

Technical Deep Dive: Product Advancements for Scaling And Securing IIoT Systems

Fernando Crespo Sanchez

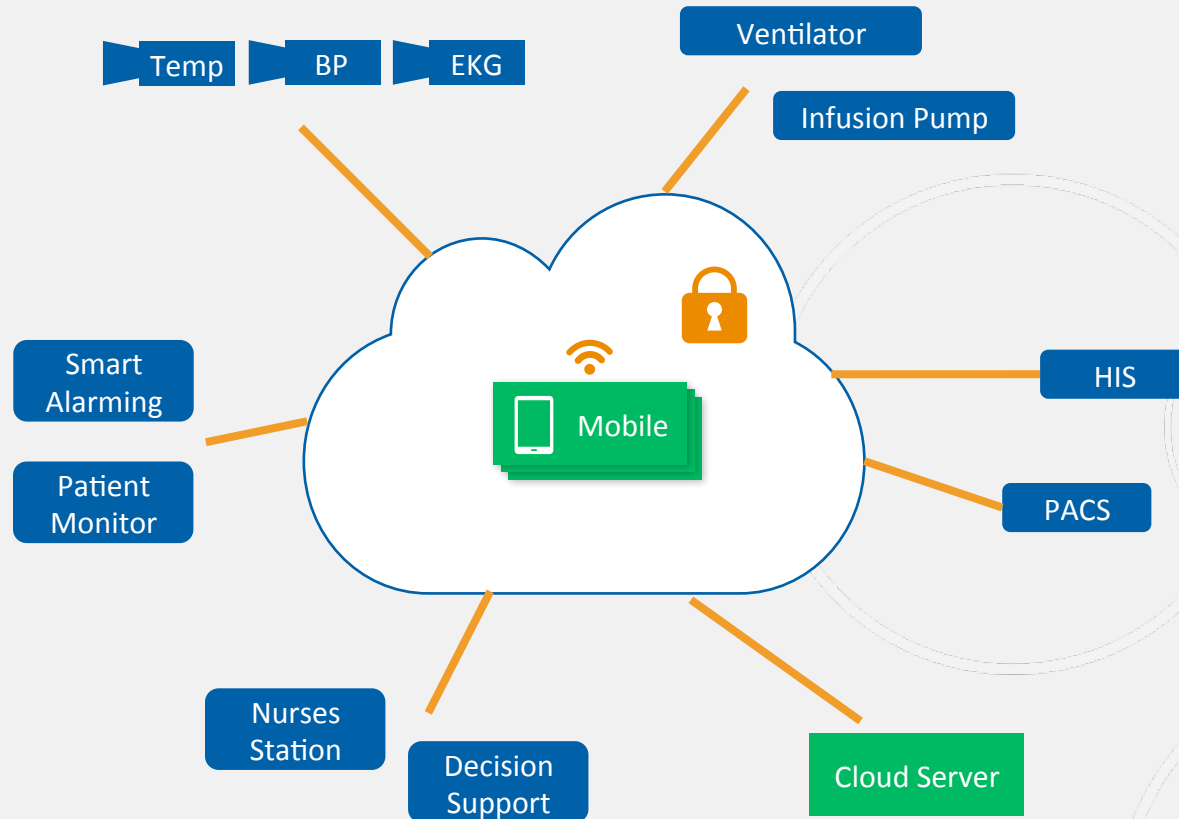
Product Architect, Real-Time Innovations



Outline

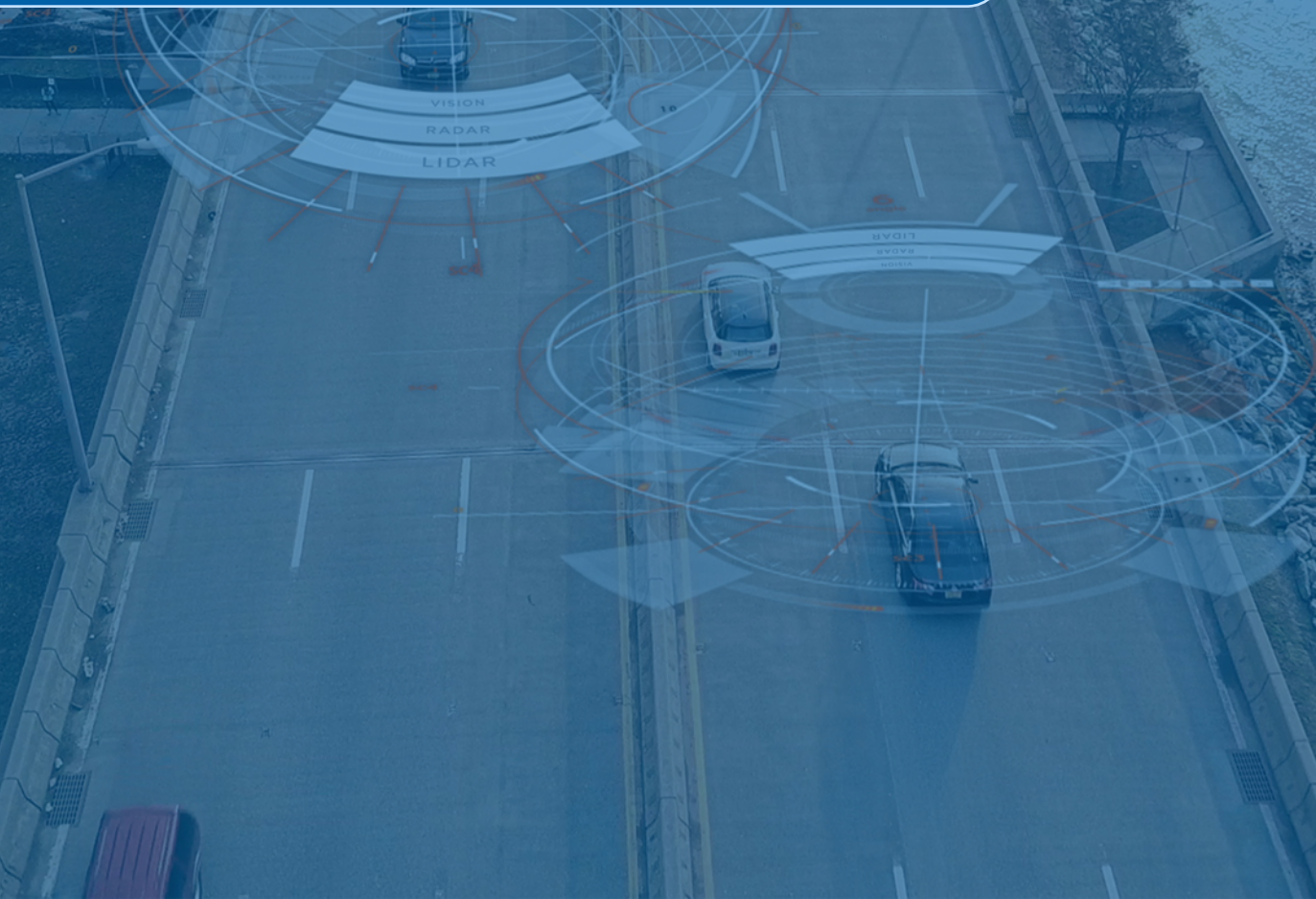
- IIoT Systems Characteristics
- Connex Platform
- Core Connectivity Platform Support
- Performance and Scalability
- Security
- Type System & Evolvability
- In the Works

IIoT System Example: Patient Monitoring



- Large Systems
 - Thousands of endpoints
 - Hundreds of users
- Heterogeneous software/hardware platforms
 - OS, networking, middleware
- Different kinds of data
 - Large and small data
 - Historical and real-time data
 - Confidential and non-confidential
 - Periodic and non periodic
- Interoperability across application versions
- Mobility

Connex Platform



Connex 6: Platform for Distributed System Connectivity



Connex DDS Professional

Connectivity software for developing and integrating IIoT systems.



Connex DDS Secure

Designed for systems requiring robust, fine-grained security.



Code
Generation



Data
Routing



Data
Persistence



Data
Queuing



Recording
& Replay



System
Administration



System
Introspection



System
Monitoring



Database
Integration



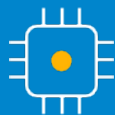
Web
Integration



Spreadsheet
Integration



3rd Party
Integrations



Connex DDS Micro

Designed for resource-constrained systems.



Connex DDS Cert

Designed for safety-certifiable systems.

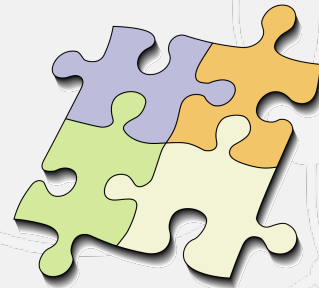
Connex DDS Professional and Micro Alignment



Connex DDS Professional and Micro Alignment

- Wire interoperable
- Type System compatible
- DDS standard API compatible
- Configuration compatible

Connex DDS
Professional



Connex DDS
Micro

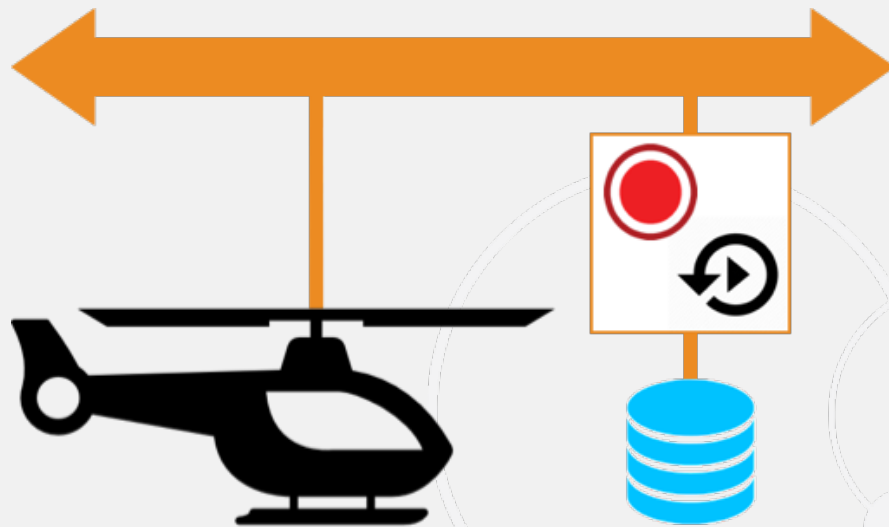
Connex DDS Micro 3 New Features

- XML Application Creation
- Large Data (RTPS fragmentation)
- Batching on the DataReader side
- X-Types 1.2 (not including TypeObject)
- Shared Memory Transport
- RTI FlatData TM
- Zero Copy Transfer over Shared Memory
- DDS Security

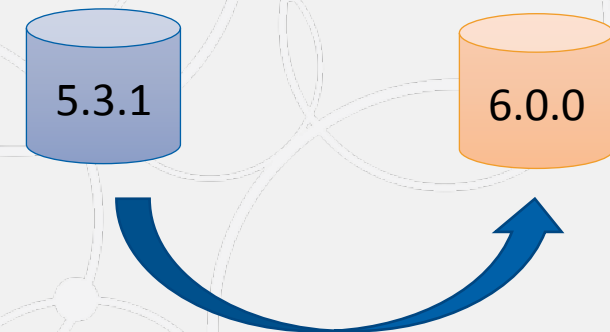
Connex 6 Recording Service



Improved Data Recording



RTI provides tools and documentation to support the **migration** to new Recording Service

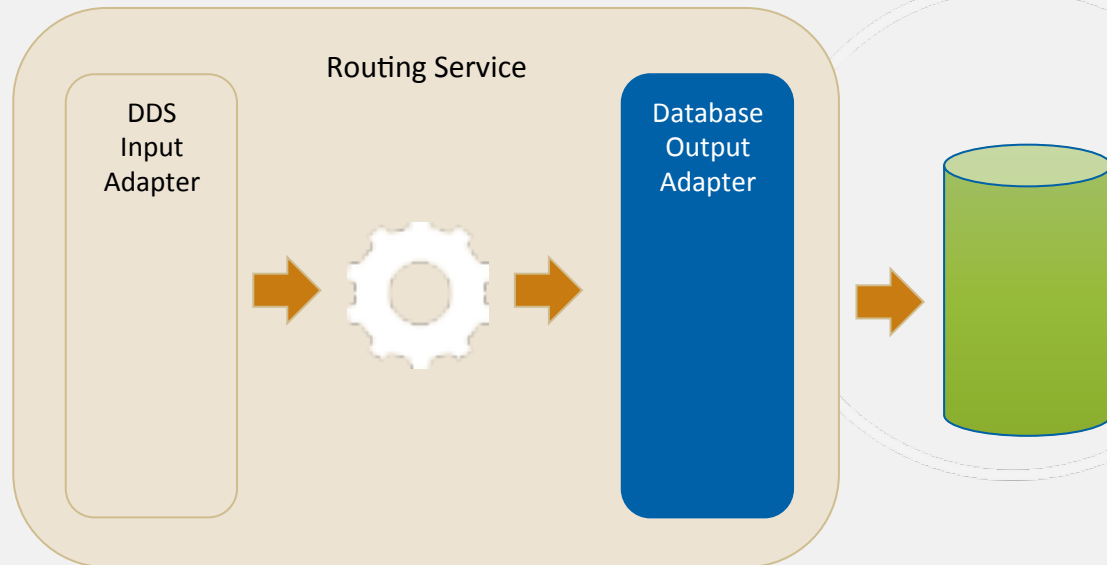


Connex 6 Recording Service improves **performance, scalability, simplifies configuration** and adds support for **custom storage plugins**

Record/Replay – Based on Connex Routing Service

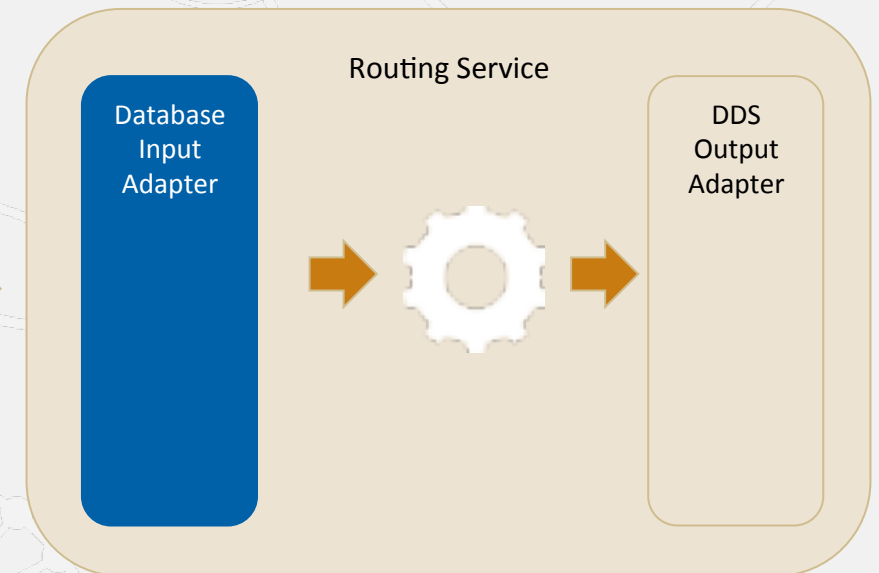
Recorder

Recording Service XML Configuration

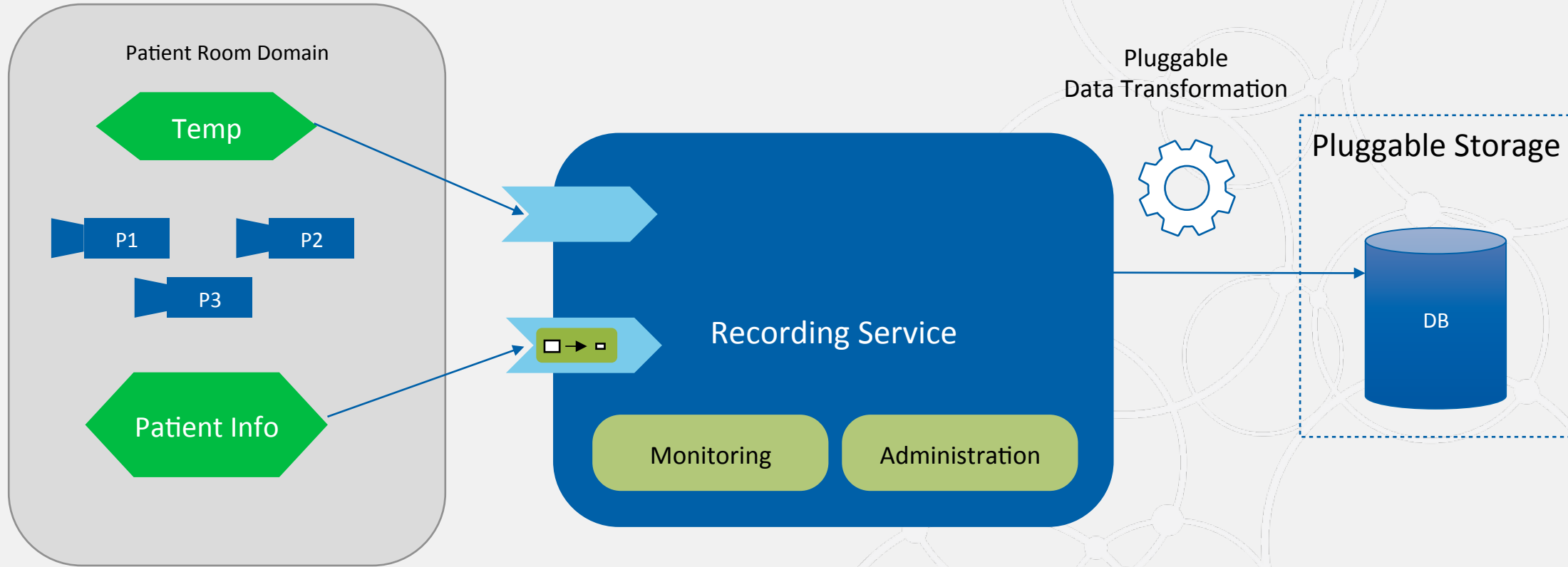


Replay

Replay Service XML Configuration

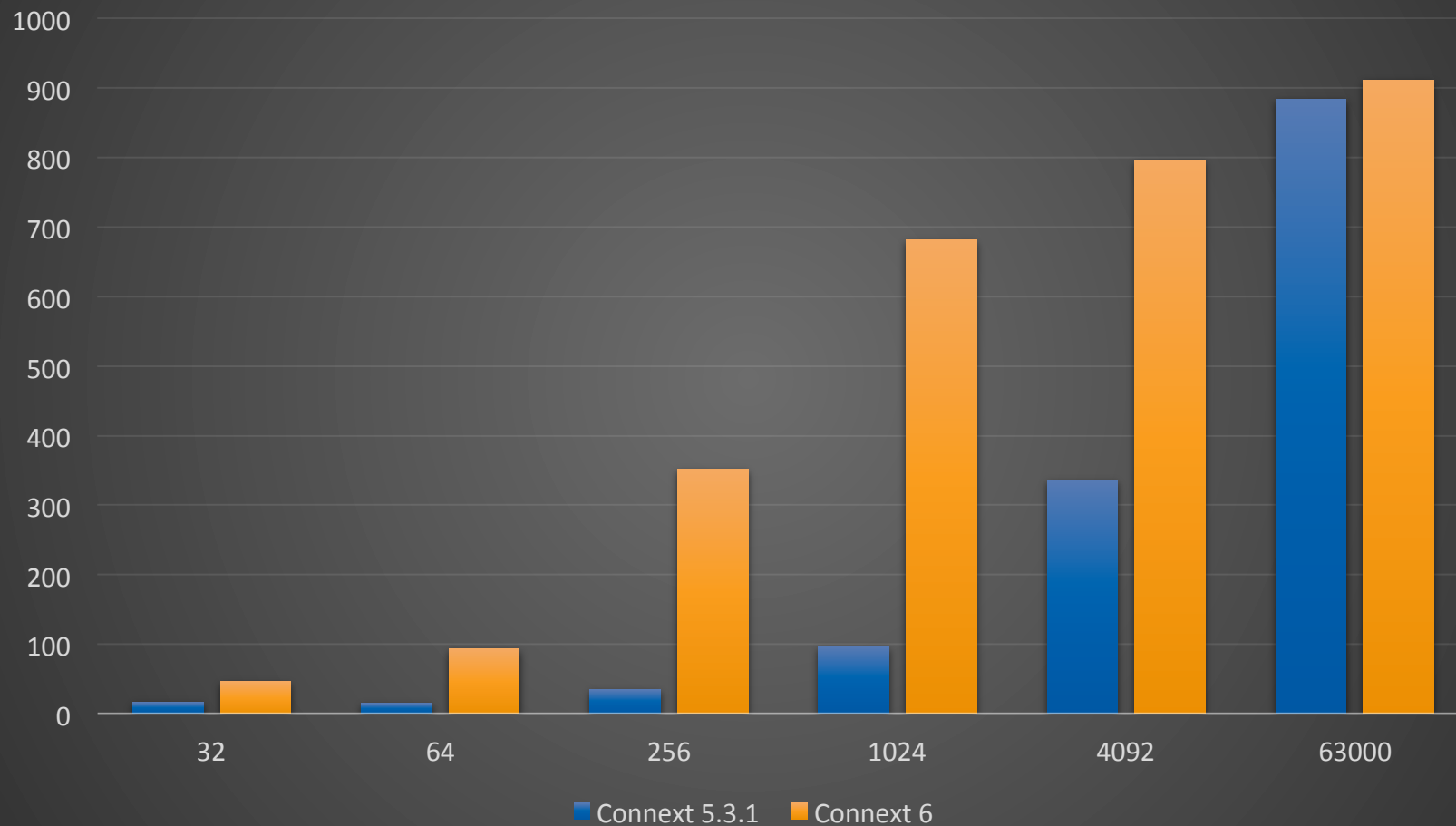


Connex 6 Recording Service: New Functionality



Performance

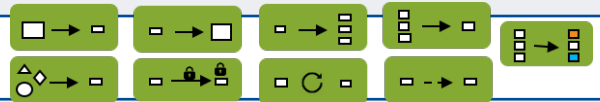
Throughput (Mbps) UDPv4 Reliable Keyed



Platform

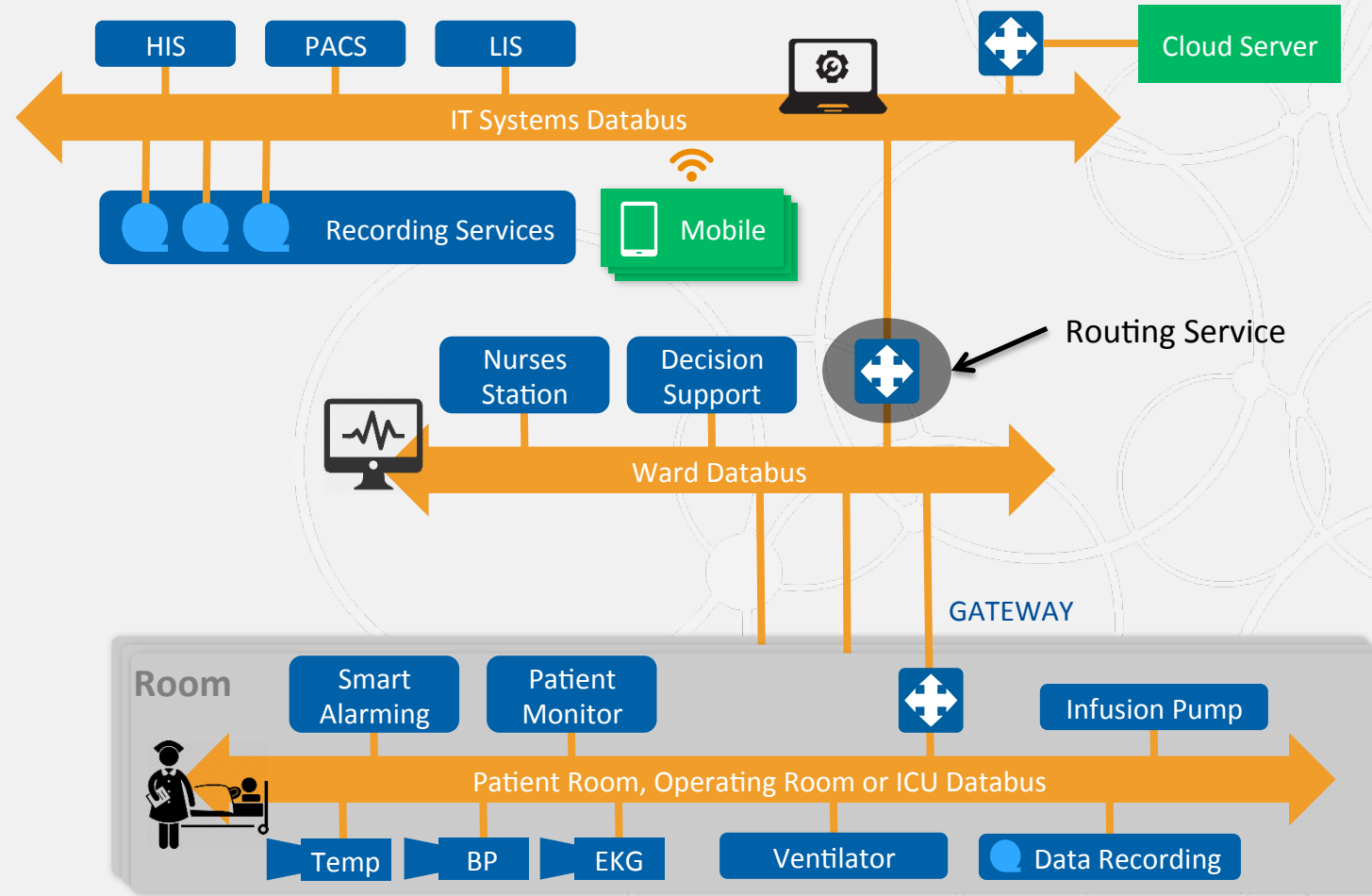
Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux
1 Gb network

