Java's Untapped Potential: High-Performance In-Memory Data Processing with Java

Markus Kett
CEO at Cyrock.AI, Inc.

m.kett@cyrock.ai
LinkedIn: /MarkusKett
X:@MarkusKett

Markus Kett

CEO at Cyrock.AI

X: @MarkusKett
LinkedIn: markuskett

Email:
m.kett@microstream.one
m.kett@cyrock.ai

- XDEV Java IDE
- RapidClipse – Visual Java IDE
- JBoss Hibernte Tools Improvements
- JPA-SQL
- Eclipse Serializer
- EclipseStore
- Eclipse Data Grid
- Project Helidon
- Micronaut

Founder of JAVAPRO Magazine
Editor in Chief

Founder of JCON Conference Series
Speaker, Author

MicroStream

EclipseStore

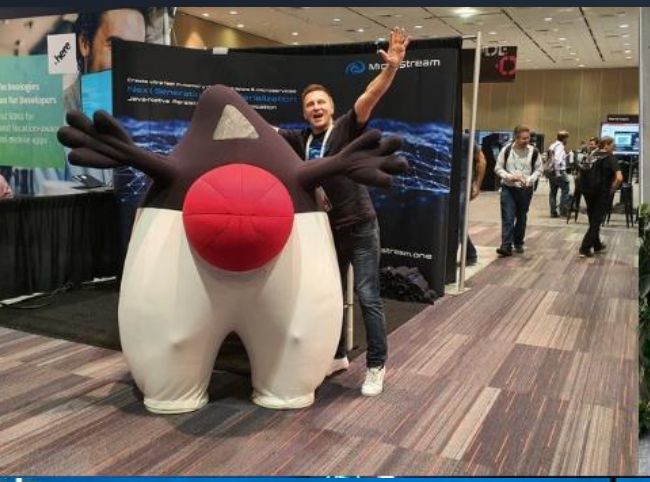
RAPID clipse™

JPA-SQL

JAVAPRO

JCON 2021

The Team Behind Eclipse Data Grid & Cyrock.AI

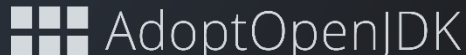


Eclipse Foundation: More Than 100 Popular Open-Source & Standardization Projects are Developed by Numerous Companies.

ECLIPSE FOUNDATION

projects.eclipse.org

We are Managing 3 Eclipse Projects:
Eclipse Data Grid, EclipseStore, and Eclipse Serializer.



In-Memory Computing is Everywhere

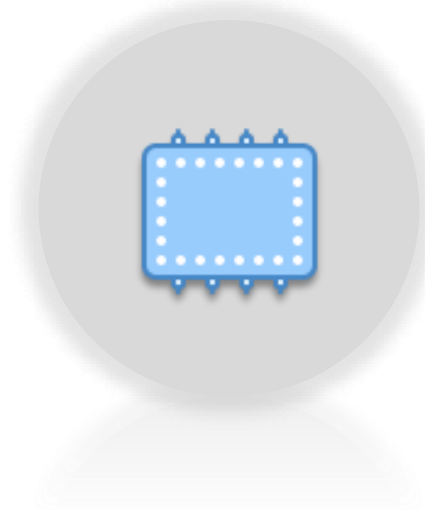
Latency Numbers Every Programmer Should Know

HDD
~ 9 ms
~ 9,000,000 ns

SSD HDD
~ 250 µs
~ 250,000 ns

DRAM
~ 65 ns

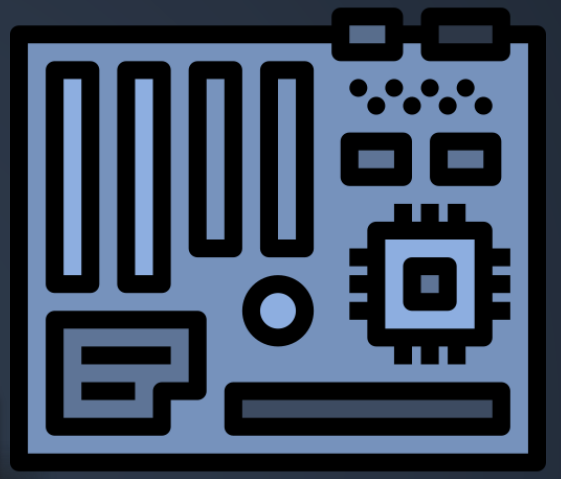
~ **100,000x faster than HDD**
~ **4,500x faster than SSD**



CPU Cache
< 1 ns

In-Memory Computing: Code Executed Inside a System.

Serverside In-Memory Computing:
Code Executed On the Mainboard.



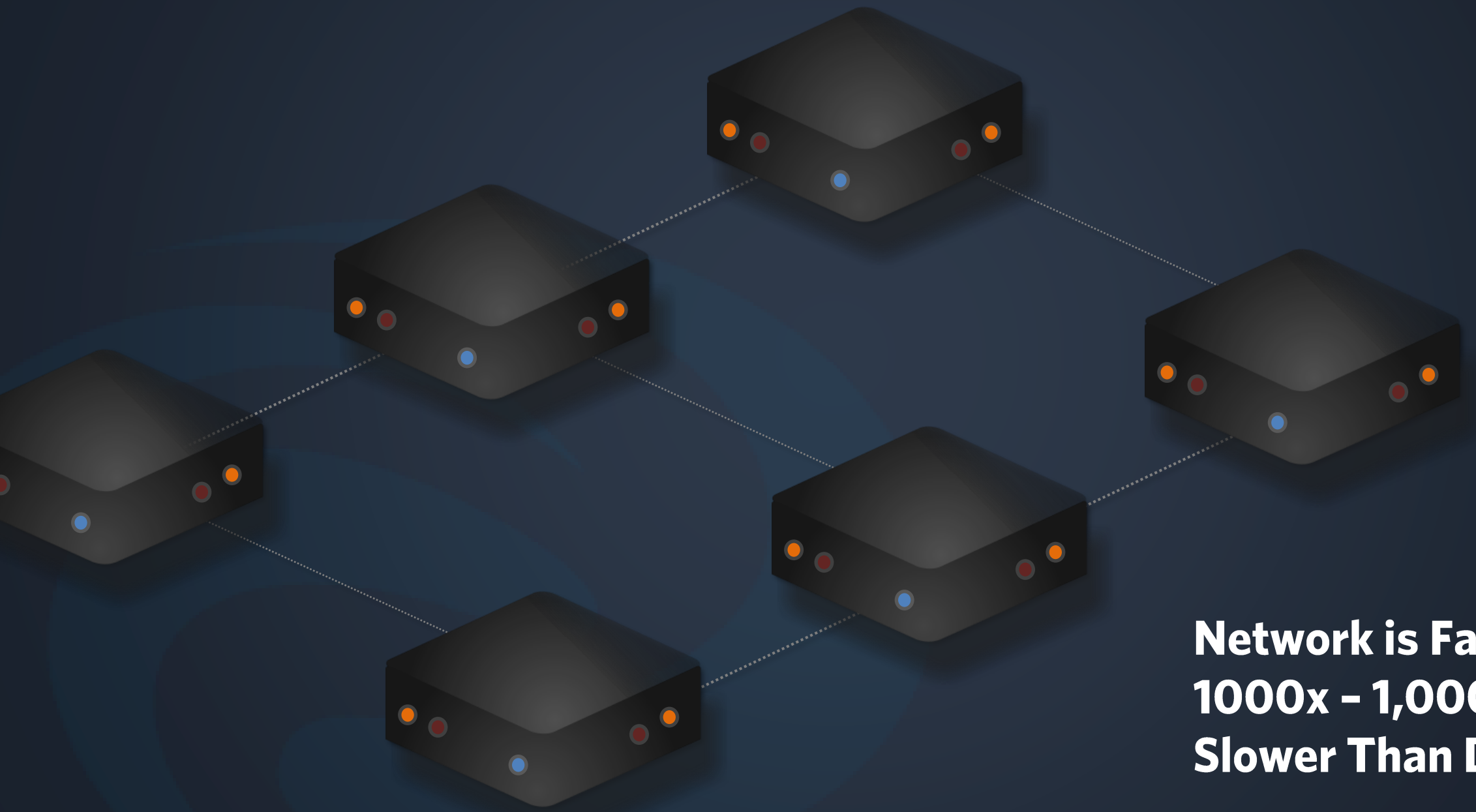
In-Memory

DRAM
~ 65 ns

~ 100,000x faster than HDD
~ 4,500x faster than SSD

HDD
~ 9 ms
~ 9,000,000 ns

Distributed In-Memory Systems are 1000x Slower

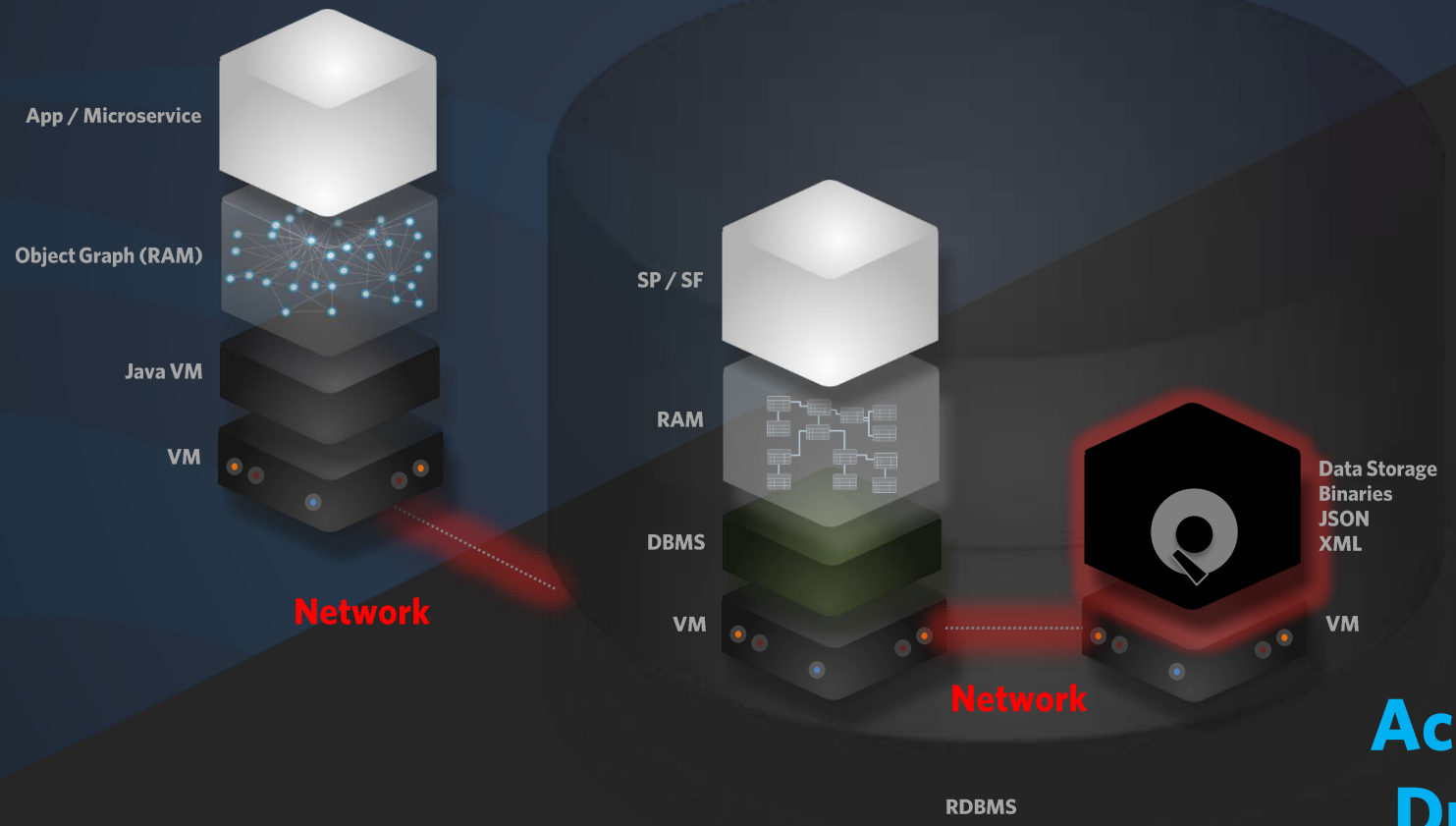


Network is Factor 1000x – 1,000,000x Slower Than DRAM.

In-Memory Computing with Java is Extremely Fast:
Microsecond Response-Time!

