



Ultra-Low Latency with Java and Terabytes of Data

Jfokus, Stockholm

▶SEARCH▶TR/01▶03▶SEARCH▶TR/01▶03
▶SEARCH▶TR/01▶03▶SEARCH▶TR/01▶03

▶RS/0211 SEARCH▶R01/0211 SEARCH...001
▶RS/0211 SEARCH▶R01/0211 SEARCH...001

▶TR/010N▶TR/01▶03▶010N▶TR/01▶03
▶TR/010N▶TR/01▶03▶010N▶TR/01▶03

▶TR/01▶03
▶TR/01▶03

▶TR/01▶03
▶TR/01▶03



```
SEARCH
SEARCH
▶TR/01▶03▶
//SYS:ONLINE
ENTER N01
SEARCH
SEARCH
▶TR/01▶03▶
//SYS:ONLINE
ENTER N01
```

▶SEARCH▶TR/01▶03▶SEARCH▶TR/01▶03
▶SEARCH▶TR/01▶03▶SEARCH▶TR/01▶03

▶RS/011
▶RS/011

▶RS/011
▶RS/011

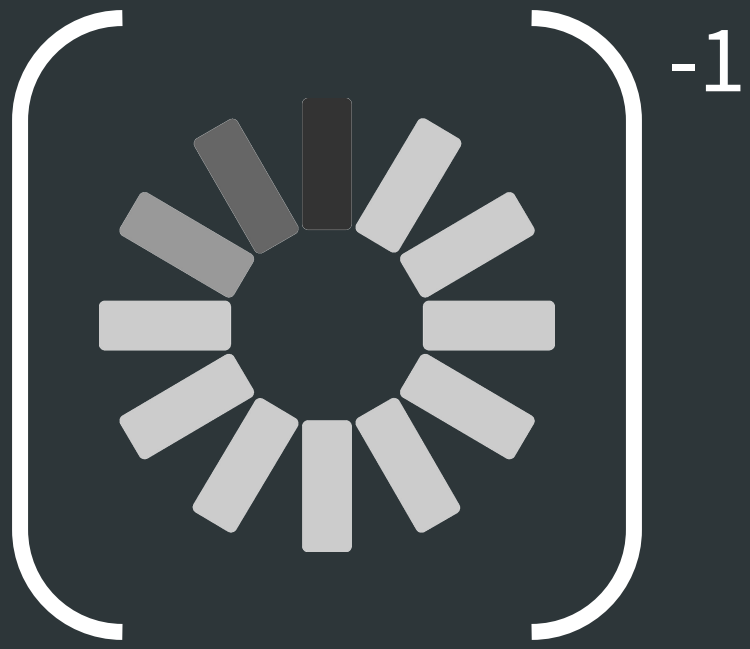
▶RS/0211TR /ON▶RS/0211TR /ON

The Presenter



Per Minborg

Inventor, Serial Entrepreneur, and Java Enthusiast. Well known blogger and alumni speaker at events like JavaOne, DevNexus and Oracle Code. Co-author of the publication “Modern Java” and writer in Oracle Java Magazine.



Ultra-Low Latency = 200 ns

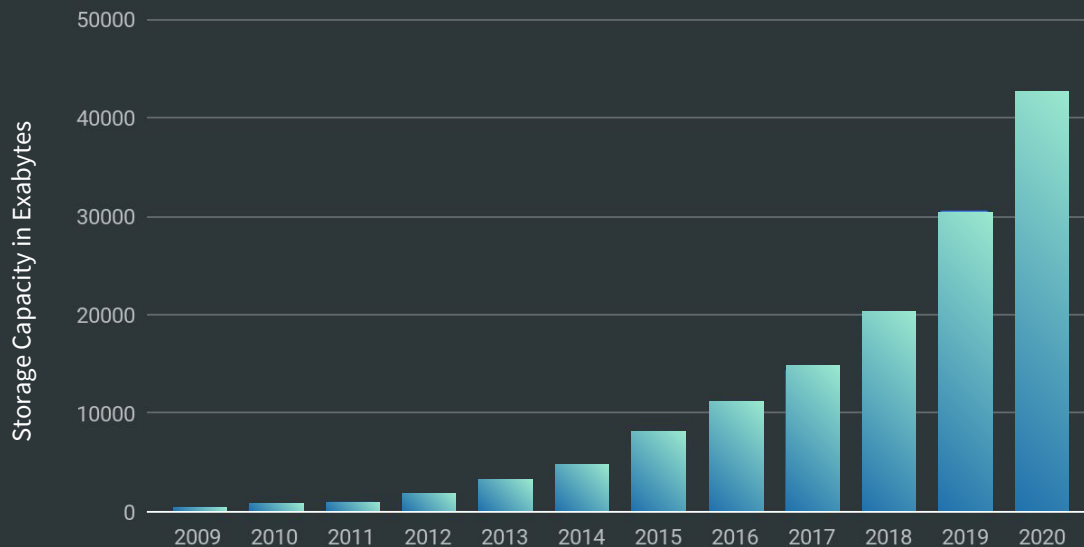


Why Are Applications Slow?

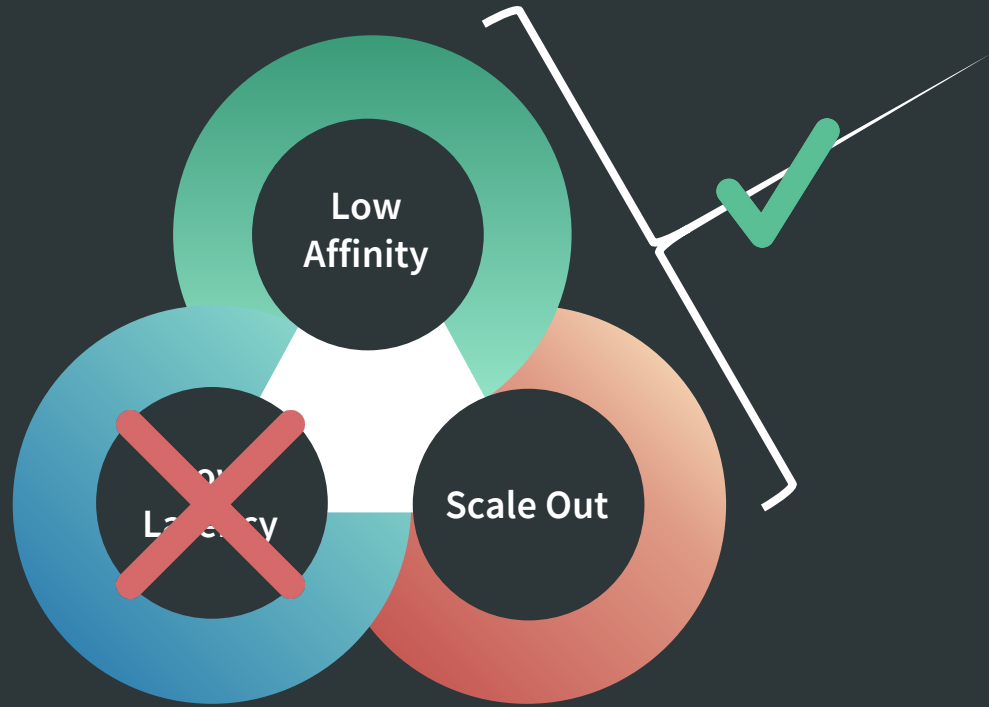
1. Slow Databases
2. Data on Several Nodes and no Affinity Across Data
3. Data is Remote
4. Unnecessary Object Creation / Garbage Collect Problem
5. Lack of Parallelism

