Scaling Out Tier Based Applications

Nati Shalom

CTO

GigaSpaces
www.gigaspaces.com

TS-1595

Objectives

- Learn how to transform existing tier-based applications into dynamically scalable services using Space Based Architecture (SBA):
  - Distributed caching
  - Parallel processing
  - Virtualization
  - Dynamic provisioning

Agenda

- Nati Shalom: CTO, GigaSpaces
  - Turning tiers into scalable services using SBA
  - www.gigaspaces.com
- John Davies: CTO, C24
  - Using SBA to deliver scalable trade execution engine
  - www.c24.biz
- Frank Greco: Chair, NYJavaSIG
  - Patterns and use-cases for using SBA to achieve scalability
  - www.javasig.com

Before We Begin

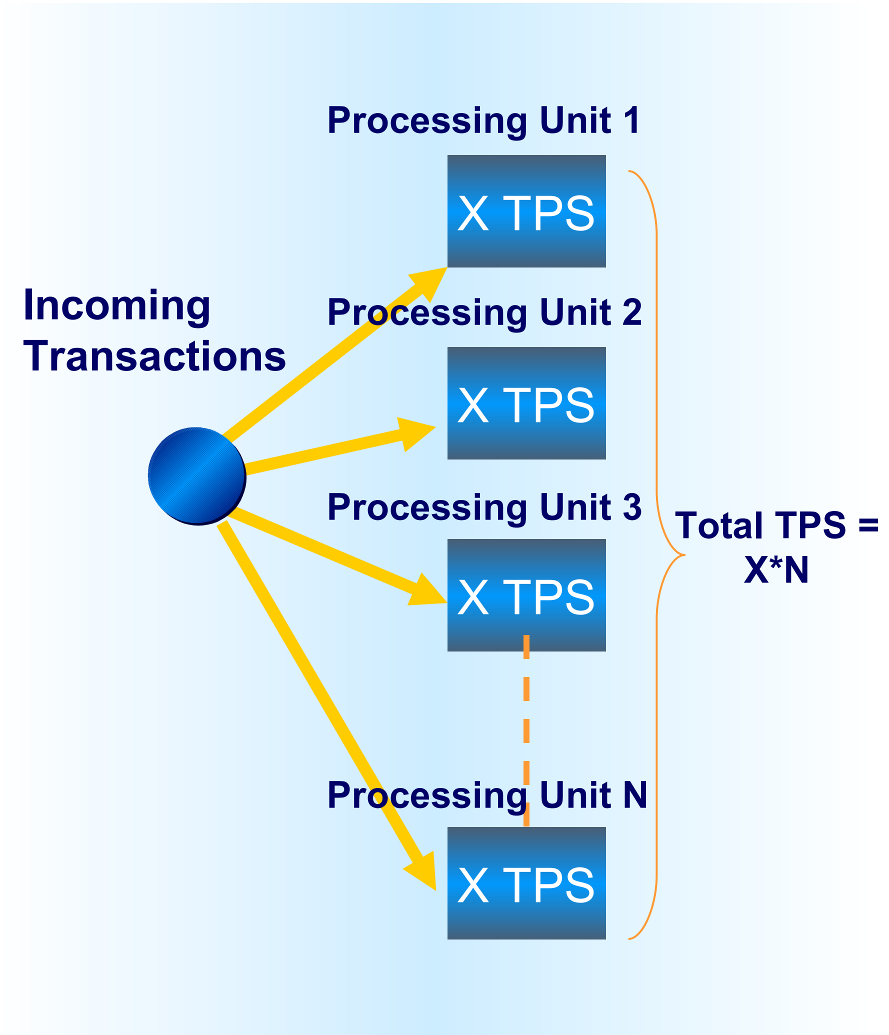
"Scalability for Dummies"

- Scale-out
  - Scale by adding more (duplicates) application processing units (services) on a dynamic pool of machines
- Scale-up
  - Scale by adding more processing power (CPU's) to a single machine
- Linear scalability
  - The overall throughput = number of processing units * throughput per unit
- Dynamic scalability
  - Scale on demand (usually using some sort of provisioning and monitoring capabilities)

The Business Motivation

- Process increasing volume of information faster and at a lower cost
- Why?
  - Financial applications
    - Electronic trading—generate more volume
    - New regulation rules—requires real time decision making
    - Low latency = who wins the deal first
  - Telco
    - Billing—pre-paid services requires real time decision making
    - Voip/3G—increasing volume of information
  - Others
    - RFID

The Ideal Scenario— "Write Once Scale Anywhere"



- Process increasing volume of information
  - Scale out to get more processing power when volume increases
- Shorter time
  - Through caching
  - Through parallelizing of transactions
- Lower cost
  - Pooling low commodity resources
  - Better utilization

The Reality

- To ensure reliability all states is stored in a centralized DB
- To reduce latency business logic is often written as stored procedures.

