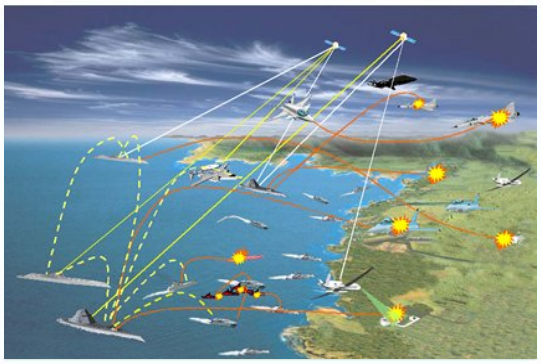


Introduction to DDS

OMG Real-Time Workshop
Washington DC
July 2008

Gerardo Pardo-Castellote, Ph.D.
Co-chair OMG DDS SIG
CTO, Real-Time Innovations
gerardo.pardo@rti.com

Agenda

- Background: Middleware Technologies
- Introduction: DDS Model & Applicability
- Details: DDS in depth

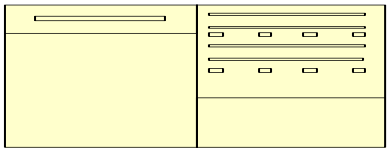
Introduction: Middleware Technologies

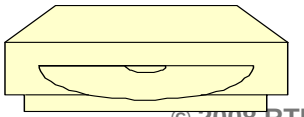
- The concept of network middleware

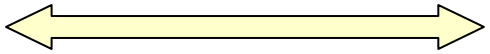
- Communications Model
- Object Model
- Architecture Model
- Protocol

With increased complexity…

End User Application







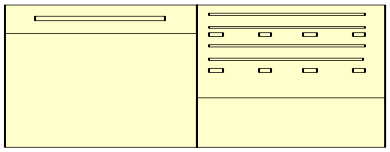
With increased complexity…

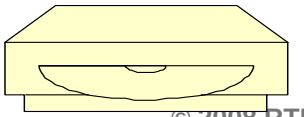
End User Application

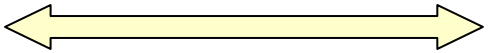
Network Stack | Routing protocols | File System

Kernel | Device Drivers

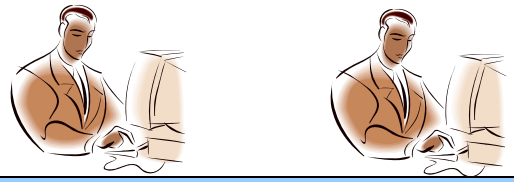
Operating System







With increased complexity…



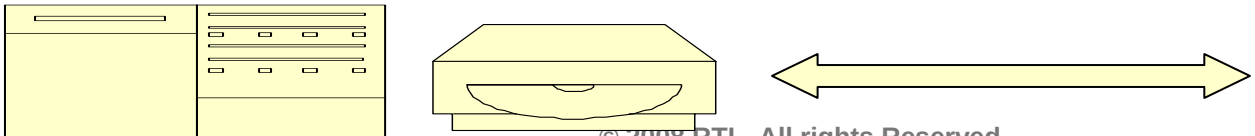
End User Application

HTTP Service | Mail Service | FTP Service | Data base

Network Stack | Routing protocols | File System

Kernel | Device Drivers

Operating System



… middleware becomes necessary


End User Application

Application Server

Message Service

Event Processing Engine

Physics Engine

Graphics Engine

Data base

Middleware

HTTP Service

Mail Service

FTP Service

Network Stack

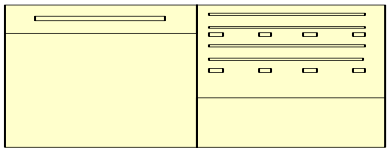
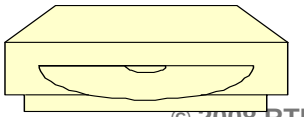
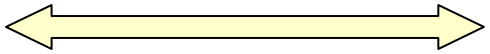
Routing protocols

File System

Kernel

Device Drivers

Operating System & Pre-Packaged Services

What is network middleware?

Middleware =

API and service layer above operating system and below “application” code that abstracts common interaction patterns

Network Middleware =

Most popular class of middleware

Middleware used for developing distributed applications

Distributed Applications =

Those requiring interaction/communication between multiple computers

Network Middleware Examples

- DCE/RPC, DCOM, CORBA, ICE
- TIBCO, 29 West
- JMS, MQ Series, ActiveMQ, SoniqMQ
- DDS, RTI DDS, OpenSplice, tao-dds, GigaSpaces
- New trend:
  - Application Servers (WebSphere, WebLogic, JBOSS)
  - Include network middleware as a component

- NOTE: Middleware “packages” are building blocks, not stand-alone applications like…
  - Skype, gtalk, …
  - BitTorrent, eMule, …

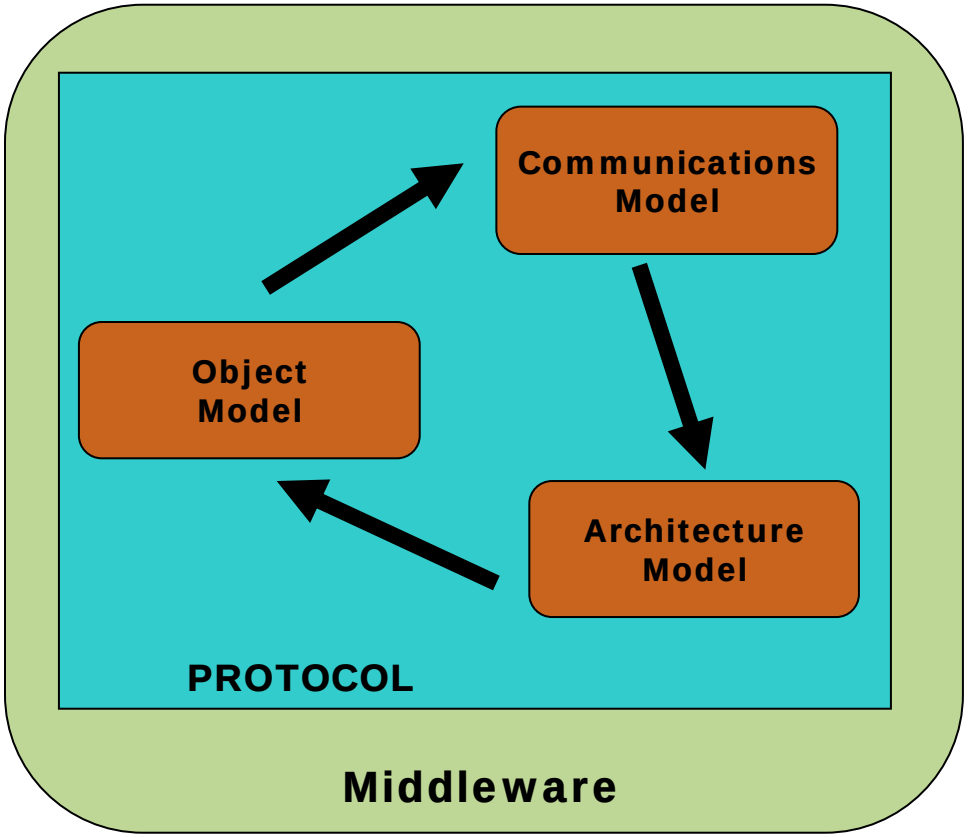
- Why aren’t ‘popular’ consumer applications built on top of middleware?

Historical note: From Telephone to Blogs

- Why so many flavors?
- Parallels evolution of general communication patterns:
  - Started with point-to-point connections
  - Then request-reply services
  - Then Message Queue Services
  - Then Publish-Subscribe Services
  - Then Data-Caching services
- Other examples:
  - FTP, email -> WEB -> Blogs, RSS -> Podcasts

Middleware = Service Model + Protocol

- Service model composed of:
  - Communications model
  - Object Model
  - Architecture model
- All these interact
- The service model and protocol are also coupled



Protocols cannot be compared in isolation!
They must be compared in the context of the service model

Middleware Service Model

Service Model = Communications Model
+ Object Model
+ Architecture Model

- Communications Model:
  - Abstract model of how applications interact:
    - Remote Method Invocation
    - Queue Based, Pub-Sub
    - Data-Distribution
    - Replicated Data
    - Distributed Transactions
- Object Model
  - Middleware entities the application uses to interact with the service:
    - Queues, Publishers, Domains, Caches, Federations, Remote Objects…
- Architecture Model
  - Centralized, Brokered, Peer-to-Peer

Agenda

- The concept of network middleware

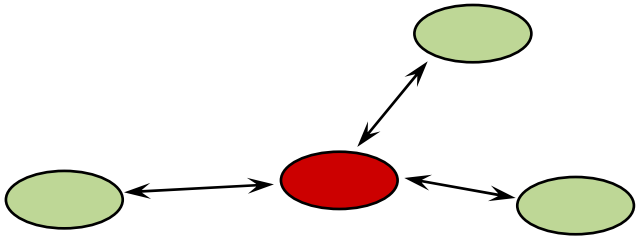
- Communications Model
- Object Model
- Architecture Model
- Protocol

Communications Model

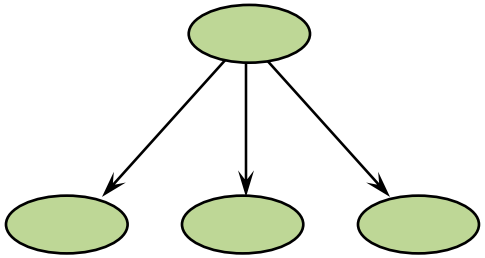
- Abstract model of how applications interact
  - Remote Method Invocation
  - Message-Oriented, Queue Based
  - Pub-Sub Data-Distribution
  - Replicated Data
  - Distributed Transactions

Middleware Communication Models

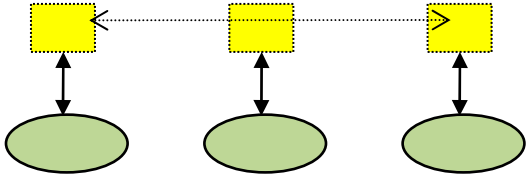
Point-to-Point
Telephone, TCP
Simple, high-bandwidth
Leads to stove-pipe systems



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



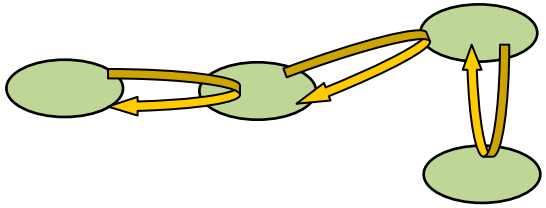
Publish/Subscribe Messaging
Magazines, Newspaper, TV
Excels at *many-to-many communication*
Excels at distributing *time-critical information*

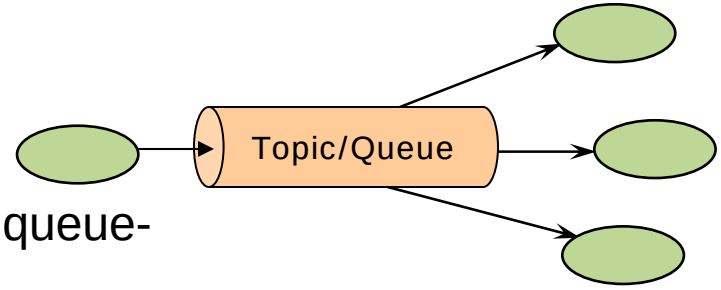


Replicated Data
Libraries, Distributed databases
Excels at data-mining and analysis

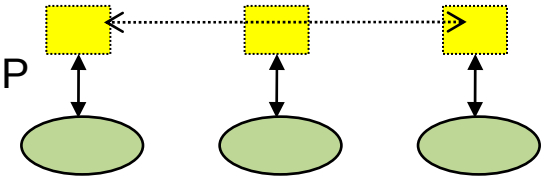
RMI vs Pub-Sub/Messaging/Data-Distribution

- RMI (WebServices, CORBA, DCOM) offer a remote method abstraction
  - Familiar OO programming model
  - Results in a tightly-coupled system
    - Forces synchronous invocations
    - Imposes global object model
    - Limited QoS (appearance of local method call)
    - Lack robustness: cascading points of failure
  - Typically built on top of TCP:
    - impacts scalability and time-determinism
  - Best-suited to smaller, closely-coupled systems



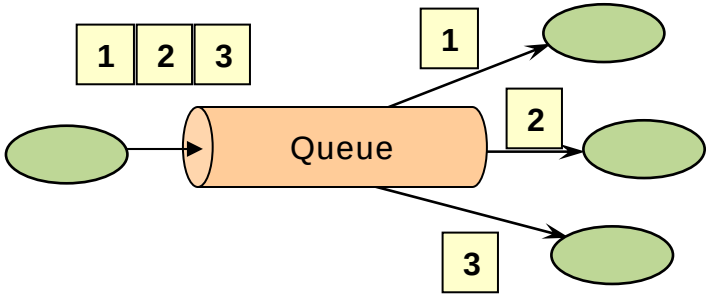


- Pub-Sub (Messaging Data-Distribution) offer a queue-based and/or replicated-data model
  - Subsystems are decoupled in time, space, and synchronization
    - Contracts established by verifying QoS compatibility
  - Supports a variety of transports including multicast UDP
  - Better suited for high-performance and real-time

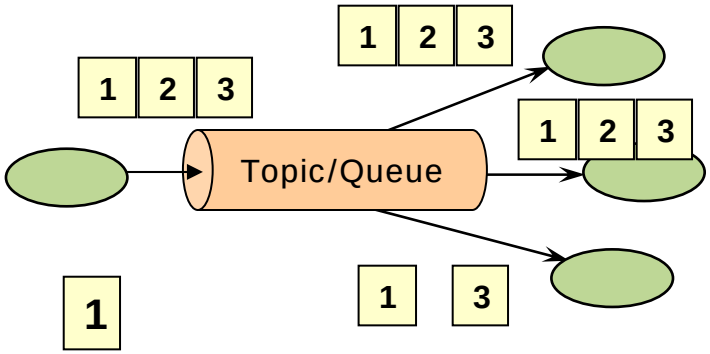


Queue versus Pub-Sub

- Queue
  - Message sent to Queue
  - Multiple readers can read from the queue
  - Each message is delivered to ONLY one reader
    - Readers “affect each other”
  - Apps:
    - Job Scheduling
    - Load Balancing
    - Collaboration

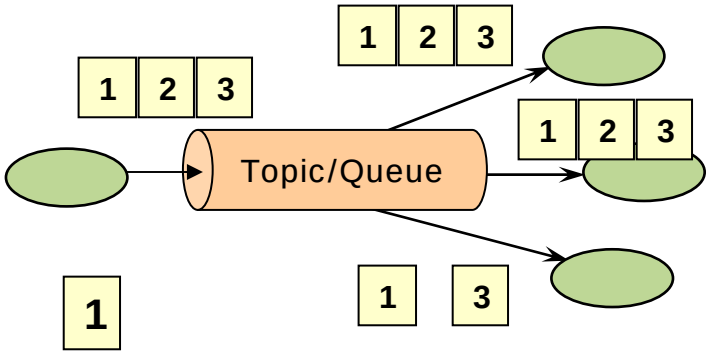


- Pub Sub
  - Message Sent to Topic
  - Multiple readers can subscribe to Topic with or without filters
  - Each message delivered to ALL subscribers that pass filter
    - Readers are decoupled
  - Apps:
    - Notifications
    - Information Distribution

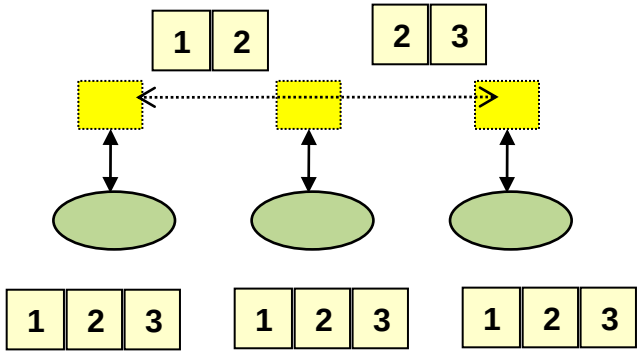


Pub-Sub versus Data-Distribution

- Pub-Sub
  - Only messages no concept of data
  - Each message is interpreted without context
  - Messages must be delivered FIFO or according to some “priority” attribute
  - No Caching of data
  - Simple QoS: filters, durability, lifespan



- Data-Distribution
  - Messages represent update to data-objects
  - Data-Objects identify by a key
  - Middleware maintains state of each object
  - Objects are cached. Applications can read at leisure
  - Smart QoS
    - Ownership
    - History (per key)
    - Deadline
  - Subsumes messaging



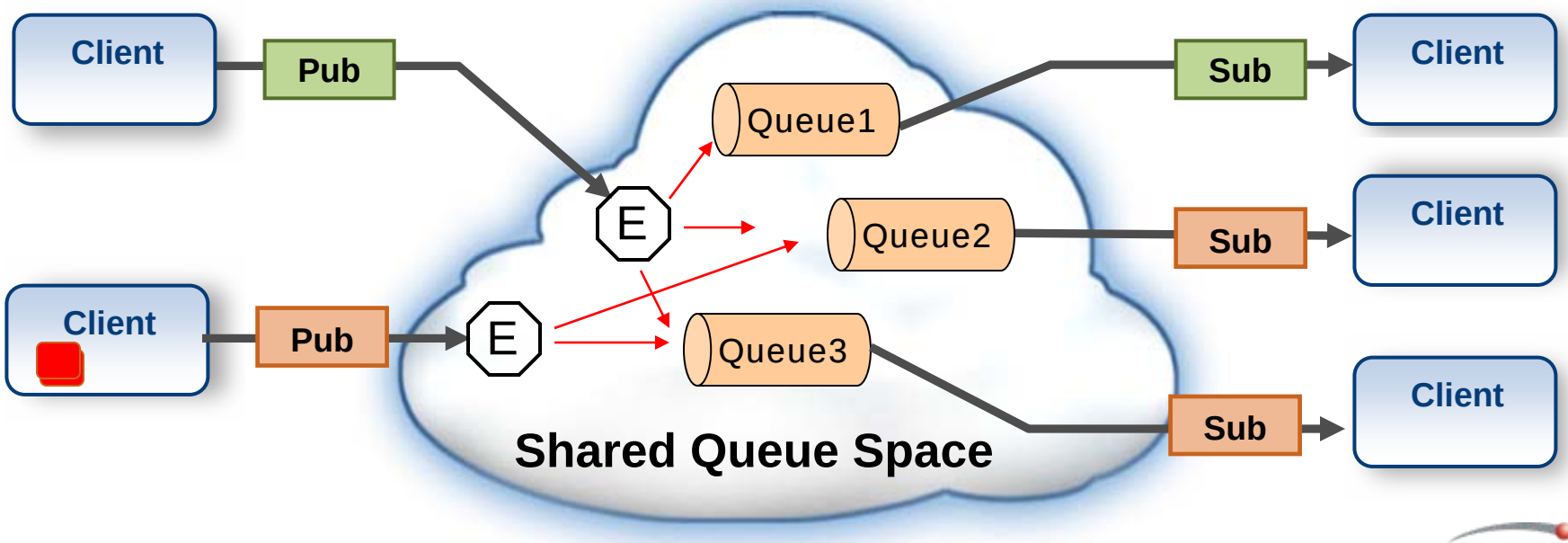
Other (non DDS) Commercial Pub-Sub Models

- Older, but widely deployed
  - TIBCO (RendezVous, EMS)
  - IBM MQSeries
- Limited deployment:
  - CORBA Event Service
  - CORBA Notification Service
- Emerging standards – not really used yet
  - WS-Eventing
  - WS-Notification
- Emerging
  - 29West
  - IBM LLM

JMS/MessageQueue Service Model: Communication Model

Provides a "**Shared Queue Space**" that is accessible to all interested applications.

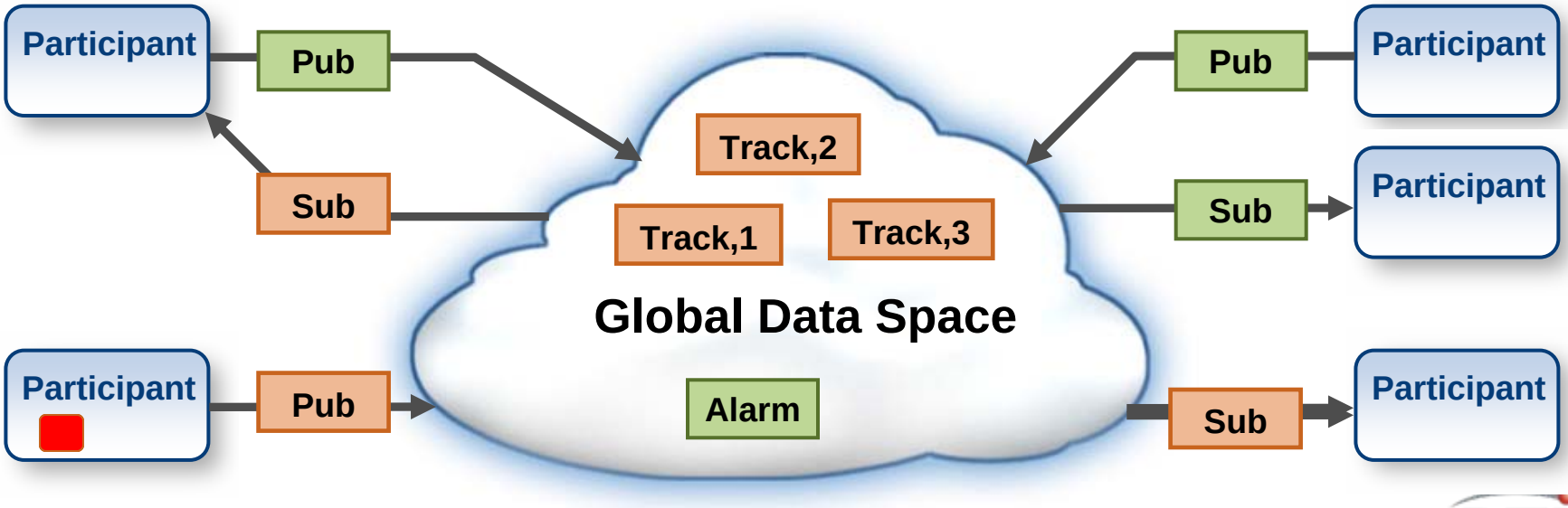
- **Message** are sent to an **Exchange**
- Each message has an associated **Routing Key**
- **Brokers** forward messages to one or more **Queues** based on the **Routing Key**
- Subscriber get messages from **named Queues**
- Only **one subscriber** can get a given message from each **Queue**



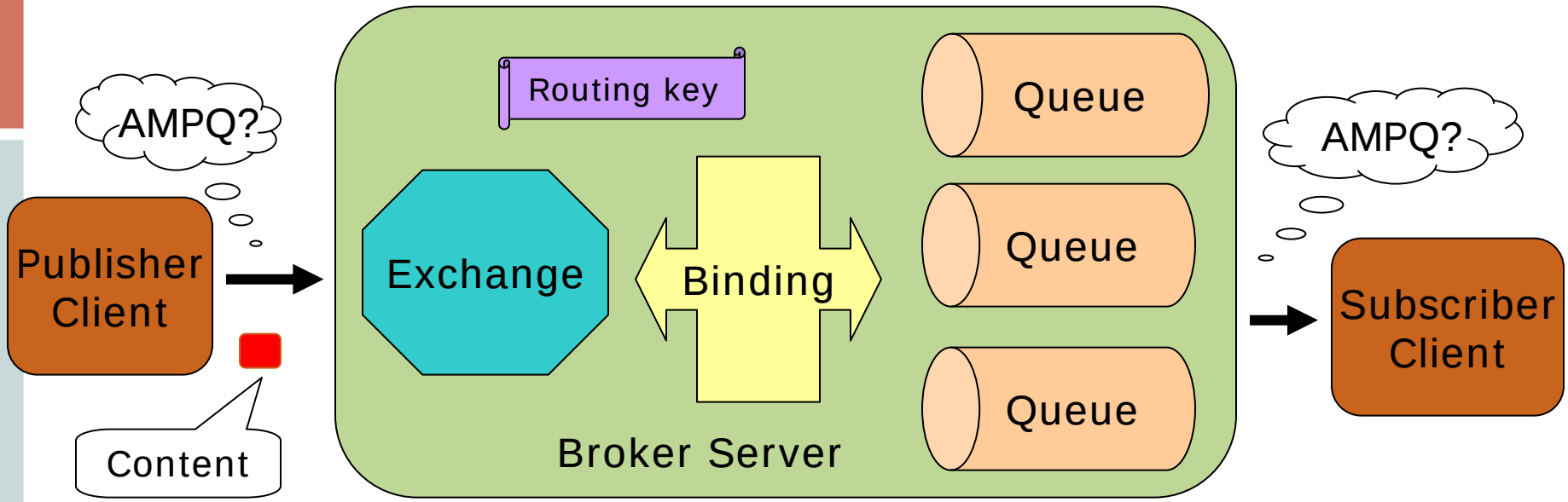
DDS Service Model: Communication Model

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

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

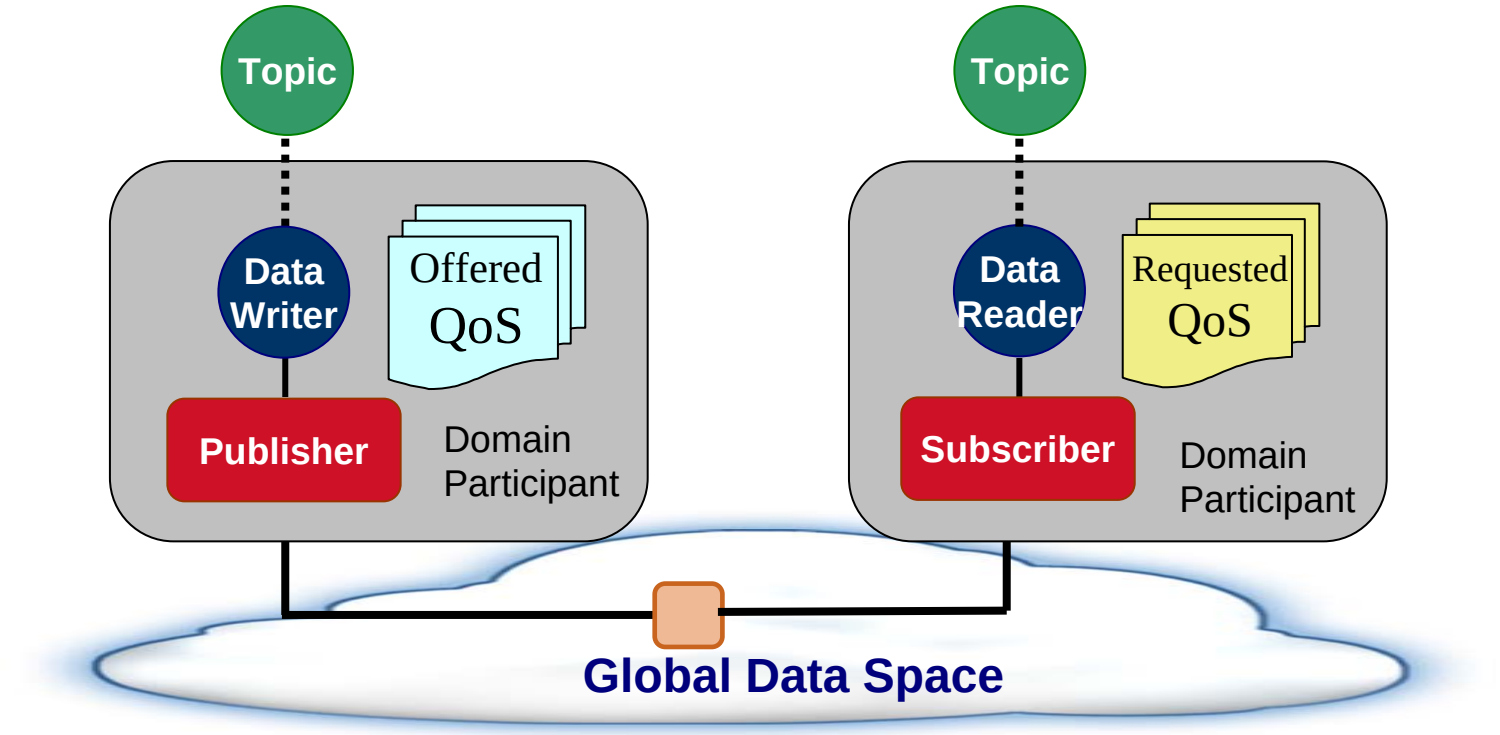


MessageQueue Service Model : Object Model



- **Exchange** – Receives messages and routes to a set of message queues
- **Queue** – Stores messages until they can be processed by the application(s)
- **Binding** – Routes messages between Exchange and Queue. Configured externally to the application
  - Default binding maps routing-key to Queue name
- **Routing Key** – label used by the exchange to route Content to the queues
- **Content** – Encapsulates application data and provides the methods to send receive, acknowledge, etc.

