Your systems. Working as one.

DDS Advanced Tutorial
The Evolution of the DataBus

Gerardo Pardo-Castellote, Ph.D.

CTO, Real-Time Innovations

Co-Chair OMG DDS SIG

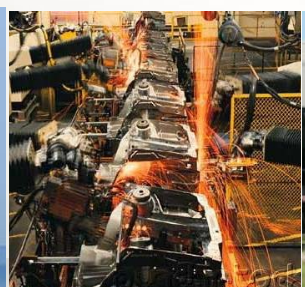
Agenda

- Introduction and Background
- Data-Centricity 101
- Software Installation
  - Exercise: Shapes
- Developer and Run-time Tools
  - Exercise: Ping, Spy, Analyzer, Recorder, Persistence
- Your first Application
- Getting started with DDS and XML
- Exercise: Adjusting QoS
- Defining your own data-types
- Building a chat application

Systems that interact with the Real World

- Must adapt to changing environment
- Cannot stop processing the information
- Live within world-imposed timing

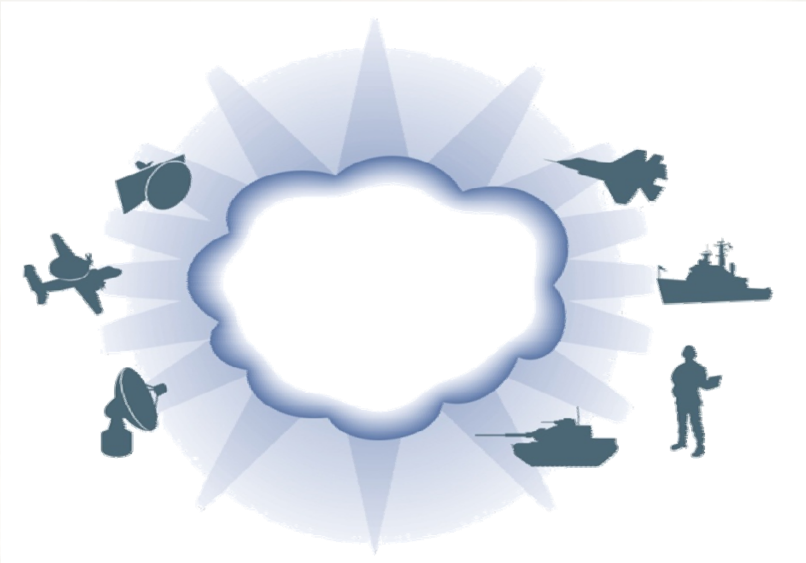
Beyond traditional interpretation of real-time

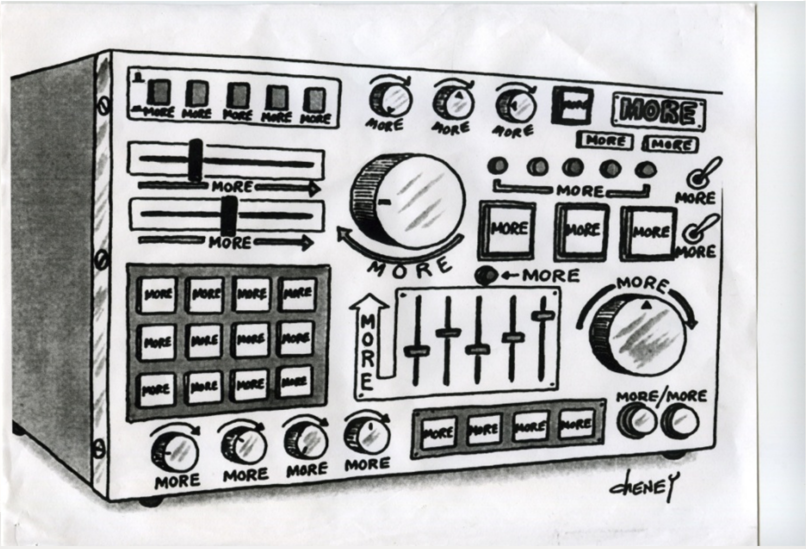




Challenge:
More Data, More Speed, More Sources

TRENDS:
- Growing Information Volume
- Lowering Decision Latency
- Increasing System Availability
- Accelerating technology insertion and deployment

Next-generation systems needs:
- Scalability
- Integration & Evolution
- Robustness & Availability
- Performance
- Security





"Real World" Systems are integrated using a Data Model

- Grounded on the "physics" of the problem domain
  - Tied to the nature of the sensors and real objects in the system (vehicles, device types, …)
- Provides governance across disparate teams & organizations
  - The "N^2" integration problem is reduced to a "N" problem
- Increased decoupling from use-cases and components
  - Avoids over constraining applications
- Open, Evolvable, Platform-Independent
  - The use-cases, algorithms might change between missions or versions of the system

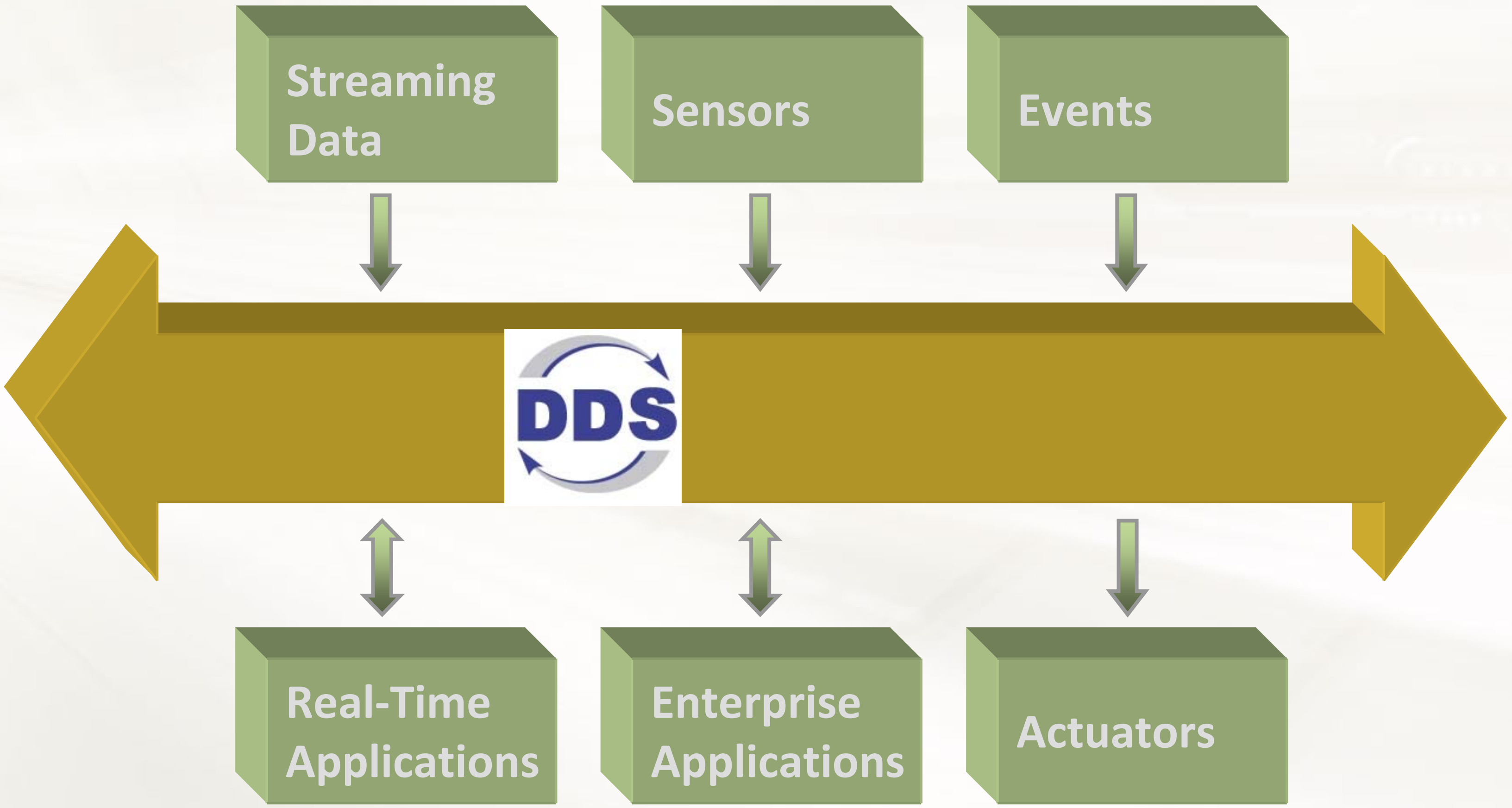
App

App

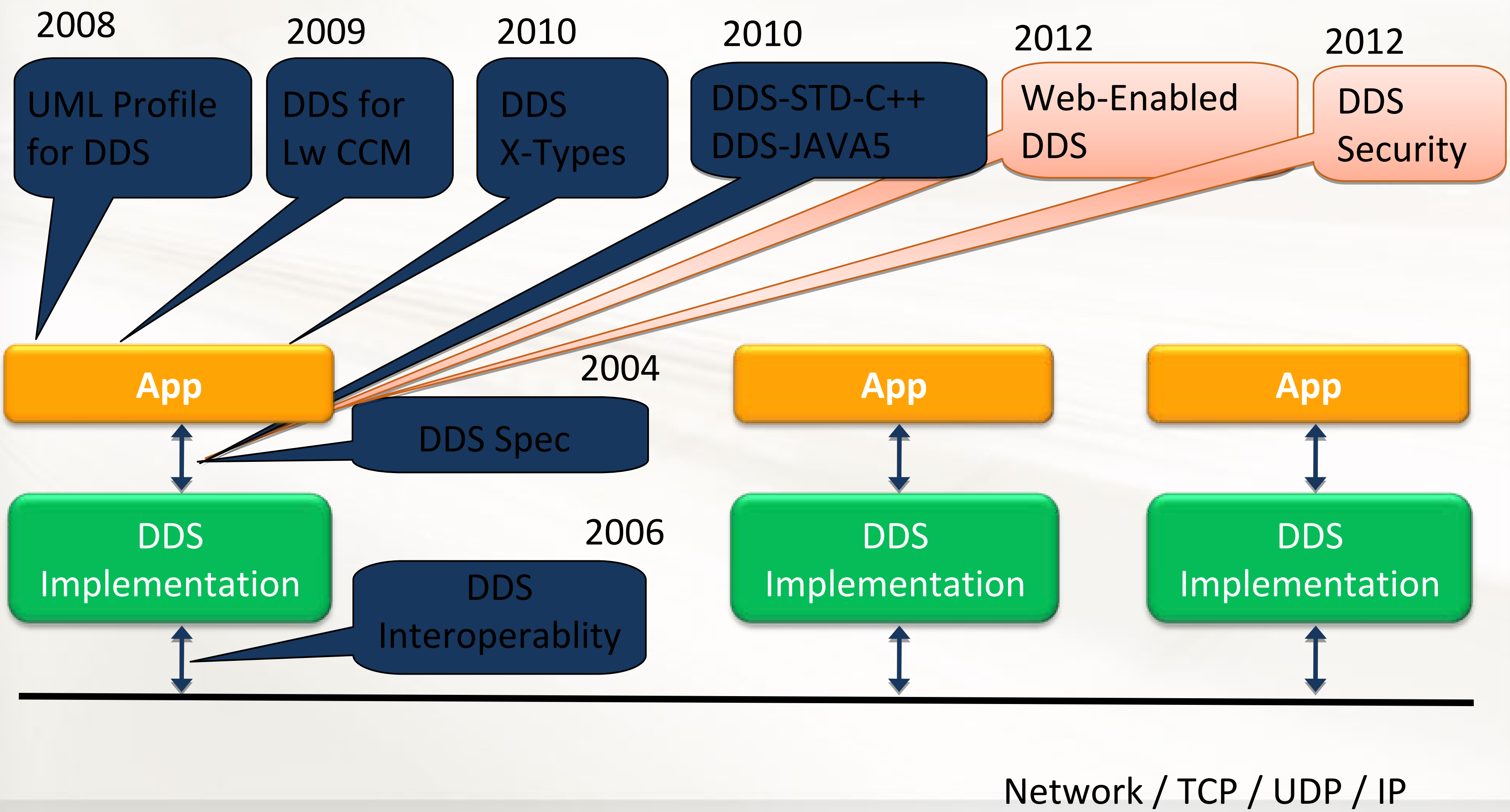
App

Realizing this data-model requires a middleware infrastructure

DDS: Standards-based Integration Infrastructure for Critical Applications

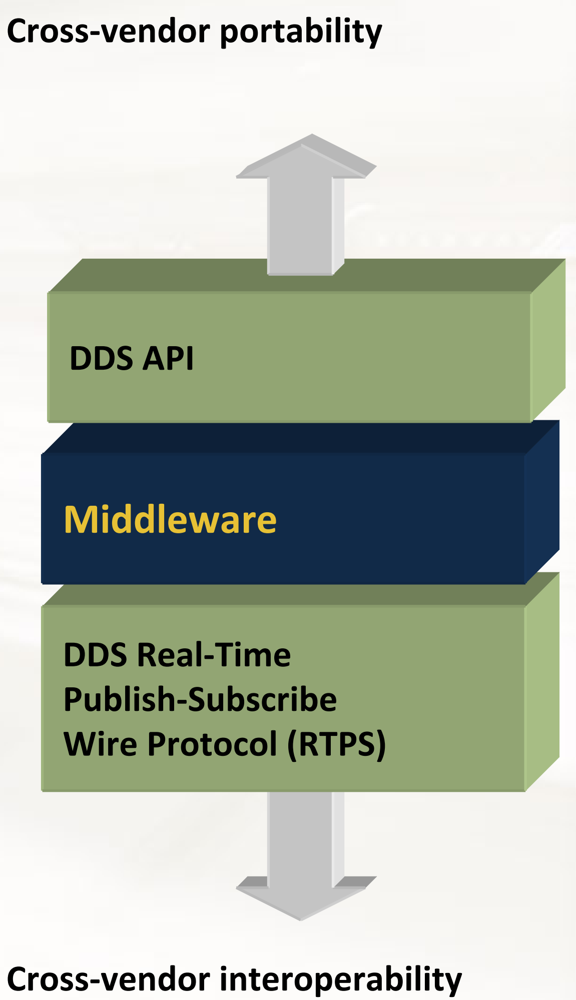


Family of Specifications



Broad Adoption

- Vendor independent
  - API for portability
  - Wire protocol for interoperability
- Multiple implementations
  - 10 of API
  - 8 support RTPS
- Heterogeneous
  - C, C++, Java, .NET (C#, C++/CLI)
  - Linux, Windows, VxWorks, other embedded & real-time
- Loosely coupled



US-DoD mandates DDS for data-distribution

- DISR (formerly JTA)
  - DoD Information Technology Standards Registry
- US Navy Open Architecture
- Army, OSD
  - UCS, Unmanned Vehicle Control
- *SPAWAR NESI*
  - *Net-centric Enterprise Solutions for Interoperability*
  - *Mandates DDS for Pub-Sub SOA*

DDS adopted by key programs

- *European Air Traffic Control*
  - *DDS used to interoperate ATC centers*
- *UK Generic Vehicle Architecture*
  - *Mandates DDS for vehicle comm.*
  - *Mandates DDS-RTPS for interop.*
- DISR
  - Mandates DDS for Pub-Sub API
  - Mandates DDS-RTPS for Pub-Sub Interop
- US Navy Open Architecture
  - Mandates DDS for Pub-Sub
- *SPAWAR NESI*
  - *Mandates DDS for Pub-Sub SOA*

Evolution of DataBus

Data-centricity basics

Everyday Example: Calendaring

Alternative Process #1 (message-centric):

1. Email: “Meeting Monday at 10:00.”
2. Email: “Here’s dial-in info for meeting…”
3. Email: “Meeting moved to Tuesday”

4. You: “Where do I have to be? When?”
5. You: (*sifting through email messages…*)

Example: Calendaring

Alternative Process #2:

1. Calendar: (*add meeting Monday at 10:00*)
2. Calendar: (*add dial-in info*)
3. Calendar: (*move meeting to Tuesday*)

4. You: “Where do I have to be? When?”
5. You: (*check calendar. Contains consolidated-state*)

The difference is state!
The infrastructure consolidates changes and maintains it

What’s the Difference? ***State***.

