Java Forum Stuttgart

Databaseless

Java In-Memory Data Processing

Modern Applications:

1) High Performance

2) Low Data Storage Costs

3) Simplicity

4) Sustainability

About me

Markus Kett
CEO at MicroStream

Contributing to Open Source Projects:
- RapidClipse – Java Low-Code DIE
- JPA-SQL
- Eclipse Serializer
- EclipseStore
- Project Helidon
- Micronaut

Editor in Chief at JAVAPRO Magazine
Founder of JCON Conference
Speaker, Author

X: @MarkusKett
LinkedIn: markuskett
Email: m.kett@microstream.one

MicroStream

EclipseStore

RAPID clipse™

JPA-SQL

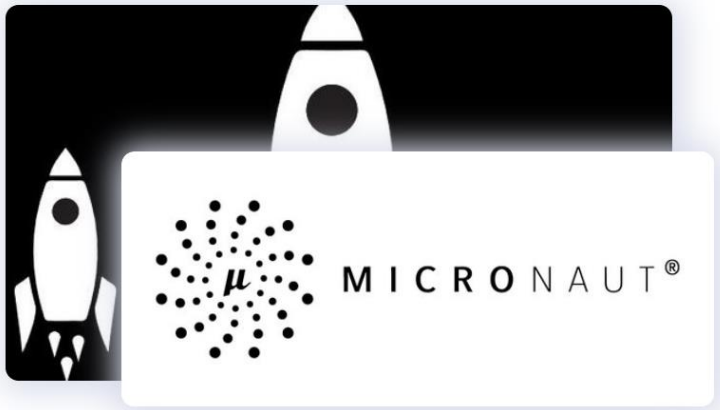
JAVAPRO

JCON 2021

About MicroStream

About MicroStream

We are Foundation Member



We Contribute to Open Source Projects

EclipseStore

helidon.io

MICRONAUT®

RAPID clipse™

helidon.io

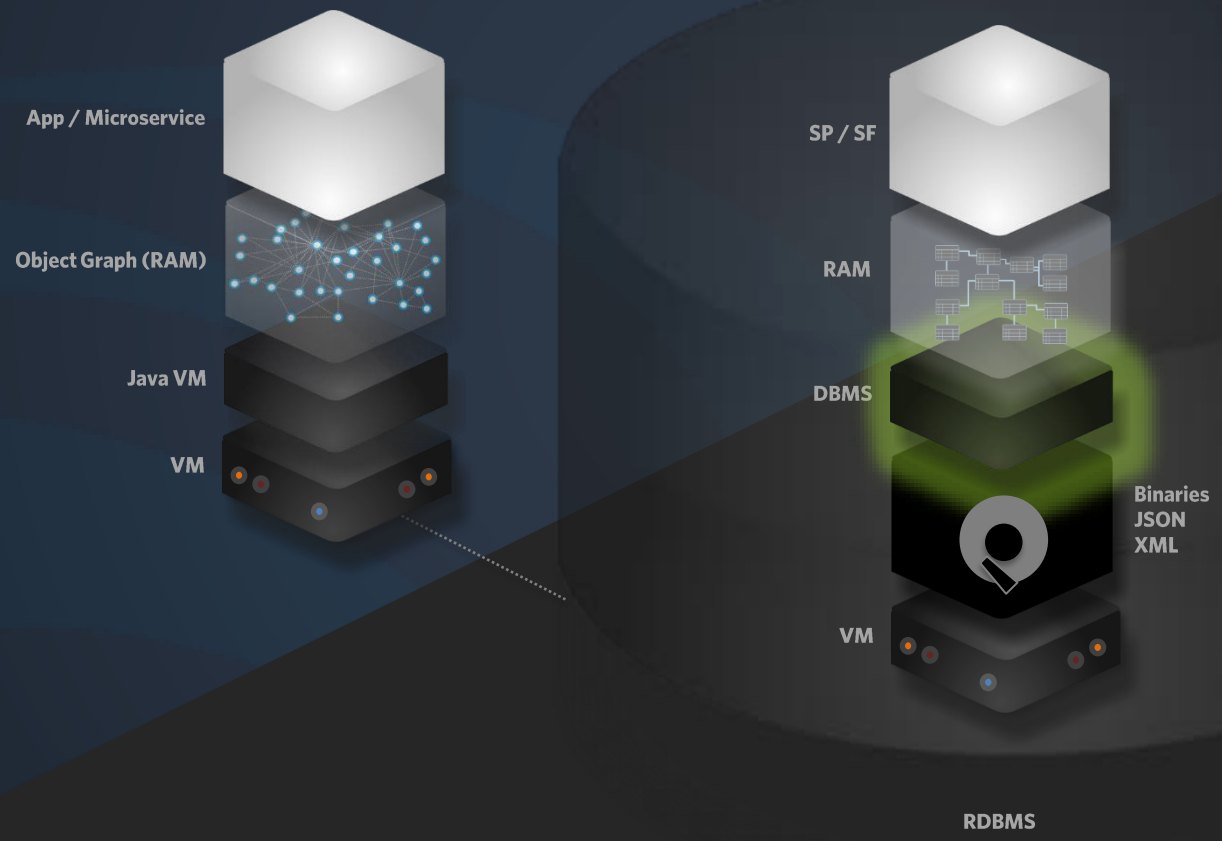
EclipseStore is Part of Helidon

helidon.io + EclipseStore

Helidon is a fast framework for developing modern cloud-native microservices with Java. Helidon is mainly developed by Oracle.

Challenges With Data Processing of Today

Database Server Concept

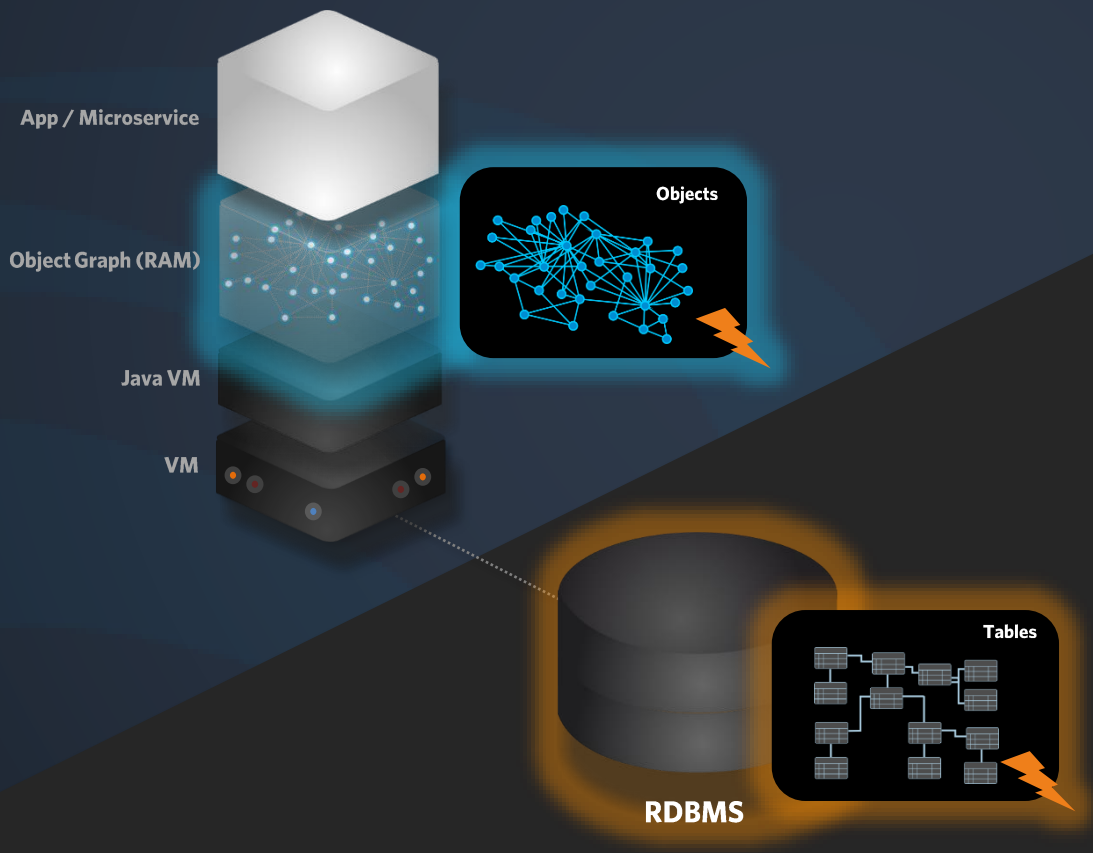


Note for later: A DB server is a big monolith, stateful and requires significant CPU power & expensive RAM.

Interestingly, the Java standard (JPA) ignores powerful database features such as SP & SF. So, what DB features do we really need in Java, especially in microservice architectures?

A Database System Behind the Scenes.

Java and Relational Databases are Incompatible



Impedance Mismatches

- Granularity mismatch
- Subtypes mismatch
- Identity mismatach
- Associations mismatch
- Data Navigation mismatch
- Data type mismatches

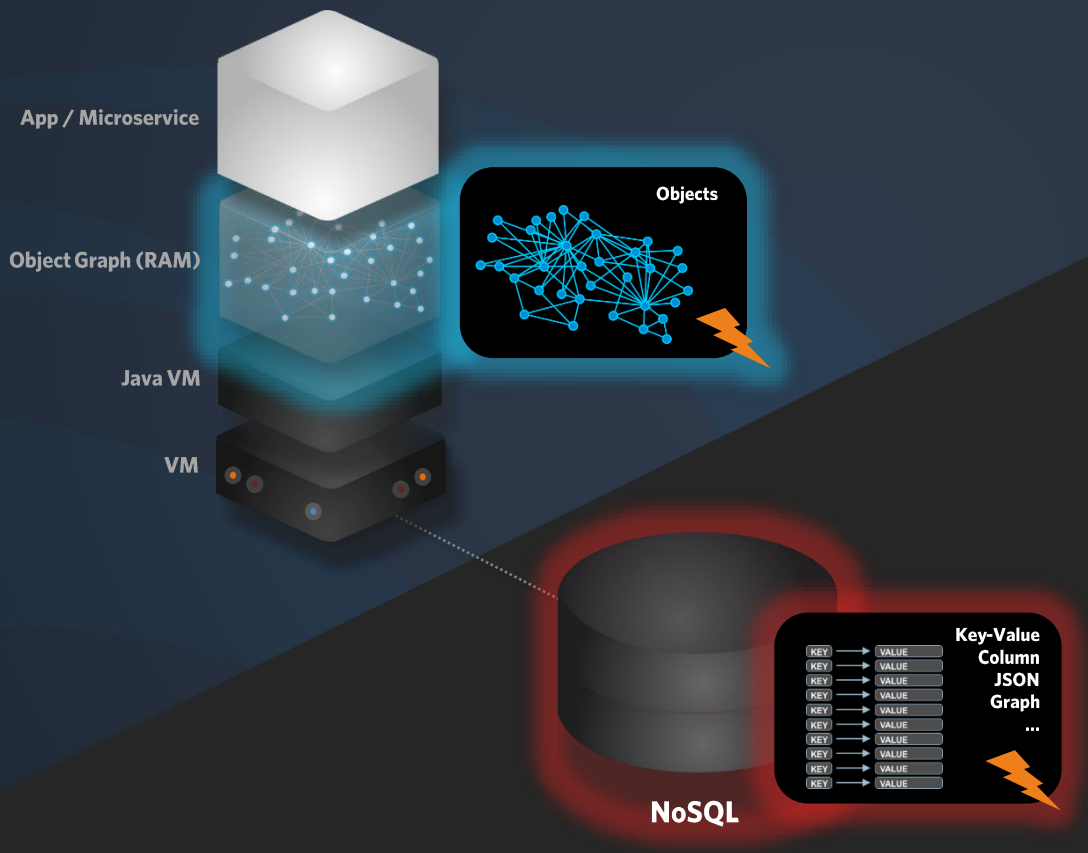
The Java object model and the realational model are incompatible. This is the core problem and leads to various technical problems, limitations & tradeoffs that can't be solved. All data solutions on the market provide only workarounds that are expensive.

