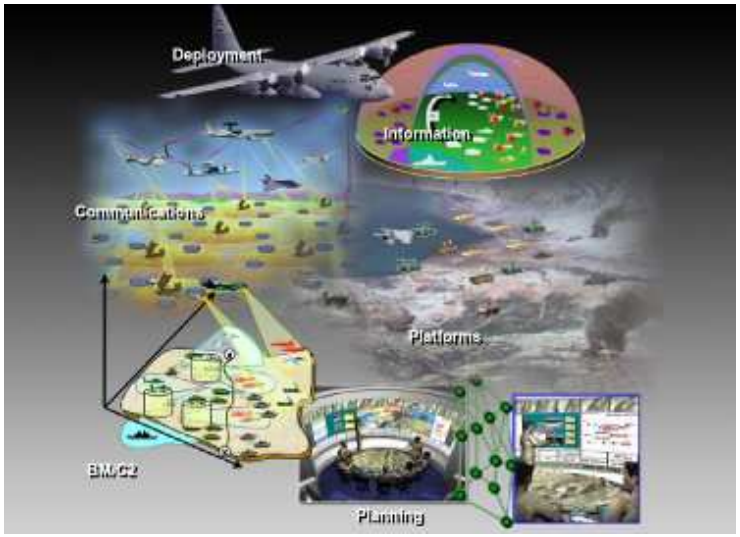
Integrating DDS and COTS Complex-Event Processing Engines

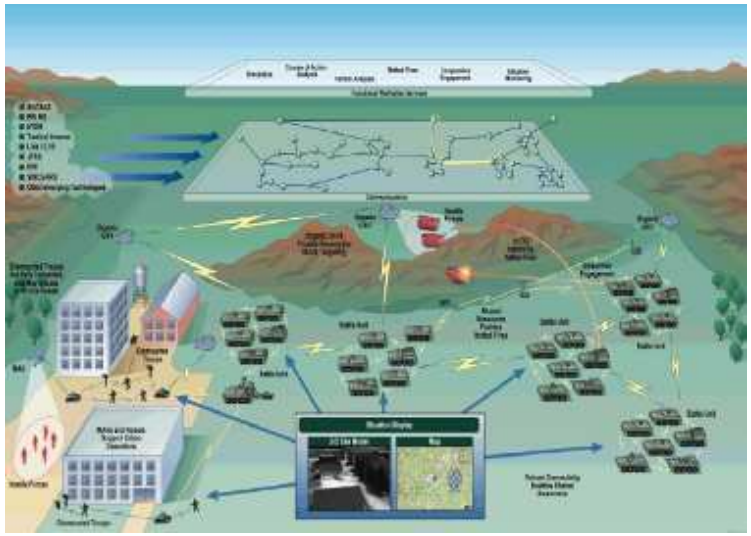
Semantic gap and Performance characteristics

July 2007

Supreet Oberoi, VP Engineering
Gerardo Pardo-Castellote, Ph.D., CTO
Real-Time Innovations, Inc.
[supreet.oberoi , gerardo.pardo]@rti.com
www.rti.com

Introduction - C4I

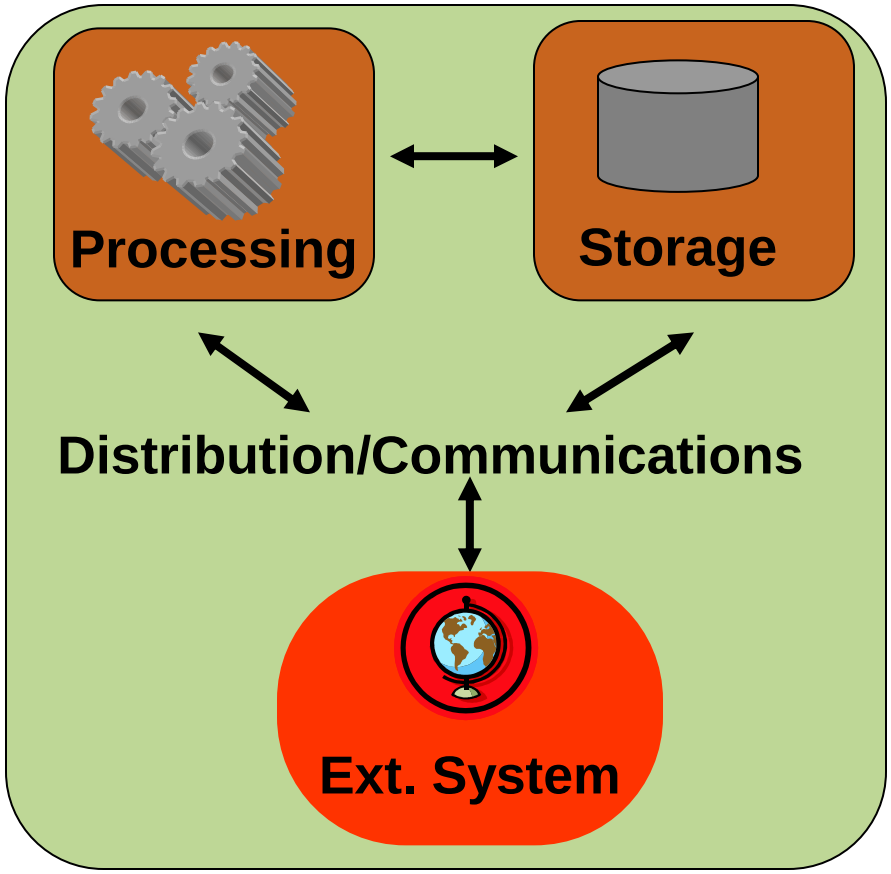




- Many sources of events
- Inferences have to be made in real time
- Monitored events are not tailored to the problem we are trying to solve
- Events need to be correlated to build the information
- Casual and temporal tracking is not easy

The Trinity: Communications + Processing + Storage

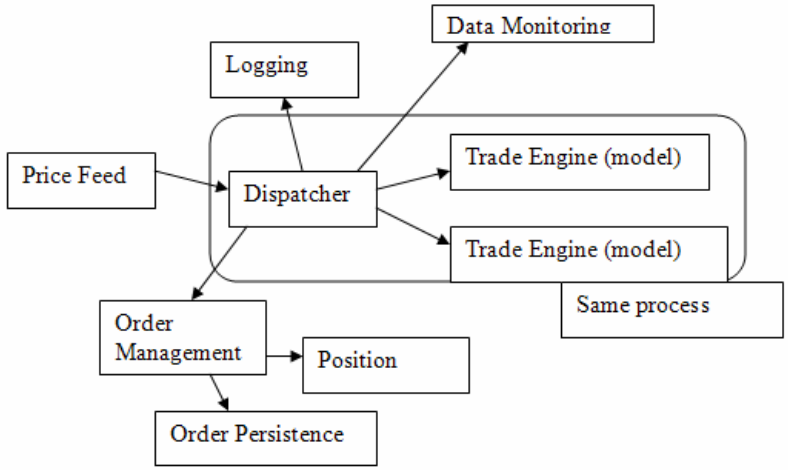
- Processing:
  - Analyze data
  - Perform transformations
  - Generate data/events
- Storage
  - Persist data
  - Organize data
  - Mine data
  - Correlate with old information
- Communications
  - Get the data from external systems
  - Act on external systems
  - Communicate btw Processing and Storage



Which one is more important…??

…depends on your vendor ☺!

