
ORACLE®

ORACLE®

Working with Data – In-place processing & Persisting Data

Lesson 5

Objectives

After completing this lesson, you should be able to:

- Understand concurrency challenges
- Describe the EntryProcessor interface
- Understand how to send the processing to the data
- Describe the methods of persisting data in Coherence
- Discuss Coherence configuration files
- Understand next steps

Concurrency

- In any environment, concurrent access to data must be considered
- Traditional processing uses locking mechanisms to control data concurrency
- Coherence provides the ConcurrentMap interface (not covered here) which allows locking/unlocking of objects.
  - Use lock, unlock methods and then get and put methods.
  - In a world where you want to minimise the number of network trips – this could take approx 14
- There is a way to do this in 4 network hops – read on…

Entry Processors

- com.tangosol.util.InvocableMap.EntryProcessors 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!

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

- Next we need to define what Class we want to invoke…

InvocableMap.EntryProcessor …

- **An entry process implements the following (and must be serializable)**
- **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)
- Now we need to see what actions we can perform with an Entry

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

Out of the Box EntryProcessors

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

- You will mostly write your own…

Example

- 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;
   }
}
```

- Now to run this on all entries

```
// now run the entry processor
stocks.invokeAll(AlwaysFilter.INSTANCE,
                 new StockSplitProcessor ());
```

Gotchas

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

Data Source Integration

- Coherence supports transparent read-write caching of any datasource, including databases, web services, packaged applications and filesystems, databases are the most common use case
- Effective caches must support both intensive read-only and read-write operations, and in the case of read-write operations, the cache and database must be kept fully synchronized.
- To accomplish this, Coherence supports Read-Through, Write-Through, Refresh-Ahead and Write-Behind caching.

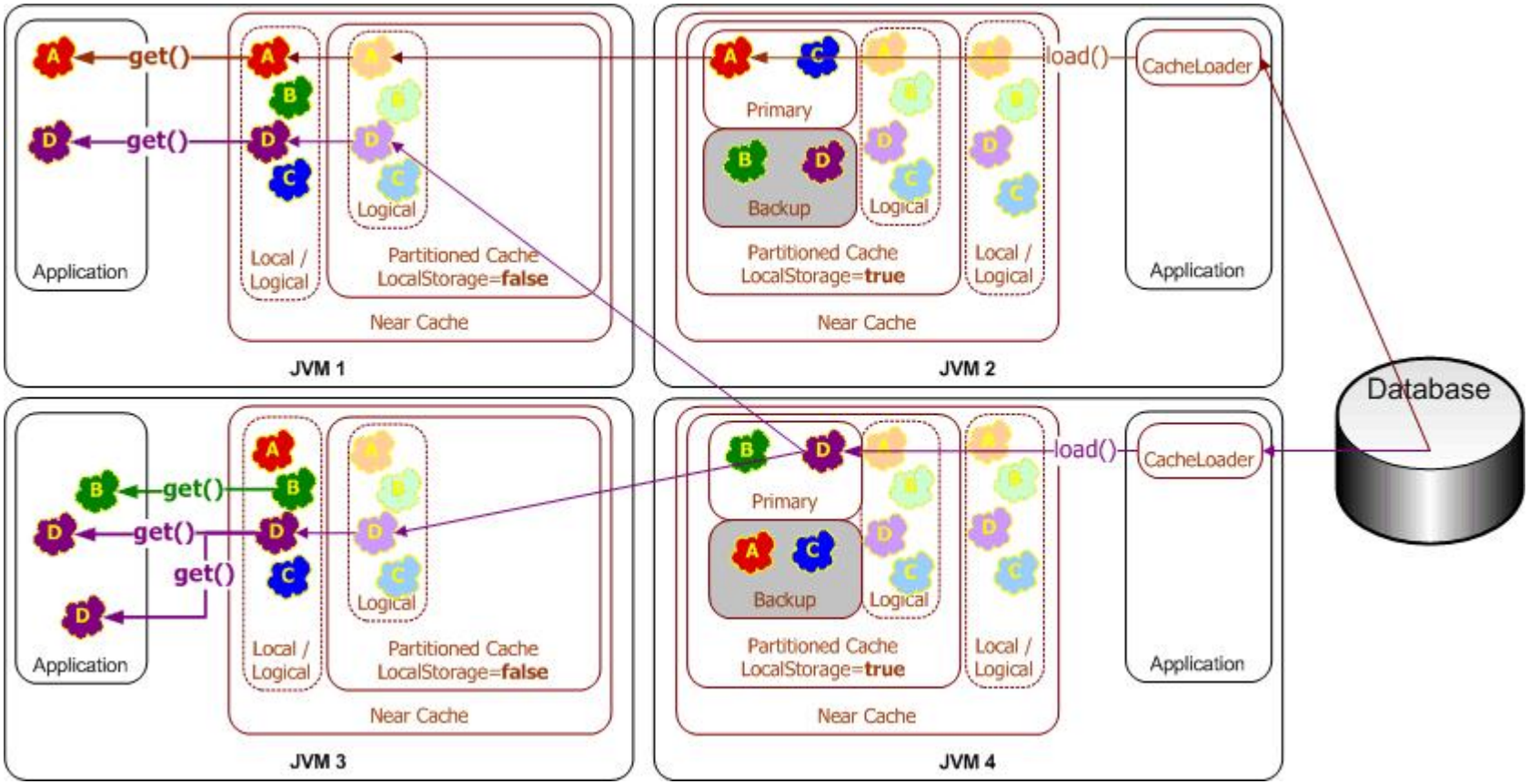
Persisting Data – The mechanics

- Backing Maps are the method by which a NamedCache persists data
- Memory is the default implementation that we have been using
- This is achieved by using a different Backing Map to persist to databases, files ,etc

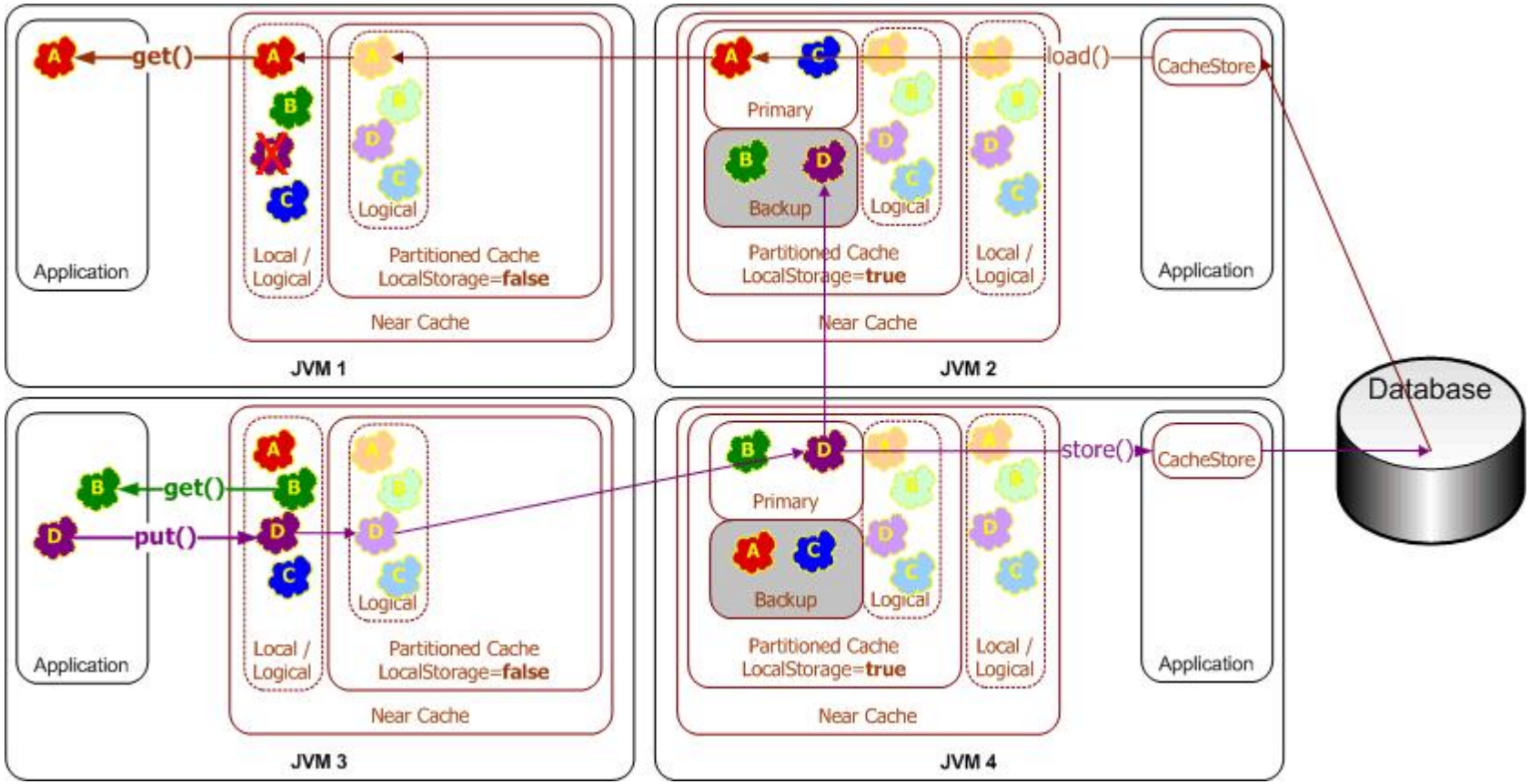
Implementations

- Read Through
  - If data is not present in the cache, then the back end data source implementation is used to read the data and place it in the cache
- Write Through
  - When writing data, the “put” method will not return until the data is written the the back end data source. E.g. syncrhonous
