OpenSplice DDS

Delivering Performance, Openness, and Freedom

Angelo Corsaro, Ph.D.
Chief Technology Officer
OMG DDS SIG Co-Chair
angelo.corsaro@prismtech.com

PRISMTECH
Powering Netcentricity

The DDS Tutorial

::Part II

Tutorial Scope

Scope & Goals

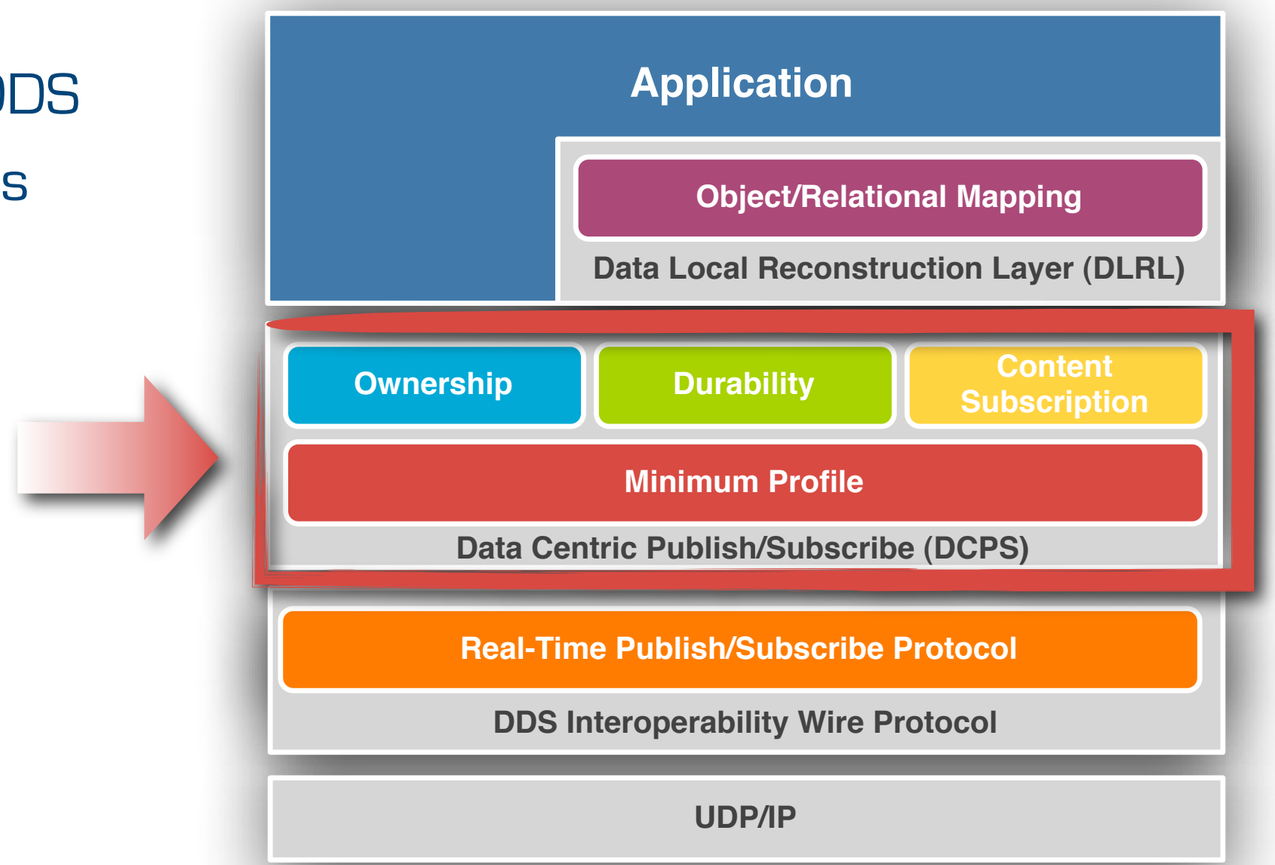
- The Tutorial will cover the DCPS layer of DDS
- It will give you enough details and examples to make sure that you can get started writing DDS applications

Software

- OpenSplice DDS
  - http://www.opensplice.org
- SIMple Dds (SIMD)
  - http://code.google.com/p/simd-cxx

Prerequisite

- Basic C++ understanding



OpenSplice DDS

PRISMTECH

What You've Learned on Part I

- Defining Topics and Topic Types
- Scoping Information with Partitions
- Writing Data
- Reading (Taking) data with Waitsets and Listeners
- Writing an example that demonstrate all of the above

What we'll Cover Today

- Content Filtered Topics and Queries
- QoS and the Request vs. Offered Model
- Setting QoS on DDS Entities
- Tuning OpenSplice DDS Configuration

Your will learn:

- What Filters and Queries are
- The available SQL92 subset
- Programming Filters and Queries

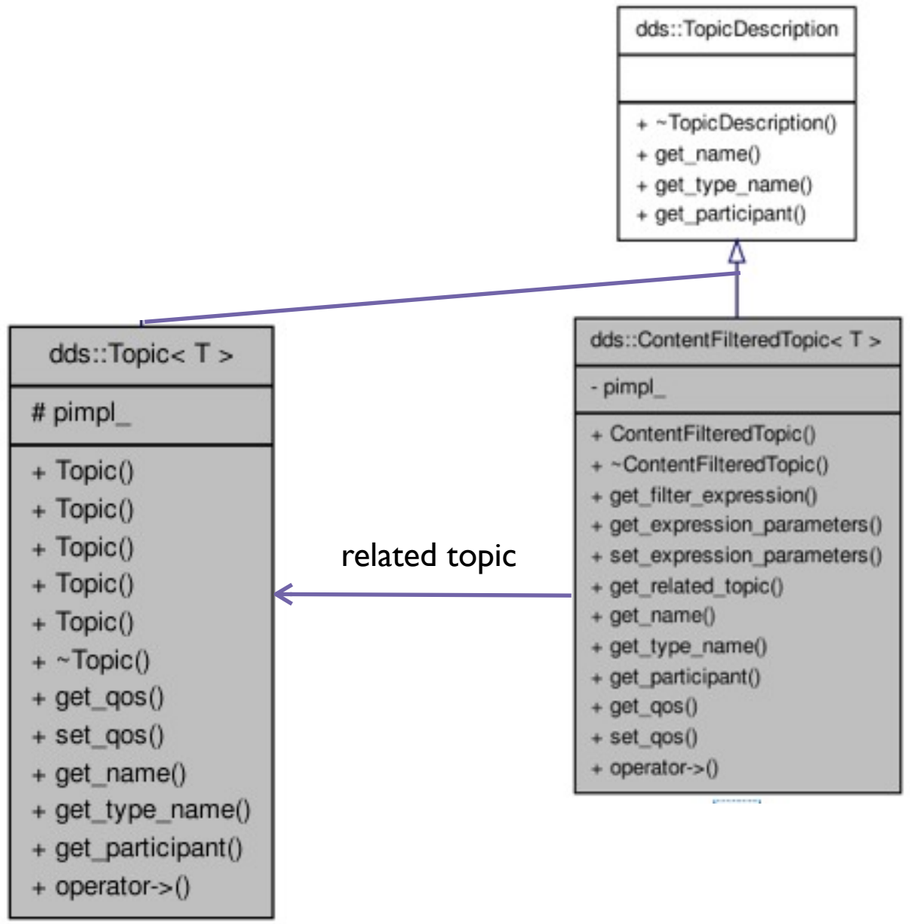
Filters and Queries

DDS Filters & Queries

- DDS provides means for **filtering data** using an application specified **condition expression**
- The two mechanism provided in order to **filter data** are:
  - `ContentFilteredTopics`
  - `QueryCondition`
- Both working in conjunction with a `DataReader`

ContentFilteredTopic

- A `ContentFilteredTopic` can be seen as a decorator of a user defined topic defining a specific filter over the data published for the given topic
- As a result, a `ContentFilteredTopic` always exist in conjunction with its related `Topic`
- A DataReader created with a specific `ContentFilteredTopic` will only receive the data that matches the filter condition
- `ContentFilteredTopic` can be thought of as Continuous Queries



QueryCondition

- A `QueryCondition` can be created over an existing `DataReader` in order to select, among received data, only the subset matching the condition associated with the `QueryCondition`
- `QueryCondition` are created for a specific `DataReader` and used as an argument of the `DataReader::read_w_condition` method

Filters & Queries Grammar

```
Condition::= Predicate
        | Condition 'AND' Condition
        | Condition 'OR' Condition
        | 'NOT' Condition | '(' Condition ')'

Predicate::= ComparisonPredicate | BetweenPredicate

ComparisonPredicate::= FIELDNAME RelOp Parameter | Parameter RelOp FIELDNAME

BetweenPredicate::= FIELDNAME 'BETWEEN' Range | FIELDNAME 'NOT BETWEEN' Range

RelOp::= '=' | '>' | '>=' | '<' | '<=' | '<>' | like

Range::= Parameter 'AND' Parameter

Parameter::= INTEGERVALUE | FLOATVALUE
     | STRING | ENUMERATEDVALUE | PARAMETER

where PARAMETER has the for %n (with n in 0..100)
```

OpenSplice DDS

PRISMTECH

ContentFilteredTopics in SIMD

```
template <typename T>
class dds::ContentFilteredTopic : public dds::TopicDescription {
public:
   ContentFilteredTopic(const std::string& name,
                        const dds::Topic<T>& t,
                        const std::string& filter,
                        const std::vector<std::string>& params);

   virtual ~ContentFilteredTopic();
public:
   std::string get_filter_expression() const;
   std::vector<std::string> get_expression_parameters() const;
   void set_expression_parameters(const std::vector<std::string>& params);

   dds::Topic<T> get_related_topic() const;

   virtual std::string get_name() const;

   virtual std::string get_type_name() const;

   virtual dds::DomainParticipant get_participant() const;

   TopicQos get_qos() const;
   void set_qos(const TopicQos& qos);
};
```

Using ContentFilteredTopics

```
// Create the "TempSensor" Topic
dds::Topic<TempSensorType> tsTopic("TempSensor");

// Create the filter parameters
std::vector<std::string> params(2);
params[0] = "30";
params[1] = "0.6";

// Create the ContentFilteredTopic
dds::ContentFilteredTopic<TempSensorType>
    cfTsTopic("TempSensor-1",
              tsTopic,
              "(temp < %0) AND (hum < %1)",
              params);

// Create the DataReader with for the ContentFilteredTopic
dds::DataReader<TempSensorType> dr(cfTsTopic);
```

```
enum TemperatureScale {
   CELSIUS,
   FAHRENHEIT,
   KELVIN
};

struct TempSensorType {
   short id;
   float temp;
   float hum;
   TemperatureScale scale;
};
#pragma keylist TempSensorType id
```

QueryCondition in SIMD

```
template <typename T>
class DataReader {
public:
   // [...] Other DataReader methods
   QueryCondition create_querycondition(const std::string& expression, const std::vector& params);

   QueryCondition create_querycondition(const SampleStateMask& samples_state,
                                        const ViewStateMask& views_state,
                                        const InstanceStateMask& instances_state,
                                        const std::string& expression, const std::vector& params);

   ReturnCode_t read_w_condition(TSeq& samples, SampleInfoSeq& infos, const QueryCondition& cond);

   ReturnCode_t read_w_condition(TSeq& samples, SampleInfoSeq& infos,
                                 const QueryCondition& cond, long max_samples);


   ReturnCode_t take_w_condition(TSeq& samples, SampleInfoSeq& infos, const QueryCondition& cond);

   ReturnCode_t take_w_condition(TSeq& samples, SampleInfoSeq& infos, const QueryCondition& cond,
                                 long max_samples);
}
```

Using QueryConditions

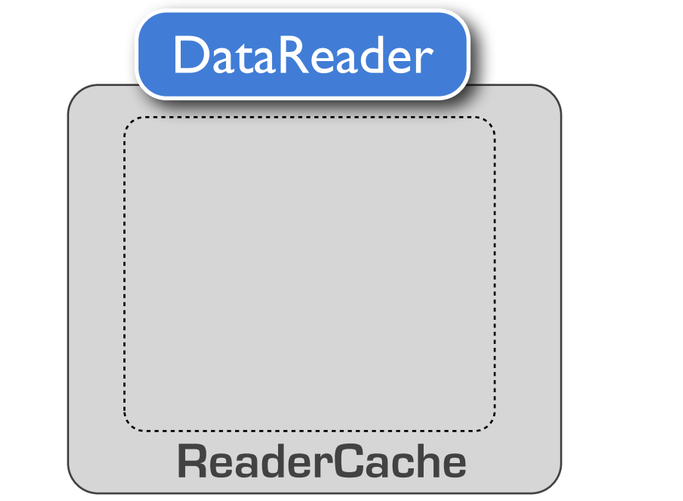
```
dds::Topic<TempSensorType> tsTopic("TempSensor");
dds::DataReader<TempSensorType> dr(tsTopic);

std::vector<std::string> params(2);
params[0] = "30";
params[1] = "0.6";
```

```
// Create Query Condition
dds::QueryCondition cond =
   dr.create_querycondition("(temp < %0) AND (hum < %1)", params);

TempSensorTypeSeq data;
SampleInfoSeq info;
// Read with Condition
dr.read_w_condition(data, info, cond);
```

Filters vs Queries



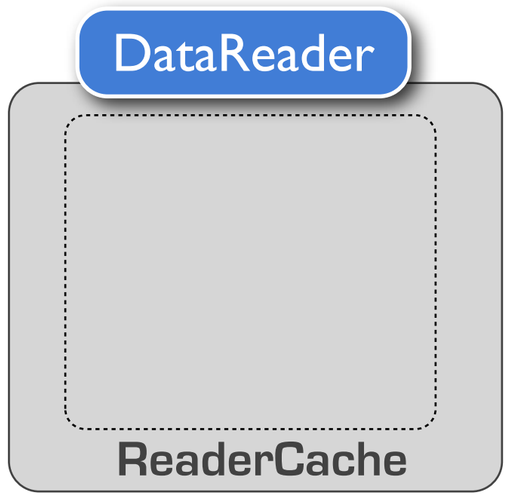
(temp < 30) AND (hum < 0.6)

ContentFilteredTopic

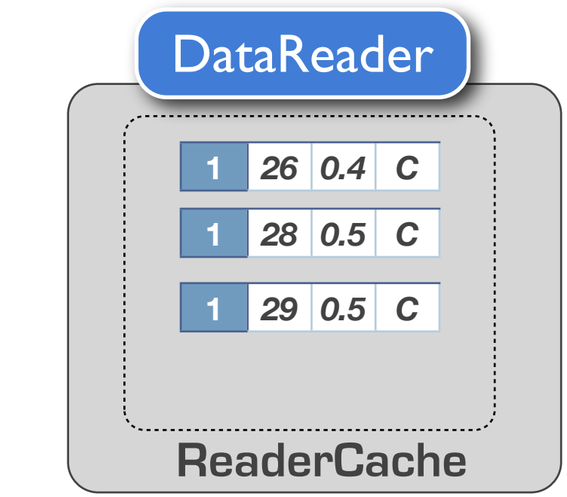
1	26	0.4	C
1	28	0.5	C
1	31	0.5	C
1	30	0.7	C
1	29	0.5	C

QueryCondition

(temp < 30) AND (hum < 0.6)



