Hazelcast

Abhinav Prashant
Shweta Amla
Yagna Namburi
Venkata Rama Krishn Pandi
Ryan Wolf

Contents

Introduction

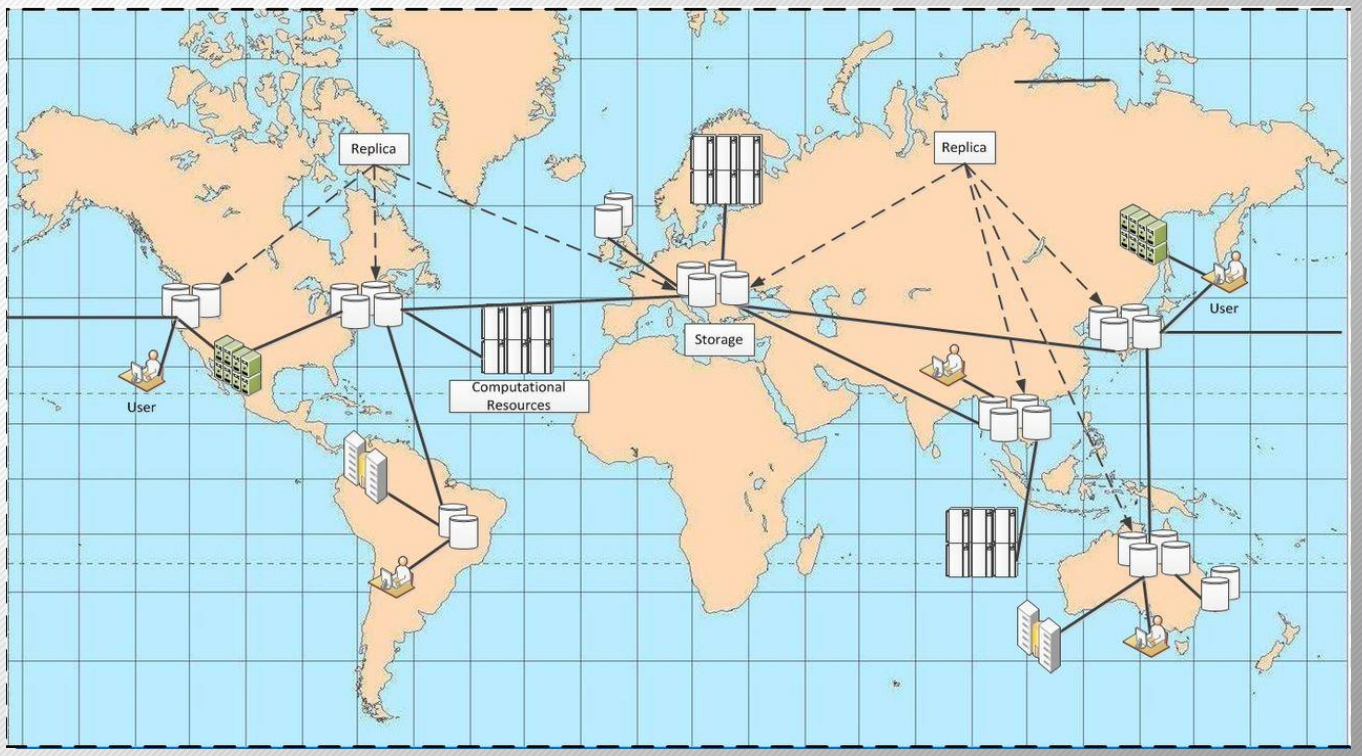
Hazelcast is a leading
Open Source In-Memory Data Grid

Written in JAVA

Startup founded in 2008
by- Talip Ozturk, Fuad Malikov

Data Grid

- Ability to access, modify and transfer humongous data
- Security concerns
- Data access using middleware applications



CAP Theorem

- Also called Brewer's theorem
- Distributed systems simultaneously cannot provide:
  - Consistency
  - Availability
  - Partition tolerance
- Hazelcast chooses availability over consistency

In-Memory Database

- Lies in main-memory for data storage
- Faster as compared to disk-optimized database
- Improved response time - critical applications

Hazelcast - In-memory data Grid

- Data stored in RAM of Servers
- Redundancy - multiple copies of data exists on different servers
- Scalability - servers can be dynamically added or removed
- Peer to peer communication – No Master Slave
- Data persistence done using Relational or other NoSQL database

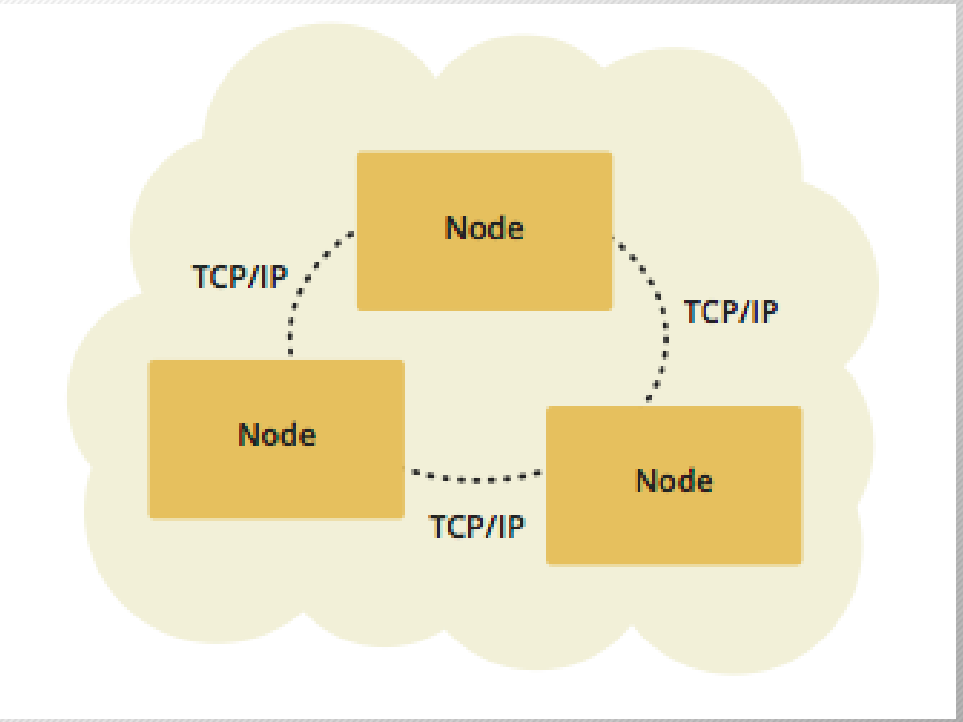
Hazelcast Topology

Hazelcast can be deployed in Two ways:

- Embedded
- Client-Server

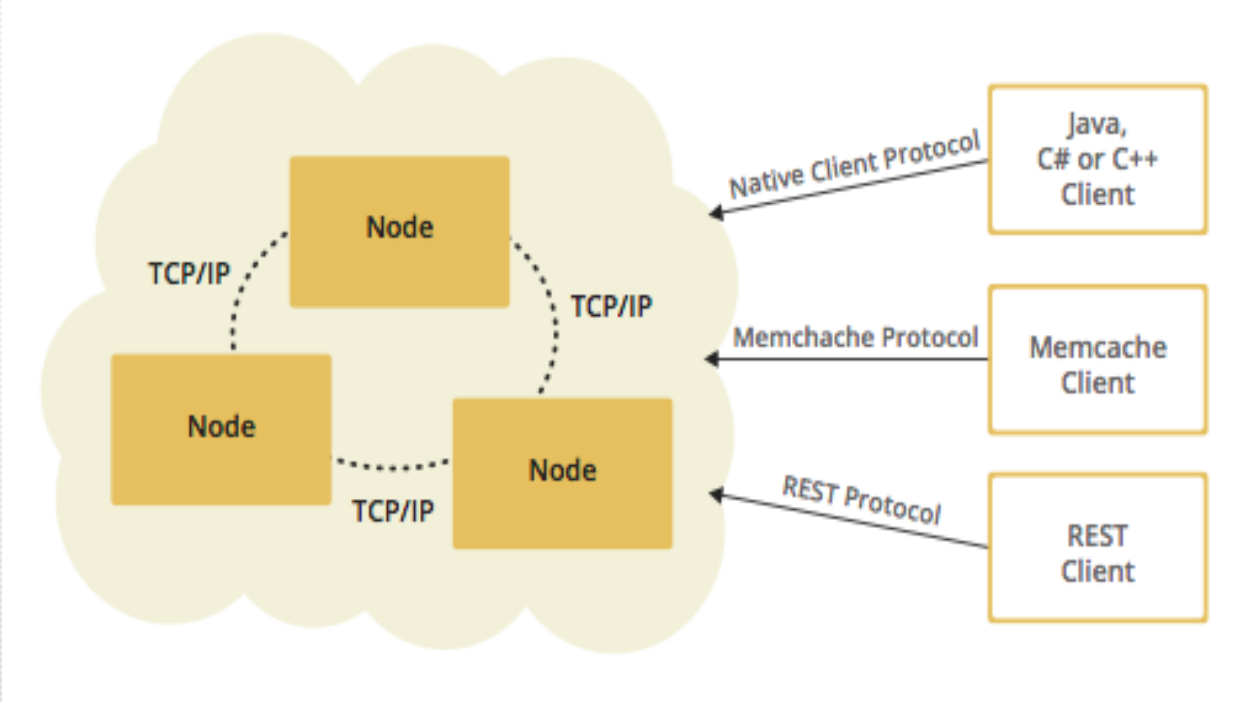
Embedded Deployment

- Applications requiring
  - High performance
  - Lots of task execution
- Members include
  - Application
  - Hazelcast data and Services
- Low-latency data access