Connex 6 Recording Service - Summary

	Connex 5.3 Recording Service	Connex 6 Recording Service
User Interface	Recording Console	Admin Console (some limitations)
Content-Filtered Topics	No	Yes
File Rollover	Yes	Yes
Logging	Yes	Yes, with additional debugging aids
Use recorded QoS when replaying	Yes, but Partition QoS only	No
Deserialized data format	SQLite default format (every field in a column)	JSON (with support for arbitrary types)
Tagging time ranges	No	Yes
Remote Shell	Yes	No (supports shell in an example)
Security (Encryption)	Yes	Yes
Enterprise Integration Patterns	Limited	Yes 
Pluggable Storage	No	Yes

Core Connectivity Platform Support

Layered Architecture

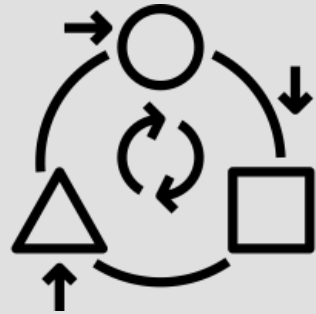


Connex 6 Routing Service

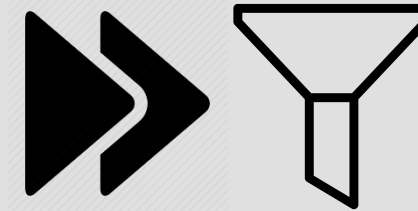


Routing Service: Key Capabilities

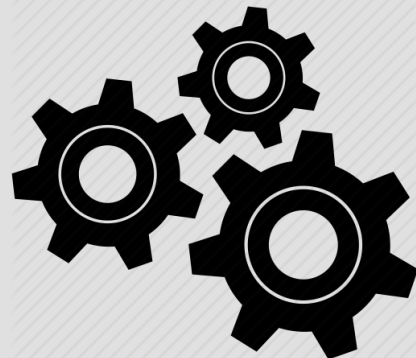
Data Transformation



Scalable Data Forwarding (content filter propagation)



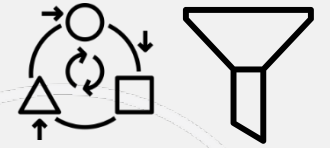
Technology integration (Adapters)



Monitoring and Administration

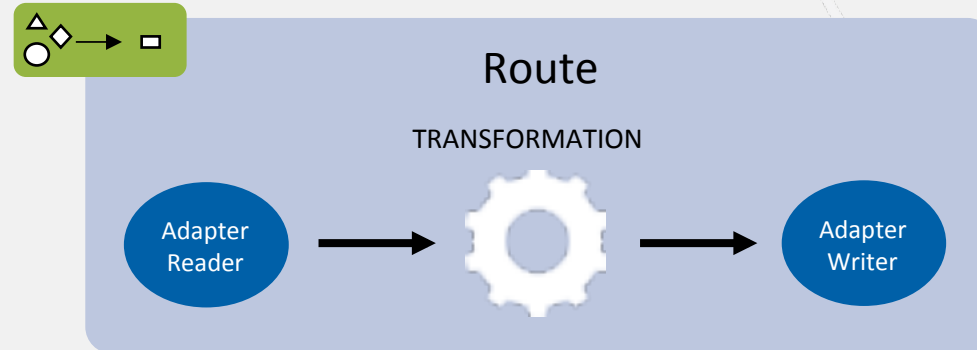


Data Transformation & Filtering



```
struct PatientVitals_EU {  
    long ID  
    string name;  
    long temperatureCelsius;  
    long pulseRate;  
};
```

Data Transformation



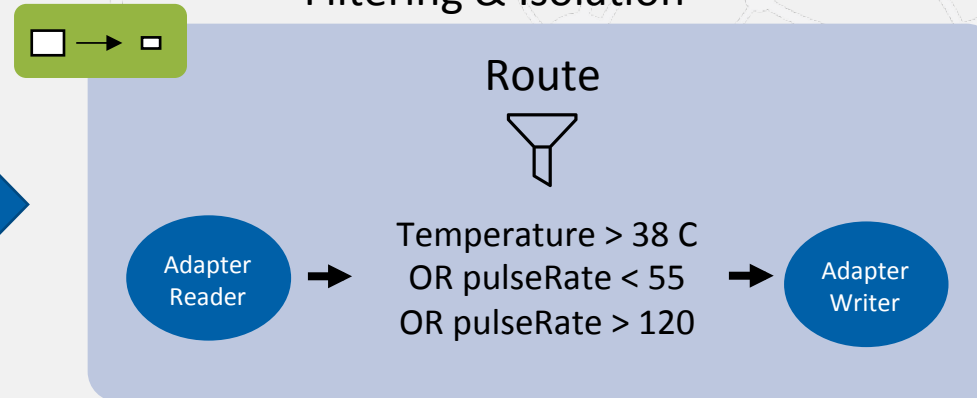
```
struct PatientVitals_US {  
    long ID  
    string name;  
    long temperatureFarenheit;  
    long pulseRate;  
};
```

Domain: Patients

```
ID=X12345  
name=Rose;  
temp=37.5;  
pulseRate=66;
```

```
ID=X12345  
name=Paul;  
temp =37.5;  
pulseRate=50;
```

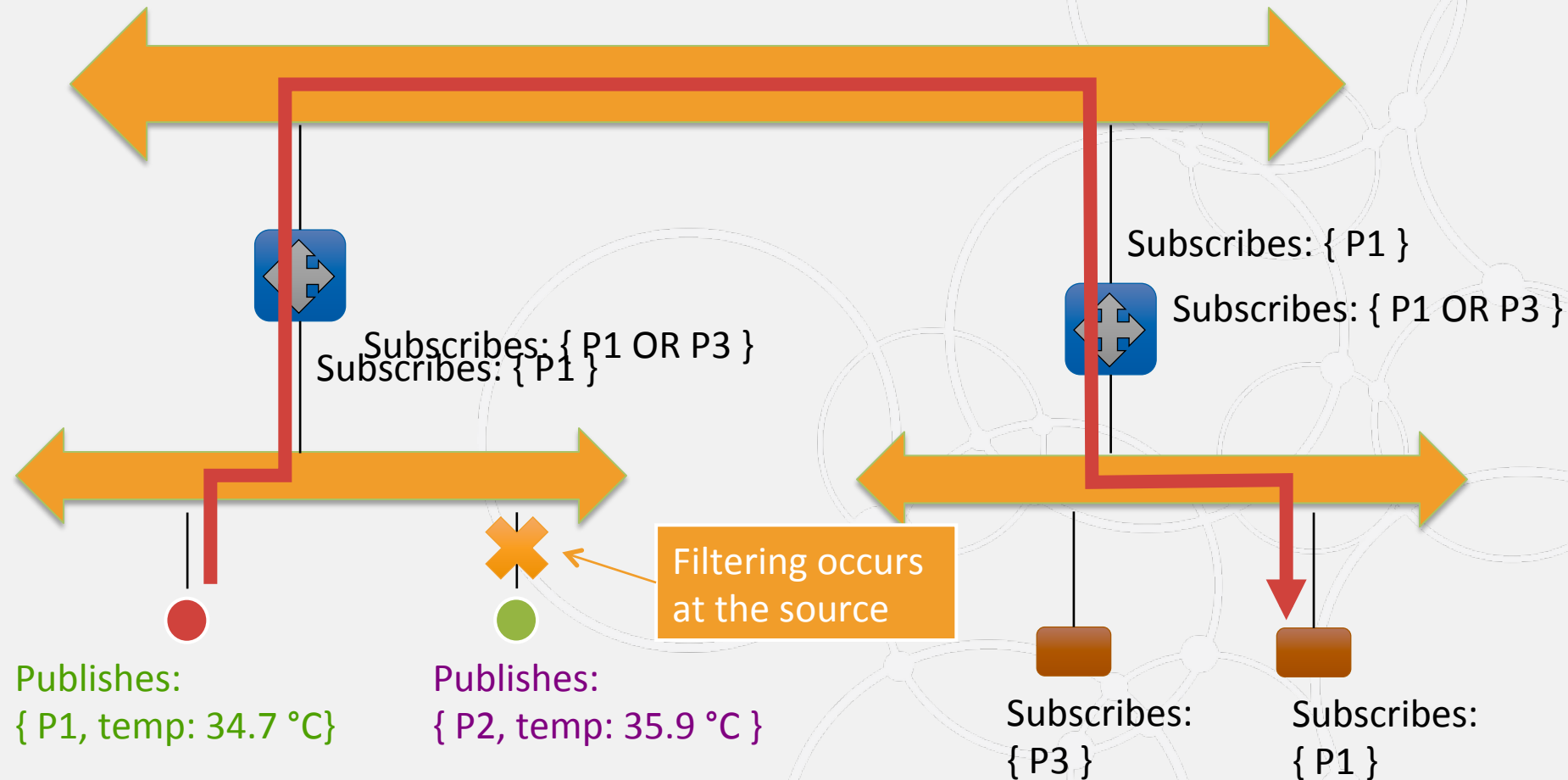
Filtering & Isolation



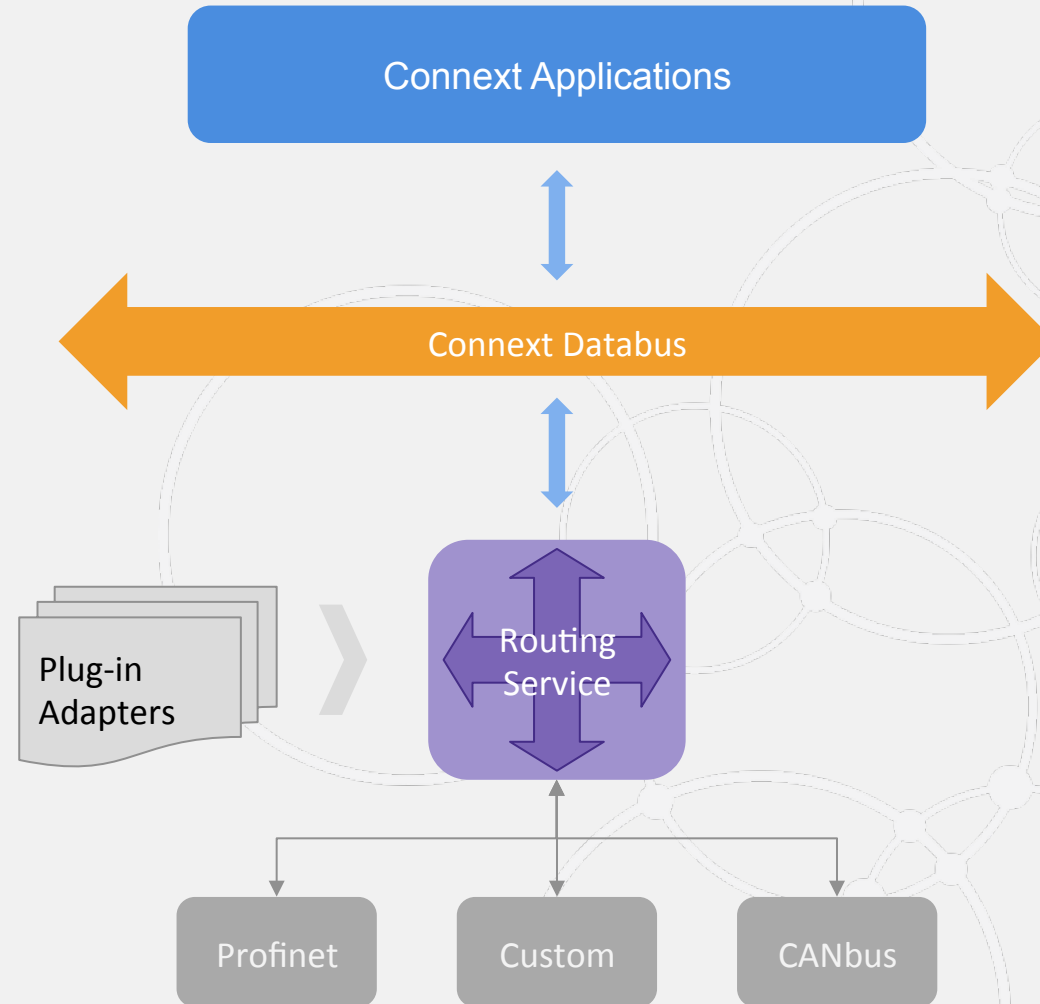
Domain: Nursery

```
ID=X12345  
name=Paul;  
temp =37.5;  
pulseRate=50;
```

Scalable Data Forwarding: CFT Propagation



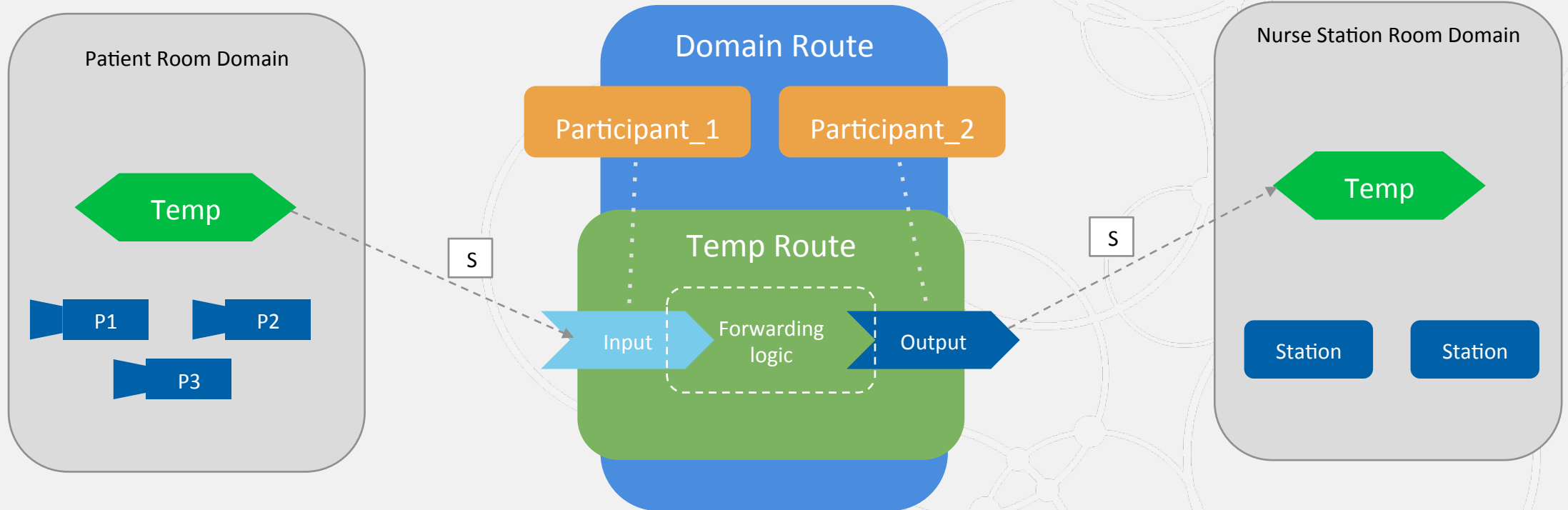
Routing Service Adapters



Connex 6 Routing Service

- Adds new capabilities for **system integration** and **scalability**
 - Routes with **multiple** forwarding paths
 - **Stateful** Adapter and Processor APIs
 - Scalable **Monitoring** and **Administration**
- Backwards compatible with previous generation

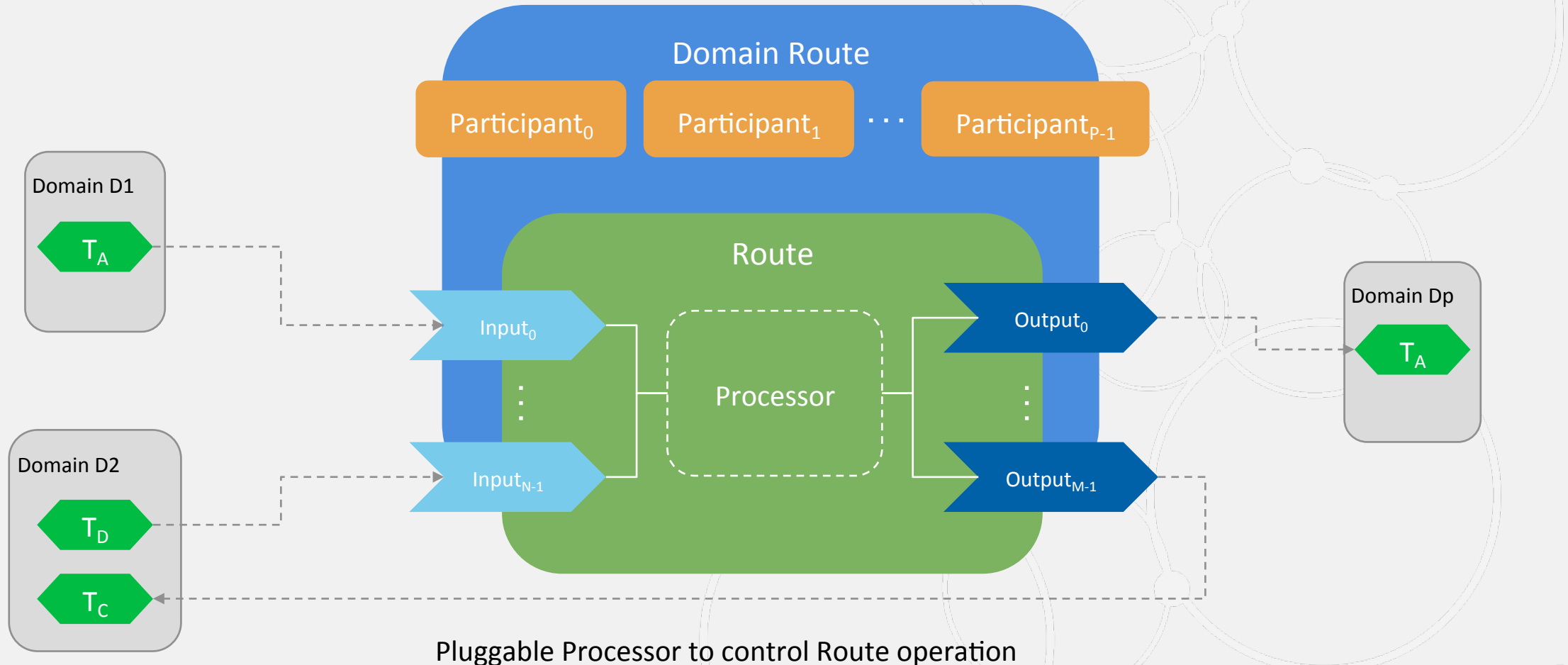
Architecture Basics



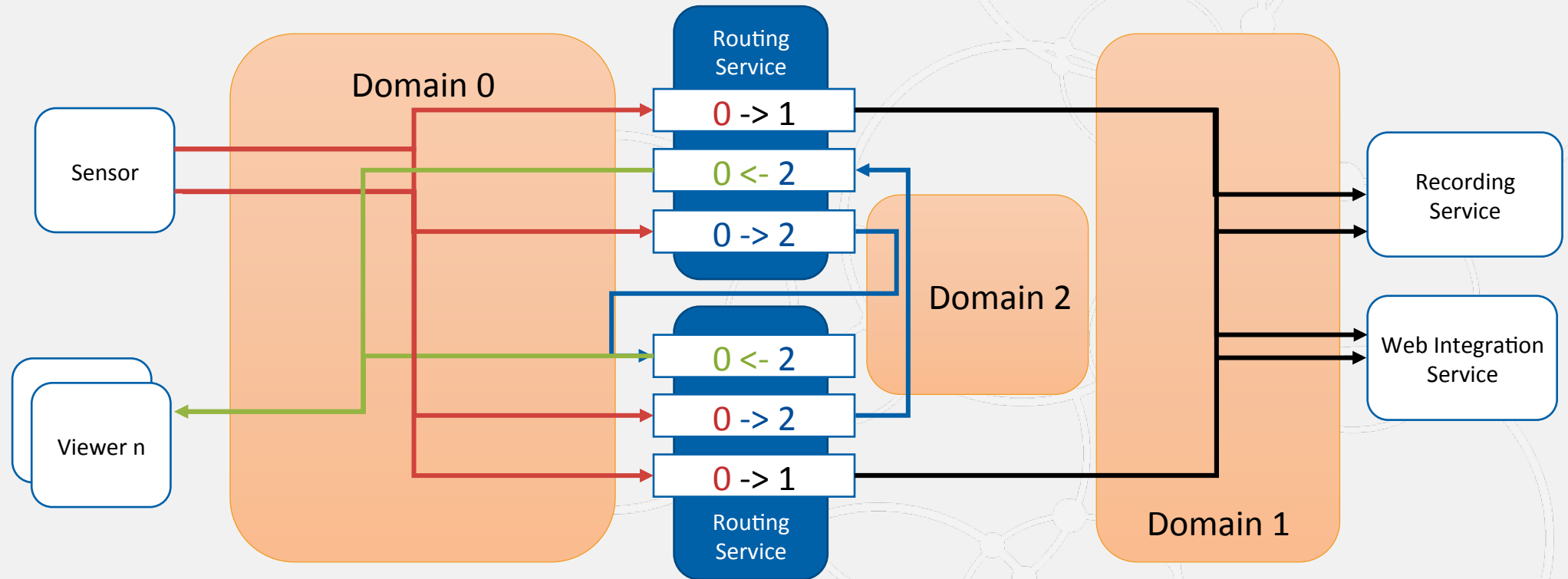
Connex 6 Routing Service: General Architecture

Multiple participants in a DomainRoute

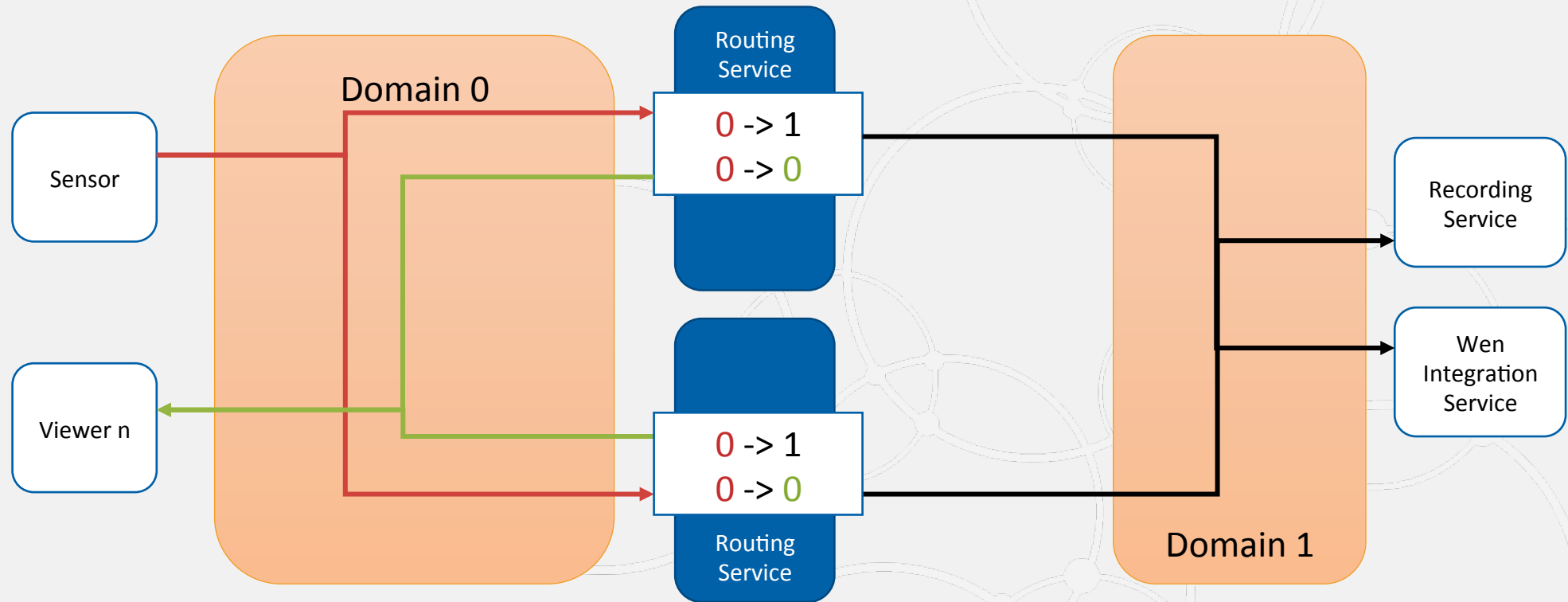
Multiple inputs and outputs in a Route



Previous Routing Service Solution: Complexity & Overhead



Connex 6 Routing Service: Simple and High Performance



Network Load – Reduced from 3 sample duplicates to 1

Router Memory - ~50% reduction

Discovery Time – ~50% reduction

SDK Updates: Processor and Adapter APIs

5.3 RS

C Support

- Callback transformation
 - transform(input, output)
- Executes based on
 - Arrival of data



Connex 6 RS

C/C++
Support

- Gets notified of **events**
 - Data on inputs
 - Events (periodic, input/output enable, ...)
- **Flexible access** to inputs/outputs
 - Read/Take/Select from Inputs
 - Write to Outputs
- **Enhanced execution** control
 - Arrival of data
 - Specified rate
 - Route events
- Production **decoupled** from consumption

Extended Support for Integration Patterns



Splitting



Aggregation



Content enrichment



Content filtering



Normalizer



Periodic action



Delayed action

...

