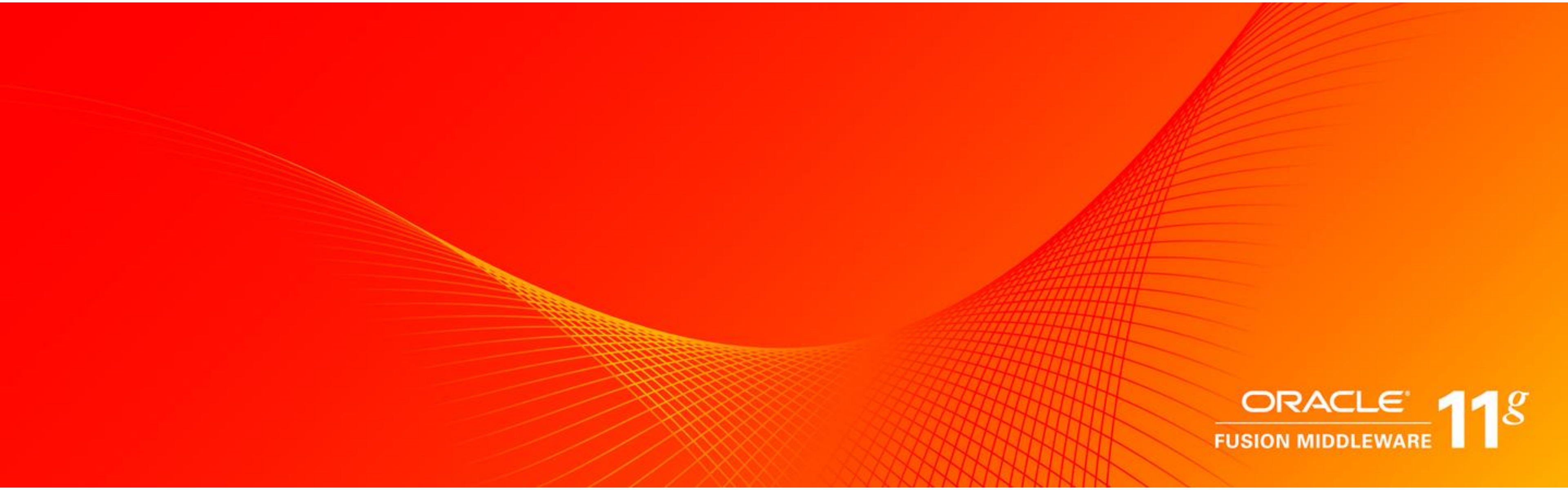
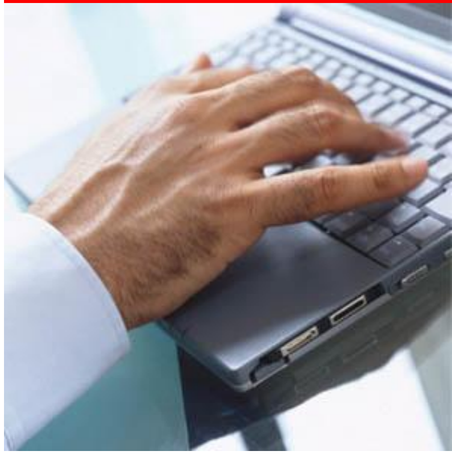


A Java Developer’s Introduction to In-Memory Distributed Computing

James Bayer
Principal Sales Consultant

Agenda



- Defining a Data Grid
- Coherence Clustering
- Data Management Options
- Data Processing Options
- The Coherence Incubator

Defining a Data Grid

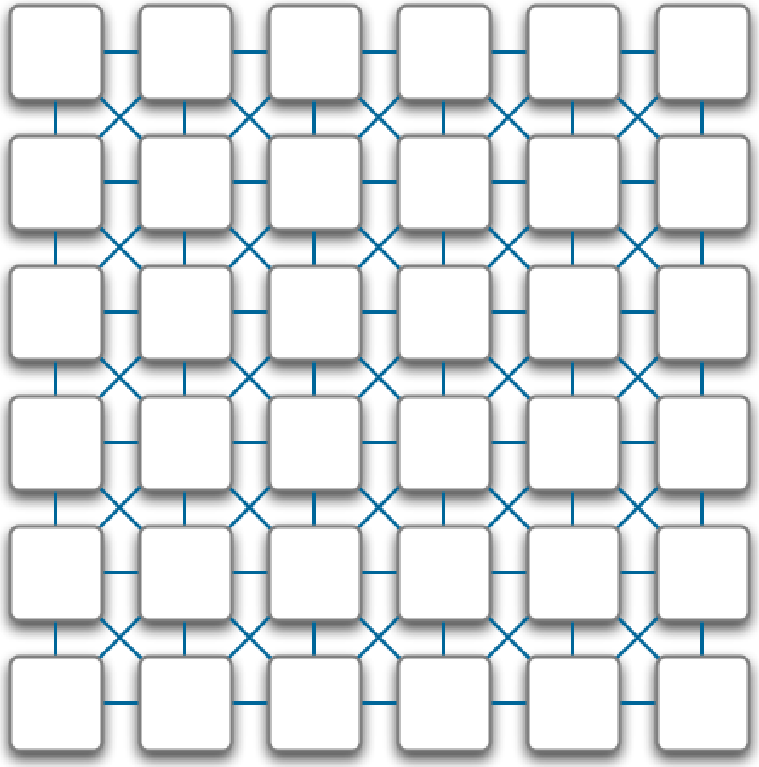
Cameron Purdy
~~CEO, Tangosol, Inc.~~
VP of Development, Oracle

“A **Data Grid** is a system composed of **multiple servers that work together to manage information** and related operations - such as computations - in a **distributed environment**.”

Coherence Clustering

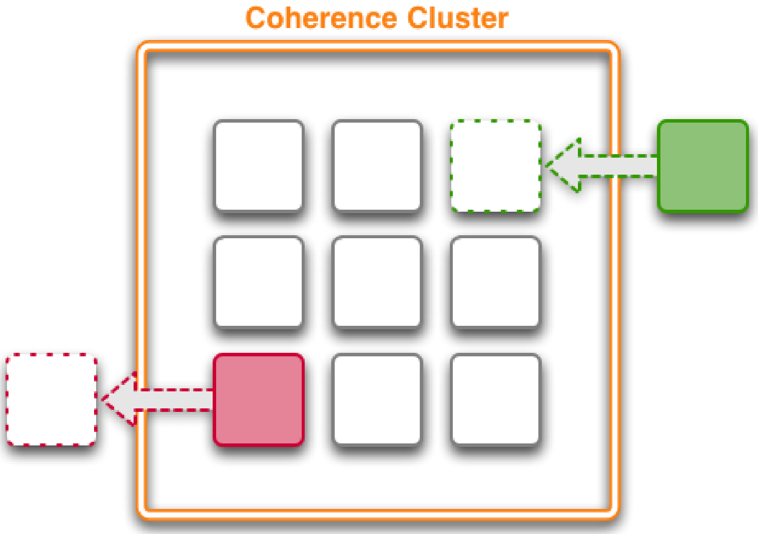
Coherence Clustering:
Tangosol Clustered Messaging Protocol (TCMP)

- Completely **asynchronous** yet **ordered** messaging built on UDP multicast/unicast
- Truly **Peer-to-Peer**: equal responsibility for **both producing and consuming** the services of the cluster
- **Self Healing** - Quorum based diagnostics
- Linearly scalable **mesh architecture**.
- TCP-like features
- Messaging throughput scales to the network infrastructure.



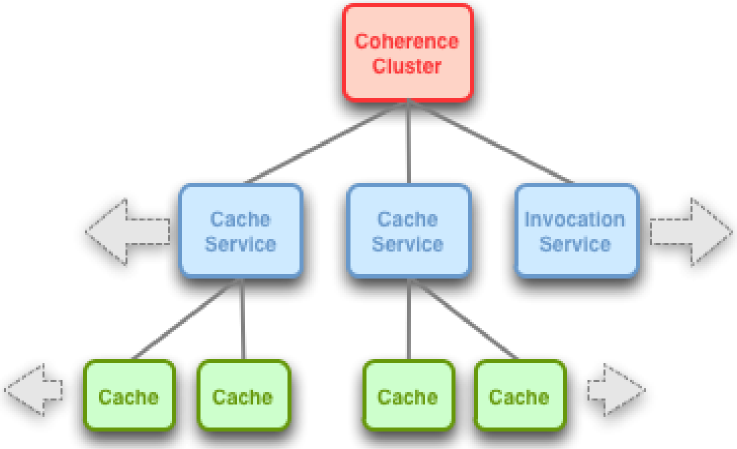
Coherence Clustering:
The Cluster Service

- **Transparent**, **dynamic** and **automatic** cluster membership management
- **Clustered Consensus: All members** in the cluster understand the topology of the **entire grid** at **all times**.
- **Crowdsourced** member **health diagnostics**



Coherence Clustering:
The Coherence Hierarchy

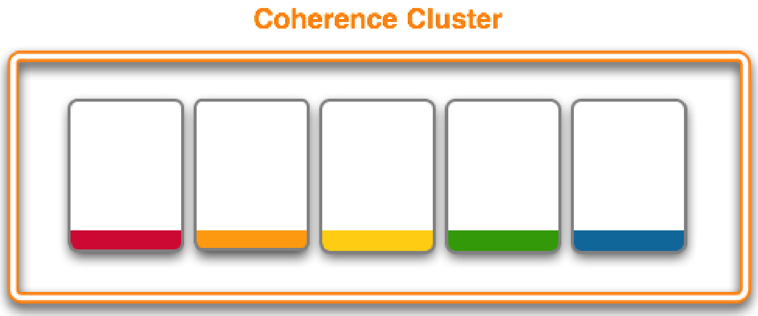
- **One Cluster** (i.e. “singleton”)
- Under the cluster there are **any number of uniquely named Services** (e.g. caching service)
- Underneath each caching service **there are any number of uniquely named Caches**



Data Management Options

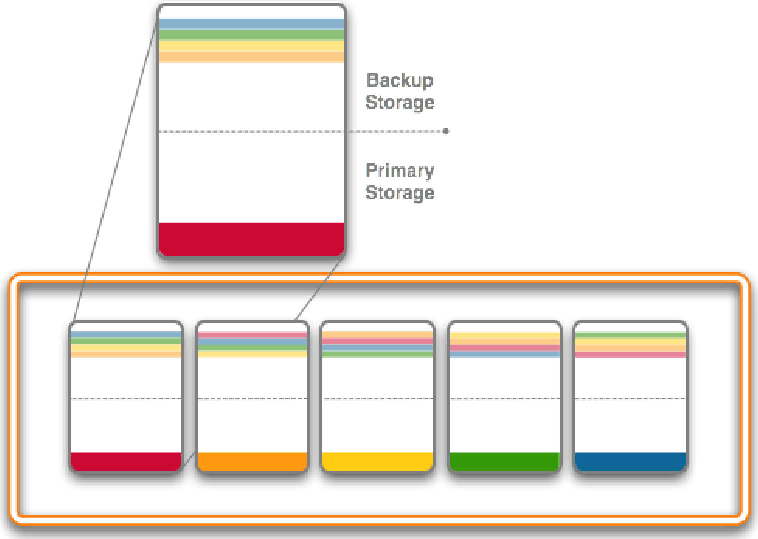
Data Management: Partitioned Caching

- **Extreme Scalability:** Automatically, dynamically and transparently partitions the data set across the members of the grid.
- **Pros:**
  - Linear scalability of data capacity
  - Processing power scales with data capacity.
  - Fixed cost per data access
- **Cons:**
  - **Cost Per Access:** High percentage chance that each data access will go across the wire.
- **Primary Use:**
    - Large in-memory storage environments
    - Parallel processing environments



