Coherence Training

TANGOSOL

Administration

Emergency Procedures

TANGOSOL

Administration

Breaks, Lunches, Internet etc...

Introductions

Me

Introductions

You?

Content Sneak Peak...

- Caching: Back to Basics
- Coherence Topologies
- Coherence Deep Dive
- Beyond Caching - Data Grids

Next Module...

- Caching (back to basics)
- To use Coherence we need...
  - To understand caching
  - To understand the challenges involved
  - A common caching vocabulary

TANGOSOL™

Module 1: Caching (back to basics)

Caching & Terminology
Why Caching is Challenging
Local, Farm & Clustered Caching
What is Coherence?
Your First Coherence Experience

TANGOSOL™

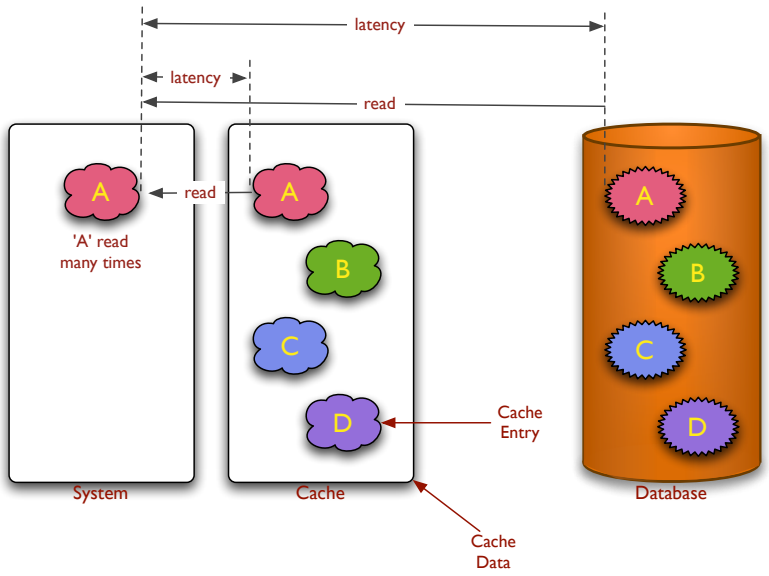
Introduction

- Caching is a Ubiquitous Technology
  - Desktop Applications
  - Enterprise Systems
  - Server Software
  - Databases
  - Operating Systems
  - CPUs
  - Memory
  - Disks
  - Networks

Caching

- Goal:
  Improve system performance when accessing high-latency Data Sources
- How?
  Maintain access efficient copies of data from a high-latency Data Source in a Cache
- Why?
  Accessing a low-latency copy instead of accessing a high-latency Data Source dramatically improves system performance.
- Where?

Cache



Latencies not to scale!

TANGOSOL

Caching

- Cache: *Contains copies of (some) data*
- Caching: *Act of maintaining copies*
- Copies: *Typically maintained in memory / on local disk*
- Access: *Typically Read-Only*
- Latency: *Interval* between requesting and receiving
- Data Sources:
  Disks, Databases, Other Systems & Software, Algorithms...

TANGOSOL

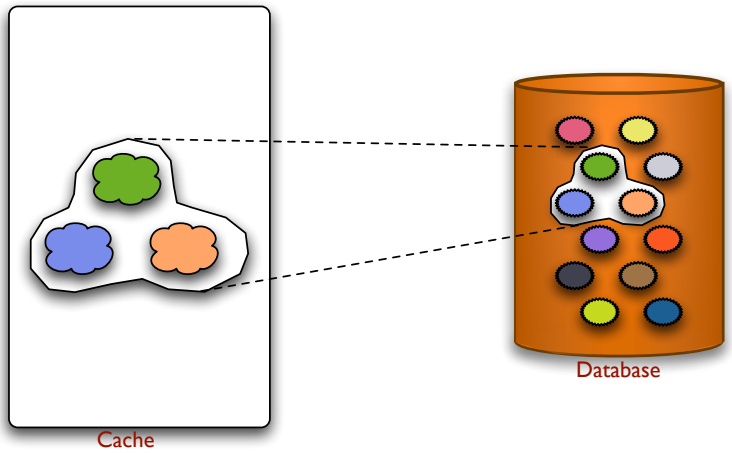
... or Caching is...

- Caching:
  Protecting a high-latency Data Source from over utilization
- Used when:
  System performance is directly impacted by Data Source latency

Caching Terminology

- Cache Data:
  The data held in a Cache from a Data Source
- Cache Entry (entry):
  An individual element of Cached Data consisting of a Cache Value and Cache Key.
- Cache Value (value):
  Actual element of Data from a Data Source
- Cache Key (key):
  A unique identifier for the Cache Entry in a Cache

Cache Window



Caching Terminology

- Cache Capacity:
  The number of Cache Entries a Cache ***may hold***
- Cache Size:
  The number of Cache Entries in a Cache at a ***point in time***
- Rule:
  Cache Size ≤ Cache Capacity

Cache Entry Lifetime

- Entries have Lifetimes (in a Cache)
- Rationale:
  Can't keep all Data Source data in Cache!
- Systems typically only require recently used / accessed data

TANGOSOL

Caching Terminology

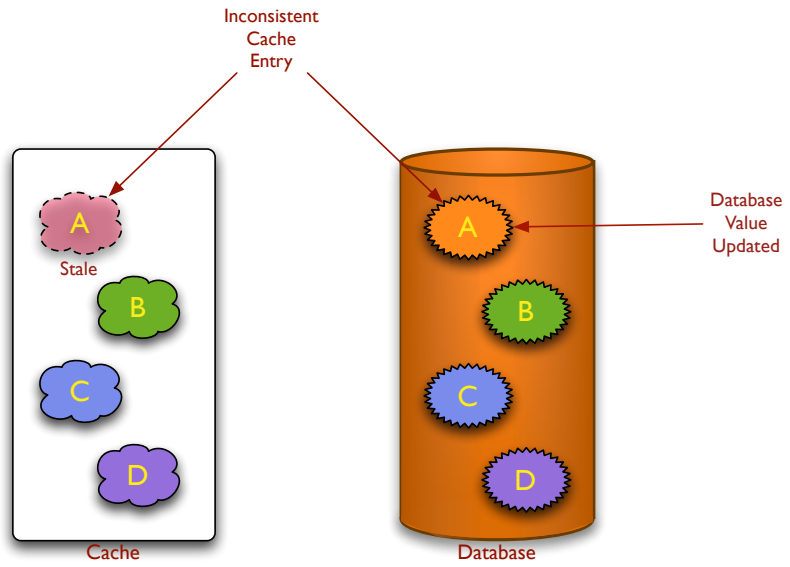
- Cache Entry Expiry:
  Instant when an Entry violates an Expiration Policy governing Entry lifetimes
- Expiration / Invalidation Policy:
  A rule to determine when a Cache Entry Expires

  Based on a time limit, access pattern (LRU/LFU) or Cache Size.
- Cache Entry Eviction / Invalidation:
  The process to remove Expired Cache Entries from a Cache

TANGOSOL

Cache Consistency

Cache Consistency



Cache Consistency

- A Cache is 'Consistent' when Entries are logically equivalent to underlying Data Source data.
- Stale Entry:

  A Cache Entry that is older from its associated Data Source data.
- Caches may be partially consistent (or inconsistent)
- Challenges:
  - Systems that depend on Consistent Caches!
  - Inconsistent Caches may lead to unpredictable results

Cache Consistency

- Requirement is domain specific
  - Not all Systems require Consistent Caches
  - Partial / Complete inconsistency acceptable if understood
  - Inconsistency possibly tolerable for short periods

Cache Consistency

- Tolerating Inconsistency:
  - Internet Search Engines.... Google
  - Delayed Market Prices... Bloomberg
  - Email Applications... Outlook
  - Internet Sites...
- Can your users tolerate inconsistency?
- Can parts of your systems be inconsistent?

Avoiding Cache Inconsistency

Avoiding Inconsistency

- Option 1: Don't Cache
  - Always refer to Data Sources for Data
  - System and Data Source performance impact!

- Who does this?

Avoiding Inconsistency

- Option 2: Regularly Expiry Cache Entries
  - Use small time frame for Expiry Policy
  - Expire often to 'catch updates'.
  - The 'refresh' strategy
  - Does not guarantee consistency
  - Cache may not be effective

- Who does this?

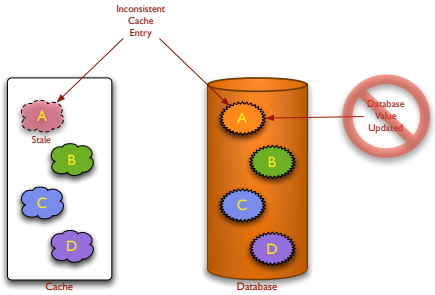
Avoiding Inconsistency

- Option 3: Force Expiry
  - Use 'messaging' to notify Cache of Data Source updates
  - 'Push' to force Entry Expiry
  - Does not guarantee consistency
- Who does this?

TANGOSOL

Avoiding Inconsistency

- Option 4: Avoid behind-cache updates
  - Best Option
  - Update Cache first, then Data Source
  - Does not guarantee consistency
    - Now Data Source not up-to-date
  - May not be architecturally possible
- Who does this?



TANGOSOL

Why is this a Challenge?

Reasoning About State

Identity Rule (I)

- All concepts / objects in a 'system' are uniquely identifiable

- No two objects have the same identity

TANGOSOL

Referential Integrity Rule (RI)

- When multiple parties converse about an identifiable object, the said object is the same for each party
- Parties naturally assume there is integrity when referring to objects by their identity
- Integrity of a conversation between parties is guaranteed by object identity

TANGOSOL

Singularity Rule (S)

- Only a single instance of an identifiable object exists in a system.

TANGOSOL

Caching Breaks Singularity!

- Caching Creates Copies!

- Caching breaks the Singularity Rule!
- We may not use standard reasoning about state!
- We deliberately build infrastructure to cope with this!
  - eg: Ensuring copies are kept consistent

TANGOSOL

Have you...

- Experienced data corruption due to staleness / versioning?
- Wrote code to maintain the singularity rule?
- Used a 'foreign key'?
- Used 'messaging' to invalidate system state?
- Used 'triggers' to notify of state changes?

Breaking Singularity

- Singularity is broken when:
  - System state is maintained in two or more places
  - When we use **Caching**
  - When we use a **Database**
  - When we use **Threads**
- Broken Singularity means:
  - We have to understand the 'versions' of system state
  - We are tempted to use 'transactions'
- Simplest way to avoid this
  - Ensure all state in a single-threaded single process (JVM)

Everything is Broken!

