

ORACLE®



ORACLE[®]

QCon: London 2009


Data Grid Design Patterns

Brian Oliver | [Global Solutions Architect](#) | brian.oliver@oracle.com
[Oracle](#) Coherence | [Oracle](#) Fusion Middleware Product Management



Agenda

- Traditional Patterns
- The Coherence Incubator
- The Command Pattern
- The Functor Pattern
- The Messaging Pattern
- And more...



The proceeding is intended to outline general product use and direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



Traditional Patterns



Traditional Patterns

- **Data Grid as a Cache**
 - Read Only (static data)
 - Read + Write (volatile data)
 - Cache Aside
 - Read Through
 - Write Through
 - Write Behind
 - Local
 - Near
 - Remote
 - Session Management



Traditional Patterns

- **Data Grid: “it’s a system of record”**
 - **Precondition:**
Data Availability, Reliability, Consistency, Coherency...
 - Observing Updates
 - Parallel Queries
 - Parallel Aggregation (Map Reduce)
 - In-Place Updates (Transactions)
 - Triggers (intercepting updates)
 - Transformers
 - Event Driven Architectures



The Coherence Incubator

The Coherence Incubator



- Repository of Projects
 - Design Patterns, Systems Integrations, Helpful Tools
- Example Implementations
 - We've be involved in 1000's of POCs
 - We're "cleaning" and publishing "generic" implementations
 - Complete Source Code & Binary Distributions
 - Complete Documentation



The Coherence Incubator

- Where?

- Web:

- <http://coherence.oracle.com/display/INCUBATOR/Home>

- Forum:

- <http://forums.oracle.com/forums/forum.jsp?forumID=558>



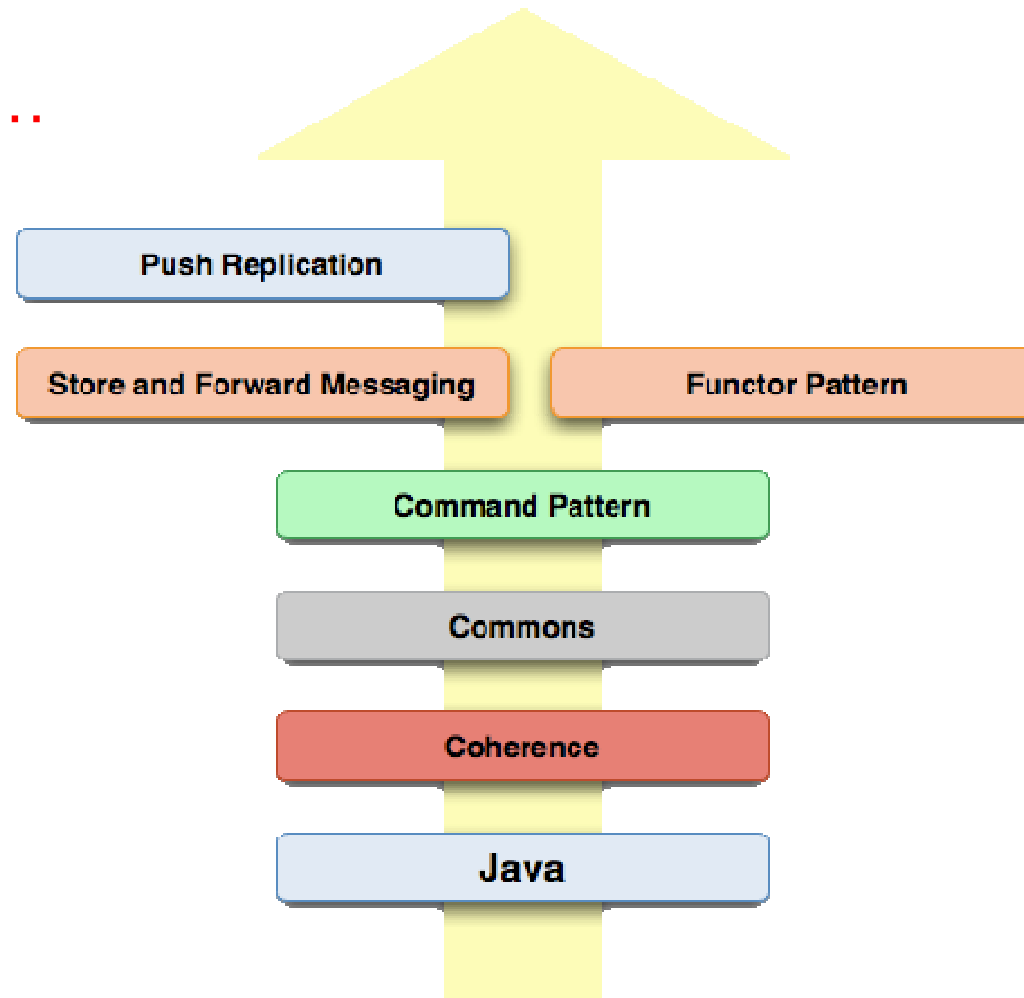
The Coherence Incubator

- Why?
 - Make everyone's life easier!
 - We've always wanted to do provide more examples!
- What? (so far...)
 - Coherence Common
 - Command Pattern
 - Functor Pattern
 - Store and Forward Messaging Pattern
 - Push Replication Pattern



The Coherence Incubator

- Building Blocks...