Filters vs Queries

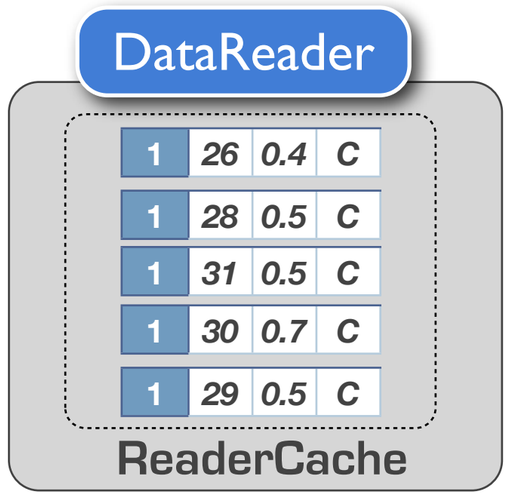



```
(temp < 30) AND (hum < 0.6)
```

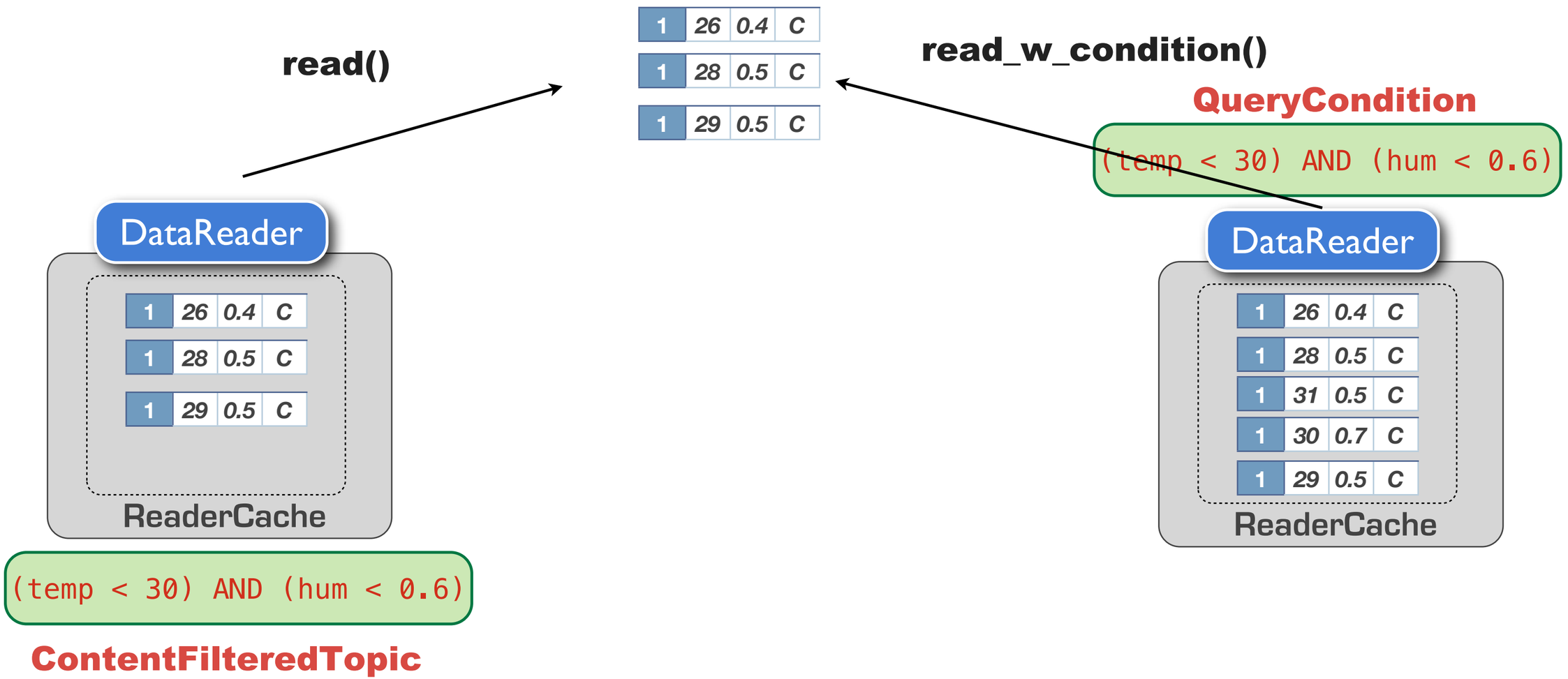
ContentFilteredTopic

QueryCondition

```
(temp < 30) AND (hum < 0.6)
```



Filters vs Queries



OpenSplice | DDS

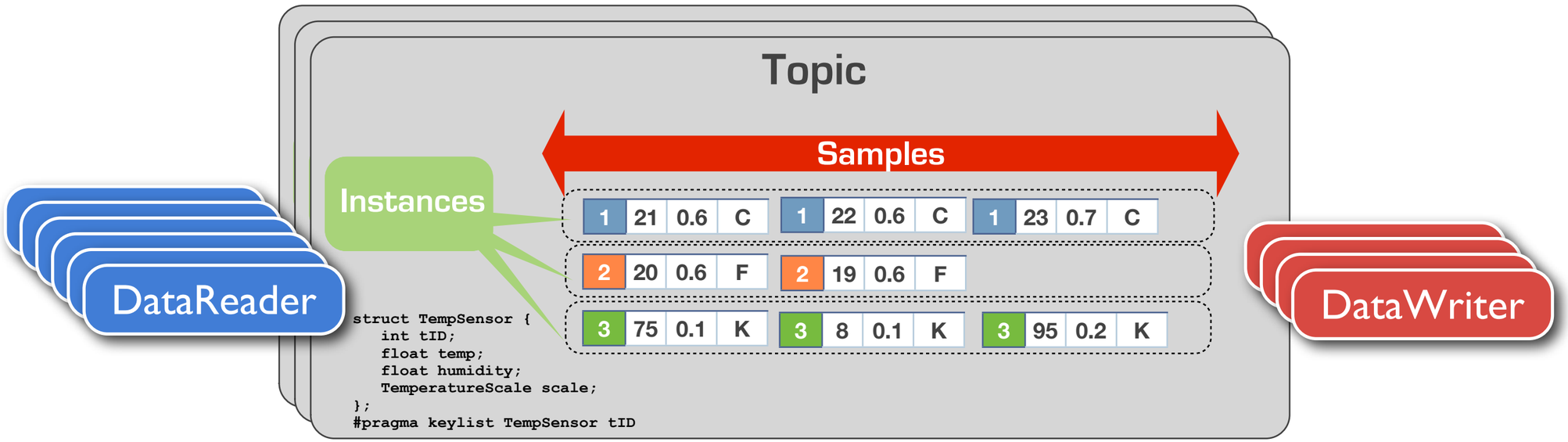
Delivering Performance, Openness, and Freedom

Your will learn:

- What can you control with QoS
- Request vs. Offered in DDS
- QoS Patterns

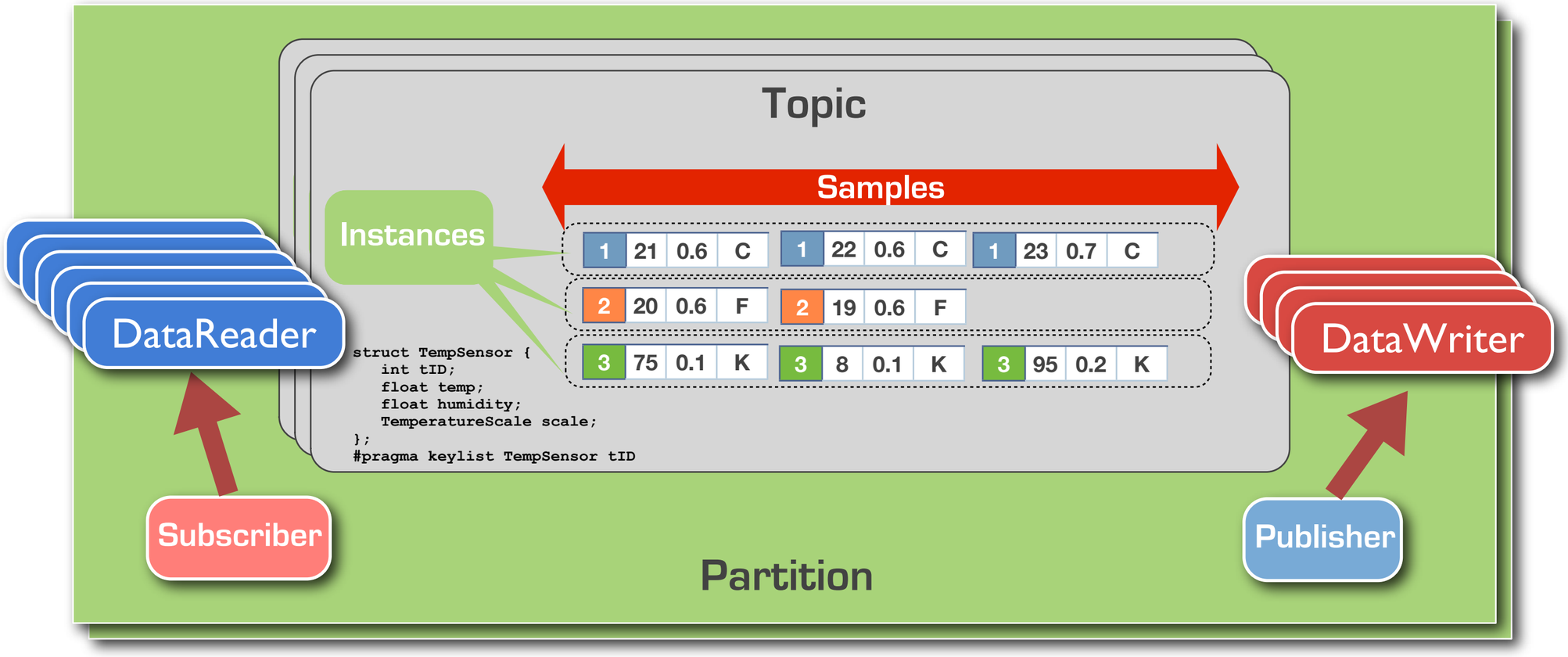
Quality of Service (QoS)

