



geecon 2022
Kraków, 11-13 May



Hazelcast 5.1

The Hitchhiker's Guide

Grzegorz Piwowarek

@pivovarit



{ 4comprehension.com }

Lead Architect @ Hazelcast

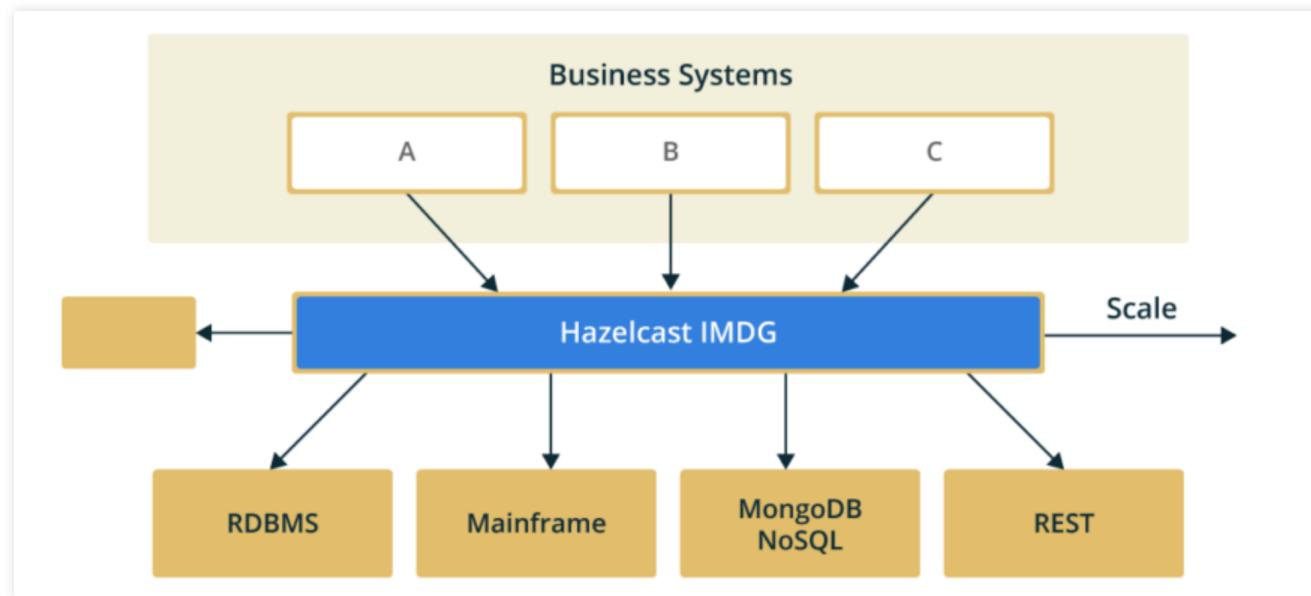
Trainer @ Bottega IT Minds

@pivovarit

@pivovarit



Born in 2008 in Turkey as a simple caching solution...