Connex 6 Routing Service: Other features

- Enhanced Monitoring and Administration architecture
 - Provides better scalability, flexibility and usability
- Improved logging
- Support for RTI FlatData [™] and Zero Copy Transfer over Shared Memory

Routing Service Integration with Admin Console



Routing Entities DDS Entities Log Routing Information Resource Charts

Metrics

bridge_domain_0_and_domain_1

TwoWayDomainRoute

Session1

AllForward

AllForward@Square

Input1

Processor

Output1

Session2

AllBackward

AllBackward@Circle

Input1

Processor

Output1

Statistics Configuration

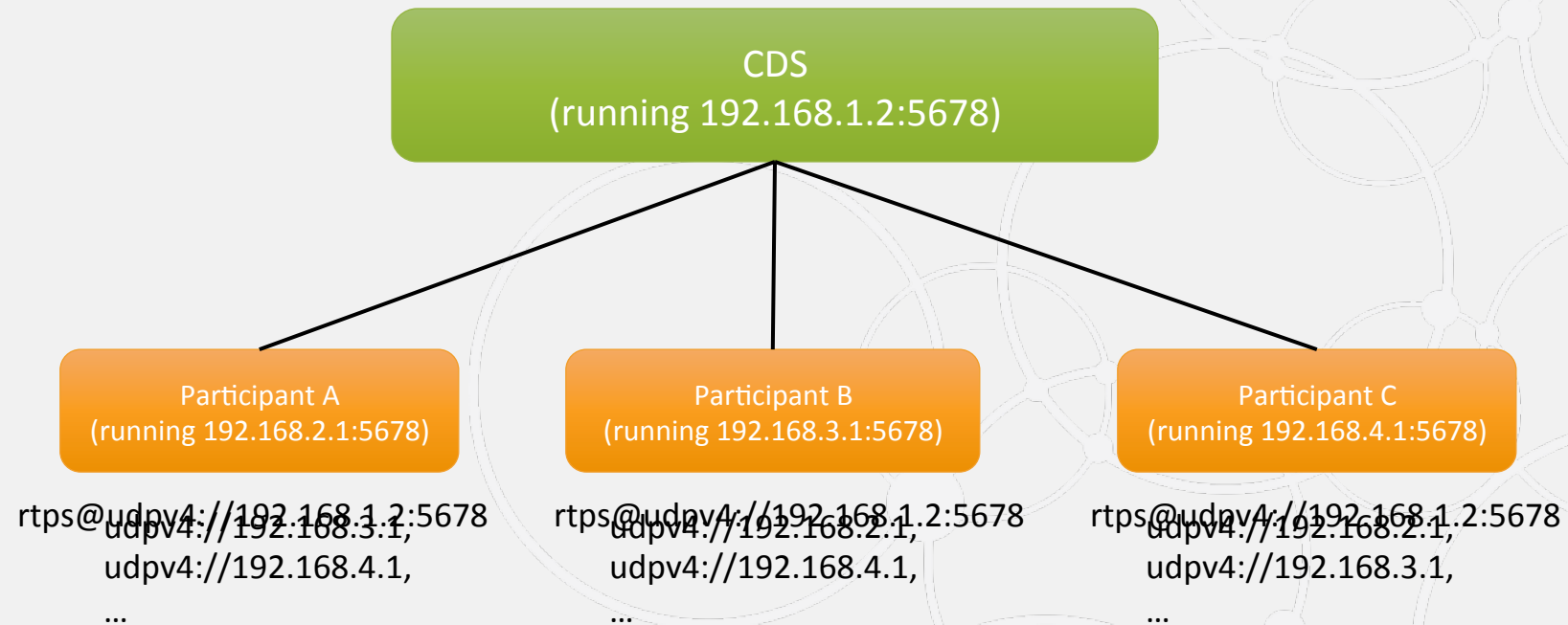
```
<auto_topic_route name="AllForward">
  <publish_with_original_info>true</publish_with_original_inf
</auto_topic_route>
```

Name	State
[RTI Routing Service] bridge_domain_0_and_domain_1	Started
[Domain Route] TwoWayDomainRoute	Started
[Connection] 1	N/A
[Connection] 2	N/A
[Session] Session1	Started
[Auto Route] AllForward	Started
[Route] AllForward@Square	Running
[Route Input] Input1	Enabled
[Processor] Processor	N/A
[Route Output] Output1	Enabled
[Session] Session2	Started
[Auto Route] AllBackward	Started
[Route] AllBackward@Circle	Running
[Route Input] Input1	Enabled
[Processor] Processor	N/A
[Route Output] Output1	Enabled

Cloud Discovery Service and Domain Tags

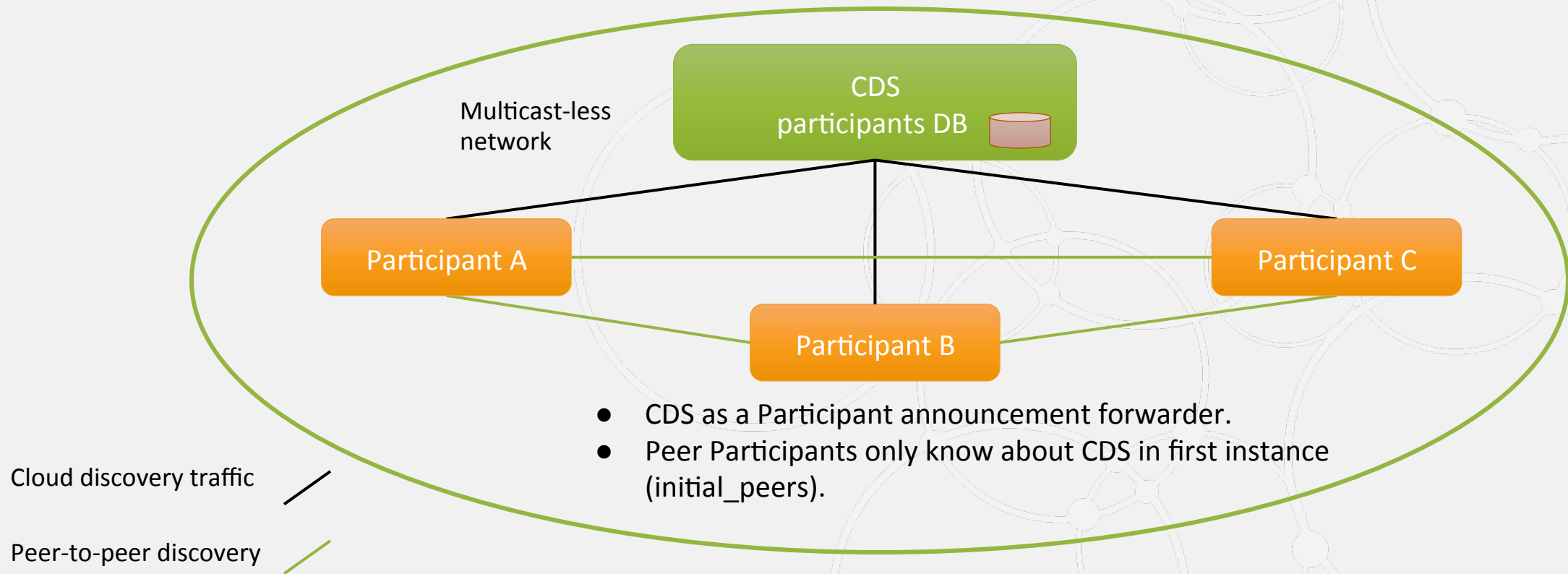


Initial Peers In Networks Without Multicast

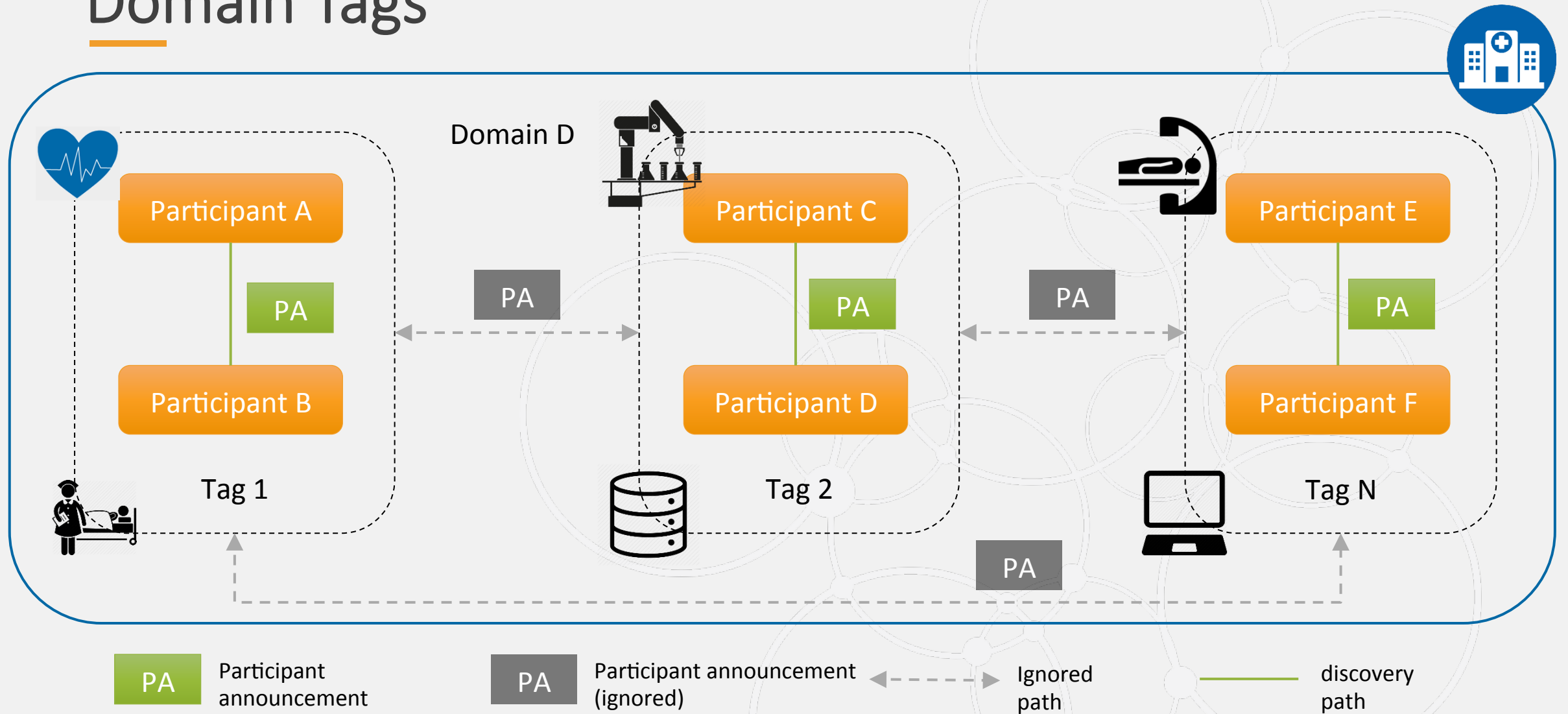


What is Cloud Discovery Service?

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



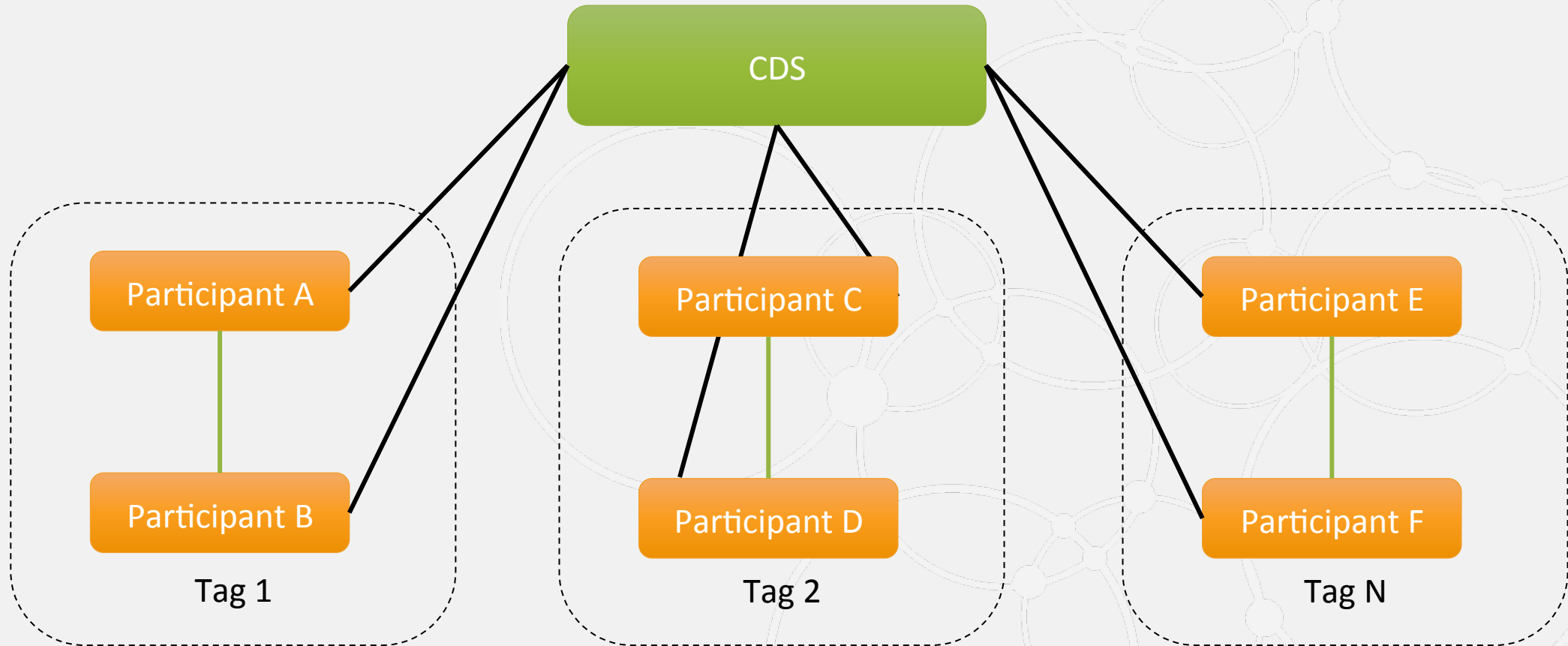
Domain Tags



- Configurable through DomainParticipant PropertyQos: **dds.domain_participant.domain_tag**