This leads to I/O bound applications that are limited to scaling up model

M
or
pr
the backend centralized

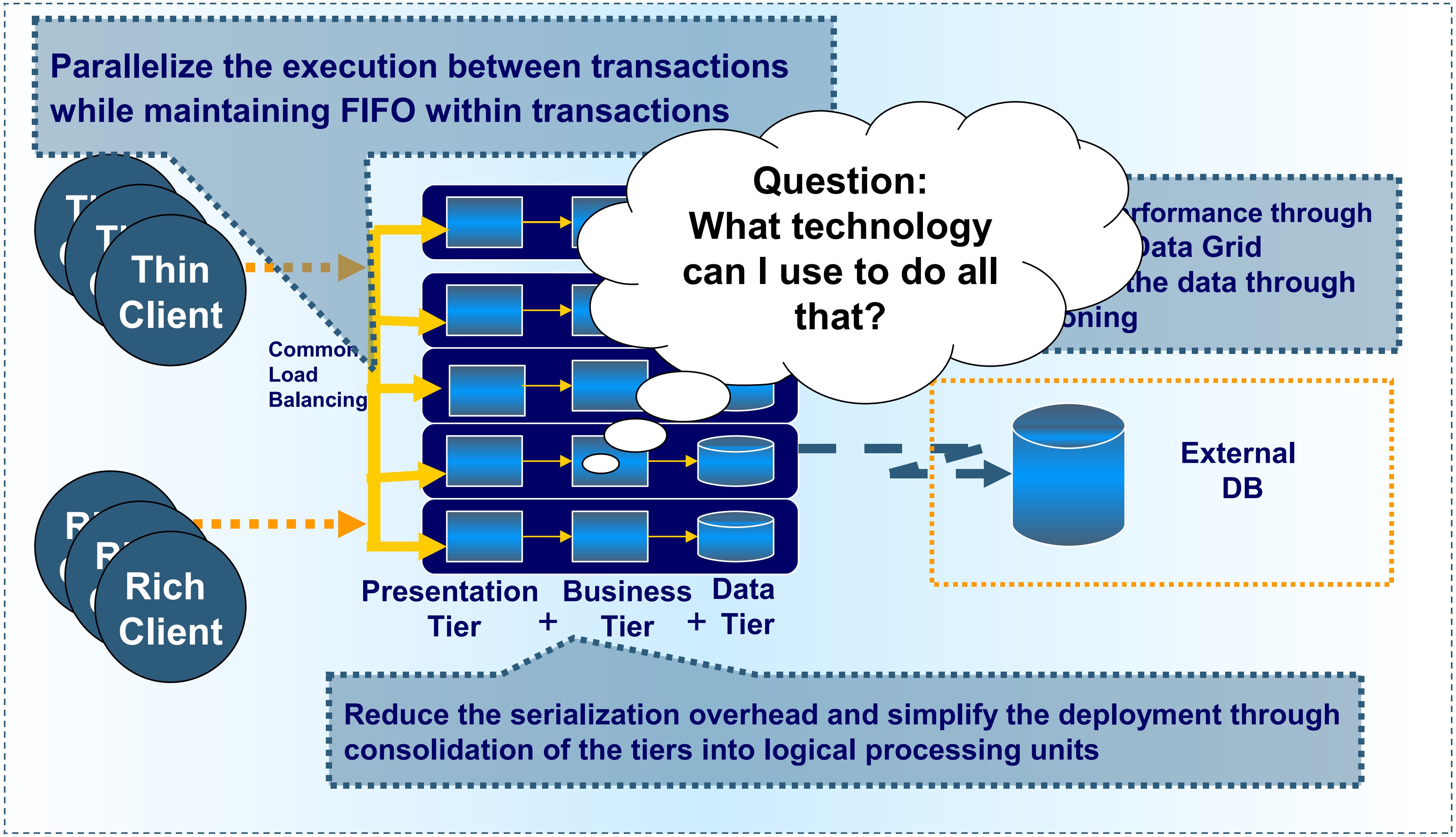
The Middleware is the Bottleneck

True scalability could not be achieved without solving the middleware bottleneck on both data and processing (Business Logic) layers

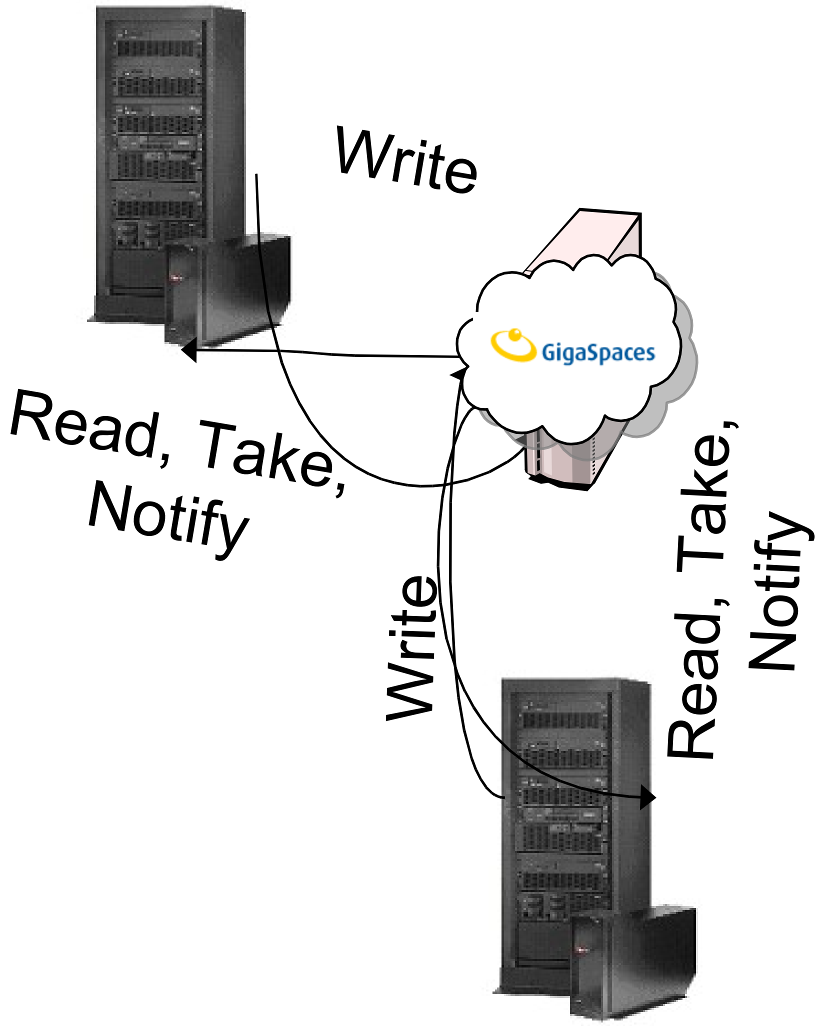
The Challenges—Scaling with Today Tier Based Approach

- Parallelizing centralized architecture
- How to scale when there are too many (independent) moving parts
- Consistency: each tier maintains its own HA and consistency model; scaling out of several tiers together while maintaining coherency of the system becomes pretty complex
- Serialization overhead: communication between sub components create significant performance overhead
- Scaling up vs. scaling out: currently enforces different architecture implementation per model; how to create a model that will enable a combination of the scale-up and scale-out models without changing the code
- Dynamic scalability: requires support from the application—to enable external life cycle management; export matrix that will trigger up scaling, down scaling events

Turning the Tiers into Scalable Virtual Services


Parallelize the execution between transactions while maintaining FIFO within transactions
Thin Client
Rich Client
Common Load Balancing
Question: What technology can I use to do all that?
rformance through
Data Grid
the data through
oning
External DB
Presentation Tier
+
Business Tier
+
Data Tier
Reduce the serialization overhead and simplify the deployment through consolidation of the tiers into logical processing units