- **Objects have identity and attributes**
  - The meeting will run **1:00–2:00** in the **conference room**.
  - My friend’s phone number is **555-1234 his email is…**
  - The car is **blue** and is **traveling north** from **Sunnyvale** at **65 mph**.
- **…whether they exist in the real world, in the computer, or both**
- **…whether or not we observe or acknowledge them**

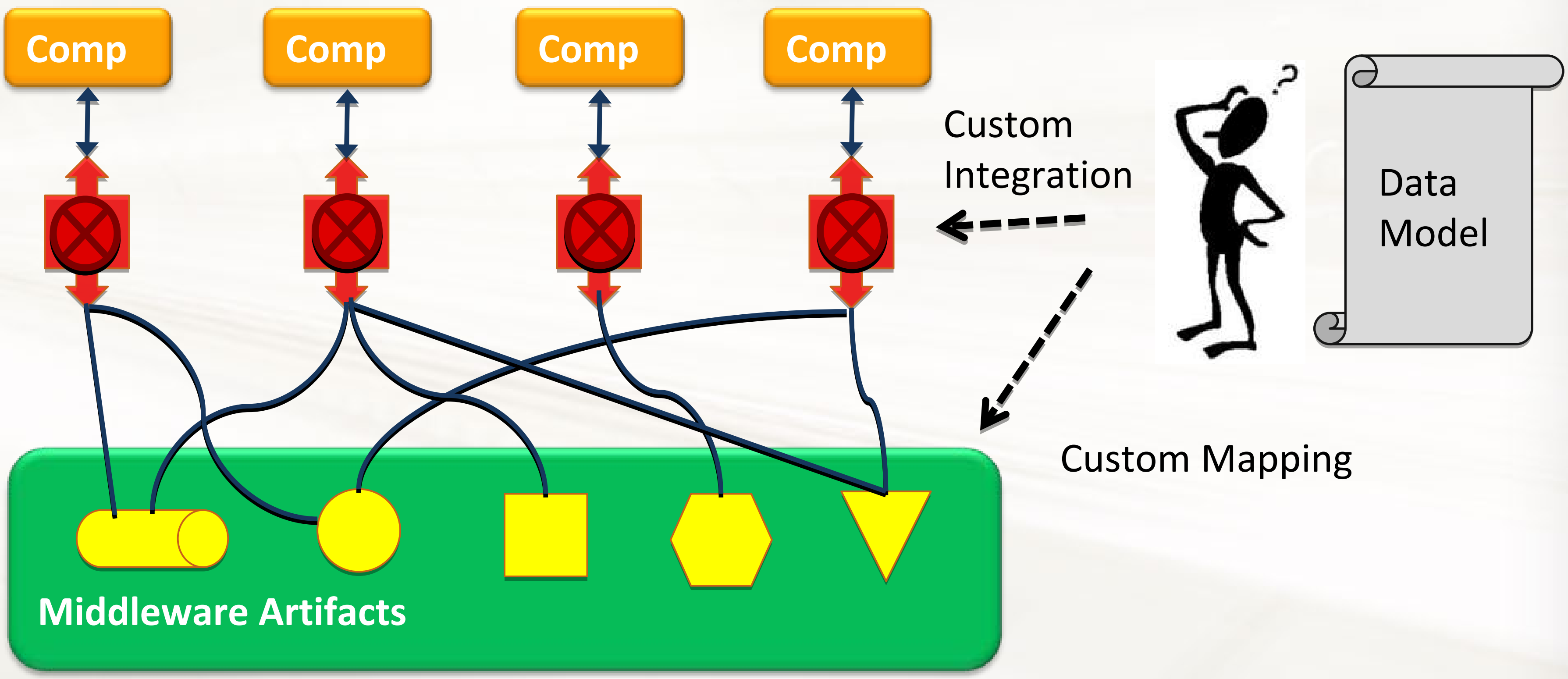
“State” (“data”) is a snapshot of those attributes and characteristics.

If the infrastructure maintains the state, the application does not need to re-construct it…

Why is it better to have the (data-centric) middleware manage the state?

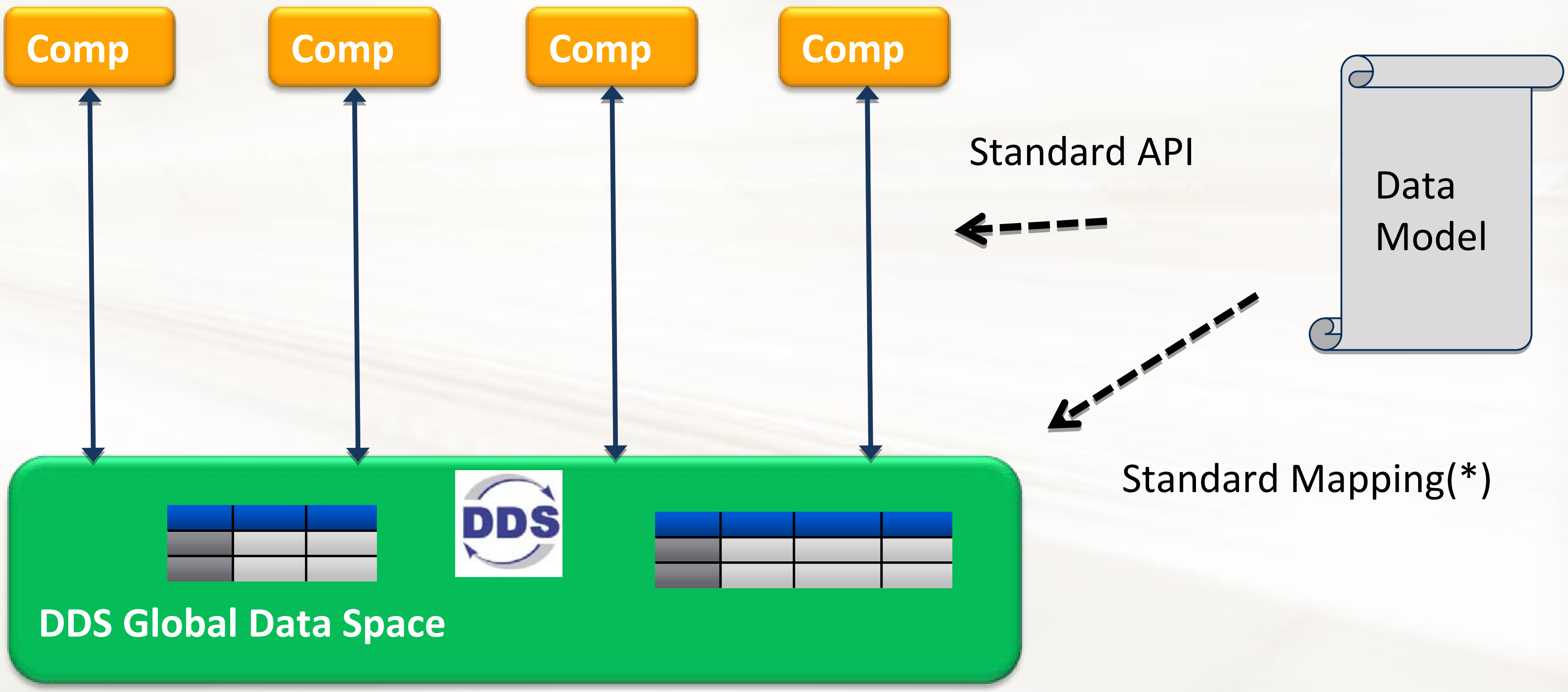
- **Reconstructing the state of an object is hard**
  - Must infer based on all previous messages
  - Maintaining all these messages is expensive
  - Each app makes these inferences
    => duplicate effort
- **Reconstructing state is not robust**
  - Many copies of state => may be different => bugs
    vs. Uniform operations on state => fewer bugs
  - Any missing change compromises integrity
- **State awareness results in better performance**
  - Middleware can be smart about what to send and when
- **Data-type awareness simplifies programming**
  - Middleware supports direct definition and instantiation of the data-model

Integrating components to generic middleware technology



Akin to implementing an OO design on a Procedural Language:
Requires mapping inheritance, encapsulation, exceptions, …

Integrating components to data-centric middleware technology



No custom mappings / code necessary

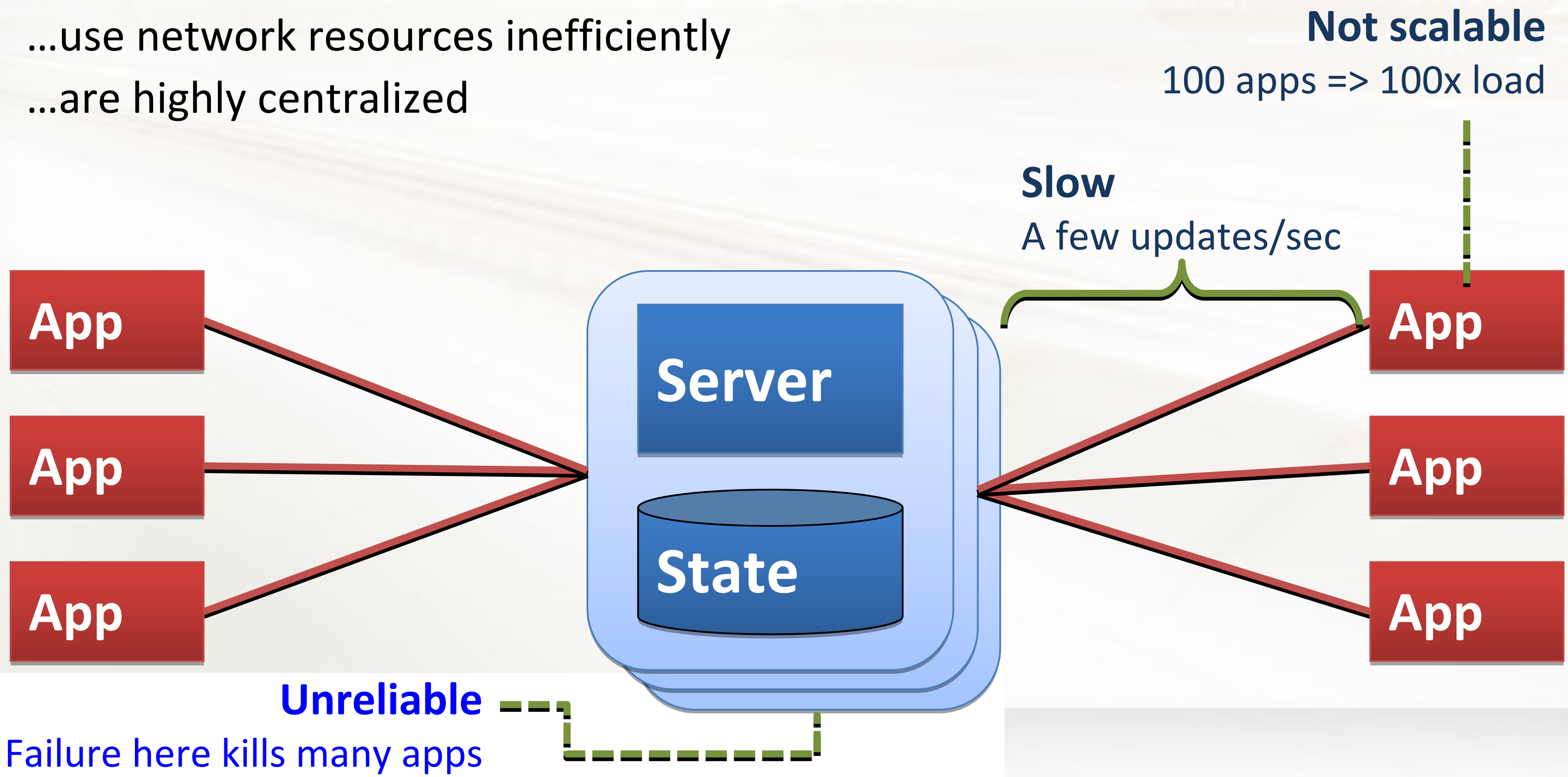
Direct support for data-centric actions: create, dispose, read/take

Traditional data-centric technologies not suited to scalable near real-time systems

Other data-centric technologies:

- Databases: SQL
- Web: HTTP (mostly)

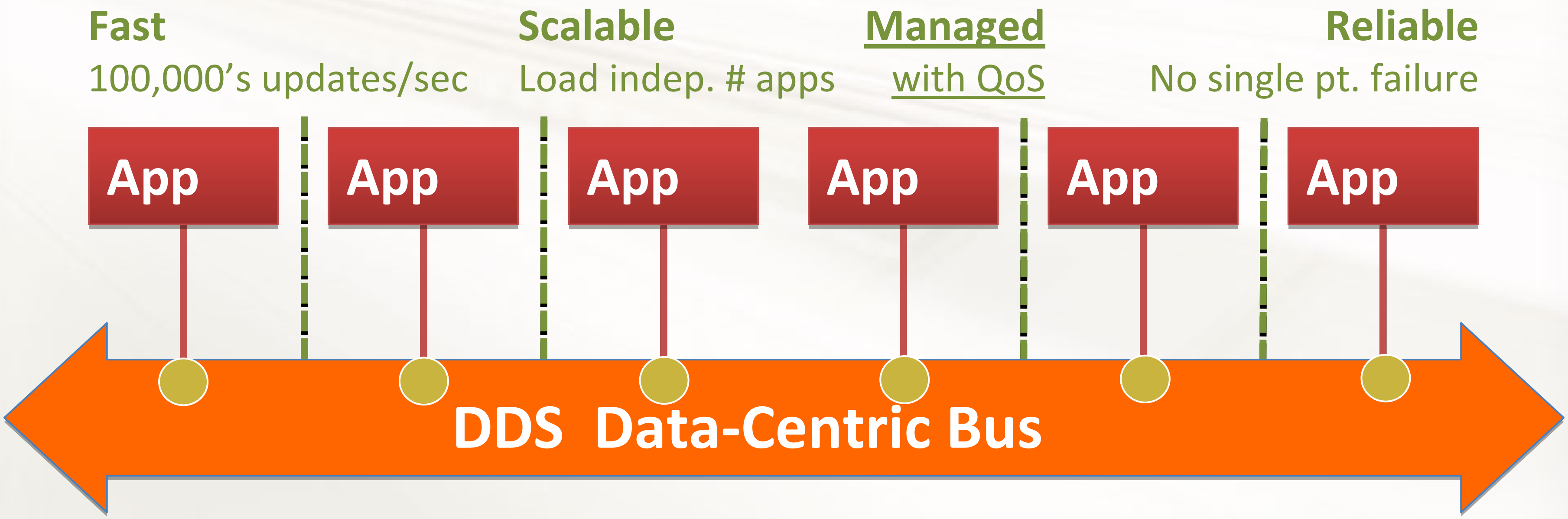
- …assume the world changes slowly
- …use network resources inefficiently
- …are highly centralized



DDS is decentralized. Can be deployed without servers/brokers

DDS:

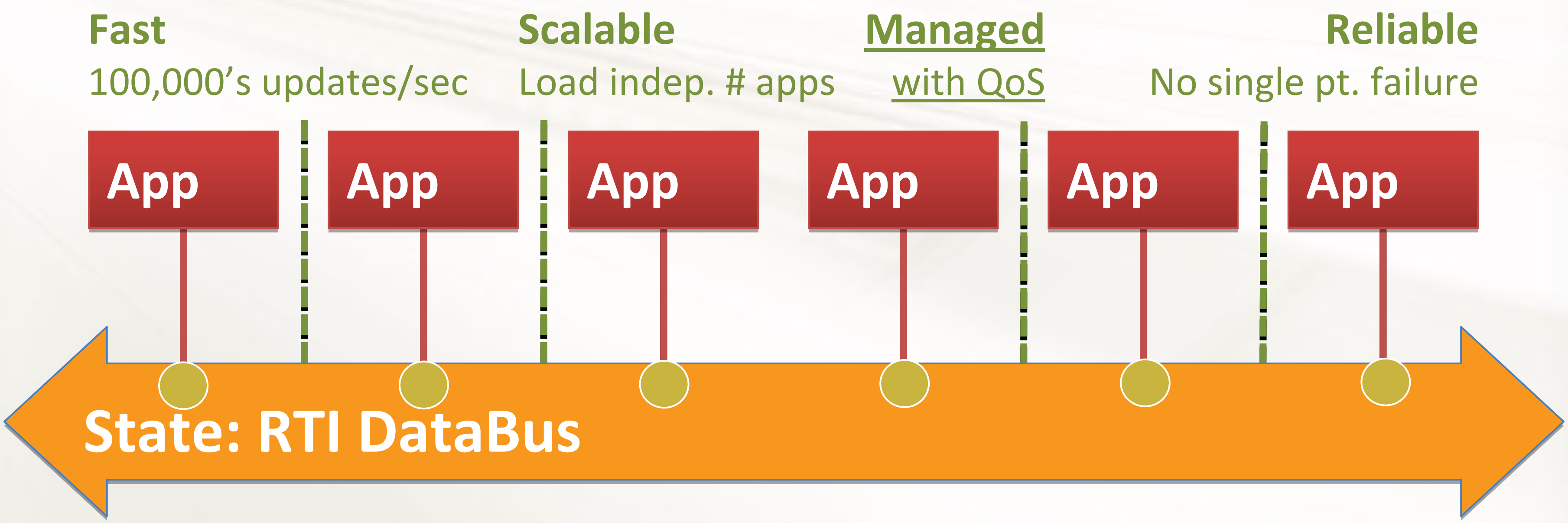
- …allows you to observe frequent changes
- …uses network resources efficiently
- …is **peer-to-peer and decentralized**



