HAZELCAST

Nurdan Kaya

29.07.2021

i2i Systems
innovation to integration

hazelcast

Content

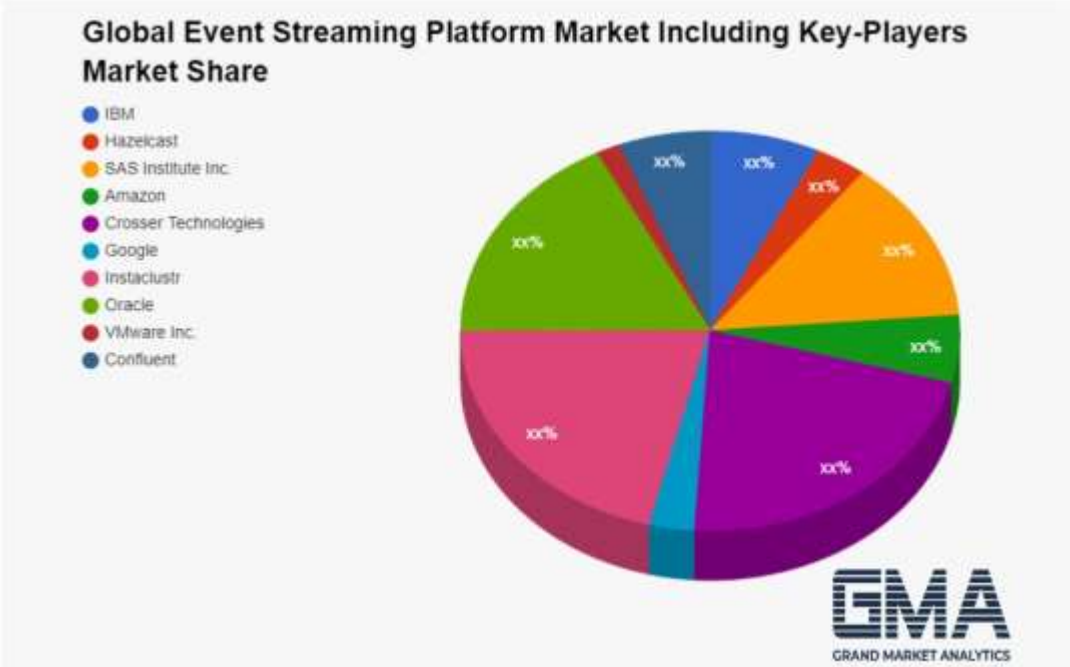
- Hazelcast statistics
- What is Hazelcast?
- In Memory Data Grid (IMDG)
- Hazelcast features
- Key components of Hazelcast
- Caching with Hazelcast
- Distributed data structures
- Data partitioning
- Data pipelines – Hazelcast Jet

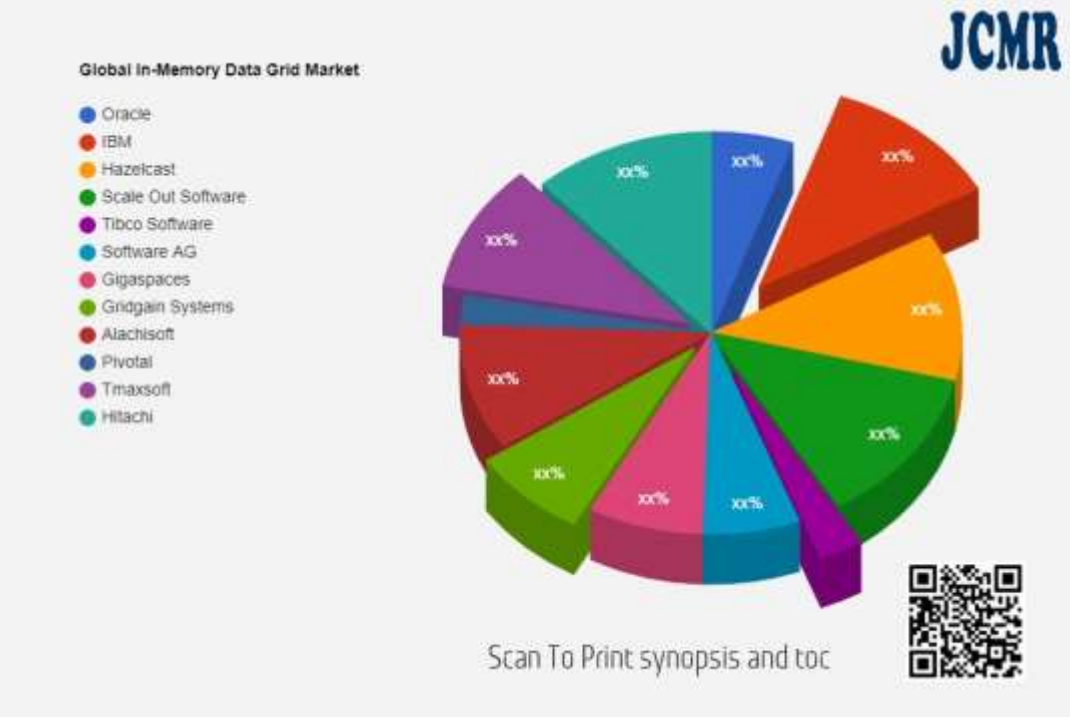
hazelcast

Hazelcast was founded by Enes Akar, Fuad Malikov and Talip Öztürk.

The companies choose Hazelcast

- **In-Memory Data Grid Market by Key Players**
  Gridgain Systems
  Alachisoft
  Oracle
  IBM
  Software AG
  Gigaspaces
  Pivotal
  Scale Out Software
  Tibco Software
  Hazelcast
  Hitachi
  Tmaxsoft





hazelcast®

What is HAZELCAST?

Distributed IN MEMORY DATA GRID(IMDG) platform

Hazelcast is a distributed computation and storage platform

Distributed implementation of java.util.(Queue, Set, List, Map)

