



Hazelcast Essentials

About Me

Rahul Gupta



Senior Solutions Architect

Worked with Terracotta

In-memory Distributed Systems since 2009

Java Programmer since 1998

follow me [@wildnez](#)

Prerequisites



Prerequisites checklist

- Laptop
- JDK 8
 - `java -version`
- Apache Maven 3.2.x
 - `mvn -v`
- IDE with Maven support
 - IntelliJ IDEA CE
 - Eclipse

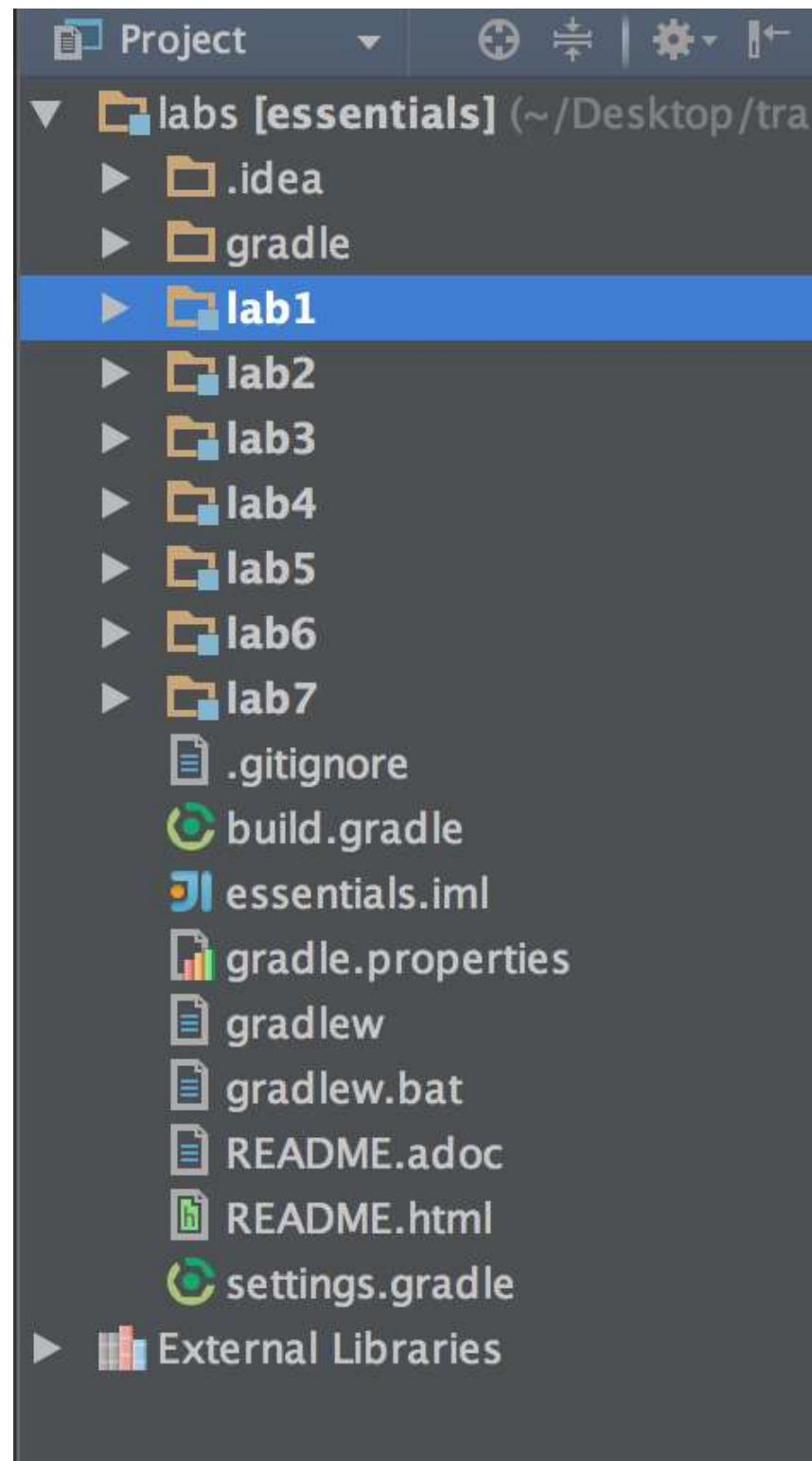


Prerequisites checklist (continue)

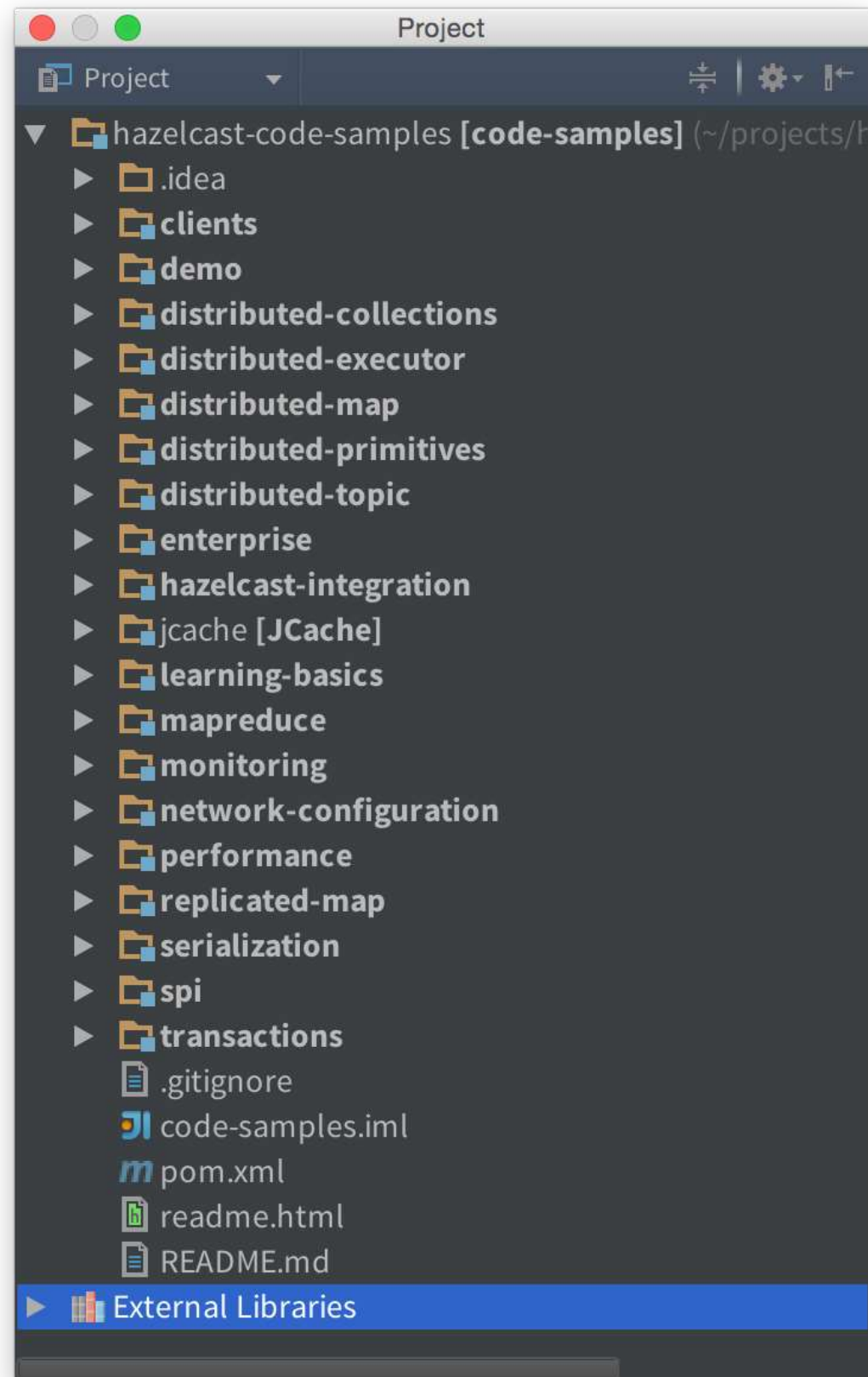
- Training Labs
 - ***zip provided***
- Download Hazelcast Samples
 - GIT - optional
 - git clone <https://github.com/hazelcast/training>
 - git clone <https://github.com/wildnez/public-training/tree/master/bootcamp/labs/lab1>
- Import to IDE



Prerequisites checklist (continue)



Prerequisites checklist (continue)



Learning Objectives

In this module you will:

Understand what an in-memory data grid is.

Identify common use cases of Hazelcast.

Recognize the advantages that Hazelcast can bring to your ecosystem.

Outline the system architecture utilized by Hazelcast.



What's Hazelcast?

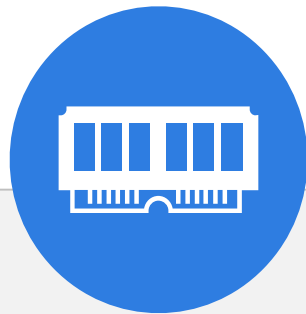


What is Hazelcast?

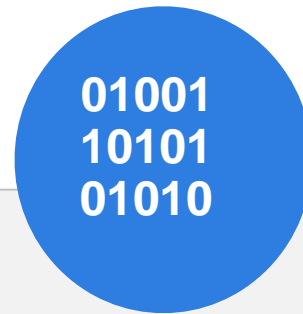
**Hazelcast is a distributed,
highly available and scalable
Open Source In-Memory Data Grid**



What is IMDG?



In Memory
Data **Storage**



In Memory
Data **Messaging**



In Memory
Data **Computing**



When you need?

- Cache to overcome legacy data bottlenecks.
- Cache for transient data.
- Primary store for modern apps.
- NoSQL database at in-memory speed.
- Data services fabric for real-time data integration.
- Compute grid at in-memory speed



IMDS - In Memory Datastore

Stores all records in memory, preventing slow disk reads

Scales to thousands of machines and terabytes of data

Evenly spreads data across cluster members

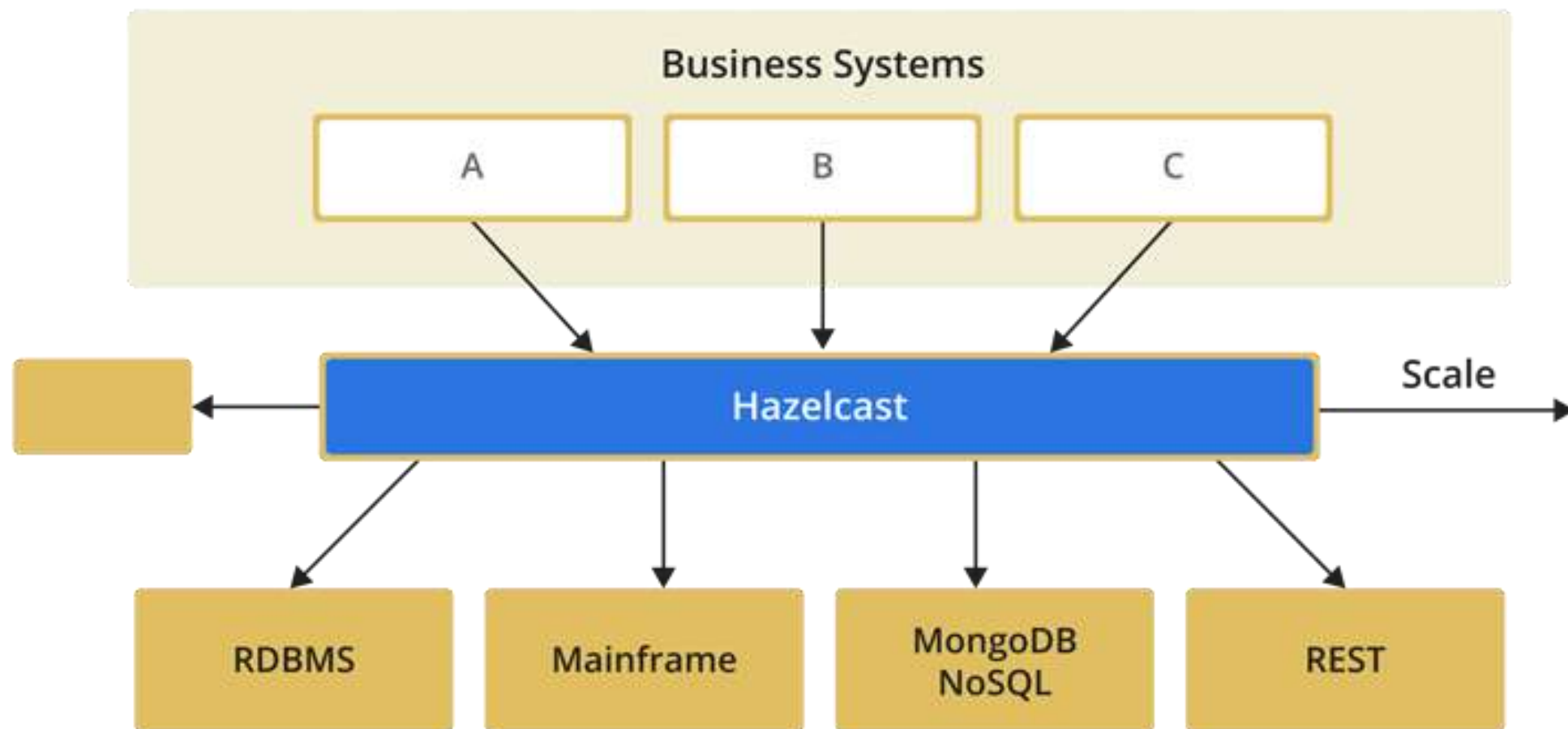
Provides fault tolerance with data backups

Allows the data to be accessed by key or queried with predicates

Allows large chunks of data to be updated in an efficient manner



IM Datastore (Caching) Use-Case



IMDC - In Memory Distributed Computing

`ExecutorService` for distributed environments

Flexible task routing (member, key owner, member subset)

Entry Processor

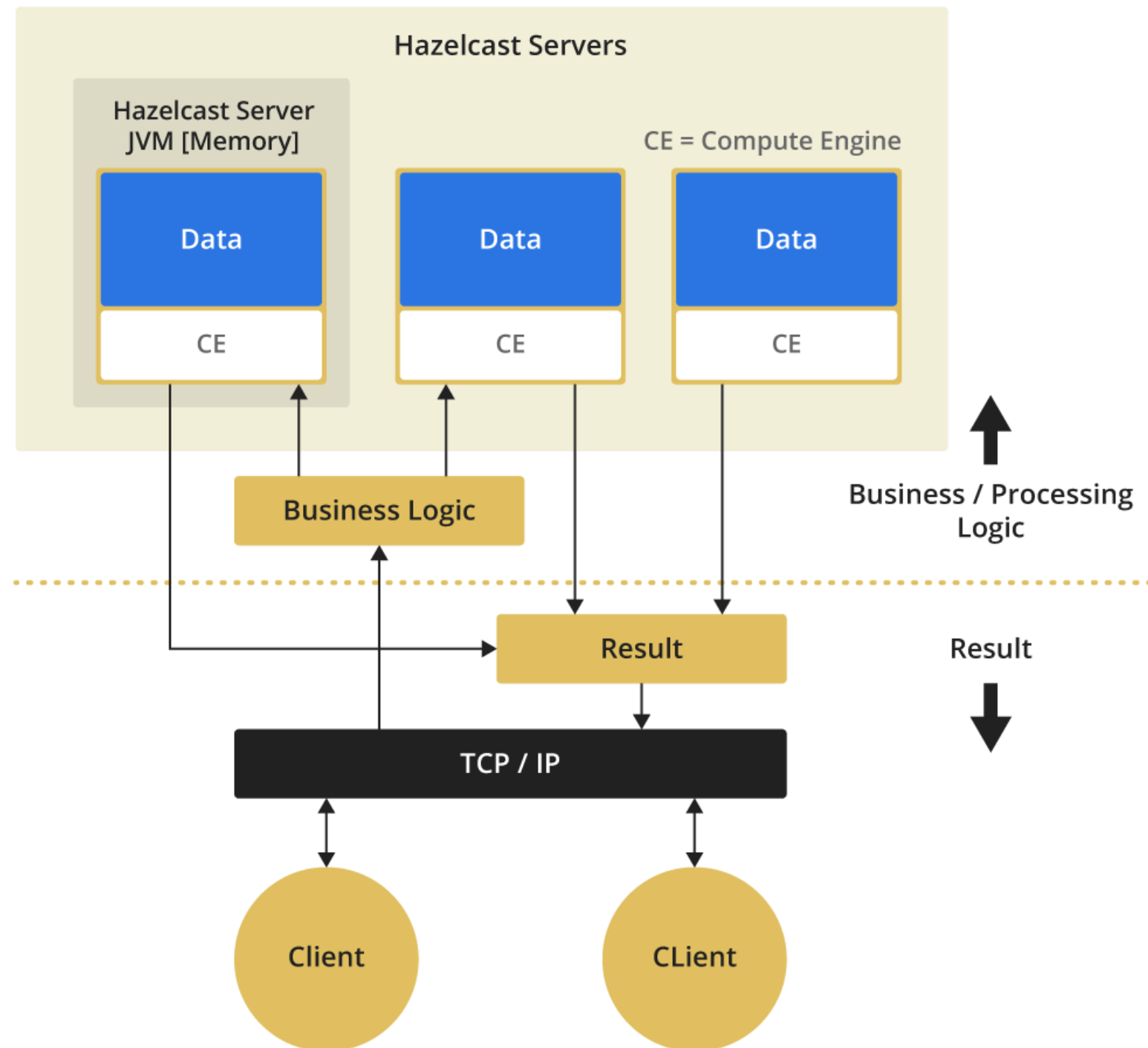
Aggregations

MapReduce

Interceptors



IM Distributed Computing Use Case



IMDM - In Memory Distributed Messaging

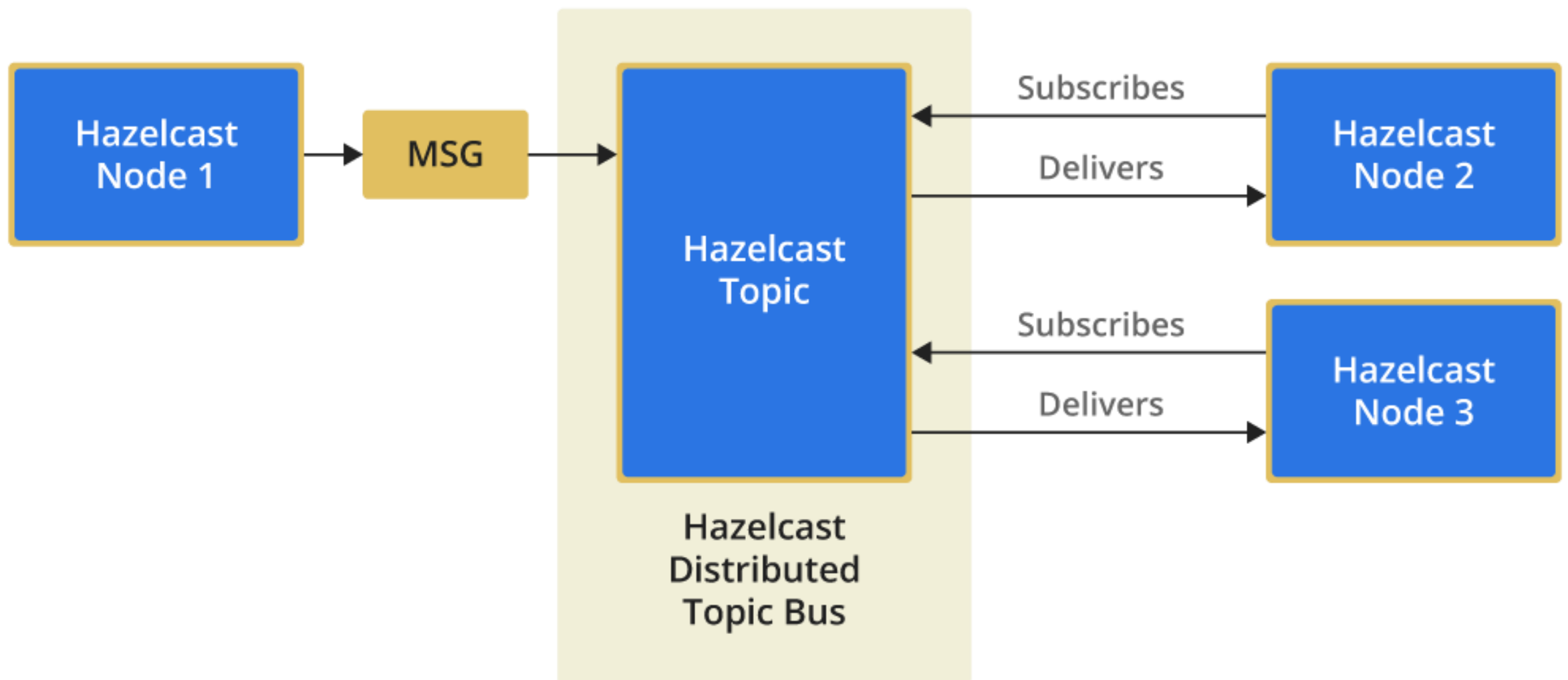
Allows you to publish messages to multiple subscribers