DDS Service Model: Object Model



- **DomainParticipant** – Allows application to join a DDS Domain (Global Data Space)
- **Topic** – A string that addresses a group of objects in the Global Data Space
  - Each Object is identified by a Key (some fields within the object data)
- **Publisher, Subscriber** – Pools resources for DataWriters and DataReaders
- **DataWriter** – Declares intent to publish a Topic and provides type-safe operations to write/send data
- **DataReader** – Declares intent to subscribe to a Topic and provides type-safe operations to read/receive data

Agenda

- The concept of network middleware
- Communications Model
- Object Model
- Architecture Model
- Protocol

Architecture Models

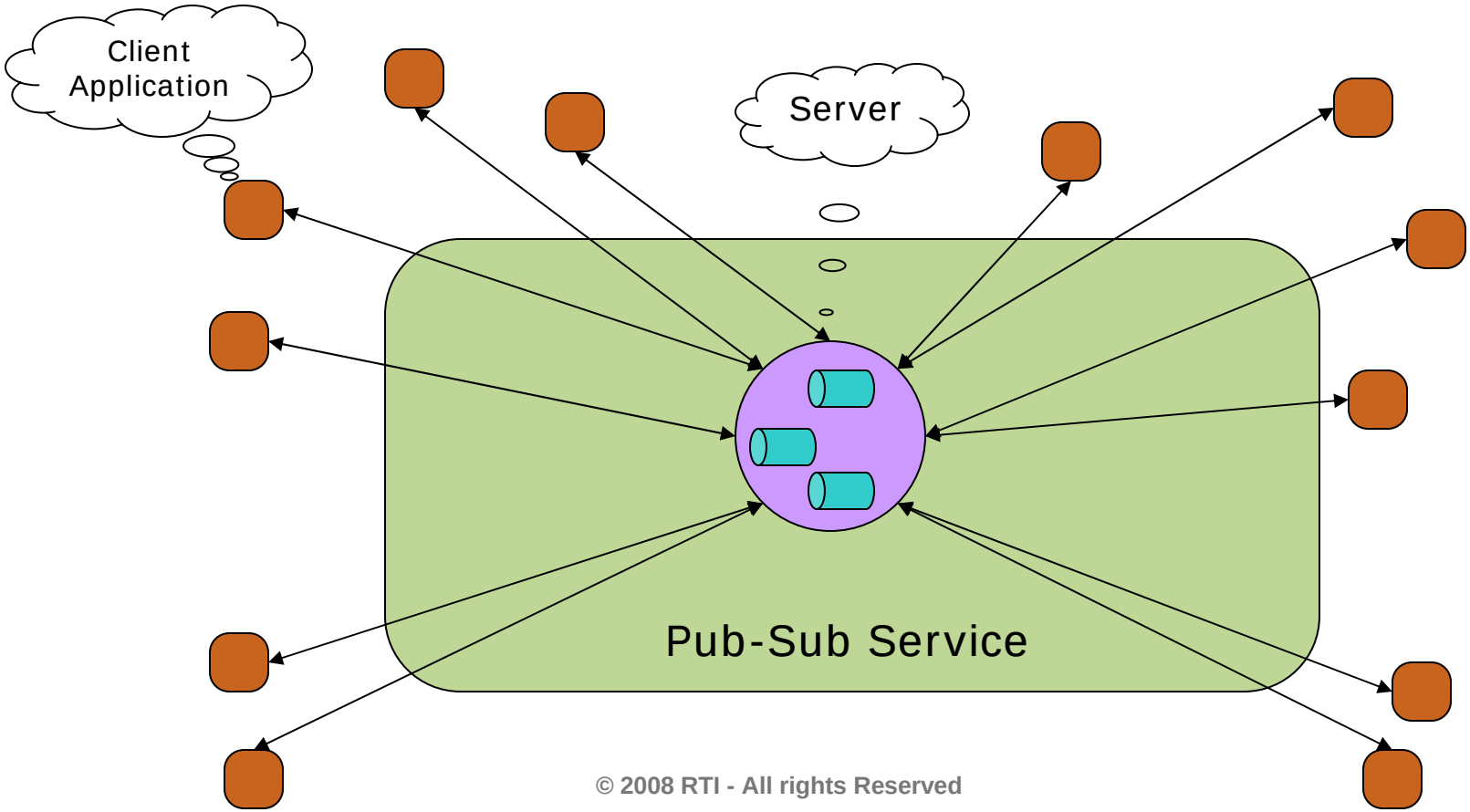
- Brokered
  - Centralized
  - Segmented
  - Federated
- Peer to Peer

Centralized Broker Pub-Sub Service

One central server materializes all middleware entities

All traffic flows via server

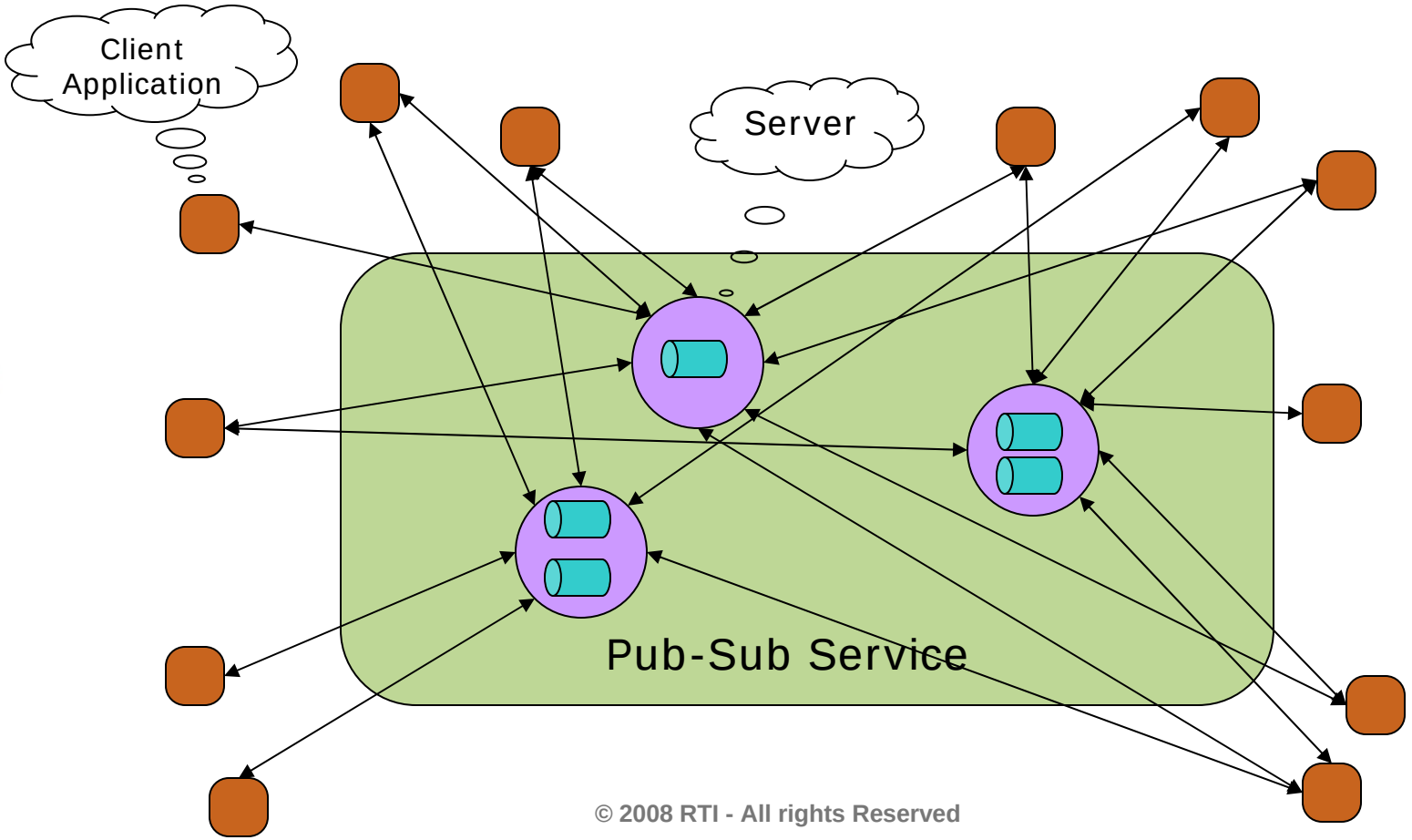
E.g. “naïve” implementations of JMS, CORBA Notification, etc.



Segmented (Grid-based) Pub-Sub Service

Each Queue/Topic Can be placed on a different Server

E.g. Better implementations of JMS, CORBA Notification, etc.

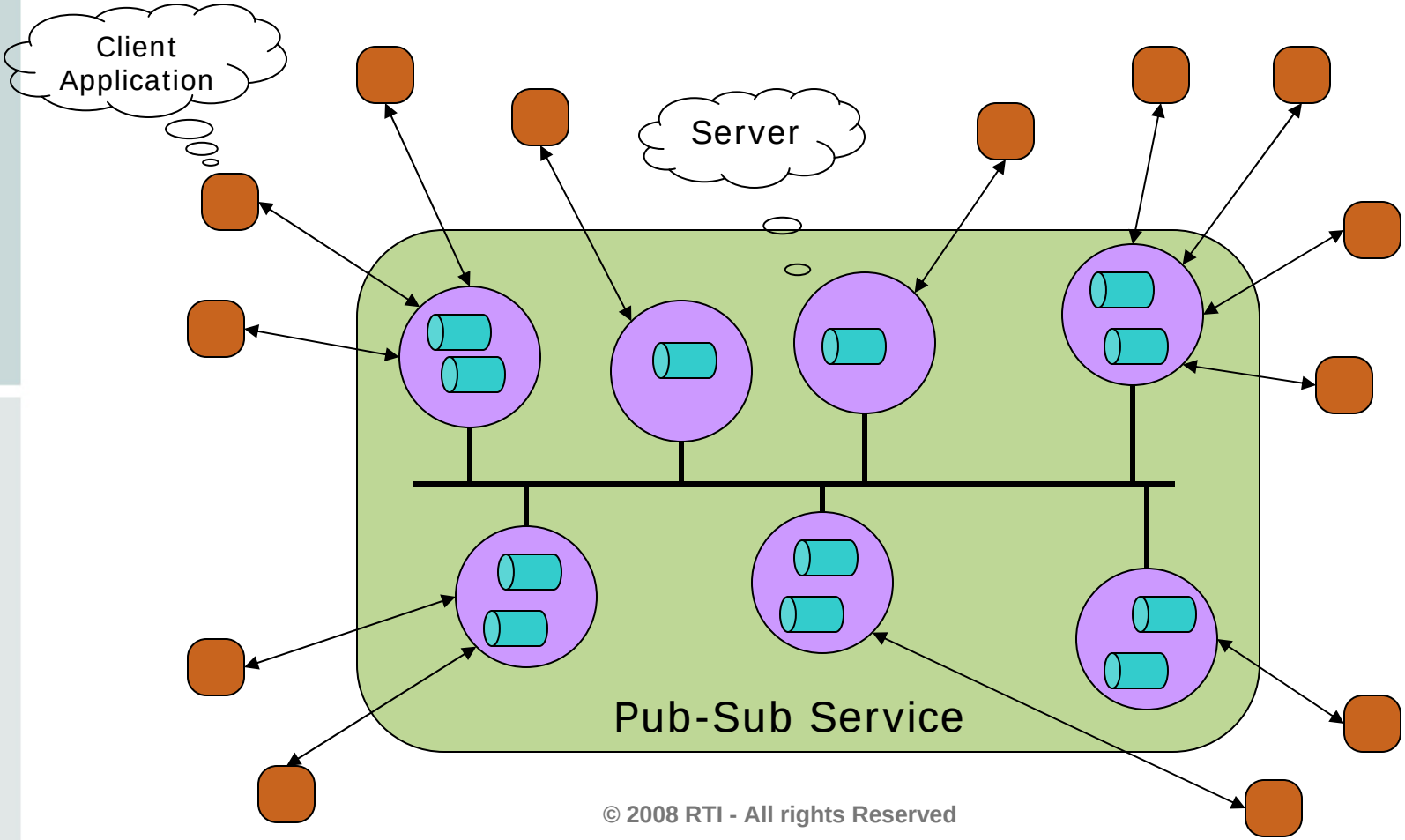


Federated Brokered Pub-Sub Service

App uses messaging or RMI to interact with Service Access points

Pub-Sub Service distributes messages internally between servers

Internally PS-Service can be peer-to-peer, hub-and-spoke, multicast, etc.

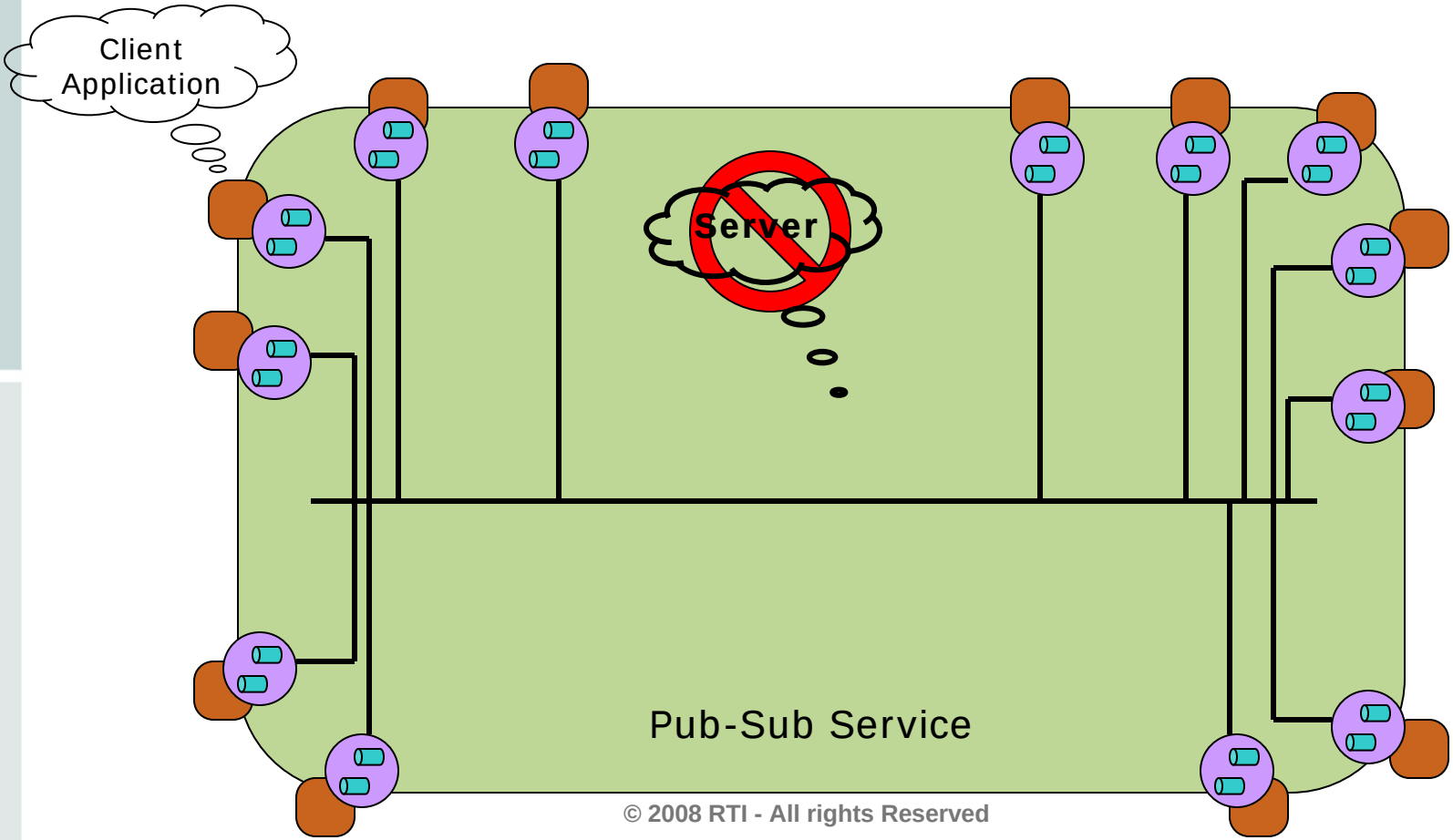


Peer-to-Peer Pub-Sub Service

App links (binds) directly with the Pub-Sub service

Queuing occurs locally on each client

Clients communicate peer-to-peer



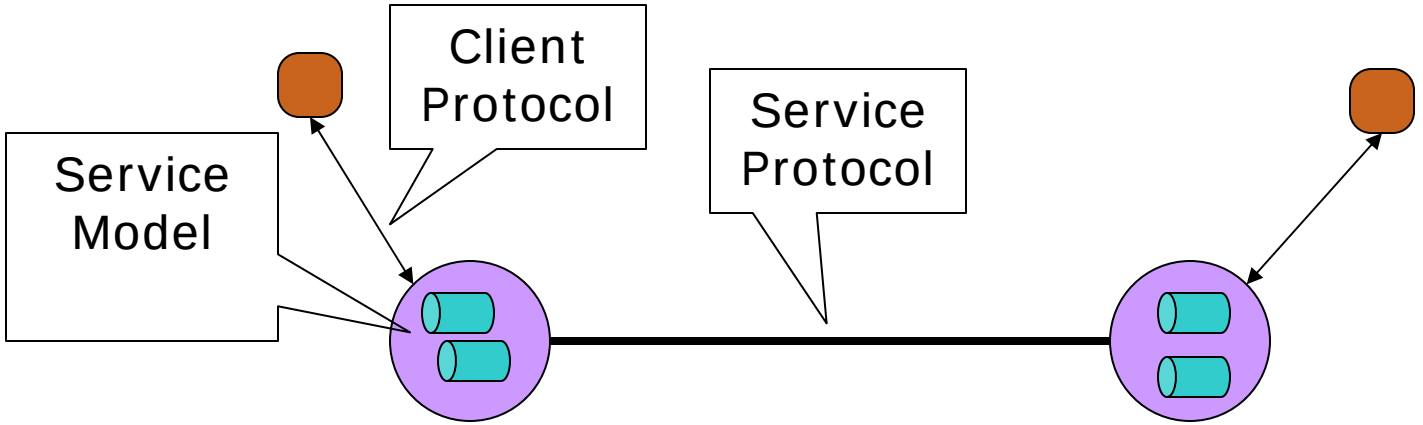
Service Model Architecture Examples

Model	*Examples*
Centralized Brokered	Typical JMS implementations
Segmented (Grid Based)	Better JMS implementations CORBA event & notification service
Federated Brokered	TIBCO RendezVous TIBCO SmartSockets IBM WebSphere MQ (MQSeries) using client connection
Peer-to-Peer Un-brokered	Most DDS implementations: RTI DDS, OpenSplice, Tao-DDS IBM LLM, 29West

Agenda

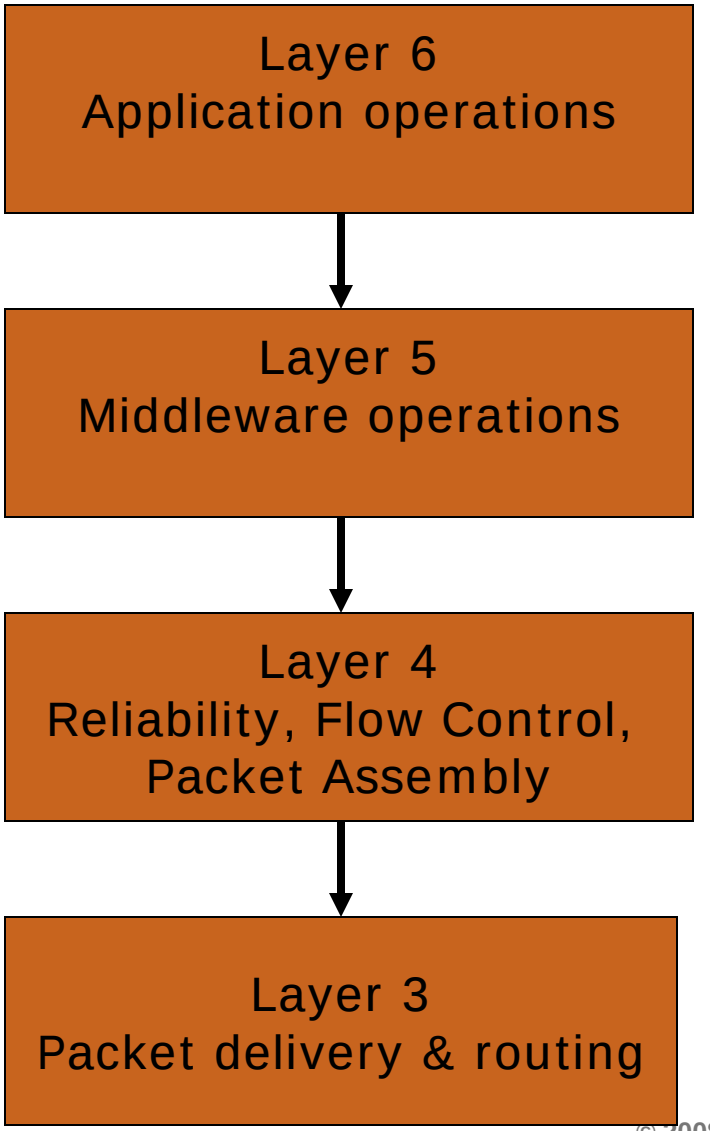
- The concept of network middleware
- Communications Model
- Object Model
- Architecture Model
- Protocol

Service Model and Protocol relationship



- There are interactions between the service model and the Protocol
- A Brokered Service model requires 2 protocols:
  - Client Protocol (used by client applications)
  - Service Protocol (used between the Brokers)
- A fully peer-to-peer middleware requires only one protocol

Protocol layers: Middleware perspective



The existing middleware perspectives are inadequate to model middleware protocols:

The 7-layer OSI model:

- **Has layers with very limited functionality: (Session Presentation)**
- **Considers TCP and UDP both "layer 4 concepts"**

The 5-layer TCP/IP model:

- **Combines all layers above Transport into single "Application Layer"**
- **Considers TCP and UDP equivalent**

Some protocols may expand multiple layers

Protocol layers (Middleware perspective)

Layer 6
Application operations

Invoke Operations with known semantics
Send messages to queues
Pub/Sub specific topics

Layer 5
Middleware operations

Multi-layer protocols

HTTP
GIOP
SOAP
FTP
TLS
DTLS
STUN

A middleware protocol at this Level must operate on top of a Layer4 protocol like TCP

Layer 4
Reliability, Flow Control, Packet Assembly

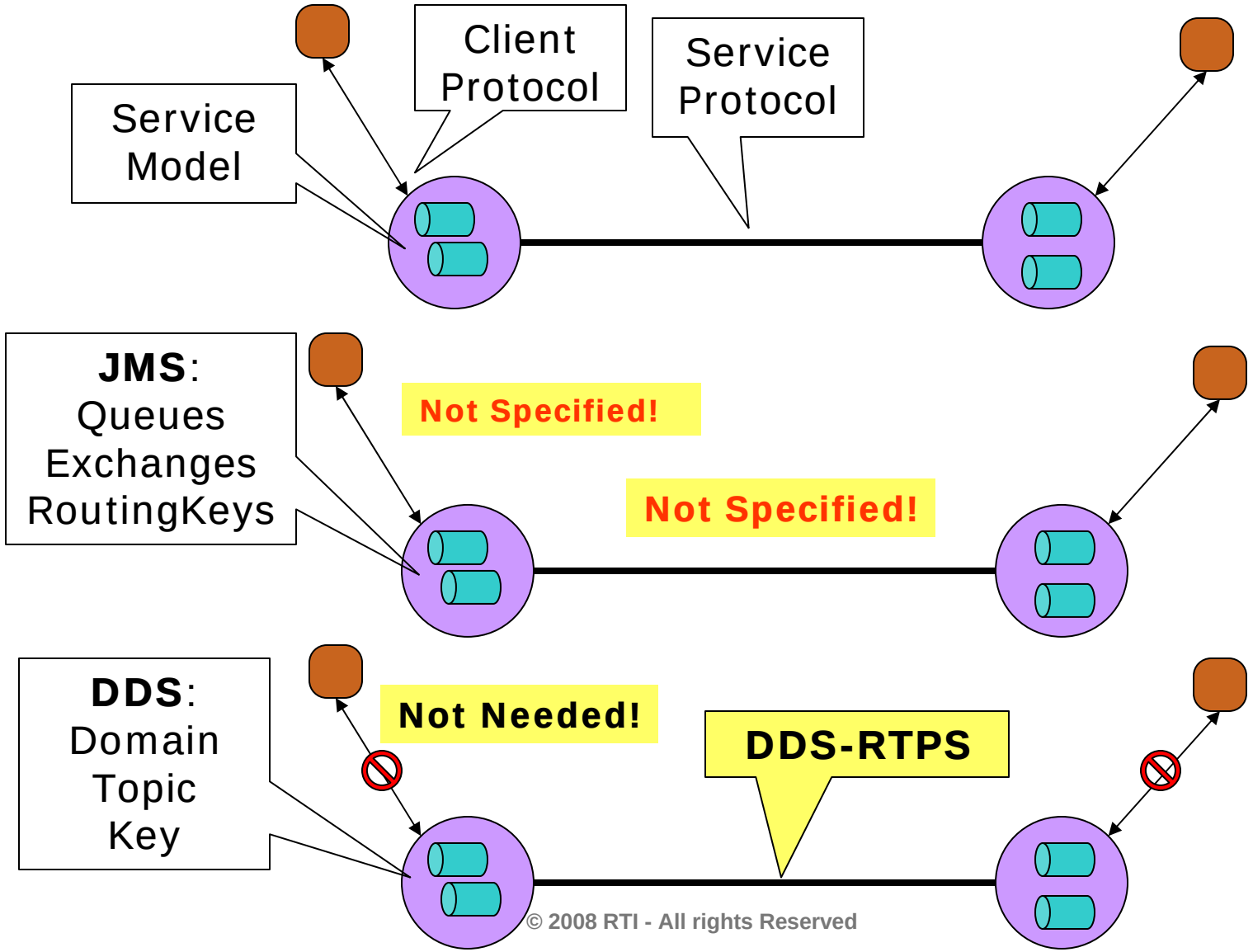
SCTP
TCP
RTP
PGM
DTLS
STUN

A middleware protocol at this Level can operate on top of UDP or Multicast

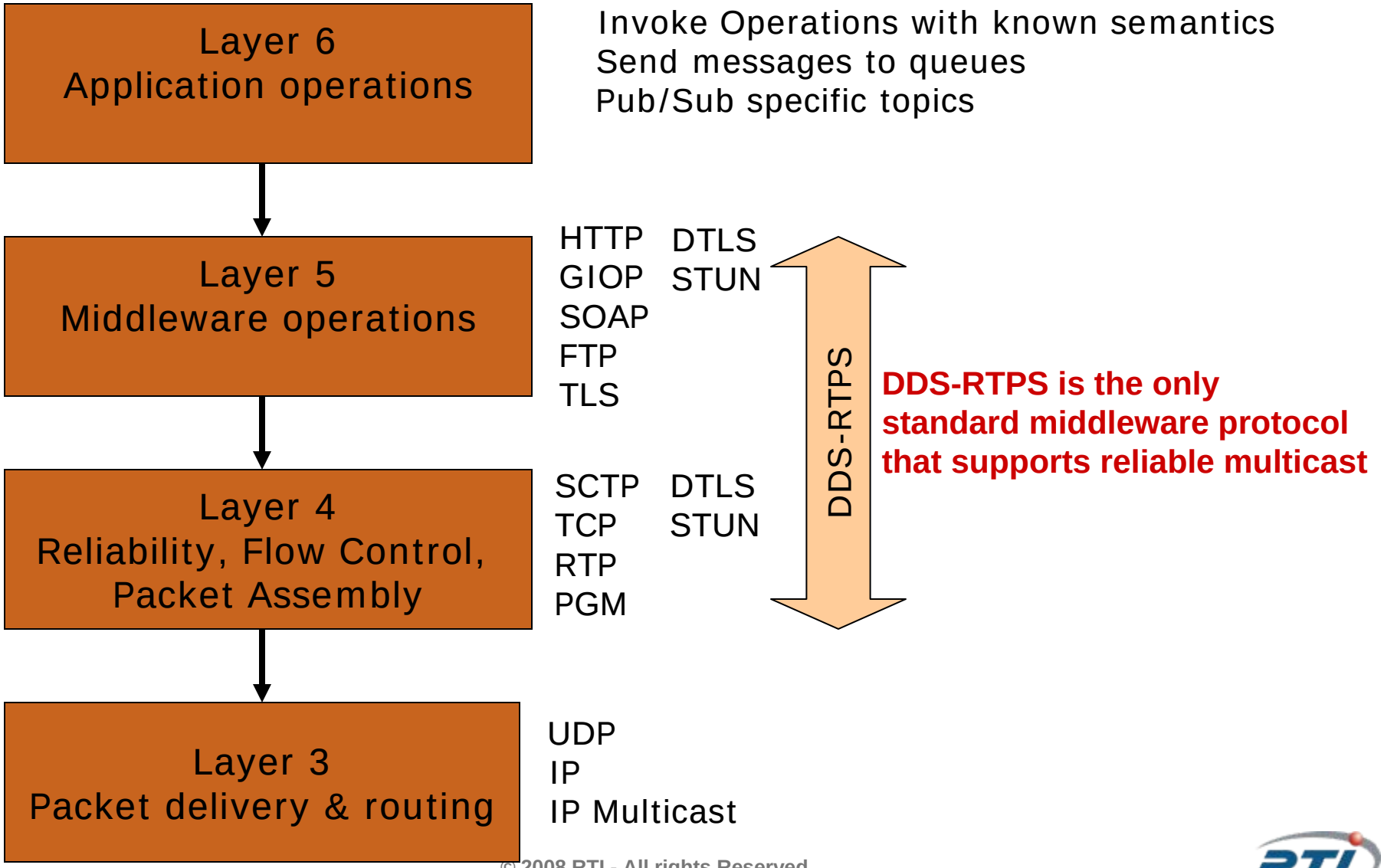
Layer 3
Packet delivery & routing

UDP
IP
IP Multicast

JMS vs DDS-RTPS Protocols: Scope



DDS-RTPS Protocol: Layers



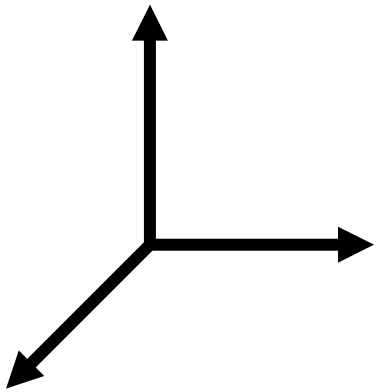
Take away points

- What and why of middleware
  - Components of network middleware
  - Common models:
    - Request Reply
    - Pub Sub
    - Queue Centric
    - Data Centric
- Architectures used by Middleware
  - Brokered, Peer to Peer
- Kinds of Protocols used by Network middleware
  - Layer 3, 4, 5
  - Client to Service. Service to Service

Take Away: Middleware Comparison Dimensions

Service Model:

- Communication Model
- Object Model
- Architecture Model (+ required brokers)

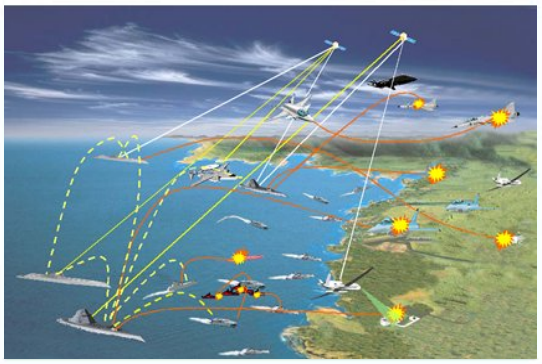


Protocol:

- Layer
- Features
- Transport Assumptions
- Overhead

Impact/Capabilities:

- Features: Reliability, Persistence, QoS, Batching
- Impact:
  - Deployment
  - Performance (protocol overhead, #intermediaries)
  - Robustness (failure conditions)



The DDS Model and its Applicability

Agenda

- History & Applications
- What is Data-Centricity?
- Top Reasons to use DDS

History: DDS the Standard(s)

- Data Distribution Service for Real-Time Systems
  - Joint submission (RTI, THALES, OIS)
  - DDS version 1.0 Adopted December 2004
  - DDS version 1.1 Adopted December 2005
  - DDS version 1.2 Adopted October 2006
- Interoperability wire protocol
  - Joint submission (RTI and PrismTech)
  - DDS-RTPS version 1.2 adopted in July 2006
  - DDS-RTPS version 2.0 adopted in June 2007
  - DDS-RTPS version 2.1 approved in June 2008
- Related Standards
  - Joint submission (Sparx, RTI, PrismTech)
  - UML Profile for DDS adopted June 2008
- Standards under Development
  - DDS for light weight CCM work in progress
  - Extensible and Dynamic Topic Types for DDS
  - Native Language C++ API for DDS

OMG Data Distribution Portal

You may discuss this page here.

The **Data-Distribution Service for Real-Time Systems (DDS)** is a recently-adopted OMG standard.

DDS is the first open international middleware standard directly addressing *publish-subscribe* communications for *real-time and embedded systems*. DDS introduces a virtual *Global Data Space* where applications can share information by simply reading and writing data-objects addressed by means of an application-defined name (**Topic**) and a **key**. DDS features fine and extensive control of *QoS* parameters, including *reliability*, *bandwidth*, *delivery deadlines*, and *resource limits*. DDS also supports the construction of local object models on top of the *Global Data Space*.

The DDS portal is maintained by the **OMG Data Distribution SIG (DDSIG)**. For the activities of the DDSIG and other events of interest to the community, please visit the EventCalendar and the DDSIG page. The portal uses a Wiki to manage the content. Before making edits please visit the PortalMaintenancePage and read the PortalUsagePolicies. You may also want to look at the WikiCourse and at HelpContents. The WikiSandBox is a good place to experiment with editing.

Learning about DDS

A general introduction to DDS can be found in DataDistributionIntro. Howtos, patterns of use of DDS and example code can be found in DataDistributionExamples.

The TutorialsPage contains presentations on DDS. More material can also be found in the WhitepapersPage and the OtherMaterialPage. The TrainingPage lists organizations that hold regular training on DDS or can prepare it on demand.

The InformationDays contains use-case and vendor presentations OMG DDS information days. These events started in 2006 and are still on-going.

The current DDS specification is version 1.2 Older and related specifications can be found in the SpecificationsPage.

The current DDS Interoperability Wire Protocol specification is version 2.1 Older and related specifications can be found in the SpecificationsPage.

Visit the VendorsPage or the ProjectsPage learn about vendors, products and projects that support or use DDS.

News and Events

2008.6.26 -- **UML Profile for DDS Specification**. OMG Technical Meeting held in Ottawa, Canada. The OMG has recommended for adoption the *UML Profile for DDS Specification*. The specification defines how UML tools can be used to model DDS systems and automatically generate supporting code. For more information visit the SpecificationsPage.

2008.6.26 -- **Extensible and Dynamic Topic Types for DDS RFP**. OMG Technical Meeting held in Ottawa, Canada. The OMG has issued and RFP with the dual goals of adding Type Extensibility to DDS Topics as well as introducing a dynamic API to allow reading and writing types for which there was no compile-time knowledge. For more information visit SpecificationsInProgress.

2008.5.16 -- **Hands-On DDS Workshops**. RTI has scheduled several hands-on DDS training workshops in 2008: May 20-21, August 20-21 and October 22-23. Whether you are evaluating, planning to use, or already using the DDS standard, these two-day workshops provide an excellent opportunity to learn about the capabilities of the standard and how to apply them to your application. For more information and to register visit http://www.rti.com/services/workshops.html.

2008.5.16 -- **DDS Demonstration Application**. RTI has developed a demonstration application that illustrates DDS concepts such as publish-subscribe, real-time QoS, and data-centric design. Download the demo at https://www.rti.com/mk/shapes_demo.html.

2008.3.14 -- **Native C++ Language DDS API RFP**. OMG Technical Meeting held in Washington DC. The OMG has issued and RFP with the objective of defining a new C++ API to DDS that takes advantage of the language features present in the ISO C++ Standard. For more information visit SpecificationsInProgress.

Navigation
- Data Distribution Portal (HOME)
- Data Distribution Intro
- Data Distribution SIG
- DDS User's Forum
- Event Calendar
- Specifications
- Tutorials
- Whitepapers
- Other Materials
- FAQs
- Vendors
- Portal Index
- Portal Search
- Portal Maintenance
- Portal Help
- Recent Changes
- SystemInfoPage

Search
Search
Titles Text

User
- GerardoPardo
- Preferences
- Logout

Recently viewed pages
- DataDistributionSIG
- DataDistributionIntro
- SpecificationsPage
- SpecificationsInProgress
- OMG Data Di...tion Portal

DDS Adoption – Aerospace & Defense

Boeing
AWAKS

E2C Hawkeye

Northtrop?
Qinetiq?
SAAB?

Boeing
Future Combat
Systems

Raytheon
SSDS

Lockheed
AEGIS

Insitu
Unmanned
Air Vehicles

DDS Adoption – Transportation, Industrial

WiTronix
Train and vehicle Tracking

Schneider Electric
Industrial Automation

Tokyo Japan
Traffic Control

Kuka
Robotics

EU and US
Air Traffic Management

Varian
Medical Instruments

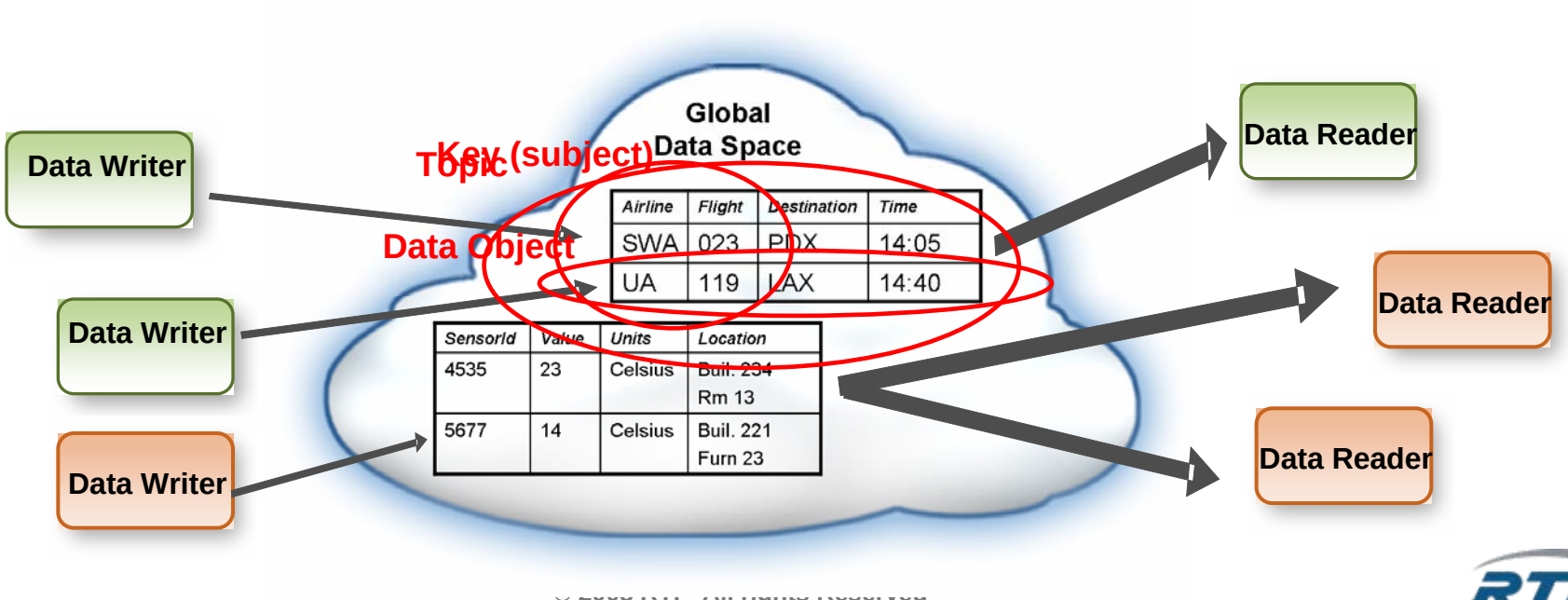
Main differences of DDS vs other Pub-Sub

- Flexibility and Power of the data-centric model
- Performance & Scalability
- Rich set of built-in services
- Interoperability across platforms and Languages
- Natural integration with SOA building-blocks

#1 RTI Data-Centric Model

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

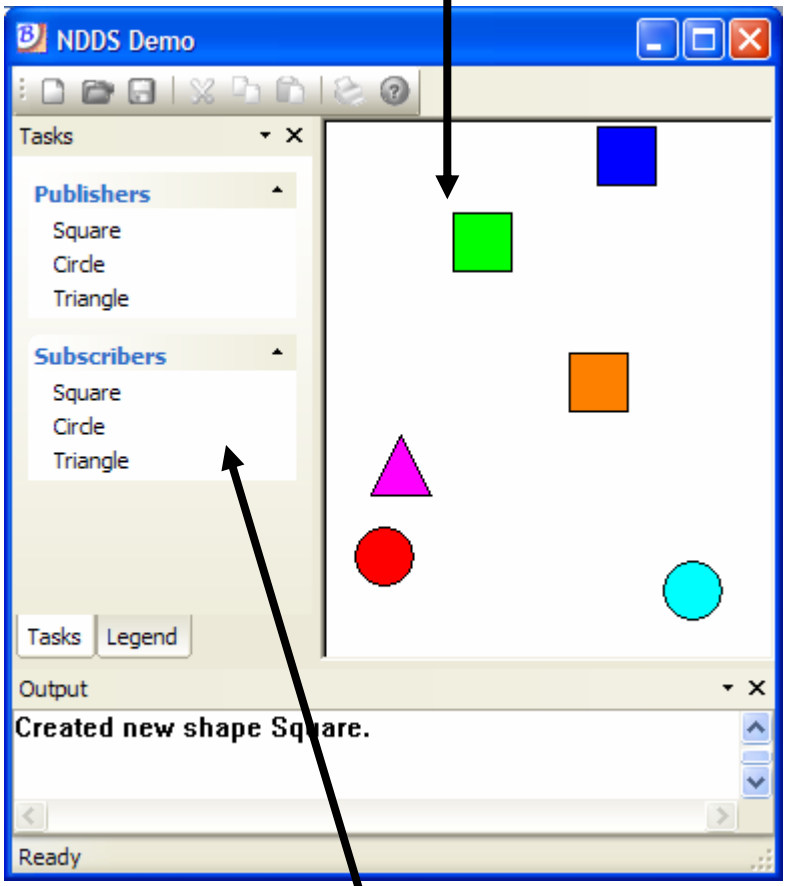
- Data objects addressed by **DomainId, Topic** and **Key**
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a "x.y.z …" formatted string



Demo: Concepts

Start demo

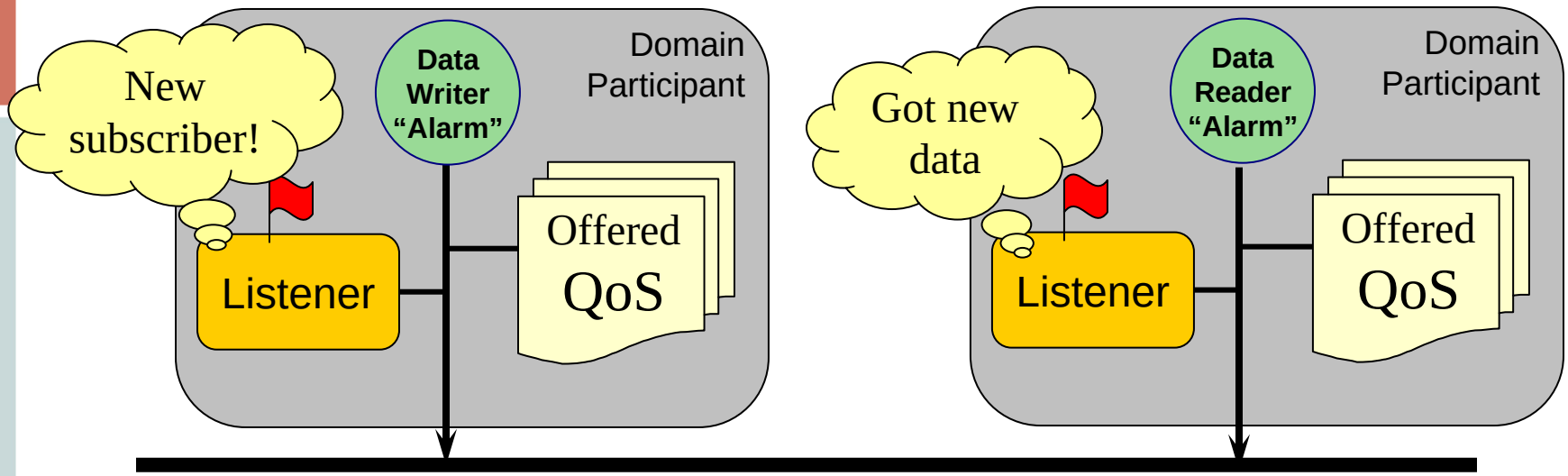
Display Area:
Shows state of objects



Control Area:
Allows selection of objects and QoS

- Topics
  - Square, Circle, Triangle
  - Attributes
- Data types (schemas)
  - Shape (color, x, y, size)
    - Color is instance Key
  - Attributes
    - Shape & color used for key
- QoS
  - Deadline, Liveliness
  - Reliability, Durability
  - History, Partition
  - Ownership

DDS communications model



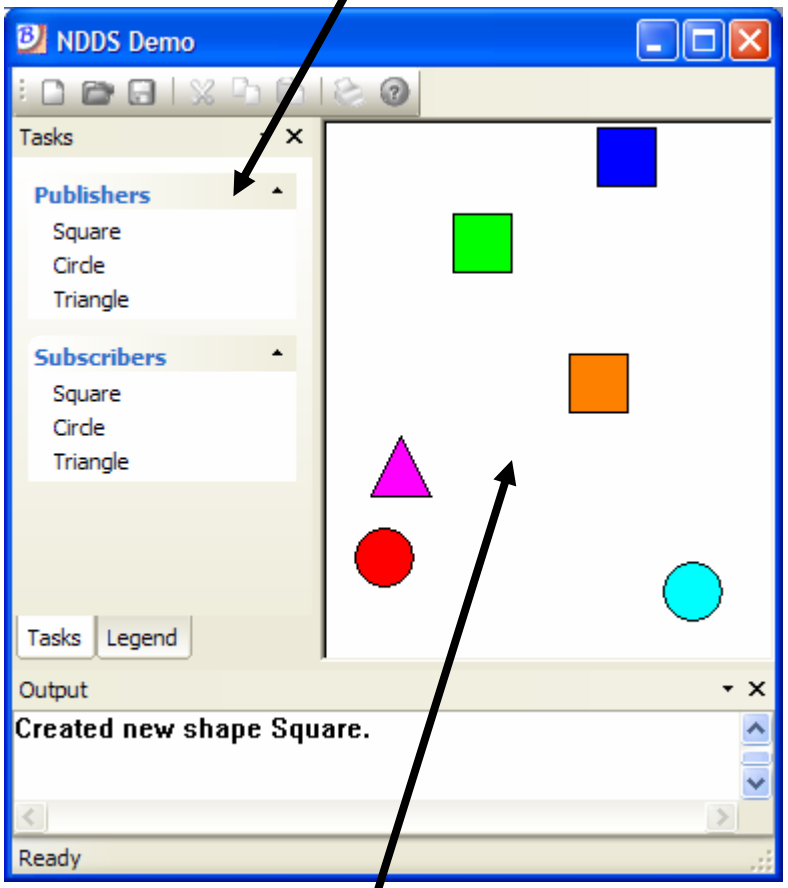
- **Participants** scope the global data space (domain)
- **Topics** define the data-objects (collections of subjects)
- **Writers** publish data on Topics
- **Readers** subscribe to data on Topics
- **QoS Policies** are used configure the system
- **Listeners** are used to notify the application of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