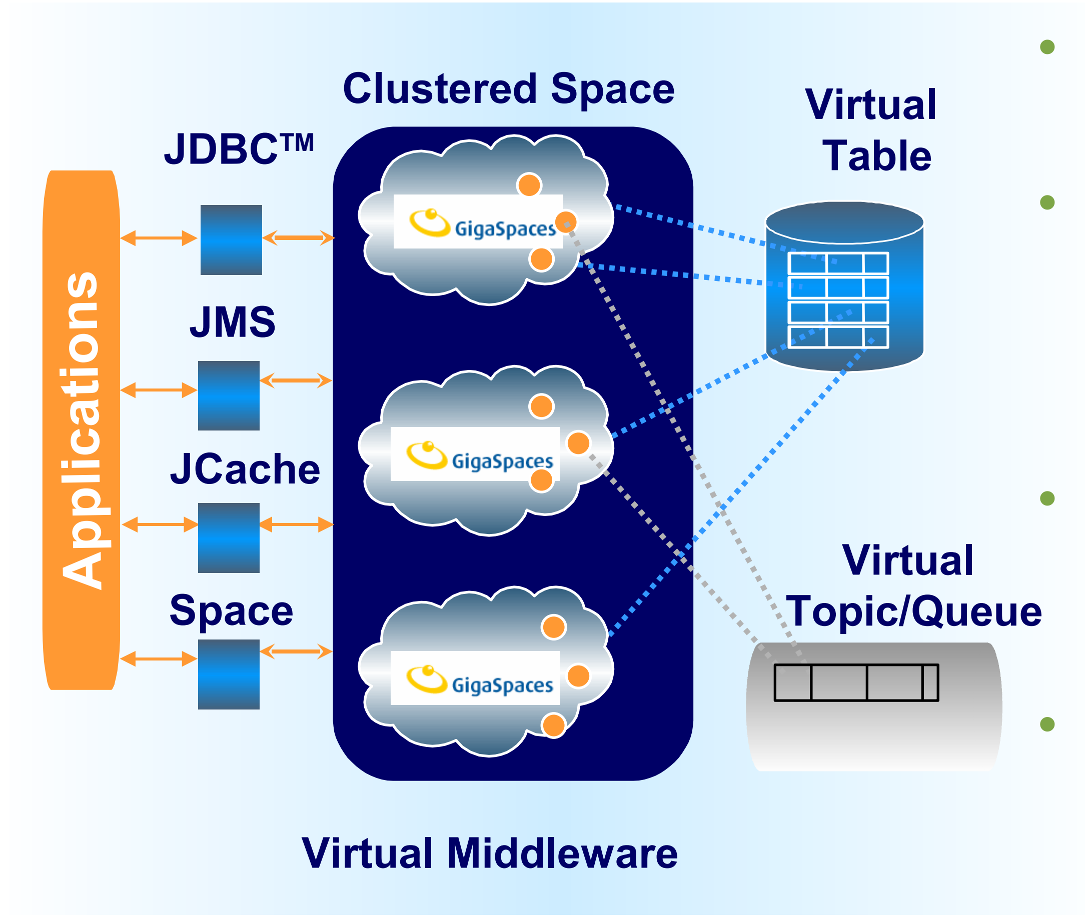
Space Based Architecture (SBA)



- Write: writes a data object
- Read: reads a copy of a data object
- Take: reads a data object and deletes it
- Notify: generates an event on data updates
- Providing virtualization middleware for Distributed Services providing:
  - Data caching
  - Load balancing
  - Messaging—1/1, */*, workflow, CBR
  - Parallel processing
- Using one common model, technology and runtime environment

Space Based Middleware Virtualization

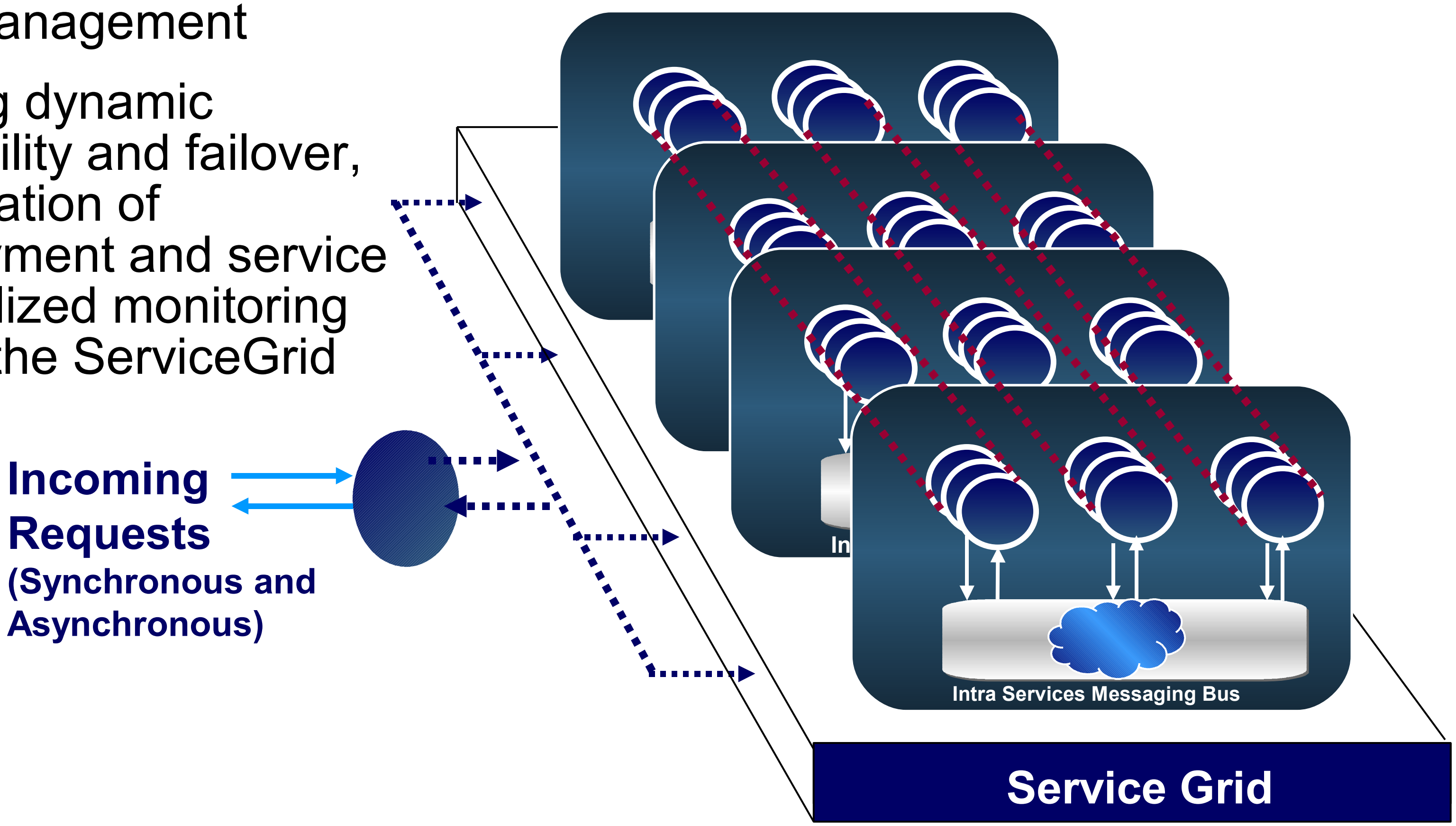
Single Virtualization Technology for Data-Tier and Processing Tier



