Hazelcast: distributed data structures to scale your app out!

Petr Pleshachkov, petr@hazelcast.com

hazelcast

Hazelcast

- The leading open source Java in-memory data grid
  - https://github.com/hazelcast, Apache 2 License
- Distributed and elastic Java collections and concurrency primitives
  - Map, Queue, Set, List, etc
  - IAtomicLong, IAtomicReference, ISemaphore and FencedLock
- Distributed computations
  - Distributed ExecutorService, EntryProcessor, messaging, etc

Distributed in-memory Data Grid

- Distributed caching
- Keeping data in local JVM
  - Fast access and processing
  - NearCache
- Elastic scalability, high throughput and low latency, high availability
  - Data partitioning and distribution
  - Data replication across cluster to tolerate failures

IMap

```
class IMap<K, V> extends ConcurrentMap<K, V> {
   void put(K key, V value) {
      // write key/value somewhere in the cluster
   }

   V get(Object key) {
      // find value associated with the key
   }
}
```

hazelcast

IMap basics

```java
public class DistributedMap {
    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance(new Config());
        ConcurrentMap<String, String> map = hz.getMap("my-distributed-map");
        map.put("key", "value");
        map.get("key");

        //ConcurrentMap methods
        map.putIfAbsent("somekey", "somevalue");
        map.replace("key", "value", "newvalue");
    }
}
```

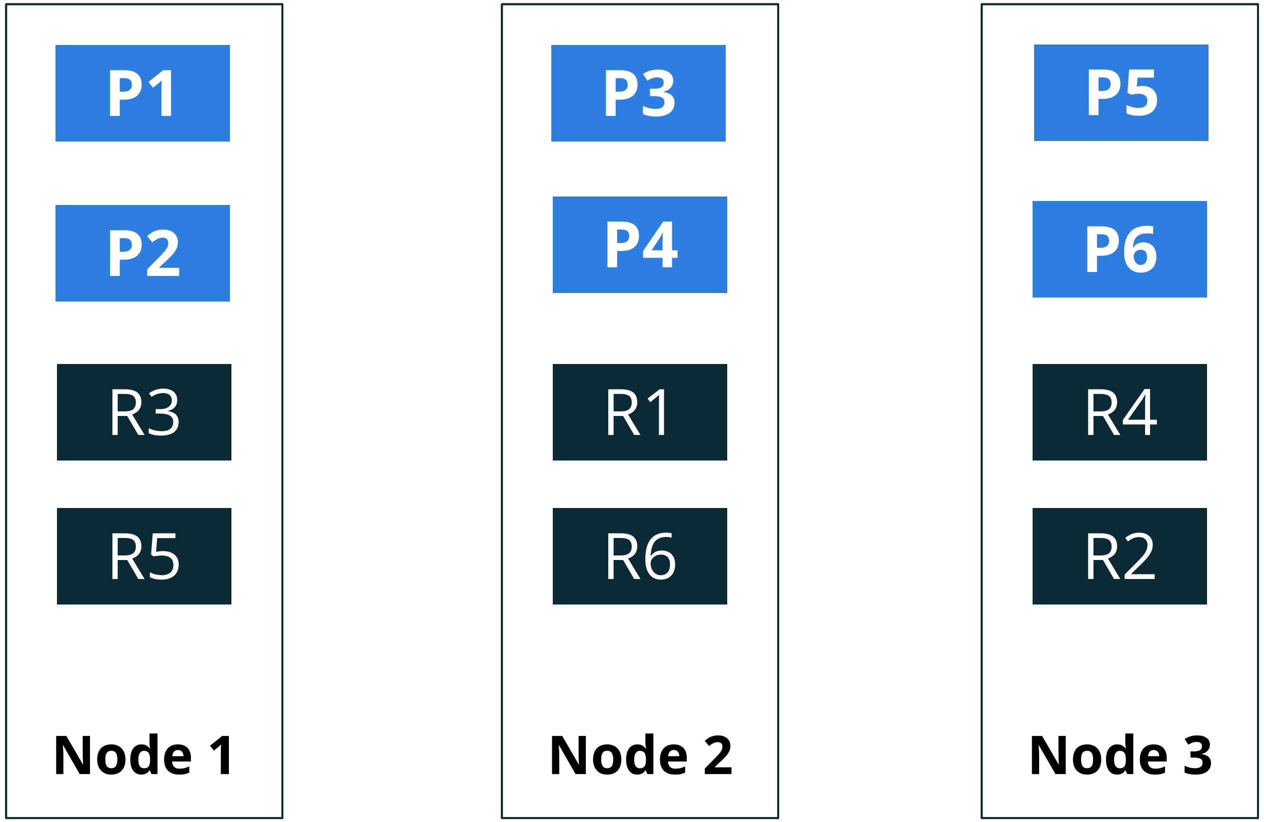
Data Partitioning

- Fixed number of partitions (default 271)
- Each key falls into a partition

  partitionId = hash(key) % PARTITION_COINT
- Partition ownerships are reassigned upon membership change
- Backup partition for redundancy

