



**HAZELCAST**

**Art of Data Distribution**

open source, in-memory data grid

**Talip Ozturk**

**@oztalip**

# Who uses Hazelcast?

Financials

Telco

Gaming

e-commerce

Every **sec.** one Hazelcast node starts around the globe



# 9th Fastest Growing Skill

► Hazelcast

## Hazelcast

▲ 176% y/y

Primary Industry: Computer Software

✓ Listed on your profile

See Suggested Skills

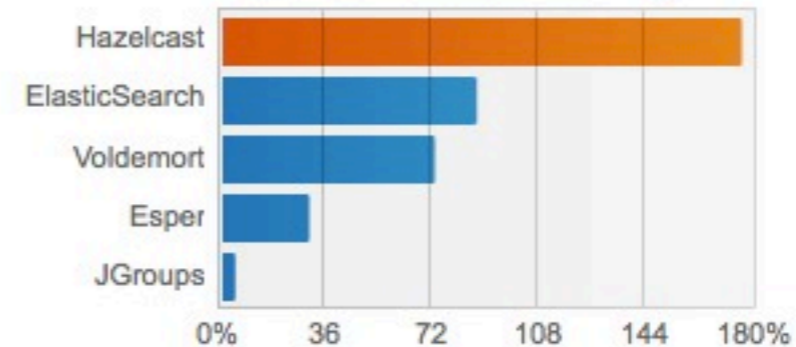
in Share

🐦 Tweet



Fastest growing skill

Relative Growth | Size | Age



## Hazelcast Professionals



**Fuad Malikov** (YOU)

Managing Partner at Hazelcast

After several years of consultancy in Tel co and Banking projects I have joined to the Hazelcast team to develop...



**Serban Balamaci** (2nd)

Senior Java Software Developer

Currently 1 year Java Developer at 1&1 Internet Mail&Media: New version of the WEB.DE portal. - New version of...



**Matias Sulik** (2nd)

Sr. Software Architect at Despegar.com

Matias has ten years of professional experience in tasks related to design and development of information systems...



## Hazelcast Jobs



**Software Development Manager**

SmartSynch - Foster City, CA or Jackson, MS



# Keywords

In-memory data grid

Distributed(Elastic) Cache

NoSQL

Clustering, Scalability, Partitioning

Cloud Computing



# Map

```
import java.util.Map;  
import java.util.HashMap;  
  
Map map = new HashMap();  
map.put("1", "value");  
map.get("1");
```



# Concurrent Map

```
import java.util.Map;  
import java.util.concurrent.*;  
  
Map map = new ConcurrentHashMap();  
map.put("1", "value");  
map.get("1");
```



# Distributed Map

```
import java.util.Map;
import com.hazelcast.core.Hazelcast;

Map map = Hazelcast.getMap("mymap");
map.put("1", "value");
map.get("1");
```



# Why Hazelcast?



To build **highly available** and **scalable** applications





# Alternatives

Oracle Coherence

IBM Extreme Scale

VMware Gemfire

Gigaspace

Redhat Infinispan

Gridgain

Terracotta

# Difference

License/Cost

Feature-set

API

Ease of use

Main focus (distributed map, tuple space, cache, processing vs. data)