Slow Databases

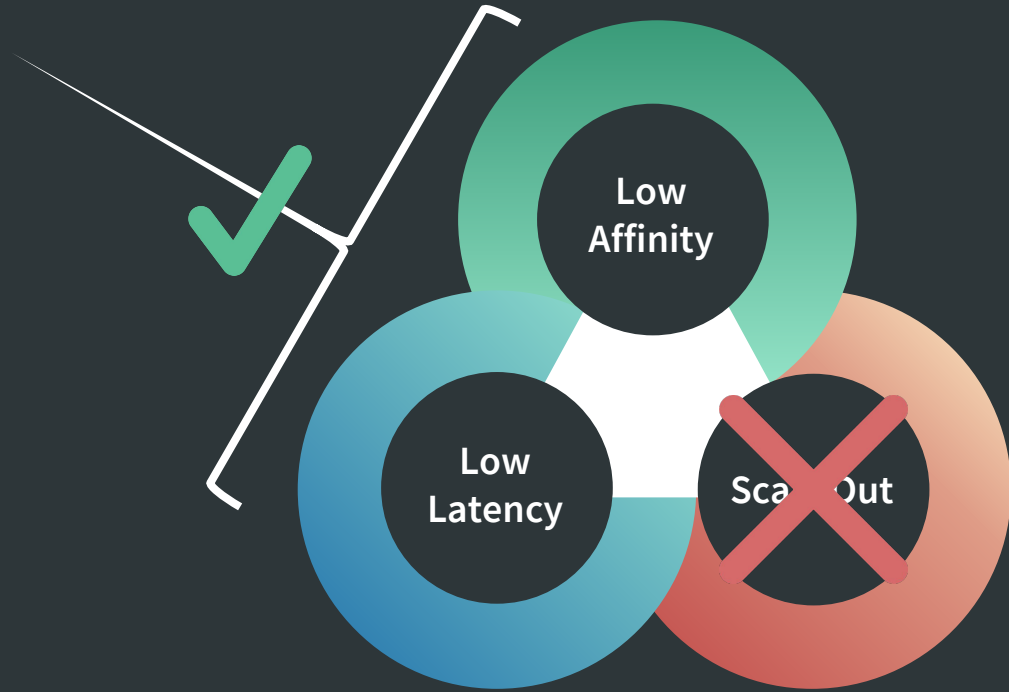
Data grows exponentially, which clogs systems



