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

PrismTech
Powering Netcentricity

Pattern Oriented DDS Architectures

OpenSplice | DDS

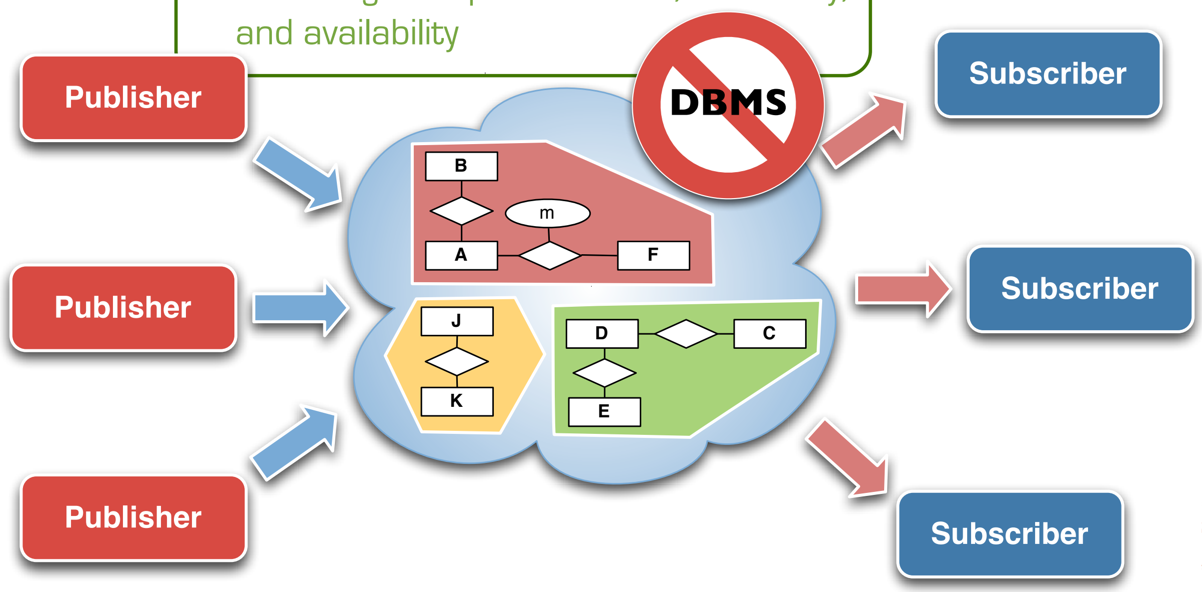
Delivering Performance, Openness, and Freedom

DDS Refresher

Data-Centric Pub/Sub

- Data-Centric Features are **built-in** and **don't rely on an external DBMS**
- Providing thus performance, scalability, and availability

- Distributed Relational Data Model
- Local Queries
- Continuous Queries / Content Based Subscriptions
- Windows (Data History)
- Object/Relational Mapping
- Support for a subset of SQL-92

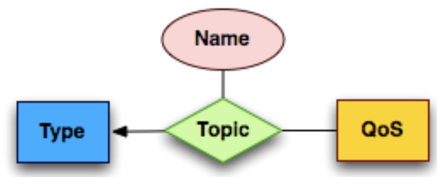


DDS allows you to deal with data cubes which can be flexibly sliced and diced

Topics and Data-Centric Pub/Sub

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

Topic



Topic Type

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

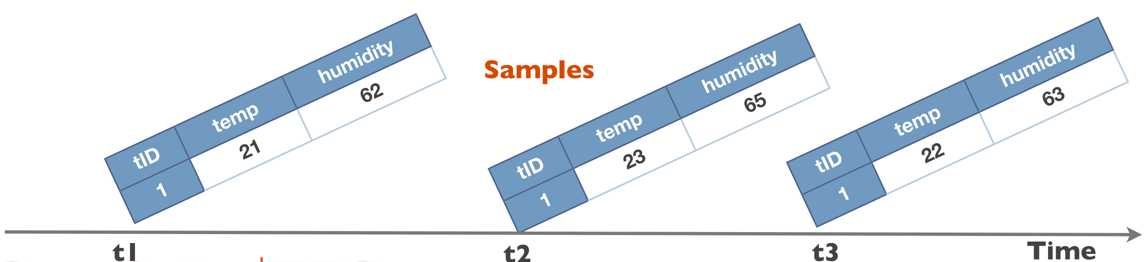
Instances

TempSensor

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

SELECT * FROM TempSensor t WHERE t.temp > 25

tID	temp	humidity
2	27	78
3	25.5	72.3

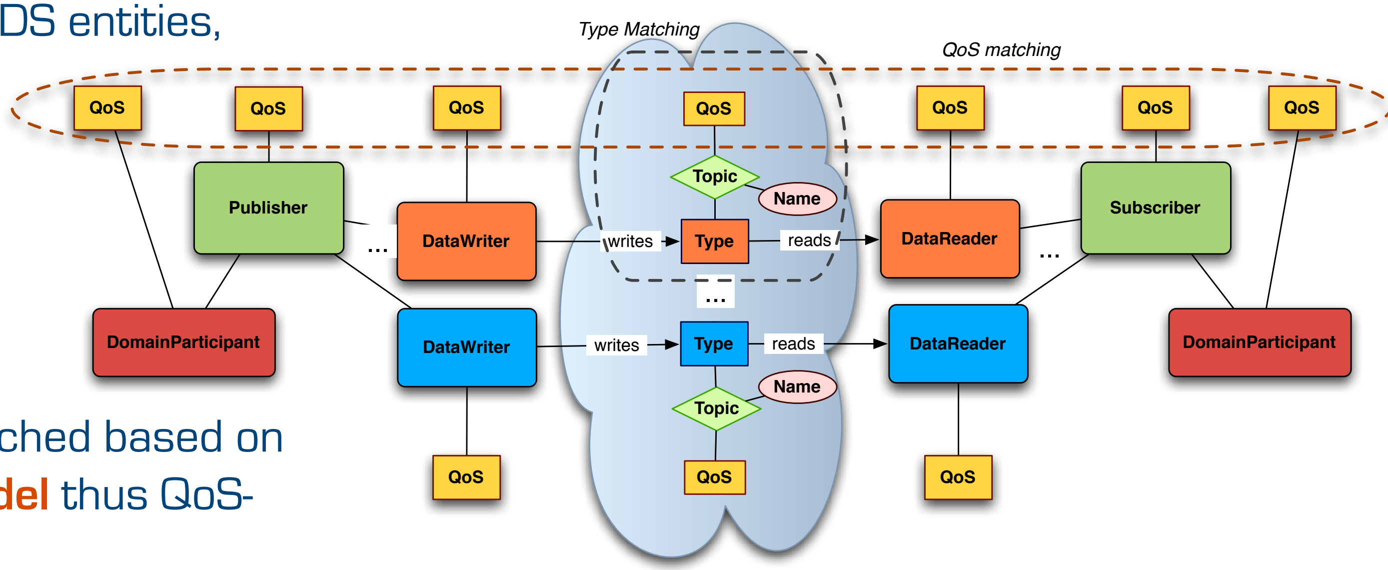


OpenSplice DDS

PrismTech

QoS Model

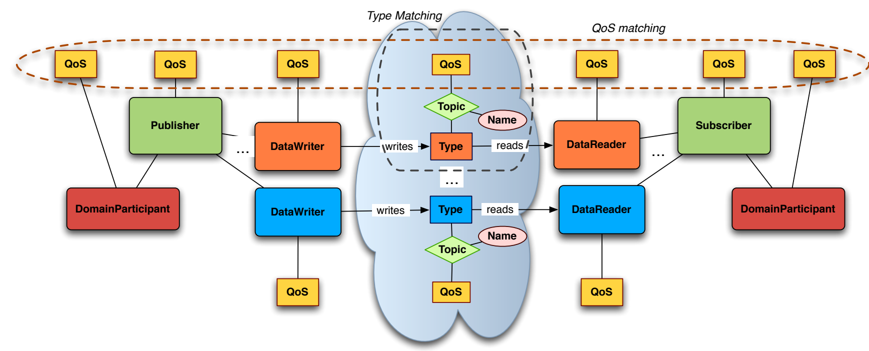
- QoS-Policies are used to control relevant properties of OpenSplice DDS entities, such as:
  - Temporal Properties
  - Priority
  - Durability
  - Availability
  - ...
- Some QoS-Policies are matched based on a **Request vs. Offered Model** thus QoS-enforcement



- Publications and Subscriptions match only if the declared vs. requested QoS are compatible
  - e.g., it is not possible to match a publisher which delivers data unreliably with a subscriber which requires reliability

Sample QoS Policies

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



- Rich set of QoS allow to configure several different aspects of data availability, delivery and timeliness
- QoS can be used to control and optimize network as well as computing resource

OpenSplice DDS

PrismTech

References

Useful Background Info

- OpenSplice DDS Crash Course
  - http://www.opensplice.com/section-item.asp?snum=4&sid=262
- Event Driven Data Centric Architectures Unvelied
  - http://www.opensplice.com/section-item.asp?snum=4&sid=224
- The YouTube OpenSplice TV
  - http://www.youtube.com/opensplicetube

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Architectural Patterns

Key DDS Architectural Patterns

- The DDS implements key architectural patterns which need to be understood in order to properly design DDS-based Architectures:
  - Lingua Franca
  - Shared Global Data Space

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Lingua Franca

Lingua Franca

Problem

- Designing large-scale interoperable distributed systems, and system of systems, is a very complex engineering endeavor
- Functional and Object-Oriented decomposition have proven to be powerful methodologies, but yet, often lead to tightly coupled systems (see 4W)
- The key challenge lies in the inherent fragility of interfaces which tend to change often throughout the lifetime of the system

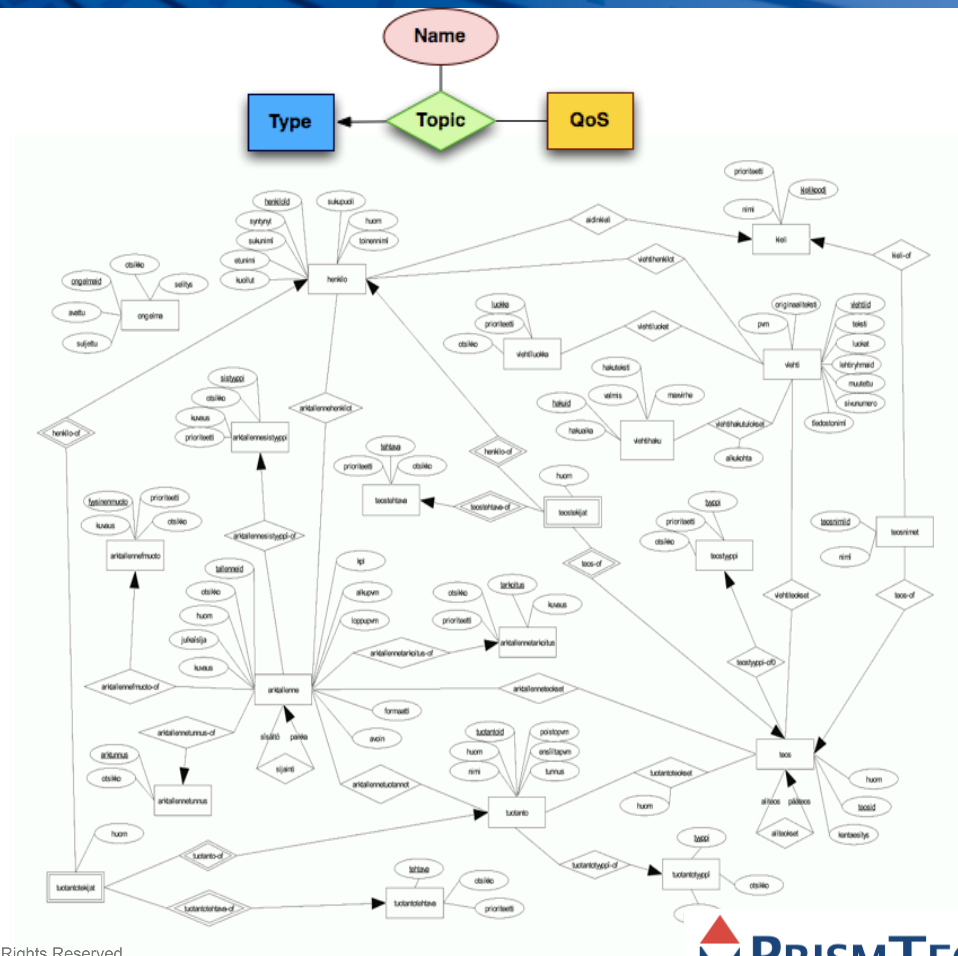
Context

- Most of the current practice in designing distributed system is based on a functional or OO decomposition whose goal is that of identifying the key Interfaces
- Different component of the system cooperate agreeing on interfaces, and invoking methods over these interfaces
- Examples are distributed systems based on CORBA, .NET, J2EE, Java RMI
- However these systems are fragile with respect to extensibility as well as integration with other technologies

Lingua Franca

Solution

- Focus on identifying the information model, i.e., **data and relationships**, underlying the distributed system, the **Lingua Franca**
- Information exchanged within and across a system is much more stable than functional interfaces
- The Lingua Franca provides the fabric that keeps together the system along with the QoS invariant capturing the non-functional requirements



Related Pattern

- **Global Data Space**. The **Lingua Franca** is often used along with the Global Data Space Pattern

Known Uses

- DDS
- DBMSs

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Shared Global Data Space

Shared Global Data Space

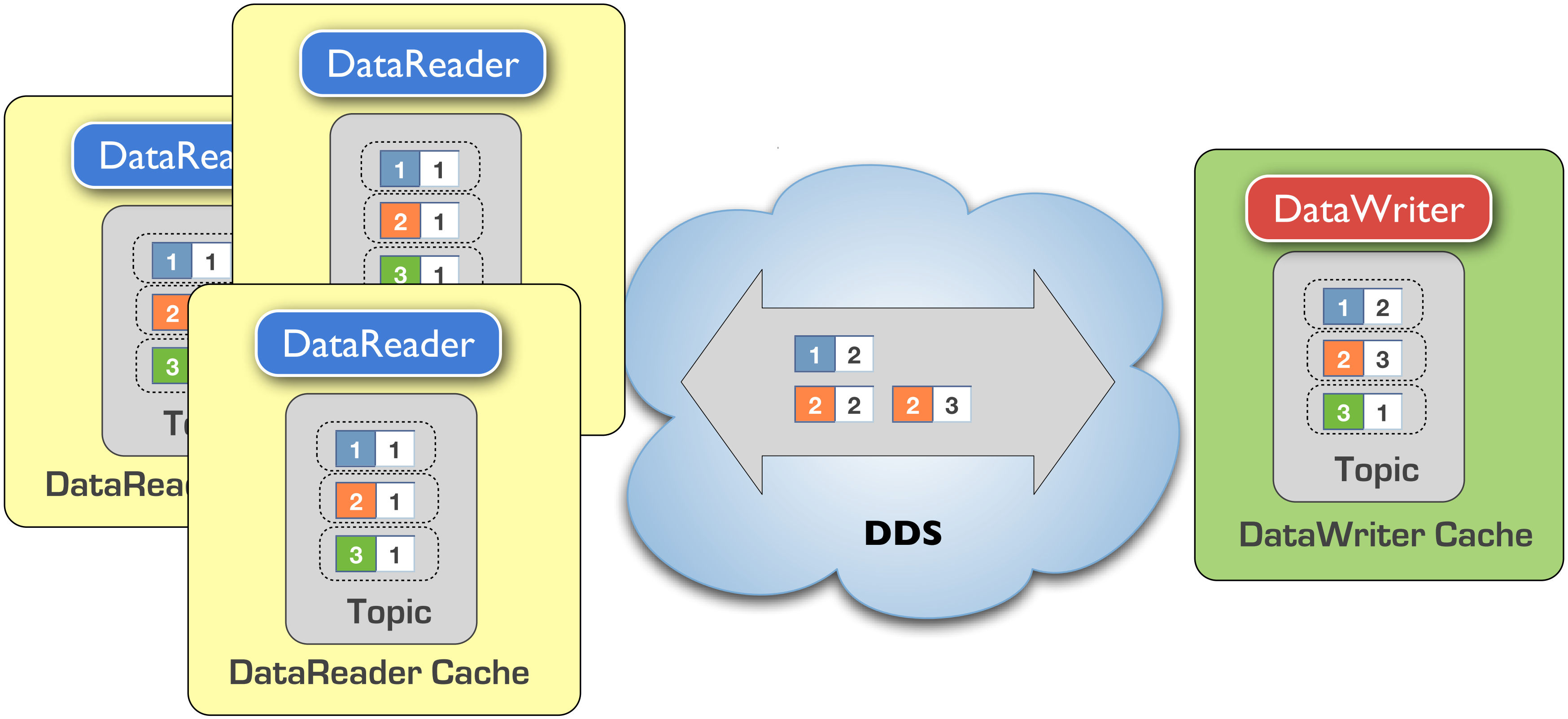
Coordination Model

- DDS applications are asynchronous and communicate by reading/writing from/to a Global Data Space
- DDS applications communicate by simply addressing items in the Global Data Space and without any direct knowledge of the parties involved in the production/consumption of data

Consistency Model

- The Shared Global Data Space implemented by DDS, can be configured to supports at most the "Eventual Consistency Model"
- Under an Eventual Consistency Model we are guaranteed that **eventually** all application in the system will have a consistent view of the "world"

Eventual Consistency & R/W Caches



Under an Eventual Consistency Model, DDS guarantees that all matched Reader Caches will eventually be identical of the respective Writer Cache

QoS Impacting the Consistency Model

The DDS Consistency Model is a property that can be associated to Topics or further refined by Reader/Writers. The property is controlled by the following QoS Policies:

- **DURABILITY**
  - VOLATILE | TRANSIENT_LOCAL | TRANSIENT | PERSISTENT
- **LIFESPAN**
- **RELIABILITY**
  - RELIABLE | BEST_EFFORT
- **DESTINATION ORDER**
  - SOURCE_TIMESTAMP | DESTINATION_TIMESTAMP

