Your systems.
Working as one.

Implementation of the Technical Vision Next Connext DDS Release 5.3

Fernando Crespo Sanchez, Product Architect

Outline

- Key New Features And Products
  - **Scalability:** Topic Queries
  - **Cloud And Accessibility:** IP mobility and Cloud Discovery Service
  - **Security:** Connext DDS Secure
  - **Usability & Debuggability:** Heap monitoring and Logging improvements
  - **Robustness**
- Other Features And Products

Topic Queries

Scalable historical data retrieval in federated large scale system

What are Topic Queries?

Feature that allows:

- Querying the sample cache of a DataWriter
- Getting historical samples through Routing Service
- Scaling historical data retrieval in federated large scale systems

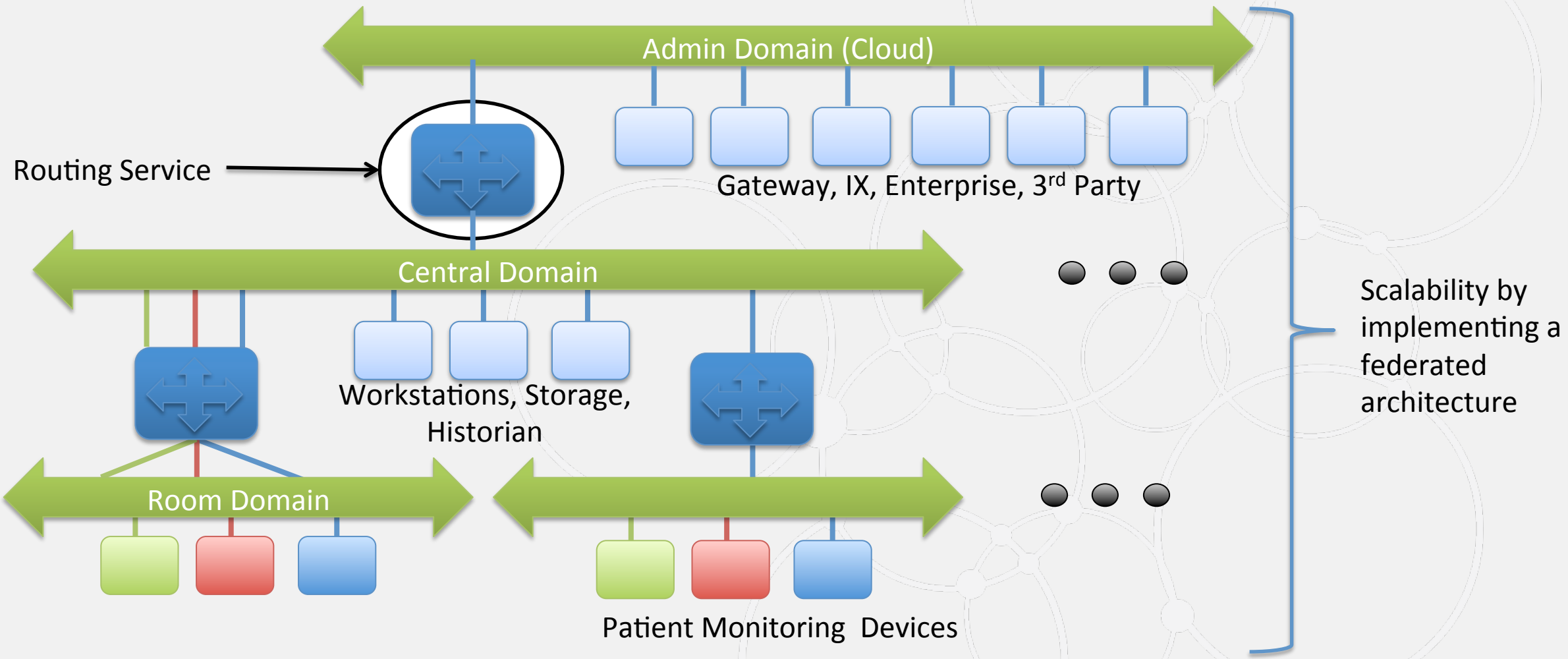
Use Cases

Large scale content-based publish-subscribe systems. Each message may be of interest to a small number of DataReaders. DataReaders interested in both live and historical data

IoT driven healthcare applications such as remote patient monitoring systems

Industrial automation applications. Many individual data points and only a small subset subscribed

Patient Monitoring Use Case

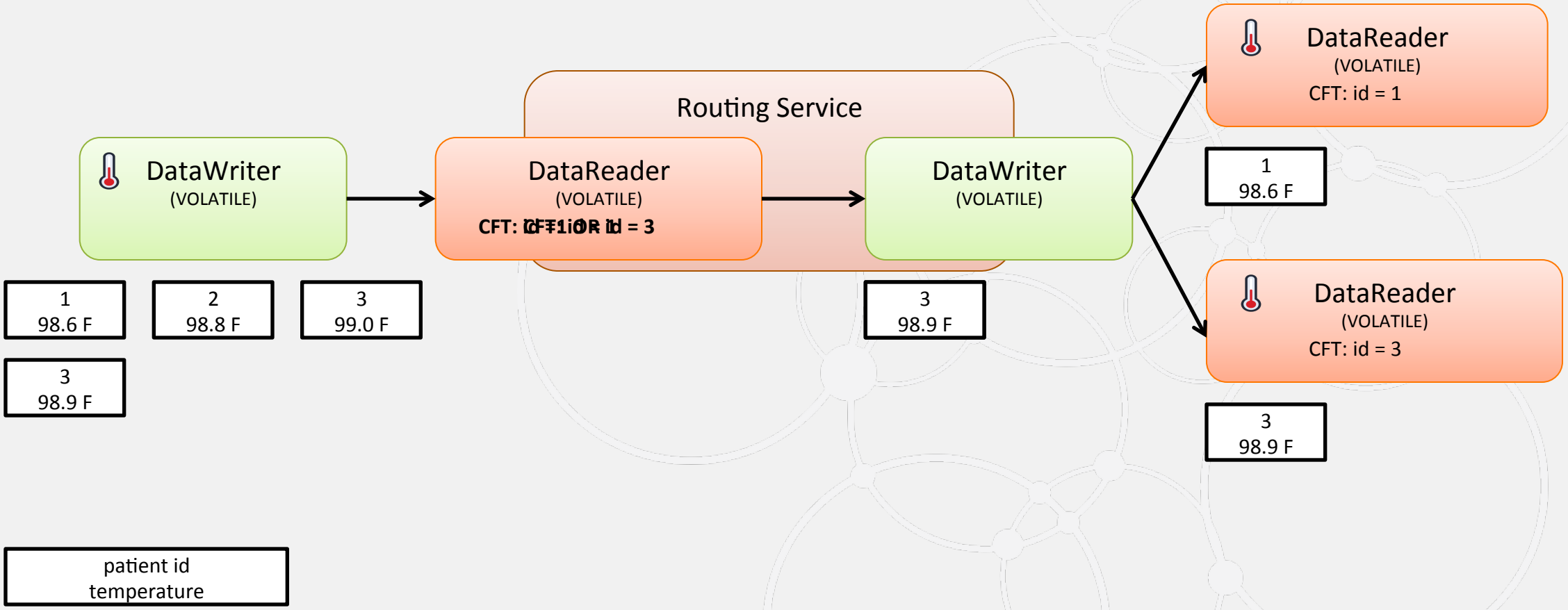


Publish-Subscribe Federated Architecture Challenges

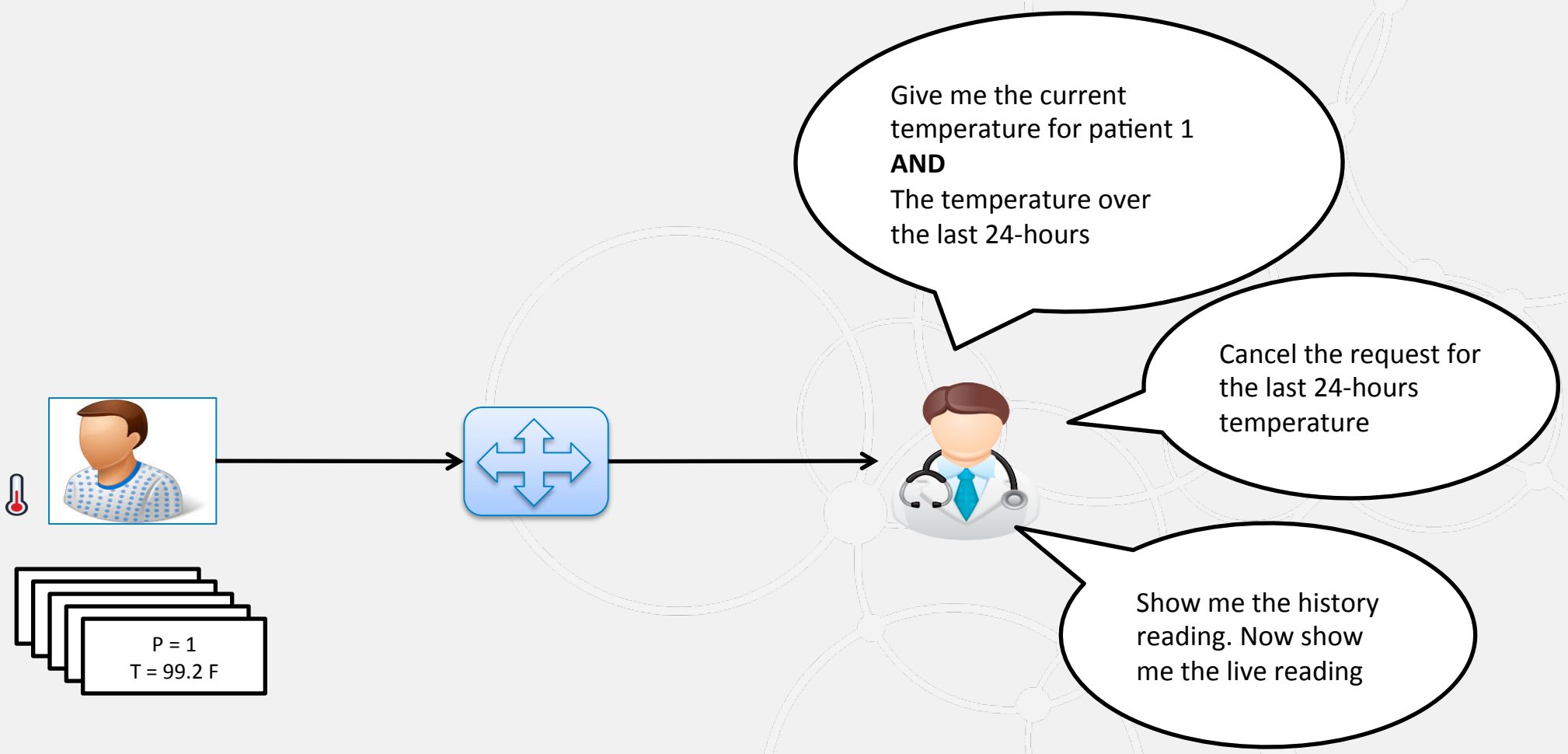
- Scalable subscription to a subset of the live data
  - For example: Subscribe to the current temperature for patient 1
- Scalable retrieval of a subset of the historical data
  - For example: Get the last 24-hour temperature for patient 1

