MicroStream

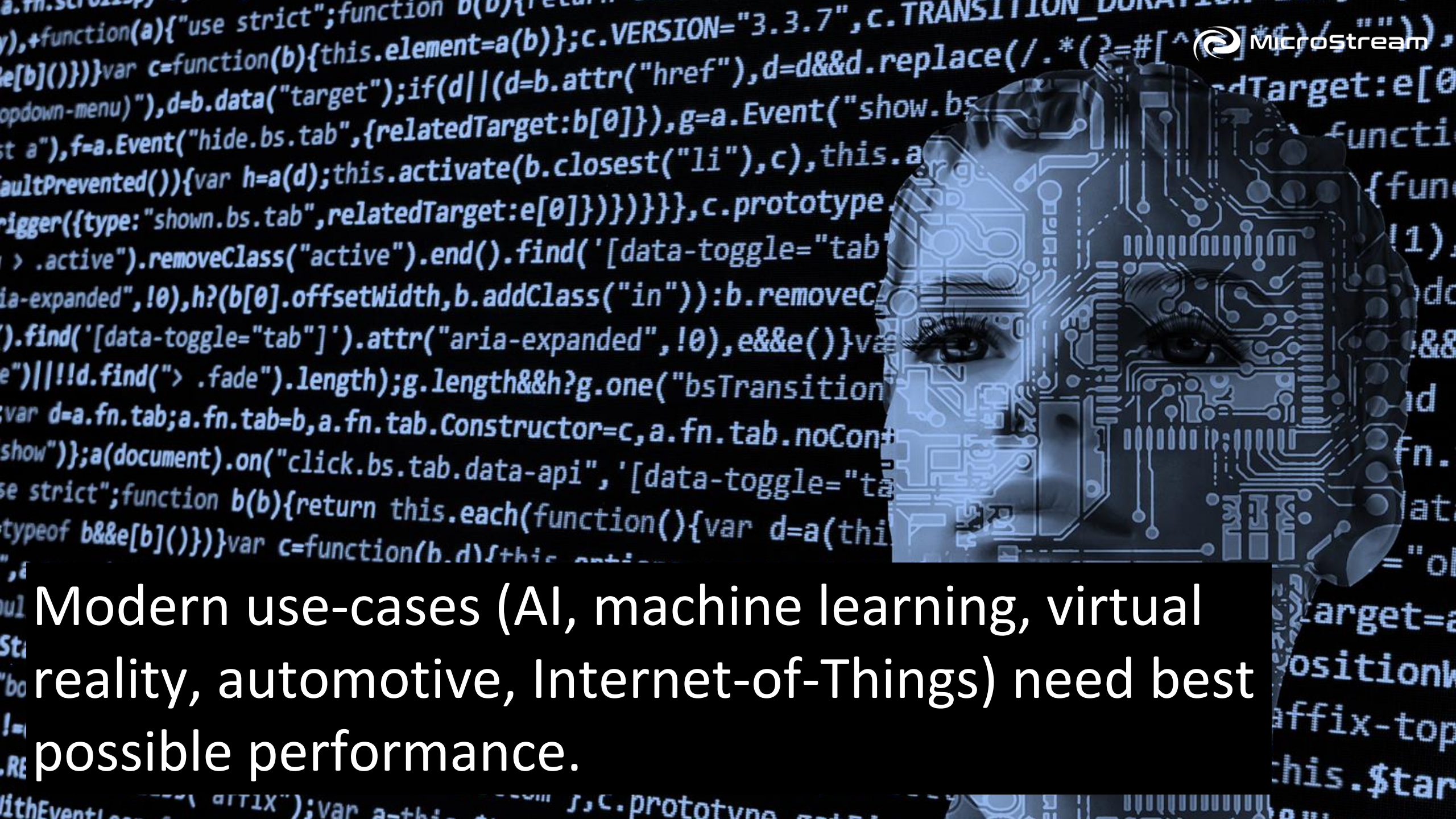
IT-Forum Frankfurt

Next Generation Data Processing & In-Memory Computing with AWS

Markus Kett, CEO MicroStream
eMail: m.kett@microstream.one
Twitter: @MarkusKett
LinkedIn: MarkusKett

MicroStream

Why do we need Performance?



Modern use-cases (AI, machine learning, virtual reality, automotive, Internet-of-Things) need best possible performance.

More and more companies have to collect more and more data, which have to be related, searched and analysed in realtime.

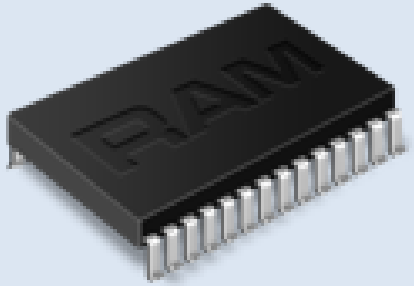
MicroStream

In-Memory Computing

Why In-Memory Computing ?

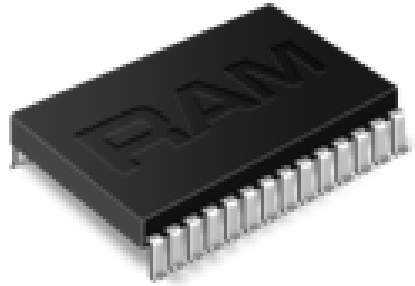
HDD
~ 9 ms
~ 9,000,000 ns

SSD HDD
~ 250 µs
~ 250,000 ns



DRAM
~ 65 ns

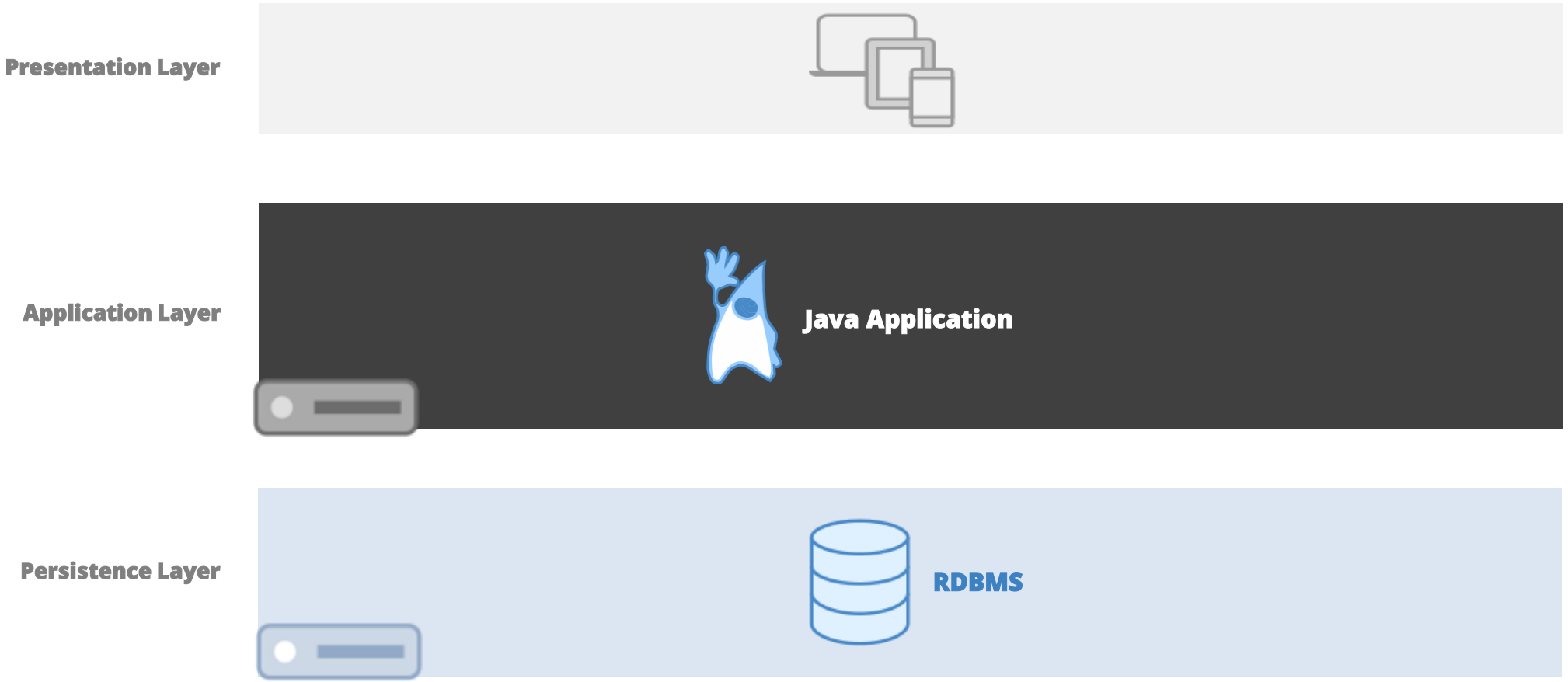
~ 138,000 times faster than HDD
~ 3,840 times faster than SSD

