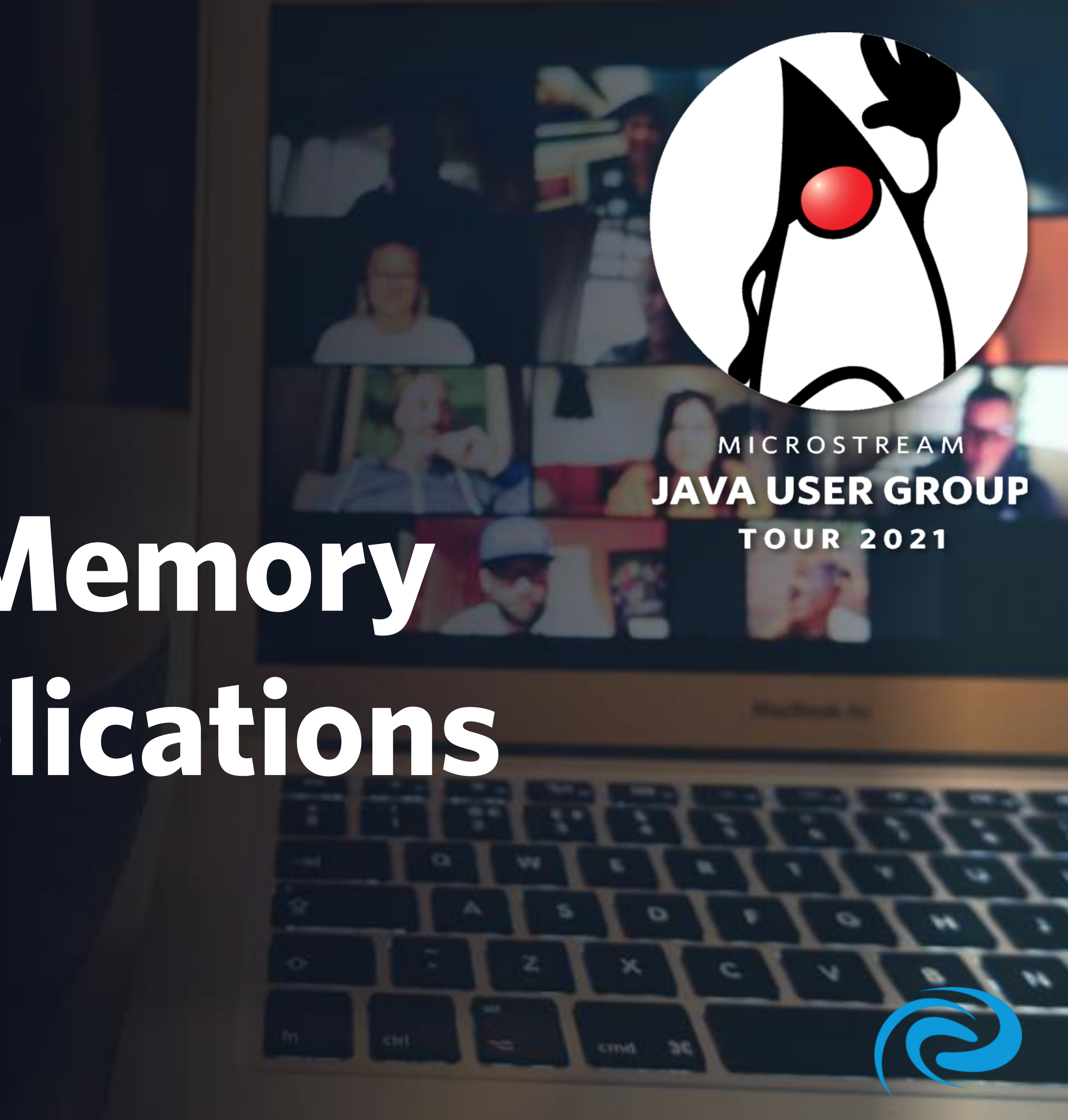
JUG Saxony

SEP 23, 2021 19:00 CEST

Ultra-fast In-Memory Database Applications



MICROSTREAM
JAVA USER GROUP
TOUR 2021

Saxony JUG !

Book your microservice framework online training course at Fast Lane for free of charge $0.00 and save $1,899. Check out the schedule, choose any course you like and enroll for free by using our booking code ...

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: **YEDlyExn**

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 us

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

Florian Habermann
CTO at MicroStream,
Contributor to Project Helidon (Oracle)
Project Lead RapidClipse Java IDE
Project Lead JPA-SQL
Conference Speaker

Twitter: @FHHabermann
LinkedIn: florian-habermann
Email: f.habermann@microstream.one

www.microstream.one

Create ultra-fast in-memory database apps & microservices
Next Generation Java Serialization
Java-Native Persistence & Communication

MicroStream

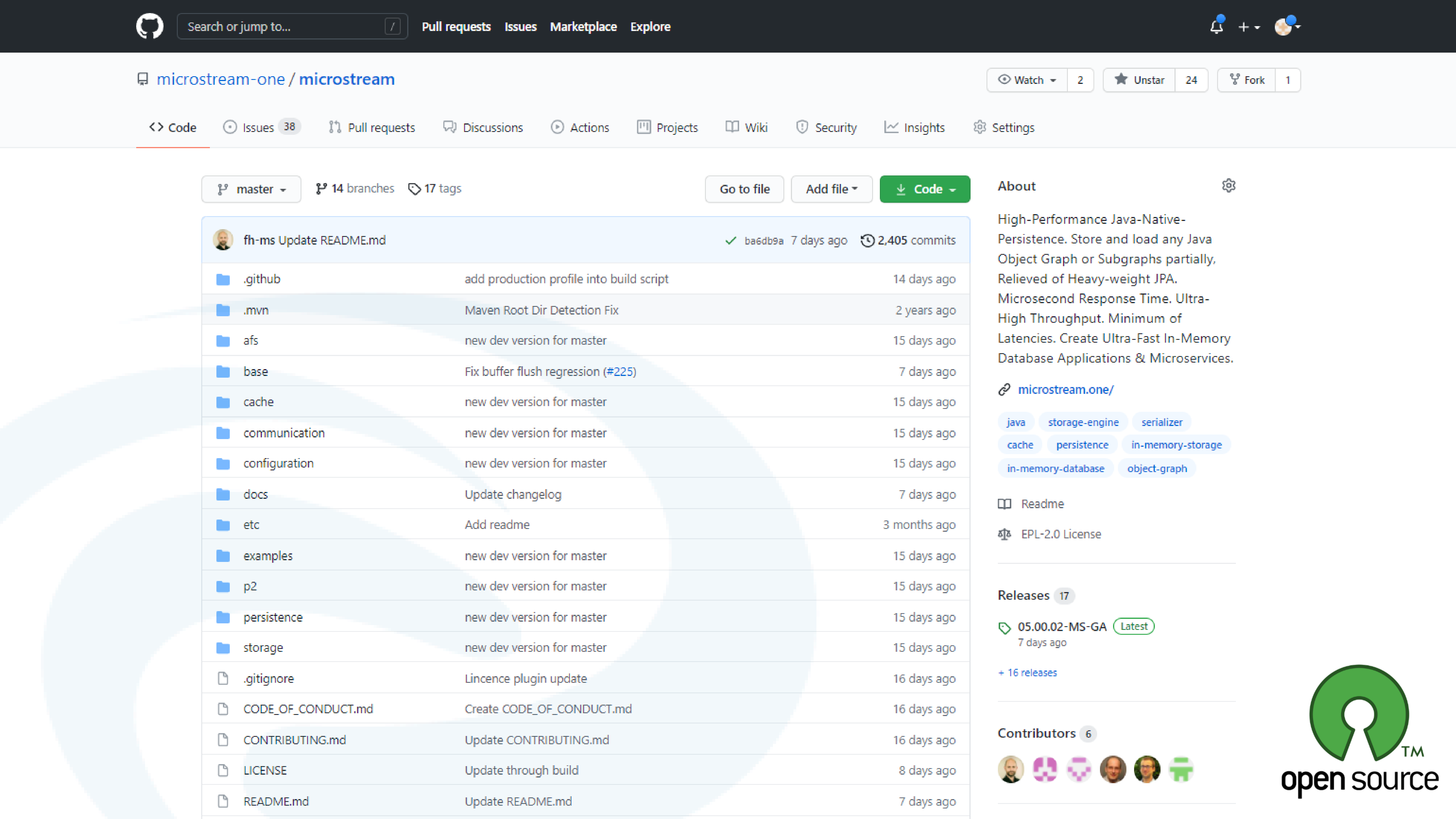
Announcement:

MicroStream is now Open Source

Build Ultra-Fast In-Memory Data Processing Apps for Android.

Gigantic Data Throughput. Gigantic Workloads. Microsecond Query Time In-Memory.

Eliminate Latencies and Save Energy. Half Your Development Effort & Time-to-Market.



microstream-one / microstream

14 branches 17 tags

fh-ms Update README.md ✓ ba6db9a 7 days ago 2,405 commits

.github	add production profile into build script	14 days ago
.mvn	Maven Root Dir Detection Fix	2 years ago
afs	new dev version for master	15 days ago
base	Fix buffer flush regression (#225)	7 days ago
cache	new dev version for master	15 days ago
communication	new dev version for master	15 days ago
configuration	new dev version for master	15 days ago
docs	Update changelog	7 days ago
etc	Add readme	3 months ago
examples	new dev version for master	15 days ago
p2	new dev version for master	15 days ago
persistence	new dev version for master	15 days ago
storage	new dev version for master	15 days ago
.gitignore	Lincence plugin update	16 days ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	16 days ago
CONTRIBUTING.md	Update CONTRIBUTING.md	16 days ago
LICENSE	Update through build	8 days ago
README.md	Update README.md	7 days ago

About

High-Performance Java-Native-Persistence. Store and load any Java Object Graph or Subgraphs partially, Relieved of Heavy-weight JPA. Microsecond Response Time. Ultra-High Throughput. Minimum of Latencies. Create Ultra-Fast In-Memory Database Applications & Microservices.

microstream.one/

java storage-engine serializer cache persistence in-memory-storage in-memory-database object-graph

Readme

EPL-2.0 License

Releases 17

05.00.02-MS-GA Latest
7 days ago

+ 16 releases

Contributors 6

open source™

Maven Download

pom.xml

```xml
<repositories>
    <repository>
        <id>microstream-releases</id>
        <url>https://repo.microstream.one/repository/maven-public/</url>
    </repository>
</repositories>
<dependencies>
    <dependency>
        <groupId>one.microstream</groupId>
        <artifactId>storage.embedded</artifactId>
        <version>04.01.00-MS-GA</version>
    </dependency>
    <dependency>
        <groupId>one.microstream</groupId>
        <artifactId>storage.embedded.configuration</artifactId>
        <version>04.01.00-MS-GA</version>
    </dependency>
</dependencies>
```

helidon.io

MicroStream now Part of Helidon

helidon.io MicroStream

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

Our Values

Main Contributor

100% focus on writing the MicroStream code.

Large Test-Coverage

We guarantee a high code quality and stable code base.