The Coherence Incubator

- Who?
 - Project Leads (from Oracle)
 - Significant Community Contributions
 - Encourage you to contribute
 - Ideas, Feedback, Documentation, etc.
 - Email Access to Project Leads
 - Dedicated Oracle Forum



The Coherence Incubator

- Next projects? (perhaps...)
 - Task Pattern (Execution Service)
 - Staged Object Processing Pattern
 - Spring Integration
 - Platform Symphony Integration



The Command Pattern



The Command Pattern

Advocates that;

1. An action to be performed zero or more times at some point in the future should be represented as an object called a Command.
2. All necessary parameters required to perform the said action, should be encapsulated as attributes within the said Command object.
3. The Command object must provide a mechanism to perform the action (typically represented as an execute method defined on the said Command object)

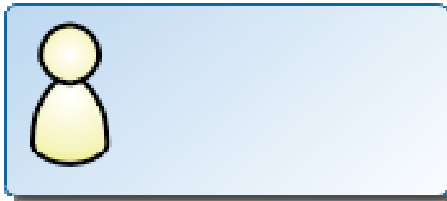


The Command Pattern

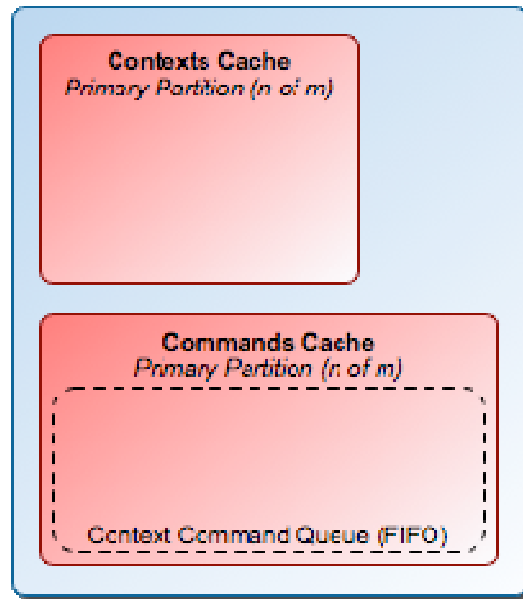
This implementation of the Command Pattern additionally advocates that;

1. A Command may only be executed with in the scope of a target object, called a Context.
2. To execute a Command it must be submitted to a Context.
3. Once submitted , Commands are executed asynchronously.
4. Commands are executed one-at-a-time, in the order in which they arrive at their Context.
5. A Command may not return a value to the application that submitted the said Command.

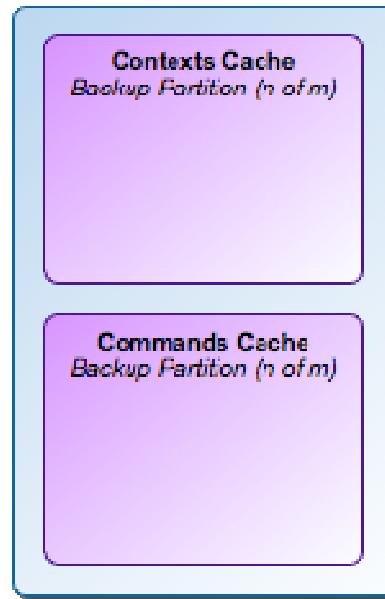
The Initial Environment



Client Application



Coherence Cache Server (JVM)

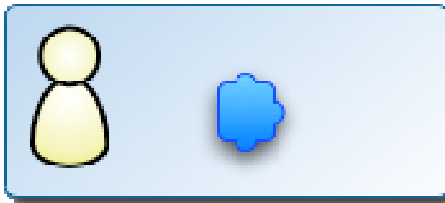


Coherence Cache Server (JVM)

