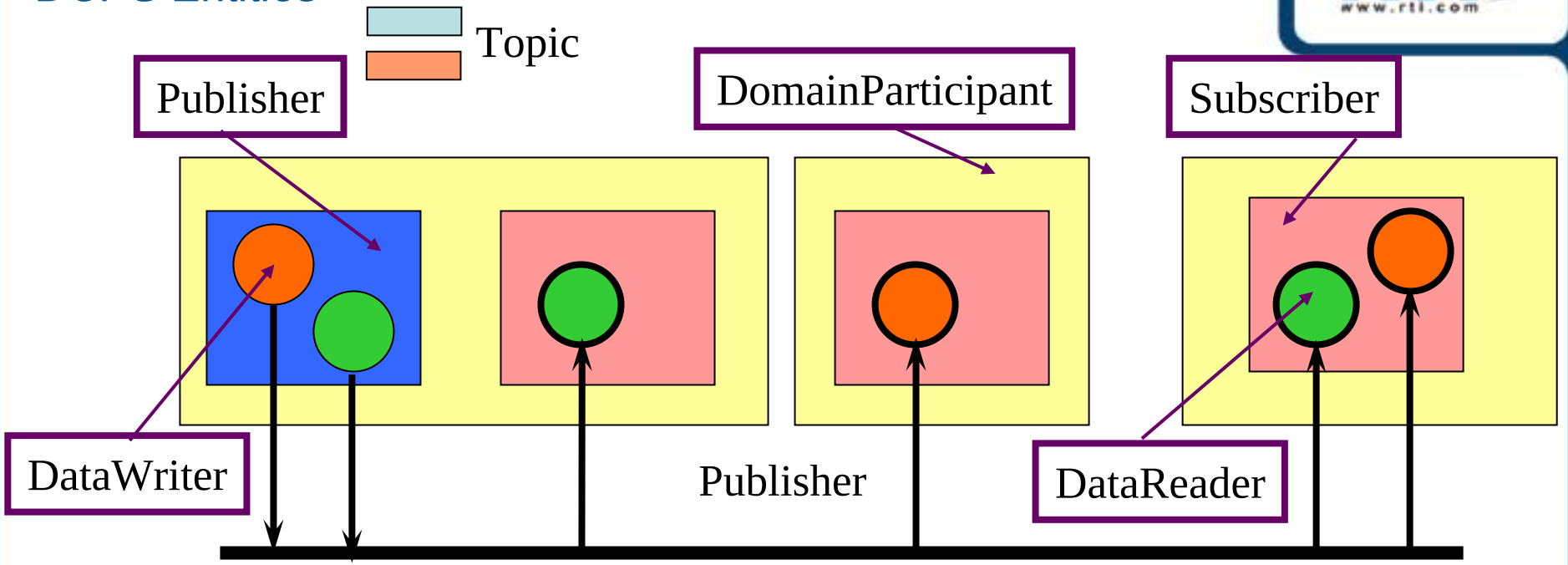
Subscriber declares information it wants and specifies the Topic

- … and the requested QoS contract
- … and an associated listener to be alerted of any significant status changes

DDS automatically discovers publishers and subscribers

- DDS ensures QoS matching and alerts of inconsistencies

DCPS Entities



DomainParticipant ~ ***Represents participation of the application in the communication collective***