Messages are processed in the same order they were published

Resilient to member failures

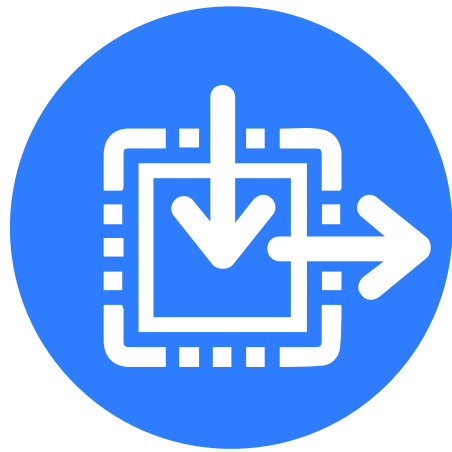


IM Distributed Messaging Use Case



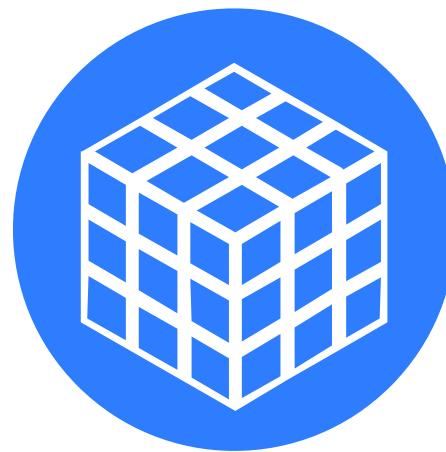
Hazelcast Features

High-Density Caching



- High-Density Memory Store, client and member
- Full JCache support
- Elastic scalability
- Super speeds
- High availability
- Fault tolerance
- Cloud readiness

In-Memory Data Grid



- Simple, modern APIs
- Automatic data recovery
- Object-oriented and non-relational
- Elastic and scalable
- Transparent database integration
- Browser-based cluster management

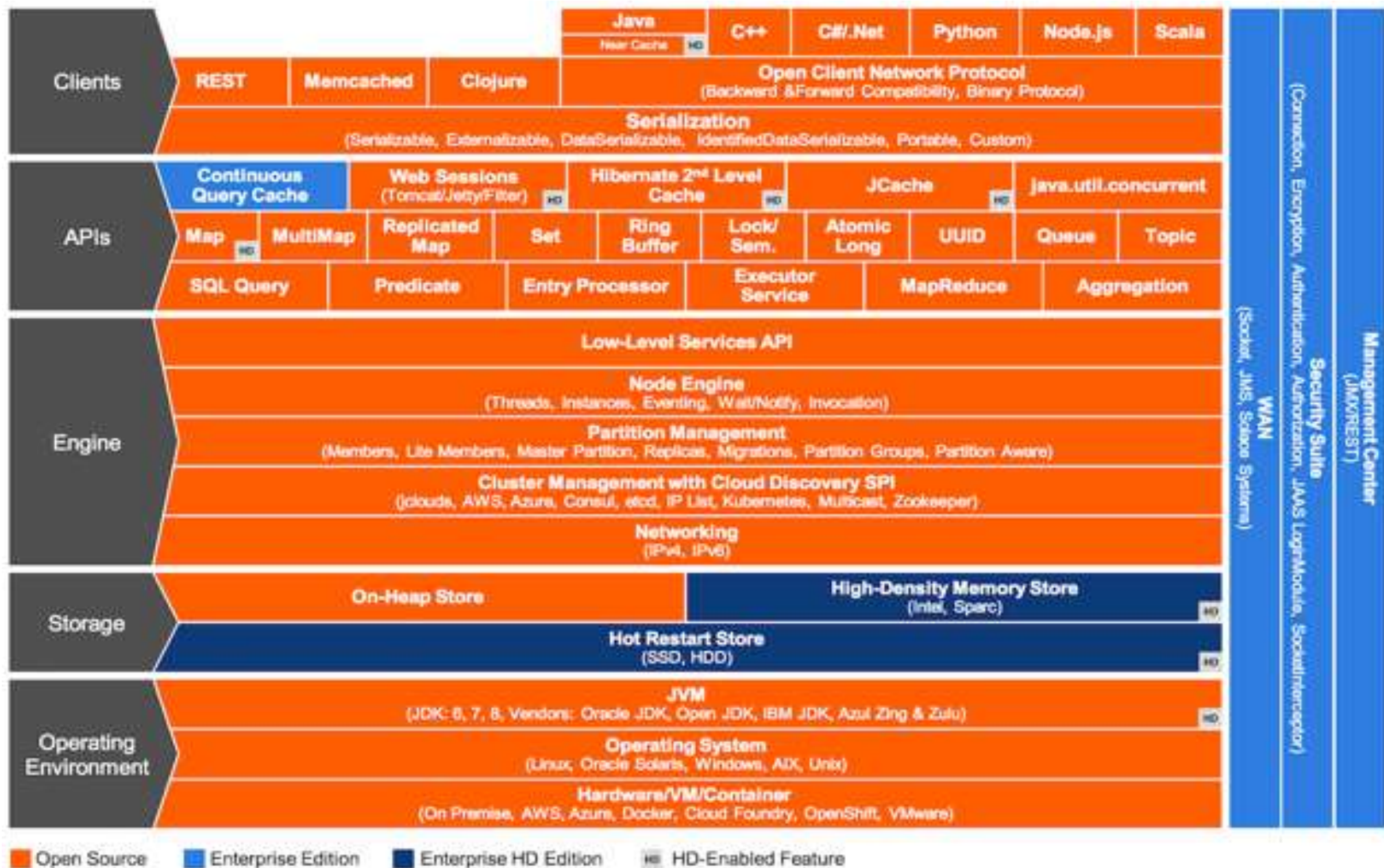
Web Session Clustering



- High performance
- No application alteration
- Easy scale-out
- Fast session access
- Off load to existing cluster
- Tomcat, Jetty and Generic



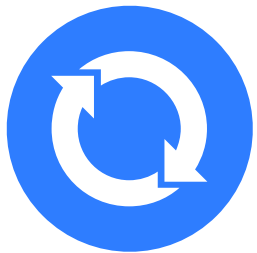
Why Hazelcast?



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



Cluster Configuration



Learning Objectives

In this module you will:

Set up a basic Hazelcast cluster

Try out different member discovery mechanisms

Get to know other network configuration options



Forming a Cluster

Hazelcast cluster members run on JVMs

Cluster members discover each other via Multicast (Default)

Use TCP/IP lists when Multicast not possible

Segregate clusters on same network via configuration

Hazelcast can form clusters on Amazon EC2.



Hazelcast Configuration

- XML configuration is used by default
- Hazelcast searches for `hazelcast.xml` on classpath
- Will fallback to `hazelcast-default.xml`
- Hazelcast can also be configured programmatically or with Spring
- Config is locked at start-up, cannot dynamically change (feature coming soon)



Lab 1 - Form a cluster on your laptop

lab1/src/main

Edit **resources/hazelcast.xml** to look like this, replace login and password with your own and then run the main class.

```
<hazelcast xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.hazelcast.com/schema/config
                               http://www.hazelcast.com/schema/config/hazelcast-config-
3.5.xsd"
           xmlns="http://www.hazelcast.com/schema/config">

  <group>
    <name>myCluster</name>
    <password>letmein</password>
  </group>

  <network>
    <join>
      <multicast enabled="true"/>
    </join>
  </network>
</hazelcast>
```



Lab 1 - TOP TIP

- The `<group>` configuration element is your friend
- It will help you to isolate your cluster on the multicast-enabled network
- Don't make a mistake of joining another developer's cluster or worse still a production cluster!



Lab 1 - Configuration via API

lab1/src/main/java

Add `GroupConfig` to the `Config` instance.

```
public class ProgrammaticMain {  
    public static void main(String[] args) {  
        Config config = new Config();  
        GroupConfig groupConfig = new GroupConfig("NAME", "PASSWORD");  
        config.setGroupConfig(groupConfig);  
        Hazelcast.newHazelcastInstance(config);  
    }  
}
```



Lab 1 - TOP TIP

- You can run multiple Hazelcast instances in one JVM.
- Handy for unit testing.

```
HazelcastInstance hz1 = Hazelcast.newHazelcastInstance();  
HazelcastInstance hz2 = Hazelcast.newHazelcastInstance();
```



Lab 1 - Configure Cluster to use TCP/IP

lab1/src/main/resources

1. Disable multicast discovery
2. Add TCP/IP configuration with your IP address

```
<hazelcast xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.hazelcast.com/schema/config
                               http://www.hazelcast.com/schema/config/hazelcast-
config-3.5.xsd"
           xmlns="http://www.hazelcast.com/schema/config">

  <group>
    <name>myCluster</name>
    <password>letmein</password>
  </group>

  <network>
    <join>
      <multicast enabled="false"/>
      <tcp-ip enabled="true">
        <member>192.168.10.10</member>
      </tcp-ip>
    </join>
  </network>
</hazelcast>
```



Wildcard Configuration

Hazelcast supports wildcards for config.

```
<map name="testmap*">  
  <time-to-live-seconds>10</time-to-live-seconds>  
</map>
```

- Beware of ambiguous config
- Hazelcast won't pick up the best match or figure out the correct configuration block by order - the results will be random!

```
<map name="testmap*">  
  <time-to-live-seconds>10</time-to-live-seconds>  
</map>  
<map name="testm*">  
  <time-to-live-seconds>2000</time-to-live-seconds>  
</map>
```



Properties

Hazelcast supports property replacement in XML config

```
<map name="testmap*">  
  <time-to-live-seconds>${testmap.ttl}</time-to-live-seconds>  
</map>
```

- Uses System Properties by default
- A Properties Loader can be configured

```
Properties properties = new Properties();  
properties.setProperty("testmap.ttl", "10");
```

```
Config config = new XmlConfigBuilder()  
    .setProperties(properties)  
    .build();
```

```
Hazelcast.newHazelcastInstance(config);
```



Distributed Map (IMap)



Learning Objectives

In this module you will:

- Get a basic overview of what a distributed map is
- Understand pros and cons of different object storage options
- Use eviction to prevent map overflow
- Perform complex queries with Predicate API
- Discover other flavors of the `IMap`:
 - `ReplicatedMap`
 - `MultiMap`



Distributed Maps

Hazelcast `IMap` implements `java.util.ConcurrentMap` interface and supports all of its functionality

It also provides several extra features

- ❑ Entry Listeners
- ❑ Aggregations
- ❑ Predicate Queries
- ❑ Locking
- ❑ Eviction



Distributed Maps

Data is distributed throughout the cluster in units called partitions

By default there are partitions

Partitions are divided among cluster members

Partitions are moved back and forth when members enter or leave the cluster to maintain partition balance

Each entry in the map is backed up on another machine

The backup operation can be synchronous (default) or asynchronous



Distributed Map Code Example

```
public class FillMapMember {  
    public static void main(String[] args) {  
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();  
        Map<String, String> map = hz.getMap("map");  
        map.put("1", "Tokyo");  
        map.put("2", "Paris");  
        map.put("3", "New York");  
        System.out.println("Finished loading map");  
        hz.shutdown();  
    }  
}
```



Bootcamp Demo - IMap



Map Eviction



Map Eviction

- Unless explicitly deleted, entries remain in the map
- Eviction prevents the JVM from running out of memory
- Hazelcast supports a multitude of options to fine tune the eviction



Map Eviction

- Hazelcast supports LFU (least frequently used) and LRU (least recently used) eviction policies
- With these algorithms eviction can be triggered based on:
 - Heap used (percentage or absolute value)
 - Maximum entry count (in an entire member or partition)
- Eviction percentage parameter specifies which fraction of entry set is removed when eviction is triggered



Map Eviction

- Eviction can also be triggered based on the following parameters:
 - Time to live
 - Maximum idle time



Map Eviction

```
<hazelcast>
  <map name="default">
    <time-to-live-seconds>10</time-to-live-seconds>
    <max-idle-seconds>10</max-idle-seconds>
    <eviction-policy>LRU</eviction-policy>
    <max-size policy="PER_NODE">50000</max-size>
    <eviction-percentage>25</eviction-percentage>
  </map>
</hazelcast>
```



Map Eviction

Time to live can also be manually assigned to individual entries

```
myMap.put( "1", "John", 50, TimeUnit.SECONDS);
```



Lab 2 - Map Eviction

lab2/src/main/resources

1. Run the Main Class looping 2mb puts
2. See if you now get an OOM.
3. Now add the eviction parameters in hazelcast.xml

```
<map name="persons">  
  <eviction-policy>LRU</eviction-policy>  
  <max-size policy="USED_HEAP_SIZE">20</max-size>  
</map>
```



In Memory Format



In Memory Format

- By default, data is stored in memory in a serialized form
- This is optimal for caching use cases since data is immediately ready to be sent over the wire

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



In Memory Format

- Data can also be stored as plain objects
- This is a better choice for tasks like entry processing since no deserialization is required

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



Map Persistence



Map Persistence

- Hazelcast is often used in conjunction with external databases
- In some use cases, a map needs to be populated from the DB
- In other, updates to map entries must be propagated to the DB
- Hazelcast provides two abstractions to deal with these problems
 - `com.hazelcast.core.MapStore`
 - `com.hazelcast.core.MapLoader`
- Individual maps can have different persistence setups



Map Persistence

- `MapLoader` allows you to fill the map from an external source at startup time
- User has fine control over the values that are loaded at startup
- If a user requests a value that is not present in the map, `MapLoader` will try to find it in external storage



Map Persistence

- `MapStore` is an extension of `MapLoader`. It allows you to read the data from an external storage as well as write to it
- Data can be written to the database in two modes:
 - **Write-through:** Update goes to the DB first and then to the Map
 - **Write-behind:** Update is applied to the Map first and after a certain delay it is propagated to the database



Lab 3 - Map Persistence

lab3/src/main/resources

1. PersonMapStore is a MapStore implementation for reading and saving objects of type "Person" using HSQL DB
2. Run WriteMember to populate DB
3. Run ReadMember to cache miss and read from DB

```
<map name="personMap">  
  <map-store enabled="true">  
    <class-name>PersonMapStore</class-name>  
  </map-store>  
</map>
```



Replicated Map



Replicated Map

- Does not partition data
- Copies Map Entry to every Cluster JVM
- Consider for immutable slow moving data like config
- ReplicatedMap interface supports EntryListeners

```
HazelcastInstance hazelcast = Hazelcast.newHazelcastInstance();
```

```
ReplicatedMap<Integer, String> replicatedMap =  
    hazelcast.getReplicatedMap("replicatedMap");
```

```
replicatedMap.put(1, "London");  
replicatedMap.put(2, "New York");  
replicatedMap.put(3, "Paris");
```



MultiMap



MultiMap

- Similar to implementations found in Google Guava and Apache Commons Collections.
- Allows multiple values to be assigned to the same key
- `MultiMap.get` returns either a Set or a List (configurable)

```
HazelcastInstance hazelcast = Hazelcast.newHazelcastInstance();  
MultiMap<Integer, String> multiMap = hazelcast.getMultiMap("multiMap");
```

```
multiMap.put(1, "London");  
multiMap.put(1, "New York");  
multiMap.put(2, "Paris");
```

```
// "cities" collection will contain London and New York  
Collection<String> cities = multiMap.get(1);
```



Querying



Querying with Predicates

Rich Predicate API provides different methods to find relevant data inside a Map

- Return only the **values** of matching entries

```
Collection<V> IMap.values(Predicate p);
```

- Return only the **keys** of matching entries

```
Set<K> IMap.keySet(Predicate p);
```

- Return entire matching entries

```
Set<Map.Entry<K, V>> IMap.entrySet(Predicate p);
```

- Return only the **keys** of **local** matching entries

```
Set<K> IMap.localKeySet(Predicate p);
```



Querying with Predicates

notEqual

instanceOf

like (%,_)

greaterThan

greaterEqual

lessThan

lessEqual

between

in

isNot

regex



Querying with Predicates

Create your own Predicates

```
/**
 * Predicate instance must be thread-safe.
 * {@link #apply(java.util.Map.Entry)} is called by multiple threads concurrently.
 *
 * @param <K>
 * @param <V>
 */
public interface Predicate<K, V> extends Serializable {

    boolean apply(Map.Entry<K, V> mapEntry);

}
```



Lab 4 - Predicates

lab4

Populate a Map with six people that have the following properties:

“*age*” (integer) and “*active*” (boolean)

Perform various queries using Predicates

Use a PredicateBuilder to create a compound predicate



SQLPredicate

- `com.hazelcast.query.SqlPredicate` allows the user to create queries with SQL expressions
- Can only be ran against values
- SQL expression is converted to a set of regular predicates prior to execution

```
HazelcastInstance hazelcast = Hazelcast.newHazelcastInstance();
IMap<String, Customer> customers = hazelcast.getMap("customers");
customers.put("1", new Customer("Peter", true, 36));
customers.put("2", new Customer("John", false, 40));
customers.put("3", new Customer("Roger", true, 20));
```

```
//Will contain Roger
```

```
SqlPredicate predicate = new SqlPredicate("active AND age < 30");
Collection<Customer> result = customers.values(predicate);
```



Indexes

- Prevent full Map scans
- Can be ordered or unordered
- Can work along the object graph (x.y.z)
- Indexed objects must implement `Comparable`
- Can be created at runtime

```
<map name="personMap">  
  <indexes>  
    <index ordered="false">name</index>  
  </indexes>  
</map>
```



Lab 5 - Indexes

lab5

Run the Index Benchmark that compares Indexed and non Indexed queries



EntryListener



EntryListener API - Overview

In some applications, user might need to observe the changes to the data stored inside the Map. This can be achieved by using one of the following interfaces:

- `com.hazelcast.map.listener.EntryAddedListener`
- `com.hazelcast.map.listener.EntryUpdatedListener`
- `com.hazelcast.map.listener.EntryRemovedListener`
- `com.hazelcast.map.listener.EntryEvictedListener`
- `com.hazelcast.map.listener.MapClearedListener`
- `com.hazelcast.map.listener.MapEvictedListener`
- `com.hazelcast.core.EntryListener` - **extends all of the above**

Additionally, there is a class `com.hazelcast.core.EntryAdapter`, which implements `EntryListener` and allows the user to override only the relevant methods



EntryListener API - Configuration

Entry listeners are configured individually for each map. Options include:

- Local (boolean) - invoke only for entry events on the registering member
- Include value (boolean) - include the value into the event object

```
Config config = new Config();
EntryListenerConfig listenerConfig = new EntryListenerConfig();
listenerConfig.setIncludeValue(true)
               .setLocal(false)
               .setClassName(org.company.Listener.class.getCanonicalName());
config.getMapConfig("customers").addEntryListenerConfig(listenerConfig);
```

```
<map name="orders">
  <entry-listeners>
    <entry-listener include-value="false" local="true">org.company.Listener</entry-listener>
  </entry-listeners>
</map>
```



EntryListener Example

Note that events on the same key are guaranteed to be processed in the same order as they were generated

```
public class MapListener extends EntryAdapter<String, String> {  
    @Override  
    public void entryAdded(EntryEvent<String, String> event) {  
        System.out.println("New entry with key " + event.getKey());  
    }  
  
    @Override  
    public void entryUpdated(EntryEvent<String, String> event) {  
        System.out.printf("Entry with key " + event.getKey() + " was updated");  
    }  
  
    @Override  
    public void entryRemoved(EntryEvent<String, String> event) {  
        System.out.println("Entry with key " + event.getKey() + " was removed");  
    }  
}
```



Data Distribution and Recovery



Learning Objectives

In this module you will learn how Hazelcast:

Distributes the load between the members

Ensures data safety in case of a node crash



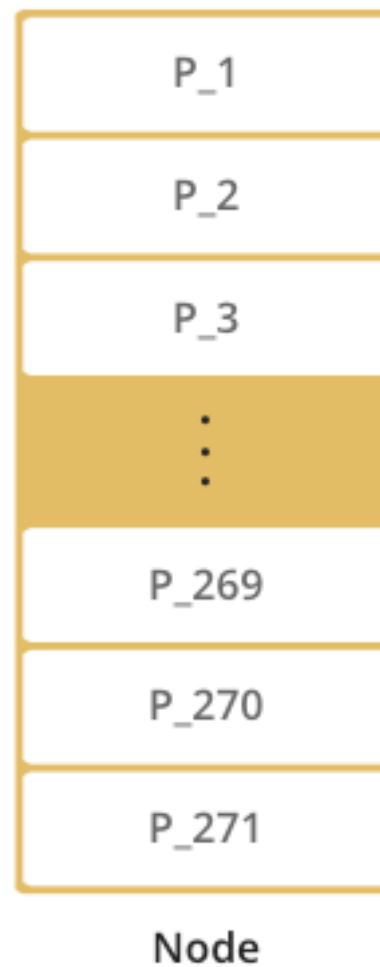
Replication vs Partitioning

Replication - Copying an entire dataset onto multiple servers. Used for improving speed of access to reference records such as master data.