The Problem of Incompatible Data Structures is Well Known as Impedance Mismatch

There are various solutions, but they are only a more or less elegant way around the problem. No matter which solution you choose - as long as the systems are different, every developer will sooner or later get to the point where his solution no longer meets one or more of the following points: Maintainability, performance, intelligibility.

https://en.wikipedia.org/wiki/Object%E2%80%93relational_impedance_mismatch

Java and NoSQL Databases are Incompatible

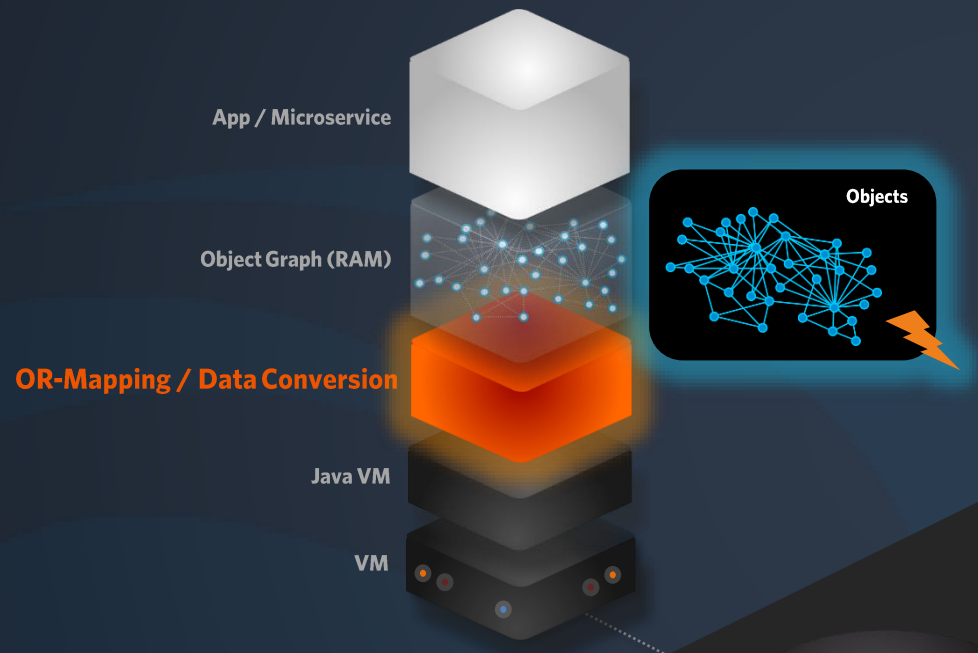


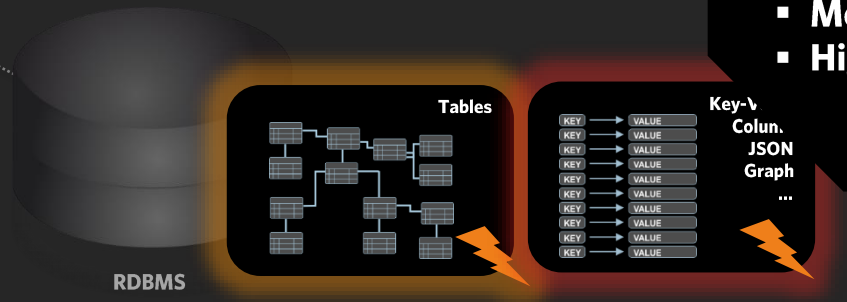
Impedance Mismatches

- **Key-Value**
- **Document** (JSON, XML)
- **Column Store**
- **Graph** (Proprietary)
- **OODB** (Proprietary)
- **Time-Series**
- **Vectors**

OODB provide an DB-specific object model, GraphDBs provide DB-specific graph model, document-based DBs provide a JSON format etc. NoSQL DBs introduce different data models and formats, but they're also incompatible with the Java object model.

Object-Relational Mapping / Data Conversion





Challenge: Storing Objects into Tables / JSON / Key Value Stores / Graphs

ORM / Data Conversion Through Every Single Query !

- Higher CPU consumption
- Higher energy consumption
- Higher CO2 emission
- Higher infrastructure costs
- Expensive latencies
- More complex architecture
- Higher costs of development

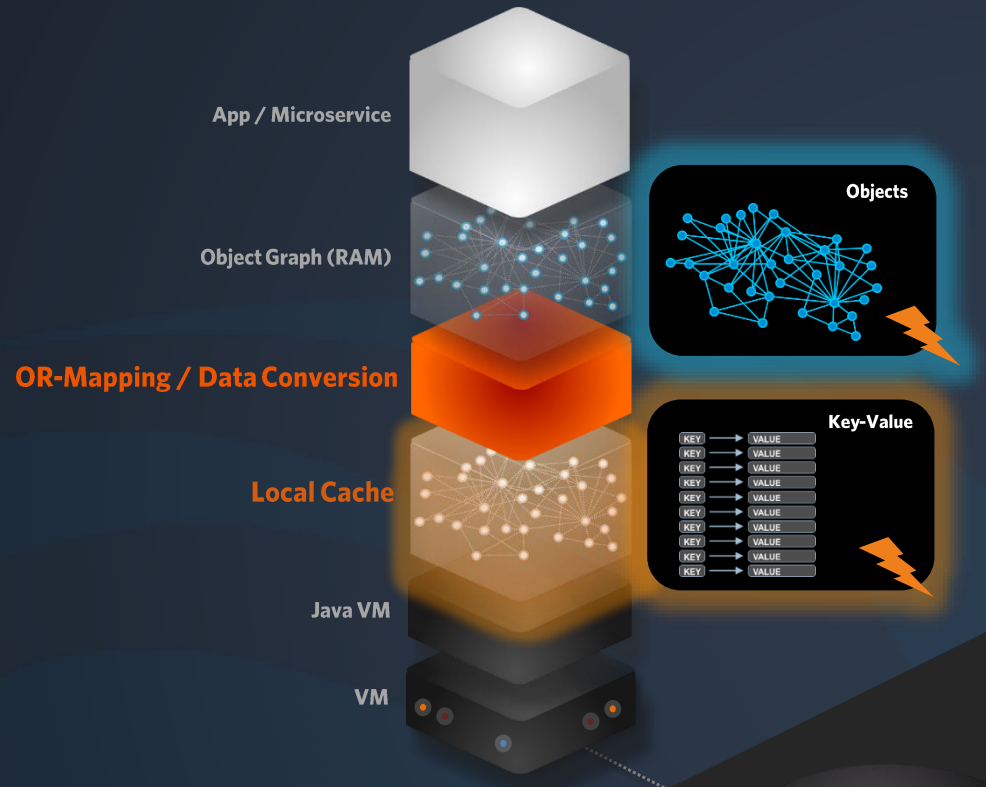
Application Layer

Storage Layer

Millisecond

Query Time

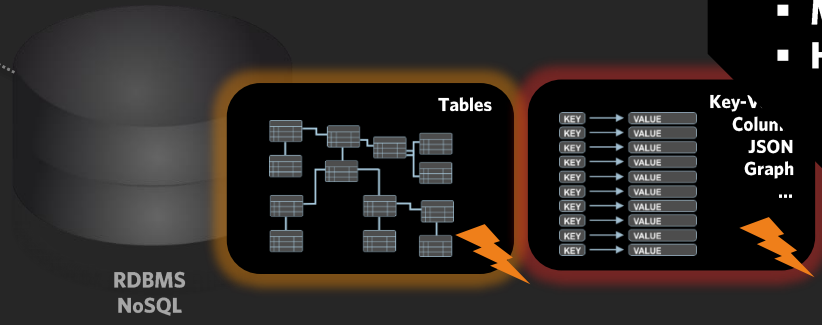
Local Cache: Now, Better Performance, but Higher Complexity



Challenge: Storing Objects into Tables / JSON / Key Value Stores / Graphs

ORM / Data Conversion Through Every Single Query !

- Higher CPU consumption
- Higher energy consumption
- Higher CO2 emission
- Higher infrastructure costs
- Expensive latencies
- More complex architecture
- Higher costs of development