Light/Heavy weight





# Apache License



Lightweight without any dependency

**1.7 MB jar**



# Introducing Hazelcast

Map, queue, set, list, lock, semaphore, topic and executor service

Native Java, C#, REST and Memcache Interfaces

Cluster info and membership events

Dynamic clustering, backup, fail-over

Transactional and secure



# Use-cases

Scale your application

Share data across cluster

Partition your data

Send/receive messages

Balance the load

Process in parallel on many JVM



# Demo



# Where is the Data?

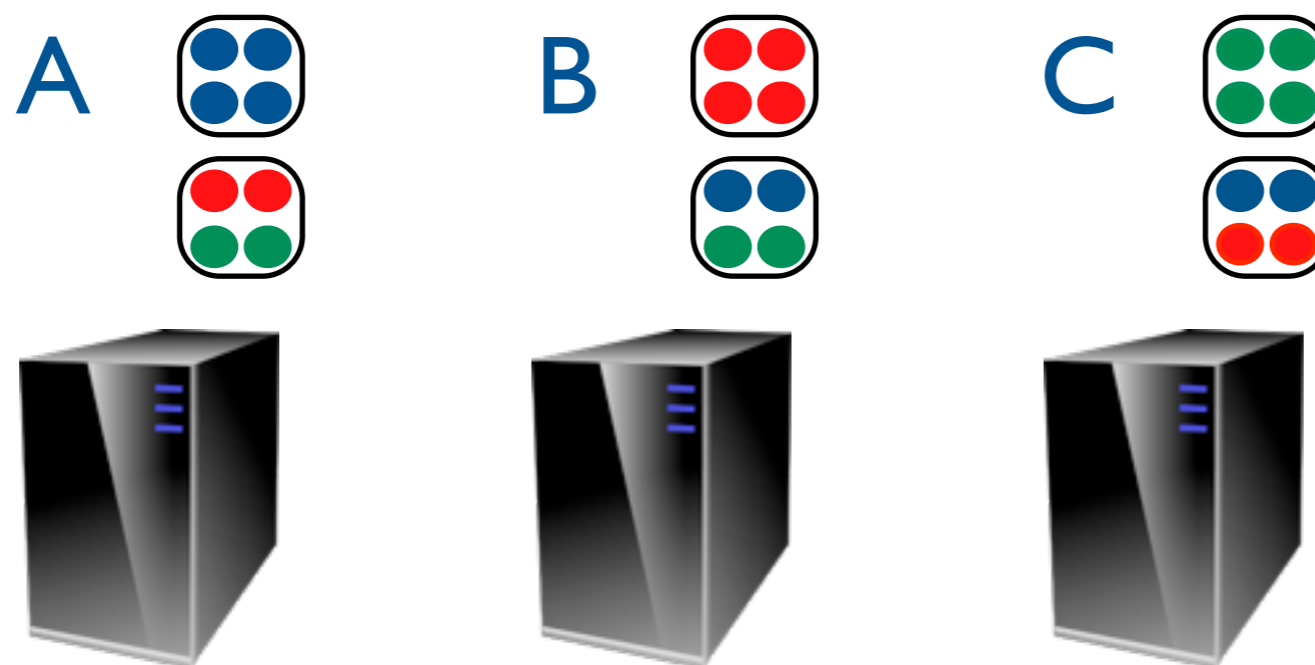
# Data Partitioning in a Cluster

Fixed number of partitions (default 271)

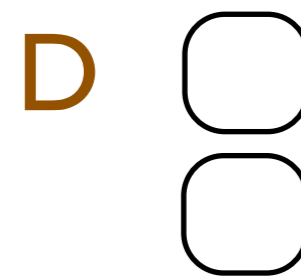
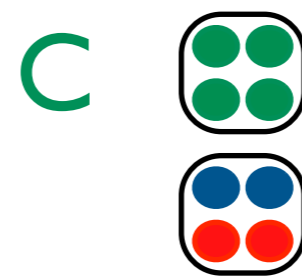
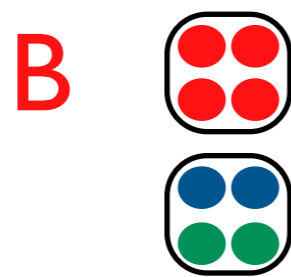
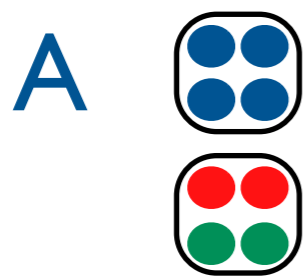
Each key falls into a partition

$\text{partitionId} = \text{hash}(\text{keyData}) \% \text{PARTITION\_COUNT}$

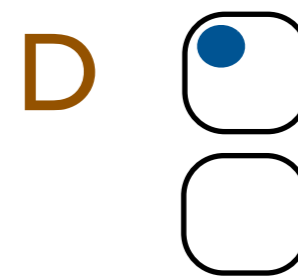
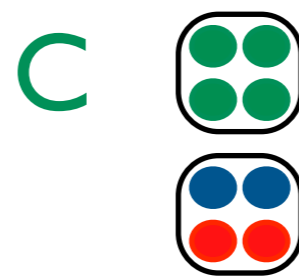
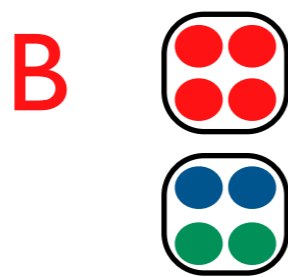
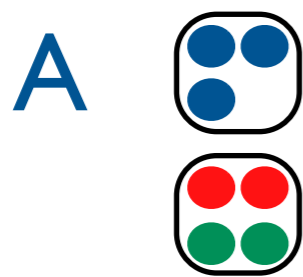
Partition ownerships are reassigned upon membership change