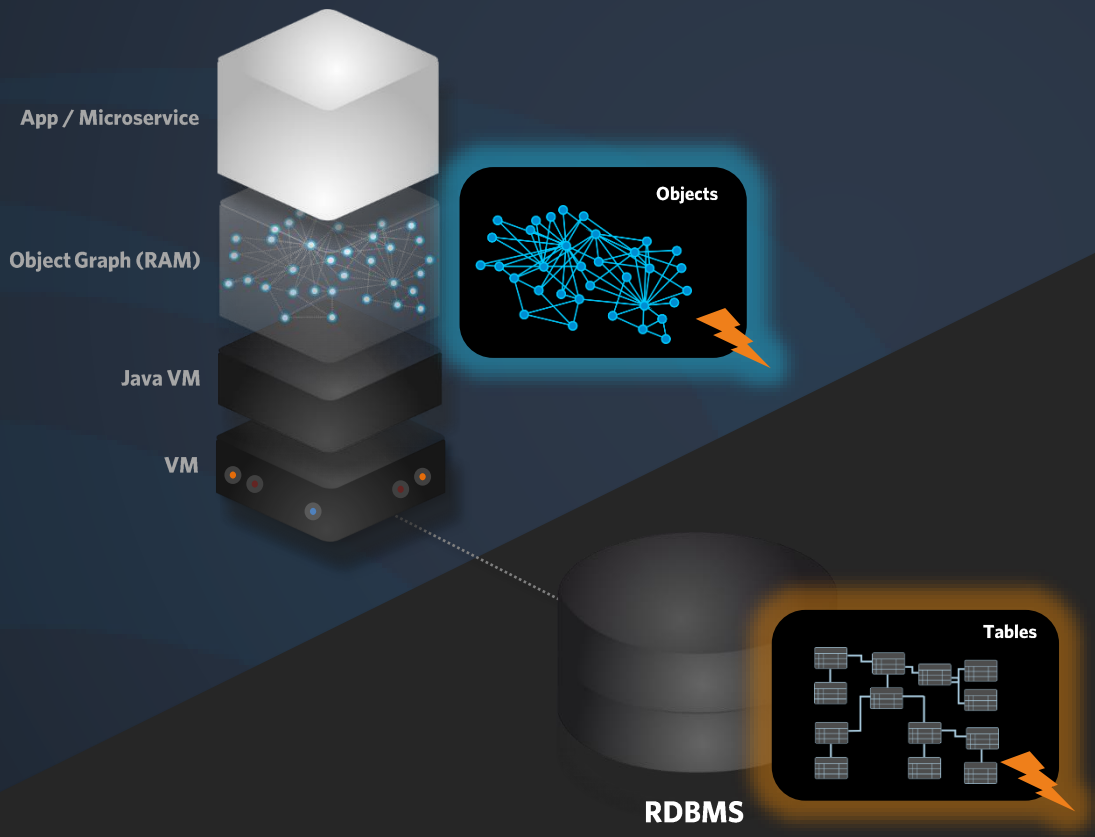
Where are the Hidden Latencies?

Traditional Database Server in the Cloud



Accessing Hard Disk Drives & Network is Most Expensive.

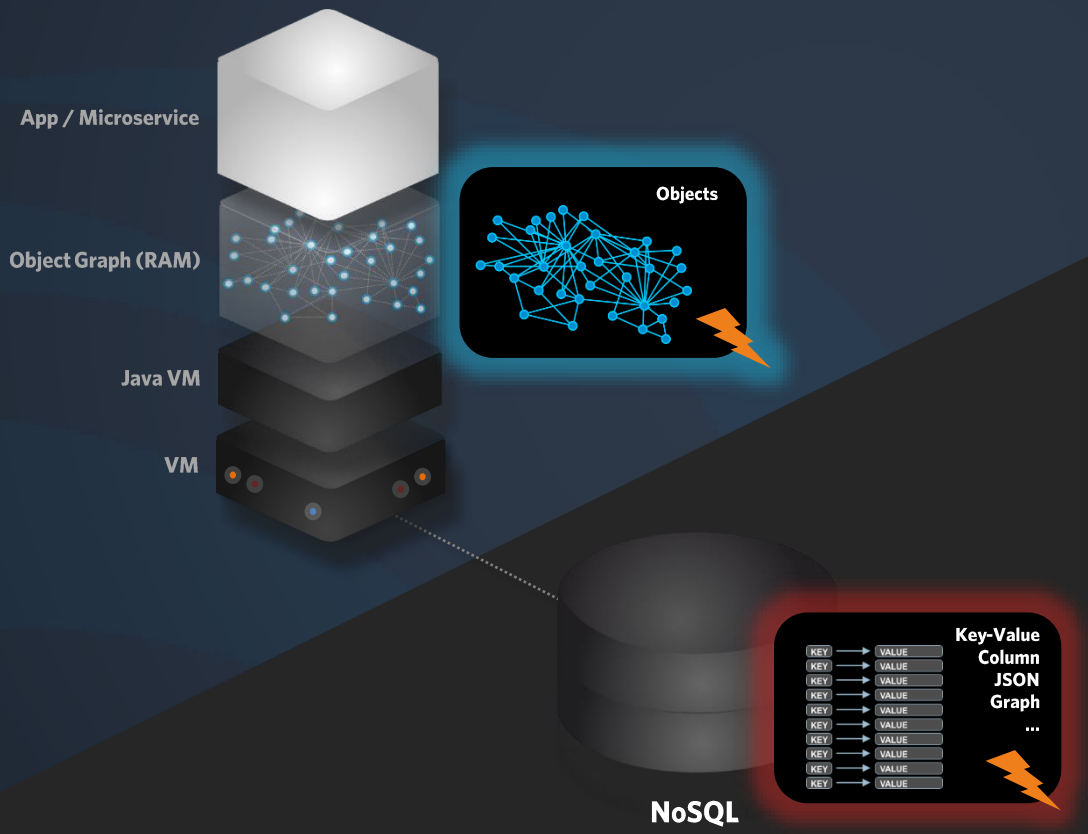
Java and Relational Databases are Incompatible



Impedance Mismatches

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

Java and NoSQL Databases are Also Incompatible



Impedance Mismatches

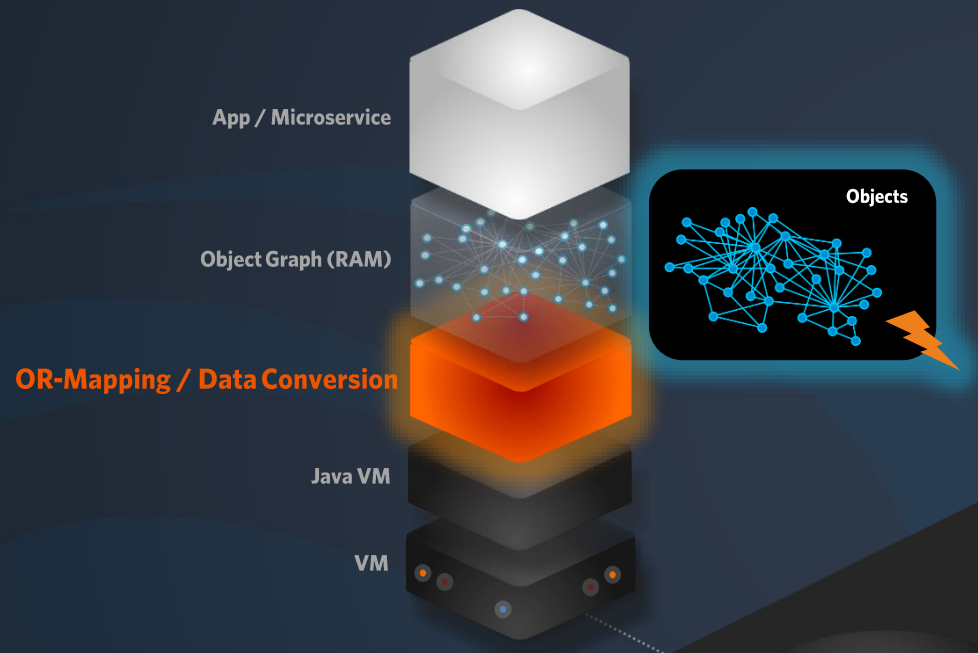
- **Key-Value**
- **Document** (JSON, XML)
- **Column Store**
- **Graph** (Proprietary)
- **OODB** (Proprietary)
- **Time-Series**

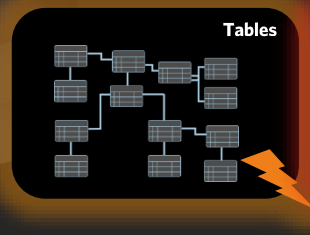
The Problem of Incompatible Data Structures is Well Known as Impedance Mismatch

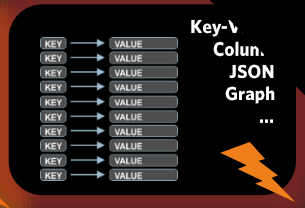
There are various solutions, but they are only a more or less elegant way around the problem. No matter which solution you choose - as long as the systems are different, every developer will sooner or later get to the point where his solution no longer meets one or more of the following points: Maintainability,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

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

Object-Relational Mapping / Data Conversion







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

Data Conversion Through Every Single Read & Write !

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

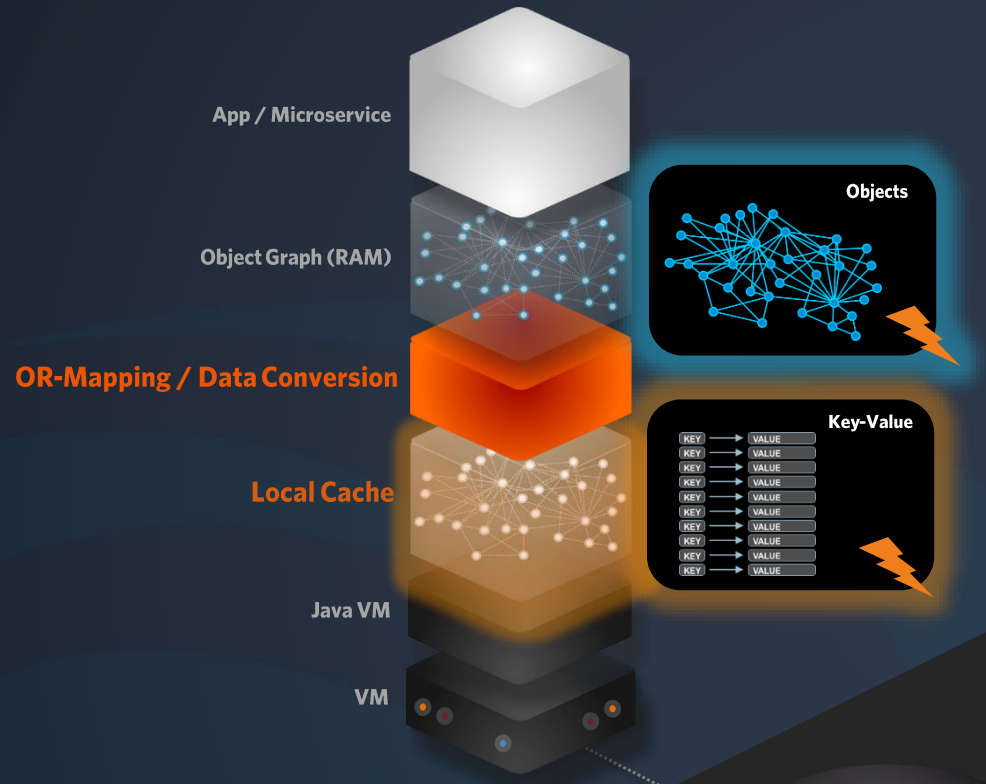
Application Layer

Storage Layer

Millisecond

Query Time

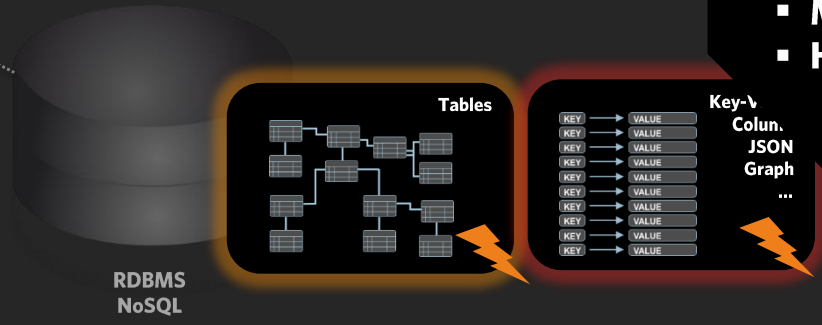
Local Cache: Better Performance, But Additional Complexity



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

Data Conversion Through Every Single Read & Write !