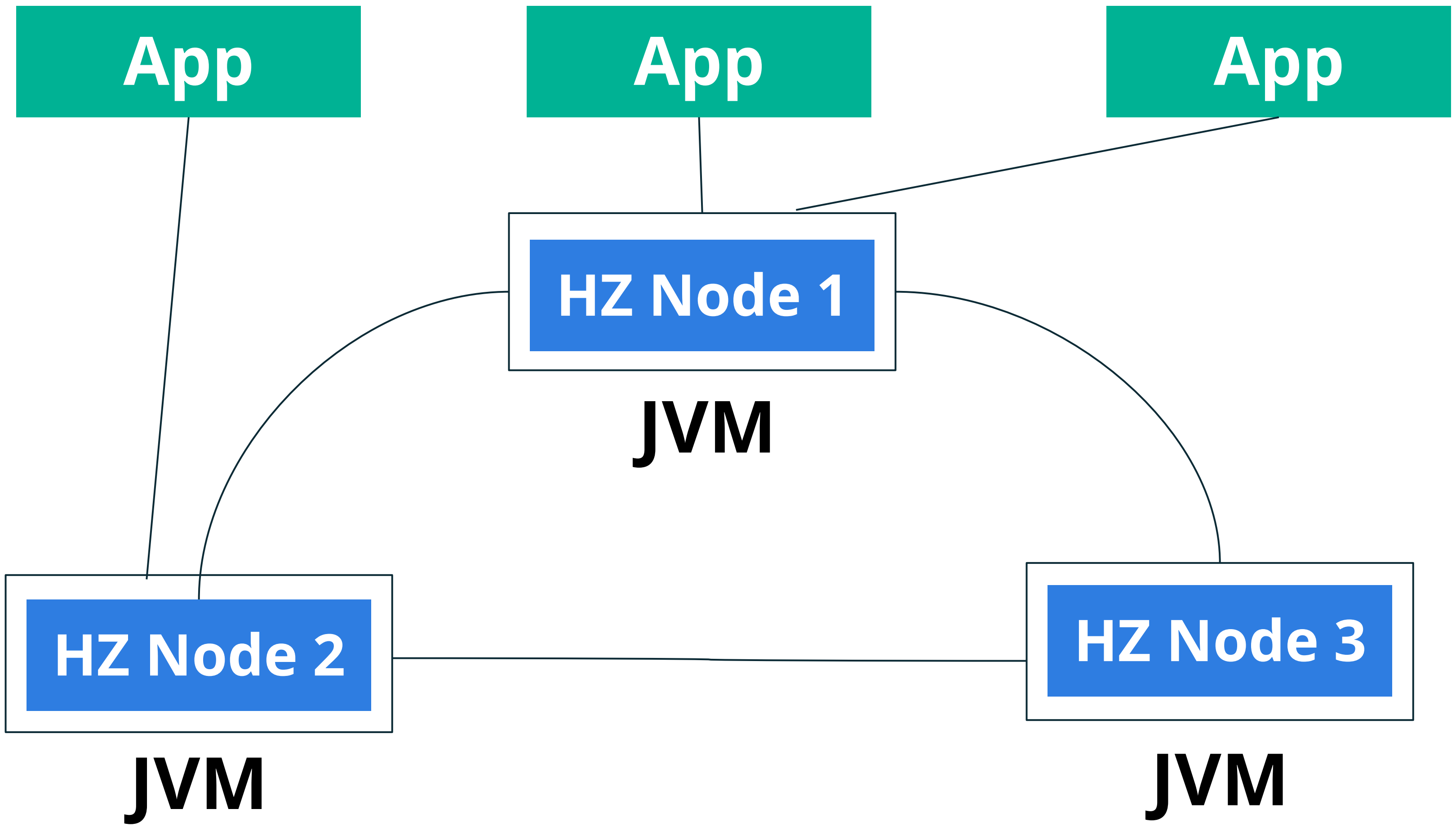
hazelcast

Data Partitioning (2)



