Webinar begins at 2:05PM, EST

***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.

***Hans van't Hag** [hans.vanthag@prismtech.com]*

OpenSplice DDS Product Manager, PrismTech

Hans has extensive experience in applying an information approach towards mission-critical and real-time net-centric systems. He is a co-author of the OMG DDS specification and has presented numerous papers on DDS and publish subscribe middleware technologies. Prior to joining PrismTech he worked for 23 years at Thales Naval Netherlands (TNN) where he was responsible as Product Manager for the development of the data-centric real-time middleware (SPLICE) as applied in TNN's TACTICOS combat system in service with 15 Navies worldwide.

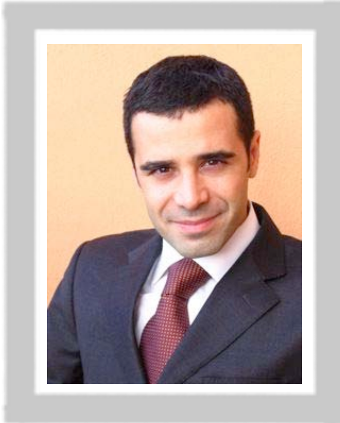
PrismTech

Productivity Tools & Middleware

OpenSplice DDS, Performance & Tuning

Agenda

- **OpenSplice DDS Overview**
- Architecting Distributed Systems
- What is “Performance”
- OpenSplice DDS Architecture
- OpenSplice DDS Deployment Tools
- The “Pother” benchmarking suite
- Demo
- Whats Next

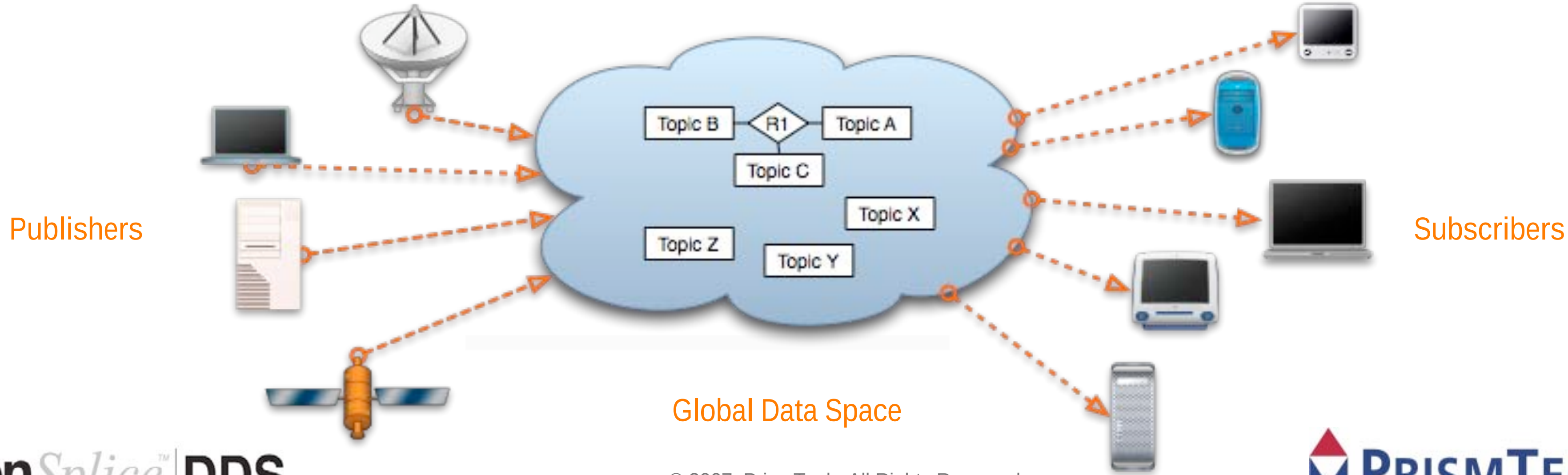


Dr. Angelo Corsaro

© 2007, PrismTech. All Rights Reserved

OpenSplice DDS

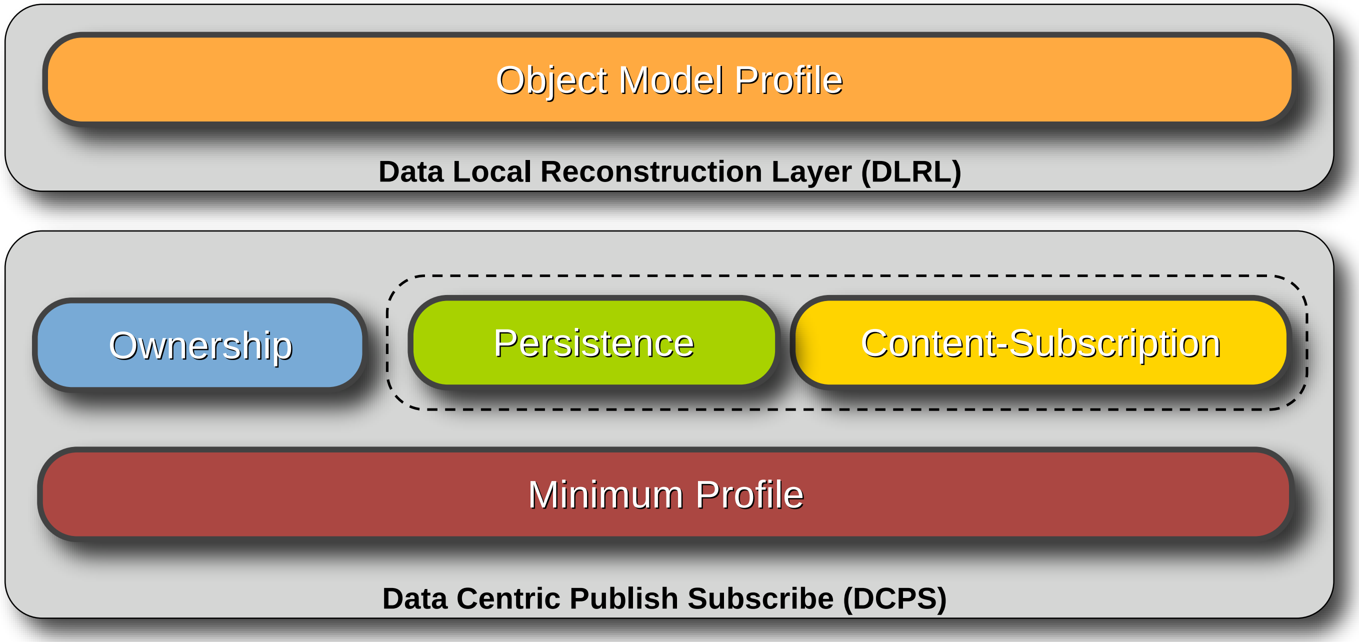
- **An High Performance Real-Time Data-Centric Publish/Subscribe Middleware**
  - *The right data, at the right place, at the right time -- all the time!*
  - *Fully distributed, high performance, highly scalable, and high availability architecture*
- **Perfect Blend of *Data-Centric* & *Real-Time Publish/Subscribe* Technologies**
  - *Content based subscriptions, queries and filters, DLRL*
  - *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*



© 2007, PrismTech. All Rights Reserved

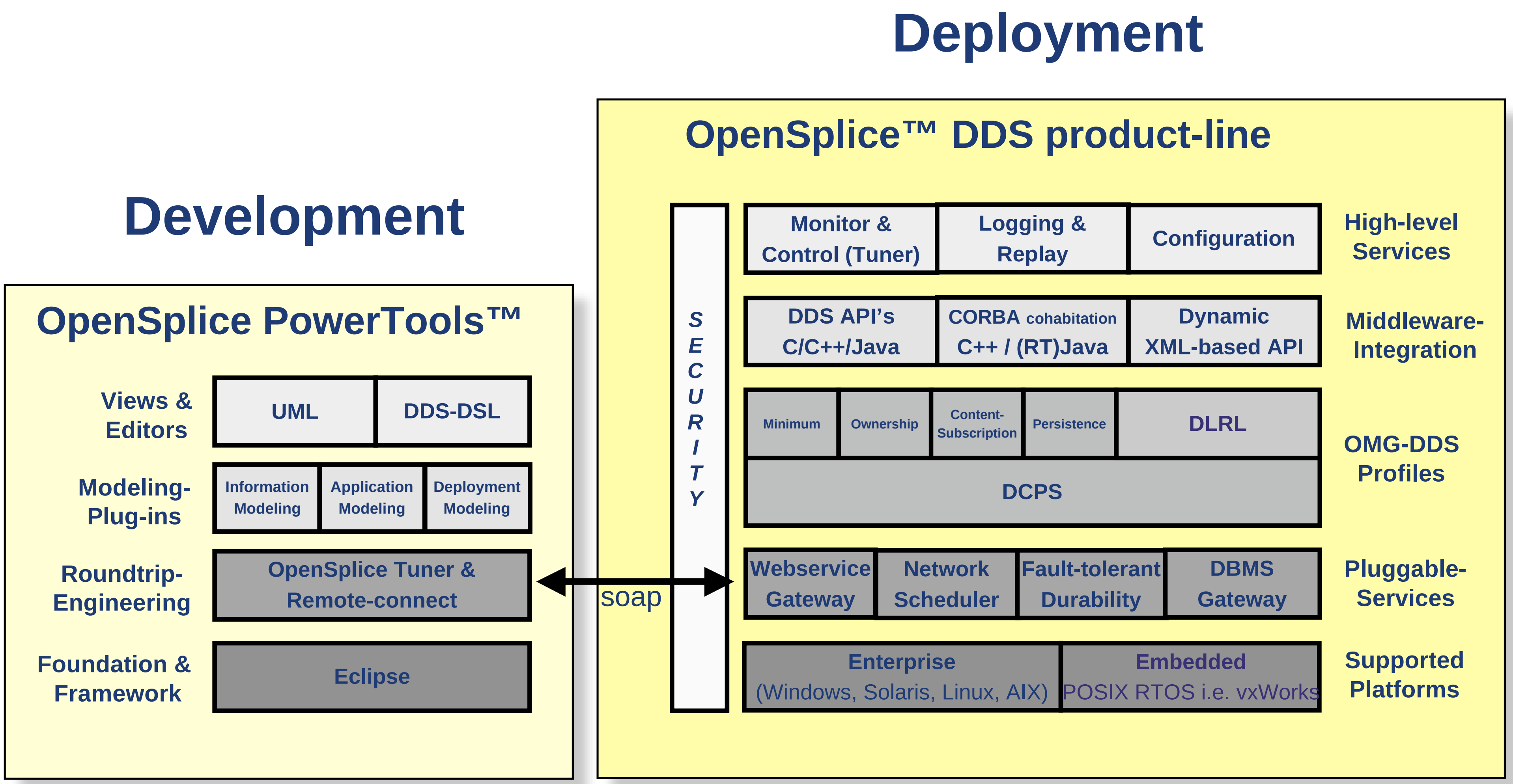
Standard Compliance

- **OpenSplice DDS is compliant with the full OMG DDS rev 1.2 Specification**



© 2007, PrismTech. All Rights Reserved

OpenSplice™ DDS Product-line



© 2007, PrismTech. All Rights Reserved

OpenSplice™ DDS v3 - In Summary

OpenSplice™ DDS

Functionality

- Full OMG-DDS specification coverage *(DCPS and DLRL)*
- Provision of a true 'fault-tolerant & secure information backbone' *(content-aware and FT-durability)*
- Wide Cohabitation and Connectivity with other Technologies *(Corba, RT-Java, DBMS, SOAP, XML)*
- Availability of (remote) deployment tools *(Tuner™ offering total & remote control)*
- Support for Information/application/deployment modeling *(DCPS/DLRL-specific roundtrip development)*

Performance

- **Scalability** w.r.t. number of applications as well as computing nodes and topics
- **Real-time determinism** by urgency (latency-budget) & importance (priority) based network-scheduling
- **Fault-tolerance** by FT-durability and reliable network-service shielding faulty applications from the network

Pedigree

- **Maturity.** Product proven, fielded,
  In service in 15 Navies world-wide
- **Fractal Architecture.** Large-scale, real-time, fault-tolerant, embedded, all in one system!
- **High Standard of Quality Assurance.** Process/procedures, QA-artefacts and regression testing w.r.t. number of applications as well as computing nodes and topics

© 2007, PrismTech. All Rights Reserved

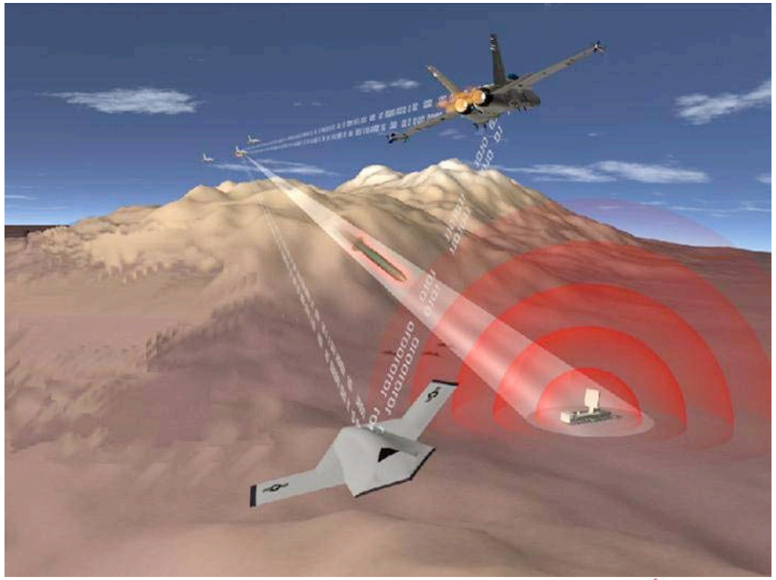
Agenda

- OpenSplice DDS Overview
- **Architecting Distributed Systems**
- Defining “Performance”
- OpenSplice DDS Architecture
- OpenSplice DDS Deployment Tools
- The “Pother” benchmarking suite
- Demo
- Whats Next

Hans van’t Hag

© 2007, PrismTech. All Rights Reserved

More Complex Systems and Requirements

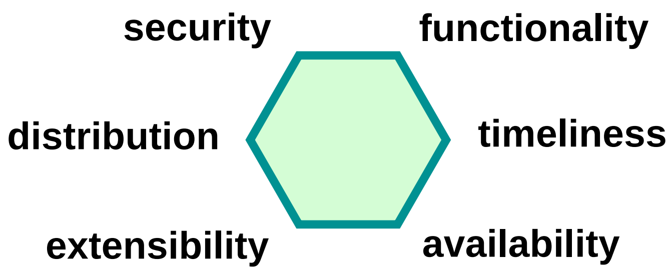


© 2007, PrismTech. All Rights Reserved

Distributed Systems: The Problem

Problem: (engineering) COST of distributed systems

- too complex
- not reactive
- not future-proof
- not fault tolerant



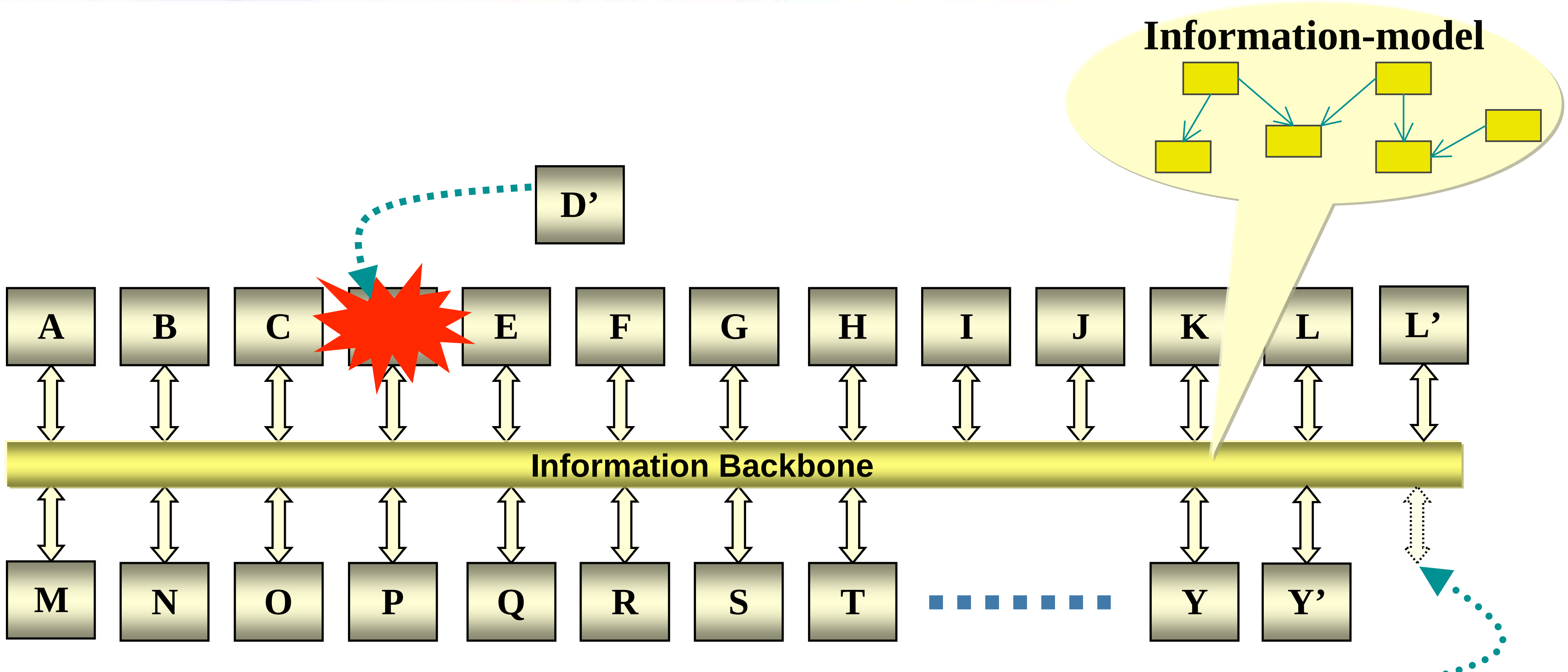
Because 'multi-dimensional engineering' is needed:

What about the current 'state-of-the-art'?