Scalable Subscription To a Subset of the Live Data: CFT Propagation

Available since 5.0.0



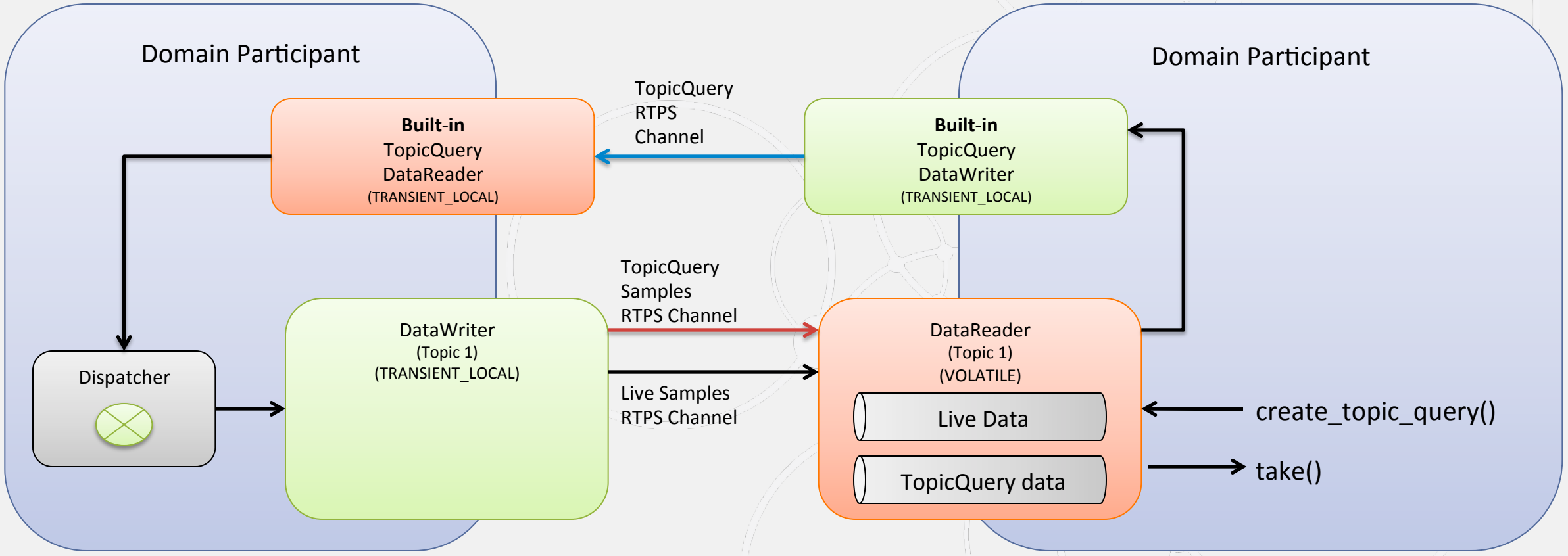
Subscription To a Subset of the Historical Data: Example



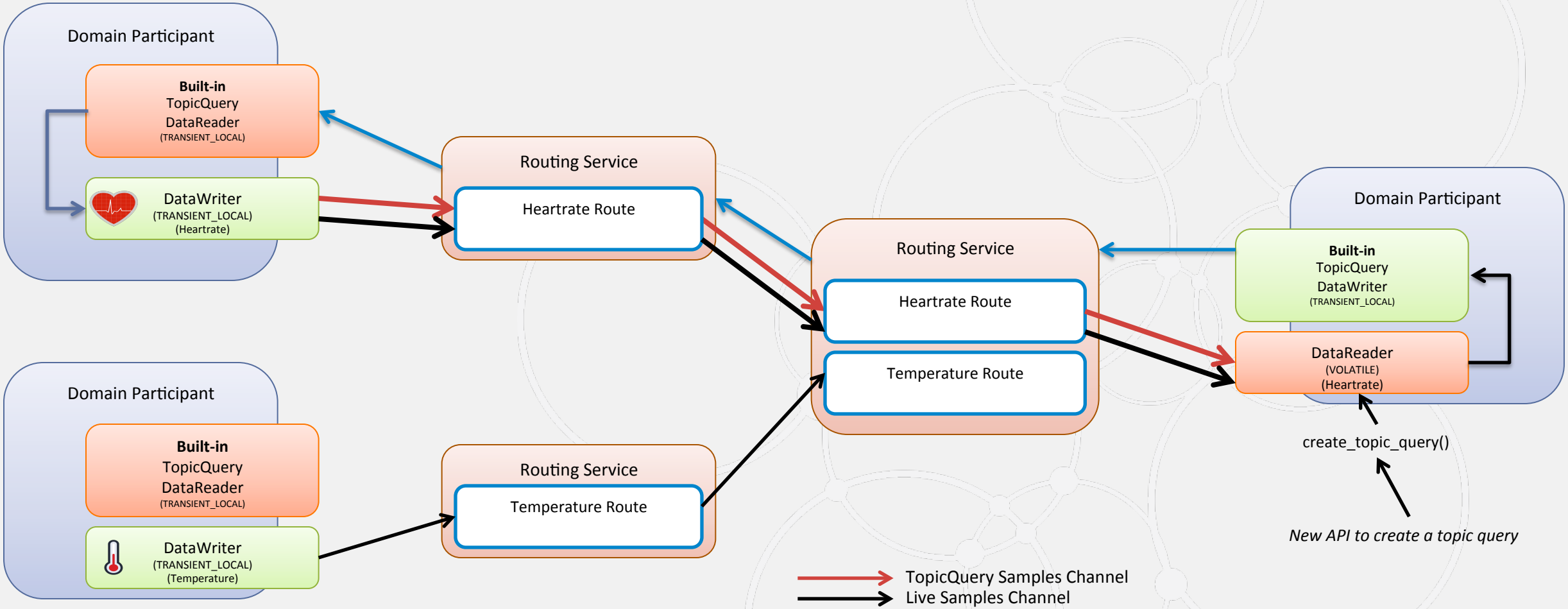
Scalable Subscription To a Subset of the Historical Data

- Historical samples sent in parallel to live data
  - Out-of-band point-to-point channel
- No caching in Routing Services unless explicitly configured (one-off requests)
  - Historical data retrieval with VOLATILE durability
- Ability to choose between reading historical samples or reading live samples
  - Enhanced ReadCondition

TopicQuery Anatomy



TopicQuery Anatomy Through Routing Service



Creating TopicQueries

- New operations on a DataReader to create a delete a TopicQuery
  - It is possible to create multiple TopicQueries

```
struct TopicQuerySelection {
    char * filter_class_name; /* By default SQL */
    char * filter_expression;
    StringSeq filter_parameters;
};

const TopicQuerySelection TOPIC_QUERY_SELECTION_ALL;
const TopicQuerySelection TOPIC_QUERY_SELECTION_DEFAULT;

TopicQuery * DataReader::create_topic_query(const
    TopicQuerySelection & selection);
TopicQuery * DataReader::create_topic_query();

void DataReader::delete_topic_query(TopicQuery * query);

void TopicQuery::get_guid( GUID_t & query_guid );
```

Creating TopicQuery

```
TopicQuerySelection selection;
TopicQuery * query;

subscriber->create_contentfilteredtopic(
    "MyCFT", topic, "P = 1 or P = 3", parameters);

reader_listener = new MyTypeListener();

reader = subscriber->create_datareader(
             topic, DATAREADER_QOS_DEFAULT,
             reader_listener, STATUS_MASK_ALL);
fooReader = FooDataReader::narrow(reader);

selection.filter_expression = "P = 1 or P = 3";
query = reader->create_topic_query(
            selection);

while (true) {
    fooReader->take(
        data_seq, info_seq, LENGTH_UNLIMITED,
        ANY_SAMPLE_STATE,
        ANY_VIEW_STATE, ANY_INSTANCE_STATE);
}

reader->delete_topic_query(query);
```

Take retrieves both, live data, and TopicQuery data.
Use **sample_info. topic_query_guid** to see if data is part of a TopicQuery

Reading TopicQuery Samples

- ReadCondition class extended to allow reading: TopicQuery samples only, Live samples only, TopicQuery and Live samples.

```
typedef enum {
    LIVE_STREAM,
    TOPIC_QUERY_STREAM
} StreamKind;

struct ReadConditionParams {
    SampleStateMask sample_states;
    ViewStateMask view_states;
    InstanceStateMask instance_states;
    StreamKindMask stream_kinds;
};

struct QueryConditionParams : ReadConditionParams {
    char *query_expression;
    DDS_StringSeq query_parameters
};

ReadCondition * DataReader::create_readcondition_w_params(const ReadConditionParams & params);

QueryCondition* DataReader::create_querycondition_w_params(const QueryConditionParams & params);
```

Reading TopicQuery Samples

- User can identify a sample as part of a TopicQuery by looking at SampleInfo

```
struct SampleInfo{
    …
    GUID_t topic_query_guid; /* INVALID value means live sample */
    SampleFlag flag; /* INTERMEDIATE_TOPIC_QUERY_SAMPLE */
    …
};
```

Reading TopicQuery Samples

```
CachedDataSelection selection;
TopicQuery * query;
ReadConditionParams condParams;
ReadCondition * cond;

subscriber->create_contentfilteredtopic(
    "MyCFT", topic, "P = 1", parameters);

reader_listener = new MyTypeListener();

reader = subscriber->create_datareader(
             topic, DATAREADER_QOS_DEFAULT,
             reader_listener, STATUS_MASK_ALL);
fooReader = FooDataReader::narrow(reader);

condParams.stream_kinds = TOPIC_QUERY_STREAM;
cond = reader->create_readcondition_w_params(condParams);

selection.filter_expression = "P = 1";
query = reader->create_topic_query(
    selection);
```

