OpenSplice DDS

Delivering Performance, Openness, and Freedom

Angelo Corsaro, Ph.D.
Product Strategy & Marketing Manager
OMG RTESS and DDS SIG Co-Chair
angelo.corsaro@prismtech.com

PRISMTECH
Powering Netcentricity

A Crash Course

Focus of the Day

What You'll See

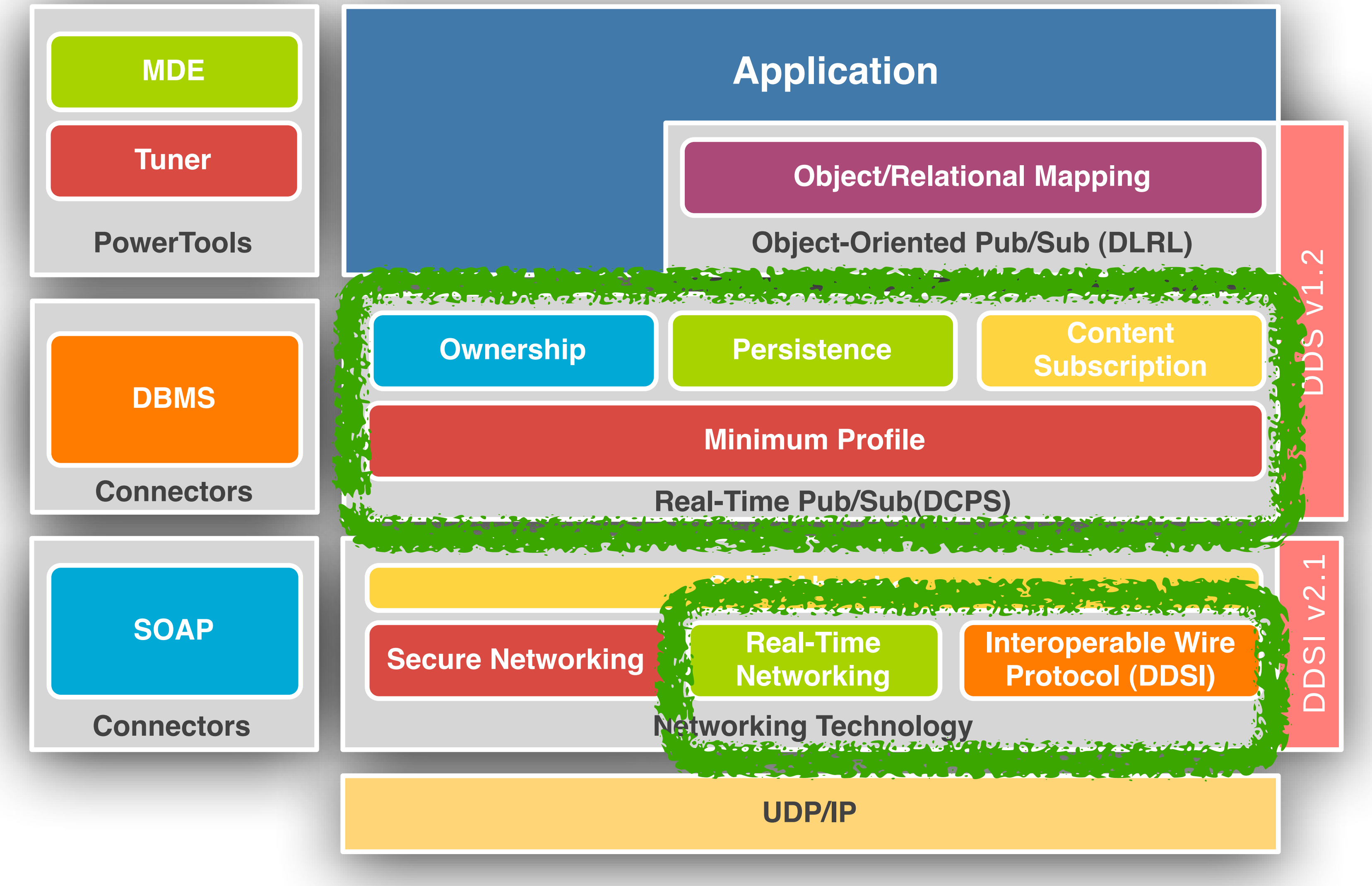
- Get up-to-speed in developing applications with the OpenSplice DDS v4.1 Community Edition
- Understanding the key concepts behind DDS Programming and Open Splice DDS

What You'll Need

- OpenSplice DDS v4.1 (from opensplice.org)
- Linux Distro with GCC v4.1 or higher
- Some C++ skills

open source

LGPL v3 Free Software



MDE

Tuner

PowerTools

DBMS

Connectors

SOAP

Connectors

Application

Object/Relational Mapping

Object-Oriented Pub/Sub (DLRL)

Ownership

Persistence

Content Subscription

Minimum Profile

Real-Time Pub/Sub(DCPS)

DDS v1.2

Secure Networking

Real-Time Networking

Interoperable Wire Protocol (DDSI)

Networking Technology

DDSI v2.1

UDP/IP

OpenSplice|DDS

Delivering Performance, Openness, and Freedom

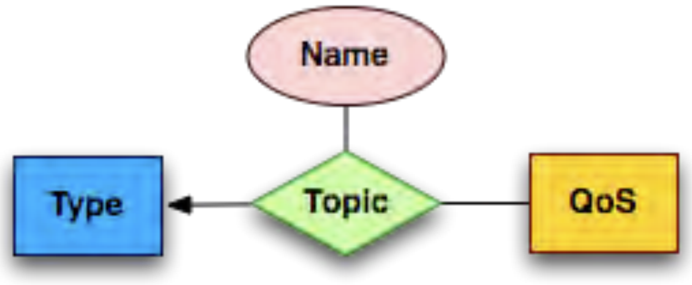
A little bit of Theory

Topics, Partitions, and Domains

Topics -- The unit of information

- **Topics**. Unit of information exchanged between Publisher and Subscribers.
- **Data Types**. Type associated to a Topic must be a structured type expressed in IDL
- **Topic Instances**. Key values in a datatype uniquely identify a Topic Instance (like rows in table)
- **Content Awareness**. SQL Expressions can be used to do content-aware subscriptions, queries, joins, and correlate topic instances

Topic



Topic Type

```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor tID
```

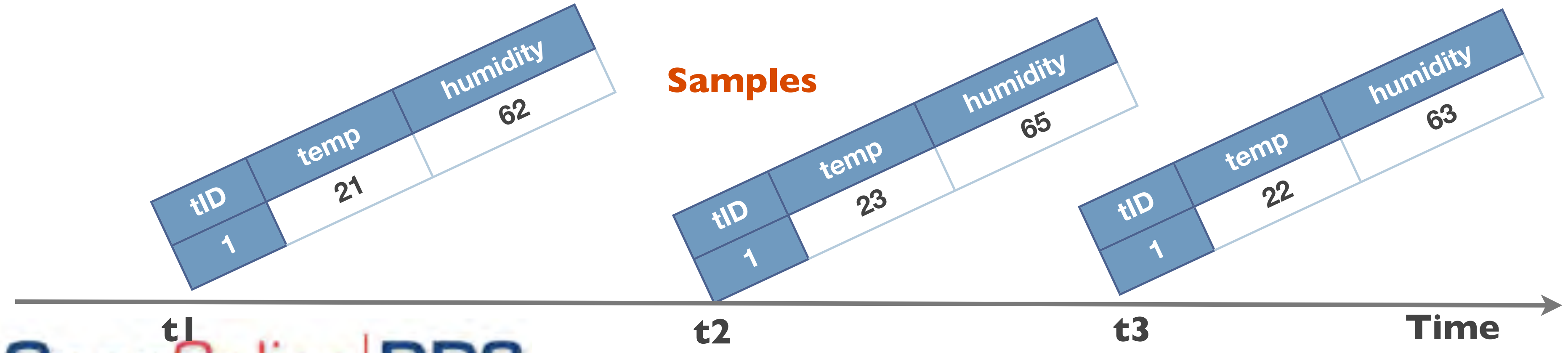
Instances

TempSensor

tID	temp	humidity
1	21	62
2	27	78
3	25.5	72.3

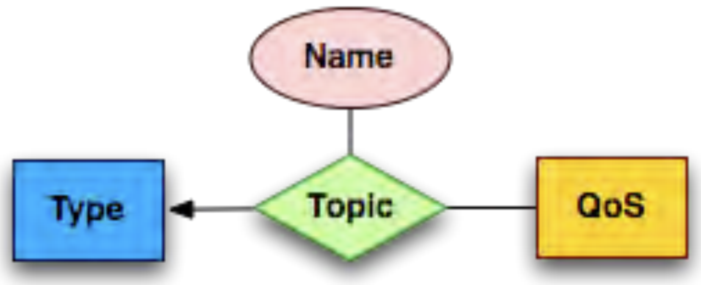
SELECT * FROM TempSensor t
WHERE t.temp > 25

tID	temp	humidity
2	27	78
3	25.5	72.3



Keyless vs. Keyed Topics

Topic



Keyless Topics

Topic Type

```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor
```

TempSensor

tID	temp	humidity
1	21	62

Keyed Topics

Topic Type

```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor tID
```

Instances

TempSensor

tID	temp	humidity
1	21	62
2	23	78
3	21.5	72.3

Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

OpenSplice DDS

PrismTech

Keyless vs. Keyed Topics

Topic Type

```
struct StockQuote {
   char<64> name;
   float    quote;
   char<4>  exchange;
};
#pragma keylist StockQuote
```

Topic “AAPL”

name	exchange	quote
Apple Inc.	NASD	165.37

Topic “GOOG”

name	exchange	quote
Google Inc.	NASD	663.97

Topic “MSFT”

name	exchange	quote
Microsoft Corp.	NASD	33.73

Keyless-Topics have only one Instance per Topic

Topic Type

```
struct StockQuote {
  char<4>  symbol;
   char<64> name;
   float    quote;
   char<4>  exchange;
};
#pragma keylist StockQuote symbol
```

Topic “StockQuote”

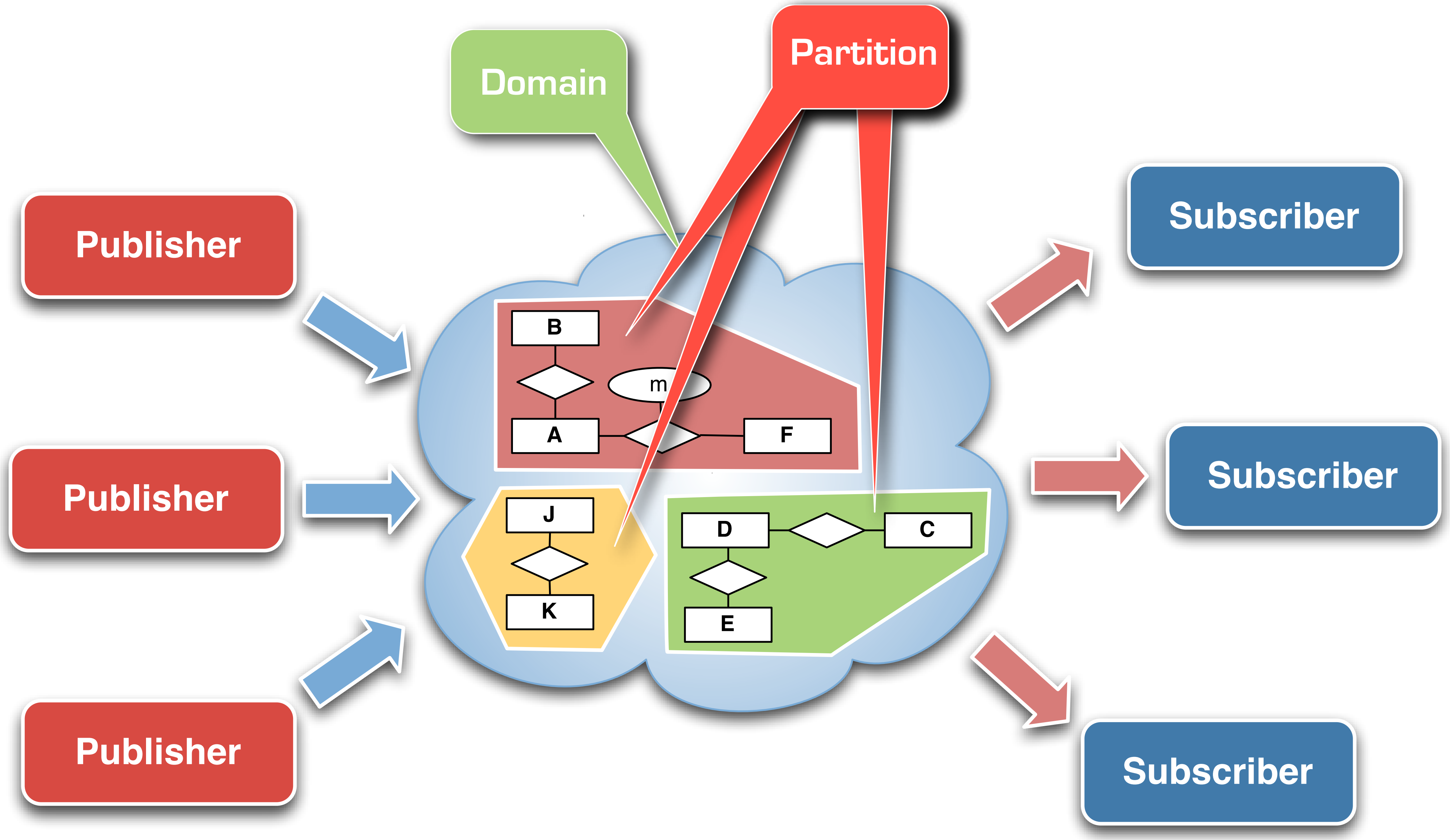
symbol	name	exchange	quote
AAPL	Apple Inc.	NASD	165.37
GOOG	Google Inc.	NASD	663.97
MSFT	Microsoft Corp.	NASD	33.73