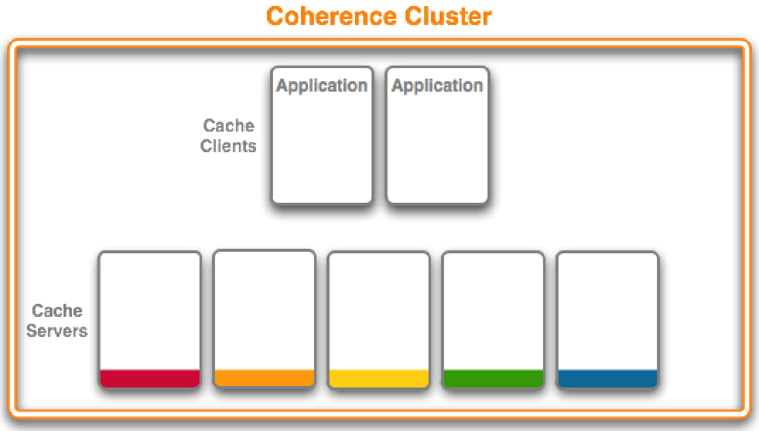
Data Management: Partitioned Fault Tolerance

- **Automatically, dynamically and transparently** manages the **fault tolerance** of your data.
- **Backups are guaranteed** to be on a separate physical machine as the primary.
- Backup responsibilities for one node's data is **shared amongst the other nodes** in the grid.



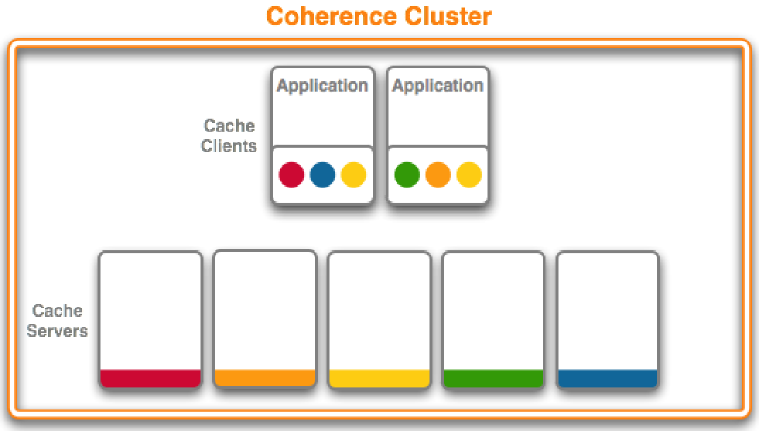
Data Management: Cache Client/Cache Server

- Partitioning can be controlled on a **member by member basis**.
- A **member is either responsible for an equal partition of the data or not** ("storage enabled" vs. "storage disabled")
- **Cache Client** – typically the application instances
- **Cache Servers** – typically stand-alone JVMs responsible for storage and data processing only.



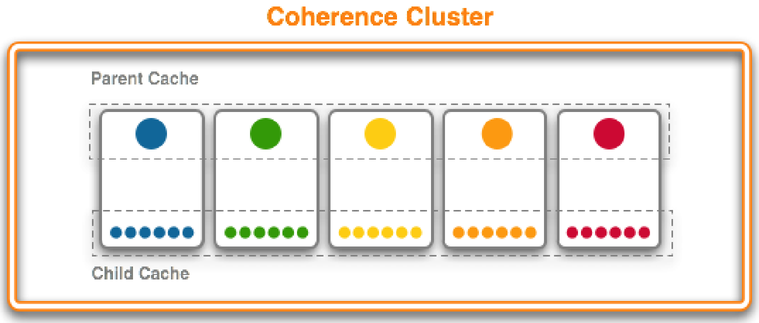
Data Management: Near Caching

- **Extreme Scalability & Performance**
  - The best of both worlds between the Replicated and Partitioned topologies. Most recently/frequently used data is stored locally.
- **Pros:**
  - All of the same Pros as the Partitioned topology plus…
  - High percentage chance data is local to request.
- **Cons:**
  - **Cost Per Update:** There is a cost associated with each update to a piece of data that is stored locally on other nodes.
- **Primary Use:**
  - Large in-memory storage environments with likelihood of repetitive data access.



Data Management: Data Affinity

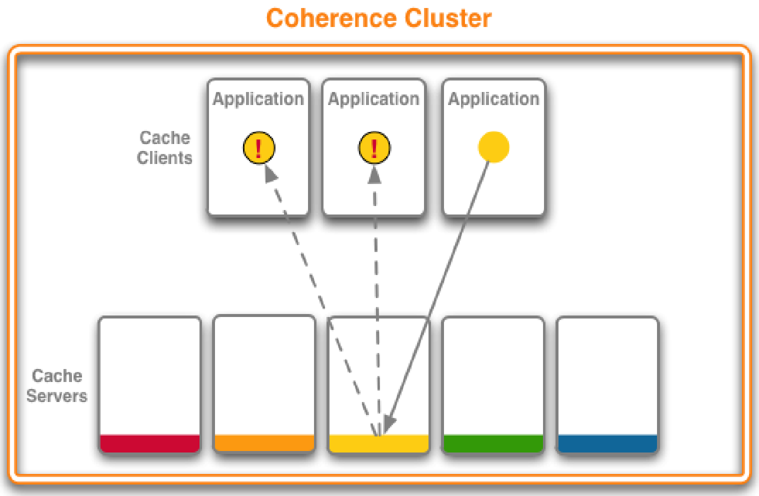
- The ability to **associate objects across caches** guaranteeing they are located **on the same member**.
- **Typical Use Case:** Parent Child relationships



Data Processing Options

Data Processing:
Events - JavaBean Event Model

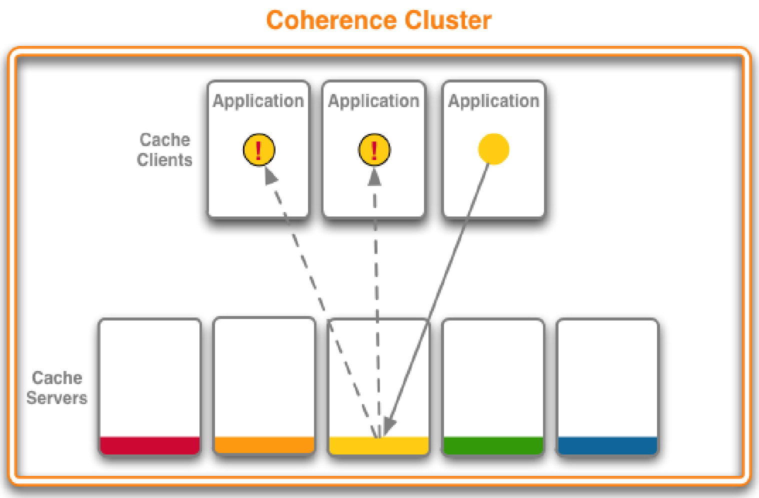
- Listen to all events for all keys
  - ENTRY_DELETED
  - ENTRY_INSERTED
  - ENTRY_UPDATED



```
NamedCache cache = CacheFactory.getCache(“myCache”);
cache.addMapListener(listener);
```

Data Processing:
Events - Key Based Event Model

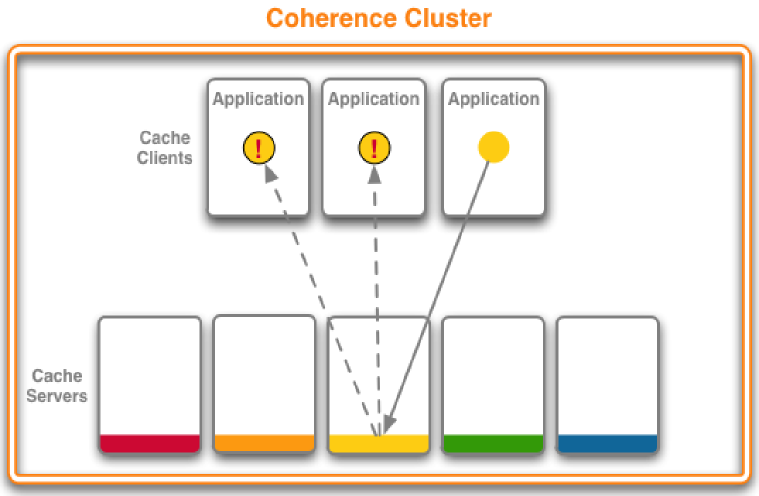
- Listen to changes to a specific key



```
NamedCache cache = CacheFactory.getCache(“myCache”);
cache.addMapListener(listener, key);
```

Data Processing:
Events - Filter Based Event Model

- Listen to a changes to data that match a specific criteria (i.e. Filter)



```
NamedCache cache = CacheFactory.getCache("myCache");
cache.addMapListener(listener, filter);
```