Several Nodes/no Affinity Across Data



Several Nodes/no Affinity Across Data



Several Nodes/no Affinity Across Data



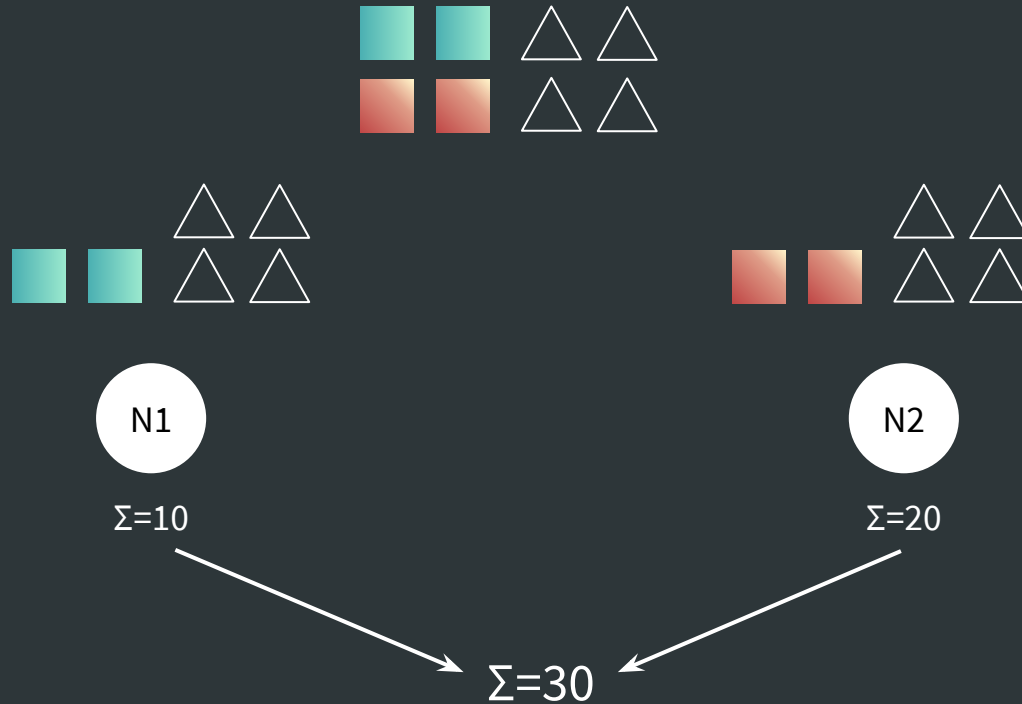
$\Sigma=10$



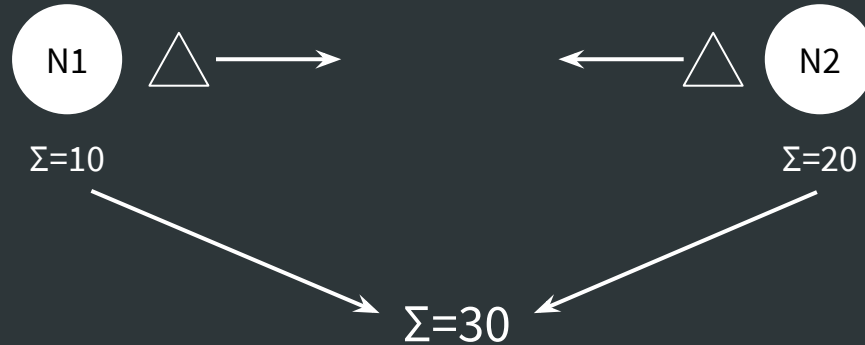
$\Sigma=20$

$\Sigma=30$

Several Nodes/no Affinity Across Data



Several Nodes/no Affinity Across Data

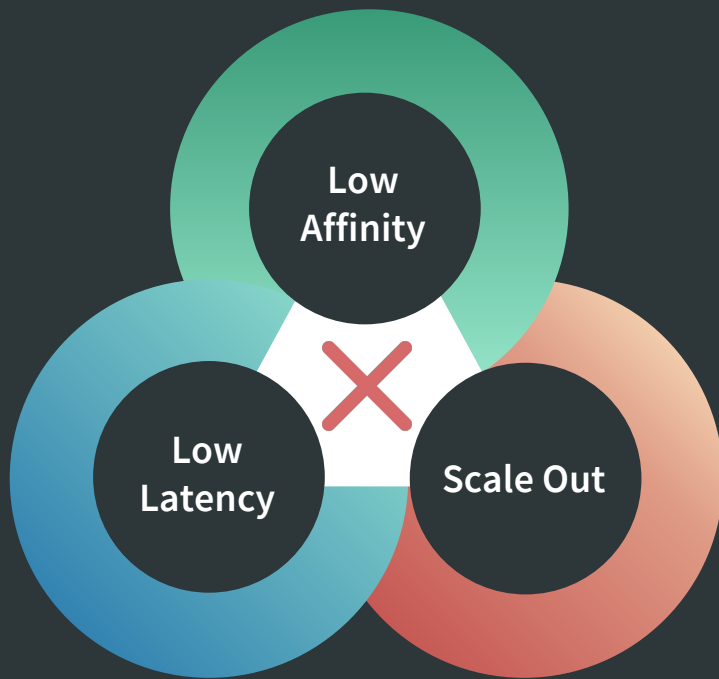


Several Nodes/no Affinity Across Data





Several Nodes/no Affinity Across Data

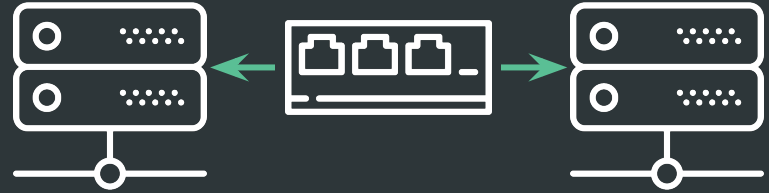


Data is Remote: Laws of Nature

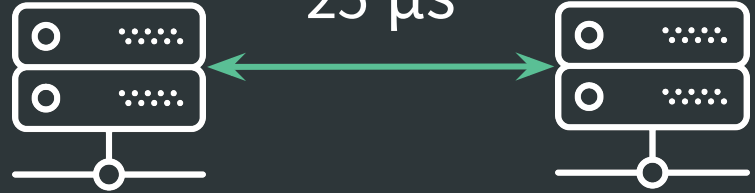
45 ms



100 μ s

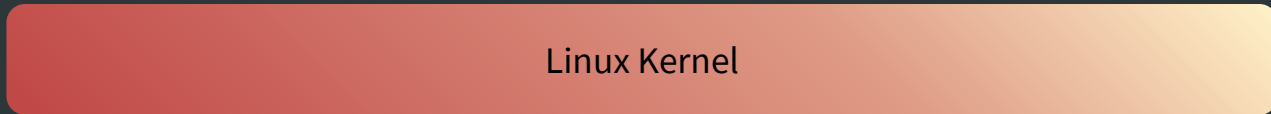
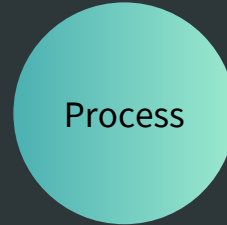
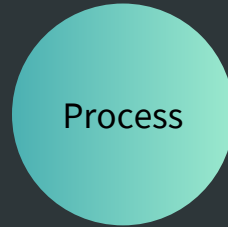
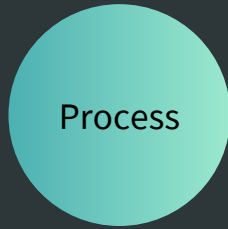
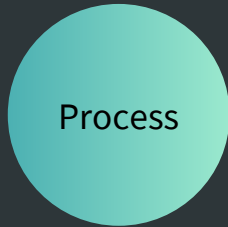


25 μ s



Data is Remote: Operating System

1-3 μ s



Unnecessary Object Creation

1s

Unnecessary Object Creation



FIG. 109. — A GARBAGE COLLECTOR.

Unnecessary Object Creation

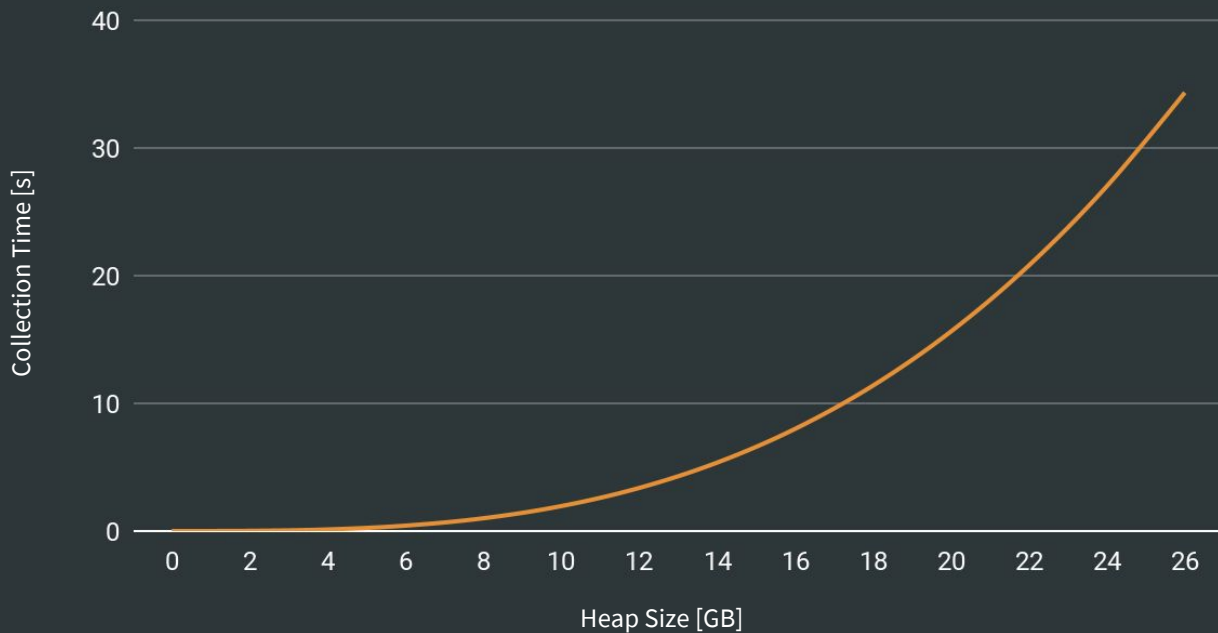
```
private Map<Integer, Film> createMap() {  
    return LongStream.range(0, (int) 1E9)  
        .mapToObj(Main::createFilm)  
        .collect(  
            toMap(  
                Film::getFilmId,  
                Function.identity()  
            )  
        );  
}
```

Unnecessary Object Creation

- Objects
 - Many extra objects such as Integer and Map.Entry are created
 - Subject to overhead and byte alignment
 - Long expected lifetime leads to repeated evaluation by GC
 - Difficult to retrieve objects (except for key/value), e.g. films longer than 60 s
- TreeMap for each field adds to the overhead
- gcExecutionTime(#objects) is not linear

Unnecessary Object Creation

Collection Time [s] vs. Heap Size [GB]



Unnecessary Object Creation

To write a single Java object to main memory takes 200 ns

```
00 00 EA B6 08 E2 02 01 00 00 01 A9 AA FF FF FE 00 01 00 10
```

Conclusion: Creating shared objects is not ultra-low latency

Lack of Parallelism

```
$ nproc -all  
32
```

```
$ top
```

PID	USER	%CPU	%MEM
2105	java	100.0	5.4
1	root	0.5	0.4

The Solution:

In-JVM-Memory

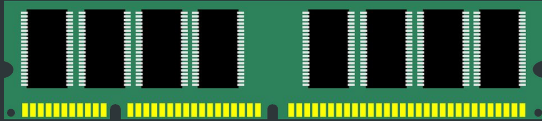
What is that...?

