

# NextGen Performance

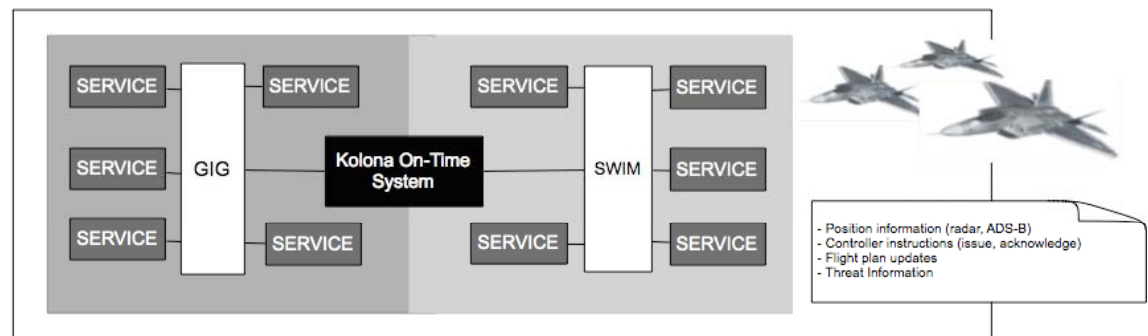
Agility, Timeliness and Linear Scalability

## TCP/IP Example SBIR - AFRL

### Military Aircraft in Civilian Airspace Worldwide

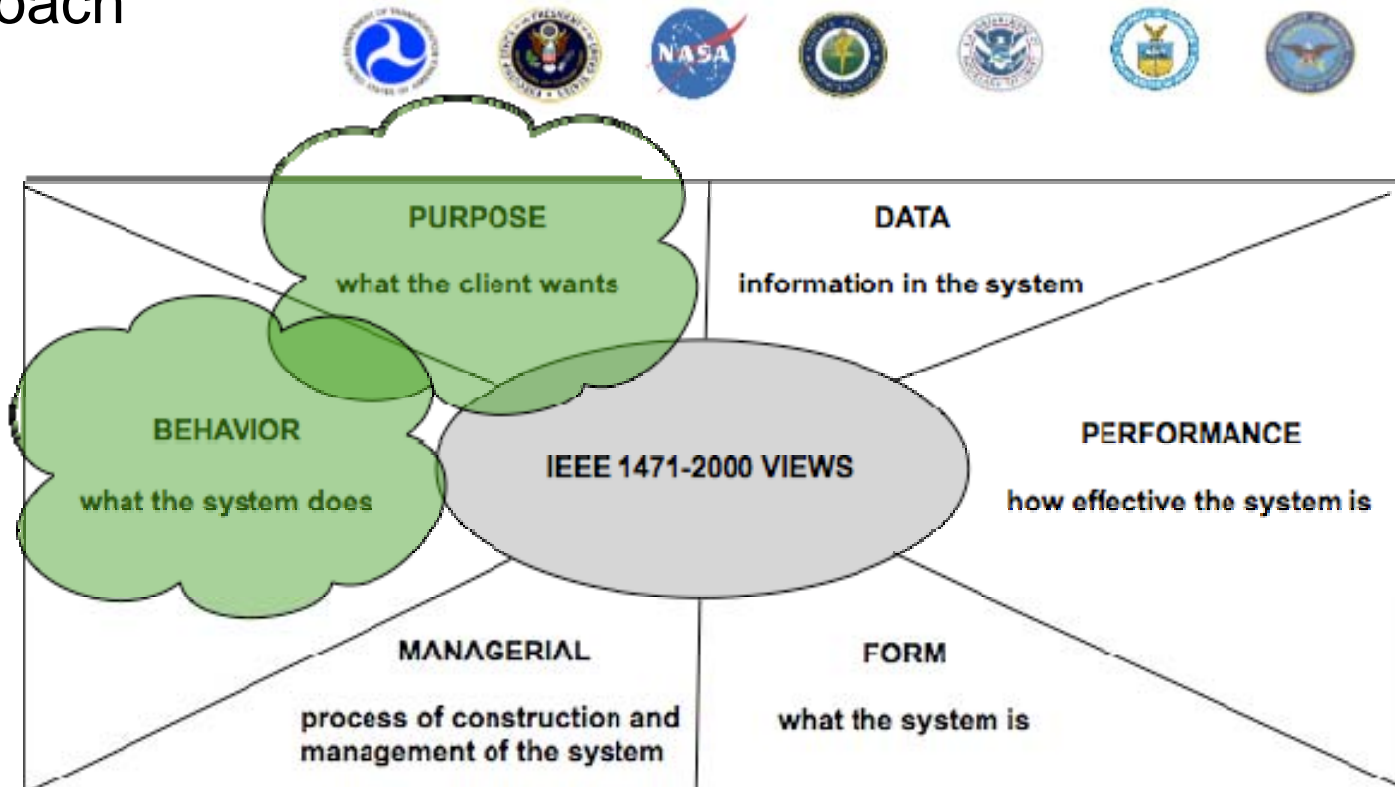
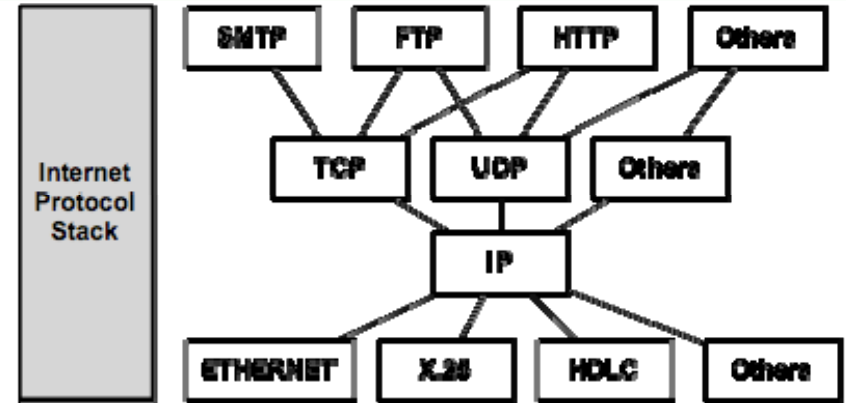
Face Facts

- 2.3 seconds enroute
- 1.5 seconds departure and descent
- 1.0 seconds runway
- 3 times todays volume :: latency the problem / separation
- cross-domain
- multilevel security
- QoS: Ultraquality



# NextGen

- TCP/IP Collaborative Networks
- JDPO: DoT, FAA, NASA, DHS, DoD, DoC: Diverse purposes and relatively complex behaviors compared to the SESAR approach