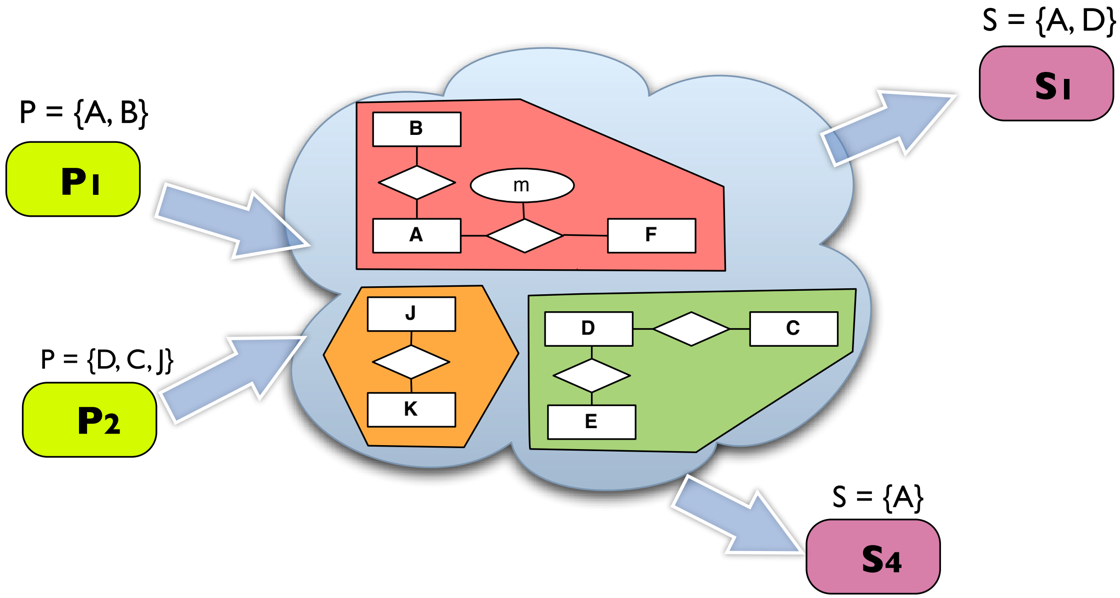
QoS Policy	Applicability	RxO	Modifiable
DURABILITY	T, DR, DW	Y	N
LIFESPAN	T, DW	-	Y
RELIABILITY	T, DR, DW	Y	N
DESTINATION ORDER	T, DR, DW	Y	N

QoS Impacting the Consistency Model

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN
Eventual Consistency (No Crash / Recovery)	VOLATILE	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.
Eventual Consistency (Crash/Recovery)	PERSISTENT	RELIABLE	SOURCE_TIMESTAMP	INF.
Weak Consistency	ANY	ANY	DESTINATION_TIMESTAMP	ANY
Weak Consistency	ANY	BEST_EFFORT	ANY	ANY
Weak Consistency	ANY	ANY	ANY	N

Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}

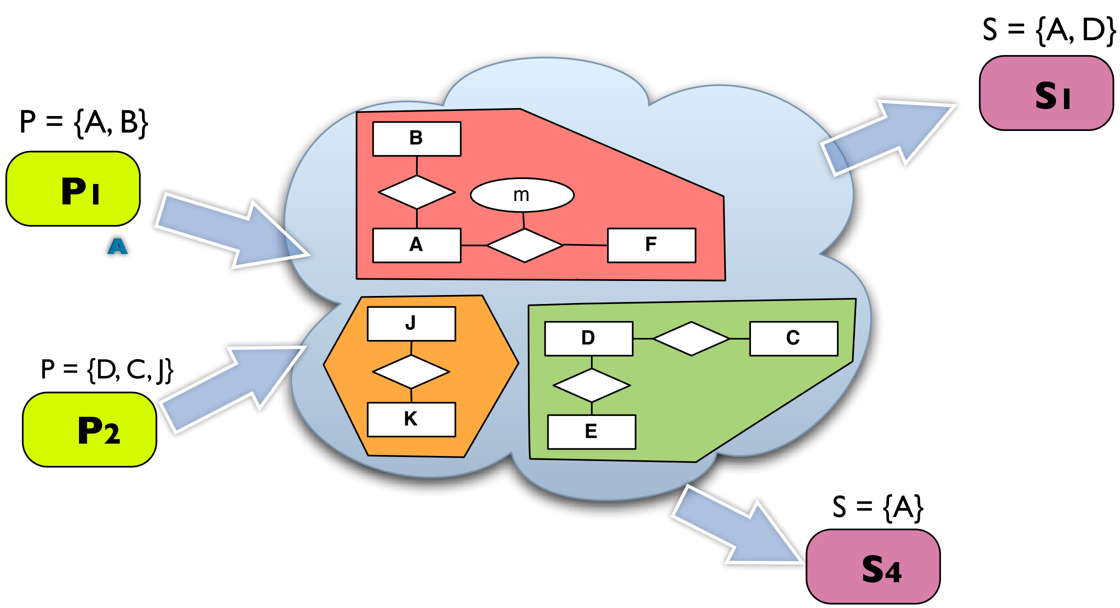


OpenSplice|DDS

PRISMTECH

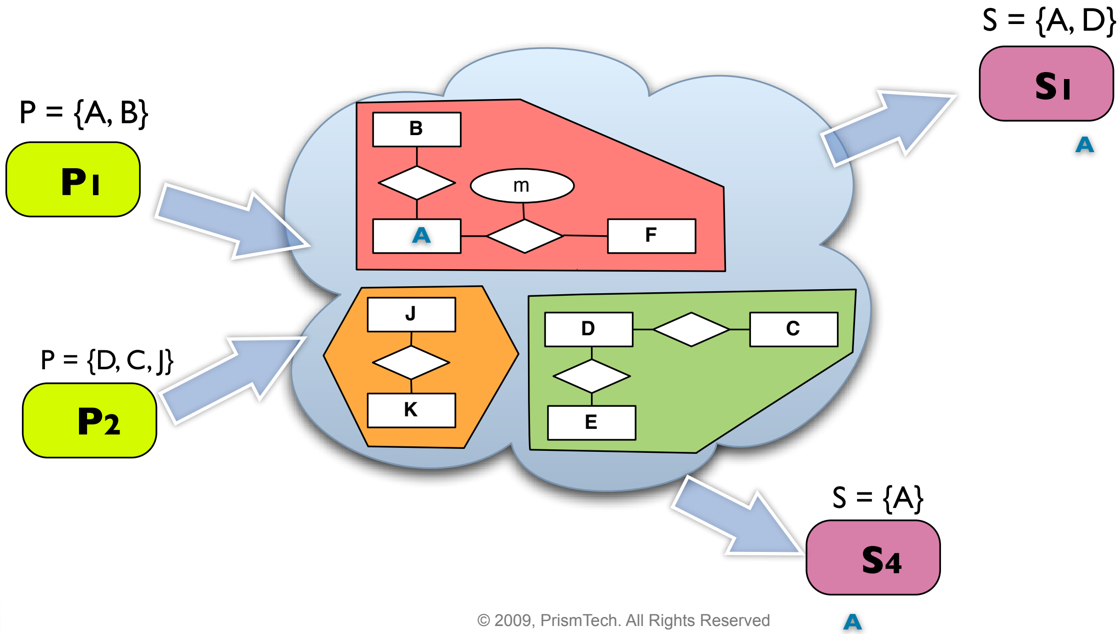
Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



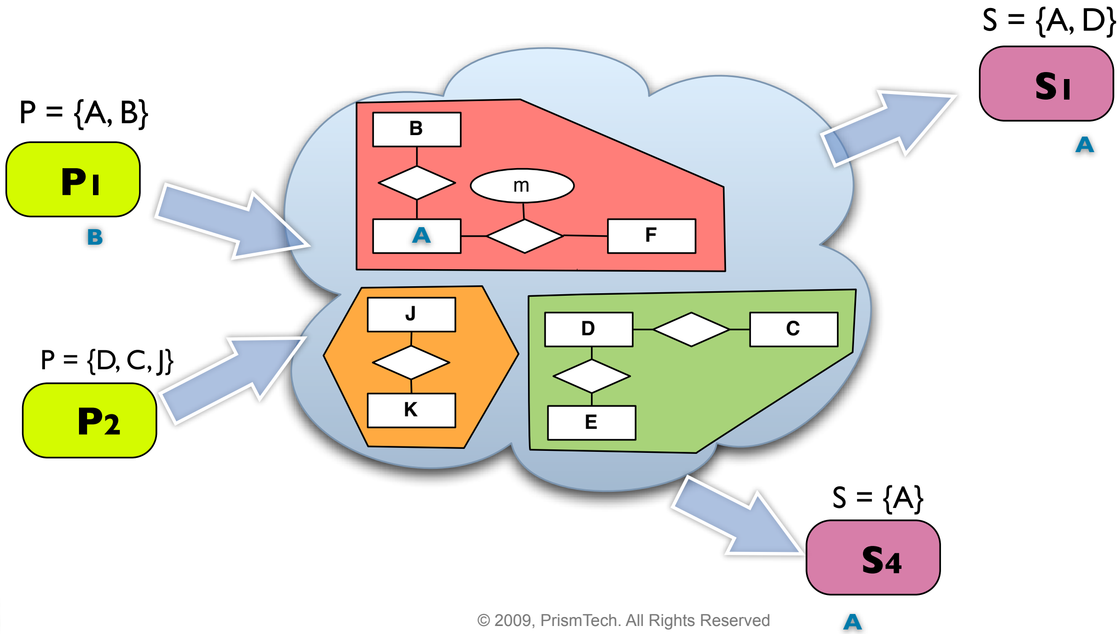
Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



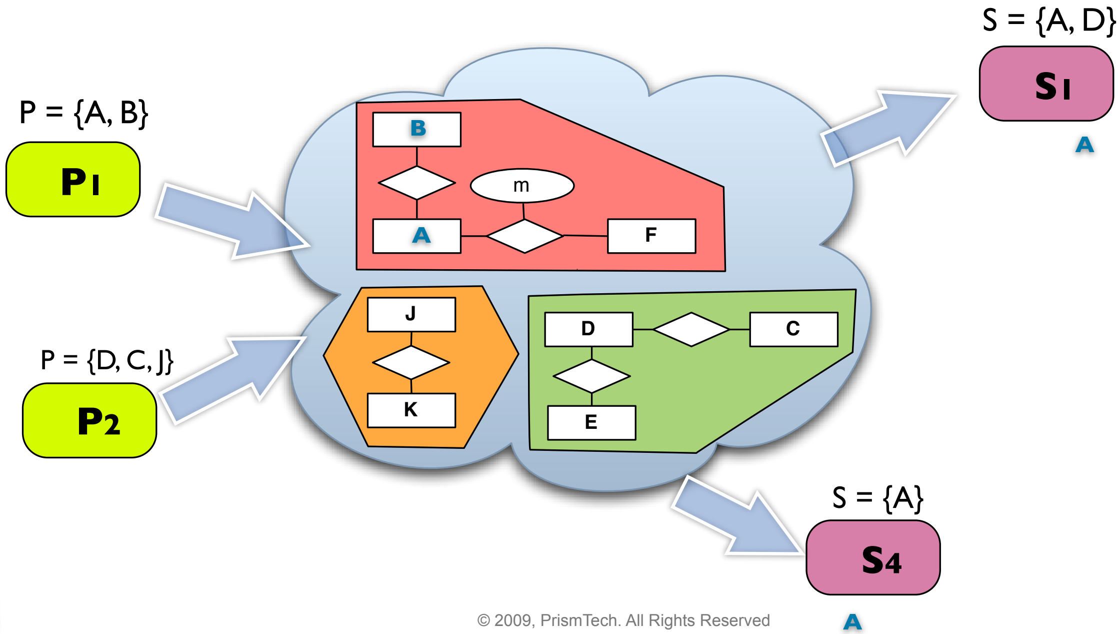
Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



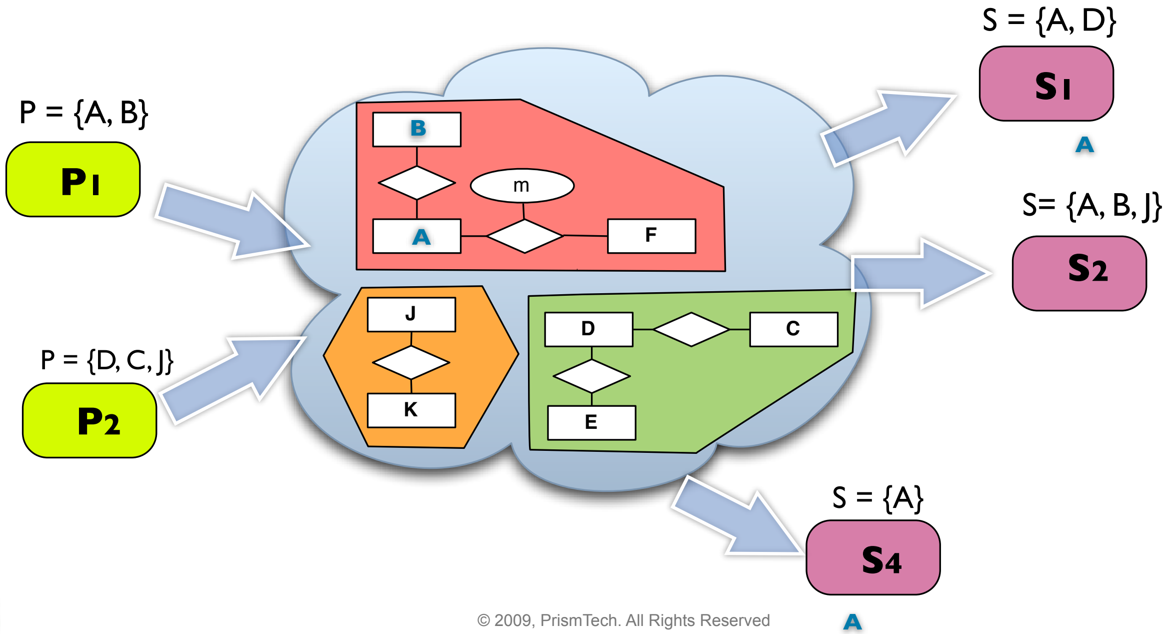
Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



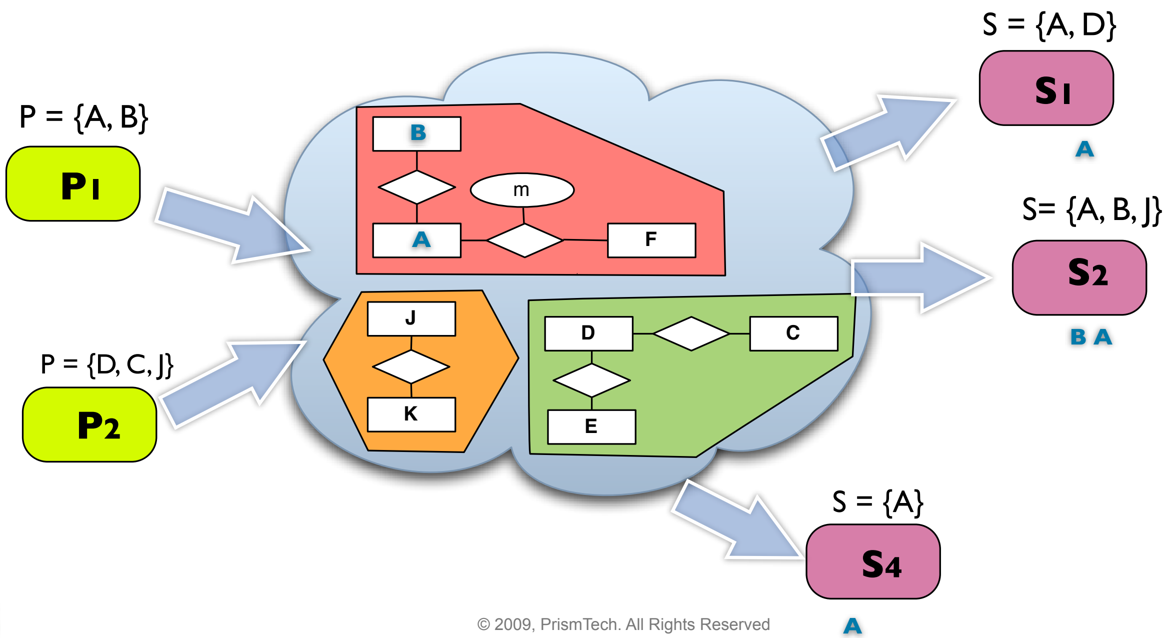
Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



Eventual Consistency @ WorK

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}

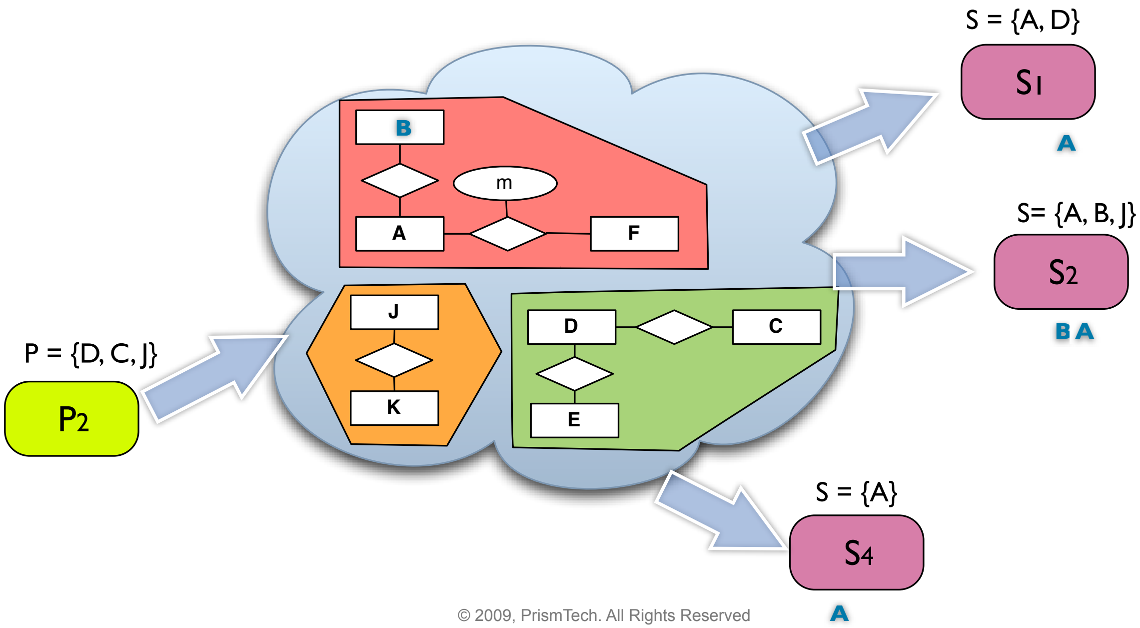


OpenSplice|DDS

PRISMTECH

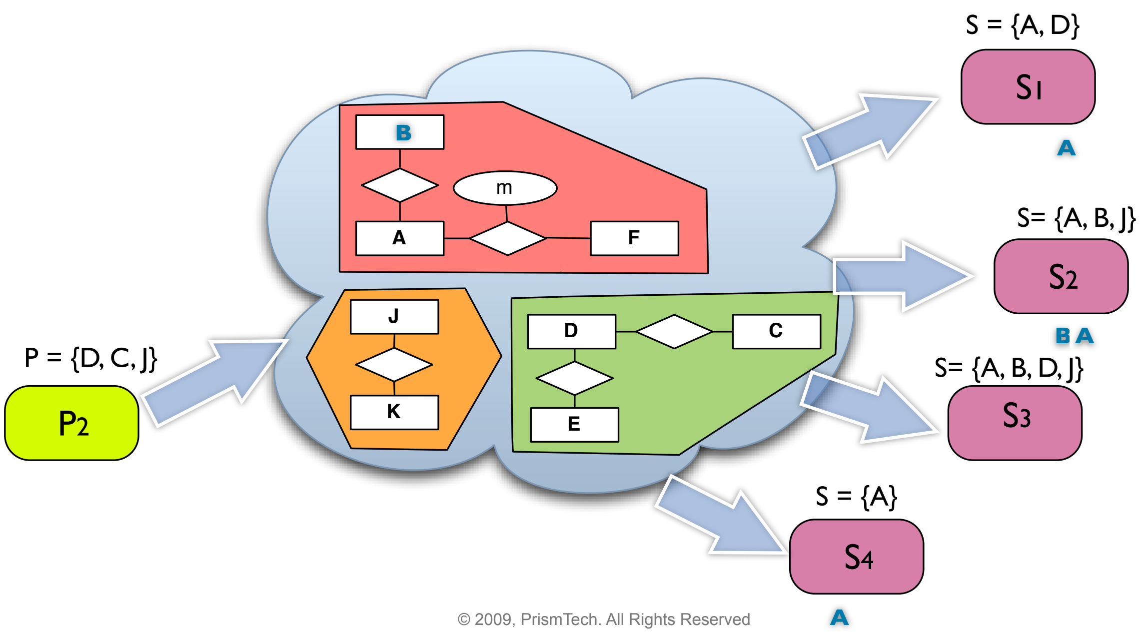
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



