Webinar will begin momentarily...

Dr. Angelo Corsaro *[angelo.corsaro@prismtech.com]*

OpenSplice DDS Product Marketing Manager, PrismTech

Angelo co-chairs the OMG Data Distribution Service (DDS) Special Interest Group and the Real-Time Embedded and Specialized Services (RTESS) Task Force. He is a well known figure in the distributed real-time and embedded systems middleware community and has a wealth of experience in hard real-time embedded systems, large-scale and very large-scale distributed systems, such as defense, aerospace, homeland security and transportation systems. Prior to joining PrismTech, he worked for the SELEX-SI CTO Directorate, a FINMECCANICA company, where his responsibilities included mapping business requirements to technology capabilities, strategic standardization and technology innovation.

Erik Hendriks *[erik.hendriks@prismtech.com]*

OpenSplice DDS Senior Software Engineer, PrismTech

Erik has more than 10 years of experience in the design and development of high performance Real-Time and Distributed Systems. Over the past few years Erik has been one of the main contributor of OpenSplice DDS, and specifically of the Data Local Reconstruction Layer (DLRL). Erik is a recognized authority on DLRL, and regularly runs tutorial on the topic. Before joining PrismTech Erik was working for THALES Naval Netherlands were he was one of the key engineers working on DDS. Erik holds a MS in Electrical Enginering from the University of Twente, Netherlands.

PRISMTECH
Productivity Tools & Middleware

OpenSplice DDS

Data Local Reconstruction Layer

--Bringing the Data Centric Paradigm on Steroids--

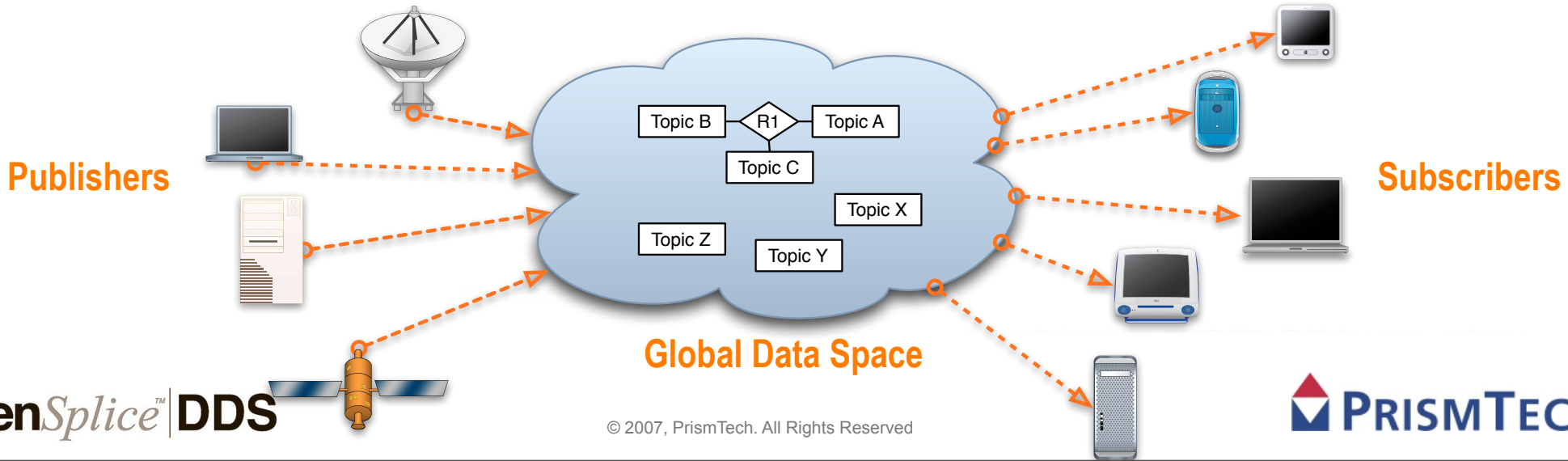
Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

Dr. Angelo Corsaro

OpenSplice DDS

- **An High Performance Real-Time Data-Centric Publish/Subscribe Middleware**
  - *The right data, at the right place, at the right time -- all the time!*
  - *Fully distributed, high performance, highly scalable, and high availability architecture*
- **Perfect Blend of *Data-Centric* and *Real-Time Publish/Subscribe* Technologies**
  - *Content based subscriptions, queries, and filters*
  - *Fine grained tuning of resource usage and data delivery and availability QoS*
  - *Optimal networking and computing resources usage*
- **Loosely coupled**
  - *Plug and Play Architecture with Dynamic Discovery*
  - *Time and Space Decoupling*
- **Open Standard**
  - *Complies with the full profile of the OMG DDS v1.2*

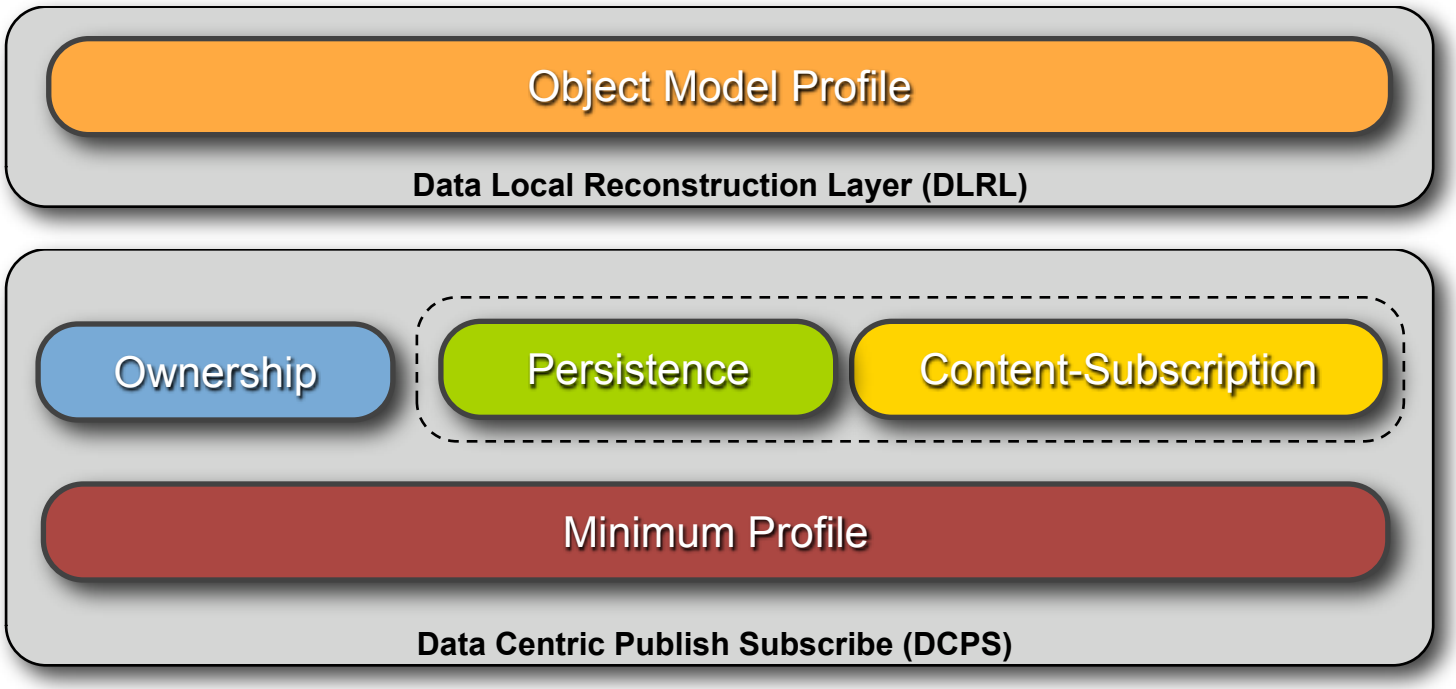


OpenSplice DDS

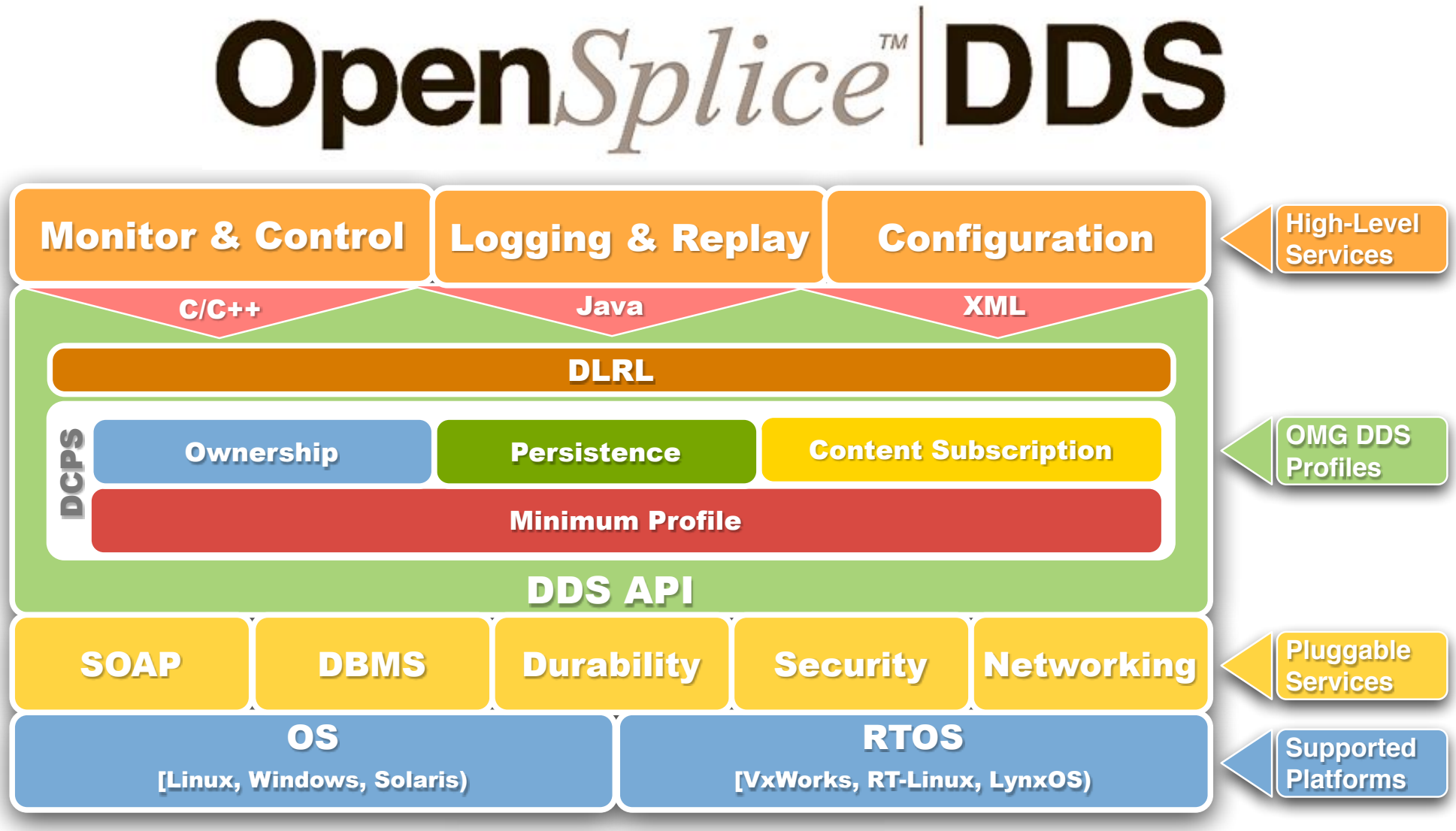
PRISMTECH

Standard Compliance

- OpenSplice DDS is compliant with the full profile specified in the OMG DDS Specification v1.2

