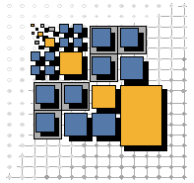


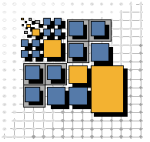
MicroStream vs. JPA
An Empirical Investigation

SummerSoC 2022 in person ☺

Benedikt Full, **Johannes Manner**, Sebastian Böhm and Guido Wirtz

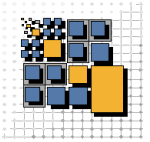
Distributed Systems Group
Otto-Friedrich-University Bamberg, Germany

Agenda



- What is MicroStream? And why is it worth looking at it?
- Methodology
  - Why another benchmark?
  - Experimental Setup
- Results
  - Bookstore Performance Demo (BSPD) Results
  - Wholesale Supplier (WSS) Results
- Discussion
  - MicroStream vs. JPA
  - Concurrency Best Practices
  - Threats to Validity
- Conclusion and Future Work

What is MicroStream? And why is it worth looking at it?



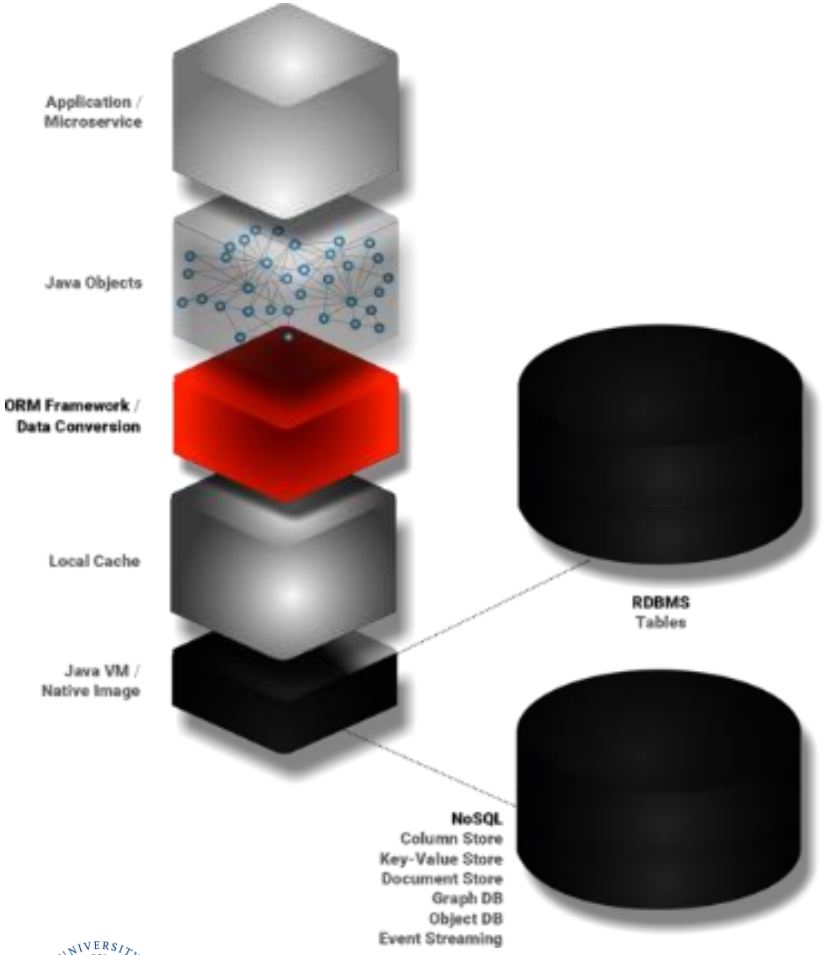
- „MicroStream is a **Java-native object graph persistence engine** for storing […] Java object graph […] and restoring it in RAM at any time by using a **fundamentally new serialization concept** designed from scratch.” (https://microstream.one/platforms/microstream-for-java/)

- Motivation for the paper – why is it worth looking at it:
  „ JVM's in-memory data processing speed + MicroStream **is proven** to be up to 1000x faster than Hibernate + EHCache.
  Minimize latency, maximize throughput & workload, save 90% CPU power & costs.”
  (https://microstream.one/)