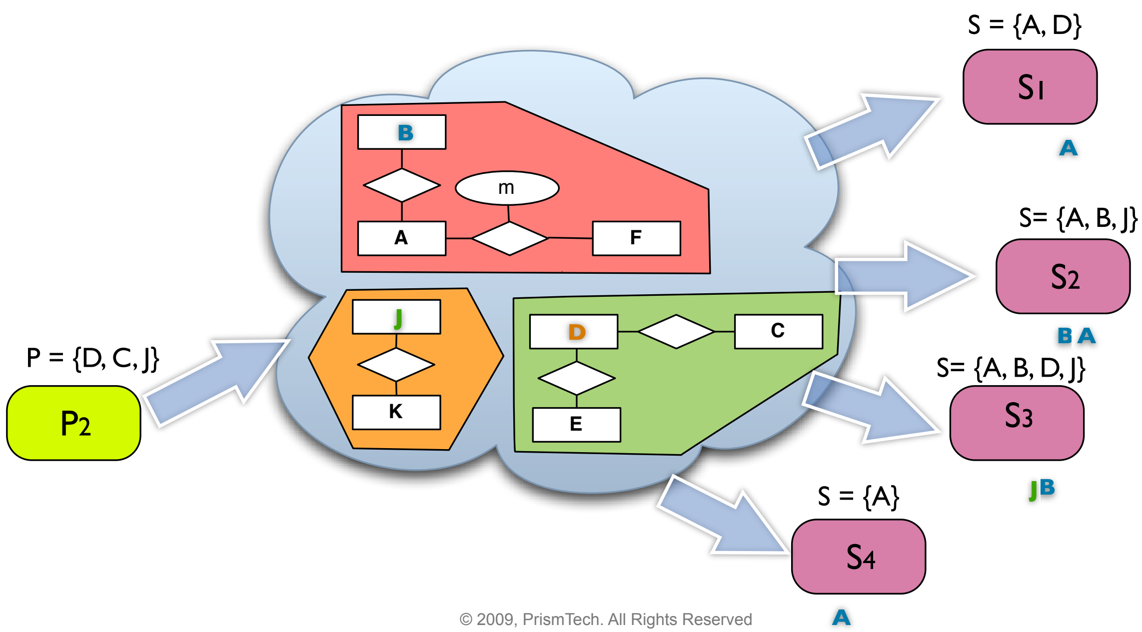
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



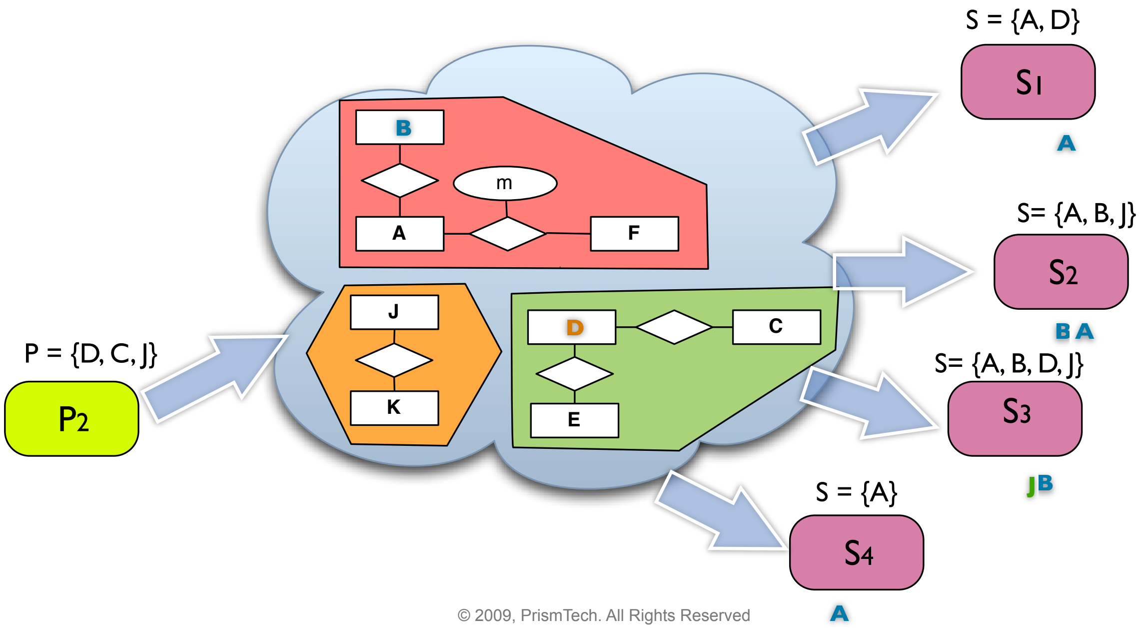
Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



Eventual Consistency @ Work

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN	
Eventual Consistency (Reader Crash / Recovery)	TRANSIENT_LOCAL	RELIABLE	SOURCE_TIMESTAMP	INF.	{A}
Eventual Consistency (Crash/Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.	{B}
Weak Consistency	ANY	ANY	ANY	N	{J}



Design Guidelines

- For all (non-periodic) Topics for which an eventually consistent model is required use the following QoS settings:

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN
Eventual Consistency (Crash / Recovery)	TRANSIENT	RELIABLE	SOURCE_TIMESTAMP	INF.

- For information produced periodically, with a period P, where P is small enough to be acceptable as a consistency convergence delay, the following QoS settings will provide an approximation of the eventual consistency:

	DURABILITY	RELIABILITY	DESTINATION_ORDER	LIFESPAN
Eventual Consistency (Crash / Recovery)	VOLATILE	BEST_EFFORT	SOURCE_TIMESTAMP	INF.

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Data Access Patterns

Topic Queues

Context

- One commonly used technique for implementing distributed real-time embedded systems is to model applications as FSA, or DFSA (Distributed Finite State Automata)

Problem

- One or more DDS applications are implemented as a (D)FSA whose transitions depends on the totally ordered history of updates for a specific topic
- How can we ensure that each application sees exactly the same set of updates in exactly the same order?

Topic Queues

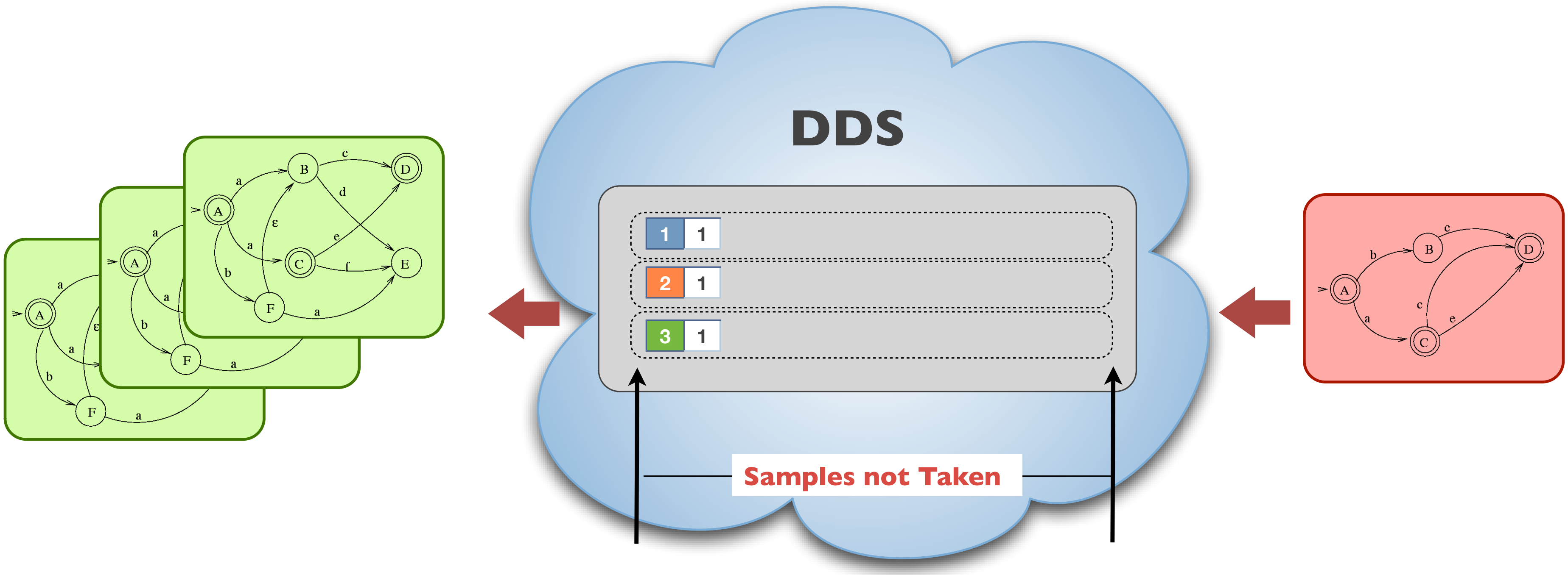
Assumptions

- Single writer exists per Topic Instance

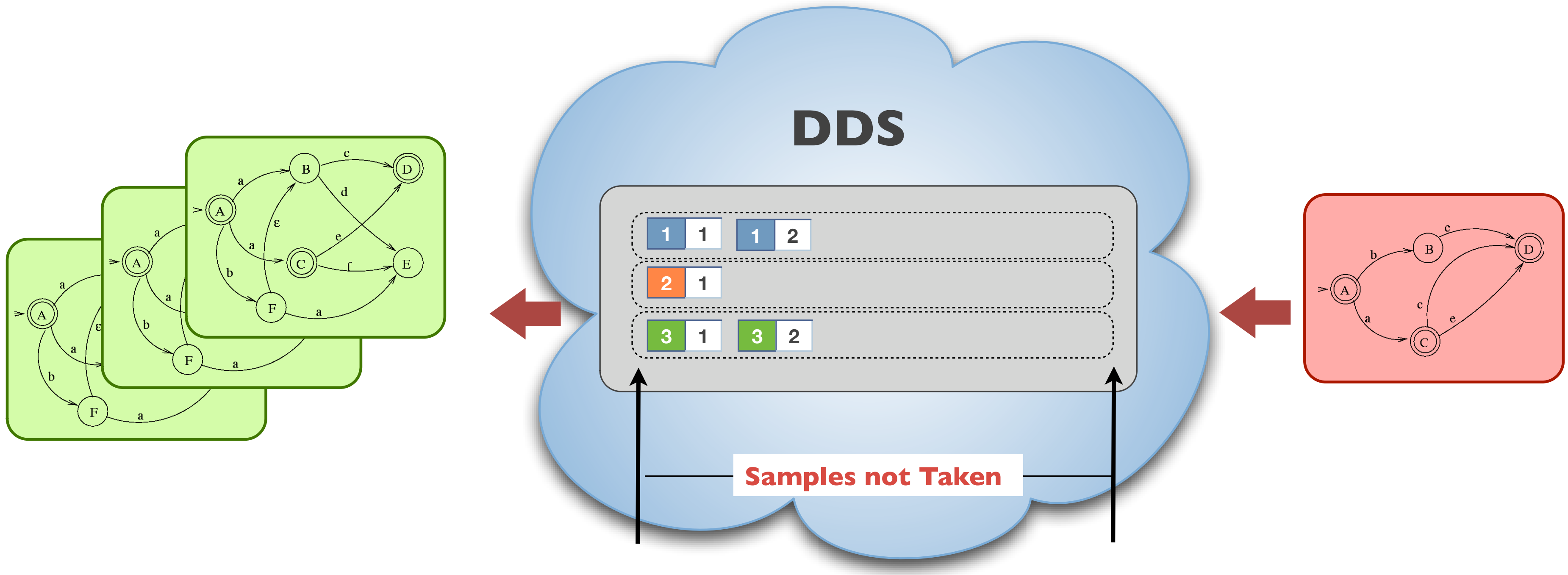
Solution

- Represent the state transition events by means of DDS Topics
- Topic Instances can be used to identify specific FSA
- Ensure that application always use the **Take Semantics** for getting data.
- Ensure that these topics have the following QoS Settings
  - **DURABILITY**: TRANSIENT | PERSISTENT
  - **HISTORY:** KEEP_ALL
  - **RELIABILITY**: RELIABLE
  - **DESTINATION_ORDER:** SOURCE_TIMESTAMP

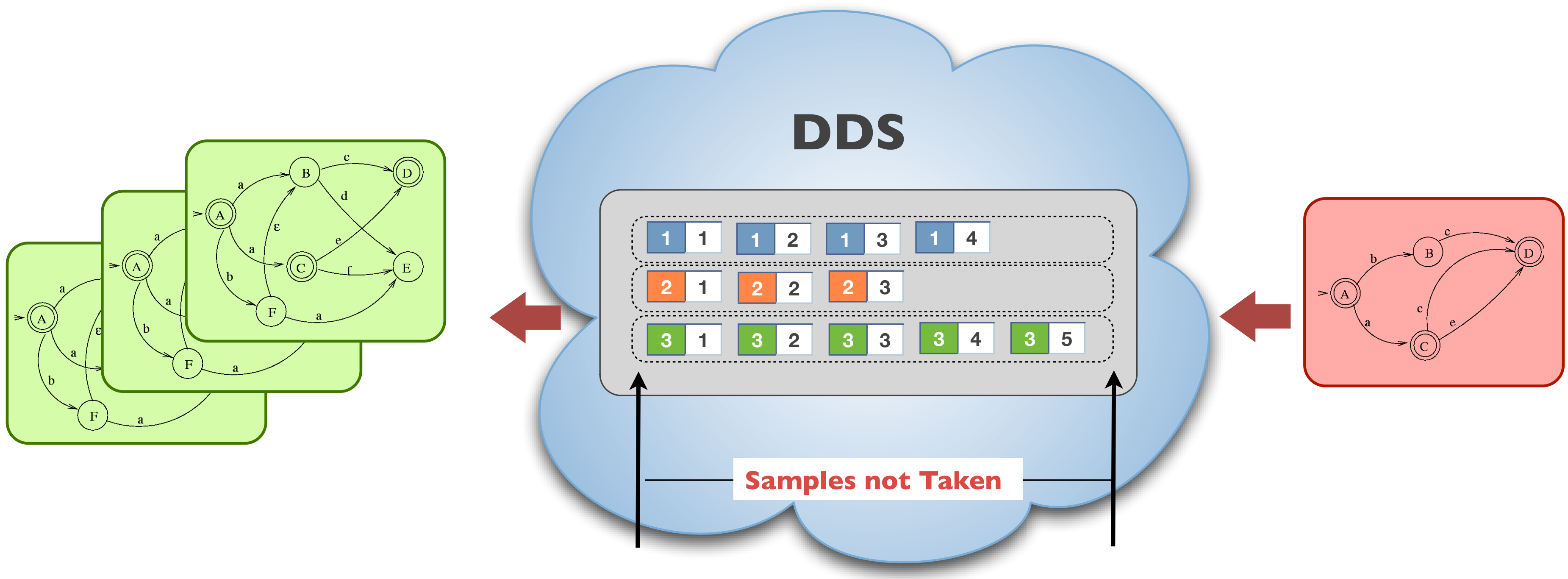
Topic Queues



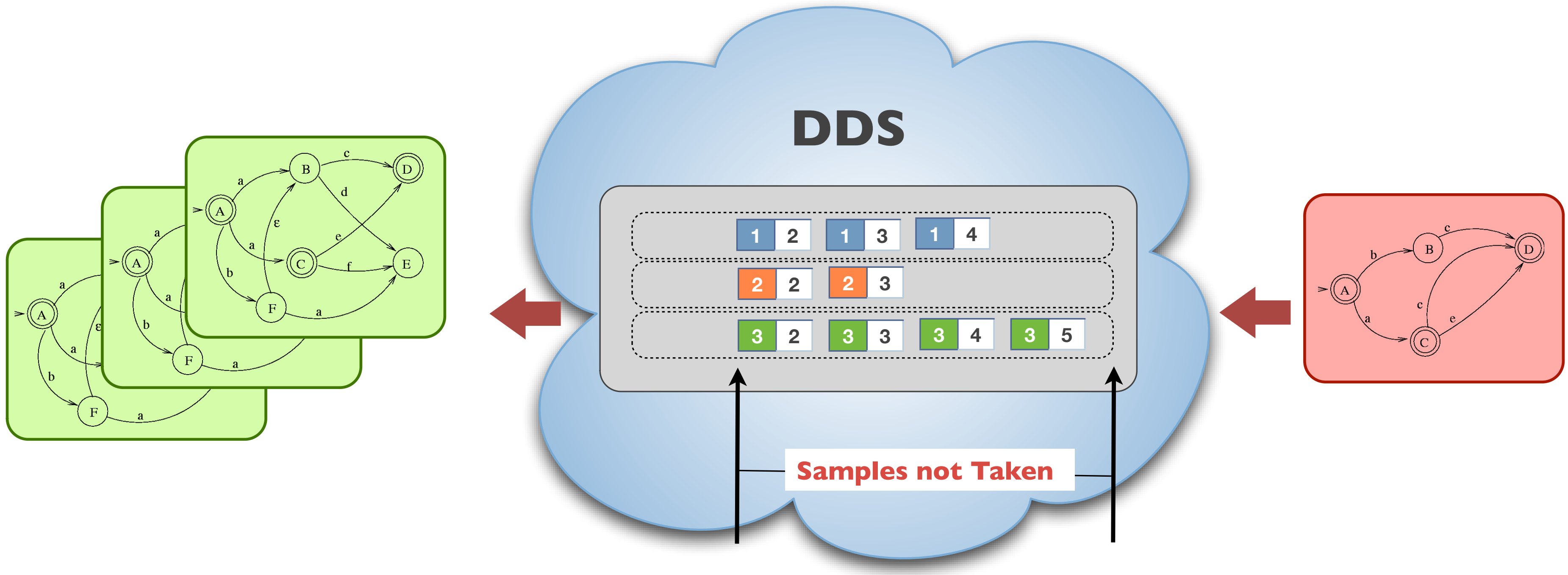
Topic Queues



Topic Queues



Topic Queues



Topic K-Queues

Context

- One commonly used technique for implementing distributed real-time embedded systems is to model applications as FSA, or DFSA (Distributed Finite State Automata)

Problem

- One or more DDS applications are implemented as a (D)FSA whose transitions depends on the totally ordered history of updates for a specific topic
- How can we ensure that each application sees exactly the same set of updates in exactly the same order?
- How can we ensure that **misbehaving applications** consume an **unbounded amount of memory**?