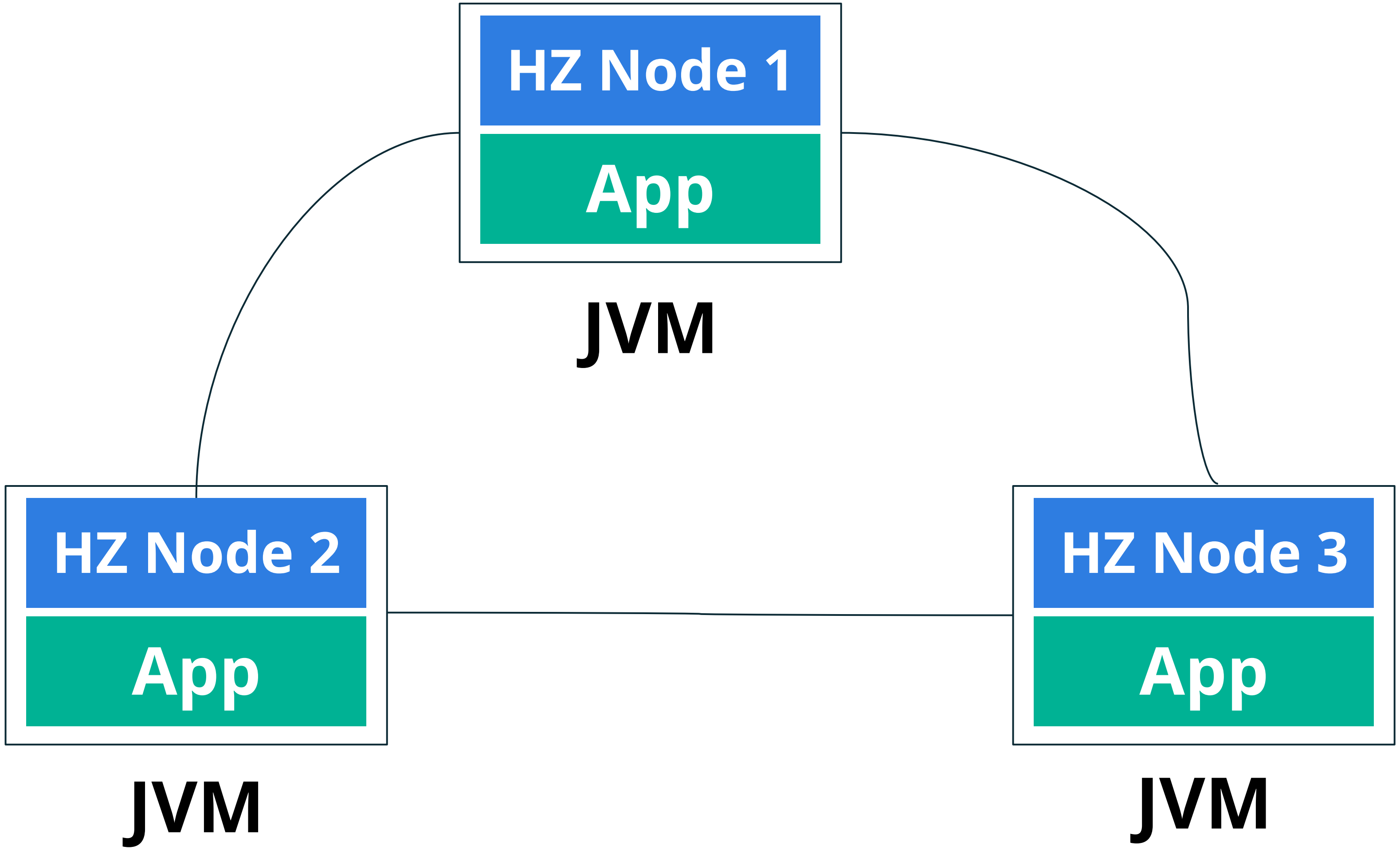
- Repartitioning occurs when a node joins/leaves the cluster
- All nodes are equal and redundant
- The minimum amount of partitions will be moved to scale out

