ORACLE®

Coherence Training
Introduction to Coherence

All Content
Proprietary and Confidential

Introductions

Course Structure

Course Structure

- This is a first for Coherence
  - 1st Three Day Course
  - 1st Field / Sales / Consultant Oriented Course
- Strategy
  - Introduction to Coherence
  - Progressively getting more technical and hands-on
- Objective
  - Explain and Sell Coherence
  - Work on Proof-of-Concepts
- Warning
  - “Coherence is technical”

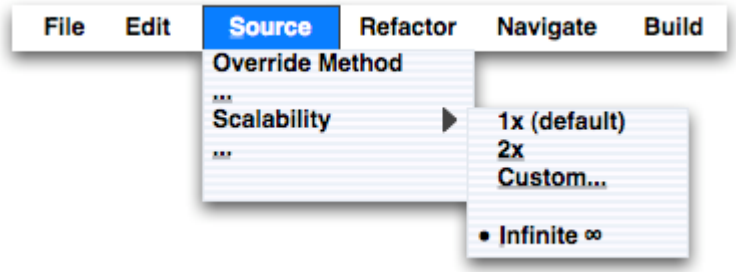
Topics for Today

- Why Coherence?
- What is Coherence?
- Coherence in the Application-Tier
- Customer Stories
- Coherence Demonstration
- How Coherence Works
- Grids and Data Grids
- How much effort?
- Coherence and other Oracle Products
- Solutions Architecture Directions
- Identifying Opportunities
- Competitive Analysis
- Proof of Concepts
- Future Directions

Why Coherence?

Application Scalability

- Scaling the Application-Tier is difficult
- If it was easy it would be an IDE option



Not possible!

- Scalability is a design option
  - Requires knowledge, care and experience
  - Developers have the “option” to consider building it in!
  - It’s not an IDE option
- Coherence is scalability infrastructure for the application-tier

A Scalability Refresher

What is Scalability?

- Scalability:

  "The degree to which the performance of a system improves when more resources are added"

- Linear Scalability:

  "When resources are increased by a factor of n, system performance improves by the factor of n"

- Predictable Scalability:

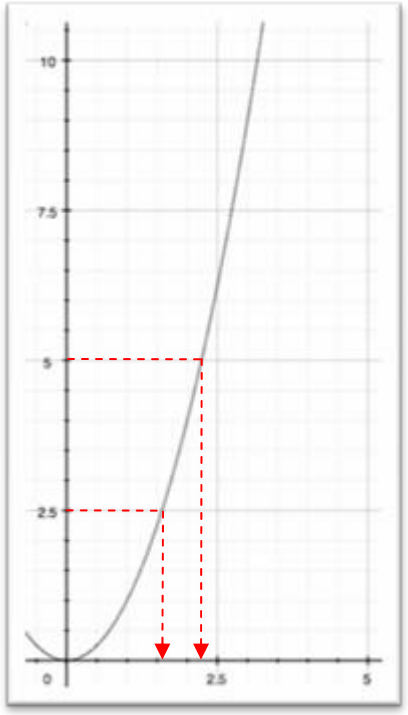
  "The ability to know in advance of adding resources the degree to which a system will scale"

Scalability Approaches

Approach	How	Advantages	Disadvantages
Vertical “scaling-up”	Increase resources in existing server(s)	❖ Relatively simple process (can be achieved overnight) ❖ Transparent to system architecture and development	❖ Comparatively expensive hardware (niche) ❖ Limited Scalability (physical limits typically encountered) ❖Increases cost of failure
Horizontal “scaling-out”	Add more servers	❖ Comparatively inexpensive hardware (commodity) ❖ Virtually unlimited scalability possible (typically greater than scale-up approach)	❖ Applicable only when a system is designed to “scale-out” ❖ May require months of rework to achieve ❖ Scalability may be limited by “network” ❖ Requires additional administration

The Scale-up Challenge

- Some systems just don't "scale well", regardless of hardware
- EG: $O(n^2)$ algorithms
  - *That is: time = data-size* ^{2}
  - *In 5 seconds, n ≈ 2.24*
  - *In 2.5 seconds (2x scale-up / twice as fast), n ≈ 1.58 (not 1.12 as you'd expect)*
  - *As n → ∞, scale-up return diminishes dramatically*



Developers and Scalability

Be aware!

- Poorly designed algorithms and data structures may not scale
- Scalability is often a non-functional requirement
- Scalability is often “left to last” and not “designed in up-front”
- Developers tend to assume that their system is scalable
- Developers are often surprised that their system is not scaling
- Developers tend to assume there is a quick fix for scaling
- Developers may assume Coherence is a drop in solution
- Coherence may not be a solution (often it is… more later)
- While a system may be scalable, often operational costs are not taken into account (it’s someone else’s problem)

Scalability and Performance

- Scalability is like a Locomotive
  - Designed to handle load and capacity
  - Add more cars and engines (scale out)

- Performance is like a Fast Car
  - Designed for speed (not capacity)
  - Improve engine and components (scale up)

- You can't just add them together! They have to be designed.

What do we mean by “Scalable”?

- High scale
  - Scales readily to ~100 servers
  - Practical limit of ~1000 servers
  - Support for thousands of simultaneous clients
  - Multiple Sites
  - Across continents & globe
- Easy scale
  - Just plug in additional machines
  - While system is running
  - No need for manual application partitioning

What do we mean by “Performance”?

- Instant access
  - Clients can maintain coherent data in local memory
  - Faster than disk or even network
- Instant awareness
  - Clients can subscribe to real time events
  - Notification to application servers or even desktops
- Parallel data processing
  - Clients can push processing to the servers
  - No data movement results in very high performance

Further Reading

- http://en.wikipedia.org/wiki/Scalability
- http://en.wikipedia.org/wiki/Amdahl%27s_law
- http://en.wikipedia.org/wiki/Algorithmic_complexity
- http://en.wikipedia.org/wiki/Big_O_notation

Coherence Scalability

- Coherence: Designed to scale-out the Application-tier
  - Standard Java Applications (JSE, non-JEE, container-less)
  - Web Applications (session state)
  - Middle-tier Applications (JEE, container-based)

- Artifacts that can been scaled
  - Application and User State (objects)
  - Object Access (crud)
  - State Mutation Notifications (events)
  - Processing (updates, transactions)

Scaling the Application-Tier (without Coherence)

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Scale-Up *"It's an infrastructure problem"*	❖ Buy Big Boxes ❖ Increase Resources (cpu, memory, hdd capacity, speed and network, etc) ❖ By specialized hardware (Azul, Infiniband…)	❖ Simple (overnight) ❖ No development ❖ No impact on internal design	❖ Expensive ❖ Will hit physical limits ❖ Will have to redesign at limit ❖ Non-graceful deterioration at limit ❖ Stop, Add, Restart required to scale

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Stateless Scale-Out *"Push state scale-out into lower Data Source layer"* *"It's the DBA's problem"*	❖ Make application stateless (eg: stateless sessions) ❖ Use lots of stateless servers ❖ Use load-balancing ❖ Use "big" and "scalable" Data Source to ensure application state scale-out	❖ Easy to develop (not overnight, but relatively simple as no state is managed) ❖ Scale-out is easy, just add more servers	❖ Only scales to match underlying Data Source performance ❖ When underlying limit is reached, have to redesign ❖ Network bottlenecks experienced as data is moved between layers

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Caching *"Keep recent copies of state"* *"We'll save the DB and DBA by caching"*	❖ Application keeps local copies (in memory or on local disk) of recently / commonly used state	❖ Seems simple ❖ Reduces Data Source and Network load ❖ Significant application performance improvements	❖ Maintaining consistency of data between Local and Data Source instances can be difficult ❖ Require "messaging infrastructure" to ensure coherency across a cluster (and application development) ❖ Typically applicable to "read only" applications and not "write a lot" applications ❖ Easy to get wrong

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Use an Application Container *"Our magical clustered container will scale our application infinitely"*	❖ Believe the vendors & the marketing ❖ Follow a "scalability paradigm" ❖ Use a "Clustering Container" … It scaled the "Pet Store" linearly, therefore our X application will also scale linearly (where X ≠ "Pet Store)	❖ Simple ❖ Well documented and communicable paradigm ❖ Easily scale development team	❖ Typically scales in-the-small ❖ Usually relies on "scale-up" rather than "scale-out" ❖ Requires specialized skills or products (out side of the standard paradigm) to really scale ❖ Clustering is primarily about High-Availability, not Scalability!

Scaling the Application-tier (without Coherence)

Approach	How	Advantages	Disadvantages
Manually partition the Application and / or Data *"Scalability is easier in small bits"*	❖ Break the application domain into independently scalable components ❖ Have separate teams deal with their own components ❖ Use "pools" of Services to perform work ❖ Use load-balancing to scale-out	❖ Seems simple ❖ The problem isn't as big as it was before ❖ Some components may actually scale better by themselves	❖ Often difficult to decompose the application ❖ What's good for one component, is often bad for another (eg: if you need 'joins') ❖ Typically introduces new bottlenecks (sharing information between components) ❖ Managing an application composed of many independent parts is more complex!

Scaling the Application-tier (without Coherence)

- In summary…

"Solving application-tier scalability is either;

a). someone else's problem, or

b). involves the complex process of partitioning and managing data, services and coherency across a collection of servers."

- Coherence provides developer solutions for b) to enable predictable application scale-out

Why Scaling-out the Application-Tier is Hard!

Why Scaling-out the Application-Tier is Hard!

- Anyone can write network software these days…
  - Java, .NET, Ruby etc… all provide network abstractions to transfer data between applications on separate servers, even around the world
  - You can learn it from the Internet
- Anyone can write code to make software communicate with other bits of software

Why Scaling-out the Application-Tier is Hard!

- However…

"It's extremely difficult to write software that ensures an unpredictably (dynamically) growing collection of servers connected by an unreliable network can continuously work together without losing information (or work) in a manner that itself is linearly scalable"

- Significance…
  - Achieving all of these things in the same product
  - Working together means "consensus" has to be maintained!

Imagine a team where some members…

- Have a different impression of the actual members of the team
- Allocate tasks and information to their members (from their perspective) but on behalf of the team
- Result?
  - Inconsistent views of team information
  - Without consensus some information will be inconsistent (at best) or be unavailable or lost (at worst / common)

Membership Consensus

- Consensus between resources is fundamental to ensure integrity of information (and work) when scaling-out

- Consensus is not about roles, it's about membership
  - It's not what X is doing, but that X is in (or out of) the team.
  - What X "is doing" is "state" that may be shared amongst the known team

Membership Consensus

- Membership Consensus:

  "A common agreement between a set of processes as to the membership of the group at a point in time"

- Without Consensus…
  - Applications can't determine their reliably work together (like a team!)
  - Partitioning of Data or Services can't reliably be performed or maintained
  - Data integrity and consistency can not be maintained across a collection of processes or servers

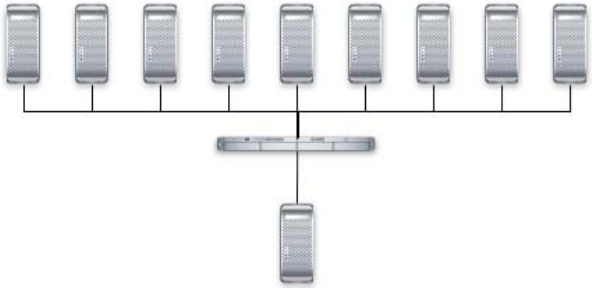
Membership Consensus

- Coherence has proprietary clustering technology that continuously guarantees consensus across a collection of applications
  - Essentially… all applications know of all other applications
- With Consensus…
  - Data and Services may be reliably partitioned across the known members
  - Data and Services may be backed-up (on other members)
  - Applications may be scaled-out while remaining stateful
  - Application state can be maintained consistently

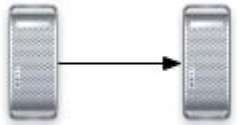
Traditional Scale-Out Approaches…

#1. Avoid the challenge of maintaining consensus

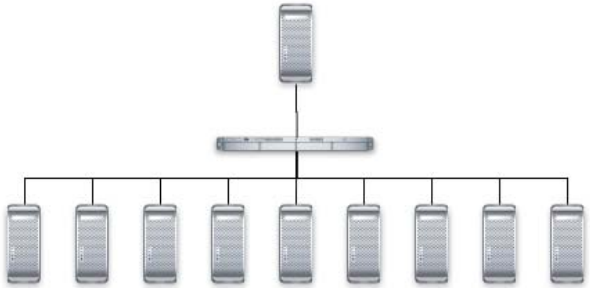
- Opt for the “single point of knowledge”



Client + Server Model
(Hub + Spoke)



Active + Passive
(High Availability)



Master + Worker Model
(Grid Agents)

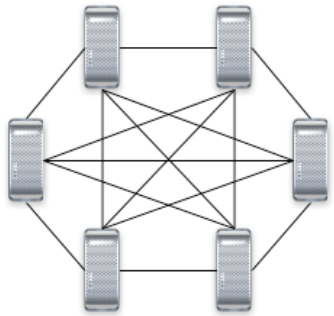
#2. Have crude consensus mechanisms, that typically fail and result in data integrity issues (including loss)

Traditional Scale-Out Consequences…

- Have unbalanced / unfair load and task management
  - Some servers have greater system responsibility than others
- Have Single Points of Bottleneck (SPoB)
- Have Single Points of Failure (SPoF)
  - “Micro outages” are magnified as you scale-out
- Exhibit Strong Coupling to Physical Resources
  - Software completely dependent on individual physical servers
- Require specialized deployment and operation for individual Resources
  - Some servers require “special attention” to operate

The Coherence Approach…

- Traditional scale-out approaches limit
  - Scalability, Availability, Reliability and Performance
- In Coherence…
  - Servers **share** responsibilities (health, services, data…)
  - No SPoB
  - No SPoF
  - Massively scalable by design
- Logically servers form a “mesh”
  - No Masters / Slaves etc.
  - Members work together as a team



The Coherence Approach…

- Consensus is key
  - Communication is more efficient (peer-to-peer)
  - No outages for voting (no need – everyone is a peer)
  - No SPoF, SPoB
  - No need for broadcast traffic (yelling at each other)
  - You can do many things once you have “consensus”.

What is Coherence?

What is Coherence?

- Coherence (deployment perspective)
  - Single Library*
  - Standard Java Archive “JAR” for Java
  - Standard Dynamically Linked Library “DLL” for .NET connectivity (.Net 1.1 and 2.0)
  - *Other libraries for integration (Databases, Spring…)
  - No 3rd party dependencies!
  - Minimal “invasion” on standard code*
  - Configurable implementations of standard Map / Dictionary interfaces (NamedCache)
  - Provides Predictable Scalable Caching
  - “RemoteException” free distributed computing

What is Coherence?

- Proprietary extensions provide powerful parallel processing capabilities
  - Query, Events, Transactions
- Use it in any Application-Tier layer

“The most expensive java.util.Map implementation in the World?”

What is Coherence?

- Coherence (architectural perspective)
  - Scale-out Applications State
  - Reliable Data Management / Data Abstraction Layer
  - Effortlessly Cluster Applications (clustering infrastructure)
  - Web (session management)
  - Front, Middle, Back Tiers
  - Thick Clients (AWT, Swing, Console, RCP…)
  - JSE or JEE
  - Remote Connectivity
  - Business Continuity and Disaster Recovery
  - Provide a Data Grid

What is Coherence?

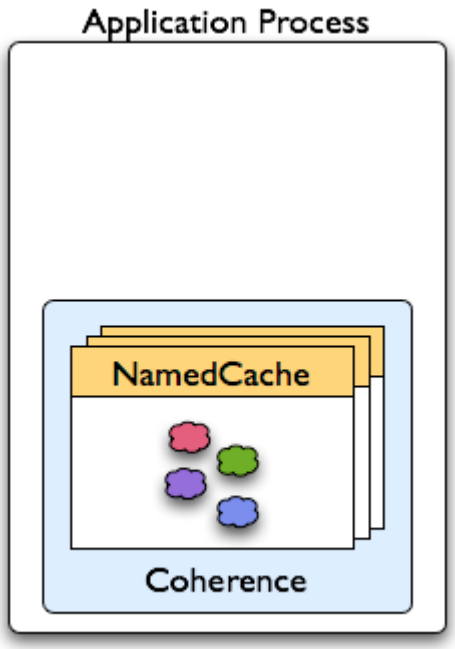
- Coherence is not
  - Messaging
  - Application Server
  - Database
- Coherence addresses gaps in existing solutions
  - Stronger data management than an Application Server
  - Better scale-out performance than an Application Server
  - Better scale-out performance than a Database Server
- This implies heavily technical differentiation and value
  - Coherence has enormous value
  - Communicating this value can be challenging

Scaling the Application-tier with Coherence

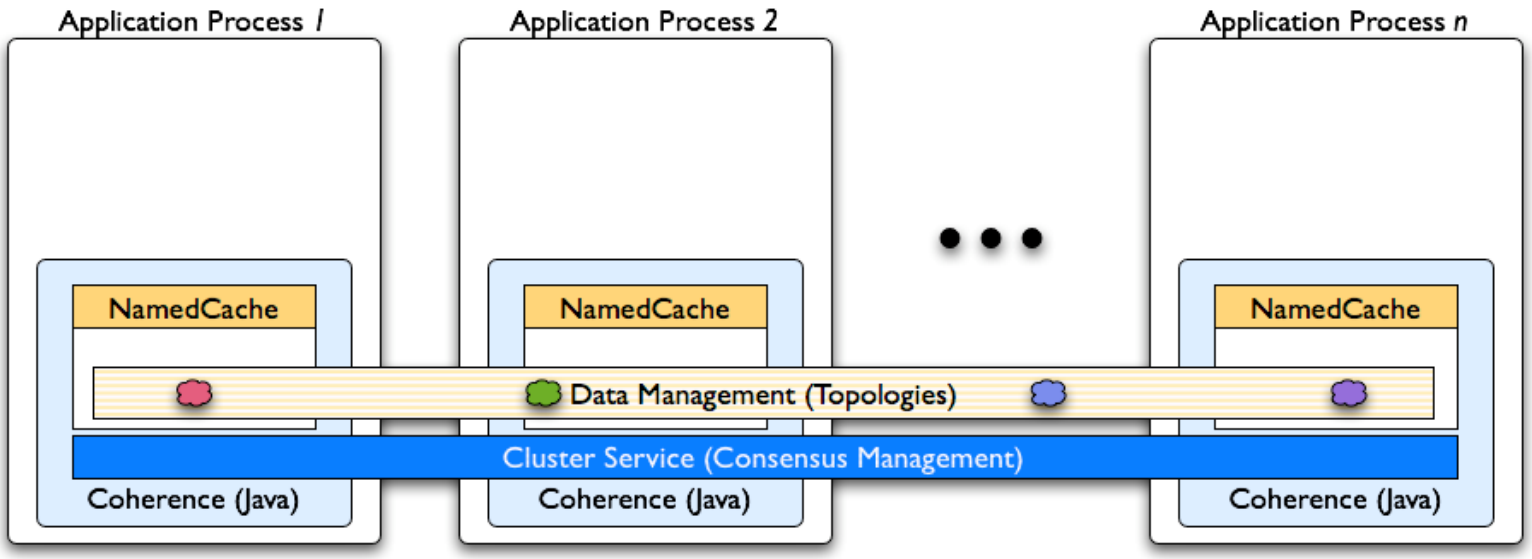
Approach	How	Advantages	Disadvantages
Use Coherence to share and manage objects (application state) *"Coherence is responsible for my objects"*	❖ Introduce Coherence libraries into Application(s) ❖ Use Coherence NamedCache API (derived from java.util.Map) to store application state ❖ Start multiple Coherence-enabled processes to scale-out (load balance) objects (data)	❖ Simple ❖ Transparent and Automatic Partitioning of Data ❖ RemoteException-free distributed computing ❖ Itself is massively scalable ❖ Displaces other technology (messaging) ❖ Extremely configurable	❖ New paradigm ❖ People tend to use old patterns with it – that don't work or are overly complicated ❖ Configuration isn't easy (at first) mainly because of the new paradigm ❖ Takes time for people to "trust" the technology ❖ Extremely configurable