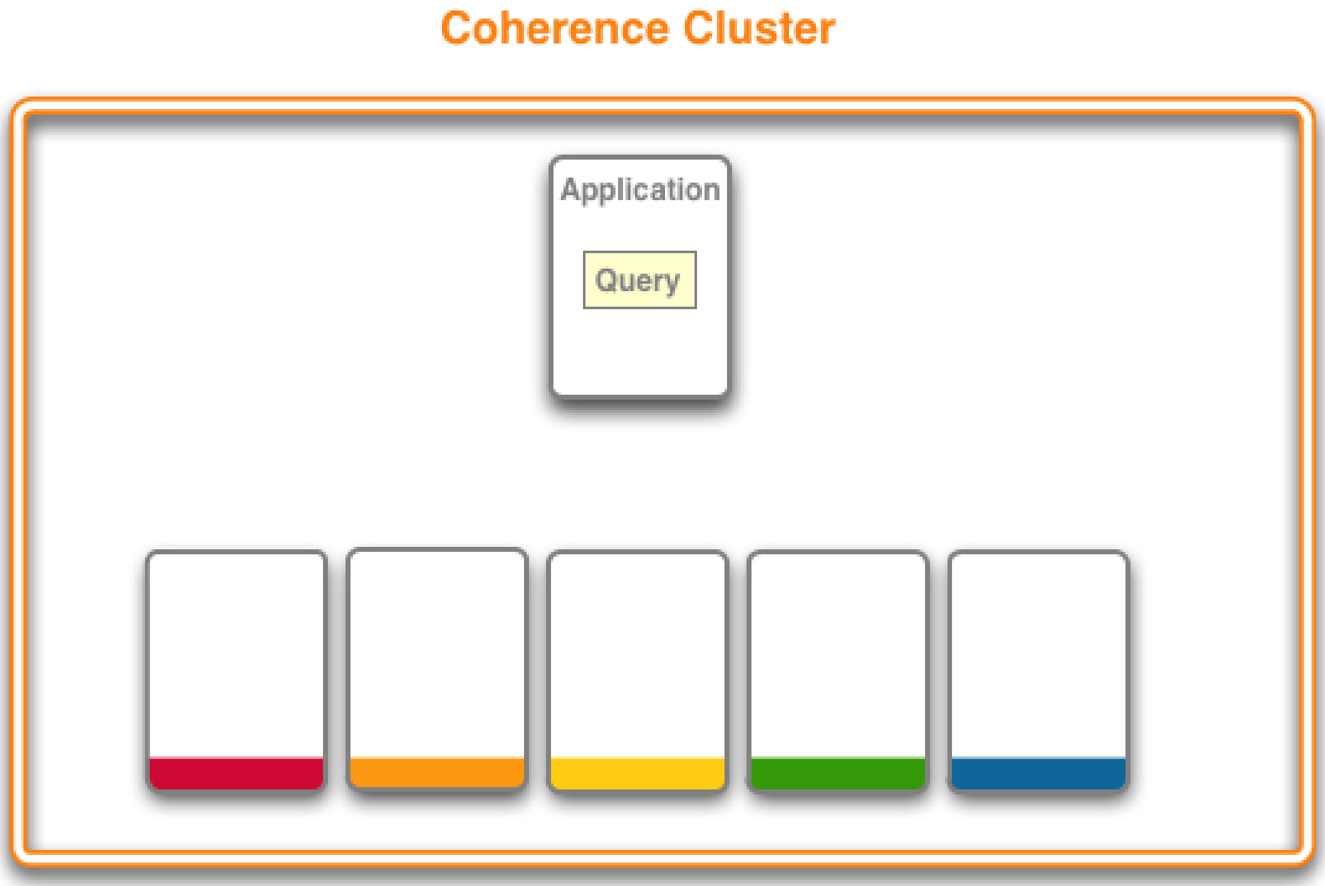
Data Processing: Parallel Query

- Programmatic query mechanism
- Queries performed in parallel across the grid
- Standard indexes provided out-of-the-box and supports implementing your own custom indexes
- Cost-based analysis of Filter application
- Standard Filters provided out-of-the-box (e.g. OR, AND, ALL, EQUALS, etc.)

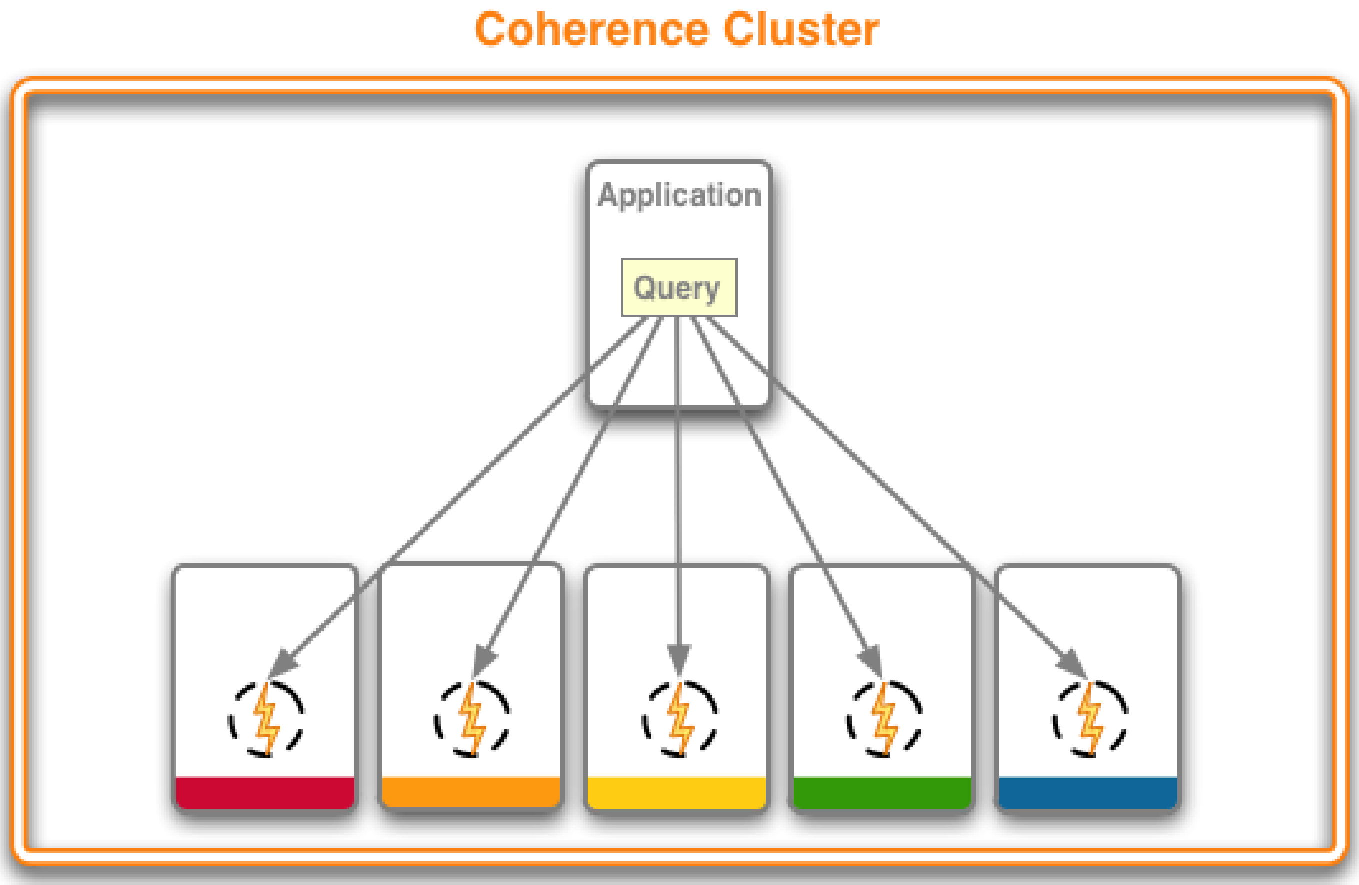
Data Processing: Parallel Query

```
// get the “myTrades” cache
NamedCache cacheTrades =
  CacheFactory.getCache(“myTrades”);

// create the “query”
Filter filter =
 new AndFilter(
  new EqualsFilter("getTrader", traderid),
  new EqualsFilter("getStatus", Status.OPEN));

// perform the parallel query
Set setOpenTrades = cacheTrades.entrySet(filter);
```

