Unifying the Global Data Space using DDS and SQL

OMG RT Embedded Systems Workshop

13 July 2006

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

Fernando Crespo Sanchez
fernando.crespo@rti.com

www.rti.com

Outline

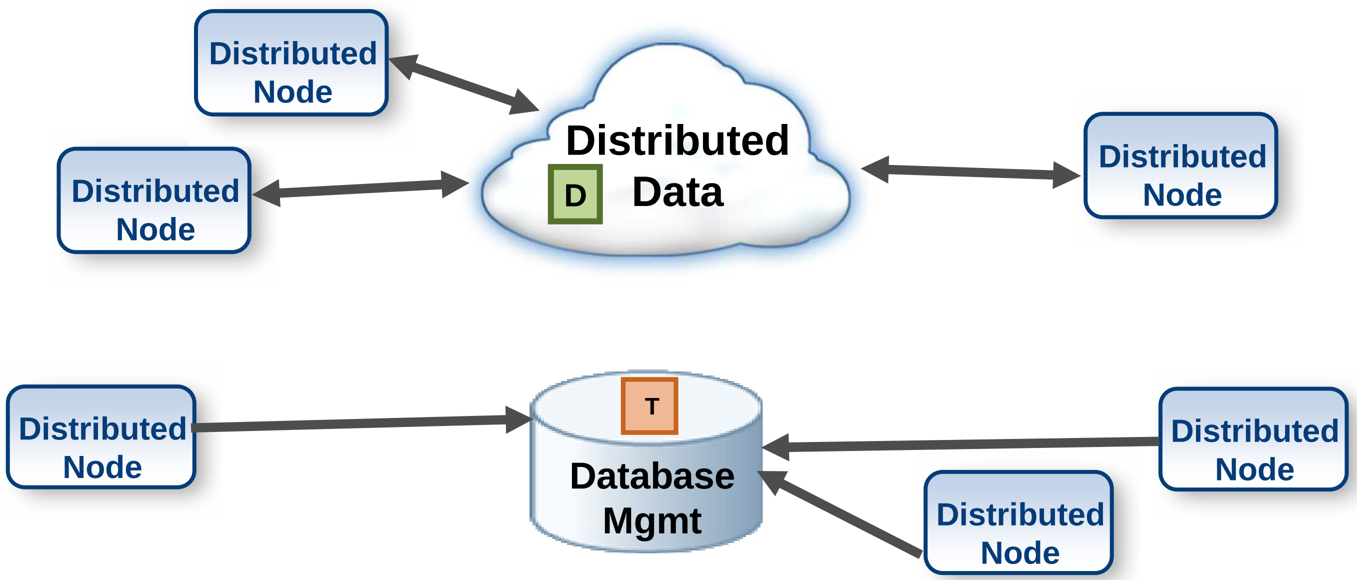
- Motivation & Background
- Use cases for Global Data
- Mapping between DDS and RDBMS
- Proof-of-concept
- Conclusion

Motivation for a Unified Global Data Space

- Leverage strength of two APIs
  - SQL/ODBC:
    - Optimized data manipulation/management
  - DDS:
    - Optimized data distribution
- Expand the universe of accessible data
  - Access to existing DB-stored data
  - Access to DDS data from legacy DB-based applications
- Exploit complementary capabilities
  - Data in flight (QoS, latency, notifications, real-time)
  - Data storage (persistence, indexing, unlimited capacity)
- Save time and money!
  - Get power & flexibility using standard API's

Background

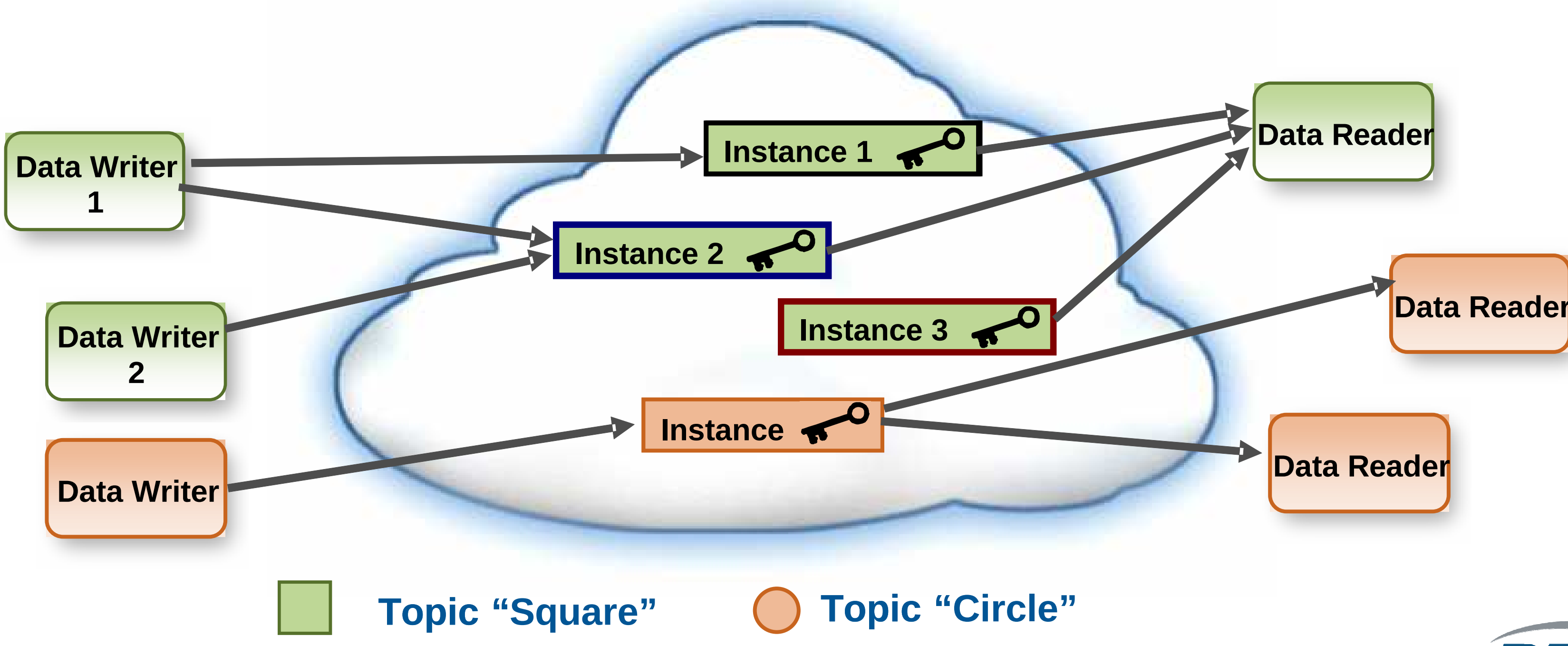
- What is DDS?
- What is a RDBMS?
- Do we need both in the same system?
- Alternative approaches