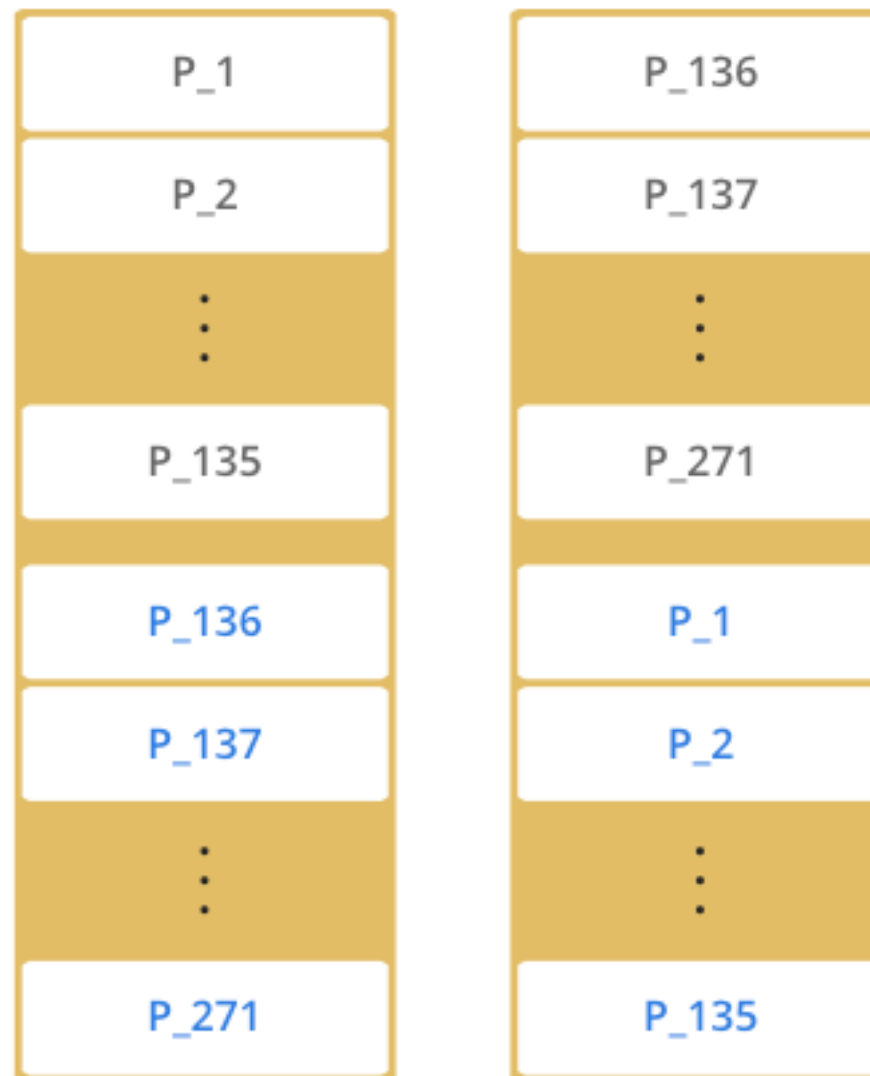
Partitioning - Splitting up a large monolithic dataset into multiple smaller sets based on data cohesion.



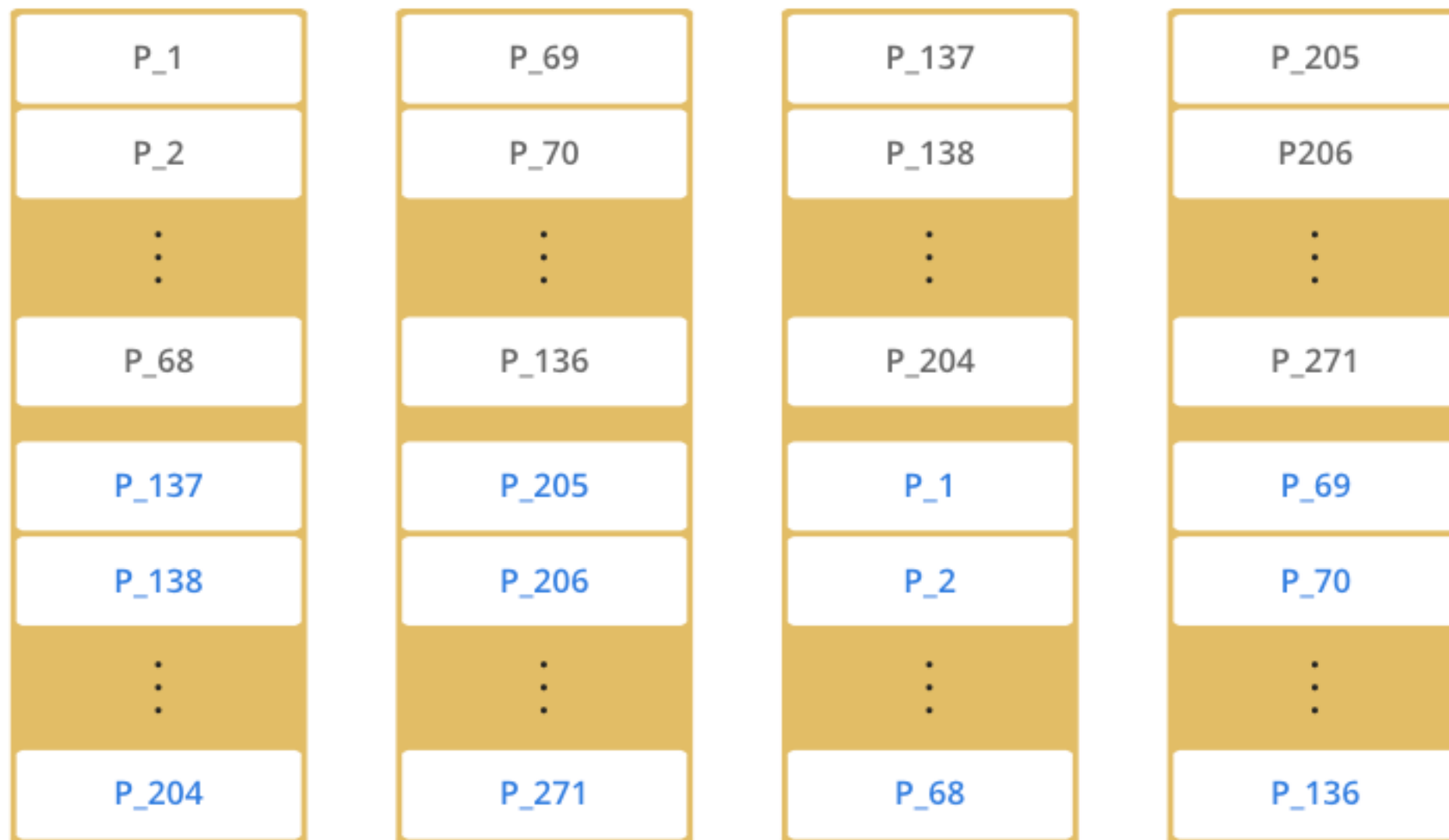
Data Partitioning



Data Partitioning



Data Partitioning





Data Distribution and Resilience

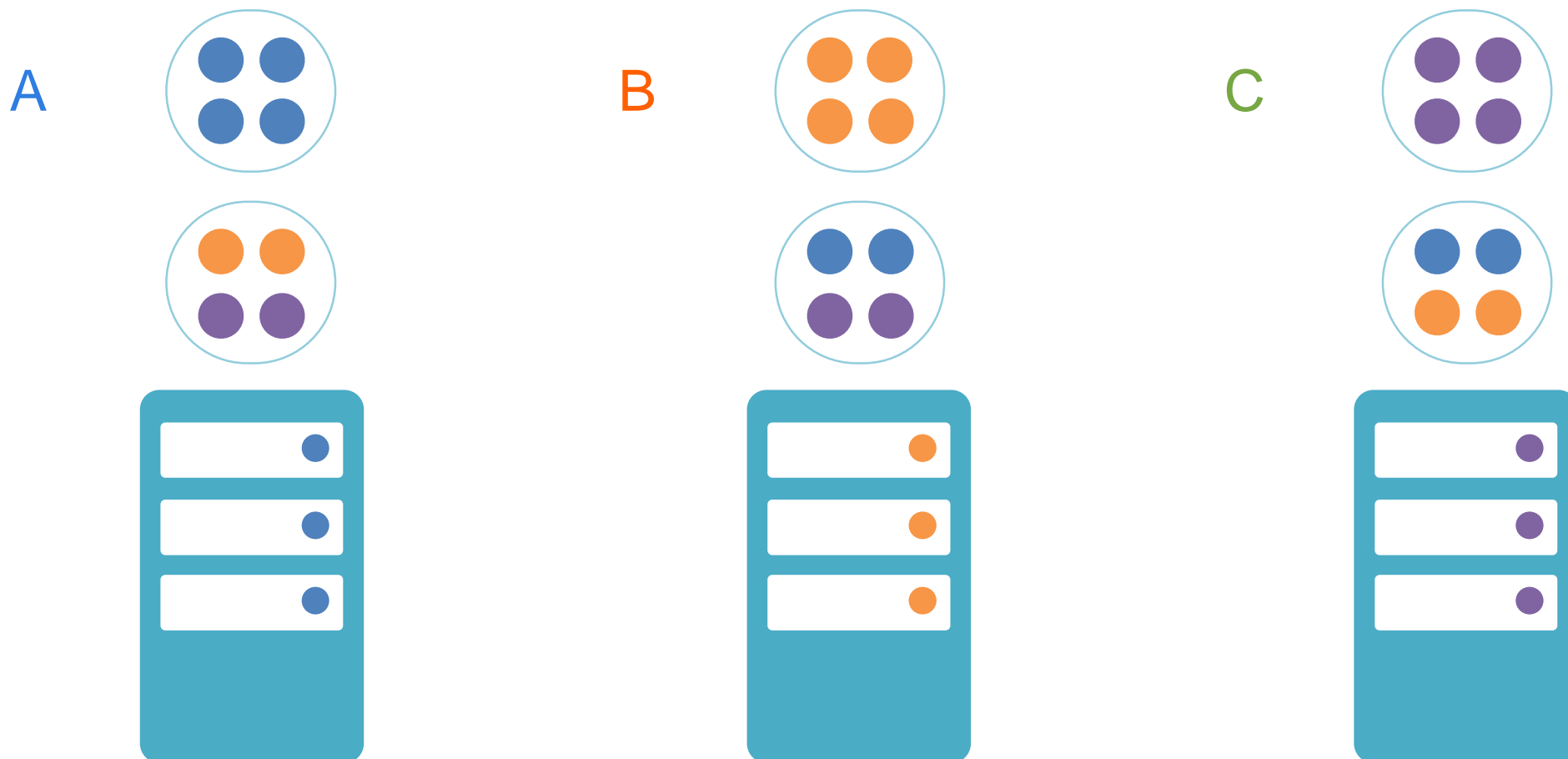
Distributed Maps

Fixed number of partitions (default 271)

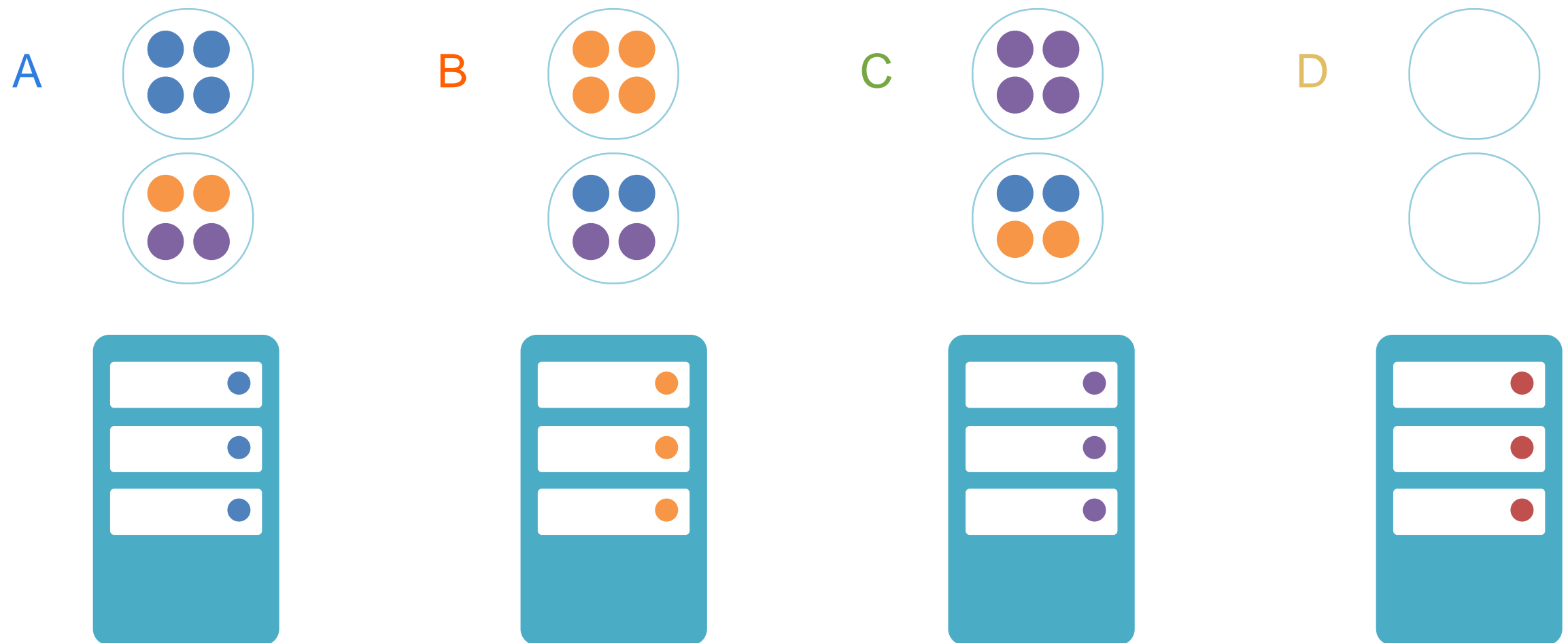
Each key falls into a partition

$partitionId = hash(keyData) \% PARTITION_COUNT$

Partition ownerships are reassigned upon membership change

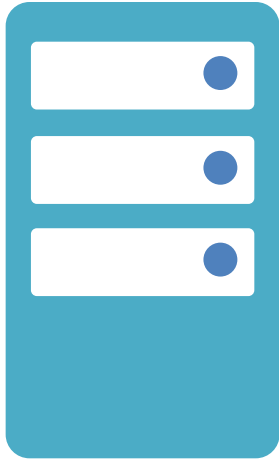
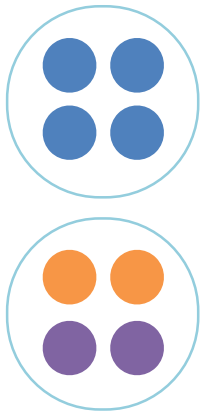


New Node Added

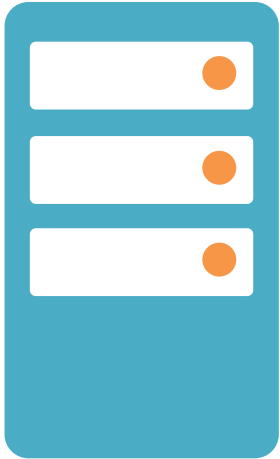
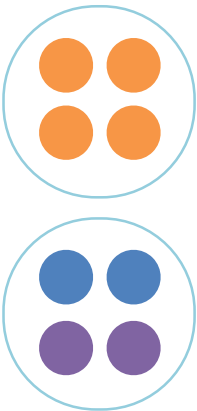


Migration

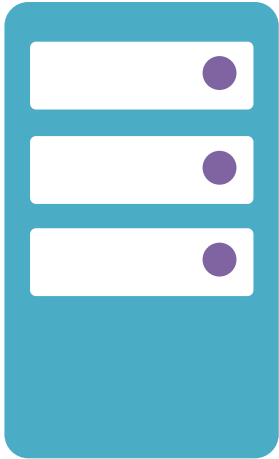
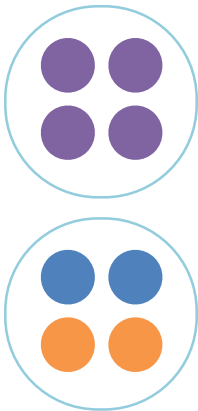
A



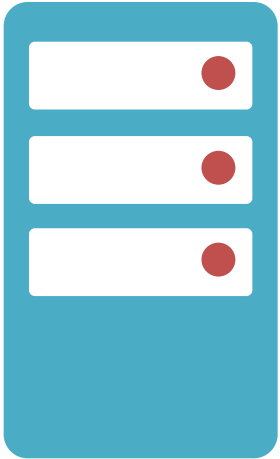
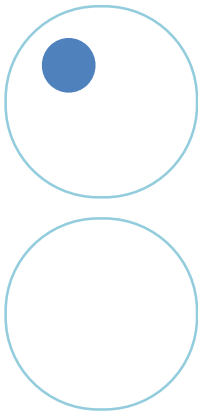
B