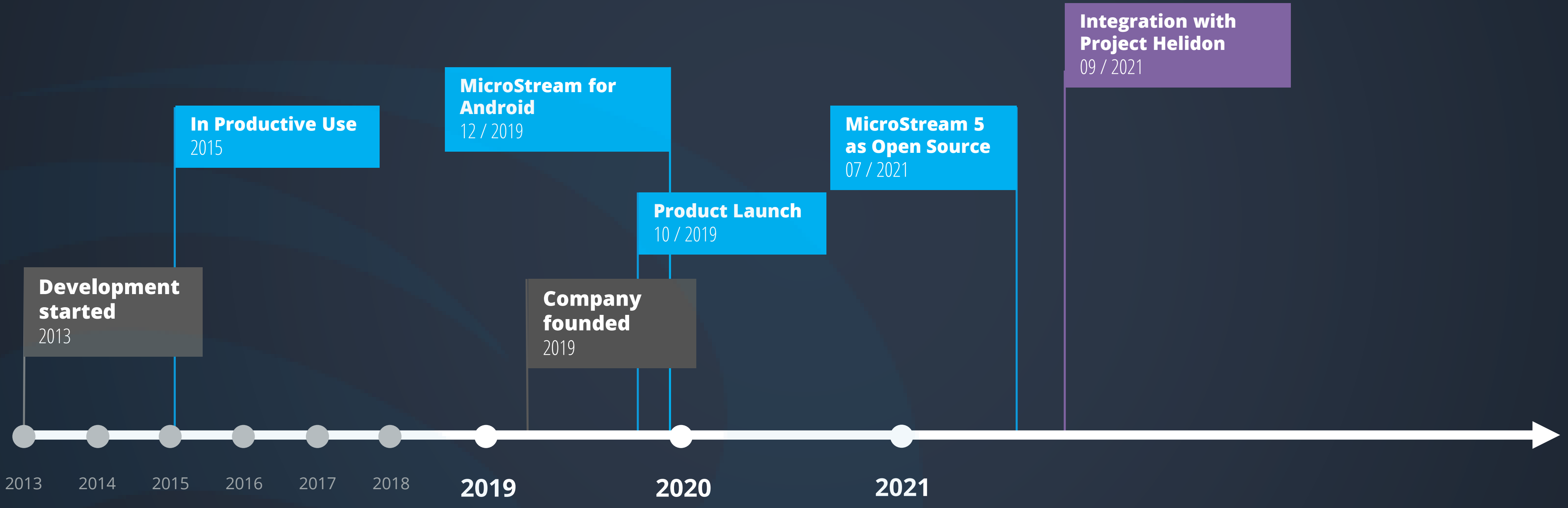
Update Warranty

Updates for all MicroStream versions for 8 years guaranteed.

Enterprise Support

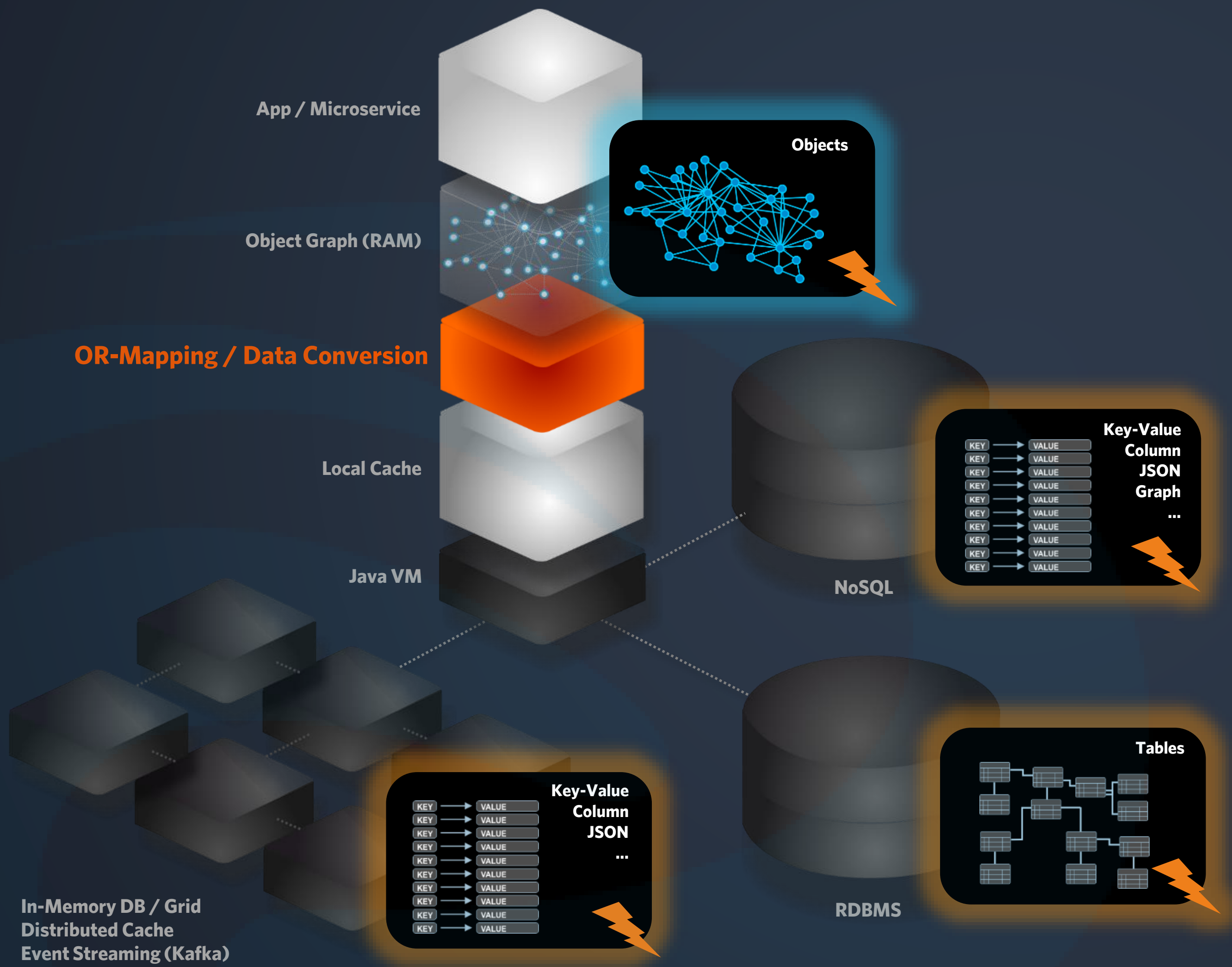
Enterprise-grade security, first-class developer, and production support.

MicroStream History



Traditional Java Persistence

Traditional Java Persistence



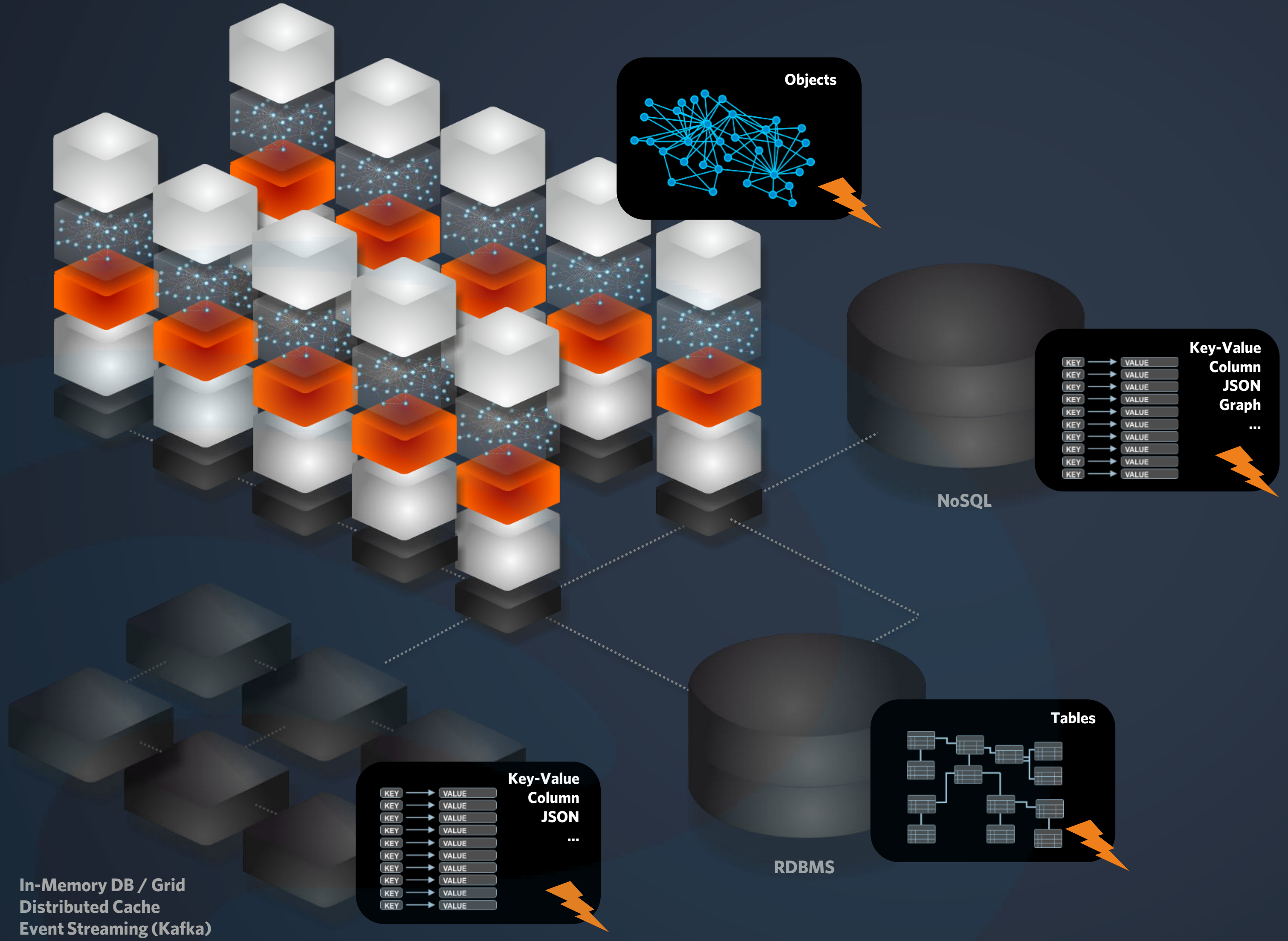
Millisecond
Query Time

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
- Complex architecture
- Expensive development process
- Inefficient concept requires expensive cluster infrastructure
- Increase your costs of infrastructure

Traditional Java Persistence



Millisecond

Query Time

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
- Complex architecture
- Expensive development process
- Inefficient concept requires expensive cluster infrastructure
- Increase your costs of infrastructure

