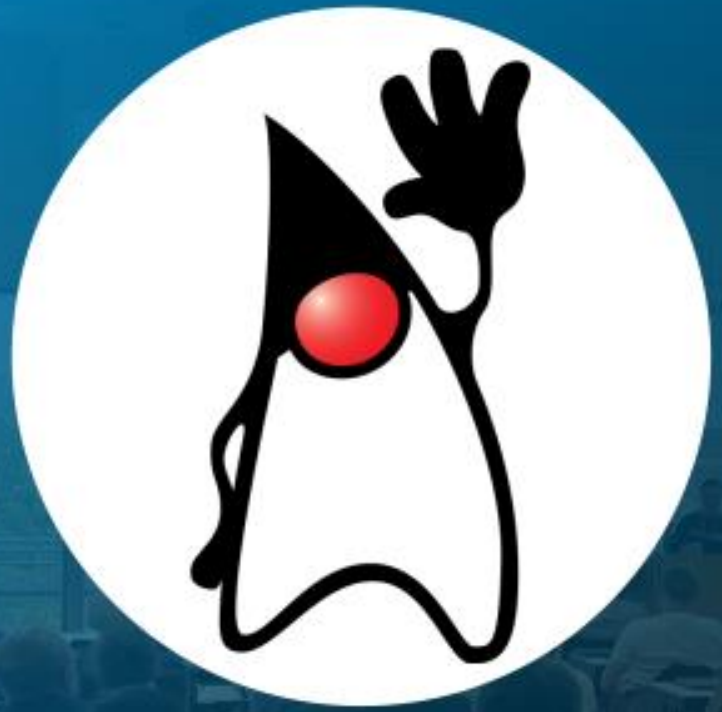
#MicroStream

MicroStream on Tour 2021

JUG Görlitz



Ultraschnelle Java In-Memory Datenbank-Anwendungen & Microservices mit MicroStream

MicroStream

Disclaimer

The following is intended to outline our general product direction. It‘s intended for informational purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for MicroStream‘s products remains at the sole discretion of MicroStream.

About me

Markus Kett
CEO at MicroStream,
Contributor to Project Helidon (Oracle)
Editor in Chief at JAVAPRO Magazine
Organizer JCON Conference
Conference Speaker

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

MicroStream

RAPID clipse™

JPA-SQL

JAVAPRO

JCON 2021

Agenda

- Challenges with database programming in Java
- Java in-memory processing approach
- MicroStream persistence
- MicroStream highly secure serialization
- MicroStream JCache
- Q&A

Developers Love

- OOP
- Type-safty
- Abstraction
- Standards
- Avoid depencencies - POJOs
- Elegant object models
- Good tested code
- Clean code
- Freedoms

Database Programming

Complicated

Inconvenient

Tinkered

Outdated

Time Consuming

Expensive

Database Applications

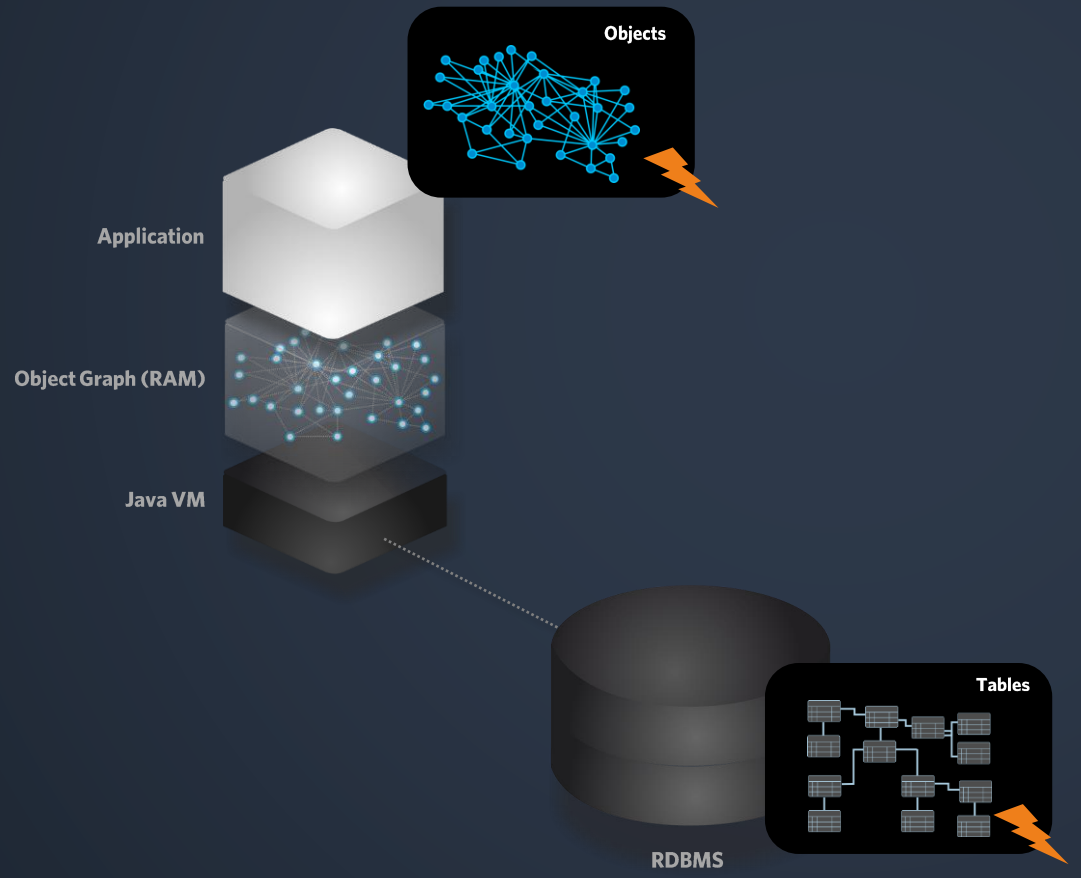
Complicated

Slow

Expensive

Traditional Java Persistence

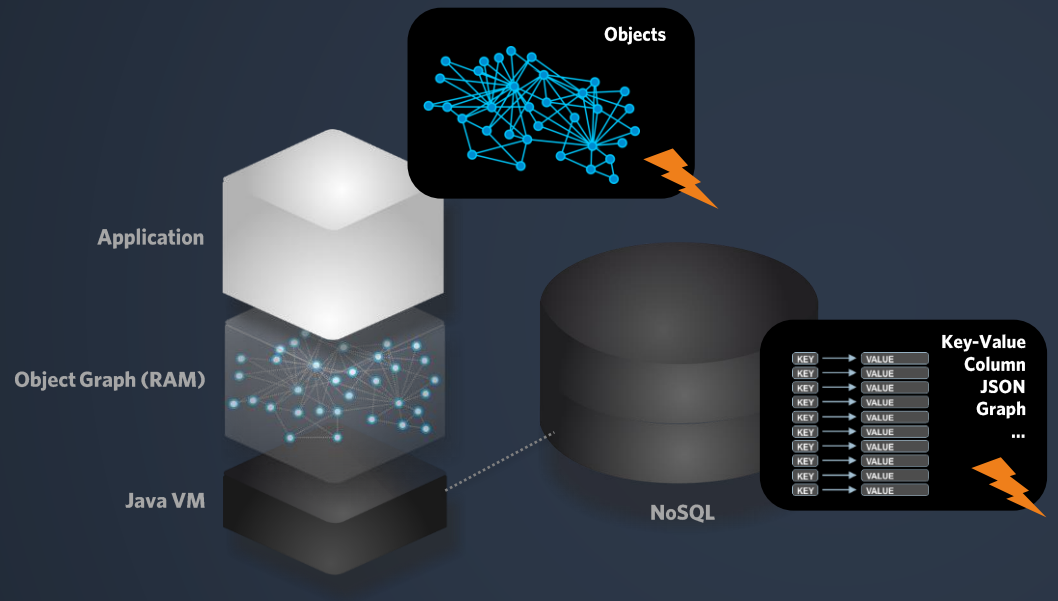
Java object graphs and RDBMS tables are incompatible.



Impedance Mismatch !

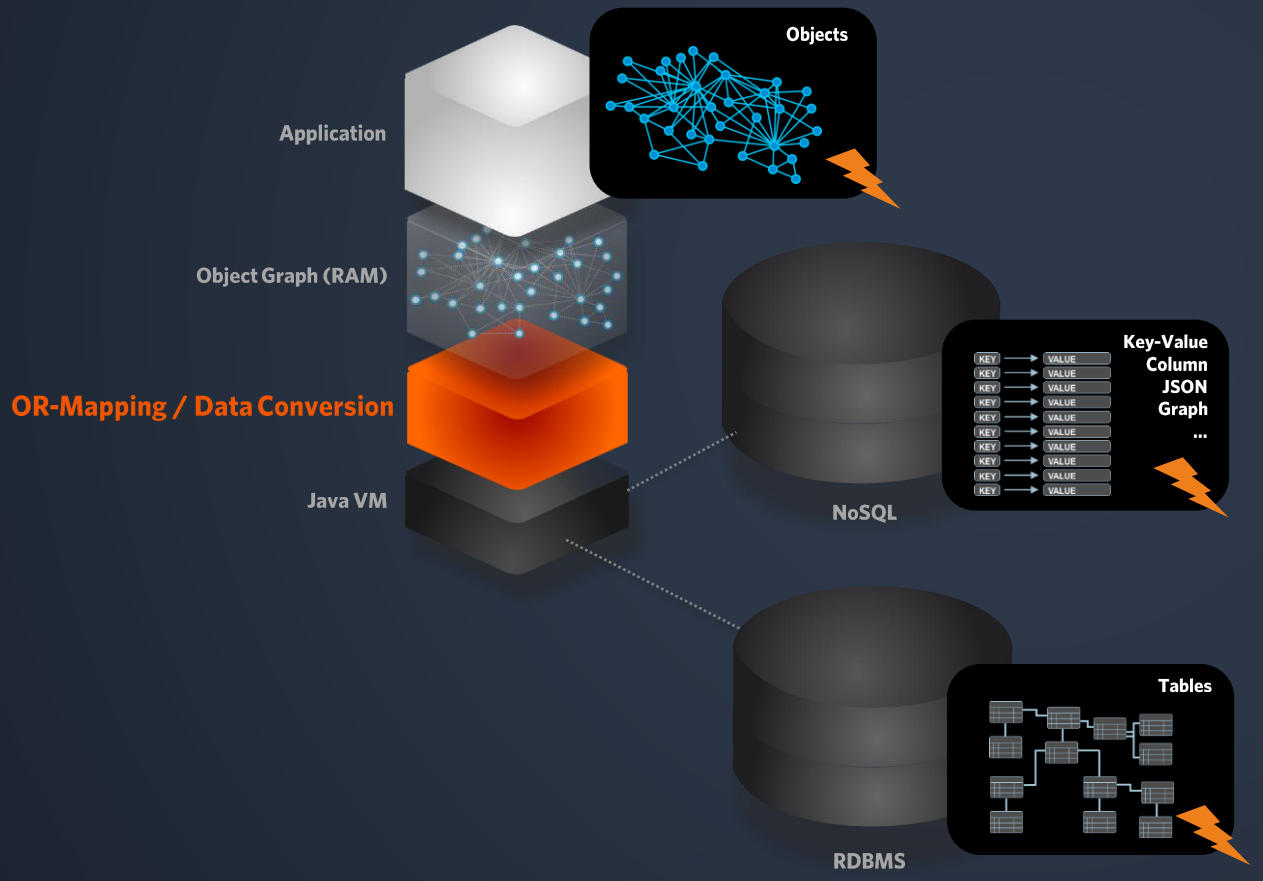
- Granularity mismatch
- Subtypes mismatch
- Identity mismatach
- Associations mismatch
- Data Navigation mismatch
- Data type differences

Modern NoSQL



ALL NoSQL data structures are also incompatible with Java object graphs. Even OO and Graph DBs are incompatible with Java object graphs. Impedance Mismatch !

OR-Mapping / Data Conversion

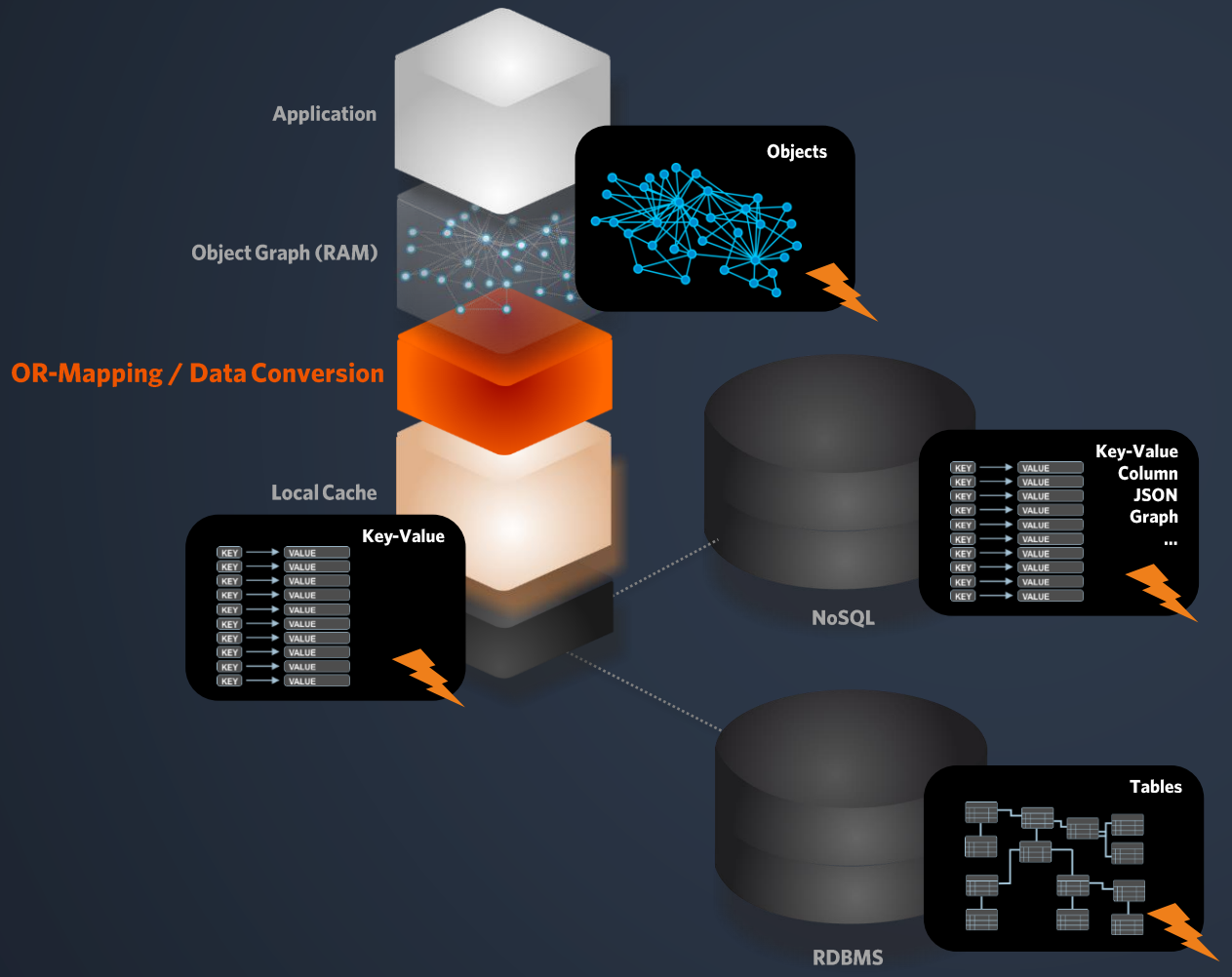


Challenge: Storing Objects into Tables / JSON / Key Value Stores / Graphs

Data Conversion Through Every Single Read & Write !