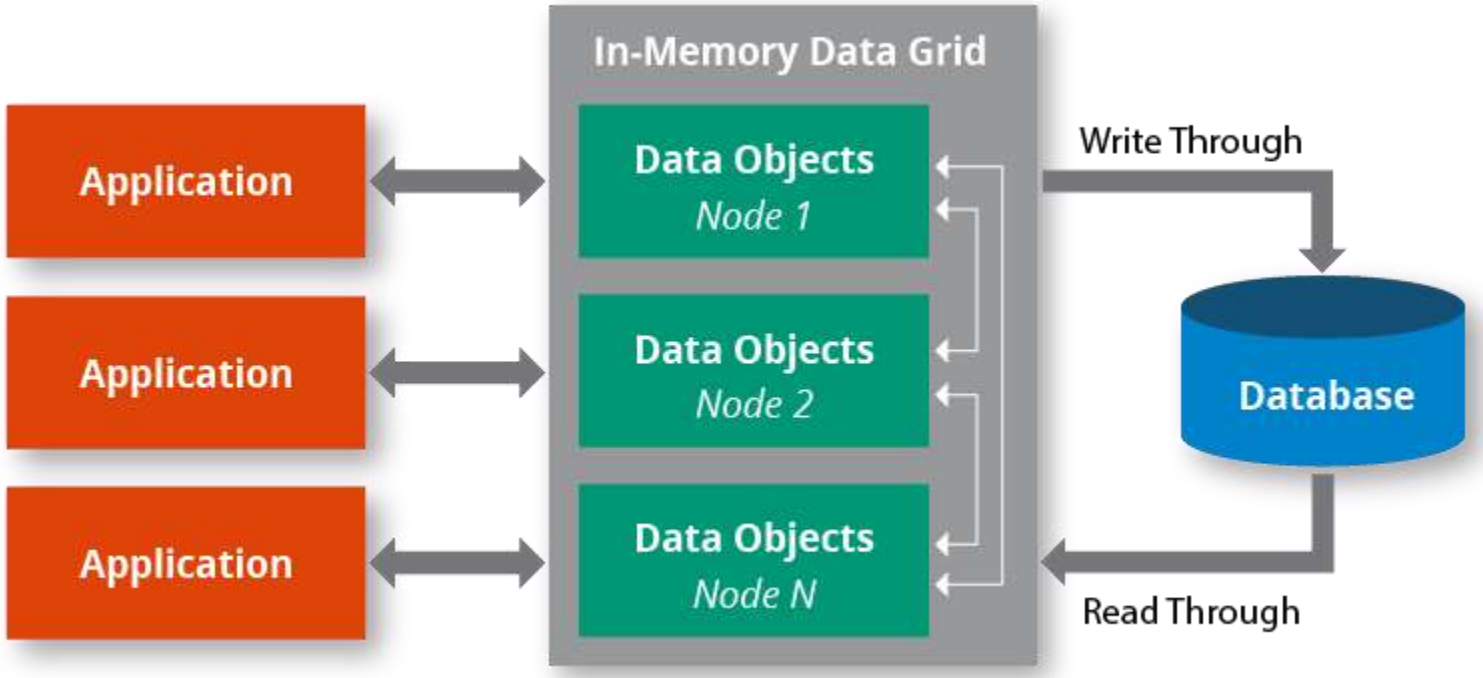
Scale Out

Not Scale Up


Parallelize Processing

IN MEMORY DATA GRID(IMDG)

IMDG is a set of networked/clustered computers that pool together their random access memory (RAM) to let applications share data with other applications running in the cluster.



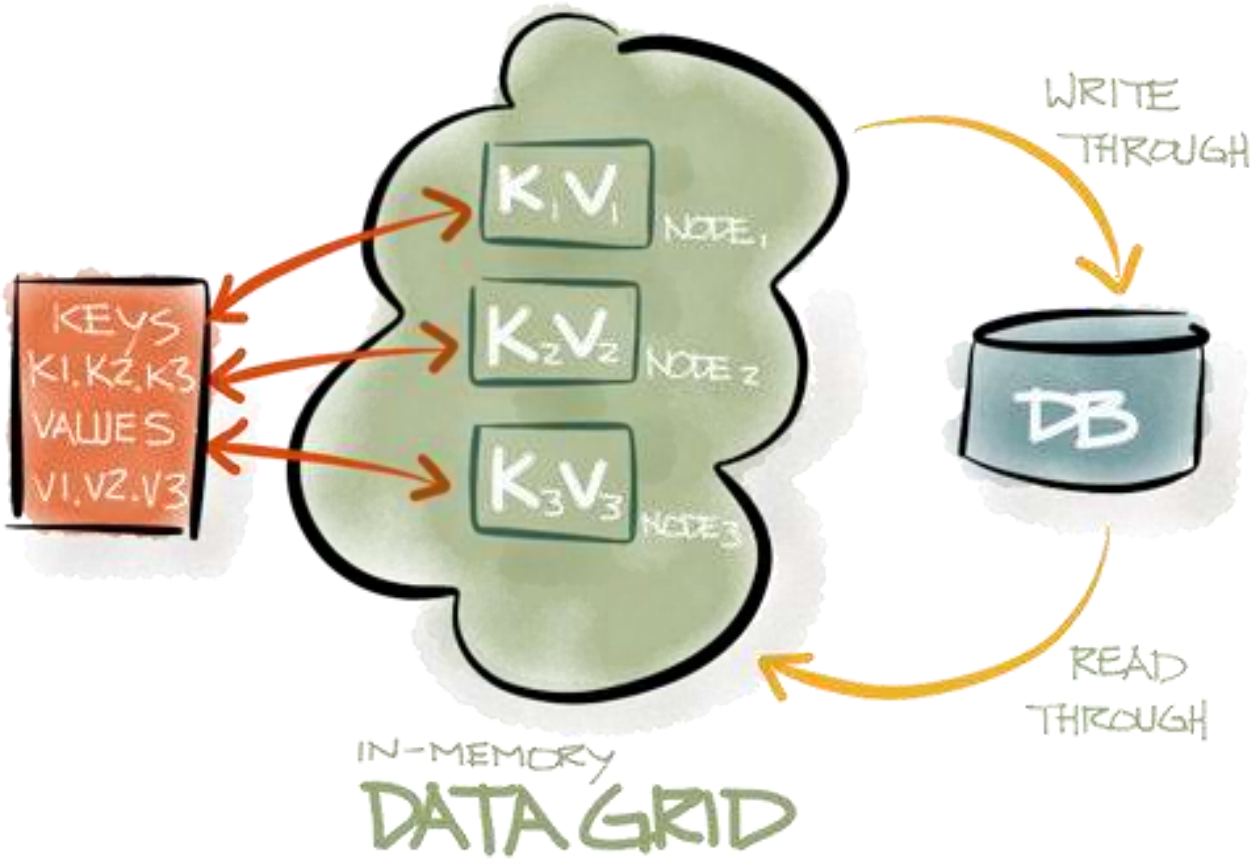
Shortly , IMDG is a distributed in-memory data store

(but, not just for storage …)

hazelcast®

IN MEMORY DATA GRID(IMDG)

- IMDGs are built for data processing at extremely high speeds.
- They are designed for building and running large-scale applications that need more RAM than is typically available in a single computer server
- This enables the highest application performance by using RAM along with the processing power of multiple computers that run tasks in parallel.
- IMDG is valuable for extensive parallel processing on large data sets.



hazelcast®

HAZELCAST FEATURES

- Open source
- Java based, simple
- **The data is always stored in-memory (RAM) of the servers**.
- Distributed computation, data structures, and events
- Streaming data processing
- Connectors to read from/write to systems like Apache Kafka, JMS, JDBC and HDMS
- Replication of web sessions (filter, Tomcat, Jetty based)
- …

hazelcast®

For more : Enterprise edition

HAZELCAST

Hazelcast is a data platform that integrates with your existing infrastructure to make your applications run faster.

Current Application Performance

Application Performance with **Hazelcast**

hazelcast®

KEY COMPONENTS

- **Member**
- **Cluster**
- **Partitions**
- **Streaming engine**
- **Storage engine**
- **Sources**
- **Sinks**
- **Connectors**

KEY COMPONENTS

Member

computational and data storage unit
(typically JVM)

Cluster

set of members communicating with each other

(Members which run Hazelcast automatically discover one another and form a cluster at runtime)

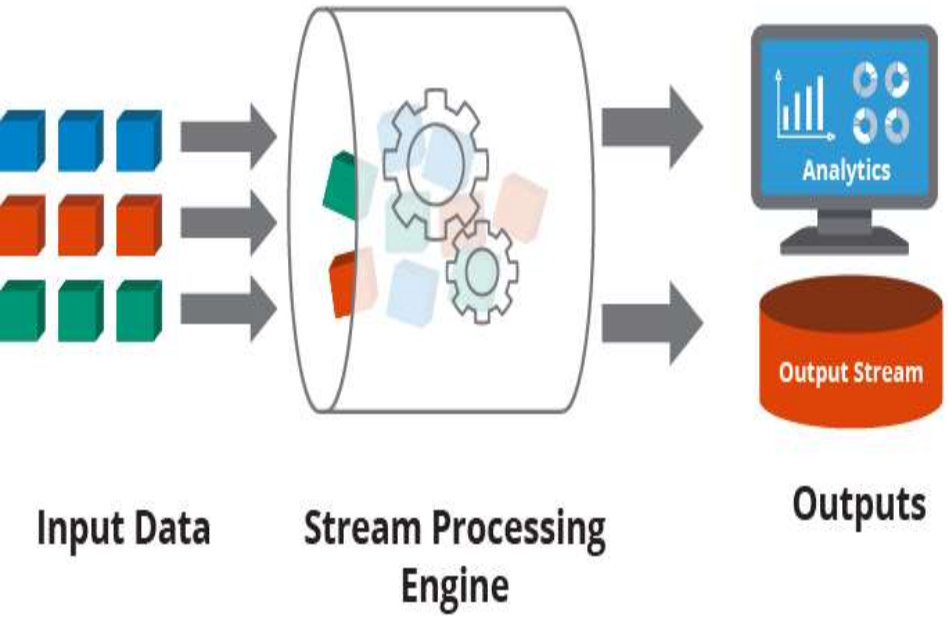
KEY COMPONENTS

Partitions

memory segments that store portions of data

- They can contain hundreds or thousands of data entries each, depending on the memory capacity of your system.
- Hazelcast automatically creates backups of these partitions which are also distributed in the cluster.