Topic K-Queues

Assumptions

- Single writer exists per Topic Instance

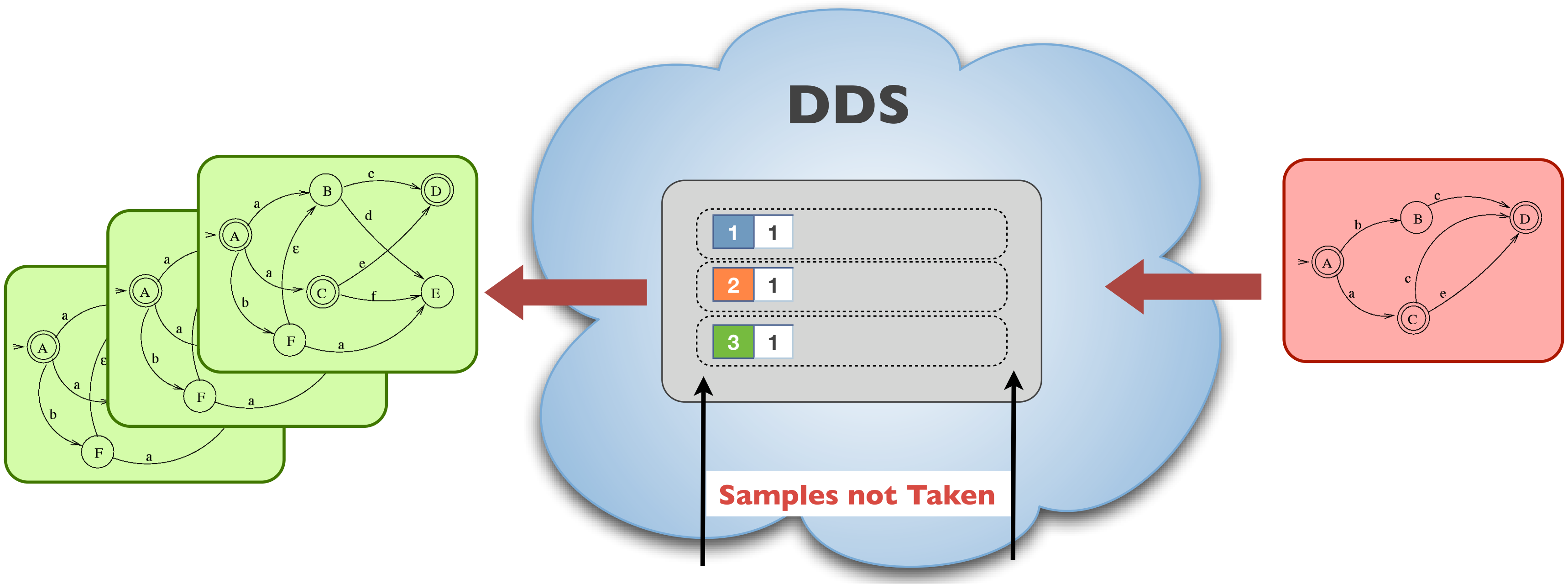
Solution

- Represent the state transition events by means of DDS Topics
- Topic Instances can be used to identify specific FSA
- Ensure that application always use the **Take Semantics** for getting data
- Ensure that these topics have the following QoS Settings
  - **DURABILITY**: TRANSIENT | PERSISTENT
  - **RELIABILITY:** RELIABLE
  - **HISTORY:** KEEP_LAST with DEPTH set to **K**
  - **DESTINATION_ORDER:** SOURCE_TIMESTAMP
- If a FSA looses a state transition "resets the state" by some other means (e.g. DURABILITY SERVICE)

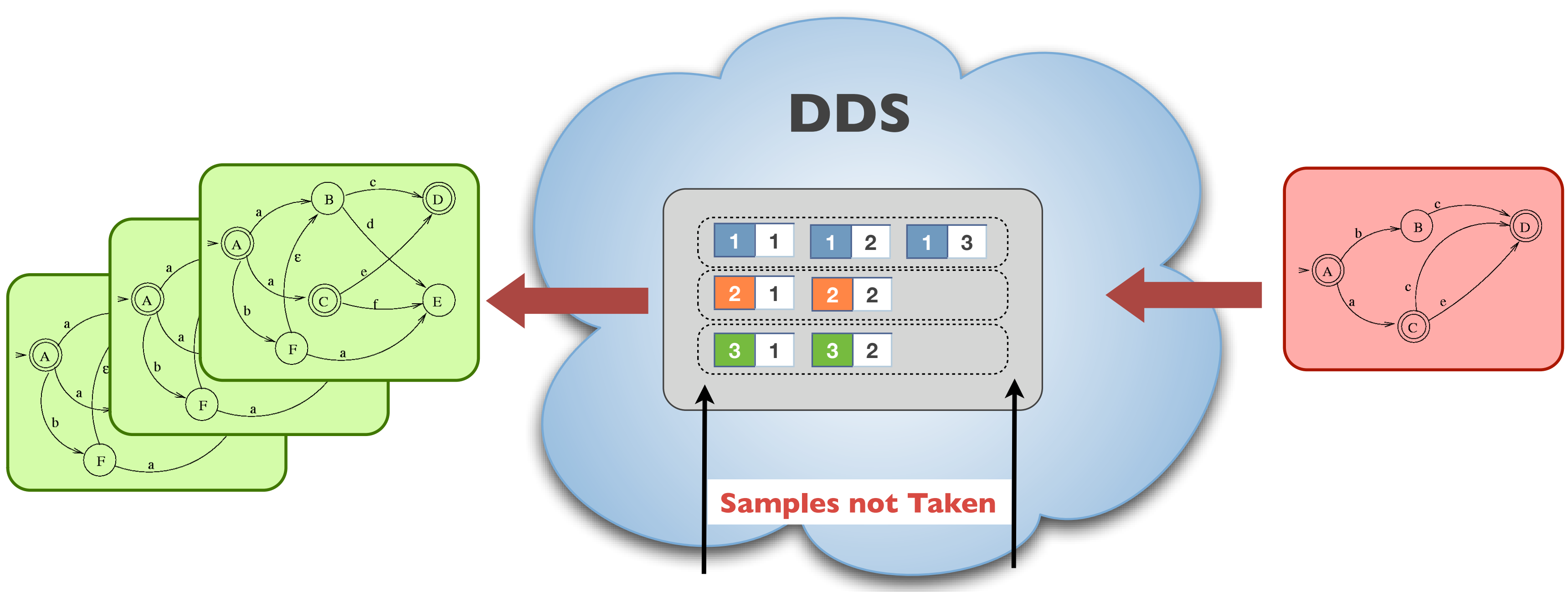
Note

- K can be dimensioned by considering the maximum burst of activity that should be tolerated along with the average time between state transitions
- e.g., if I want to tolerate 12 sec of overload and state transition occur every 4 sec then K=3

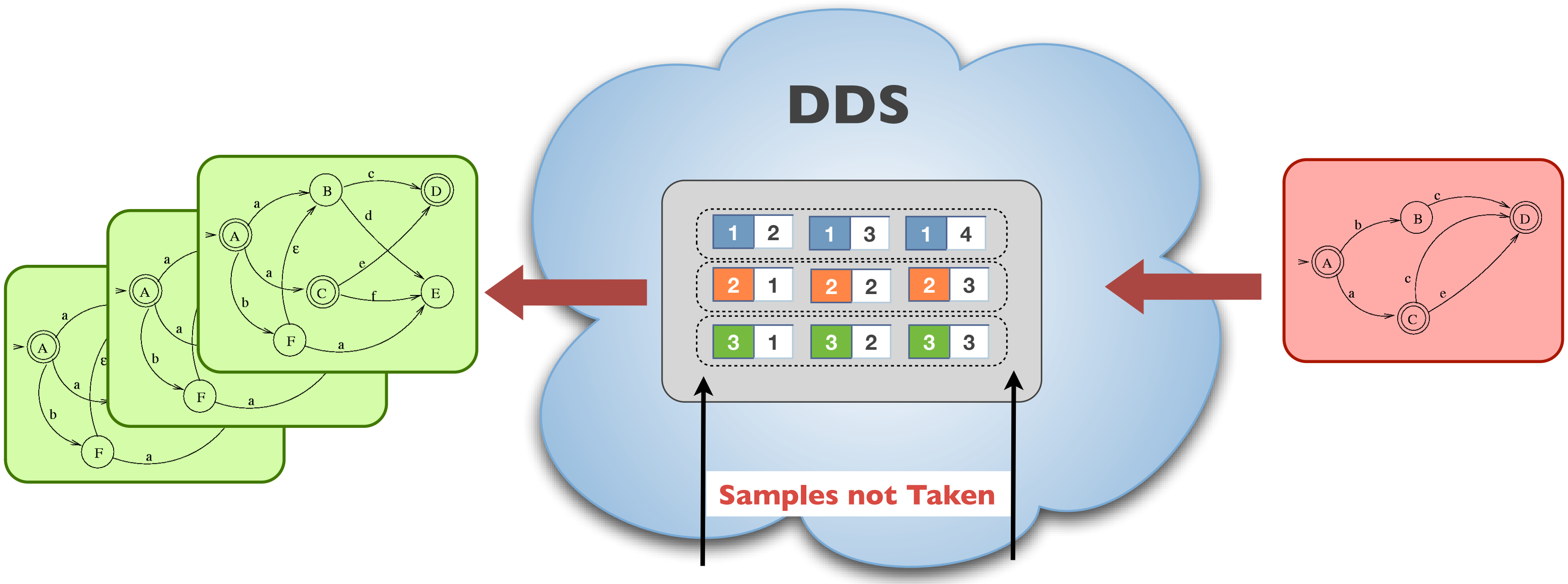
Topic K-Queues



Topic K-Queues

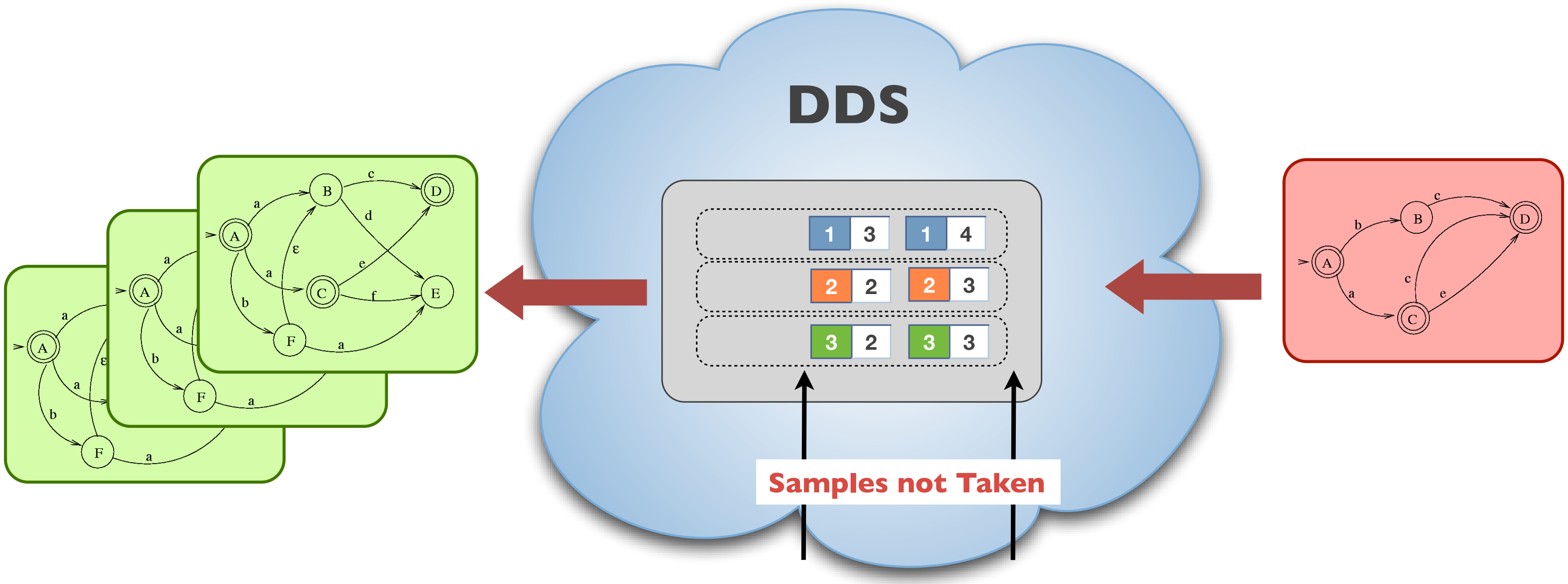


Topic K-Queues



- One FSA has missed a sample... but does not know it yet.

Topic K-Queues



- One FSA has missed a sample... but does not know it yet.
- It detects this and gets the sample from the DURABILITY SERVICE

Topic Caches

Problem

- Distributed applications often have to deal with "Hard State", meaning state that is conceptually shared among various elements.
- This "Hard State" often needs to be accesses very efficiently, likewise changes in state should also consistently diffused in a timely manner

Topic Caches

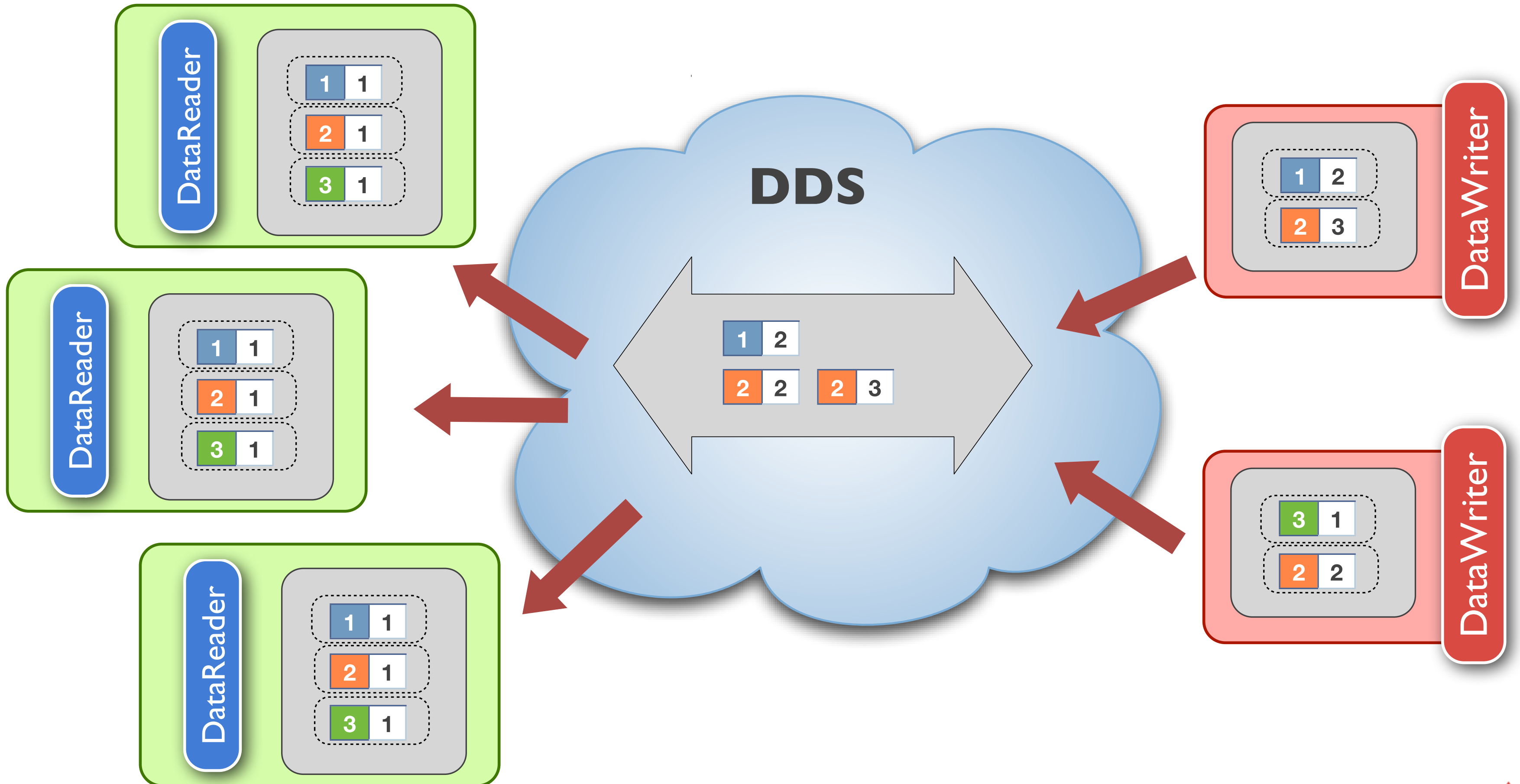
Solution

- Represent the “Hard State” by means of DDS Topics
- Application should favor the use the **Read Semantics** for getting data.
- Ensure that these topics have the following QoS Settings
  - **DURABILITY**: TRANSIENT | PERSISTENT
  - **HISTORY:** KEEP_LAST
  - **RELIABILITY**: RELIABLE
  - **DESTINATION_ORDER:** SOURCE_TIMESTAMP