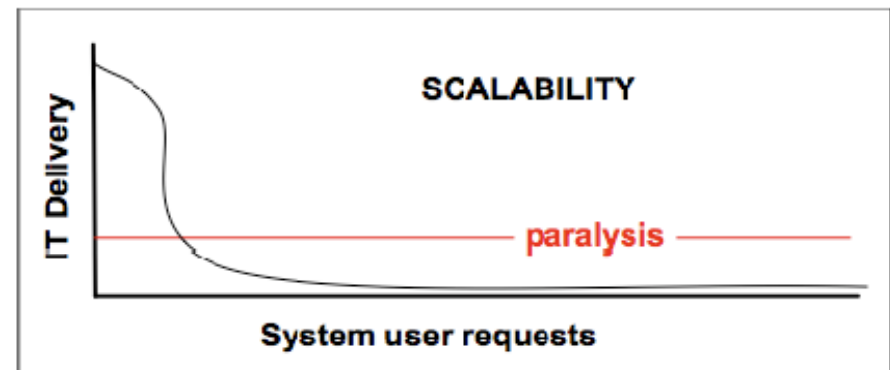
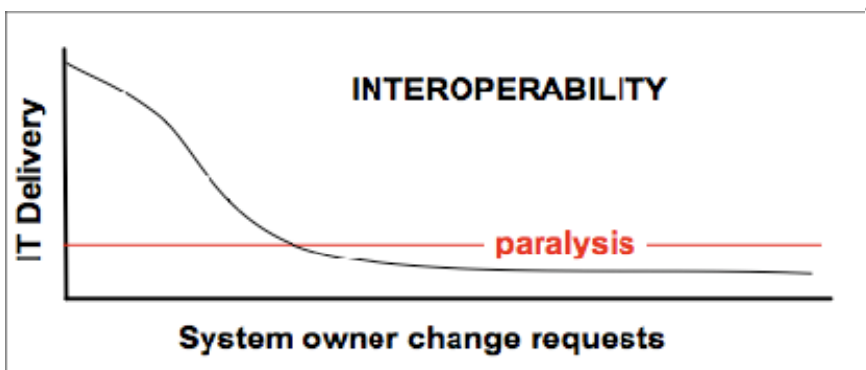


# NextGen Functional Challenges

TODAY	NextGen
Ground-based Navigation and Surveillance	Satellite-based Navigation and Surveillance
Air Traffic Control Communications by Voice	Routine Information Sent Digitally
Disconnected Information Systems	Information More Readily Accessible
Air Traffic Control (ATC)	Air Traffic Management
Fragmented Weather Forecasting	Forecasts Embedded into Decisions
Airport Operations Limited by Visibility Conditions	Operations Continue into Lower Visibility Conditions
Forensic Safety Systems	Prognostic Safety Systems

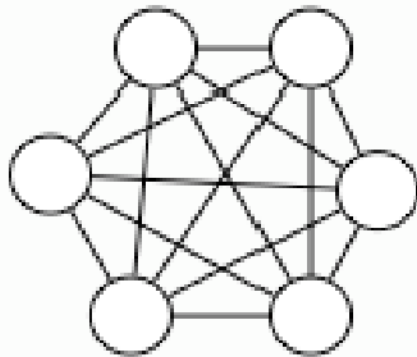
# NextGen Non-Functional Challenges

accessibility	extensibility	recoverability	accountability	adaptability
administrability	affordability	agility	availability	composability
configurability	customizability	degradability	demonstrability	dependability
distributability	durability	embedability	evolvability	failover
flexibility	installability	integrability	interoperability	maintainability
manageability	mobility	modifiability	modularity	nomadicity
openness	performance	portability	predictability	reliability
retractability	reusability	robustness	scalability	seamlessness
security	serviceability	simplicity	stability	survivability
tailorability	testability	understandability	usability	



# Point-to-Point versus Integrator

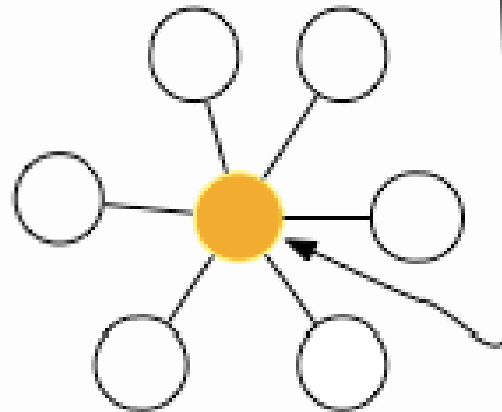
Triangular Figurate Number  
of Adapters



**Point-to-Point**

**SOA / ESB**

Bus  
(Integrator)