Introduction – Market Data Systems



Typical Market Data System

- Real-time data rate > 50k message/sec
  - Cannot "store *then* query"
- System makes trade (buy/sell) decisions by analyzing market data in real-time on short-term trend lines
  - Cannot wait to persist, then process the data
- Trading "algos" get changed/tweaked on a weekly basis

Unresolved Challenges (1/2)

- Need to process data before enterprise-wide use
  - May need to eliminate duplicates, perform syntactic and semantic transformations on real-time data
  - Only some data from real-time systems may be relevant (example: alarm conditions)
- Data needs to be correlated
  - To generate meaningful event data, multiple *streams* and *topics* needs to be correlated
  - Historical and real-time data needs to be correlated
  - Need to correlate topics from other middleware and message-buses
- Lack of time window
  - Raw data may have short half-life; need semantics for time in the correlation queries

RTI

Unresolved Challenges (2/2)

- Need to process data at high-throughput rates
  - Traditional correlation tasks require database persistence -- can lead to:
    - Persistence bloat (not all stored data will be useful)
    - Unacceptable application performance
- Code for event detection is heavy & static
  - Application code to infer events should adapt for short-development lifecycles
  - Developers for writing complex-pattern detection code need rich set of *filters* to infer events

RTI

History of Complex-event Processing

- Before market data explosion, pilot RFID programs caused an explosion of data intended for processing
  - First popular references to terabyte heard
- To avoid this persistence bloat, changed approach from "persist, then process" to "process, then persist, then process"
  - Derived need for an in-memory database to run SQL-like queries to filter, transform data, and enrich data in memory before committing it to physical storage.
- In-memory data sometimes had shelf-life, which caused the need to develop new semantic constructs like windows

RTI

Needs for Event-Management System

- Ability to track casualty of events
- Ability to aggregate low-level events into high-level events of significance
- Ability to rapidly adapt to changes in the environment
  - Design systems to enable on-the-fly event-detection code
  - Decouple event-detection code and event-handling code
- Ability to use time dimension in queries
- Ability to filter and correlate data from different events

RTI

Classification

- Event is classified as an **Event class**
- All operations are defined at the class level
- *classified-to(e, E)* → *instance-of(E, e)*
  - If e is an instance of E, then all the functionality associated with E applied to e
  - e can behave as an instance of E in other ways besides direct classification

Event Classification

`StockTrade` Class

Column	Datatype	Description
tradeid	Integer	a unique sequence number that identifies each trade
symbol	String	the stock symbol associated with this trade (such as MSFT, DELL or ORCL)
volume	Integer	the number of shares exchanged for this trade
price	Float	the price of each share exchanged for this trade

`StockTrade` Instances

timestamp	tradeid	symbol	volume	price
07:15:01	5001	MSFT	500	$29.00
07:15:02	5002	ORCL	1500	$18.00
07:15:03	5003	MSFT	1000	$28.00
07:15:05	5004	MSFT	1000	$31.00
07:15:06	5005	AAPL	500	$81.00
07:15:09	5006	AAPL	800	$82.00
07:15:13	5007	ORCL	500	$20.00
07:15:15	5008	AAPL	2000	$83.00
07:15:16	5009	MSFT	1500	$28.00
07:15:18	5010	AAPL	1000	$83.00
07:15:21	5011	MSFT	500	$29.00

Classification

- Event is classified as an **Event class**
- All operations are defined at the class level
- *classified-to(e, E)* → *instance-of(E, e)*
  - If e is an instance of E, then all the functionality associated with E applied to e
  - e can behave as an instance of E in other ways besides direct classification

Association

- Association is the relationship between two events **and a condition** that describes the reasons for the relationship
  - Creates an additional virtual event class for which there are no events that are directly classified

Instance-of(E, e) AND Association-of(E, E', COND) AND COND → Instance-of(E', e)

- Event model also supports uncertain generalization and association based on a certainly value that designates the strength of the relationship

Event-processing Semantics

Use Case	Continuous Computation Language (CCL) Queries

Windows & Aggregation	`INSERT INTO StreamVWAP`
	`SELECT Symbol, SUM(Price*Volume)/SUM(Volume)`
	`FROM StreamTrades KEEP 5 MINUTES`
	`GROUP BY Symbol`
	`OUTPUT EVERY 1 MINUTE`

Correlations	`INSERT INTO CombinedStockOption`
	`SELECT InStock.Symbol, InOption.OptionSymbol,`
	`InStock.Price, InOption.Price`
	`FROM InOption, InStock KEEP 10 SECONDS`
	`WHERE InStock.Symbol=InOption.StockSymbol`

Event Pattern Matching	`INSERT INTO ProcessAlerts`
	`SELECT StreamA.id`
	`FROM StreamA a, StreamB b, StreamC c, StreamD d`
	`MATCHING [10 SECONDS: a, b || c, !d]`
	`ON a.id = b.id = c.id = d.id`

Windows Semantics

```
-- A window to keep data for ten minutes, populated by stream StreamClicks

CREATE WINDOW RecentHistory(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP 10 MINUTES;

INSERT INTO RecentHistory
SELECT *
FROM StreamClicks;

-- A window to keep the last value for each IPAddress:

CREATE WINDOW LastValues(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP LAST PER IPAddress;

-- A window that keeps the 1000 largest downloads. Each download is kept for one
hour, or until a larger download displaces it from the window:

CREATE WINDOW LargestDownloads(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP LARGEST BY FileSize KEEP 1 HOUR;
```

Aggregation Semantics

```
-- Compute one-minute "bars" (Avg, Max, Min, Closing) and VWAP (Volume-Weighted
-- Average Price). Output results continuously, as soon as results change.

INSERT INTO OneMinuteBarView
SELECT Symbol, AVG(Price), MAX(Price), MIN(Price), SUM(Price*Volume)/SUM(Volume)
FROM StreamFeed KEEP 1 MINUTE;

-- Determine the total bandwidth consumed by each IP address for the
-- last 1000 downloads. Output results every 15 second.

INSERT INTO SteramConsumedBandwidht
SELECT IP, SUM(FileSize)
FROM StreamDownloadLog KEEP 1000 ROWS
GROUP BY IP
OUTPUT EVERY 15 SECONDS;
```

Real-time And Historical Data

- Context to real-time data often exists in historical sources like RDBMS
  - Example: Mapping of RFID reader ID → Product SKU ID
- CEP enables correlating data stream with RDBMS

Table: `RFID_SKU_MAP`

Column	DataType	Description
RFID	Integer	RFID reader tag
SKU_ID	Integer	Product SKU ID for RFID

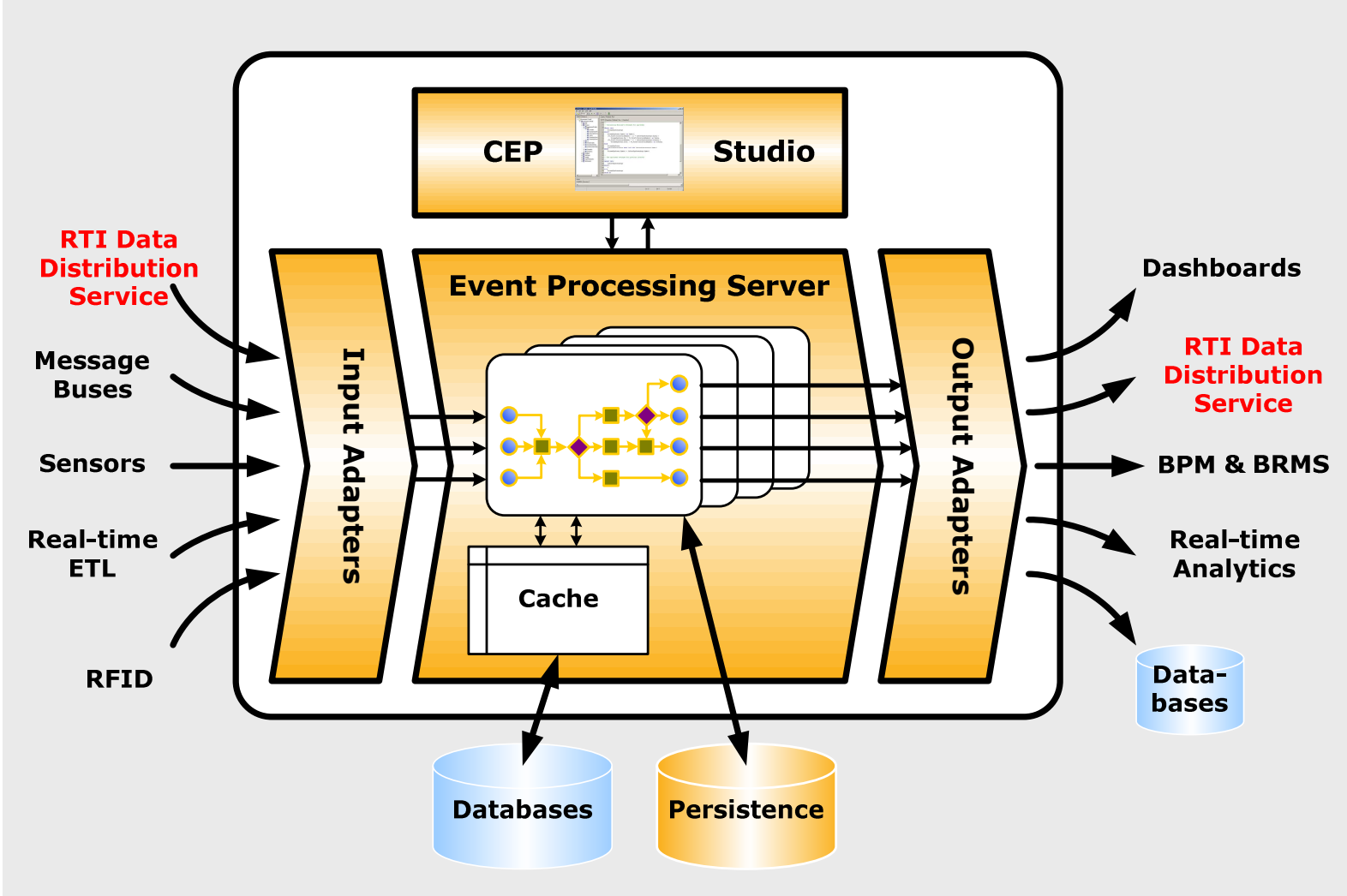
```
INSERT INTO RFIDWithSKU
SELECT current.rfid, current.location, product.sku_id
FROM RFIDReader AS current
(DATABASE 'ProductCatalog' SCHEMA (SKU_ID integer)
[[SELECT sku_id FROM rfid_sku_map
WHERE rfid = ?current.rfid]]) AS product
```