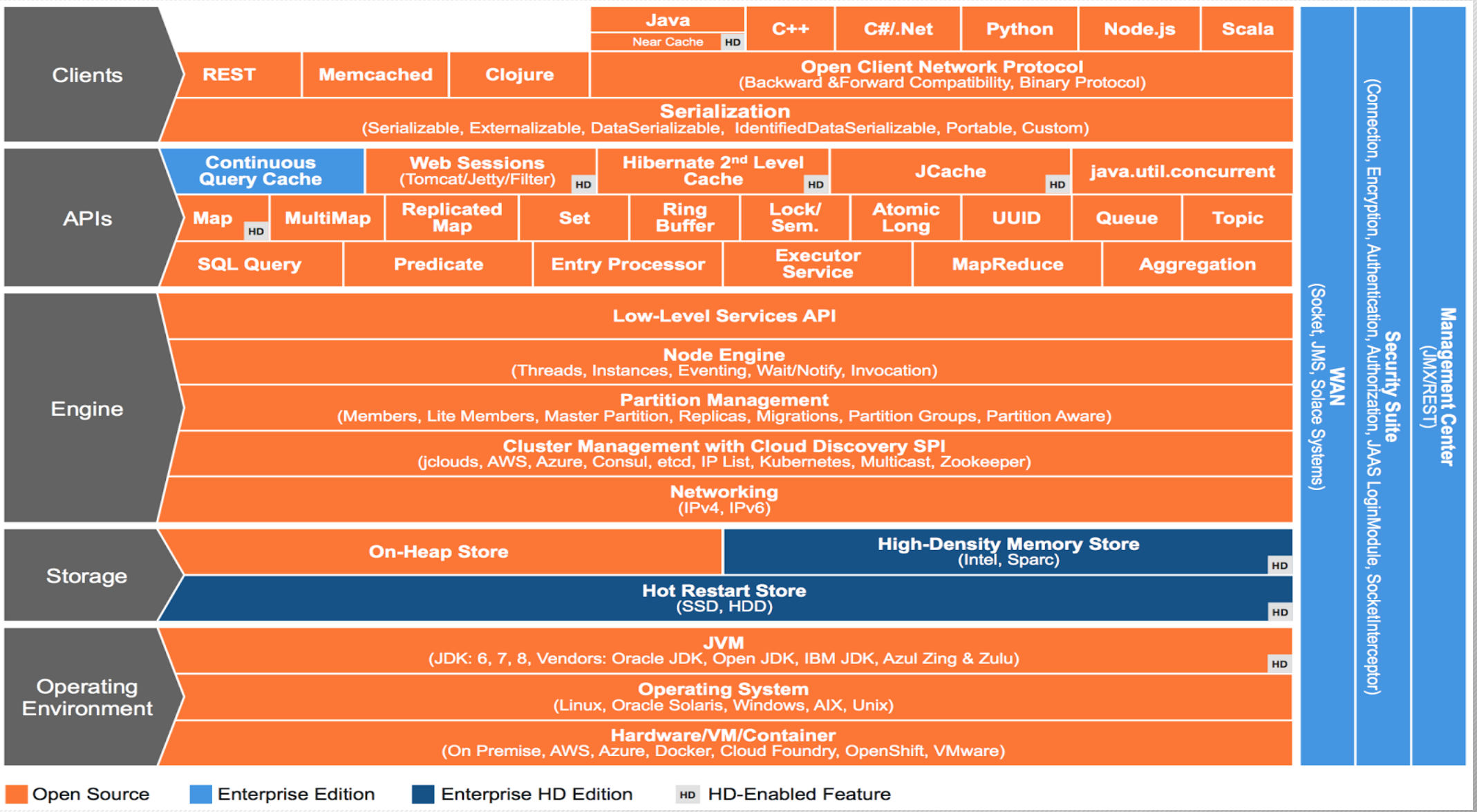
Client-Server Deployment

- Centralized - accessed by applications through clients
- Server members -independently created and scaled
- Predictable and reliable Hazelcast performance
- Scalability handled independently.



Contents

Architecture



Cluster Management

- Cluster auto detects new node
- Discovery mechanisms available:
  - Multicast
  - TCP
  - EC2 Cloud
- Oldest node is the cluster leader
- Oldest node listens to join requests

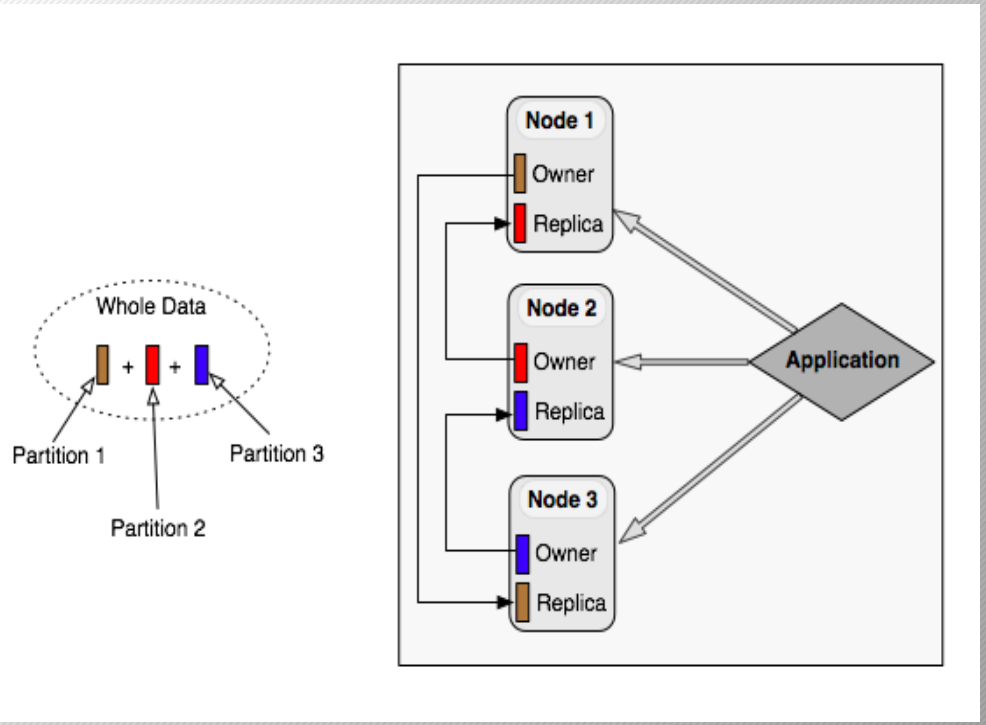
```
<network>
     <join>
         <multicast enabled="true">
             <multicast-group>224.2.2.3</multicast-group>
             <multicast-port>54327</multicast-port>
             <multicast-time-to-live>32</multicast-time-to-live>
             <multicast-timeout-seconds>2</multicast-timeout-seconds>
             <trusted-interfaces>
                <interface>192.168.1.102</interface>
             </trusted-interfaces>
         </multicast>
         <tcp-ip enabled="false">
         </tcp-ip>
         <aws enabled="false">
         </aws>
     </join>
<network>
```

Data Partitioning

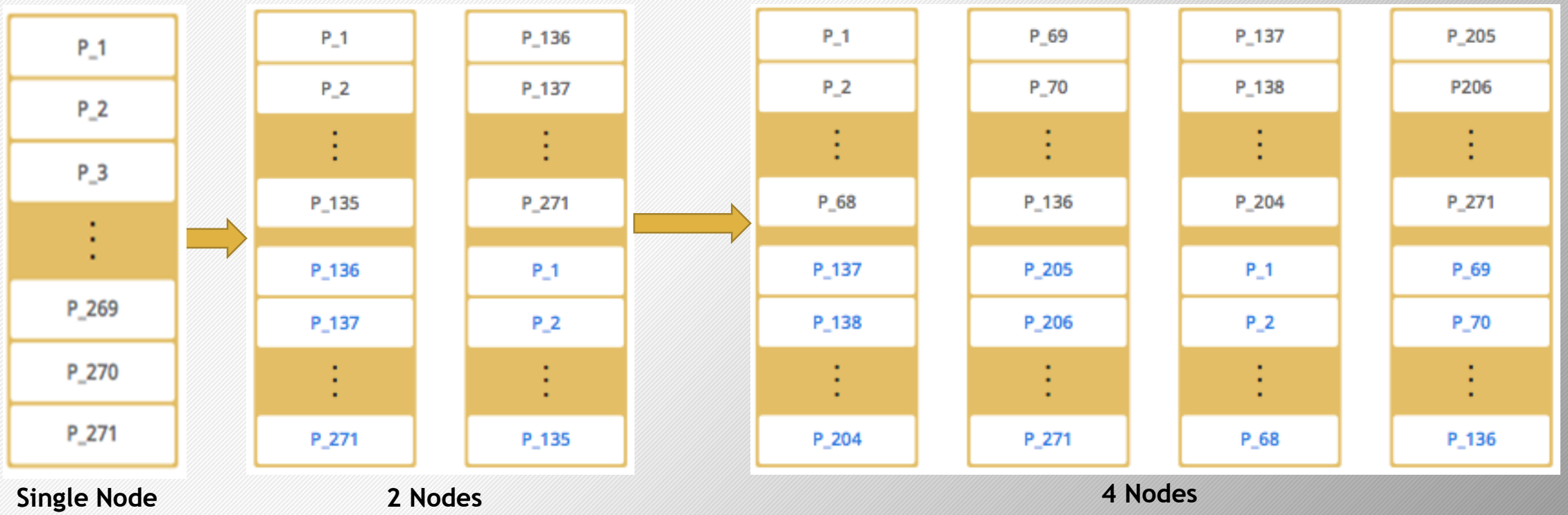
Why?