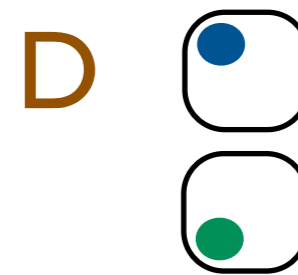
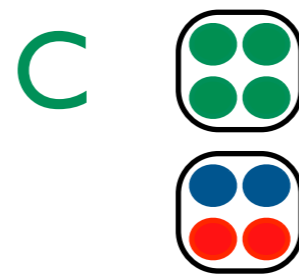
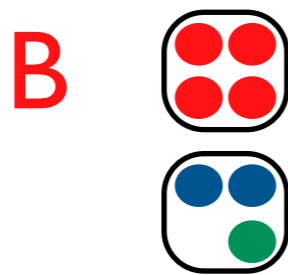
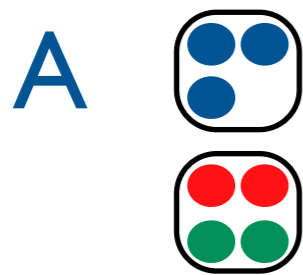
# New Node Added



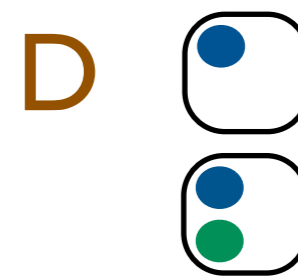
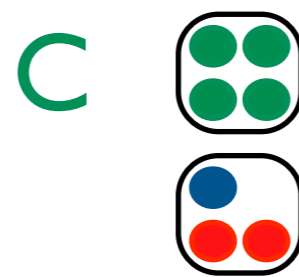
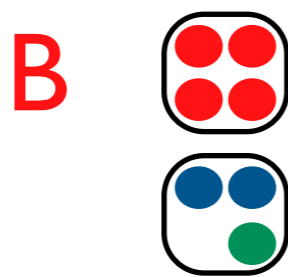
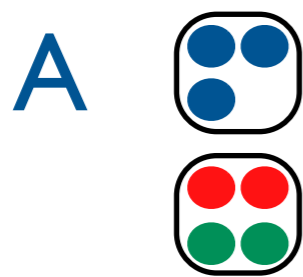
# Migration



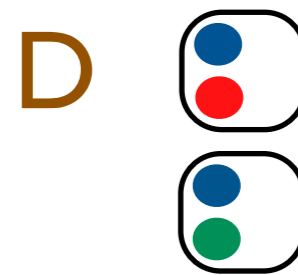
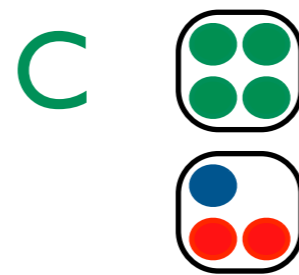
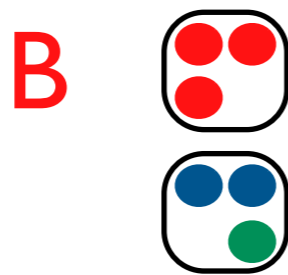
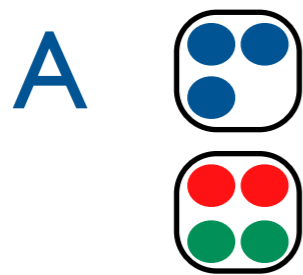
# Migration



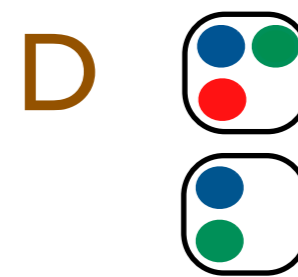
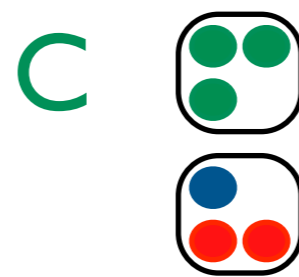
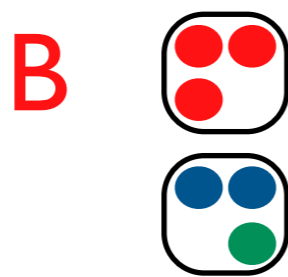
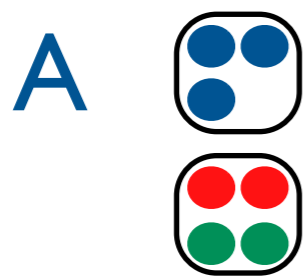
# Migration