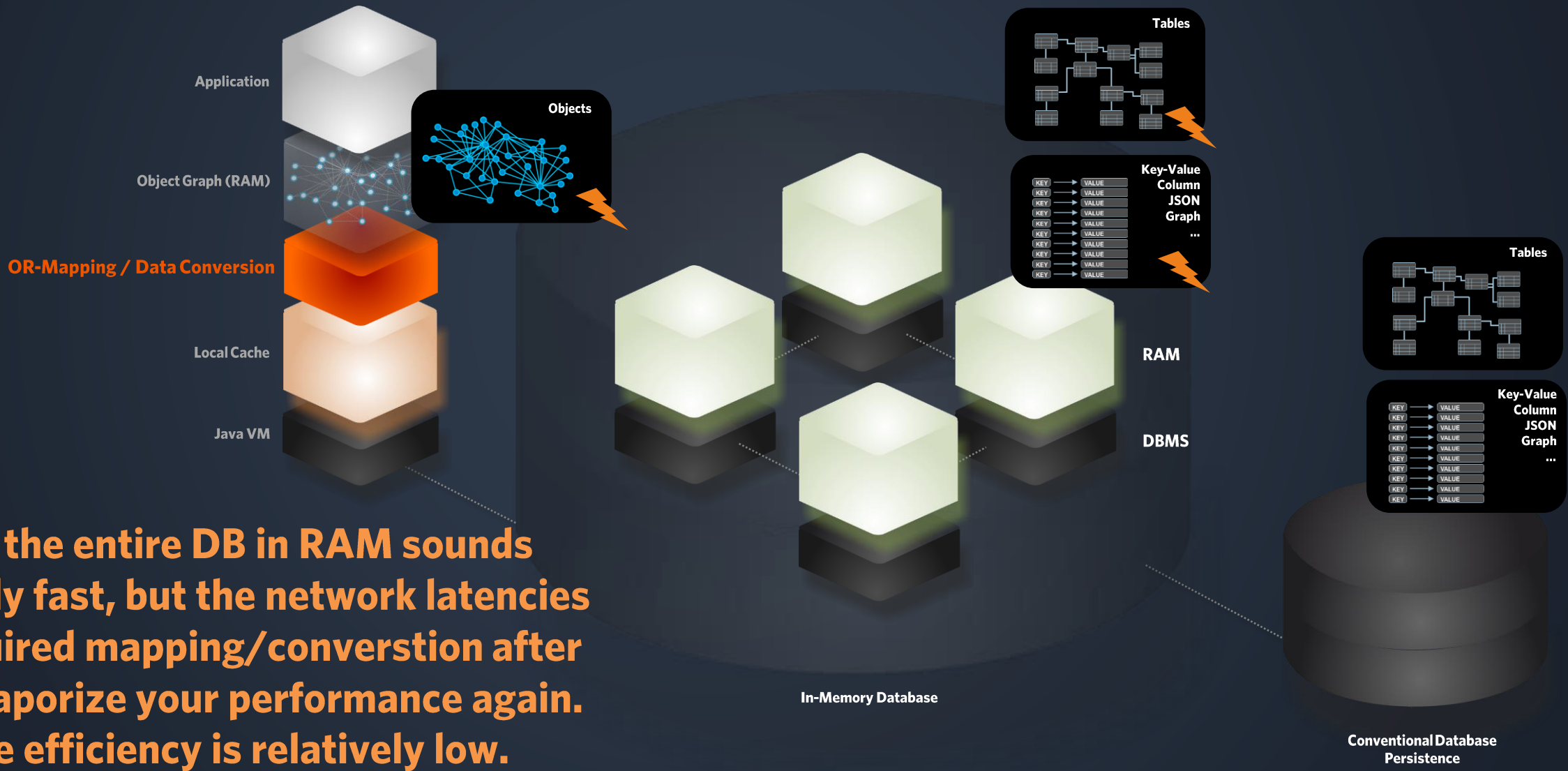
- **Requires lots of CPU power**
- **Reduces your performance**
- **Expensive latencies** (mapping + network)
- **Complex architecture**
- **Expensive development process**
- **Inefficient concept requires expensive cluster infrastructure**
- **Increase your costs of infrastructure**

Caching (Local Cache)



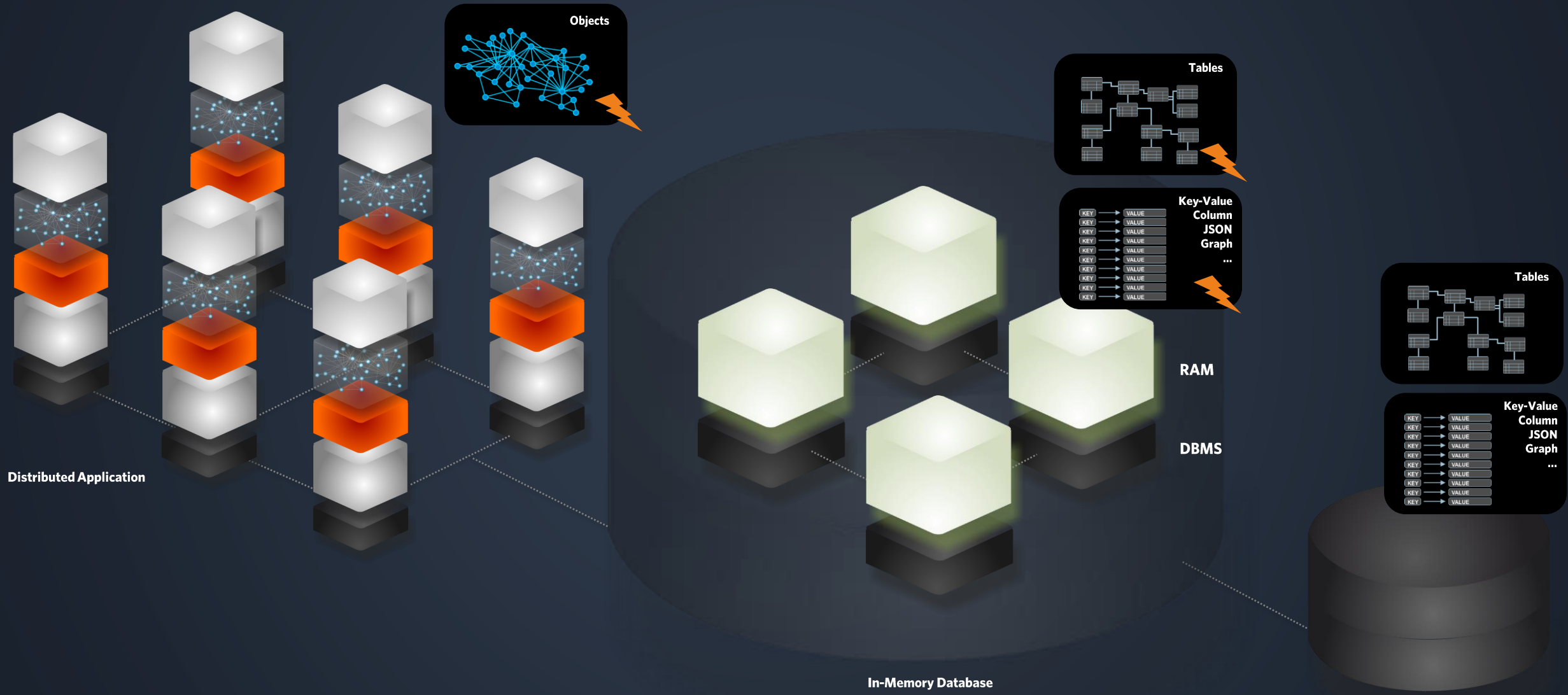
Even though reading data from a local cache, mapping/conversion is required.

In-Memory Database



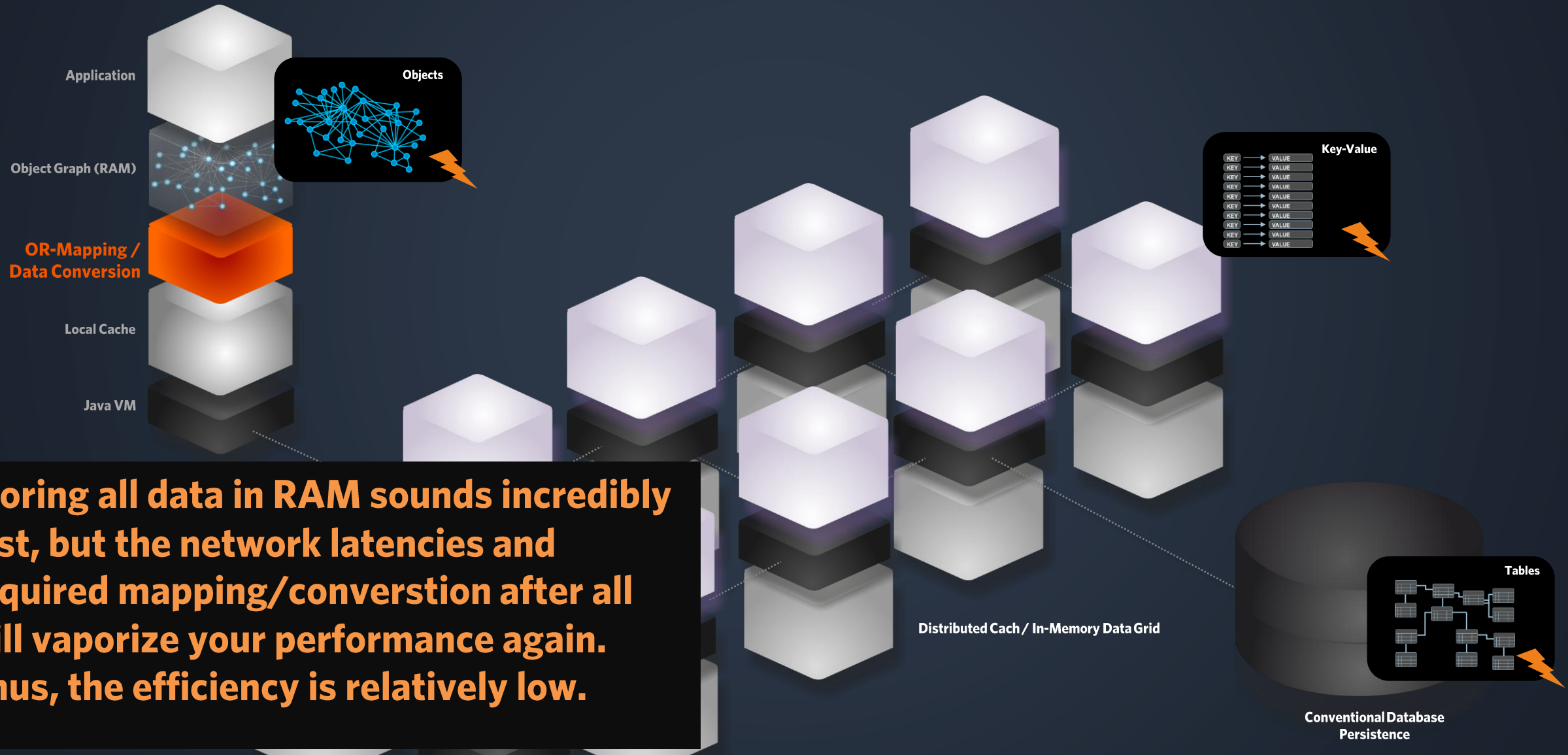
Running the entire DB in RAM sounds incredibly fast, but the network latencies and required mapping/converstion after all will vaporize your performance again. Thus, the efficiency is relatively low.

Distributed Applications with In-Memory Database



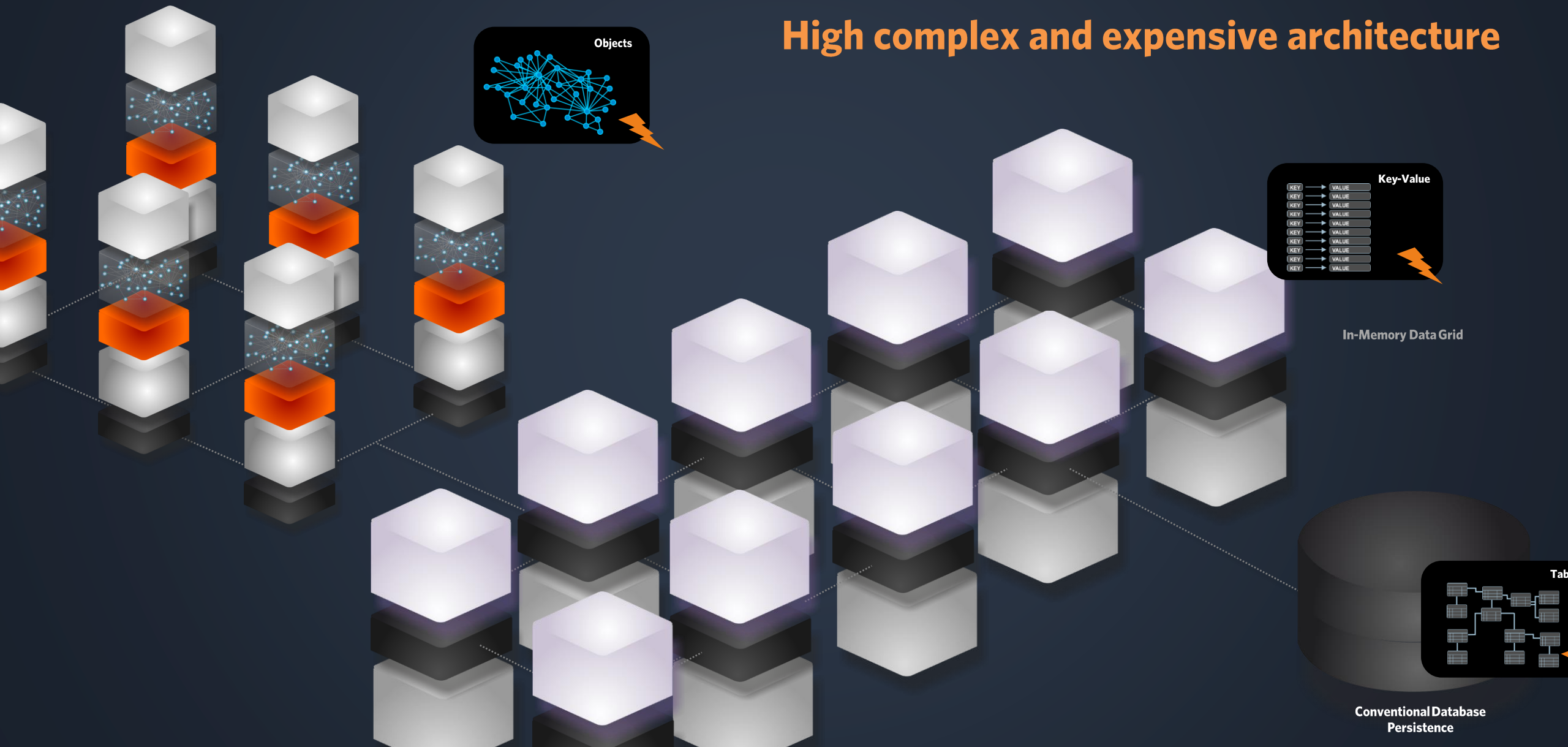
High complex and expensive architecture

Distributed Cache / In-Memory Data Grid (IMDG)

