Data Distribution Service (DDS)

Tutorial

Gerardo Pardo-Castellote

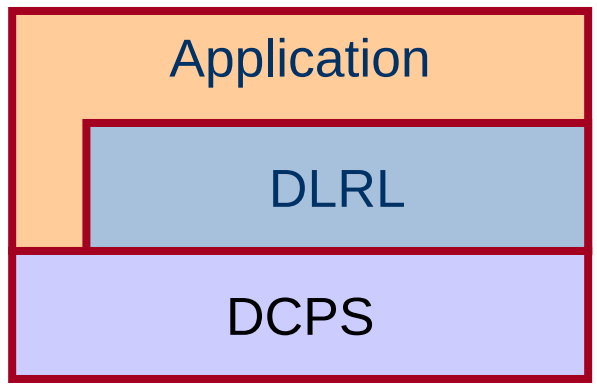
Real-Time Innovations, Inc.

The DDS Standard

- Data Distribution Service for Real-Time Systems
  - Adopted in June 2003
  - Finalized in June 2004 (pending)
  - Joint submission (RTI, THALES, MITRE, OIS)
  - Specification of API required to facilitate the Data-Centric Publish-Subscribe communication environment for real-time distributed systems.

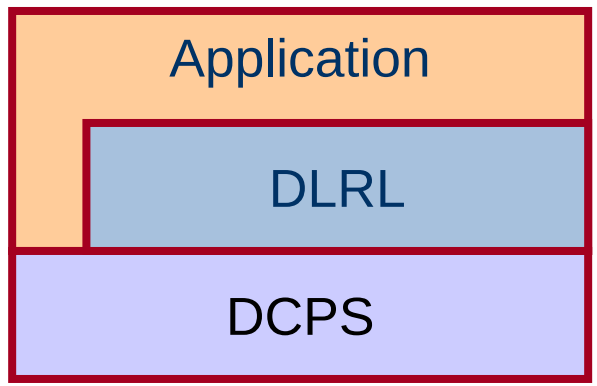
What is DDS

- DCPS = Data Centric Publish_Subscribe
  - Purpose: Distribute the data
- DLRL = Data Local Reconstruction Layer
  - Purpose: provide an object-based model to access data ‘as if’ it was local



What is DDS

- DCPS = Data Centric Publish_Subscribe
  - Purpose: Distribute the data
- DLRL = Data Local Reconstruction Layer
  - Purpose: provide an object-based model to access data ‘as if’ it was local

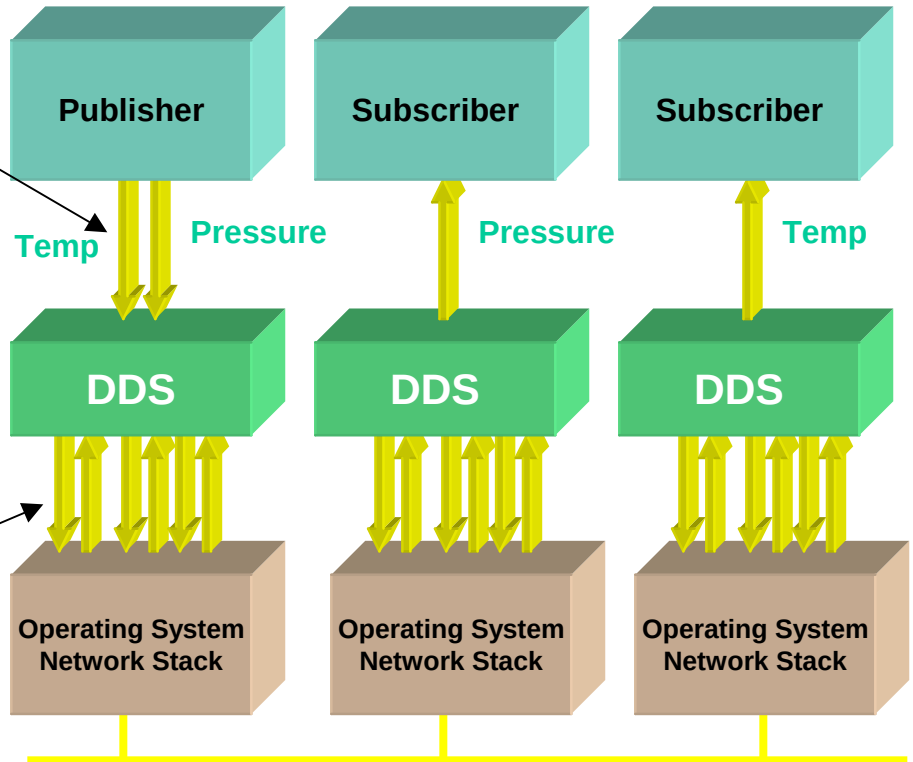


Data Distribution Service - DCPS

What is DCPS?

- Data-Distribution for Real-Time Systems
  - Just declare your intent to publish or receive data.
  - No need to make a special request for every piece of data.

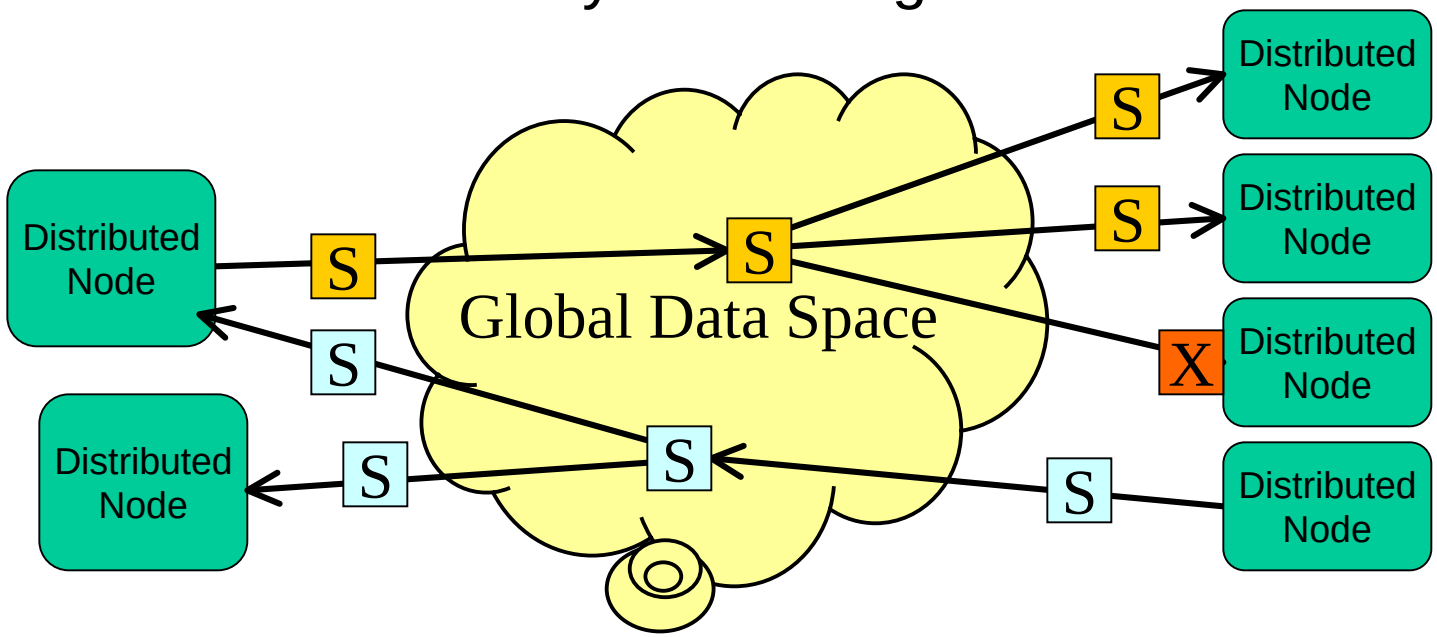
Applications just send or receive data with a standard API.



DDS does the addressing, data conversion, sending, receiving, and retries.

DDS/DCPS

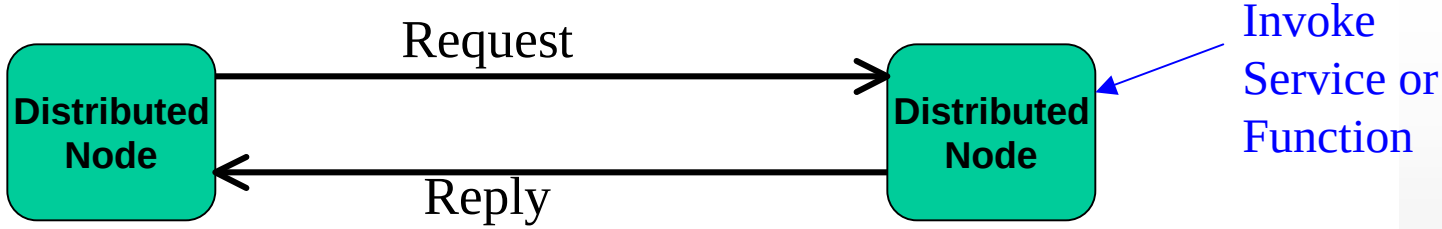
- Provides a “Global Data Space” that is accessible to all interested applications.
  - Subscriptions are decoupled from Publications
  - Contracts established by means of QoS
  - Automatic discovery and configuration



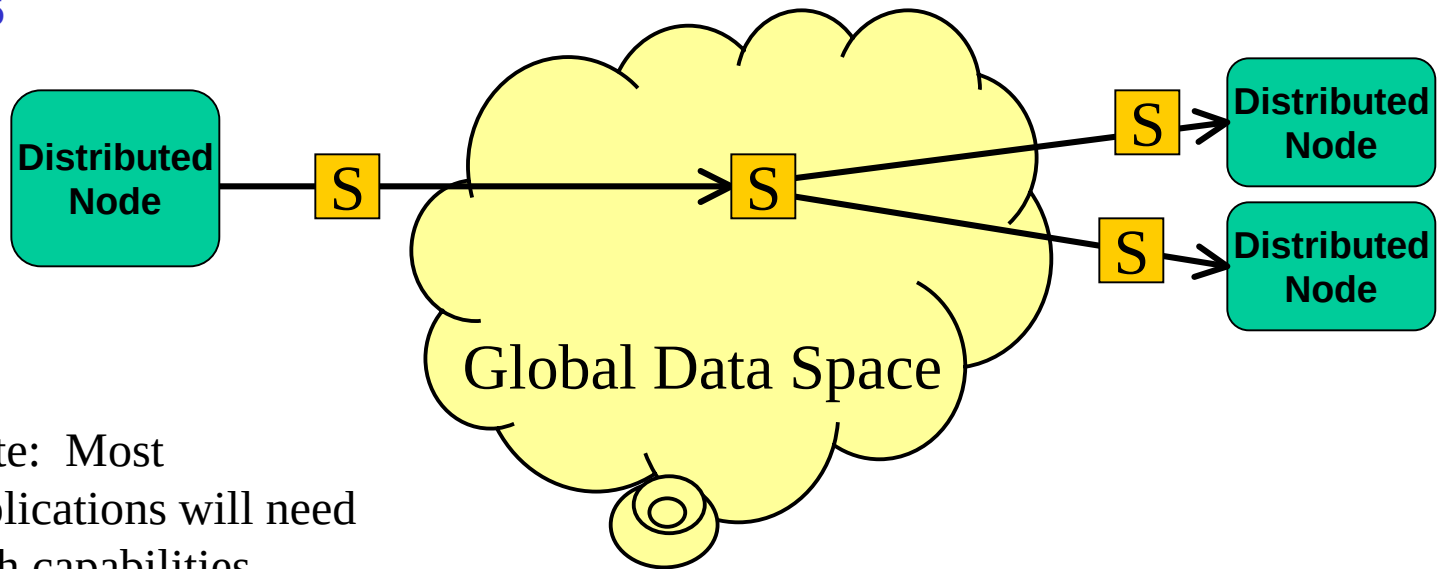
Why DDS/DCPS?