Domain Tags

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



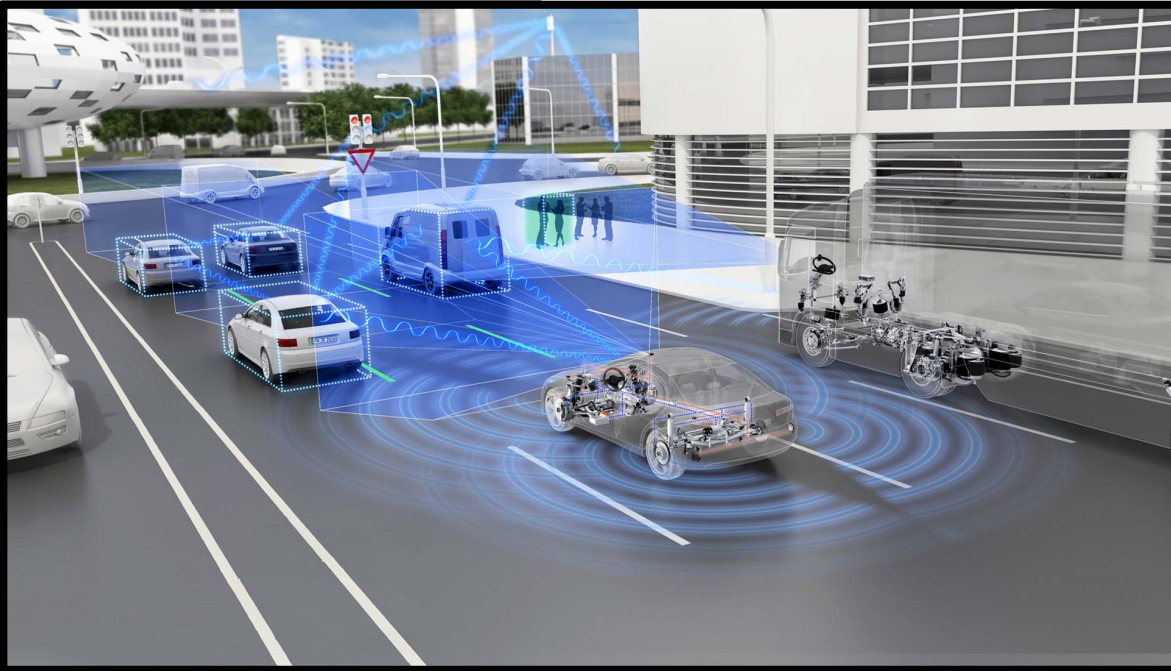
Performance And Scalability

Large Data Streaming Performance Improvements



Large Data Streaming Use Cases

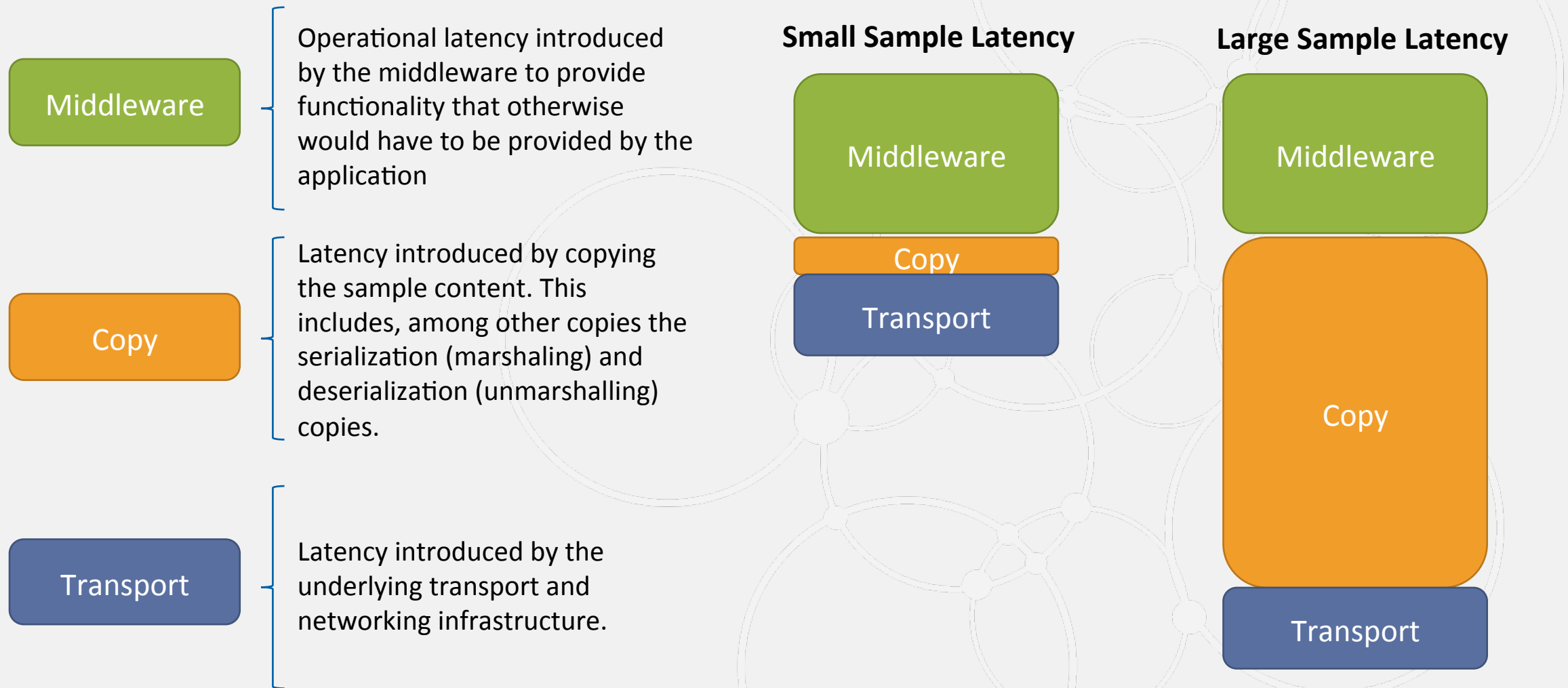
Autonomous Driving



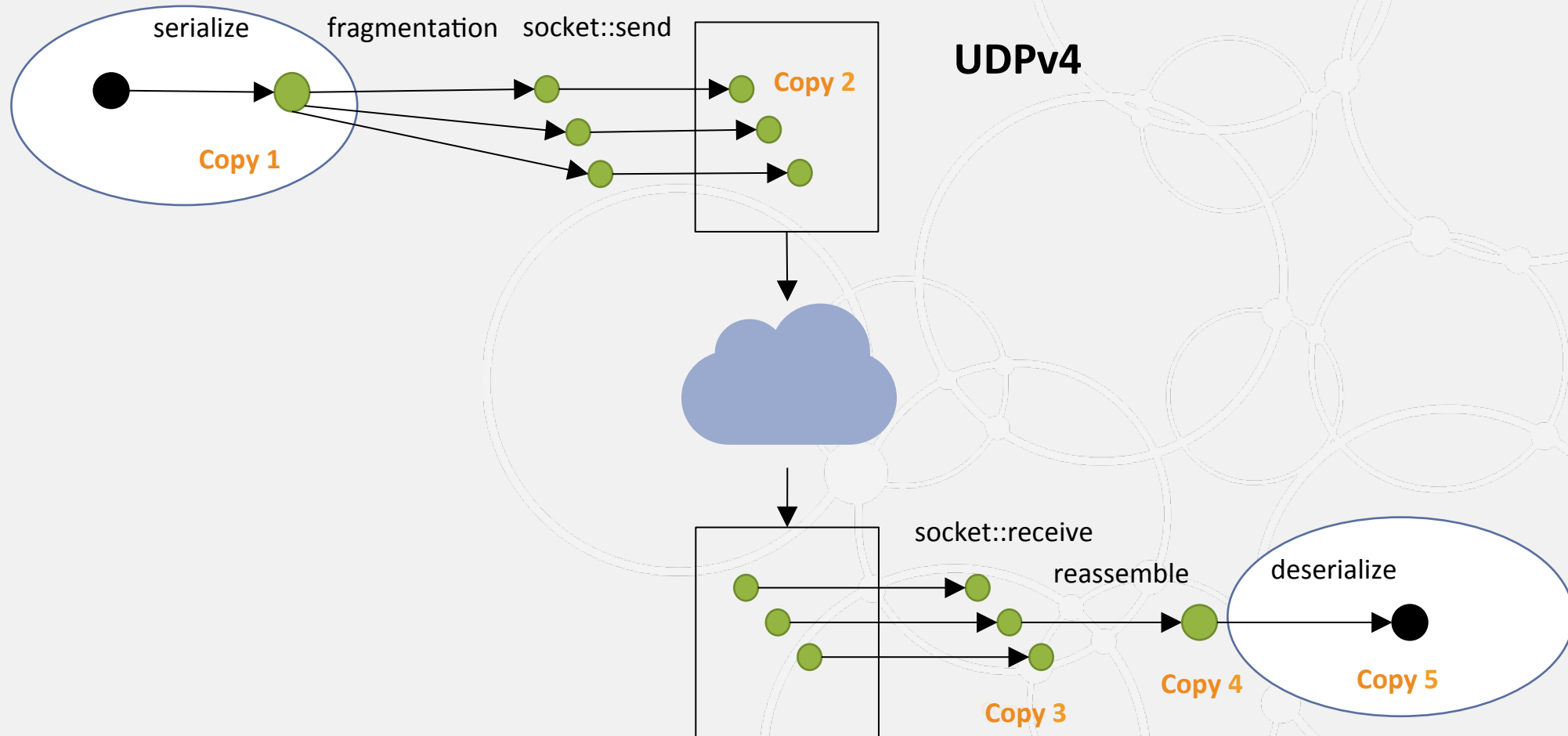
Medical Imaging



Communication Latency Components

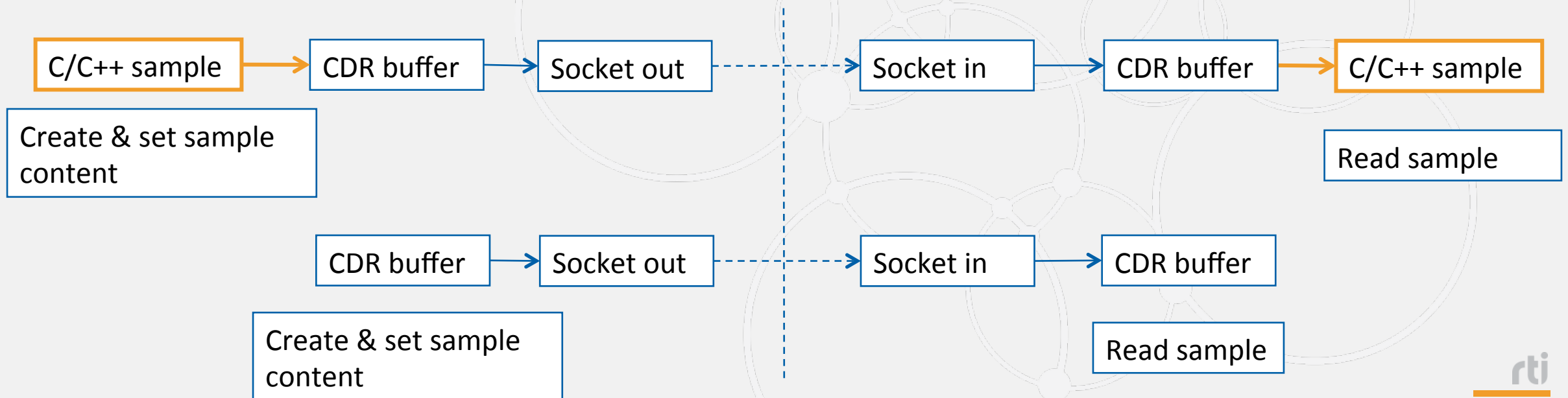


Large Data Streaming Problem: Copies

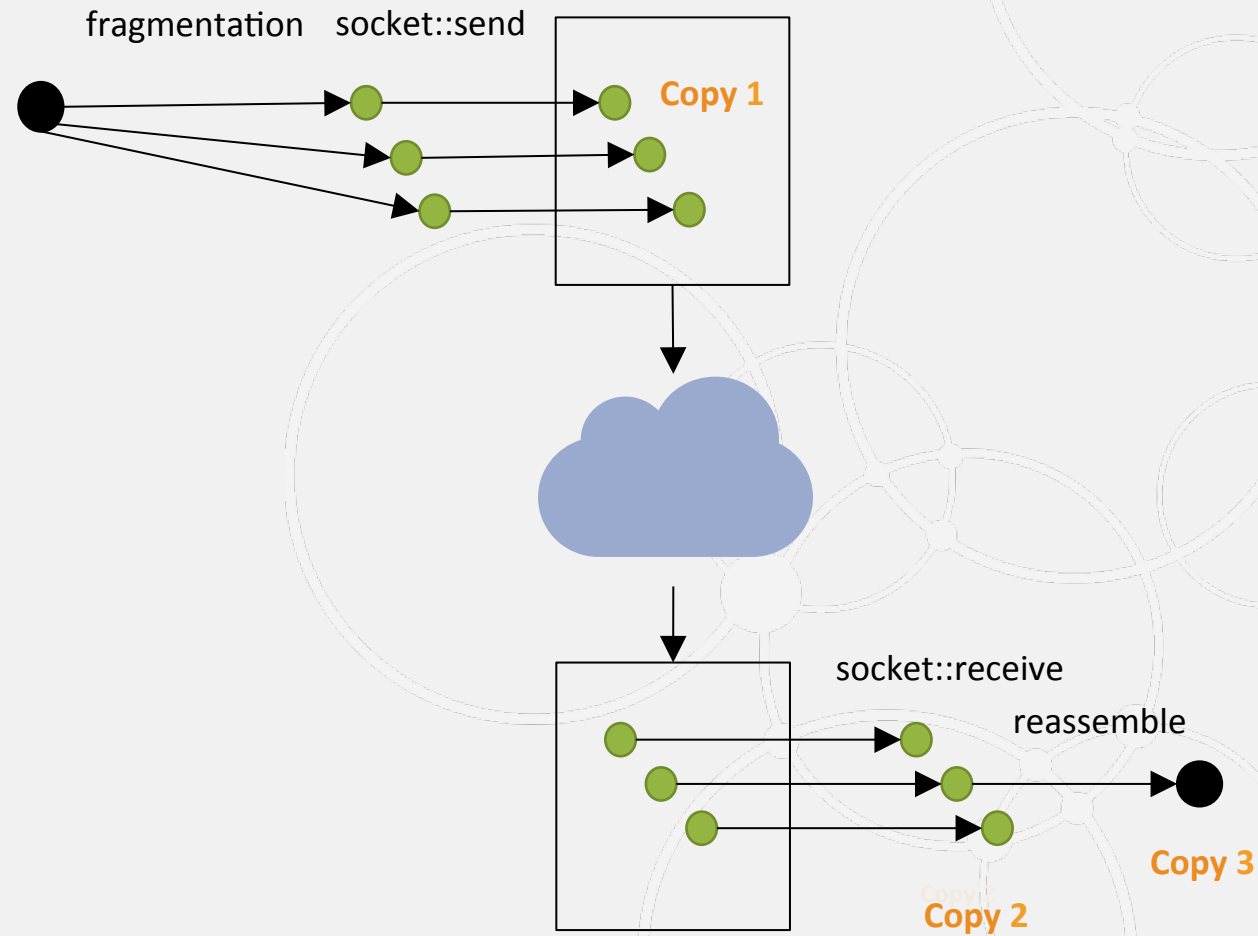


What is FlatData?

- Language Binding in which Wire Representation is equal to Memory Representation
 - Memory representation = Wire representation CDR v2
- Serialization and deserialization cost is zero



FlatData over UDPv4



How to use FlatData

- Use an annotation in IDL
 - Annotate types requiring FlatData language binding with `@language_binding()` in IDL
 - Special type plugin code generation
- No additional library needed

```
@language_binding(FLAT_DATA)
@final
struct Dimension {
    uint16 width;
    uint16 height;
};

@language_binding(FLAT_DATA)
@final
struct Image {
    int64 timestamp;
    Dimension dimension;
    uint8 pixels[30000000];
};
```

How to use FlatData

Publication Example (Modern C++)

```
// Get an image
Image *data = writer.extensions().get_loan();

// Update timestamp and dimension
auto image = data->root();
image.timestamp(3000);

auto dimension = image.dimension();
dimension.width(100);
dimension.height(200);

// Get pointer to image buffer and populate it
auto pixels= image.pixels();
uint8_t *pixels_ptr = rti::flat::plain_cast(pixels);
populate_image_pixels(pixels_ptr);

// Write image
writer.write(*data);
```

```
@language_binding(FLAT_DATA)
@final
struct Dimension {
    uint16 width;
    uint16 height;
};

@language_binding(FLAT_DATA)
@final
struct Image {
    int64 timestamp;
    Dimension dimension;
    uint8 pixels[30000000];
};
```

How to use FlatData

Subscription Example (Modern C++)

```
// Take
auto samples = rti::sub::valid_data(reader.take());

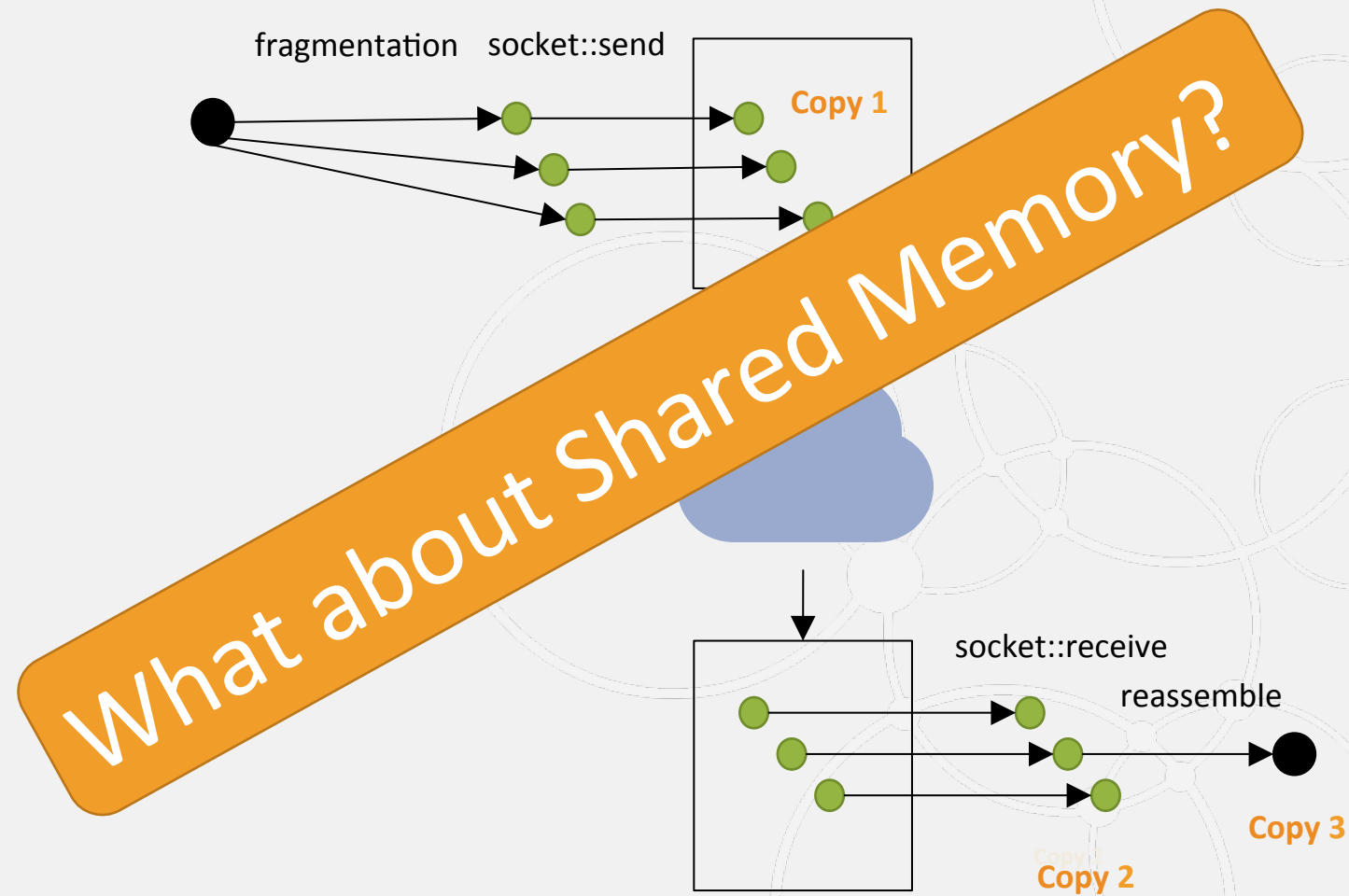
for (const auto& sample : samples) {
    // Copy timestamp, width, and height
    auto image = sample.data().root();
    int64_t ts = image.timestamp();
    uint16_t width = image.dimension().width();
    uint16_t height = image.dimension().height();

    // Get a pointer to the image buffer
    const uint8_t *pixels_ptr =
        rti::flat::plain_cast(image.pixels());
}
```

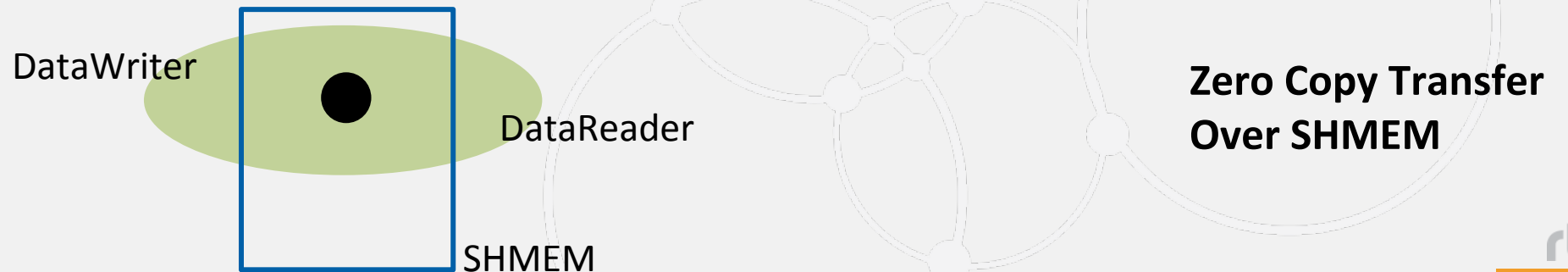
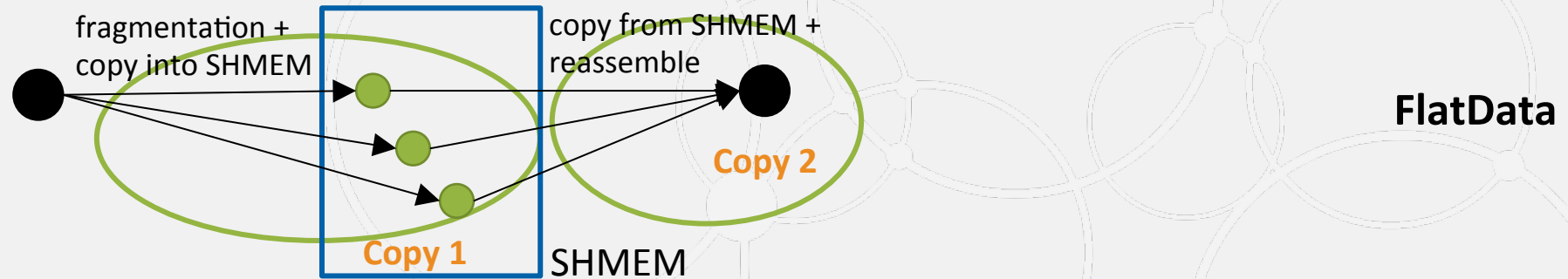
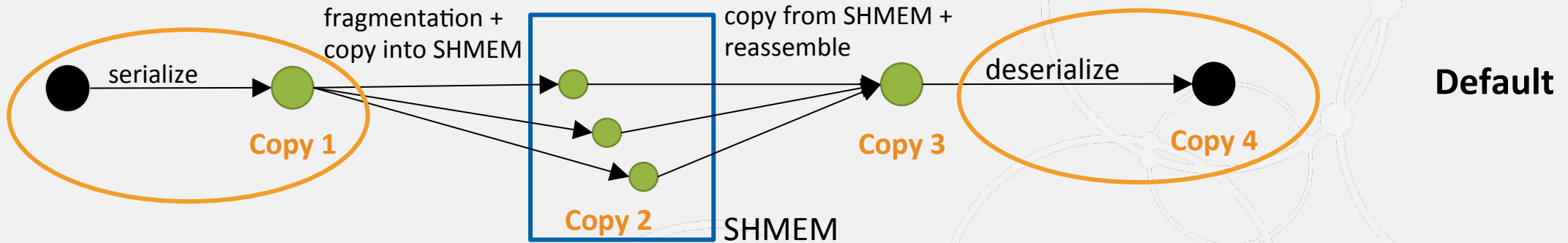
```
@language_binding(FLAT_DATA)
@final
struct Dimension {
    uint16 width;
    uint16 height;
};

@language_binding(FLAT_DATA)
@final
struct Image {
    int64 timestamp;
    Dimension dimension;
    uint8 pixels[30000000];
};
```

FlatData over UDPv4



Saving Copies when Using SHMEM



How to use Zero Copy SHMEM Transfer

- Use an Annotation in IDL
 - Annotate types with `@transfer_mode(SHMEM_REF)` in IDL
 - Only top level types need the annotation
 - Special type plugin code generation
 - Type validation
- Link the application with optimized SHMEM library

```
struct Dimension {  
    uint16 width;  
    uint16 height;  
};  
  
@transfer_mode(SHMEM_REF)  
struct Image {  
    int64 timestamp;  
    Dimension dimension;  
    uint8 pixels[30000000];  
};
```

How to use Zero Copy SHMEM Transfer

Publication Example (C++)

```
// Get data
Image *data= writer->get_loan();

// Update sample
data->timestamp = ts;
data->dimension.width = width;
data->dimension.height = height;
populate_pixels_data(data->pixels, buffer, size);

// Write sample
retcode = imageWriter->write(*data, instance_h);
```

```
struct Dimension {
    uint16 width;
    uint16 height;
};

@transfer_mode(SHMEM_REF)
struct Image {
    int64 timestamp;
    Dimension dimension;
    uint8 pixels[30000000];
};
```

How to use Zero Copy SHMEM Transfer

Subscription Example (C++)

```
// Take
retcode = imageReader->take(data_seq, info_seq,
    DDS_LENGTH_UNLIMITED, sample_state,
    view_state, instance_state);

for (i = 0; i < data_seq.length(); ++i) {
    if (info_seq[i].valid_data) {
        // Get and process sample
        image = data_seq[i];
        ...
        // Check sample consistency
        is_consistent = imageReader->is_sample_consistent(
            image, info_seq[i]);
    }
}
```

```
struct Dimension {
    uint16 width;
    uint16 height;
};

@transfer_mode(SHMEM_REF)
struct Image {
    int64 timestamp;
    Dimension dimension;
    uint8 pixels[30000000];
};
```

FlatData/Optimized SHMEM Summary

UDPv4

SHMEM

Default

(5 copies)

serialize + (send + fragment)
+ **receive**
+ reassemble + **deserialize**

(4 copies)

serialize + (send + fragment)
+ reassemble + **deserialize**

FlatData

@language_binding
(FLAT_DATA)

(3 copies)

(send + fragment)
+ **receive**
+ reassemble

(2 copies)

(send + fragment)
+ reassemble

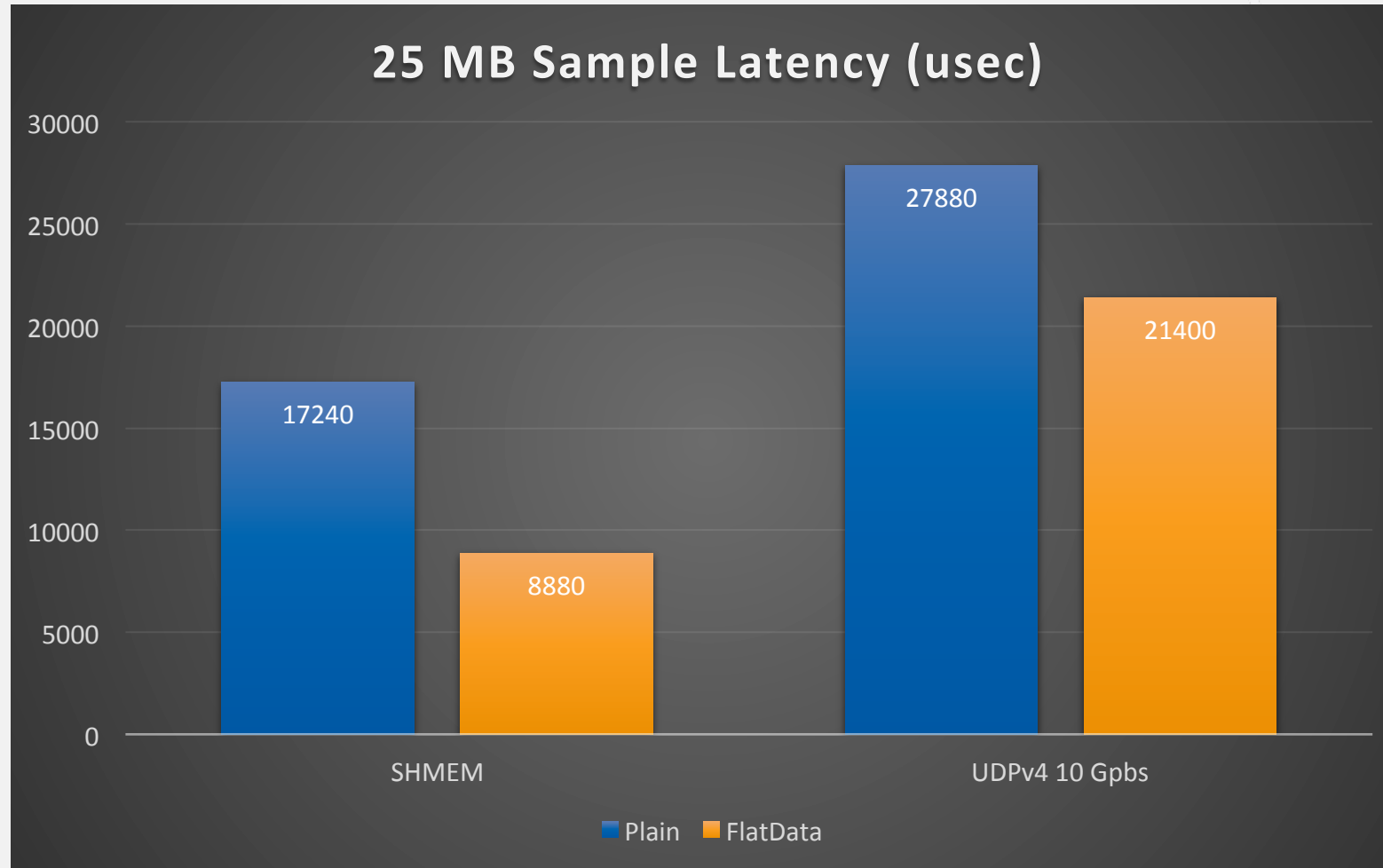
Zero Copy Transfer Over SHMEM

@transfer_mode
(SHMEM_REF)
@language_binding
(FLAT_DATA)

N/A

(0 copies)

FlatData Performance Across Nodes



Platform

Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux
10 Gb network

25 MB sample copy
time = 3370 usec

Large Data Performance Single Node



Platform

Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux

Optimized shared
memory latency
constant
independently of
the message size

Serialization/Deserialization Performance Improvements



Optimizing the Code Generation Process

- Optimization 0: No optimization
- Optimization 1: Resolves typedef to the most basic type

```
typedef double Temperature;  
typedef int32 PulseRate;  
typedef int32 RespirationRate;  
typedef int32 BloodPressure;  
  
@final  
struct VitalSigns {  
    Temperature temperature;  
    PulseRate pulse;  
    RespirationRate respiration;  
    BloodPressure diastolic_pressure;  
    BloodPressure systolic_pressure;  
};
```



```
@final  
struct VitalSigns {  
    double temperature;  
    int32 pulse;  
    int32 respiration;  
    int32 diastolic_pressure;  
    int32 systolic_pressure;  
};
```