# Migration

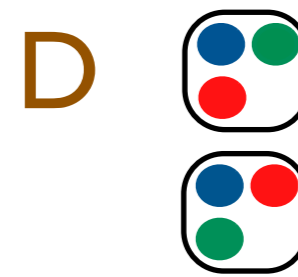
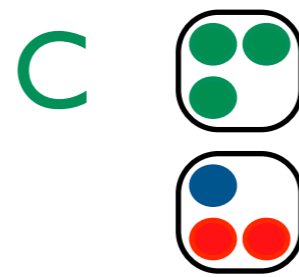
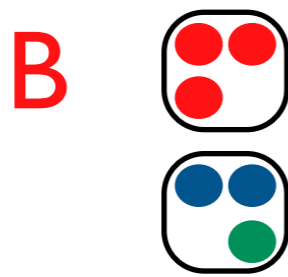
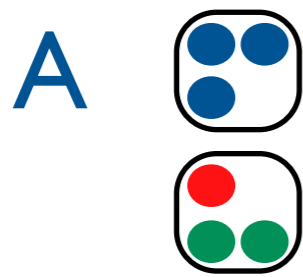


# Migration

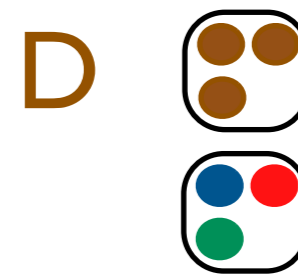
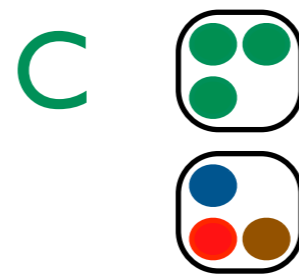
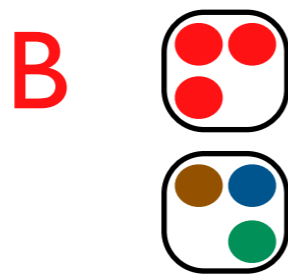
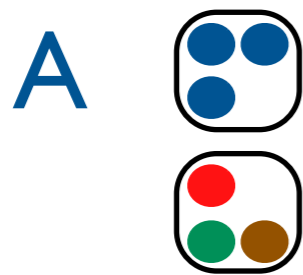




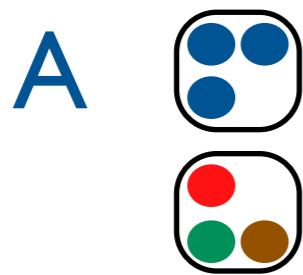
# Migration



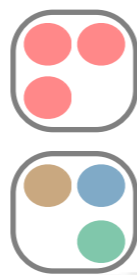
# Migration Complete



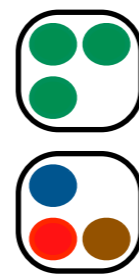
# Node Crashes



B



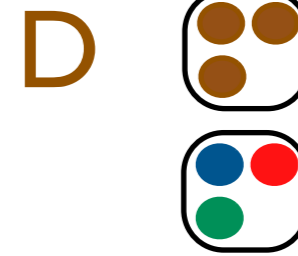
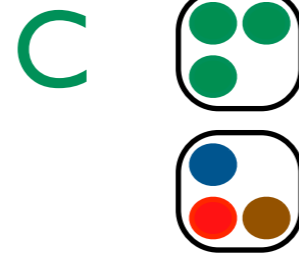
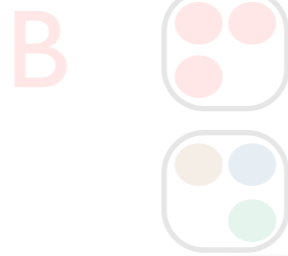
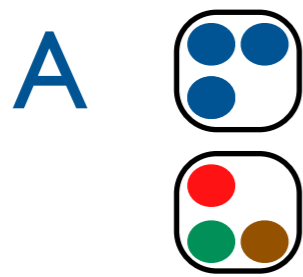
C