*(1) Add Location/Routing  
(2) Decouple Integration*

# Enterprise Service Bus (ESB)

- Anonymous
- Producer initiates conversation
- Event-based
- Adds location

*(1) Add Location/Routing  
(2) Decouple Integration*

**Integration Logic Decoupled**

		+	-
+	<b>ESB</b>	<b>JMS</b>	
-	<b>EAI</b>	<b>JEE</b>	

**Distributed**

# Java Messaging Service (JMS)

- Asynchronous

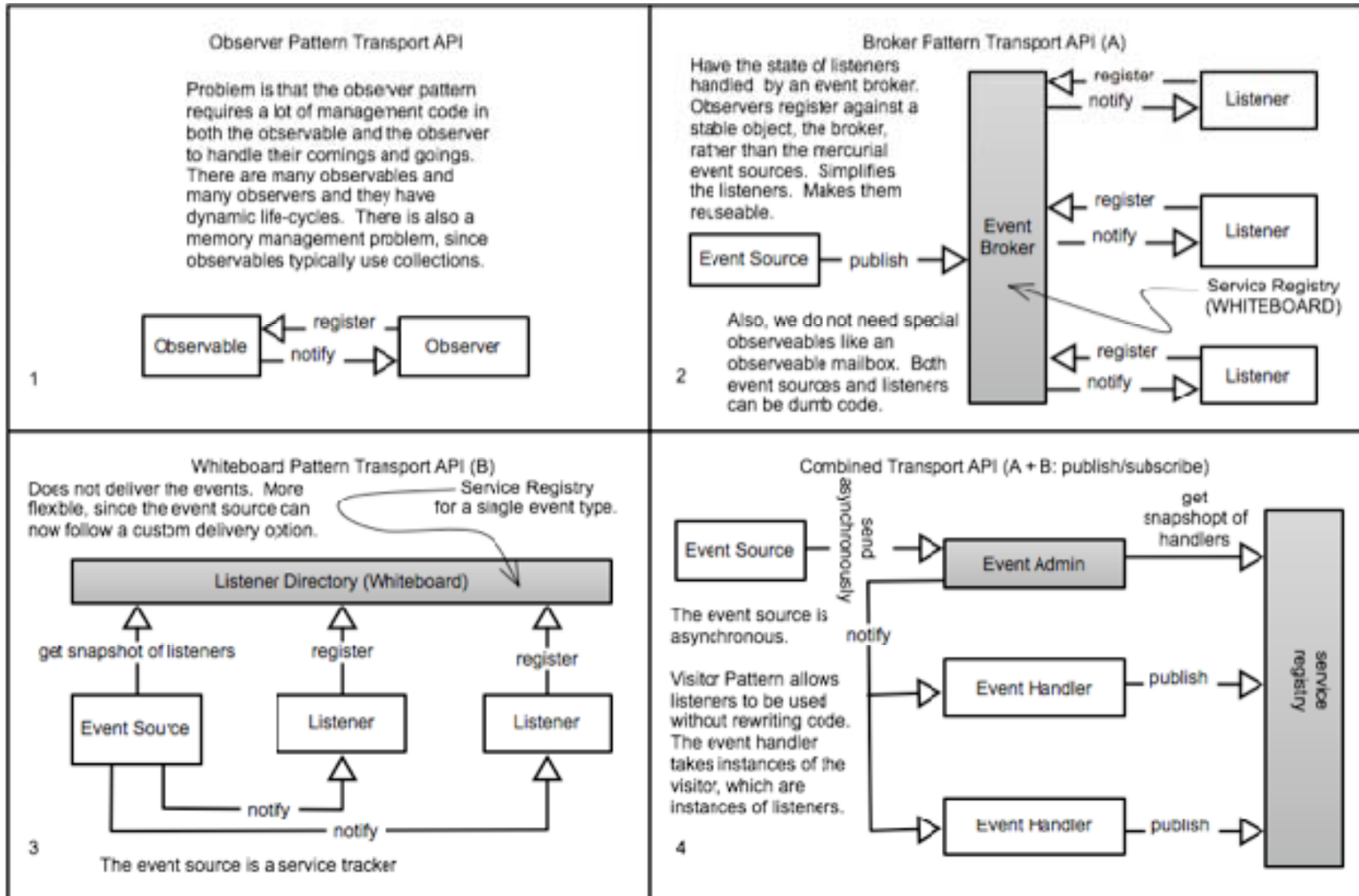
Application	Business Logic
ESB	Integration Logic
JMS	Transport Logic
Sockets	Transport

**Addressee Known**

		+	-
-	<b>call-back</b>	<b>event-based</b>	
+	<b>request/reply</b>	<b>anonymous request/reply</b>	

**Consumer Initiated**

# JMS Broker Virtualization



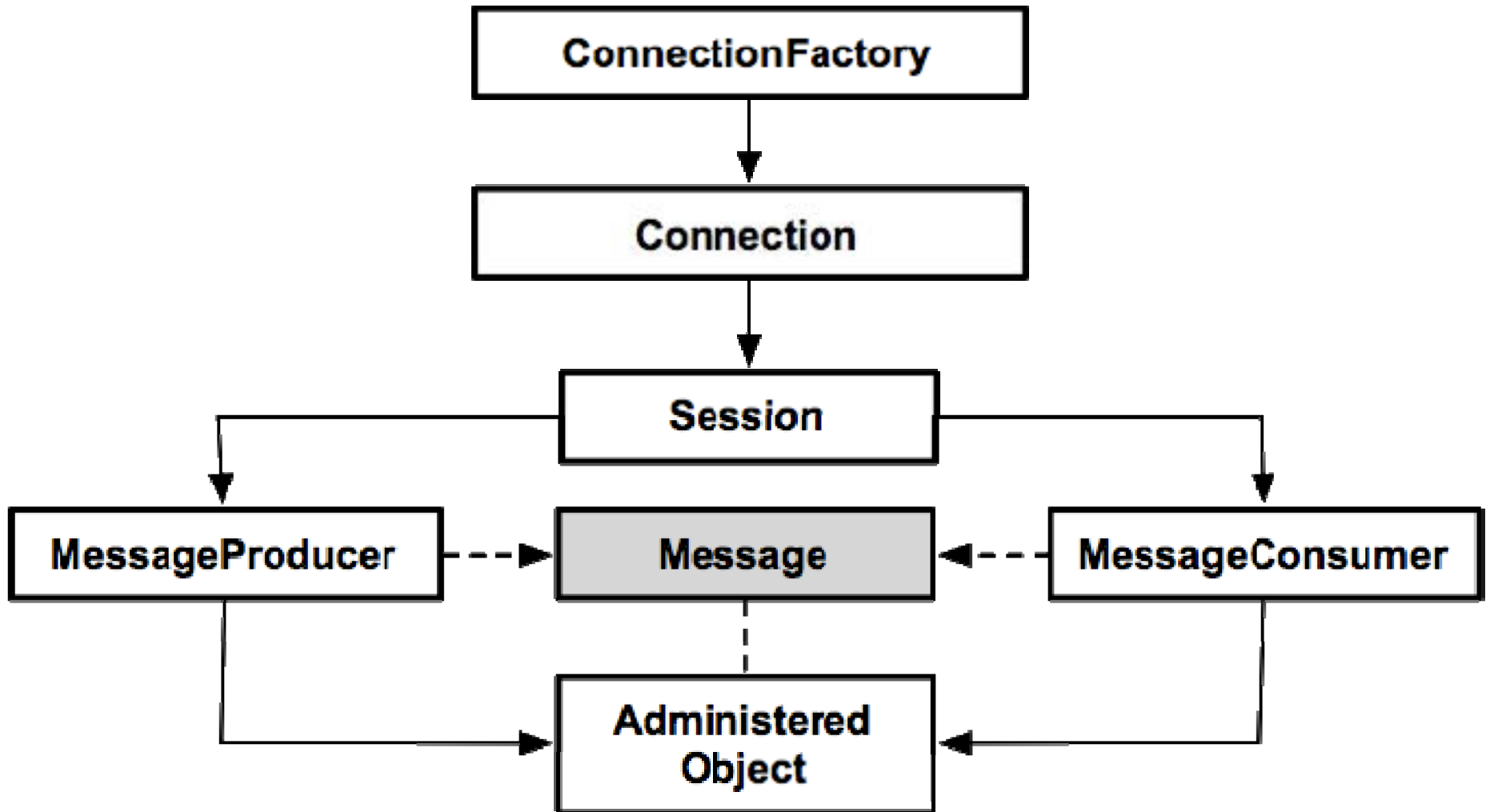
# Data Distribution Service (DDS) Broker Free Manifesto

Can your  
broker do  
this?