Different Approaches to CEP

- SQL based
  - Input and output streams are modeled as database tables. SQL with extended semantics (for time, causality, pattern matching) provided to generate events.
  - Examples: Coral8, StreamBase, Aleri, Esper
- Database-centric approach
  - Have running queries inside the database. First persist the data, the process to generate events
  - Example: Stored PL/SQL procedures and Triggers in Oracle
- Rules based
  - Use tradition rules-engine semantics to identify and generate events
  - Examples: Tibco BusinessEvents, IBM Amit (Research), Rulecore
- Application based
  - UI, Proprietary API and configuration files to manage events
  - Examples: Apama, Vyahu

RTI

“Stream-oriented” CEP Architecture



Middleware Integration

The middleware is responsible for delivering data to the CEP engine, and for disseminating the events from the CEP engine. As a result, a suitable middleware should have the following characteristics:

-1- Low latency and High-throughput

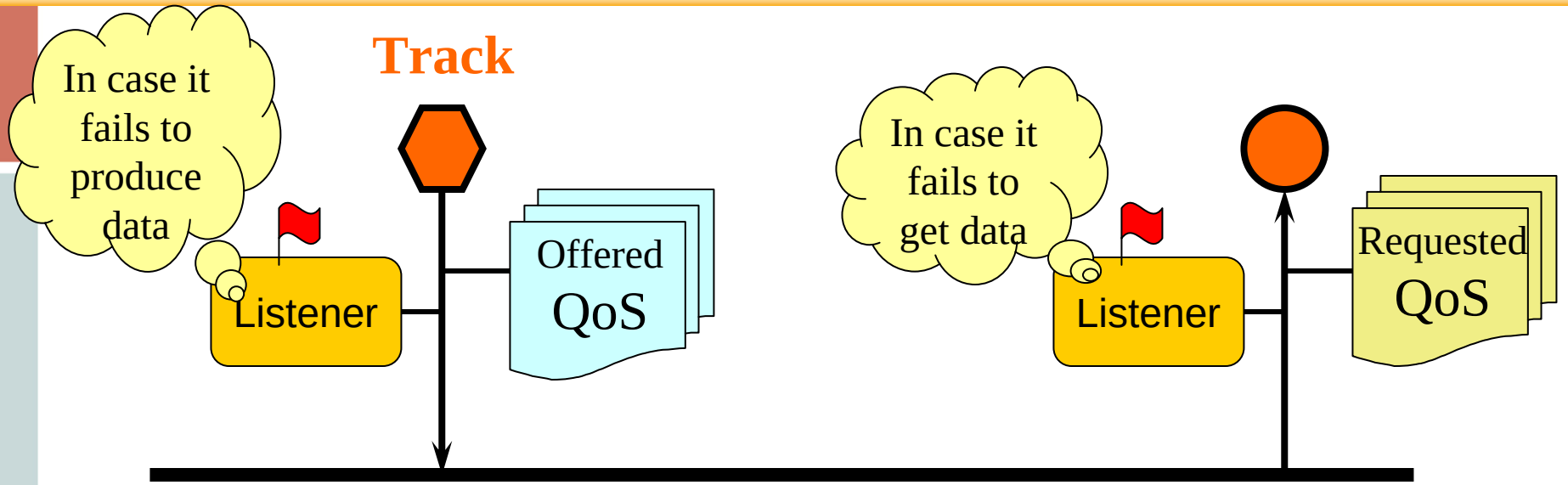
-2- Time and Content-based filter

-3- Ownership strength (ability to support multiple data feeds)

-4- Persistence (ability to provide fault-tolerance for data feeds)

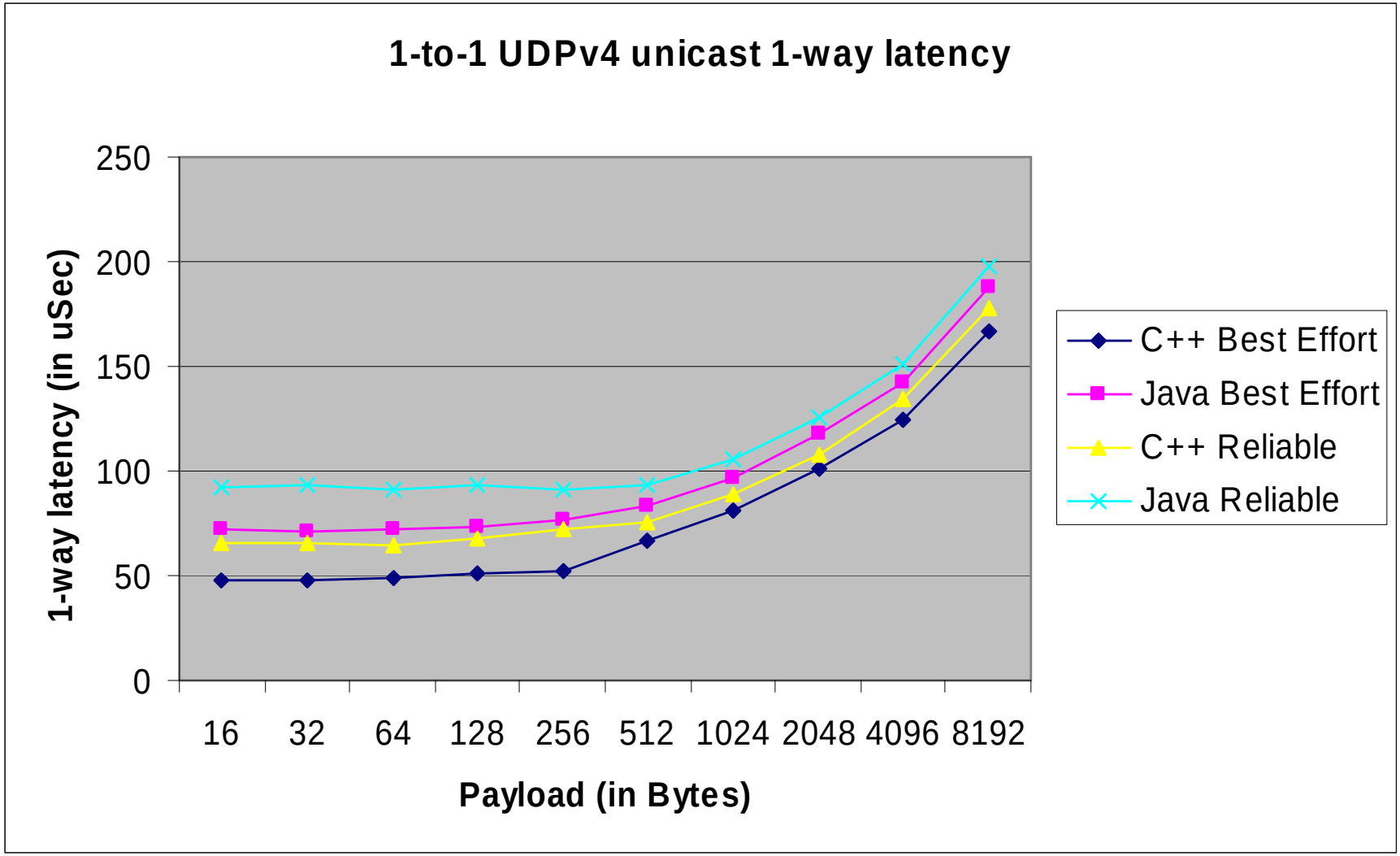
-5- History (ability for a late-joining consumer to get previous data)