Anatomy of a DDS Application



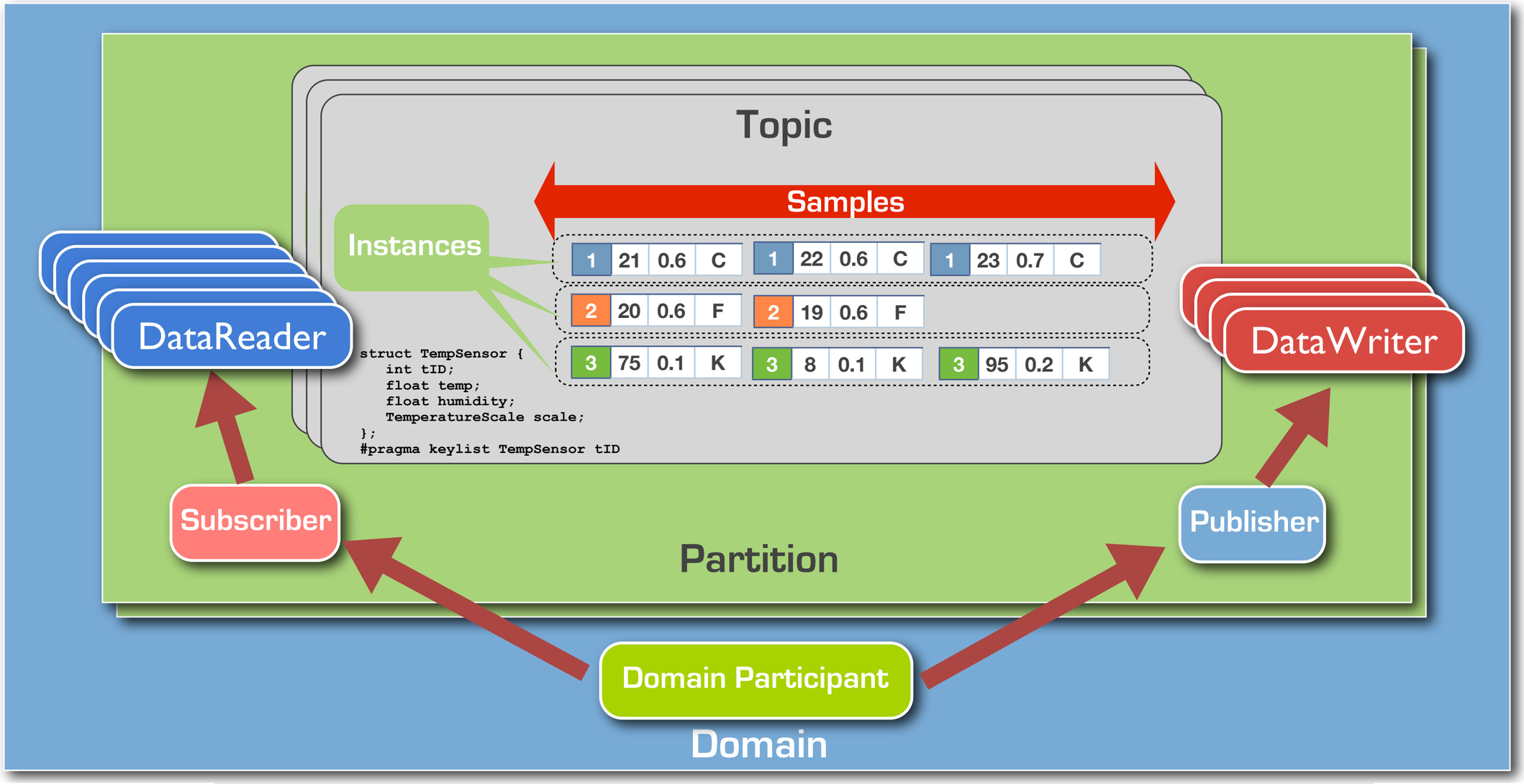
Anatomy of a DDS Application

Arrows show structural relationships, not data-flows



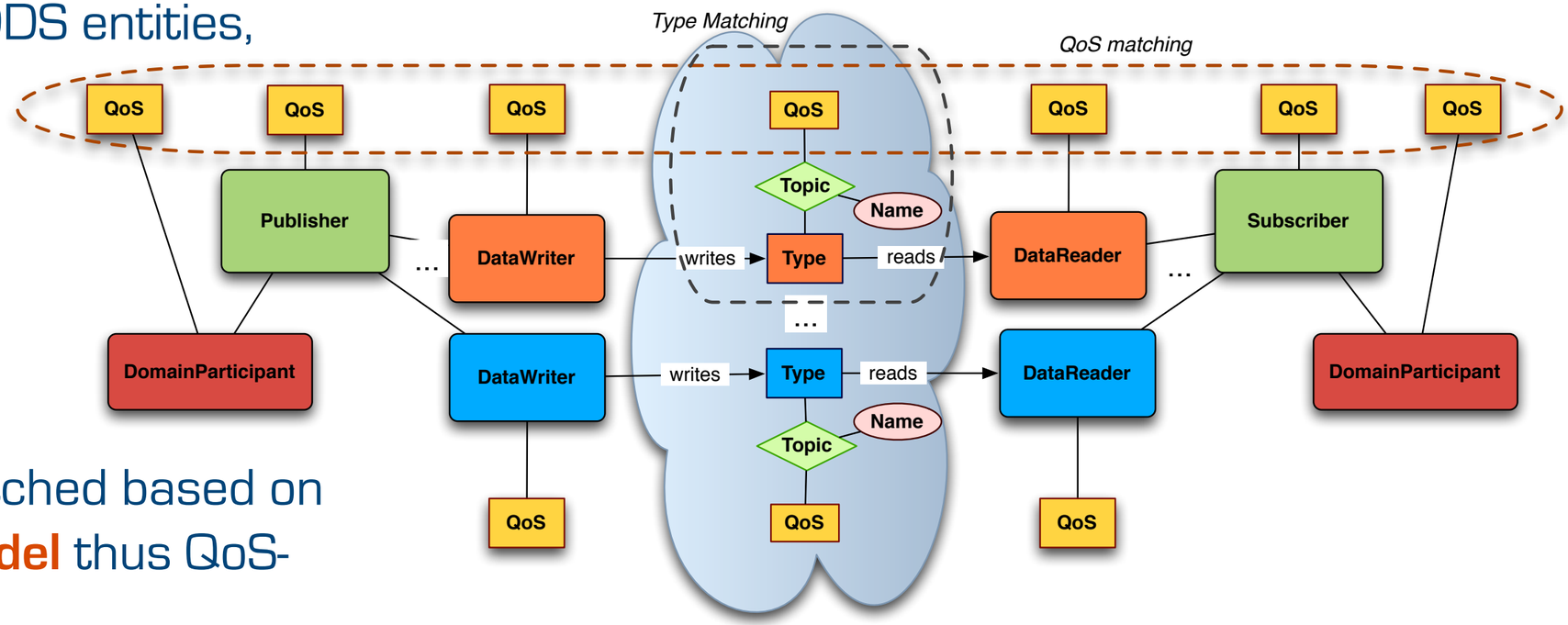
Anatomy of a DDS Application

Arrows show structural relationships, not data-flows



QoS Model

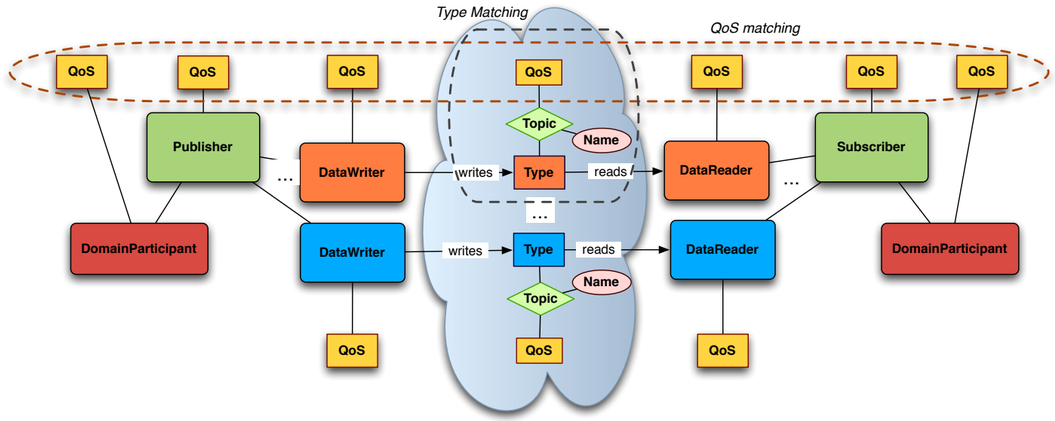
- QoS-Policies are used to control relevant properties of OpenSplice DDS entities, such as:
  - Temporal Properties
  - Priority
  - Durability
  - Availability
  - ...
- Some QoS-Policies are matched based on a **Request vs. Offered Model** thus QoS-enforcement



- Publications and Subscriptions match only if the declared vs. requested QoS are compatible
  - e.g., it is not possible to match a publisher which delivers data unreliably with a subscriber which requires reliability

QoS Policies

QoS Policy	Applicability	RxO	Modifiable	
DURABILITY	T, DR, DW	Y	N	Data Availability
DURABILITY SERVICE	T, DW	N	N	
LIFESPAN	T, DW	N/A	Y	
HISTORY	T, DR, DW	N	N	
PRESENTATION	P, S	Y	N	Data Delivery
RELIABILITY	T, DR, DW	Y	N	
PARTITION	P, S	N	Y	
DESTINATION ORDER	T, DR, DW	Y	N	
OWNERSHIP	T, DR, DW	Y	N	
OWNERSHIP STRENGTH	DW	N/A	Y	
DEADLINE	T, DR, DW	Y	Y	Data Timeliness
LATENCY BUDGET	T, DR, DW	Y	Y	
TRANSPORT PRIORITY	T, DW	N/A	Y	
TIME BASED FILTER	DR	N/A	Y	Resources
RESOURCE LIMITS	T, DR, DW	N	N	
USER_DATA	DP, DR, DW	N	Y	Configuration
TOPIC_DATA	T	N	Y	
GROUP_DATA	P, S	N	Y	



- Rich set of QoS allow to configure several different aspects of data availability, delivery and timeliness
- QoS can be used to control and optimize network as well as computing resource

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

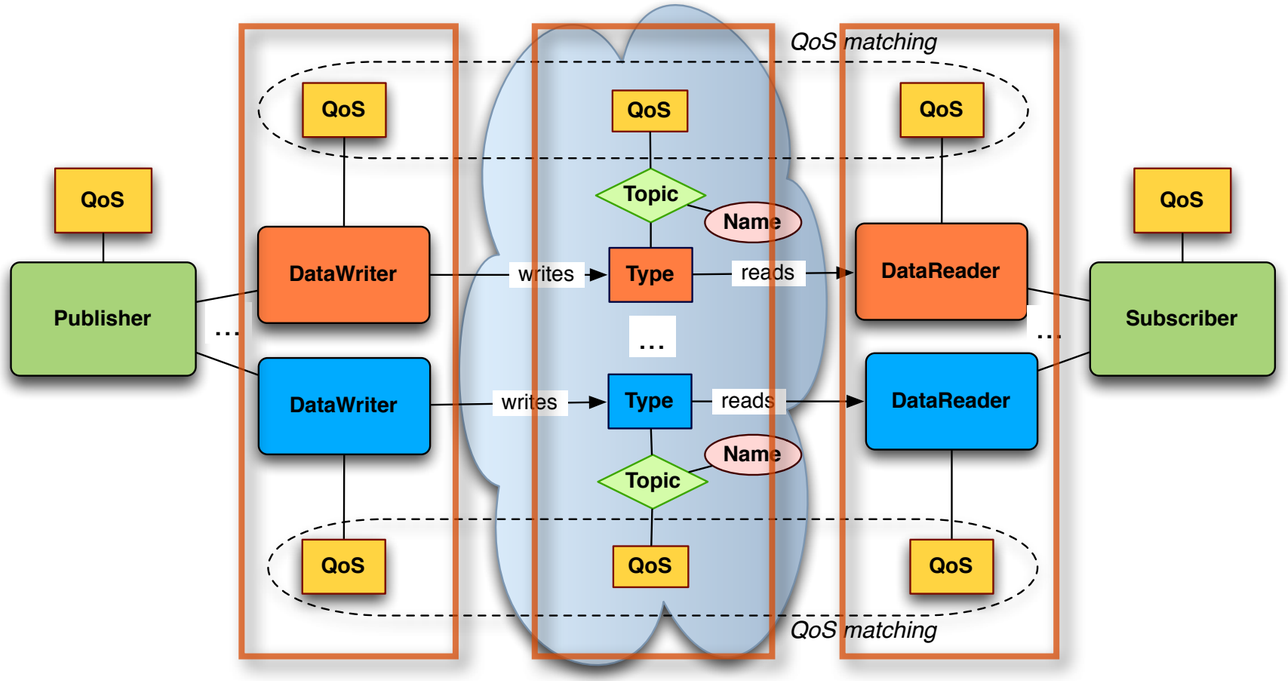
Controlling Reliability

Reliability

The **RELIABILITY** QoS indicate the level of guarantee offered by the DDS in delivering data to subscribers. Possible variants are:

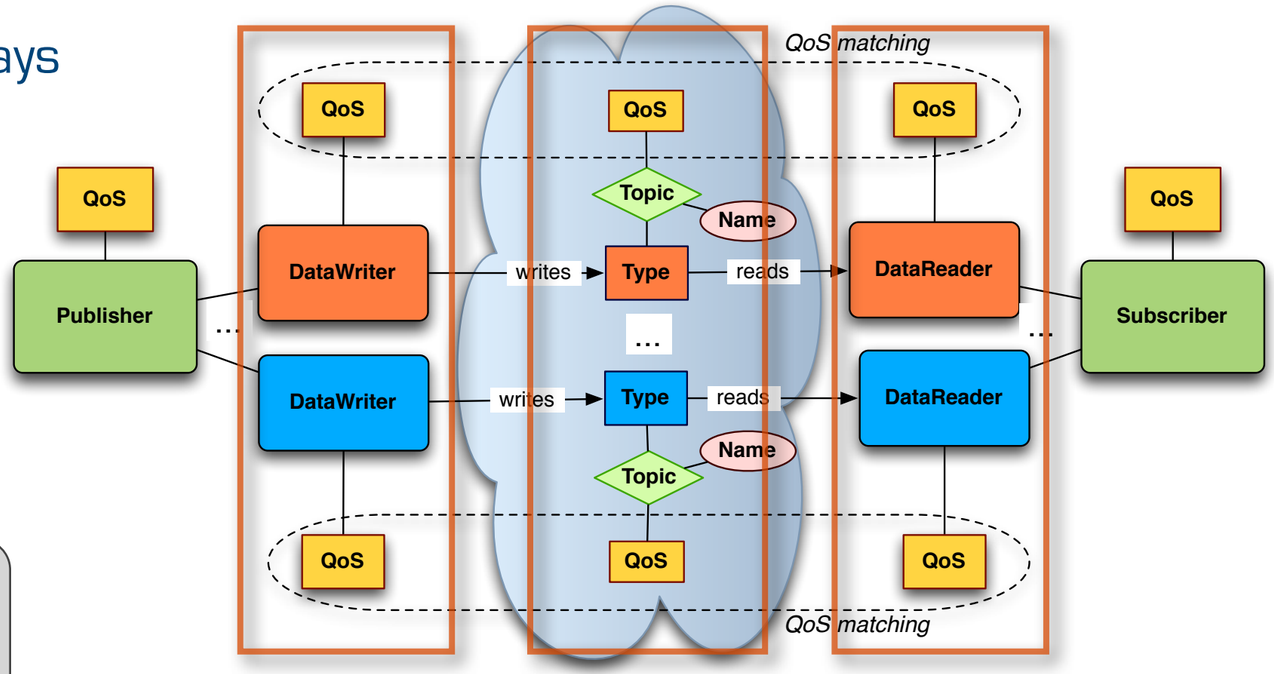
- **Reliable.** In steady-state the middleware guarantees that all samples in the DataWriter history will eventually be delivered to all the DataReader
- **Best Effort.** Indicates that it is acceptable to not retry propagation of any samples

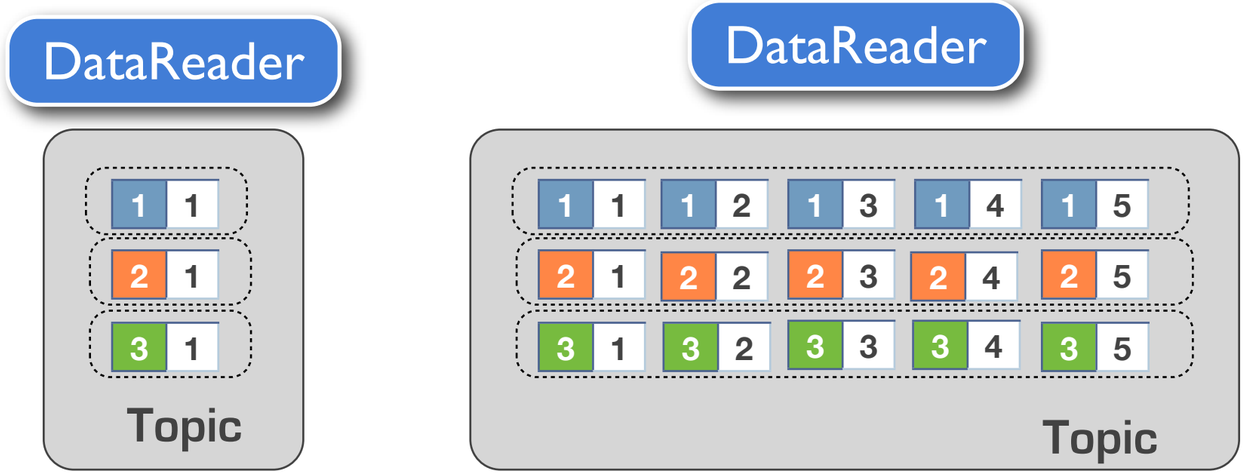
QoS Policy	Applicability	RxO	Modifiable
RELIABILITY	T, DR, DW	Y	N



History

- The History QoS Controls the number of samples-per-instance that will be stored by the middleware on behalf of a Reader
- **Keep Last K**. The History QoS can be set so to always have the latest **K** samples
- **Keep All.** The History QoS can be set so keep all samples produced by the writer and not yet taken, until resource limits are not reached





History Depth = 1 (DDS Default)

History Depth = 5

QoS Policy	Applicability	RxO	Modifiable
HISTORY	T, DR, DW	N	N

OpenSplice DDS

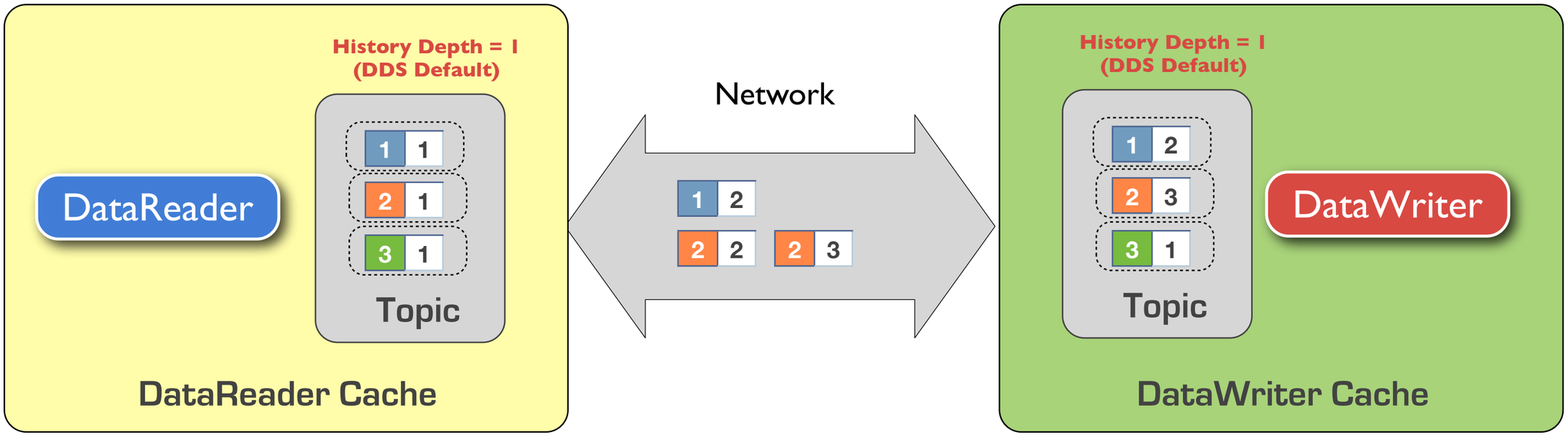
PRISMTECH

Topic Used in next section.

```
struct Counter {
   int cID;
   int count;
};
#pragma keylist Counter cID
```