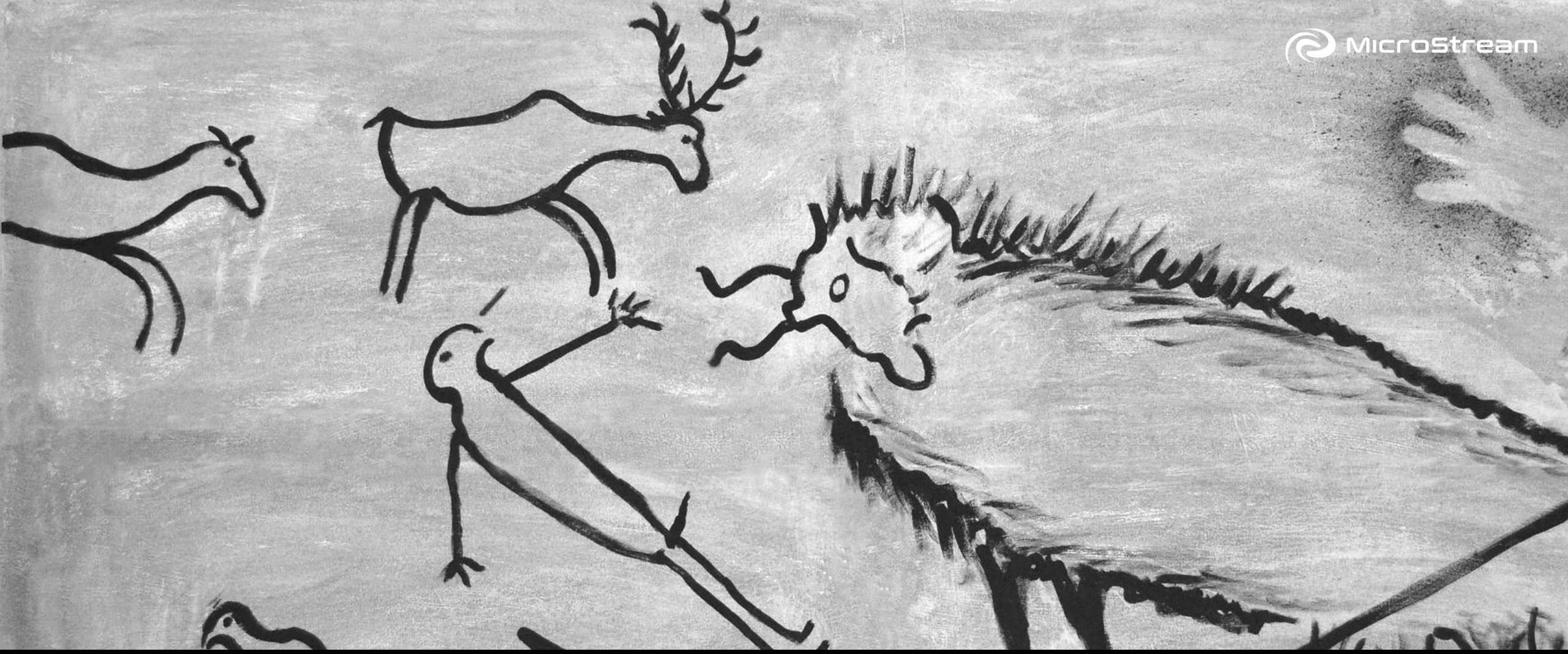


CPU Cache
SRAM < 1 ns

MicroStream

3-Tier Architecture

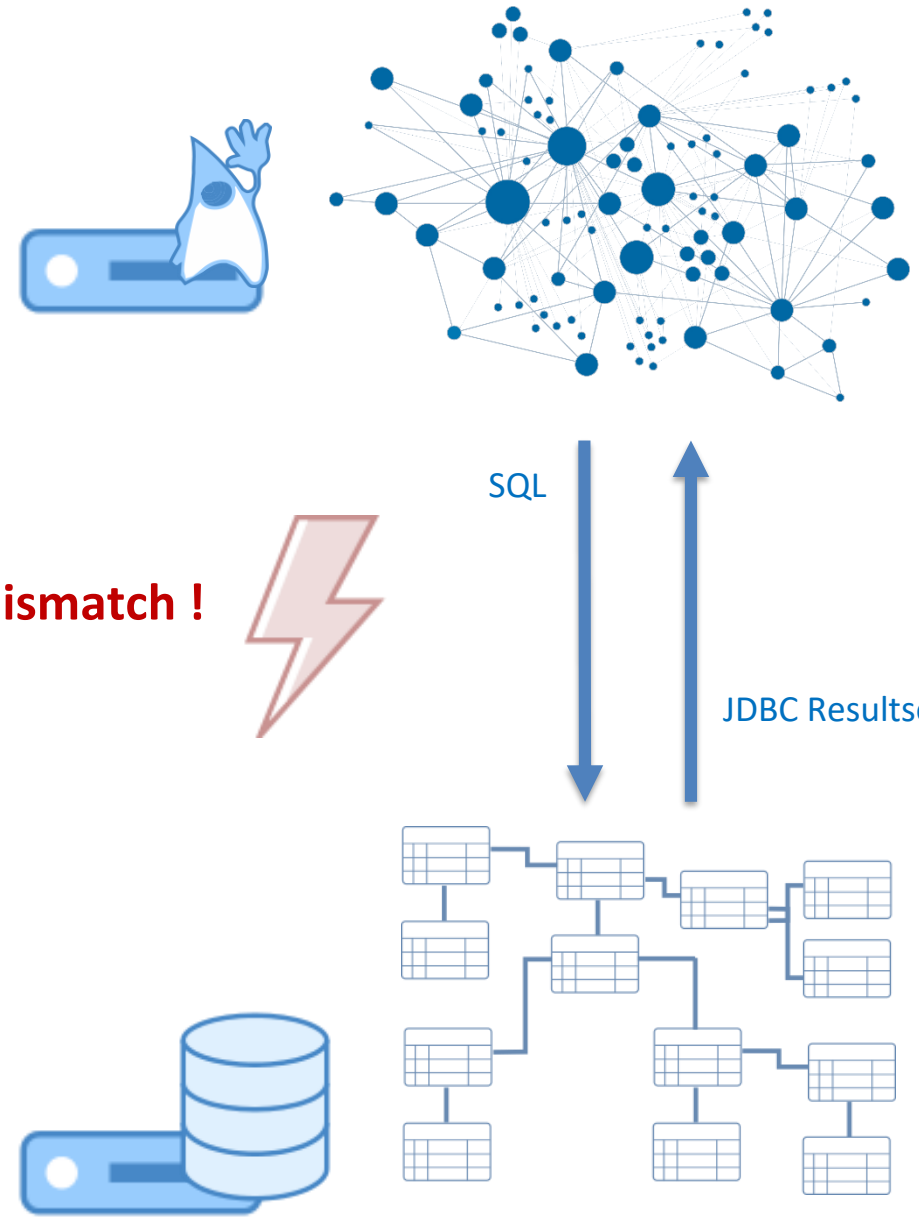




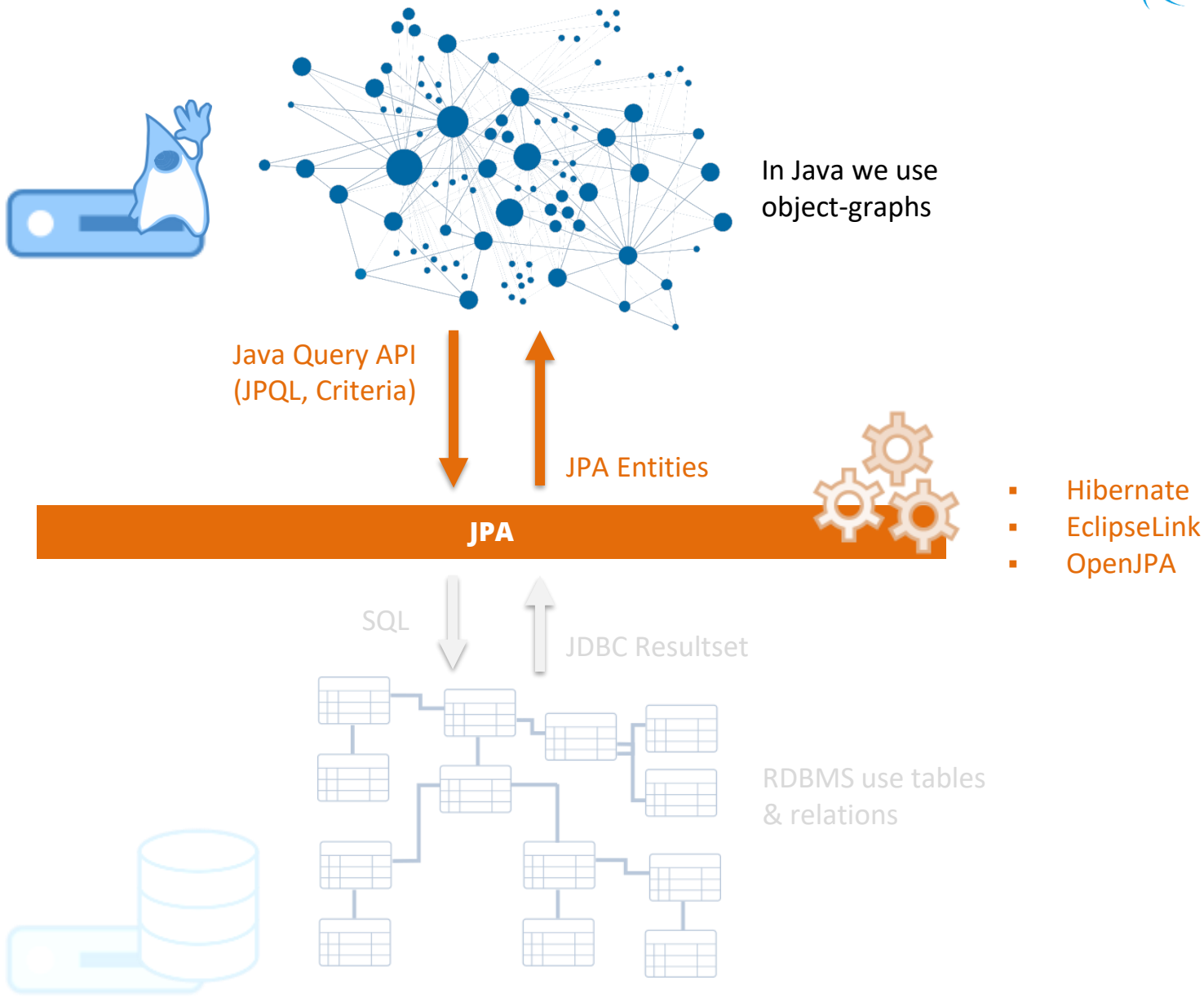
What‘s the problem?