Coherence in the Application-Tier

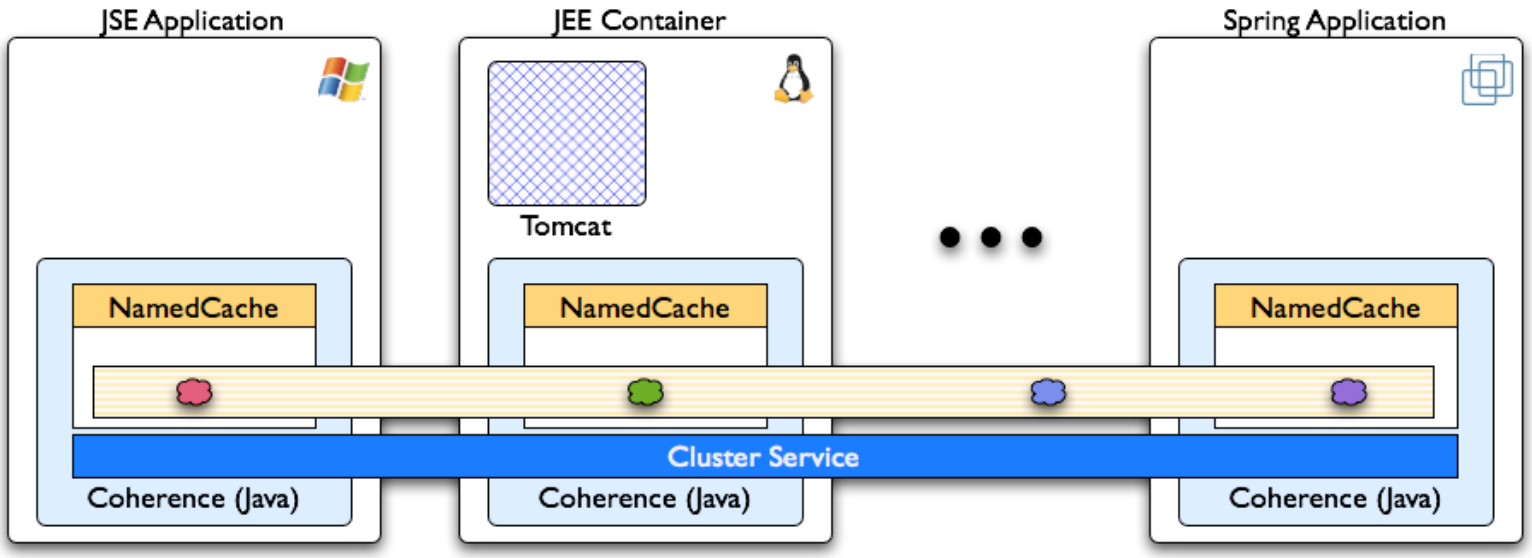
Coherence in the Application-Tier



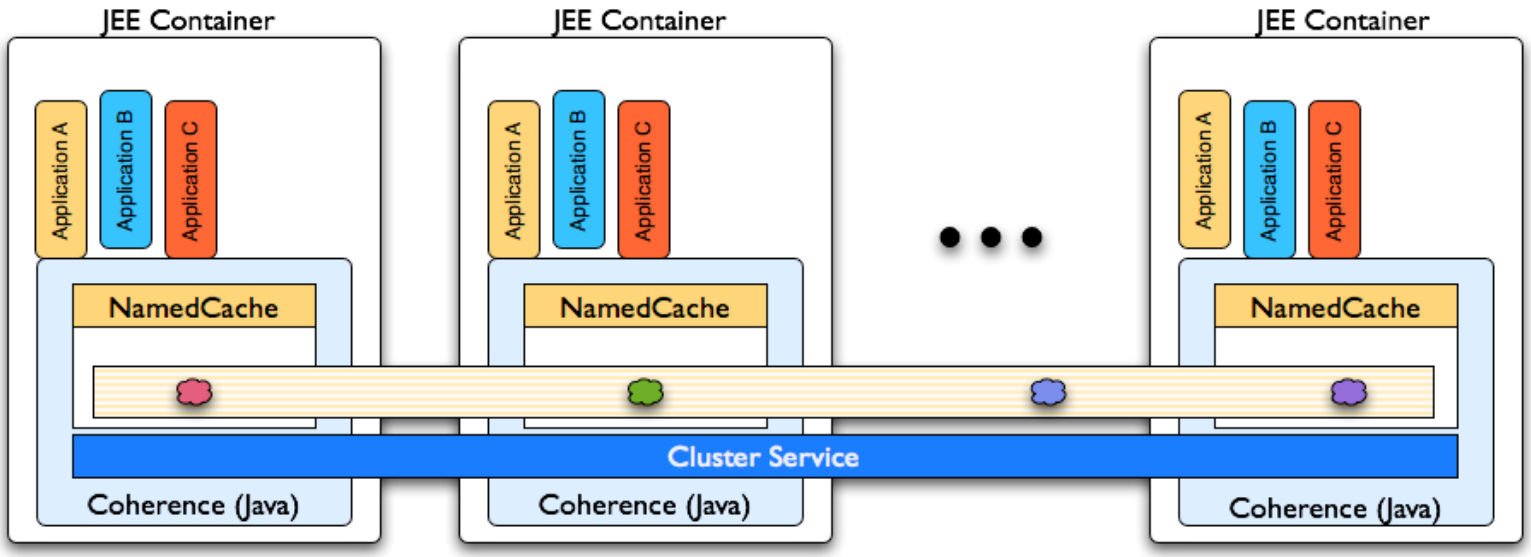
Coherence in the Application-Tier



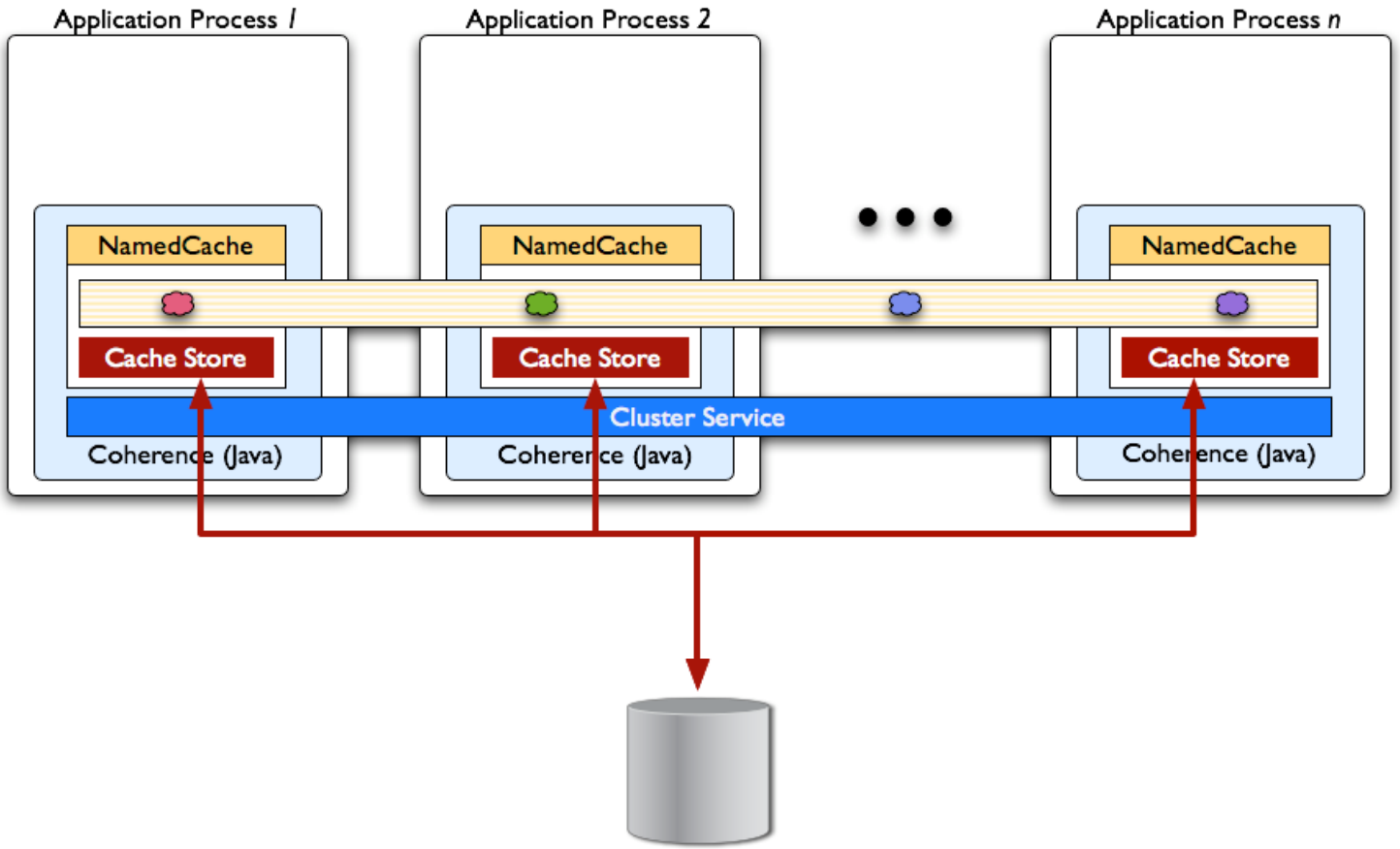
Coherence in the Application-Tier



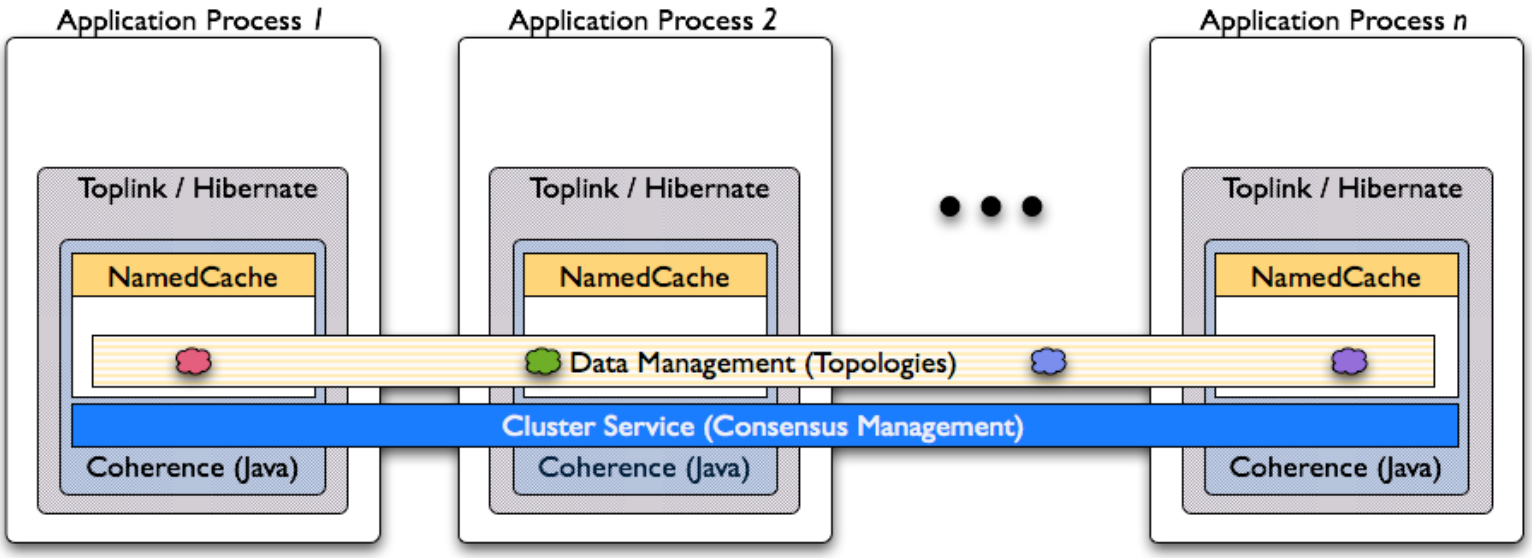
Coherence in the Application-Tier



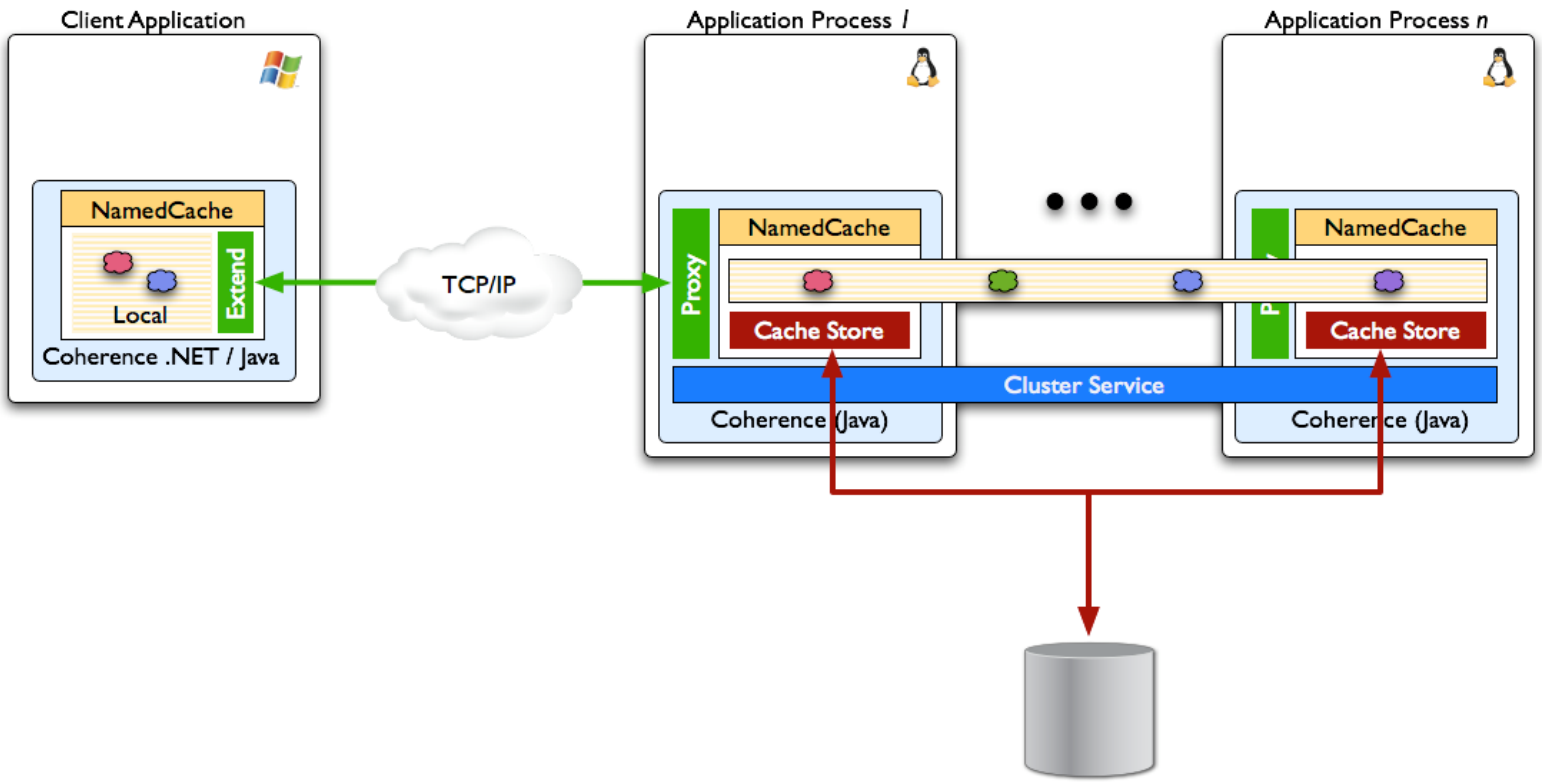
Coherence in the Application-Tier



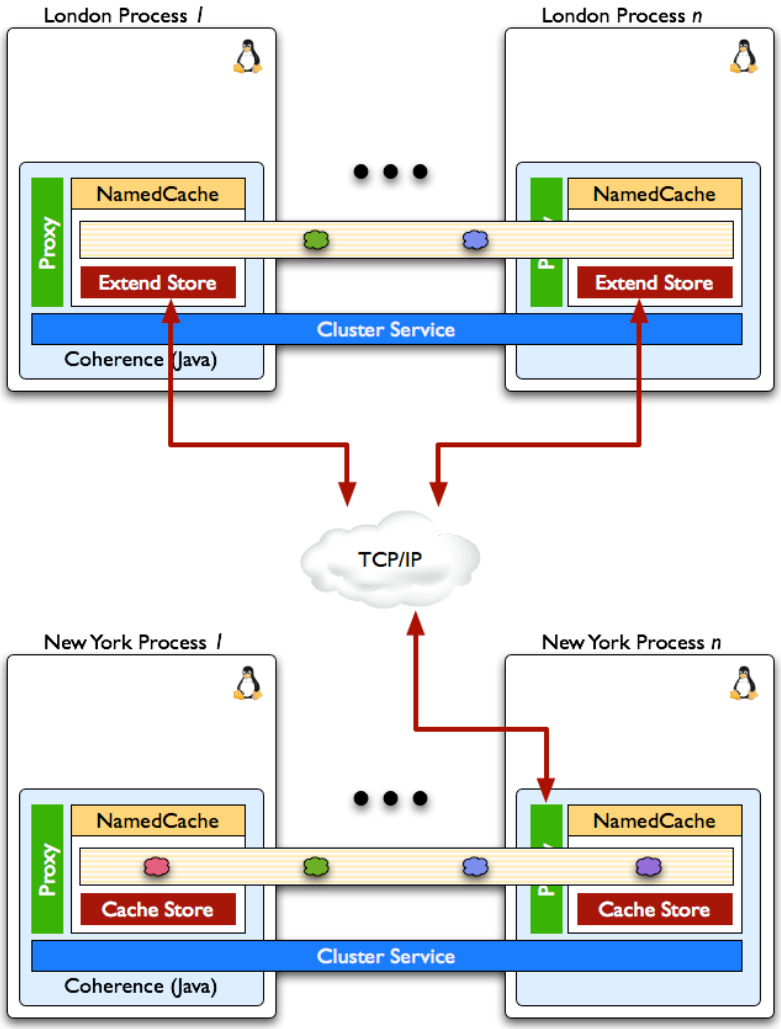
Coherence in the Application-Tier



Coherence in the Application-Tier



Coherence in the Application-Tier



Coherence in the Application-Tier

"Wherever developers use java.util.Map they can use Coherence."

"The sky is the limit."

Coherence Demonstration

Customer Stories

Retailer

- Brick-and-mortar retailer has several online storefronts with read-heavy access to data such as catalogs and inventory
- Bringing up an application server results in heavy load on the database. Bringing up dozens of these instances in a short period of time results in system outages.
- Customer loads the data once into Coherence, and then all subsequent accesses are against Coherence

Insurance Company

- Insurance provider has self-service website for customers
- Database was being crushed by persistence of enormous user profiles (>1MB each) for thousands of concurrent users
- Coherence allows all data to be managed in-memory. User profiles are persisted to the database once, at the end of the user session.
- Bonus: By managing the full dataset in memory, application was able to survive a significant database outage (including deliberate outages)

Hospitality Company

- Need to enable thousands of customer service representatives to maximize per-stay hotel revenue through a price optimization engine
- Throughput challenges due to volume of transactions exceeding database server capacity
- Coherence provides both scale-out transactional data management and instantaneous access to data for the price optimization engine

Gaming Company

- Gaming revenue tied directly to customer activity. Need for high-throughput, low-latency solution for transactions
- Matching engine supporting several thousand matches per second, with intense “hot spots” on specific instruments
- In-memory performance required to manage these hot spots, some of which could account for close to half of all transactions
- Need to scale-out to enable more markets. Markets created and managed dynamically
- Coherence used to manage data and perform transactions

Hedge Fund

- Uses analytical techniques (trading models) to predict trades (no human discretion) from real-time market data
- Real-time market data volumes compounding every few months
- Trading Models need to be constantly available
- Need to trade more markets to reduce risk
- Need to scale system out for real-time event matching and historical back testing

How Coherence Works

Introduction to NamedCaches

- Developers use NamedCaches to manage data
- NamedCache
  - Logically equivalent to a Database table
  - Store related types of information (trades, orders, sessions)
  - May be hundreds / thousands of per Application
  - May be dynamically created
  - May contain any data (no need to setup a schema)
  - No restriction on types (homogeneous and heterogeneous)
  - Not relational (but may be)

Introduction to NamedCaches

- NamedCache implementations are configurable
  - Permit different mechanisms for organizing data
  - Permit different runtime characteristics (capacity, performance etc…)
- A mechanism for organizing data is often called a Topology or more generically, a Scheme
- Coherence ships with some standard schemes
  - You may configure / override / create your own!

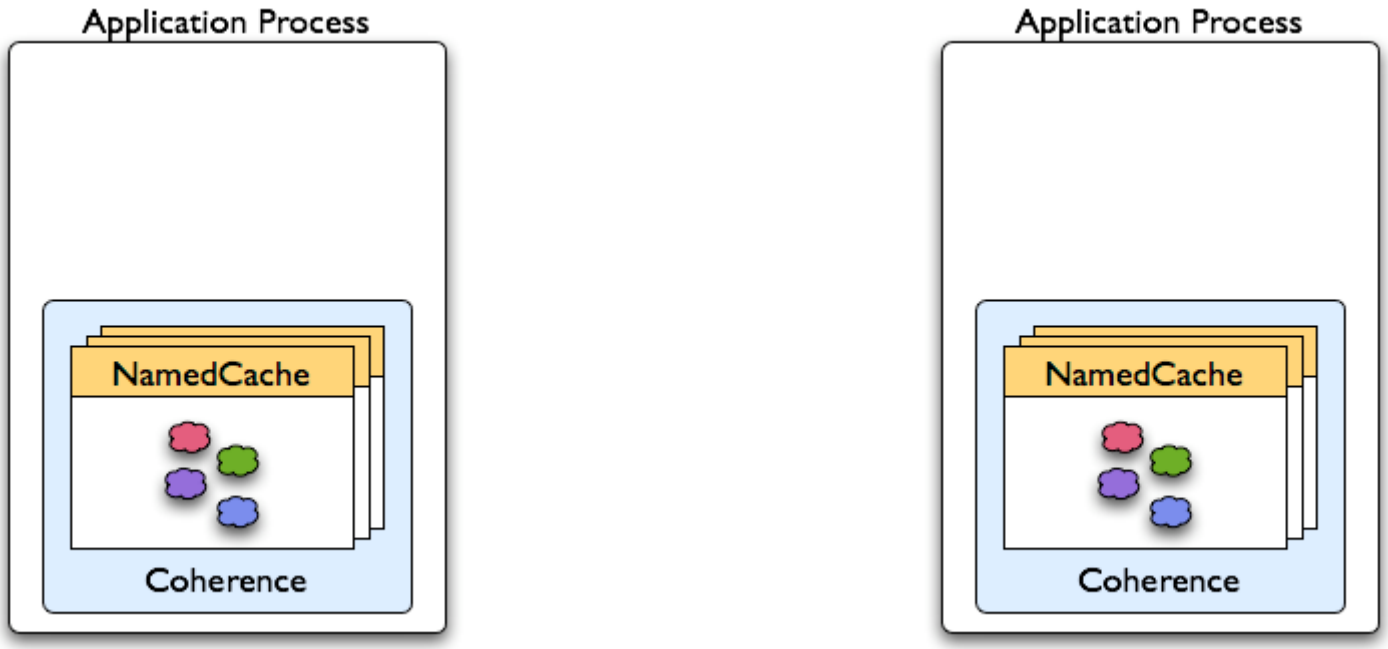
Coherence Schemes

The Local Scheme

The Local Scheme

- Non-Clustered Local Cache
  - Contains a local references of POJOs in Application Heap
- Why:
  - Replace in-house Cache implementations
  - Compatible & aligned with other Coherence Schemes
- How:
  - Based on SafeHashMap (high-performance, thread-safe)
  - Size Limited (if specified)
- Configurable Expiration Policies:
  - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