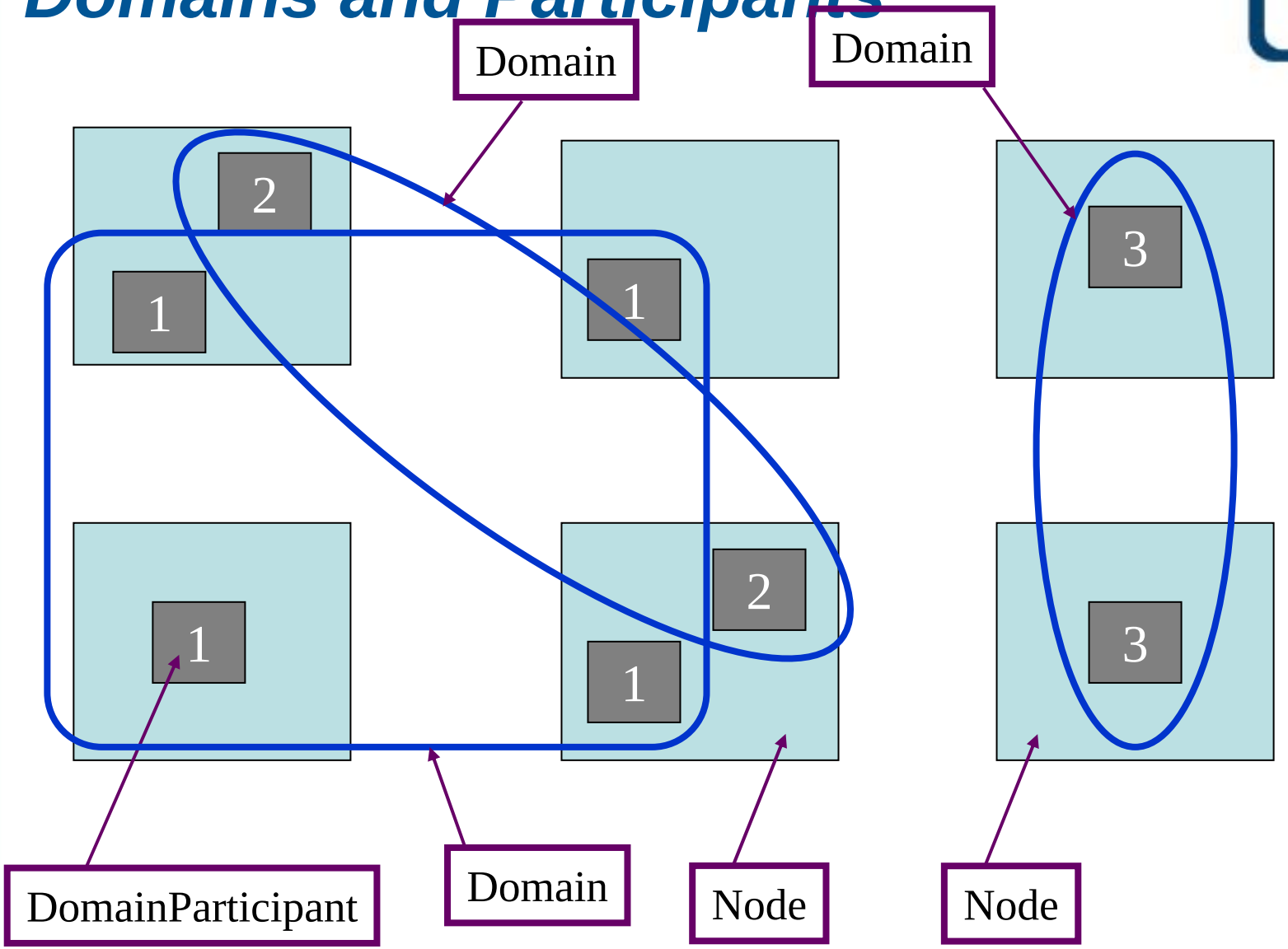
DataWriter ~ ***Accessor to write typed data on a particular*** *Topic*

Publisher ~ ***Aggregation of DataWriter