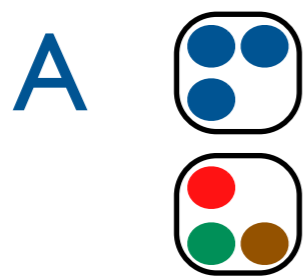
D



# Restoring Backups



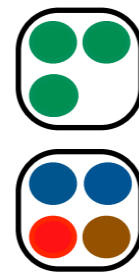
# Restoring Backups



B



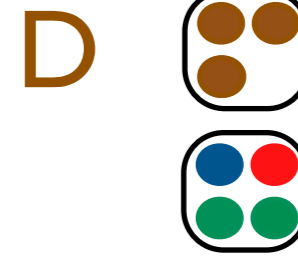
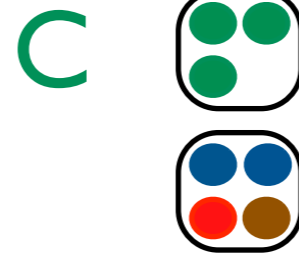
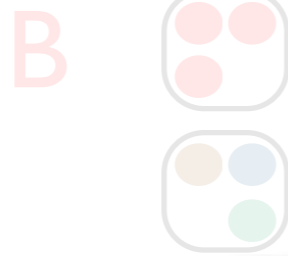
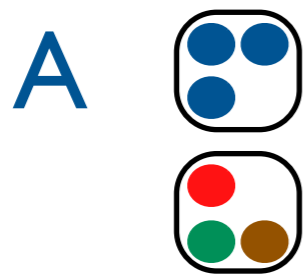
C



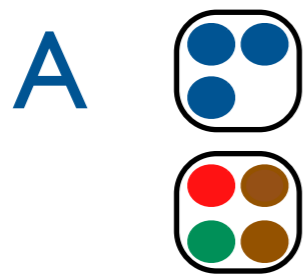
D



# Restoring Backups



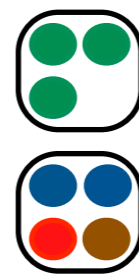
# Restoring Backups



B



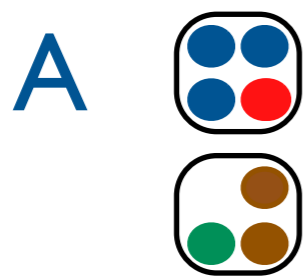
C



D



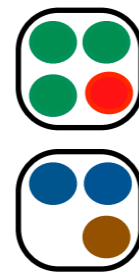
# Restoring Data from Backup



B



C

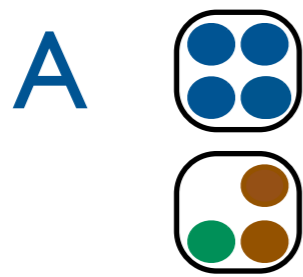


D





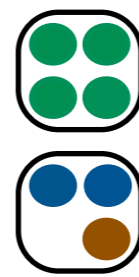
# Data is Recovered from Backup



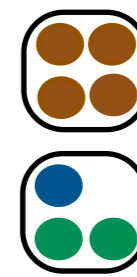
B



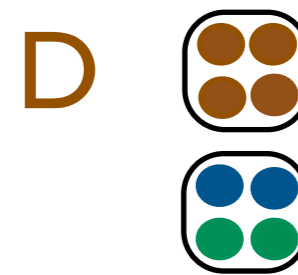
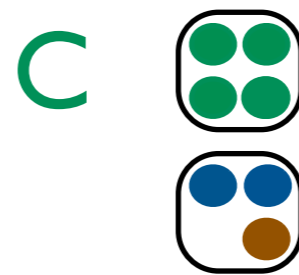
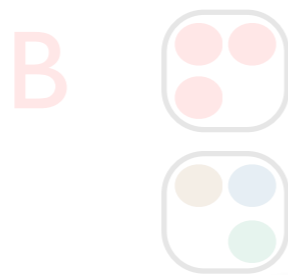
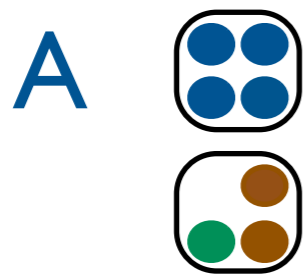
C