The Local Scheme



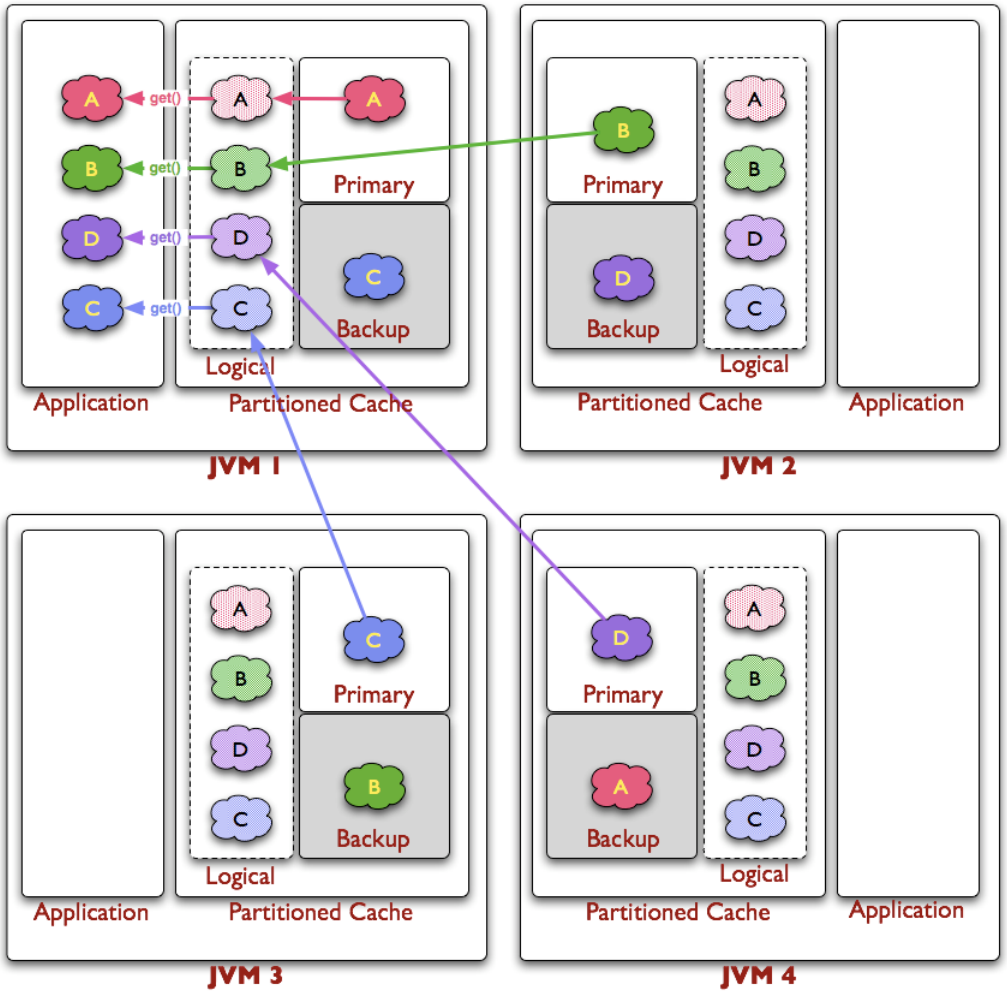
Coherence Schemes

The Distributed Scheme

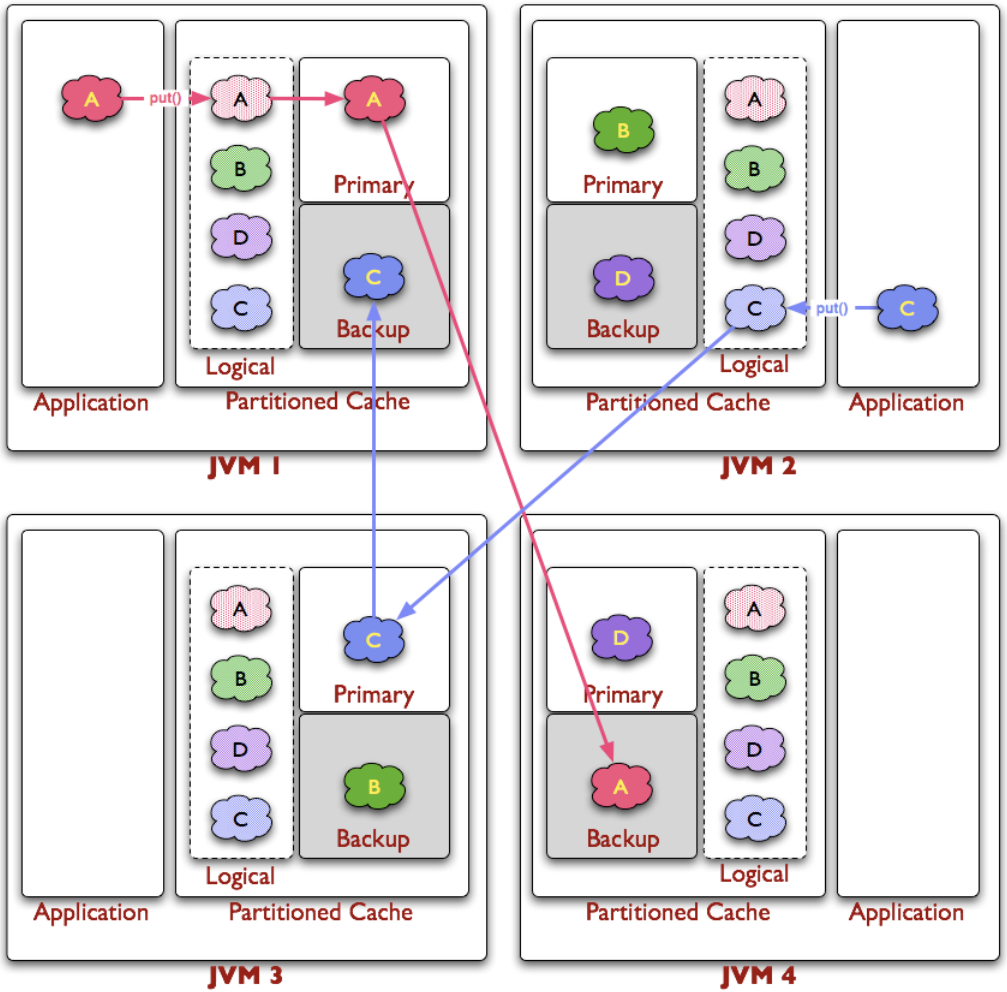
The Distributed Scheme

- Sophisticated approach for Clustered Caching
- Why:
  - Designed for extreme scalability
- How:
  - Transparently partition, distribute and backup cache entries across Members
  - Often referred to as ‘Partitioned Topology’
- Configurable Expiration Policies:
  - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

The Distributed Scheme



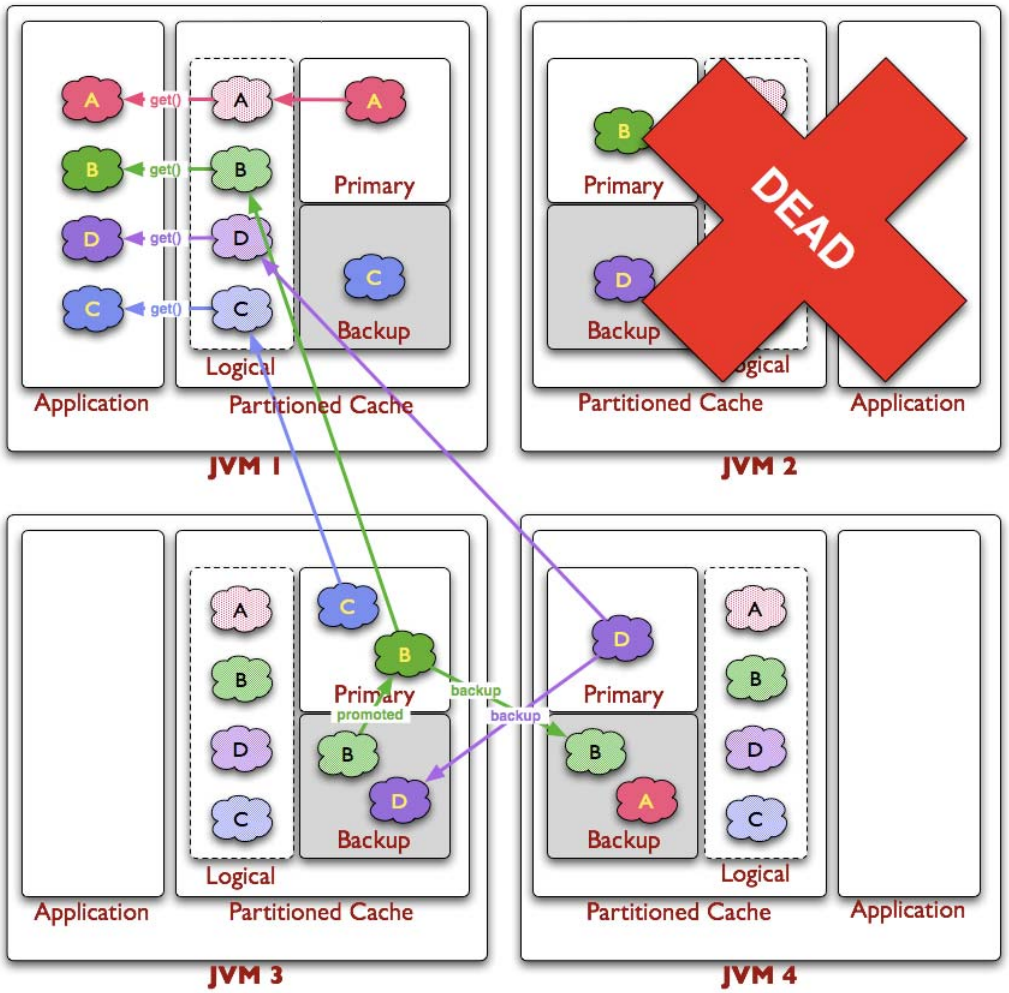
The Distributed Scheme



The Distributed Scheme

- Each Member has logical access to all Entries
  - At most 1 network-hop for Access
  - At most 4 network-hops for Update
  - Regardless of Cluster Size
- Linear Scalability
  - Cache Capacity Increases with Cluster Size
  - Coherence Load-Balances Partitions across Cluster
  - Point-to-Point Communication
  - No multicast required (sometimes not allowed)

The Distributed Scheme



The Distributed Scheme

- Seamless Failover and Failback
  - Backups ‘promoted’ to be Primary
  - Primary ‘makes’ new Backup(s)
- Invisible to Application
  - Apart from some latency on entry recovery
- Recovery occurs in Parallel
  - Not 1 to 1 like Active + Passive architectures
- Any Member can fail without data loss
- Configurable # backups
- No Developer or Infrastructure intervention

The Distributed Scheme

- Benefits:
  - Deterministic Access and Update Latency (regardless of Cluster Size)
  - Cache Capacity Scales with Cluster Size Linearly
  - Dynamically scalable without runtime reconfiguration
- Constraints:
  - Cost of backup (but less than Replicated Topology)
  - Cost of non-local Entry Access (across the network)
    - (use Near Scheme)

The Distributed Scheme

- Lookup-free Access to Entries!
  - Uses sophisticated ‘hashing’ to partition and load-balance Entries onto Cluster Resources
  - No registry is required to locate cache entries in Cluster!
  - No proxies required to access POJOs in Cluster!
- Master / Slave pattern at the Entry level!
  - Not a sequential JVM-based one-to-one recovery pattern
- Cache still operational during recovery!

Group Exercise

Under what conditions should Coherence Failover?

Coherence Schemes

Distributed Scheme Clients & Servers

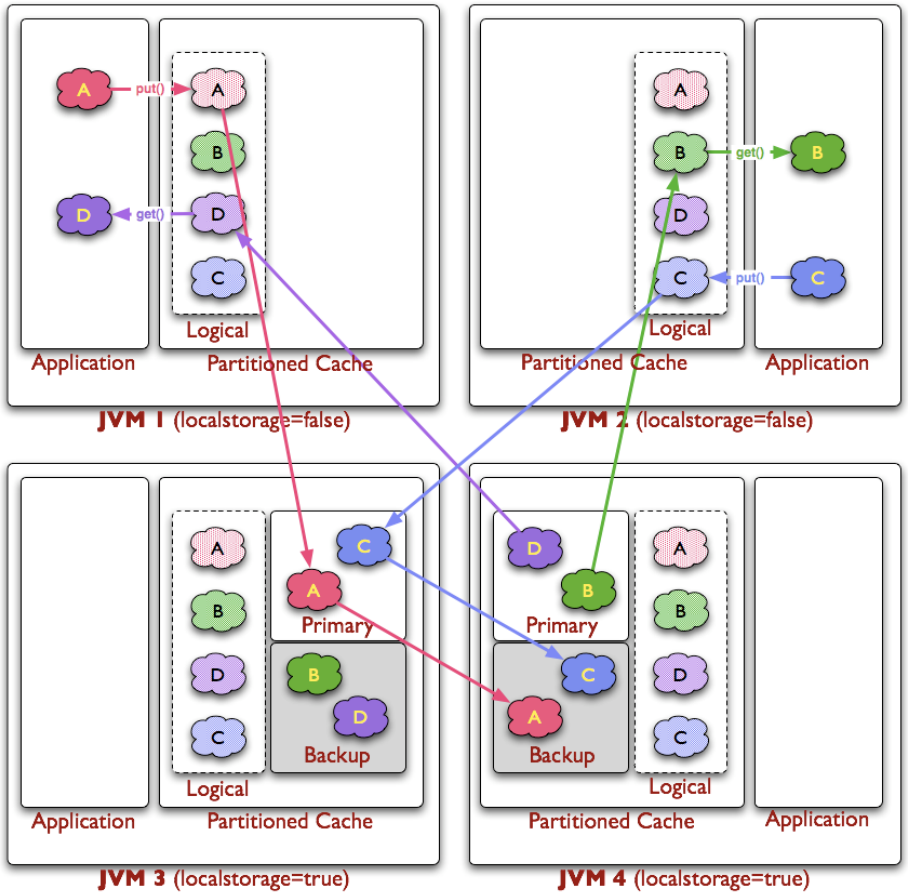
Distributed Scheme
Clients & Servers

- Sometime Members should not store Data
  - Members lifetime in the cluster is short
  - Members join and leave frequently
- Each time Membership changes, partitioning and distribution needs to be re-assessed
- To reduce impact, Members may be ‘storage disabled’

Distributed Scheme
Clients & Servers

- Cache Client
  - Member has storage disabled for Partitioned Topologies
- Cache Server
  - Member has storage enabled for Partitioned Topologies
- Same Cache API
- Transparent to developer
- Storage is (re)configured outside of code

Distributed Scheme
Clients & Servers



Coherence Schemes

Scheme Composition

Scheme Composition

- Schemes may be ‘composed’ to address system requirements and SLAs.
  - Keep recently used data in-memory, the rest on disk
- Base Schemes
  - Class, Local, Replicated, Distributed, Extend*
- Composite Schemes:
  - Near, Overflow (to disk)
  - Allow other schemes to be ‘plugged in’

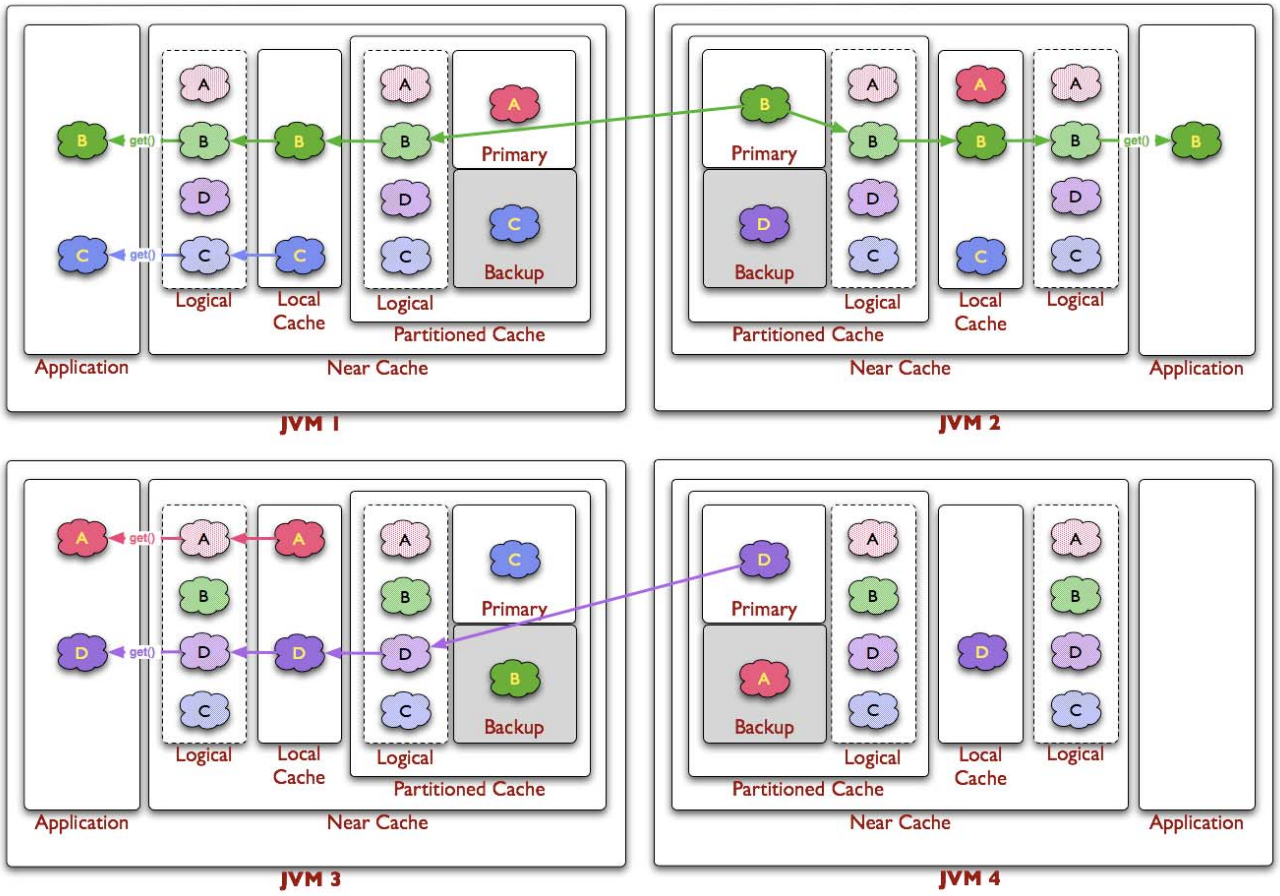
Coherence Schemes

The Near Scheme

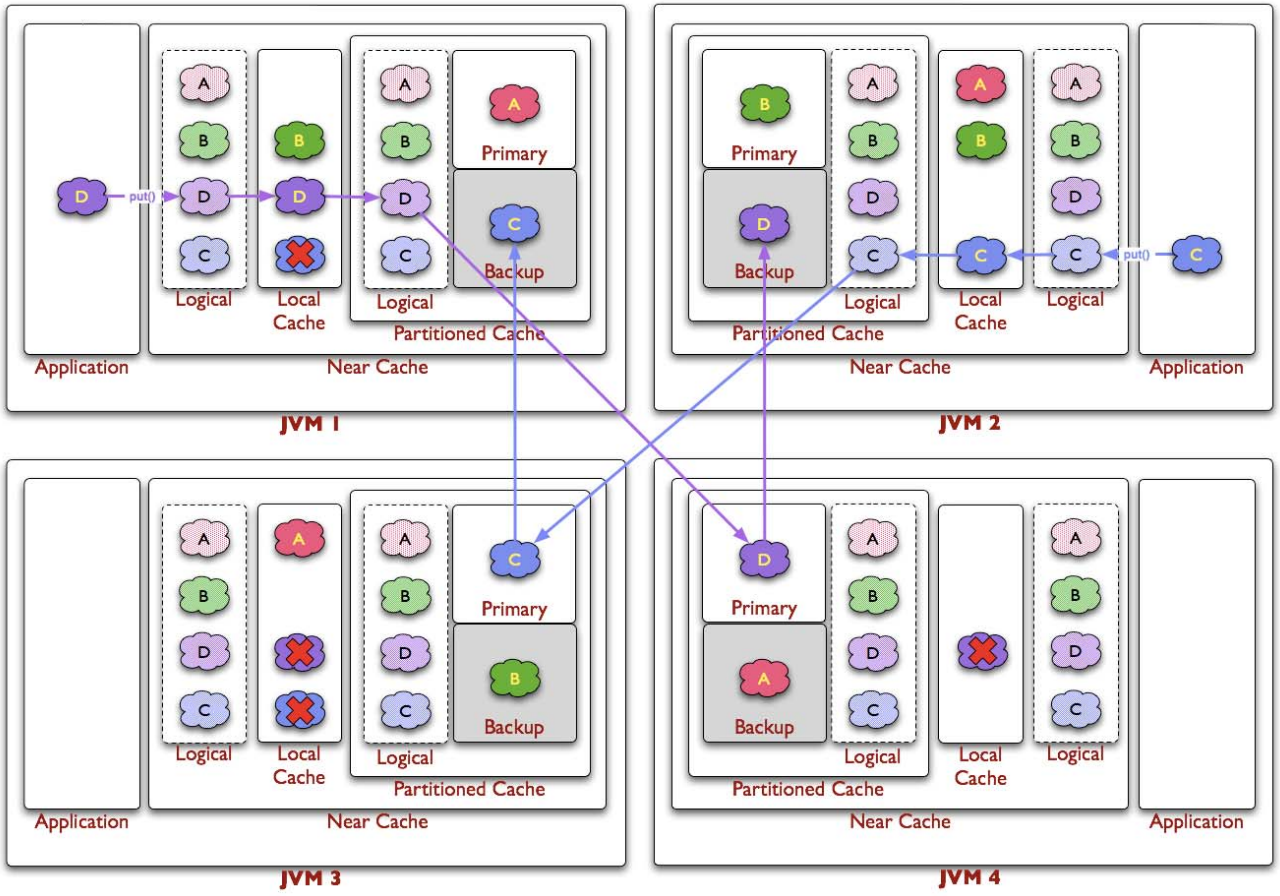
The Near Scheme

- A composition of pluggable Front and Back schemes
  - Provides L1 and L2 caching (cache of a cache)
- Why:
  - Partitioned Topology may always go across the wire
  - Need a local cache (L1) over the distributed scheme (L2)
  - Best option for scalable performance!
- How:
  - Configure ‘front’ and ‘back’ topologies
- Configurable Expiration Policies:
  - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