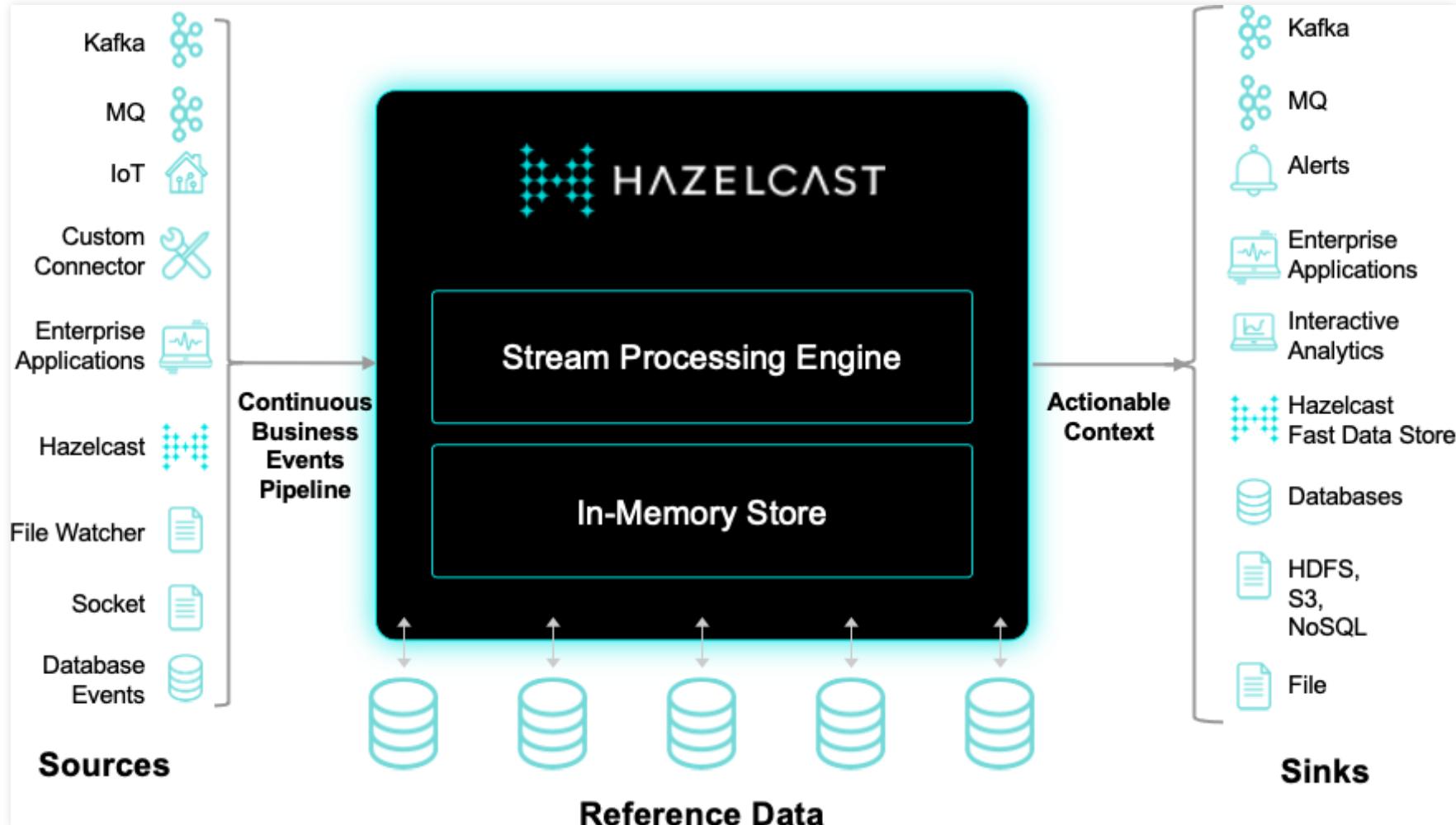


source: Hazelcast



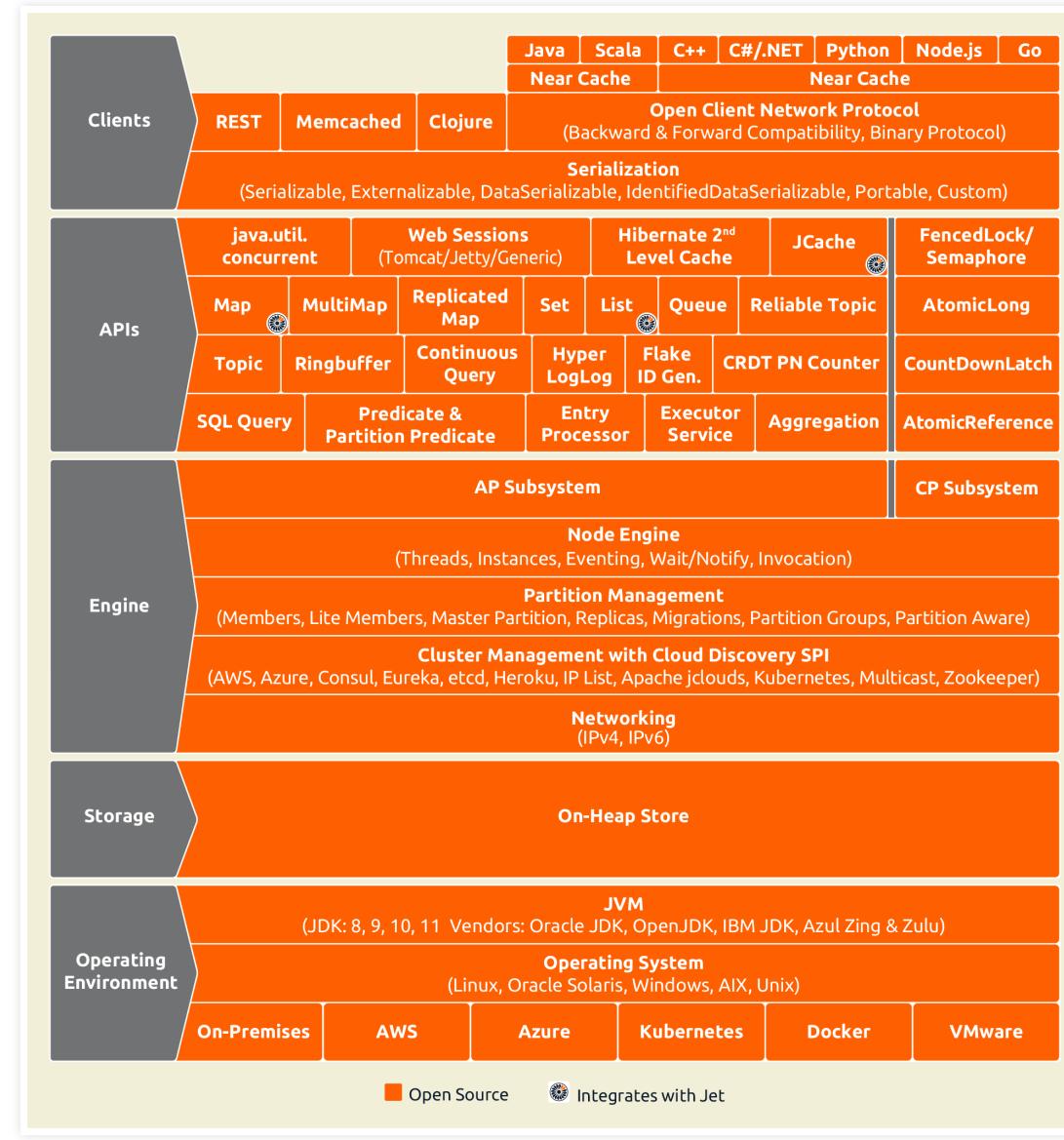
The Hazelcast IMDG is an operational in-memory computing platform that manages data and distributes processing using memory and parallel execution for breakthrough application speed and scale.