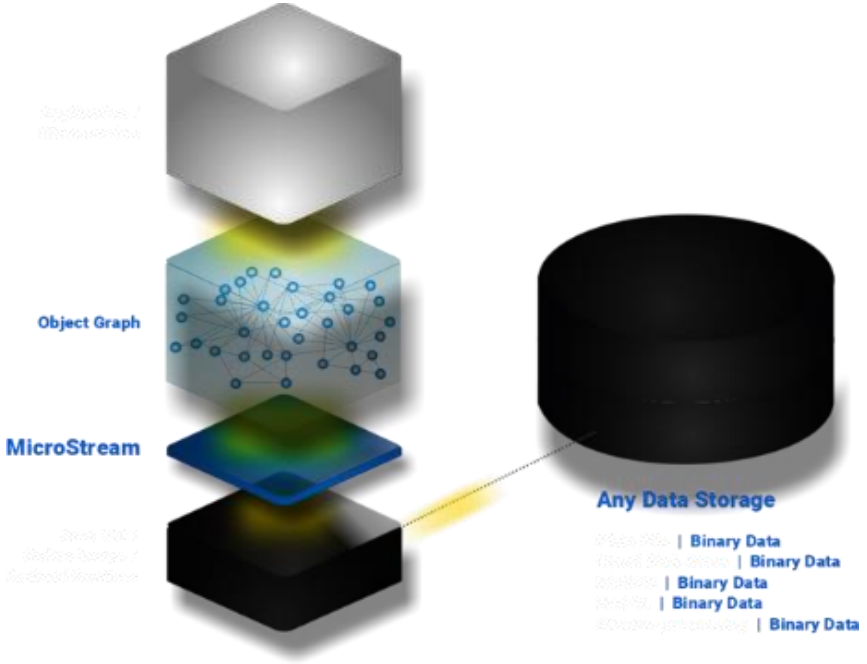
- Problem:
  Only a few (literally no) built-in concurrency mechanisms

TechStack

JPA Solution

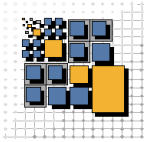


MicroStream Solution



https://microstream.one/platforms/microstream-for-java/

Methodology - Why another benchmark?



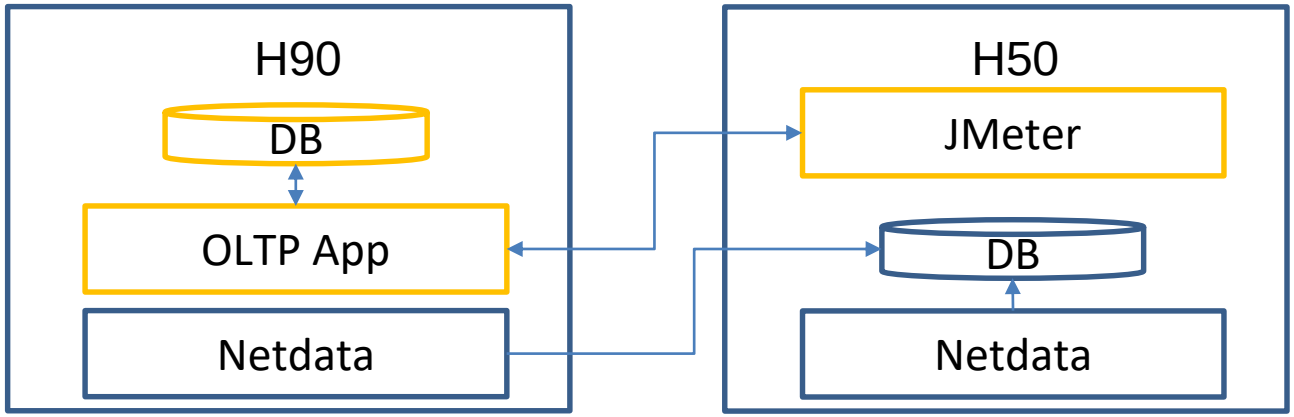
- Application from MicroStream used to repeat the performance claim they stated (Bookstore Performance Demo (BSPD))
- BSPD is vendor-provided and is not standardized (strengthen the positive aspects, hide drawbacks)