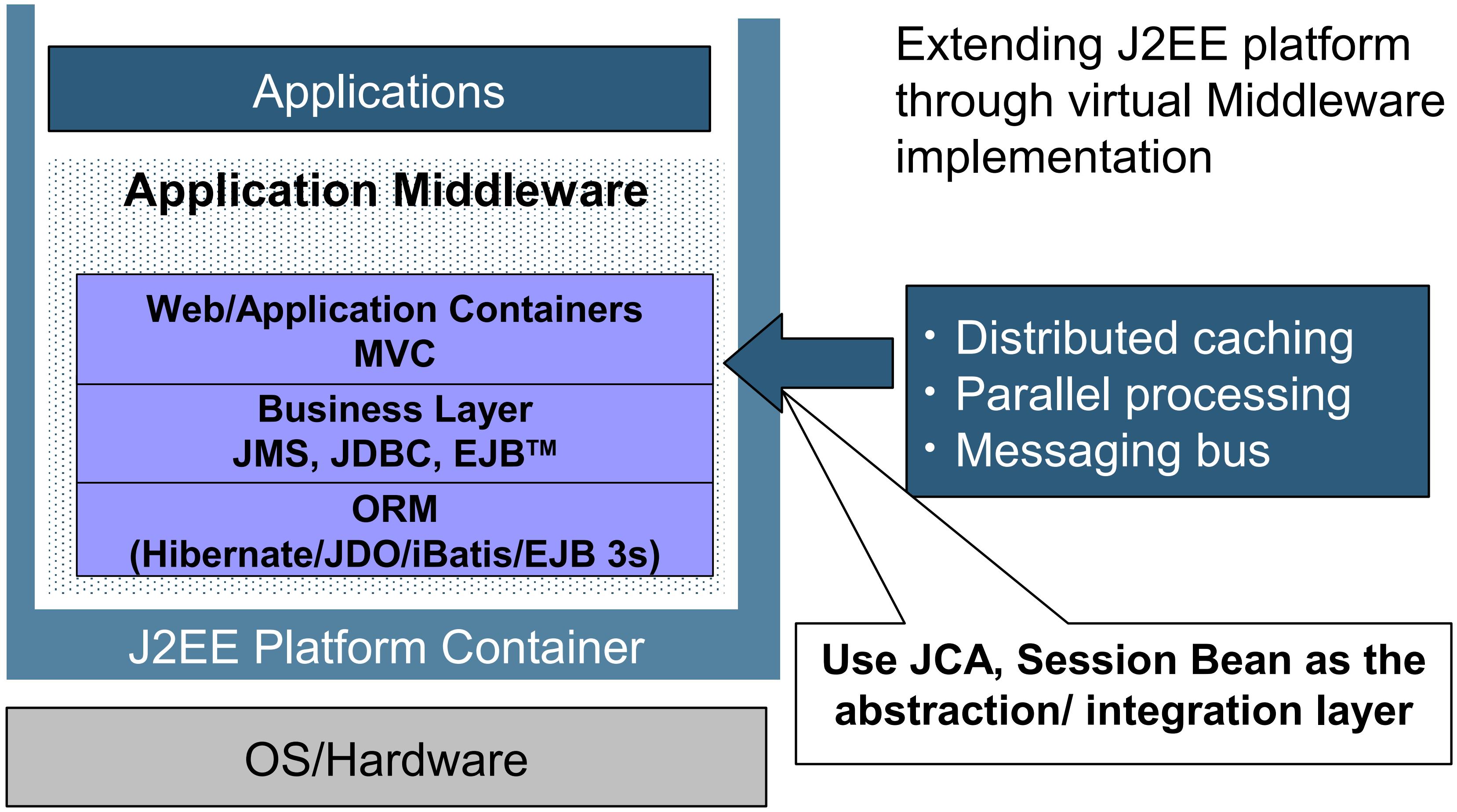
- Same data can be viewed through different interfaces!
- A single runtime for maintaining scalability, redundancy across all systems
- Reduces both the maintenance overhead and development complexity
- Provides Grid capabilities to existing applications

Dynamic Scalability Through Virtualization of the Container

- SLA driven deployment and management
- Adding dynamic scalability and failover, automation of deployment and service centralized monitoring using the ServiceGrid



Can this Work with Existing J2EE™ Platform-based Apps?