Topology



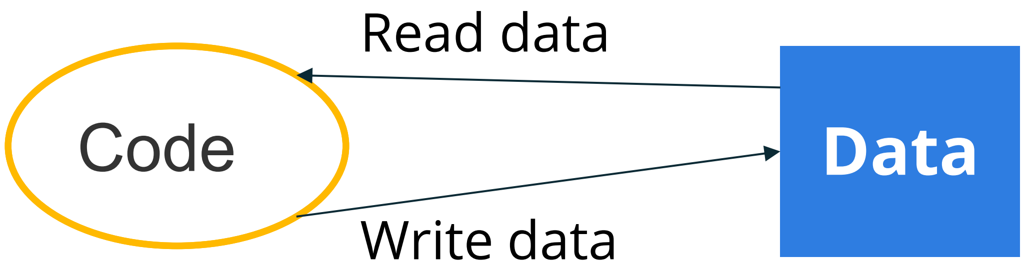
hazelcast

Embedded Topology

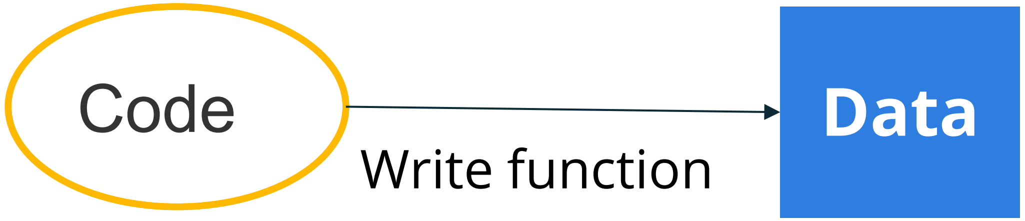


hazelcast

Entry Processor



BAD: send Data to Function



Good: send Function to Data

```java
public class EntryProcessorMain {

    public static void main(String[] args) {
        HazelcastInstance hz = Hazelcast.newHazelcastInstance();
        IMap<String, Integer> map = hz.getMap("map");
        map.put("key", 0);
        map.executeOnKey("key", new IncEntryProcessor());
        System.out.println("new value:" + map.get("key"));
    }

    public static class IncEntryProcessor extends
AbstractEntryProcessor<String, Integer> {
        public Object process(Map.Entry<String, Integer> entry) {
            int oldValue = entry.getValue();
            int newValue = oldValue + 1;
            entry.setValue(newValue);
            return null;
        }
    }
}
```

Cluster management

- A Hazelcast cluster is managed by a single node, which is called the *master*.
- Hazelcast master election is simple and practiacal.
- The oldest member in the cluster becomes the master node.
- Hazelcast maintains two pieces of information about the cluster: *member list* and *partition table*.

hazelcast

Cluster management (2)

- Member failures are detected by socket errors and heartbeat timeouts.
- When a failure is detected, that member is marked as suspect.
- From a member's view, if all members before itself in the list are suspect;
    - That member claims its mastership.
    - It forms a cluster with the members that accept its claim.
    - Members which don't accept or respond to the claim are excluded in the cluster, and they eventually become split.