RTI Data-Centric DataBus
Meets the needs of Operational Systems

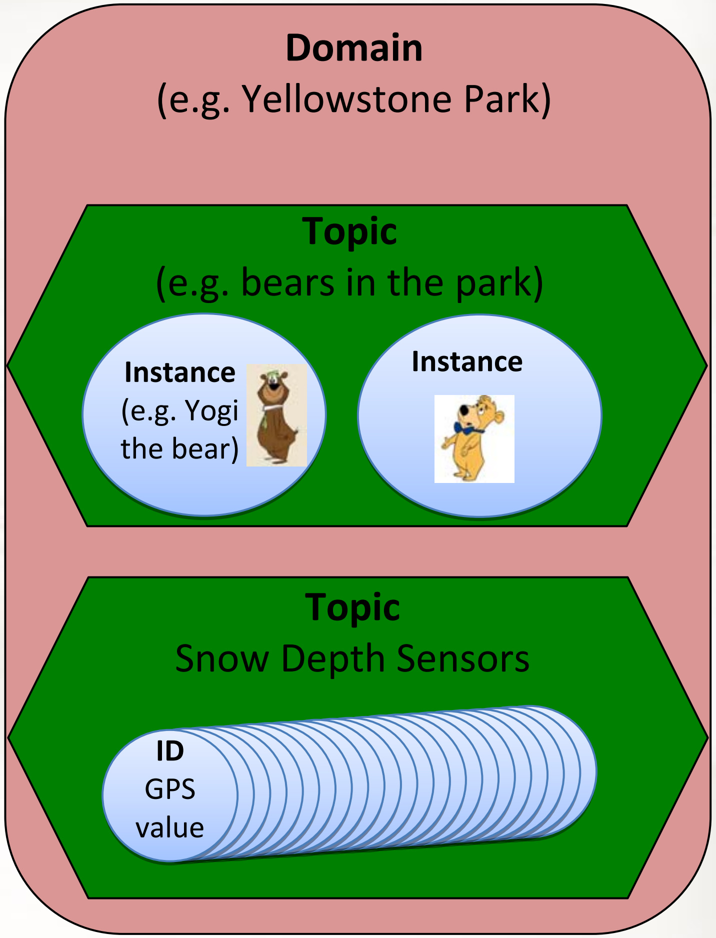
Blue-Force tracker replaced home-brew messaging w/ RTI Connext DDS:

- Tracks 20x more objects with fewer failures
- …with 97% less code (*1.5M lines → 50K*)
- …with 99% less CPU resources (*88 cores → 0.8*)



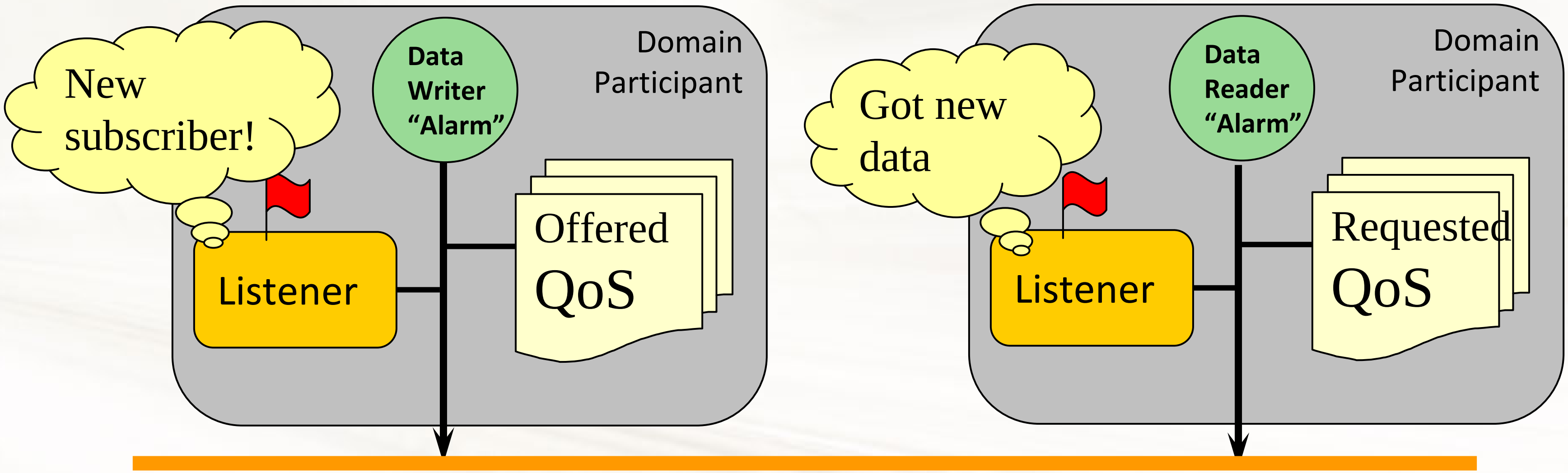
DDS Adressing:
Data-Objects in the Global Data Space

- **Domain:** world you're talking about
- **Topic**: group of similar objects
  - Similar structure ("type") *what*
  - Similar way they change over time ("QoS") *when* *how*
- **Instance**: individual object
- **DataWriter**: source of observations about a set of data-objects (Topic)
- **DataReader**: observer of a set of data-objects (Topic)



Data-Centric Communications Model

example



- **Participants** scope the global data space (domain)
- **Topics** define the data-objects (collections of subjects)
- **DataWriters** publish data on Topics
- **DataReaders** subscribe to data on Topics
- **QoS Policies** are used configure the system
- **Listeners** are used to notify the application of events

Data-Centric Qos-Aware Pub-Sub Model

Virtual, decentralized global data space

Source (Key)	Speed	Power	Phase
WPT1	37.4	122.0	-12.20
WPT2	10.7	74.0	-12.23
WPTN	50.2	150.07	-11.98

DDS

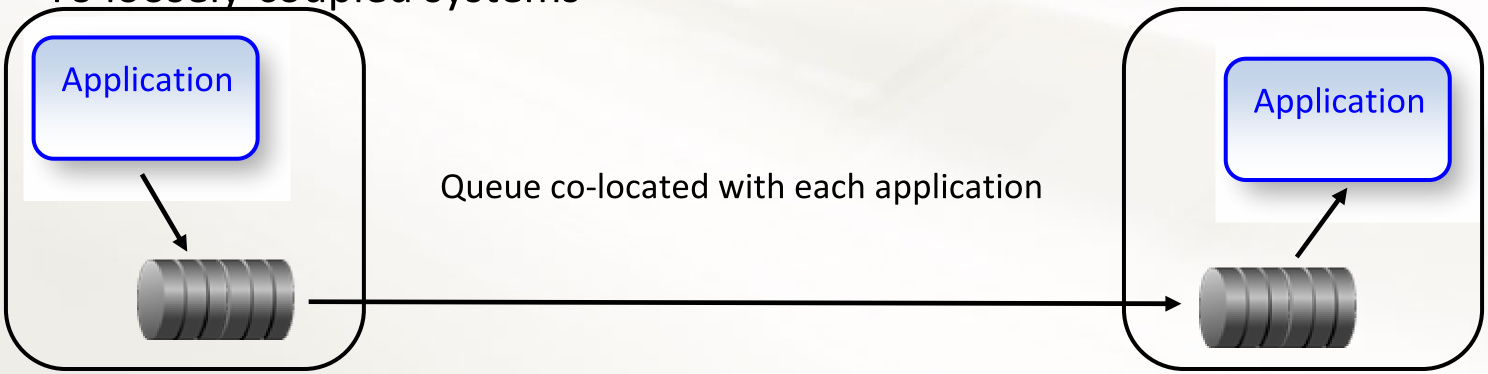
Persistence Service

Recording Service

CRUD operations

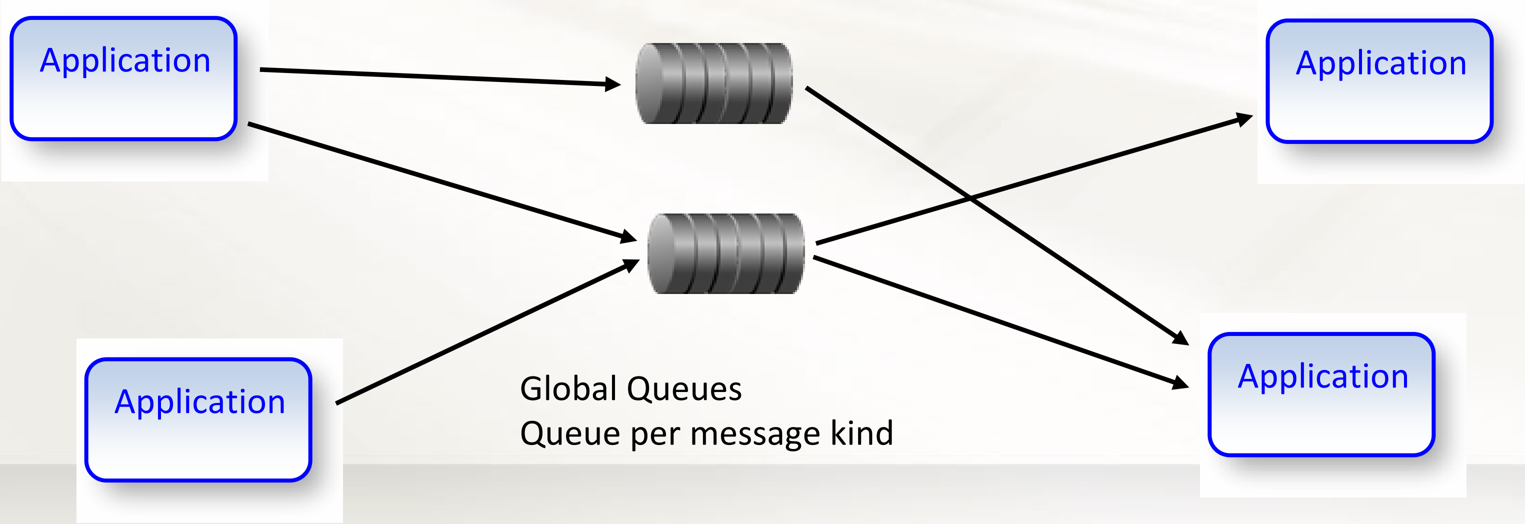
Do-yourself Message-Centric System

- Model
  - 1-1, FIFO
- Applications coupled in Lifespan & Content
  - Both must be present simultaneously
  - Everything sent is received
- Excellent performance
- Doesn't scale
  - To large-scale systems
  - To loosely-coupled systems



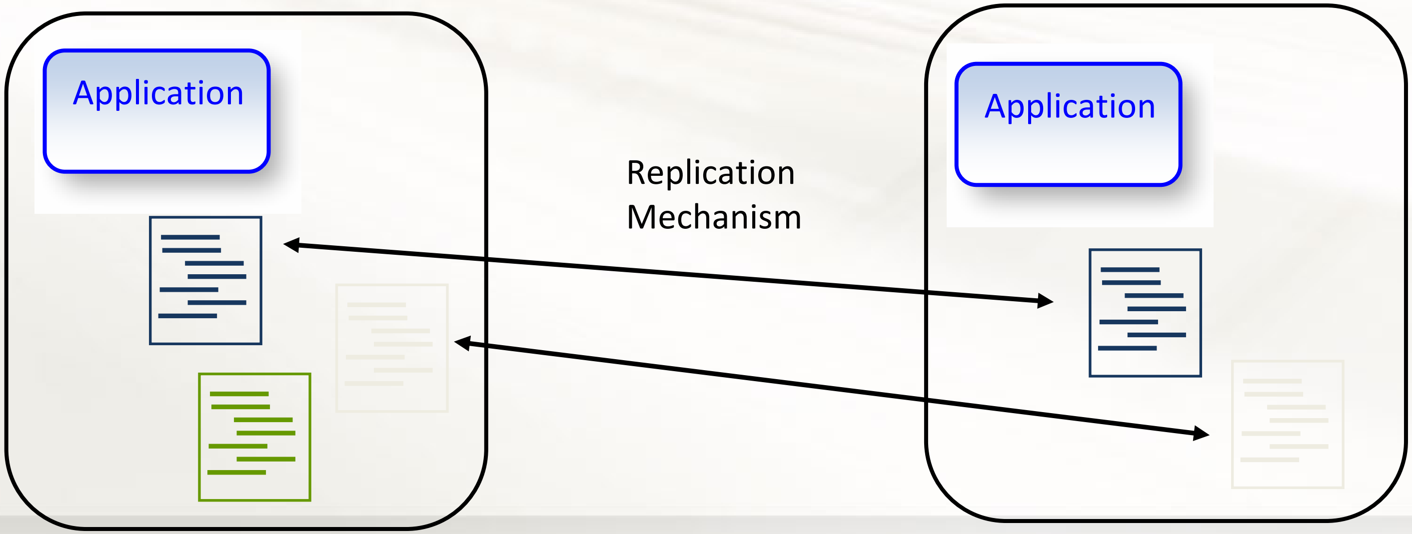
Middleware-based Message-Centric Systems (JMS)

- Model
  - Broker-based, n➔n communication
  - Independent messages, No state
- Coupling
  - Not coupled in Lifespan
  - Coupled in Order and Content (presentation)
- Worse performance
- Better scalability



Do-yourself Data-Centric System

- Model
  - Shared/replicated structured data/state
  - Asynchronous, Selective sharing
- Coupling:
  - Coupled in Lifespan, Decoupled in Presentation & Content
- Excellent performance & scalability



Middleware-based Data-Centric Systems (DDS)

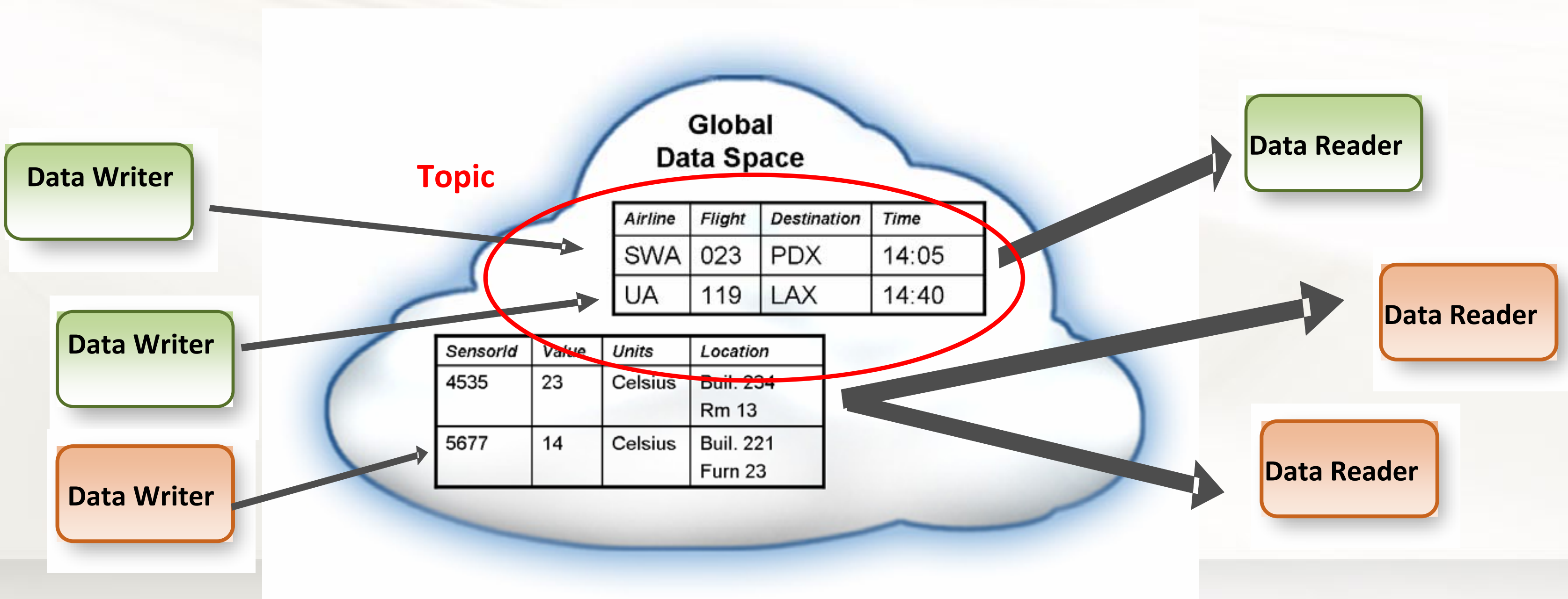
- Shared data-space, shared state
- Asynchronous communication
- Decoupled in Lifespan, Content, presentation
- Excellent performance & scalability
- Subsumes Message-Centric via QoS

Data-Centric Model

[example](#)

"**Global Data Space**" generalizes Subject-Based Addressing

- Data objects addressed by **DomainId, Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a "x.y.z …" formatted string