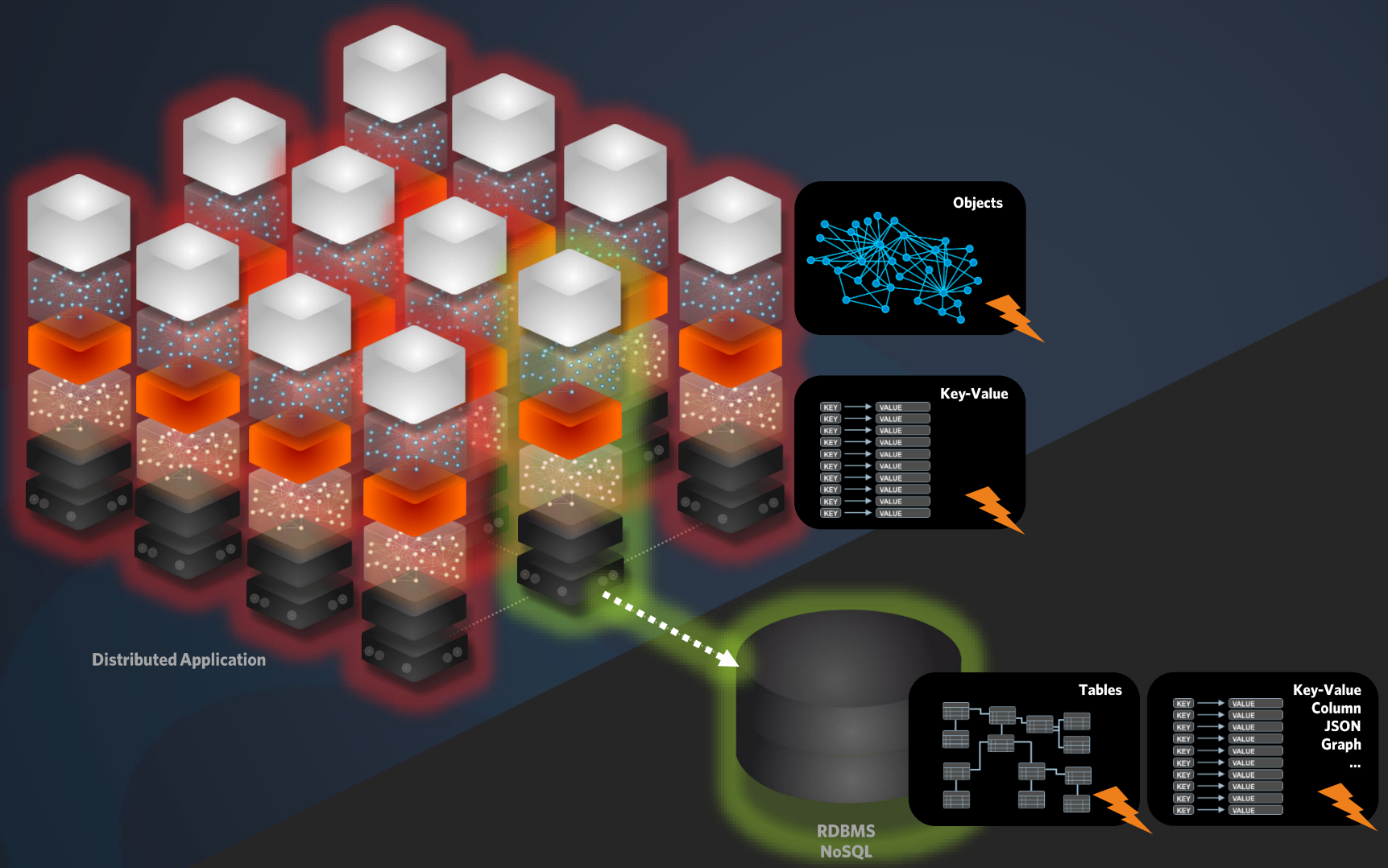
Application Layer

Storage Layer

Millisecond

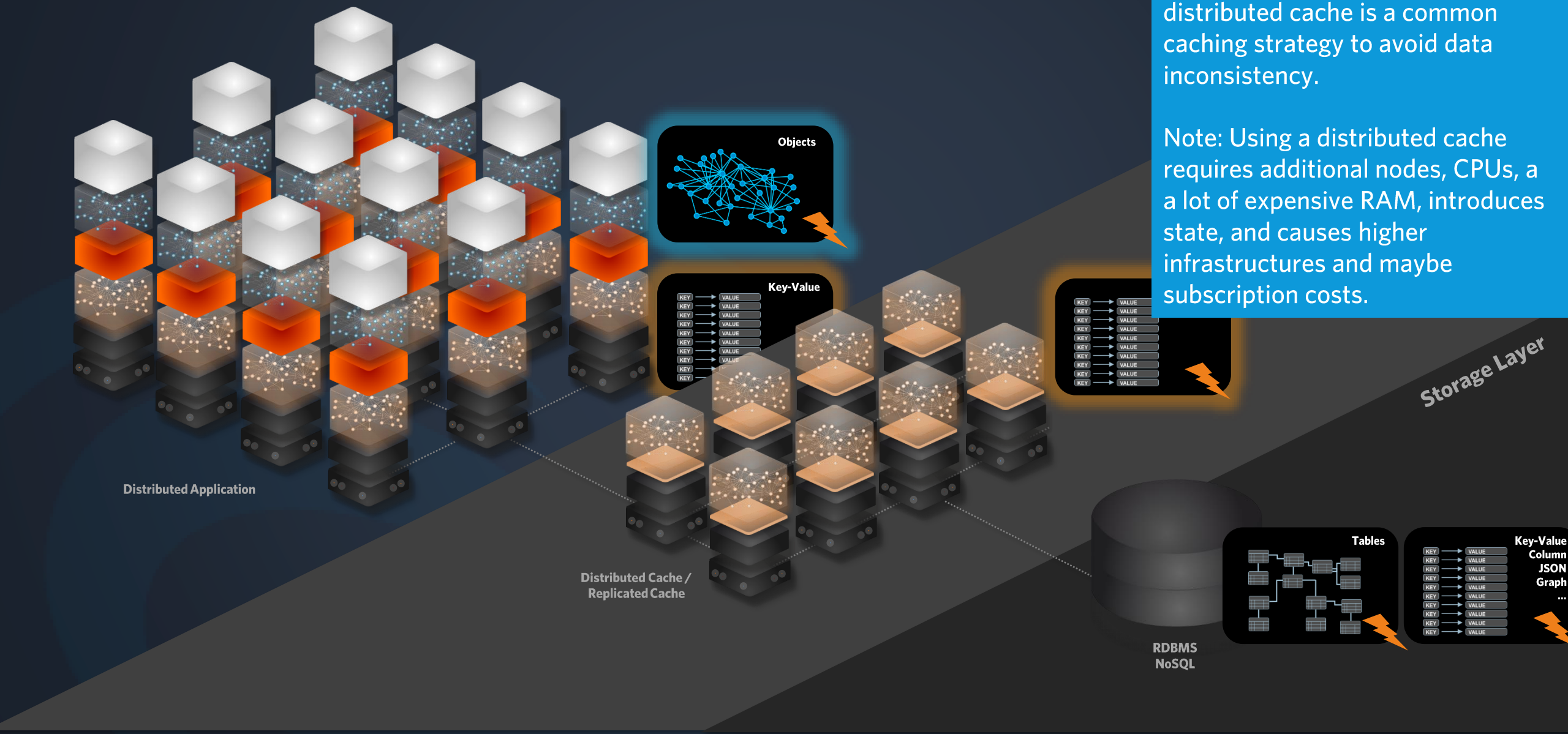
Query Time

Distributed Applications



In distributed systems, local caches can easily lead to data inconsistency when data is changed on a one of the app nodes.

Distributed Cache

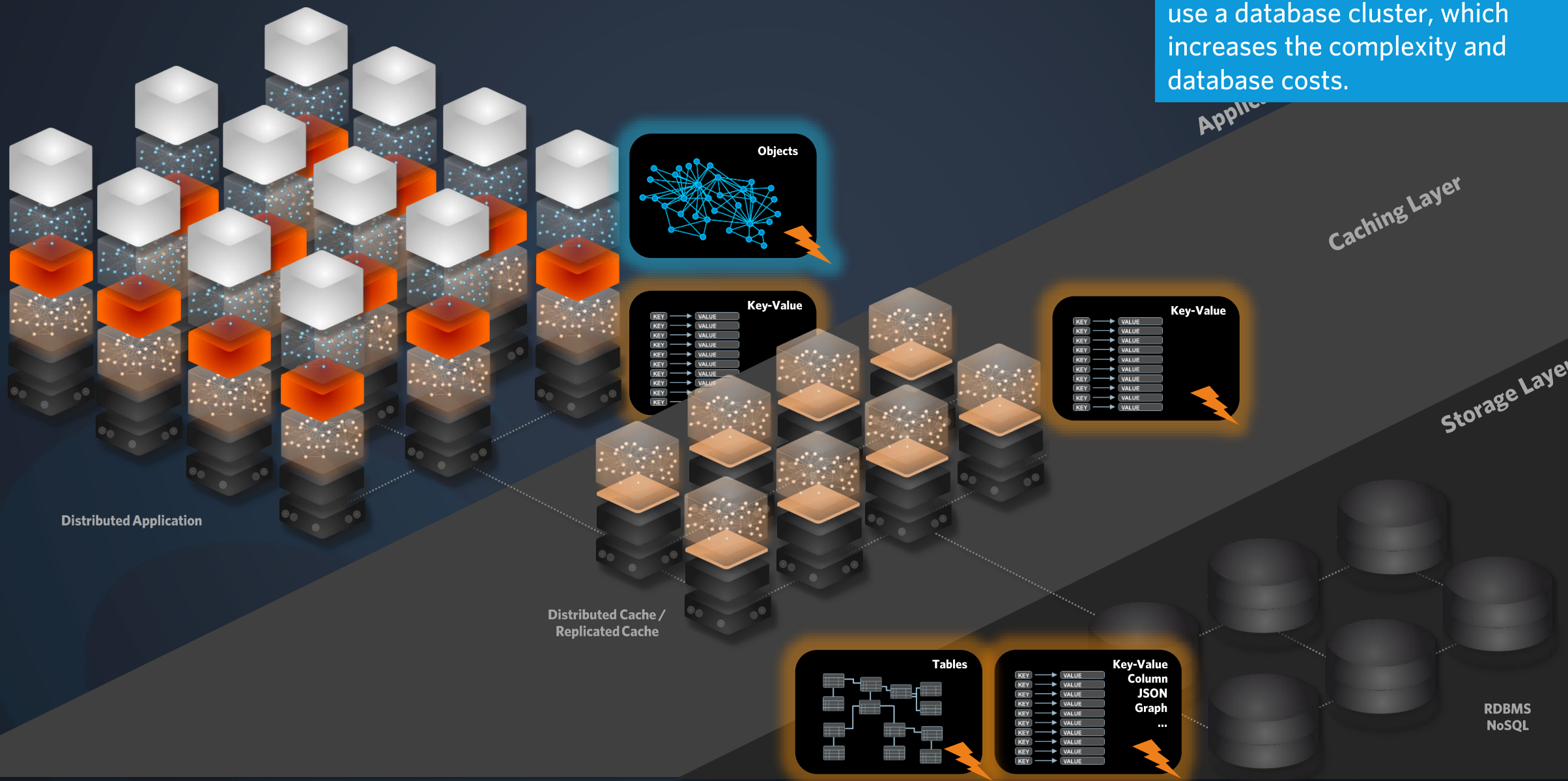


In distributed systems, using a distributed cache is a common caching strategy to avoid data inconsistency.

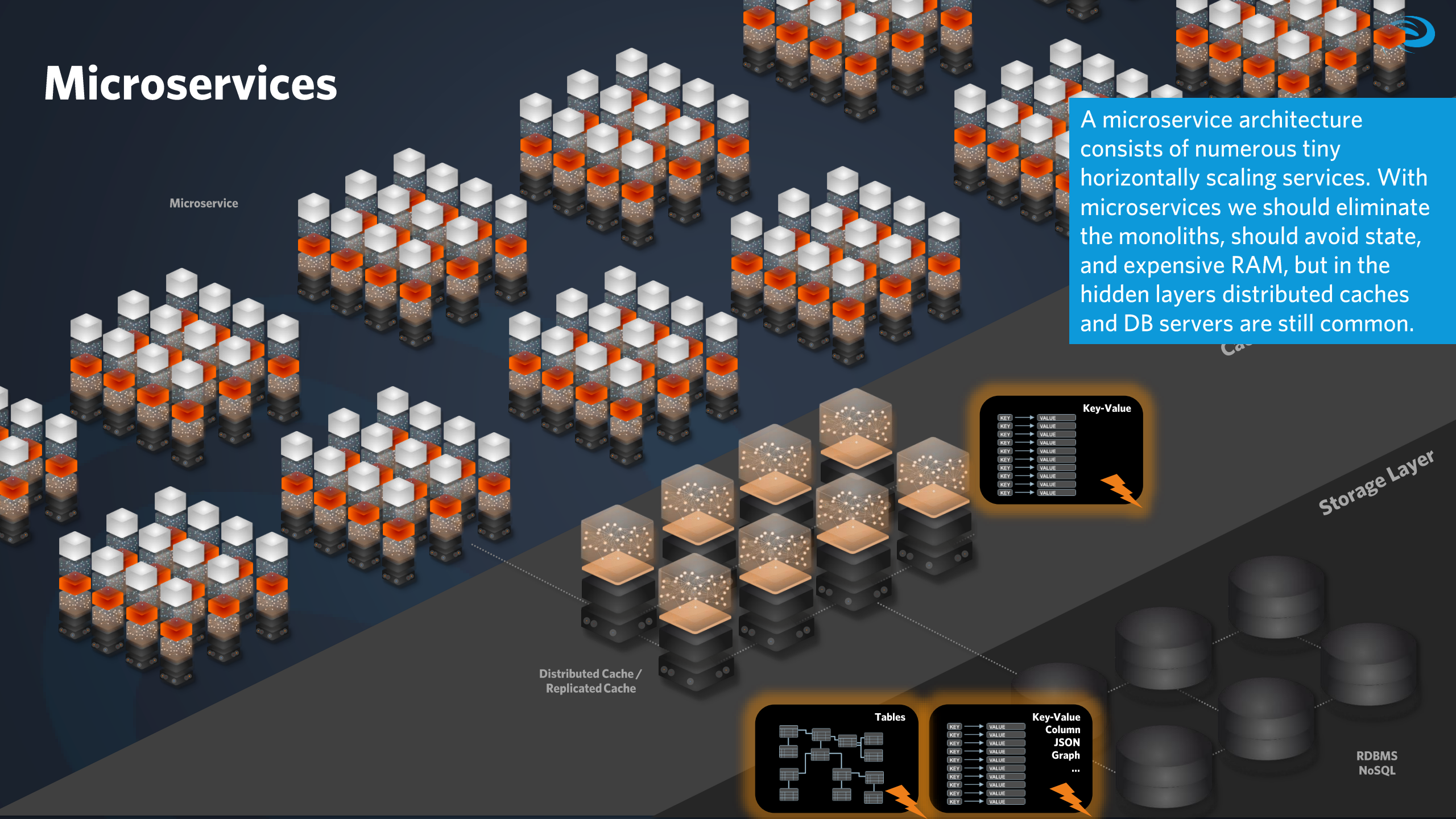
Note: Using a distributed cache requires additional nodes, CPUs, a a lot of expensive RAM, introduces state, and causes higher infrastructures and maybe subscription costs.

Database Cluster

Big database applications usually use a database cluster, which increases the complexity and database costs.