High Development Effort

- 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

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

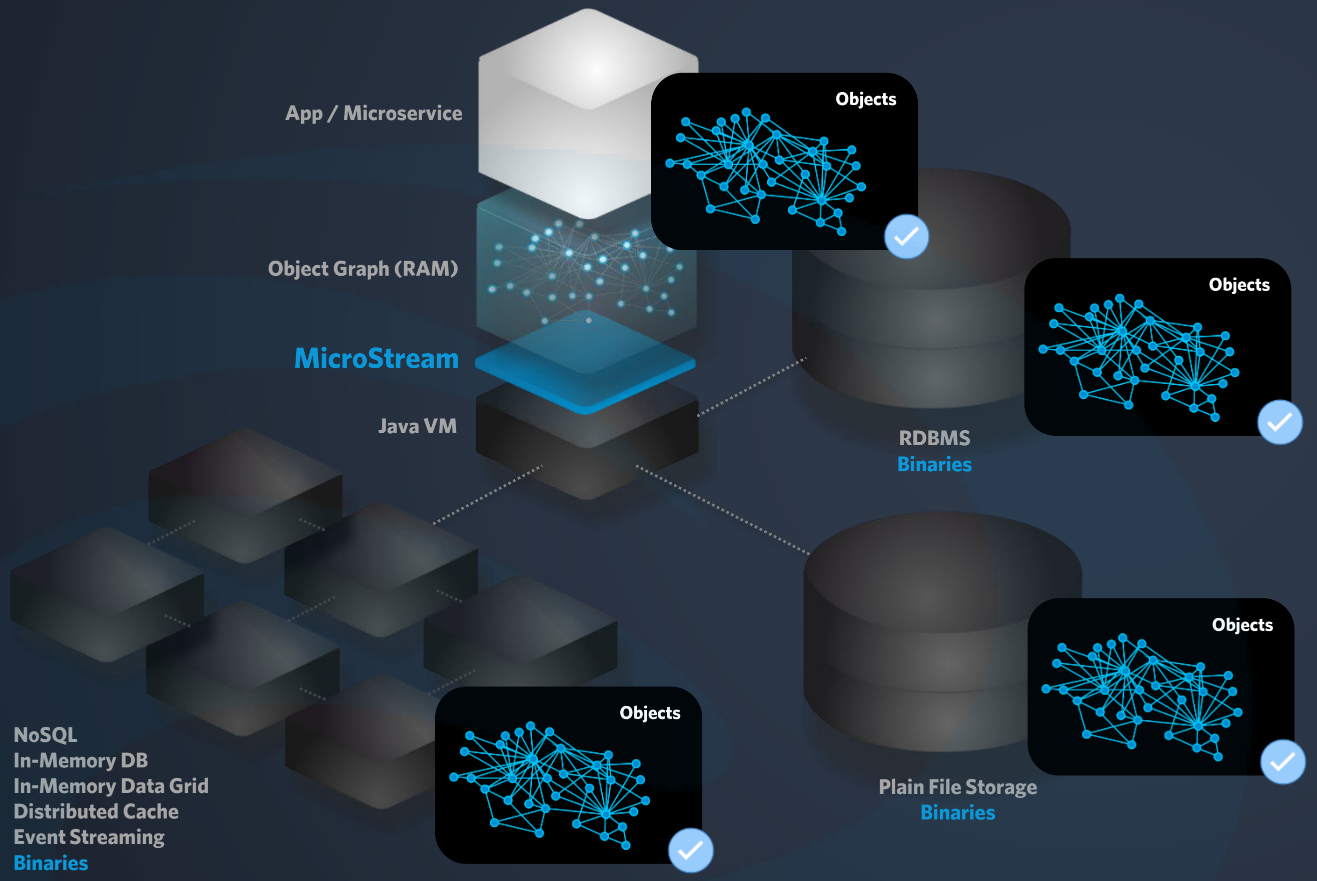
WIKIPEDIA
The Free Encyclopedia

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

MicroStream Java-Native Persistence

MicroStream Persistence



1000x faster Queries

Microsecond

Query Time

Streaming Objects Directly Into any Database

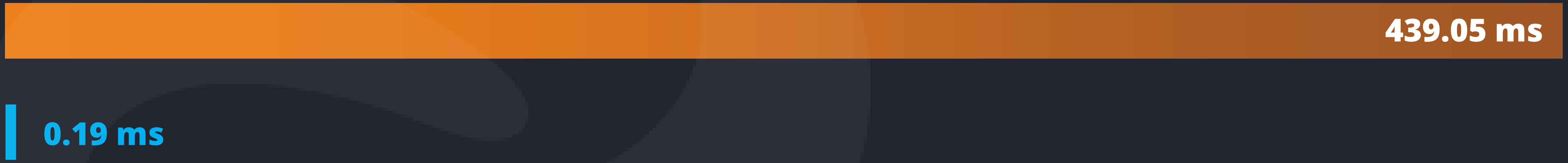
Conversion Eliminated !

- 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

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**	**190.11** **Queries / Second**	Database: **Oracle**	**83x** **Queries / Second**



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

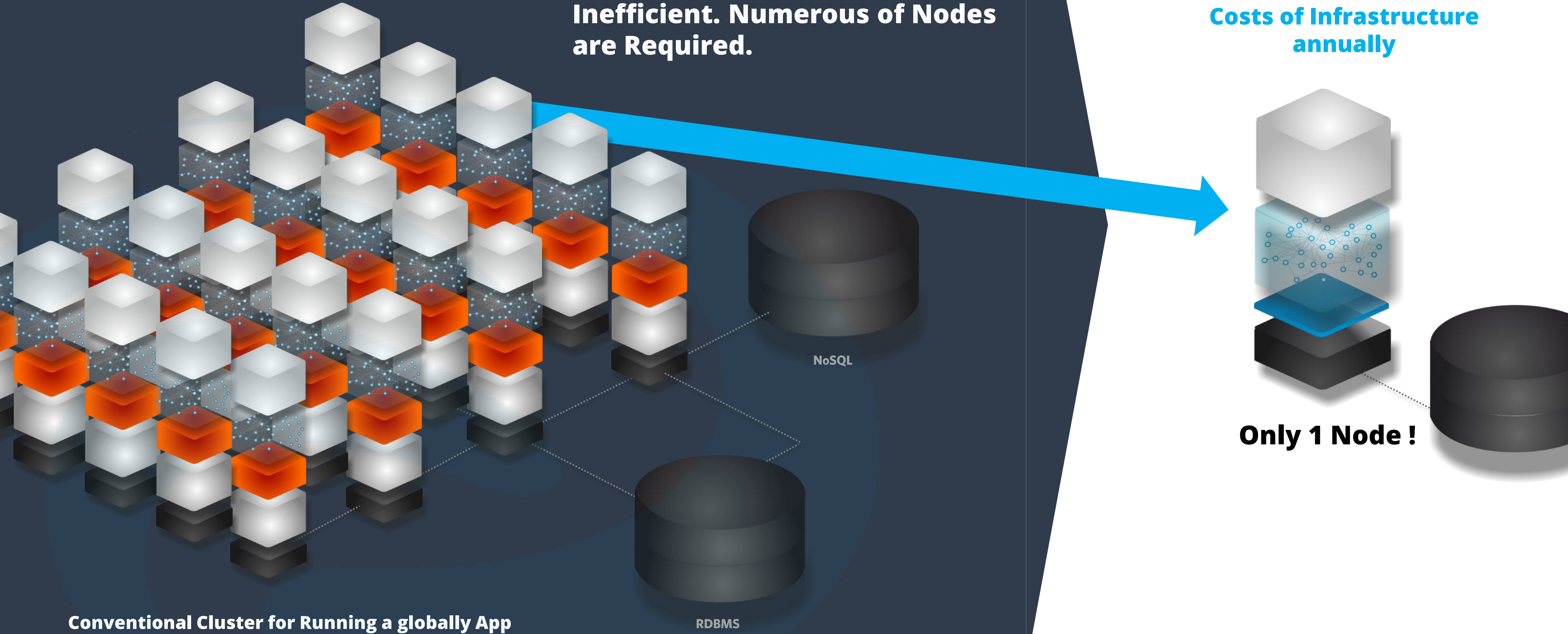
Save up to 90% Cloud Costs

Traditional Persistence Is Inefficient. Numerous of Nodes are Required.

Today with MicroStream:

- 87.5 %

Costs of Infrastructure annually

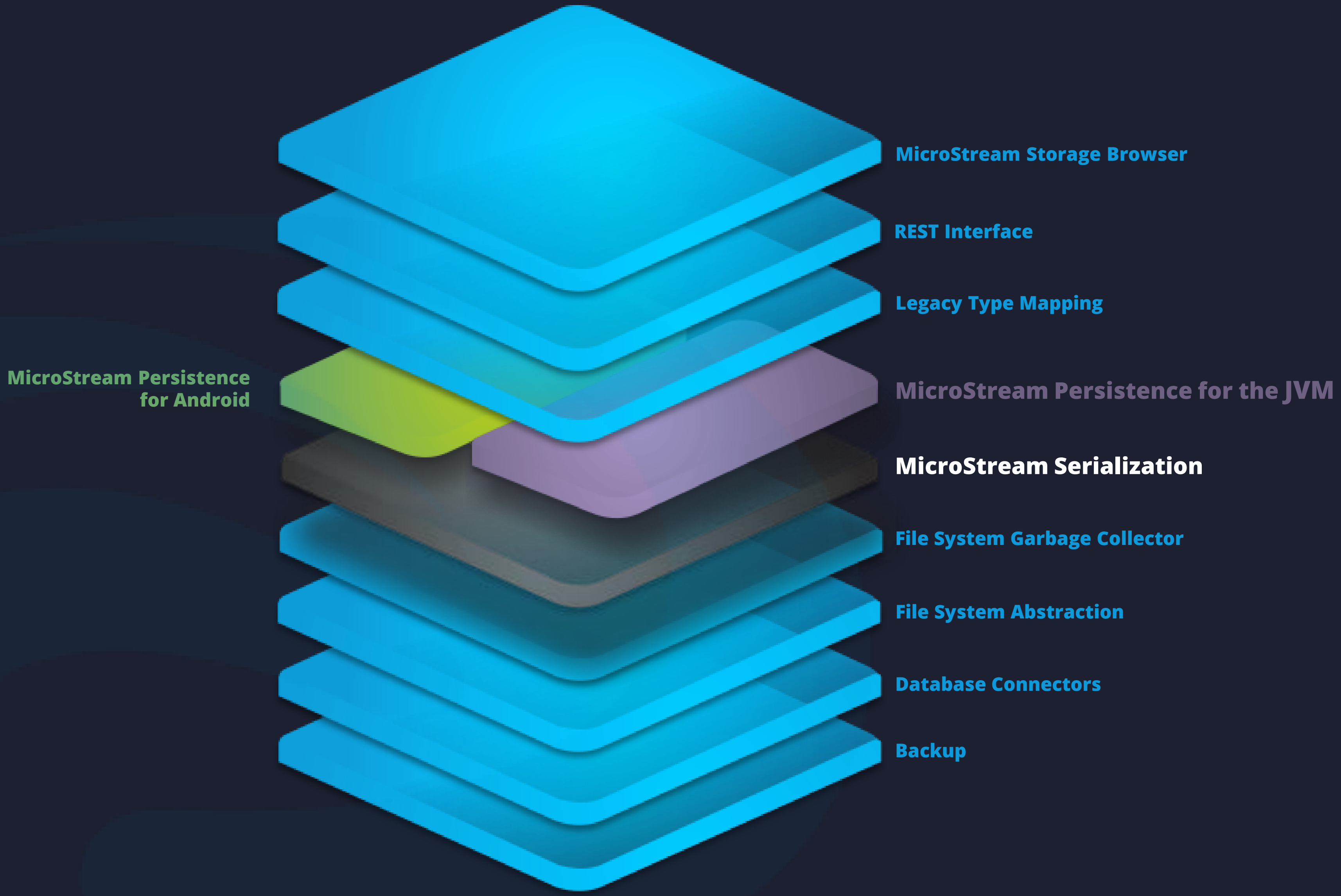


Conventional Cluster for Running a globally App

Simplifies Your Development Process

- 1 data structure (Java object graph)
- 1 data model, your object model, POJOs only
- Freely design of your Java object-model
- No mappings, no impedance mismatch
- No ORM framework
- Query language: Java Streams API, GraphQL
- No local cache needed
- Core Java only

MicroStream Components


MicroStream Storage Browser
REST Interface
Legacy Type Mapping
MicroStream Persistence for Android
MicroStream Persistence for the JVM
MicroStream Serialization
File System Garbage Collector
File System Abstraction
Database Connectors
Backup

Supported Storages

RDBMS

ORACLE DATABASE | ORACLE TIMESTEN | MySQL | PostgreSQL | MariaDB | amazon DynamoDB | SQLite

NoSQL

mongoDB | redis | kafka | hazelcast | ORACLE Coherence | ORACLE NOSQL DATABASE

Cloud Object Store

amazon webservices S3 | Microsoft Azure Blob Storage | ORACLE CLOUD

Runs Wherever Java Runs

Desktops

On-Premise

Cloud

Container

Native Image

Microservices

Android

JDK 8+

Use any JVM Technology

Java GraalVM Kotlin Scala Groovy Clojure android

How Does MicroStream Work ?