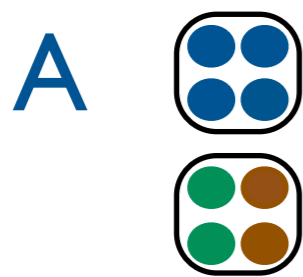
D



# Backup for Recovered Data



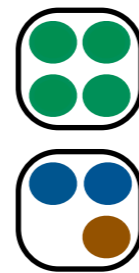
# Backup for Recovered Data



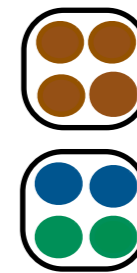
B



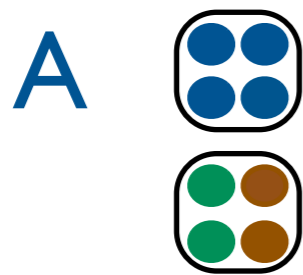
C



D



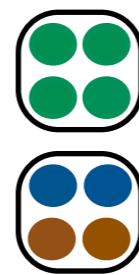
# Backup for Recovered Data



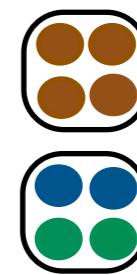
B



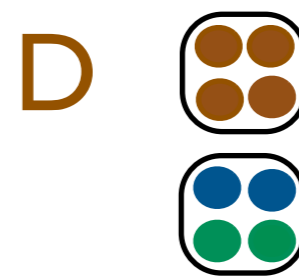
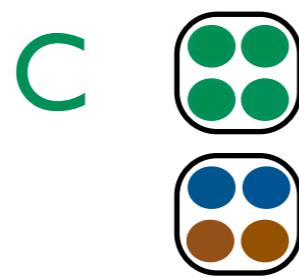
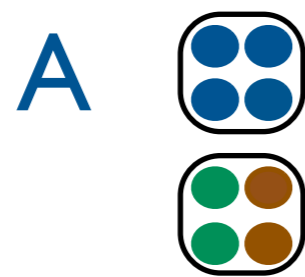
C