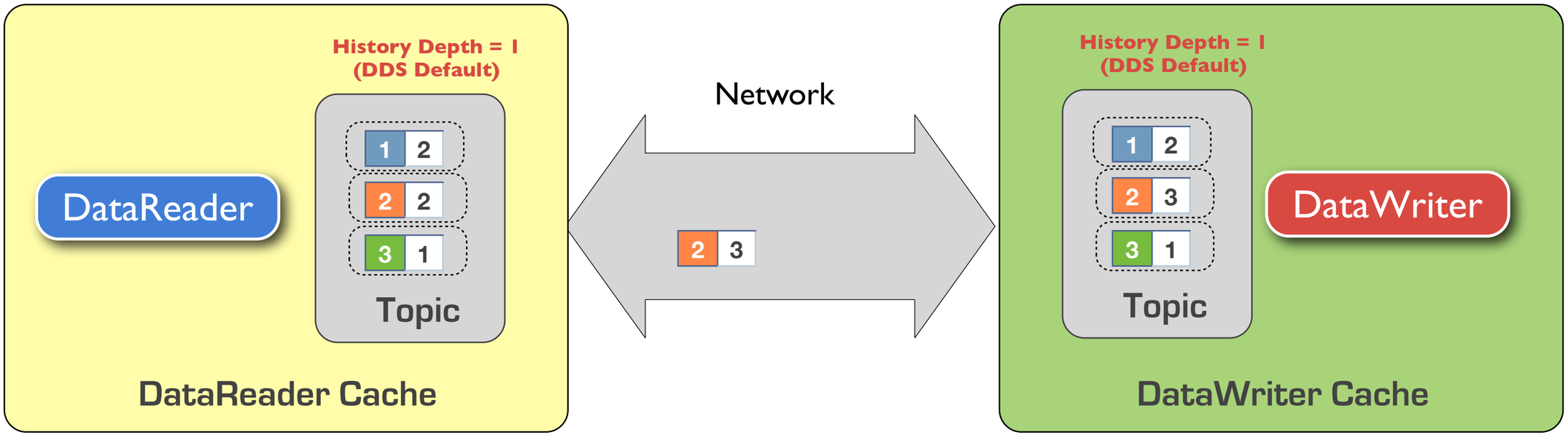
History in Action

History in Action



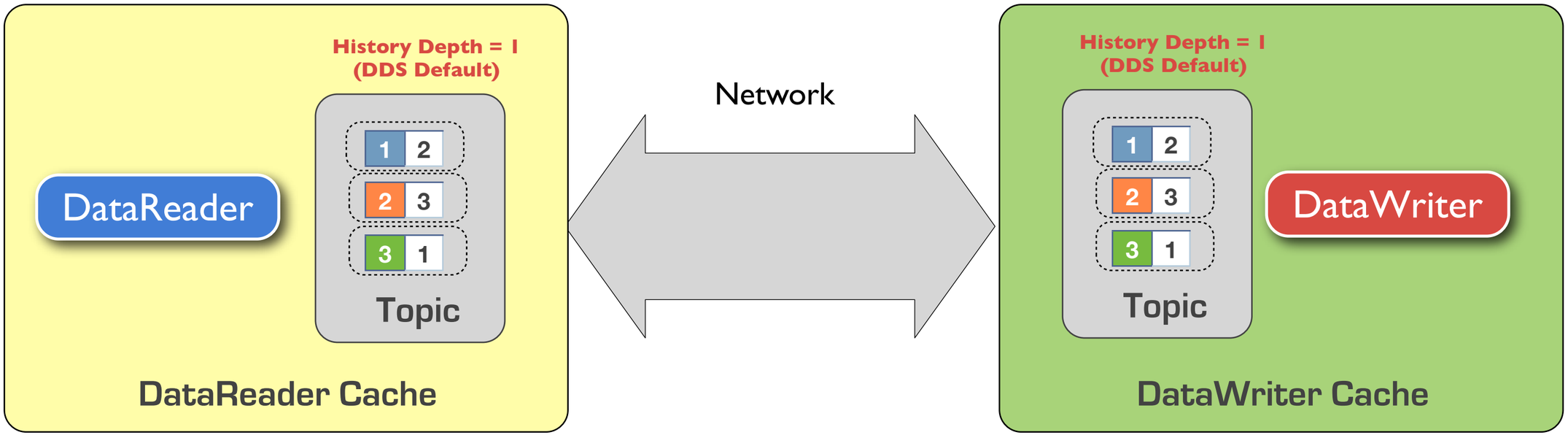
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



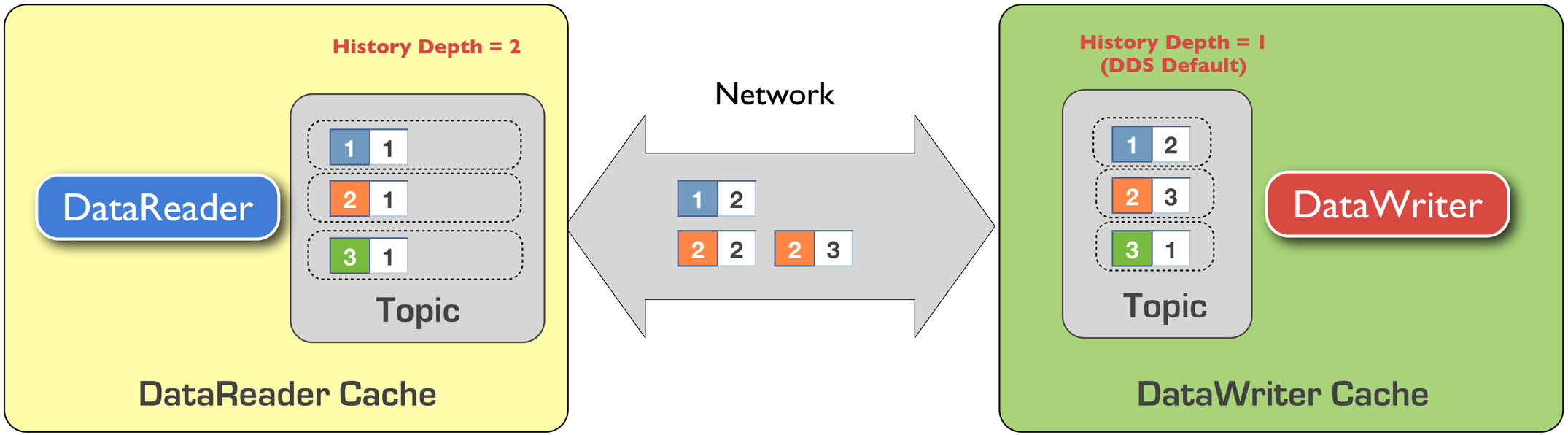
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



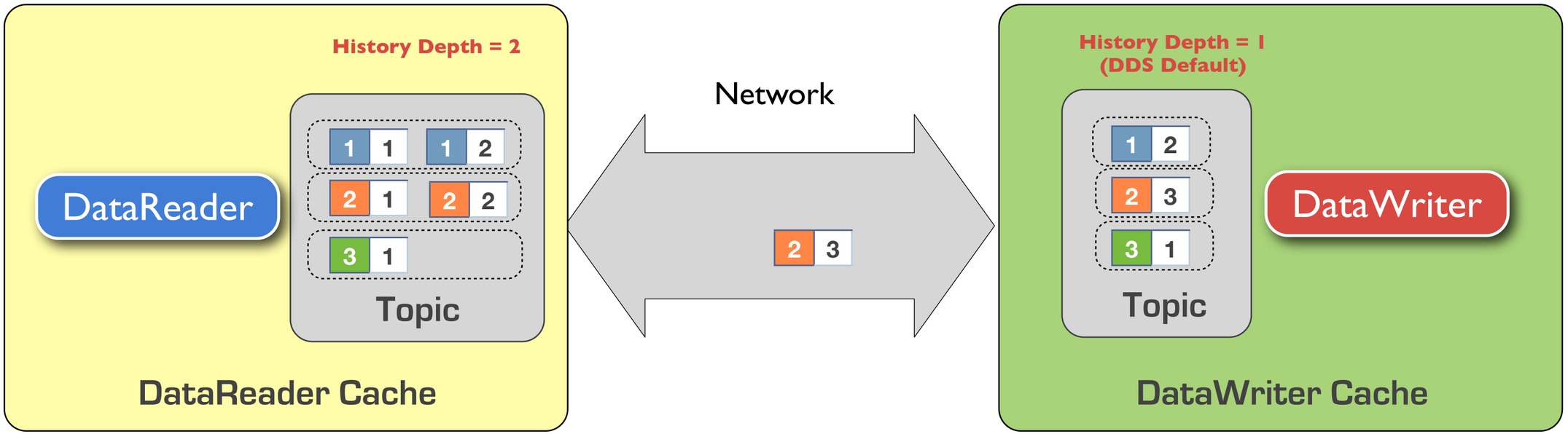
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



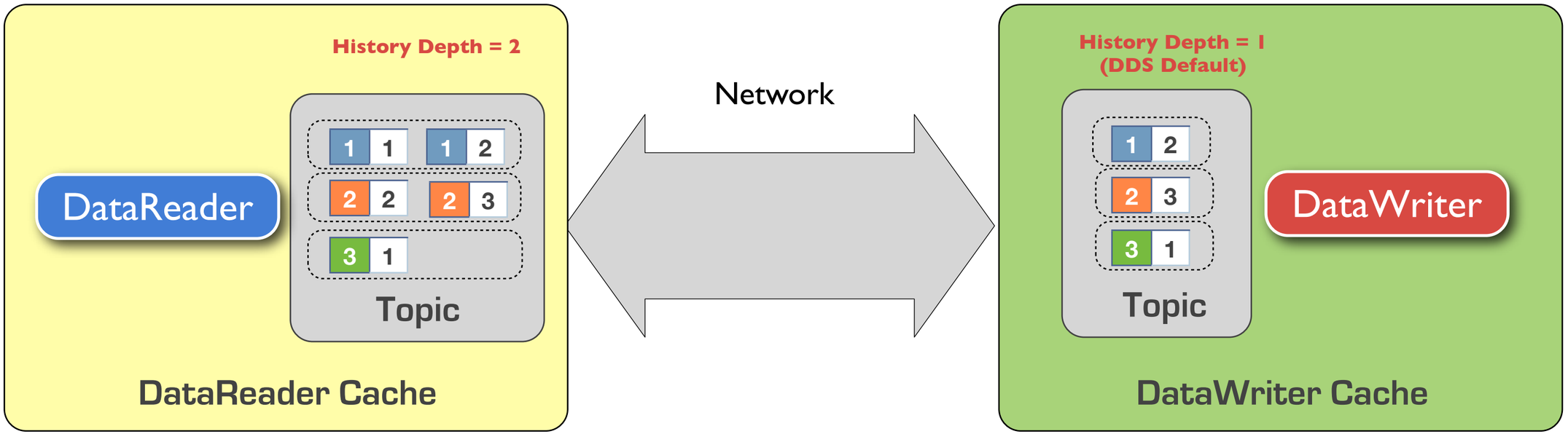
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action

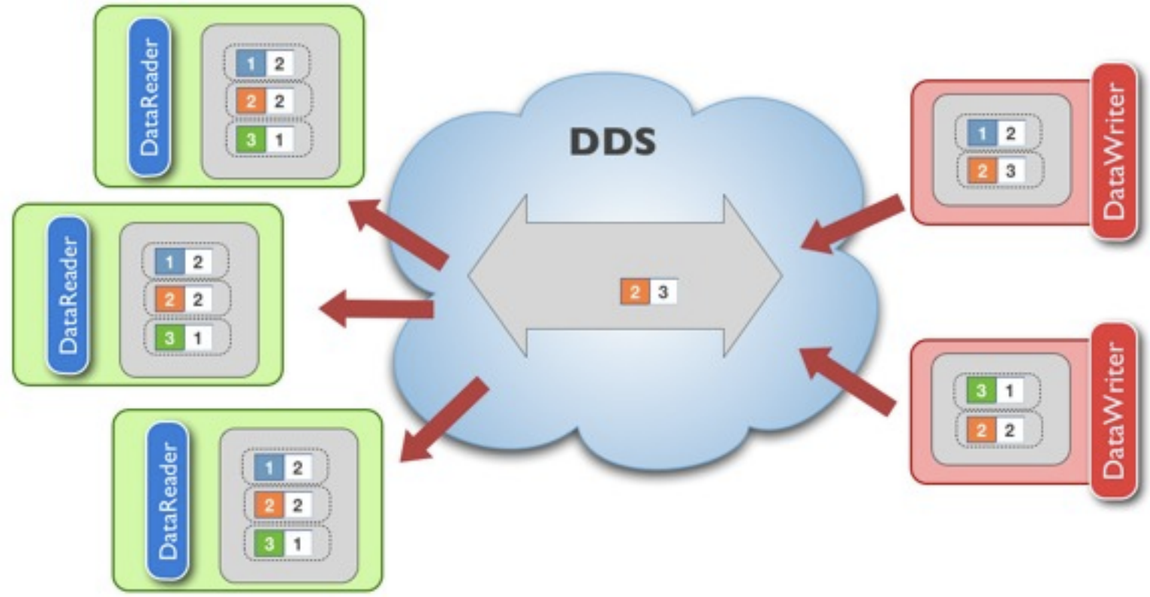


Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

Putting it All Together

The reliability with which data is delivered to applications is impacted in DDS by the following qualities of service

- RELIABILITY
  - BEST_EFORT
  - RELIABLE
- HISTORY
  - KEEP_LAST (K)
  - KEEP_ALL



- **Theoretically**, the only way to assure that **an application will see all the samples** produced by a writer is to use **RELIABLE+KEEP_ALL**. Any other combination could induce to samples being discarded on the receiving side because of the HISTORY depth

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Controlling Real-Time Properties

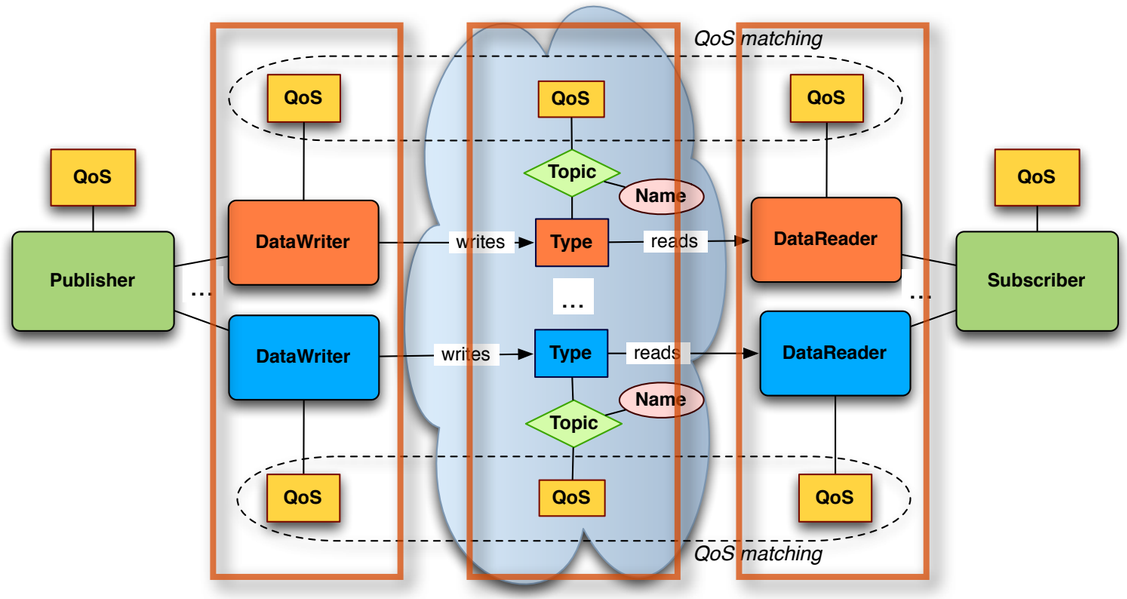
Deadline

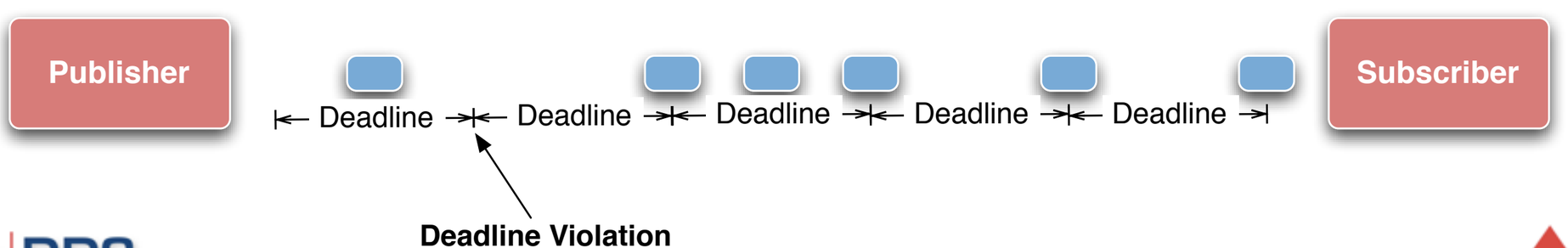
You can't be later than...