- architectures: C/S, MOM, SOA
- most efforts fall short in a number of dimensions:
- typically:
    - limited RT performance *(high-volume & low-latency balance)*
    - exploding complexity *(dependencies in many dimensions)*
    - costly evolution *(impact of changes & extensions)*

© 2007, PrismTech. All Rights Reserved

DDS: AN INFORMATION-CENTRIC ARCHITECTURE



Autonomous components

Interacting only with the information (model)

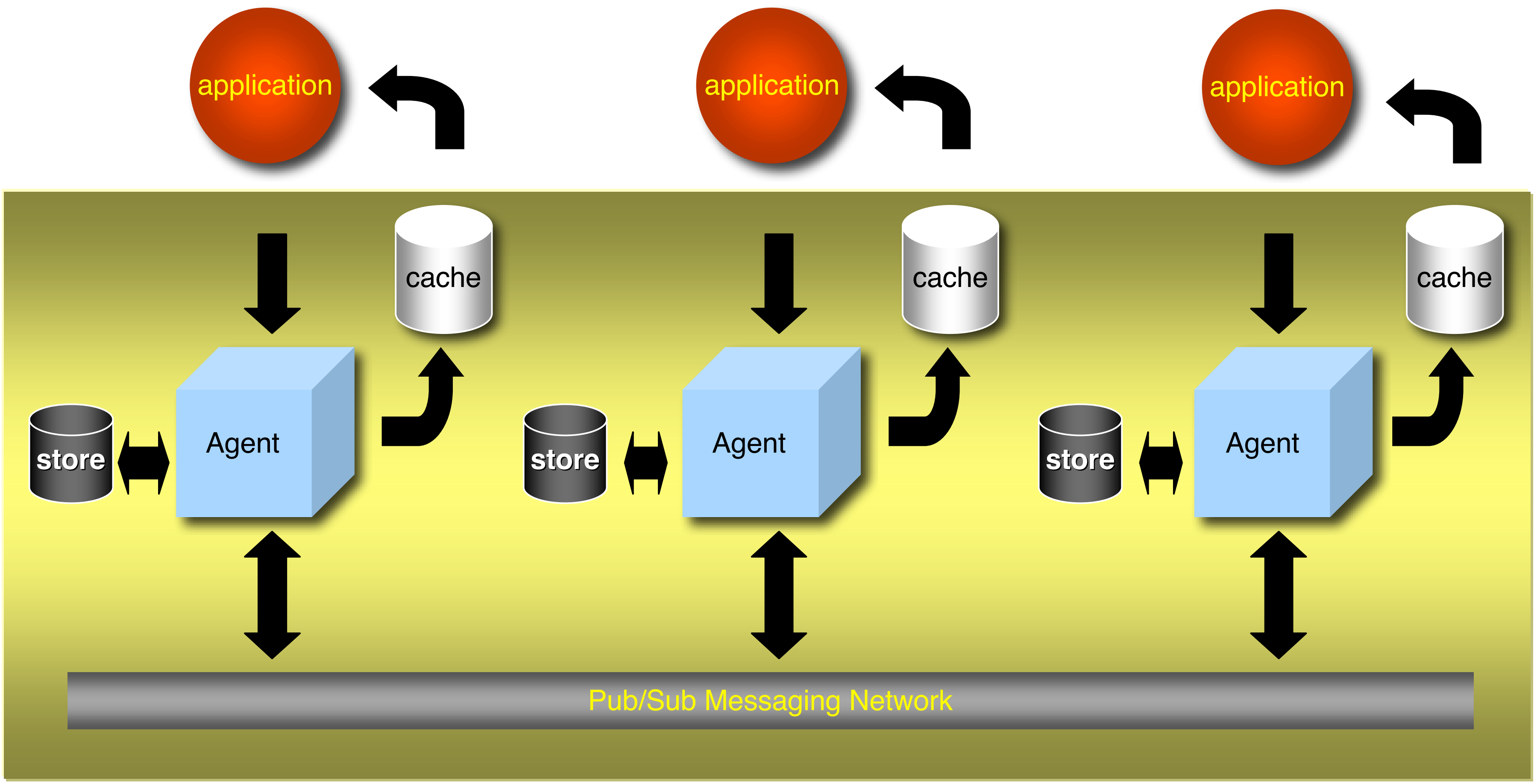
Spontaneous: Z, **Self-healing:** D'

Redundant & Replicated: L', Y'

QOS-driven Data Distribution Service **(urgency, importance, durability):** DDS

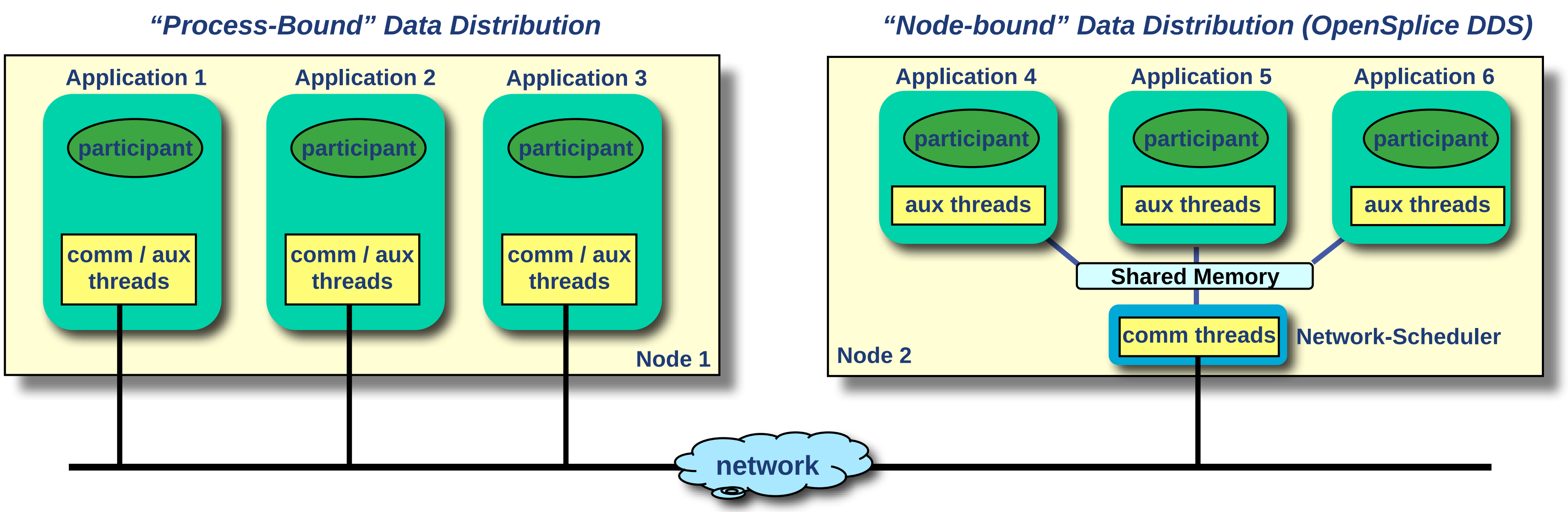
© 2007, PrismTech. All Rights Reserved

DDS: 'Under the Hood'



© 2007, PrismTech. All Rights Reserved

DDS: Implementation Architectures



- **Process bound DDS (RTI/OCI)**
  - Application level networking
- **Node bound DDS (OpenSplice)**
  - Node level networking

OpenSplice DDS

© 2007, PrismTech. All Rights Reserved

PRISMTECH

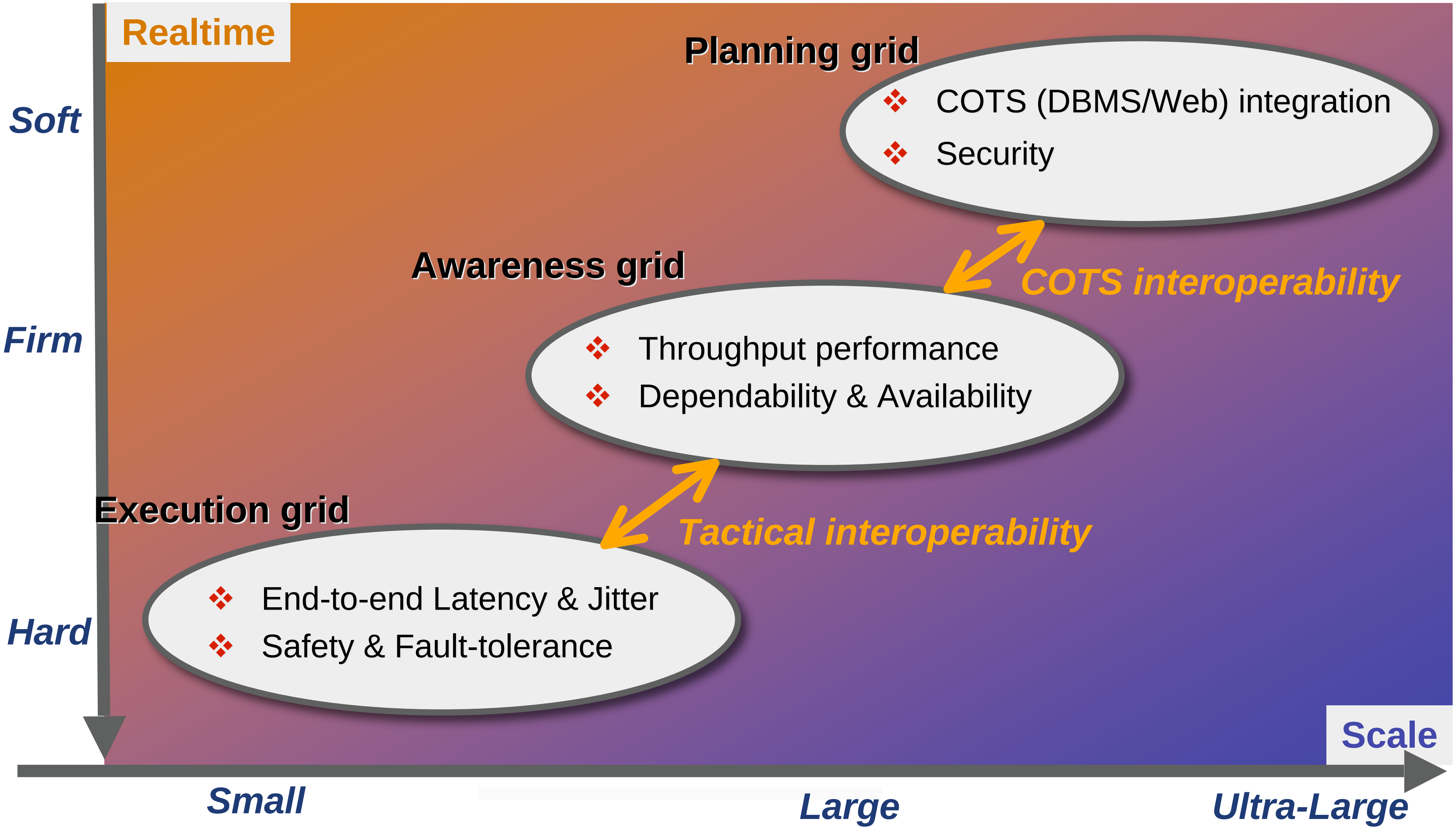
Agenda

- OpenSplice DDS Overview
- Architecting Distributed Systems
- Defining “Performance”
- OpenSplice DDS Architecture
- OpenSplice DDS Deployment Tools
- The “Pother” benchmarking suite
- Demo
- Whats Next