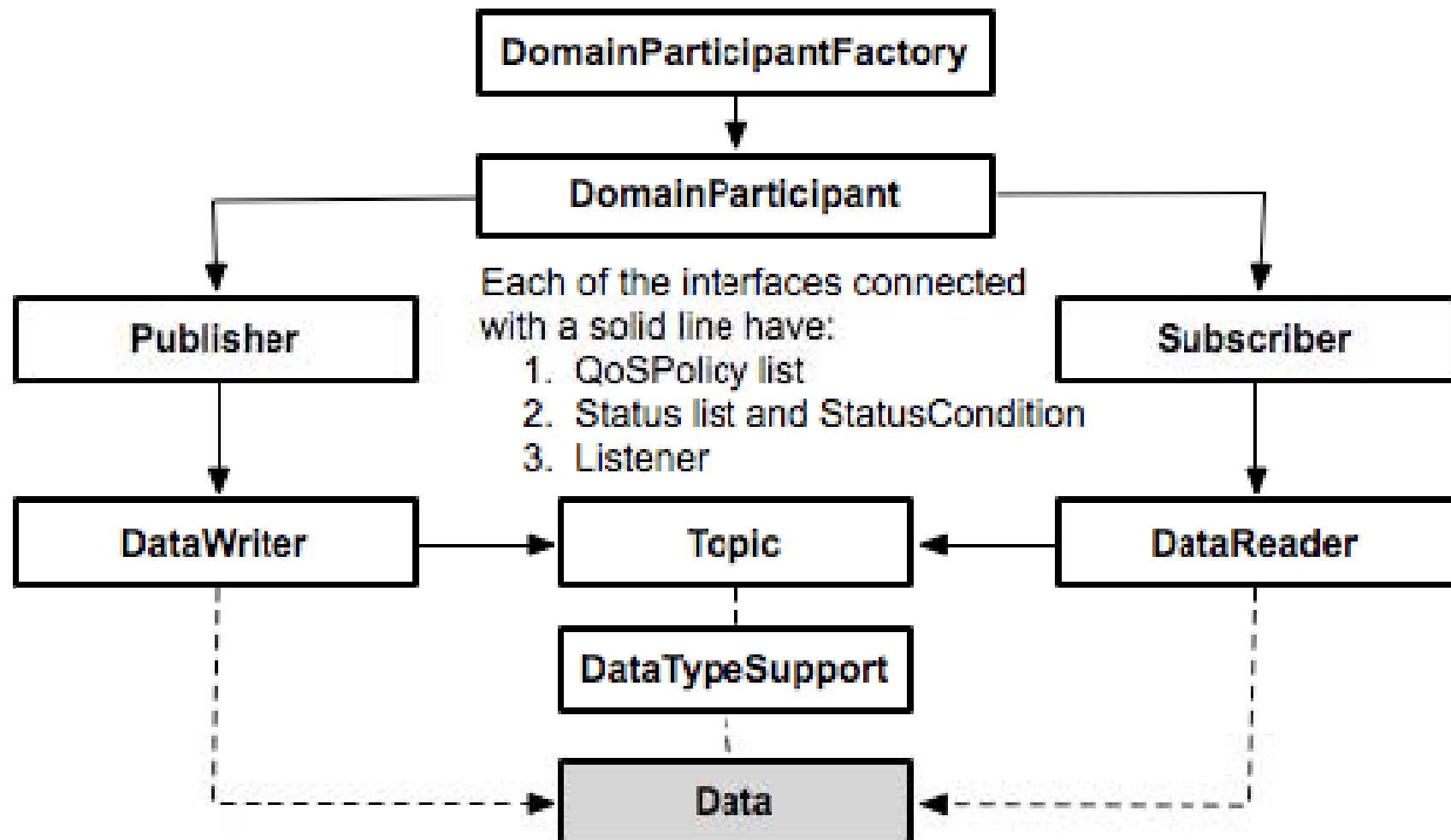


- ⑩ Systems are limited by middleware throughput
- ⑩ Systems using brokers at orders of magnitude disadvantage
  - With brokers: network hops, software layers, operating system context switches.
  - Without brokers: **DATA-CENTRIC**
    - Only limits are the network: hundreds of millions of network operations per second are possible, provided application endpoints have linear scalability. Minimized latency and improved security.
    - No data metadata - data caching - access at memory speed rather than query speed.