- Augment existing distributed object services

CORBA/ RMI



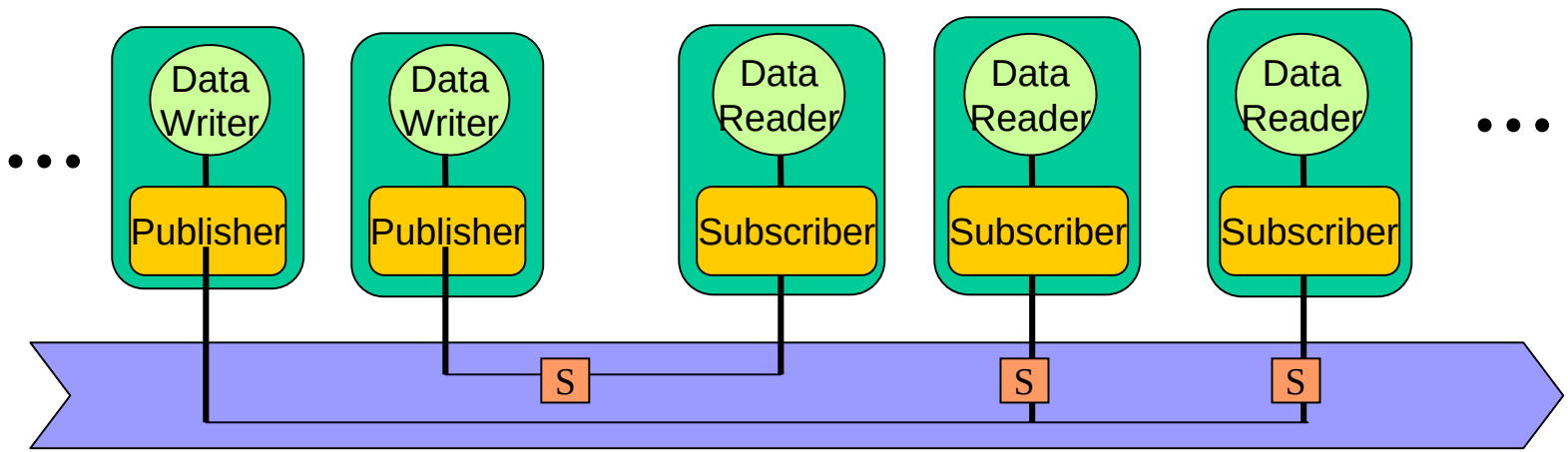
DDS



Note: Most applications will need both capabilities

Benefits of DDS/DCPS

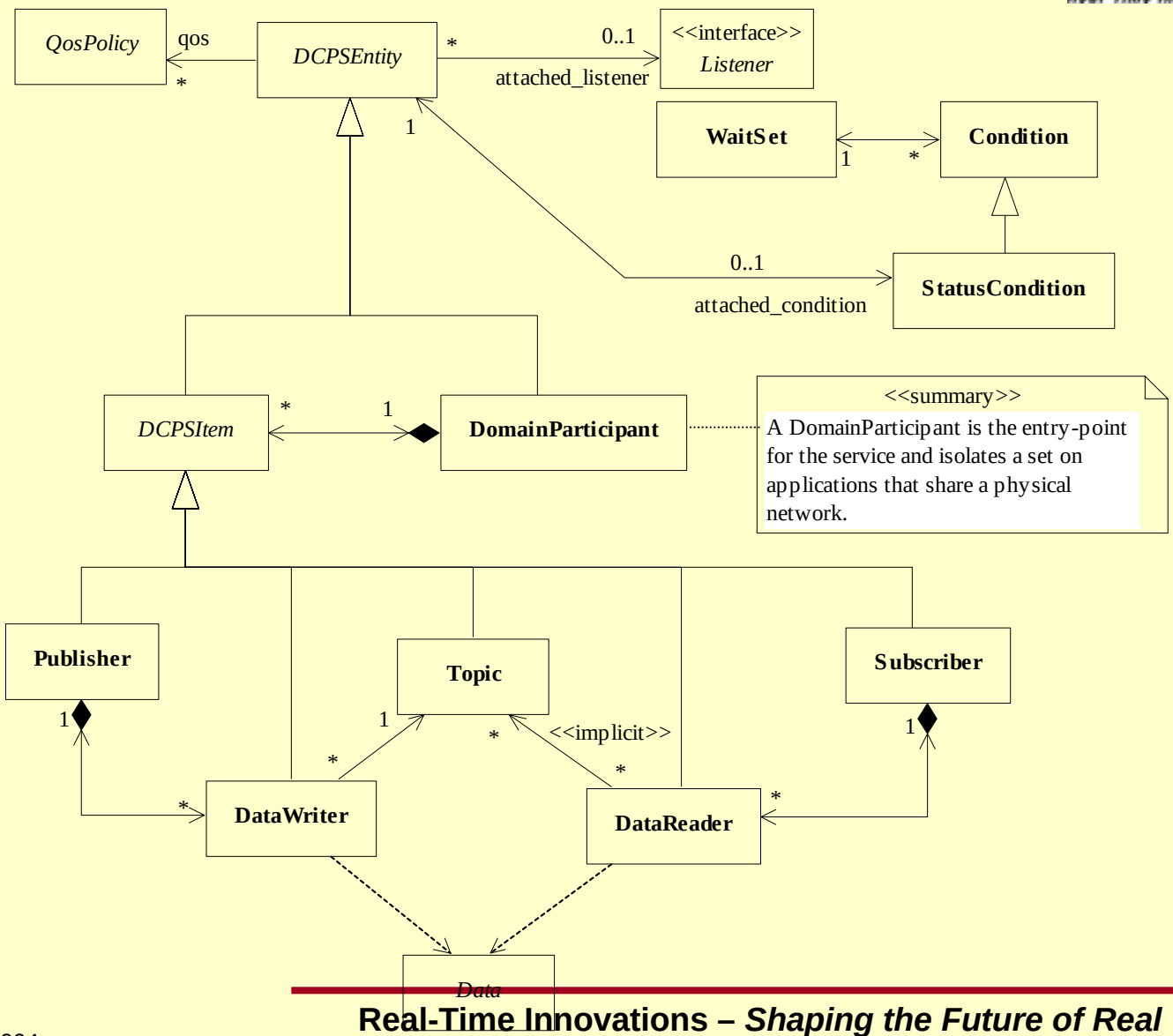
- Predictable and/or reliable data distribution with minimal overhead
  - No reply confirmation needed.
  - Reliable ←→ acyclic messages
  - QoS parameters used to set up communication determinism / reliability.
- Plug & Play
  - Dynamic, automatic discovery



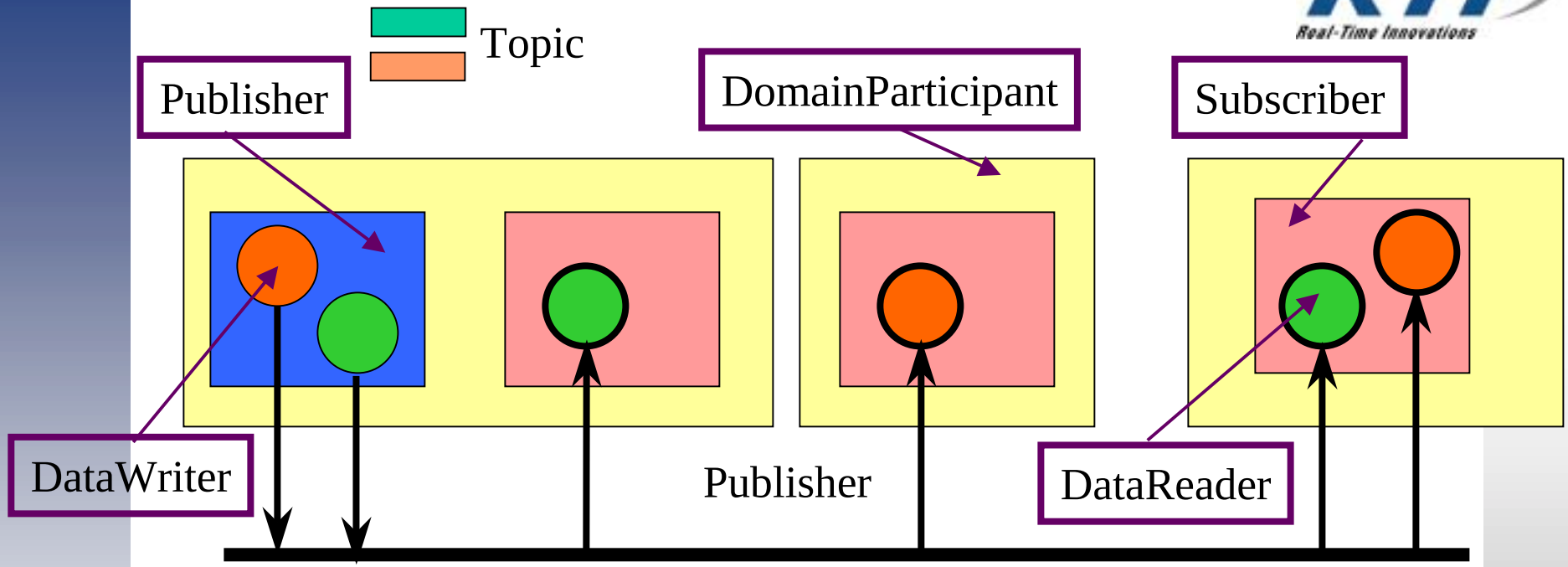
The DDS/DCPS Model

- Data Domains
- Topics
- Publications & Subscriptions
  - One to One, One to Many, Many to One
  - State propagation
- Quality of Service
  - Reliability
  - Predictability
  - Fault Tolerance
  - Scalability

PIM Overview

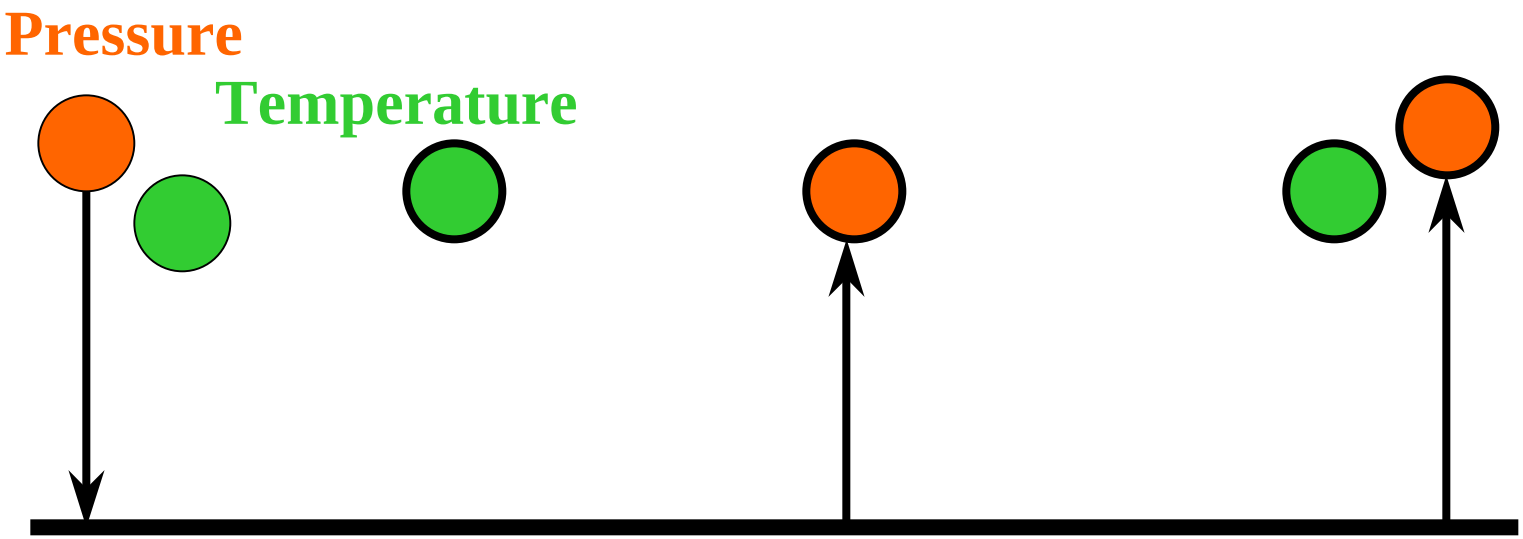


DCPS Entities



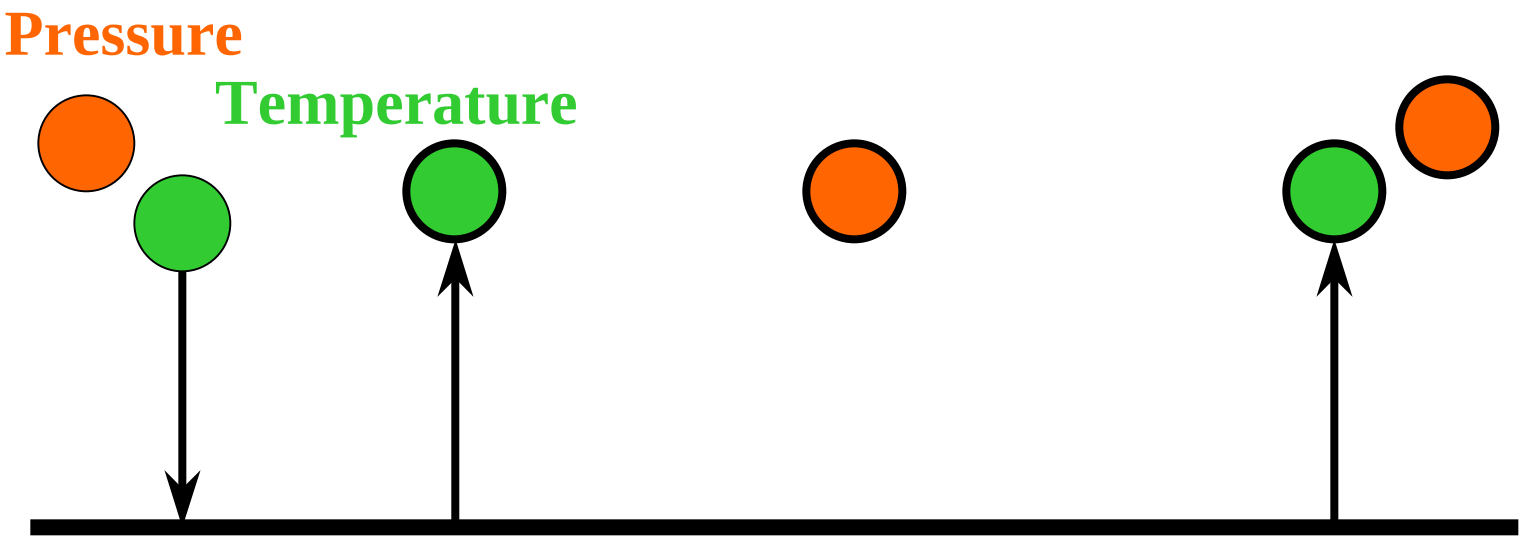
- **DomainParticipant** ~ Represents participation of the application in the communication collective
- **DataWriter** ~ Accessor to write typed data on a particular **Topic**
- **Publisher** ~ Aggregation of DataWriter objects. Responsible for disseminating information.
- **DataReader** ~ Accessor to read typed data regarding a specific **Topic**
- **Subscriber** ~ Aggregation of DataReader objects. Responsible for receiving information

Topic-based publish-subscribe



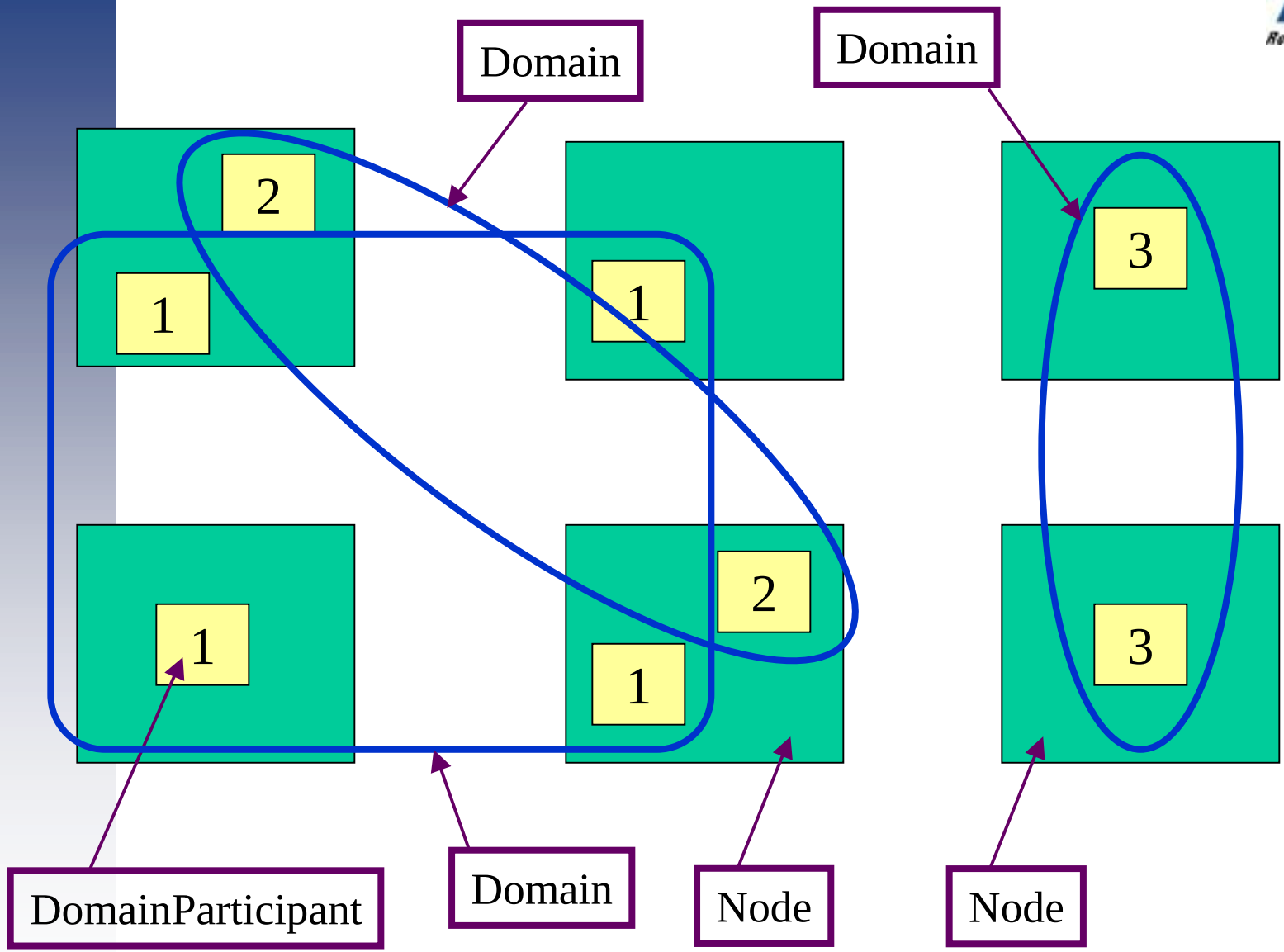
- **DataWriter** is bound (at creation time) to a single **Topic**
- **DataReader** is bound (at creation time) with one or more topics (**Topic, ContentFilteredTopic**, or **MultiTopic**)
- **ContentFilteredTopic** and **MultiTopic** provide means for content-based subscriptions