Fast data operations across cluster

- Number of partitions fixed across cluster
- Partitions are equally divided across active nodes
- Partition table is used to keep track of partitions
- Leader node periodically send updated partition table to peer nodes



Hazelcast Partition



Data Model

- Data stored as Key-Value pairs

- To store key, hashing is used to find the partition:
  - Keys are serialized
  - Byte array hashed using hashing algorithms
  - Hash result is mod by number of partitions

hash(key) mod partition count

In Memory Format

- By default, data stored in serialized format

```
<map name="binaryMap">
    <in-memory-format>BINARY</in-memory-format>
</map>
```

- Option to store data as objects in de-serialized format

```
<map name="objectMap">
    <in-memory-format>OBJECT</in-memory-format>
</map>
```

Indexing

- Applied to frequently queried fields
- Implemented using Maps of map
- Two types:
  - Ordered
  - Unordered
- Costly for write intensive application
- Backup data not indexed

Implementing Indexing

Two methods:

a. Using API

b. Using Hazelcast config file

```
IMap map = hazelcastInstance.getMap( "employees" );
// ordered, since we have ranged queries for this field
map.addIndex( "age", true );
// not ordered, because boolean field cannot have range
map.addIndex( "active", false );
```

```
<map name="default">
  ...
  <indexes>
    <index ordered="false">name</index>
    <index ordered="true">age</index>
  </indexes>
</map>
```

Eviction

- Unless explicitly deleted, entries remain in the memory
- Eviction prevents the JVM from running out of memory
- Hazelcast supports two eviction policies
  - LFU (least frequently used)
  - LRU (least recently used)

Eviction Contd.

- Eviction can be triggered based on:
  - Heap used (percentage or absolute value)
  - Maximum entry count (in an entire member or partition)
- Eviction percentage parameter - fraction of entry set to be removed
- Eviction can also be triggered based on the following parameters:
  - Time to live
  - Maximum idle time

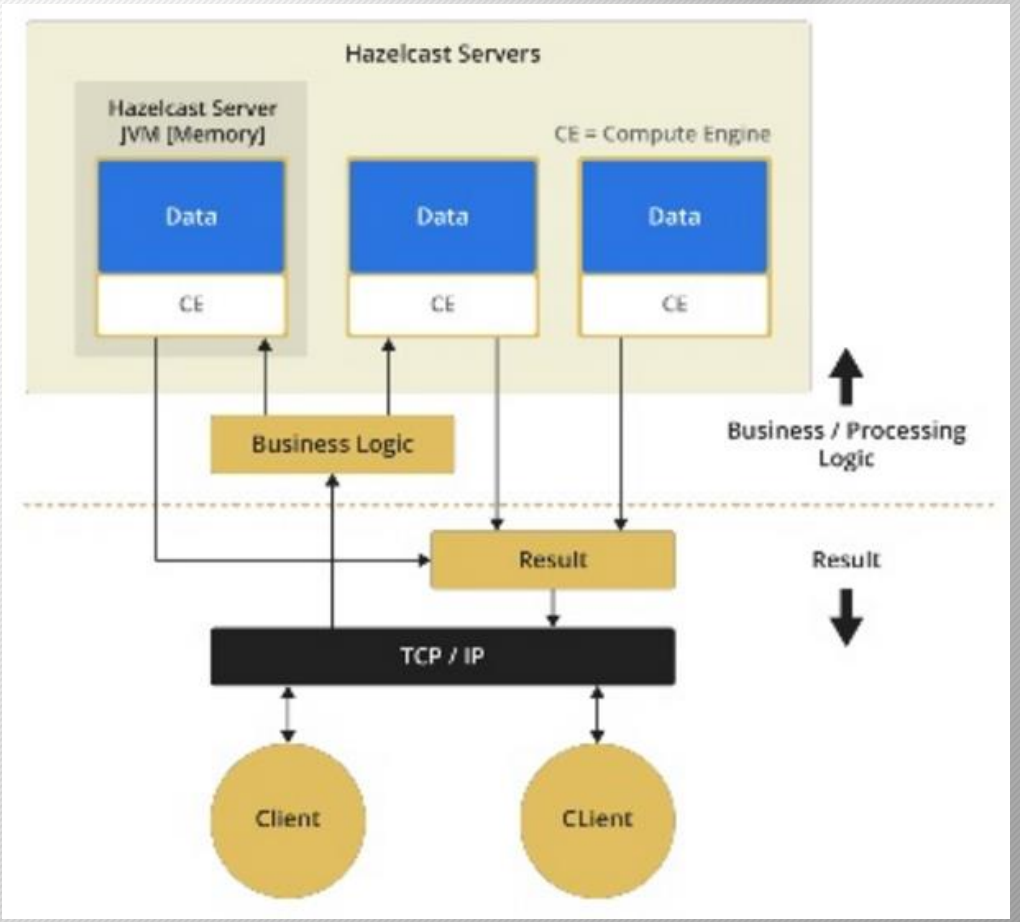
Contents

Key Features

- Distributed Computing
- High Density Memory Store
- Database Caching/Persistence
- Distributed Messaging
- Web Session Clustering
- Distributed Caching
- Distributed Data Structures

Distributed Computing

- In-memory computations over distributed datasets
- Asynchronous task execution

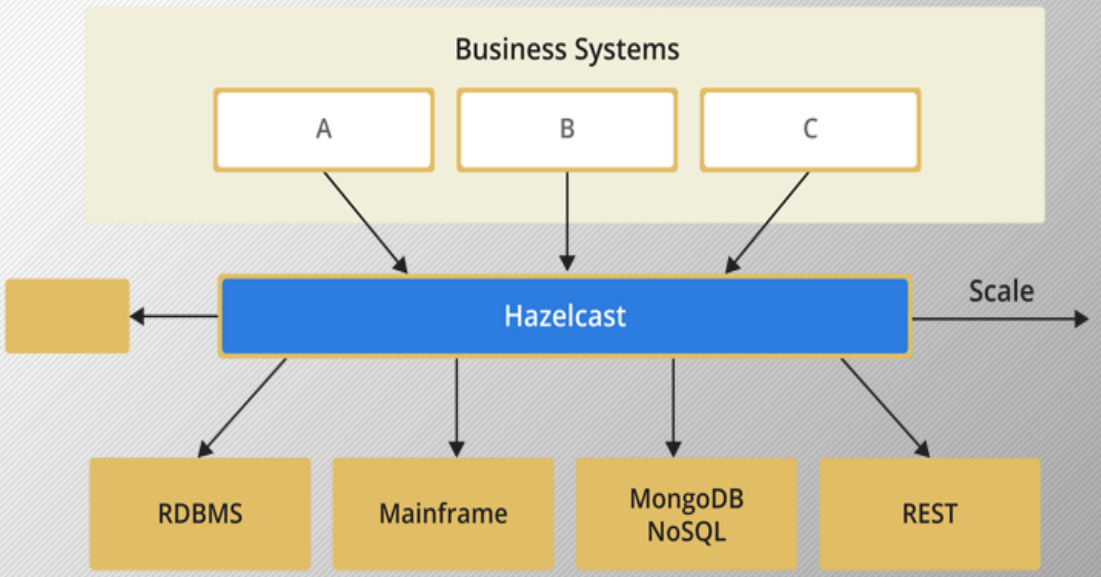


High Density Memory Store

- High Density Memory Store
- Jcache provider
- Hazelcast High Density caching

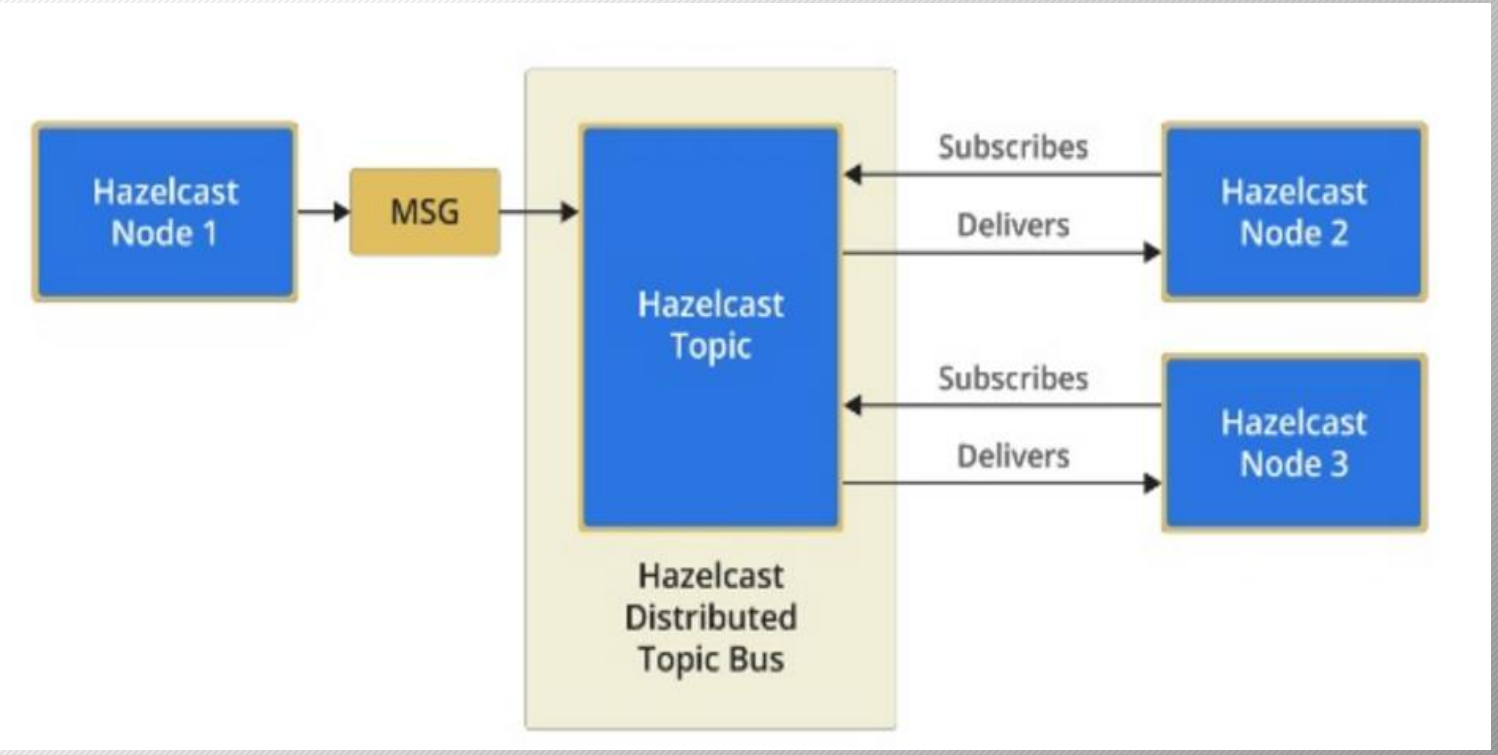
Database Caching/Persistence

- Caching solution for
  - NoSQL databases
  - RDBMS databases
- Performs hit/miss mechanism
- Improves performance



