hazelcast

Thinking Distributed: The Hazelcast Way

RAHUL GUPTA

SR SOLUTIONS ARCHITECT

About Me

Rahul Gupta

Senior Solutions Architect

Worked with Terracotta

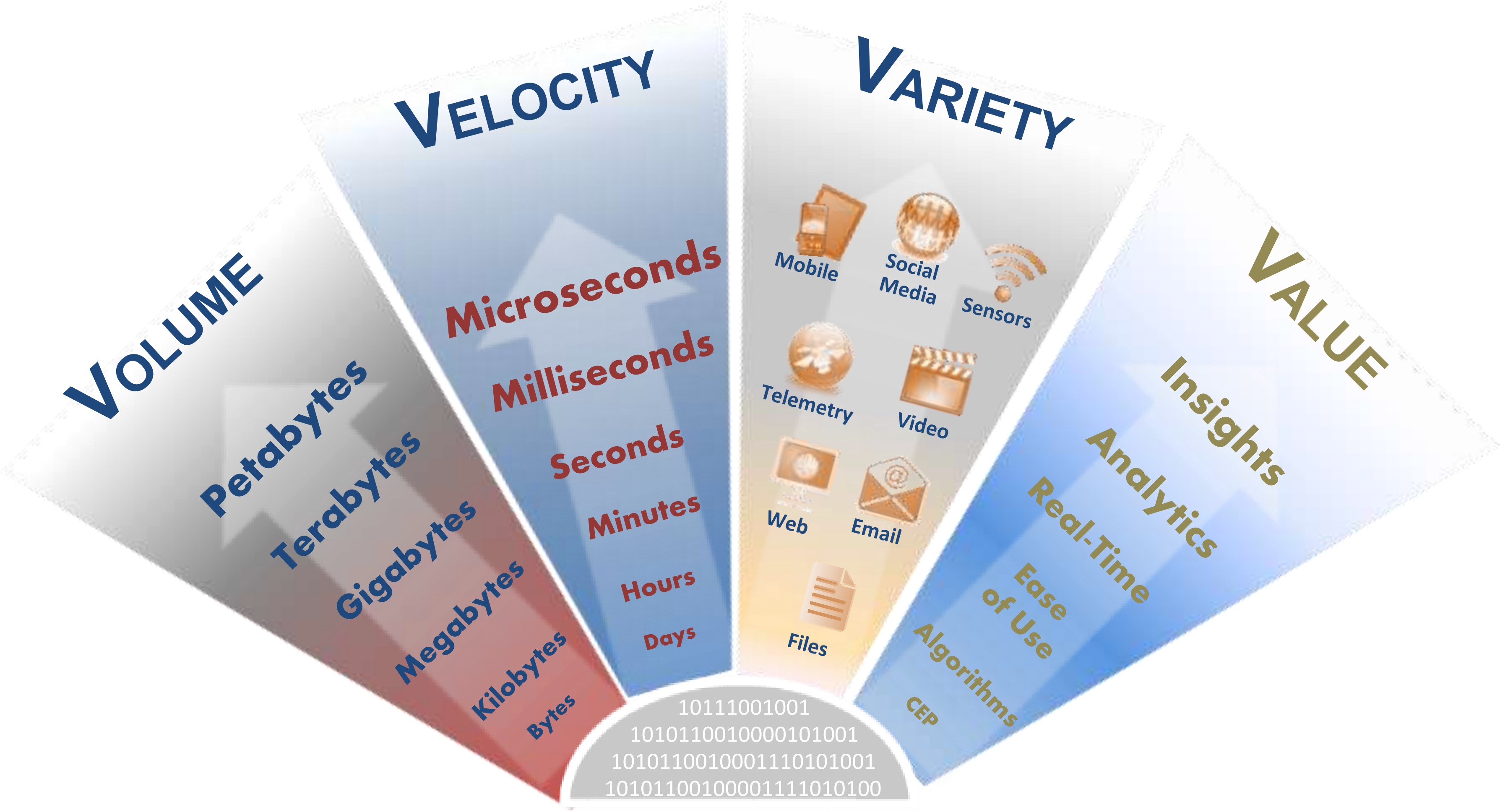
In-memory Distributed Systems since 2009

Java Programmer since 1998

rahul@hazelcast.com

follow me @wildnez

Challenges


VOLUME
Petabytes
Terabytes
Gigabytes
Megabytes
Kilobytes
Bytes
VELOCITY
Microseconds
Milliseconds
Seconds
Minutes
Hours
Days
VARIETY
Mobile
Social Media
Sensors
Telemetry
Video
Web
Email
Files
VALUE
Insights
Analytics
Real-Time
Ease of Use
Algorithms
CEP
10111001001
1010110010000101001
1010110010001110101001
10101100100001111010100

Why Hazelcast?

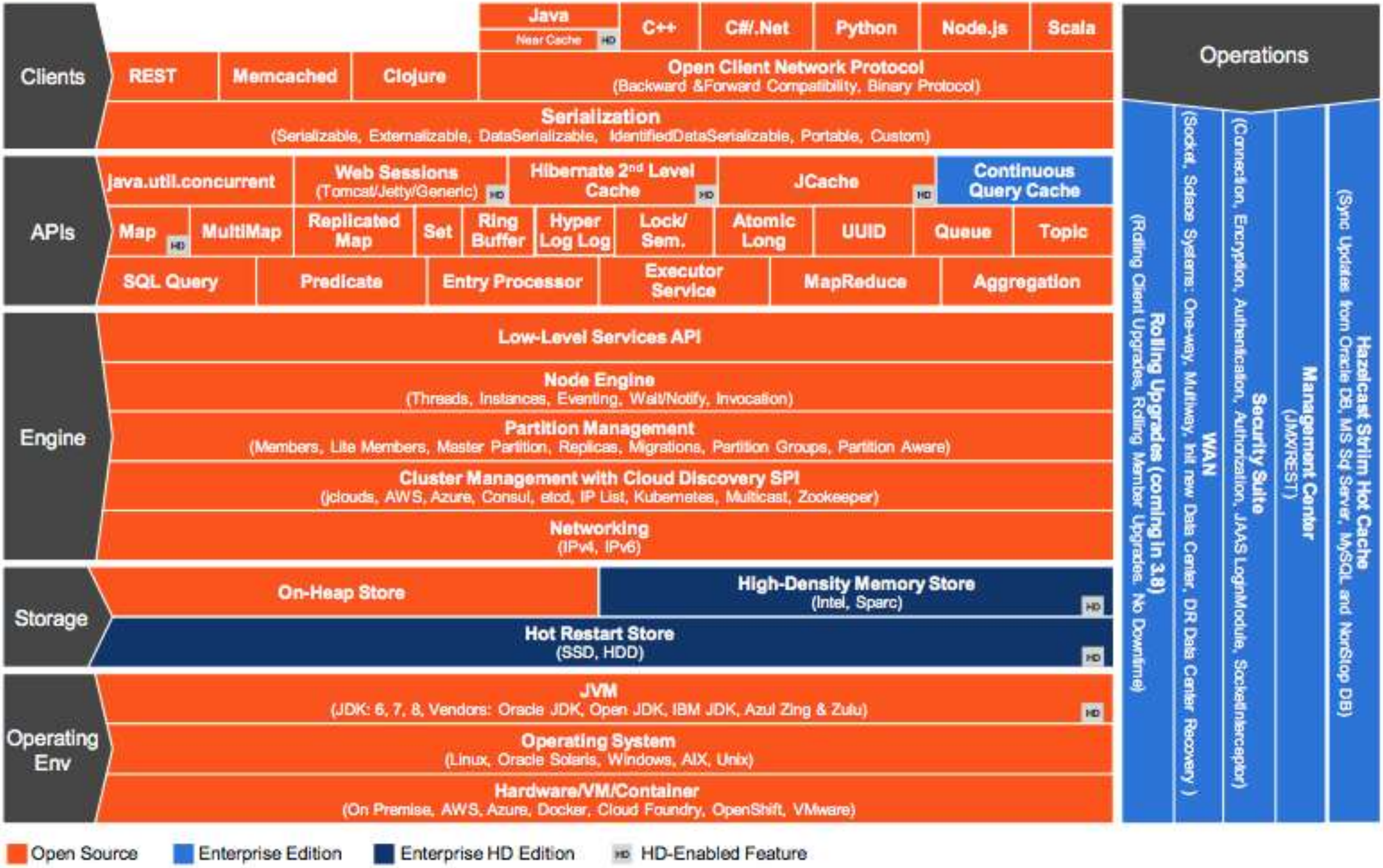
- **Scale-out Computing** enables cluster capacity to be increased or decreased on-demand

- **Resilience** with automatic recovery from member failures without losing data while minimizing performance impact on running applications

- **Programming Model** provides a way for developers to easily program a cluster application as if it is a single process

- **Fast Application Performance** enables very large data sets to be held in main memory for real-time performance

Hazelcast IMDG 3.7/3.8



Clients

Java

Near Cache HD

C++

C#/.Net

Python

Node.js

Scala

REST

Memcached

Clojure

Open Client Network Protocol
(Backward &Forward Compatibility, Binary Protocol)

Serialization
(Serializable, Externalizable, DataSerializable, IdentifiedDataSerializable, Portable, Custom)

APIs

java.util.concurrent

Web Sessions
(Tomcat/Jetty/Generic) HD

Hibernate 2nd Level Cache HD

JCache HD

Continuous Query Cache

Map HD

MultiMap

Replicated Map

Set

Ring Buffer

Hyper Log Log

Lock/ Sem.

Atomic Long

UUID

Queue

Topic

SQL Query

Predicate

Entry Processor

Executor Service

MapReduce

Aggregation

Engine

Low-Level Services API

Node Engine
(Threads, Instances, Eventing, Wait/Notify, Invocation)

Partition Management
(Members, Lite Members, Master Partition, Replicas, Migrations, Partition Groups, Partition Aware)

Cluster Management with Cloud Discovery SPI
(jclouds, AWS, Azure, Consul, etcd, IP List, Kubernetes, Multicast, Zookeeper)

Networking
(IPv4, IPv6)

Storage

On-Heap Store

High-Density Memory Store
(Intel, Sparc) HD

Hot Restart Store
(SSD, HDD) HD

Operating Env

JVM
(JDK: 6, 7, 8, Vendors: Oracle JDK, Open JDK, IBM JDK, Azul Zing & Zulu) HD

Operating System
(Linux, Oracle Solaris, Windows, AIX, Unix)

Hardware/VM/Container
(On Premise, AWS, Azure, Docker, Cloud Foundry, OpenShift, VMware)

Operations

Rolling Upgrades (coming in 3.8)
(Rolling Client Upgrades, Rolling Member Upgrades, No Downtime)

WAN
(Socket, Solace Systems; One-way, Multiway, Init new Data Center, DR Data Center Recovery)

Security Suite
(Connection, Encryption, Authentication, Authorization, JAAS LoginModule, SocketInterceptor)

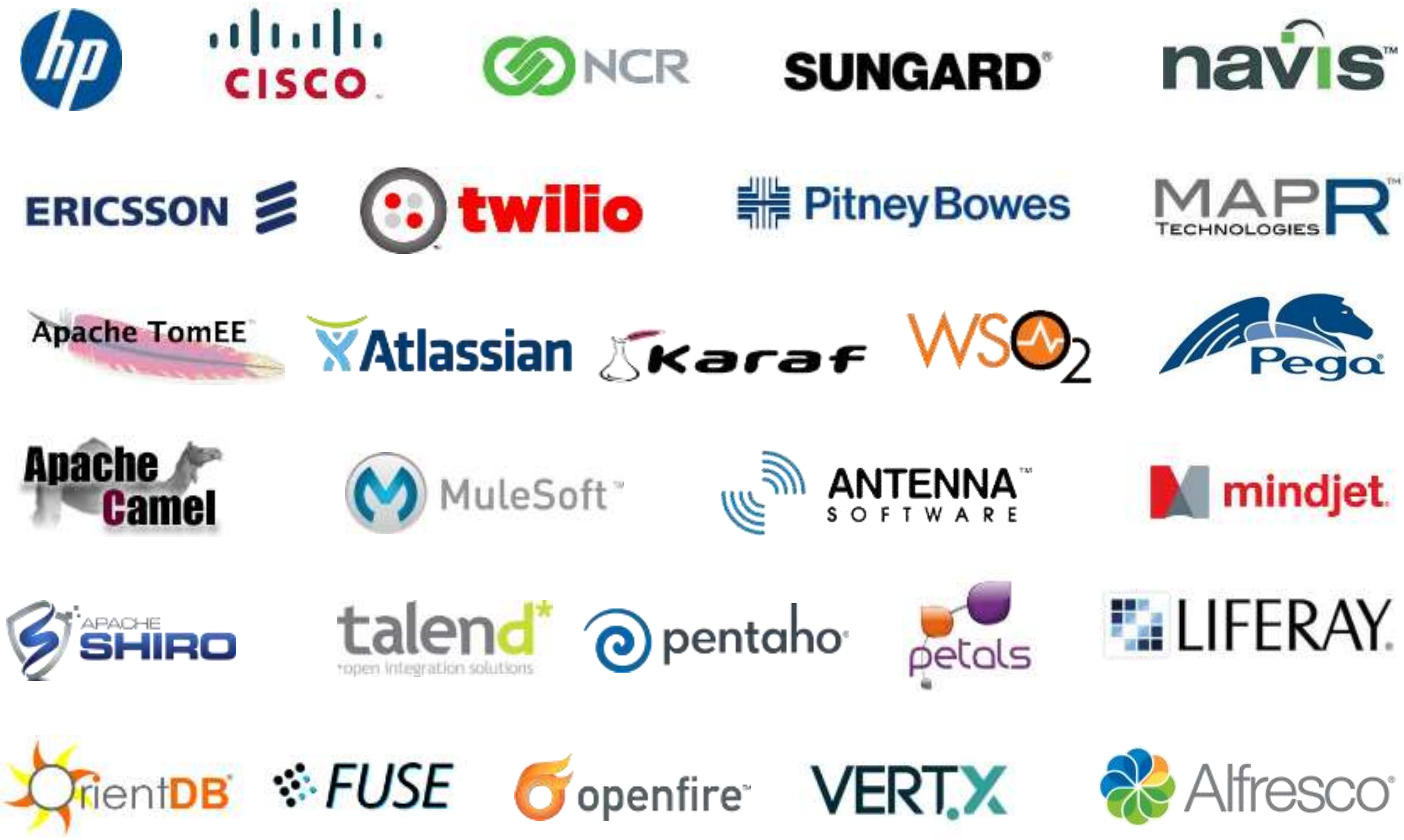
Management Center
(JMX/REST)

Hazelcast Striim Hot Cache
(Sync Updates from Oracle DB, MS Sql Server, MySQL and NonStop DB)

🟧 Open Source 🟦 Enterprise Edition ■ Enterprise HD Edition HD HD-Enabled Feature

Ecosystem Traction

Dozens of Commercial and Open Source Projects Embed Hazelcast



In Memory Data Grid

In Memory
Data Storage

+

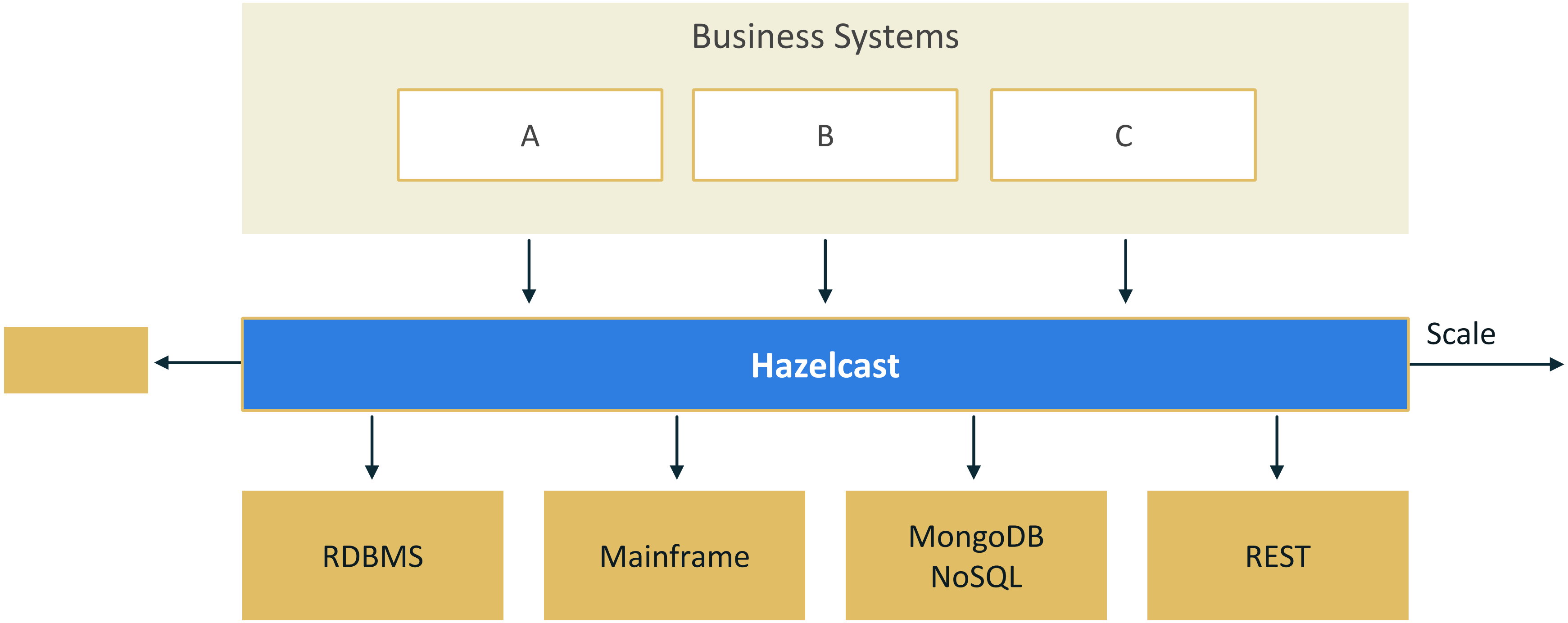
In Memory
Data Computing

+

01001
10101
01010

In Memory
Data Messaging

In-Memory Caching



Business Systems

A

B

C

Hazelcast

Scale

RDBMS

Mainframe

MongoDB
NoSQL

REST

java.util.concurrent.ConcurrentMap

```
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

public static void main(String[] args) {
        ConcurrentMap<Integer, String> map = new ConcurrentHashMap<>();
        map.put(1, "Paris");
        map.put(2, "London");
        map.put(3, "San Francisco");

        String oldValue = map.remove(2);
}
```

Distributed Map

```
import java.util.concurrent.ConcurrentMap;
import com.hazelcast.core.Hazelcast;
import com.hazelcast.core.HazelcastInstance;

public static void main(String[] args) {
        HazelcastInstance h = Hazelcast.newHazelcastInstance();

        ConcurrentMap<Integer, String> map = h.getMap("myMap");
        map.put(1, "Paris");
        map.put(2, "London");
        map.put(3, "San Francisco");

        String oldValue = map.remove(2);
}
```

DEMO

Persistence API

```
public class MapStorage
    implements MapStore<String, User>, MapLoader<String, User> {
  // Some methods missing ...
  @Override public User load(String key) { return loadValueDB(key); }
  @Override public Set<String> loadAllKeys() { return loadKeysDB(); }
  @Override public void delete(String key) { deleteDB(key); }
  @Override public void store(String key, User value) {
    storeToDatabase(key, value);
  }
}

<map name="users">
  <map-store enabled="true">
    <class-name>com.hazelcast.example.MapStorage</class-name>
    <write-delay-seconds>0</write-delay-seconds>
  </map-store>
</map>
```

JCache API

```
// Retrieve the CachingProvider which is automatically baced by
// the chosen Hazelcast server or client provider
CachingProvider cachingProvider = Caching.getCachingProvider();

// Create a CacheManager
CacheManager cacheManager = cachingProvider.getCacheManager();

// Cache<String, String> cache = cacheManager
//     .getCache( name, String.class, String.class );

// Create a simple but typesafe configuration for the cache

CompleteConfiguration<String, String> config =
   new MutableConfiguration<String, String>()
        .setTypes( String.class, String.class );
```

JCache API

```
// Create and get the cache
Cache<String, String> cache = cacheManager
        .createCache( "example", config );
// Alternatively to request an already existing cache
Cache<String, String> cache = cacheManager
    .getCache( name, String.class, String.class );

// Put a value into the cache
cache.put( "world", "Hello World" );

// Retrieve the value again from the cache
String value = cache.get( "world" );

System.out.println( value );
```

Eviction

- Unless deleted, entries remain in the map.
- Use eviction policies to prevent OOM situations.
- Eviction Triggers run LFU or LRU.

time-to-live
max-idle-seconds
max-size (PER_NODE(number of entries),
PER_PARTITION(number of entries),
USED_HEAP_SIZE(mb),
USED_HEAP_PERCENTAGE(mb))

- Setting eviction-percentage removes that % of entries when eviction is triggered.