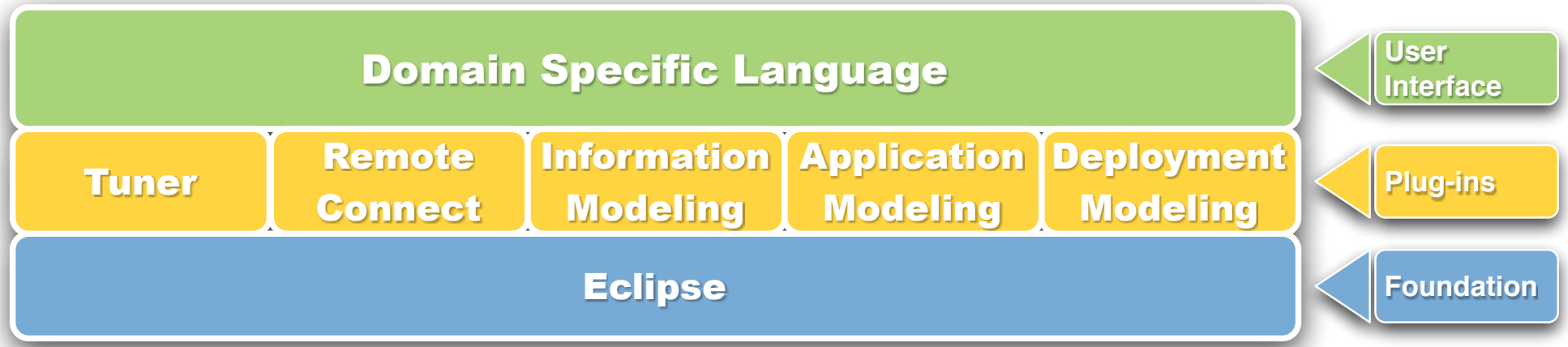


OpenSplice™ DDS



OpenSplice DDS PowerTools

OpenSplice™ DDS



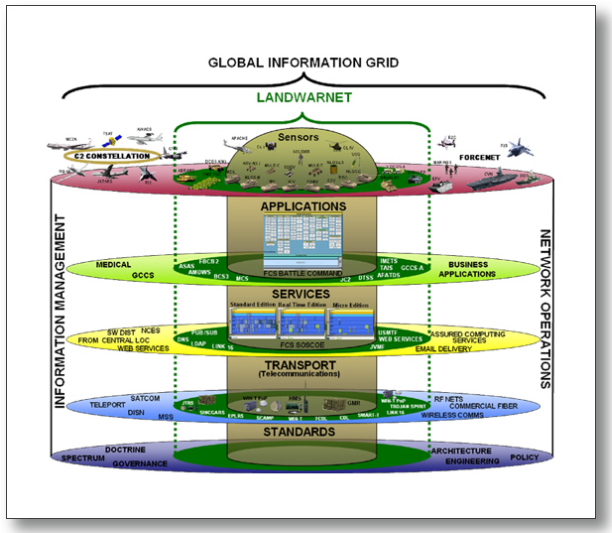
Who is using OpenSplice DDS

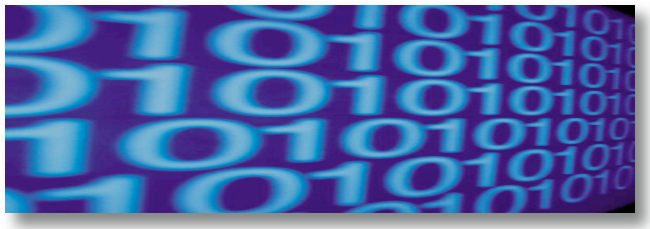
Defense

- **TACTICOS-CMS:** THALES Naval Netherlands' CMS, 26 ships classes, >100 ships
  - > 2.000 deployed runtimes (running on Solaris-servers, Linux-consoles, and vxWorks embedded subsystems)
  - 15 Navies worldwide (Netherlands, Germany, Turkey, Greece, Oman, Qatar, Portugal, South Korea, Japan, Poland,…)
- **USA programs**: LCS/GD, ENFMC/NG, LHA-LHD/DRS
- **Brazilian Navy**
- **Australia: DSTO**, ADI (Australia)
- THALES Naval NL's Flycatcher system
  - 4 army's, >400 deployments
- **NSWC**: Open Architecture Test Facility (OA-TF)

Tactical networks

- **Ultra Electronics** (US, UK): OpenSplice DDS selected over competition for superior scalability and fault-tolerance





Who is using OpenSplice DDS

Transportation

- **Amsterdam Metro**
- **CoFlight** Flight-plan management system upgrades for France, Italy, Switzerland

Aerospace

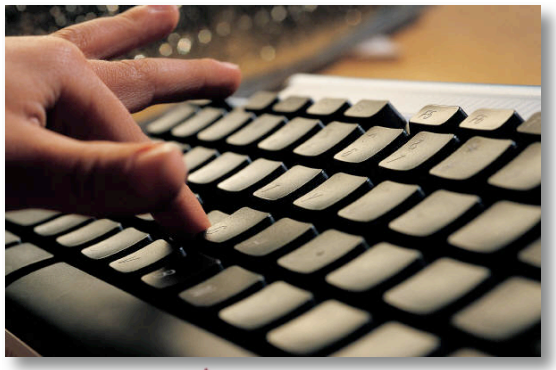
- **NASA Kennedy Space Center**: Constellation Program for next generation ARES Rocket Launch System

SCADA

- **Chemtech/Siemens** in Brazil: since 2006

Homeland Security

- **IDA**: 'Cybercrime Defense' in WAN environment



Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

Erik Hendriks

OpenSplice DDS -- In Summary

OpenSplice™ DDS

Functionality

- Full OMG DDS v1.2 specification coverage
- High Performance, Fault-Tolerant, and Secure Information Backbone
- Wide Technology cohabitation and Integration
- Support for MDE with Power Tools

Performance

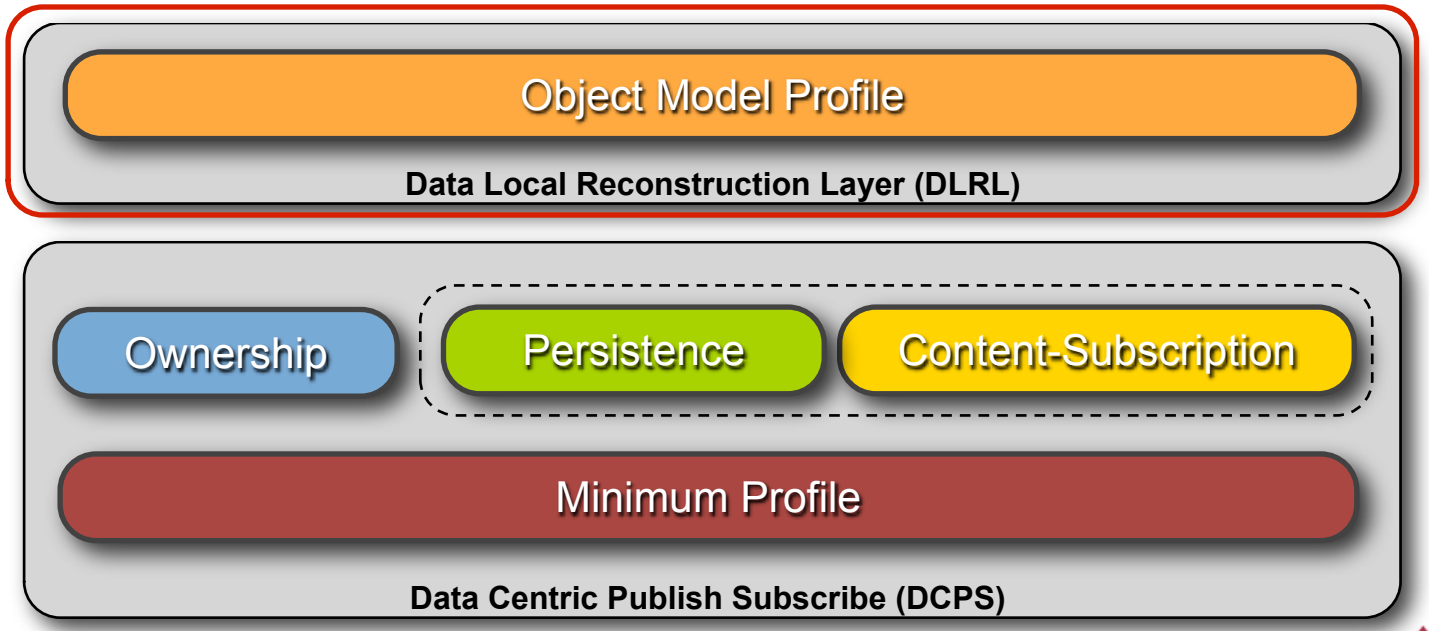
- **Great Scalability** in the number of nodes, applications, and Topics
- **Real-Time Determinism** with priority and latency budget driven network scheduling
- **Fault-Tolerant** architecture, with no single point of failure, and safe isolation between application and critical resources, *e.g.*, network

Pedigree

- **Maturity**. Proven, and fielded in more than 15 navies worldwide
- **Fractal Architecture**. Large-scale, real-time, fault-tolerant, embedded, all in 1 system
- **High Standards of Quality Assurance.** Process/procedures, QA-artefacts and regression testing w.r.t. number of applications as well as computing nodes and topics

Introduction to DLRL

- DLRL offers an Object Oriented (OO) view on the global information model
  - OO-views may be application specific (hence the '*L*' for *Local)*.
  - Allows OO concepts like inheritance, aggregation, composition in your application specific view.
  - Adds "relational awareness" to your application.
- Basically performs OO to Relational Mapping
  - Transport is delegated to the underlying DCPS.
  - DCPS is based on a Relational Model where the 'Tables' are represented by Topics.



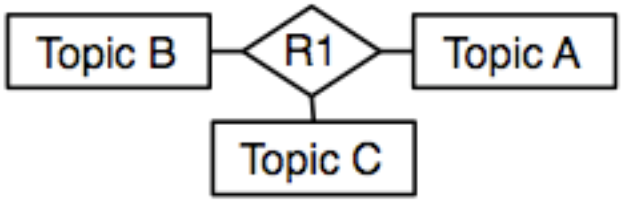
Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- **Information Modeling for DLRL**
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

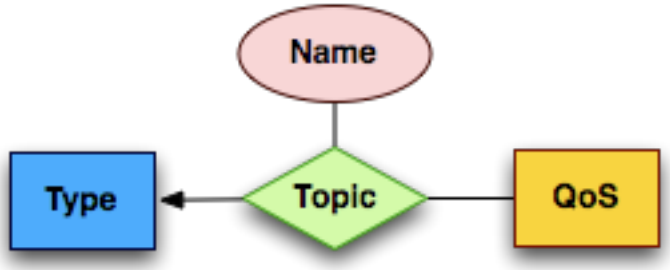
Erik Hendriks

Information Modelling in DCPS

- **Modeling.** As in a Relational DB, a DCPS information model can be represented by means of Entity Relationship (ER) diagrams



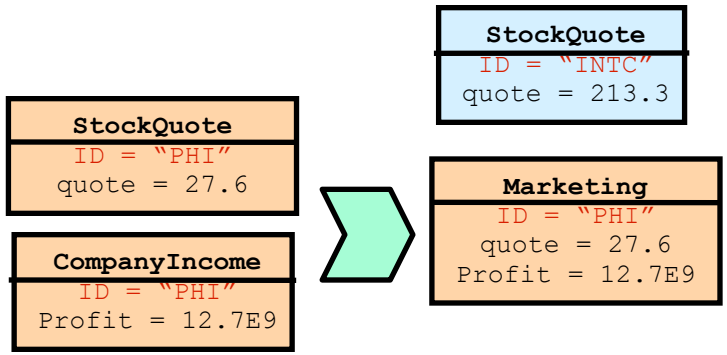
- **Topics.** The entities, represented by means of Topics, are in turns an association between a data **type** and a set of **QoS** and identified by a unique name (like tables in an RDBMS).



- **Data Types.** The data type associated to a Topic must be a structured type expressed in IDL.

```
struct StockQuote {
   string ID;
   float  quote;
};
#pragma keylist StockQuote ID
```

- **Instances.** Key values in a datatype uniquely identify an instance (like rows in table).



- **Correlation.** SQL Expressions can be used to correlate information by means of key values.

Example of a typical DCPS data model

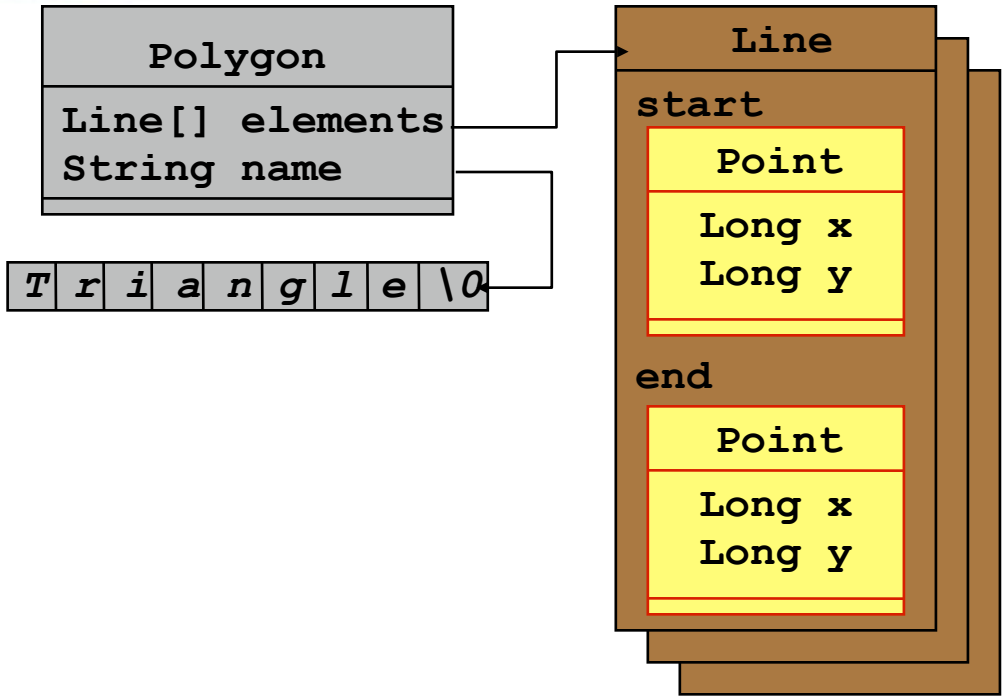
dcps_figures.idl

```
struct Point {
    long x;
    long y;
};

struct Line {
    Point start;
    Point end;
};

typedef
  sequence<Line> LineSeq;

Struct Polygon {
    LineSeq elements;
    string name;
};
```



Limitations of such a data-model:

- No encapsulation of information.
- No notion of instances.
  - To distinguish between instances, key-fields need to be added which must be managed by hand.
- Limited automatic correlation capabilities.
  - Topics can only be joined for matching key-fields.