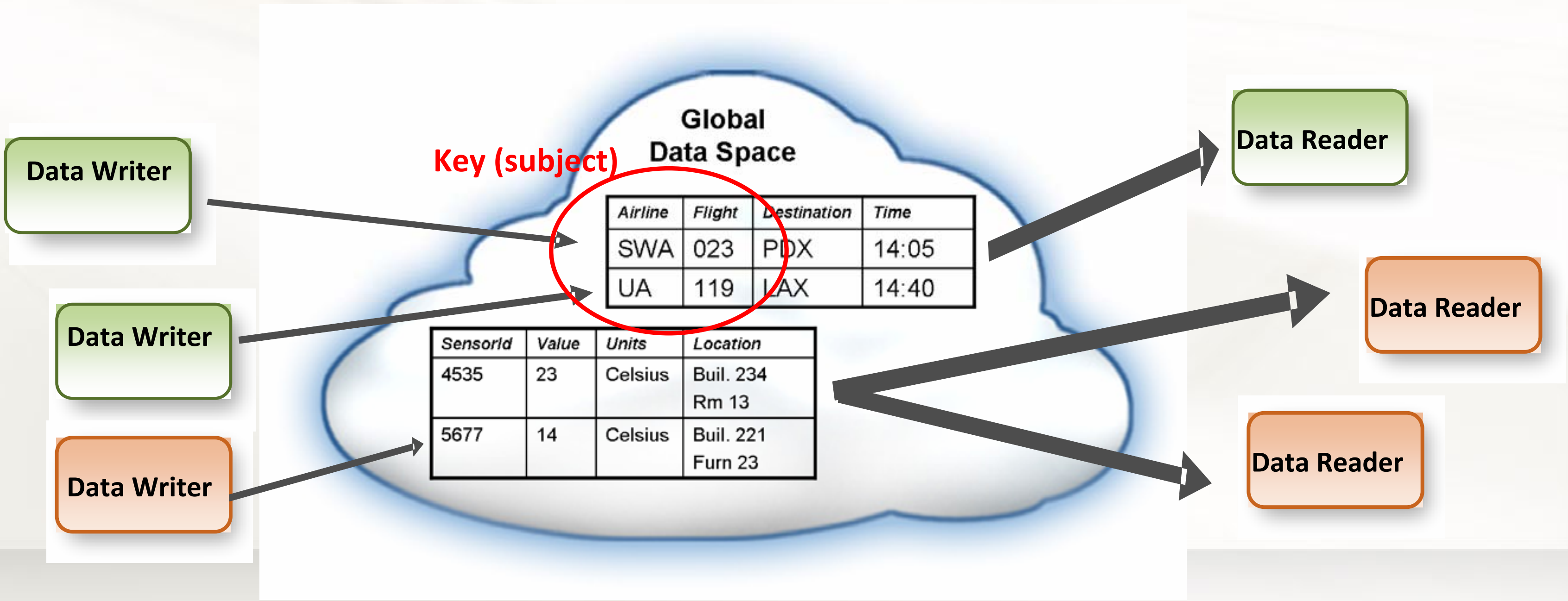


Data-Centric Model

example

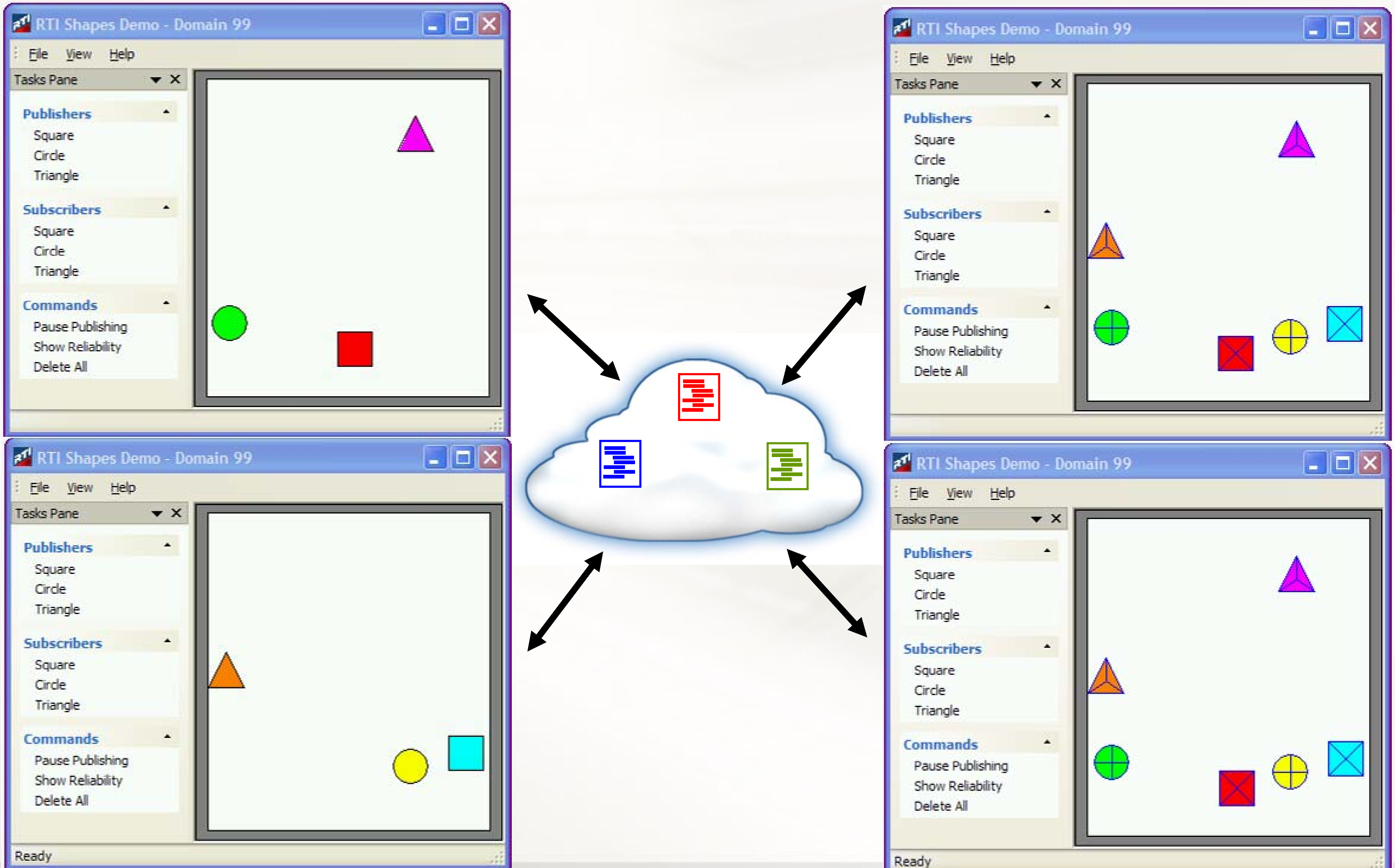
"**Global Data Space**" generalizes Subject-Based Addressing

- Data objects addressed by **DomainId, Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a "x.y.z …" formatted string



Demo: Publish-Subscribe

ShapesDemo



Real-Time Quality of Service (QoS)

QoS Policy
DURABILITY
HISTORY
READER DATA LIFECYCLE
WRITER DATA LIFECYCLE
LIFESPAN
ENTITY FACTORY
RESOURCE LIMITS
RELIABILITY
TIME BASED FILTER
DEADLINE
CONTENT FILTERS

Volatility

Infrastructure

Delivery

QoS Policy
USER DATA
TOPIC DATA
GROUP DATA
PARTITION
PRESENTATION
DESTINATION ORDER
OWNERSHIP
OWNERSHIP STRENGTH
LIVELINESS
LATENCY BUDGET
TRANSPORT PRIORITY