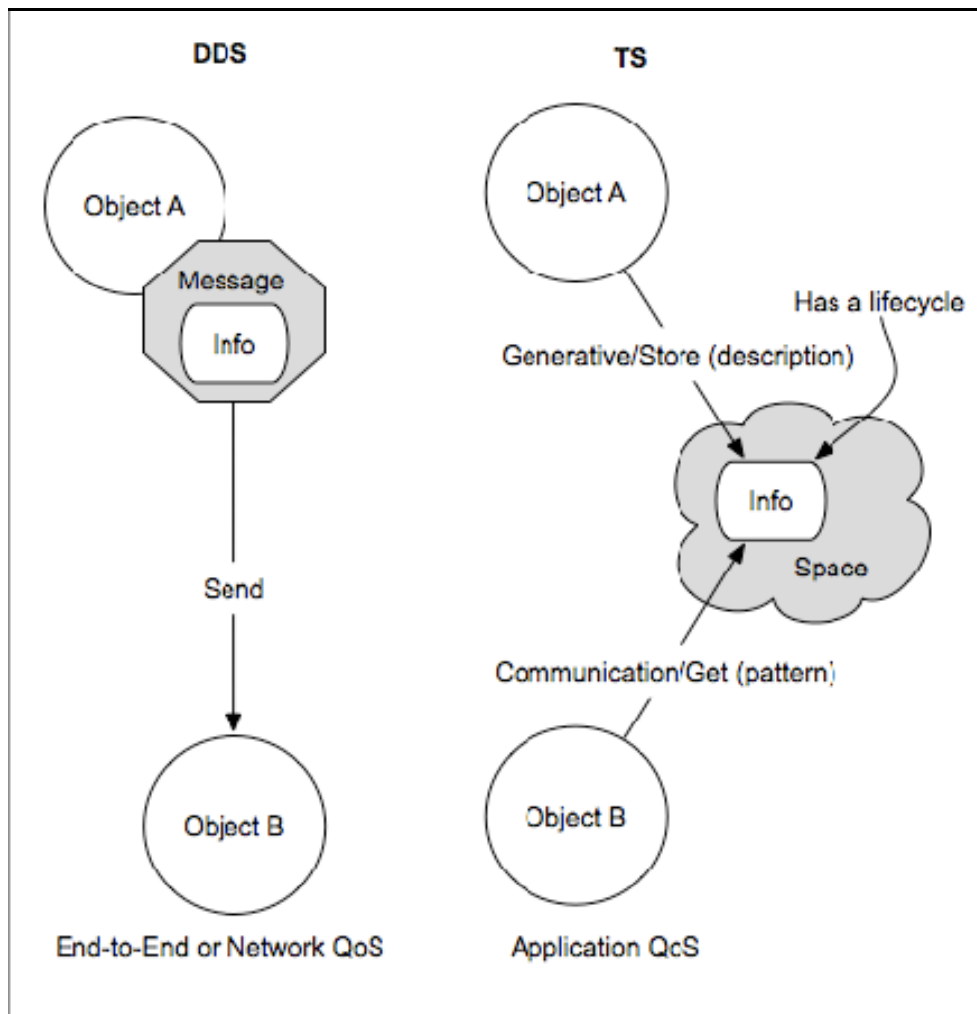
# JMS: Data Model = Application Level



# DDS: Data Model = Middleware Level In-Memory = Messaging Level State



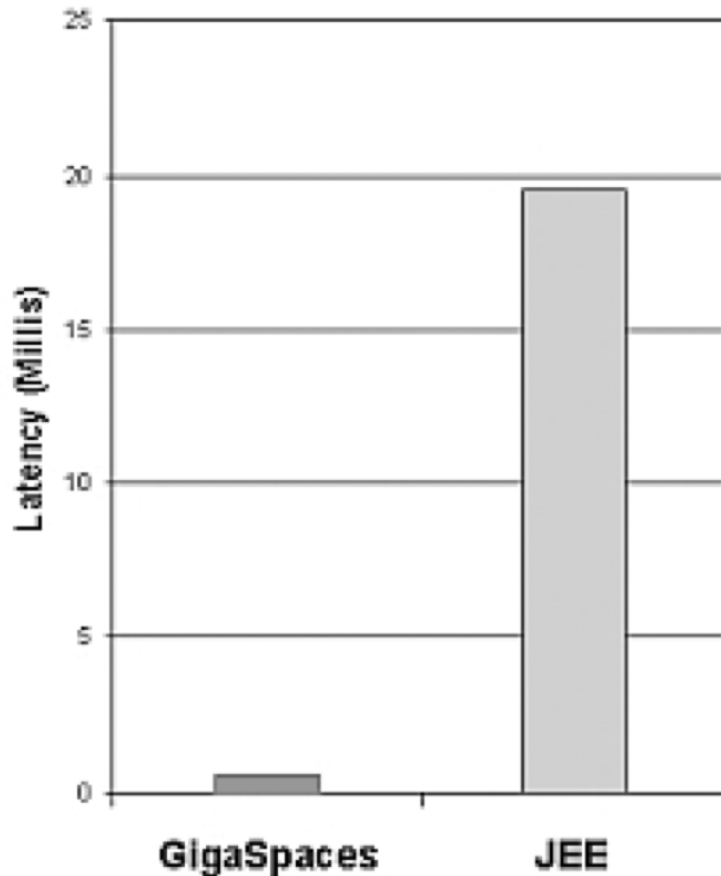
# Tuple Spaces: Data Model = System Level In-Memory = Application State



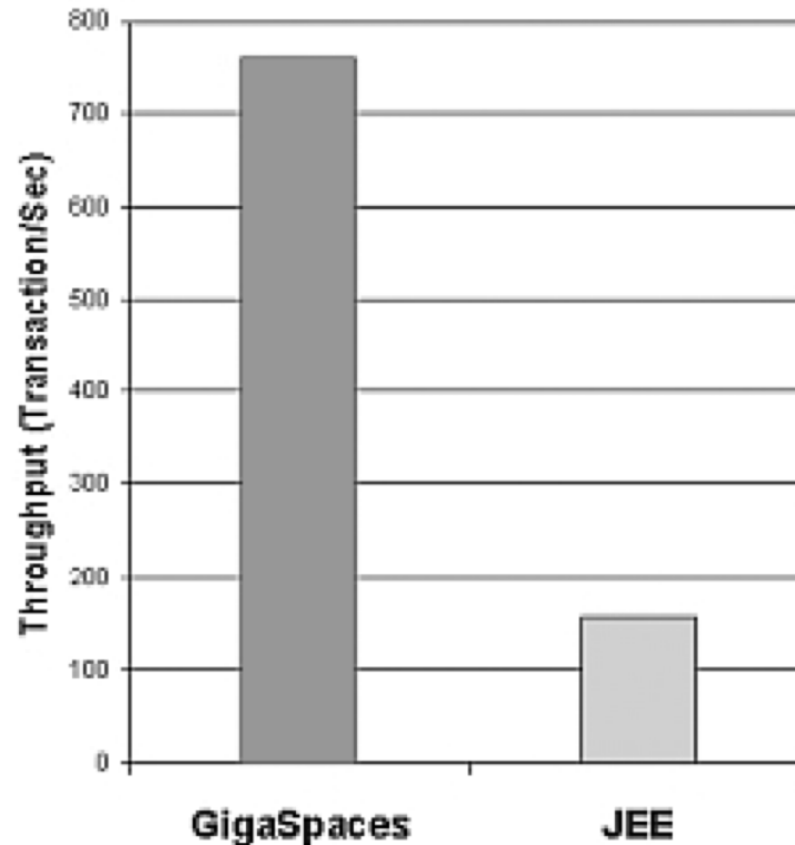
- DDS: Very fast messaging
- DDS: Looses anonymity and asynchronicity
- DDS: Problems with the “ilities”
- TS: Different level anti-brokering
- TS: Info belongs to system, has a lifecycle