Optimizing the Code Generation Process

- Optimization 2: More aggressive optimization techniques including inline expansion of nested types and serialization/deserialization of consecutive members with a single copy

```
@final
struct PixelRGB {
    int16 r;
    int16 g;
    int16 b;
};

@final
struct Image{
    int16 width;
    int16 height;
    PixelRGB data[786432];
};
```

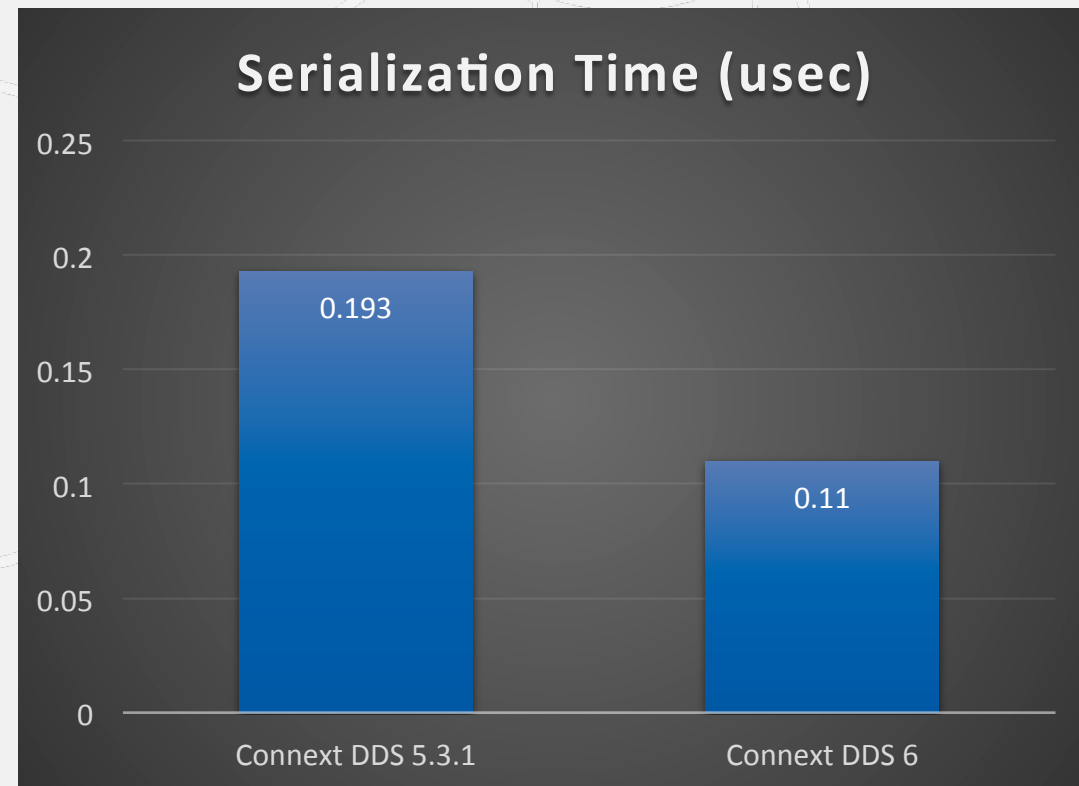


```
@final
struct Image{
    /* width and height are part of the array */
    int16 member[2359298];
};
```

Serialization/Deserialization Improvements

- Alias optimization (-optimization 1)

```
typedef double Temperature;  
typedef int32 PulseRate;  
typedef int32 RespirationRate;  
typedef int32 BloodPressure;  
  
@final  
struct VitalSigns {  
    Temperature temperature;  
    PulseRate pulse;  
    RespirationRate respiration;  
    BloodPressure diastolic_pressure;  
    BloodPressure systolic_pressure;  
};
```

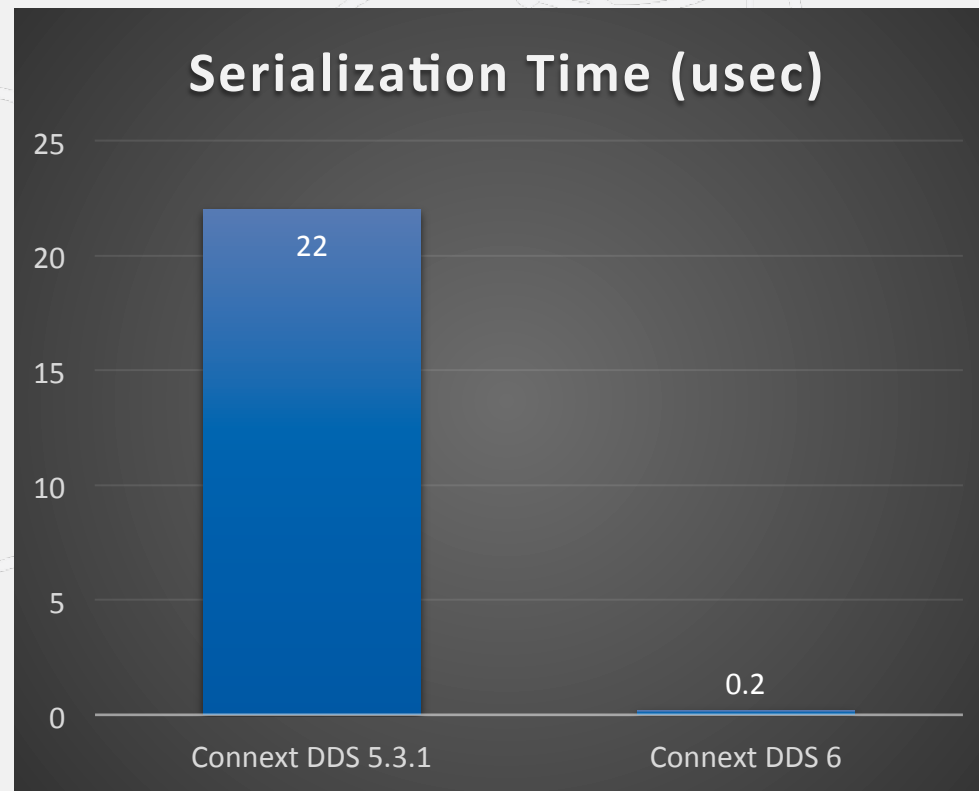


Serialization/Deserialization Improvements

- Inline expansion of nested types (-optimization 2)

```
@final
struct PixelRGB {
    int16 r;
    int16 g;
    int16 b;
};

@final
struct Image{
    PixelRGB data[786432];
};
```

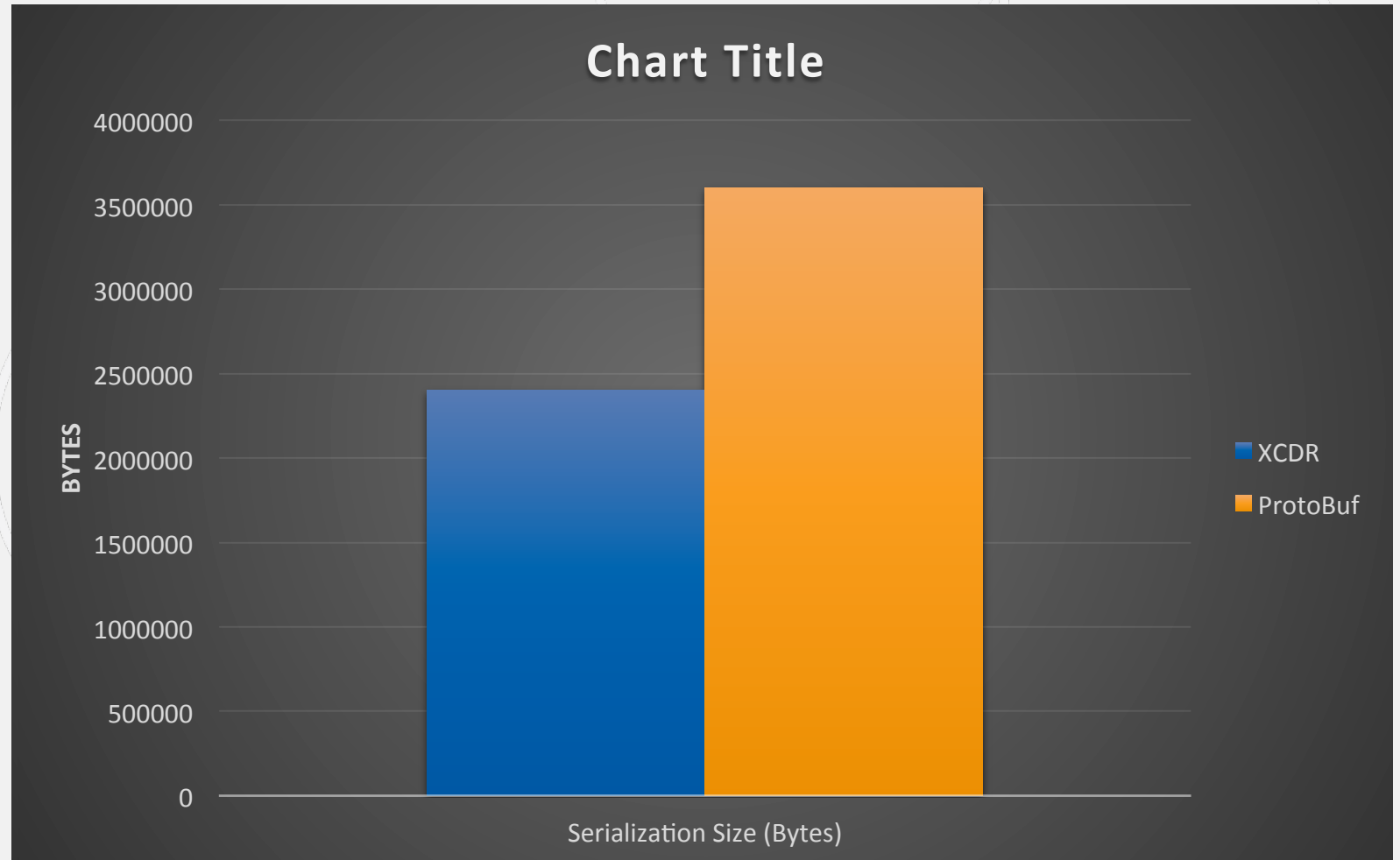


XCDR versus ProtoBuf (Serialization Size)

```
@final
struct Point
{
    float A;
    float B;
};

typedef sequence< Point,
maxSize> SequenceOfPoints;

@final
struct RadarSweep
{
    ...
    SequenceOfPoints samples;
};
```



DynamicData Performance Improvements



DynamicData Performance Improvements

- DynamicData implementation has been rewritten from scratch to make it more performant
 - Better in-memory representation for out-of-order member access
 - Uses same engine for serialization/deserialization than generated code

DynamicData Performance Improvements

- Any API that takes member name can use a hierarchical member name. No need to bind.

```
struct Bar {  
    long member_long;  
};  
  
struct Foo {  
    Bar member_bar;  
};  
  
long longVal = foo_dyndata.get_long(  
    "member_bar.member_long",  
    DDS_DYNAMIC_DATA_MEMBER_ID_UNSPECIFIED);
```

- Hierarchical name access is as efficient as bind/unbind access

Performance Improvements Accessing Members

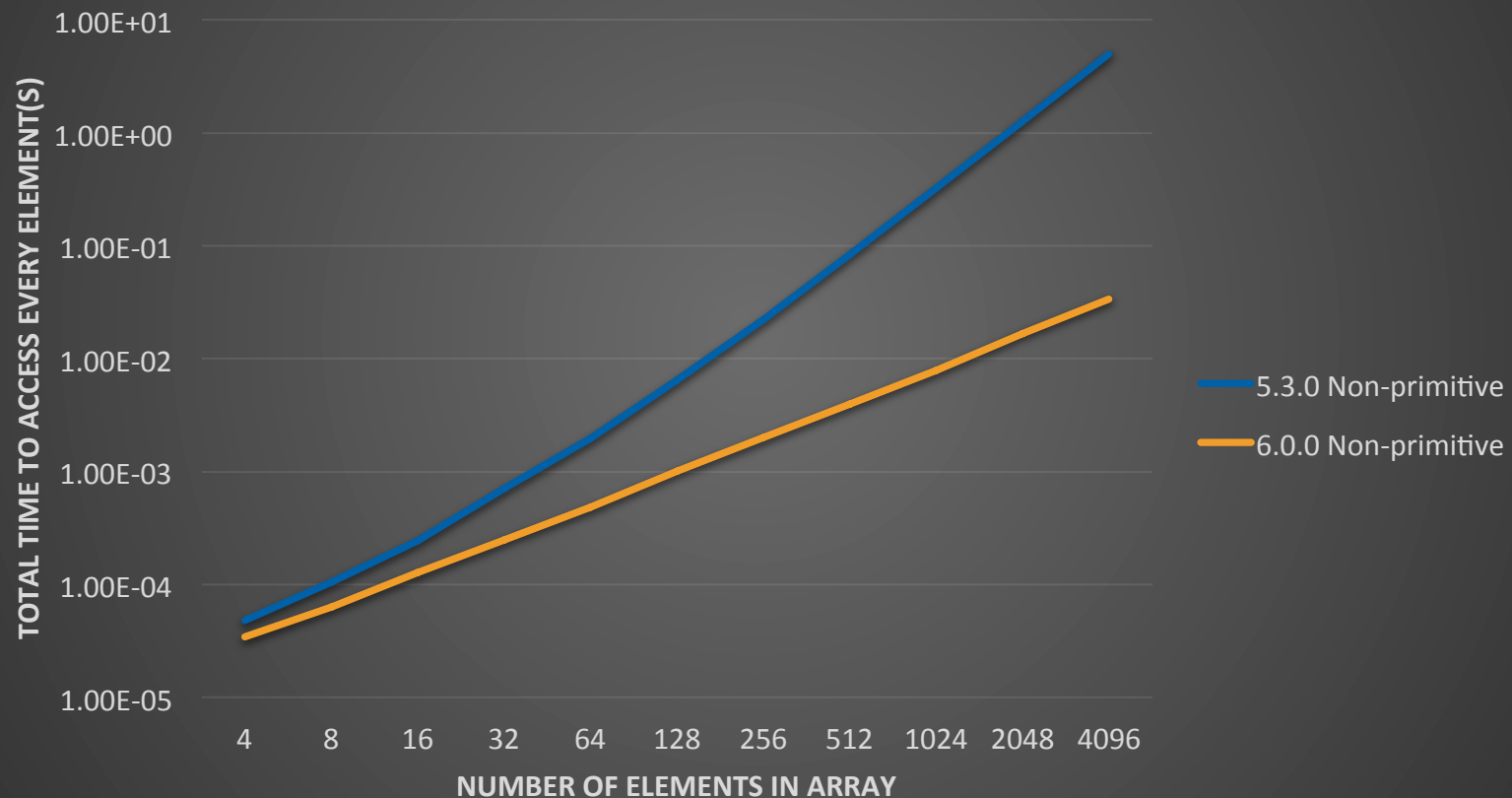
- One-level binding performance improvements: **3.5x faster** (bind, set_long, unbind, bind, get_long, unbind)
- Set/Get complex member (set_complex_member, get_complex_member): **2.7x faster**
- Set/Get string: **1.3x faster**
- Setting large octet sequence: **hundreds of times faster**
- Setting and getting members of a nested complex sequence (100 elements) using hierarchical name: **305x faster**

Connector Performance Improvements

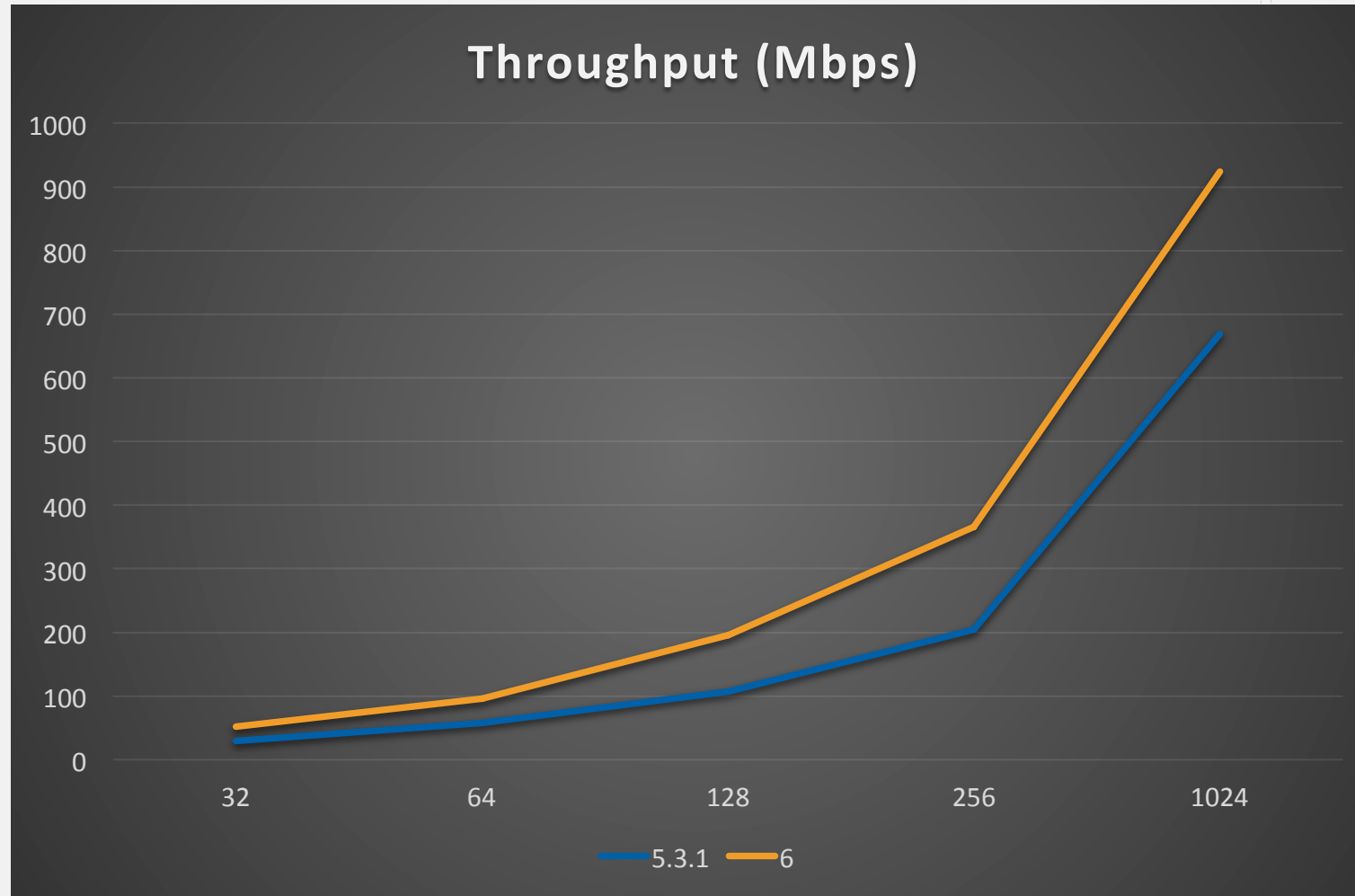
Platform

Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux
1 Gb network

Time to access every element in an array of a user defined type



Performance Improvements Sending Samples



Platform

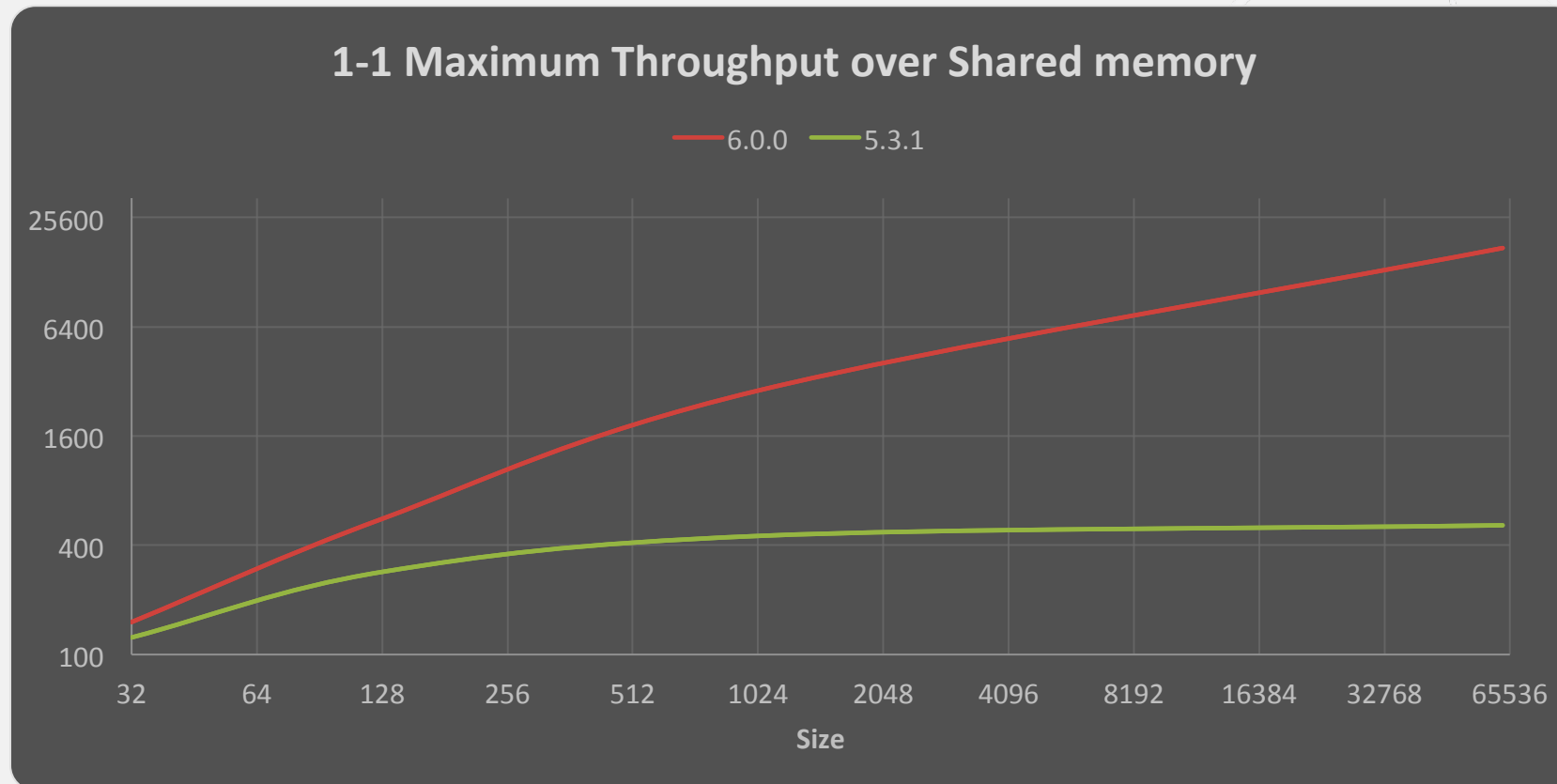
Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux
1 Gb network

Content Filter Performance Improvements



Content Filter Topics Enhancements

- Performance optimization for CFT
 - Only deserializing the filtered fields (instead of the whole sample)



Platform

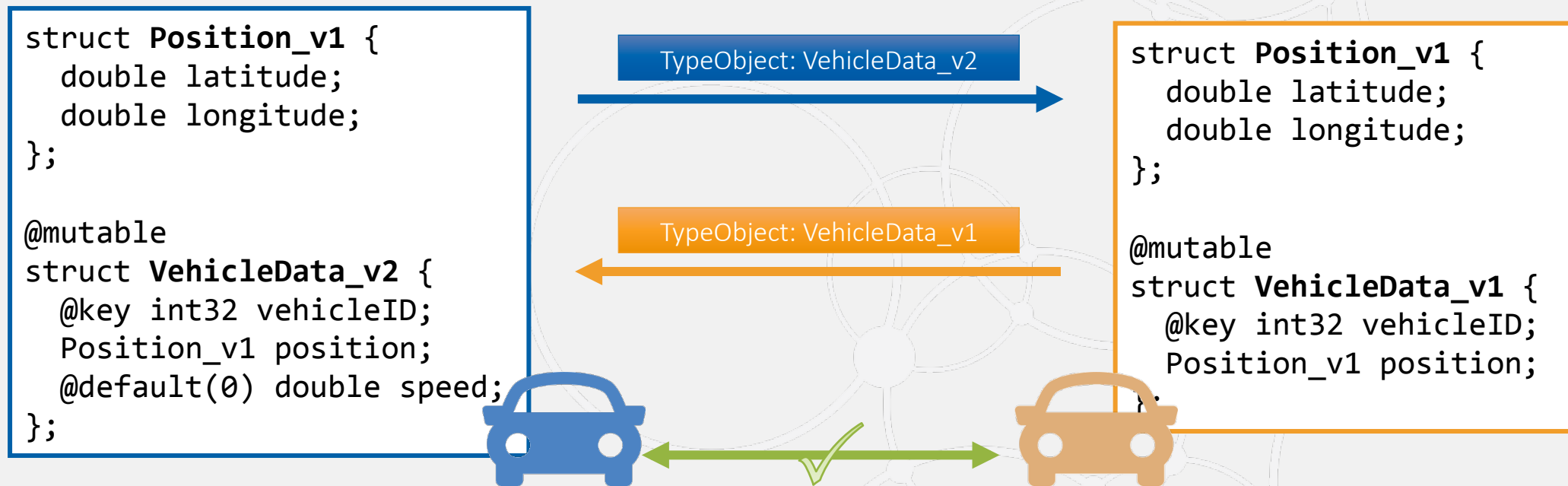
Intel i7 6-core CPU
3.33GHz
12 GB RAM
CentOS Linux