- Refresh Ahead
  - Data that is about to expire will be refreshed before its expiry time, so as to not delay any reads
- Write Behind
  - Data is written asynchronously to the back end data source with a configurable delay. E.g. ensure that the data is written by a max of **n** seconds

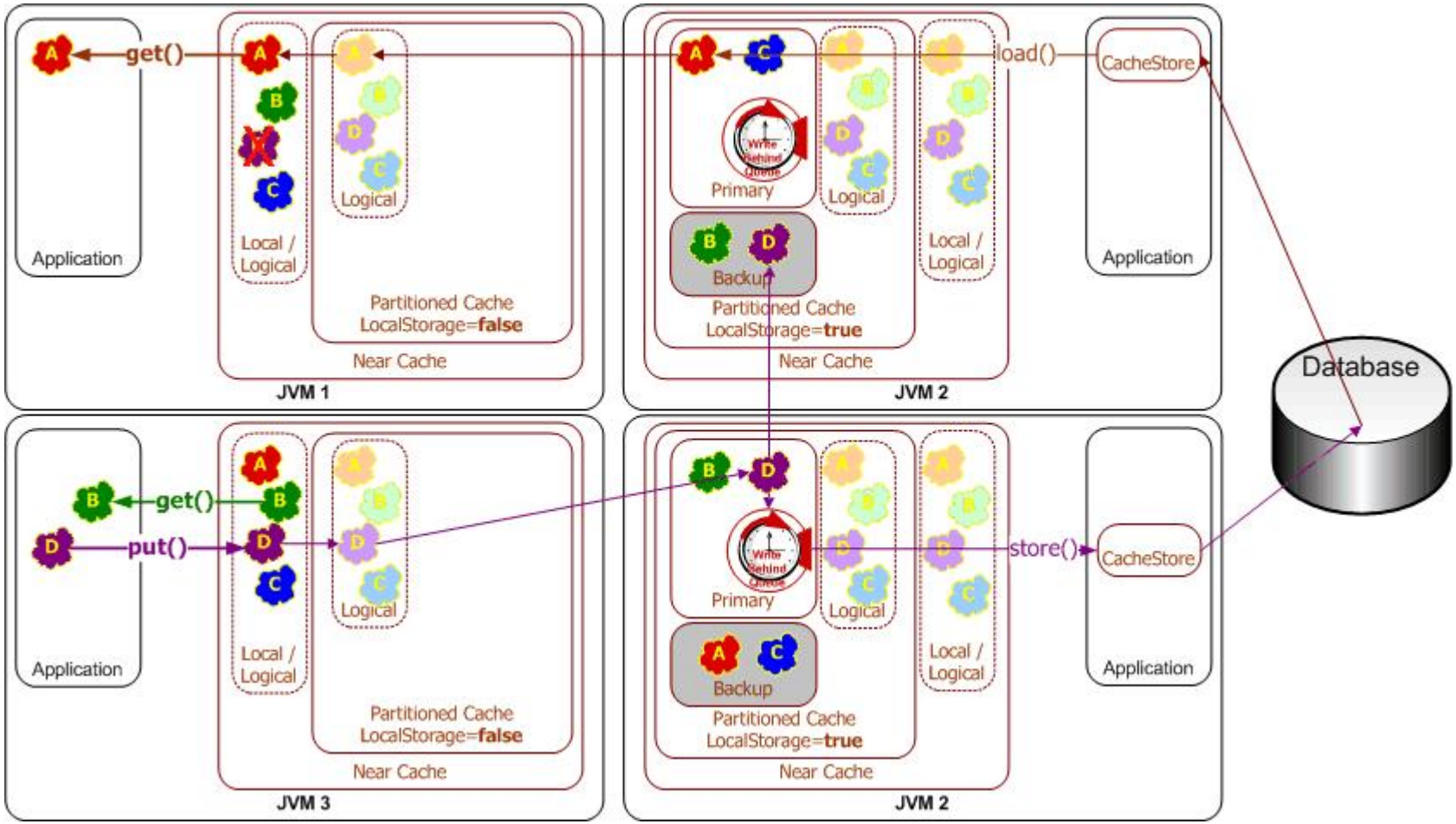
Read Through



Write Through



Write Behind



Data Source Integration

There are a number of out of the box integrations:

- Hibernate
- Toplink Essentials
- Java Persistance Architecture (JPA)
- Simple JDBC
- Filesystem

- You can create your own implementations…

Configuration Files

- Up to now we have not specified any configuration to use our caches. How does Coherence know what sort of cache toplogy to use?
- Default configuration has been loaded from coherence.jar!/tangosol-coherence.xml
  - You should be able to see this from the messages on the startup of your cache servers
- You can specify the config file using the following Java parameter
  - -Dtangosol.coherence.cacheconfig=file.xml
  - You can also point to a http location – very useful with large number of members in a cluster
- Ships with out-of-the-box wildcard-based Cache Names
- Wildcard Cache Names map to out-of-the-box Topologies!

Configuration Files (cont…)

- There are two main sections:
  - Cache scheme mapping
  - Definition of Cache scheme
- Cache Scheme Matching
  - Matches names of caching with schemes