hazelcast

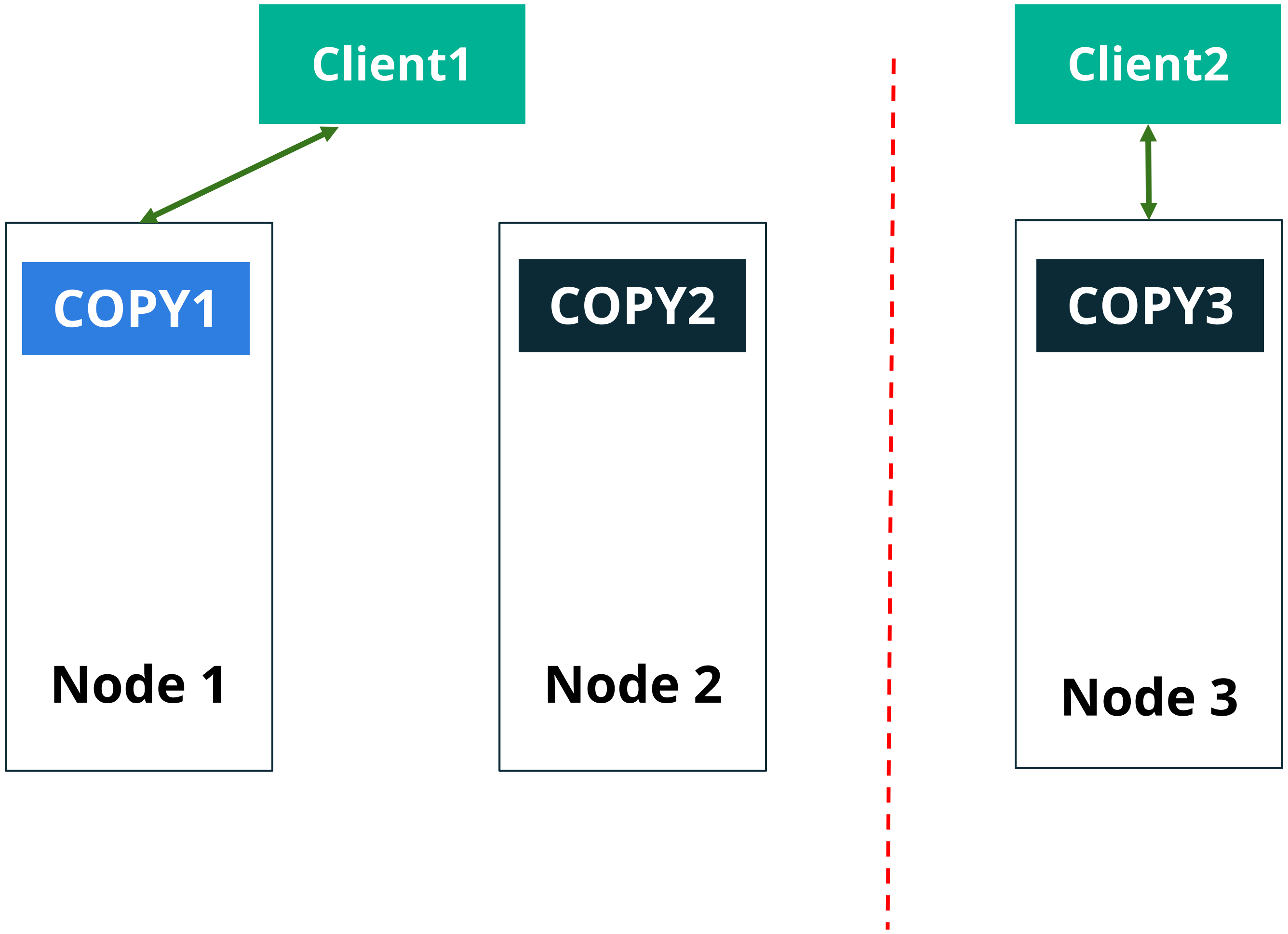
Replication challenges

- Where to perform reads and writes?
- How to keep replicas sync?
- How to handle read/write concurrency?
- How to handle failures?