QoS: Quality of Service

QoS Policy	QoS Policy
DURABILITY	USER DATA
HISTORY (per subject)	TOPIC DATA
READER DATA LIFECYCLE	GROUP DATA
WRITER DATA LIFECYCLE	PARTITION
LIFESPAN	PRESENTATION
ENTITY FACTORY	DESTINATION ORDER
RESOURCE LIMITS	OWNERSHIP
RELIABILITY	OWNERSHIP STRENGTH
TIME BASED FILTER	LIVELINESS
DEADLINE	LATENCY BUDGET
CONTENT FILTERS	TRANSPORT PRIORITY

Demo: Quality of Service (QoS)

Writers and readers state
Their needs



RTI DDS delivers

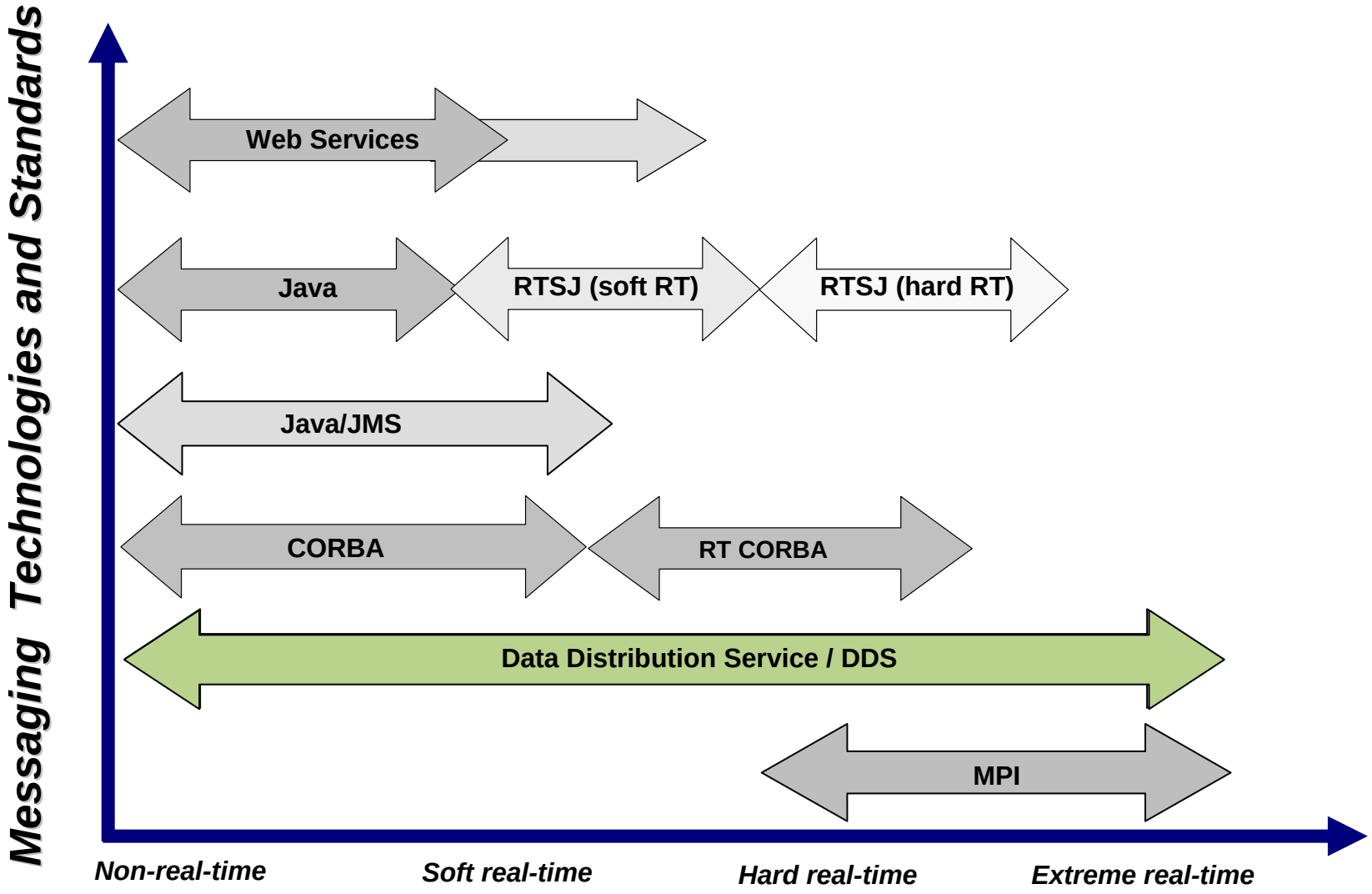
Start demo

- Topics
  - Square, Circle, Triangle
  - Attributes
- Data types (schemas)
  - Shape (color, x, y, size)
    - Color is instance Key
  - Attributes
    - Shape & color used for key
- QoS
  - Deadline, Liveliness
  - Reliability, Durability
  - History, Partition
  - Ownership

#2 Performance & Scalability

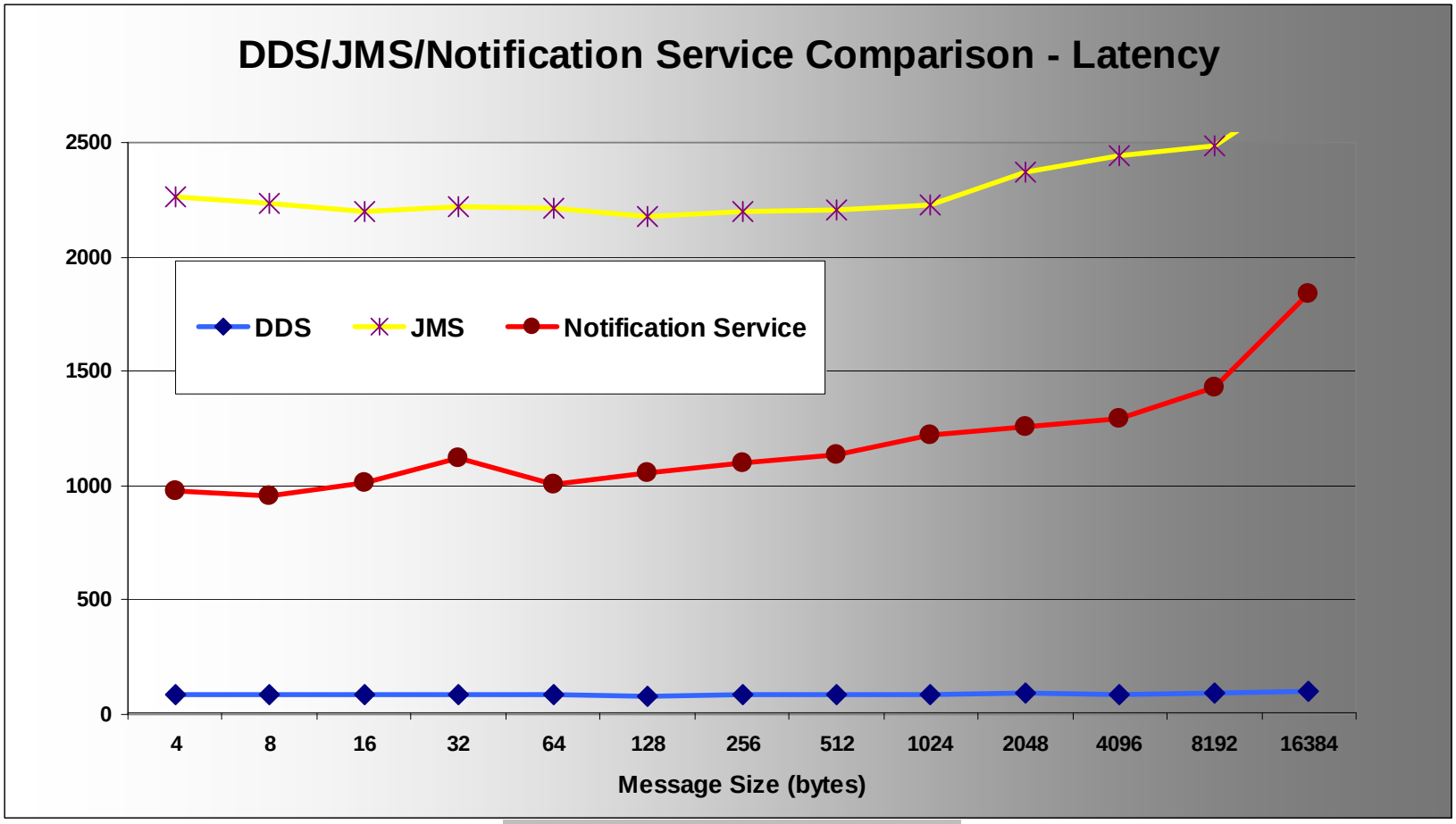
- DDS was designed to support high performance
- RTI DDS was developed to maximize performance and minimize jitter
- Advanced techniques employed:
  - Pre-allocation of memory
    - Never allocate/free memory in the critical path
  - Use dedicated threads per receive port
    - Minimize thread switching
    - Avoid expensing operating system calls (e.g. select())
  - Maximize concurrency
    - Carefully design critical sections
    - Patented concurrent mutex-free thread-safe data structures
  - Employ high-performance data-access APIs
    - Read data by array (no additional copies)
    - Scatter/gather APIs to access transport.
    - Buffer loaning for zero copy access

Data-Distribution and Real-Time



Adapted from NSWC-DD OA Documentation

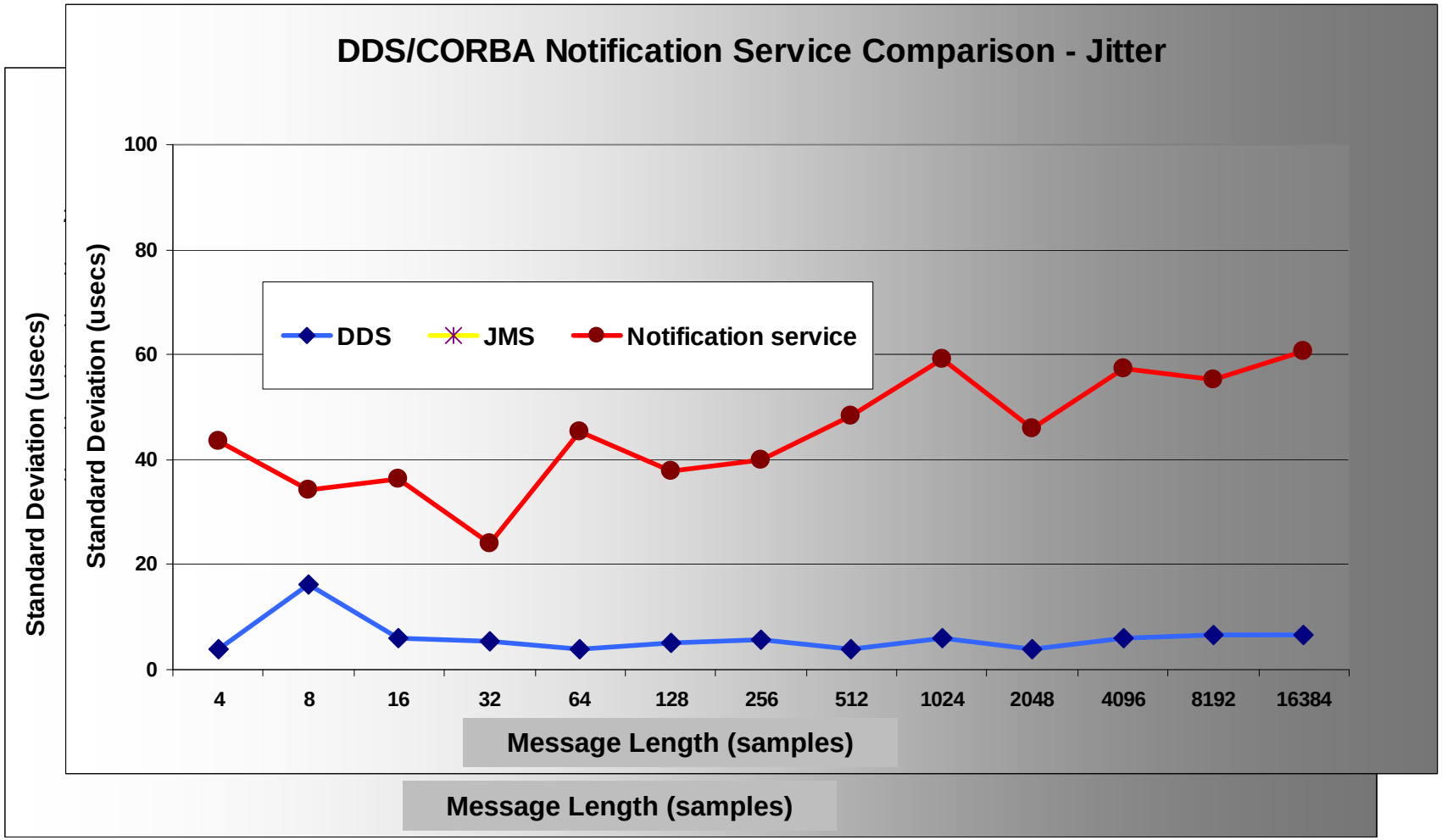
Latency – (Linear Scale)



Message Length (samples)

Adapted from Vanderbilt presentation at July 2006 OMG Workshop on RT Systems

Jitter – (Linear Scale)

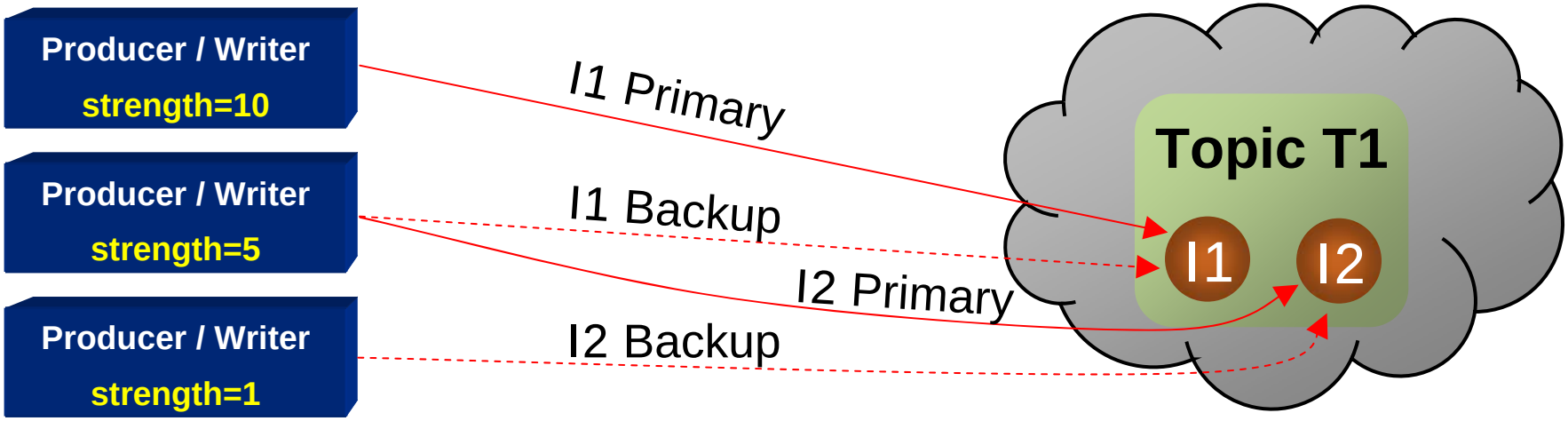


Source: Vanderbilt presentation at July 2006 OMG Workshop on RT Systems

#3 Powerful Services

- Redundancy & Failover
- Persistent Data
- Last value cache
- Historical cache
- Recording service

Ownership and High Availability

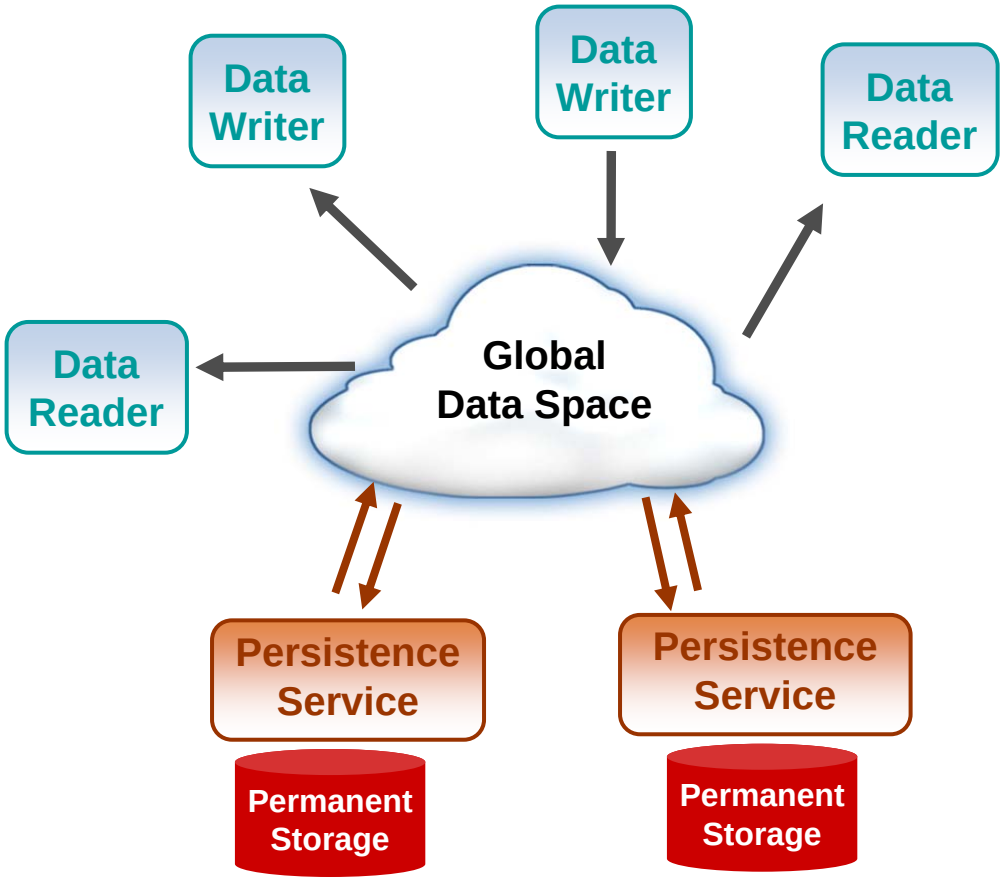


- Owner determined per subject
- Only extant writer with highest strength can publish a subject (or topic for non-keyed topics)
- Automatic failover when highest strength writer:
  - Loses liveliness
  - Misses a deadline
  - Stops writing the subject

- Shared Ownership allows any writer to update the subject

Data Persistence

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



Last value cache

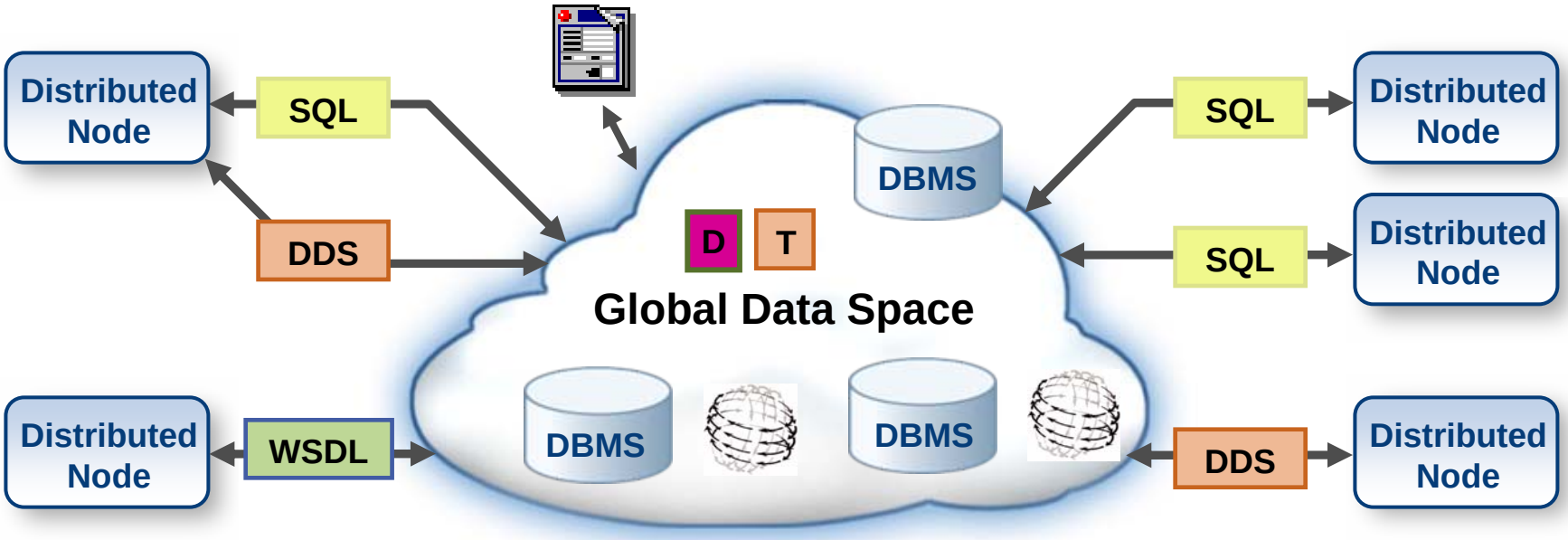
- A last-value cache is already built-in into every Writer in the system
  - Can used in combination with a Durable Writer
- A late joiner will automatically initialize to the last value
- Last value cache can be configure with history depth greater than 1
- The Persistence Service can be used to provide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

Historical cache

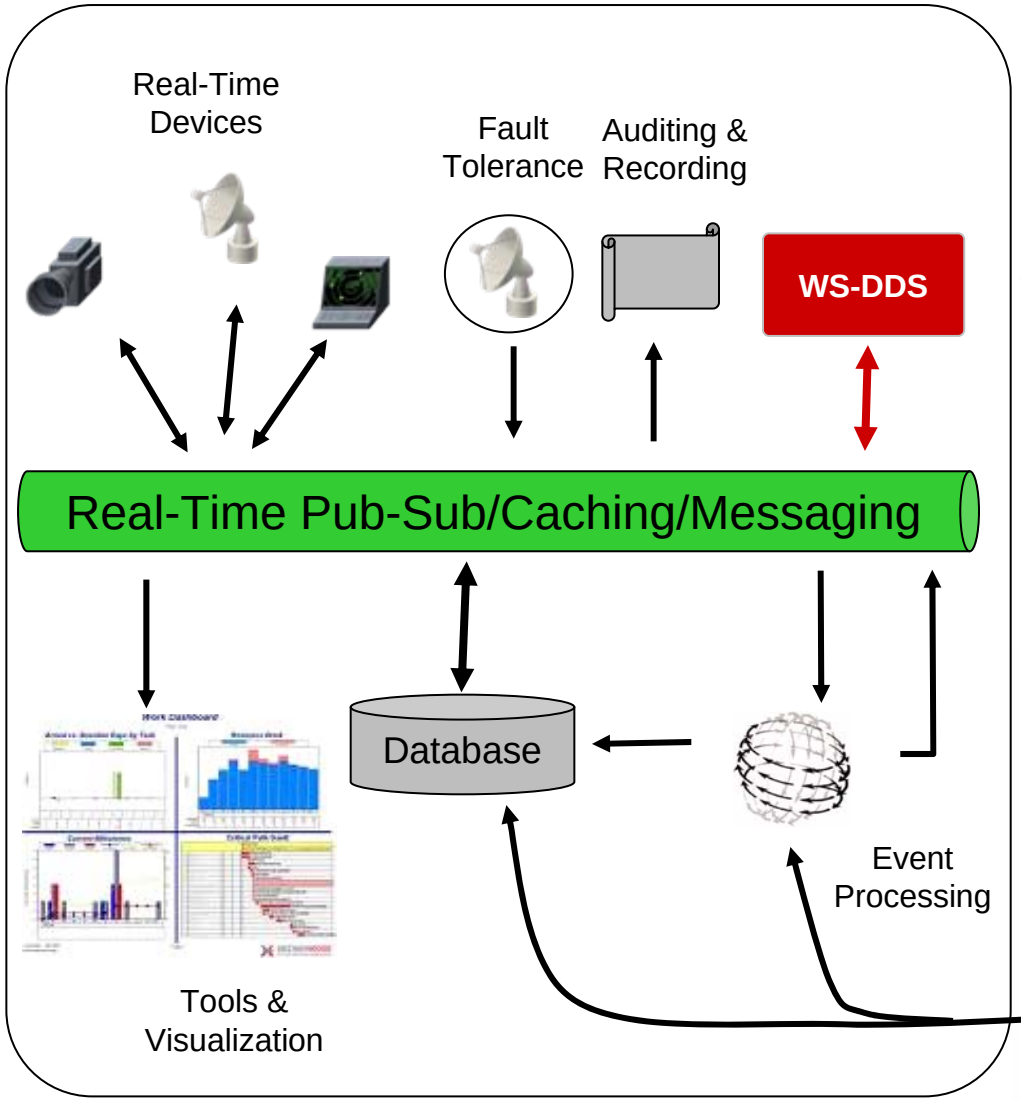
- A partial historical cache is already built-in into every Writer in the system
  - Can used in combination with a Durable Writer
- The Persistence Service can be used to provide a historical cache with larger/unlimited depth
- A late joiner will automatically initialize to the desired history
  - Currently amount of history can only be specified as message count.
  - Next release will also allow a age/time based specification.
- Request for historical cache is done by creating a Reader with the desired history depth specified as a QoS