D

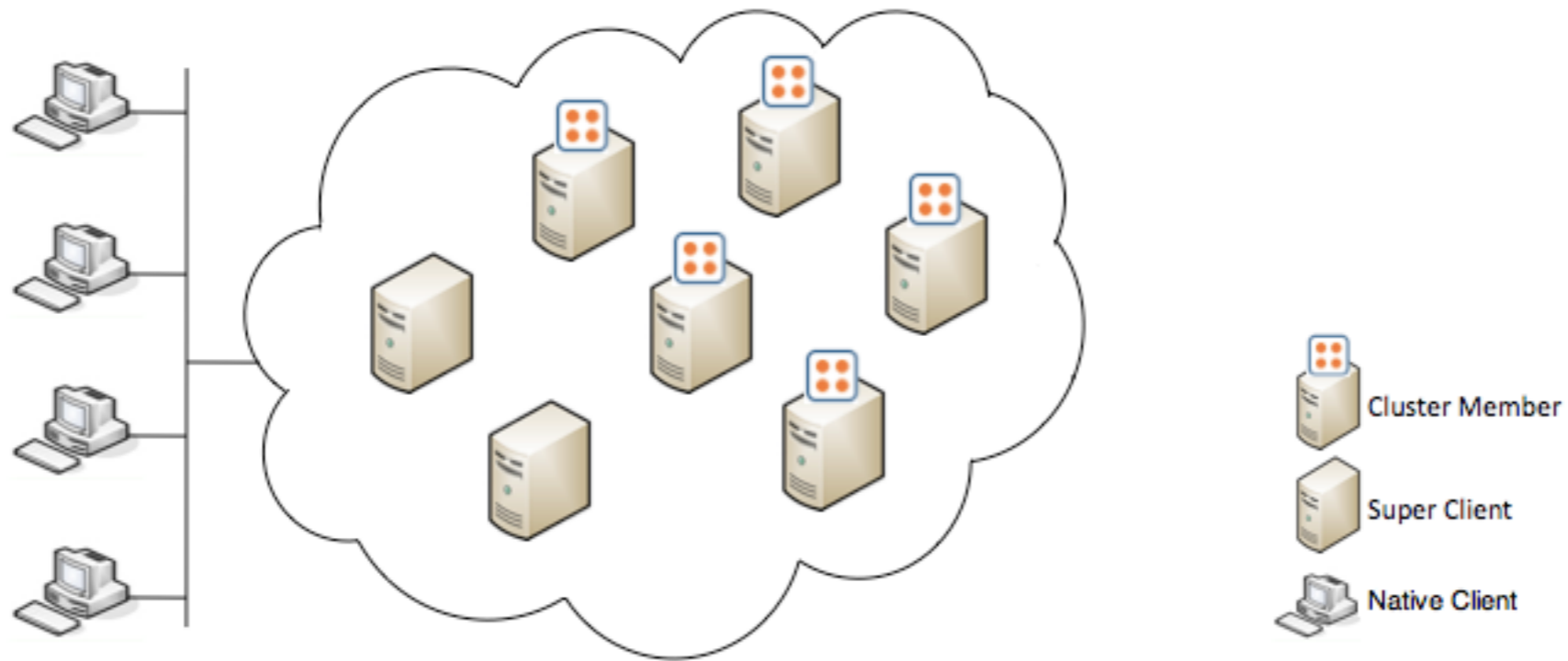


# All Safe



# Node Types

# Topology



## Native Client:

```
HazelcastInstance client = HazelcastClient.newHazelcastClient(clientConfig);
```

## Lite Member:

```
-Dhazelcast.lite.member=true
```



# Hazelcast Enterprise



# Community vs Enterprise

Enterprise =

Community +

Elastic Memory + Security + Man. Center



# Elastic Memory is OFF-HEAP storage

```
<hazelcast>
  ...
  <map name="default">
    ...
    <storage-type>OFFHEAP</storage-type>
  </map>
</hazelcast>
```



# JAAS based Security

Credentials

Cluster Login Module

Cluster Member Security

Native Client Security

Authentication

Authorization

Permission



# Code Samples

# Hazelcast

## Hazelcast is thread safe

```
Map<String, Employee> = Hazelcast.getMap("employees");  
List<Users> = Hazelcast.getList("users");
```

## Many instances on the same JVM

```
Config config = new Config();  
HazelcastInstance h1 = Hazelcast.newHazelcastInstance(config)  
HazelcastInstance h2 = Hazelcast.newHazelcastInstance(config)
```

## All objects must be serializable

```
class Employee implements java.io.Serializable  
    //better  
class Employee implements com.hazelcast.nio.DataSerializable
```

# Cluster Interface

```
import com.hazelcast.core.*;
import java.util.Set;

Cluster cluster = Hazelcast.getCluster();
cluster.addMembershipListener(listener);

Member localMember = cluster.getLocalMember();
System.out.println (localMember.getInetAddress());

Set<Member> setMembers = cluster.getMembers();
```



# Distributed Map

```
import com.hazelcast.core.*;
import java.util.ConcurrentMap;

Map<String, Customer> map = Hazelcast.getMap("customers");
map.put ("1", customer);
Customer c = map.get("1");

//ConcurrentMap methods
map.putIfAbsent ("2", new Customer("Chuck Norris"));
map.replace ("3", new Customer("Bruce Lee"));
```



# Distributed Queue

```
import com.hazelcast.core.Hazelcast;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.TimeUnit;

BlockingQueue<Task> queue = Hazelcast.getQueue("tasks");

queue.offer(task);
Task t = queue.poll();

//Timed blocking Operations
queue.offer(task, 500, TimeUnit.MILLISECONDS)
Task t = queue.poll(5, TimeUnit.SECONDS);

//Indefinitely blocking Operations
queue.put(task)
Task t = queue.take();
```





# Distributed Lock

```
import com.hazelcast.core.Hazelcast;
import java.util.concurrent.locks.Lock;

Lock mylock = Hazelcast.getLock(mylockobject);
mylock.lock();
try {
    // do something
} finally {
    mylock.unlock();
}

//Lock on Map
IMap<String, Customer> map = Hazelcast.getMap("customers");
map.lock("1");
try {
    // do something
} finally {
    map.unlock("1");
}
```



# Distributed Topic

```
import com.hazelcast.core.*;

public class Sample implements MessageListener {
    public static void main(String[] args) {
        Sample sample = new Sample();
        ITopic<String> topic = Hazelcast.getTopic ("default");
        topic.addMessageListener(sample);
        topic.publish ("my-message-object");
    }
    public void onMessage(Object msg) {
        System.out.println("Got msg :" + msg);
    }
}
```