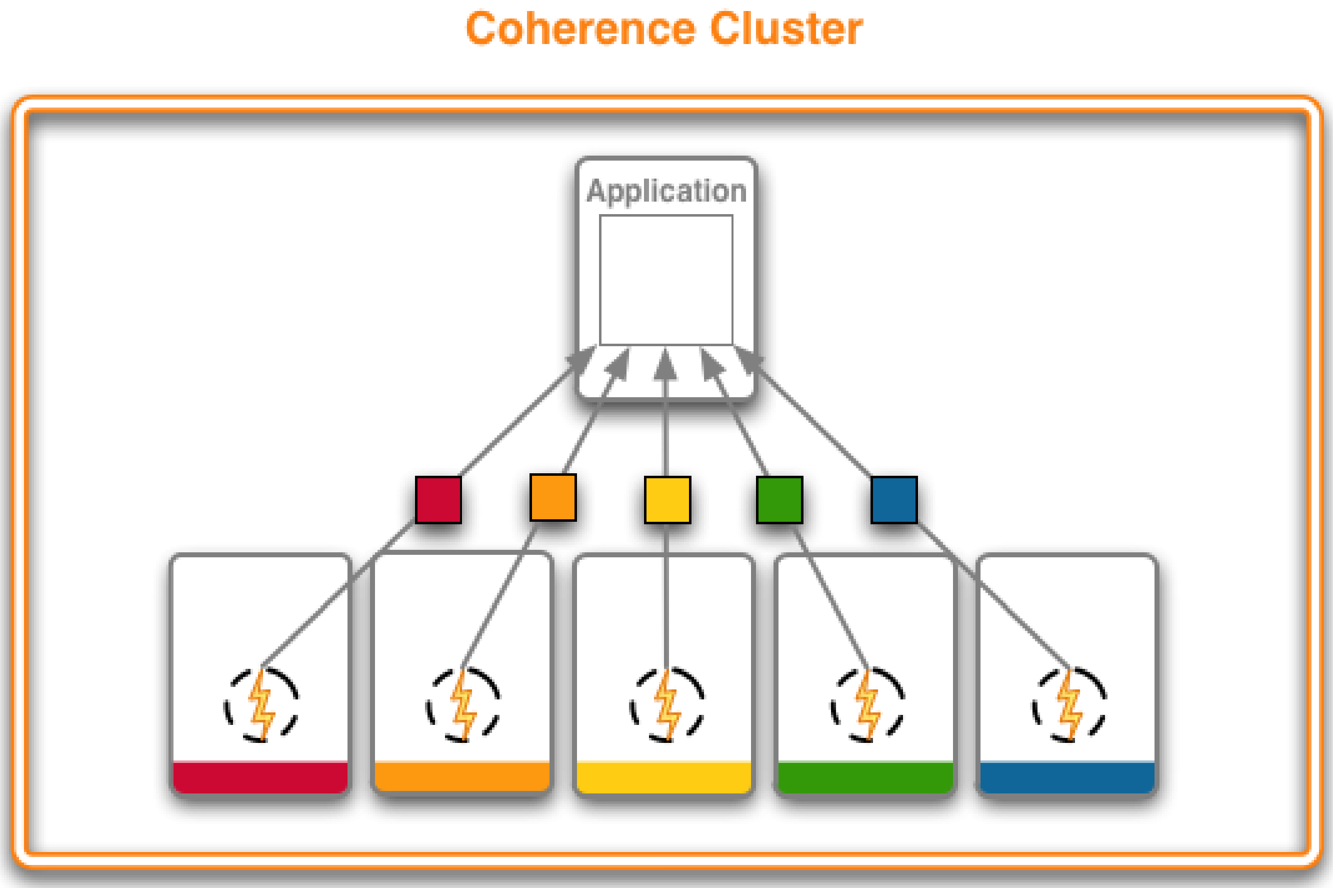
Data Processing:
Parallel Query



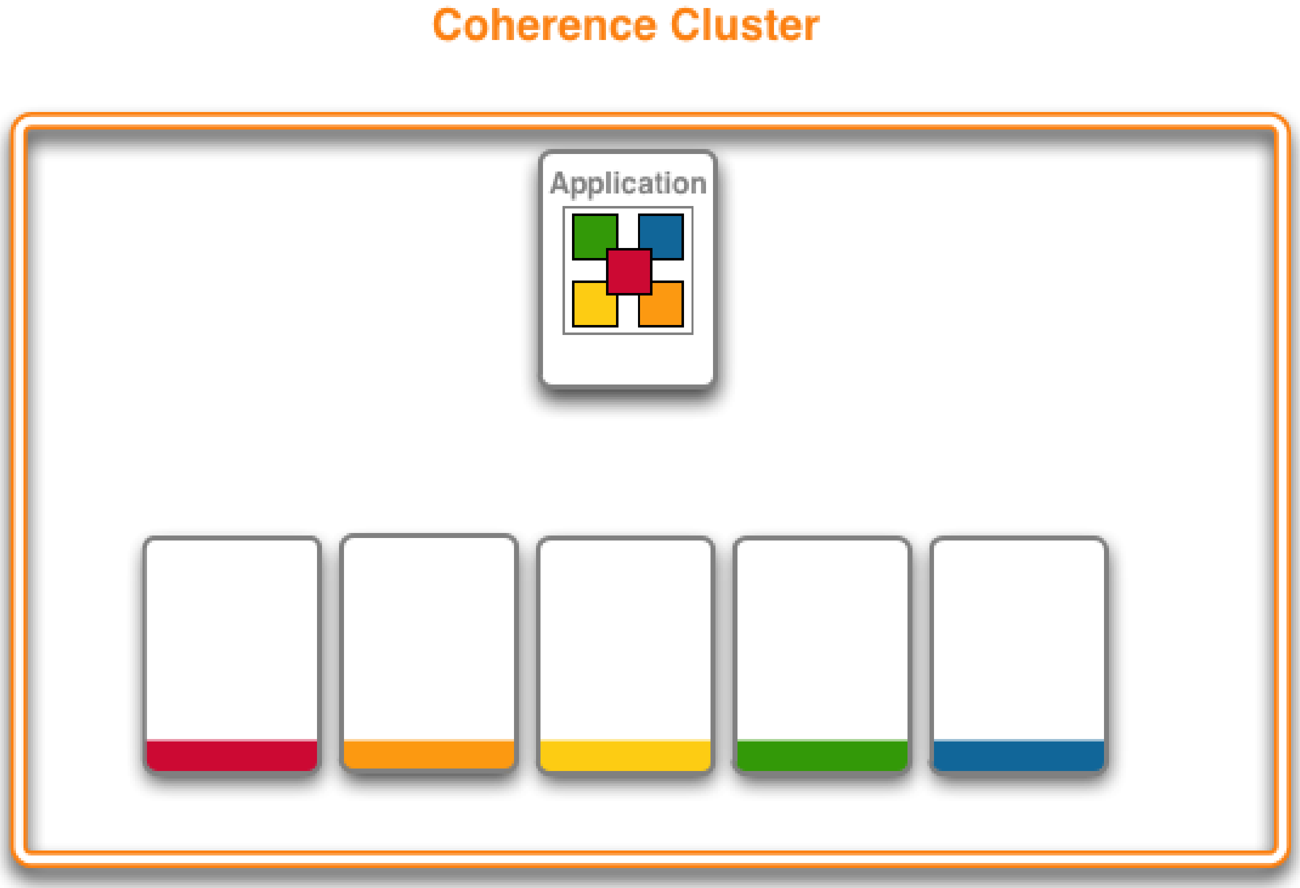
Data Processing:
Parallel Query



Data Processing: Parallel Query



Data Processing: Parallel Query



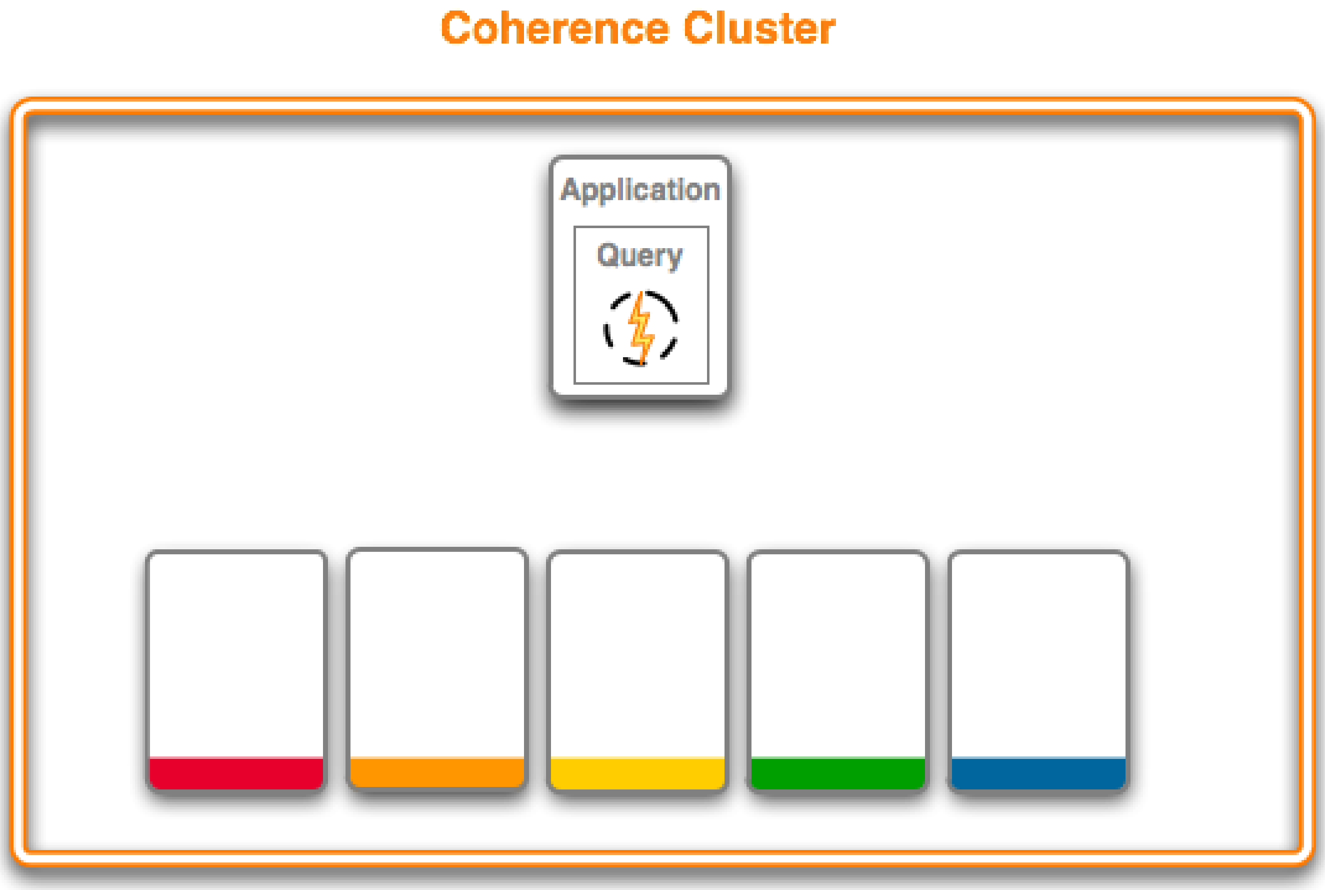
Data Processing:
Continuous Query Cache

- Automatically, transparently and dynamically maintains a view locally based on a specific criteria (i.e. Filter)
- Same API as all other Coherence caches
- Support local listeners.
- Supports layered views

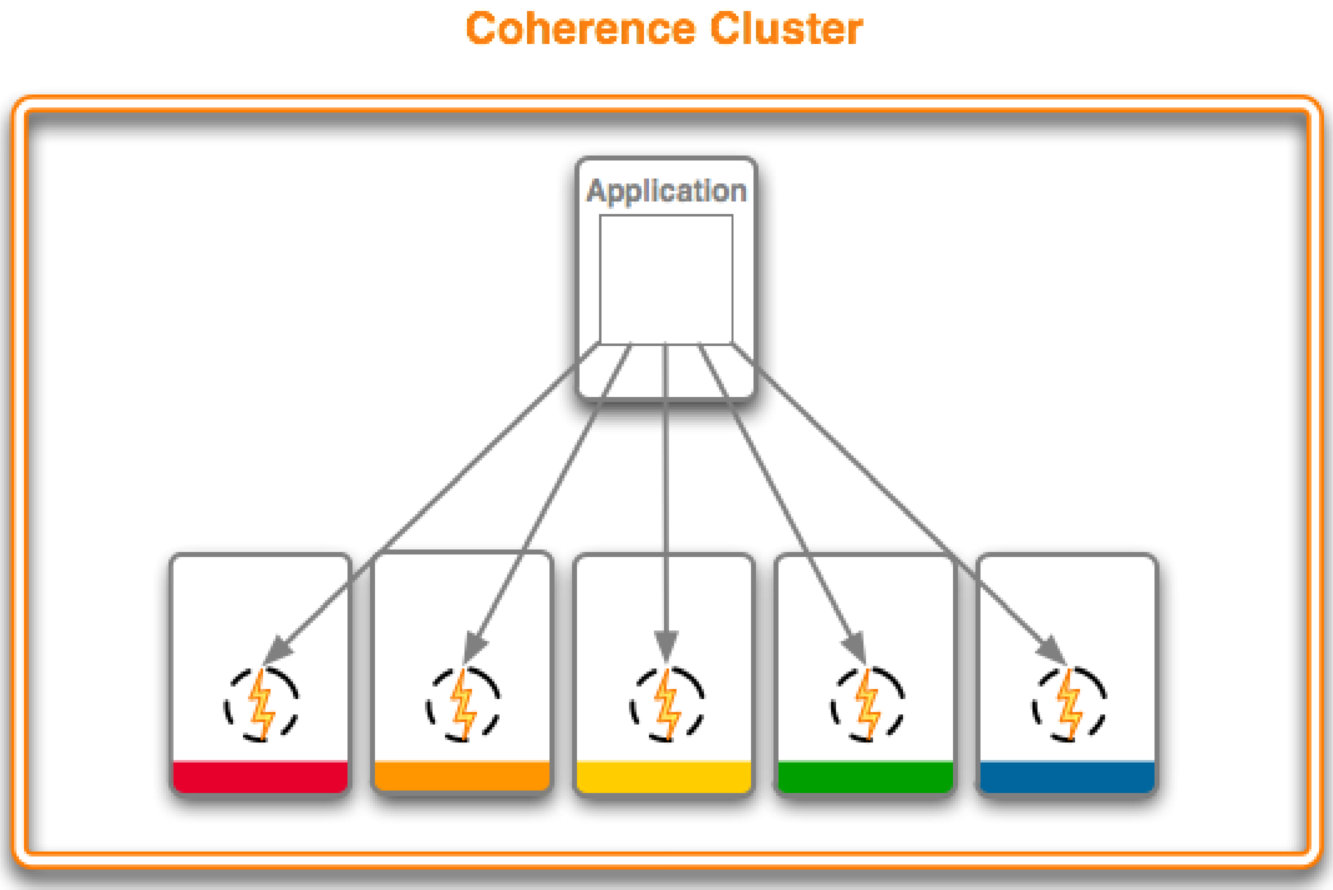
Data Processing:
Continuous Query Cache

```
// get the “myTrades” cache
NamedCache cacheTrades = CacheFactory.getCache(“myTrades”);

// create the “query”
Filter filter =
 new AndFilter(
  new EqualsFilter("getTrader", traderid),
  new EqualsFilter("getStatus", Status.OPEN));

// create the continuous query cache
NamedCache cqcOpenTrades = new
  ContinuousQueryCache(cacheTrades, filter);
```