DLRL: Responding to the OO-trend

Modern applications are more and more designed in UML using the Object Oriented facilities it provides.

- More and more demand for support of OO-information models.

DLRL facilitates the use of IDL valuetypes for these cases, and offers the following OO-constructs:

- **Automatic Instance management.**
- **Encapsulation.**
  - Attributes are only accessible through dedicated getter/setter operations.
- **(Local) Operations.**
  - Besides getters/setters, all other kind of manipulations can be done using custom operations.
- **Inheritance.**
  - Only single inheritance between DLRL objects.
- **(Navigable) Relationships.**
  - Mono relationships.
  - Multi Relationships (Set, Map, List): requires some extra annotations in XML.

Example of a typical DLRL object model

dlrl_company.idl

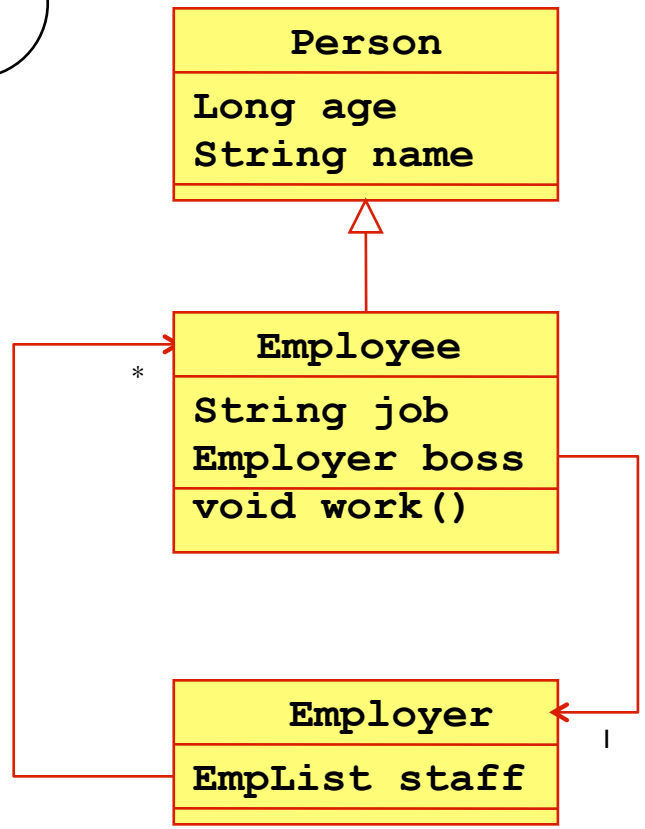
```
valuetype Employer;

valuetype Person : DDS::ObjectRoot
{
    public long age;
    public string name;
};

valuetype Employee : Person {
    public string job;
    public Employer boss;
    void work();
};

valuetype EmpList; // List<Employee>

valuetype Employer : DDS::ObjectRoot
{
    public EmpList staff;
};
```



Mapping a DLRL Model onto a DCPS Model

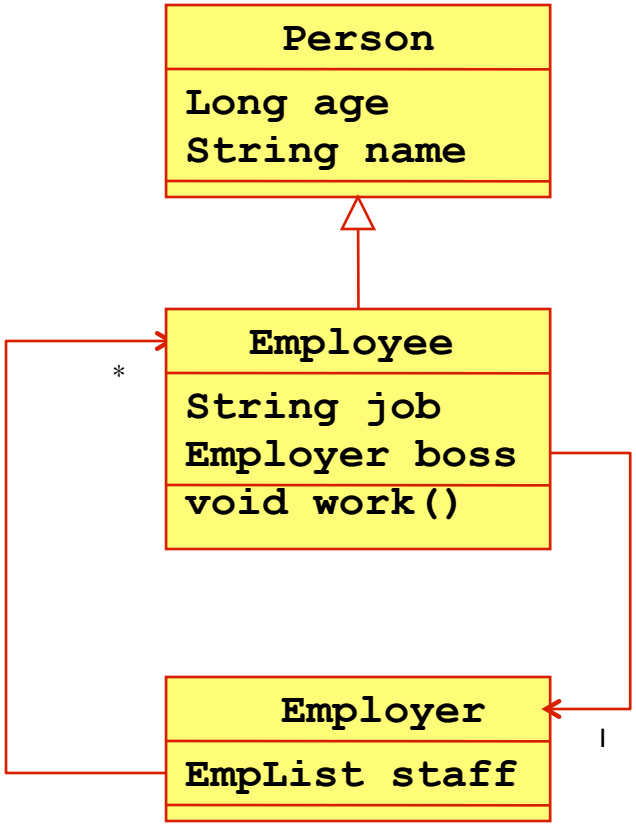
DLRL offers a local Object view on a global Topic model.

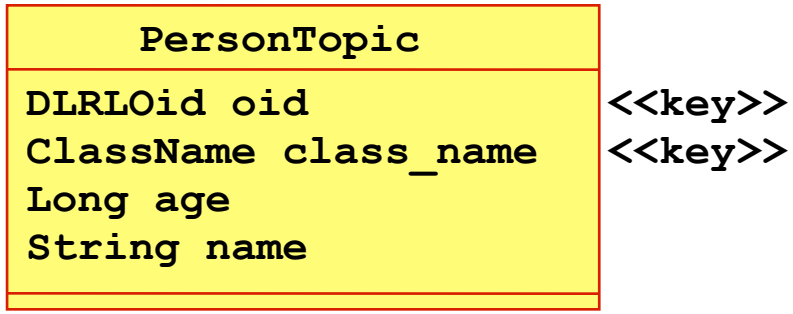
- Each object will need to be mapped onto a corresponding Topic.

Mapping is performed in a separate XML file:

- The identity of the Object is mapped onto a number of key-fields.
- Each Object attribute is mapped onto a corresponding Topic field.
- Each mono Relationship is mapped onto a number of foreign/ shared key fields.
  - The number of keys that models the relationship is based on the number of keys that represents the identity of the targeted topic.
- Each multi Relationship is mapped onto an external Topic.
  - External topics are required to account for the extra dimension introduced by the Collection.
  - The XML element used to define this mapping also annotates the IDL model with the Collection type that is to be used.