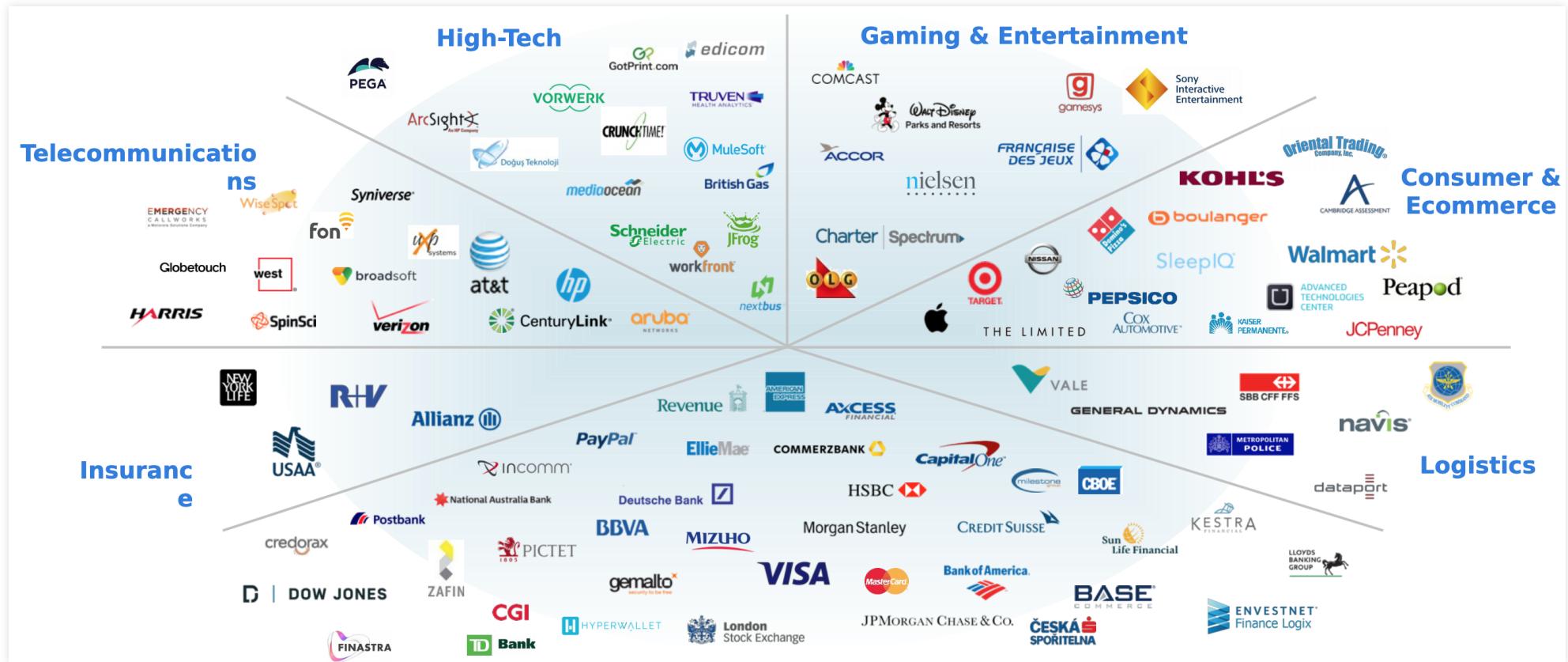
source: Hazelcast

@pivovarit



source: Hazelcast

@pivovarit



source: Hazelcast



source: Hazelcast

@pivovarit

IMDG

In-Memory Data Grid

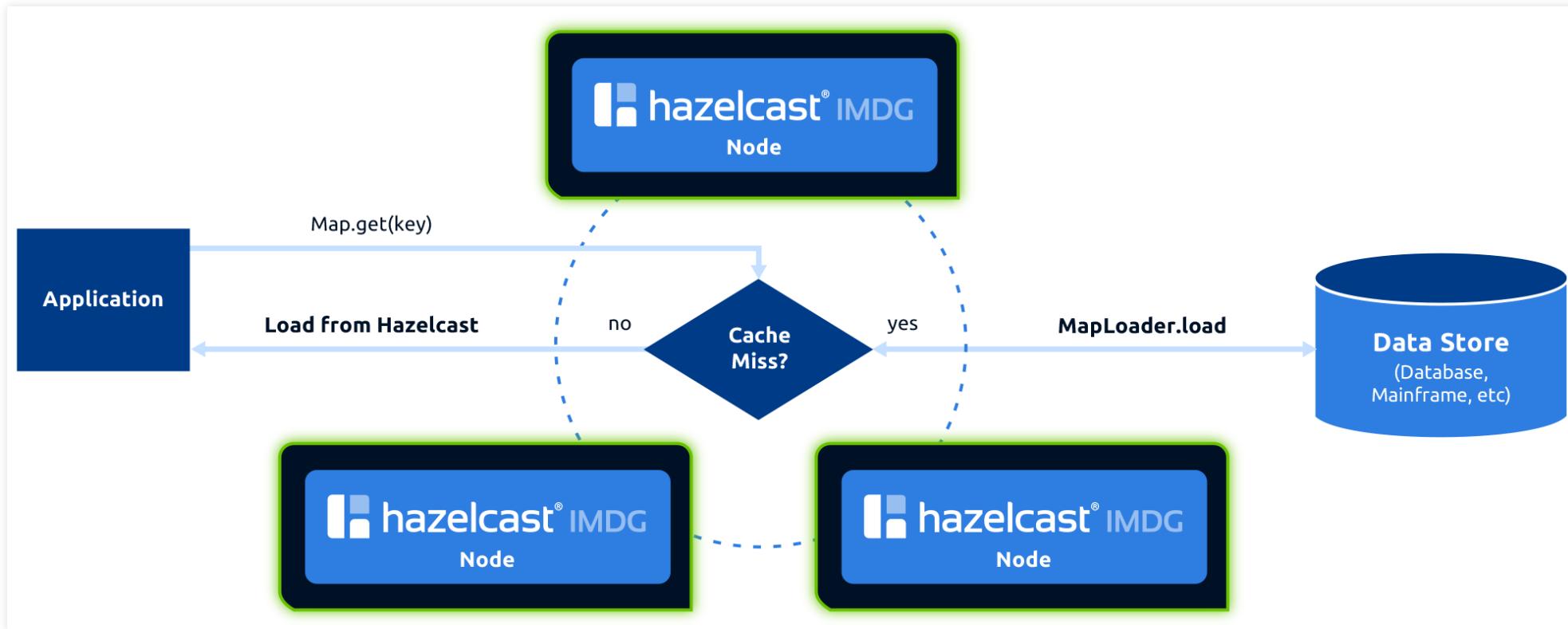
- In-Memory (no persistence)
 - Storage/Computation
- Multiple processes working as a group

- Embedded
- Client-Server

Cache Access Patterns

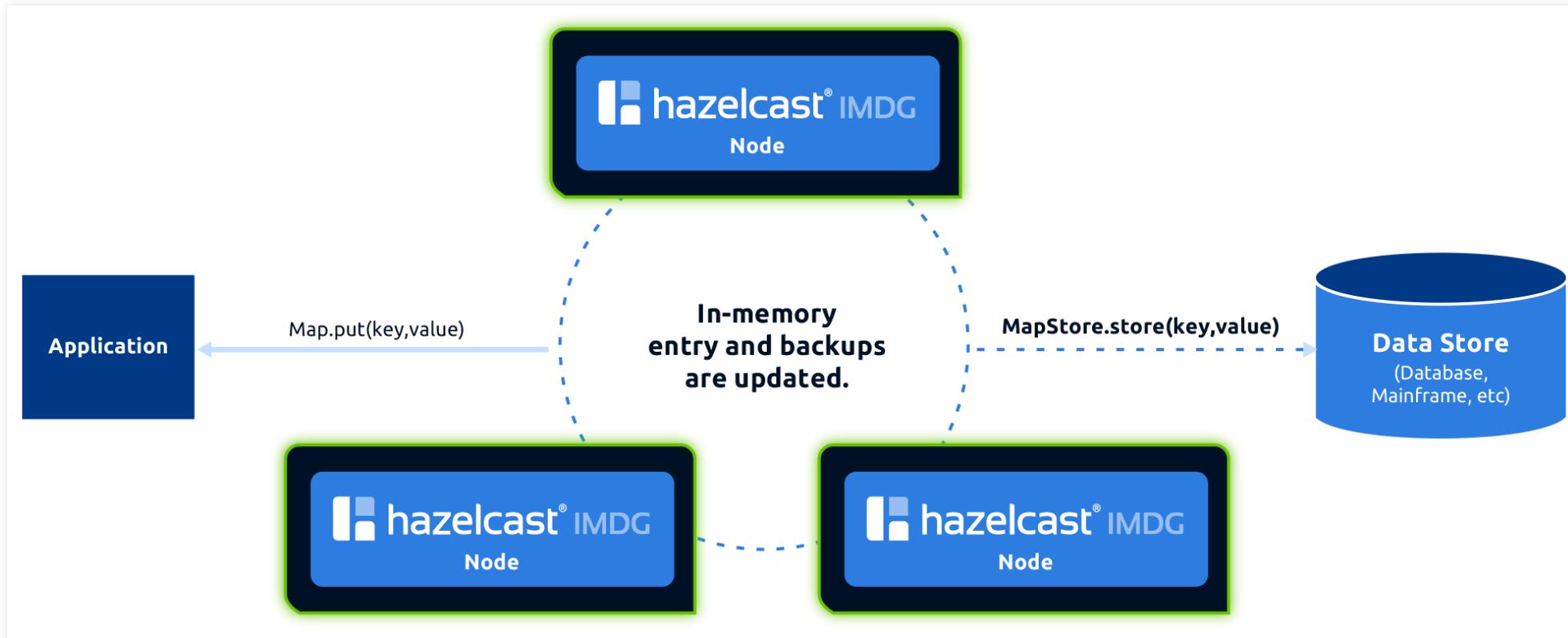
- Read-through
- Write-through
- Write-behind
- Near Cache