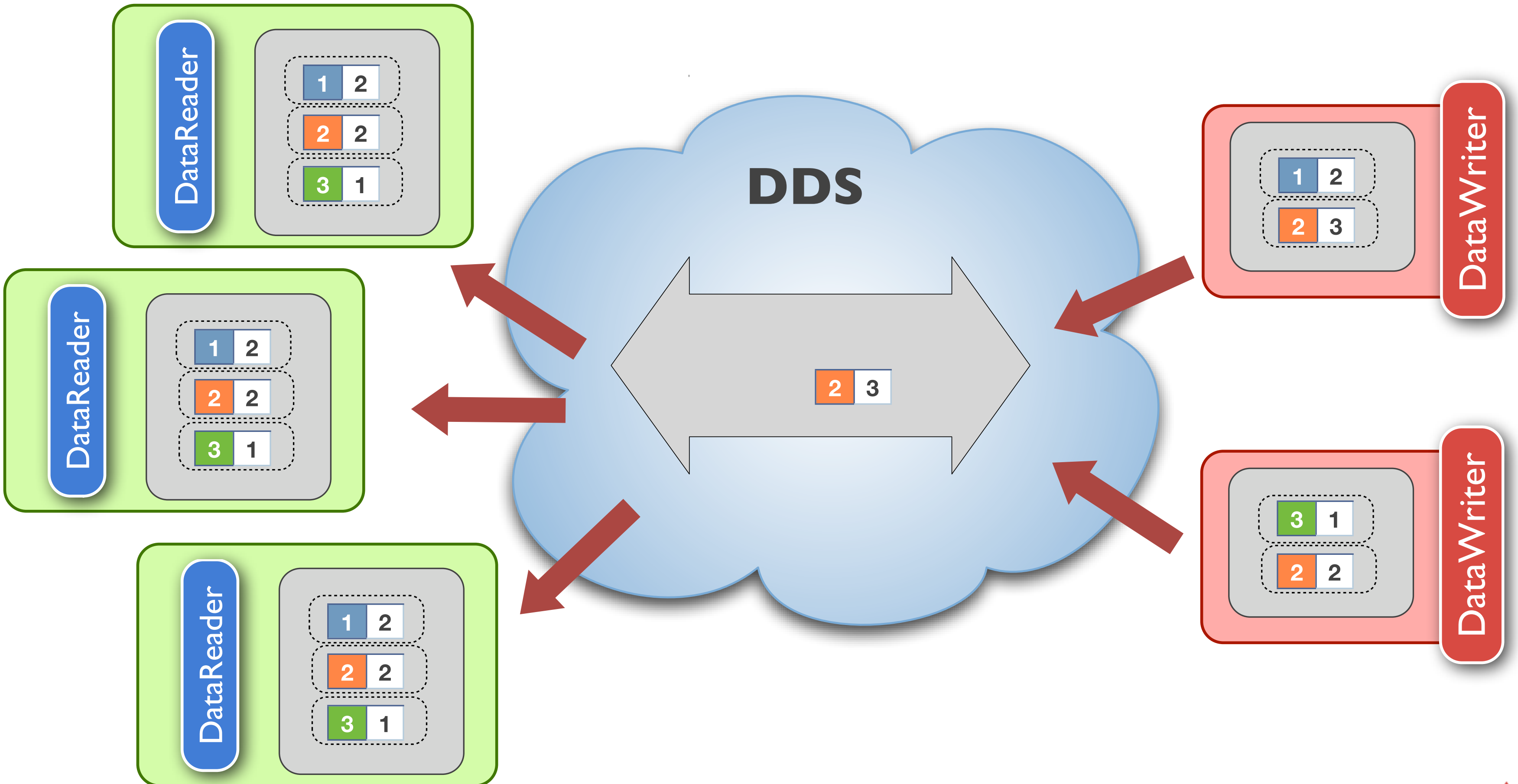
Note

- Notice that “Hard State” will be eventually consistent for all reader regardless of the number of writers
- This technique can be exploited for writing **self-stabilizing applications**

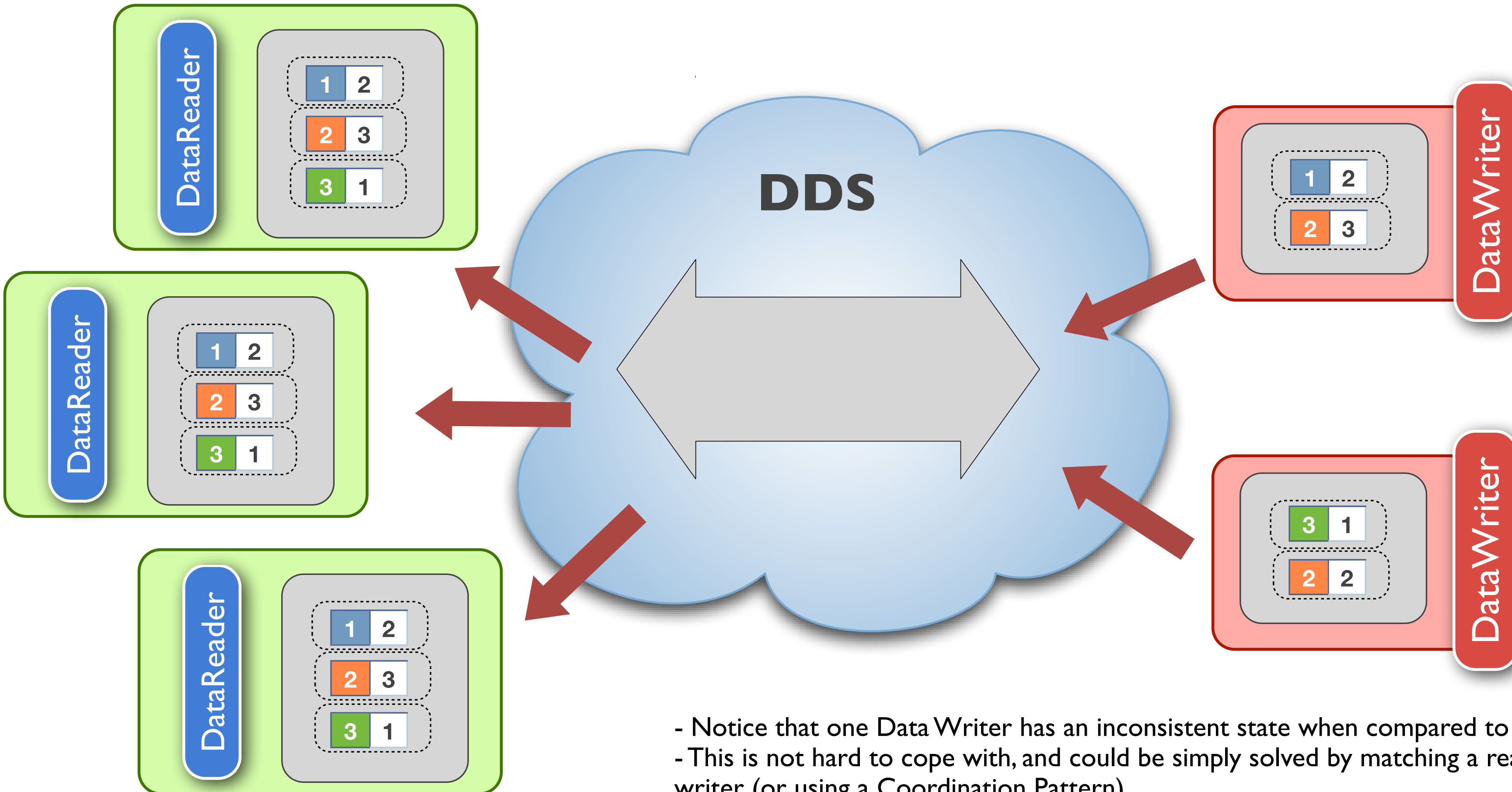
Topic Caches



Topic Caches



Topic Caches



- Notice that one Data Writer has an inconsistent state when compared to the system state.
- This is not hard to cope with, and could be simply solved by matching a reader with the writer (or using a Coordination Pattern)

OpenSplice DDS

PrismTech

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

Coordination Pattern

Sequencer

Problem

- Often occurs in a distributed system that applications need to coordinate and take turn in performing certain actions.
- However DDS does not provide built-in coordination / distributed synchronization mechanisms. How can this be overcome?

Solution

- Define a Sequencer in your system that coordinates access to resources.

Sequencer

Detailed Solution

- Define in your system the following Topics:

```
struct TAccessRequest {
  long resource_guid;
  long request_guid;
  time_t timeout;
};
#pragma keylist AccessRequestTopic resource_guid
```

```
struct TAccessGrant {
  long resource_guid;
  long request_guid;
  time_t timeout;
};
#pragma keylist AccessRequestTopic resrouce_guid
```

```
struct TReleaseAccessGrant {
  long resource_guid;
  long request_guid;
};
#pragma keylist AccessRequestTopic resrouce_guid
```

Sequencer

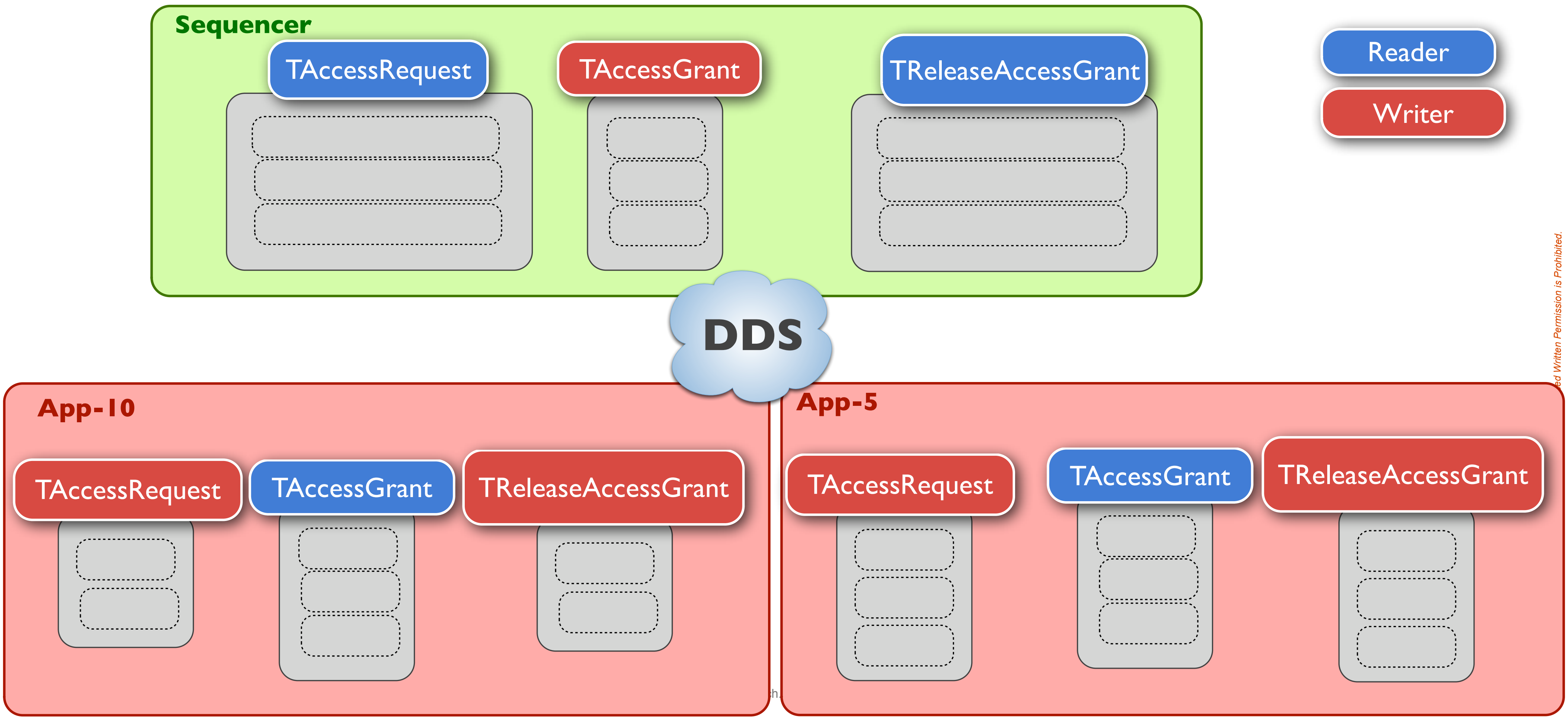
Detailed Solution

- Make `TAccessRequest` and `TReleaseAccessGrant` **Topic-Queues** (with DESTINATION_ORDER set to RECEPTION_TIMESTAMP)
- Make `TAccessGrant` a **Topic-Cache**
- Use the following protocol to request/grant/release access

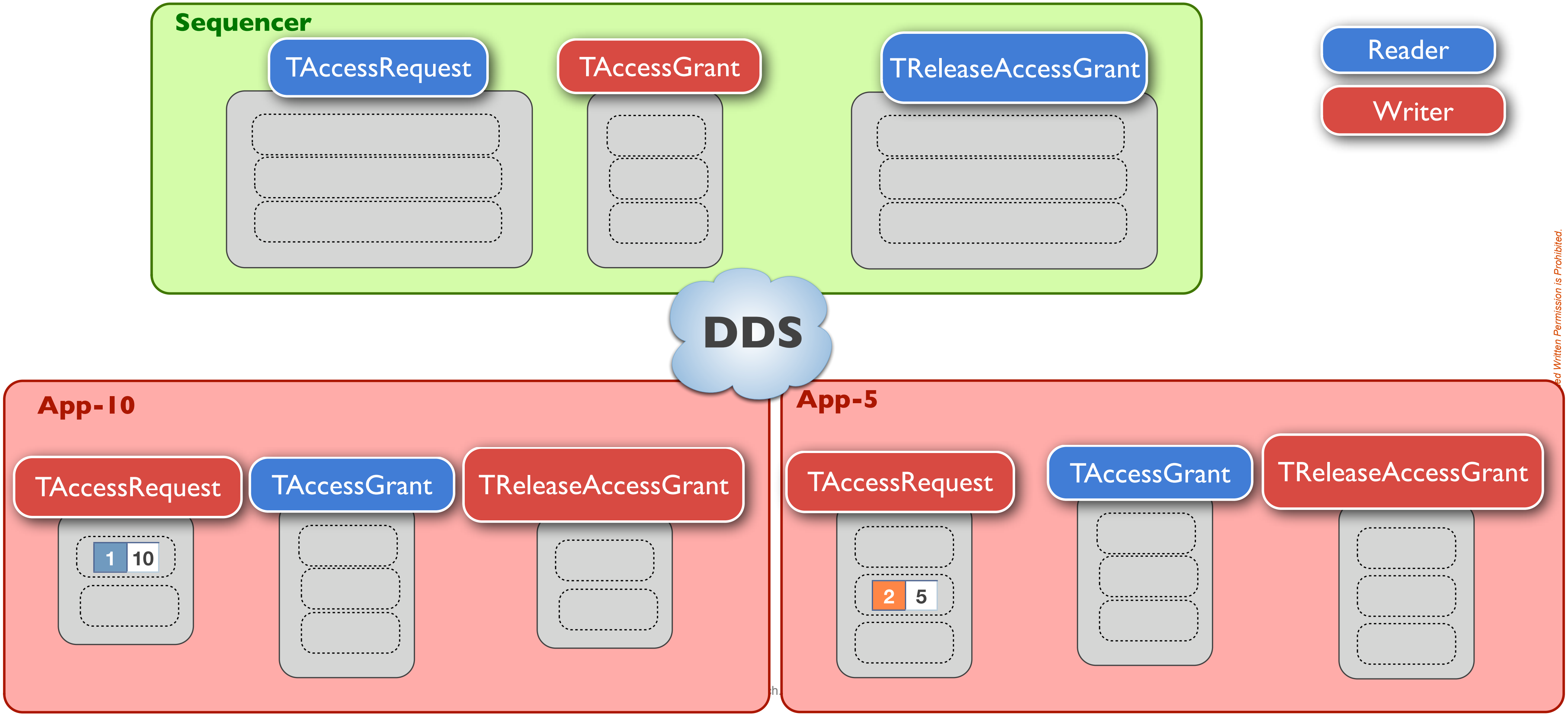
```
Sequencer (per instance to keep it simpler)

while (true) {
   take next sample from TAccessRequest
   write TAccessGrant
   wait  on TReleaseAccessGrant
}
```

```
Application

bool granted = false;
write TAccessRequest
while (!granted) {
   wait  on TAccessGrant
   if TAccessGrant == myTAccessGrant
      granted = true;
}
// Do Critical Section
write TReleaseAccessGrant
```