The Near Scheme



The Near Scheme

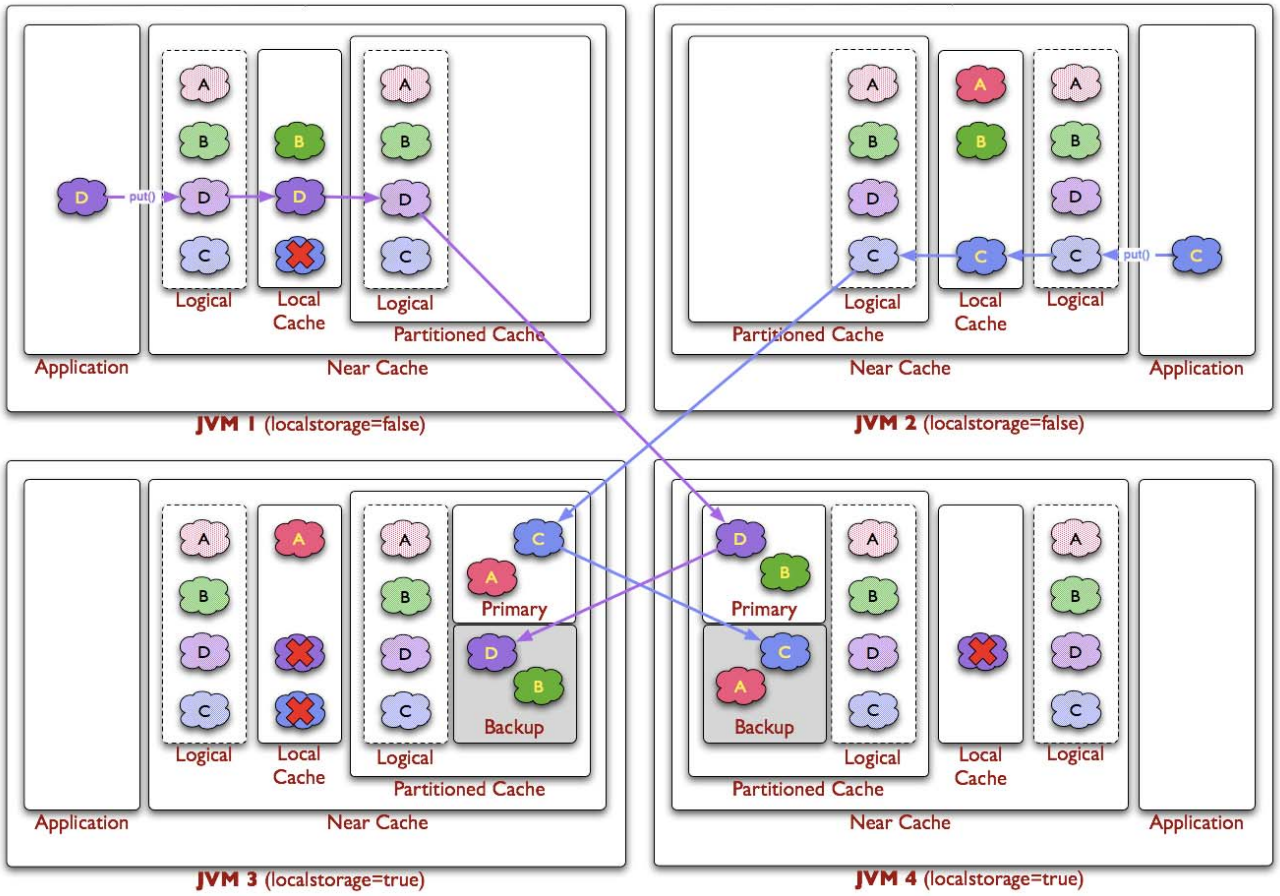


The Near Scheme Coherency Options

- Local Cache Coherency Options
  - Seppuku: Event-Based ‘Kill Yourself’ Invalidation
  - Standard Expiry: LFU, LRU, Hybrid, Custom

- No messaging system required for invalidation!
  - Built into infrastructure
  - High-performance

The Near Scheme



Coherence Schemes

The Mechanics of Schemes

The Mechanics of Schemes

- Schemes themselves are customizable (pluggable)
- The underlying component that manages data in a scheme can be replaced / customized!
  - This map is called the BackingMap
- Examples:
  - Handle ‘overflow’ by writing to disk
  - Handle ‘cache misses’ by reading from Data Source
  - Write to Data Source on ‘put’
  - Use Extend* to connect to and access other Clusters

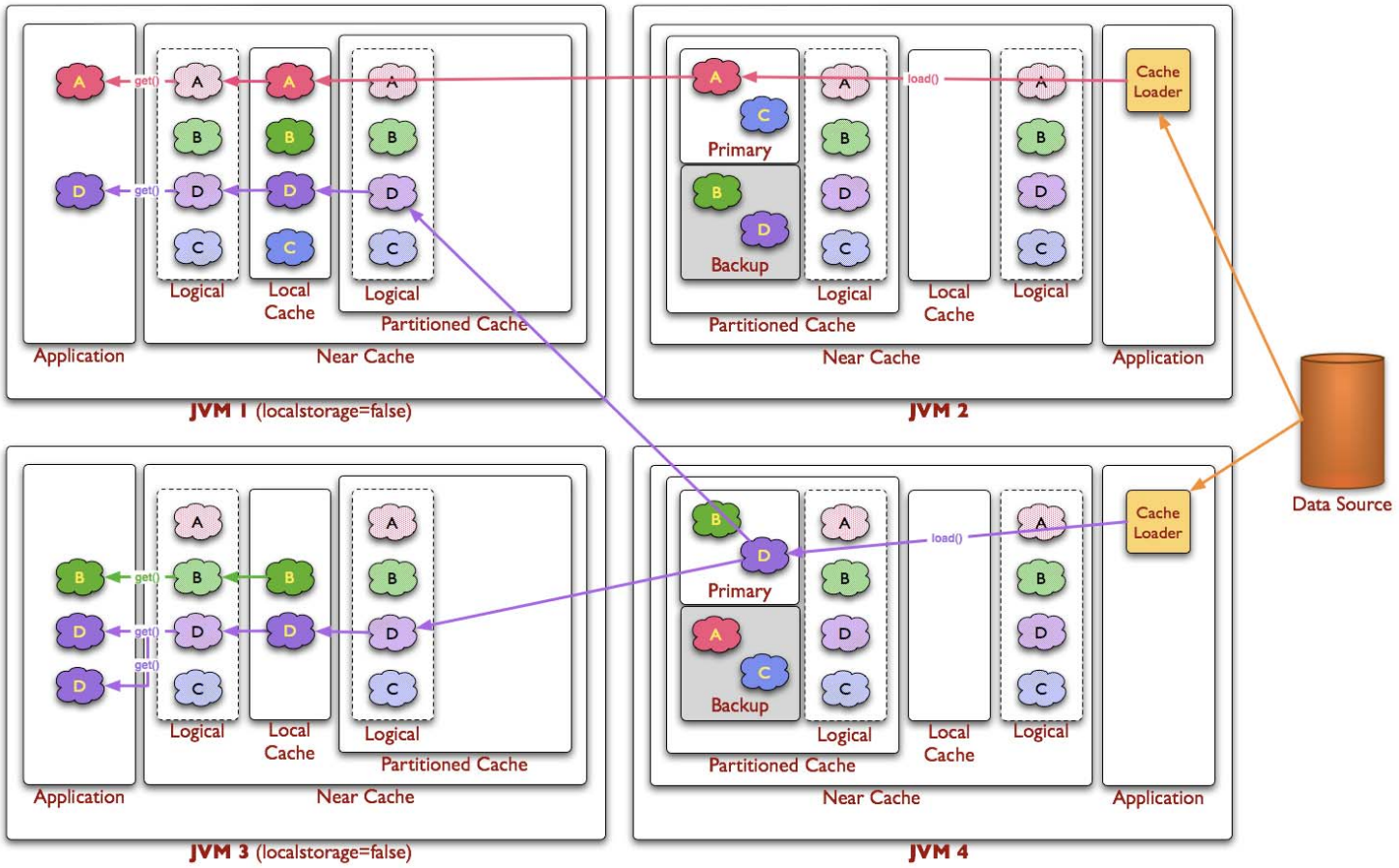
Coherence Schemes

Data Source Integration

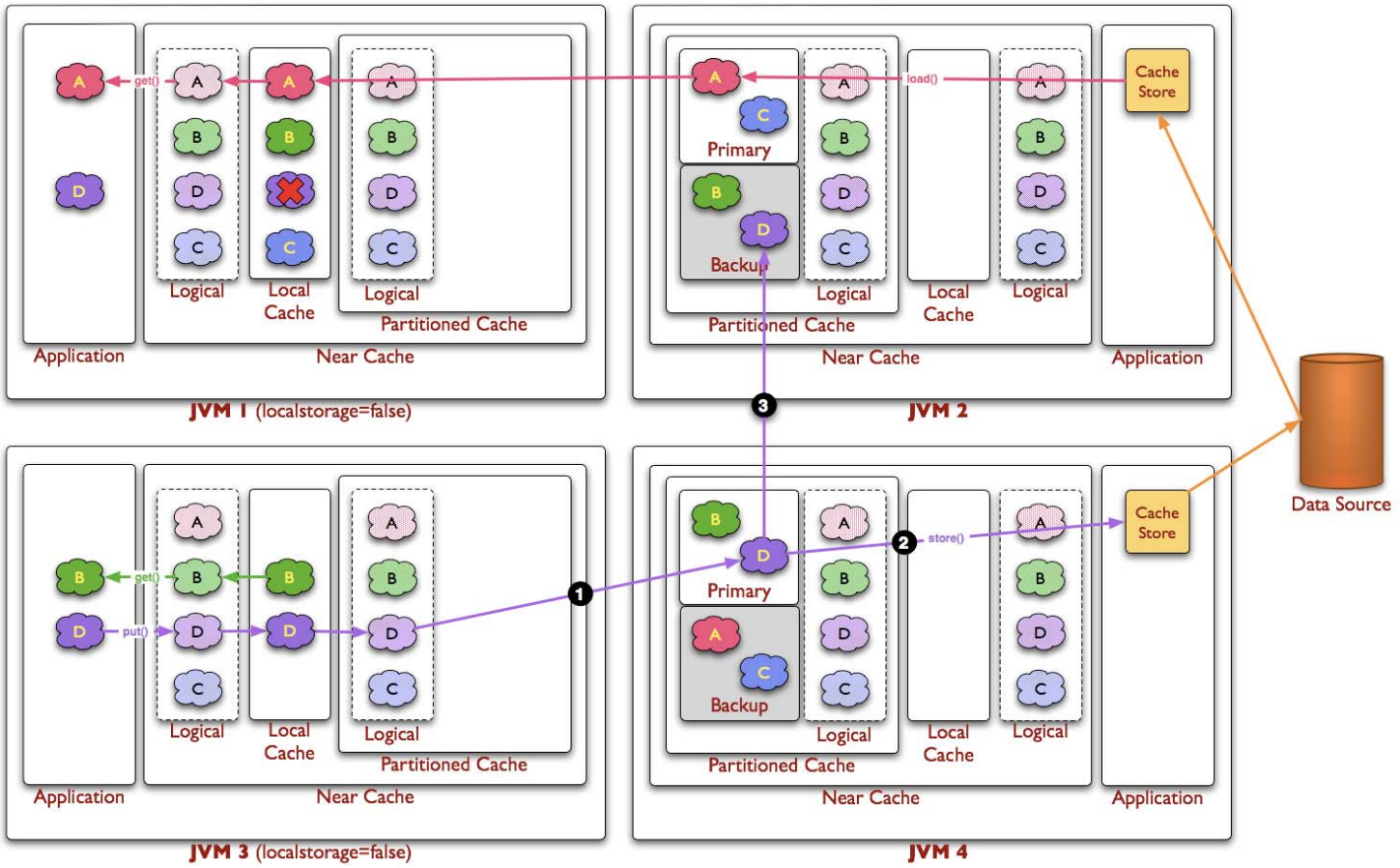
Read Write Backing Map

- One of many BackingMaps that can be used to customize Coherence
- Read Write Backing Map provides a mechanism to integrate directly with a Data Source.

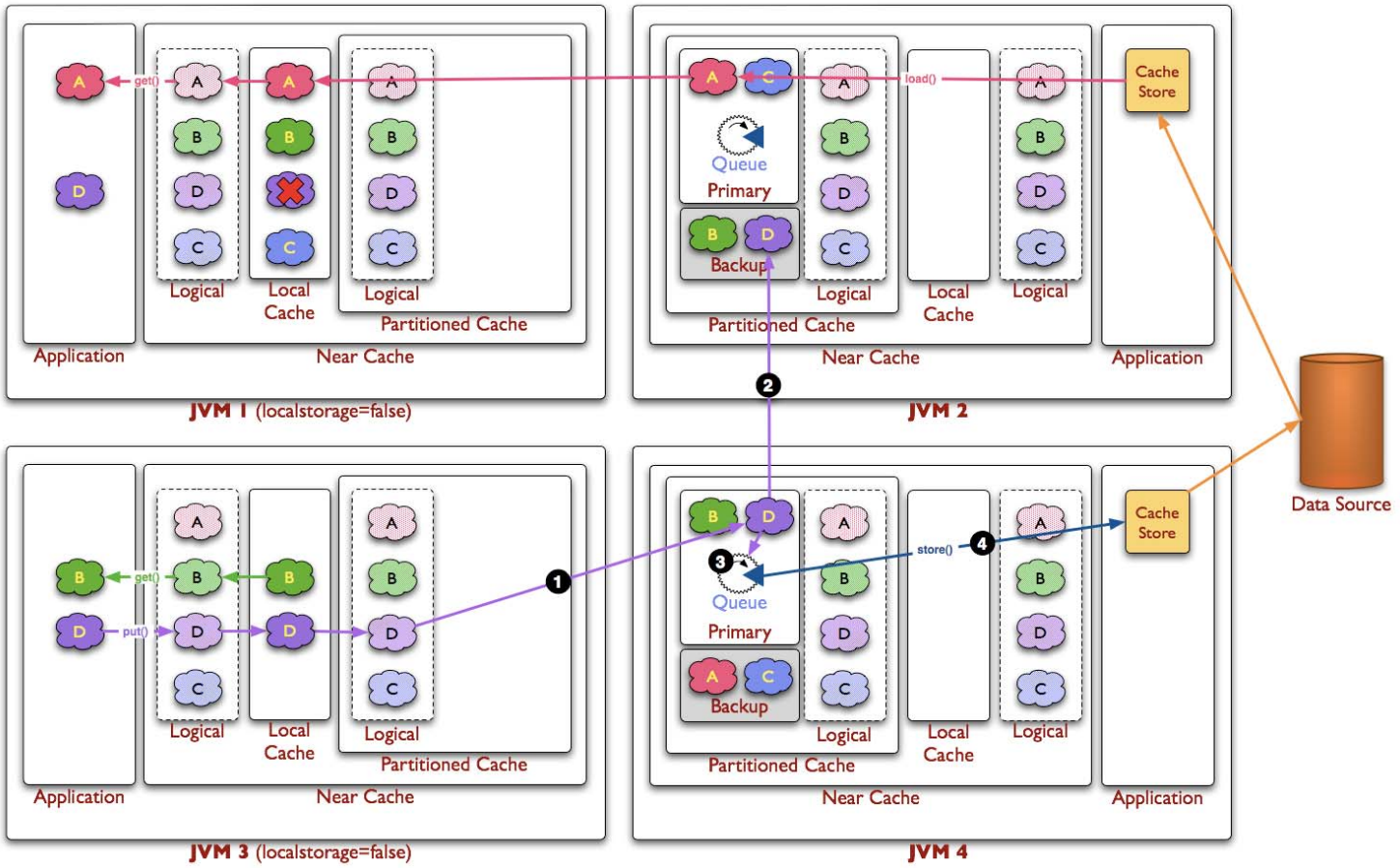
Data Source Integration



Data Source Integration



Data Source Integration



Coherence Demonstration (Revisited)

Grids and Data Grids

Different uses of “Grid”

- Compute Grids
  - Platform Symphony and LSF (batch)
  - DataSynapse GridServer
- Shared Infrastructure Provisioning
  - Platform EGO
  - DataSynapse FabricServer
  - WebSphere XD
  - GigaSpaces Enterprise
  - VMWare

Different uses of “Grid”

- Database Grids
  - Oracle RAC
- Data Grids
  - **Oracle Coherence**
  - IBM ObjectGrid
  - Gemstone Gemfire
  - GigaSpaces Enterprise

Clusters, Grids and Data Grids

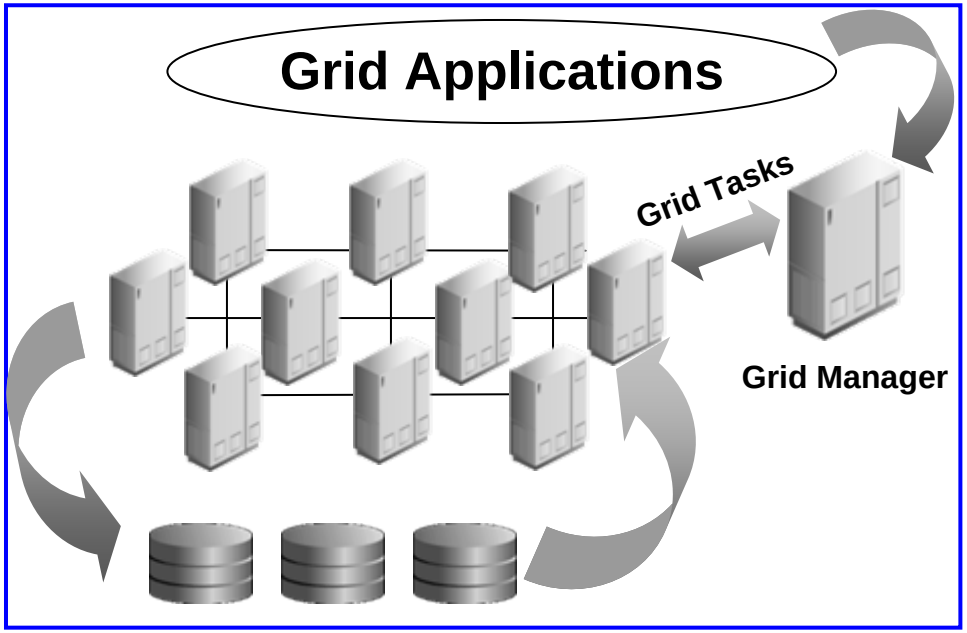
	Clusters	Compute Grids	Data Grids (Coherence)
Goals	Availability Performance	Scalability Capacity	Availability Scalability Capacity Performance
Resources	Homogeneous	Heterogeneous	Both
Utilization	Fixed Single Purpose	Dynamic Multi-purpose	Dynamic Multi-purpose
Configuration	Static	Dynamic	Dynamic
Scale-Out Process	Add Resources Reconfigure	Add Resources (on the fly)	Add Resources (on the fly)

Clusters, Grids and Data Grids

	Clusters	Compute Grids	Data Grids (Coherence)
Data / Service Partitioning	Manually Configured	Dynamic	Dynamic
Hardware Coupling	Tight	Loose (Virtualized)	Loose (Virtualized)
Failover / Recovery Process	Manually Configured (one to one)	Transparent (one to many)	Transparent (one to many)
Processing	Client-Based	Grid-Based	Both

Grid Computing Evolution: Part I

Traditional Compute Grid



- Emphasis on orchestrating tasks out to compute nodes in grid
- Data Set either loaded locally or pulled off of back end data source
- Applications Highly Customized for Grid Environment

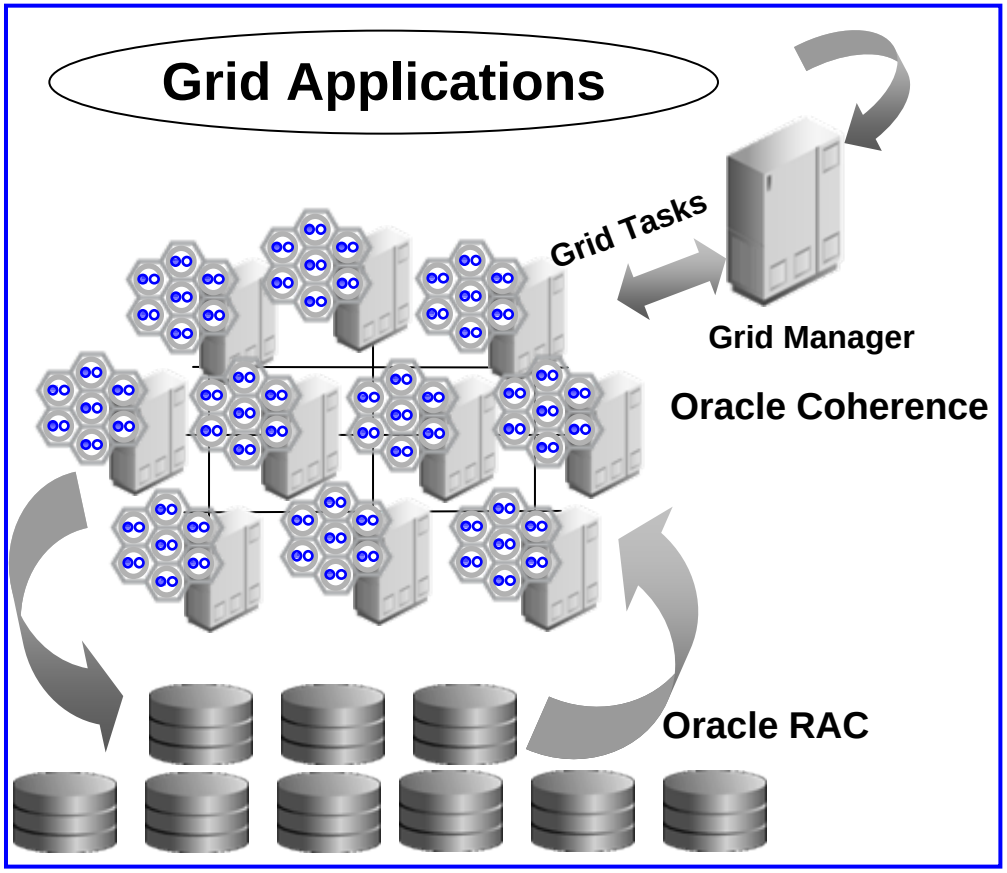
Great processing scalability with inevitable data bottlenecking

Orchestration can be point of bottleneck as well

Grid Computing Evolution: Part II

Traditional Compute Grid with Data Scale Out

High Performance Computing (HPC)



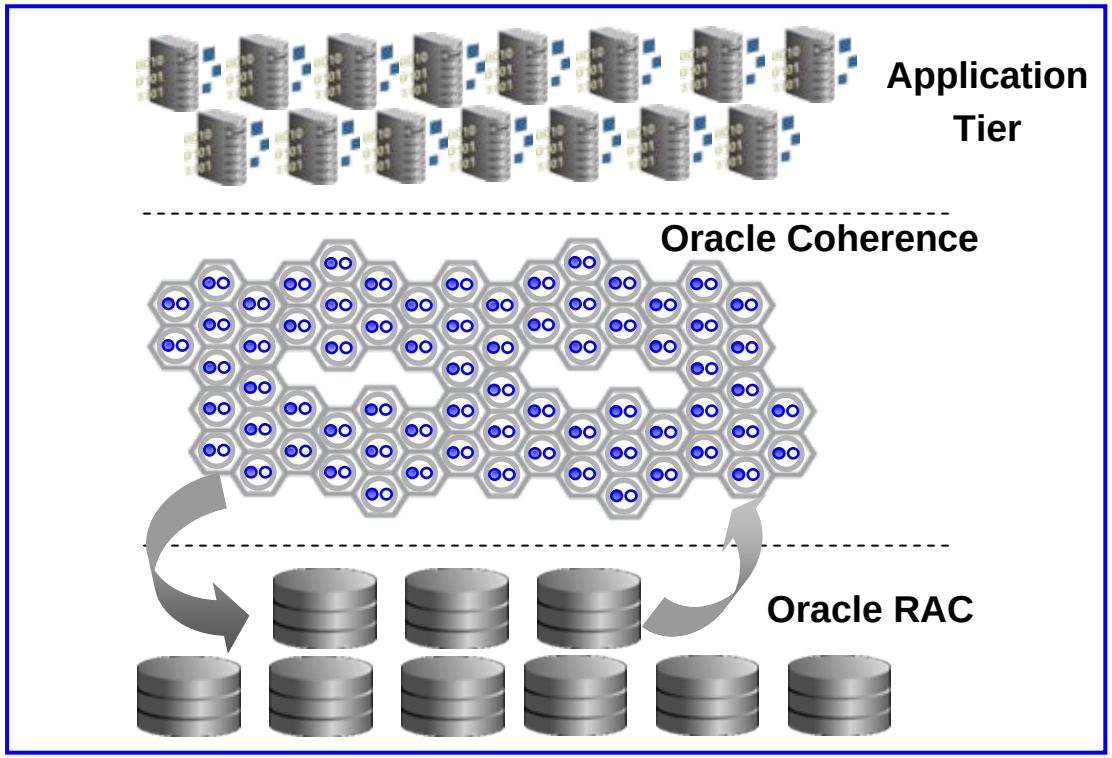
- Oracle Coherence Data Grid Overlay onto Compute Grid
- Compute Grid Scale Out with Data Fault Tolerance
- Massive Persistent Scale Out with Oracle RAC

Applications still Highly Customized for Grid Environment!

