Real-Time Data Distribution Service (DDS) Tutorial

Workshop on Real-Time, Embedded and Enterprise-Scale Time-Critical Systems May 2010

The Real-Time Middleware Experts

Jens Pillgram-Larsen
Team Lead
jens@rti.com

DDS in a Nutshell

- DDS is Real-Time Data-Centric Networking Middleware
- DDS focuses on
  - Performance
  - Configurability (Quality of Service)
  - Scalability
  - Integration

DDS IS MUCH MORE THAN ENHANCED TCP!!!

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
  - UML Profile for DDS adopted June 2008
  - DDS for light weight CCM adopted 2008
  - Extensible and Dynamic Topic Types for DDS adopted 2010
- Standards under Development
  - Native Language C++ API for DDS
  - Web-Enabled DDS
  - Java 5 PSM

Agenda

- Examples of DDS Usage
- Benefits of DDS
- The DDS Model

DDS Mandated for Data-Distribution

- DISR (formerly JTA)
  - DoD Information Technology Standards Registry
- US Navy Open Architecture
- FCS SOSCOE
  - Future Combat System – System of System Common Operating Environment
- *SPAWAR NESI*
  - *Net-centric Enterprise Solutions for Interoperability*
  - *Mandates DDS for Pub-Sub SOA*

DDS Adoption – Aerospace & Defense

Boeing
AWACS

E2C Hawkeye

Boeing
Future Combat
Systems

Raytheon
SSDS

Lockheed
AEGIS

Insitu
Unmanned
Air Vehicles

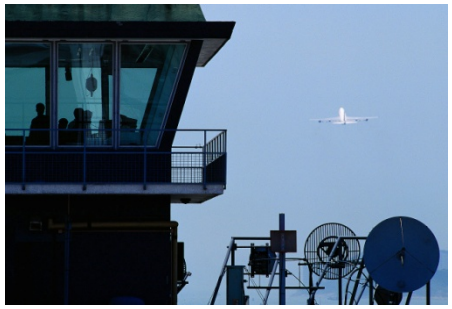
DDS Adoption – Transportation, Industrial

WiTronix
Train and vehicle Tracking

Schneider Electric
Industrial Automation

Tokyo Japan
Traffic Control

Kuka
Robotics



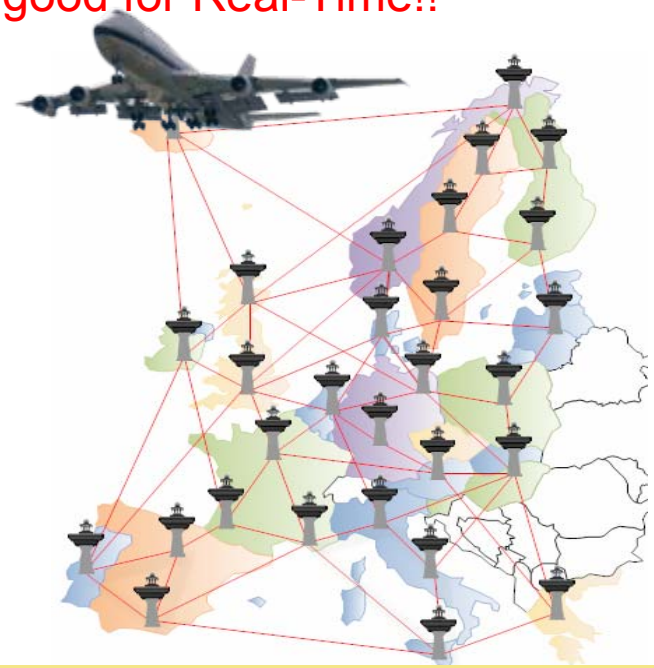
EU and US
Air Traffic Management

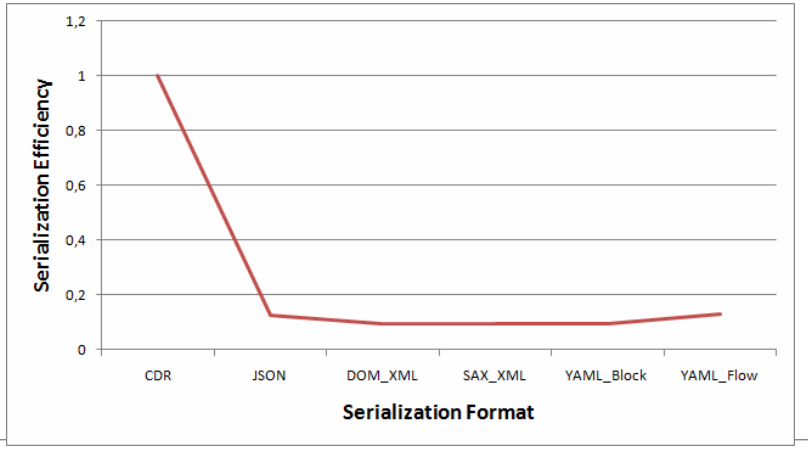
Varian
Medical Instruments

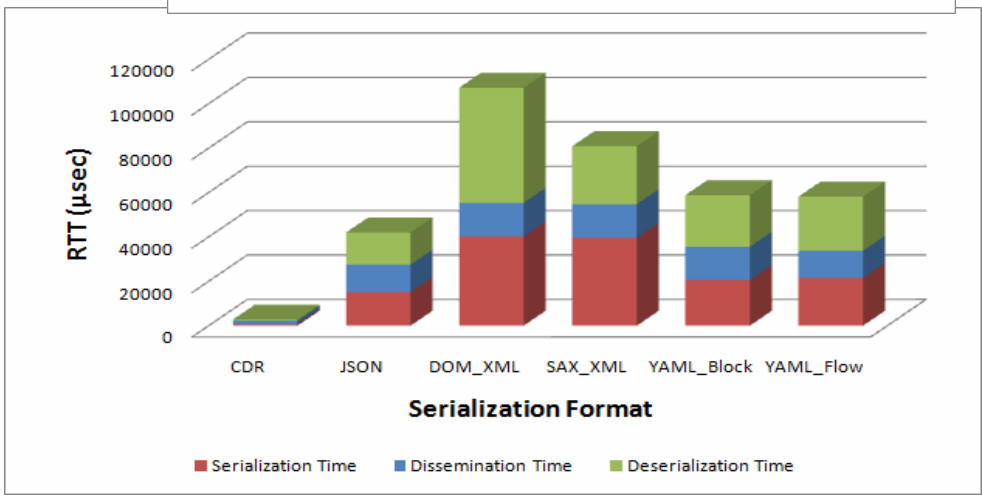
Study on impact of WS technologies for future European ATC: XML is not suitable for European Flight Data Distribution

- Data size explodes 10X vs. CDR
  - Flight Plans go from 100KB to 1MB
- Communication speed drops 20X

Not good for Real-Time!!







Source: Christian Esposito and Domenico Cotroneo, Dario Di Crescenzo.
SELEX-SI/Consorzio SESM/University of Naples.
"Flexible Communication Among DDS Publishers and Subscribers"
July 2008, Real-Time Systems Workshop, Washington, DC

Spacecraft Launch Control

- The NASA KSC Constellation program will be the next generation of American manned spacecraft.
- DDS delivered 300k instances, at 400k msgs/sec with 5x the required throughput, at 1/5 the needed latency

DDS connects thousands of sensors and actuators

Real-Time Pre-Trade Compliance

- DDS value: replication between compliance servers -- no single point of failure
- Driver: enforce and monitor regulatory and client-imposed pre-trade investment restrictions
- Integrate proprietary compliance engine with trading system

System went live less than 3 months after licensing

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

DDS bridges high speed networking to the CAN bus

Agenda

- Examples of DDS usage
- Benefits of DDS
- The DDS Model

Main Advantages of DDS

- 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 Data-Centric vs. Message-Centric Design

- Anyone can publish and subscribe
  - DDS
  - JMS
  - AMQP
  - WS-Eventing
  - REST-MS
- …but different technologies have very different models

  Treating them as interchangeable has ramifications:
  - Increased cost, risk
  - Decreased return on technology investment
  - Decreased performance