# Linear Scalability and Higher Throughput

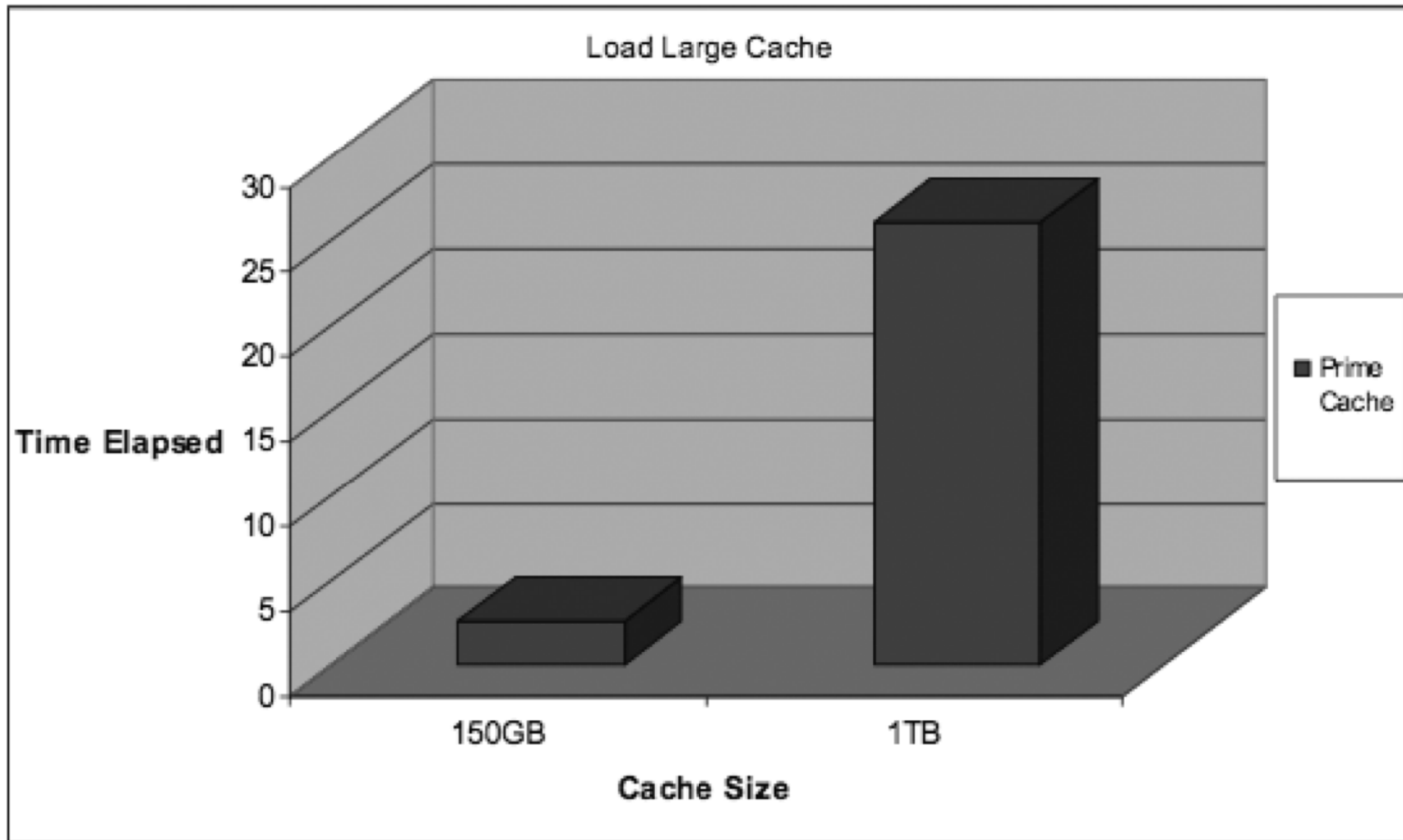
**Latency - JEE vs GigaSpaces  
(Lower Is Better)**



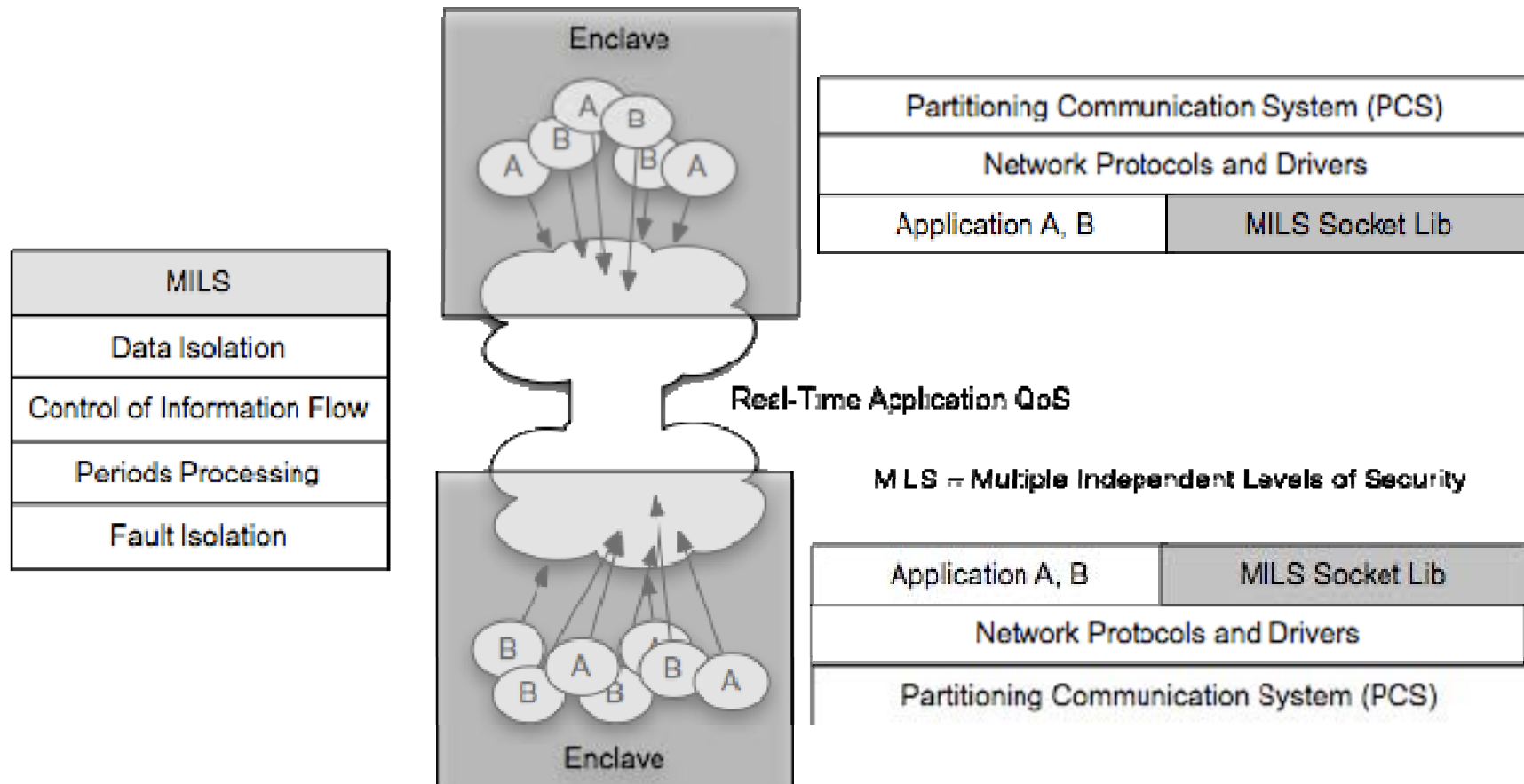
**Throughput - JEE vs gigaSpaces  
(Higher Is Better)**