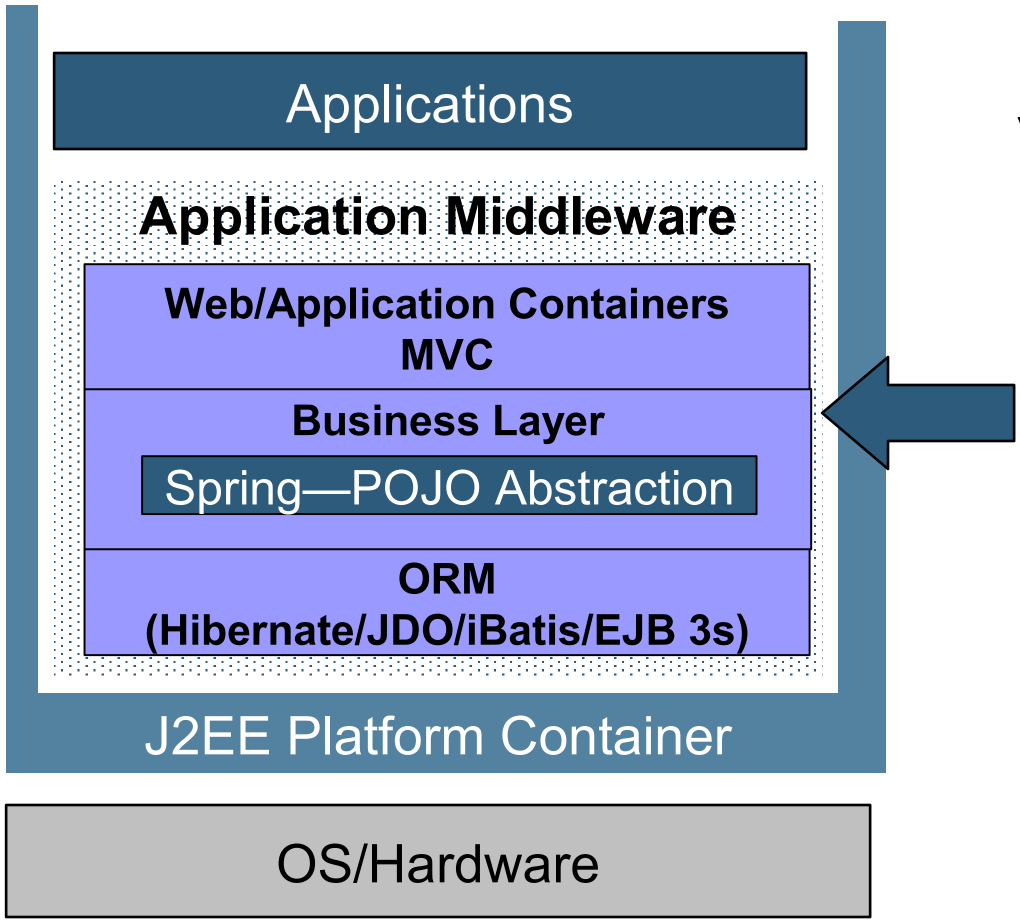


Applications
Application Middleware
Web/Application Containers
MVC
Business Layer
JMS, JDBC, EJB™
ORM
(Hibernate/JDO/iBatis/EJB 3s)
J2EE Platform Container
OS/Hardware
Use JCA, Session Bean as the abstraction/ integration layer

Extending J2EE platform through virtual Middleware implementation

- Distributed caching
- Parallel processing
- Messaging bus

Spring Makes it Seamless Transition



Using Spring to slide-in virtual middleware while keeping your POJO unaware of that

- Distributed caching
- Parallel processing
- Messaging bus

Using SBA to Deliver Scalable Trade Execution Engine

John Davies
CTO

C24 Solutions

www.c24.biz

TS-1595

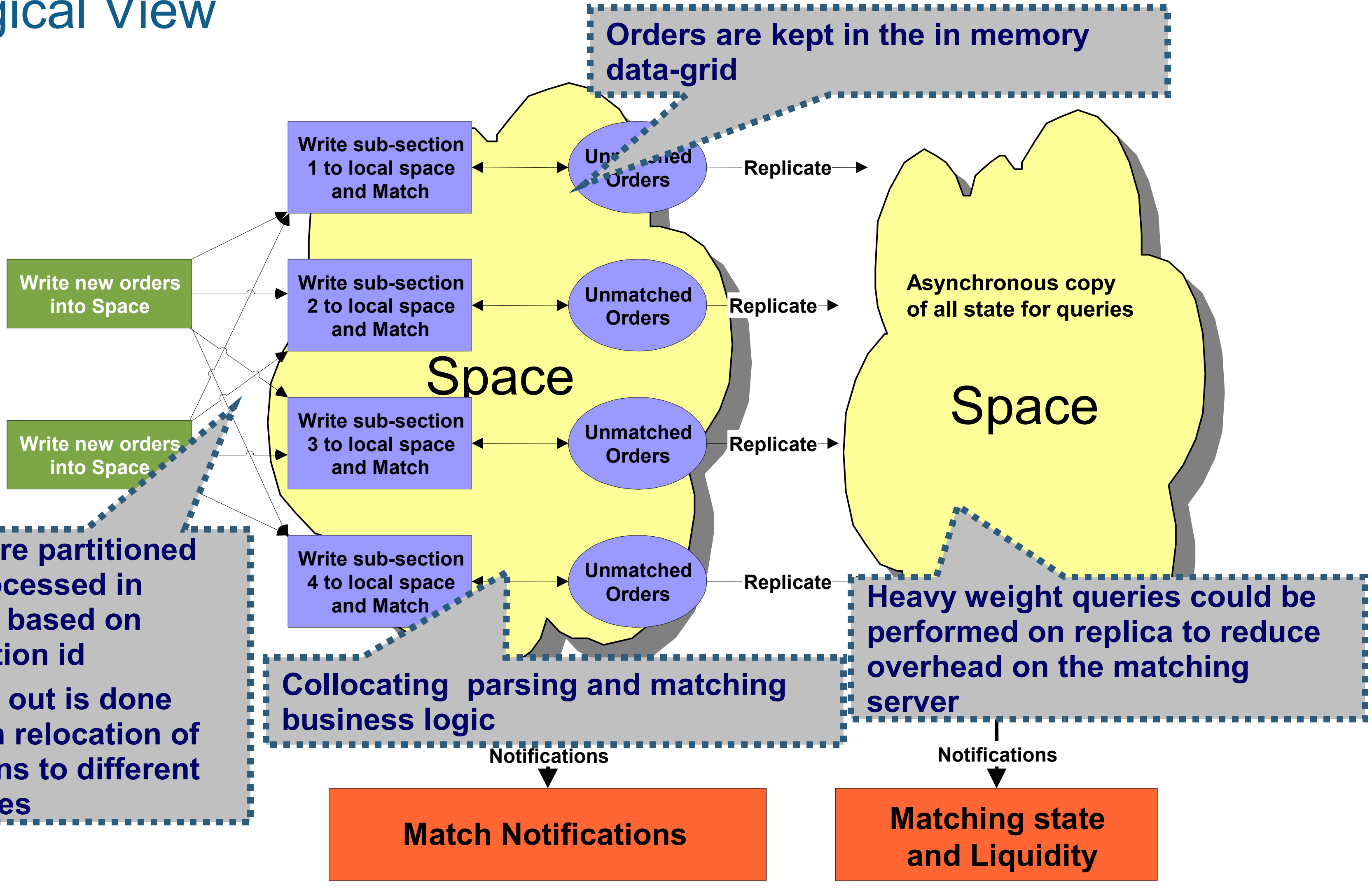
Pre-Trade Execution

Typically ECNs and OMSs

- >5 million orders per day
  - Typically Foreign Exchange (FX) and Equities
- >1000 orders per second
- Confirmation notification <5ms
- Near real-time liquidity monitoring (<100ms)
- Highly available and resilient
- Dynamically scalable
  - Without taking the system down