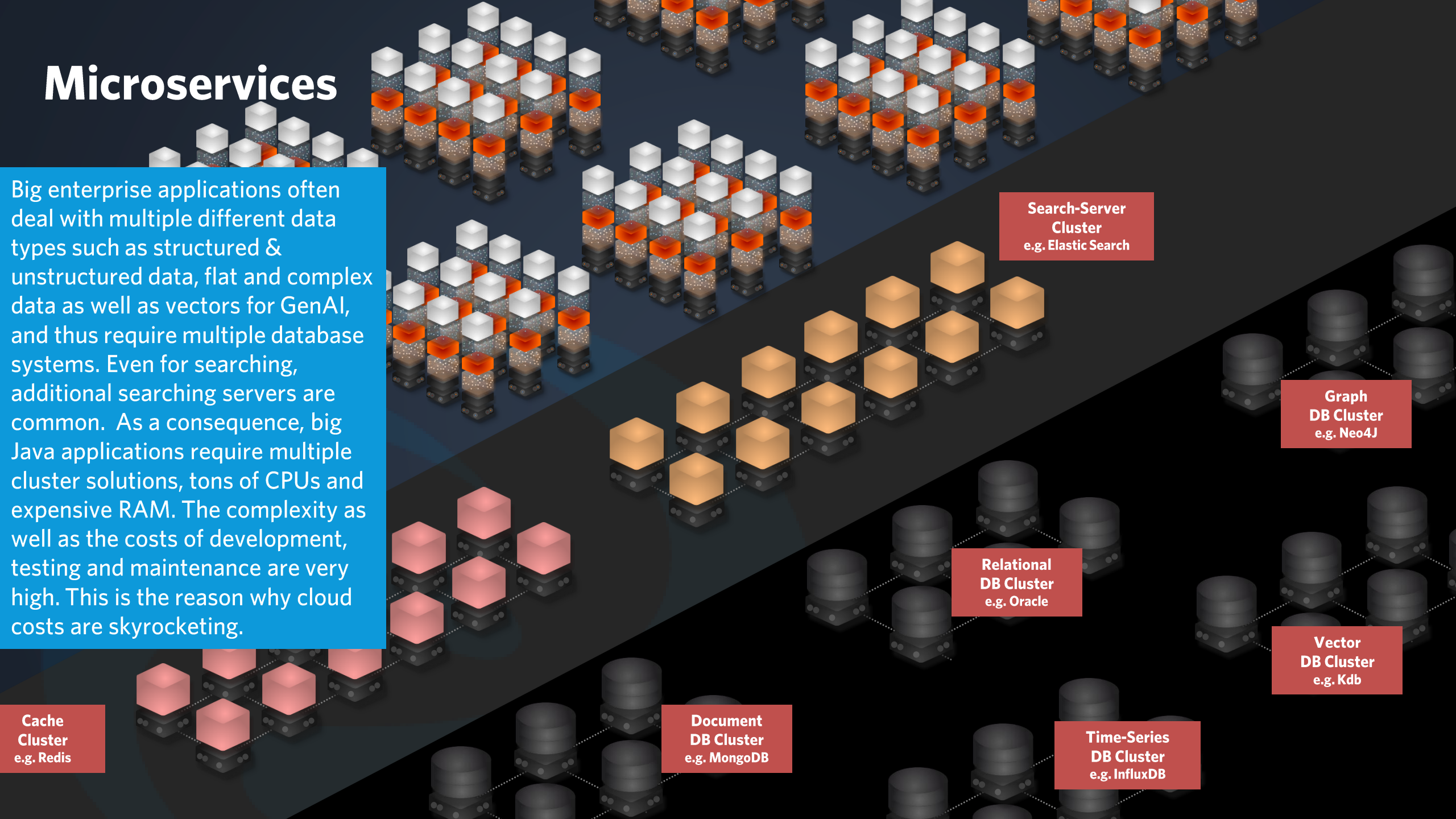
Microservices



A microservice architecture consists of numerous tiny horizontally scaling services. With microservices we should eliminate the monoliths, should avoid state, and expensive RAM, but in the hidden layers distributed caches and DB servers are still common.

Microservices

Big enterprise applications often deal with multiple different data types such as structured & unstructured data, flat and complex data as well as vectors for GenAI, and thus require multiple database systems. Even for searching, additional searching servers are common. As a consequence, big Java applications require multiple cluster solutions, tons of CPUs and expensive RAM. The complexity as well as the costs of development, testing and maintenance are very high. This is the reason why cloud costs are skyrocketing.



Traditional Databases are Expensive in the Cloud

AWS RDS – PostgreSQL

PostgreSQL, 2 CPU, 8 GB RAM
1 Terabyte

$4,508/year

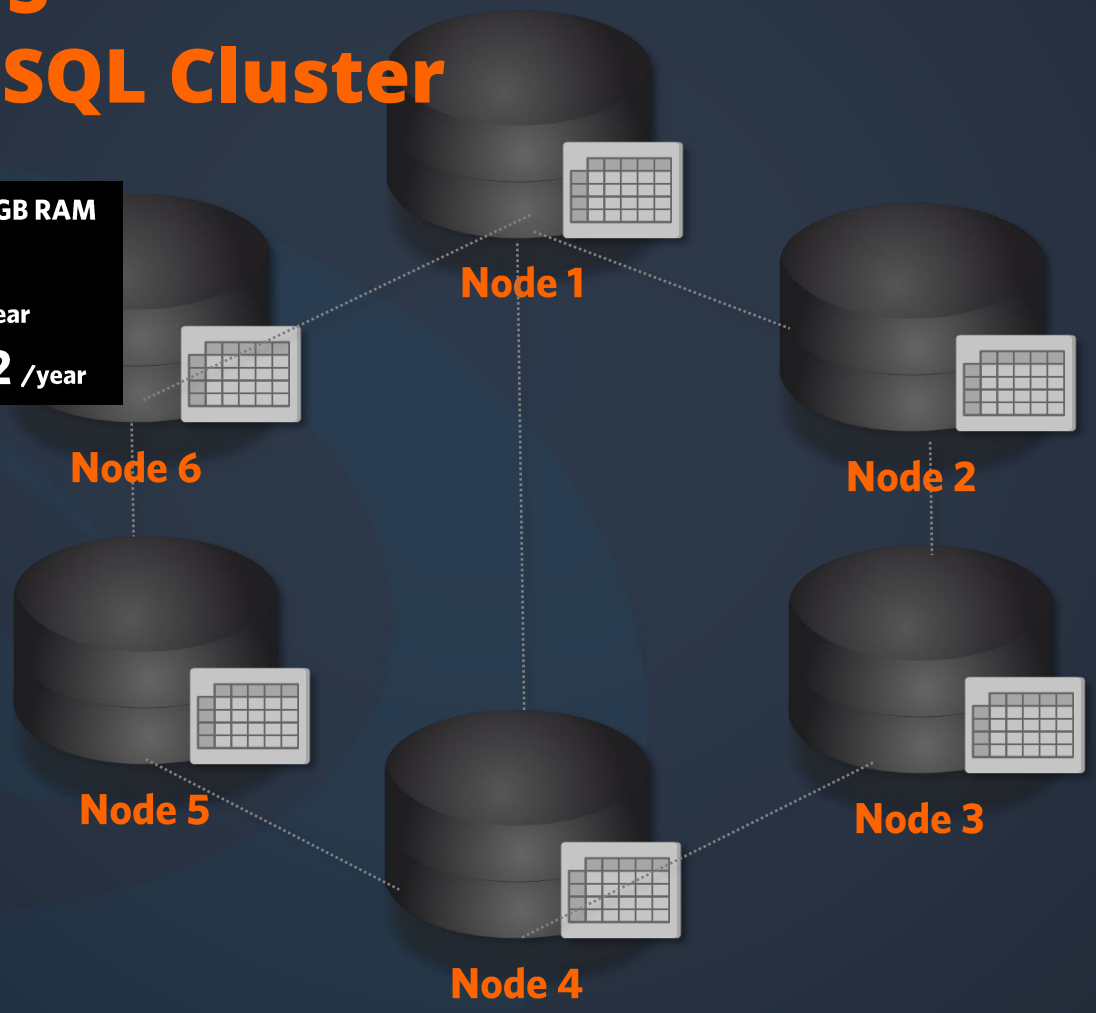
618 kg CO2 /year

1 Node

Traditional Databases are Expensive in the Cloud

AWS RDS – PostgreSQL Cluster

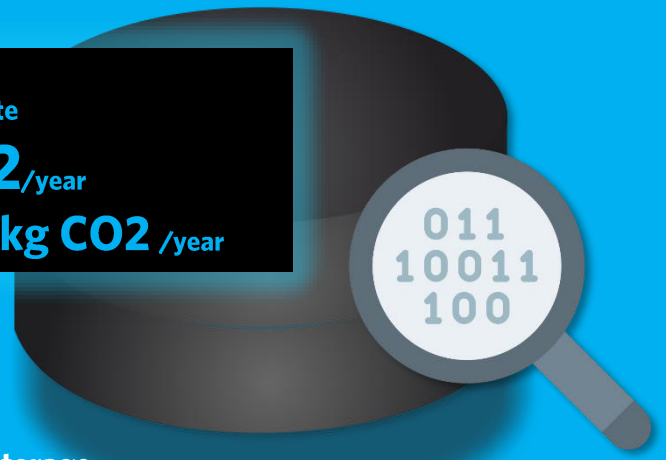
PostgreSQL, 2 CPU, 8 GB RAM
1 Terabyte
$27,048/year
3,608 kg CO2 /year



Plain Storage is 93% Cheper & Saves 99.84% CO2 Emission.

Cloud Binary Data Storage is up to 93% Cheaper than Any Cloud Database.

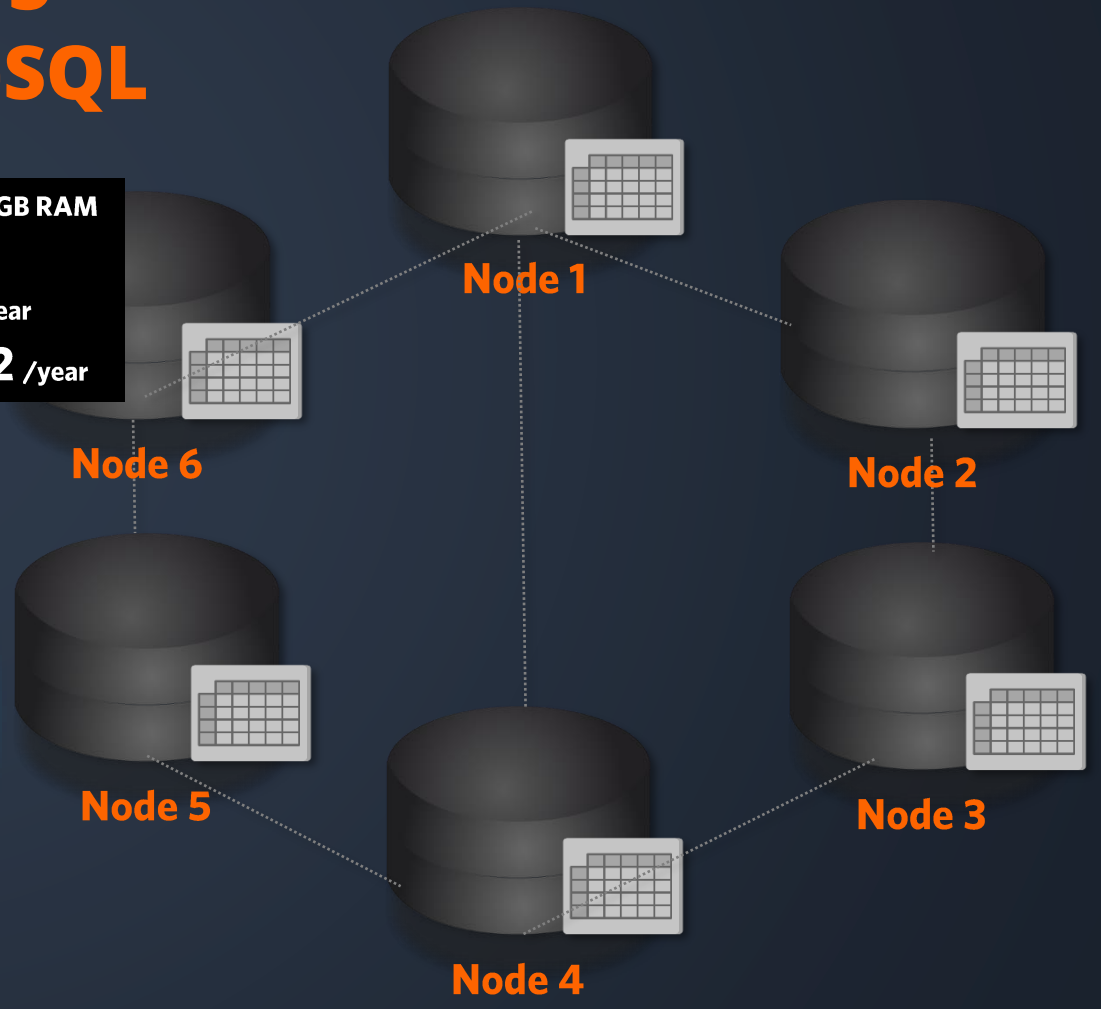
AWS S3
1 Terabyte
$312/year
0.98 kg CO2 /year



- AWS S3
- Azure Blob Storage
- Google Cloud Blob Storage
- Oracle Cloud Object Storage
- IBM Cloud Object Storage

AWS RDS – PostgreSQL

PostgreSQL, 2 CPU, 8 GB RAM
1 Terabyte
$27,048/year
3,608 kg CO2 /year



Use the Incredible JVM Performance for In-Memory Data Processing With with Java !