More Distributed Structures

Features	Description
MultiMap	Store multiple values against one Key
Replicated Map	Cluster wide replication, all entries on all nodes. Good for read heavy use cases
Near Cache	Map Entries on Client/Application local memory
RingBuffer	Stores data in a ring-like structure, like a circular array with given capacity

IM Data Store (Caching) Features

Java Collection API: Map, List, Set, Queue
Jcache
High Density Memory Store
Hibernate 2nd Level Cache
Web Session Replication: Tomcat, Jetty
Predicate API: Indexes, SQL Query
Persistence: Map/Queue Store & Loader. Write Behind/Through
Spring Compliance
Transactions: Local & XA
WAN & DR Replication

Java: Will it make the cut?

Garbage Collection limits heap usage. G1 and Balanced aim for <100ms at 10GB.

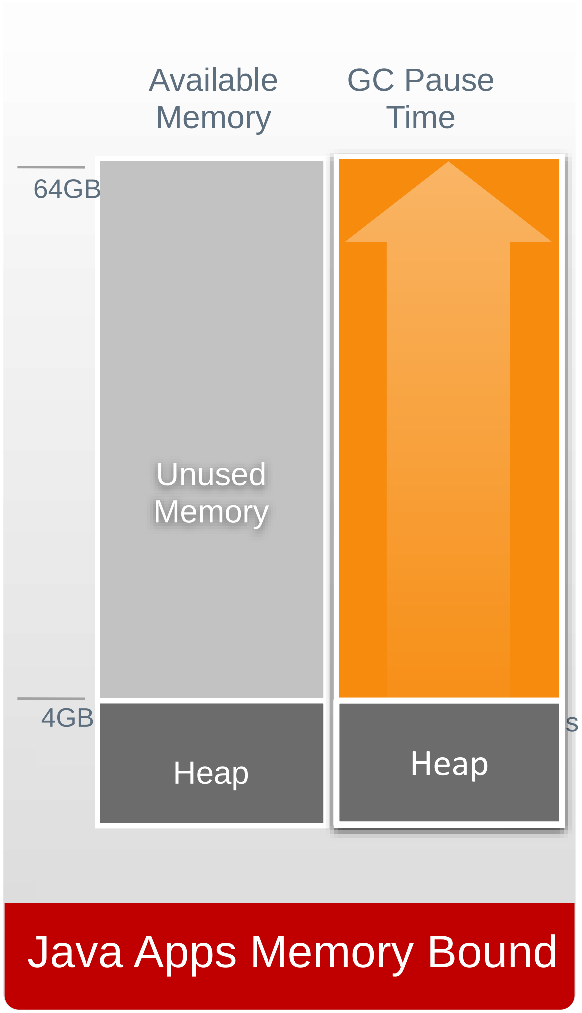
▼

Off-Heap Storage

No low-level CPU access

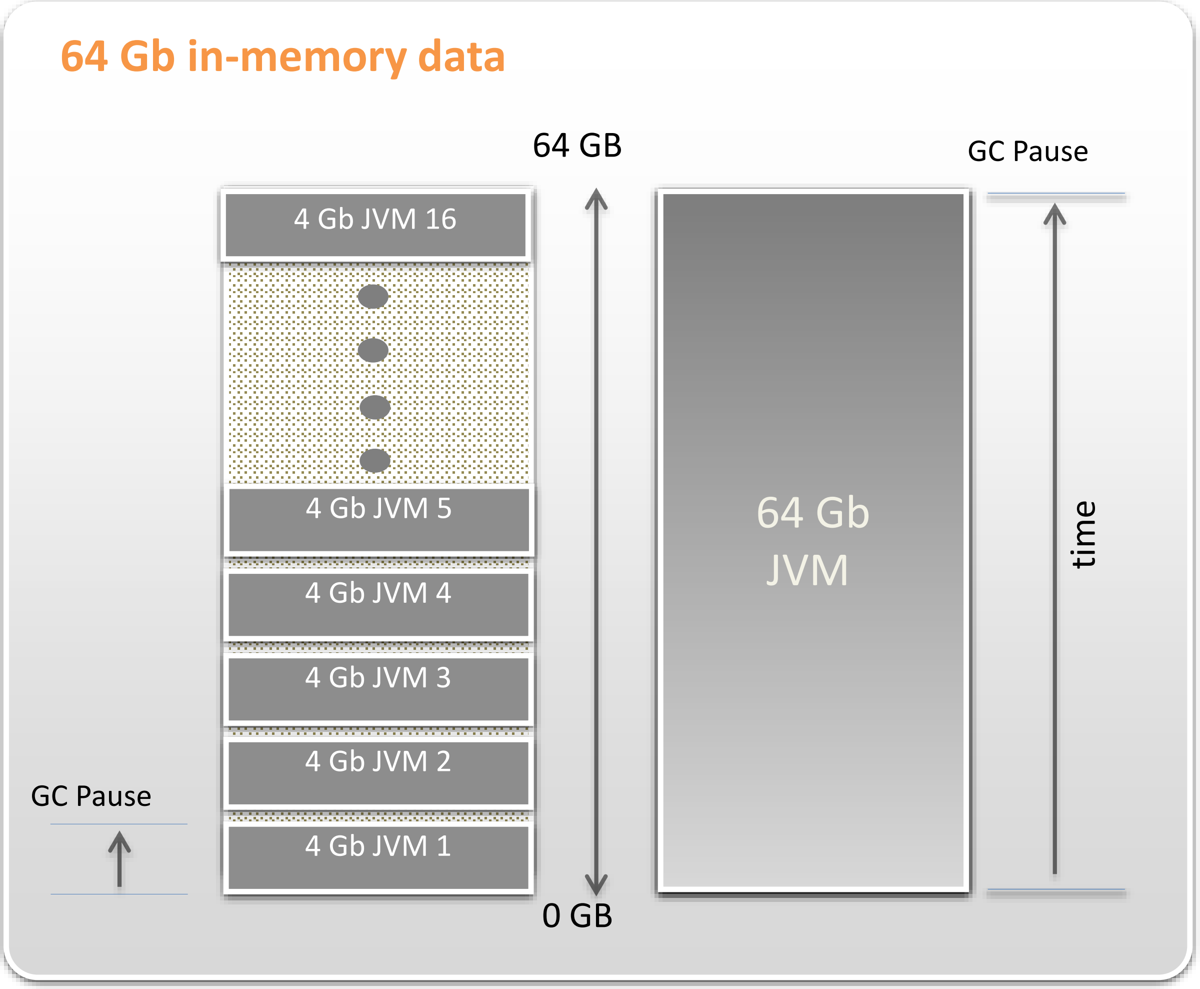
▼

Java is challenged as an infrastructure language despite its newly popular usage for this

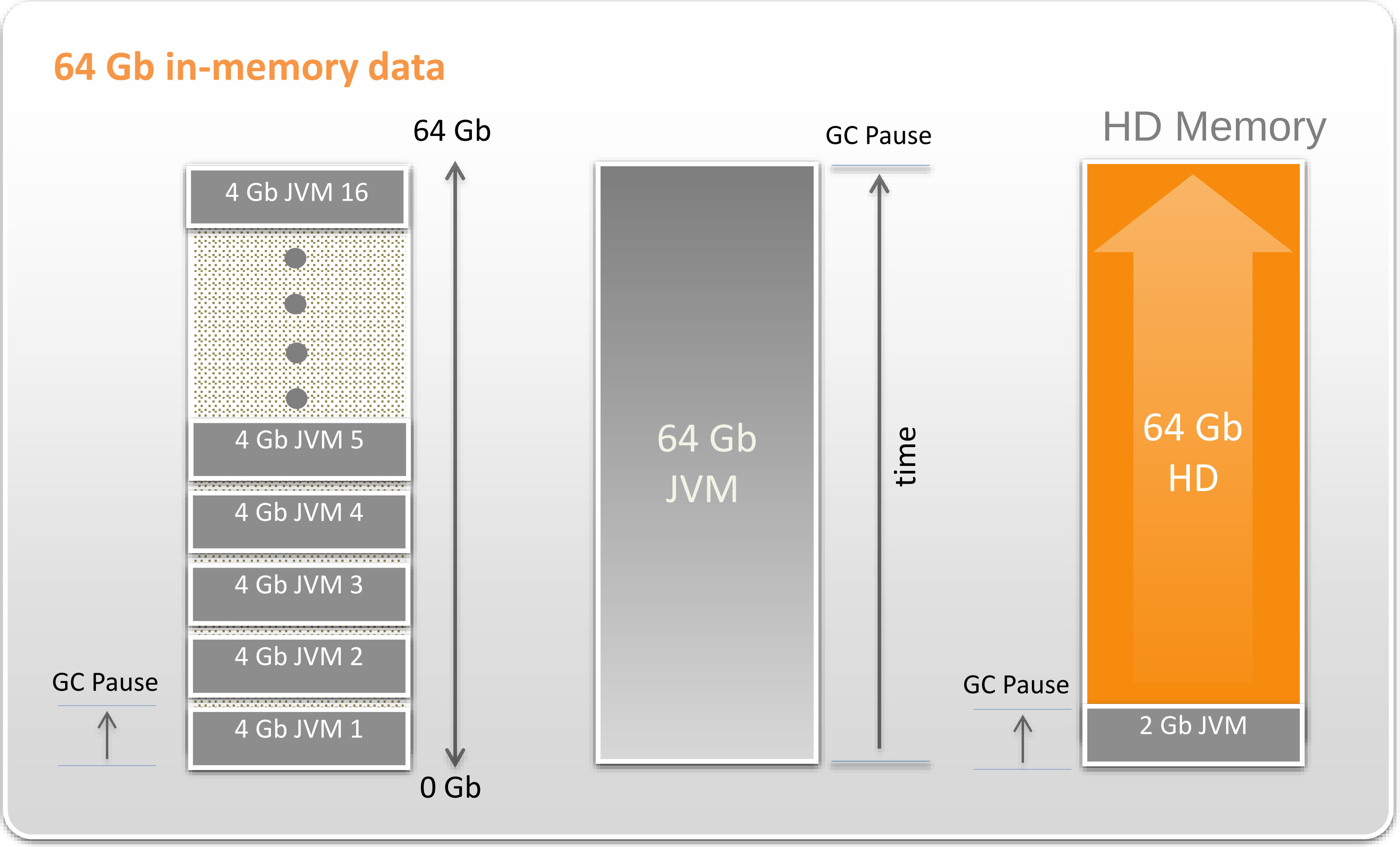


Java Apps Memory Bound

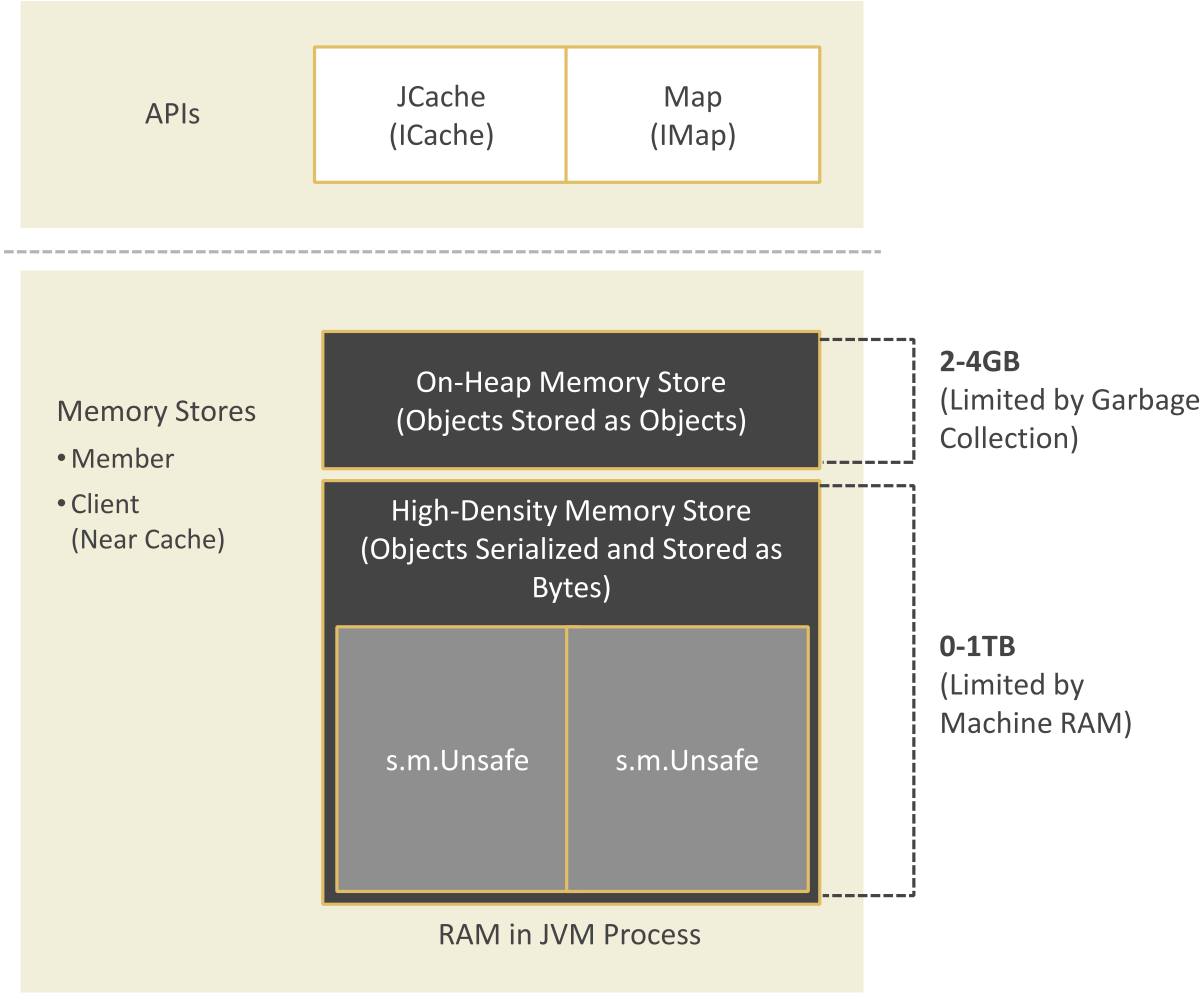
Standard Impediments of Caching



Caching with HD Memory

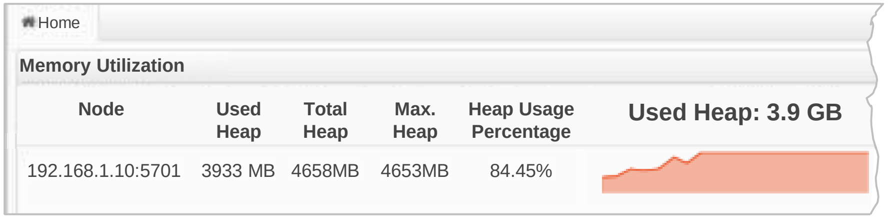


HD Memory


APIs
JCache
(ICache)
Map
(IMap)
Memory Stores
• Member
• Client
(Near Cache)
On-Heap Memory Store
(Objects Stored as Objects)
High-Density Memory Store
(Objects Serialized and Stored as Bytes)
s.m.Unsafe
s.m.Unsafe
RAM in JVM Process
2-4GB
(Limited by Garbage Collection)
0-1TB
(Limited by Machine RAM)

On Heap Vs. High-Density Memory Management

On Heap Memory		HD Memory
0 MB	*HD*	*3.3 GB*
3.9 GB	*Heap Storage*	*0.6 GB*
9 (4900 ms)	*Major GC*	*0 (0 ms)*
31 (4200 ms)	*Minor GC*	*356 (349 ms)*

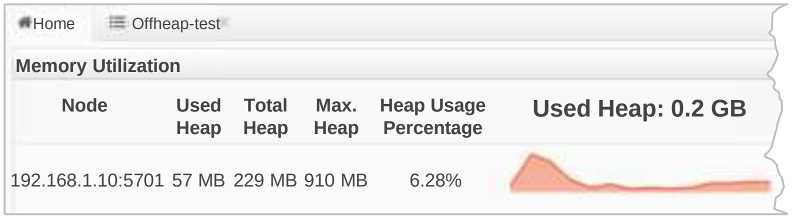


Memory Utilization

Node	Used Heap	Total Heap	Max. Heap	Heap Usage Percentage
192.168.1.10:5701	3933 MB	4658MB	4653MB	84.45%

Used Heap: 3.9 GB

Example: On Heap Memory



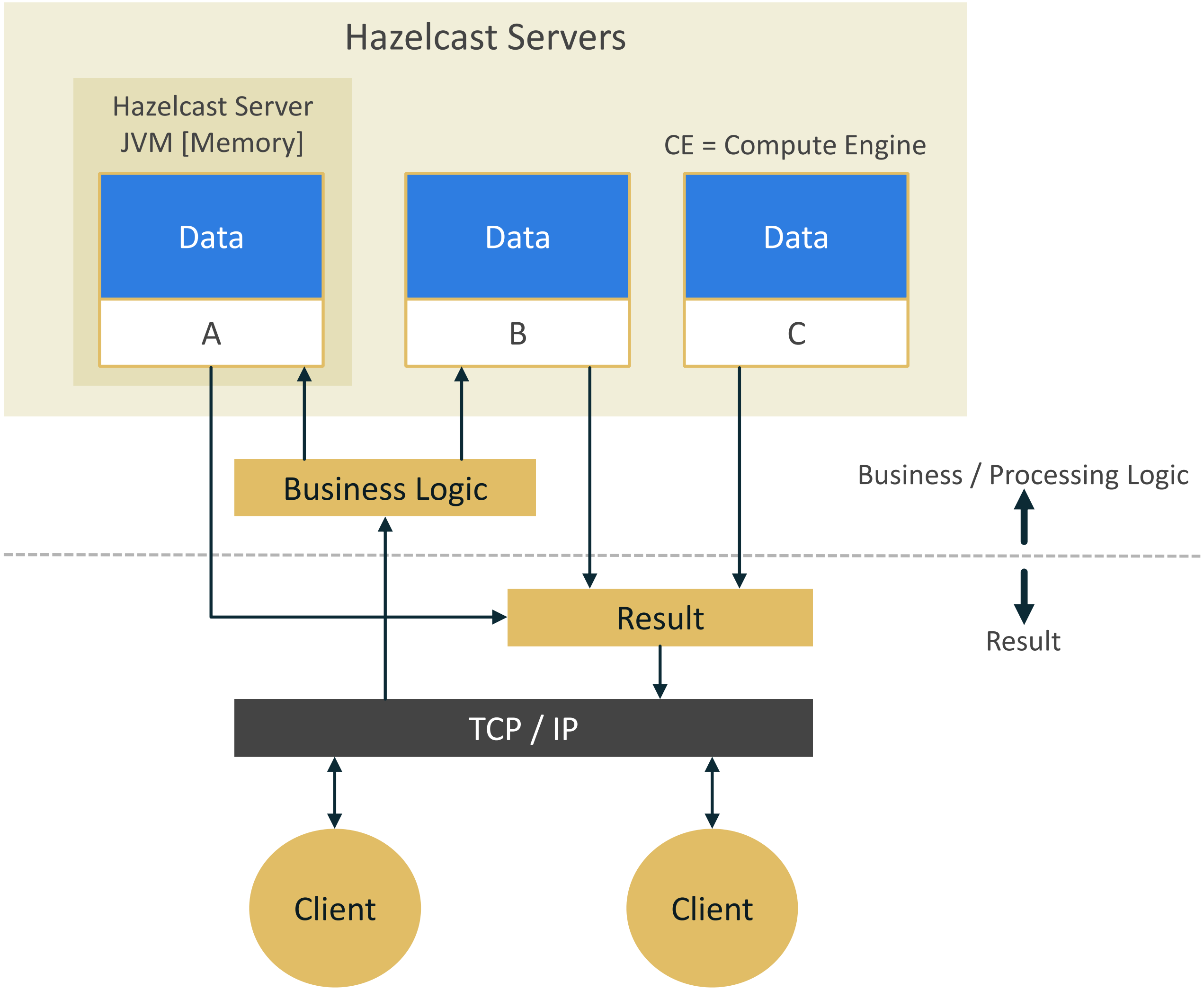
Memory Utilization

Node	Used Heap	Total Heap	Max. Heap	Heap Usage Percentage
192.168.1.10:5701	57 MB	229 MB	910 MB	6.28%