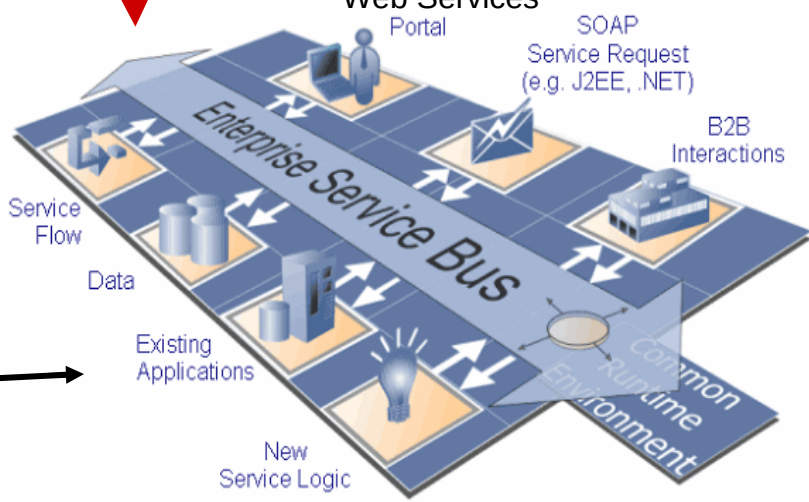
#4 Interoperability

- Protocol
  - Interoperability Wire protocol adopted in 2006
- Languages: C/C++, Java, ADA, .NET
- Systems/Platforms/Models
  - Data distribution (publishers and subscribers): **DDS**
  - Data management (storage, retrieval, queries): **SQL**
  - ESB Integration, Business process integration: **WSDL**

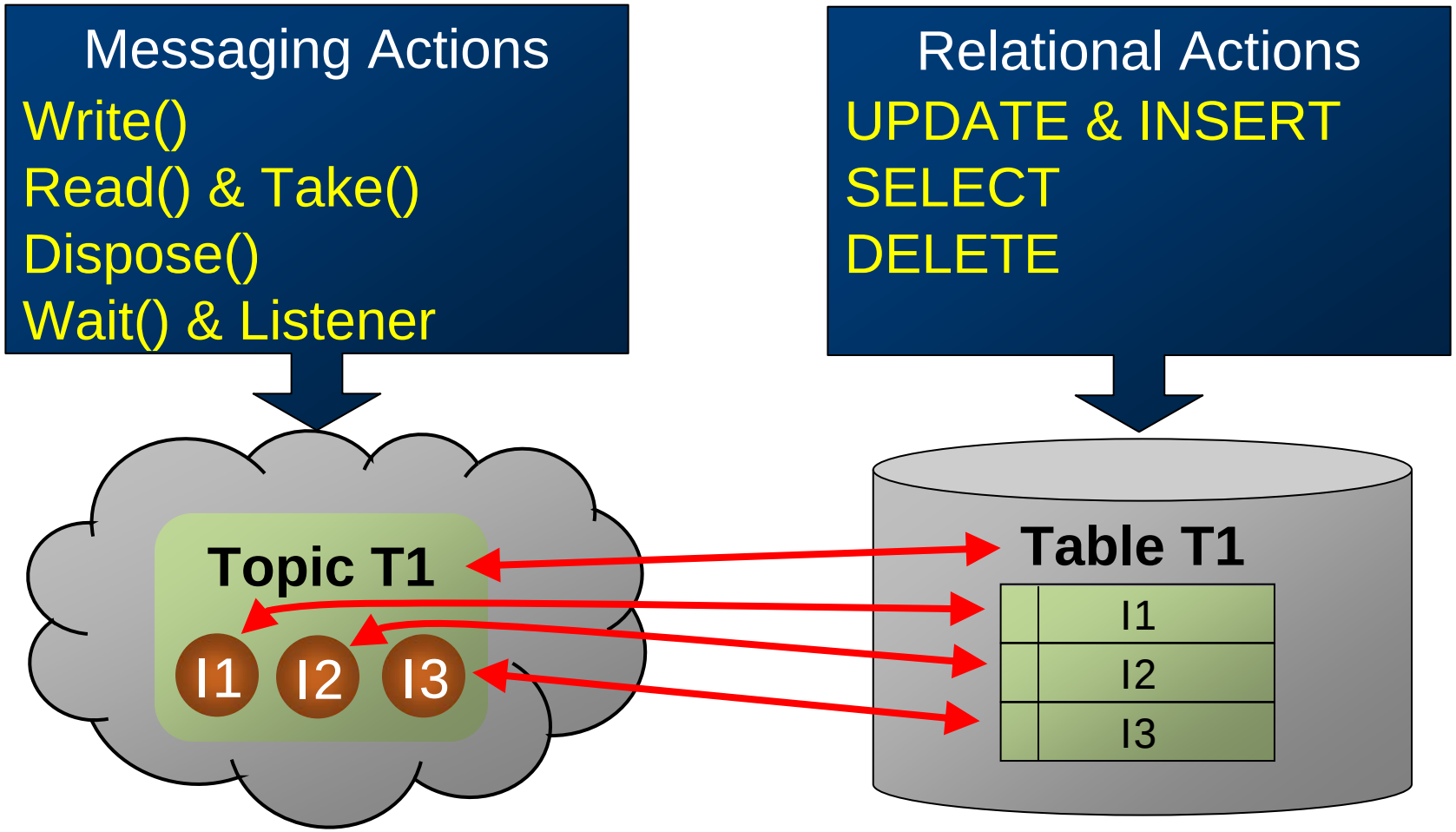


#5 Natural with SOA building blocks





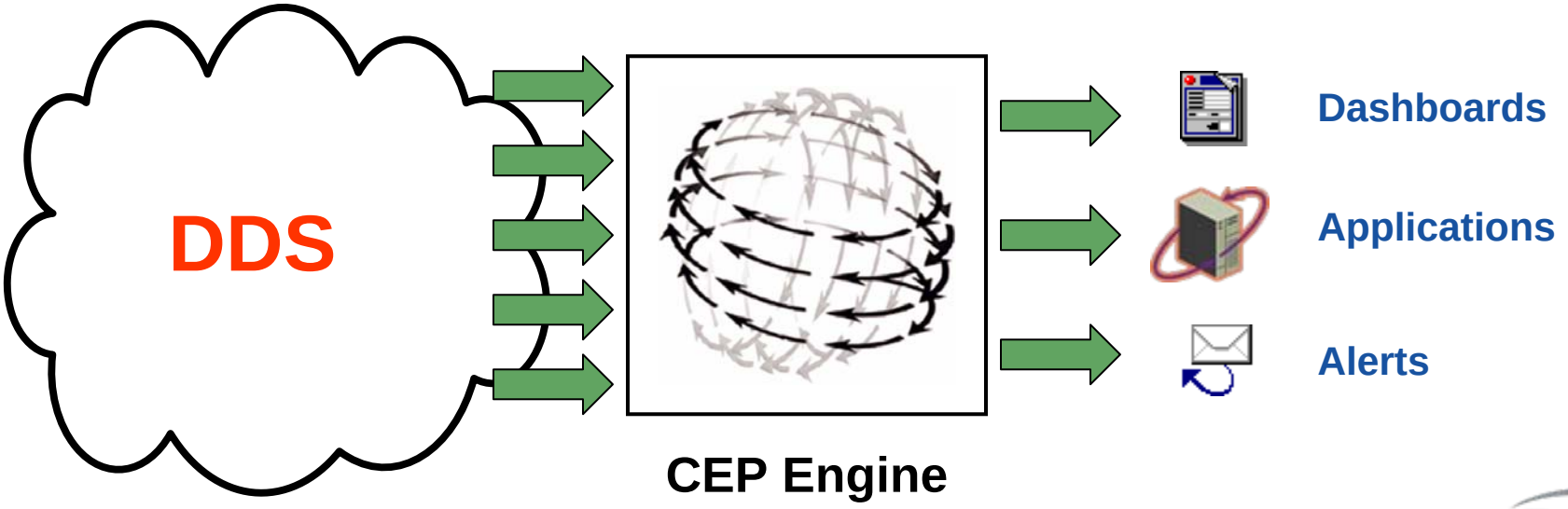
Relational Database Integration



Event driven – The fastest way to observe database changes!

Complex Event Engine Integration

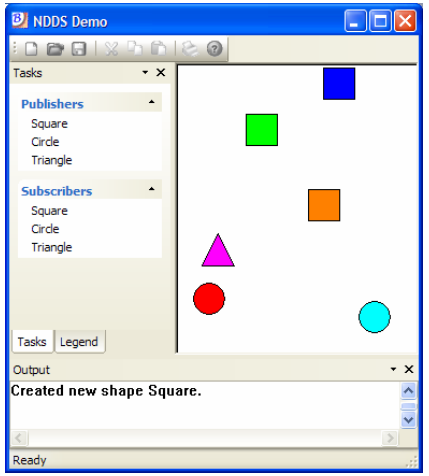
- CEP: programmable engines used to transform “data” into “information”
- CEP engines are programmed using a derivative of SQL
- CEP engines save time: They can implement a lot of the application logic:
  - Classification, Correlation, Aggregation, Filter, Cleansing, Pattern Detection, etc.
- DDS is the perfect ‘data’ and ‘information’ pipe for CEP engines
  - Use high-speed data streams (1,000-1,000,000 msg/sec)
  - Require latency measured in sub-milliseconds
  - Demand access to events from a heterogeneous systems



DDS as a Web Service

Web
Enabled
Protocols

Hi-Performance
Embedded
Real-Time



HTTP

Generic
Web Client
(.NET / Java)

SOAP

WS-DDS
DDS Web Service

IONA
Web Client

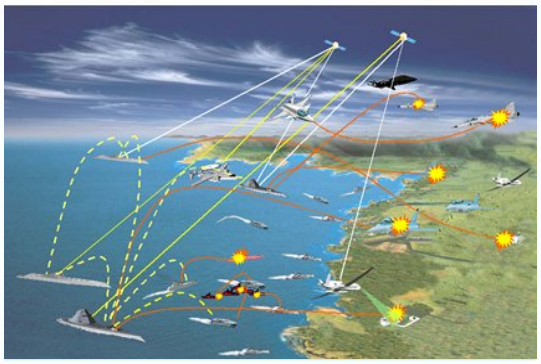
IONA Artix Transports
SOAP, JMS, IIOP

DDS-RTPS

Global
Data Space
RTI, DDS, Real-Time

DDSParticipant

DDSParticipant



DDS In depth

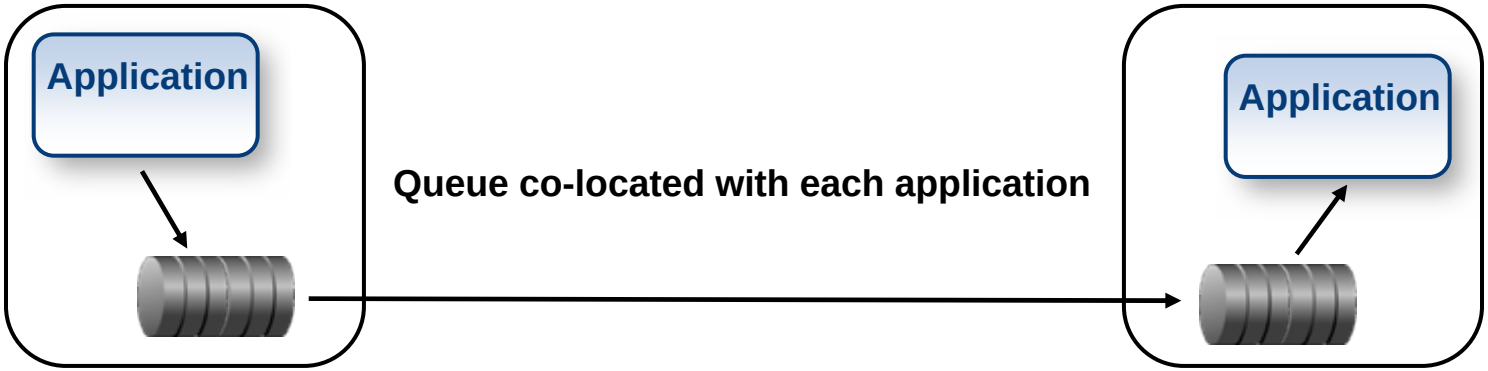
Agenda

- Part I -- Concepts
  - Middleware Models: Messaging & Data-Distribution
  - Publish/Subscribe & Data-Centric Design

- Part II -- Patters & Use-Cases
  - Sending Data
  - State Information
  - Alarms and Events
  - Discovery
  - Sample Application Requirements mapped to QoS

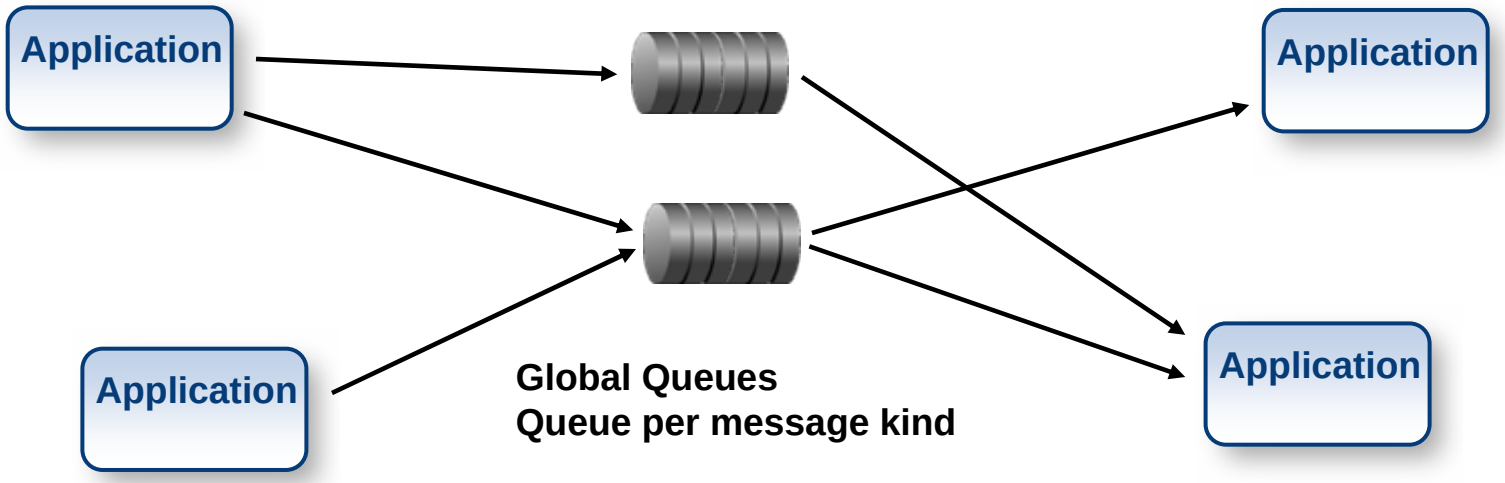
Do-yourself Message-Centric System

- 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



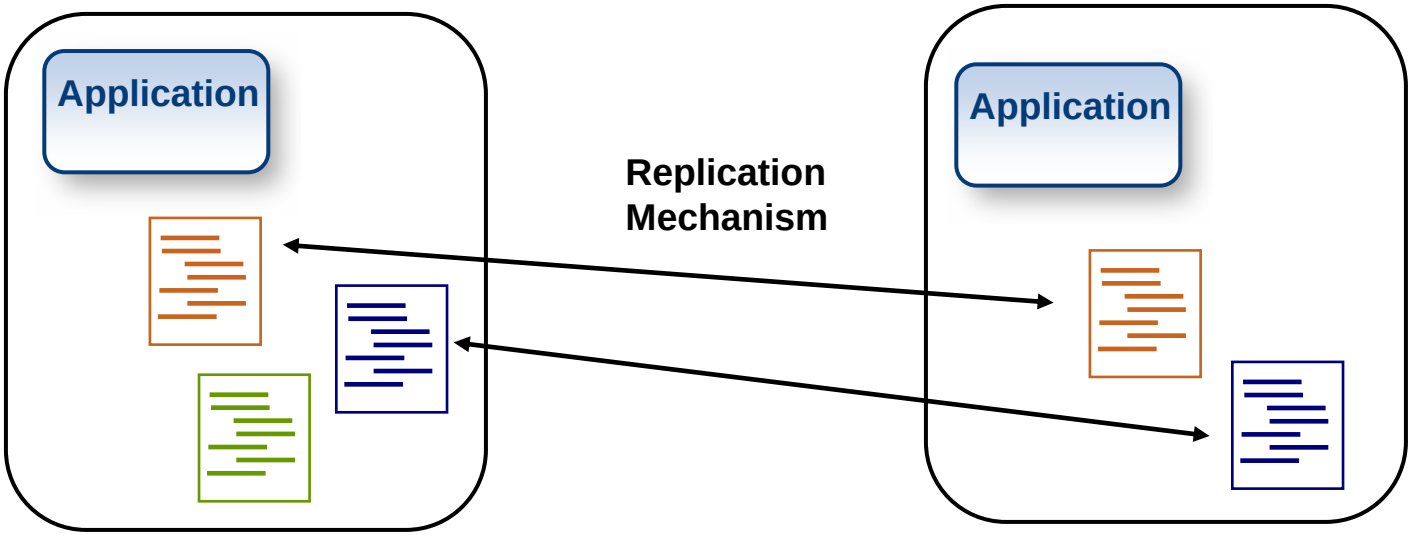
Middleware-based Message-Centric Systems (JMS)

- 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



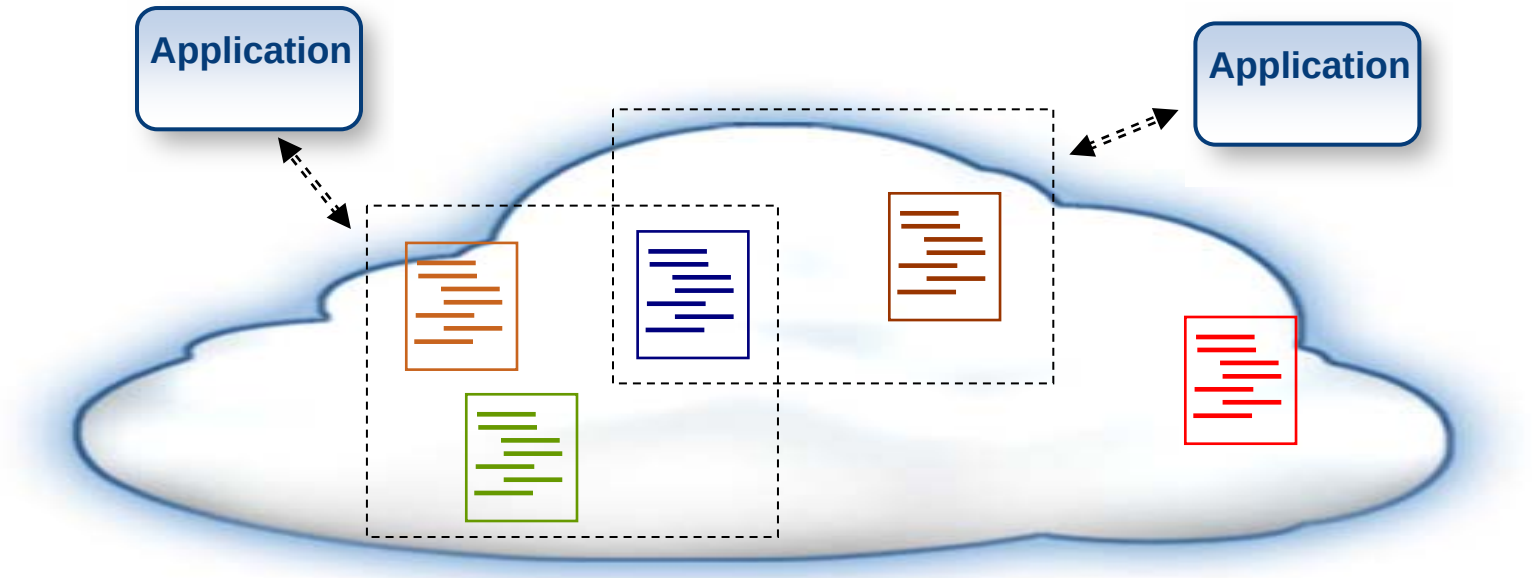
Do-yourself Data-Centric System

- 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)

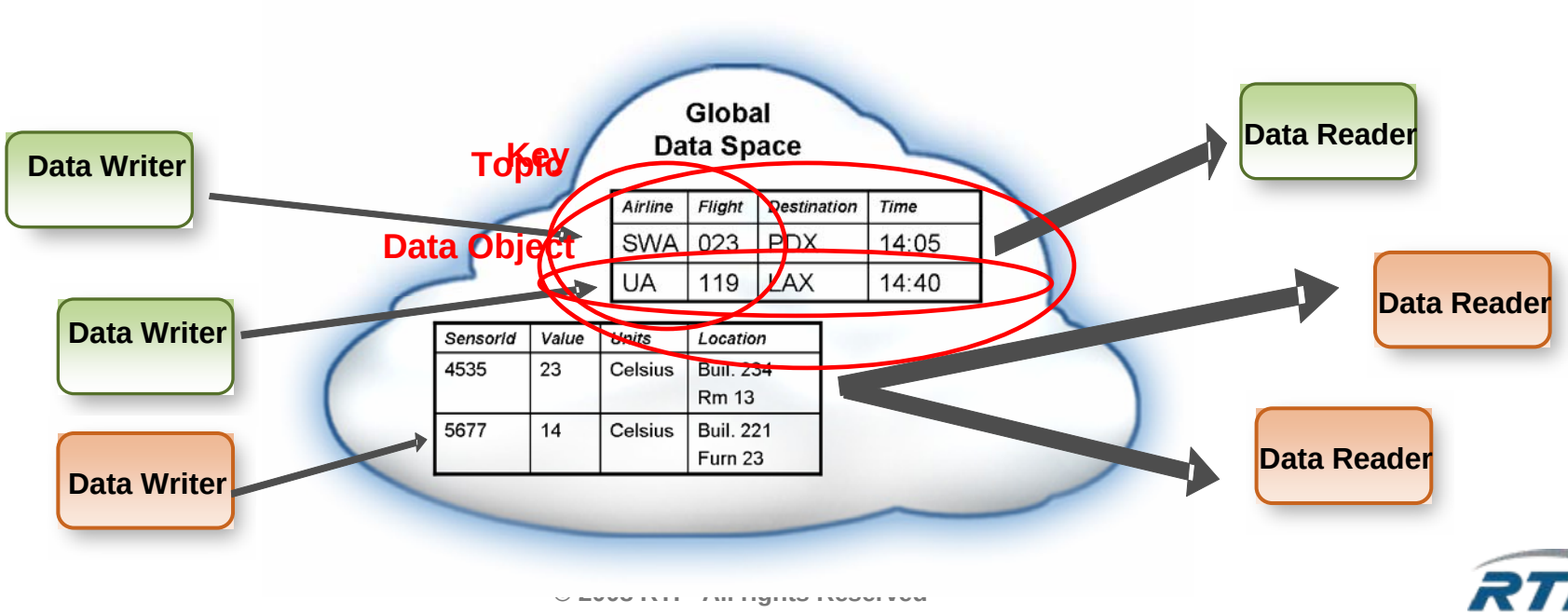
- Shared data-space, shared state
- Asynchronous communication
- Decoupled in Lifespan, Content, presentation
- Excellent performance & scalability
- Subsumes Message-Centric via QoS



DDS Data-Centric Model

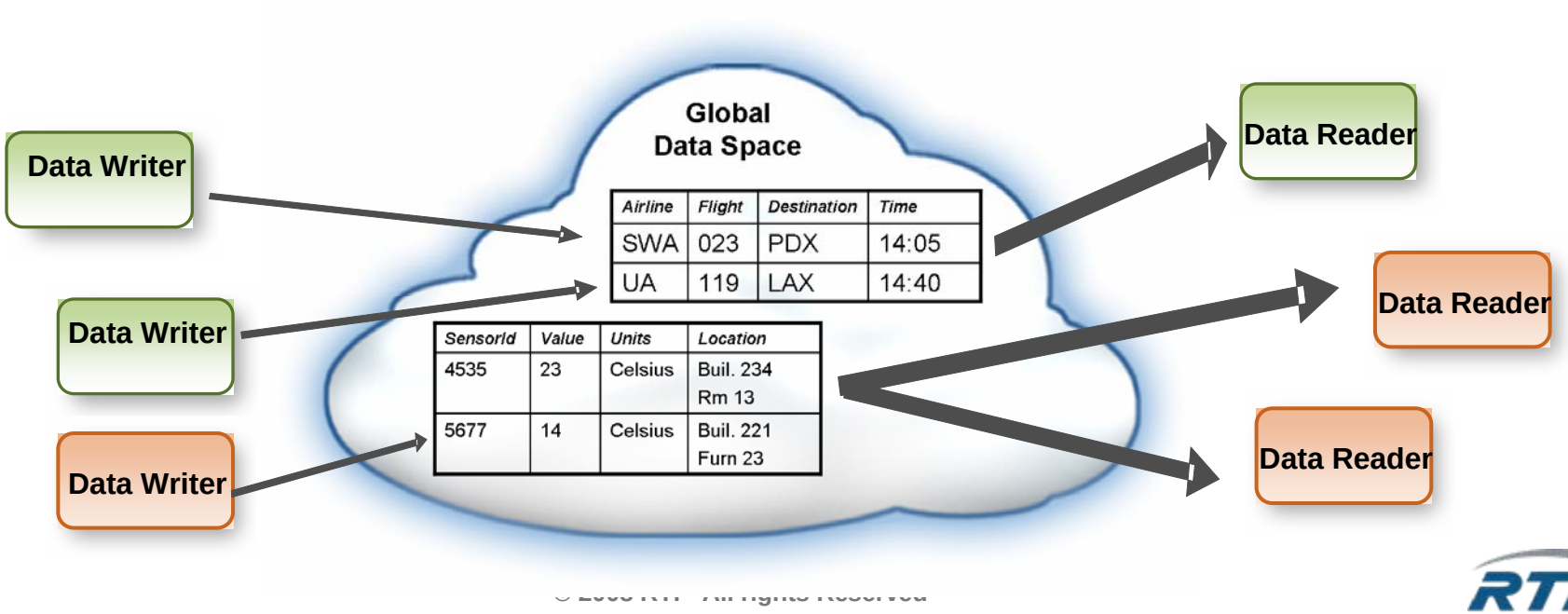
Provides a virtual "**Global Data Space**" that is accessible to all interested applications.