Keyed-Topic have one Instance per key-value

Organizing Information

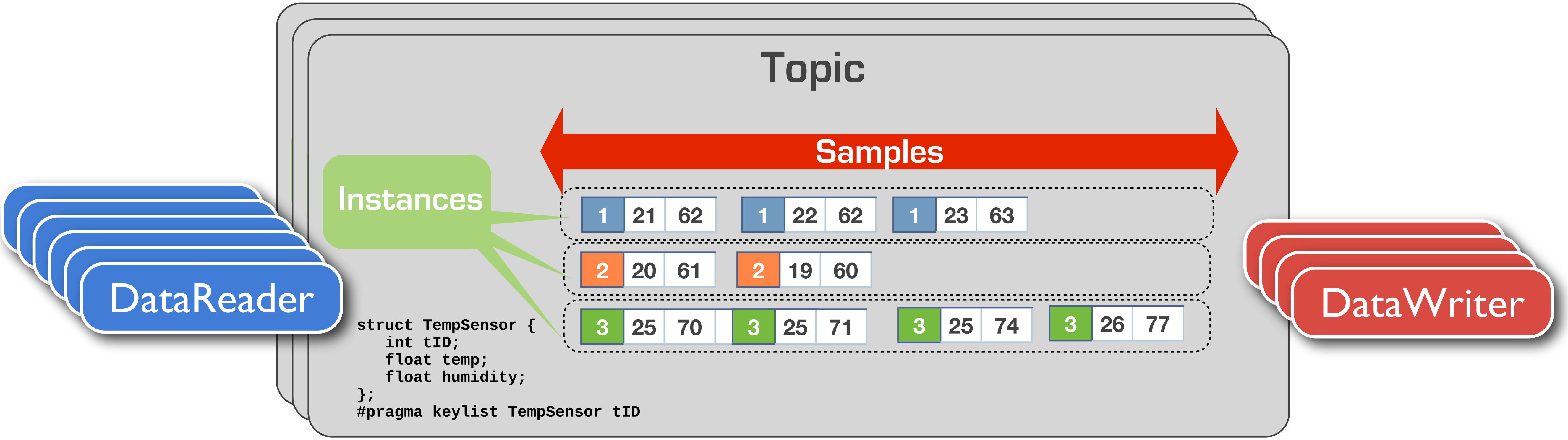
- All DDS communication is organized within a Domain
- Domain can be divided into DDS Partitions
- Topics are published and subscribed across on or more partitions

- **Note:** OpenSplice DDS futher adds the concept of **Network Partitions** which allows to map logical DDS Partitions to **“Physical Network Partitions”**



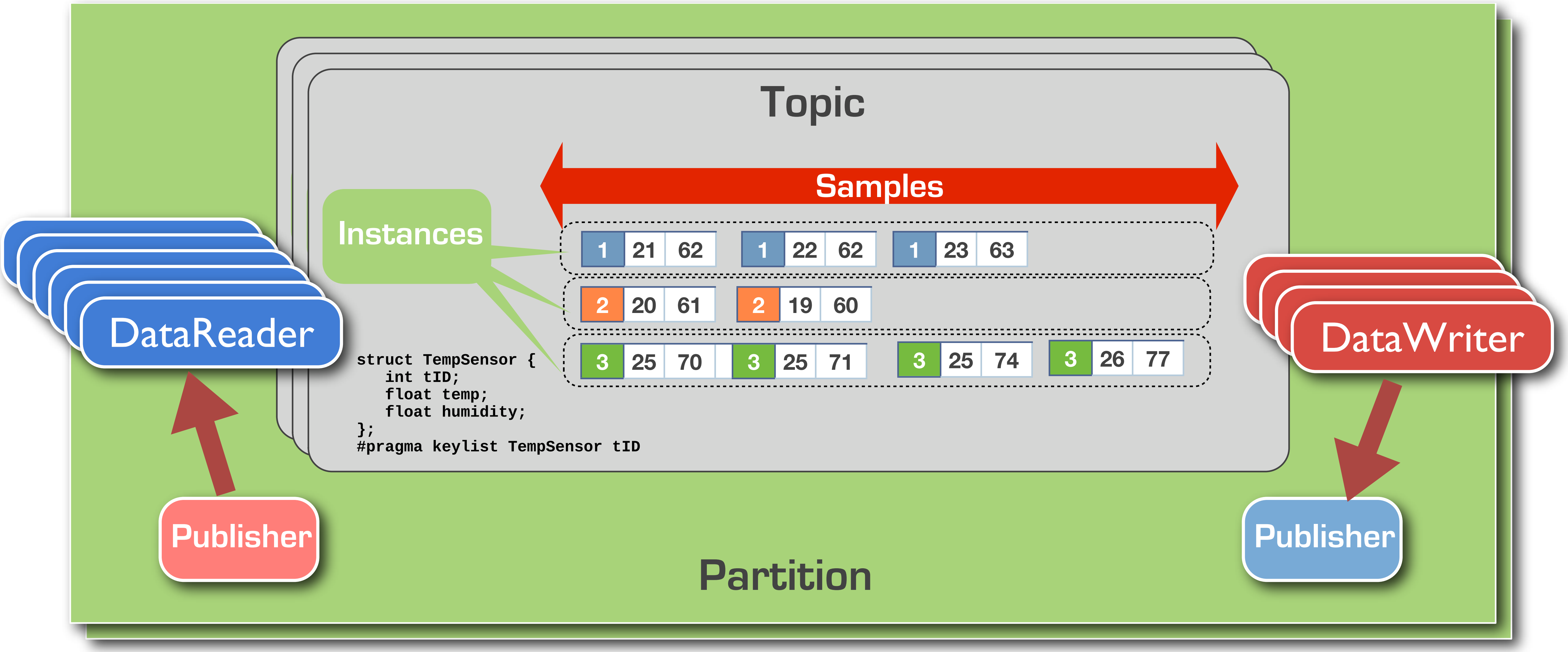
Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Communicating with Topics, Partitions and Domains



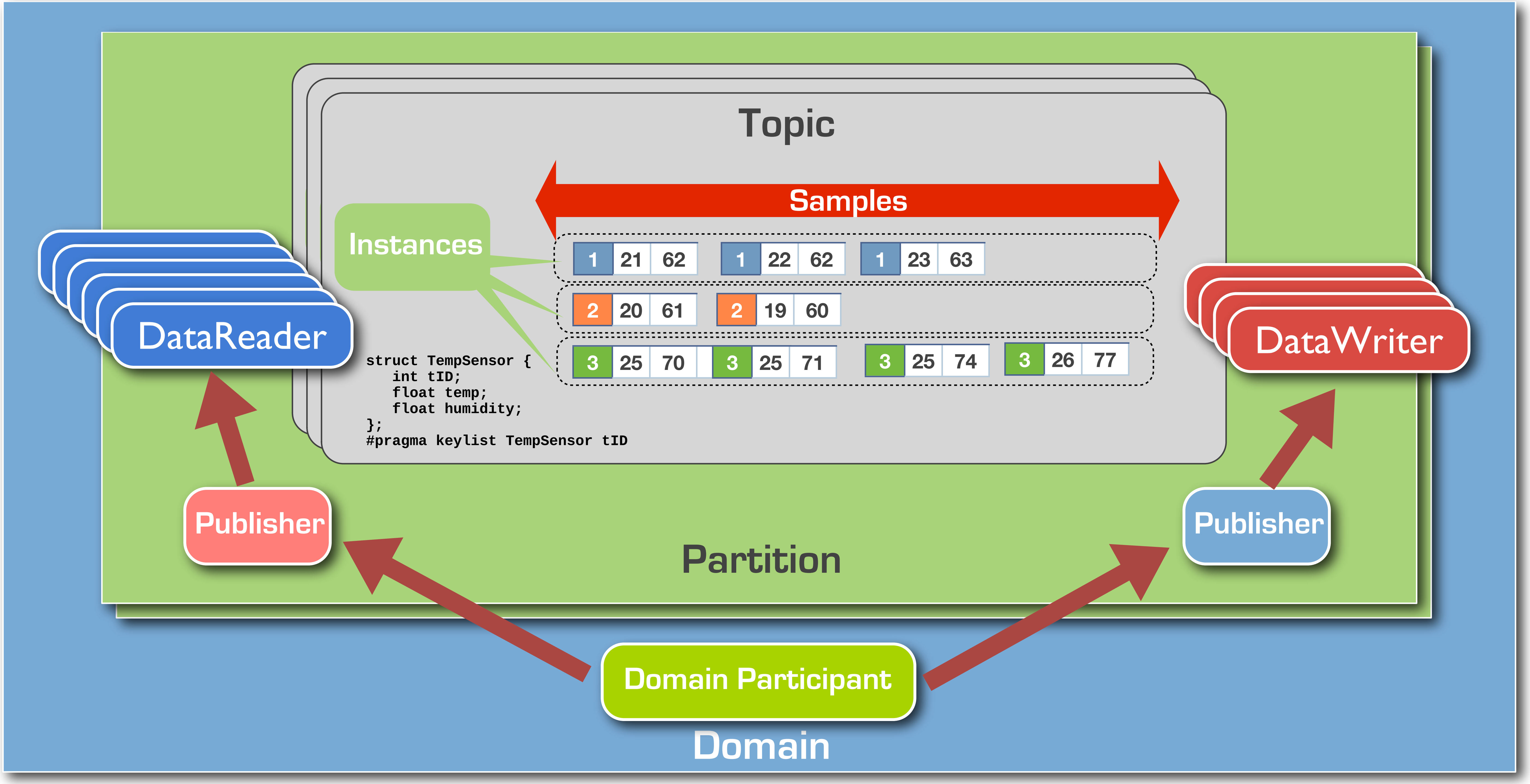
```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor tID
```

Communicating with Topics, Partitions and Domains



```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor tID
```

Communicating with Topics, Partitions and Domains



```
struct TempSensor {
   int tID;
   float temp;
   float humidity;
};
#pragma keylist TempSensor tID
```