Reading TopicQuery Samples

```
/* Read TopicQuery samples first, up to MAX_SECS */
count = 0;
startReadingLive = false;

while (!startReadingLive) {
    /* The 'take' call will only take TopicQuery samples */
    fooReader->take_w_condition(
        data_seq, info_seq, cond);

    for (int i = 0; i < data_seq.length(); ++i, count++) {
        if (count == 0) {
            startTT = info_seq[i].source_timestamp;
        } else {
            endTT = info_seq[i].source_timestamp;

            if ((endTT.sec - startTT.sec) > MAX_SECS ||
                   !(info_seq[i].flags &
                     INTERMEDIATE_TOPIC_QUERY_SAMPLE)) {
                startReadingLive = true;
            }
        }
    }

    /* Sample logic */
}
```

```
/* Then read both live and TopicQuery samples */
while (true) {
    /* The 'take' call will take both: LIVE and
OLD samples in
       * no specific order */
    fooReader->take(
        data_seq, info_seq,
        LENGTH_UNLIMITED,
        ANY_SAMPLE_STATE,
        ANY_VIEW_STATE, ANY_INSTANCE_STATE);

    /* Sample logic */
}
```

Dispatching TopicQueries

- TopicQueries are dispatched by **TRANSIENT_LOCAL** DataWriters matching the DataReader that issued the query
  - One thread per Publisher to send TopicQuery samples

```
struct AsynchronousPublisherQosPolicy {
    DDS_Booolean disable_asynchronous_topic_query_write;
    ThreadSettings_t asynchronous_topic_query_thread;
    …
};

struct PublisherQos {
    AsynchronousPublisherQosPolicy asynchronous_publisher;
    …
};

struct TopicQueryDispatchQosPolicy {
    DDS_Boolean enable;
    Duration_t publication_period;
    DDS_Long samples_per_period;
};

struct DataWriterQos {
    …
    TopicQueryDispatchQosPolicy topic_query;
```

Debugging Cached Data Request

- TopicQueries are sent on a new built-in Topic: **ServiceRequestBuiltinTopic**
  - User will be able to monitor TopicQueries by using the ServiceRequestBuiltinTopic DataReader on a DomainParticipant

```
const int TOPIC_QUERY_SERVICE_CLASS = 0;

struct ServiceRequest{
    int service_class;      /* key field */
    GUID_t request_id;      /* key field */
    OctetSeq request_body; /* Encapsulates TopicQuery */
}; /* EXTENSIBLE_EXTENSIBILITY */

struct TopicQuerySelection {
    char* filter_class_name;
    char* filter_expression;
    StringSeq filter_parameters;
}; /* MUTABLE_EXTENSIBILITY */

struct TopicQueryInfo {
    TopicQuerySelection selection;
    SequenceNumber sync_sequence_number;
    char * topic_name;
    GUID_t original_related_reader_guid;
    …
};/* MUTABLE_EXTENSIBILITY */
```

Debugging TopicQuery Dispatching

- Received TopicQuery is dispatched to a DataWriter only when the DataReader issuing the TopicQuery matches the DataWriter
- New DataWriter Status to notify about TopicQuery matching

```
struct ServiceRequestMatchedStatus{
    DDS_Long total_count;
    DDS_Long total_count_change;
    DDS_Long current_count;
    DDS_Long current_count_peak;
    DDS_Long current_count_change;
    GUID_t last_request_id;
    int last_service_class; /* TOPIC_QUERY_SERVICE_CLASS */
};

void DataWriterListener::on_service_request_matched(DataWriterListener *writer, const
ServiceRequestMatchedStatus& status);
```

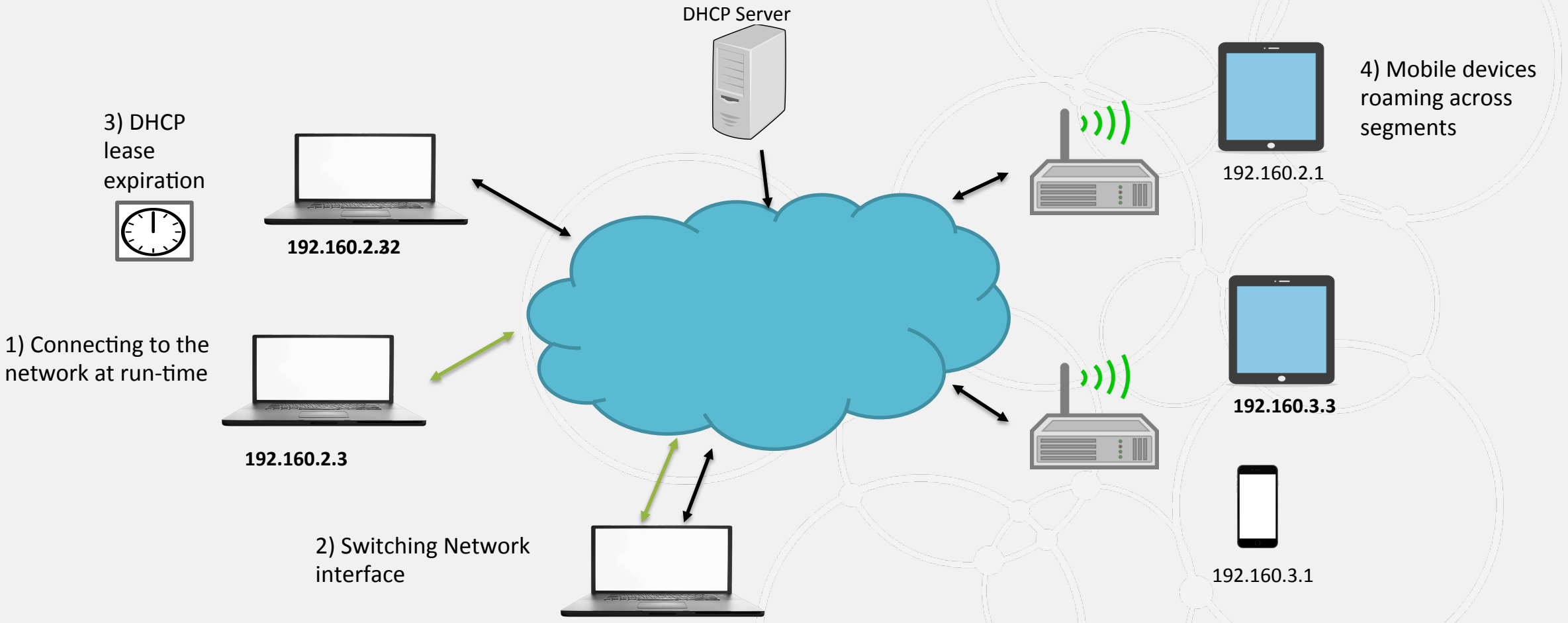
IP Mobility

Enabling communications in mobile networks

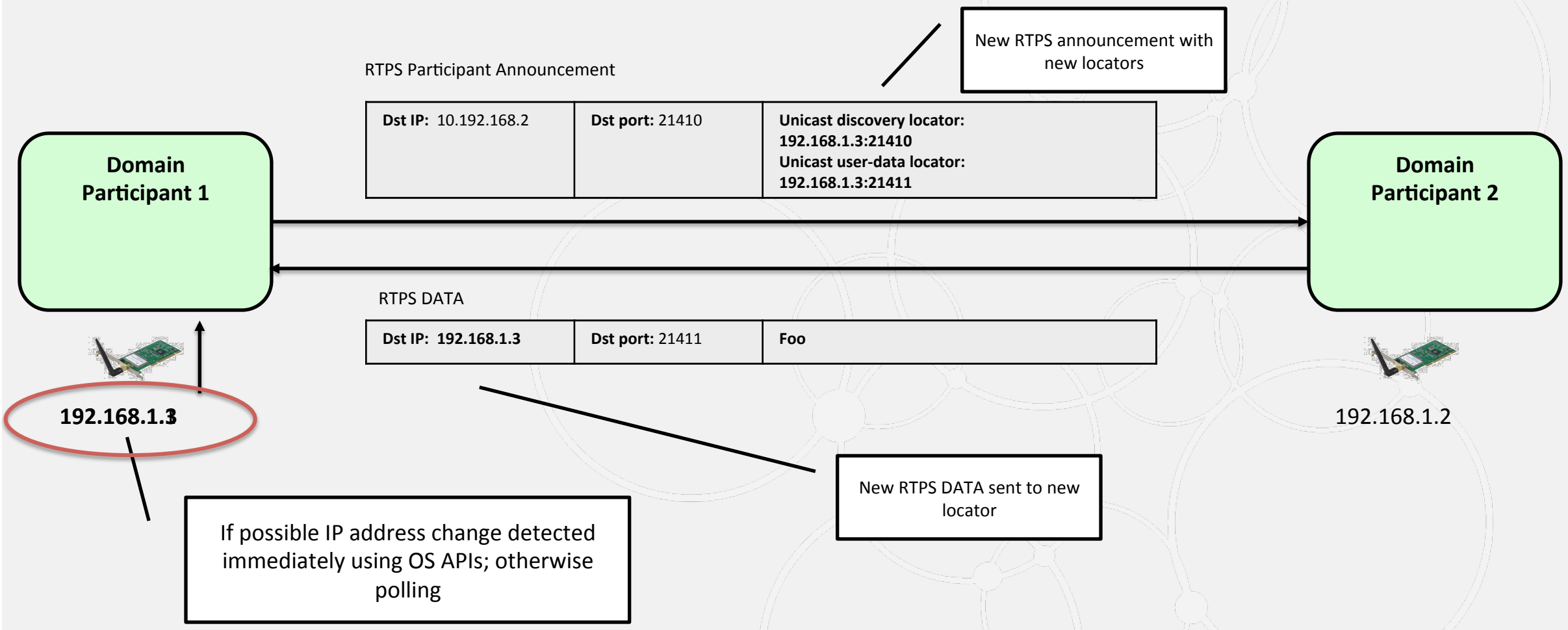
Problem Description

- Changes of IP addresses not supported after DomainParticipant is enabled → Communication will **STOP**
- Affected Transports: UDPv4, UDPv6, TCPv4 and TLSv4 in symmetric mode, DTLSv4, WAN, LBRTPS, ZRTPS

