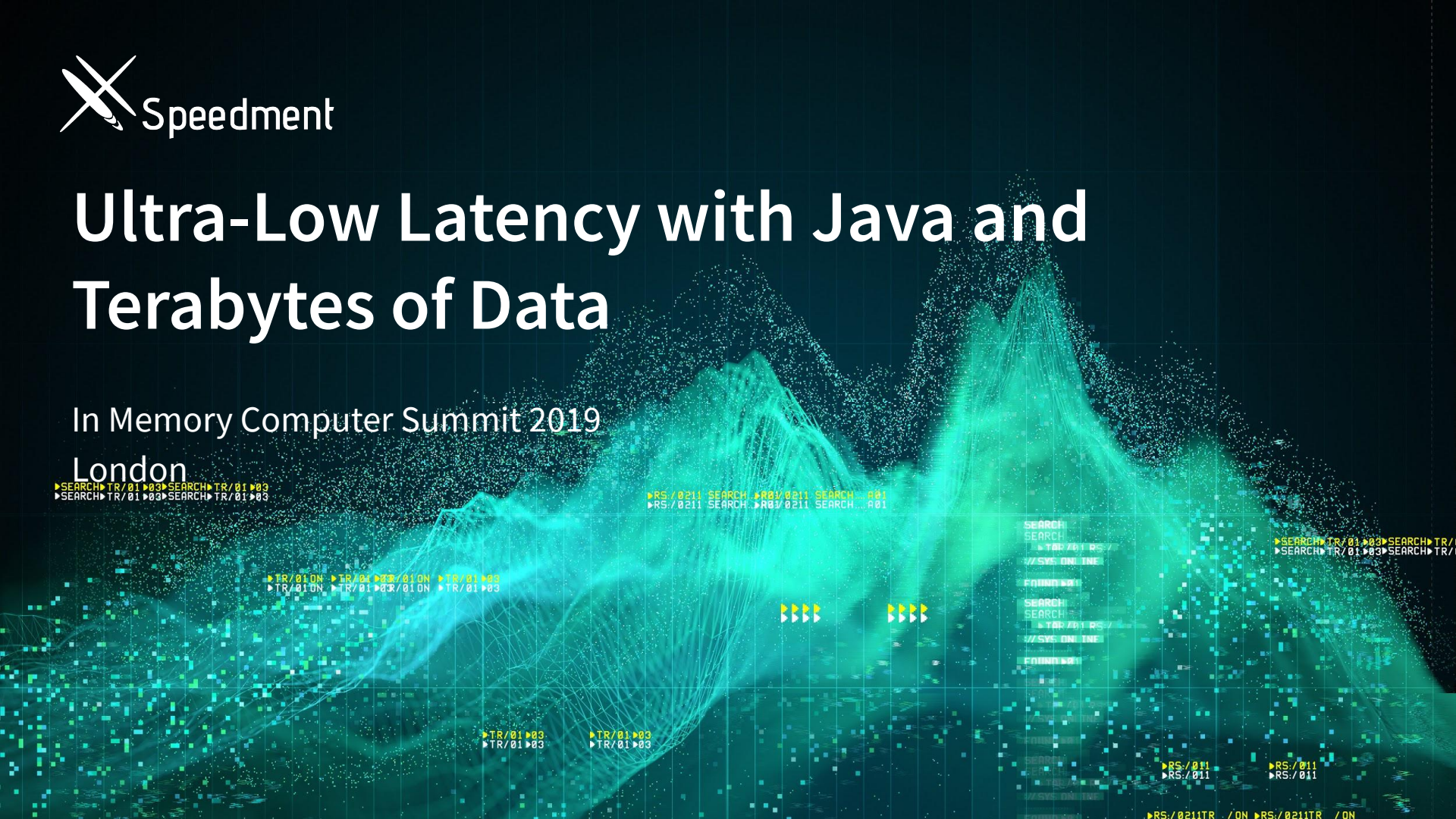
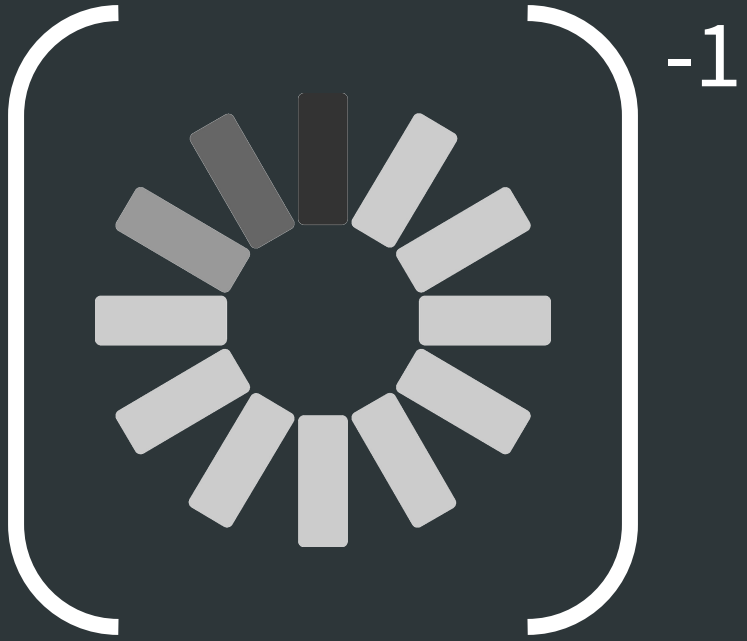