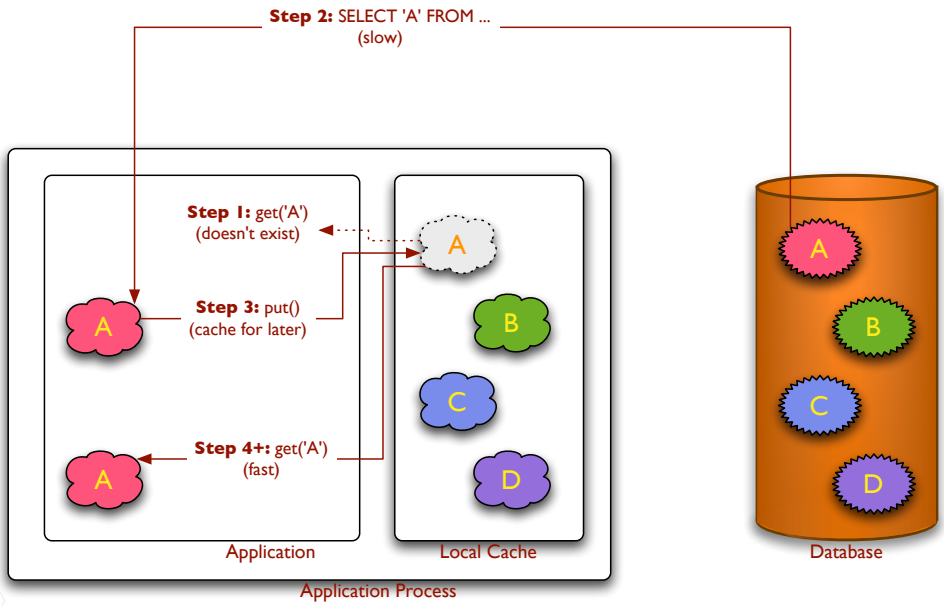
Forms of Caching

Local Caching

Local Caching

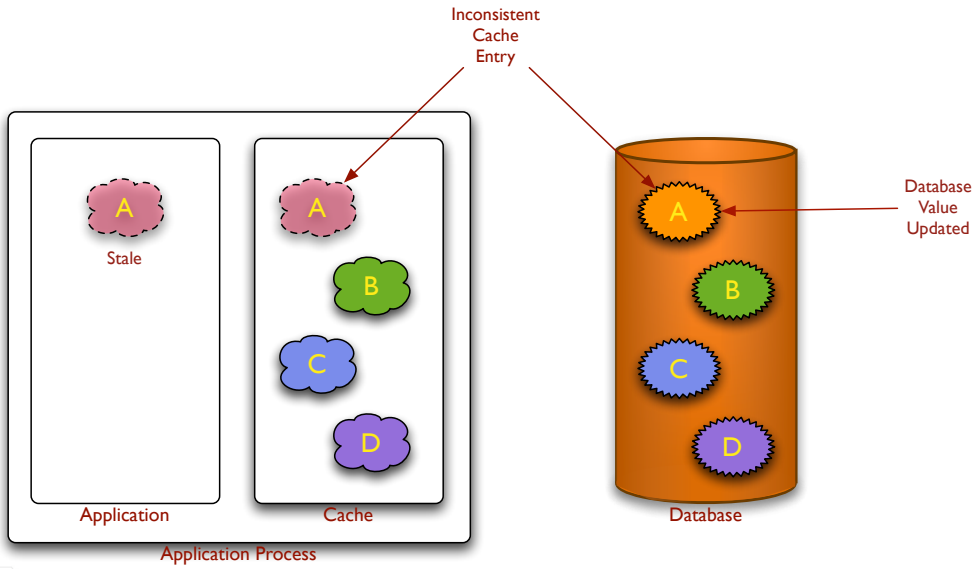
- What we consider 'normal' when discussing Caches
- Cache is part of and internal to Application / Device
- Benefits:
  - Easy to construct
  - Use a Map-based Structure (Key,Value) pairs
- Constraints:
  - Size is extremely limited!
  - Does not guarantee consistency!
  - In multi-threaded Applications, each thread may hold a copy from Local Cache!

Local Caching



TANGOSOL

Inconsistent Local Cache



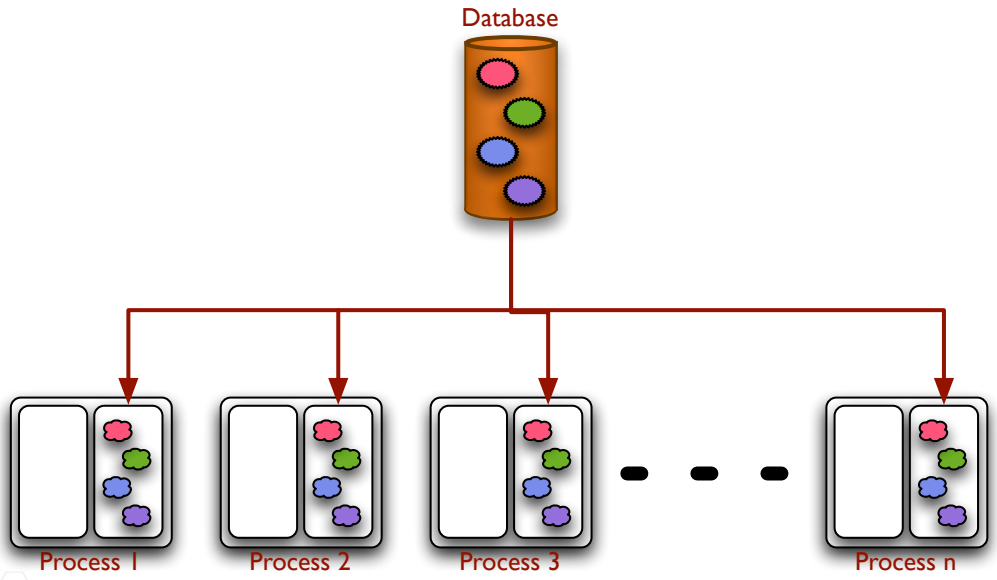
TANGOSOL

Farm Caching

Farm Caching

- Farm:
  A collection of homogeneous applications / devices working independently of each other
- Farm Caching:
  Each member of farm has an independent Local Cache
- Who uses this?
- Where do you use it?
- Why do you use it?

Farm Caching



TANGOSOL

Farm Caching

- Benefits:
  - Same as Local Cache
  - May now scale out
- Constraints:
  - Same as Local Cache - but now worse - across Farm!
  - Singularity broken between members (Incoherent)
  - Members have own copies of Entries
  - No cost savings in making copies to members
  - Cache capacity doesn't increase with Farm size

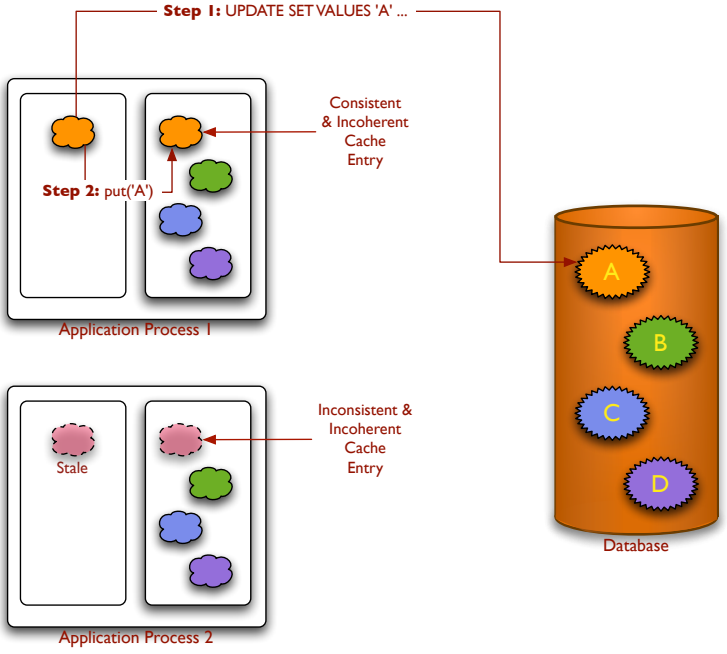
TANGOSOL

Consistency & Coherency

- Consistent Cache:
  Cache Entries logically consistent with Data Source

- Coherent Cache:
  Associated Cache Entries across Members are logically consistent

Consistency & Coherency



Consistency & Coherency

- Coherent & Consistent:
  Associated Entries coherent across Members and consistent with Data Source
- Coherent & Inconsistent:
  Associated Entries coherent across Members but (partially) inconsistent with Data Source
- Incoherent & Inconsistent:
  Associated Entries incoherent across Members and (partially) inconsistent with Data Source

What is Acceptable?

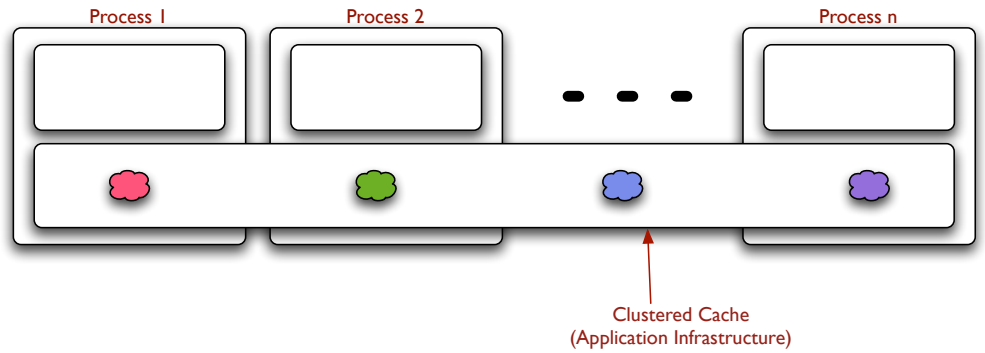
Clustered Caching

Clustered Caching

- Cluster:
  A collection of homogeneous applications / devices working together to perform some task that involves sharing information

- Clustered Caching:
  Each Member of the Cluster shares the responsibility to manage and provide Caching to an Application

Clustered Caching



TANGOSOL

Clustered Caching

- Single System Image (SSI):
  - Each Member logically has access to all Cache Entries
  - Location of Entries typically transparent to Members
  - Entries managed by Members
- Different Clustered Cache Topologies used to implement SSI.

TANGOSOL

Coherent Clustered Caching

- Coherent Clustered Cache:
  All Cluster Members have access to same Cache Entries (therefore coherent)
- Underlying Cache Topology ensures Coherency.
- Coherent and Consistent Cache behavior across Cluster
- No 3rd Party Messaging Solution Required!
- Clustered Cache = Application Infrastructure

Clustered Caching

- Benefits:
  - May scale-out like Farm Caching
  - Coherency guaranteed across Cluster Members
  - While ensuring Coherency isn't 'free', better than accessing high-latency Data Source
- Constraints:
  - Singularity still broken between Members due to network latencies

Clustered Caching Topologies

- Replicated:
  - Each Member has complete copy of all Cache Entries
- Master / Slave:
  - Master Member has copy of all Cache Entries
  - Slave Member used for fail-over and recovery
- Distributed / Partitioned:
  - Each Member owns a partition of the Cache Entries
  - Requires mechanism to find Cache Entries in partitions*

Tangosol Coherence?

What is Tangosol Coherence?

- Coherent Clustered Caching Solution
- Infrastructure for constructing Scalable Stateful Applications
- Core Technology:
  - Reliable, Scalable, Low-Latency Java-based Clustering Infrastructure
  - Service based implementation
  - Supports many shared state programming models - like clustered caching
  - Best of breed IRIS solution for Distributed Applications
- Coherence Caching APIs built upon Core Technology

What is Tangosol Coherence?