Oracle Grid Computing: Enterprise Ready

Enterprise Application Grid

Extreme Transaction Processing XTP



- **Common Shared Application Infrastructure *(Application Virtualization)***
- **Data Virtualization *(Data as a Service)***
- **Middle tier scale out for Grid Based OLTP**
- **Massive Persistent scale out with Oracle RAC**

How much effort?

How much effort?

"Coherence is the easiest means of achieving massive scalable performance"

- Customer developer takes on added responsibility
- Developer explicitly controls data management
  - Slightly lower-level than the usual SQL queries and transactions
  - Working with Objects, not relational rows
- In practice is often easier than working with a database
    - Developers tend to dislike databases but like simple APIs

How much effort?

- How does a developer access the Data Grid?
- Customer development team modifies the application to access Coherence via an API
- Some “drop in” functionality
  - HTTP session management
  - Clustered caches for Hibernate, TopLink, BEA Portal, etc.

How much effort?

- Simple caching or session management
  - A week or two of work to change the application and set up the production environment
- Extreme Transaction Processing (XTP)
  - Several months of development followed by several weeks of preparation for production
- ROI
  - Massive Predictable Scalability
  - High Availability, High Performance, Parallelisation

How much effort?

- Most customers start off simple
  - Then grow into more advanced usage
  - Buy-in for simple caching, then want more
  - Buy-in for advanced functionality, but start off with the basics to get quick value and also gain production experience
  - Recommended Approach!
- Most customers, an iterative approach
  - Incrementally taking advantage of more and more functionality

How much effort?

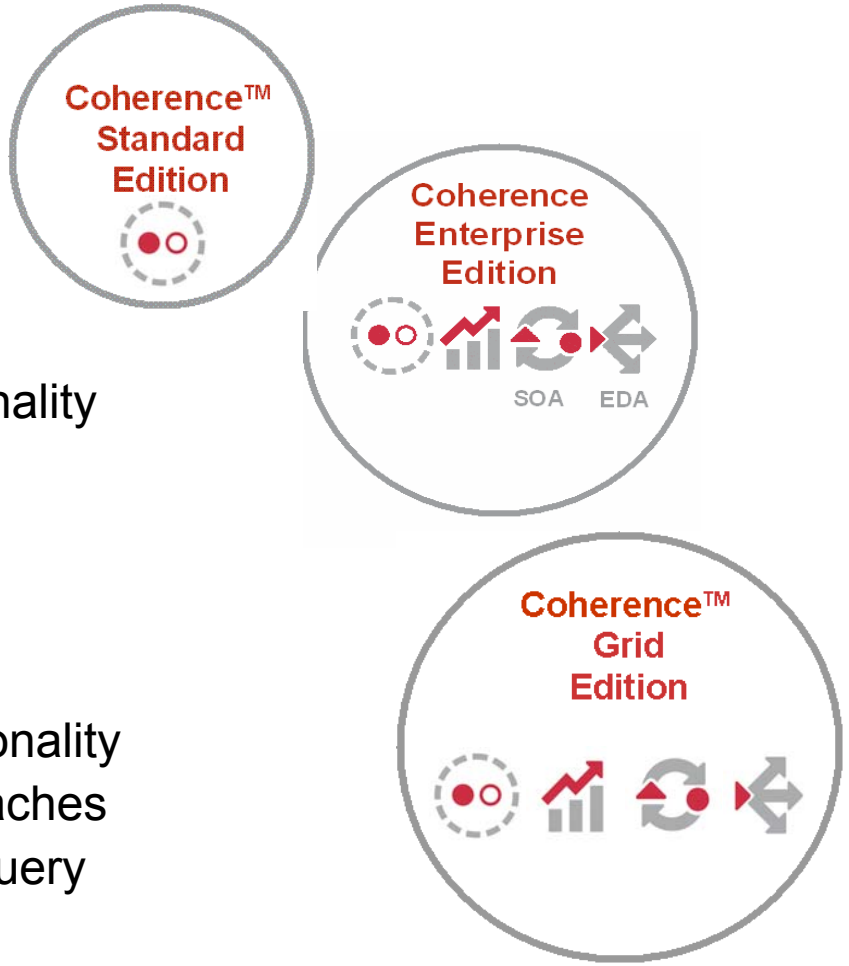
- Many customers can go into production without any assistance
  - But we try to be involved to avoid surprises in production
  - ie: support!

- For the typical customer:
  - 2-5 days of onsite support for POC and initial development
  - A few discussions with support via email/phone
  - 1 day review of production plans
  - The customer handles the rest

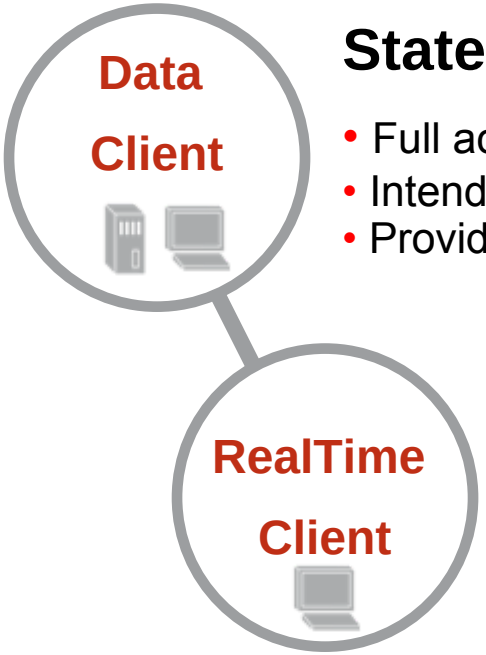
Coherence Editions

Oracle Coherence Product Set

- **Standard Edition**
  - *Baseline Functionality*
  - Clustered Caching
- **Enterprise Edition**
  - *Application Server Market*
  - Includes Standard Edition Functionality
  - Read/Write Through Data
  - Transactional
- **Grid Edition**
  - *Grid Market*
  - Includes Enterprise Edition Functionality
  - Desktop Clients and Near/Local caches
  - Real Time Clients – Continuous Query



Coherence Grid Clients



Stateless desktop and server access to the data grid

- Full access to data and services
- Intended for enterprise-wide distribution
- Provided in all editions

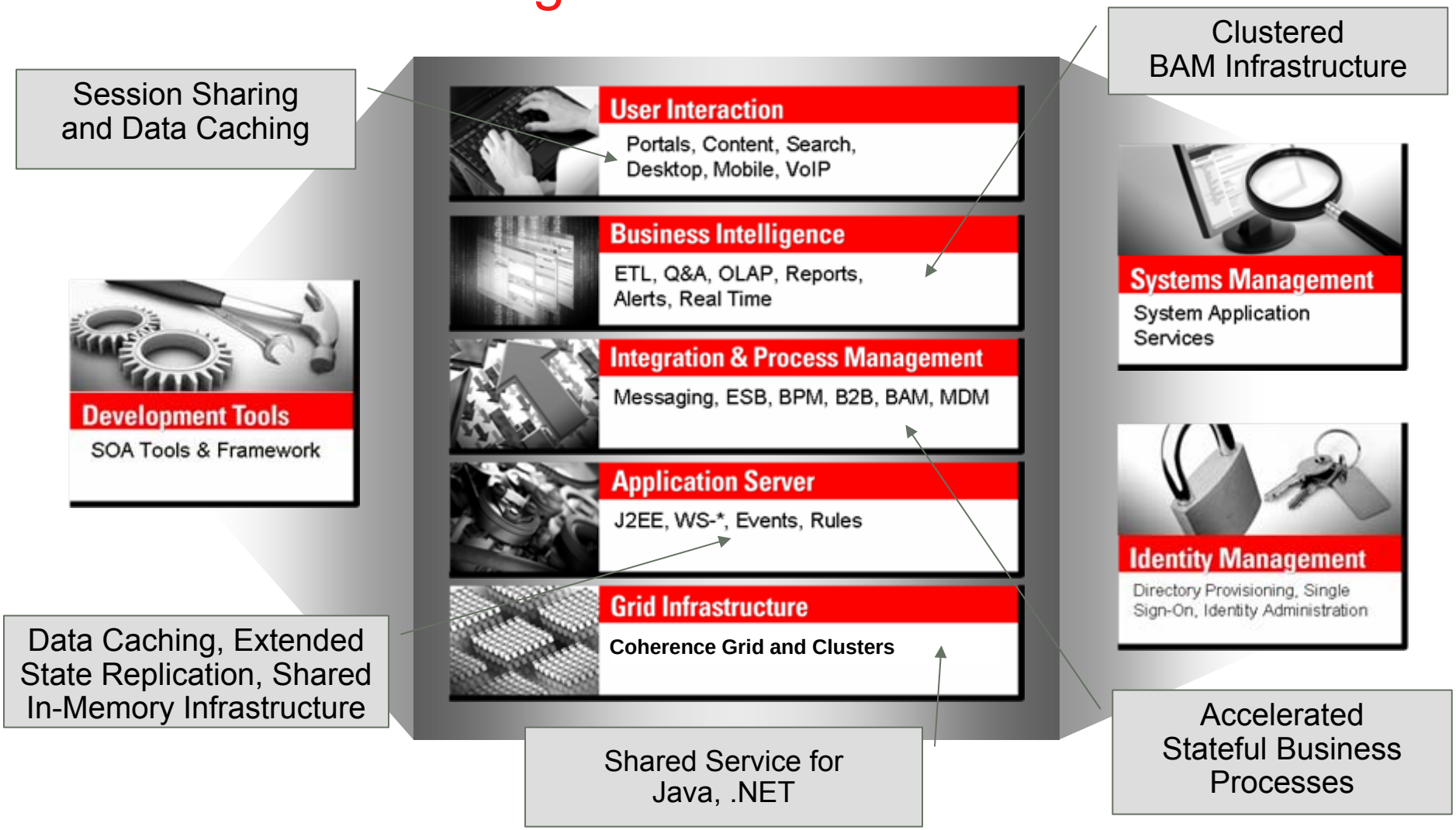
Provides instantaneous view of data on user desktops whenever it changes in the data grid

- Real time data feeds to the desktop (positions, prices, logistics)
- Ready for transactional usage
- First class access to data across the entire enterprise

		Standard Edition	Enterprise Edition	Grid Edition
Edition Benefits	Fault-tolerant data caching	✓	✓	✓
	Data management including write-behind, transactions, analytics and events		✓	✓
	Support for heterogeneous clients			✓
Connectivity	Embedded Data Client and Real Time Client functionality	✓	✓	✓
	TCMP cluster technology	✓	✓	✓
	Multicast-free operation (WKA)		✓	✓
Management & Monitoring	Management host	✓	✓	✓
	Manageable via clustered JMX		✓	✓
Caching	Local cache, near cache, continuous query cache, real-time events	✓	✓	✓
	Fully replicated data management	✓	✓	✓
	Partitioned data management	✓	✓	✓
	Data source integration via read-through/write-through caching	✓	✓	✓
Integration	Hibernate integration	✓	✓	✓
	HTTP session management for application servers	✓	✓	✓
Analytics	Parallel InvocableMap & QueryMap	✓	✓	✓
Transactions	Write-behind caching		✓	✓
	J2CA Resource Adapter	✓	✓	✓
	TransactionMap support	✓	✓	✓
Compute Grid	InvocationService	✓	✓	✓
	WorkManager	✓	✓	✓
Enterprise Data Grid	WAN support	✓	✓	✓
	Support for Compute Clients			✓
	Support for cross-platform Real Time			✓

Fusion Middleware Integration (Available Now)

Oracle Fusion Middleware
Coherence Integration Points



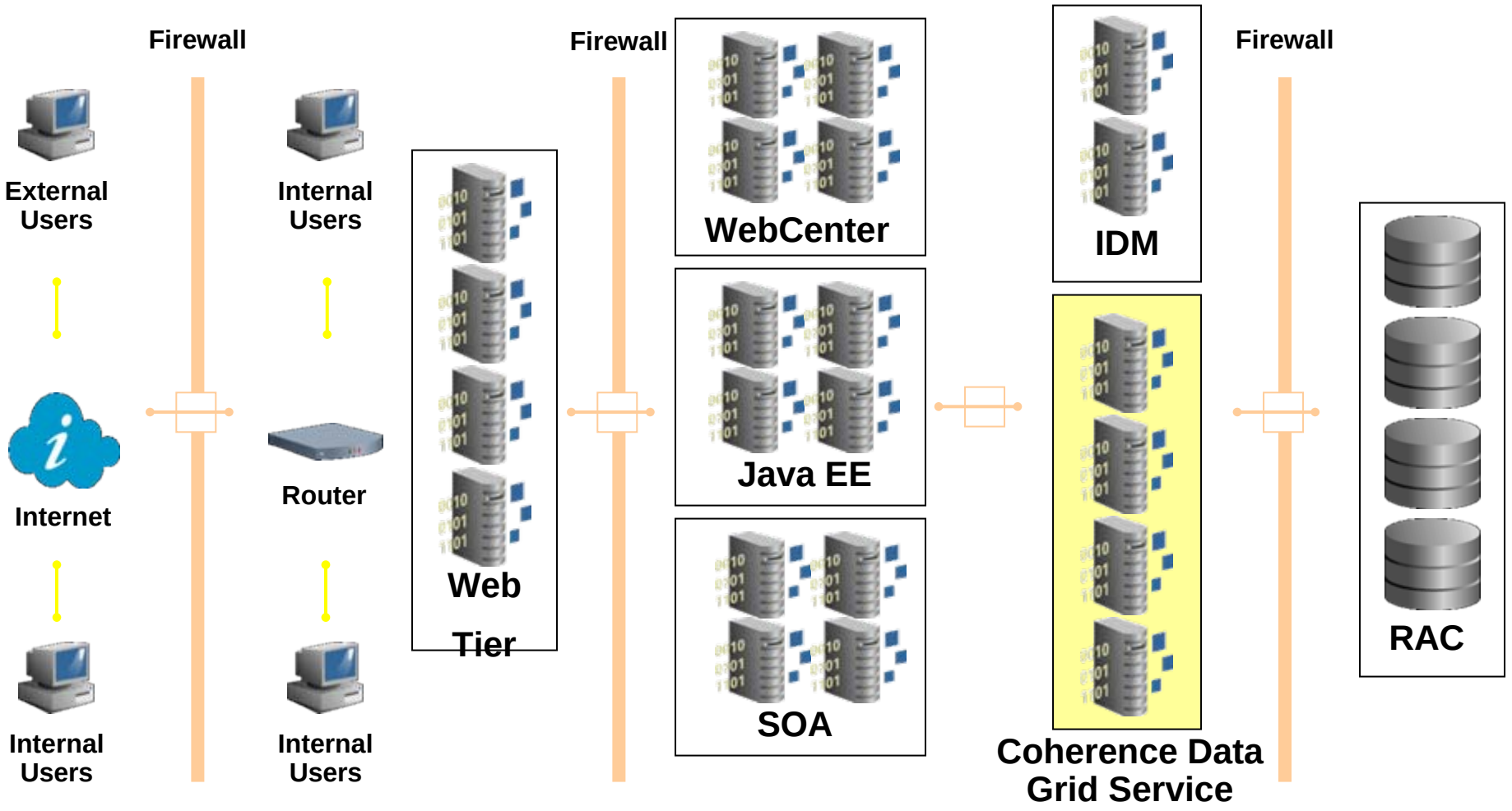
Coherence and Fusion Middleware Short Term Integration

- Fusion Middleware with Coherence Grid
- HTTP Session State Management
- Coherence Persistence with TopLink
- SOA Integration
- Maximum Availability Architecture

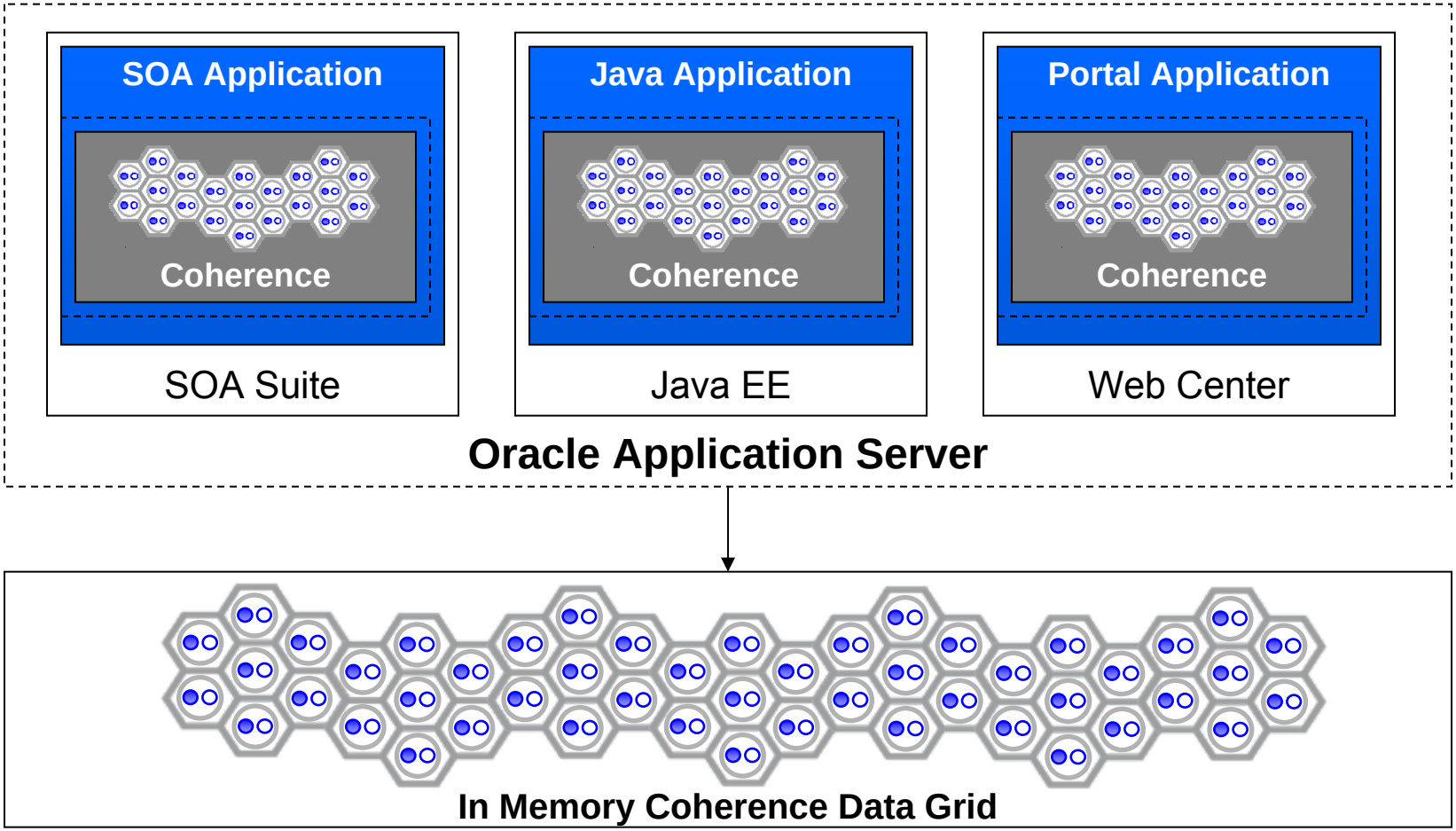
Coherence Grid with Fusion Middleware

- Deployed as separate tier
  - Provides shared in-memory data grid to all FMW
- Embedded in middleware applications
  - Provides in-memory data grid for application layer
- Deploy both in middleware and separate tier
  - Common scenario

Coherence as a Separate Tier with FMW



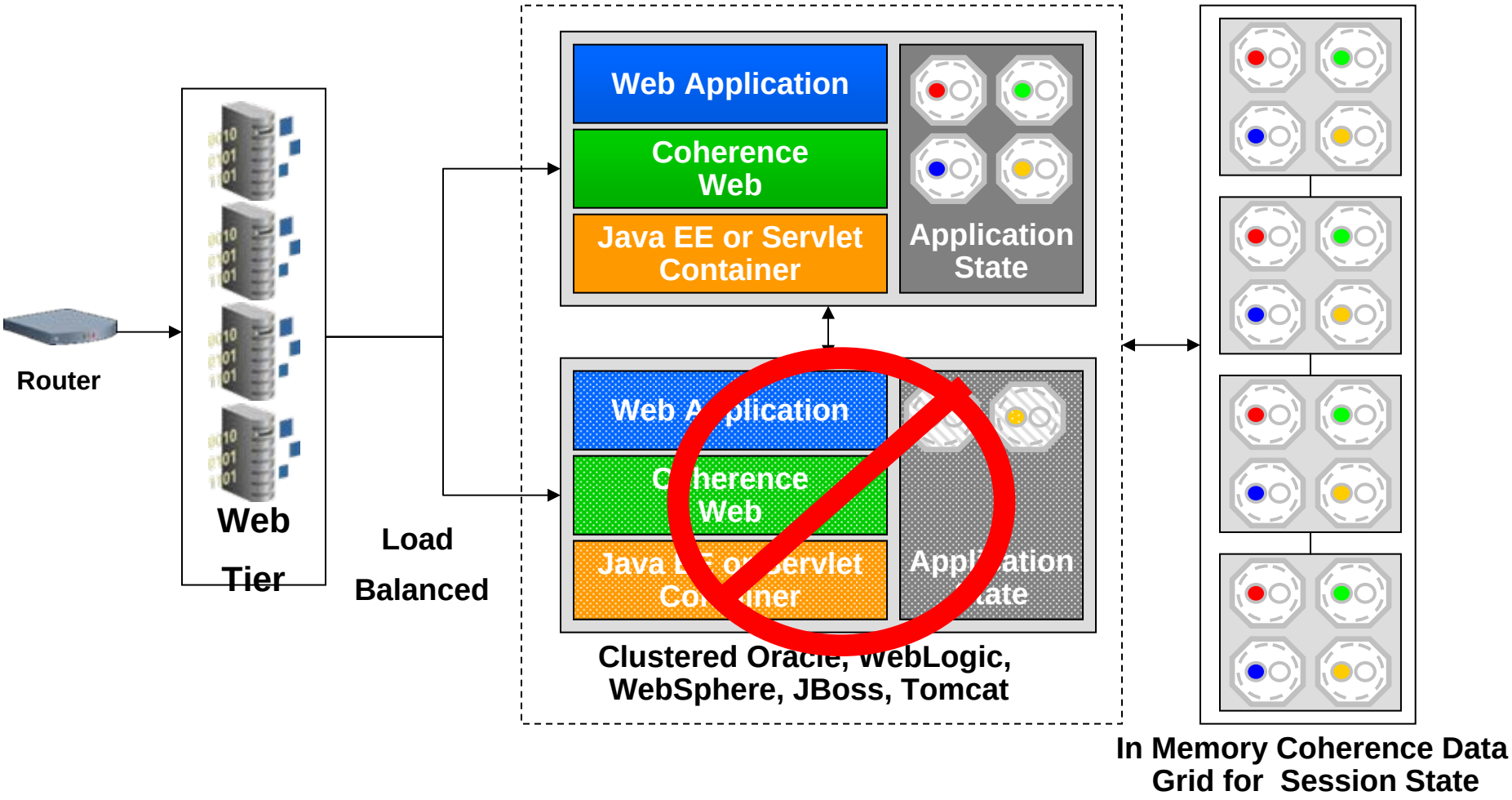
Coherence Embedded and Separate Tier



Session State Management Integration

- **Coherence*Web is a generalized state replication framework for any application server**
  - Certified with JBoss, WebSphere, WebLogic, Tomcat, SunOne
- **Plugs directly into Oracle Application Server**
  - HTTP session only
  - Augments existing HTTP session state replication
  - Stateful EJB replication uses existing OracleAS infrastructure
- **Value with Oracle Application Server**
  - More sophisticated state replication - policy based
  - Transactionality for session replication
  - Offload state replication to independent tier from application server