Data-Centric vs. Message-Centric Design

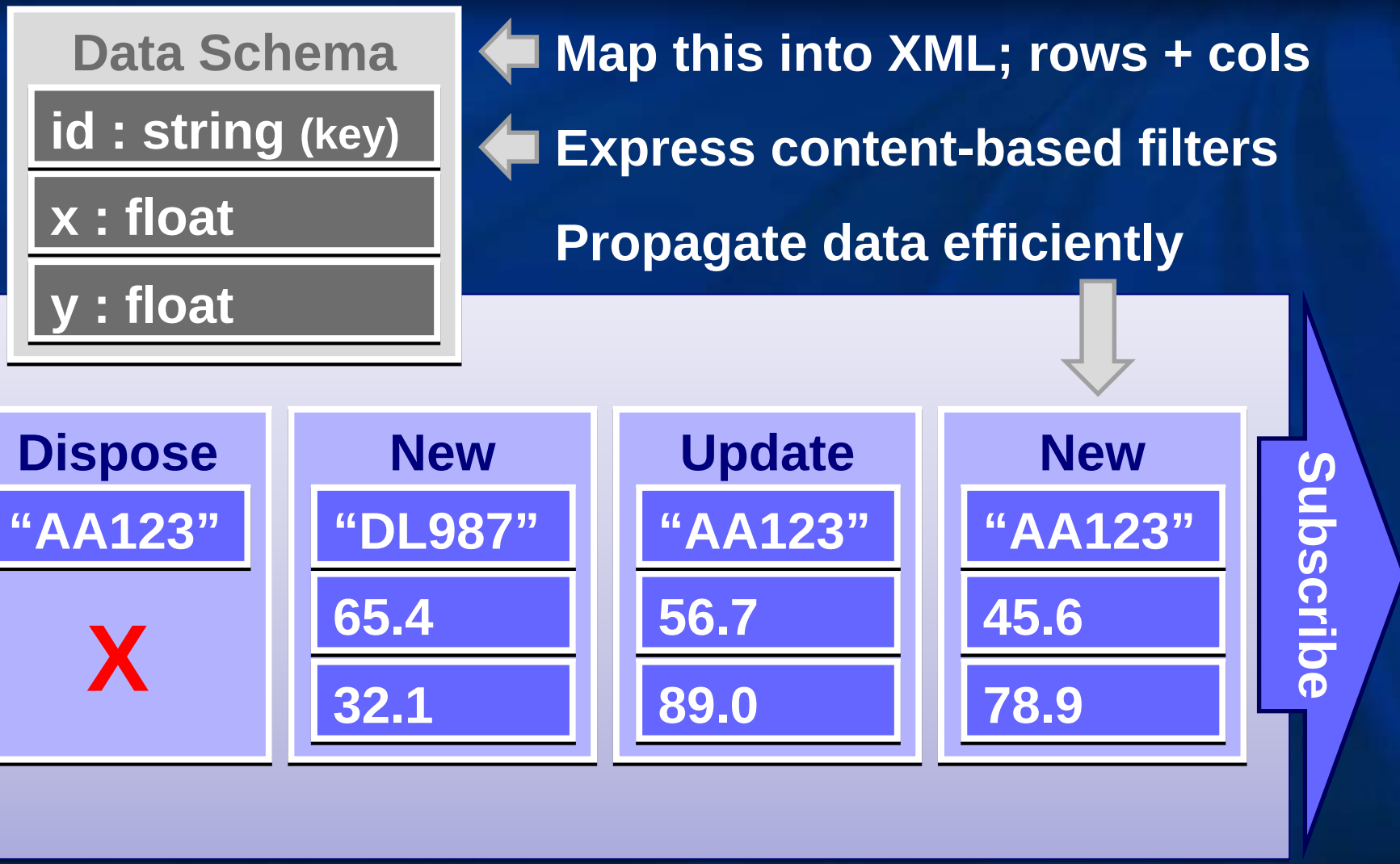
Data-Centric

- Infrastructure ***does*** understand your data
  - What data schema(s) will be used
  - Which objects are distinct from which other objects
  - What their lifecycles are
  - How to attach behavior (e.g. filters, QoS) to individual objects
- Example technologies:
  - DDS API
  - RTPS (DDSI) protocol

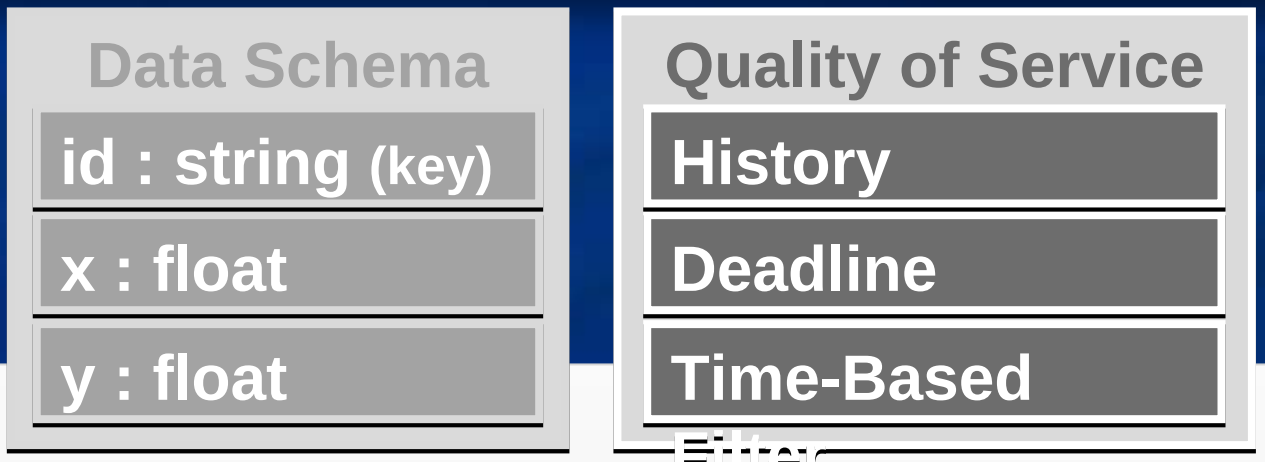
Message-Centric

- Infrastructure ***does not*** understand your data
  - Opaque contents vary from message to message
  - No object identity; messages indistinguishable
  - Ad hoc lifecycle mgmt
  - Behaviors can only apply to whole data stream
- Example technologies:
  - JMS API
  - AMQP protocol

Example: Data-Centric Track Data



Example: Data-Centric Track Data

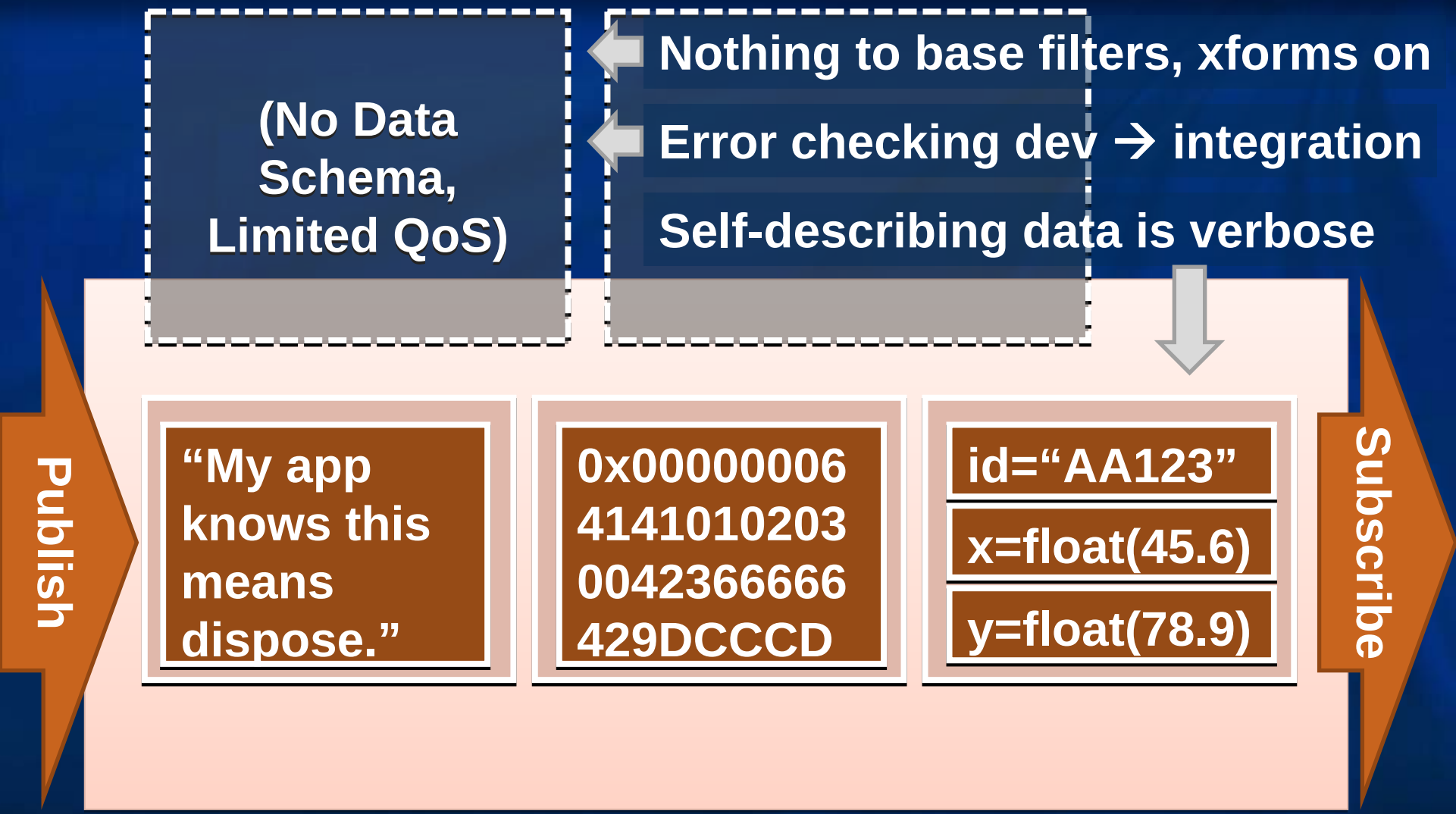


Publish

Subscribe

- Once infrastructure understands objects, can attach **QoS contracts** to them
- "Keep only the latest value" or "I need updates at this rate" **make no sense unless per-object**
  - Flight AA123 updates shouldn't overwrite DL987, even if AA123 is updated more frequently
  - Update rate for one track shouldn't change just because another track appeared

Example: Message-Centric Track Data



When Message-Centric Design Works Well

(Example: Securities order processing system)

- **No notion of objects or state** beyond individual message; e.g. “Buy 12 shares IBM @ $12”
- **No need to filter based on content**; e.g. all orders need to be processed eventually
- **No need for real-time QoS**; e.g. maybe to you, “real-time” just means “fast”
- **Messaging interface is not integration interface** between systems, providers, versions: one team implements both sides of interaction at same time

When Data-Centric Design Works Well

(Examples: Distribution of track data, weather data)