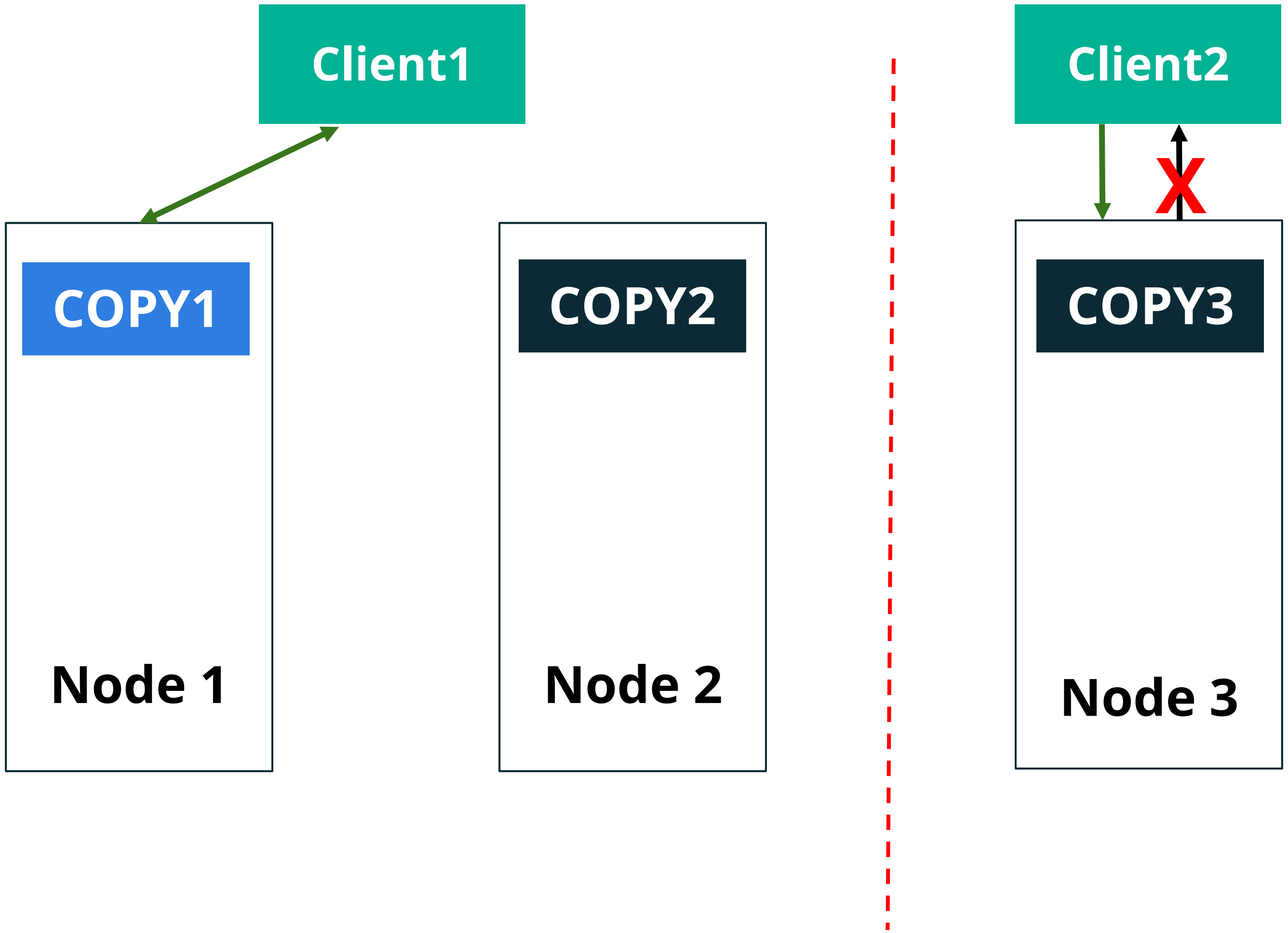
CAP theorem

- **C**onsistency
- **A**vailability
- **P**artition tolerance
- Eric Brew's **CAP theorem** implies that in the presence of a network partition, one has to choose between consistency and availability.
- **CP** versus **AP**