We have to accelerate 1960s technology that we still use today everyw

Problem

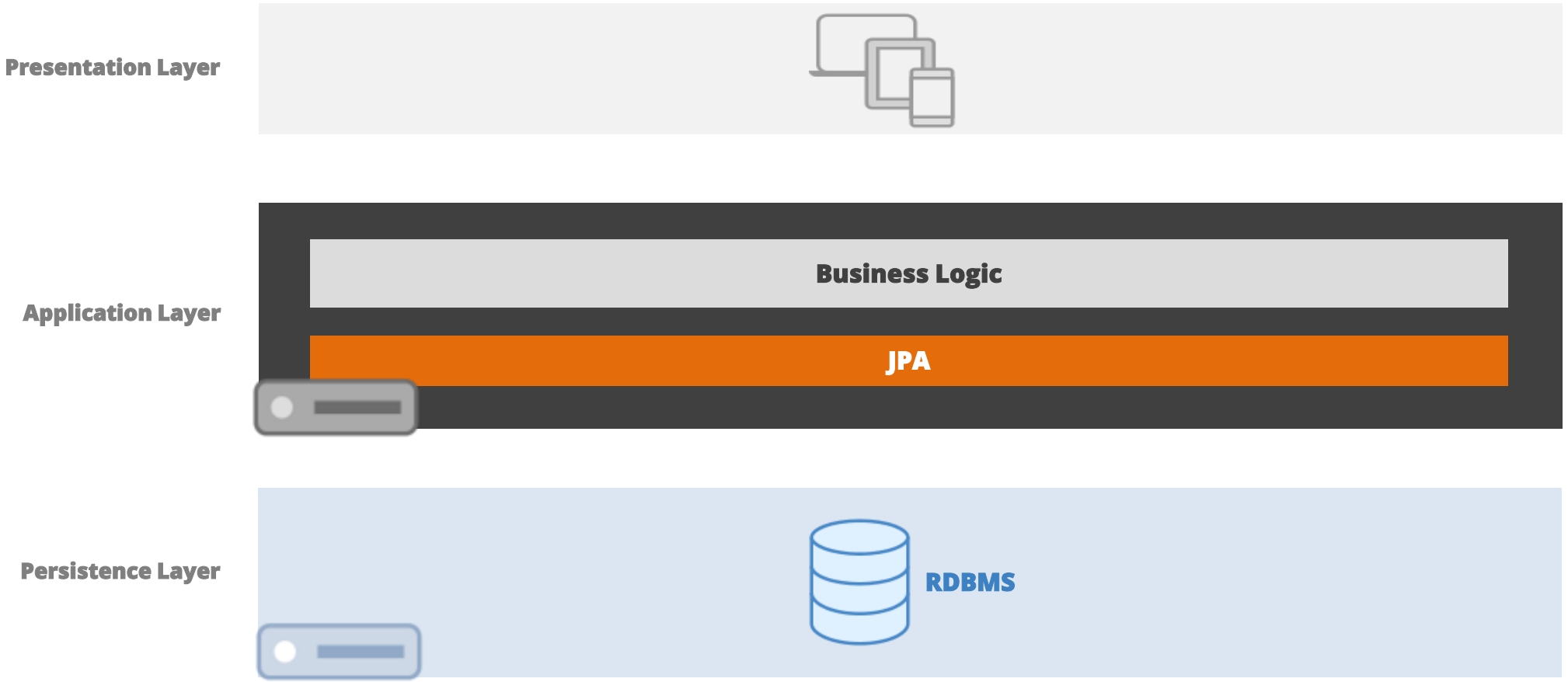

In Java we use
object-graphs
SQL
Impedance Mismatch !
JDBC Resultset
RDBMS use tables
& relations

OR-Mapping



To solve the problem we use OR-mapping. Since 10 years OR-mapping is a standard called JPA. Hiberante is the leading ORM-framework.

JPA Architecture

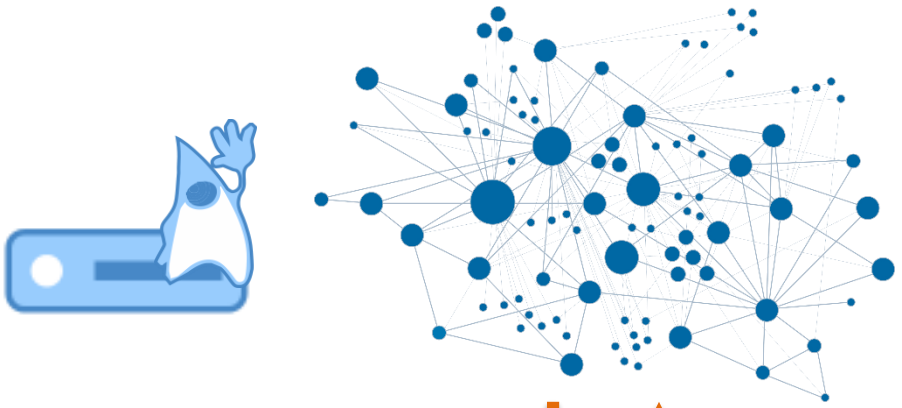


Effort of Development

- 2 data model
  - Java classes
  - Database data model
- ORM Framework
- Data-Cache

Data-Cache

Data Caching



In Java we use object-graphs

Java Query API (JPQL, Criteria)

JPA

Data Cache

- Oracle Coherence
- Memcached
- EHCache

...

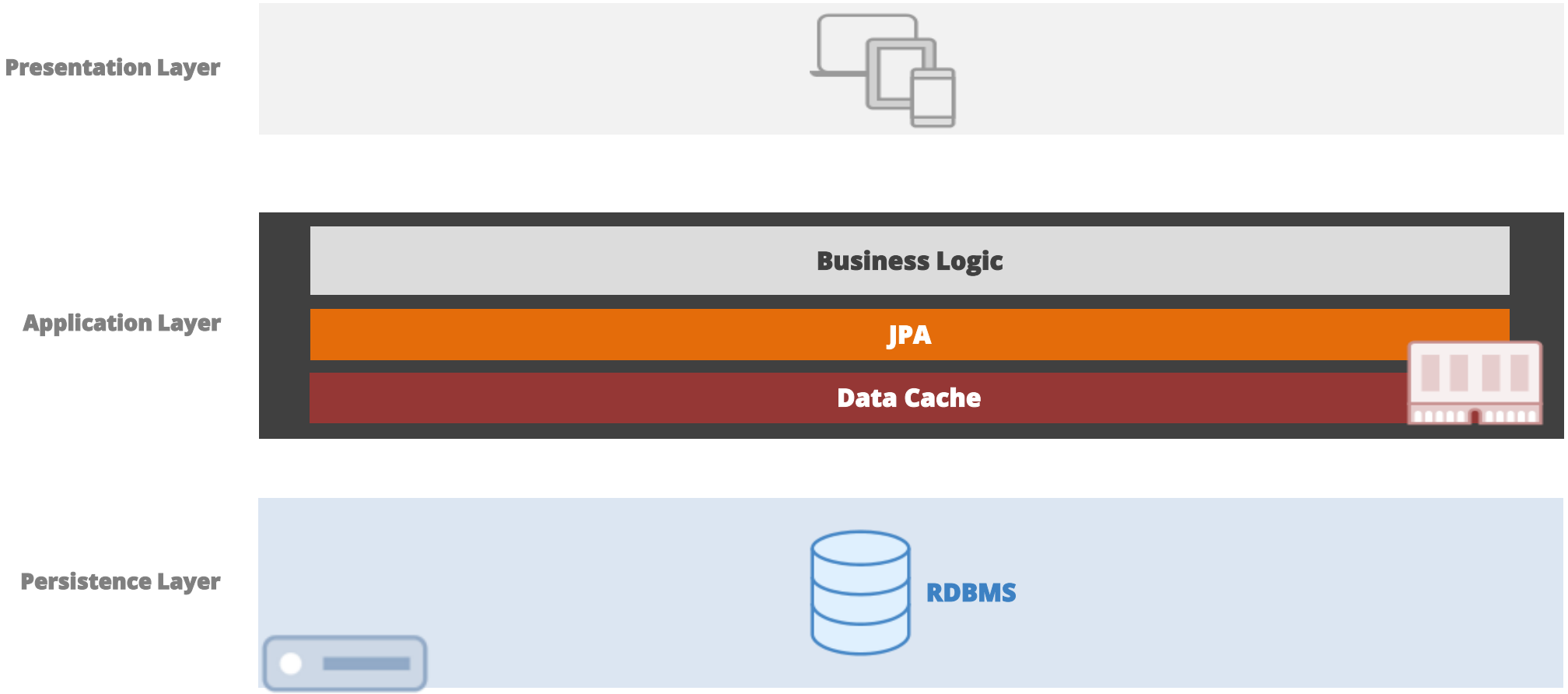
SQL

JDBC Resultset

RDBMS use tables & relations

OR-mapping after every single database access is super expensive and brakes your performance. Thus, in most cases we need an additional data-cache.