Hans van’t Hag

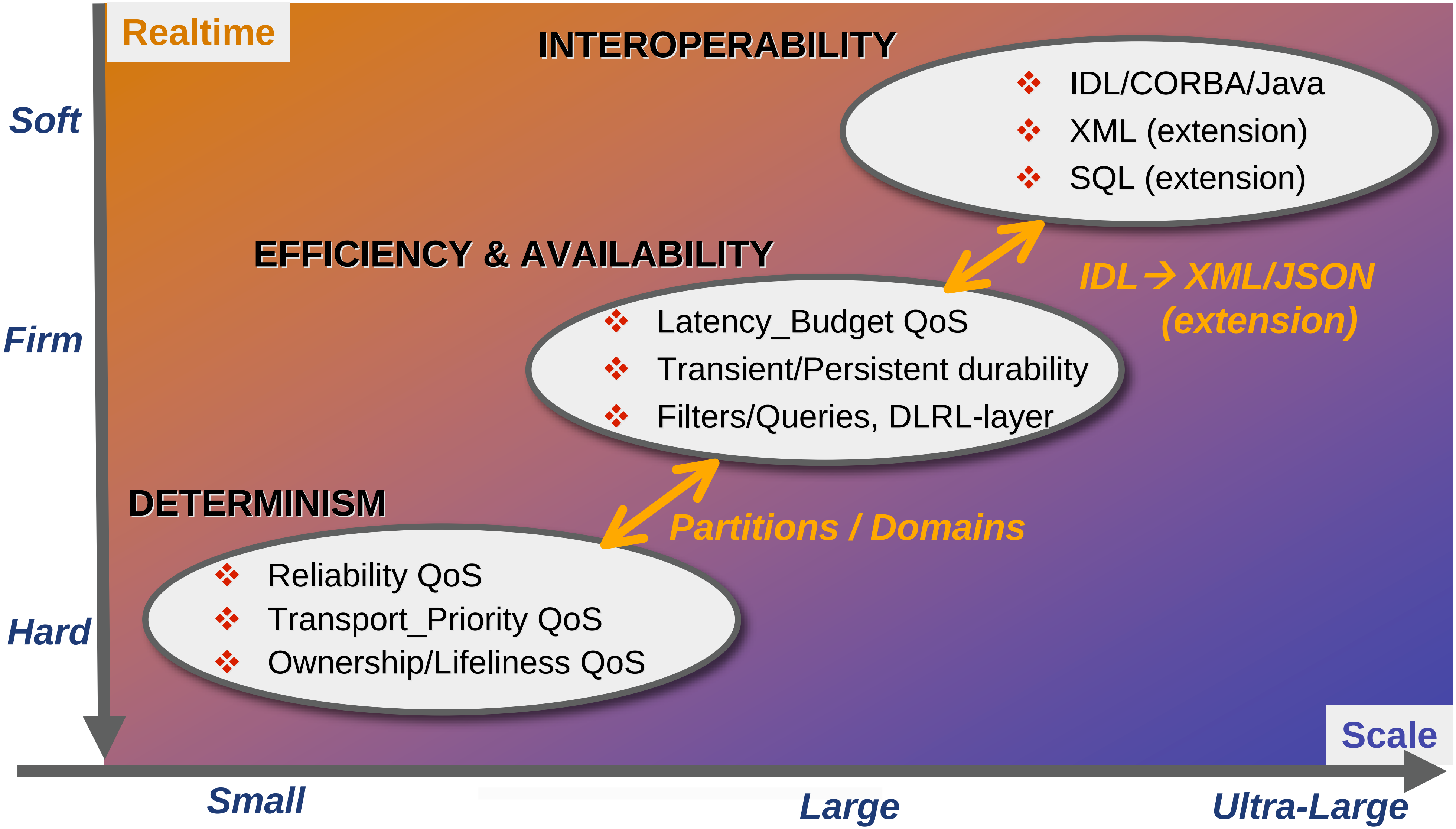
© 2007, PrismTech. All Rights Reserved

PERFORMANCE REQUIREMENTS IN NCW



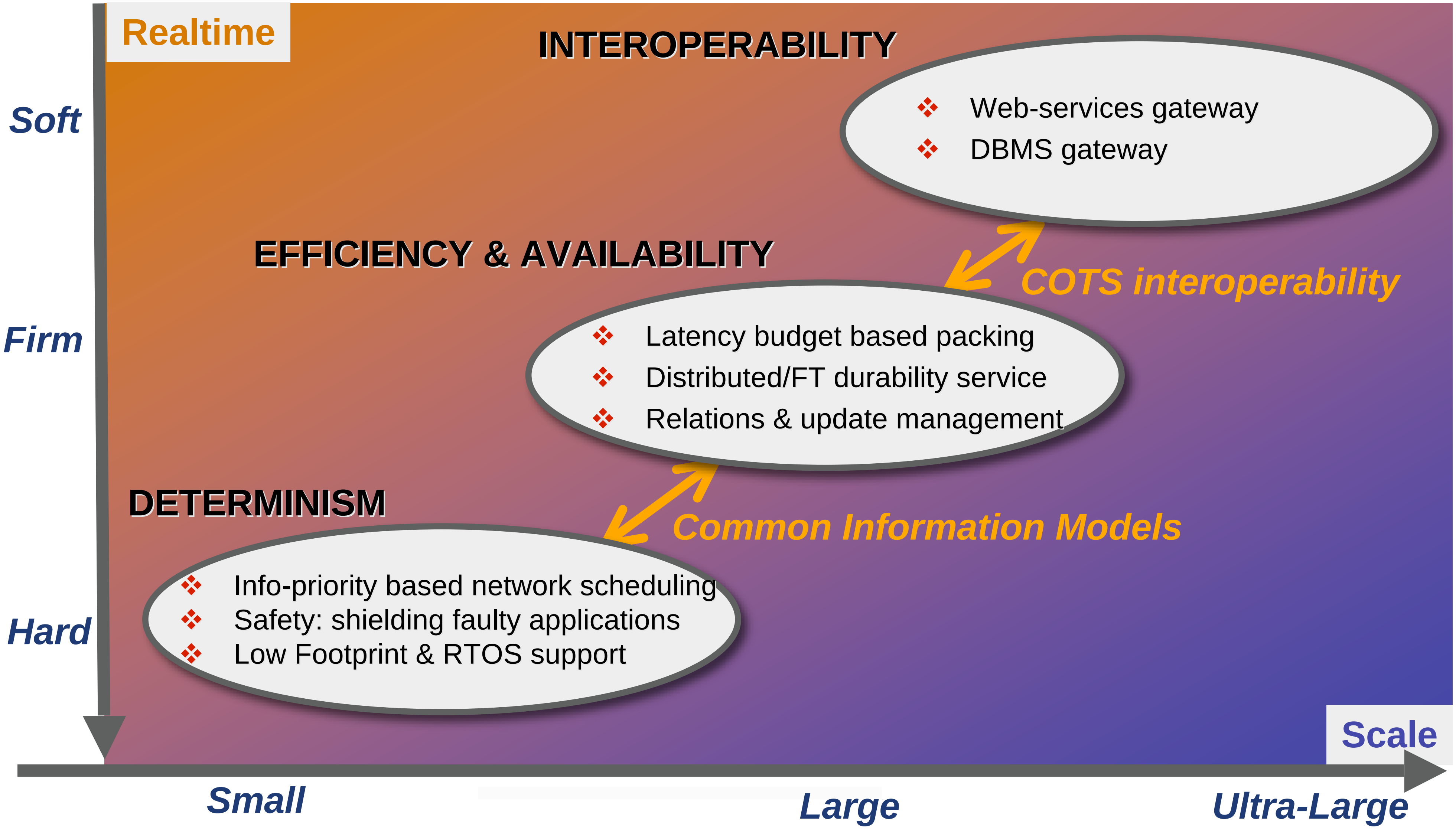
© 2007, PrismTech. All Rights Reserved

DDS APPLICABILITY: Functional Specification & QoS



© 2007, PrismTech. All Rights Reserved

DDS APPICABILITY: OpenSplice DDS Implementation



© 2007, PrismTech. All Rights Reserved

(1) Performance: Determinism & Latency

User Problem

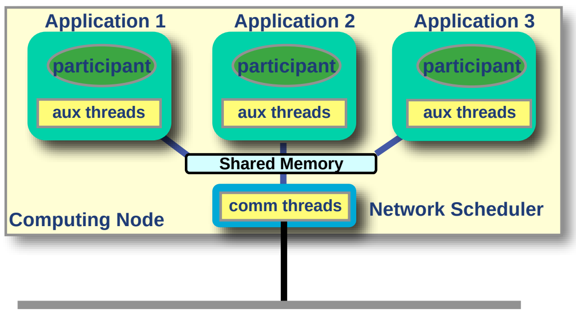
Data distribution in a system typically requires the ability to handle **different levels of importance**. In mission critical systems it is essential the *"The right (important) data always gets to the right place*" also in face of temporary overload condition

DDS Features (OMG-DDS specification/API)

The DDS provide the concept of **Transport Priority** as a mean of expressing data **importance**. This QoS can be used by DDS implementations to ensure that the distribution of more important data always **take precedence** over less important data

OpenSplice DDS Features (OMG-DDS/API implementation)

The combination of an **advanced Network Scheduler** and **Priority Bands** allow OpenSplice to enforce **information priority** across all nodes making up the distributed system



(2) Performance: Efficiency & Throughput

User Problem

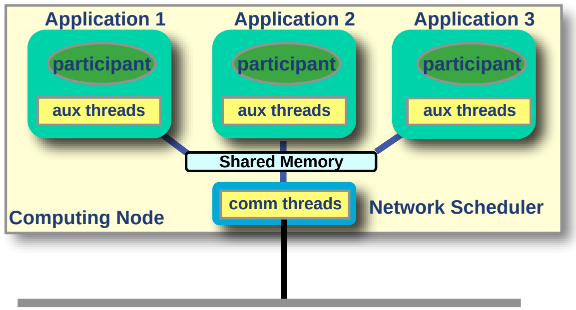
Complex distributed applications often require extraordinary **high throughput**. Achieving it requires **smart management** of the networking resources without introducing accidental complexity in the solution space

DDS Features (OMG-DDS specification/API)

The DDS provides the concept of **Latency Budget** as a means of specifying the **urgency** of data. This QoS can be used by DDS implementations to optimize the utilization of networking resources thus increasing efficiency and related data-distribution throughput for applications

OpenSplice DDS Features (OMG-DDS/API implementation)

OpenSplice's **Network Scheduler** takes advantage of Latency Budget in order to perform **data aggregation across topics & applications**. Moreover, its **federated architecture** reduces communication overhead, and greatly improves the overall application throughput



(3) Performance: Scalability & Footprint

User Problem

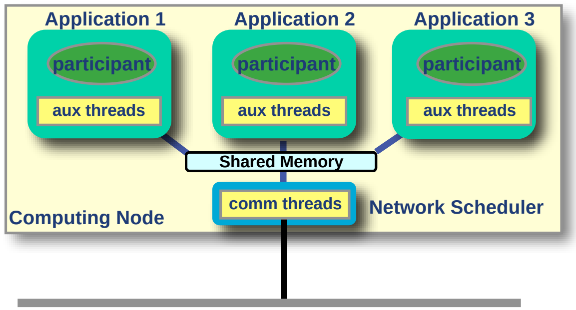
With Multi-Cores and Blades becoming more accessible many systems are contracting to a single box. Thus it becomes more and more relevant to have **efficient intra-nodal communication**. Moreover, as already experienced in OS such as Linux (dbus), Pub/Sub middleware provides the right level of decoupling, and facilitate **plug an play** behavior.

DDS Features (OMG-DDS specification/API)

The DDS is by nature **location agnostic**, thus is a perfect candidate for making applications location independent. Thanks to its potentially very high performance, it is also extremely suitable as a high throughput and low latency **intra-nodal communication** means

OpenSplice DDS Features (OMG-DDS/API implementation)

OpenSplice's architecture is **optimized for distributed as well as co-located applications**. Thanks to its Shared Memory optimizations, it delivers maximum performance at minimal footprint in local and distributed scenarios, thus allowing seamless localization or distribution of the application components



(4) Performance: Point to Point Communication

User Problem

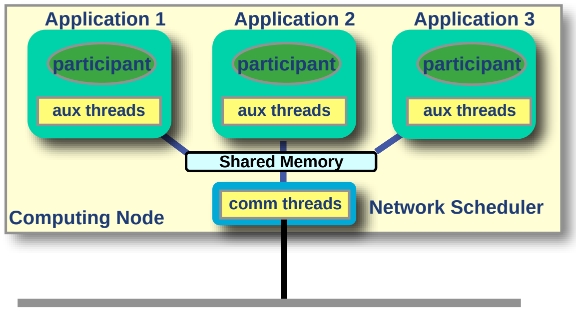
Some distributed application, along with one to many and many to many communication, require **high throughput point to point** communication

DDS Features (OMG-DDS specification/API)

Although DDS is agnostic of the underlying transport protocol, it does supports the notion of dynamic logical '**Partitions**'. This QoS policy can be used **to scope and bound the global dataspace** in the sense that communication between writers and readers is bounded to the shared notion of these Partitions as defined by the respective publishers and subscribers

OpenSplice DDS Features (OMG-DDS/API implementation)

OpenSplice's architecture allows the **dynamic mapping of logical partitions to physical network partitions (multicast groups)** to optimize the throughput of scoped (e.g. point-to-point) communication, as well as to minimize the impact of high throughput communication on other system elements



(5) Performance: Delivery & Availability

User Problem

Delivery **reliability** controls whether the data will **always** make its way to interested parties. **Availability**, controls when and **for how long** the data will be available. As reliability and availability have a cost, being able to quantify it gives useful guidance on how to design a system

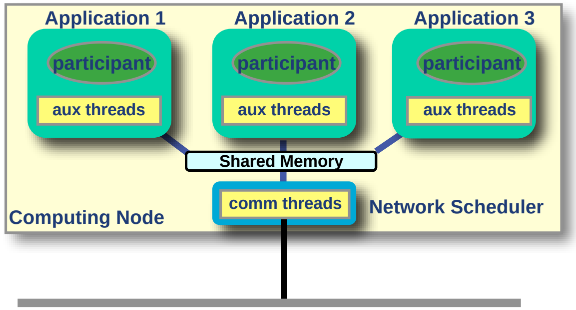
DDS Features (OMG-DDS specification/API)

The DDS provides a set of features, such as **Reliability**, and **Persistency** that allow to configure how data will be distributed and for how long it will be kept available for late-joining applications

OpenSplice DDS Features (OMG-DDS/API implementation)

- OpenSplice's communication architecture **minimizes** the protocol **overhead** of achieving reliable communication

- OpenSplice's distributed durability implementation provides **fault-tolerant availability** of non-volatile data



(6) Performance: Discovery Latency

User Problem

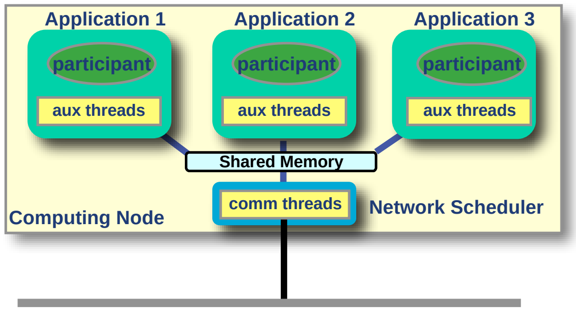
Large scale mission critical systems have stringent requirements on the time that can elapse between when the **system start-ups**, or **recovers** from a failure, and when the system becomes operational.

DDS Features (OMG-DDS specification/API)

The DDS features **dynamic discovery** which allows for plug and play interoperability of applications. However, if not implemented properly, its performance can adversely impact the startup and/or recovery time of large scale distributed systems

OpenSplice DDS Features (OMG-DDS/API implementation)

OpenSplice features a **constant time discovery** mechanism which allows application of **any scale** to be operational as soon as they are running. This provides application with unprecedented responsiveness to change of environment as well as change of operational mode



(7) Performance: Portability, Re-usability & Complexity

User Problem

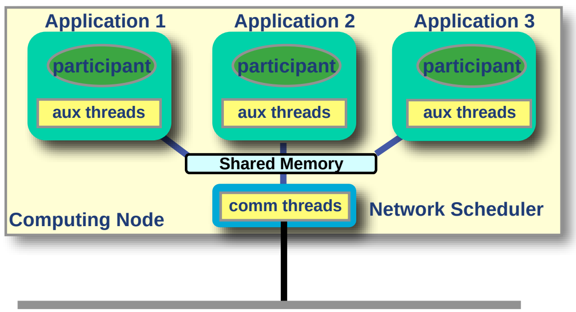
Applications should be **re-usable** (location agnostic), **portable** (DDS vendor agnostic) and **simple** (configuration agnostic)

DDS Features (OMG-DDS specification/API)

The DDS concept allows for a clear **separation of concerns** w.r.t. information-modeling (topics), application processing (business logic) and dynamic deployment (discovery) and as such provides a clear decoupling in space (location) and time (information persistence).

OpenSplice DDS Features (OMG-DDS/API implementation)

OpenSplice, as a full DDS implementation exploits these features 'to the fullest' w.r.t. **full API compliance** to the specification, **no vendor-specific extensions** and **application-agnostic configuration and tuning** possibilities of the DDS runtime-system



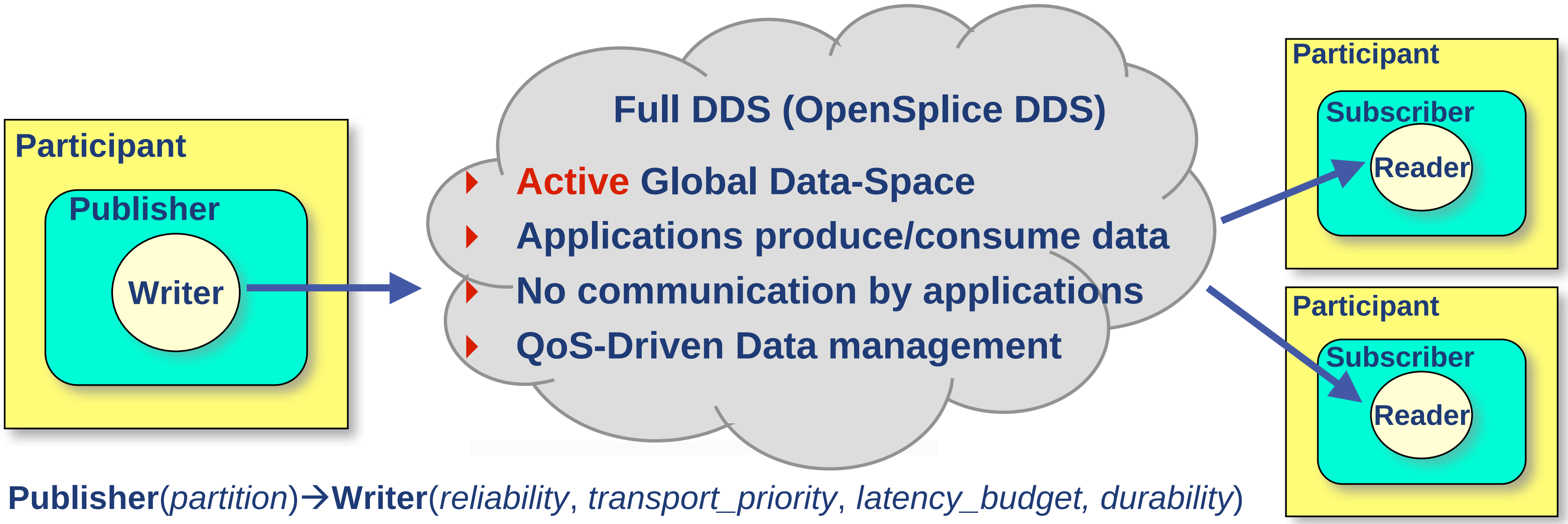
Agenda

- OpenSplice DDS Overview
- Architecting Distributed Systems
- Defining "Performance"
- **OpenSplice DDS Architecture**
- OpenSplice DDS Deployment Tools
- The "Pother" benchmarking suite
- Demo
- Whats Next

Hans van't Hag

© 2007, PrismTech. All Rights Reserved

DDS Architecture Impact: Minimal versus Full DDS

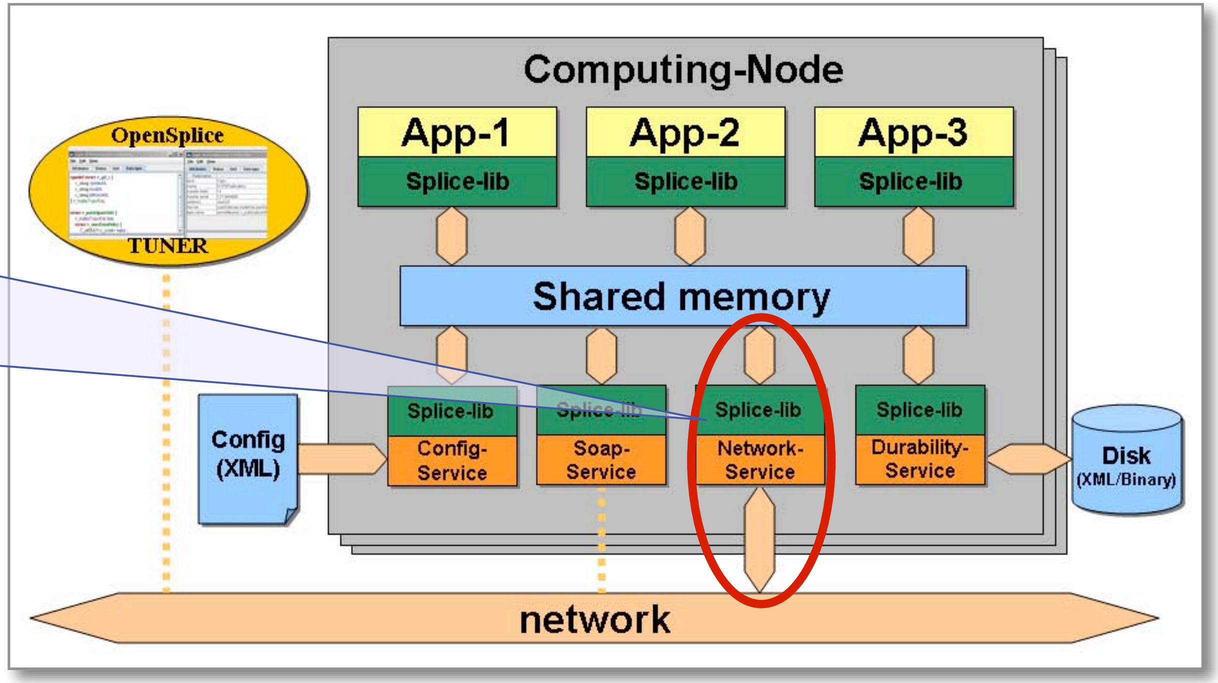


Publisher(*partition*)→**Writer**(*reliability*, *transport_priority*, *latency_budget*, *durability*)

© 2007, PrismTech. All Rights Reserved

DDS Architecture Impact: OpenSplice DDS architecture

- **Network-channels**
  - Priority bands
- **Network-partitions**
  - Multicast groups
- **Traffic-shaping**
  - Burst/throughput