→ Solution: Benchmark application based on a specification

- TPC-C specification (1992) -> Wholesale Supplier (WSS) application
  Adapted the specification to an object oriented model

	BSPD (immutable data model)	**WSS** (mutable data model)
JPA	Java™	Java™
MS JACIS		Java™
MS Sync	Java™	Java™

Experimental Setup



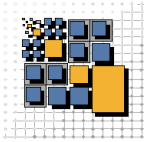
Ubuntu 20.04 server image on H90 & H50

Fujitsu Esprimo P757
Intel i7-7700, 4 cores
210 GFLOPS peak
32 GB RAM
256 GB SSD as primary device

Fujitsu Esprimo P700
Intel i7-2600, 4 cores
92 GFLOPS peak
16 GB RAM
240 GB SSD as primary device

- „Isolate“ OLTP app as much as possible performance-wise (not introducing side effects)
- JMeter for generating the load (10 users in parallel doing sequential work)
- Measured user perceived performance and server processing time

Results – BSPD/WSS



JPA/MS	#Requests	Median (in ms)	Speed-up
[BSPD1]	6931/8380	68.12 / 2.84	**24.03**
[BSPD2]	6935/8383	3.64 / 0.91	**3.99**
[BSPD3]	6934/8382	7.87 / 0.93	**8.42**
[BSPD4]	6936/8385	2.8 / 0.12	**23.3**
[BSPD5]	6931/8376	38.24 / 14.59	**2.62**
[BSPD6]	6929/8376	305.06 / 0.72	**426.61**
[BSPD7]	6933/8381	3.26 / 1.11	**2.93**

BSPD6: Getting purchases of foreigners (customer:purchase – 1:N, fetch type LAZY)

JPA / MS-JACIS / MS-Sync	[WSS1] (479/479/479)	[WSS2] (478/478/479)	[WSS3] (5388/5386/5390)	[WSS4] (5147/5145/5148)	[WSS5] (479/478/479)
Median	8.65 / 87.8 / 0.03 **10.15** / **264.3** 87 / 162 / 76 **1.86** / **1.14**	36.77 / 148.25 / 2.03 **4.03** / **18.14** 115 / 223 / 78 **1.94** / **1.47**	21.72 / 35.16 / 4.94 **1.62** / **4.39** 100 / 109 / 80 **1.09** / **1.25**	10.41 / 9.96 / 5.22 **1.05** / **1.99** 89 / 84 / 81 **1.06** / **1.1**	60.44 / 152.12 / 21.52 **2.52** / **2.81** 138 / 226 / 97 **1.64** / **1.42**