In-Memory vs. In-JVM-Memory

In-Memory

Data is in RAM

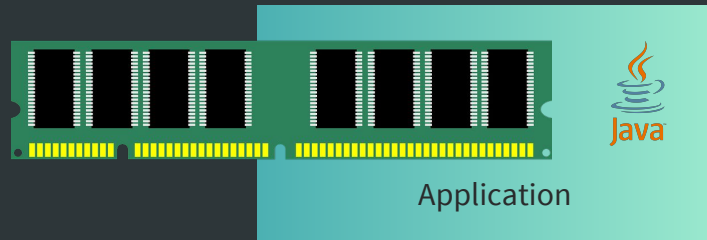
The application is remotely connected to a grid, other machine, other process



In-JVM-Memory

Data is in RAM

The application and data resides in the same JVM



In-JVM-Memory Makes Ultra-Low Latency Possible

CPU Cache Latencies

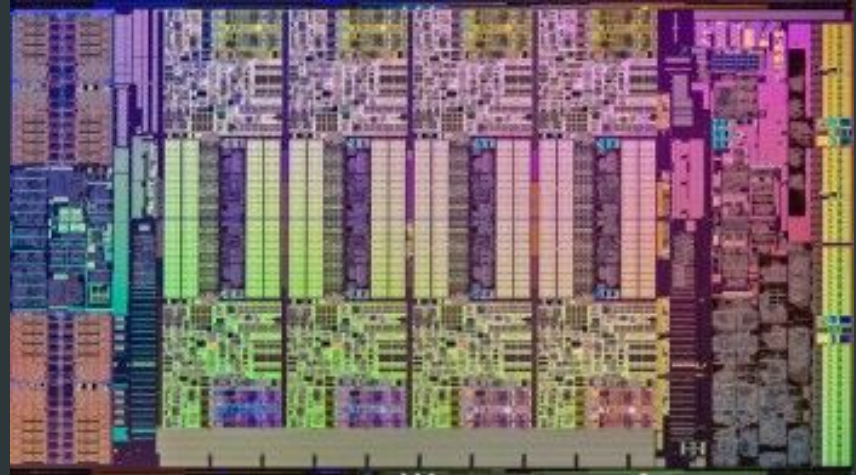
L1 ~0.5 ns

L2 ~7 ns

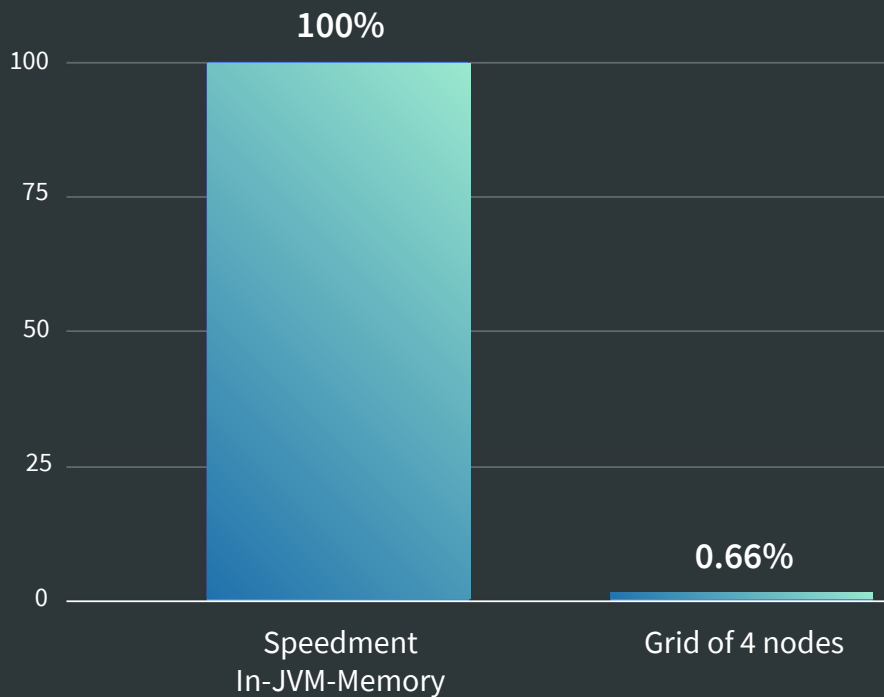
L3 ~20 ns

64-bit Main Memory Read

~100 ns



In-Memory vs. In-JVM-Memory: Performance



In-JVM-Memory Scalability

Is that even possible...?



Scaling up In-JVM-Memory

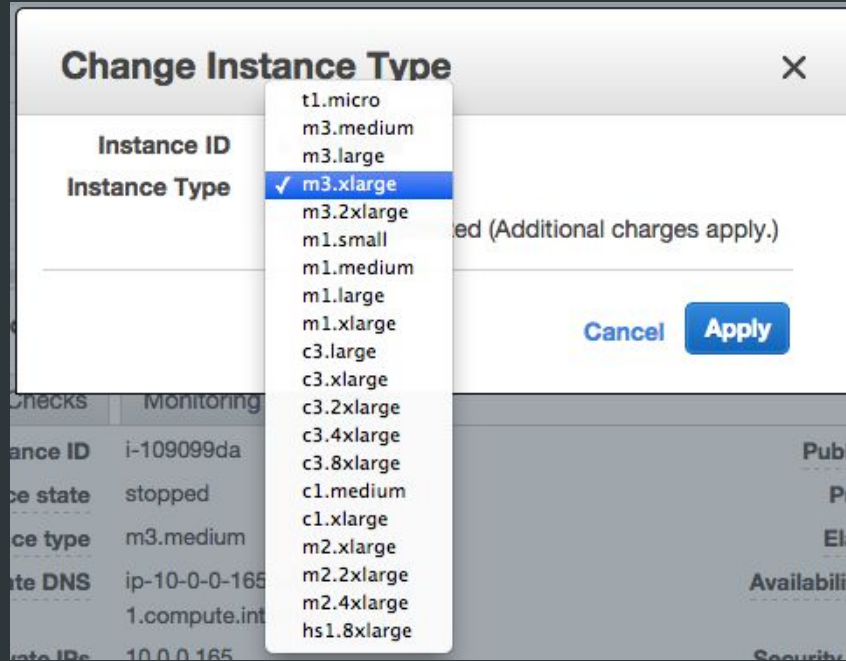
Today: Scale up to 12 TB (Intel® Xeon® Processor E7-8855 v4 * 4)

Instance Name	Memory	Logical Processors	Dedicated EBS Bandwidth	Network Bandwidth
u-6tb1.metal	6 TiB	448	14 Gbps	25 Gbps
u-9tb1.metal	9 TiB	448	14 Gbps	25 Gbps
u-12tb1.metal	12 TiB	448	14 Gbps	25 Gbps



Soon: Scale up to 48 TB

Scaling up In-JVM-Memory



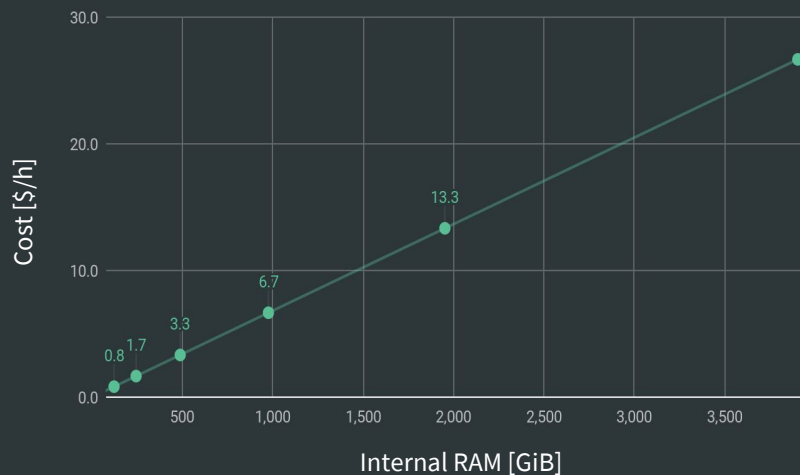
Increase Memory in the Cloud as You Grow

General Belief



Fact

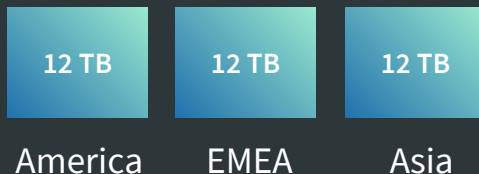
AWS “x1e.Nxlarge”



What if I Have More Than 12 TB?