Used Heap: 0.2 GB

Example: HD Memory

Distributed Computing



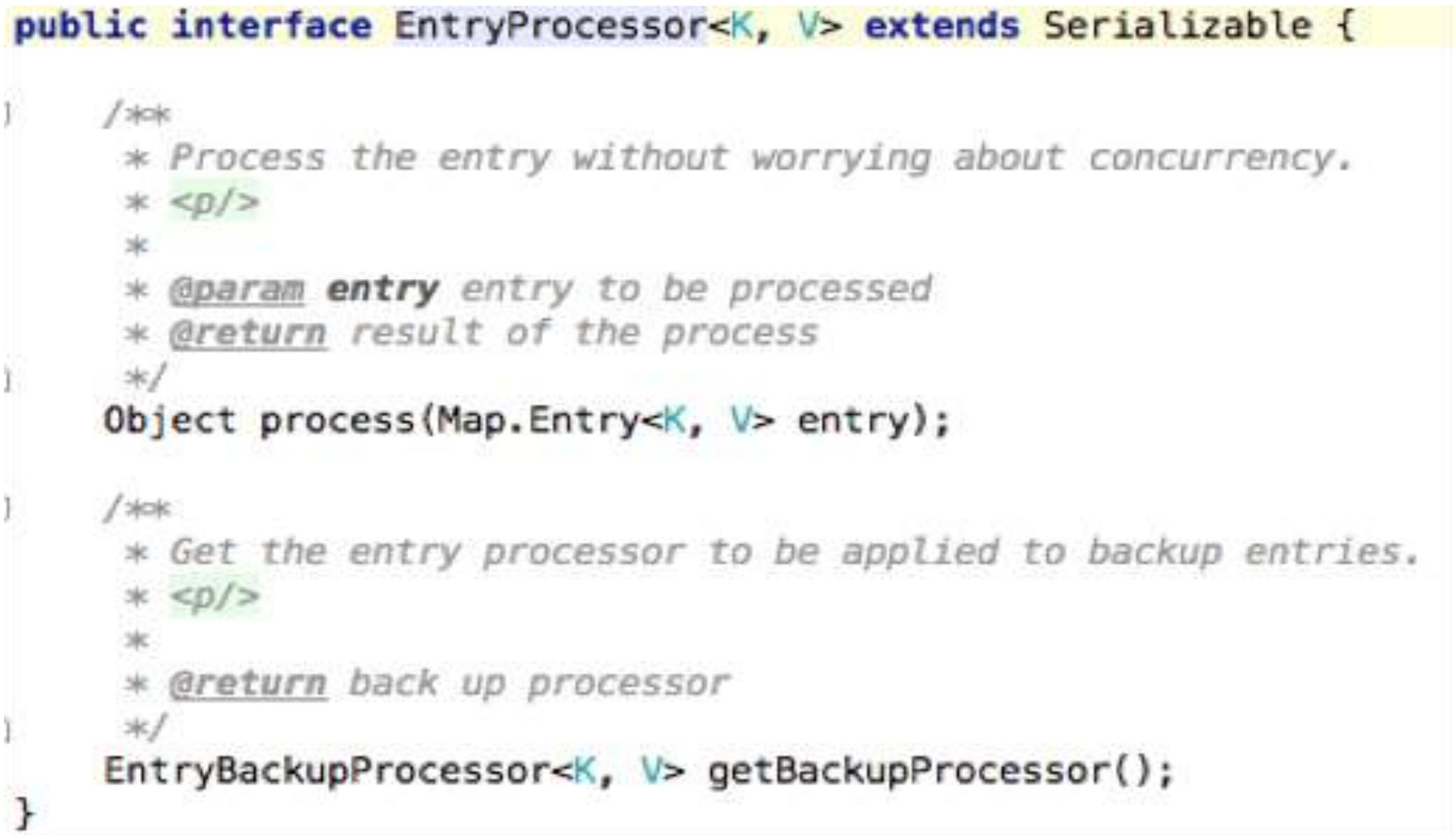
IM Distributed Computing Feature

Java Concurrency API (Lock, Semaphore, AtomicLong, AtomicReference, Executor Service, Blocking Queue)
Entry and Item Listeners
Entry Processor
Aggregators
Map/Reduce
Data Affinity
Continues Query
Map Interceptors
Delta Update

Executor Service API

```
public interface com.hazelcast.core.IExecutorService
                extends java.util.concurrent.ExecutorService

HazelcastInstance hz = getHazelcastInstance();

//java.util.concurrent.ExecutorService implementation
IExecutorService es = hz.getExecutorService("name");
es.executeOnAllMembers(buildRunnable());
es.executeOnKeyOwner(buildRunnable(), "Peter");
es.execute(buildRunnable());

Map<..> futures = es.submitToAllMembers(buildCallable());
Future<..> future = es.submitToKeyOwner(buildCallable(), "Peter");

es.submitToAllMembers(buildCallable(), buildCallback());
es.submitToKeyOwner(buildCallable(), "Peter", buildCallback());
```

EntryProcessor API



```java
public interface EntryProcessor<K, V> extends Serializable {

    /**
     * Process the entry without worrying about concurrency.
     * <p/>
     *
     * @param entry entry to be processed
     * @return result of the process
     */
    Object process(Map.Entry<K, V> entry);

    /**
     * Get the entry processor to be applied to backup entries.
     * <p/>
     *
     * @return back up processor
     */
    EntryBackupProcessor<K, V> getBackupProcessor();
}
```

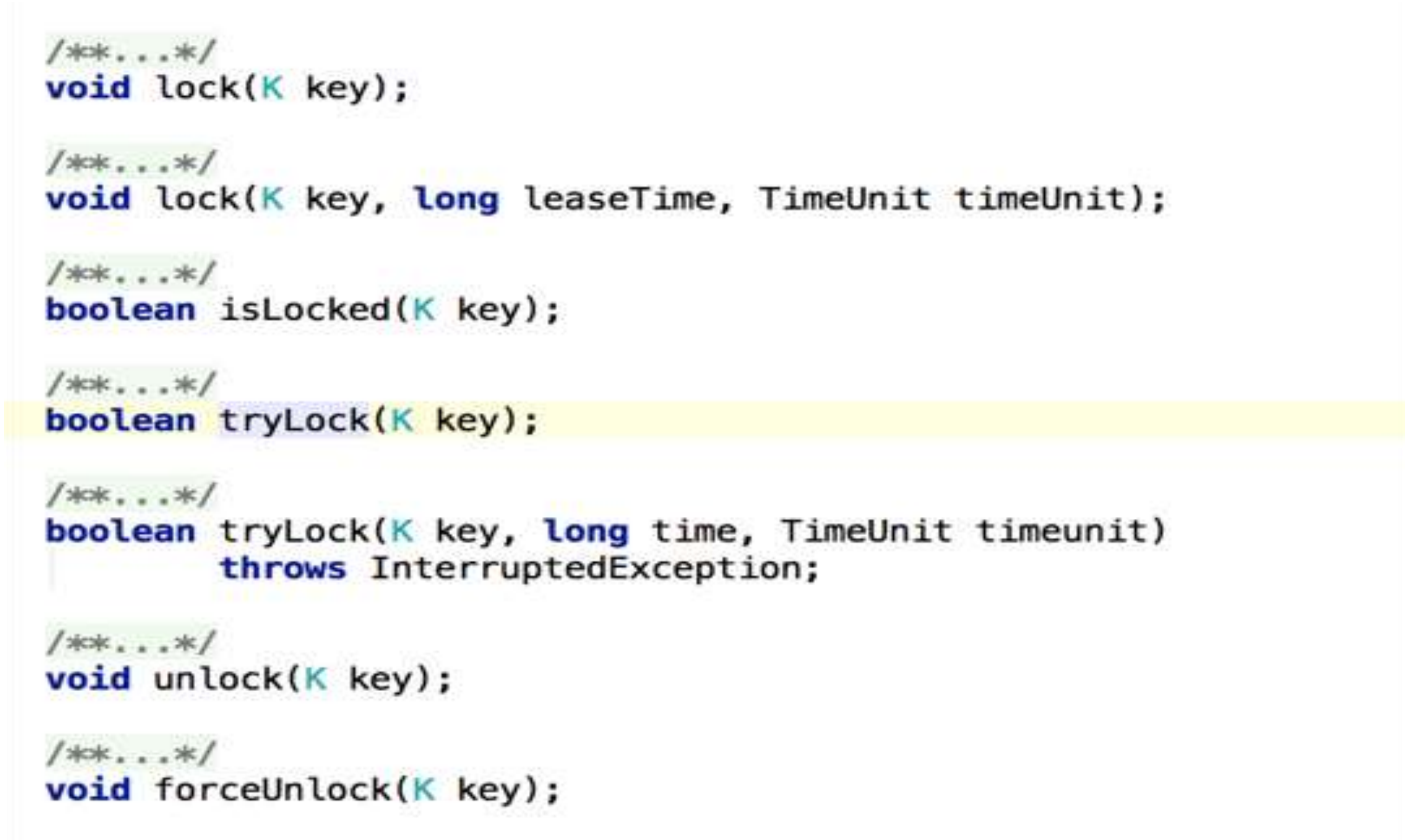
Lock API

Distributed Lock

```
HazelcastInstance hz = getHazelcastInstance();

// Distributed Reentrant
Lock lock = hz.getLock("myLock");
lock.lock();
try {
  // Do something
} finally {
  lock.unlock();
}
```

Lock API

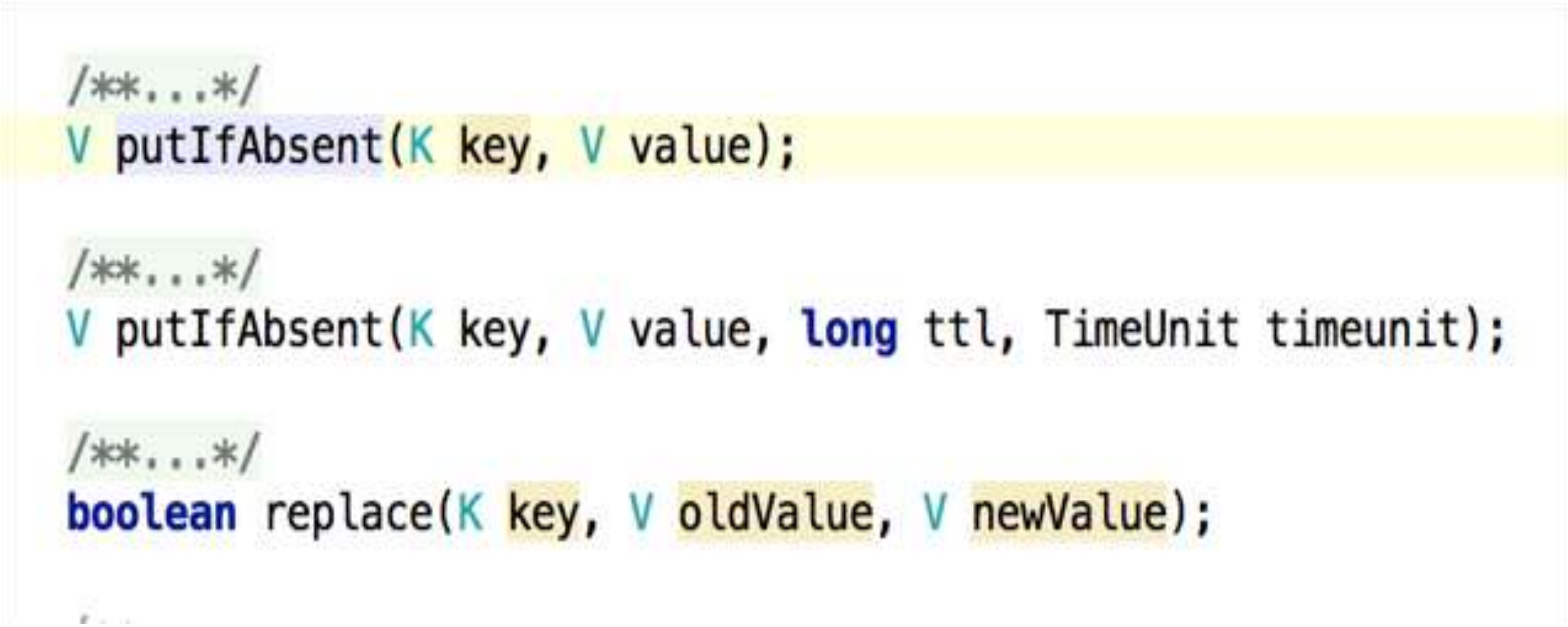
Pessimistic Locking (IMap)



```
/**...*/
void lock(K key);

/**...*/
void lock(K key, long leaseTime, TimeUnit timeUnit);

/**...*/
boolean isLocked(K key);

/**...*/
boolean tryLock(K key);

/**...*/
boolean tryLock(K key, long time, TimeUnit timeunit)
        throws InterruptedException;

/**...*/
void unlock(K key);

/**...*/
void forceUnlock(K key);
```

Lock API

Optimistic Locking



```
/**...*/
V putIfAbsent(K key, V value);

/**...*/
V putIfAbsent(K key, V value, long ttl, TimeUnit timeunit);

/**...*/
boolean replace(K key, V oldValue, V newValue);
```

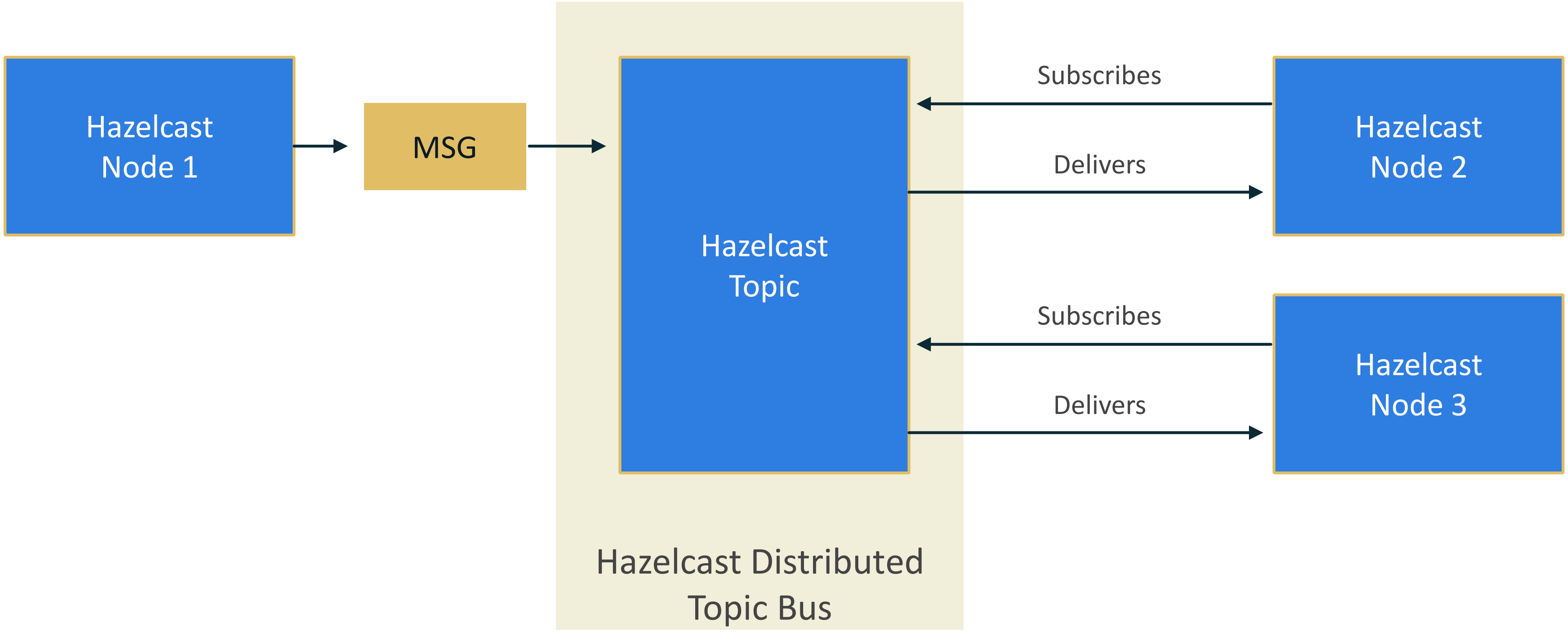
Map/Reduce API

```
HazelcastInstance hz = getHazelcastInstance();

Map users = hz.getMap("users");
JobTracker tracker = hz.getJobTracker("default");

KeyValueSource source = KeyValueSource.fromMap(users);
Job job = tracker.newJob(source);

ICompleteFuture future = job.mapper(new MyMapper())
               .reducer(new MyReducer())
               .submit();

Map result = future.get();
```

Aggregations API

```
HazelcastInstance hz = getHazelcastInstance();

Map users = hz.getMap("users");

int sum = users.aggregate(
    Supplier.all((user) -> user.getSalary()),
    Aggregations.longSum()
);
```

Distributed Messaging



IM Distributed Messaging Features

Queue

Topic (Pub/Sub)

Event Listeners

Ring Buffers

Queue API

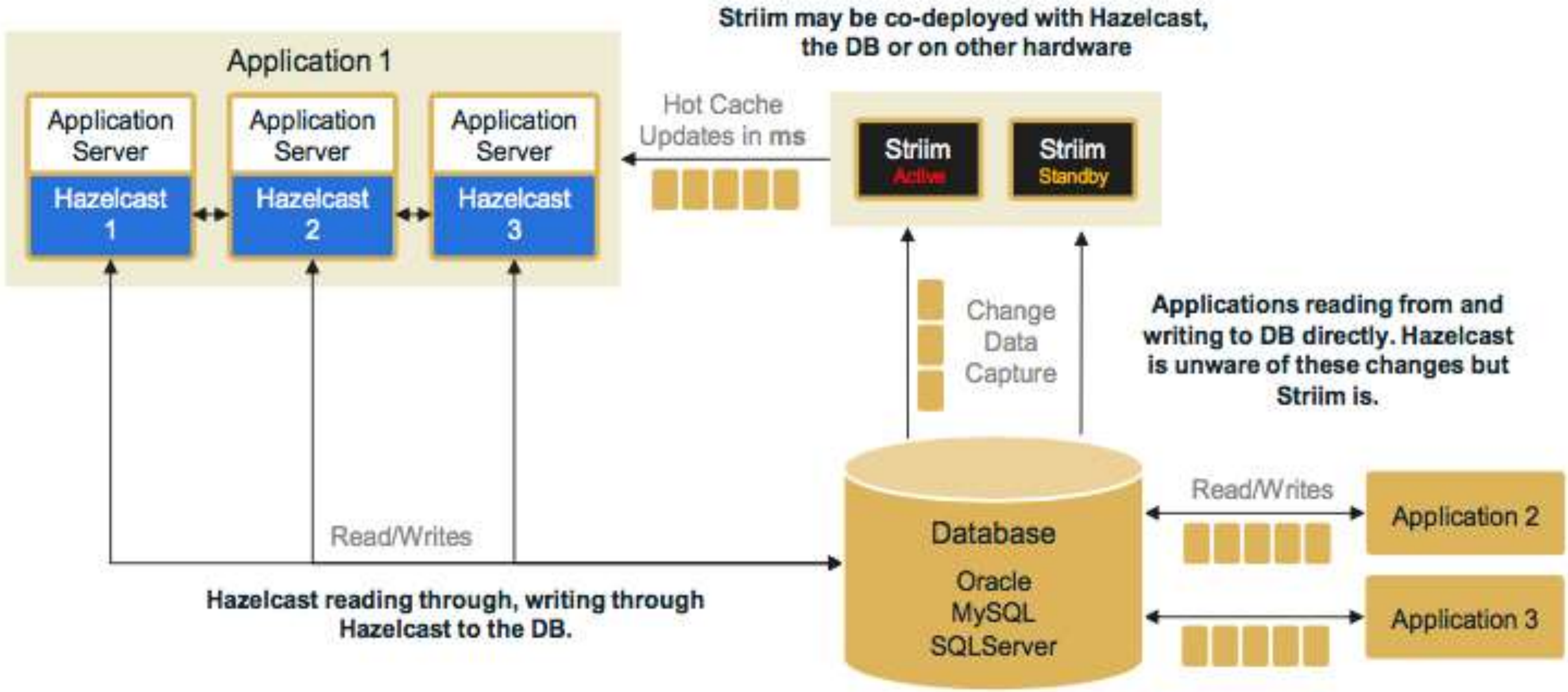
```
interface com.hazelcast.core.IQueue<E>
          extends java.util.concurrent.BlockingQueue

HazelcastInstance hz = getHazelcastInstance();

//java.util.concurrent.BlockingQueue implementation
IQueue<Task> queue = hz.getQueue("tasks");

queue.offer(newTask());
queue.offer(newTask(), 500, TimeUnit.MILLISECONDS);

Task task = queue.poll();
Task task = queue.poll(100, TimeUnit.MILLISECONDS);
Task task = queue.take();
```

Topic API

```
public class Example implements MessageListener<String> {
  public void sendMessage {
    HazelcastInstance hz = getHazelcastInstance();
    ITopic<String> topic = hz.getTopic("topic");
    topic.addMessageListener(this);
    topic.publish("Hello World");
  }

  @Override
  public void onMessage(Message<String> message) {
    System.out.println("Got message: " + message.getMessageObject());
  }
}
```

Hazelcast Striim Hot Cache

- Updates to the Database are pushed to Hazelcast Maps and Caches via Striim
- Solves the Consistency with the System of Record Problem, with updates applied in ms
- Configure your object relational mapping and any other transformations required in the Striim configuration.



Data Distribution and Resilience

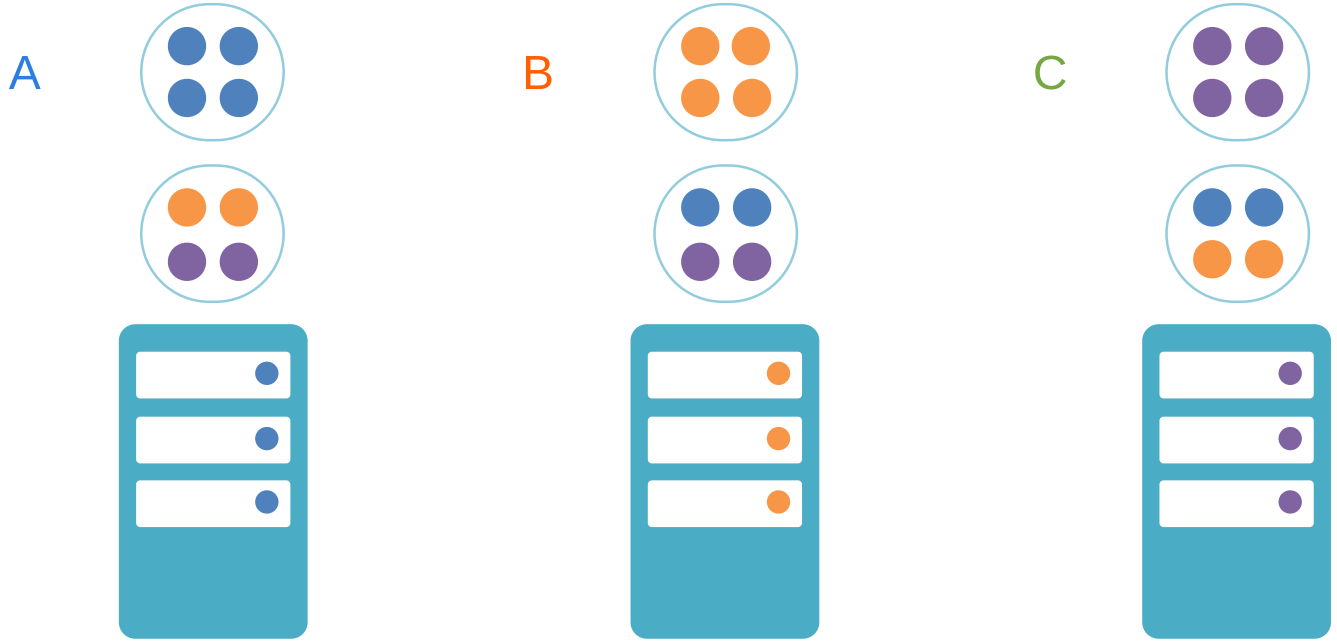
Distributed Maps

Fixed number of partitions (default 271)

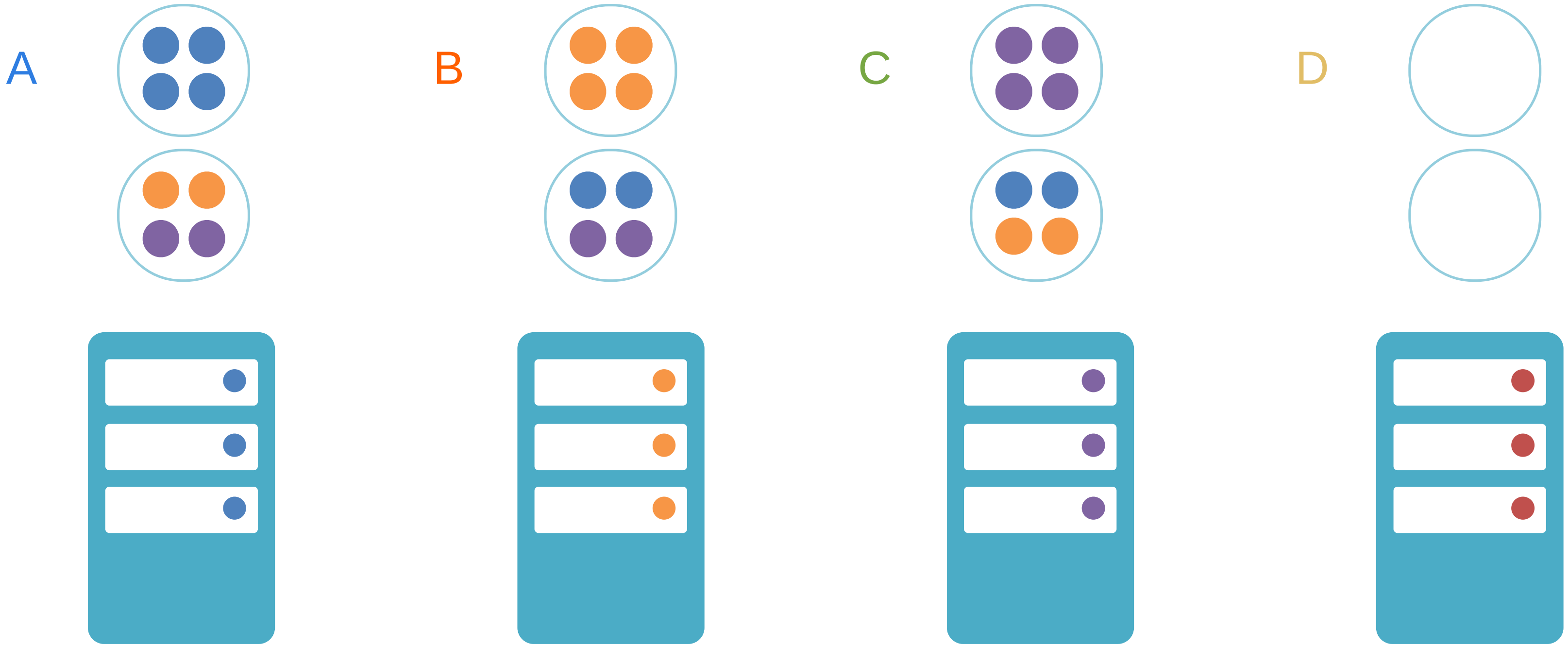
Each key falls into a partition

partitionId = hash(keyData)%PARTITION_COUNT

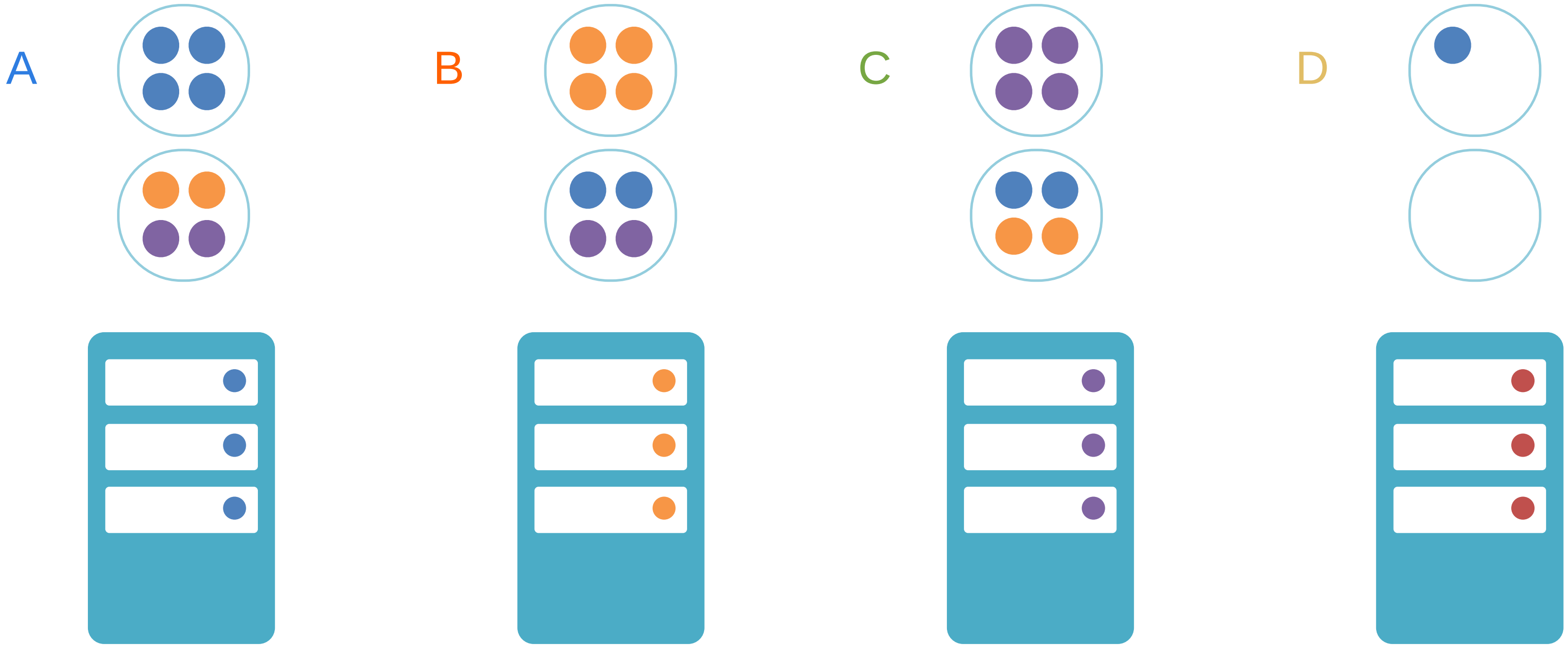
Partition ownerships are reassigned upon membership change



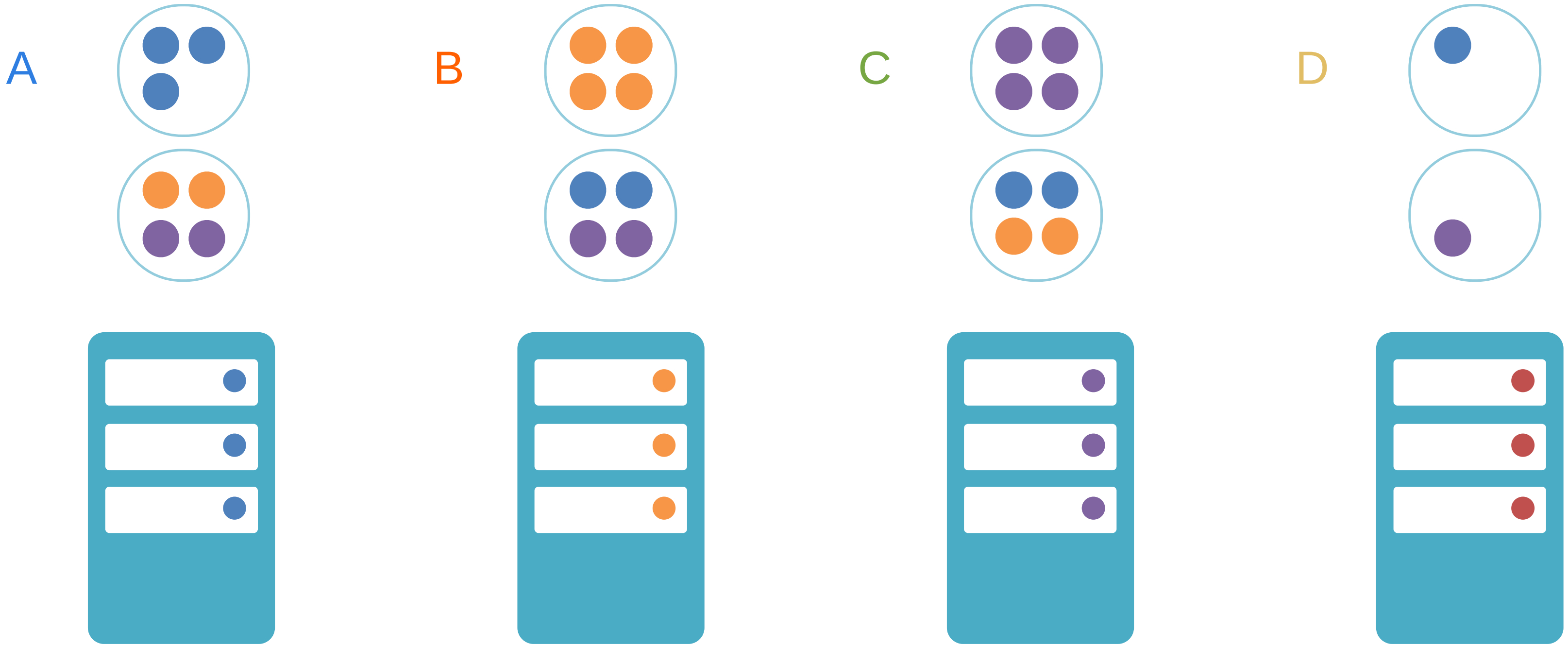
New Node Added



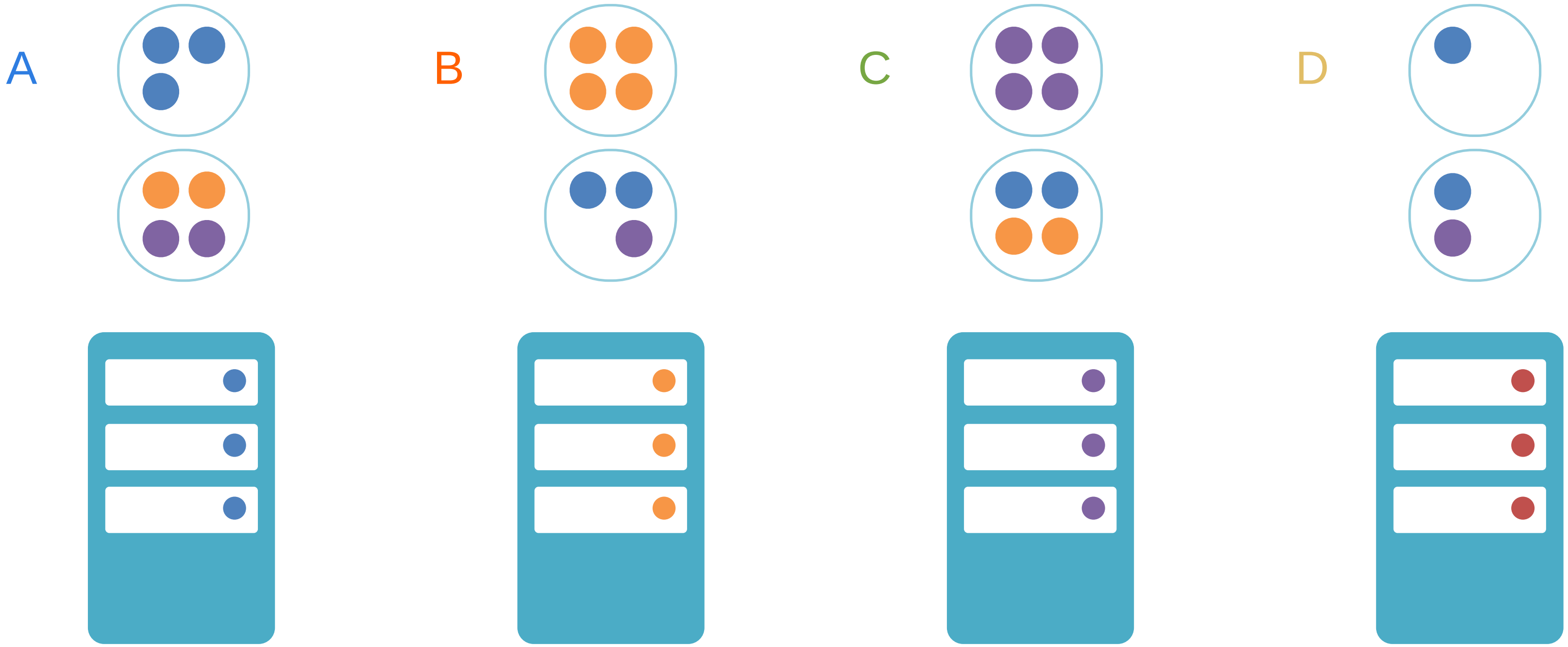
Migration



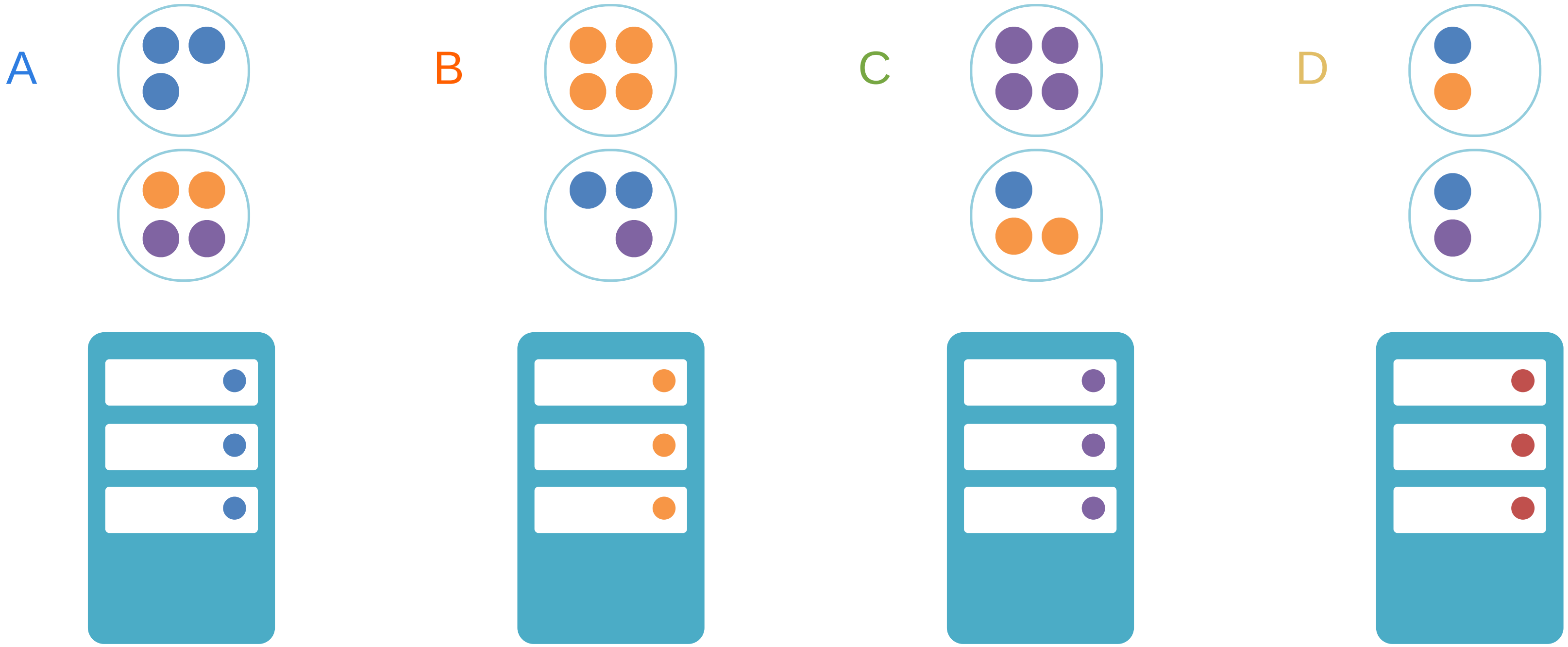
Migration



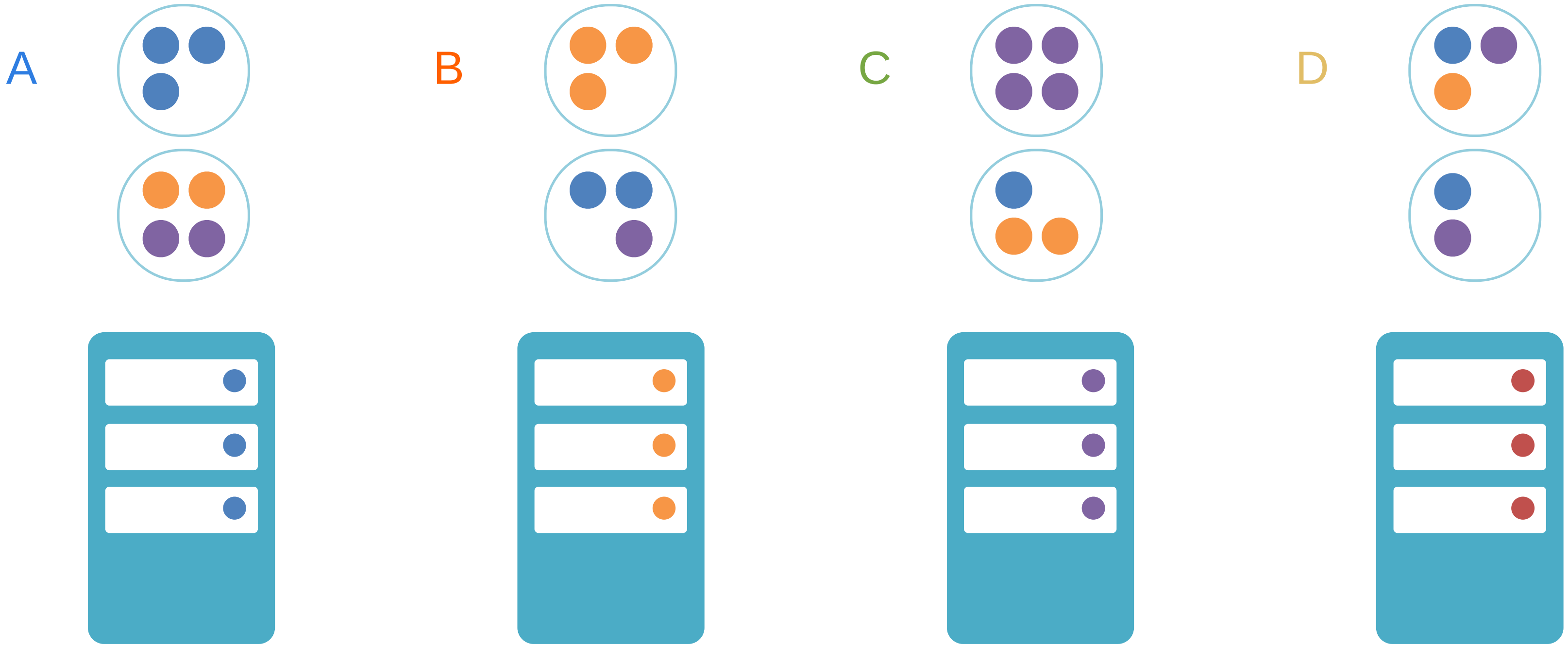
Migration



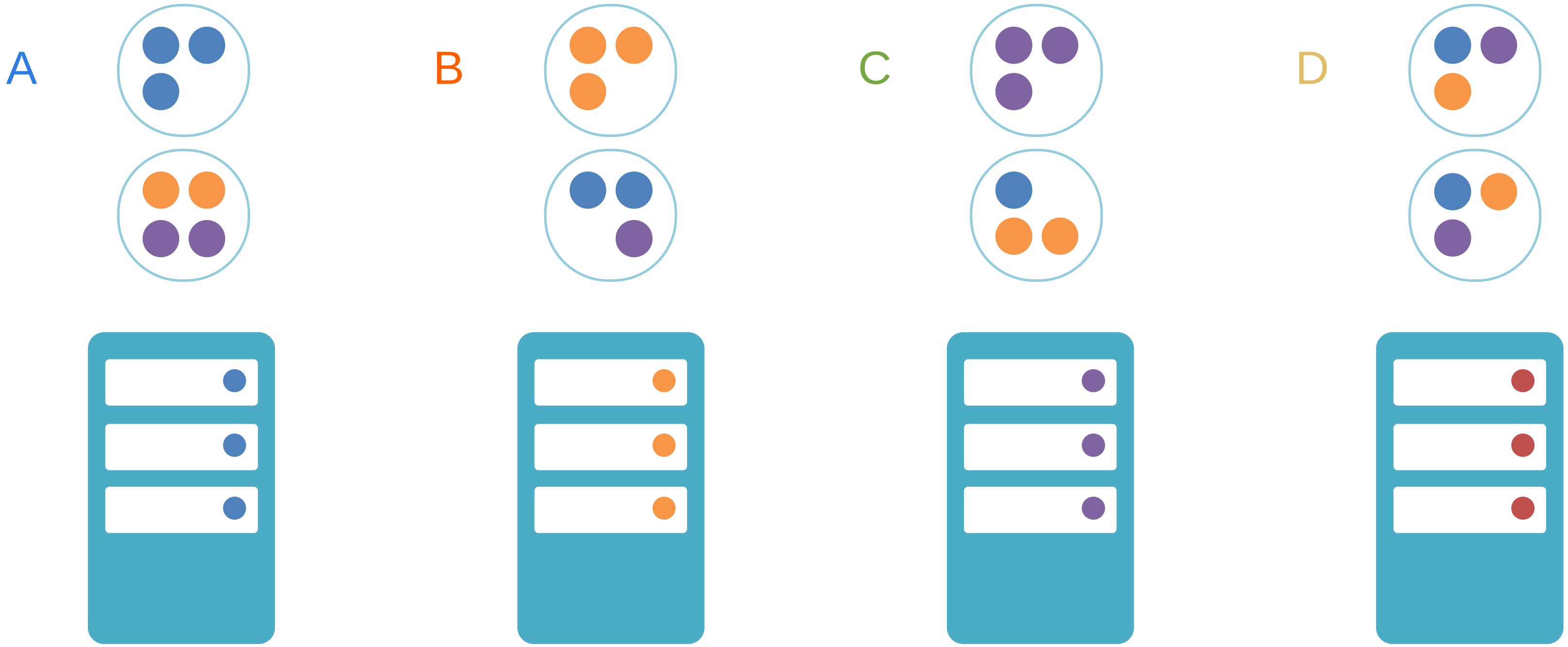
Migration



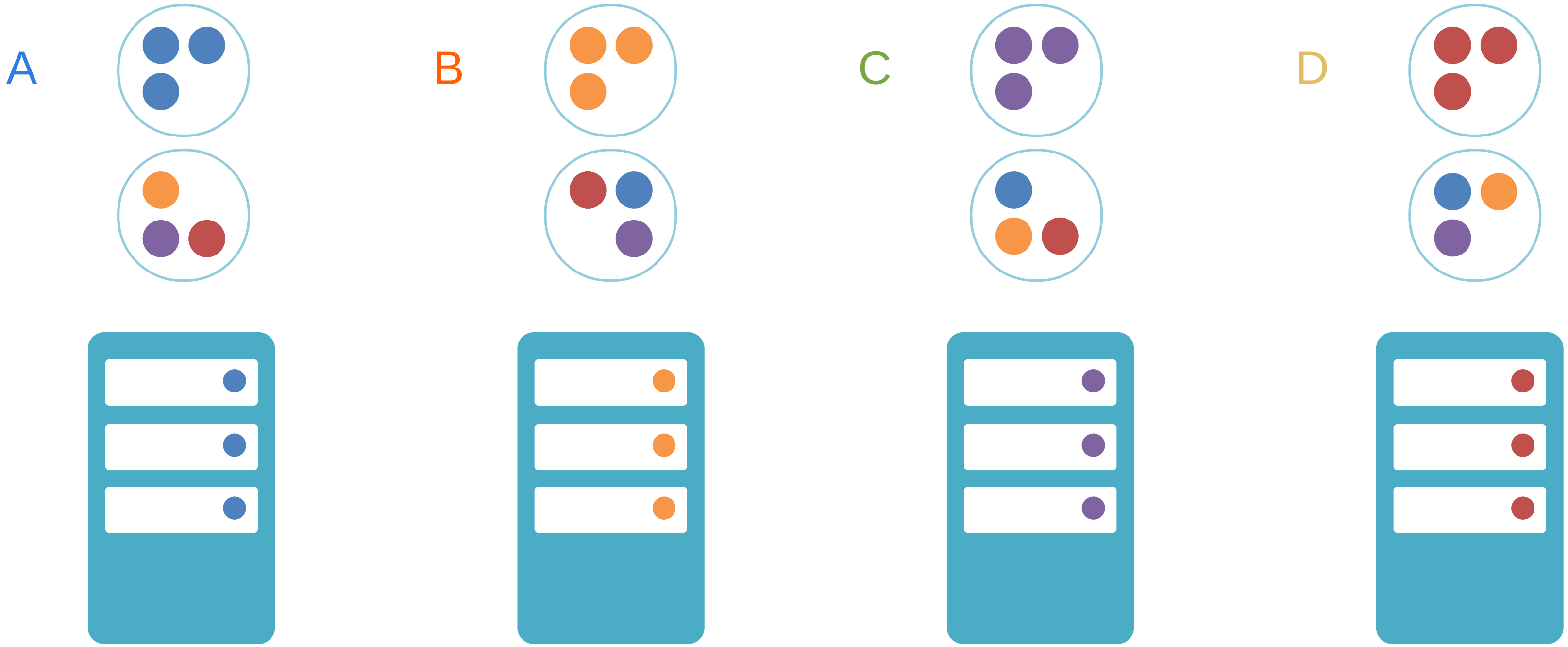
Migration



Migration

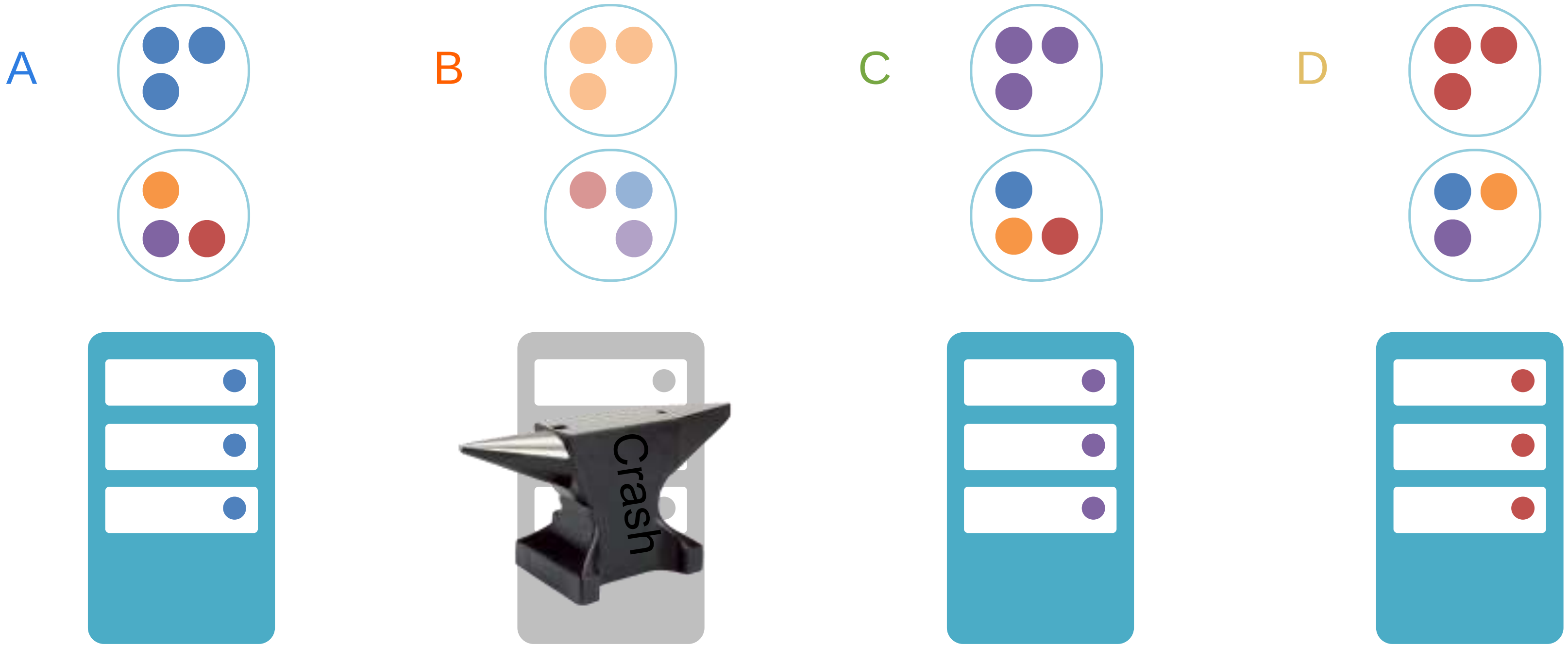


Migration Complete

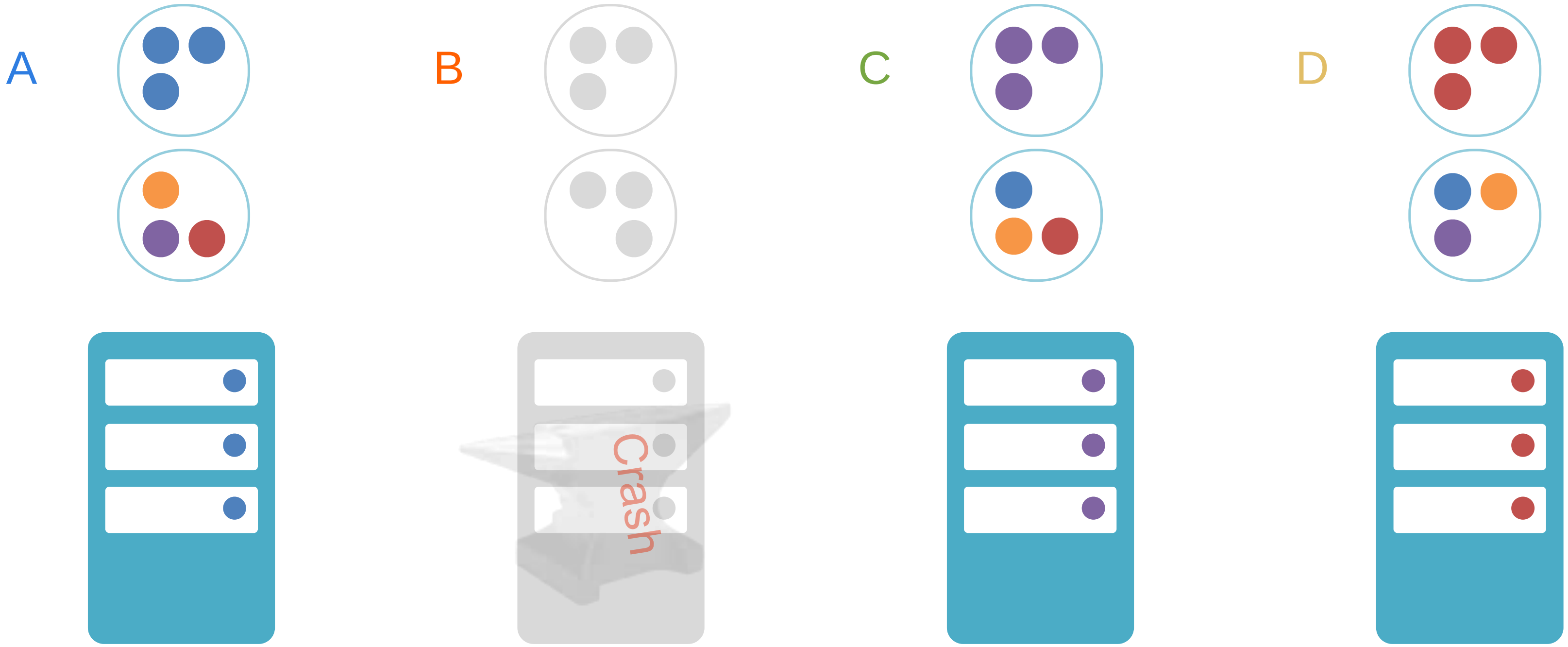


Data Safety on Node Failure

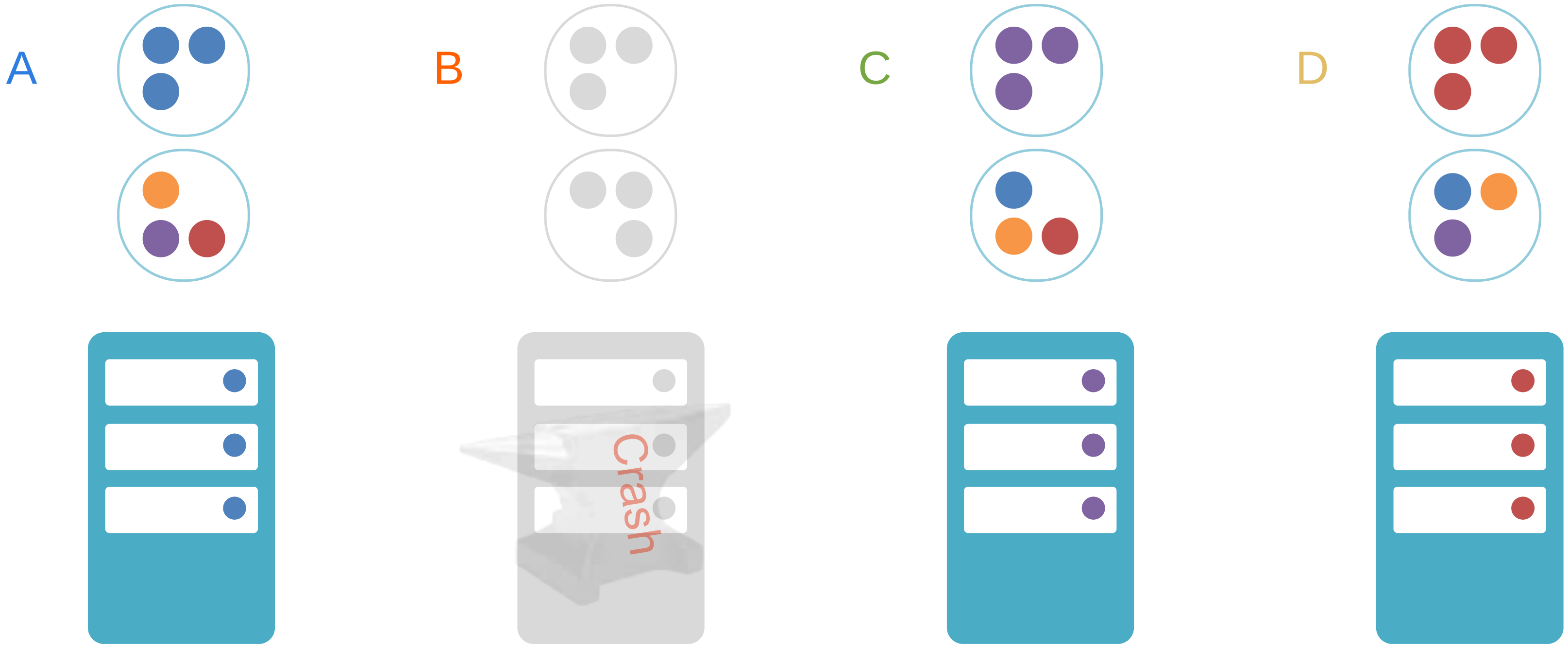
Node Crashes



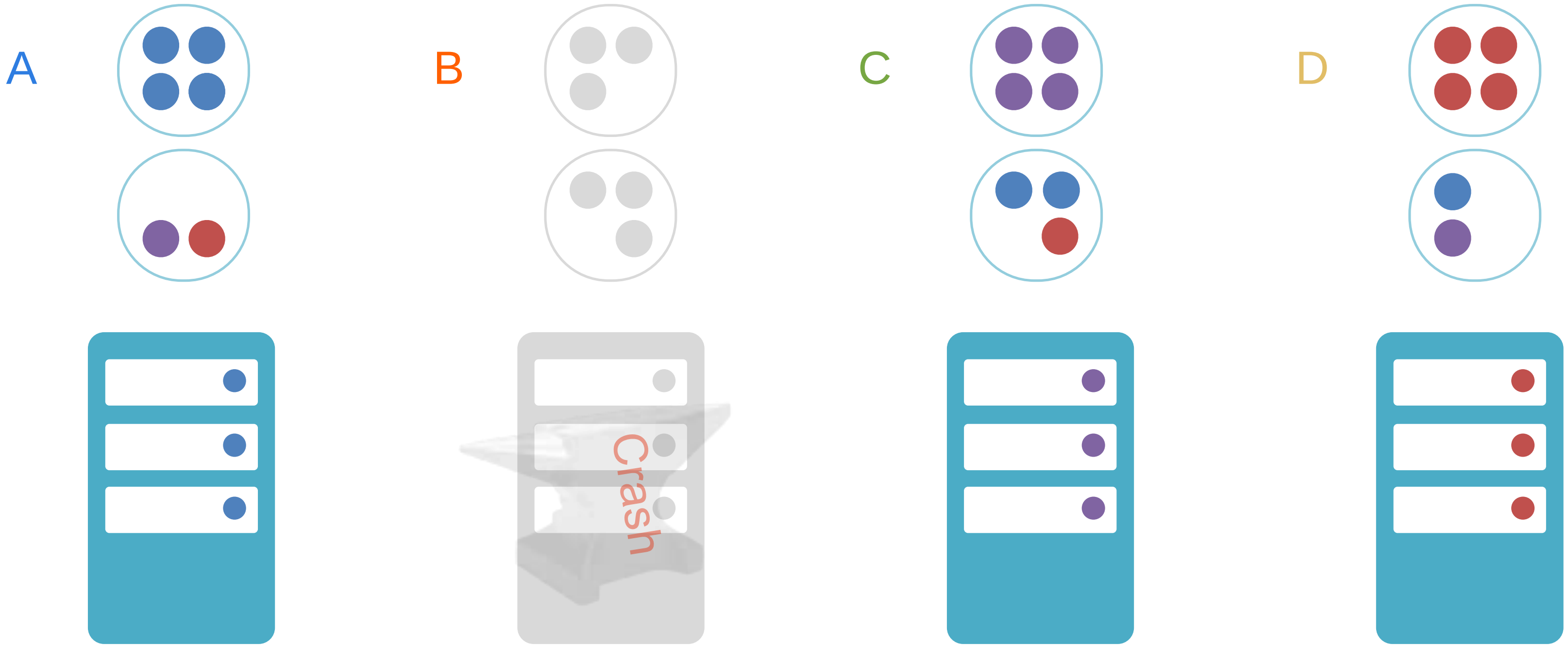
Backups Are Restored



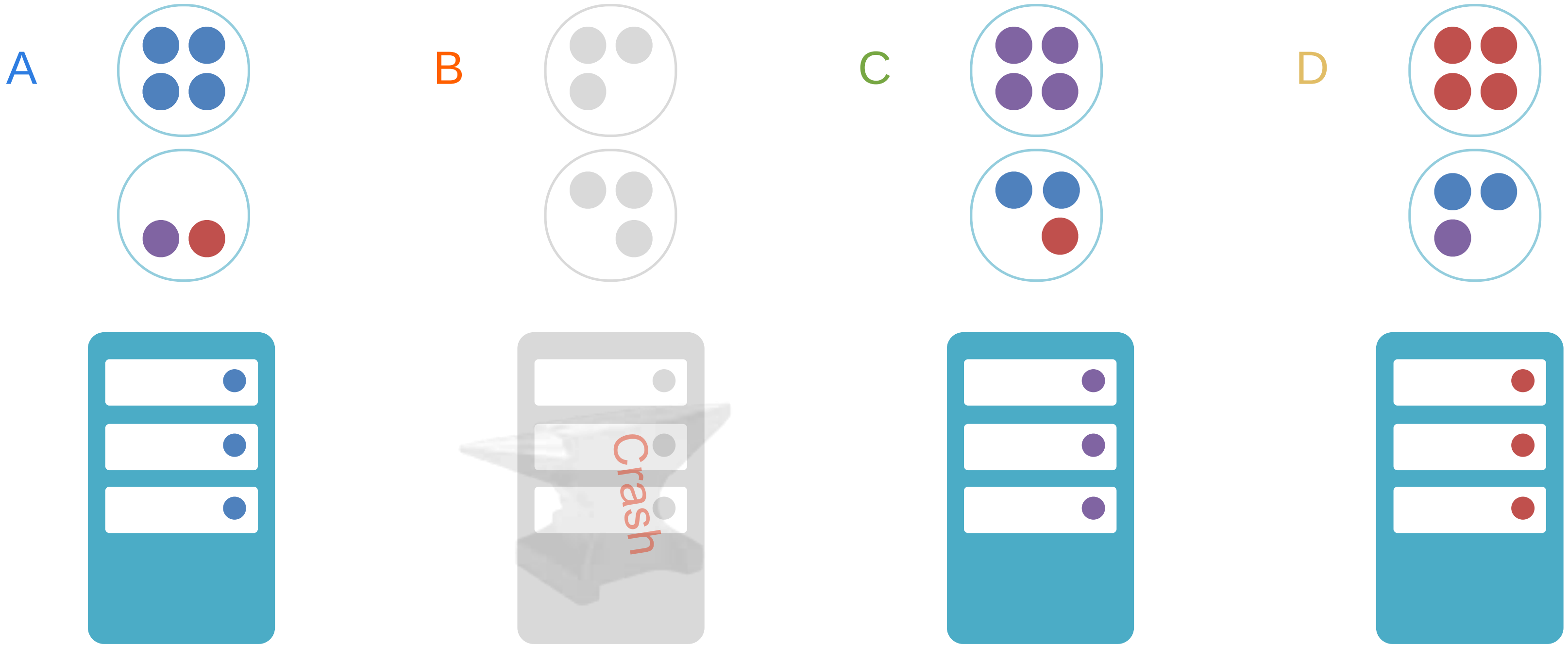
Backups Are Restored



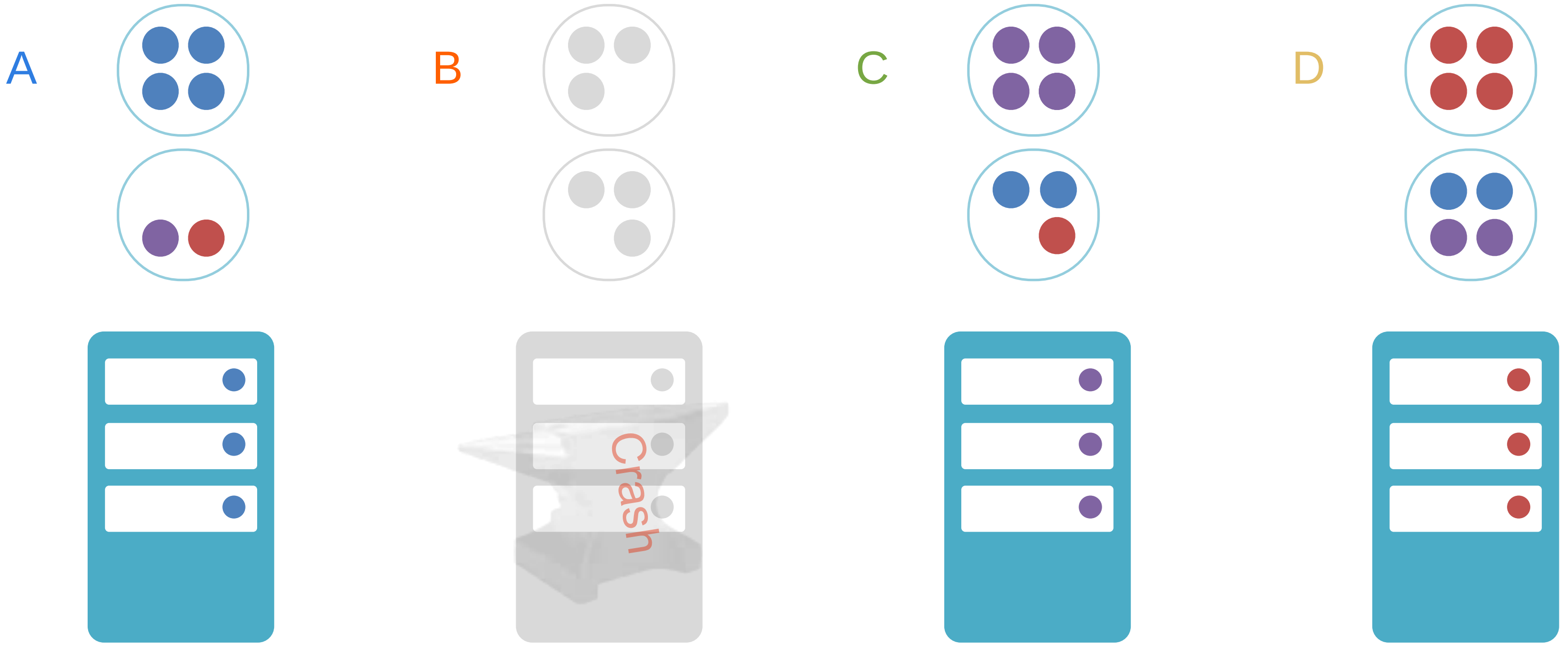
Backups Are Restored



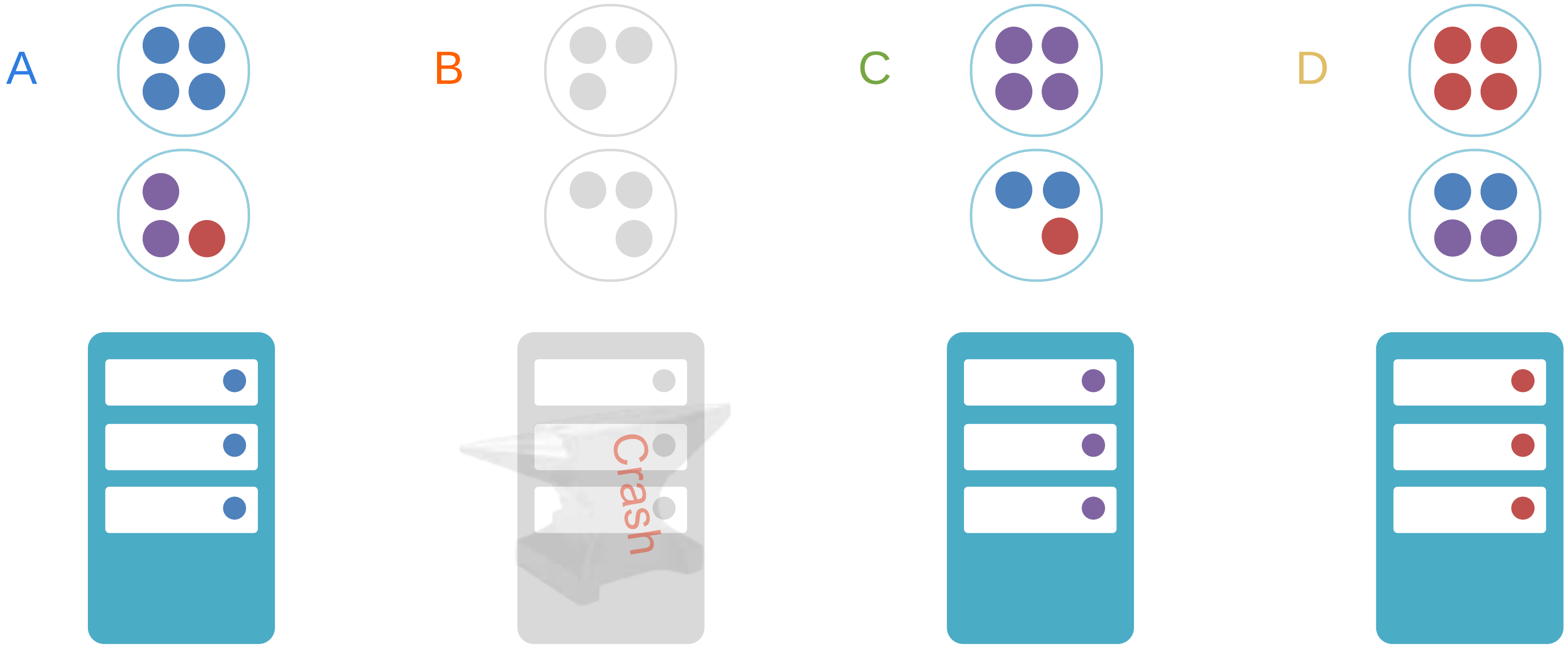
Backups Are Restored



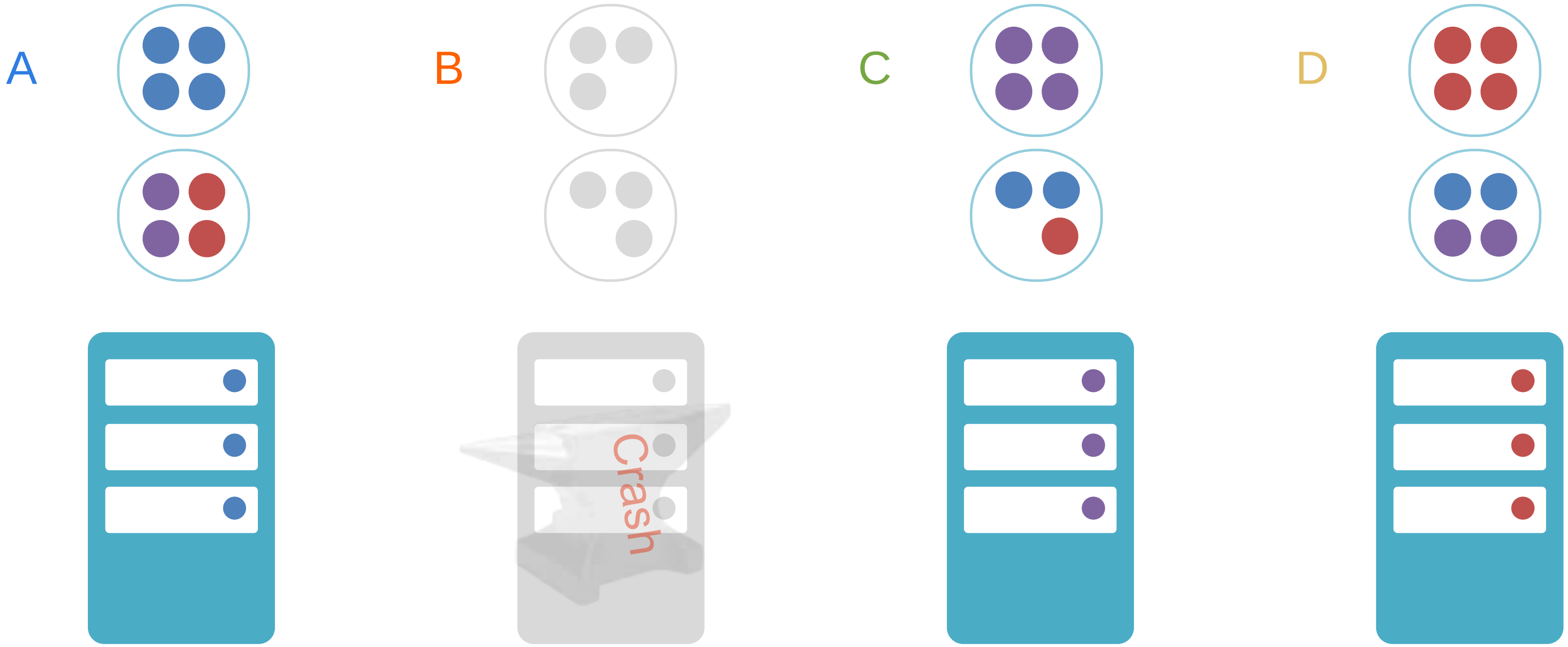
Backups Are Restored



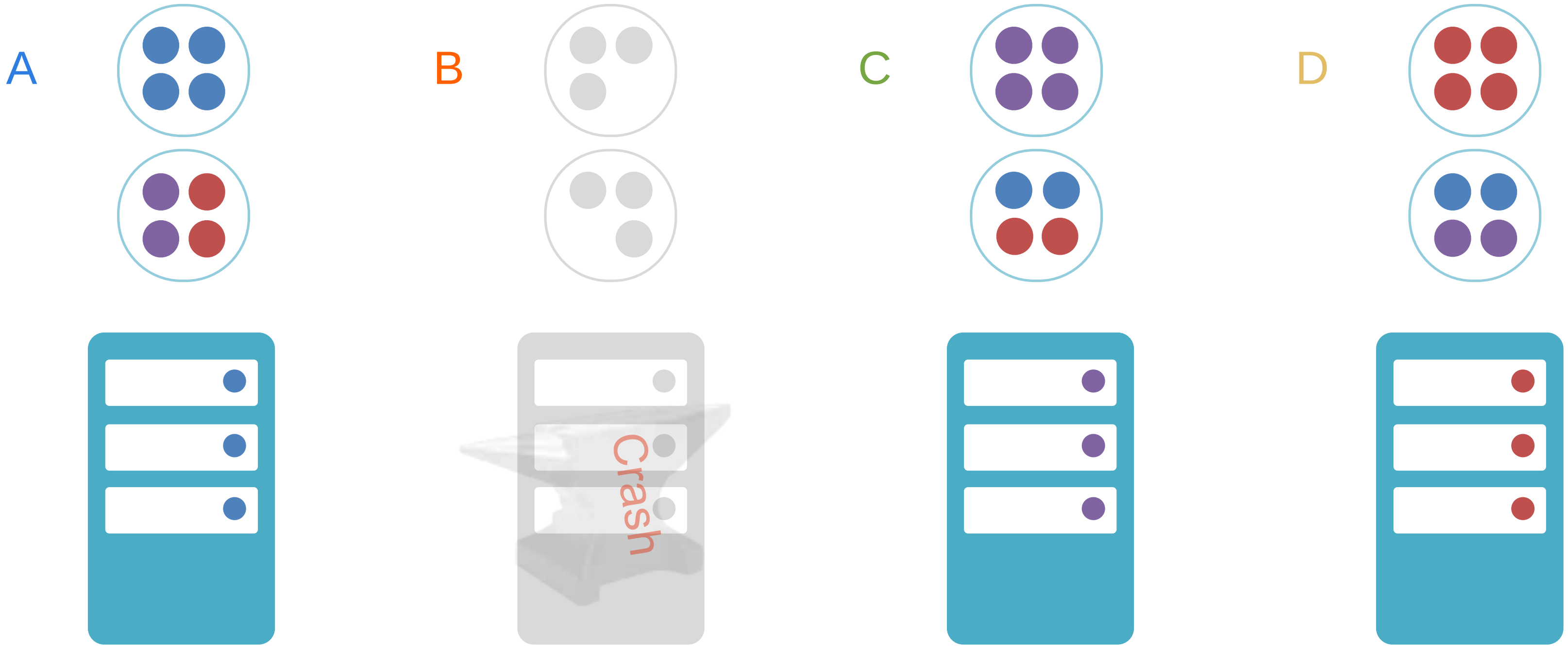
Backups Are Restored



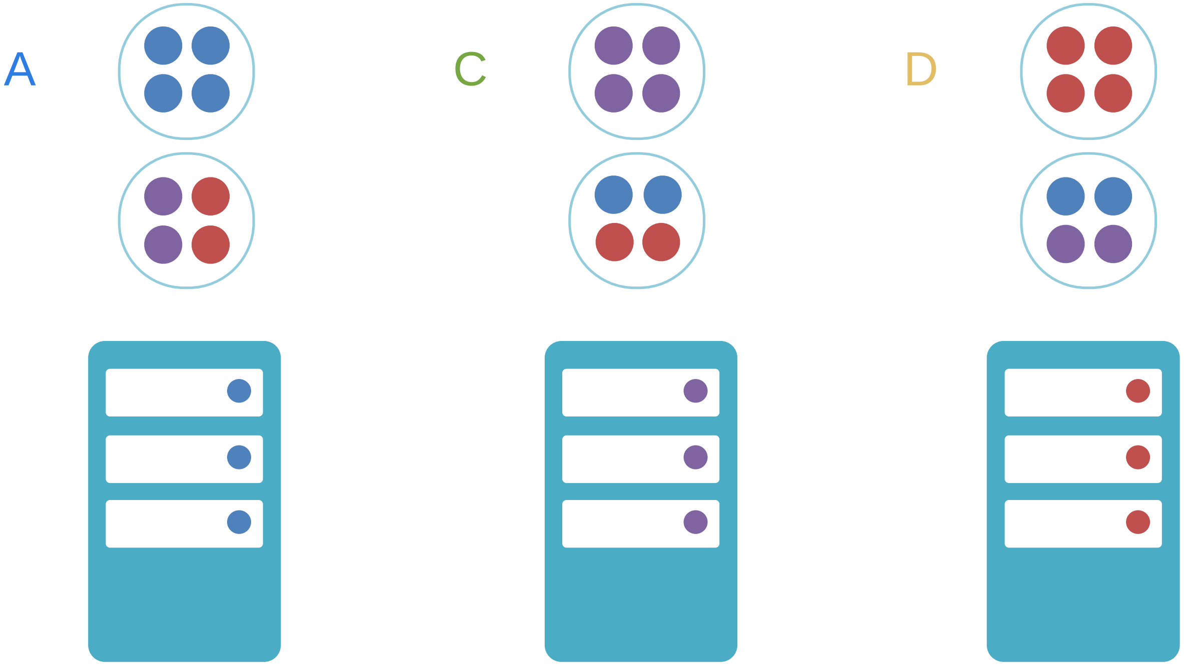
Backups Are Restored



Backups Are Restored



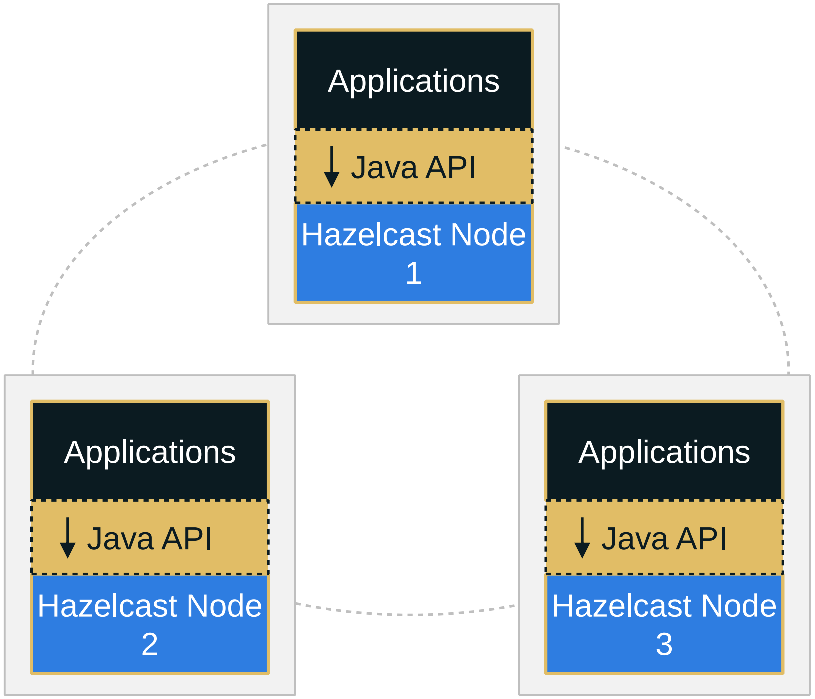
Recovery Is Complete



Deployment Strategies

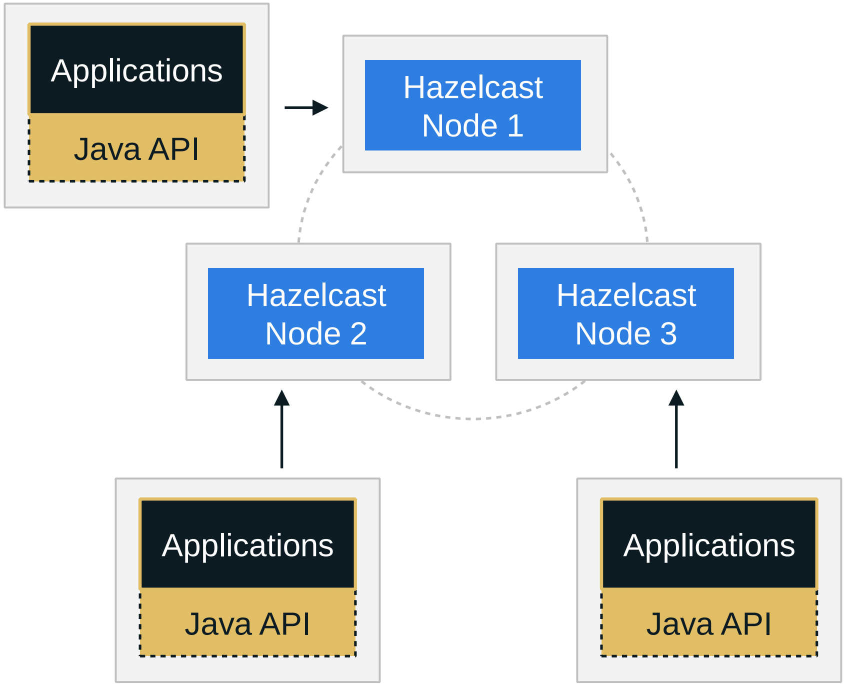
Deployment Options

Embedded Hazelcast



Great for early stages of rapid application development and iteration

Client-Server Mode



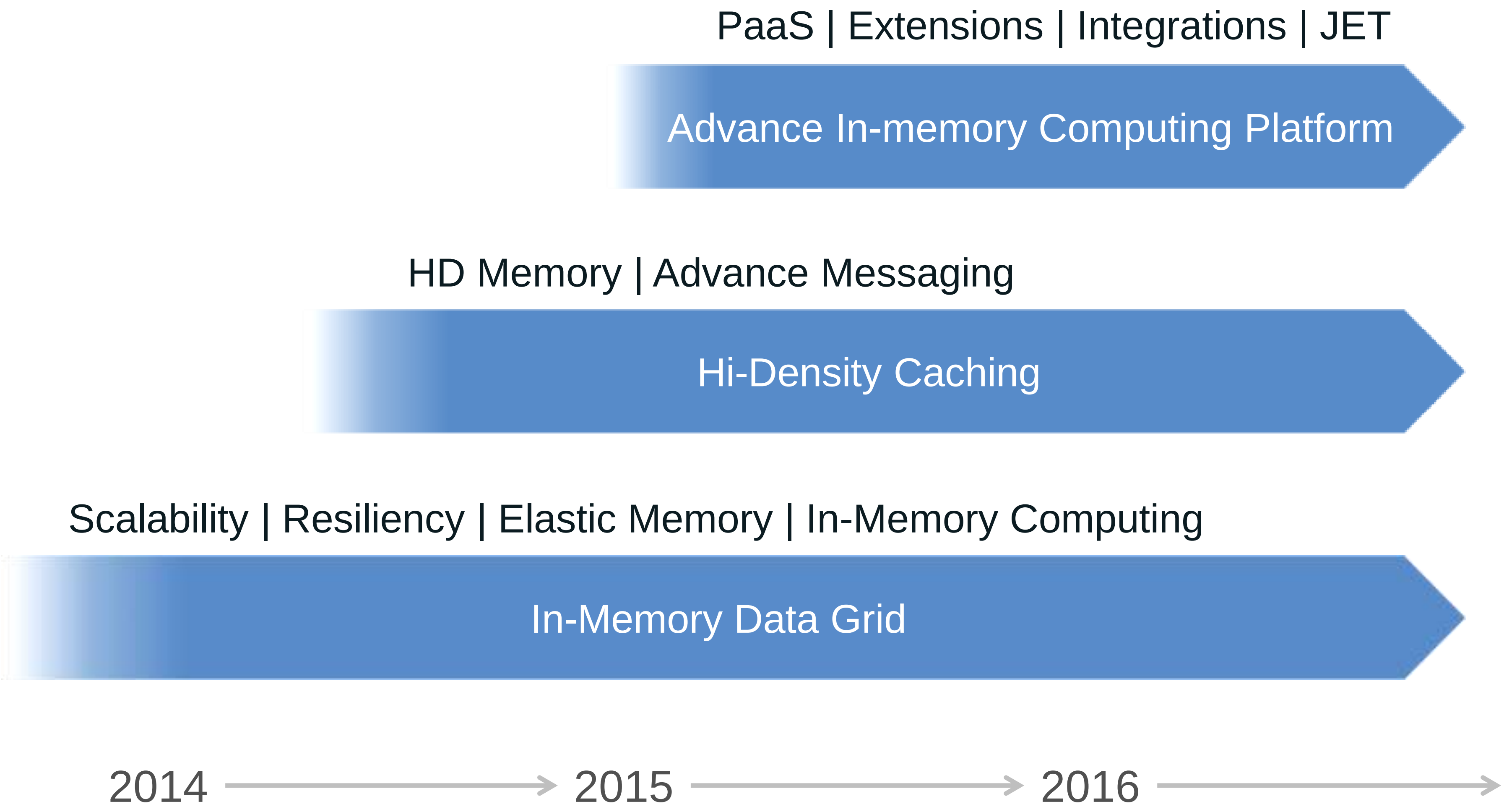
Necessary for scale up or scale out deployments – decouples upgrading of clients and cluster for long term TCO