Read-through



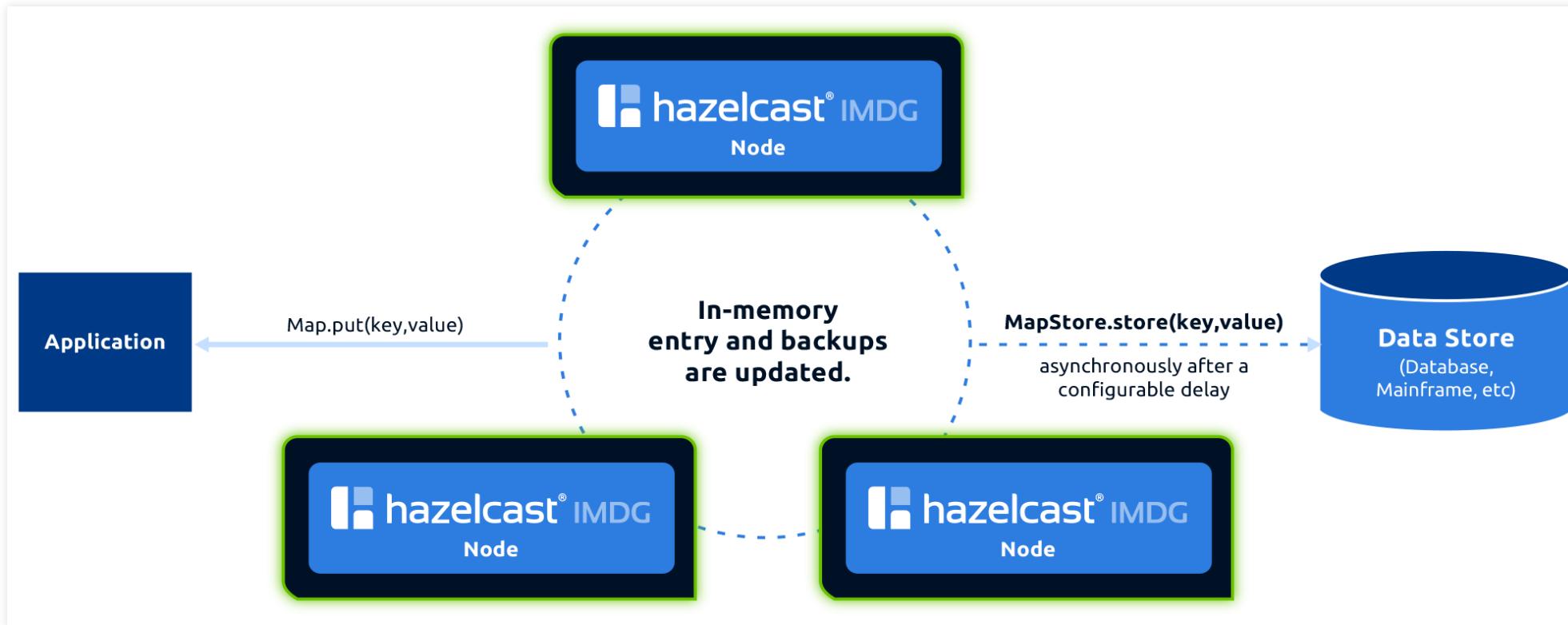
source: Hazelcast

Write-through



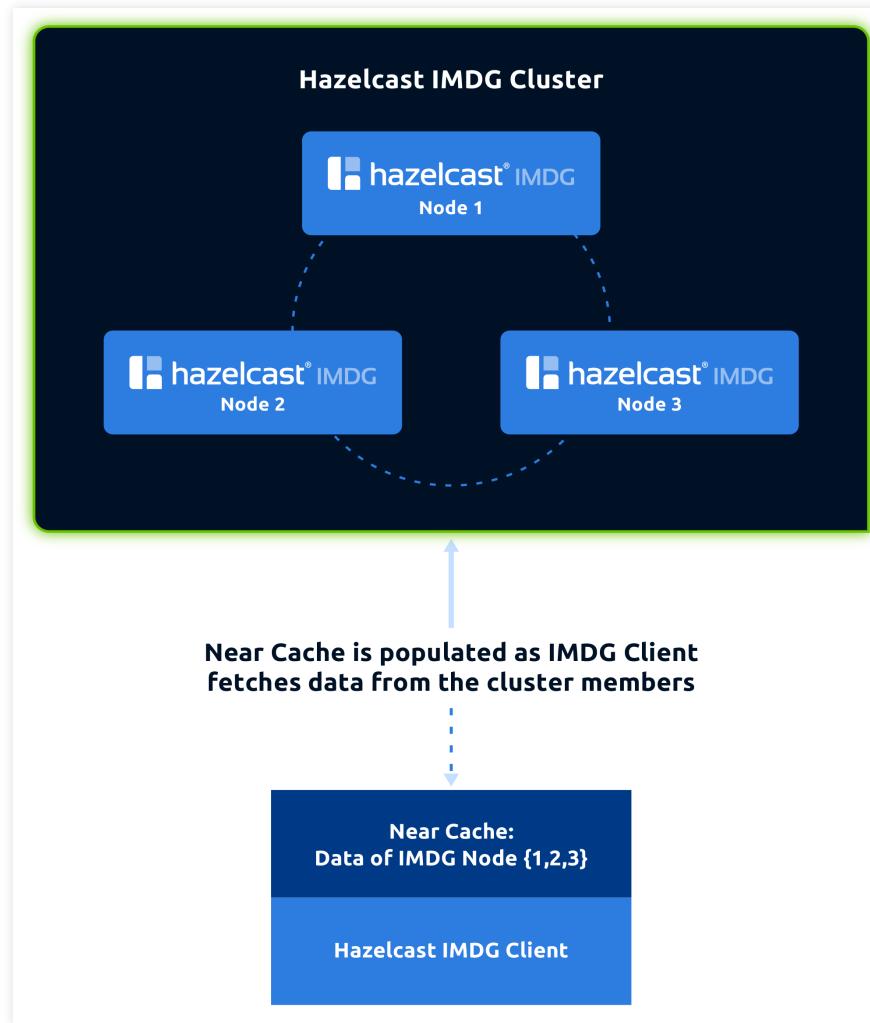
source: Hazelcast

Write-behind



source: Hazelcast

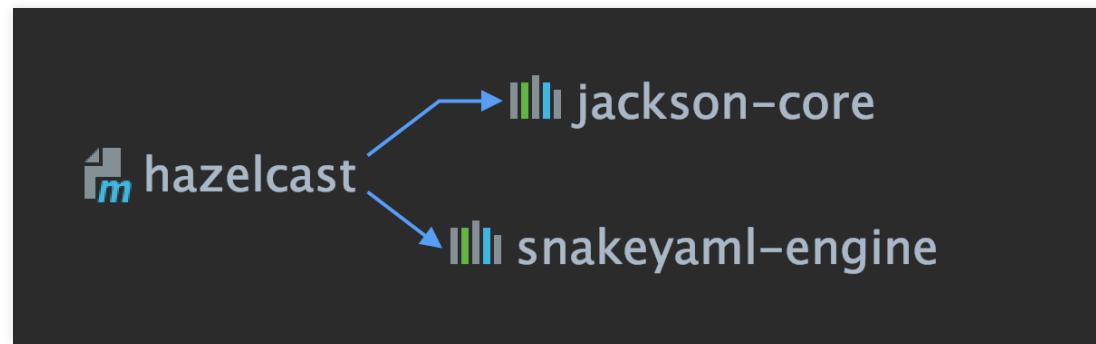
Near Cache



source: Hazelcast

@pivovarit

```
<dependencies>
    <dependency>
        <groupId>com.hazelcast</groupId>
        <artifactId>hazelcast</artifactId>
        <version>5.1</version>
    </dependency>
</dependencies>
```



Shaded

Embedded

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

Client-Server

```
HazelcastInstance hazelcastInstance = HazelcastClient.newHazelcastClient();
```

Clients

Binary Client Protocol

Java

Scala

C++

C#/.NET

Python

Node.js

Go

```
// import com.hazelcast.map.IMap;

HazelcastInstance hazelcastInstance = Hazelcast.newHazelcastInstance();

IMap<String, Integer> simpleMap = hazelcastInstance.getMap("simpleMap");

simpleMap.put("foo", 42);

simpleMap.get("foo"); // 42
```

```
// import java.util.Map;

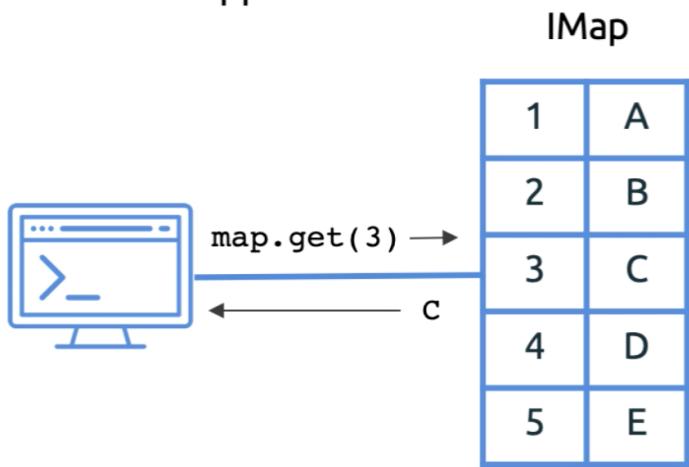
HazelcastInstance hazelcastInstance = Hazelcast.newHazelcastInstance();

Map<String, Integer> simpleMap = hazelcastInstance.getMap("simpleMap");

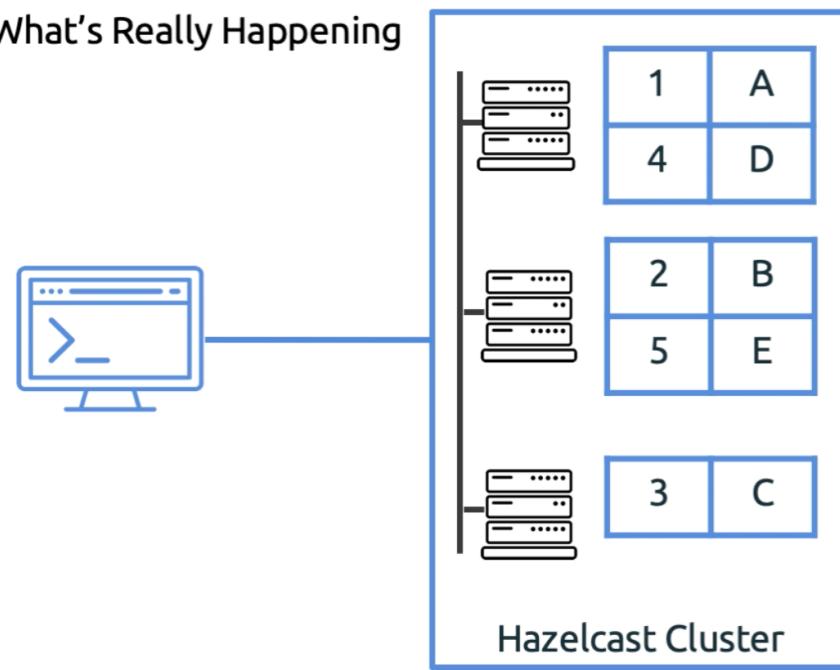
simpleMap.put("foo", 42);

simpleMap.get("foo"); // 42
```

What the App "sees"



What's Really Happening



source: Hazelcast

@pivovarit

Map is just the beginning

- MultiMap