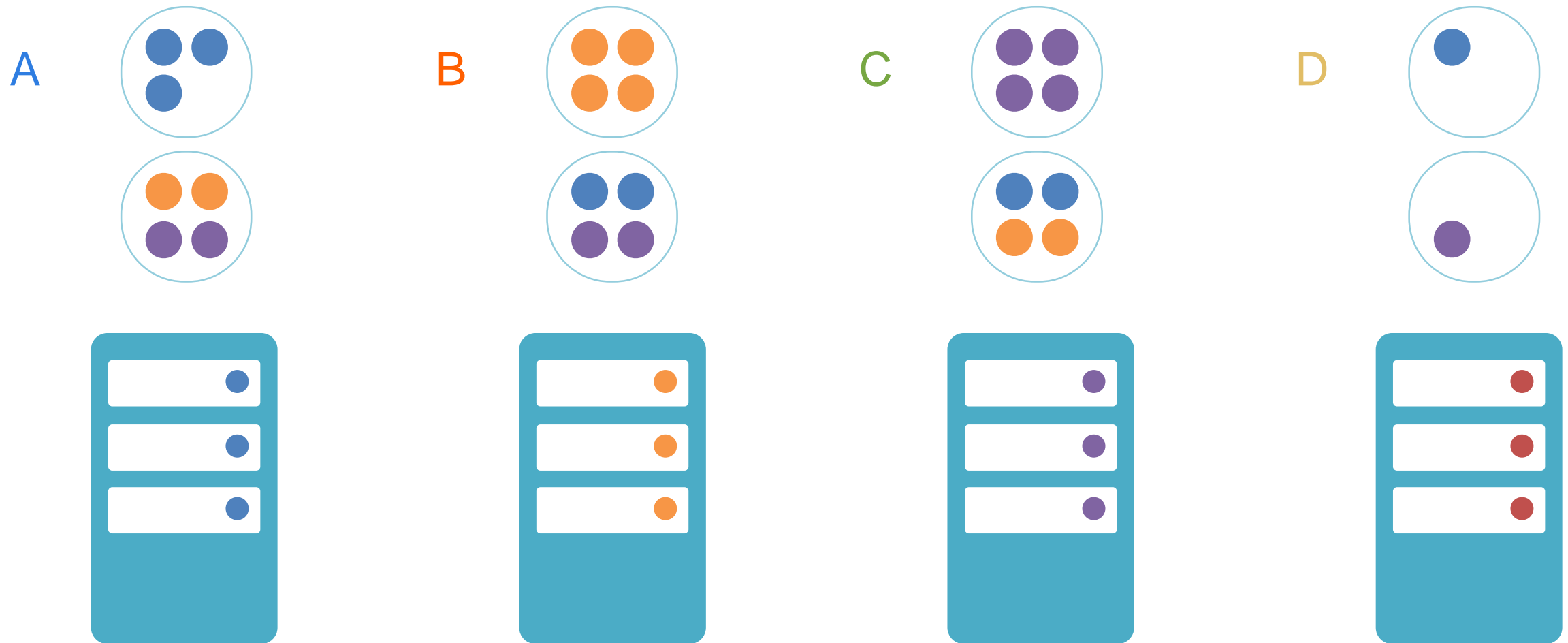
C



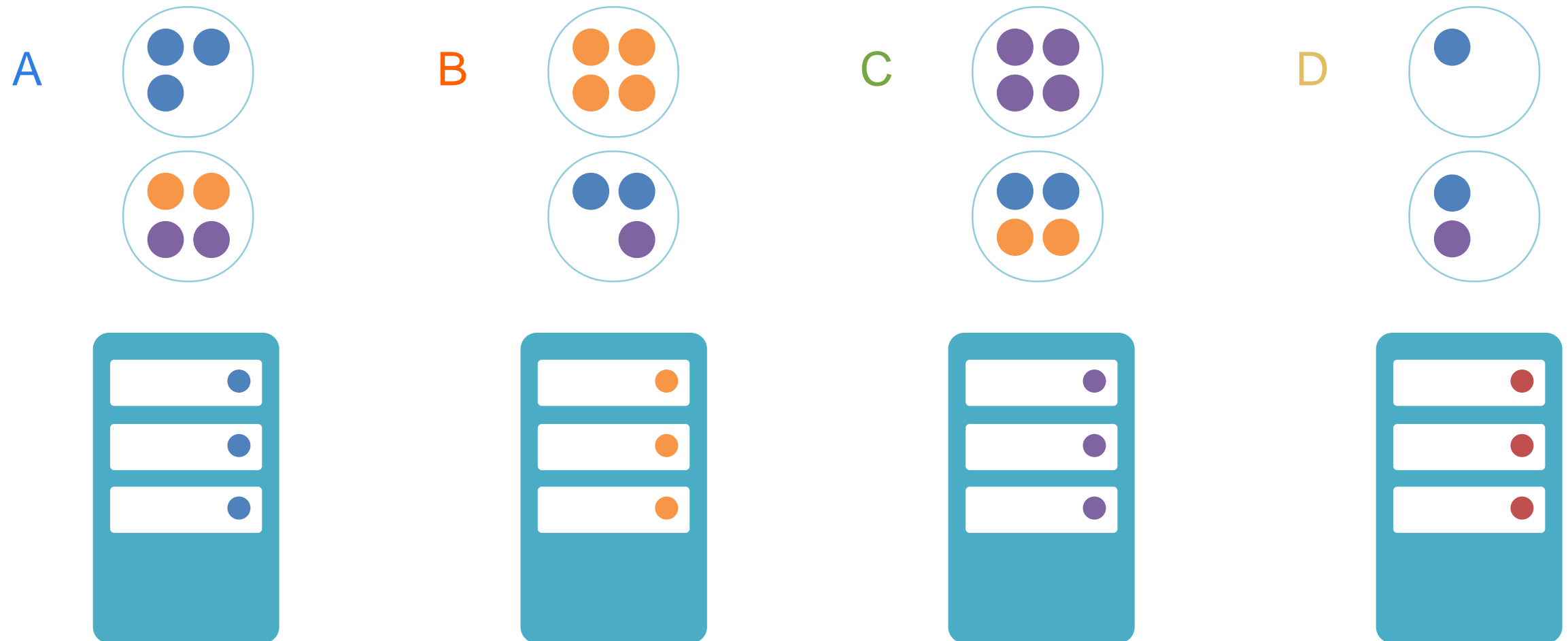
D



Migration

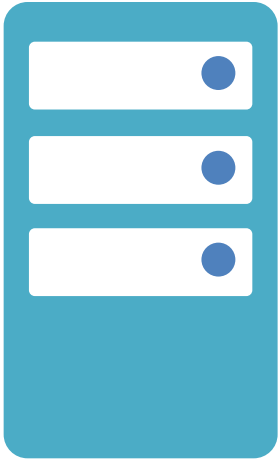
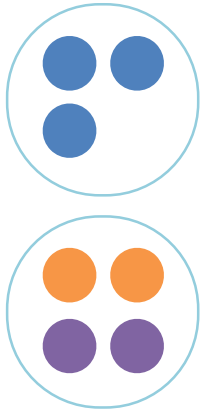


Migration

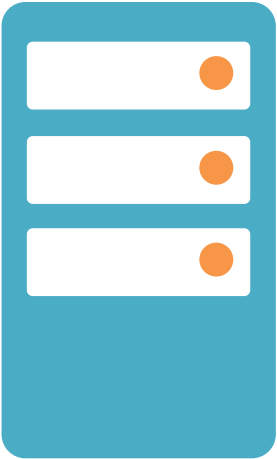
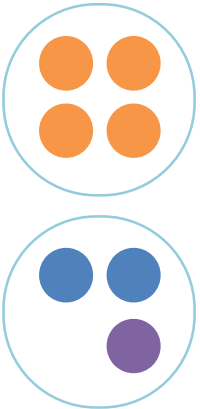


Migration

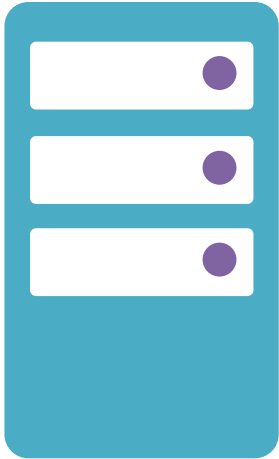
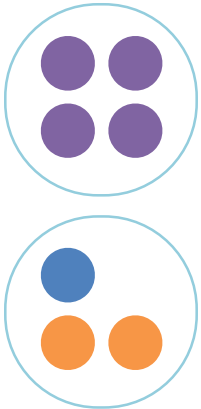
A



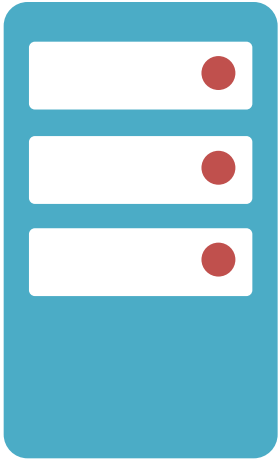
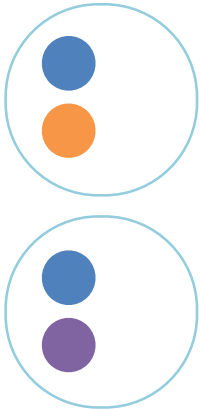
B



C

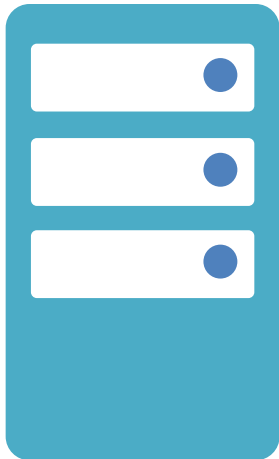
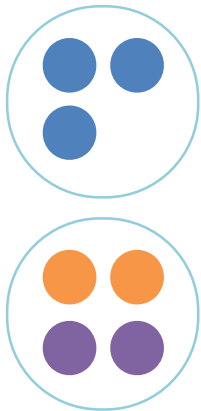


D

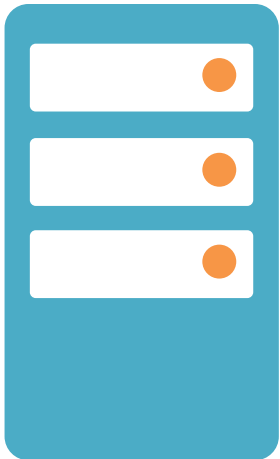
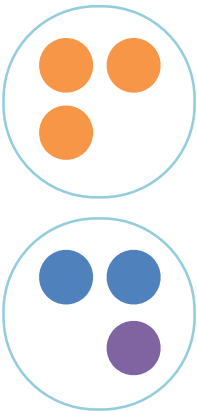


Migration

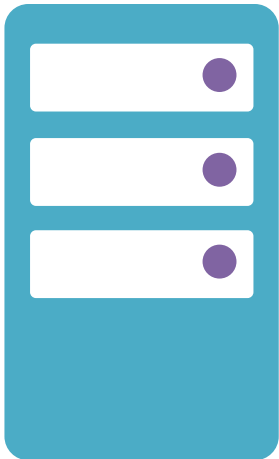
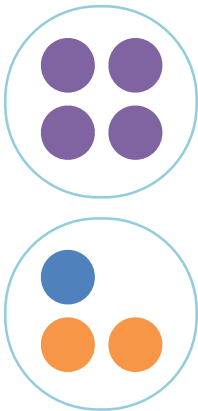
A



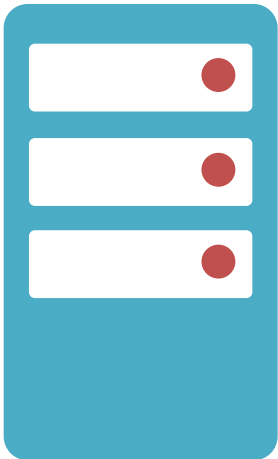
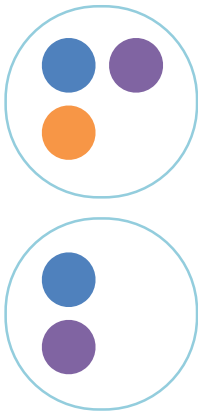
B



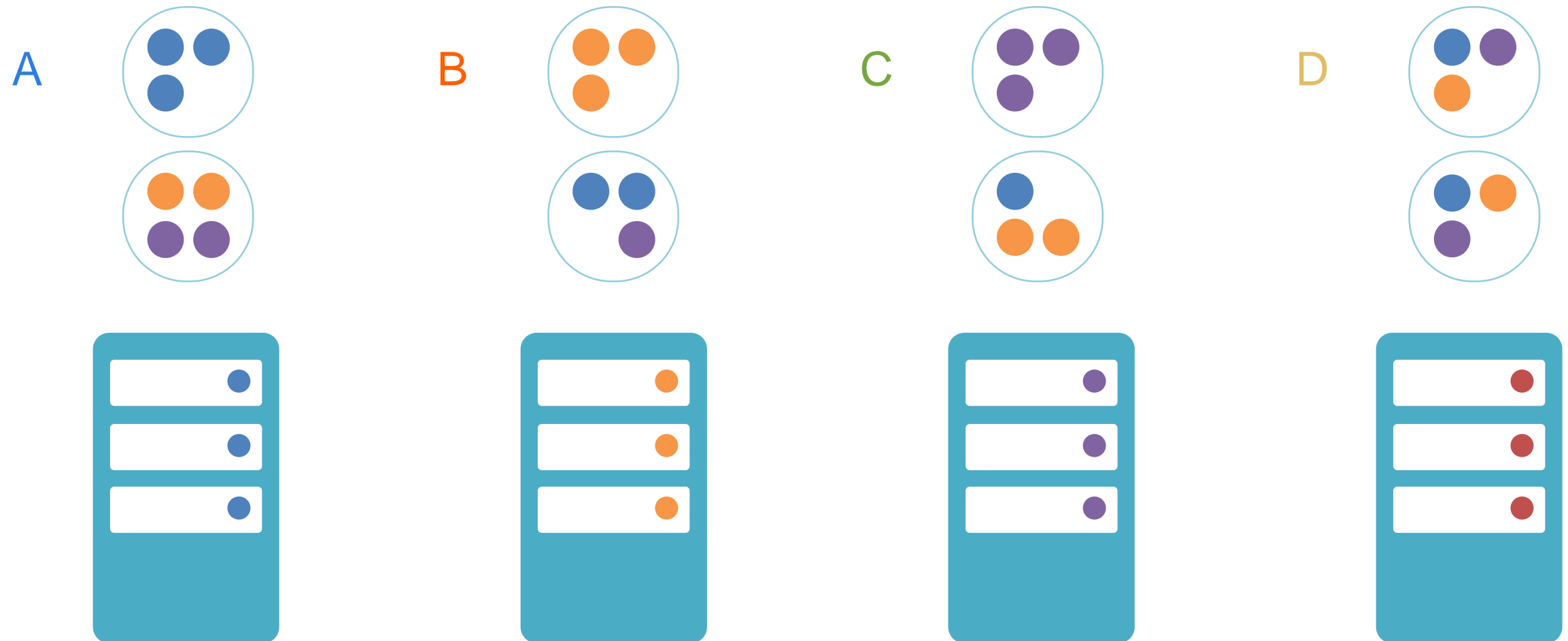
C



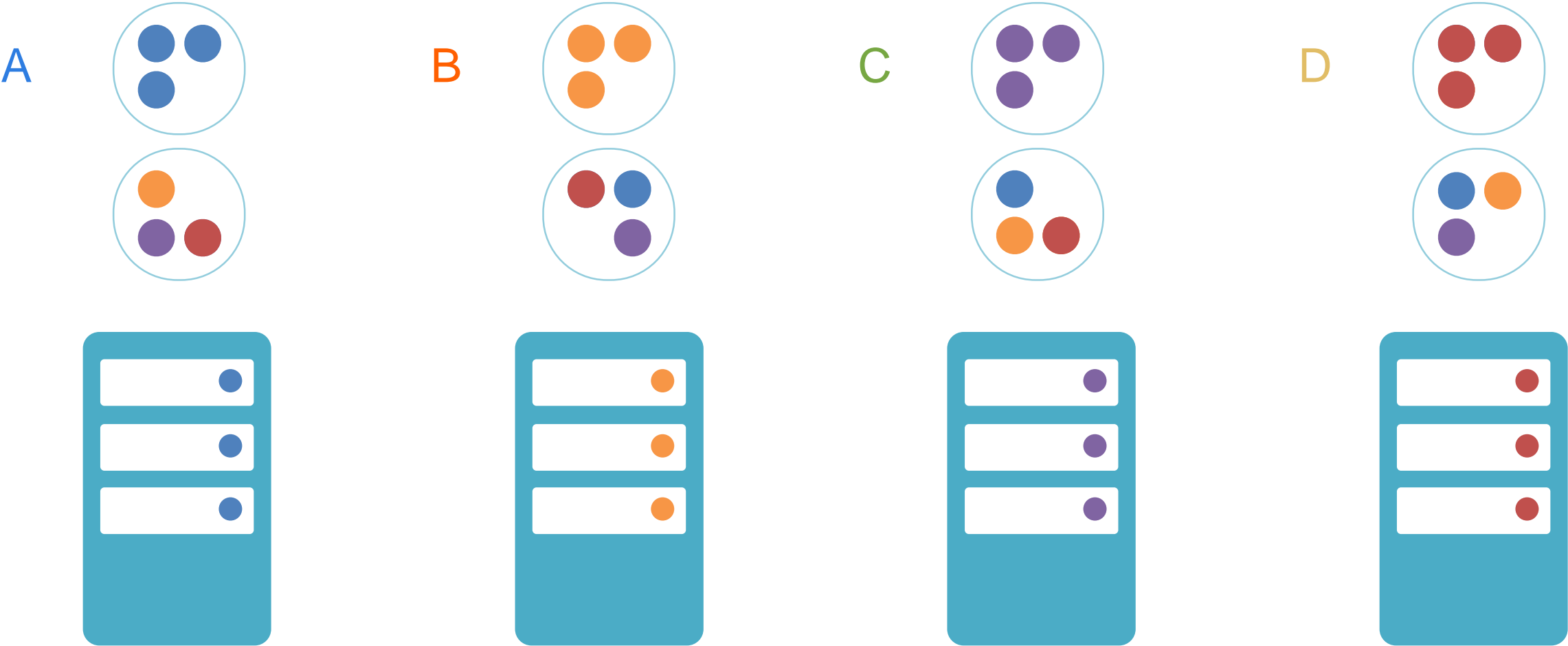
D



Migration



Migration Complete

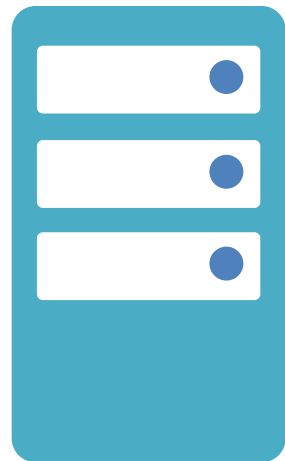
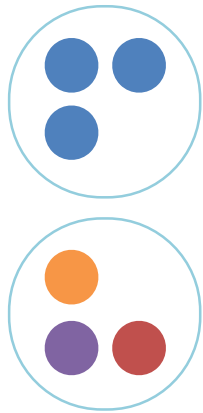




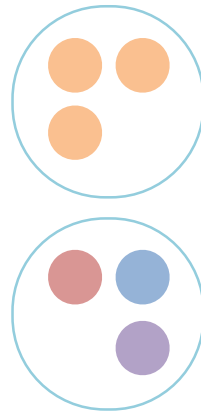
Data Safety on Node Failure

Node Crashes

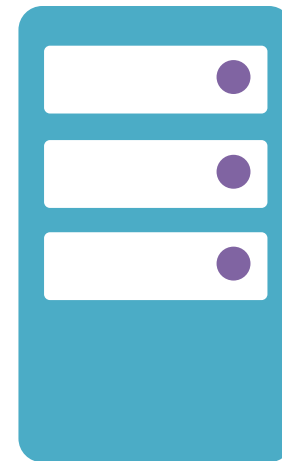
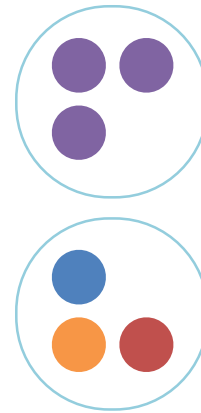
A