Discovery Bandwidth Reduction: TypeObject Compression



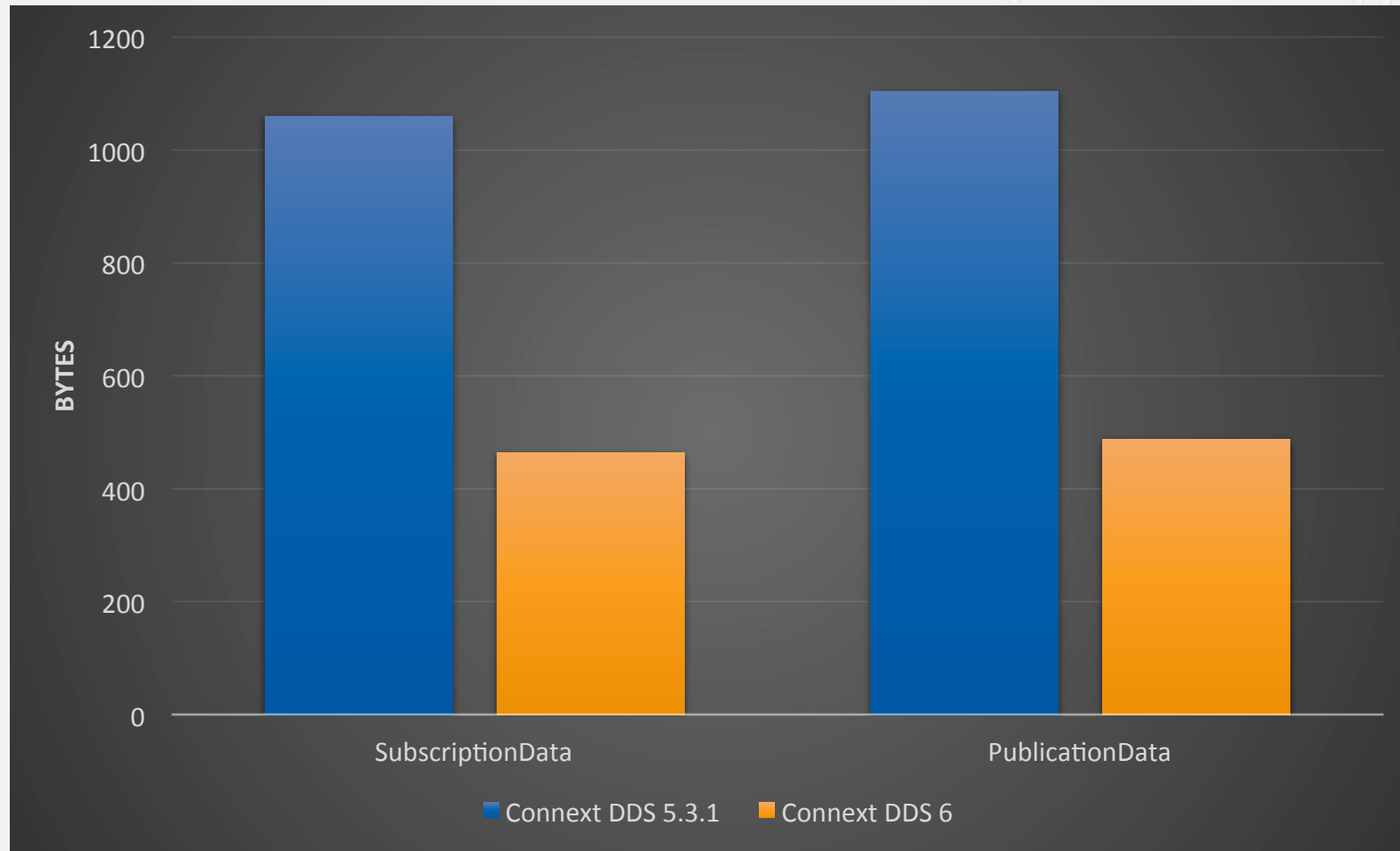
TypeObject Compression

- **TypeObject** describes data types: used for type matching

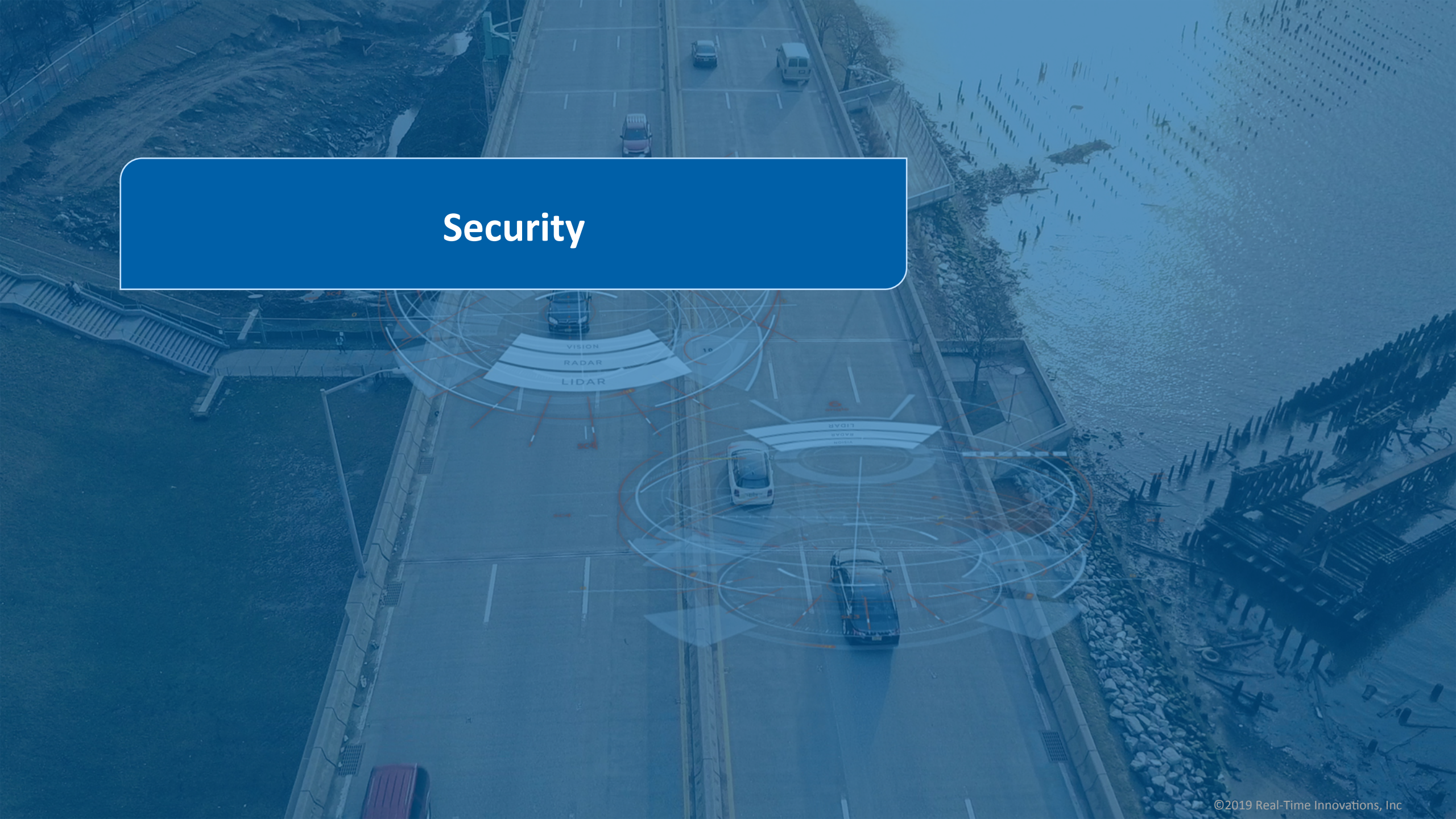


- For complex types, TypeObject could be pretty large
- 6.0.0 drastically reduces TypeObject's serialized size
- Compression enabled by default. Configurable through **DDS_DiscoveryConfigQosPolicy's** **endpoint_type_object_lb_serialization_threshold**

Discovery Bandwidth Improvements



Security



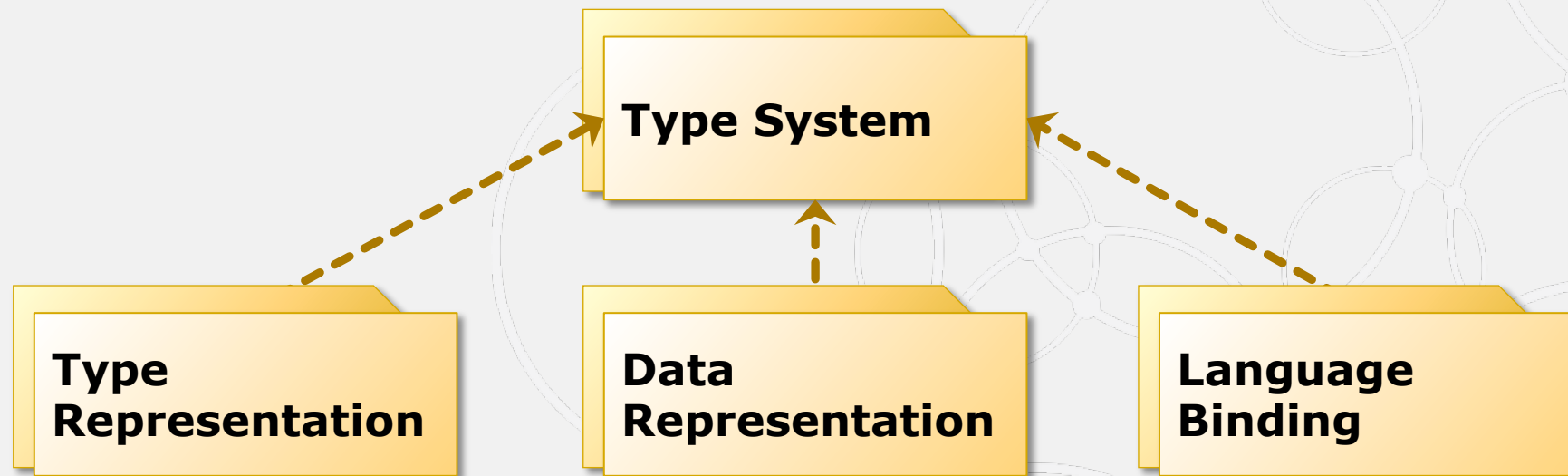
Connex 6 Security Support

- **Support across product line** including Micro, Pro, Infrastructure Services, and tools
- Full Compliance with OMG DDS Security 1.1 Built-in Plugins:
 - Plugins Configuration
 - Protection Kinds
 - Data Tagging
- **OMG DDS Security 1.1:**
 - New Protection Kinds
 - Governance Compatibility Detection
 - Revised Permissions
 - AuthRequest
 - Other (API changes, XSD changes)

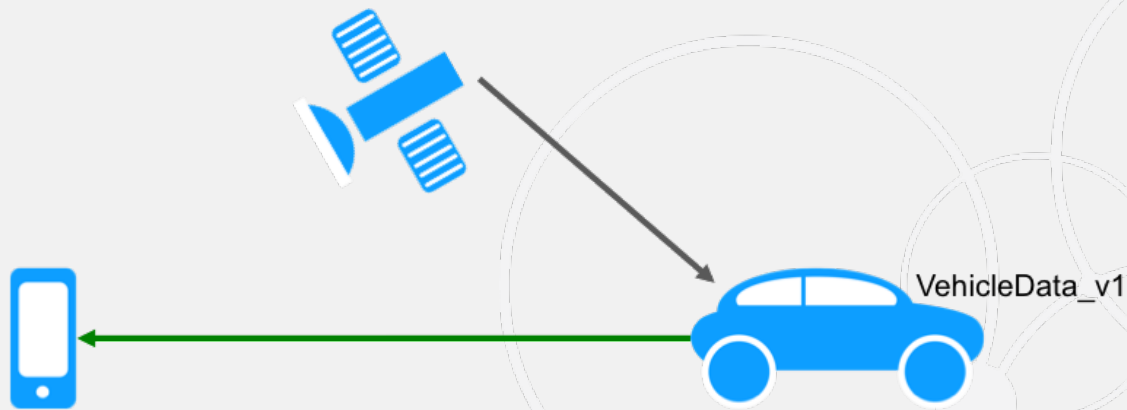
Extensible Types



X-Types Specification: Components



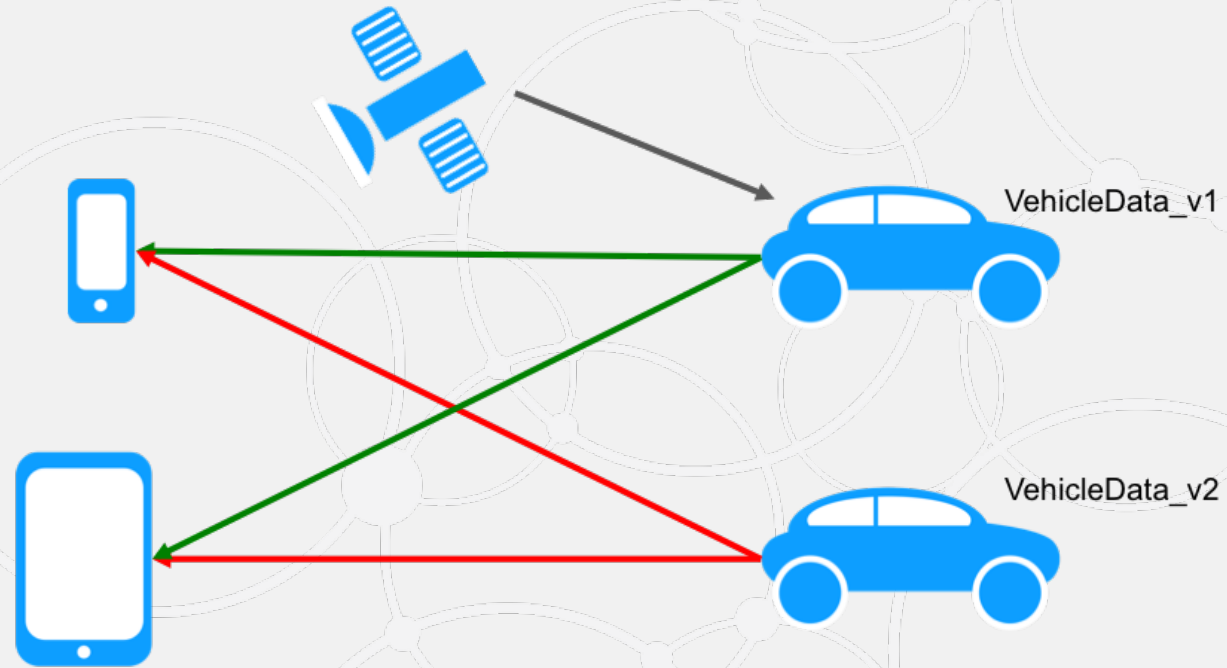
X-Types Specification: Type Evolution



```
struct Position_v1 {  
    double latitude;  
    double longitude;  
};  
  
struct VehicleData_v1 {  
    @key int32 vehicleID;  
    Position_v1 position;  
};
```

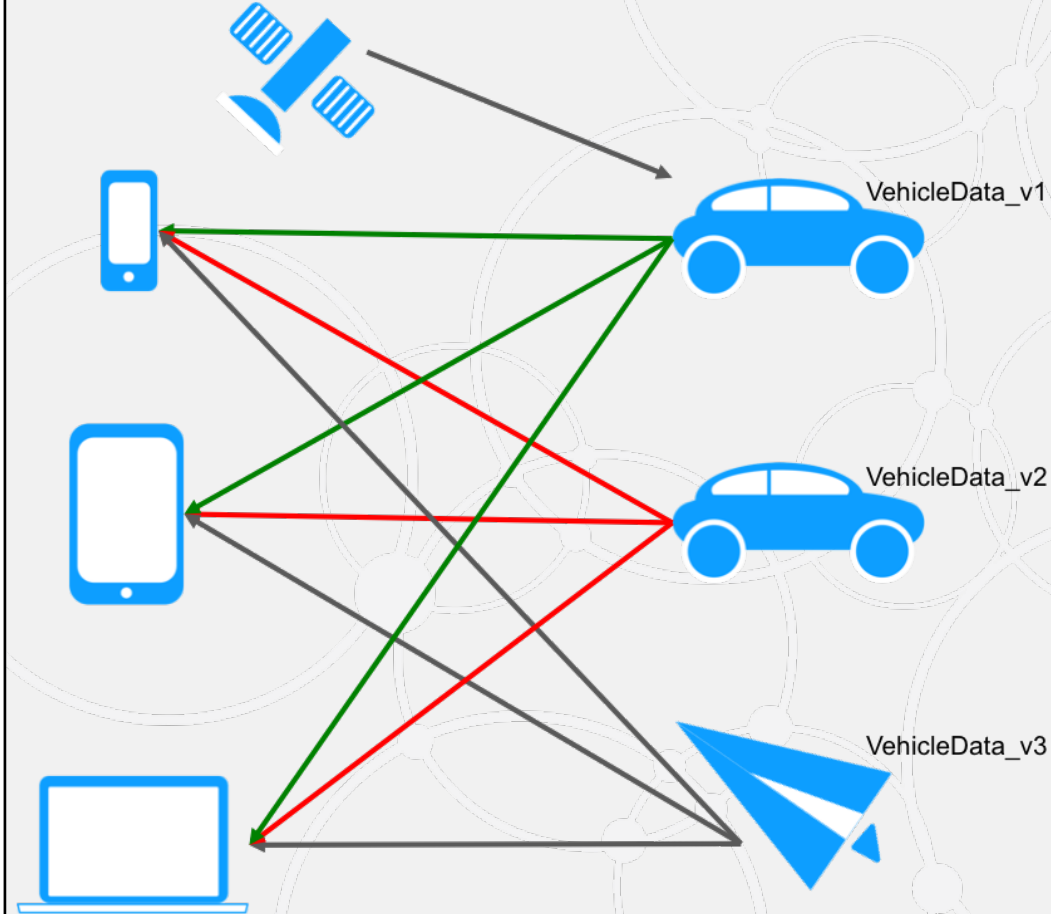
X-Types Specification: Type Evolution

```
struct Position_v1 {  
    double latitude;  
    double longitude;  
};  
  
struct VehicleData_v1 {  
    @key int32 vehicleID;  
    Position_v1 position;  
};  
  
struct VehicleData_v2 {  
    @key int32 vehicleID;  
    Position_v1 position;  
    @default(0) double speed;  
};
```



X-Types Specification: Type Evolution

```
struct Position_v1 {  
    double latitude;  
    double longitude;  
};  
  
struct Position_v2 {  
    double latitude;  
    double longitude;  
    @default(0) double altitude;  
};  
  
@mutable  
struct VehicleData_v1 {  
    @key int32 vehicleID;  
    Position_v1 position;  
};  
  
@mutable  
struct VehicleData_v2 {  
    @key int32 vehicleID;  
    Position_v1 position;  
    @default(0) double speed;  
};  
  
@mutable  
struct VehicleData_v3 {  
    @key int32 vehicleID;  
    Position_v2 position;  
    @default(0) double speed;  
};
```



X-Types 1.2 and IDL 4.1/4.2 in Connex 6

- More Flexible EXTENSIBLE Extensibility: **APPENDABLE**
- More efficient data representation (wire format) through **XCDR2** configurable using new DataRepresentationQosPolicy
 - Final Types
 - Extensible (Append) Types
 - Mutable Types
- Fine-grained control for type matching
- New IDL notation for fix-width integer types: int8, int16, int32, int64, uint8, uint16, uint32, uint64
- New annotations: **@appendable**, **@final**, **@mutable**, **@value**, **@default**, **@range**, **@min**, **@max**, **@unit**, **@default_literal**

More Flexible EXTENSIBLE Extensibility (APPEND)

Before XCDR2, appendable (extensible) types became incompatible if nested structure is extended with additional fields at the end

```
@appendable
struct Coordinates1 {
    float x;
    float y;
};
```

```
@appendable
struct Coordinates2 {
    float x;
    float y;
    float z; // Extra field
};
```



```
@appendable
struct ObservedPosition1 {
    Coordinates1 position;
    int64 timestamp;
};
```

```
@appendable
struct ObservedPosition2 {
    Coordinates2 position;
    int64 timestamp;
};
```



```
@appendable
struct Rectangle1 {
    Coordinates1 bottom_left;
    Coordinates1 top_right;
};
```

```
@appendable
struct Rectangle2 {
    Coordinates2 bottom_left;
    Coordinates2 top_right;
};
```



New IDL Annotations

```
struct Position {  
    @range(-90,90)  
    double latitude;  
    @min(-180)  
    @max(180)  
    double longitude;  
};
```

@range, @max, @min

- Enforced during serialization/deserialization
- Not propagated with TypeObject
- Not used for matching

@unit

- Only informative. Not enforced
- Not propagated with TypeObject
- Not used for matching

@default_literal

- Used to specify default value for Enums
- Not used for matching
- Not propagated with TypeObject

```
enum Color {  
    @value(3)  
    RED,  
    @value(4)  
    @default_literal  
    BLUE,  
    @value(127)  
    GREEN  
};
```

```
struct Position {  
    @default(45)  
    double latitude;  
    @default(90)  
    double longitude;  
};
```

@default

- Used to initialize primitive member
- Not used for matching
- Not propagated with TypeObject

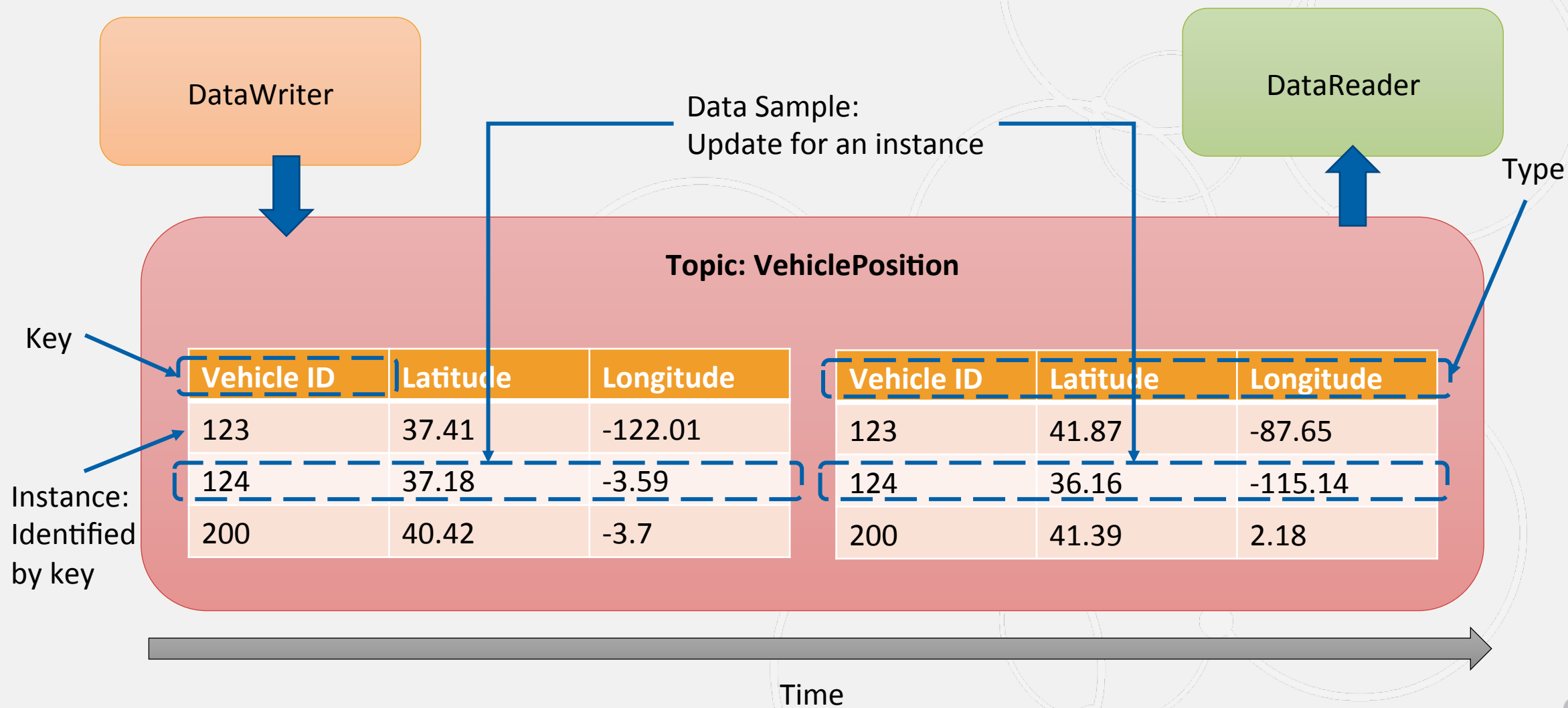
In the works ...