# Prime Cache Scaling (GigaSpaces)



# Cross Domain : Multilevel Security



# Tuple Spaces Scalability Focused

1. **Parallel query support** – Client running special thread pool used to query each Data Grid instance in parallel manner. Final result aggregated at the client side. ***Provides constant deterministic response time no matter the Data Grid size.***
2. **Parallel method execution support** – Client running a special thread pool that transfers business logic object to the different Data Grid instances to be executed at the same memory address where the data is stored. ***Provides constant and deterministic processing time no matter the amount of Data Grid instances or data to process.***
3. **Data eviction support** – Data Grid evicts data based on amount of free memory, amount of objects or based on user custom business logic. This allows the Data-Grid to act as a front end cache for one or more databases or external source resources. ***This allows the system to scale its backend resources.***
4. **Optimized Database Access** – When multiple concurrent clients trying to access the same data object , and a cache miss happens, the lazy load operation (database read operation) performed only once.
5. **Local cache support** – Client running a 2<sup>nd</sup> level cache that is loaded on demand. ***Avoid extra unneeded calls to the Data-Grid in case of repeated read calls for the same data object.***
6. **Large data objects support** – Spaces provides smart packet buffering handling, insures client and server socket buffer would not be overloaded. ***Provides stable, robust and deterministic data flow.***
7. **Large data set handling** – Smart data set iterator over distributed Data Grid instances. Allows clients to access vast data sets without impact on application or Data Grid stability

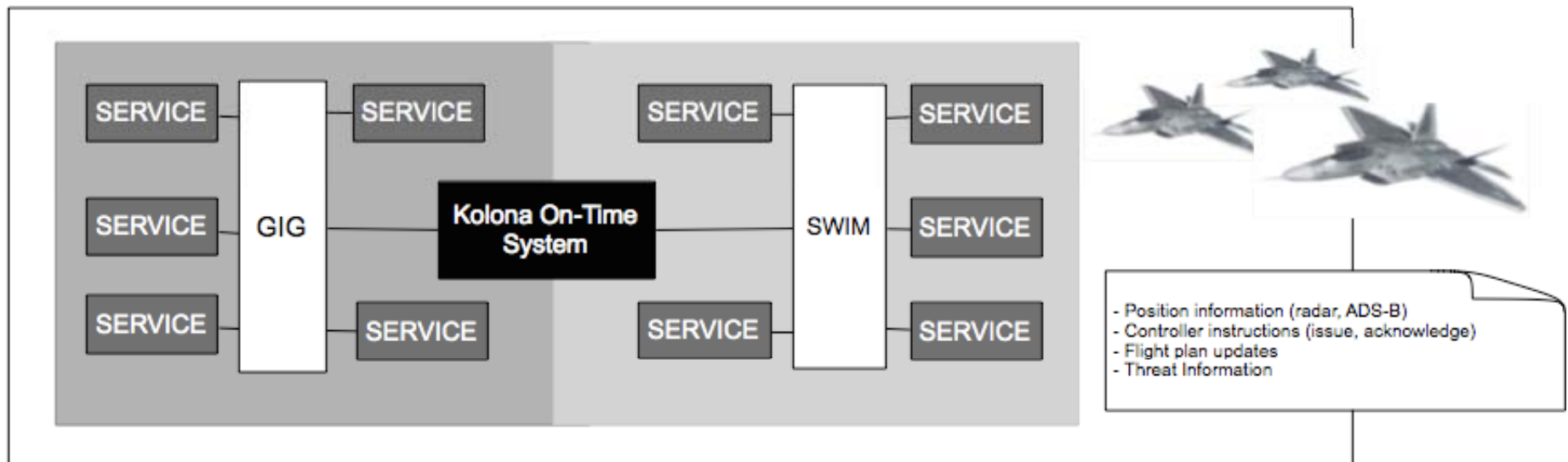
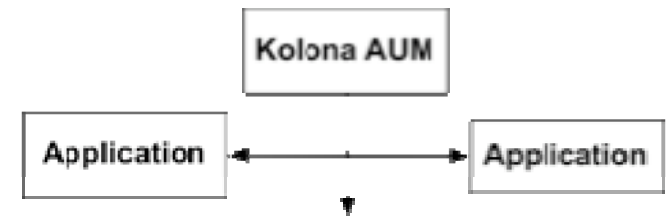
# Tuple Space High Availability Focused

1. **Automatic self healing** – Automatic data grid instance provisioning in case of a failure. Allows the system to recover itself – **this means that data objects will always have a primary and at least one backup located within the Data Grid.**
2. **Data recovery** – Data Grid instance running as a replica or a backup recovers its data from the primary via multi parallel replication channel. **Insures fast recovery.**
3. **Standard Data admin interfaces** – Data Grid administration and management can be done via JMX standard API. **Allows users to plug-in Data Grid management into any Enterprise management tools.**

# Tuple Spaces Performance Focused

1. **Fast data query support** - Data is indexed in-memory using Hash and AVL index. Allows very fast search without the need to scan the data.
2. **Native SQL support** – SQL Query is parsed natively and translated to Data Grid operations. No need to use 3<sup>rd</sup> party to query the data in-memory or rely on the database to fetch matching data.
3. **Generic universal object storage** - Object Data is transported to a special data structure that is maintained in memory allows the Data Grid to materialize the data into every client runtime object structure in a native manner – i.e. Java Client Write Java Object into the Data grid , .Net or C++ client can read it and vice versa. Allows fast data interoperability without the need to use 3rd party protocol or database.
4. **Data compaction support** - Non primitive fields can be compressed. Optimized network utilization.
5. **Data serialization control** - at the client and server side. Optimized network utilization.
6. **Data replication and replication control** - Data replication filtering based on object content or target Data Grid instances. Optimized network utilization.
7. **Native Java support** - The Data-Grid engine is 100% Java based. This allows Java code to run natively collocated with the data sharing the same memory address.