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



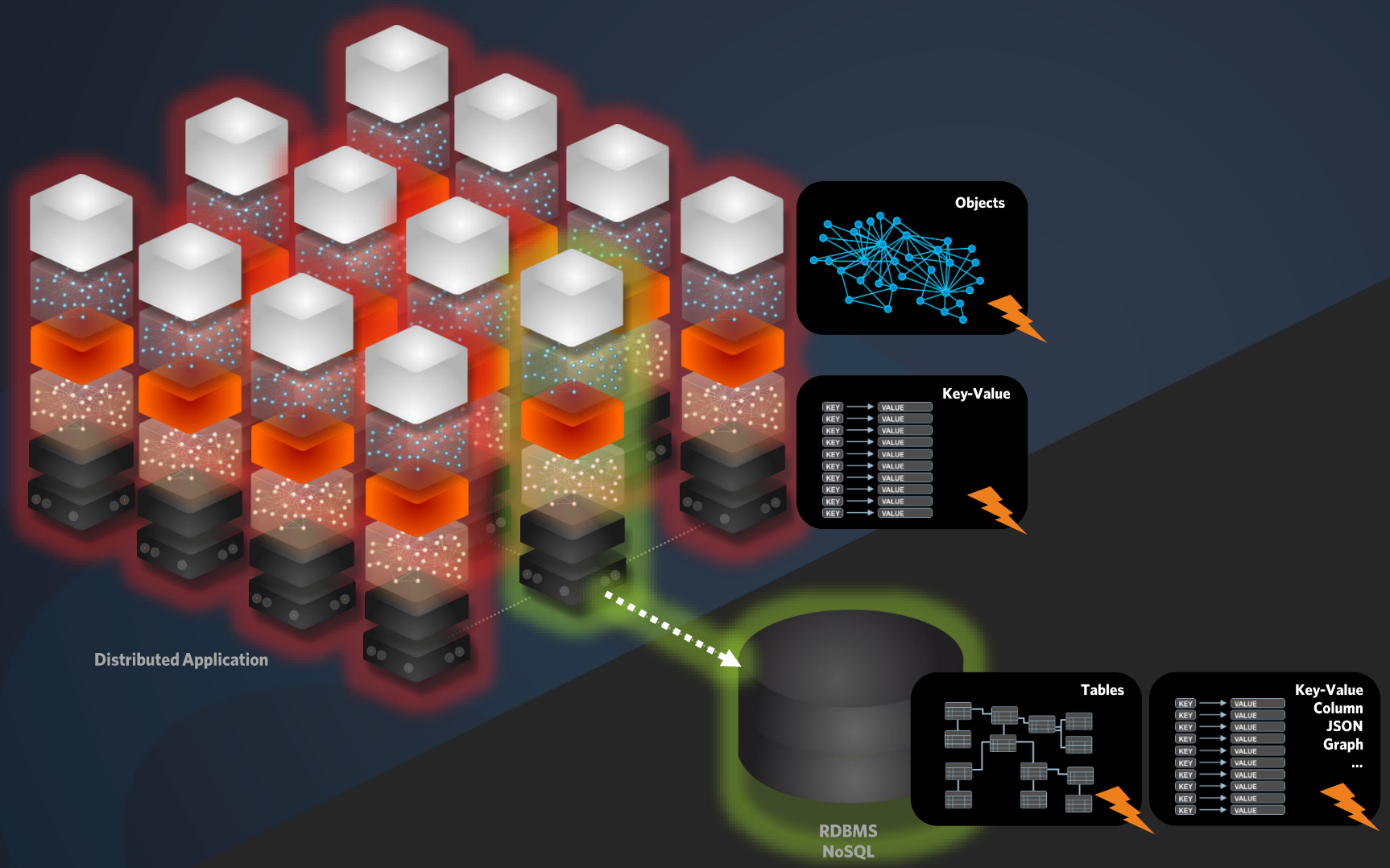
Application Layer

Storage Layer

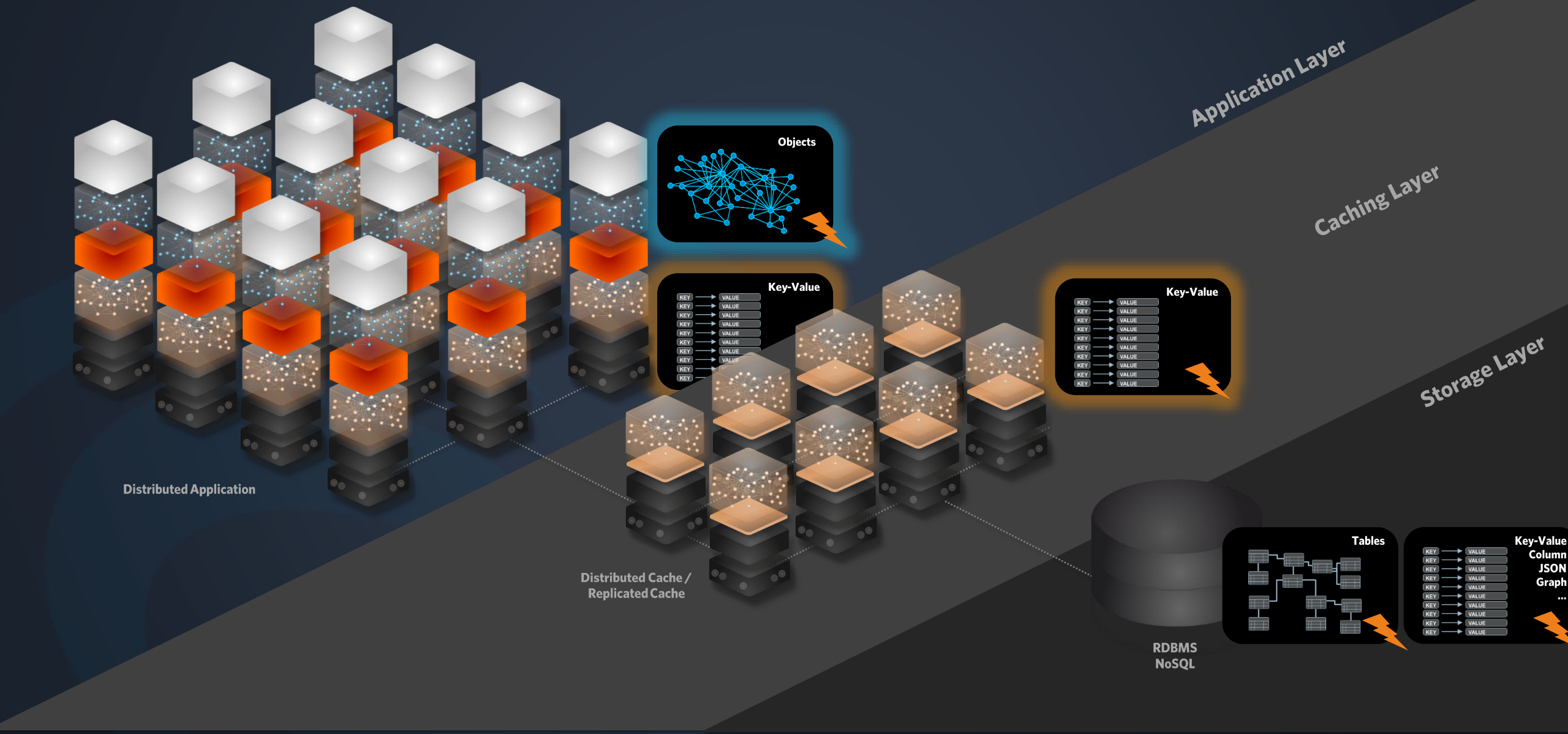
Millisecond

Query Time

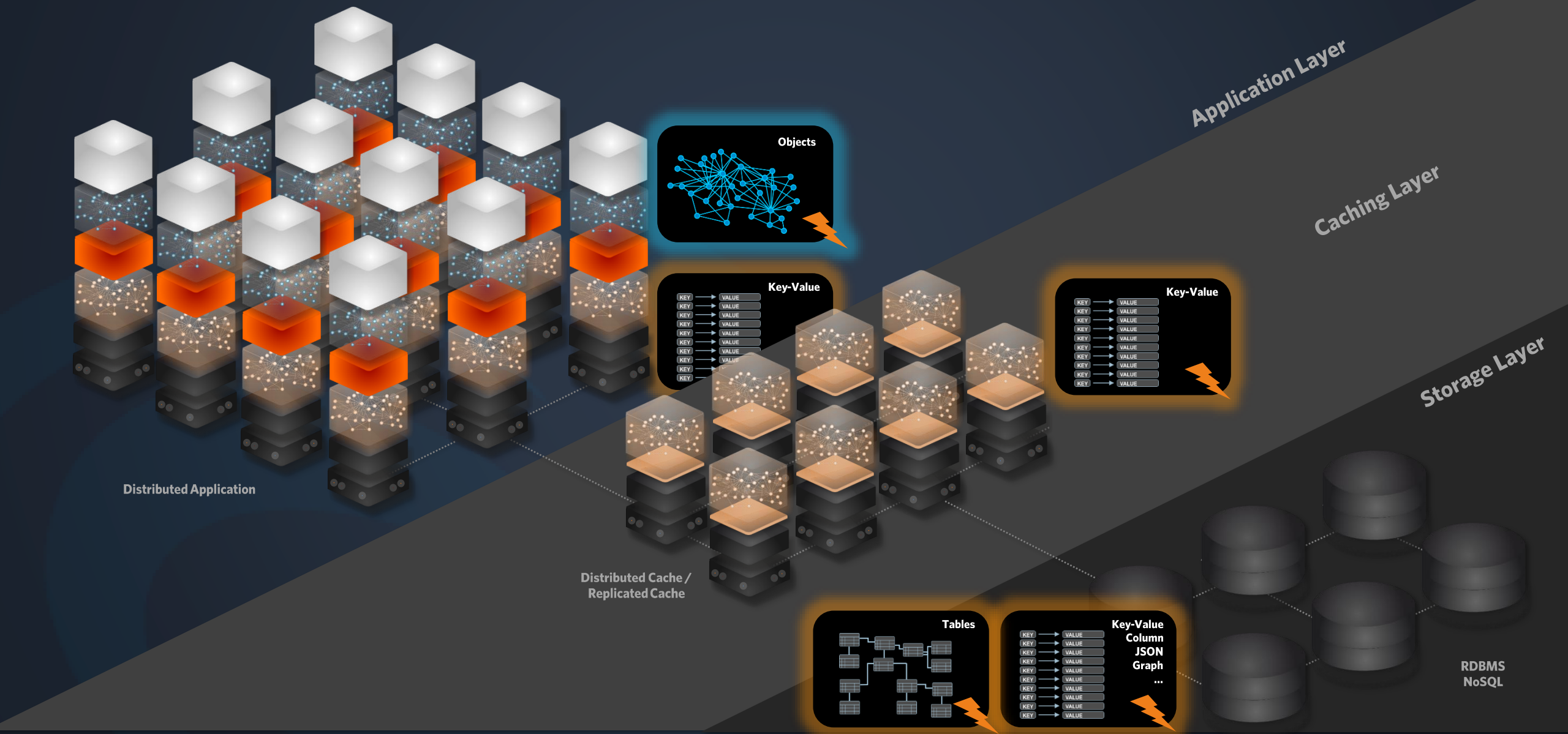
Local Caches in Distributed Apps Can Cause Inconsistent Data