Instance Scalable Distribution and Management

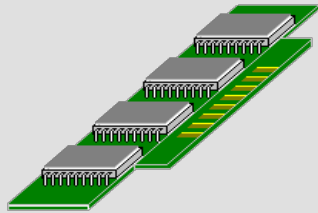


Instances

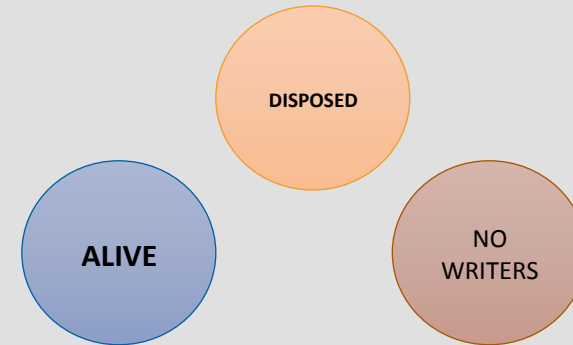


Focus Areas

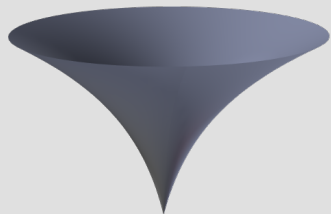
Resource Management



Instance State Transmission



Content Filtering



Debuggability

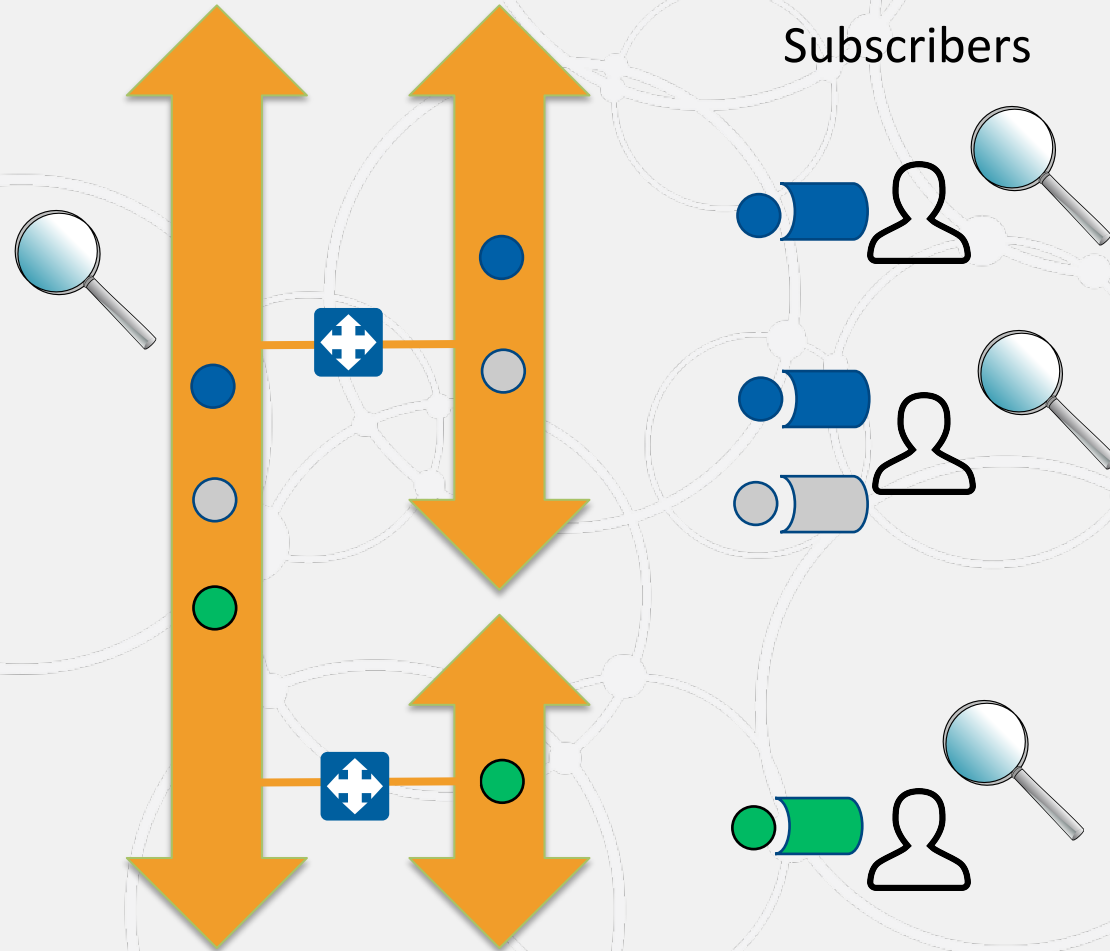
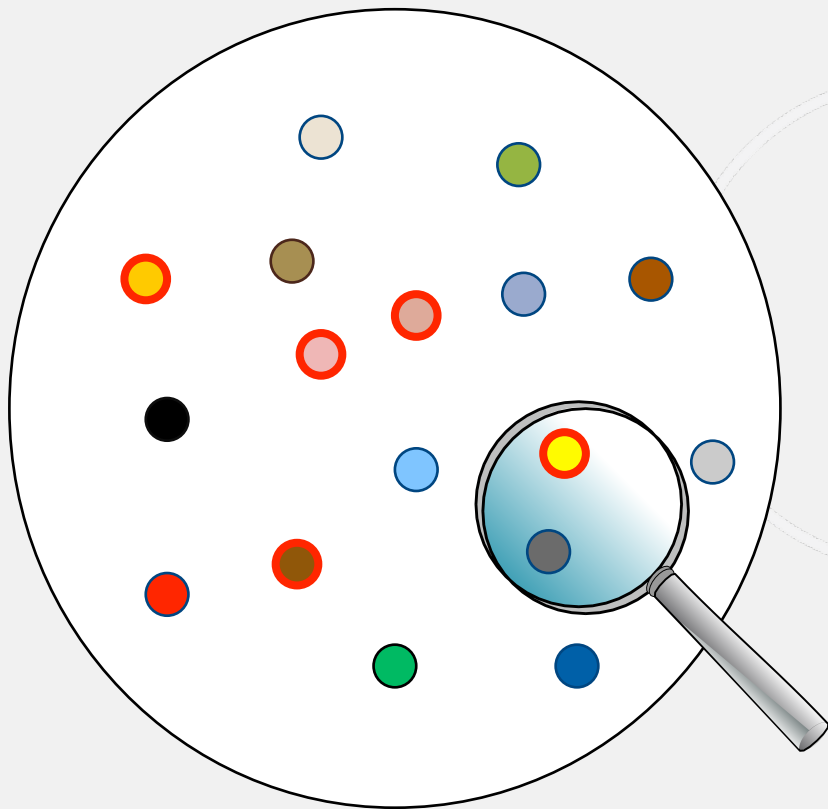


Efficient Instance Content Filtering

Information World is Sparsely Subscribed

Instance Universe
(Potentially Hundreds of Thousands)

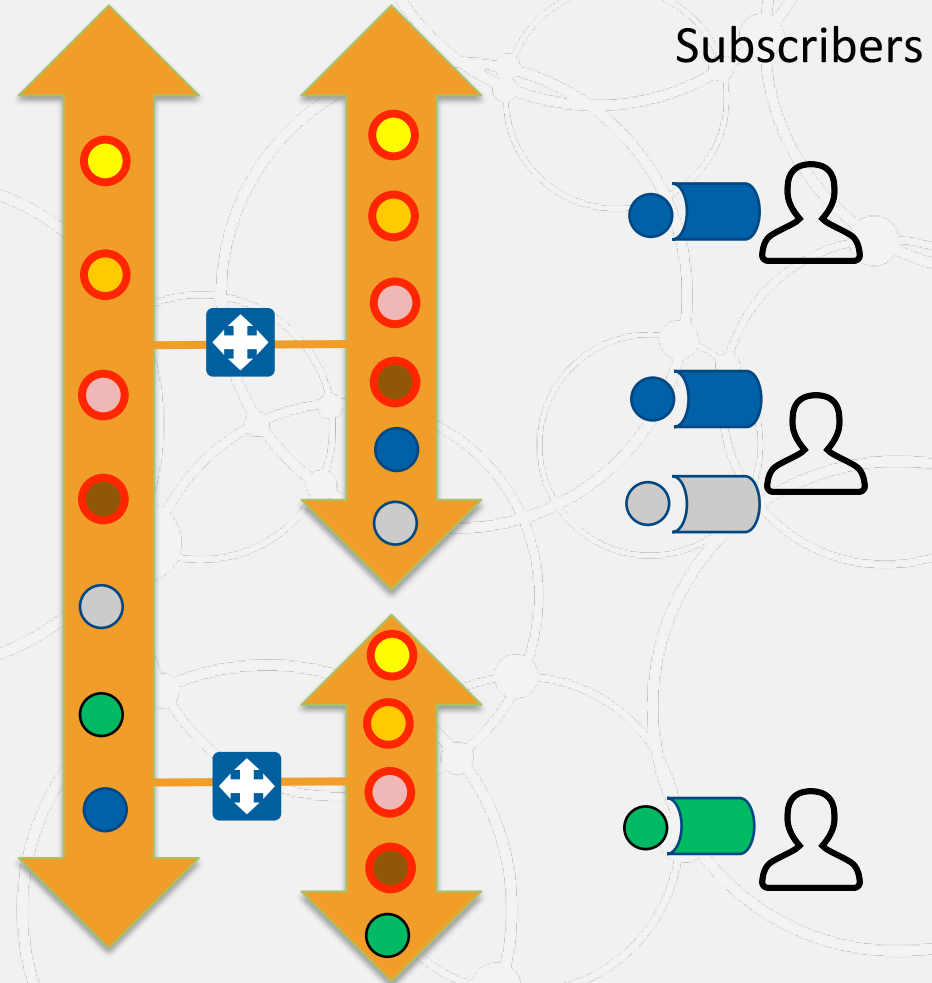
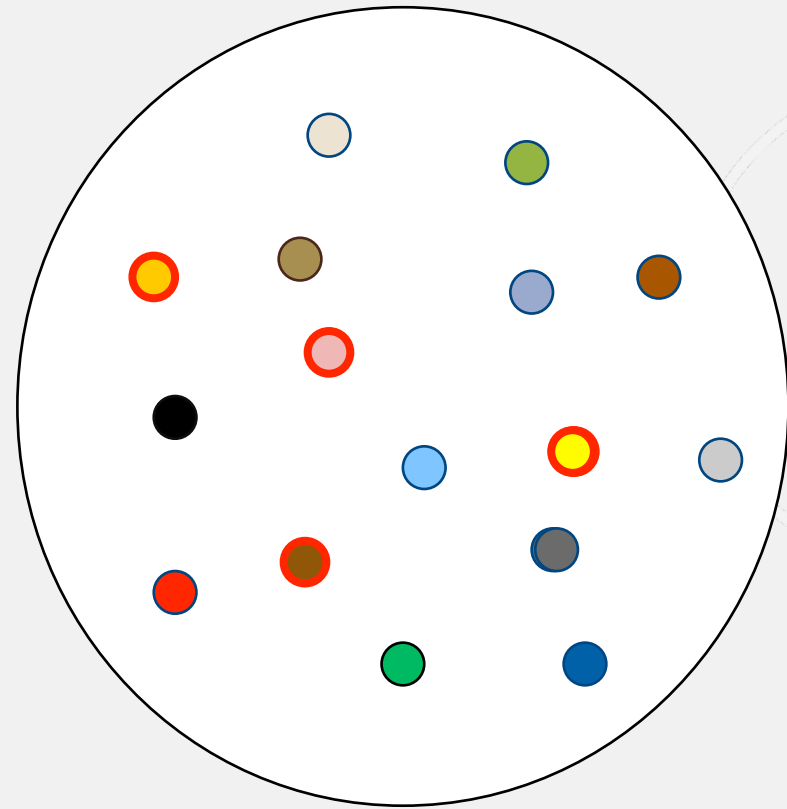
Subscribers



Efficient Instance Content Filtering

Information World is Sparsely Subscribed

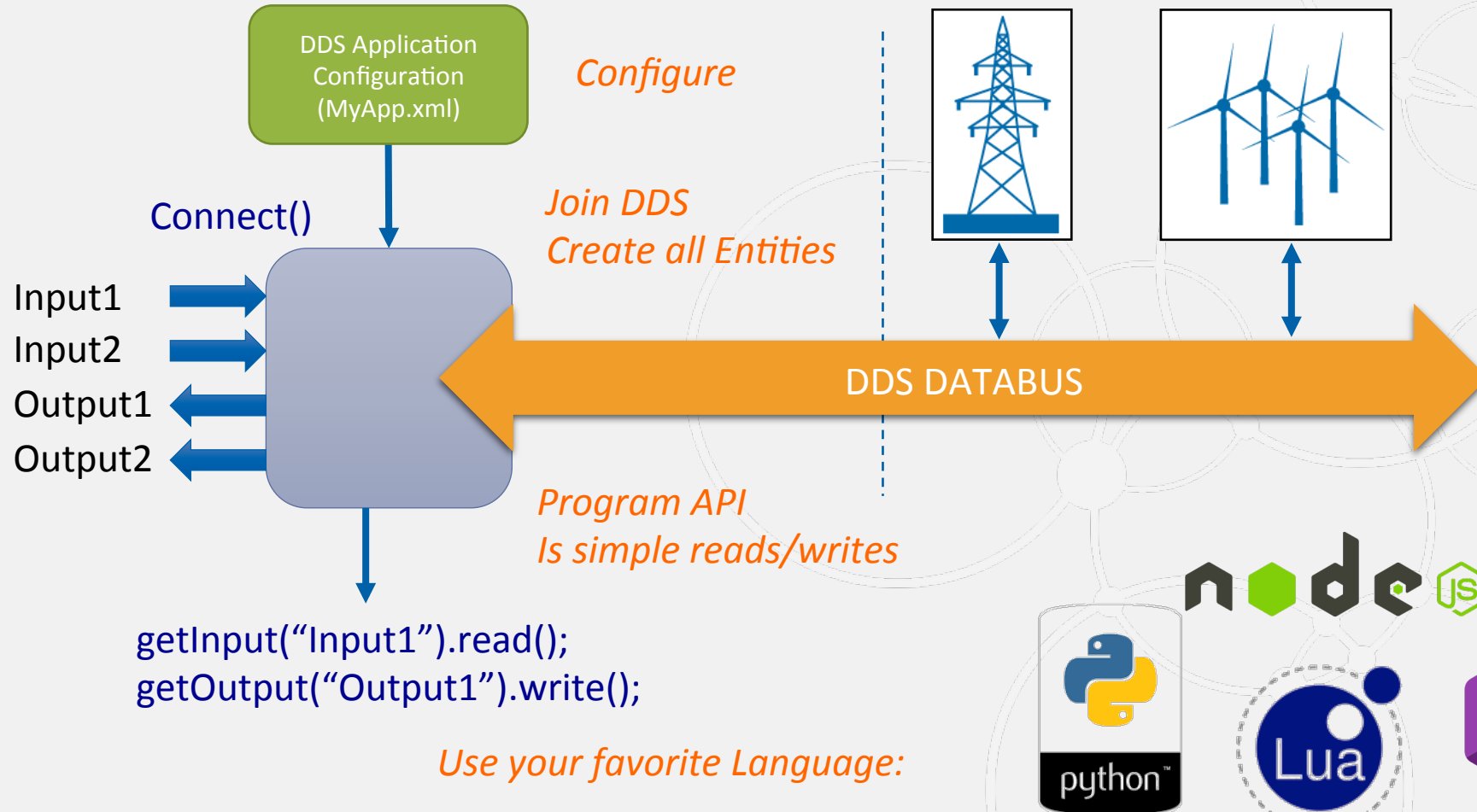
Instance Universe
(Potentially Hundreds of Thousands)



Connector: Python and Javascript APIs



RTI Connector



- Simplified DDS API
- Leverage XML definition of DDS entities
- Get started in minutes (just pip/npm install)

Python and Javascript APIs

```
import rticonnextdds_connector as rti

with rti.open_connector(
    config_name = "MyParticipantLibrary::MySubParticipant",
    url = file_path + "../ShapeExample.xml") as connector:

    input = connector.get_input("MySubscriber::MySquareReader")

    print("Waiting for publications...")
    input.wait_for_publications() # wait for at least one matching publication

    print("Waiting for data...")
    for i in range(1, 500):
        input.wait() # wait for data on this input
        input.take()
        for sample in input.samples.valid_data_iter:
            # You can get all the fields in a get_dictionary()
            data = sample.get_dictionary()
            x = data['x']
            y = data['y']

            # Or you can access the field individually
            size = sample.get_number("shapsize")
            color = sample.get_string("color")
            print("Received x: " + repr(x) + " y: " + repr(y) +
                  " size: " + repr(size) + " color: " + repr(color))
```

```
<domain_participant name="MySubParticipant" domain_ref="MyDomainLibrary::MyDomain">
  <subscriber name="MySubscriber">
    <data_reader name="MySquareReader" topic_ref="Square" />
  </subscriber>
</domain_participant>
```

- Simplified DDS API
- Leverage XML definition of DDS entities
- Get started in minutes (just pip/npm install)

Productized this year (Q4)

- Major robustness improvements
- Redesigned APIs: python- and js-friendly
- New functionality:
 - Dispose and unregister
 - Waiting for data, for discovery, and for acks
 - More flexible and efficient access to the data samples
 - Support for XTypes
- New documentation ([preview](#))
- .NET Core and Go APIs available as experimental

Coherent Access Across Topics



Coherent Access Across Topics



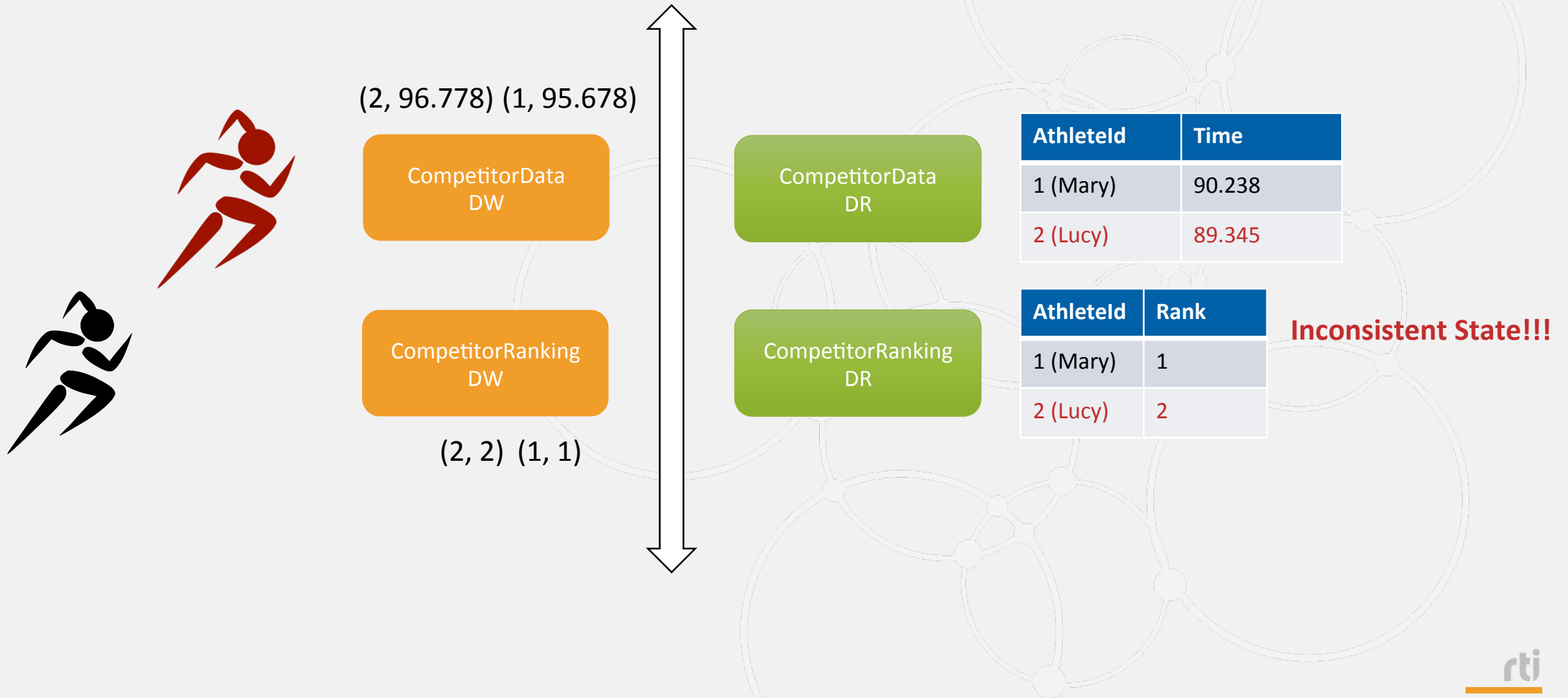
Topic 1
CompetitorData

```
struct CompetitorData  
{  
    @key long Athleteld;  
    long long Time;  
};
```

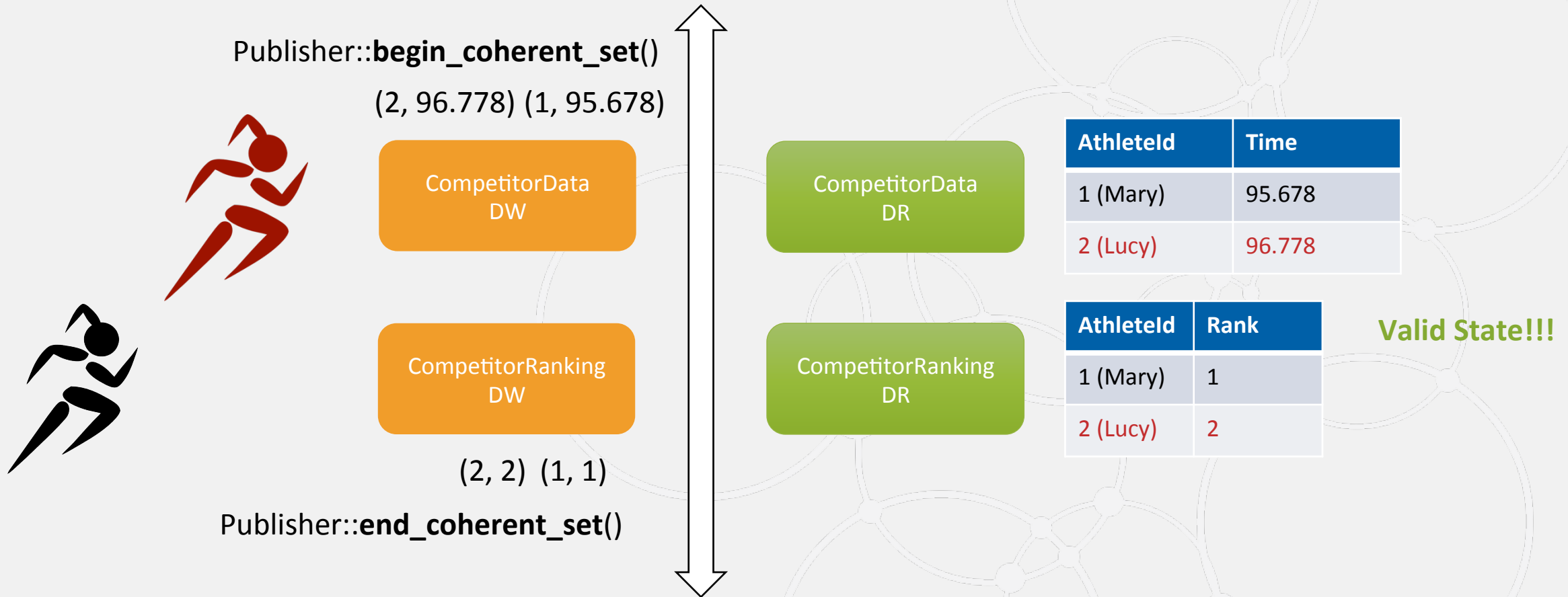
Topic 2
ComptetitorRanking

```
struct  
CompetitorRanking  
{  
    @key long Athleteld;  
    short Rank;  
};
```