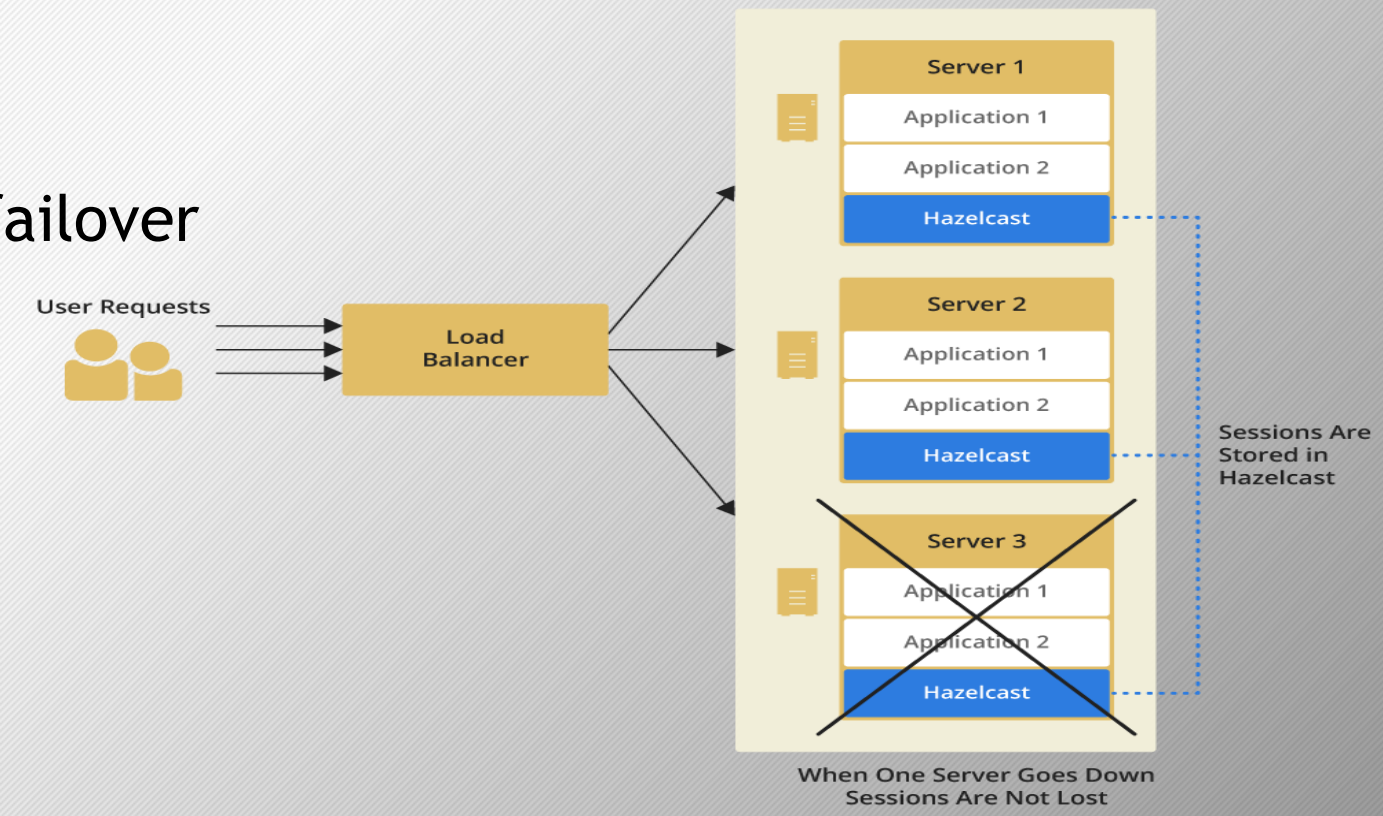
Distributed Messaging

- Publish and subscribe are cluster wide
- Ordered messaging



Web Session Clustering

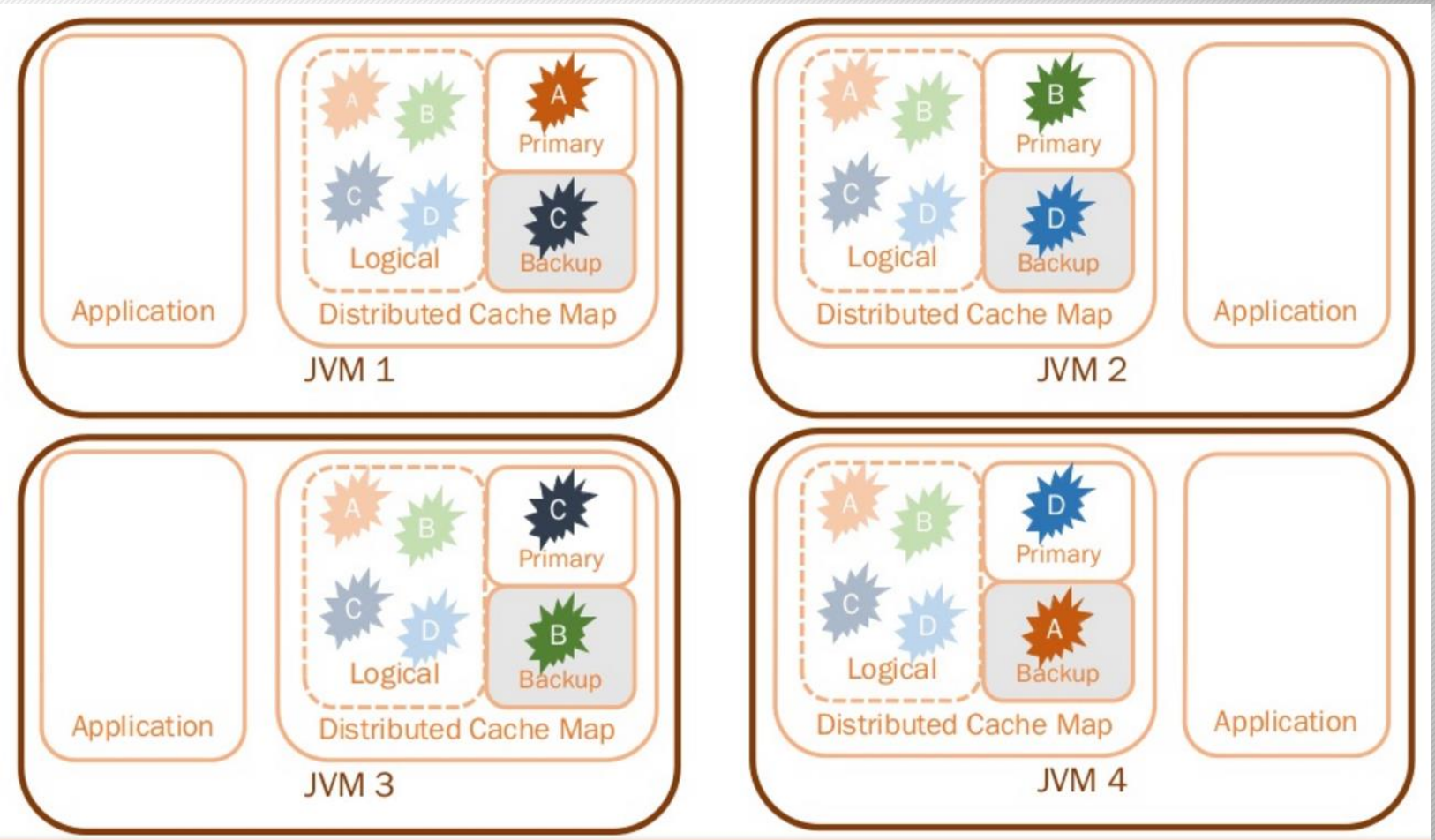
- Maintains session for user
- Balances load across sessions at failover