DDS Enables Global Data

Address in Global Data Space = (DomainId, Topic, Key)

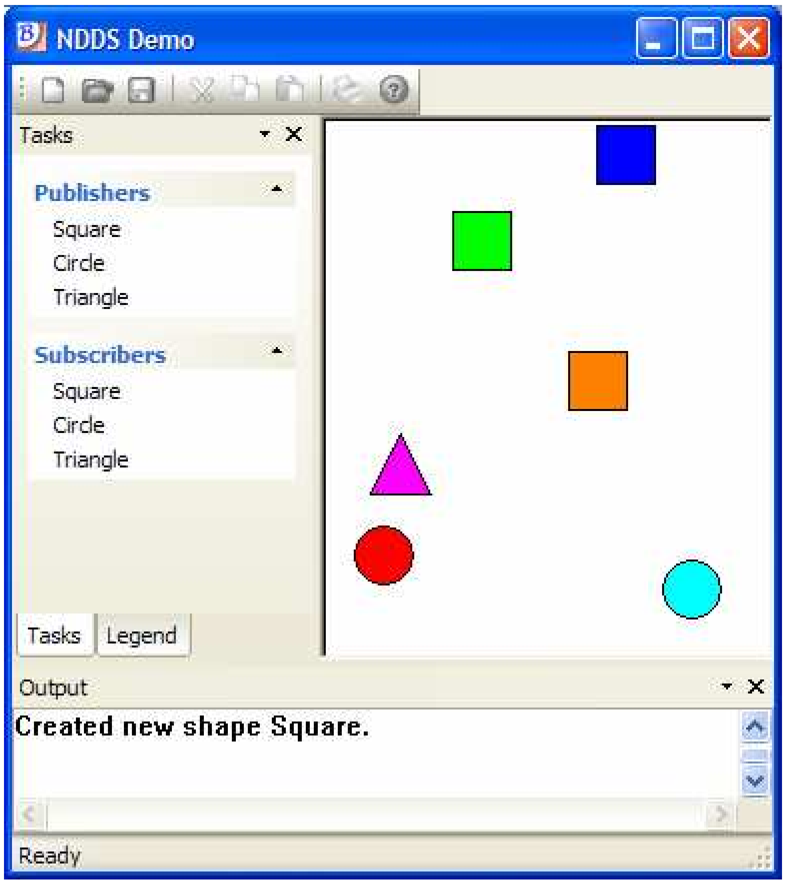
- Each topic corresponds to a multiple data instances
- Each DataWriter can write to multiple instances of a single topic
- Multiple DataWriters may write to the same instance
- Each DataReader can read from multiple instances of a single topic
- Multiple DataReaders may read from the same instances



DDS elements

- Metamodel: Data Distribution PIM
  - Concept of Global Data Space
  - Domain, Topic, Key,
  - Concept of Entity, DomainParicipant, DataReader, DataWriter,
  - Concept of Entity Listener, QoS, Conditions, WaitSets
- Abstract API: DDS PIM
  - Classes on the DDS PSM
  - Operations on the DDS Entities and semantic meaning
  - Description of the available QoS and related behavior
- Concrete programming APIs (C++, Java, etc.)
  - Actual programming API's

DDS Example (boxes demo)



- Topics
  - Square, Circle, Triangle
    - all of type Shape
  - Attributes
    - of type Atributes
- IDL data types:

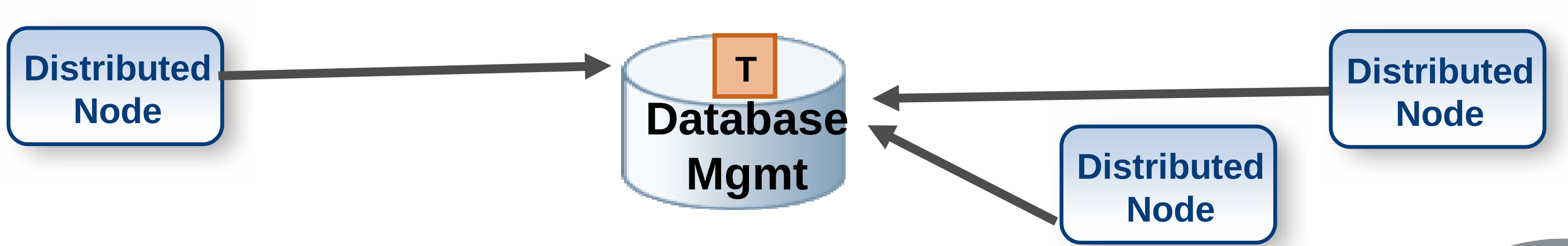
```
struct Shape {
        string<16> color; // key
        int x;
        int y;
        int size;
};

struct Attributes {
        string<16> shape; // key
        string<16> color;   // key
        float speed;
}
```

RDBMS enables persistence and manipulation of data

Address in a RDBMS data space

= (Database, Table, PrimaryKey)

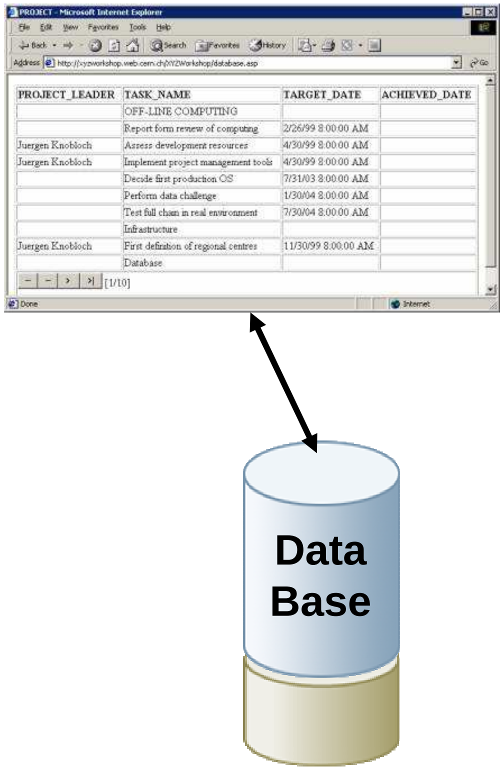
- Each Table stores multiple records, all with the same schema (data-type)
- Each record represents a unique data element
- Records are identified by a primary key
- Applications can read and modify records or groups of records
  - SQL language provides means to address groups of records
- New tables can be 'materializes' by combining records from different tables



RDBMS Elements

- Metamodel: Relational Model
  - Tables, Records, Primary Keys, Foreign Keys
  - Views
- Abstract API: SQL
  - SQL schema
  - SQL statements SELECT, INSERT, UPDATE
  - SQL clauses (WHERE, ORDERED BY, etc.)
- Concrete programming APIs
  - ODBC (C API)
  - JDBC (Java API)
  - other e.g. Oracle's PL/SQL (C API)

RDBMS Example (boxes demo)



"Squares" Table

color	*x*	*y*	*size*
"red"	14	23	70
"blue"	200	67	50

"Circles" Table

color	*x*	*y*	*size*
"red"	43	22	50
"black"	132	66	100

"Attributes" Table

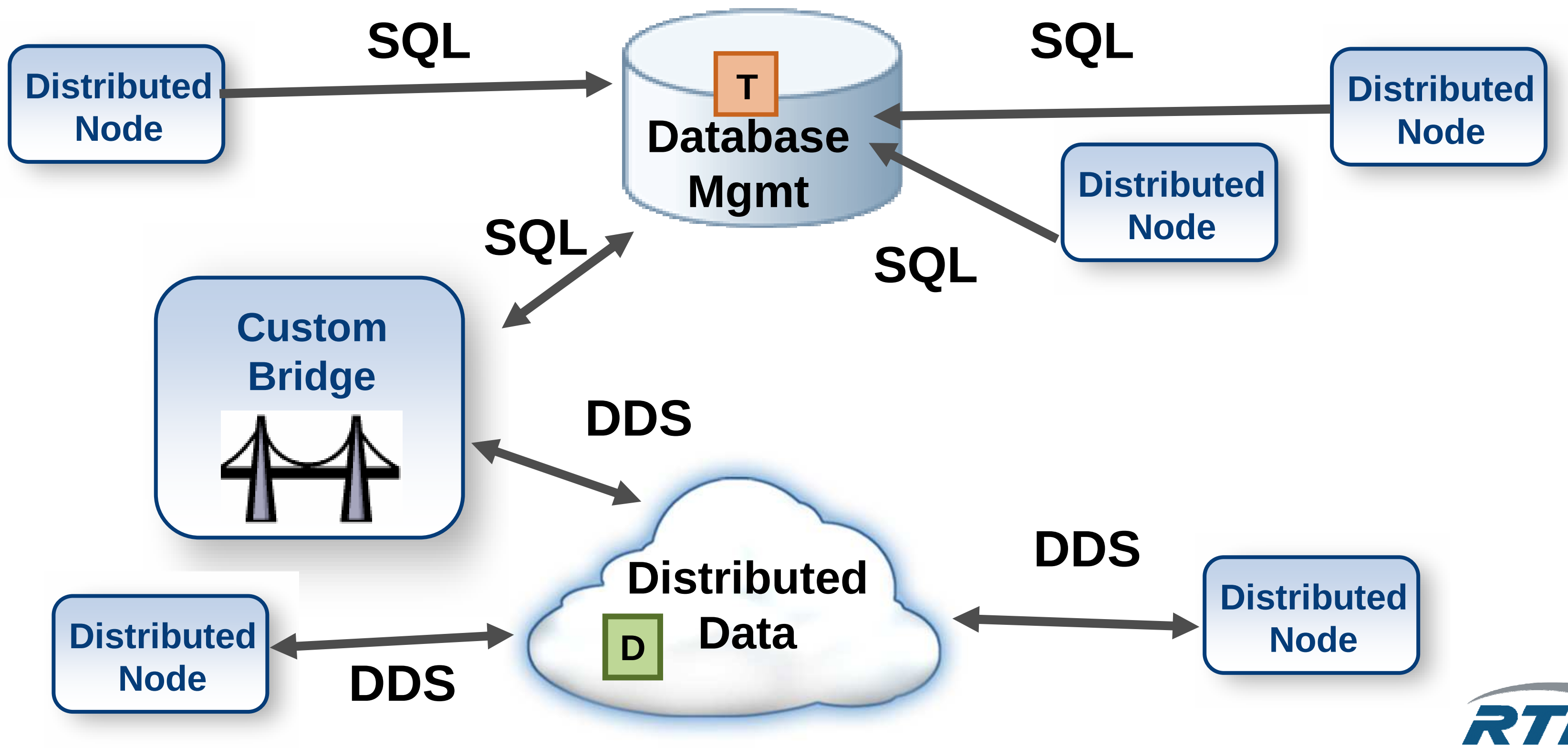
shape	*color*	*size*	*speed*
"square"	"red"	70	5.6
"square"	"blue"	50	22.45
"circle"	"red"	70	101.3
"circle"	"black"	100	45.4