Use Cases



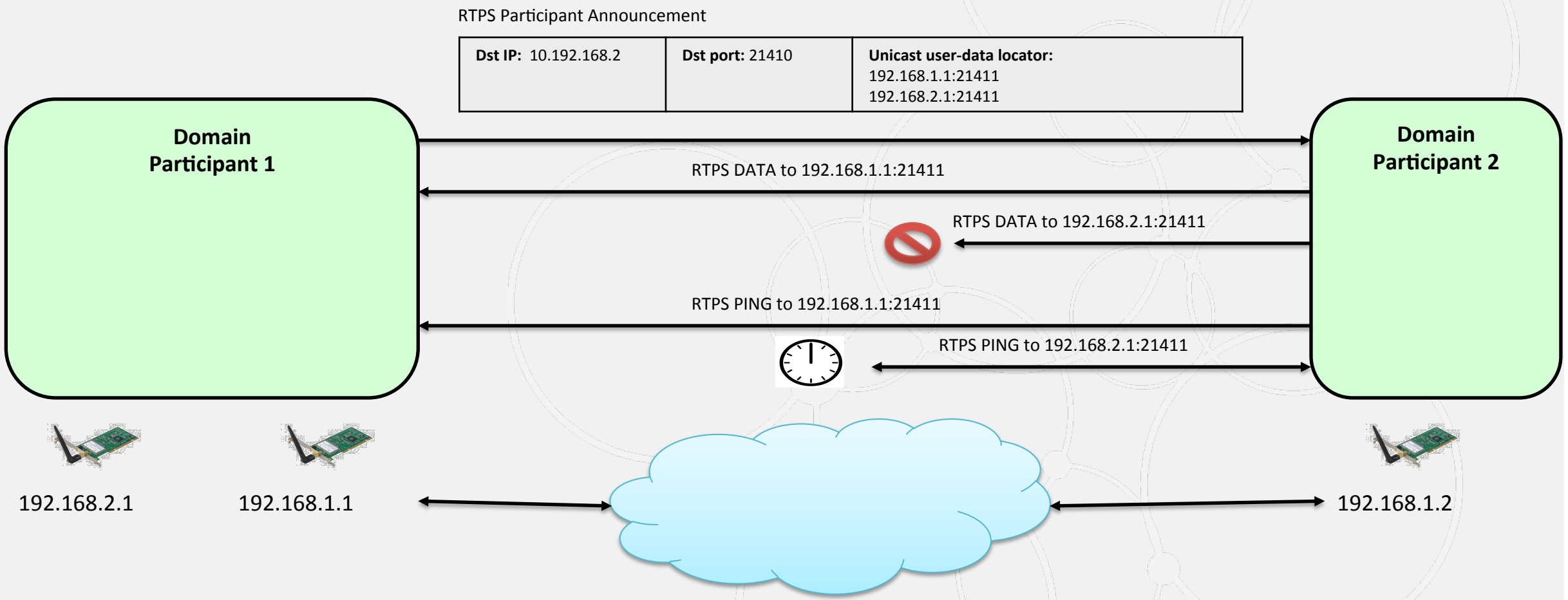
Feature I. Detection and Notification of IP Address Changes



Feature I. Configuration

- New GUID generation mode (not enabled by default):
  - Set wire_protocol.rtps_auto_id_kind to **DDS_RTPS_AUTO_ID_FROM_UUID**
- Interface polling period configured using transport property when automatic detection not enabled:
  - dds.transport.UDPv4.builtin.interface_poll_period
- Backward compatibility option
  - dds.transport.UDPv4.builtin.disable_interface_tracking

Feature II. Don’t Send Data To Non-Reachable IP Addresses



Feature II. Configuration

- Feature **not enabled by default**
- Configuration. Local Participant
  - **participant_qos.discovery_config.locator_reachability_assert_period**: Period at which local Participant PINGs all the locator it has discovered.
  - **participant_qos.discovery_config.locator_reachability_lease_duration** : If a remote Participant does not receive a PING for one if its locators before the local Participant locator_reachability_lease_duration expires, the locator will be considered non-reachable and this will be notified to the local Participant.
  - **participant_qos.discovery_config.locator_reachability_change_detection_period**: the maximum period at which a local Participant will notify remote Participants about non reachable locators.

Feature II. Built-in Channels

- Two new Built-in Channels:
  - One Stateless Channel to send PING messages for remote locators
  - One TRANSIENT_LOCAL RELIABLE channel to announce the set of locators in which a Participant can be reached from a remote Participant

Feature III. IP Address Selection (Not Supported in 5.3)

RTPS Participant Announcement

Dst IP: 10.192.168.2	**Dst port:** 21410	**Unicast user-data locator:** **192.168.1.2:21411** → Priority **LOW**
		192.168.1.2:21411 → Priority **LOW**

Domain Participant 1

Domain Participant 2

RTPS DATA to **192.168.1.2:21411**

192.168.1.2

192.168.1.1

Disable

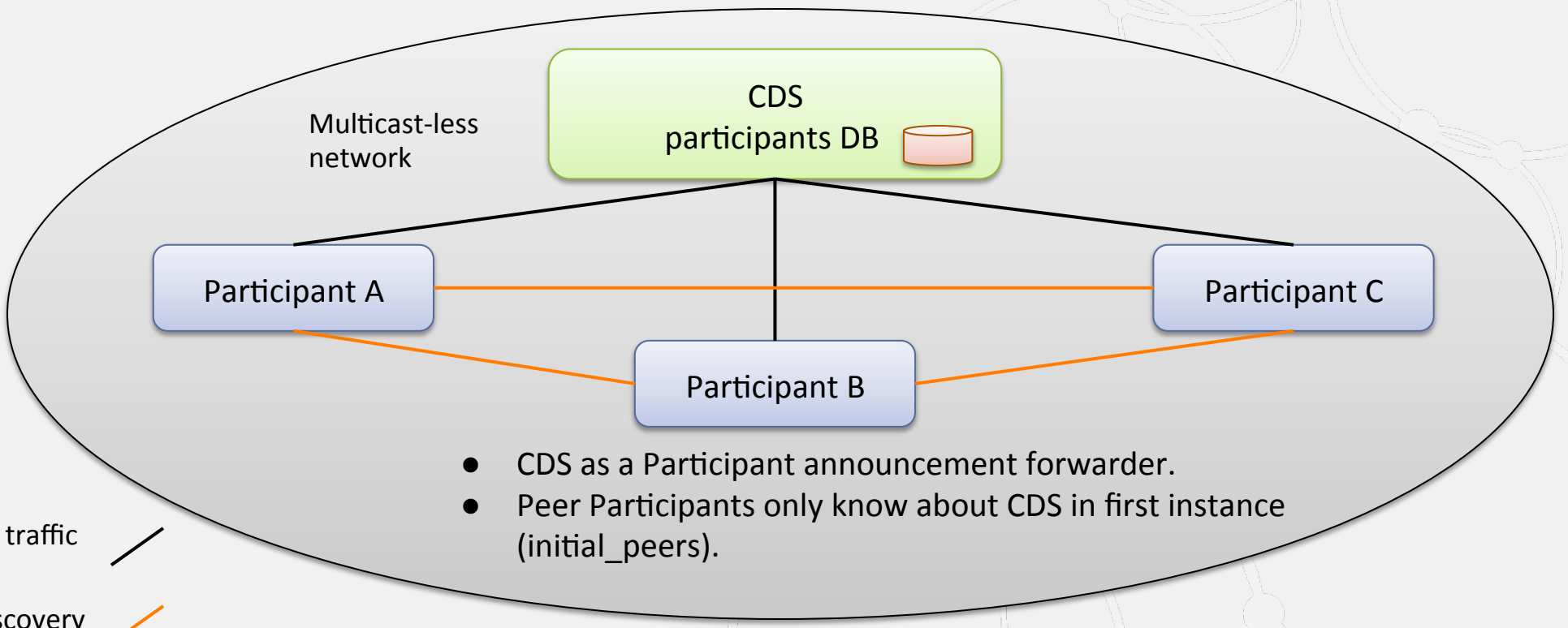
192.168.1.3

Cloud Discovery Service

Provisioning discovery in cloud-based environment

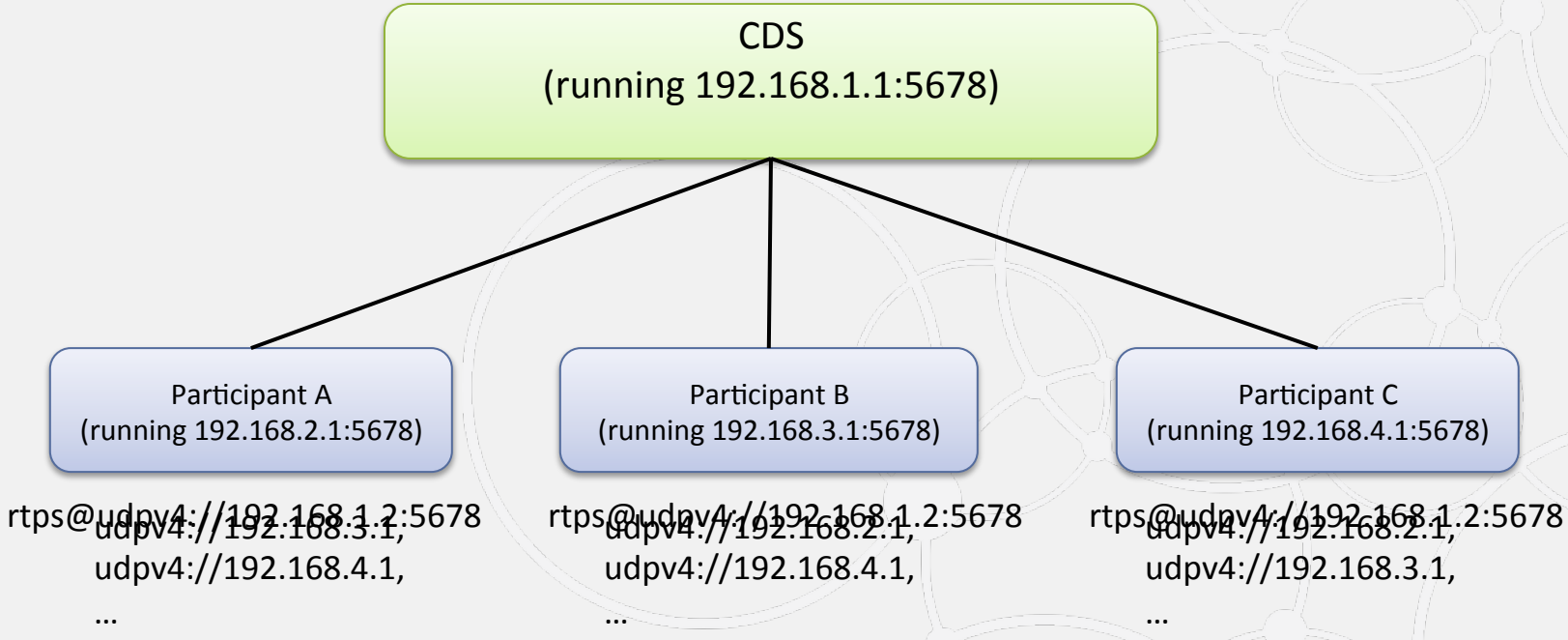
What is Cloud Discovery Service?