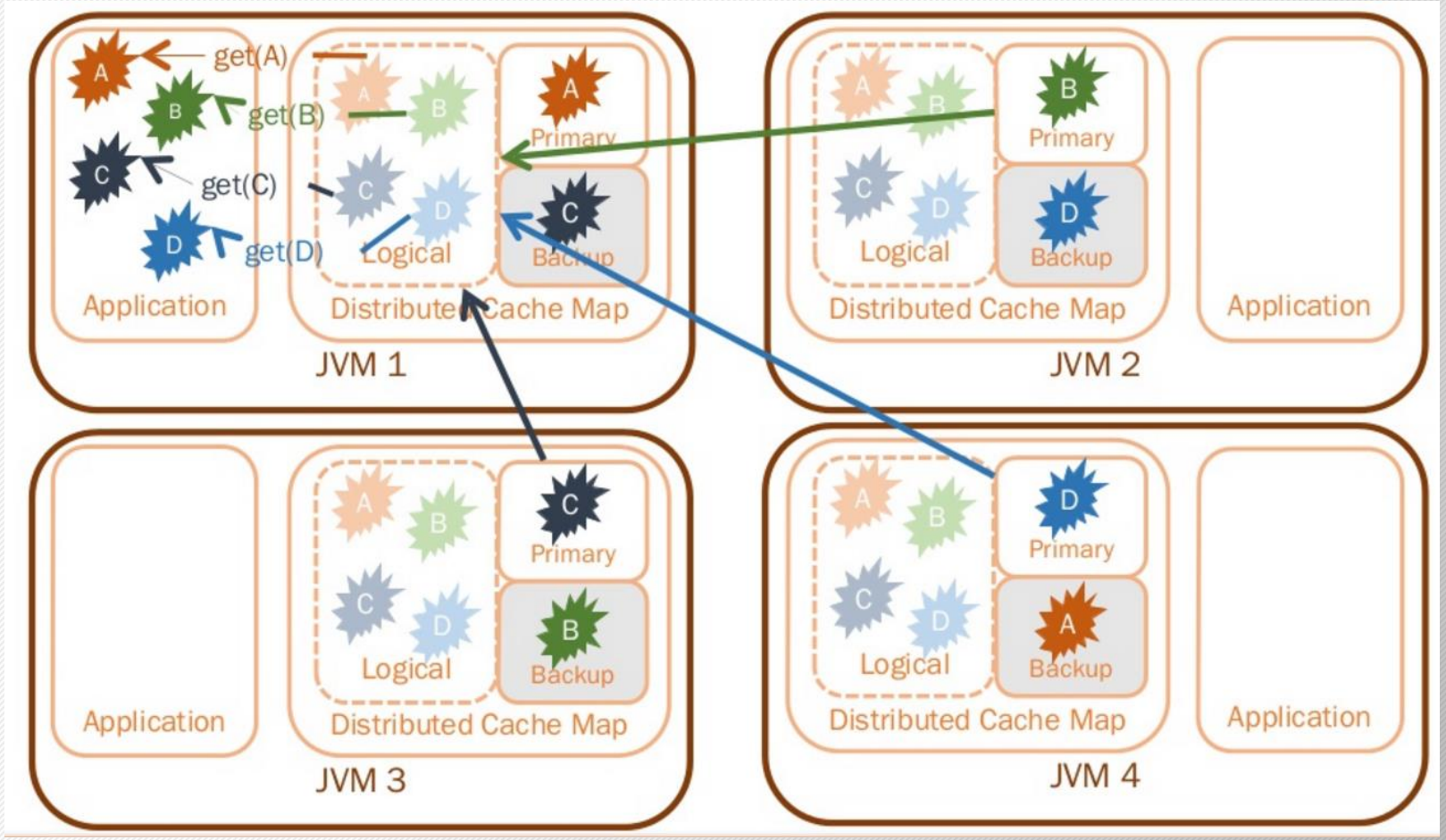
Distributed Support

- Hibernate Second Level Cache
  - Stale data problem!!!! Hazelcast to the rescue
- Spring Support
  - hazelcast-all.jar or hazelcast-spring.jar