- ReplicatedMap (non-partitioned Map)
 - Set
 - List
- Queue
- Topic

Data Structures is just the beginning

EntryProcessor

Instead of fetching the data, processing it and sending it back, let a cluster process it. A processed entry remains locked during processing

```
@BinaryInterface  
@FunctionalInterface  
public interface EntryProcessor<K, V, R> extends Serializable {  
    R process(Entry<K, V> entry);  
}
```

ExecutorService

```
public interface IExecutorService  
    extends ExecutorService, DistributedObject {  
    void execute(Runnable command, MemberSelector memberSelector);  
    void executeOnAllMembers(Runnable command);  
    // ...  
}
```

Aggregation

```
IMap<String, Employee> employees = hz.getMap("employees");

double avgSalary = employees.aggregate(
    new Aggregator<Map.Entry<String, Employee>, Double>() {
        protected long sum;
        protected long count;

        @Override
        public void accumulate(Map.Entry<String, Employee> entry) {
            count++;
            sum += entry.getValue().getSalaryPerMonth();
        }
    } // ...
);
```

Queries

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

SQL

GA in 5.0

```
try (SqlResult result = hazelcastInstance.getSql()  
    .execute("SELECT name FROM emp WHERE age < ?", 30)) {  
    for (SqlRow row : result) {  
        String name = row.getObject(0);  
        System.out.println(name);  
    }  
}
```