objects. Responsible for disseminating information.***

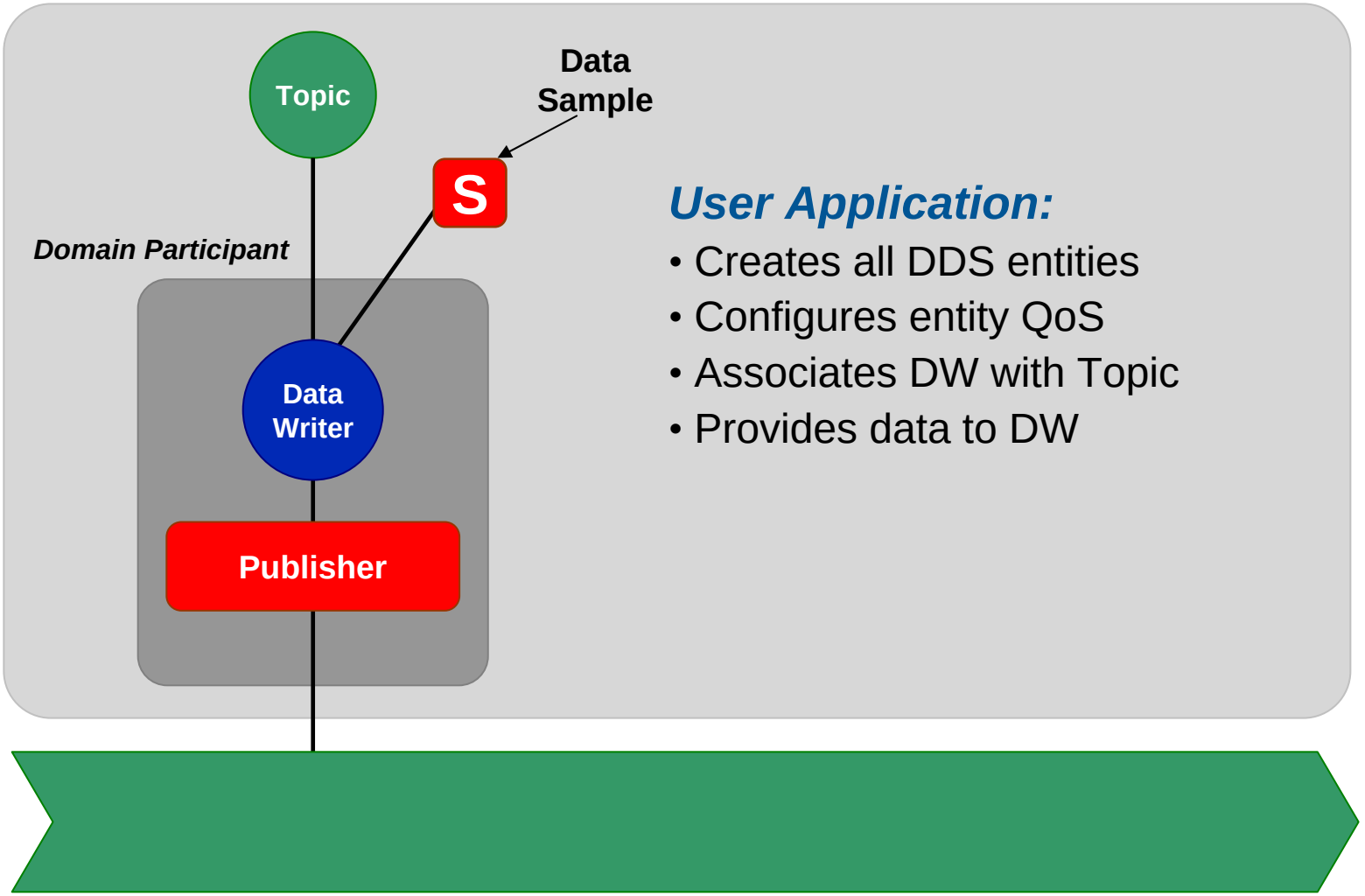
DataReader ~ ***Accessor to read typed data regarding a specific*** *Topic*