Arrows show structural relationships, not data-flows

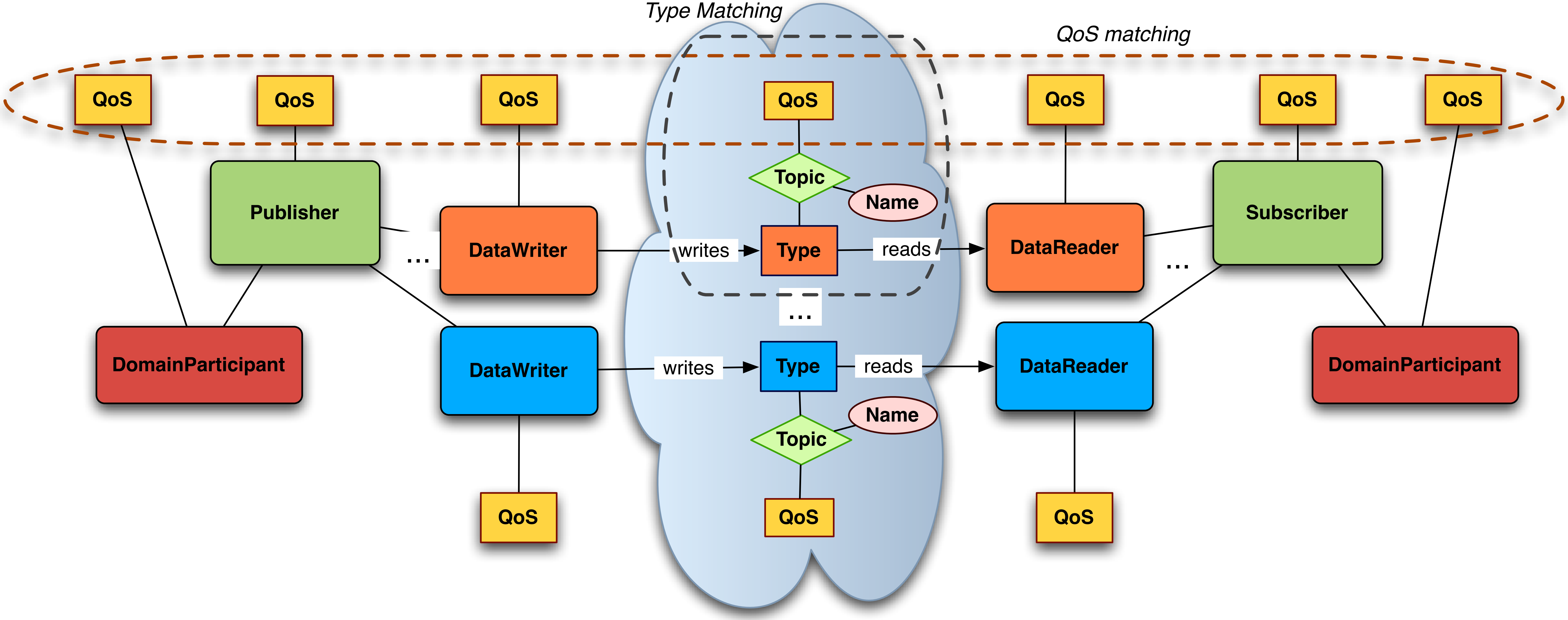
OpenSplice|DDS

Delivering Performance, Openness, and Freedom

A little bit of Theory

QoS

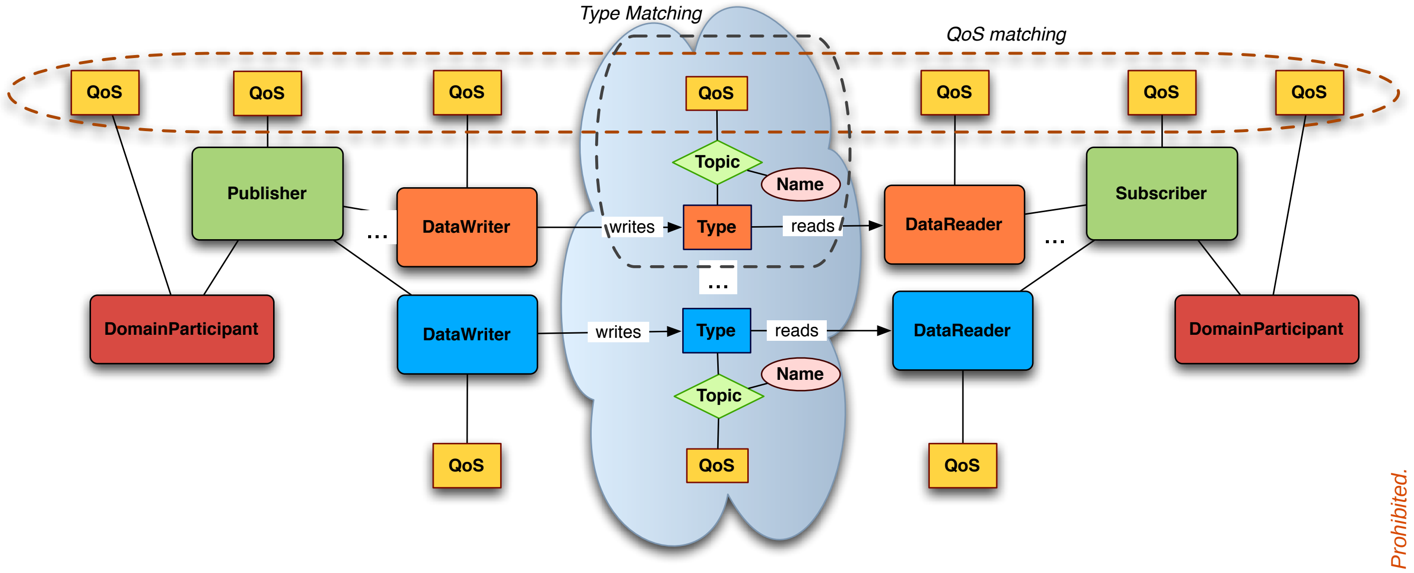
QoS Policies



Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Sample QoS Policies

QoS Policy	Applicability	RxO	Modifiable	
DURABILITY	T, DR, DW	Y	N	Data Availability
DURABILITY SERVICE	T, DW	N	N	
LIFESPAN	T, DW	-	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	-	Y	
DEADLINE	T, DR, DW	Y	Y	Data Timeliness
LATENCY BUDGET	T, DR, DW	Y	Y	
TRANSPORT PRIORITY	T, DW	-	Y	
TIME BASED FILTER	DR	-	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

PRISMTECH

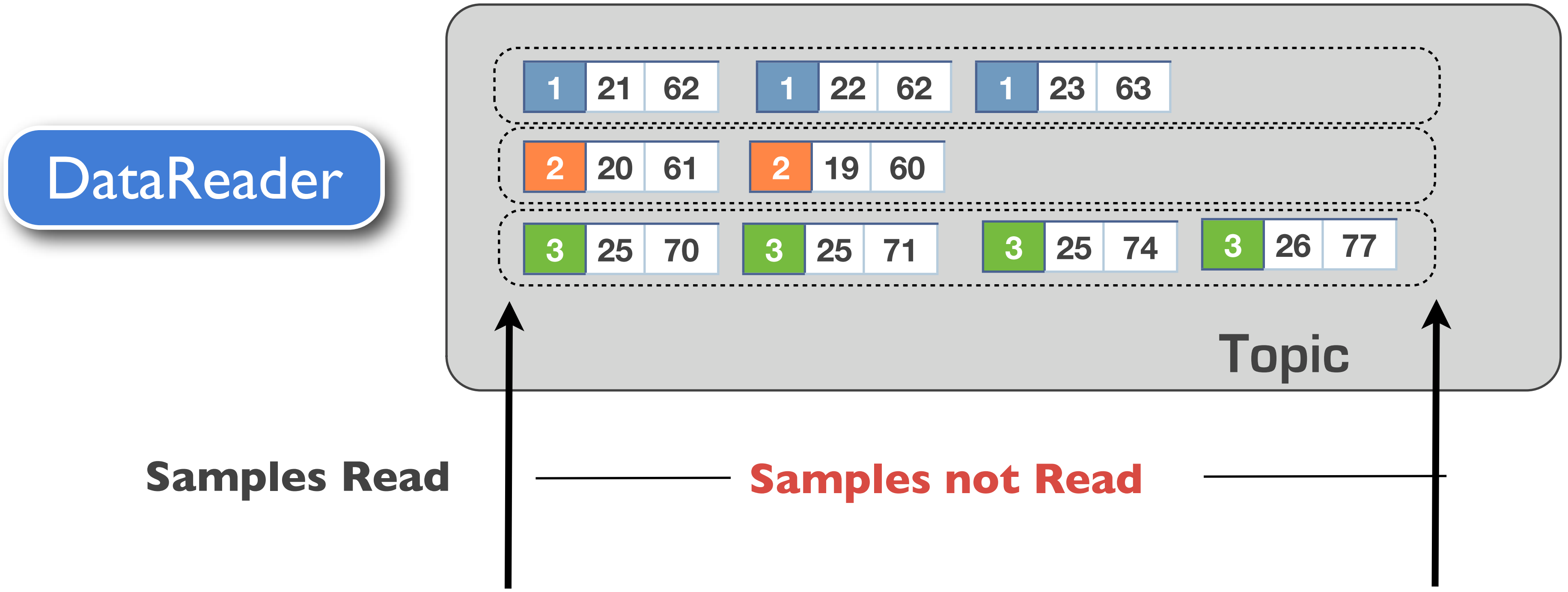
OpenSplice|DDS

Delivering Performance, Openness, and Freedom

A little bit of Theory

Reading Data

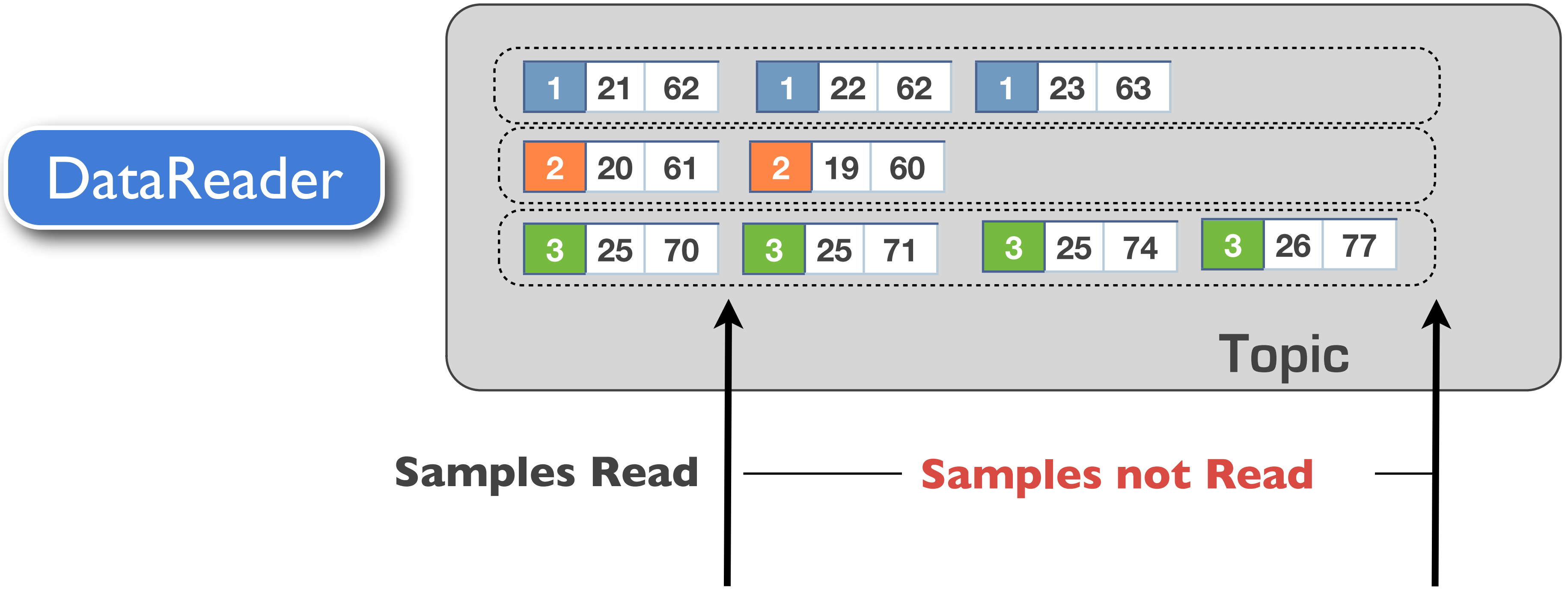
Reading Samples



- Read iterates over the available sample instances
- **Samples** are **not removed from the local cache** as result of a read
- Read samples can be read again, by accessing the cache with the proper options (more later)

Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Reading Samples



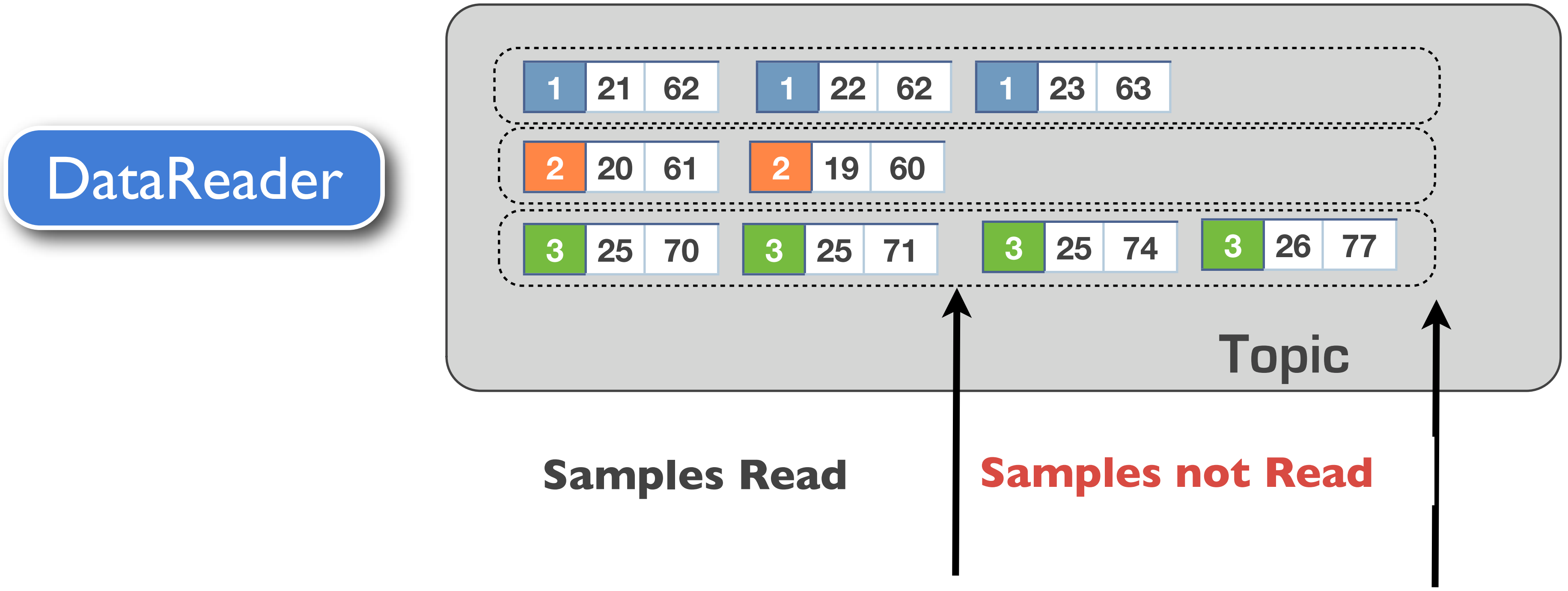
1 21 62 | 1 22 62 | 1 23 63

2 20 61 | 2 19 60

3 25 70 | 3 25 71 | 3 25 74 | 3 26 77

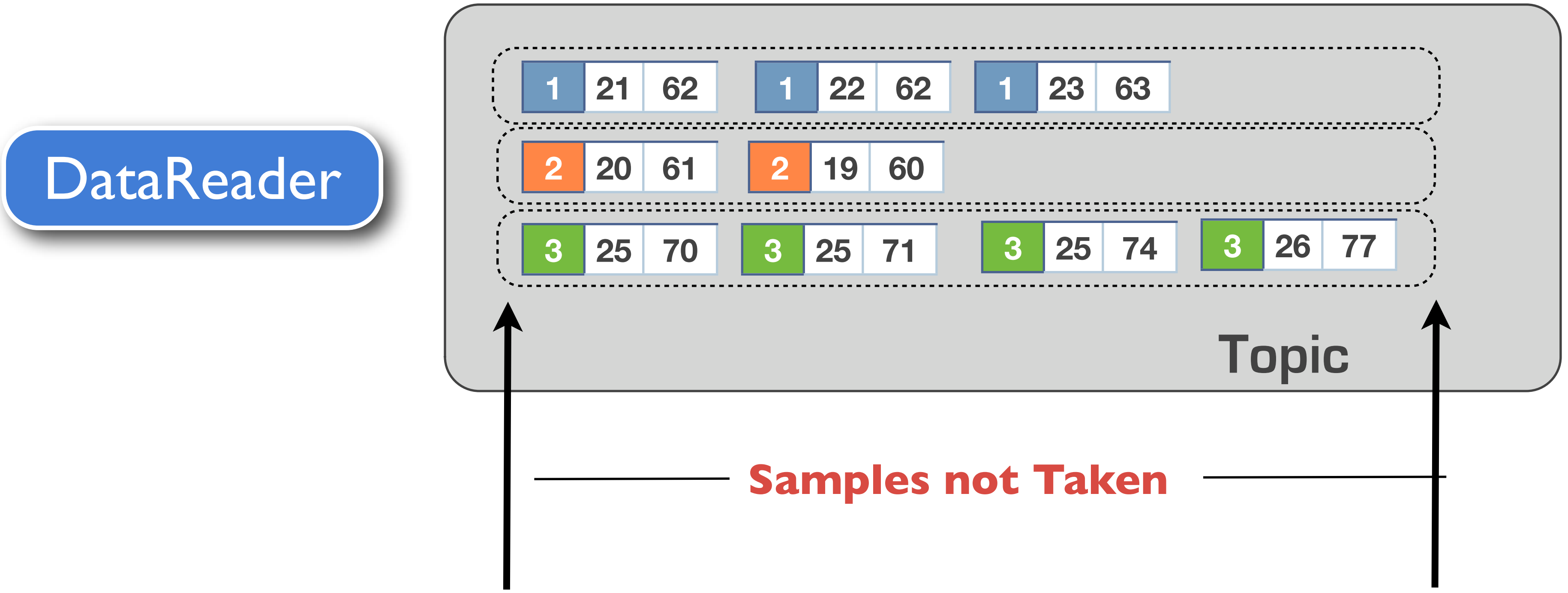
- Read iterates over the available sample instances
- **Samples** are **not removed from the local cache** as result of a read
- Read samples can be read again, by accessing the cache with the proper options (more later)

Reading Samples



- Read iterates over the available sample instances
- **Samples** are **not removed from the local cache** as result of a read
- Read samples can be read again, by accessing the cache with the proper options (more later)

Taking Samples



DataReader

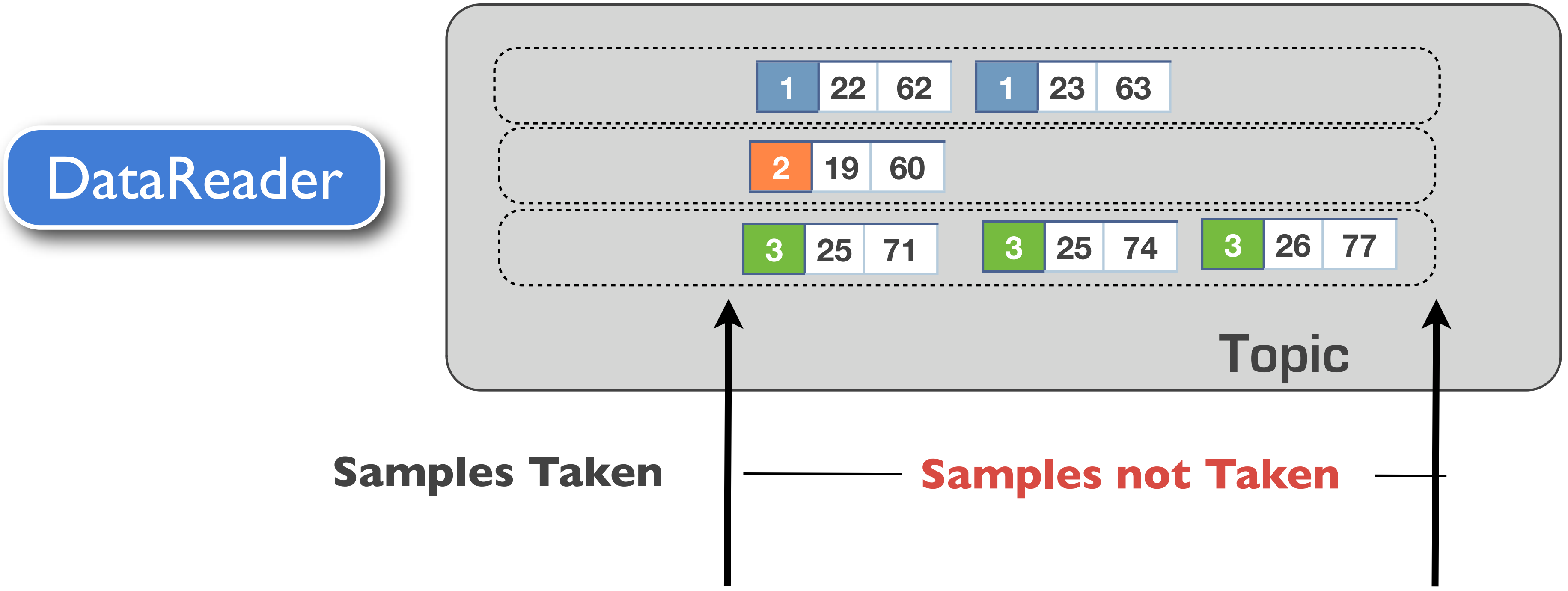
1	21	62	1	22	62	1	23	63			
2	20	61	2	19	60						
3	25	70	3	25	71	3	25	74	3	26	77

Topic

Samples not Taken

- Take iterates over the available sample instances
- Taken Samples are **removed from the local cache** as result of a take

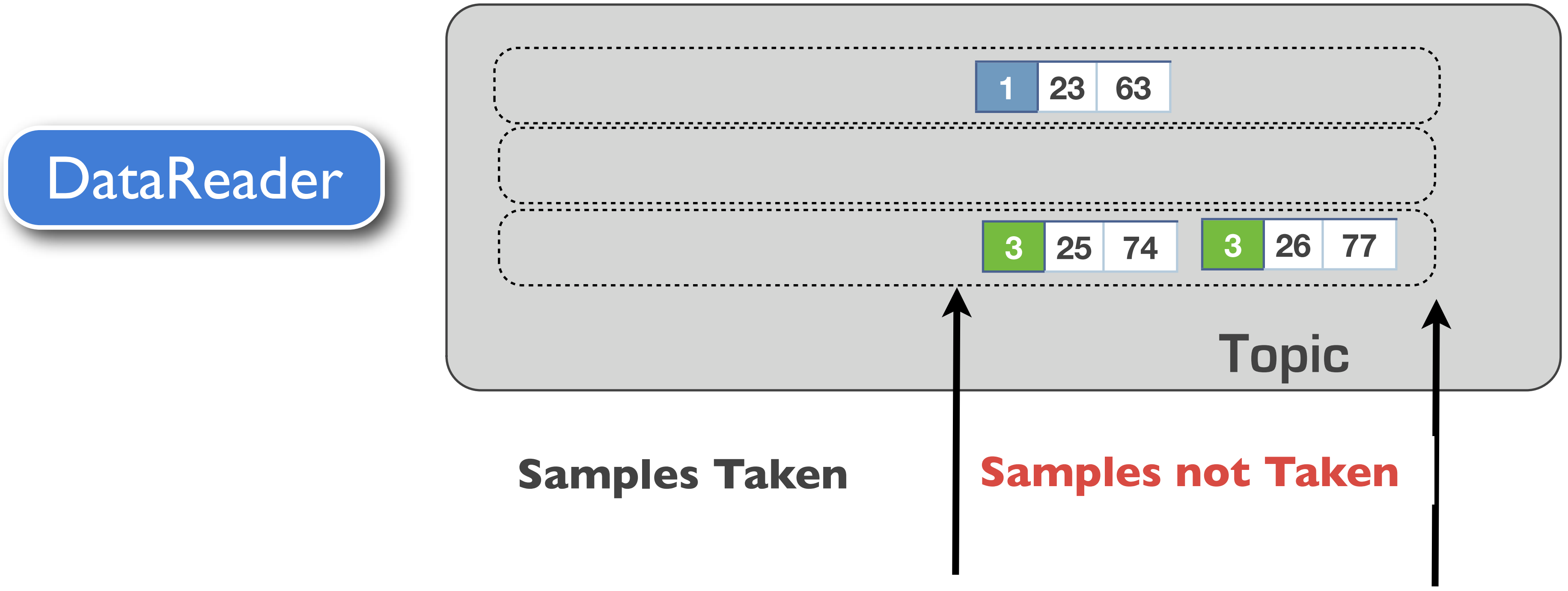
Taking Samples



- Take iterates over the available sample instances
- Taken Samples are **removed from the local cache** as result of a take

Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Taking Samples



- Take iterates over the available sample instances
- Taken Samples are **removed from the local cache** as result of a take