Speedment
Ultra-Low Latency with Java and Terabytes of Data
In Memory Computer Summit 2019
London

About Me

Per Minborg

- **Alumni speaker:** JavaOne, DevNexus and Oracle Code
- **Writer:** Modern Java , Oracle Java Magazine, DZone, blog
- **Inventor, entrepreneur:** Speedment
- **Silicon Valley, Gothenburg**


-1

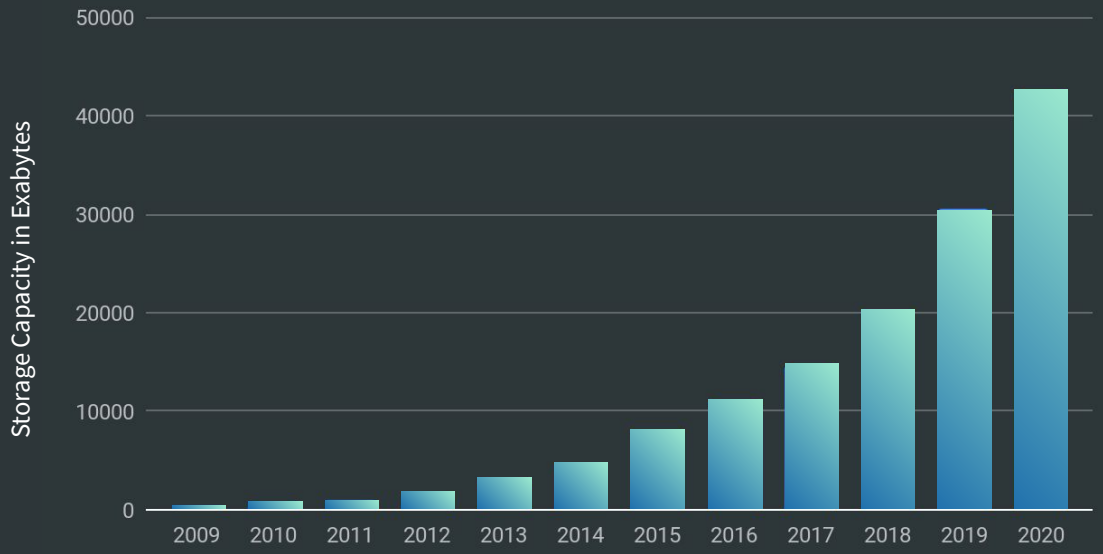
Ultra-Low Latency = 200 ns

Why Are Applications Slow?