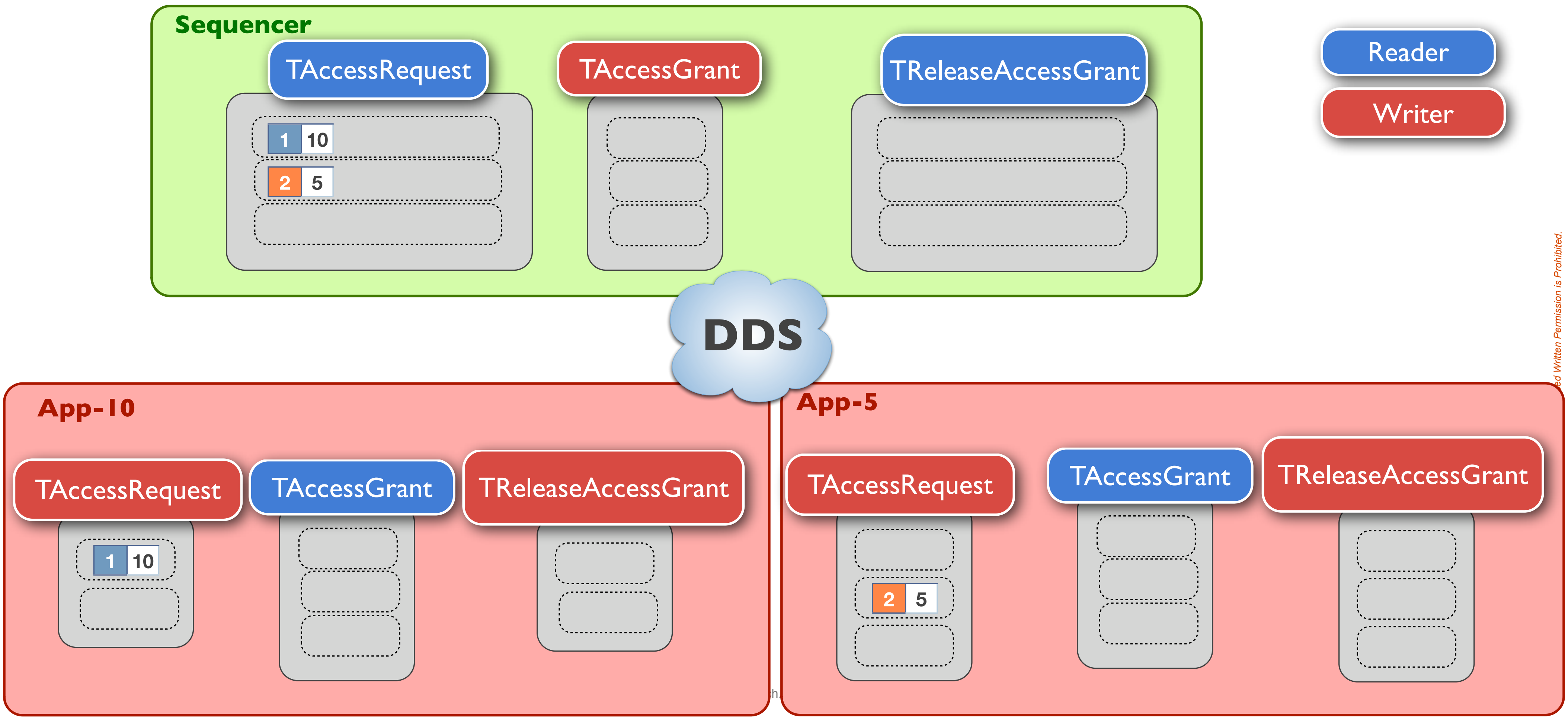
Sequencer @ Work



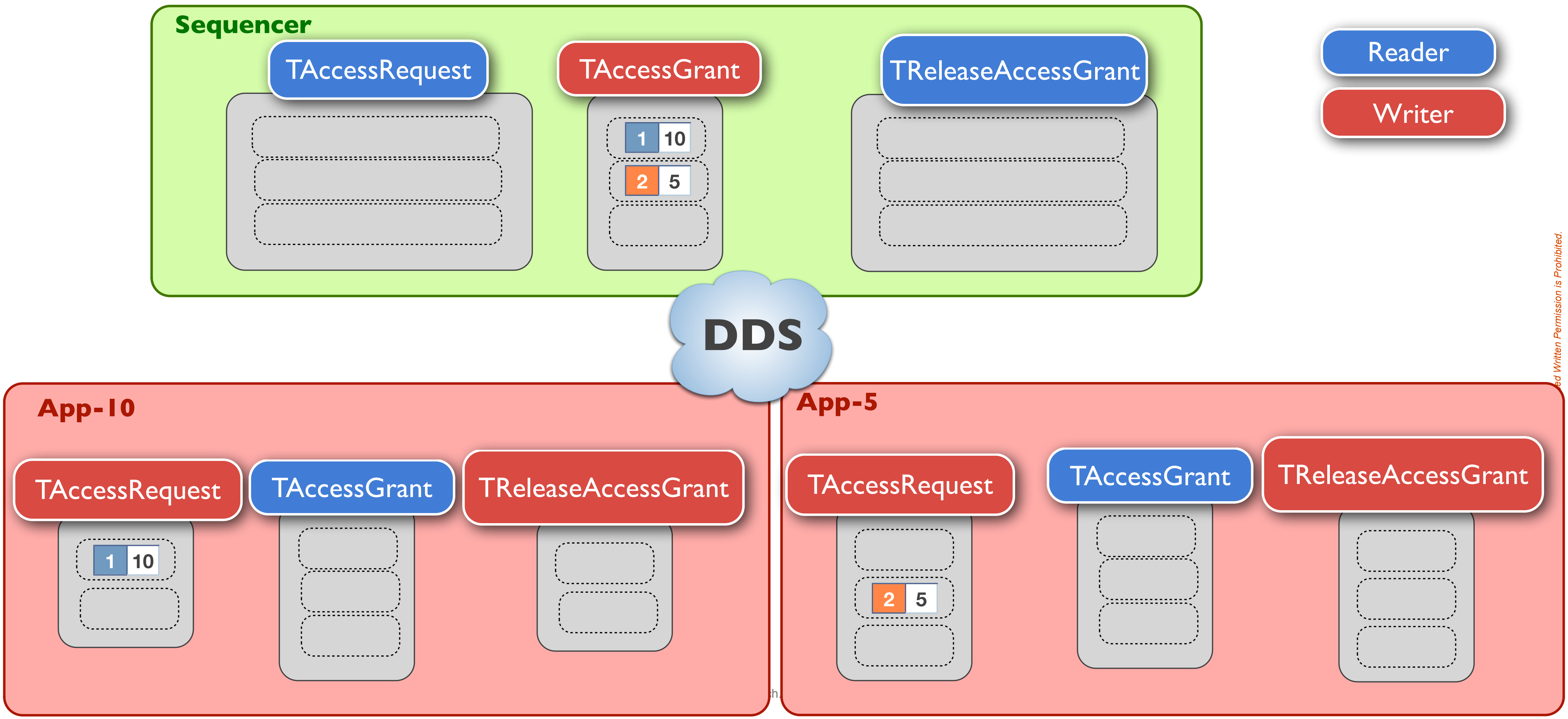
Sequencer @ Work



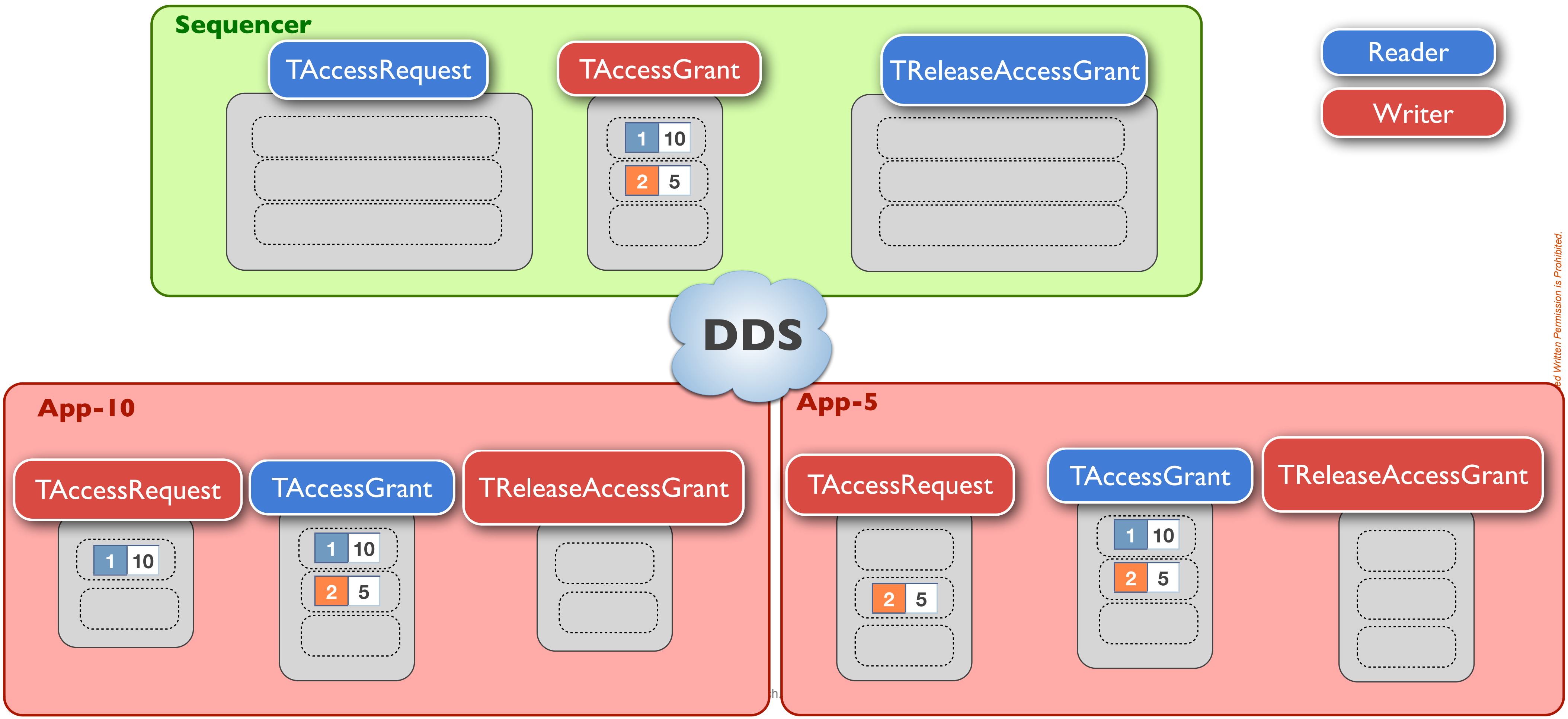
Sequencer @ Work



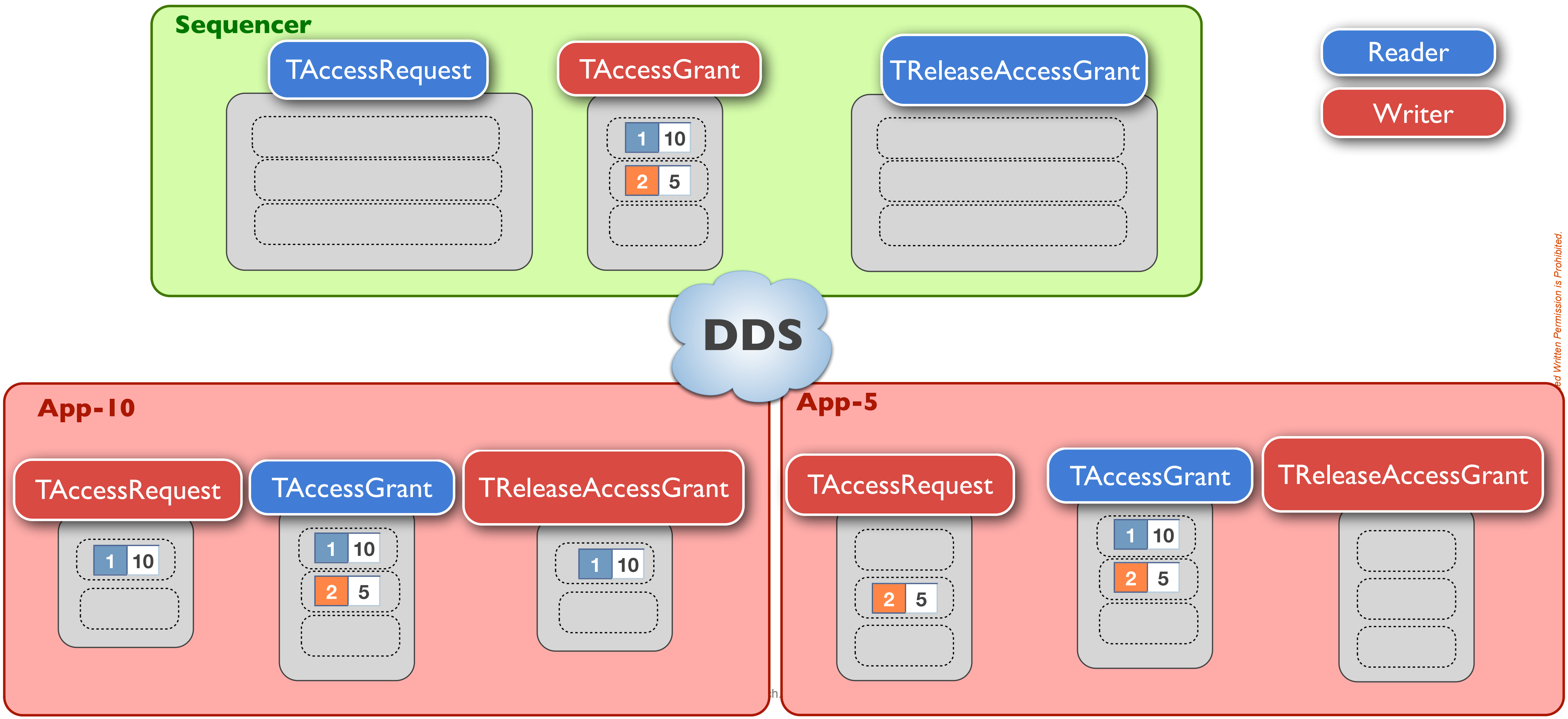
Sequencer @ Work



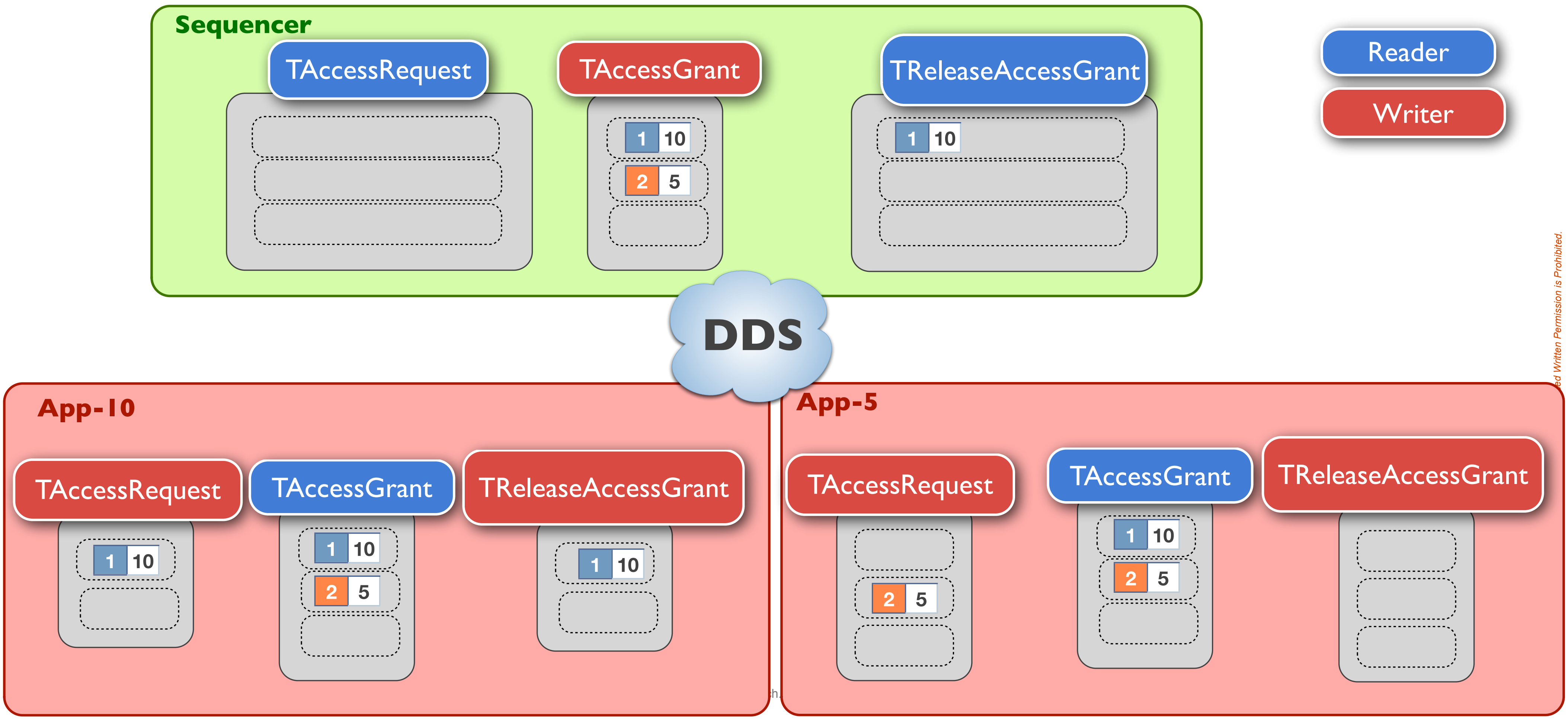
Sequencer @ Work



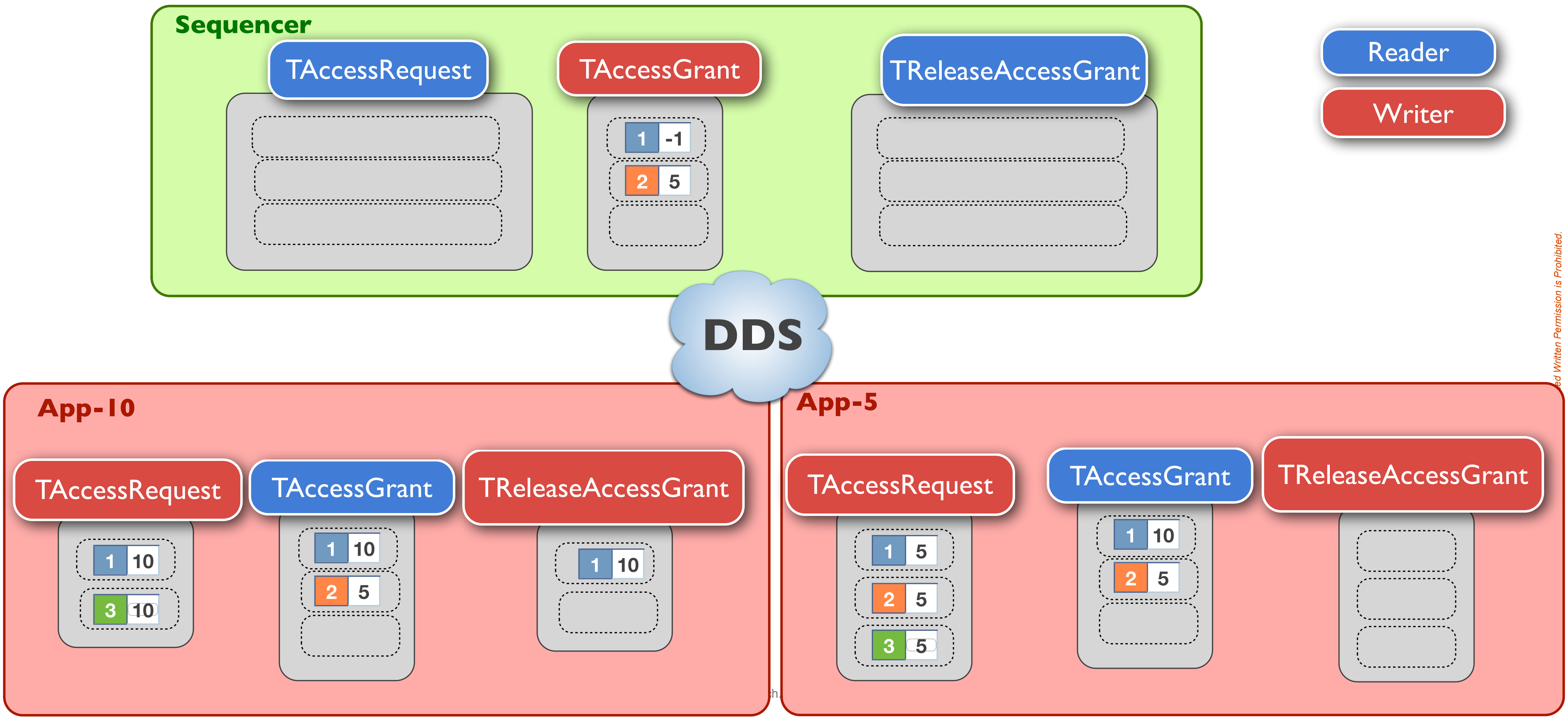
Sequencer @ Work



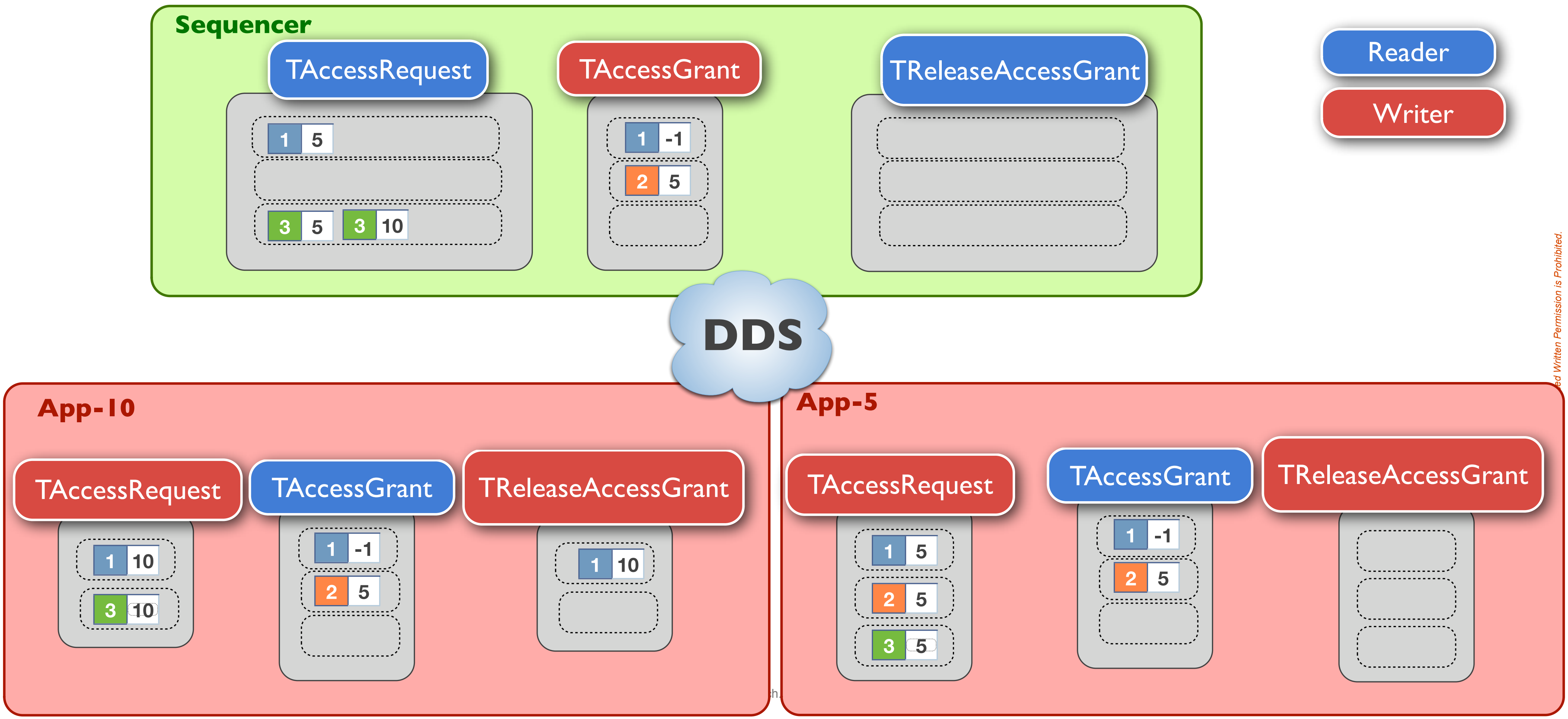
Sequencer @ Work



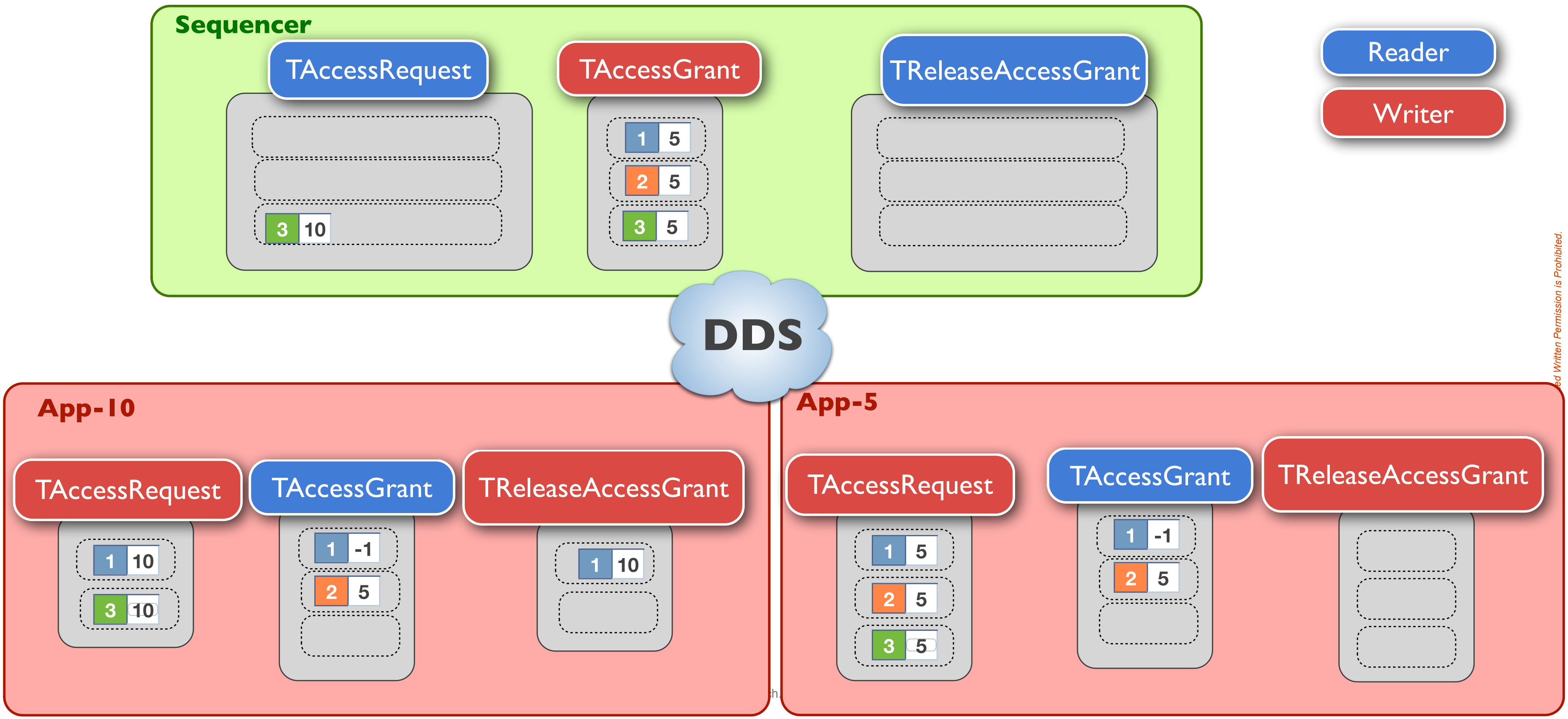
Sequencer @ Work



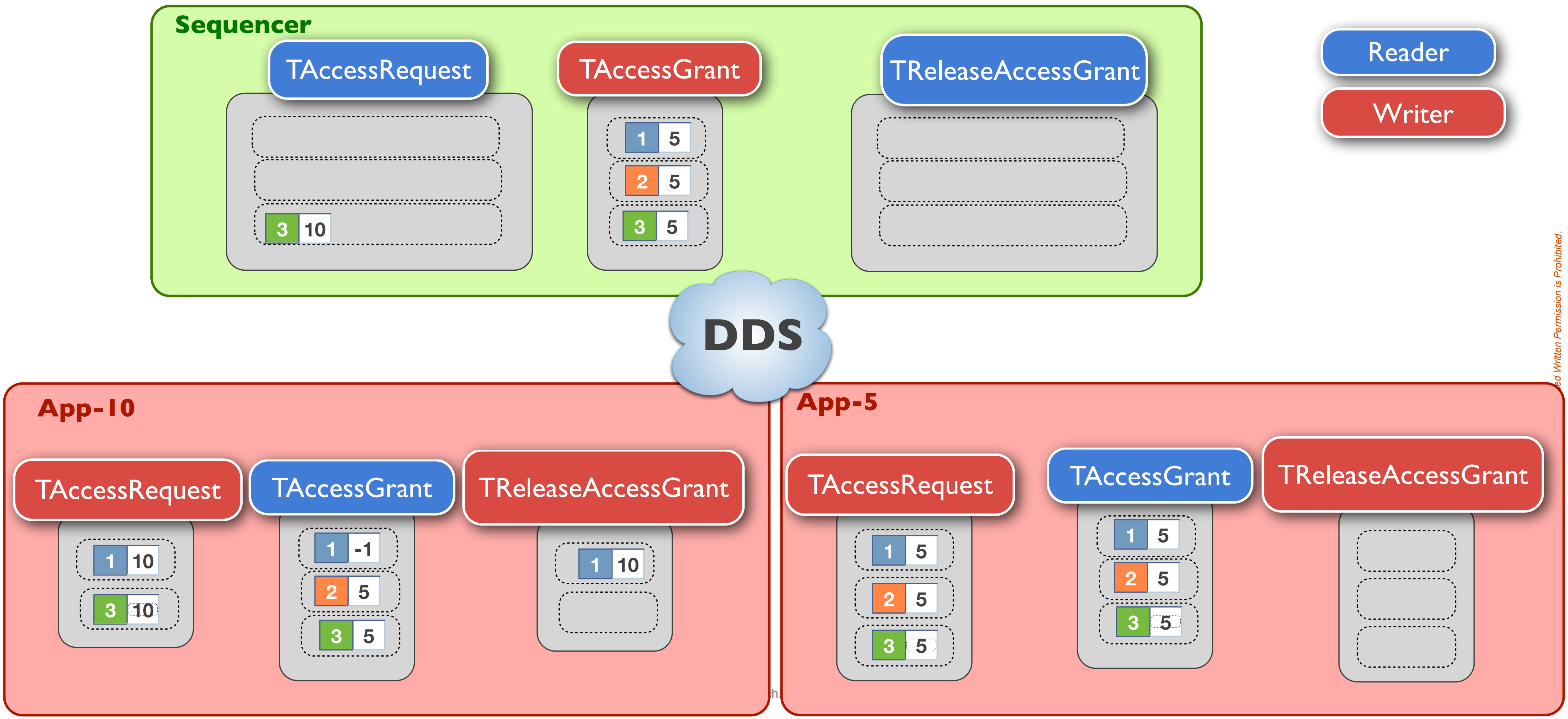
Sequencer @ Work



Sequencer @ Work



Sequencer @ Work



Barriers

Problem

- Often occurs in a distributed system that applications need to reach a common step in computation before proceeding.
- This is required for distributed application creating software pipelines, or even to ensure proper state evolution of the distributed system.
- However DDS does not provide built-in coordination / distributed synchronization mechanisms. How can this be overcome?