OpenSplice|DDS

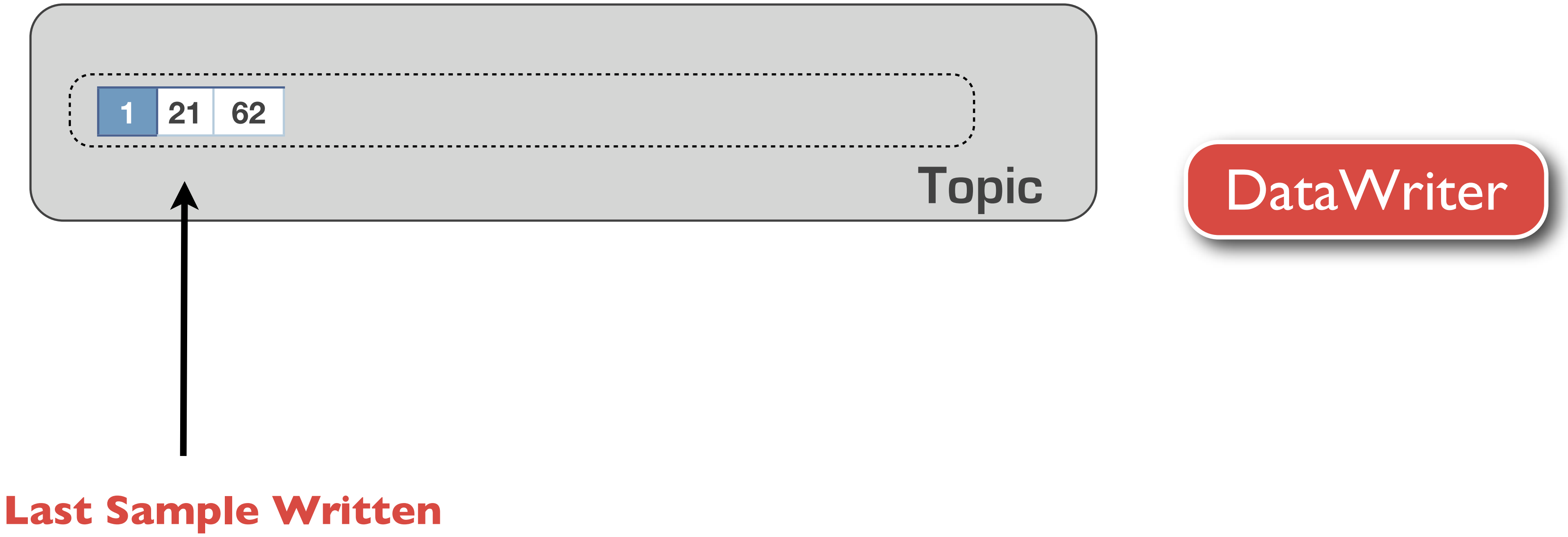
Delivering Performance, Openness, and Freedom

A little bit of Theory

Writing Data

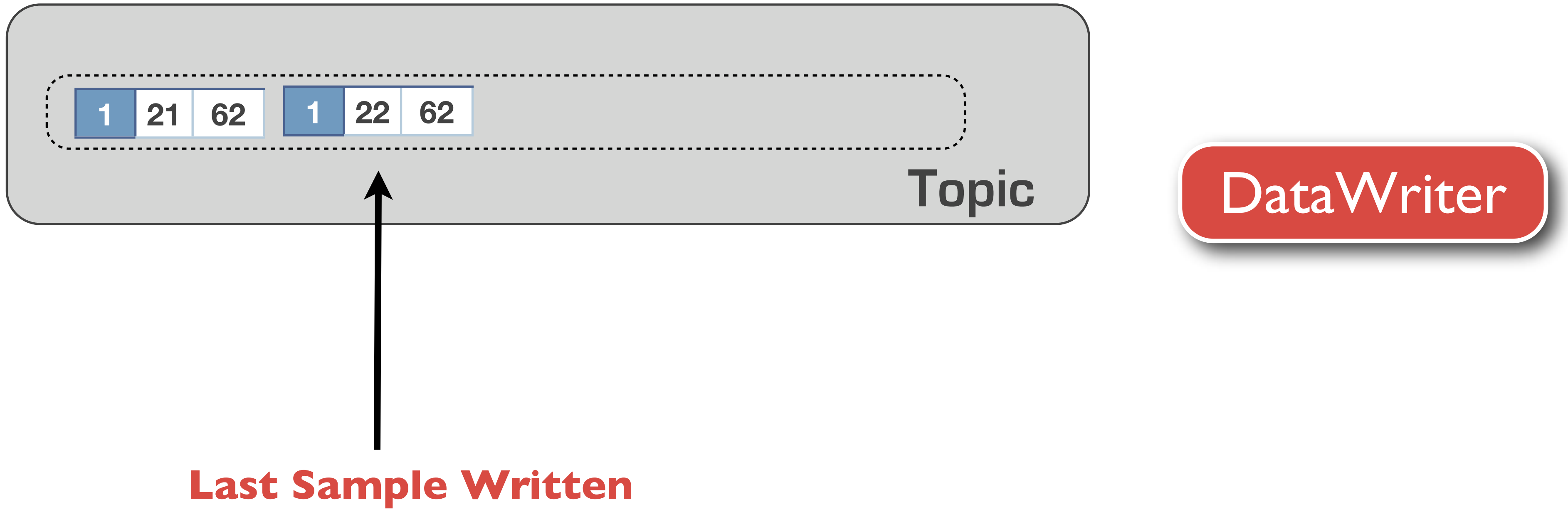
Writing Samples

- Samples are written in the local cache
- Writer control the creation of instances
- The DDS ensures that the local caches for the matched DataReader will be **eventually consistent** with that of the Data Writer



Writing Samples

- Samples are written in the local cache
- Writer control the creation of instances
- The DDS ensures that the local caches for the matched DataReader will be **eventually consistent** with that of the Data Writer



OpenSplice|DDS

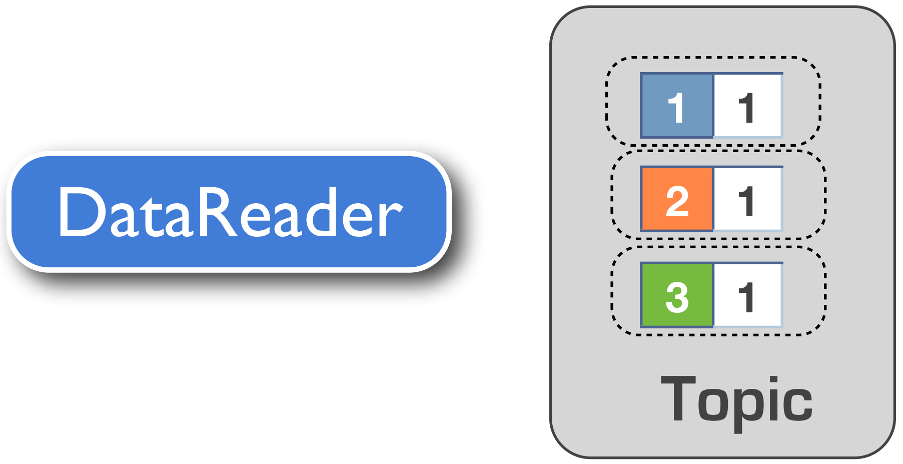
Delivering Performance, Openness, and Freedom

A little bit of Theory

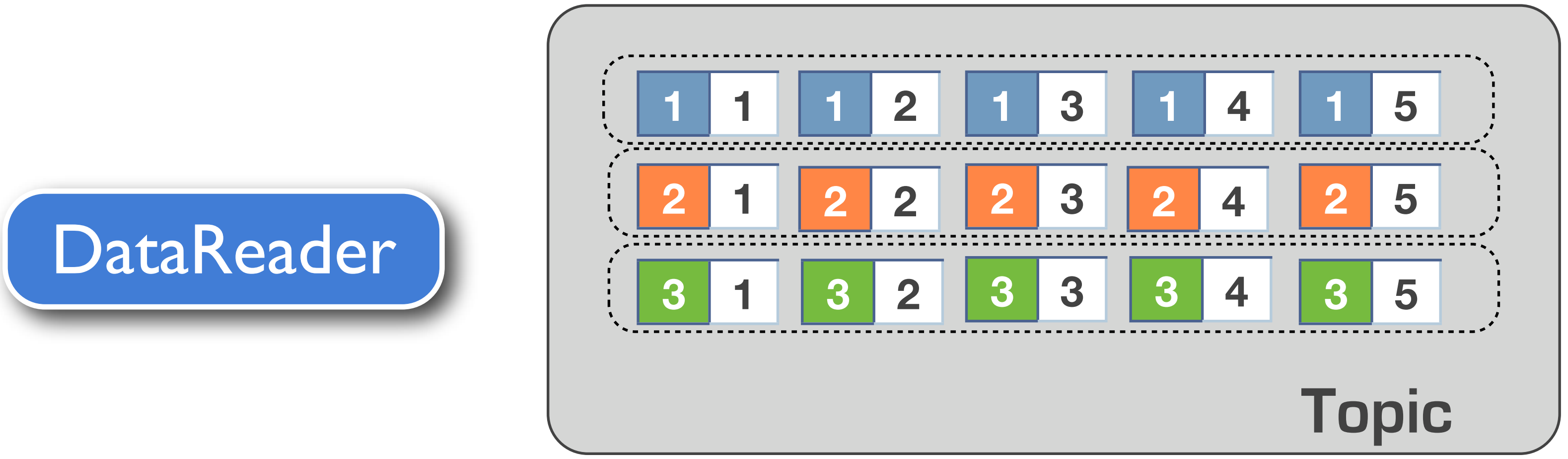
Managing Data History

How many samples?

History Depth = 1 (DDS Default)