Pre-Trade Execution Architecture

Logical View



Orders are kept in the in memory data-grid
Write new orders into Space
Write new orders into Space
Write sub-section 1 to local space and Match
Write sub-section 2 to local space and Match
Write sub-section 3 to local space and Match
Write sub-section 4 to local space and Match
Unmatched Orders
Unmatched Orders
Unmatched Orders
Unmatched Orders
Replicate
Replicate
Replicate
Replicate
Space
Asynchronous copy of all state for queries
Space
Order are partitioned and processed in parallel based on correlation id
Scaling out is done through relocation of partitions to different machines
Collocating parsing and matching business logic
Heavy weight queries could be performed on replica to reduce overhead on the matching server
Notifications
Notifications
Match Notifications
Matching state and Liquidity

Java™ Technology Binding Gives More Flexibility

- Many message standards are non-XML based
  - e.g. SWIFT, FIX, etc.
  - FpML is a rare exception but the validation rules extends beyond XML Schema
- Binding messages to Java technology provides better integration, performance and flexibility
- Most SOA/ESB vendors are XML centric, this is fine outside of the grid/bus but not internally
- IONA and C24 provide high-performance object binding for better SOA performance and integration

Post-Trade Matching Engine

For Post-Trade Matching

- >1 million trade pairs per day
- >1000 trades per second
- Match confirmation <5ms
- Near real-time liquidity monitoring (<100ms)
- Highly available and resilient
- Dynamically scalable
  - Without taking the system down

JavaSpaces™ Technology Provides a More Flexible Grid/Bus

3-Tier Has Had Its Day

- Slide 1 and 4 are the same, the architecture is flexible
- Better still the same grid can be used for both solutions—simultaneously
- JavaSpaces technology + Java technology binding are quick to develop
  - A lot of the code is generated
- JavaSpaces technology is simpler to understand

  than Servlets

SBA Works for ESB and SOA

- Try to think out of the box—SOA is not new
- We can still expose services without XML
  - Just bind them to an Object
  - XML can still be used but don’t mandate it
- A true SOA or ESB must handle non-XML services
  - If it communicates internally without XML it’s going to be more flexible
- A Space Based Architecture (SBA) provides the flexibility needed for true SOA and ESB

No More Tiers— Space-Based Computing Use Cases

Frank Greco

Chair

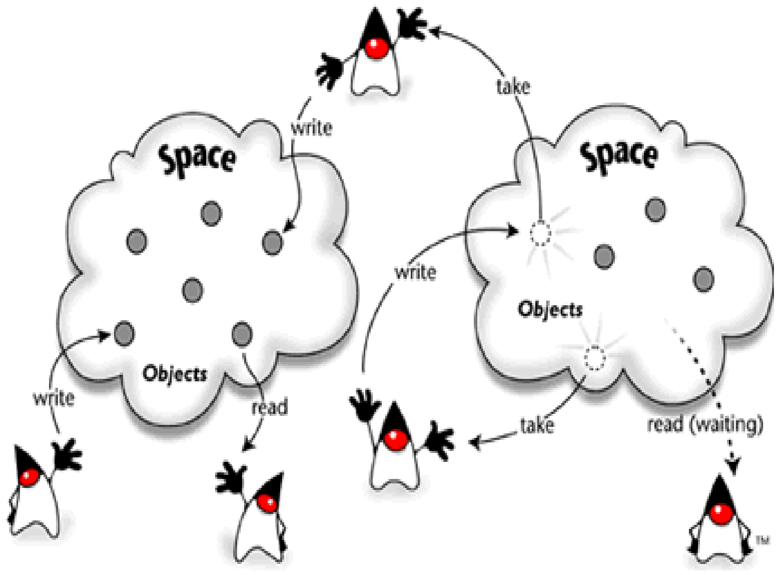
NYJavaSIG, NY Java User Group
www.javasig.com

TS-1595

No More Tiers—Use Cases for JavaSpaces Technology (Some Ideas)

More than Just a Data Cache or a Generic Dispatcher

- Middleware Replacement
- Distributed Grid Computing
- JMX™ API Repository
- Grid in a Box
- Enterprise Service Bus
- SOA Mesh