Coherent Access Across Topics



Coherent Access Across Topics



Coherent Access Across Topics: How to Configure

- DataWriters that require ordered and/or coherent access must belong to the same Publisher
- DataReaders must belong to the same DDS Subscriber
- PresentationQosPolicy in Publisher/Subscriber must be set as follows:

access_scope = GROUP_PRESENTATION_QOS

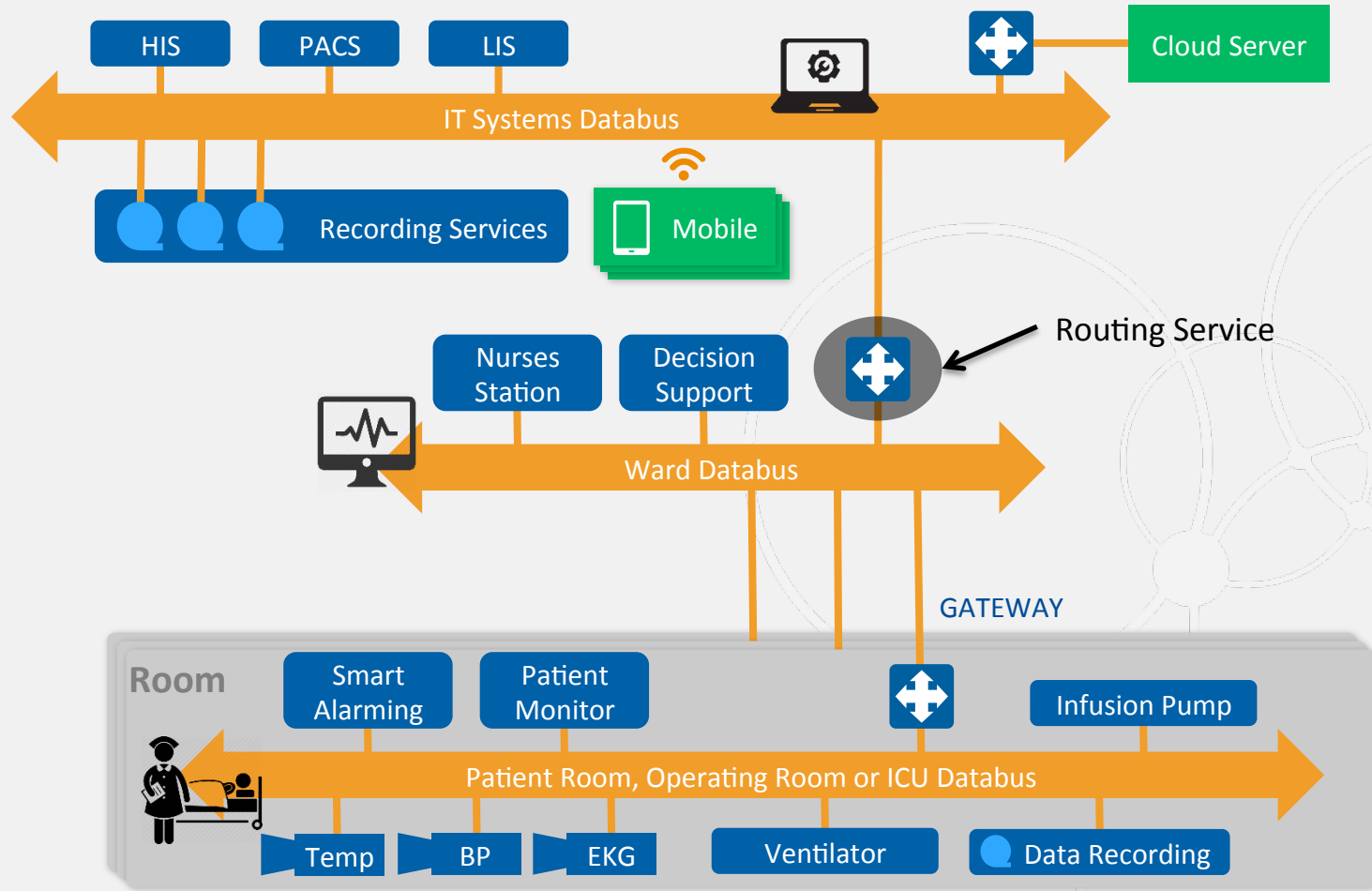
ordered_access = TRUE (for ordered access)

coherent_access = TRUE (for coherent access)

Debuggability



Challenges in IIoT System: Debuggability



- Complex systems
- Difficult to know what is going on
- Our goal: to make easier to debug these systems

Customer 'Y': Memory leak issue



- RS was crashing
- System was running out of memory after running RS for some time
- Several weeks back and forward

NDDS[WARNNG]: RTIOsapiHeap_allocateBufferAligned:errno = 12

Customer 'Y': Memory leak issue



```
<reader_data_lifecycle>
  <autopurge_nowriter_samples_delay>
    <sec>20</sec>
    <nanosec>0</nanosec>
  </autopurge_nowriter_samples_delay>
</reader_data_lifecycle>
```

- ... a configuration issue!!!! They were leaking instances
- The problem:
 - Instance lifecycle
 - We knew about it, and solved in QoS, BUT
 - Customer changed QoS profile

Focus Areas

Statistics Collection



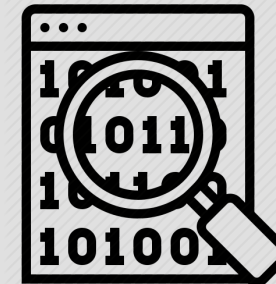
Monitoring



Logging Enhancements



Enhanced Debugging Information



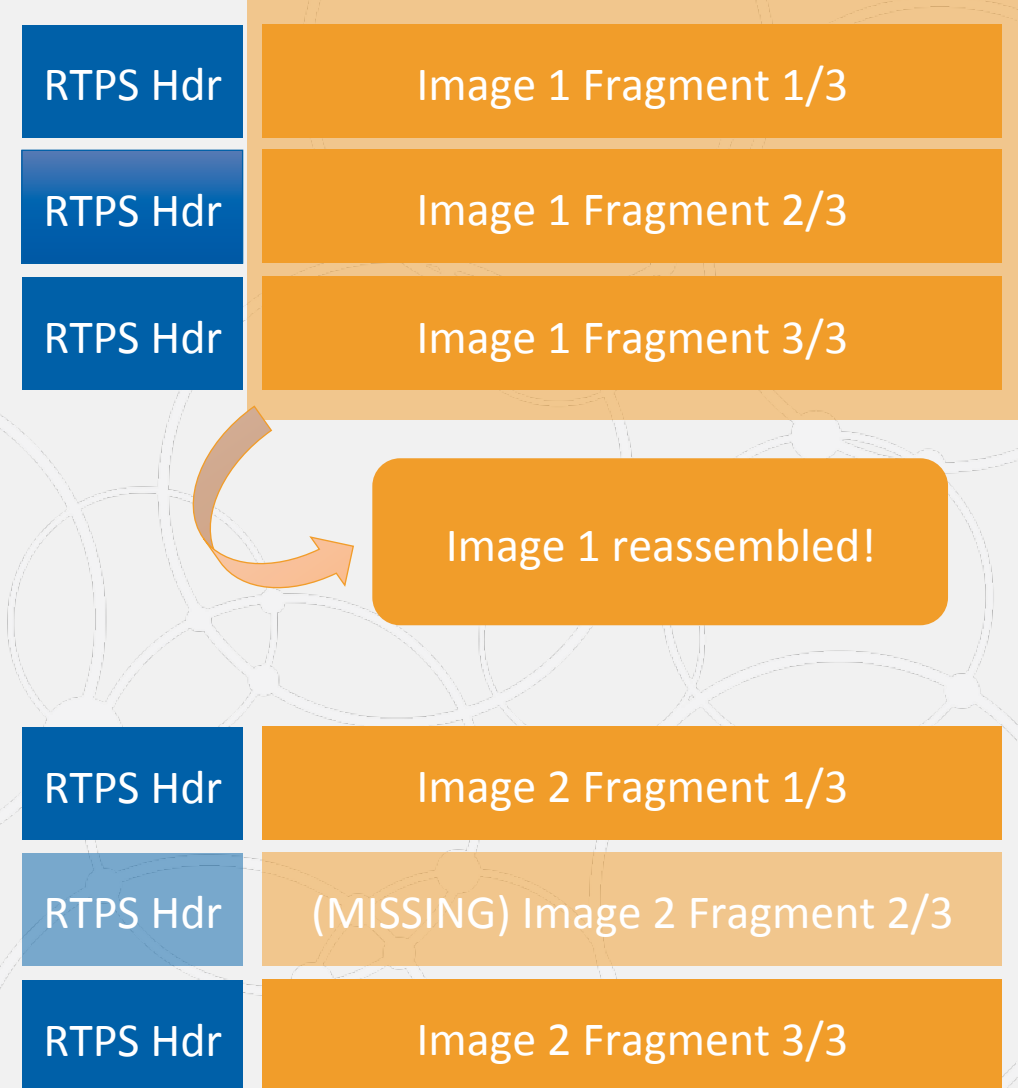
Instance Statistics

- Gather instance statistics:
 - How many instances are there in the system?
 - What was the maximum number of instances?
- Expose those statistics
 - As part of entity's statuses
 - As part of the monitoring libraries

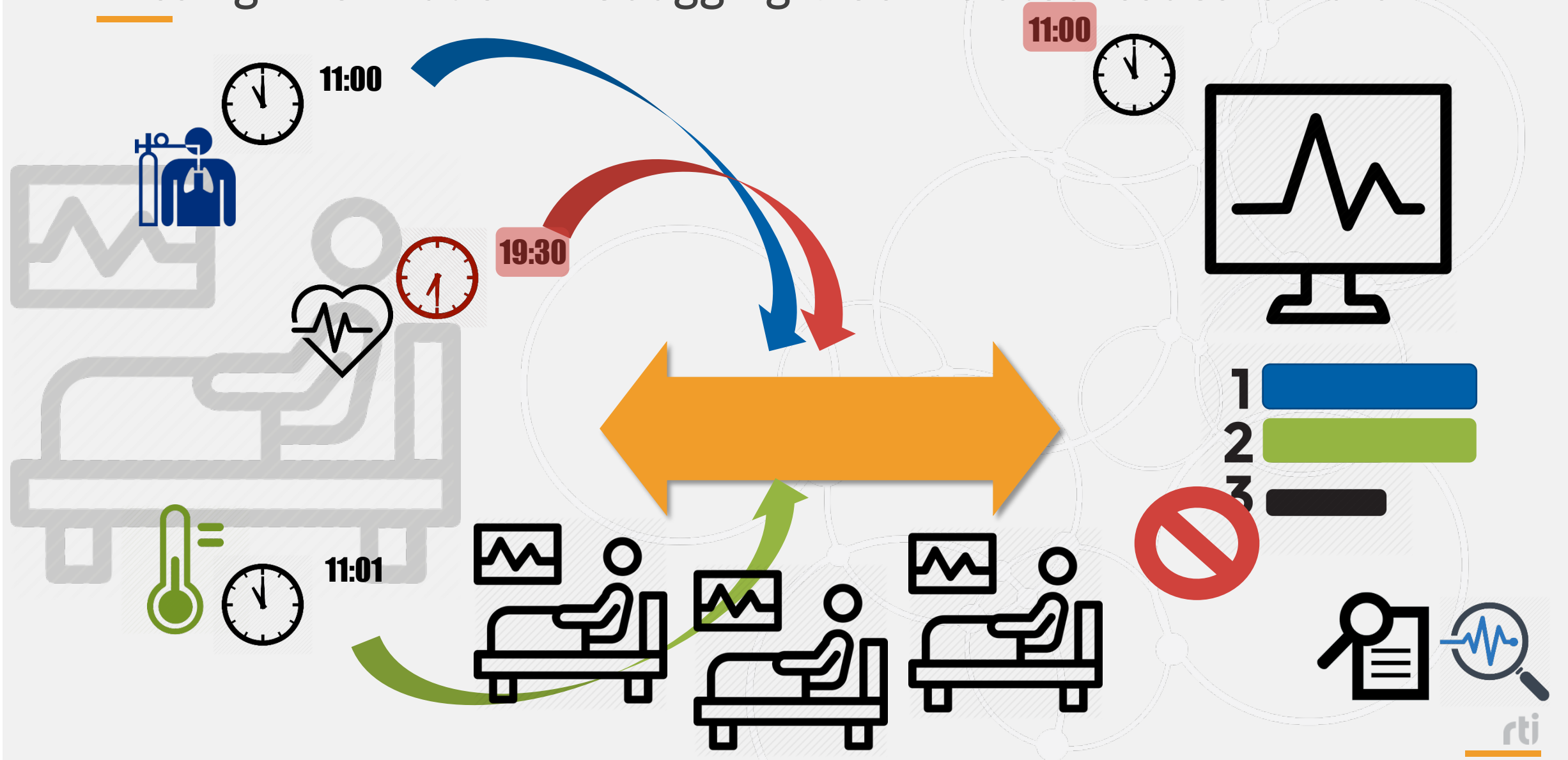


Large Data statistics

- RTPS fragmentation use cases:
 - Systems with large data types
 - Networks with no IP-fragmentation
- Currently hard to identify when there are communication issues
- Gather fragments statistics:
 - Reassembled samples
 - Incomplete samples
 - Received fragments
 - Missing fragments
 - Sent fragments
 - ...
- Expose those statistics
 - As part of writer/reader statuses
 - As part of the monitoring libraries



Missing Information: Debugging Clock-related Issues is Hard!



Debugging Enhancements: Thread Information

- Added thread info to all RTI threads:

```
(gdb) info thread
```

	Id	Target	Id	Frame	
*	1	Thread	0x7ffff7fda740	(LWP 2642)	"HelloWorldApp" ...
	2	Thread	0x7ffff67d8700	(LWP 2653)	"rcxDtb065f77db375" ...
	3	Thread	0x7ffff5fd7700	(LWP 2654)	"rcxEvt065f77db375" ...
	4	Thread	0x7ffff553c700	(LWP 2655)	"rcxITrudpv4" ...
	5	Thread	0x7ffff4d3b700	(LWP 2656)	"rcxR00065f77db375" ...
	[...]				
	10	Thread	0x7fffe5ffb700	(LWP 2691)	"rcxDtb065d35bda78" ...
	11	Thread	0x7fffe56e9700	(LWP 2692)	"rcxEvt065d35bda78" ...
	12	Thread	0x7fffe4c4e700	(LWP 2693)	"rcxITrudpv4" ...
	13	Thread	0x7fffc7fff700	(LWP 2694)	"rcxR00065d35bda78" ...
	14	Thread	0x7fffc77fe700	(LWP 2695)	"rcxR01065d35bda78" ...
	15	Thread	0x7fffc6ffd700	(LWP 2696)	"rcxR02065d35bda78" ...
	16	Thread	0x7fffc67fc700	(LWP 2697)	"dds_c1Testerd" ...
	17	Thread	0x7fffc5ffb700	(LWP 2698)	"rcxR04065d35bda78" ...



Enhanced Logging



- 5.2.0:

`PRESWriterHistoryDriver_initializeSample:!serialize` (Not really helpful)

- 6.0.0 enhancements:

`PRESWriterHistoryDriver_initializeSample:serialize sample error in topic 'PatientMonitoring' with type 'PatientMetrics'`

- Future Configurable context:

- Allow to configure the logging of relevant information (context) to all the log messages: **Domains, topic and type names...**

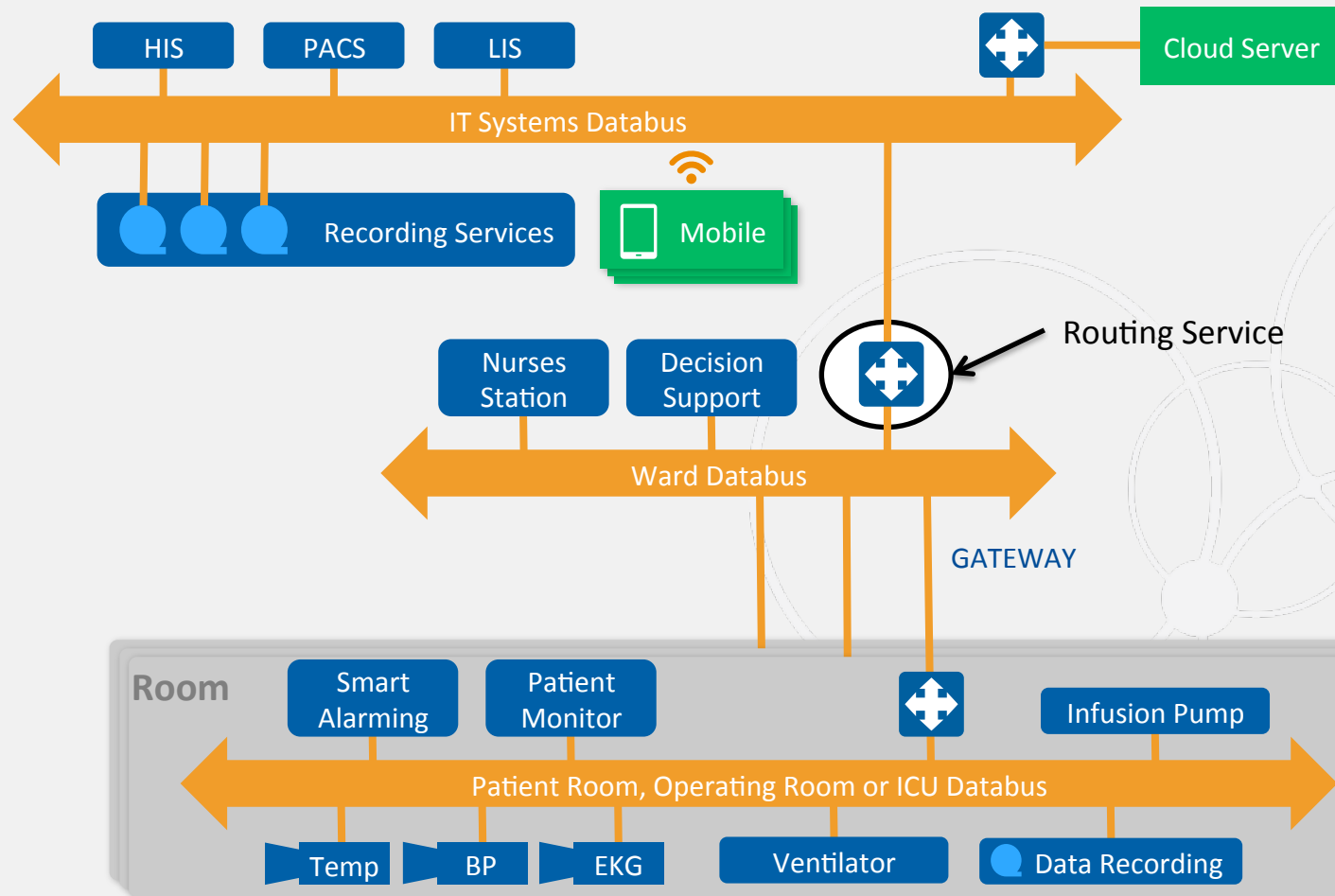
[D=64 | MyParticipant/MyDataWriter | PatientMonitoring:PatientMetrics | WRITE]

`PRESWriterHistoryDriver_initializeSample:serialize sample error`

Final Remarks



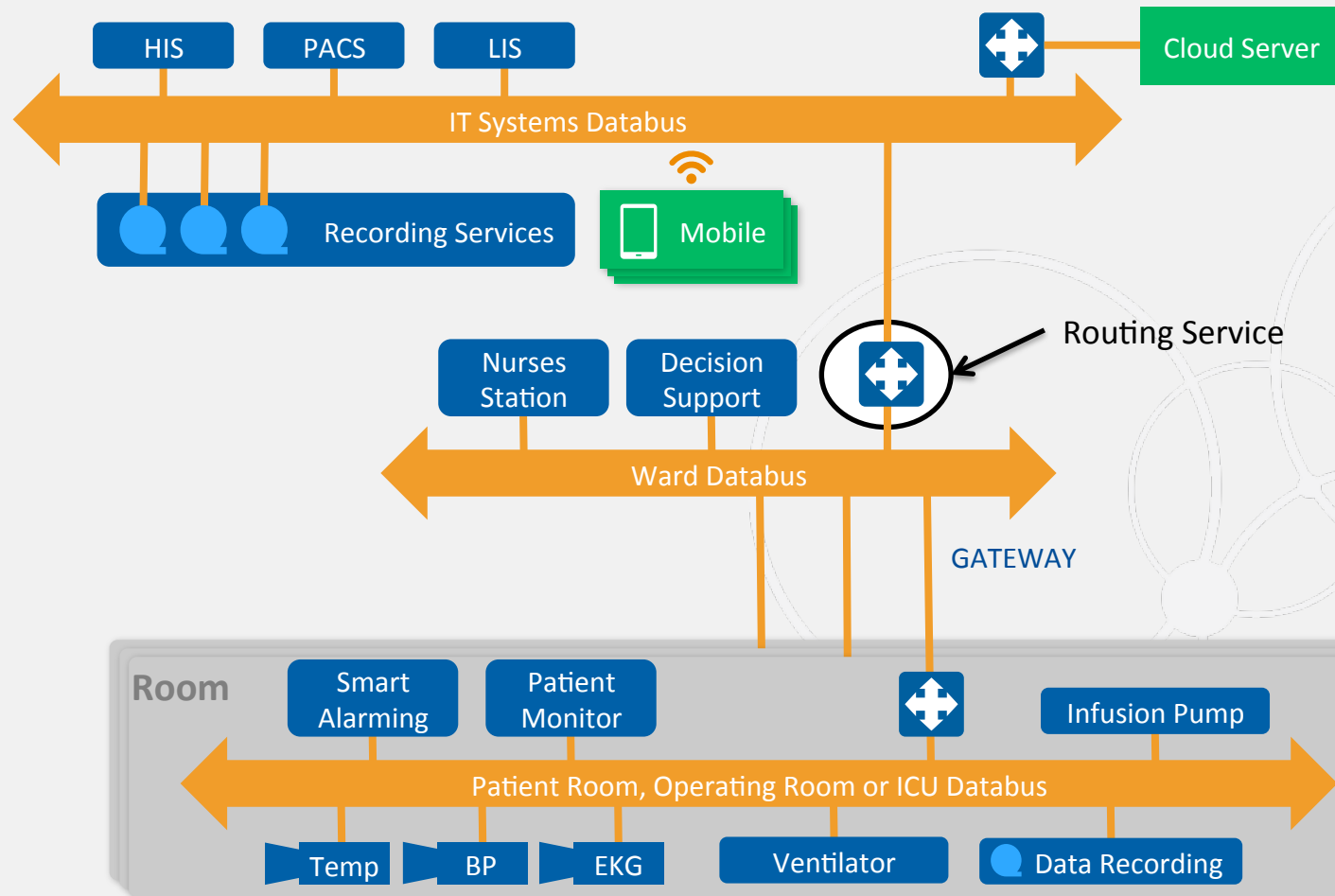
Expanding IIoT Systems: Layered Architecture



Connext 6 Capabilities

- **Routing Service and Recording Service:** New key capabilities for system Integration and better scalability
- New features to improve **Large Data** transmission performance
- **Connext DDS Secure:** New Data-Centric Security capabilities
- **Extensible Types:** Wire efficiency and type evolution improvements
- Various performance improvements: **Discovery, DynamicData, Content-Filter, Serialization**
- Better **RTI platform** integration

Expanding IIoT Systems: Layered Architecture



Future Work

- **Instance** management and scalability improvements
- **Debuggability** improvements
- **Integration with other middleware technologies:** OPC-UA, MQTT, AUTOSAR

Where to Find More Information

- RTI Blog: <https://www.rti.com/blog>
- RTI Website: <http://www.rti.com>
- RTI Community: <https://community.rti.com/>



Thank you

fernando@rti.com