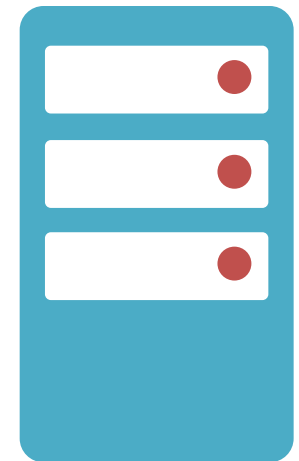
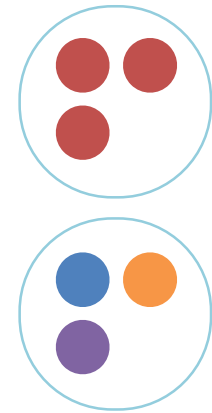
B



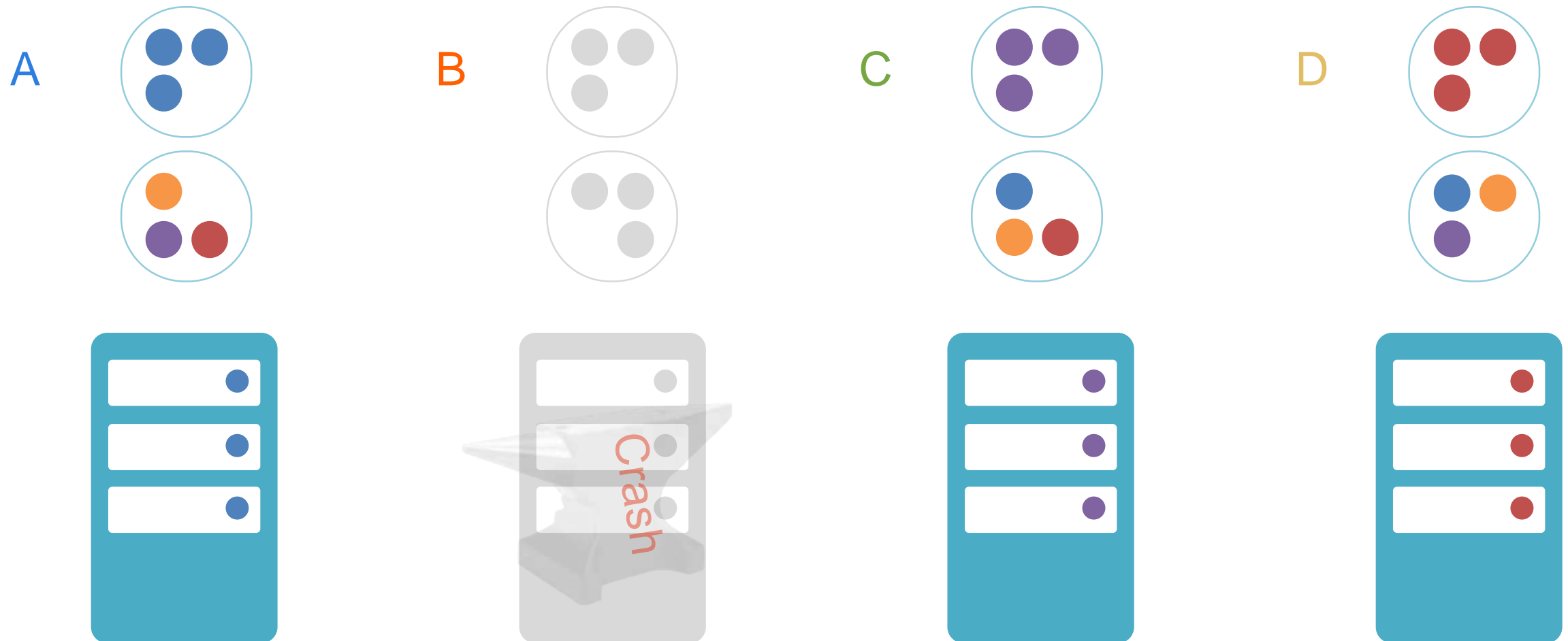
C



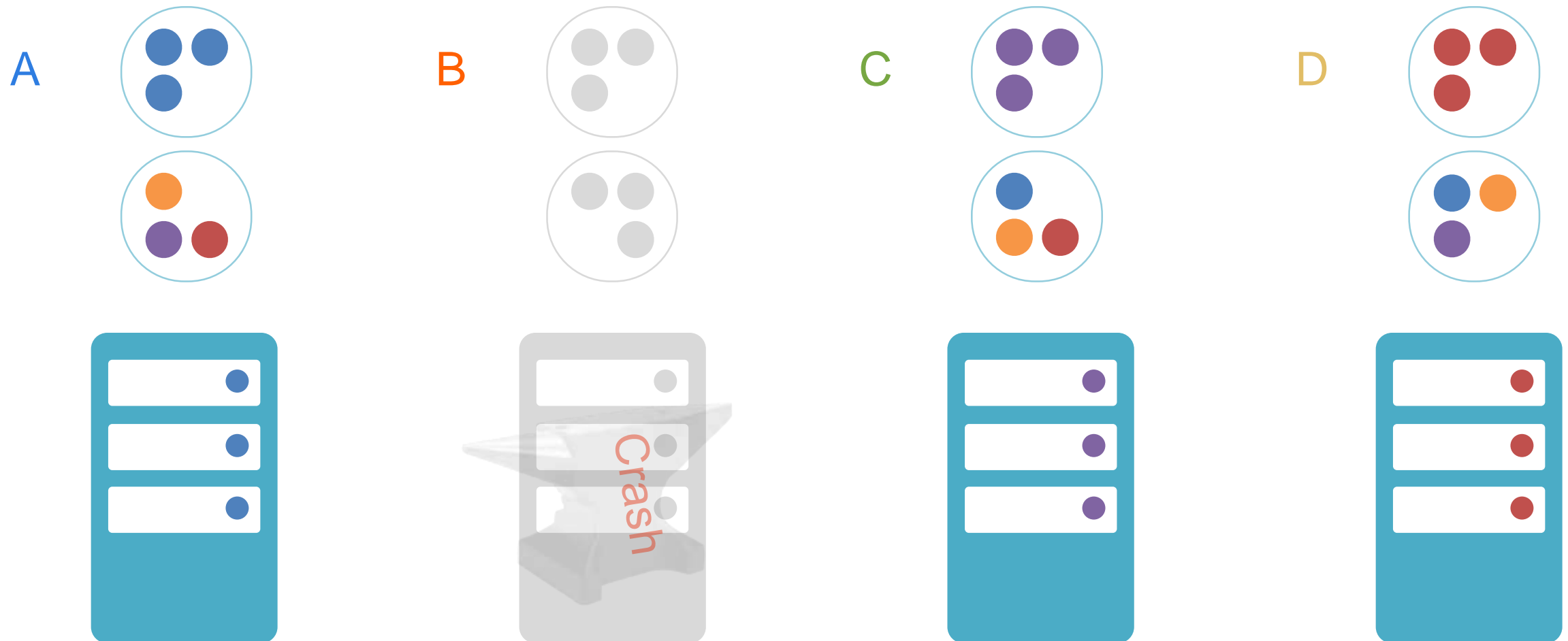
D



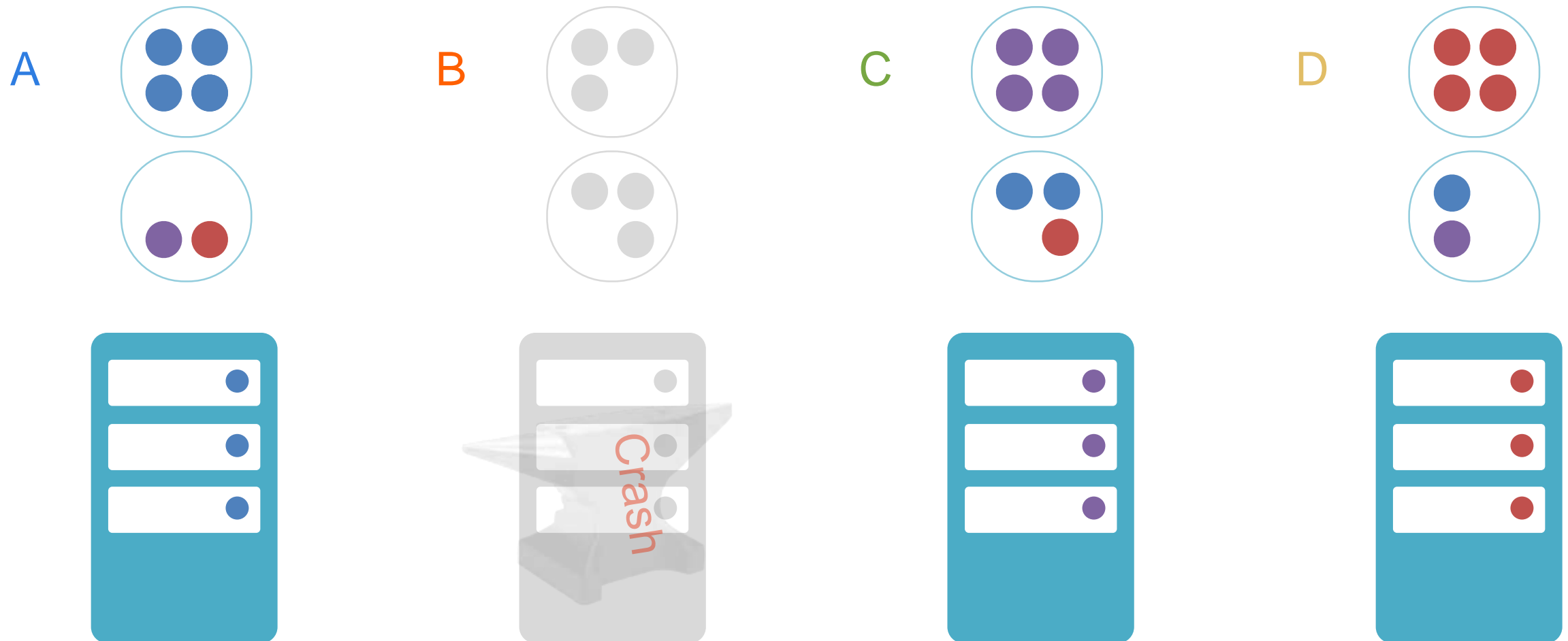
Backups Are Restored



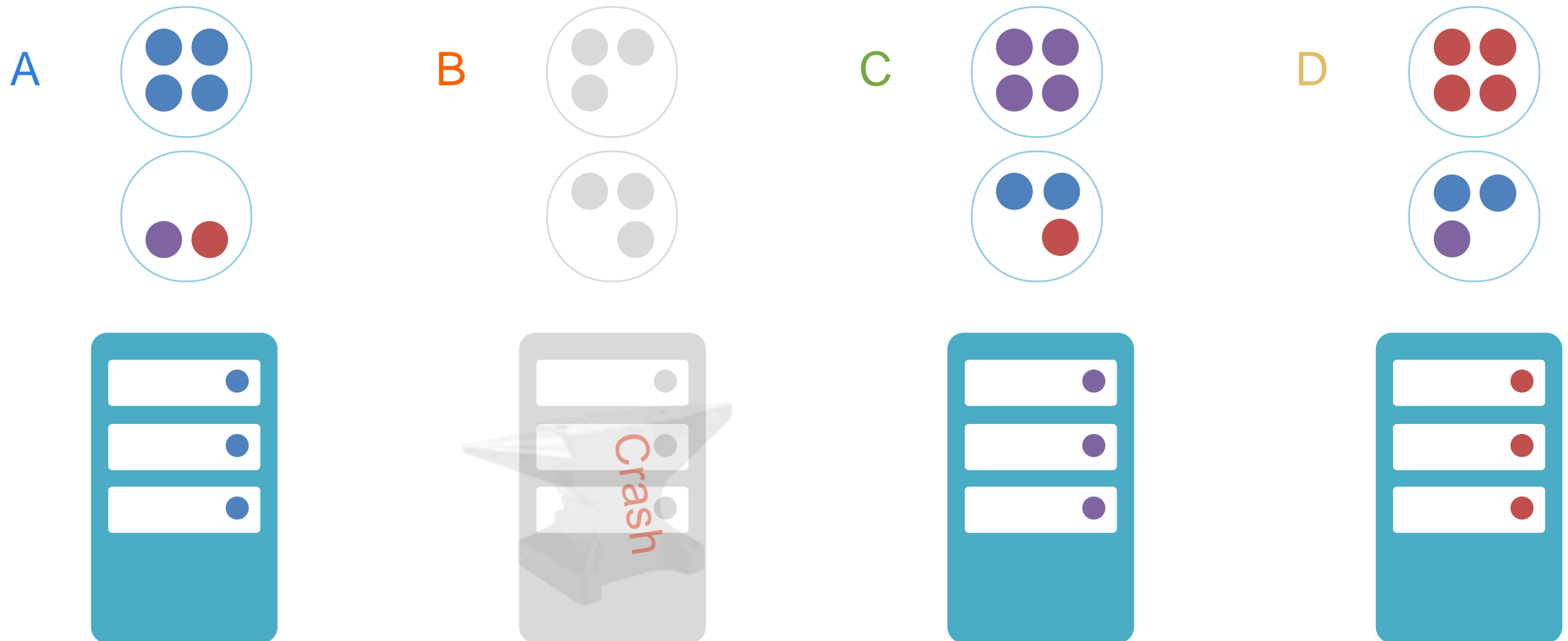
Backups Are Restored



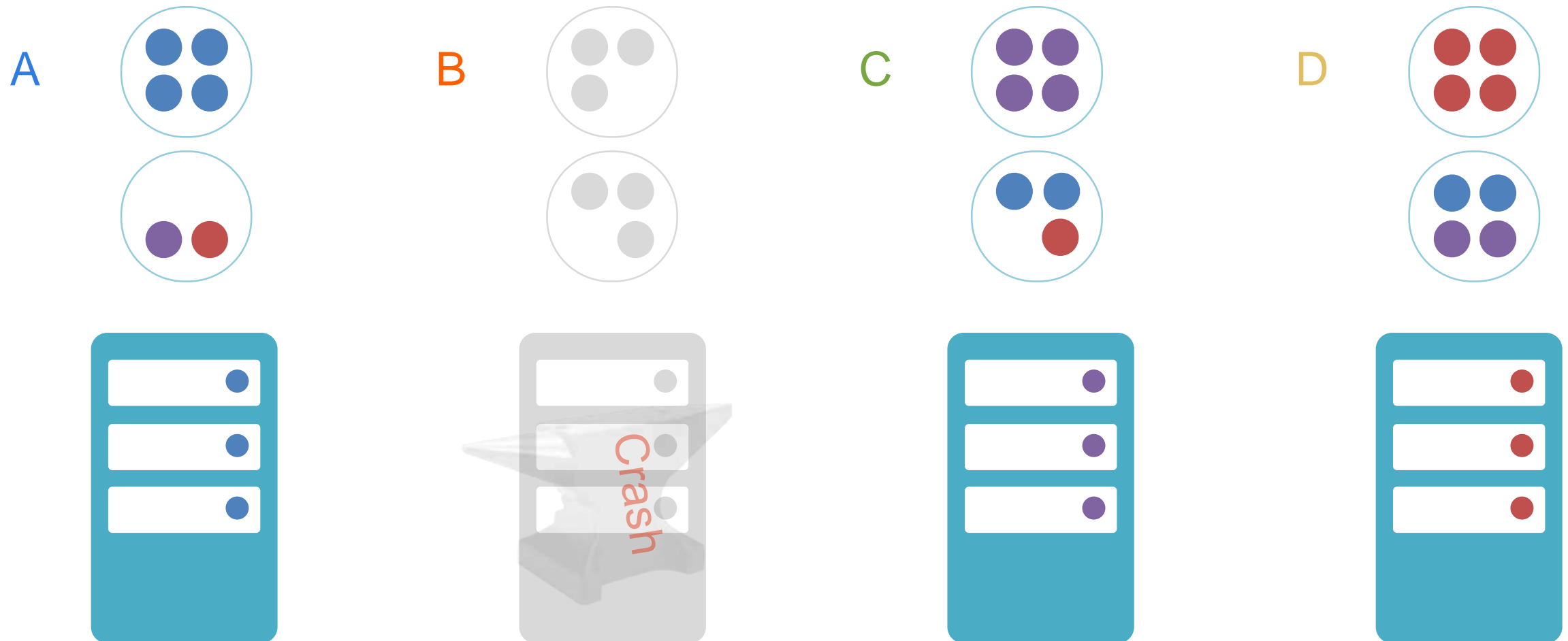
Backups Are Restored



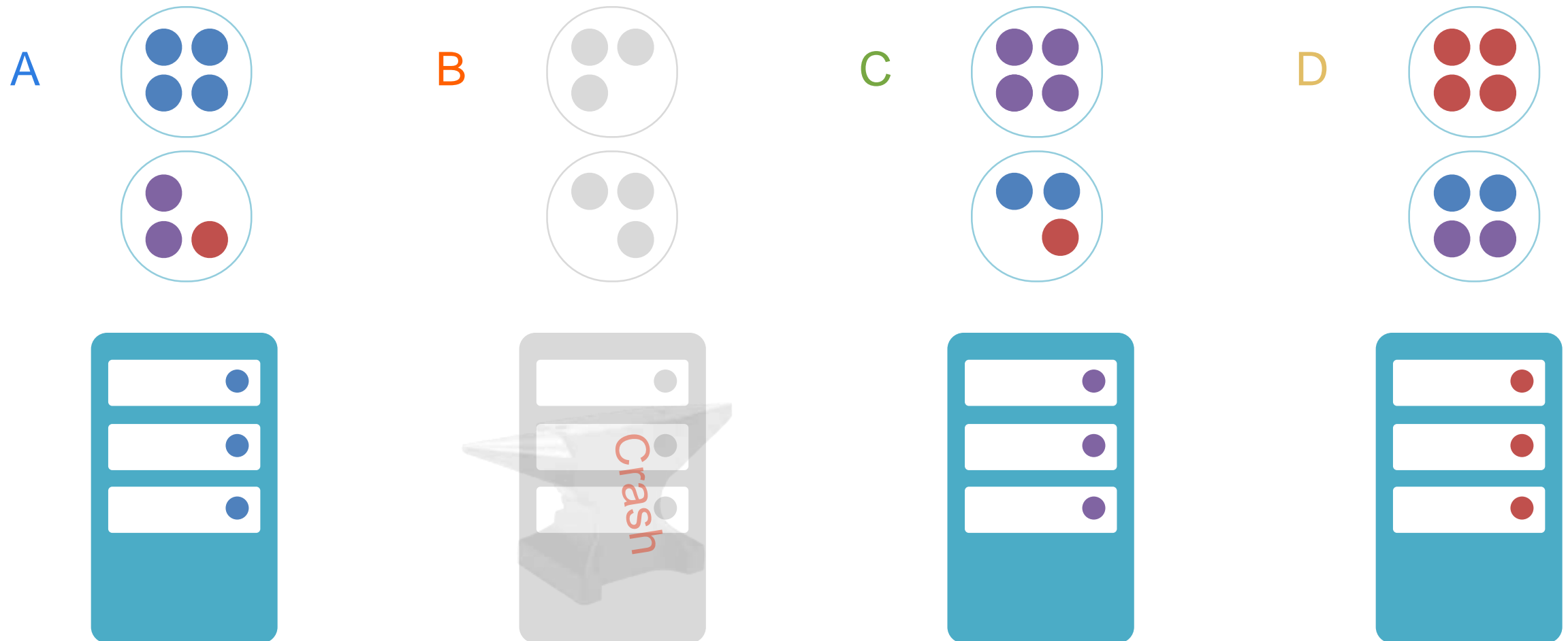
Backups Are Restored



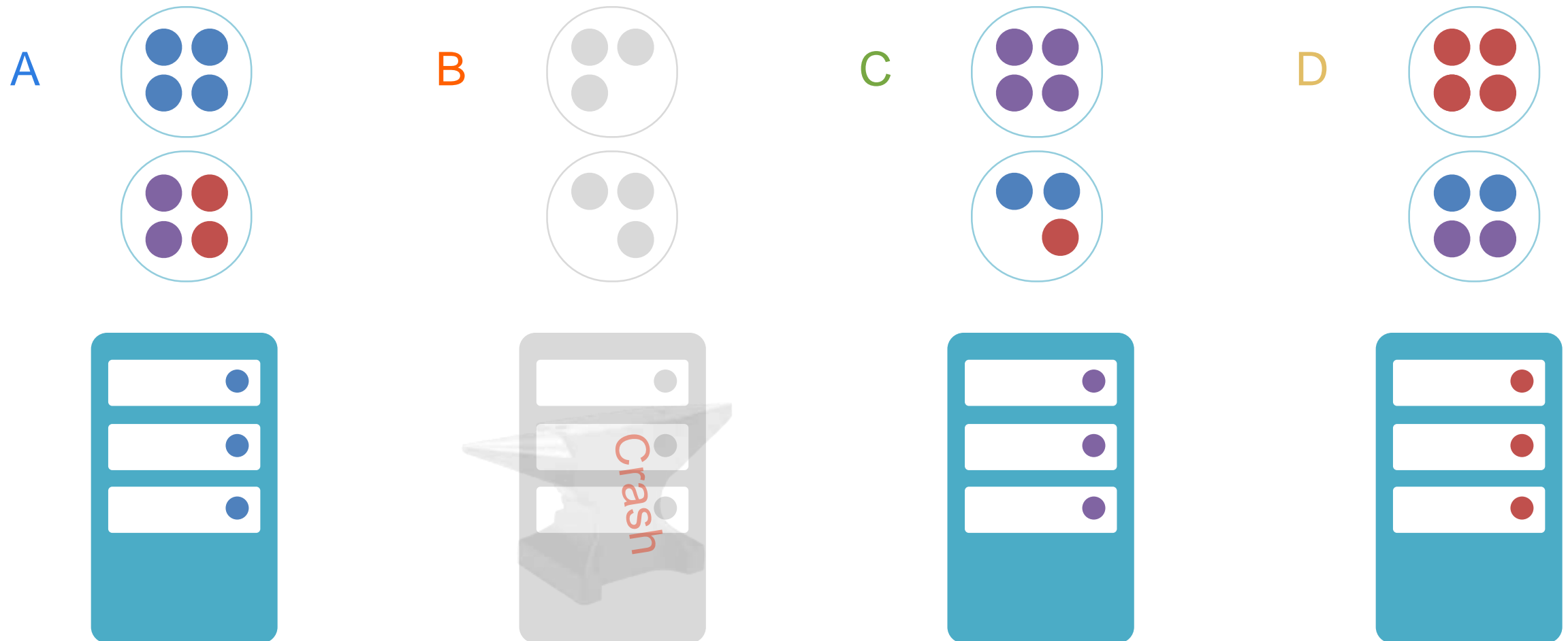
Backups Are Restored



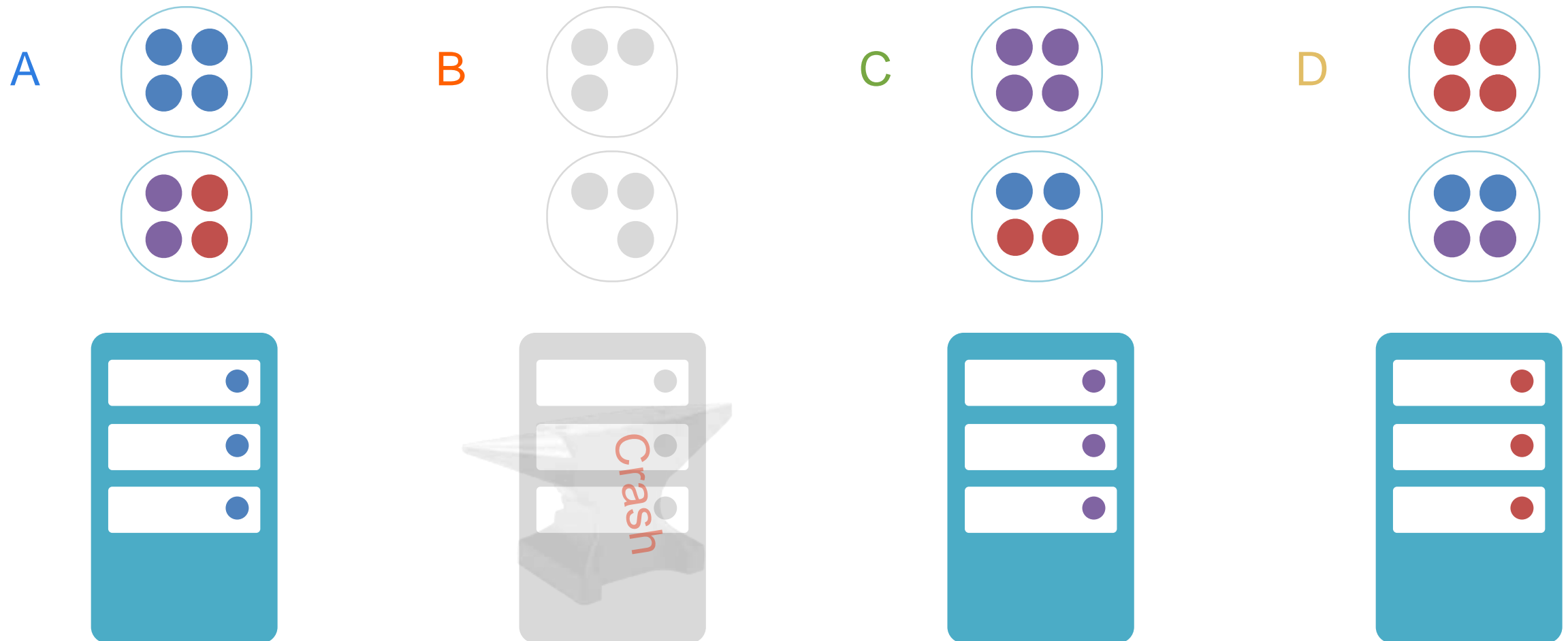
Backups Are Restored



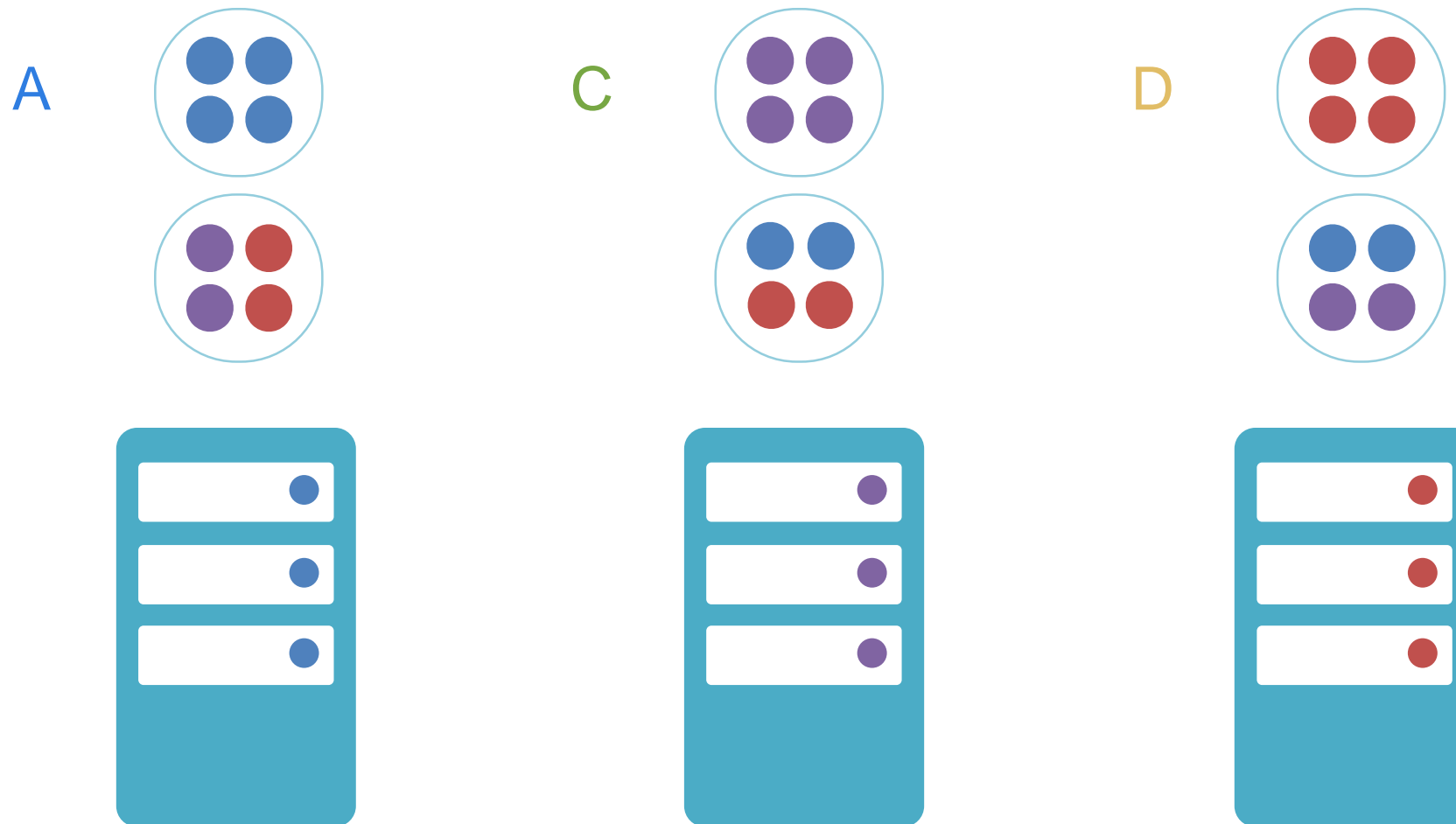
Backups Are Restored



Backups Are Restored



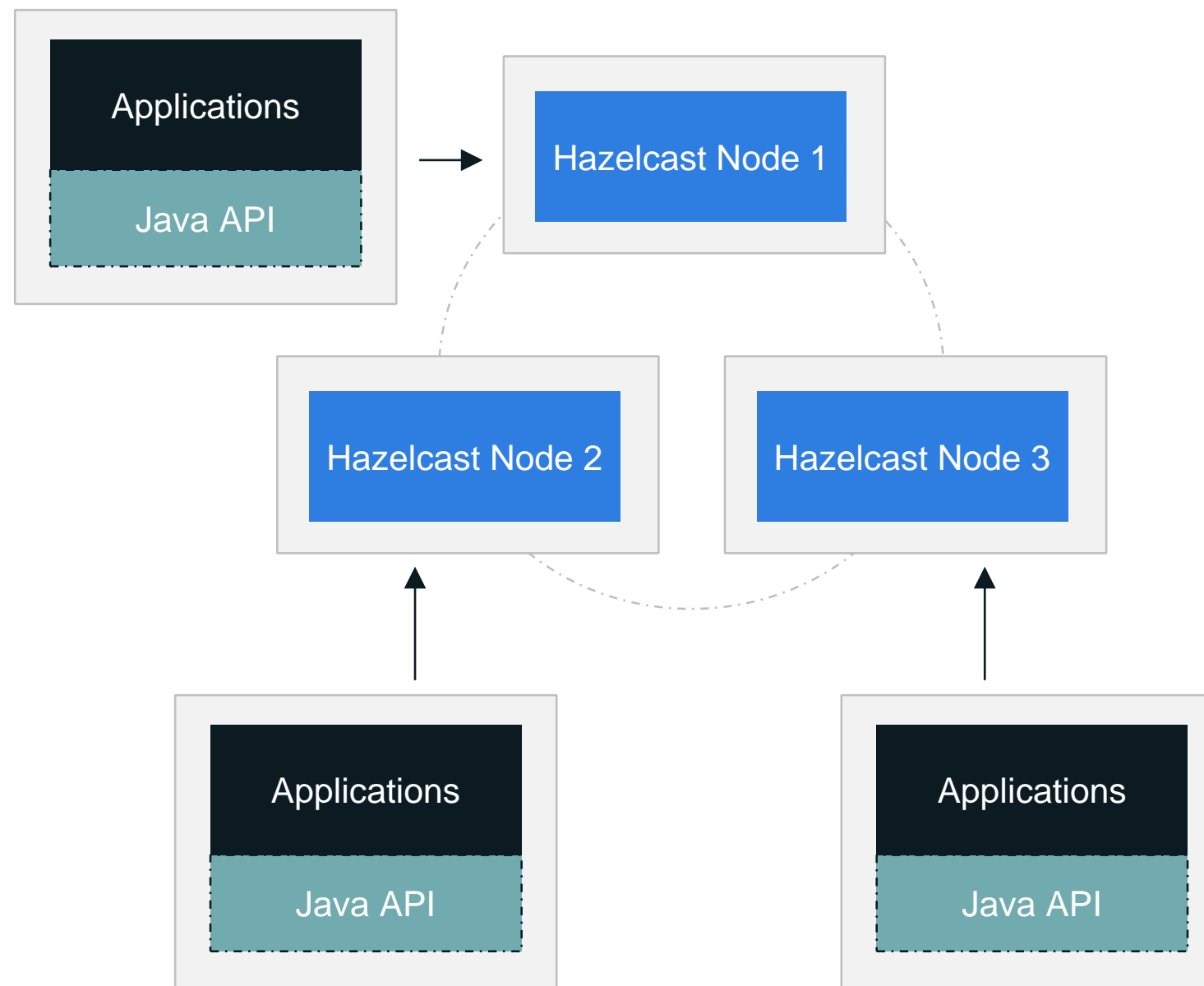
Recovery Is Complete



Deployment Architectures aka Topologies



Client-Server



Necessary for scale up or scale out deployments.
Decouples upgrading of clients and cluster for long term TCO.



Client-Server

Follows the traditional Client-Server approach

Clients do not change cluster topology