Coherence*Web: Session State Management

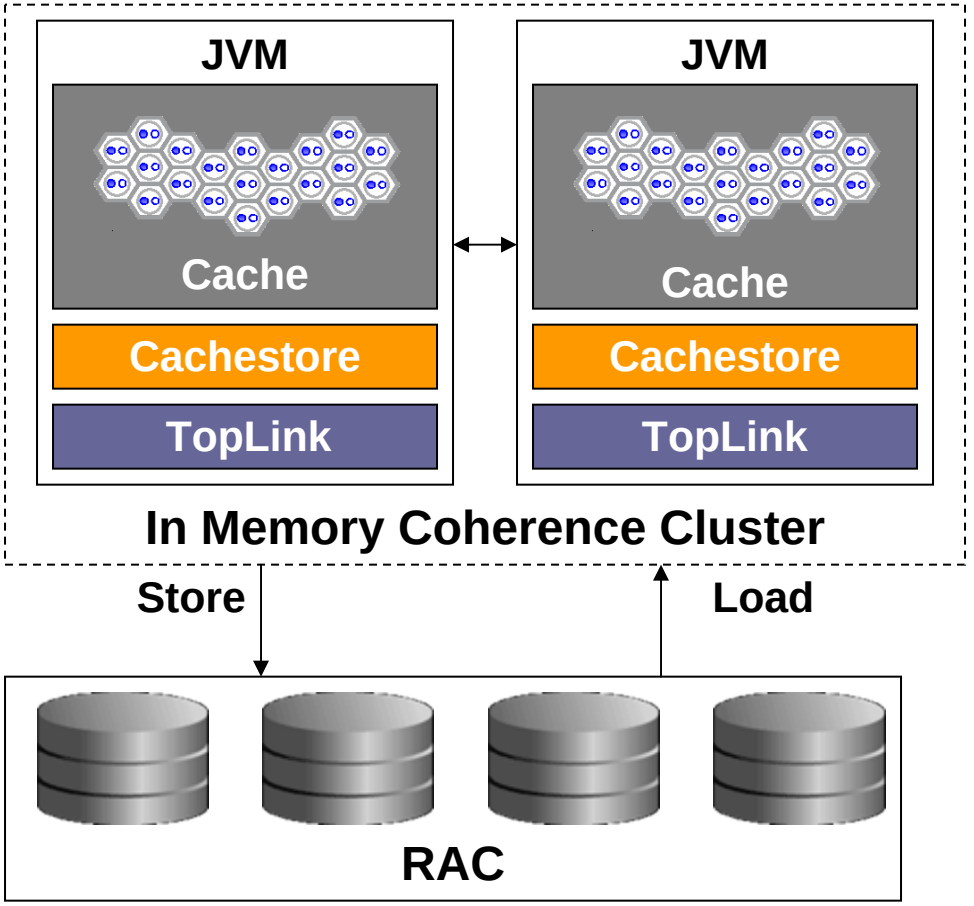


Coherence Persistence Integration

- **Coherence integrates tightly with databases**
  - Read through – pass query through to database
  - Write through – persist cache data to database
  - Refresh Ahead – refresh cache from database
  - Write Behind – asynchronously persist to database
- **Persistence solution integration**
  - Out of box with TopLink
  - Out of box with Hibernate
  - Simple JDBC

Oracle Coherence: Persistence Integration

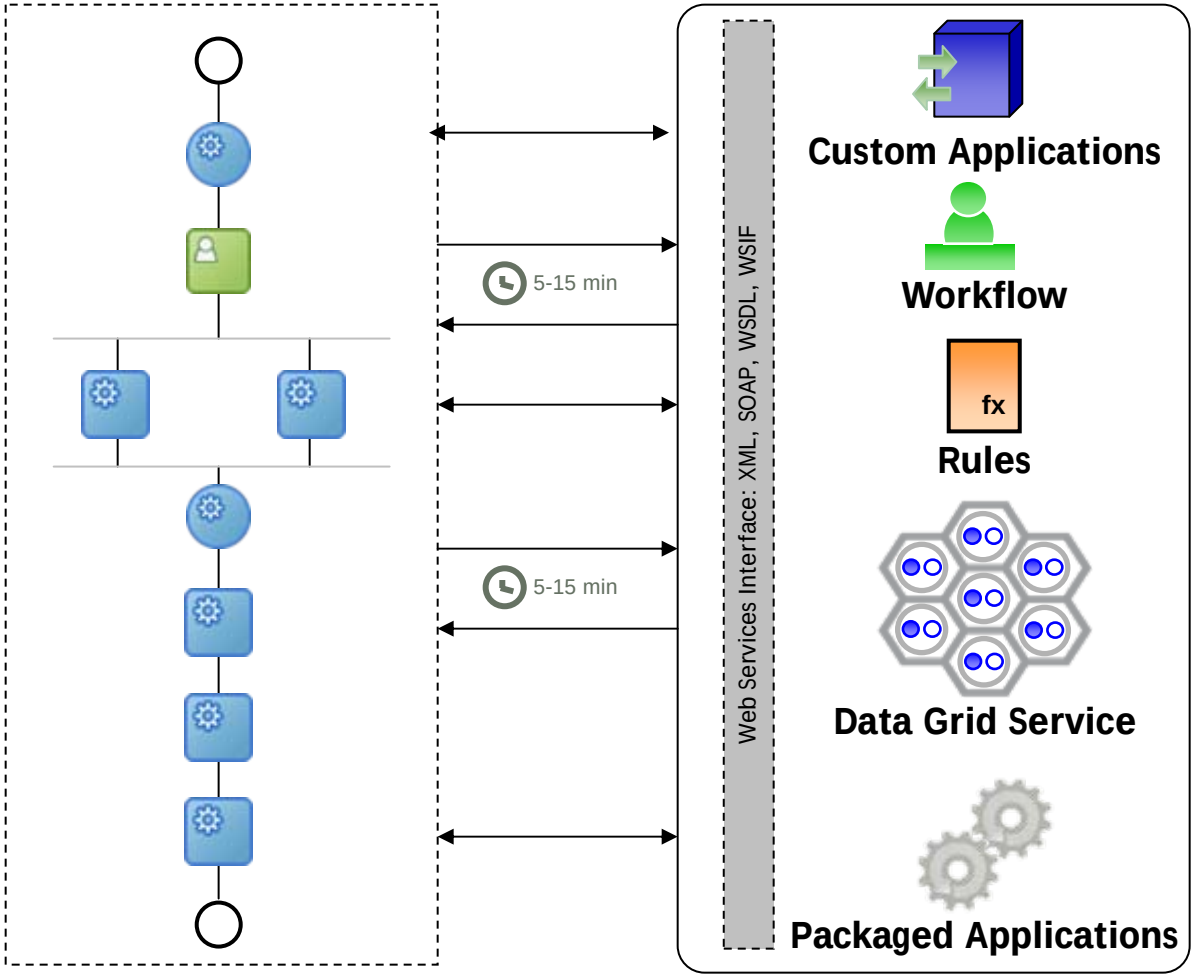
- Built in support for read-through and write-through caching
- Built in support refresh-ahead and write-behind caching
- Support for TopLink, Hibernate, JDO and custom



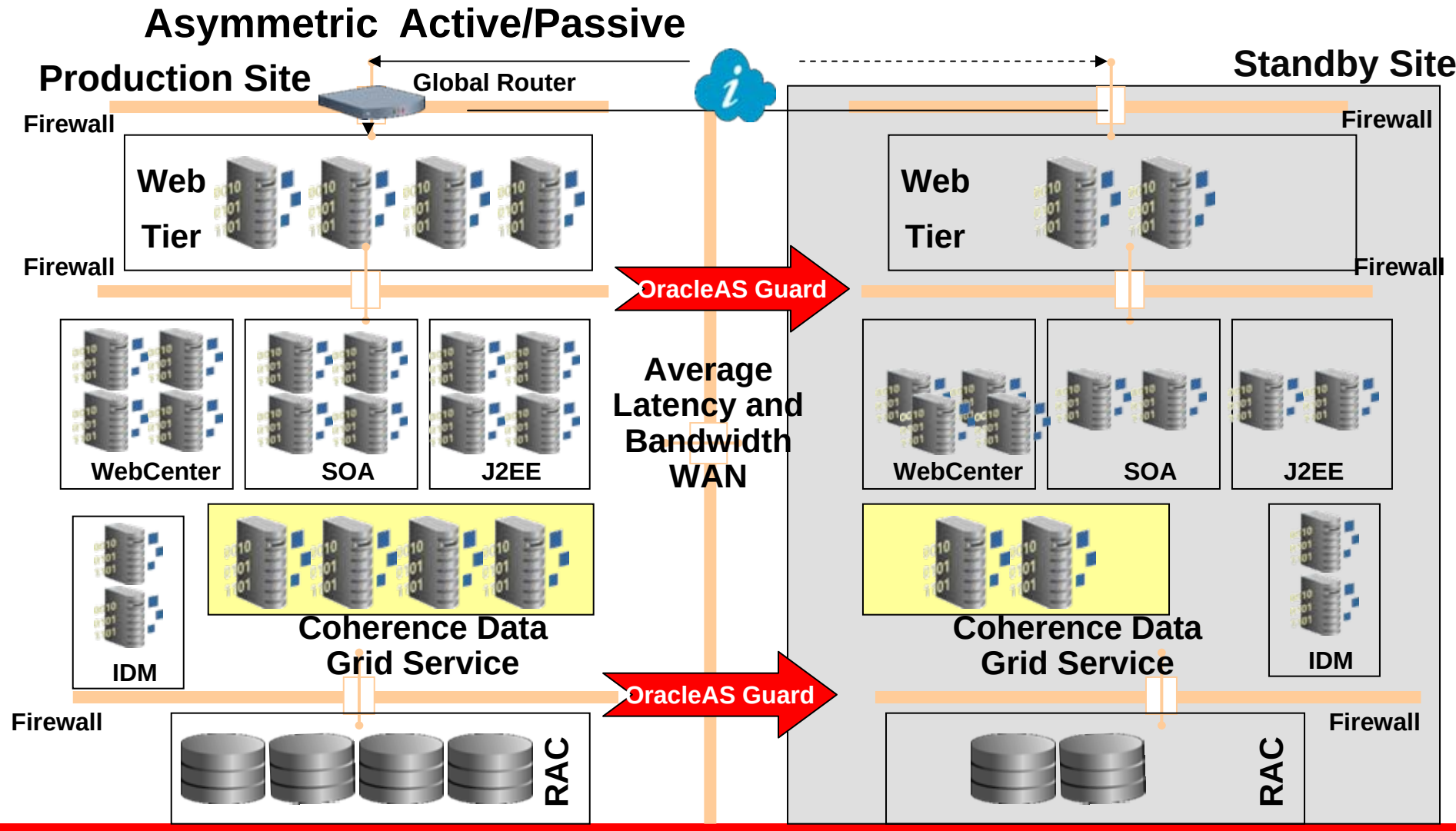
SOA Integration

- Coherence is a shared in-memory data grid service
  - Has standard client libraries for Java and .NET
- Any Java client can invoke this service
  - Web Center, SOA Suite, EDA Suite …
- BPEL and ESB can invoke Coherence via a WSIF bindings
  - This is custom development effort

Oracle Coherence: SOA Integration



Maximum Availability Architecture



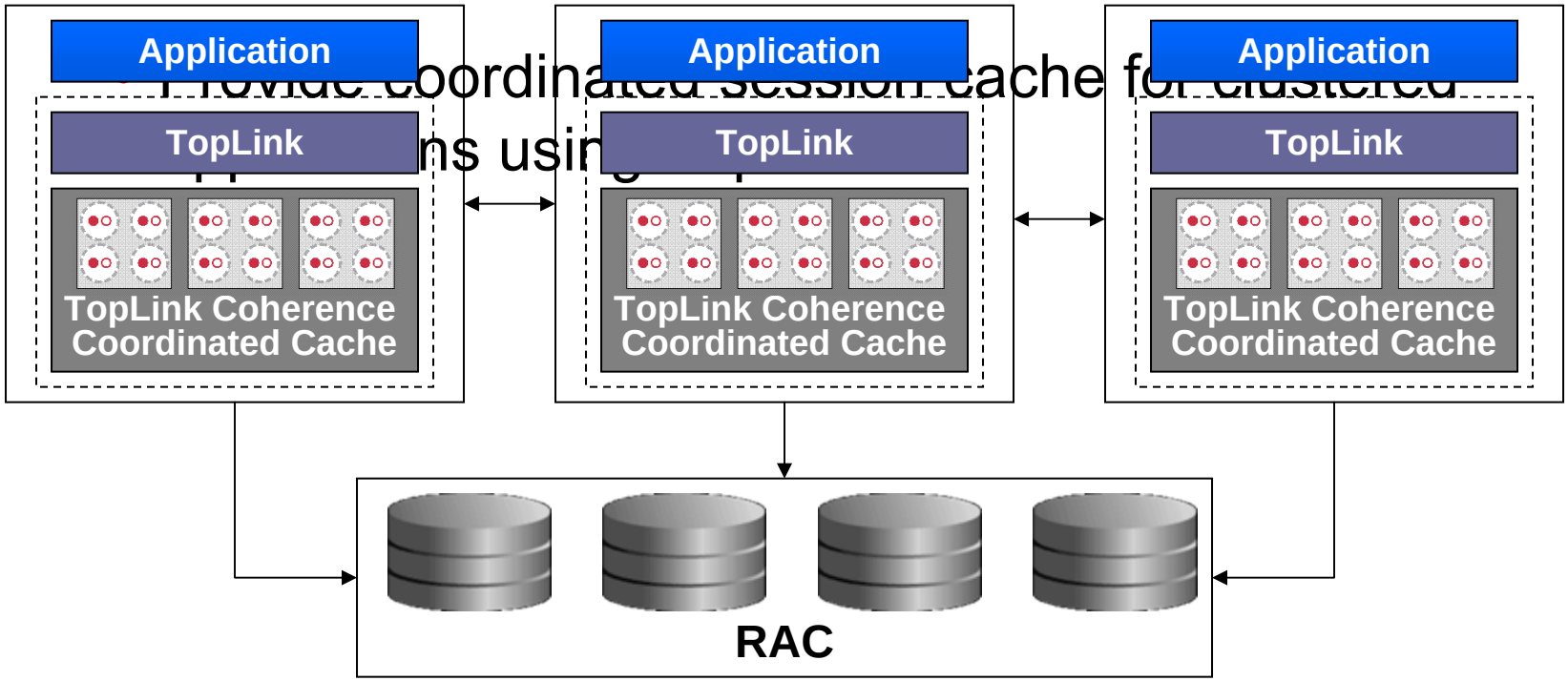
Future Fusion Middleware Integration Points

Future Integration Points

- TopLink Cache Coordination
- OC4J JMS Clustering
- BPEL Clustered In-Memory Dehydration
- Portal Session Sharing
- Maximum Availability Architecture
- Oracle BAM
- Oracle Service Delivery Platform

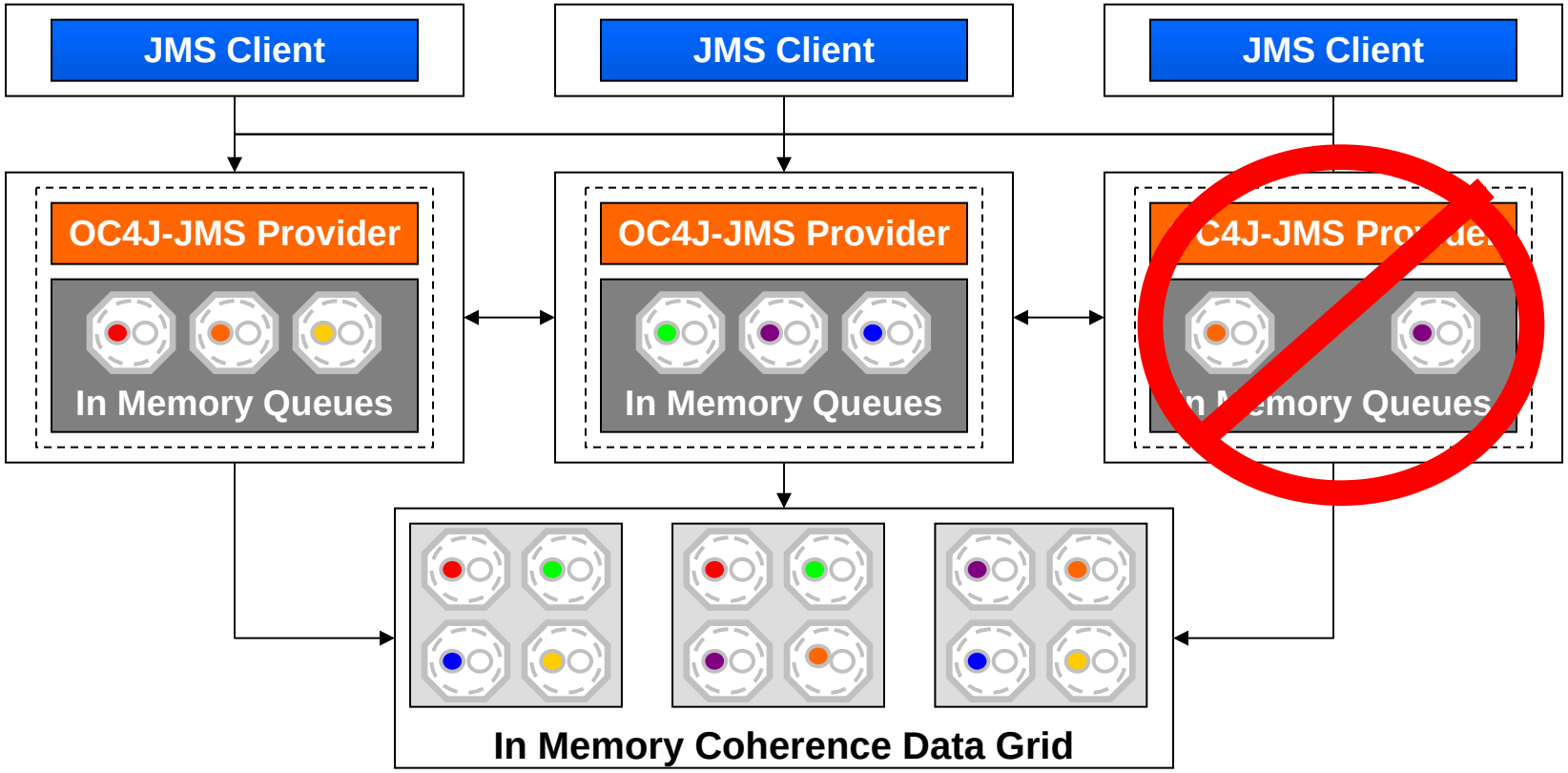
These are proposed projects

Cache Coordination for TopLink with Oracle Coherence



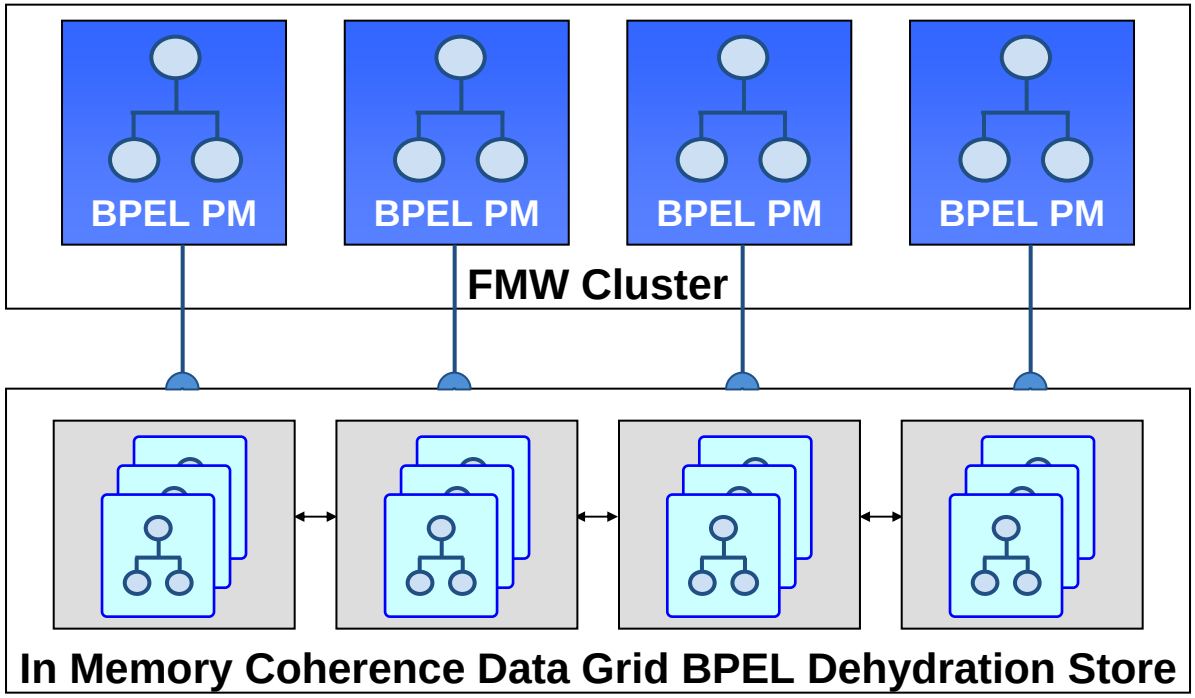
Oracle Application Server JMS Clustering with Coherence