Easy API

```
// Creating a new Hazelcast node
HazelcastInstance hz = Hazelcast.newHazelcastInstance();


// Getting a Map, Queue, Topic, ...
Map map = hz.getMap("my-map");
Queue queue = hz.getQueue("my-queue");
ITopic topic = hz.getTopic("my-topic");

//Creating a Hazelcast Client
HazelcastInstance client = HazelcastClient.newHazelcastClient();

// Shutting down the node
hz.shutdown();
```

Roadmap and Latest

Hazelcast High Level Roadmap



PaaS | Extensions | Integrations | JET

Advance In-memory Computing Platform

HD Memory | Advance Messaging

Hi-Density Caching

Scalability | Resiliency | Elastic Memory | In-Memory Computing

In-Memory Data Grid

2014 → 2015 → 2016 →

Hazelcast 3.7 Release

New Hazelcast 3.7 Features

Modularity	In 3.7, Hazelcast is converted to a modular system based around extension points. So clients, Cloud Discovery providers and integrations to third party systems like Hibernate etc will be released independently. 3.7 will then ship with the latest stable versions of each.
Redesign of Partition Migration	More robust partition migration to round out some edge cases.
Graceful Shutdown Improvements	More robust shutdown with partition migration on shutdown of a member
Higher Networking Performance	A further 30% improvement in performance across the cluster by eliminating notifyAll() calls.