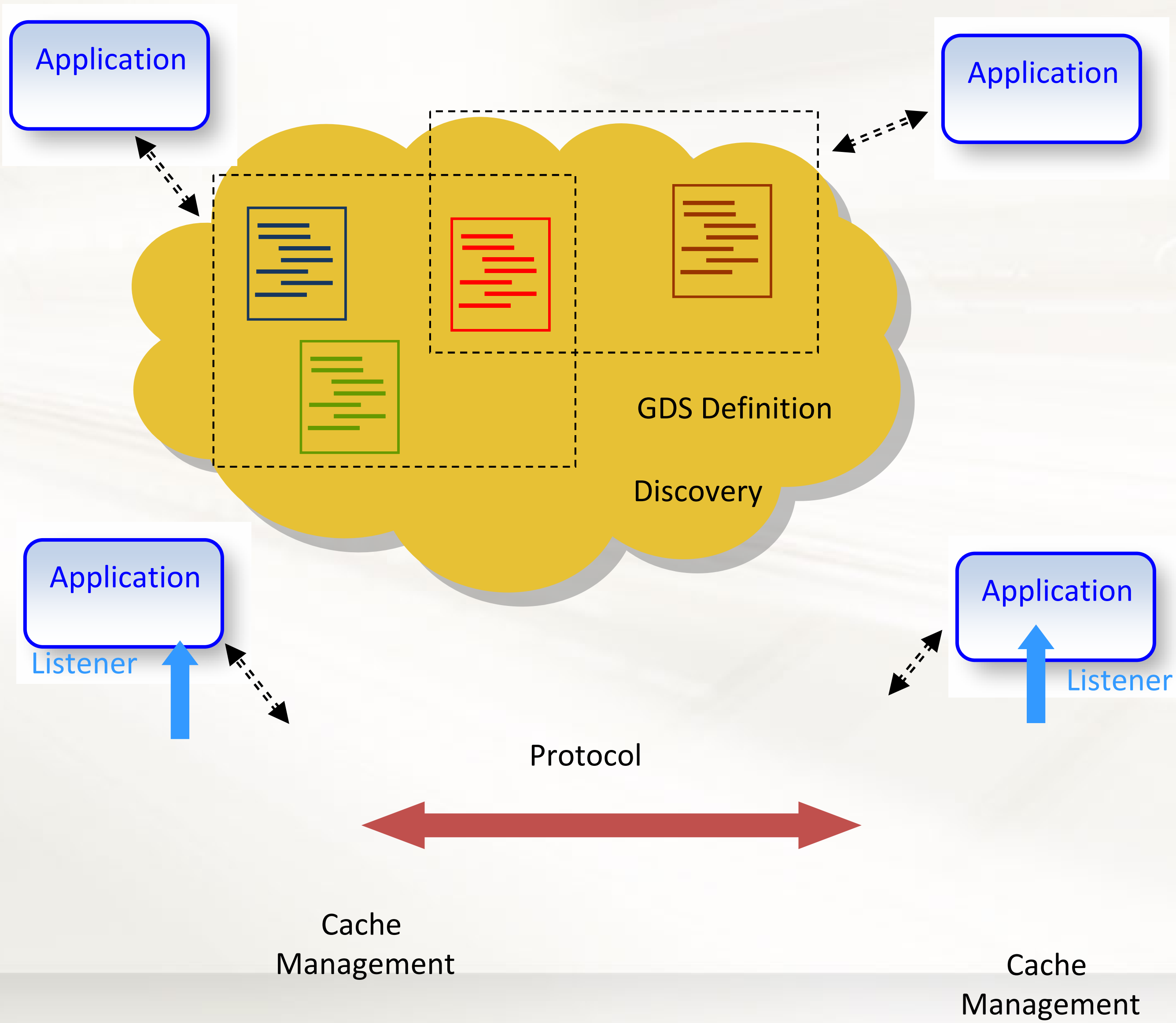
User QoS

Presentation

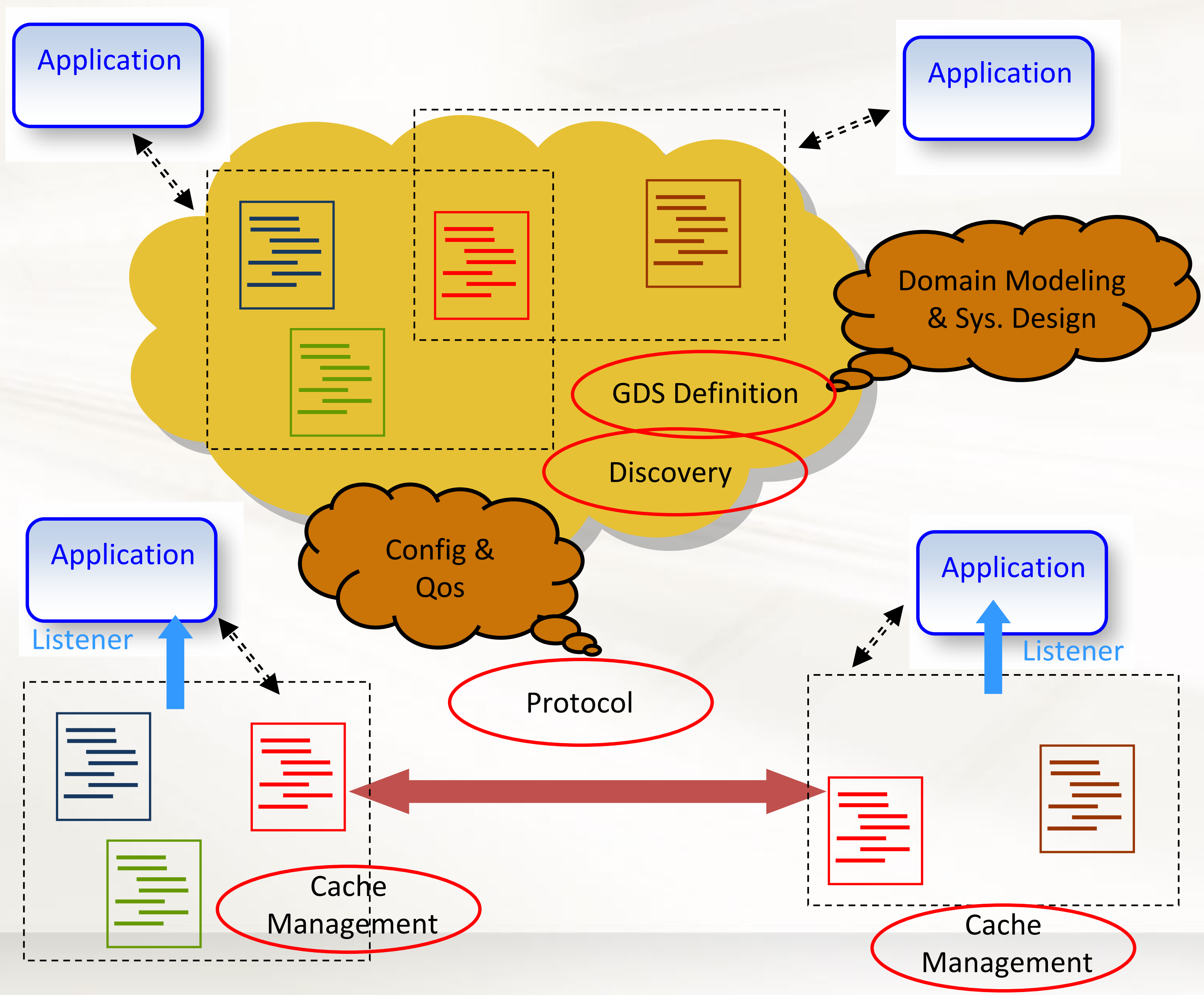
Redundancy

Transport

Components/Mechanics of the GDS

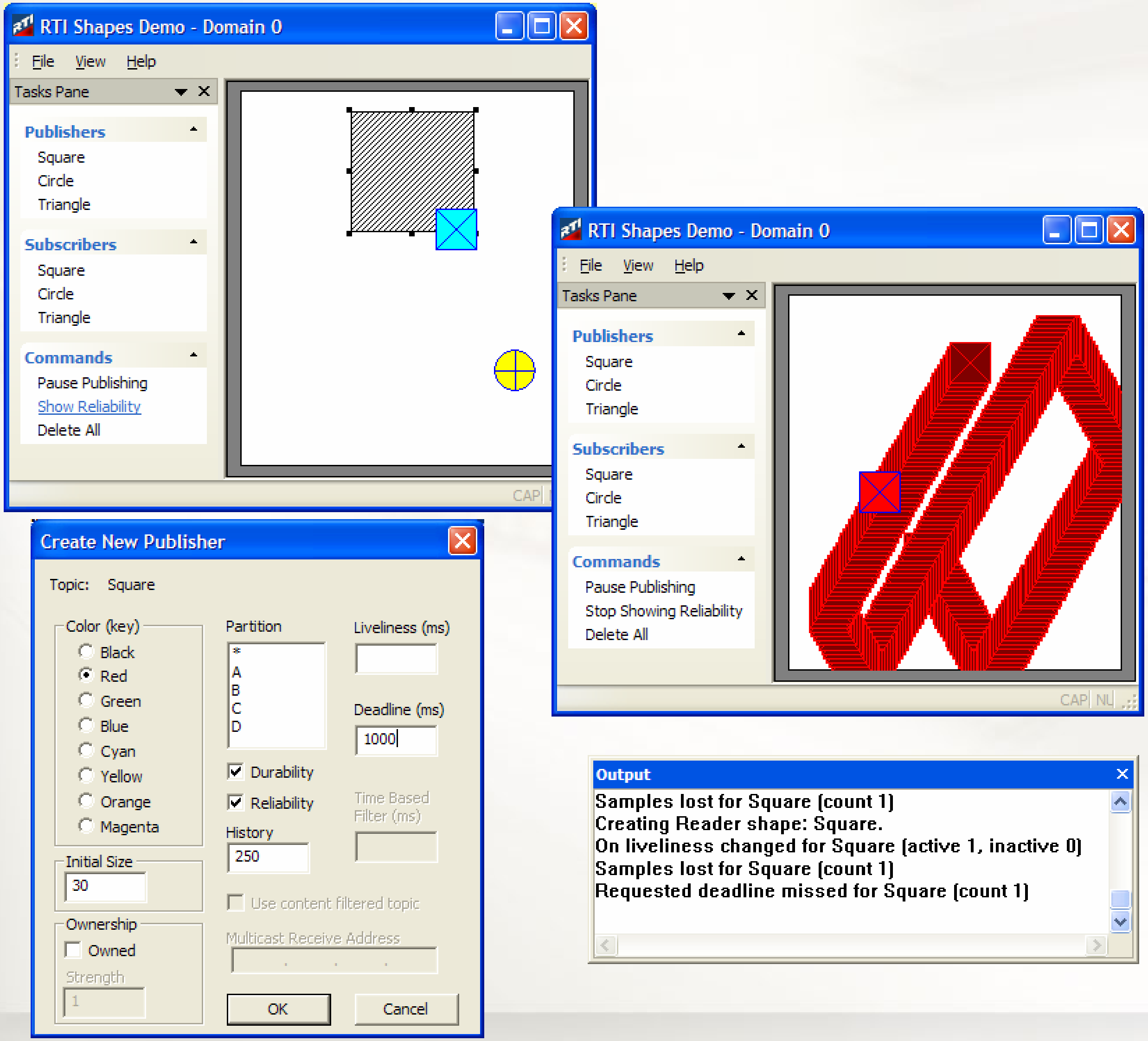


Components/Mechanics of the GDS



Let’s try the QoS

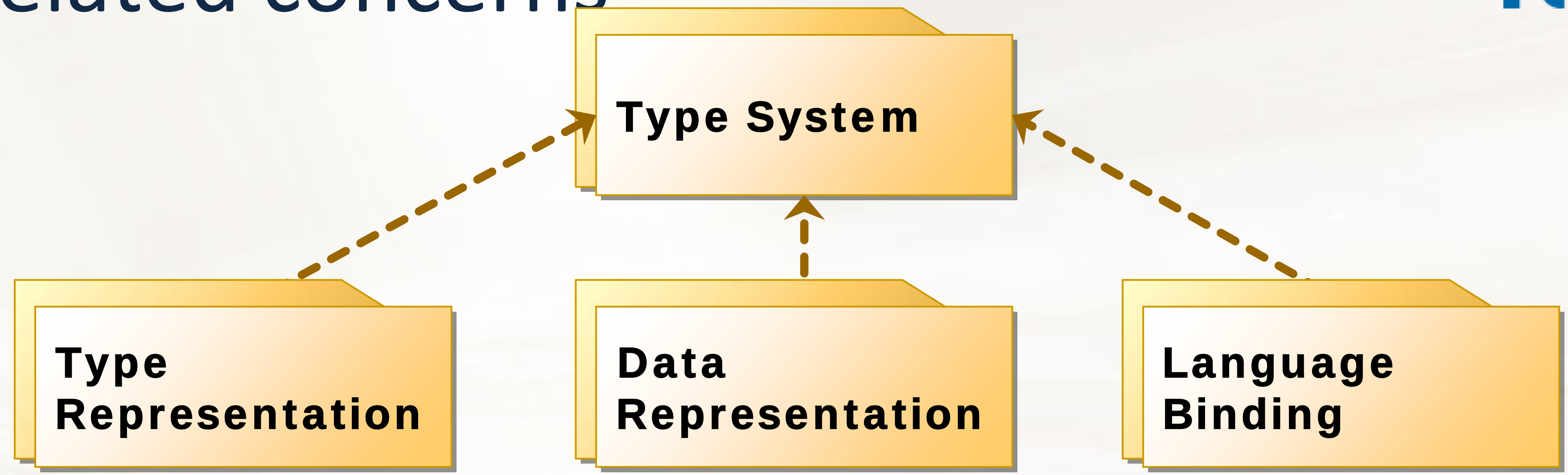
[ShapesDemo]



- Detecting presence
- History cache
- Deleting objects
- Ownership
- Liveliness
- Filtering
- Durability

DDS Type-System
(X-Types Specification)

Related concerns



- **Type System**: DDS data objects have a type
- **Language Binding**: Objects are manipulated using a Language Binding to some programming language
- **Data Representation**: Objects can be serialized for file storage and network transmission

- **Language Binding**: Types are manipulated using a Language Binding to some programming language
- **Type Representation**: Types can be serialized for file storage and network transmission

X-Types Overview

1. **Type System**: abstract definition of what types can exist
   - Expressed as UML *meta-model*
   - Including *substitutability*, *compatibility* rules
   - Mostly *familiar* from IDL
2. **Type Representations**: languages for describing types
   - *IDL*
   - *XML* and *XSD*
   - *TypeObject*
3. **Data Representations**: languages for describing data
   – *CDR*
   – *XML*

X-Types Overview

4. **Language Binding**: programming APIs
   – *"Plain Language"*: extension of existing IDL-to-*language* bindings
   – *Dynamic*: reflective API for types and objects, defined in UML (conceptual model) and IDL (API)

5. **Use by DDS**: application of type/data representations to middleware
   – Data *encapsulation*, QoS *compatibility*
   – *Type compatibility* as applied to endpoint matching
   – *Built-in types*

Example

Type Representation

IDL:

Foo.idl

```
struct Foo {
 string name;
 long ssn;
};
```

Language Binding

IDL to Language Mapping:

Foo.h

Foo.c

FooTypeSupport.c

```
    struct Foo {
      char *name;
      int ssn;
    };

Foo f = {"hello", 2};
```

Data Representation

IDL to CDR:

```
00000006
68656C6C
6F000000
00000002
```

Type System Overview

* *Type System in not defined in terms of IDL. I'm explaining in terms of IDL for clarity.*

- Entirely familiar from IDL:
  - Primitives
  - Strings, narrow and wide
  - Arrays and sequences
  - Aliases (`typedefs`)
  - Unions
  - Modules
- As in IDL, but extended:
  - *Structures*— **including single inheritance**
  - *Enumerations*— **specify bit width and constant values**
- New relative to IDL:
  - *Maps*—like `std::map` or `java.util.Map`;
  - *Annotations*—for extensibility

```
struct OriginalLandData {
    long x;
    long y;
};
```

Type System Extensibility: Metadata

- **Annotation = structured metadata** attached to type or type member
  - *Annotation type* defines annotation members, their names, and their types
  - *Annotation usage* provides values for annotation type's members
- **Established programming practice**
  - In Java: `@MyAnnotation  int x;`
  - In C#: `[MyAnnotation] int x;`

DB Table
name: string

DB Table
name = "Track"

Metadata in IDL: Non-Standard Metadata

- **Java-like annotation syntax**: already familiar
- **Annotation type**:

```
@Annotation local interface MyAnnotation {
    long value1();
    double value2();
};
```

- **Annotation usage**:

```
struct MyStruct {
    @MyAnnotation(value1 = 42, value2 = 42.0)
    long my_field;
};
```

The Example

- Radar system uses “Original Land Data” (OLD):

```
struct OriginalLandData {
    long x;
    long y;
};
```

Type System Extensibility: Structures

Collection of members, of same or different types.

Each member has:

- A *unique name* (string)
- A *unique ID* (unsigned integer)
- Additional *metadata*:
  - *Key / NotKey*
  - *Optional vs. required*—Does value always exist? (*6.5.6, 6.5.19*)
    - Important semantics: Think null vs. non-null value
    - Universal concept: C, C++, Java, .Net, SQL, XSD, …
    - Important for interop: If I have a field in my type that you don't, what do I do with data from you?
    - Important for representational efficiency: Skip a field with well-defined semantics
    - *Note*: Keys always required; otherwise, identity breaks down
  - *Shareable/ NotShareable*
    - Member data should not be stored in-line with container
    - Certain language mappings will map to a pointer
    - Allows writing shared data without making extra copies

Type Extensibility: Example

is-assignable-from

```
// OLD
struct OriginalLandData {
     long x;
     long y;
};
```

! is-assignable-from

```
// NEW1
struct NextEnhancedWorldview1 {
     long x;
     long y;
     TrackKindEnum kind;
};

// NEW2
struct NextEnhancedWorldview2 {
     long y; // out of order!
     long x; // out of order!
};
```

But see later how to make it assignable…

Type Compatibility: Example (Revisited)

```
// OLD
@Extensibility(MUTABLE_EXTENSIBILITY)
struct OriginalLandData {
    long x;
    long y;
};
```

is-assignable-from

is-assignable-from

```
// NEW1
@Extensibility(MUTABLE_EXTENSIBILITY)
struct NextEnhancedWorldview1 {
    long x;
    long y;
    TrackKindEnum kind;
};
```

```
// NEW2
@Extensibility(MUTABLE_EXTENSIBILITY)
struct NextEnhancedWorldview2 {
    @ID(1) long y;
    @ID(2) long z;
    @ID(0) long x;
};
```

Hand’s on DDS

The first application

Hands-on Example (C++)

Type Definition

MyType.idl

rtiddsgen

MyType.sln

MyType.h

MyTypeSupport.c

MyTypePublisher.cpp

MyTypeSubscriber.cpp

compiler

Publisher.exe

Subscriber.exe

Three minutes to a running app!!

1. Define your data
2. Create your project
3. Build
4. Run: publisher subscriber

Aux:

File Browser

Console

Delete Files

rtiddsspy

Example #1 - Hello World

We will use this data-type :

```
const long MSG_LEN=256;
struct Hello {
  string<MSG_LEN> user; //@key
  string<MSG_LEN> msg;
};
```

Side Note:
IDL vs. XML

The same data-type can also be described in XML. This is part of the DDS X-Types specification

```
<const name="MSG_LEN" type="long" value="256"/>
<struct name="Hello">
  <member name="user" key="true” type="string"
     stringMaxLength="MSG_LEN”/>
  <member name="msg" type="string”
     stringMaxLength="MSG_LEN" />
</struct>
```

Generate type support (for C++) [Linux]

rtiddsgen HelloWorld.idl -language C++ -example i86Linux2.6gcc4.4.3\
-replace -ppDisable

- Look at the directory you should see:
  - makefile_hello_i86Linux2.6gcc4.4.3
  - And Several other files…
- Open the source files:
  HelloMsgPublisher.cxx
  HelloMsgSubscriber.cxx
- Compile:
  make –f makefile_hello_i86Linux2.6gcc4.4.3

Generate type support (for Java)

```
rtiddsgen HelloWorld.idl -language Java -example i86Linux2.6gcc4.4.3jdk\
                    -replace -ppDisable
```

- Look at the directory you should see:
  - makefile_hello_i86Linux2.6gcc4.4.3jdk
  - And Several other files…
    - Look at HelloMsgPublisher.java
    - Look at HelloMsgSubscriber.java

- You can use the makefile to build and the Java programs:

  gmake –f makefile_hello_i86Win32jdk

Execute the program [Windows]

- C++:
  - On one window run:
    - objs\i86Win32VS2005\HelloMsgPublisher.exe
  - On another window run:
    - objs\i86Win32VS2005\HelloMsgSubscriber.exe
- Java
  - On one window run:
    - gmake –f makefile_hello_i86Win32jdk HelloMsgPublisher
  - On another window run:
    - gmake –f makefile_hello_i86Win32jdk HelloMsgSubscriber

- You should see the subscribers getting an empty string…

Execute the program [Linux]

- C++:
  - On one window run:
    - objs/i86Linux2.6gcc4.4.3/HelloMsgPublisher.exe
  - On another window run:
    - objs/i86Linux2.6gcc4.4.3/HelloMsgSubscriber.exe
- Java
  - On one window run:
    - gmake –f makefile_hello_i86Linux2.6gcc4.4.3jdk HelloMsgPublisher
  - On another window run:
    - gmake –f makefile_hello_i86Linux2.6gcc4.4.3jdk HelloMsgSubscriber

- You should see the subscribers getting an empty string…

Example: Publication

```
// Entities creation
DomainParticipant participant =
    TheParticipantFactory->create_participant(
   domain_id, participant_qos, participant_listener);

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

Topic topic = domain->create_topic(
    “MyTopic”, “Text”, topic_qos, topic_listener);

DataWriter writer = publisher->create_datawriter(
    topic, writer_qos, writer_listener);

TextDataWriter twriter = TextDataWriter::narrow(writer);


TextStruct my_text;
twriter->write(&my_track);
```

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 )
{
    TextSeq received_data;
    SampleInfoSeq sample_info;
    TextDataReader reader = TextDataReader::narrow(reader);

    treader->take( &received_data, &sample_info, …)
    // Use received_data
    printf(“Got: %s\n”, received_data[0]->contents);
}
```

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);

// Wait returns when there is data (or timeout)
FooSeq received_data;
SampleInfoSeq sample_info;

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

// Use received_data
printf(“Got: %s\n”, received_data[0]->contents);
```

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 choice:

- 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
RUQESTED_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;
}
```

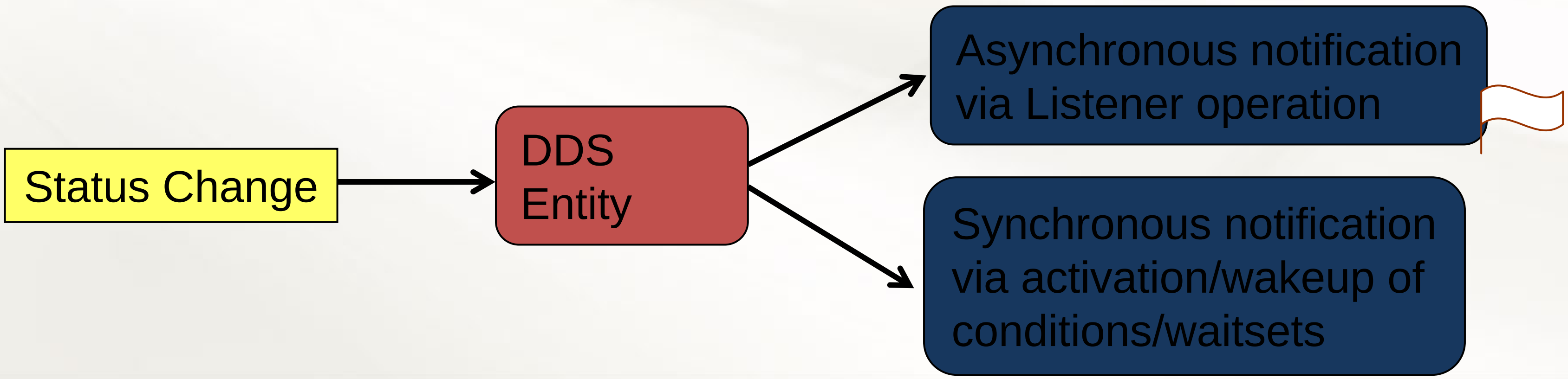
Listeners, Conditions and Statuses

- 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
RUQESTED_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



Example #2 - Command-Line Shapes

We will use this data-type :