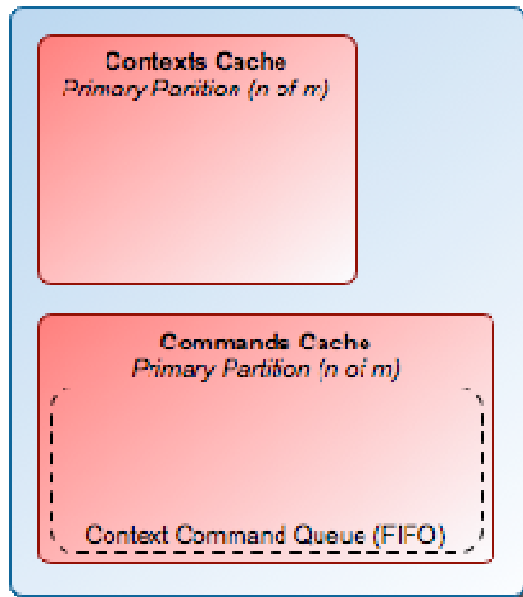
- One Client Application
- Two Coherence Cache Servers
- All part of the same Coherence Cluster *

* The Client Application may be external to the Coherence Cluster when using Coherence *Extend

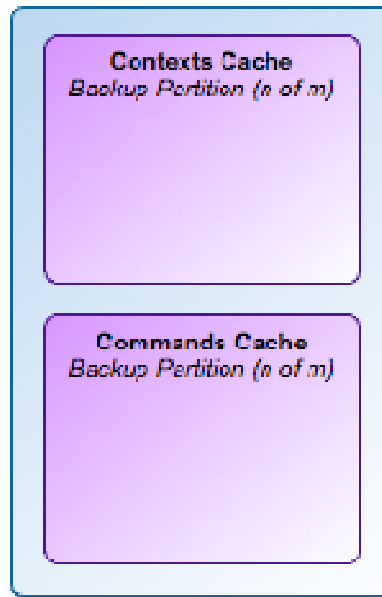
Context Creation



Client Application



Coherence Cache Server (JVM)

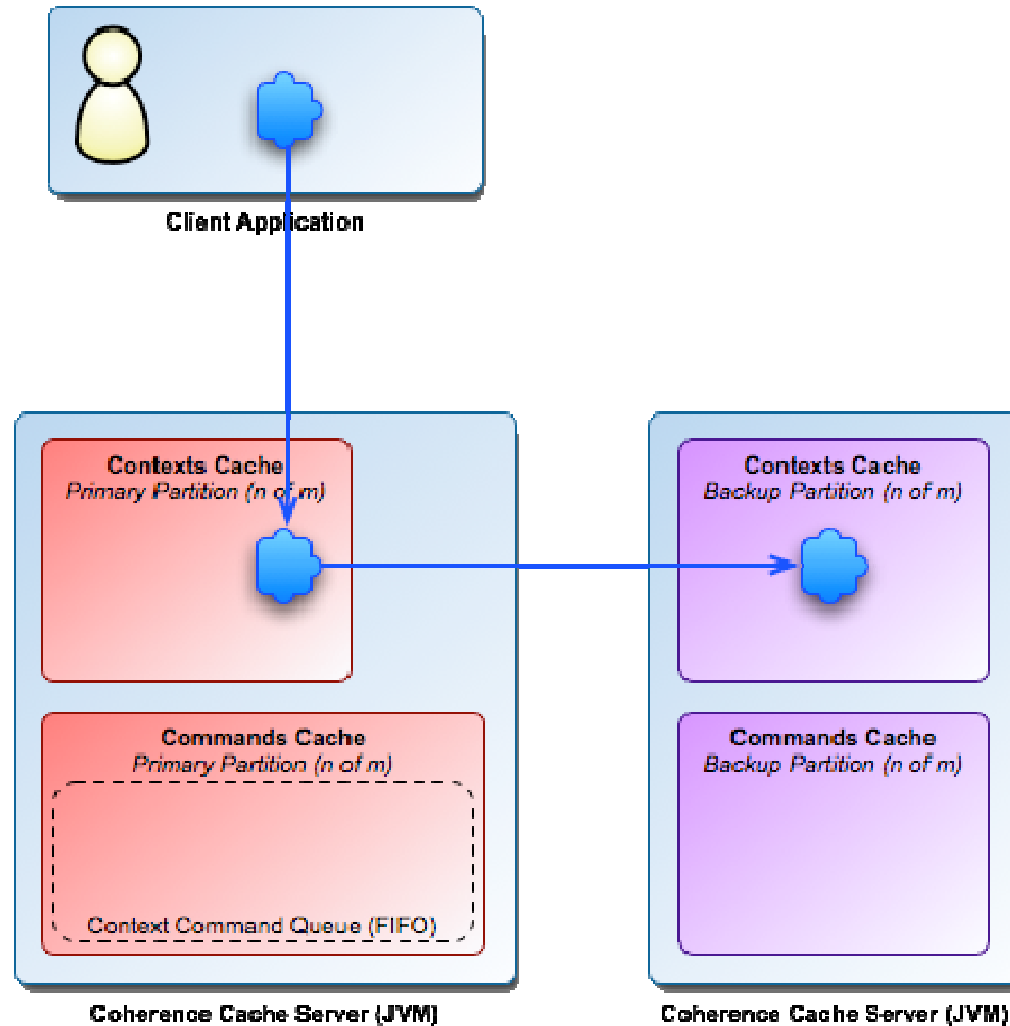


Coherence Cache Server (JVM)

Coherence Cluster

- The client application creates an instance of a Context

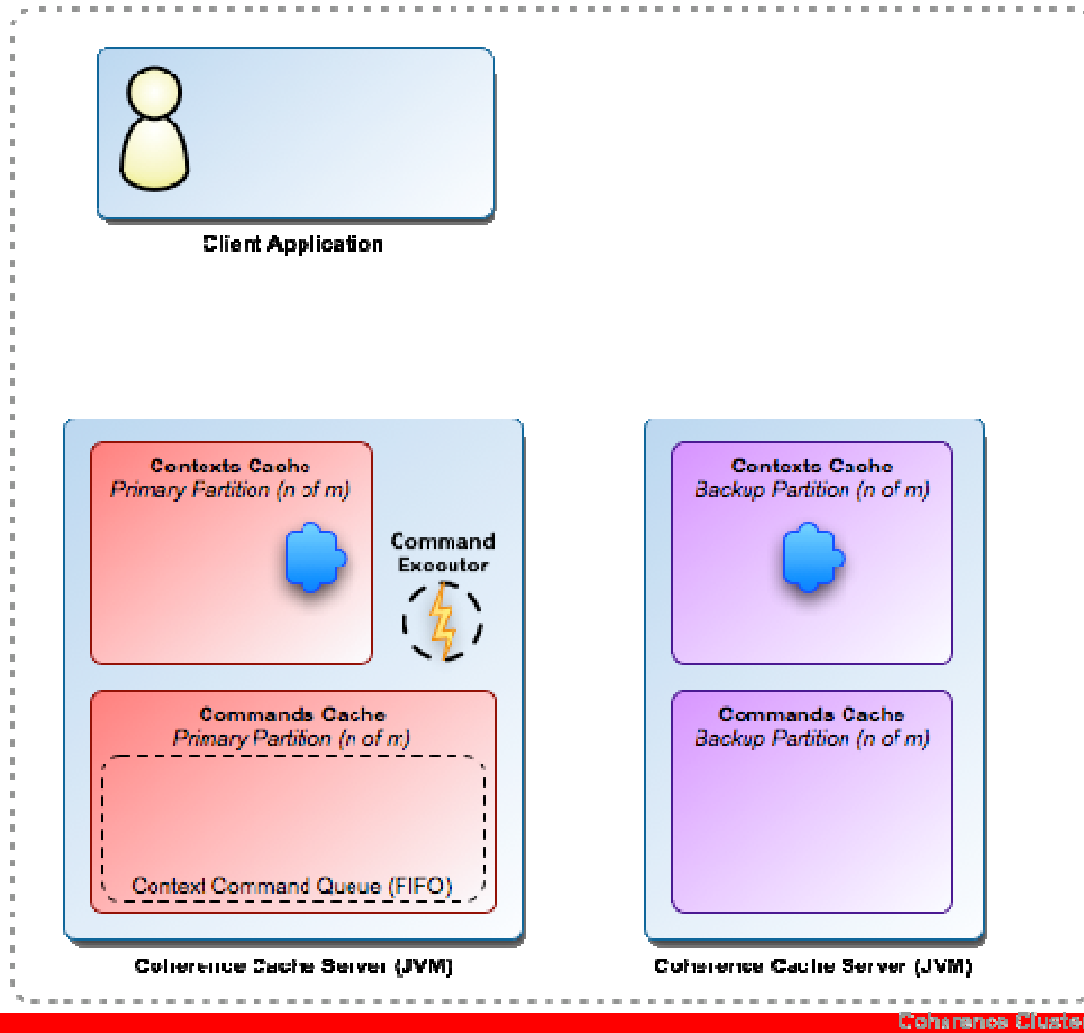
Context Registration



- The client application registers the created Context using a ContextsManager *
- The ContextsManager places the Context into a "primary partition" of the "contexts" Distributed Cache.
- Coherence ensures a backup of the Context is made to a "backup partition" (on separate JVMs and different machines where possible)

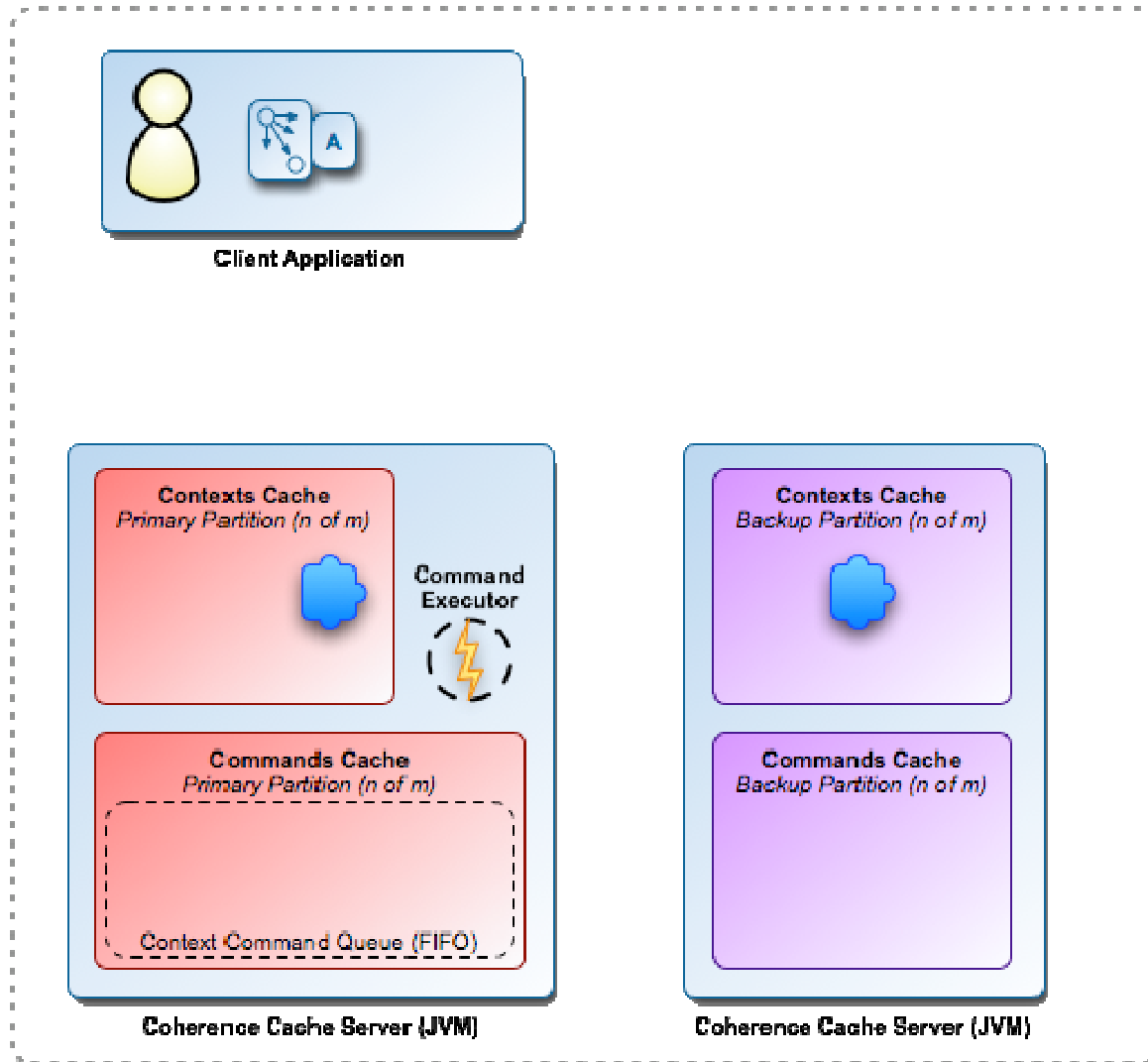
* Typically an instance of the DefaultContextsManager

Establishing the CommandExecutor (automatic)



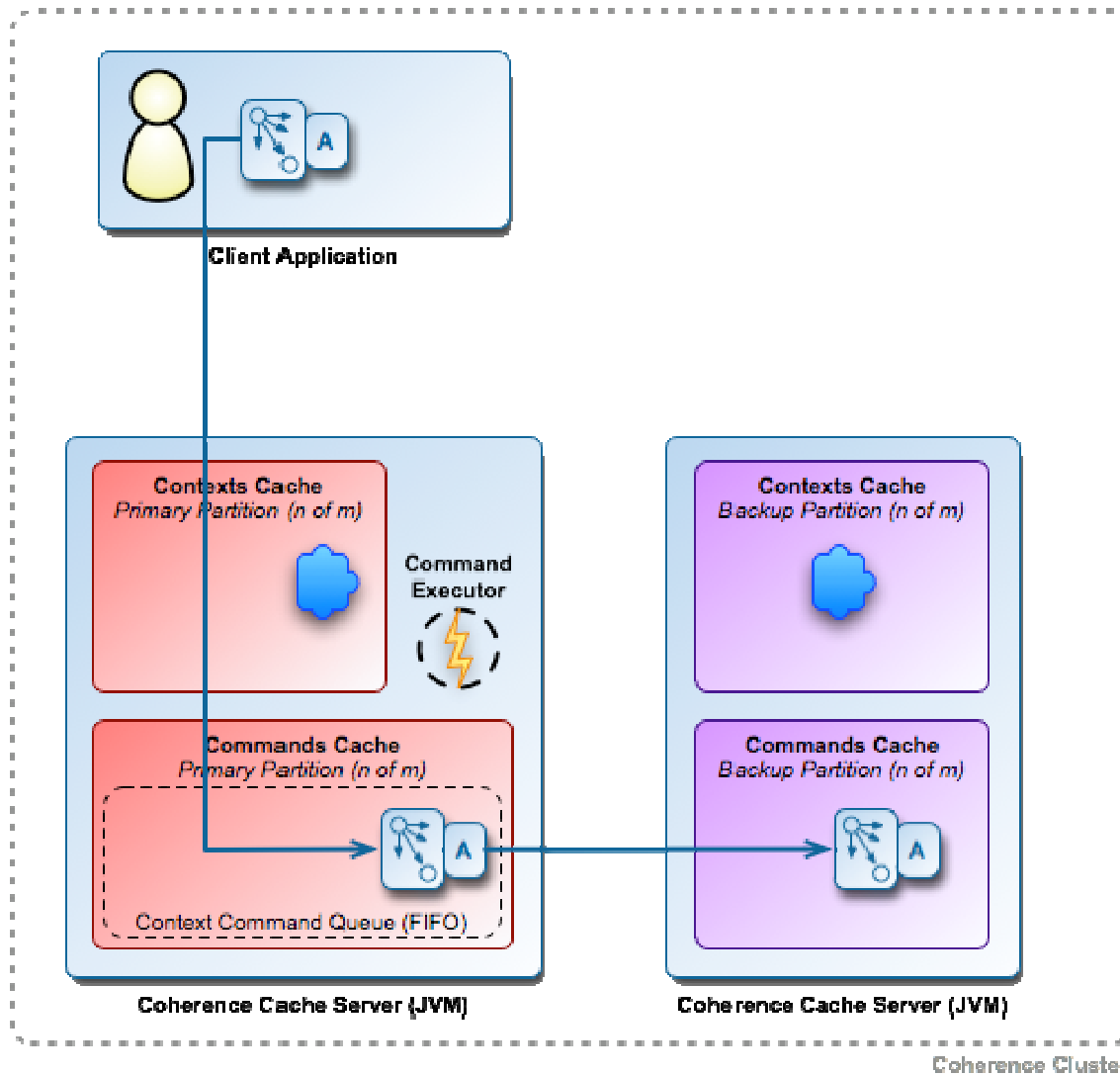
- When a Context is registered, an internal event establishes an appropriate CommandExecutor and necessary infrastructure.

Creating a Command



- The client application creates an instance of a Command

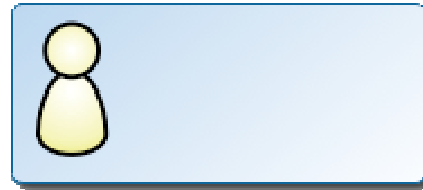
Submitting a Command



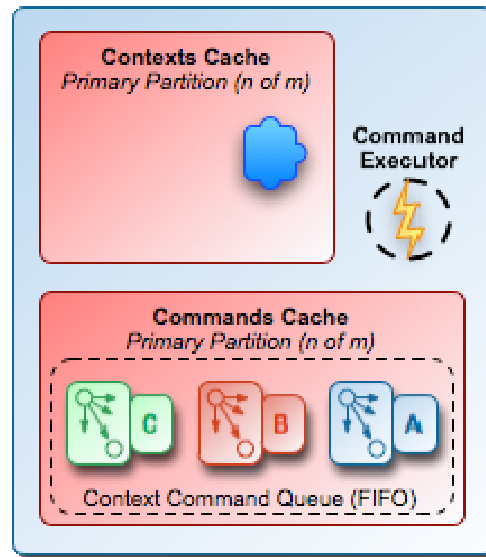
- The client application uses a CommandSubmitter to submit a Command for execution with an identified Context *
- The submitted Command is placed into the "commands" Distributed Cache (and automatically backed up)
- The submitted Command is then automatically queued (FIFO) and scheduled for asynchronous execution by the CommandExecutor for the Context

* An individual Command instance may be submitted any number of times for execution to one or more Contexts. There is no need to create new instances of the same Command if it needs to be submitted for execution multiple times.

Numerous Commands Submitted



Client Application



Coherence Cache Server (JVM)

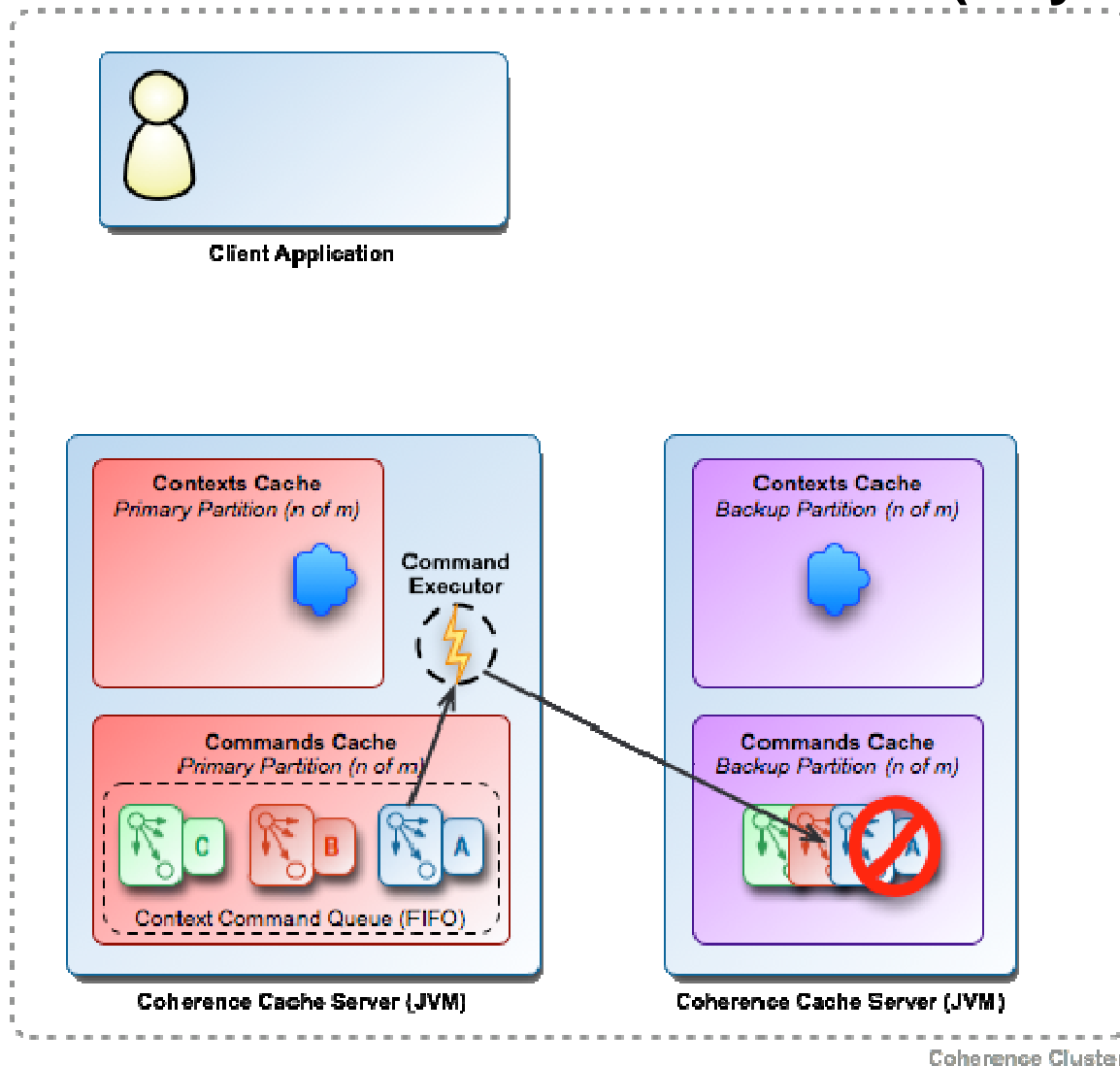


Coherence Cache Server (JVM)

Coherence Cluster

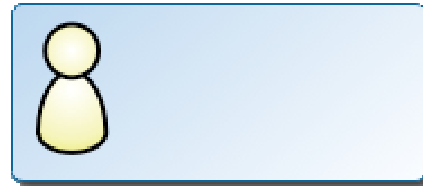
- Multiple Commands may be submitted for execution at once.
- Commands are queued for execution (FIFO) in order of arrival at the Context

Commands are executed by the CommandExecutor (Asynchronously)

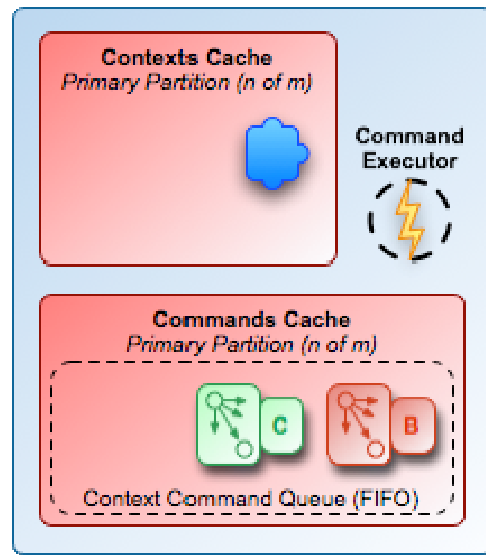


- When Commands are queued for execution, an internal event notifies the CommandExecutor to start executing the Commands
- For efficiency, the CommandExecutor may execute Commands in batches (but remaining in order).

Commands Automatically Cleaned-up when Executed.



Client Application



Coherence Cache Server (JVM)



Coherence Cache Server (JVM)

Coherence Cluster

- Once a Command has been executed, it is removed from the "commands" cache (as is the backup of the Command).



What's the big deal?

- Everyone can do Command Pattern!
- This is distributed computing without all of the effort
 - Embedded in your Application (Grid)
 - No Servers to setup
 - Highly Available
 - Completely Monitorable (via JMX)
- Yours to modify/own/reuse
- Increases the value of an existing Data Grid



The Functor Pattern



The Store and Forward Messaging Pattern



What's the big deal?

- Everyone can do Messaging!
- This is embedded messaging in your application (Grid)
 - No Servers to setup
 - Highly Available
 - Completely Monitorable (via JMX)



The Messaging Pattern

Advocates that;

- Payload, typically represented as a Message object, may be sent to a Destination from a Sender (also commonly known as a Publisher).
- It is the responsibility of the infrastructure managing the Destination to ensure that Messages (arriving at the said Destination) are then stored (in some manner) and consequently forwarded (in the order in which they arrived at the said Destination) to one or more Receivers (also commonly known as Subscribers).



The Messaging Pattern

- The Subscribers appropriately consume (receive and acknowledge receipt) of the said Messages from the Destination in the order in which they were forwarded to the said Subscribers.
- The infrastructure managing the Messages appropriately clean-up (remove and garbage collect) the said Messages that have been consumed by Subscribers.
- The type of the Destination determines the method of delivery to the Subscribers on that Destination.



The Messaging Pattern

- A Topic Destination (or Topic) will store and forward Messages to all of the Subscribers of the said Topic Destination.

This form of Message delivery is often called "publish-and-subscribe messaging", "one-to-many messaging" or "the observer pattern".

- A Queue Destination (or Queue) will store and forward Messages to at most one of the Subscribers of the said Queue Destination.

For each Message a different Subscriber may be used, but this is implementation and runtime dependent. This form of Message delivery is often called "point-to-point messaging" or "one-to-one messaging".



The Messaging Pattern

- **A Message may be Persistent or Non-Persistent.**
In the case of Persistent Messages, the infrastructure managing the Destination must safely store the said Messages to a persistent (and recoverable) storage device so that in the case of infrastructure failure, Messages may be recovered (not lost).
- **A Subscriber to a Topic is either Durable or Non-Durable.**
Durable Subscriptions allow the system implementing the Subscriber to terminate and return without losing Messages that may have been delivered during the outage (or disconnection).



The Messaging Pattern

Next Release

1. Full support for Topics and Queues
2. Support for Durable and Non-Durable Subscriptions
3. Support for Transactions (by subscribers)

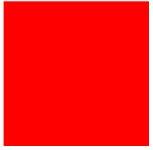


The Push Replication Pattern



What's the big deal?

- Push Replication Pattern solves the “I want this data somewhere* else” problem.
 - In the order that it was inserted/updated/deleted
 - Managed and sent in batches
 - With automatic failover / recovery
 - Between multiple sites / devices
 - In multiple directions
 - Asynchronously
 - With pluggable conflict resolution



Thanks...



ORACLE IS THE INFORMATION COMPANY