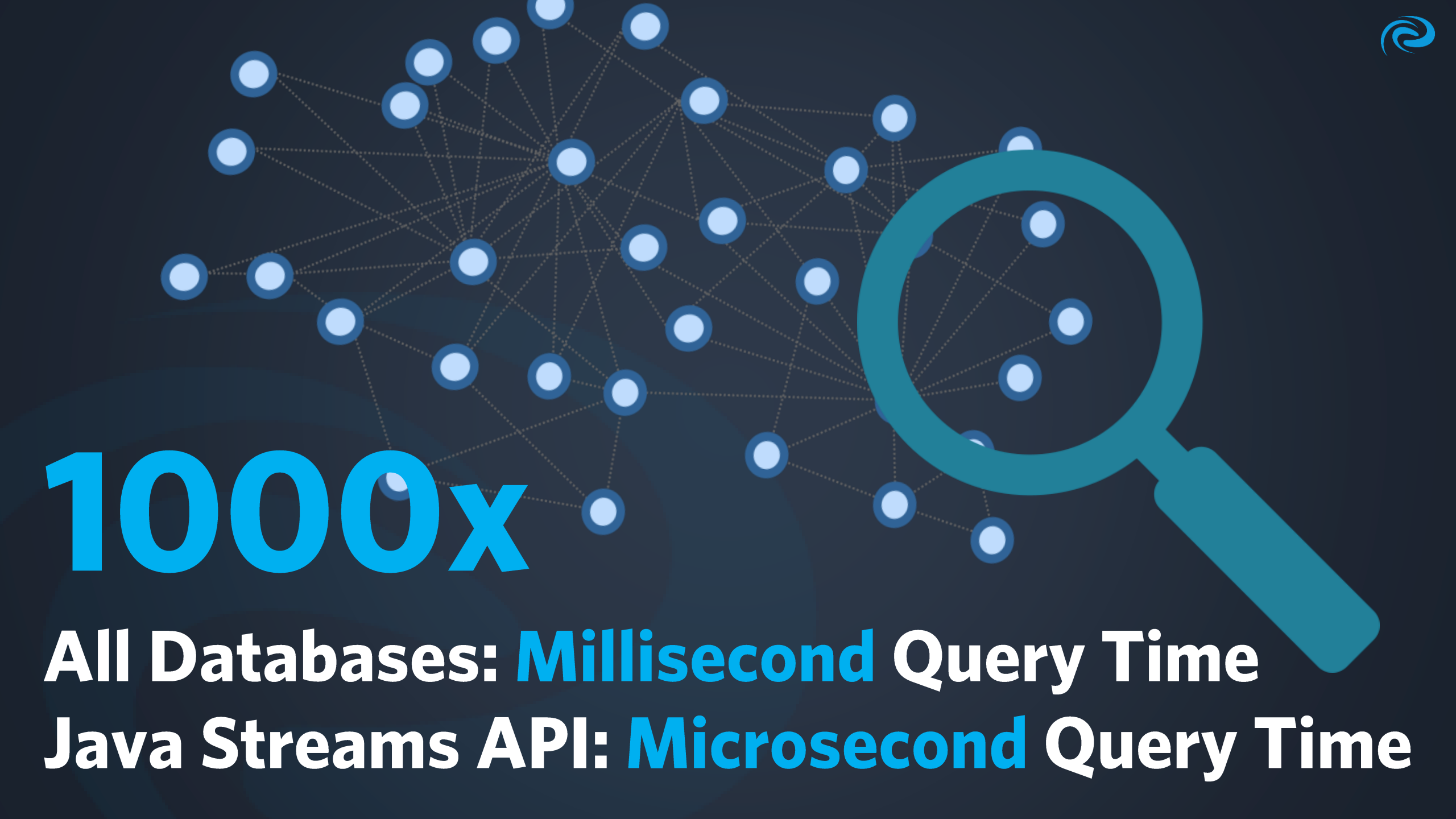
Any Data Model

- Objects
- Collections
- Graphs
- Documents (JSON, XML)
- Vectors

Java's Object Graph Model is a Multi-Model Data Strucutre

```java
public static void booksByAuthor()
{
      final Map<Author, List<Book>> booksByAuthor =
            ReadMeCorp.data().books().stream()
            .collect(groupingBy(book -> book.author()));

       booksByAuthor.entrySet().forEach(e -> {
           System.out.println(e.getKey().name());
           e.getValue().forEach(book -> {
               System.out.print('\t');
               System.out.println(book.title());
           });
       });
}
```

Searching & Filtering with Java Streams API


1000x
All Databases: Millisecond Query Time
Java Streams API: Microsecond Query Time

The Only Feature that is Missing in Java:

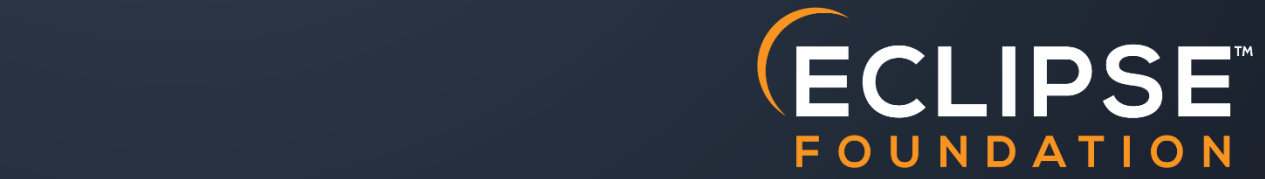
A Java-Native

Persistence

EclipseStore

1000x Faster Data Processing.
93% Cloud Database Cost Savings.
Java-Native & Simplicity.
Save 99.84% Energy & CO2 Emission

www.eclipsestore.io



EclipseStore

MicroStream is a micro persistence engine to store any Java object graph of any size and complexity into any binary data storage, and to restore it completely or partially in RAM on demand.

Get Started

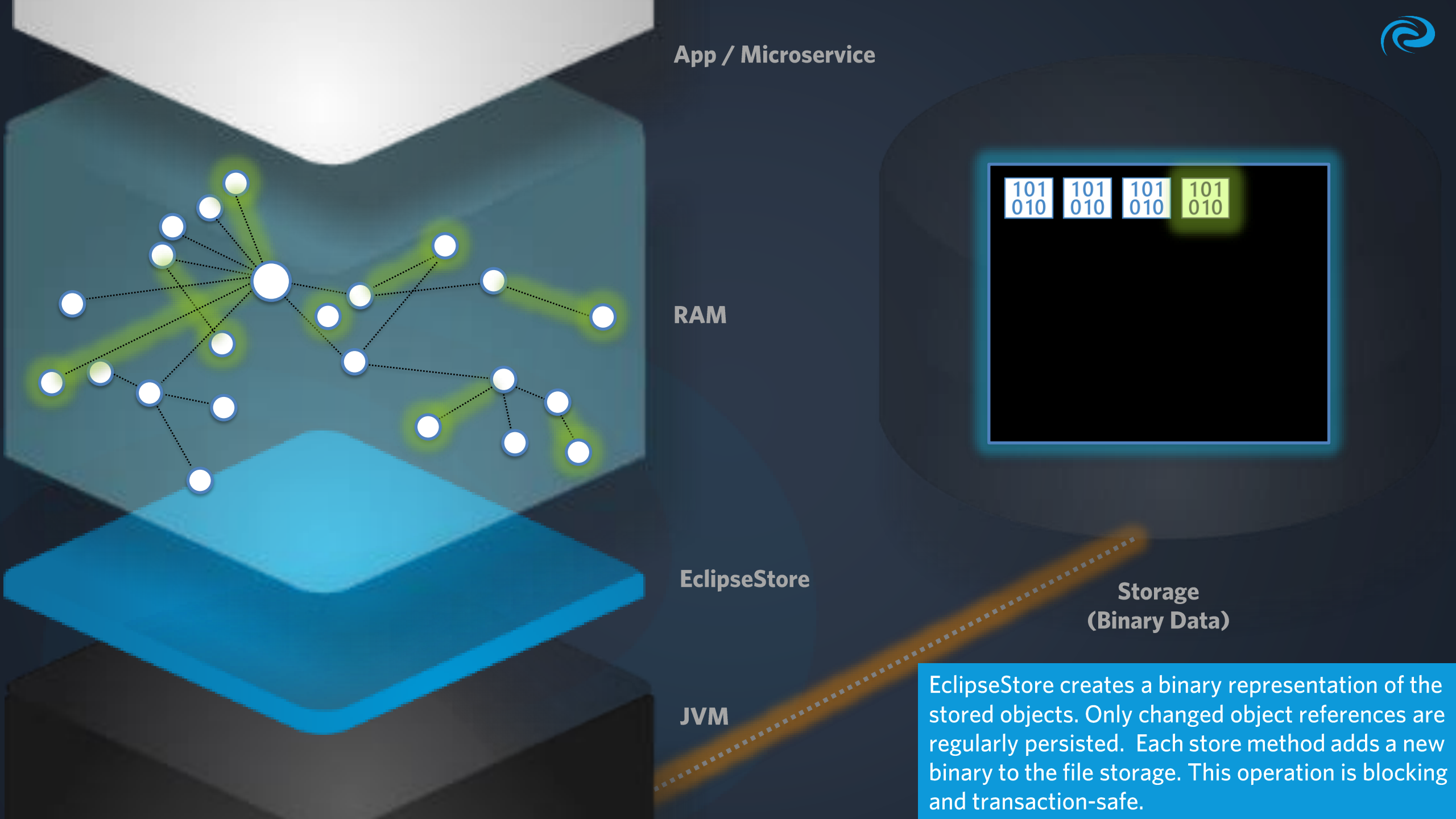
pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.eclipse.store</groupId>
        <artifactId>storage-embedded</artifactId>
        <version>1.3.2</version>
    </dependency>
</dependencies>
```

Persisting Objects

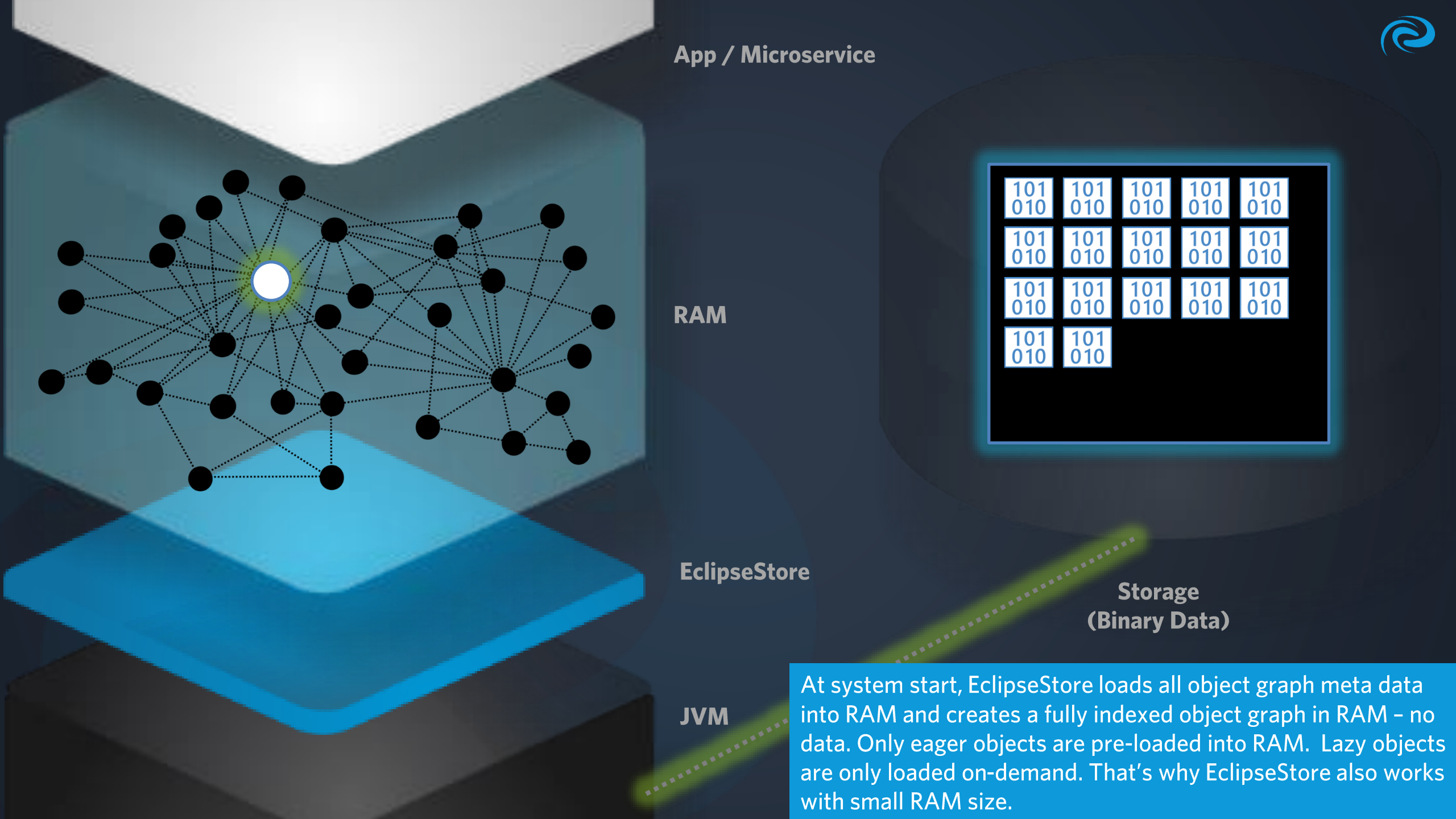
```
DataRoot root = microstreamDemo.root();
   root.getCustomers().add(customer);