```
const long STR_LEN=24;
struct ShapeType {
  string<MSG_LEN> color; //@key
  long x;
  long y;
  long shapesize;
};
```

Example #2 - Command-Line Shapes

- Edit the publisher and subscriber
  – Change the TopicName to “Square” (or “Circle” or “Triangle”)

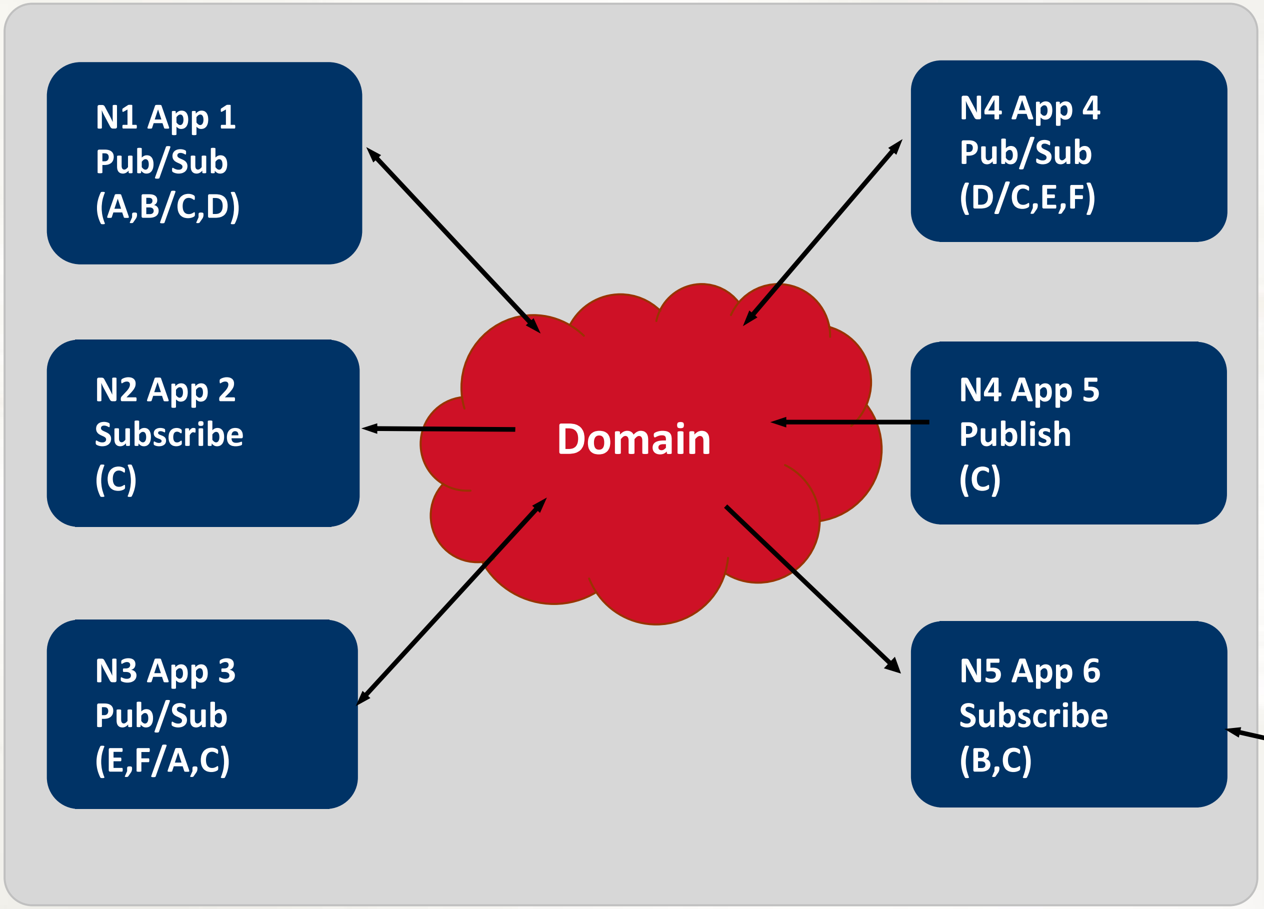
- Change the publisher to do something interesting
  - Use colors such as “GREEN” “RED” “YELLOW”
  - Keep the ‘x’ and ‘y’ between 0 and 260
  - Keep the ‘shapesize’ between 0 and 80

Using DDS
Common Use-Cases

Common use cases

1. Isolating Subsystems
2. Detecting presence of applications
3. Discovering who is publishing/subscribing what
4. Publishing data that outlives its source
5. Keeping a “last-value” cache of objects
6. Monitoring and detecting the health of application elements
7. Building a highly-available system
8. Limiting data-rates
9. Controlling data received by interest set

1. Isolating Subsystems: Domain and Domain Participants

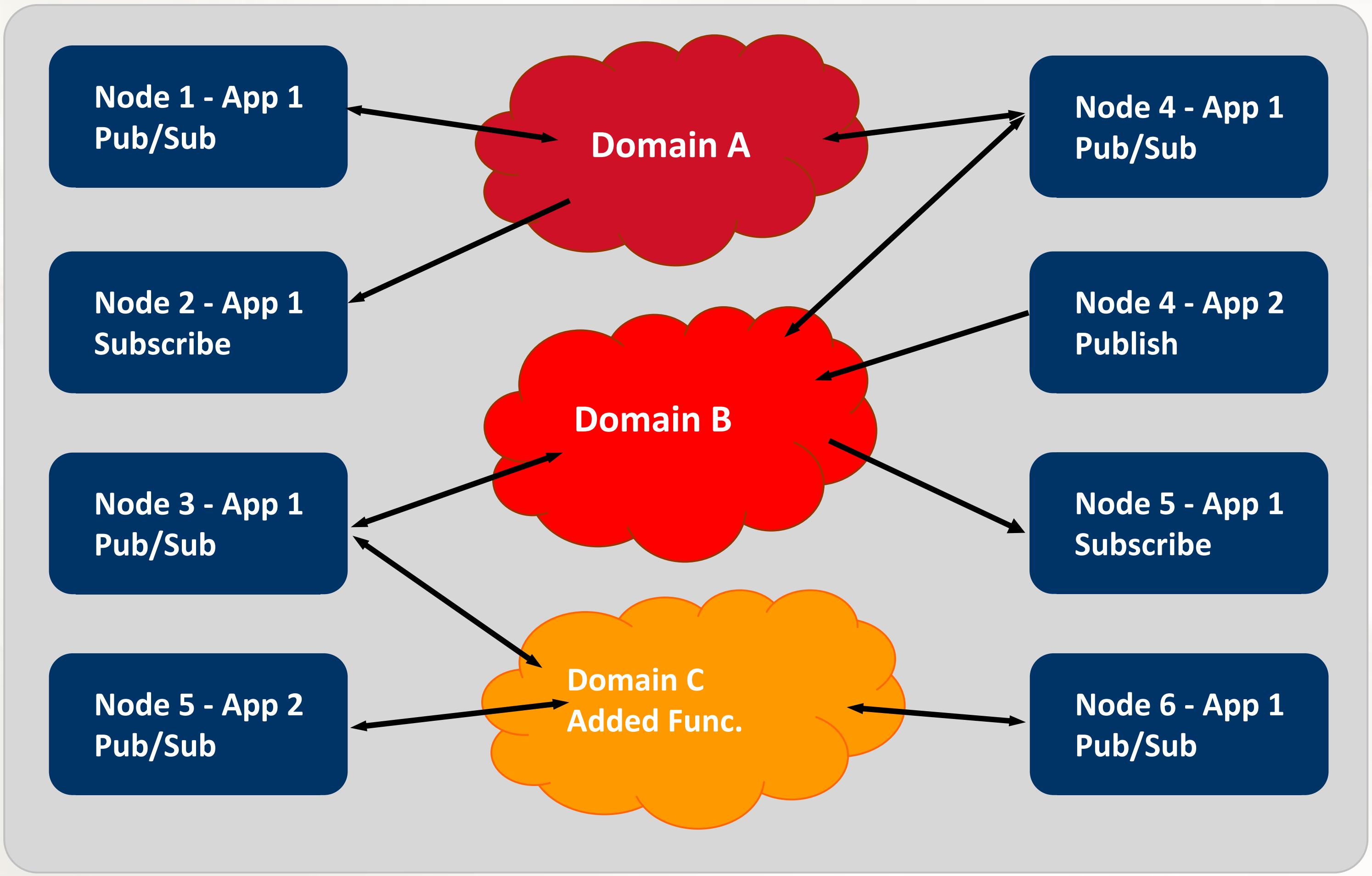


Single 'Domain' System

- Container for applications that want to communicate
- Applications can join or leave a domain in any order
- New Applications are "Auto-Discovered"
- An application that has joined a domain is also called a "Domain Participant"

1. Isolating Subsystems: Domain and Domain Participants

Using Multiple domains for Scalability, Modularity & Isolation



[demo_domain_0] Domain System [demo_domain_1]

2. Detecting presence of applications DDS builtin Discovery Service

- DDS provides the means for an application to discover the presence of other participants on the Domain
  - The Topic “DCPSParticipants” can be read as a regular Topic to see when DomainParticipants join and leave the network
- Applications can also include meta-data that is sent along by DDS discovery

shapes_demo **discovery_in_excel**

3. Discovering who is publishing/subscribing
DDS builtin Discovery Service

- DDS provides the means for an application to discover all the other DDS Entities in the Domain
  - The Topics “DCPSPublications”, “DCPSSubscriptions”, “DCPSTopics”, and “DCPSParticipants” be read to observe the other entities in the domain

shapes_demo

discovery_in_excel

Example: Accessing discovery information

```
reader = participant
  ->get_builtin_subscriber()
  ->lookup_datareader("DCPSSubscription");

reader_listener = new DiscoveryListener();
reader->set_listener(reader_listener);
```

Example: Displaying discovery information

```
SubscriptionBuiltinTopicData *subscriptionData =
   new SubscriptionBuiltinTopicData();
SampleInfo *info = new SampleInfo();

do {
   retcode =
      subscriptionReader->take_next_sample( *subscriptionData, *info);

   SubscriptionBuiltinTopicDataTypeSupport
      ::print_data (subscriptionData);