Do I need both in a system?

- Yes. They both have complementary capabilities!
  - DDS is good for:
    - Data “in-fight”: sending & receiving data
    - High performance communication and notifications
    - Access to real-time nodes
    - QoS and resource control
  - RDBMS/SQL is good for
    - Storing and persisting data
    - Data analysis, correlations, fusion
    - Data organization
    - Integration with ODBC clients (GUIs, web-services)
- ... most complex systems are already doing it...

How to use DDS and RDBMS?

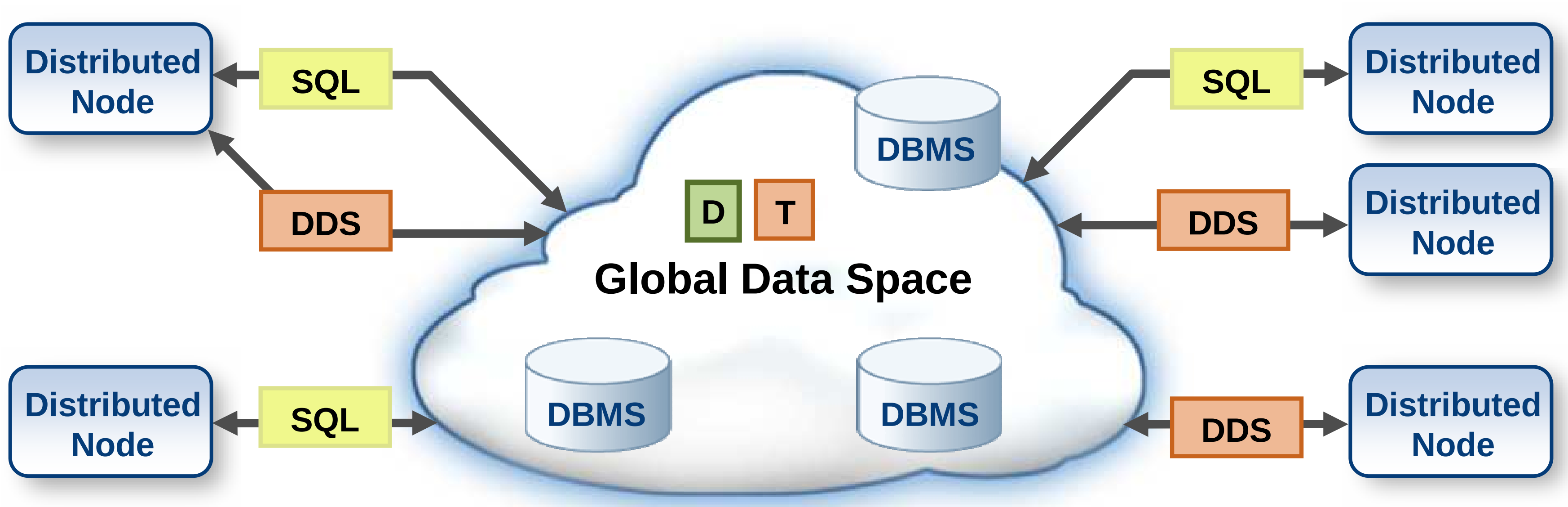
- Until now:
  - Two disconnected data-spaces
  - Custom mapping of data-models
  - Custom application-level bridging



A better way: Standards-Based Global Data Space

All data accessible to all interested applications using either API

- Single, unified, data-space
- Transparent mapping of data-models
- No need for application-level bridging