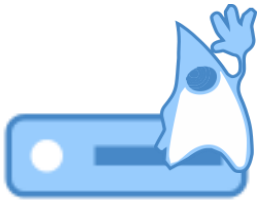
JPA Architecture

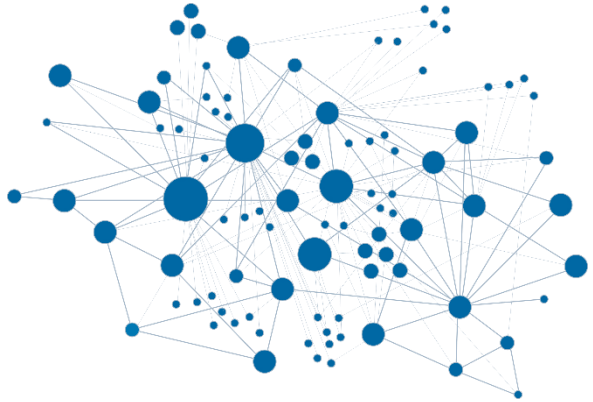


MicroStream

NoSQL

NoSQL



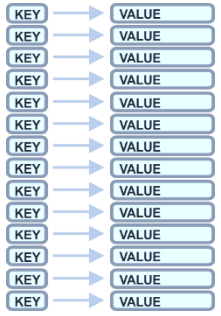


In Java we use object-graphs

SQL, proprietary Query Languages

Impedance Mismatch !

Result



NoSQL DBMS use many different data structure

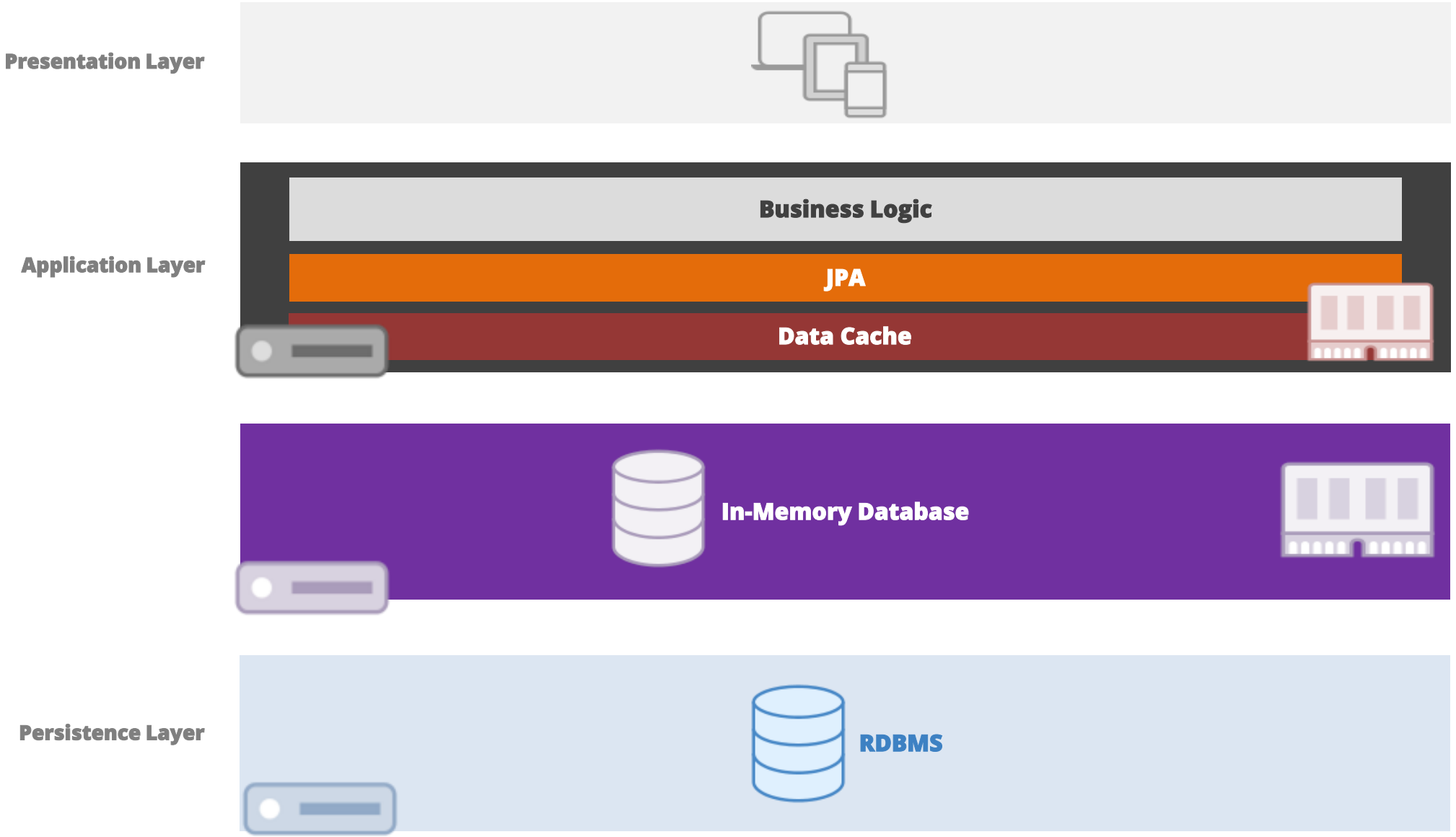
(Key-value, document-, column-, graph-, object-store)

MicroStream

In-Memory Database (IMDB)

MicroStream

In-Memory Database



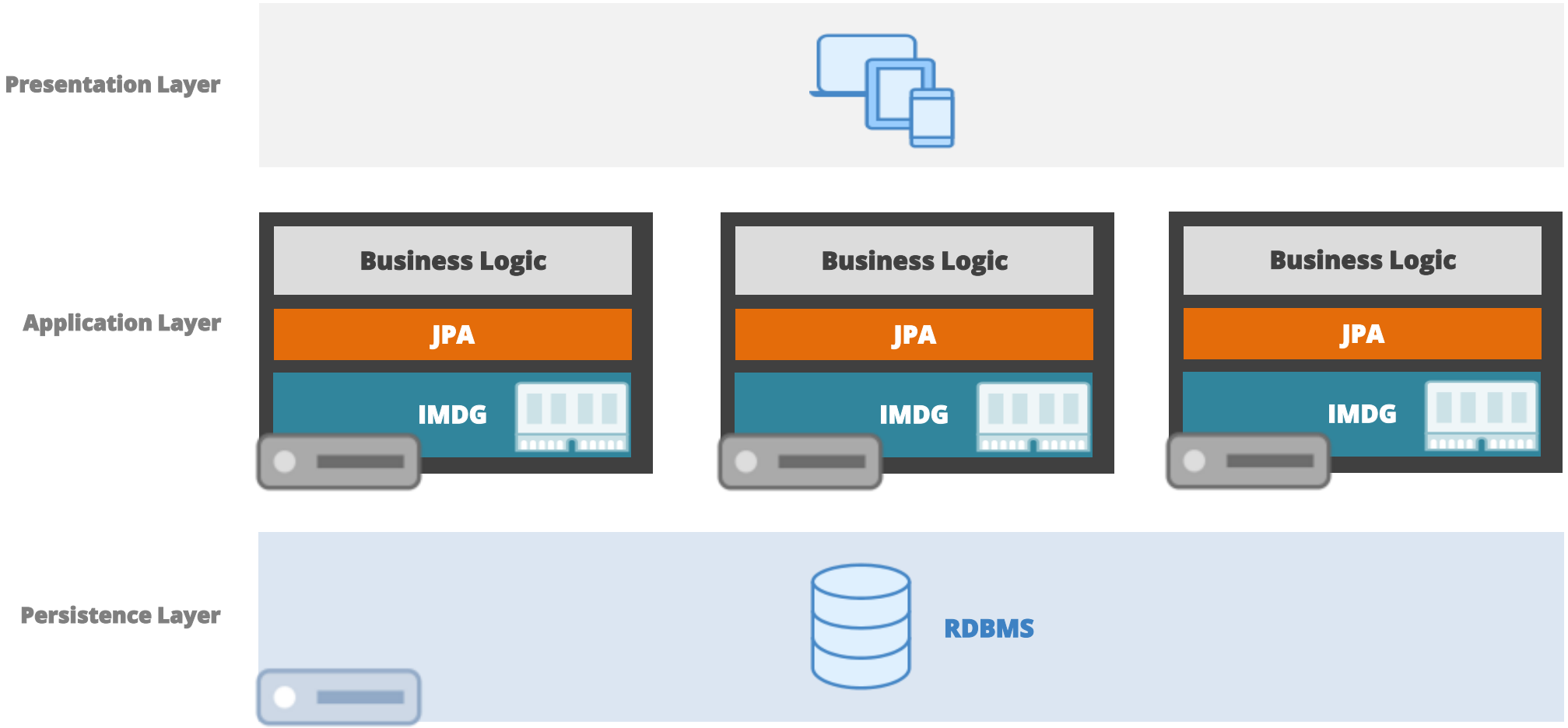
In-Memory Database

- Full support for SQL
- No changes to existing applications
- Familiar administration
- Increasing performance of existing business applications (ERP, CRM, data warehouse, business analytics)

MicroStream

In-Memory Data Grid (IMDG)