}  while ( retcode != RETCODE_NO_DATA );
```

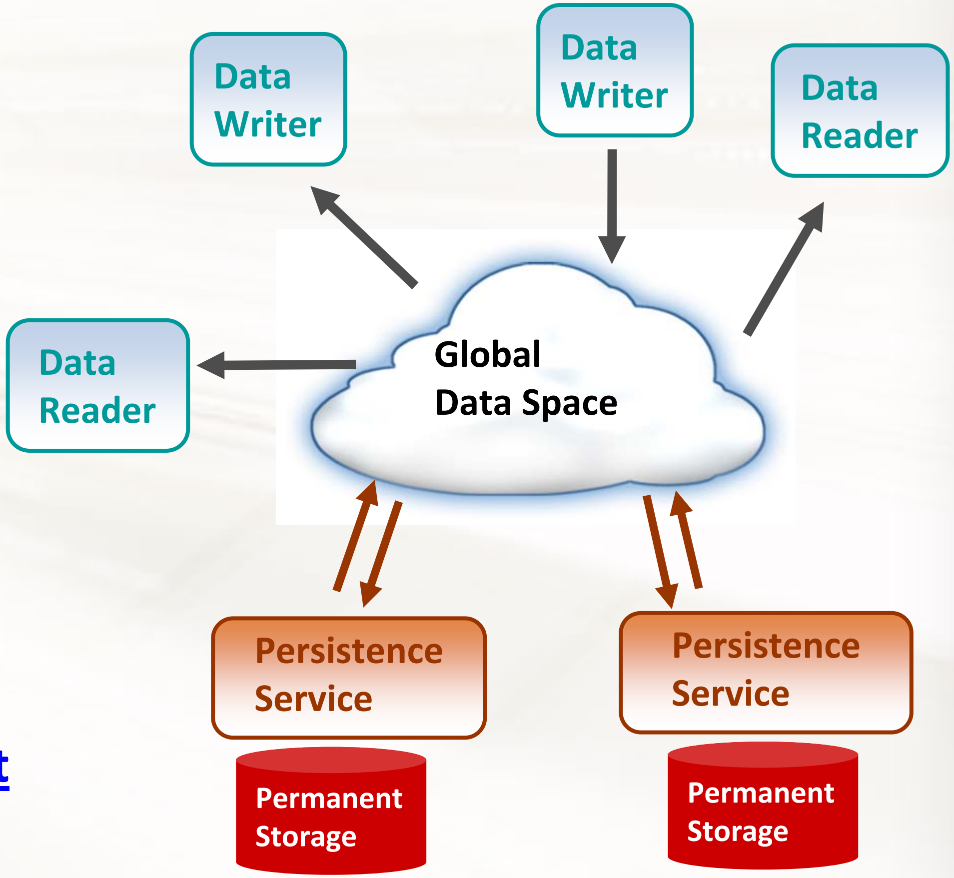
4. Publishing data that outlives its source: DDS DURABILITY QoS

DURABILITY QoS can be set to:

- VOLATILE -- No durability (default)
- TRANSIENT_LOCAL
  - Durability provided by the DataWriter
  - Late joiners will get data as long as writer is still present
- TRANSIENT
  - Durability provided by external “persistence” service
  - Late joiners will get data as long as persistence is still present
- PERSISTENT
  - Durability provided by external “persistence” service
  - Persistence service must store/sync state to permanent storage
  - Persistence service recover state on re-start
  - Late joiners will get data even if persistence service crashes and re-starts

4. Publishing data that outlives its source: Persistence Service

A service that persists data outside of the context of a DataWriter



Demo:

1. PersistenceService
2. ShapesDemo
3. Application failure
4. Application (ShapesDemo) re-start

5. Keeping a "Last value" cache

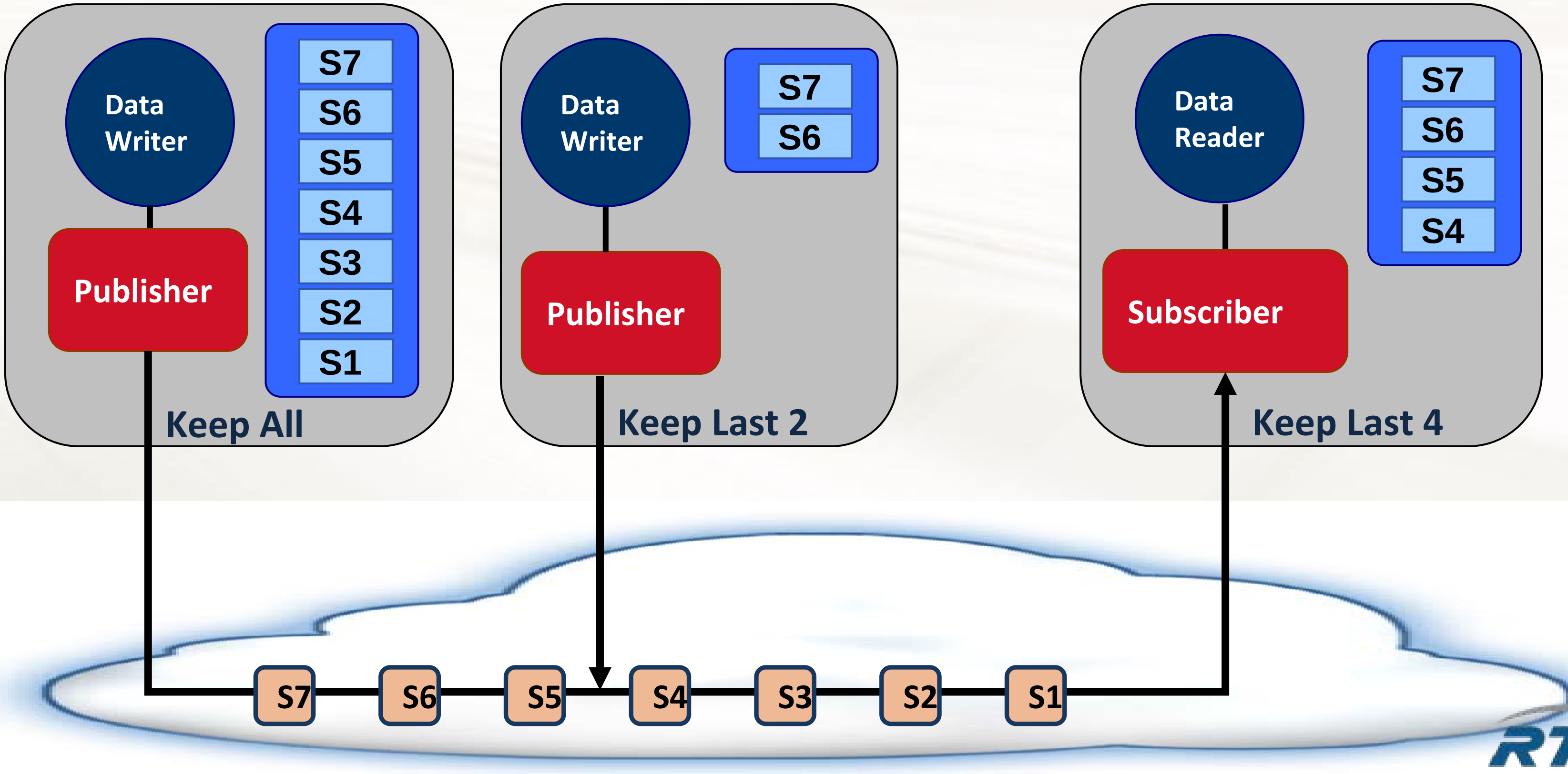
- A last-value cache is already built-in into every Writer in the system
  - Can used in combination with a Durable Writer
- A late joiner will automatically initialize to the last value
- Last value cache can be configure with history depth greater than 1
- The Persistence Service can be used to provide a last value cache for durable data

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

demo_history



5. Monitoring the health of applications: Liveliness QoS – Classic watchdog/ deadman switch

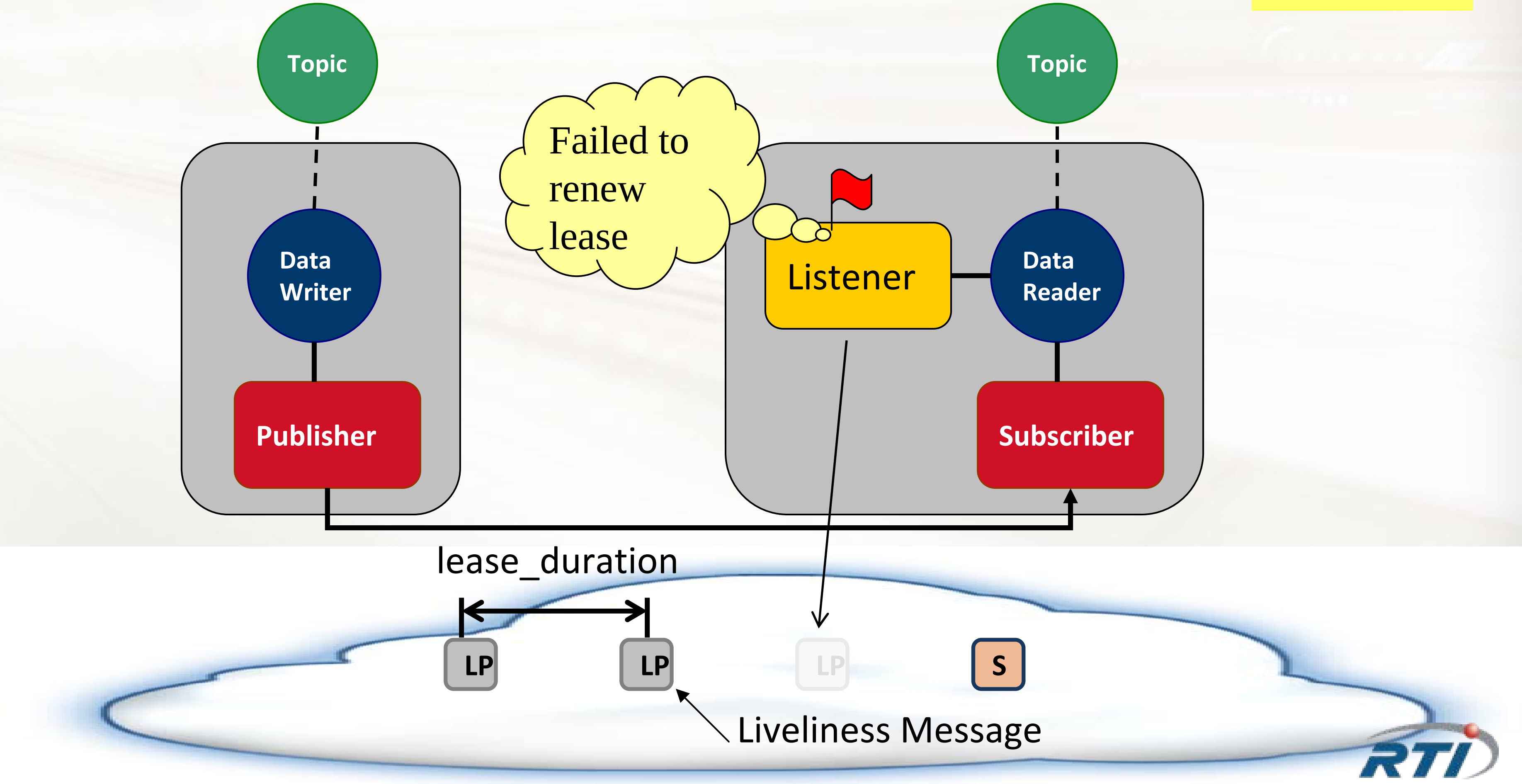
- DDS can monitor the presence, health and activity of DDS Entities (Participant, Reader, Writer)
- Use Liveliness QoS with settings
  - AUTOMATIC
  - MANUAL_BY_PARTICIPANT
  - MANUAL_BY_TOPIC
- This is a request-offered QoS
- Answers the question: “Is no news good news?”

QoS: Liveliness: Type and Duration

liveliness_example

Type: Controls who is responsible for issues of 'liveliness packets'
AUTOMATIC = Infrastructure Managed
MANUAL = Application Managed

kill_apps

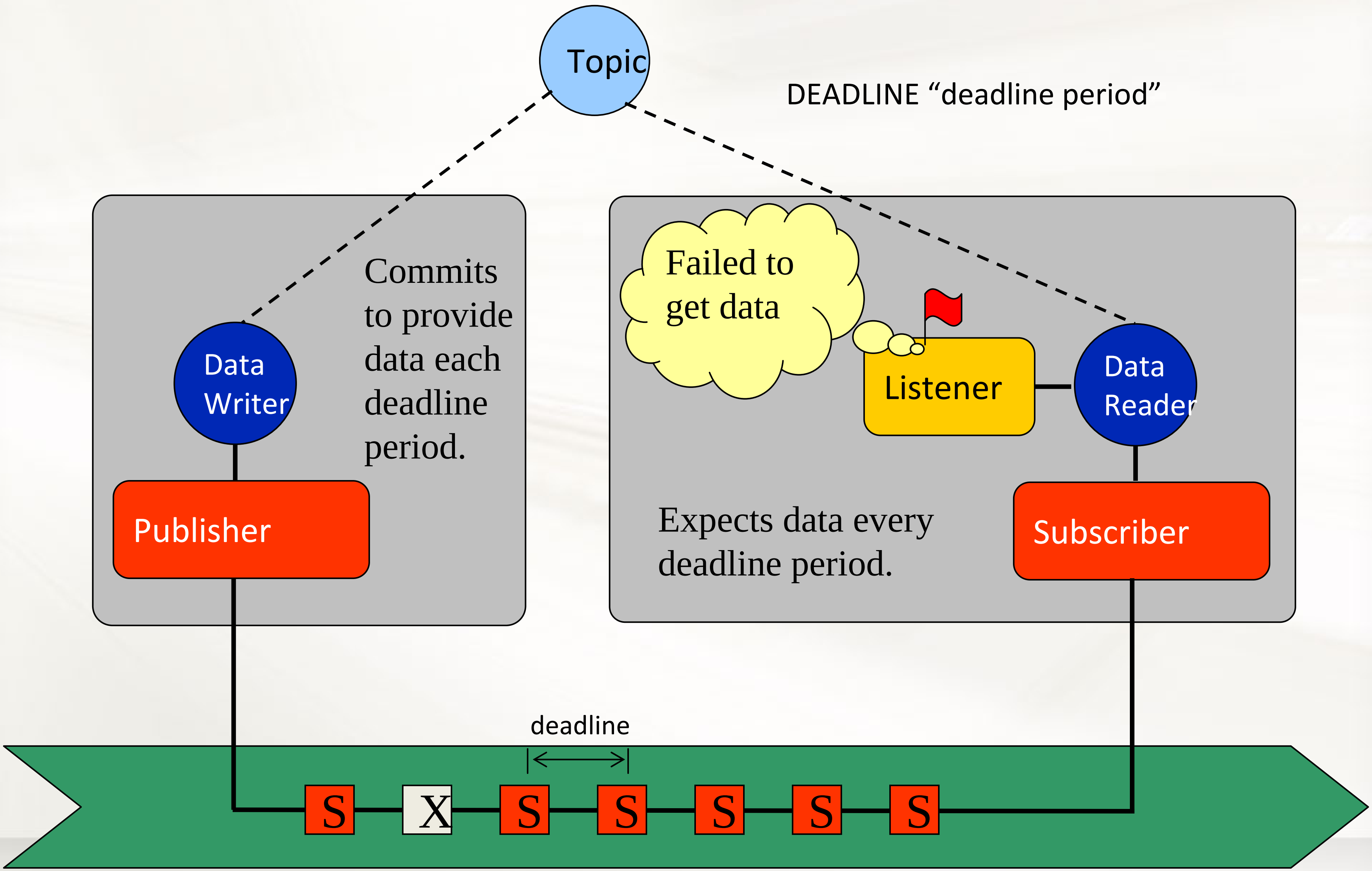


5. Monitoring the health of data-objects: Deadline QoS

- DDS can monitor activity of each individual data-instance in the system
- This is a request-offered QoS
- If an instance is not updated according to the contract the application is notified.
- Failover is automatically tied to this QoS

QoS: Deadline

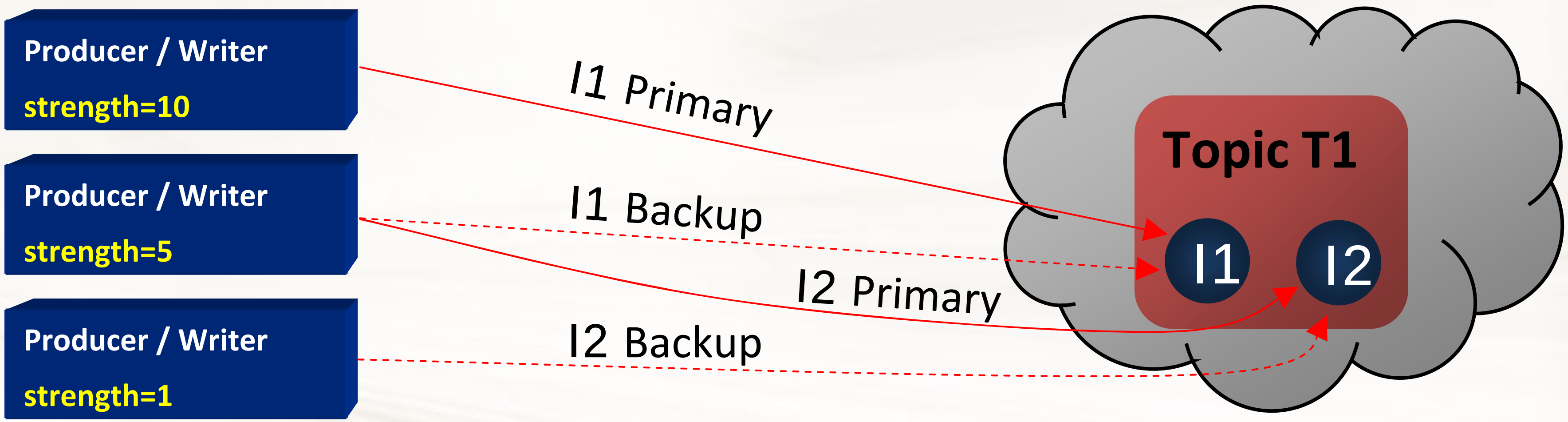
deadline example



5. Building a highly-available system

- HA systems require combining multiple patters, many directly supported by DDS:
  - Detection of presence -> DDS Discovery
  - Detection of Health and activity -> DDS LIVELINESS
    -> DDS DEADLINE
  - Making data survive application & system failures
    -> DDS DURABILITY
  - Handling redundant data sources and failover
    -> DDS OWNERSHIP

Ownership and High Availability



- Owner determined per subject
- Only extant writer with highest strength can publish a subject (or topic for non-keyed topics)
- Automatic failover when highest strength writer:
  - Loses liveliness
  - Misses a deadline
  - Stops writing the subject

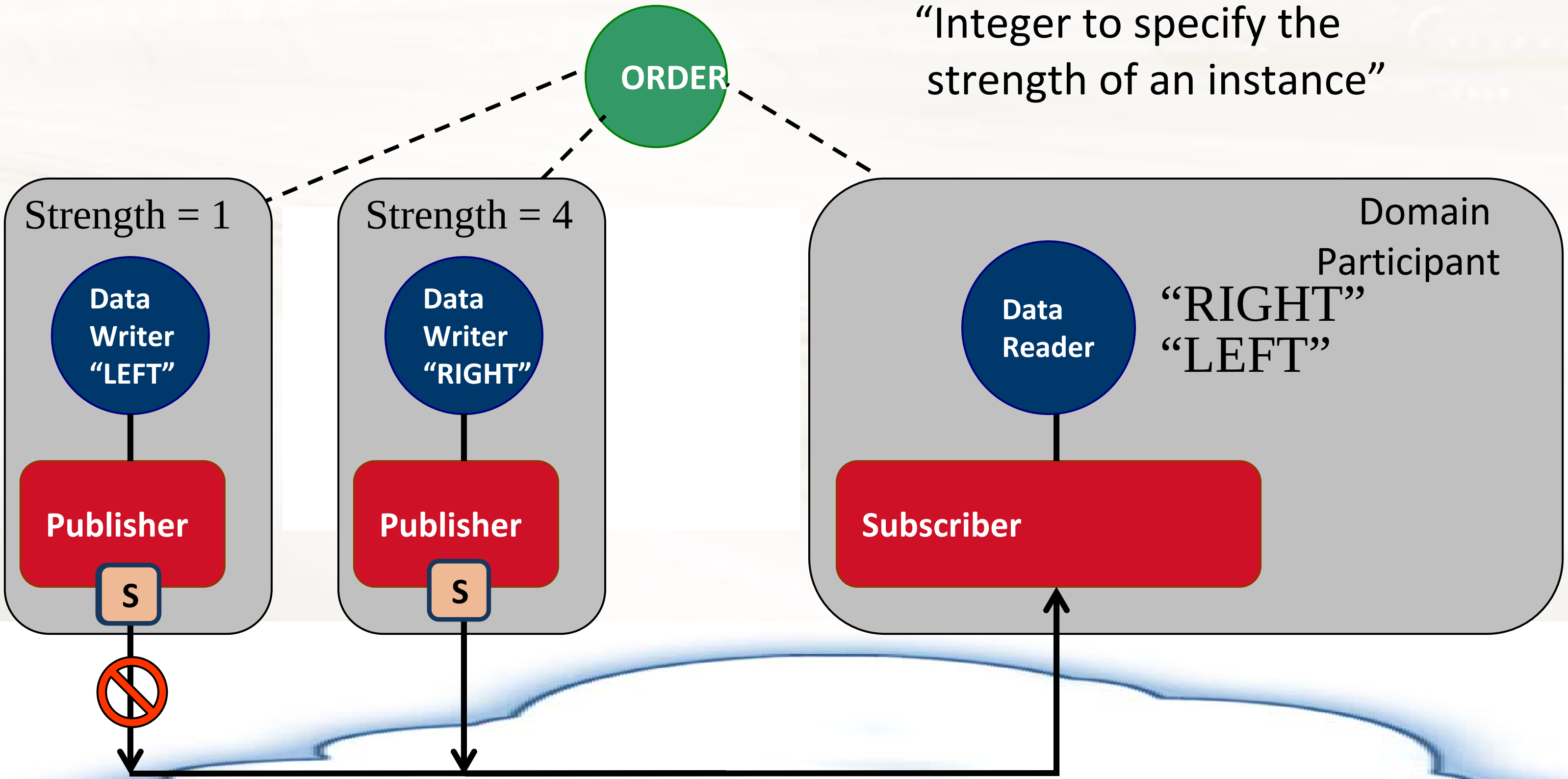
Start demo

- Shared Ownership allows any writer to update the subject [demo_ownership](#)

QoS: Ownership Strength

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

OWNERSHIP_STRENGTH
"Integer to specify the strength of an instance"

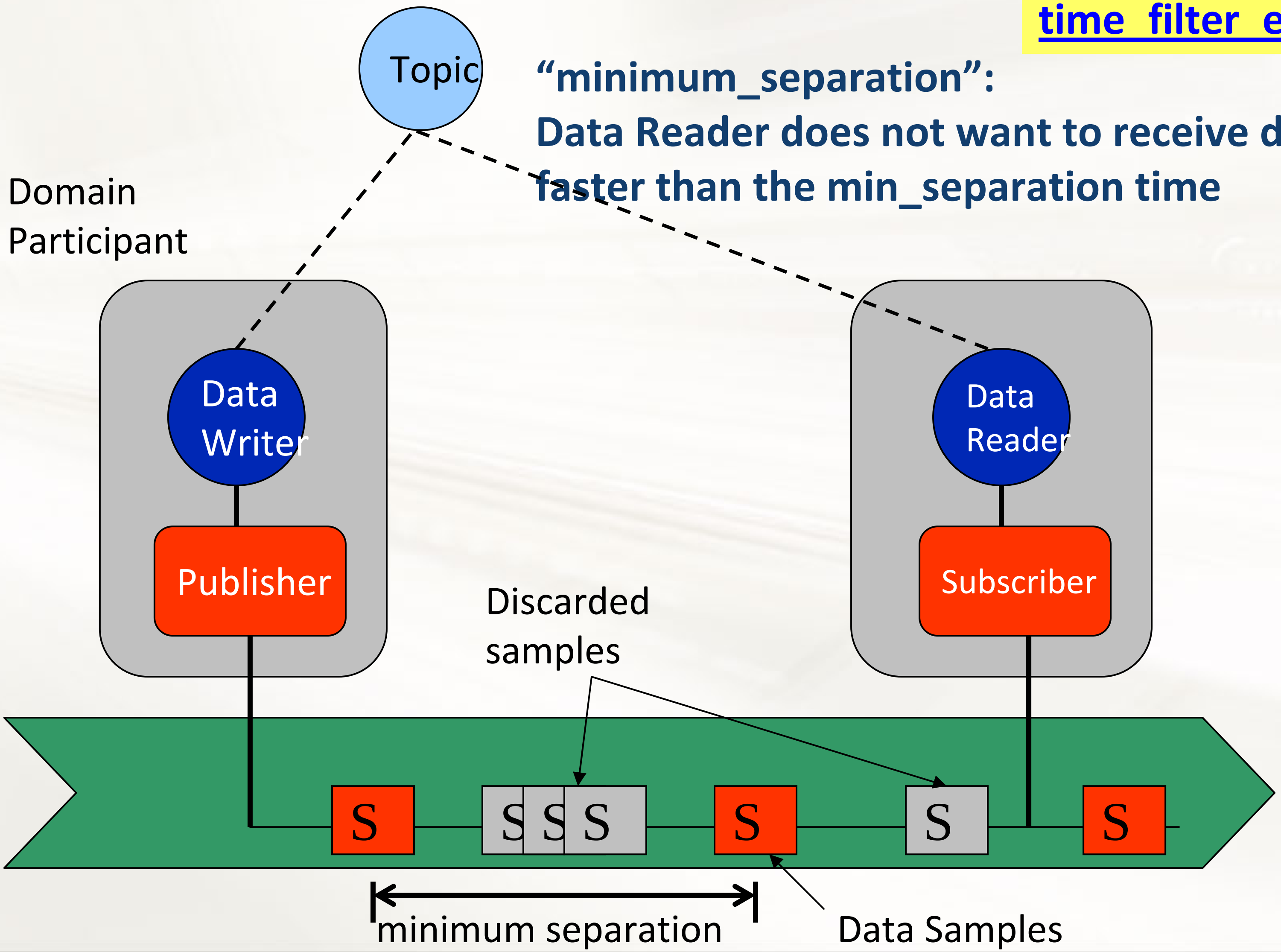


Note: Only applies to Topics with Ownership = Exclusive

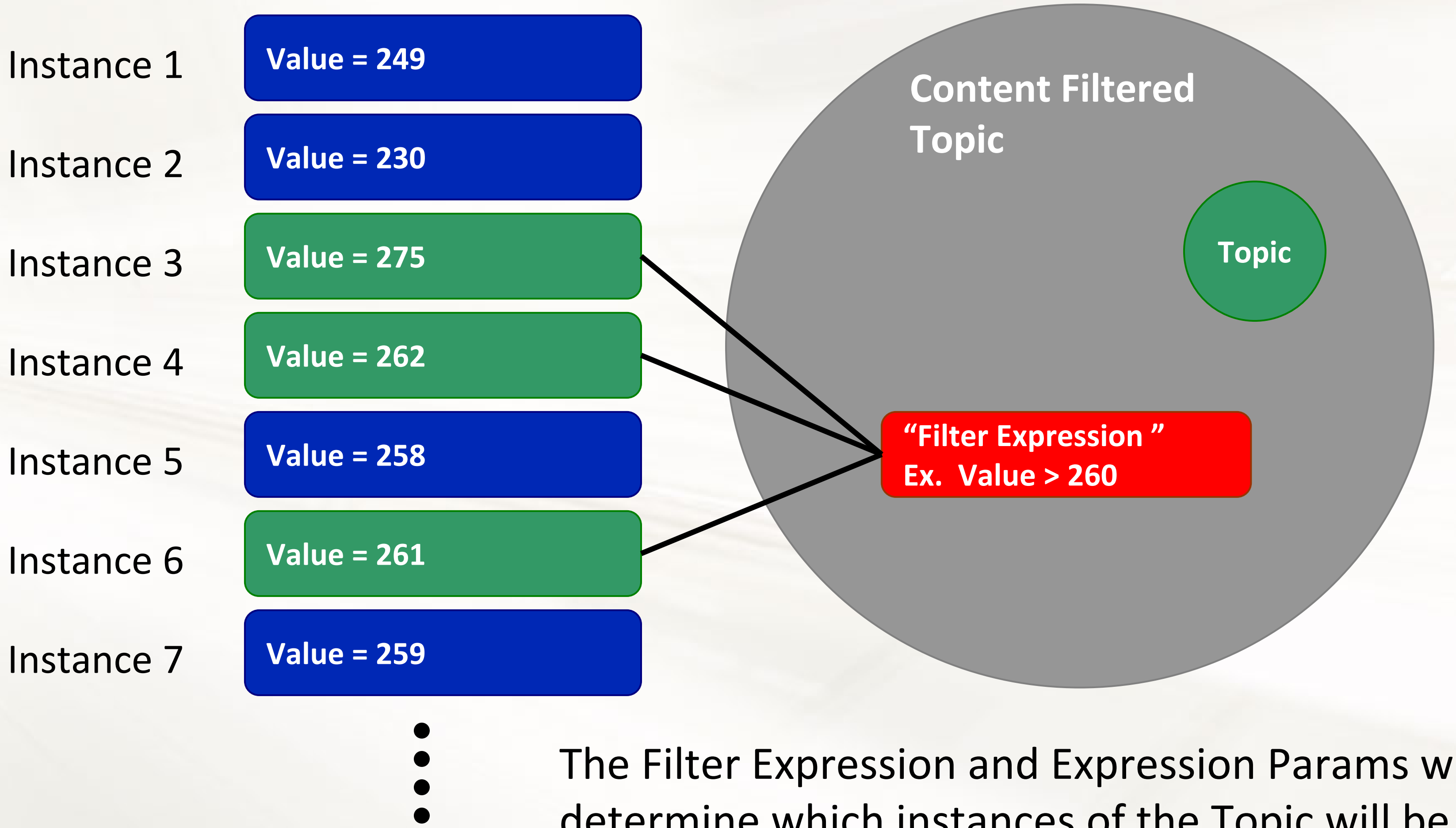
8. Limiting data-rates: QoS: TIME_BASED_FILTER

time_filter_example

"minimum_separation":
Data Reader does not want to receive data faster than the min_separation time



9. Controlling data received by interest set Content-Based Filtering



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

content_filter_example

Using DDS

Best Practices

Best Practices Summary

1. Start by defining a data model, then map the data-model to DDS domains, data types and Topics.
2. Fully define your DDS Types; do not rely on opaque bytes or other custom encapsulations.
3. Isolate subsystems into DDS Domains.
4. Use keyed Topics. For each data type, indicate the fields that uniquely identify the data object.
5. Large teams should create a targeted application platform with system-wide QoS settings.

See http://www.rti.com/docs/DDS_Best_Practices_WP.pdf

Why defining the proper keys for your data types is important

Many advanced features in DDS depend on the use of keys

- History cache.
- Ensuring regular data-object updates.
- Ownership arbitration and failover management.
- Integration with other data-centric technologies (e.g. relational databases)
- Integration with visualization tools (e.g. Excel)
- Smart management of slow consumers and applications that become temporarily disconnected.
- Achieving consistency among observers of the Global Data Space

Find out more…

www.rti.com

community.rti.com