The **DEADLINE** QoS policy allows to define the **maximum inter-arrival time** between data samples

QoS Policy	Applicability	RxO	Modifiable
DEADLINE	T, DR, DW	Y	Y

- DataWriter indicates that the application commits to write a new value at least once every deadline period
- DataReaders are notified by the DDS when the DEADLINE QoS contract is violated





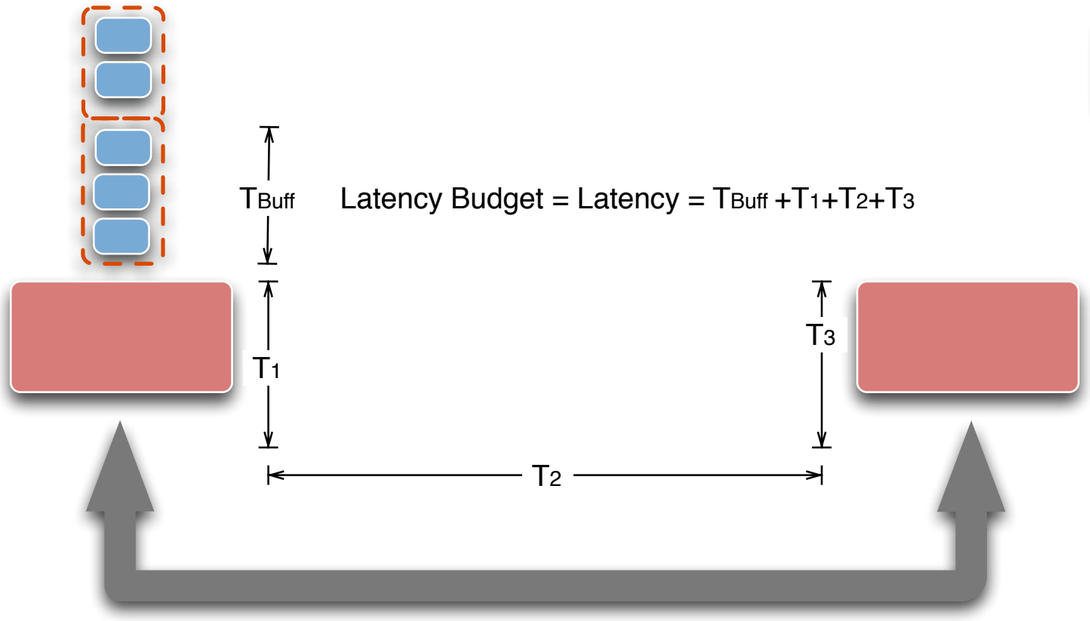
Latency Budget

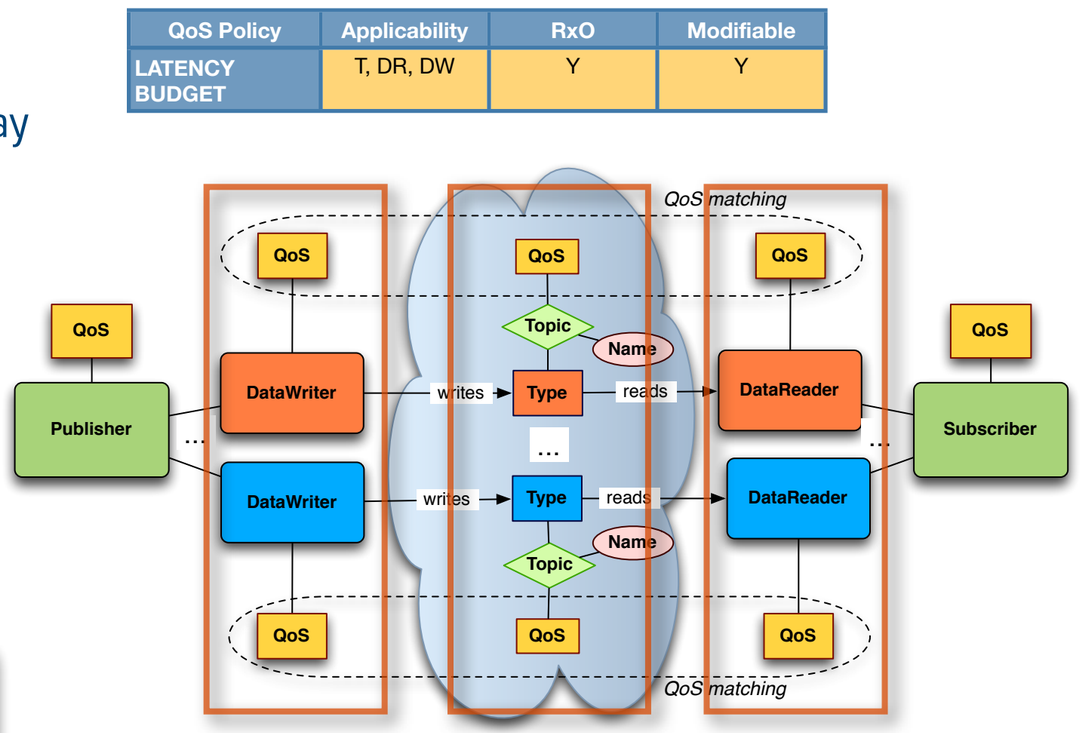
I need to get there in at most...

The **LATENCY_BUDGET QoS** policy specifies the maximum acceptable delay from the time the data is written until the data is inserted in the receiver's application-cache

QoS Policy	Applicability	RxO	Modifiable
LATENCY BUDGET	T, DR, DW	Y	Y

- The default value of the duration is zero indicating that the delay should be minimized
- This policy is a hint to the DDS, not something that must be monitored or enforced.



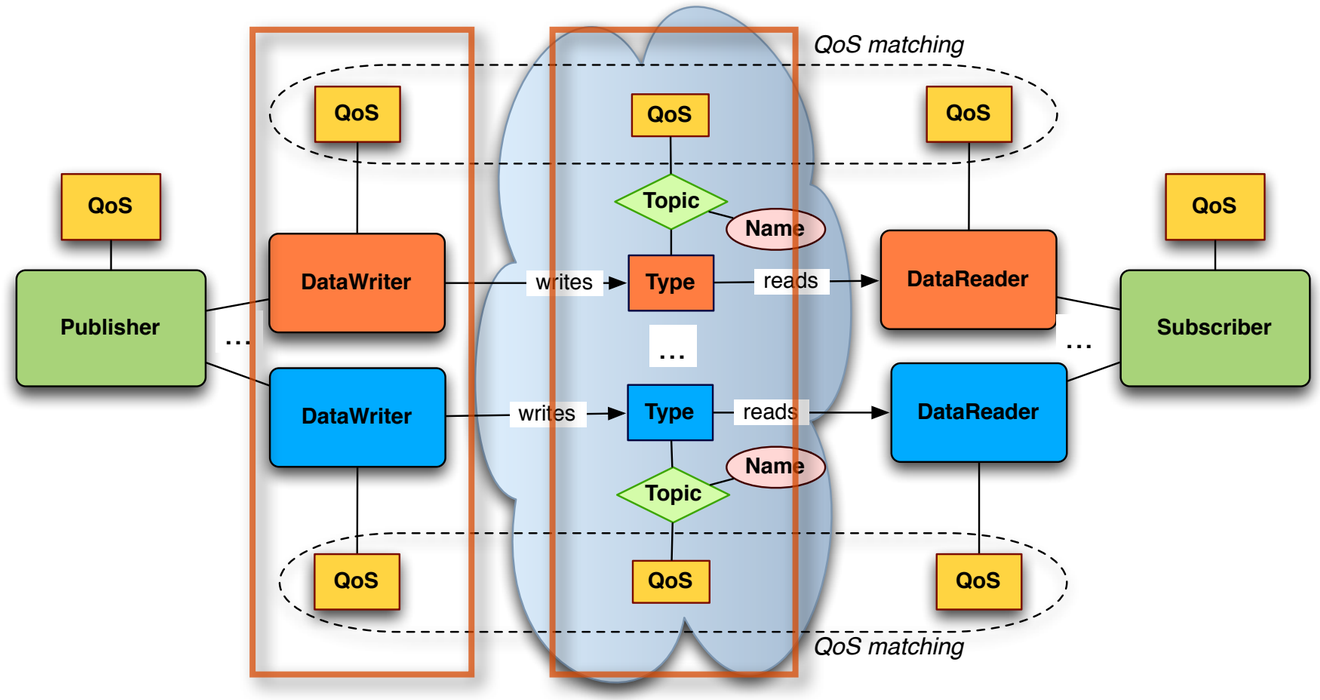


Transport Priority

VIP Data, stay clear!

QoS Policy	Applicability	RxO	Modifiable
TRANSPORT PRIORITY	T, DW	-	Y

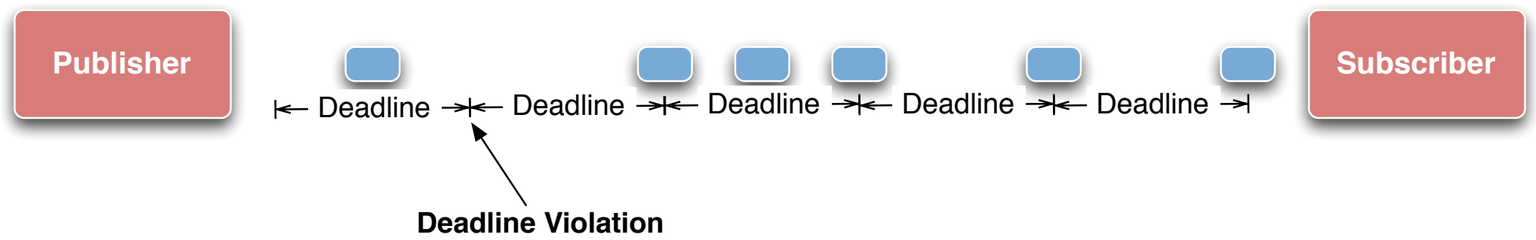
The **TRANSPORT_PRIORITY** QoS policy is a hint to the infrastructure as to how to set the priority of the underlying transport used to send the data.



Putting it all Together

The real-time properties with which data is delivered to applications is impacted in DDS by the following qualities of service:

- **TRANSPORT_PRIORITY**
- **LATENCY_BUDGET**



- In addition, DDS provides means for detecting performance failure, e.g., Deadline miss, by means of the **DEADLINE** QoS
- Given a periodic task-set {T} with periods Di (with $D_i < D_{i+1}$) and deadline equal to the period, than QoS should be set as follows:
  - Assign to each task T_i a TRANSPORT_PRIORITY Pi such that $P_i > P_{i+1}$
  - Set for each task T_i a DEADLINE QoS of D_i
  - For maximizing throughput and minimizing resource usage set for each Ti a LATENCY_BUDGET QoS between $D_i/2$ and $D_i/3$ (this is a rule of thumb, the upper bound is Di-(RTT/2))

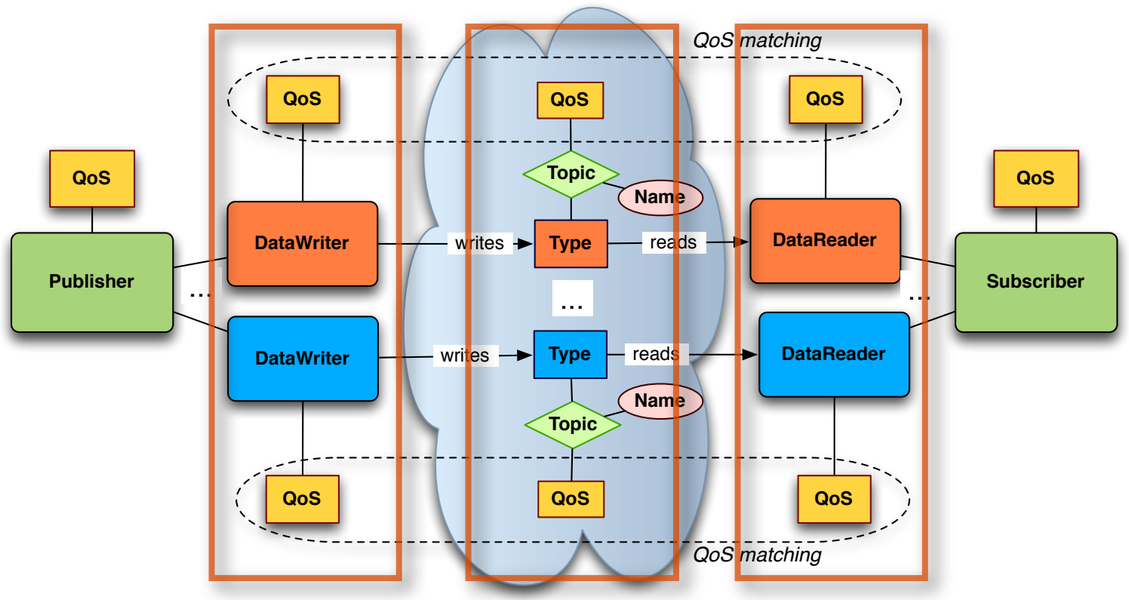
Controlling the Consistency Model

Durability

The **DURABILITY** QoS controls the data availability w.r.t. late joiners, specifically the DDS provides the following variants:

QoS Policy	Applicability	RxO	Modifiable
DURABILITY	T, DR, DW	Y	N
DURABILITY SERVICE	T, DW	N	N

- **Volatile.** No need to keep data instances for late joining data readers
- **Transient Local.** Data instance availability for late joining data reader is tied to the data writer availability
- **Transient.** Data instance availability outlives the data writer
- **Persistent.** Data instance availability outlives system restarts