+ a JDBC driver

<https://hazelcast.com/blog/jdbc-driver-4-2-is-released/>

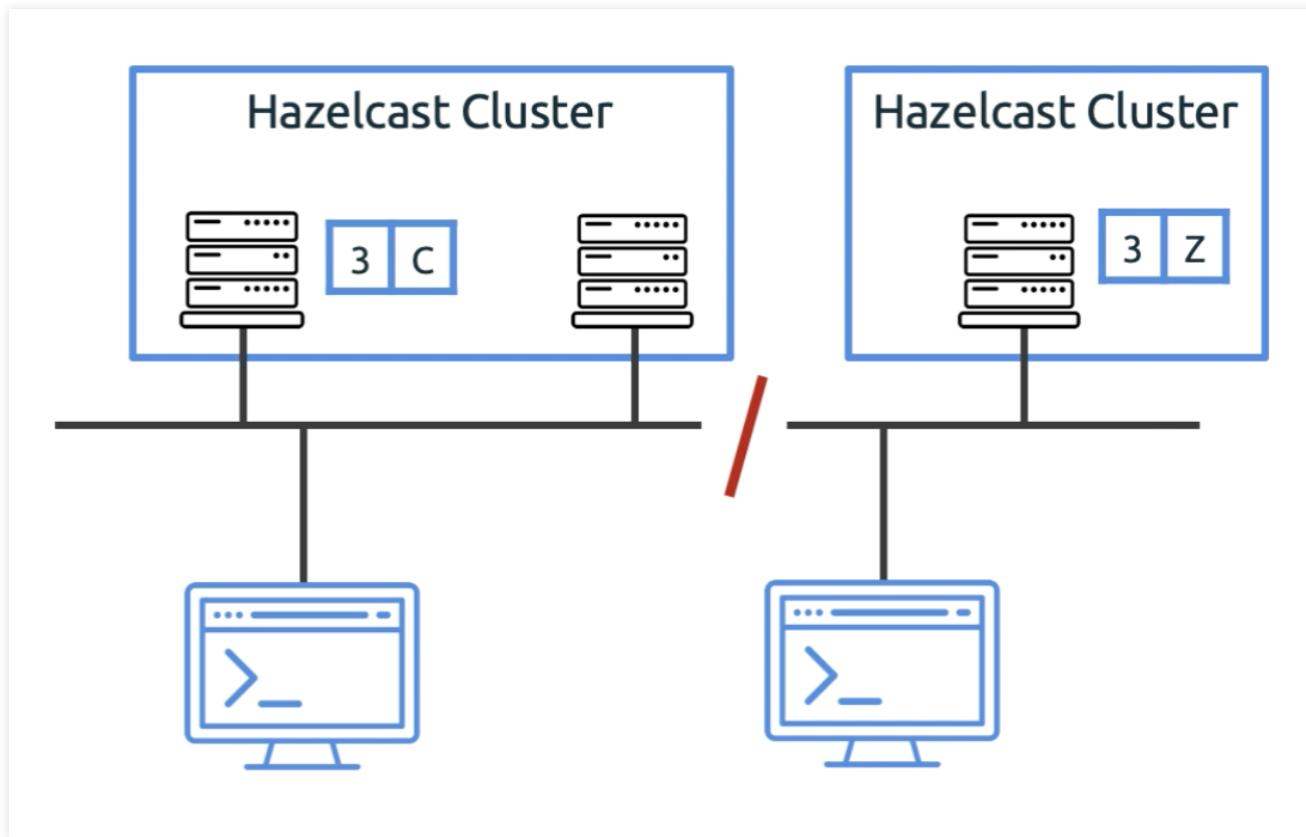
CAP Theorem

Consistency

Availability

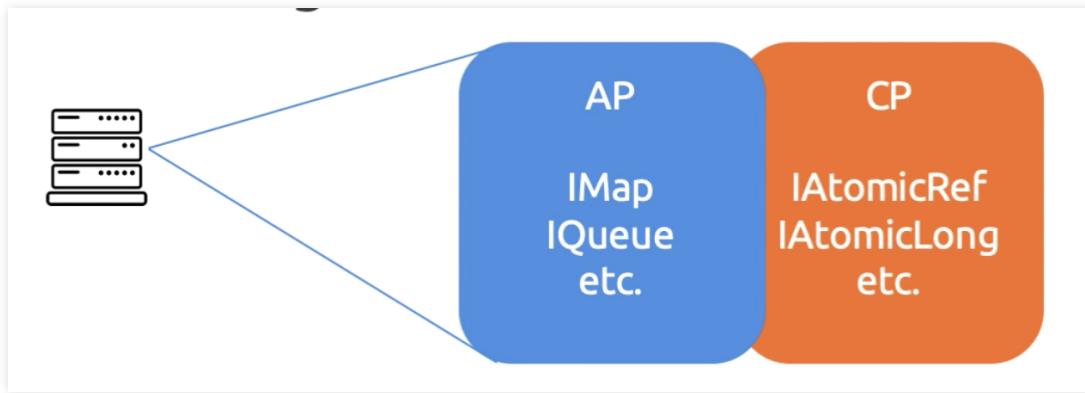
Partition Tolerance

Pick two ...but always make sure P is one of them



source: Hazelcast

CP Subsystem



source: Hazelcast

- Linearizable, distributed implementations of Java concurrency primitives
 - Raft-based(<http://thesecretlivesofdata.com/raft/>)
 - Jepsen-verified
 - Requires minimum 3 members

- Semaphore
- CountDownLatch
- AtomicReference
- AtomicLong

Advanced Data Structures

RingBuffer

A fixed-capacity data structure that overwrites oldest elements when capacity is exceeded

HyperLogLog

A probabilistic data structure used to estimate the cardinality(number of unique elements) of a data set.

The HyperLogLog algorithm can estimate cardinalities well beyond 10^9 with a relative accuracy (standard error) of 2% while only using 1.5kb of memory.

Fangjin Yang Fast, Cheap, and 98% Right: Cardinality Estimation for Big Data

Flake Id Generator

Cluster-wide ordered and unique identifiers IDs without any coordination between members
safe even in split-brain scenario

CRDT PN Counter

A conflict-free alternative to AtomicLong

Other Integrations

Hibernate 2LC

Hazelcast provides a distributed second level cache for your Hibernate entities, collections and queries.

<http://github.com/hazelcast/hazelcast-hibernate/>

Spring Data

Spring Data Hazelcast Integration

<https://github.com/hazelcast/spring-data-hazelcast>

Quarkus Hazelcast Client

<https://github.com/hazelcast/quarkus-hazelcast-client>

Tomcat Session Manager

Each HttpSession object is kept in the Hazelcast Map

<https://github.com/hazelcast/hazelcast-tomcat-sessionmanager>

Apache Shiro

Apache Shiro is a security framework that performs authentication, authorization, cryptography, and session management. Shiro uses Hazelcast for caching and session clustering.