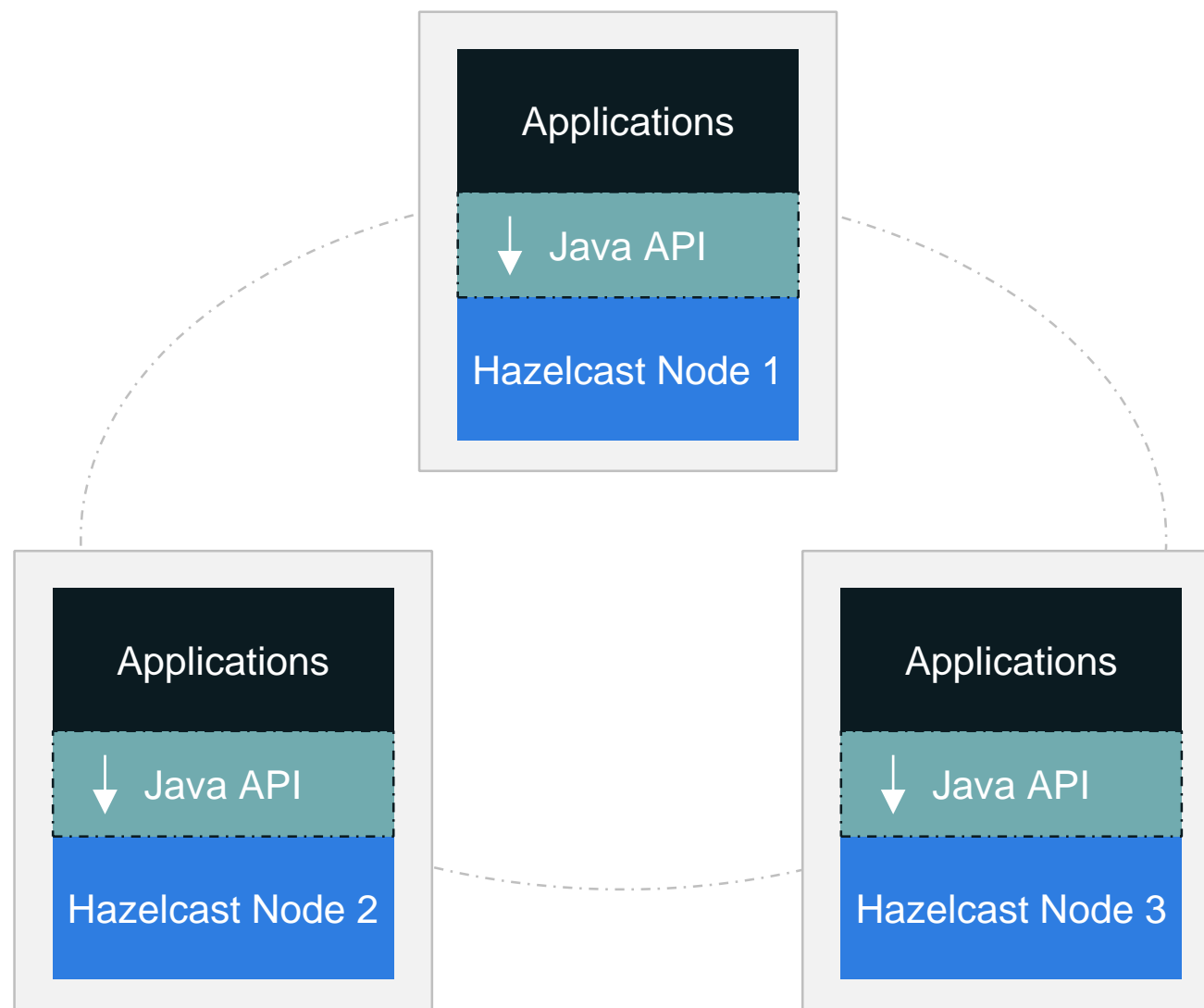
Allows you to segregate service from storage

Regular clients connect to a single cluster member and use it as a proxy to issue requests

Smart clients connect to all cluster members. Operations are routed directly to a member holding the necessary data



Embedded Hazelcast



Great for early stages of rapid application development and iteration

Topologies

Embedded model, for example in a J2EE container
Service and storage within one JVM
Client API same as Server API - HazelcastInstance.
Clients in Java, C#, C++, Memcache, REST



Serialization



Learning Objectives

In this module you will learn more about:

Serialization basics

Pros and cons of different serialization methods

Integration with third-party serialization libraries

Serialization configuration



What is Serialization?

Serialization: conversion of a Java object into a binary form

Deserialization: resurrection of a Java object from a binary form

When is it used? All the time!