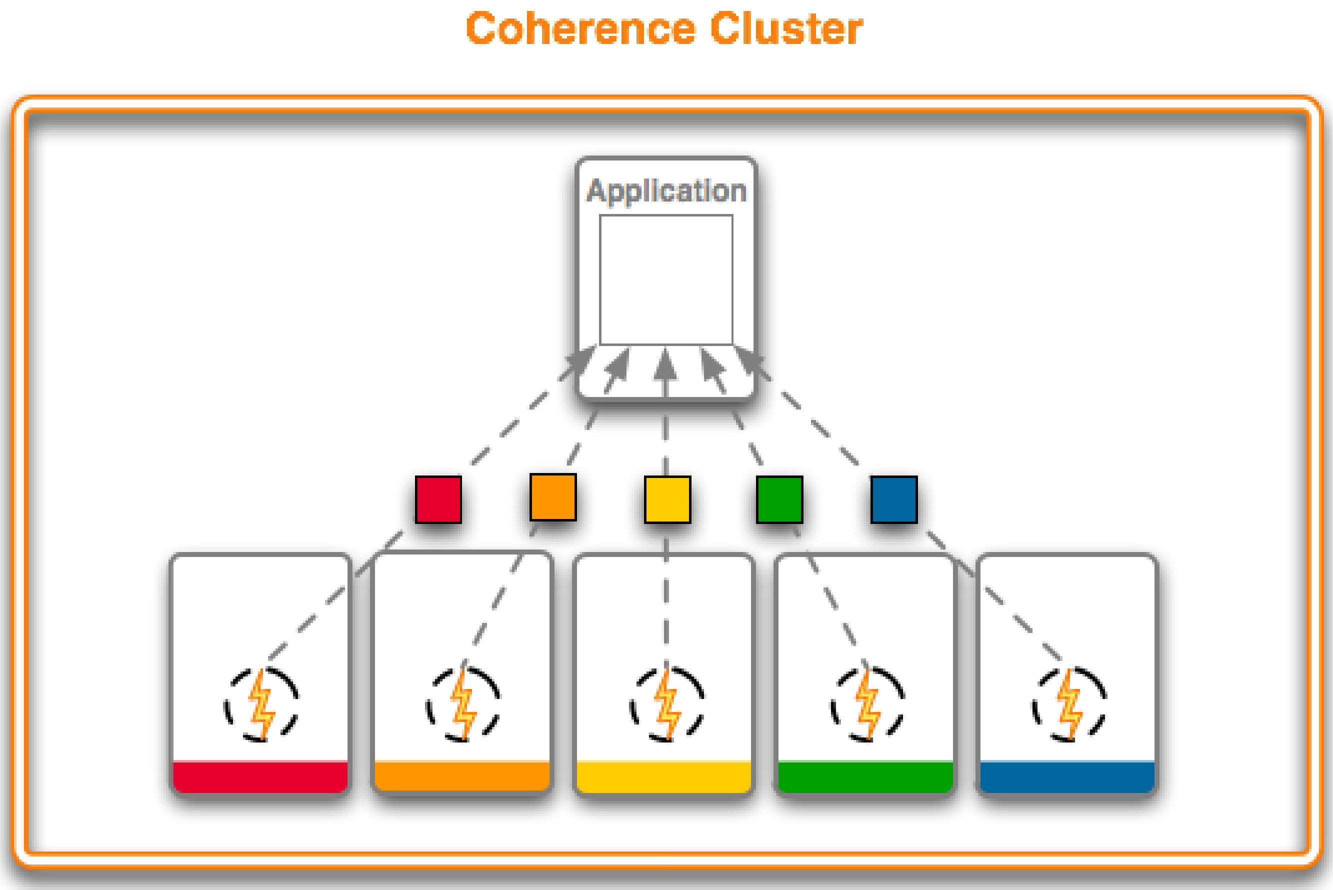
Data Processing: Continuous Query Cache



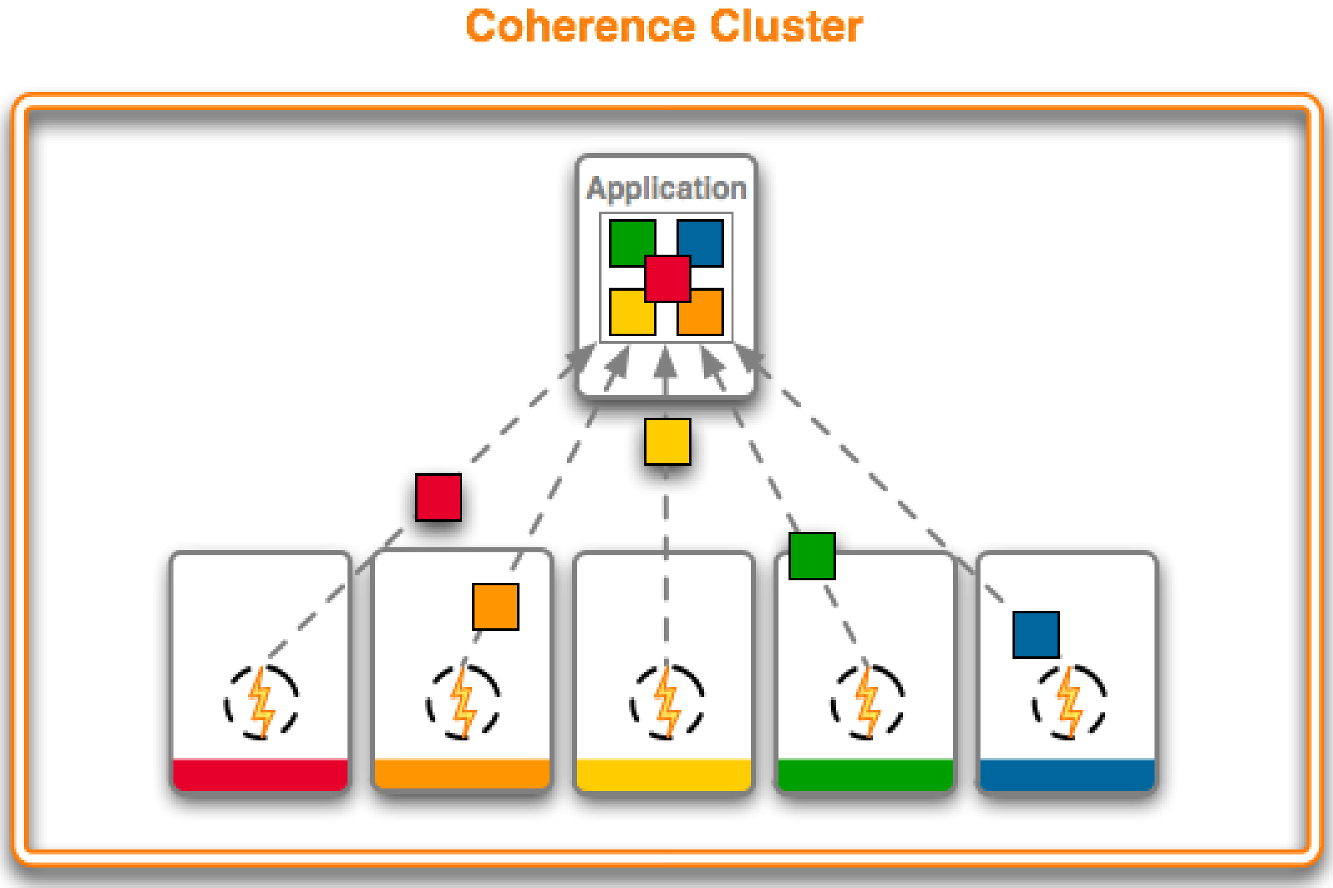
Data Processing:
Continuous Query Cache



Data Processing:
Continuous Query Cache



Data Processing:
Continuous Query Cache



End Of Part I

Part II

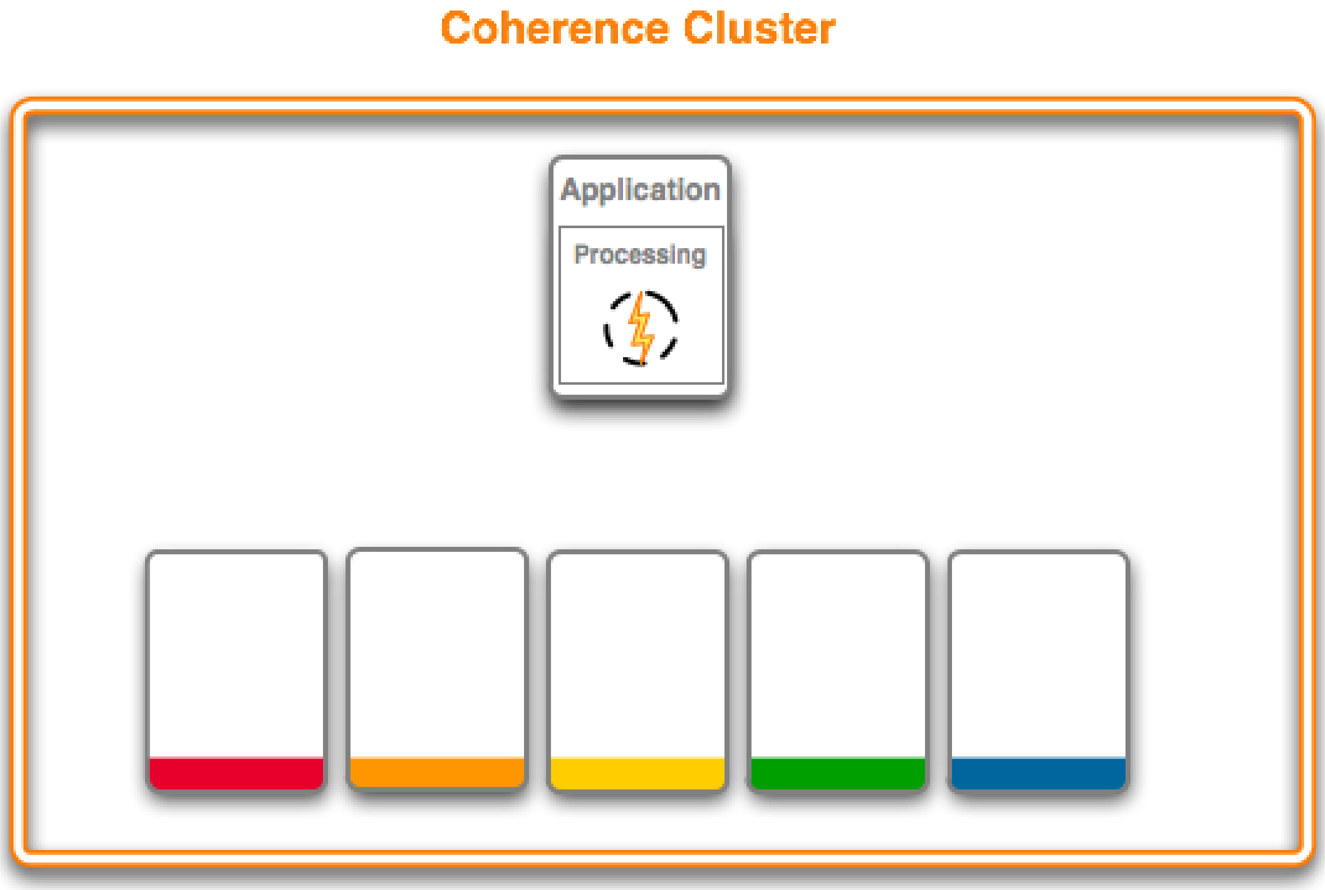
Data Processing:
Invocable Map

- The **inverse of caching**
- Sends the processing (e.g. EntryProcessors) to where the data is in the grid
- Standard EntryProcessors provided Out-of-the-box
- Once and only once guarantees
- Processing is automatically fault-tolerant
- Processing can be:
    - Targeted to a specific key
    - Targeted to a collection of keys
    - Targeted to any object that matches a specific criteria (i.e. Filter)