A mapping example

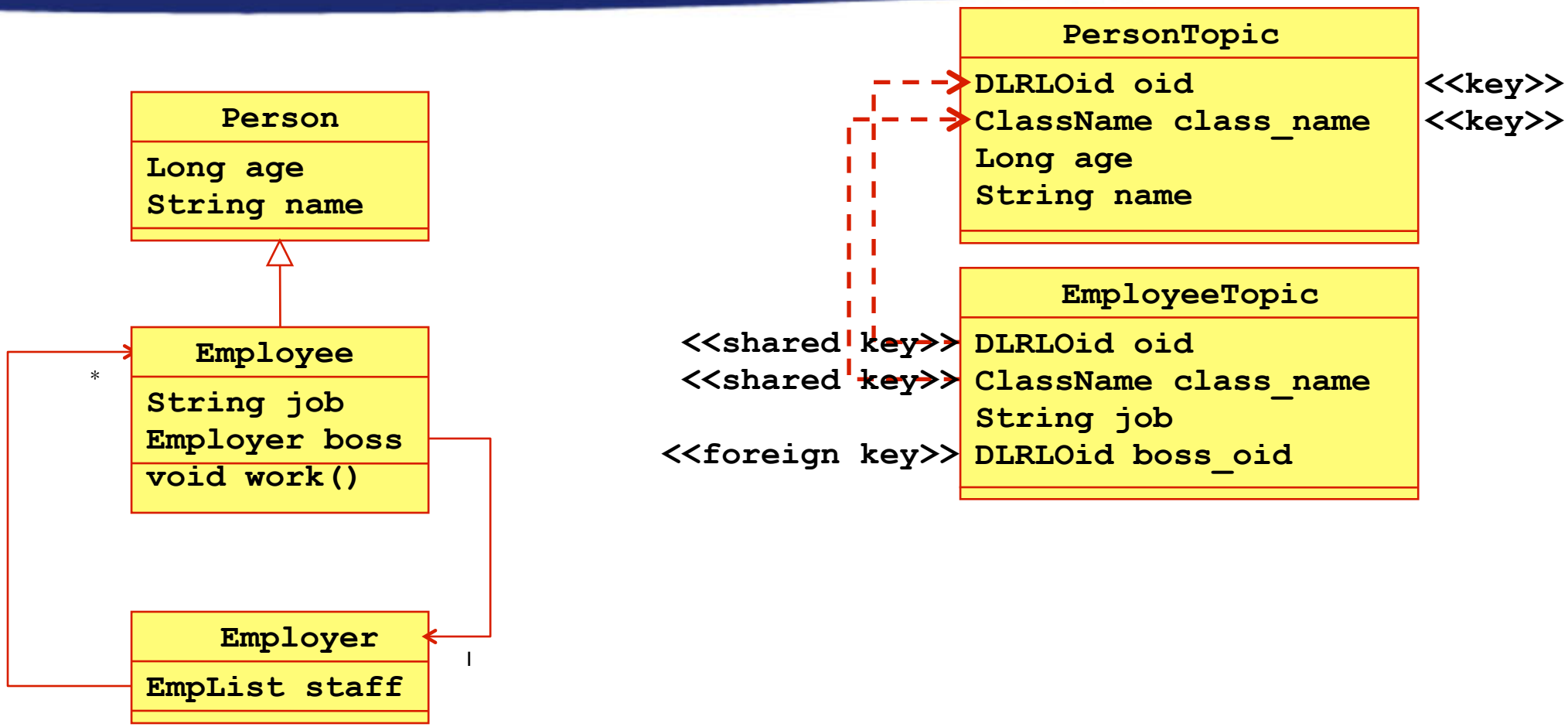




DLRL Object Model

DCPS Topic Model

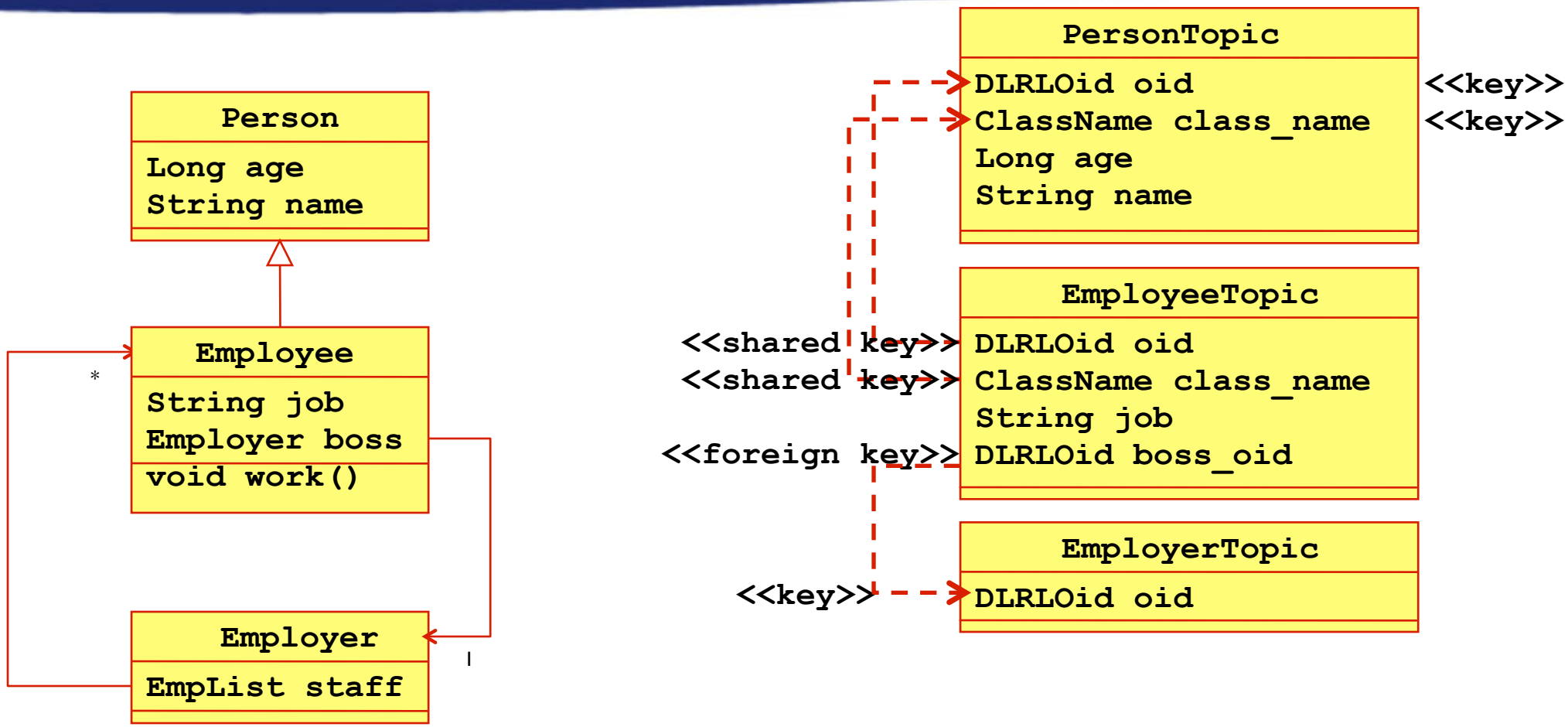
A mapping example



DLRL Object Model

DCPS Topic Model

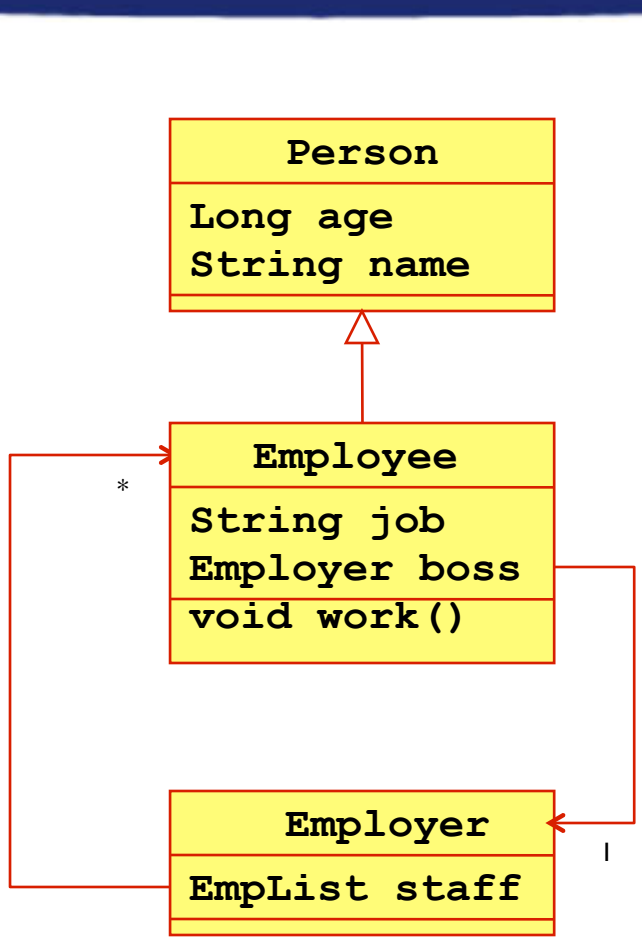
A mapping example



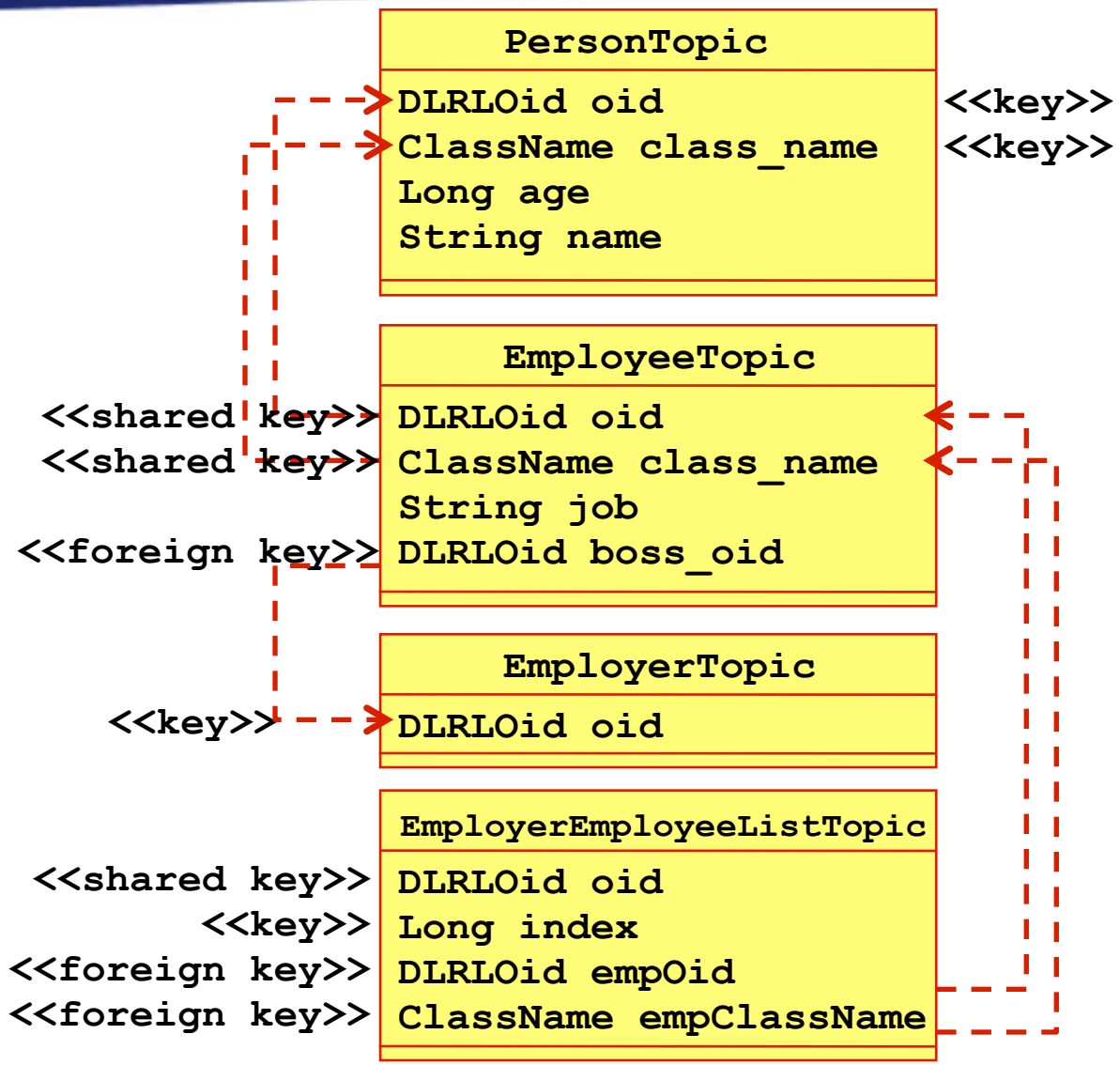
DLRL Object Model

DCPS Topic Model

A mapping example



DLRL Object Model

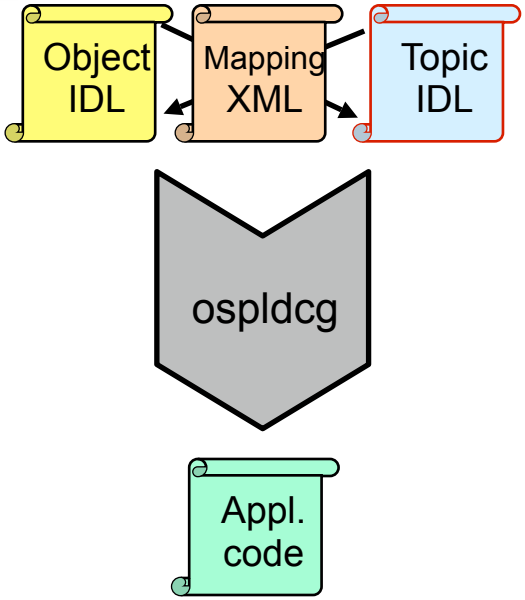


DCPS Topic Model

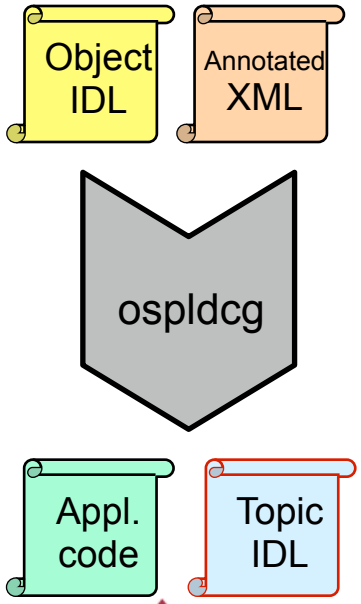
Modelling approaches.

Object Models always need to be mapped onto an underlying Topic Model.

- This can be an existing Topic Model:
  - Mapping needs to be performed manually.
  - Different Object Models can match the same Topic Model
  - Useful in hybrid systems



- A matching Topic Model can be generated for a given (annotated) Object Model.
  - Result is a dedicated Topic Model that is probably only usable in the context of this specific Object Model.



Agenda

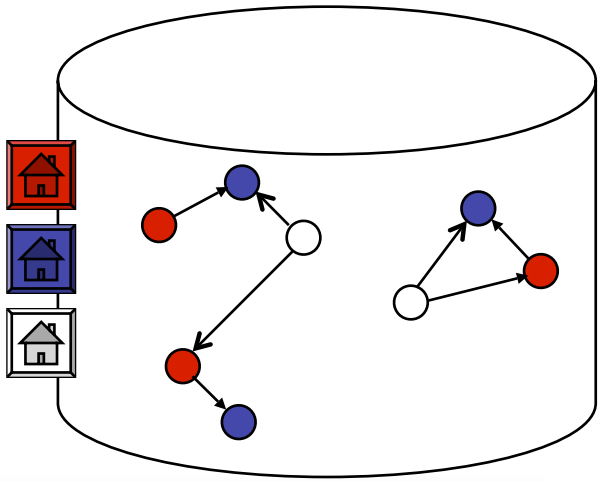
- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

Erik Hendriks

DLRL General Overview

The mechanism at the foundation of DLRL is a managed Object Cache:

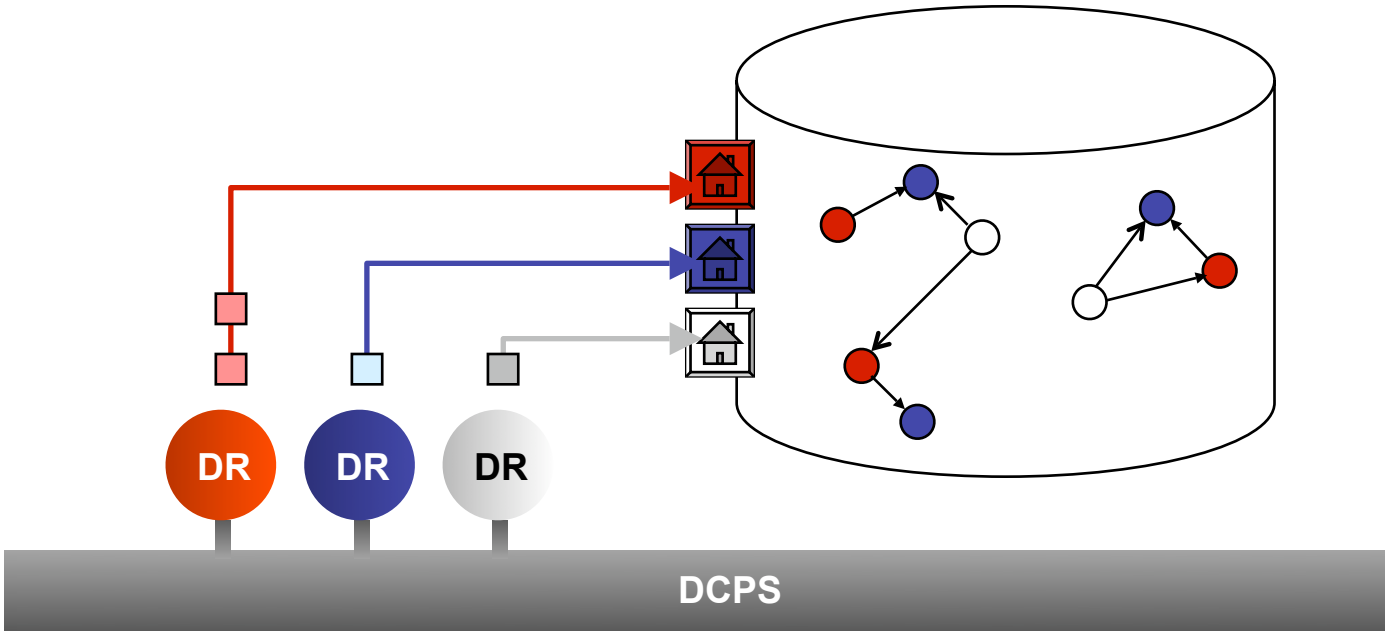
- A DLRL Cache can be populated by different types of DLRL Objects.
- Each object type has its own manager called an ObjectHome.
  - They can inform the application about object creation/modification/deletion.
- Objects may contain navigable relationships to other objects.
- A DLRL Object type may inherit from 1 other DLRL Object type.