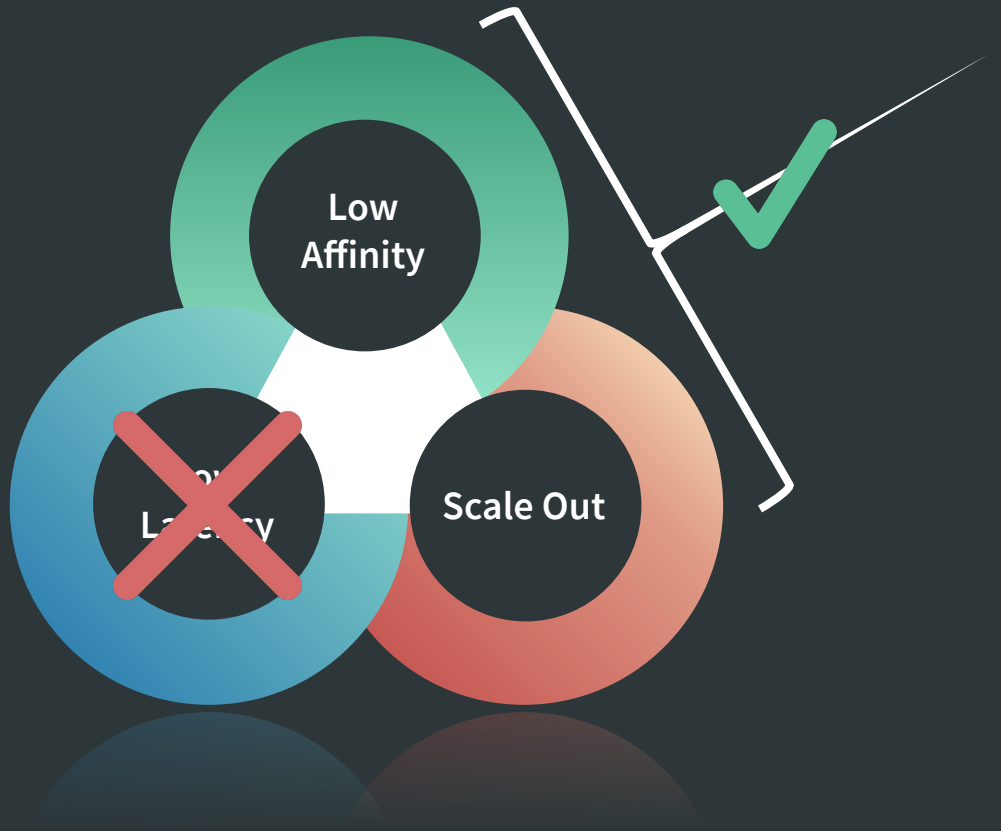
1. Slow Databases
2. Data on Several Nodes and no Affinity Across Data
3. Data is Remote
4. Unnecessary Object Creation / Garbage Collect Problem
5. Lack of Parallelism

Slow Databases

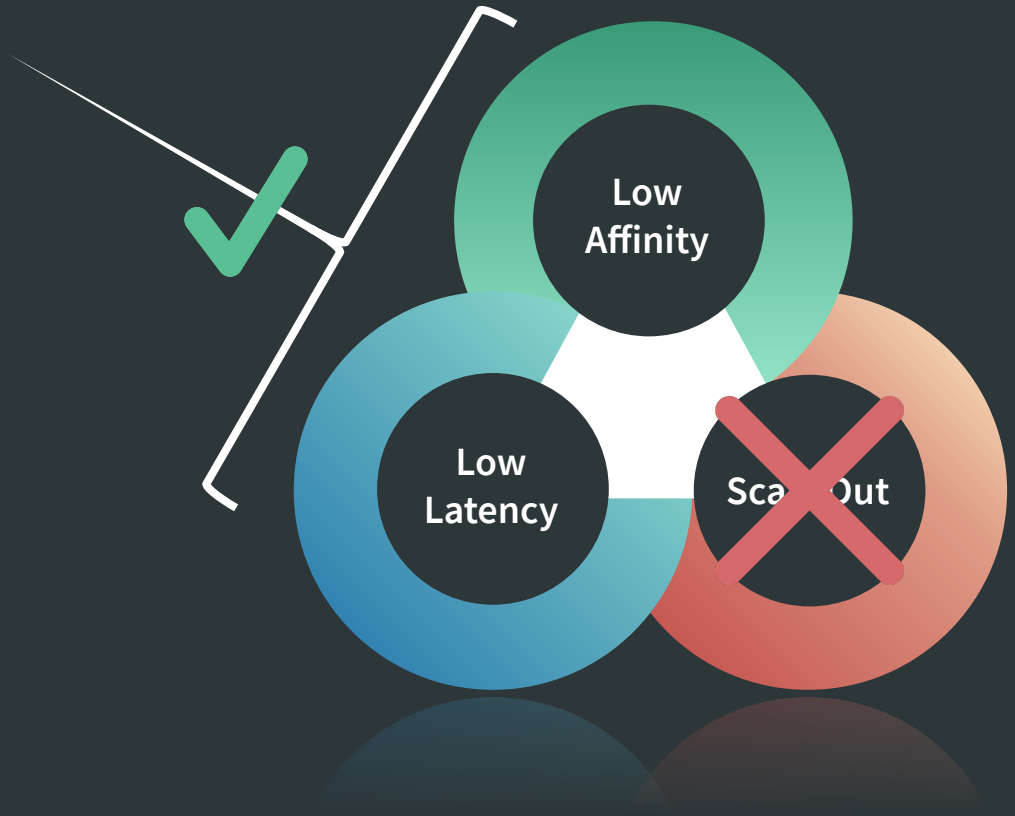
Data grows exponentially, which clogs systems