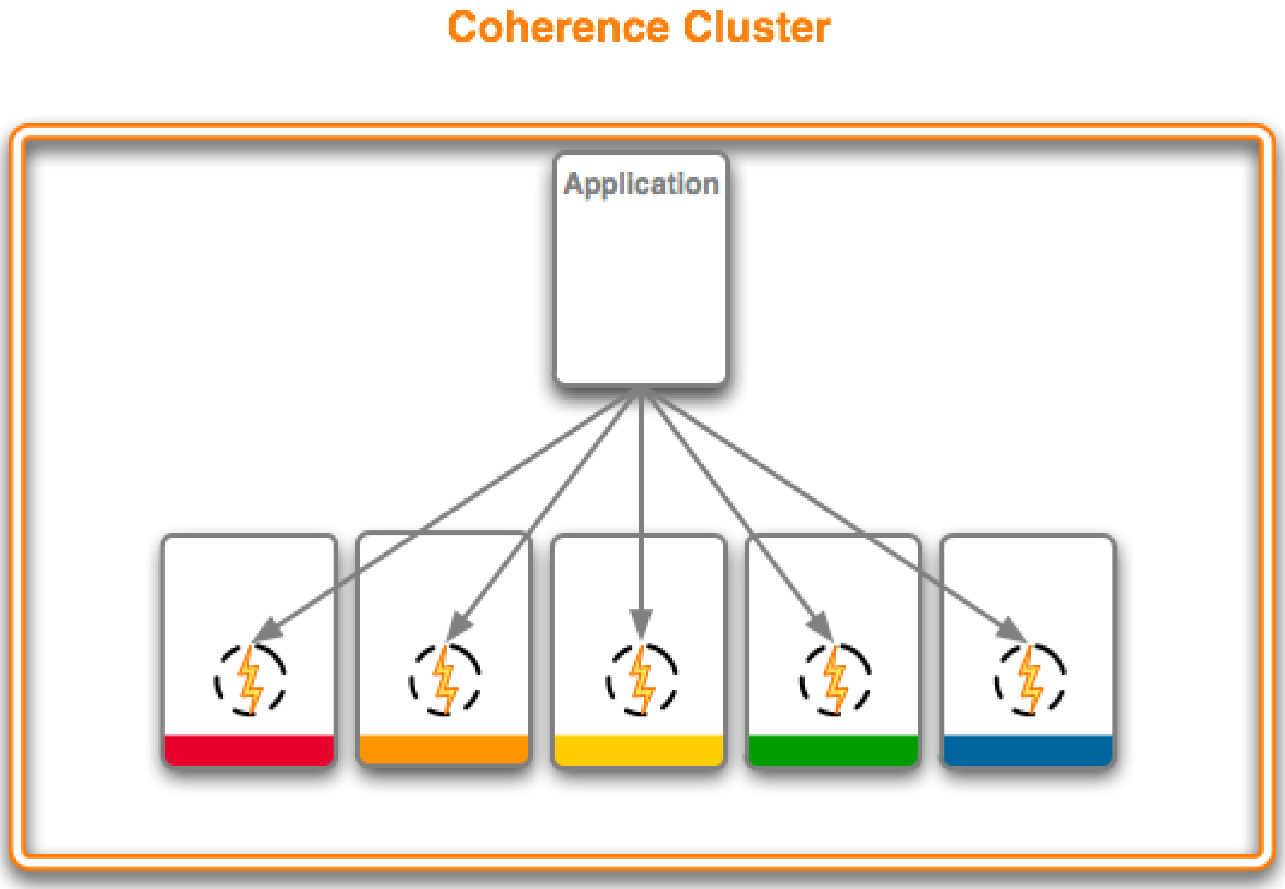
Data Processing: Invocable Map

```
// get the “myTrades” cache
NamedCache cacheTrades =
  CacheFactory.getCache(“myTrades”);

// create the “query”
Filter filter =
 new AndFilter(
  new EqualsFilter("getTrader", traderid),
  new EqualsFilter("getStatus", Status.OPEN));

// perform the parallel processing
cacheTrades.invokeAll(filter, new
  CloseTradeProcessor());
```