KEY COMPONENTS



Streaming engine

It focuses on data transformation while it does all the heavy lifting of getting the data flowing and computation running across a cluster of members.

It supports working with both bounded (batch) and unbounded (streaming) data.

KEY COMPONENTS

Storage engine

It is distributed, fast, and operational data store dealing with persistence of data.

Sources

Sources are where Hazelcast pulls the data

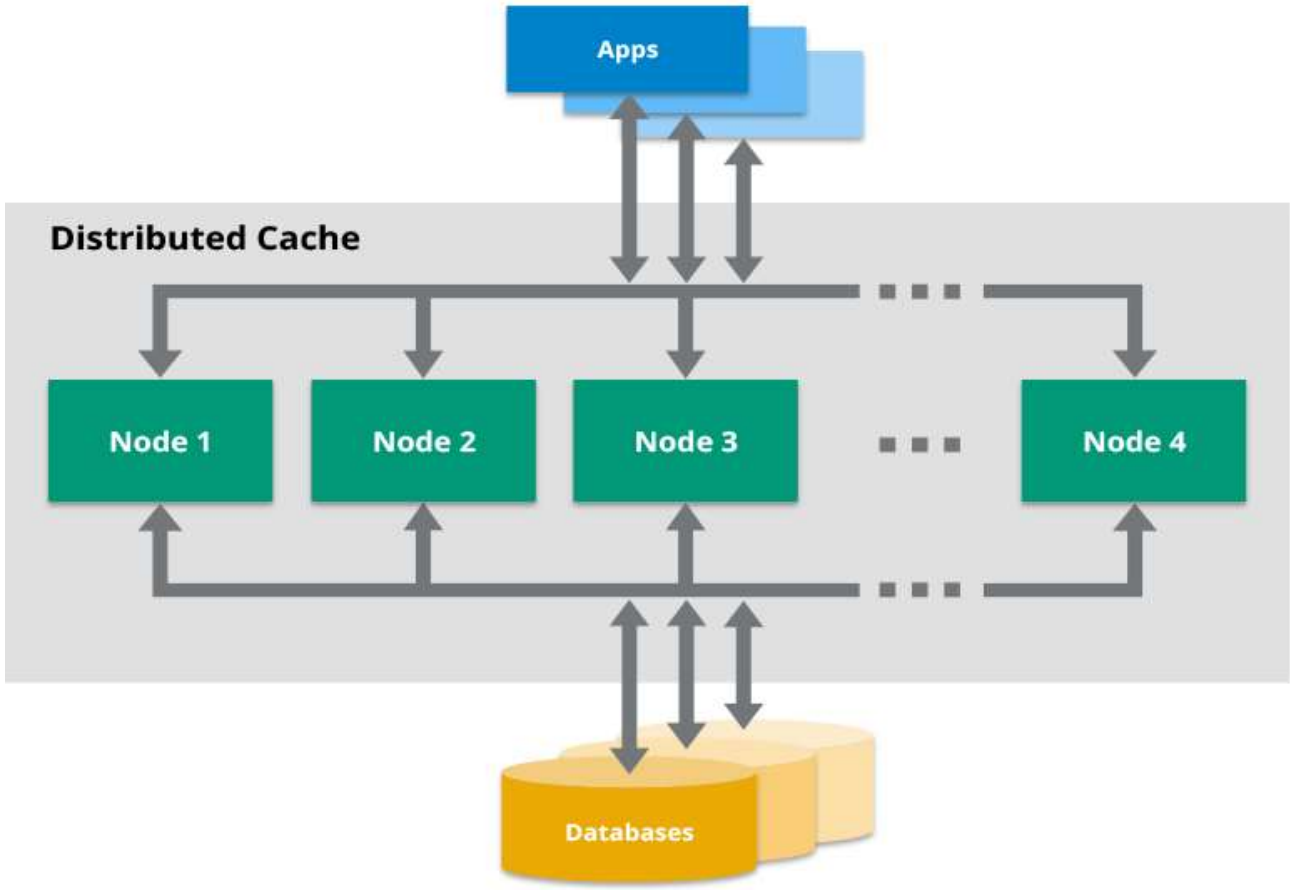
Connectors

Sinks

Sinks are where it outputs the processed data result

hazelcast®

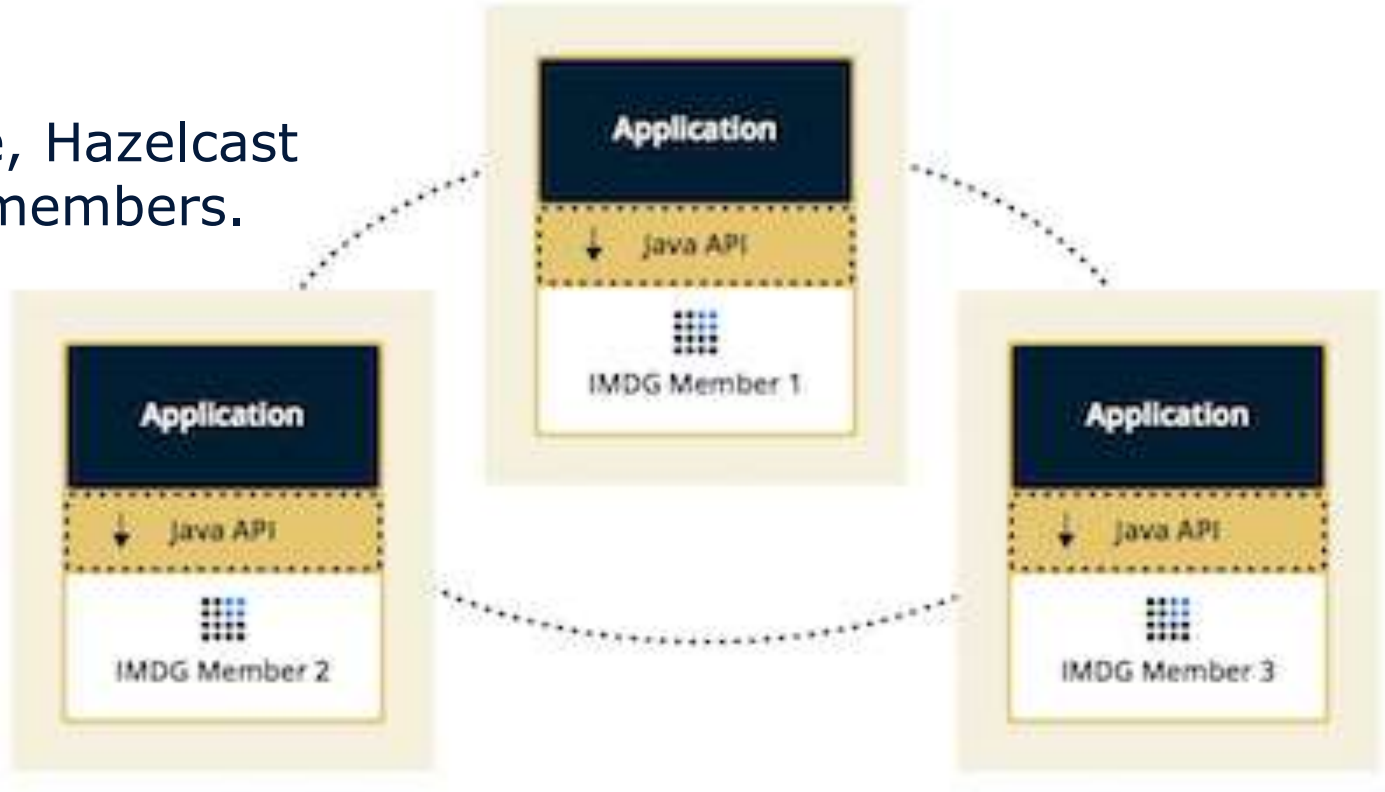
CACHING WITH HAZELCAST



hazelcast®

Caching Topologies – EMBEDDED MODE

- Application and the cached data are stored on the same device.
- When a new entry is written to the cache, Hazelcast takes care of distributing it to the other members.