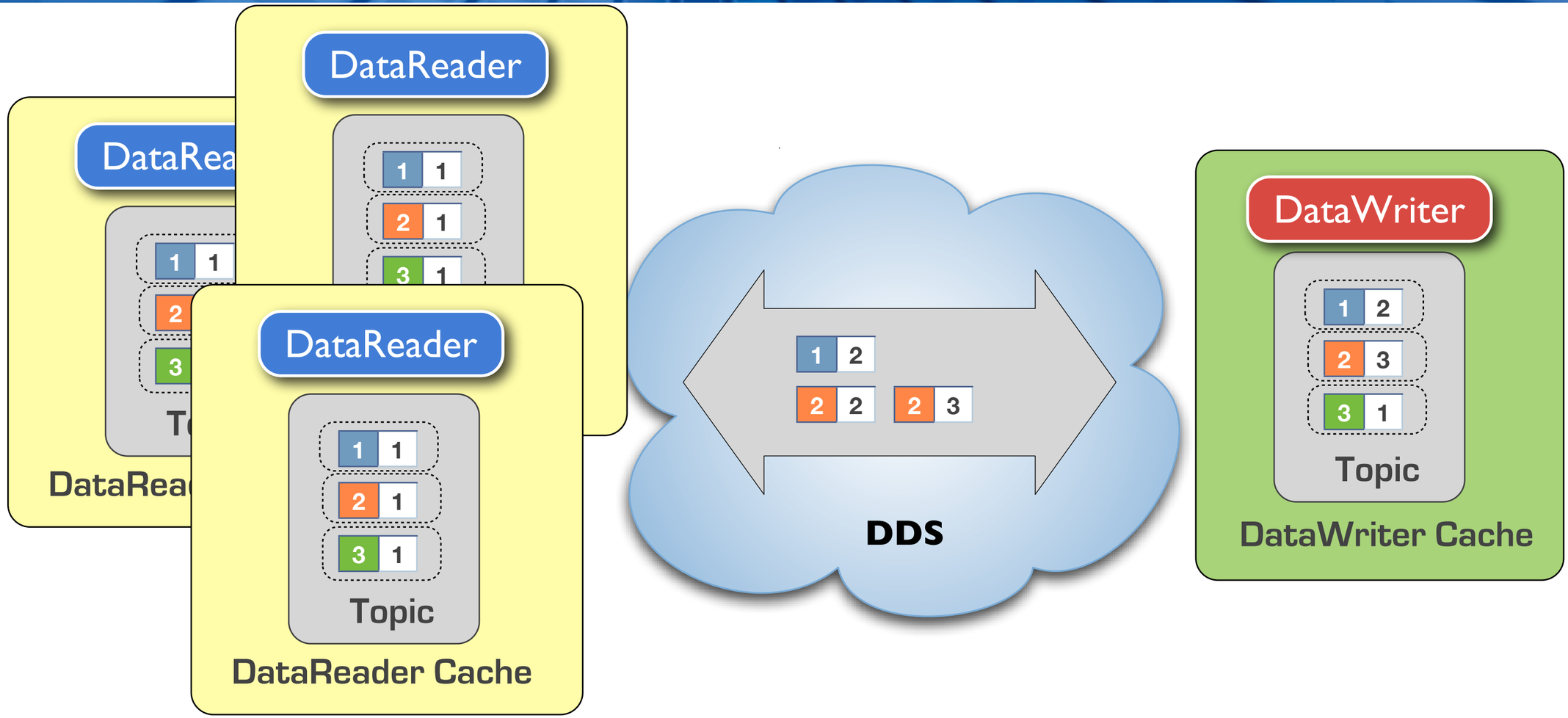


The **DURABILITY_SERVICE QoS** provide control over configuration of the service that implements the transient and persistent durability features

OpenSplice DDS

PRISMTECH

Eventual Consistency & R/W Caches



Under an Eventual Consistency Model, DDS guarantees that all matched Reader Caches will eventually be identical of the respective Writer Cache

QoS Impacting the Consistency Model

The DDS Consistency Model is a property that can be associated to Topics or further refined by Reader/Writers. The property is controlled by the following QoS Policies:

- **DURABILITY**
  - VOLATILE | TRANSIENT_LOCAL | TRANSIENT | PERSISTENT
- **LIFESPAN**
- **RELIABILITY**
  - RELIABLE | BEST_EFFORT
- **DESTINATION ORDER**
  - SOURCE_TIMESTAMP | DESTINATION_TIMESTAMP

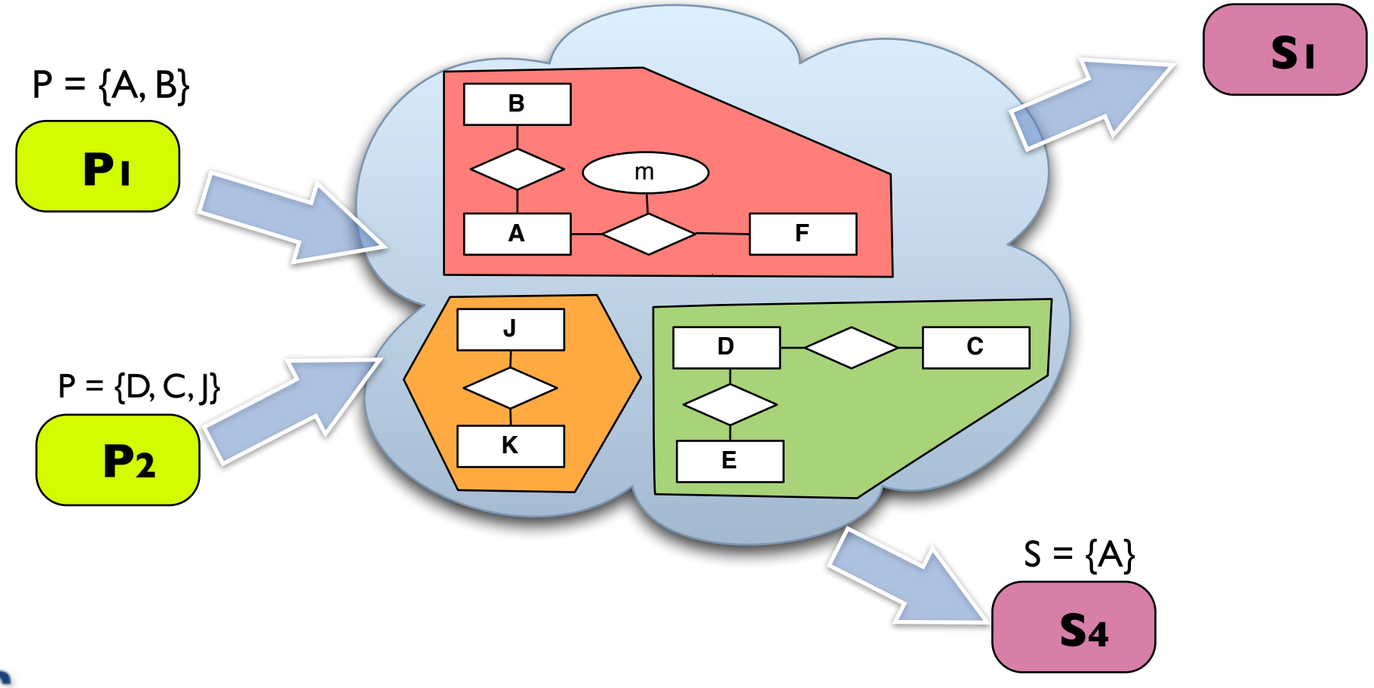
QoS Policy	Applicability	RxO	Modifiable
DURABILITY	T, DR, DW	Y	N
LIFESPAN	T, DW	-	Y
RELIABILITY	T, DR, DW	Y	N
DESTINATION ORDER	T, DR, DW	Y	N

QoS Impacting the Consistency Model

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN
Eventual Consistency (No Crash / Recovery)	VOLATILE	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Crash/Recovery)	PERSISTENT	RELIABLE	SOURCE_TIMESTAMP	INF.
Weak Consistency	ANY	ANY	DESTINATION_TIMESTAMP	ANY
Weak Consistency	ANY	BEST_EFFORT	ANY	ANY
Weak Consistency	ANY	ANY	ANY	N

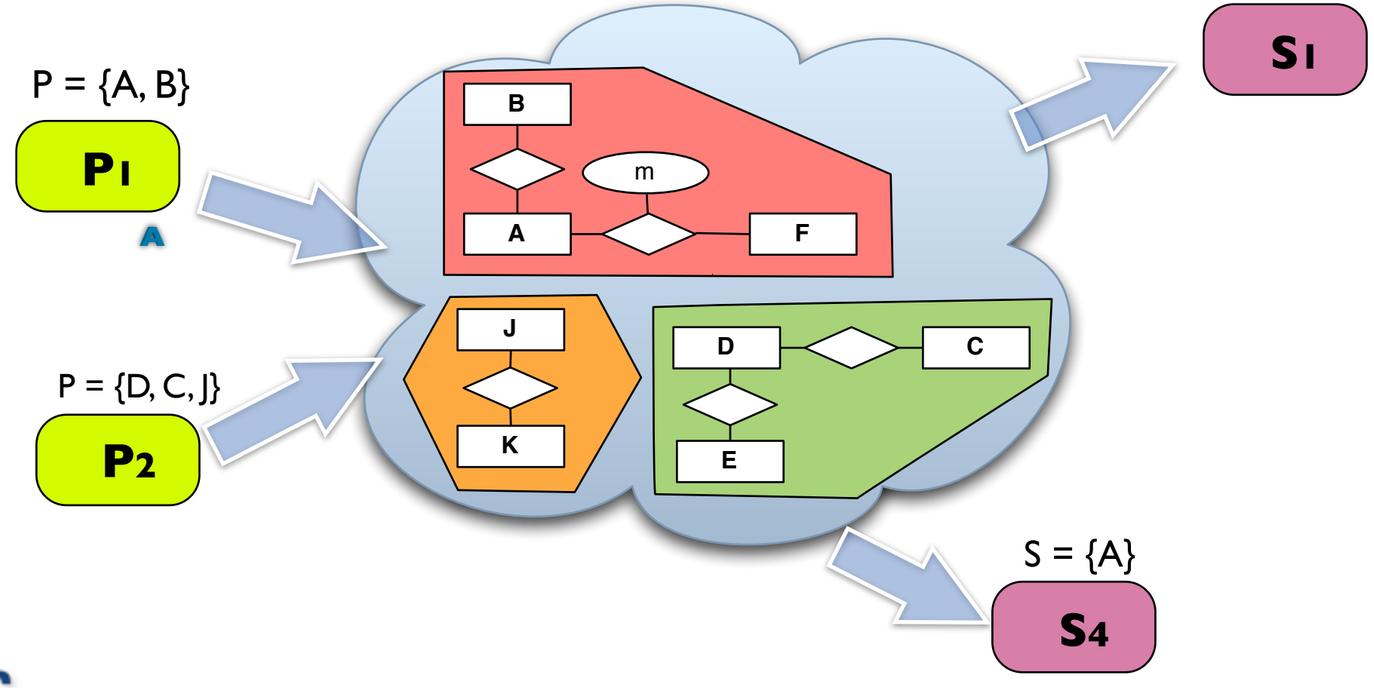
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



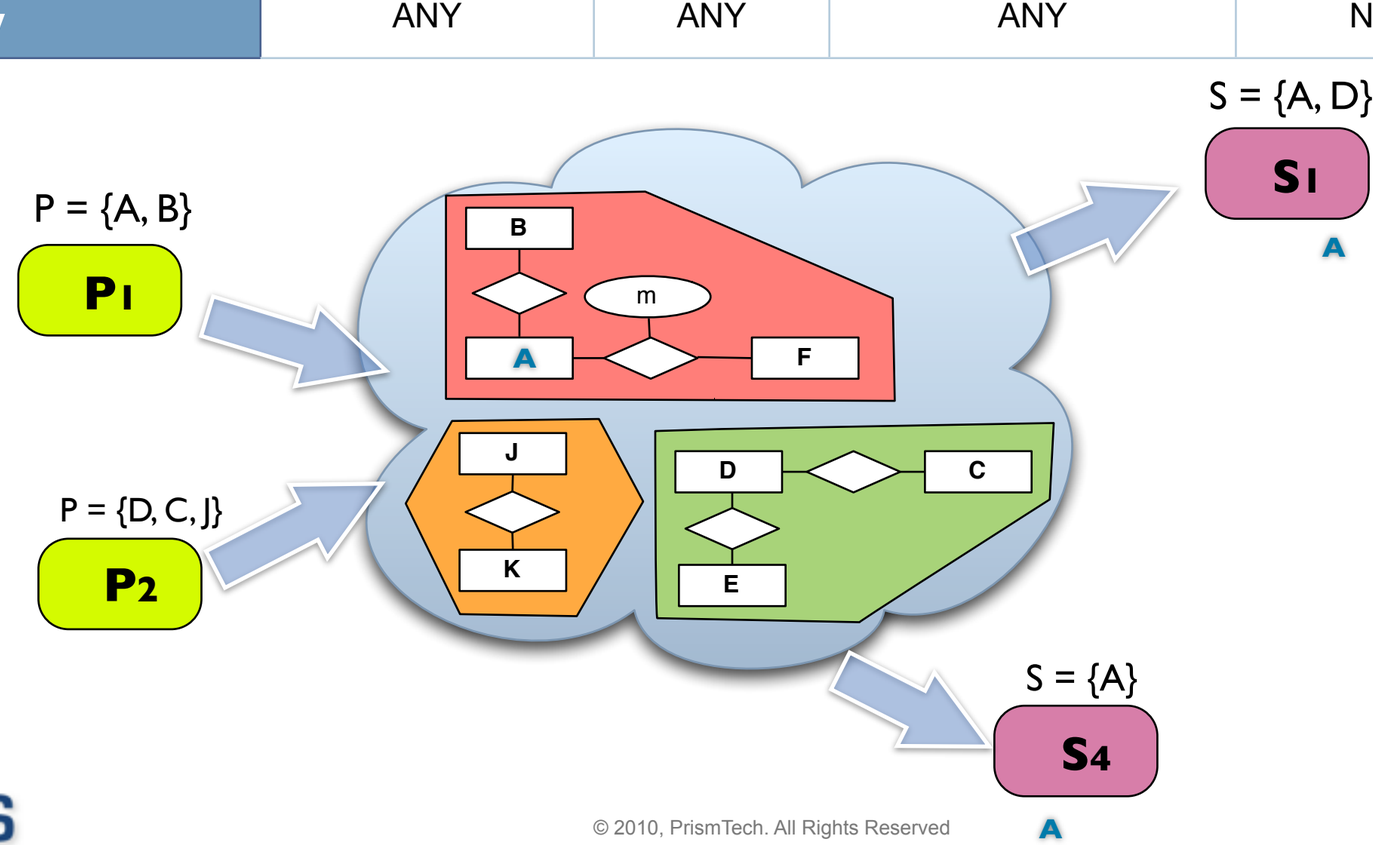
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