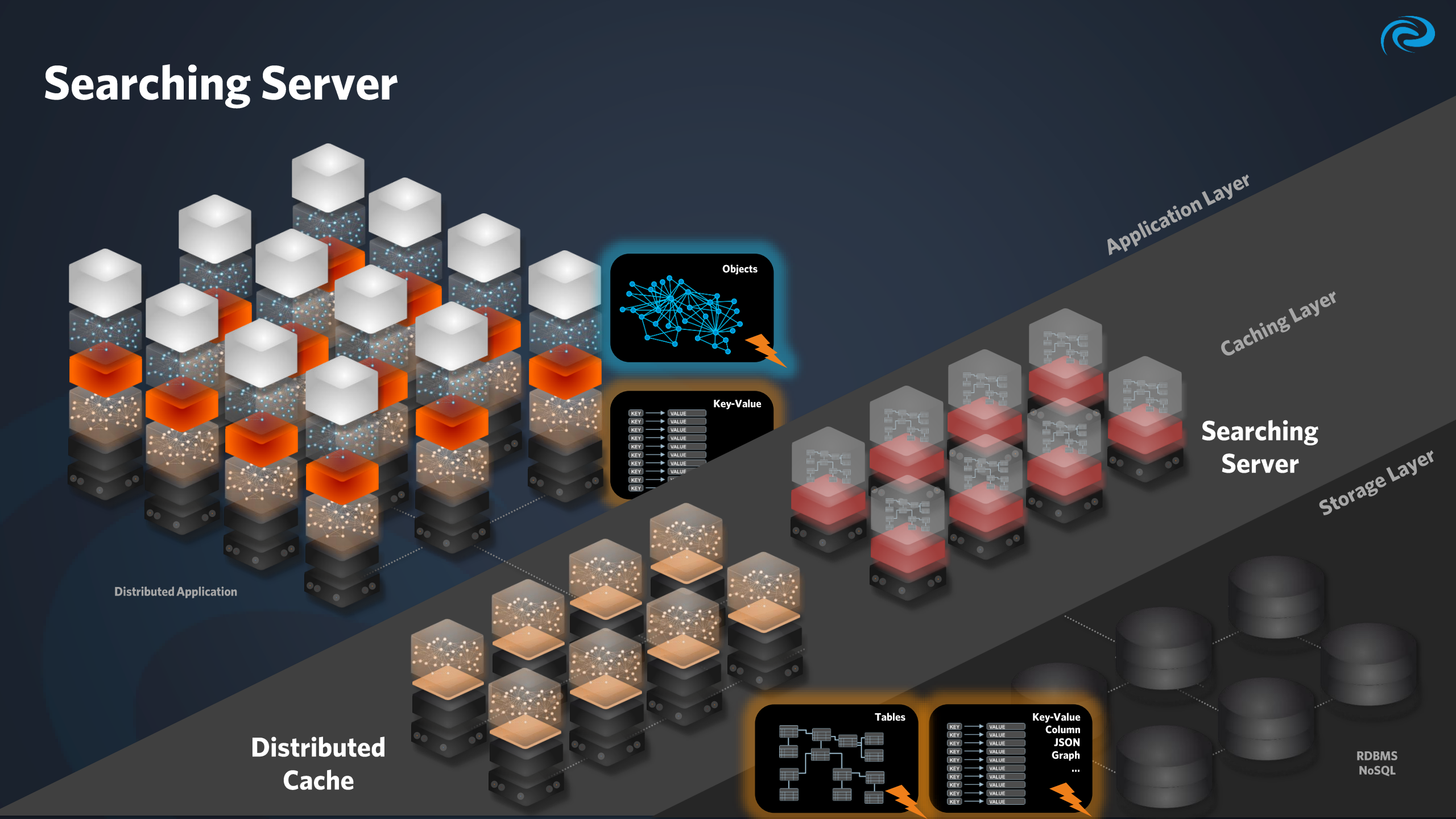
Distributed Cache Architecture



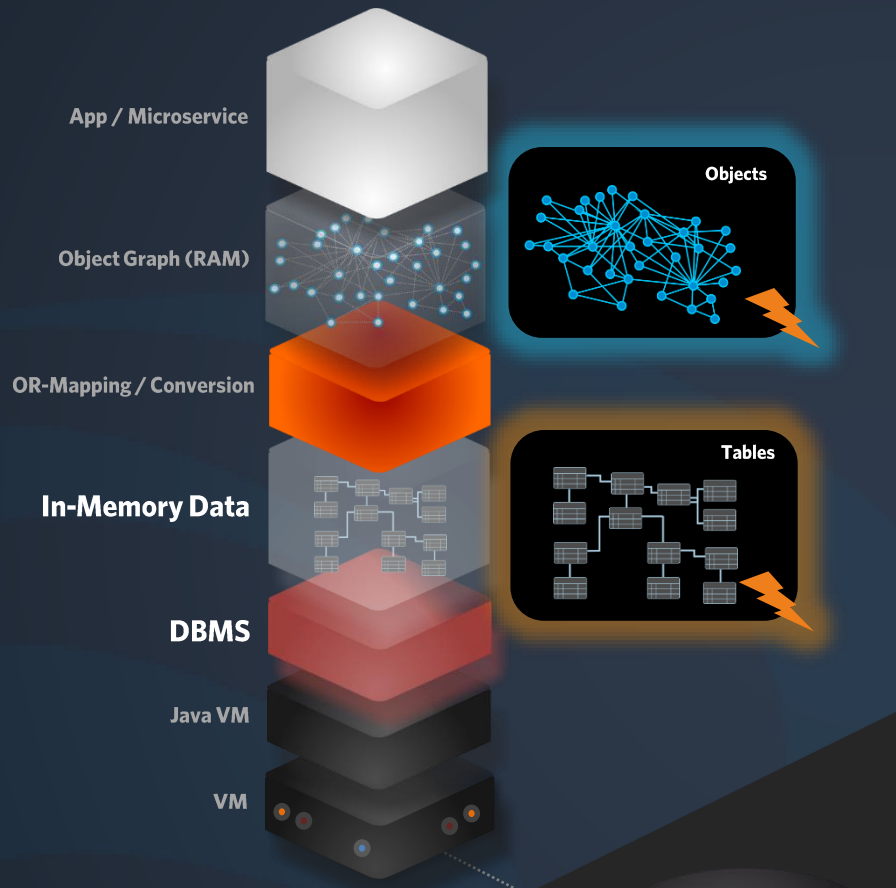
Distributed Systems: 3 Distributed System Layers in Total



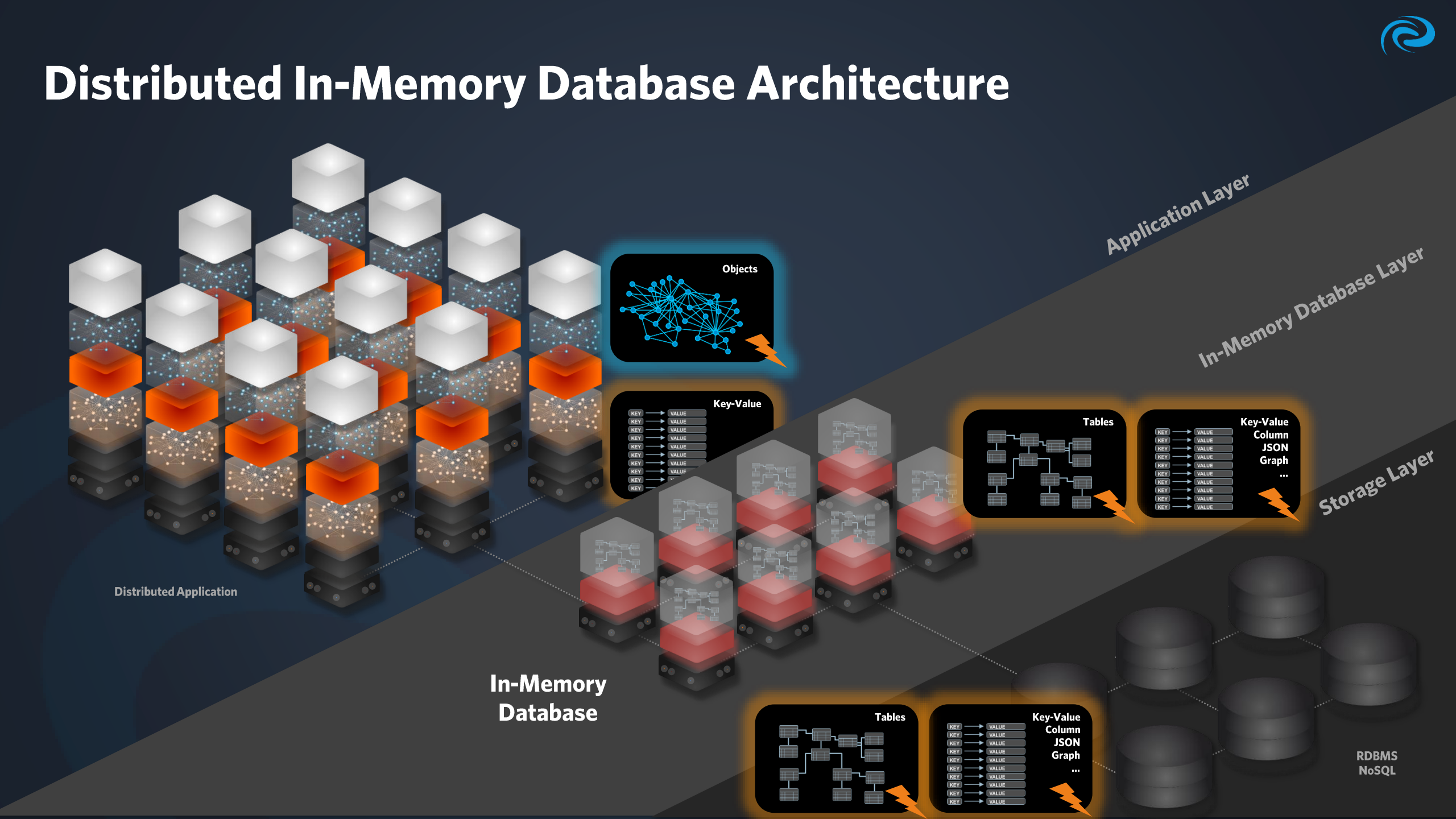
Searching Server



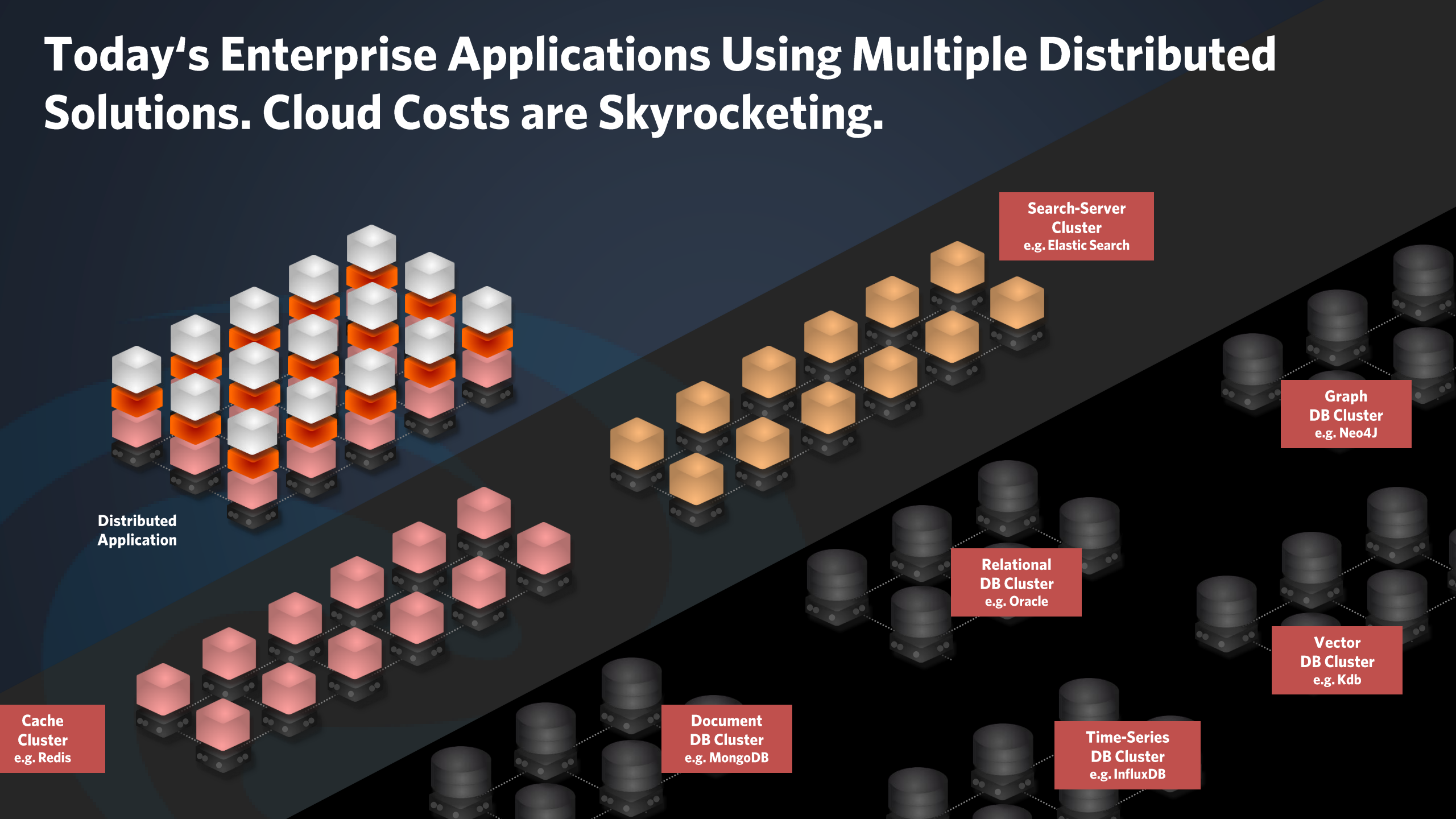
Embedded In-Memory Database Architecture



Distributed In-Memory Database Architecture



Today's Enterprise Applications Using Multiple Distributed Solutions. Cloud Costs are Skyrocketing.



Java In-Memory Data Processing

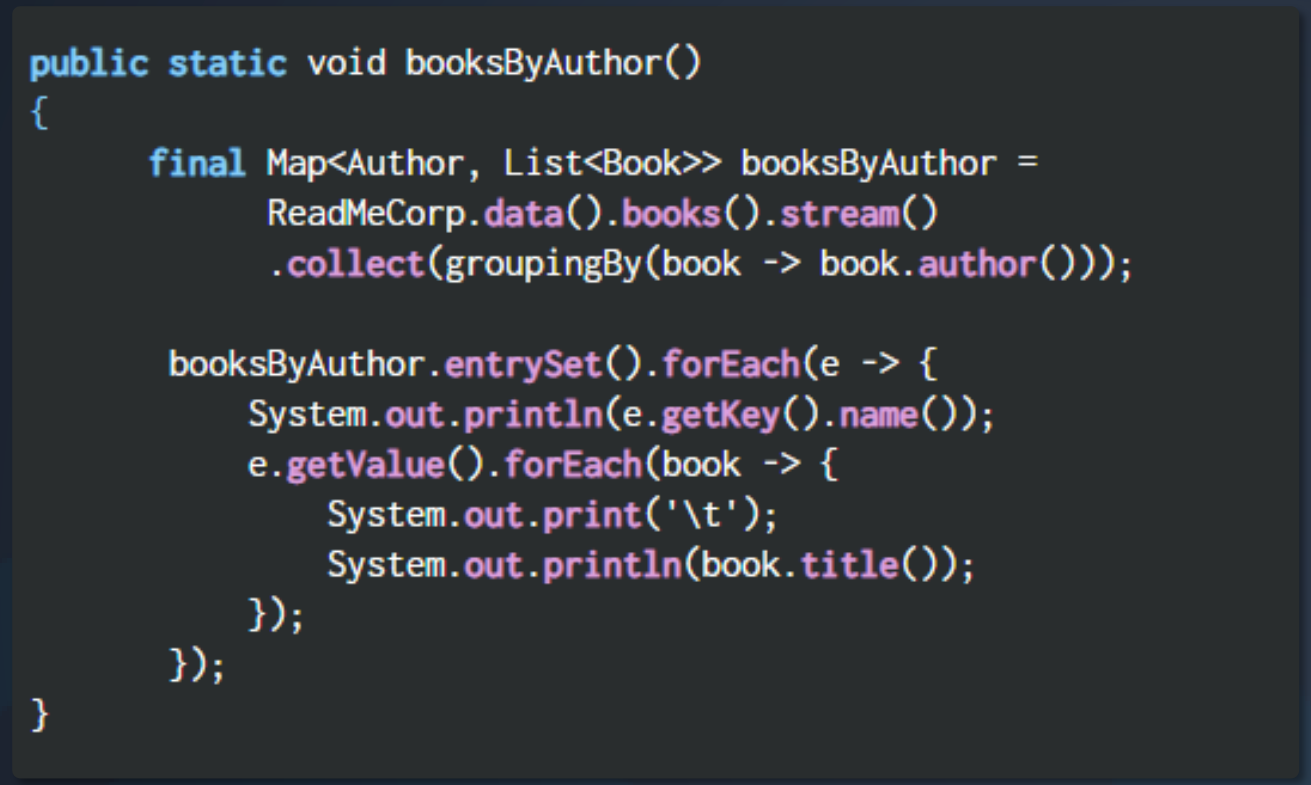
Java is extremely Fast. The JVM's JIT compiler compiles your code int machine code and optimizes it constantly through the runtime. By using newer Java versions, your code will become even better and faster. Thus, Java can in some cases even outperform C programs which are static. So, everything you build and run with pure Java in the memory will be incredible fast: Data acccess and searching times in microseconds, realtime responsivness at pure Java simplicity.