Introducing Data Distribution Service

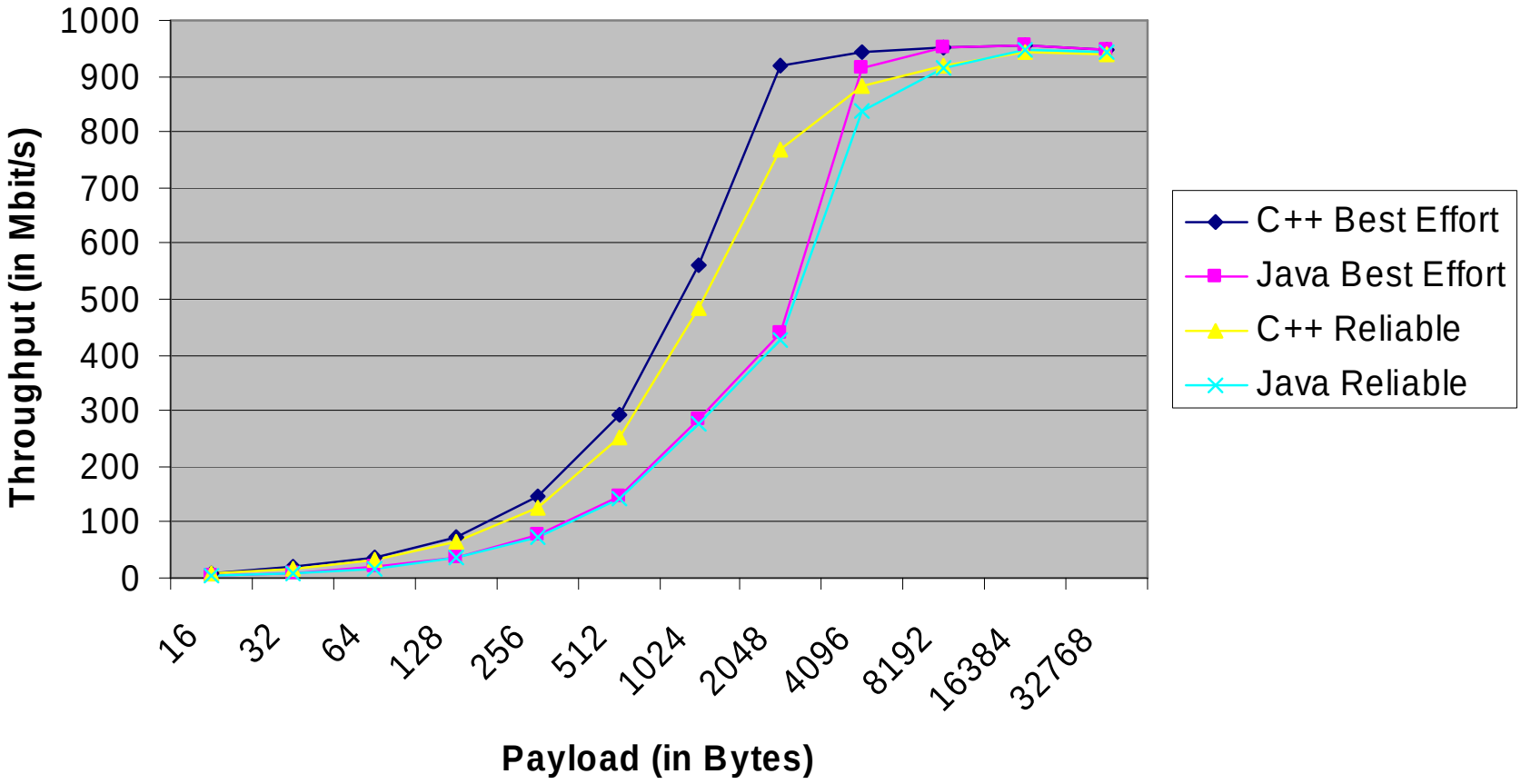


- Publisher declares information it has by specifying the Topic…
  - …and the offered QoS contract
  - …and an associated listener to be alerted of any significant status changes
- Subscriber declares information it wants by specifying the Topic…
  - …and the requested QoS contract
  - …and an associated listener to be alerted of any significant status changes
- DDS automatically discovers publishers and subscribers
  - DDS ensures QoS matching and alerts of inconsistencies

Latency

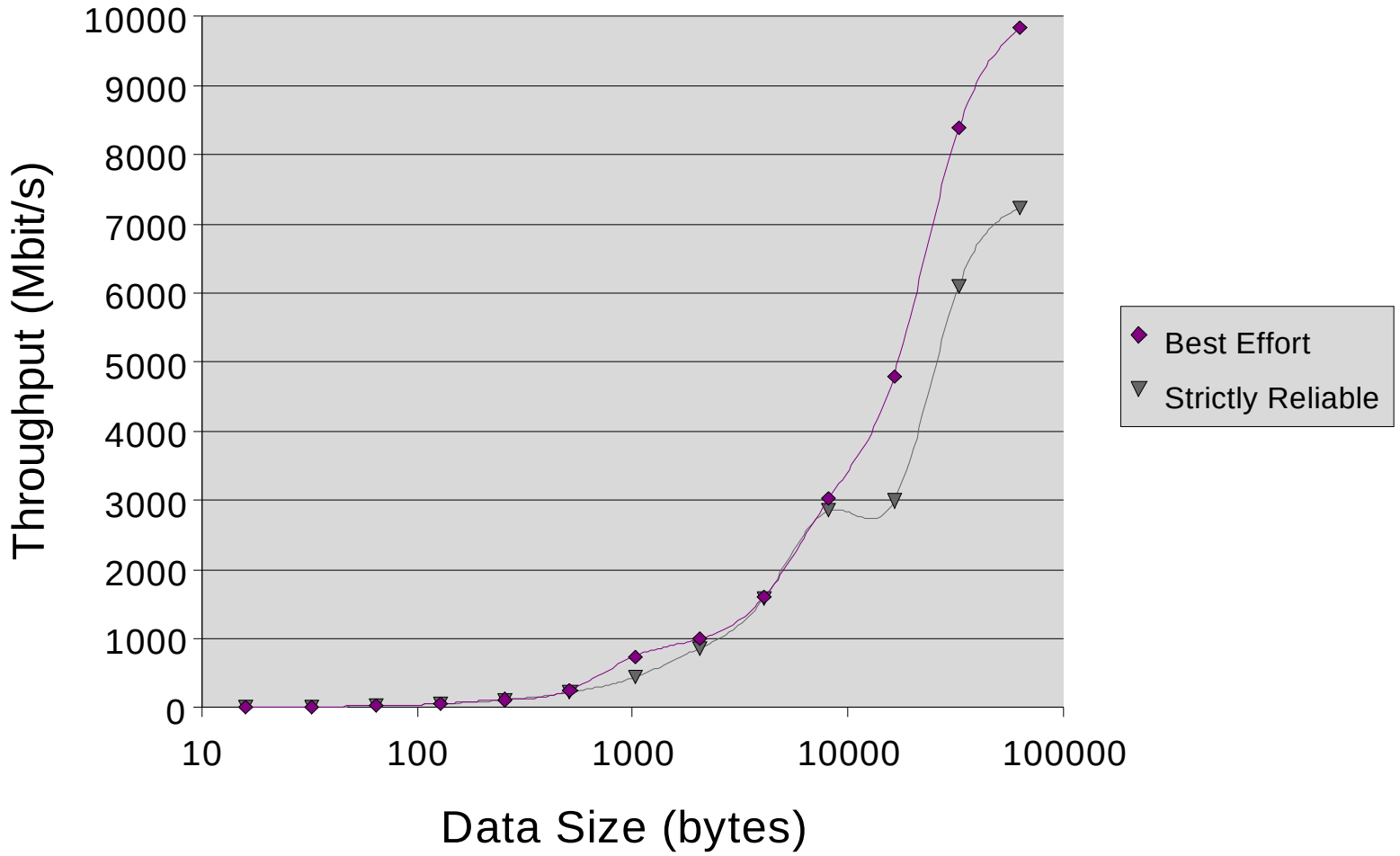

1-to-1 UDPv4 unicast 1-way latency
1-way latency (in uSec)
250
200
150
100
50
0
16
32
64
128
256
512
1024
2048
4096
8192
Payload (in Bytes)
C++ Best Effort
Java Best Effort
C++ Reliable
Java Reliable

Throughput (1/2)



Throughput (2/2)

1-to-1 Unicast IP over Infiniband Throughput



Latency for CEP

- Latencies of around 100 microseconds for ***small*** messages (12 fields or so) and a ***pass-through or filter*** query.
  - Message size makes a difference.
- The CCL query makes a huge difference.
  - Typically, we see sub-millisecond latencies with moderate size queries.
  - But we also see tens of millisecond maximum latency when an external RDBMS query is involved.