\+ Data access is faster because applications don’t need to send a request to the cache cluster over the network.

\- It can be embedded only in Java apps. To use this mode, you must write application with Java.

hazelcast®

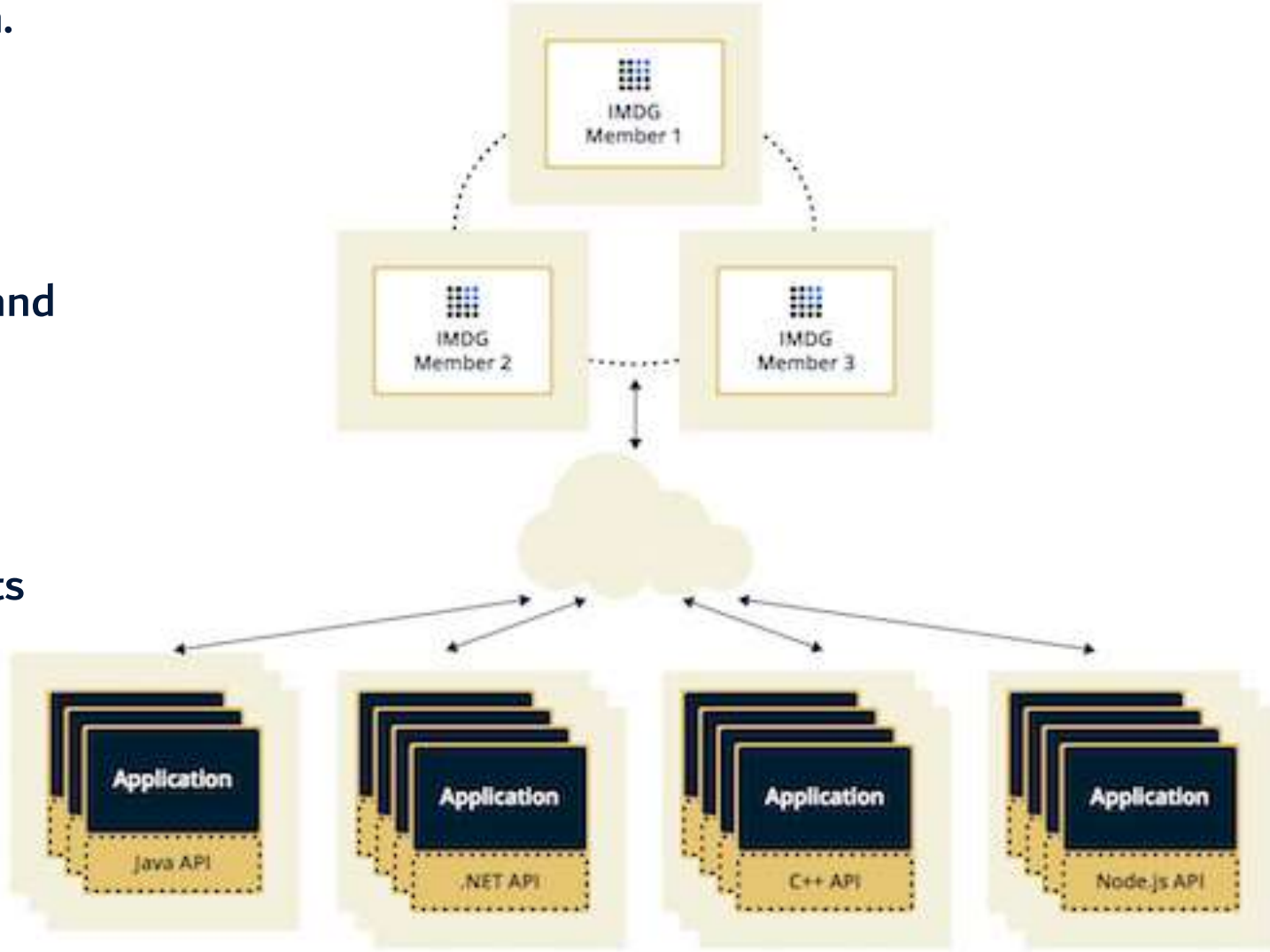
Caching Topologies – CLIENT/SERVER MODE

- Cached data is separated from the application. (Hazelcast cluster is independant of your application, which means that they can be independently created and scaled.)

- Hazelcast members run on dedicated servers and applications connect to them through clients.

\+ Supports independent scaling.

\- To read from a cache or write to it, clients need to make network requests, which leads to higher latency than embedded mode.



hazelcast®

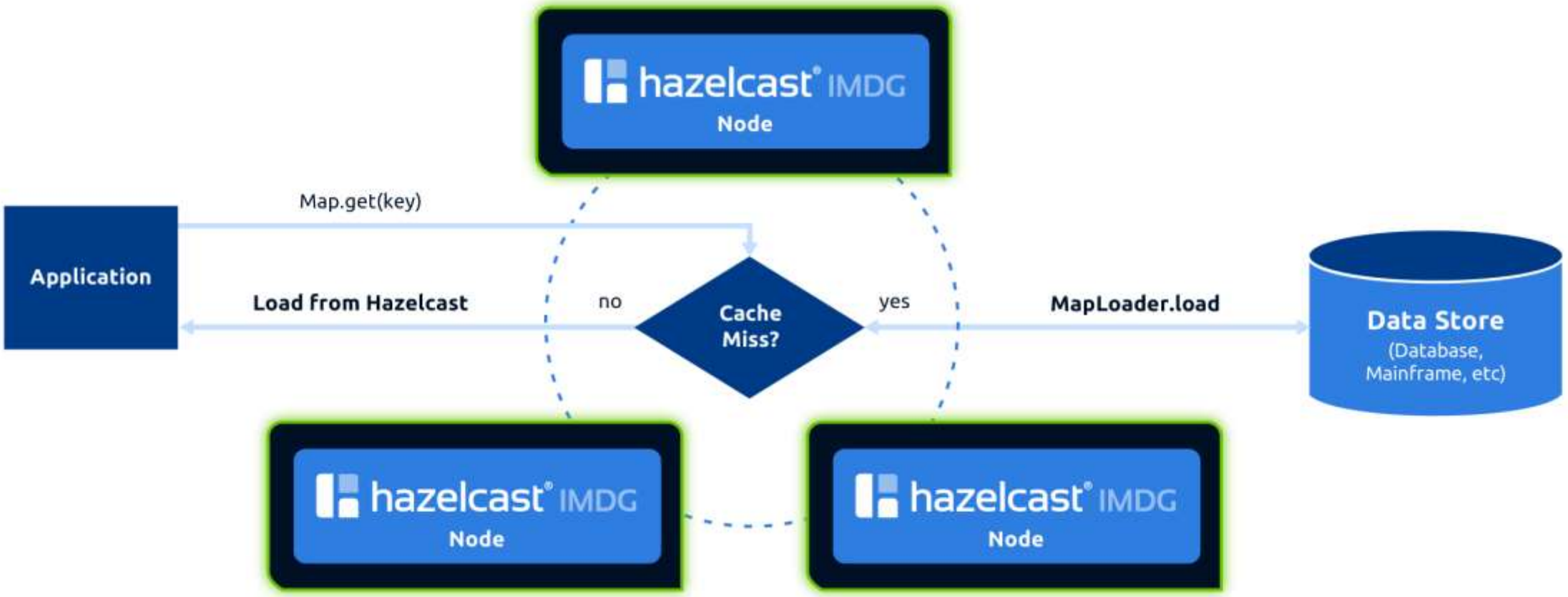
EMBEDDED VS CLIENT/SERVER

Table 1. Comparison of Hazelcast Modes

	Embedded	Client/Server
Low-latency	Yes	If used with Near Cache to store frequently used data in the client's local memory.
Scalability	The application and the cluster must be scaled together	You can scale the cluster independently of your application
Supported clients	Java	• C++ • C# • Go • Java • Memcache • Node.js • Python • REST