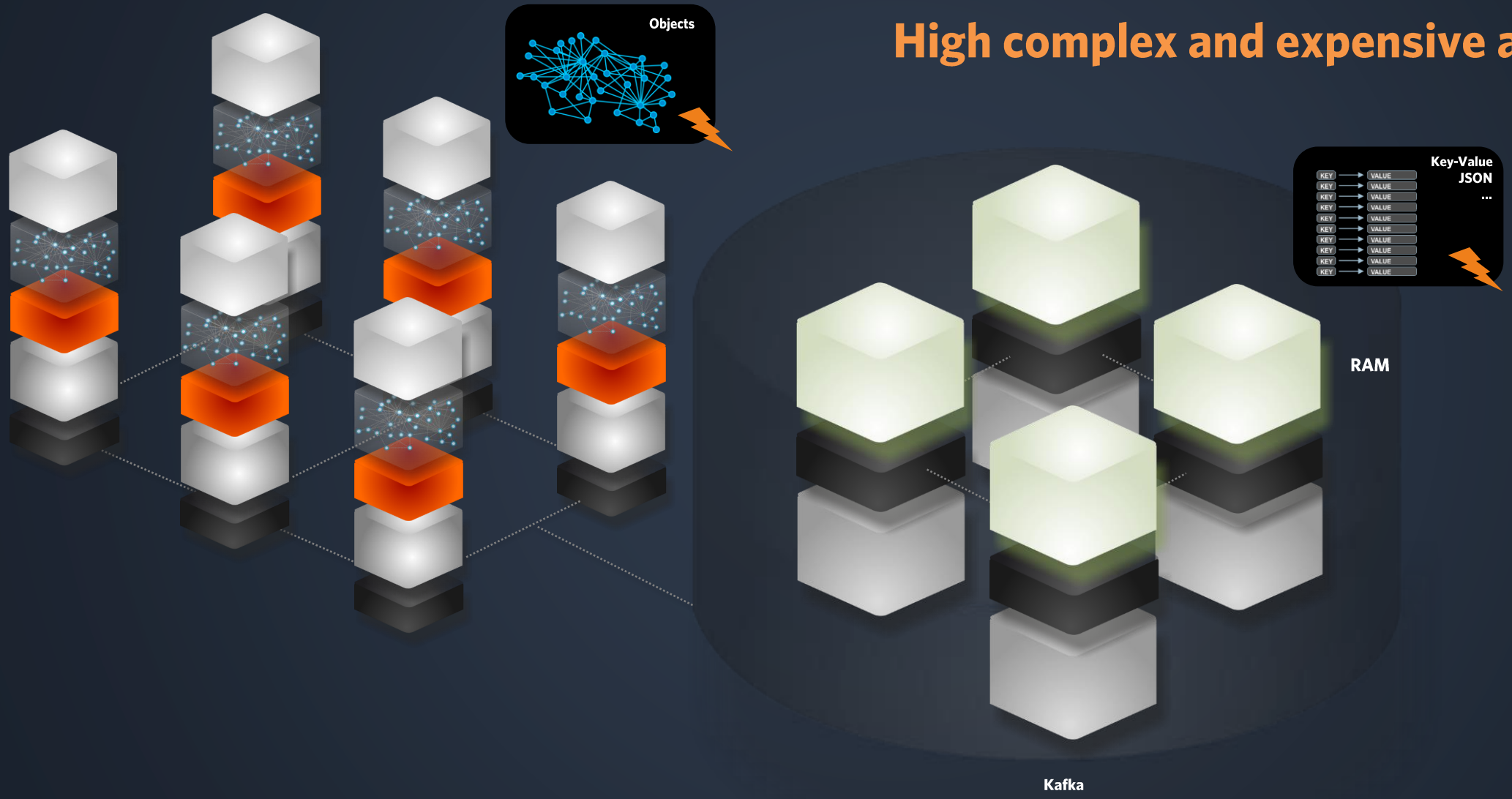


Storing all data in RAM sounds incredibly fast, but the network latencies and required mapping/converstion after all will vaporize your performance again. Thus, the efficiency is relatively low.

Distributed Application with Distributed Cache / IMDG

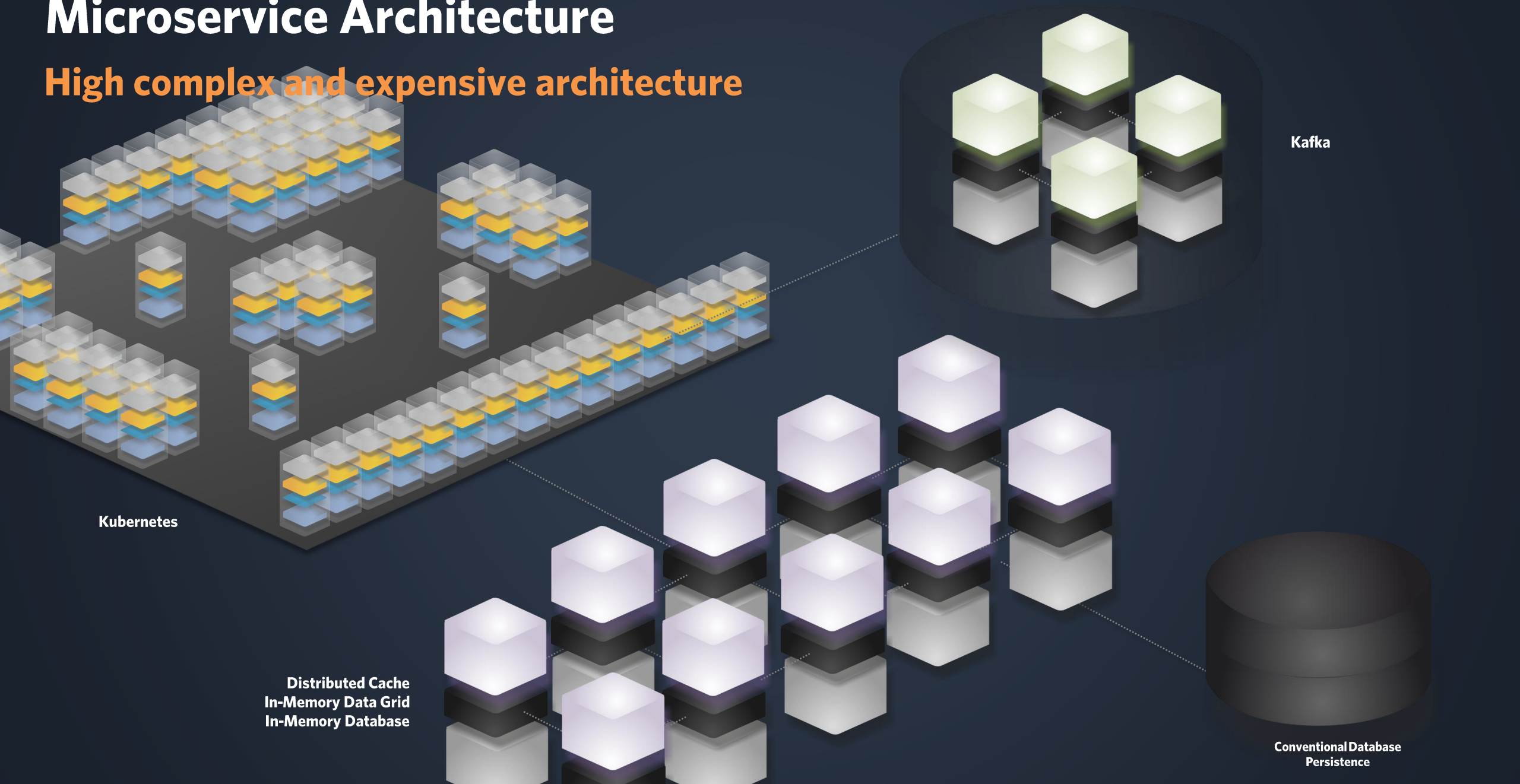


Distributed Application with Event Streaming (Kafka)



Microservice Architecture

High complex and expensive architecture



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

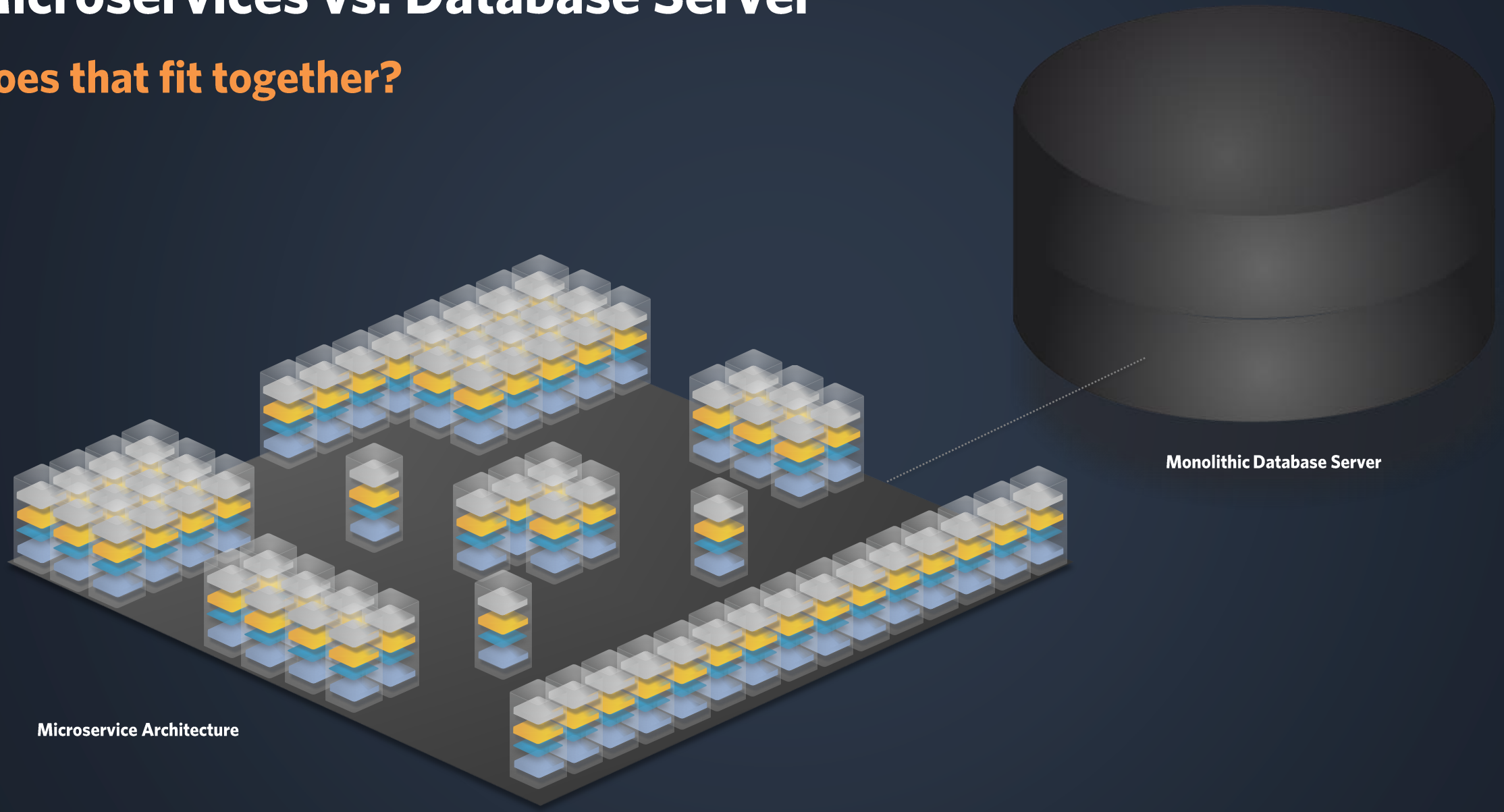
High Effort for Developers

- 2 data models (Java classes + DB data model)
- Data type mapping
- Complex ORM frameworks
- Additional caching Layers (local Cache, distributed cache, IMDG)
- Complex architecture
- Strong limitations (data model design)
- Mixing different paradigm, reduntantly and competing concepts
- Heavyweight dependencies
- Effortful testing and deployment process