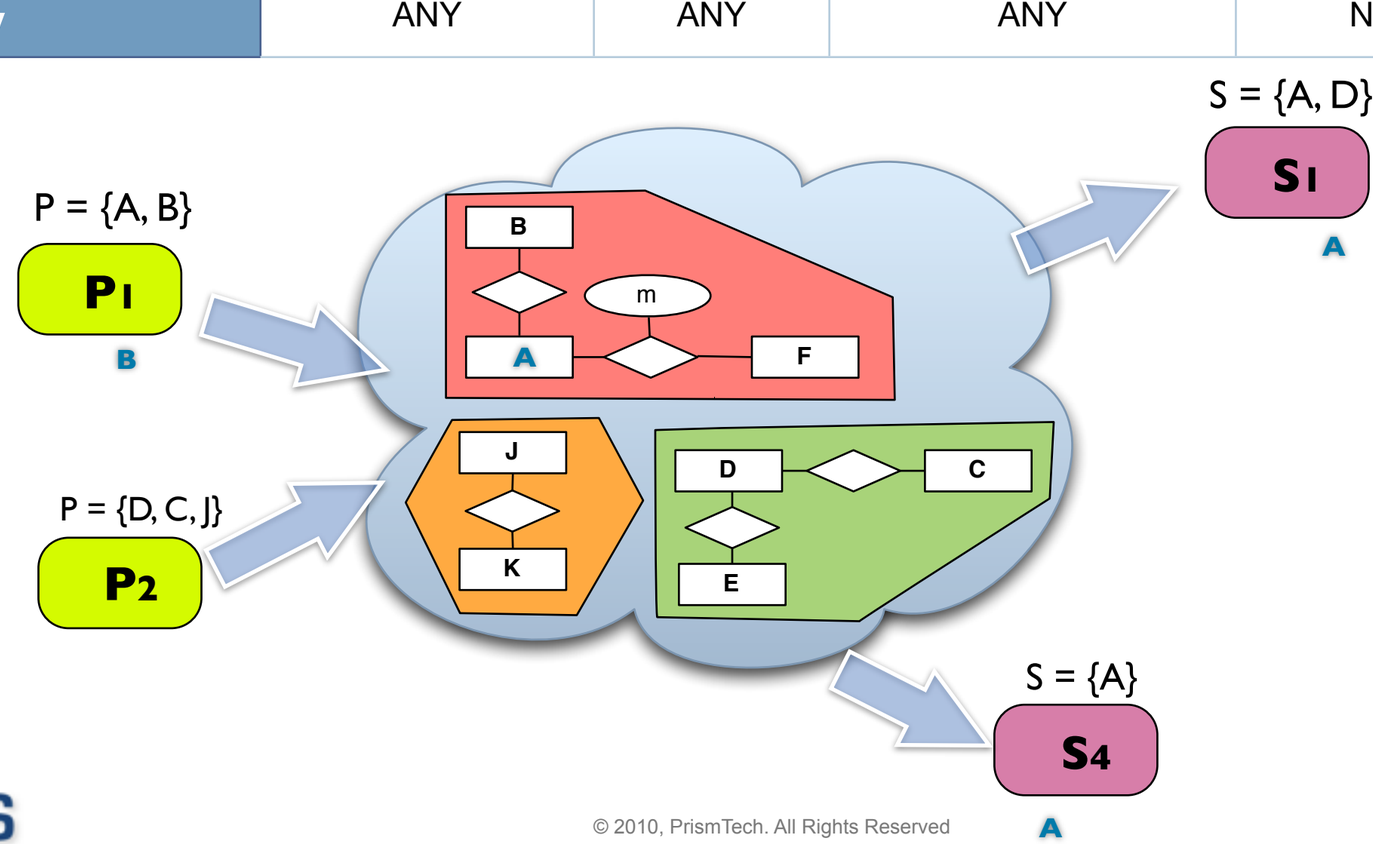
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



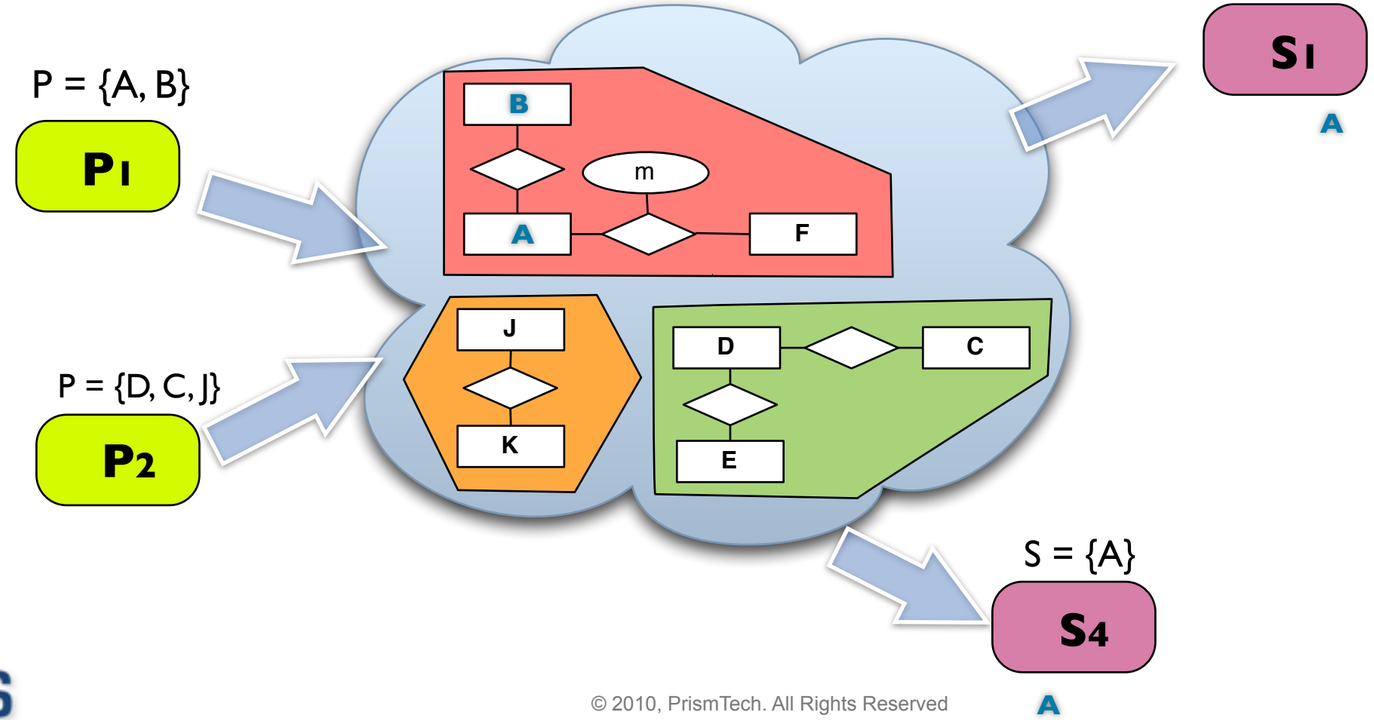
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



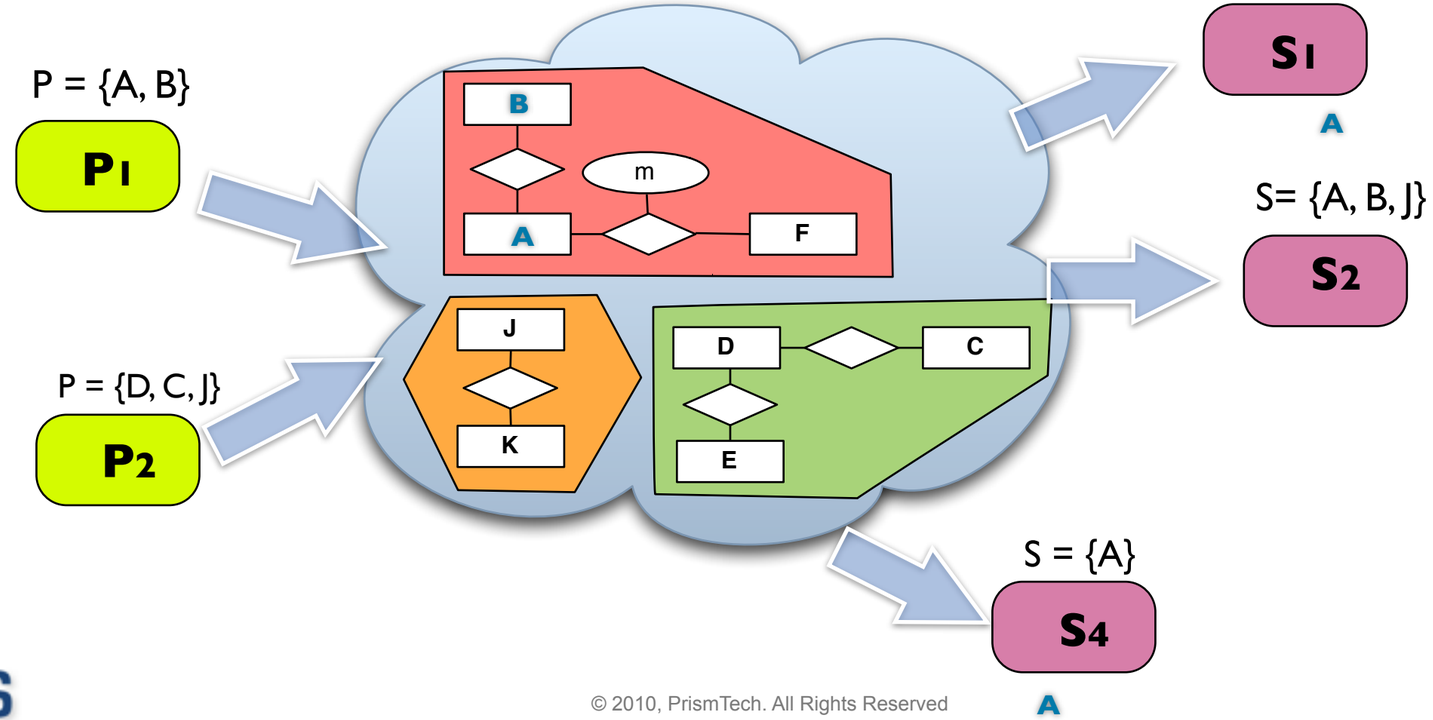
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



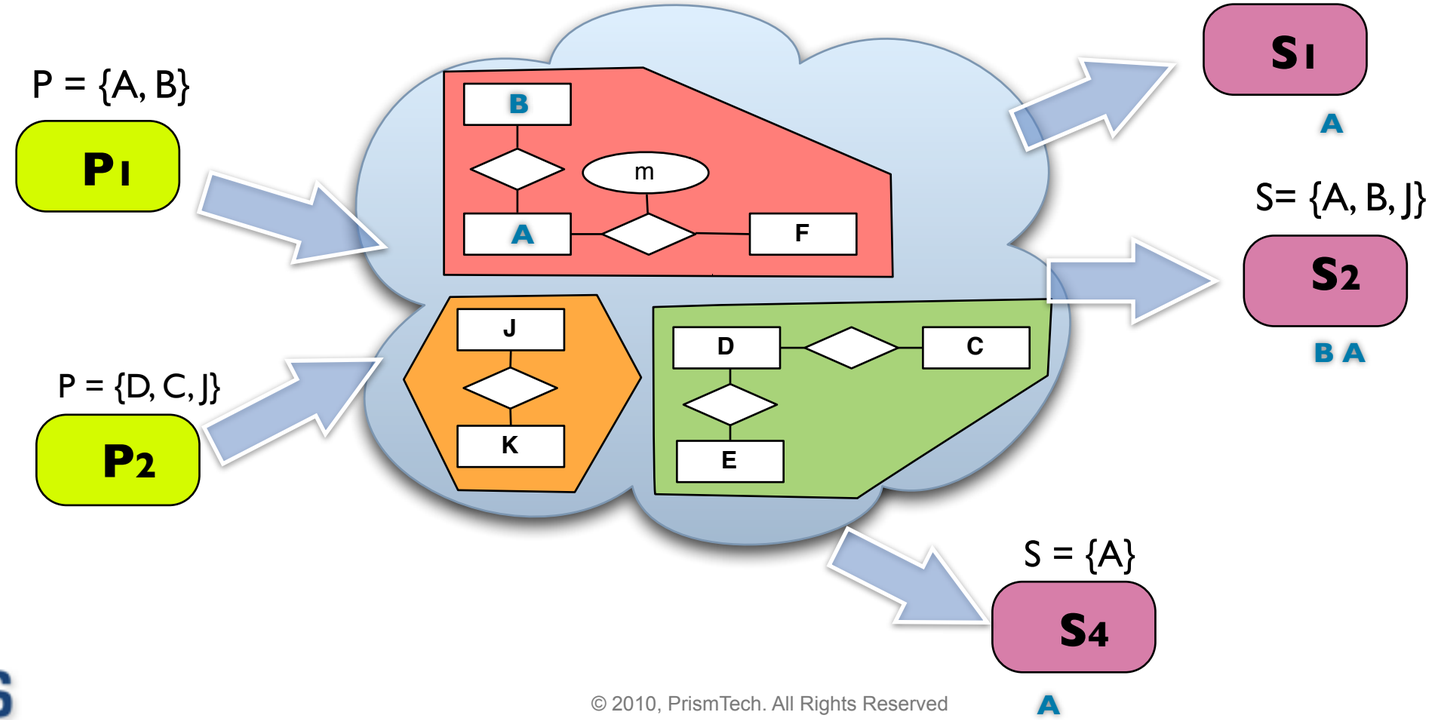
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



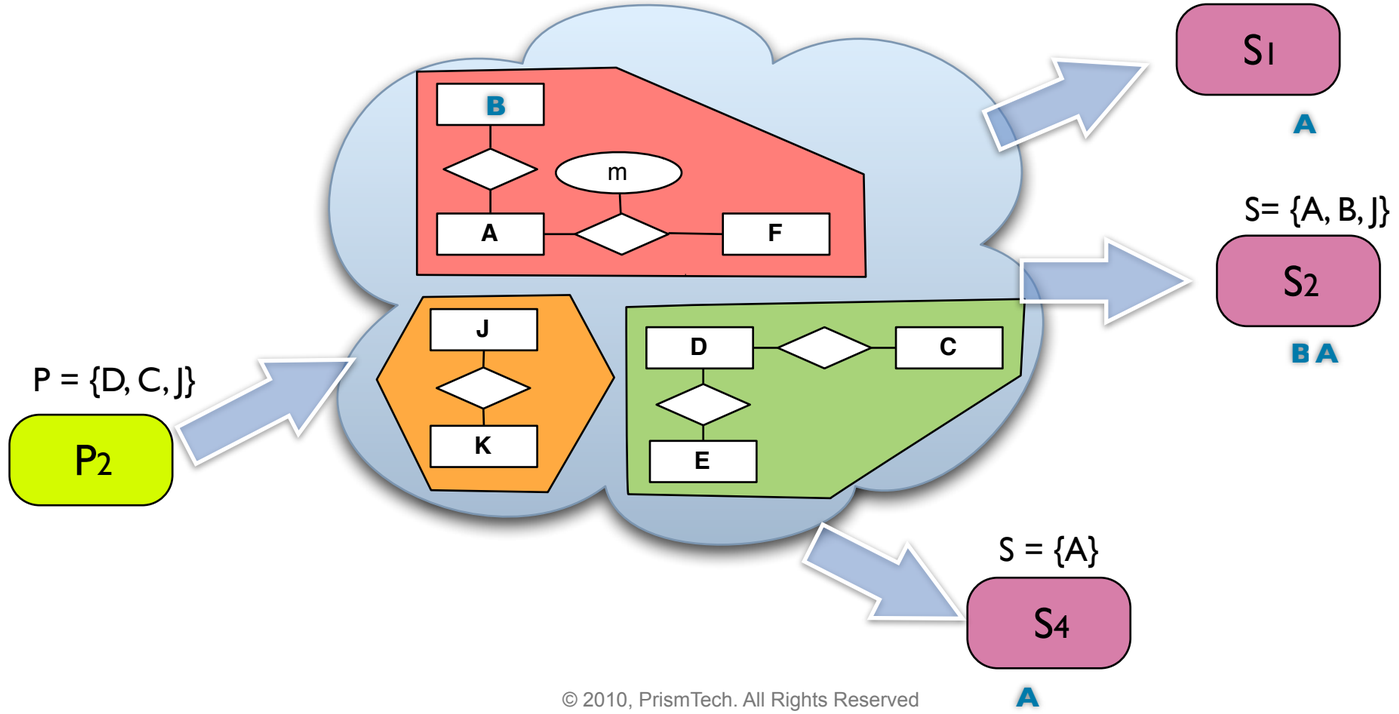
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