Map.putAll() Performance Speedup	Implement member batching.
Rule Based Query Optimizer	Make queries significantly faster by using static transformations of queries.
Azul Certification	Run Hazelcast on Azul Zing for Java 6, 7 or 8 for less variation of latencies due to GC.
Solaris Sparc Support	Align HD Memory backed data structure's layouts so that platforms, such as SPARC work. Verify SPARC using our lab machine.
New Features for JCache	Simple creation similar to other Hazelcast Data Structures. E.g.
Command Line Interface	New command line interface for common operations performed by Operations.
Non-blocking Vert.x integration	New async methods in Map and integration with Vert.x to use them.

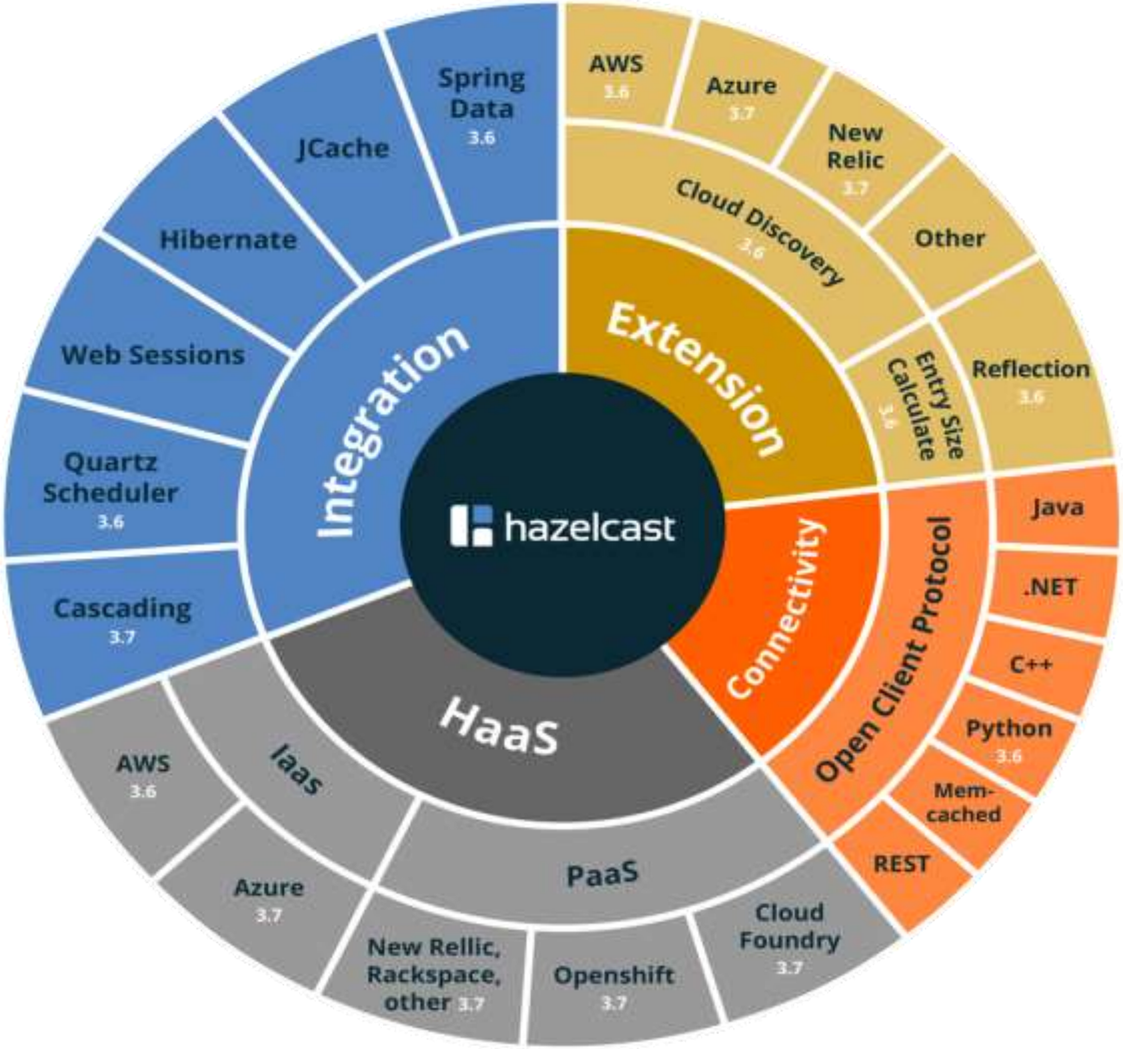
New Hazelcast 3.7 Clients and Languages

	Scala integration for Hazelcast members and Hazelcast client. Implements all Hazelcast features. Wraps the Java client for client mode and in embedded mode uses the Hazelcast member directly.
Node.js	Native client implementation using the Hazelcast Open Client protocol. Basic feature support.
Python	Native client implementation using the Hazelcast Open Client protocol. Supports most Hazelcast features.
Clojure	Clojure integration for Hazelcast members and Hazelcast client. Implements some Hazelcast features. Wraps the Java client for client mode and in embedded mode uses the Hazelcast member directly.