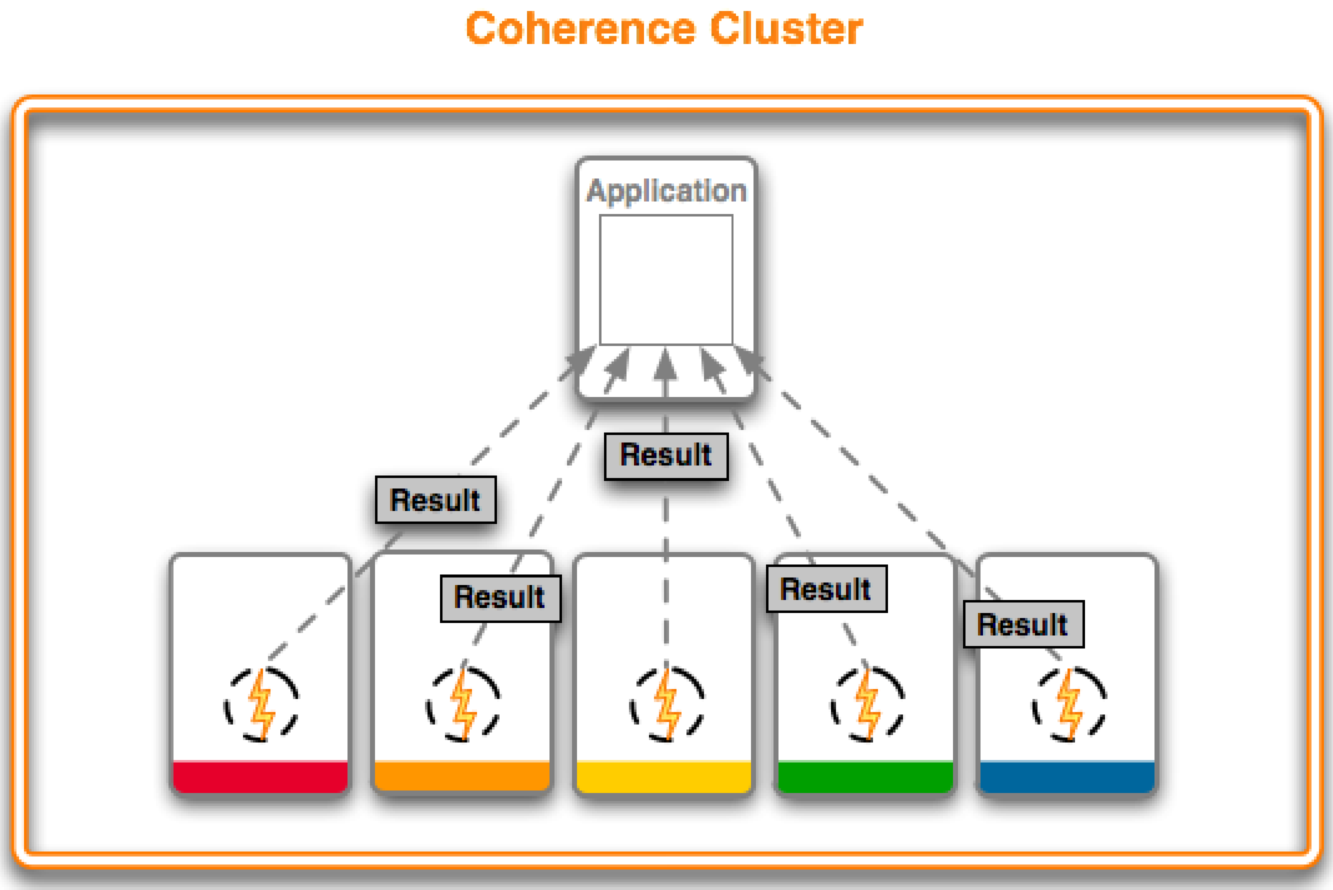
Data Processing: Invocable Map



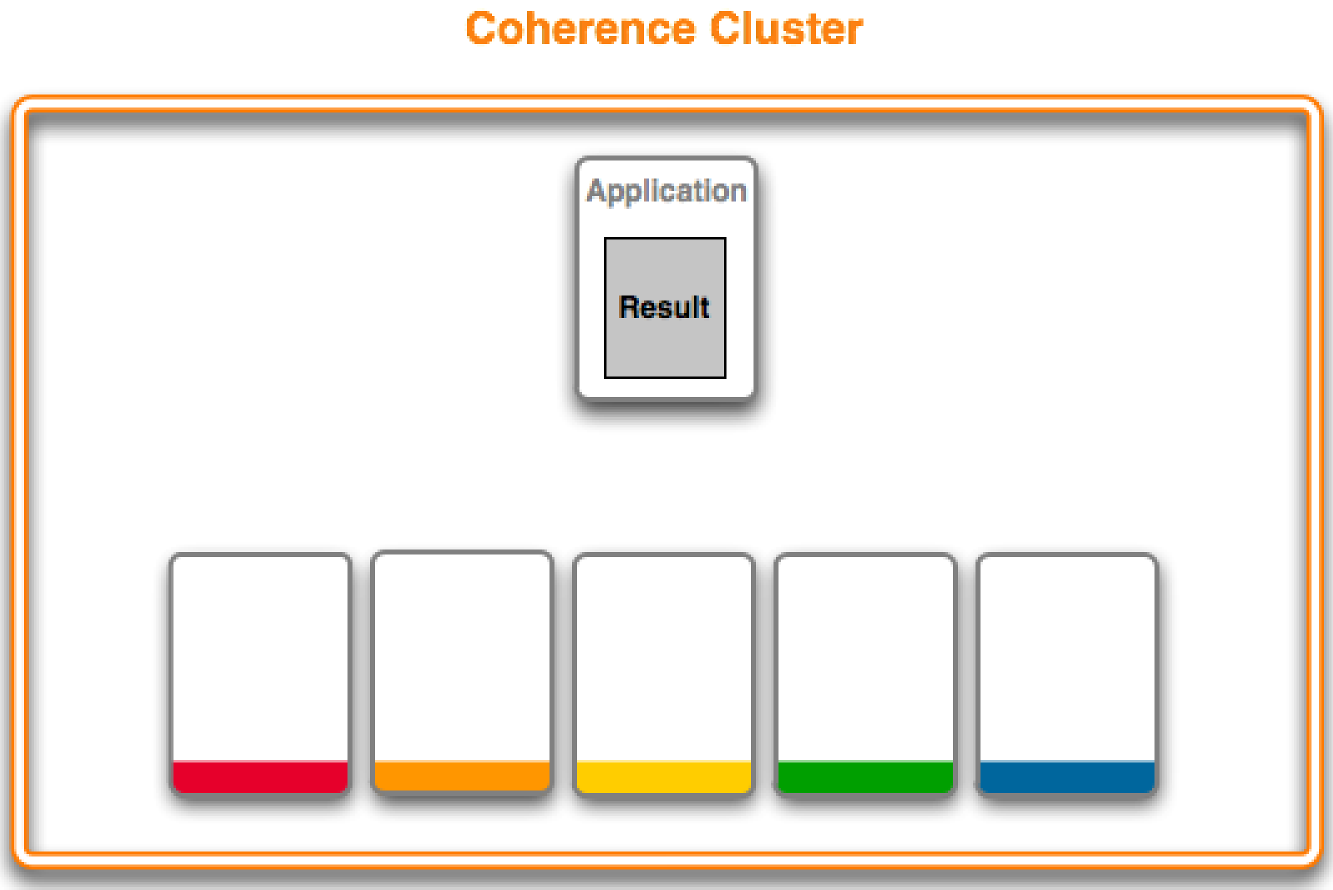
Data Processing: Invocable Map



Data Processing: Invocable Map



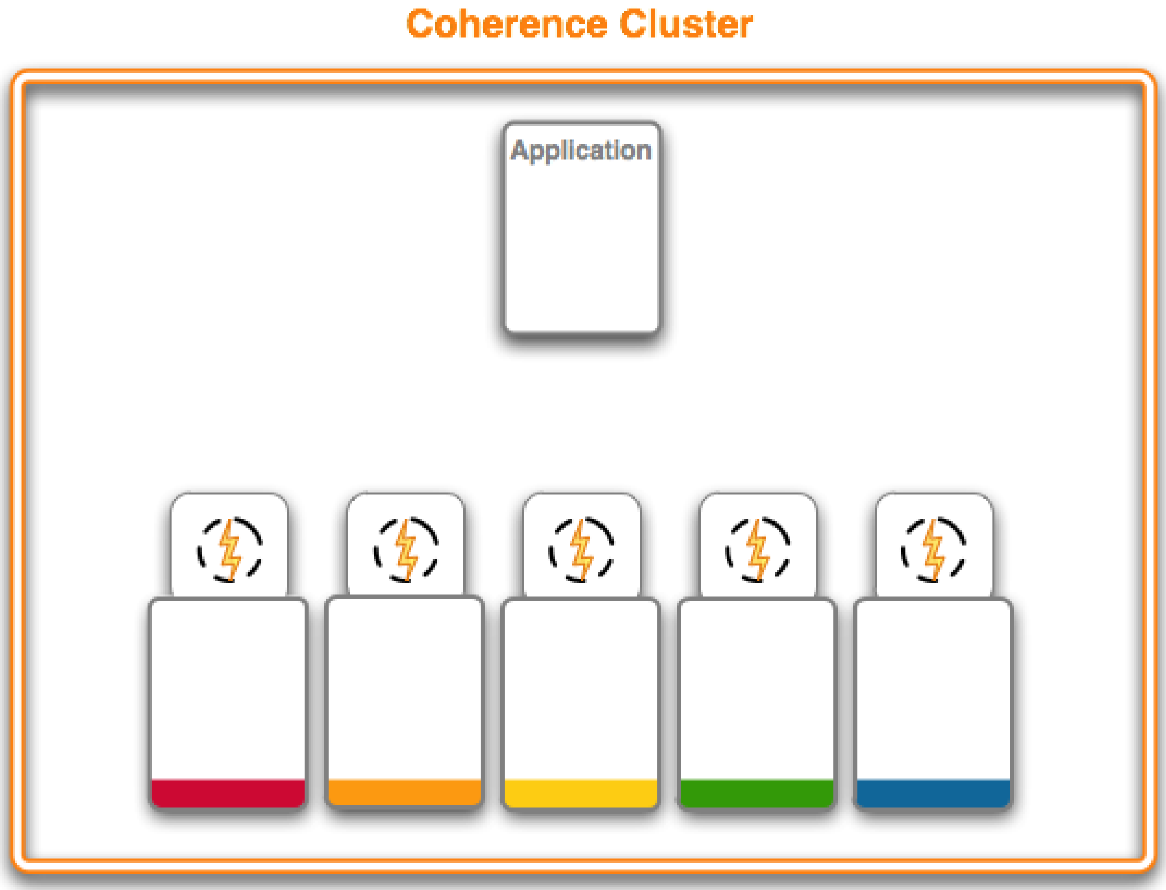
Data Processing: Invocable Map



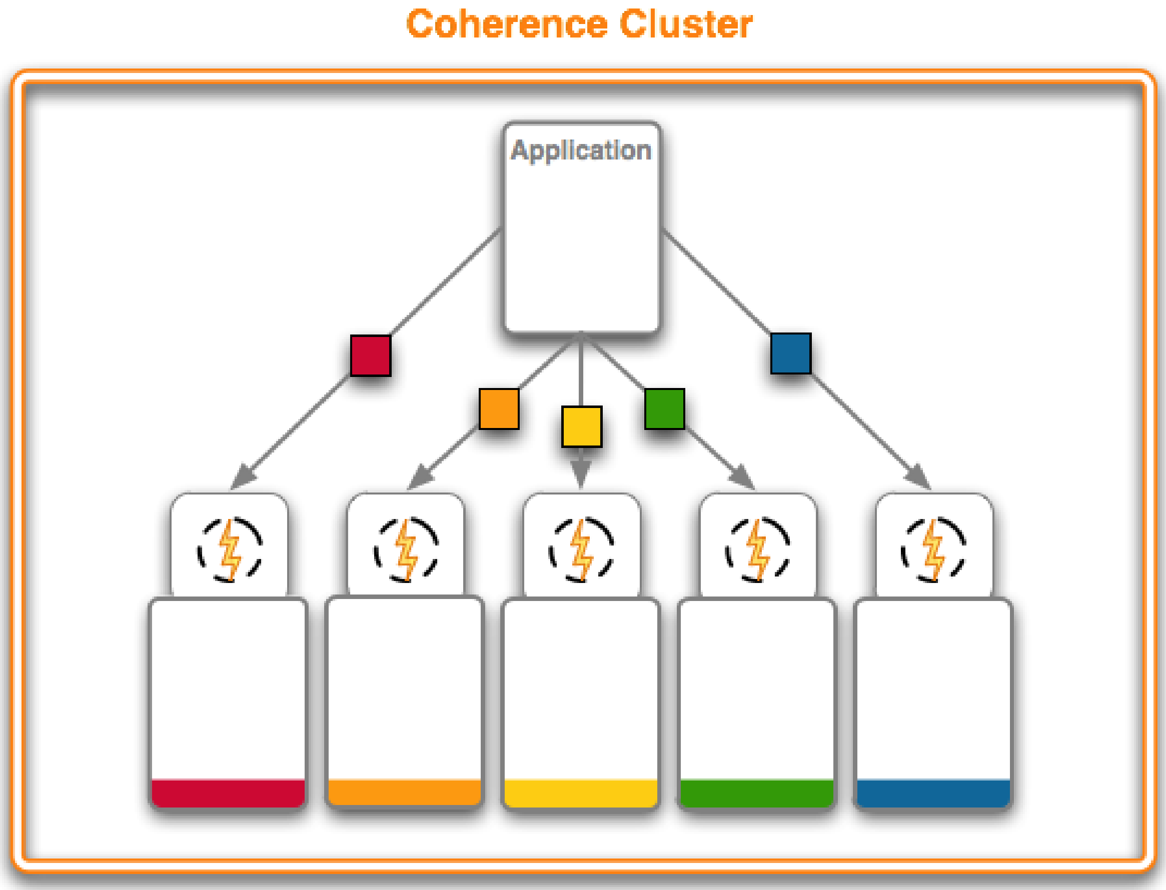
Data Processing: Triggers

- Inject pre-processing logic to data being added to a cache.
- Similar to EntryProcessors, but fired before a mutation takes place.
- They allow your “process” method to override, replace, decorate, remove or fail a cache mutation.
- Adds veto ability to data insertion.
- Common Uses:
  - Prevent invalid transactions;
  - Enforce complex security authorizations;
  - Enforce complex business rules;
  - Gather statistics on data modifications;