- Coherence is an implementation of JCache (JSR107)
- Tangosol CEO (Cameron Purdy) is JSR Spec Lead
- Coherence provides more than Clustered Caching
  - Analytics across Entries
  - Transactions against Entries
  - Query entries
  - Event Handling of Entry modifications
  - Processing Entries

Coherence Programming Model

- Core API based on Java Map data-structure API
  - (key,value) pairs
  - get, put, remove, contains, containsKey, size operations
  - random access
  - drop-in replacement for home-grown caches ;)
-

Coherence Programming Model

- Extensions to API to support
  - Concurrency Control / Synchronization (locking) primitives
  - Events, Queries, Processing, Aggregation etc.
  - Storage of Cache Entries
- Term 'map' often used instead of 'cache'
-
- **Why?**

Exercise: Command Line Tool

Coherence Command Line

- Provides ability to work with Clustered Caches
- Without writing an Application!
- Part of standard Coherence distribution
- in %TANGOSOL_HOME%/bin
- coherence.sh (*nix)
- coherence.cmd (windows)

- **Training Versions (in class materials)**
  - coherence-training.sh
  - coherence-training.cmd

Coherence Command Line

```
prompt:~/coherence-home: .bin/coherence-training.sh

Map (?): help

Map (?): cache test

Map (test): put message "hello world"
null (why?)

Map (test): get message
hello world

Map (test): size
1

Map (test): bye
prompt:~/coherence-home:
```


Coherence Command Line

- Now start two instances of coherence-training.sh
- What happens when you use get & put between instances of the same Cache (map)?
- What happens to your data if you exit one of the instances?
- What happens if you start another instance?

Exercise: Cache Server

Coherence Cache Server

- Head-less (no display / GUI / front-end) Application
- Stores Cache Data between Application Instances!
- Part of standard Coherence distribution
- in %TANGOSOL_HOME%/bin
- cache-server.sh (*nix)
- cache-server.cmd (windows)

- **Training Versions (in class materials)**
  - cache-server-training.sh
  - cache-server-training.cmd

Coherence Cache Server

```
prompt:~/coherence-home: .bin/cache-server-training.sh

Group{Address=224.3.2.0, Port=32363, TTL=0}

MasterMemberSet (ThisMember=Member(Id=2, Timestamp=2006-12-12
17:33:41.455, Address=10.0.1.2:8089, MachineId=2818)
OldestMember=Member(Id=1, Timestamp=2006-12-12 17:28:06.053,
Address=10.0.1.2:8088, MachineId=2818))

Services ( ... )

^C (to stop)

prompt:~/coherence-home:
```


Coherence Cache Server

- Shutdown coherence-training.sh instances
- Start two instances of cache-server-training.sh
- Start a new coherence-training.sh instance
- Put some data into your test cache
- What happens to your data if you exit all coherence-training.sh instances?
- What happens if you start another instance?
- What happens if you start another cache-server-training.sh?

Development Exercise

Development Exercise

- Examine the CacheFactory class (see API)
- Construct a simple Console Application
- Use a CacheFactory to create a simple NamedCache
- Put some values (Strings) into it
- Get some values (Strings) from it
- What happens when your application exits?
- What happens when you use a Cache Server?
- What is the life-time of your values?

Setting up your IDE

- Add the following to your class path
  - %TANGOSOL_HOME%/lib/coherence.jar
  - %TANGOSOL_HOME%/lib/tangosol.jar

Clustered Applications

- Clustered Caches manage state (as POJOs)
- Lifetime of POJOs in a Clustered Cache = lifetime of Cluster
- Lifetime of Cluster ≠ Application Process Lifetime
  - Unit tests may break because of this
- While single node clusters (across processes) are possible, developers must test in a multi-node cluster!
- Remember the Network!
  - Network is slow
  - What is the latency between inter-process communication across a network?

Questions

- Name two areas in your current systems that violate IRIS
- How do you attempt to resolve this? (without Coherence)
- How could Coherence help?
- What is the difference between a Cache Value and a Cache Entry?
- What is Entry Invalidation?
- What is an Eviction Policy?
- What is the life-time of a Coherence Cache Entry?

Next Module

- Cache Topologies
- To use Coherence...
  - We need to be aware of Cache Topologies
  - Each Topology has it's own performance and scalability characteristics

Module 2: Cache Topologies

Local, Replicated and Partitioned Topologies
Data Storage
Topology Composition
Near Topology
Cache Naming

TANGOSOL

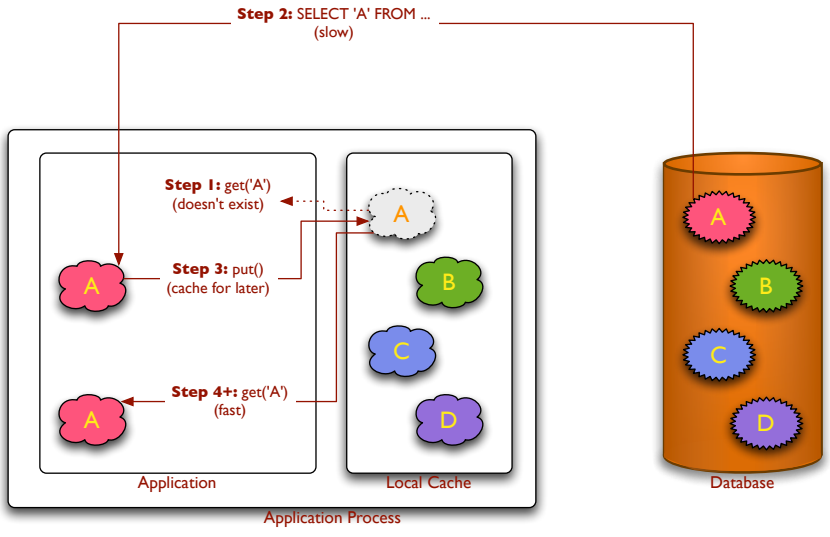
Local Topology

TANGOSOL

Local Topology

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

Local Topology

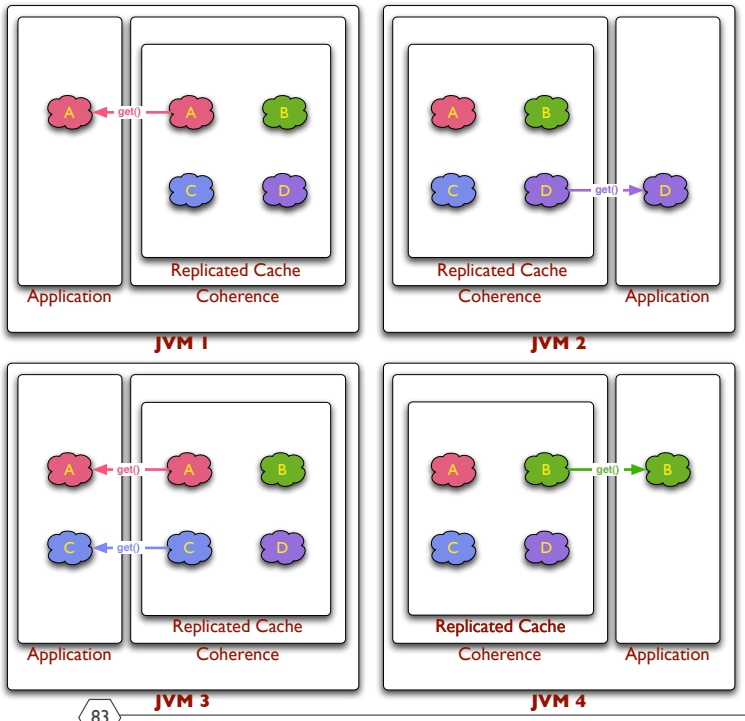


Replicated Topology

Replicated Topology

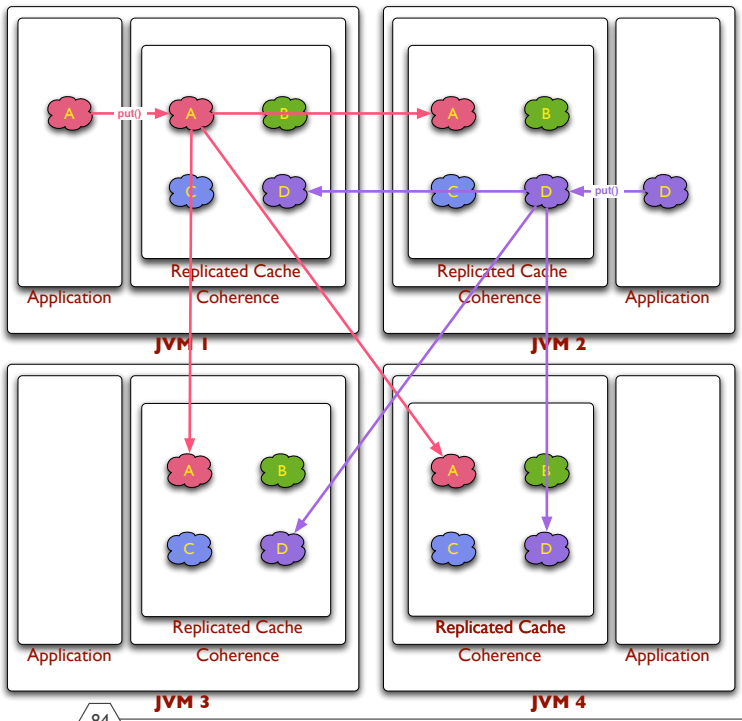
- What:
  Bruce-force implementation of Clustered Caching
- Why:
  Designed for extreme read performance
- How:
  Replicate and maintain copies of all Entries in all Members
  Zero latency access as all Entries are local to Members
  Replication process transparent to developer
- Configurable Expiration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

Replicated Topology



TANGOSOL

Replicated Topology



TANGOSOL

Replicated Topology

- Cost Per Update
  - Each Member must be updated!
  - Not scaleable for heavy writes!
- Cost Per Entry
  - Each Entry consumes *N*x memory (*N* = #Members)
  - 1x for each Member
  - Not scalable for large caches!
- **Any** technology based on replication has these limits!

Quick Quiz

- What happens if simultaneously...
  - JVM 1 is updating A
  - JVM 4 is reading A

Race Conditions...

