How to speed up your application using JCache

GREG LUCK, CO-SPEC LEAD JSR107
@GREGRLUCK CEO | HAZELCAST
10 FEBRUARY 2016

Agenda

- Theory of Caching
- Java Caching (JCache), JSR-107
- Code Demo

Introduction to Caching

Benefits of Caching

- Performance
- Offload expensive or non-scalable parts of your architecture
- Scale up – get the most out of one machine
- Scale out – add more capacity with more machines
- Excellent Buffer against load variability

And…

- **Usually very fast and easy to apply**

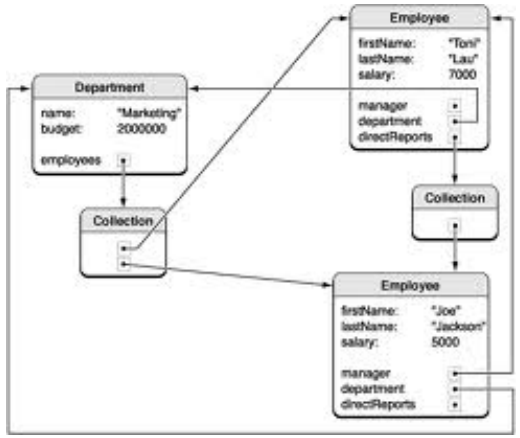
When to Use Caching

- When applications use the same data **more than once**
- When cost (time / resources) of **making an initial copy is less** than fetching or producing the data again or when faster to request from a Cache

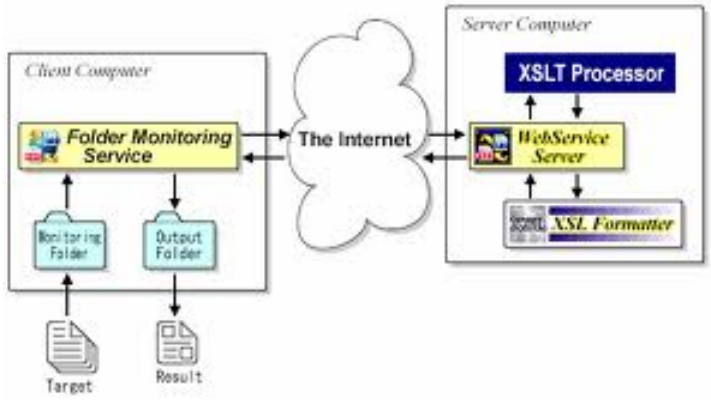
Common Problem Areas that Benefit

1 Anything Web Scale

2 Compound Data Objects

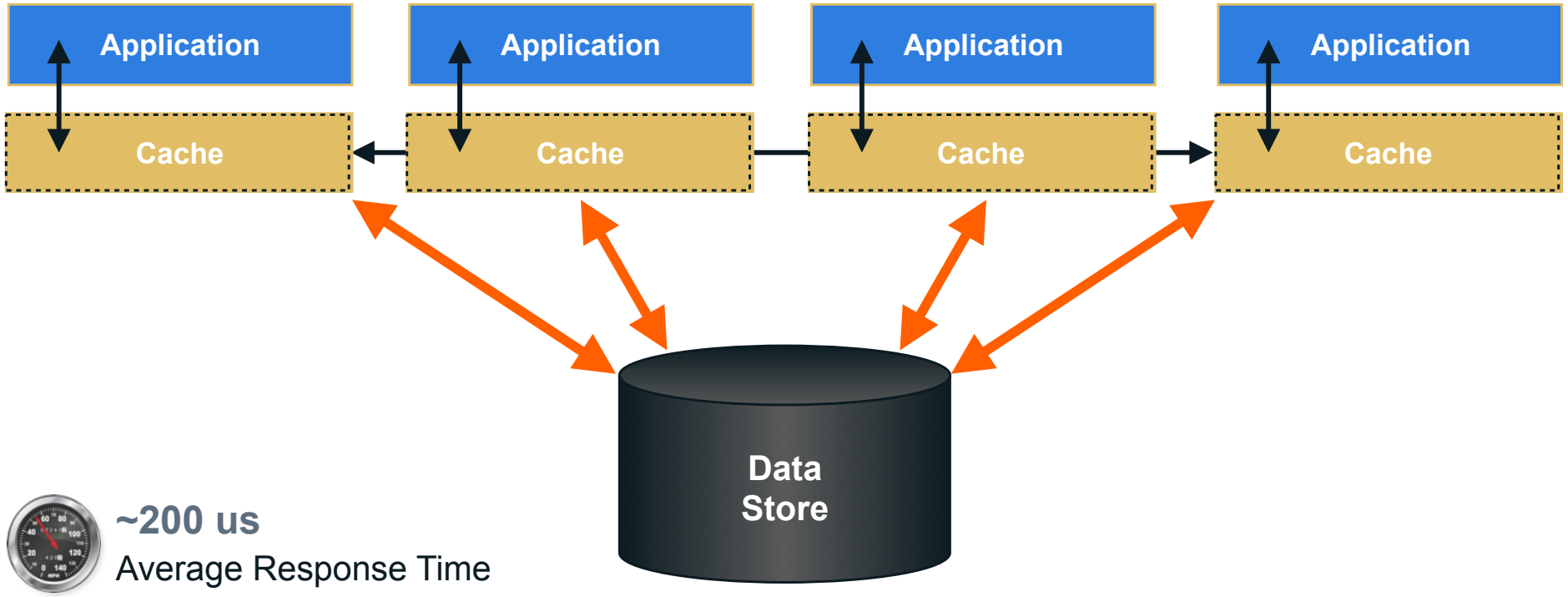


3 Anything where the data is across the network



4 Data Persistence

Database Caching

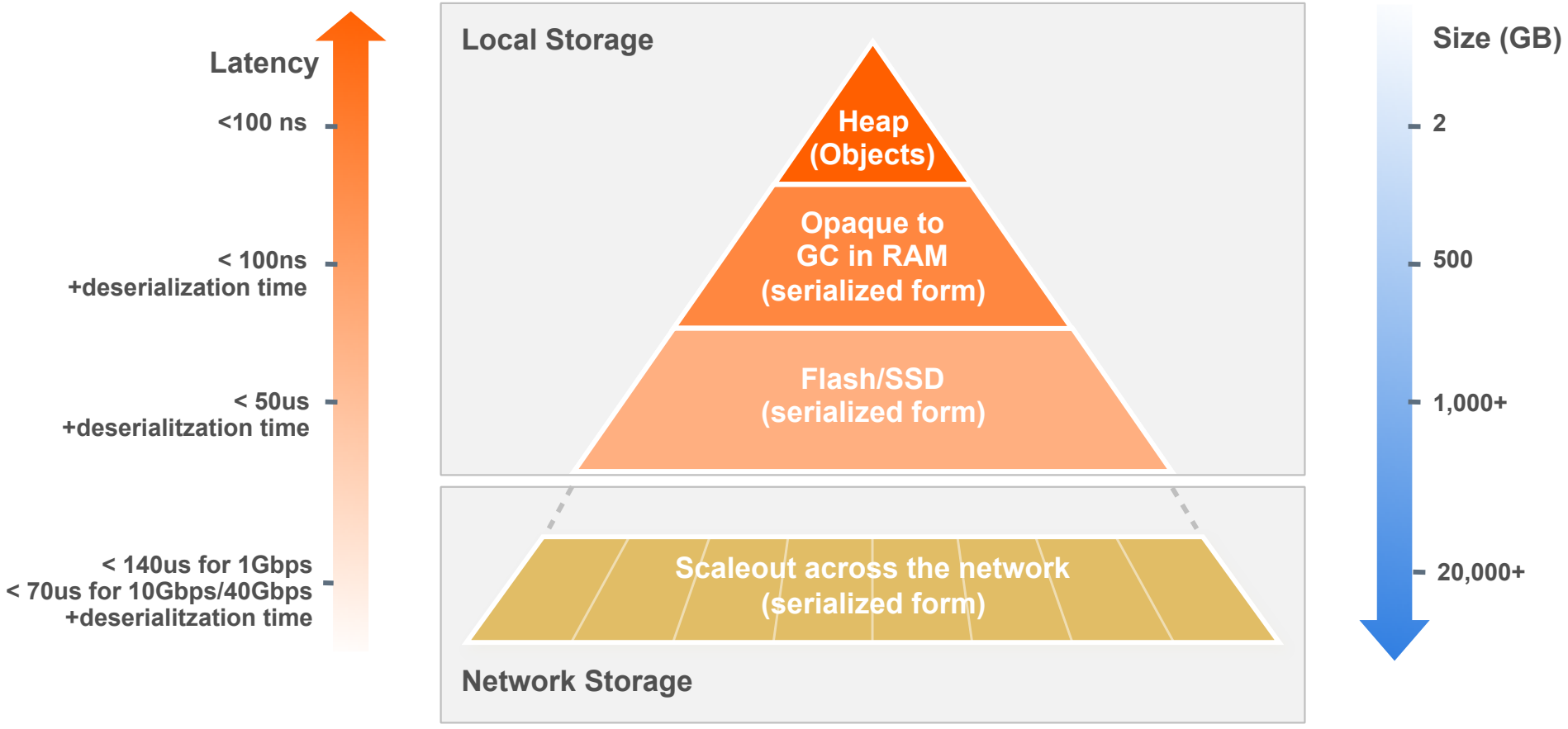


~200 us
Average Response Time

Speed	Costs	Scalability
■ ■ ✓	■ ■ ✓	■ ■ ✓

Moving data from the database into the cache increases processing speed and can reduce database licensing and maintenance costs.

Caches are built primarily in RAM in-process or distributed



Estimated Performance Improvements

amdahl's law

Predicted System Speedup

=

1 / ((1 – Proportion Sped Up) + Proportion Sped Up / Speed up))

Cache Efficiency

Cache Efficiency = cache hits / total hits

- High efficiency = high offload
- High efficiency = high performance

- How to increase:
  - Put reference data in the cache
  - Put long lived in the cache.
  - Consider frequency of mutability of data
  - Put highly used data in cache
  - Increase the size of the cache. Today you can create TB sized caches

Problems to Consider

- Standalone Caches and the N * problem
  - As each entry expires, the backing system gets N requests for data where n is the number of standalone caches. Solution: Use a distributed cache
- Consistency with the System of Record
  - How to keep the cache in sync with changes in a backing system. Solution: Match mutability of data with data safety configuration. Update the cache and backing store at the same time.
- Consistency with other cache nodes
  - How to keep all cache nodes in sync: Solution: Use a distributed cache and match consistency configuration with data mutability

New JCache Standard (JSR107)

Java Caching (JCache)

What?

- Java Caching (JCache) standardized Caching for the Java Platform*
- A common mechanism to create, access, update and remove information from Caches

How?

- JSR-107: Java Caching Specification (JCache)
- Java Community Process (JCP) 2.9

Java Caching (JCache)

Why?

- Standardize! Standardize! Standardize!
  - Core Caching Concepts
  - Core Caching API
- Provide application portability between Caching solutions
  - Big & Small, Open & Commercial
- Caching is ubiquitous!
- Allows frameworks to depend on JCache and stop creating specific integrations to each and every cache

Java Caching (Jcache)

Recent History

Item	*Date*
JCache Final Spec Released	*18 March 2014*
Spring 4.1	***September 2014***
Hazelcast 3.3.1 TCK Compliant	***September 2014***
Hazelcast 3.4 (with High-Density Memory Store)	***November 2014***
Hazelcast 3.5 - added HD Memory Store to near cache)	***June 2015***
Most project vendors create implementations	***June 2014 - June 2015***
Hazelcast 3.6 (split brain handler, quorums)	***January 2016***
JCache 1.1 Maintenance Release	***March 2016***

Java Caching (JCache)

Which Platform?

java.util.Map (Java 6/7)	*Target Platform*
Specification (SPEC)	Java 6+ (SE or EE)
Reference Implementation (RI)	***Java 7+ (SE or EE)***
Technology Compatibility Kit (TCK)	***Java 7+ (SE or EE)***
Demos and Samples	***Java 7+ (SE or EE)***

Implementations

Implementations

- JCache Reference Implementation
- Ehcache
- Hazelcast
- Oracle Coherence
- Infinispan
- GridGain/Apache Ignite
- TayzGrid*
- Caffeine* (Ben Manes)

Keep Track

- https://jcp.org/aboutJava/communityprocess/implementations/jsr107/index.html
- * Being verified by spec leads

Non-Implementation: Gemfire/Geode

Gemfire/Geode have no plans to implement PIVOTAL GemFire

Why?

- Gemfire and Geode are directly supported in Spring
- Because Spring supports JCache cache annotations you can use Gemfire/Geode from Spring but that is it.

See:

- http://apache-geode-incubating-developers-forum.70738.x6.nabble.com/JCache-JSR-107-support-td1255.html

Campaign to have Pivotal Support JCache:

#PivotalJCache campaign for @PivotalGemFire to support #JSR107 #JCache. Please retweet or mention #PivotalJCache

Relationship to NoSQL

Difficult/Impossible for NoSQL to fully implement the spec

- Server Side code execution including: Entry Processors, Listeners, Write-Through etc.
- Strong Consistency is the default consistency model and is not supported by most/all NoSQL.

Likely that Couchbase will release a partial implementation leaving out `EntryProcessor` and some other methods with an `UnsupportedOperationException` if these methods are called. They have Developer Preview 2 out.

Using With NoSQL

- Use NoSQL like a database and read-through/write-though to it using `CacheLoader/CacheWriter`.
- NoSQL gives you scale our persistence - cache gives you very low latencies

Java Caching (JCache)

Project Hosting

- JCP Project:
  - http://jcp.org/en/jsr/detail?id=107
- Source Code:
  - https://github.com/jsr107
- Forum:
  - https://groups.google.com/forum/?fromgroups#!forum/jsr107

Java Caching (JCache)

How to get it

- Apache Maven (via Maven Central Repository)

```
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
    <version>1.0</version>
</dependency>
```

Caches and Caching

Caches and Caching

JSR107 Cache Definition:

A high-performance, low-latency data-structure* in which an application places a **temporary copy** of information that is likely to be used **more than once**

Maps vs Cache APIs

java.util.Map (Java 6/7)
Key-Value Based API
Supports Atomic Updates
Entries Don’t Expire
Entries Aren’t Evicted
Entries Stored On-Heap
Store-By-Reference

javax.cache.Cache (Java 6)
Key-Value Based API
Supports Atomic Updates
Entries May Expire
Entries May Be Evicted
Entries Stored Anywhere (ie: topologies)
Store-By-Value and Store-By-Reference
Supports Integration (ie: Loaders / Writers)
Supports Observation (ie: Listeners)
Entry Processors
Statistics

JCache: Features

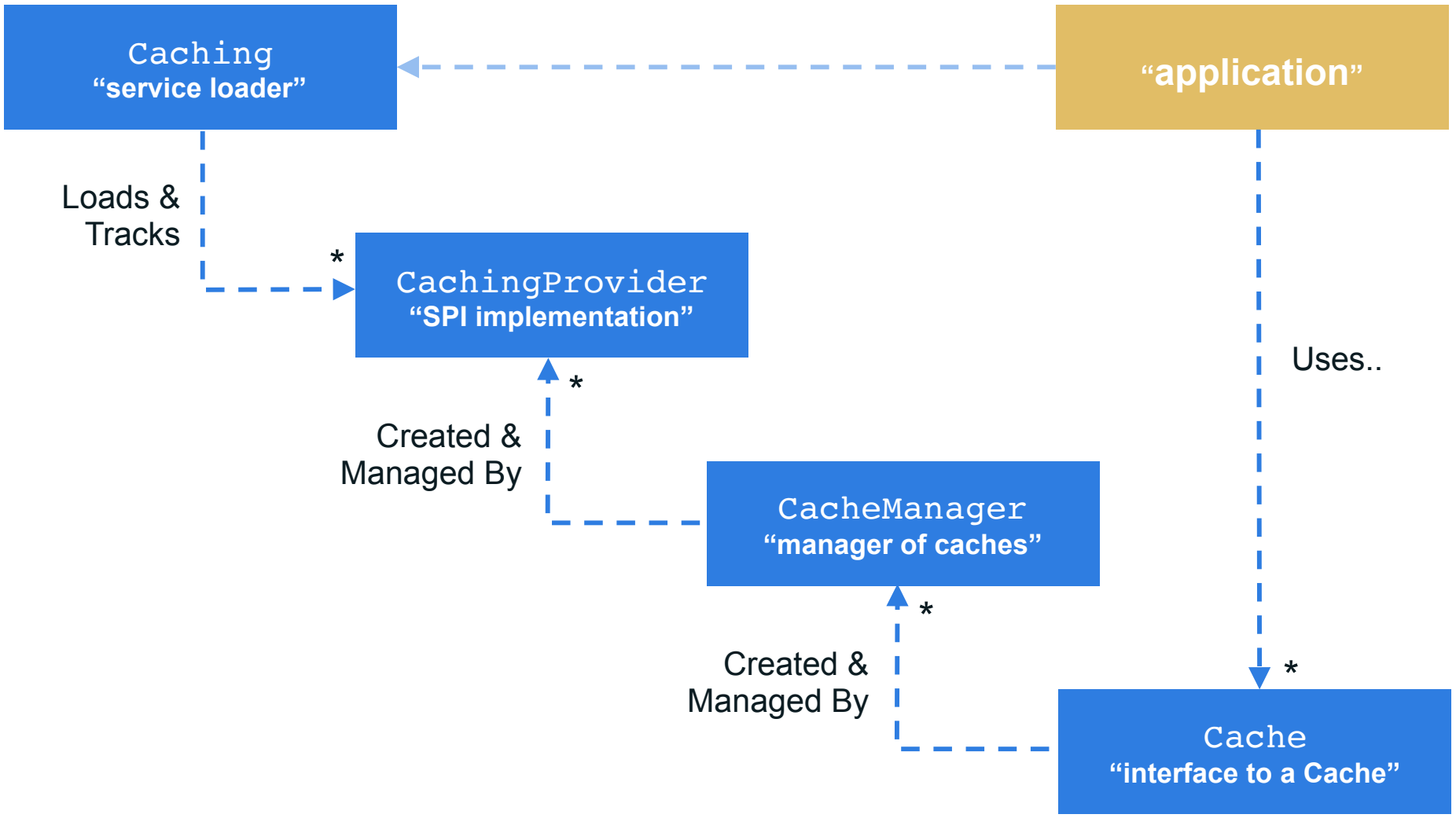
- `java.util.ConcurrentMap` like API
- Atomic Operations
- Lock-Free
- Read-Through / Write-Through Integration Support
- Cache Event Listeners
- Fully Generic API = type-safety
- Statistics
- Annotations (for frameworks and containers)
- Store-By-Value semantics (optional store-by-reference)

JCache: Features

- Topology Agnostic
  - Topologies not defined or restricted by the specification
- Efficiently supports:
  - “local” in-memory Caching and
  - “distributed” server-based Caching

JCache Key Classes/Interfaces

JCache: Runtime Structure



JCache: Cache Managers

`javax.cache.CacheManager`

- Establishes, configures, manages and owns named Caches
  - Caches may be pre-define or dynamically created at runtime
- Provides Cache infrastructure and resources
- Provides Cache “scoping” (say in a Cluster)
- Provides Cache ClassLoaders (important for store-by-value)
- Provides Cache lifecycle management

JCache: Hello World

(via a Cache Manager)

```
// acquire the default CacheManager
CacheManager manager = Caching.getCacheManager();

// acquire a previously configured cache (via CacheManager)
Cache<Integer, String> cache = manager.getCache("my-cache",
    Integer.class, String.class);

// put something in the cache
cache.put(123, "Hello World");

// get something from the cache
String message = cache.get(123);
```

Cache Interface & Methods (in IDE)

JCache: Entry Processors

Custom atomic operations for everyone!

```
// acquire a cache
Cache<String, Integer> cache = manager.getCache("my-cache",
    String.class, Integer.class);

// increment a cached value by 42, returning the old value
int value =
    cache.invoke("key", new IncrementProcessor<>(), 42);
```

JCache: Entry Processors

Custom atomic operations for everyone!

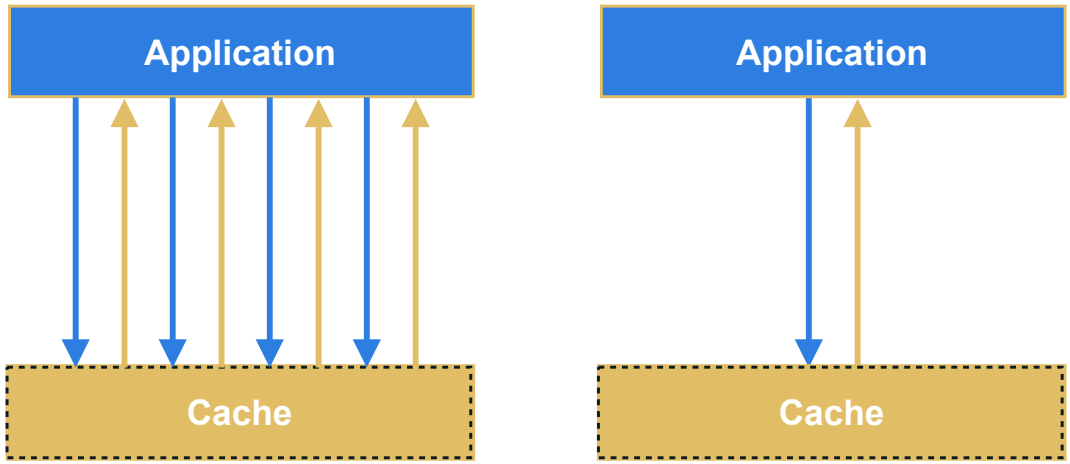
```
public class IncrementProcessor<K>
    implements EntryProcessor<K, Integer, Integer>, Serializable {

    @Override
    public Integer process(MutableEntry<K, Integer> entry,
    Object... arguments) {
        if (entry.exists()) {
            int amount =
                arguments.length == 0 ? 1 :(Integer)arguments[0];
            int current = entry.getValue();
            entry.setValue(count + amount);
            return current;
        } else {
            throw new IllegalStateException("no entry exists");
        }
    }
}
```

JCache: Entry Processors

Custom atomic operations for everyone!

- Eliminate Round-Trips! (in distributed systems)



- Enable development of a Lock-Free API! (simplifies applications)

*May need to be Serializable (in distributed systems)

JCache: Entry Processors

Which is better?

```
// using an entry processor?
int value = cache.invoke(
    “key”, new IncrementProcessor<>(), 42);

// using a lock based API?
cache.lock(“key”);
int current = cache.get(“key”);
cache.put(“key”, current + 42);
cache.unlock(“key”);
```

Annotations

- JSR107 introduces a standardized set of caching annotations, which do **method level caching interception** on annotated classes running in **dependency injection containers.**

- Caching annotations are becoming increasingly popular:
  - Ehcache Annotations for Spring
  - Spring 3's caching annotations.