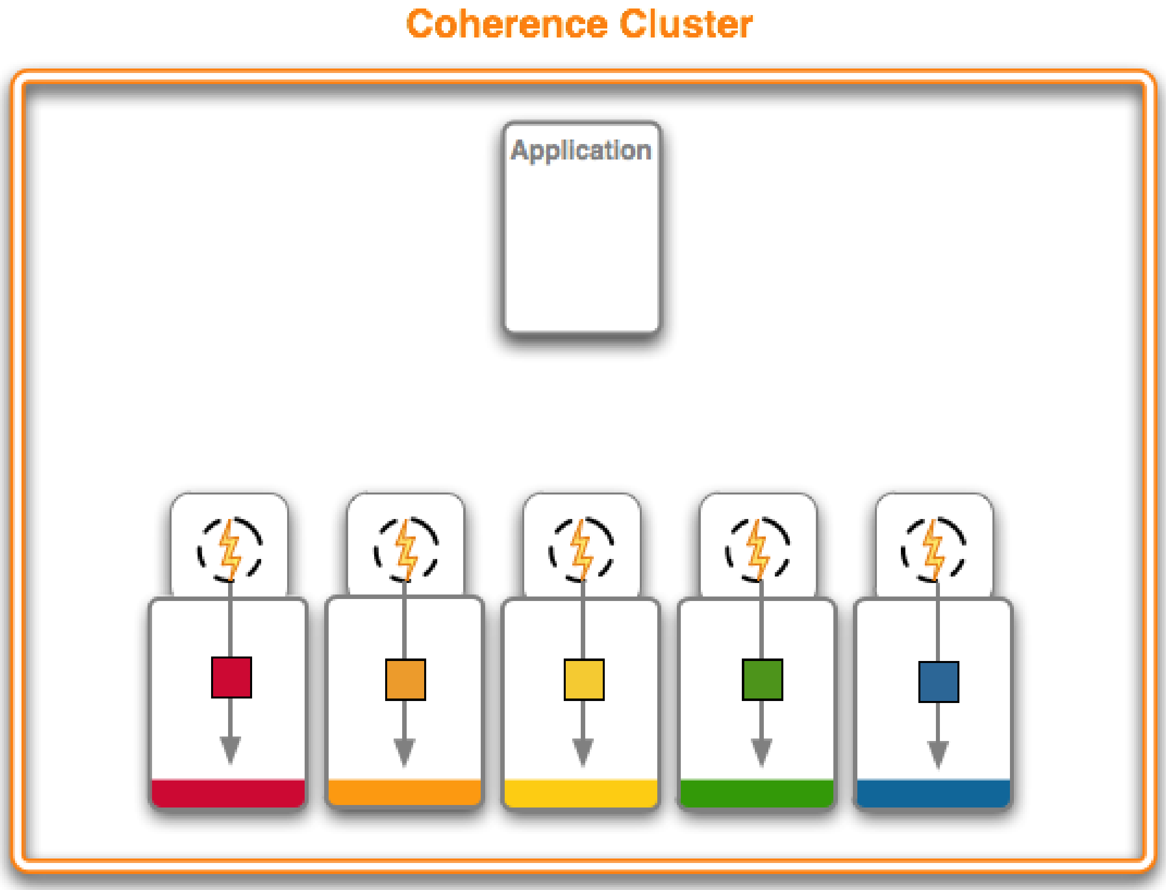
Data Processing: Triggers



Data Processing: Triggers



Data Processing: Triggers



The Coherence Incubator

http://coherence.oracle.com/display/INCUBATOR/

The Coherence Incubator

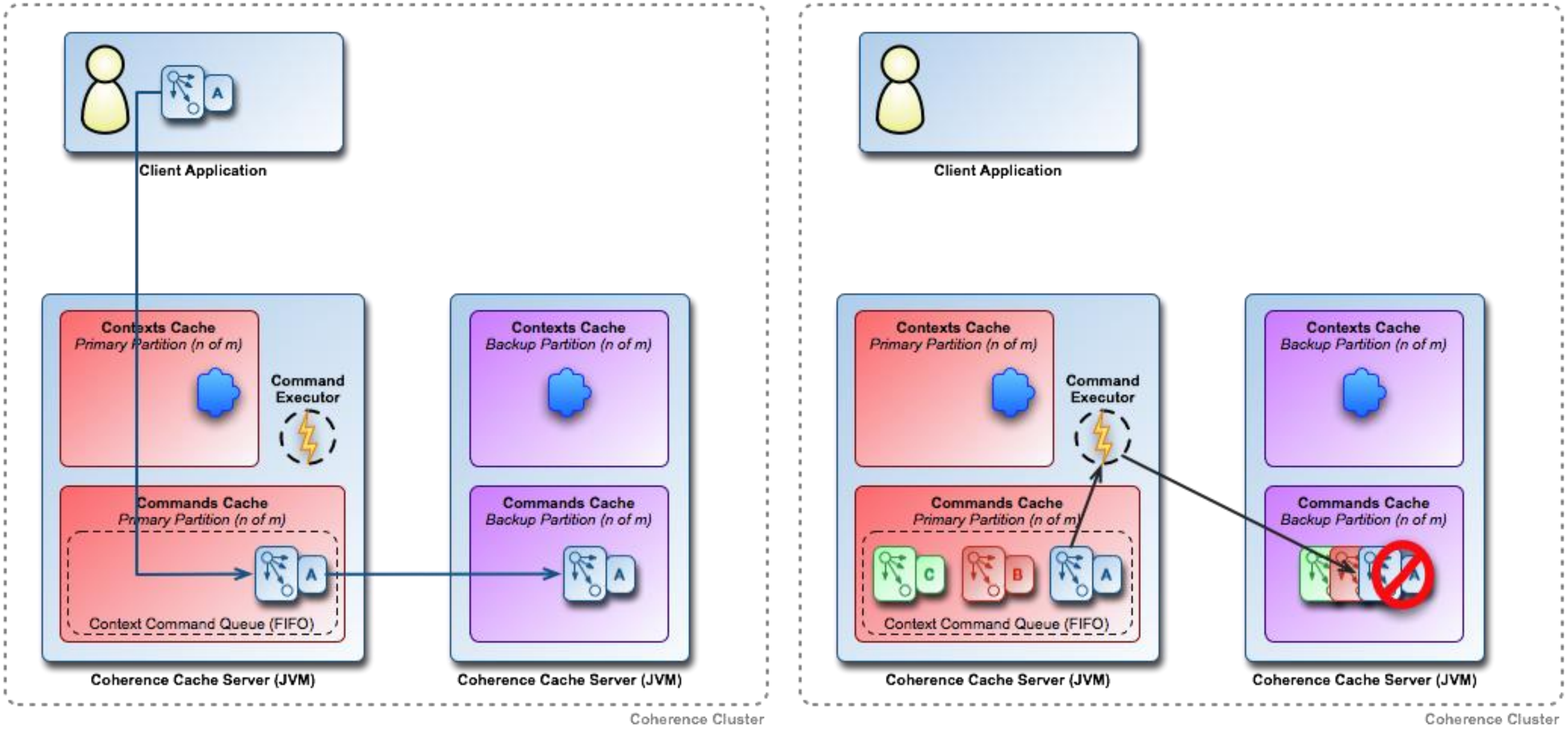
The Coherence Incubator **hosts a repository of projects** providing **example implementations** for commonly used design patterns, system integration solutions, distributed computing concepts and other artifacts designed to enable rapid delivery of solutions to potentially complex business challenges **built using or based on Oracle Coherence**.

The Coherence Incubator:
The Command Pattern

Distributed implementation of the classic Command Pattern

- Useful alternative to EntryProcessors with the advantage that Commands are executed asynchronously.
- Provides essential infrastructure for several other Incubator projects to permit **guaranteed, in-order, asynchronous processing of Commands**.

The Coherence Incubator: The Command Pattern



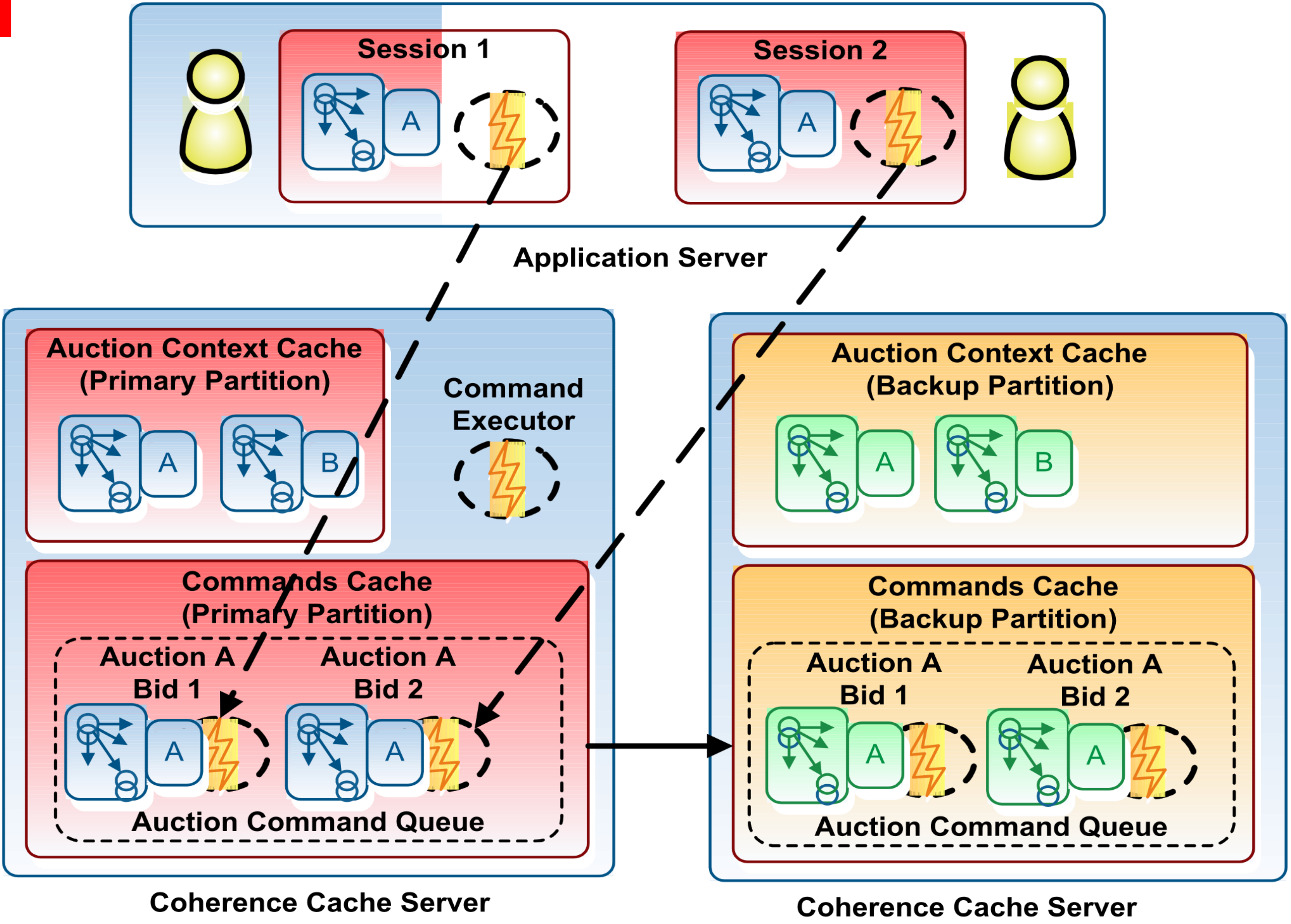
Client Application
A
Contexts Cache
Primary Partition (n of m)
Command Executor
Commands Cache
Primary Partition (n of m)
Context Command Queue (FIFO)
Coherence Cache Server (JVM)
Contexts Cache
Backup Partition (n of m)
Commands Cache
Backup Partition (n of m)
Coherence Cache Server (JVM)
Coherence Cluster
Client Application
Contexts Cache
Primary Partition (n of m)
Command Executor
Commands Cache
Primary Partition (n of m)
C
B
A
Context Command Queue (FIFO)
Coherence Cache Server (JVM)
Contexts Cache
Backup Partition (n of m)
Commands Cache
Backup Partition (n of m)
Coherence Cache Server (JVM)
Coherence Cluster

The Coherence Incubator:
The Functor Pattern

- This is an **example implementation** of Function Objects (Wikipedia) or as it is also known, the Functor Pattern, built with Coherence.
- The Functor Pattern is an extension to the Command Pattern. In fact the semantics are identical with the exception that the Functor Pattern additionally provides a mechanism to return values (or re-throw exceptions) to the *Submitter* (using Java 5+ Futures) where as the Command Pattern does not provide such capabilities.

Functor Pattern: Quick Overview Of Auction App

- Goal Demonstrate the Grid Differentation in WLS Suite
  - Show a close to real life application
  - Coherence
    - Coherence Patterns
  - Grid Messaging
    - Shows WLS JMS and AQ integration
  - Eclipse JPA
    - Best of breed JPA implementation
  - Integration Check points
    - Coherence Web
  - Administrative
    - WLST and Domain templates



Session 1

A

Session 2

A

Application Server

Auction Context Cache
(Primary Partition)

A

B

Command
Executor

Commands Cache
(Primary Partition)

Auction A
Bid 1

A

Auction A
Bid 2

A

Auction Command Queue

Coherence Cache Server

Auction Context Cache
(Backup Partition)

A

B

Commands Cache
(Backup Partition)

Auction A
Bid 1

A

Auction A
Bid 2

A

Auction Command Queue

Coherence Cache Server

What Happens when you create an Auction

- Uses EclipseLink JPA to store in Oracle RDBMS
- Registers the Auction in Coherence
- Enqueues a message to be Delivered in the FUTURE

What happens during bidding?

- A submit bid goes to coherence context registered
  - Request gets co-located and queued
  - WLS Returns
- On Coherence
  - Each bid is processed in order
  - Rules are checked
  - Price is updated
  - Stored in Oracle in RDBMS

How do Auctions Close?

- MDB listens for a dequeue….
- If reserve has been meet move to settlement
- If not mark as closed
- Auction is removed from the bidding engine

The Coherence Incubator

http://coherence.oracle.com/display/INCUBATOR/

For More Information

search.oracle.com

Oracle coherence

or

oracle.com

For More Information

Get Started

- Visit the Oracle Fusion Middleware 11g web site at http://www.oracle.com/fusionmiddleware11g
- Oracle WebLogic Server on oracle.com http://www.oracle.com/appserver
- Oracle Application Grid on oracle.com http://ww.oracle.com/goto/applicationgrid
- Oracle Fusion Middleware on OTN http://otn.oracle.com/middleware

Resources

- App Grid Blog http://blogs.oracle.com/applicationgrid
- For WebLogic Server technical information: http://www.oracle.com/technology/products/weblogic/
- For Application Grid technical information http://www.oracle.com/technology/tech/grid/

ORACLE®