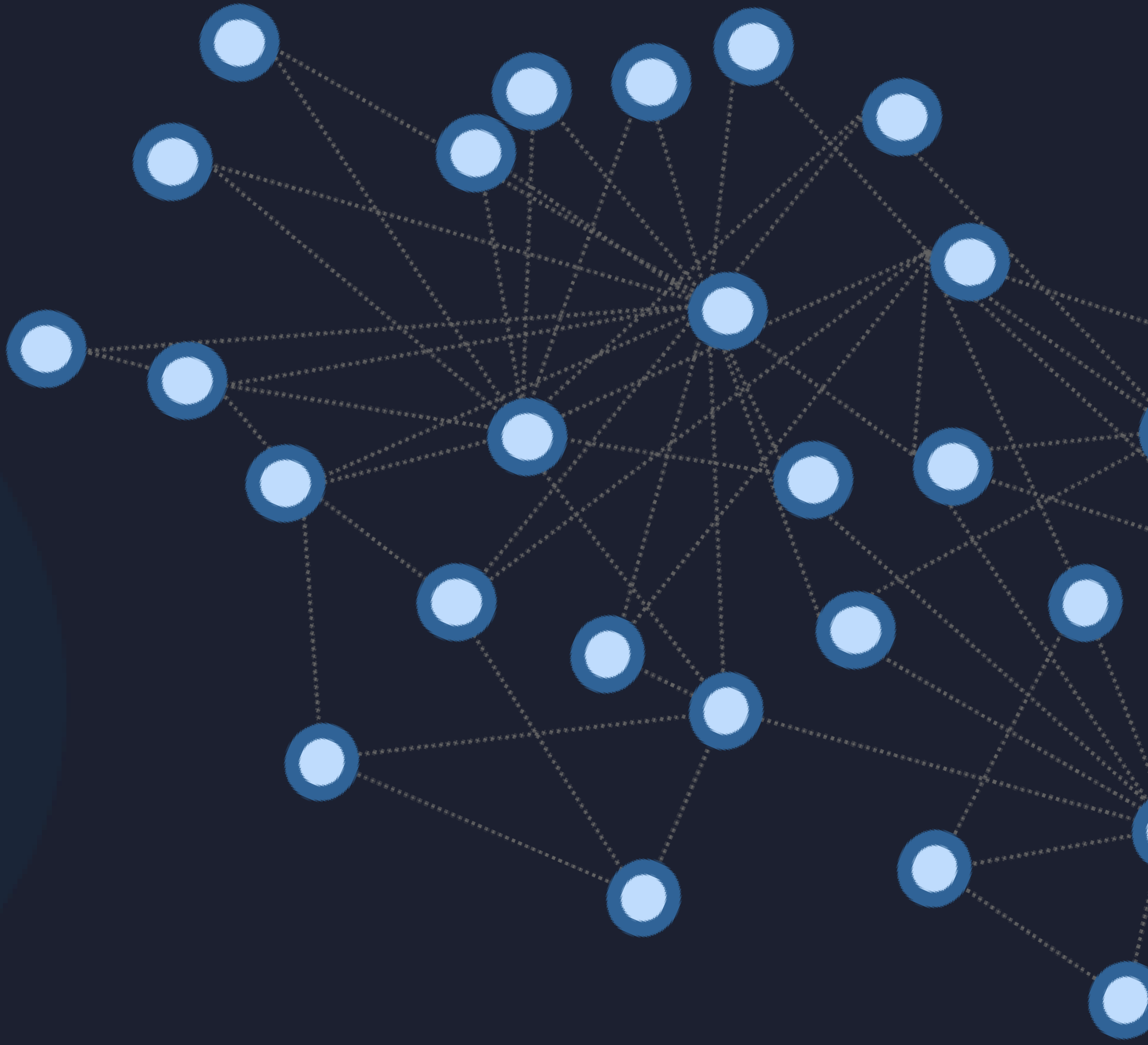
Data Model: Just POJOs

```java
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**

Freely Design Your Object Model

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

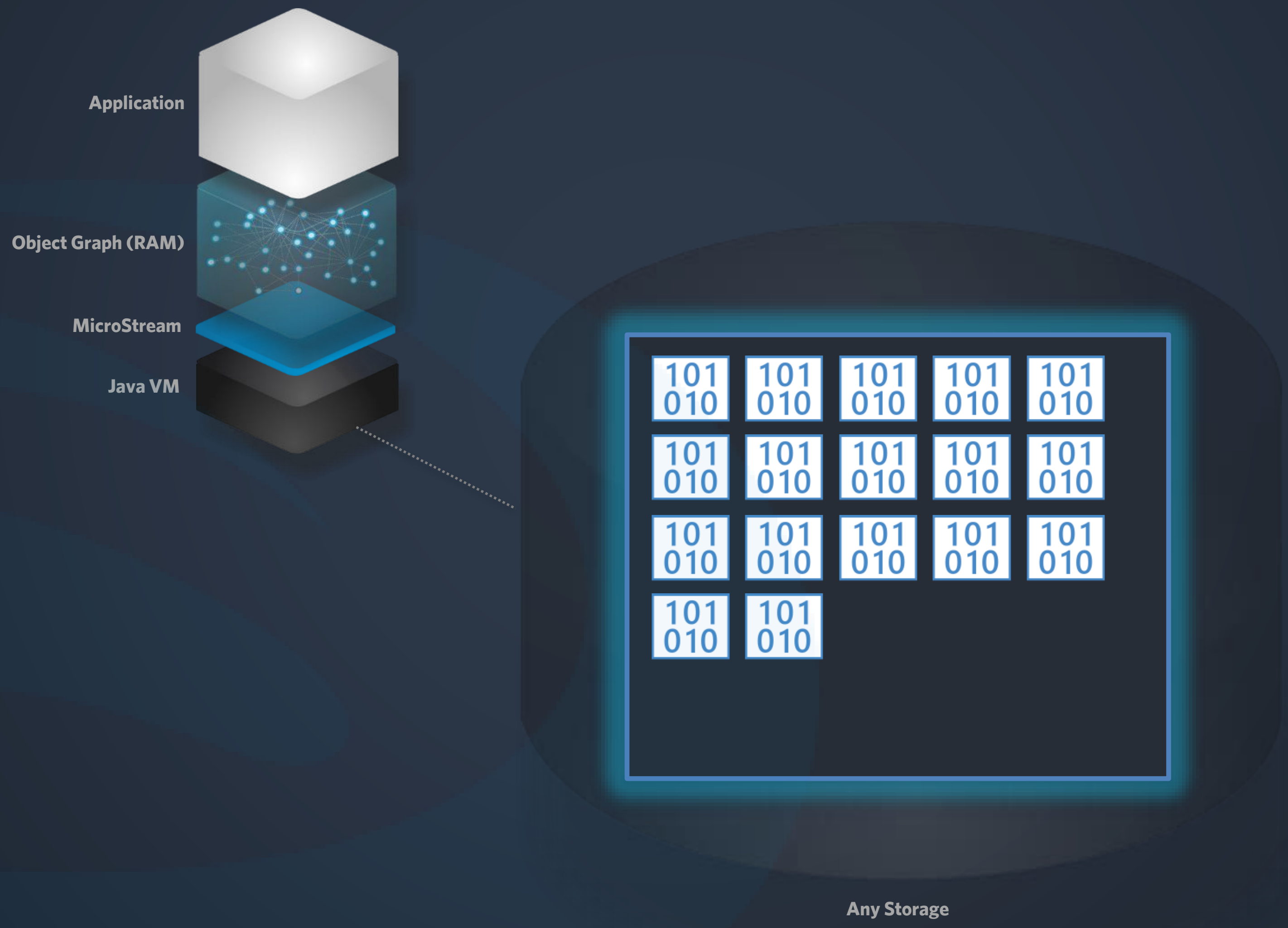


Persisting Objects

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

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

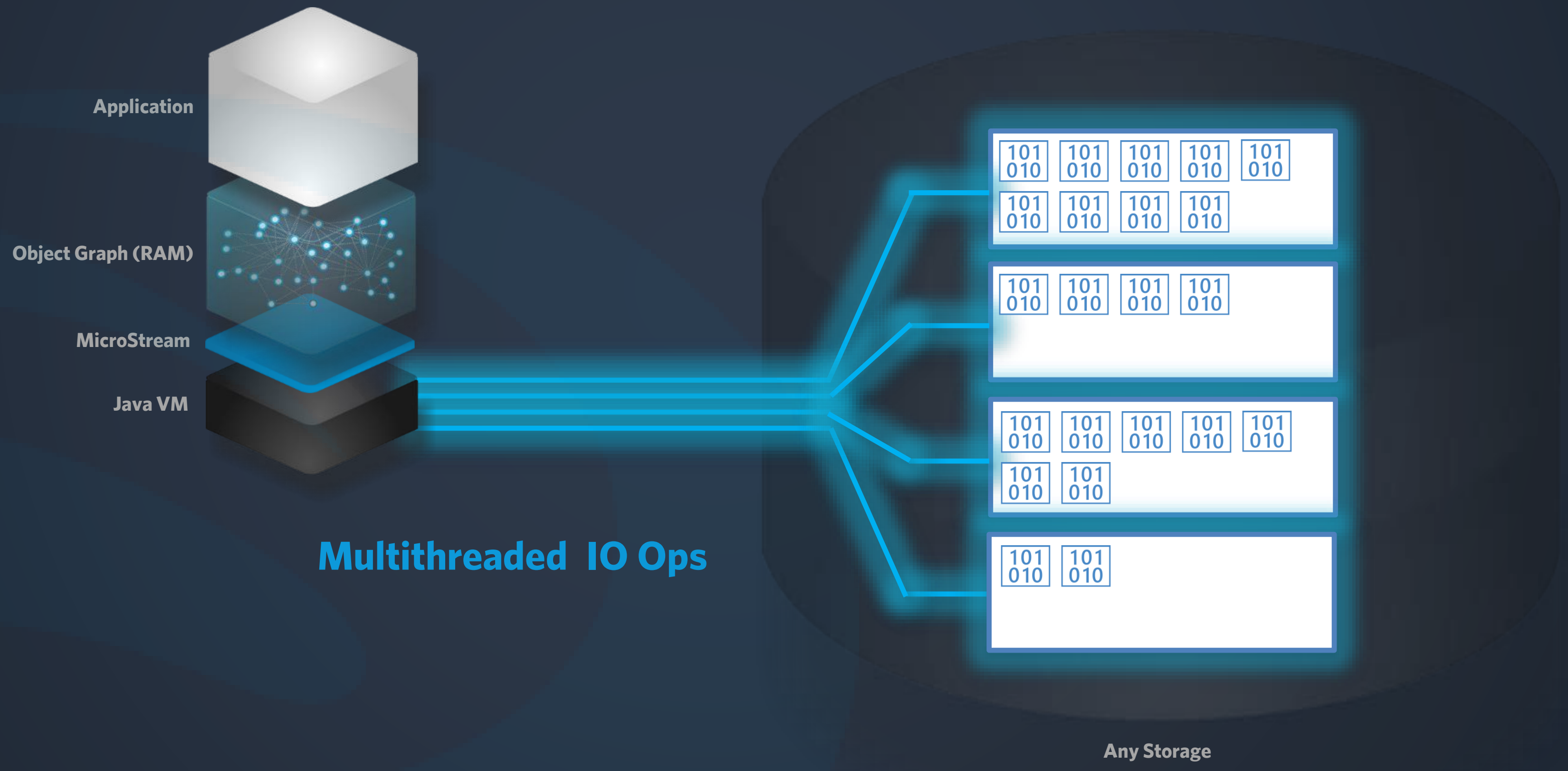
- Store any single object or subgraph explicitly
- Binary data format, no expensive mappings
- Append-only log strategy
- Custom-tailored type handling for best performance
- Store any Java type, any suited type is supported
- Atomic operation and ACID transaction-safe
- Multithreaded write ops for max performance
- Replaces 3 CRUD ops: Create, Update & Delete
- Using inheritance is trouble-free
- Strong consistency
- Gigantic data throughput

Append-only Log

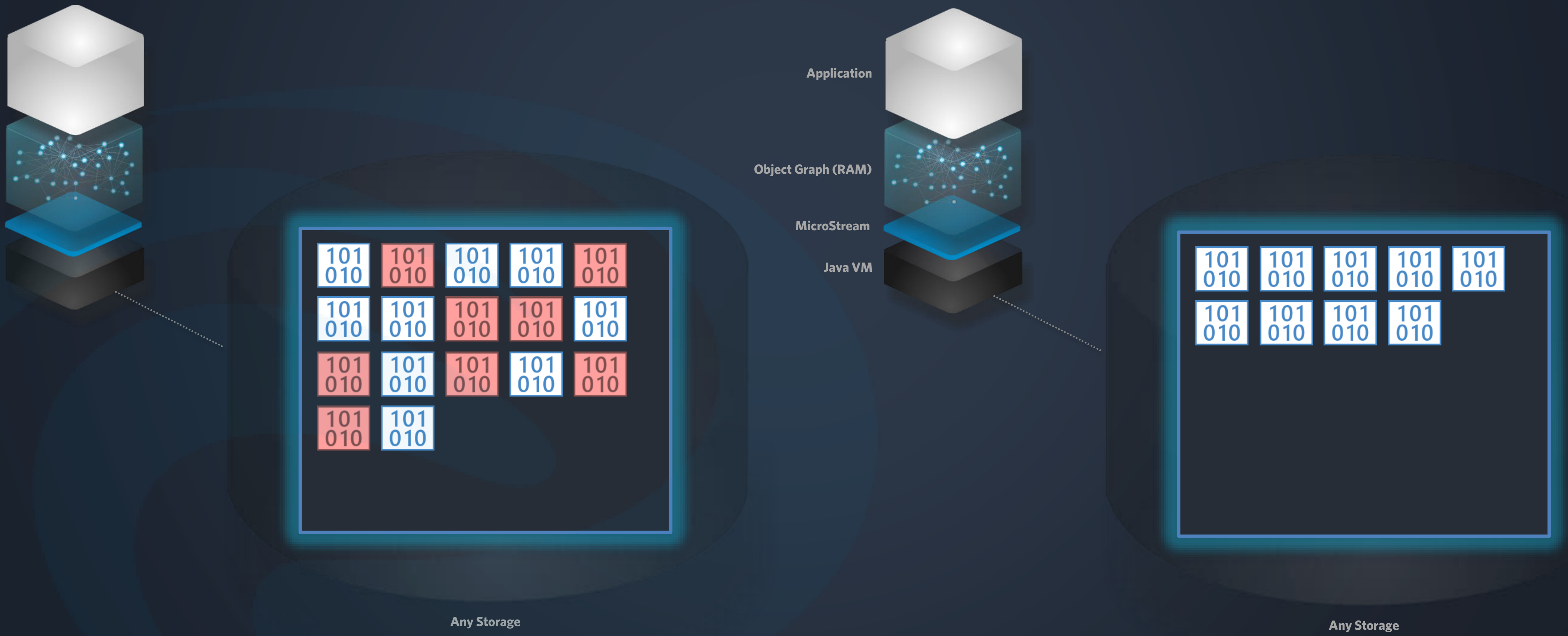


MicroStream Channels

4 Cores – 4 Channels
64 Cores – 64 Channels



Storage Garbage Collector



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);
  }

  ...
}
```

