

Data Distribution Service (DDS) Tutorial

dds/2006-09-01

Gerardo Pardo-Castellote - RTI

Erik Hendriks - PrismTech

26 September 2006



© 2006 RTI and THALES - All rights Reserved

THALES₁

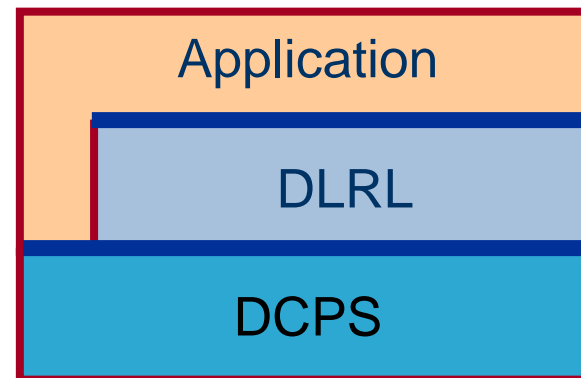


- Data Distribution Service for Real-Time Systems
 - Adopted in June 2003
 - Finalized in June 2004
 - Revised June 2005
 - Joint submission (RTI, THALES, OIS)
 - Specification of API required to facilitate the Data-Centric Publish-Subscribe communication environment for real-time distributed systems.





- DDS made of two layers
 - DCPS = Data Centric Publish/Subscribe
 - Purpose: distribute the data
 - Close to Relational model
 - DLRL = Data Local Reconstruction Layer
 - Purpose: provide an object-based model to access data 'as if' it was local



Data Distribution Service - DCPS

Outline



- Background
 - Middleware information models
 - Publish / Subscribe
 - Topic-based Publish / Subscribe
- Focus on Topics
 - Topic definition, keys
 - ContentFilteredTopic, MultiTopic
- Publication & Subscription
 - Related DDS Entities
 - DDS Publication
 - DDS Subscription
 - Dual mechanism to access incoming information
 - Listeners
 - WaitSets and Conditions
- Quality of Service

Outline



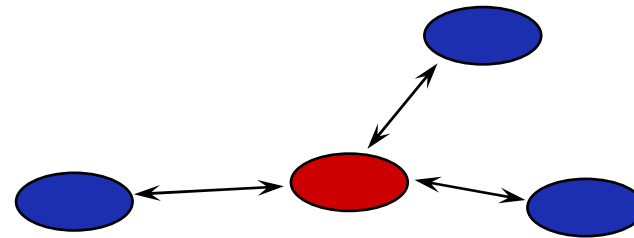
- Background
 - ▣ Middleware information models
 - ▣ Publish / Subscribe
 - ▣ Topic-based Publish / Subscribe
- Focus on Topics
 - ▣ Topic definition, keys
 - ▣ ContentFilteredTopic, MultiTopic
- Publication & Subscription
 - ▣ Related DDS Entities
 - ▣ DDS Publication
 - ▣ DDS Subscription
 - ▣ Dual mechanism to access incoming information
 - Listeners
 - WaitSets and Conditions
- Quality of Service

Middleware Information Models



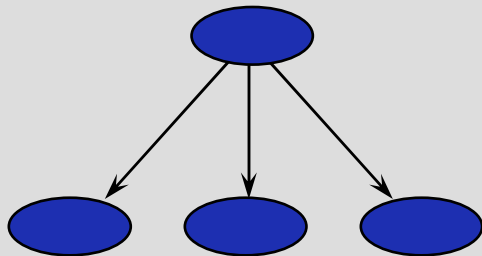
Point-to-Point

Telephone, TCP
Simple, high-bandwidth
Leads to stove-pipe systems



Client-Server

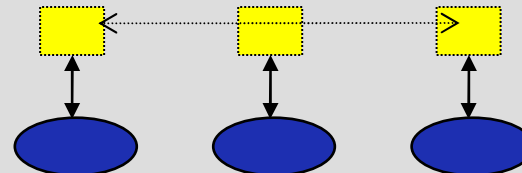
File systems, Database, RPC, CORBA, DCOM
Good if information is naturally centralized
Single point failure, performance bottlenecks



Publish/Subscribe Messaging

Magazines, Newspaper, TV
Excels at *many-to-many* communication
Excels at distributing *time-critical* information

DDS



Replicated Data

Libraries, Distributed databases
Excels at data-mining and analysis

Publish Subscribe Model



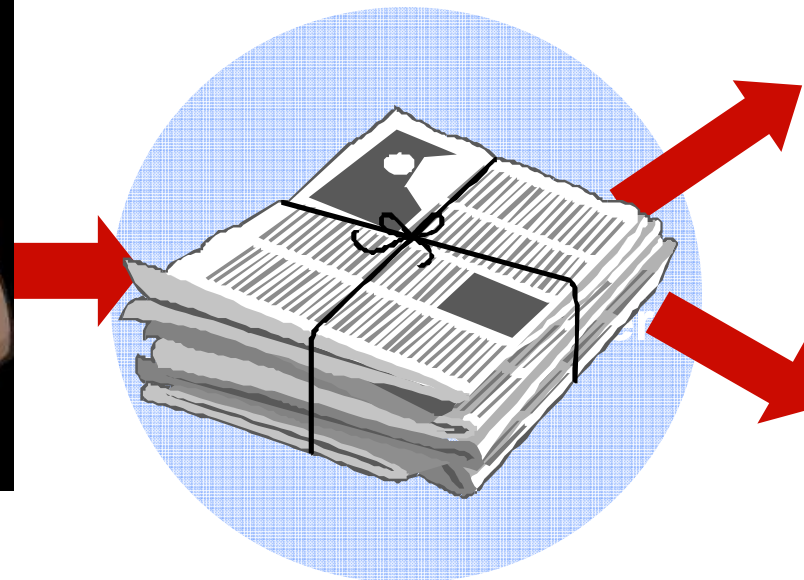
Efficient mechanism for data communications

Reporter does not need to know where subscribers live.

Subscribers do not need to know where reporter lives.



Data Producer

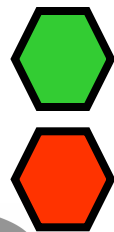
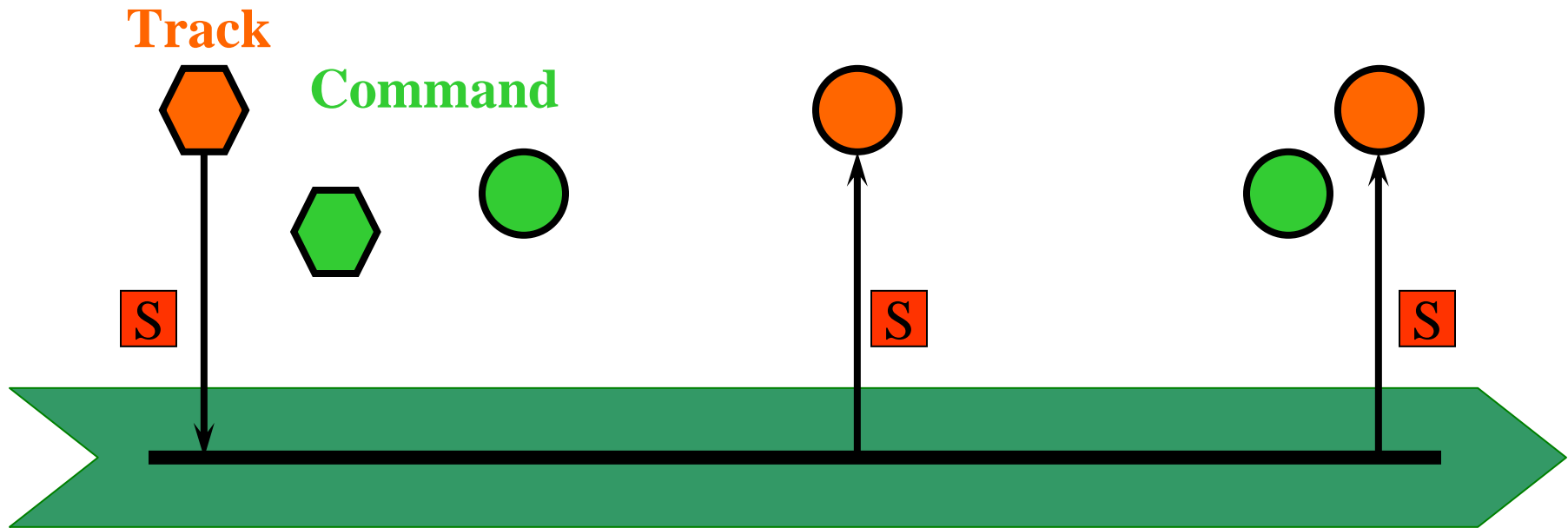


Middleware

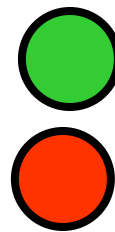


Consumers

Topic-Based Publish-Subscribe

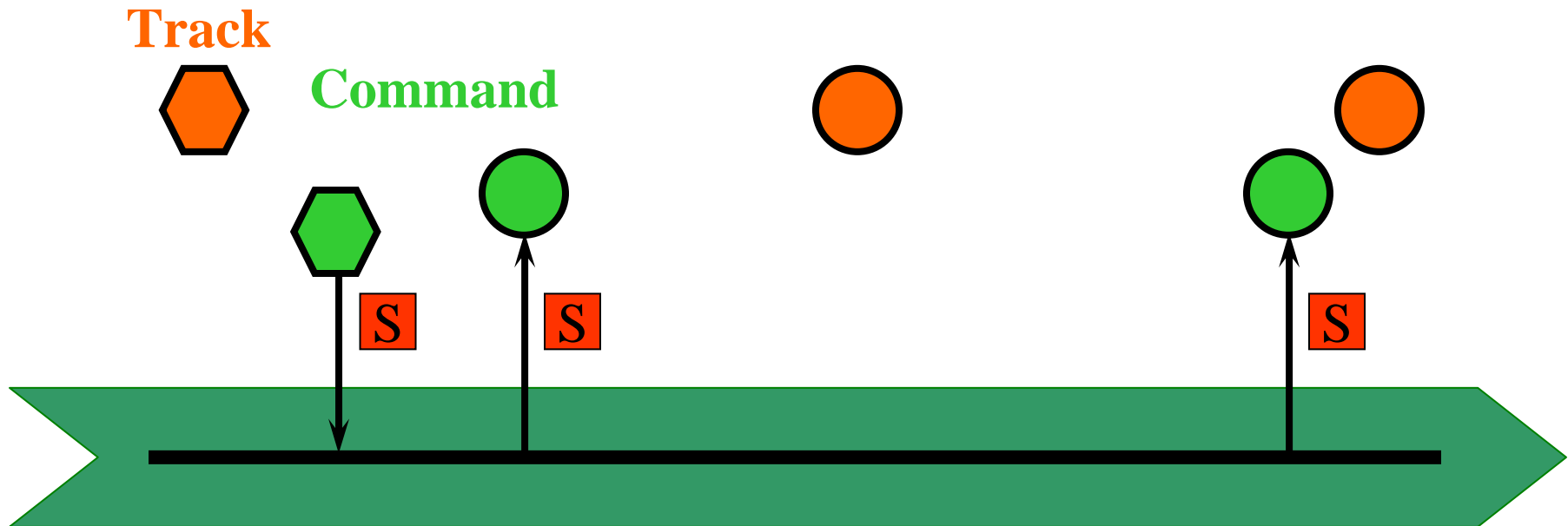


Publishers



Subscribers

Topic-Based Publish-Subscribe

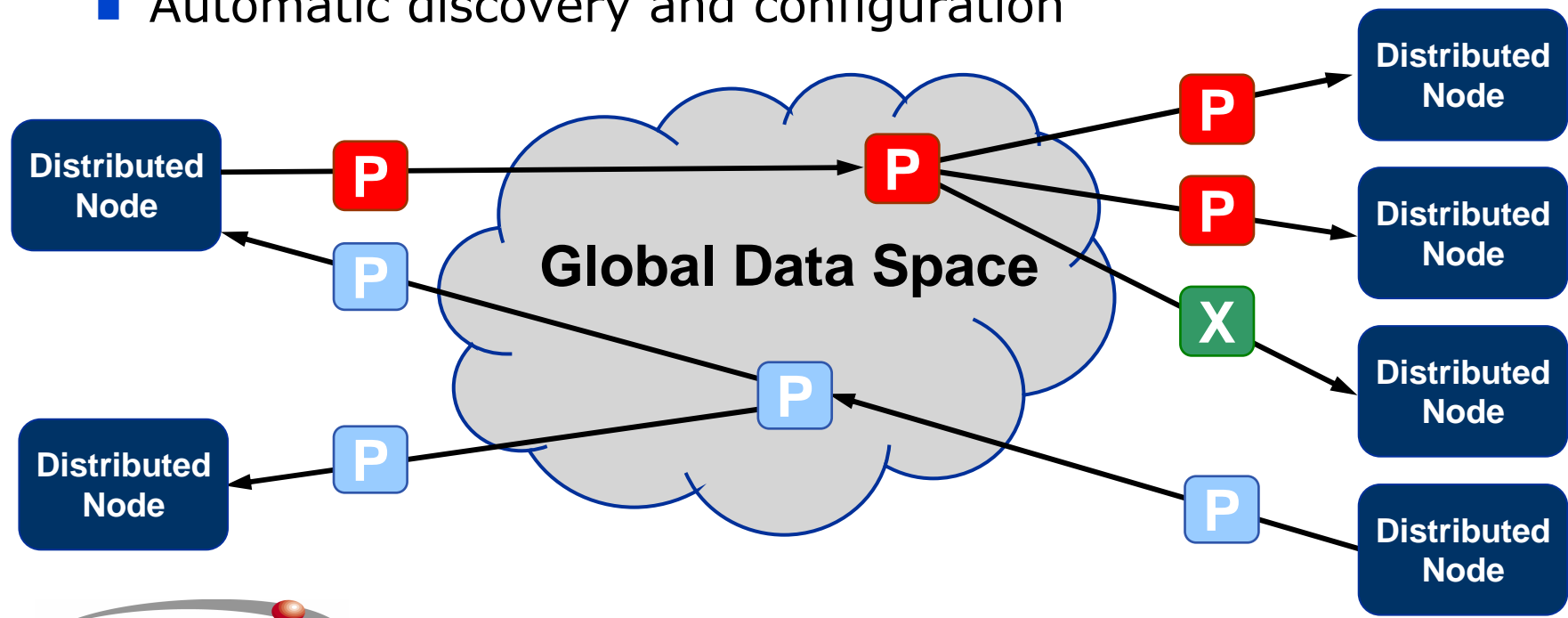


- Publish-subscribe allows infrastructure to prepare itself...
- ... Such that when the data is written it is directly sent to the subscribers