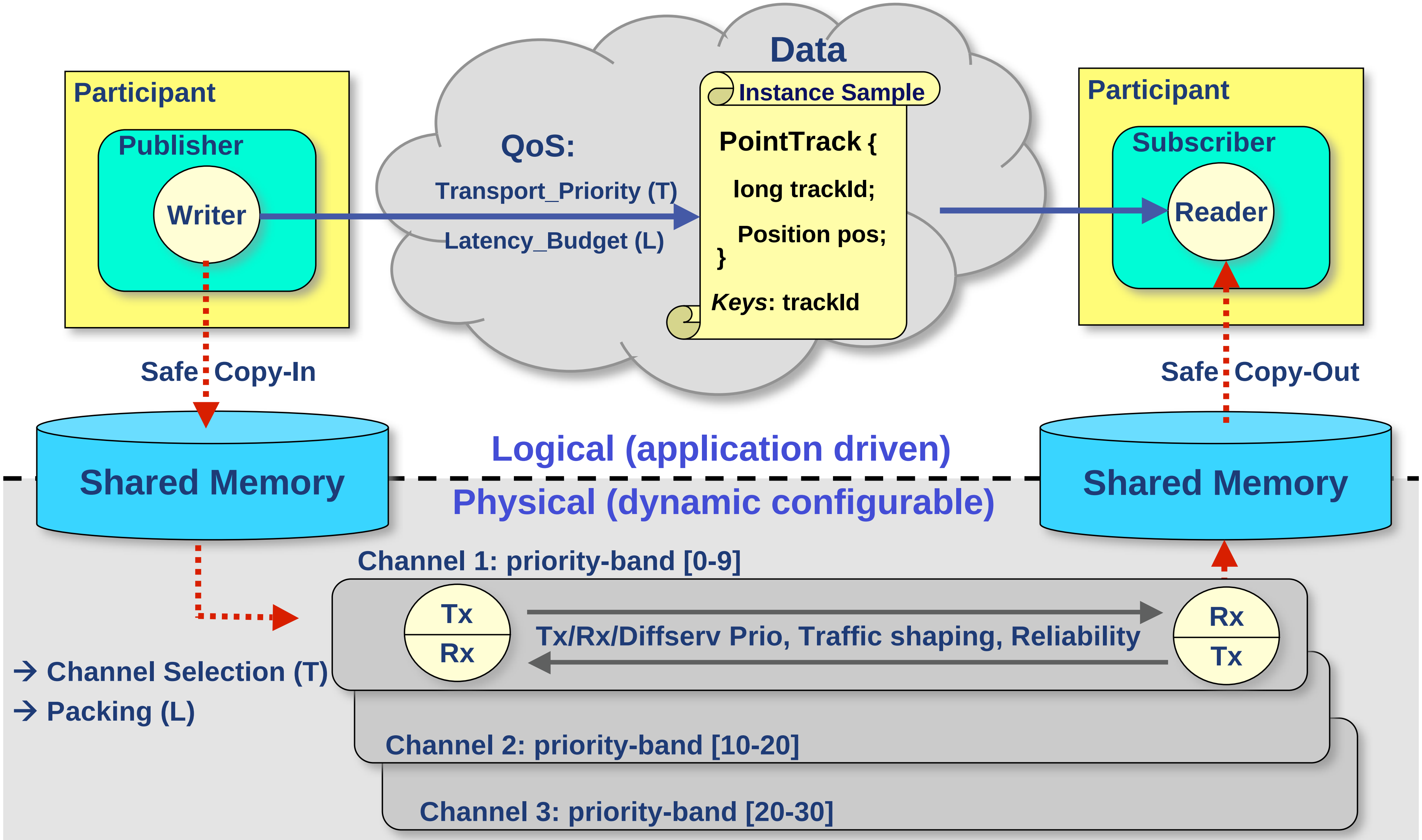


- **Scalability & Efficiency**
  - **Single shared library for applications & services** *(code-footprint)*
  - **Ring-fenced shared memory segment** *(single copy regardless of nr. of applications)*
  - **Data urgency driven network-packing** *(Latency_Budget QOS drives packing per channel)*
  - **Constant serialization/deserialization effort** *(one-time only regardless of nr. of applications)*

- **Determinism & Safety**
  - **Pre-emptive network-scheduler** *(traffic-shaping per priority-band)*
  - **Data importance based network-channel selection** *(Transport_Priority QoS of actual data)*
  - **Partition based multicast-group selection** *(dynamic mapping of logical DDS partitions)*
  - **Managed critical network-resource** *(limited impact/damage of faulty-applications)*

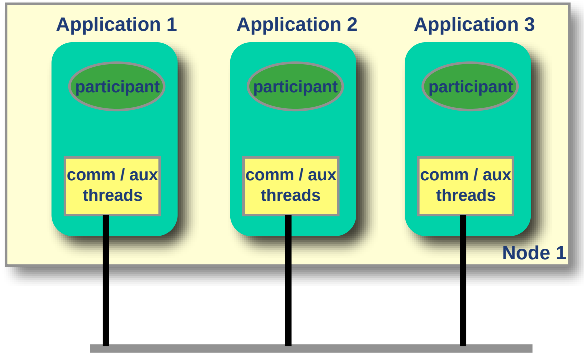
© 2007, PrismTech. All Rights Reserved

DDS Architecture Impact: OpenSplice DDS architecture (3/3)

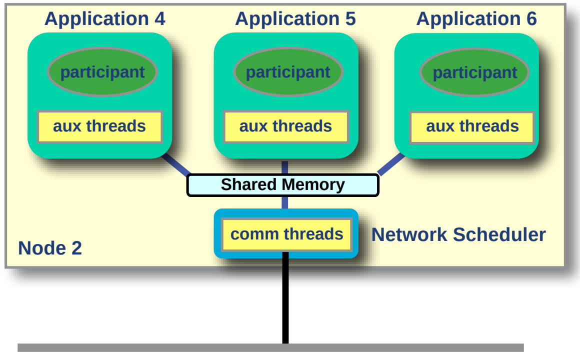


Priority Pre-emptive Urgency-based Network Scheduling !!

DDS Architecture Impact: Performance (1/2)



Process-bound Data Distribution



Node-bound Data Distribution (OpenSplice DDS)

- **Process bound DDS:** communication **BY** *applications* **BETWEEN** *individual readers/writers*
  - Application-process threads responsible for performing/maintaining system-wide communication
  - Application-level configuration for each reader/writer required
  - Real-world example: *private Car for transportation*
    - user needs driving-capabilities and must maintain his car
    - Every house has a driveway to the highway *(only fast when no traffic and car OK)*
- **OpenSplice:** communication **BY** *network-scheduler* **BETWEEN** *computing nodes*
  - Application-processes NOT responsible for communication over the network
  - Nodal Network-scheduler takes care of urgency/importance driven data-distribution
  - Network-scheduler 'populates' the shared-memory with relevant data from other nodes
  - Real-world example: *public transportation*
    - user needs no driving-capabilities nor needs to maintain the train
    - commuter/express trains can deliver massive amounts of people '*at the right place, at the right time*'

Trust Networking to Trusted Middleware !!

© 2007, PrismTech. All Rights Reserved

DDS Architecture Impact: Performance (2/2)

- **Example scenario:**
  - 200 nodes
  - 10 applications per node
  - 40 readers per application
  - 20 writers per application

- **Example OpenSplice configuration:**
  - 4 priority-bands (low/med/high/expedited)
  - So 4 best-effort and 4 reliable ‘channels’
  - Transport_Priority to express *data-importance*
  - Latency_Budget to express *data-urgency*

- **Impact of scale on Discovery Times**
  - **Process Bound DDS** *(using statefull & ‘typed RTPS-channels’ between individual readers/writers)*
    - 200 * 10 * 60 = **120,000** networking-endpoints (RTPS readers/writers) that need to be discovered
    - For each reader/writer peer-state has to be built-up before communication can occur
    - Discovery times can ‘explode’ with expanding scale
  - **OpenSplice DDS** *(using self-describing data sent over RTPS-like untyped channels between nodes)*
    - 200 * 8 = **1,600** networking-endpoints (replicating the relevant parts of the ‘shared-dataspace’ to other nodes)
    - **Zero discovery** times because of Self-describing data (‘inline-QoS’ overhead 50 bytes per sample)
    - Reliable-channels have an optional ‘discovery’ (of remote nodes) to prevent network-traffic if no remote nodes

- **Impact of scale on Data Distribution Performance**
  - **Process bound DDS** *(each writer forwards copies of each sample to each reader)*
    - **120,000** readers/writers that maintain copies of published/subscribed data
    - No packing of multiple-topics to increase efficiency (typed RTPS channels between DomainParticipants)
  - **OpenSplice DDS** *(only 1 copy of any sample maintained within 1 node and shared between all applications)*
    - **Only 1 copy** of any topic sample required that populates ALL relevant writer and reader caches/histories per node
    - Latency-budget driven packing (of ALL topics within a priority-band) dramatically increases efficiency
    - De-serialization only once per node instead of once per participant/reader

System Scalability requires a scalable DDS architecture

DDS Architecture Impact: Dependability

- **Determinism scenario:**
  - Track producer:
    - Normal / High-Prio tracks
  - High-priority tracks must pre-empt low-priority tracks

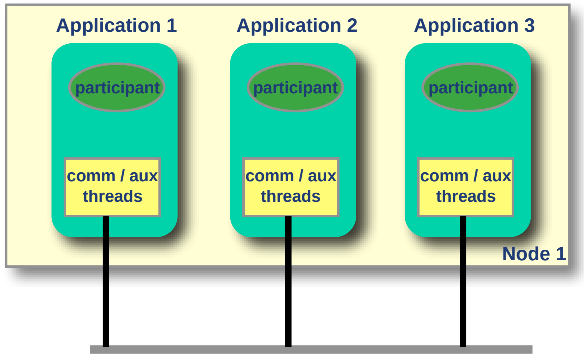
- **Safety scenario (misbehaving application):**
  - 10 applications at different priorities
  - High-priority process publishes at high rate
  - Low priority process can't execute

- **Impact of scale on Determinism**
  - **Process bound DDS** *(Application bound Data Distribution)*
    - Data-Priority (importance of data) = Processing Priority (importance of processing)
    - Track publisher has to (dynamically) create communication-threads at right priority, will travel over same channel
    - Track subscriber has no means to handle high/low priority samples
  - **OpenSplice DDS** *(Node bound Data Distribution)*
    - Writers can set TRANSPORT_PRIORITY QoS 'per sample' to indicate IMPORTANCE of the data
    - High-priority data will pre-empt low(er) priority data both in sending AND receiving node

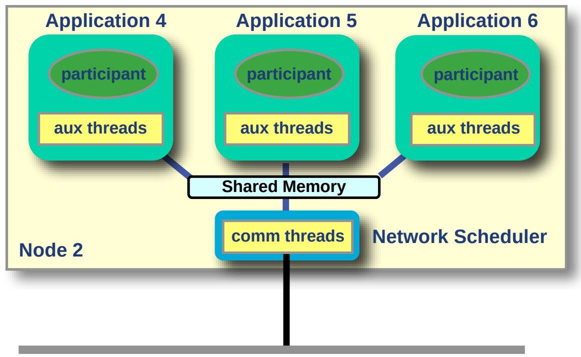
- **Impact of scale on System Safety**
  - **Process bound DDS** *(Application bound Data Distribution)*
    - Non-responsive low-priority process can trigger system-wide retransmissions
    - Over-responsive high-priority process can overload system-wide network & processing resources
    - So every application is a potential single-point-of-(system)failure !
  - **OpenSplice DDS** *(Node bound Data Distribution)*
    - Traffic-shaped (reactivity, max-throughput, burst-size) network-channels are managed by trusted middleware
    - Application misbehaviour (under/over responsiveness) can only have limited/bounded impact on other nodes

Mission critical systems need arbitration: for processes AND data !!

DDS PERFORMANCE: Portability, Re-usability & Complexity (1/4)



Process-bound Data Distribution



Node-bound Data Distribution (OpenSplice DDS)

- **General Issues**
  - Application **location** awareness (configuration) → Re-usability
  - Application **vendor** awareness (API/QoS compliance, required vendor-specific extensions) → Portability
- **Application Portability** *(between DDS implementations)*
  - OpenSplice is fully DDS-compliant (DDS rev1.2)
  - OpenSplice doesn't require vendor-specific API extensions unlike most process-bound DDS implementations
- **Application Re-usability** *(in different systems & environments)*
  - OpenSplice does NOT require application-level transport-configuration, unique ID's etc.
  - OpenSplice runtime configuration does NOT impact application-code
  - Application QoS policy settings can be tuned at runtime by OpenSplice Tuner™ (supporting MDE roundtrip-engineering)
- **System Complexity**
  - OpenSplice supports a clear **separation of concerns** w.r.t.
    - Information **modeling** : shared information model annotated with QoS policies for global behavior: reliability, urgency, importance, persistence
    - Application **development** : re-usable (location/deployment agnostic) applications with tool-supported code-generation (PowerTools™ MDE-suite)
    - System **integration** : dynamically configurable & Tunable deployment environment without impacting application-code

There's more to performance than pt-to-pt latency

© 2007, PrismTech. All Rights Reserved

PRISMTECH

DDS API compliance: Portability, Re-usability & Complexity (2/4)

	DDS1	DDS2	DDS3
DomainParticipant Factory	compliant	compliant	proprietary function
Register Data Types	static method	member method	member method
Spec Operations	extra argument (newer spec)	compliant	compliant
Key Declaration	`//@key`	single `#pragma`	pair of `#pragma`
Required App. IDs	publisher & subscriber	none	publisher
Required App. Transport Config	code-based	none	file-based or code-based

DDS2=OpenSplice

DDS API compliance: Portability, Re-usability & Complexity (3/4)

Description	Differences	Compliance Issue?
type of DomainId_t (native in spec IDL)	**OpenSplice** - char* **RTI DDS** - signed 32-bit int **OpenDDS** - signed 32-bit int	NO (the DDS spec example is signed 32-bit int but it's not required)
use of namespace DDS	**OpenSplice** - yes **RTI DDS** - yes, but must include extra header file **OpenDDS** - yes	NO (C++ mapping requires it, but it's obtainable from all vendors)
mapping of IDL modules to namespaces	**OpenSplice** - yes **RTI DDS** - not by default (needs command line option) **OpenDDS** - yes	NO (C++ mapping requires it, but it's obtainable from all vendors)
use of CORBA _ptr and _var types	**OpenSplice** - yes **RTI DDS** - no (without *RTI CORBA Compatibility Kit*) **OpenDDS** - yes	YES (It's an IDL to C++ mapping issue)
use of CORBA basic types	**OpenSplice** - yes **RTI DDS** - no (proprietary typedefs without *RTI CORBA Compatibility Kit*) **OpenDDS** - yes	YES (**RTI not compliant**, Not a CORBA issue but IDL to C++ mapping - see section 1.3)
scope of generation from implied IDL	**OpenSplice** - same as original IDL **RTI DDS** - same as original IDL **OpenDDS** - configurable (global scope by default)	UNKNOWN (I can't find any reference to it in the DDS spec)
type registration	**OpenSplice** - *foo*TypeSupport instantiated on stack **RTI DDS** - no instantiation (register_type() is static) **OpenDDS** - *foo*TypeSupport instantiated on heap	YES (Only OpenDDS is compliant here - see section 1.3 of IDL C++ mapping. OpenSplice also allows heap instantiation, but the mapping forbids direct instantiation of interface classes)
type of [*datatype*]Seq max length	**OpenSplice** - CORBA::ULong **RTI DDS** - signed long **OpenDDS** - CORBA::ULong	YES (**RTI not compliant**, C++ mapping prescribes IDL sequence length as CORBA::ULong)

DDS API compliance: Portability, Re-usability & Complexity (4/4)

Description	Differences	Compliance Issue?
resolution of DomainParticipantFactory	**OpenSplice** - static instance() method **RTI DDS** - static instance() method **OpenDDS** - proprietary global function	YES (**OpenDDS is non-compliant**)
passing of ConditionSeq to wait()	**OpenSplice** - by reference **RTI DDS** - by reference **OpenDDS** - does not support WaitSets or conditions	NO (The signature of this operation changed from passing ConditionSeq as an OUT parameter - which takes a pointer to be converted to the ConditionSeq_out class - in DDS 1.0, to passing ConditionSeq as in INOUT parameter - which maps to a reference for sequences - in DDS 1.1.)
passing of [*datatype*]Seq and SampleInfoSeq to take()	**OpenSplice** - by reference **RTI DDS** - by reference **OpenDDS** - by pointer	YES (**OpenDDS is not compliant** with IDL C++ mapping)
identifier for generated downcasting method	**OpenSplice** - _narrow **RTI DDS** - narrow **OpenDDS** - _narrow	YES (**RTI DDS is non-compliant** with IDL C++ mapping)
StatusMask arg in create_* methods	**OpenSplice** - yes **RTI DDS** - yes **OpenDDS** - no	YES (RTI DDS and OpenSplice compliant with DDS 1.1 & 1.2, **OpenDDS compliant only with DDS 1.0**)
proprietary listener methods	**OpenSplice** - no **RTI DDS** - no **OpenDDS** - yes (in DataReaderListener and DataWriterListener)	YES (the extra methods are pure virtual, and must be recognized and implemented)
type of *datatype*Seq[index] index variable between brackets	**OpenSplice** - CORBA::ULong **RTI DDS** - signed long **OpenDDS** - CORBA::ULong	YES (C++ mapping prescribes IDL sequence length as CORBA::ULong. Even though **RTI DDS doesn't use CORBA basic types** without the optional kit, it would still be better if it used an unsigned type)
proprietary QoS settings required	**OpenSplice** - no **RTI DDS** - yes **OpenDDS** - no	YES **RTI DDS has several non-spec members in the ParticipantQod struct, some of which (host id and app id) need to be set on publisher and subscriber, with another (participant index) needing to be set on the subscriber only**

Agenda

- OpenSplice DDS Overview
- Architecting Distributed Systems
- Defining “Performance”
- OpenSplice DDS Architecture
- **OpenSplice DDS Deployment Tools**
- The “Pother” benchmarking suite
- Demo
- Whats Next