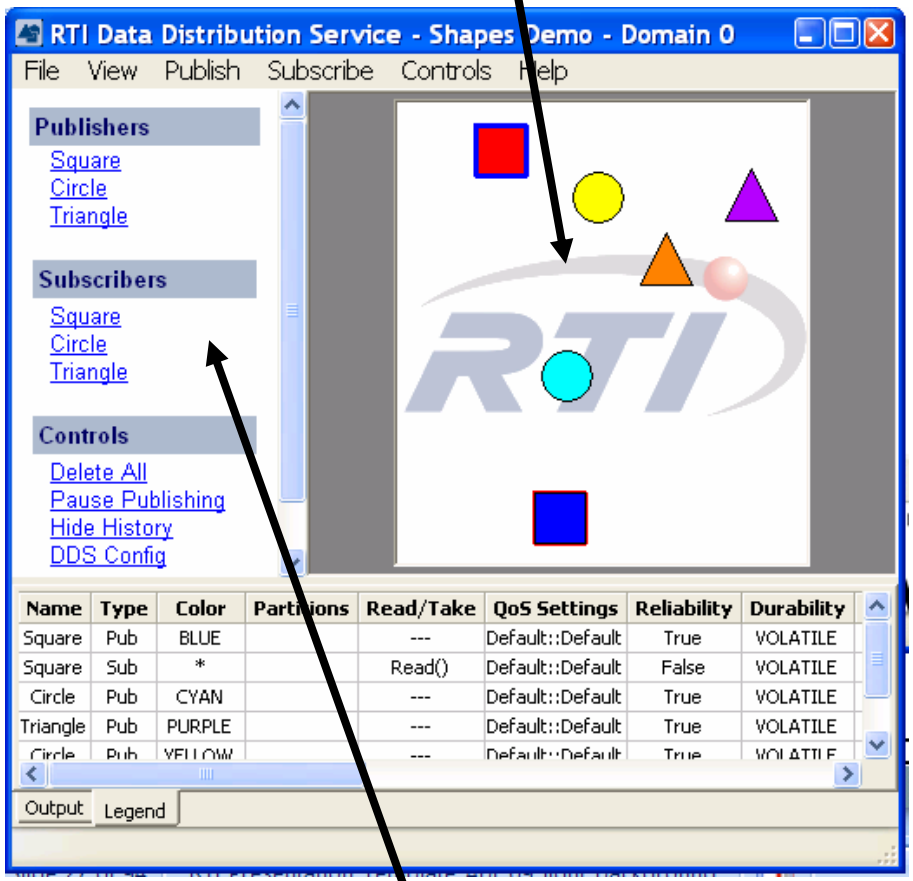
- **Object lifecycle spans multiple updates**;
  e.g. “Track AA123” or “Weather at (45.6°, 78.9°)”
- **Topic-per-object is impractical** because
  - …objects are too numerous and/or
  - …their identities are unknown *a priori* and/or
  - …commonalities make them more manageable as a group.
  - Independent topic for each of 10K tracks?
    For each (latitude, longitude) tuple?
- **Need data-aware filtering and/or QoS enforcement**; e.g.
  - “Give me the current state of all tracks” or
  - “Show me a weather map within this geographical region”
- **Integrating independently developed components**
  and/or systems

Don’t Confuse Architecture and Technology

- Can implement message-centric design with data-centric technology
  - Use generic data schemas (e.g. an opaque binary buffer)
  - Don’t define QoS contracts
- Can implement data-centric design with message-centric technology
  - Build layers on top to handle data schemas, data caching, QoS definition and enforcement, discovery, etc.
  - Capture service definitions informally in documentation
- **Define data/service architecture first**, *then* select appropriate technology

Demo: DDS Concepts

Display Area:
Shows state of objects

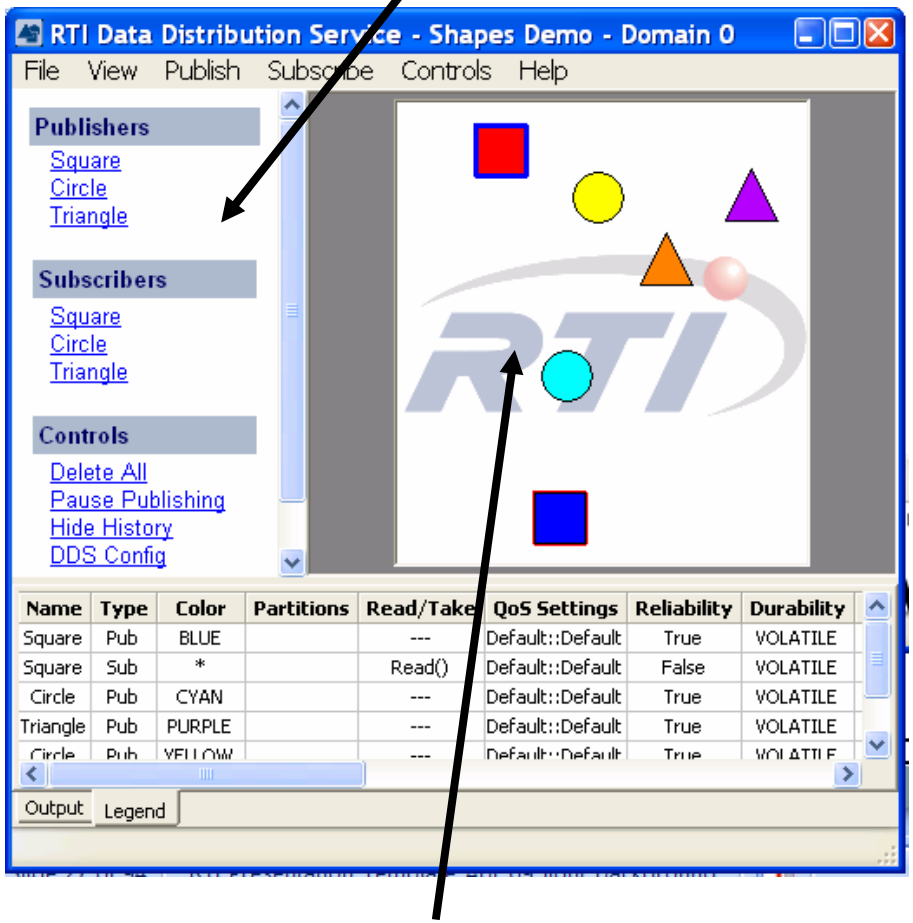


Control Area:
Allows selection of objects and QoS

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

Demo: Quality of Service (QoS)

Writers and readers state their needs



DDS delivers

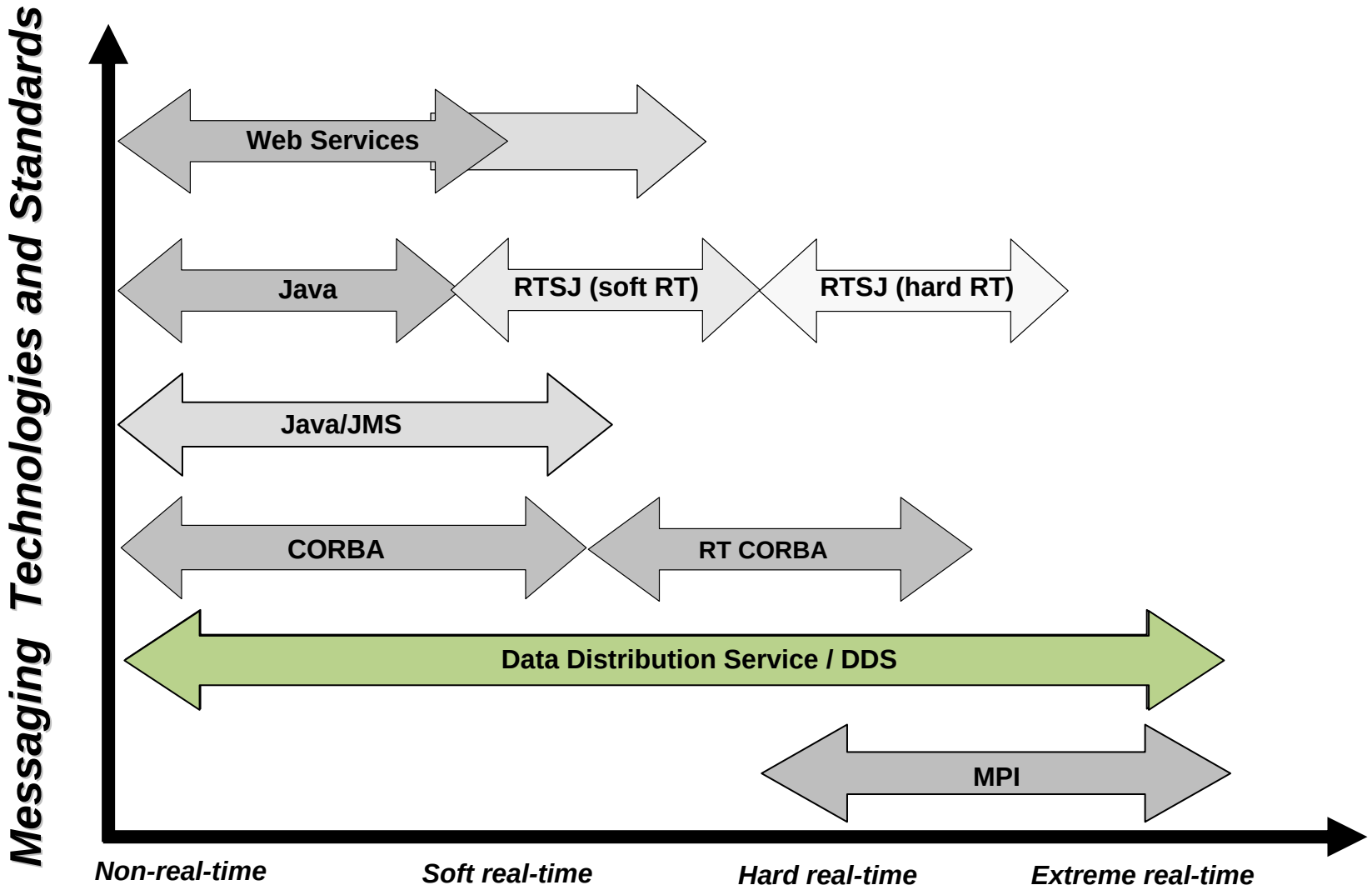
- Topics
  - Square, Circle, Triangle
- Data types (schemas)
  - Shape (color, x, y, size)
    - Color is instance Key
- QoS
  - Deadline, Liveliness
  - Reliability, Durability
  - History, Partition
  - Ownership