"A race condition or race hazard is a flaw in a system or process whereby the output of the process is unexpectedly and critically dependent on the sequence or timing of other events" - wikipedia

The term originates with the idea of two threads racing each other to influence the output first

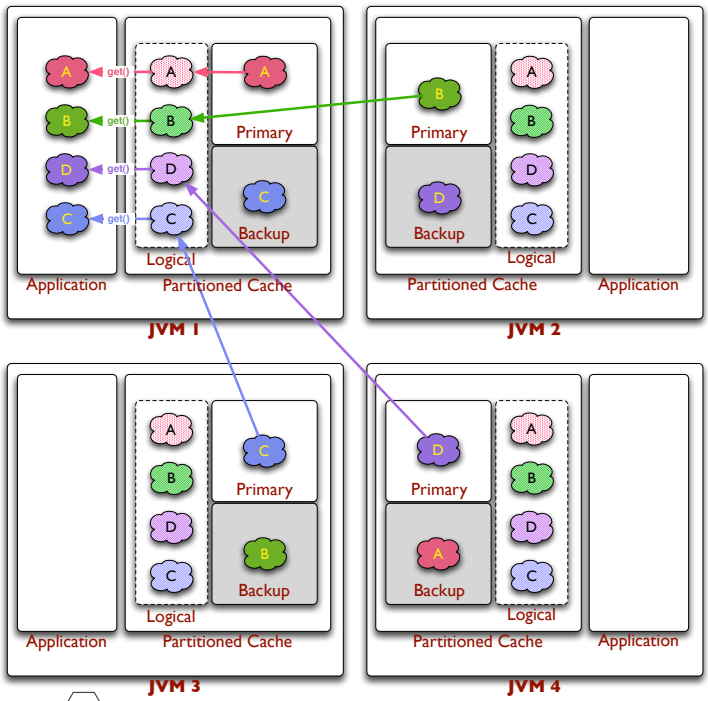
Partitioned Topology

Partitioned Topology

- What:
  Sophisticated approach for coherent scalable clustered caching
- Why:
  Designed for extreme scalability
- How:
  Transparently partition, distribute and backup cache entries across Members
- Often referred to as 'Distributed Topology'
- Configurable Expiration Policies:
  LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

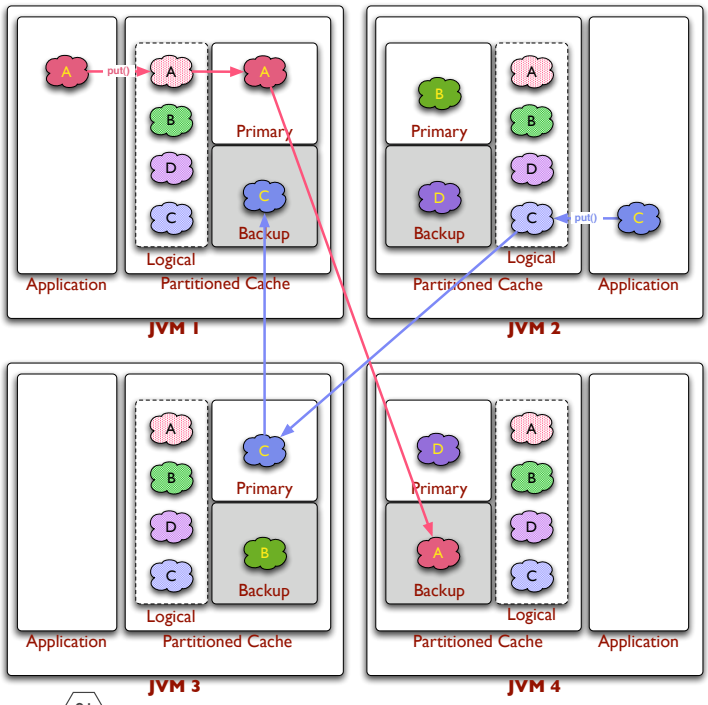
TANGOSOL

Partitioned Topology



TANGOSOL

Partitioned Topology



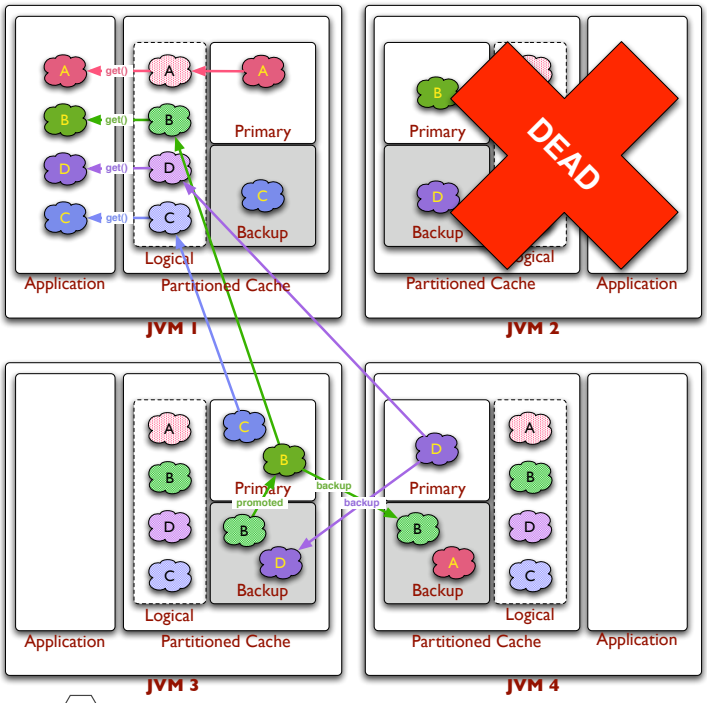
TANGOSOL

Partitioned Topology

- Single System Image
  - 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 Scaleability
  - Cache Capacity Increases with Cluster Size
  - Coherence Load-Balances Partitions across Cluster
  - Point-to-Point Communication
    - No multicast required

TANGOSOL

Partitioned Failover



TANGOSOL

Partitioned Failover

- Seamless Failover and Failback
  - Backup 'promoted' to be Primary
  - Primary 'makes' new Backup(s)
- Invisible to Application
  - Apart from some latency on entry recovery
- Recovery occurs in Parallel
- Any Member can fail without data loss
- Configurable # backups
- No Developer or Infrastructure intervention

TANGOSOL

Partitioned Topology

- Benefits:
  - Deterministic Access and Update Latency (regardless of Cluster Size)
  - Transparent Failover & Failback
  - 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) (solution next ;)

Partitioned Topology

- 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!
  - A one-to-many recovery pattern (occurs in parallel)
  - Not a sequential JVM-based one-to-one recovery pattern
- Cache still operational during recovery!

Question: How could a JVM die?

Partitioned Data Storage

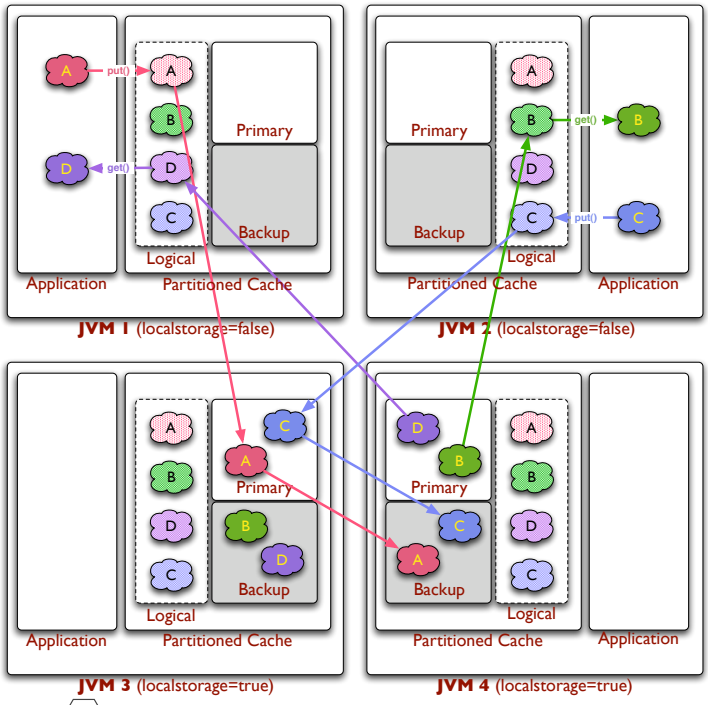
Partitioned Data Storage

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

Partitioned Data Storage

- 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

Partitioned Data Storage



TANGOSOL

Topology Composition

TANGOSOL

Topology Composition

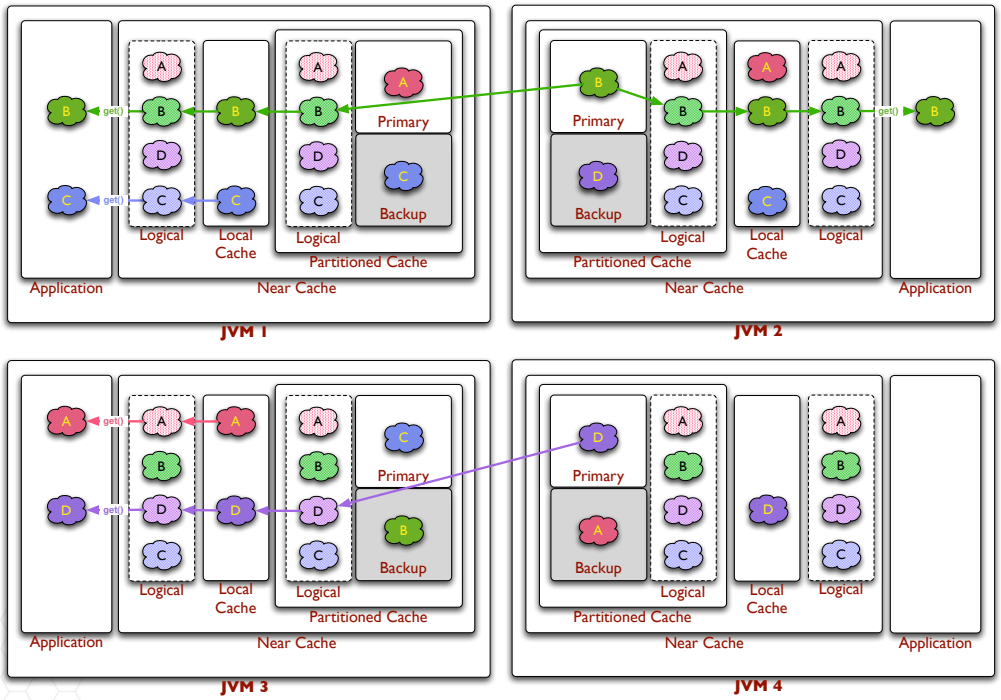
- Coherence Cache Topologies may be 'composed' to form new Cache Topologies to suit address system requirements and SLAs.
- Base Topologies:
  - Local, Replicated, Distributed, Extend*
- Composite Topologies:
  - Near, Overflow (to disk)
  - Allow other topologies to be 'plugged in' to form new Topologies