   microstreamDemo.store(root.getCustomers());
```

- Changes Only (Subgraphs)
- Blocking
- Synchronous
- ACID Transaction-Safe
- Full Consistent
- Append-Log Strategy
- Multithreaded Writes



EclipseStore creates a binary representation of the stored objects. Only changed object references are regularly persisted. Each store method adds a new binary to the file storage. This operation is blocking and transaction-safe.

Initial System Start

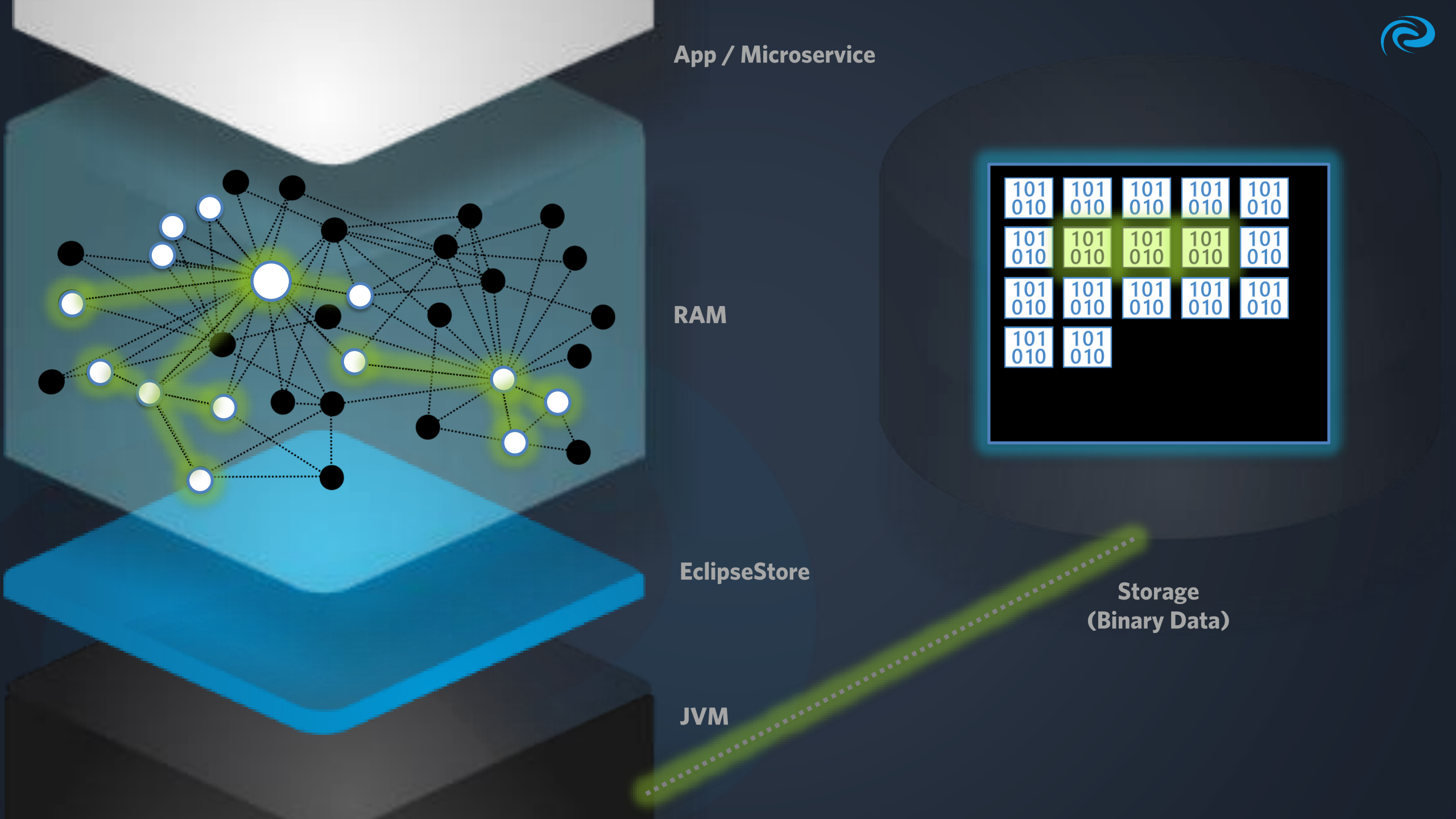

App / Microservice
RAM
EclipseStore
JVM
Storage
(Binary Data)
101
010
At system start, EclipseStore loads all object graph meta data into RAM and creates a fully indexed object graph in RAM – no data. Only eager objects are pre-loaded into RAM. Lazy objects are only loaded on-demand. That's why EclipseStore also works with small RAM size.

Lazy-Loading

Load any Object-References Dynamically Into RAM

```
public class Customer {

  ...
  private Lazy<Set<Order>> orders;
  ...

  public Set<Order> getOrders() {
     return Lazy.get(this.orders);
  }

  public void getOrders(final Set<Order> orders) {
     this.orders = Lazy.Reference(orders);
  }

  ...
}
```

- Loading Subgraphs
- Object Graph is Updated
- No Object Copies
- Multithreaded IO Op

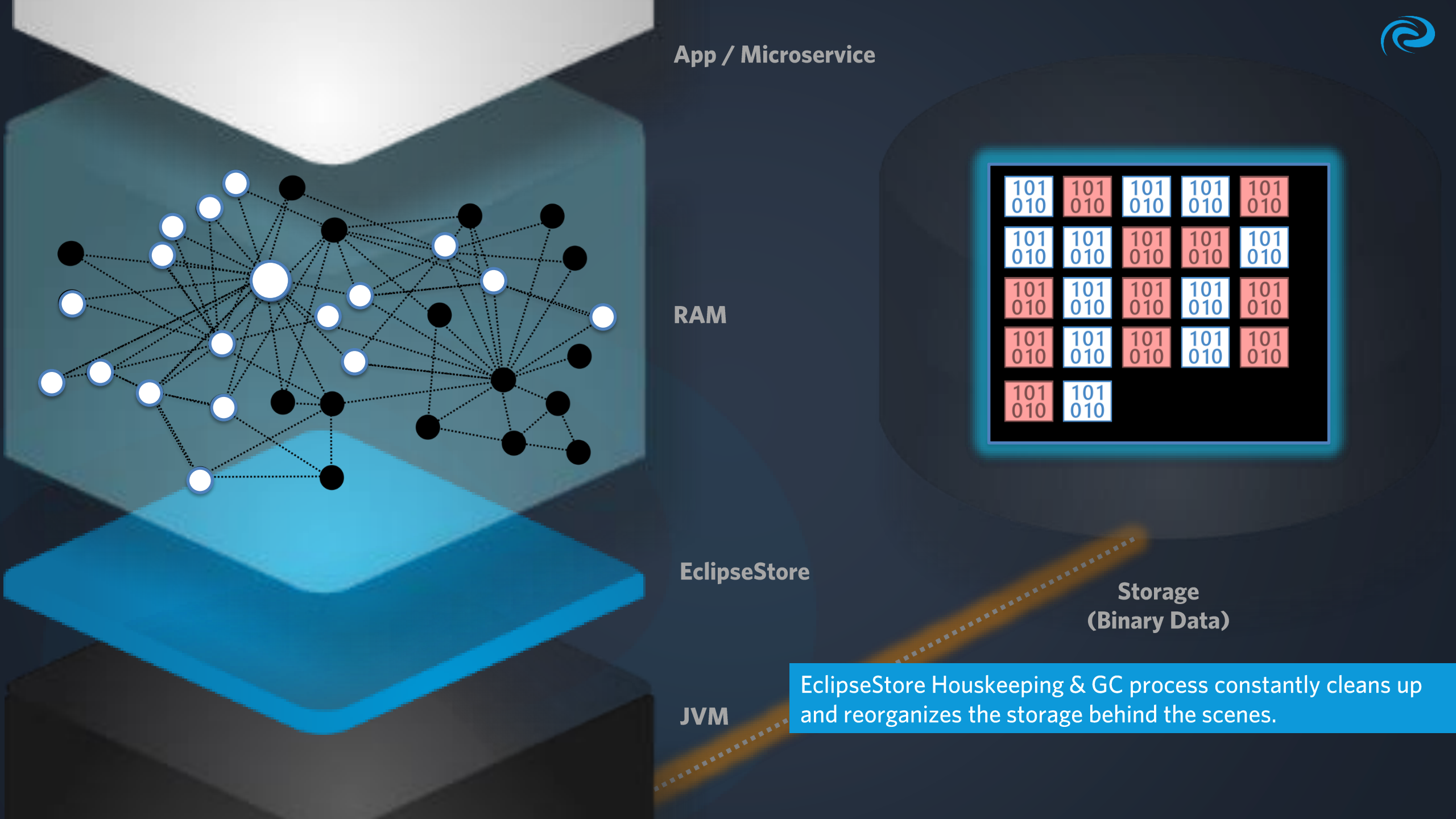

App / Microservice
RAM
EclipseStore
JVM
Storage
(Binary Data)
101
010

Queries

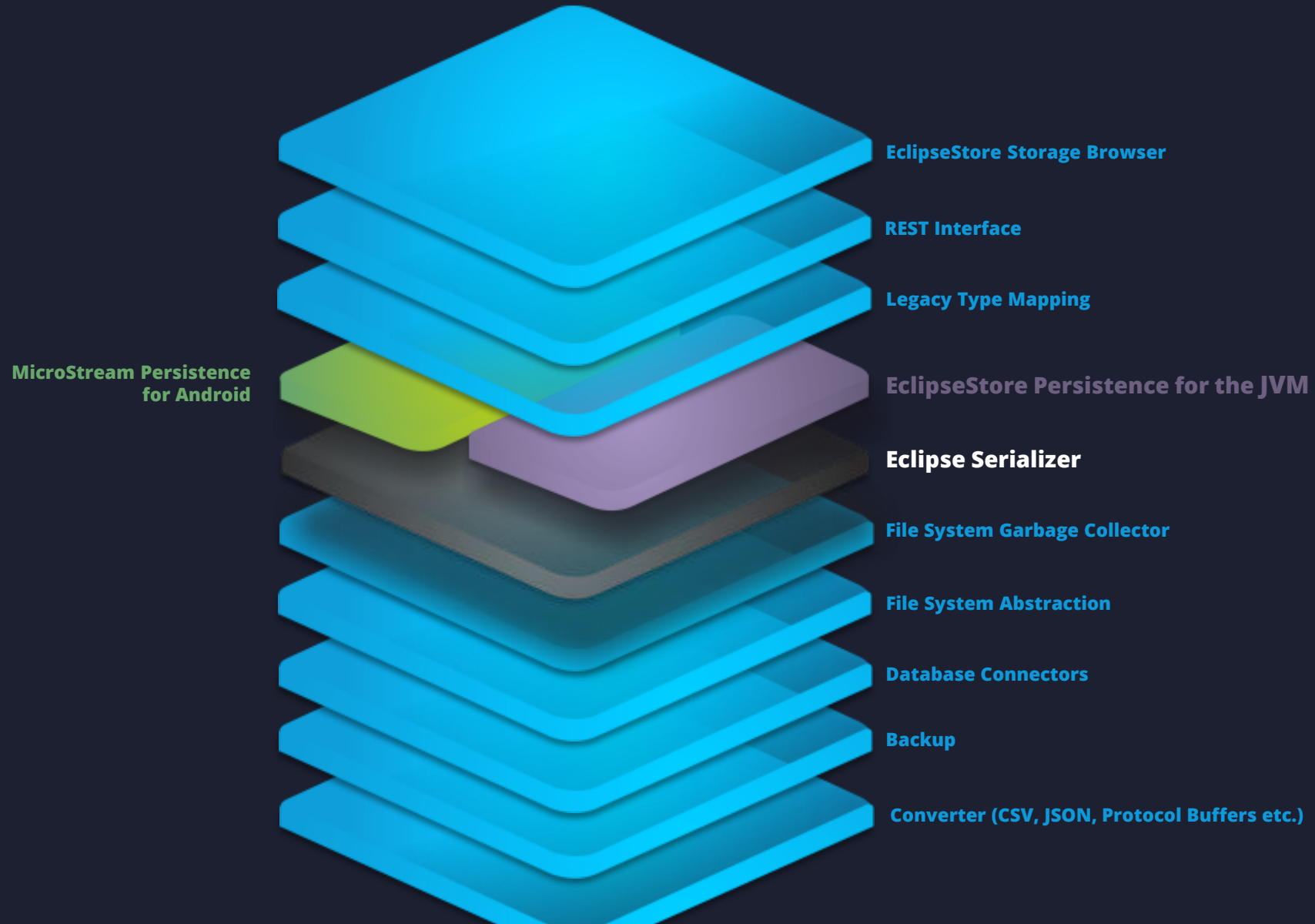
```
public static void booksByAuthor()
{
      final Map<Author, List<Book>> booksByAuthor =
            ReadMeCorp.data().books().stream()
            .collect(groupingBy(book -> book.author()));

