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
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
Delivering Performance, Openness, and Freedom

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`
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.
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

\begin{align*}
\text{Condition} &::= \text{Predicate} \\
& \quad \mid \text{Condition} \text{ 'AND' Condition} \\
& \quad \mid \text{Condition} \text{ 'OR' Condition} \\
& \quad \mid \text{'NOT' Condition} \mid \text{ '(' Condition ')'} \\
\text{Predicate} &::= \text{ComparisonPredicate} \mid \text{BetweenPredicate} \\
\text{ComparisonPredicate} &::= \text{FIELDNAME RelOp Parameter} \mid \text{Parameter RelOp FIELDNAME} \\
\text{BetweenPredicate} &::= \text{FIELDNAME 'BETWEEN' Range} \mid \text{FIELDNAME 'NOT BETWEEN' Range} \\
\text{RelOp} &::= '=' \mid '>' \mid '>=' \mid '<' \mid '<=' \mid '<>' \mid \text{like} \\
\text{Range} &::= \text{Parameter 'AND' Parameter} \\
\text{Parameter} &::= \text{INTEGERVALUE} \mid \text{FLOATVALUE} \\
& \quad \mid \text{STRING} \mid \text{ENUMERATEDVALUE} \mid \text{PARAMETER} \\
\text{where} \text{ PARAMETER has the for } \%n \text{ (with } n \text{ in } 0..100)
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
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);
};
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

ReaderCache

(DataReader)

ContentFilteredTopic

(temp < 30) AND (hum < 0.6)

QueryCondition

(DataReader)

(temp < 30) AND (hum < 0.6)

ReaderCache

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
Filters vs Queries

(DataReader)

ReaderCache

1 26 0.4 C
1 28 0.5 C
1 29 0.5 C

(ContentFilteredTopic)

ReaderCache

(DataReader)

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

(QueryCondition)

(temp < 30) AND (hum < 0.6)
Filters vs Queries

read()

DataReader

1 26 0.4 C
1 28 0.5 C
1 29 0.5 C

ReaderCache

(temp < 30) AND (hum < 0.6)

ContentFilteredTopic

read_w_condition()

QueryCondition

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

DataReader

ReaderCache

(temp < 30) AND (hum < 0.6)
Your will learn:
- What can you control with QoS
- Request vs. Offered in DDS
- QoS Patterns
Anatomy of a DDS Application

```
struct TempSensor {
    int tID;
    float temp;
    float humidity;
    TemperatureScale scale;
};
```

```
#pragma keylist TempSensor tID
```
Anatomy of a DDS Application

Arrows show structural relationships, not data-flows

struct TempSensor {
    int tID;
    float temp;
    float humidity;
    TemperatureScale scale;
};

#pragma keylist TempSensor tID
Anatomy of a DDS Application

Arrows show structural relationships, not data-flows

struct TempSensor {
    int tID;
    float temp;
    float humidity;
    TemperatureScale scale;
};

#pragma keylist TempSensor tID

Samples

<table>
<thead>
<tr>
<th>Topic</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21 0.6 C</td>
</tr>
<tr>
<td>2</td>
<td>20 0.6 F</td>
</tr>
<tr>
<td>3</td>
<td>75 0.1 K</td>
</tr>
</tbody>
</table>

| 1     | 22 0.6 C  |
| 2     | 19 0.6 F  |
| 3     | 8 0.1 K   |
| 3     | 95 0.2 K  |

DataWriter

DataReader

Publisher

Subscriber

Domain Participant

Domain
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

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

- 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
Controlling 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.
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

**QoS Policy**

<table>
<thead>
<tr>
<th>History Depth</th>
<th>DDS Default</th>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HISTORY</td>
<td>T, DR, DW</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Diagram:**

- Publisher
- DataWriter
- Topic
- DataReader
- Subscriber
- QoS matching

**DataReader**

<table>
<thead>
<tr>
<th>History Depth = 1</th>
<th>History Depth = 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 2 3</td>
<td>1 1 2 3 4 1 5</td>
</tr>
<tr>
<td>2 1</td>
<td>2 1 2 3 2 4 2 5</td>
</tr>
<tr>
<td>3 1</td>
<td>3 1 3 2 3 3 4 3 5</td>
</tr>
</tbody>
</table>

© 2010, PrismTech. All Rights Reserved
struct Counter {
    int cID;
    int count;
};
#pragma keylist Counter cID
History in Action

Note: The Reliability QoS controls whether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders.
**Note:** The Reliability QoS controls whether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders.
History in Action

History Depth = 1 (DDS Default)

DataReader Cache

DataReader

DataWriter Cache

DataWriter

Network

Note: The Reliability QoS controls whether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders
Note: The Reliability QoS controls whether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders.
Note: The Reliability QoS controls whether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders.
Note: The Reliability QoS controls whether 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.
Controlling Real-Time Properties
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
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.