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



Outline



- Background
 - ▣ Middleware information models
 - ▣ Publish / Subscribe
 - ▣ Topic-based Publish / Subscribe
- Focus on Topics
 - ▣ Topic definition, keys
 - ▣ ContentFilteredTopic, MultiTopic
- Publication & Subscription
 - ▣ Related DDS Entities
 - ▣ DDS Publication
 - ▣ DDS Subscription
 - ▣ Dual mechanism to access incoming information
 - Listeners
 - WaitSets and Conditions
- Quality of Service



Central DDS Entity

- Topics gather all 'instances' of a given data type related to a given purpose
- Topic **Keys**
 - Needed to model dynamic objects (e.g. tracks)
 - Can dramatically decrease system size
 - Used for reliable many-to-one (i.e. ALARM topic)
- Topic **QoS**
 - Convenient way to describe information model
- Data types specified in **IDL** and can be reused
 - eg a data structure received by DDS can be used as is in a CORBA call

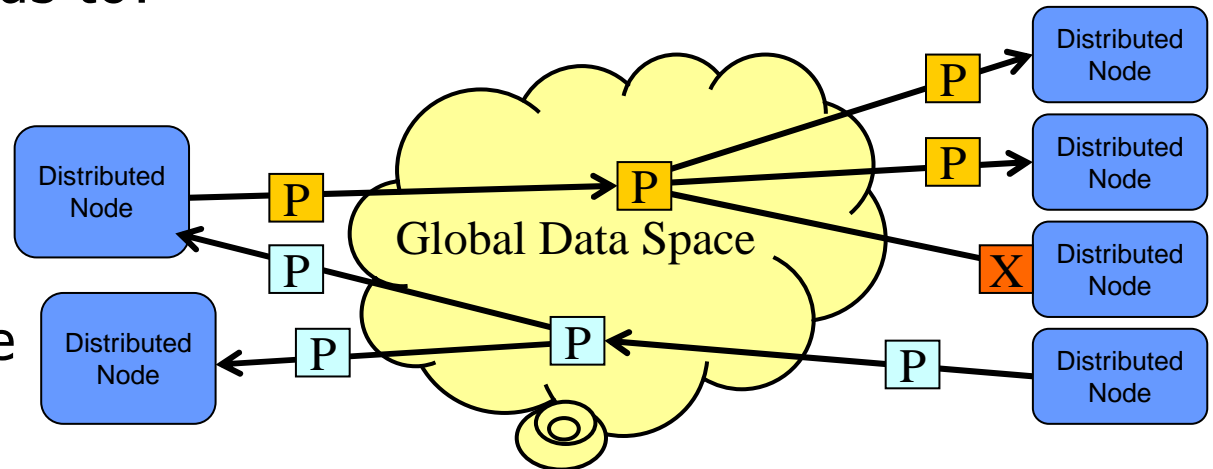
ContentFilteredTopic and **MultiTopic** control subscription scope

Topic Definition



Good topic definition leads to:

- Better interfaces
- Easier integration
- Improved scalability
- Decreased system size
- Faster startup and discovery times



Choosing the proper Topics is the central design decision

By Sender "Role"

- MixerTank3Data

By Receiver "Role"

- AirTrackCorrelator

By Message ID

- Filter23ToGUI12

By Data "Role"

- AAWTracks

By Data Type

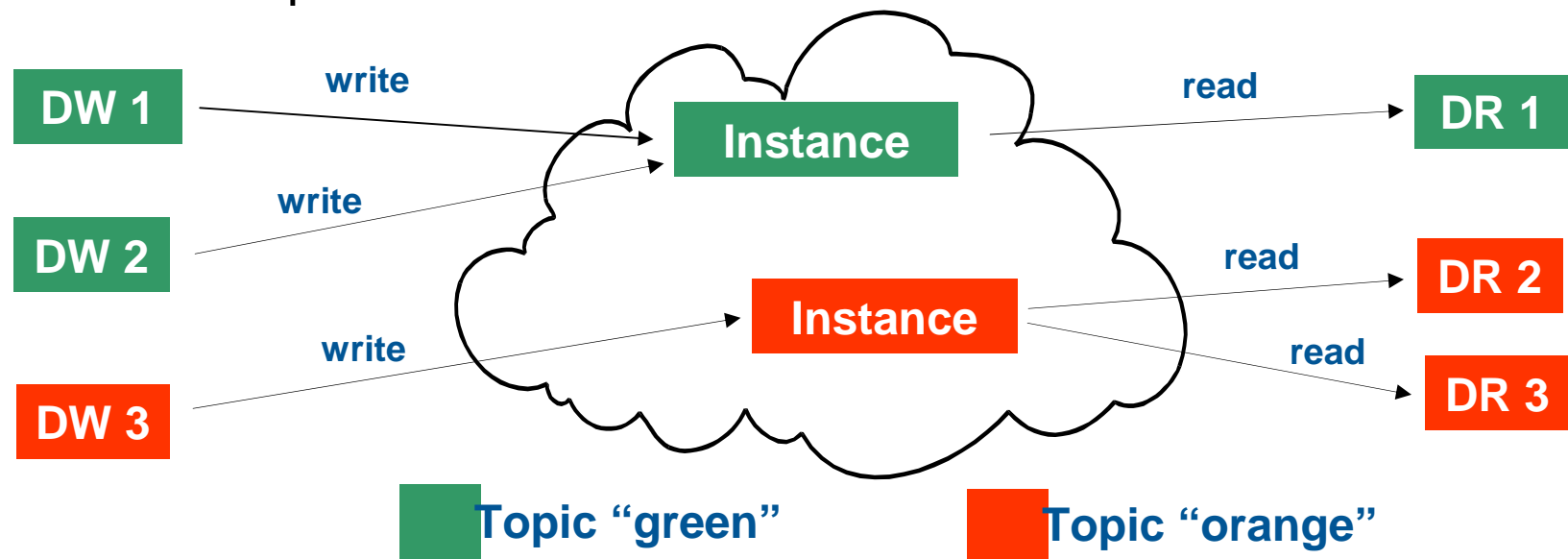
- CommandString

Example without Keys



When **not** using **keys**:

- Each Topic corresponds to a single data instance.
- A DataWriter associated with a Topic can write to the instance corresponding to That topic.
- Multiple DataWriters may write to the same instance.
- A DataReader specifies the Topic (instance) it wants to receive updates from.

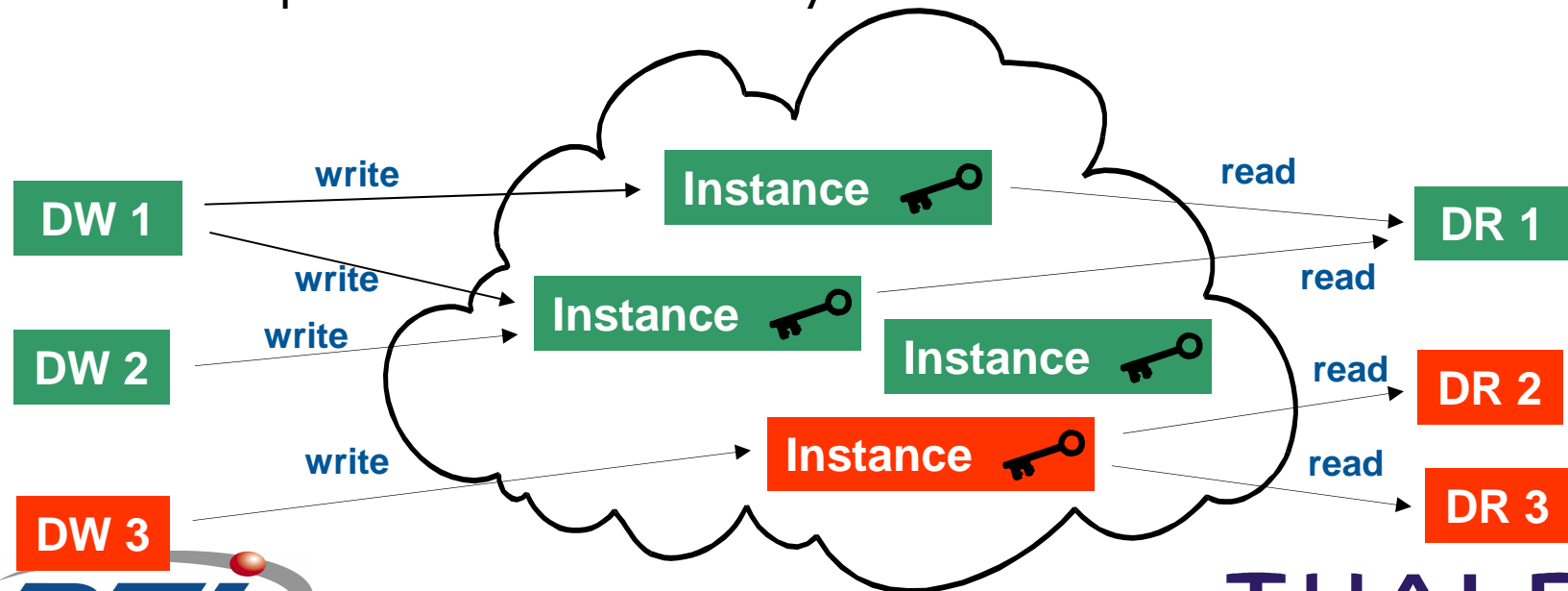


Example with Keys



Address in Global Data Space = (Topic, Key)

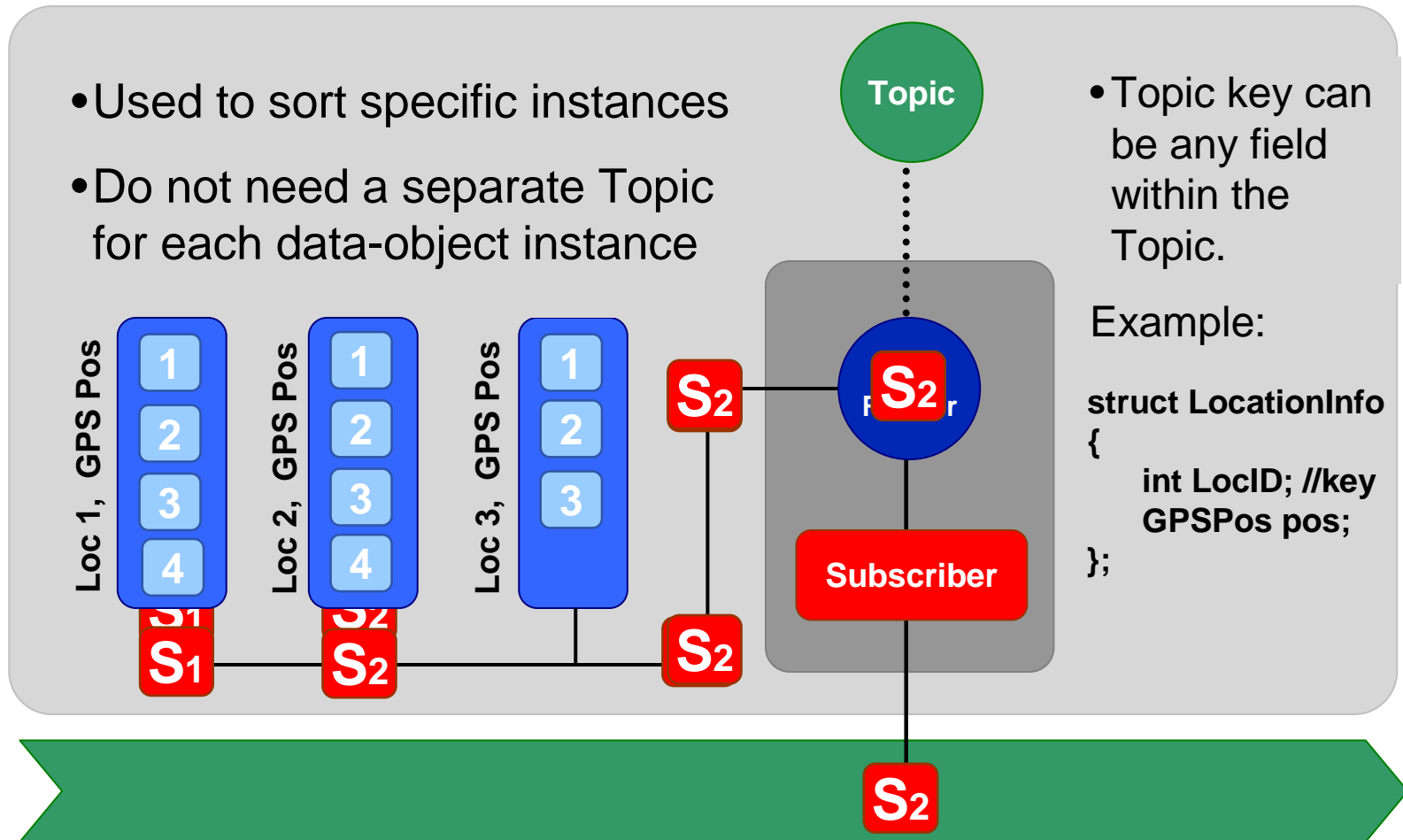
- Each Topic corresponds to multiple data instances
- Each DataWriter can write to multiple instances of a single Topic
- Multiple DataWriters may write to the same instance
- Each DataReader can receive updates from multiple instances of a single Topic
- Multiple DataReaders may read from the same instances