Cloud Integrations

Zone-awareness + autodiscovery

- AWS
- GCP
- Azure

Hazelcast Cloud

Hazelcast IMDG cluster as a service



<https://cloud.hazelcast.com>

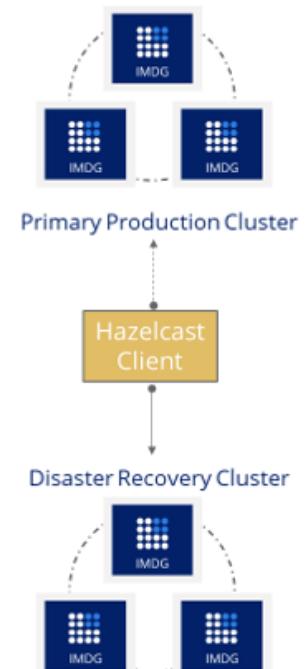
Enterprise Features

- Management Center
 - Security Suite
 - Disaster Recovery
 - WAN Replication
 - Rolling Upgrades
 - HD Memory

Disaster Recovery

› Automatic Disaster Recovery Failover

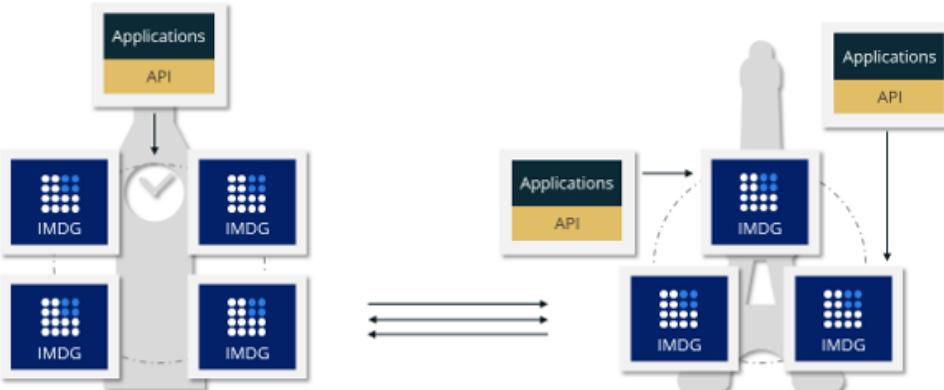
Automatic disaster recovery fail-over provides clients connected to Hazelcast® Enterprise IMDG clusters with the ability to automatically fail-over to a Disaster Recovery cluster should there be an unplanned outage of the primary production Hazelcast cluster.



source: Hazelcast

WAN Replication

› WAN Replication: Deployment Options



Independent clusters, sharing selected data to maintain alignment

One-way or two-way

Choose which data to share, select by type, can de-select records by attributes

One-way -- keep DR site populated and ready-to-go

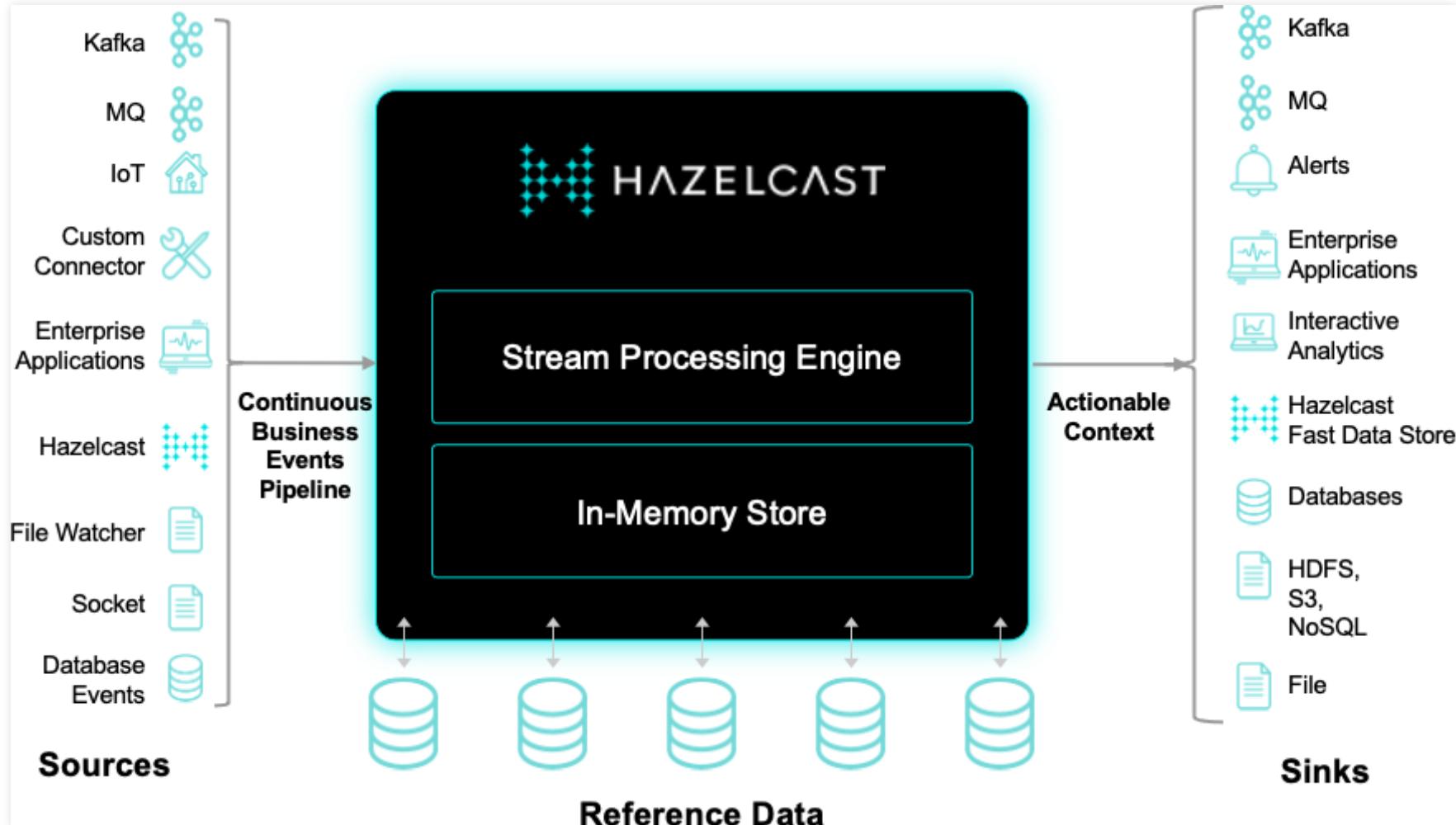
Two-way -- worldwide operations, access your nearest data copy