Topic-based publish-subscribe

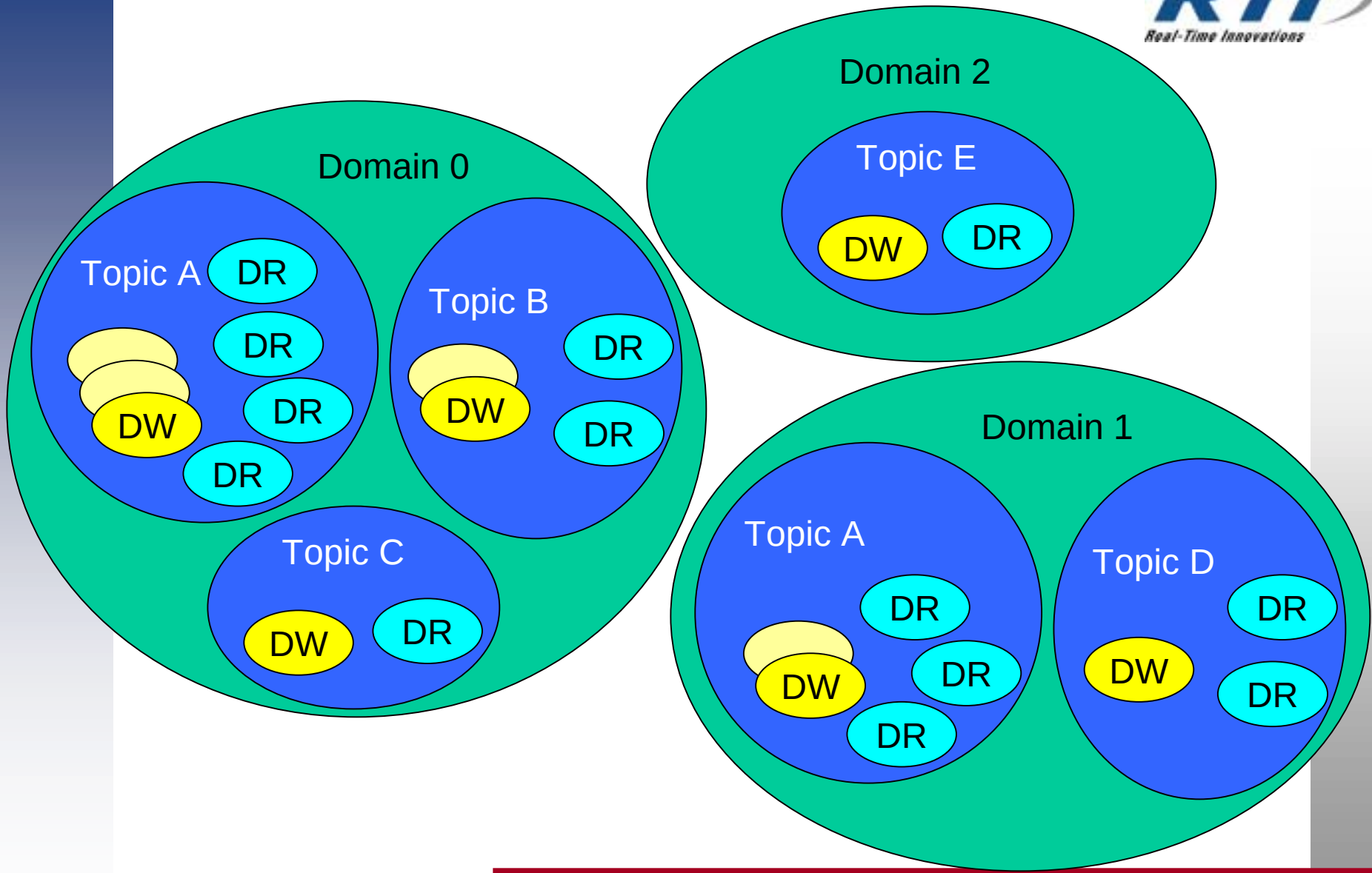


- **DataWriter** is bound (at creation time) to a single **Topic**
- **DataReader** is bound (at creation time) with one or more topics (**Topic, ContentFilteredTopic**, or **MultiTopic**)
- **ContentFilteredTopic** and **MultiTopic** provide means for content-based subscriptions

Domains and Participants

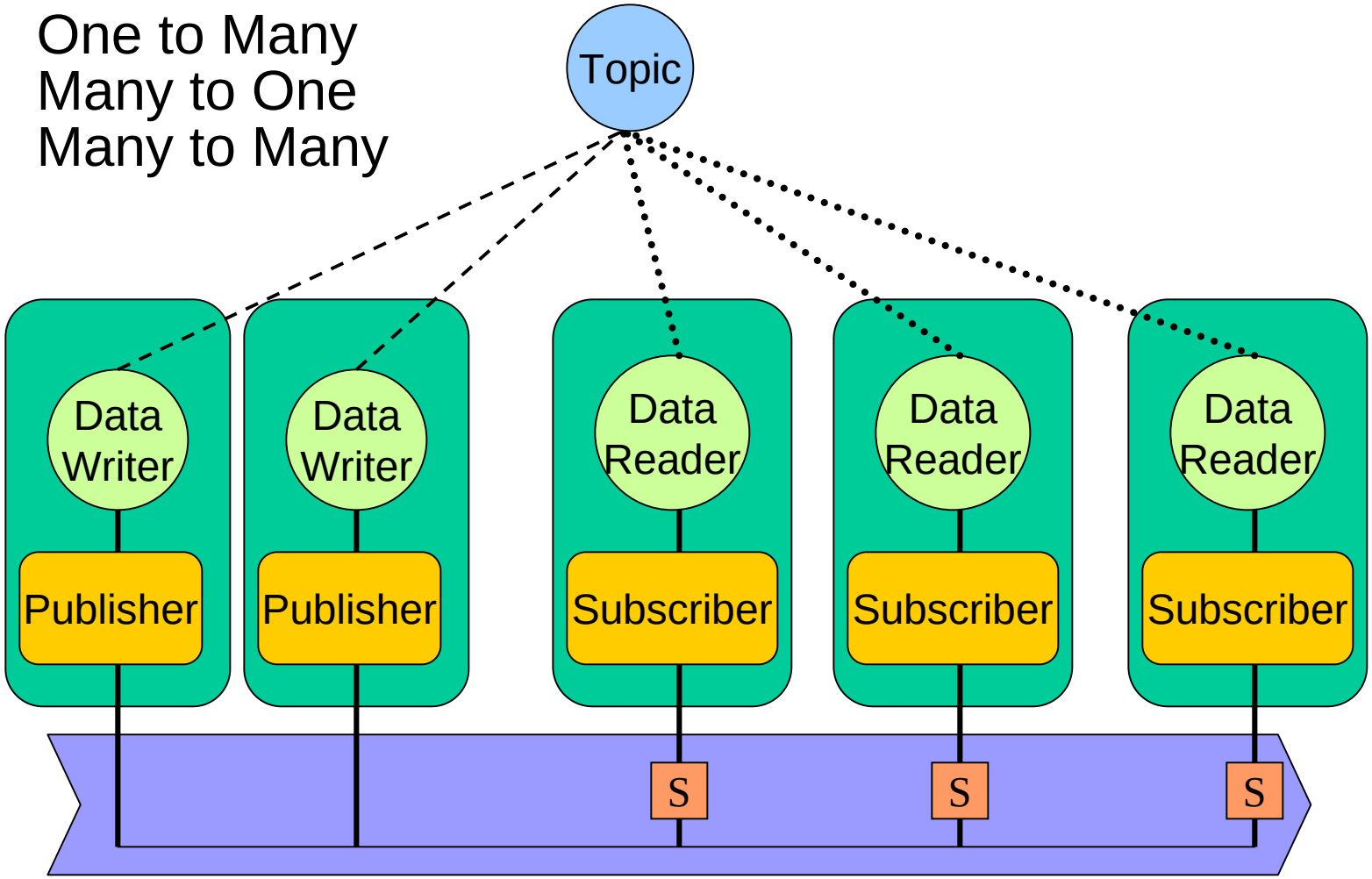


Domains: Topic view

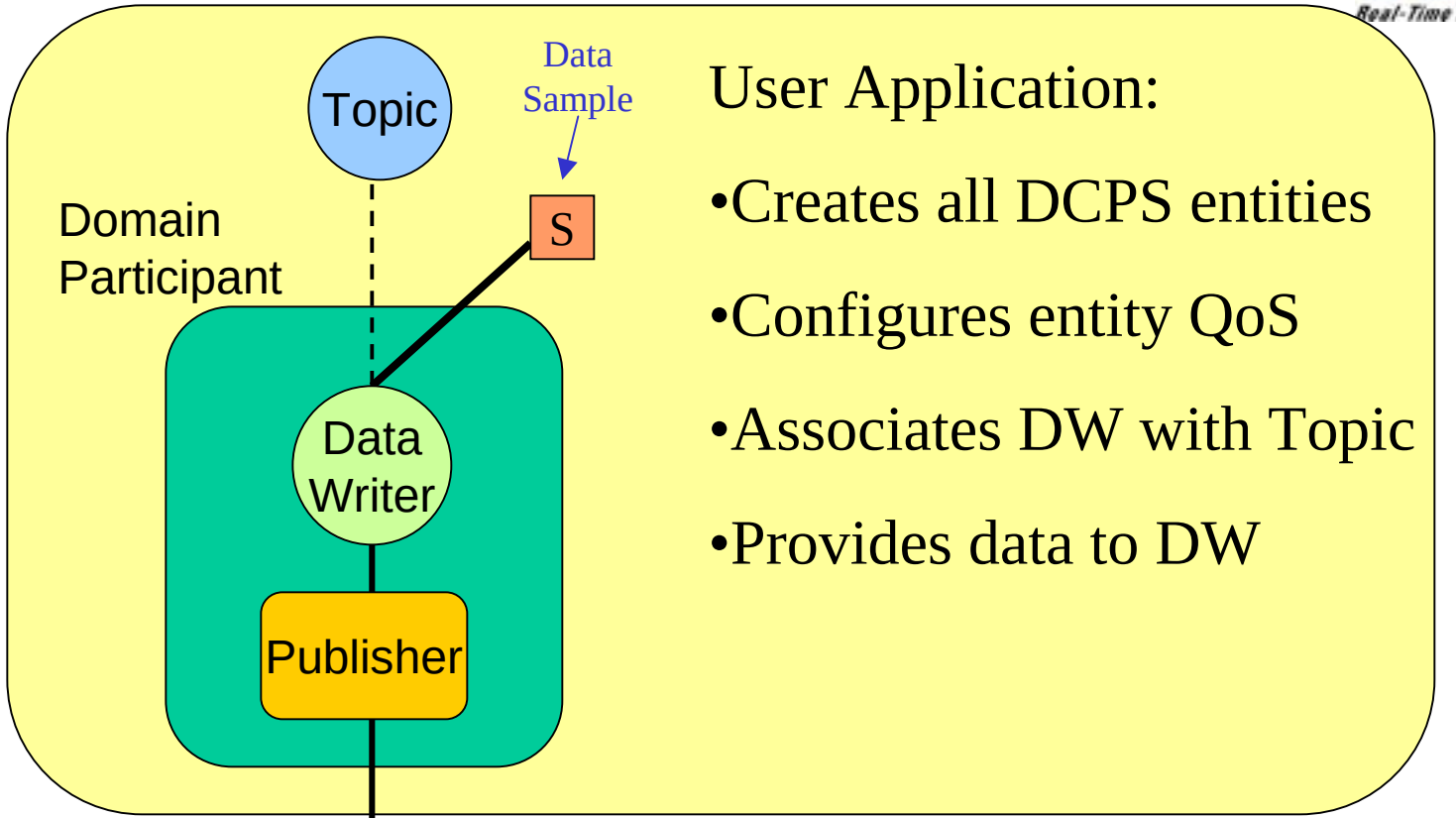


Pub/Sub Scenarios

One to One
One to Many
Many to One
Many to Many



DCPS Publication Objects



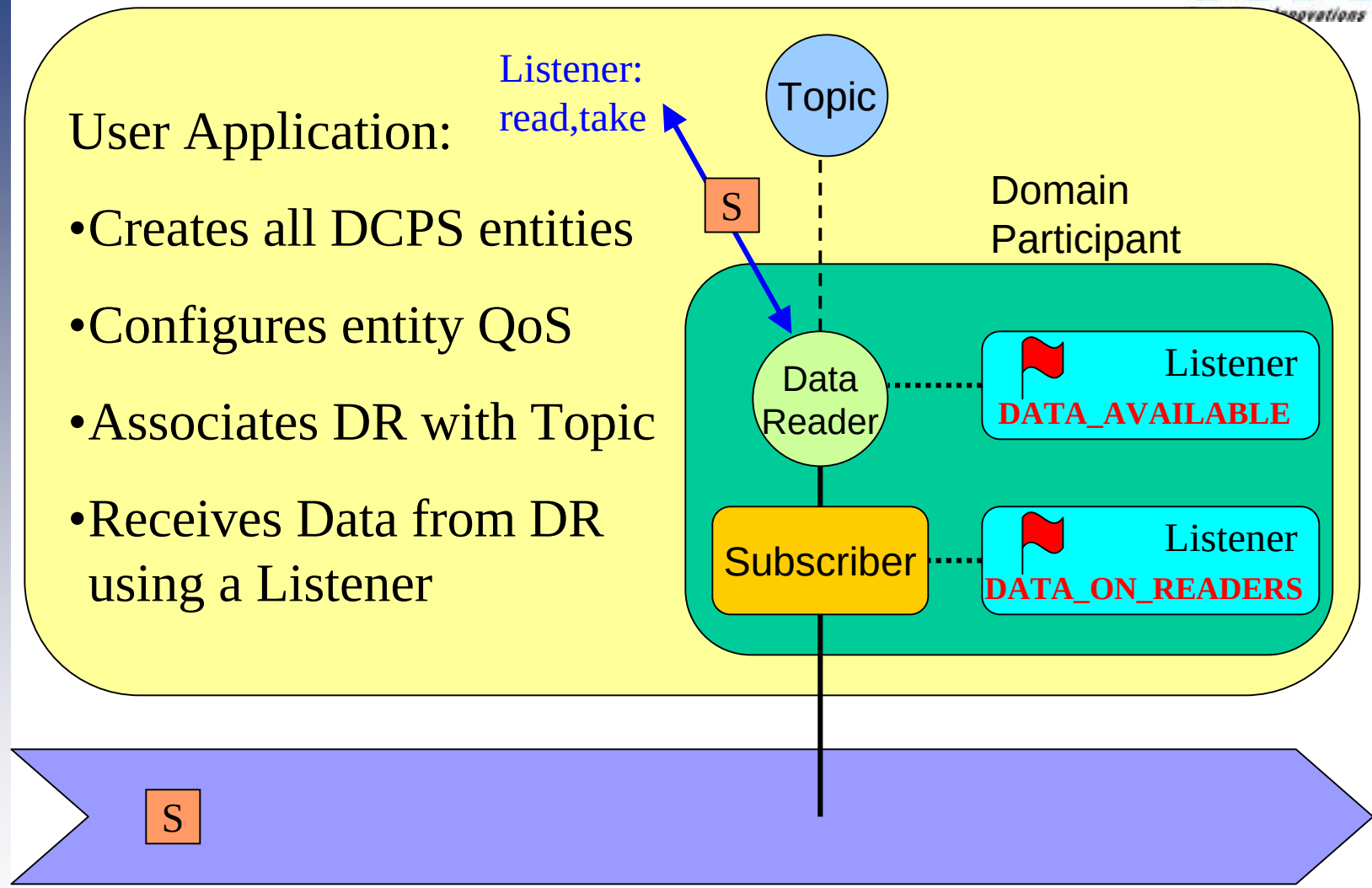
User Application:

- Creates all DCPS 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);
```

DCPS 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
```

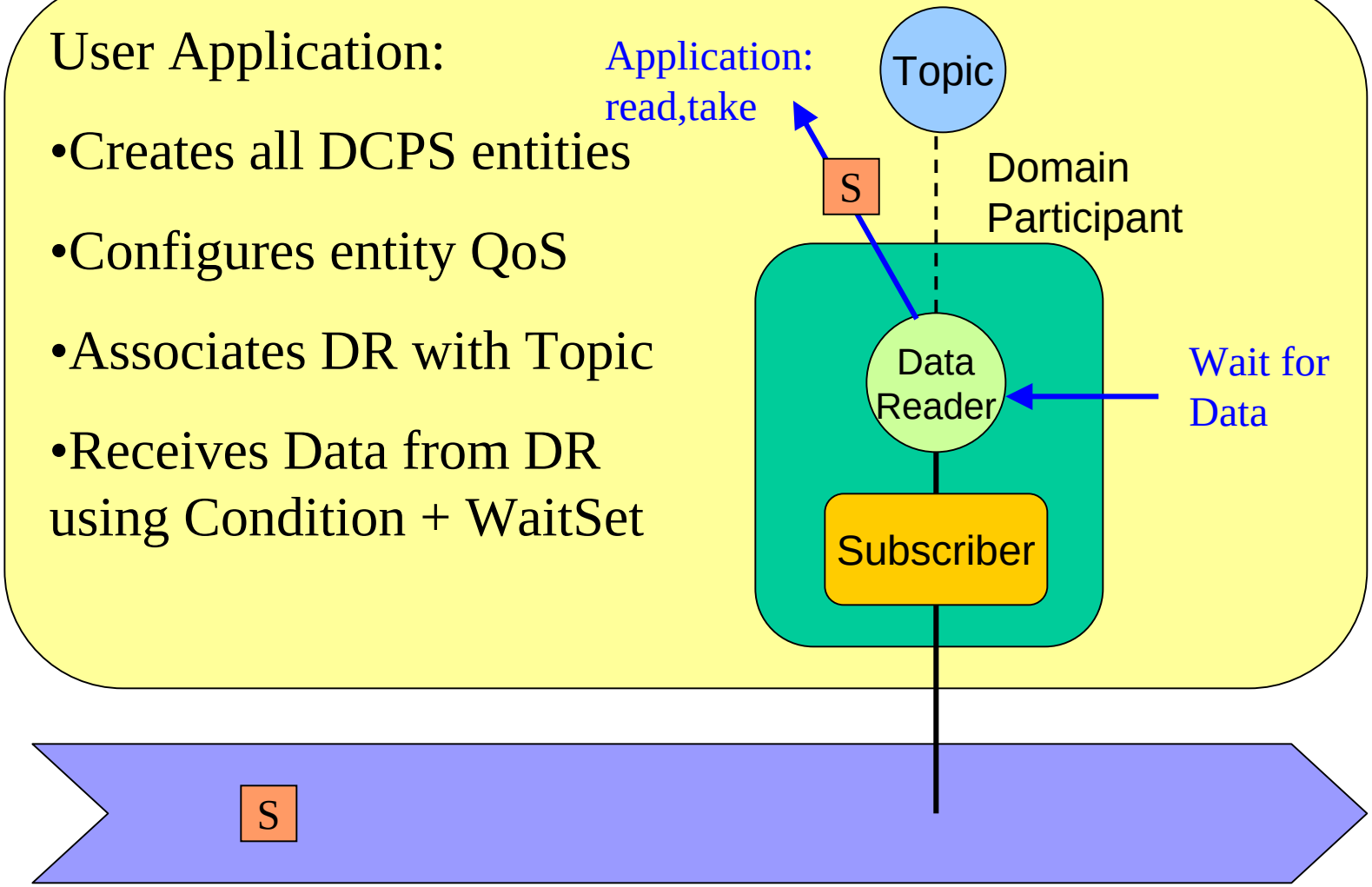
How to get data (listener-based)

```
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
}
```

DCPS Subscription Wait-Set

User Application:

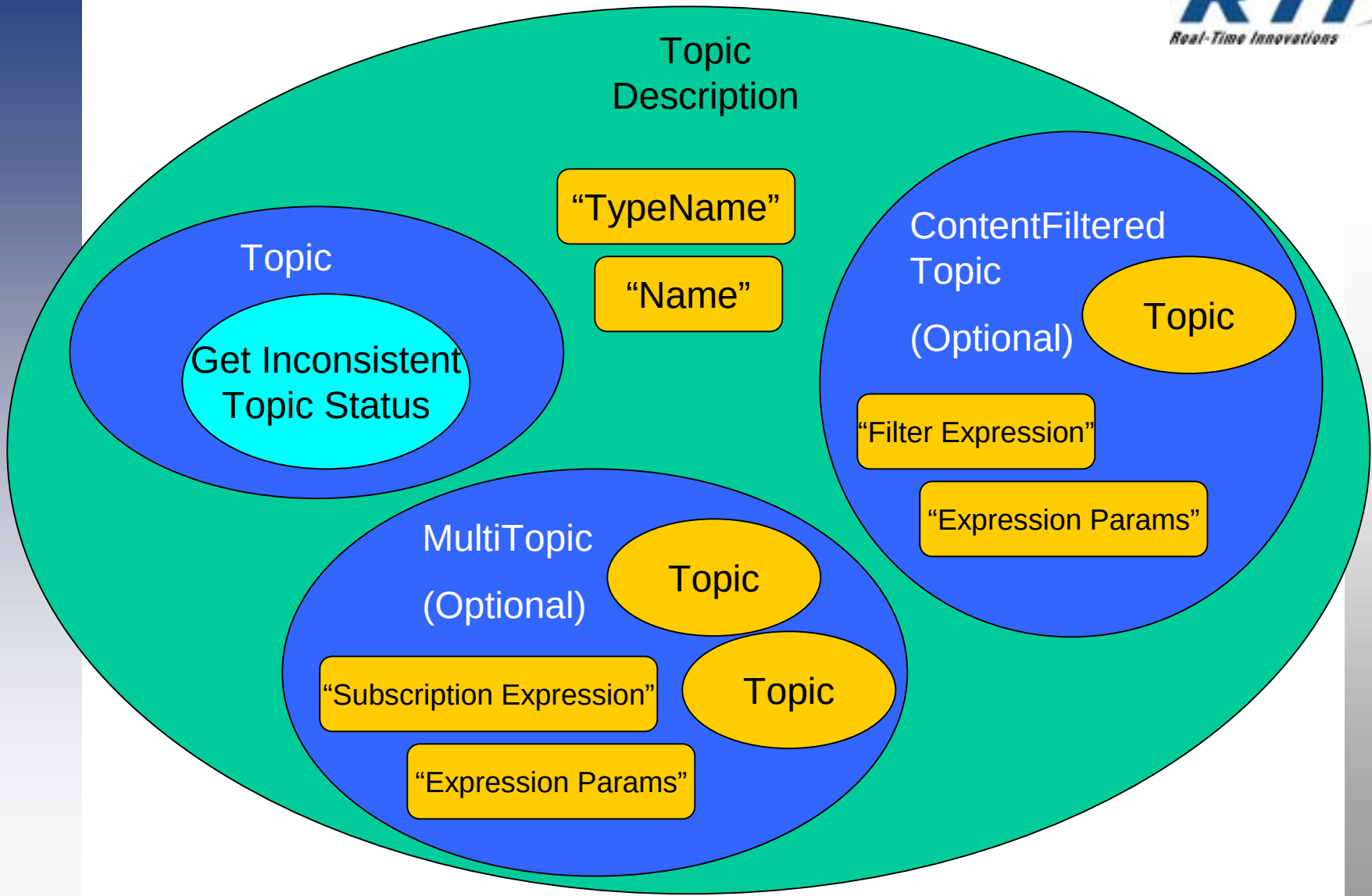
- Creates all DCPS entities
- Configures entity QoS
- Associates DR with Topic
- Receives Data from DR using Condition + WaitSet



How to get data (wait-based)

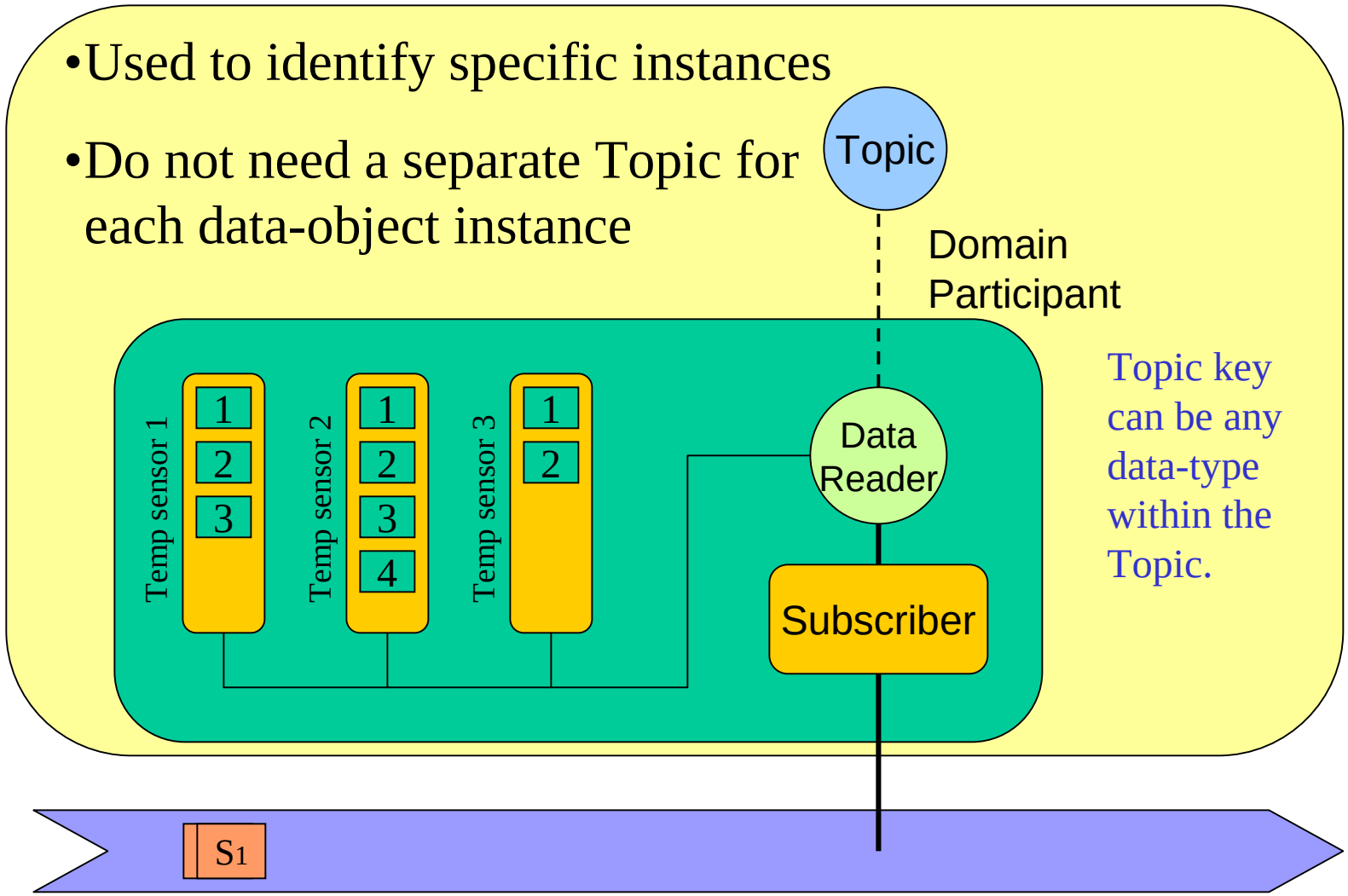
```
Condition  foo_condition =
   treader->create_readcondition(…);

waitset->add_condition(foo_condition);

ConditionSeq active_conditions;
waitset->wait(&active_conditions, timeout);
…
FooSeq received_data;
SampleInfoSeq sample_info;

treader->take_w_condition(&received_data,
                          &sample_info,
                          foo_condition);
// Use received_data
```

NDDS 4.0 Topics

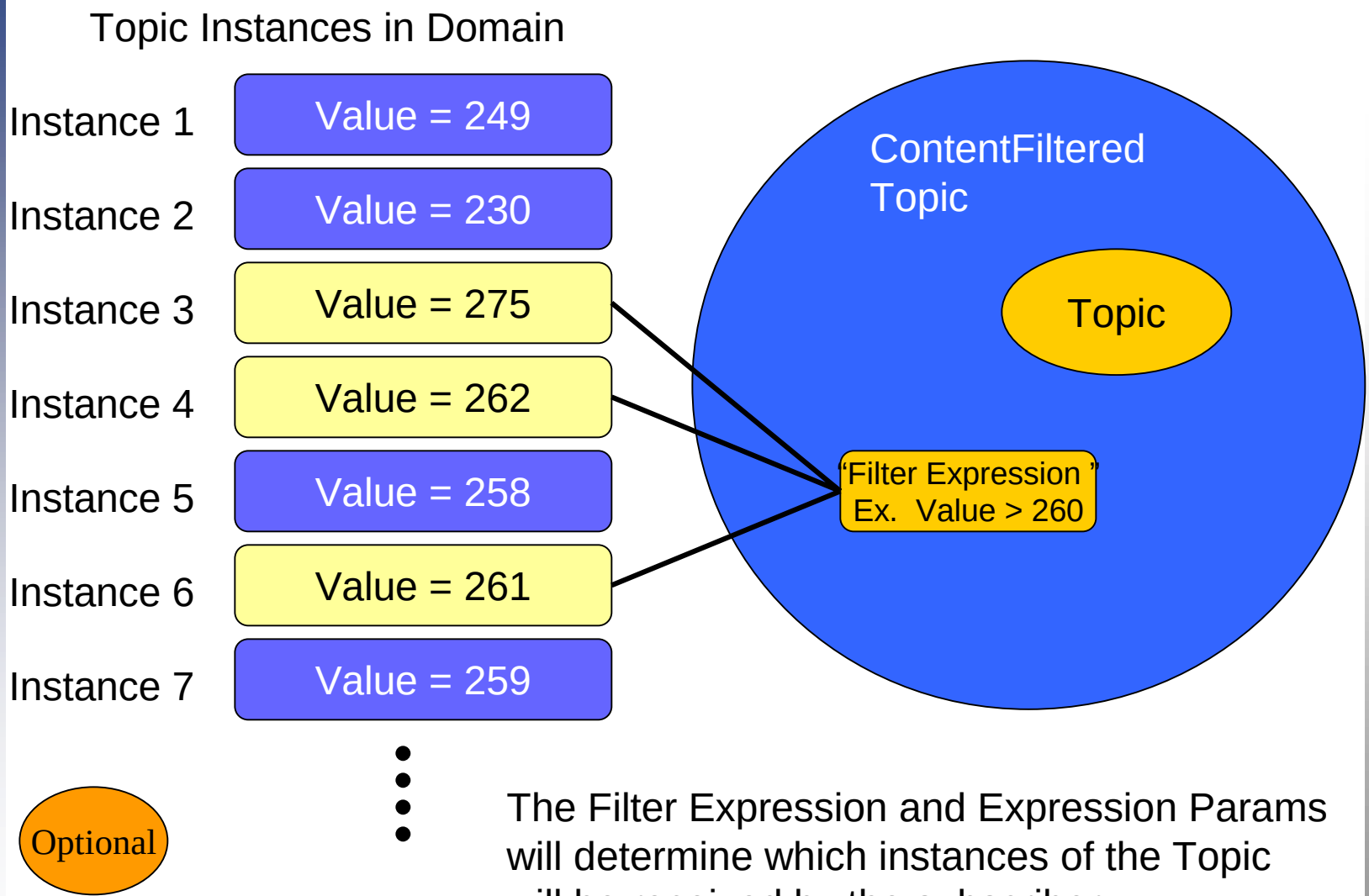


Keys (data-object identification)