Throughput for CEP

- Typically, see thousands of simple messages (100-150Bytes) per second through ***simple*** queries.
  - Message size makes a difference.
- The CCL query makes a huge difference. Typically we see 10's of thousands through moderate sized set of queries, while simple queries might be 400K. This is all on a single processor.

Sample Benchmark (Internal)

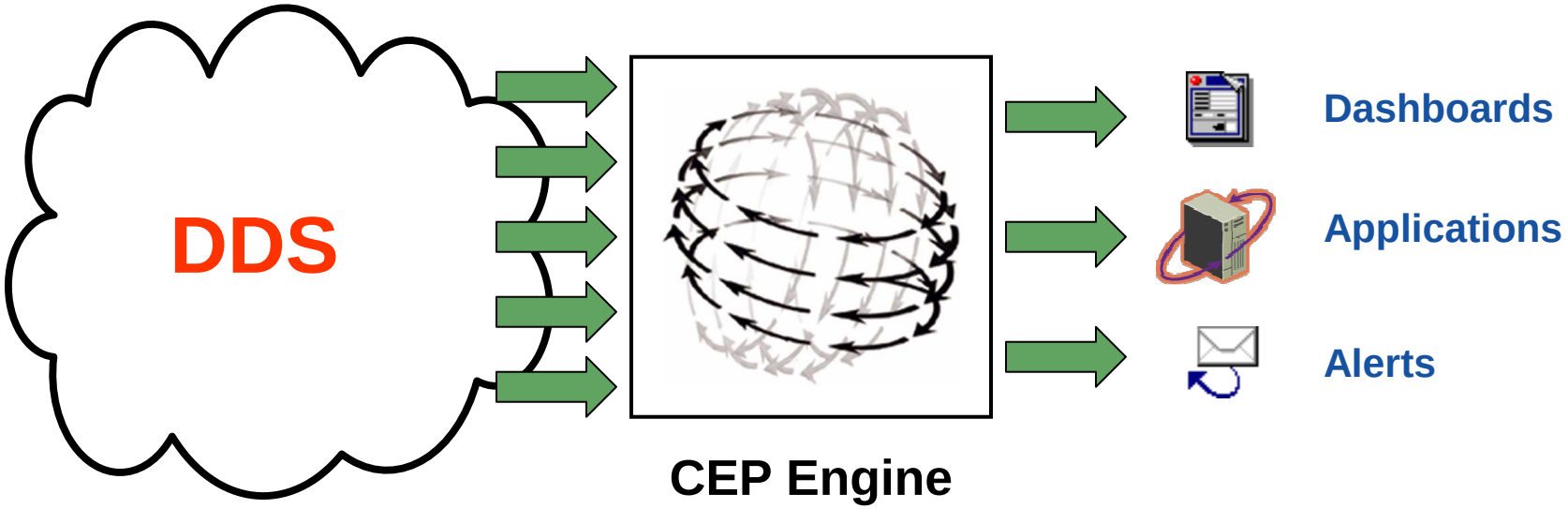
CONFIGURATION:

Computer: Intel(R) Pentium(R) 4 CPU 3.00GHz 2.99 GHz, 1.00GB of RAM
OS: WINDOWS XP
Database: Oracle 10.1.0
Current as of: January 2007

Benchmark Query	Messages per Second
Filter	380,000
Aggregation	105,000
Inner Join	130,000
Pattern Matching	54,000
Database Subquery	19,000

DDS Ideal for CEP When…

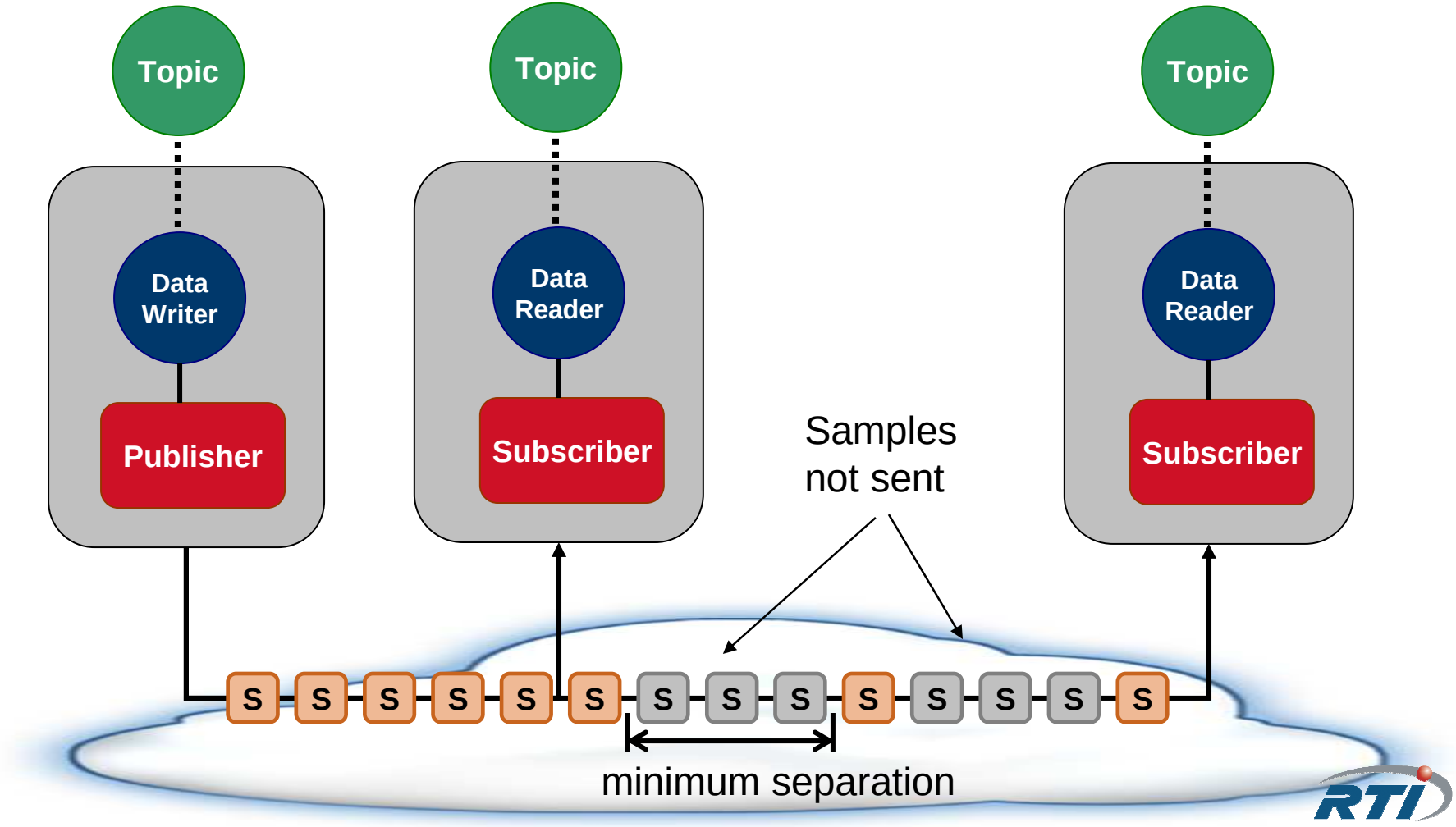
- Applications have the following characteristics:
  - Require data centricity for event dissemination
  - Use high-speed data streams (1,000-1,000,000 msg/sec)
  - Require latency measured in sub-milliseconds
  - Demand access to events from a heterogeneous distributed systems, including embedded systems