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

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

1000x faster Queries

Microsecond

Query Time

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

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

Accessing Your Data

Expose your data to external applications or services to enable accessing your data. Learn more

Manage Terabyte RAM Sizes

Azul's JVM Platform Prime minimizes garbage collection pause time and enables your Java app to handle up to 20 Terabyte RAM size trouble-free. Learn more

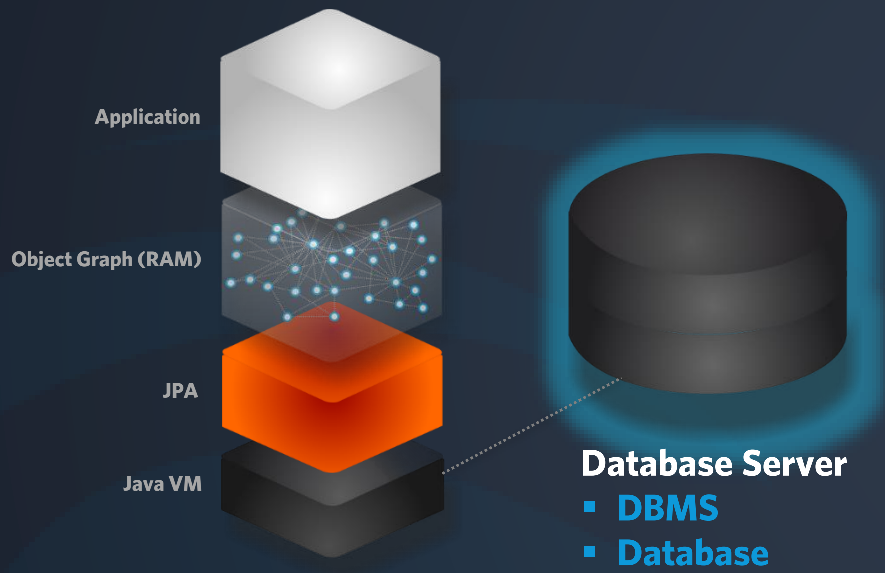
Fulltext Search

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

Rules & Challenges

Your Object Graph in RAM is Your Database

Traditional Database Server Paradigm

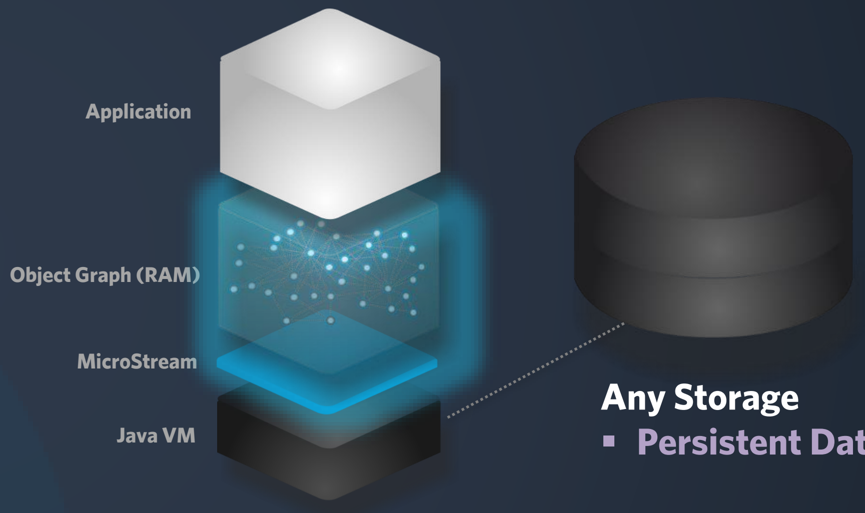


Database Server

- DBMS
- Database
- Queries
- Persistent Data

Your Database is here,
Queries are executed here

Java In-Memory Data Processing Paradigm



Any Storage

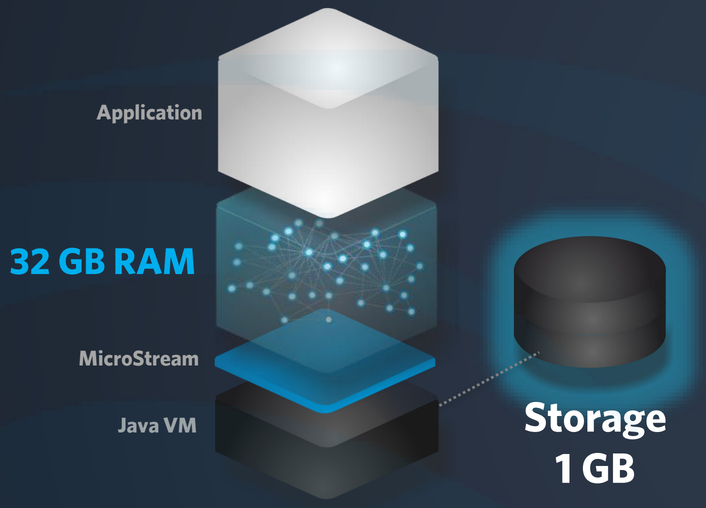
- Persistent Data

- Database (In-Memory)
- Queries

Your Database is here,
Queries are executed here

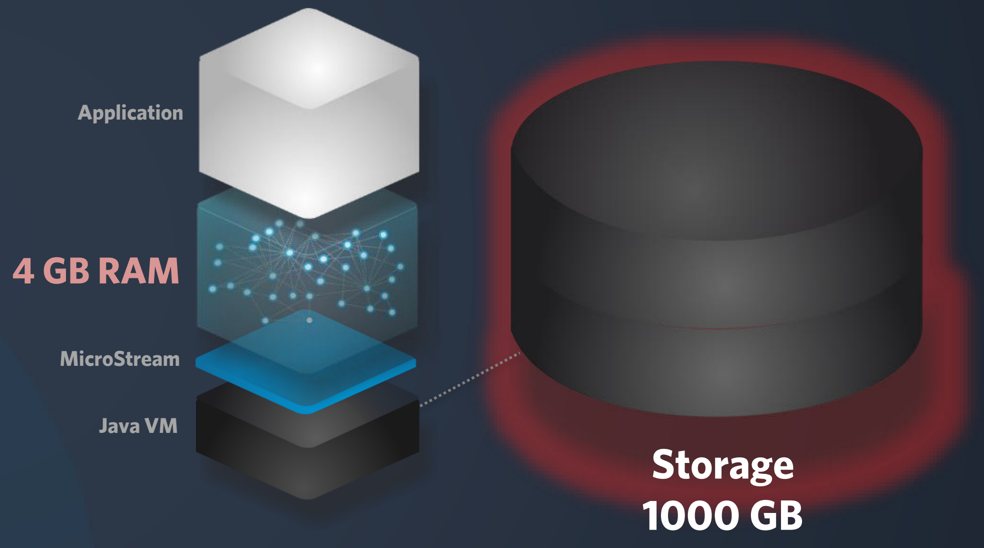
All In-Memory or Lazy-Loading ?

Enough RAM available:



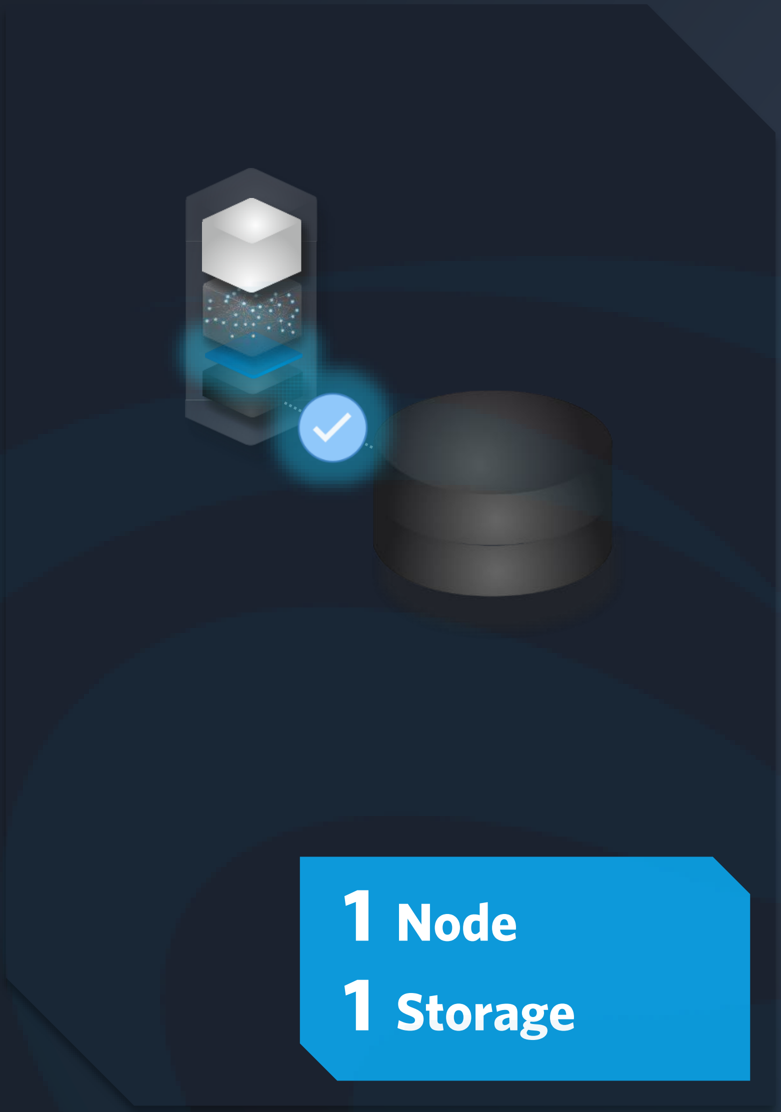
- Load your entire DB into RAM
- Pure in-memory computing
- No latencies
- Super fast
- Lower startup time