Cloud Discovery Service (CDS) is a mediator for the discovery process in environments where multicast is not available.



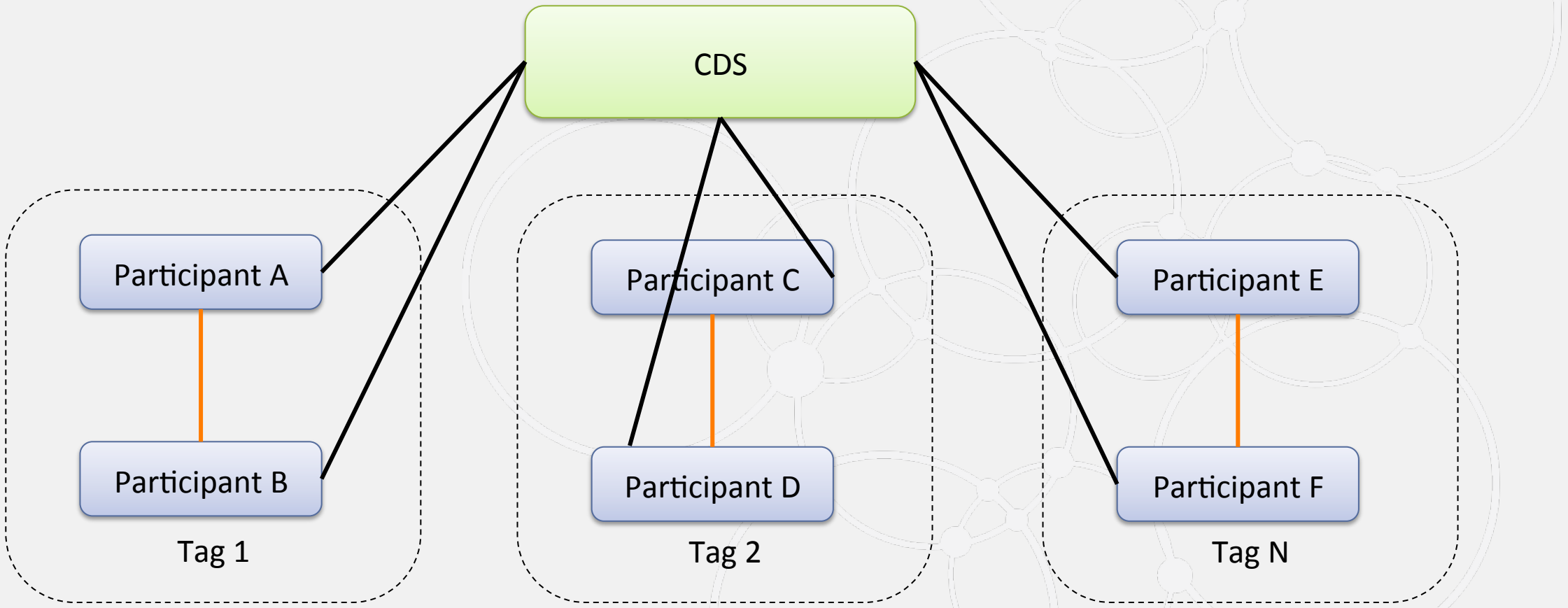
- CDS as a Participant announcement forwarder.
- Peer Participants only know about CDS in first instance (initial_peers).

Initial Peers

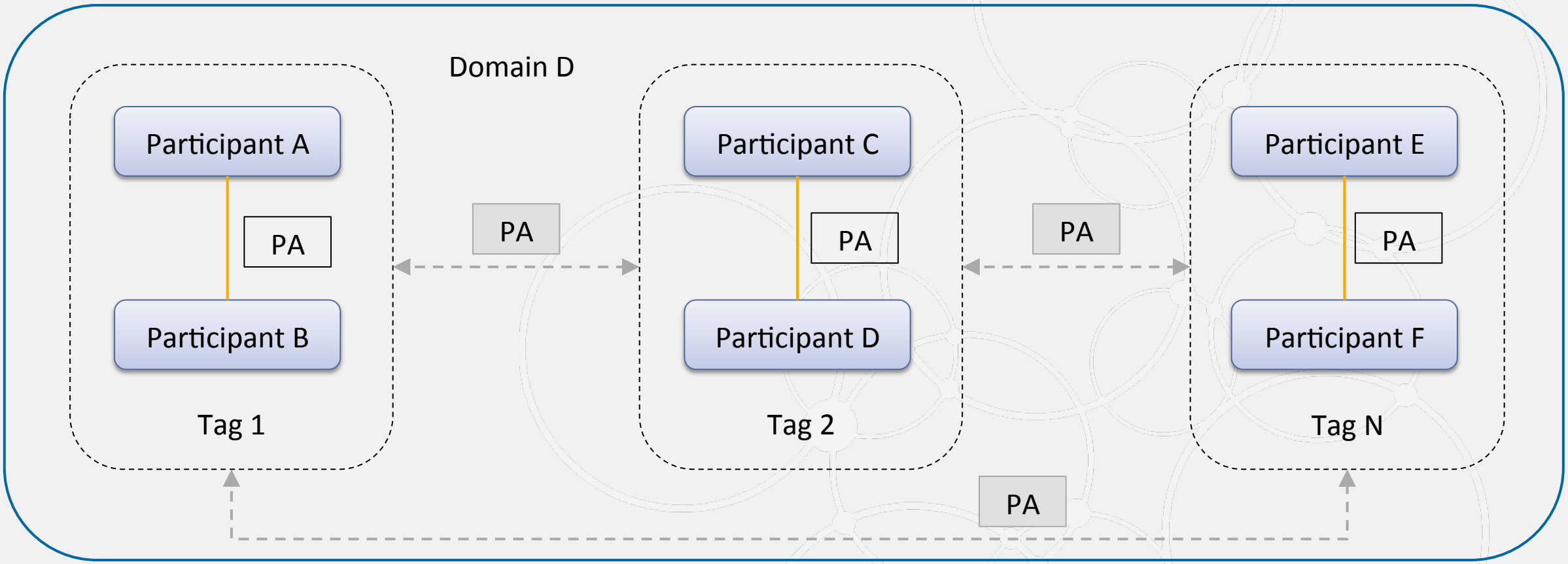


Domain Tags

A domain tag is a logical space within a domain. Domain tags are isolated from each other.



Domain Tags



XML Configuration Example

```xml
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../xsd/rti_cloud_discovery_service.xsd">

    <cloud_discovery_service name="allDomainsTCPv4Wan">
        <annotation>
            <documentation><![CDATA[
                Forwards all domains using pre-registered TCPv4 WAN
                transport.
                ]]>
            </documentation>
        </annotation>

        <domain_list>
            <allow_domain_id>
                [30-40]
            </allow_domain_id>
        </domain_list>

        <transport>
            <element>
                <alias>tcpv4_wan</alias>
                <receive_port>7400</receive_port>
            </element>
        </transport>
    </cloud_discovery_service>

</dds>
```

Allowed/Denied domain IDs are configurable

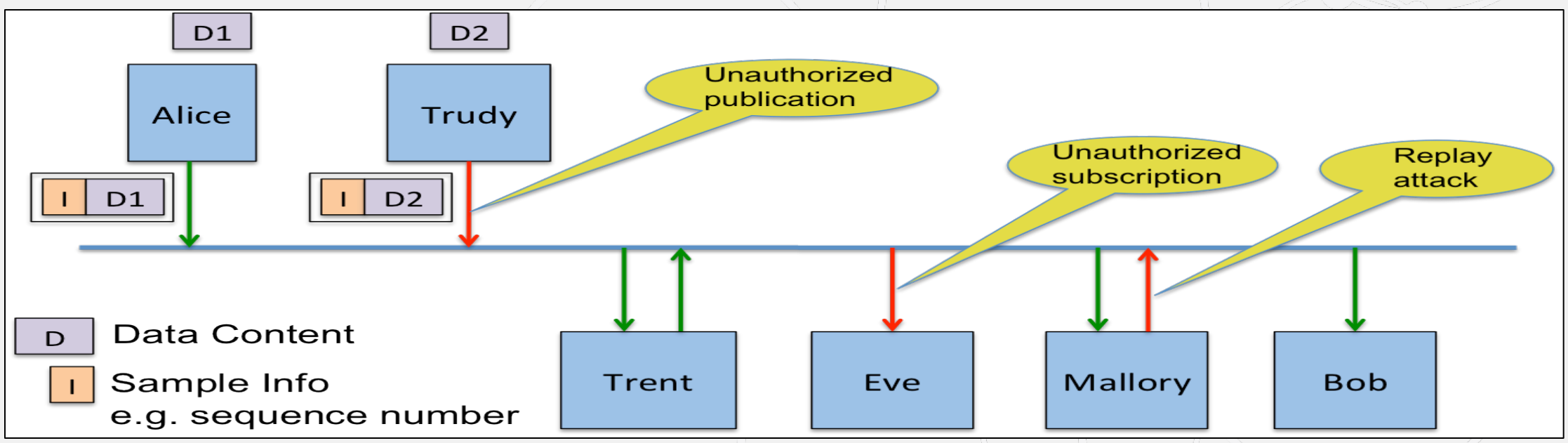
Support for TCPv4, TLSv4, UDPv4, UDPv6

Connext DDS Secure

Securing the DDS bus data

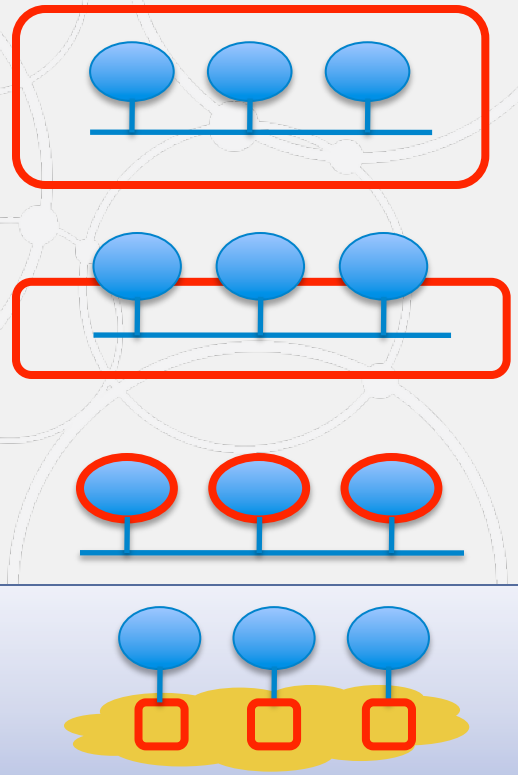
Security Threats to a DDS System

Alice: Allowed to publish topic T
Bob: Allowed to subscribe to topic T
Eve: Non-authorized eavesdropper
Trudy: Intruder
Trent: Trusted infrastructure service
Mallory: Malicious insider