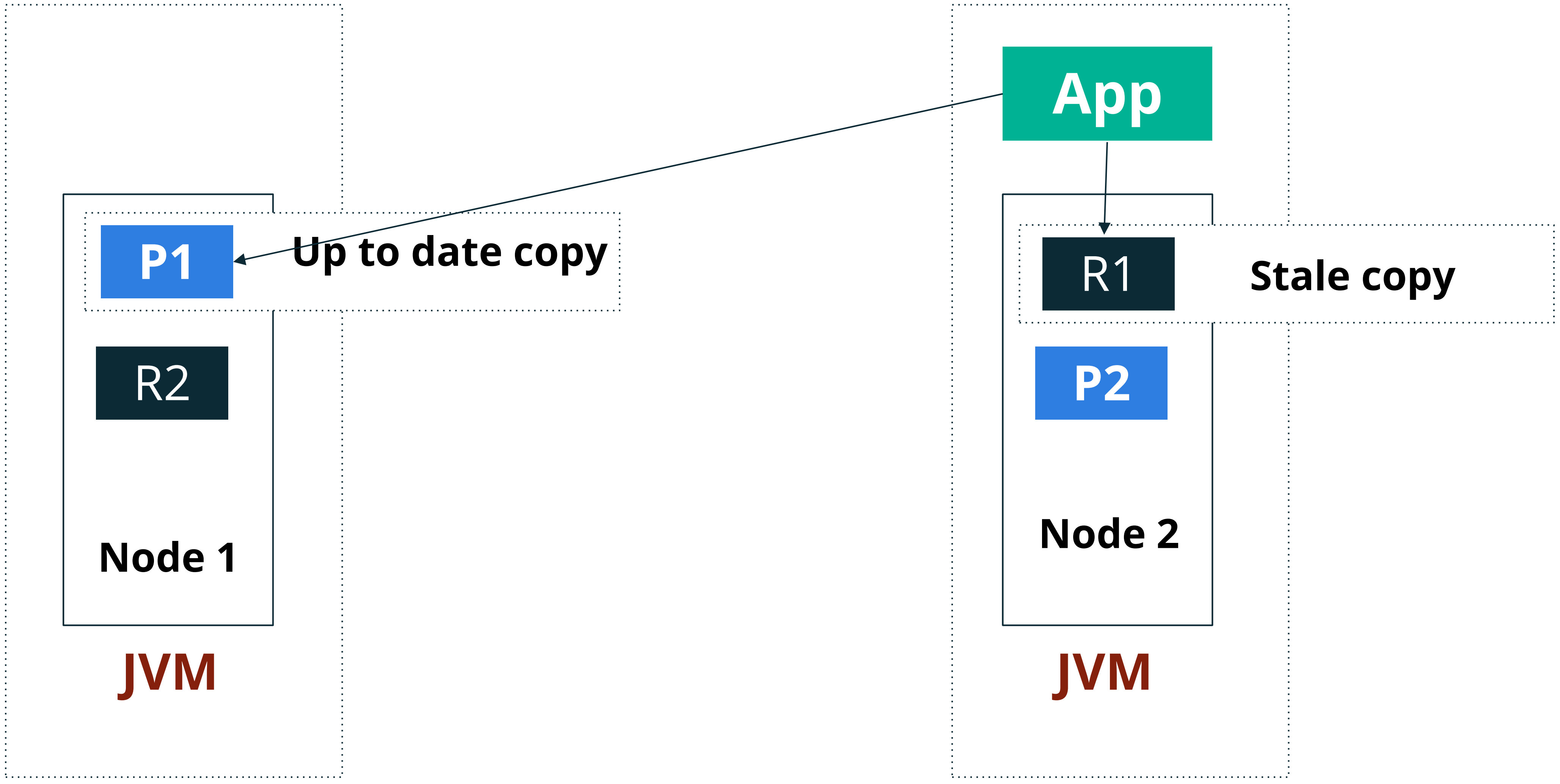
AP system



CP system



Consistency/Latency trade-off



PACELC theorem

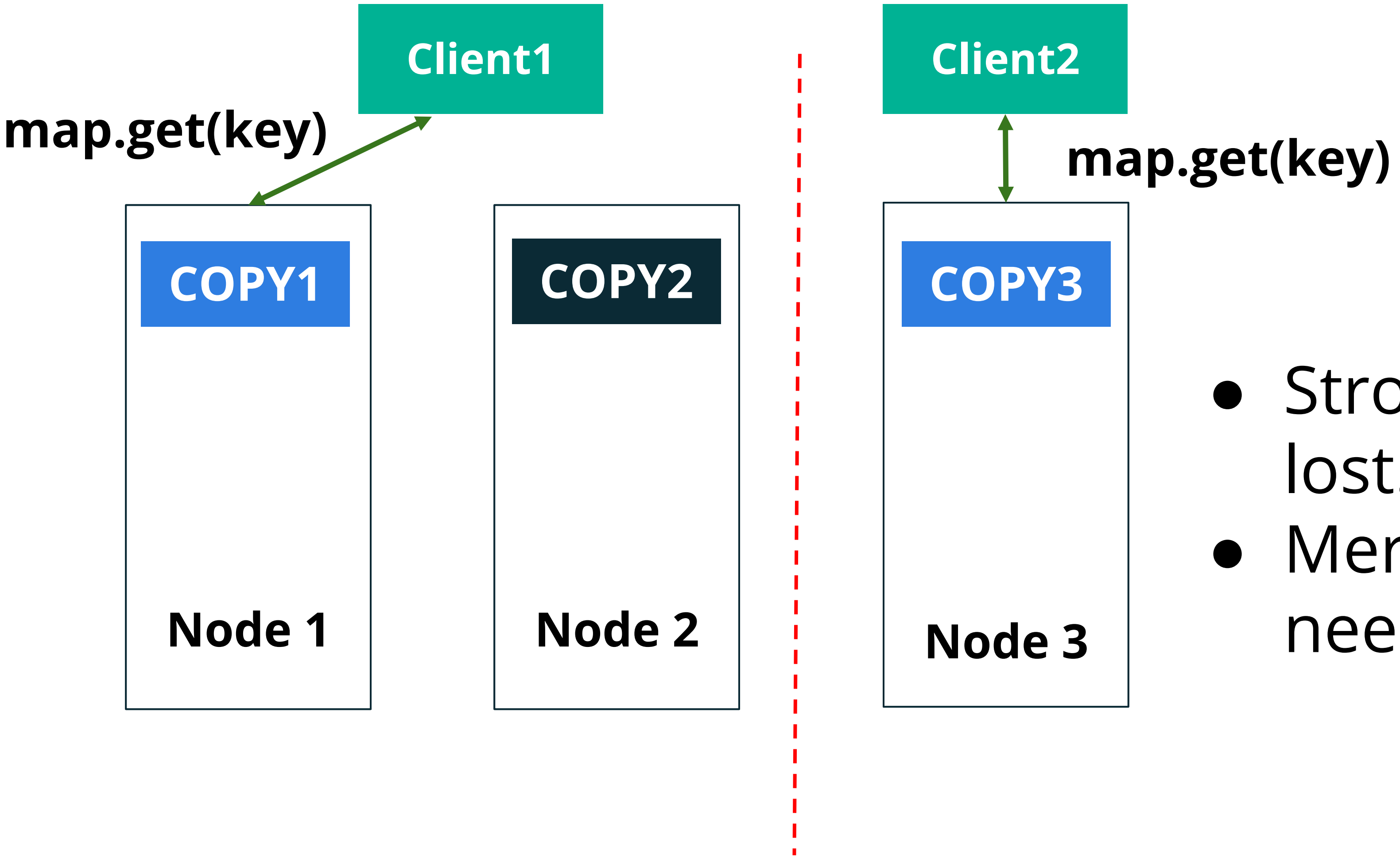
- **CAP** theorem is relevant only in a rare case of network partitioning
- Daniel Abadi's PACELC theorem:
    - If there is partitioning (**P)**, choose between consistency (**C**) and availability (**A**)
    - Else (**E**), during normal operation, choose between latency and consistency (**LC**)