#2 Performance & Scalability

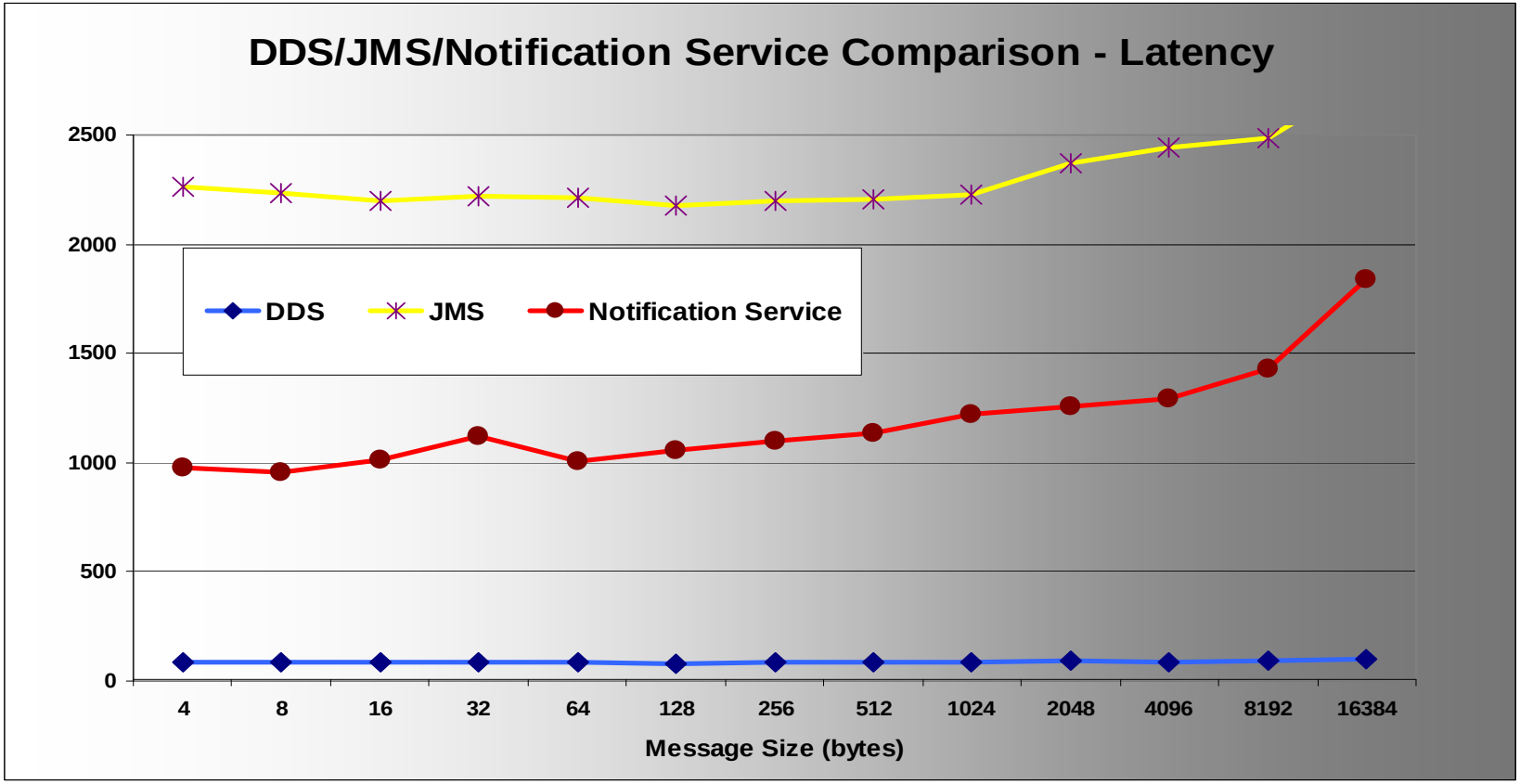
DDS was designed to support high performance

- DDS uses high-performance data-access APIs
  - Read data by array (no additional copies)
  - Buffer loaning for zero copy access
- DDS Supports advanced features such as:
  - Message prioritization (via latency budget QoS)
  - Network prioritization (via transport priority QoS)
  - Source filtering (via Content-based and Time-based filters)
- DDS does not require the presence of intermediate brokers
  - Applications can communicate directly peer-to-peer
- DDS Protocol supports UDP, multicast and reliable multicast
  - Multicast can be used for high-performance scalability
  - Use of UDP avoids head-of-line blocking
  - Best efforts can be used for repetitive time-critical data

Data-Distribution and Real-Time



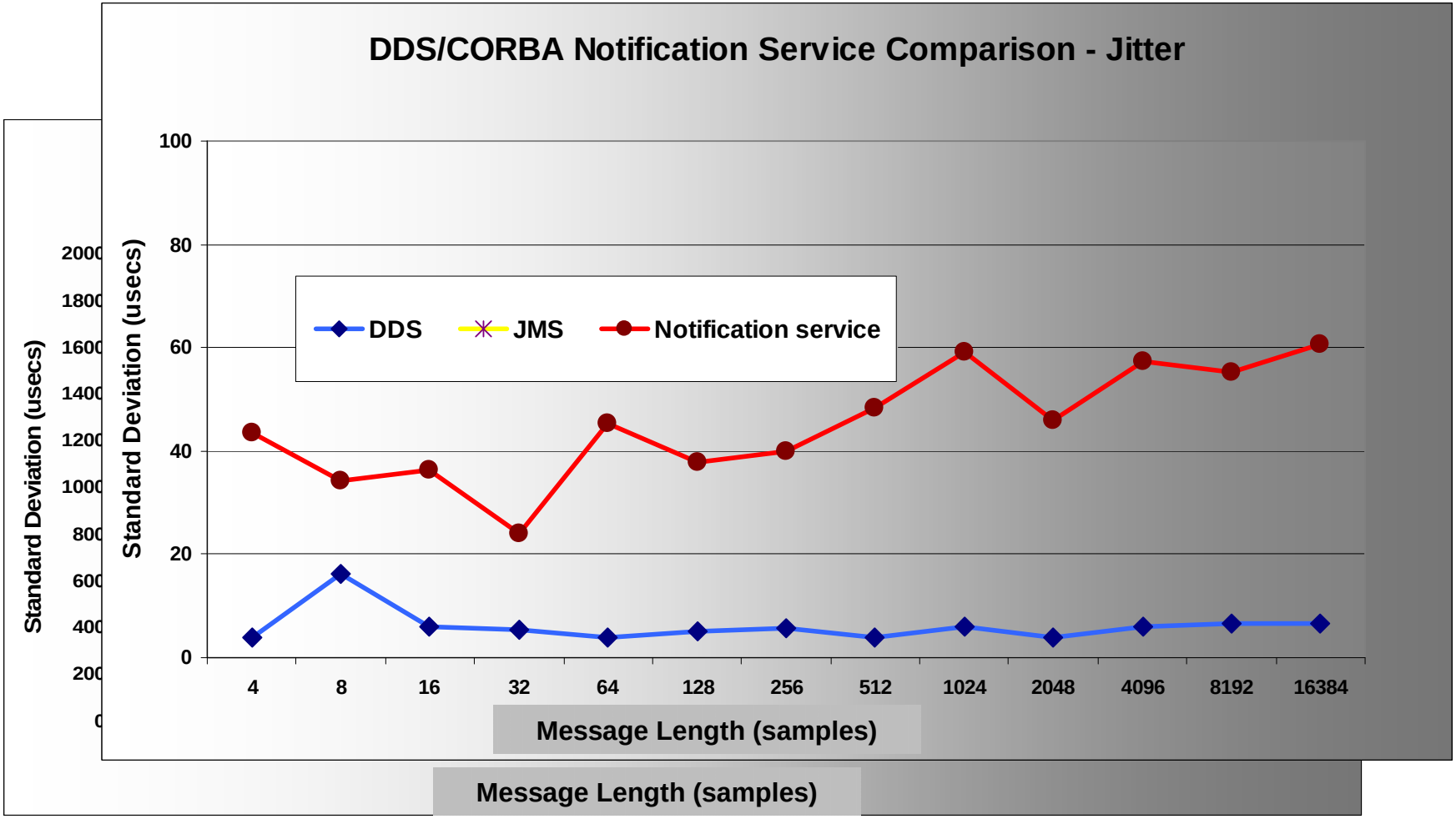
Latency – (Linear Scale)



Message Length (samples)

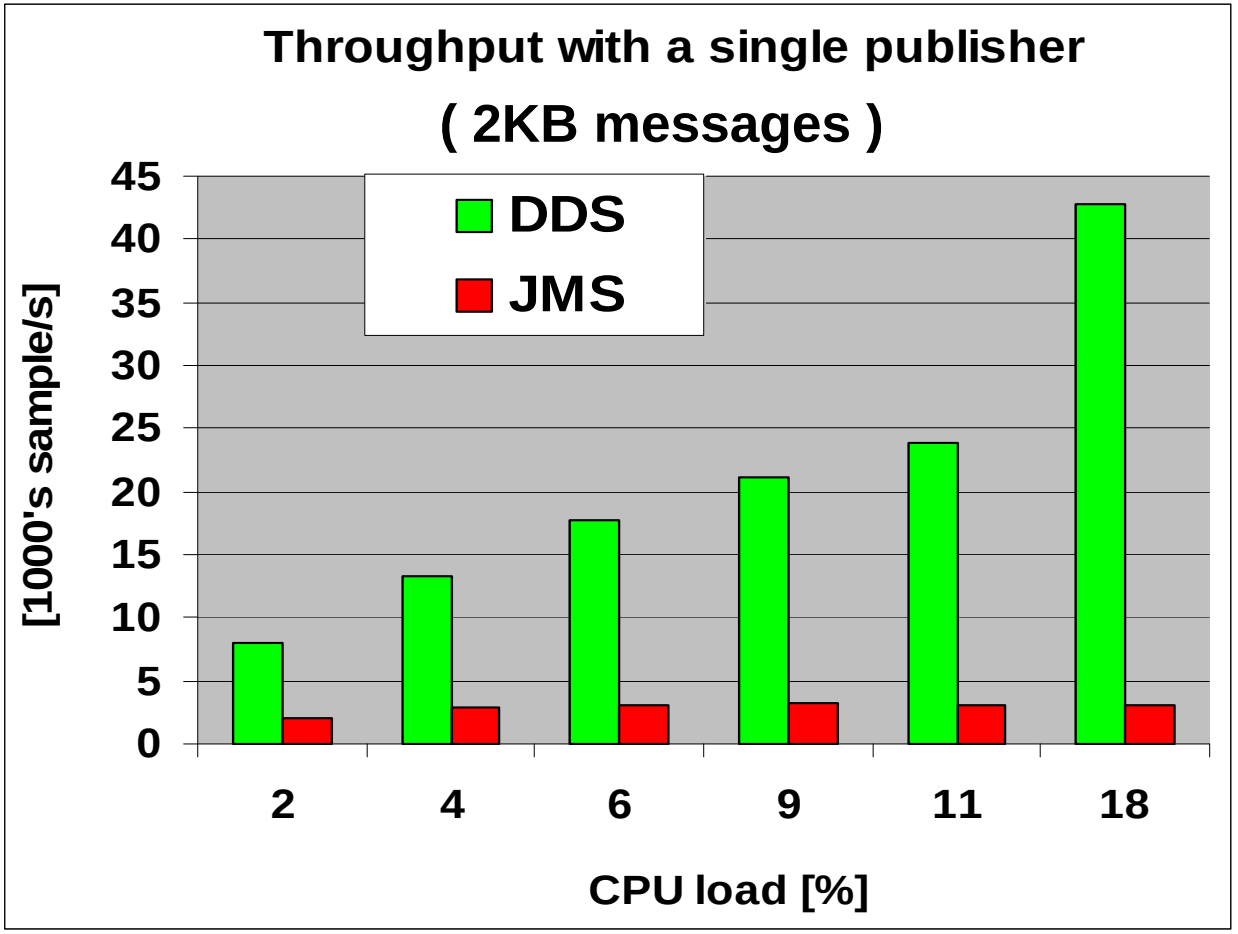
Adapted from Vanderbilt presentation at July 2006 OMG Workshop on RT Systems