New Hazelcast 3.7 Cloud Features

Azure Marketplace	Ability to start Hazelcast instances on Docker environments easily. Provides Hazelcast, Hazelcast Enterprise and Management Center.
Azure Cloud Provider	Discover Provider for member discovery using Kubernetes. (Plugin)
AWS Marketplace	Deploy Hazelcast, Hazelcast Management Center and Hazelcast Enterprise clusters straight from the Marketplace.
Consul Cloud Provider	Discover Provider for member discovery for Consul (Plugin)
Etcd Cloud Provider	Discover Provider for member discovery for Etcd (Plugin)
Zookeeper Cloud Provider	Discover Provider for member discovery for Zookeeper (Plugin)
Eureka Cloud Provider	Discover Provider for member discovery for Eureka 1 from Netflix. (Plugin)
Docker Enhancements	Docker support for cloud provider plugins

Hazelcast Platform: Hazelcast Everywhere



Hazelcast on Cloud

Hazelcast on Cloud – IaaS, PaaS

Features	Description
Amazon EC2	EC2 Auto discovery – upgraded with Discovery SPI
Microsoft Azure	Available on Azure Marketplace
Pivotal Cloud Foundry	Only distributed IMDG to provide on-demand service broker and disk based persistence
OpenShift	Native compliancy

Hazelcast on Cloud – SaaS, IaaS, PaaS

Other off-the-shelf cloud based compliancy

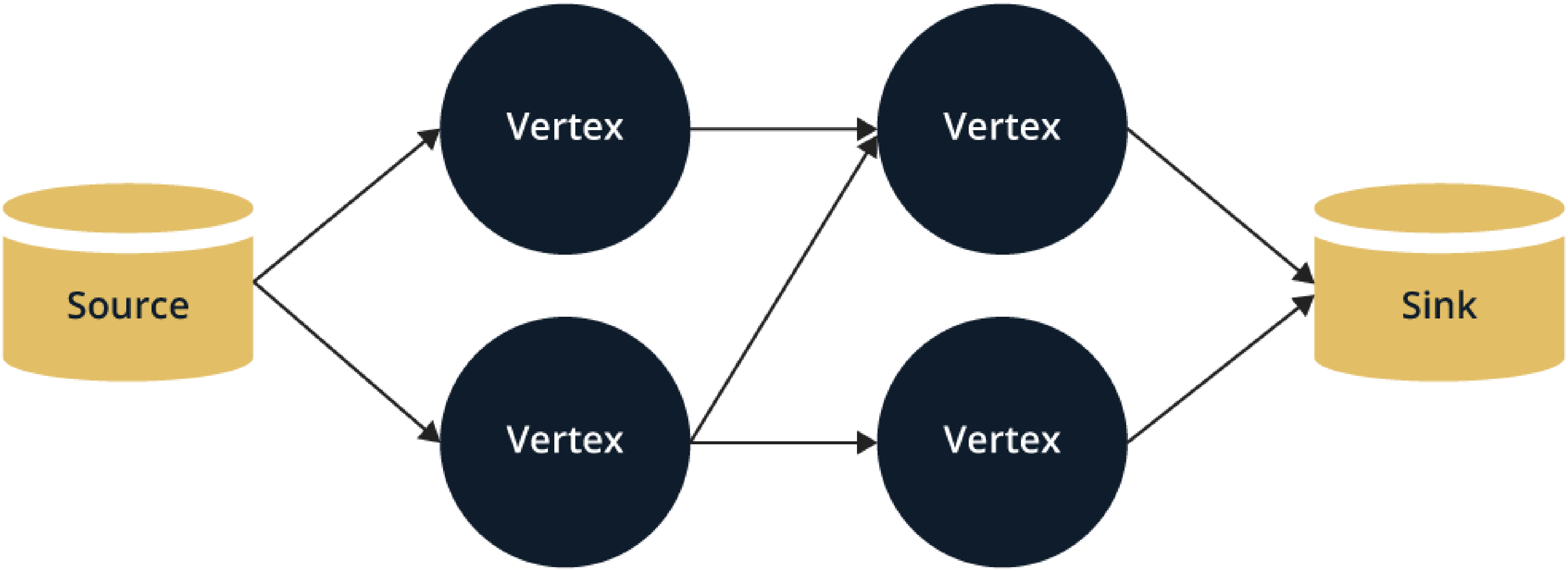
- OpenStack
- Google Compute Engine
- Google Platform Services
- jClouds
- Discovery SPI – Everything Everywhere

hazelcast JET

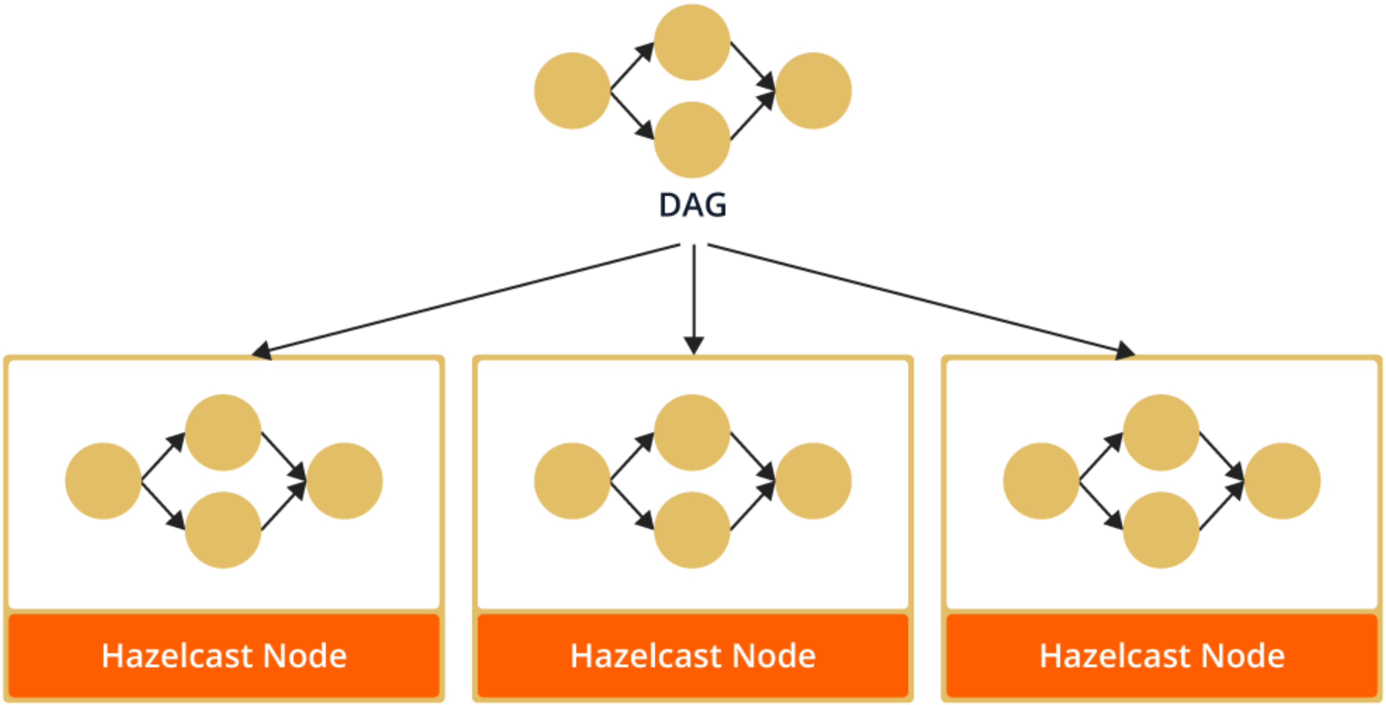
What’s Hazelcast Jet?

- General purpose distributed data processing framework
- Based on Direct Acyclic Graph to model data flow
- Built on top of Hazelcast
- Comparable to Apache Spark or Apache Flink

DAG



Job Execution



Hazelcast Services

Service Offerings

Hazelcast (Apache Licensed)

- Professional Subscription – 24x7 support*

Hazelcast Enterprise Support

- Available with Hazelcast Enterprise software subscription - 24x7 support*

Additional Services

- Development Support Subscription – 8x5 support*
- Simulator TCK
- Training
- Expert Consulting
- Development Partner Program

* All subscriptions include Management Center

Support Subscriptions

What's Included

100% SUCCESS RATE ON CUSTOMER ISSUES:

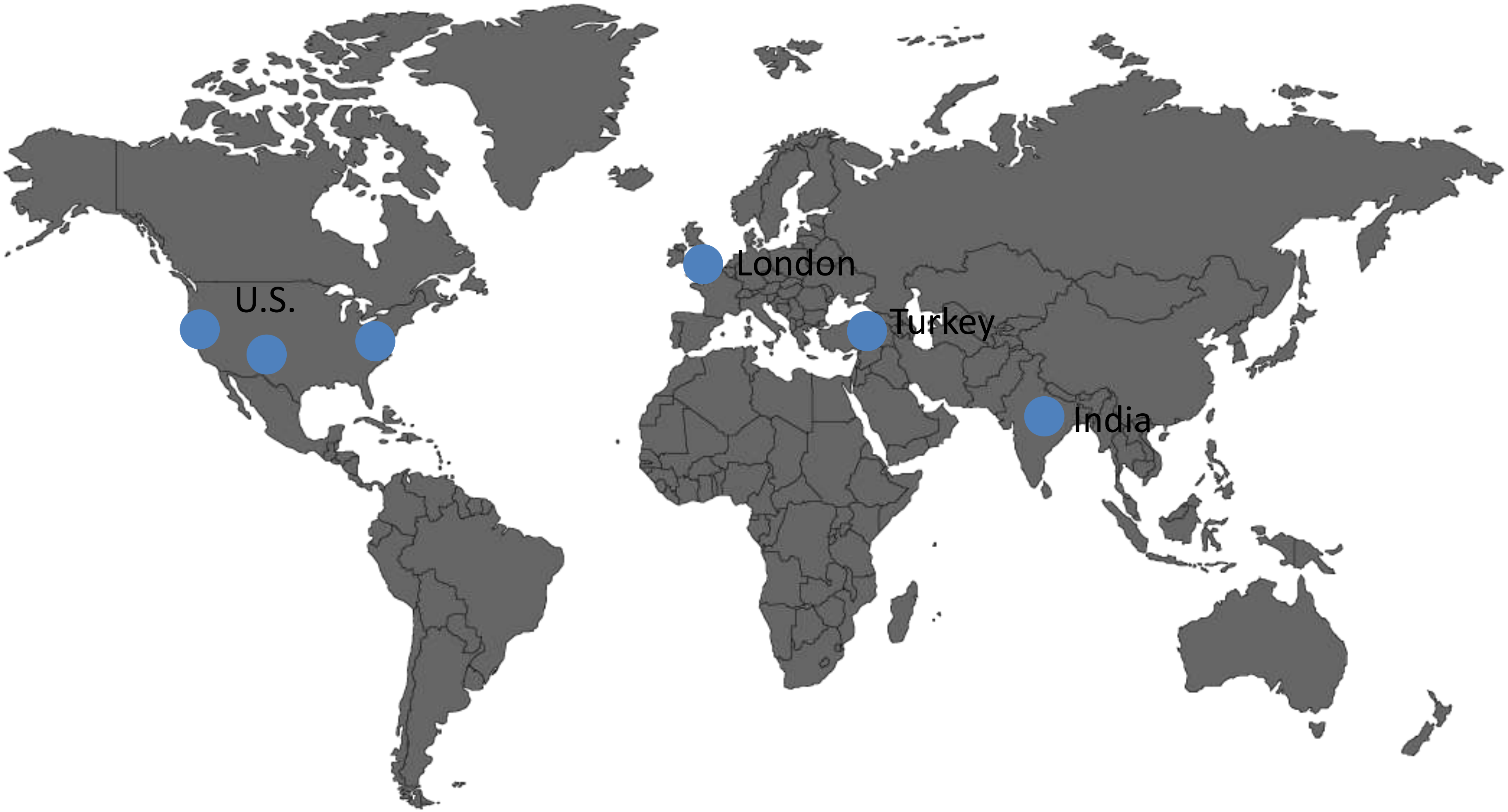
> ***"As usual, the response was timely beyond expectations, and very good technical content returned. Exemplary support, hard to find in any company…"***
>
> - Fortune 100 Financial Services Customer

	ENTERPRISE HD	ENTERPRISE	PROFESSIONAL	OPEN SOURCE
SUPPORT WINDOW	24/7	24/7	24/7	
RESPONSE TIME FOR CRITIAL ISSUES	1 Hour	1 Hour	2 Hours	
SUPPORTED SOFTWARE	Hazelcast & Hazelcast Enterprise	Hazelcast & Hazelcast Enterprise	Hazelcast	
SUPPORT CONTACTS	4	4	2	
SUPPORT CHANNELS	Email, IM & Phone	Email, IM & Phone	Email, IM & Phone	
PATCH LEVEL FIXES	▯	▯	▯	
REMOTE MEETINGS (via GoToMeeting)	▯	▯	▯	
CODE REVIEW (with a Senior Solutions Architect)	2 Hours	2 Hours	2 Hours	
QUARTERLY REVIEW OF FEATURE REQUES*	▯	▯		
QUARTERLY REVIEW OF HAZELCAST ROADMAP*	▯	▯		

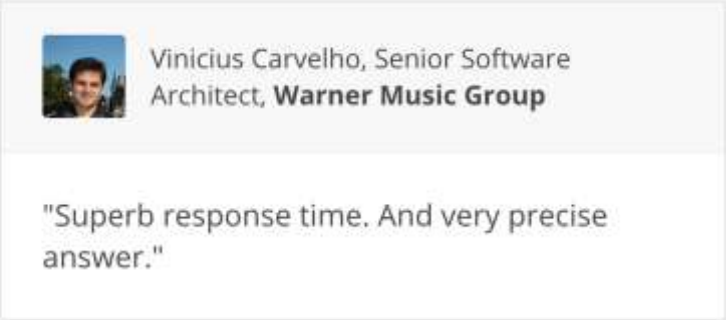
Best In Class Support

- Support from the Engineers who wrote the code
- SLA Driven – 100% attainment of support response time
- Follow the Sun
- Portal, Email and Phone access
- Go Red, Go Green. Reproduction of issues on Simulator. Proof of fix on Simulator.
- Periodic Technical Reviews
- Meet your production schedule and corporate compliance requirements
- Ensure the success of your development team with training and best practices

Hazelcast Support Coverage

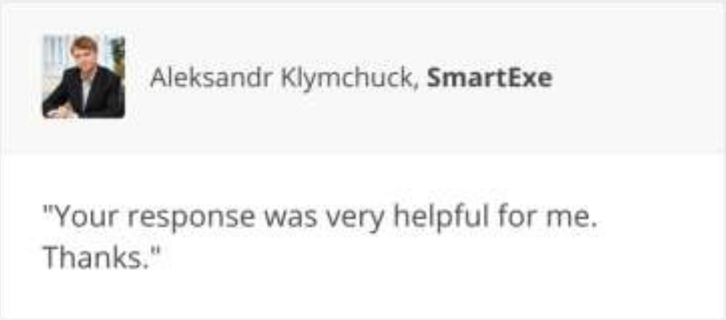


Support Testimonials



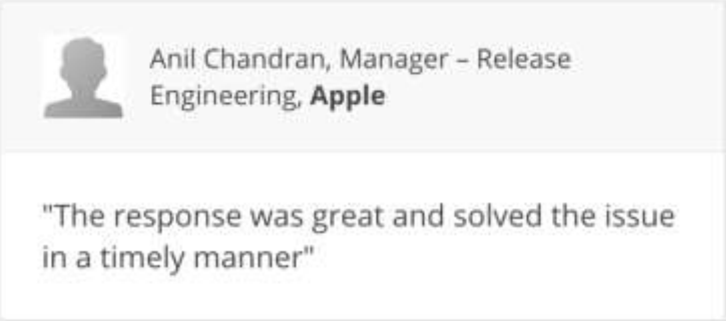
Vinicius Carvelho, Senior Software Architect, **Warner Music Group**

"Superb response time. And very precise answer."



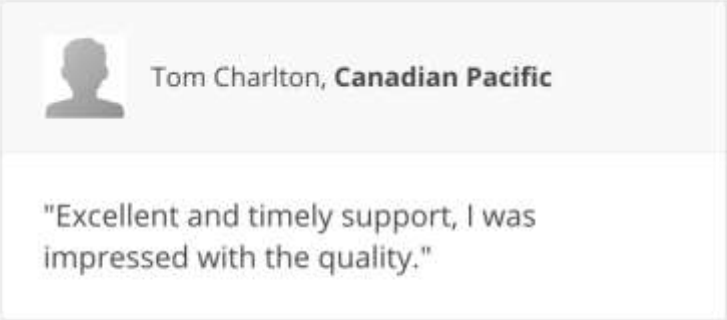
Aleksandr Klymchuck, **SmartExe**

"Your response was very helpful for me. Thanks."



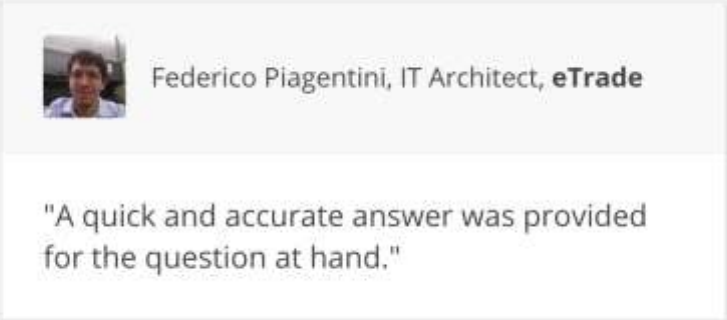
Anil Chandran, Manager – Release Engineering, **Apple**

"The response was great and solved the issue in a timely manner"



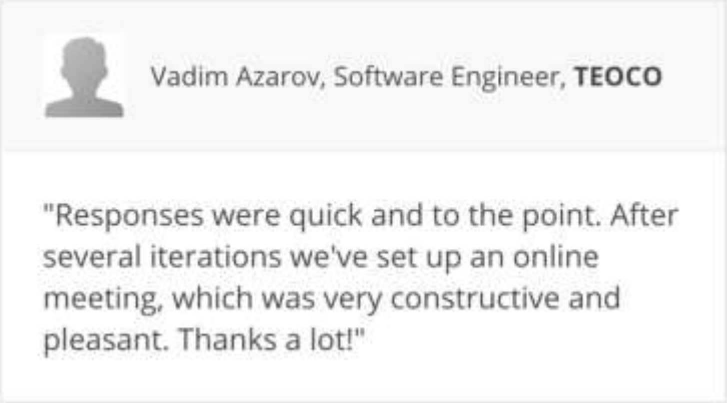
Tom Charlton, **Canadian Pacific**

"Excellent and timely support, I was impressed with the quality."



Federico Piagentini, IT Architect, **eTrade**

"A quick and accurate answer was provided for the question at hand."



Vadim Azarov, Software Engineer, **TEOCO**

"Responses were quick and to the point. After several iterations we've set up an online meeting, which was very constructive and pleasant. Thanks a lot!"

Release Lifecycle

- **Regular Feature release** each 4-5 months, e.g. 3.3, 3.4, 3.5
- **Maintenance release** approximately each month with bug fixes based on the current feature release, e.g. 3.4.1
- **For older versions**, patch releases made available to fix issues
- **Release End of Life** per support contract

Thank you