

ORACLE®

12.1.3 Hidden Gems

Harvey Raja
Consulting Member Technical Staff
Fusion Middleware, Coherence
Month 07, 2014



Safe Harbor Statement

The following is intended to outline our general product 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.

Note: The speaker notes for this slide include instructions for when to use Safe Harbor Statement slides.

Tip! Remember to remove this text box.

Program Agenda

- 1 ➤ Deserialization Accelerator
- 2 ➤ Asynchronous Index Build
- 3 ➤ Guardian stats
- 4 ➤ Project Shrapnel
- 5 ➤ Asynchronous EntryProcessors
- 6 ➤ Pop the hood...

Deserialization Accelerator

- Anybody want ObjectLocalBackingMap?
- Hold deserialized value in MapIndex
- ReflectionExtractor queries (BinaryEntry.getValue) will use deserialized value

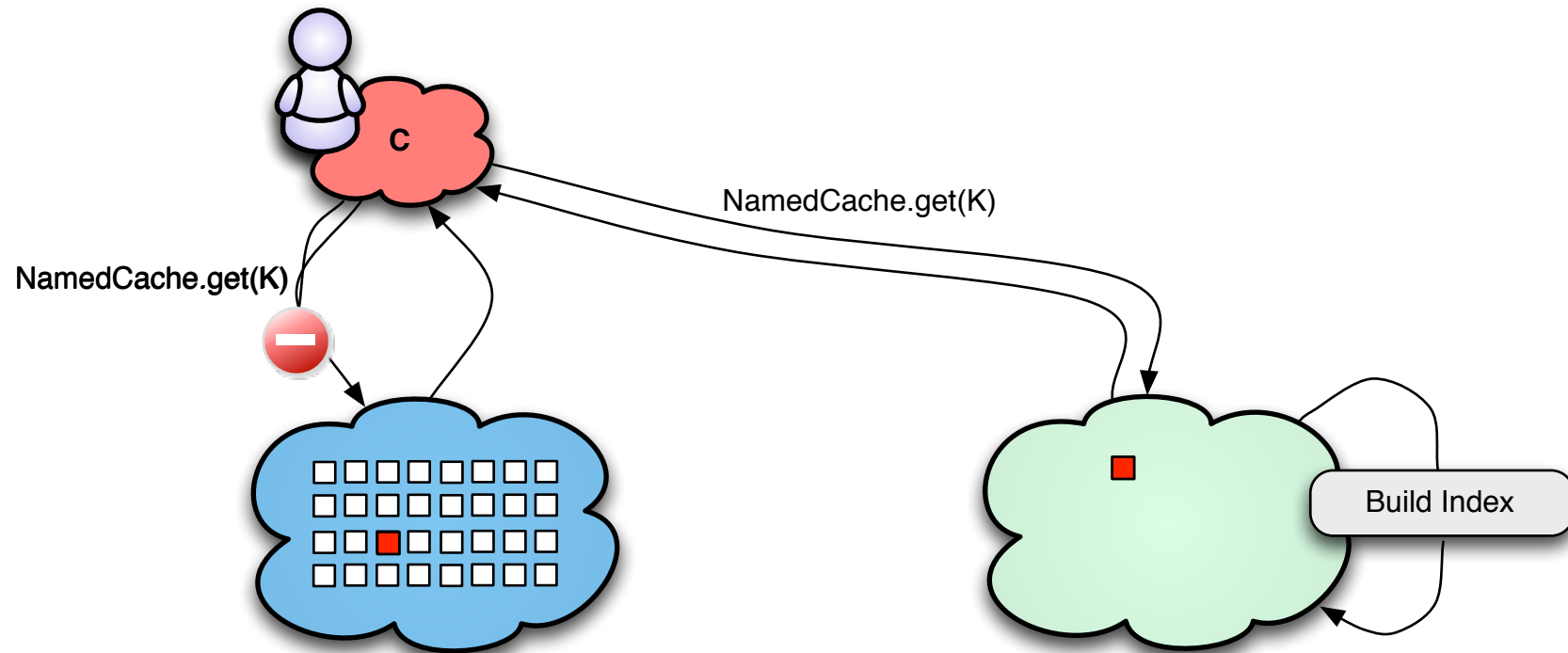
```
cache.addIndex(new DeserializationAccelerator(), false, null);  
cache.addIndex(IdentityExtractor.INSTANCE, false, null);
```