Backup

QoS: Time Based Filter

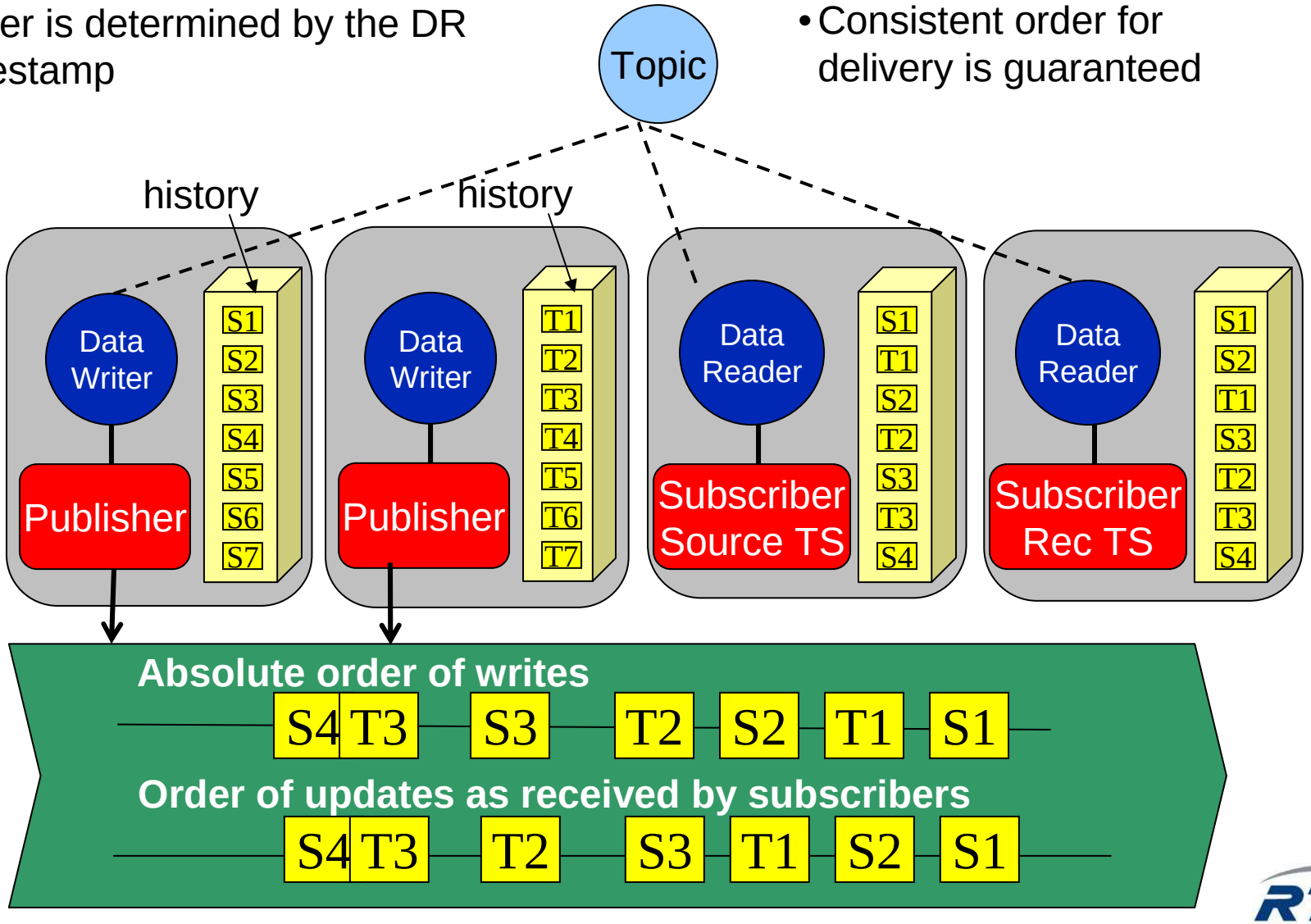
"minimum_separation": DataReader does not want to receive data faster than the min_separation time



QoS: Destination_Order

BY_RECEPTION_TIMESTAMP
- Order is determined by the DR timestamp

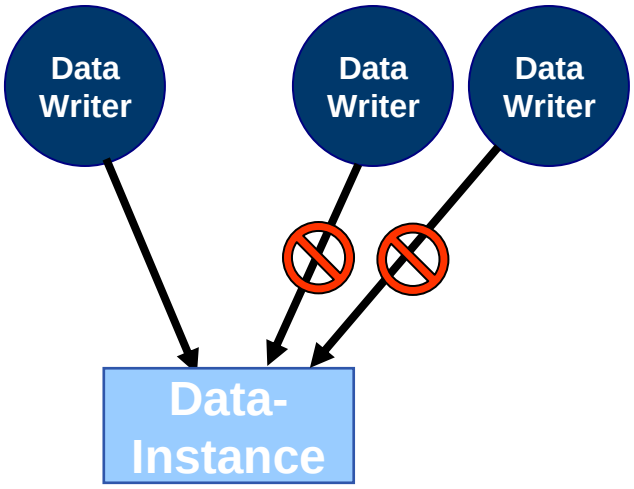
BY_SOURCE_TIMESTAMP
- Consistent order for delivery is guaranteed



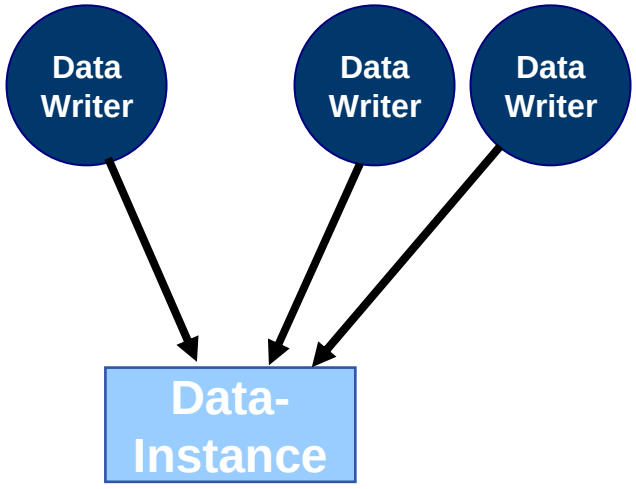
QoS: Ownership

Specifies whether more than one DataWriter can update the same instance of a data-object

Ownership = EXCLUSIVE
"Only highest-strength data writer can update each data-instance"



Ownership = SHARED
"All data-writers can each update data-instance"