- **Provide reliable clustered in-memory JMS infrastructure**

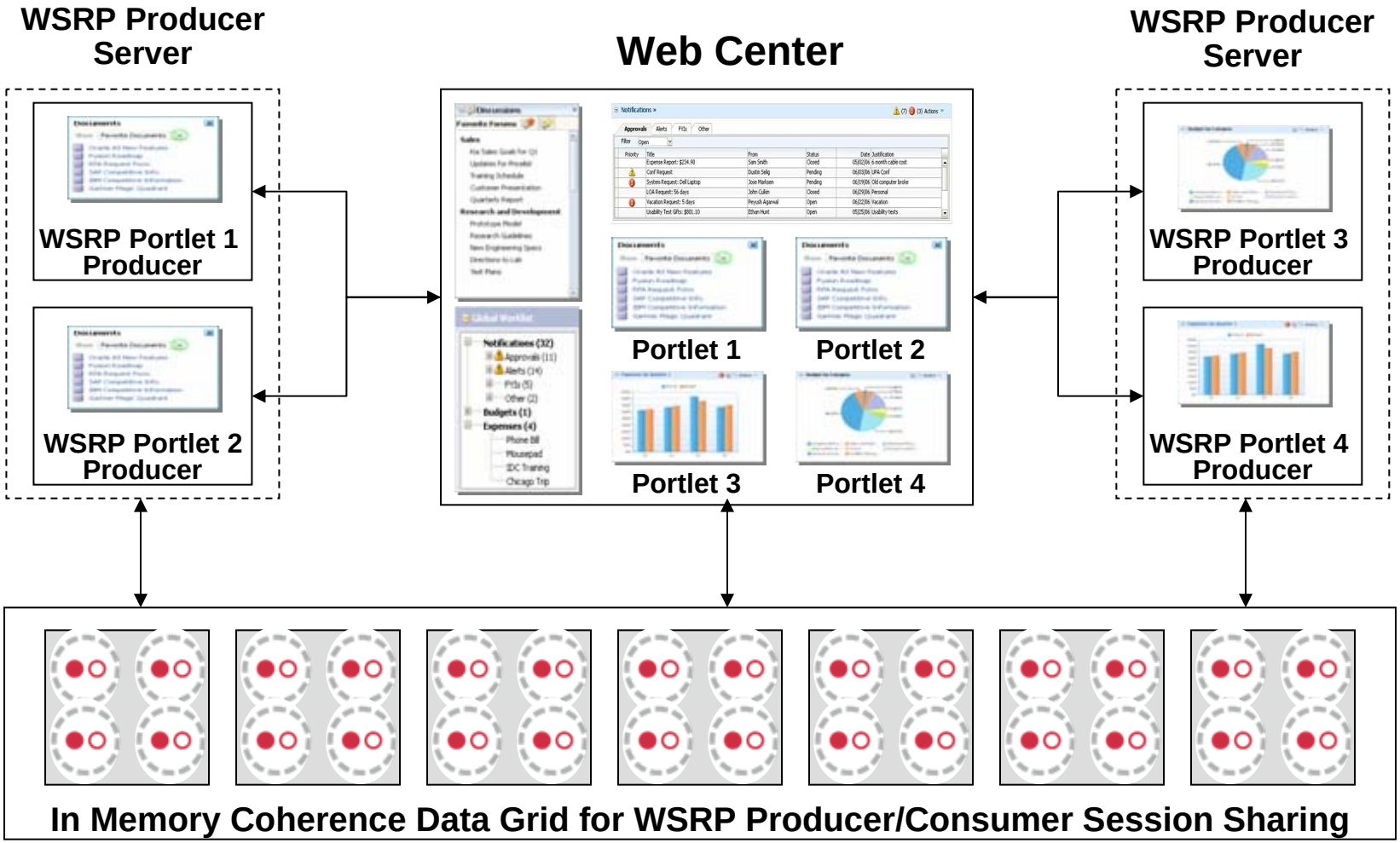


Accelerating BPEL Performance with Coherence

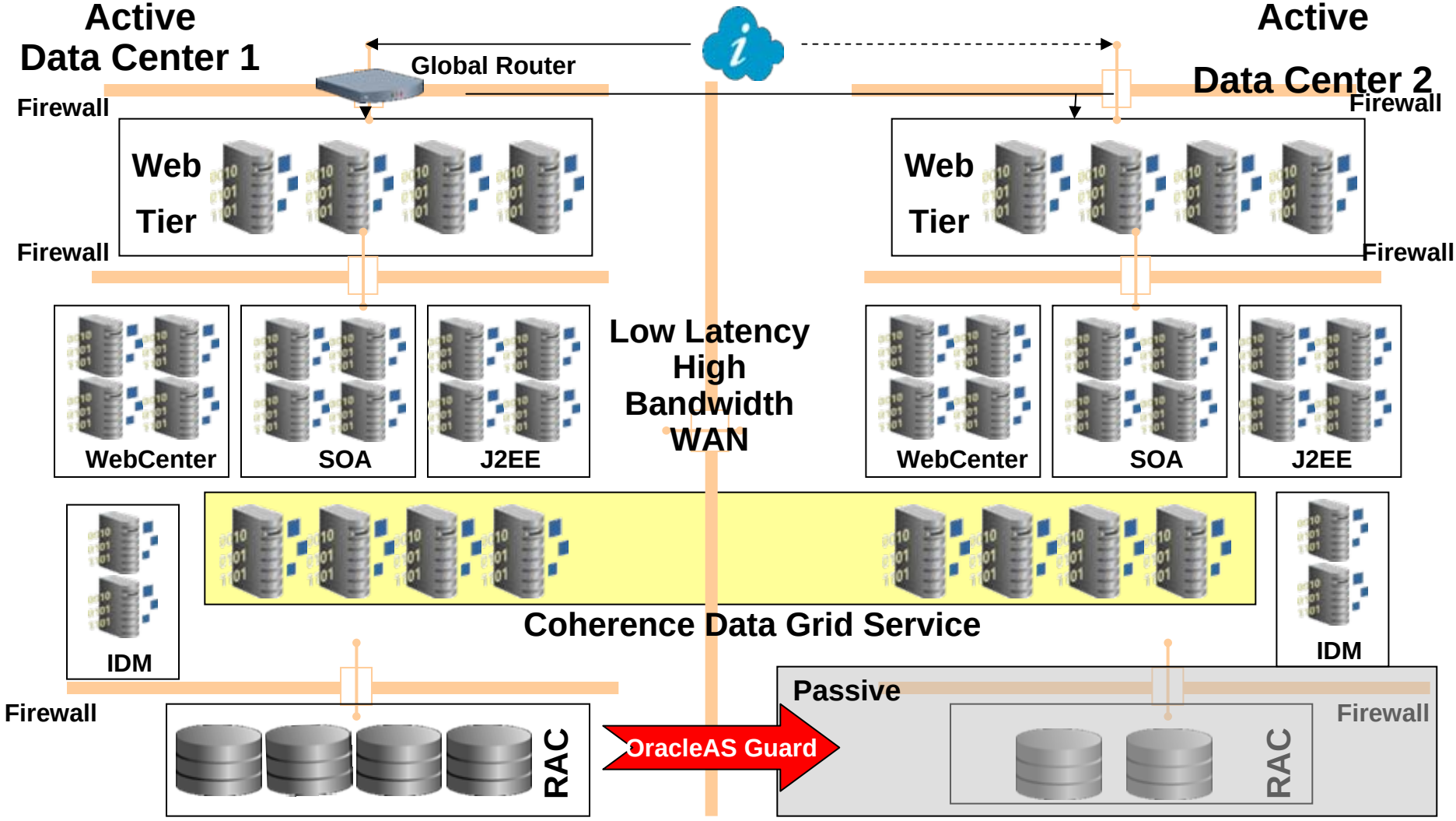
- Extreme BPEL performance using in memory clustered Coherence for dehydration store



Oracle Web Center Portlet Session Sharing



Maximum Availability Architecture
Active/Active



Database Positioning

Database Integration

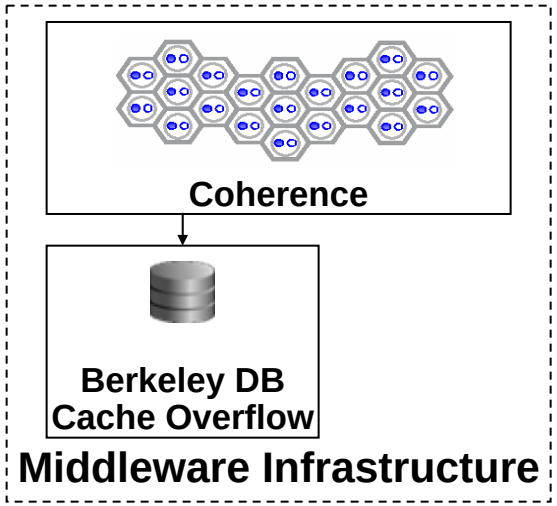
- Oracle Database and RAC
  - Middleware applications using Coherence require high QoS persistence – Oracle RAC
- Berkeley Database
  - Provides disk based cache overflow for Coherence
- TimesTen Database
  - Planned utilization of Coherence clustering technology

Oracle DBMS, TimesTen, Berkeley

Natural Integration Points

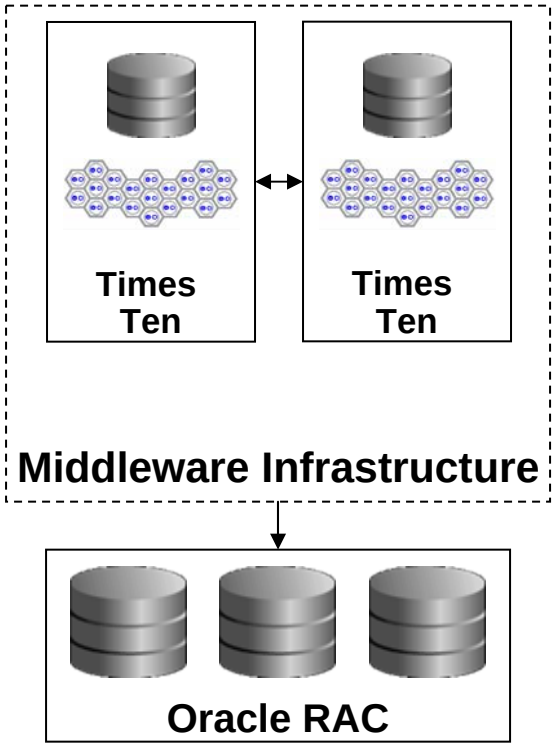
Berkeley DB

Cache Overflow Integration with Coherence



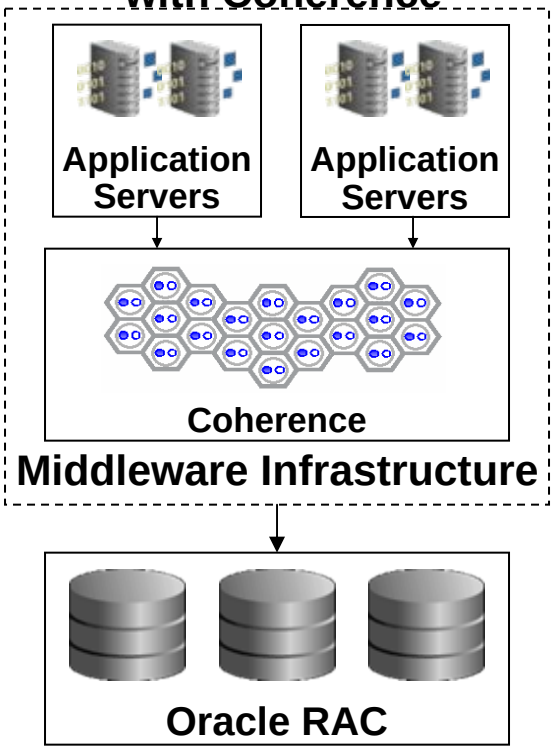
TimesTen

Clustered Caching with Coherence

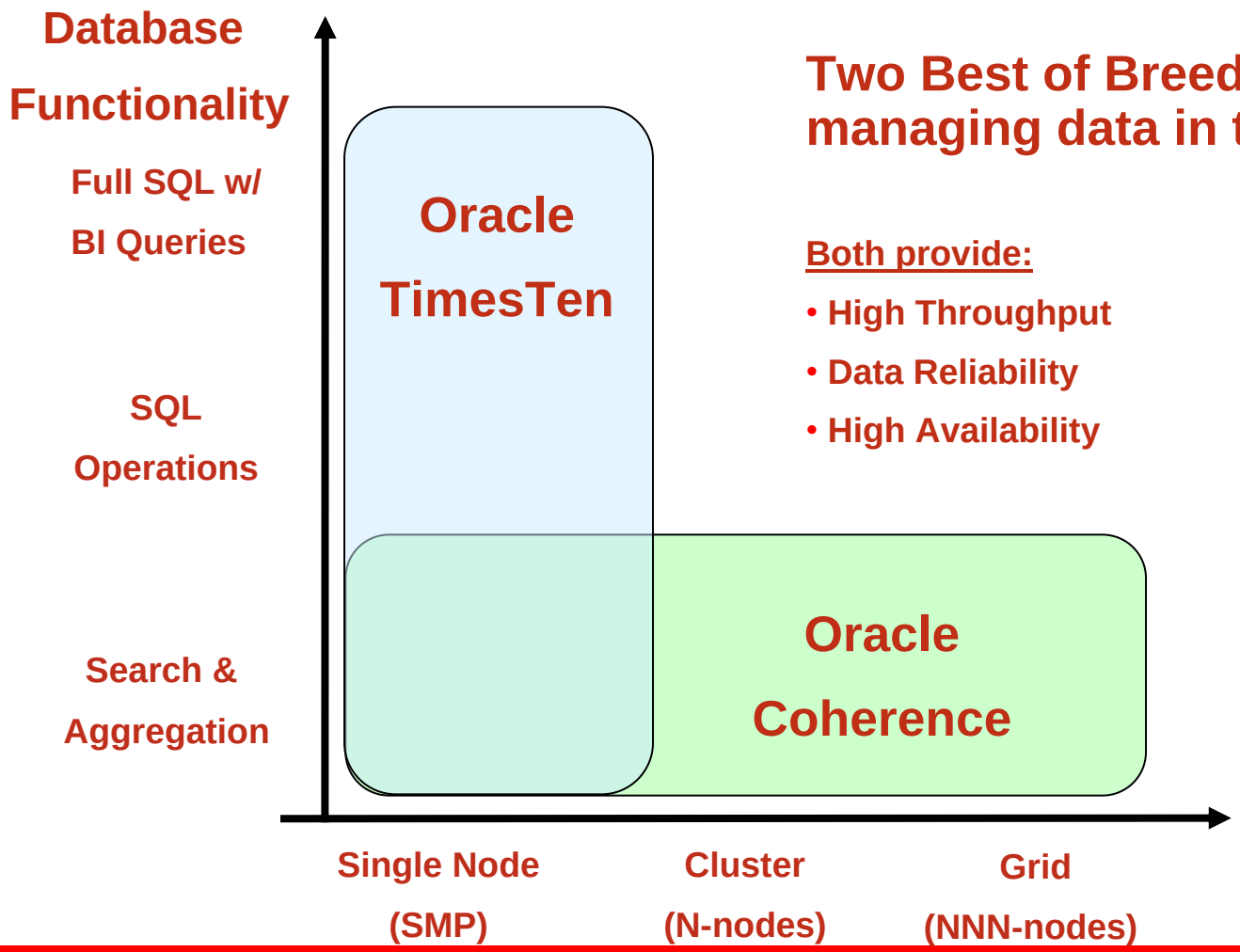


Oracle RAC

Persistence QoS with Coherence



Coherence and TimesTen



Two Best of Breed Solutions for managing data in the middle tier

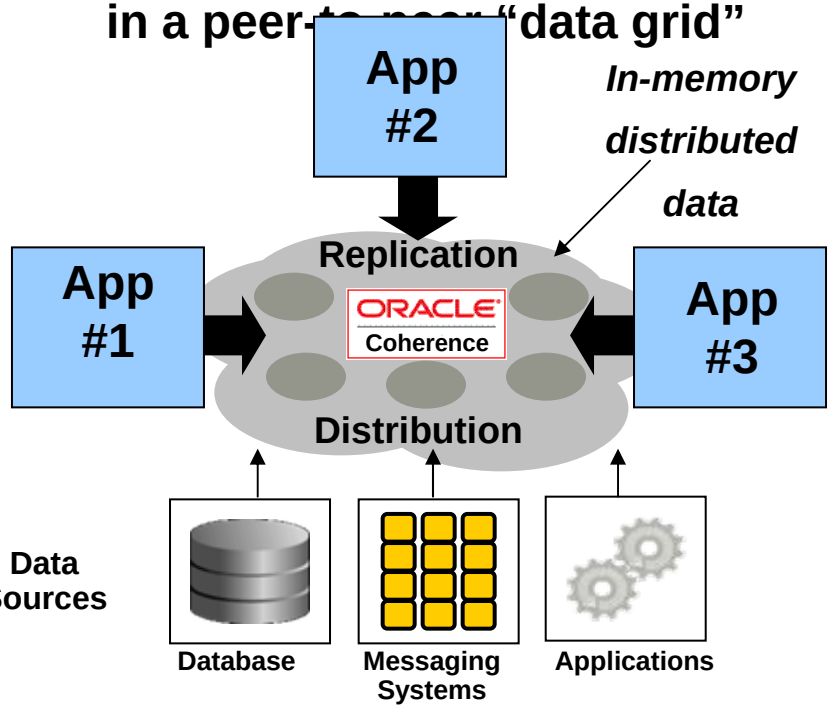
Both provide:

- High Throughput
- Data Reliability
- High Availability

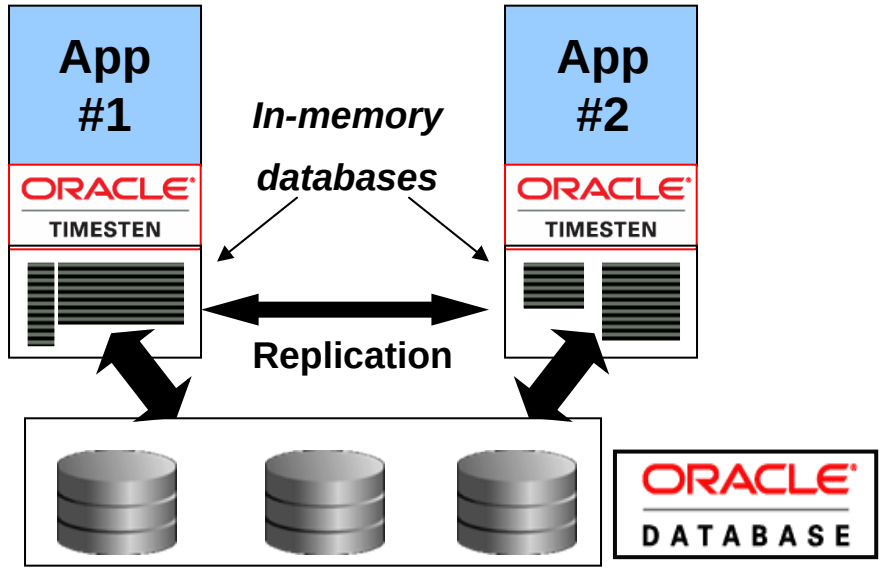
Real-Time Data in the Middle Tier

Oracle's products offer complementary approaches to staging data in the middle tier for high performance

Oracle Coherence provides shared access to distributed data in a peer-to-peer "data grid"



Oracle TimesTen provides a relational database cache for data shared via Oracle database



Coherence and TimesTen

Feature / Capability	Coherence In-Memory Data Grid	TimesTen In-Memory Database Cache
Performance Technique	In-memory distributed data in the middle tier	In-memory relational database / cache in the middle tier
Data Model	Object model (objects/attributes)	Relational database (tables/rows/columns)
Data Access	Standard Languages (Java, .NET, other languages)	Standard API's (ODBC/JDBC) Standard SQL
Data Sources	Database, Message Infrastructure (JMS, AQ), Applications	Database
System Focus & Core Value	Real-time access/transactions against shared, distributed in-memory data	Real-time access to mid-tier data or cached database tables
System Scope & Scale	Large, multi-node grid	Group of replicated servers
Query Capability	Parallel filters over data	SQL, including BI queries & joins
Database Integration	Via object/relational mapping (Toplink/Hibernate) or JDBC	Built-in caching to Oracle database & Oracle RAC

Coherence and other Oracle Products (summary)

How does Coherence compare to other Oracle products?