In-Memory Data Grid



In-Memory Data Grid

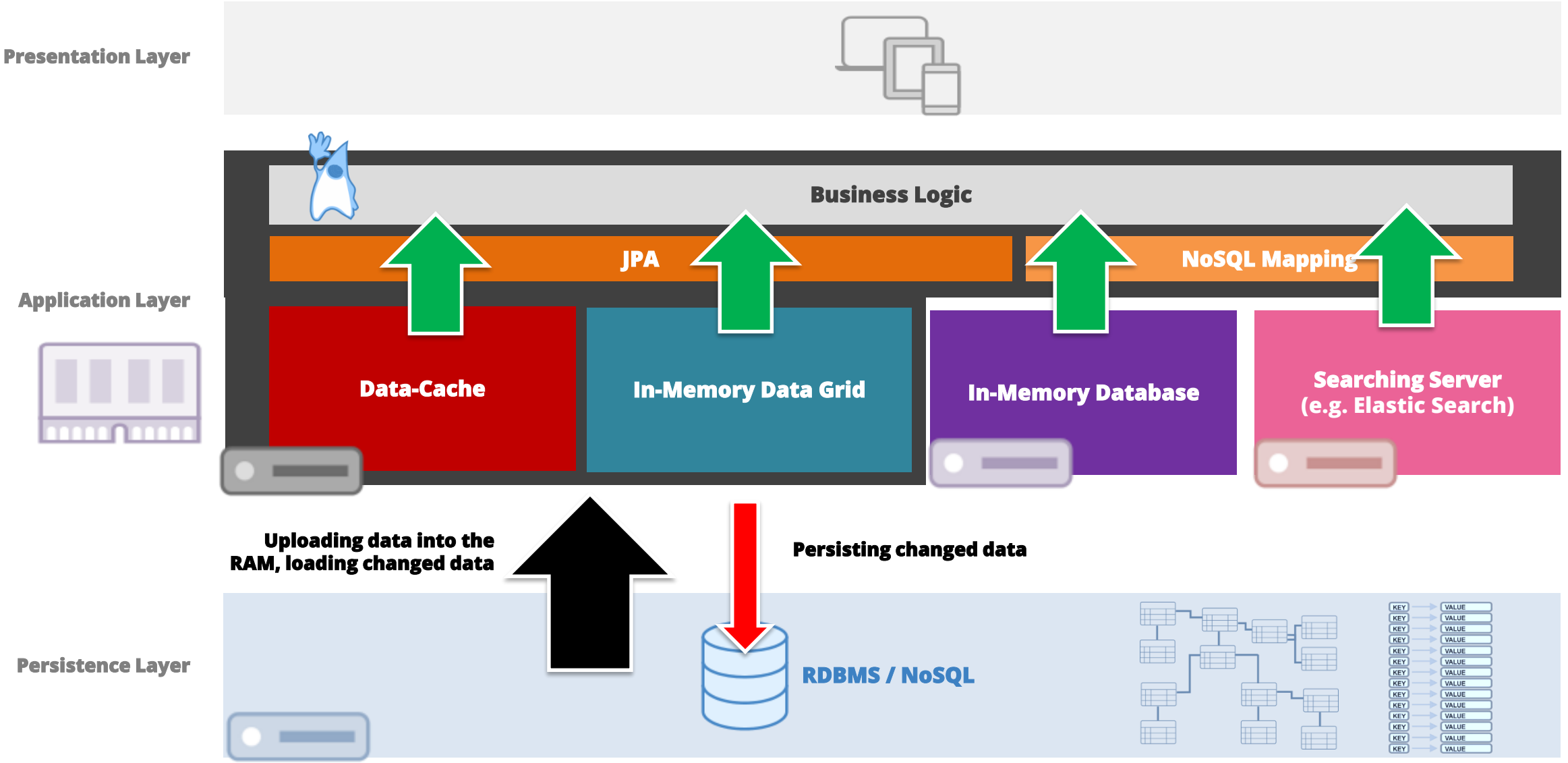
- Very flexible, scaling horizontally hundreds to thousands nodes
- Scale-out is automated
- Support any type of application
- High-availability
- Data structure is key-value
- Persistence via conventional DBMS
- Used for web-apps that require extreme performance and scale
- Real-time big data apps

Situation today …

In-Memory Computing today

- Conventional database-systems are too slow for modern applications
- In-Memory technologies (Data-Caches, IMDBs, IMDGs) become more and more the main data source for our applications
- Conventional DBMS are more and more used as a backup strategy
- Today‘s in-memory technologies
  - use the data structure of the conventional DBMS
  - fit with conventional DBMS
  - Do not fit with object-graphs

In-Memory Computing & Database Applications today



MicroStream

Pure Java In-Memory Computing

Pure Java Application + Java-native Persistence

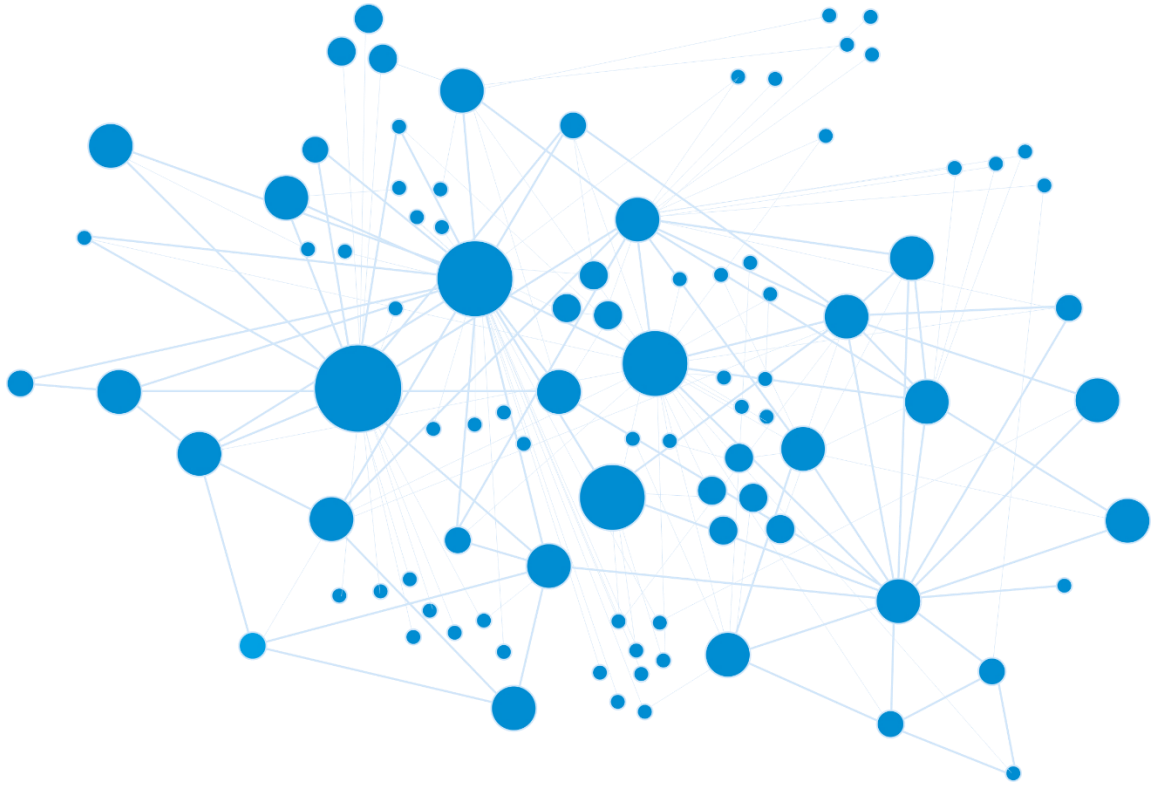
MicroStream

Pure-Java In-Memory Computing

- Integrates into the Java object model
- Consistent architecture and data model
- Low complexity
- 100% pure Java: best fitting data structure, fully objectoriented, completely typesafe, clean code
- Full JVM support (JIT compiler optimization)
- Ultra-fast

Only 1 Data Structure

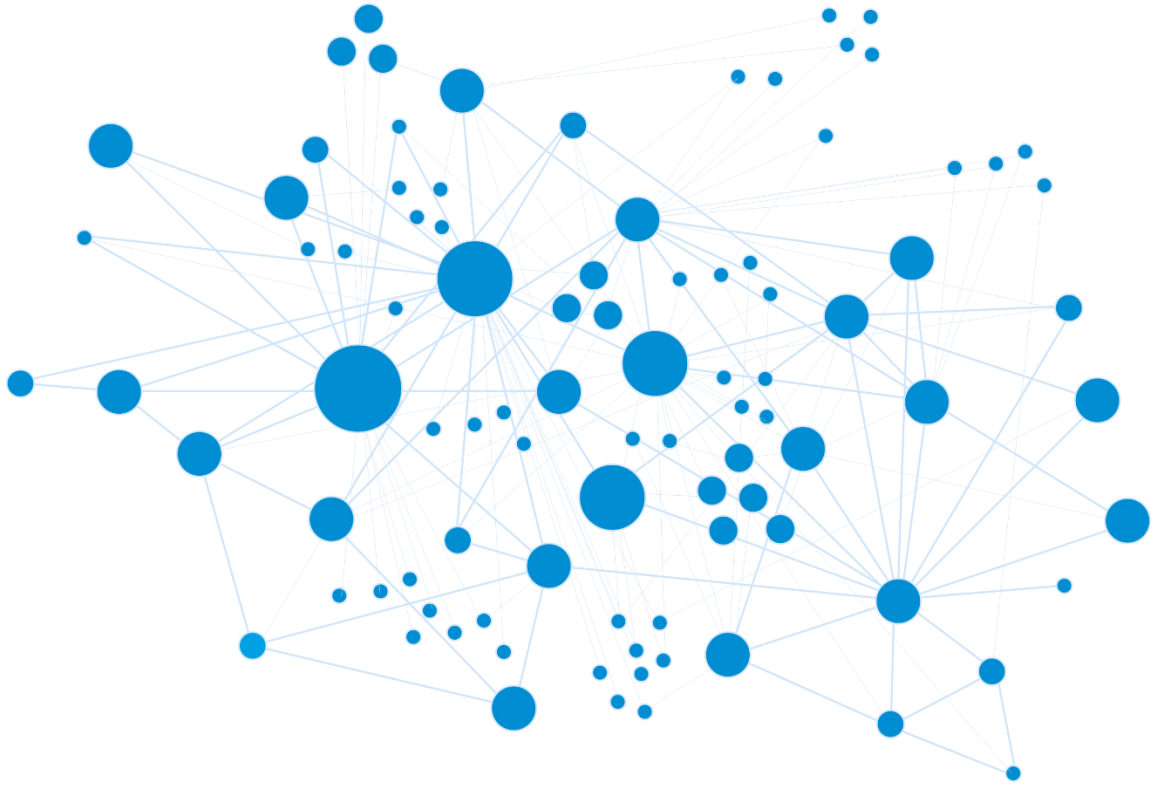
- Java Object-graph
- We use it for everything
- Predestined for complex data structure
- In-Memory
- Fully managed by the JVM