Near Topology

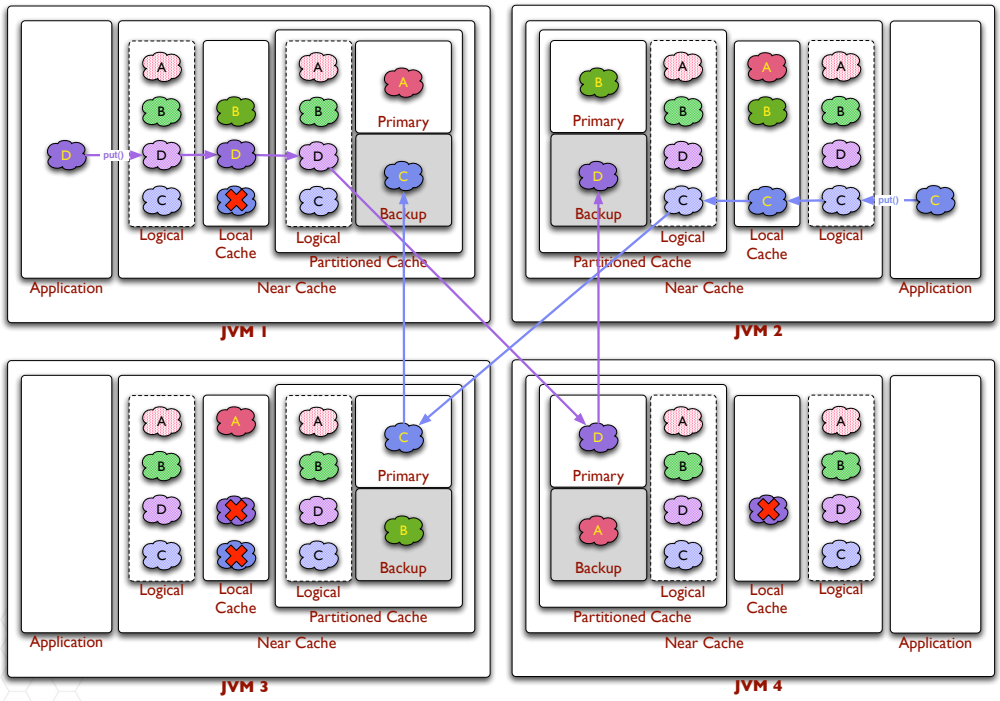
Near Topology

- What:
  Combine Front and Back topologies to provide L1 and L2 caching (cache of a cache)
- Why:
  Partitioned Topology may always go across the wire
  Need a local cache (L1) over the Partitioned Topology (L2)
  Best option for scalable performance!
- How:
  Configure 'front' and 'back' topologies (covered later)
- Configurable Expiration Policies:
  LFU, LRU, Hybrid (LFU+LRU), Time-based, Never, Pluggable

Near Topology



Near Topology



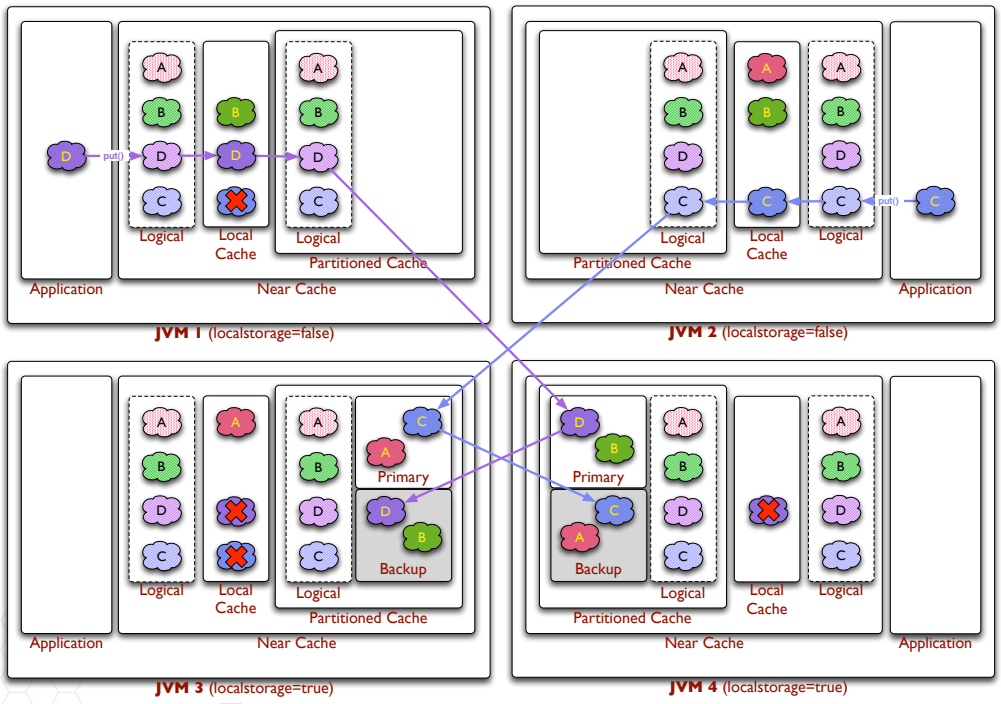
TANGOSOL

Near Topology Coherency

- Local Cache Coherency Options
  - Seppuku: Event-Based 'Kill Yourself' Invalidation
  - Standard Expiry: LFU, LRU, Hybrid, Custom
- Notice: No messaging system required for invalidation!
  - Built into infrastructure
  - High-performance

TANGOSOL

Near Topology



TANGOSOL

Other Topologies

TANGOSOL

Other Topologies

- Replace Front and Back Maps with other Maps
- Replace actual Map implementations with your own
- 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 is extremely configurable
- Configuration and Topologies virtually unlimited
- We'll cover actual configuration later (entire module!)

Out-Of-The-Box Topologies

Out-Of-The-Box

- Coherence Cache Configuration
  - Maps Cache Names to Topologies
  - Ships with out-of-the-box wildcard-based Cache Names
  - Wildcard Cache Names map to out-of-the-box Topologies!
- No need to configure Coherence to use Topologies
- We'll cover cache configuration later

Out-Of-The-Box Cache Names

- dist-* (Partitioned/Distributed Topology)
- near-* (Near Topology)
- repl-* (Replicated Topology)
- opt-* (Optimistic Topology)
- local-* (Local Topology)
- * (Partitioned/Distributed Topology)

Coherence Command Line

```
prompt:~/coherence-home: .bin/coherence-training.sh

Map (?): help

Map (?): cache near-test

Map (near-test): put message "hello world"
null

Map (near-test): get message
hello world

Map (near-test): size
1

Map (near-test): bye
prompt:~/coherence-home:
```


Development Exercise

Development Exercise

- Develop an application to store phone numbers for a collection of people
- Phone numbers can be Strings
- Names of people can be Strings
- Run your application with at least 4 Cache Servers
- Experiment with each Topology
- What are the performance characteristics for each?

Questions

- What is the Optimistic Topology?
- Under what scenarios would you use each of the Topologies mentioned in this Module?

Next Module

- Coherence Deep Dive
  - Caching APIs
  - Object Serialization
  - Concurrency Primitives

TANGOSOL

Module 3: Coherence Deep Dive

- The NamedCache Interface
- Using CacheFactories
- Caching Non-Primitive Objects
- Serialization
- Locking & Synchronization

TANGOSOL

NamedCache Interface

NamedCache Interface

- The NamedCache interface exposes Cache functions
  - com.tangosol.net.NamedCache
- All Coherence Caches
  - Are Named
  - Have Lifetime scoped by the Cluster Instance in which they exist (or member if local)
  - Implement the NamedCache Interface

NamedCache Interface

- Two or more independent Cluster Instances may use the same name for a Cache
  - Cache Entries will not overlap!

Cluster Instance

- Cluster Instance (Cluster):
  One or more Java processes running Coherence and forming a Coherence Cluster

- Multiple Clusters may exist in the same Network
  - eg: Personal, Development, Test, QA may all exist in the same network
  - Take care in configuring Clusters (later) to ensure they are separate
- Each Cluster has Cluster Services
  - Examine trace output from Coherence Cache Server!

Cluster Service

- Cluster Service (Service):
  A Java Daemon running in each Member that provides clustering services to Coherence

- Services provide infrastructure for Coherence features
  - Cluster Service = managed membership
  - Distributed Service = distributed cache management
  - Replicated Service = replicated cache management

- Services are configurable (later)

Cluster Instance

- Each Cluster Instance must share a common Cluster Service configuration

- Each JVM may only belong to a single Cluster Instance
  - Mechanisms exist for accessing multiple Clusters from a single JVM

NamedCache Interface

- Standards Based (JCache - JSR 107)
  - Extends java.util.Map
  - Like Map, get(...) and put(...) are not thread-safe
- Extensions to Map provide other features
  - Locking & Synchronization, Storage Integration
  - Queries, Events, Aggregation, Transactions (beyond caching)
- Implementation (topology) is specified through configuration - swap out without code changes!

NamedCache Interface

```
void clear()

boolean containsKey(Object key)

boolean containsValue(Object value)

Set entrySet()

Object get(Object key)

boolean isEmpty()

Set keySet()
```

NamedCache Interface

```
Object put(Object key, Object value)

void putAll(Map t)

Object remove(Object key)

int size()

Collection values()

Map getAll(Collection colKeys)

Object put(Object oKey, Object oValue, long cMillis)
```


NamedCache Interface

```
void destroy()

String getCacheName()

CacheService getCacheService()

void release()
```

Question: What types of Cache Services Are Available?

Cache Factories

CacheFactory

- Topology agnostic mechanism to access NamedCaches
- Mechanisms to manage underlying Cluster Instance
- Mechanisms to manage Membership lifecycle
- Mechanisms to work with NamedCaches transactionally (covered later)

CacheFactory

```
static Cluster ensureCluster()

static void shutdown()

static NamedCache getCache(String sName)

static NamedCache getCache(String sName, ClassLoader loader)
```

Question: Why is it important to shutdown?

Development Exercise

- Write an Application to determine the Services currently running in a Cluster
- What are the types of Services?
- Does it make any difference if a Cache Service is running as to the type of Services that are available?

Caching Non-Primitive Values

Caching Non-Primitive Objects

- All Cache Keys and Values must be Serializable
  - Each will be transmitted across a process boundary at somepoint in time
- Some objects should not be Cached
  - External resources like threads, file handles, connections, streams etc.
  - However, the content of the above may be Cached

Serialization Refresher

- Implement java.io.Serializable Marker Interface
- Mark non-serializable fields as transient
- Declare a serialVersionUID
- Declare an accessible (usually public) no args constructor
- (optional) Define writeObject and readObject to handle special case class-base serialization requirements.

- Recommended
  - Implement hashcode(), equals() and toString()

Serializable Person Class

High-Performance Serialization

- Java Serialization not fit for purpose!
  - Designed to serialize entire graphs of Objects
  - Designed to handle cyclic references
  - Has a large overhead for each object, class and attribute
  - Has a large overhead for deserialization
- Typical use of Coherence
  - POJOs - not object graphs
  - Minimal Composition Relationships (Parents + Children)

High-Performance Serialization

- Coherence provides:

  com.tangosol.io.ExternalizableLite

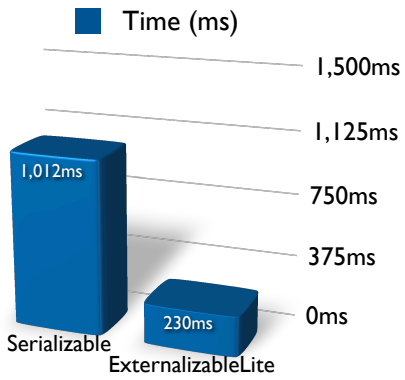
- ExternalizableLite extends java.io.Serializable
  - So you must follow standard Serialization rules!
- Provides low-level access to byte-based serialization stream (not object-based)

ExternalizableLite Interface

```
void readExternal(DataInput in)

void writeExternal(DataOutput out)
```


ExternalizableLite Interface

- ExternalizableLite Serialization
  - Typically 6x faster
  - Resulting serialization is much smaller
  - Emits less garbage for both serialization and deserialization



- Trade-off for Performance (and size)
  - Have to fully implement serialization for each attribute (and ignore transient attributes)
  - Have to understand impacts on versioning of classes

ExternalizableLite Interface

- Where possible Coherence will use ExternalizableLite implementations over Java Serialization
- com.tangosol.util.ExternalizableHelper provides helper methods to serialize objects and collections with ExternalizableLite

ExternalizableLite Person Class

ExternalizableHelper Class

- Provides methods to implement ExternalizableLite
- Read and Write:
  - Primitive Types
  - UTF Strings (safely)
  - Common Composite Types (Arrays, Maps etc)
  - Objects (with null support)
  - ExternalizableLite objects
- Use where possible

Development Exercise

- Develop an Application that serializes an Order object using ExternalizableLite into a 'dist-orders' cache.
- Attributes of the Order class include;
  - orderId
  - orderDate
  - portfolioId
  - stockId
  - side (buy / sell)
  - quantityRequired
  - desiredPrice

Cache Keys

Cache Keys

- Cache Keys must be Serializable or ExternalizableLite
- They should also correctly implement:
  - hashCode()
  - equals()
  - toString()

Cache Key Recommendations

- Create your own Opaque Immutable Type-Safe Keys
  - Don't use Strings / Integers for all Keys!
  - Helps with runtime type-safety
  - Hides implementation (business keys) logic
  - Ensures you capture requirements for IRIS Identity!