- Oracle RAC
  - Scale-out database server
- Oracle TimesTen
  - High-performance in-memory database
- Oracle Caching Solutions
  - Oracle Web Cache for content
  - Oracle Java Object Cache (JOC) for Java objects
- Oracle TopLink
  - Object-Relational Mapping solution
- Oracle OC4J / SOA
  - Oracle Application Server

Coherence and RAC

- RAC is a database
  - Scale-out persistence (storage to disk)
  - Ad hoc query support
  - Mature transactional engine
  - Incredible 3rd party support
- Coherence is a data grid
  - Application works with data in Object form
  - Brings data management to the application tier
  - Explicit control over data management results in higher scalability
  - In-memory access for better performance

Coherence and TimesTen

- Coherence
  - “Scale-out object data management”
- TimesTen
  - “Ultra-fast relational database”
  - Native relational view of data
  - Minimizes impact on the application
  - Single-server queries have more predictable performance
  - Ad hoc query support

Coherence and Oracle Caching Solutions

- Minimal overlap
- Oracle Web Cache
  - Content Caching
- Oracle Java Object Cache
  - Read Caching of Java Objects
- Oracle Coherence
  - Read Caching (typically high-scale and/or high-contention)
  - Scaling out Stateful Applications
  - In-memory transactions

Coherence and TopLink

- TopLink
  - Ability to manage complex data models
    - Hundreds of tables (types of entities)
  - Support for read caching only
  - Scalability ultimately determined by the underlying database
- Coherence
  - Difficult to manage complex data models
    - Typically fewer than 50 tables (types of entities)
  - Able to do all data management in-memory
  - Ability to scale independently of the underlying database

Coherence and OC4J/SOA

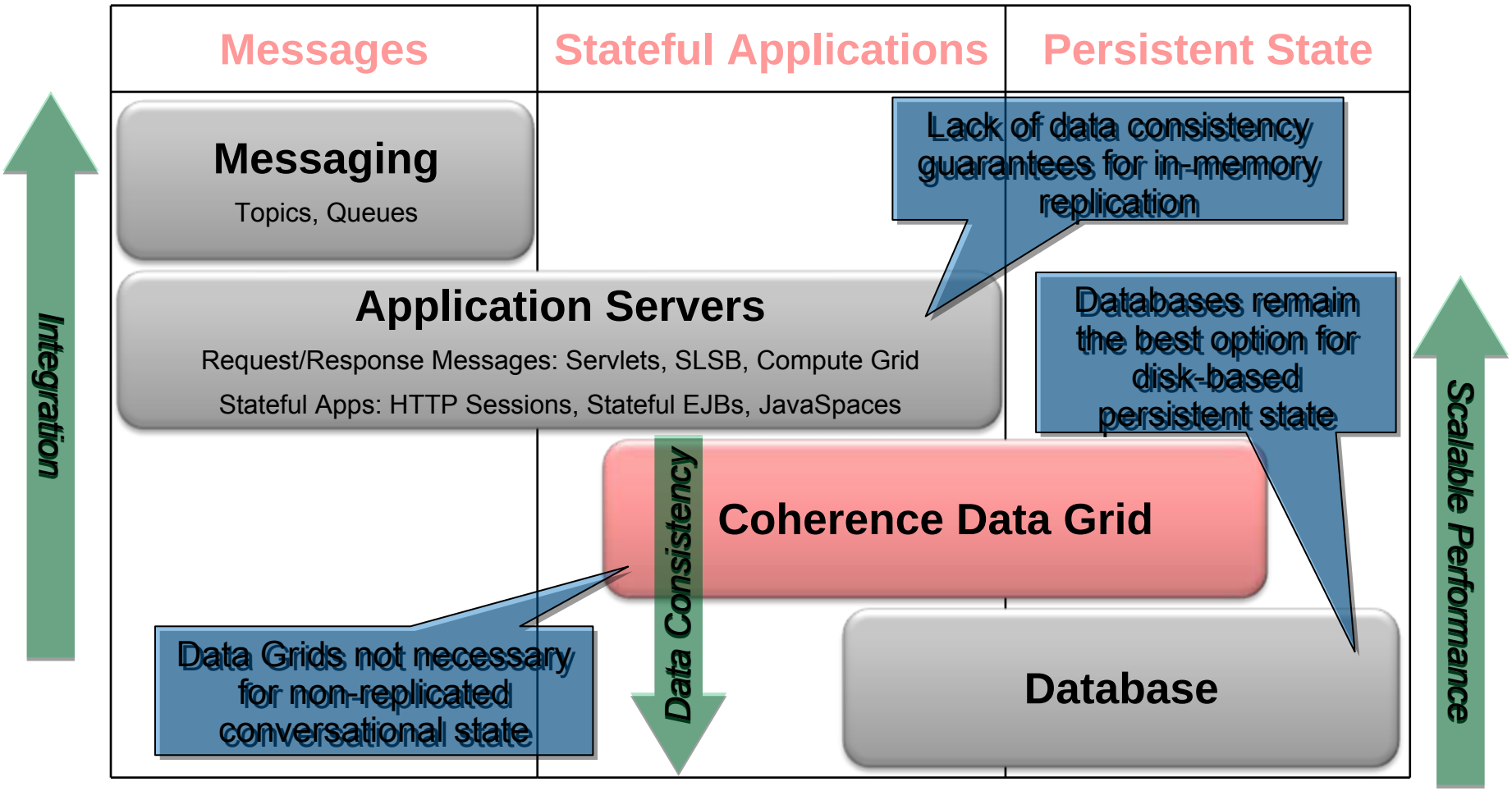
- Coherence*Web already provides high-scale HTTP session management for OC4J
  - Future integration plans are in the works
- Coherence can be used to implement SOA
  - Requirements for massive scalability and availability
- Coherence is not SOA/EAI
  - Not ESB solution (Policy enforcement, metadata management, etc)
  - Not ETL solution (Data integration, format translation, etc)

Long-Term Positioning

- Oracle Web Cache
  - No overlap
- Oracle Java Object Cache
  - Always lead with Coherence
  - Coherence will eventually supplant JOC
  - No timeline has been determined
- TimesTen
  - In-memory relational data management
- RAC
  - Scale-out relational data management
- Coherence
  - Scale-out data management in the application tier
  - Grid-enable Fusion Middleware

Solutions Architecture Directions

The Spectrum of Solutions Architecture



Types of Coherence Adoption

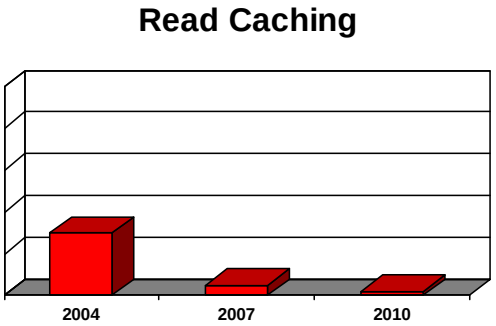
- Read Caching
  - Read data from Coherence instead of a backend data source
- Scaling Stateful Applications
  - Scale-out of stateful applications (HTTP sessions, stateful EJBs, etc.)
- Extreme Transaction Processing (XTP)
  - Partially (or fully) offload transaction processing from a database by managing data in the data grid
  - Provide real time visibility into these transaction via event processing

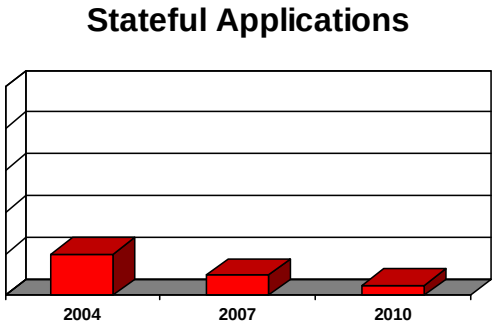
Coherence Value

More reliable than application servers	Read Caching
	Stateful Applications
Better scalable performance than databases	XTP • Query and Analytics • Concurrency Control • Persist in-memory or in-database • Real-time events

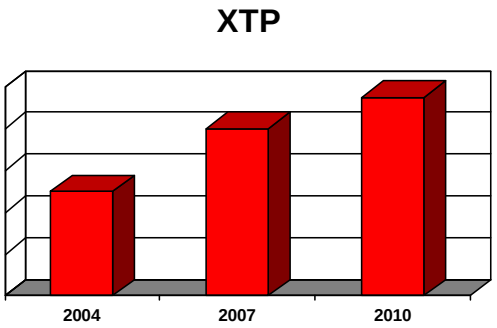
Customer Use Cases are Shifting

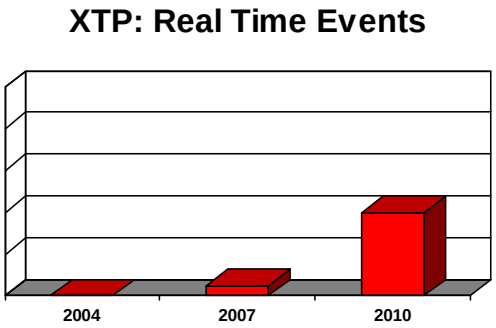
- Read Caching and Stateful Applications are less critical





- XTP workloads are driving core Coherence sales
  - Real time event processing is starting to gain importance





Identifying Opportunities

Identifying Opportunities

Degrees of Buy-In

Implications on Lead Qualification

- Read Caching
  - Only very high scale applications require Coherence
  - Smaller opportunities were generally not pursued at Tangosol
  - Indicator: Customers buy Coherence for large scale or to mitigate risk
- Stateful Applications
  - In most (but not all) cases, limited revenue opportunity
  - Indicator: Massive pain with current session management

Implications on Lead Qualification

- XTP
  - Primary revenue stream for Coherence (>90% of customer value?)
  - Indicator: Customer can’t (or won’t) achieve required throughput or performance with database technologies

Implications on Lead Qualification

- Buy in at bottom end (caching), work up (to XTP)
- Customers with high-value requirements
  - Buy higher-end Coherence editions
  - Often start off conservatively (e.g. read caching) to gain production experience with Coherence (if they have the luxury of time)
  - Then move on to more advanced Coherence functionality

Implications on Lead Qualification

- Customers with low-value requirements
  - Buy low-end Coherence (Standard Edition)
  - Often come to Coherence for read caching or for help scaling stateful applications
  - Over time start to use more advanced Coherence functionality
  - Eventually upgrade to Enterprise Edition or Grid Edition

Implications on Lead Qualification

- Due to very restricted sales and support resources …
  - Tangosol was extremely careful with lead qualification
  - Focus almost exclusively on high-value sales
- Oracle impact
  - Low-value deals may become less expensive to pursue
    - Economies of scale
  - Low-value deals are also less expensive sales
    - Easier implementation
    - Easier budget approval
  - Low-value deals may outnumber high-value deals

Identifying Opportunities

Read Caching

Read Caching

- Traditional use of the term “caching”
  - This is the only form of caching that can be truly “transparent”
- Use: Presentation layer
  - (Almost) all content data is stale by definition
- Use: Optimistic Concurrency
  - Update database if the cached value is still correct

Presentation Layer

- Cached data
  - Almost never static data
  - Cached pages or other content
  - Inventory levels, current prices (e.g. 60-second freshness)
- Benefits
  - Application instances can start up rapidly without crushing database
  - Easy to manage huge (100GB+) data sets
  - Avoid long GC pauses in application servers
  - Cycle application servers without flushing cache
  - Resistance to unintentional Denial of Service attacks
    - Thousands of users clicking repeatedly to see the current score of a sporting event, or the current price of an auction
    - Automated “bots” used to monitor and act on websites
- Content caching is usually a low-value use of Coherence
  - Often using Coherence for ease-of-use and stability under load

Optimistic Transactions

- Caching and Optimistic Transactions
  - Database accessed in READ_COMMITTED mode
  - Cache acts as READ_COMMITTED as well
  - Updates to the database use optimistic commit pattern
    - `Update table set X = :newValue if X = :oldValue`
- Use Coherence as a plug-in
  - TopLink, Hibernate, Apache OpenJPA, BEA WebLogic Portal Server, etc.

Read Caching Assumptions

- Very minimal requirements
  - Coherence does more than required
  - And can't be detuned (other than disabling partition backups)
- Cache data is a subset of the database
  - Always recoverable
- Cache data is accessed as READ_COMMITTED
  - Only requirement is a cached value must have existed in the database at some point in the past
  - Strictly speaking, no requirement for concurrency control in the cache
- Cache access is usually identity-based (e.g. primary key)
  - Coherence supports queries, but the application needs to

Qualify In: Read Caching

- Slow startup of servers
  - Cache once in Coherence, then load from there
  - Servers can come and go with no impact on database
- Caching issues with ORM
  - Hibernate, TopLink, OpenJPA, etc.
  - Built-in caches are not fully coherent or very scalable
- Very large caches (more than can comfortably fit in a single JVM)
  - Cache in Coherence to avoid garbage collection pauses
- Large, high-load deployments
  - Use Coherence to provide stability under load

Qualify Out: Read Caching

- Most read caching use cases do not require Coherence
- If the application meets all of these requirements:
  - Runs on a small cluster (<4 servers)
  - Is very read-heavy
  - Needs only a small amount of cached data
  - Can tolerate slightly stale data
  - Is not running under heavy load
- … then it probably does not need Coherence
- Fortunately, there are still hundreds (thousands?) of applications that do not meet these requirements

Identifying Opportunities

Stateful Applications

Stateful Applications

- Stateful applications manage data that
  - Exists beyond the scope of a single request
  - Is not externally visible (stored to database, sent via message queue)
- Coherence supports
  - Generic application state
  - Spring integration (“Spring Beans”)
  - HTTP sessions via Coherence*Web

Stateful Applications

- Coherence does not support
  - Stateful EJB
    - Can use Stateless Session Bean to manage clustered state (more scalable)
  - JavaSpaces
    - An interesting niche technology that is completely unsuitable for the mass market

Stateful EJB

- No direct support for Stateful Beans
  - Requires container support
  - Few apps use these anyway due to historic scaling issues
- For EJB-centric applications …
  - Typically use Stateless Session Beans (SLSB)
  - Any inter-request state can be managed in Coherence
  - Persistence via ORM (TopLink, Hibernate, homegrown, etc)

Coherence*Web

- Sessions managed in Coherence cache
  - Pluggable session models provide mapping
  - Supports database sync, query, etc (with some effort)
- Fully coherent
  - No data corruption under load or rebalancing
- Scalability to hundreds of nodes
  - Near cache with “present” invalidation strategy
  - “Sticky” session locking

Coherence*Web

- Common session data format
  - Support for most web containers
  - Share across different web containers (OC4J, WebLogic, etc)
  - Share across different web applications in single container
  - Share across multiple web sites (e.g. clothing or automotive brands)
- Support for huge sessions
  - 1MB+
- No requirement for sticky load balancer

Coherence*Web

- Two-step install
  - Coherence*Web analyzes web application (.EAR, .WAR or exploded)
  - Coherence*Web instruments the web application
- More details
  - http://dev2dev.bea.com/pub/a/2005/05/session_management.html

Identifying Opportunities

XTP

Varying Levels of Commitment

- Level “Zero” is Read Caching
- Level One
  - Concurrency control (locking)
- Level Two
  - Real Time queries and analytical processing
- Level Three
  - Commit transactions to Coherence
- Level Four
  - Reacting to real time events

* Not a strict hierarchy, just a common pattern of adoption

Concurrency Control

- Scalable distributed locking engine
  - Perform all locking against cached data
  - Perform all reads against cached data
  - Commit changes to both cache and database
- Predictable performance under load
  - No unexpected rollbacks or conflicts
    - If the data grid is the only system updating the database*
      - All updates through the data grid make this a degree of buy-in

* Still useful even with a shared database (though less so depending on number of conflicting updates)

Queries and Analytics

- Queries
  - Programmatic API (SQL-style queries)
  - Filter, aggregate, sort … but no joins
  - Dealer.com *"Show all 4-door BMW sedans built after 1995"*
- Analytics
  - Coherence provides the scalable performance
    - Analytical processing logic is up to the developer
    - Very extensible yet simple programming model
  - Processed in parallel across the entire cluster
- Same underlying technology for both

Queries and Analytics

- Throughput
  - Process lots of data … in a reasonable amount of time
  - *100 requests/sec, 1MB/request, <200ms/request (risk calculations)*
  - *10,000 requests/sec, 10KB/request, <1 sec/request (inventory queries)*
- Real Time
  - Process a reasonable amount of data … immediately
  - *1,000 requests/sec, 10KB/request, <10ms/request (algorithmic trading)*

Simple Queries

- Reasons to not use Coherence for queries
  - Not intended for ad hoc querying
  - Not optimized for query
    - A disk-based database may be faster for some types of queries
  - Latency may increase with larger cluster sizes
    - “Slowest responder” problem exacerbated by Java GC pauses in clusters with hundreds of nodes

Simple Queries

- Reasons to use Coherence
  - Data is already managed by Coherence
  - Many simple queries (e.g. hundreds) per user interaction
  - Getting queries off of overloaded database infrastructure
  - More predictable performance in production

Custom Analytics

- General Approach
  - Send sub-calculations out to the cache servers
  - Compute sub-calculations against local data partitions
  - Bring sub-results back to the client and apply final processing
- Requirements
  - Sub-results must be much smaller than the local data partitions
  - The compute (CPU) on each node should be greater than the cost of sending the request to the node (network)
  - In most cases, the data set should be large enough to justify spreading it across multiple nodes

Write-Through Transactions

- Coherence sets up the transactions
  - All reads and queries, all concurrency control
  - Possibly even constraint verification
- Commit to the database
  - Regular database transactions

Write-Through Transactions

- Benefits
  - Substantial offload of database
  - Application has more control over concurrency operations
  - Database acts as a normal System of Record
- Drawbacks
  - Some application code changes required
  - Coherence limitations affect application design

Write-Behind Transactions

- Everything in Coherence
  - All reads and queries
  - All concurrency control
  - All constraint verification
  - Even Commits

Write-Behind Transactions

- Persist to database at *some* point in the future
  - Bundling results in a few large database transactions
  - Reduce database load by (e.g.) 100x
- Longer persistence delays
  - Increase risk exposure (“What happens if the whole cluster fails?”)
  - But reduce load on database
  - Even a small delay (sub-second) can have huge benefits

Real Time Events

- Maintain real time visibility into data changes
- Desktops
  - The usual example is the “Trader desktop”
  - Watch data change in near real time
    - Typically a few milliseconds
- Servers
  - Monitoring data to trigger additional processing
    - Event Driven Architecture within the data grid
  - Very wide-ranging set of use cases
  - Not many common patterns of usage

Competitive Analysis

Competition by Usage Patterns

	GigaSpaces	Gemstone Gemfire	IBM ObjectGrid	Terracotta (DSO)
Read Caching	Yes	Yes	Yes	Yes
Stateful Applications	Yes (incl. JavaSpaces)	Yes	Yes	Yes
Transactions • Query and Analytics • Concurrency Control • In-memory or in-database • Real-time events	Reliability issues Development challenges	Scalability issues Complexity issues	We have very limited knowledge of ObjectGrid	No

Competitive Overview

Need to eliminate “half-truths”

- Reliability shell game: don’t worry “its magic”
- We can scale to the moon: “show me”
- “Datagriditis”: if I call it a data grid, it makes it one

Competitive Overview

- Need to educate the consumer!
  - Don’t assume anything!
- Better competitive differentiation earlier in the sales cycle
- Gain clarity of “mission critical-ness” of opportunity
- Expose their weaknesses and ensure they are part of the POC

Key Coherence Differentiators

- Technology
  - Reliability (esp. data consistency)
  - Scalability
  - Ease of use
- Adoption
  - Largest Deployments
  - Largest Direct Installed Base
  - Largest Indirect Installed Base
- Partnerships
  - Provisioning: DataSynapse, Platform
  - Influencers: Oracle, Intel, IBM, BEA

Competitive Field vs Coherence

	Gigaspaces	Gemstone	Terracotta	IBM
Positioning	"Jack of all trades" -- we consistently win on data grid	Legacy vendor, all custom sales, don't see them much	Hub-and-spoke, poorly defined HA, almost no production installs	New entrant, immature product
Presence	All but the smallest engagements	Elephant hunters: large, complex	Minimal, but always the "free" option	Minimal
Company Revenue	~ $10mm	~ $10mm	~ $0	
Product Revenue	Similar	Minimal	~ $0	Minimal?
Data Grid Revenue	50%	10%	0%	Minimal?
Level of Desperation?	"We'll give you the software"	"We'll give you the software, but you have to buy consulting"	"Buy consulting"	None

Competitive Field vs Coherence

	Gigaspaces	Gemstone	Terracotta	IBM
Other revenue sources	ROC, Service and Compute Grids	Legacy software (Smalltalk)	n/a	
FUD	Shelfware Not all data grids Significant ROC	Complicated product built as a series of one-offs	Couldn't sell it so made it free	New entrant Lack of internal support
Licensing	Dozen+ licenses	Dozen+ licenses	?	Clean
Glaring weaknesses	Data loss SBA / ROC Significant Operational Cost	No Scalability Hard to use	Non-scalable hub Immature HA	

SWOT: Gigaspaces

Strengths	Opportunities
• JavaSpaces “religion” • SQL/ESB/JMS functionality • C++ today • SLA provisioning, deployment • GUI • Ability to manage market perception	• Address weaknesses
Weaknesses	**Threats**
• JavaSpaces “baggage” • No data consistency guarantees • Non-native .NET/C++ implementation • Complexity • Poor SQL implementation • Third party technology	• Merger with DataSynapse • Intel VC • Major OEM deals • Buy-Out (BEA?)

SWOT: GemStone GemFire

Strengths	Opportunities
• Ability to pop up in large accounts • Feed off of legacy installed base • Production experience • Able/willing to do custom integration to exotic systems • Able to deliver C++ support today • Wealthy new backer	• Address weaknesses • Win at Citigroup
Weaknesses	**Threats**
• Poor scalability • Price • Poor .NET/C++ implementations • Reputation (non-strategic)	• Get acquired • Gemfire architecture is strikingly similar to Coherence

SWOT: IBM ObjectGrid

Strengths	Opportunities
• IBM scale/inertia • Clustering experience from WebSphere XD	• Quickly add additional functionality
Weaknesses	**Threats**
• How dependent is it on WebSphere? • Early days of product (technology and installed base) • IBM sells Hardware and Services, not Software • IBM scale/inertia	• Starts to be pushed widely by IBM

SWOT: Terracotta

Strengths	Opportunities
• Open Source	• Open source mindshare
Weaknesses	**Threats**
• Open sourced because they couldn't sell it • "Transparent" clustering requires massive XML configuration and intensive knowledge of Java Memory Model (JSR 133), and in many cases rewriting the application • Hub-and-spoke architecture	• Adoption by low-end volume market • Many customers don't actually care about data correctness or scalability

Proof of Concepts

PoC (Good for Coherence)

- **Focus on reliability!**
- **Focus on reliability! (again)**
- **.NET**
  - Our implementation is best by large margin
  - Any strategic product should support .NET ;-)
- **5+ physical servers**
  - Gemstone falls apart, GigaSpaces configuration PITA
- **Lots of data**
  - Increased JVM count per machine (harder to scale)
  - Or increased GC pauses (data corruption)

PoC (Good for Coherence)

- **Lots of writers, lots of contention**
  - Data corruption more likely when multiple writers of same data
- **Test multi-server consistency**
  - Operations spanning data on multiple servers
  - Run queries during server failure-failback
  - Multi-server updates in GigaSpaces are distributed txns!

PoC (Bad for Coherence)

- **Complex, replicated object graphs**
  - The one thing Terracotta does well
- **C++**
- **<5 physical servers**
  - Never use 2!
- **Large SMP servers**
  - Favors Gemstone
- **Sun T2000 (Niagara) w/ small datasets**
  - Twofold: We don't use the TCP accelerator …
  - And the poor per-core performance hurts our network speed
  - Becoming less of an issue but full impact unknown at present…

PoC (Bad for Coherence)

- **Disk**
  - Coherence doesn't (directly) support permanent storage
  - Gemstone is very fast on disk
- **Unsupported APIs**
  - Messaging (JMS)
  - JDBC / SQL / Ad hoc query

Future Directions

Integration Plans

- **TopLink**
  - Coherence can use TopLink to access data sources
  - TopLink can use Coherence as an L2 cache
- **Long Term**
  - Merger just completed (officially June 1)
  - Plans are still in progress
  - More details in Q1FY08 (Calendar Q3)

Coherence Roadmap

- **Patching up some weaknesses**
  - Event processing
  - Transaction processing improvements
  - Simplifying no-downtime upgrades
  - C++ client coming end of 2007

- **Longer term**
  - Isolating workloads on a shared data grid
  - Provisioning improvements