Jitter – (Linear Scale)



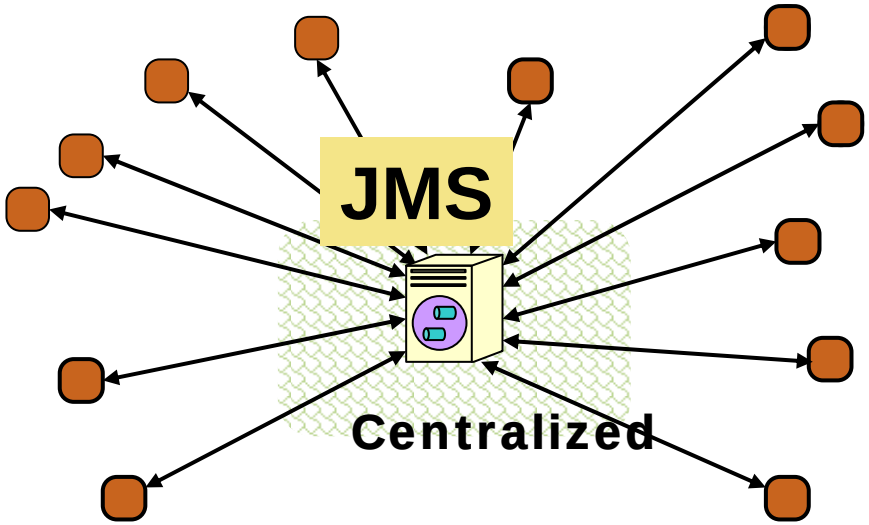
Source: Vanderbilt presentation at July 2006 OMG Workshop on RT Systems

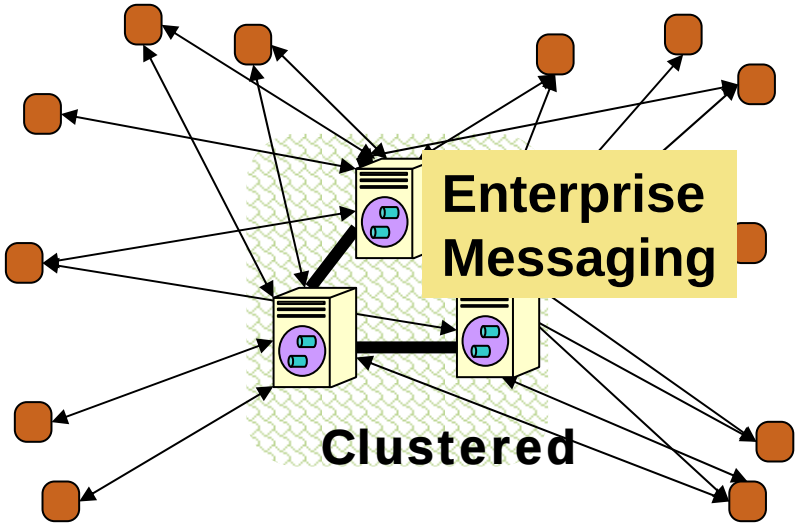
DDS compared to JMS

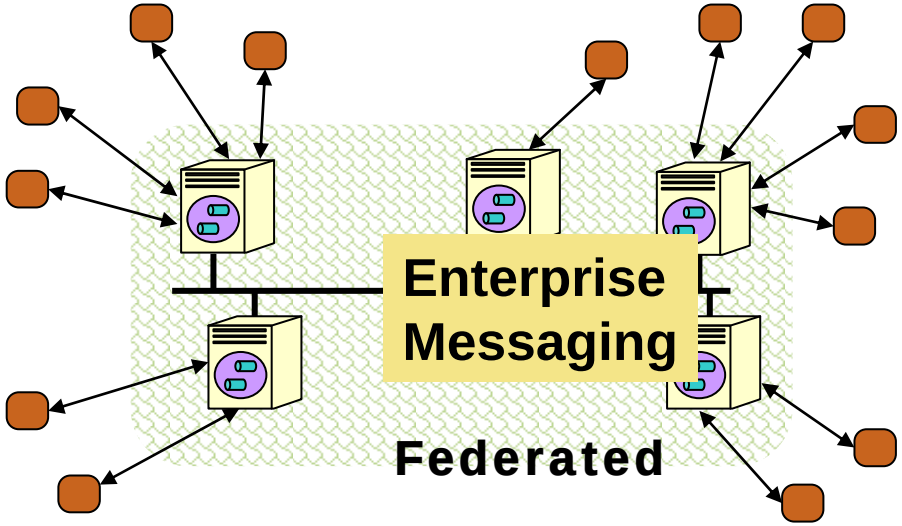


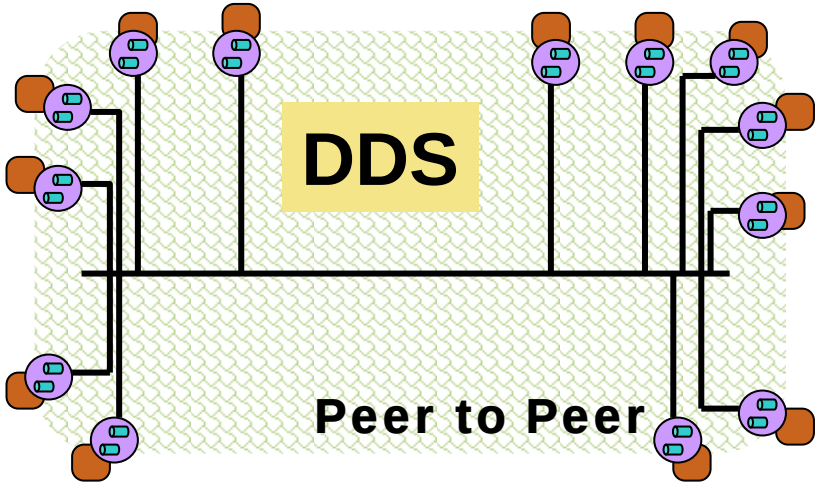
Platform: Linux 2.6 on AMD Athlon, Dual core, 2.2 GHz

Message bus architectures









#3 Powerful Services

Included in the Standard:

- Discovery
- Ownership, Redundancy & Failover
- Persistence (Durable) Data
- Last value and historical caches

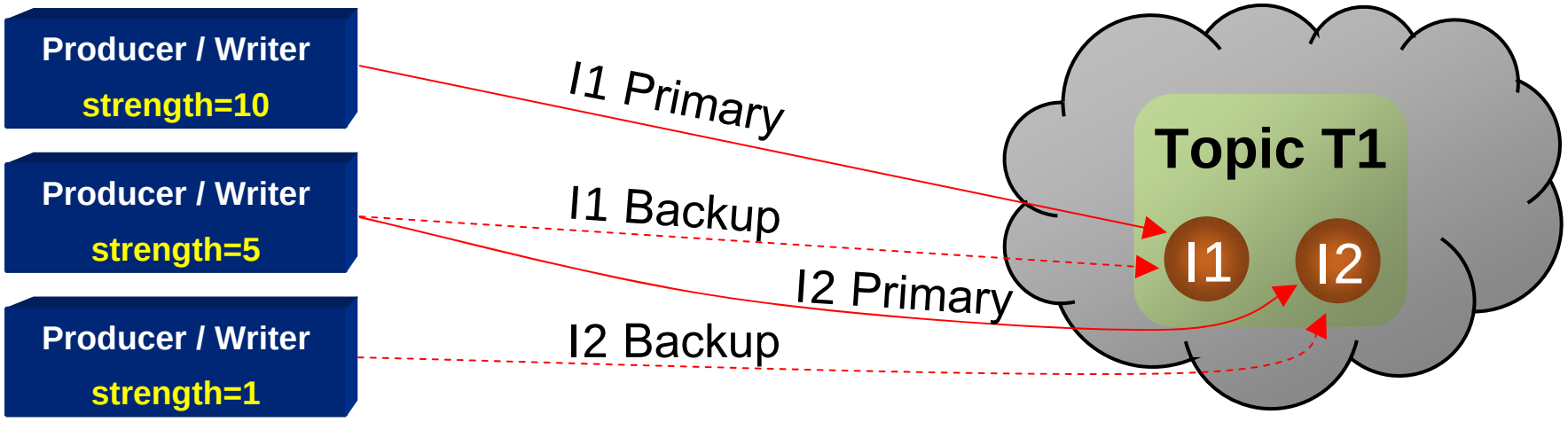
Enabled by the DDS protocol:

- Recording Service
- Database Service
- Web-Integration Service

Discovery Service

- DDS provides the means for an application to discover other participants in the Domain
  - And the Topics they Publish and Subscribe
  - And the Quality of Service of the remote endpoints
- A participant can determine who it is communicating with…
  - And selectively decide who to communicate with…