Hans van’t Hag

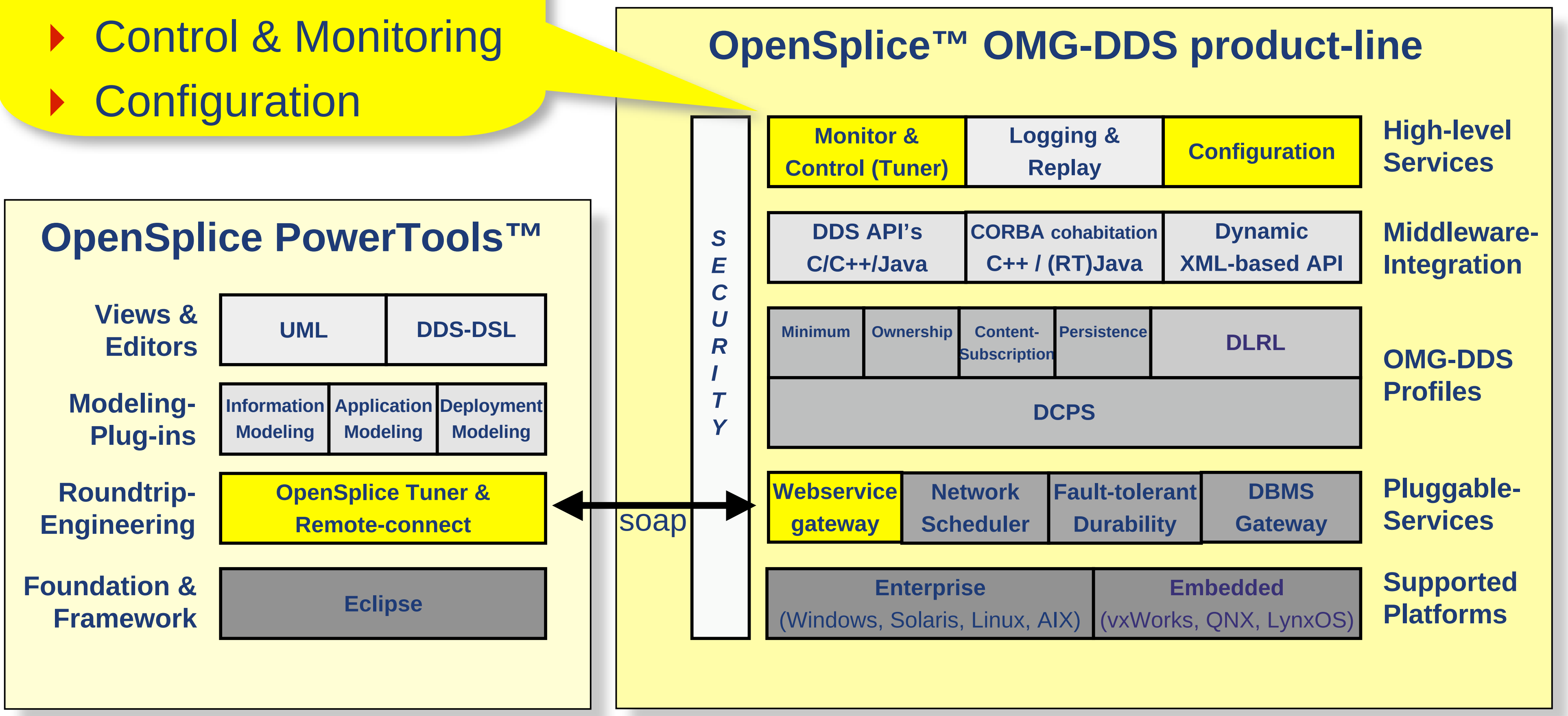
© 2007, PrismTech. All Rights Reserved

OpenSplice Tuner™: *local & remote monitor & Control*

Monitor & control

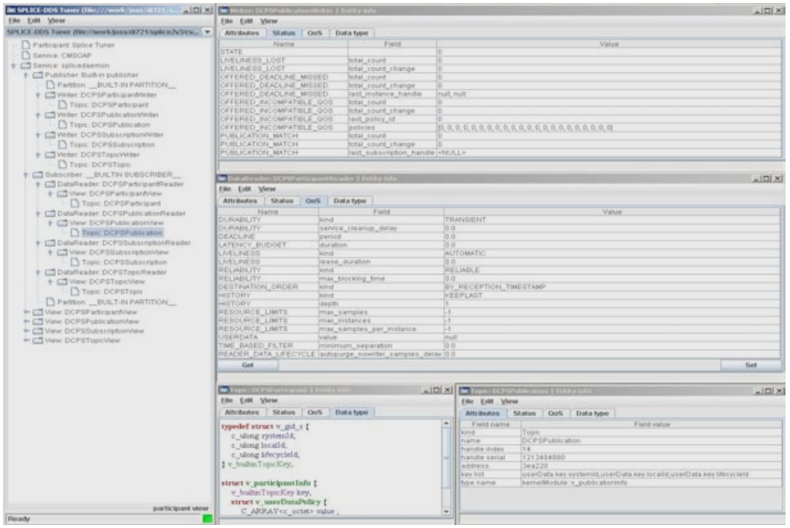
- Local & Remote
- Control & Monitoring
- Configuration

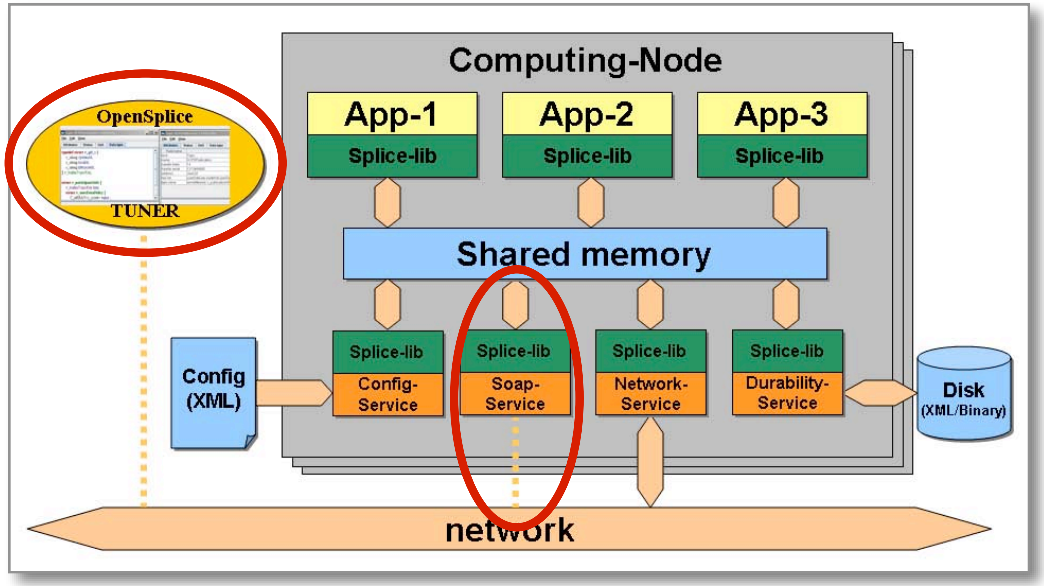
Deployment



© 2007, PrismTech. All Rights Reserved

OpenSplice DDS Advantages: *OpenSplice Tuner*





- **Features**
  - **Design** stage: *deploy* the information model even without applications by dynamicallly created readers/writers
  - **Development** stage: *inject* (write) test-data, capture (read/store) application responses
  - **Deployment** stage: *inspect* reader/writer caches, QoS and performance metrics
  - **Maintenance** stage: *log/inject* datasets (both volatile and/or persistent)
- **Characteristics**
  - 100% Java application, direct or remote connection to any OpenSplice™ system using SOAP™
  - Dynamic discovery of all DDS entities (participants, subscribers, publishers, readers, writers, services)
  - Finetune QoS parameters (at runtime)
  - Support Roundtrip-engineering (SpliceTuner as OpenSplice PowerTools™ MDE eclipse-plugin)

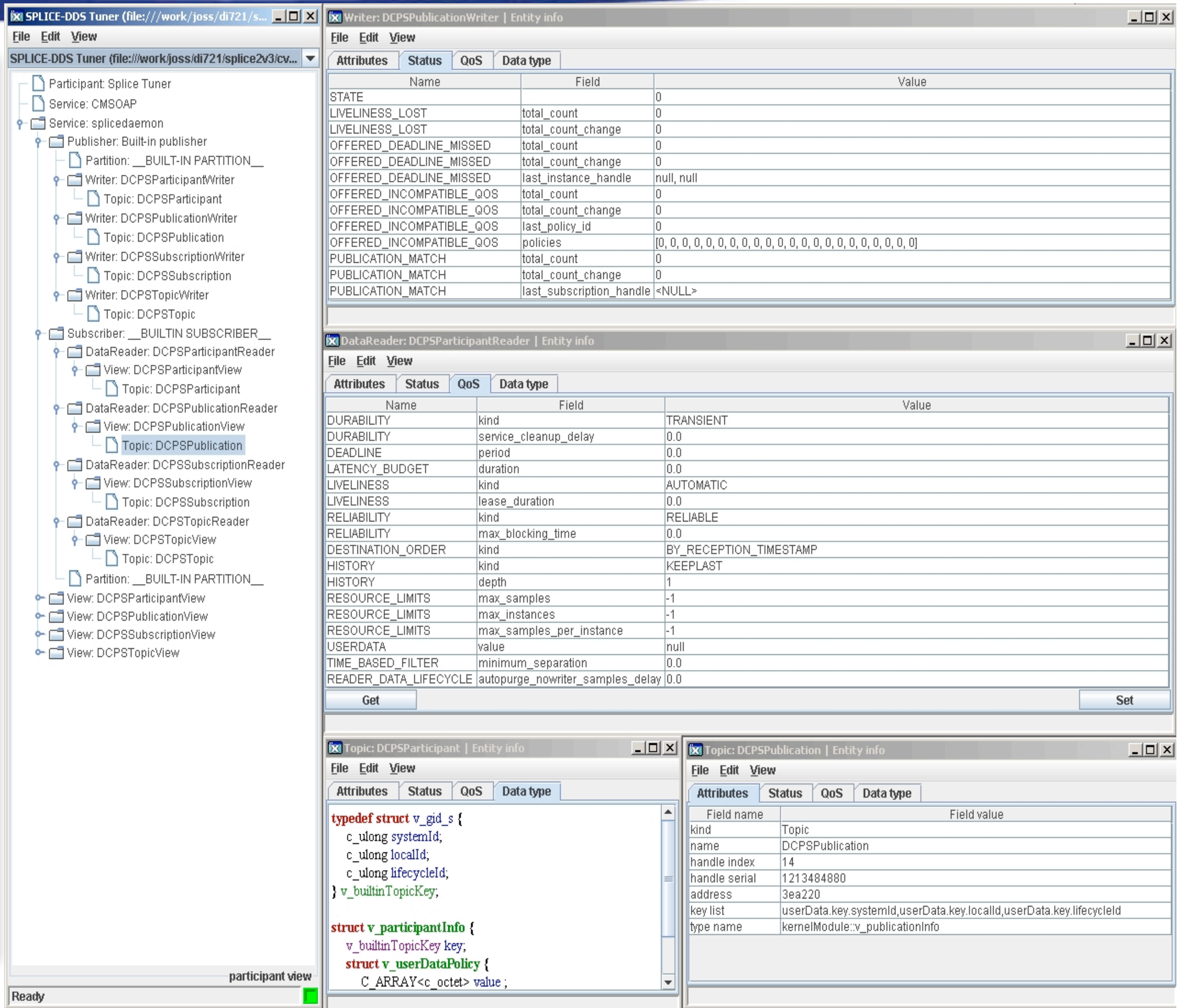
© 2007, PrismTech. All Rights Reserved

OpenSplice TUNER™

Splice-Tuner:

TOTAL SYSTEM CONTROL:

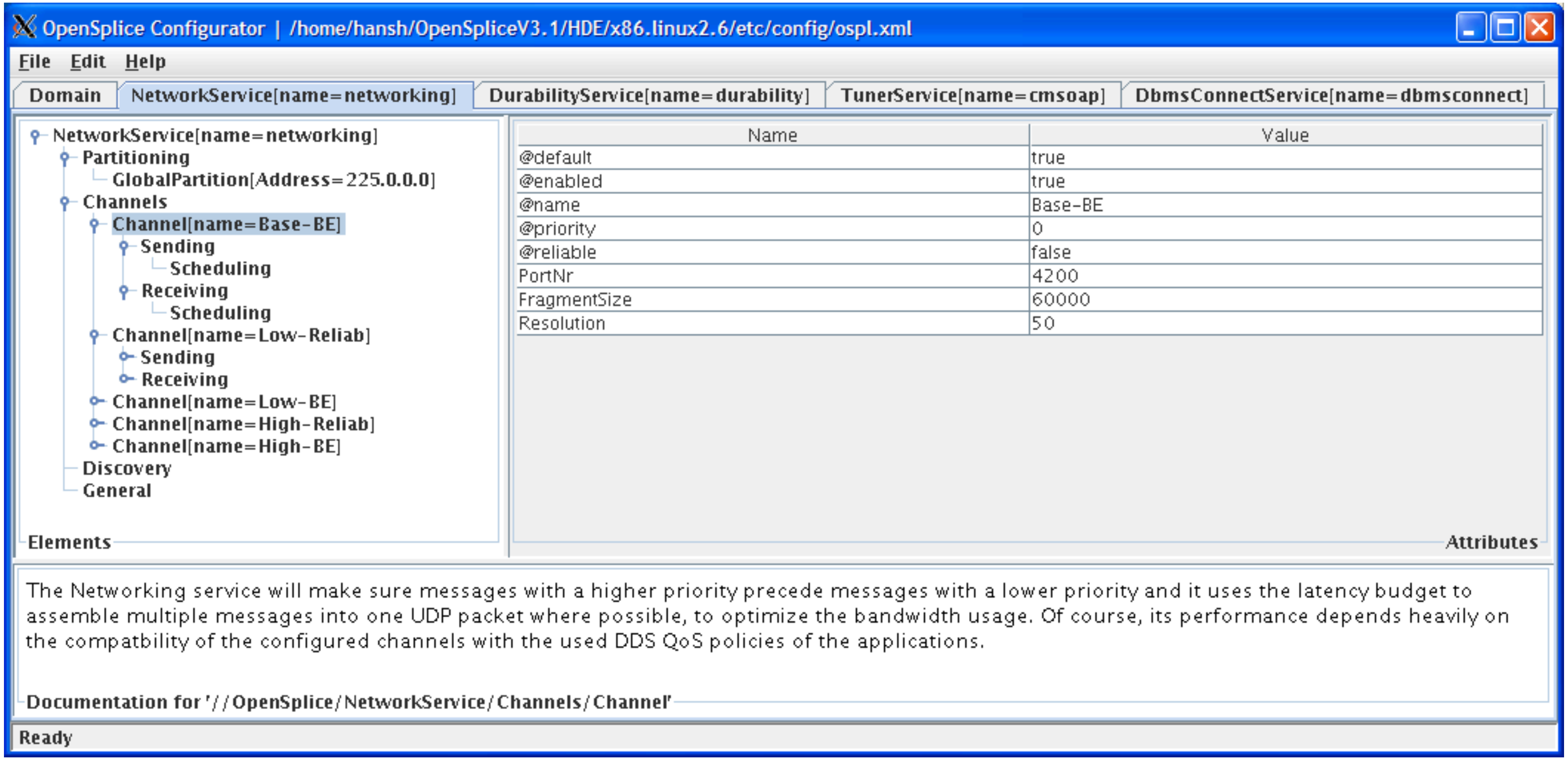
- 100 % **Java-based**
- **Remote connect** via SOAP
- **Monitor & Control**:
  - all DDS-entities & relations
  - all QoS settings
  - all services such as:
    - communication
    - durability-service
- **Interactive browsing**:
  - inspect any data-cache
  - make cache-snapshots
  - view statistics
- **Reading/Writing data**:
  - create readers/writers
  - read/write any data
- **Multiple views**:
  - participant view
  - topic view
  - partition view
- **Dynamic creation** of:
  - readers (with filters/queries)
  - writers (with input validation)
- **Automatic discovery** of:
  - Partitions & participants
  - Topics with name/type
  - related publishers/writers
  - related subscribers/readers



© 2007, PrismTech. All Rights Reserved

PRISMTECH

OpenSplice Configurator™



- User-friendly, intuitive generation and management of OpenSplice (XML) configuration files
- Context-sensitive help (deployment-information)
- Enforced correctness
- Deployment help
- 100% Java tool so platform independent (also doesn't depend on OpenSplice to be installed)

© 2007, PrismTech. All Rights Reserved

PRISMTECH

Agenda

- OpenSplice DDS Overview
- Architecting Distributed Systems
- Defining “Performance”
- OpenSplice DDS Architecture
- OpenSplice DDS Deployment Tools
- The “Pother” benchmarking suite
- Demo
- Whats Next

Hans van’t Hag

© 2007, PrismTech. All Rights Reserved

Benchmarking for 'real-life' systems

Goal

- Evaluate the performance of an application benchmark **relevant to the application domain**
- Evaluate & highlight how **predictability and throughput** are maintained **with expanding scale**
- Showcase **DDS implementation** features:
  - QoS support
  - Distribution and Discovery
  - Determinism, efficiency and scalability