- Optimized MapIndex implementation

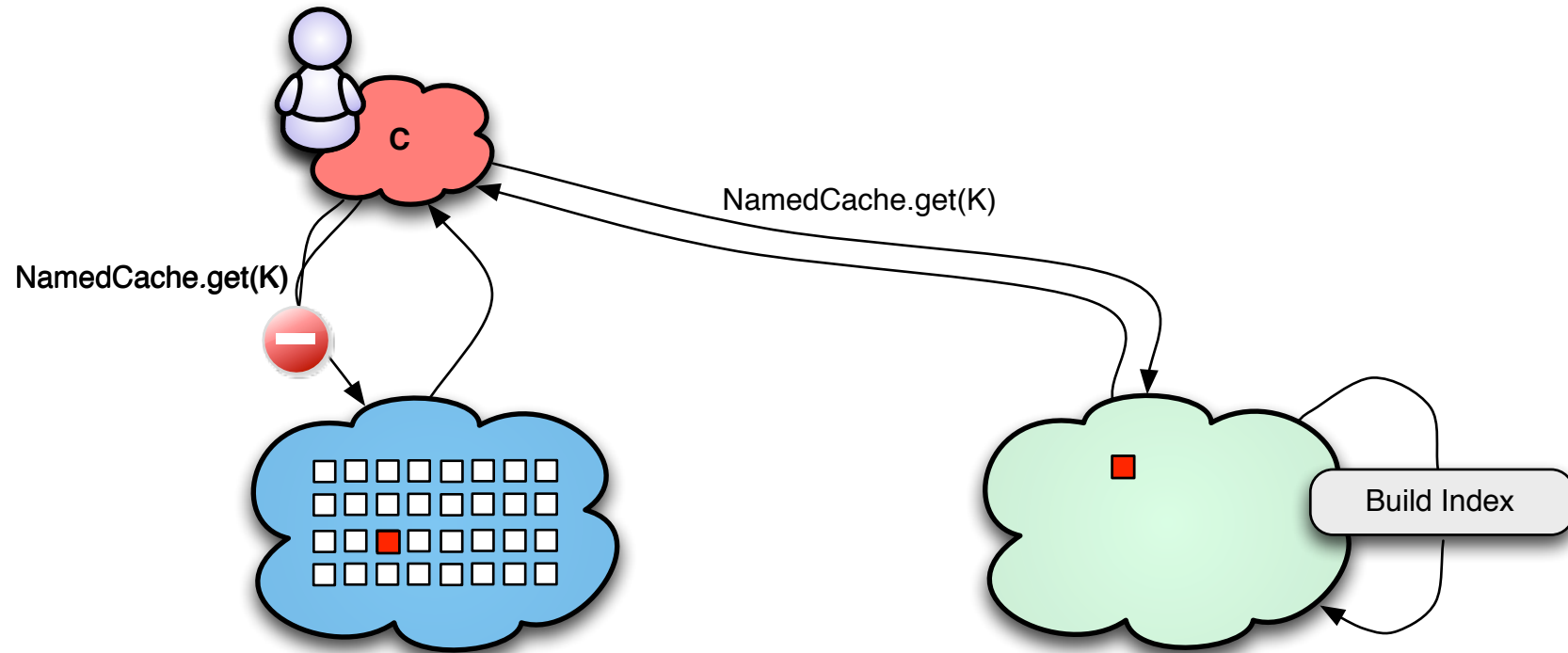
Asynchronous Index Maintenance

- Partition Transfer blocks access to the partition for the window of transfer
- Pre 12.1.3 this included updated MapIndex with the newly owned entries
- 12.1.3 MapIndex is updated asynchronously by a thread in the pool
- Query request waits until index update job completes

Synchronous Index Update



Asynchronous Index Update



Guardian Improvements

- Public Guardian

```
Guardian guardian = CacheFactory.getCluster().getResourceRegistry().getResource(Guardian.class);  
guardian.guard(me);
```

- JMX Statistics:

The screenshot shows the JMX console for the process `com.tangosol.net.CacheFactory (pid 24820)`. The left pane displays the MBeans tree, and the right pane shows the attributes of the selected MBean. The `GuardRecoverCount` and `GuardTerminateCount` attributes are highlighted with a red box.