- Multiple instances of the same topic

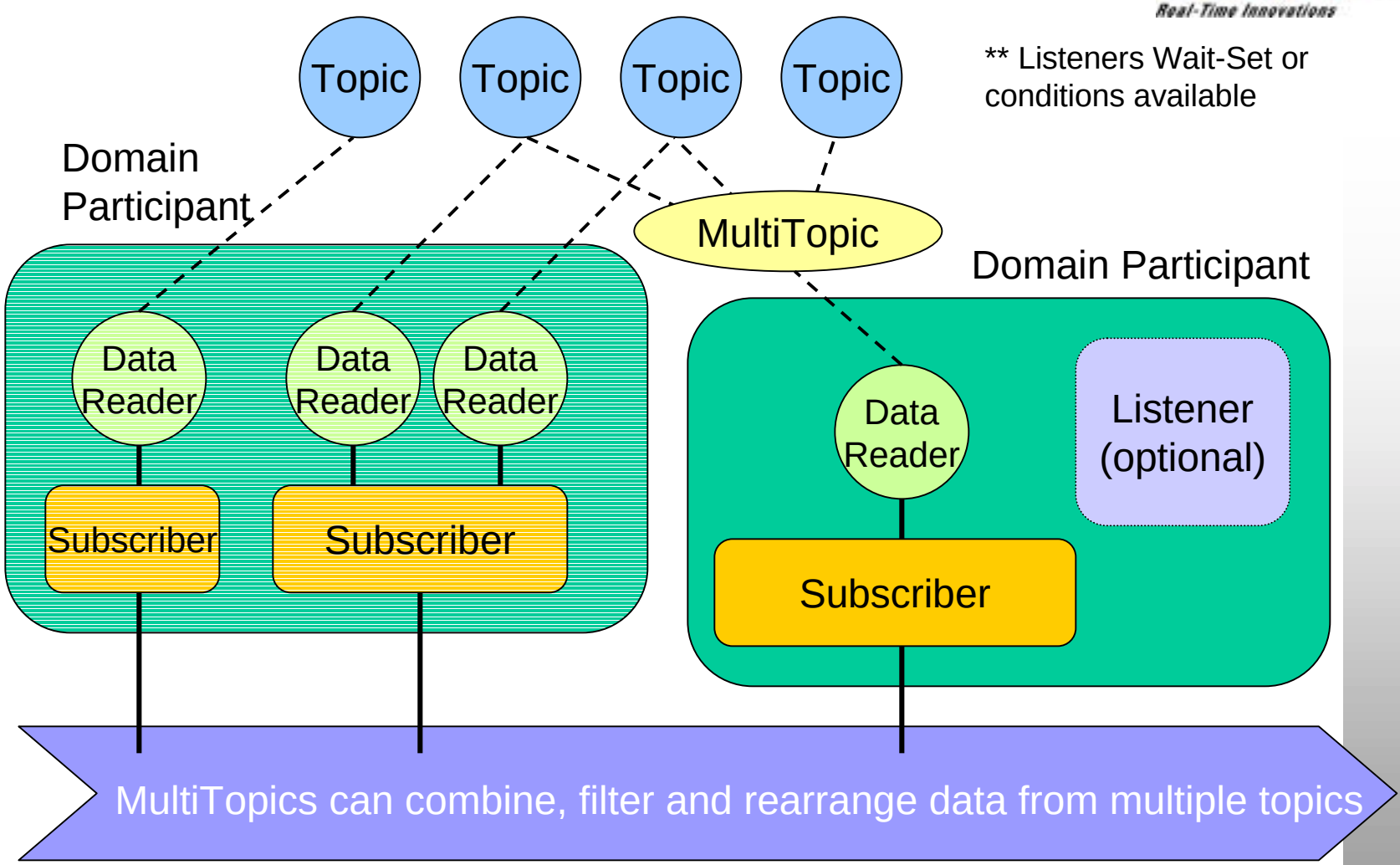


DCPS Content Filtered Topics



The Filter Expression and Expression Params will determine which instances of the Topic will be received by the subscriber.

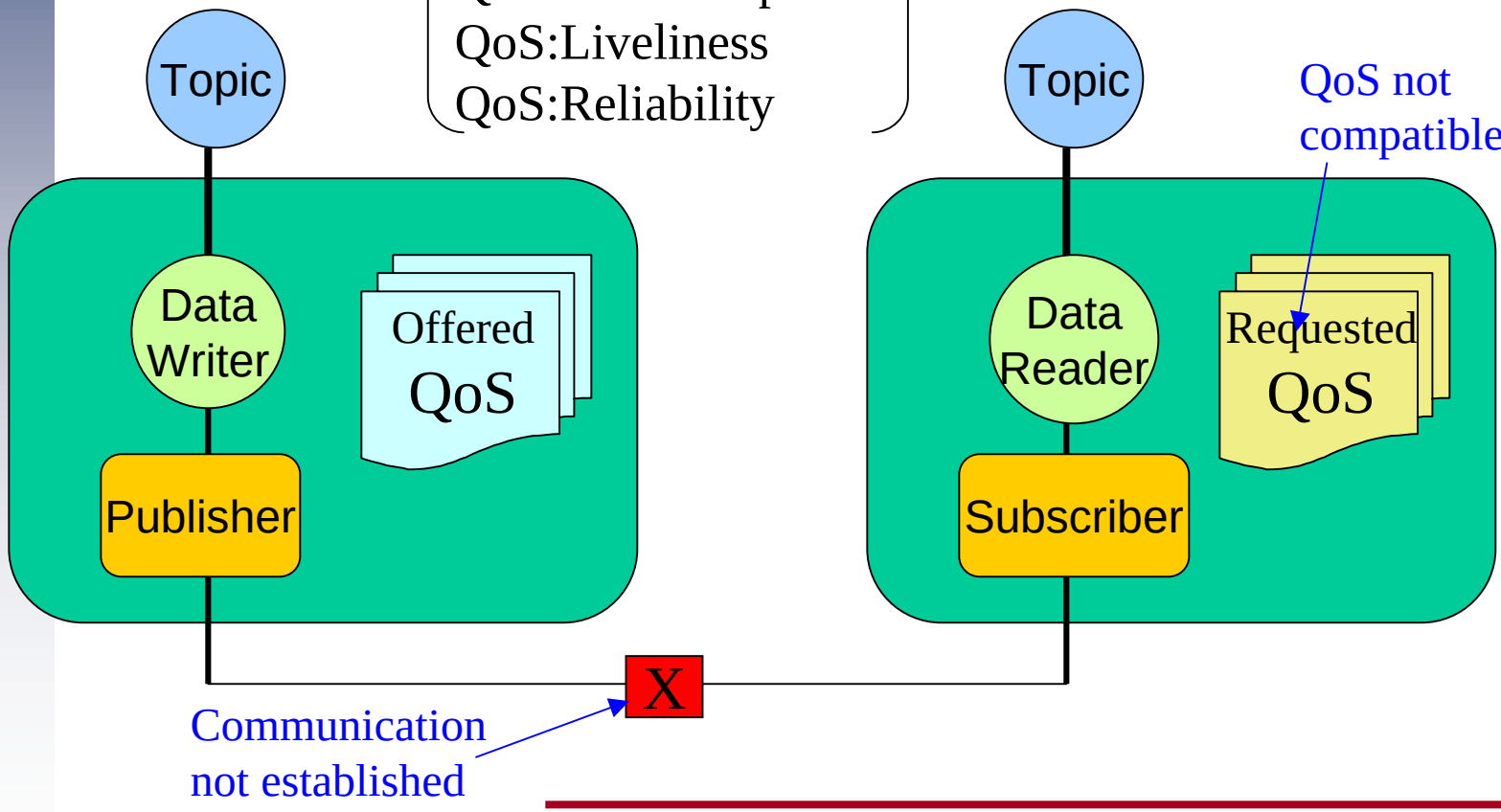
DCPS Subscription Objects (MultiTopic)



QoS Contract "Request / Offered"

QoS:Durability
QoS:Presentation
QoS:Deadline
QoS:Latency_Budget
QoS:Ownership
QoS:Liveliness
QoS:Reliability

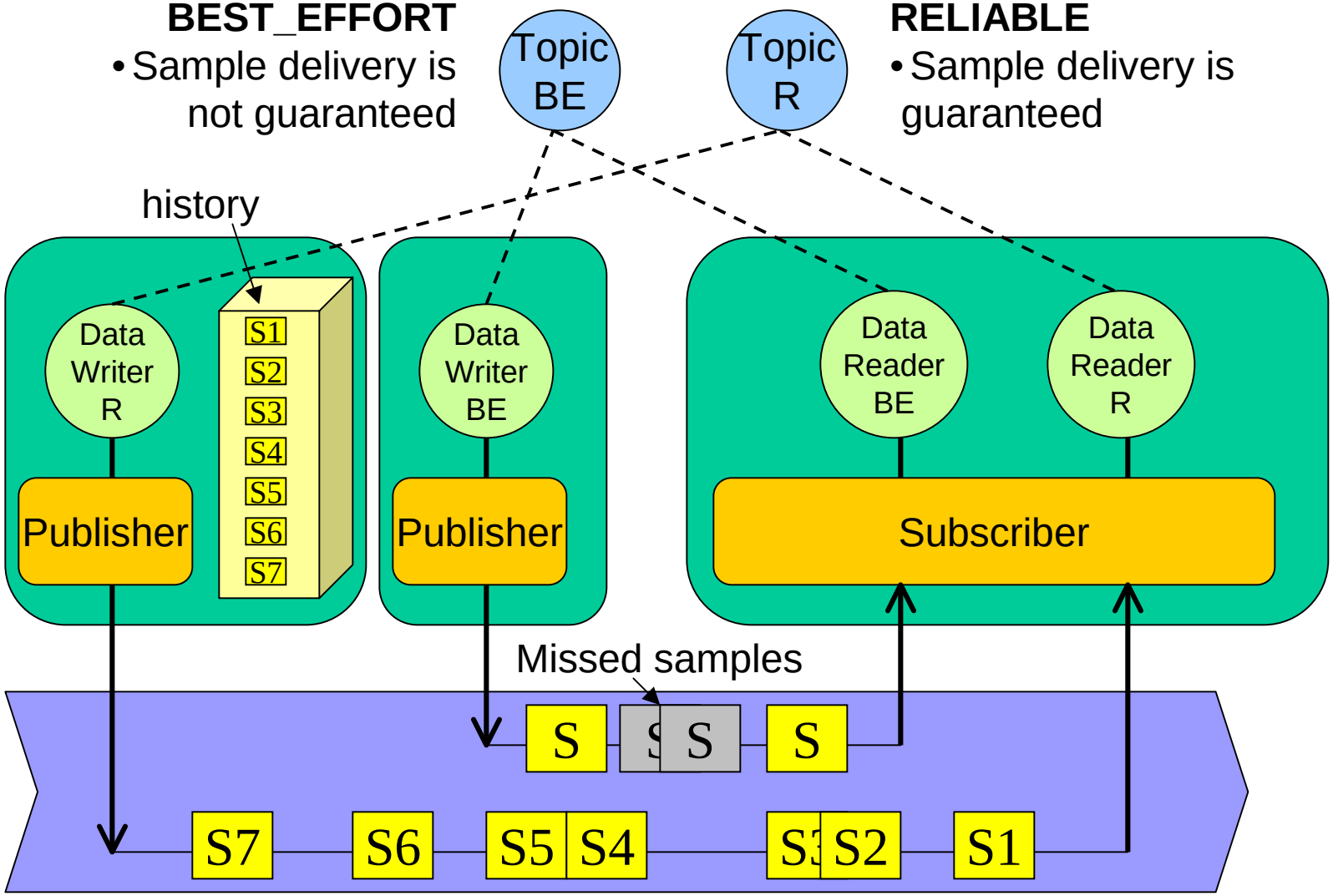
QoS Request / Offered:
Ensure that the compatible QoS parameters are set.