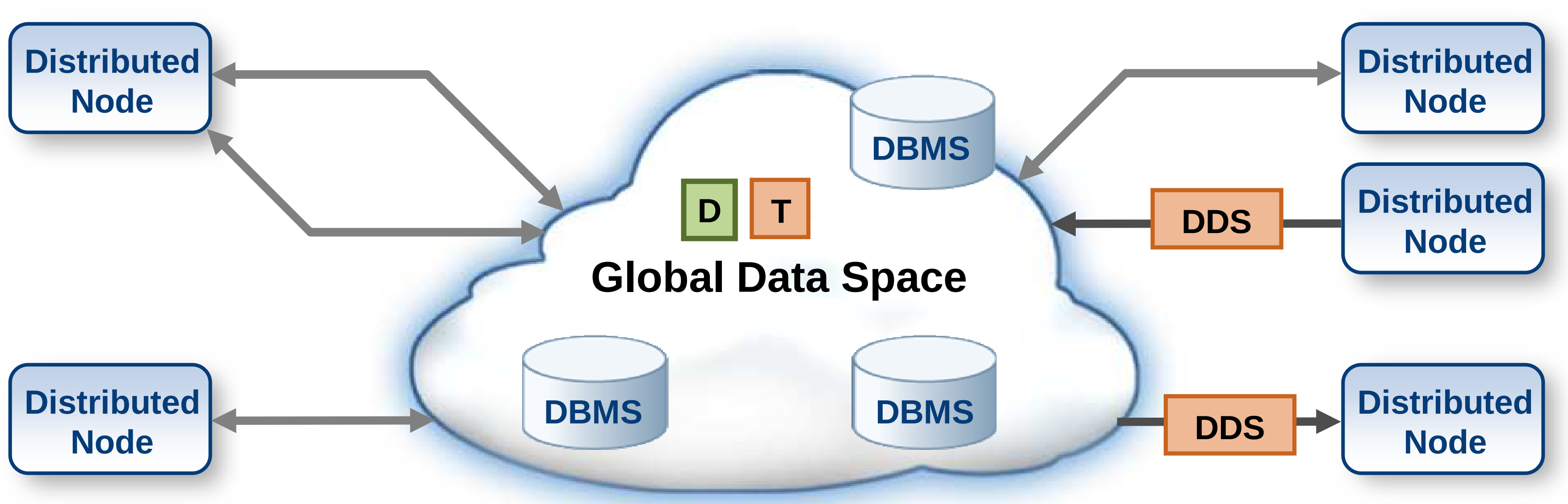


Outline

- Motivation & Background
- Use cases for Global Data
- Mapping between DDS and RDBMS
- Proof-of-concept
- Conclusion

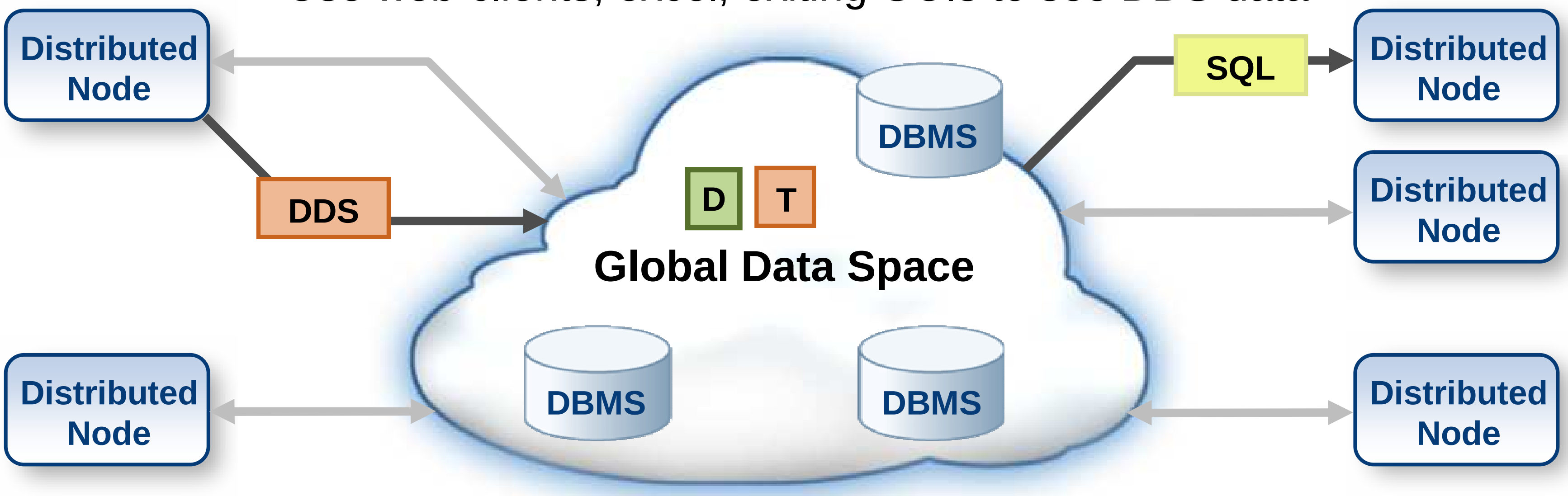
Use cases: DDS to DDS

- DDS write to DDS read:
  - Real-Time data distribution
  - QoS aware communications
  - High performance messaging and events
  - Publish-subscribe communications