Further Challanges

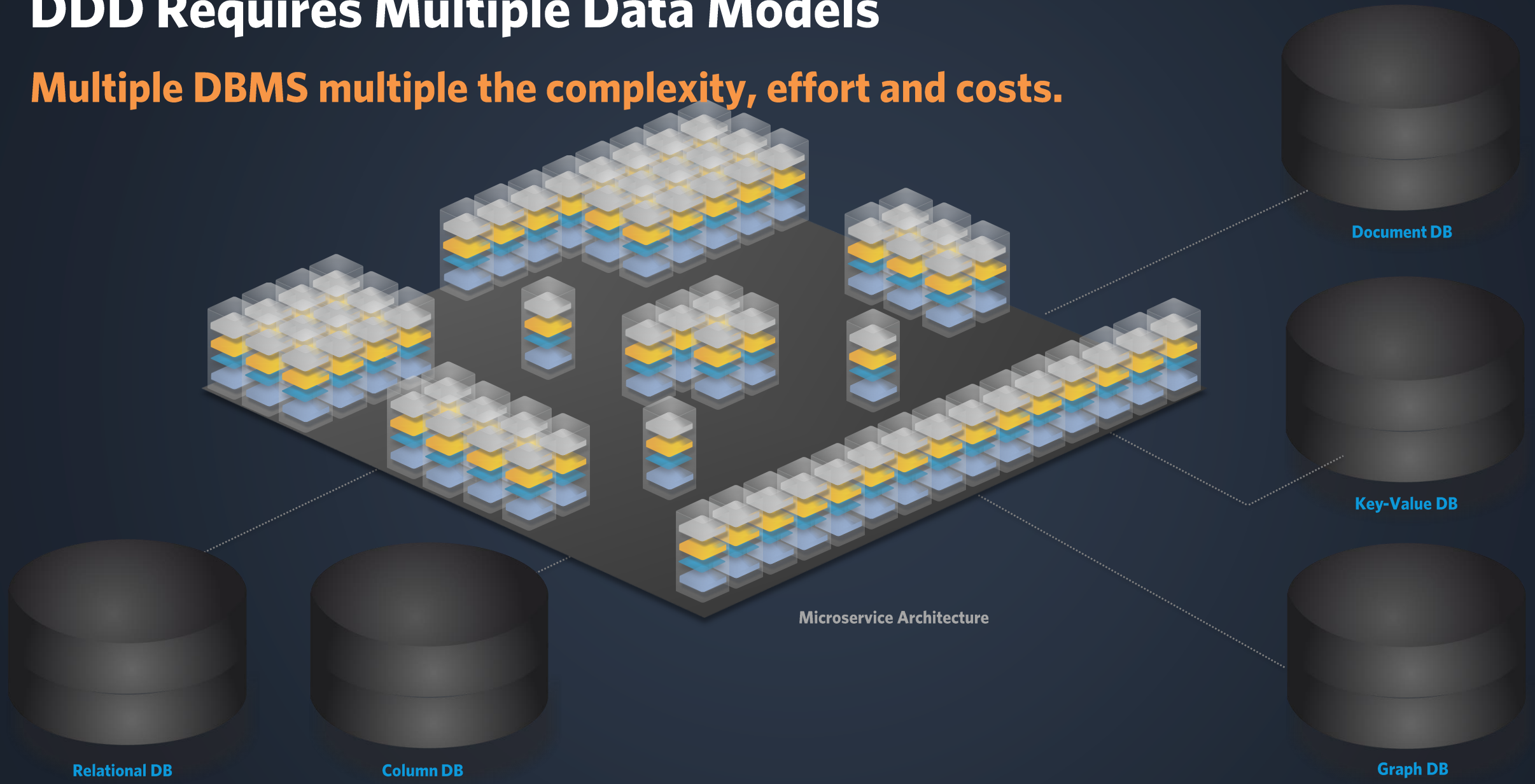
Microservices vs. Database Server

Does that fit together?

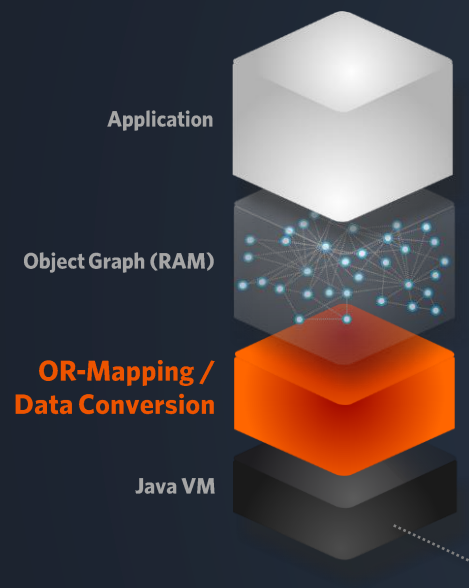


DDD Requires Multiple Data Models

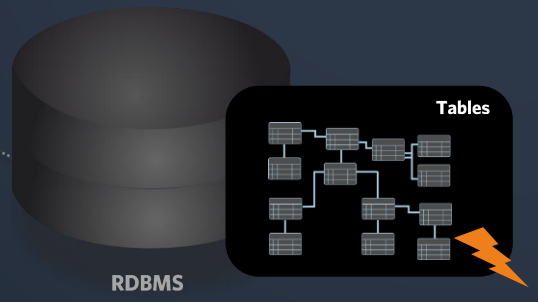
Multiple DBMS multiple the complexity, effort and costs.



Competing Concepts



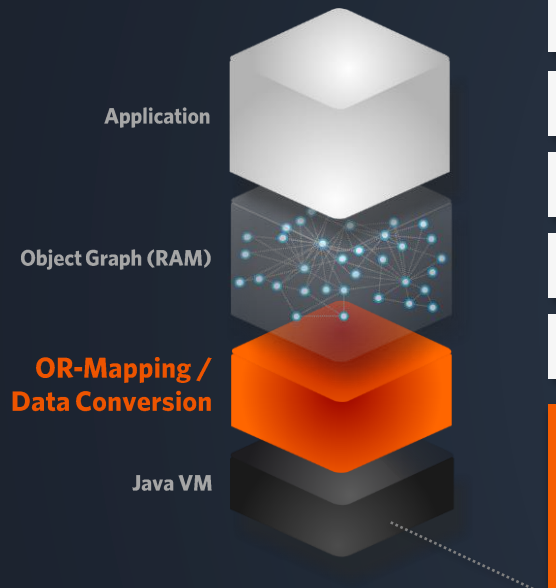
- Data structure (object graphs)
- Business logic (classes, objects, methods, etc.)
- Concurrency (sessions, connections, caching, etc.)
- User management
- IO (e.g. REST)



- Data structure (tables & relations)
- Business logic (trigger, SP, SF, Views, PL-SQL, etc.)
- Concurrency (sessions, connections, caching, etc.)
- User management
- IO
- Storage engine (write, read, caching, backup etc.)

Competing Concepts

In Java we already abstract the DB and ignore many native DB features.



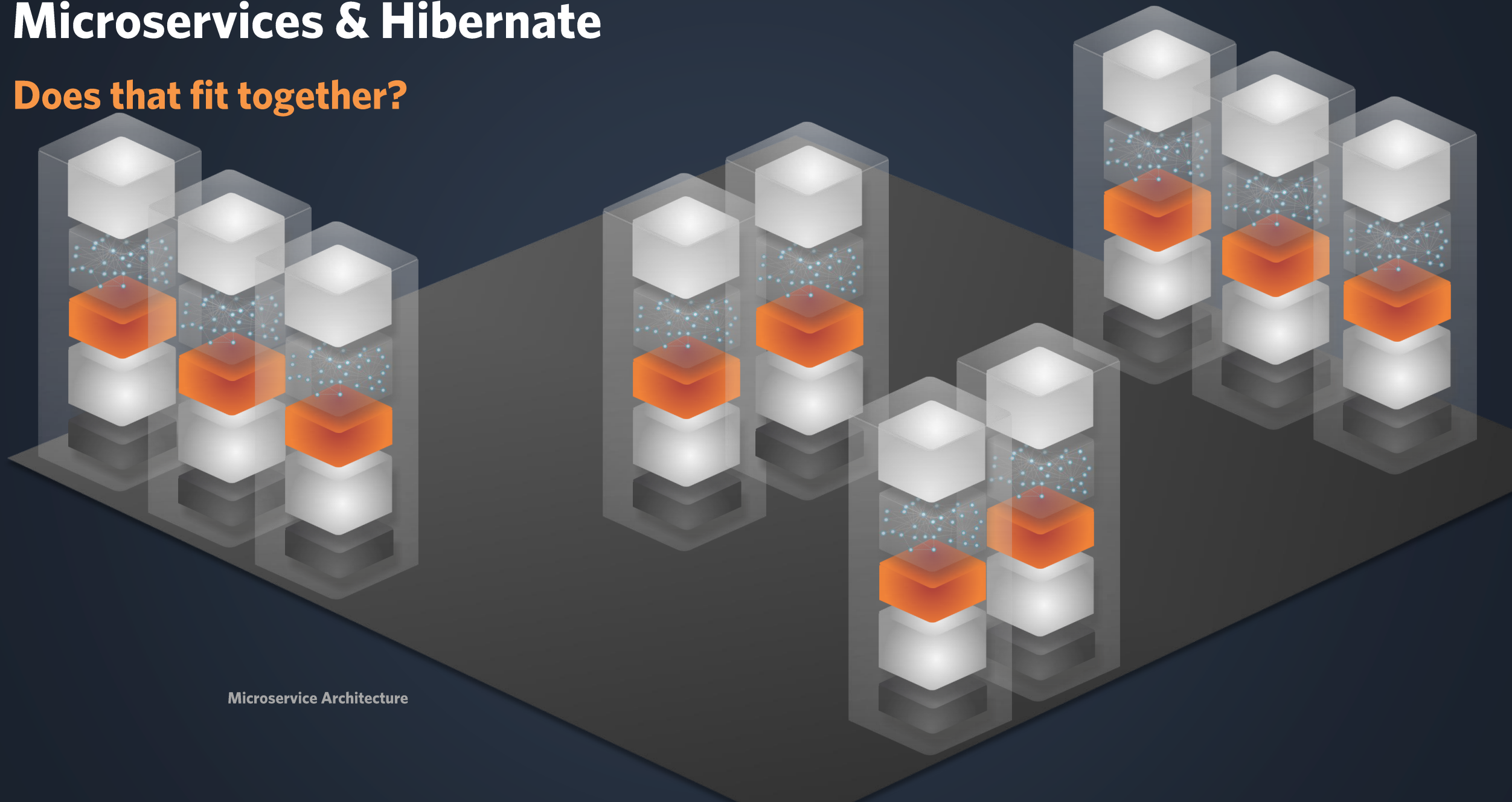
- Data structure (object graphs)
- Business logic (classes, objects, methods, etc.)
- Concurrency (sessions, connections, caching, etc.)
- User management
- IO (e.g. REST)

JPA

- Data structure (tables & relations)
- Business logic (trigger, SP, SF, Views, PL-SQL, etc.)
- Concurrency (sessions, connections, caching, etc.)
- User management
- IO
- Storage engine (write, read, caching, backup etc.)

Microservices & Hibernate

Does that fit together?

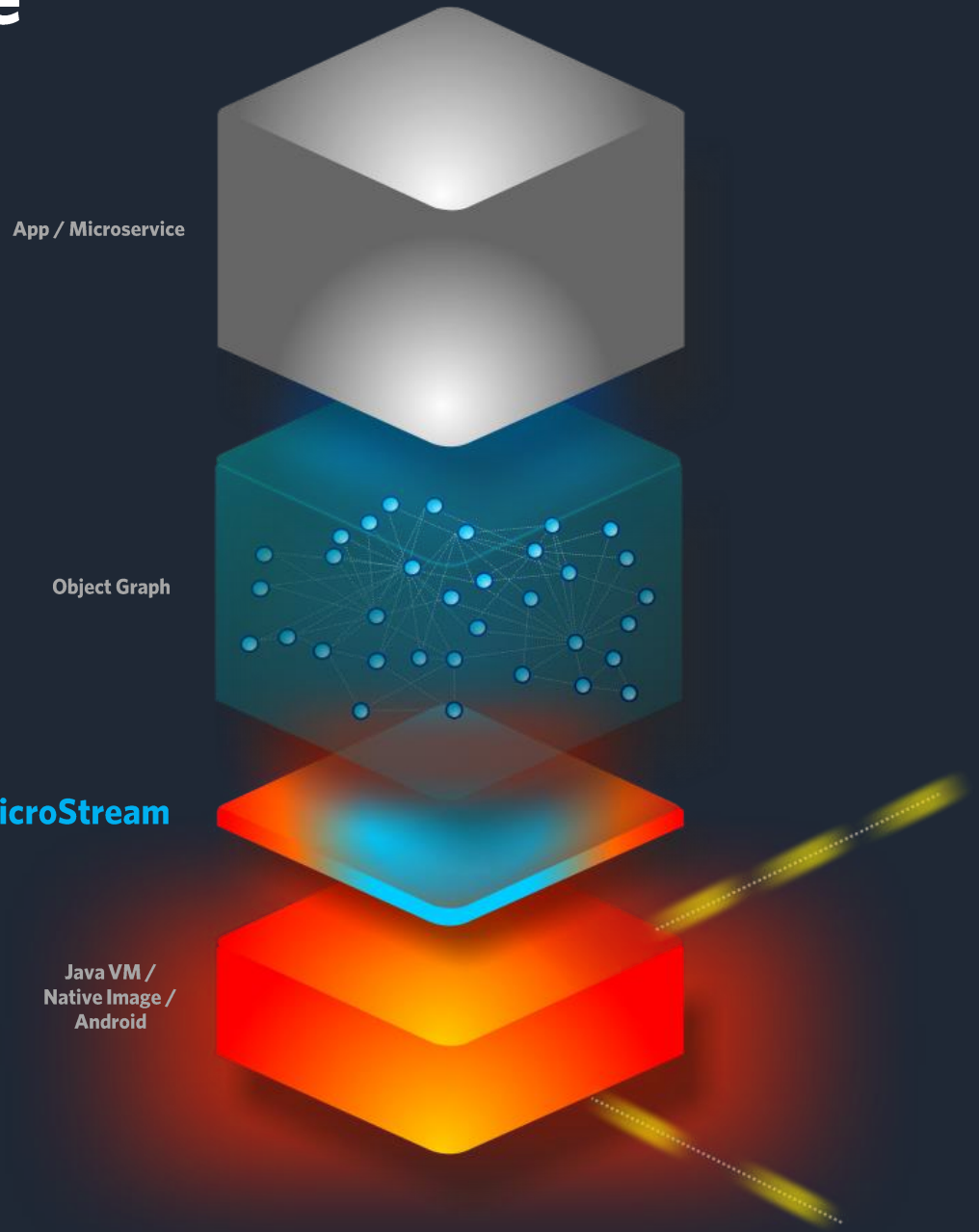


Microservice Architecture

Ultra-fast
Java In-Memory Data Processing

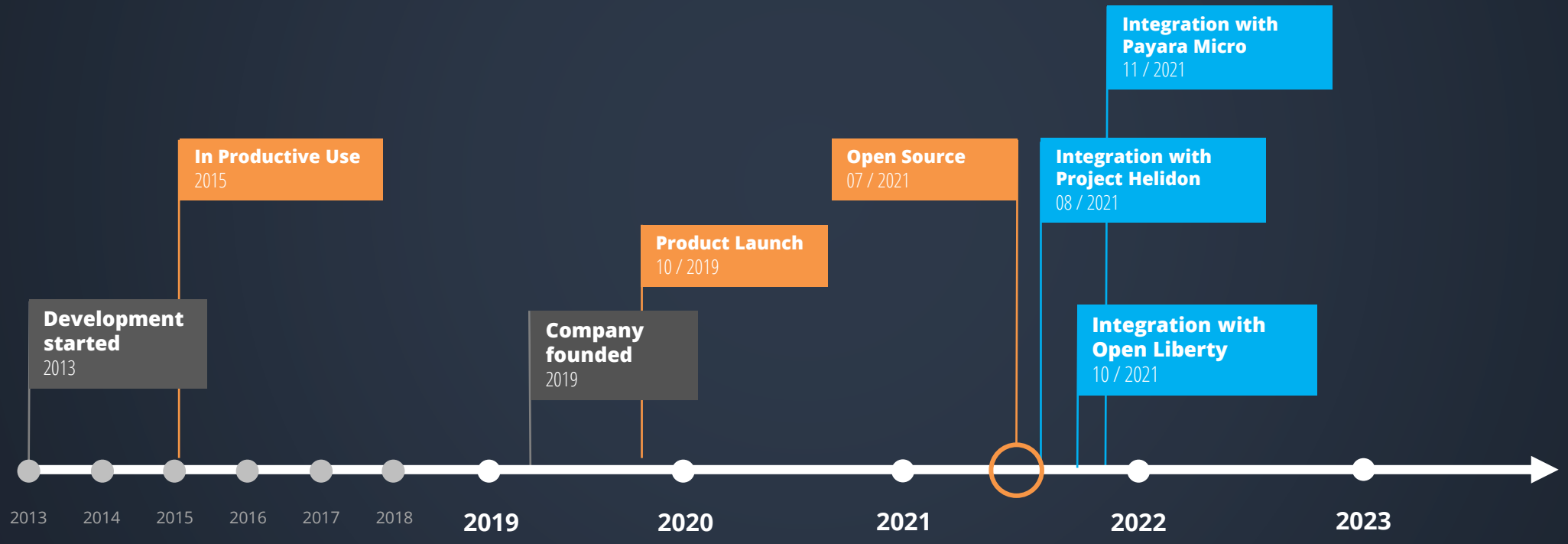
Java is Perfect for High-Performance In-Memory Data Processing