QoS: Reliability

BEST_EFFORT
- Sample delivery is not guaranteed

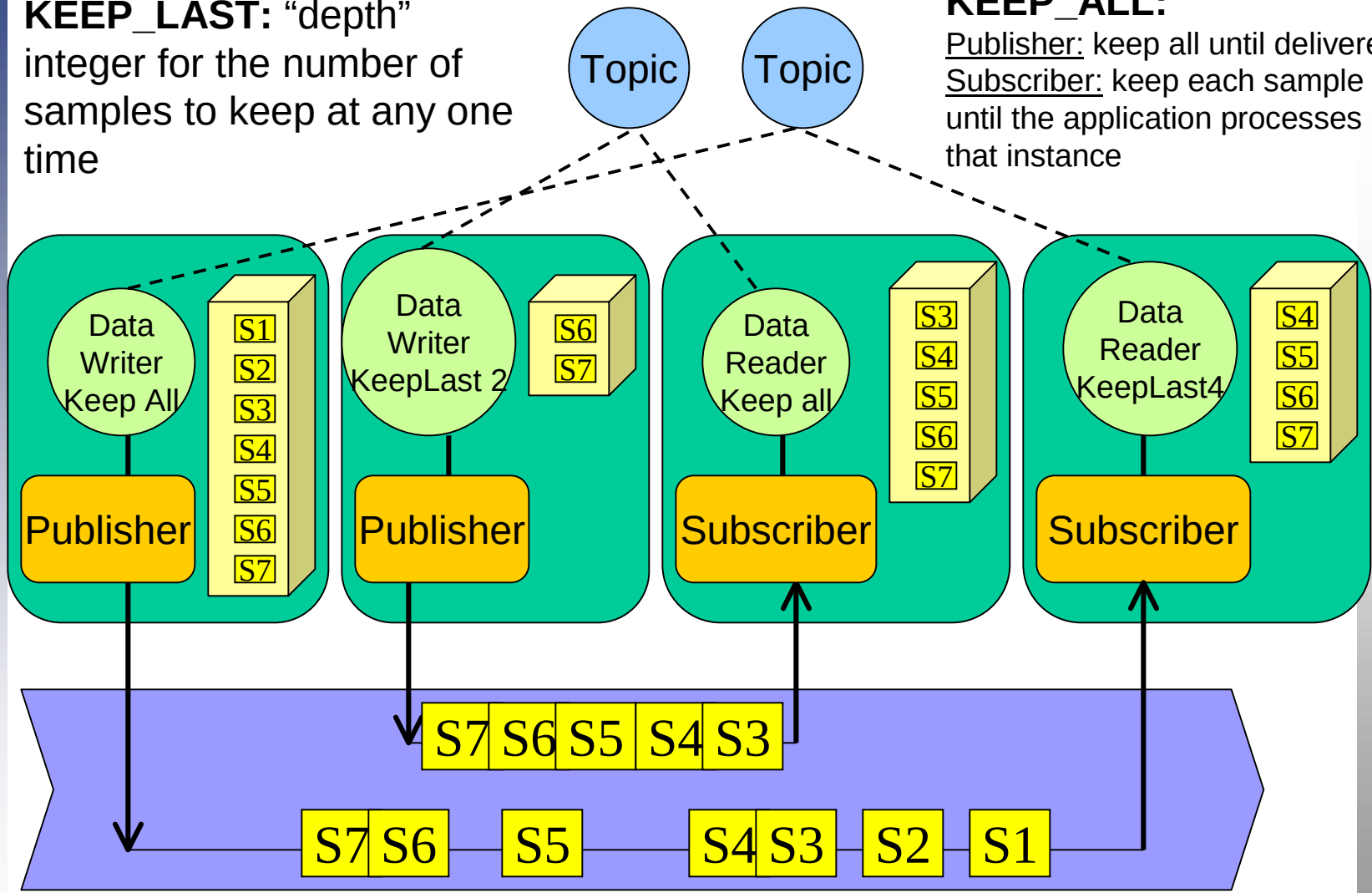
RELIABLE
- Sample delivery is guaranteed



QoS: History: Last x or All

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

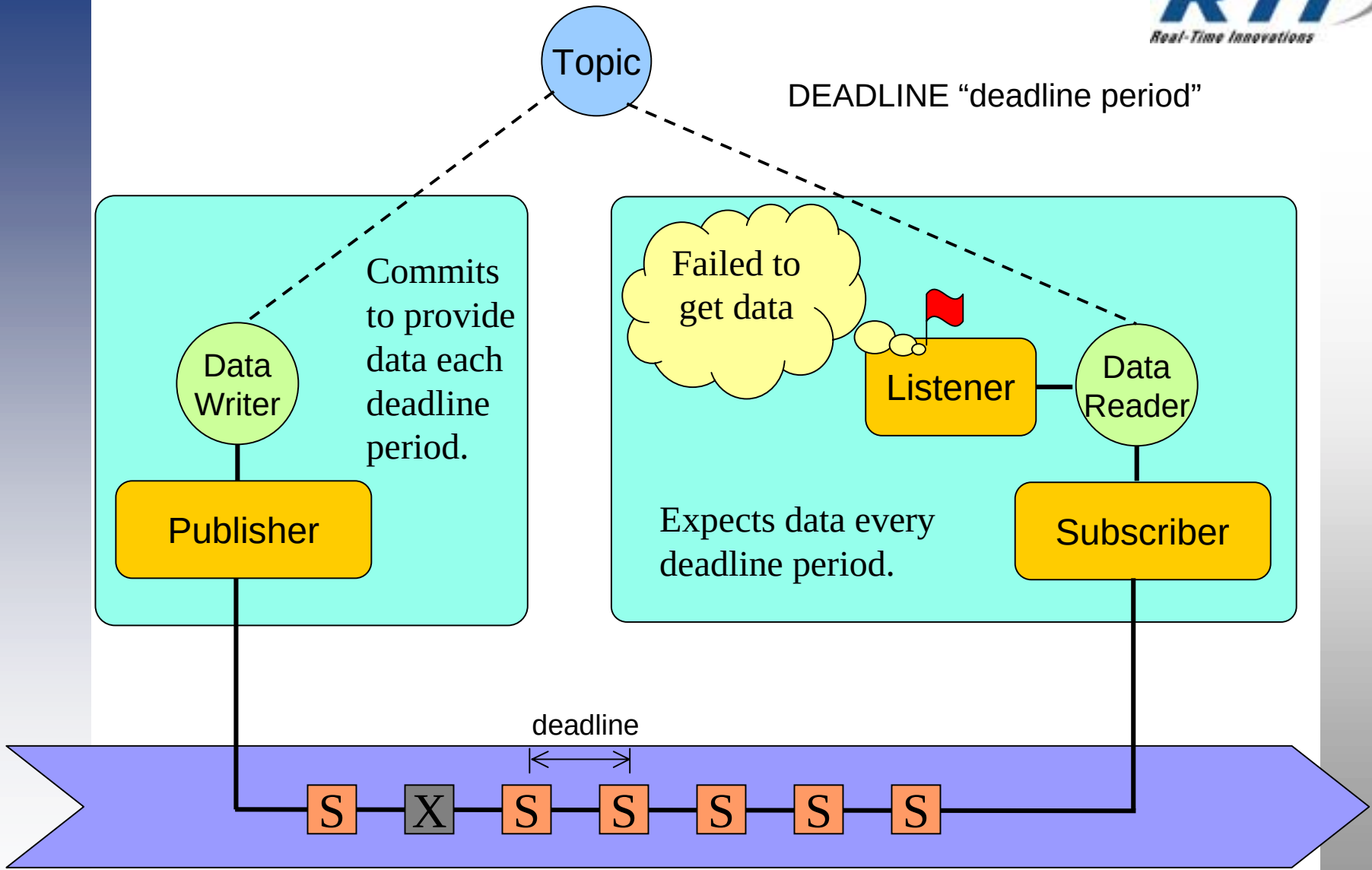
KEEP_ALL:
Publisher: keep all until delivered
Subscriber: keep each sample until the application processes that instance



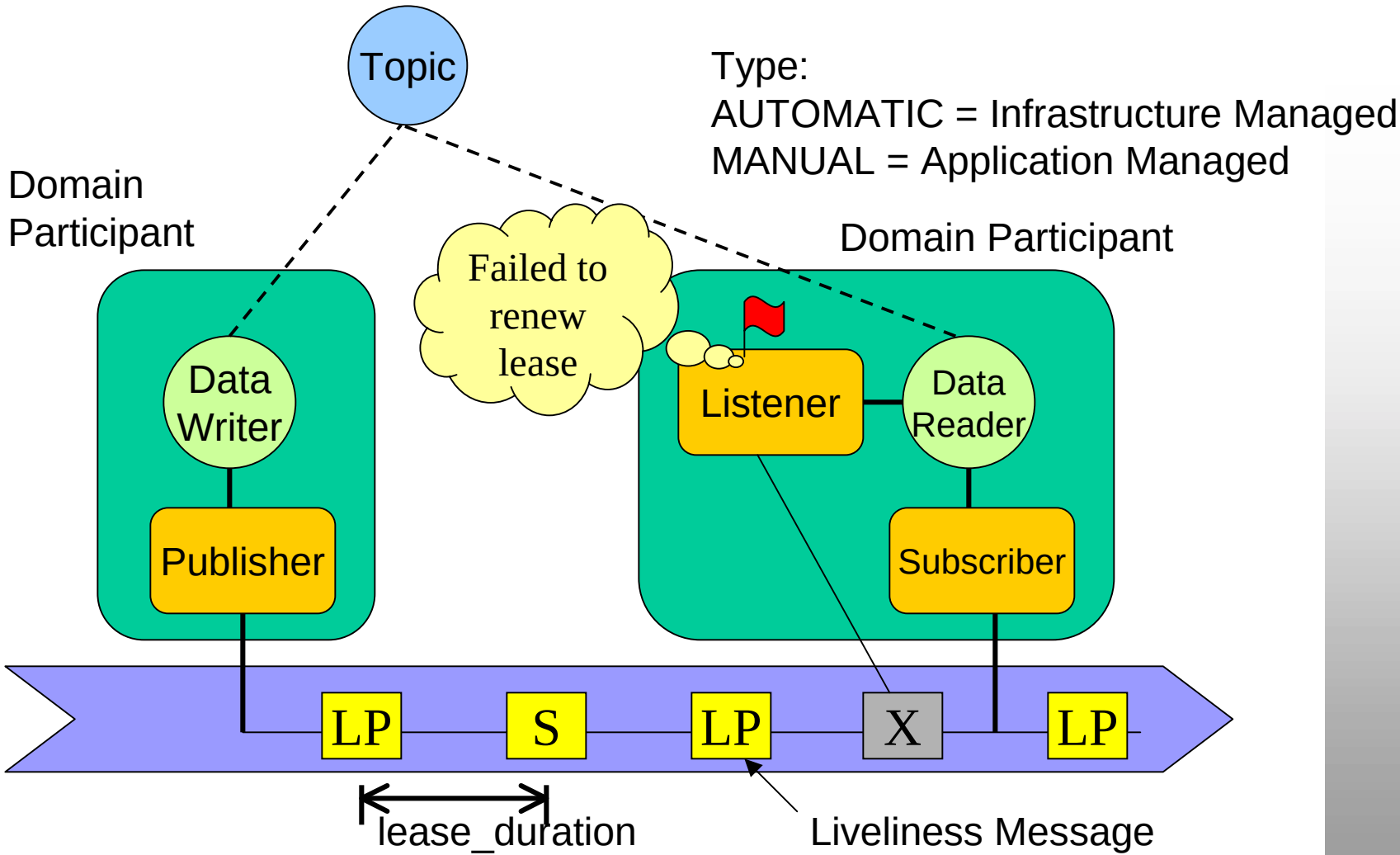
State propagation

- System state
  - Information needed to describe future behavior of the system
    - System evolution defined by state and future inputs.
  - Minimalist representation of past inputs to the system
- State variables
  - Set of data-objects whose value codifies the state of the system
- Relationship with DDS
  - DDS well suited to propagate and replicate state
  - Topic+key can be used to represent state variables
  - KEEP_LAST history QoS exactly matches semantics of state-variable propagation
- Significance: Key ingredient for fault-tolerance and also present in many RT applications

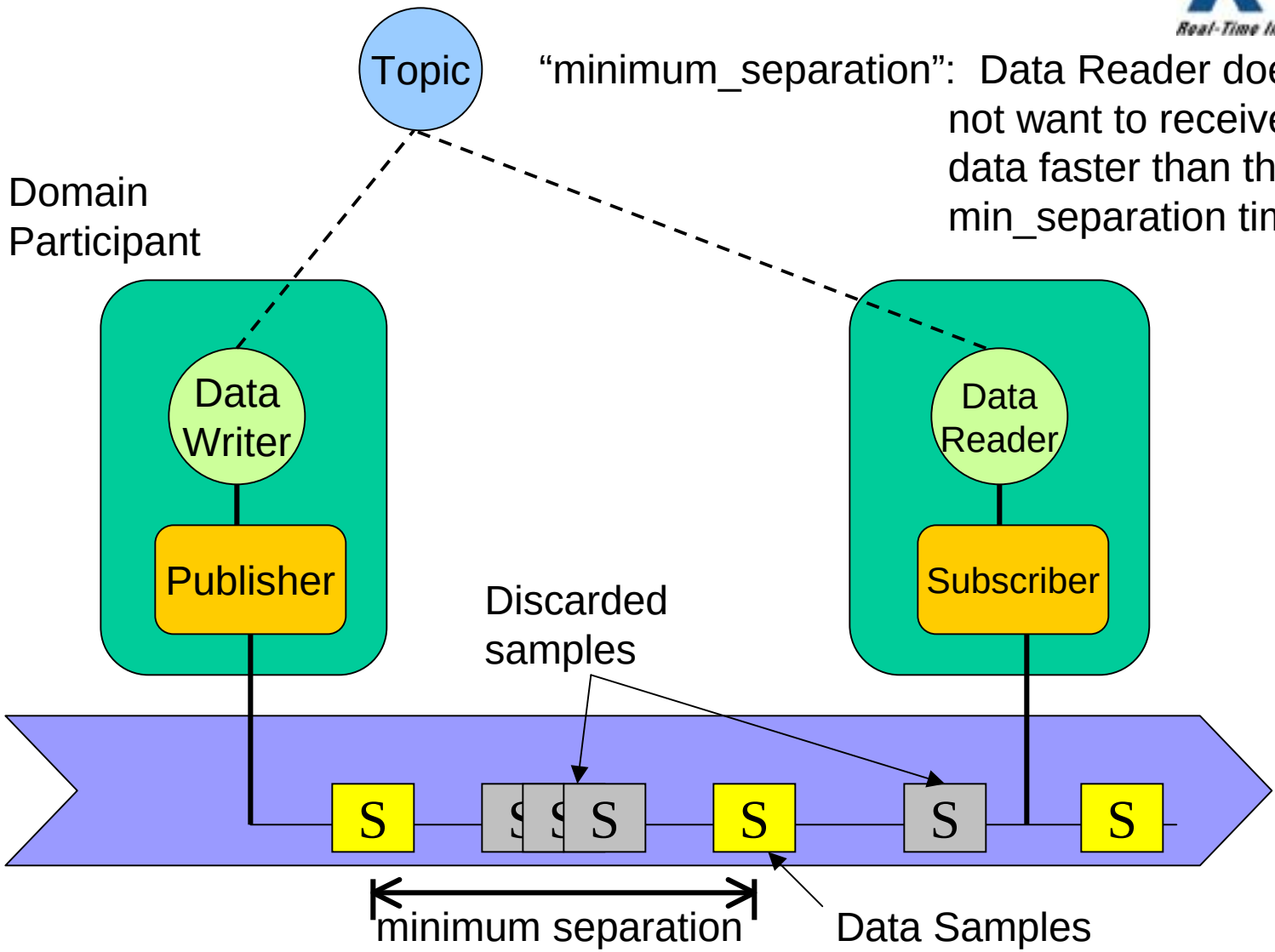
QoS: Deadline



QoS: Liveliness – Type, Duration



QoS: Time_Based_Filter



QoS: Ownership

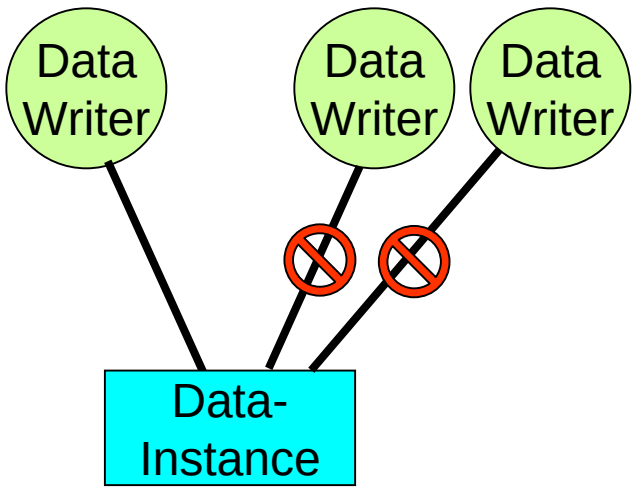
Ownership: Specifies whether more than one Data Writer can update the same instance of a data-object