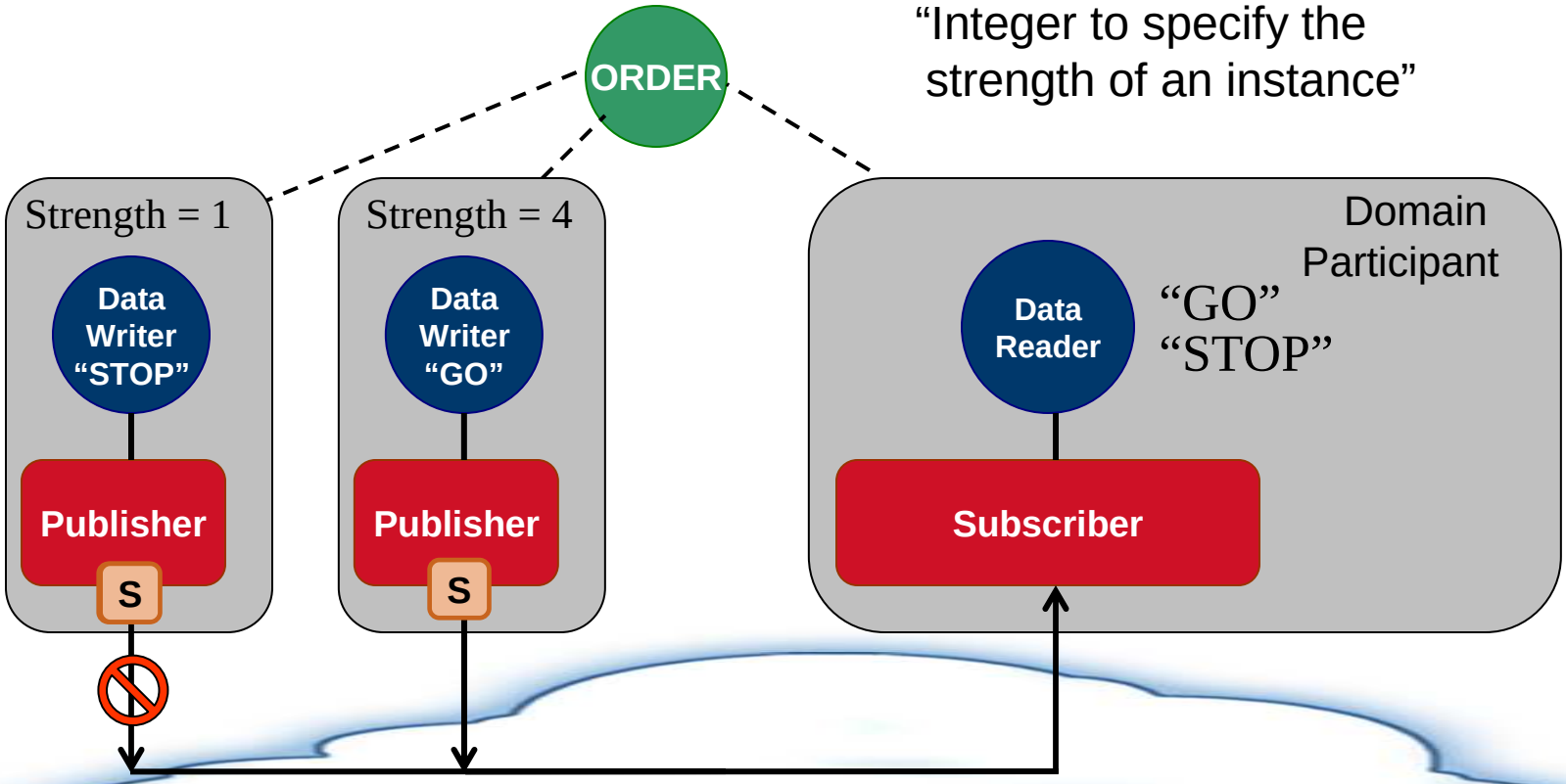


Provides fast, robust, transparent replacement for fail-over and/or take-over.

QoS: Ownership Strength

Specifies which DataWriter is allowed to update the values of data-objects

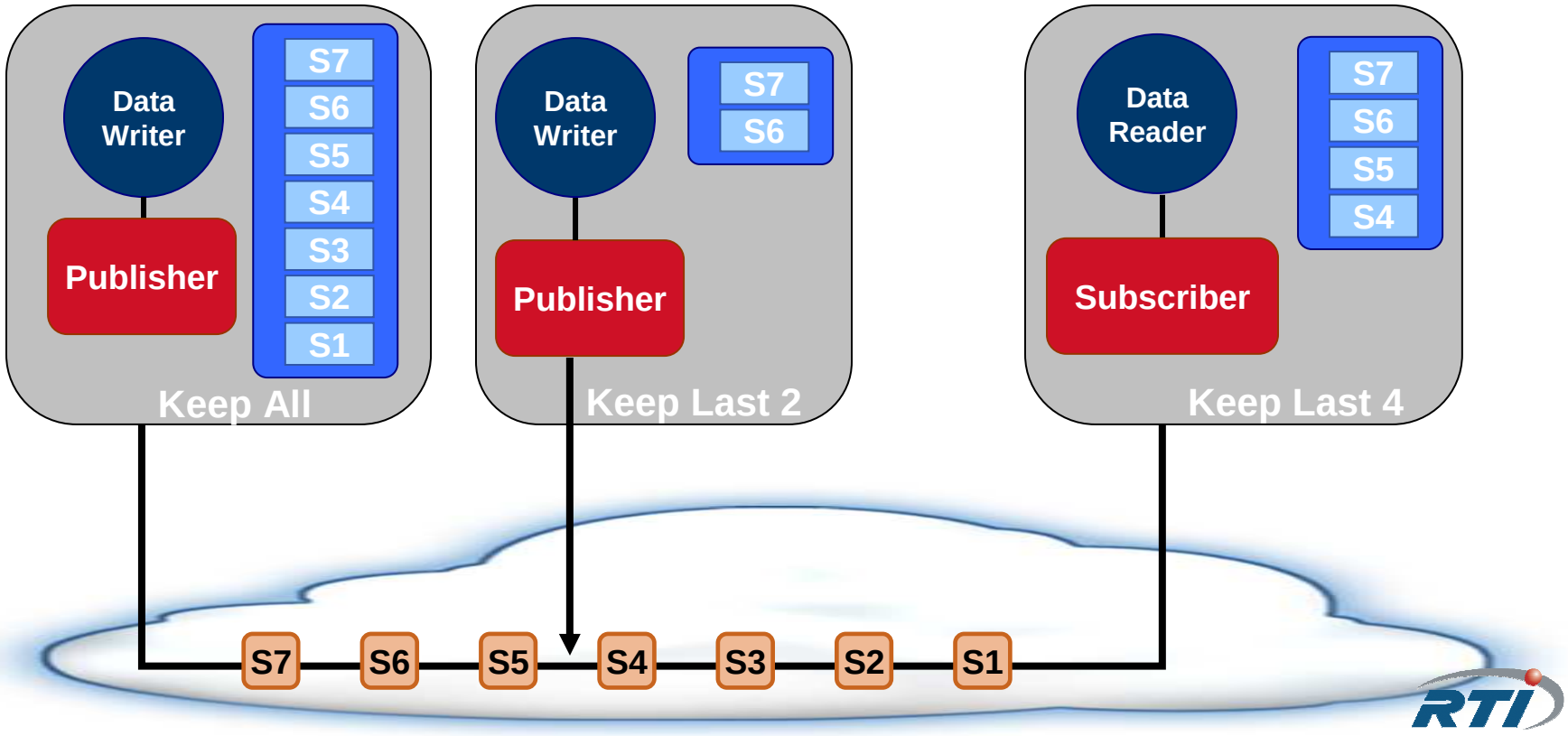
OWNERSHIP_STRENGTH
"Integer to specify the strength of an instance"