Any Data Model

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

ECLIPSE FOUNDATION OpenJ9

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



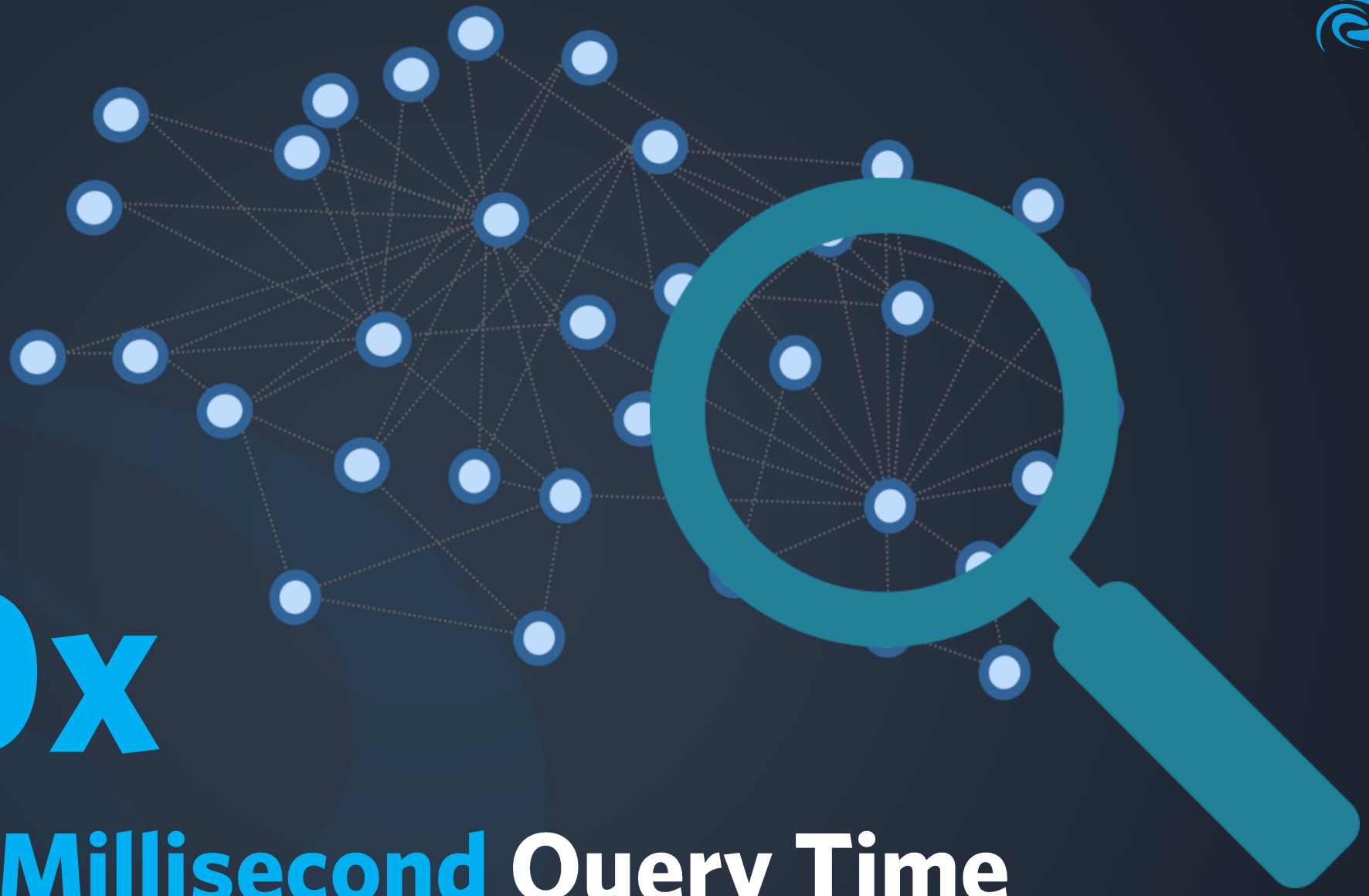
```
public static void booksByAuthor()
{
      final Map<Author, List<Book>> booksByAuthor =
            ReadMeCorp.data().books().stream()
            .collect(groupingBy(book -> book.author()));

       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

ECLIPSE FOUNDATION OpenJ9

ECLIPSE FOUNDATION OpenJ9



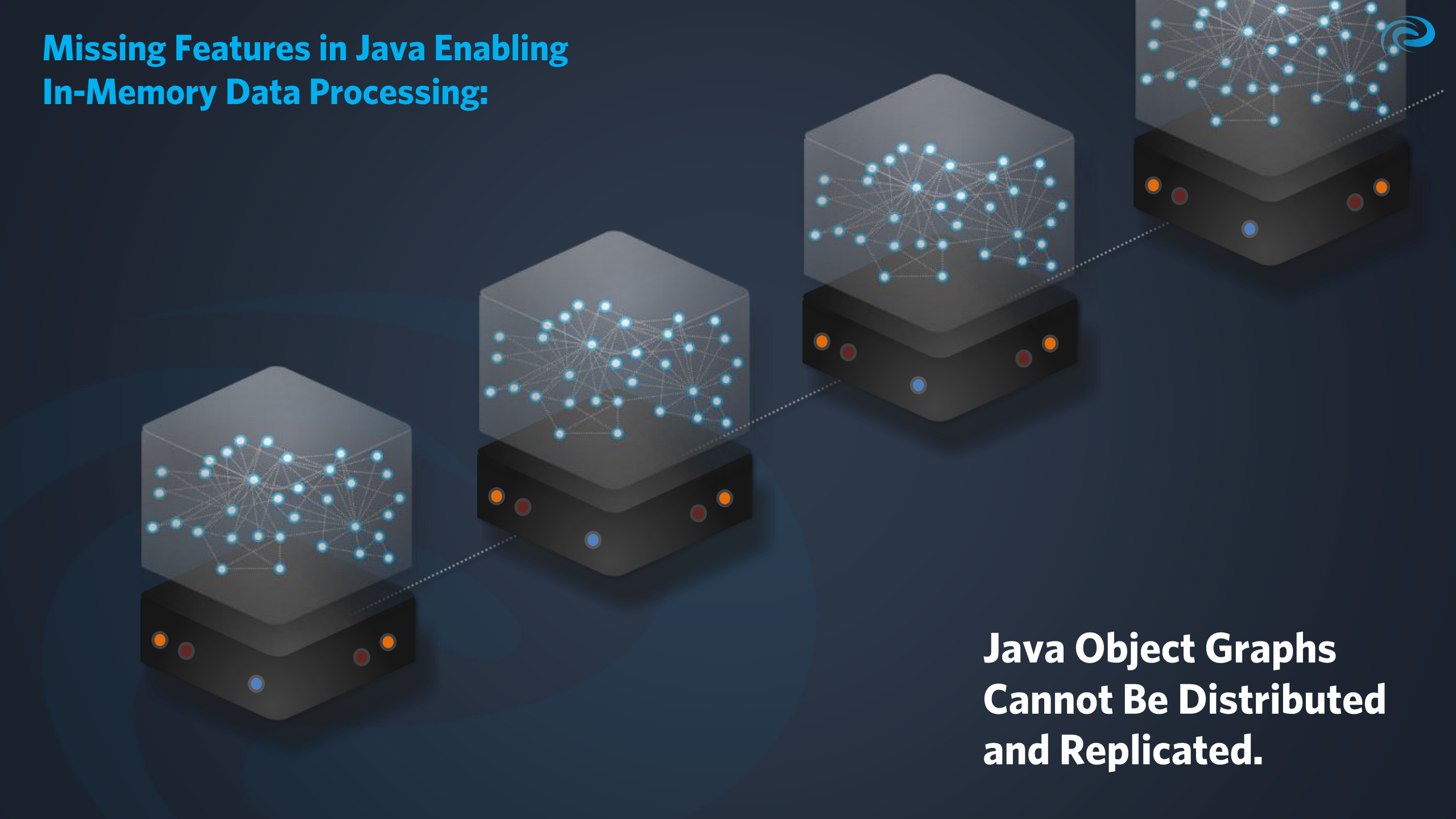
1000x

Databases: Millisecond Query Time

Java Streams API: Microsecond Query Time

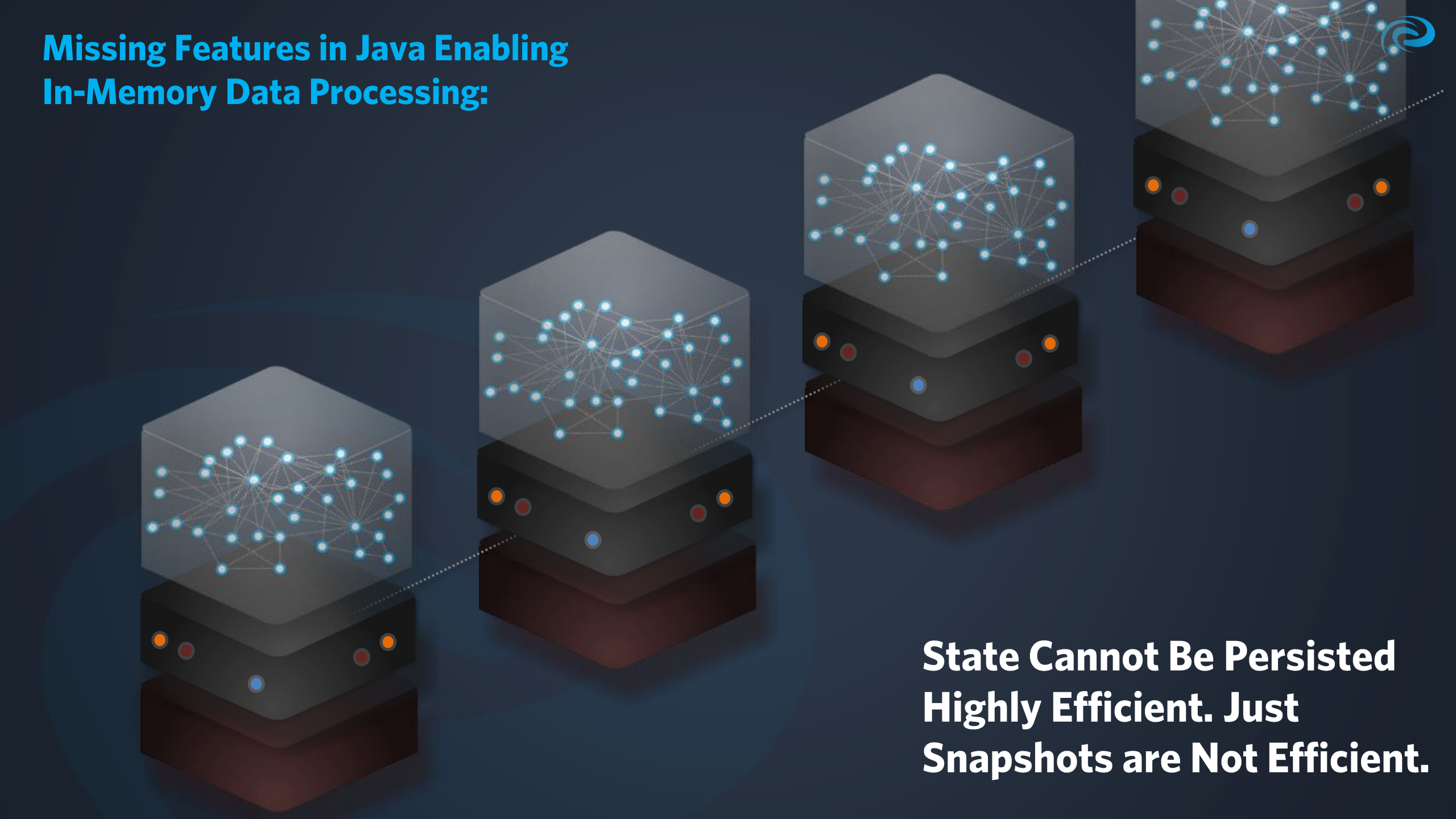
Missing Features in Java For In-Memory Data Processing

Missing Features in Java Enabling In-Memory Data Processing:



Java Object Graphs Cannot Be Distributed and Replicated.

Missing Features in Java Enabling In-Memory Data Processing:

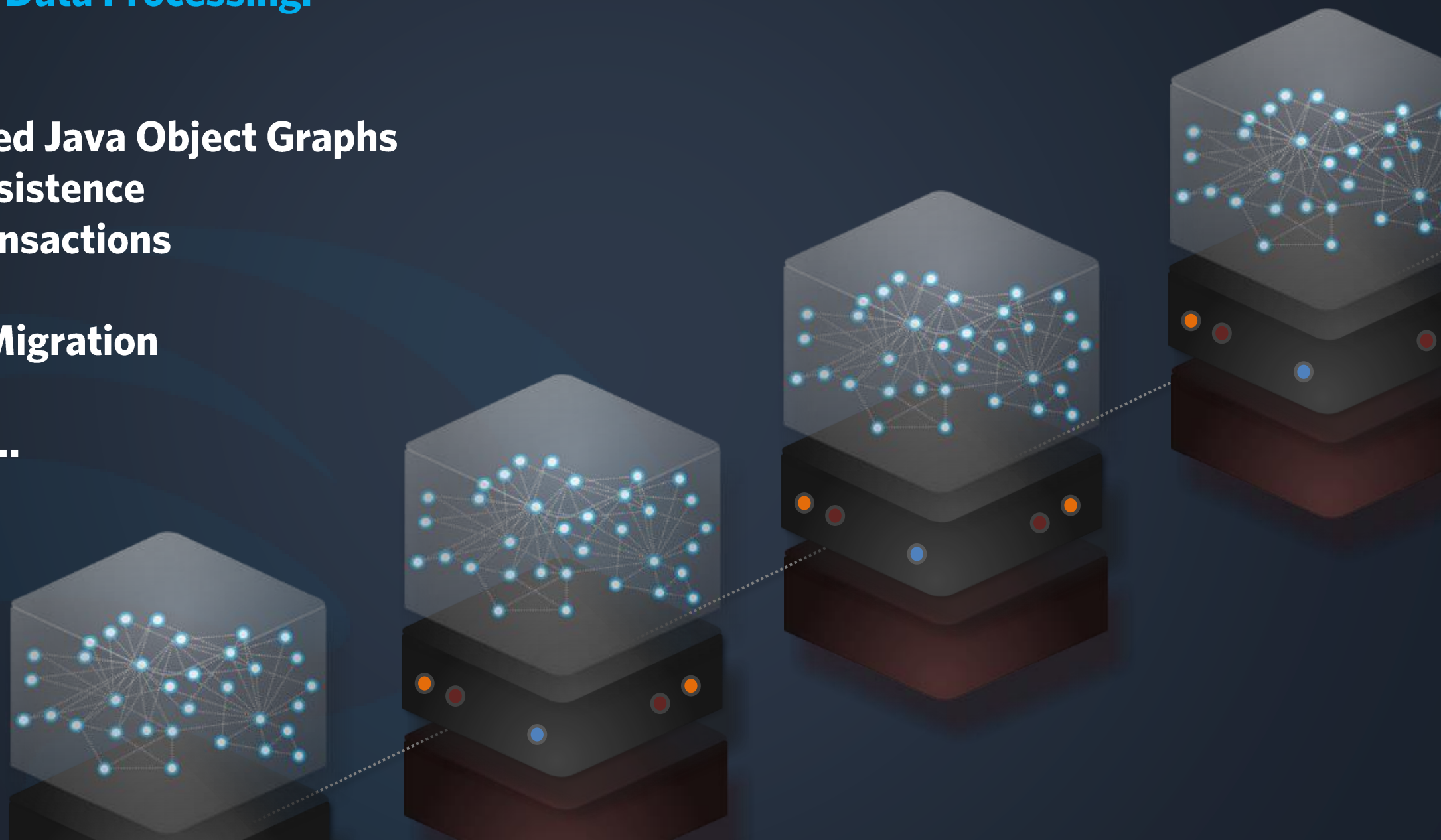


State Cannot Be Persisted Highly Efficient. Just Snapshots are Not Efficient.

Missing Features in Java Enabling In-Memory Data Processing:

- Distributed Java Object Graphs
- State Persistence
- ACID Transactions
- Indexing
- Schema Migration

and more ....



Imagine You Get All of These Features ...

Eclipse Data Grid

Ultra-Fast In-Memory Data Processing with Pure Java.

Caching. Searching. Indexing. Java Business Logic. Persistence. Distributed. Open-Source.

Java In-Memory Data Processing Foundation.

Eclipse Data Grid Features

Indexing

Extreme Fast Off-Heap Bitmap Index to Minimize IO Operations.

GigaMap

Powerful Map Enabling Fully Automated Lazy-Loading.

Locking API

Enabling Locking and Simplifying Concurrency Handling on the Java Object Graph Level.

Lucene Integration

Extends Eclipse Data Grid With Full-Text Search Capabilities.

ACID Persistence

Persisting Any Java Object Graph Transaction-Safe In Various Storage Targets.

Highly-Secure Serialization

Persisting Any Java Object Graph Transaction-Safe In Various Storage Targets.

Object Graph Replication

Distributing Any Java Object Graph Through Multiple JVMs in a Cluster.

Cluster

Configuration Files For Creating an Eclipse Data Grid Cluster Environment Based on Kubernetes.