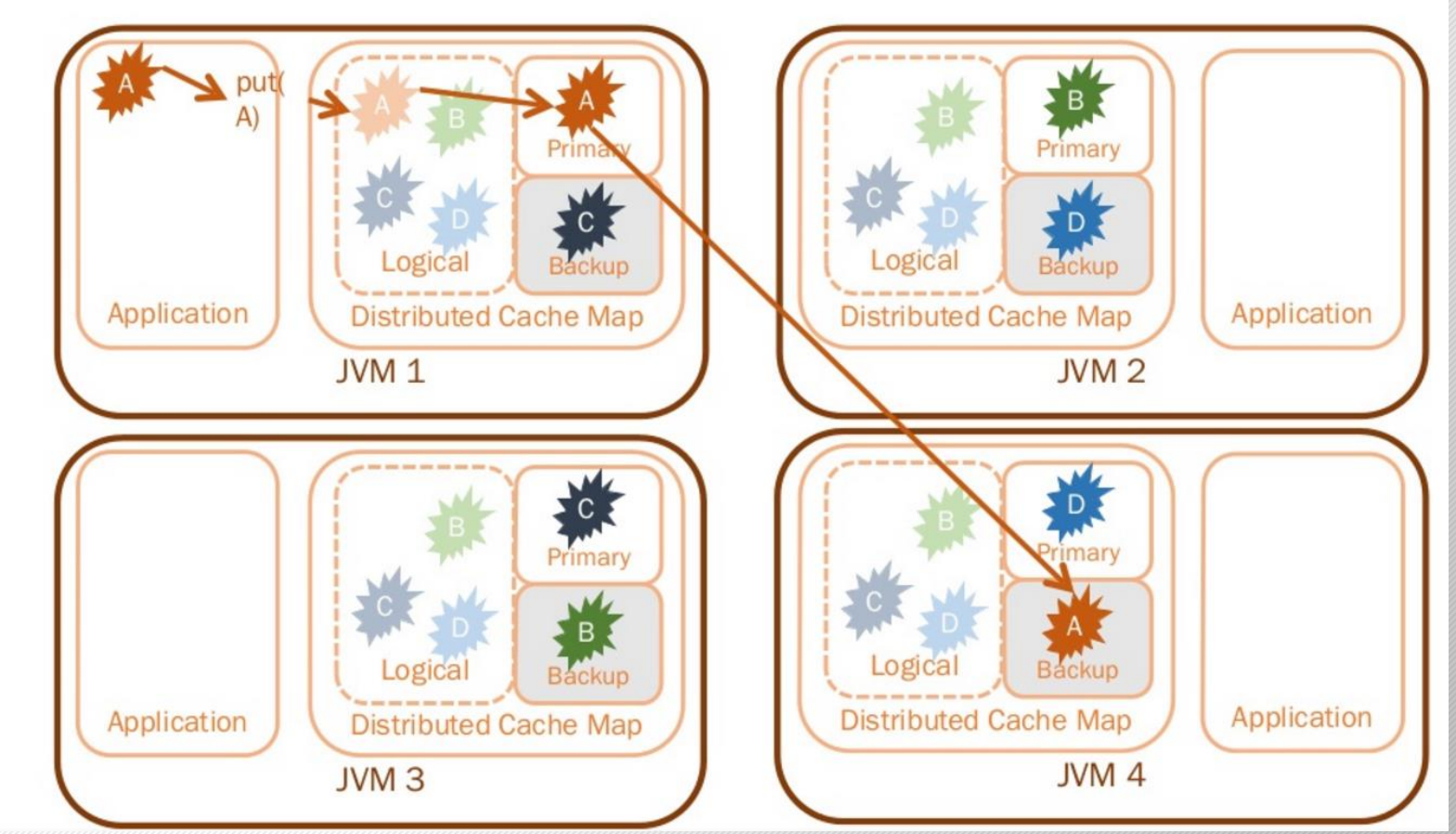
Distributed Caching



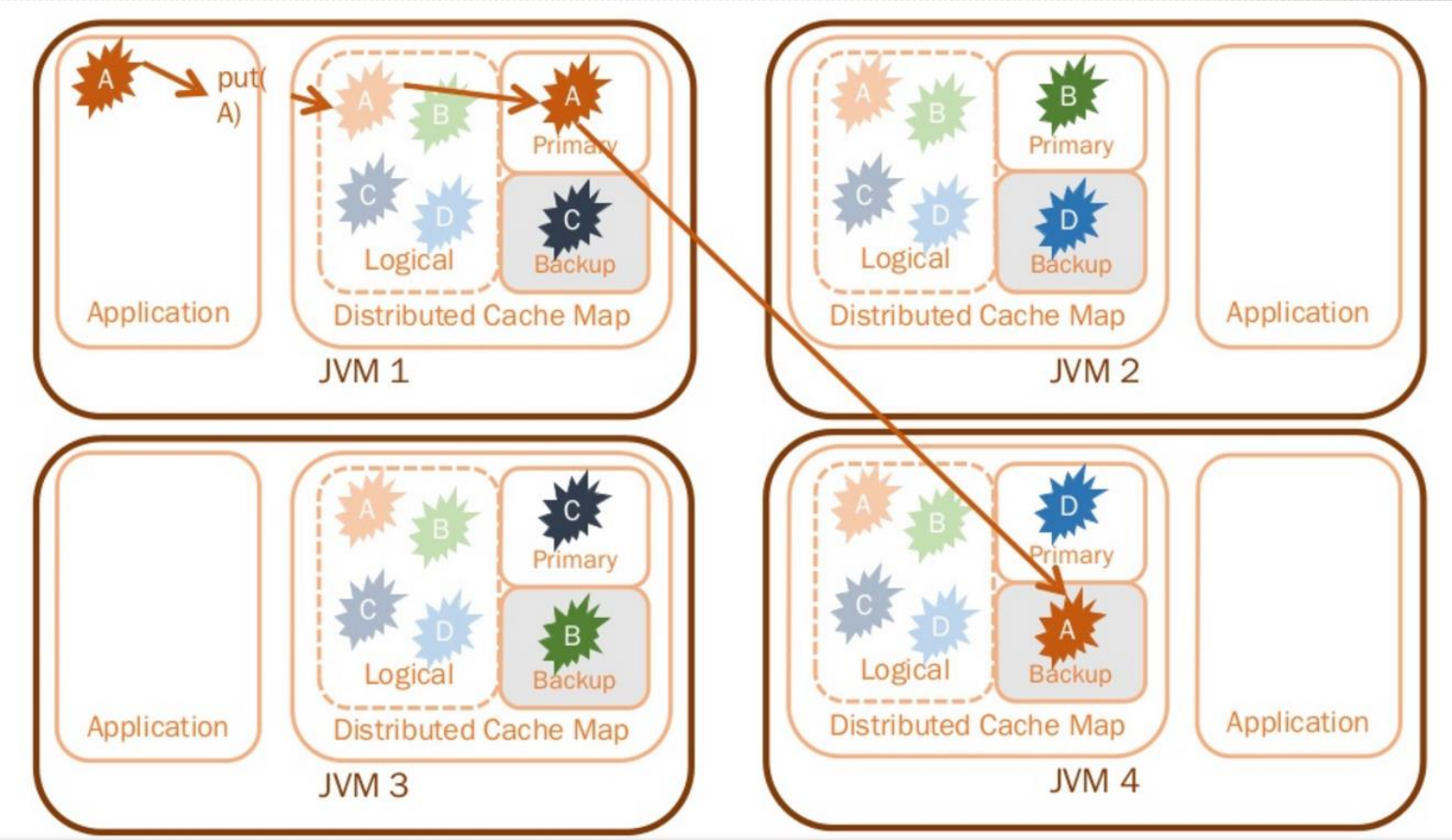
Get in Distributed Cache Map



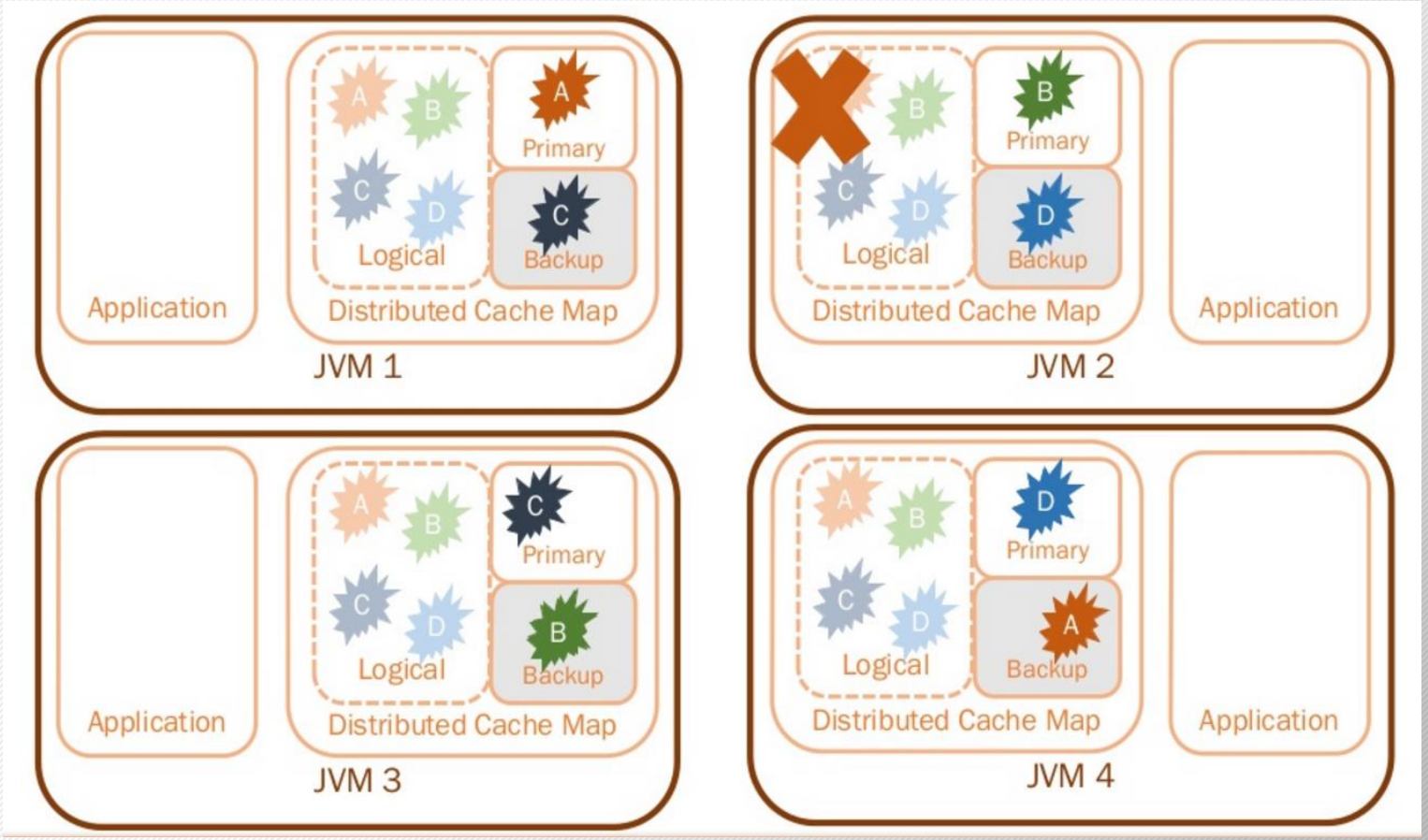
Put in Distributed Map



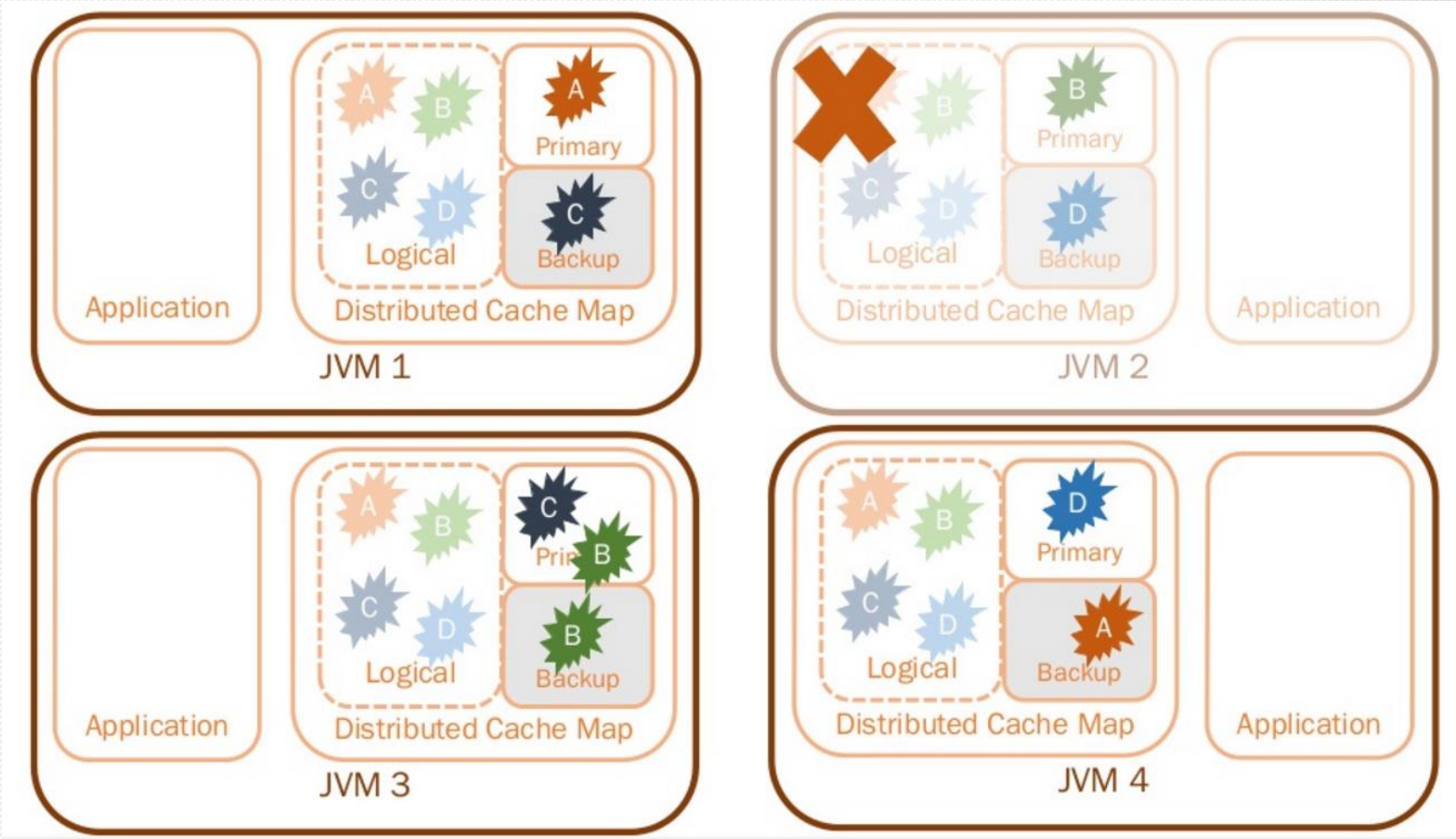
Put in Distributed Map



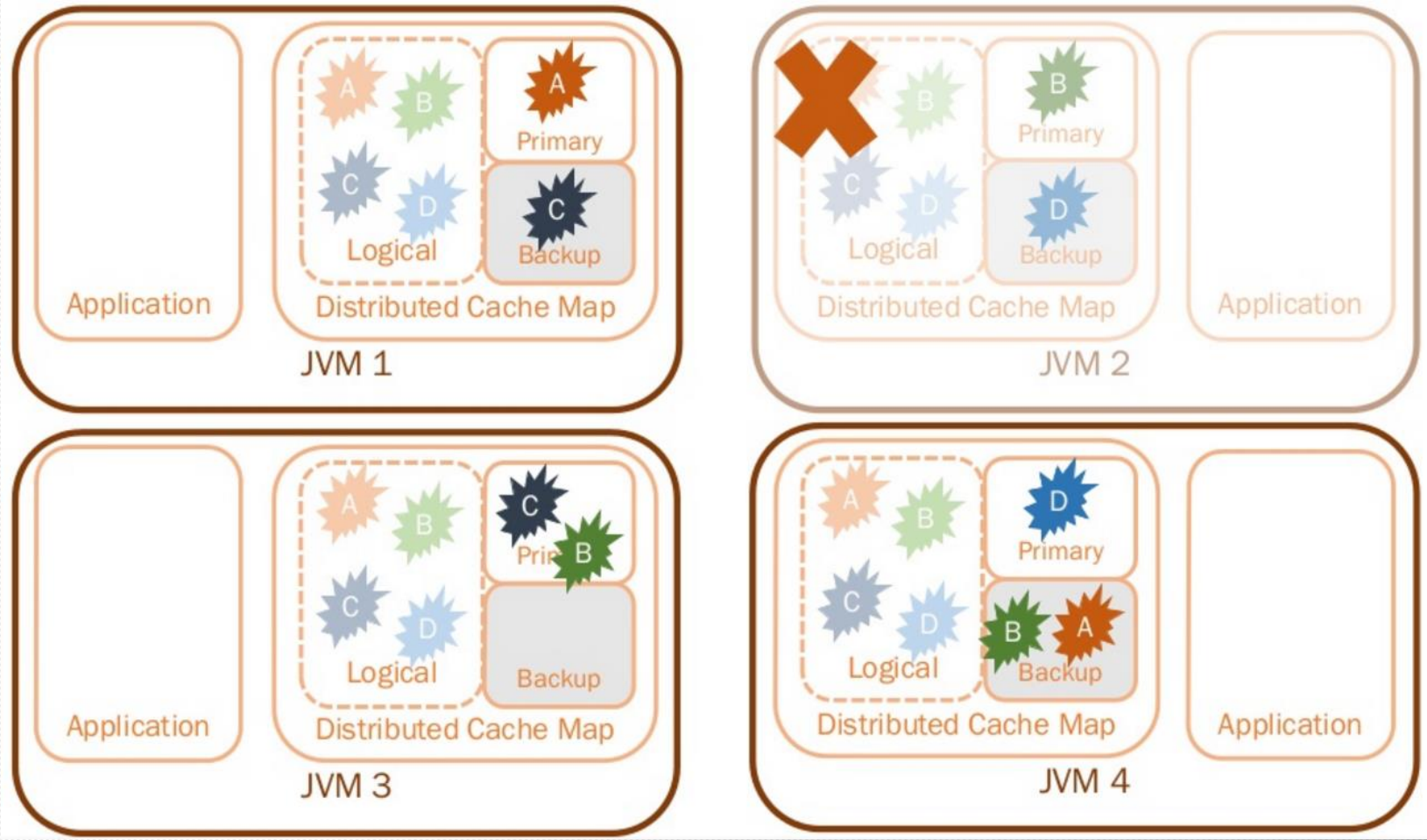
Failover in Distributed Cache



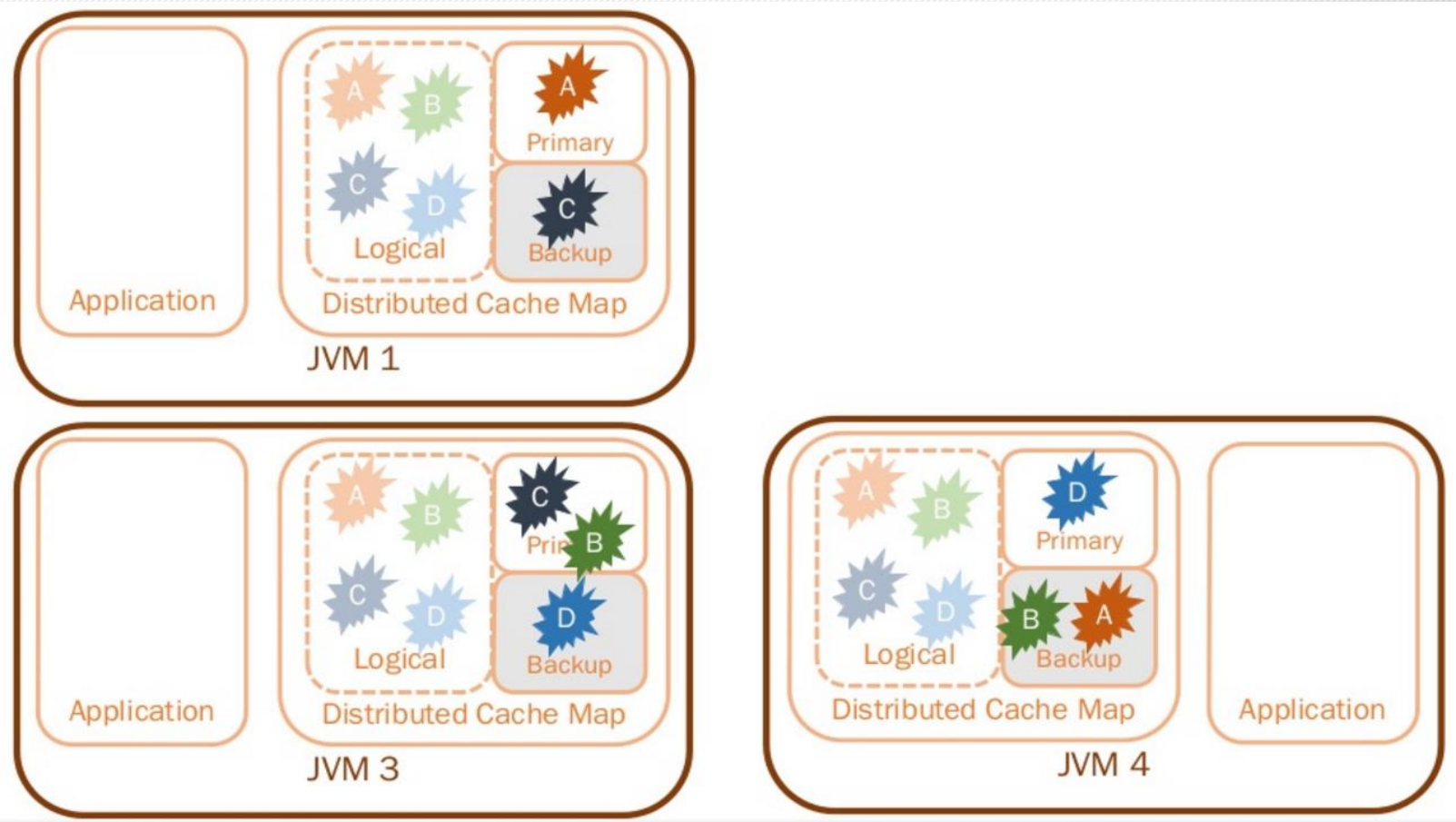
Failover in Distributed Cache



Failover in Distributed Cache



Failover in Distributed Cache



Distributed Data Structures

- Map
- Queue
- Multimap
- Set
- List

Contents

Query Mechanisms

- Queries are distributed
- Predicate filters accordingly
- Merge all results
- Fast and Concurrent

Criteria API

Predicate Class provides operators like:

Equal, notEqual, greaterThan, greaterEqual

Example

```
public class Employee implements Serializable {
private String name;
private int age;
private boolean active;
private double salary;
```

```
IMap<String, Employee> map = hazelcastInstance.getMap( "employee" );

EntryObject e = new PredicateBuilder().getEntryObject();
Predicate predicate = e.is( "active" ).and( e.get( "age" ).lessThan( 30 ) );

Set<Employee> employees = map.values( predicate );
```

Querying with SQL

- SqlPredicate takes regular Where clause

```
IMap<Employee> map = hazelcastInstance.getMap( "employee" );
Set<Employee> employees = map.values( new SqlPredicate( "active AND age < 30" ) );
```

- And/Or: <expression> AND <expression>
- Equality: =,!=,<,<=,>,>=
- Between:<attribute> [NOT] Between <values1> AND <values2>

- Indexing

```
map.addIndex( "active", false );
```

Aggregators

- Supplier

```
Supplier<...> supplier = Supplier.fromKeyPredicate(
    lastName -> "Jones".equalsIgnoreCase( lastName )
);
```

- Average

```
map.aggregate( Supplier.all( person -> person.getAge() ),
               Aggregations.integerAvg() );
```

- Sum

```
map.aggregate( Supplier.all( person -> person.getAge() ),
               Aggregations.integerSum() );
```

Aggregator's Contd.