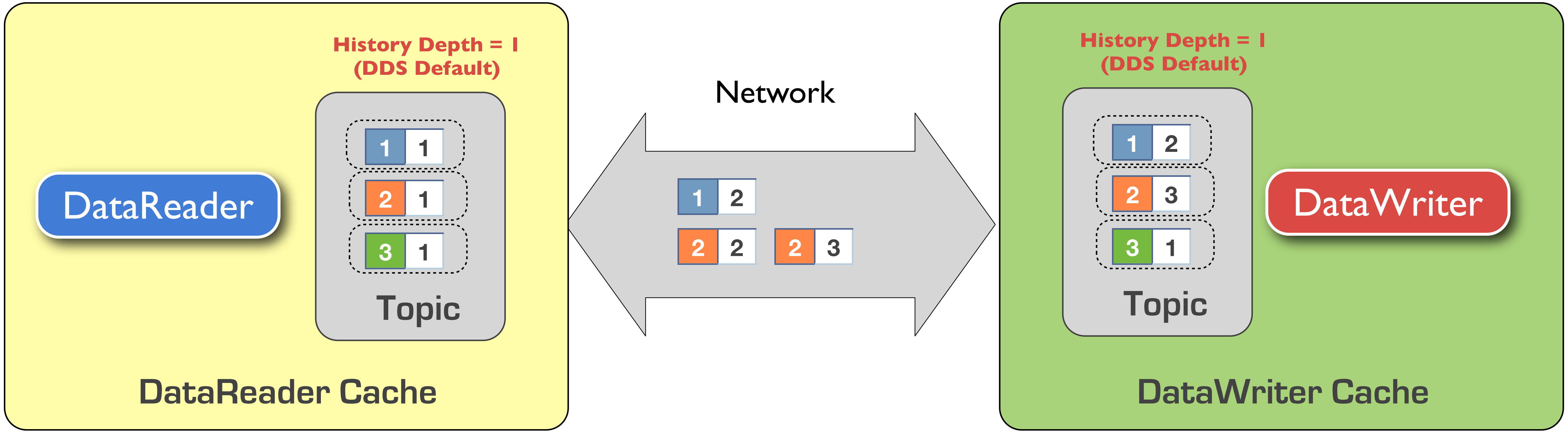


History Depth = 5



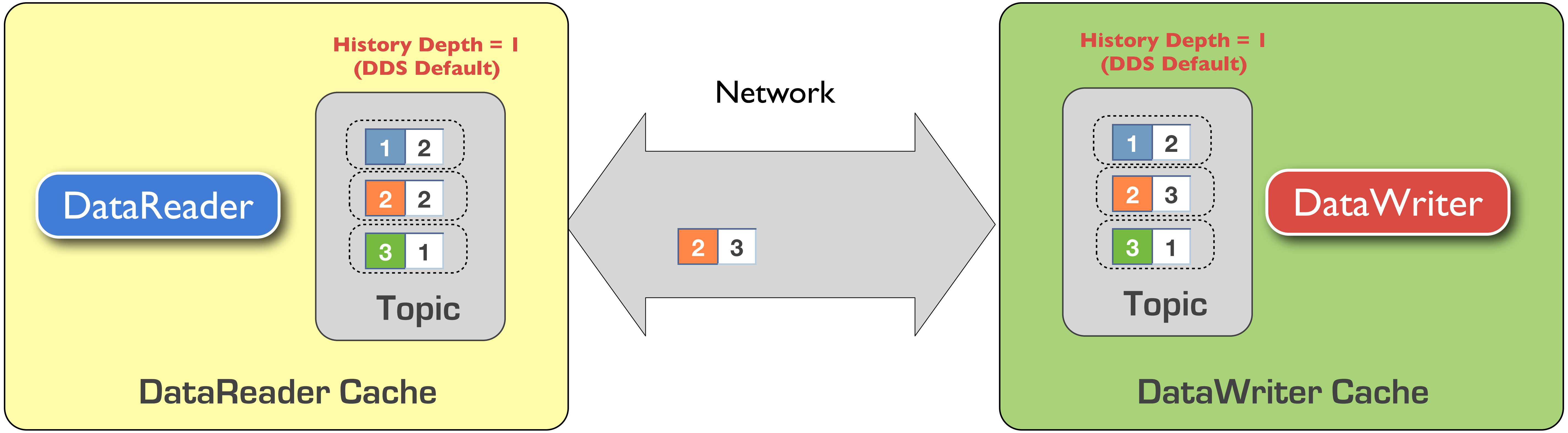
- The History QoS Controls the number of samples-per-instance that will be stored by the middleware on behalf of a Reader
- **Keep Last K**. The History QoS can be set so to always have the latest **K** samples
- **Keep All.** The History QoS can be set so keep all samples produced by the writer and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

History in Action



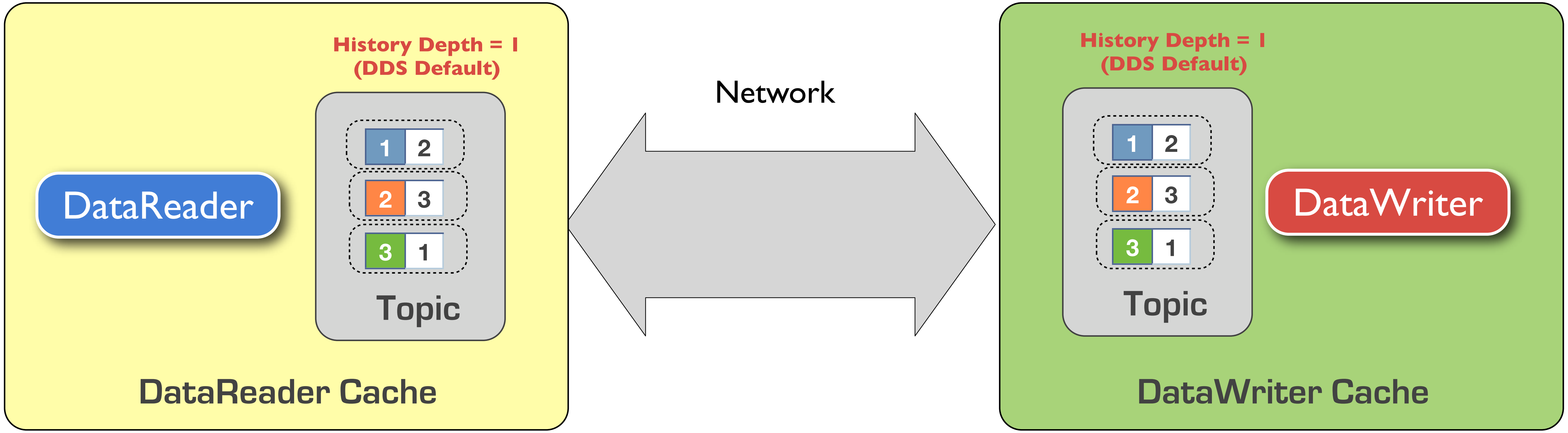
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



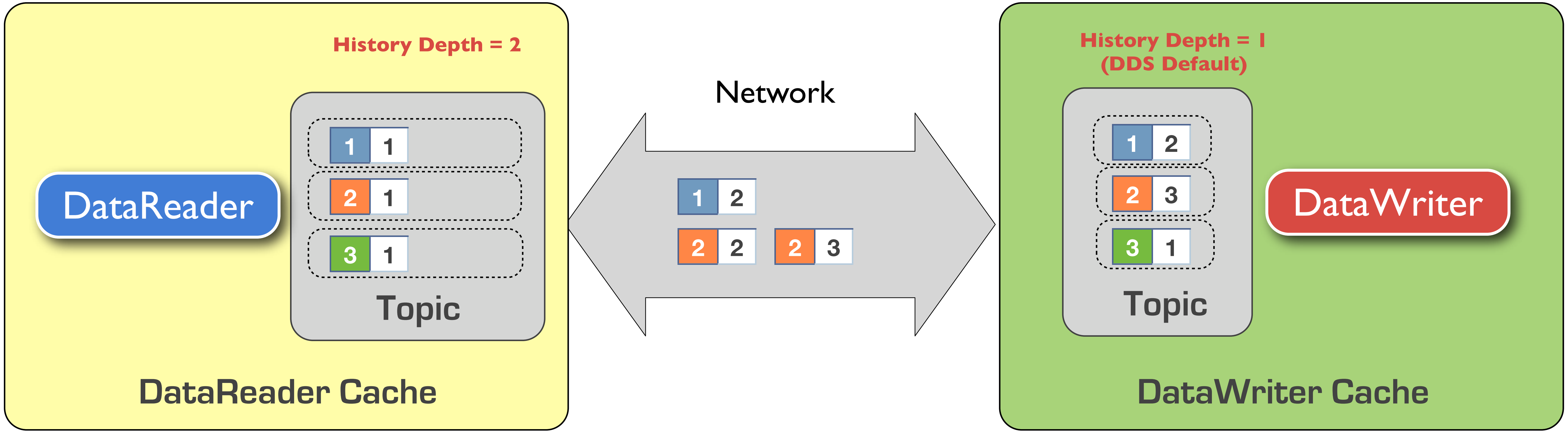
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



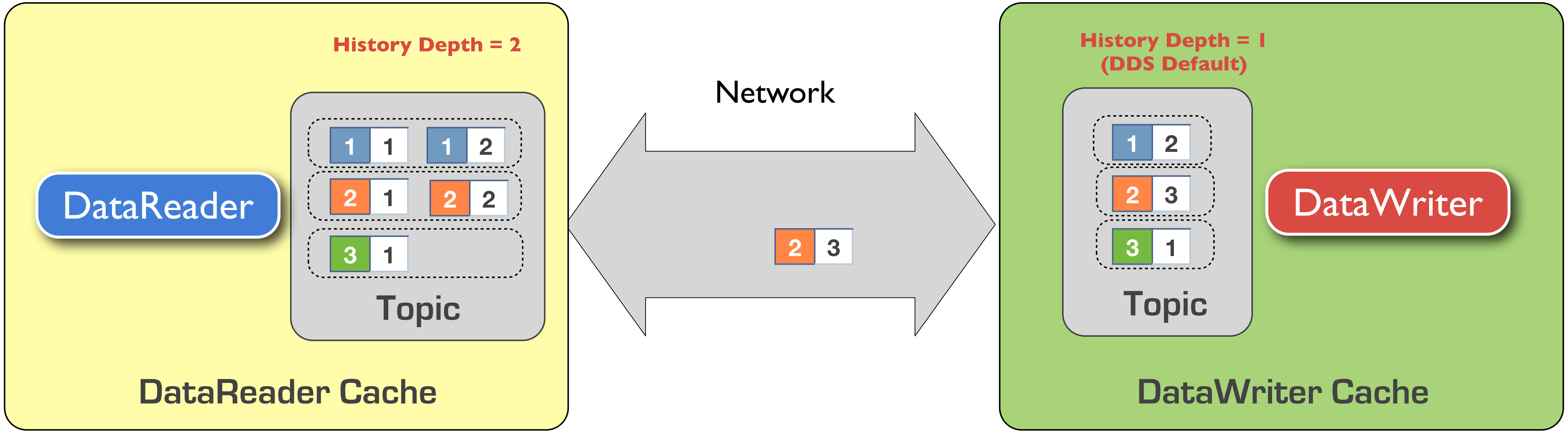
Note: The Reliability QoS controls wether data is sent reliably, or best-effort, from the DataWriter to matched DataReaders