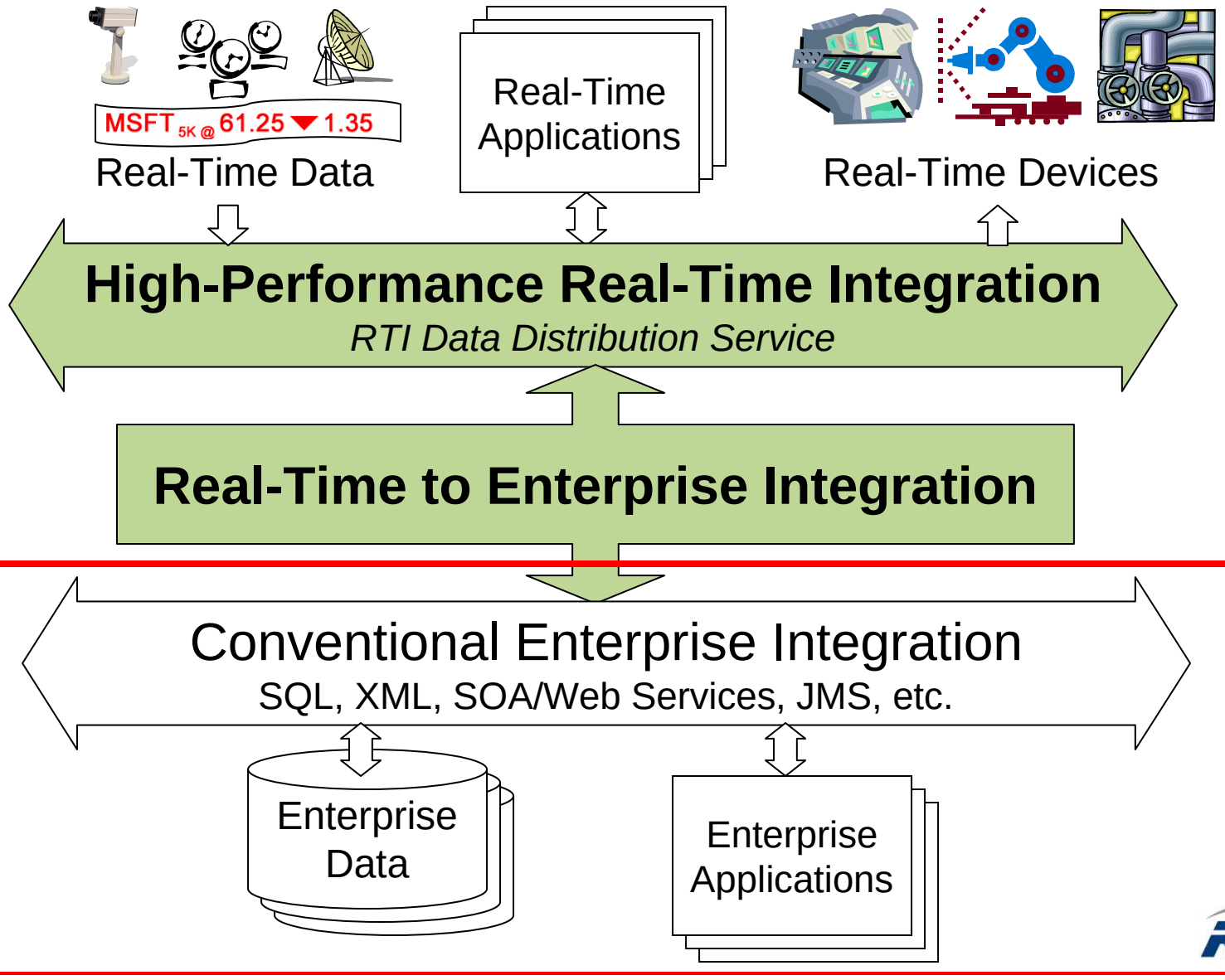
QoS: History – Last x or All

KEEP_ALL:
Publisher: keep all until delivered
Subscriber: keep each sample until the application processes that instance

KEEP_LAST: "depth" integer for the number of samples to keep at any one time



Data Distribution Service



Aggregation

- Event can be viewed as a set of attributes that have values specific to the event
- Attributes are defined at class level
- Attributes can have different types:
  - Numeric
  - Strings
  - References to Events and data structures

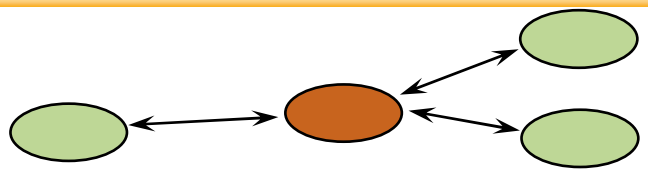
Generalization

- Relationship in which one Event class is based on another Event class.
- Child Event in generalizations inherits the attributes, operations, and relationships of the parent Event
- Event model supports **Conditional Generalization**

Instance-of(E, e) AND Generalization-of(E, E') AND COND → Instance-of(E', e)

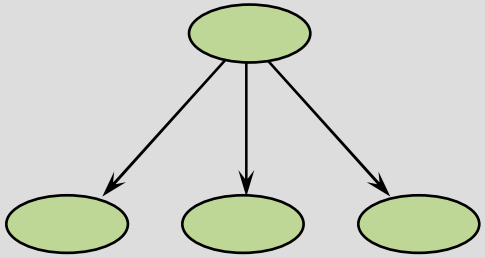
Middleware Information Models

Point-to-Point
Telephone, TCP
Simple, high-bandwidth
Leads to stove-pipe systems

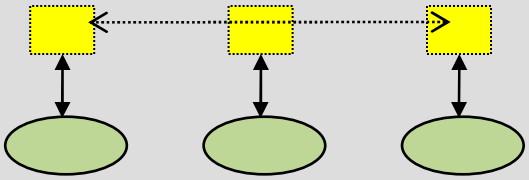


Client-Server
File systems, Database, RPC, CORBA, DCOM
Good if information is naturally centralized
Single point failure, performance bottlenecks

DDS & DDM



Publish/Subscribe Messaging
Magazines, Newspaper, TV
Excels at *many-to-many communication*
Excels at distributing *time-critical information*



Replicated Data
Libraries, Distributed databases
Excels at data-mining and analysis

Introduction To CCL (1/2)

Column	Datatype	Description
tradeid	Integer	a unique sequence number that identifies each trade
symbol	String	the stock symbol associated with this trade (such as MSFT, DELL or ORCL)
volume	Integer	the number of shares exchanged for this trade
price	Float	the price of each share exchanged for this trade

`StockTrade` Stream

timestamp	tradeid	symbol	volume	price
07:15:01	5001	MSFT	500	$29.00
07:15:02	5002	ORCL	1500	$18.00
07:15:03	5003	MSFT	1000	$28.00
07:15:05	5004	MSFT	1000	$31.00
07:15:06	5005	AAPL	500	$81.00
07:15:09	5006	AAPL	800	$82.00
07:15:13	5007	ORCL	500	$20.00
07:15:15	5008	AAPL	2000	$83.00
07:15:16	5009	MSFT	1500	$28.00
07:15:18	5010	AAPL	1000	$83.00
07:15:21	5011	MSFT	500	$29.00

Introduction To CCL (2/2)

Use Case	Continuous Computation Language (CCL) Queries

Windows & Aggregation	`INSERT INTO StreamVWAP`
	`SELECT Symbol, SUM(Price*Volume)/SUM(Volume)`
	`FROM StreamTrades KEEP 5 MINUTES`
	`GROUP BY Symbol`
	`OUTPUT EVERY 1 MINUTE`

Correlations	`INSERT INTO CombinedStockOption`
	`SELECT InStock.Symbol, InOption.OptionSymbol,`
	`InStock.Price, InOption.Price`
	`FROM InOption, InStock KEEP 10 SECONDS`
	`WHERE InStock.Symbol=InOption.StockSymbol`

Event Pattern Matching	`INSERT INTO ProcessAlerts`
	`SELECT StreamA.id`
	`FROM StreamA a, StreamB b, StreamC c, StreamD d`
	`MATCHING [10 SECONDS: a, b || c, !d]`
	`ON a.id = b.id = c.id = d.id`

CCL Example: Windows

```
-- A window to keep data for ten minutes, populated by stream StreamClicks

CREATE WINDOW RecentHistory(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP 10 MINUTES;

INSERT INTO RecentHistory
SELECT *
FROM StreamClicks;

-- A window to keep the last value for each IPAddress:

CREATE WINDOW LastValues(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP LAST PER IPAddress;

-- A window that keeps the 1000 largest downloads. Each download is kept for one
hour, or until a larger download displaces it from the window:

CREATE WINDOW LargestDownloads(IPAddress LONG, URLvisite STRING, FileSize LONG)
KEEP LARGEST BY FileSize KEEP 1 HOUR;
```

CCL Example: Aggregation

```
-- Compute one-minute "bars" (Avg, Max, Min, Closing) and VWAP (Volume-Weighted
-- Average Price). Output results continuously, as soon as results change.

INSERT INTO OneMinuteBarView
SELECT Symbol, AVG(Price), MAX(Price), MIN(Price), SUM(Price*Volume)/SUM(Volume)
FROM StreamFeed KEEP 1 MINUTE;

-- Determine the total bandwidth consumed by each IP address for the
-- last 1000 downloads. Output results every 15 second.

INSERT INTO SteramConsumedBandwidht
SELECT IP, SUM(FileSize)
FROM StreamDownloadLog KEEP 1000 ROWS
GROUP BY IP
OUTPUT EVERY 15 SECONDS;
```

Real-time And Historical Data

- Context to real-time data often exists in historical sources like RDBMS
  - Example: Mapping of RFID reader ID → Product SKU ID
- CEP enables correlating data stream with RDBMS

Table: `RFID_SKU_MAP`

Column	DataType	Description
RFID	Integer	RFID reader tag
SKU_ID	Integer	Product SKU ID for RFID

```
INSERT INTO RFIDWithSKU
SELECT current.rfid, current.location, product.sku_id
FROM RFIDReader AS current
(DATABASE 'ProductCatalog' SCHEMA (SKU_ID integer)
[[SELECT sku_id FROM rfid_sku_map
WHERE rfid = ?current.rfid]]) AS product
```