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

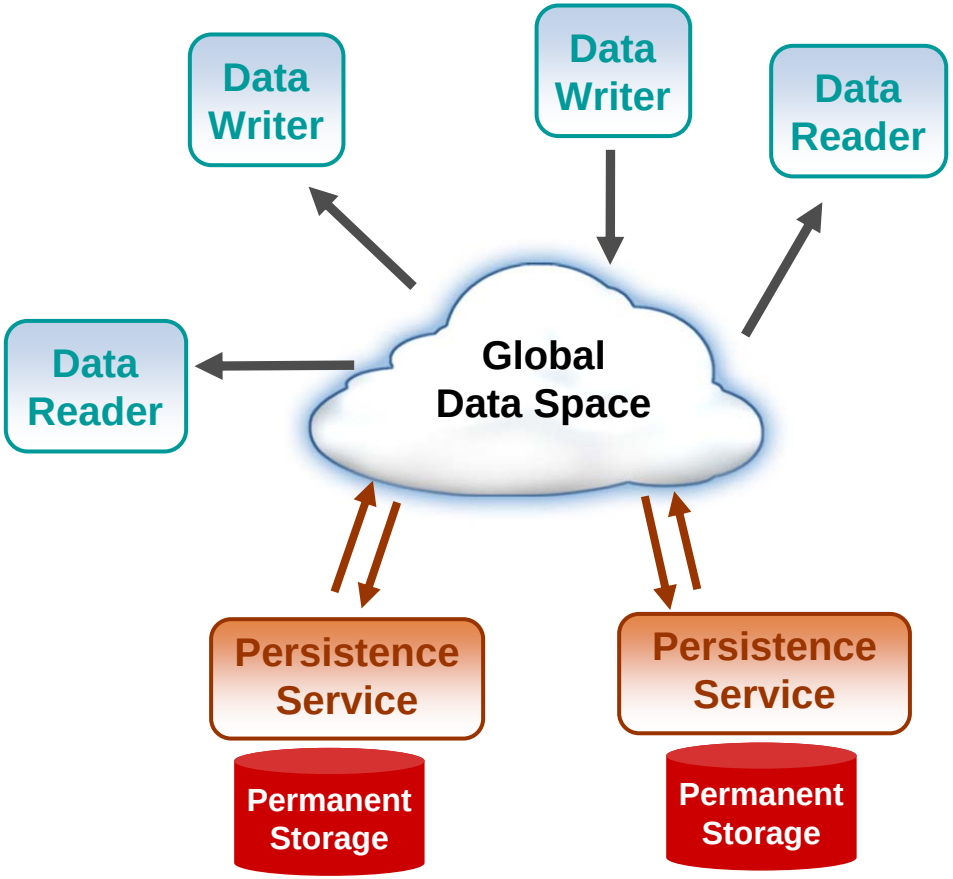
- Shared Ownership allows any writer to update the subject

Data Persistence

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

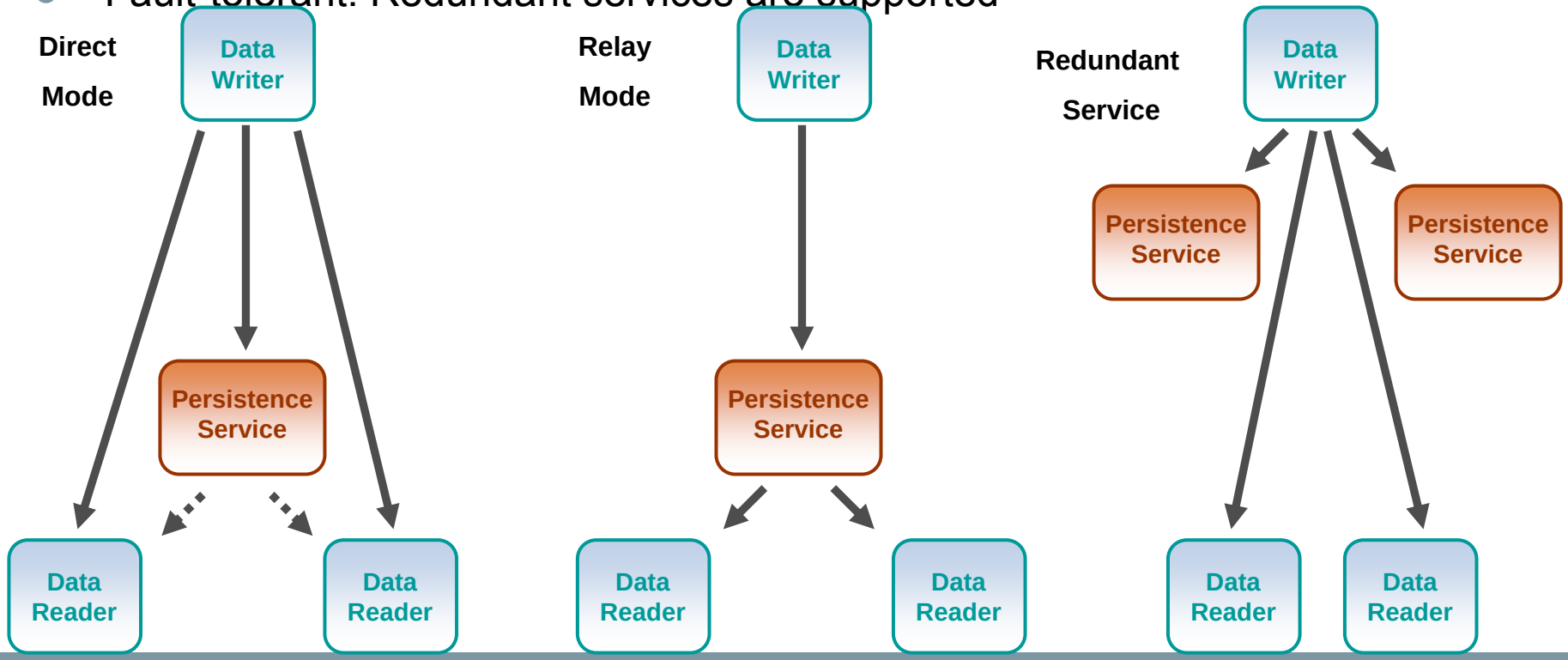
Can be configured for:

- **Redundancy**
- **Load balancing**



Persistent Data Service

- Modes:
  - High-performance “direct” model. Data flows in directly to subscribers and to the persistent service
  - Transactional “relay” mode persists first, then sends to subscribers
- Fault-tolerant: Redundant services are supported



Last value cache

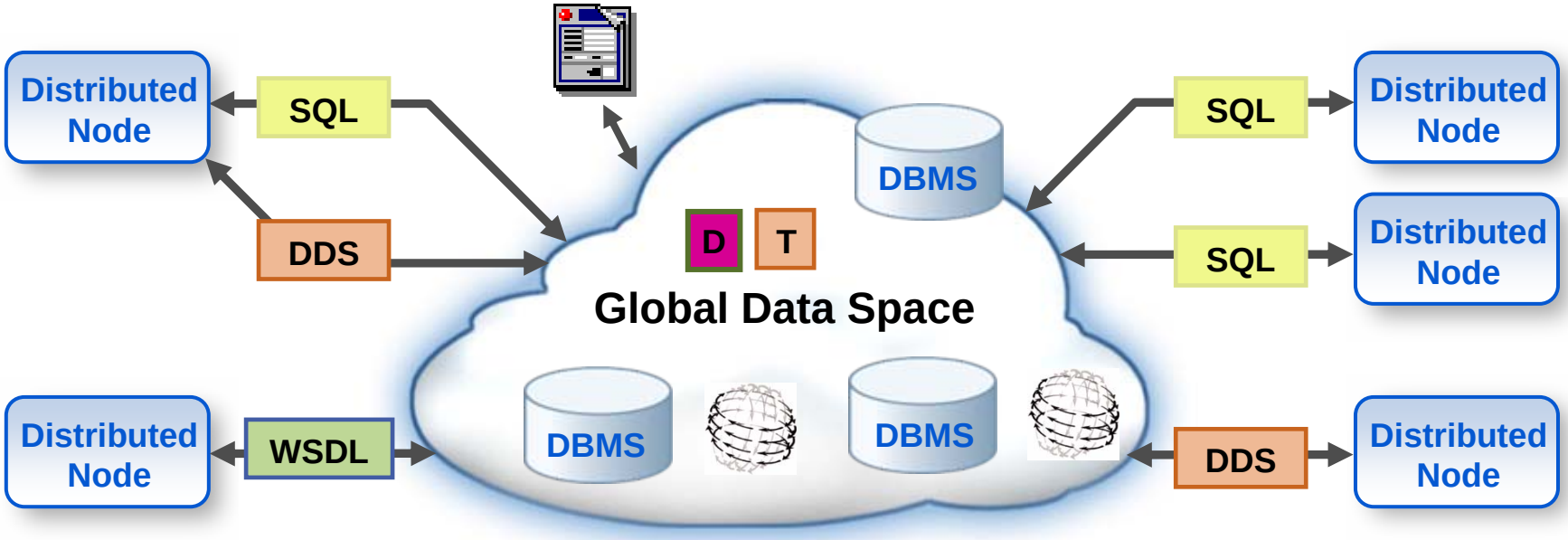
- A last-value cache is already built-in into every Writer in the system
  - Can used in combination with a Durable Writer
- A late joiner will automatically initialize to the last value
- Last value cache can be configure with history depth greater than 1
- The Persistence Service can be used to provide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