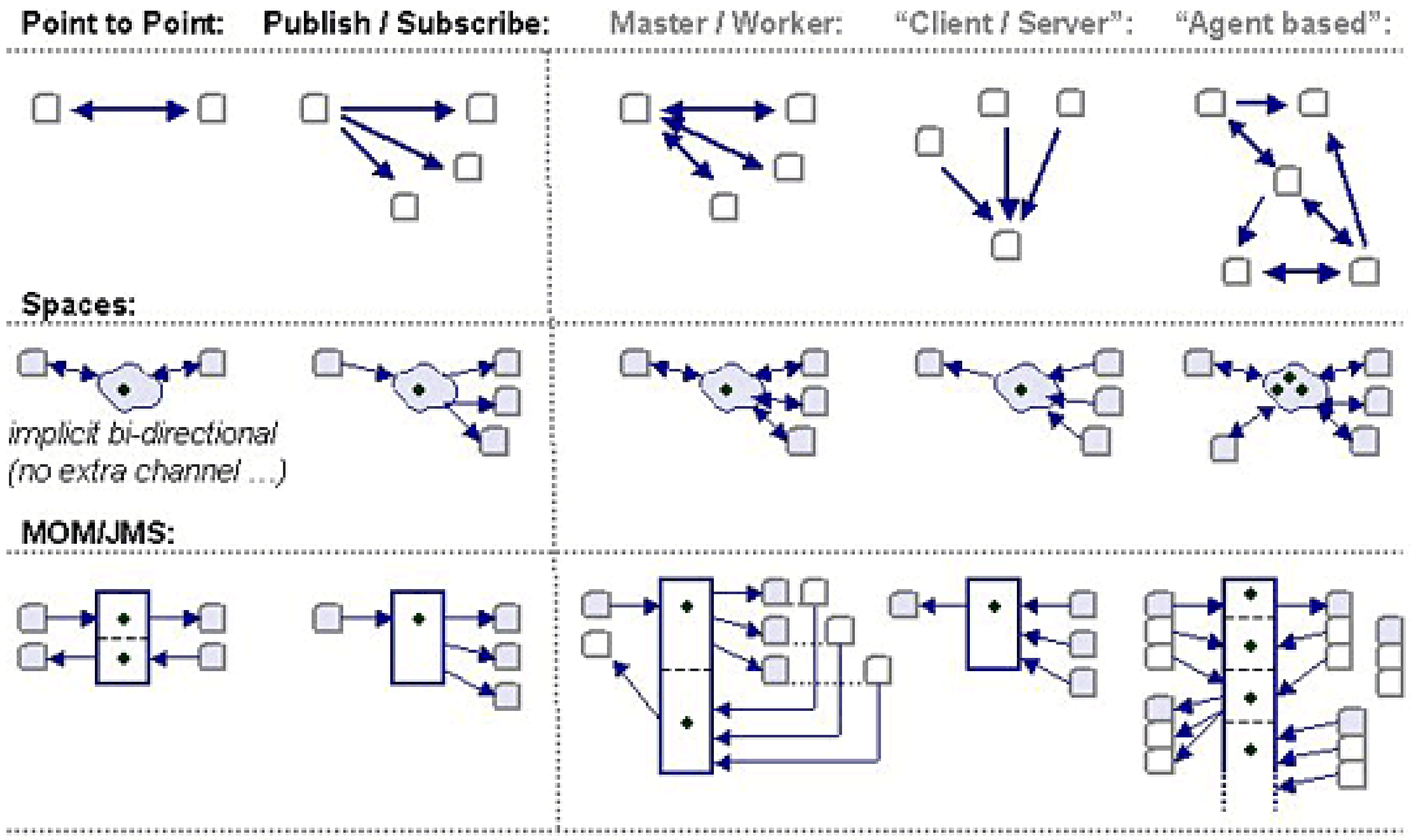


Middleware Replacement

- Improvement over RPC
  - Loosely coupled
  - Built-in reliability
  - More scalable
  - More SOA-friendly
  - Pull model—naturally load-balancing
- Improvement over CORBA/SOAP
  - Implicit retries
  - Higher performance
  - More scalable
  - Can easily work with SOAP and other WS

Middleware Replacement

- Improvement over Publish/Subscribe (Java Message Service, Tibco Rendezvous)
  - Peer-to-peer vs. hub-and-spoke
  - No single point of failure
  - Not ‘Fire and Forget’ (lighter-weight guaranteed delivery)
  - Formal specification for object notification
  - Distributed cache “side affect” benefits state
  - Scale up is straightforward
- JavaSpaces technology is Functional Superset

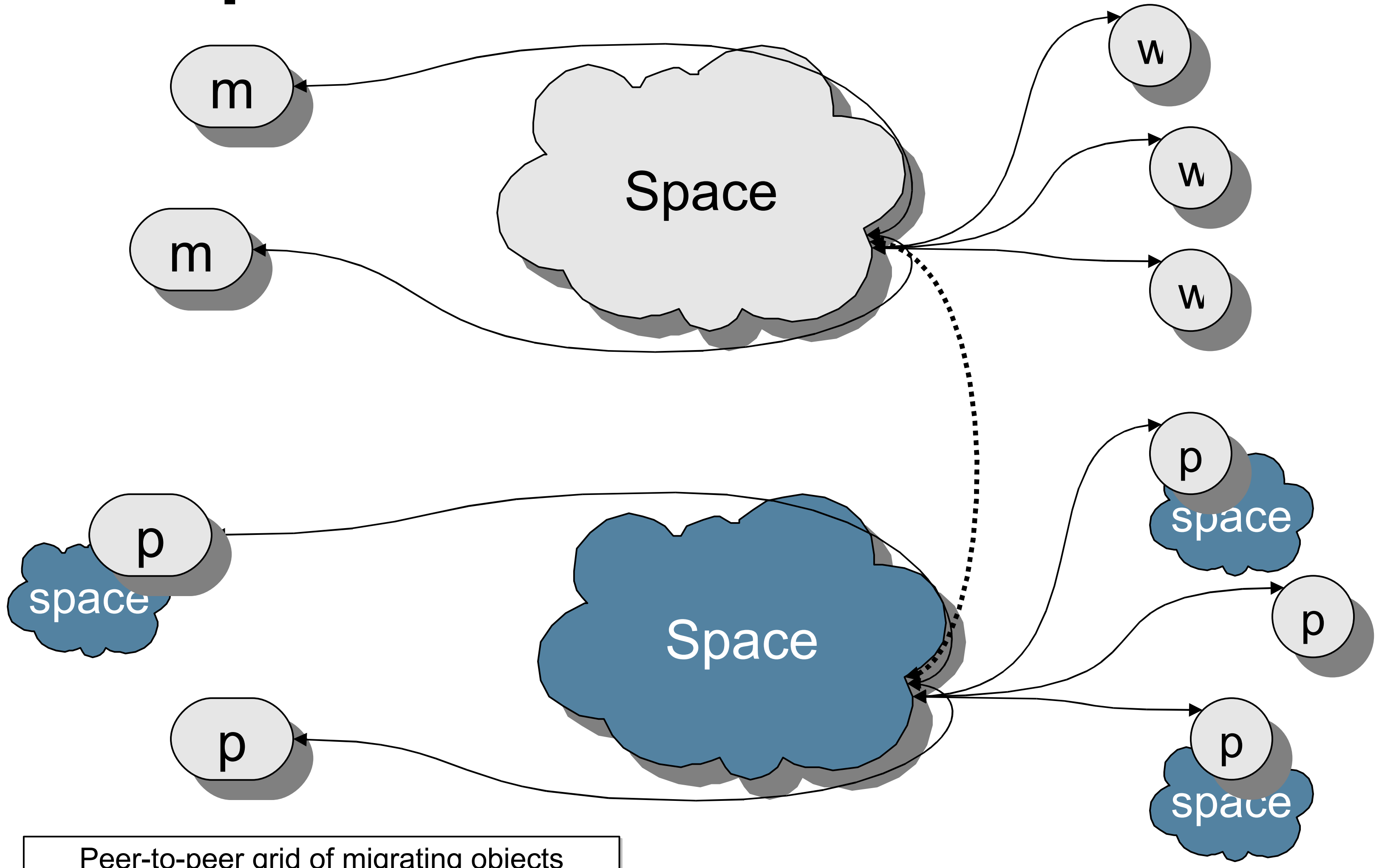


[1]Loosely Coupled Communication and Coordination in Next-Generation Java Middleware
http://today.java.net/pub/a/today/2005/06/03/loose.html