Imagine you would never have to store data, because your Oracle Cloud will never go down – then your Java object-graph would be your database.

Ultra-fast Queries

- Java Streams API
- Typesafe
- Even huge and complex object-graphs can be searched in only microseconds
- No more loss of computing time through a mapping
- No network bottleneck and latencies
- JIT-compiler optimization
- Every query is executed up to 100,000 times faster

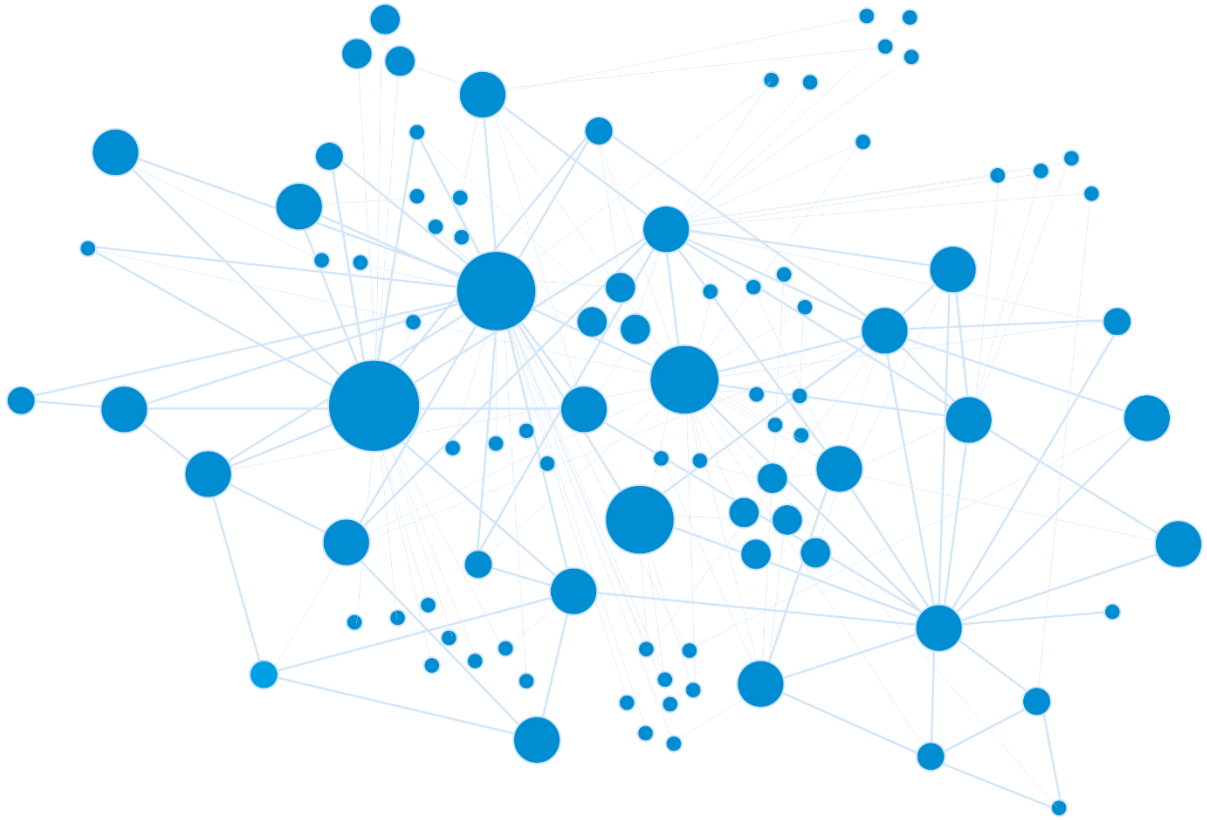


Imagine you would never have to store data, because your Oracle Cloud will never go down – then your Java object-graph would be your database.

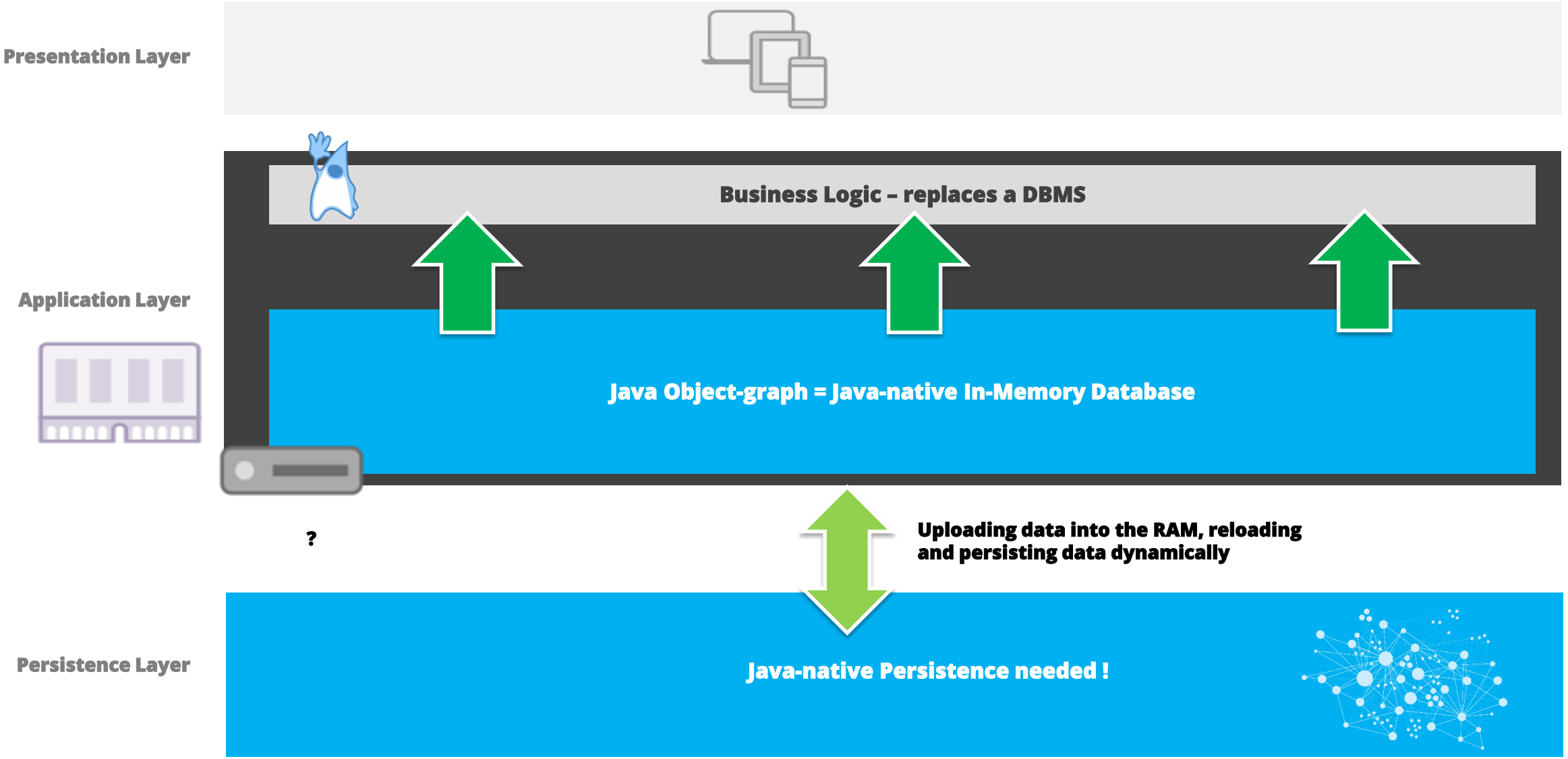
MicroStream

Simplifies your entire development process

- 1 data structure
- 1 data model (Your Java classes only)
- No impedance mismatch
- No mapping needed
- No JPA needed
- No Data-Cache needed
- No need to adjust your classes, no special superclass or interfaces, no specuse your classes as they are
- Freely design of your Java object-model