Solution

- Use Barriers in your system to coordinate application progress.

Barriers

Detailed Solution

- Define in your system the following Topics:

```
struct TBarrier {
  long computation_guid;
  long task_guid;
  long status;
};
#pragma keylist Barrier computation_guid task_guid
```

```
struct TBarrierCondition {
  long computation_guid;
  long status;
  long cardinality;
};
#pragma keylist Barrier computation_guid
```

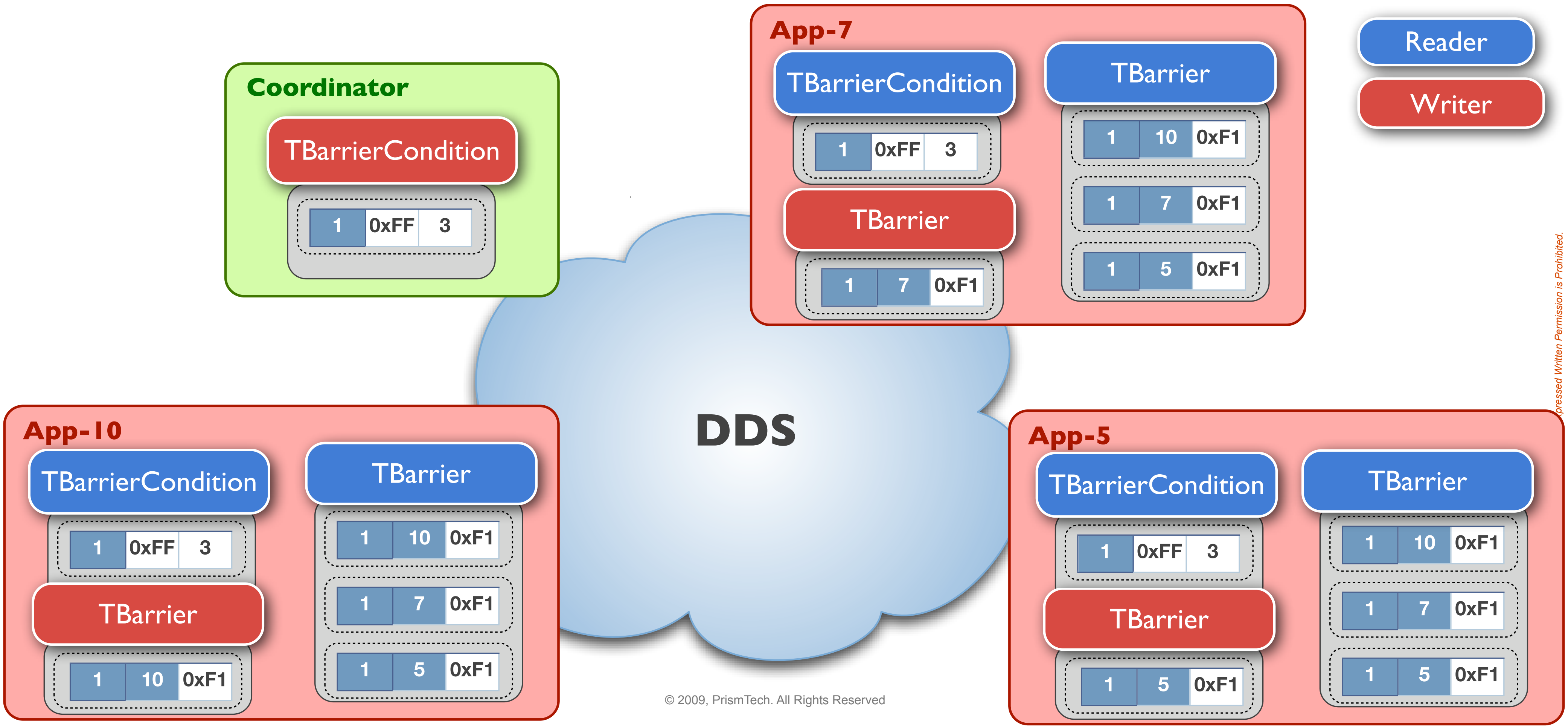
Barriers

Detailed Solution

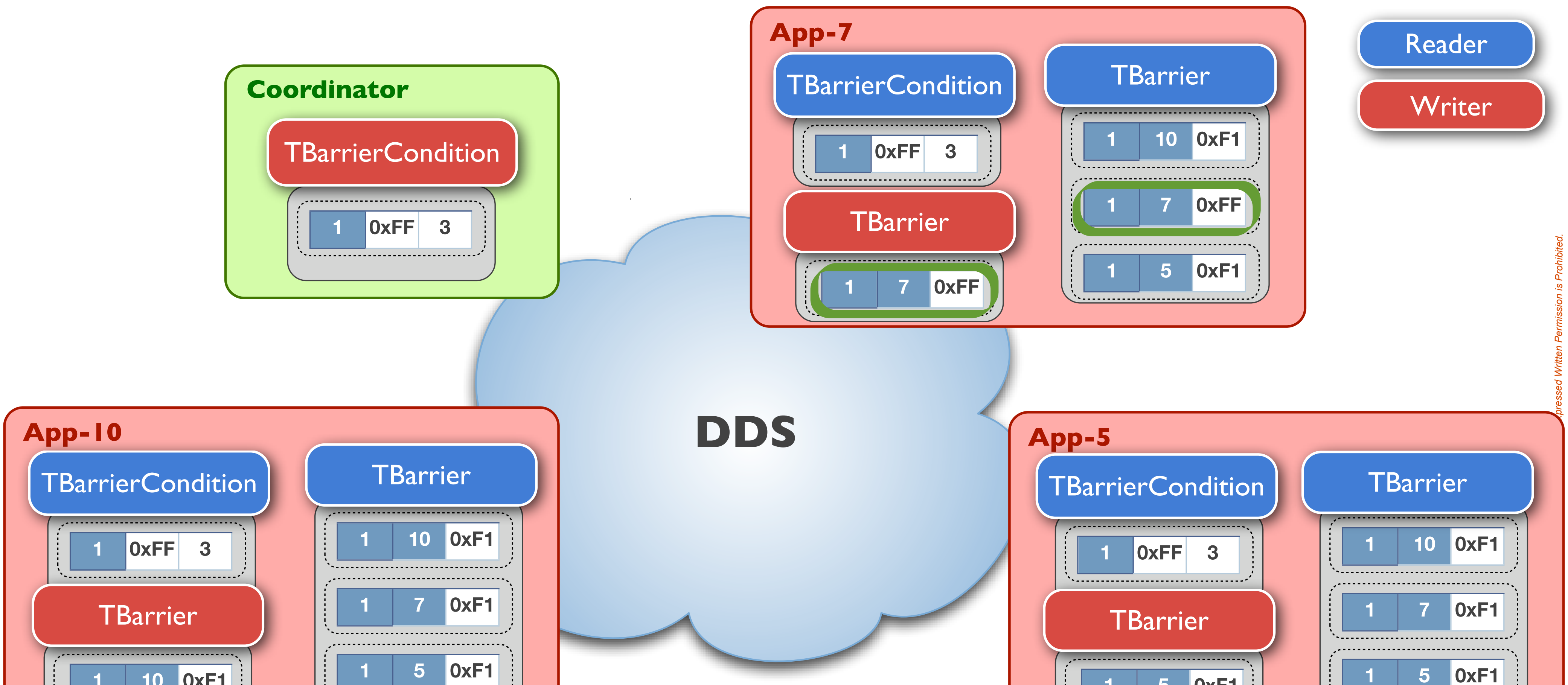
- Make TBarrier and TBarrierCondition a **Topic-Cache**
- Use the following protocol to request/grant/release access

```
Application

// Do Computation

// Notify others
write Barrier

wait for BarrierCondition.cardinality
Barrier instances to have the proper
Barrier.status

// Barrier has been passed...
// Take next computational step
```