       booksByAuthor.entrySet().forEach(e -> {
           System.out.println(e.getKey().name());
           e.getValue().forEach(book -> {
               System.out.print('\t');
               System.out.println(book.title());
           });
       });
}
```

- Java Streams API
- Core Java
- Type-Safe
- Microsecond Query Time

Housekeeping & Garbage Collection


App / Microservice
RAM
EclipseStore
JVM
Storage
(Binary Data)
101
010
EclipseStore Houskeeping & GC process constantly cleans up and reorganizes the storage behind the scenes.

EclipseStore Components



Use any JVM Technology

Runs Wherever Java Runs

Desktops

On-Premise

Cloud

Container

Native Image

Microservices

Android

JDK 8+

Supported Storages

Further Connectors are in Progress

Cloud Object Store

RDBMS

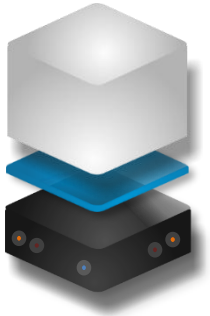
NoSQL

Traditional databases can be used as a storage target, because they can store EclipseStore's binary files. But, keep in mind, in any case, EclipseStore stores binaries. There is no database data model and no mappings or data conversion!

Enterprise-Grade Open-Source

EclipseStore Open Source

- Embedded
- 1 Server/Node Only
- Java & Android Only
- Not Supported

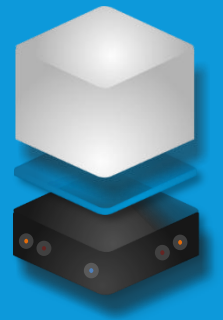


Java android

MicroStream Enterprise

Enterprise Edition

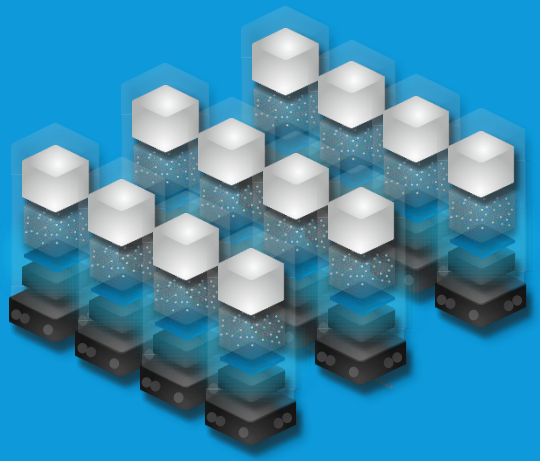
- Indexing
- Automated Lazy-Loading
- Asynchronous Write Ops
- Enterprise Support
- Embedded
- 1 Server/Node Only
- Java & Android Only



Java android

Cluster

- Distributed
- Unlimited Nodes
- Rolling Updates
- Enterprise Support



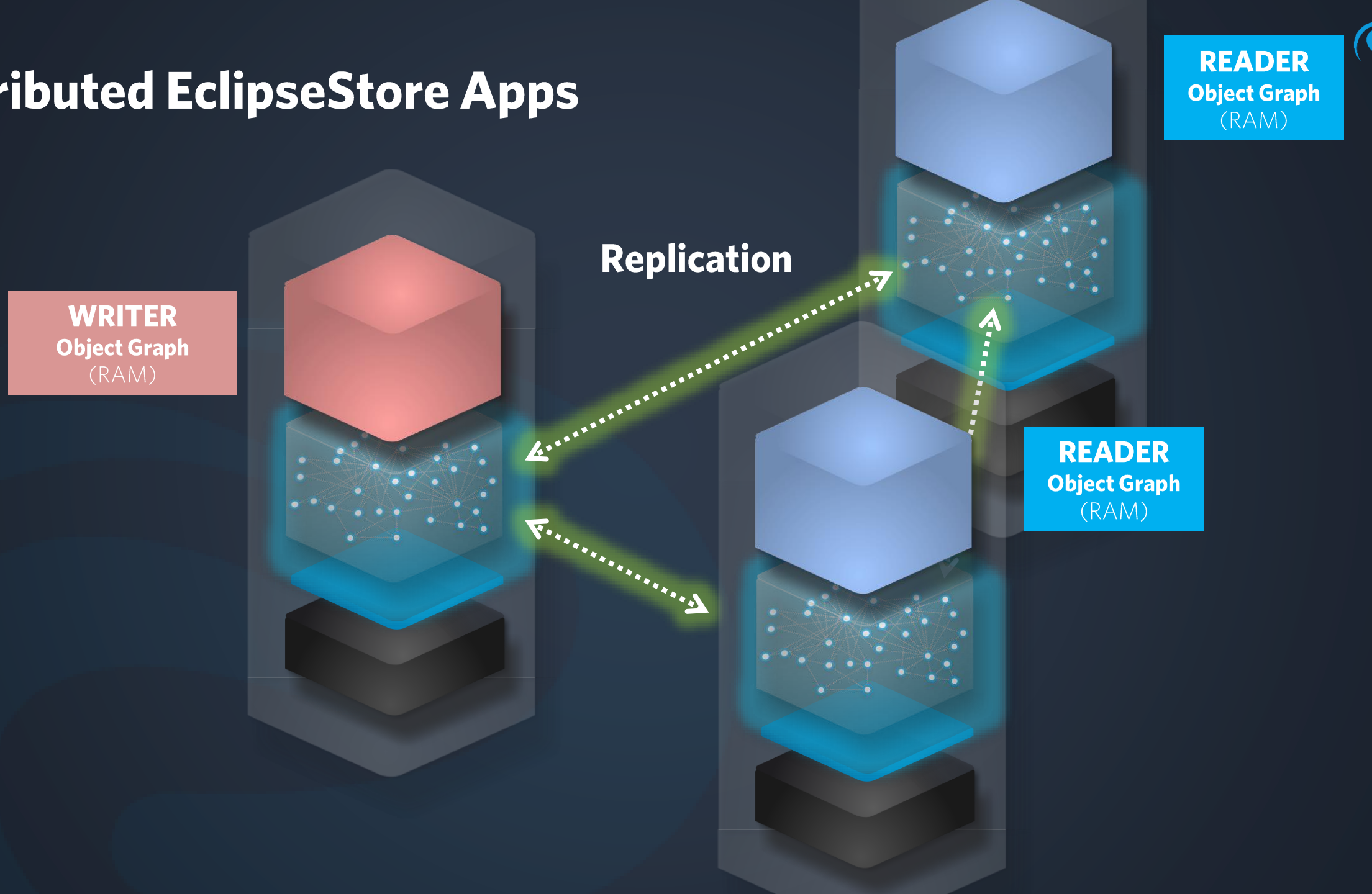
SaaS

- Fully Managed
- Elastic Scaling

On-Prem

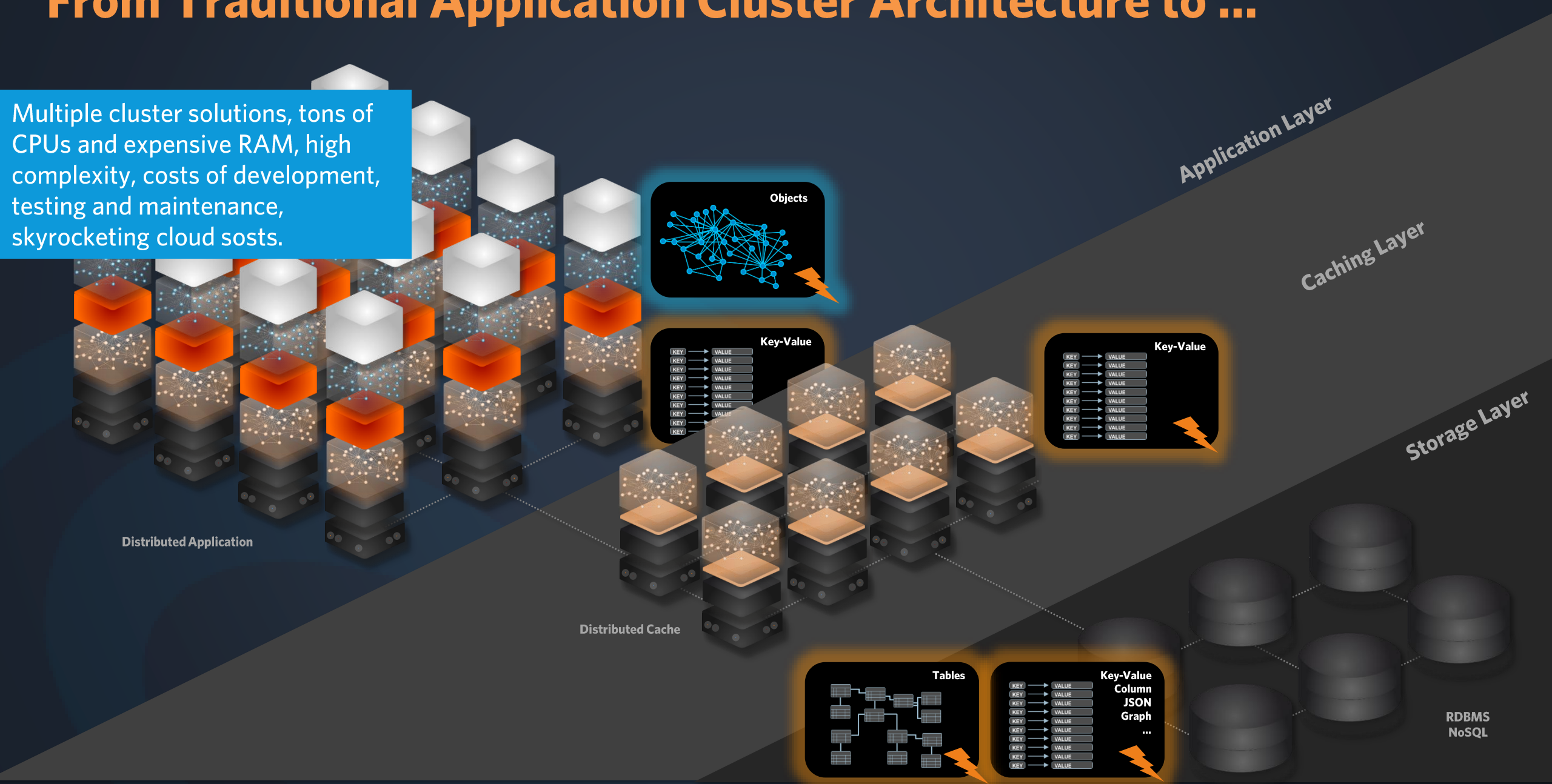
- Based on Kubernetes
- Elastic Scaling

Distributed EclipseStore Apps



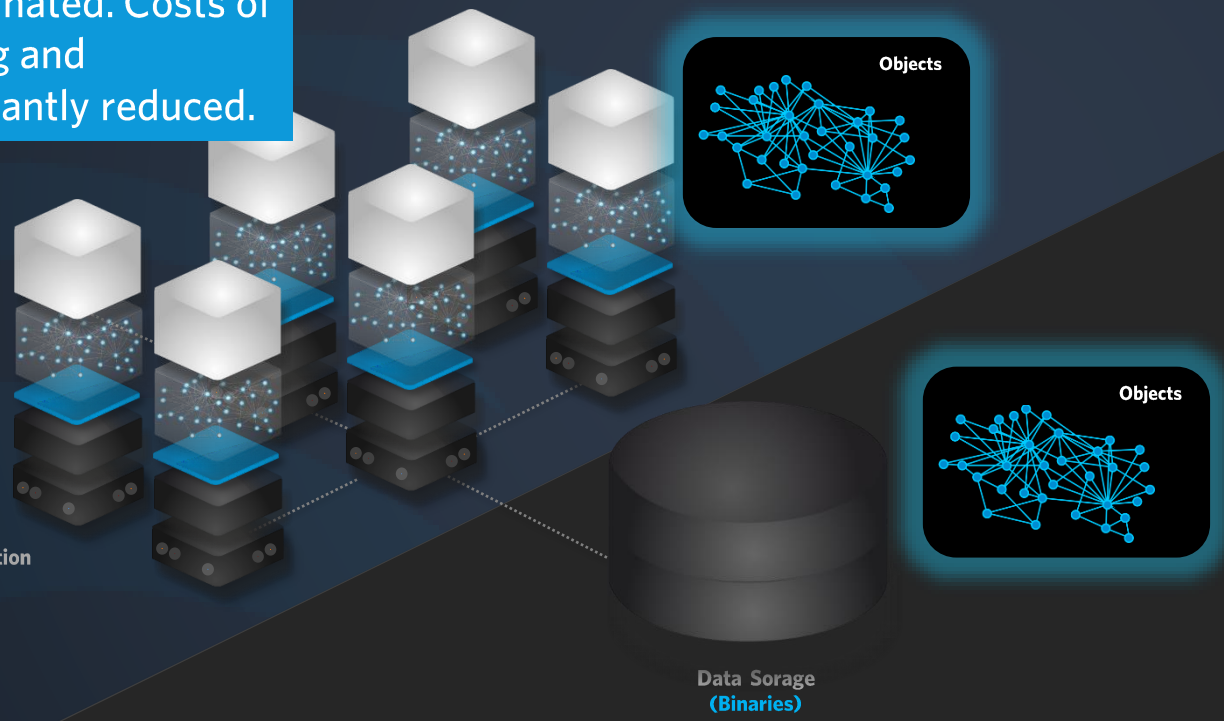
From Traditional Application Cluster Architecture to ...

Multiple cluster solutions, tons of CPUs and expensive RAM, high complexity, costs of development, testing and maintenance, skyrocketing cloud sosts.



EclipseStore Cluster

Ultra-fast and highly efficient Java in-memory data processing. Only one cluster solution, tons of CPUs and complexity eliminated. Costs of development, testing and maintenance significantly reduced.



Streaming Objects Directly Into any Database

Conversion Eliminated !

- Simple architecture
- Faster time to market
- Saves lots of vCPU power
- Minimizes latencies
- In-memory queries executed in microseconds
- Saves 93% cloud database costs

1000x faster Queries

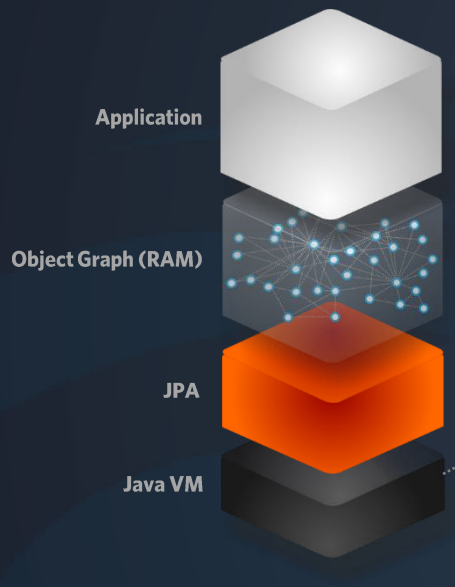
Microsecond

Query Time

Rules & Challenges

Your Object Graph in RAM is Your Database

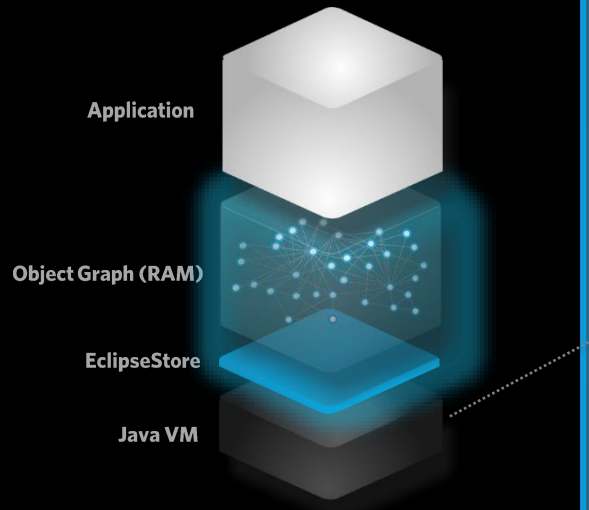
Traditional Database Server Paradigm



Database Server
- Database
- DBMS
- Queries
- Persistent Data

Your Database is here,
Queries are executed here

Java In-Memory Data Processing Paradigm



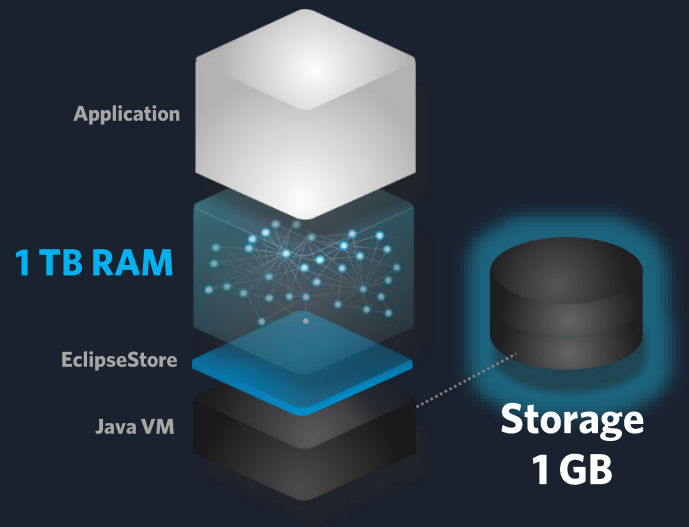
- Database
- Queries