Native Java Features

Native Java Object Model

Eclipse Data Grid Supports the Native Java Object Model. No Mappings Or Conversions. Just POJOs.

Java Streams API

Searching Java Object Graphs with Java Streams API.

Any Logic in Java

Implement Any Business Logic in Java.

OpenJ9 JVM

OpenJ9 Is Highly Performant and Memory Efficient, and Provides Fast Start-Up Time.

Java

ECLIPSE FOUNDATION

OpenJ9

Step 1: Java In-Memory Computing: Pure Java Concpets Only.

Forget database concepts, focus on Java concepts only! Use Core Java concepts and any Java library and framework.

- Design any complex Java object model
- Use any Java types
- Use just POJOs
- Search with Java Streams

...

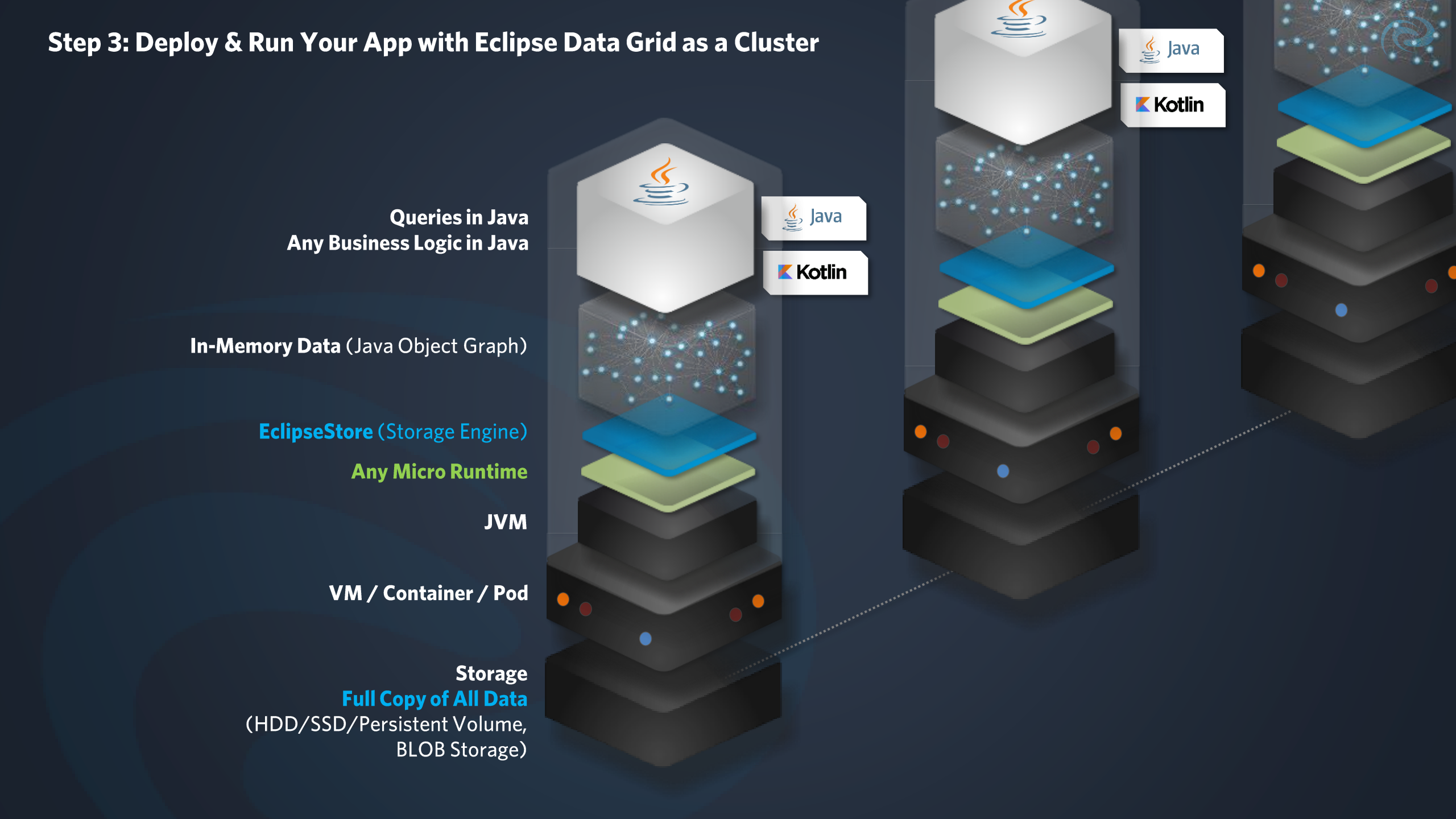
Step 2: Built-In Persistence and Replications with EclipseStore

EclipseStore

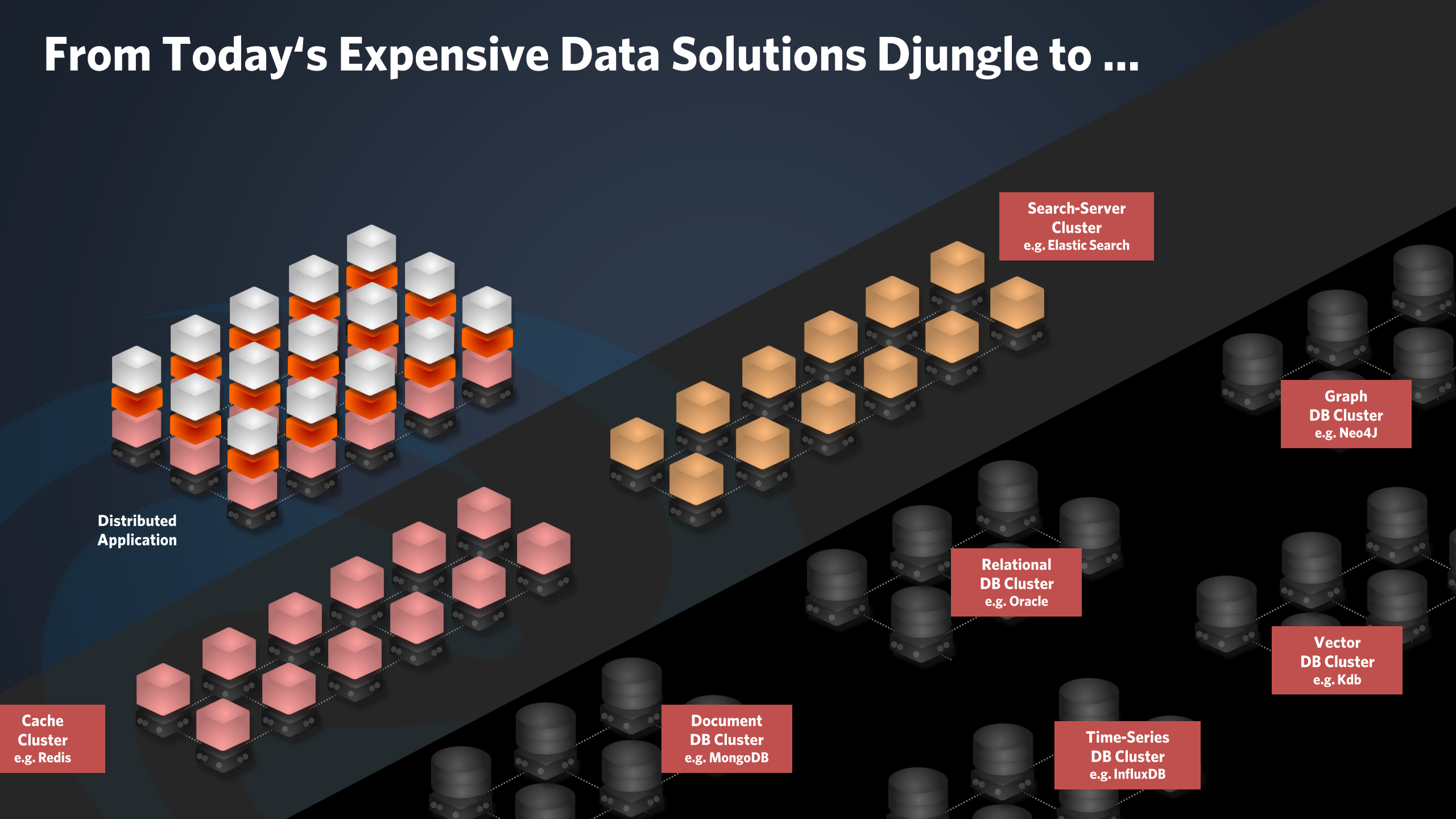
EclipseStore is a persistence engine to store any Java object graph of any size and complexity into any binary data storage, and to restore it completely or partially in RAM on demand.

www.eclipsestore.io

Step 3: Deploy & Run Your App with Eclipse Data Grid as a Cluster

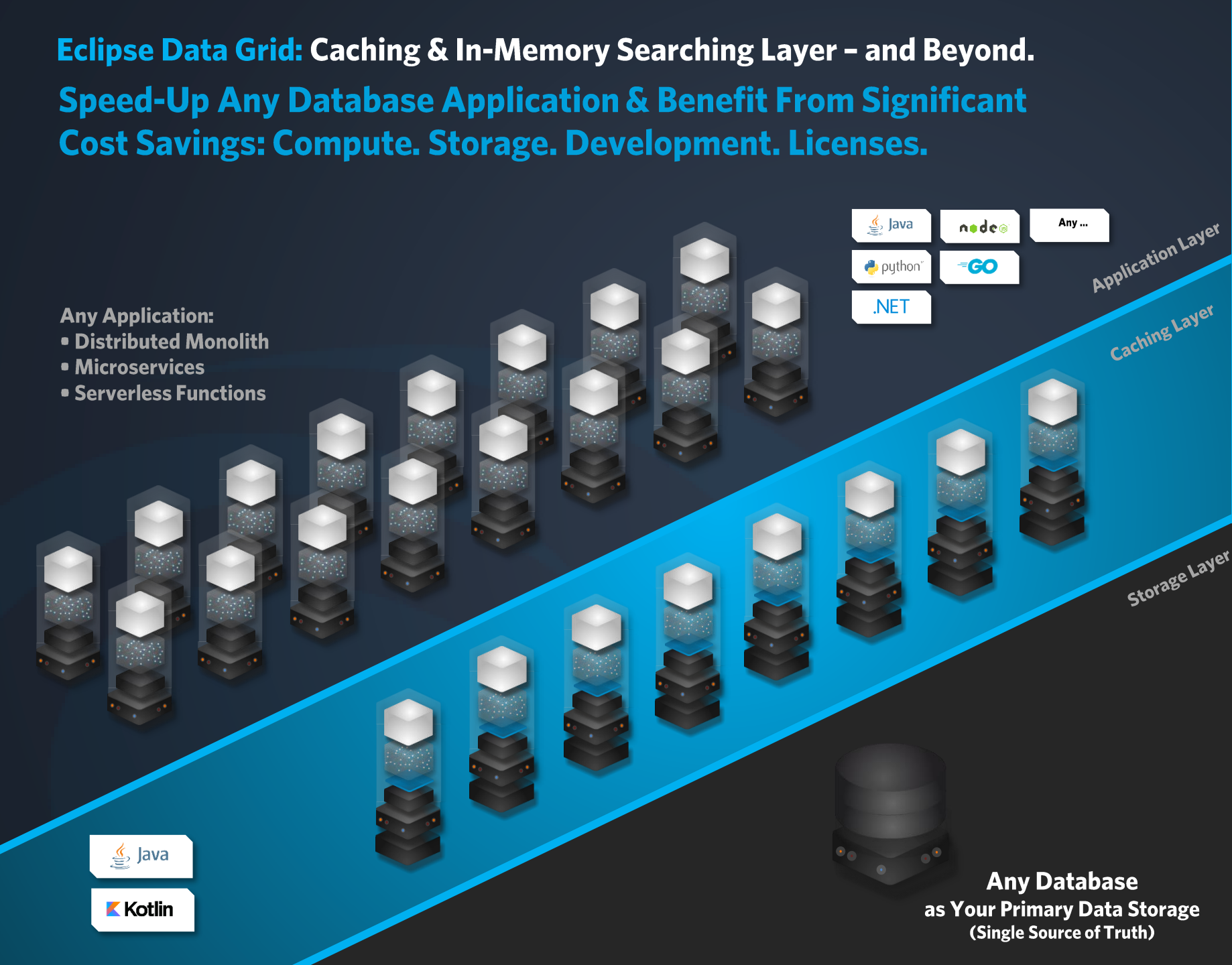


From Today's Expensive Data Solutions Djungle to ...



Eclipse Data Grid: Caching & In-Memory Searching Layer – and Beyond.

Speed-Up Any Database Application & Benefit From Significant Cost Savings: Compute. Storage. Development. Licenses.



Eclipse Data Grid

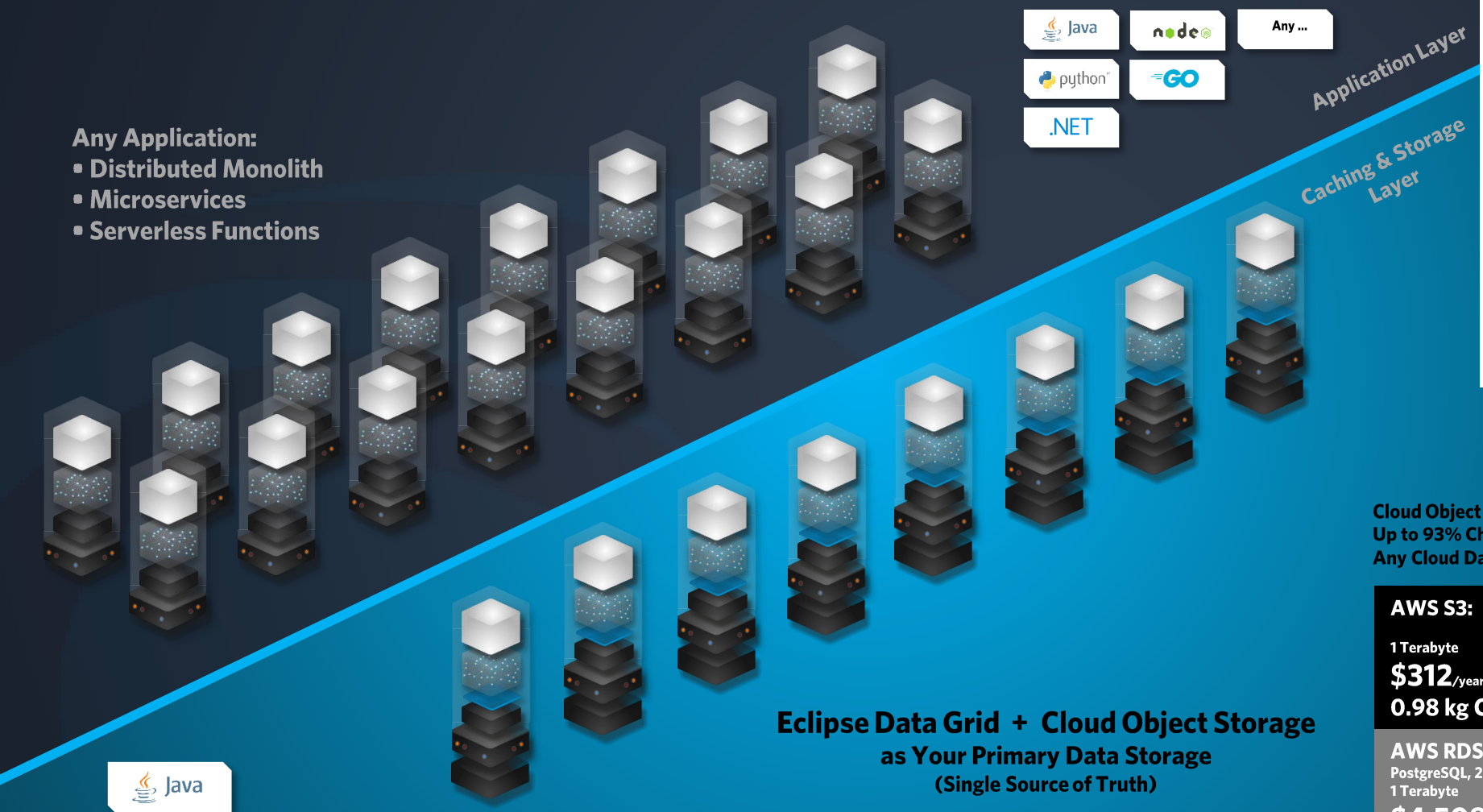
- ✓ Caching
- ✓ Searching
- ✓ Indexing
- ✓ Native Java Object Model
- ✓ Any Logic in Java
- ✓ ACID Persistence

1000x
Faster Queries In-Memory

IBM Cloud

Eclipse Data Grid: As Primary Data Storage.

Speed-Up Any Database Application & Benefit From Dramatic Cost Savings: Compute. Storage. Development. Licenses.



Eclipse Data Grid + Cloud Object Storage
as Your Primary Data Storage
(Single Source of Truth)

amazon web services · Azure · Google Cloud · ORACLE CLOUD · IBM Cloud

Java · Kotlin

Note: This architecture is most powerful and efficient, but in practice it is mostly refused by traditionl database teams which is the biggest show-stopper for this solution.

Cloud Object Storage is Up to 93% Cheaper Than Any Cloud Database:

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

AWS RDS:
PostgreSQL, 2 CPU, 8 GB RAM
1 Terabyte
$4,508/year
618 kg CO2 /year

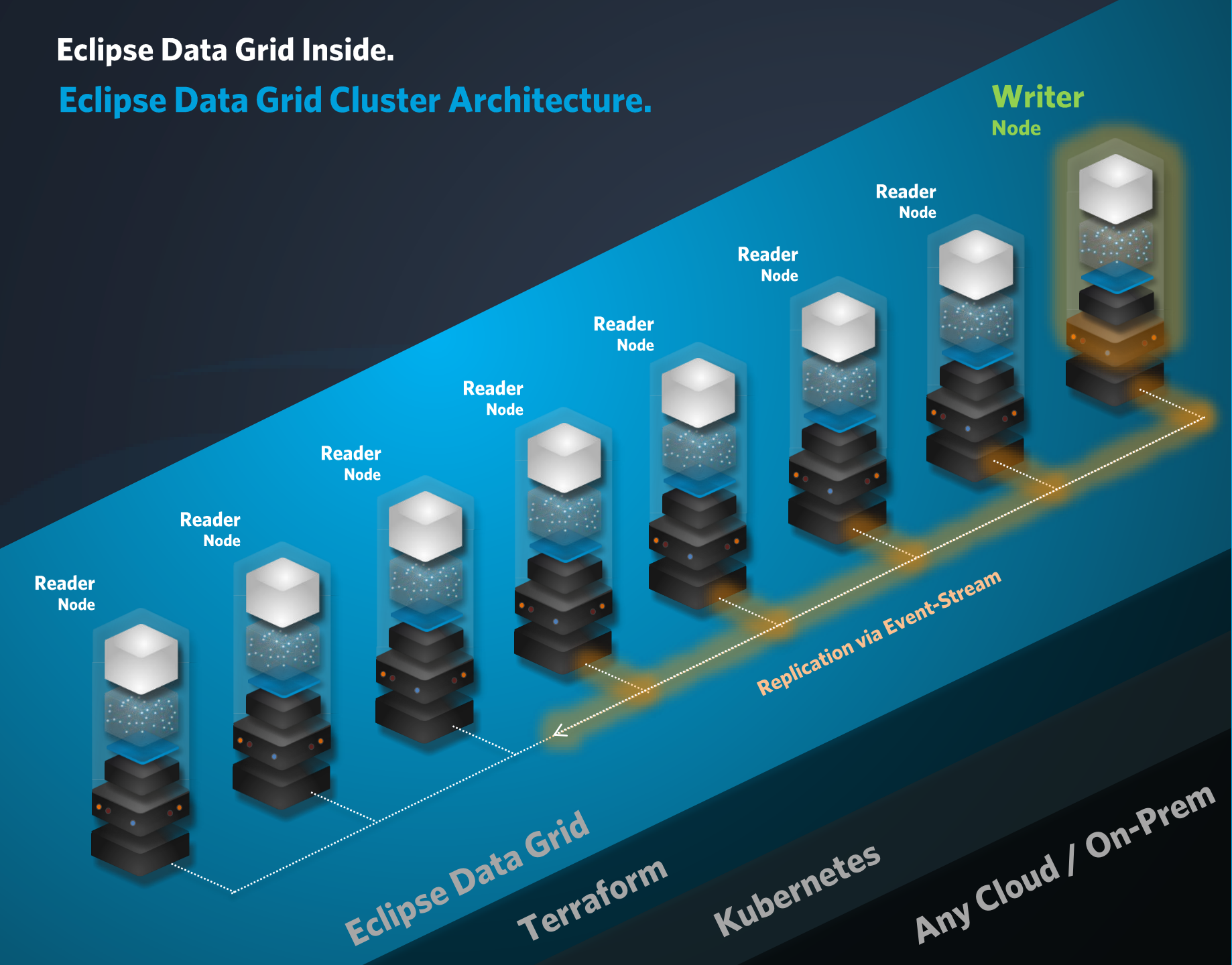
1000x
Faster Queries In-Memory

93%
Database Cost Savings

99%
Energy & CO2 Emission Savings

Eclipse Data Grid Inside.

Eclipse Data Grid Cluster Architecture.



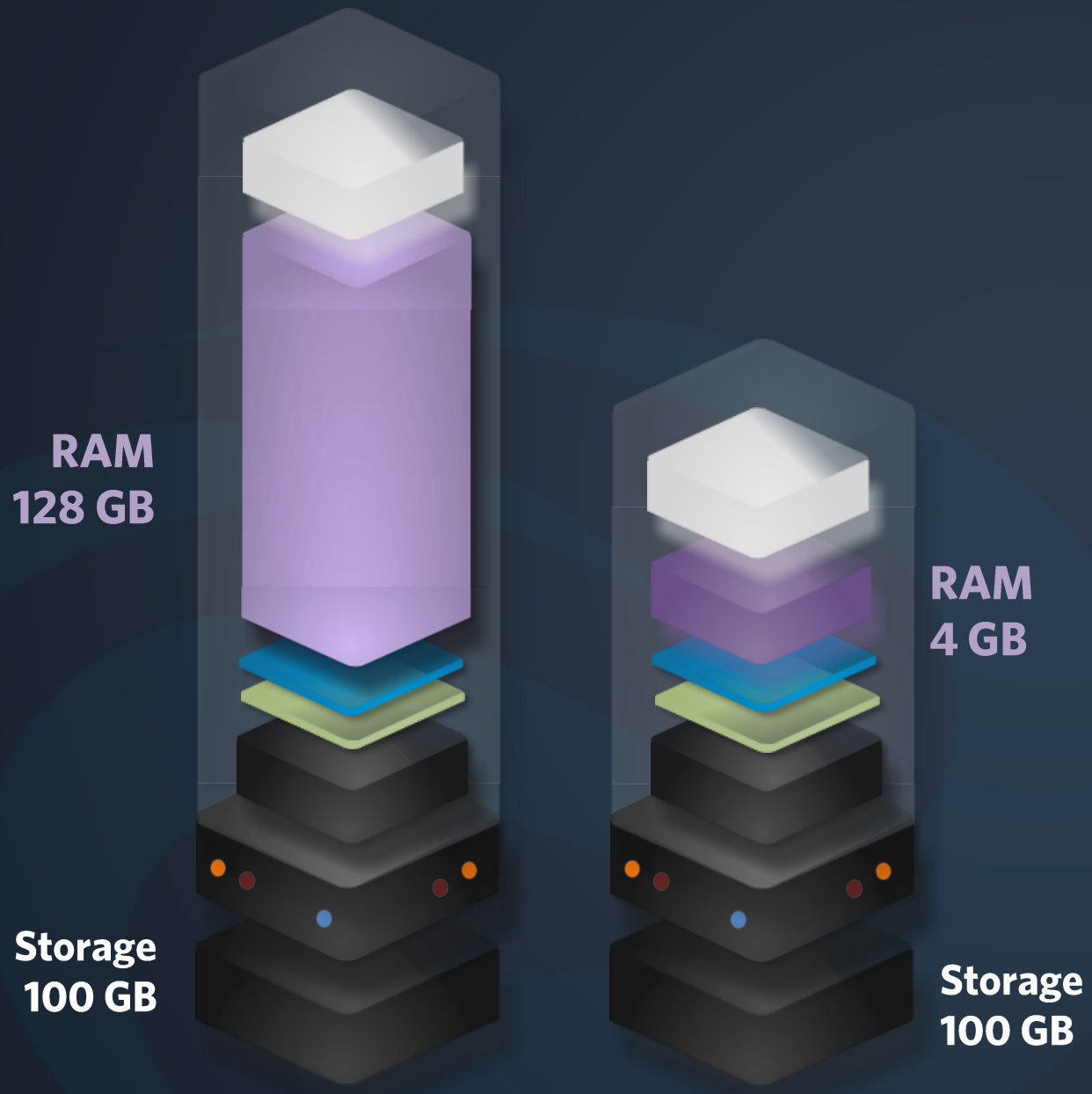
Eclipse Data Grid

- ✔ Single Writer
- ✔ Replication
- ✔ Auto Fail-Over
- ✔ Data Redundancy
- ✔ High Availability
- ✔ Elastic Scalable
- ✔ Self Healing
- ✔ Multi Region Backup

Consistency Levels:

- Local Node: Strong Consitency
- Writer Node: Strong Consistency
- Cluster:
  - Standard: Eventual Consistency
  - Customizable: Strong Consistency

Small RAM vs. Big RAM: Choose a Storage <> RAM Ratio that Fits For Your Use-Case.



Lazy-Loading Retrieving Data from SSD Drives is Enormously Fast: Mostly Millisecond Latencies

Full In-Memory Computing is the Fastest Approach.

	Small RAM	Big RAM
IO Ops	More	Less
Performance	Lower	Higher
Start-Up	Faster	Slower
Infrastructure Costs	Lower	Higher

The OpenJ9 Advantage

- Superior RAM Footprint & Efficiency: Up to 65% Smaller
- Advanced Garbage Collection for In-Memory Performance
- Faster Ramp-Up to Peak Throughput
- JIT Optimization Levels

Eclipse Data Grid – Enterprise-Grade Open Source.

CYROCK.AI

Open Source

Eclipse Data Grid
In-Memory Data Processing Platform

EclipseStore
Java Native Persistence

Eclipse Serializer
High-Secure Java Serialization

Eclipse OpenJ9 / Any JVM
Java Virtual Machine

Recommended, but not Mandatory!

Enterprise

SaaS
Multi - Cloud

- Managed
- Built-In Security
- Serverless
- Management UI
- Multi-Cloud

On-Prem
Requires only Kubernetes

- Tested Builds
- Full Ownership
- Data Sovereignty
- Customizing

- Enterprise Support
- Consulting, Implementation, Partner Support & Mentoring

ECLIPSE FOUNDATION OpenJ9

amazon

kubernetes

Move Complex Data Processing & Manipulation to a Java In-Memory Layer

Why Eclipse Data Grid?

High Performance:
Microsecond Response- & Query-Time

Data Model:
Native Java Object Model

Multi-Data-Model:
Supporting Any Java Type, any Document, any Structure, any Complexity

Pure Java:
Write Query & Any Business Logic in Pure Java

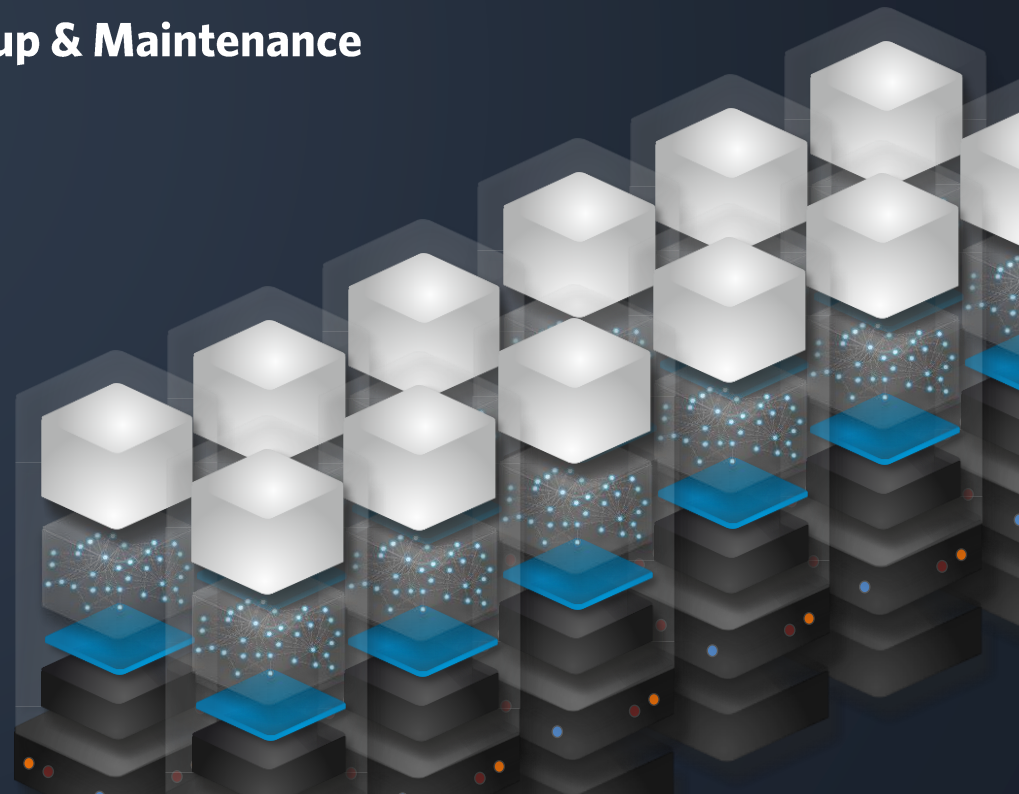
Persistence:
ACID-Compliant Persistence. Can Be Used as a Primary Data Storage

Distributed:
Distributed, Replication, Data Redundancy

Security:
Built-In Security

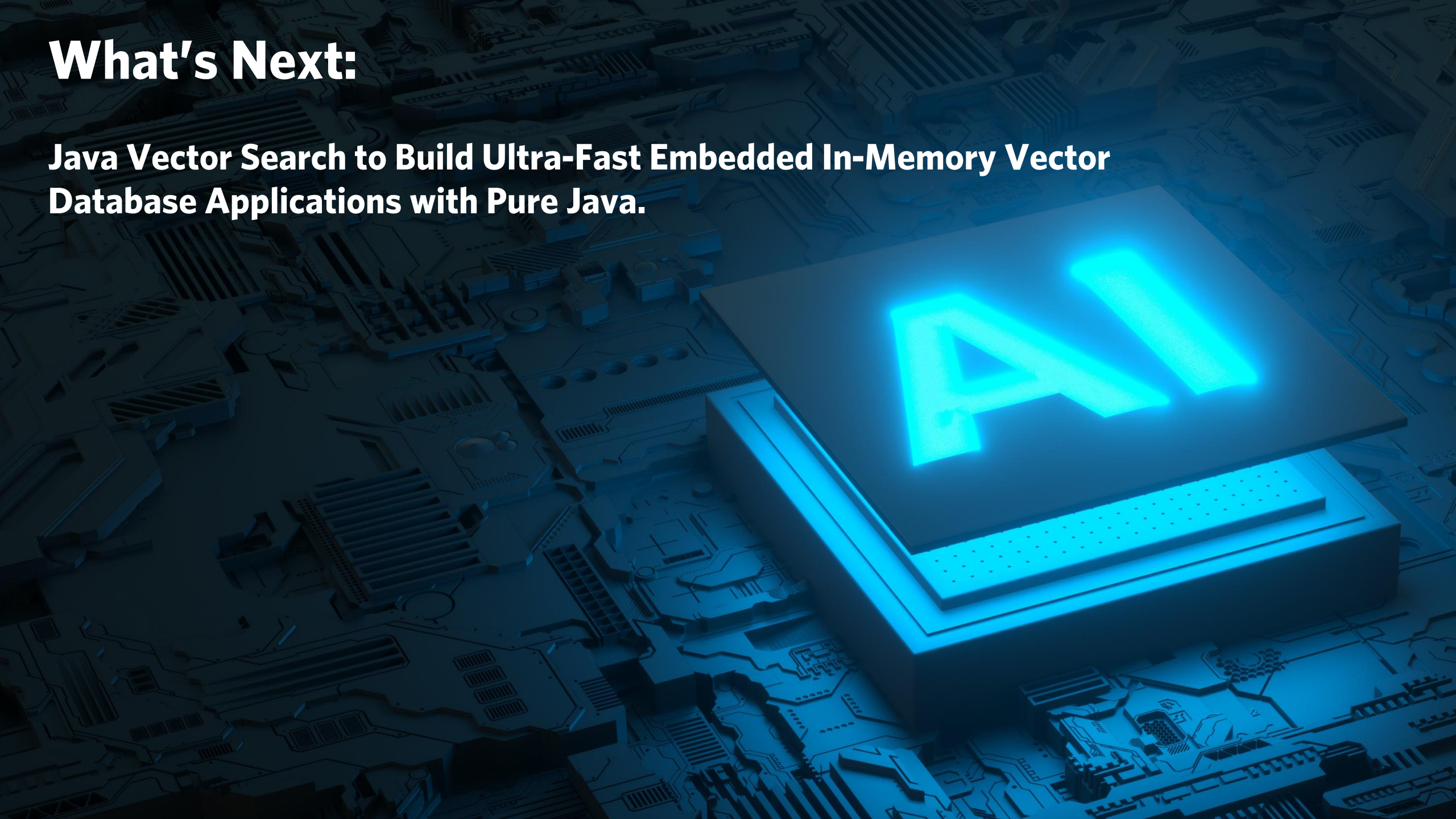
Simplicity:
Convenient & Easy Setup & Maintenance

Available Platforms:
SaaS & On-Prem



What's Next:

Java Vector Search to Build Ultra-Fast Embedded In-Memory Vector Database Applications with Pure Java.



What's Next:

Vector Search

NEW !

Cyrock.AI: Java-Native Embedded In-Memory Vector Database.

VectraLink

NEW !

Synchronizing Traditional Databases with Your Java-Native Vector Database.

Eclipse Data Grid Features

Indexing

Extreme Fast Off-Heap Bitmap Index to Minimize IO Operations.

GigaMap

Powerful Map Enabling Fully Automated Lazy-Loading.

Locking API

Enabling Locking and Simplifying Concurrency Handling on the Java Object Graph Level.

Lucene Integration

Extends Eclipse Data Grid With Full-Text Search Capabilities.

ACID Persistence

Persisting Any Java Object Graph Transaction-Safe In Various Storage Targets.

Highly-Secure Serialization

Persisting Any Java Object Graph Transaction-Safe In Various Storage Targets.

Object Graph Replication

Distributing Any Java Object Graph Through Multiple JVMs in a Cluster.

Cluster

Configuration Files For Creating an Eclipse Data Grid Cluster Environment Based on Kubernetes.

Cyrock.AI

Embedded Java In-Memory Vector Database

Queries in Java
Any Business Logic in Java

In-Memory Data (Java Object Graph)

Java Vector Search

EclipseStore (Storage Engine)

Micro Runtime

JVM

VM / Container / Pod

Storage
Full Copy of All Data
(HDD/SSD/Persistent Volume)

The Classic Java Monolith + EclipseStore:

Ultra-Fast Java In-Memory Data Monster.

Cyrock.AI VectraLink:

CYROCK.AI

Eclipse Data Grid

Java In-Memory Data Platform

https://www.microstream.one

CYROCK.AI

Embedded Java In-Memory Vector Database

https://github.com/cyrock-ai/early-access/

EclipseStore

Java Native Persistence

https://www.eclipsestore.io

Contact:

Markus Kett
Co-Founder & CEO

m.kett@cyrock.ai
Cell: +49 16090766677
LinkedIn: MarkusKett

CYROCK.AI, Inc.

info@cyrock.ai
www.cyrock.ai