Cache Key Recommendations

- Use Helpers / Factory Pattern to create Type-Safe Keys
  - Control access to key creation
  - Allow overloaded / polymorphic key creation
  - Encapsulate creation in a 'Business Service Layer'
  - Future-Proof key implementation
  - Enable multi-valued keys
  - Enable Data Affinity (partitioning override)
    - Data Affinity = keeping related data together (in a partition)
  - Implement ExternalizableLite

Person.Key Class

Development Exercise

- Extend your previous application to create an Order.Key class
- Instances of the Order.Key class will be used as keys for Orders in a Cache.

Locking & Synchronization

Locking & Synchronization

- NamedCache additionally implements ConcurrentMap
- ConcurrentMap provides explicit locking and synchronization primitives for Cache Entries

ConcurrentMap Interface

```
boolean lock(Object oKey)   //lock no wait

boolean lock(Object oKey, long cWaitMillis)

boolean unlock(Object oKey)
```


Locking Semantics

- Locks on block locks (Cluster-wide)
- Locking permitted on non-existent keys!
- Locking equivalent to synchronized(key)
  - Cluster-wide-synchronization
  - Synchronized doesn't work across a Cluster!
- Lock owner may be:
  - Thread of Member
  - Member of Cluster

Locking Semantics

- Re-entrant – but no entrance reference counting.
  - Acquiring multiple locks (from the same lock owner) on the same key ok.
  - Only one 'unlock' required to unlock!
- Blocking
  - Time limited or Infinite
- Non-Blocking
  - Immediate Return

Locking Semantics

- Maximum Lock Lease Time Configurable
- Server Failure:
  - Locks 'recovered' and maintained on 'failover server'
  - Lock owners unaware of failure & recovery
  - No loss of locks on Server failure
- Application / Client Failure:
  - Locks released
  - Always use try-catch-finally blocks around locks!

Locking Semantics

- For repeatable reads (without Transactions)
  - Lock before Get
  - Do it consistently
- Or use a TransactionMap (covered later)

Locking Semantics

- If an application uses locks, ensure that it is done consistently and respects locks!
- Locking follows Java synchronization semantics (but across a cluster)

Quick Quiz

Development Exercise

- Develop an Application that uses locks to update an Order object in a cache
- To perform the update, reduce the desiredQuantity by a set amount
- Create multiple threads to perform the update on a number of orders.

Questions

- How would you develop a cluster-safe counter using locks?
- How do you attempt to lock an entry and wait an indefinite period?
- How do you attempt to lock an entry and wait for the default lease time?
- What may happen if locks aren't in try-catch-finally blocks?
- How many network hops are required to lock and unlock an Object?
- What is repeatable read?

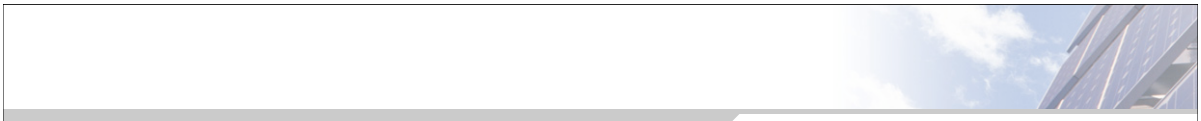
Next Module

- Beyond Caching
  - Observing Data Changes
  - Querying Data
  - Aggregating Data
  - In-Place Processing of Data

Module 4: Beyond Caching

- ObservableMap Interface (Events)
- QueryMap Interface (Queries + Analytics)
- Continuous Queries (Queries + Analytics + Events)
- InvocableMap Interface (Processing + Analytics)

TANGOSOL



ObservableMap Interface

TANGOSOL

ObservableMap Interface

- Provides ability to "observe" changes in Cache Entries
  - `com.tangosol.util.ObservableMap`
- Standard Bean Event Model (Listener Pattern)
  - extends `java.util.EventListener`
- All NamedCaches implement ObservableMap
- History
  - Originally designed to provide pluggable invalidation and cache pruning (internal use)
  - Now used for reacting to Entry changes

ObservableMap Interface

```
void addMapListener(MapListener listener)

void addMapListener(MapListener listener,
                    Filter filter,
                    boolean fLite)

void addMapListener(MapListener listener,
                    Object oKey,
                    boolean fLite)


void removeMapListener(MapListener listener)
void removeMapListener(MapListener listener, Filter filter)
void removeMapListener(MapListener listener, Object oKey)
```

MapListener Interface

```
void entryDeleted(MapEvent mapEvent)

void entryInserted(MapEvent mapEvent)

void entryUpdated(MapEvent mapEvent)
```


ObservableMap Interface

- Register MapListeners for...
  - All cache events, those satisfying a Filter or a specific key
  - Lite == network optimization. (reduce event payload)
- MapListener Interface implementations...
  - Handlers for Insert, Update and Deleted Events
- MapEvent Class captures event information
  - Id (event type), Entry Key, Old Value, New Value
  - Lite means old and new values *may* not be present in event

ObservableMap Interface

```
namedCache.addMapListener(new MapListener() {

    public void entryDeleted(MapEvent mapEvent) {
       //TODO... handle deletion event
    }

    public void entryInserted(MapEvent mapEvent) {
       //TODO... handle inserted event
    }

    public void entryUpdated(MapEvent mapEvent) {
       //TODO... handle updated event
    }

});
```


MapEvent Class

```
//Event Id's (types of event)
static int ENTRY_DELETED
static int ENTRY_INSERTED
static int ENTRY_UPDATED

int getId()  //the Id (type) for the event

Object getKey()  //the key on which the event occurred

ObservableMap getMap()  //map on which the event occurred

Object getNewValue()  //may be null

Object getOldValue()  //may be null
```

MapListener Implementations

- AbstractMapListener Class
  - Empty implementations for each MapListener method
  - Simplify your implementations by overriding default implementations
- MultiplexingMapListener Class
  - Introduces abstract onMapEvent method
  - All MapListener methods delegated to onMapEvent
  - Simplify your implementations by overriding onMapEvent

AbstractMapListener Class

```
namedCache.addMapListener(new AbstractMapListener() {

    //other MapListener methods implemented in super-class

    public void entryUpdated(MapEvent mapEvent) {
       //TODO... handle just the updated event
    }

});
```

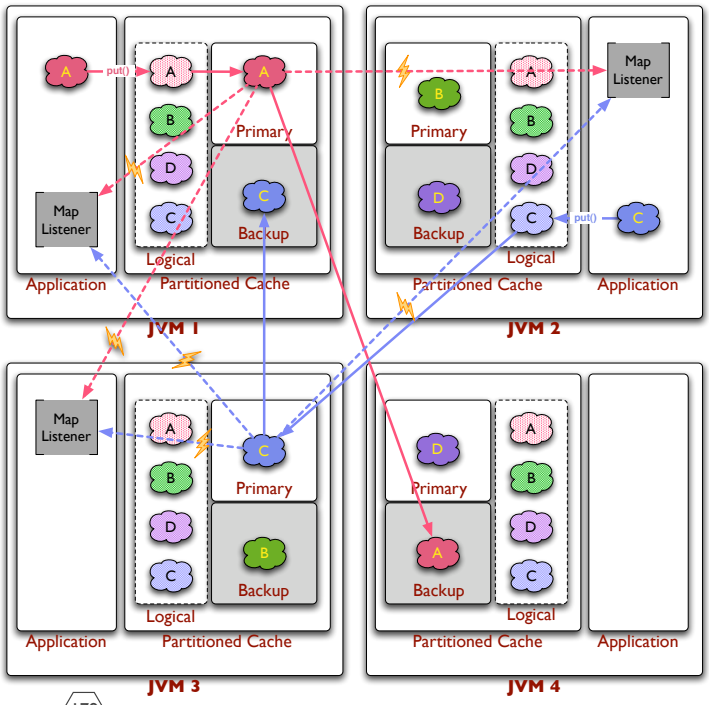
MultiplexingMapListener Class

```
namedCache.addMapListener(new MultiplexingMapListener() {

    public void onMapEvent(MapEvent mapEvent) {
       //TODO... handle all event (use Id to determine type)
    }

});
```


Event Publishing Semantics

- Events are delivered asynchronously to MapListeners from Cache Entry Owners
  - eg: In Partitioned Topology the Primary Partitions (Cache Entry Owners) are responsible for event delivery
  - Clients updating Cache, don't pay cost of event delivery
    - Event delivery is asynchronous from their point-of-view
    - Option exists for synchronous delivery
- Filtering is performed by Cache Entry Owner(s)
  - Only desired events delivered to MapListener implementations!

Event Publishing Semantics



TANGOSOL

Event Publishing Semantics

- Event handlers are executed on Coherence owned Service Threads
- MapListener instances may be registered many times to handle different events (and on different caches)
- Consequently multiple events may be delivered to a single event handler at any point in time

TANGOSOL

Event Publishing Semantics

- Event delivery is **not** guaranteed when;
  - Cluster repartitioning in progress
- Some possibilities:
  - Update acknowledged + events delivered + then primary fails (all ok)
  - Update acknowledged + events not delivered + then primary fails (no events delivered and not retried)
  - Update acknowledged, events partially delivered + then primary fails (some events delivered and not retired)
  - Update completed (not acknowledged) + events partially delivered + then primary fails (some events redelivered)

Event Publishing Semantics

- Order of event delivery is **not** guaranteed (from client perspective) when;
  - Using asynchronous event listeners AND
  - Multiple service threads AND
  - Multiple application threads
- Example (one of many)
  - Place A in Cache A and B in Cache B (in that order)
  - Cache A service threads are experiencing load
  - Cache B service threads not experiencing load
  - Insert events for Cache B arrive before Cache A

Event Publishing Semantics

- Make listeners synchronous to guarantee event delivery order
  - Only from the point-of-view of the mutating client thread
  - Use synchronous listeners with extreme care!
- Example:
  - Clients A, B and C have synchronous listeners on Cache N
  - Client A inserts X, Y & Z into a Cache N (in that order)
  - Client A is guaranteed to receive events in the insert order
  - Clients B and C may receive out of order events.