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



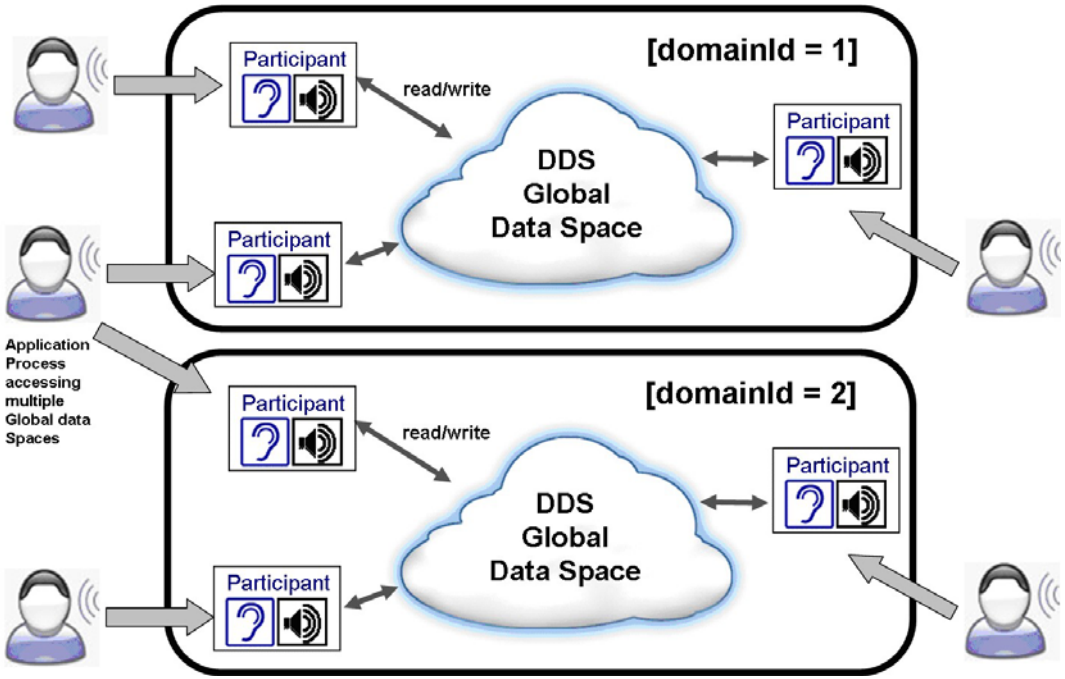
DDS Global Data

- Address in Global Data Space = (DomainId, Topic, Key)
  - Each topic corresponds to a multiple data instances with a common schema
  - A DataWriter can write to any instances of a single topic
  - Multiple DataWriters may write to the same instance
  - A DataReader receives updates from all instances of a single topic
  - Multiple DataReaders may read from the same instances & values



DDS Global Data: Domains

- Address in Global Data Space = (DomainId, Topic, Key)
  - Each Domain is identified by the value of the domainId
  - Each Domain is a separate Global Data Space
    - The same Topic name can mean different things in different domains
    - The same Topic can have different Types on each Domain
  - An application may join multiple Domains
  - Domains can be used for isolation, scalability, modulariy



Example: Publication

```
// Entities creation
DomainParticipant participant =
    TheParticipantFactory->create_participant(
       domain_id, participant_qos, participantA_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

RTI

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;
}
```

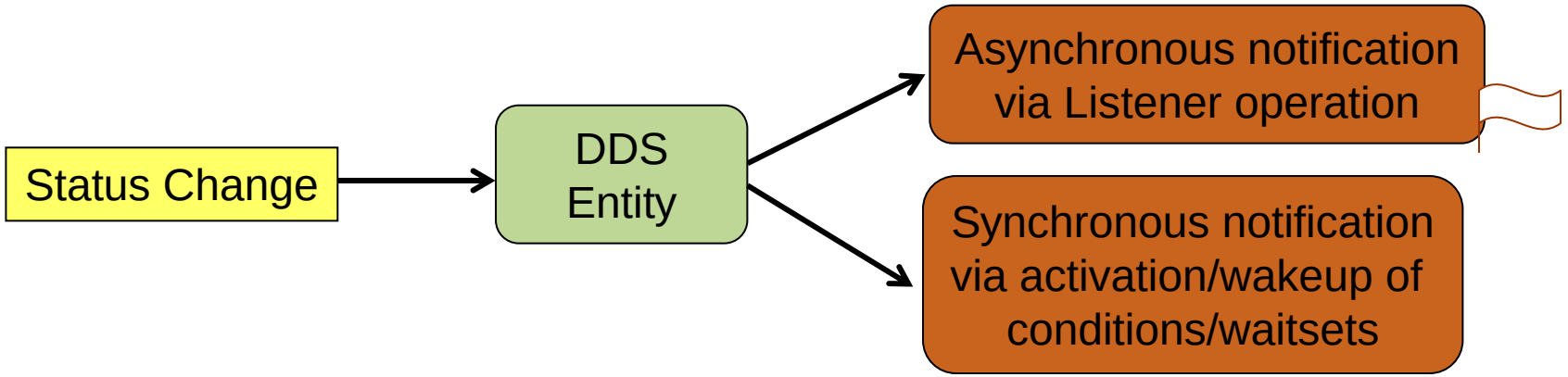
Listeners, Conditions and Statuses

- 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

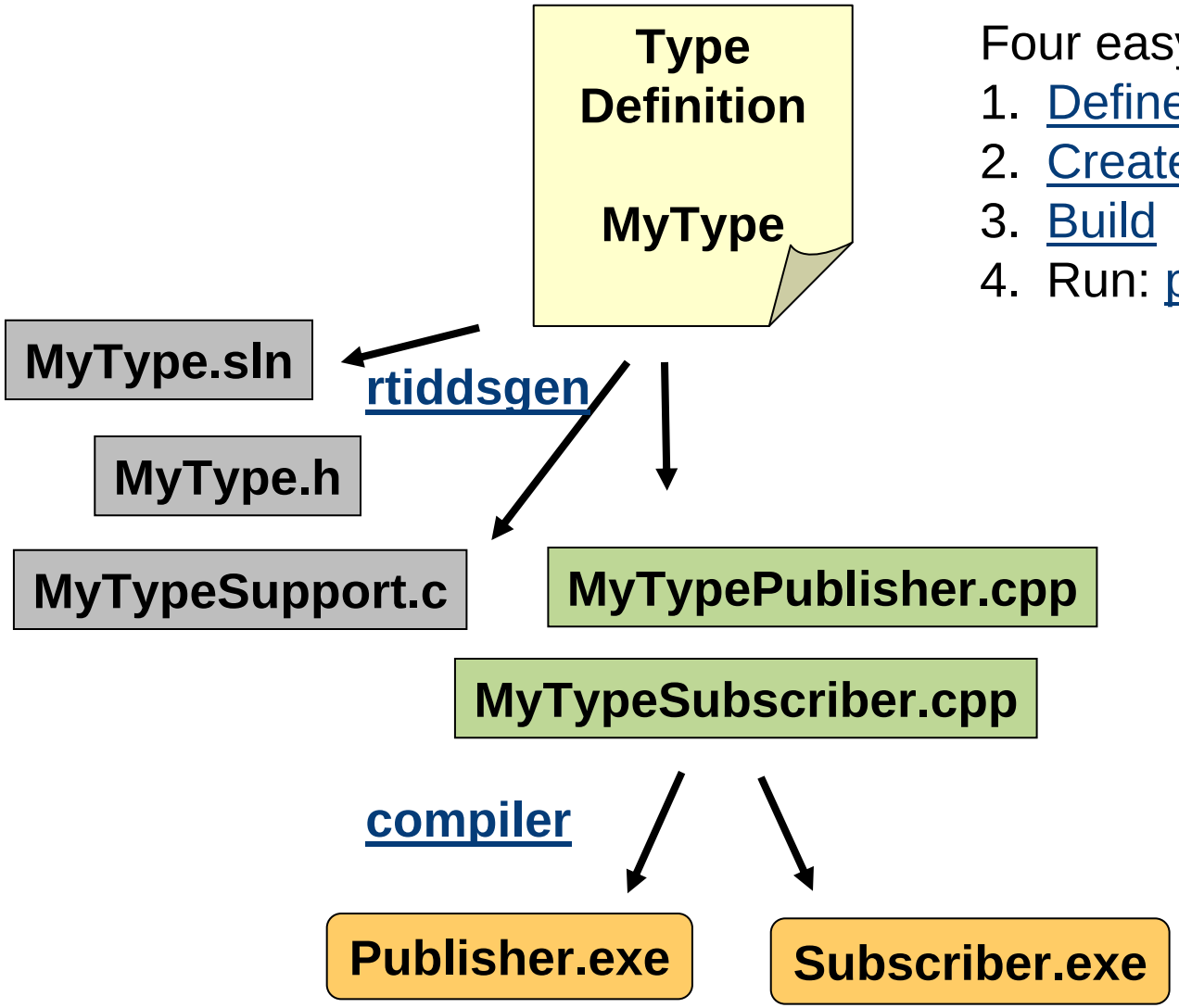
STATUS	Entity	Listener operation
INCONSISTENT_TOPIC	Topic	on_inconsistent_topic
DATA_ON_READERS	Subscriber	on_data_on_readers
LIVELINESS_CHANGED	DataReader	on_liveliness_changed
REQUESTED_DEADLINE_MISSED	DataReader	on_requested_deadline_missed
RUQESTED_INCOMPATIBLE_QOS	DataReader	on_requested_incompatible_qos
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



Hands-on Example (C++)



Four easy steps:

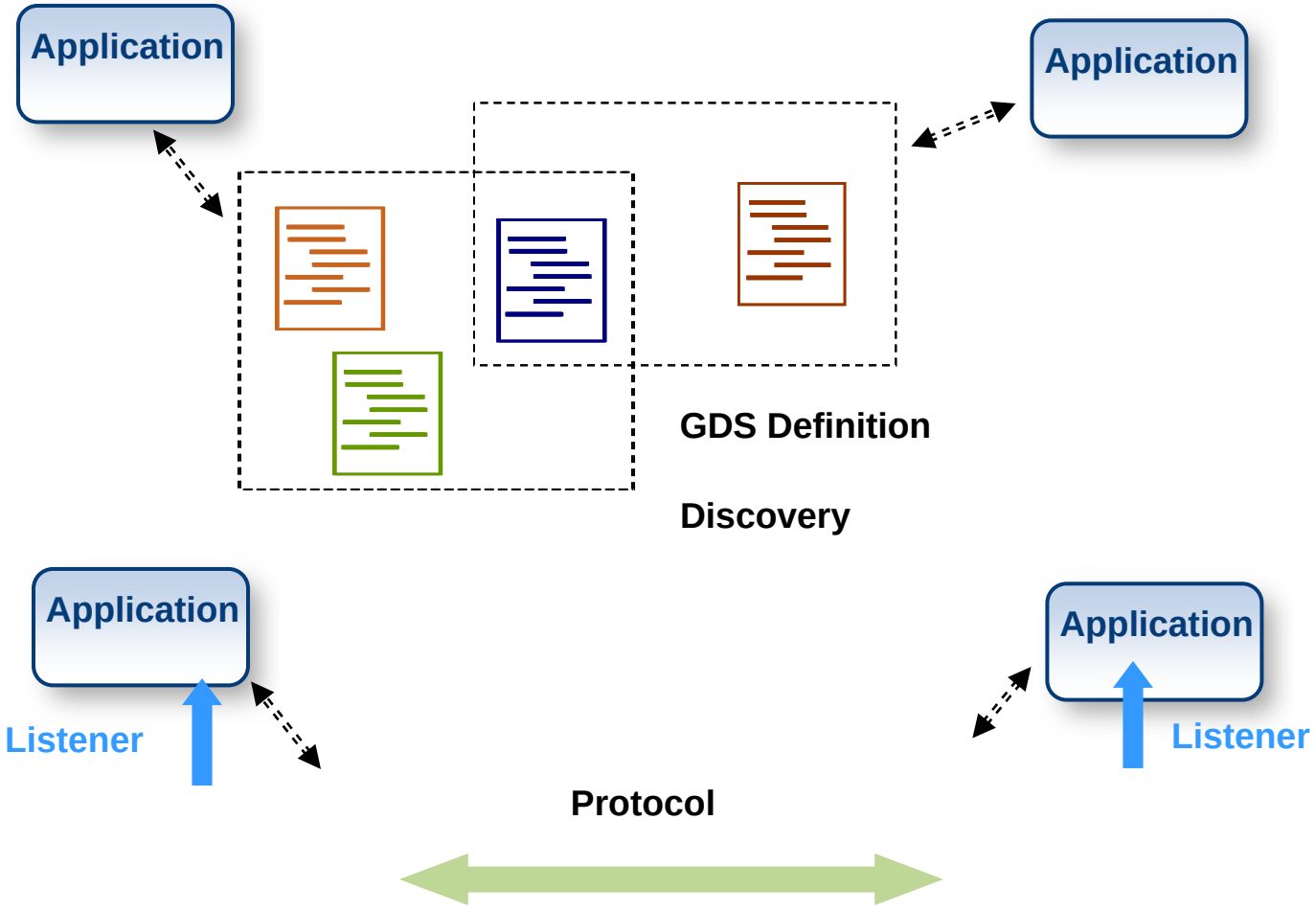
1. Define your data
2. Create your project
3. Build
4. Run: publisher subscriber

Aux:

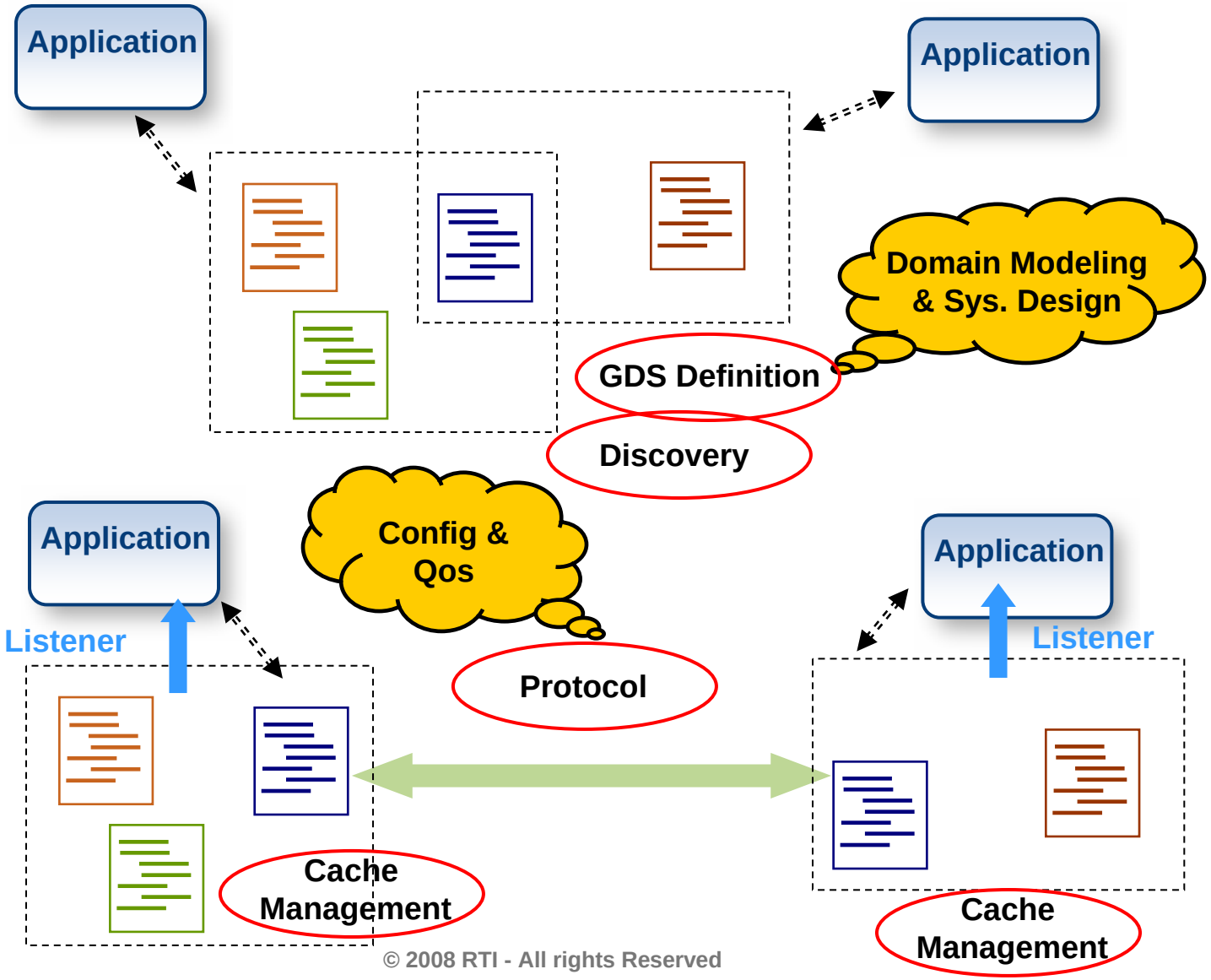
File Browser

Console

Components/Mechanics of the GDS



Components/Mechanics of the GDS



Designing a Data-Centric System

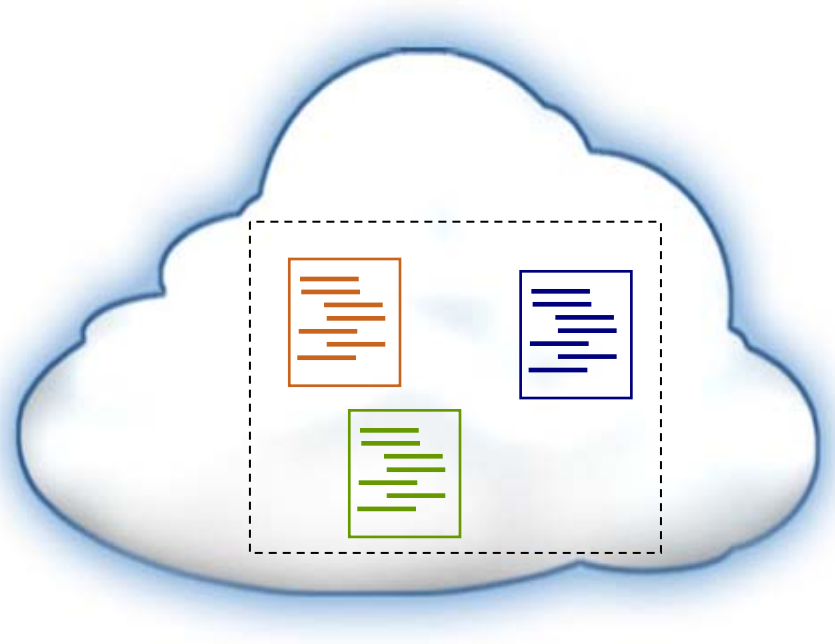
- Define/Model the Global Data Space
- Configure the Cache Management
- Configure Discovery
- Configure the Protocol

- Configure/Use hooks for
  - Fault detection
  - Controlled access

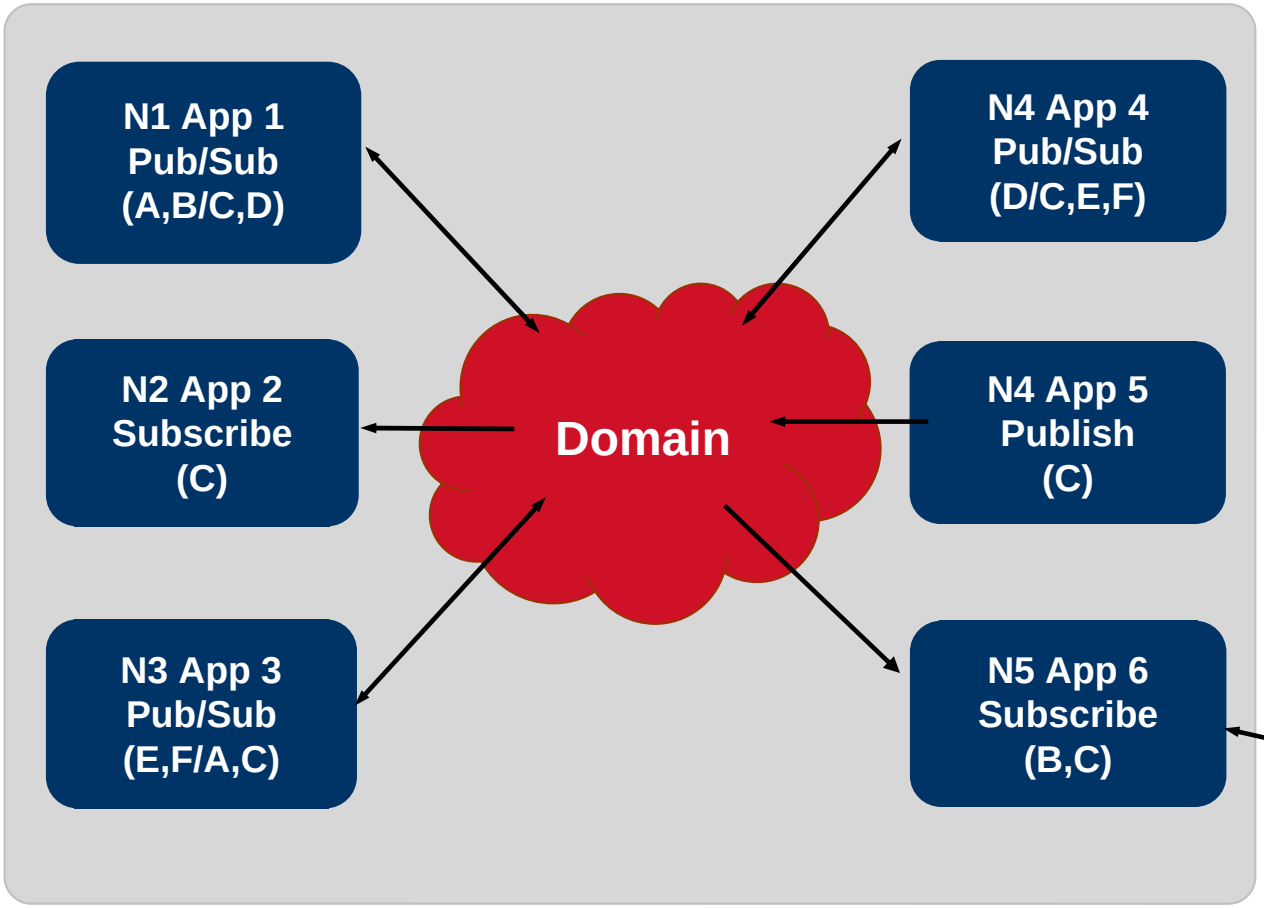
Global Data Space / Global State

- Identify the number of domains

- Domain Information model
  - Topics
  - Types
  - Keys
  - Ownership



Domain and Domain Participants

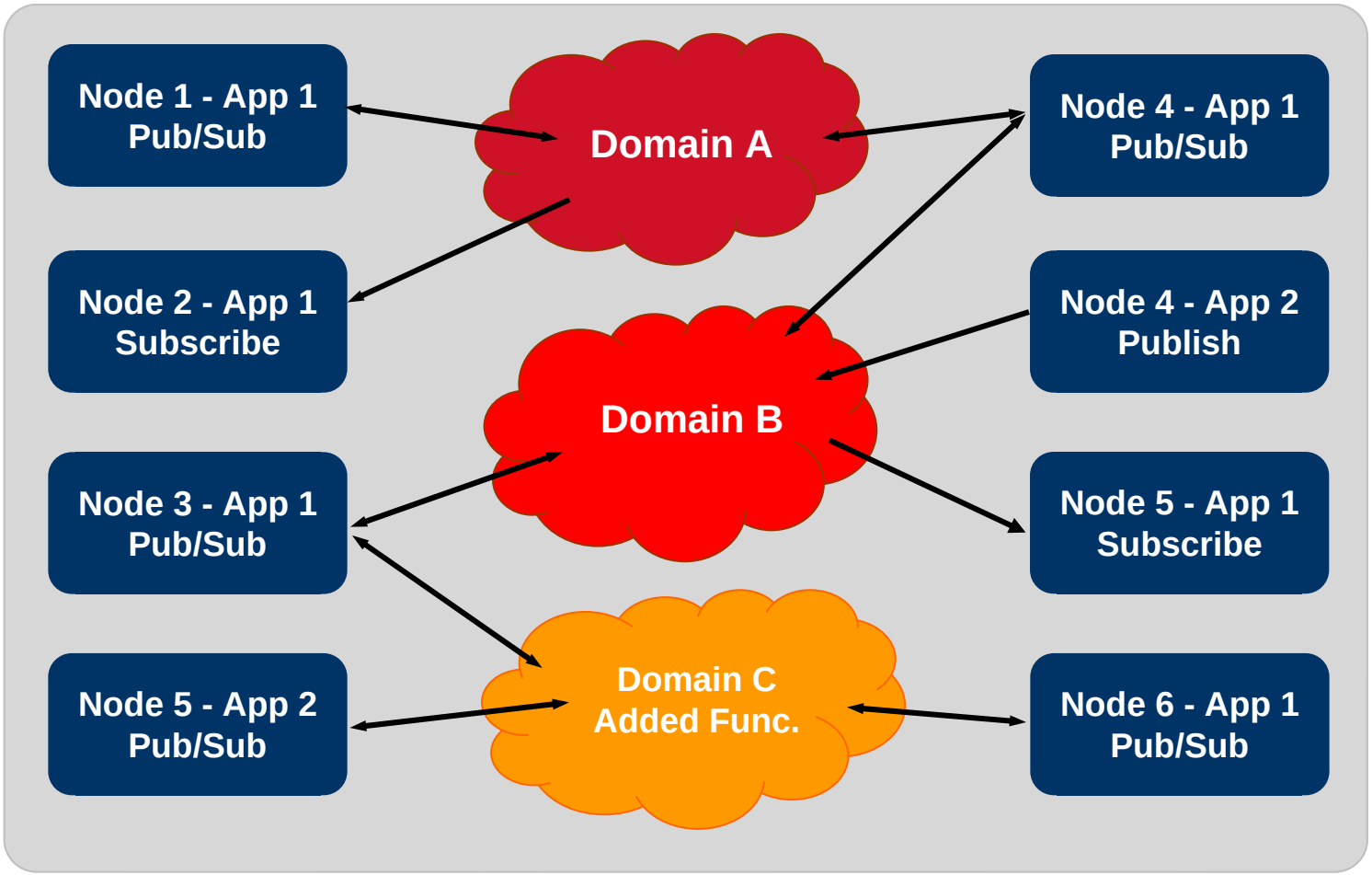


- Container for applications that want to communicate
- Applications can join or leave a domain in any order
- New Applications are “Auto-Discovered”
- An application that has joined a domain is also called a “Domain Participant”