Approach

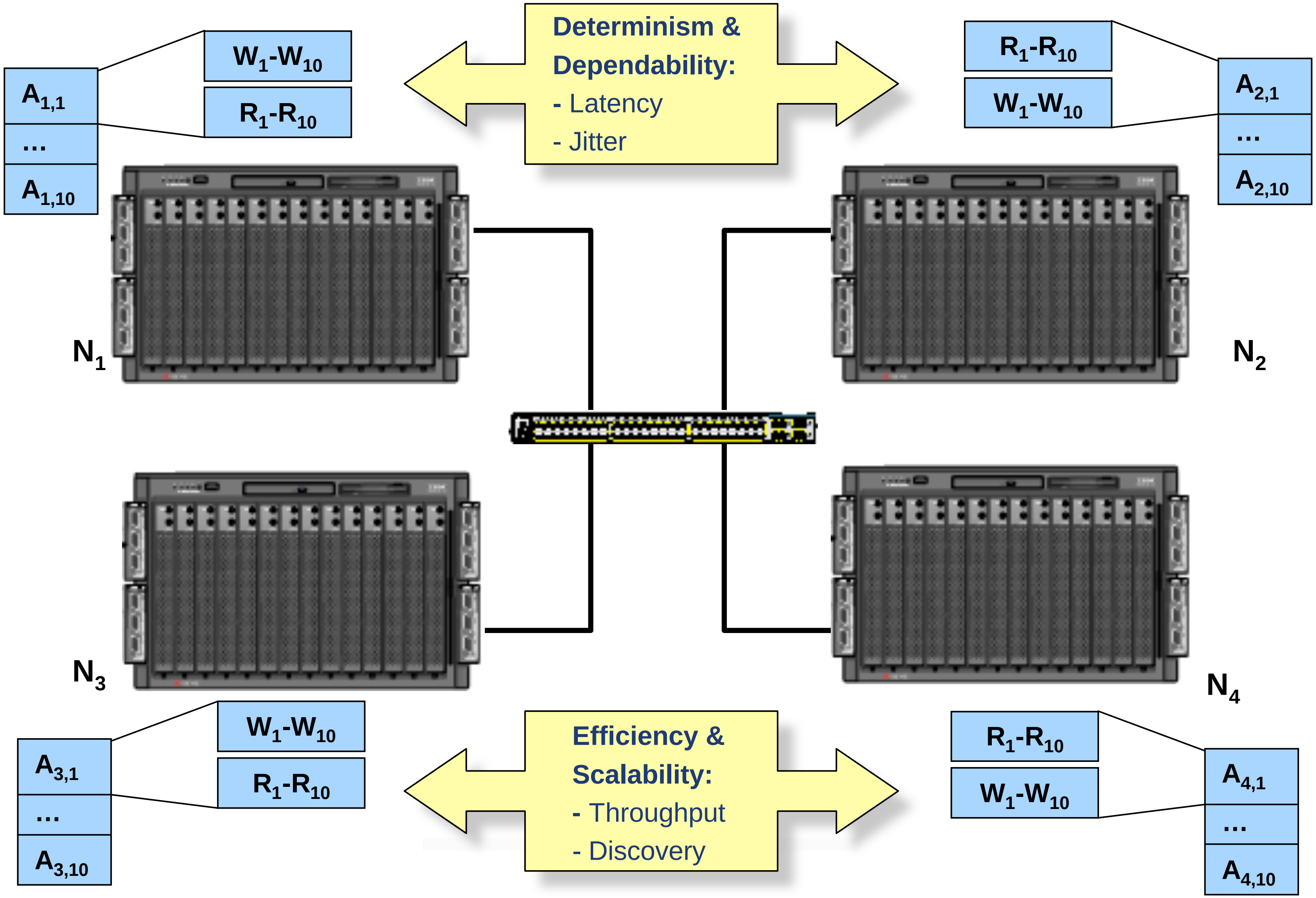
- Compose an **representative application benchmark** by providing a generic scenario-driven benchmarking suite that is fully DDS_compliant (and thus vendor independent)
- Provide the full benchmark (code, documentation, scenario's) to the DDS community
- Sollicitate feedback to improve/enhance the benchmark suite

Constraint

- Take to the minimum the number of nodes needed in order to showcase scalability/determinism superiority (one cannot expect users to have test-bed with hundreds of nodes)
- Make the test configurable so to be deployable on any number of nodes
- Assure the test is portable & vendor-agnostic i.e:
  - compliant to the DDS-specification
  - non-reliant on vendor-specific extensions

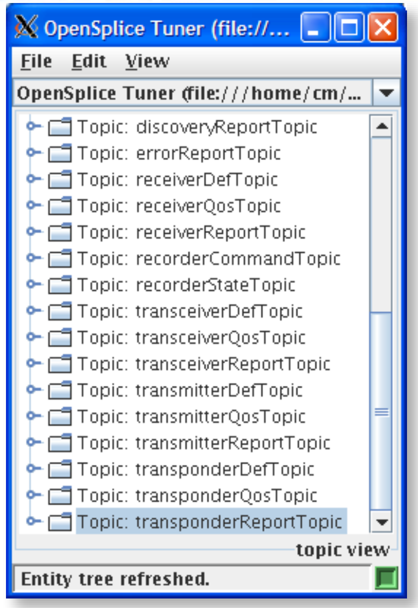
© 2007, PrismTech. All Rights Reserved

Benchmarking: Deployment example



© 2007, PrismTech. All Rights Reserved

Benchmarking Overview: Executables

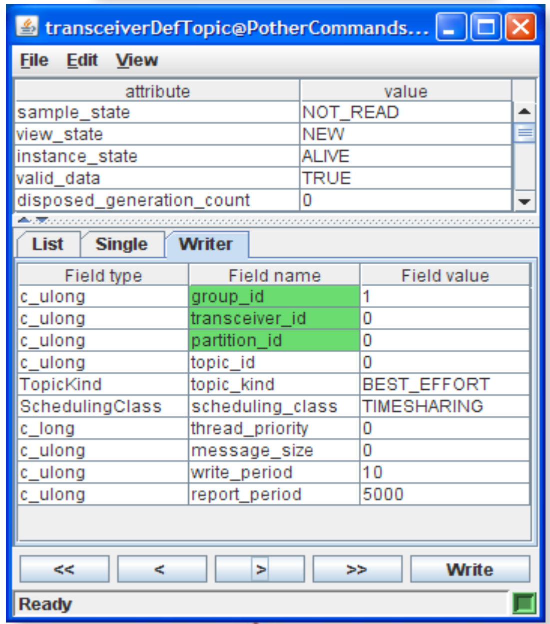


General characteristics:

- One generic program '***Pother***' to perform latency/discovery/throughput tests
- **Multiple instances** running at multiple machines
- Identified by <***application-Id***> and <***group-Id***> startup parameters
- Each *Pother* instance can perform **multiple tests** simultaneously
- Threads are **created dynamically** at proper scheduling-class/priority
- All interfacing (input *settings*, output *results*) done via **DDS Topics ➔**
- Relevant QoS policies can also be dynamically set/changed via Dedicated Topics

Pother Benchmark Suite Programs:

- *Pother* : main program (for latency/throughput/discovery testing)
- *Watcher* : basic result reporting for latency & throughput performance
- *Spotter* : basic result reporting for discovery performance
- *ErrorLog* : basic error reporting application
- *Recorder* : scenario logging & replay tool
- *Excellerator* : comma-separated logging (to feed into excel)

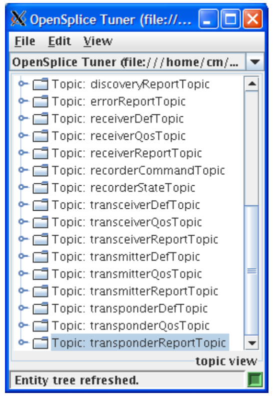


OpenSplice Tuner™ Usage:

- **To define, control & monitor the benchmarking**
  - By dynamic creation of Readers/Writers for any of the involved topics
  - This creation can be logged/replayed by the provided '*Recorder*' tool
- **Define**
  - Setup & adapt 'transceiver/transponder' pairs for latency/jitter testing
  - Setup & adapt 'transmitters' & 'receivers' for throughput/discovery testing
- **Control**
  - Set & change 'QoSTopics' to drive the behaviour of OpenSplice DDS
  - Typically: *Reliability, Transport-Priority, Latency_Budget & Deadline* QoS policies
- **Monitor**
  - By dynamic creation of Readers for the result topics ("*Watcher*" does this statically)
  - Typically: latency/jitter, throughput/discovery and notifications results

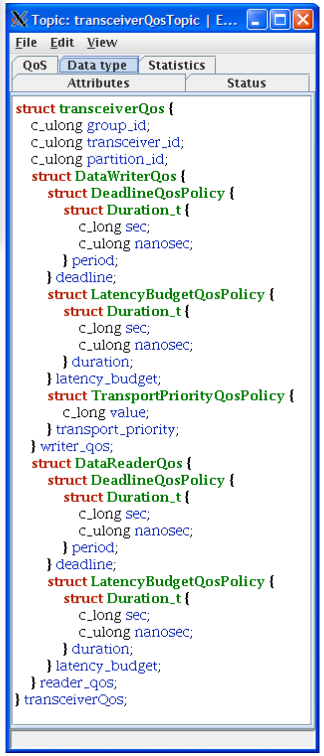
© 2007, PrismTech. All Rights Reserved

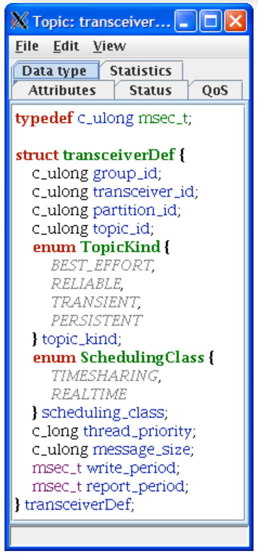
Overview: Operation of Pother main program

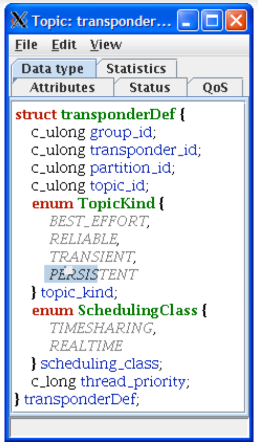


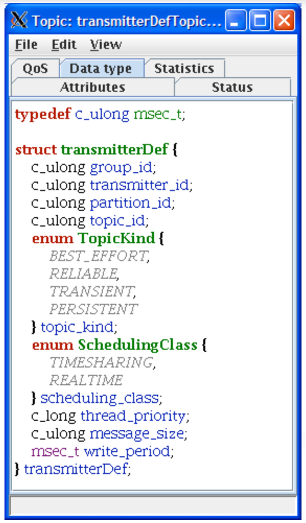
_Pother <application_id> [group_id]_

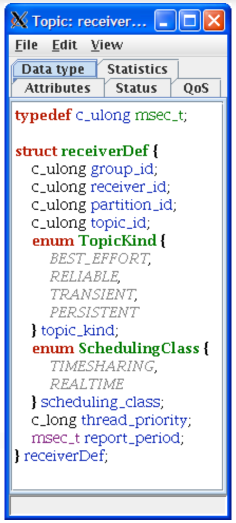
- ***Parameters***
  - *Application_id: unique identification for each Pother instance*
  - *Group_id (optional) : commands identification (to 'broadcast' commands)*
- ***Operation***
  - *Several Pother instances can be started on 1 or more computing nodes*
  - *to perform determinism (latency & Jitter) and efficiency (throughput & discovery) benchmarking*
- ***Determinism (Latency & Jitter) benchmarking***
  - *RTT (Round Trip Time) & Jitter measurement between 'transceiver' and 'transponder'*
  - *Driven by topics: Transceiver Definition/QoS topics and Transponder Definition/QoS topics*
- ***Efficiency (Throughput & Discovery) benchmarking***
  - *Point-to-point Throughput and reader/writer discovery times benchmarking*
  - *Driven by topics: Transmitter Definition/QoS topics and Receiver Definition/QoS topics*

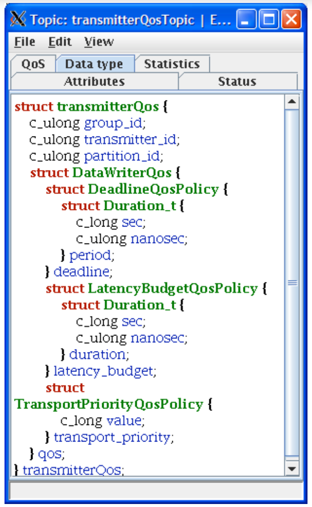






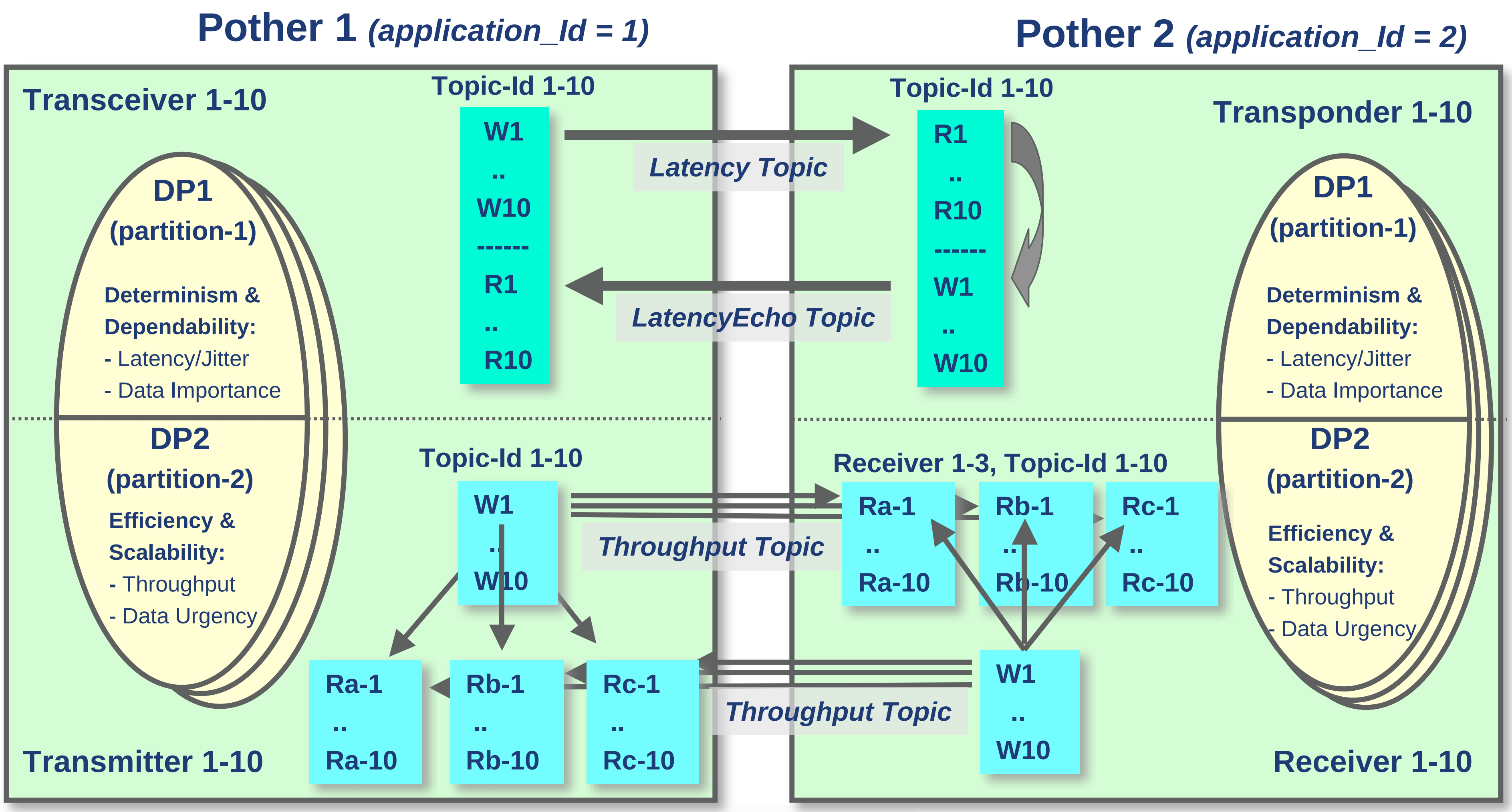






© 2007, PrismTech. All Rights Reserved

Pother Design: Overview



© 2007, PrismTech. All Rights Reserved

Example Benchmark: SCALABILITY (throughput/efficiency)