DLRL at work

DLRL processes information in so called ‘update rounds’:

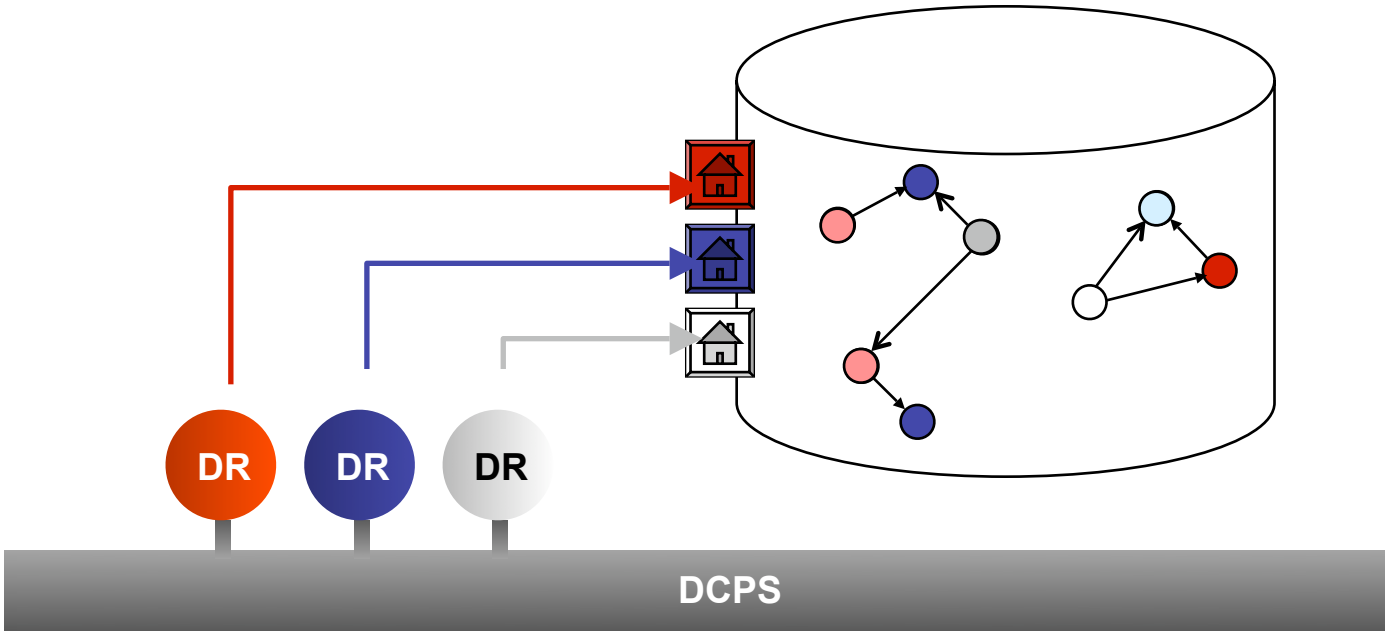
- All ObjectHomes read incoming information from DCPS and refresh their corresponding objects in the Cache accordingly.
  - In contrast to DCPS, where each update is a new sample, a DLRL Object is allocated once and recycled on subsequent updates. It always contains the latest available object state.
- DLRL offers both a push and a pull mode.
  - In push mode, updates are applied automatically and Listeners will notify the application about this.
  - In pull mode, the application can determine the start of each update round manually.



DLRL at work

DLRL processes information in so called ‘update rounds’:

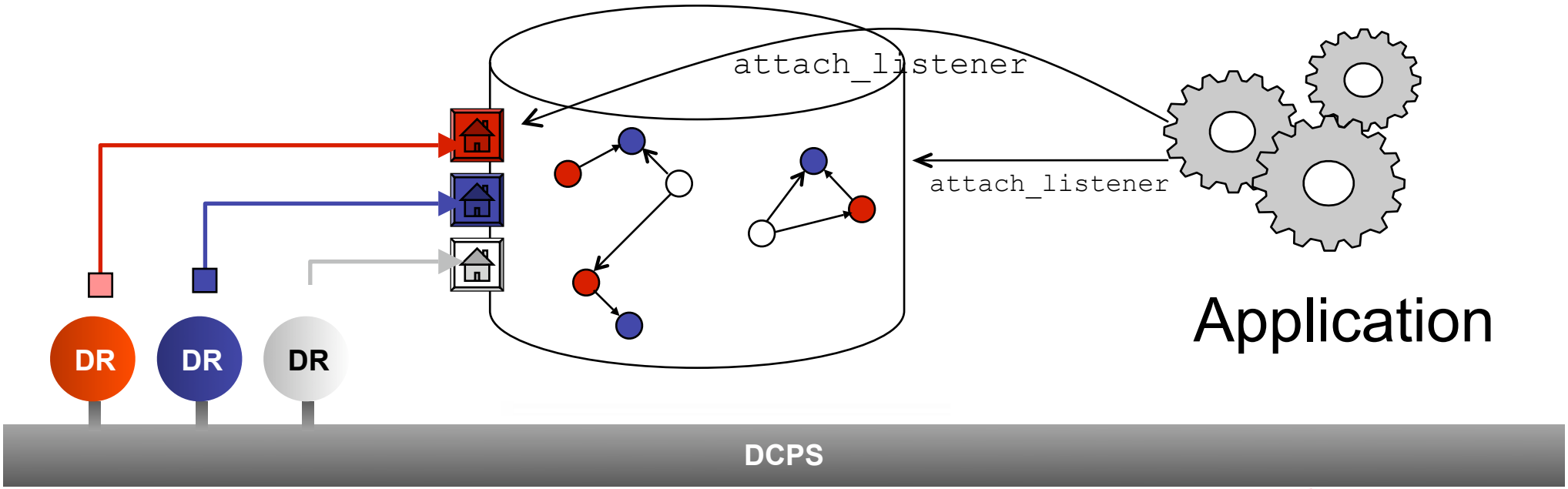
- All ObjectHomes read incoming information from DCPS and refresh their corresponding objects in the Cache accordingly.
  - In contrast to DCPS, where each update is a new sample, a DLRL Object is allocated once and recycled on subsequent updates. It always contains the latest available object state.
- DLRL offers both a push and a pull mode.
  - In push mode, updates are applied automatically and Listeners will notify the application about this.
  - In pull mode, the application can determine the start of each update round manually.



Notification patterns

DLRL offers two ways to get notified of incoming information:

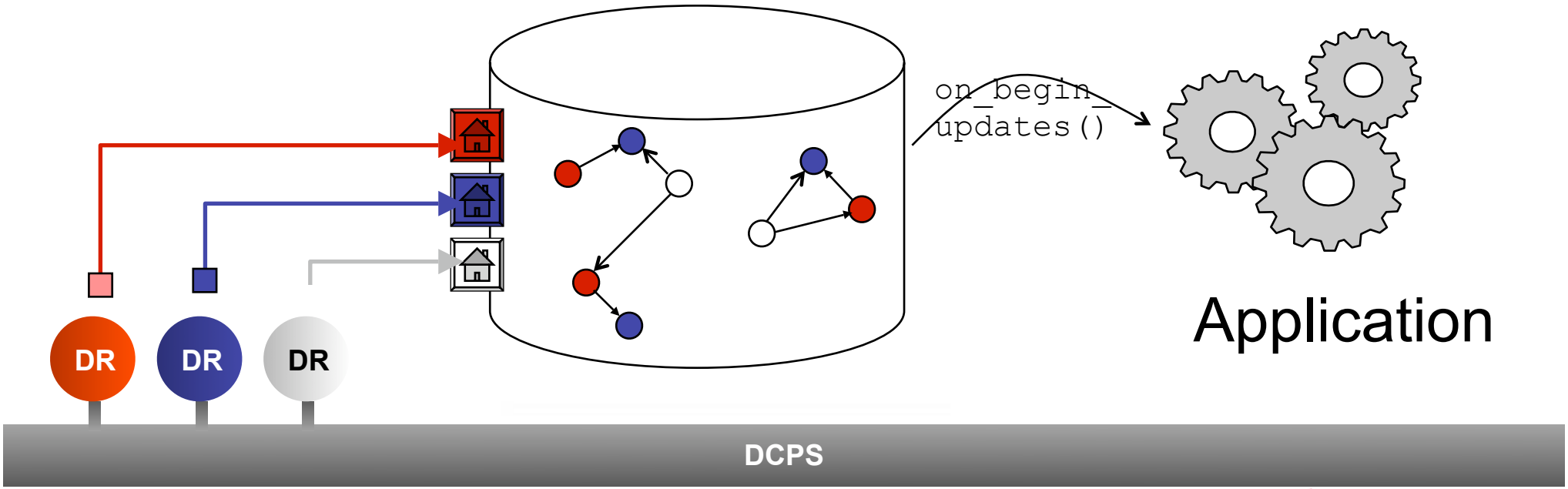
- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Notification patterns

DLRL offers two ways to get notified of incoming information:

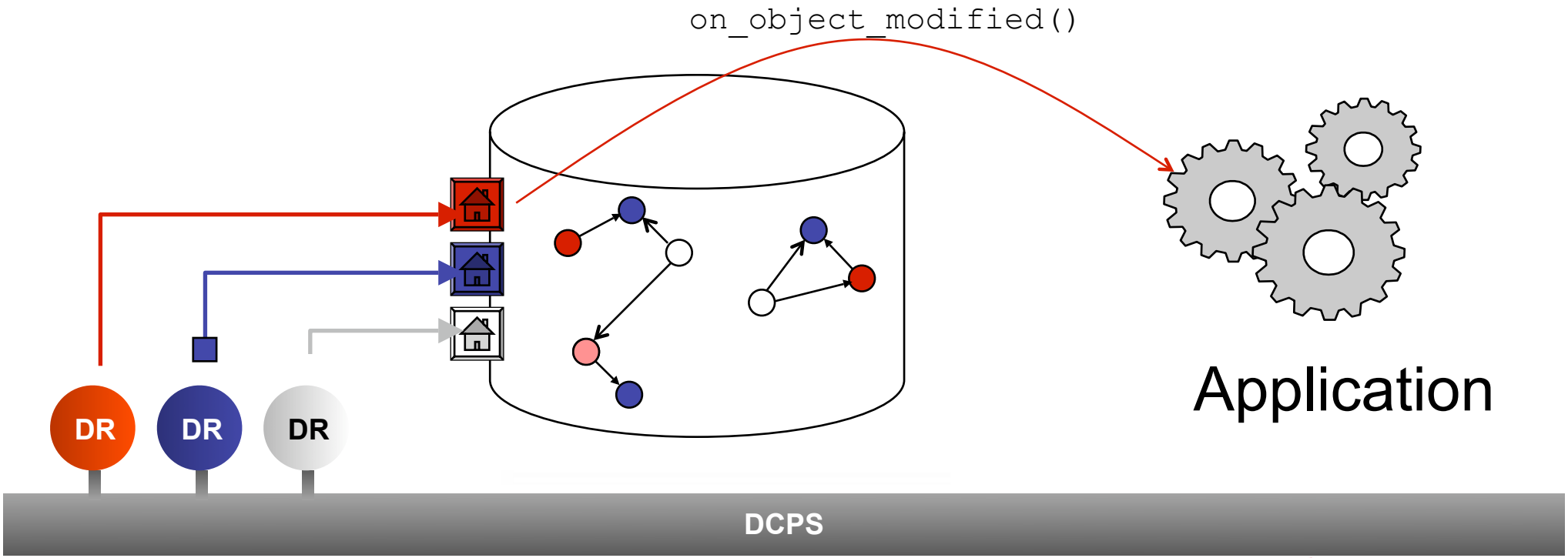
- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Notification patterns

DLRL offers two ways to get notified of incoming information:

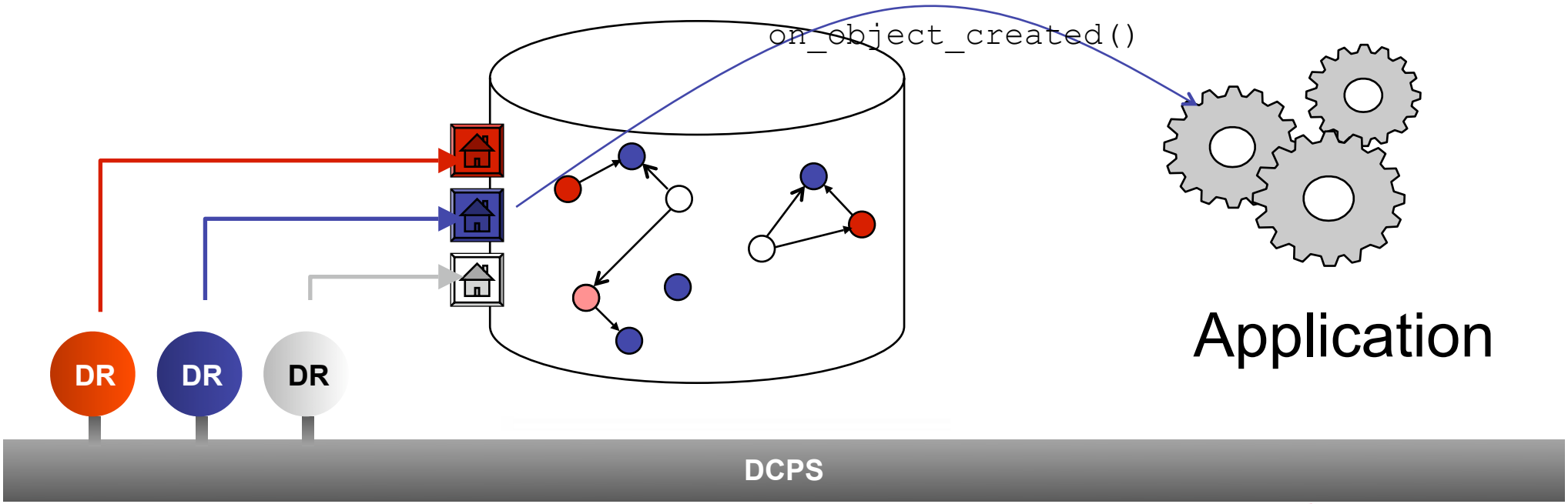
- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Notification patterns

DLRL offers two ways to get notified of incoming information:

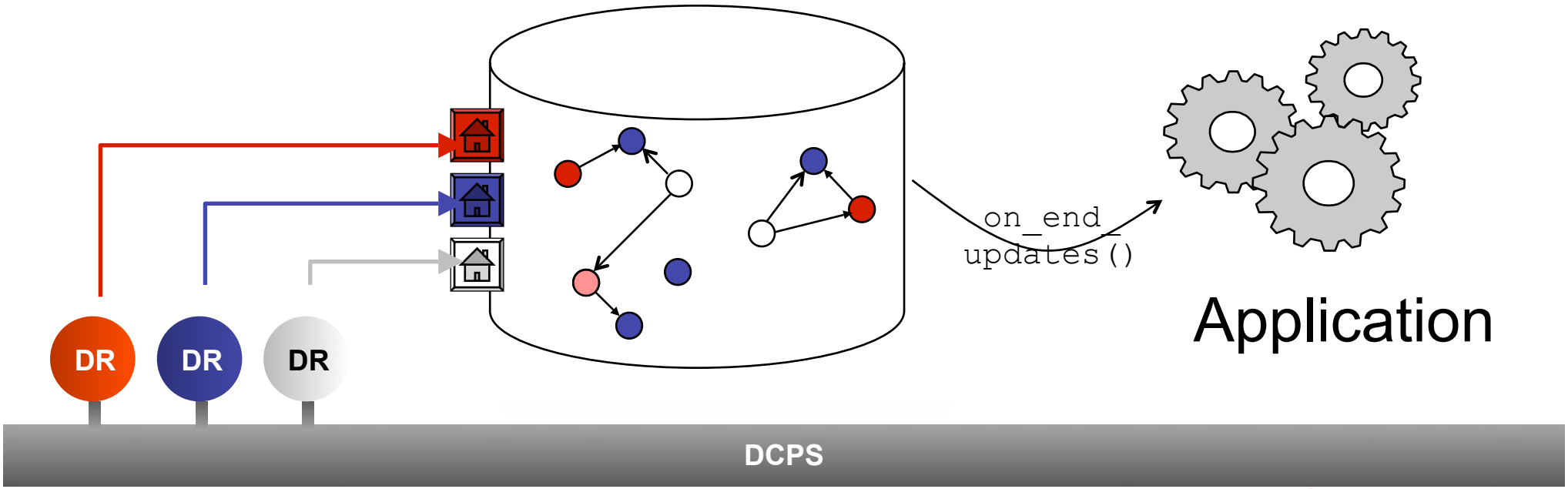
- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Notification patterns

DLRL offers two ways to get notified of incoming information:

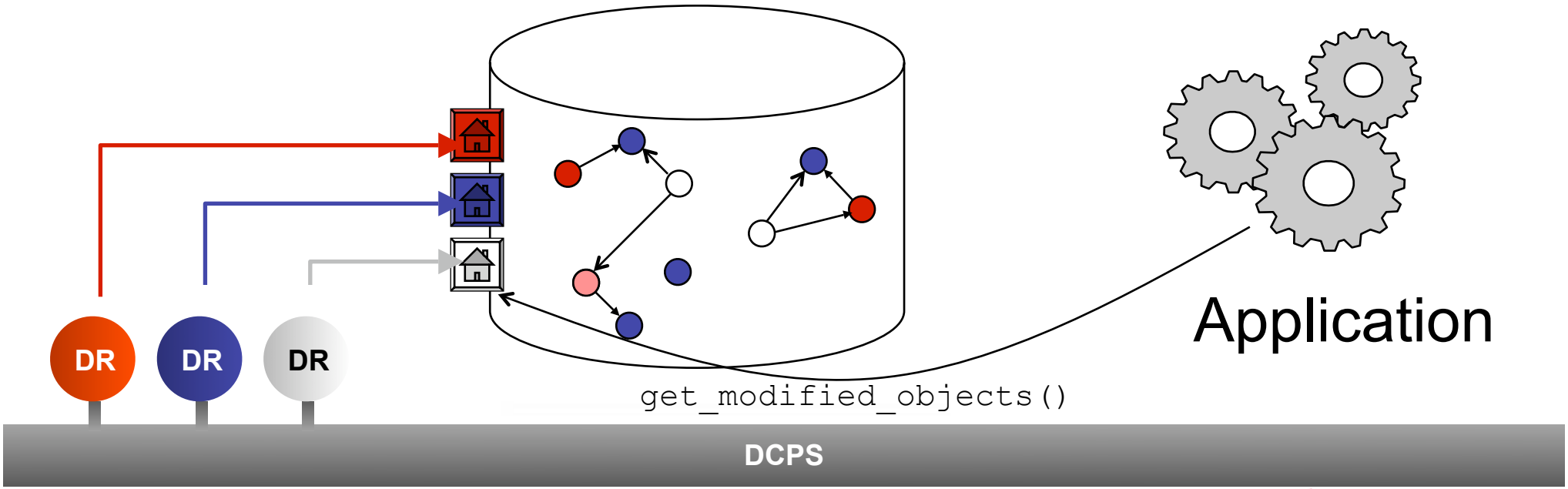
- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Notification patterns

DLRL offers two ways to get notified of incoming information:

- Listeners can be triggered for each modification of an object's state.
  - Listeners registered to the Cache indicate the start and end of each update round.
  - Listeners registered to the ObjectHome pass the modifications back as callback arguments.
  - With a simple mechanism this can be translated into callbacks for Listeners on individual objects.
- It is possible to get a separate list of all objects that have been created/ modified/deleted in the current update round.



Using DLRL to auto-correlate information

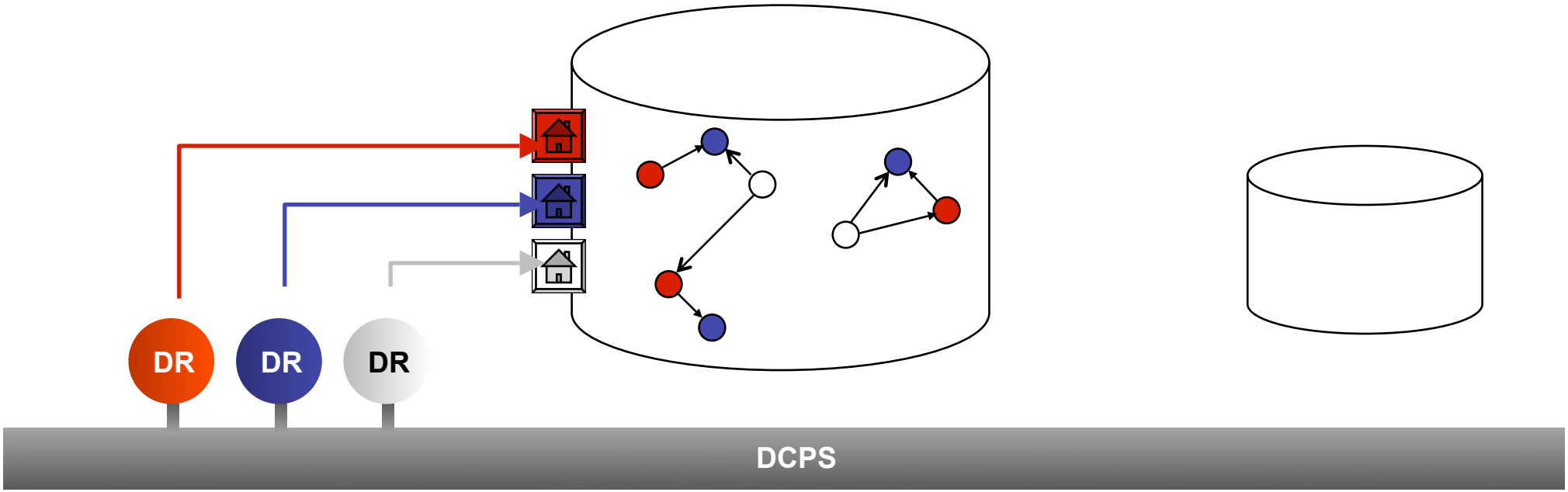
Relationships in DLRL can very easily correlate information for you:

- By defining a relationship in your object model, you can easily navigate from one object to the other.
  - DLRL will translate (foreign) keys on its incoming topics to objects.
  - Minimal overhead when resolving relationships (depending on the number and kind of keyfields and the number of potential target objects).
  - Tailor your object model in such a way that only the relationships you actually plan to navigate are resolved. This may save you some overall overhead.
- Advantages of relationships when compared to DCPS aggregation:
  - DCPS Aggregation can only be performed in case of shared keys.
  - Better visibility of information. (Aggregation only shows a combined set when it is complete).
  - Granularity of notifications is more fine grained. (combined type vs. a dedicated atom).
  - Aggregation is only available on the Reader side, relationships are also available on the Writer side.
  - DLRL allows you to choose lazy instantiation for related objects, removing processing and memory overhead of information that is not accessed by the application.

CacheAccess: Examining objects in isolation

Some applications want to be able to store temporal ‘snapshots’:

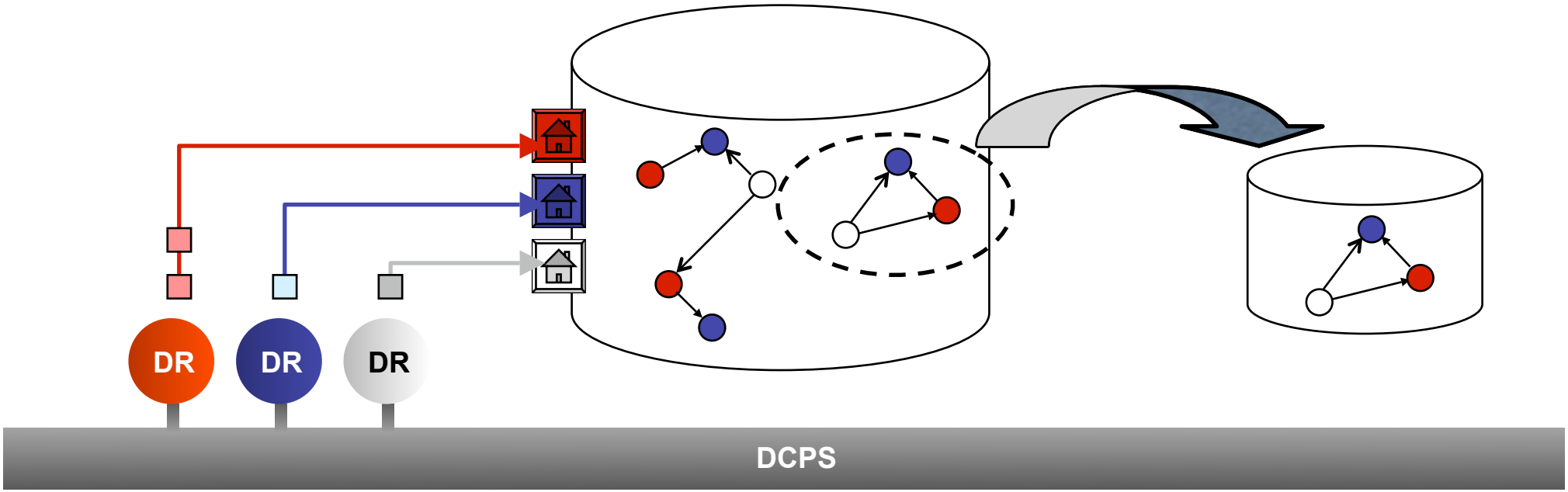
- A CacheAccess can be used to contain a temporal graph of objects.
- Objects must physically be cloned from Cache to CacheAccess.
- A CacheAccesses is not automatically kept in sync with the main Cache.
- A ‘refresh’ operation can be used to resync the contents of CacheAccess with the contents of the main Cache .