Distributed Grid Computing

- Classical Master/Workers grid model
- Transactional
- Workers can have dynamic behaviors
  - Hey… they're objects
- Spring allows POJOs to be grid-enabled
  - From Enterprise JavaBeans™ (EJB) architecture to grid by configuration
- Conventional compute grid or data grid
- Scaling the grid by merely adding more workers or more Space servers

Compute Grid or Data Grid or Both

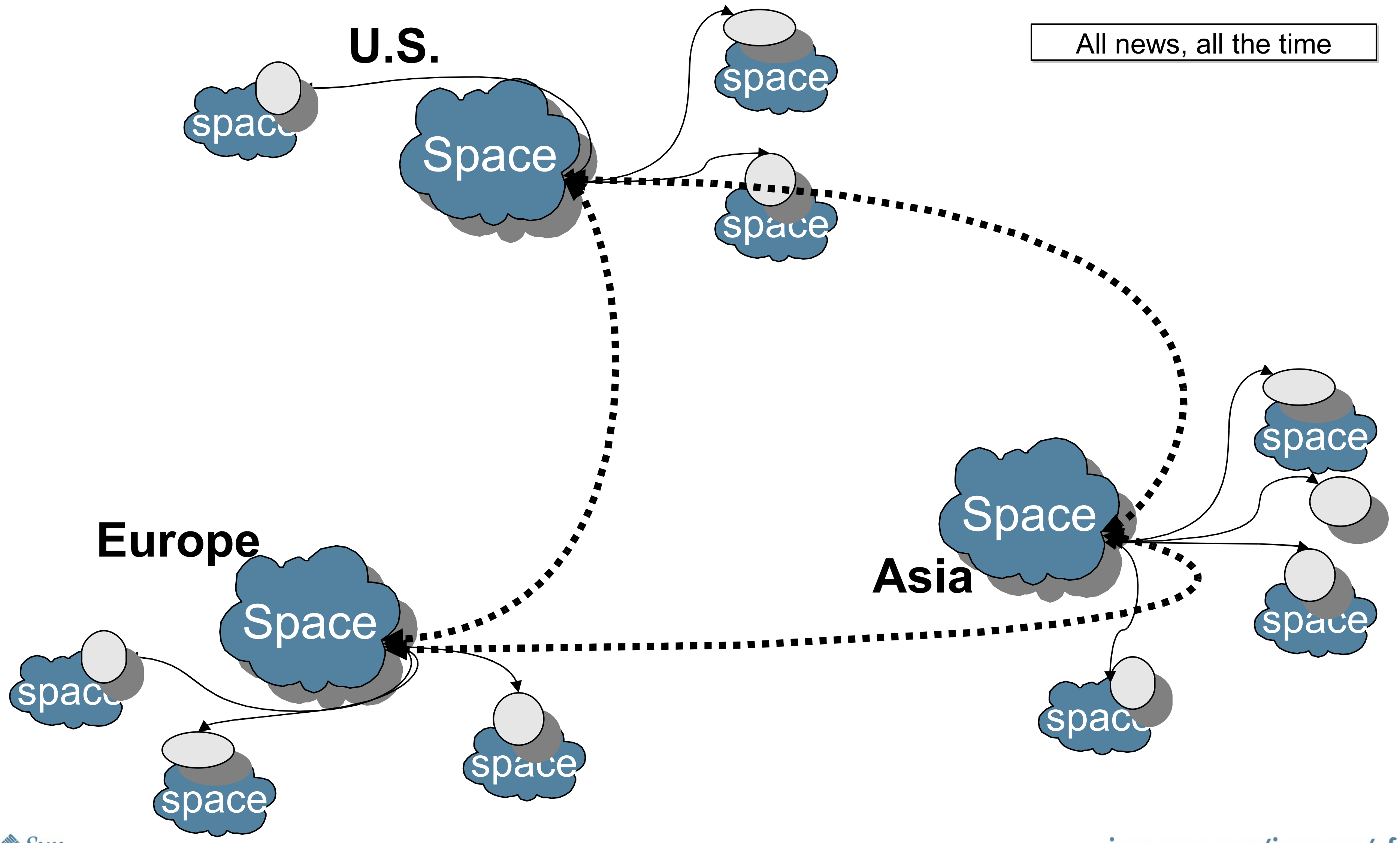


Peer-to-peer grid of migrating objects

JMX API Repository

- JMX API: Java Management eXtensions
- Match JMX Technology Events with Spaces reliable delivery
- JMX Technology Events saved via Spaces-based repository
  - Can be distributed geographically
  - Replicated for robustness and performance
  - JMX API Containers with Spaces
- To scale, just add more Space servers for increased monitoring loads

Geographic JMX API Monitorability



Grid in a Box

- Both OS and CPU are increasingly more parallel with threads and multi-cores
- Blades on high-speed, low-latency bus
  - Grid in a box
  - Blades are grid nodes with fast shared memory
- Sun's Niagara/Project Rock, Azul Systems
  - Grid on a chip
  - Need an improved programming model—Spaces! [New heavily-threaded languages would help too]
- Spaces model fits!

Enterprise Service Bus

- Services and queues
- Loosely-coupled architectures with API’s (e.g., SOAP, JMS, RV)
- Spaces enhances reliability/robustness
- Extension of grid with state (data grid)
- Spaces can accommodate non-Java language (C++, C#), SOAP, .NET, et al.
- Gigaspaces Enterprise Application Grid
- mule.codehaus.org—ESB with Spaces

SOA Mesh/Fabric

- Extension of ESB, borrowed from Telecom
- Services grid—more ‘A’ in “SOA”
- Services dynamically join federation
- Multiple federations in the fabric
  - e.g., Can run “prod”, “staging”, “testing”, “dev”, “DR” in the same fabric
- Scale up by adding machine/services
  - Services announce themselves
  - State always available

Summary

- Current tier based implementation cannot meet the business requirement

Simple!—"Write Once Scale Anywhere"

Try it out for free—www.GigaSpaces.com

- GigaSpaces is a pioneer in that field—providing a complete solution for dynamic scalability based on SBA in an evolutionary path

Q&A

Nati Shalom

John Davies

Frank Greco

Scaling Out Tier Based Applications

Nati Shalom

CTO

GigaSpaces
www.gigaspaces.com

TS-1595