Subscriber ~ ***Aggregation of DataReader objects. Responsible for receiving information***

Domains and Participants



DDS Publication



User Application:

- Creates all DDS entities
- Configures entity QoS
- Associates DW with Topic
- Provides data to DW

Example: Publication

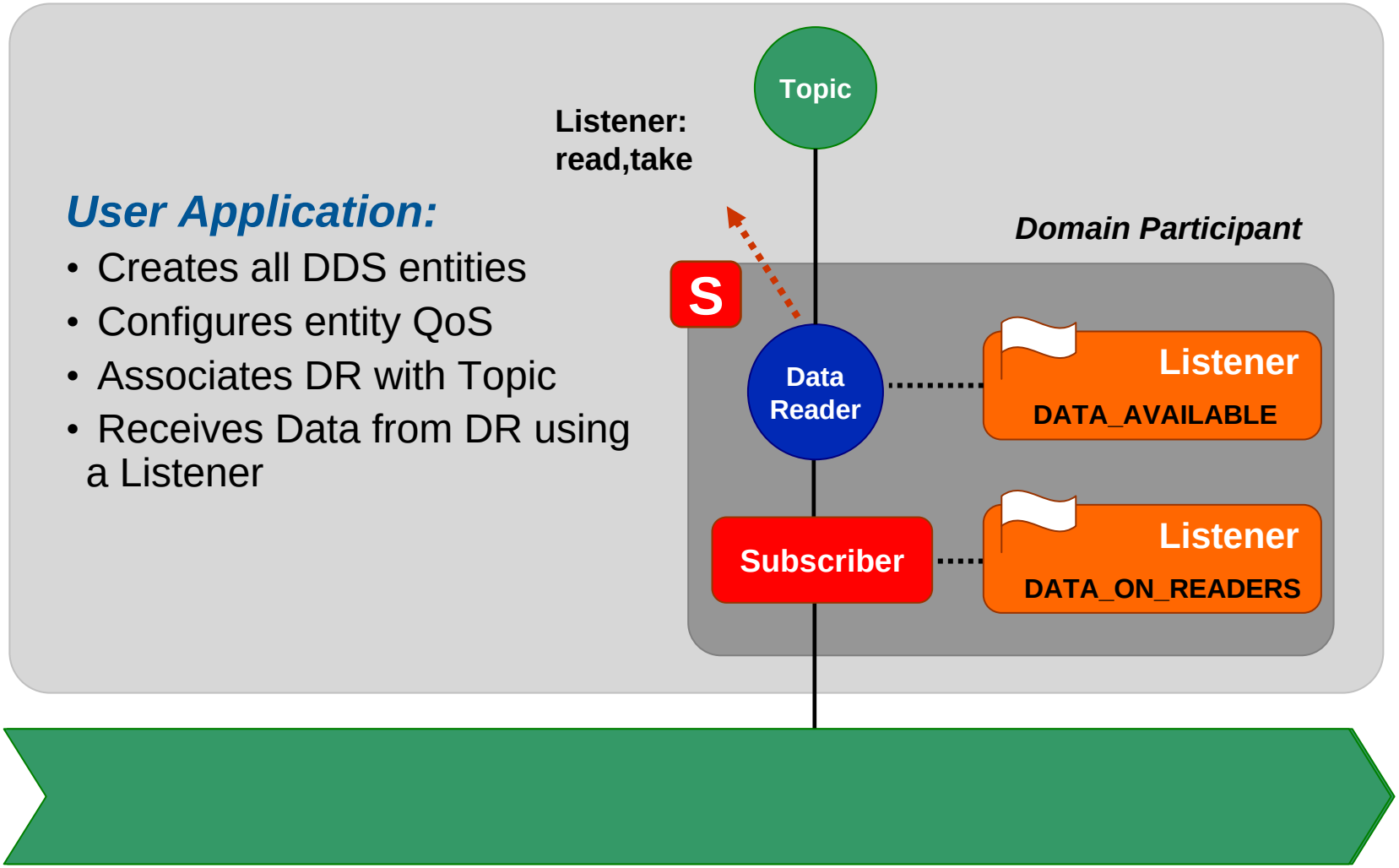
```
Publisher publisher = domain->create_publisher(
        publisher_qos,
        publisher_listener);

Topic topic = domain->create_topic(
        “Track”, “TrackStruct”,
        topic_qos, topic_listener);

DataWriter writer = publisher->create_datawriter(
        topic, writer_qos, writer_listener);
TrackStructDataWriter twriter =
      TrackStructDataWriter::narrow(writer);

TrackStruct my_track;
twriter->write(&my_track);
```