Results WSS

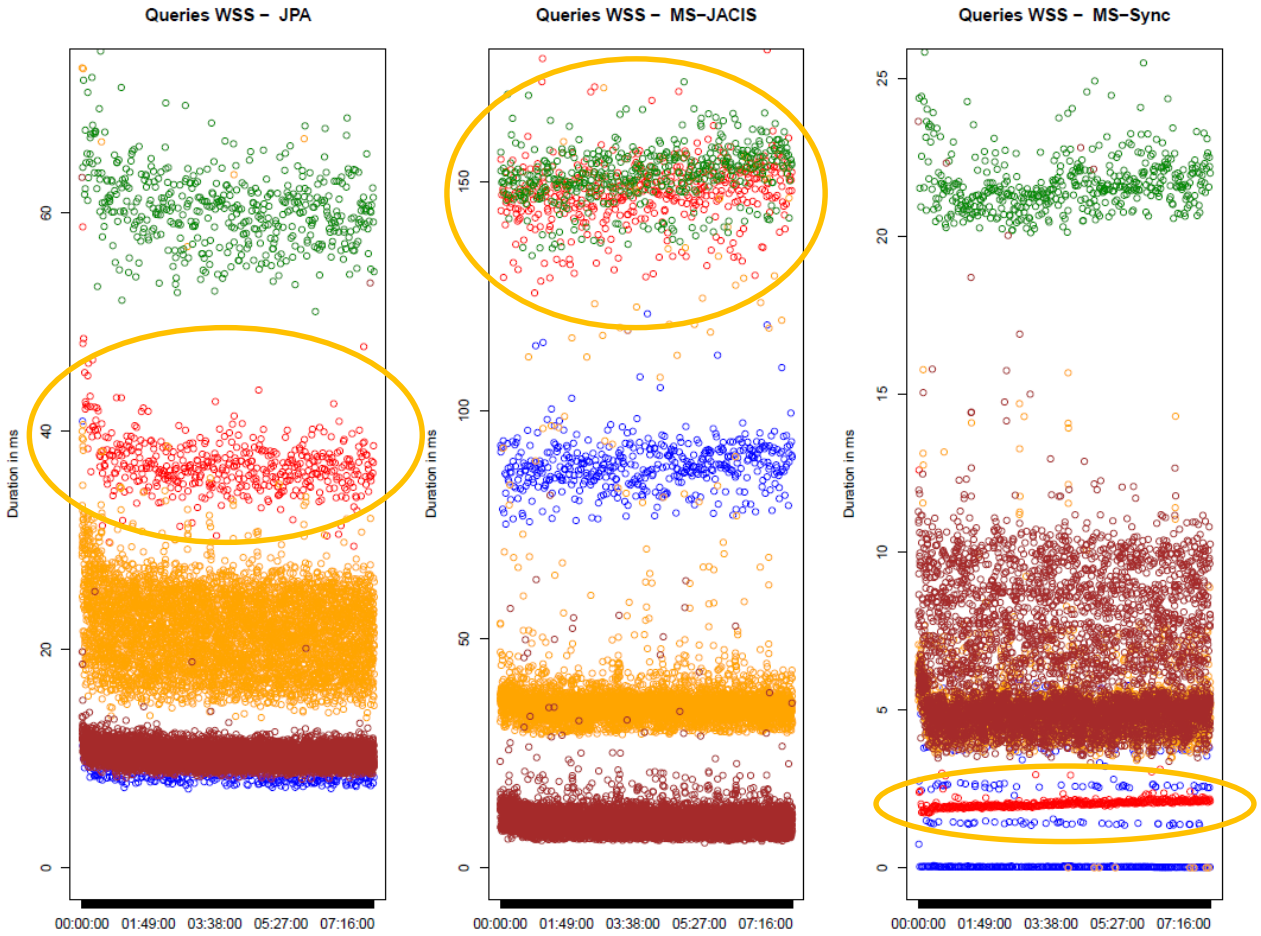
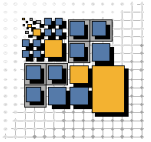


Fig. 4. Wholesale Supplier business transactions: *Order-Status* (blue), *Stock-Level* (red), *New-Order* (orange), *Delivery* (green), and *Payment* (brown).

Research Questions

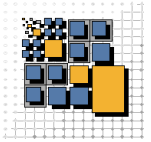


1. Is a MicroStream-based solution up to a thousand times faster than a comparable JPA-based implementation utilizing Hibernate?

2. How can we achieve concurrency control for a mutable data model with the MicroStream in-memory data engine?

3. What are potential usage scenarios where MicroStream-based persistence should be used instead of JPA-based persistence?

RQ1: MicroStream vs. JPA



„MicroStream **is proven** to be up to 1000x faster than Hibernate"
(https://microstream.one/)

- BSPD6 experienced the most significant speed-up (**427 times faster**), BUT nature of this query was crucial – complex joins in JPA case)
- In other BSPD cases MS is by factors of tens faster than JPA

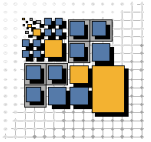
- MS + JACIS (currently the project having a MS adapter for transaction handling) is **8-11 times slower compared to JPA**
- MS Sync is indeed **also up to 264 times faster** only for WSS1
- Other WSS cases, also by factors of tens not thousands