- 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.

\[ \text{Latency Budget} = \text{Latency} = T_{\text{Buf}} + T_1 + T_2 + T_3 \]
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 \( D_i \) (with \( D_i < D_{i+1} \)) and deadline equal to the period, then QoS should be set as follows:

- Assign to each task \( T_i \) a **TRANSPORT_PRIORITY** \( P_i \) 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 \( T_i \) a **LATENCY_BUDGET** QoS between \( D_i/2 \) and \( D_i/3 \) (this is a rule of thumb, the upper bound is \( D_i - (RTT/2) \))
Durability

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

- **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

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURABILITY</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DURABILITY SERVICE</td>
<td>T, DW</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

The **DURABILITY_SERVICE** QoS provide control over configuration of the service that implements the transient and persistent durability features.
Under an Eventual Consistency Model, DDS guarantees that all matched Reader Caches will eventually be identical of the respective Writer Cache.
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

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURABILITY</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LIFESPAN</td>
<td>T, DW</td>
<td>-</td>
<td>Y</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>DESTINATION ORDER</td>
<td>T, DR, DW</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
### QoS Impacting the Consistency Model

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (No Crash / Recovery)</td>
<td>VOLATILE</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Reader Crash / Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>PERSISTENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>DESTINATION_TIMESTAMP</td>
<td>ANY</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>BEST_EFFORT</td>
<td>ANY</td>
<td>ANY</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

- P = {A, B}  
- P = {D, C, J}  
- S = {A}  
- S = {A, D}  

**OpenSplice DDS**

© 2010, PrismTech. All Rights Reserved
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Reader Crash / Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>
### Eventual Consistency @ Work

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Reader Crash / Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

- Eventual Consistency (Reader Crash / Recovery) for set `{A}`
- Eventual Consistency (Crash/Recovery) for set `{B}`
- Weak Consistency for set `{J}`

**Diagram:**

- **P = {A, B}**
  - Node `B`
  - Node `A`
  - Node `m`
- **P = {D, C, J}**
  - Node `J`
  - Node `D`
  - Node `C`
  - Node `E`
- **S = {A, D}**
  - Node `S1`
- **S = {A}**
  - Node `S4`
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Reader</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Crash / Recovery)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eventual Consistency (Crash/</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Recovery)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

\[ P = \{A, B\} \]

\[ S = \{A, D\} \]

\[ P = \{D, C, J\} \]

\[ S = \{A\} \]
### Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P = {A, B}</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>P = {D, C, J}</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
<tr>
<td>S = {A}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S = {B}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S = {J}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:**

- **P1:** \(P = \{A, B\}\)
- **P2:** \(P = \{D, C, J\}\)
- **S1:** \(S = \{A, D\}\)
- **S4:** \(S = \{A\}\)
### Eventual Consistency @ Work

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Reader Crash / Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

- **P** = {A, B}
- **P** = {D, C, J}
- **S** = {A}
- **S** = {A, B, J}
- **S** = {A, D}
- **S** = {A, 4}

---

![Eventual Consistency Diagram](image-url)
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

\[ P = \{A, B\} \]

\[ S = \{A, D\} \]

\[ P = \{D, C, J\} \]

\[ S = \{A, B, J\} \]

\[ S = \{A\} \]

\[ S = \{A, D\} \]
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th></th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Reader Crash / Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>
## Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

- \( S = \{A\} \)
- \( S = \{B\} \)
- \( S = \{J\} \)

- \( P = \{D, C, J\} \)
- \( S = \{A, D\} \)
- \( S = \{A, B, J\} \)
- \( S = \{A, B, D, J\} \)
- \( S = \{A\} \)
- \( S = \{A\} \)

### Eventual Consistency (Crash/Recovery)

- **Weak Consistency**

- **Eventual Consistency (Crash/Recovery)**

- **Eventual Consistency (Reader Crash / Recovery)**
### Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

- $S = \{A\}$
- $\{B\}$
- $\{J\}$

**Diagram:**

- $S = \{A, D\}$
- $S = \{A, B, J\}$
- $S = \{A, B, D, J\}$
- $S = \{A\}$
- $S = \{A, D, J\}$

**Notation:**

- $P = \{D, C, J\}$
- $S_1$
- $S_2$
- $S_3$
- $S_4$
Eventual Consistency @ Work

<table>
<thead>
<tr>
<th>Eventual Consistency (Reader Crash / Recovery)</th>
<th>DURABILITY</th>
<th>RELIABILITY</th>
<th>DESTINATION_ORDER</th>
<th>LIFESPAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventual Consistency (Crash/Recovery)</td>
<td>TRANSIENT_LOCAL</td>
<td>RELIABLE</td>
<td>SOURCE_TIMESTAMP</td>
<td>INF.</td>
</tr>
<tr>
<td>Weak Consistency</td>
<td>ANY</td>
<td>ANY</td>
<td>ANY</td>
<td>N</td>
</tr>
</tbody>
</table>

S = \{A, B, D, J\}

S = \{A, D\}

S = \{A, B, J\}

S = \{A, B, D, J\}

S = \{A\}

S = \{A\}

P = \{D, C, J\}

S1

S2

S3

S4

\[ S = \{A, D\} \]

\[ S = \{A, B, J\} \]

\[ S = \{A, B, D, J\} \]

\[ S = \{A\} \]
Controlling Replication
Ownership

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.
Ownership Strength

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

<table>
<thead>
<tr>
<th>QoS Policy</th>
<th>Applicability</th>
<th>RxO</th>
<th>Modifiable</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWNERSHIP STRENGTH</td>
<td>DW</td>
<td>-</td>
<td>Y</td>
</tr>
</tbody>
</table>

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