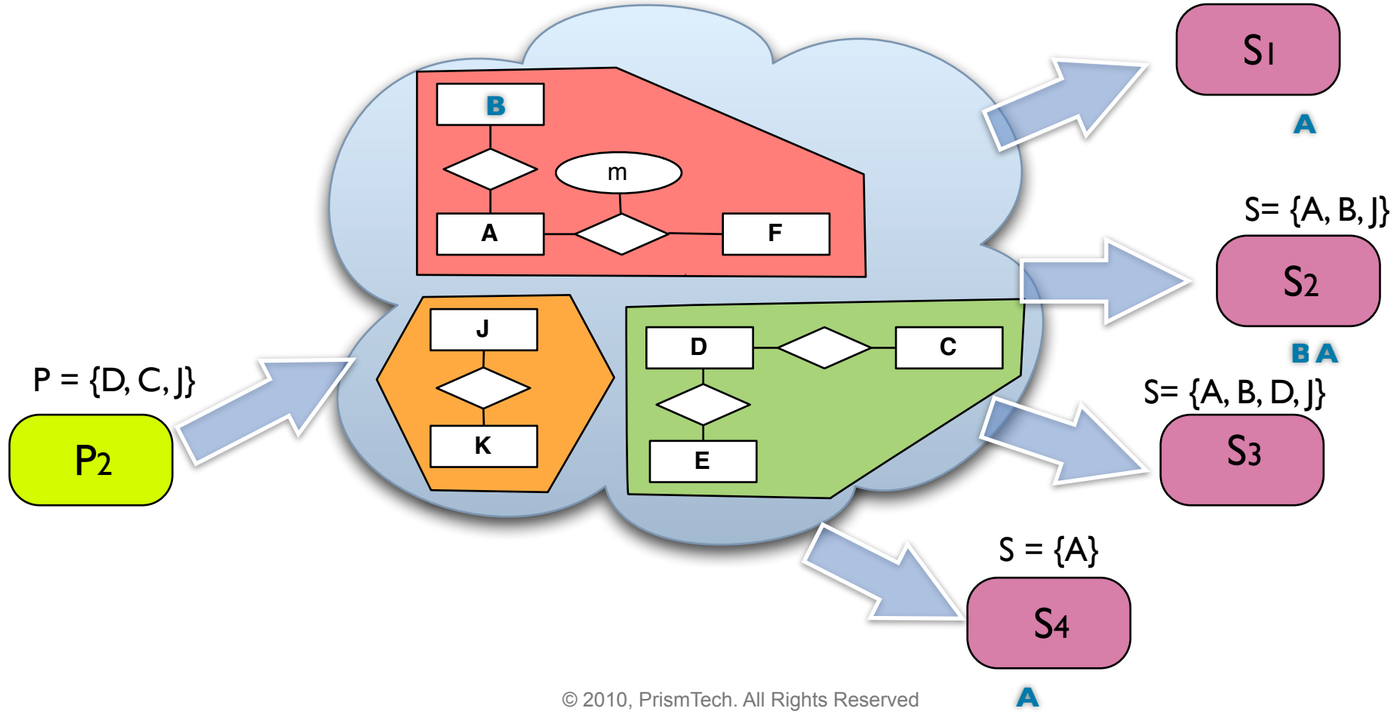
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



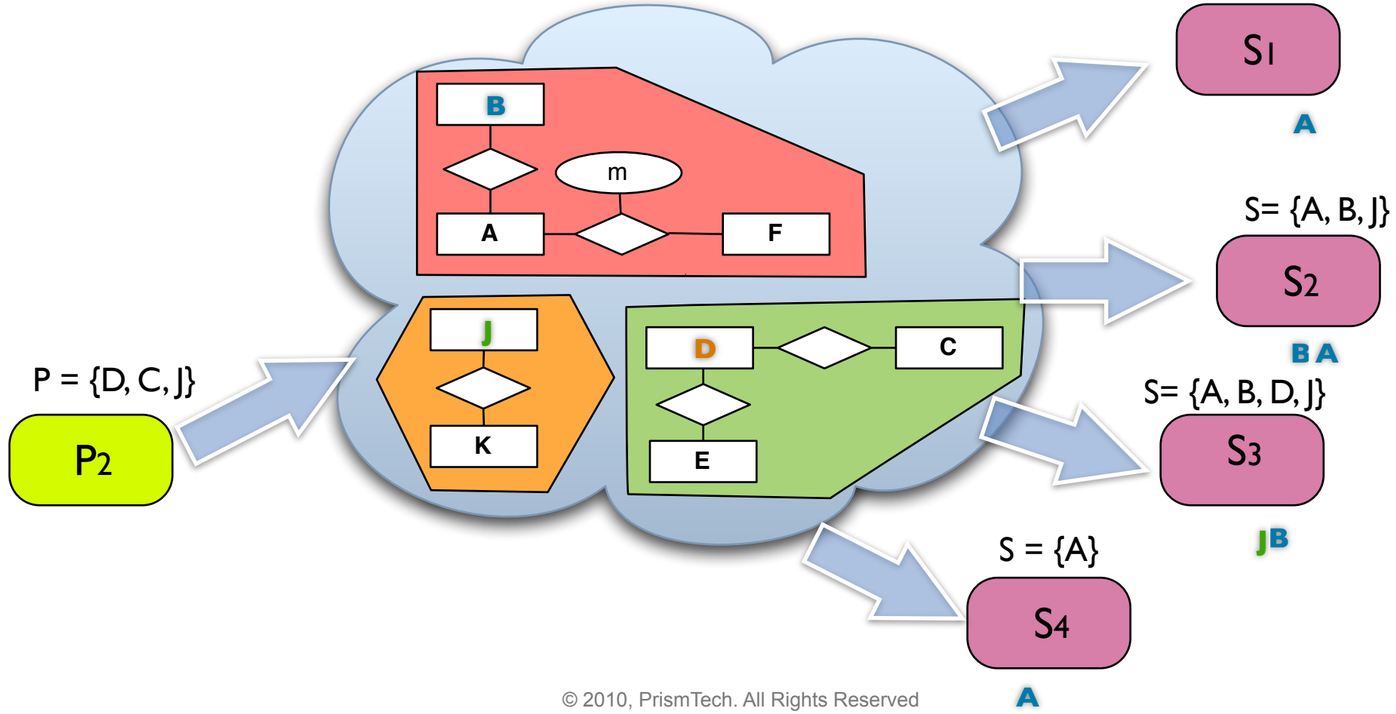
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



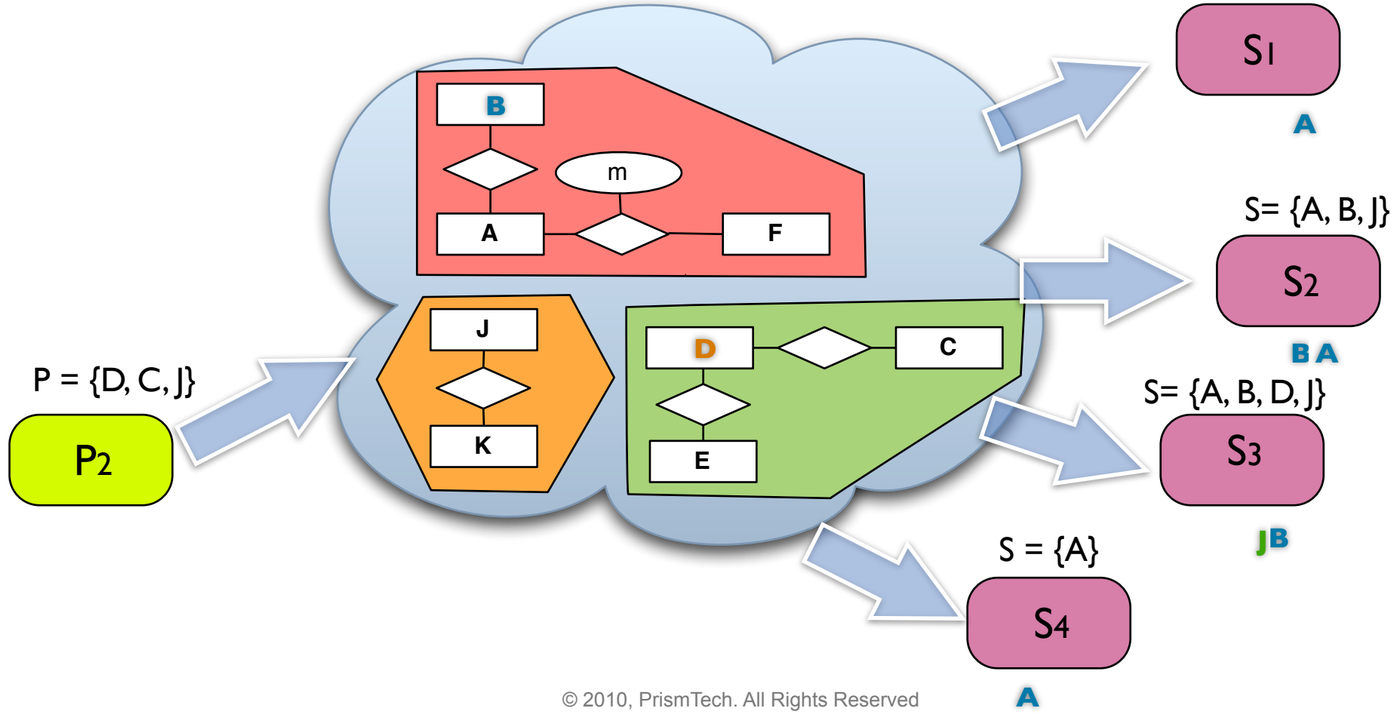
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



Controlling Replication

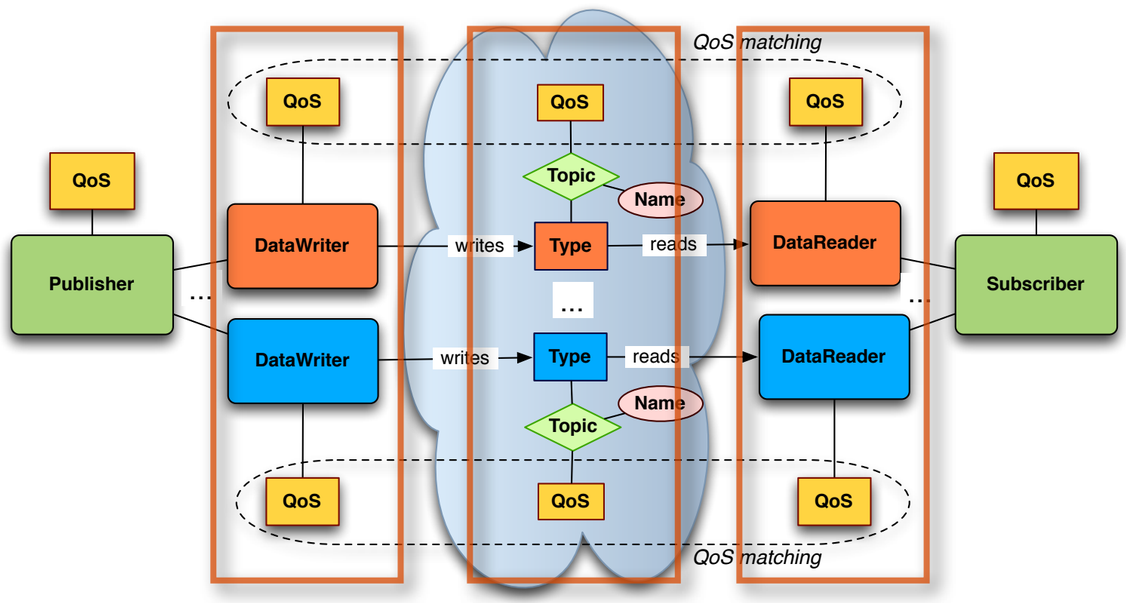
Ownership

Who owns the data?

The **OWNERSHIP** QoS specifies whether it is allowed for multiple DataWriters to write the same instance of the data and if so, how these modifications should be arbitrated. Possible choices are:

- **Shared.** Multiple writers are allowed to update the same instance and all the updates are made available to the reader
- **Exclusive.** Indicates that each instance can only be owned by one DataWriter, but the owner of an instance can change dynamically – due to liveliness changes
- The selection of the owner is controlled by the setting of the OWNERSHIP_STRENGTH **QoS policy**

QoS Policy	Applicability	RxO	Modifiable
OWNERSHIP	T, DR, DW	Y	N



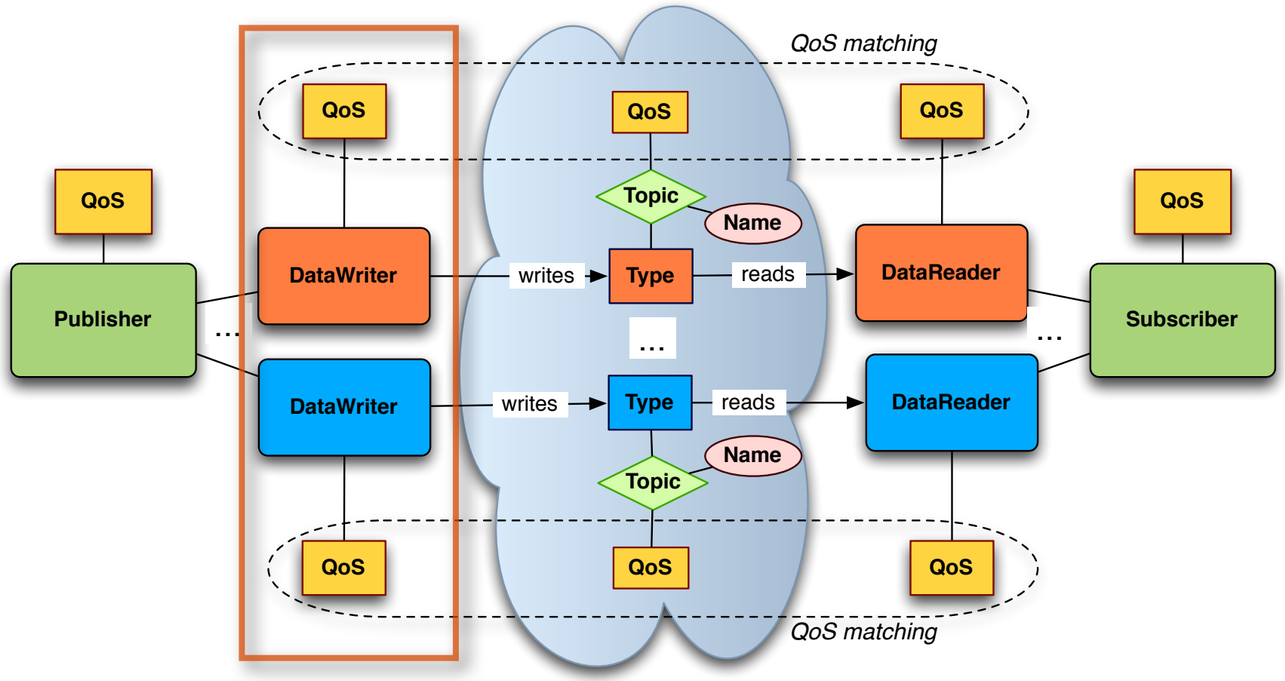
Ownership Strength

How strong are you?

The **OWNERSHIP_STRENGTH** Specifies the value of the "strength" used to arbitrate among DataWriters that attempt to modify the same data instance

- Data instance are identified by the couple (Topic, Key)
- The policy applies only if the OWNERSHIP is EXCLUSIVE

QoS Policy	Applicability	RxO	Modifiable
OWNERSHIP STRENGTH	DW	-	Y



Next Steps

- By now you've learned most of what you need to write complex DDS applications

- However, as the wise Confucius used to say:
  - I hear an I forget. I see and I remember. I do I understand.

- The best way of really getting into DDS is to write some DDS applications, utilities or extensions

Online Resources

OpenSplice DDS

Delivering Performance, Openness, and Freedom

- http://www.opensplice.com/
- emailto:opensplicedds@prismtech.com

slideshare

Present Yourself

- http://www.slideshare.net/angelo.corsaro

webex™

- http://bit.ly/1Sreg

twitter

- http://twitter.com/acorsaro/

YouTube

- http://www.youtube.com/OpenSpliceTube

Blogger

- http://opensplice.blogspot.com