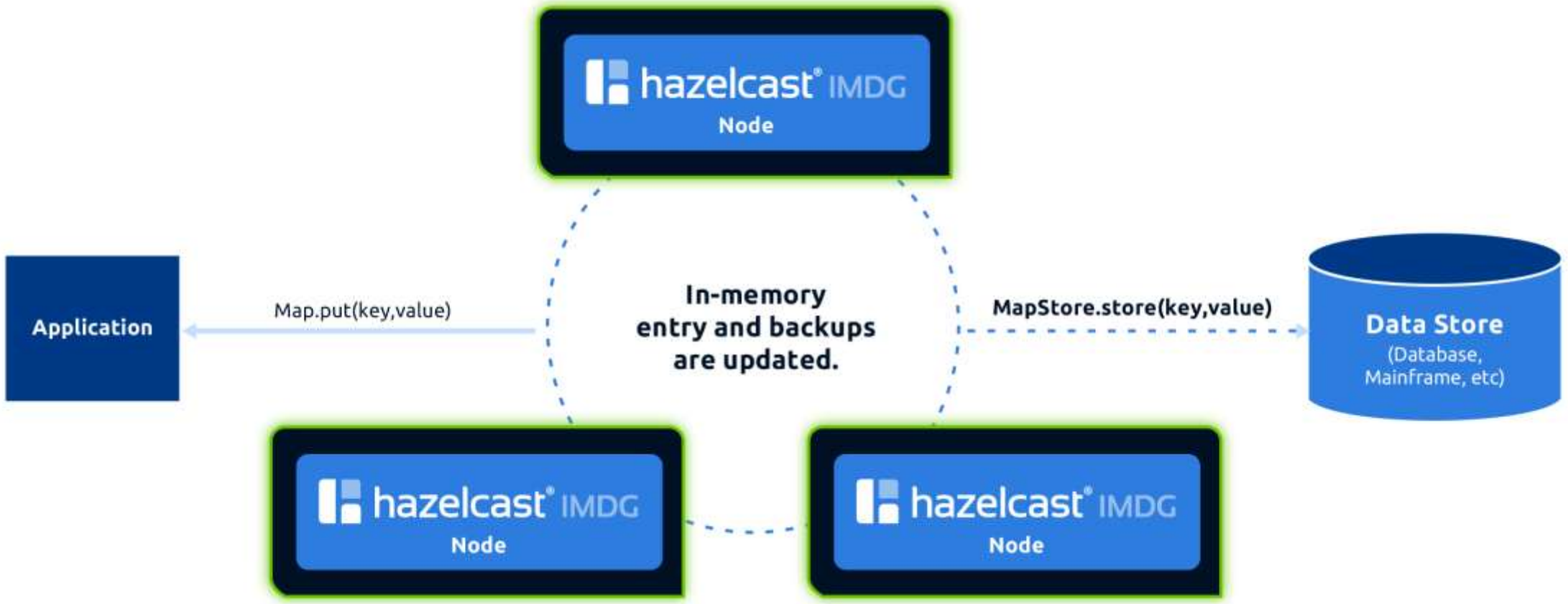
hazelcast®

Cache Access Patterns



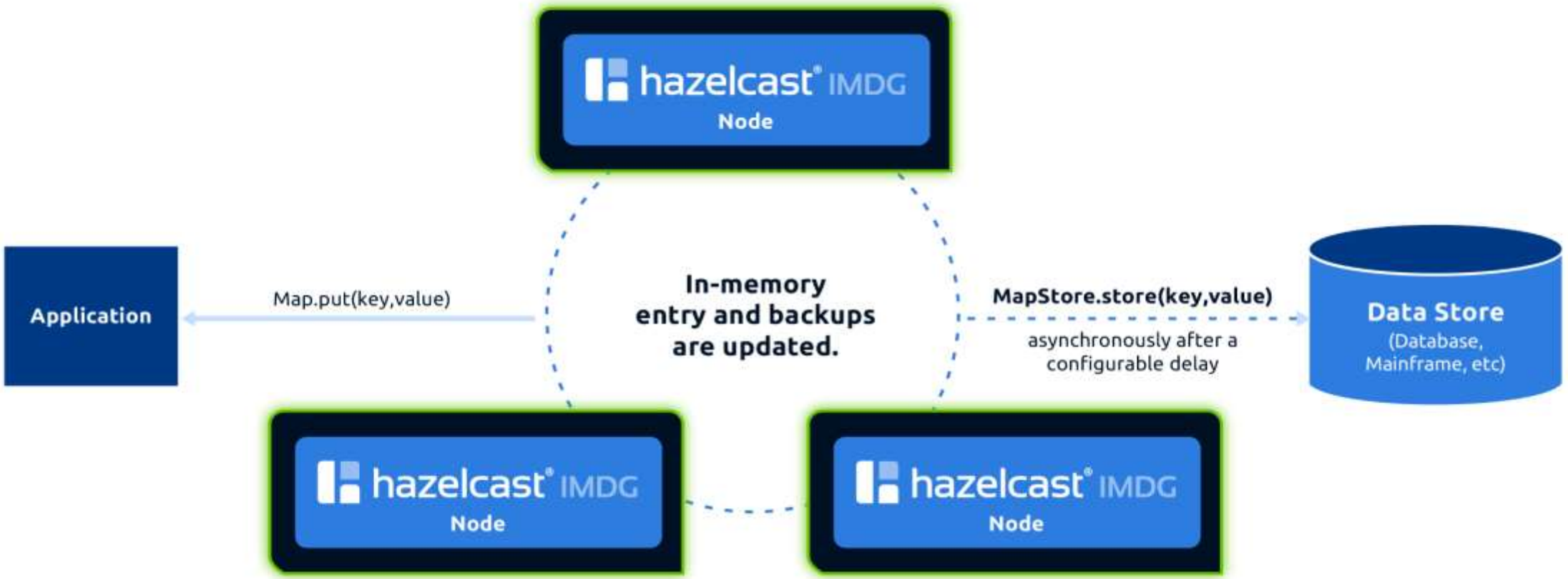
read-through-cache

Cache Access Patterns



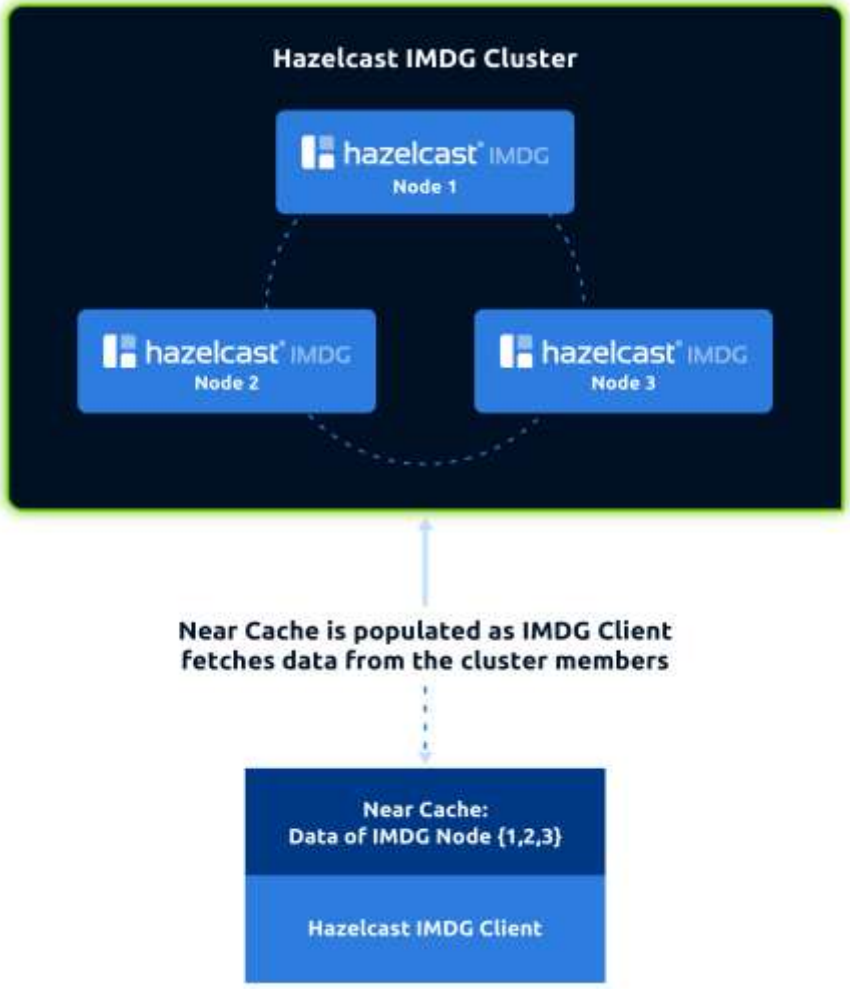
write-through-cache

Cache Access Patterns



Write-Behind Cache

Cache Access Patterns



Near Cache

Distributed Data Structures

Hazelcast has two types of distributed objects in terms of their partitioning strategies:

1. Data structures where each partition stores a part of the instance, namely **partitioned** data structures.

2. Data structures where a single partition stores the whole instance, namely **non-partitioned** data structures.

partitioned Hazelcast data structures:

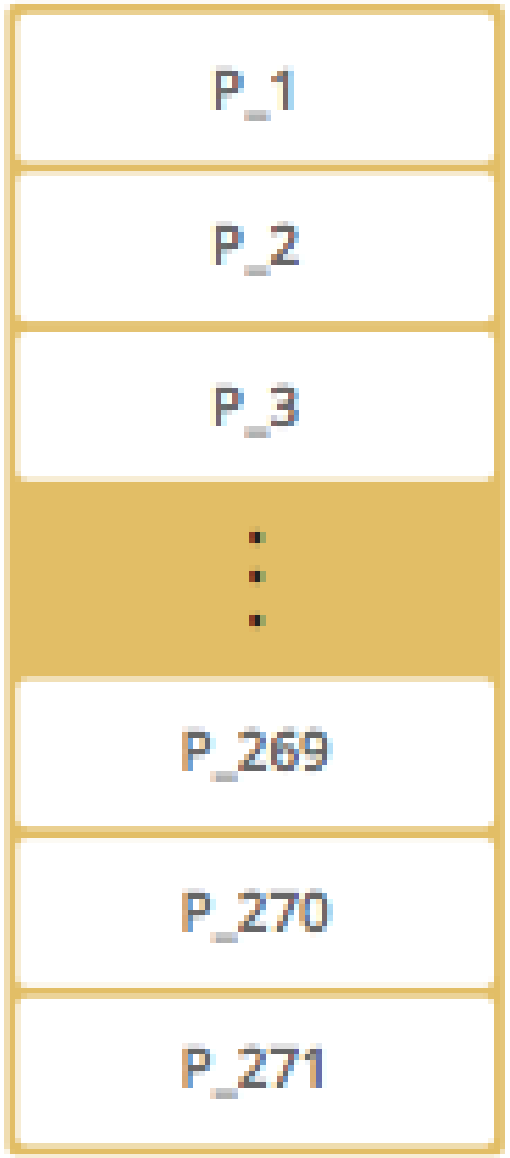
- Map
- MultiMap
- Cache (Hazelcast JCache implementation)