CacheAccess: Examining objects in isolation

Some applications want to be able to store temporal ‘snapshots’:

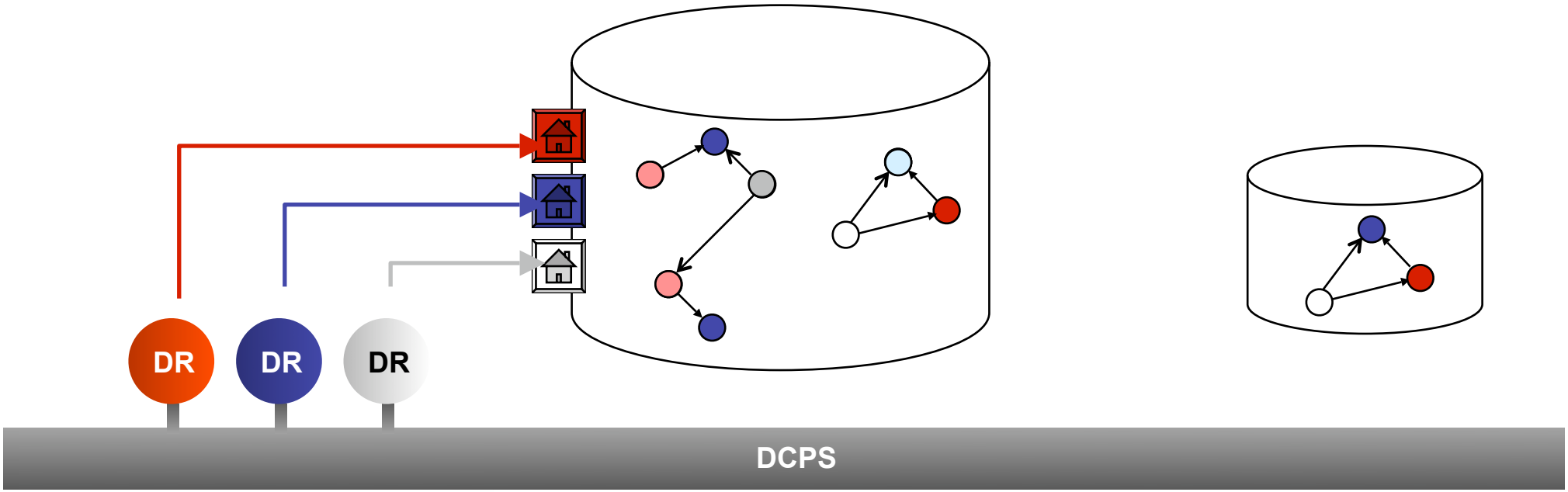
- A CacheAccess can be used to contain a temporal graph of objects.
- Objects must physically be cloned from Cache to CacheAccess.
- A CacheAccesses is not automatically kept in sync with the main Cache.
- A ‘refresh’ operation can be used to resync the contents of CacheAccess with the contents of the main Cache .



CacheAccess: Examining objects in isolation

Some applications want to be able to store temporal ‘snapshots’:

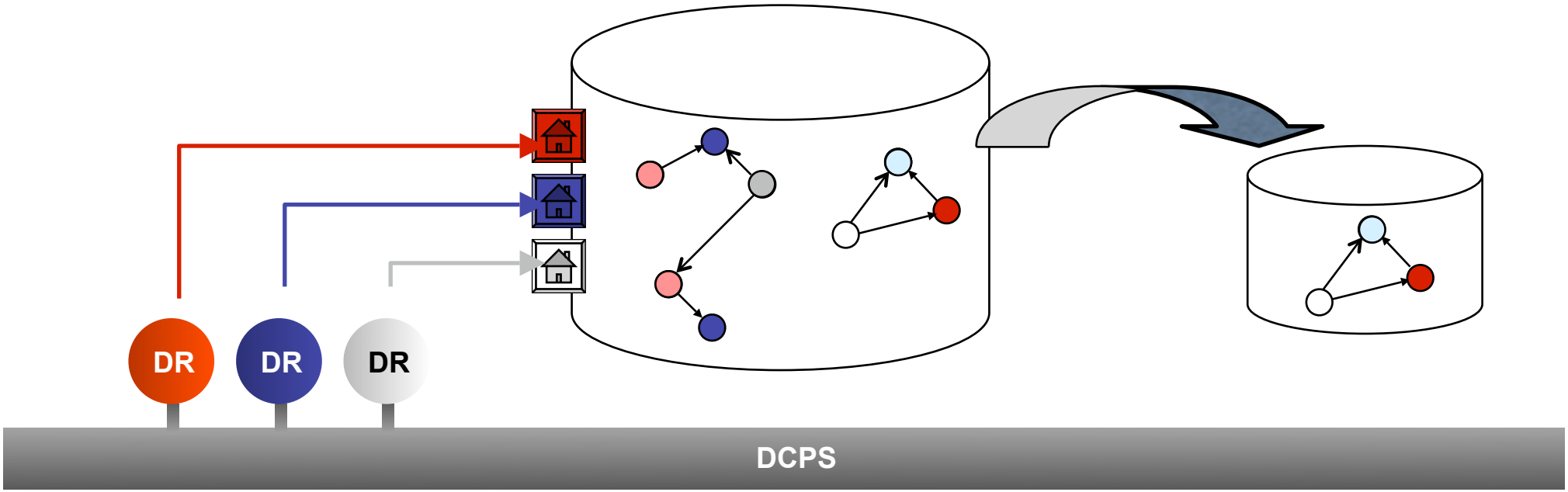
- A CacheAccess can be used to contain a temporal graph of objects.
- Objects must physically be cloned from Cache to CacheAccess.
- A CacheAccesses is not automatically kept in sync with the main Cache.
- A ‘refresh’ operation can be used to resync the contents of CacheAccess with the contents of the main Cache .



CacheAccess: Examining objects in isolation

Some applications want to be able to store temporal ‘snapshots’:

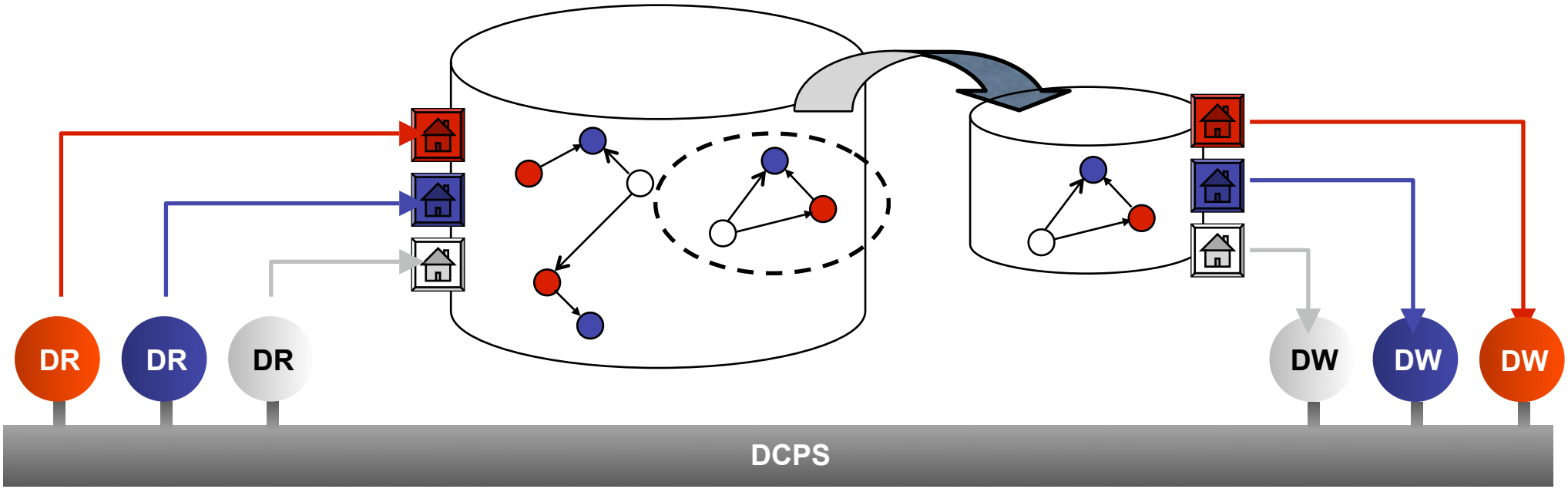
- A CacheAccess can be used to contain a temporal graph of objects.
- Objects must physically be cloned from Cache to CacheAccess.
- A CacheAccesses is not automatically kept in sync with the main Cache.
- A ‘refresh’ operation can be used to resync the contents of CacheAccess with the contents of the main Cache .



CacheAccess: modifying and creating objects

Some applications want to be able to modify or create certain objects:

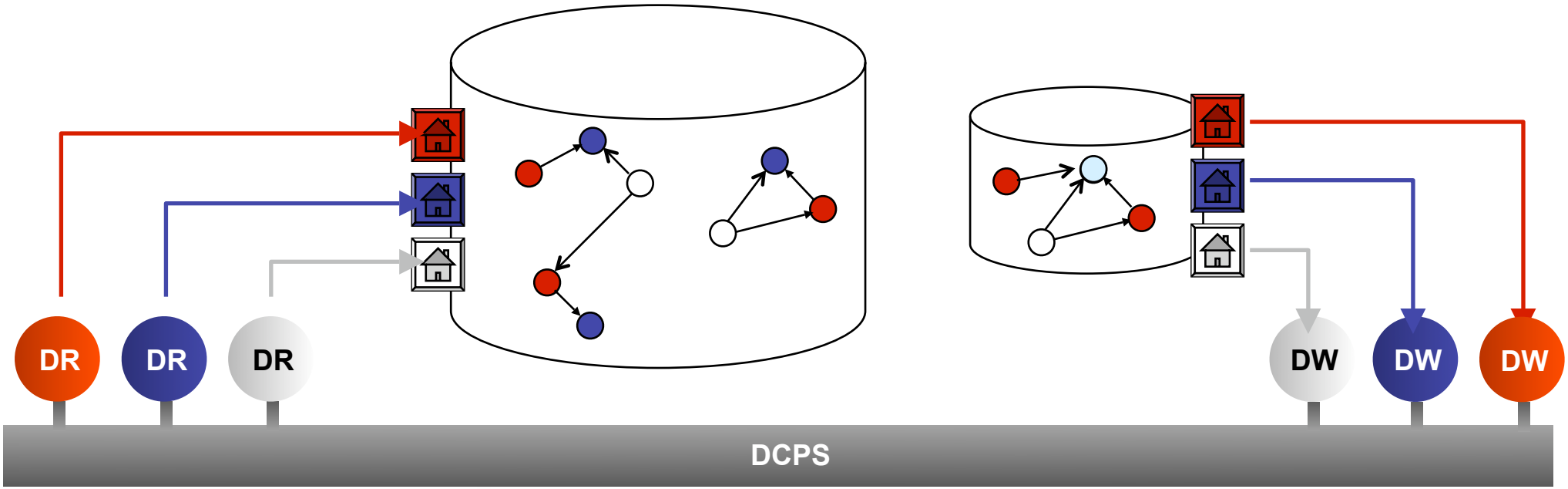
- An initial set of Objects may be cloned into a writeable CacheAccess.
- Available objects may then be modified locally.
- New objects can be created in the CacheAccess as well.
- The ‘write’ operation instructs the ObjectHomes to write any modifications into the system.



CacheAccess: modifying and creating objects

Some applications want to be able to modify or create certain objects:

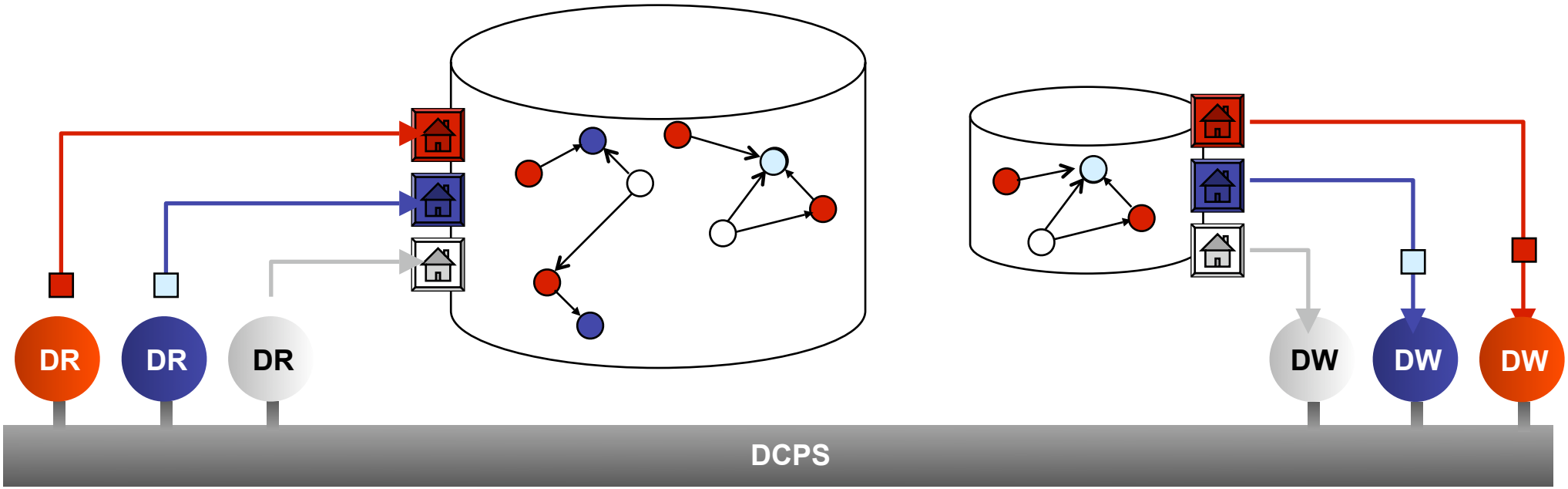
- An initial set of Objects may be cloned into a writeable CacheAccess.
- Available objects may then be modified locally.
- New objects can be created in the CacheAccess as well.
- The ‘write’ operation instructs the ObjectHomes to write any modifications into the system.