Data Instances Addressing: Keys



- Address in Global Data Space = (Topic, Key)
=> multiple instances of the same Topic



DDS Subscription (ContentFilteredTopic)

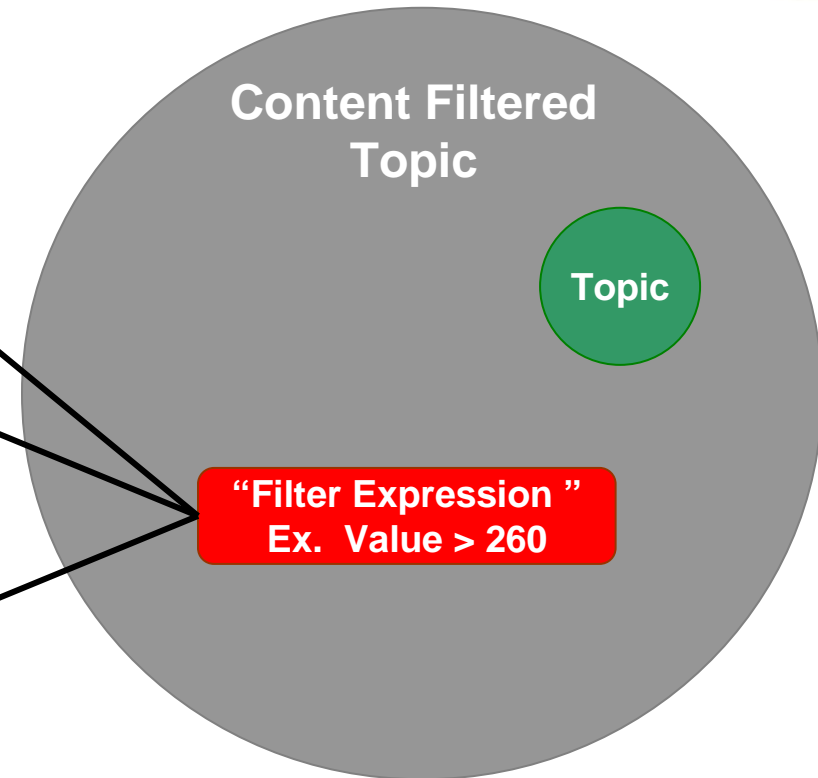


Topic Instances in Domain

Instance 1	Value = 249
Instance 2	Value = 230
Instance 3	Value = 275
Instance 4	Value = 262
Instance 5	Value = 258
Instance 6	Value = 261
Instance 7	Value = 259



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



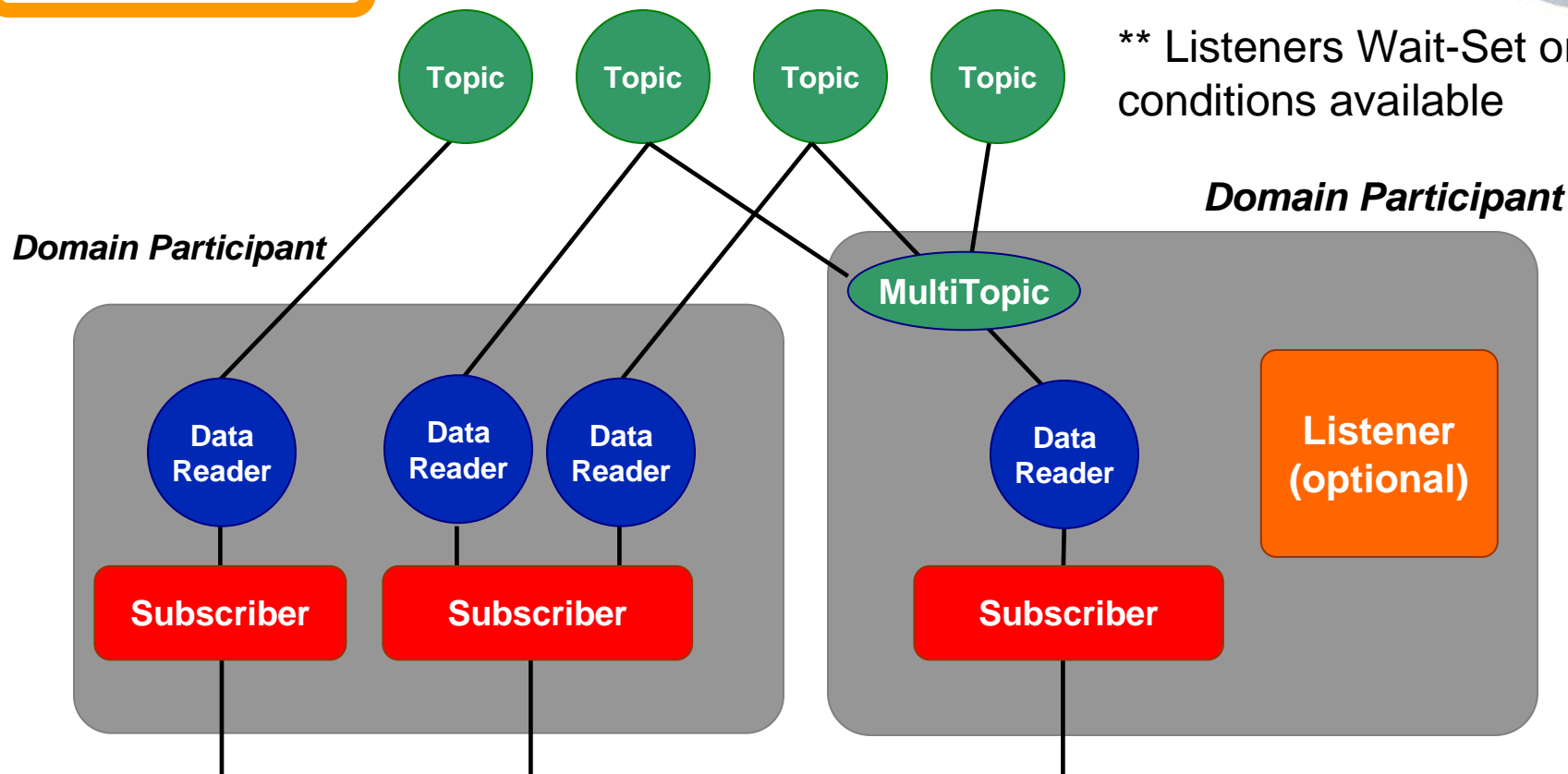
optional

DDS Subscription (MultiTopic)



optional

** Listeners Wait-Set or conditions available

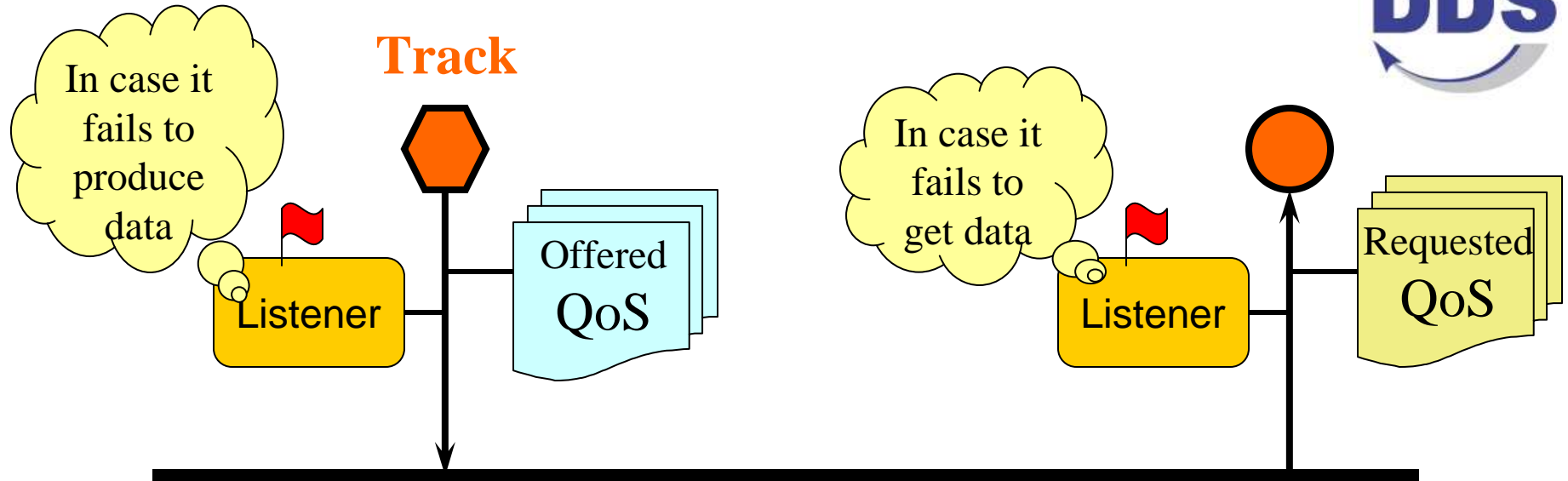


Outline



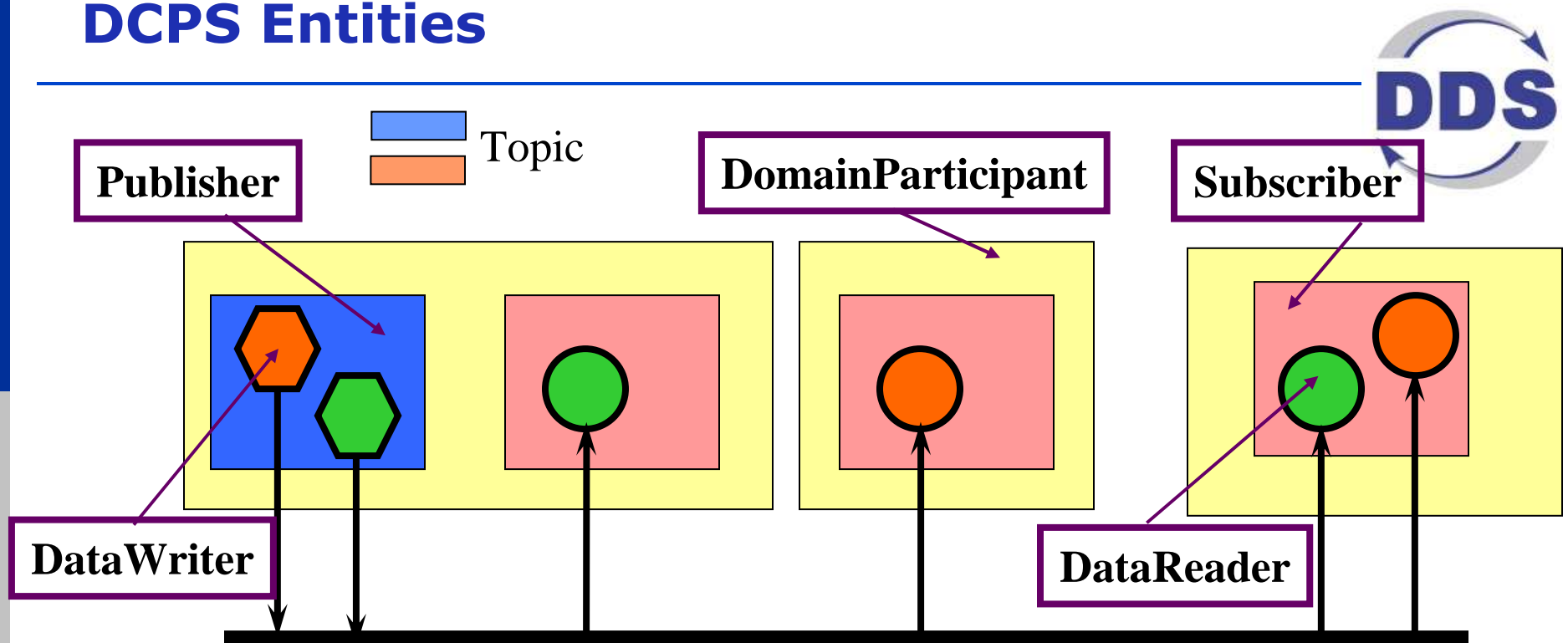
- Background
 - ▣ Middleware information models
 - ▣ Publish / Subscribe
 - ▣ Topic-based Publish / Subscribe
- Focus on Topics
 - ▣ Topic definition, keys
 - ▣ ContentFilteredTopic, MultiTopic
- Publication & Subscription
 - ▣ Related DDS Entities
 - ▣ DDS Publication
 - ▣ DDS Subscription
 - ▣ Dual mechanism to access incoming information
 - Listeners
 - WaitSets and Conditions
- Quality of Service

DDS Communication Model



- Publisher declares information it has by specifying the Topic...
 - ...and the offered QoS contract
 - ...and an associated listener to be alerted of any significant status changes
- Subscriber declares information it wants by specifying 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