- JSR107 Annotations will be added to:
  - Java EE 8 (planned?)
  - Spring 4.1 (released)

Annotation Operations

The JSR107 annotations cover the most common cache operations:

- `@CacheResult`
- `@CachePut`
- `@CacheRemove`
- `@CacheRemoveAll`

Fully Annotated Class Example

```
@CacheDefaults(cacheName = "blogManager")
public class BlogManager {

    @CacheResult
    public Blog getBlogEntry(String title) {...}

    @CacheRemove
    public void removeBlogEntry(String title) {...}

    @CacheRemoveAll
    public void removeAllBlogs() {...}

    @CachePut
    public void createEntry(@CacheKey String title,
        @CacheValue Blog blog) {...}

    @CacheResult
    public Blog getEntryCached(String randomArg,
        @CacheKey String title){...}

}
```

JCache With Spring

- Spring
  - uses JCache since 4.1 http://bit.ly/1V0q1Kp
  - Added support for JCache cache annotations which can be mixed and matched with Spring ones
- Spring Boot
  - Auto-configuration for any JCache Provider http://bit.ly/1TPQKLx

JCache With Java EE

- Java EE
  - Can add JCache and an implementation to any Java EE app by adding the jars and configuring it outside of EE.
- Java EE 8
  - JCache added to EE8
  - Add JCache Annotations
  - Other integration possibilities:
    - ejb timer store
    - jbatch store
    - JPA
  - Adam Bien is prepared to lead a JSR to get JCache into EE8. 4 contributors so far. See See https://abhirockzz.wordpress.com/2016/01/21/jcache-in-java-ee-8/
  - Please see me at the Hazelcast booth if you want to join in.

The Future

- **JCache 1.1 (2016)**
  - Maintenance Release being worked on, by me ☺
  - Just bug fixes
- **Java EE 8 Integration (2017)**
- **JCache 2.0 (Later)**
  - Transactions
  - Async API
  - Servlet 4.0 Integration / Session Caching

Working with Hazelcast JCache

Hazelcast JCache Support

Full TCK Compliant implementation for:

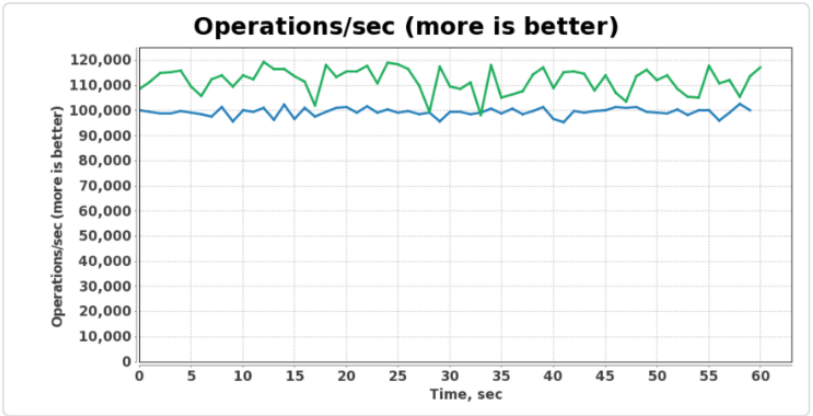
- Features:
- Embedded Members
- Clients (caches are stored in Members)
- HD Memory Store for members and near cache
- Very Fast Persistence with the Hot Restart Store
- Docs: http://bit.ly/1Q52yDz

Hazelcast JCache Performance

- Fastest IMDG using competitors own benchmarks
- See http://bit.ly/1T36n1m
- We added JCache put/get to Yardstick and will add it to RadarGun.

Color	Benchmark	Configurations
	HazelcastPutGetJcacheBenchmark	hz-jcache-put-get-3.6-EA
	IgnitePutGetBenchmark	ignite-full-syncatomic-put-

ThroughputLatencyProbe



	Avg	Min	Max	SD
	111,997.38	98,377.00	119,243.00	4,930.86
	99,411.22	95,286.00	102,625.00	1,627.75

Questions?

Greg Luck

- @gregrluck
- greg@hazelcast.com

Thank you