Your Database is here,
Queries are executed here

Any Storage
- Persistent Data

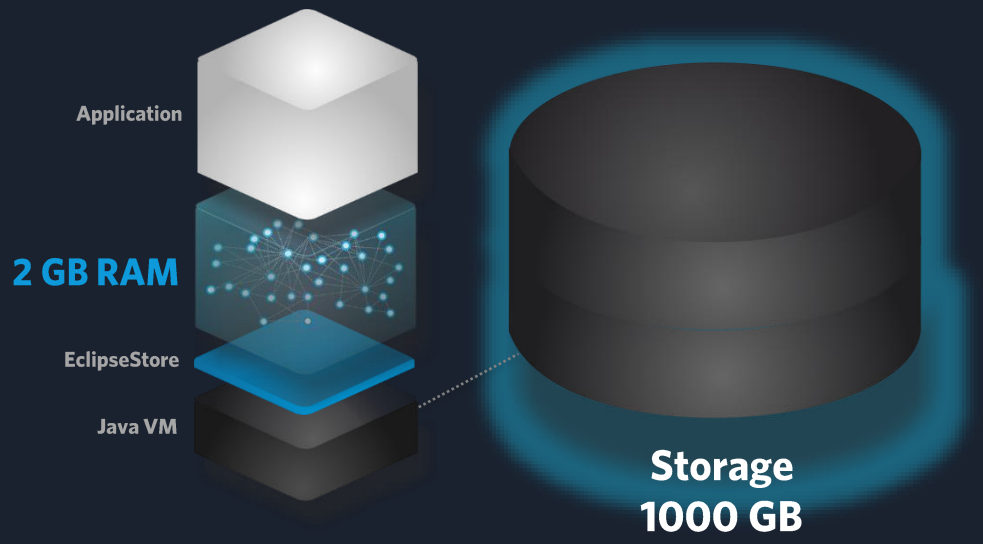
Fully In-Memory is Not Mandatory

Enough RAM available:



- Load your entire DB into RAM
- Pure in-memory computing
- No latencies
- Super fast
- Lower startup time

Data Storage is bigger than RAM:



- Preload most important data only (eager loading)
- Use lazy-loading to load data on demand only
- Clear lazy references which are not used anymore
- Faster startup time

Challenges with EclipseStore

- Java / JVM languages only
- Built for Java developers
- Paradigm shift in database programming
- No more relational model – Build any Java object model
- RDBMS features are replaced by Core Java Features or Business Logic (user management, validation, concurrency, logging, locking etc.)
- No drop-in replacement
- No SQL support
- DB tools are not suited
- Not suited for DBAs

Conclusion

Simplicity

- Simplest possible architecture & implementation
- No mappings or data conversion
- No specific requirements for your classes, just use POJOs
- No specific client API required
- No specific query language
- Core Java concepts only
- Get a clustered app / microservice out-of-the-box

High Performance

- Ultra-fast in-memory data processing
- Microsecond query time
- Low-latency realtime responsiveness
- Gigantic workloads & throughput

Cost Savings

- 93% cloud database costs
- Saving costs of development, testing and maintainence

Sustainability

- **99.84% savings of CPU power, energy & CO2 emission**

JAVAPRO

Free Tickets: EclipseStore Online Training

May 23, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Fundamentals Training: Free
June 11, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 1 Training
July 3, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 2 Training
September 17, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Fundamentals Training: Free
October 2, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 1 Training
October 17, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 2 Training
November 6, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Fundamentals Training: Free
November 26, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 1 Training
December 12, 2024	1 Day	17:00 - 21:00 CET	Europe, Middle East, Africa	EclipseStore Advanced 2 Training

www.javapro.io/training

Open Source (EPL 2.0)

www.eclipsestore.io

Download & Get Started Coding:

https://docs.eclipsestore.io/manual/storage/getting-started.html

Technical Questions:

https://github.com/eclipse-store/store/discussions

Enterprise Edition, Cluster, Consulting, Book a Team Intro Call:

www.microstream.one

EclipseStore

1000x Faster Data Processing.
93% Cloud Database Cost Savings.
Java-Native & Simplicity.
Save 99.84% Energy & CO2 Emission

www.eclipsestore.io

Contact:
Markus Kett, CEO

m.kett@microstream.one
Cell: +49-160-90766677
Linked-In: MarkusKett

MicroStream

ECLIPSE™ FOUNDATION