- Event Journal

non-partitioned Hazelcast data structures:

- Queue
- Set
- List
- Ringbuffer
- FencedLock
- ISemaphore
- IAtomicLong
- IAtomicReference
- FlakeIdGenerator
- ICountdownLatch
- Cardinality Estimator
- PN Counter

DATA PARTITIONING

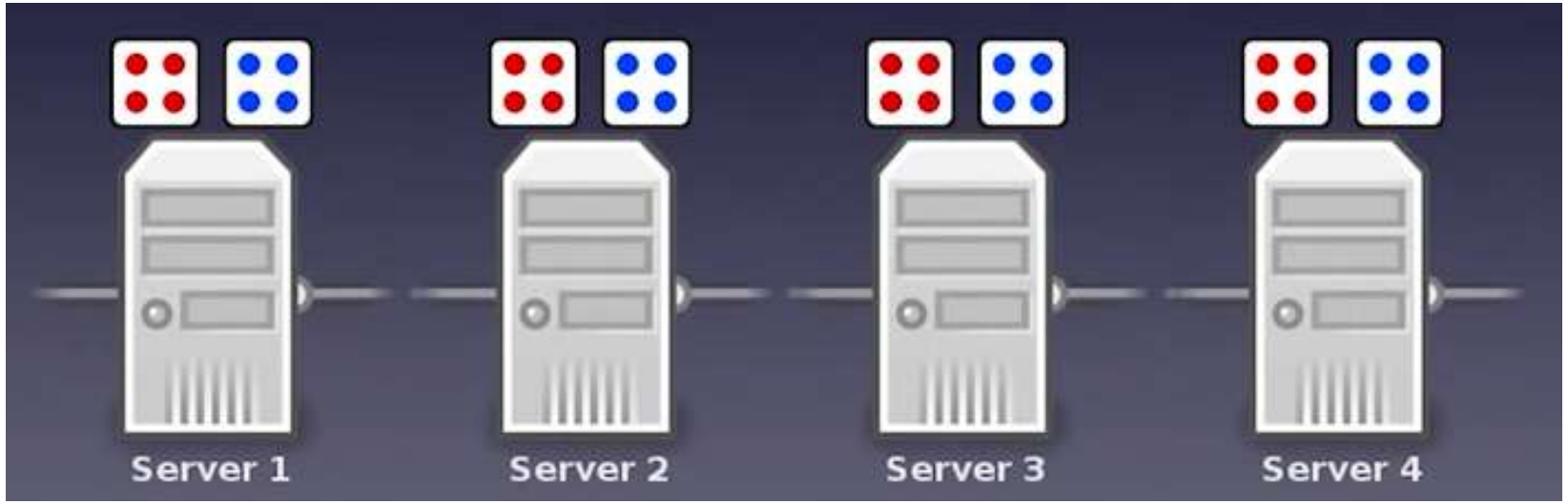
- The memory segments in Hazelcast IMDG = partitions
- Size of the partitions, i.e., amount of data entries they can store, is limited by the physical capacity of your system.
- The partitions are distributed equally among the members of the cluster.
- By default, Hazelcast creates a single copy/replica of each partition.
  - One of these replicas --> **PRIMARY**
  - Others --> **BACKUPS**
- The cluster member which owns the "primary" replica of a partition is called the --> **"partition owner"**