Data Storage is bigger than RAM:

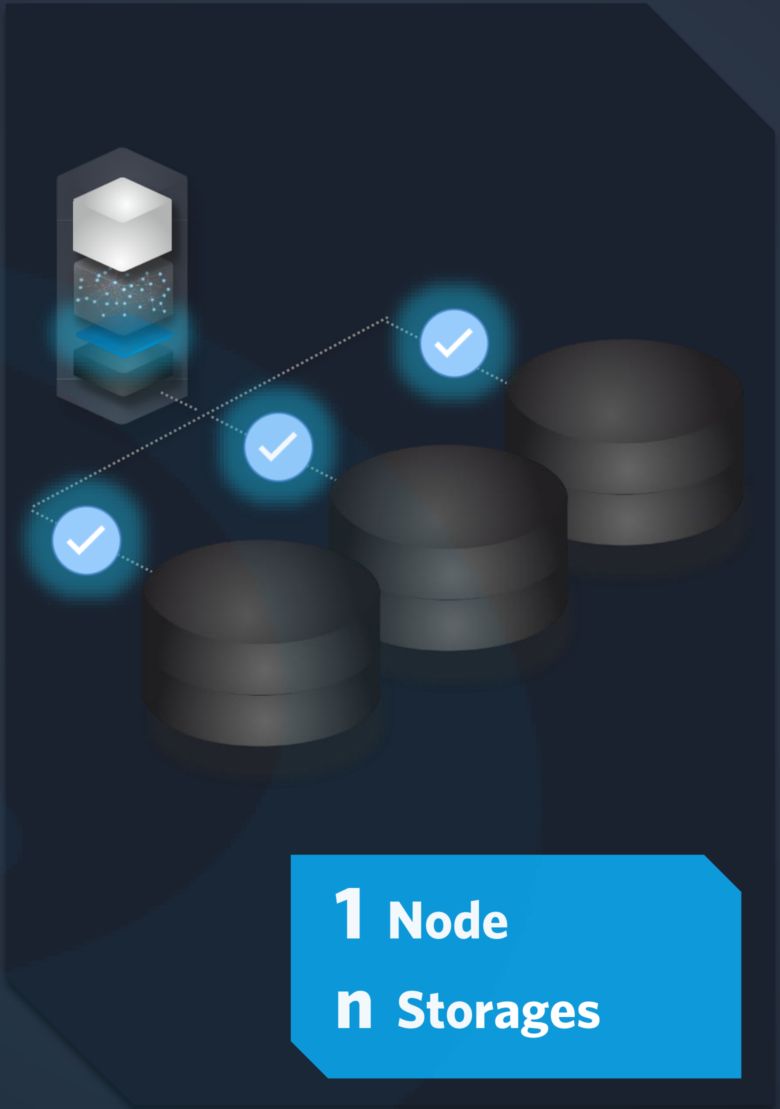


- Preload most important data only (eager loading)
- Use lazy-loading to load data on demand only
- Clear lazy references which are not used anymore
- Faster startup time

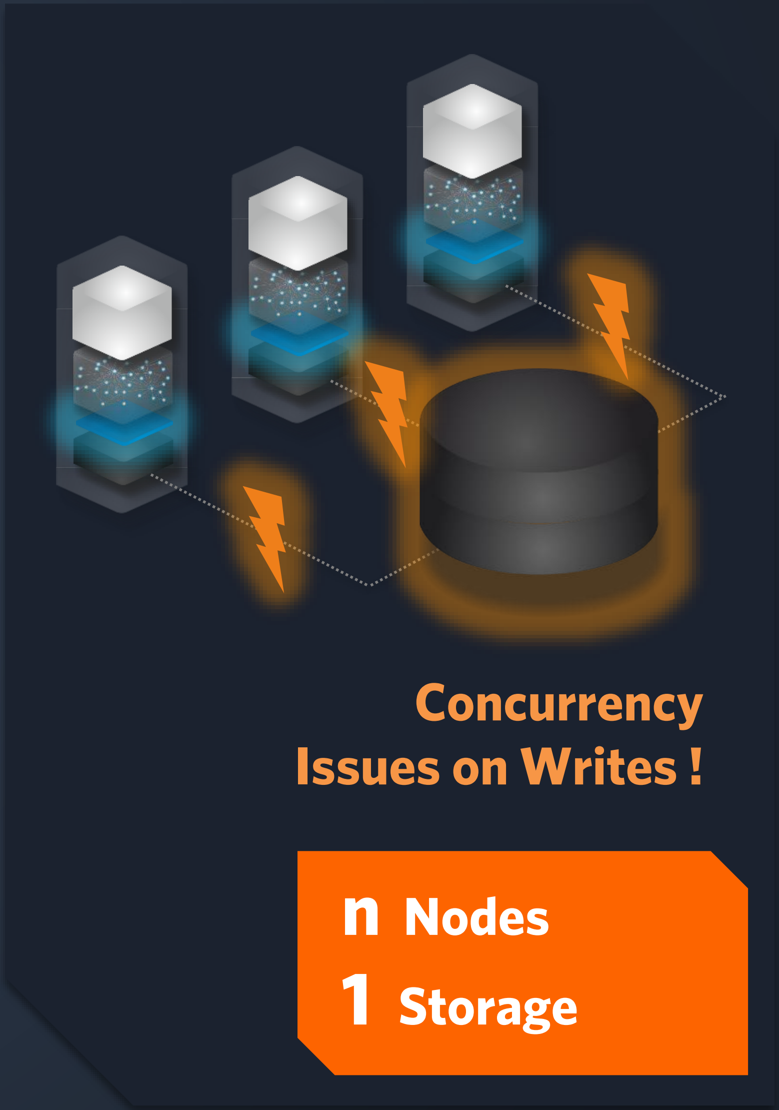
MicroStream Architecture Rules



1 Node
1 Storage



1 Node
n Storages



Concurrency
Issues on Writes !

n Nodes
1 Storage

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

Challenges with MicroStream

- **Built for Java developers**
- **Paradigm shift in database programming**
- **No SQL support**
- **MicroStream is a storage engine, but not a DBMS**
  - **Your application must cover DBMS tasks**
  - **You must care vor validation**
  - **You have to care for concurrency**
- **Not suited for DBAs**

MicroStream

Beyond Persistence ...

MicroStream Serialization

Java Serialization is the biggest security issue in Java. 50% of all vulnerabilities are linked to serialization.

Mark Reinhold

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

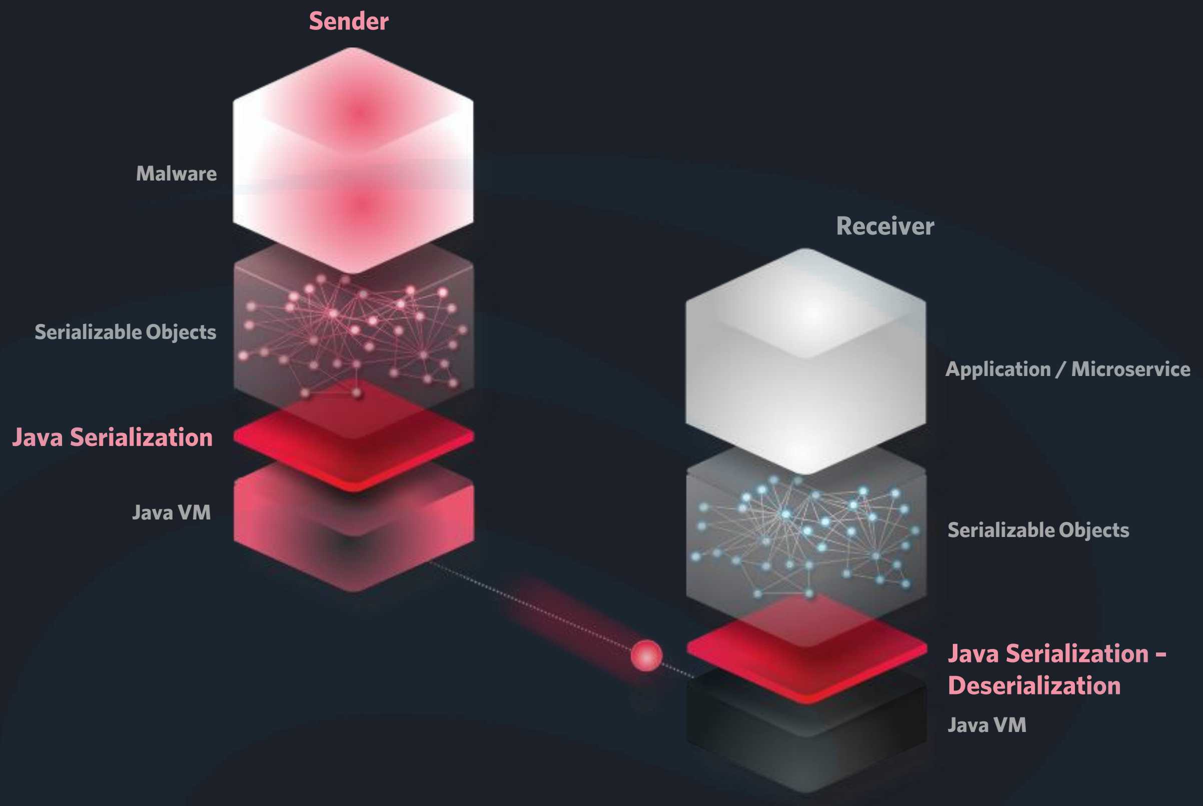
Java's serialization makes nearly every mistake imaginable and, poses an ongoing tax for library maintainers, language developers, and users.