Pure-Java In-Memory Computing Paradigm



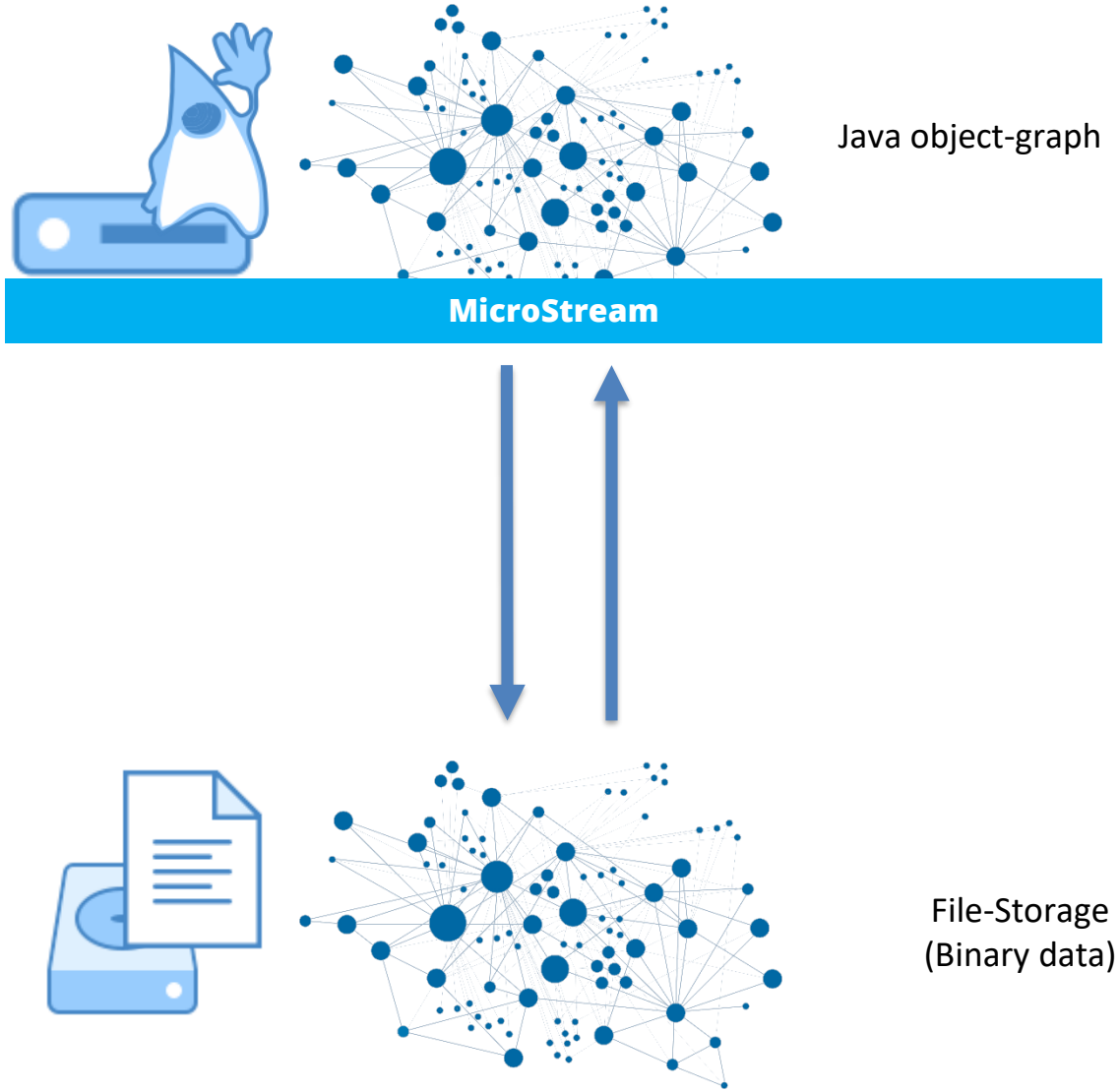
MicroStream

MicroStream - Java-native Persistence

Storing data directly with Java
without the need to use external DBMS
that work totally different to Java.

MicroStream – Java-native Persistence

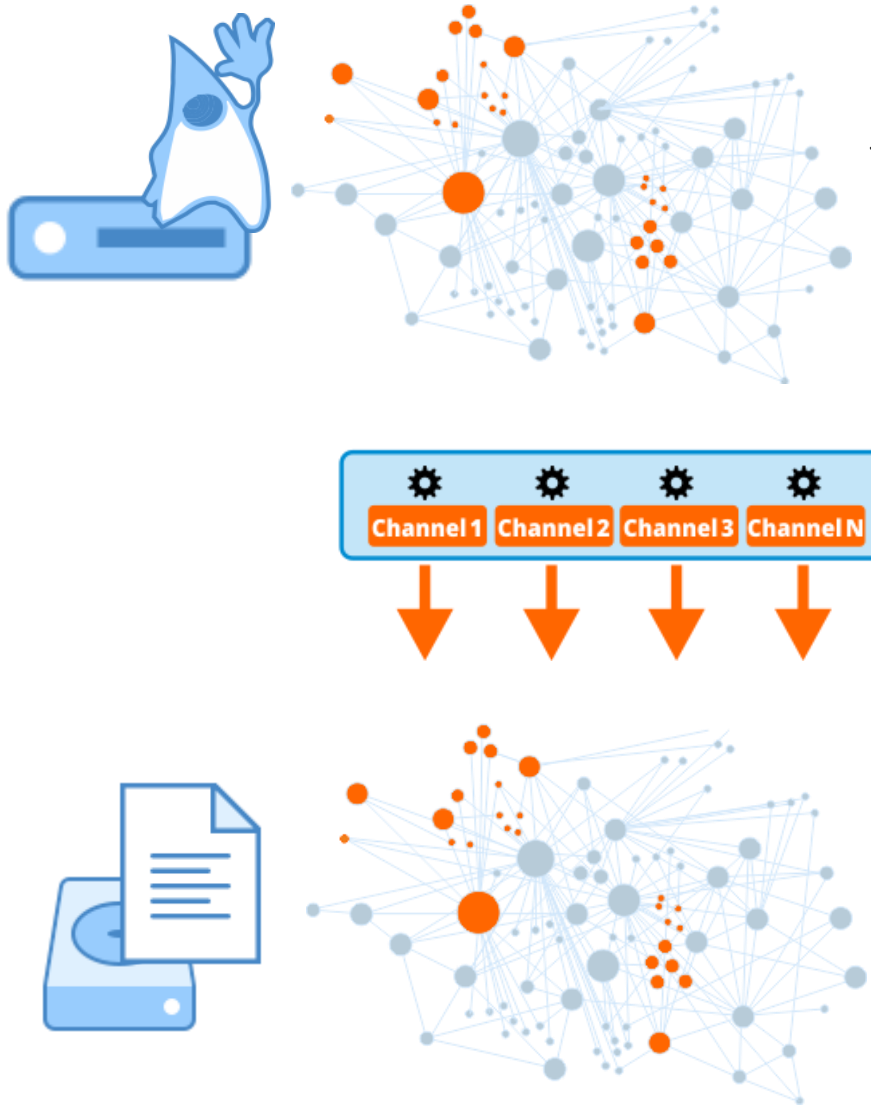
- MicroStream Serialization
- Storing Java object-graphs
- Loading Java object-graphs
- **Revolutionary new: Updating Java object-graphs partially**



MicroStream feels like native Java store method that stores and loads your data fully automatically without any database programming.

Storing data

- CRUD support
- Single objects can be stored like entities
- Append strategy
- Transaction-safe
- Multithreaded



Java object-graph

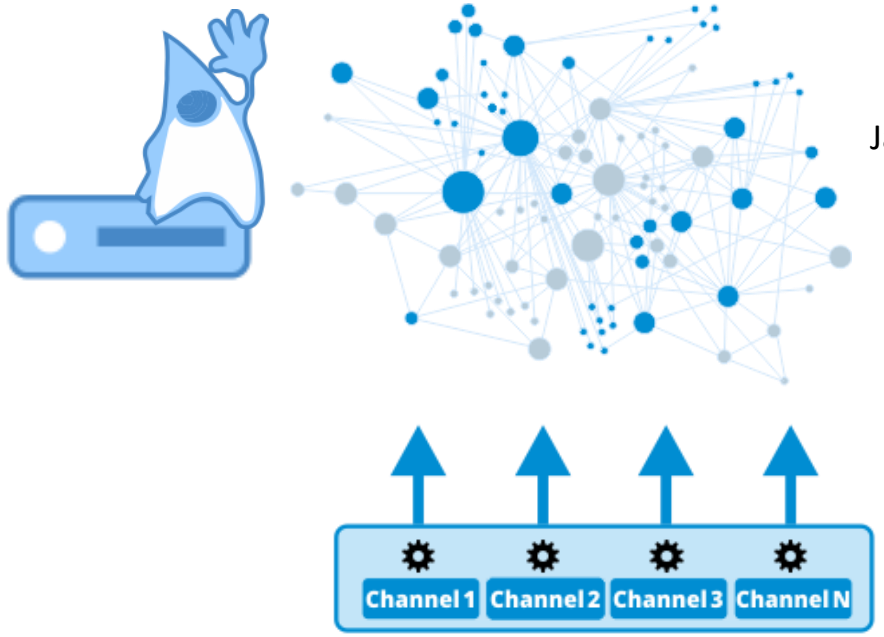
File-Storage
(Binary data)

Storing data example

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

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

Loading data

- If enough RAM available: You can load the entire database into the RAM
- Not enough RAM available: Use lazy-loading to load single references on-demand
- The object-graph in the memory is updated partially
- No object copies
- Multithreaded



Java object-graph

File-Storage
(Binary data)