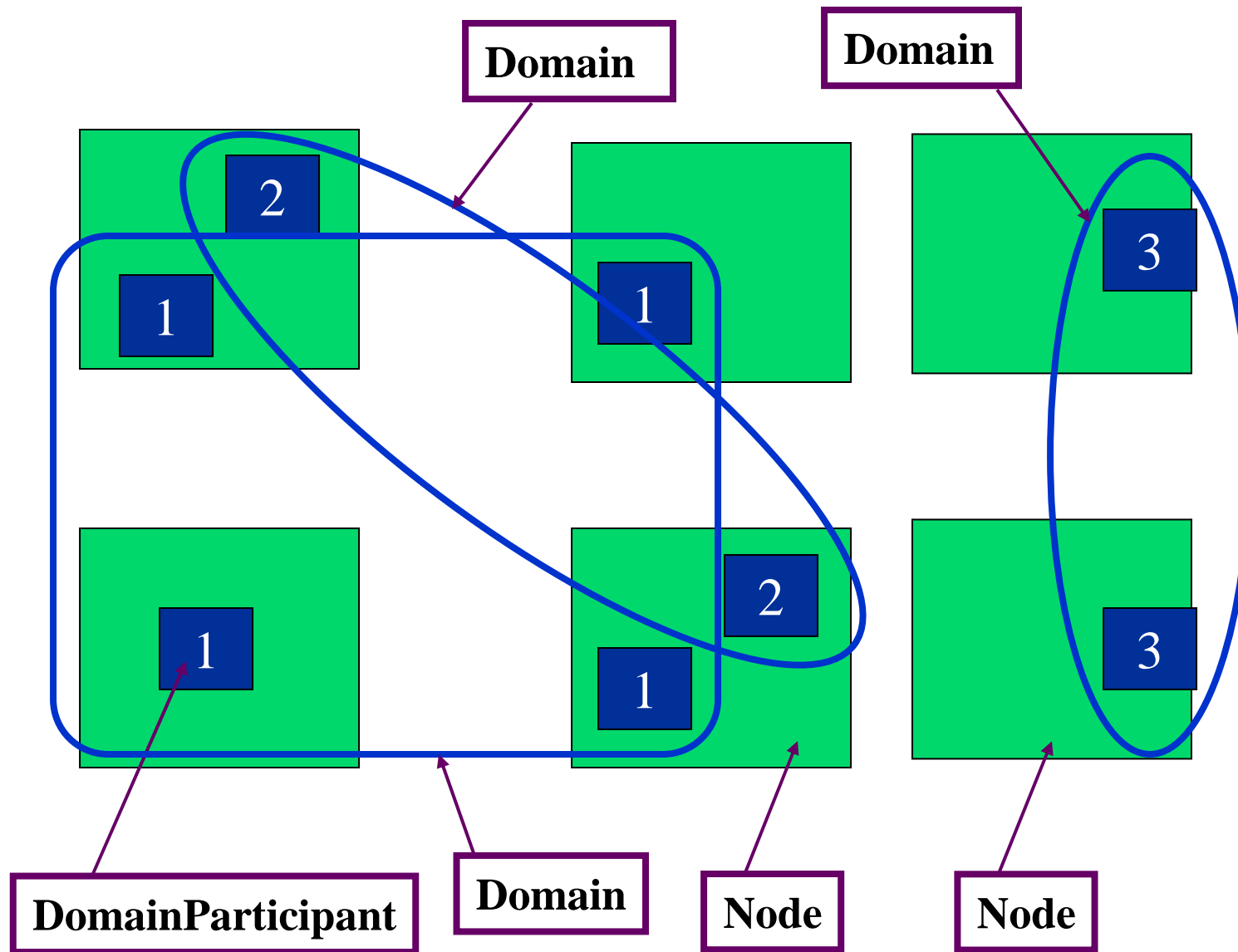
DataWriter ~ Accessor to write typed data on a particular Topic

Publisher ~ Aggregation of DataWriter objects
Responsible for disseminating information.

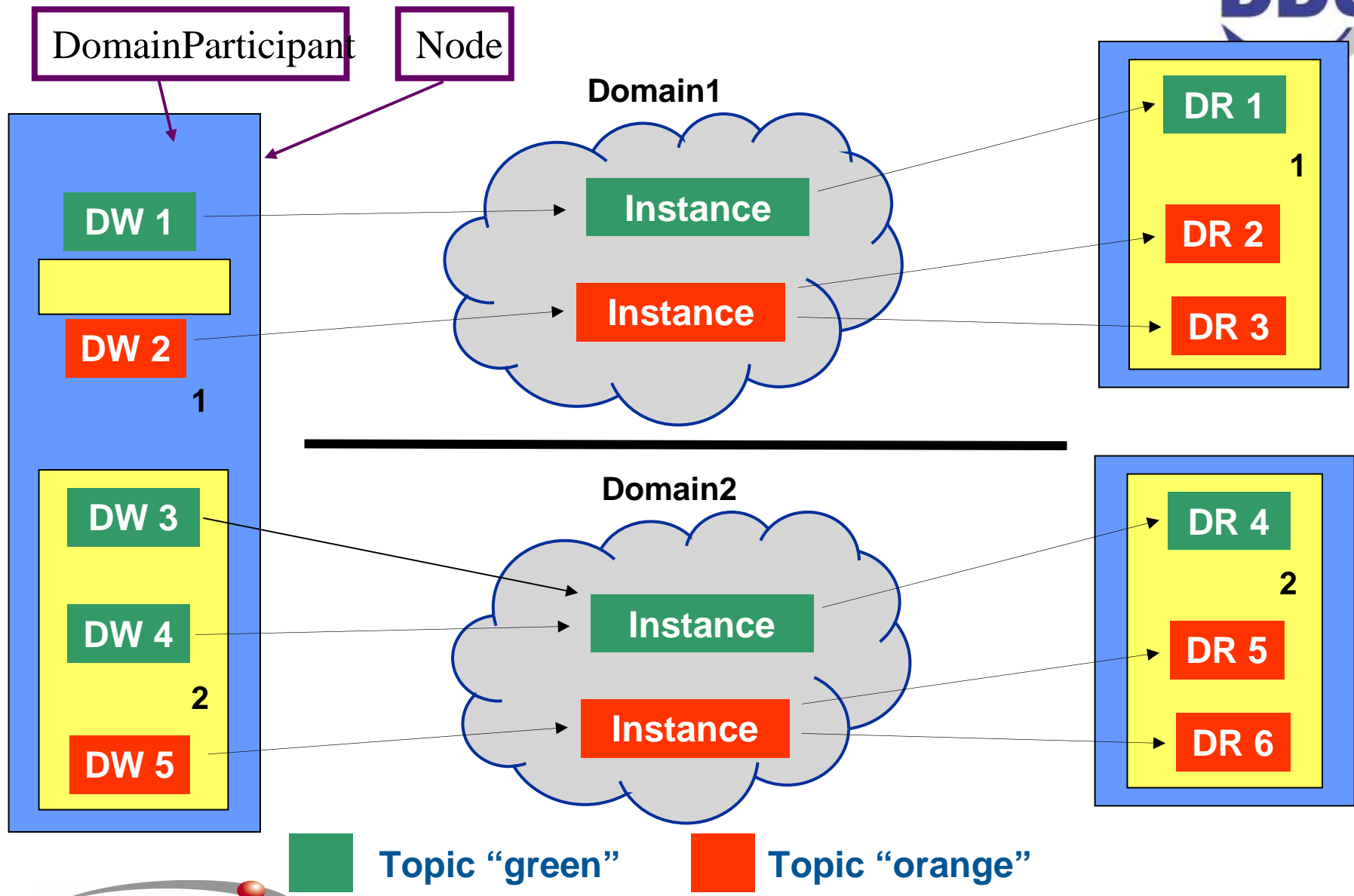
DataReader ~ Accessor to read typed data regarding a specific Topic

Subscriber ~ Aggregation of DataReader objects
Responsible for receiving information

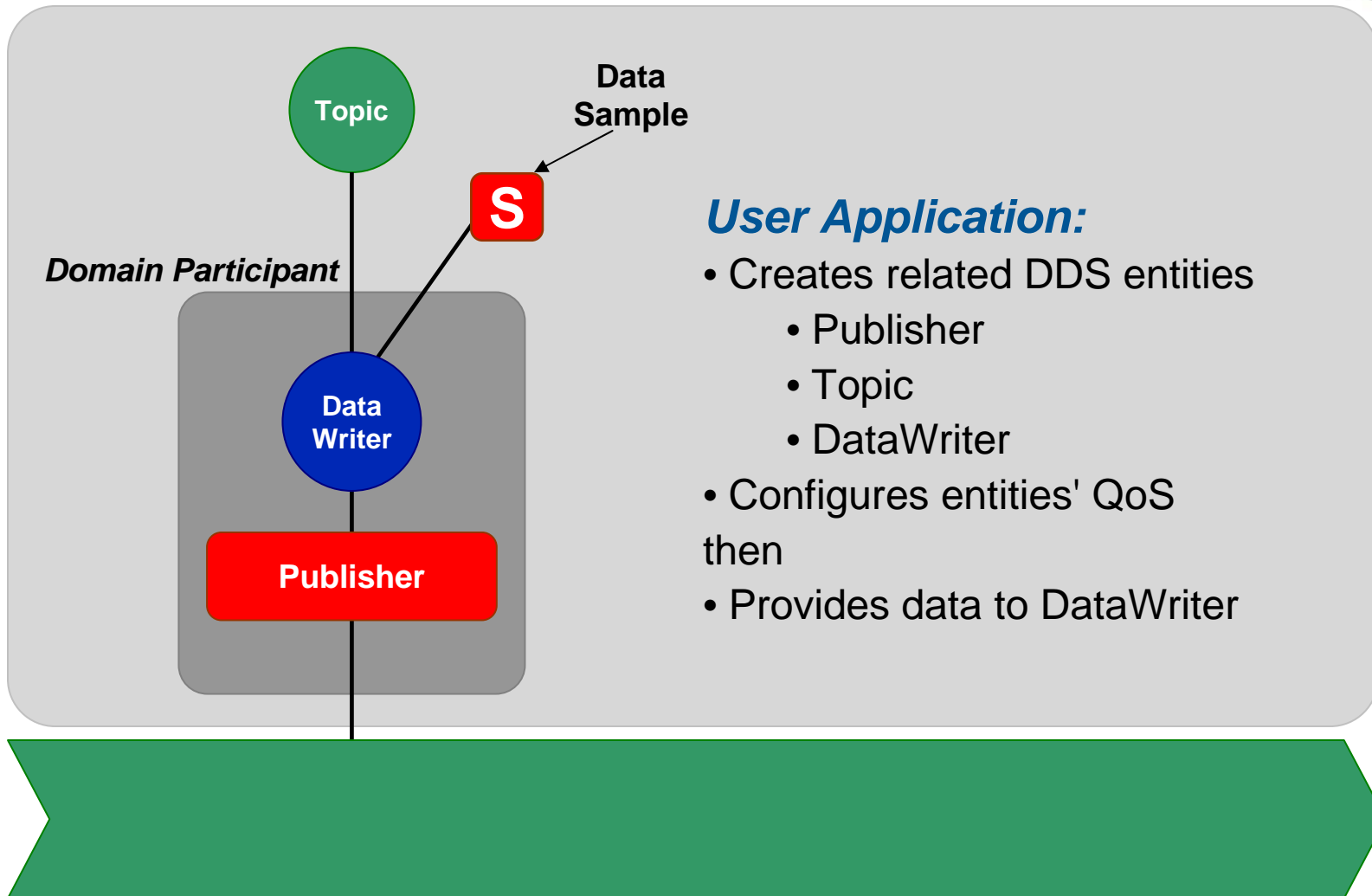
Domains and Participants



Domain Partitioning



DDS Publication



Example: Publication



```
// Entities creation
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;
// (Repeat each time data needs to be written)
twriter->write(&my_track);
```

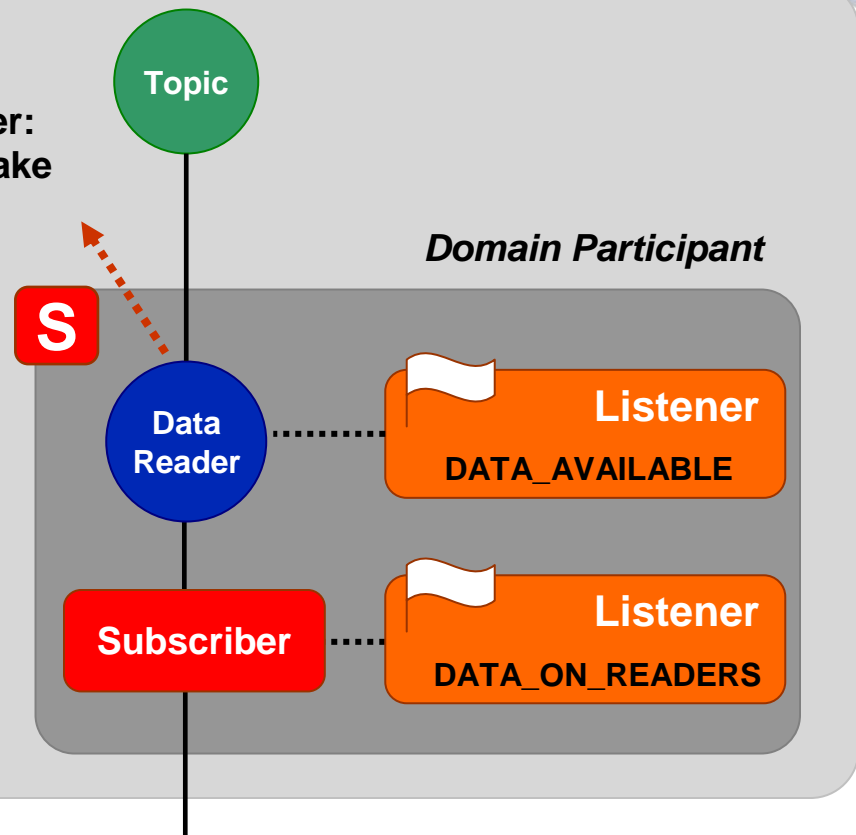
DDS Subscription with Listeners



User Application:

- Creates related DDS entities
 - Subscriber
 - Topic
 - DataReader
- Configures entities' QoS and attach listeners
- Receives Data from DataReader through attached listeners

Listener:
read, take

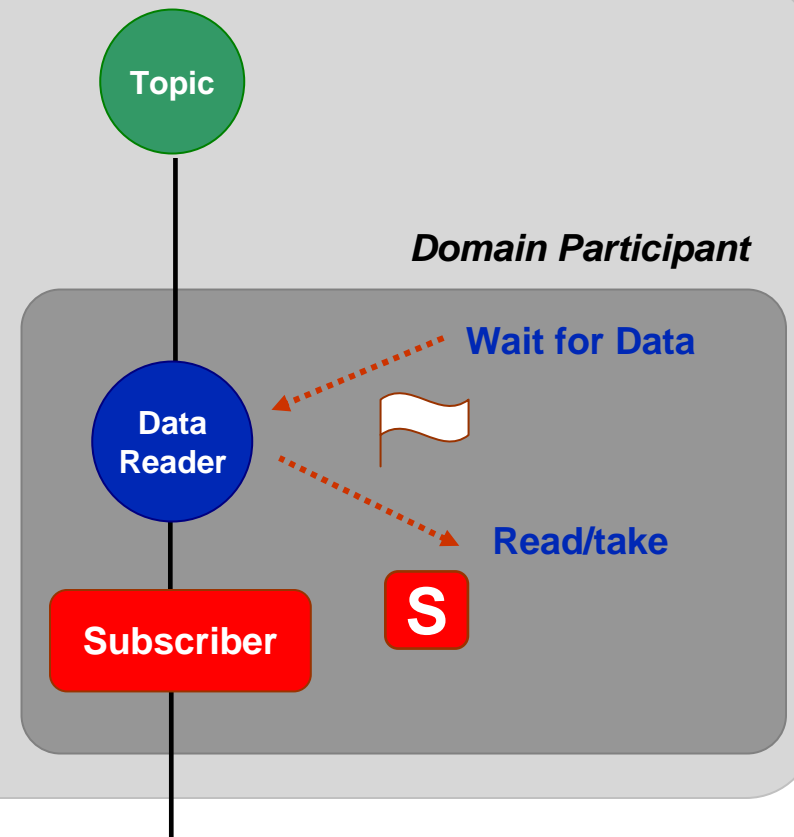


DDS Subscription with Wait-Set



User Application:

- Creates related DDS entities
 - Subscriber
 - Topic
 - DataReader
- Configures entities' QoS
- Creates a Condition and attaches it to a WaitSet
- Waits on the WaitSet until data arrive, then picks it on the DataReader



Example: Subscription



```
// Entities creation
```

```
Subscriber subscriber = 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 creation and attachment
```

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

```
// Listener code
```

```
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  
}
```

How to Get Data? (WaitSet-Based)



```
// Creation of condition and attachement
Condition foo_condition =
    treader->create_readcondition(...);
waitset->add_condition(foo_condition);

// Wait
ConditionSeq active_conditions;
waitset->wait(&active_conditions, timeout);
// active_conditions[0] == foo_condition
// => data is there, ready to be picked
FooSeq received_data;
SampleInfoSeq sample_info;

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

// Use received_data
```

Listeners, Conditions & WaitSets



Middleware must notify user application of relevant events

- Arrival of data
- But also:
 - QoS violations
 - Discovery of relevant entities
- These events may be detected asynchronously by the middleware
 - ... Same issue arises with POSIX signals

DDS allows the application to choose:

- Either to get notified asynchronously using a **Listener**
- Or to wait synchronously using a **WaitSet**

Both approaches are unified using STATUS changes

Status Changes



DDS defines

- A set of enumerated STATUS
- The statuses relevant to each kind of DDS Entity

DDS entities maintain a value for each STATUS

STATUS	Entity
INCONSISTENT_TOPIC	Topic
DATA_ON_READERS	Subscriber
LIVELINESS_CHANGED	DataReader
REQUESTED_DEADLINE_MISSED	DataReader
REQUESTED_INCOMPATIBLE_QOS	DataReader
DATA_AVAILABLE	DataReader
SAMPLE_LOST	DataReader
SUBSCRIPTION_MATCH	DataReader
LIVELINESS_LOST	DataWriter
OFFERED_INCOMPATIBLE_QOS	DataWriter
PUBLICATION_MATCH	DataWriter

```
struct LivelinessChangedStatus
{
    long active_count;
    long inactive_count;
    long active_count_change;
    long inactive_count_change;
}
```

Listeners, Conditions and Statuses



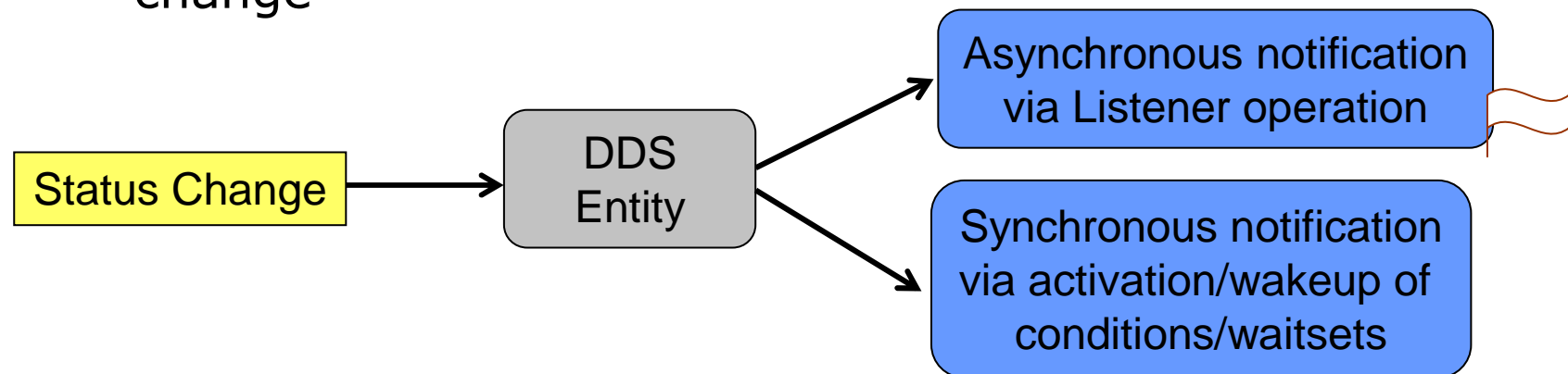
- A DDS Entity is associated with:
 - ▣ A listener of the proper kind (if attached)
 - ▣ A StatusCondition (if activated)
- The Listener for an Entity has a separate operation for each of the relevant statuses

STATUS	Entity	Listener operation	
INCONSISTENT_TOPIC	Topic	on_inconsistent_topic	
DATA_ON_READERS	Subscriber	on_data_on_readers	
LIVELINESS_CHANGED	DataReader	on_liveliness_changed	
REQUESTED_DEADLINE_MISSED	DataReader	on_requested_deadline_missed	
REQUESTED_INCOMPATIBLE_QOS	DataReader	on_requested_incompatible_qos	
DATA_AVAILABLE	DataReader	on_data_available	
SAMPLE_LOST	DataReader	on_sample_lost	
SUBSCRIPTION_MATCH	DataReader	on_subscription_match	
LIVELINESS_LOST	DataWriter	on_liveliness_lost	
OFFERED_INCOMPATIBLE_QOS	DataWriter	on_offered_incompatible_qos	
PUBLICATION_MATCH	DataWriter	on_publication_match	

Listeners & Condition duality



- A StatusCondition can be selectively activated to respond to any subset of the statuses
- An application can wait changes in sets of StatusConditions using a WaitSet
- Each time the value of a STATUS changes DDS
 - ▢ Calls the corresponding Listener operation
 - ▢ Wakes up any threads waiting on a related status change



Outline



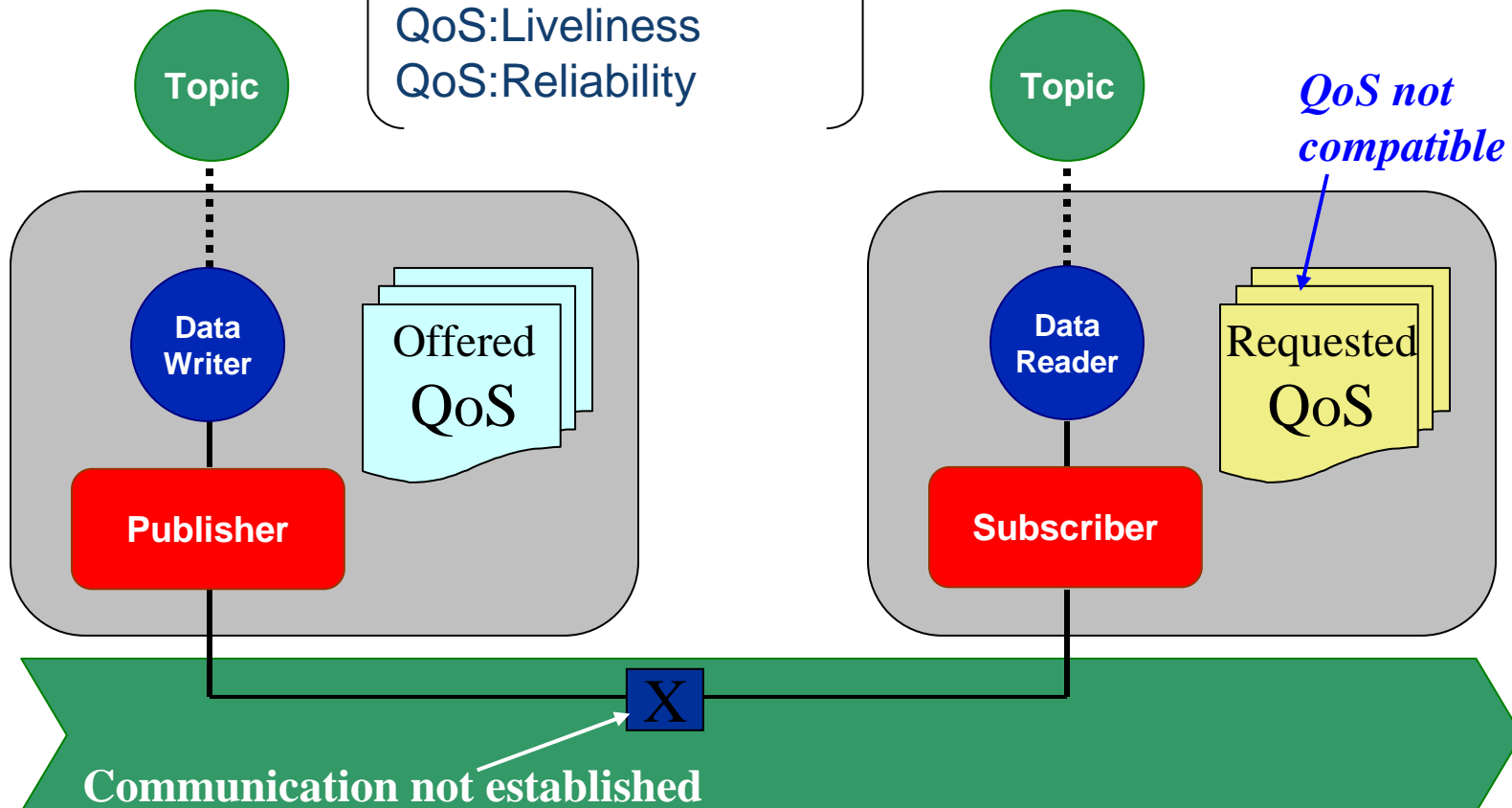
- Background
 - ▣ Middleware information models
 - ▣ Publish / Subscribe
 - ▣ Topic-based Publish / Subscribe
- Focus on Topics
 - ▣ Topic definition, keys
 - ▣ ContentFilteredTopic, MultiTopic
- Publication & Subscription
 - ▣ Related DDS Entities
 - ▣ DDS Publication
 - ▣ DDS Subscription
 - ▣ Dual mechanism to access incoming information
 - Listeners
 - WaitSets and Conditions
- Quality of Service

QoS Contract "Request / Offered"