- Minimum

```
map.aggregate( Supplier.all( person -> person.getAge() ),
               Aggregations.integerMin() );
```

- Maximum

```
map.aggregate( Supplier.all( person -> person.getAge() ),
               Aggregations.integerMax() );
```

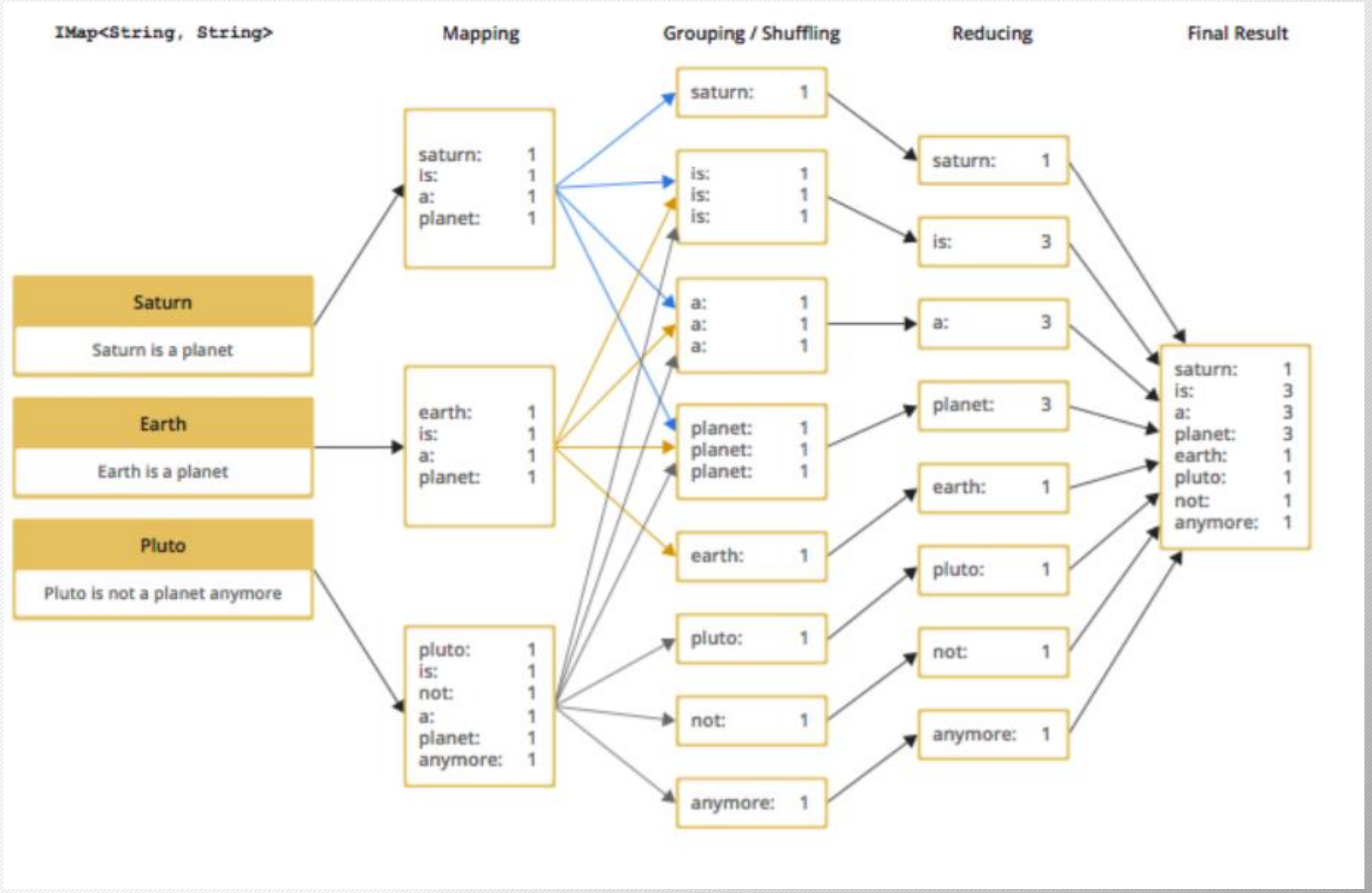
- Count

```
map.aggregate( Supplier.all(), Aggregations.count() );
```

MapReduce

Summary of main steps:

1. Read the source data.
2. Map the data to one or multiple key-value pairs.
3. Reduce all pairs with the same key.


IMap<String, String>
Mapping
Grouping / Shuffling
Reducing
Final Result
Saturn
Saturn is a planet
Earth
Earth is a planet
Pluto
Pluto is not a planet anymore
saturn: 1
is: 1
a: 1
planet: 1
earth: 1
is: 1
a: 1
planet: 1
pluto: 1
is: 1
not: 1
a: 1
planet: 1
anymore: 1
saturn: 1
is: 1
is: 1
is: 1
a: 1
a: 1
a: 1
planet: 1
planet: 1
planet: 1
earth: 1
pluto: 1
not: 1
anymore: 1
saturn: 1
is: 3
a: 3
planet: 3
earth: 1
pluto: 1
not: 1
anymore: 1
saturn: 1
is: 3
a: 3
planet: 3
earth: 1
pluto: 1
not: 1
anymore: 1

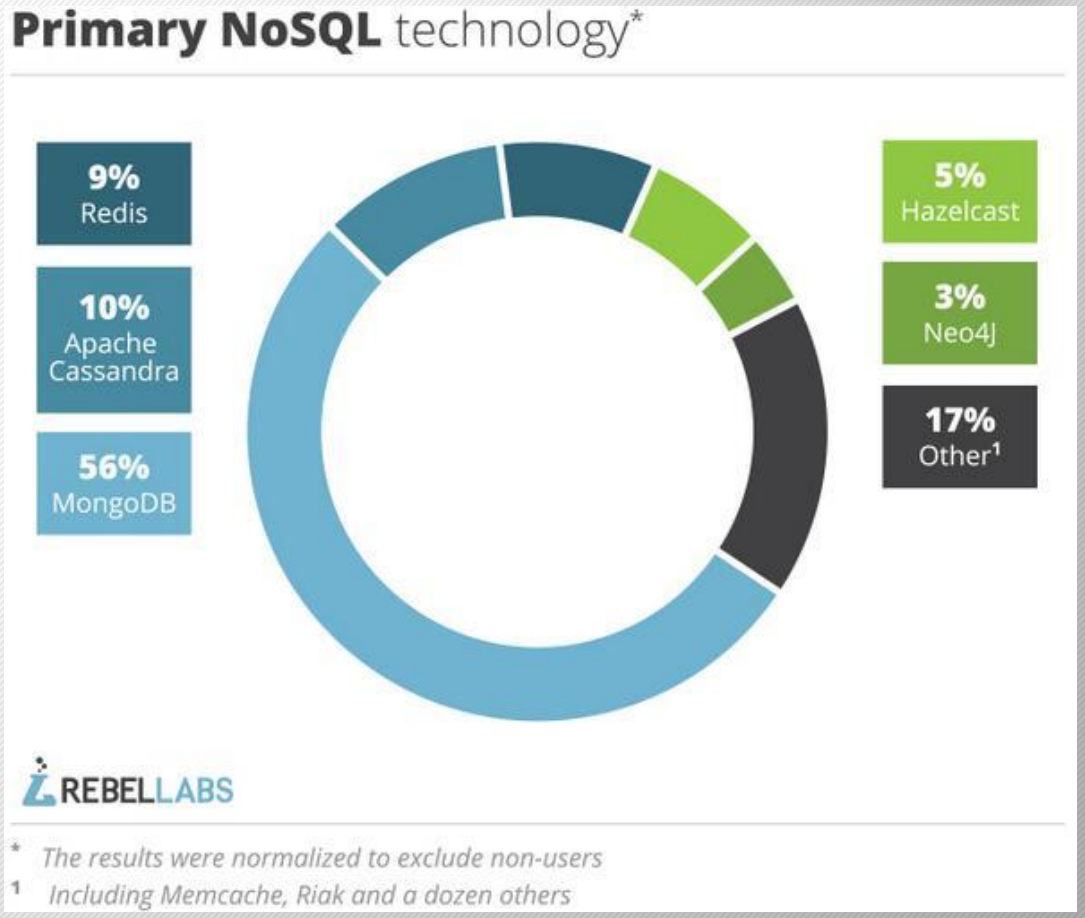
Contents

1. Introduction
2. Architecture/Data Model
3. Features
4. Query Language
5. Market Comparison

Hazelcast as NoSQL Database

Young, but popular

- Speed
- Memory
- Caching



Hazelcast as In-Memory Store

- Popular for:
  - Licensing
  - Support
  - Languages
- Competition:
  - Oracle Coherence,
  - VmWare Gemfire
  - Terracotta
- Persistence
  - Mapping
- No Single Point of Failure
- Avoids Garbage Collection delays

Hazelcast vs Redis

- Architecture
  - Language
  - Threading
- Scaling
  - Adding/Removing Nodes
- Failover
  - Node Relationship
- Clustering
  - Type
  - Addresses
  - Discovery
- Persistence
- Querying

Hazelcast vs Oracle Coherence

- Per-core pricing model
  - Cost
  - Flexibility
- Aging Technology
  - Features
  - 10 year old Java
- De-prioritization of Coherence
  - 12c
  - Support and Attention
- Complex Deployment
  - Time and Consulting

Hazelcast vs Couchbase

- Consistency
  - Backups and Replicas
- Serialization
  - Data types

Feature	Couchbase	Hazelcast
Licensing	Apache v2	Apache v2
Enterprise Support	Couchbase	Hazelcast
Implementation Language	Eglang, C Couchbase uses C for the Memory Cache (based on Memcached), Erlang for the Communications Manager.	Java
Stored Structures	Key-value, Document	Multiple storage structures. Key/Value (Map, MultiMap), Queue, Set, List, Topic.
Persistence	Yes	Yes The Java Interface in which Cache Misses and Puts can be coded against any backing store: for example, another RDBMS, NoSQL, HDFS, or network.
Near Cache	No	Yes
Scalability	Horizontal	Horizontal
REST API	Yes	Yes
Memcached Protocol	Yes	Yes Text-based version of a protocol only
Messaging	No	Yes Distributed Events. Queues / Topics.
Query Language	PUT/GET constructs, REST API	Java APIs, SQL Using SqlPredicate, it is possible to query IMaps with SQL-like syntax.

Hazelcast vs Apache Cassandra

- Why is Cassandra used?
  - Time
  - Data
  - License
  - Querying
  - Flexible
- But…

Industry use of Hazelcast

References

http://docs.hazelcast.org/docs/3.5/manual/html/

https://hazelcast.org/getting-started/

https://hazelcast.org/features/

https://hazelcast.com/use-cases/caching/

http://www.slideshare.net/tmatyashovsky/distributed-applications-using-hazelcast