User perceived performance more important

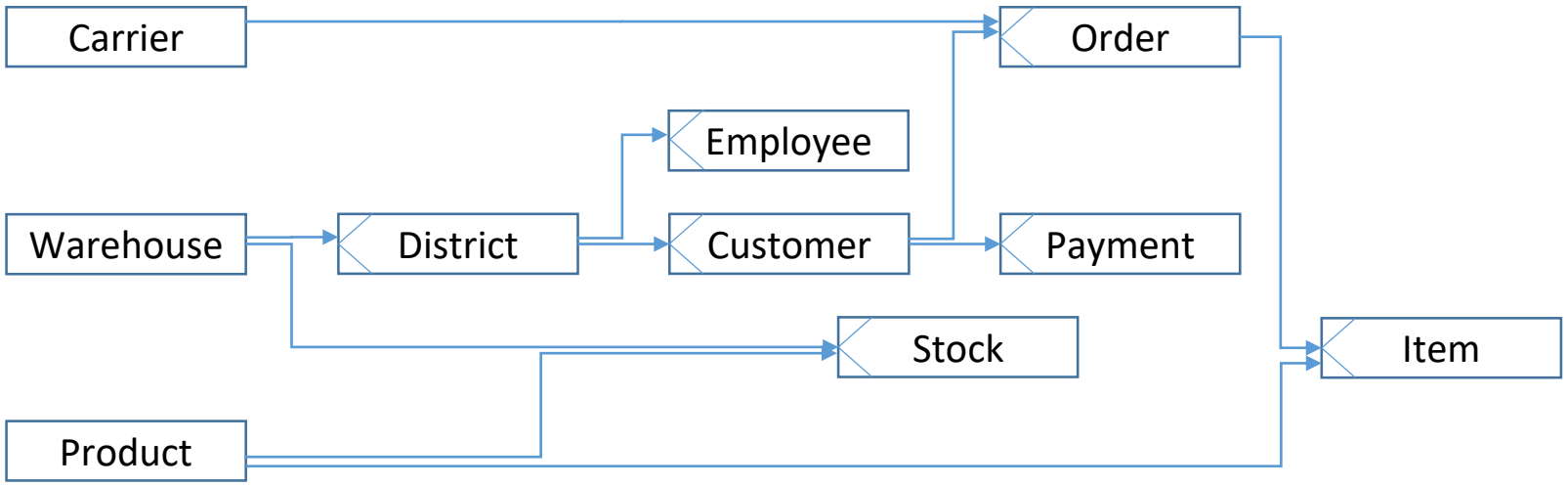
→ Scheduling, network latency has a major influence

→ In this use case, MS Sync is 10% to 47% faster than JPA based solution

RQ2: Concurrency Control



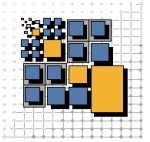
- Structured Entity Relationsship Model (SERM) for modelling the domain



→ Nesting of the locks is derived from the domain model

→ Helps avoiding deadlocks

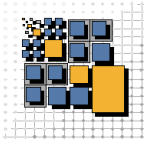
RQ2: Concurrency Control



- Dedicated collection locks for performing a number of actions on a set of instances of the same class
- Only a single write operation is possible at a time (design of MS)
- Synchronized blocks help in especially reading consistent data

```
// method for updating order status and customers
public void deliverOldestOrders ( . . . oldestOrders , . . . ) {
    synchronized ( this.storageManger ) {
        for (OrderData order : oldestOrders ) {
            . . .
            synchronized ( customer.getId ( ) ) {
                synchronized ( order.getId ( ) ) {
                    . . .
        this.storageManger.storeRoot ( ) ;
    }
}
```

RQ3: Usage Scenarios



- MS is especially suited for “Micro persistence for microservices & serverless Java functions” (https://microstream.one/)
- Decentralized data management principle for microservices

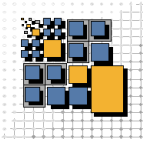
Good use cases

- Mostly immutable data models
- Java/Cloud-native mircoservices

Where we see need for improvement

- Concurrency control
  - JACIS introduces performance shortcomes
  - Low level synchronization is error-prone
- Adapters for mature storage solutions
  (currently only BLOB storage of object graph)

Conclusion



MS is an alternative to Hibernate and can boost performance.

Threads to Validity

- No Lazy references for MS (all data were in RAM)
- Custom benchmark application (WSS) – not 100% objective
- Used experimental setup

Future Work

- Compare MS with other in-memory database solutions like Redis
- Bottleneck detection tool for looking at the specific hardware configuration

Johannes Manner
M.Sc. Applied Computer Sciences

johannes.manner@uni-bamberg.de

/johannes-manner

/profile/Johannes-Manner

Institution and department
Otto-Friedrich-Universität Bamberg
Department of Applied Computer Sciences
Distributed Systems Group