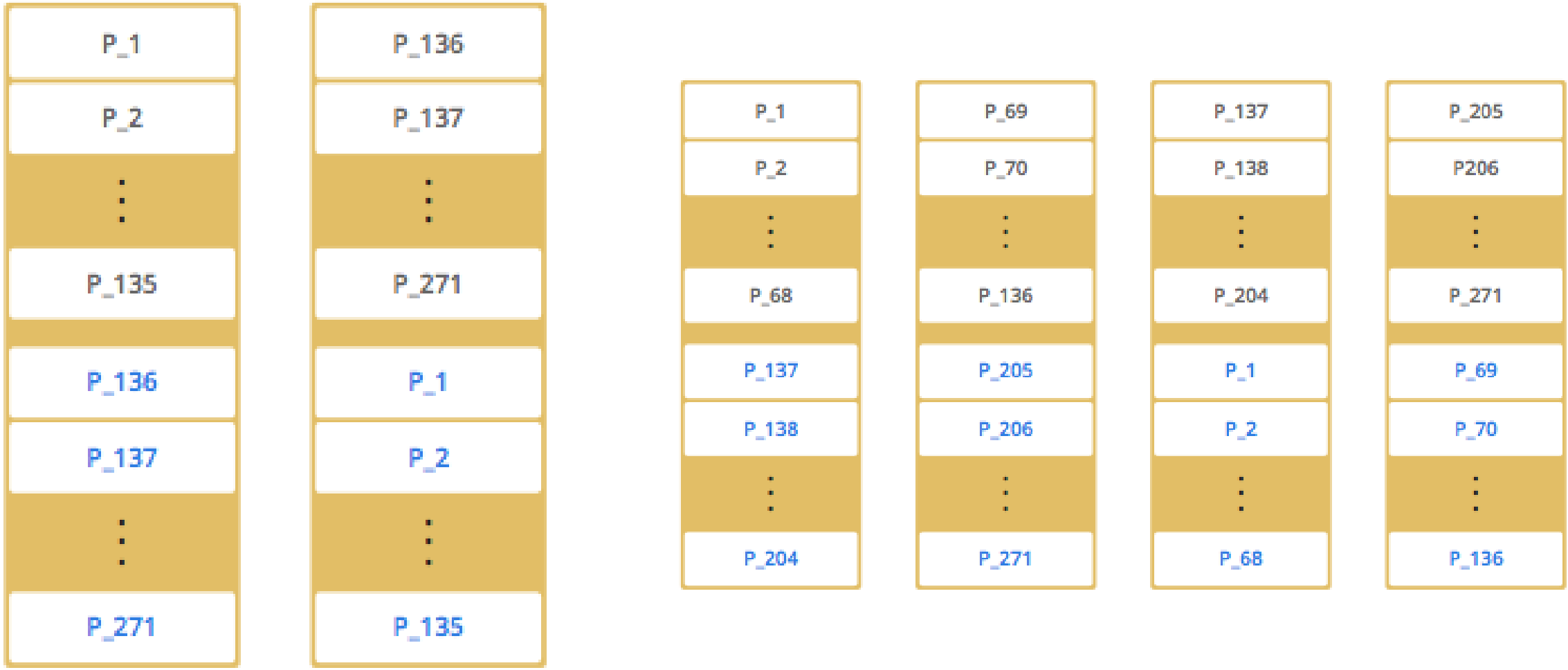


hazelcast®

DATA PARTITIONING

With 4 cluster nodes every server holds ¼ real data and ¼ backups




P_1
P_2
P_135
P_136
P_137
P_271
P_136
P_137
P_271
P_1
P_2
P_135
P_1
P_2
P_68
P_137
P_138
P_204
P_69
P_70
P_136
P_205
P_206
P_271
P_137
P_138
P_204
P_1
P_2
P_68
P_205
P206
P_271
P_69
P_70
P_136

DEMO

DATA PIPELINES

- Pipeline is a series of steps for processing data

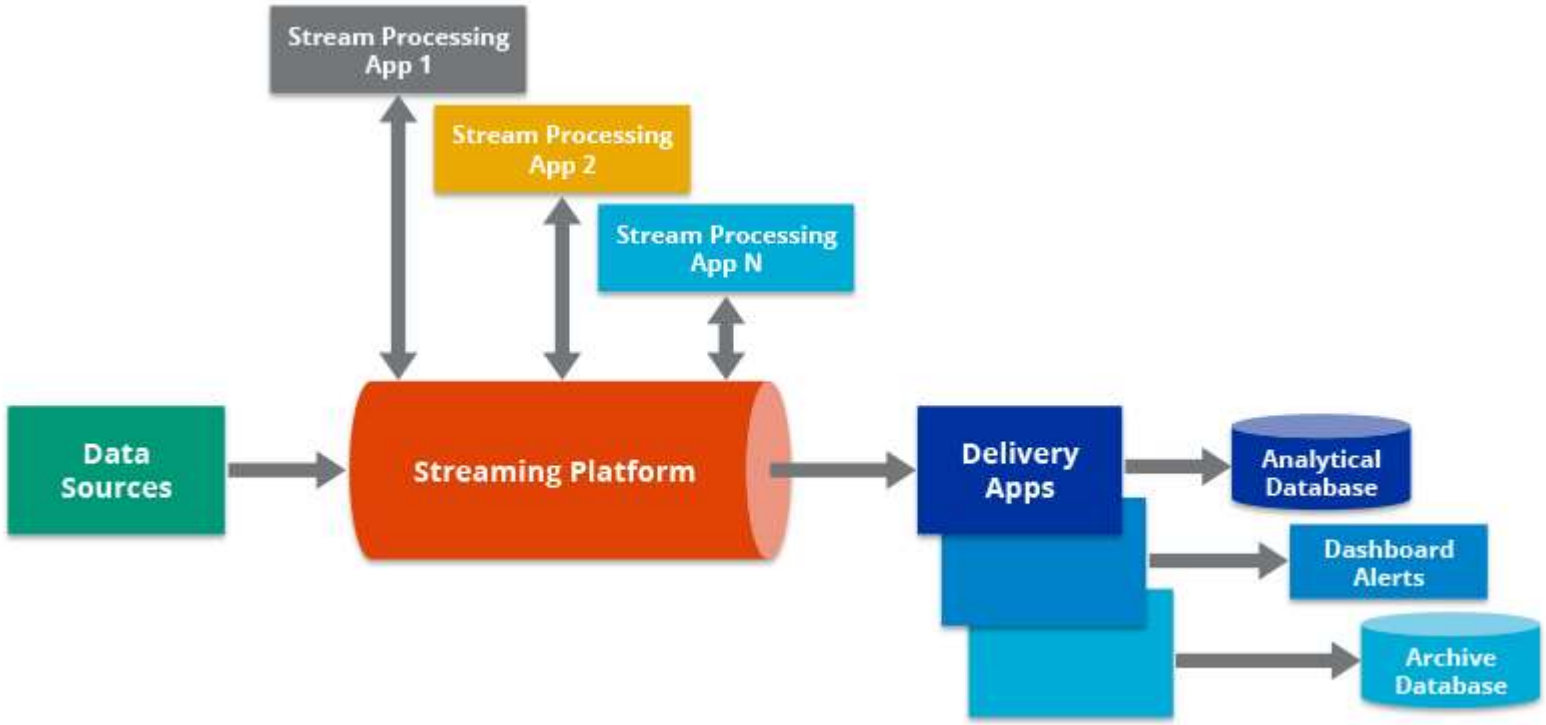
Pipeline consists of three elements:

- One or more sources: Where you take your data from.
- Processing stages: What you do to your data.
- At least one sink: Where you send the results of the last processing stage.

DATA PIPELINES

- Pipelines allow you to process data that's stored in one location and send the result to another such as from a data lake to an analytics database, or into a payment processing system.

- Depending on the data source, pipelines can be used for the following use cases:
  - **Stream processing**: Processes an endless stream of data such as events to deliver results as the data is generated.
  - **Batch processing**: Processes a finite amount of static data for repetitive tasks such as daily reports.