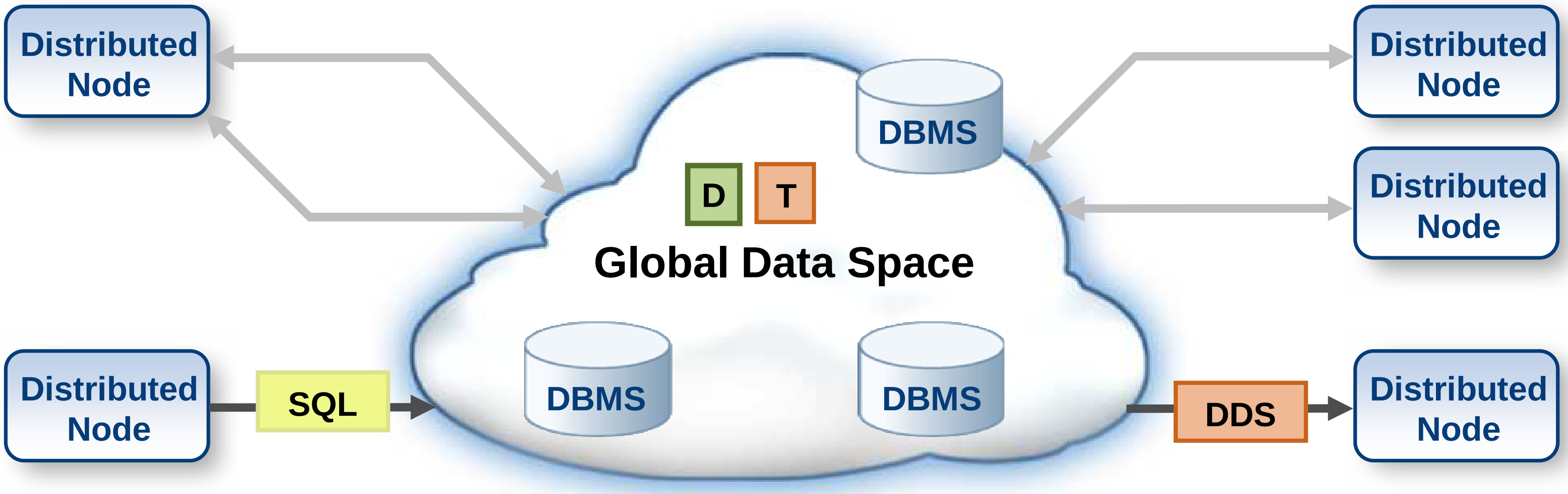
Use cases: DDS write to SQL read

- DDS write to SQL read:
  - Logging
    - Store all historic data values
  - Data analysis/Data mining
    - Look at trends, correlate data from multiple topics
    - Create custom data-views
  - Integration to ODBC analysis/monitoring clients
    - Use web-clients, excel, exiting GUIs to see DDS data



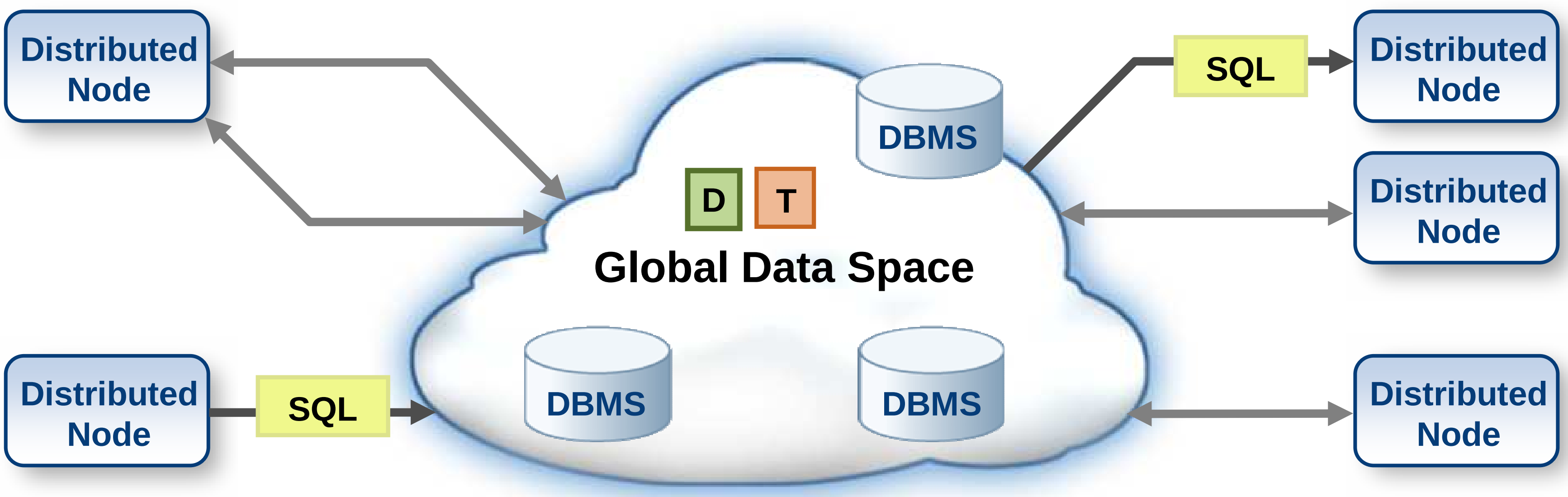
Use cases: SQL write to DDS read

- SQL write to DDS read:
  - Real-Time distributed database data monitoring
    - Get notified when any record is modified
    - Get notification anywhere, even RT nodes
  - Changes to data-sets
    - E.g. Change departure time of all aircraft leaving after 5pm and delay one hour