DDS Subscription Listener



Example: Subscription

```
Subscriber  subs = domain->create_subscriber(
       subscriber_qos, subscriber_listener);

Topic topic = domain->create_topic(
       “Track”, “TrackStruct”,
       topic_qos, topic_listener);

DataReader reader = subscriber->create_datareader(
       topic, reader_qos, reader_listener);

// Use listener-based or wait-based access
```

How to get data (listener-based)

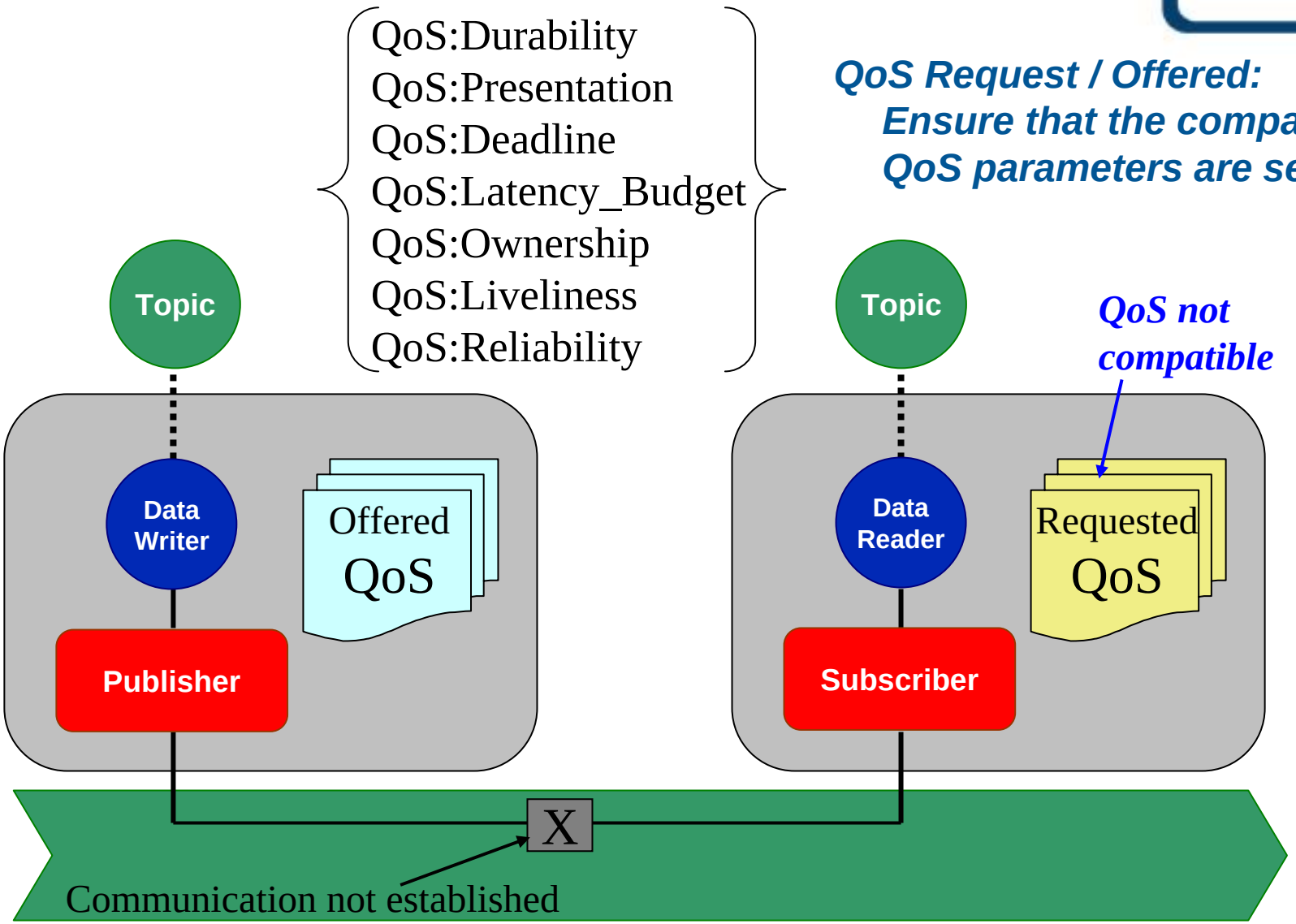
```
Listener listener = new MyListener();
reader->set_listener(listener);

MyListener::on_data_available( DataReader reader )
{
    TrackStructSeq received_data;
    SampleInfoSeq sample_info;
    TrackStructDataReader treader =
        TrackStructDataReader::narrow(reader);

    treader->take( &received_data,
                   &sample_info, …)

    // Use received_data
}
```