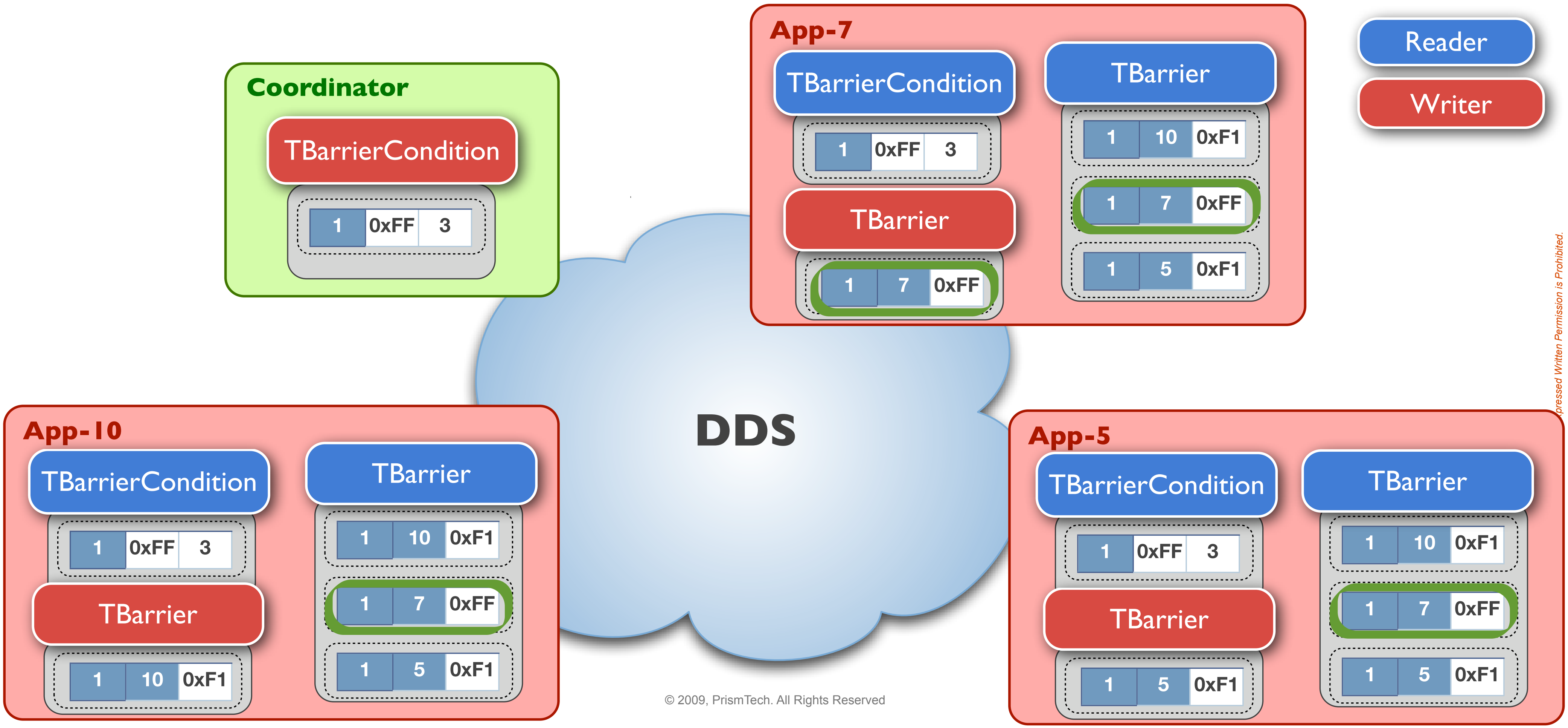
Barriers @ Work



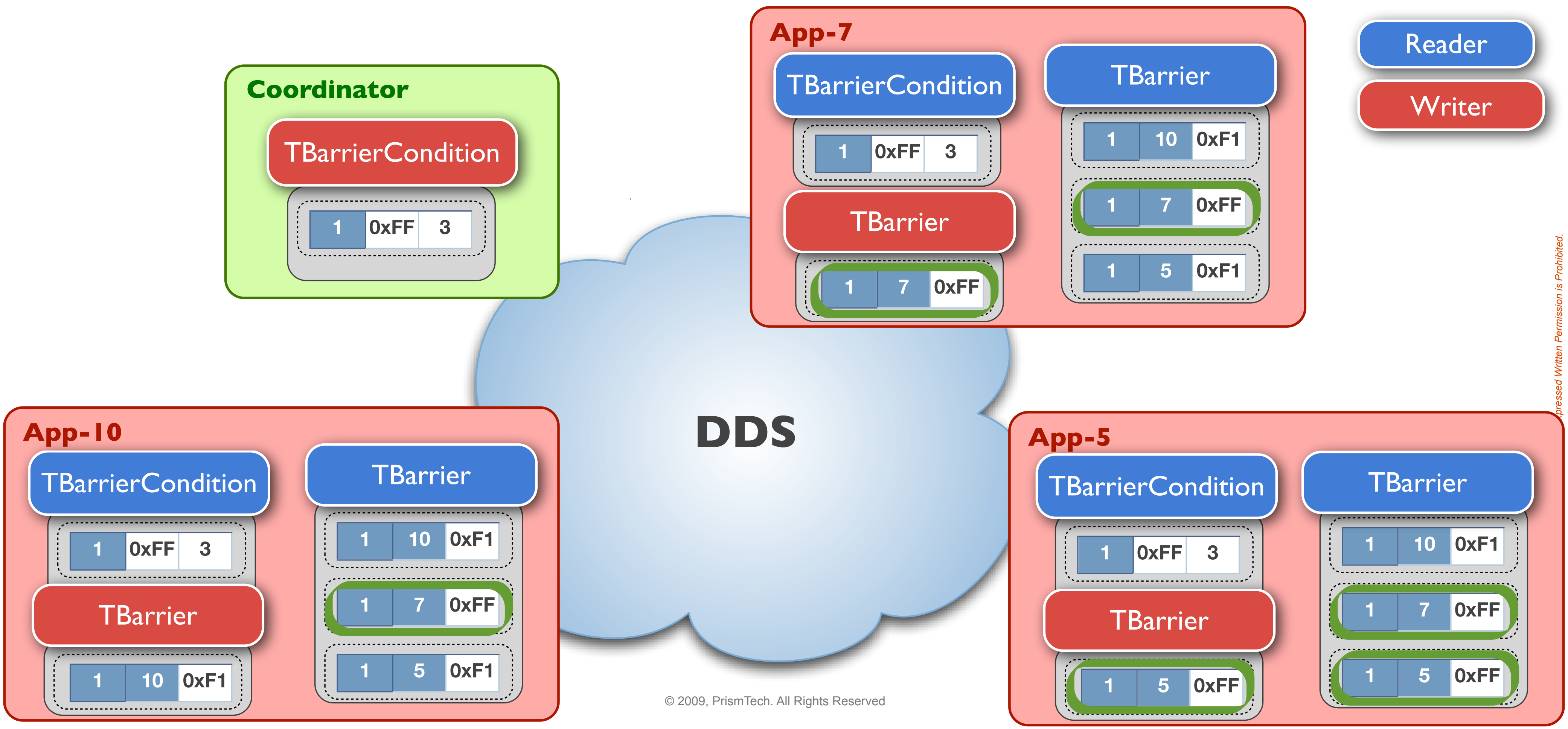
Barriers @ Work



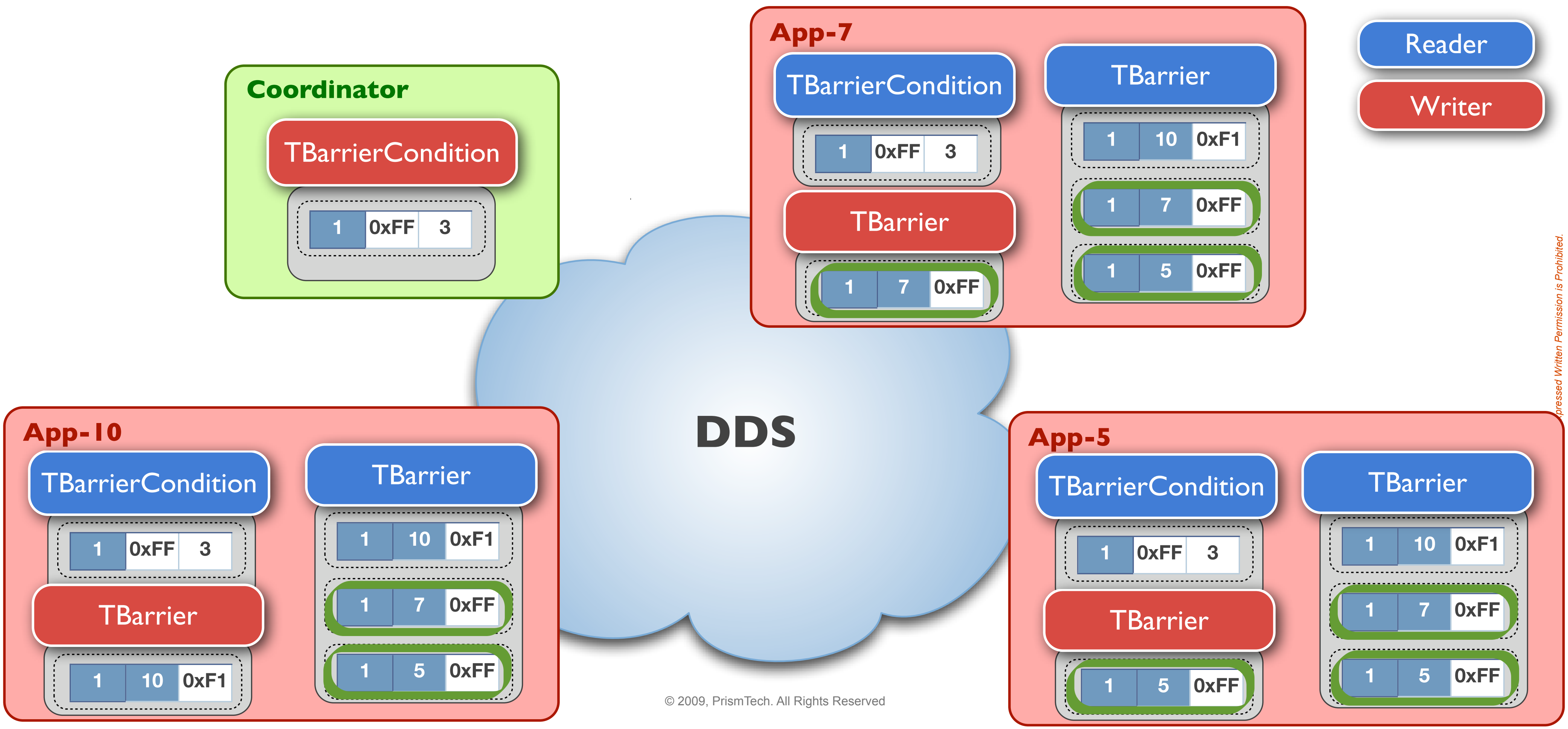
Barriers @ Work



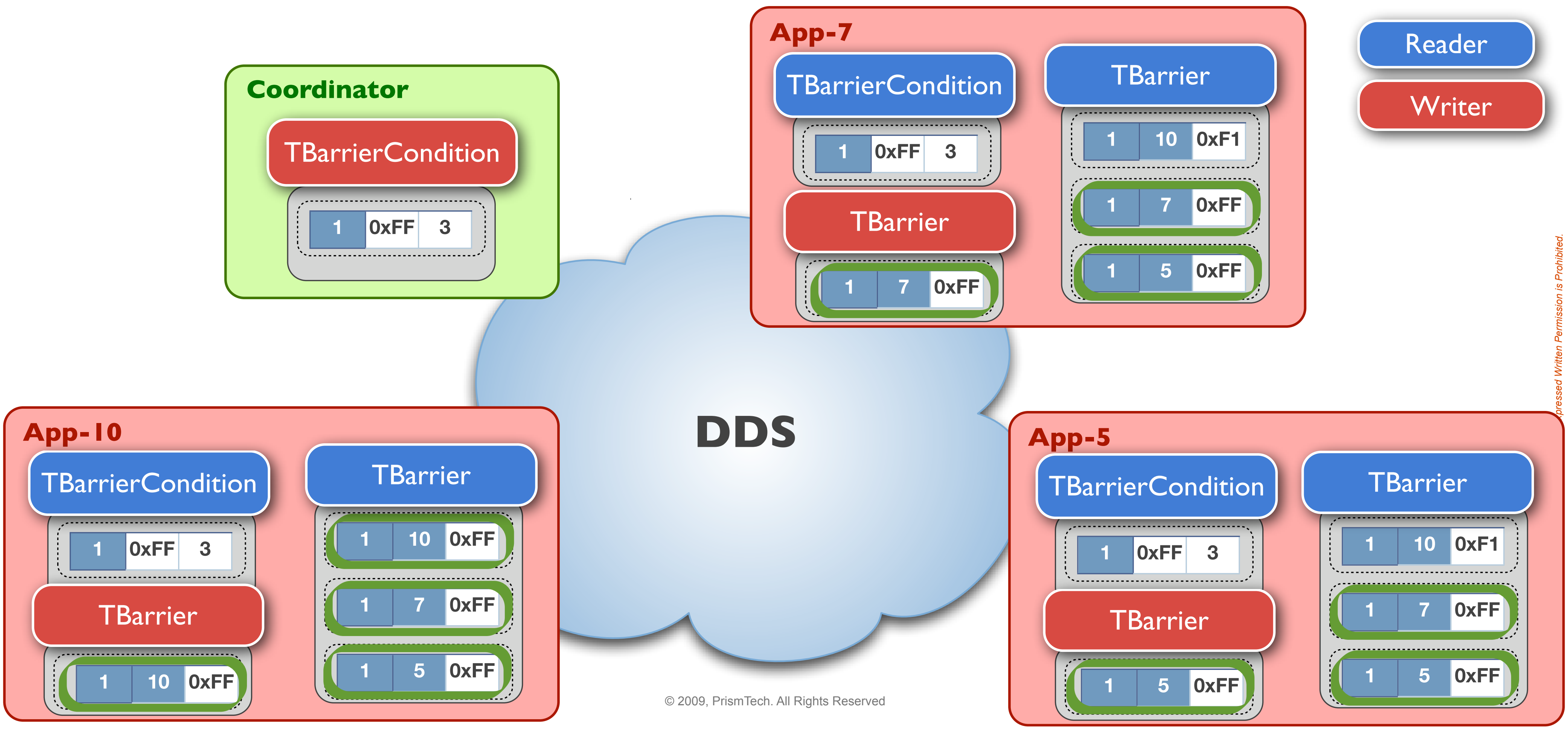
Barriers @ Work



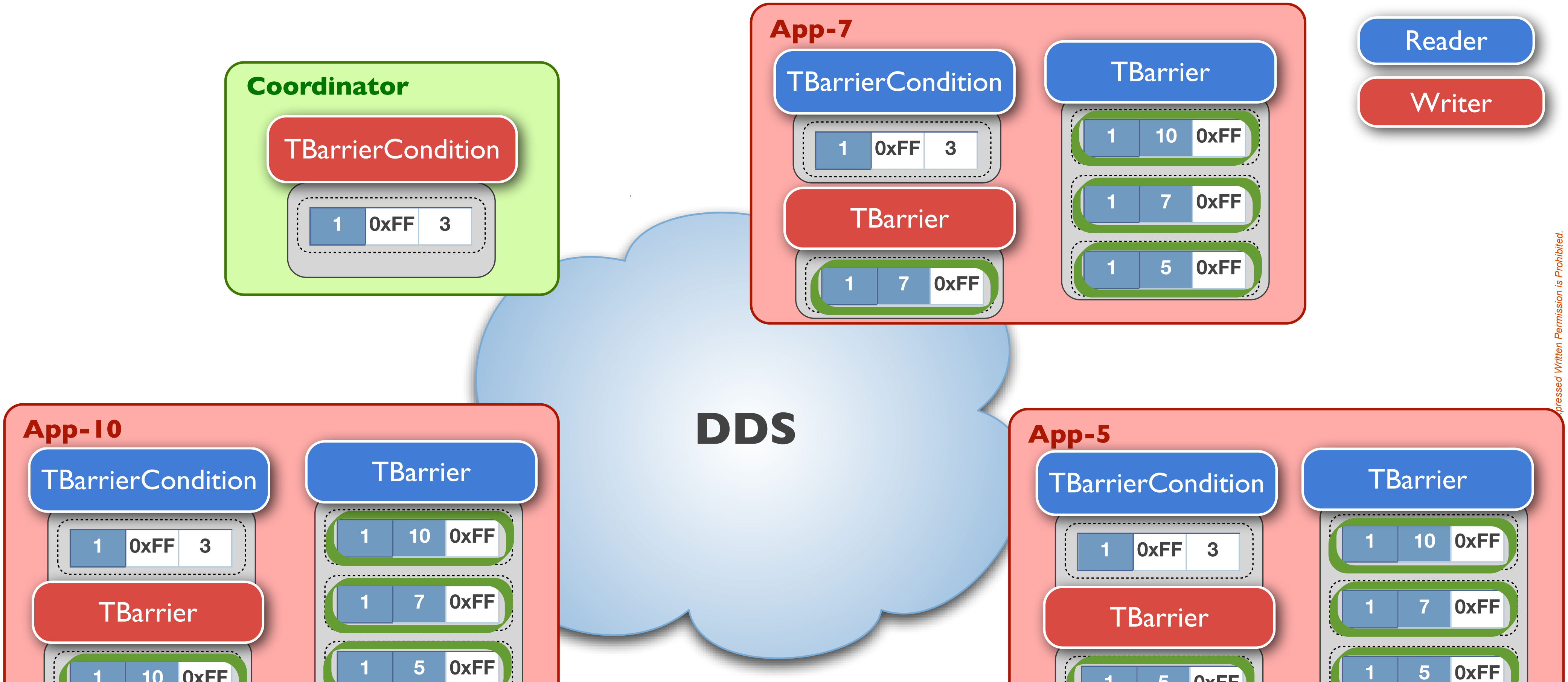
Barriers @ Work



Barriers @ Work



Barriers @ Work



Leader Election

Problem

- Several distributed algorithms require some form of leader
- Problems requiring a leader, can be addressed using the Sequencer Pattern
- However, what if the sequencer crashes?

Solution

- Use the DDS OWNERSHIP_STRENGH as a mechanism to do leader election via DDS

Observation

- The basic Leader Election functionality provided by DDS can be used to easily replicate service implementation such as those of the Sequencer
- If the leader is stateful, its state should be stored within DDS

OpenSplice | DDS

Delivering Performance, Openness, and Freedom

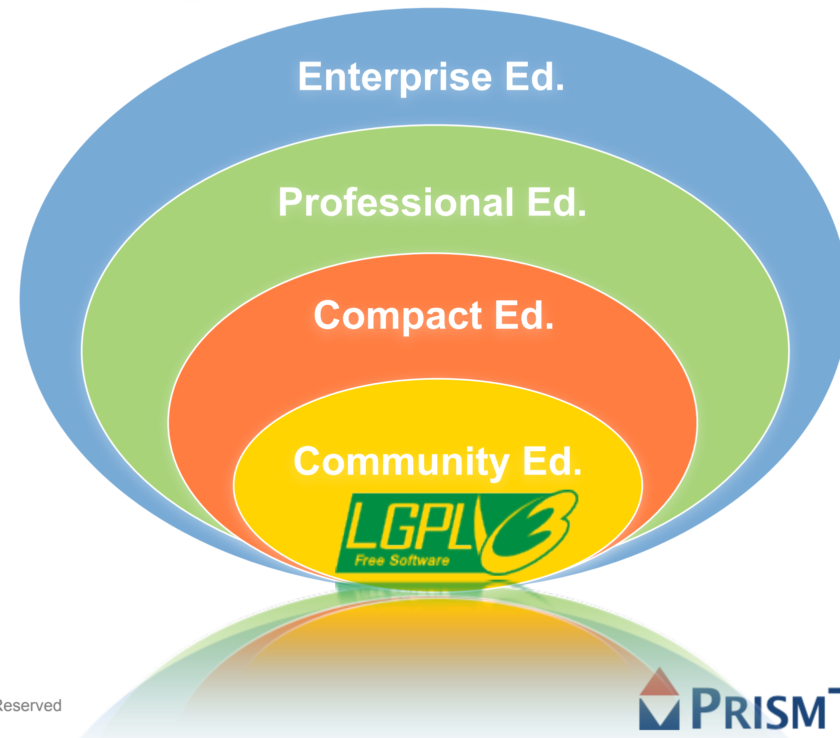
Summing Up

Concluding Remarks

- DDS provides a very powerful infrastructure for building sophisticated distributed systems
- QoS Policies can be used to tune the consistency model at a Topic/Reader/Writer granularity
- Several powerful coordination techniques can be implemented in DDS very efficiently and effectively
- All the Patterns and Techniques presented in this Webcast can be composed (as shown in some instances) to create more sophisticated functionalities

OpenSplice DDS

Delivering Performance, Openness, and Freedom



Online Resources

OpenSplice DDS
Delivering Performance, Openness, and Freedom

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

twitter

* http://twitter.com/acorsaro/

webex

* http://bit.ly/1Sreg

Blogger

* http://opensplice.blogspot.com

YouTube

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

OMG Object Management Group DDS

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