- **Object graph:** multi-model data structure that supports any Java type
- **Data model:** Java classes only – database-specific data models are not needed at all
- **Query language:** searching object graphs in-memory with Java Streams or GraphQL
- Incredible in-memory high-performance enables queries in microseconds
- Pure Java, fully object-oriented, typesafe, elegant programming model
- **MicroStream:** persisting any object graph into any storage solution

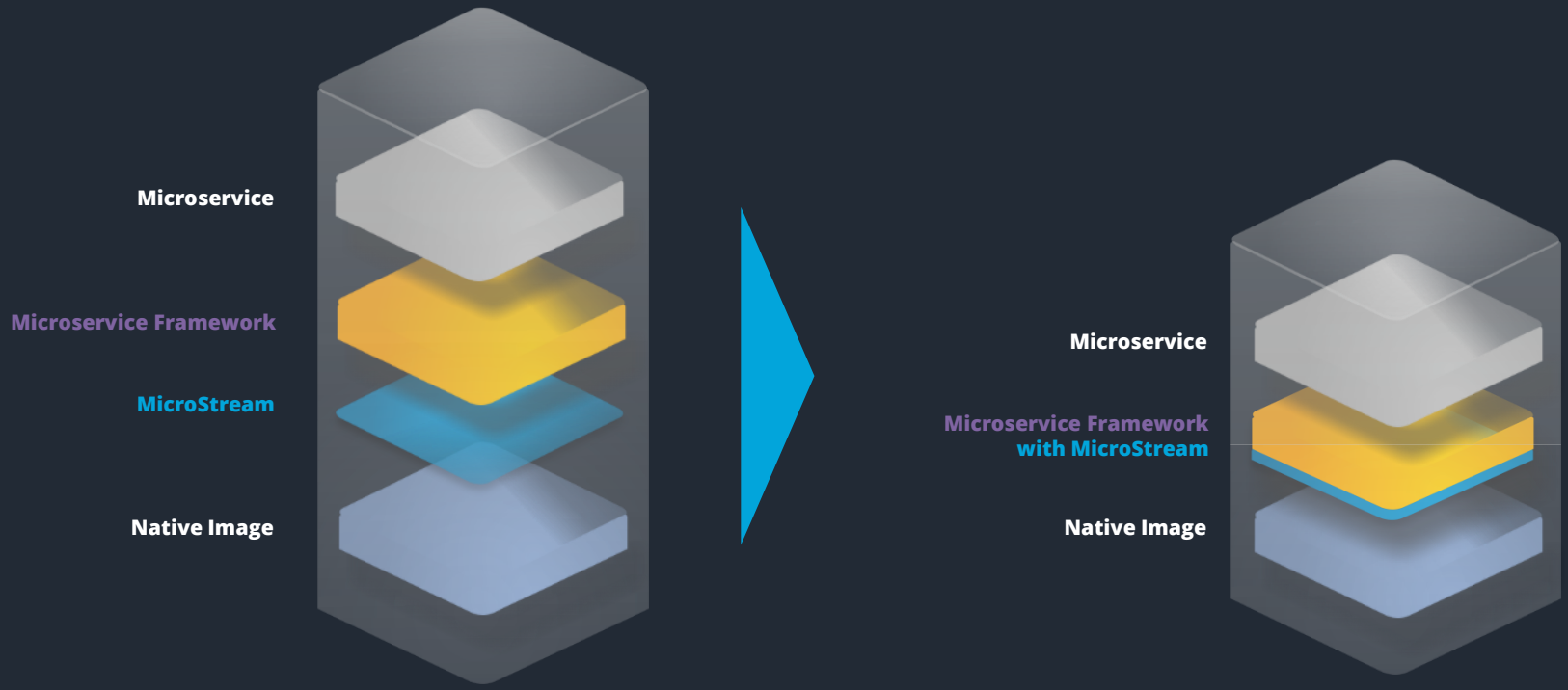


MicroStream Persistence

MicroStream History



MicroStream will be Integrated and Delivered with Microservices Frameworks



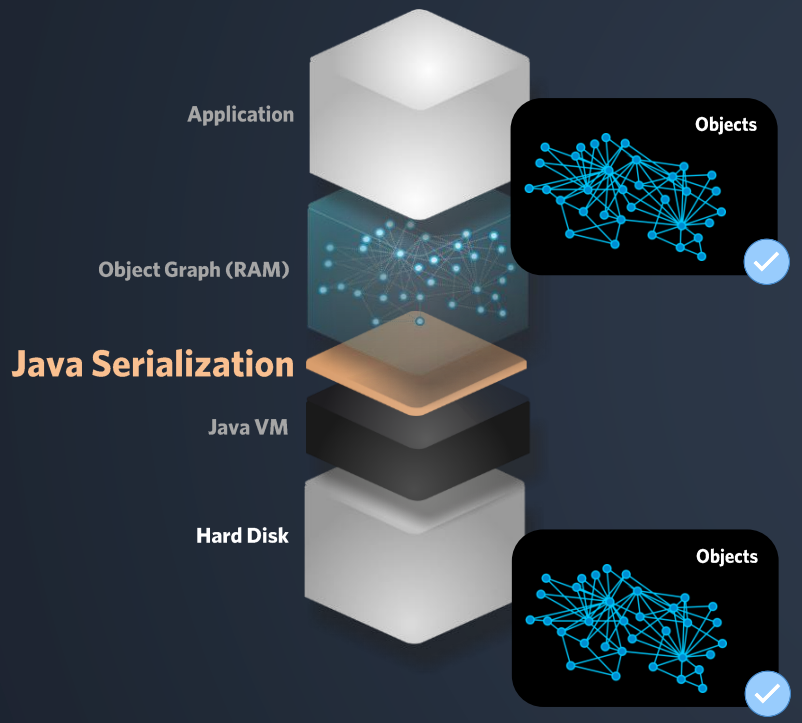
helidon.io

Open Liberty

Our Partners

Some of our Customers

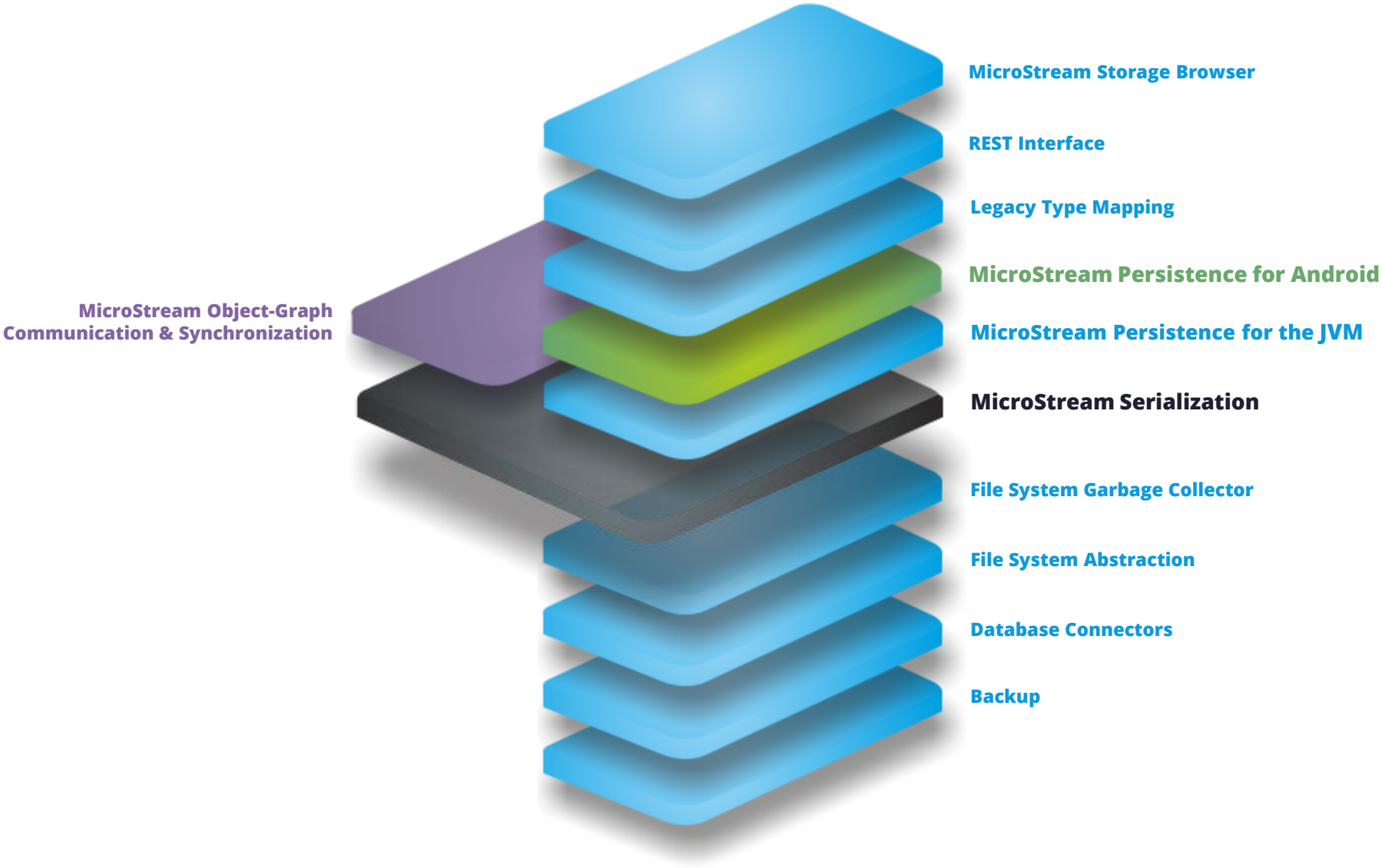
Old Java Developer's Dream



Loading only single objects or subgraphs on-demand and updating the object graph automatically, is not covered by the Java serialization. Additionally, Java serialization is limited and slow. Beyond serialization, there are numerous challenges in terms of persistence that are not covered by Java serialization.

MicroStream Makes the Old Java Developer's Dream Come True.

What is MicroStream ?

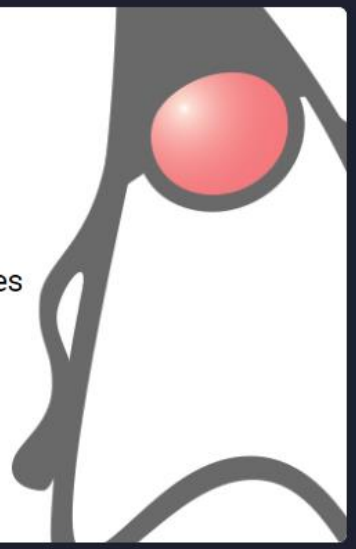


Platforms

Get MicroStream for

Java

MicroStream for Cloud-Native Microservices and Classic Java Enterprise Applications.

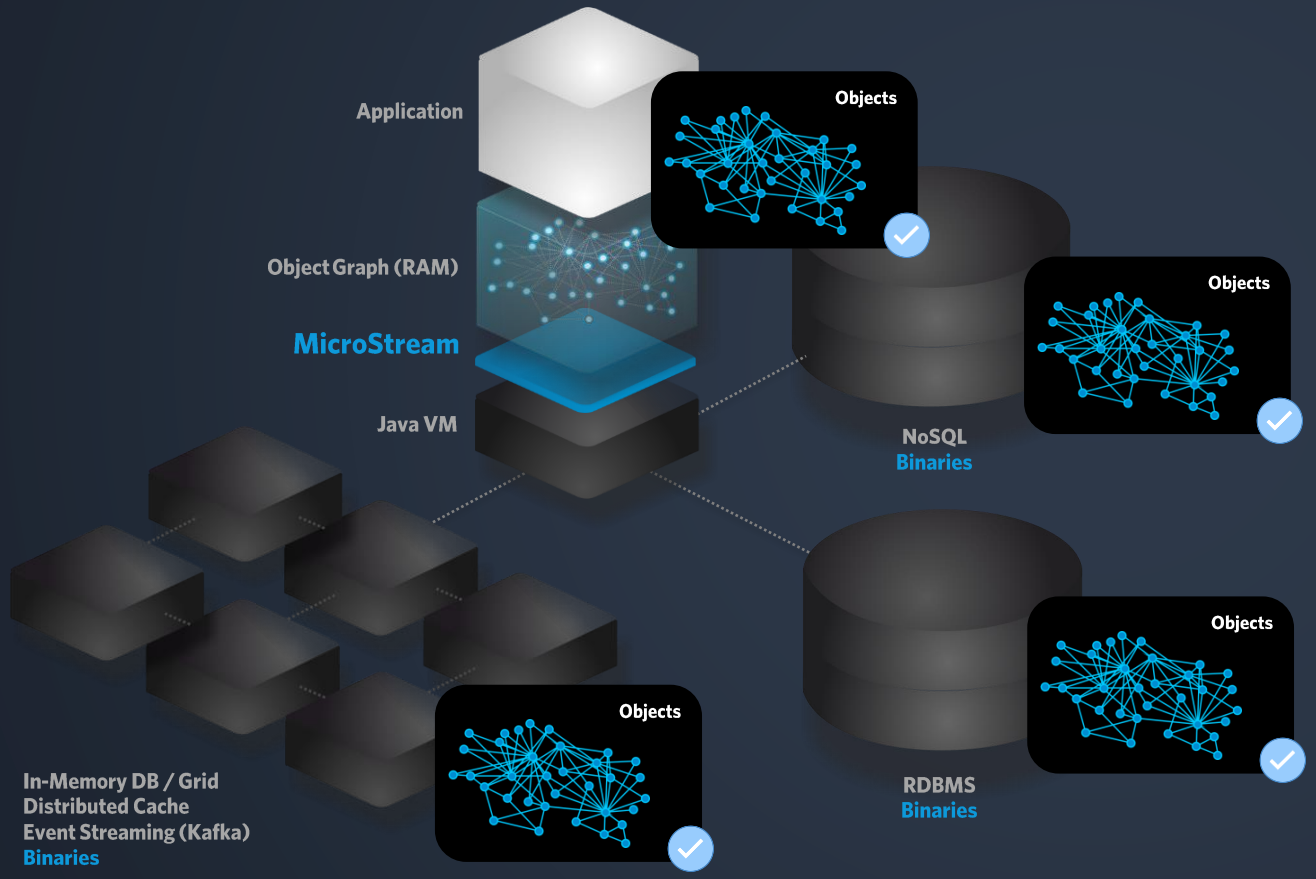


Get MicroStream for

Android

MicroStream for Mobile, Embedded and, Edge Devices.