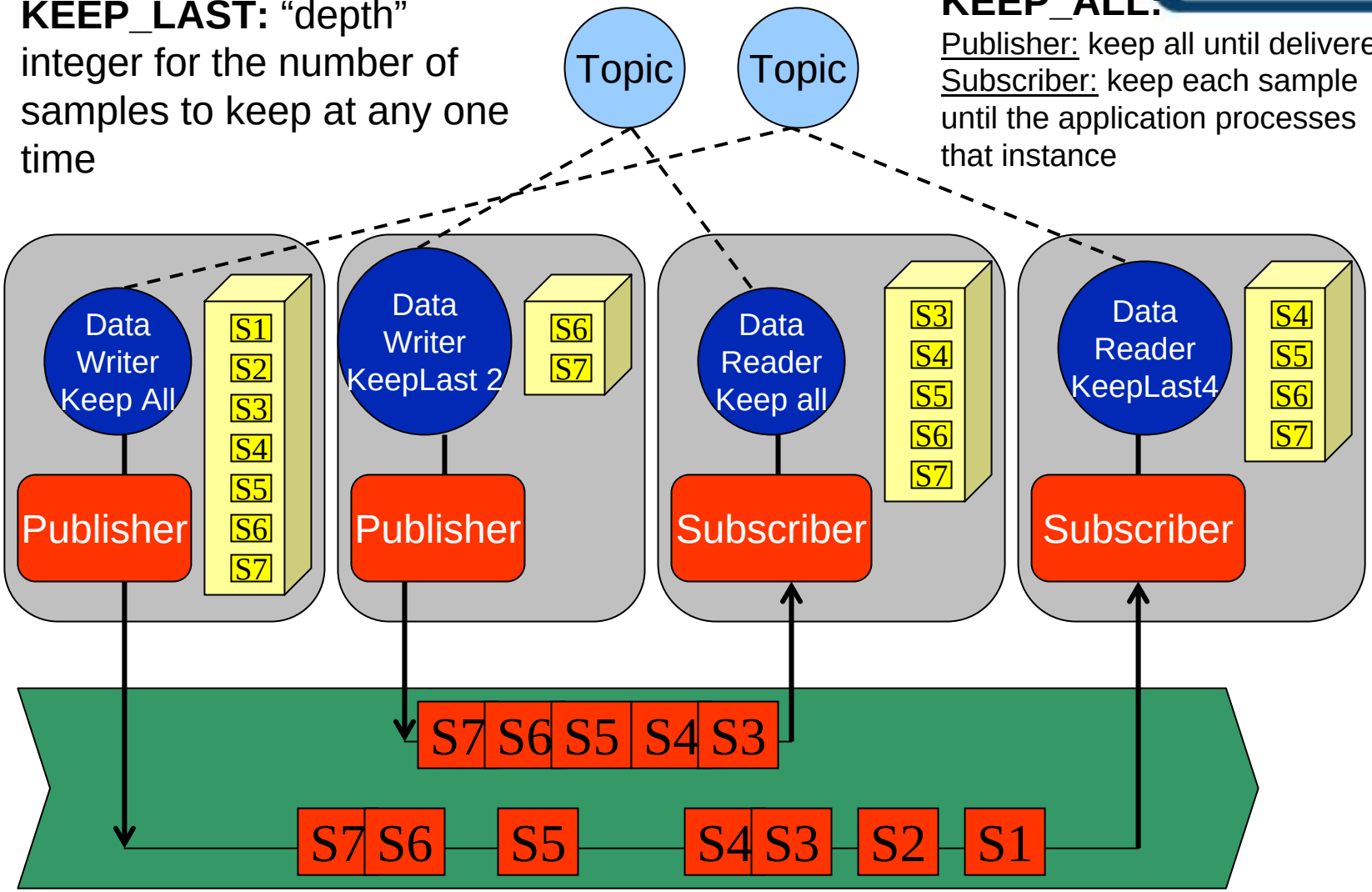
QoS Contract "Request / Offered"