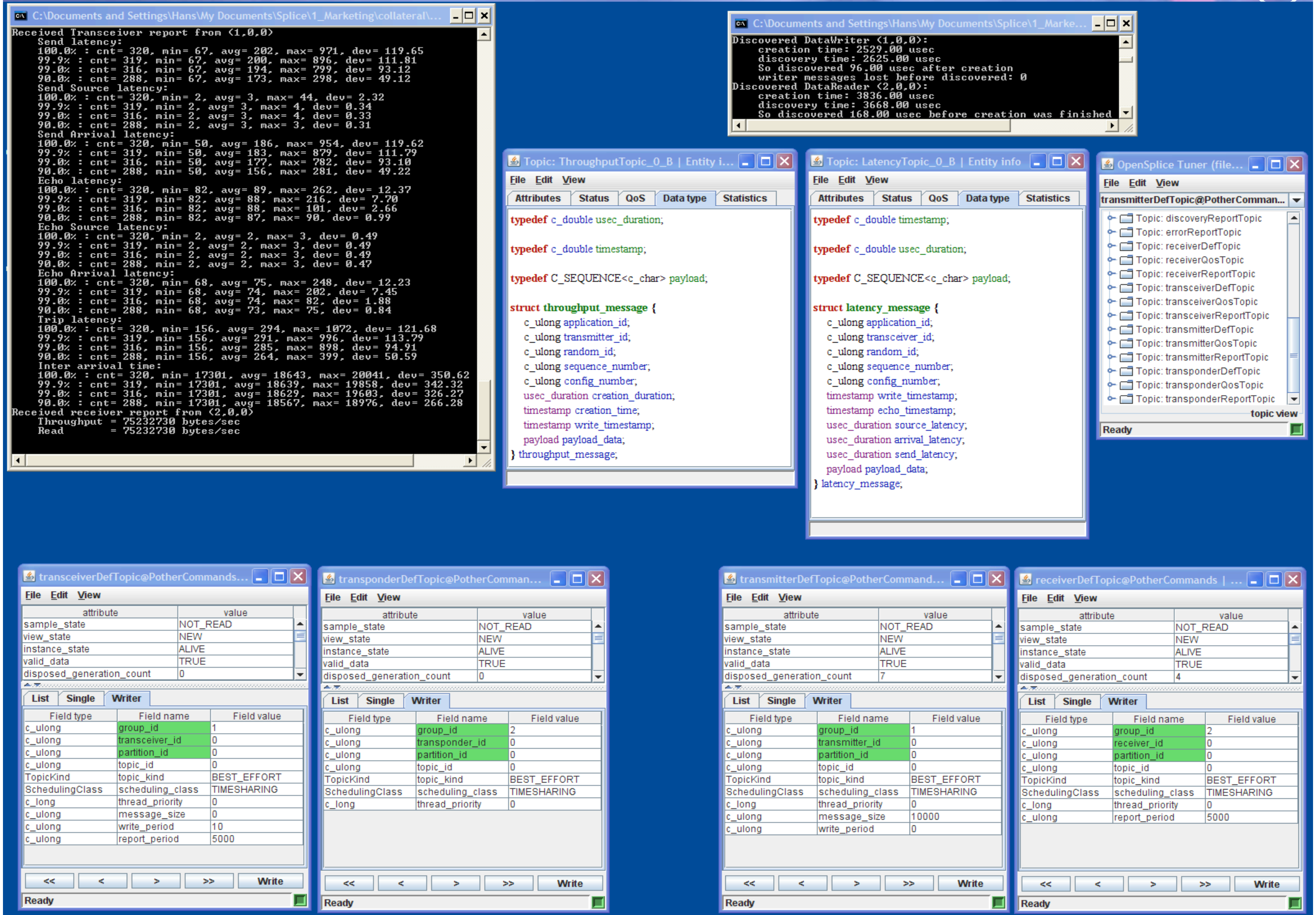
- **Goal**: Measure (one-way) throughput, report throughput (theoretical/achieved) per application/node/system
  - With 4 distributed nodes & 10 applications per node
  - With 30 readers and 10 writers per application, so a total of **1,600 readers/writers** (400 writers, 1,200 readers)
- **Per application**: 10 writers for 10 broadcast topics and 3*10=30 readers for these 10 topics
  - i.e., data going to readers in all applications (including 'own') in all nodes (including 'own')
  - Writing: 10 writers (W1-10) write 10 different topics (T1-10)
  - Reading: 30 readers (Ra1-10, Rb1-10, Rc1-10) read these to different topics (T1-10)
  - So a single reader reads data from: 4 nodes * 10 App's * 1 writer/app = 40 writers
  - So per application 3 * 10 = 30 readers receive data from 10 * 40 = 400 writers (at 4 nodes)
- **Node**/Network impact:
  - Data Frequency, Size and Urgency **settings**
    - Frequency (each writer): **2 Hz writes** for each writer
    - Size (topic payload size): **1,000 bytes**
    - Latency budget**:** **200 msec**
  - TX **network** traffic per node
    - 10 applications * 10 writers/application = 100 writers that write 1,000 bytes at 2Hz: **200 Kbyte/s**
    - Packing: typically sends at 5 Hz (instead of 200), meaning a **packing of 40 messages** (40 Kbytes)
  - Rx **network** Traffic: 3 nodes * 200 Hz
    - So 600 Hz * 1000 Bytes = **600 Kbyte/s**
- **Calculated (theoritical) throughput**

Reader-throughput	= 40 writers * 2 Hz * 1,000 bytes	= 80 Kbyte/s
Application-throughput	= 30 readers * 80 Kbyte/s	= 2.4 Mbyte/s
Node-throughput	= 10 applications * 2.4 Mbyte/s	= 24 Mbyte/s
System-throughput	= 4 nodes * 24 Mbyte/s	= 96 Mbyte/s

- **Actual (measured) throughput (on 4 Linux DELL bladeservers, dual Opteron 2.4 Ghz. CPU's)**
  - Actual throughput measured = 96 Mbyte/s (Same as theoretical) at < 50 % CPU load

© 2007, PrismTech. All Rights Reserved

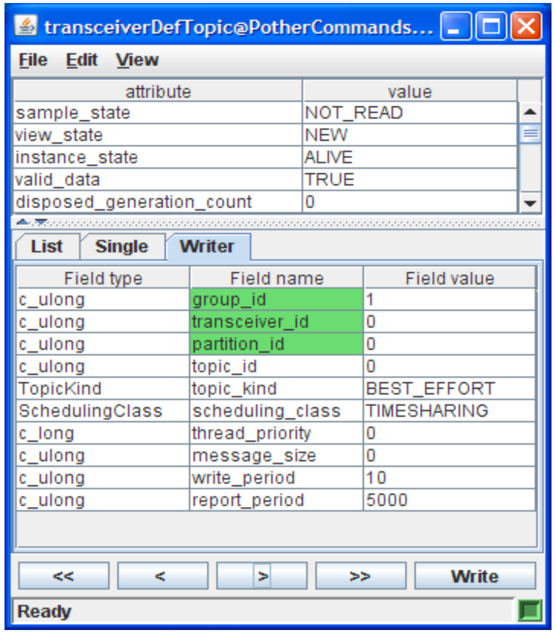
Overview: Example (Windows-XP)



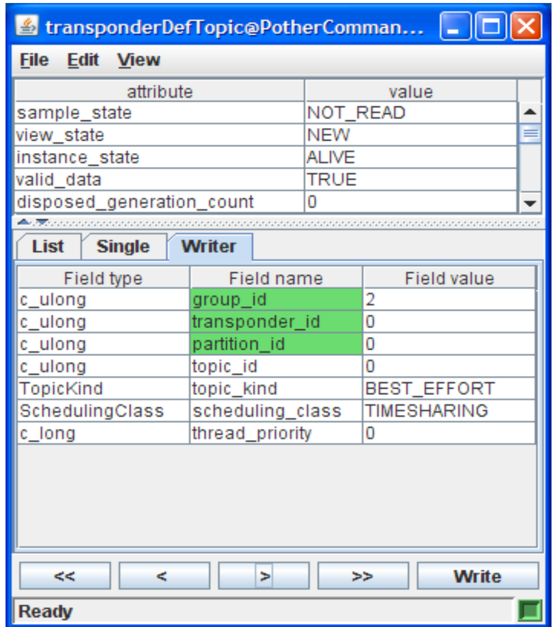
Determinism benchmarks: Latency

T1 = Data about to be written
T2 = Middleware has the data

T3 = Data has been delivered
T4 = Data has been read



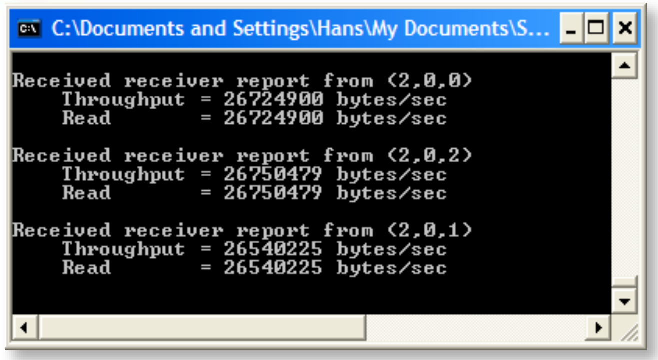

```
Received Transceiver report from (1,0,0)
    Send latency:
    100.0% : cnt= 319, min= 81, avg= 87, max= 149, dev= 9.95
    99.9% : cnt= 318, min= 81, avg= 87, max= 145, dev= 9.33
    99.0% : cnt= 315, min= 81, avg= 86, max= 128, dev= 7.52
    90.0% : cnt= 287, min= 81, avg= 84, max= 92, dev= 2.05
    Send Source latency:
    100.0% : cnt= 319, min= 2, avg= 3, max= 4, dev= 0.36
    99.9% : cnt= 318, min= 2, avg= 3, max= 4, dev= 0.35
    99.0% : cnt= 315, min= 2, avg= 3, max= 4, dev= 0.34
    90.0% : cnt= 287, min= 2, avg= 3, max= 3, dev= 0.25
    Send Arrival latency:
    100.0% : cnt= 319, min= 65, avg= 70, max= 126, dev= 7.91
    99.9% : cnt= 318, min= 65, avg= 69, max= 113, dev= 7.26
    99.0% : cnt= 315, min= 65, avg= 69, max= 110, dev= 5.96
    90.0% : cnt= 287, min= 65, avg= 67, max= 74, dev= 1.69
    Echo latency:
    100.0% : cnt= 319, min= 85, avg= 88, max= 107, dev= 2.93
    99.9% : cnt= 318, min= 85, avg= 88, max= 106, dev= 2.73
    99.0% : cnt= 315, min= 85, avg= 88, max= 103, dev= 2.16
    90.0% : cnt= 287, min= 85, avg= 87, max= 89, dev= 0.66
    Echo Source latency:
    100.0% : cnt= 319, min= 1, avg= 2, max= 3, dev= 0.50
    99.9% : cnt= 318, min= 1, avg= 2, max= 3, dev= 0.50
    99.0% : cnt= 315, min= 1, avg= 2, max= 3, dev= 0.50
    90.0% : cnt= 287, min= 1, avg= 2, max= 3, dev= 0.48
    Echo Arrival latency:
    100.0% : cnt= 319, min= 72, avg= 74, max= 85, dev= 1.90
    99.9% : cnt= 318, min= 72, avg= 73, max= 84, dev= 1.79
    99.0% : cnt= 315, min= 72, avg= 73, max= 83, dev= 1.47
    90.0% : cnt= 287, min= 72, avg= 73, max= 74, dev= 0.59
    Trip latency:
    100.0% : cnt= 319, min= 170, avg= 177, max= 257, dev= 12.52
    99.9% : cnt= 318, min= 170, avg= 177, max= 254, dev= 11.71
    99.0% : cnt= 315, min= 170, avg= 176, max= 232, dev= 9.53
    90.0% : cnt= 287, min= 170, avg= 174, max= 183, dev= 2.40
    Inter arrival time:
    100.0% : cnt= 319, min= 18099, avg= 18642, max= 18798, dev= 46.99
    99.9% : cnt= 318, min= 18099, avg= 18641, max= 18789, dev= 46.24
    99.0% : cnt= 315, min= 18099, avg= 18640, max= 18737, dev= 44.77
    90.0% : cnt= 287, min= 18099, avg= 18634, max= 18683, dev= 41.75
```



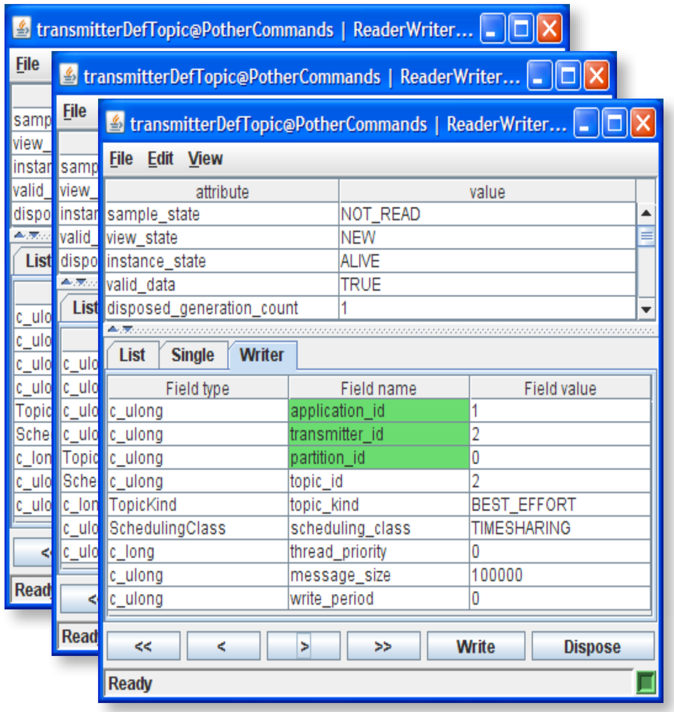
Example output for Windows
(XP, 2.0Ghz Xeon, local-latency)

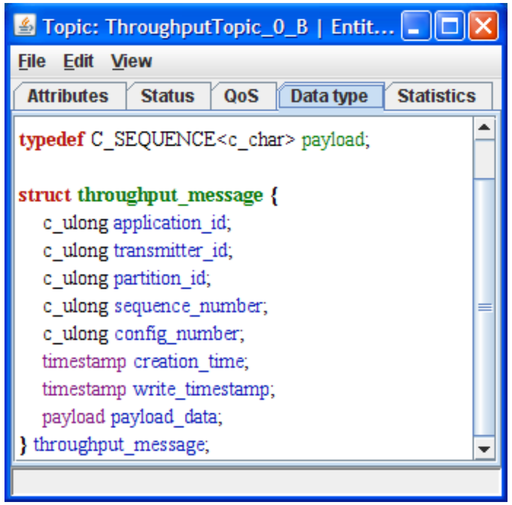
Scalability benchmarks: Throughput

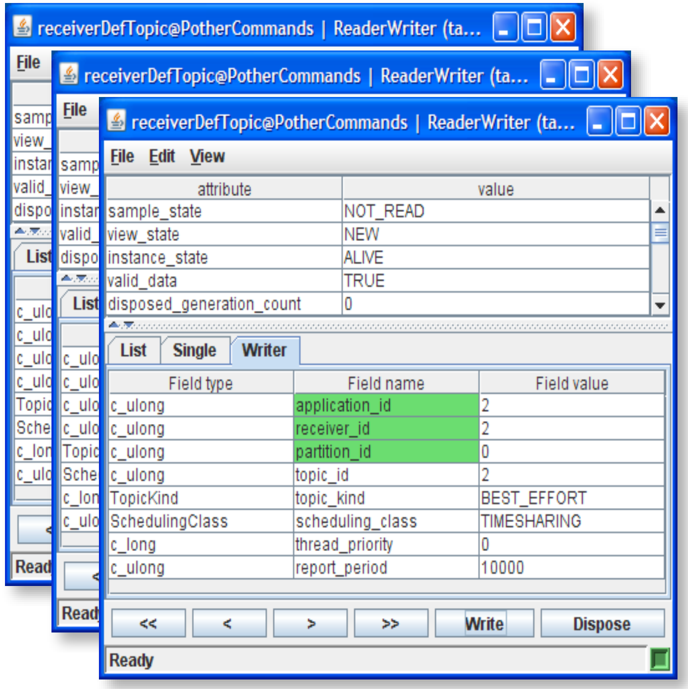
- ***Transmitter(s) with defined size & frequency (period)***
- ***Receiver(s) report throughput:***
  - *Theoretical Throughput, based upon sequence-numbers*
  - *Actual 'Read' throughput, based upon amount of data read*
  - *Those can differ (if system gets overloaded):*
    - *if reader can't keep-up (history-depth = 1, keep_last)*
    - *If samples are lost during transport (reported as 'error')*



Aggregate Throughput = 80 Mbyte/s
(XP, 2.0Ghz Xeon, local-throughput)
(on dual-CPU Opteron: 450 Mbyte/s)



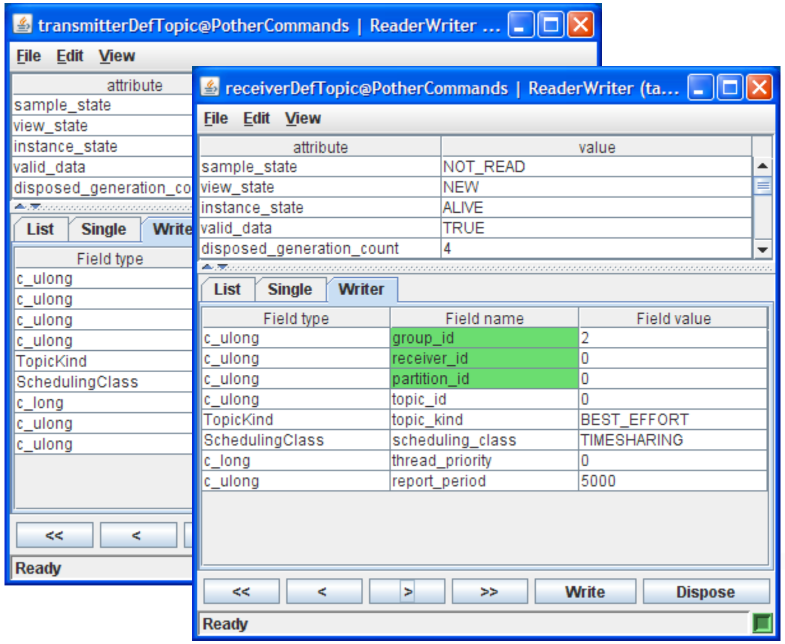


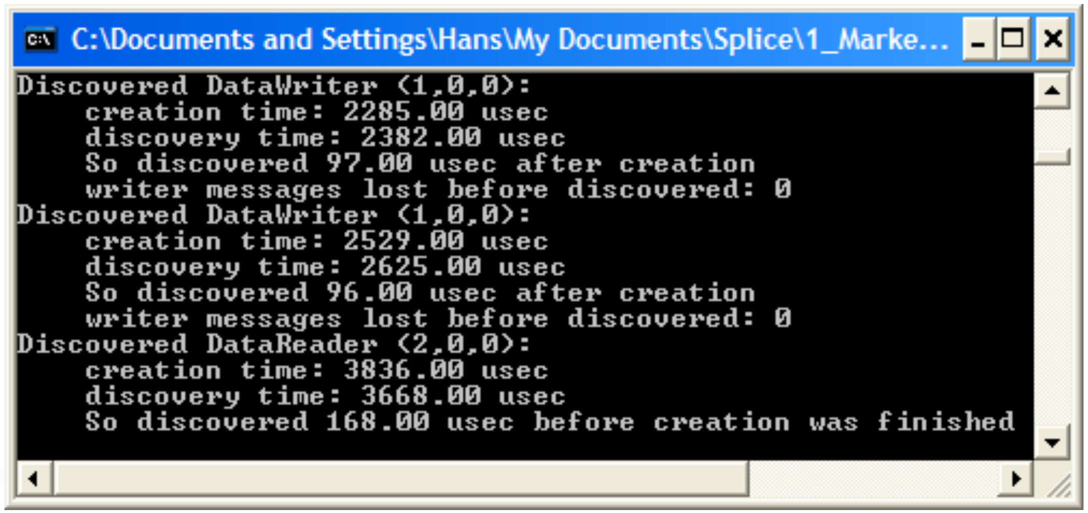


© 2007, PrismTech. All Rights Reserved

Scalability Benchmarks: Discovery

- ***Discovery definition***
  - *"after entity creation, how long before entity actually active"*
- ***Reader Discovery (adding a reader to system with writers)***
  - *After entity creation, how long before reader discovered & first data actually received*
- ***Writer Discovery (adding a writer to system with readers)***
  - *After entity creation, how long before writer discovered & first data actually received*