Attribute	Value
Name	
BufferPublishSize	32
BufferReceiveSize	513
CpuCount	8
FlowControlEnabled	true
GuardRecoverCount	12
GuardTerminateCount	2
LoggingDestination	stdout
LoggingFormat	{date}/{uptime} {product} {version} <[level]> {thread}={thr...}
LoggingLevel	8
LoggingLimit	2147483647
MachineId	60314
MachineName	localhost
MemberName	n/a
MemoryAvailableMB	492
MemoryMaxMB	505
MulticastAddress	/224.255.255.255
MulticastEnabled	true
MulticastPort	7777
MulticastTTL	0
MulticastThreshold	25
NackEnabled	true
NackSent	0
PacketDeliveryEfficiency	1.0
PacketsBundled	2
PacketsReceived	297
PacketsRepeated	0
PacketsResent	3
PacketsResentEarly	0
PacketsResentExcess	0
PacketsSent	270
Priority	0
ProcessName	24820
ProductEdition	Grid Edition

Shrapnel

- PartialResultAggregator
 - Single invocation of aggregatePartialResults per member
- NamedCache.putAll/invokeAll -> CacheStore.storeAll
- Deadlock avoidance
 - Always were able to detect
 - Tried to prevent
 - Now avoid
- Service Quiesce

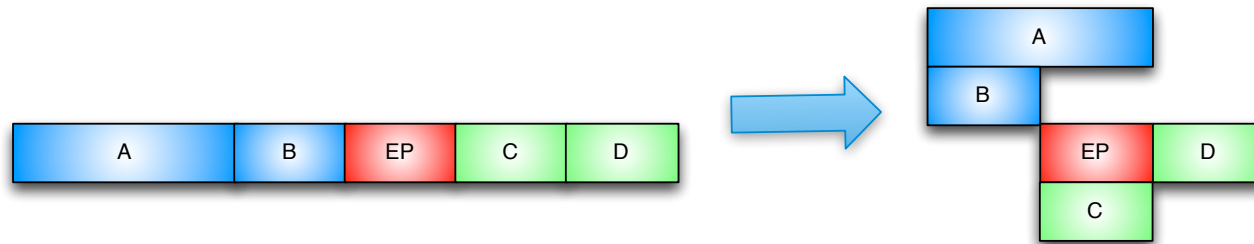
- 1 Deserialization Accelerator
- 2 Asynchronous Index Build
- 3 Guardian stats
- 4 Project Shrapnel
- 5 Asynchronous EntryProcessors
- 6 Pop the hood...

Entry Processors

- Avoids lock(K), get(K), mutate(V), put(K, V), unlock(K)
- Executed on the member that owns K
- Possible to be invoked with a Collection<K> or a Filter (Predicate)
- Allow features such as Partition Lite Transactions

Entry Processors

- Are the steps in a processing pipeline parallelizable?



- If so the answer is simple:

```
final NamedCache cache = CacheFactory.getCache("foo");  
Thread thread = new Thread(new Runnable()  
{  
    public void run()  
    {  
        cache.invoke(1, new ConditionalPut(AlwaysFilter.INSTANCE, 1));  
    }  
});  
thread.start();  
  
// do some other work  
  
// no that all parallelizable work is complete, make sure the EP thread finished  
thread.join();
```

Entry Processors

- As of 12.1.3 OOTB we provide an `AsynchronousProcessor`
- Wrapper processor
- `AsynchronousProcessor` extends `AsyncAgent` implements **Future**
- Can specify
 - Unit of Order {default: `Thread.getId()`}
 - Automatic Flow Control {default: `true`}

Hardware and Software Engineered to Work Together

ORACLE®