MicroStream Persistence



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 up to 92% costs of infrastructure**

Supported Storages

RDBMS

NoSQL

Cloud Object Store

Accelerating Queries up to 1000x

Query: Revenue of the whole shop

JPA – Hibernate (Java Standard)		**MicroStream**		**Factor**
439.05 Milliseconds	Persistence: **Hibernate**	0.19 Milliseconds	Persistence: **MicroStream**	2260x
	Cache: **EHCache**		Cache: **-**	
2.28 Queries / Second	Database: **Oracle DB**	190.11 Queries / Second	Database: **Oracle NoSQL**	83x Queries / Second

0.19 ms

Live-Demo: **www.microstream.one**

Runs Wherever Java Runs

Desktops

On-Premise

Cloud

Container

Native Image

Microservices

Android

JDK 8+

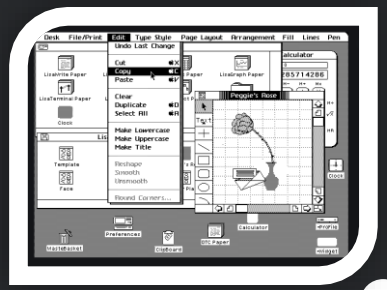
Use any JVM Technology


Your Benefits

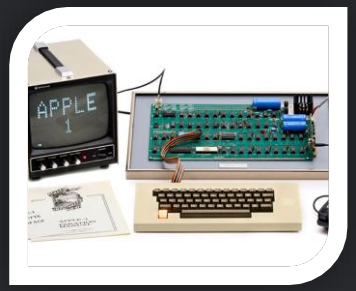
Performance Enables Revolutionary New Innovations, Features and Products

Today – AI, ML, IoT, Automotive

2007 – The Smartphone



1983 – The Graphical User Interace



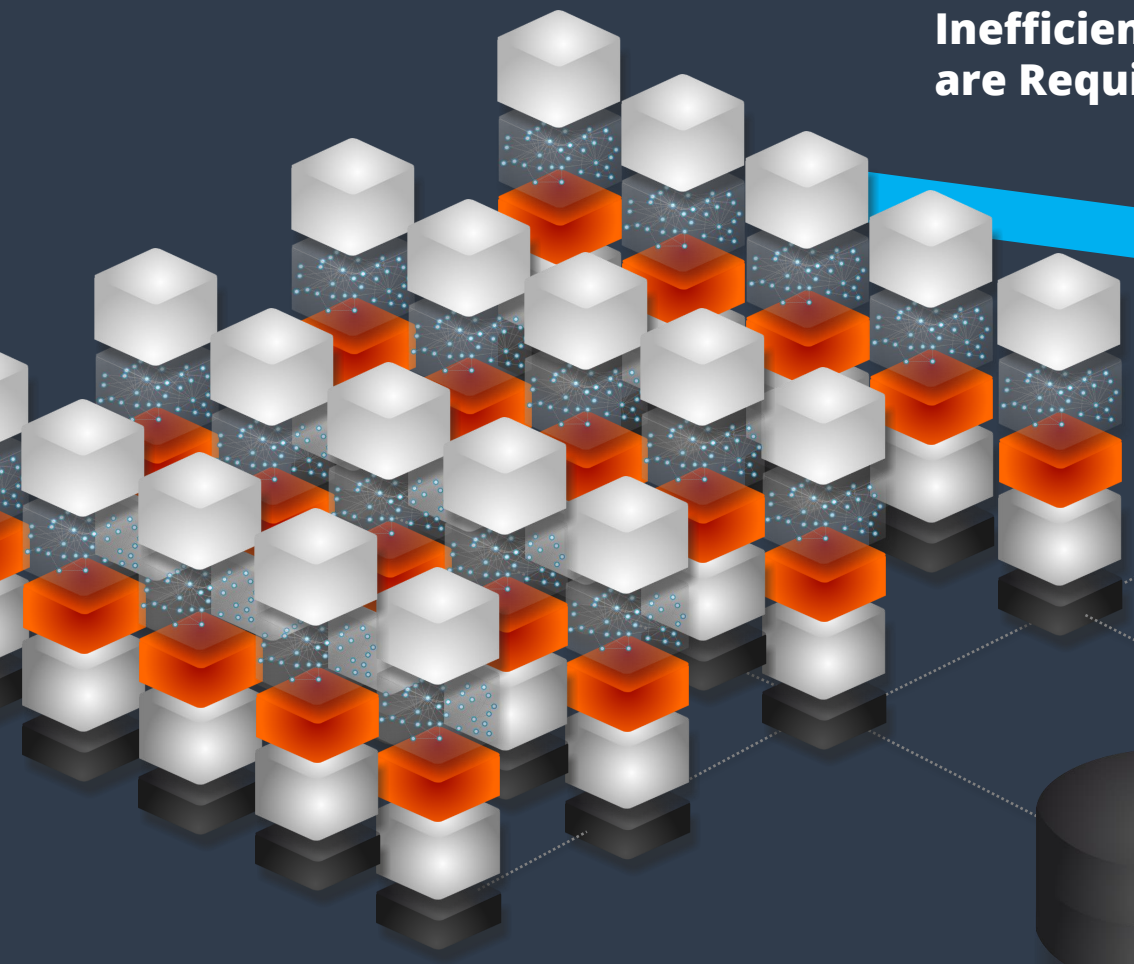
1976 – The Personal Computer

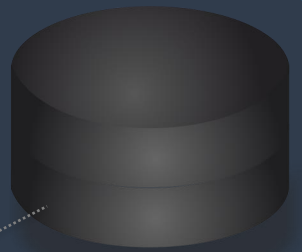
1960s – The Main Frame

MicroStream

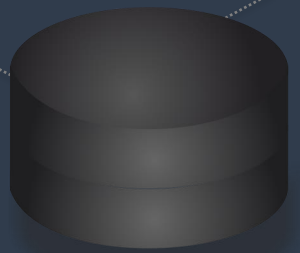
Save up to 90% Cloud Costs

Traditional Persistence Is Inefficient. Numerous of Nodes are Required.





NoSQL



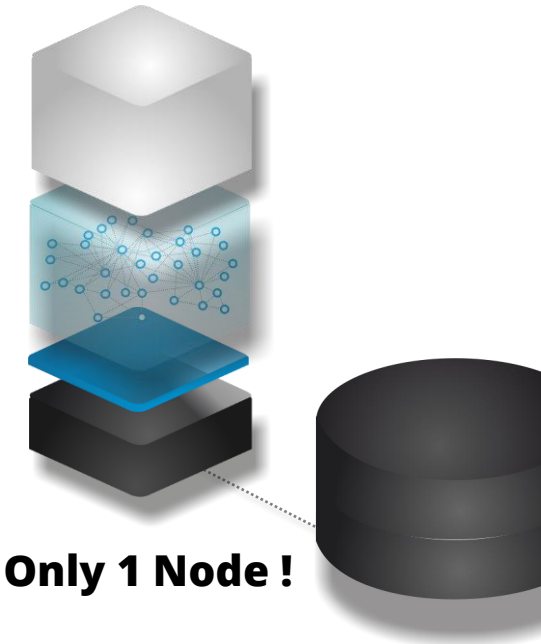
RDBMS

Conventional Cluster for Running a globally App

Today with MicroStream:

- 87.5 %

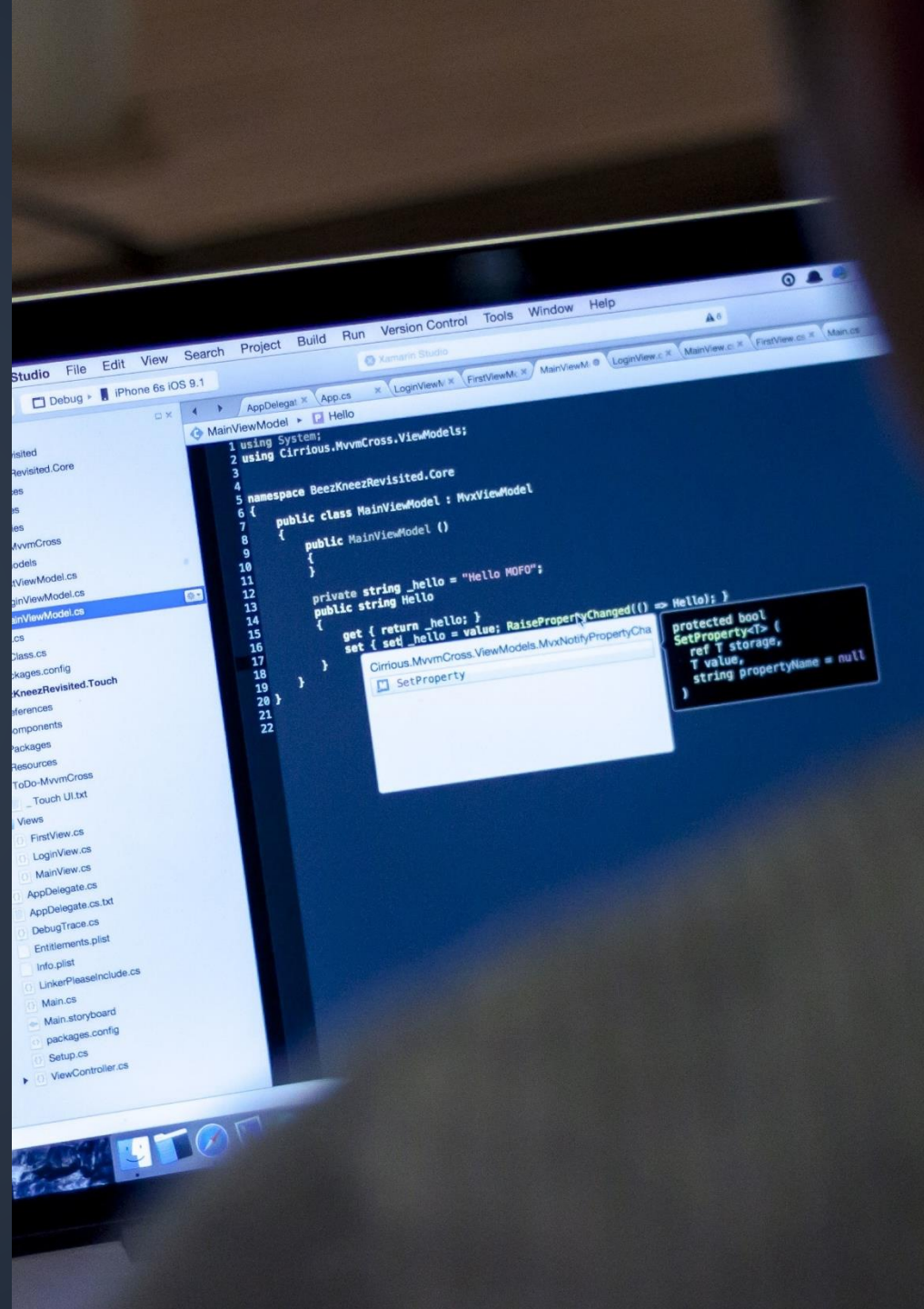
Costs of Infrastructure annually



Only 1 Node !

Simplifies your Development Process

- 1 data structure
- 1 data model - Java classes only
- No mapping, no impedance mismatch
- No JPA
- Query language: Java Streams API
- No local cache needed
- No dependencies, no special superclass or interfaces, no annotations, just POJOs
- Freely design of your Java object-model
- Core Java only



How Does MicroStream Work ?

Data Model: Just POJOs

```
public class Customer {

   private String firstname;
   private String lastname;
   private String email;
   private LocalDate dateOfBirth;
   private Boolean active;
   private Set<Order> orders;

   ...
}
```

Data model:
Java classes only

No dependencies,
just use POJOs

No need for special superclasses,
interfaces or annotations

Use existing classes as they
are, no strings attached

Any Java types
are supported

Use any types
from 3rd party APIs

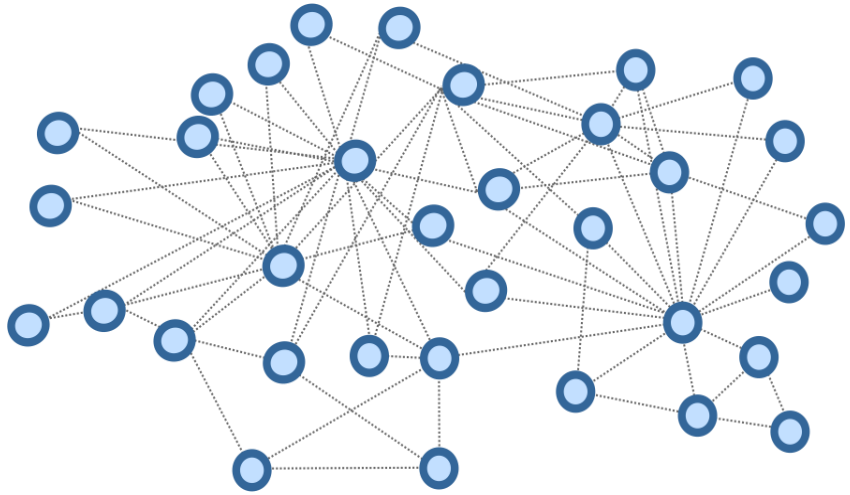
Design your object model
freely without any limitations

Using inheritance is
trouble-free

Migrating to MicroStream
is trouble-free

Design Your Object Graph Completely Freely

- Use any Java type
- Use collections
- Use object references
- Use circle references
- Use any object from 3rd party libraries



Persisting Objects

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

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

Store any single object or subgraph explicitly

Store any Java type, any suited type is supported

Using inheritance is trouble-free

Binary data format, no expensive mappings

Atomic operation and ACID transaction-safe

Strong consistency

Append-only log strategy

Multithreaded write ops for max performance

Gigantic data throughput

Custom-tailored type handling for best performance

Replaces 3 CRUD ops: Create, Update & Delete

Loading Objects 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);
  }

  ...
}
```

Sufficient RAM available:
Restore the entire object-graph

RAM limited: Load single objects or subgraphs on-demand

Loaded objects are merged into the object graph automatically

No inconvenient object copies

No more classic selects, simply call getter

Minimizing expensive IO ops

Multithreaded read ops for max performance

Gigantic data throughput

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());
           });
       });
}
```

Microsecond Query-Time with Java Streams API

Core Java instead of database query languages

Queries are executed in-memory

Simultaneously query execution with Parallel Streams

No network bottlenecks, no latency.