Security Boundaries

- System Boundary
- Network Transport
- Host
- Data & Information Flows

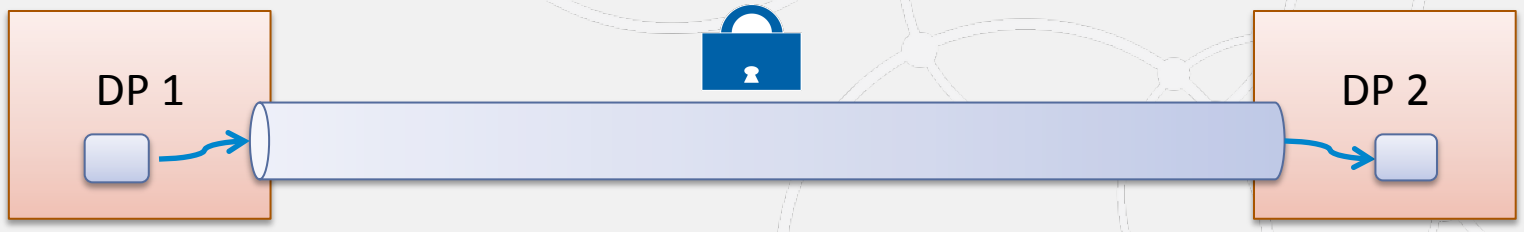


Approaches to Protect DDS

- Transport Layer Security
- Fine-Grained Security

Transport-Level Secure Data Transfer

1. Authenticate
   - Verify your identity
2. Securely exchange cryptographic keys
3. Use keys to:
   - Encrypt data
   - Add a message authentication code



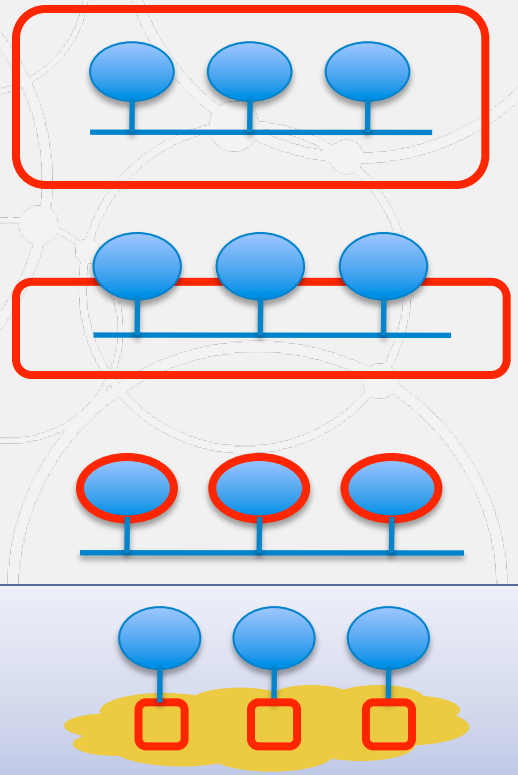
Transport-Level Secure Data Transfer In RTI Connext DDS

Three Connext DDS transports available in Connext DDS

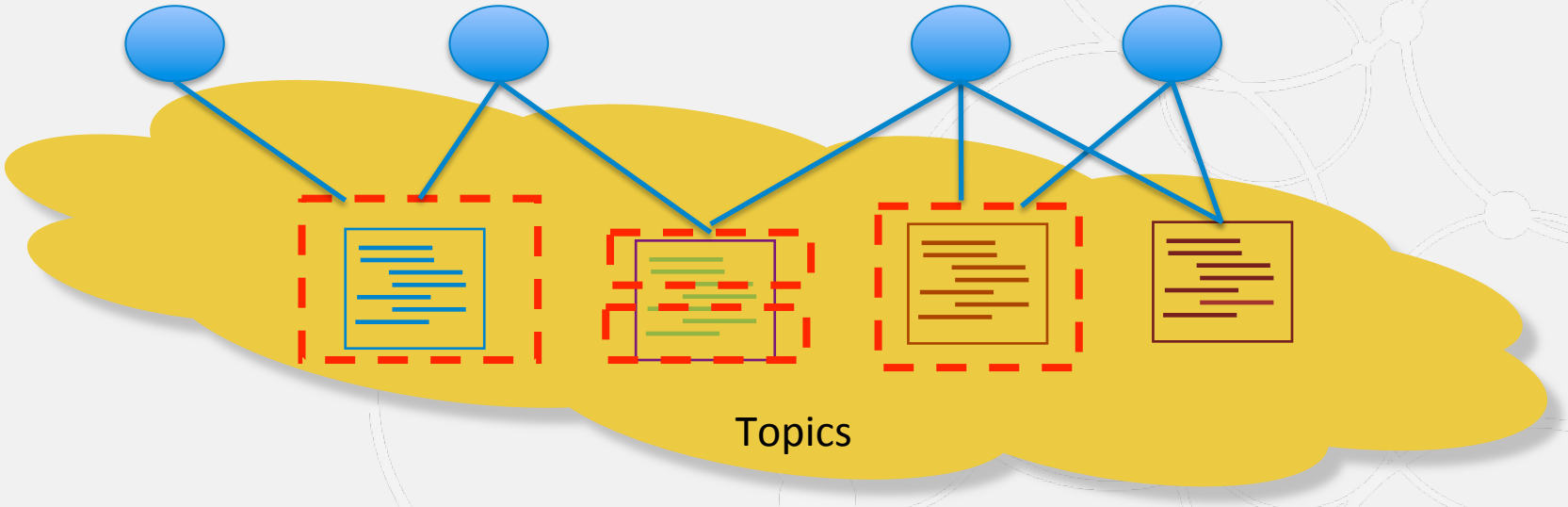
- RTI Secure WAN Transport
  - WAN UDP transport that uses UDP hole punching to traverse NATs
  - Optional transport authentication and encryption using DTLS
- RTI Secure DTLS Transport
  - LAN UDP transport
  - Transport authentication and encryption using DTLS
- RTI Secure TCP Transport
  - WAN/LAN TCP transport
  - Optional transport authentication and encryption using TLS

Security Boundaries

- System Boundary
- Network Transport
- Host
- Data & Information Flows



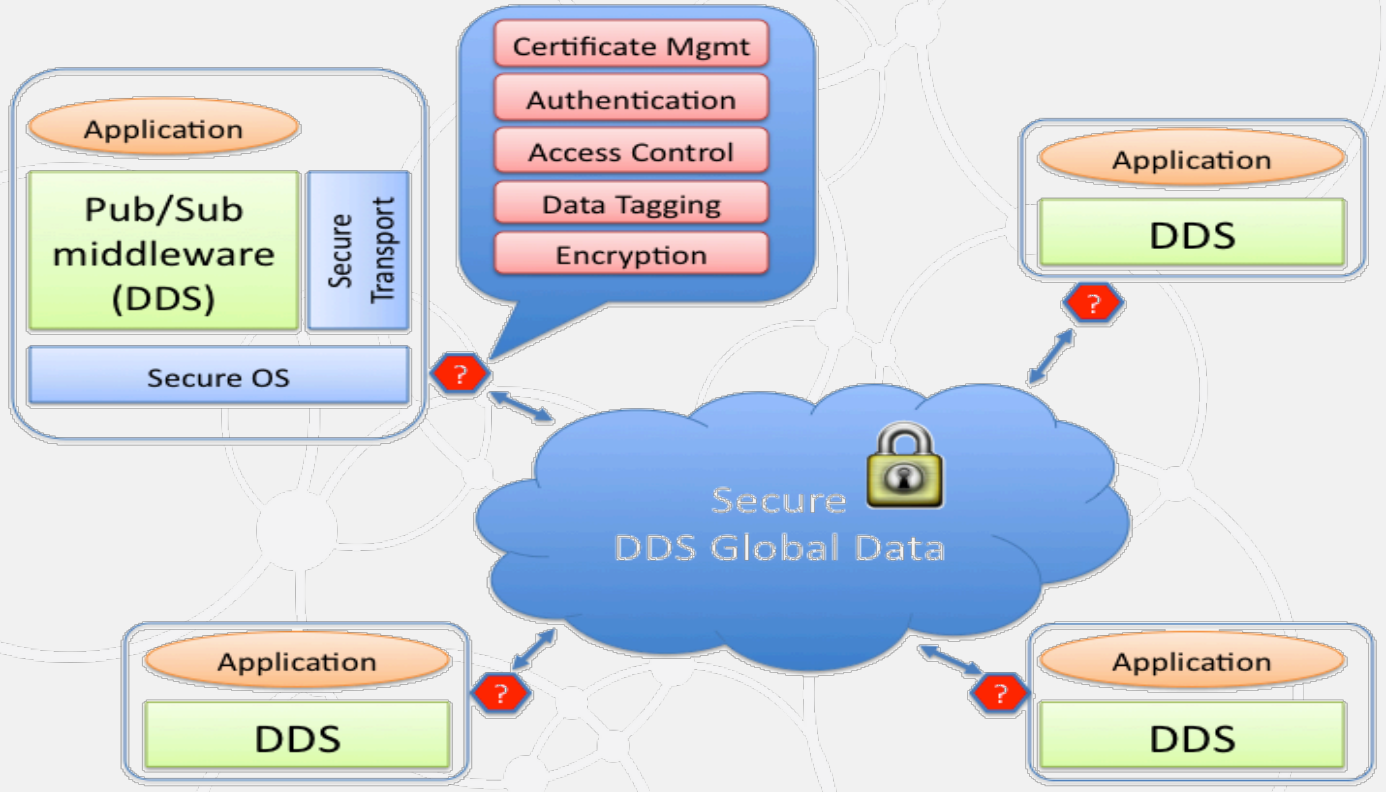
Fine-Grained Data-Centric Security



- Access control per Topic
- Read versus-write permissions
- Instance-specific permissions