Single ‘Domain’ System

Domain and Domain Participants

Using Multiple domains for Scalability, Modularity & Isolation



Multiple Domain System

Topics & Datatypes, Keys & Subjects

Topic “MarketData”

Data-type (name-type-value pairs)

source	type	symbol	Exchange	volume	bid	ask
OPRA		IBM	NYSE	200000	118.30	118.36
OPRA		AAPL	NASDAQ		171.20	171.28
RTFP	EQ					

Key fields → Subject

Additional fields (payload)

Topic “OrderEntry”

Exchange	type	Symbol	Order num	number	limit	stop	expiration
NYSE	BUY	IBM	11956	500	120	-	DAY
NYSE	BUY	IBM	11957	1000	124.5	124	DAY
NASDAQ	SELL	AAPL	11958	400	-	160	DAY

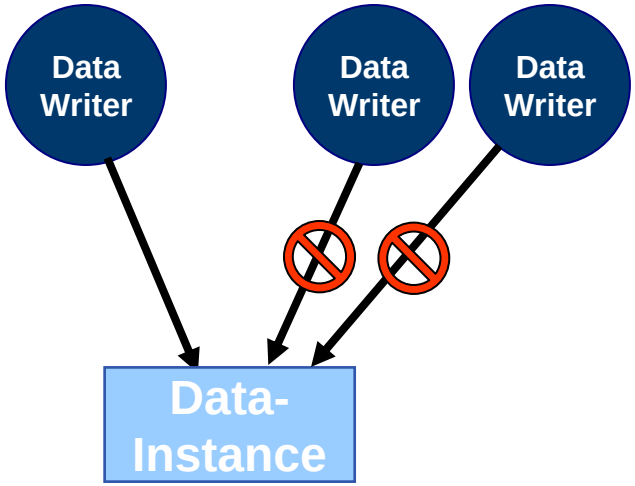
Subject

Key fields

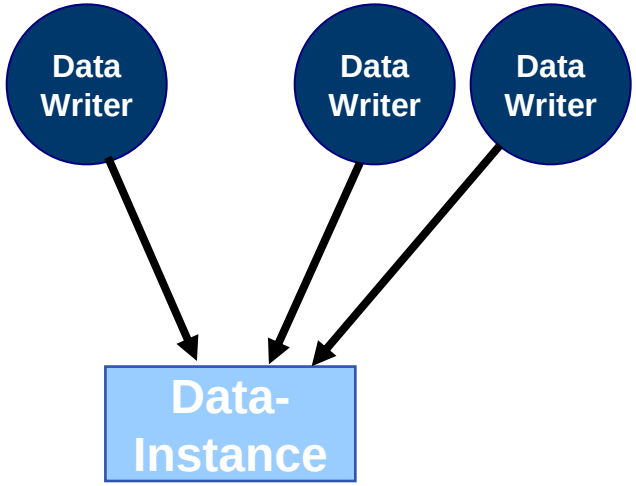
QoS: Ownership

Specifies whether more than one DataWriter can update the same instance of a data-object

Ownership = EXCLUSIVE
"Only highest-strength data writer can update each data-instance"



Ownership = SHARED
"All data-writers can each update data-instance"

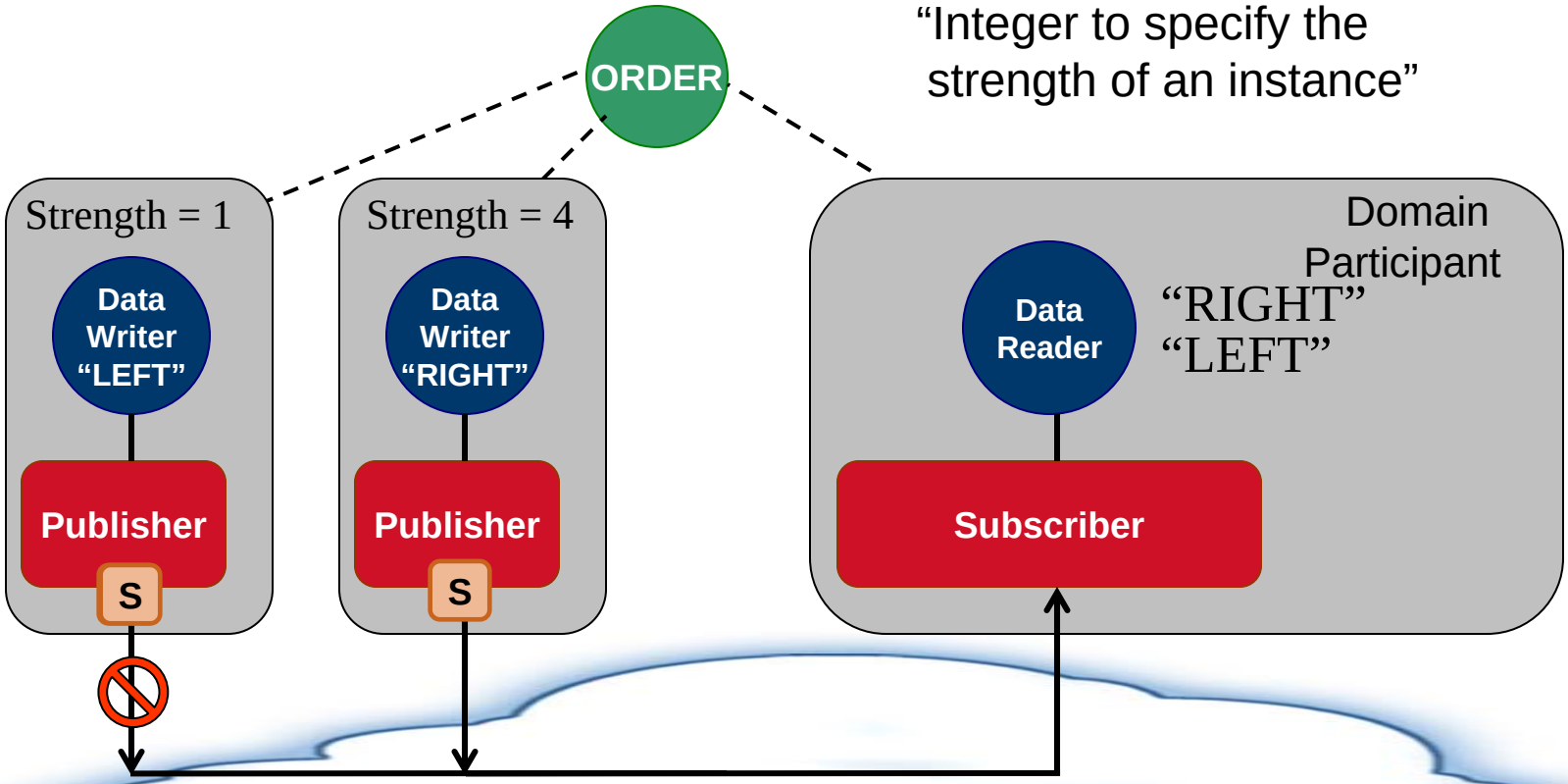


Provides fast, robust, transparent replacement for fail-over and/or take-over.

QoS: Ownership Strength

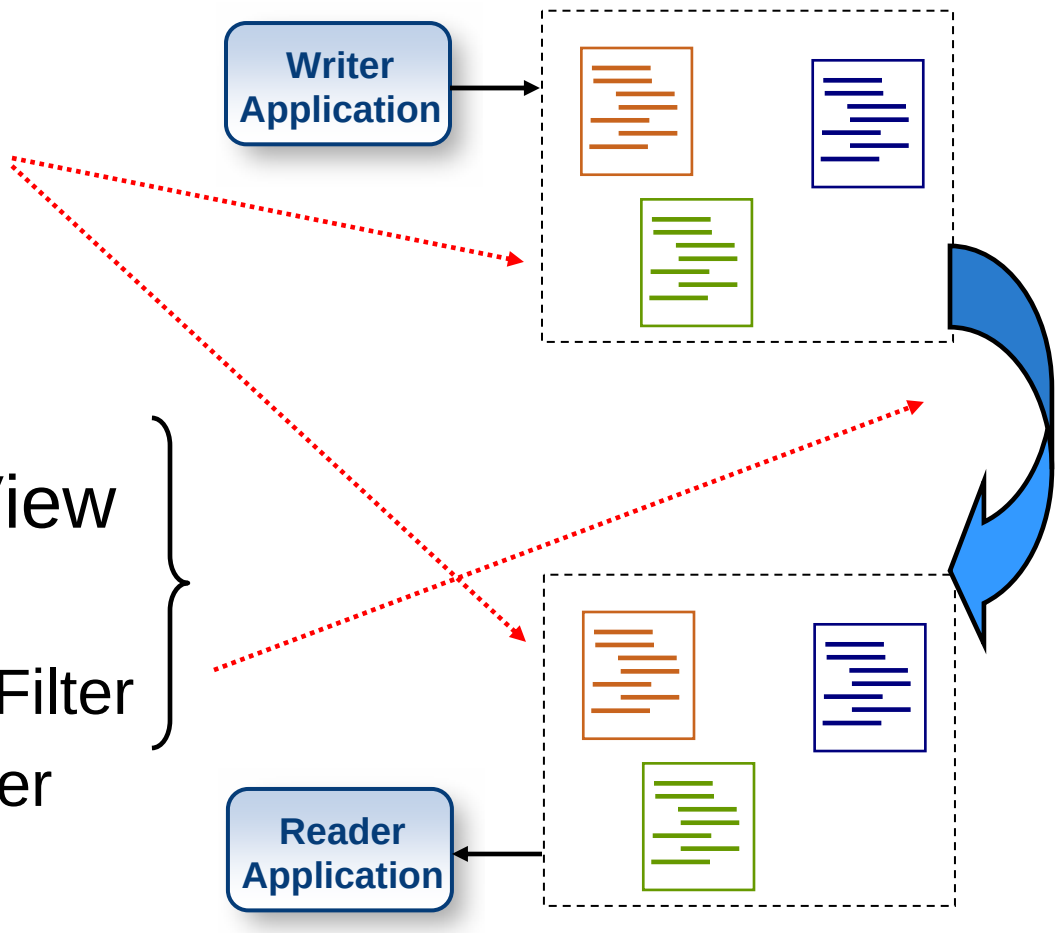
Specifies which DataWriter is allowed to update the values of data-objects

OWNERSHIP_STRENGTH
"Integer to specify the strength of an instance"



Configure the Cache Management

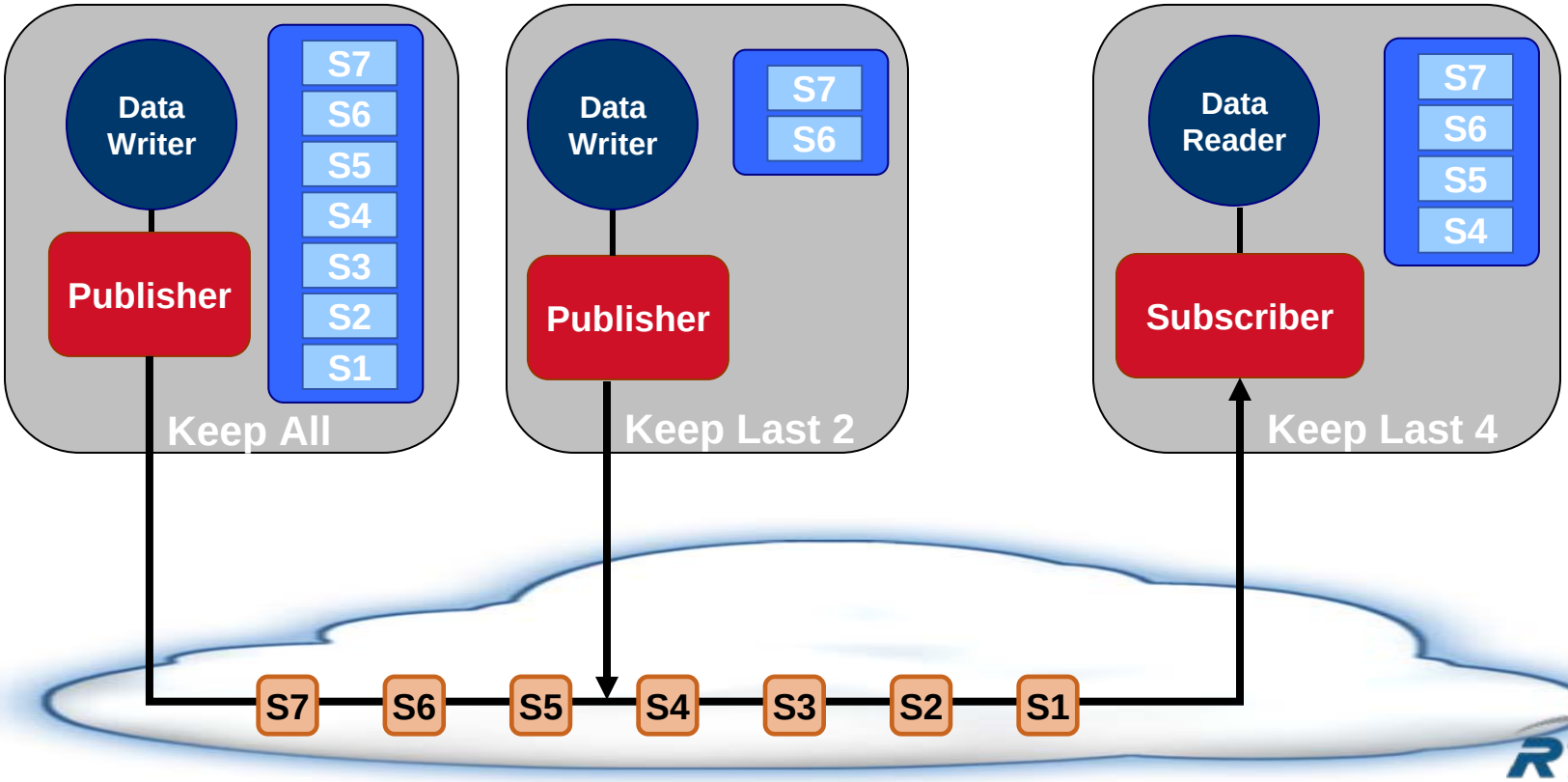
- Cache State Content
  - History
  - Lifespan
  - Persistence
  - Resources
- Reader Cache View
  - Partitions
  - Content-Based Filter
  - Time-Based Filter
  - Order



QoS: History – Last x or All

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

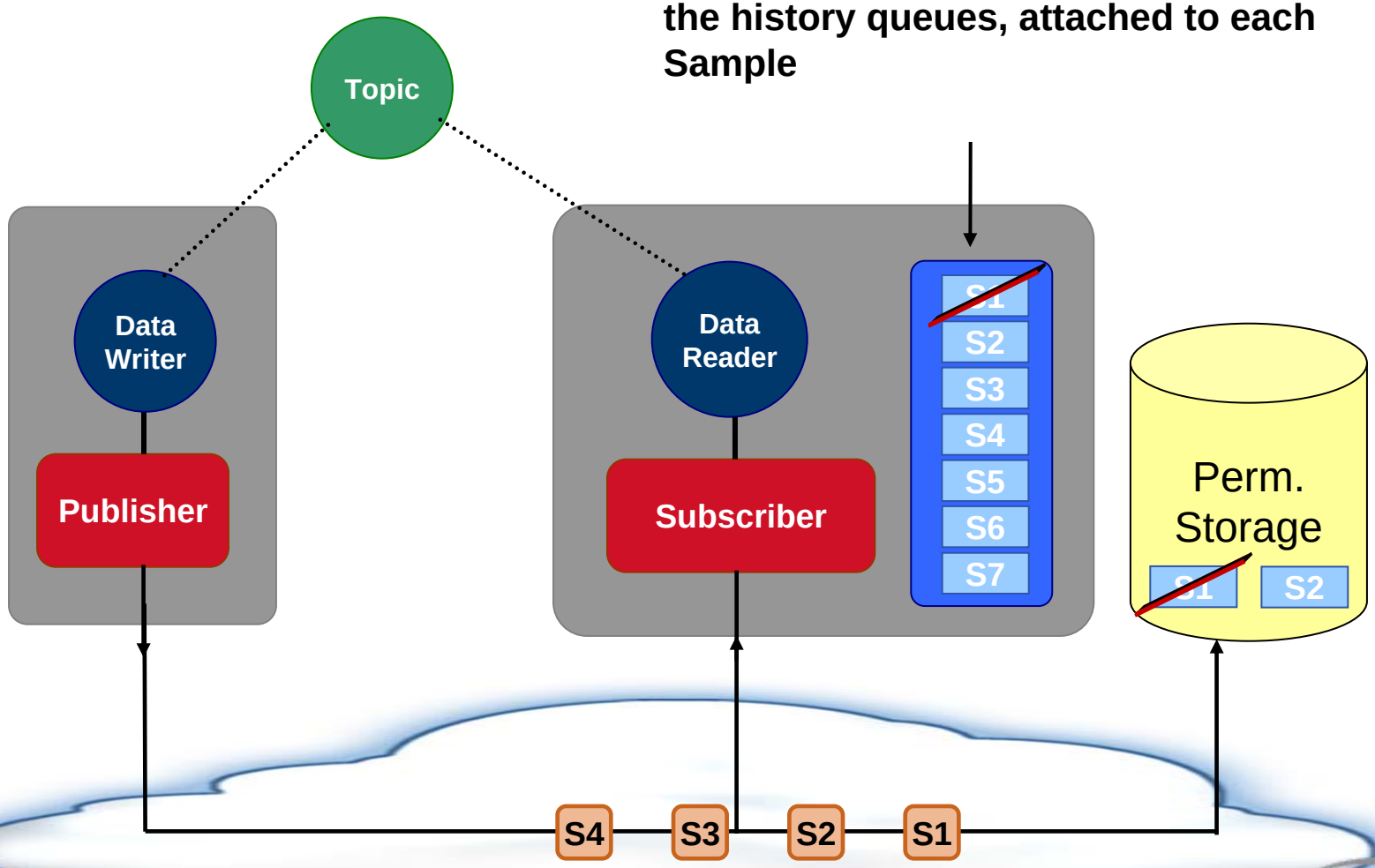
KEEP_LAST: "depth" integer for the number of samples to keep at any one time



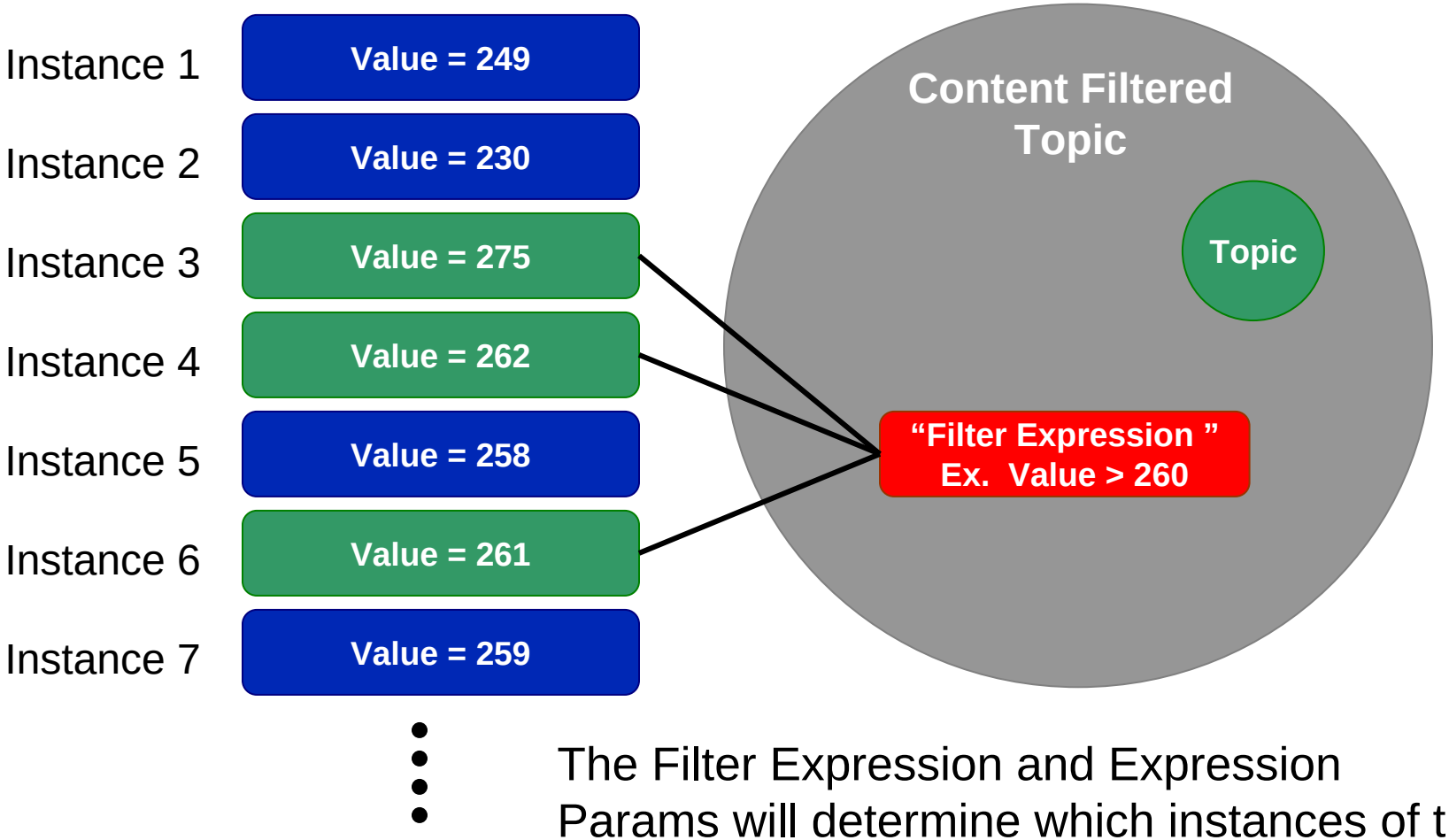
QoS: Lifespan

User can set lifespan duration

Manages samples in the history queues, attached to each Sample



Content-Based Filtering



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

Subscriptions: By Topic, Subject, Content

Topic: "Market Data"