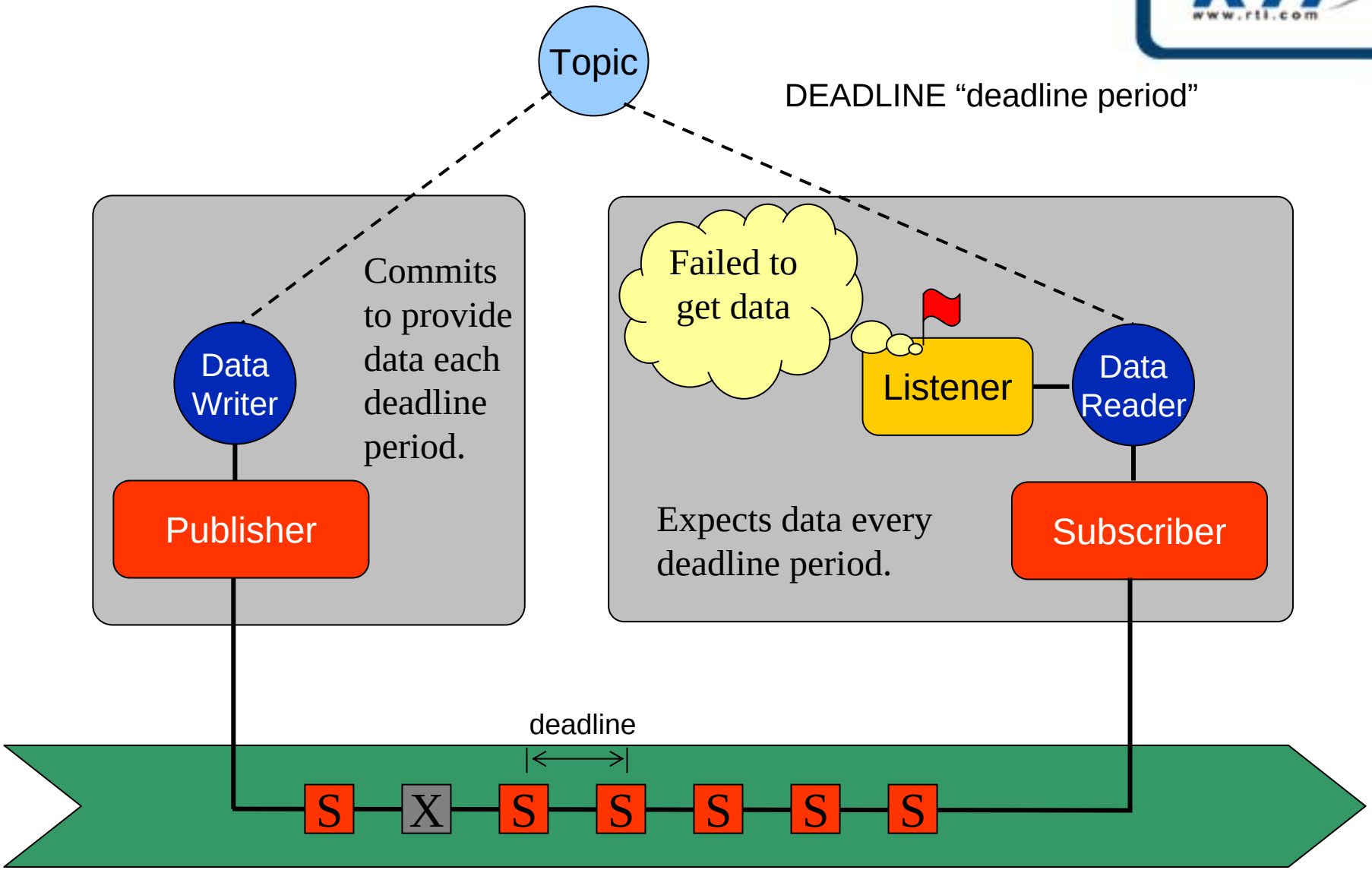
QoS: History: Last x or All

KEEP_LAST: "depth" integer for the number of samples to keep at any one time

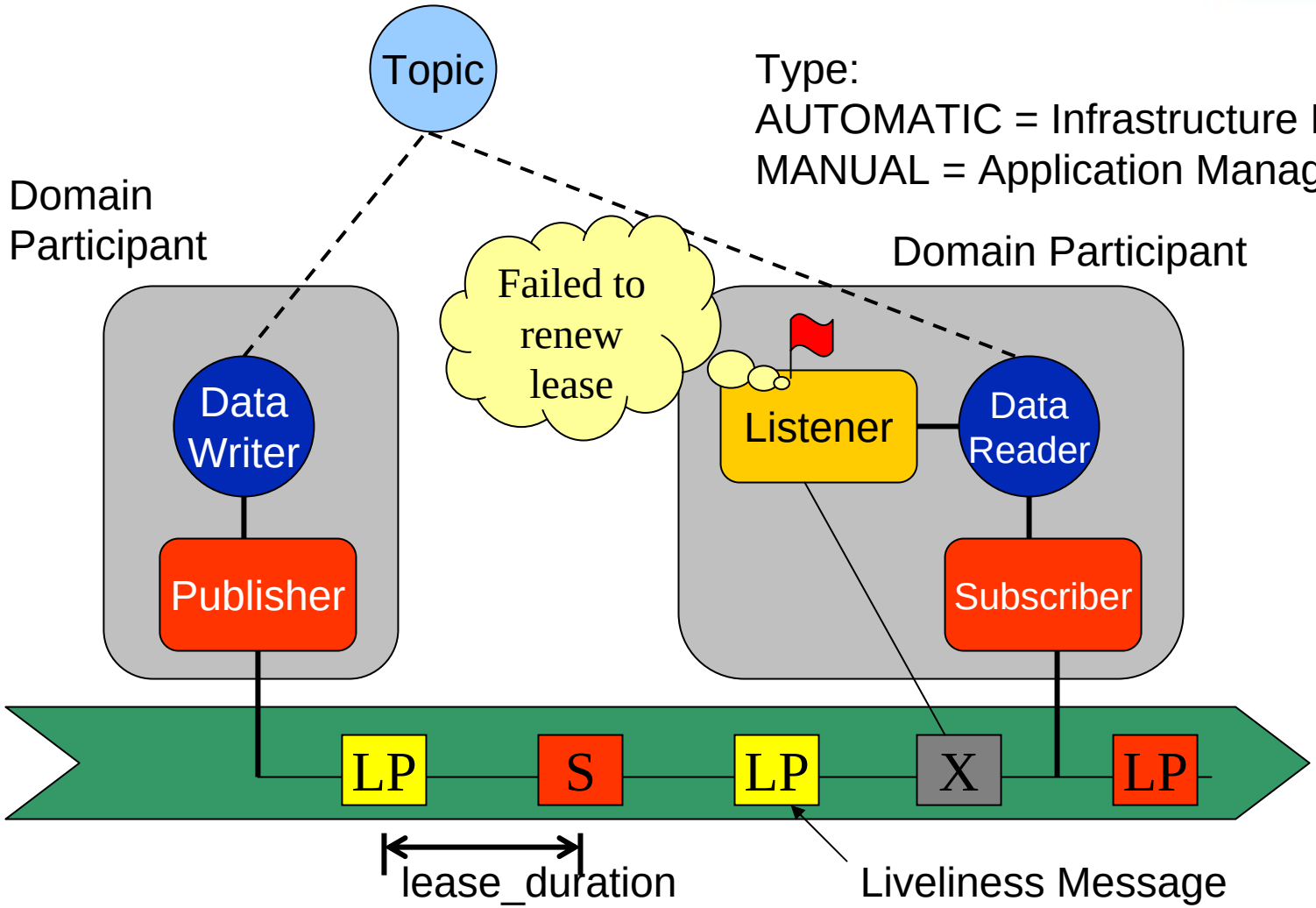
KEEP_ALL:
Publisher: keep all until delivered
Subscriber: keep each sample until the application processes that instance