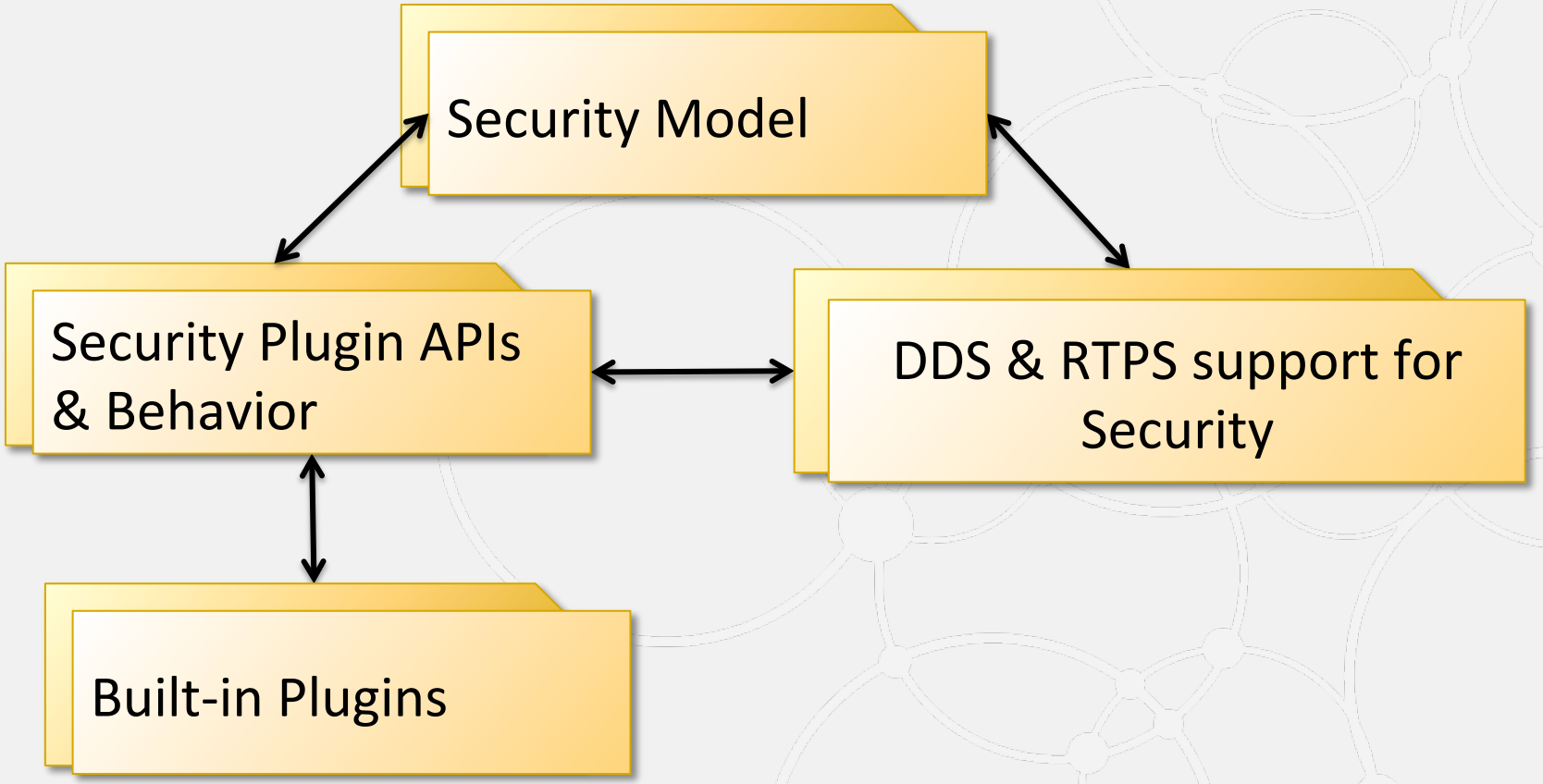
DDS Security Standard

- DDS entities are authenticated
- DDS enforces access control for domains/Topics/…
- DDS maintains data integrity and confidentiality
- DDS enforces non-repudiation
- DDS provides availability through reliable access to data



…while maintaining DDS interoperability & high performance

DDS Security Standard Covers Four Related Concerns



DDS Security Model

Concept	DDS Security Model
Subject	DomainParticipant Application joining a DDS domain
Protected Objects	Domain (by domain_id) Topic (by topic name) DataObjects (by instance/key)
Protected Operations	Domain.join Topic.read Topic.write Data.createInstance Data.writeInstance Data.deleteInstance
Access Control Policy Control	Configurable via Plugin
Built-in Access Control Mode	Per-DomainParticipant Permissions: What Domains and Topics it can JOIN/READ/WRITE

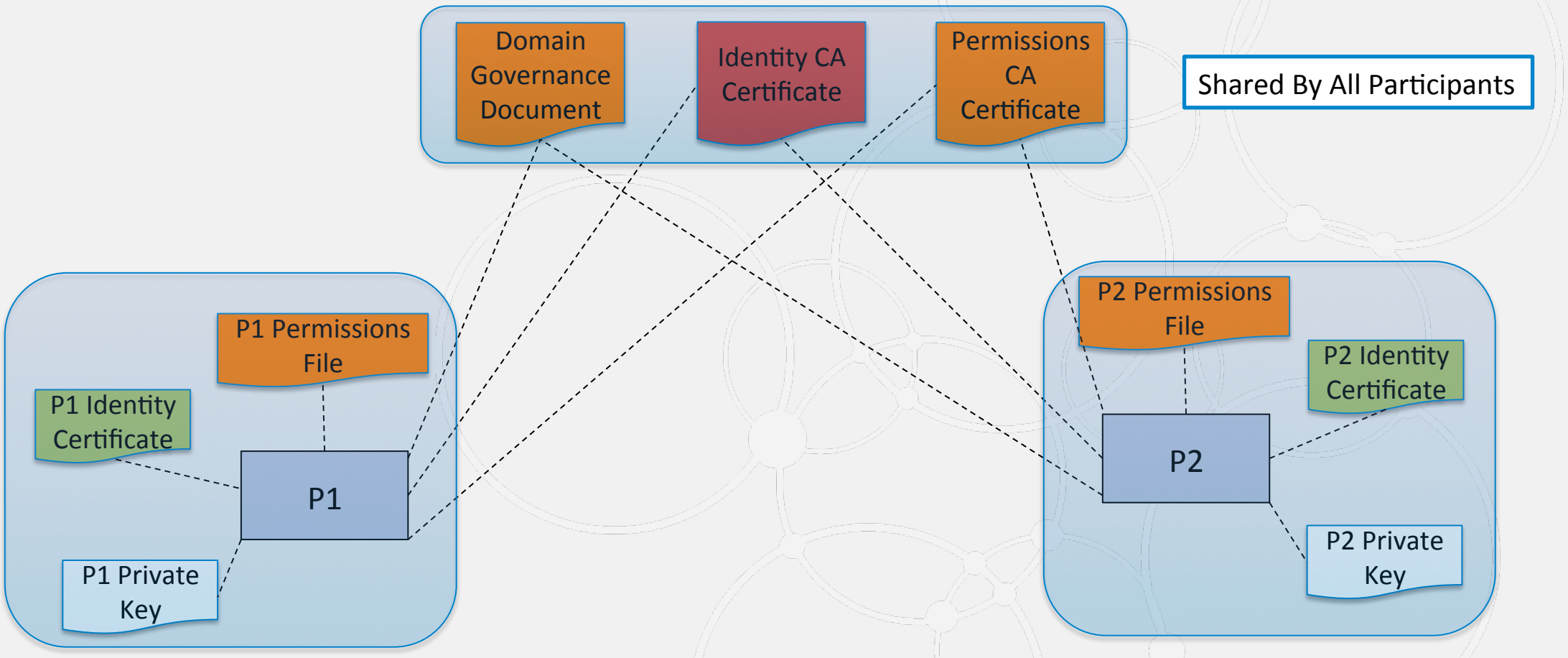
Pluggable Architecture

Service Plugin	Purpose	Interactions
Authentication	Authenticate the principal that is joining a DDS Domain. Handshake and establish shared secret between participants	The principal may be an application/process or the user associated with that application or process. Participants may messages to do mutual authentication and establish shared secret
Access Control	Decide whether a principal is allowed to perform a protected operation.	Protected operations include joining a specific DDS domain, reading a Topic, writing a Topic, etc.
Cryptography	Perform the encryption and decryption operations. Create & Exchange Keys. Compute digests, compute and verify Message Authentication Codes. Sign and verify signatures of messages.	Invoked by DDS middleware to encrypt data compute and verify MAC, compute & verify Digital Signatures
Logging	Log all security relevant events	Invoked by middleware to log
Data Tagging	Add a data tag for each data sample	

Built-in Plugins

SPI	Built-in Plugin	Notes
Authentication	DDS:Auth:PKI-DH	Uses PKI with a pre-configured shared Certificate Authority. DSA and Diffie-Hellman for authentication and key exchange Establishes shared secret
AccessControl	DDS:Access:Permissions	Governance Document and Permissions Document Each signed by shared Certificate Authority **Security configuration per Domain and Topic** **Access control per Domain and Topic**
Cryptography	DDS:Crypto:AES-GCM-GMAC	**Automatic key distribution** AES-128/192/256-GCM for encryption SHA1 and SHA256 for digest AES-128/192/256-GMAC for MAC **Separate keys per DW and DR** **Transparent secure multicast**
Logging	DDS:Logging:DDS_LogTopic	