Type-safe, clean and great testable code

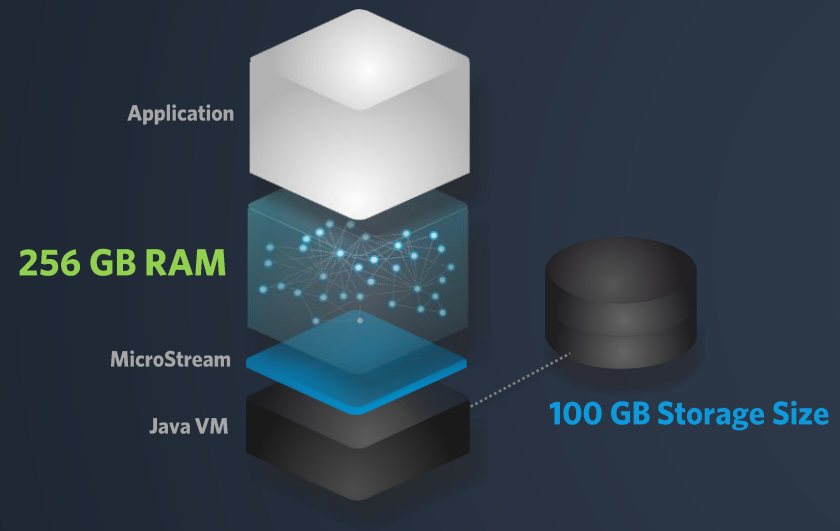
Minimizing expensive IO ops

Memory Management

- Memory is fully managed by the JVM
- Use lazy references if possible
- Clear your lazy references which are not used anymore
- In case of garbage collector issues, try OpenJ9 or Azul JVM

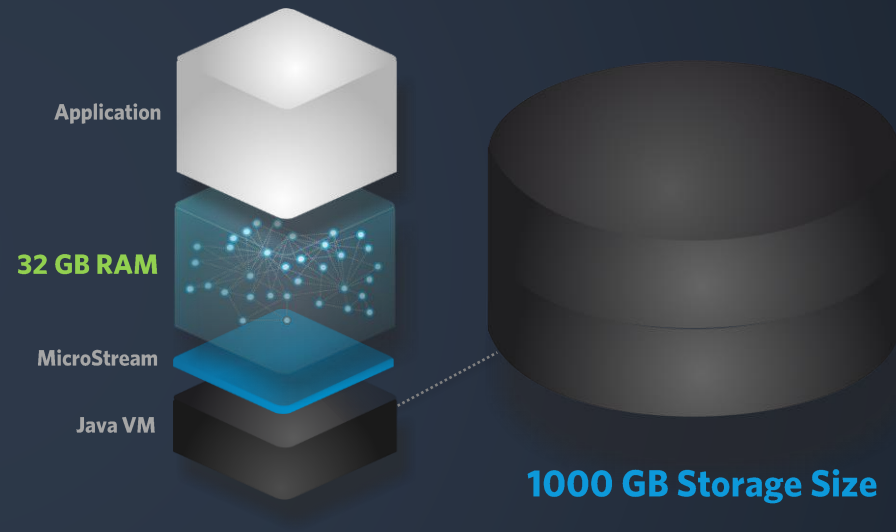
Full In-Memory vs. Lazy-Loading

Enough RAM available:



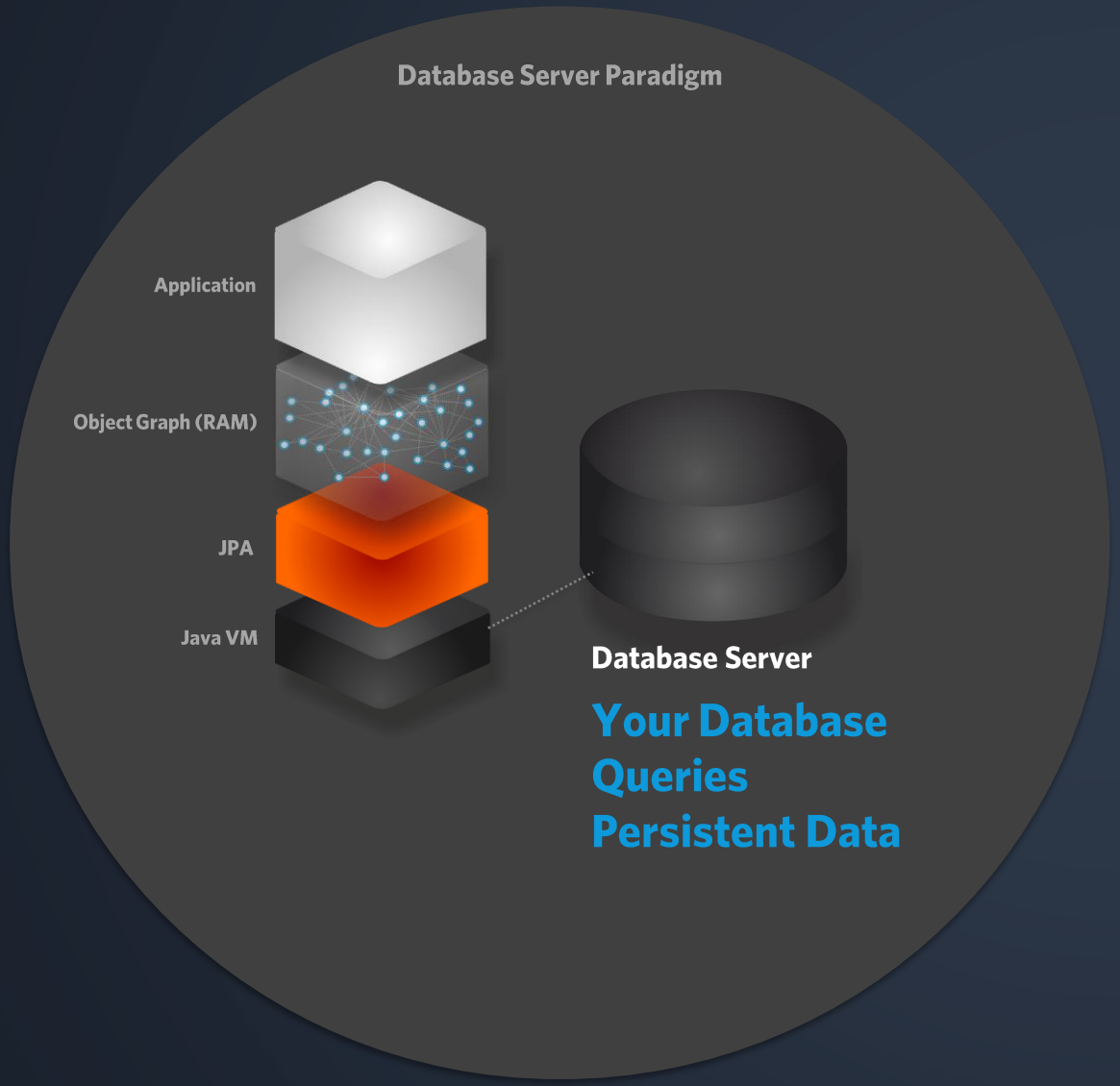
- You can load your whole DB into RAM
- Pure in-memory computing
- No latencies
- Super fast
- Lower startup time

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

Note: Your Object Graph is Your In-Memory Database



MicroStream Features

Tiny Java Library

MicroStream is a tiny Java library without any dependencies wihch you can download via Maven. It runs within your app's JVM process.

Data Model: Java Classes Only

Only 1 data model: Java classes. No more specific database model. No expensive mappings or data conversion. Design your model freely.

Multi-Model Data Structure

A Java object graph is by nature a multi-model data structure. You can add any object, lists and other collections, key-value pairs as well as any document.

No Annoying Restrictions

No need for special superclasses, interfaces such as Serializable, annotations or any other internal configurations. Just use POJOs.

Store Any Java Type

Any meaningful Java types can be persisted. Storing any types from 3rd party APIs is trouble-free.

Dynamic Store Ops

Store any single object, any subgraph, or the complete object graph by calling only one store method. In any case, only the delta will be stored.

ACID Transaction-Safty

Any meaningful Java types can be persisted. Storing any types from 3rd party APIs is trouble-free.

Append-Only Log

Each store operation adds the objects appended to your storage by using a binary data format for best performance.

Lazy-Loading

Each store is an atomic operation, ACID transaction-safe, and strong consistent.

No Object Copies

Loaded objects are fully automated merged into your object graph. You don't have to deal with inconvenient object copies and persistent contexts.

Queries: Streams & GraphQL

The Java Streams API enables you to search even huge and complex object graphs in memory in microsecond query time.

No Classic Selects, Just Getter

Loaded objects are fully automated merged into your object graph. You don't have to deal with inconvenient object copies and persistent contexts.

Memory Management

With MicroStream, RAM is still fully managed by the JVM, but you can remove lazy-loaded references at any time to free up RAM.

Multithreaded IO Ops

By using channels, IO operations will be executed multithreaded which increases the performance of your application.

Class Change Handling

Different versions of your classes are handled automatically through the runtime. No refactorings required.

Storage Garbage Collector

Legacy and corrupt objects in the storage are removed by the MicroStream garbage collector automatically through the runtime.

REST Interface

MicroStream provides you a REST API that enables remote access to your persistent storage data.

Storage Viewer

MicroStream comes with a web interface that allows you to browse through your persistent storage data.

Backup

Reliable and fully individual configurable data backup processes. Alternatively, you can use the backup function of your database.

Simple Migration

Both, migrating the data to or away from MicroStream is simple by using CSV import/export.

Runs Wherever Java Runs

MicroStream runs on desktops, on the server, in containers, in the cloud, on mobile & edge devices, as a native image & is pferfect for microservices.

Use Powerful Features From the Java Ecosystem

Fulltext Search

Apache Lucene is a powerful search engine for Java. Lucene allows you to add such as full-text search to your MicroStream app.

Manage Big RAM Sizes

Eclipse OpenJ9 is a very powerful open source JVM optimized for big RAM sizes and providing 63% less memory footprint.

Manage Terabyte RAM Sizes

Azul's JVM Platform Prime minimizes garbage collection pause time and enables your Java app to handle sdfs d Terabyte RAM size trouble-free. dg dg f

MicroStream Serialization

Mark Reinhold
Chief Architect of the Java Platform

”Java Serialization was a horrible Mistake.”

Serialization was a horrible mistake.
Half of all Java vulnerabilities are linked to serialization.

Mark Reinhold
Chief Architect of the Java Platform at Oracle

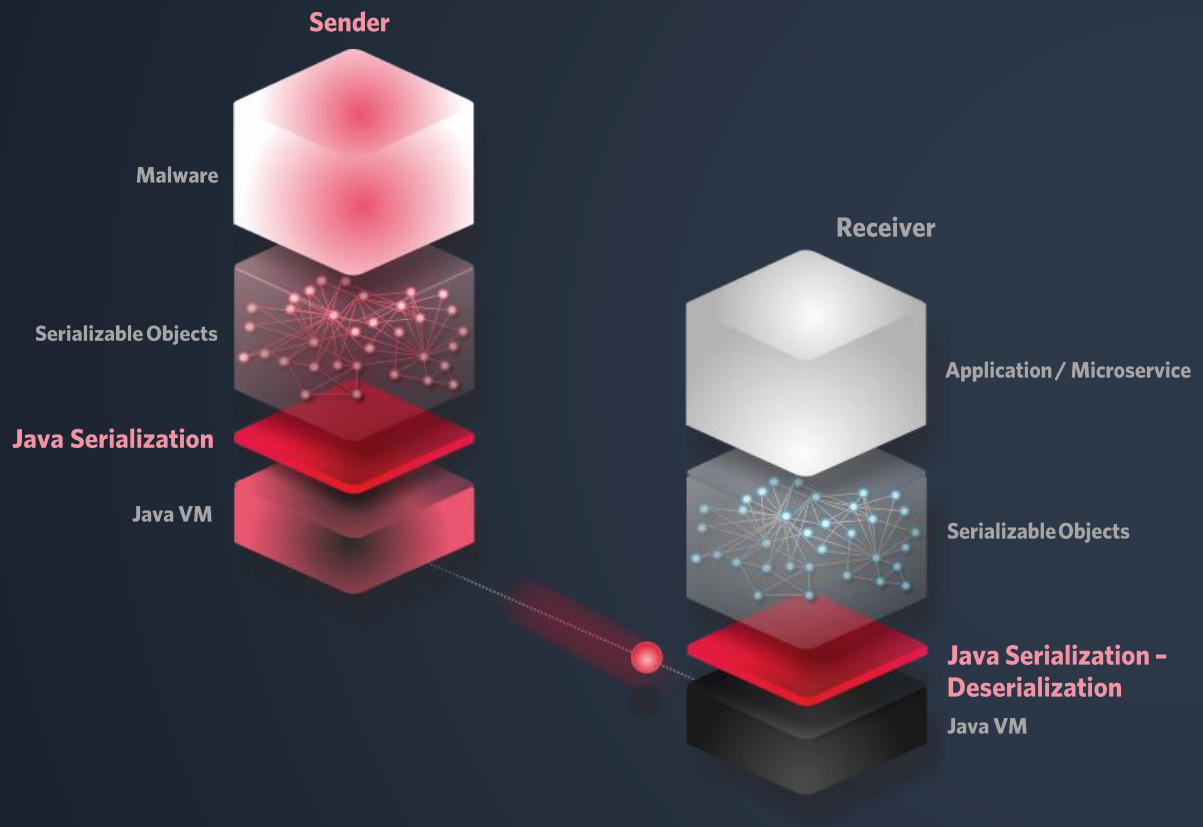
Java's serialization makes nearly every mistake imaginable and, poses an ongoing tax for library maintainers, language developers, and users.