Field	Source	Symbol	Type	Exchange	Payload: Volume	Bid	Ask	…
Value	*	*	*	*	*			

Topic: "Order Entry"

Field	Symbol	Type	Exchange	Payload: OrderNumber	Symbol	OrderKind	Stop	Limit	…
Value	*	*	*NYSE*	*					

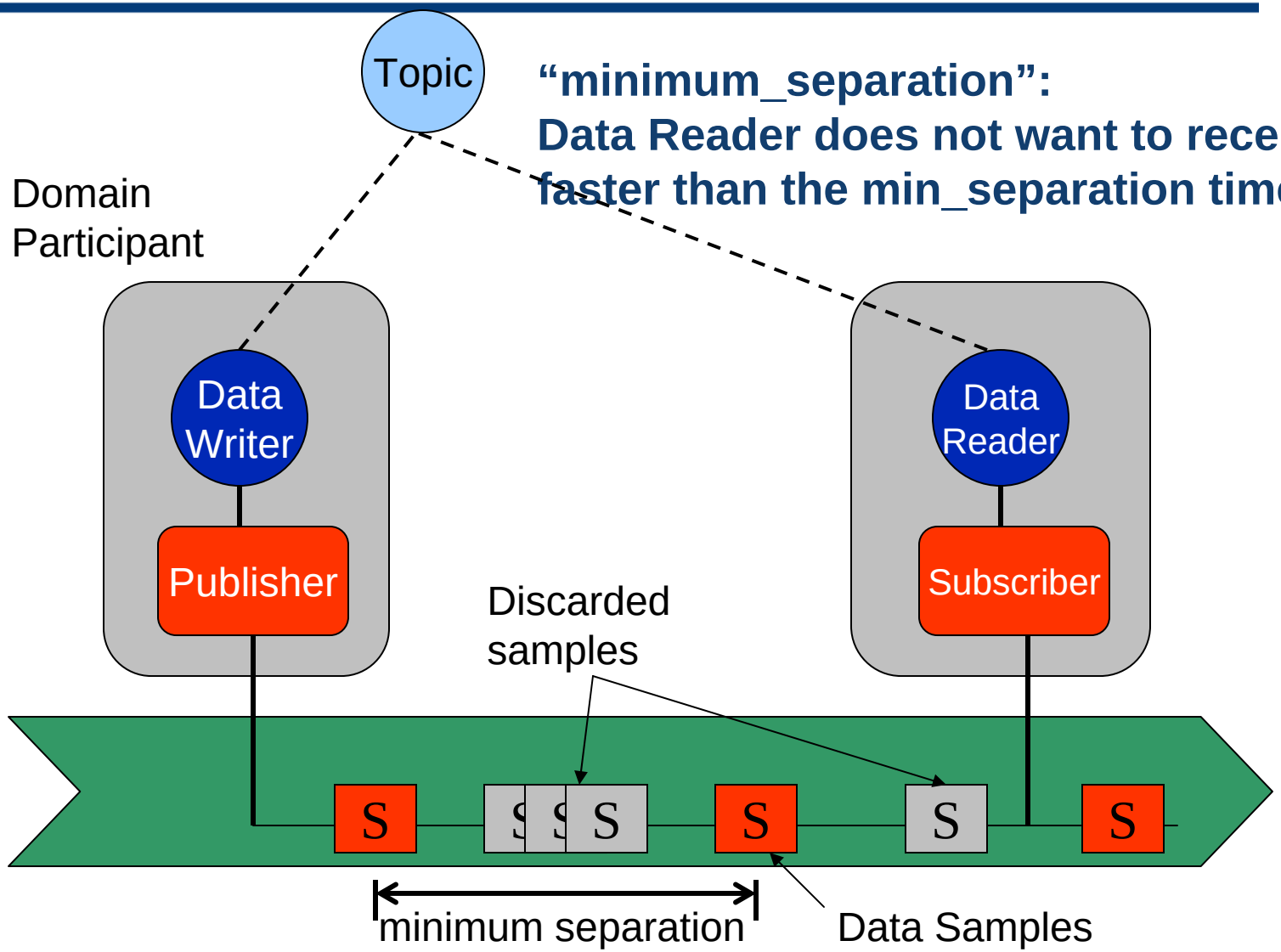
Subject Filter (for a Reader)

Topic: "Market Data"

Field	Source	Symbol	Type	Exchange	Payload
Value	*REUTERS*	*	*EQ*	*NYSE*	*Volume > x, Ask < y*

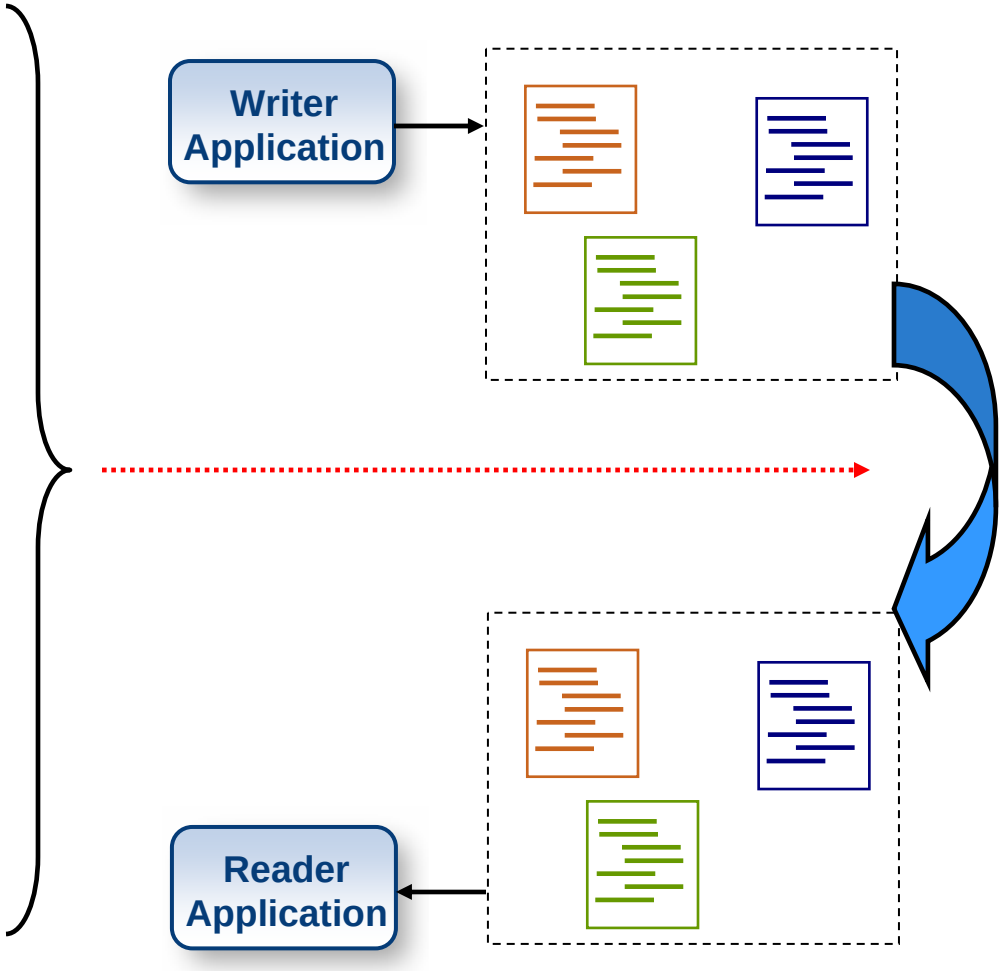
Subject Filter (for a Reader) **Payload Filter (for a Reader)**

QoS: TIME_BASED_FILTER



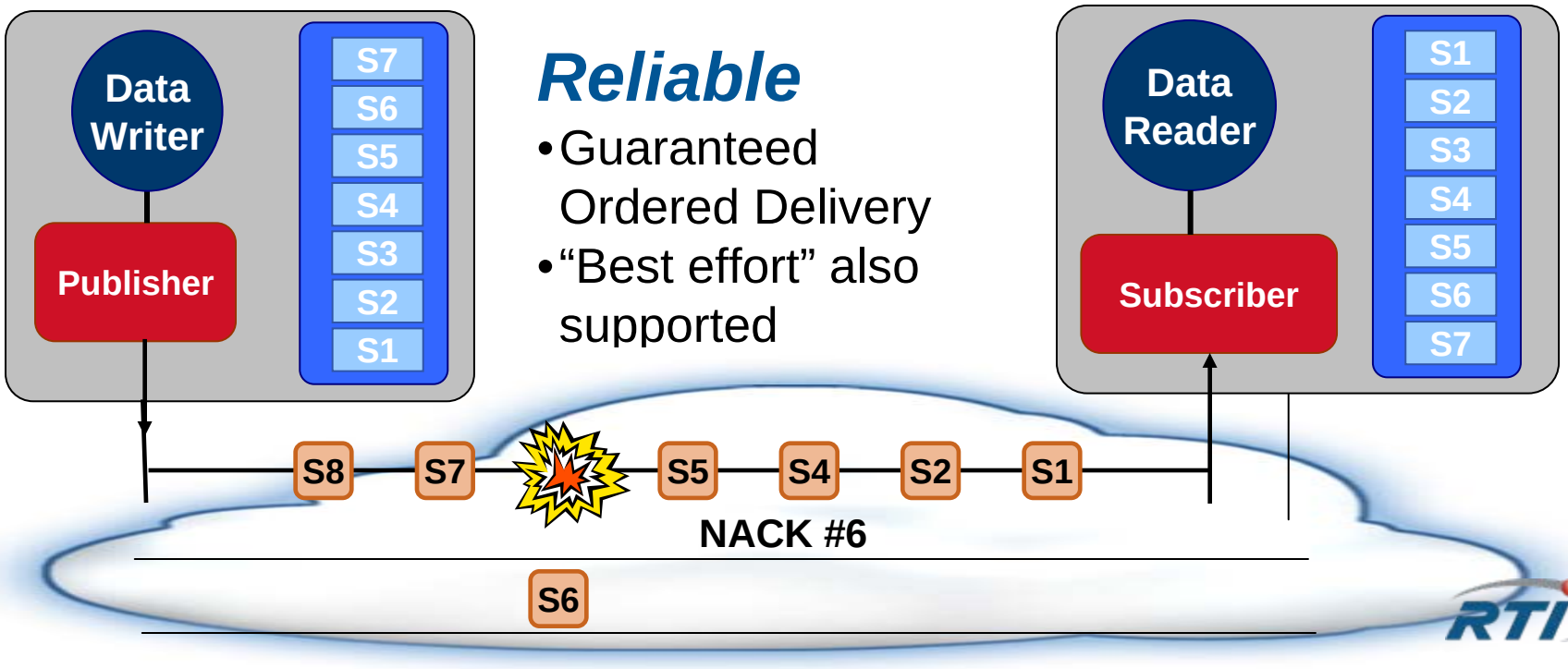
Configure the Protocol

- Discovery
- Reliability
- Liveliness
- Flow Control
- Asynchronous write
- Network Configuration
  - Enabled Transports + transport properties
  - Multicast addresses
  - Transport Priority
- OS settings
  - Threads
  - Memory



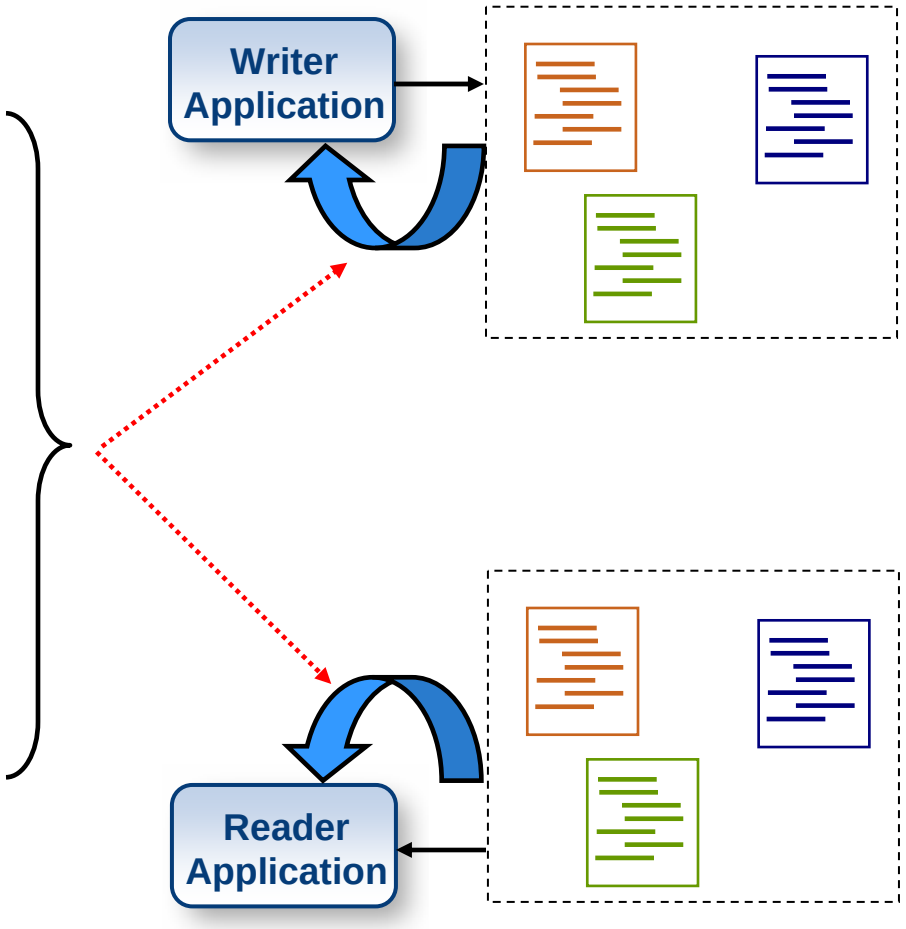
Tunable Reliability Protocol

- Configurable AckNack reply times to eliminate storms
- Fully configurable to bound latency and overhead
  - Heartbeats, delays, buffer sizes
- Performance can be tracked by senders and recipients
  - Configurable high/low watermark, Buffer full
- Flexible handling of slow recipients
  - Dynamically remove slow receivers

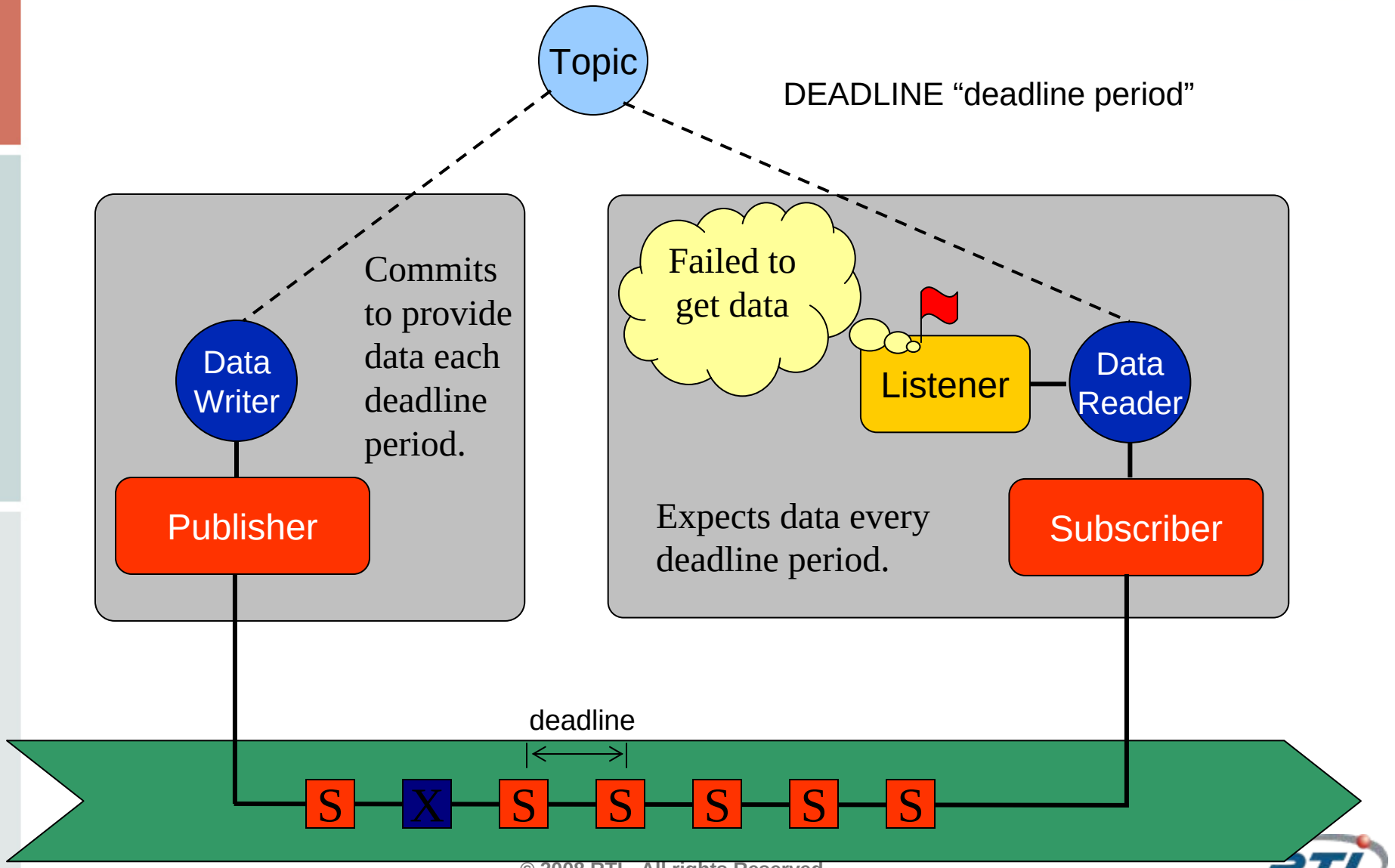


Configure Notifications, Fault Detection & Management

- Listeners
- Deadline Qos
- Liveliness Qos
- Built-in Readers
- Notification of matching



QoS: Deadline

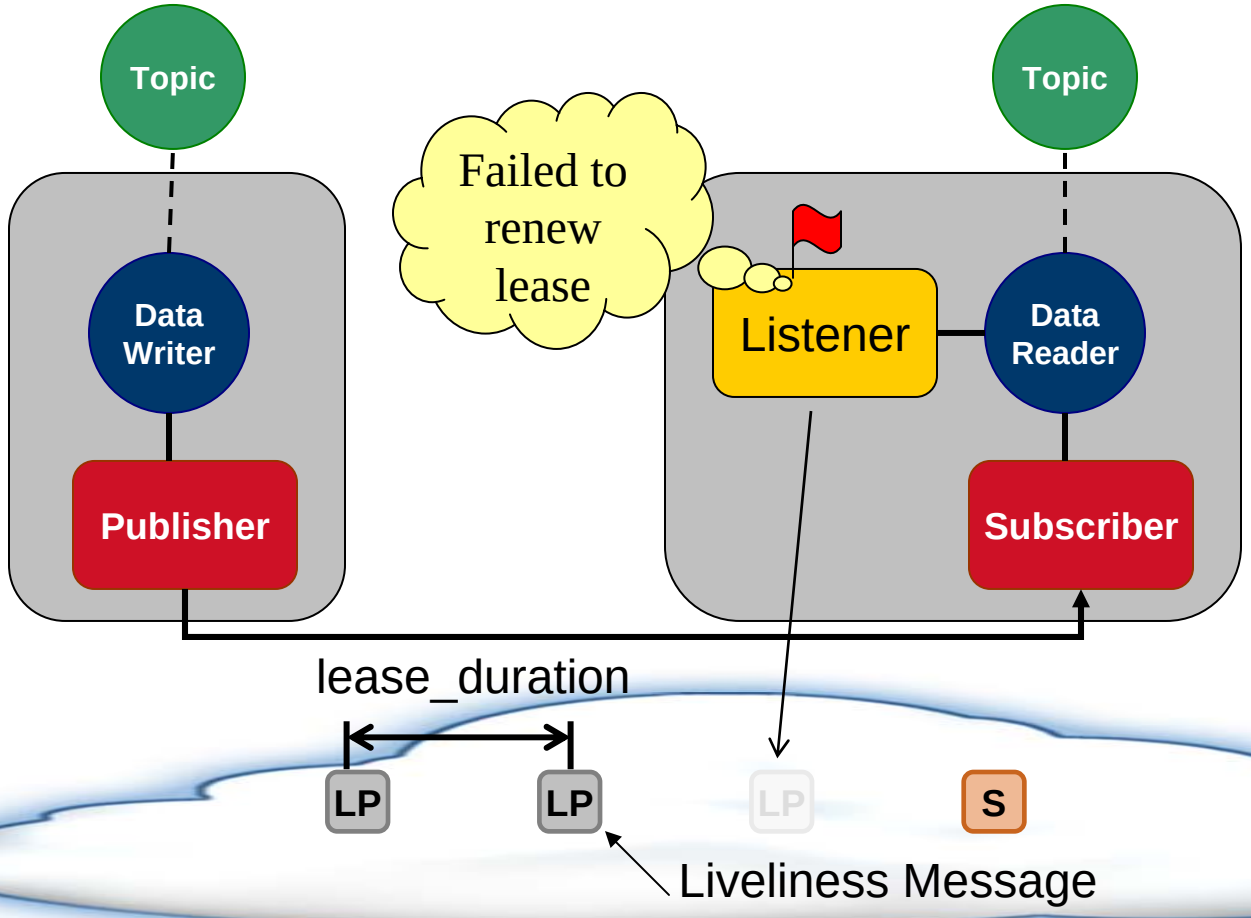


QoS: Liveliness – Type and Duration

Type: Controls who is responsible for issues of 'liveliness packets'

AUTOMATIC = Infrastructure Managed

MANUAL = Application Managed



Putting it All Together

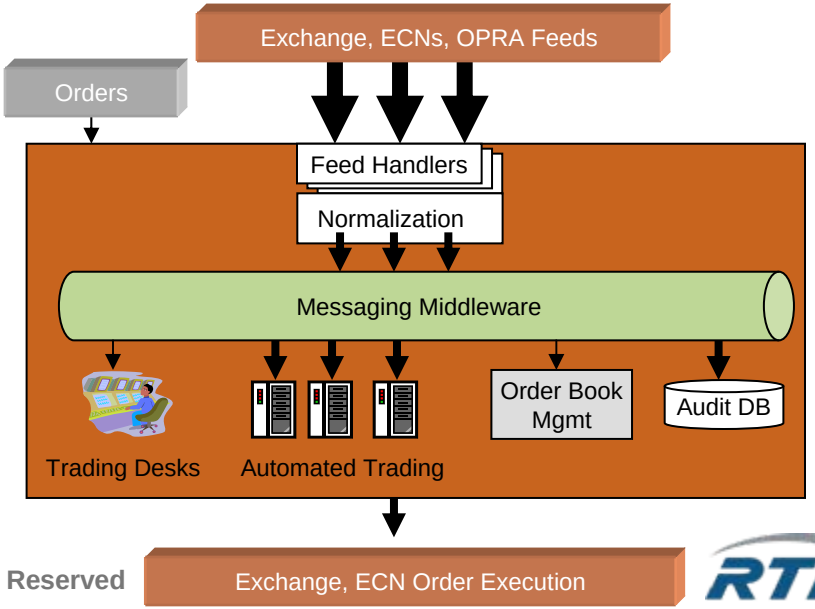
Scenario: Fire-control radar system, gets updates from multiple sensors up to 500x/sec

- Requirements
  - Radars
    - Multiple radars, may track same objects, but some higher performance/accuracy than others
  - Fire control
    - Needs every possible update quickly; last is best
    - Needs to know when a track is lost
  - Display console
    - Can only display 10Hz updates
  - Logger
    - Must record all information in a database for later analysis
- Quality of Service Settings
  - Radars
    - Key for each track
    - Ownership= Exclusive; Assign strength by accuracy
      - (each object tracked by best radar that sees it)
    - Publishers offer reliability
  - Fire Control
    - Maximize determinism
    - Best efforts; get every sample
    - Use key state/deadline to know when objects lost
  - Display console
    - Don't waste bandwidth
    - Best efforts
    - Time-based filter at 0.1sec
  - Logger
    - Reliable subscription

Application Example: Financial Trading

- Requirements
  - Feed handlers
    - Take direct market data
    - Publish 750k msgs/sec
  - Trading displays
    - Watch only “interesting” symbols
    - Can handle only 2Hz
  - Automated trading
    - Watch for transient price differences in specific symbols
    - 300 machines watching various symbols/aspects of market
  - Audit database
    - Save everything

- Quality of Service Settings
  - Feed handlers
    - Offer multicast, reliable
  - Trading displays
    - Subscribe at low bandwidth with time-based filters
  - Automated trading
    - Partition market data to divide load



Summary

- Designing a fault-tolerant distributed system is not a simple task
- A powerful middleware framework can provide a lot of value and help you focus on the business logic
- The middleware can save you a lot of time and effort.
  - It is worth learning how to use its power!

Questions

Gerardo Pardo-Castellote, Ph.D.
gerardo.pardo@rti.com