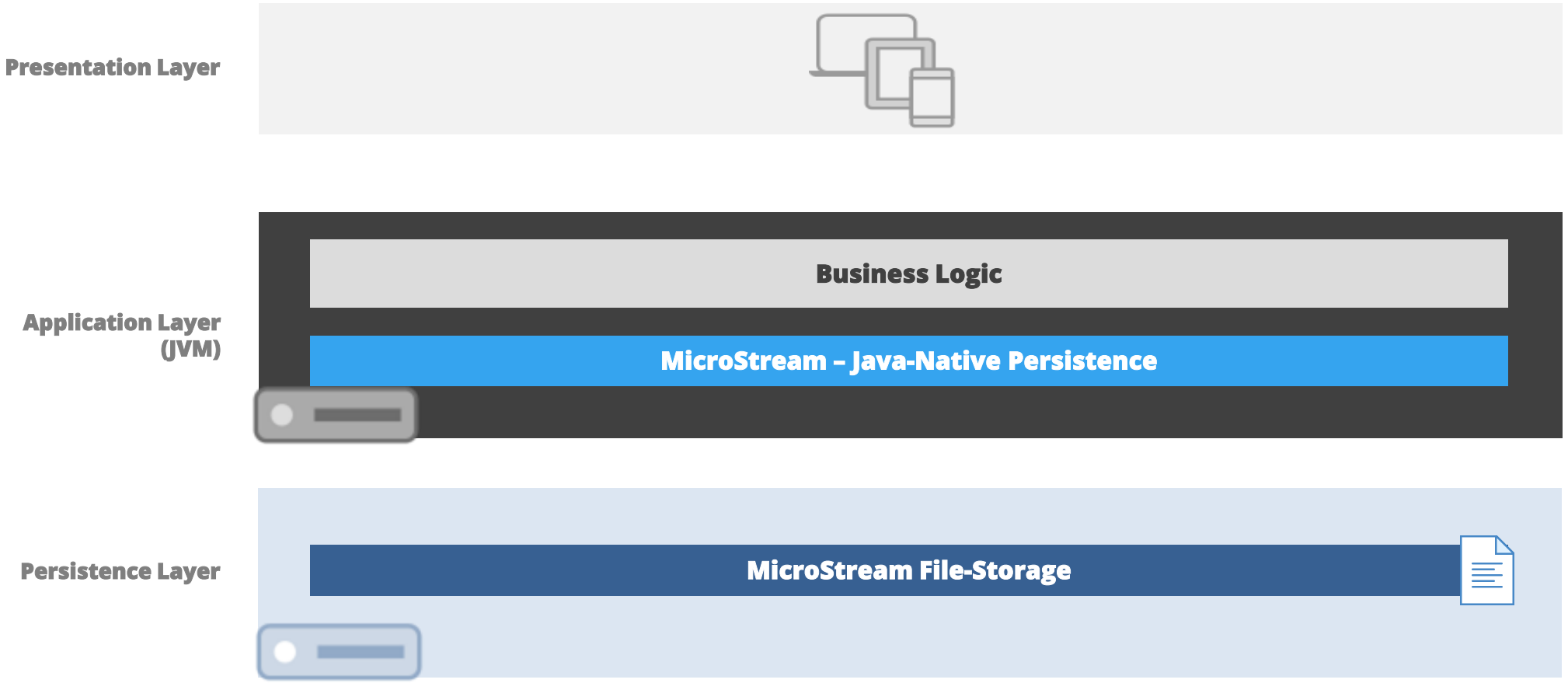
Loading data example

```
public class Customer {
    ...
    private Lazy<Set<Order>> orders;
    ...


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

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

MicroStream core features

- MicroStream Serialization
- Lazy-loading
- File-based garbage collection
- Class versioning & legacy type mapping
- Backup

MicroStream architecture



Easy to use

- Only 1 data structure
- Only 1 data model – your Java classes
- No mapping
- No need to adapt your classes
- No special superclasses
- No special interfaces
- No annotations
- No other internal configurations
- Use your classes as they are

```
public class Customer {

   private String firstname;
   private String lastname;
   private String mail;
   private Integer age;
   private Boolean active;
   private Set<Order> orders;

   ...
```

Use-Cases

Java & JVM languages

Kotlin

Server

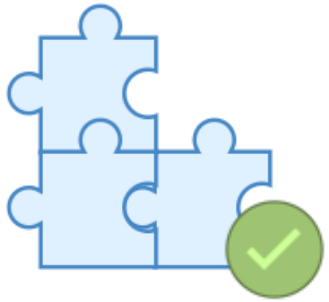
Desktop

Android Mobile

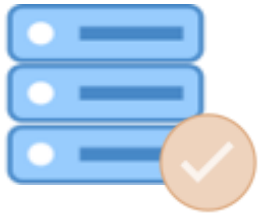
GraalVM

Cloud

Container



Microservices



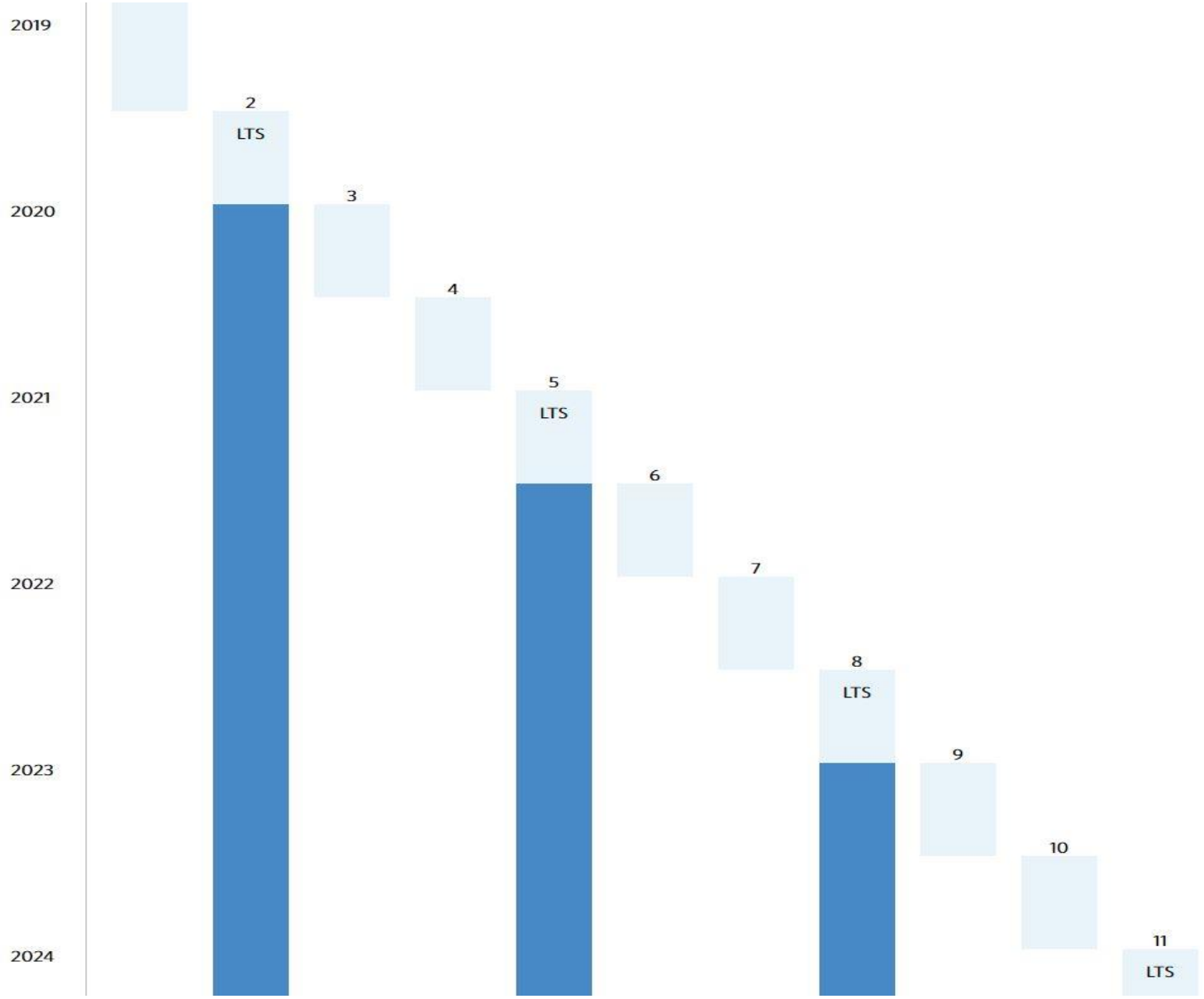
Data-Cache

Note

- You have to be a Java developer
- Your business logic replaces the DBMS
- You have to care for concurrency issues
- No support for SQL
- Admin tools coming soon

Release-Cycle

- MicroStream is free
- New major release every 6 months
- Free support for 6 months
- Long-Term-Support for 8 years
- Next releases
  - **2.0 – 09-2019 LTS**
  - 3.0 – 04-2020
  - 4.0 – 10-2020
  - **5.0 – 04-2021 LTS**



Roadmap

- 12-2019
  - MicroStream for Android
  - MicroStream Layer Entities
- 2020
  - MicroStream Data-Storage Viewer
  - MicroStream Stand-alone Storage-Process
  - MicroStream Cluster
  - MicroStream Concurrency Framework

MicroStream Support

- MicroStream Forum
- StackOverflow
- Professional Support Ticket System
- Videos
- Training
  - Ticket System
  - Flex Support
  - Consulting for consultants

Download free:

MicroStream 2.1 final relaese is now available !

Free support:

Free developer support until 2020 !

www.microstream.one

Twitter: @microstreamOne