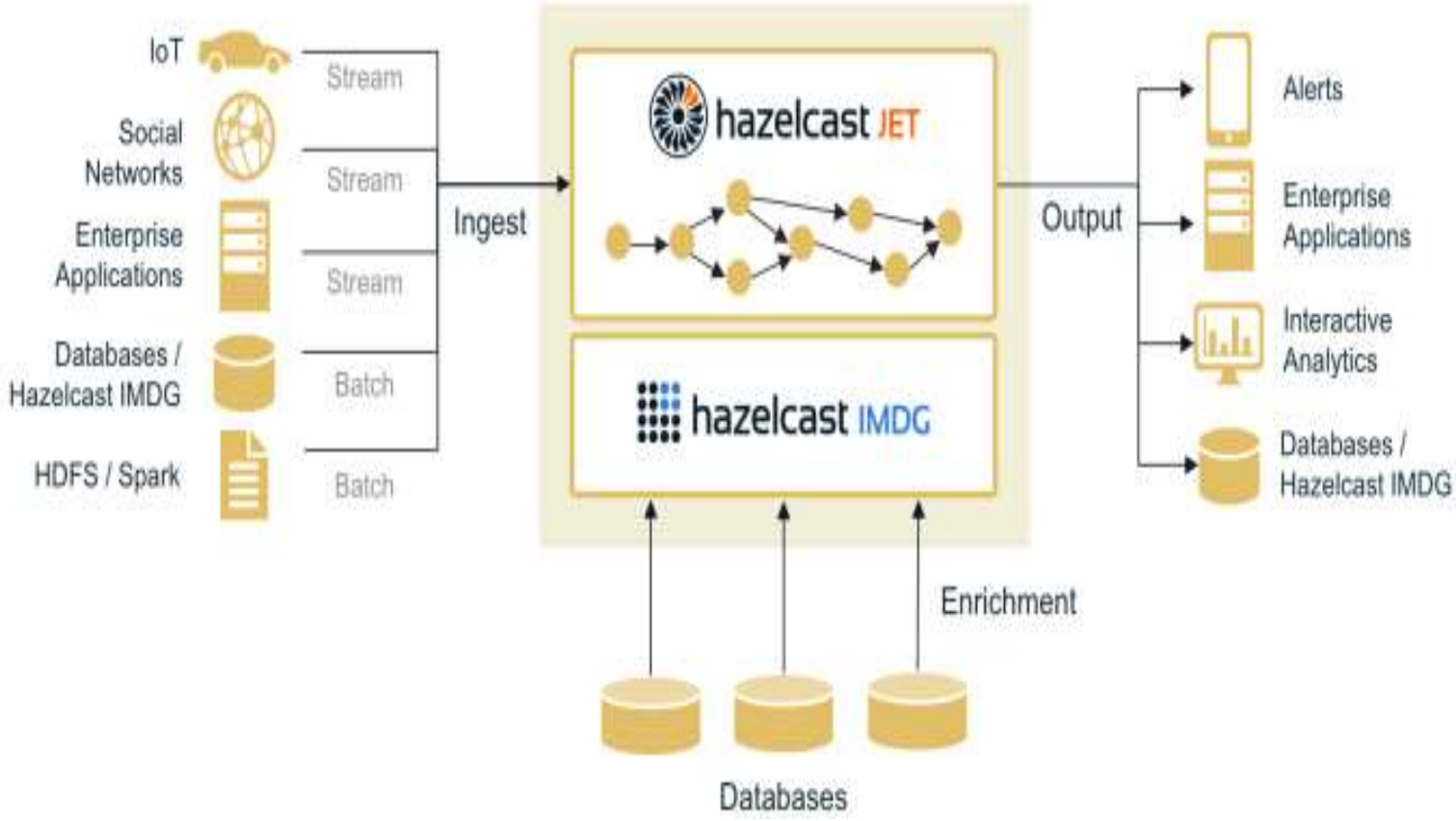
hazelcast®

Jet Engine

- The Jet engine is a batch and stream processing system that allows Hazelcast members to do both stateless and statefu computations over large amounts of data with consistent low latency.

- Jet integrates out of the box with many popular data storage systems such as Apache Kafka, Hadoop, relational databases, message queues and many more.

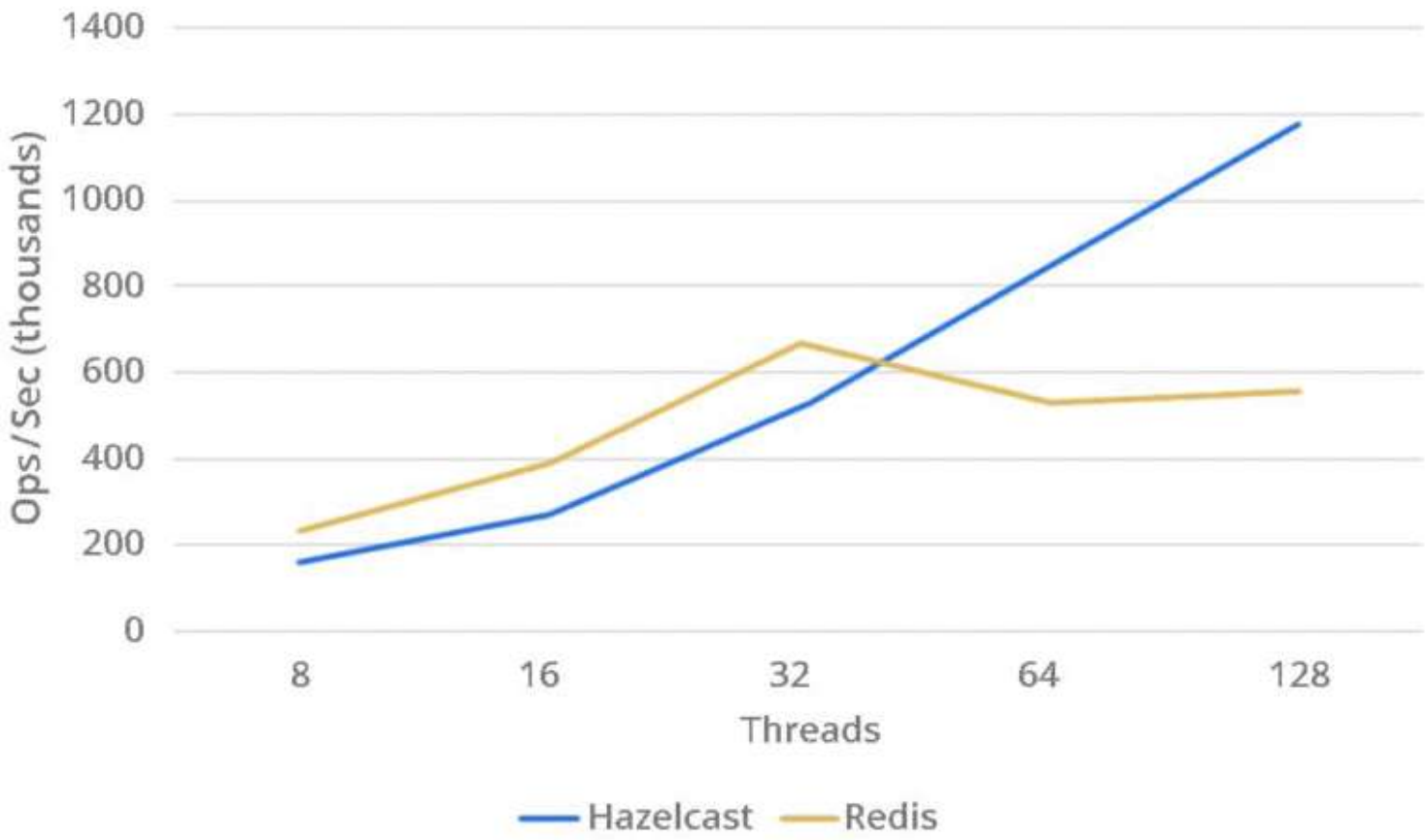
- Jet also comes with a fully-featured, in-memory key-value store. Use it to cache results, store reference data or as a data source itself.



HAZELCAST USE CASES

- Increasing the transactions per second and system uptime in payment processing
- Authorizing and authenticating the credit cards using multiple algorithms within milliseconds for fraud detection
- Decreasing order processing times with low latencies in e-commerce
- Being a real-time streamer to detect application performance
- Clustering highly changing data with event notifications, e.g., user based events, and queueing and distributing background tasks
- Being a distributed topic (publish/subscribe server) to build scalable chat servers for smartphones
- Increasing the transactions per second and system uptime in payment processing

Hazelcast vs Redis



hazelcast®

References

- https://hazelcast.com/
- https://www.youtube.com/watch?v=66Mb72btt2E
- https://www.youtube.com/watch?v=DL405QMJ8M8&ab_channel=IstanbulCoders
- https://www.javainuse.com/hazelcast

ANY QUESTİONS?

Thank you ！ (: