
ORACLE®

ORACLE

Caching Schemes & Accessing Data

Lesson 2

Objectives

After completing this lesson, you should be able to:

- Describe the different caching schemes that Coherence offers
- Understand their uses
- Understand how to create a NamedCache
- Understand how to access and update data in the cache

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!

Topologies at a glance

- Local Scheme
- Replicated Scheme
- Distributed Scheme
- Near Scheme

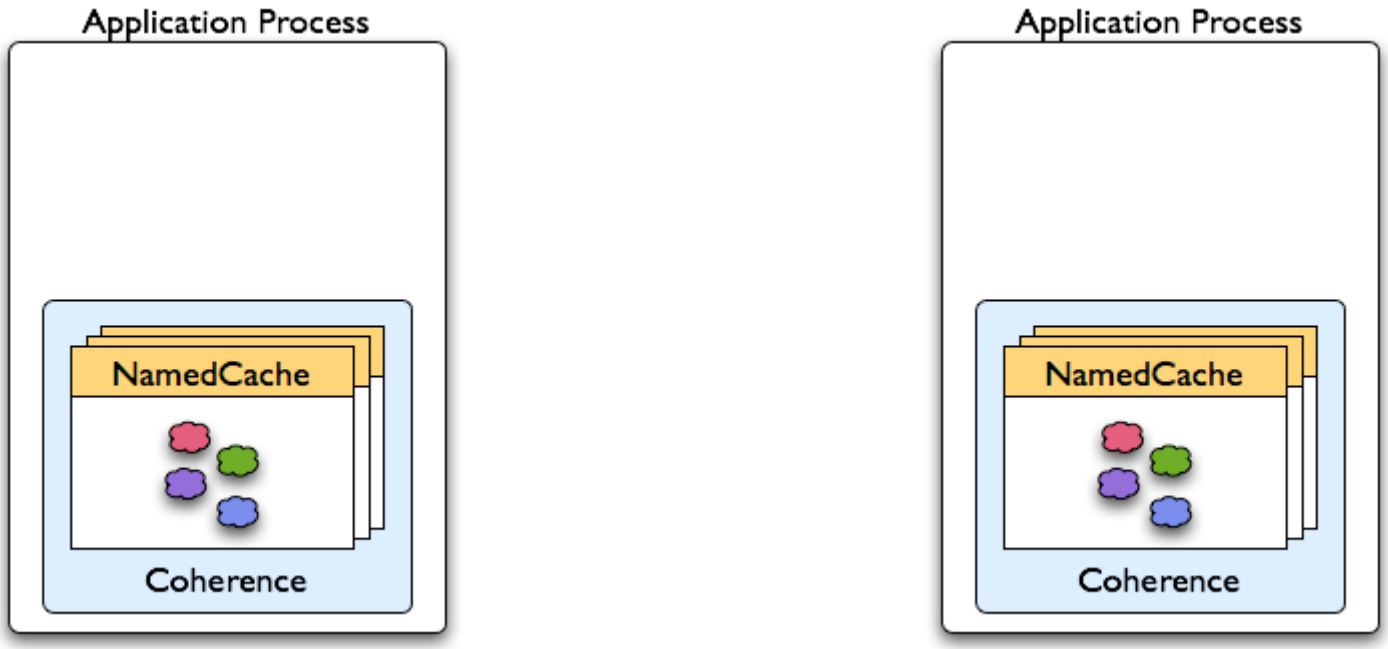
Coherence Schemes

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



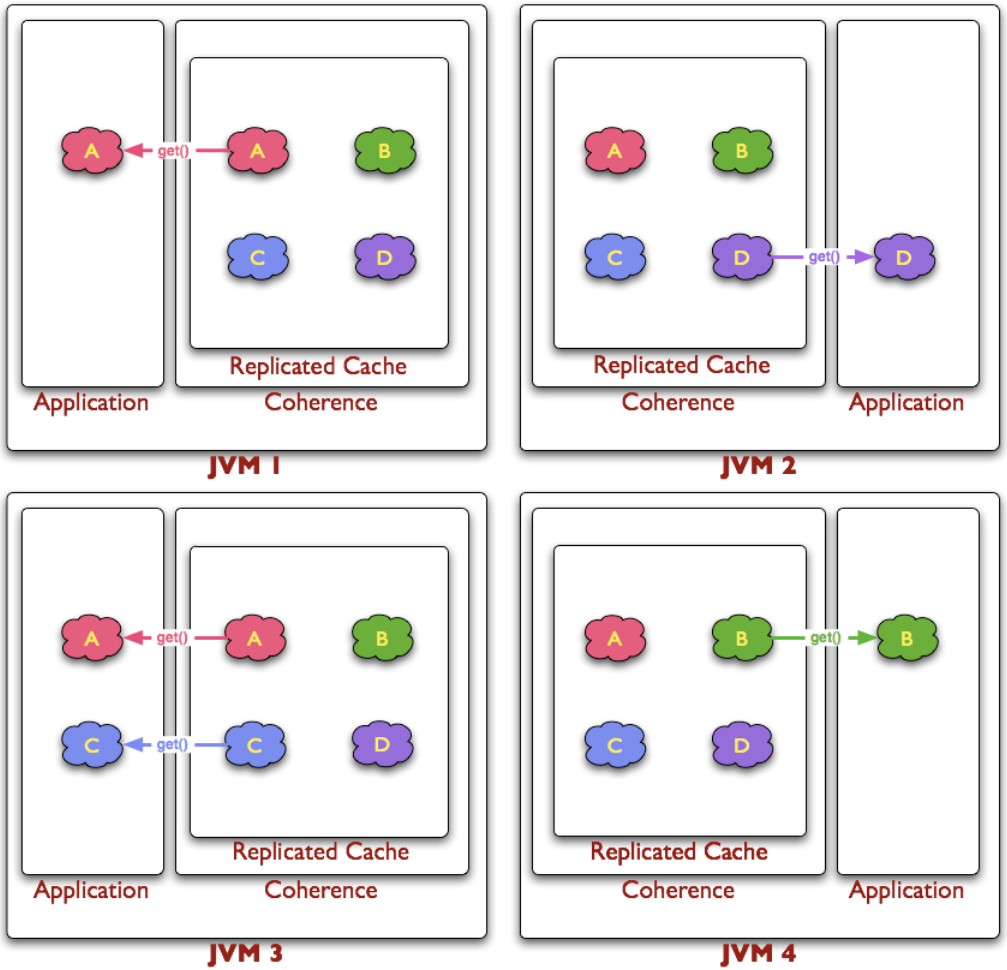
Coherence Schemes

The Replicated Scheme

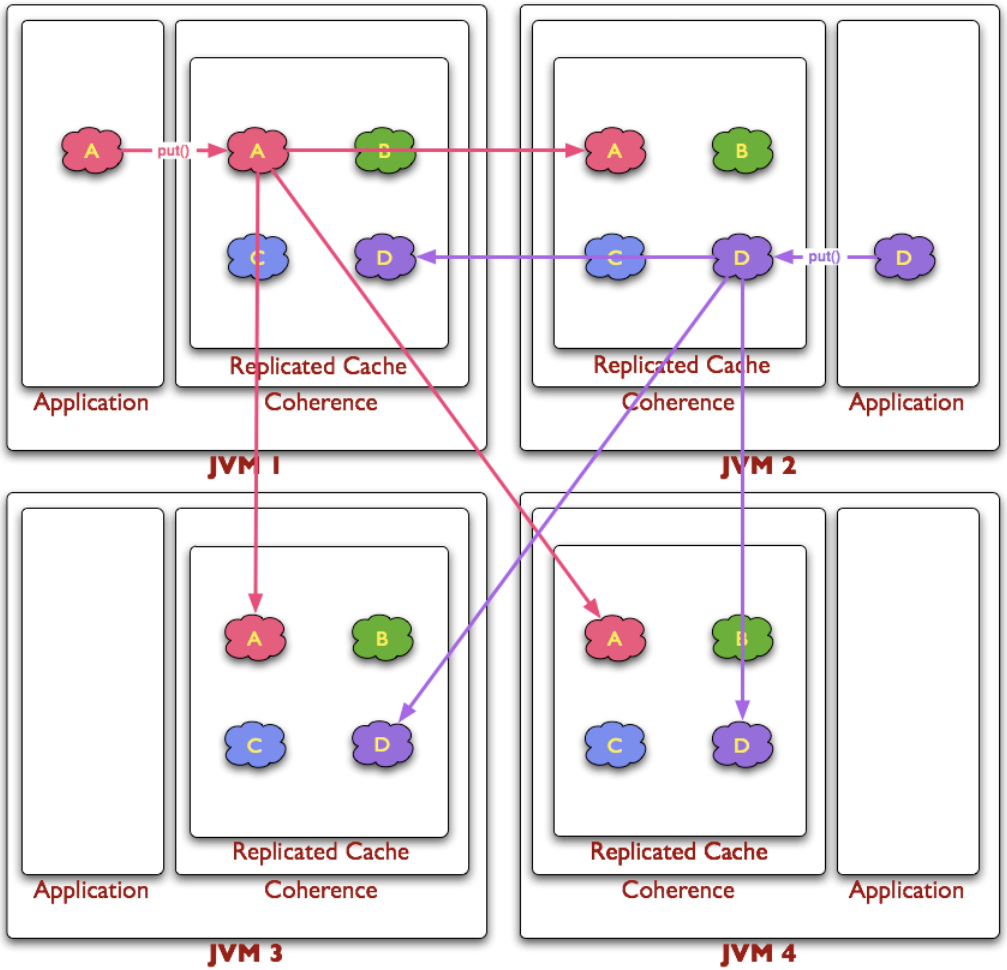
The Replicated Scheme

- Bruce-force implementation of Clustered Caching
- Why:
  - Designed for extreme read performance
- How:
  - Replicate and maintain copies of all entries in all Members
  - Zero latency access as all entries are local to Members
  - Replication and syncing process transparent to developer
- Configurable Expiration Policies:
  - LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

The Replicated Scheme



The Replicated Scheme



The Replicated Scheme

- Cost Per Update
  - Each Member must be updated!
  - Not scalable for heavy writes!
- Cost Per Entry
  - Each Entry consumes Nx memory (N = #Members)
  - 1x for each Member
  - Not scalable for large caches!

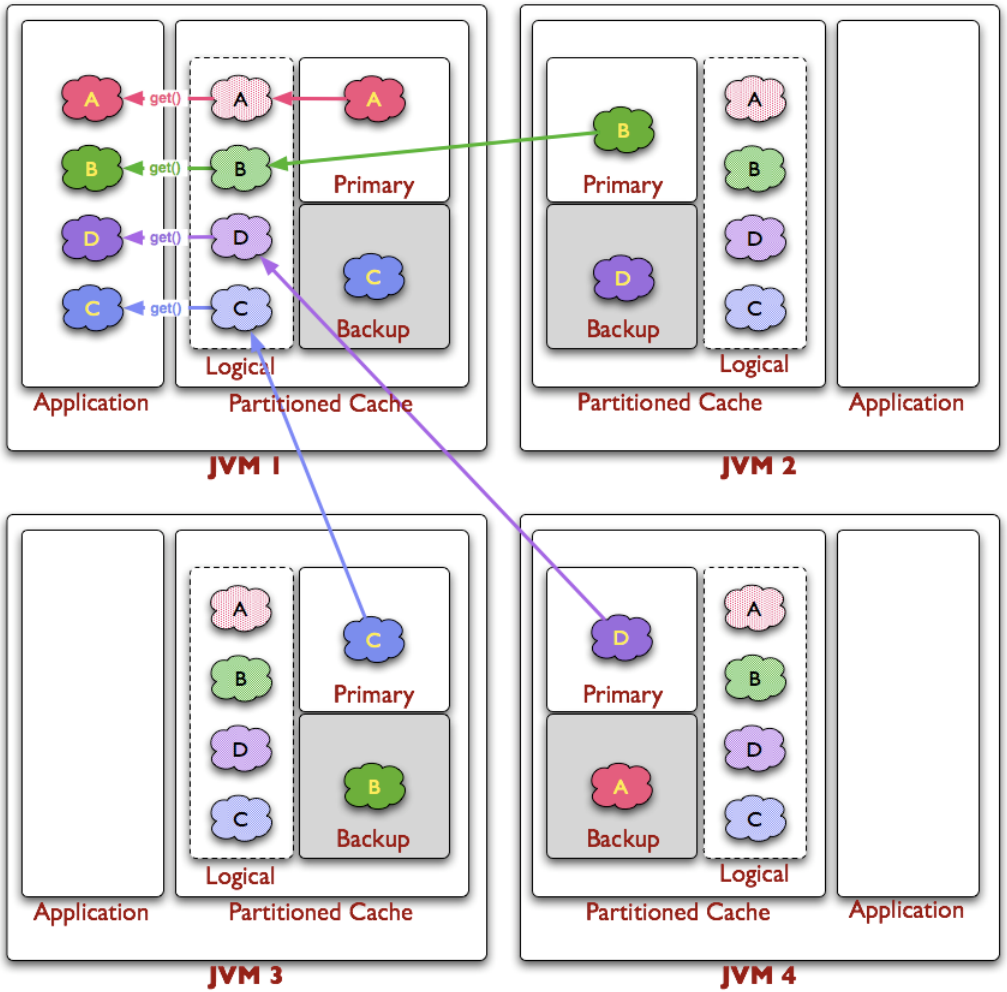
Coherence Schemes

The Distributed Scheme

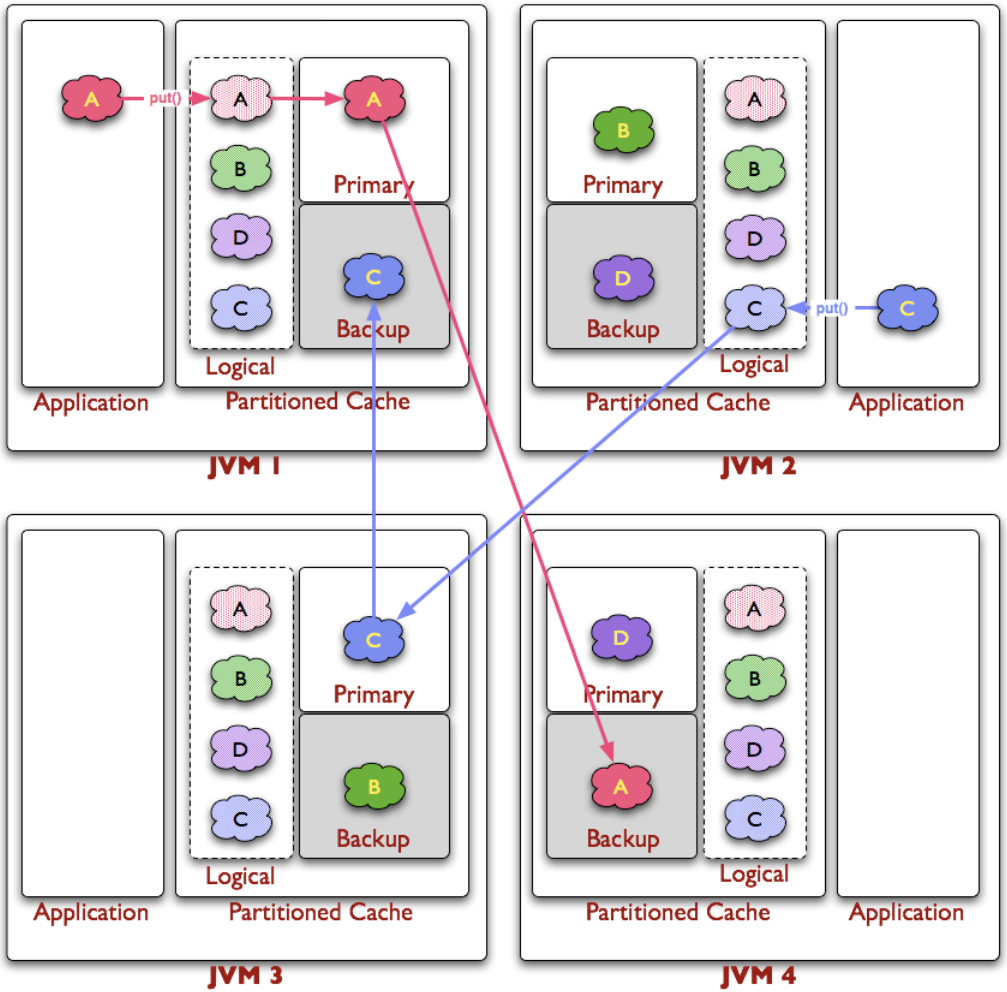
The Distributed Scheme

- 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



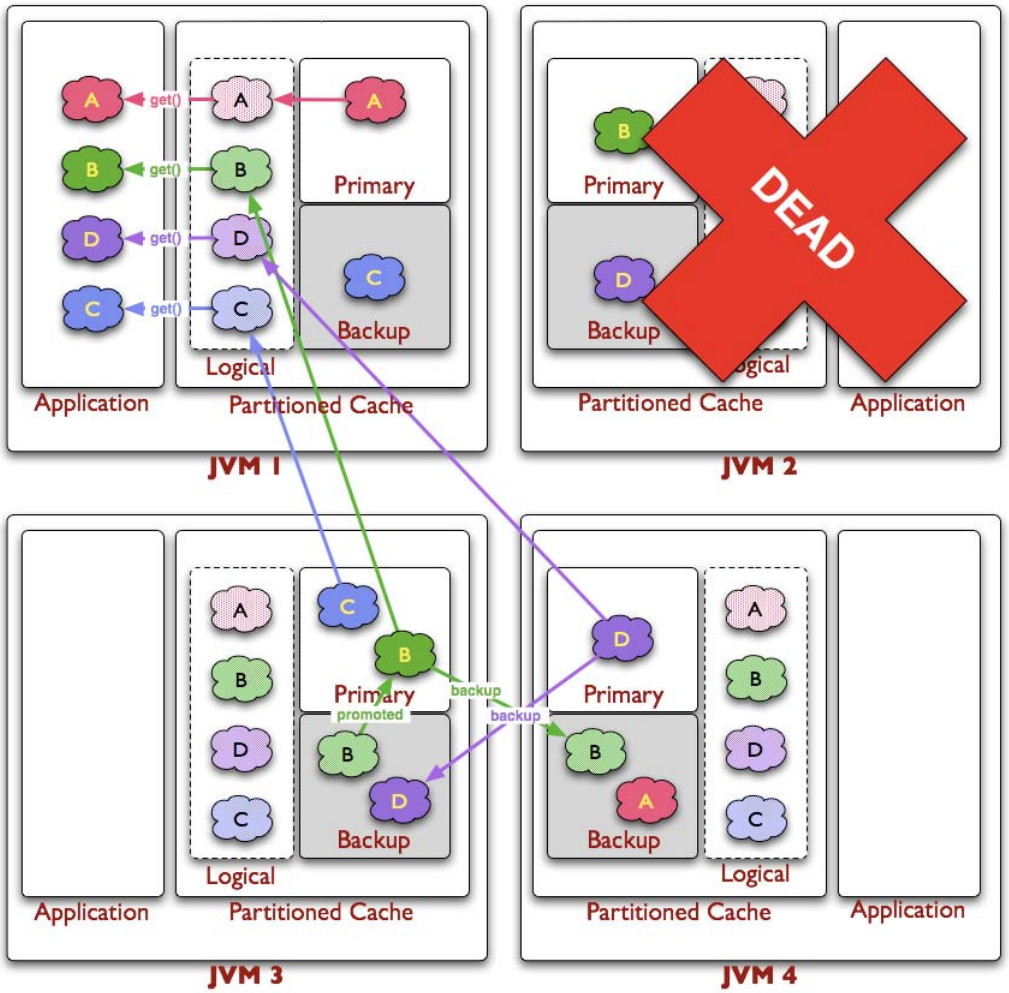
The Distributed Scheme



The Distributed Scheme

- Each Member has logical access to all Entries
  - At most 2 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!

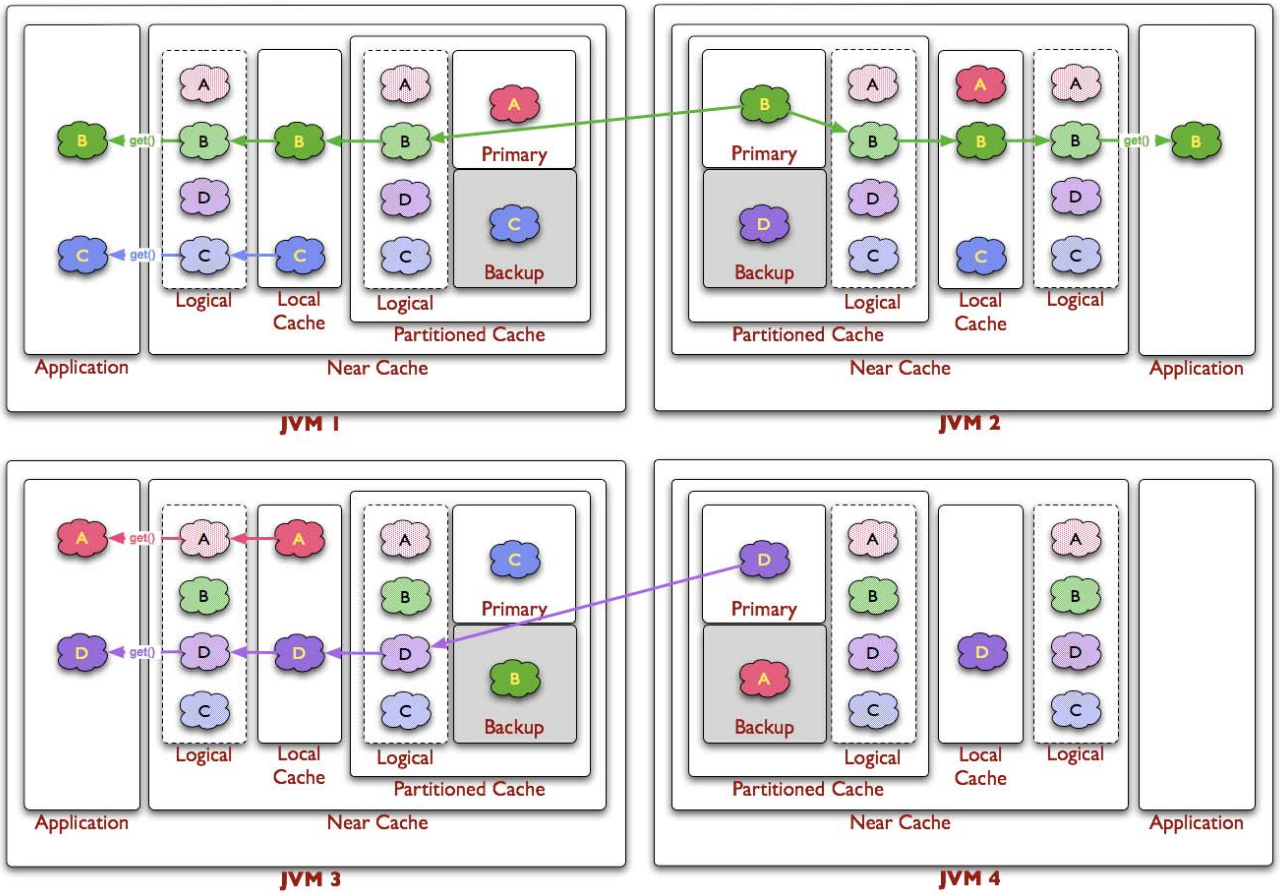
Coherence Schemes

The Near Scheme

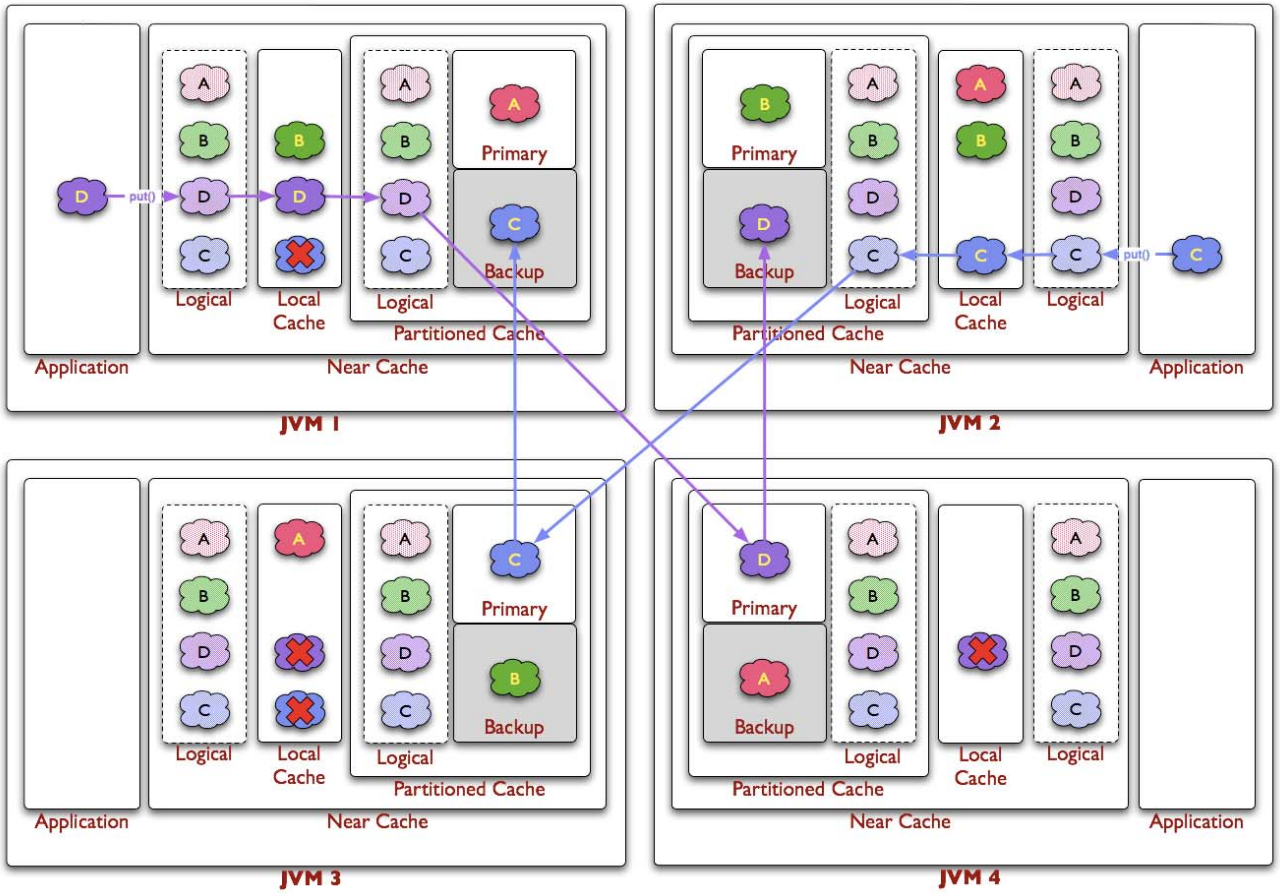
The Near Scheme

- 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



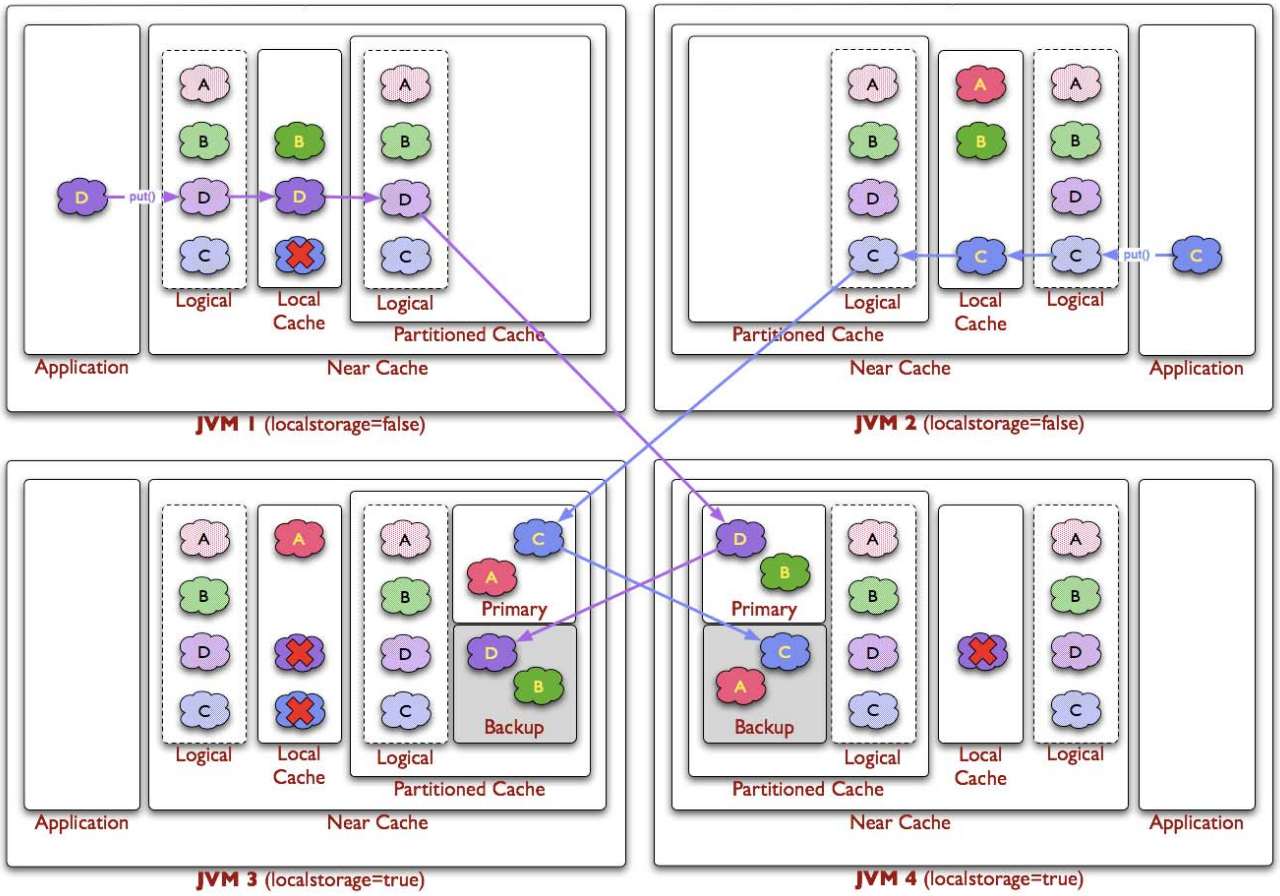
The Near Scheme



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



Accessing & Updating data in a cache

- The CacheFactory is a topology agnostic way to access NamedCaches
- It provides:
  - Mechanisms to manage underlying Cluster Instance
  - Mechanisms to manage Membership lifecycle
  - Mechanisms to work with NamedCaches transactionally (not covered in this course)
- Useful methods

```
static Cluster ensureCluster()
static void shutdown()
static NamedCache getCache(String sName)
```

Accessing & Updating data in a cache

- To create a named cache:

```
CacheFactory.ensureCluster();
NamedCache myCache = CacheFactory.getCache("employees");
```

- The NamedCache interface implements java.util.Map, so you can use the standard map methods such as:

  `get, put, putAll, size, clear, lock, unlock…`

- Also implements JCache

Some useful NamedCache methods

- **Object put(Obejct key, Obejct value)** – put an object in the named cache. (Blocking call)
- **Object get(Obejct key)** – get the entry from the named cache for that key
- **void clear() –** removes all entries from the named cache
- **boolean containsKey(Obejct key)** – returns true if the named cache contains a entry for the key
- **booelan containsValue(Object value)** – returns true if there is at least one entry with this value in the named cache
- **Object remove(Obejct key)**
- **Set entrySet()** – returns a set of key, value pairs
- **Collection values()** – gets all values back as a collection

Accessing & Updating data in a cache

- To put data into the cache use:

```
myCache.put(“Name”,”Tim Middleton”);
```

- To retrieve data use:

```
String name = (String)myCache.get(“Name”);
```

Clustered Hello World

```
public void main(String[] args) throws IOException {
   NamedCache nc = CacheFactory.getCache(“test”);
   nc.put(“key”, “Hello World”);
   System.out.println(nc.get(“key”));

   System.in.read();  //may throw exception
}
```

- **Joins / Establishes a cluster**
- **Places an Entry (key, value) into the Cache “test” (notice no configuration)**
- **Retrieves the Entry from the Cache.**
- **Displays it.**
- **“read” at the end to keep the application (and Cluster) from terminating**

Summary

In this lesson, you should have learned how to:

- Describe the different caching schemes that Coherence offers
- Understand their uses
- Understand how to create a NamedCache
- Understand how to access and update data in the cache

Labs 3 & 4

- Lab 3
  - Create Java classes to access data in a Coherence data grid
- Lab 4
  - Leaving strings behind – create a Person object to store within the data grid


ORACLE®