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

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

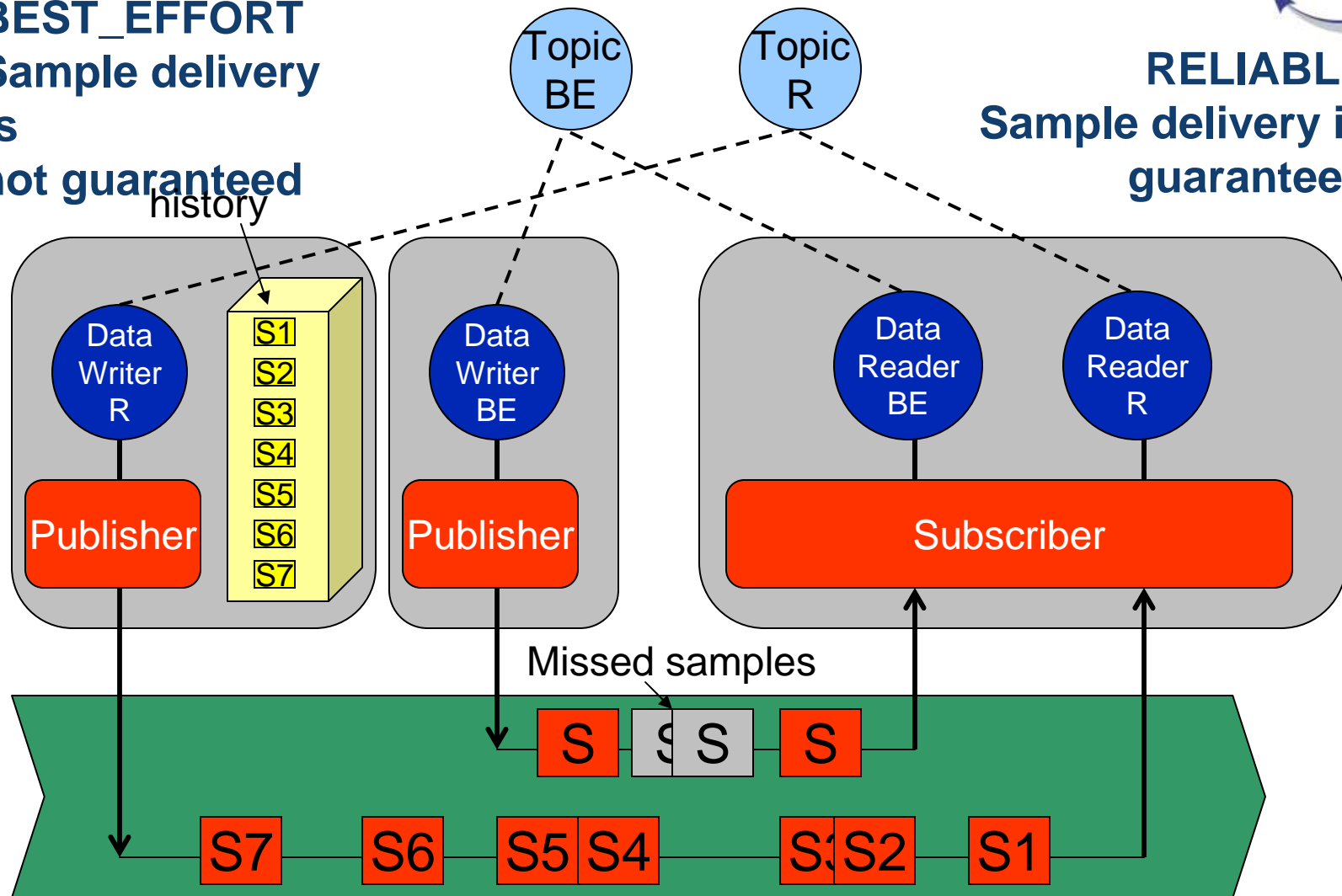


QoS: RELIABILITY



BEST_EFFORT
Sample delivery
is
not guaranteed

RELIABLE
Sample delivery is
guaranteed



QoS: HISTORY – Last x or All

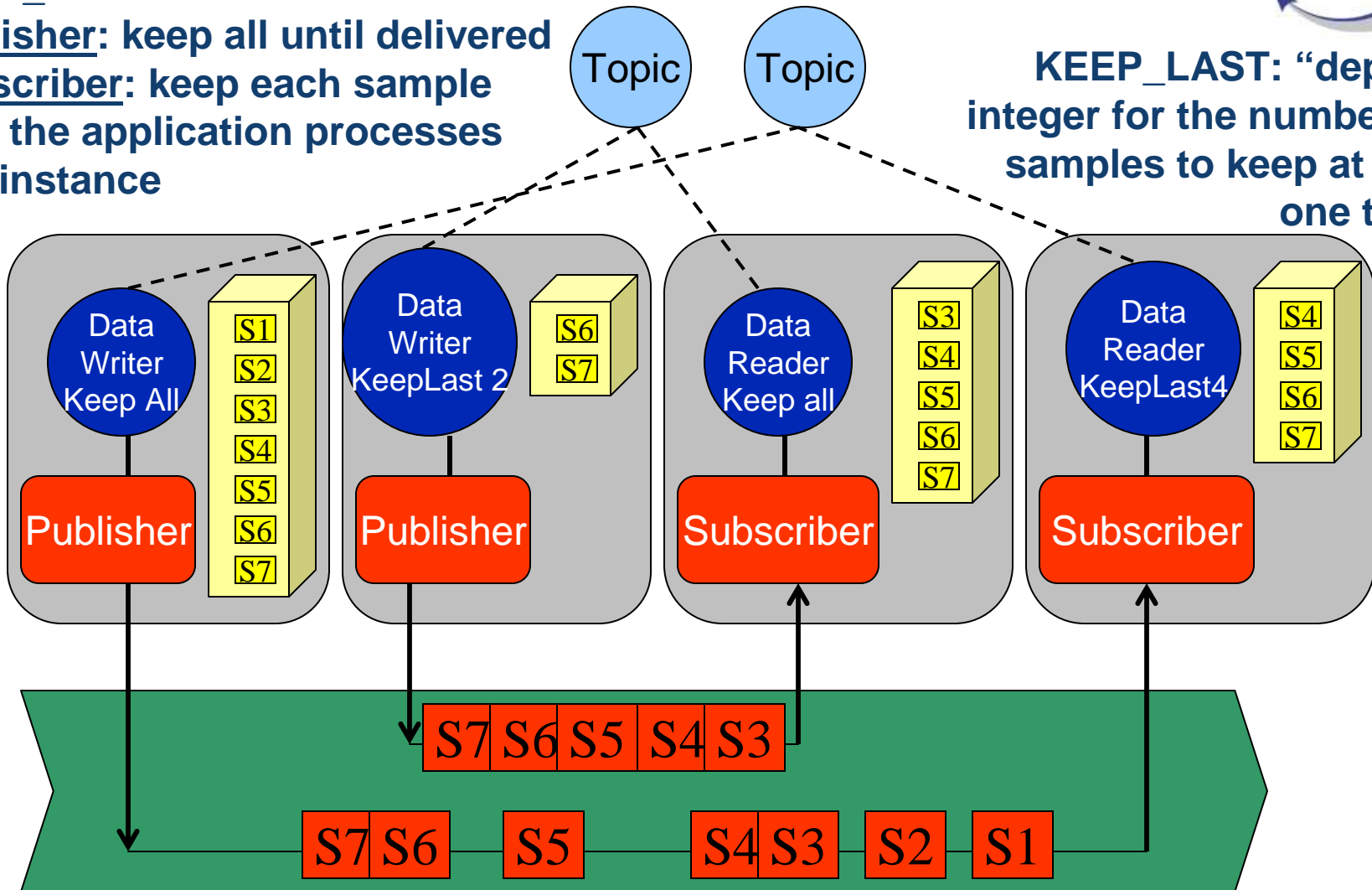


KEEP_ALL:

Publisher: keep all until delivered

Subscriber: keep each sample until the application processes that instance

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



State Propagation

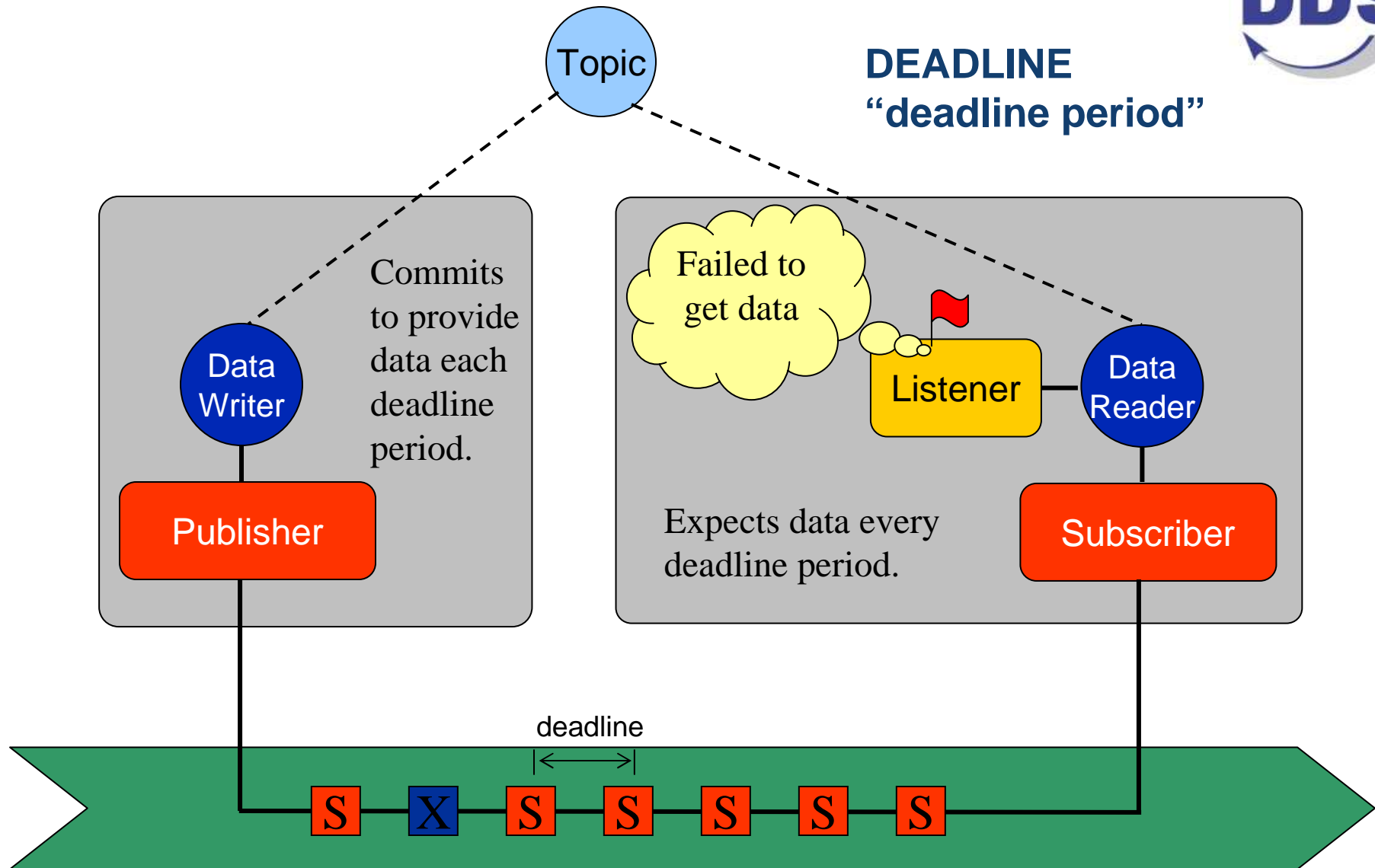


- System state
 - Information needed to describe future behavior of the system
 - System evolution defined by state and future inputs.
 - Minimalist representation of past inputs to the system
- State variables
 - Set of data-objects whose value codifies the state of the system
- Relationship with DDS
 - 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**