Brian Goetz
Architect of the Java Language at Oracle

Other encoding (JSON, XML, Protocol Buffers, etc.) **is obscure and inefficient. Switching to another encoding doesn't solve the main problem of serialization.**

Brian Goetz
Architect of the Java Language at Oracle

Java Serialization



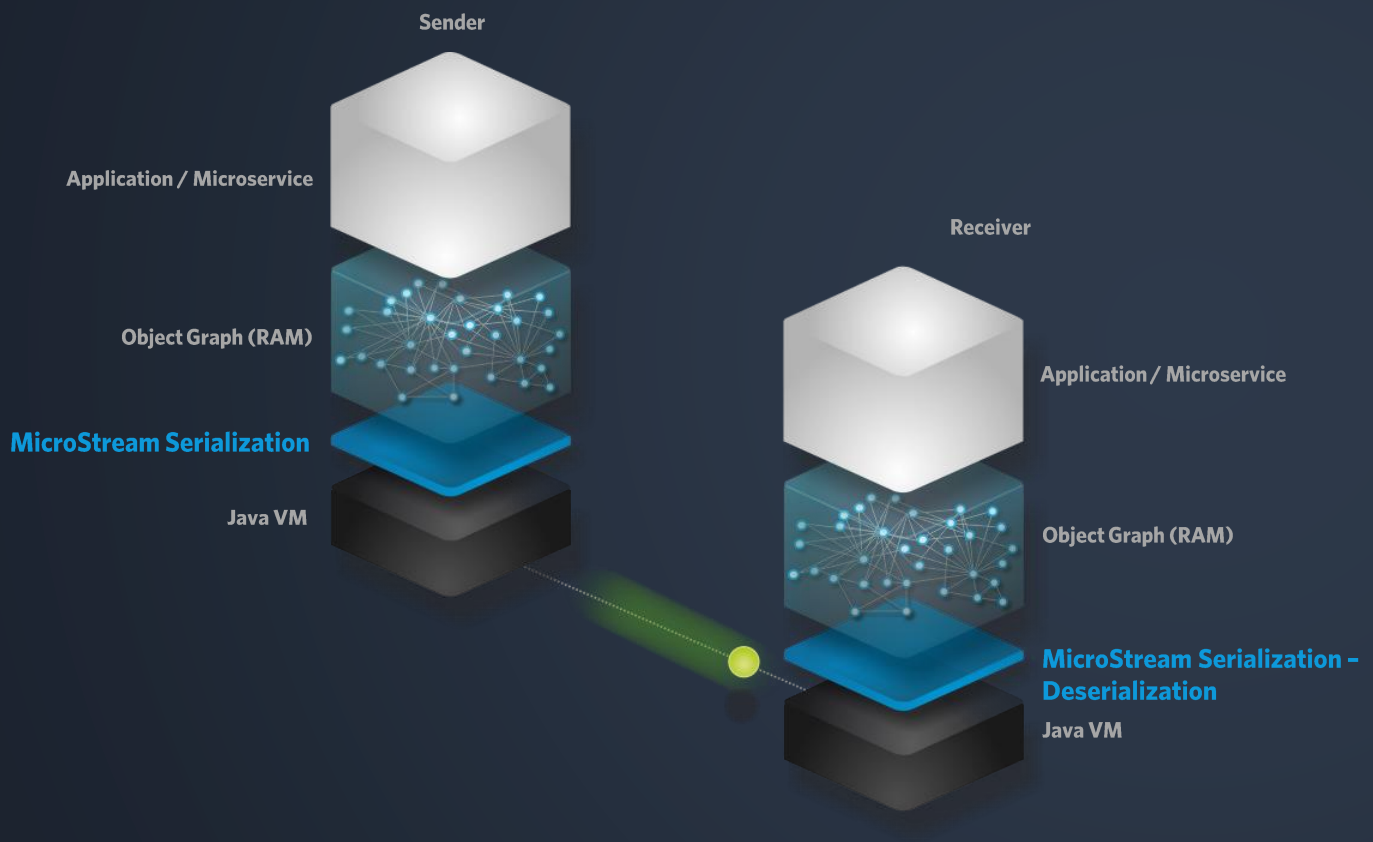
High-Security Risk

- Class information are transferred to the receiver
- All serializable classes in the classpath are executed automatically through deserialization
- Creating and injecting malicious code is scarily easy
- Most of your dependencies use serialization
- Using simplistic black- and white-list techniques are insufficient.

Limitations

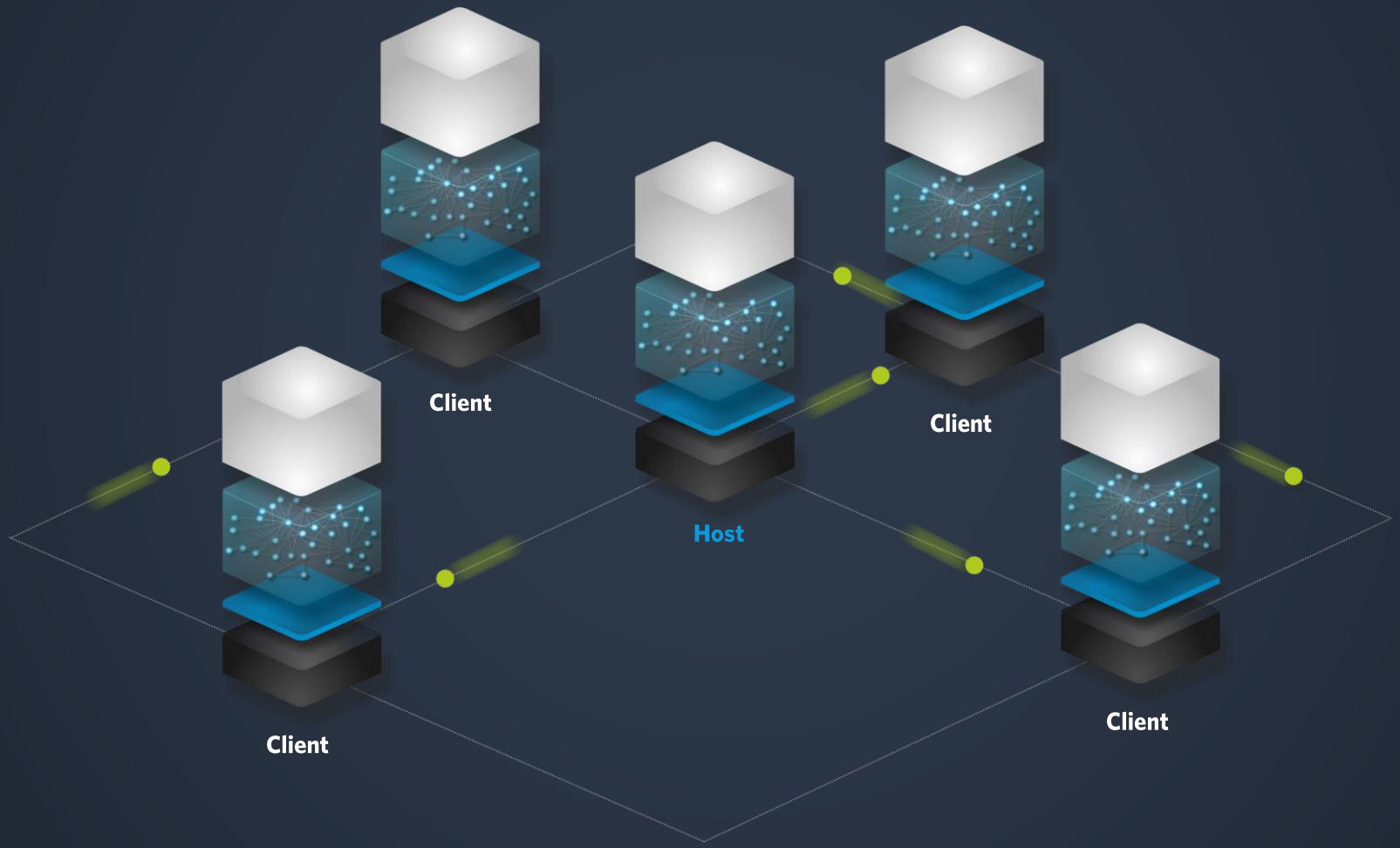
- Classes must implement the interface **java.io.Serializable**
- Objects from 3rd party APIs that haven't implemented **Serializable** can't be serialized
- After deserialization you get an object copy in any case
- Keeping your object graph synchronous is not possible
- Java serialization is slow

MicroStream Serialization



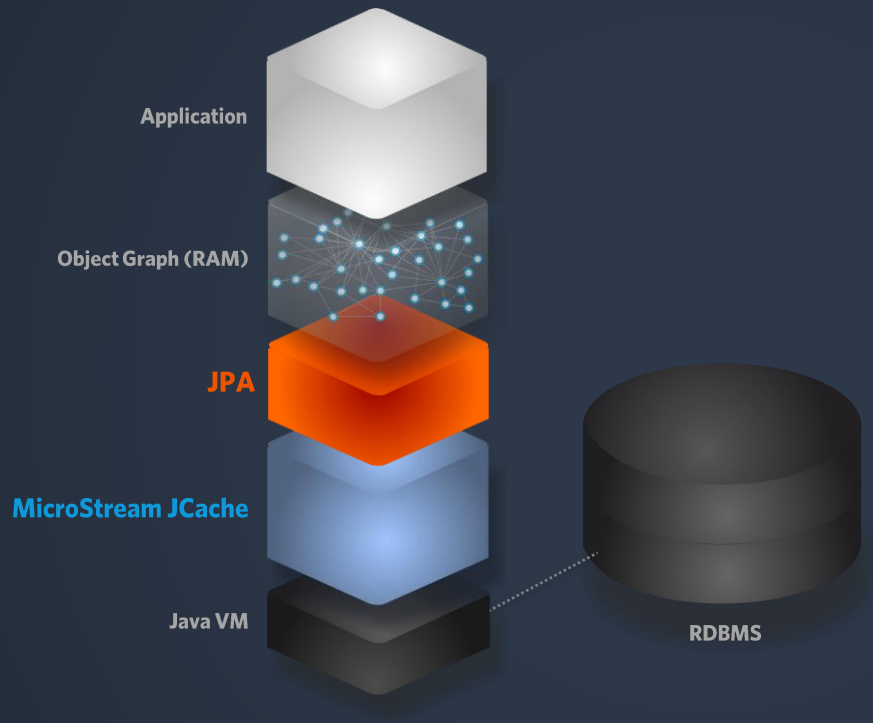
- Separation of data and metadata
- No code is executed at deserialization
- Injecting malicious code is impossible
- Bigest security leak of Java eliminated
- Supports object graph synchronization
- Migrating to MicroStream is easy

MicroStream Object Graph Synchronization



MicroStream JCache

MicroStream JCache



MicroStream is JCache-compatible and can be used as a local cache for your JPA application.

Get Started with MicroStream

Download: www.microstream.one

Docu: https://manual.docs.microstream.one/data-store/getting-started

Videos on YouTube: https://www.youtube.com/c/MicroStream/videos

www.youtube.com/c/MicroStream/videos

MicroStream
102 Abonnenten

ABONNIERT

ÜBERSICHT | VIDEOS | PLAYLISTS | KANÄLE | DISKUSSION | KANALINFO

Uploads ALLE WIEDERGEBEN

SORTIEREN NACH

$7,500 and GraalVM Award Winner of the MicroStream...

36 Aufrufe • vor 1 Monat

$3,000 and 2nd Rank Winner of the MicroStream...

37 Aufrufe • vor 1 Monat

$2,000 and 3rd Rank Winner of the MicroStream...

41 Aufrufe • vor 2 Monaten

$2,500 and Helidon Award Winner of the MicroStream...

54 Aufrufe • vor 2 Monaten

MicroStream Deep Dive

68 Aufrufe • vor 3 Monaten

Heldion with Project Loom

283 Aufrufe • vor 3 Monaten

DevSecOps - Low Hanging Fruits (German)

64 Aufrufe • vor 4 Monaten

MicroStream Hackathon Weekly Q&A | Edition 10...

19 Aufrufe • vor 4 Monaten

Building Apps with Helidon & MicroProfile

338 Aufrufe • vor 4 Monaten

Helidon + Micronaut Data: Productivity without Bloat

379 Aufrufe • vor 4 Monaten

MicroStream Hackathon Weekly Q&A | Edition 9...

44 Aufrufe • vor 4 Monaten

CIO of Tomorrow - Performance is Everything...

54 Aufrufe • vor 4 Monaten

MicroStream Hackathon Weekly Q&A - Helidon |...

GraalVM and MicroStream: Native Image & Ultra Fast...

MicroStream Hackathon Weekly Q&A | Edition 7...

Helidon DB Client

132 Aufrufe • vor 4 Monaten

Fast UI Development with RapidClipse (German)

GraalVM: Native Image - Cooking Guide

Jetzt kostenloses JUG Ticket für alle 4 Tage sichern !

www.jcon.one

Fast Lane

JUG Görlitz aufgepasst !!!

Jetzt könnt Ihr kostenlos Online-Trainings im Wert von 1.699 EUR bei Fast Lane buchen. Einfach einen beliebigen Kurs und Termin aussuchen und mit unserem Buchungs-Code für 0,00 EUR buchen ...

Fast Lane

Book Any Course for Free !

GraalVM - Online Training Live		
GraalVM: Build Native Images	1 Tag	**890 €**

MicroStream - Online Training Live		
MicroStream Fundamentals	2 Tage	**1.690 €**
MicroStream Advanced	2 Tage	**1.890 €**

Helidon - Online Training Live		
Helidon & MicroProfile Fundamentals	2 Tage	**1.690 €**
Helidon MP & MicroProfile Advanced	2 Tage	**1.890 €**
Helidon SE Advanced	2 Tage	**1.890 €**

Open Liberty - Online Training Live		
Open Liberty & MicroProfile Fundamentals	2 Tage	**1.690 €**
Open Liberty & MicroProfile Advanced	2 Tage	**1.890 €**

Quarkus - Online Training Live		
Quarkus & MicroProfile Fundamentals	2 Tage	**1.690 €**
Quarkus & MicroProfile Advanced	2 Tage	**1.890 €**

Payara Micro - Online Training Live		
Payara Micro & MicroProfile Fundamentals	2 Tage	**1.690 €**
Payara Micro & MicroProfile Advanced	2 Tage	**1.890 €**

Micronaut - Online Training Live		
Micronaut Fundamentals	2 Tage	**1.690 €**
Micronaut Advanced	2 Tage	**1.890 €**

Spring Boot - Online Training Live		
Spring Boot Cloud-Native - Fundamentals	2 Tage	**1.690 €**
Spring Boot Cloud-Native - Advanced	2 Tage	**1.890 €**

www.microservices.education

JUG Booking Code: **TeQ-QyoBvsDJ**

LinkedIn: markuskett | Email: m.kett@microstream.one