CacheAccess: modifying and creating objects

Some applications want to be able to modify or create certain objects:

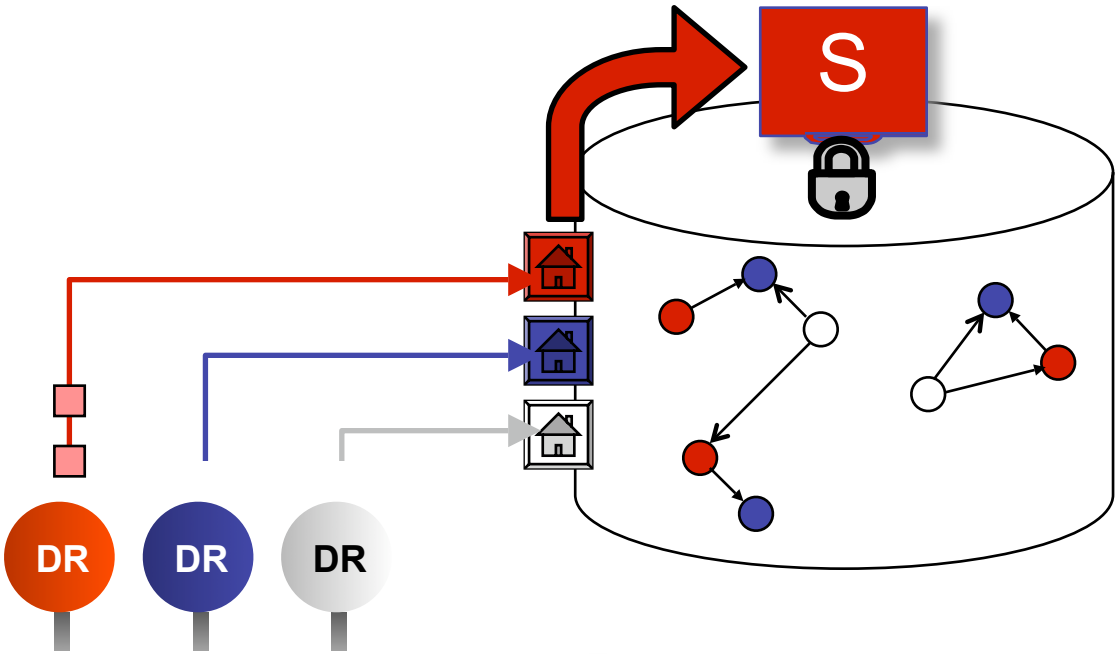
- An initial set of Objects may be cloned into a writeable CacheAccess.
- Available objects may then be modified locally.
- New objects can be created in the CacheAccess as well.
- The ‘write’ operation instructs the ObjectHomes to write any modifications into the system.

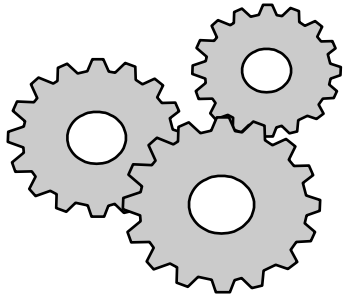


Using Selections to manage subsets

DLRL offers a Selections mechanism to keep track of subsets of information:

- Selections are created and managed by the ObjectHomes.
- A Criterion plugged into a Selection determines the boundaries of a subset:
  - A QueryCriterion determines boundaries based on an SQL statement.
  - A FilterCriterion determines boundaries based on user-defined callback filters.
- Selections can notify the application when objects enter and leave it.



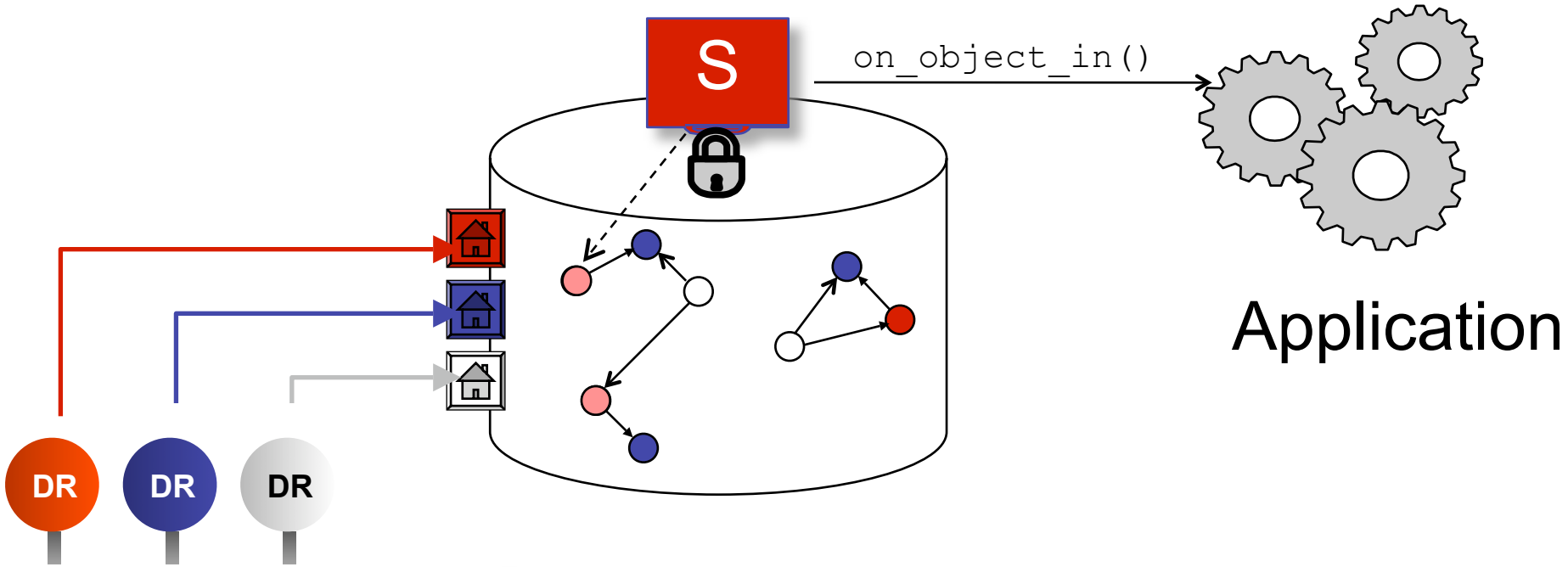


Application

Using Selections to manage subsets

DLRL offers a Selections mechanism to keep track of subsets of information:

- Selections are created and managed by the ObjectHomes.
- A Criterion plugged into a Selection determines the boundaries of a subset:
  - A QueryCriterion determines boundaries based on an SQL statement.
  - A FilterCriterion determines boundaries based on user-defined callback filters.
- Selections can notify the application when objects enter and leave it.



Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- **OpenSplice DDS-DLRL Benefits**
- What's Next
- Concluding Remarks

Erik Hendriks

Benefits of the OpenSplice-DDS DLRL

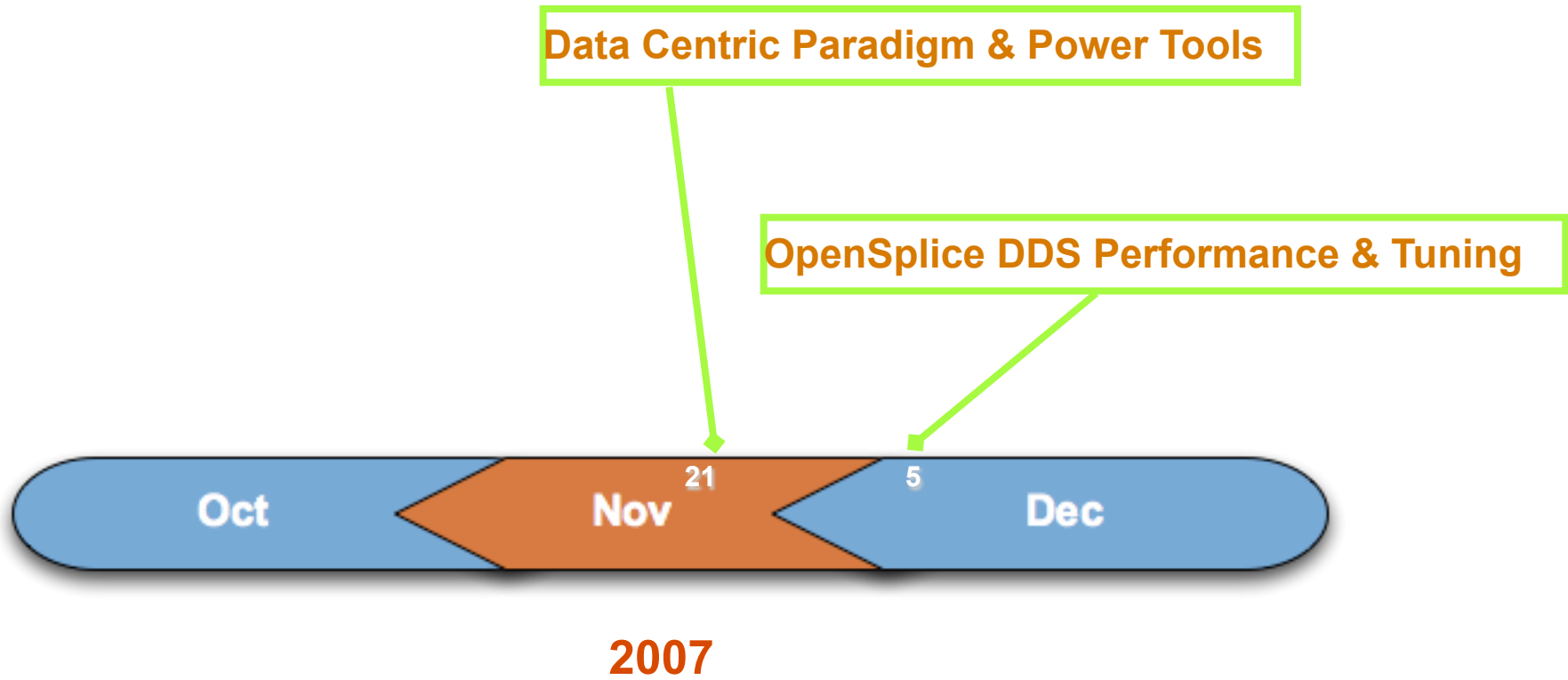
- The DLRL API is compliant with the latest DDS V1.2 specification.
- It is available for both C++ and Java.
  - Allows the use in mixed language environments.
- It is fully integrated with the DCPS layer
  - Tight integration allows smart 'short-cuts' and removes redundant administration.
  - This allows for better performance and a smaller memory footprint.
  - DCPS functionality is almost fully orthogonal: all DCPS QoSPolicies function as expected.
  - Creating hybrid systems (even mixing DCPS and DLRL applications on a single node) is very well possible and comes with no extra overhead.
- OpenSplice DDS PowerTools help you focus on your domain knowledge instead of on the DLRL API itself.
  - Information modeller helps you design your Object Model and to map it on the Topics.
  - Application designer helps you graphically compose your application from default building blocks and generates most of the code for you.

Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

Dr. Angelo Corsaro

Upcoming Webinars



Registration: http://www.prismtech.com/section-item.asp?id=731&sid=29&sid2=15&sid3=289

Agenda

- OpenSplice DDS Overview
- Introduction to DLRL
- Information Modeling for DLRL
- DLRL Overview
- OpenSplice DDS-DLRL Benefits
- What's Next
- Concluding Remarks

Dr. Angelo Corsaro

Concluding Remarks

Information Modeling and Management

- The DLRL provides a more powerful information modeling and management support than DCPS, as it allows to deal with subtypes, relationships, etc.
- The mapping between DLRL and DCPS information models allow to easily migrate from DCPS-based to DLRL-based solutions

Performance

- The OpenSplice DDS-DLRL layer is designed to minimize performance overhead over the DCPS layer
- Lazy instantiation, and finer control over the information model, allow the application for an even greater control over resources than DCPS

Open Architecture

- OpenSplice DDS-DLRL is the only implementation in the world which is compliant with the latest OMG DDS v1.2 standard

OpenSplice DDS is the best solution available on the market for solving your data distribution problems!

Contact Us

OpenSplice™ DDS

- OpenSpliceDDS Resource Center
  - http://www.prismtech.com/opensplice-dds/
- Evaluate OpenSplice DDS
- Training and Consulting
  - sales@prismtech.com
- OMG DDS Information
  - http://www.dds-forum.org/
  - http://portals.omg.org/dds/

Thank You!