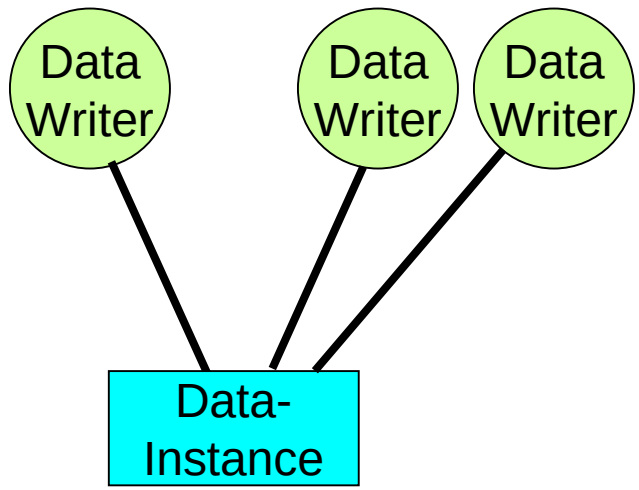
Ownership = EXCLUSIVE

- Only highest-strength data writer can update data-instance

Ownership = SHARED

- All data-writers can update data-instance

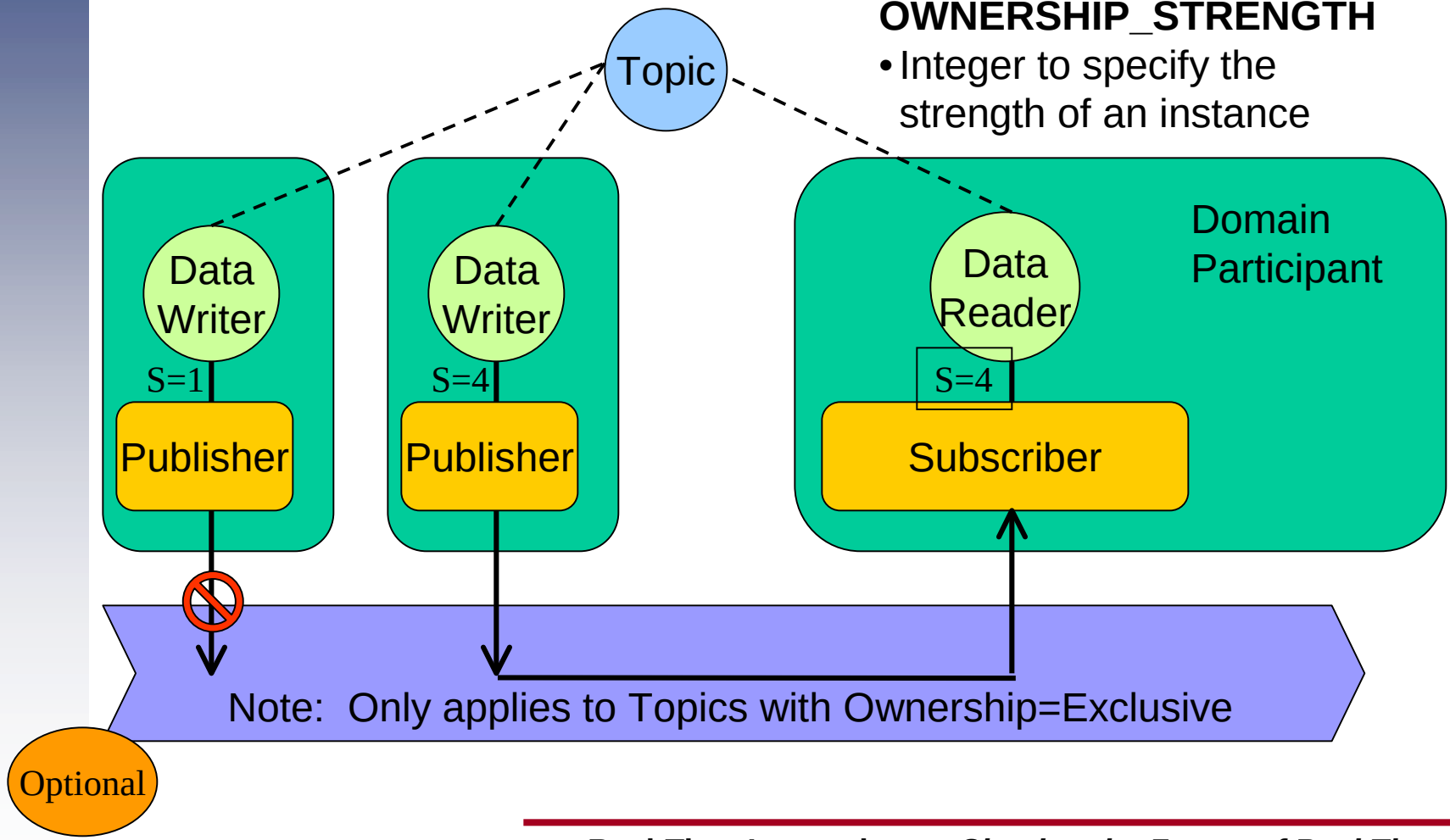




Optional

QoS: Ownership_Strength

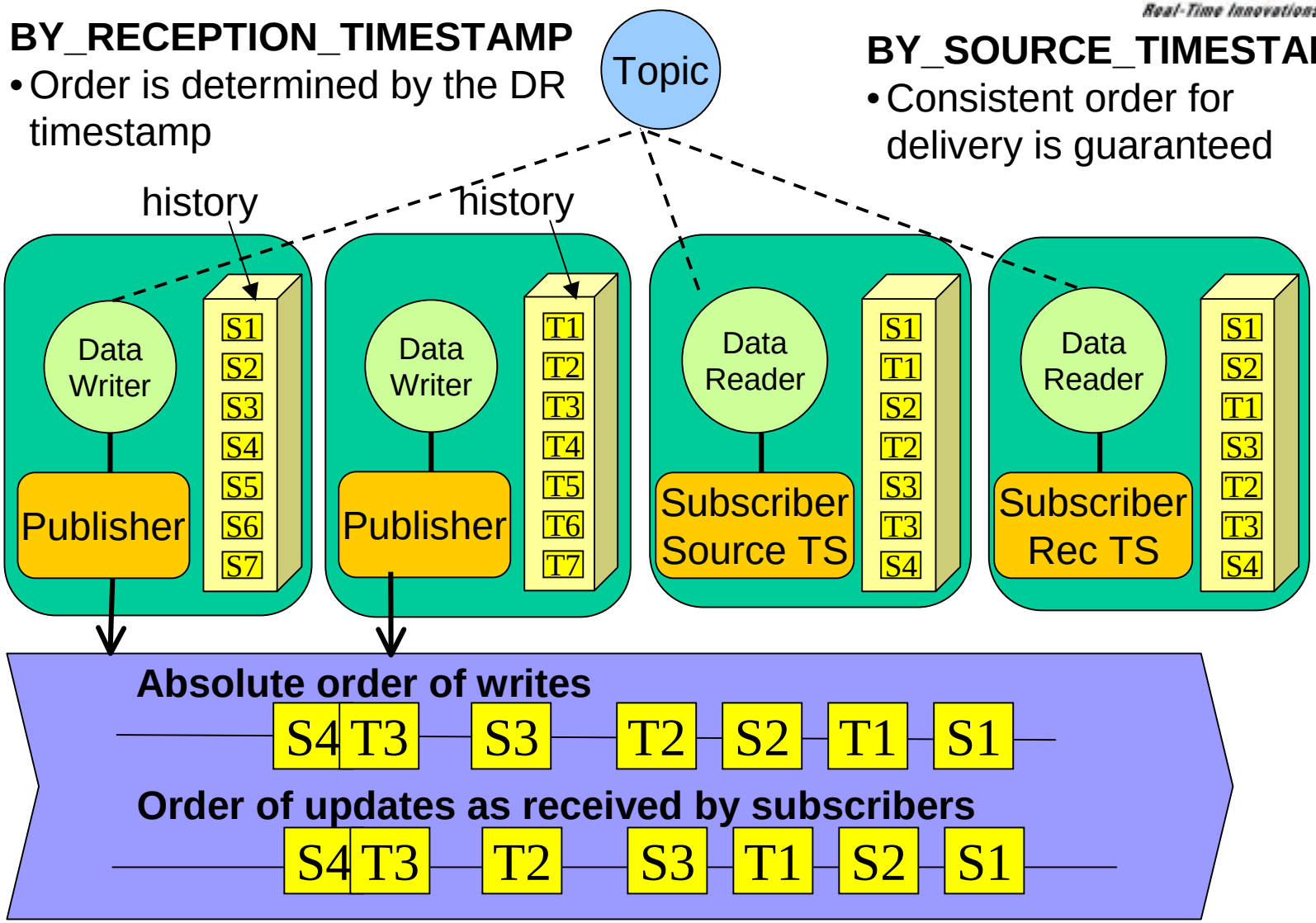
Ownership Strength: Specifies which writer is allowed to update the values of data-objects



QoS: Destination_Order

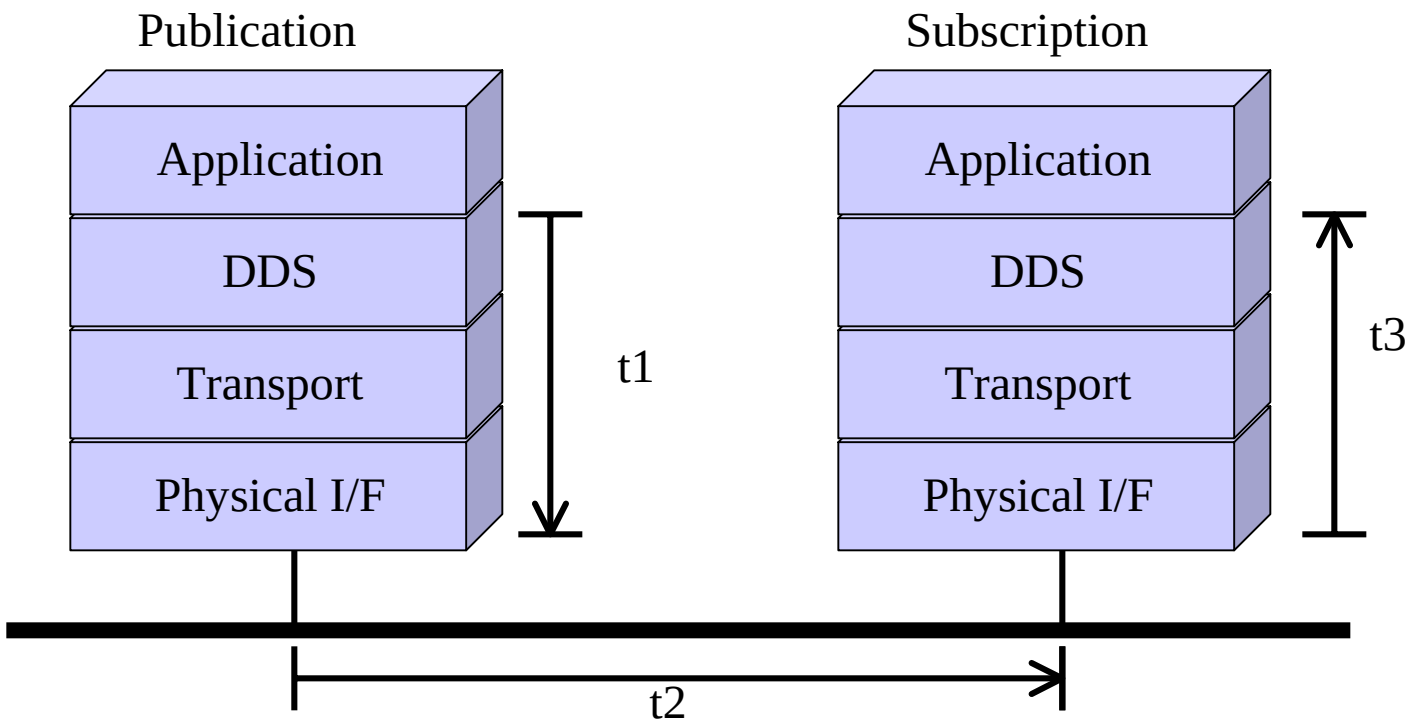
BY_RECEPTION_TIMESTAMP
- Order is determined by the DR timestamp

BY_SOURCE_TIMESTAMP
- Consistent order for delivery is guaranteed



QoS: Latency_Budget

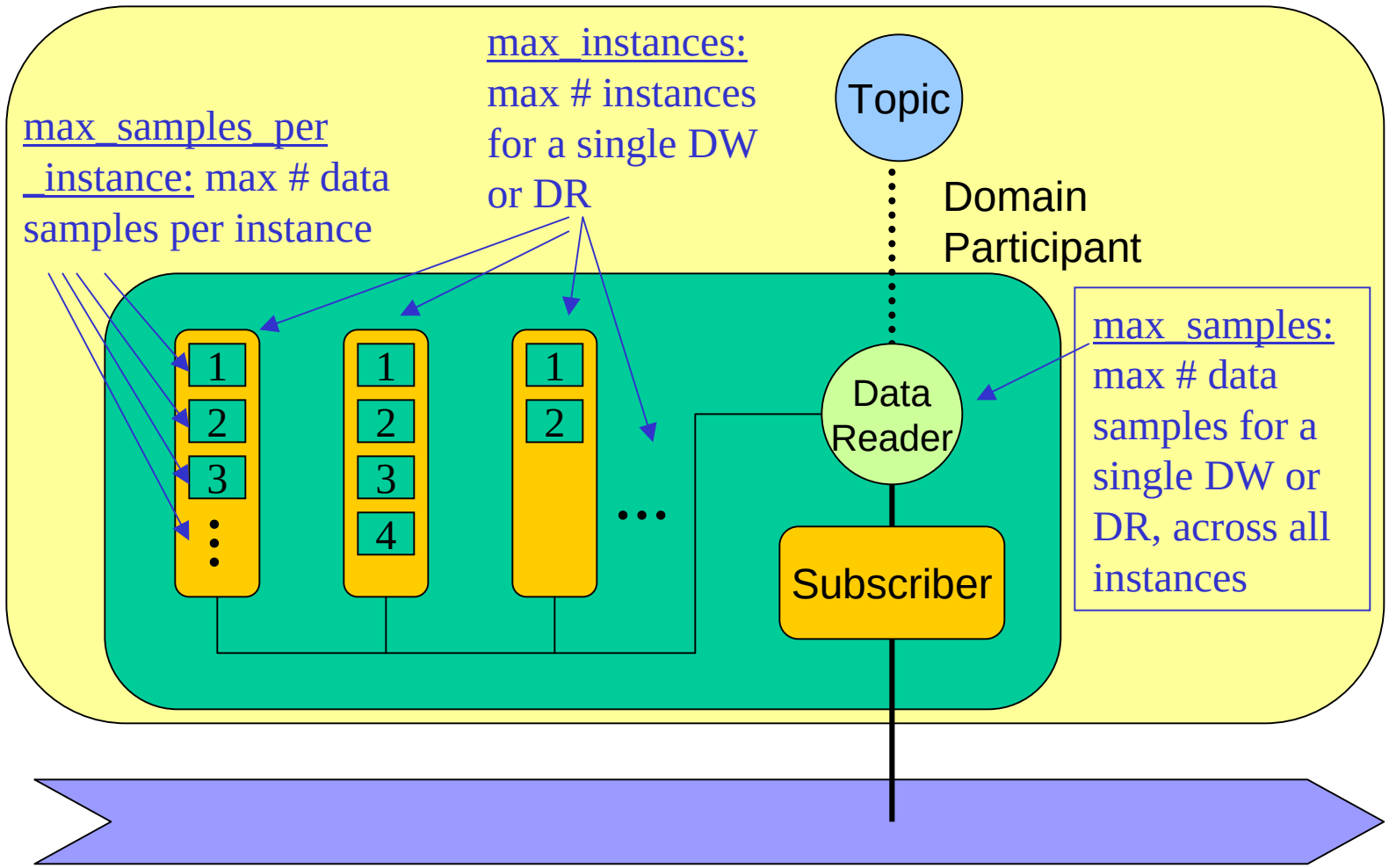
- Intended to provide time-critical information to the publisher for framework tuning where possible.
- Will not prevent data transmission and receipt.