demo.rti.com

www.youtube.com/realtimeinnovations

blogs.rti.com

www.twitter.com/RealTimeInnov

www.facebook.com/RTIsoftware

www.slideshare.net/GerardoPardo

About RTI
Global Leader in DDS

RTI: Global leader in DDS

- Over 70% worldwide embedded messaging middleware market share
- First with…
  - DDS API (2004)
  - RTPS interoperability protocol (2007)
- Active in OMG standardization
  - Board of Directors member
  - Co-chair DDS SIG
  - Chair DDS standard revision committees
- Most mature solution
  - 12+ years of commercial availability
  - Diverse range of industries: defense, finance, medical, industrial control, power generation, communications
  - 350+ commercial customers, 100+ research projects
  - 350,000+ licensed copies

RTI Global Presence

Software development centers

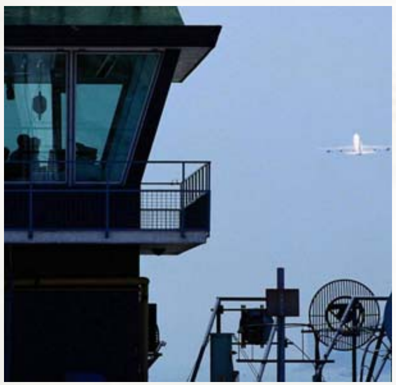
Sales/Consulting offices

(RTI Connext) DDS Application Examples

Full-immersion simulation

National Highway Transportation Safety Authority

Migrated from CORBA, DCOM for performance



Air-Traffic Management

INDRA.

Deployed in

UK, Germany, Spain

Standards, Performance, Scalability

Industrial Control

Schneider Electric

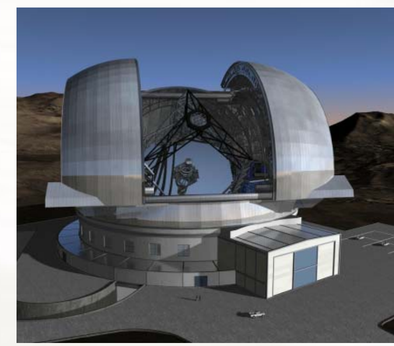
VxWorks-based PLCs

communicate via RTI-DDS

Signal Processing

PLATH GMBH

RTI supports modular programming across product line



Large Telescopes

European Southern Observatory

Performance & Scalability

1000 mirrors, 1sec loop

Radar Systems

AWACS upgrade

Evolvability, Mainteinability, and supportability

RTI Connext DDS Application Examples

Aegis Weapon System

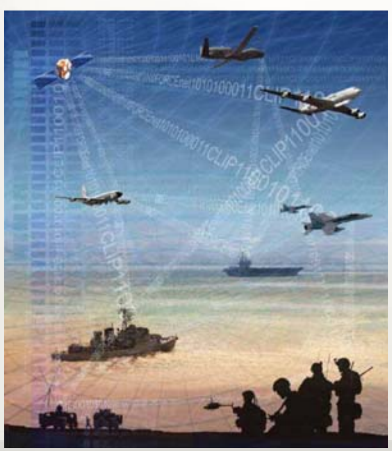
Lockheed Martin

Radar, weapons, displays, C2

B-1B Bomber

Boeing

C2, communications, weapons



Common Link Integration Processing (CLIP)

Northrop Grumman

Standards-compliant interface to legacy and new tactical data links

Air Force, Navy, B-1B and B-52

ScanEagle UAV

Boeing

Sensors, ground station

Advanced Cockpit Ground Control Station

Predator and SkyWarrior UAS

General Atomics

Telemetry data, multiple workstations

RoboScout

Base10

Internal data bus and link to communications center

(RTI Connext) DDS Application Examples

Multi-ship simulator

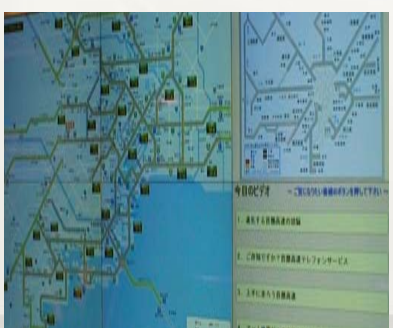
FORCE Technology

Controls, simulation display

Mobile asset tracking

Wi-Tronix

GPS, operational status over wireless links



Highway traffic monitoring

City of Tokyo

Roadway sensors, roadside kiosks, control center

Driver safety

Volkswagen

vision systems, analysis, driver information systems

Medical imaging

NMR and MRI

Sensors, RF generators, user interface, control computers

Automated trading

Automated Trading Desk (ATD, now Citigroup)

Market data feed handlers, pricing engines, algorithmic trading applications

Schneider Programmable Logic Controllers

- Modern factories require the exchange of up-to-the-minute data on manufacturing processes, even with resource-constrained devices
- Challenge to incorporate devices with limited memory or processing power
- RTI with Schneider created a compact real-time publish-subscribe service – resides & executes in under 100 kb!

Industrial Automation

NASA KSC Launch Control

The Constellation program will be the next generation of American manned spacecraft.

RTI delivered 300k instances, at 400k msgs/sec with 5x the required throughput, at 1/5 the needed latency

NASA used RTI's Architecture Study to lower risk.

RTI connects thousands of sensors and actuators

Automotive Safety

The VW Driver Assistance & Integrated Safety system

- Provides steering assistance when swerving to avoid obstacles
- Detects when the lane narrows or passing wide loads
- Helps drivers to safely negotiate bends

RTI Connext integrates diverse legacy buses in the Car

Medical Imaging

“RTI delivered great functionality at a low cost. Using RTI middleware saved us a lot of money, time, and effort compared to our previous in-house developed solution.”

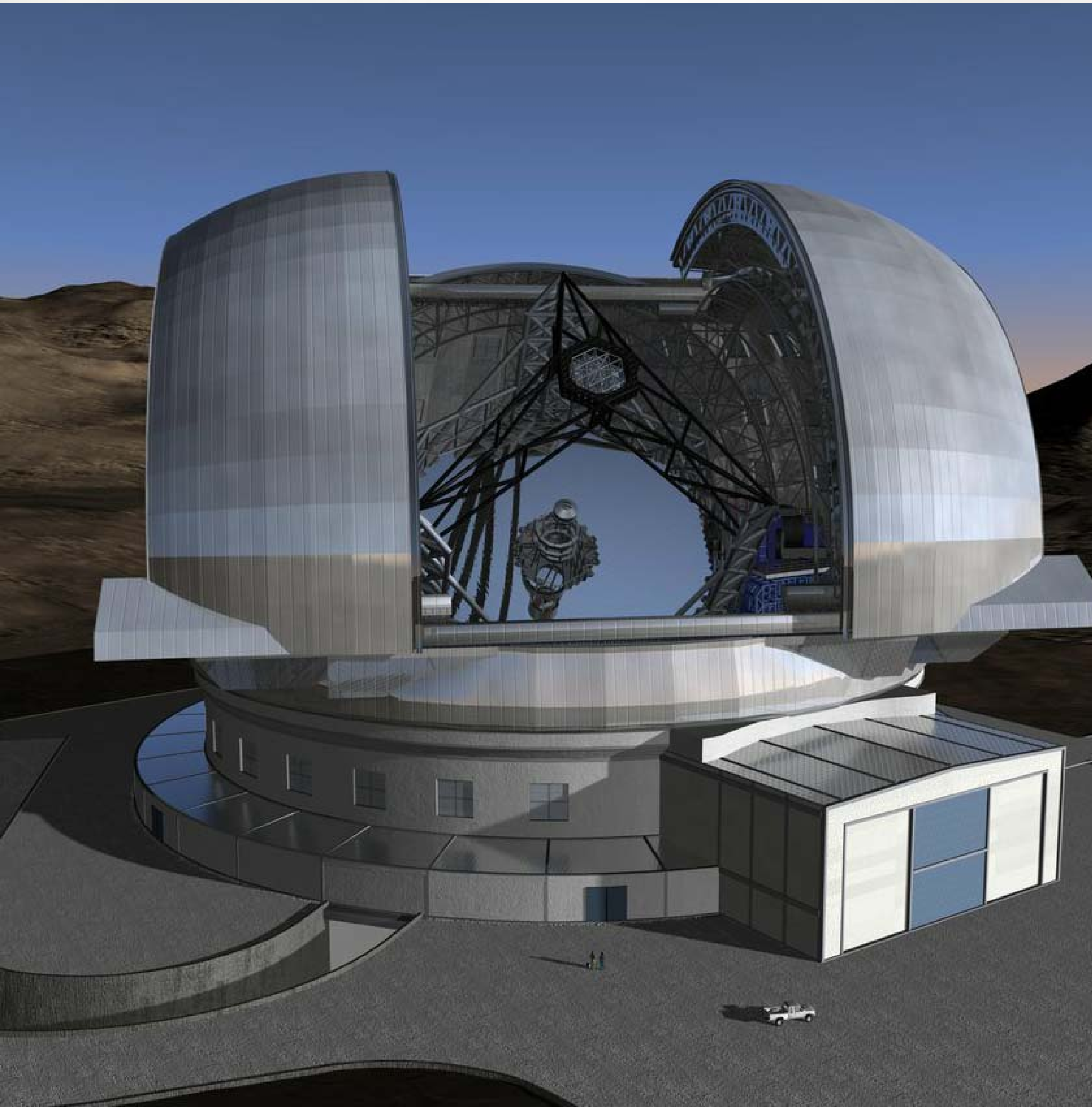
RTI powers Varian’s entire NMR and MRI product lines

A single MRI receiver can saturate a 1Gbit network. An instrument may have 16…

RTI Connext DDS flexible and powerful QoS optimizes network use

RTI Connext handles megabytes of data

European Southern Observatory



ESO's Very Large Telescope array has over 900 mirrors

Each mirror can be separately controlled in position and orientation (6 DOF)

Each second all mirrors are reposition to compensate for atmospheric disturbances

RTI coordinates thousands of servo mirrors and operational paramaters.

Instrumentation

RTI middleware coordinates control and measurement

Lockheed Martin US Navy Aegis Open Architecture Weapon System

Next-generation of the U.S. Navy Aegis Weapon System

- Challenge to share time-critical data across highly distributed system including radar, weapons, displays and controls
- Need to maximize future scalability and flexibility
- RTI provides real-time communication infrastructure. Standards-based & extensible for future system enhancements

Thank you