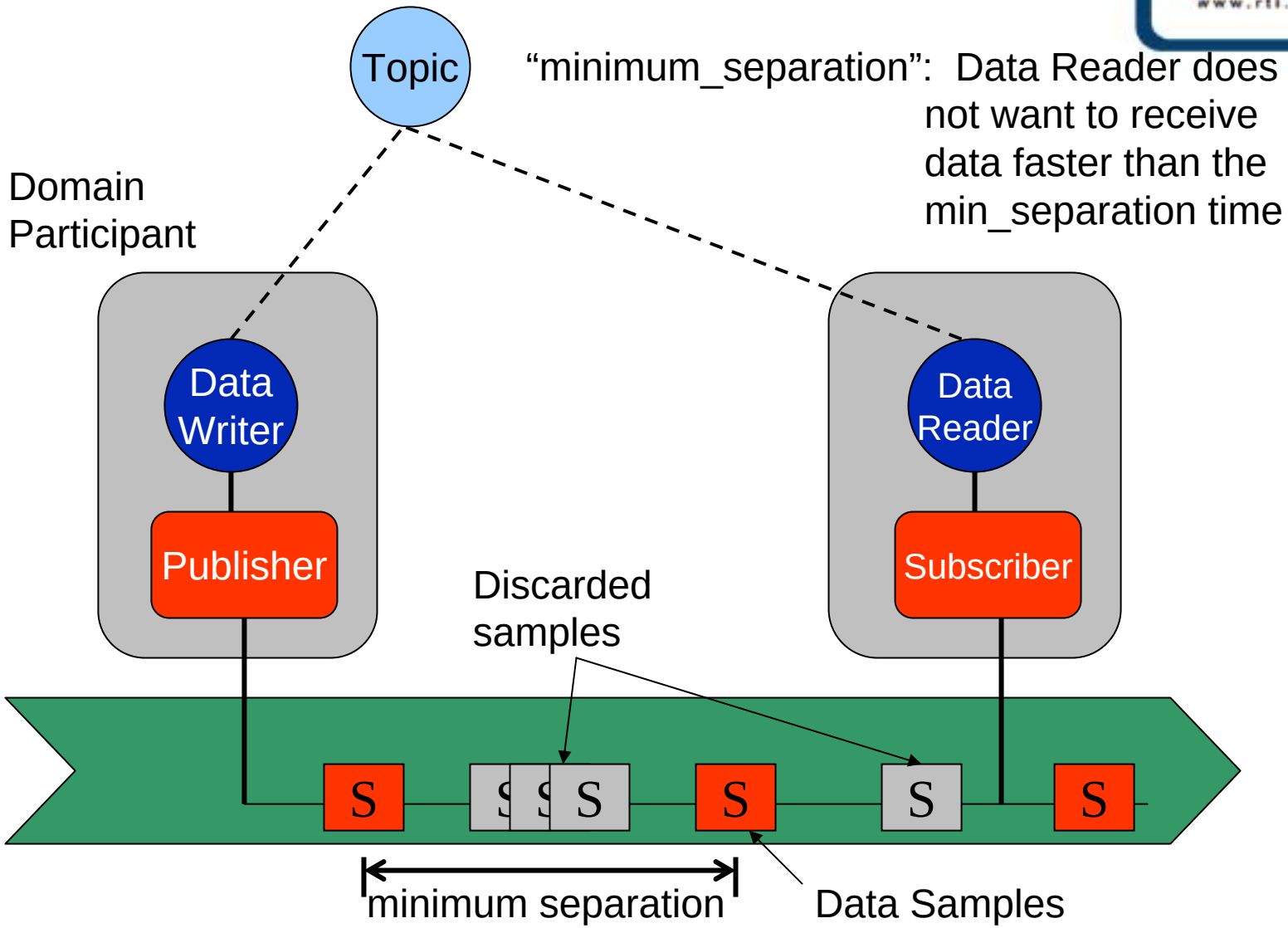
QoS: Deadline



QoS: Liveliness – Type, Duration



QoS: Time_Based_Filter



QoS: Quality of Service (1/2)

QoS Policy	Concerns	RxO	Changeable
DEADLINE	T,DR,DW	YES	YES
LATENCY BUDGET	T,DR,DW	YES	YES
READER DATA LIFECYCLE	DR	N/A	YES
WRITER DATA LIFECYCLE	DW	N/A	YES
TRANSPORT PRIORITY	T,DW	N/A	YES
LIFESPAN	T,DW	N/A	YES
LIVELINESS	T,DR,DW	YES	NO
TIME BASED FILTER	DR	N/A	YES
RELIABILITY	T,DR,DW	YES	NO
DESTINATION ORDER	T,DR	NO	NO

QoS: Quality of Service (2/2)

QoS Policy	Concerns	RxO	Changeable
USER DATA	DP,DR,DW	NO	YES
TOPIC DATA	T	NO	YES
GROUP DATA	P,S	NO	YES
ENTITY FACTORY	DP, P, S	NO	YES
PRESENTATION	P,S	YES	NO
OWNERSHIP	T	YES	NO
OWNERSHIP STRENGTH	DW	N/A	YES
PARTITION	P,S	NO	YES
DURABILITY	T,DR,DW	YES	NO
HISTORY	T,DR,DW	NO	NO
RESOURCE LIMITS	T,DR,DW	NO	NO

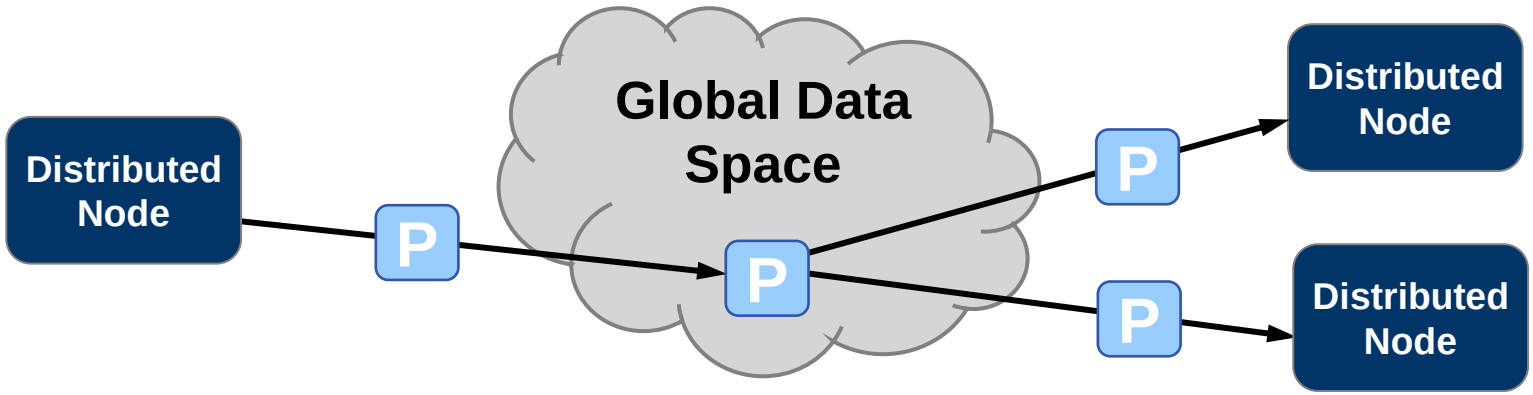
Summary

DDS targets applications that need to distribute data in a real-time environment

DDS is highly configurable by QoS settings

DDS provides a shared "global data space"

- Any application can publish data it has
- Any application can subscribe to data it needs
- Automatic discovery
- Facilities for fault tolerance
- Heterogeneous systems easily accommodated



Thank you

References:

OMG DDS specification:

http://www.omg.org/cgi-bin/doc?ptc/04-04-12

General material on DDS and RTI's implementation:

http://www.rti.com/dds

Comments/questions: gerardo@rti.com