Historical cache

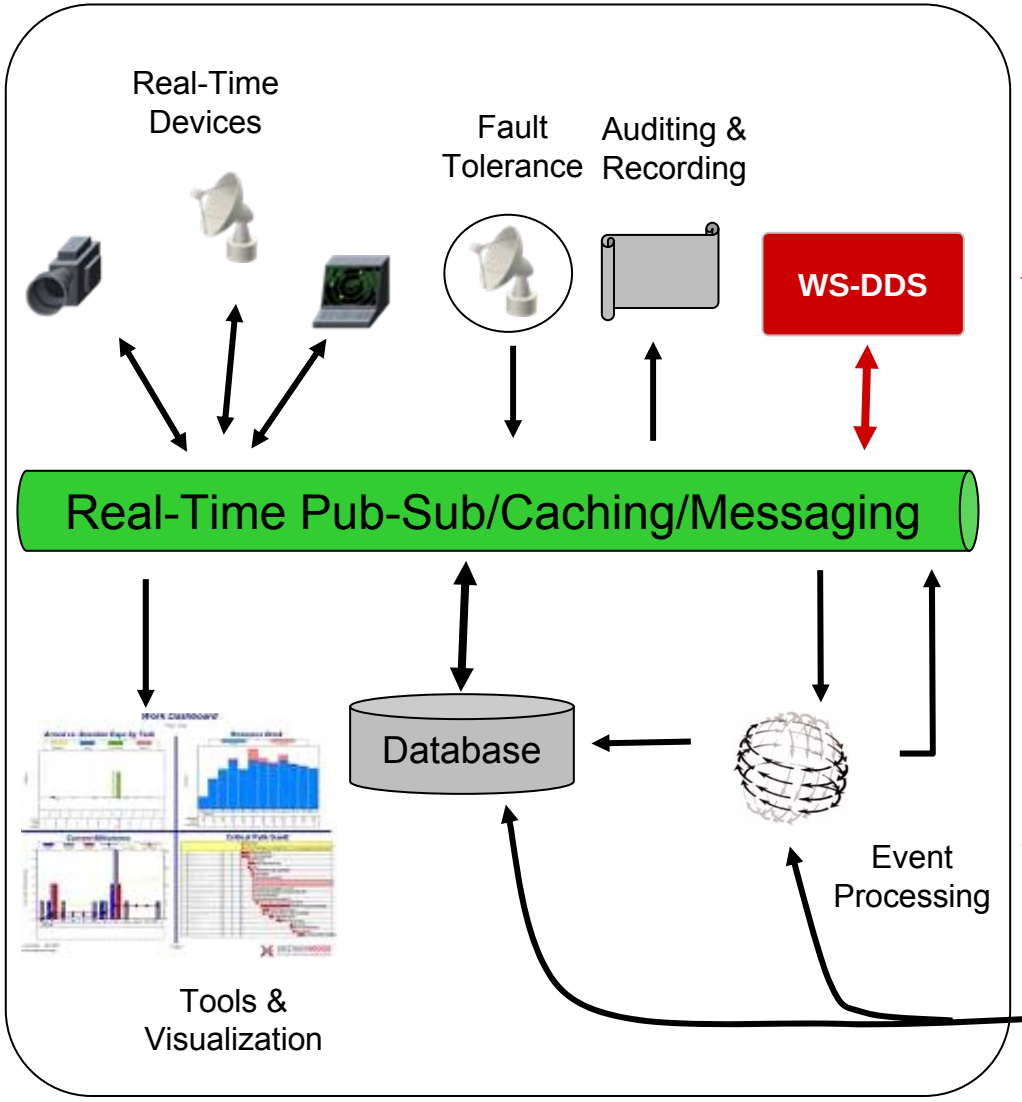
- A partial historical cache is already built-in into every Writer in the system
  - Can used in combination with a Durable Writer
- The Persistence Service can be used to provide a historical cache with larger/unlimited depth
- A late joiner will automatically initialize to the desired history
  - Currently 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

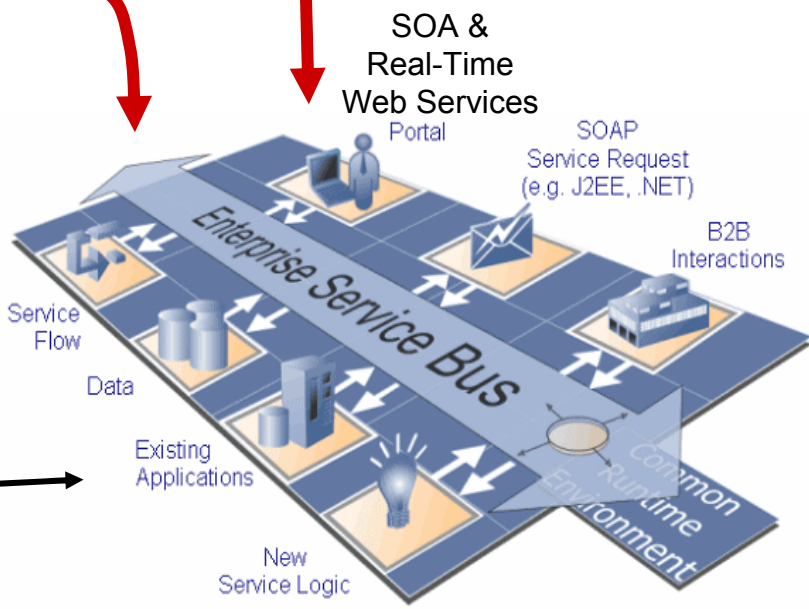
#4 Interoperability

- 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

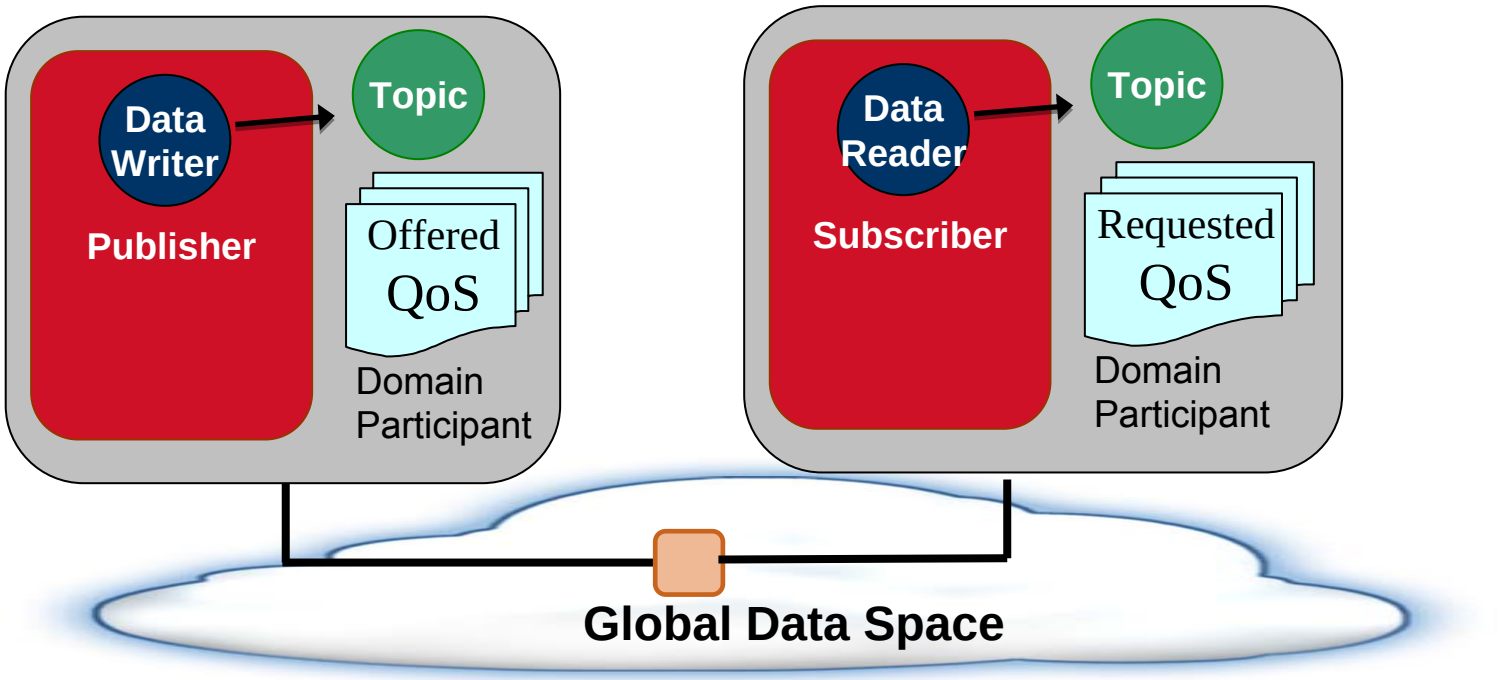




Agenda

- Examples of DDS usage
- Main Benefits of DDS
- The DDS Model

DDS 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

DDS Type Model

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*
   - *TypeCode*
3. **Data Representations**: languages for describing data
   - *CDR*
   - *XML*

DDS Type Model

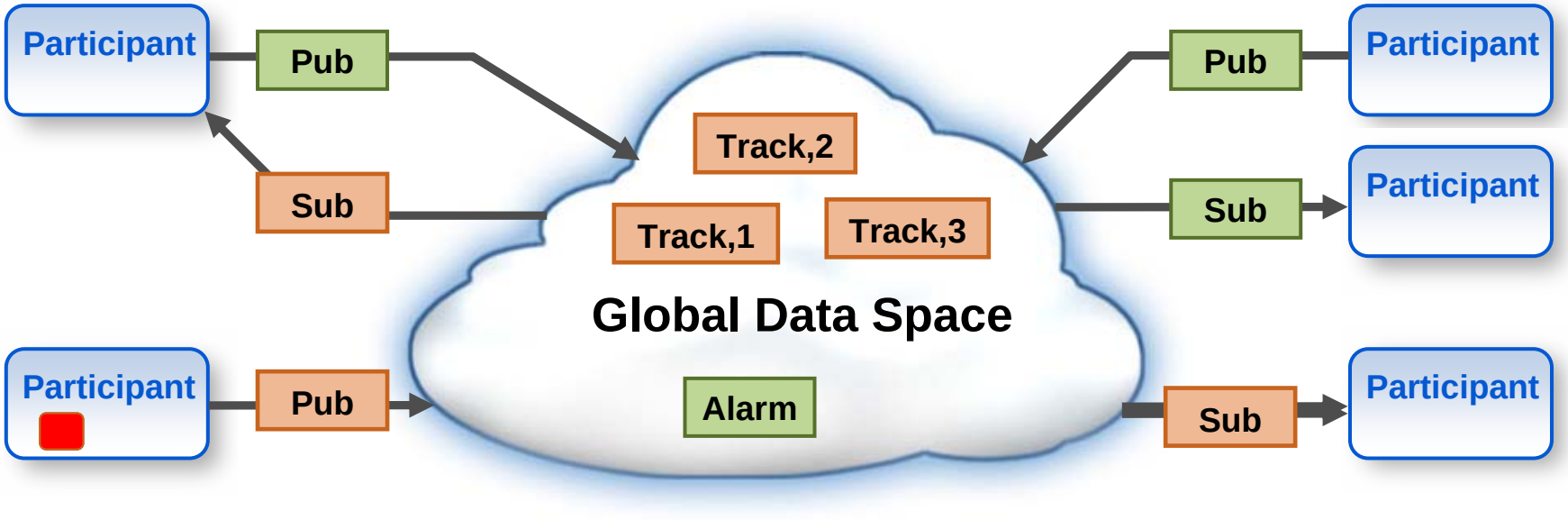
4. **Language Binding**: programming APIs
   - *"Plain Language"*: extension of existing IDL-to-*language* bindings
   - *Dynamic*: reflective API for types and objects, defined in UML (conceptual model) and IDL (API)