Creation of an entry in a Map

Submission of a task to an `ExecutorService`

Transformation of Map contents with `EntryProcessor`

Publishing messages to a Topic

Other

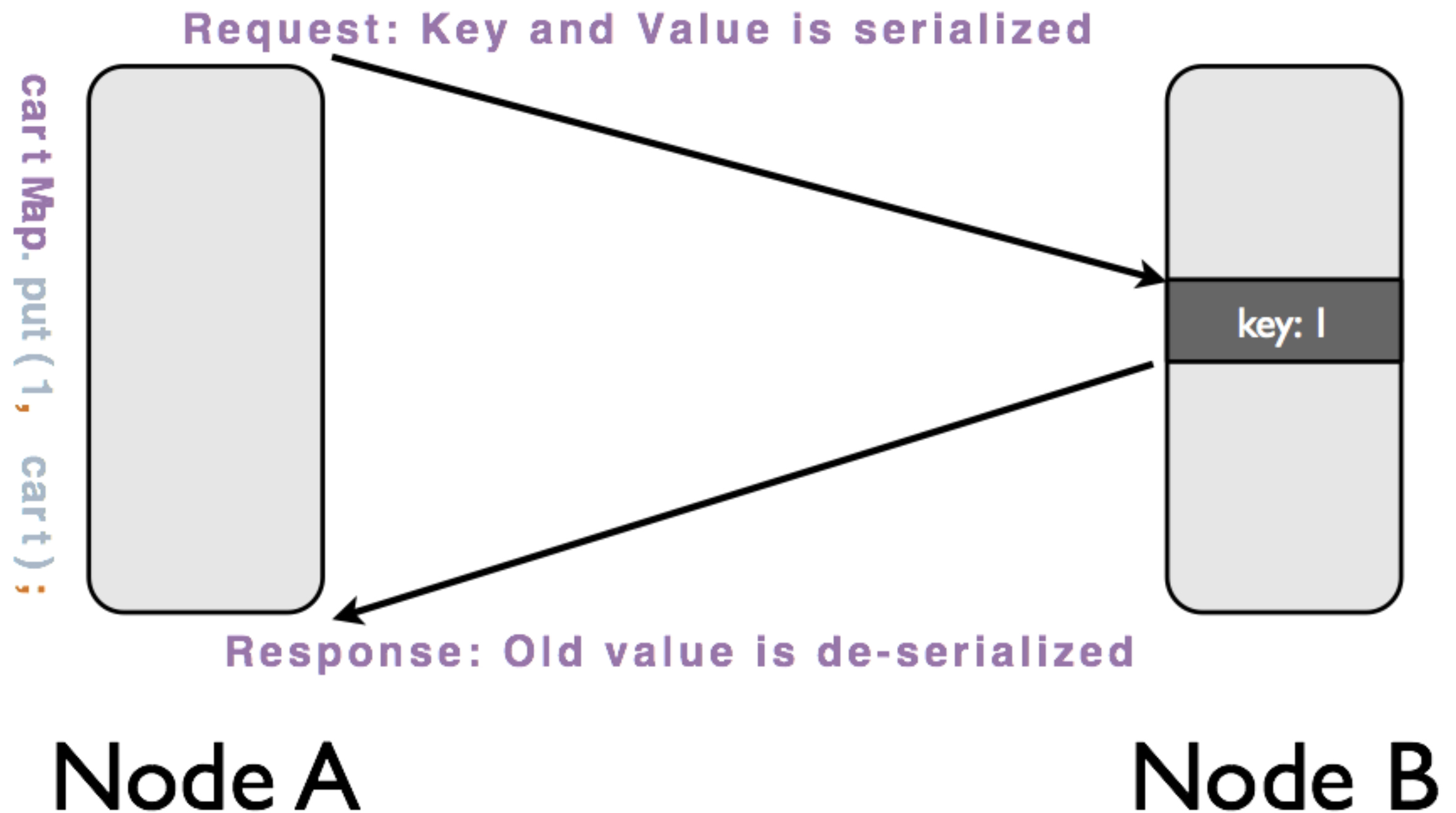


Example

```
Object oldItem = cartMap.put(id, item); //    serialization +  
                                         //    de-serialization  
taskQueue.offer(task); // serialization  
iLock.lock(accountId); // serialization  
executor.execute(new MyRunnable()); // (de-)serialization  
topic.publish(myMessageObject); // serialization  
Object item = iMap.get(id); // de-serialization
```



Put Operation Serialization Cycle



Optimized types

Serialization of these types are optimized in Hazelcast

Byte	byte[]	String
Boolean	char[]	Date
Character	short[]	BigInteger
Short	int[]	BigDecimal
Integer	long[]	Class
Long	float[]	Enum
Float	double[]	
Double		



Serialization Types

- **Standard Java Serialization**

- `java.io.Serializable`
- `java.io.Externalizable`

- **Hazelcast specific**

- `com.hazelcast.nio.serialization.DataSerializable`
- `com.hazelcast.nio.serialization.IdentifiedDataSerializable`
- `com.hazelcast.nio.serialization.Portable`

- **Custom (Pluggable) Serialization via**




- `com.hazelcast.nio.serialization.ByteArraySerializer`
- `com.hazelcast.nio.serialization.StreamSerializer`



Sample Domain – Shopping Cart

Shopping Cart

Items to buy now

		Price	Quantity
	Total Immersion: The Revolutionary Way To Swim Better, Faster, and Easier - Terry Laughlin; Paperback Prime In Stock <input type="checkbox"/> This is a gift (Learn more) Delete · Save for later	\$9.85 You save: \$7.14 (42%)	<input type="text" value="1"/>
	Asics Asics Men's Gel-Kayano 19 Running Shoe (13 D(M) Us, Charcoal/Sunburst/Fl) - ASICS Prime Only 1 left in stock. <input type="checkbox"/> This is a gift (Learn more) Delete · Save for later	\$109.95 You save: \$40.05 (27%)	<input type="text" value="1"/>
	Kindle Fire HDX 7", HDX Display, Wi-Fi, 16 GB - Includes Special Offers - Amazon Prime In Stock <input type="checkbox"/> This is a gift (Learn more) Delete · Save for later	\$199.00 You save: \$30.00 (13%)	<input type="text" value="1"/>
		Subtotal: \$318.80	



java.io.Serializable



java.io.Serializable

Pros:

Standard

Works out of the box without custom code

Cons:

Least efficient in terms of CPU load

Biggest resulting payload



ShoppingCartItem class

```
public class ShoppingCartItem implements Serializable {  
    public long cost;  
    public int quantity;  
    public String itemName;  
    public boolean inStock;  
    public String url;  
}
```



ShoppingCart class

```
public class ShoppingCart implements Serializable {  
    public long total = 0;  
    public Date date;  
    public long id;  
    private List<ShoppingCartItem> items = new ArrayList<>();  
  
    public void addItem(ShoppingCartItem item) {  
        items.add(item);  
        total += item.cost * item.quantity;  
    }  
  
    public void removeItem(int index) {  
        ShoppingCartItem item = items.remove(index);  
        total -= item.cost * item.quantity;  
    }  
  
    public int size() {  
        return items.size();  
    }  
}
```



Benchmark – 100,000 times

```
@Override
public void writePerformance() {
    Random random = new Random();
    for (int k = 0; k < OPERATIONS_PER_INVOCATION; k++) {
        ShoppingCart cart = createNewShoppingCart(random);
        cartMap.set(cart.id, cart);
    }
}

@Override
public void readPerformance() {
    Random random = new Random();
    for (int k = 0; k < OPERATIONS_PER_INVOCATION; k++) {
        long orderId = random.nextInt(maxOrders);
        cartMap.get(orderId);
    }
}

private ShoppingCart createNewShoppingCart(Random random) {
    ShoppingCart cart = new ShoppingCart();
    cart.id = random.nextInt(maxOrders);
    cart.date = new Date();
    int count = random.nextInt(maxCartItems);
    for (int k = 0; k < count; k++) {
        ShoppingCartItem item = createNewShoppingCartItem(random);
        cart.addItem(item);
    }
    return cart;
}
```



java.io.Serializable - Results

Read Performance

56 ops in ms

Write Performance

46 ops in ms

Binary object size

514 bytes



java.io.Externalizable



java.io.Externalizable

Pros

Standard

More efficient than Serializable in terms of CPU and memory consumption

Cons

Requires a developer to implement the actual serialization methods



ShoppingCartItem - implementation

```
public class ShoppingCartItem implements Externalizable {

    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeLong(cost);
        out.writeInt(quantity);
        out.writeUTF(itemName);
        out.writeBoolean(inStock);
        out.writeUTF(url);
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException,
        ClassNotFoundException {
        cost = in.readLong();
        quantity = in.readInt();
        itemName = in.readUTF();
        inStock = in.readBoolean();
        url = in.readUTF();
    }
}
```



ShoppingCart- implementation

@Override

```
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeExternal(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}
```

@Override

```
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readExternal(in);
        items.add(item);
    }
}
```



java.io.Externalizable - Results

Read Performance

67 ops in ms

Write Performance

70 ops in ms

Binary object size

228 bytes



DataSerializable



DataSerializable

Hazelcast specific

Pros

More efficient than Serializable in terms of CPU and memory consumption

Cons

Requires you to implement the actual serialization
Uses Reflection while de-serializing



ShoppingCartItem - implementation

@Override

```
public void writeData(ObjectDataOutput out) throws IOException {  
    out.writeLong(cost);  
    out.writeInt(quantity);  
    out.writeUTF(itemName);  
    out.writeBoolean(inStock);  
    out.writeUTF(url);  
}
```

@Override

```
public void readData(ObjectDataInput in) throws IOException {  
    cost = in.readLong();  
    quantity = in.readInt();  
    itemName = in.readUTF();  
    inStock = in.readBoolean();  
    url = in.readUTF();  
}
```



ShoppingCart- implementation

```
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeData(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}

@Override
public void readData(ObjectDataInput in) throws IOException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readData(in);
        items.add(item);
    }
}
```



DataSerializable- Results

Read Performance

80 ops in ms

Write Performance

78 ops in ms

Binary object size

261 bytes



IdentifiedDataSerializable



IdentifiedDataSerializable

Hazelcast specific

Pros

More efficient than Serializable in terms of CPU and memory consumption

Doesn't use Reflection while de-serializing

Cons

Requires you to implement the actual serialization

Requires you to implement a Factory and configuration



ShoppingCartItem - implementation

```
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(cost);
    out.writeInt(quantity);
    out.writeUTF(itemName);
    out.writeBoolean(inStock);
    out.writeUTF(url);
}
```

```
@Override
public void readData(ObjectDataInput in) throws IOException {
    cost = in.readLong();
    quantity = in.readInt();
    itemName = in.readUTF();
    inStock = in.readBoolean();
    url = in.readUTF();
}
```

```
@Override
public int getFactoryId() {
    return 1;
}
```

```
@Override
public int getId() {
    return 1;
}
```



ShoppingCart- implementation

```
@Override
public void writeData(ObjectDataOutput out) throws IOException {
    out.writeLong(total);
    out.writeLong(date.getTime());
    out.writeLong(id);
    out.writeInt(items.size());
    items.forEach(item -> {
        try {
            item.writeData(out);
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}
```

```
@Override
public void readData(ObjectDataInput in) throws IOException {
    total = in.readLong();
    date = new Date(in.readLong());
    id = in.readLong();
    int count = in.readInt();
    items = new ArrayList<>(count);
    for (int i = 0; i < count; i++) {
        ShoppingCartItem item = new ShoppingCartItem();
        item.readData(in);
        items.add(item);
    }
}
```

```
@Override
public int getFactoryId() {
    return 1;
}
```

```
@Override
public int getId() {
    return 2;
}
```



IdentifiedDataSerializable – Factory Implementation

```
public class ShoppingCartDSFactory implements DataSerializableFactory {  
    @Override  
    public IdentifiedDataSerializable create(int i) {  
        switch (i) {  
            case 1:  
                return new ShoppingCartItem();  
            case 2:  
                return new ShoppingCart();  
            default:  
                return null;  
        }  
    }  
}
```

```
Config config = new Config();  
config.getSerializationConfig().addDataSerializableFactory(1, new ShoppingCartDSFactory());  
hz = Hazelcast.newHazelcastInstance(config);
```



IdentifiedDataSerializable- Results

Read Performance

80 ops in ms (85 with unsafe=true)

Write Performance

78 ops in ms (80 with unsafe=true)

Binary object size

192 bytes



Portable



Portable

Hazelcast specific

Pros

More efficient than Serializable in terms of CPU and Memory

Doesn't use Reflection while de-serializing

Supports versioning

Supports partial de-serialization during Queries

Cons

Requires you to implement the actual serialization

Requires you to implement a Factory and Class Definition

Class definition is also sent together with Data but stored only once per class



ShoppingCartItem - implementation

```
@Override
public int getFactoryId() {
    return 2;
}

@Override
public int getClassId() {
    return 1;
}

@Override
public void writePortable(PortableWriter out) throws IOException {
    out.writeLong("cost", cost);
    out.writeInt("quantity", quantity);
    out.writeUTF("name", itemName);
    out.writeBoolean("stock", inStock);
    out.writeUTF("url", url);
}

@Override
public void readPortable(PortableReader in) throws IOException {
    url = in.readUTF("url");
    quantity = in.readInt("quantity");
    cost = in.readLong("cost");
    inStock = in.readBoolean("stock");
    itemName = in.readUTF("name");
}
```



ShoppingCart- implementation

```
@Override
public int getFactoryId() {
    return 2;
}

@Override
public int getClassId() {
    return 2;
}

@Override
public void writePortable(PortableWriter out) throws IOException {
    out.writeLong("total", total);
    out.writeLong("date", date.getTime());
    out.writeLong("id", id);
    Portable[] portables = items.toArray(new Portable[]{});
    out.writePortableArray("items", portables);
}

@Override
public void readPortable(PortableReader in) throws IOException {
    Portable[] portables = in.readPortableArray("items");
    items = new ArrayList<>(portables.length);
    for (Portable portable : portables) {
        items.add((ShoppingCartItem) portable);
    }
    id = in.readLong("id");
    total = in.readLong("total");
    date = new Date(in.readLong("date"));
}
```



Portable – Factory Implementation

```
public class ShoppingCartPortableFactory implements PortableFactory {  
    @Override  
    public Portable create(int i) {  
        switch (i){  
            case 1: return new ShoppingCartItem();  
            case 2: return new ShoppingCart();  
            default: return null;  
        }  
    }  
}
```

```
Config config = new Config();  
config.getSerializationConfig().addPortableFactory(2, new ShoppingCartPortableFactory());  
  
ClassDefinitionBuilder builder0 = new ClassDefinitionBuilder(2, 1);  
builder0.addIntField("quantity").addLongField("cost").addUTFField("name").  
    addBooleanField("stock").addUTFField("url");  
ClassDefinition shoppingCartItemClassDef = builder0.build();  
  
ClassDefinitionBuilder builder1 = new ClassDefinitionBuilder(2, 2);  
builder1.addLongField("total").addLongField("date").addLongField("id")  
    .addPortableArrayField("items", shoppingCartItemClassDef);  
ClassDefinition shoppingCartClassDef = builder1.build();  
  
config.getSerializationConfig().addClassDefinition(shoppingCartClassDef);  
config.getSerializationConfig().addClassDefinition(shoppingCartItemClassDef);  
  
hz = Hazelcast.newHazelcastInstance(config);
```



Portable- Results

Read Performance

73 ops in ms (77 with unsafe=true)

Write Performance

68 ops in ms (70 with unsafe = true)

Binary object size

417 bytes



Custom serialization



Pluggable – (ex: Kryo)

Pros

Doesn't require class to implement an interface

Very convenient and flexible

Can be stream based or byte array based

Cons

Requires you to implement the actual serialization

Requires you to deal with third-party libraries



Stream and ByteArray Serializers

```
public interface StreamSerializer<T> extends Serializer {  
    void write(ObjectDataOutput out, T object) throws IOException;  
    T read(ObjectDataInput in) throws IOException;  
}
```

```
public interface ByteArraySerializer<T> extends Serializer {  
    byte[] write(T object) throws IOException;  
    T read(byte[] buffer) throws IOException;  
}
```



ShoppingCartItem - Implementation

```
public class ShoppingCartItem {  
    public long cost;  
    public int quantity;  
    public String itemName;  
    public boolean inStock;  
    public String url;  
}
```



ShoppingCart - Implementation

```
public class ShoppingCart {  
    public long total = 0;  
    public Date date;  
    public long id;  
    private List<ShoppingCartItem> items = new ArrayList<>();  
  
    public void addItem(ShoppingCartItem item) {  
        items.add(item);  
        total += item.cost * item.quantity;  
    }  
  
    public void removeItem(int index) {  
        ShoppingCartItem item = items.remove(index);  
        total -= item.cost * item.quantity;  
    }  
  
    public int size() {  
        return items.size();  
    }  
}
```



ShoppingCart Kryo StreamSerializer

```
public class ShoppingCartKryoSerializer implements StreamSerializer<ShoppingCart> {
    private static final ThreadLocal<Kryo> kryoThreadLocal
        = new ThreadLocal<Kryo>() {

        @Override
        protected Kryo initialValue() {
            Kryo kryo = new Kryo();
            kryo.register(AllTest.Customer.class);
            return kryo;
        }
    };

    @Override
    public int getTypeId() {
        return 0;
    }

    @Override
    public void destroy() {
    }

    @Override
    public void write(ObjectDataOutput objectDataOutput, ShoppingCart shoppingCart) throws IOException {
        Kryo kryo = kryoThreadLocal.get();
        Output output = new Output((OutputStream) objectDataOutput);
        kryo.writeObject(output, shoppingCart);
        output.flush();
    }

    @Override
    public ShoppingCart read(ObjectDataInput objectDataInput) throws IOException {
        InputStream in = (InputStream) objectDataInput;
        Input input = new Input(in);
        Kryo kryo = kryoThreadLocal.get();
        return kryo.readObject(input, ShoppingCart.class);
    }
}
```



Pluggable Serialization Configuration

```
Config config = new Config();  
config.getSerializationConfig().getSerializerConfigs().add(  
    new SerializerConfig().  
        setTypeClass(ShoppingCart.class).  
        setImplementation(new ShoppingCartKryoSerializer()));  
  
hz = Hazelcast.newHazelcastInstance(config);
```



Kryo - Results

Read Performance

70 ops in ms

Write Performance

57 ops in ms

Binary object size

198 bytes



Additional configuration



Native Byte Order & Unsafe

Enables fast copying of primitive arrays like byte[] and long[]
Default is big endian.

```
config.getSerializationConfig()  
    .setAllowUnsafe(true)  
    .setUseNativeByteOrder(true);
```



Compression

Compresses the data

Can be applied to Serializable and Externalizable only

Very slow (~1000 times) and CPU consuming

Can reduce 514 bytes to 15 bytes

```
hz.getConfig().getSerializationConfig().setEnableCompression(true)
```



SharedObject

Disabled by default

Will back-reference an object pointing to a previously serialized instance

```
hz.getConfig().getSerializationConfig().setEnabledSharedObject(true)
```



Summary

Serializable

R:56 ops/ms, W: 46 ops/ms, Size: 514 bytes

Externalizable

R:67 ops/ms, W: 70 ops/ms, Size: 228 bytes

DataSerializable

R:80 ops/ms, W: 78 ops/ms, Size: 261 bytes

IdentifiedDataSerializable

R:80(85) ops/ms, W: 78 (80) ops/ms, Size: 192 bytes

Portable

R:73(77) ops/ms, W: 68(70) ops/ms, Size: 417 bytes

Kryo

R:70 ops/ms, W: 57 ops/ms, Size: 198 bytes



Events and Listeners



Event types

Hazelcast's event system allows you to respond to many different kinds of events, including:

Cluster membership changes

Client connection events

Lifecycle events of `HazelcastInstance`

Creation and destruction of Hazelcast data structures

Partition migration and partition loss

Changes to contents of Maps, Queues and Lists



Good to know

- Listeners will be triggered by an event regardless of its origin
- Listeners are instantiated once in the node that registered it
- Listeners should not contain long-running tasks. Instead, they should be off-loaded to an `ExecutorService`
- Event is produced only if there is listener subscribed to it
- Events are accumulated in a bounded queue
- If the event queue overflows, the event will be discarded



Membership Listener

**Triggered when a node joins or leaves the cluster
or when its attribute is changed**

```
public class ClusterMembershipListener implements MembershipListener {  
    @Override  
    public void memberAdded(MembershipEvent event) {  
        InetSocketAddress address = event.getMember().getSocketAddress();  
        System.out.println("Member joined from address " + address);  
    }  
  
    @Override  
    public void memberRemoved(MembershipEvent event) {  
        InetSocketAddress address = event.getMember().getSocketAddress();  
        System.out.println("Member left from address " + address);  
    }  
  
    @Override  
    public void memberAttributeChanged(MemberAttributeEvent event) {  
        System.out.println("Attribute " + event.getKey() + " changed");  
    }  
}
```



Distributed Object Listener

**Triggered when a distributed data structure
is constructed or shut down**

```
public class MapListener implements DistributedObjectListener {  
    @Override  
    public void distributedObjectCreated(DistributedObjectEvent event) {  
        if (event.getDistributedObject() instanceof IMap) {  
            System.out.println("Map created");  
        }  
    }  
  
    @Override  
    public void distributedObjectDestroyed(DistributedObjectEvent event) {  
        if (event.getDistributedObject() instanceof IMap) {  
            System.out.println("Map destroyed");  
        }  
    }  
}
```



Migration Listener

Triggered when partition migration is started, finished or failed

```
public class ClusterMigrationListener implements MigrationListener {  
    @Override  
    public void migrationStarted(MigrationEvent event) {  
        System.out.printf("Moving partition %d from %s to %s %n",  
            event.getPartitionId(), event.getOldOwner(), event.getNewOwner());  
    }  
  
    @Override  
    public void migrationCompleted(MigrationEvent event) {  
        System.out.printf("Partition %d successfully moved from %s to %s %n",  
            event.getPartitionId(), event.getOldOwner(), event.getNewOwner());  
    }  
  
    @Override  
    public void migrationFailed(MigrationEvent event) {  
        System.out.println("Partition migration failed");  
    }  
}
```



Partition Lost Listener

Allows you to get notified of potential data loss

```
public class ConsoleLoggingPartitionLostListener implements PartitionLostListener {  
    @Override  
    public void partitionLost(PartitionLostEvent event) {  
        if (event.getLostBackupCount() > 1) {  
            String message =  
                "Oh no! The partition " + event.getPartitionId() + " is lost";  
            System.out.println(message);  
        }  
    }  
}
```



Lifecycle Listener

Triggered *locally* on the start and finish of the following operations:

Members joining or leaving a cluster

Clients connecting to or disconnecting from a cluster

Members merging together after a split-brain situation

```
public class ClientConnectionListener implements LifecycleListener {  
    @Override  
    public void stateChanged(LifecycleEvent event) {  
        if (event.getState().equals(LifecycleEvent.LifecycleState.CLIENT_CONNECTED)) {  
            System.out.println("New client connected!");  
        }  
    }  
}
```





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.