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

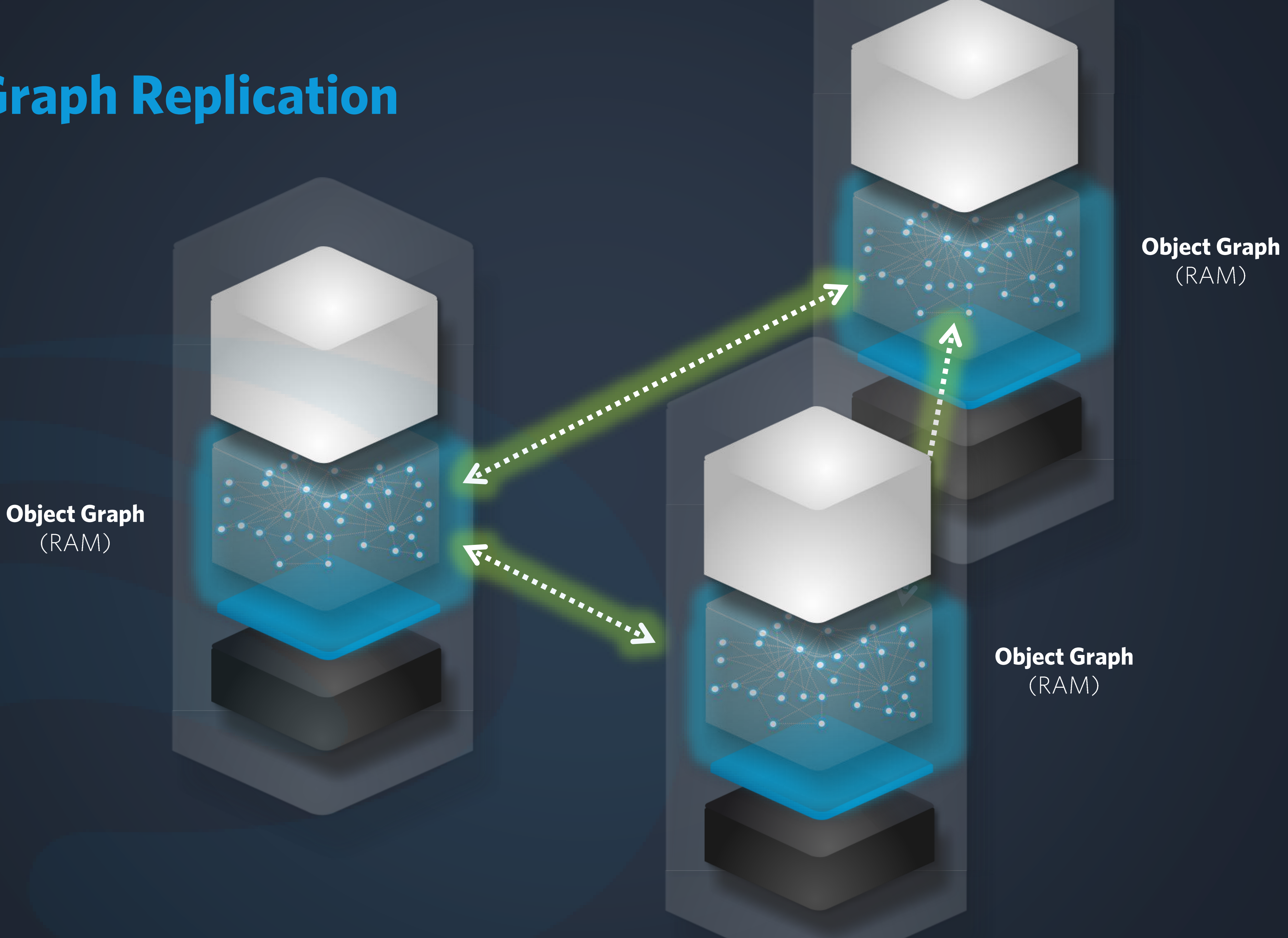
MicroStream

Roadmap

Coming soon ...

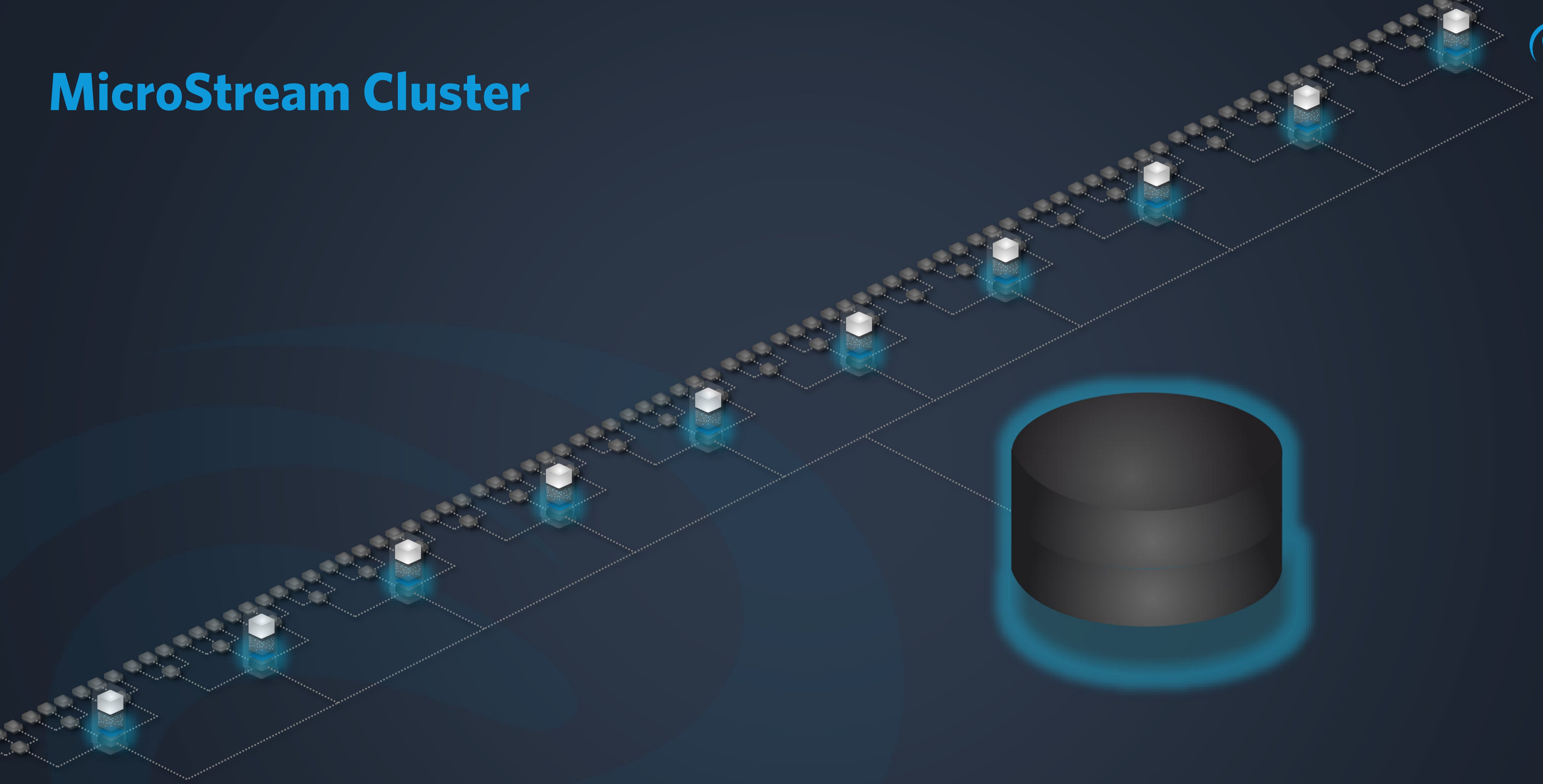
Distributed Systems with MicroStream Cluster

Object Graph Replication



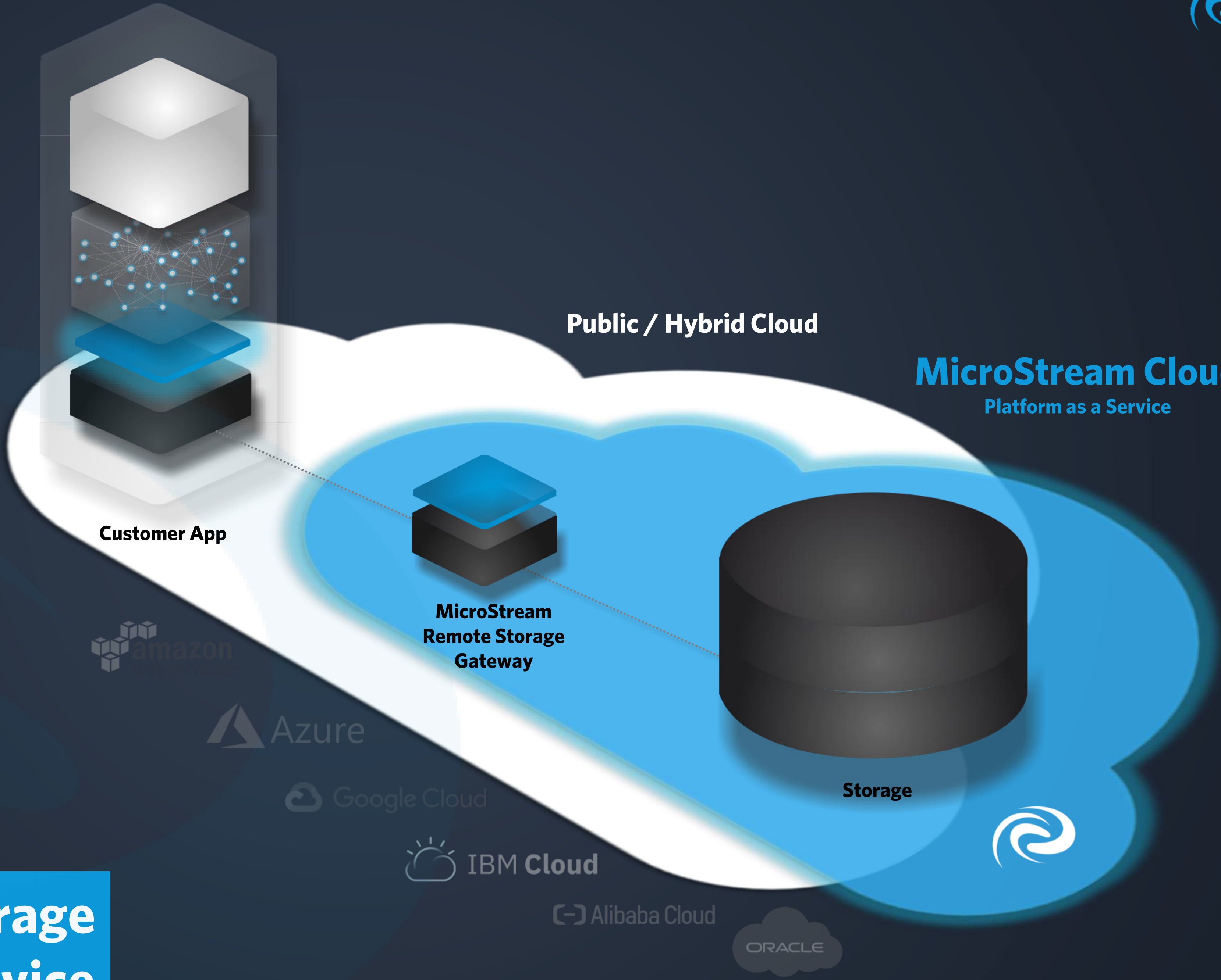
MicroStream Cluster

MicroStream Cluster



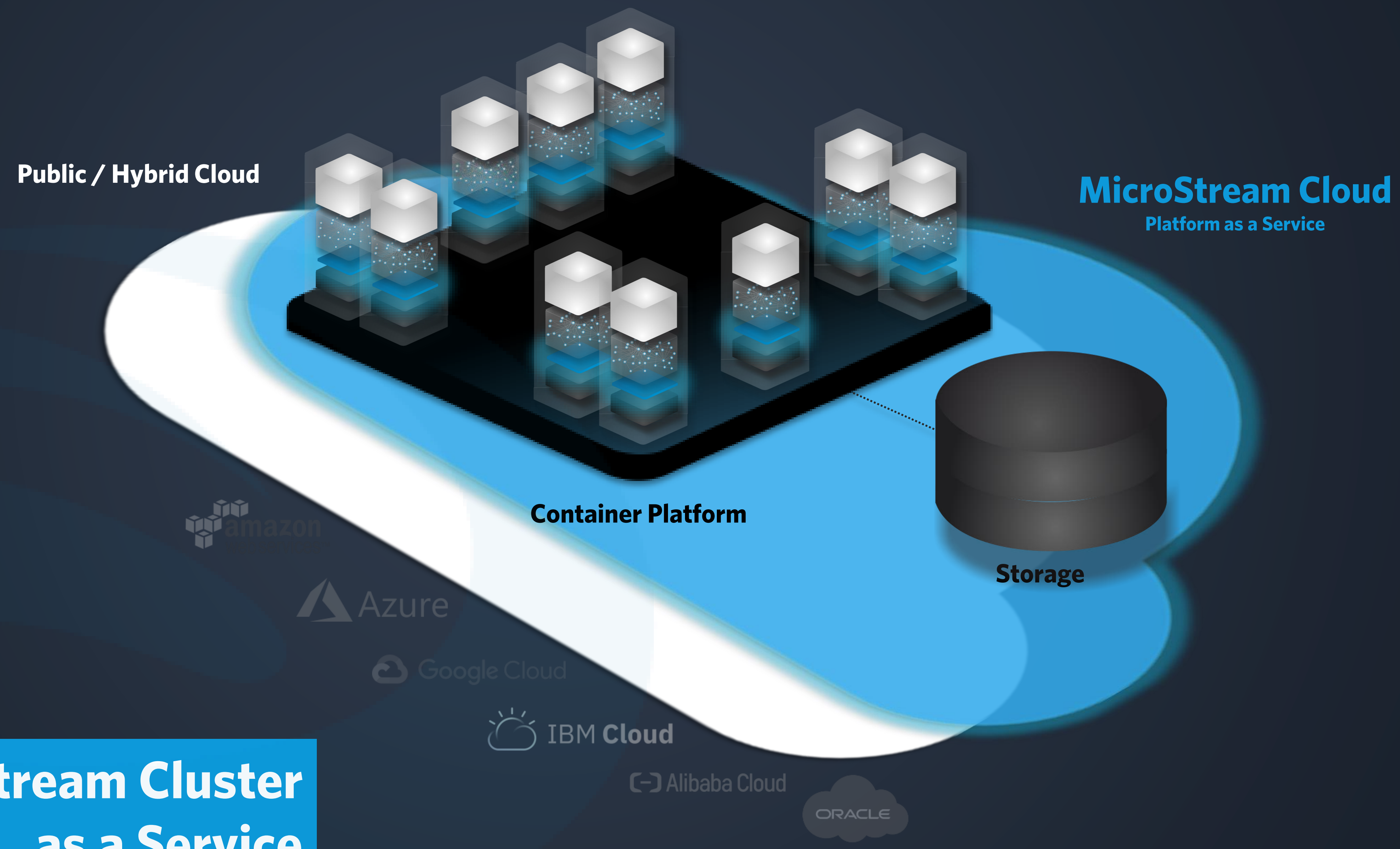
MicroStream Cloud
MicroStream Single Storage & MicroStream Cluster as a Service