High Level Sharding

Per year, region, segment



Memory Mapping

(e.g. IMDT)

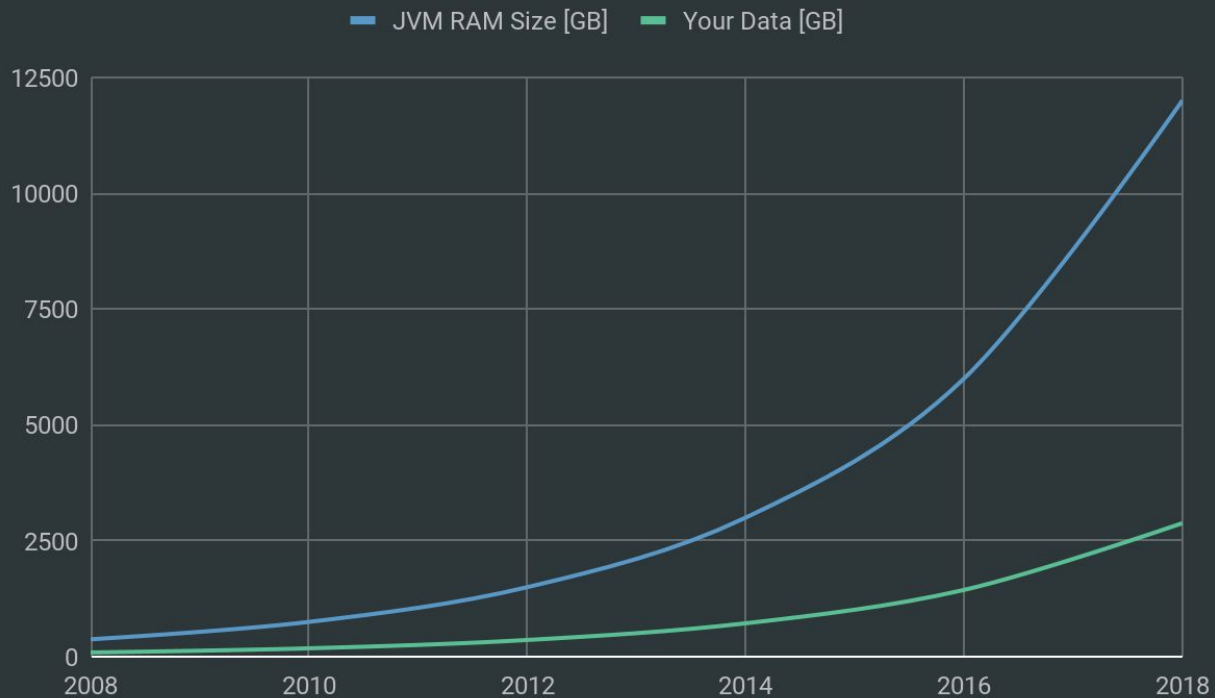


In-JVM-Memory Solution

Add-on for your current solution
for part of your data



What if My Data Grows?