History in Action



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

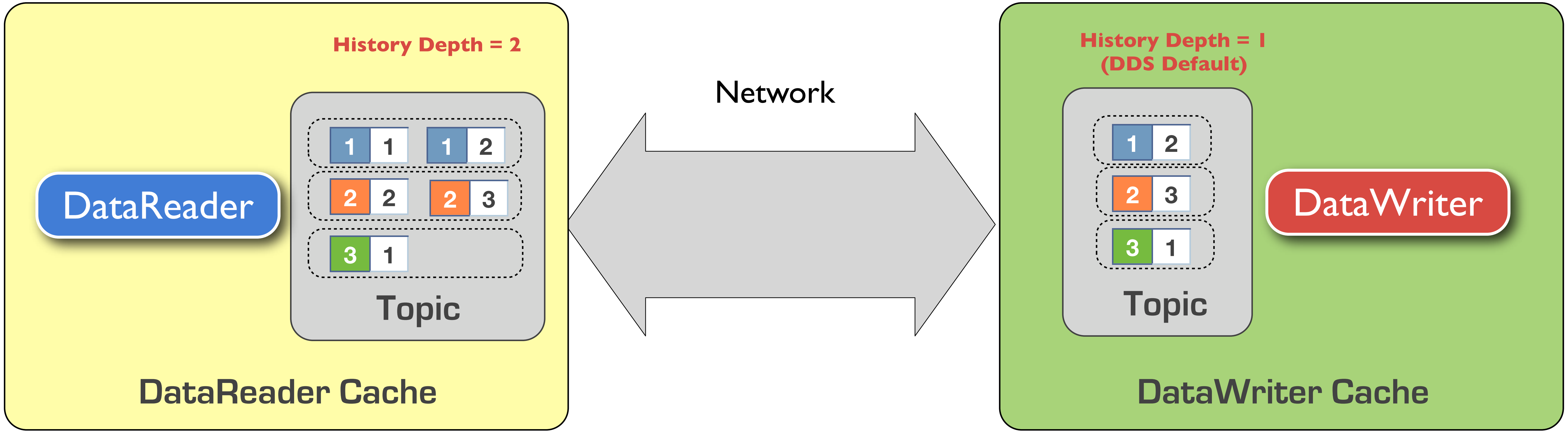
History in Action



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

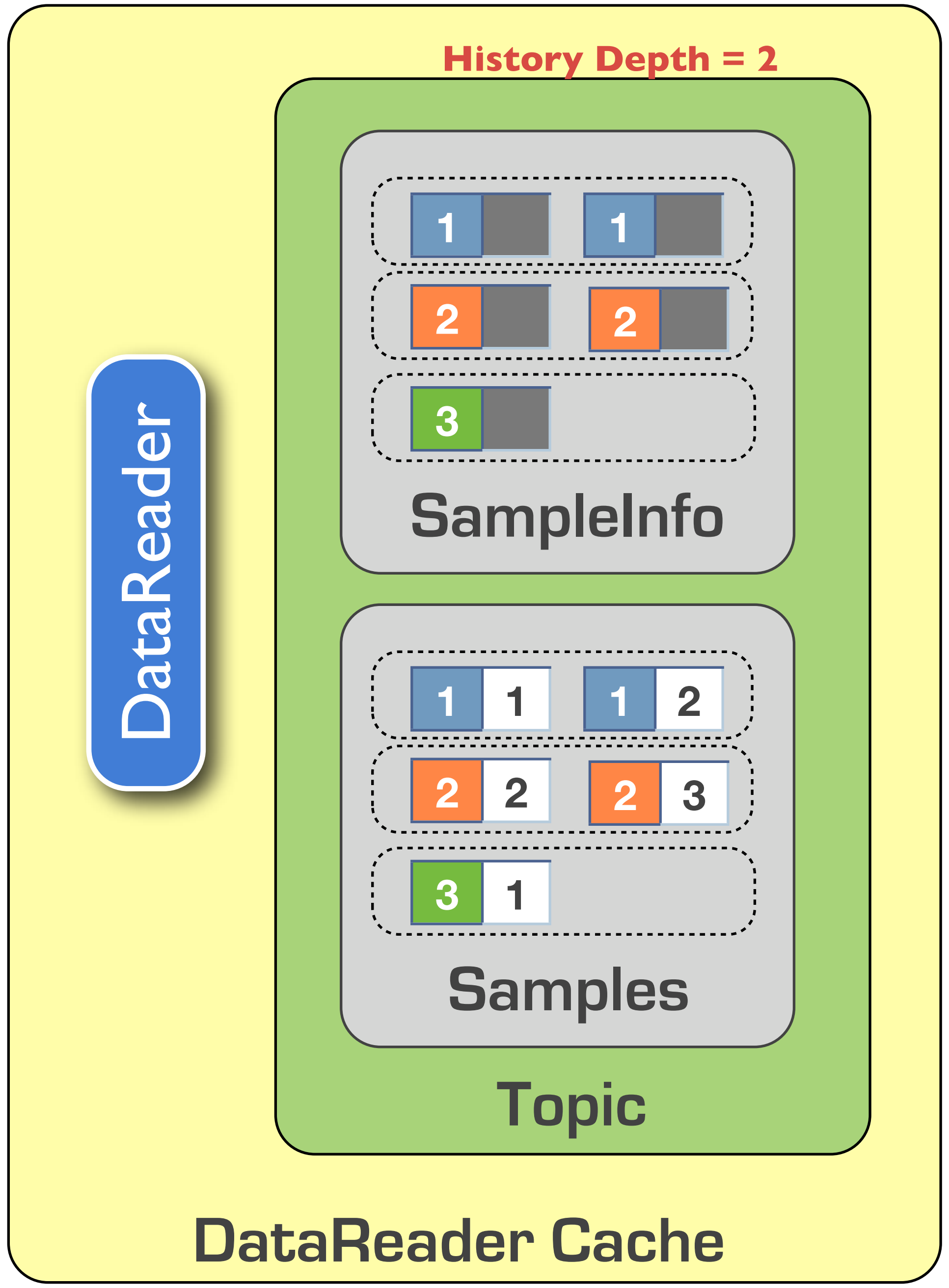
Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

History in Action



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

Sample, Instance and View States



- Along with data samples, DataReaders are provided with state information that allows to detect relevant transitions in the life-cycle of data as well as data writers
- **Sample State (READ | NOT_READ):** Determines wether a sample has already been read by this DataWriter or not.
- **Instance State (ALIVE, NOT_ALIVE, DISPOSED).** Determines wether (1) writer exist for the specific instance, or (2) no matched writers are currently available, or (3) the instance has been disposed
- View State (**NEW, NOT_NEW**). Determines wether this is the first sample of a new (or re-born) instance

Application / DDS Coordination

DDS provides three main mechanism for exchanging information with the application

- **Polling.** The application polls from time to time for new data or status changes. The interval might depend on the kind of applications as well as data
- **WaitSets.** The application registers a WaitSet with DDS and waits (i.e. is suspended) until one of the specified events has not happened.
- **Listeners.** The application registers a listener with a specific DDS entity to be notified when relevant events occur, such as state changes or

OpenSplice|DDS

Delivering Performance, Openness, and Freedom

In Practice...

Steps for Writing a DDS Application

Writing a DDS application can be decomposed in the following few simple steps:

- **Step#1:** Define Topics
- **Step #2:** Identify QoS representing key non-functional invariants for your system
  - Transport Priority
  - Deadline
  - Durability
- **Step #3:** Define Topics / Partition Mapping
- **Step #4:** Identify Topic Readers/Writers
- **Step #5:** Define QoS requirements for Readers/Writers
  - History
  - Latency Budget
  - Auto-Dispose
  - Transport Priority
  - Deadline
- **Step #6:** Code-it in your favorite programming language

Making the Hello DDS World

Step #1: Topic definition

- We are going to define a simple key-less topic that will carry the name to greet.

```
module swatch {

  struct hello {
    string<256> name;
  };
#pragma keylist hello

};
```

Making the Hello DDS World

Step #2: Topic QoS

- **Reliability QoS:** RELIABLE
- **Durability QoS:** TRANSIENT

```
module swatch {

  struct hello {
    string<256> name;
  };
#pragma keylist hello

};
```

Making the Hello DDS World

Step #3: Topics/Partitions Mapping

- swatch::hello will be mapped into the default-partition (thus no action to take)

Making the Hello DDS World

Step #4: Readers/Writers

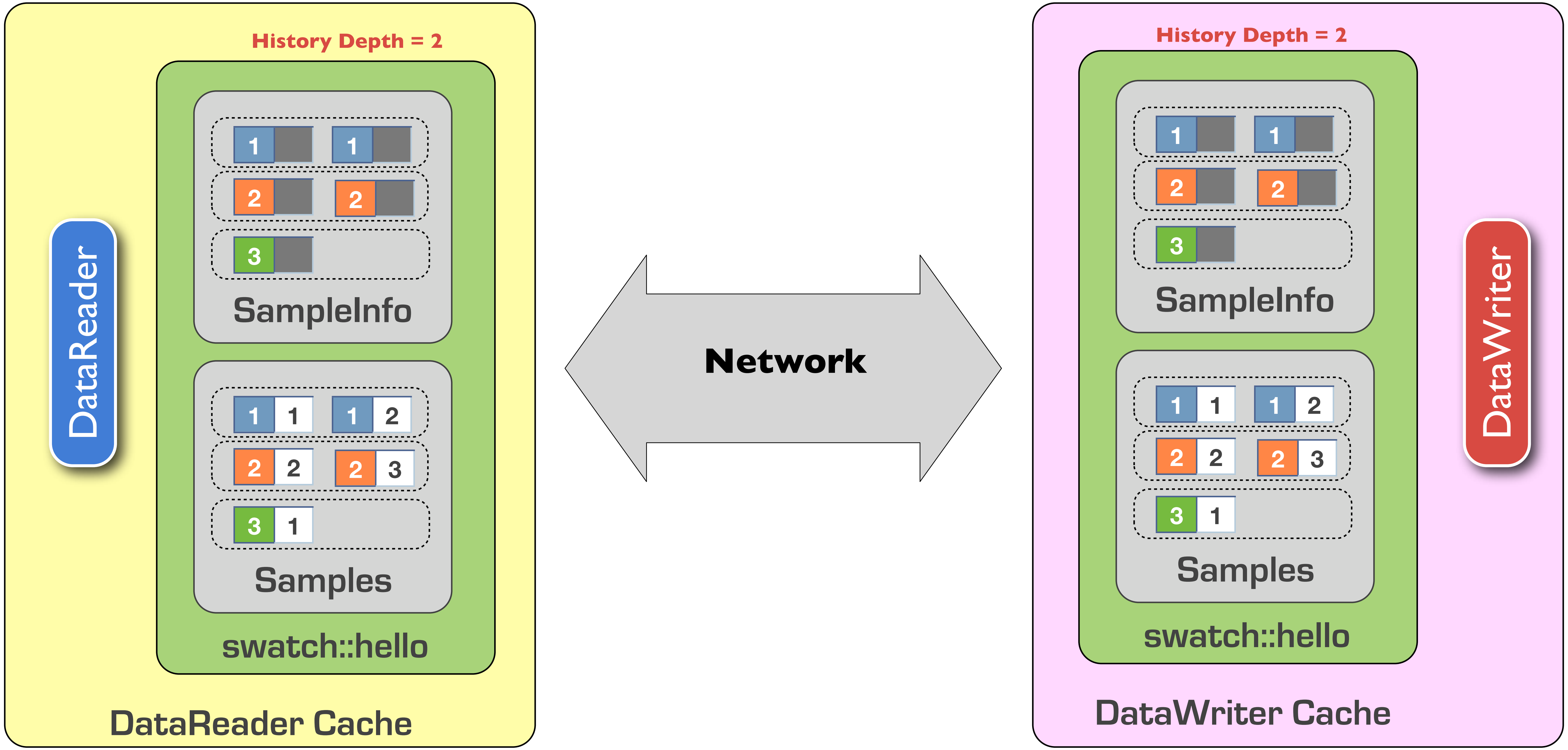
- A Generic DataReader that will read the topic swatch:: hello
  - We'll be able to run as many of this as we want
- A Generic DataWriter that will read the topic swatch:: hello
  - We'll be able to run as many of this as we want

Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Making the Hello DDS World

- **Step #5:** Define QoS requirements for Readers/Writers
- Writer:
  - Inherit TopicQoS, and
  - No-Auto Dispose
  - History QoS: Keep Last N
- Reader
  - Inherit TopicQoS, and
  - History QoS: Keep Last N

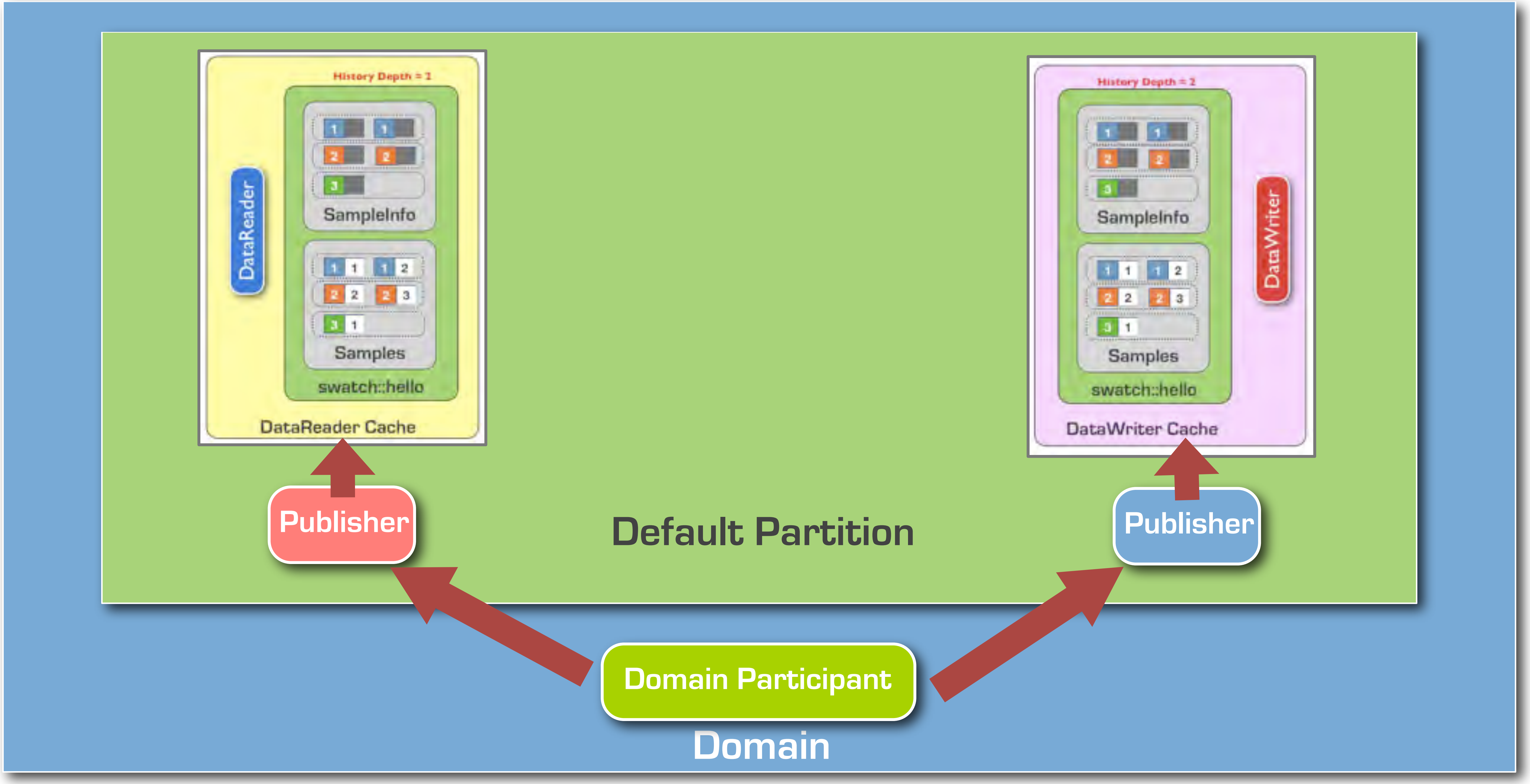
Hello World in a Conceptual Picture



OpenSplice|DDS

PrismTech

Visualizing the Strucutre...