# Kolona On-Time System



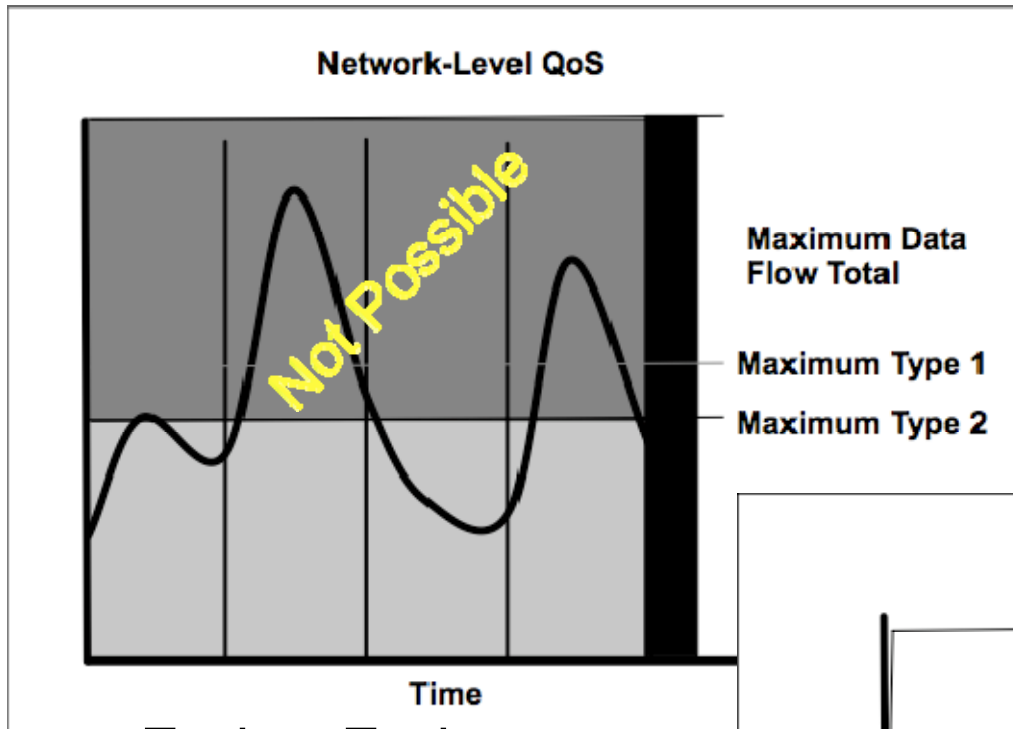
# Kolona On-Time System

quality assurance\*  
hot failover  
linear scalability  
secure  
modular  
very high integrity

\* On-time scheduling and dispatch to 10 microseconds ( $\mu$ s)  
GC. Cross-domain solutions

Application	Application	Business Logic
Kolona OverDrive	Application	Systemic Audit, Monitor, Manage
Kolona On-Time	Application	On-Time Scheduling and Dispatch
ESB	Integration	Pluggable End Points
		SEDA Architecture Integration
UM - OSGI	Management	Middle Management
Space-Based Architecture	Transport	Messaging
Kolona AUM	Automation	Administration
Kolona Kernel	Kernel	Spring Local Services (AOP, Security)
		Spring (Inversion of Control)
		MetrX
		Start Up
Java Real-Time	Libraries	Real-Time
OS (MILS enabled)	OS	OS
Server	Computer	Server

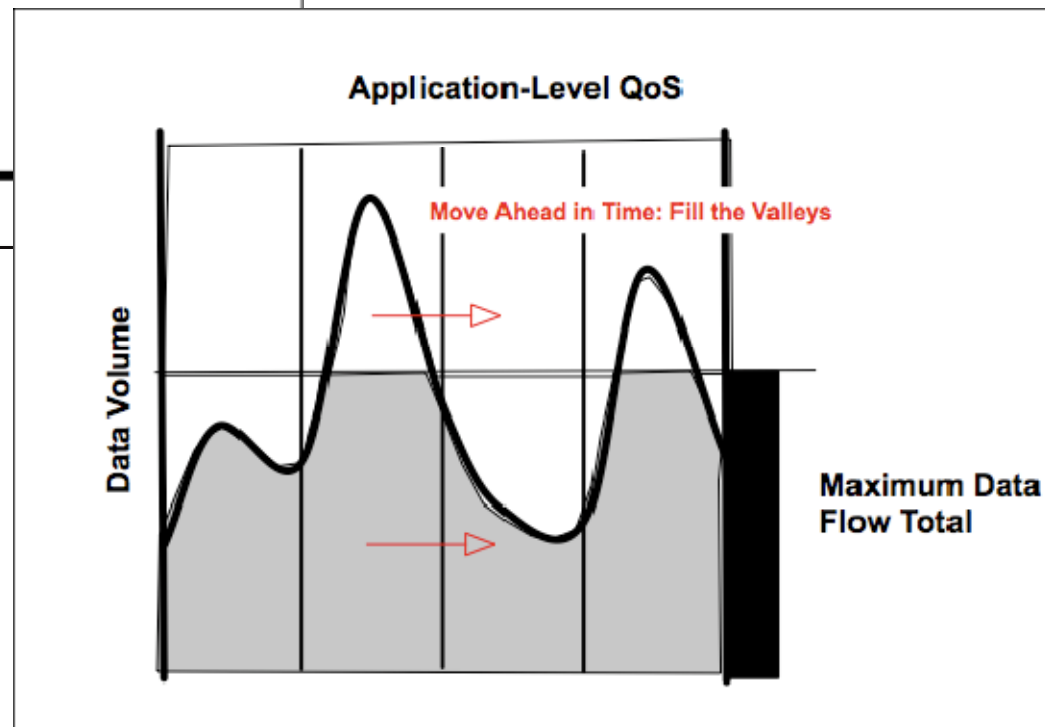
# Example: QoS



End-to-End  
Deterministic

## Multi-Dimensional System

Dynamic  
Deterministic



The End

# DDS and TS

## Interoperability – Scalability

**DDS:** In-memory data storage for message state.

**TS:** in-memory data storage for application state.

**DDS:** Cannot replace database.

**TS:** Can replace database.

**DDS:** Cannot act as data grid.

**TS:** Can act as data grid.

**DDS:** Problems are scalability and high-availability.

**TS:** Strengths are scalability and high-availability.

**TS:** The data-grid and services deployed into it can stretch and move themselves across different machines.

**TS:** Data-grid can colocate with relevant business logic and assure affinity (cohesion). Far faster than any messaging, including RTI.