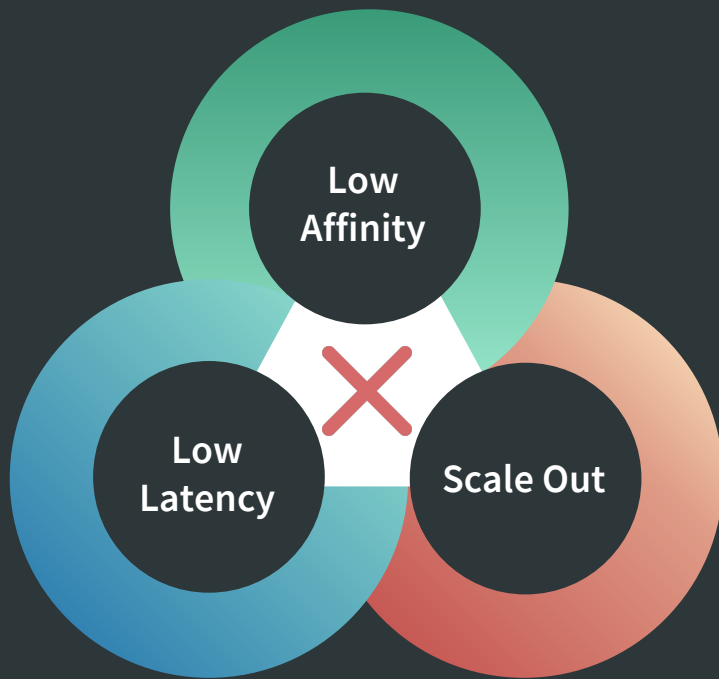


In-JVM-Memory vs. In-Memory Performance

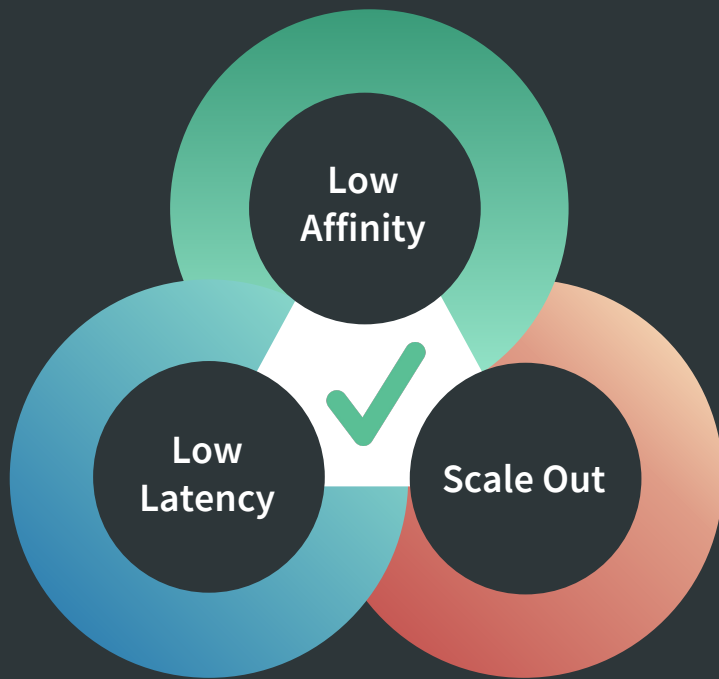
Data with 75% correlation



Recap



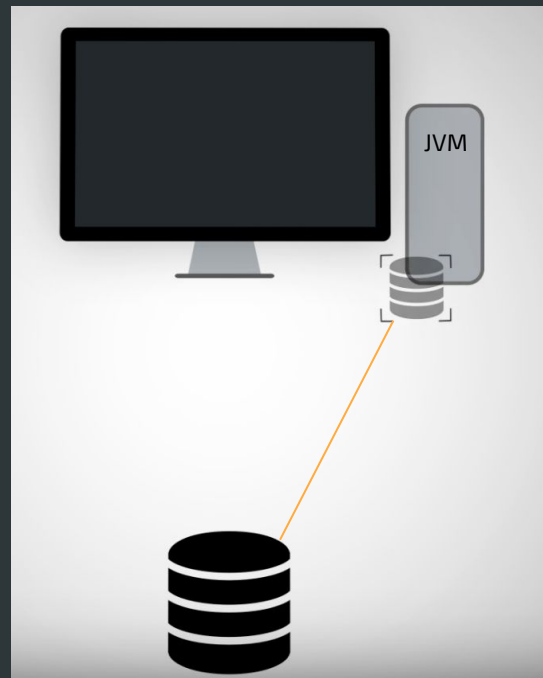
In-JVM-Memory Makes it Possible



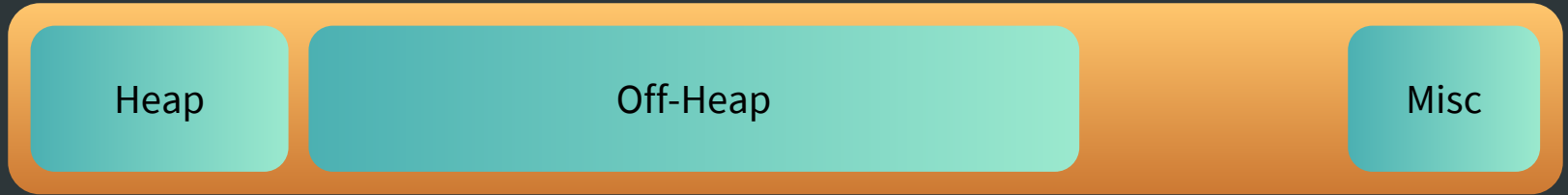
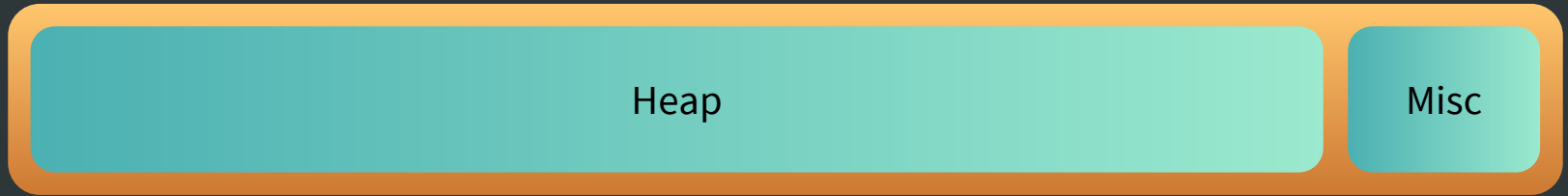
In-JVM-Memory Solution: Speedment

Speedment: In-JVM-Memory DataStore

- Continuously creates data snapshots from a data source
- Places the copy within the JVM
- Off-Heap Data
- Off-Heap Indexing
- Operations $O(1)$ or $O(\log(N))$
- No Impact on Garbage Collect




Speedment: In-JVM-Memory DataStore



Speedment API: Java Stream ORM

```
java.util.stream.Stream
```

Speedment API: Java Stream ORM



Java™ magazine
By and for the Java community

MAY/JUNE 2017

Libraries

- 10 LOMBOK: ANNOTATIONS FOR CLEANER CODE
- 16 JDEFERRED'S ASYNC EVENT MANAGEMENT
- 34 DATABASE ACCESS WITH STREAMS
- 28 BEST PRACTICES FOR LIBRARY DESIGN

ORACLE.COM/JAVAMAGAZINE

ORACLE

//databases /



PER MINBORG

Database Actions Using Java 8 Stream Syntax Instead of SQL

Speedment 3.0 enables Java developers to stay in Java when writing database applications.

Why should you need to use SQL when the same semantics can be derived directly from Java 8 streams? If you take a closer look at this objective, it turns out there is a remarkable resemblance between the verbs of Java 8 streams and SQL commands, as summarized in [Table 1](#).

Streams and SQL queries have similar syntax in part because both are declarative constructs, meaning they describe a result rather than state instructions on how to compute the result. Just as a SQL query describes a result set rather than the operations needed to compute the result, a Java stream describes the result of a sequence of abstract functions without dictating the properties of the actual computation.

The open source project Speedment capitalizes on this similarity to enable you to perform database actions using Java 8 stream syntax instead of SQL. It is available on [GitHub](#) under the business-friendly Apache 2 license for open source databases. (A license fee is required for commercial databases.) Feel free to clone the entire project.

About Speedment

Speedment allows you to write pure Java code for entire database applications. It uses lazy evaluation of streams, meaning that only a minimum set of data is actually pulled from the database into your application and only as the elements are needed.

In the following example, the objective is to print out all `Film` entities having a rating of PG-13 (meaning "parents are strongly cautioned" in the US). The films are located in a database table represented by a Speedment Manager variable

SQL COMMAND	JAVA 8 STREAM OPERATIONS
FROM	stream()
SELECT	map()
WHERE	filter() (BEFORE COLLECTING)
HAVING	filter() (AFTER COLLECTING)
JOIN	flatMap() OR map()
DISTINCT	distinct()
UNION	concat(s0, s1).distinct()
ORDER BY	sorted()
OFFSET	skip()
LIMIT	limit()
GROUP BY	collect(groupingBy())
COUNT	count()

Table 1. SQL commands and their counterpart verbs in Java 8 streams



Speedment API: Java Stream ORM

Declarative Constructs in SQL and Stream

```
SELECT * FROM FILM  
WHERE RATING = 'PG-13'
```

```
films.stream()  
    .filter(Film.RATING.equal("PG-13"))
```

Process Data without Creating Intermediate Objects

```
films.stream()  
  .filter(Film.RATING.equal("PG-13"))  
  .count();
```

Process Data without Creating Intermediate Objects

```
films.stream()  
  .filter(Film.RATING.equal("PG-13"))  
  .collect(toJsonLengthAndTitle());
```

Speedment Can Process Data without Creating Intermediate Objects

```
films.stream()  
  .filter(Film.RATING.equal("PG-13"))  
  .collect(toJsonLengthAndTitle());
```

index	film_id	length	rating	year	language	title
[0]	0	267	267	0	0	0
[1]	267	0	0	267	267	267
[2]	523	523	523	523	523	523

index	film_id	length	rating	year	language	Title
[0]	1	123	PG-13	2006	1	ACAD..
[267]	2	69	G	2006	1	ACE G...
[523]	3	134	PG-13	2006	1	ADAP...

Speedment: Off-Heap Joins/Aggregations

```
var join = joinComponent
    .from(FilmManager.IDENTIFIER)
    .innerJoinOn(Language.LANGUAGE_ID).equal(Film.LANGUAGE_ID)
    .build(Tuples::of);
```

Speedment: Off-Heap Joins/Aggregations

```
var offHeapAggregator = Aggregator.builder(Result::new)
    .on(Film.LANGUAGE_ID).key(Result::setLanguage)
    .on(Film.RATING).key(Result::setRating)
    .on(Film.LENGTH).average(Result::setAverage)
    .build();
```

Speedment: Off-Heap Joins/Aggregations

```
var result = join.stream()  
                .collect(offHeapAggregator);
```

Speedment: Parallel Processing

```
join.stream()  
    .parallel()  
    .collect(offHeapAggregator);
```

Speedment: Parallel Processing

```
$ nproc -all
```

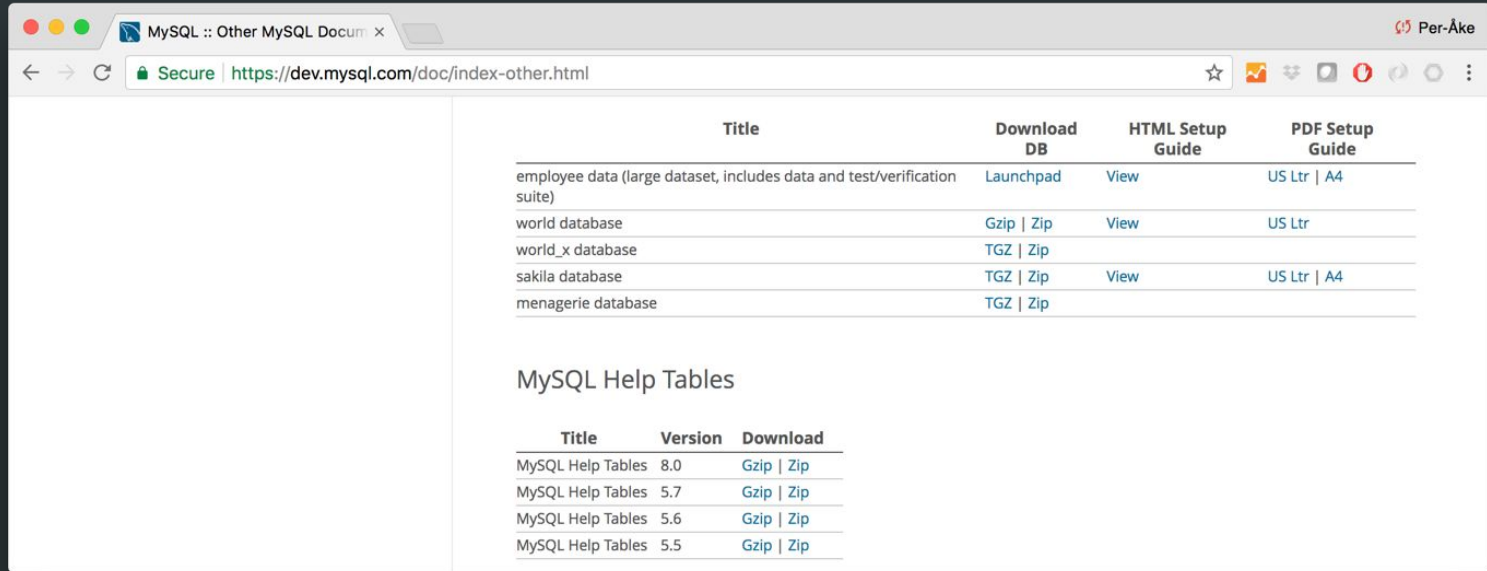
```
32
```

```
$ top
```

PID	USER	%CPU	%MEM
2107	java	3170.0	5.4
1	root	0.5	0.4

Hands on Demo

Demo: Download Sakila Example Database



The screenshot shows a web browser window with the URL <https://dev.mysql.com/doc/index-other.html>. The page displays a table of database download options and a section for MySQL Help Tables.

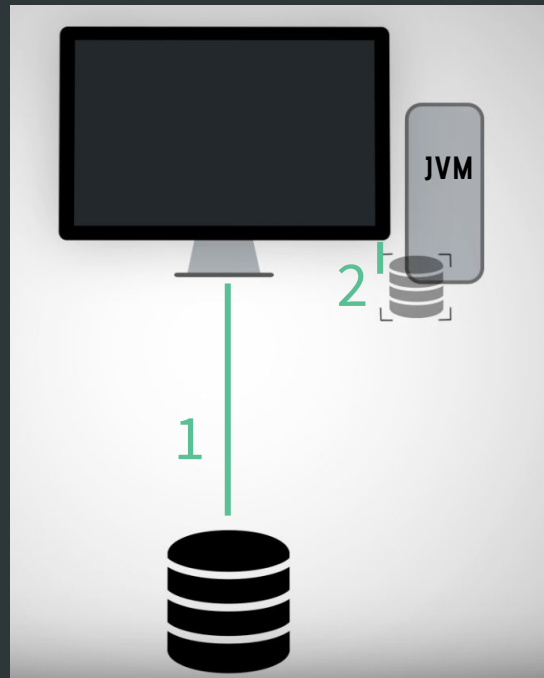
Title	Download DB	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	Launchpad	View	US Ltr A4
world database	Gzip Zip	View	US Ltr
world_x database	TGZ Zip		
sakila database	TGZ Zip	View	US Ltr A4
menagerie database	TGZ Zip		

MySQL Help Tables

Title	Version	Download
MySQL Help Tables	8.0	Gzip Zip
MySQL Help Tables	5.7	Gzip Zip
MySQL Help Tables	5.6	Gzip Zip
MySQL Help Tables	5.5	Gzip Zip

Demo: Stream

```
@Benchmark
public long filterAndCount() {
    return films.stream()
        .filter(RATING_EQUALS_PG_13)
        .count();
}
```



Demo: Initialize the Project

SPEEDMENT INITIALIZER

Database Type MySQL PostgreSQL
 MariaDB Oracle DB2
 AS400 SQL Server SQLite

JDBC Driver Version

Java Version Java 8 Java 9 Java 10
 Java 11

Plugins Enums Spring JSON

In-memory Acceleration Enable Disable

GroupId

ArtifactId

Version

[DOWNLOAD](#)

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <mssql.version>7.0.0.jre8</mssql.version>
    <speedment.version>3.1.12</speedment.version>
  </properties>
</project>
```

Demo: Connect to the Sakila Database

Speedment

CONNECT TO DATABASE

Database Type: MySQL

Database Host: 127.0.0.1 3306

Username: root

Password: ●●●●●●

Database Name: sakila

Use Connection URL

```
jdbc:mysql://127.0.0.1:3306?
useUnicode=true&character
Encoding=UTF-8&useServer
PrepStmts=true&zeroDateTi
meBehavior=convertToNull
&nullNamePatternMatchesA
ll=true&useLegacyDatetime
```

➔ Connect

Demo: Generate the Domain Model

The screenshot displays the Speedment application interface. The window title is "Speedment". The menu bar includes "File", "Edit", "Window", and "Help". There are "Reload" and "Generate" buttons in the top left, and an "Upgrade Now" button with the Speedment logo in the top right.

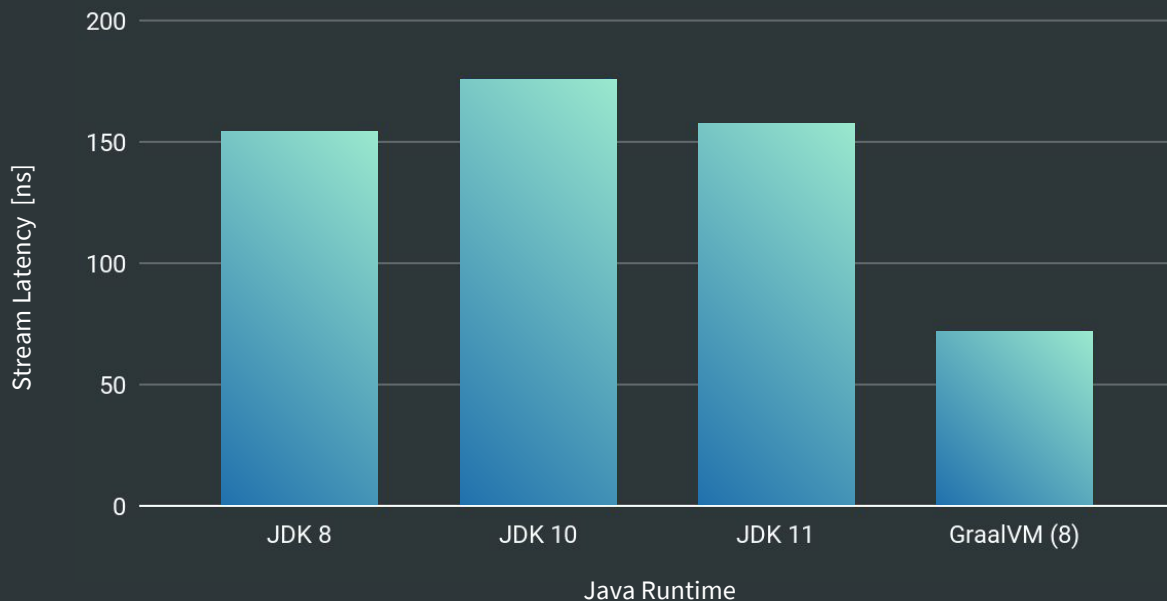
The interface is divided into two main sections:

- Node Hierarchy:** A tree view showing the database schema. The "sakila" database is expanded, showing tables like "actor", "address", "category", "city", "country", "customer", and "film". The "film" table is selected, showing its columns: "film_id", "title", "description", "release_year", "language_id", "original_language_id", "rental_duration", "rental_rate", "length", "replacement_cost", "rating", "special_features", and "last_update". It also shows foreign keys and indexes.
- Speedment settings for table 'film':** A configuration panel with the following settings:
 - Enabled:**
 - Table Name:**
 - Java Alias:** Auto
 - Package Name:** Auto

Demo

Ultra-Low Latency (Lower is Better)

Stream Latency vs. JDK



Use Existing Infrastructure

Easy Integration: Any Data Source



PostgreSQL



Speedment

CONNECT TO DATABASE

Database Type:

Database Host:

Username:

Password:

Database Name:

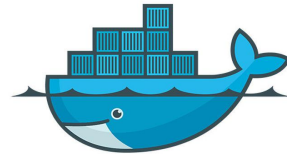
Use Connection URL