Replication in Hazelcast

- Operations are sent to primary copy
- All operations on the same partition are handled by the same thread
- Strong consistency when primary is reachable
- A primary copy is elected for every partition
- Lazy replication model
  - The async mode works as fire and forget
  - In sync mode, the caller block until replica updates are applied and acknowledgments are sent back to the caller
- High throughput and availability

hazelcast

Split-brain syndrome



- Strong consistency is lost!
- Merge policies are needed!

Split-brain merge policy

```
public interface SplitBrainMergePolicy<V, T extends MergingValue<V>>
      extends DataSerializable {

   V merge(T mergingValue, T existingValue);
}
```

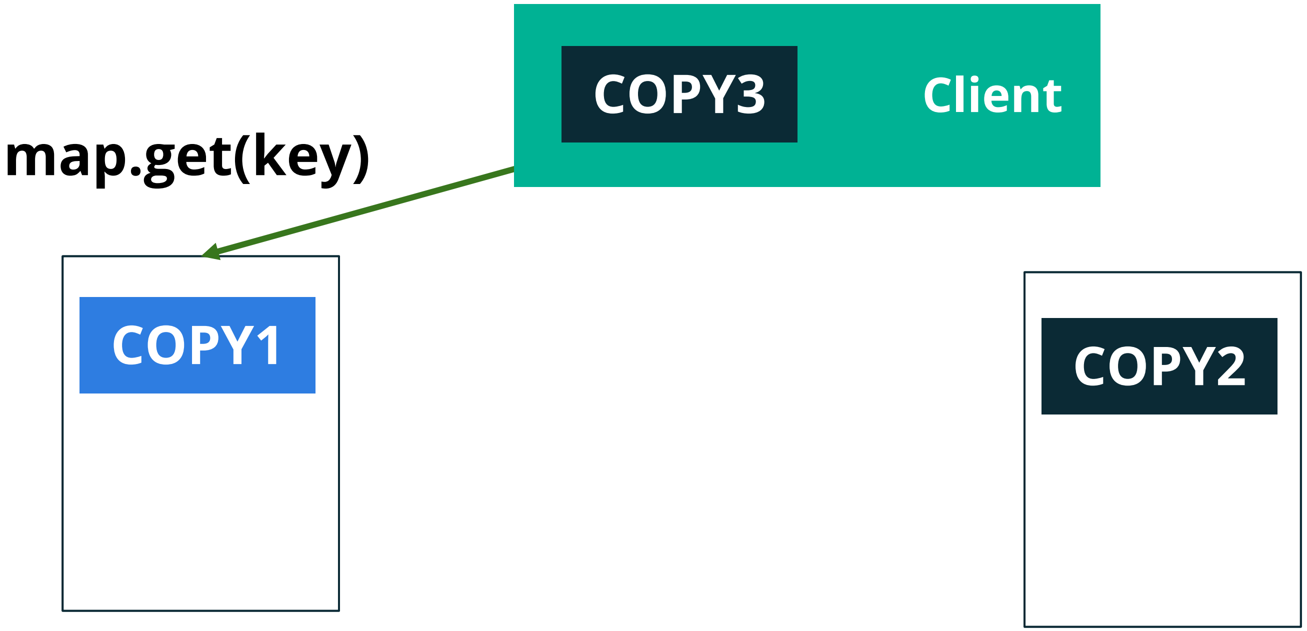
- *DiscardMergePolicy, LatestUpdateMergePolicy, LatestAccessMergePolicy, HigherHitsMergePolicy, etc.*
- Merging may cause lost updates!

hazelcast

Hazelcast is AP/EC

- Consistency is traded to availability, **AP**
- Consistency - latency trade-off is minimal during normal operation, **EC**

NearCache



- **NearCache** mechanism mitigates latency concern
  - Retains data on the client process which requested it
  - Second request processed locally
  - Updates asynchronously broadcasted to the clients
- NearCache is eventually consistent!

hazelcast

Hazelcast CP susbstem

- Concurrency APIs on top of the **Raft** consensus algorithm
- CP with respect to the CAP principle
- Linearizability in all cases, including client and server failures, network partitions
- Prevent split-brain syndrome
- Verified via extensive Jepsen test suite
- IAtomicLong, IAtomicReference, ISemaphore, and FencedLock

Thanks

- https://github.com/hazelcast
- petr@hazelcast.com