© 2007, PrismTech. All Rights Reserved

Writer/Reader discovery on Linux cluster

Writer discovery within one node

Writer at 1,000 Hz, 40 bytes

Create writer

- Creation-time 784 usec
- Discovery-time **791** usec
- **So discovered 7 usec. after creation**
- lost 0 msgs. before discovered

Create Participant/Publisher/Writer

- Creation-time 1021 usec
- Discovery-time 1038 usec
- **So discovered 17 usec. after creation**
- lost 0 msgs. Before discovered

Reader discovery within one node

Writer at 1,000 Hz, (*so 1 ms 'extra' worst-case*), 40 bytes

Create reader

- Creation-time 1103 usec
- Discovery-time **866** usec (see above)
- **So discovered 237 usec. before creation finished**

Create Participant/Subscriber/Reader

- Creation-time 1270 usec
- Discovery-time **1157** usec
- **So discovered 113 usec. before creation finished**

Writer Discovery between nodes

Writer at 1,000 Hz, 40 bytes

Create writer

- Creation-time 621 usec
- Discovery-time **624** usec
- **So discovered 3 usec. after creation**
- lost 0 msgs. Before discovered

Create Participant/Publisher/Writer

- Creation-time 2459 usec
- Discovery-time **2461** usec
- **So discovered 2 usec. after creation**
- lost 0 msgs. before discovered

Reader discovery between nodes

Writer at 1,000 Hz, (*so 1 ms 'extra' worst-case*), 40 bytes

Create reader

- Creation-time 1218 usec
- Discovery-time **1241** usec
- **So discovered 23 usec. after creation**

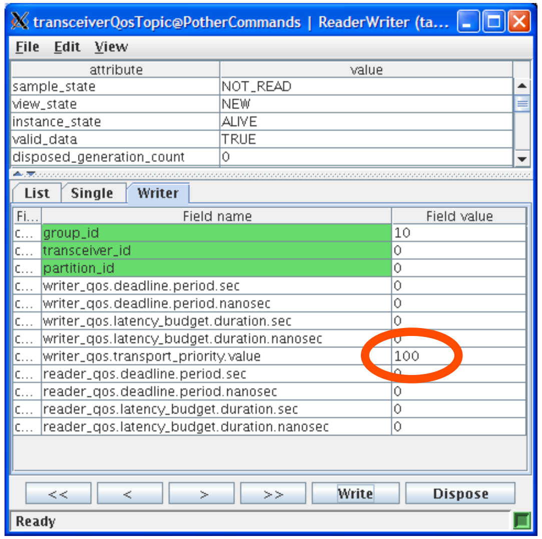
Create Participant/Subscriber/Reader

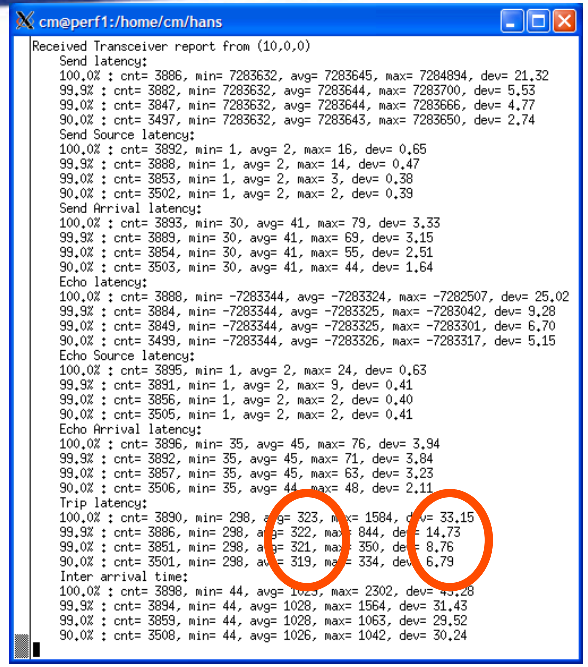
- Creation-time 1253 usec
- Discovery-time **1280** usec
- **So discovered 27 usec. after creation**

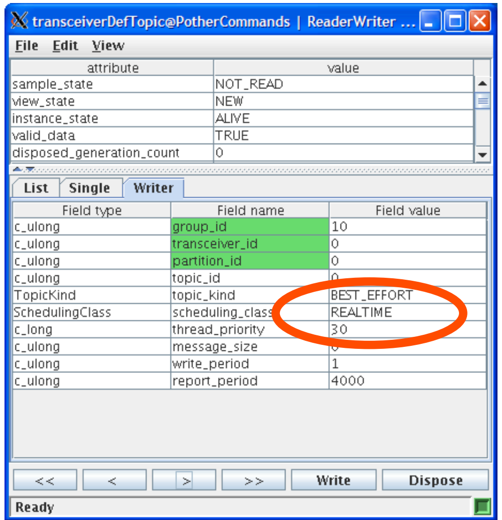
© 2007, PrismTech. All Rights Reserved

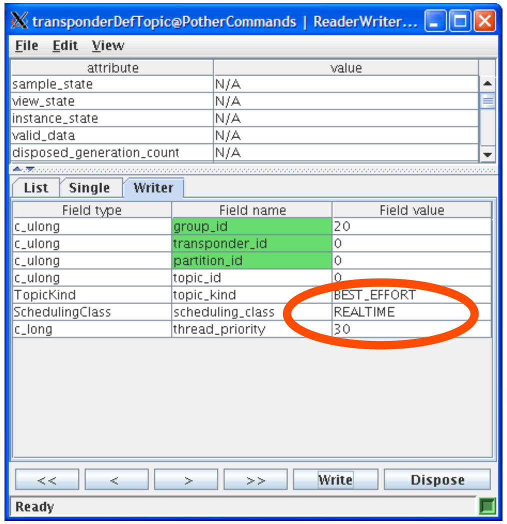
Network Latency figures on Linux (application on RT/30)

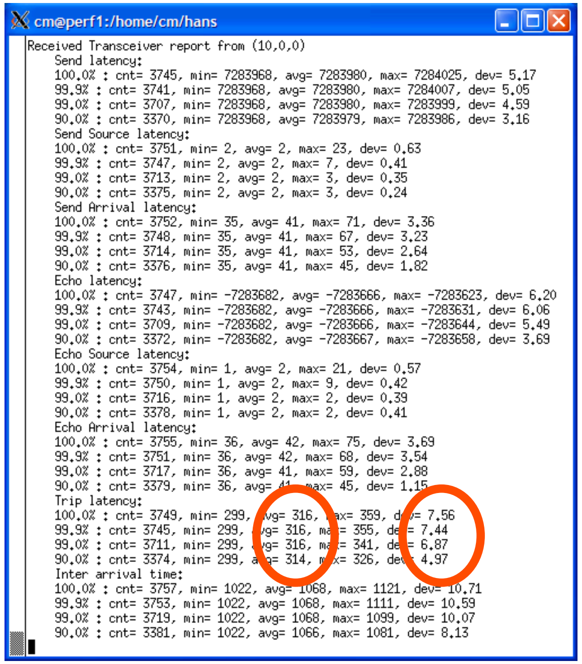
- ***Configuration***
  - *transceiver/transponder at RT/30 prio*
  - *Transport priority 0 and 100*
  - *1000 Hz. Frequency*
- ***Roundtrip at Transport_Priority 0***
  - *Lowest priority band (timeshare)*
  - *Roundtrip latency = 323 usec.*
  - *Roundtrip Jitter < 34 usec*
- ***Roundtrip at Transport_Priority 100***
  - *Highest priority band (realtime)*
  - *Roundtrip latency = 316 usec*
  - *Roundtrip jitter < 8 usec (!!)*







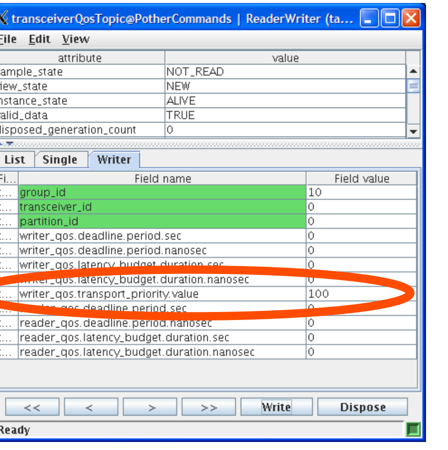


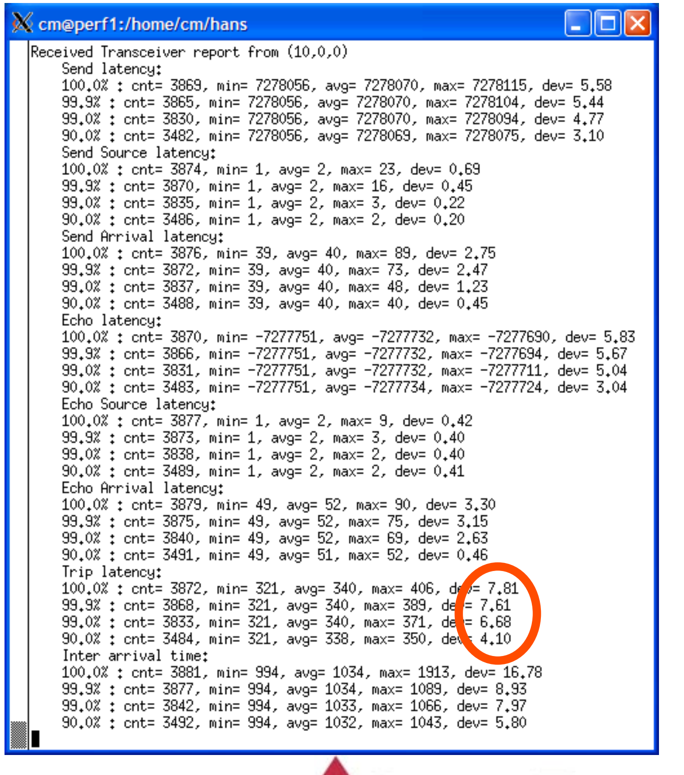


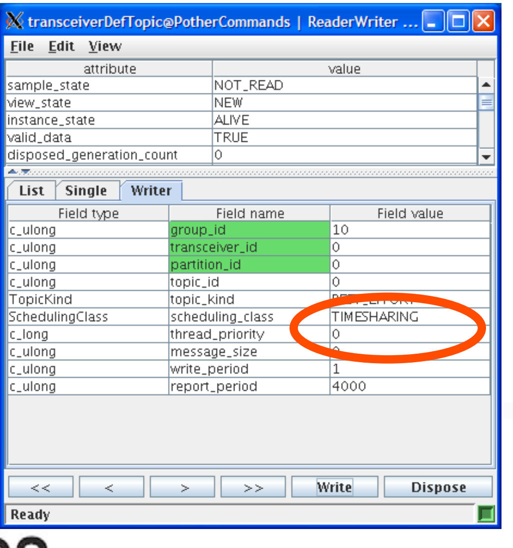
© 2007, PrismTech. All Rights Reserved

Network Latency figures on Linux (application on TS)

- ***Configuration***
  - *transceiver/transponder at Timeslicing / prio-0*
  - *1000 Hz. Frequency*
  - *Payload = 0 bytes*
- ***Roundtrip at Transport_Priority 100***
  - *Highest priority band (realtime)*
  - *Roundtrip latency = 316 usec*
  - *Roundtrip jitter < 8 usec (!!)*
- ***Conclusion***
  - ***Even low-prio applications can send high-prio data***
  - ***With extreme low latency***
  - *Send-Arrival Latency only can have some jitter*

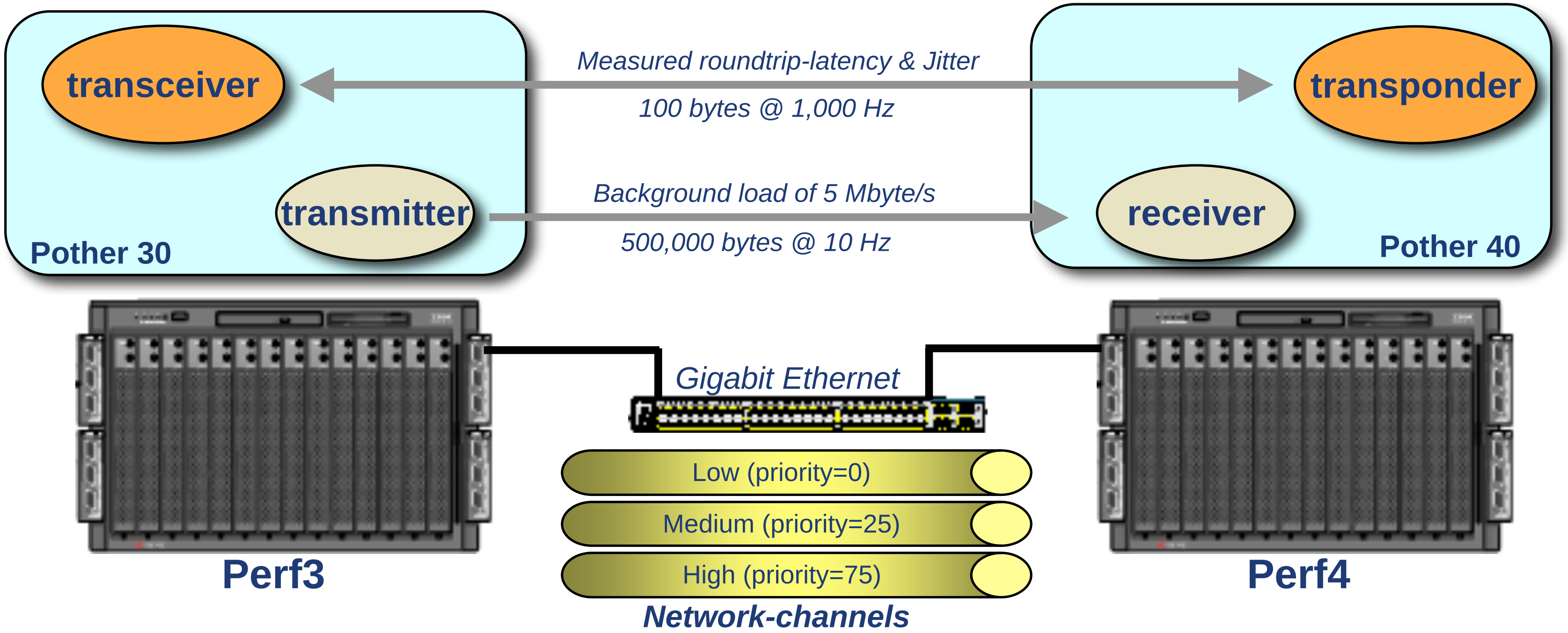






© 2007, PrismTech. All Rights Reserved

DEMO



Goal:

- Measure **determinism** (roundtrip jitter) under heavy background load
- Determine impact of **information-priority** (TRANSPORT-PRIORITY) versus **application priority** (OS scheduling prio)
- Show that low-application priority process can send high-priority information with low Jitter & low Latency

Configuration:

- Background load: “Transmitter” & “Receiver” at Real-time/Prio-30 (**high appl. Prio**), Transport-priority 0 (**low info prio**)
- Network-channels for **3 priority-bands**: 0-25 (base-prio), 25-75 (medium prio), >75 (high-prio)
  - Low-prio channel : *Tx/Rx threads at timesharing priority 0, DIFSERV priority 0x0*
  - Medium-prio channel : *Tx/Rx threads at Real-time OS priority 25, DIFSERV priority 0x2*
  - High-prio channel : *Tx/Rx threads at Real-time OS priority 75, DIFSERV priroity 0x4*

© 2007, PrismTech. All Rights Reserved

Agenda

- OpenSplice DDS Overview
- Architecting Distributed Systems
- What is “Performance”
- OpenSplice DDS Architecture
- OpenSplice DDS Deployment Tools
- The “Pother” benchmarking suite
- Demo
- Whats Next

Dr. Angelo Corsaro

© 2007, PrismTech. All Rights Reserved

Concluding Remarks

Architecture

- OpenSplice DDS has an architecture that **explicitly targets determinism and scalability** for real-time and mission-critical distributed systems
- OpenSplice DDS provides **full OMG-DDS rev1.2 functional coverage** combining pub/sub messaging with elaborate information management
- OpenSplice DDS can therefore significantly **reduce system complexity** and enhance component re-use while maintaining proper performance levels

Performance & Tuning

- OpenSplice DDS's advanced network-scheduler utilizes several DDS QoS policies to **optimize efficiency** while maintaining **proper determinism**
- OpenSplice DDS's federated architecture provides **excellent scalability** while maintaining **low discovery** times
- OpenSplice Tuner™ provides **system-monitoring and performance tuning** capabilities for local as well as remotely deployed nodes

OpenSplice DDS is the best implementation available on the market providing the highest-performance DDS solution!

© 2007, PrismTech. All Rights Reserved

Contact Us

- OpenSpliceDDS Home Page
  - http://www.prismtech.com/opensplice-dds/
- For Information on OpenSplice DDS contact:
  - opensplicedds@prismtech.com -or-
  - sales@prismtech.com
- OMG DDS Information
  - http://www.dds-forum.org/
  - http://portals.omg.org/dds/

© 2007, PrismTech. All Rights Reserved