Event Publishing Semantics

- Making Synchronous Listeners:
  - Implement marker interface

    `MapListenerSupport.SynchronousListener`

  - Wrap Listener with

    `MapListenerSupport.WrapperSynchronousListener`

  - Example:

```
namedCache.addListener(
    new MapListenerSupport.WrapperSynchronousListener(
        ... your listener ...));
```

Questions

- Identify another circumstance in which cache events may be processed out of order.
- Where do MapListener classes need to be deployed?

Filter Interface

Filters

- Filters (`com.tangosol.util.Filter`) are used in many Coherence interfaces, including ObservableMap, QueryMap and InvocableMap
- Together with Indexes, Filters are a very flexible, expressive, scalable and high-performance means to find, view and manipulate data sets
- Typically Filters are always evaluated by the Cache Entry Owners
  - Scaleable (top a point) and minimize network traffic

Filters

- Coherence provides a range of out-of-the-box Filters
  - MapEventFilter (for MapEvents)
  - Reflection-based Filters (for Cache Entries)
    - AllFilter, AlwaysFilter, AndFilter, AnyFilter, ArrayFilter, BetweenFilter, ClassFilter, ComparisonFilter, ContainsAllFilter, ContainsAnyFilter, ContainsFilter, EqualsFilter, ExtractorFilter, GreaterEqualsFilter, GreaterFilter, InFilter, InKeySetFilter, IsNotNullFilter, IsNullFilter, KeyAssociatedFilter, KeyFilter, LessEqualsFilter, LessFilter, LikeFilter, LimitFilter, MapEventFilter, NeverFilter, NotEqualsFilter, NotFilter, NullFilter, OrFilter, PresentFilter, ValueChangeEventFilter, XorFilter ...
- ... or create your own!
- Coherence Filters may be dynamically composed at runtime!

The Filter Interface

```
boolean evaluate(Object object)
```

NOTE: The type of "object" is dependent on the context in which the Filter is being used.

eg: When used to register MapListeners, the type of object provided to the filter will be a MapEvent

eg: When used for Cache Entries, the type of object provided to the filter will be an Entry

Filtering MapEvents

- To filter MapEvents, ensure that the specified Filter expects MapEvents!
  - Otherwise you'll get runtime type errors!
  - eg: Don't do this:
    - addListener(new EqualsFilter("getName","Brian"), ...);
    - As the method "getName" doen't exist on MapEvent

Filtering MapEvents

- Recommendation:
  Use `com.tangosol.util.MapEventFilter` class

- MapEventFilter provides ability to filter
  - types of events, including masks
  - specific changes in underlying cache Entries

The MapEventFilter Class

Event Filter Masks

```
static int E_ALL
Indicates that all events should be evaluated.

static int E_DELETED
Indicates that ENTRY_DELETED events should be evaluated.

static int E_INSERTED
Indicates that ENTRY_INSERTED events should be evaluated.

static int E_UPDATED
Indicates that ENTRY_UPDATED events should be evaluated.
```

The MapEventFilter Class

Event Filter Masks Continued

static int E_UPDATED_ENTERED
Indicates that ENTRY_UPDATED events should be evaluated, but only if additionally specified filter evaluation is false for the old value and true for the new value.

static int E_UPDATED_LEFT
Indicates that ENTRY_UPDATED events should be evaluated, but only if additionally specified filter evaluation is true for the old value and false for the new value.

static int E_UPDATED_WITHIN
Indicates that ENTRY_UPDATED events should be evaluated, but only if additionally specified filter evaluation is true for both the old and the new value.

The MapEventFilter Class

MapEventFilter Constructors

MapEventFilter(int nMask)
Construct a MapEventFilter that evaluates MapEvent objects based on the specified combination of event types.

MapEventFilter(int nMask, Filter filter)
Construct a MapEventFilter that evaluates MapEvent objects based on the specified combination of event types. The specified filter processes Map.Entry and not MapEvent

MapEventFilter Examples

A filter that evaluates to true if an Employee object is inserted into a cache with a value of Married property set to true.

```
new MapEventFilter(MapEventFilter.E_INSERT,
    new EqualsFilter("isMarried", Boolean.TRUE));
```

A filter that evaluates to true if any object is removed from a cache.

```
new MapEventFilter(MapEventFilter.E_DELETE);
```


MapEventFilter Examples

A filter that evaluates to true if there is an update to an Employee object where either an old or new value of LastName property equals to "Smith"

```
new MapEventFilter(MapEventFilter.E_UPDATED,
    new EqualsFilter("LastName", "Smith"));
```

Exercise

- Develop a simple chat client
- Develop a simple console application to display trades that have been removed from a trades cache which had a value over $1000
- Where do custom Filter classes need to be deployed?

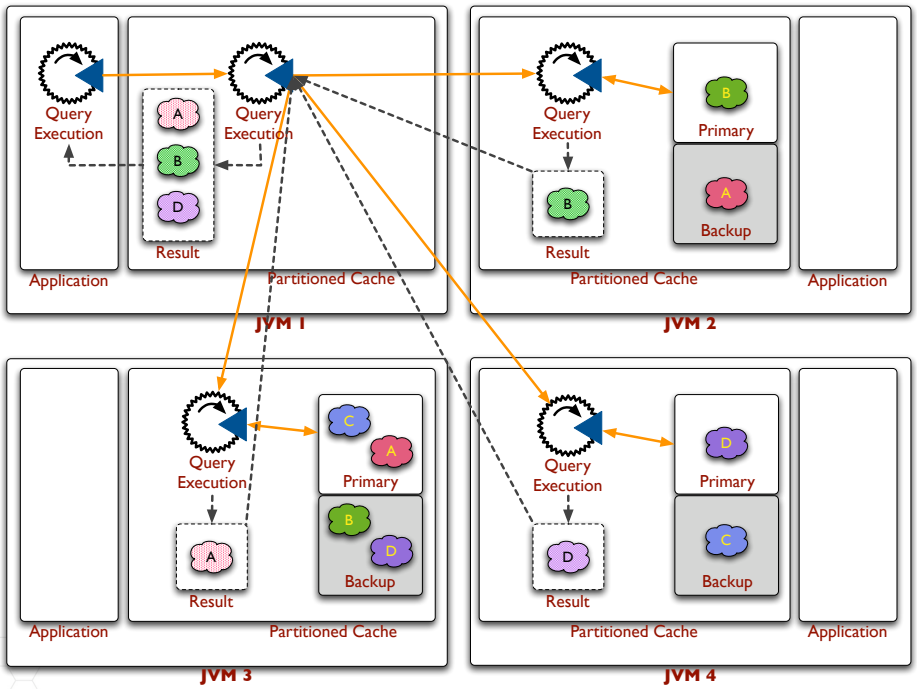
QueryMap Interface

QueryMap Interface

- Use `com.tangosol.util.QueryMap` interface to search for Values or Keys
- Use Filters to restrict searching and thus results
- Filtering occurs at Cache Entry Owner
  - ie: In Partitioned Topology, Primary Partitions do the filtering
- Use QueryMap interface to define Indexes to allow for search optimization

TANGOSOL

Parallel Query Execution



TANGOSOL

QueryMap Interface

```
Set entrySet(Filter filter)
```
Return a set view of the entries that satisfy the criteria expressed by the filter.

```
Set entrySet(Filter filter, Comparator comparator)
```
As above but iteration over the set will occur in ascending ordered according to the comparator.

```
Set keySet(Filter filter)
```
Return a set view of the keys contained in this map for entries that satisfy the criteria expressed by the filter.

```
void addIndex(ValueExtractor extractor, boolean fOrdered,
              Comparator comparator)
```
Add an index to a QueryMap.

```
void removeIndex(ValueExtractor extractor)
```
Remove an index from this QueryMap.

QueryMap Examples

A set containing all of the open trades
```
Set openTrades = trades.entrySet(
                  new EqualsFilter("isOpen", BOOLEAN.TRUE));
```

A set containing people with a last name beginning with “Mac”
```
Set macPeople = people.entrySet(
                 new LikeFilter("getLastName", “Mac%”));
```

A set of keys of people with a last name beginning with “Mac” or “Mc”
```
Set macPeopleKeys = people.keySet(
                     new OrFilter(
                      new LikeFilter("getLastName", “Mac%”),
                      new LikeFilter("getLastName", “Mc%”)));
```

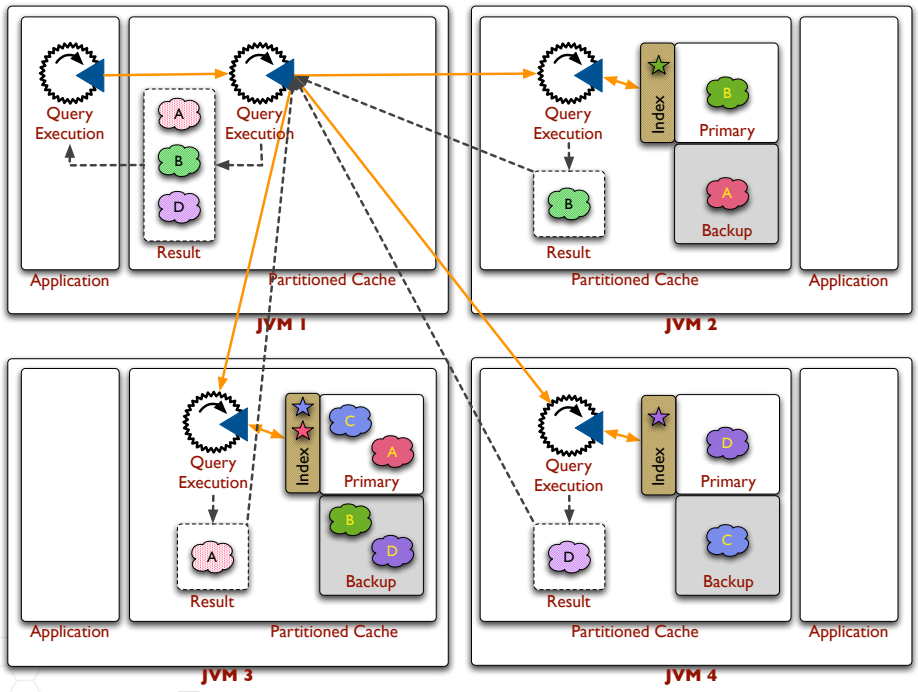
Indexes

- Coherence allows the definition of Indexes against Caches to optimize searches (and other processing).
- Indexes act as a hint to cache implementations for search optimization.
- Indexes may be ignored by a cache if indexes are not supported or if the desired index already exists.
- Application can suggest indexes to Coherence even if an index already exist (previously suggested)

Indexes

- Each application using Coherence may suggest the same set of indexes when it starts
- There is no downside to an application blindly suggesting indexes regardless of whether another application has already suggested the same indexes
- Indexes are maintained by Cache Entry Owners
  - ie: For Partitioned Topology, the Primary Partitions maintain their own indexes

Application of Indexes



TANGOSOL

ValueExtractors

- Coherence uses `com.tangosol.util.ValueExtractors` to determine the specific values from Entries to index
- Coherence doesn't index all of your object attributes
  - eg: You may be caching blobs or chunks of XML!
  - You have to tell Coherence which parts of an Object to index.
  - Do this using ValueExtractors

TANGOSOL

ValueExtractor Interface

```
Object extract(Object oTarget)
Extract a value from the passed object.
```


ValueExtractors

- Coherence ships with a set of standard ValueExtractor implementations
  - `ReflectionExtractor` - uses reflection to determine a value from the Entry
  - `ChainedExtractor` - chain a series of extractors together (evaluated left-to-right)
  - `IdentityExtractor` - returns the object itself
  - `KeyExtractor` - like the ReflectionExtractor but forces values to be extracted from the Entry Key instead of the Entry Value
- You may construct your own!