Use cases: SQL to SQL

- DDS write to SQL read:
  - Classic data-base applications
  - Database replication full or partial
    - Selected table replication
    - Mix of heterogeneous (multi-vendor) databases
    - Mix of in-memory and disk-based databases



Outline

- Motivation & Background
- Use cases for Global Data
- Mapping between DDS and RDBMS
- Proof-of-concept
- Conclusion

RDBMS Elements

- Metamodel: Relational Model
  - Tables, Records, Primary Keys, Foreign Keys
  - Views
- Abstract API: SQL
  - SQL schema
  - SQL statements SELECT, INSERT, UPDATE
  - SQL clauses (WHERE, ORDERED BY, etc.)
- Concrete programming APIs
  - ODBC (C API)
  - JDBC (Java API)
  - other e.g. Oracle's PL/SQL (C API)

DDS ⇔ RDBMS mapping at the metamodel and abstract levels

DDS

- Metamodel: Data Distribution PIM
  - Global Data Space
  - Domain, Topic, Key,
- Abstract API: DDS PIM
  - Classes and operations on the DDS PSM
  - QoS
- Concrete programming APIs

RDBMS

- Metamodel: Relational Model
  - Tables, Records,
  - Primary Keys, Foreign Keys
- Abstract API: SQL
  - SQL schema
  - SQL statements
- Concrete programming APIs

Mapping (behavior)

Data Distribution Service (DDS)	Database Management System (DBMS)
IDL Type	**Table Schema**
Topic	**Table**
Data-object Instance	A single **Row** in a Table if no history is stored; or **Multiple Rows** in a Table if history is stored.
Instance Key	**Primary Key** in Table
DataWriter::write*() [all variations]	*INSERT* or UDPATE
DataWriter::dispose()	*DELETE*

Mapping (data model)

- IDL struct ➔ flattened out into a table schema
  - column name ➔ full hierarchical member name
  - field type → mapping of IDL primitive type to corresponding SQL type
- IDL union ➔ same as struct with additional discriminator column
- IDL sequence ➔ several possible mappings
  - In-line
    - length + nullable max-length columns [bounded only]
  - Extra table
    - Can be shared per sequence type or dedicated
  - Octet sequence ➔ SQL binary

Mapping (data model)

- IDL array ➔ in-line expansion
  - Octet array ➔ SQL binary
- IDL valuetype ➔ two options
  - Performance-oriented:
    - Same as struct (including base valuetype fields)
  - Relational-model oriented
    - Expansion tables as in the DLRL mapping

Almost 1-1 mapping of primitive types

IDL Type	IDL Field Name	SQL Type	Table Field Name
CHAR	my_field	CHAR(1)	“my_field”
WCHAR	my_field	WCHAR	“my_field”
OCTET	my_field	BINARY(1)	“my_field”
BOOLEAN	my_field	TINYINT	“my_field”
SHORT	my_field	SMALLINT	“my_field”
UNSIGNED SHORT	my_field	SMALLINT	“my_field”
LONG	my_field	INTEGER	“my_field”
UNSIGNED LONG	my_field	INTEGER	“my_field”
DOUBLE	my_field	DOUBLE	“my_field”
FLOAT	my_field	REAL	“my_field”
STRING<LENGTH>	my_field	VARCHAR(LENGTH)	“my_field”
WSTRING<LENGTH>	my_field	WVARCHAR(LENGTH)	“my_field”
LONG LONG	my_field	BIGINT	“my_field”
UNSIGNED LONG LONG	my_field	BIGINT	“my_field”
LONG DOUBLE	my_field	BINARY(128)	“my_field”

Mapping examples

IDL Type | **SQL Table Schema**

```
struct MyStruct {
    long my_key_field; //@key
    short my_short_field;
};
```

```
Create Table "MyStruct" (
    "my_key_field" INTEGER NOT NULL,
    "my_short_field" SMALLINT NOT NULL,

    PRIMARY KEY("my_key_field")
);
```

```
struct MyStructContainer {
    long my_key_field; //@key

    MyStruct my_struct_field;
};
```

```
Create Table "MyStructContainer" (
    "my_key_field" INTEGER NOT NULL,

    "my_struct_field.my_key_field"  INTEGER NOT NULL,
    "my_struct_field.my_short_field" SMALLINT NOT NULL,

    PRIMARY KEY("my_key_field")
);
```

```
struct MySequenceContainer {
    long my_key_field; //@key

    sequence<long,2> my_seq_field;
};
```

```
Create Table "MySequenceContainer" (
    "my_key_field"  INTEGER NOT NULL,

    "my_seq_field#length" INTEGER NOT NULL,
    "my_seq_field[0]" INTEGER,
    "my_seq_field[1]" INTEGER,

    PRIMARY KEY("my_key_field")
);
```

Mapping extends to the wire representation

DDS Type

DDS-DBMS
Auto-Generated Code
(UserDataReader, UserDataWriter)