Latency = t1 + t2 + t3

QoS: Resource_Limits

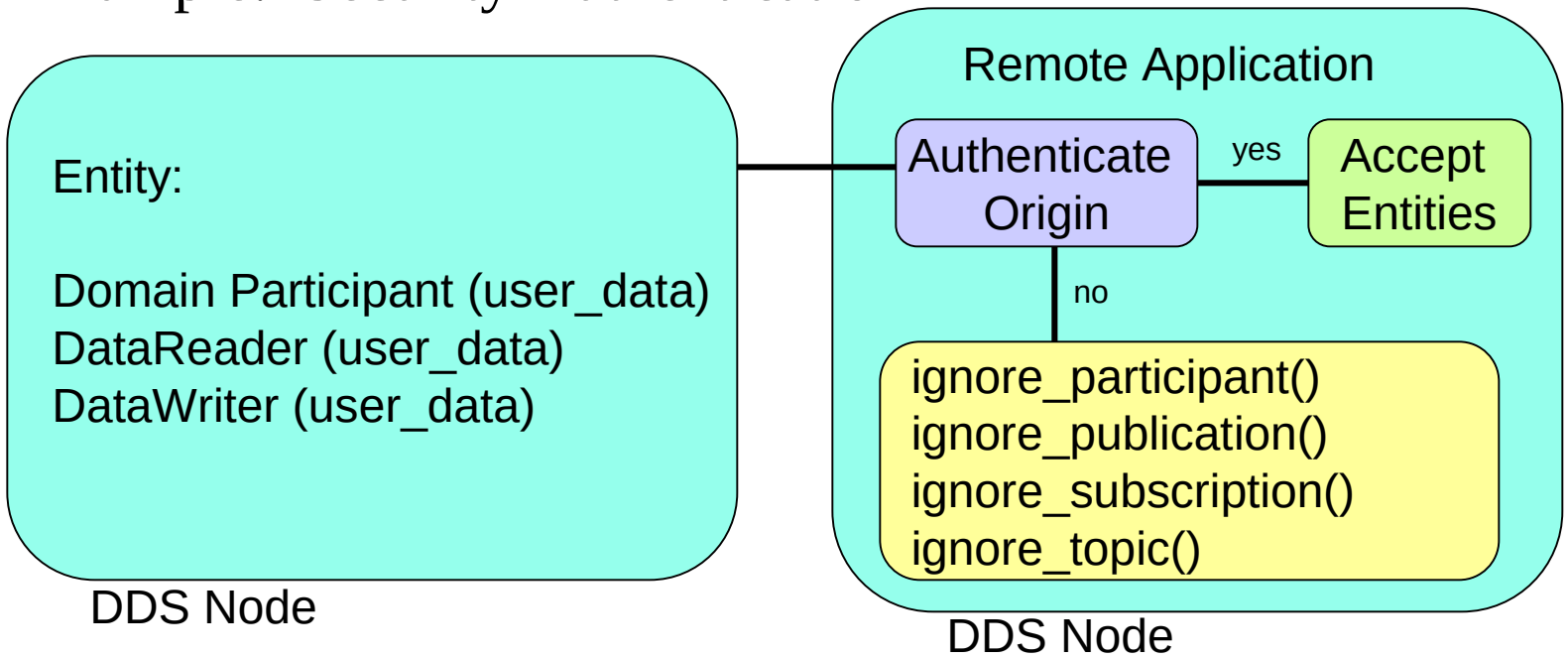
- Specifies the resources that the Service can consume to meet requested QoS



QoS: USER_DATA

Definition: User-defined portion of Topic metadata

Example: Security Authentication

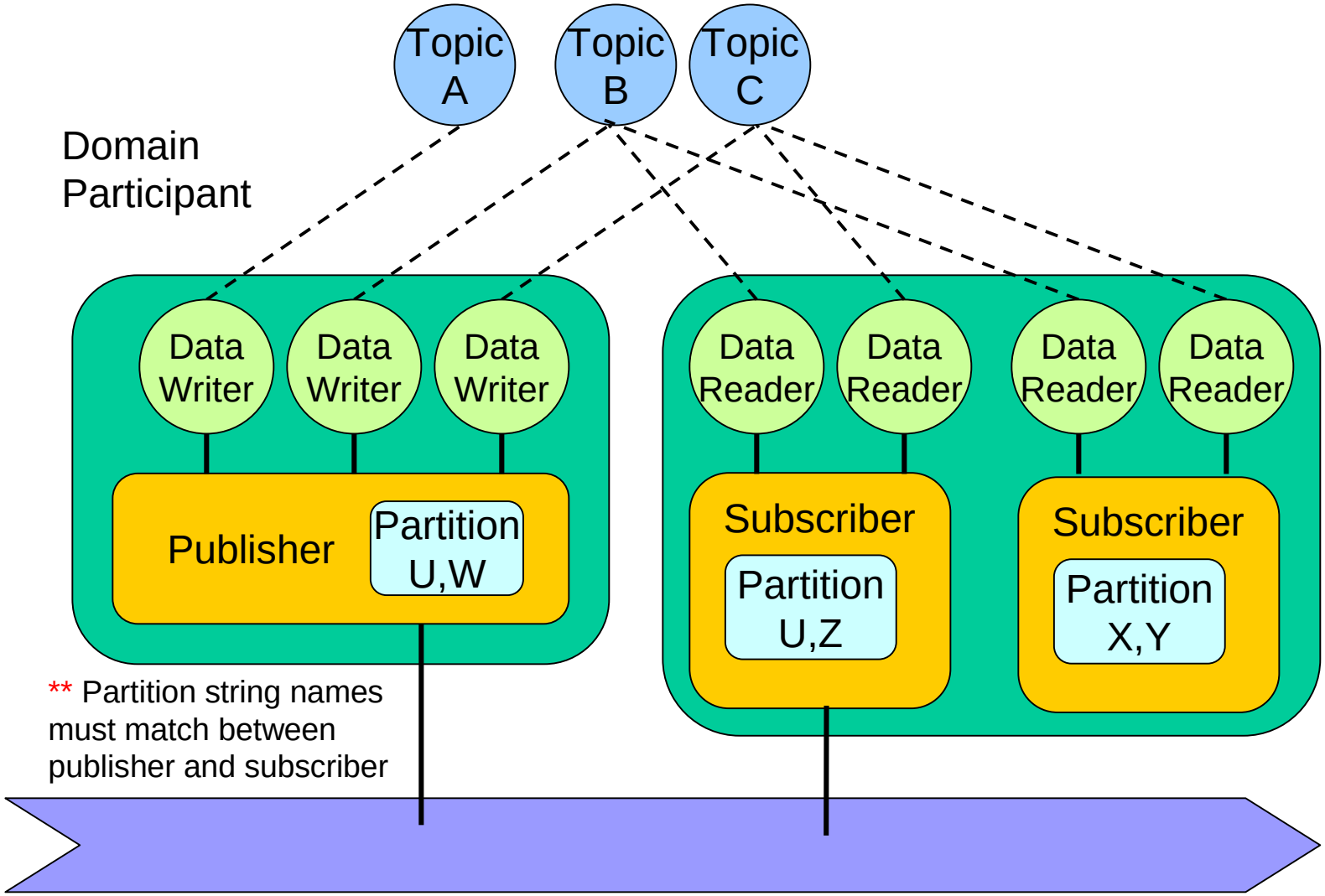


User data can be used to authenticate an origination entity.

Note: USER_DATA is contained within the DDS metadata.

QoS: Partition

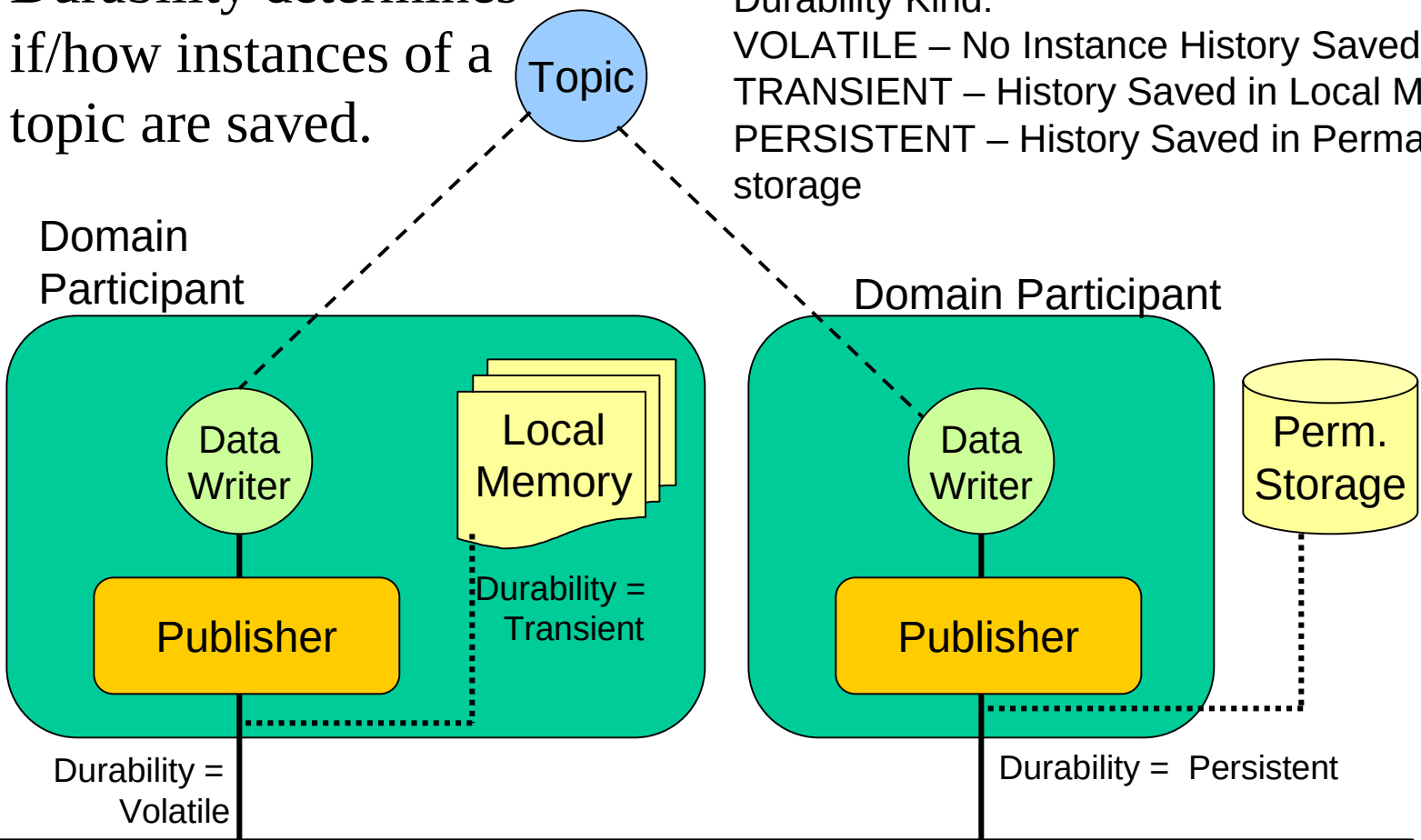
Partition: Logical “namespace” for topics



QoS: Durability

Durability determines if/how instances of a topic are saved.

Durability Kind:
VOLATILE – No Instance History Saved
TRANSIENT – History Saved in Local Memory
PERSISTENT – History Saved in Permanent storage



saved in Transient affected by QoS: History and QoS: Resource_Limits

QoS: Quality of Service

QoS Policy	Concerns	RxO	Changeable
USER_DATA	**DP,DR,DW**	**NO**	**NO**
DURABILITY	**T,DR,DW**	**YES**	**NO**
PRESENTATION	**P,S**	**YES**	**NO**
DEADLINE	**T,DR,DW**	**YES**	**YES**
LATENCY_BUDGET	**T,DR,DW**	**YES**	**YES**
OWNERSHIP	**T**	**YES**	**NO**
OWNERSHIP_STRENGTH	**DW**	**N/A**	**YES**
LIVELINESS	**T,DR,DW**	**YES**	**NO**
TIME_BASED_FILTER	**DR**	**N/A**	**YES**
PARTITION	**P,S**	**NO**	**YES**
RELIABILITY	**T,DR,DW**	**YES**	**NO**
TRANSPORT_PRIORITY	**T,DW**	**N/A**	**YES**
DESTINATION_ORDER	**T,DR**	**NO**	**NO**
HISTORY	**T,DR,DW**	**NO**	**NO**
RESOURCE_LIMITS	**T,DR,DW**	**NO**	**NO**

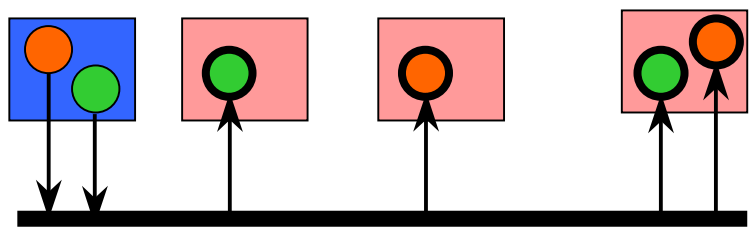
RxO: Request / Offered

QoS: Presentation

- **Governs how related data-instance changes are presented to the subscribing application.**
- **Type: Coherent Access and Ordered Access**
  - Coherent access: All changes (as defined by the Scope) are presented together.
  - Ordered access: All changes (as defined by the Scope) are presented in the same order in which they occurred.
- **Scope: Instance, Topic, or Group**
  - Instance: The scope is a single data instance change. Changes to one instance are not affected by changes to other instances or topics.
  - Topic: The scope is all instances by a single Data Writer.
  - Group: The scope is all instances by Data Writers in the same Subscriber.

DDS-DCPS Conclusion

- DDS-DCPS targets applications that need to distribute data in a real-time environment
- DDS-DCPS provides a shared “global data space” where any application can publish the data it has & subscribe to the data it needs
- DDS-DCPS provides automatic discovery (plug & play) and facilities building fault-tolerant systems
- DDS-DCPS is highly configurable by means of QoS settings. Heterogeneous systems can be easily accommodated



Thank you for your time today.

gerardo@rti.com

http://www.rti.com