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>dist-*</cache-name>
      <scheme-name>example-distributed</scheme-name>
      <init-params>
        <init-param>
          <param-name>back-size-limit</param-name>
          <param-value>10000</param-value>
        </init-param>
      </init-params>
    </cache-mapping>
```

Configuration Files (cont…)

- Scheme Definition
  - Defines the actual scheme

```
<caching-schemes>
    <!-- Distributed caching scheme.  -->
    <distributed-scheme>
      <scheme-name>example-distributed</scheme-name>
      <service-name>DistributedCache</service-name>

      <backing-map-scheme>
        <local-scheme>
          <scheme-ref>example-backing-map</scheme-ref>
        </local-scheme>
      </backing-map-scheme>

      <autostart>true</autostart>
    </distributed-scheme>
```

Next Steps

- Hopefully this has given you a taste of what is possible within Oracle Coherence
- Potential Next steps:
  - Visit wiki.tangosol.com for more in-depth technical information on Coherence including many examples
  - Investigate .NET integration
  - HTTP session state management
  - JPA Lab in your own time
  - Continuous Queries
  - Much more…

Summary

In this lesson, you should have learned how to:

- Understand concurrency challenges
- Describe the EntryProcessor interface
- Understand how to send the processing to the data
- Describe the methods of persisting data in Coherence
- Discuss Coherence configuration files
- Understand next steps


ORACLE®