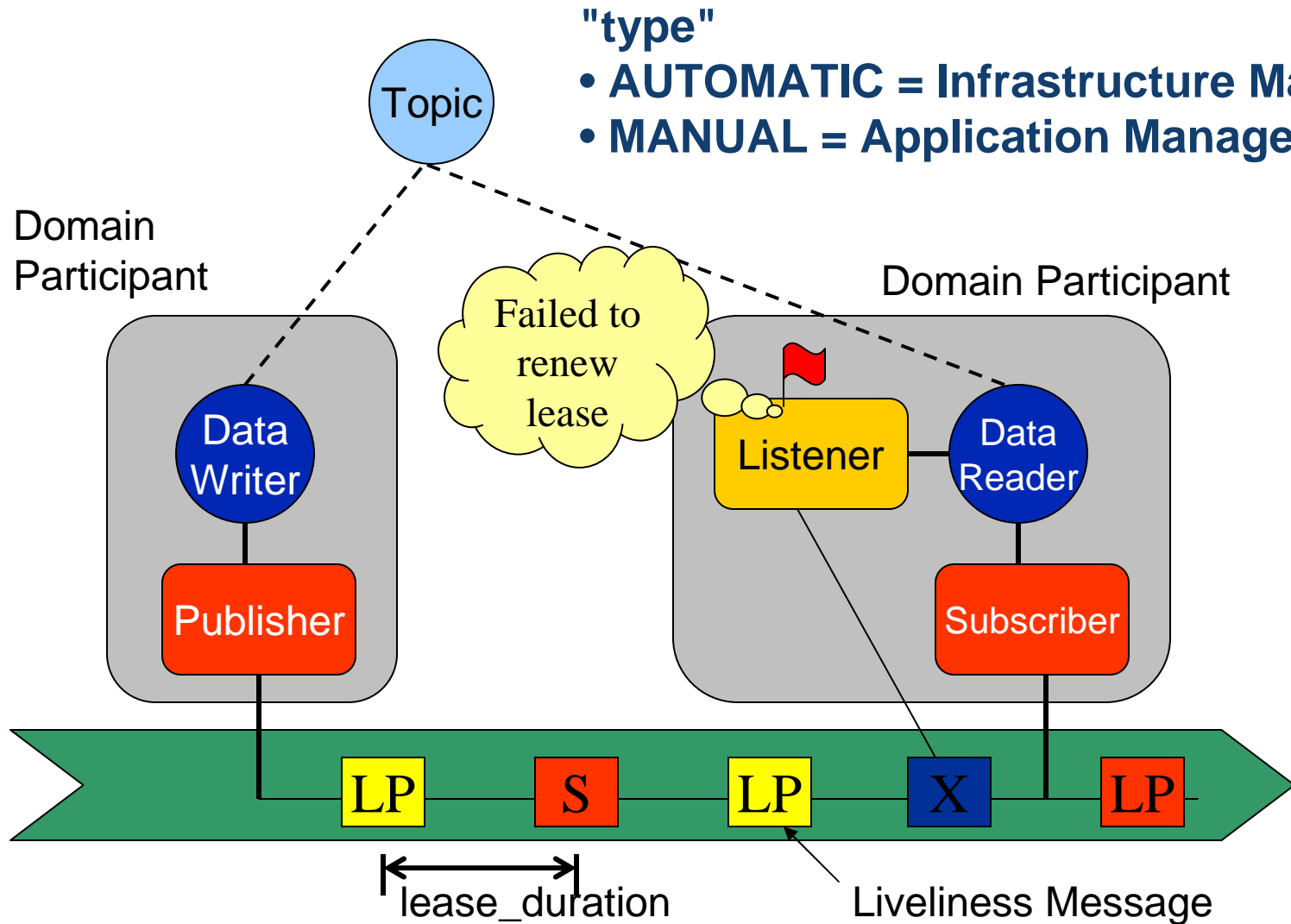
Present in many RT applications

Key ingredient for fault-tolerance

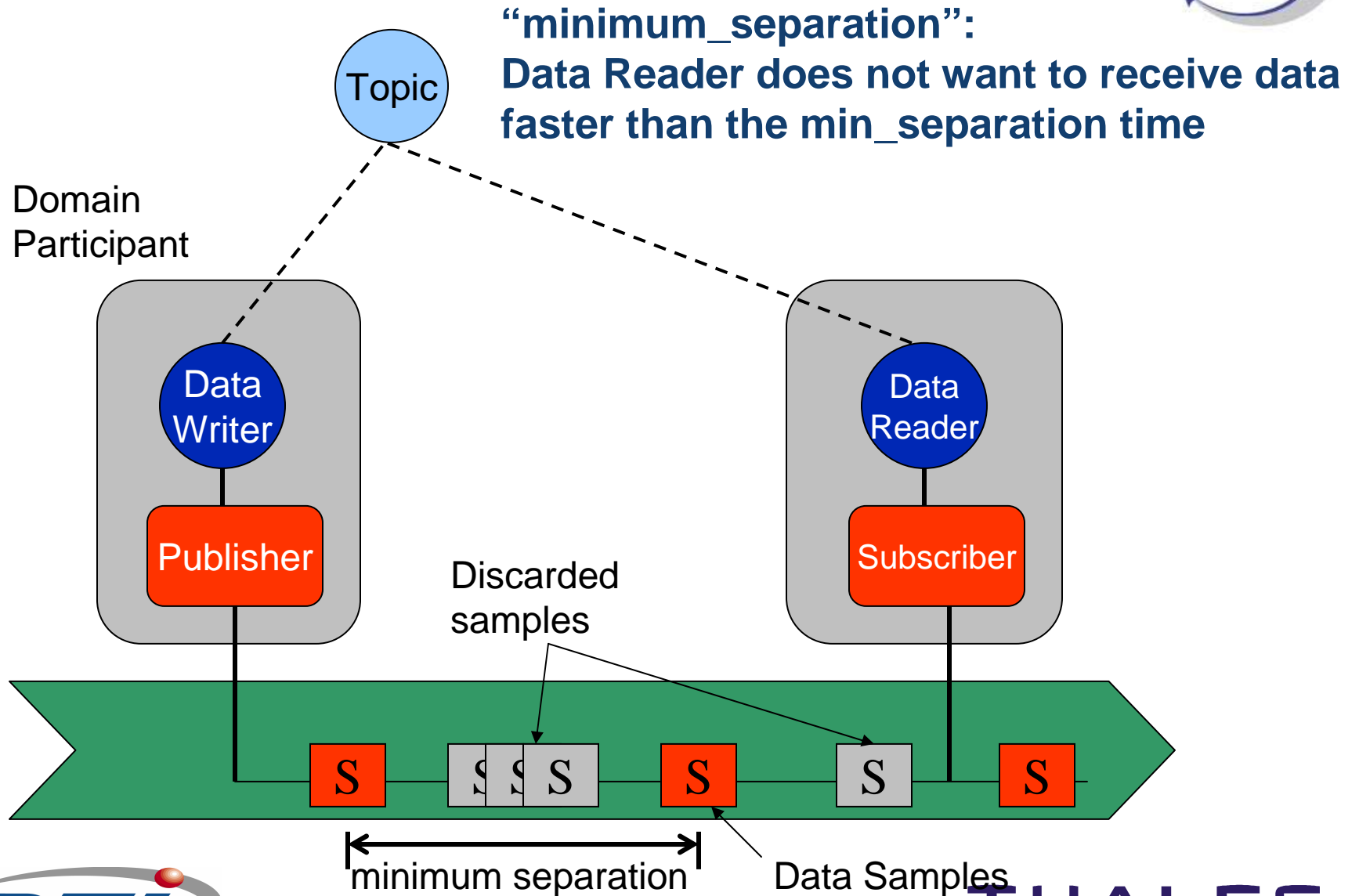
QoS: DEADLINE



QoS: LIVELINESS – Type, Duration



QoS: TIME_BASED_FILTER

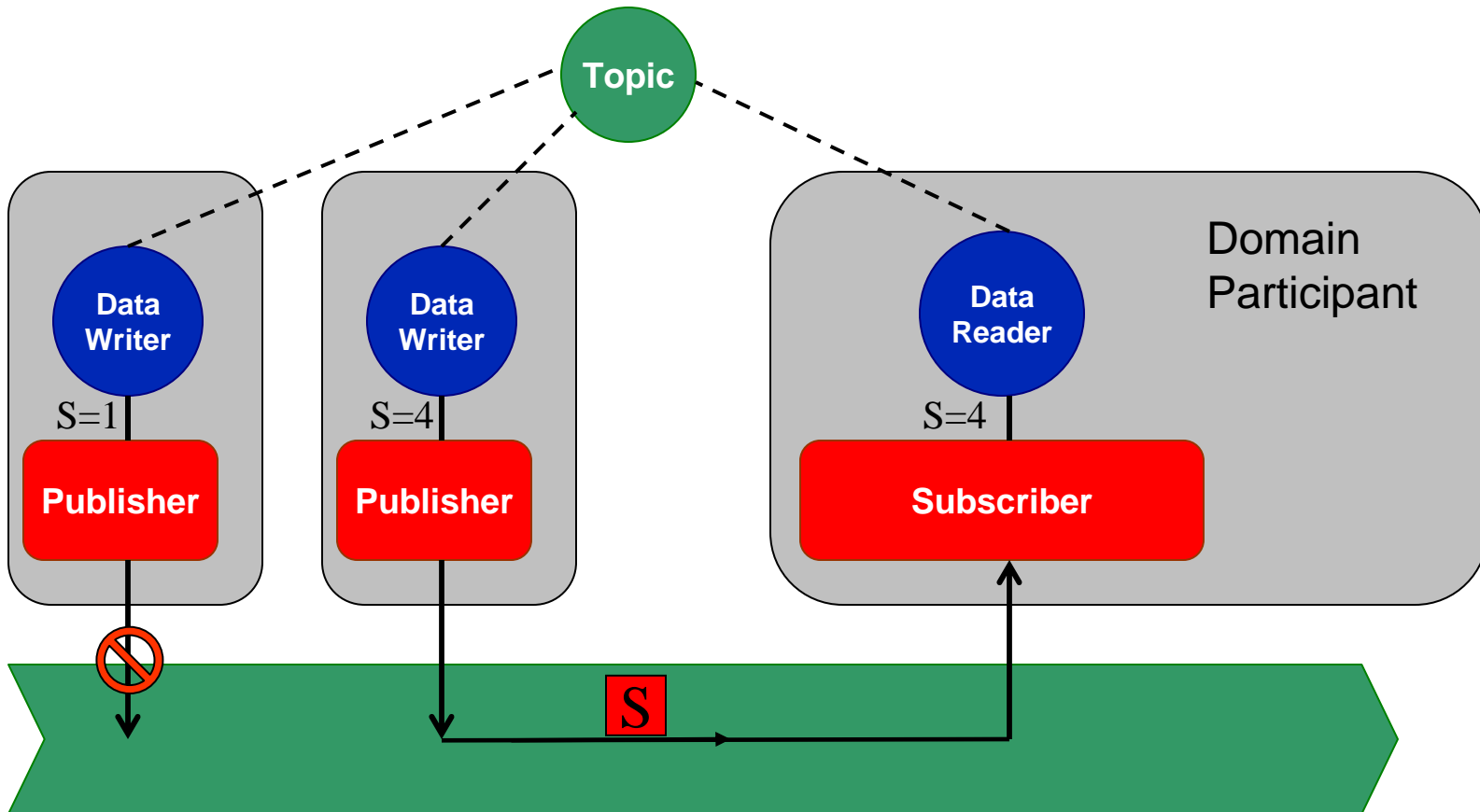


QoS: OWNERSHIP_STRENGTH



OWNERSHIP_STRENGTH:

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

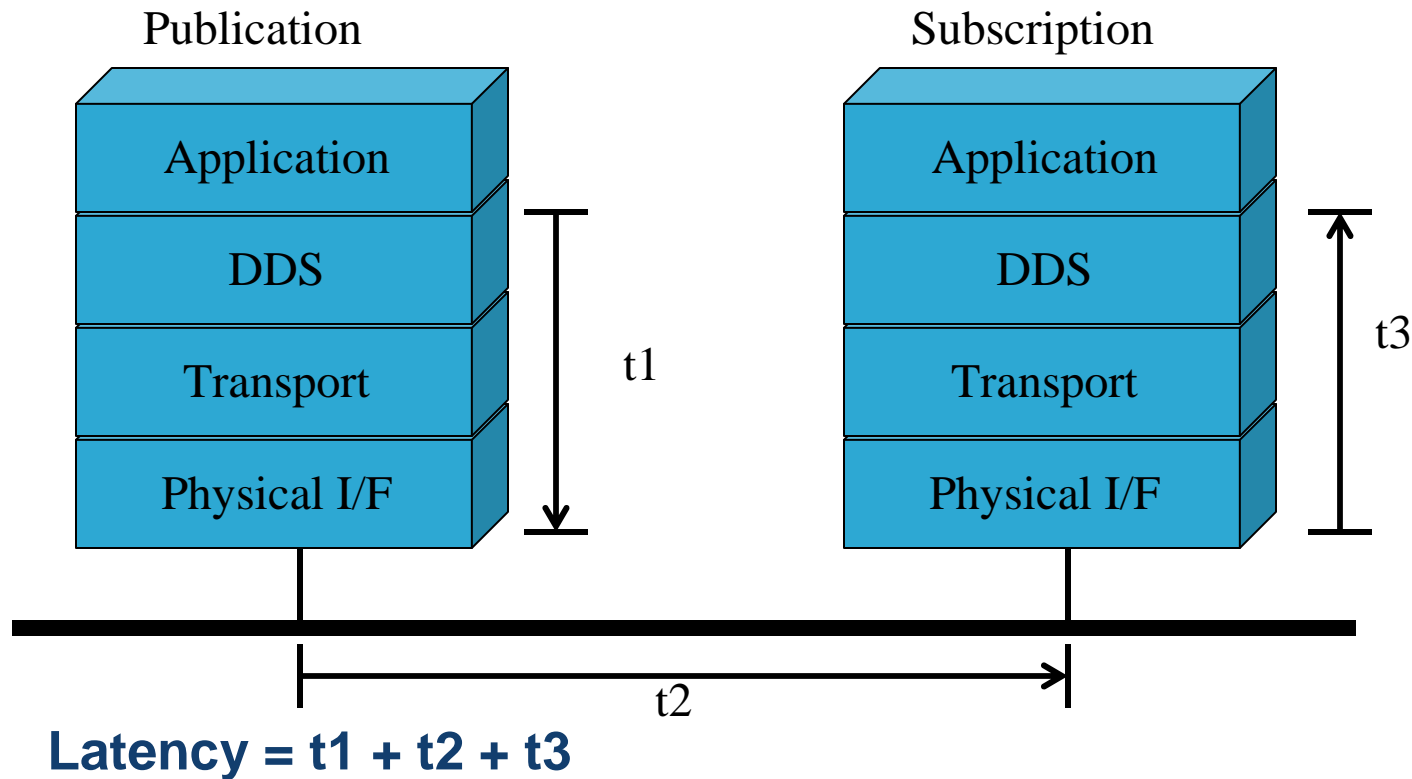


Note: Only applies to Topics with OWNERSHIP=Exclusive

QoS: LATENCY_BUDGET



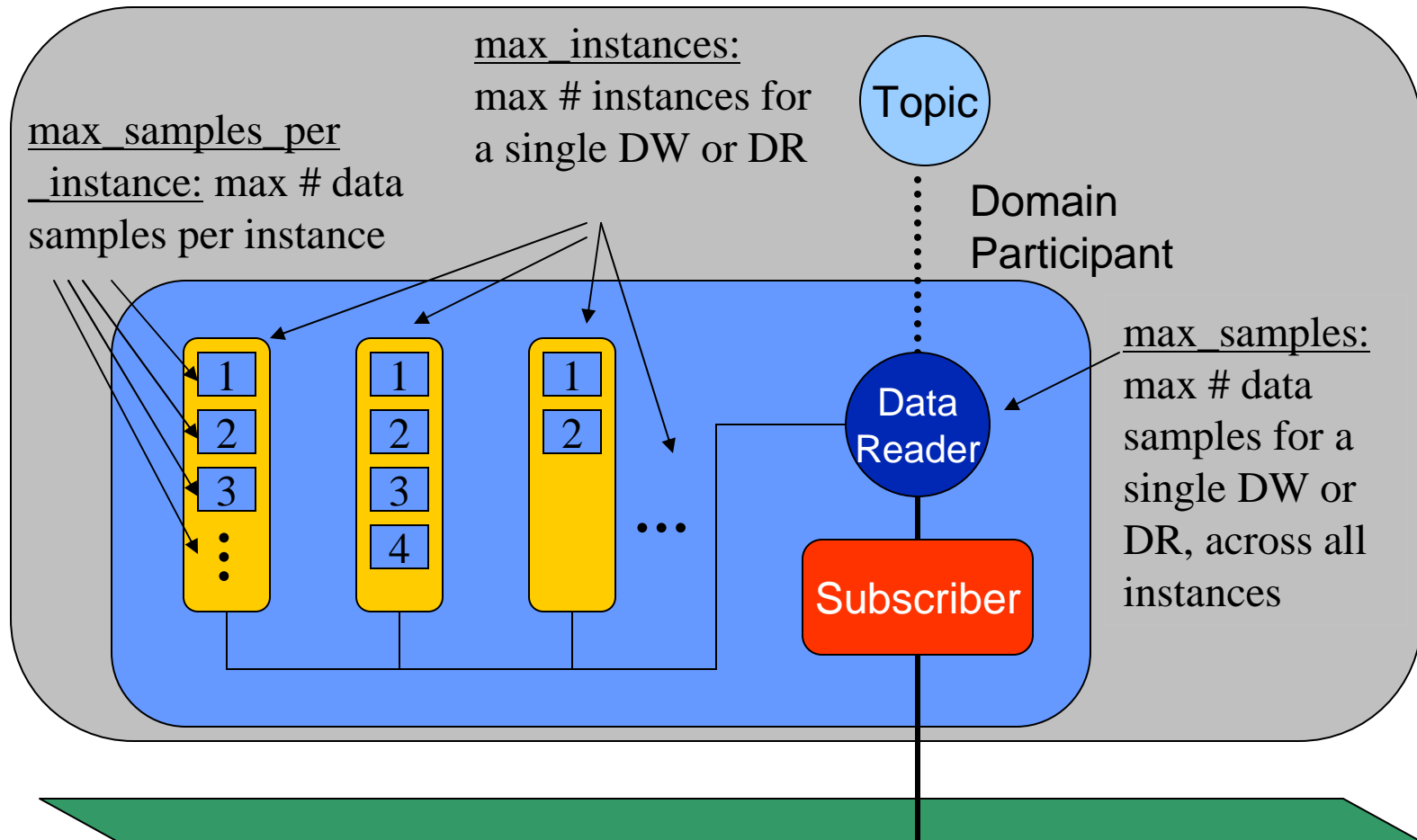
- Intended to provide time-critical information to the publisher for framework tuning where possible.
- Will not prevent data transmission and/or receipt.



QoS: RESOURCE_LIMITS



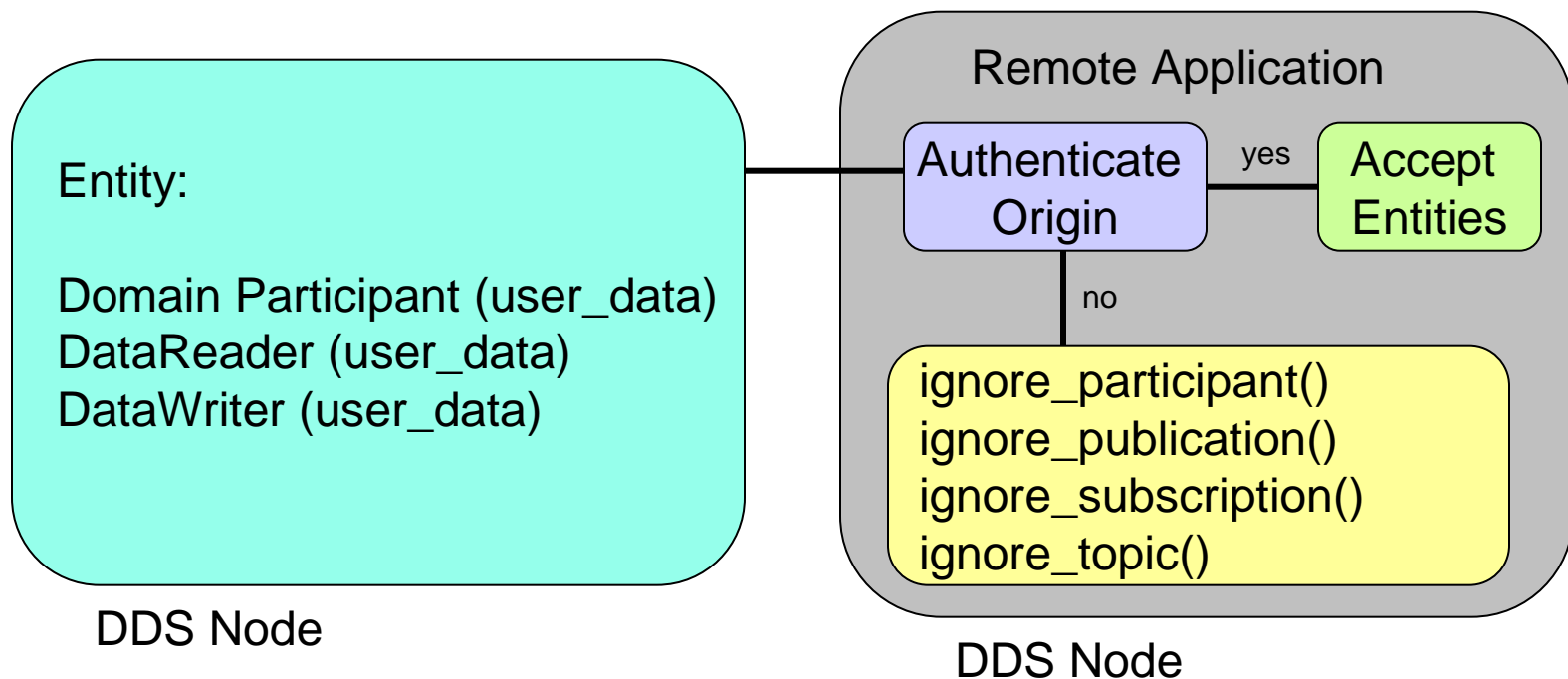
- Specifies the resources that the Service can consume to meet requested QoS



QoS: USER_DATA



- User-defined portion of Topic metadata
- Example of use: Security Authentication



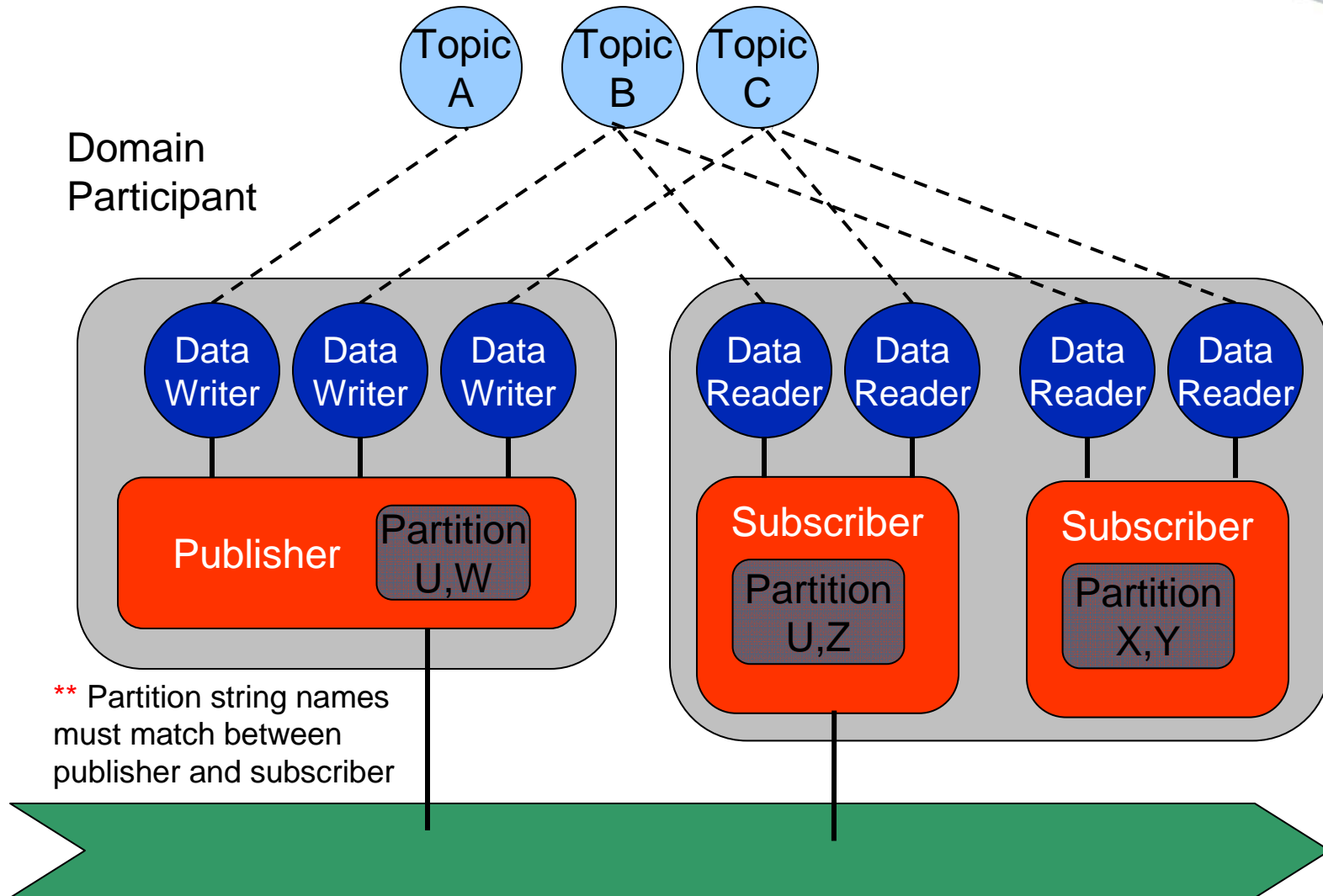
USER_DATA can be used to authenticate an origination entity.

Note: USER_DATA is contained within the DDS metadata.

QoS: PARTITION



Logical “namespace” for topics



QoS: DURABILITY



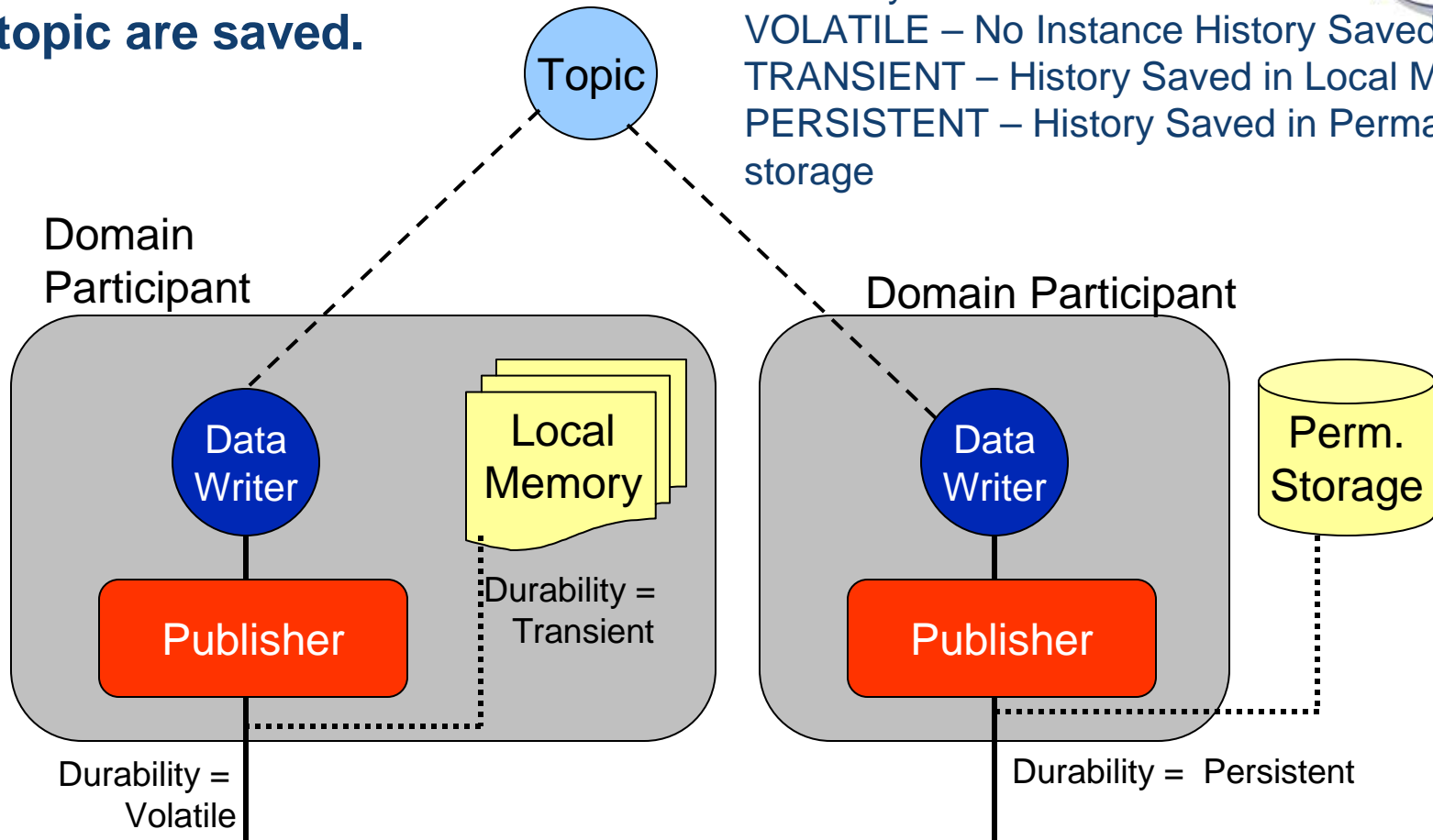
Determines if/how instances of a topic are saved.

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: 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.

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

DDS-DCPS 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