MicroStream Cloud



MicroStream Single Storage as a Service

MicroStream Cloud

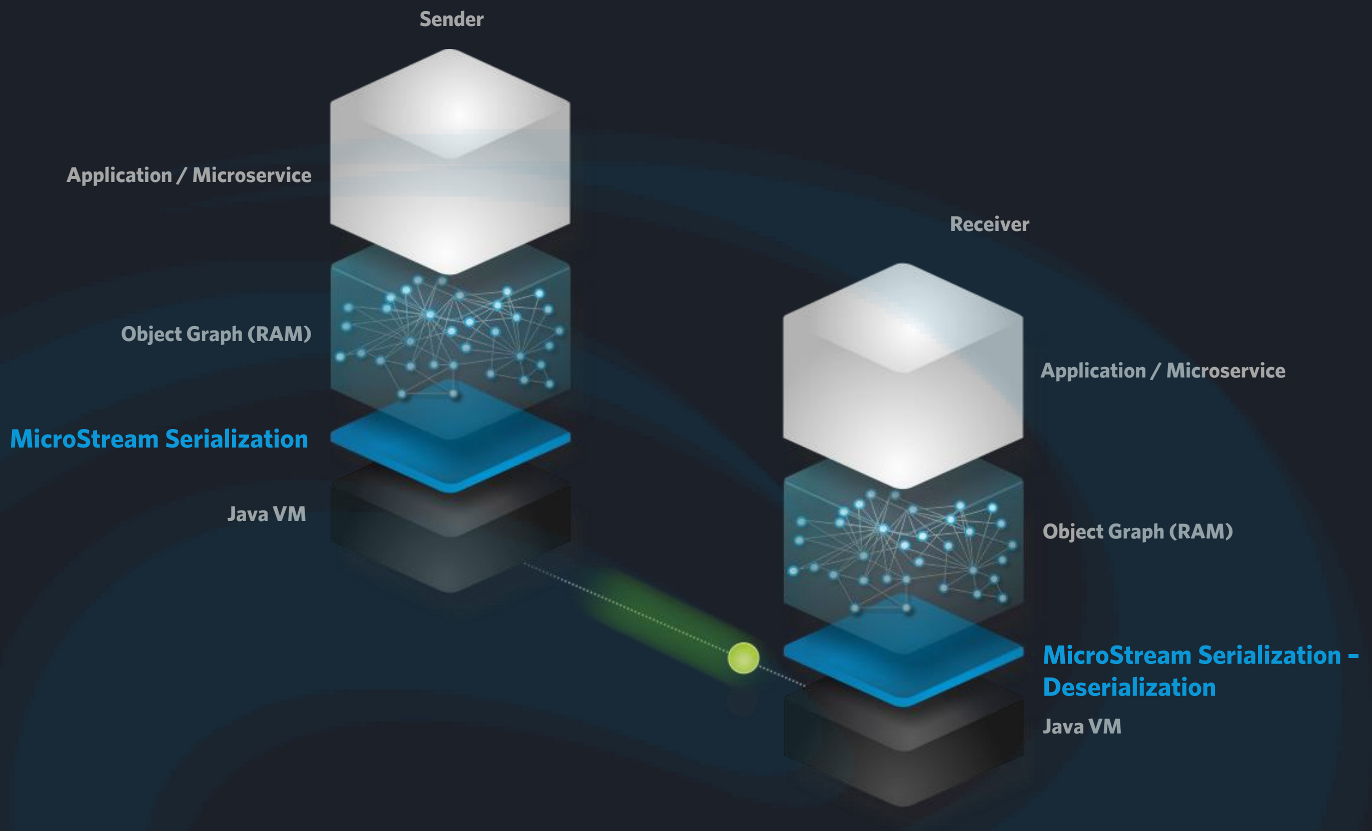


MicroStream Cluster as a Service

Benefits

- Simple setup
- Simpler cost control
- No cloud know-how required
- Fewer DevOp resources required
- High-availability
- Auto-scale
- Fully-managed services
- Enterprise-grade security
- Production support by MicroStream

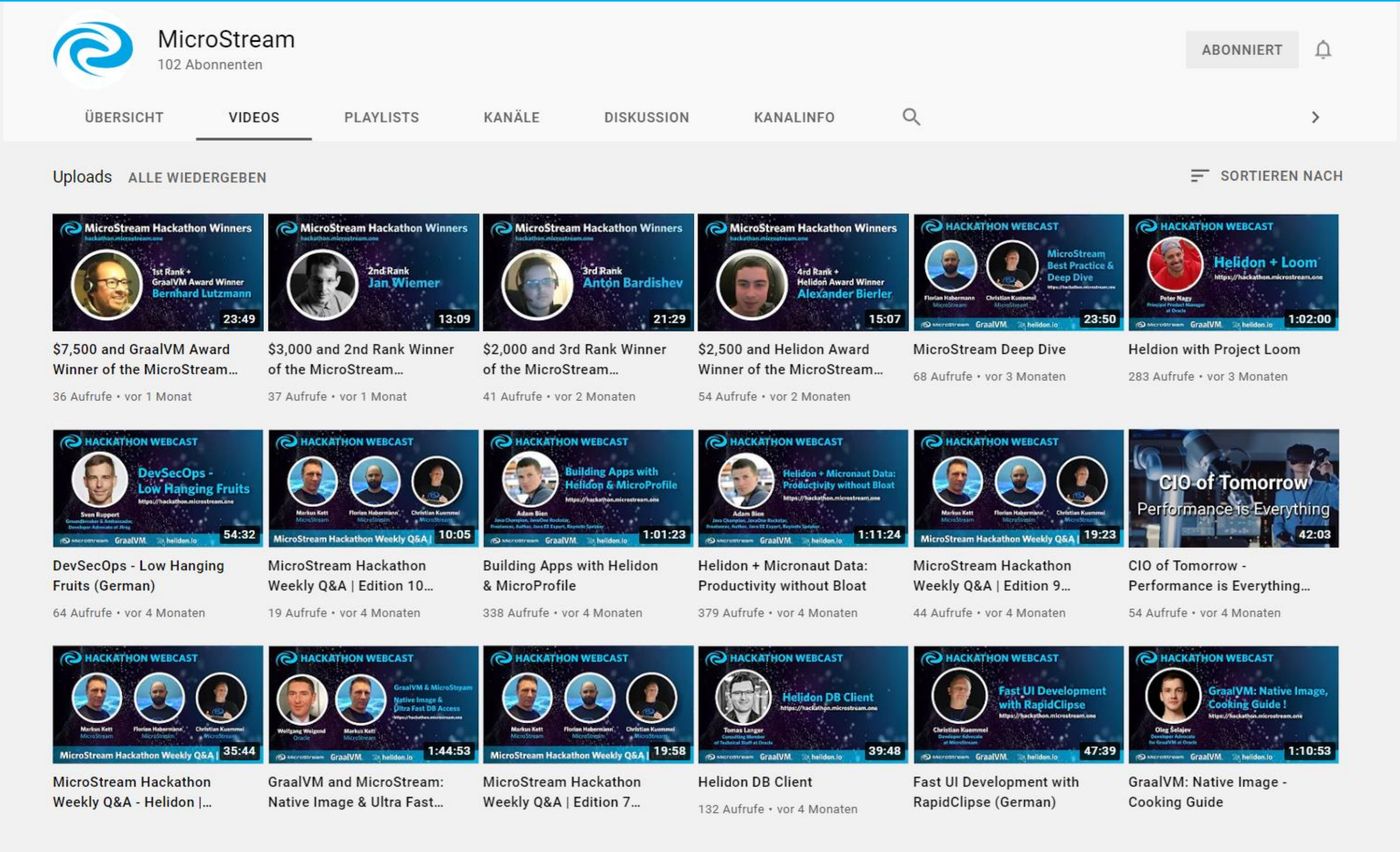
MicroStream Serialization


Sender
Application / Microservice
Object Graph (RAM)
MicroStream Serialization
Java VM
Receiver
Application / Microservice
Object Graph (RAM)
MicroStream Serialization - Deserialization
Java VM

- 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

Get Started ...

www.youtube.com/c/MicroStream/videos



MicroStream
102 Abonnenten

ABONNIERT

ÜBERSICHT · VIDEOS · PLAYLISTS · KANÄLE · DISKUSSION · KANALINFO

Uploads ALLE WIEDERGEBEN

SORTIEREN NACH

23:49
$7,500 and GraalVM Award Winner of the MicroStream...
36 Aufrufe • vor 1 Monat

13:09
$3,000 and 2nd Rank Winner of the MicroStream...
37 Aufrufe • vor 1 Monat

21:29
$2,000 and 3rd Rank Winner of the MicroStream...
41 Aufrufe • vor 2 Monaten

15:07
$2,500 and Helidon Award Winner of the MicroStream...
54 Aufrufe • vor 2 Monaten

23:50
MicroStream Deep Dive
68 Aufrufe • vor 3 Monaten

1:02:00
Heldion with Project Loom
283 Aufrufe • vor 3 Monaten

54:32
DevSecOps - Low Hanging Fruits (German)
64 Aufrufe • vor 4 Monaten

10:05
MicroStream Hackathon Weekly Q&A | Edition 10...
19 Aufrufe • vor 4 Monaten

1:01:23
Building Apps with Helidon & MicroProfile
338 Aufrufe • vor 4 Monaten

1:11:24
Helidon + Micronaut Data: Productivity without Bloat
379 Aufrufe • vor 4 Monaten

19:23
MicroStream Hackathon Weekly Q&A | Edition 9...
44 Aufrufe • vor 4 Monaten

42:03
CIO of Tomorrow - Performance is Everything...
54 Aufrufe • vor 4 Monaten

35:44
MicroStream Hackathon Weekly Q&A - Helidon |...

1:44:53
GraalVM and MicroStream: Native Image & Ultra Fast...

19:58
MicroStream Hackathon Weekly Q&A | Edition 7...

39:48
Helidon DB Client
132 Aufrufe • vor 4 Monaten

47:39
Fast UI Development with RapidClipse (German)

1:10:53
GraalVM: Native Image - Cooking Guide

Get Started with MicroStream

Learn: www.microstream.one

GitHub: https://github.com/microstream-one/microstream

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

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

Free courses at Fast Lane: https://www.microservices.education