```
jdbc:mysql://127.0.0.1:3306?
useUnicode=true&character
Encoding=UTF-8&useServer
PrepStmts=true&zeroDateTi
meBehavior=convertToNull
&nullNamePatternMatchesA
ll=true&useLegacyDatetime
```

Deploy Anywhere



On Premise



docker

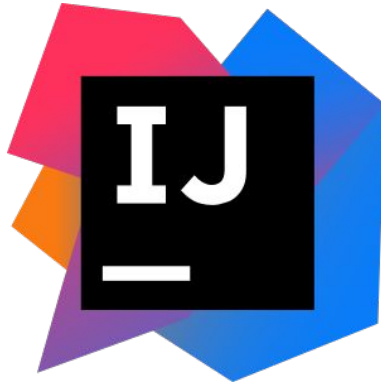


Google Cloud Platform

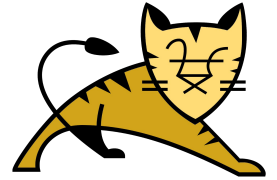


kubernetes

IDE Integration



Web Service Integration



Thanks!

E-mail

minborg@speedment.com

Free Trial:

www.speedment.com/initializer



Speedment Can Process Data without Creating Intermediate Objects

```
films.stream()  
  .filter(Film.RATING.equal("PG-13"))  
  .collect(toJsonLengthAndTitle());
```

index	film_id	length	rating	year	language	title
[0]	0	267	267	0	0	0
[1]	267	0	0	267	267	267
[2]	523	523	523	523	523	523

index	film_id	length	rating	year	language	Title
[0]	1	123	PG-13	2006	1	ACAD..
[267]	2	69	G	2006	1	ACE G...
[523]	3	134	PG-13	2006	1	ADAP...

Outline

1. Objects on the Stack
2. Proper Performance Testing
3. Short-circuit Streams for Massive Performance Gain
4. Holding Terabytes of Data in the JVM with no GC impact
5. Create Large Aggregations of Data without Intermediate Objects

In-JVM-Memory Makes Ultra-Low Latency Possible

CPU Cache Latencies

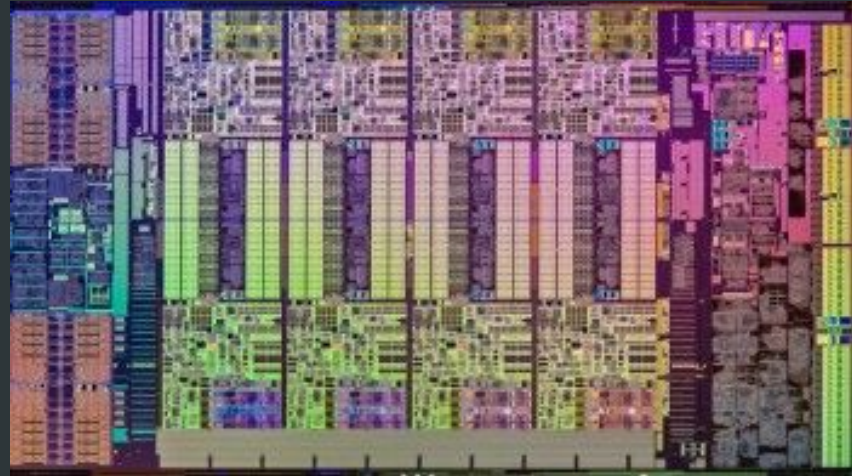
L1 ~0.5 ns

L2 ~7 ns

L3 ~20 ns

64-bit Main Memory Read

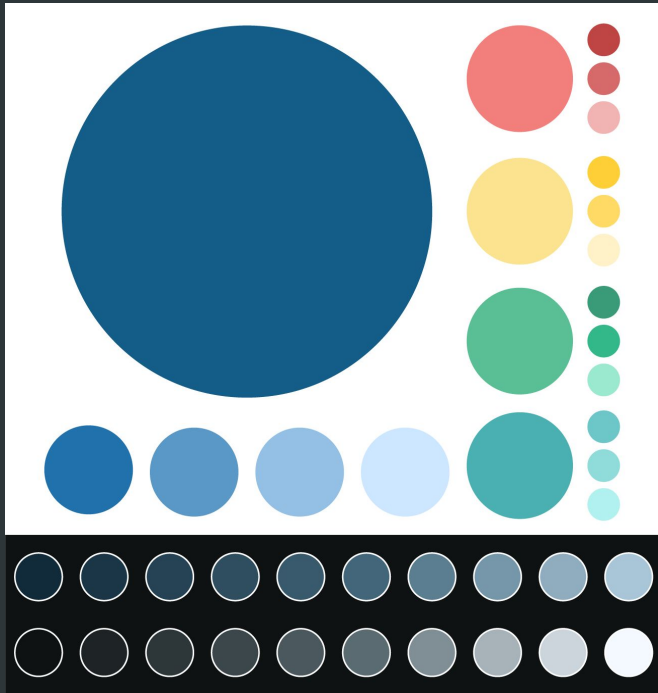
~100 ns



Process Data without Creating Intermediate Objects

```
films.stream()  
  .filter(Film.RATING.equal("PG-13"))  
  .count();
```

Använd dessa färger i figurer



#155d89 #2171ad #5a98c8 #93c0e4 #cce7ff

#f27f7c #bf4545 #d66969 #f2b3b3

#fce390 #ffcf38 #ffda64 #fff2c8

#5abf95 #3a9b79 #33b88a #9be9ce

#4bb0b2 #6dc6c7 #8fcdcb #b1f2f0

#112B3A #1B3747 #254354 #2F4E60 #395A6D #43667A

#5C7E91 #7696A9 #90ADC0 #A9C5D8

#0F1213 #1E2426

#4B595F #5A6B72

#808F95 #A7B2B9 #CDD5DC #F3F9FF