Index Examples

```
Suggest an index for trades based on their portfolio.  Ensure
the index is ordered, but use natural ordering (hence the
null).
trades.addIndex(
         new ReflectionExtractor("getPortfolio"),
         true,
         null);


Suggest an index for trades based on their market.  Don't use
ordering.
trades.addIndex(
         new ReflectionExtractor("getMarket"),
         false,
         null);
```


Filtering with Custom Extractors

- If you implement your own ValueExtractor class...
- And want to use it for indexing...
- You must...
  - Ensure that you correctly implement hashcode and equals
  - Use it within your Filters
- Coherence will only use an index for filtering when...
  - It can locate an appropriate identical ValueExtractor!
- Otherwise your index may not be used

Questions

- If an Entry Key has two methods, getFirstName and getLastName, how would you write a query to determine only those Entry Values where the lastname starts with "Mc"?
- Assuming Portfolios have a "getManager" method, how would you create an index over the trades cache for portfolio managers?
- What does the following index?
  cache.addIndex(IdentityExtractor.INSTANCE,true,null);

Questions

- If you create your own Filter and/or ValueExtractor implementations, where should they be deployed?

Continuous Query Cache

Continuous Query Cache

- Provides the ability to construct and have automatically maintained a locally managed, continuously up-to-date view of a Cache
  - `com.tangosol.net.cache.ContinuousQueryCache`
  - Implements NamedCache
  - Implements ObservableMap, QueryMap & InvocableMap
  - "Local View" may contain Keys or entire Entries
  - Ideal for "thick clients"
- Internal implementation combines the concepts of Queries, Filters and MapListeners

ContinuousQueryCache Class

```
ContinuousQueryCache(NamedCache cache, Filter filter)
Create a locally materialized view of a NamedCache using a
Filter

ContinuousQueryCache(NamedCache cache,
                     Filter filter,
                     boolean fCacheValues)
Create a materialized view of a NamedCache using a Filter,
specifying whether entire values should be locally cached (or
just keys)

ContinuousQueryCache(NamedCache cache,
                     Filter filter,
                     MapListener listener)
Create a materialized view of a NamedCache using a Filter,
specifying a default listener to receive underlying cache
events
```


Continuous Query Examples

```
Construct a continuous view of my trades
NamedCache myTrades =
    new ContinuousQueryCache(
            trades,
            new EqualsFilter("getOwner", myTraderId));


Construct a continuous view of pending orders over $1000
NamedCache pendingValuableOrders =
    new ContinuousQueryCache(
            orders,
            new AndFilter(
                new EqualsFilter("isPending", Boolean.True),
                new GreaterThanFilter("getAmount", 1000));
```

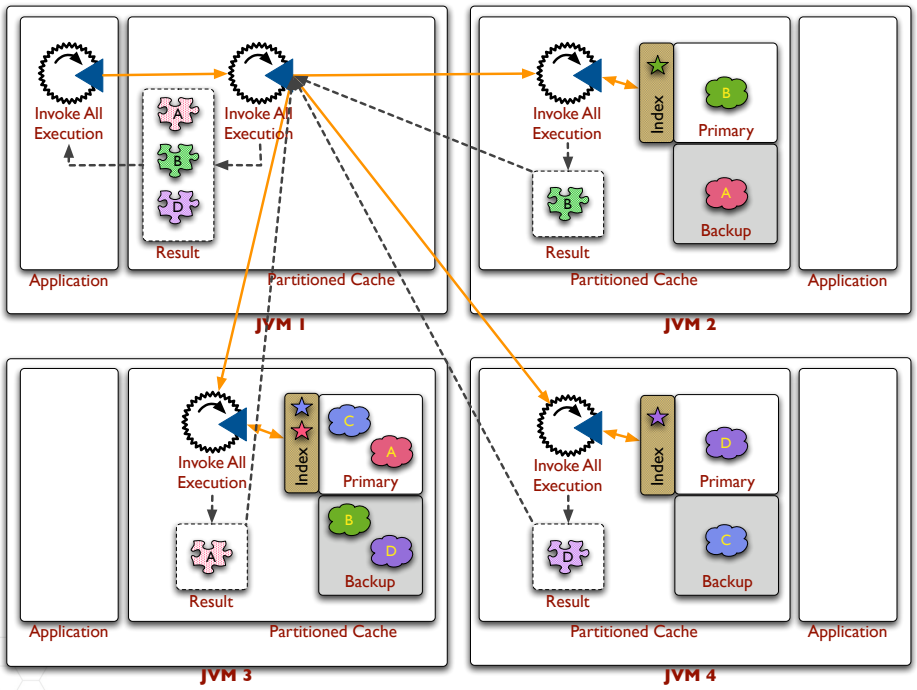
InvocableMap Interface

InvocableMap Interface

- Use `com.tangosol.util.InvocableMap` interface to Process or Aggregate Entries
  - Combine with Filters to restrict processing or aggregation!
- Filtering, Processing and Aggregation occurs at Cache Entry Owner
  - In Partitioned Topology, Primary Partitions do the work
  - Indexing will automatically be used to optimize processing

```
positions.invokeAll(
              new EqualsFilter("getTicker", "ORCL"),
              new StockSplitProcessor(2.0));
```

InvocableMap Execution



TANGOSOL

Entry Processors

- `com.tangosol.util.InvocableMap.EntryProcessor`s are agents that perform processing against Entries directly where they are being managed
  - Requests are sent directly to owners to do work
- Equivalent to "agents" executing services in parallel on the data in the cluster
- Processing...
  - may mutate cache entries, including creating, updating or removing, or
  - just perform calculations, or anything else!

TANGOSOL

InvocableMap Interface

```
Object invoke(Object oKey,
              InvocableMap.EntryProcessor processor)
```
Invoke the passed EntryProcessor against the Entry specified by the passed key, returning the result of the invocation

```
Map invokeAll(Collection keys,
              InvocableMap.EntryProcessor processor)
```
Invoke the passed EntryProcessor against the entries specified by the passed keys, returning the result of the invocation for each Entry

```
Map invokeAll(Filter filter,
              InvocableMap.EntryProcessor processor)
```
Invoke the passed EntryProcessor against the set of entries that are selected by the given Filter, returning the result of the invocation for each Entry

InvocableMap.EntryProcessor...

```
Object process(InvocableMap.Entry entry)
```
Process a Map.Entry object (yours to implement!)

```
Map processAll(Set setEntries)
```
Process a Set of InvocableMap.Entry objects (implementation typically provided by a super-class)

InvocableMap.Entry Interface

```
Object getKey()
Return the key corresponding to this entry

Object getValue()
Return the value corresponding to this entry

boolean isPresent()
Determine if this Entry exists in the Map

void remove(boolean isSynthetic)
Remove this Entry from the Map if it is present in the Map

Object setValue(Object value)
Store the value corresponding to this entry

void setValue(Object value, boolean isSynthetic)
Store the value corresponding to this entry
```


Entry Processors

- There are a number of provided EntryProcessors
  - AbstractProcessor, CompositeProcessor, ConditionalProcessor, ConditionalPut, ConditionalPutAll, ConditionalRemove, ExtractorProcessor, NumberIncrementor, NumberMultiplier, PreloadRequest, PropertyProcessor, UpdaterProcessor, VersionedPut, VersionedPutAll

Entry Processors

- Usually you create your own custom implementations
  - Simply sub-class
    `com.tangosol.util.processors.AbstractProcessor`

```
class StockSplitProcessor extends AbstractProcessor {

  ...

  Object process(Entry entry) {
    Position position = (Position)entry.getValue();
    position.setAmount(position.getAmount() * factor));
    entry.setValue(position);
    return null;
  }
}
```


Entry Processor Semantics

- EntryProcessors against the same key will be logically queued
  - This means lock-free (high through-put) processing!
- EntryProcessors can return any "serializable" value
  - Including `null` if a result is not required
- You can invoke EntryProcessors against entries that don't yet exist
  - Use `Entry.isPresent()` to determine if an entry exists

Entry Processor Semantics

- Exceptions thrown within EntryProcessors will be wrapped and re-thrown to application calling thread
- Failure to "set" or "remove" a value will mean no Cache Entry mutation will occur!
- Cache Entries ONLY updated AFTER successful execution (no Exceptions thrown) of the processors

Entry Processor Semantics

- If fatal failure occurs during execution (eg: JVM death)...
  - EntryProcessor execution will be rescheduled & executed again (guaranteed to execute) by clients
- You MUST ensure EntryProcessors are IDEMPOTENT
  - ie: If executed again, the EntryProcessor must produce the same value (and external side-effects)

Entry Processor Semantics

- EntryProcessors may invoke other EntryProcessors
  - BUT: They must be against different cache services
  - This is to avoid re-entrancy and deadlock
  - This is very advanced and requires configuration*

Questions

- Where do custom EntryProcessors need to be deployed?
- How could you use an EntryProcessor to insert a value into a Cache and set the time of insertion on the value as it is placed in the Cache?
- How would you develop a solution to synthetically remove entries from a trades cache that are older than 25 days?

Entry Aggregators

- `com.tangosol.util.InvocableMap.EntryAggregator` are agents that aggregate values from Entries
  - Sum, Average, Count, Max, Min, Distinct, GroupBy, Having...
- Equivalent to "agents" executing services in parallel on the data in the cluster
- Aggregation...
  - must not mutate Entries
  - is for data extraction and aggregation only!

InvocableMap Interface

```
Object aggregate(Collection keys,
                 InvocableMap.EntryAggregator aggregator)
```
Perform an aggregating operation against the entries specified by the passed keys

```
Object aggregate(Filter filter,
                 InvocableMap.EntryAggregator aggregator)
```
Perform an aggregating operation against the set of entries that are selected by the given Filter

Aggregation Examples

```
The total value of the open orders
BigDecimal result =
    orders.aggregate(
        new EqualsFilter("isOpen", Boolean.True),
        new BigDecimalSum("getValue"));


The categories of books on sale over $25
Set categories =
    stock.aggregate(
        new AndFilter(
            new EqualsFilter("isOnSale", Boolean.True),
            new GreaterThenFilter("getPrice", 25)),
        new DistinctValue("getCategory"));
```


Questions

- Where do custom EntryAggregators need to be deployed?
- How could you use an EntryAggregator to return the distinct first and last name of people in a cache?
- How could develop a solution to return all of the first and last names of people in a cache?