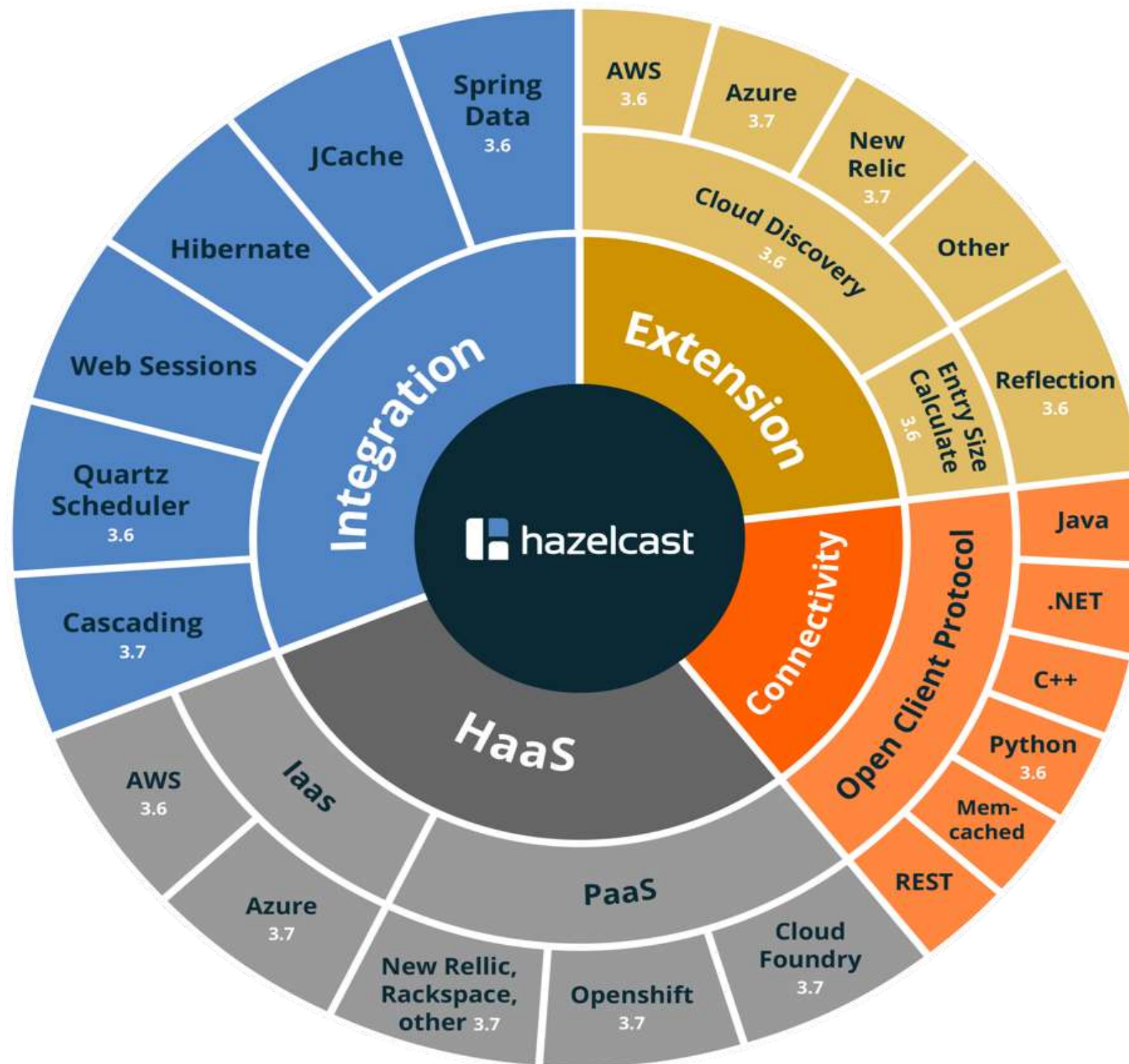
New Hazelcast 3.7 Clients and Languages

	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 Foundry

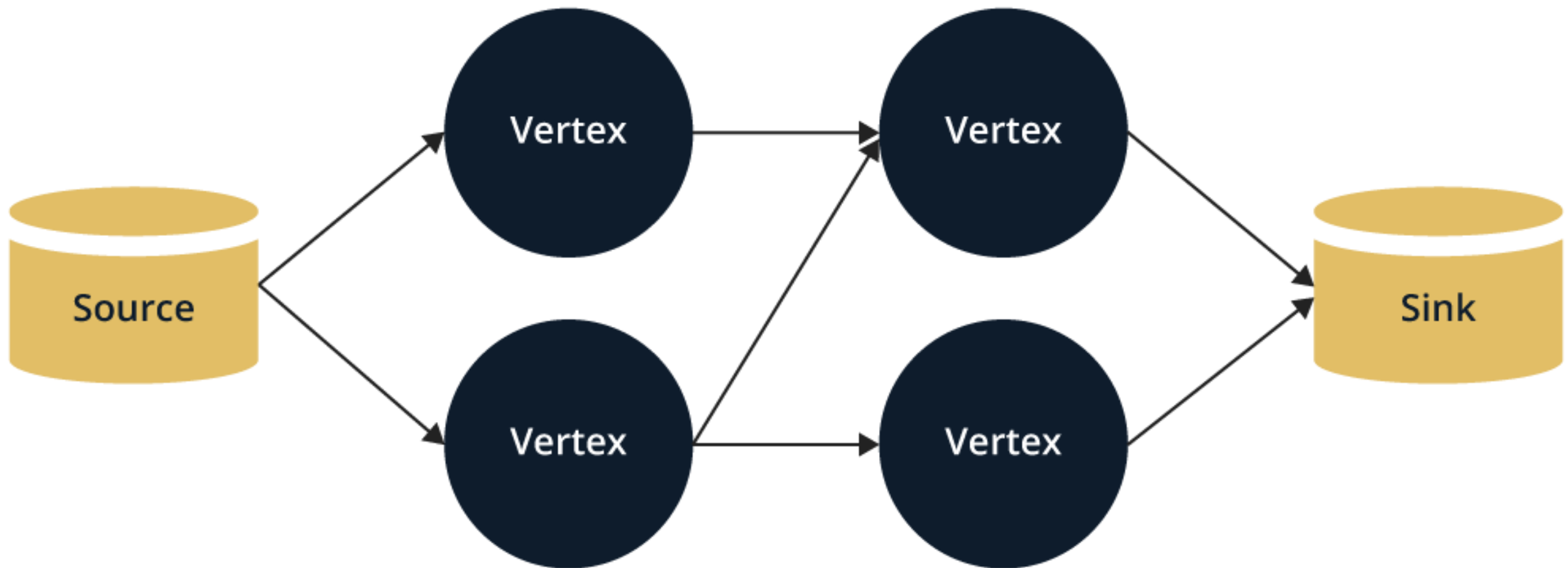


hazelcast **JET**

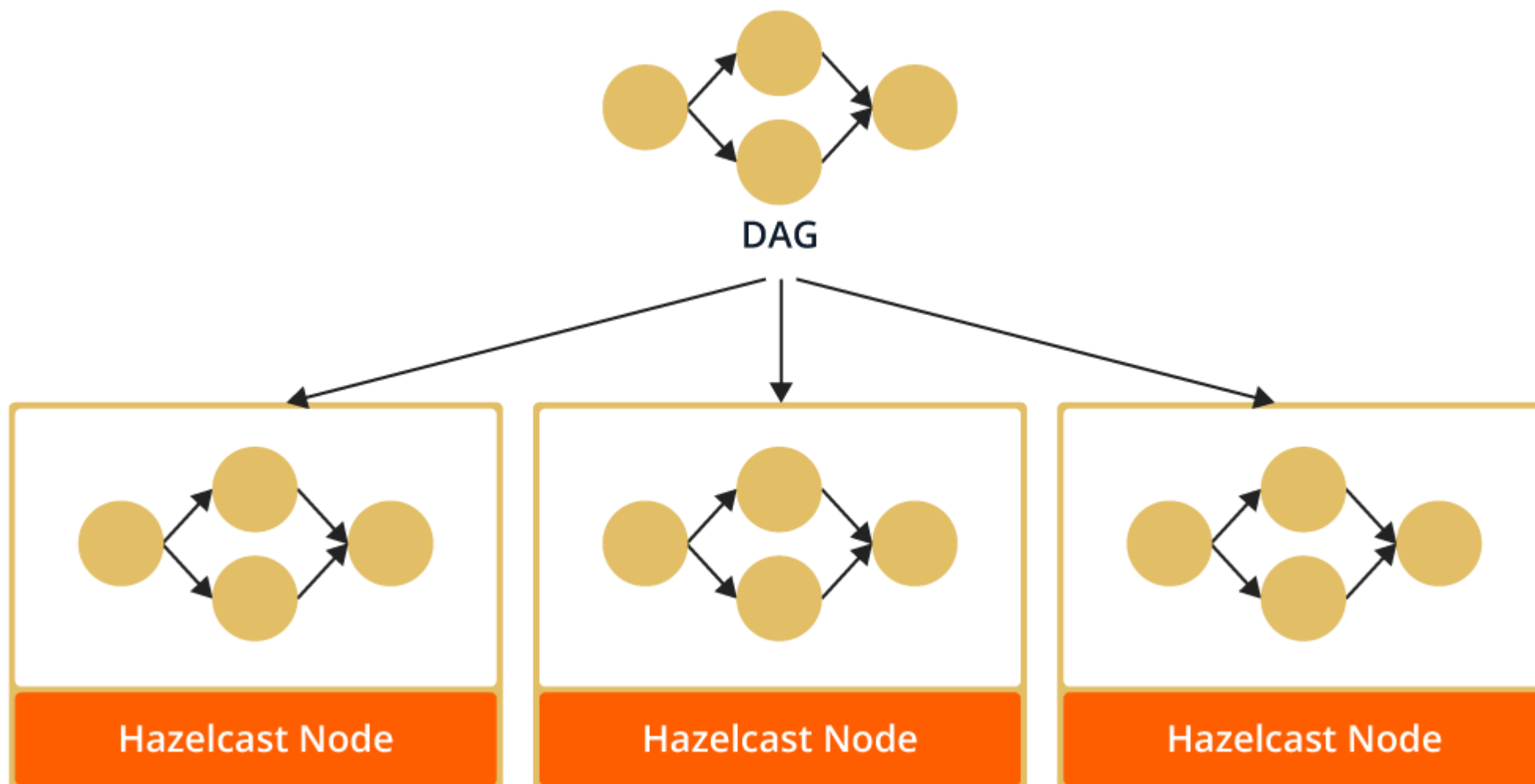
What's Hazelcast Jet?

- 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)

- Basic Subscription – 8x5 support*
- 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 CRITICAL 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!"

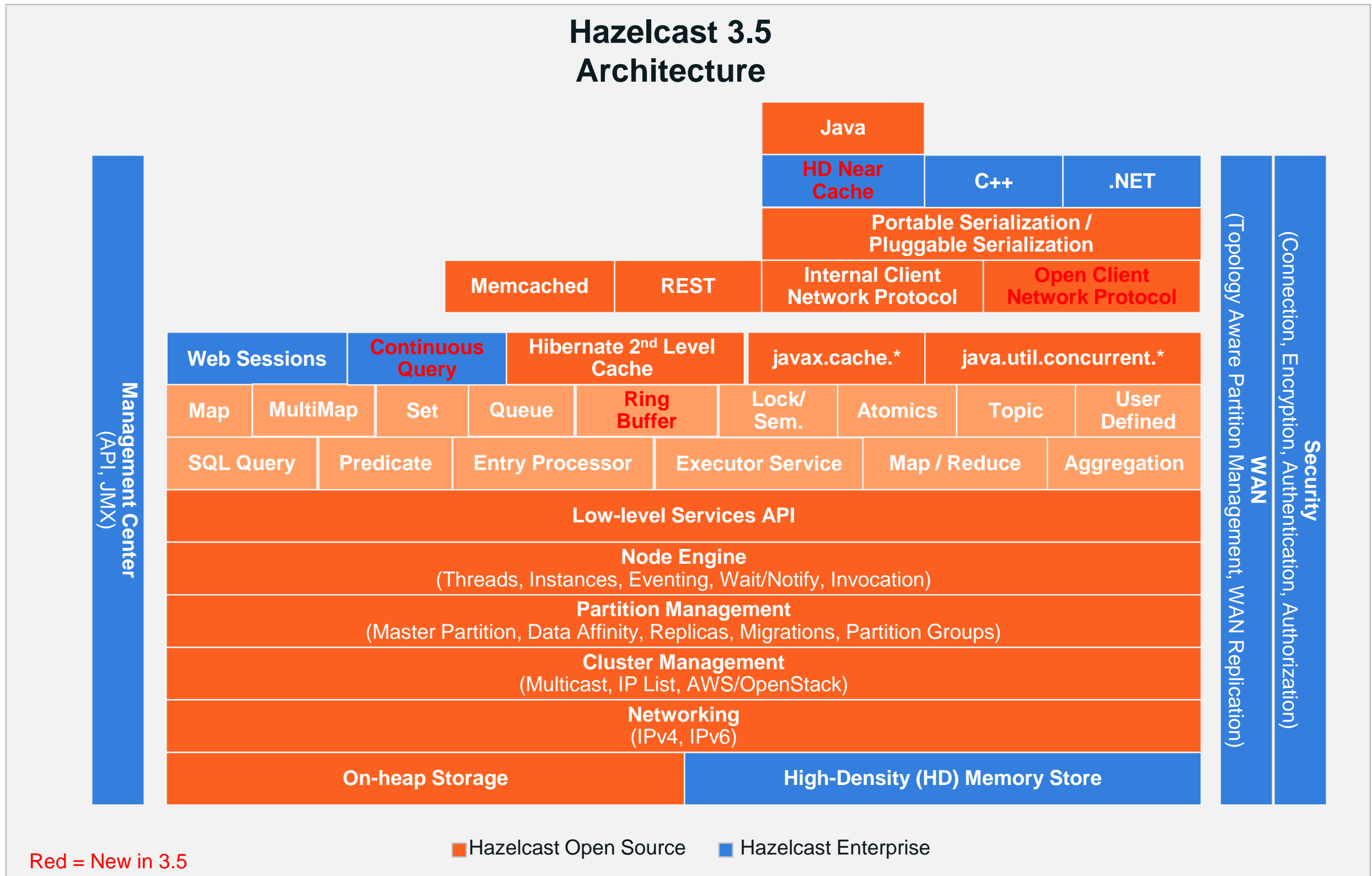
Release Lifecycle

- **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

Hazelcast Enterprise



Hazelcast In-memory Computing Platform



OSS and Enterprise

HAZELCAST ENTERPRISE LAUNCH

Hazelcast Enterprise represents a significant investment in the next generation of Hazelcast products.

FOUNDATION

- Hazelcast Enterprise is built on the foundation of Hazelcast 3.
- Because of this it inherits all features of Hazelcast 3.
- This includes all of the familiar data structures and utilities such as Map, List, Set Queue.
- Hazelcast v3 features a new Service API which enables developers to develop their own data structures and services on top of Hazelcast data partitioning services.
- Hazelcast v3 includes interfaces such as JCache, Hibernate, Memcache and Spring.

HD MEMORY STORE

- Distributed and client data can be stored off the Java heap, eliminating lengthy Garbage Collection pauses.
- This enables single VMs to reliably scale up to access Terabytes of main memory.
- Data can be stored off-heap in both the clients and server for maximum performance and flexibility.

HAZELCAST

HAZELCAST ENTERPRISE

DISTRIBUTED DATA STRUCTURES

Map & Multimap	✓	✓
Set	✓	✓
Queues	✓	✓
Topics	✓	✓
Atomics	✓	✓
Locks and Semaphores	✓	✓

DISTRIBUTED COMPUTE

Executor Service	✓	✓
Entry Processor	✓	✓
User Defined Services	✓	✓

DISTRIBUTED QUERY

Query	✓	✓
Map/Reduce	✓	✓
Aggregators	✓	✓
Continuous Query		✓

INTEGRATED CLUSTERING

Hibernate Second Level Cache	✓	✓
Tomcat Clustered Web Sessions		✓

STANDARDS

javax.cache (JCache)	✓	✓
java.util.concurrent	✓	✓

STORAGE

On-Heap	✓	✓
HD Memory Store Server Side		✓
HD Memory Store Java Client		✓

WAN

--	--	--



OSS and Enterprise cont.

	HAZELCAST	HAZELCAST ENTERPRISE
WAN Replication		✓
MANAGEMENT		
Statistics API per node	✓	✓
JMX API per node	✓	✓
Management Center	Optional w/ Support Subscription	✓
Clustered JMX		✓
Clustered REST		✓
SERVER PROTOCOLS		
Hazelcast Network Protocol	✓	✓
Memcache Server	✓	✓
REST	✓	✓
SECURITY		
Allowed Connection IP Ranges		✓
Pluggable Socket Interceptor		✓
Encryption: Asymmetric and Symmetric		✓
Authentication		✓
Authorization		✓
JAAS Module		✓
Security Interceptor		✓
SMART CLIENTS		
Portable Serialization	✓	✓
Pluggable Serialization	✓	✓
Java Client	✓	✓
C# Client		✓
C++ Client		✓

WAN

Global as well as regionally replicated deployments are supported. Both Active-Active and Active-Passive replication modes can be established. Regional warm backup scenarios and more sophisticated rack and topology-aware clustering are also available.

SECURITY

Best in class security featuring:

- Connection level security for clients and servers via IP lists or with a pluggable socket interceptor for security systems such as Kerberos.
- Symmetric and Asymmetric socket encryption.
- JAAS Modules for Authentication and Authorization.
- Map level ACL lists.
- Security context aware Pre and Post pluggable interceptors for server operations. This provides a flexible extension point to easily build advanced security features such as fine-grained authorization and audit logging.