DBMS Table

```
struct MyType
{
   int i;
   double x;
   string<100> s;
};
```

MyType			Slot #
i INTEGER	x DOUBLE	s VARCHAR(100)	

DDS
MyTypeSupport

DBMS-DDS
Generic type serialization/de-serialization
using the Table Schema
(BridgeDataWriter, BridgeDataReader)

Serialized Wire Format Representation on the "Network"

Configuration

- Table name ➔ DDS topic_name + domain_id
  - Can allow multiplexing:
    - Topics of same type can be mapped into a single table
    - A table can be published into multiple topics
- Partial/custom mappings
  - Unmapped Topics/Tables
  - Unmapped fields
    - DBMS only fields
    - DDS only fields
  - Custom mappings
- DDS QoS
- Database history

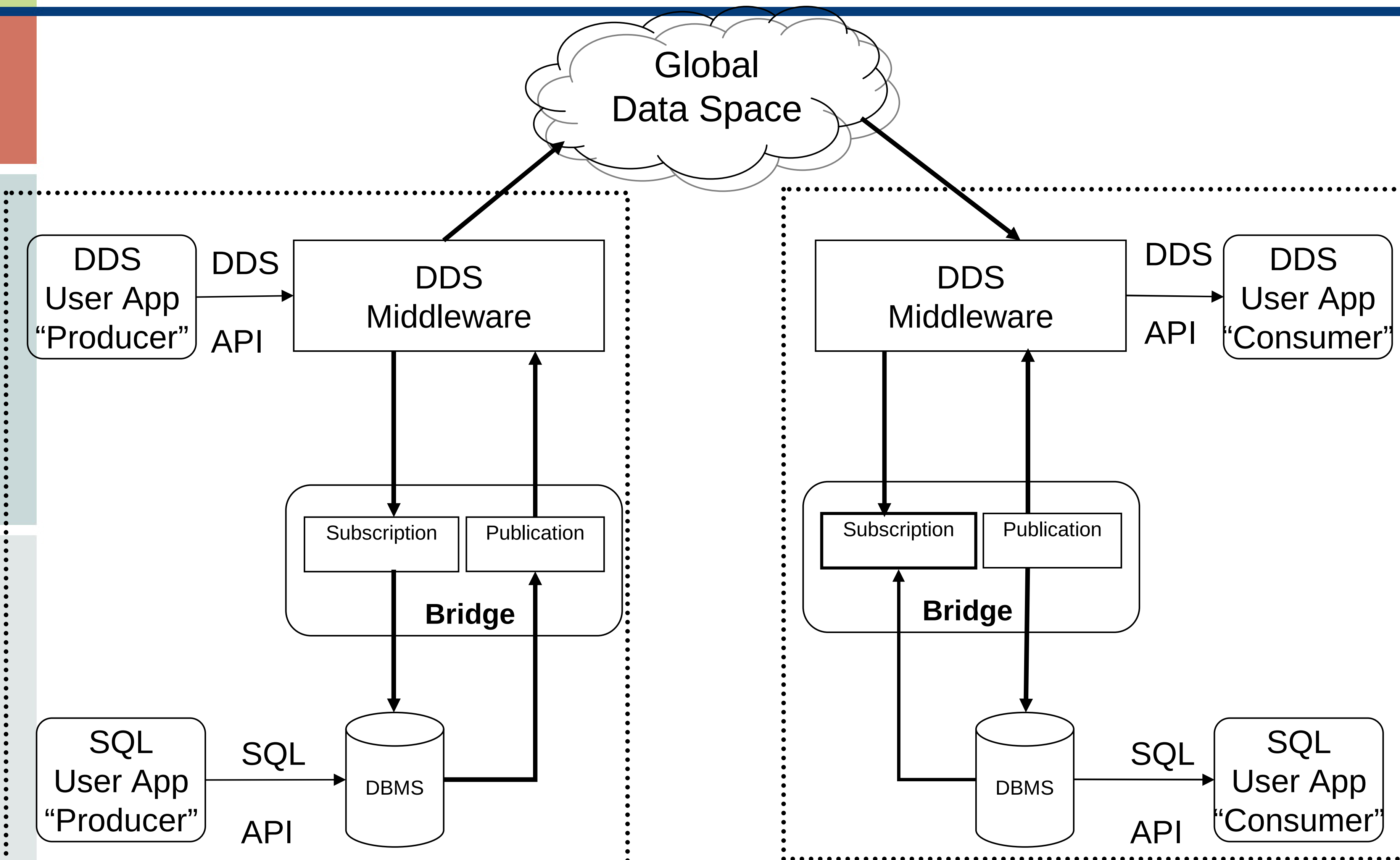
Configuration approaches

- Configuration file (static)
- Configuration tables (static or dynamic)
  - Publication, Subscription tables
  - Can use SQL or DDS to configure
- Other?
  - command-line
  - dedicated API e.g. web-service

Outline

- Motivation & Background
- Use cases for Global Data
- Mapping between DDS and RDBMS
- Proof-of-concept
- Conclusion

Proof of concept



Conclusion

The integration of relational databases and DDS:

- Can be formally defined
- Is practical and feasible
- ... and offers powerful benefits:
  – Leverages strength of two API standards
  – Expands the universe of accessible data
  – Exploits complementary capabilities
  – Saves time and money!

Thank you

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

Fernando Crespo Sanchez
fernando.crespo@rti.com

www.rti.com