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*

DDS Communication Model

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

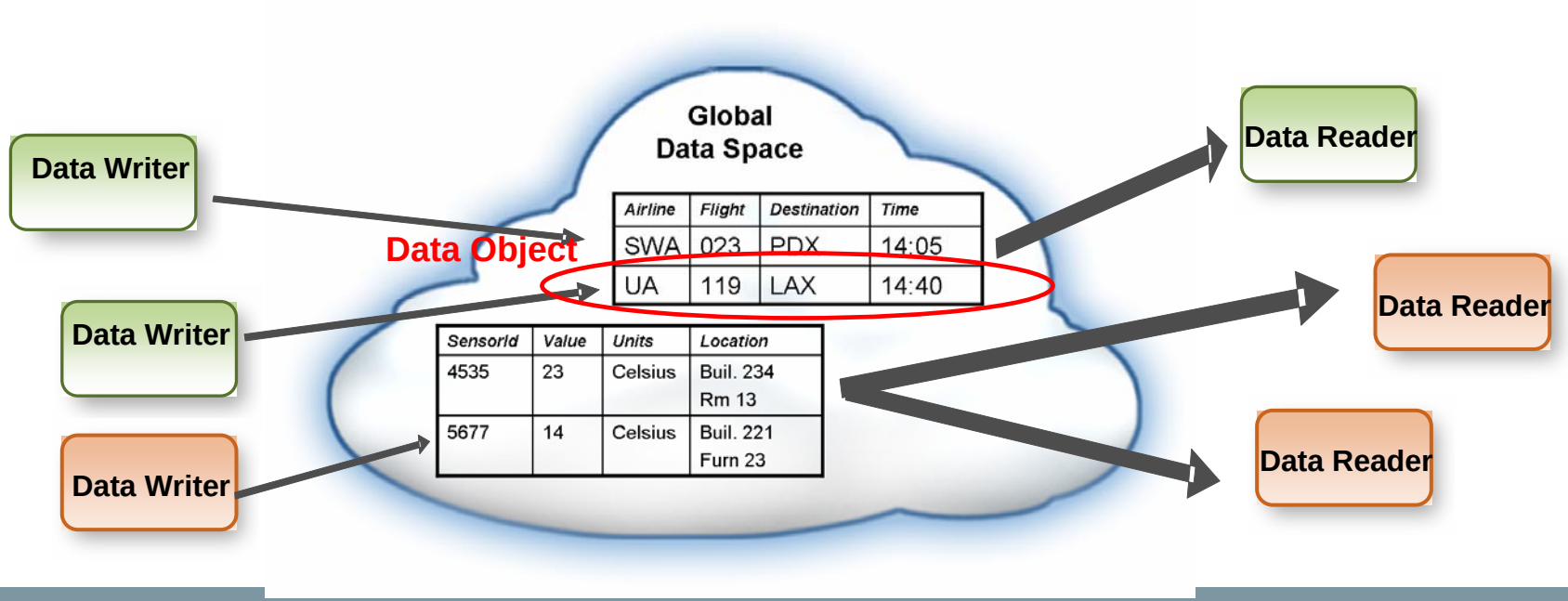
- Data objects addressed by **Domain, Topic** and **Key**
- Subscriptions are **decoupled** from Publications
- Contracts established by means of **QoS**
- Automatic **discovery** and **configuration**



DDS Data-Centric Model

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

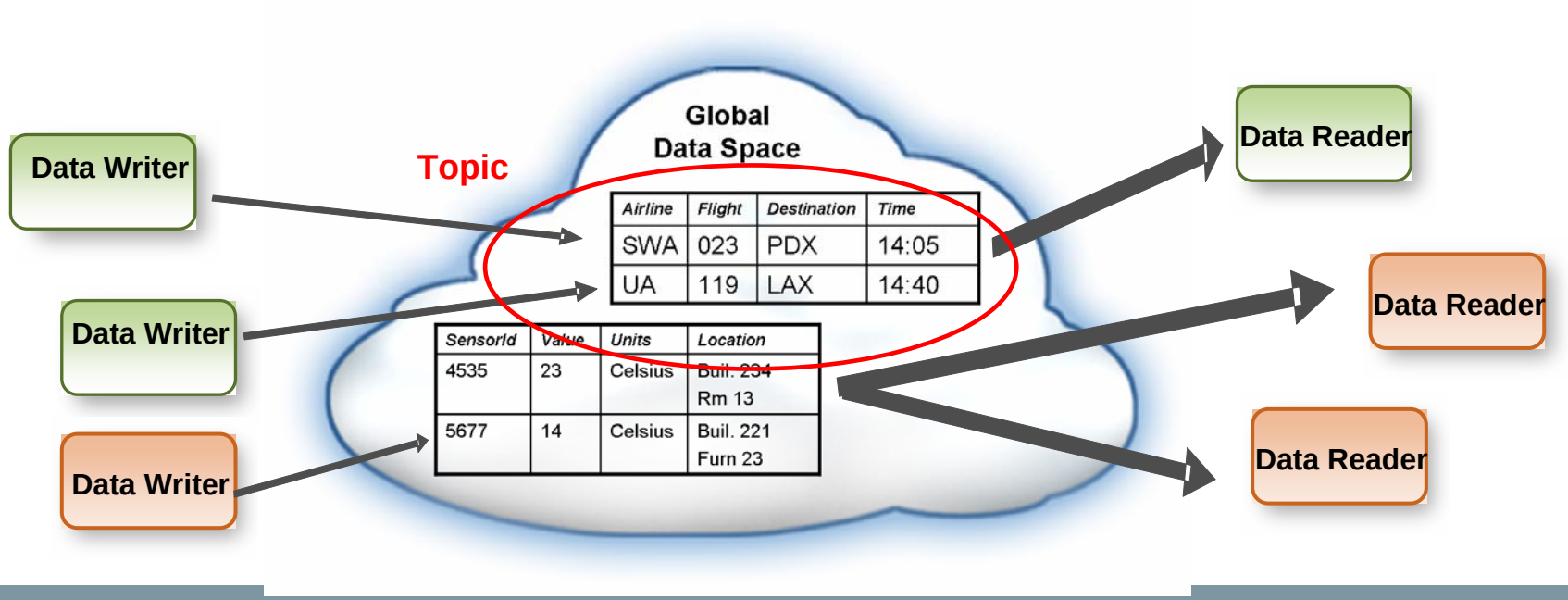
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a "x.y.z …" formatted string



DDS Data-Centric Model

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

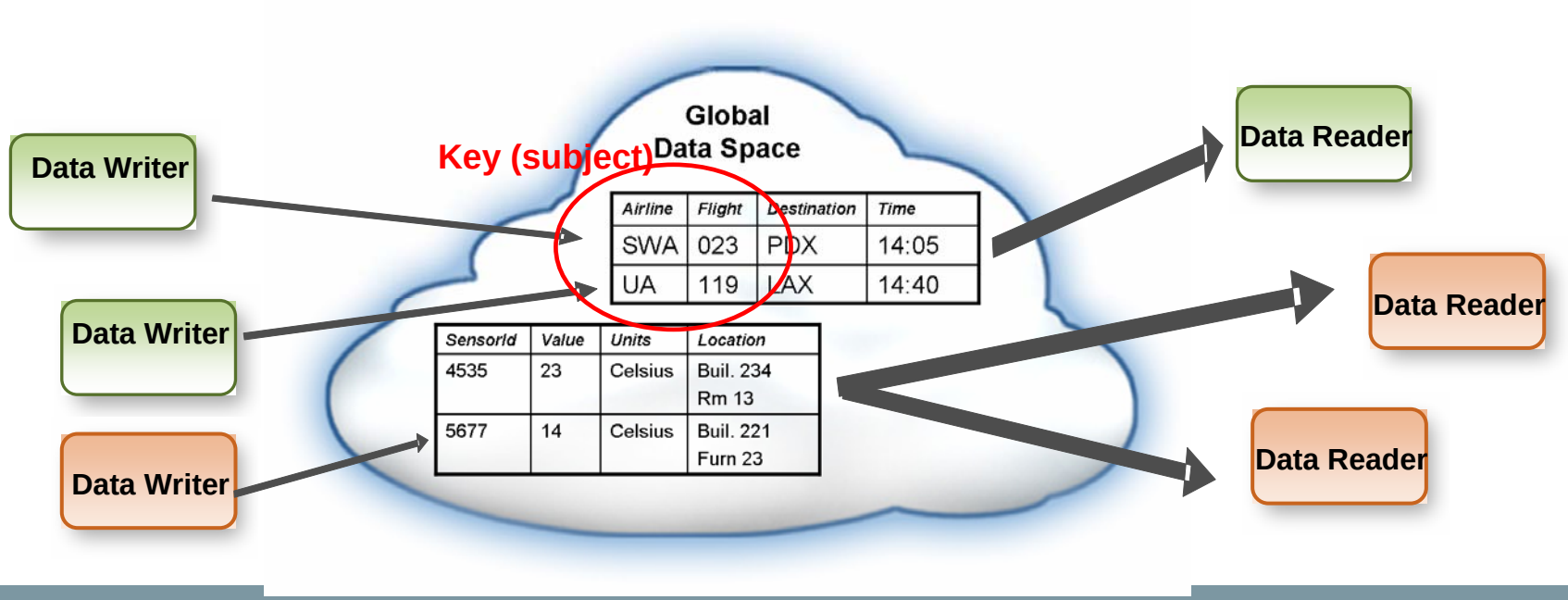
- **Domains** provide a level of isolation
- **Topic** groups homogeneous subjects (same data-type & meaning)
- **Key** is a generalization of **subject**
  - **Key** can be any set of fields, not limited to a "x.y.z …" formatted string



DDS Data-Centric Model

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

- **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



DDS Data-Centric Model

- DDS is Pub-Sub of **data objects**
  - The object is strongly typed
  - DDS publishes the state of the data object
  - DDS data objects have an **identity**

A DDS Sample is more than a Message

DDS 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

Q/A

Thank You!