source: Hazelcast

HD Memory(off-heap store)
Hot Restart Store



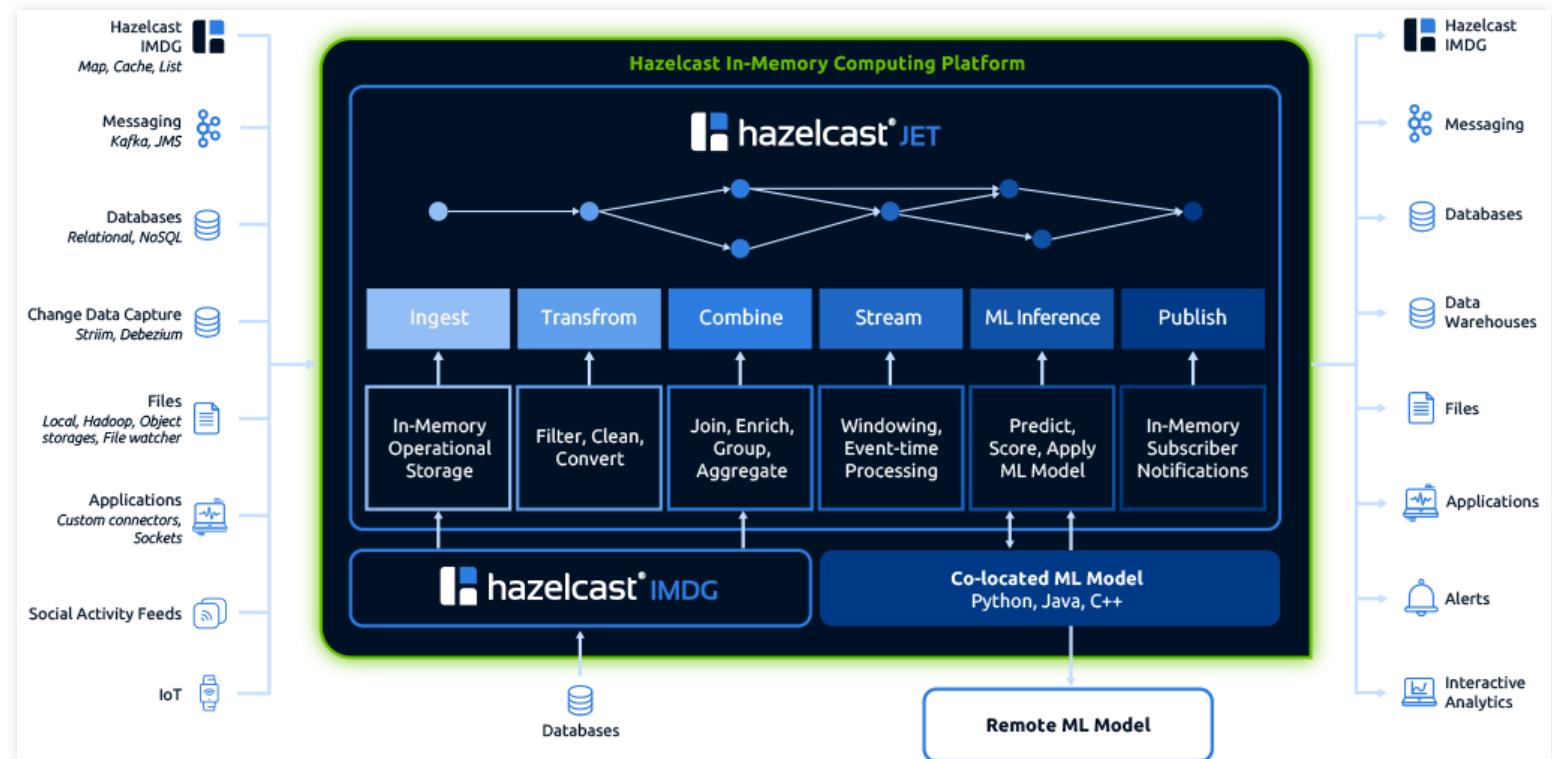
@pivovarit



source: Hazelcast

@pivovarit

Hazelcast Jet



Hazelcast Jet Whitepaper

Computer Science > Distributed, Parallel, and Cluster Computing

[Submitted on 18 Mar 2021]

Hazelcast Jet: Low-latency Stream Processing at the 99.99th Percentile

Can Gencer, Marko Topolnik, Viliam Ďurina, Emin Demirci, Ensar B. Kahveci, Ali Gürbüz Ondřej Lukáš, József Bartók, Grzegorz Gierlach, František Hartman, Ufuk Yılmaz, Mehmet Doğan, Mohamed Mandouh, Marios Frakoulis, Asterios Katsifodimos

Jet is an open-source, high-performance, distributed stream processor built at Hazelcast during the last five years. Jet was engineered with millisecond latency on the 99.99th percentile as its primary design goal. Originally Jet's purpose was to be an execution engine that performs complex business logic on top of streams generated by Hazelcast's In-memory Data Grid (IMDG): a set of high-performance, in-memory, partitioned and replicated data structures. With time, Jet evolved into a full-fledged, scale-out stream processor that can handle out-of-order streams and exactly-once processing guarantees. Jet's end-to-end latency lies in the order of milliseconds, and its throughput in the order of millions of events per CPU-core. This paper presents main design decisions we made in order to maximize the performance per CPU-core, alongside lessons learned, and an empirical performance evaluation.

Subjects: **Distributed, Parallel, and Cluster Computing (cs.DC)**; Databases (cs.DB)

Cite as: [arXiv:2103.10169 \[cs.DC\]](#)

(or [arXiv:2103.10169v1 \[cs.DC\]](#) for this version)

<https://arxiv.org/abs/2103.10169>

```
private static Pipeline buildPipeline() {
    Pipeline p = Pipeline.create();
    p.readFrom(TestSources.itemStream(100, (ts, seq) -> nextRandomNumber()))
        .withIngestionTimestamps()
        .window(WindowDefinition.tumbling(1000))
        .aggregate(AggregateOperations.topN(TOP, ComparatorEx.comparingLong(l -> l)))
        .map(WindowResult::result)
        .writeTo(Sinks.observable(RESULTS));
    return p;
}
```

Hazelcast Jet Design Documents

<https://jet-start.sh/docs/design-docs/>

Hazelcast Jet Architecture

<https://jet-start.sh/docs/architecture/distributed-computing>

Tutorials

<https://training.hazelcast.com>

The screenshot displays a grid of six course cards, each with a title, a brief description, and an 'Enroll' button. The courses are categorized by color-coded sections: Development (orange), Operations (blue), and Enterprise (dark blue).

Section	Course Title	Description	Action
Development	Stream Processing Essentials	Course Time: 191 min – Free The basics of stream processing using Hazelcast Jet	Enroll
Operations	Migrating to IMDG 4.0	Free A guide to successfully migrating your applications to IMDG 4.0	Enroll
Enterprise	Hazelcast Security Features	Course Time: 27 min – Free Learn how to secure your Hazelcast clusters and clients	Enroll
Development	Distributed Concurrency	Course Time: 45 min – Free Concurrency in a distributed system? Here's how we do it.	Enroll
Operations	Hazelcast IMDG Overview	Course Time: 45 min – Enrolled A high-level overview of Hazelcast IMDG technology and operations	Resume
Operations	Hazelcast IMDG Configuration	Course Time: 22 min – Free How to configure Hazelcast IMDG	Enroll

Thank You!

Q/A time!

<https://pivovarit.github.io/talks/hazelcast>



Twitter: @pivovarit