# Distributed Events

```
import com.hazelcast.core.*;

public class Sample implements EntryListener {
    public static void main(String[] args) {
        Sample sample = new Sample();
        IMap map = Hazelcast.getMap ("default");
        map.addEntryListener (sample, true);
        map.addEntryListener (sample, "key");
    }
    public void entryAdded(EntryEvent event) {
        System.out.println("Added " + event.getKey() + ":" +
                           event.getValue());
    }
    public void entryRemoved(EntryEvent event) {
        System.out.println("Removed " + event.getKey() + ":" +
                           event.getValue());
    }
    public void entryUpdated(EntryEvent event) {
        System.out.println("Updated " + event.getKey() + ":" +
                           event.getValue());
    }
}
```



# Executor Service

# Hello Task

## A simple task

```
public class HelloTask implements Callable<String>, Serializable{
    @Override
    public String call(){
        Cluster cluster = Hazelcast.getCluster();
        return "Hello from " + cluster.getLocalMember();
    }
}
```

## Execute on any member

```
ExecutorService ex = Hazelcast.getExecutorService();
Future<String> future = executor.submit(new HelloTask());
// ...
String result = future.get();
```

## Attach a callback

```
DistributedTask task = ...
task.setExecutionCallback(new ExecutionCallback() {
    public void done(Future f) {
        // Get notified when the task is done!
    }
});
```



# Hazelcast can execute a task ...

1. On a specific node
2. On any available node
3. On a collection of defined nodes
4. On a node owning a key

```
ExecutorService executor = Hazelcast.getExecutorService();
FutureTask<String> task1, task2;

// CASE 1: Run task on a specific member.
Member member = ...
task1 = new DistributedTask<String>(new HelloTask(), member);
executor.execute(task1);

// CASE 2: Run task on a member owning the key.
Member member = ...
task1 = new DistributedTask<String>(new HelloTask(), "key");
executor.execute(task1);

// CASE 3: Run task on group of members.
Set<Member> members = ...
task = new MultiTask<String>(new HelloTask(), members);
executor.execute(task2);
```



# Executor Service Scenario

```
public int removeOrder(long customerId, long orderId){
    IMap<Long, Customer> map = Hazelcast.getMap("customers");
    map.lock(customerId);
    Customer customer = map.get(customerId);
    customer.removeOrder (orderId);
    map.put(customerId, customer);
    map.unlock(customerId);
    return customer.getOrderCount();
}
```



# Add Bonus Task

```
public class OrderDeletionTask implements Callable<Integer>, PartitionAware, Serializable {

    private long customerId;
    private long orderId;

    public OrderDeletionTask(long customerId, long orderId) {
        super();
        this.customerId = customerId;
        this.orderId = orderId;
    }

    public Integer call () {
        IMap<Long, Customer> map = Hazelcast.getMap("customers");
        map.lock(customerId);
        Customer customer = map.get(customerId);
        customer.removeOrder (orderId);
        map.put(customerId, customer);
        map.unlock(customerId);
        return customer.getOrderCount();
    }

    public Object getPartitionKey() {
        return customerId;
    }
}
```





# Send computation over data

```
public static int removeOrder(long customerId, long orderId){
    ExecutorService es = Hazelcast.getExecutorService();
    OrderDeletionTask task = new OrderDeletionTask(customerId, orderId);
    Future future = es.submit(task);
    int remainingOrders = future.get();
    return remainingOrders;
}
```



# Persistence

# Persistence

```
import com.hazelcast.core.MapStore,
import com.hazelcast.core.MapLoader,

public class MyMapStore implements MapStore, MapLoader {

    public Set loadAllKeys () {
        return readKeys();
    }

    public Object load (Object key) {
        return readFromDatabase(key);
    }

    public void store (Object key, Object value) {
        saveIntoDatabase(key, value);
    }

    public void remove (Object key) {
        removeFromDatabase(key);
    }

}
```

# Persistence

Write-Behind : asynchronously storing entries

Write-Through : synchronous

Read-Through : if `get(key)` is null, load it



# Recap

- Distributed implementation of
  - Map
  - Queue
  - Set
  - List
  - MultiMap
  - Lock
  - Executor Service
  - Topic
  - Semaphore
  - AtomicLong
  - CountdownLatch
- Embedded, but accessible through
  - Native Java & C# Client
  - REST
  - Memcache
- Dynamic
  - Cluster
  - Add/ remove nodes
  - Backup
  - Fail-over



# Q & A



**JOIN US!**

**[github.com/hazelcast/](https://github.com/hazelcast/)**

**[hazelcast@googlegroups.com](mailto:hazelcast@googlegroups.com)**

**[@hazelcast](#)**

**[Hazelcast Hackers Linkedin group](#)**