Configuring & Deploying DDS Security



Built-in Plugins: XML Governance Document

- Specifies how a domain should be secured

- Signed by the Permissions CA

- Provided to the plugins using the PropertyQosPolicy on the DomainParticipantQos

```
<?xml version="1.0" encoding="UTF-8"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../schema/dds_security_governance.xsd">
    <domain_access_rules>
        <domain_rule>
            <domains>
                <id_range>
                    <min>0</min>
                </id_range>
            </domains>
            <allow_unauthenticated_participants>false</allow_unauthenticated_participants>
            <enable_join_access_control>true</enable_join_access_control>
            <discovery_protection_kind>ENCRYPT</discovery_protection_kind>
            <liveliness_protection_kind>ENCRYPT</liveliness_protection_kind>
            <rtps_protection_kind>SIGN</rtps_protection_kind>
            <topic_access_rules>
                <topic_rule>
                    <topic_expression>*</topic_expression>
                    <enable_discovery_protection>true</enable_discovery_protection>
                    <enable_read_access_control>true</enable_read_access_control>
                    <enable_write_access_control>true</enable_write_access_control>
                    <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
                    <data_protection_kind>ENCRYPT</data_protection_kind>
                </topic_rule>
            </topic_access_rules>
        </domain_rule>
    </domain_access_rules>
</dds>
```

Built-in Plugins: XML Permissions Document

- Contains the permissions of the Domain Participants

- Signed by the Permissions CA

- Provided to the plugins using the PropertyQosPolicy

on the DomainParticipantQos

```
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../schema/dds_security_permissions.xsd">
    <permissions>
        <grant name="ParticipantA">
            <subject_name>C=US, ST=CA, O=Real Time Innovations, CN=dtlsexample/emailAddress=me@rti.com</subject_name>
            <validity>
                <!-- Format is CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm] in GMT -->
                <not_before>2013-06-01T13:00:00</not_before>
                <not_after>2023-06-01T13:00:00</not_after>
            </validity>
            <allow_rule>
                <domains>
                    <id>0</id>
                </domains>
                <publish>
                    <topics>
                        <topic>Cir*</topic>
                    </topics>
                    <partitions>
                        <partition>P1*</partition>
                    </partitions>
                </publish>
                <subscribe>
                    <topics>
                        <topic>Sq*</topic>
                    </topics>
                    <partitions>
                        <partition>P2*</partition>
                    </partitions>
                </subscribe>
                <subscribe>
                    <topics>
                        <topic>Triangle</topic>
                    </topics>
                    <partitions>
                        <partition>P*</partition>
                    </partitions>
                </subscribe>
            </allow_rule>
            <default>ALLOW</default>
        </grant>
    </permissions>
</dds>
```

Configuration Possibilities

- Are “legacy” or un-identified applications allowed in the Domain?
  - Yes (if configured) unauthenticated applications will:
    - See the “unsecured” discovery Topics
    - Be allowed to read/write the “unsecured” Topics
- Is a particular Topic discovered over protected discovery?
  - If so it can only be seen by “authenticated applications”

Configuration Possibilities

- Is the access to a particular Topic protected?
  - If so only authenticated applications with the correct permissions can read/write
- Is data on a particular Topic protected? How?
  - If so data will be sent signed or encrypted+signed
- Are all protocol messages signed? Encrypted?
  - If so only authenticated applications with right permissions will see anything

Connext DDS 5.3 Built-in Plugins Limitations

- Only sign for RTPS supported
- Only encryption + sign for discovery an application topics is supported

Connext DDS 5.3 Custom Plugins Limitations

- Data tagging not supported
- Instance access control not supported
- Revoking DomainParticipants identities not supported

Key Benefits

More Powerful Than Other Secure Middleware Technologies

- Standard & Interoperable
- Scalable: Supports multicast
- Fine-grain: Control Topic-level aspect
- Flexible: Build your own plugins
- Generic: Works over any transport
- Transparent: No changes to Application Code!

Usability & Debuggability

Heap Monitoring

- Feature to monitor middleware heap memory allocations in native memory space
  - Useful to debug/analyze unexpected memory growth
- New API to enable/disable heap monitoring and take heap allocations snapshots
- Platform-independent feature
- Works with Release/Debug libraries
- Supported by all infrastructure services

Heap Monitoring Usage

DDS_Boolean	**NDDS_Utility_enable_heap_monitoring** () Starts monitoring the heap memory used by RTI Connext.
void	**NDDS_Utility_disable_heap_monitoring** () Stops monitoring the heap memory used by RTI Connext.
DDS_Boolean	**NDDS_Utility_pause_heap_monitoring** () Pauses heap monitoring.
DDS_Boolean	**NDDS_Utility_resume_heap_monitoring** () Resumes heap monitoring.
DDS_Boolean	**NDDS_Utility_take_heap_snapshot** (const char *filename, **DDS_Boolean** print_details) Save the current heap memory usage into a file.

Two new command-line parameters for infrastructure services:

-heapSnapshotPeriod <sec>

-heapSnapshotDir <dir>

Heap Snapshot File Example

Current process vsize 6803566592
Current process rsize 1069948928
Current heap usage 210951592
High watermark 212340328
Alloc count 56309122
Free count 54350123

block_id, timestamp, block_size, pool_alloc, pool_buffer_size, pool_buffer_count, topic_name, activity, alloc_method_name, type_name

12830, 1492838970, 104, MALLOC, 0, 0, PRESServiceRequest, PRESCstReaderCollator_new, RTIOsapiHeap_allocateStructure, struct REDAFastBufferPool

Logging Improvements

- Support for writing into multiple files with logging infrastructure

```
bool NDDSConfigLogger::set_ouput_file_set(
        const char *file_prefix,
        const char *file_suffix,
        int max_capacity,
        int max_files)
```

- Serialization/Deserialization errors print TopicName and TypeName and error cause (for example unexpected enum value)

Robustness

- Endurance test
- Static code analysis using cppcheck
- Linux warning free compilation
- 10 Gb performance
- Multicast scalability test
- AIT (Automated Install Testing)

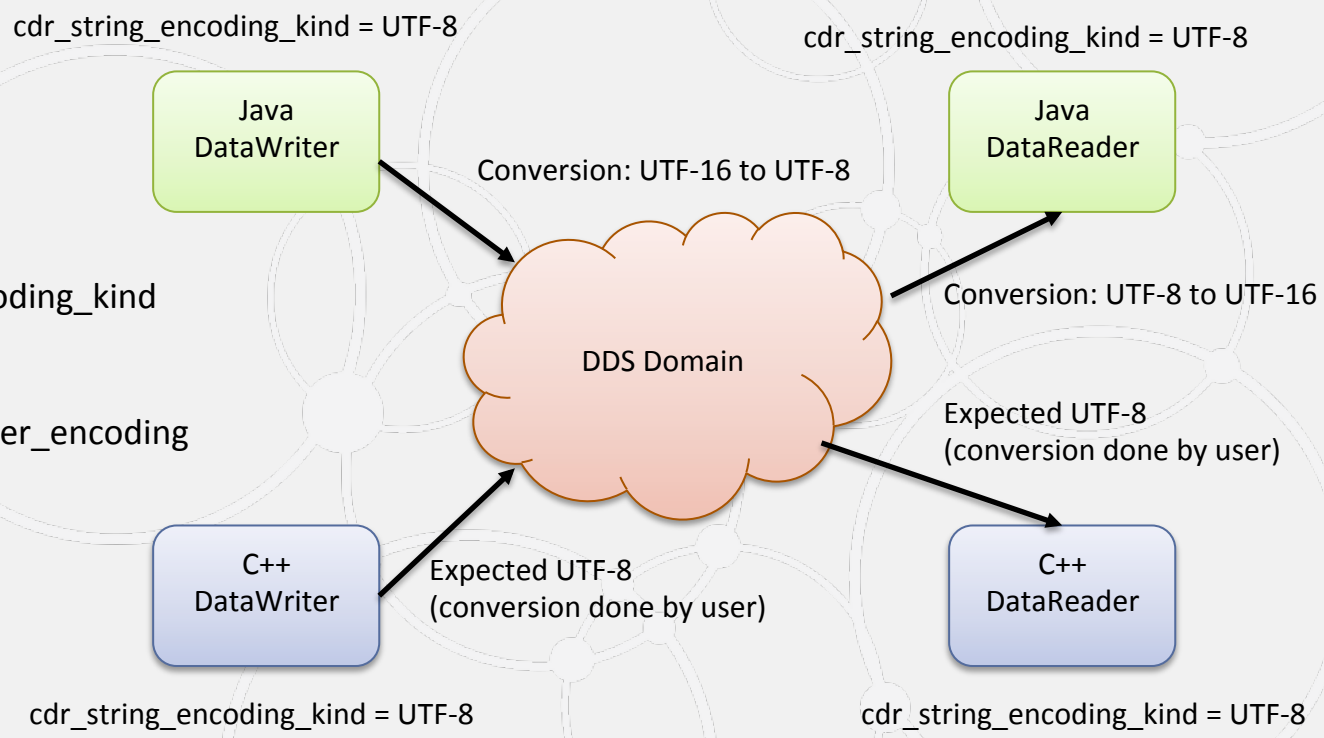
Other Features And Products

RTI Connext DDS

- **Usability:**
  - Support for IDL string encoding configuration: UTF-8, and ISO-8859-1 (default)

Generated types and built-in types:
dds.data_xxxx.type_support.cdr_string_encoding_kind

DynamicData:
DDS_DynamicDataProperty_t.string_character_encoding



RTI Code Generator

- **Usability:**
  - New modern C++ (C++03, C++11) TypePlugin that maps IDL strings to std::string and IDL sequences to std::vector
    - To enable, use **–stl** command-line option in rtiddsgen
  - Ability to generate constructor/destructor and map IDL string to std::string for not modern C++ (C++)
    - To enable, use **–constructor** and **–useStdString** in rtiddsgen

RTI Code Generator

- **Usability:**
  - New data_to_string API with multiple output formats: DEFAULT, XML, JSON

```
DDS_ReturnCode_t FooTypeSupport_data_to_string(
    Foo*sample,
    char *str,
    DDS_UnsignedLong *str_size,
    const struct DDS_PrintFormatProperty *property)
```

RTI Code Generator

- **Standard Compliance:**
  - Support for prefix syntax to apply built-in annotations

```
struct MyType {
    @key long keyMember;
}
```

  - Support for new built-in annotations: autoid, hashid, external (previously '*'), nested (previously top-level), value (previously '=' for enumerator values), appendable, mutable, final
  - Parsing of custom annotations, ignore them
  - Negative values in enums
  - Support for empty structures

Other products

- Web Integration Service:
  - Promoted to GAR
  - Integration with Admin Console
- Database Integration Service:
  - Support for PostgreSQL (data subscription only)
  - Support for JSON storage in PostgreSQL and MySQL
  - Storage of Source/Destination Timestamp

Other products

- Connector:
  - Promoted to Experimental

Thank you