Step #6: Coding

How many lines of code is going to take this example?

SIMple Dds == SIMD!

- Today we’ll write our DDS application using SIMD
- SIMD is a C++ library that takes advantage of C++ Template Meta-Programming to:
  - Vastly Improve Productivity
  - Simplify Usage
  - Automate Resource Management (All DDS Entities are Garbage Collected via Ref-Counting)
  - Zero Overhead

SIMD

Hello-pub.cpp (Default QoS)

```
int main(int argc, char* argv[]) {
  if (!parse_args(argc, argv))
    return 1;

  // -- init the SIMD runtime
  simd::Runtime::init();

  // -- create the DDS Topic
  simd::Topic<swatch::hello> helloTopic("helloTopic");

  // -- create the DDS DataWriter
  simd::DataWriter<swatch::hello> writer(helloTopic,
                                        dwqos);
```

Only 3-lines of DDS-Specific Code

```
  swatch::hello sample;
  std::stringstream ss;

  for (int i = 0; i < N; ++i) {
    ss << i;
    std::string tmp = ss.str() + "." + message;
    ss.str("");
    sample.name = DDS::string_dup(tmp.c_str());
    std::cout << "<<= " <<  sample.name << std::endl;
    writer.write(sample);
    usleep(period*1000);
  }
  std::cout << "[done]" << std::endl;
  return 0; }
```

Business Logic

Hello-pub.cpp

```
int main(int argc, char* argv[]) {
  if (!parse_args(argc, argv))
    return 1;
  // -- init the SIMD runtime
  simd::Runtime::init();

  simd::TopicQos tqos;
  tqos.set_reliable();
  tqos.set_transient();

  // -- create the DDS Topic
  simd::Topic<swatch::hello> helloTopic("helloTopic",
                                        tqos);
  simd::DataWriterQos dwqos(tqos);
  dwqos.set_keep_last(history_depth);
  dwqos.set_auto_dispose(false);

// -- create the DDS DataWriter
  simd::DataWriter<swatch::hello> writer(helloTopic,
                                         dwqos);
```

```
swatch::hello sample;
std::stringstream ss;

for (int i = 0; i < N; ++i) {
  ss << i;
  std::string tmp = ss.str() + "." + message;
  ss.str("");
  sample.name = DDS::string_dup(tmp.c_str());
  std::cout << "<<= " <<  sample.name << std::endl;
  writer.write(sample);
  usleep(period*1000);
}
std::cout << "[done]" << std::endl;
return 0; }
```

Business Logic

Proprietary Information - Distribution without Expressed Written Permission is Prohibited.

Hello-sub.cpp (Default QoS)

```
int main(int argc, char* argv[]) {
  if (!parse_args(argc, argv))
    return 1;

  // -- init the SIMD runtime
  simd::Runtime::init();

  // -- create the DDS Topic
  simd::Topic<swatch::hello> helloTopic("helloTopic");

  // -- create the DDS DataReader
  simd::DataReader<swatch::hello> reader(helloTopic);
```

Only 3-lines of DDS-Specific Code

```
swatch::helloSeq samples;
  DDS::SampleInfoSeq infos;

  while (true) {
    reader.read(samples, infos);
    for (int i = 0; i < samples.length(); ++i) {
      std::cout << "=>> " <<  samples[i].name
                << std::endl;
    }
    if (samples.length() > 0)
      std::cout << "--" << std::endl;
    reader.return_loan(samples, infos);
    usleep(period*1000);
  }
  return 0;}
```

Business Logic

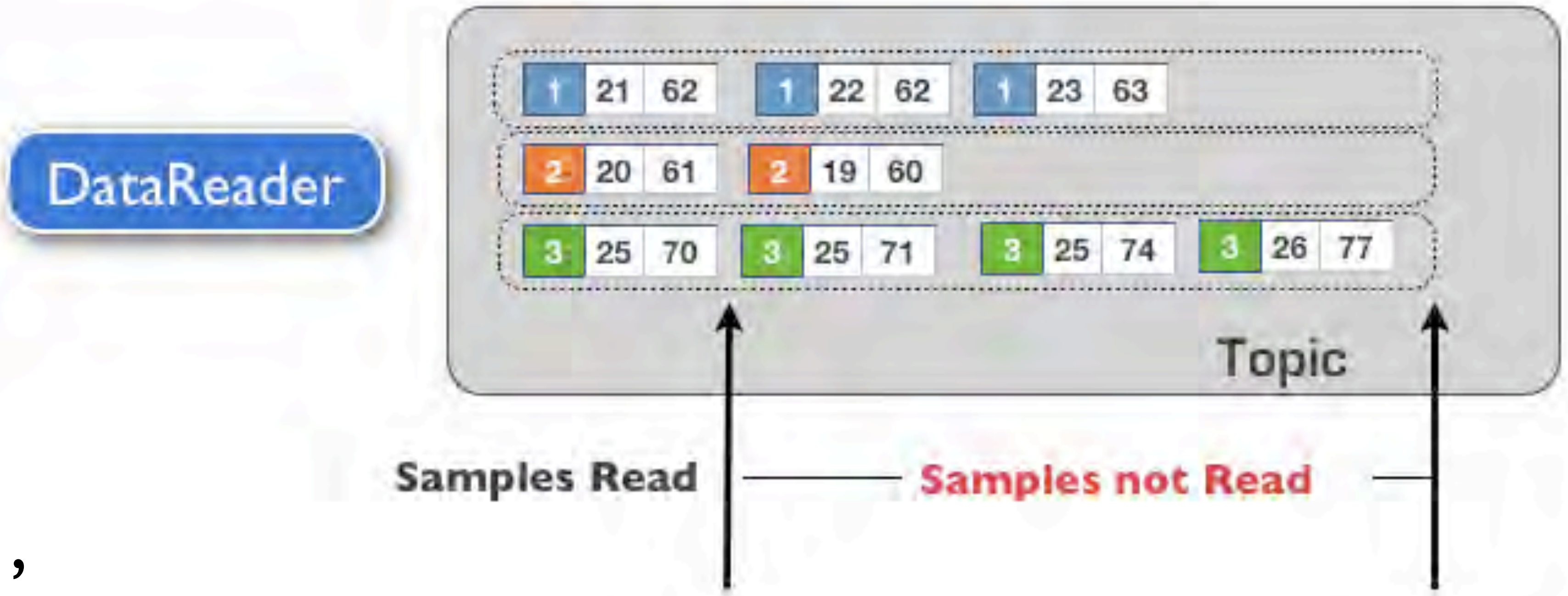
Hello-sub.cpp

```
int main(int argc, char* argv[]) {
  if (!parse_args(argc, argv))
    return 1;
  // -- init the SIMD runtime
  simd::Runtime::init();

  simd::TopicQos tqos;
  tqos.set_reliable();
  tqos.set_transient();

  // -- create the DDS Topic
  simd::Topic<swatch::hello> helloTopic("helloTopic",
                                         tqos);
  simd::DataReaderQos drqos(tqos);
  drqos.set_keep_last(history_depth);

  // -- create the DDS DataReader
  simd::DataReader<swatch::hello> reader(helloTopic,
                                          drqos);
```

```
swatch::helloSeq samples;
  DDS::SampleInfoSeq infos;

  while (true) {
    reader.read(samples, infos);
    for (int i = 0; i < samples.length(); ++i) {
      std::cout << "=>> " <<  samples[i].name
                << std::endl;
    }
    if (samples.length() > 0)
      std::cout << "--" << std::endl;
    reader.return_loan(samples, infos);
    usleep(period*1000);
  }
  return 0;}
```

Business Logic

The Anatomy of a DDS Read



```
DDS::ReturnCode_t read(
                  TSeq& samples,
                  DDS::SampleInfoSeq& infos,
                  long max_samples,
                  DDS::SampleStateMask samples_state,
                  DDS::ViewStateMask   views_state,
                  DDS::InstanceStateMask instances_state)
```

OpenSplice|DDS

Delivering Performance, Openness, and Freedom

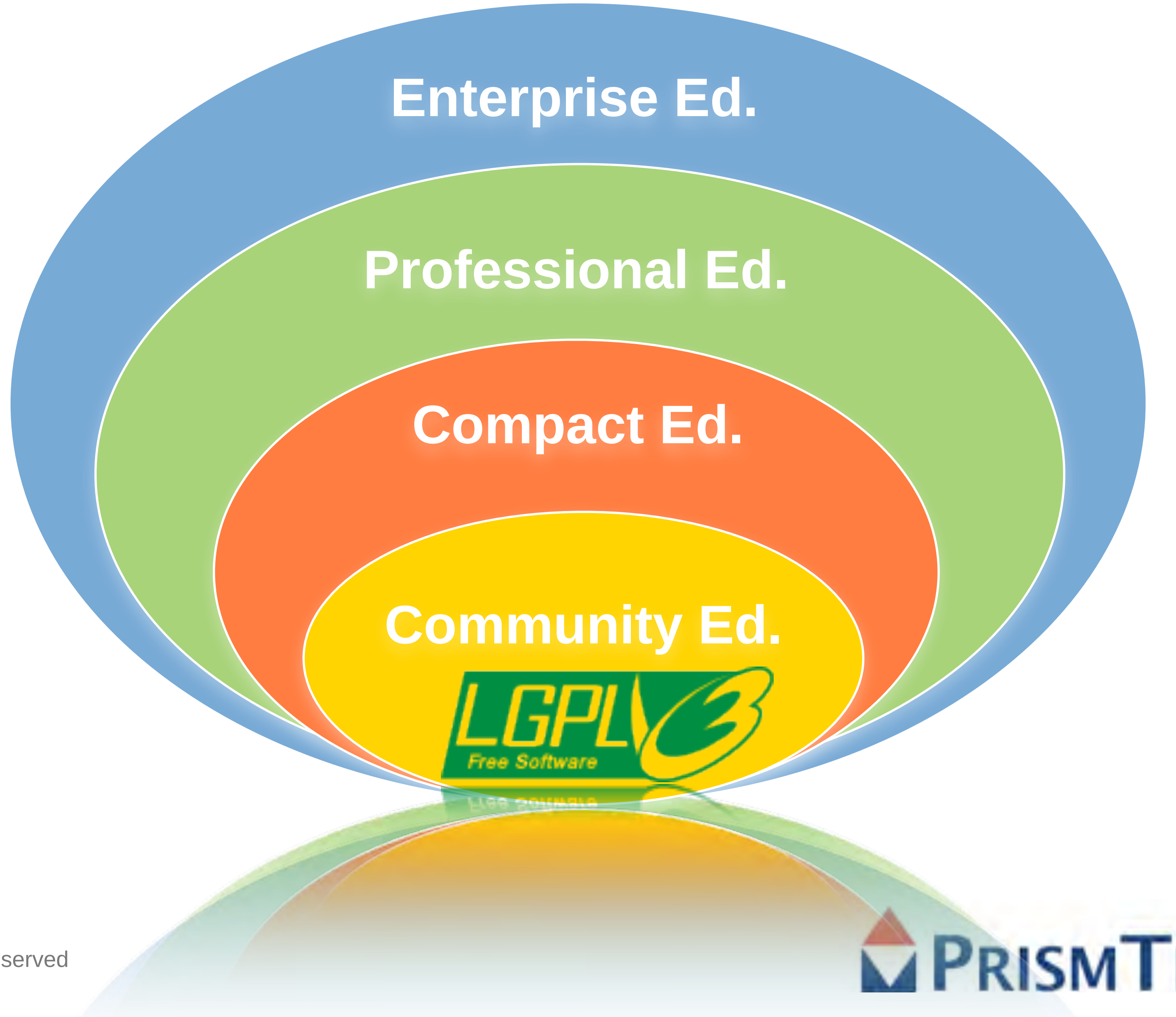
Demo!

Concluding Remarks

- DDS is very Powerful, yet decomposable in Simple aspects.
- DDS is a very powerful Pub/Sub Technology that provides full control over all relevant aspects data and life-cycle
- As demonstrated during the Webcast, writing DDS applications is not hard at all!
- Thus, happy hacking and OpenSplice DDS!

OpenSplice DDS

Delivering Performance, Openness, and Freedom



Online Resources

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

- http://twitter.com/acorsaro/

- http://bit.ly/1Sreg

- http://opensplice.blogspot.com

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

- http://www.dds-forum.org
- http://portals.omg.org/dds