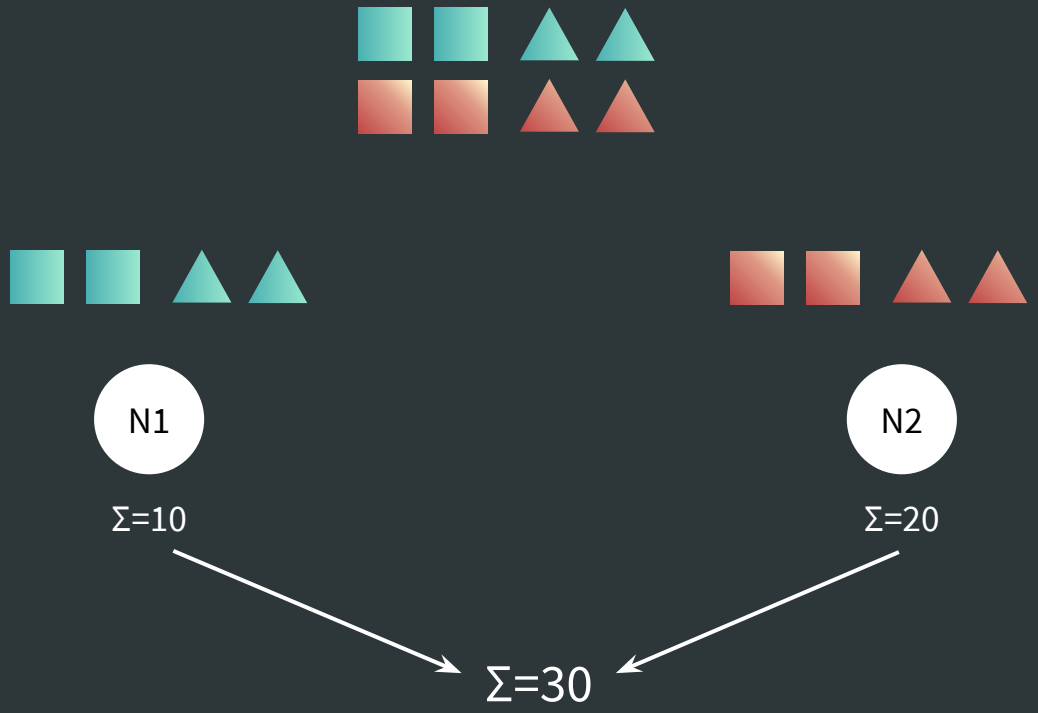
Several Nodes/no Affinity Across Data



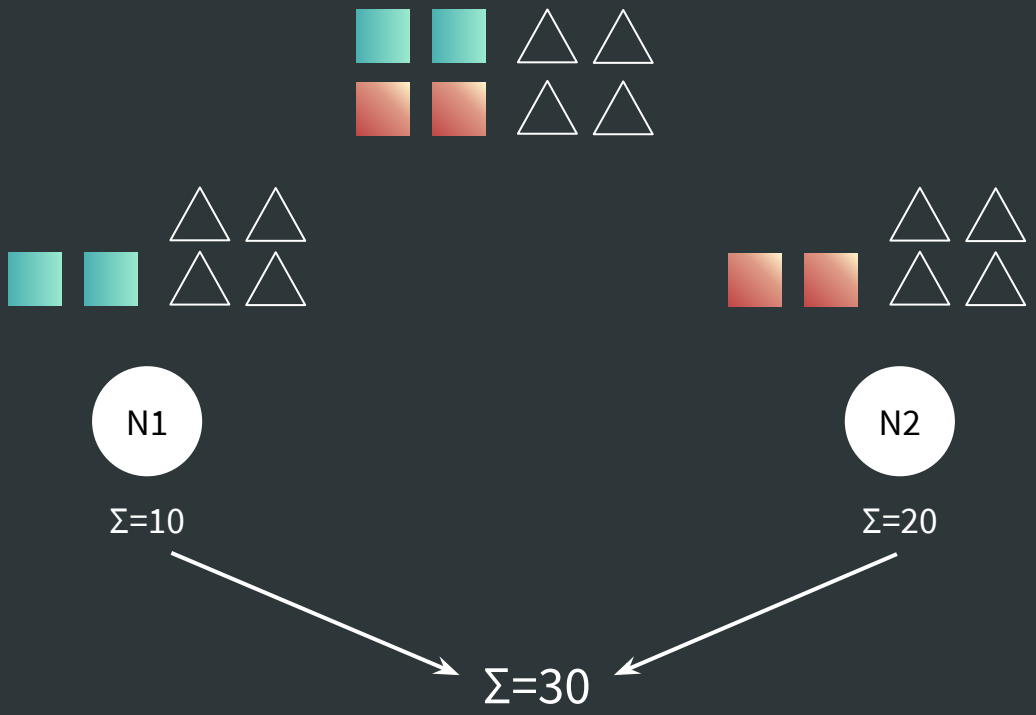
Several Nodes/no Affinity Across Data



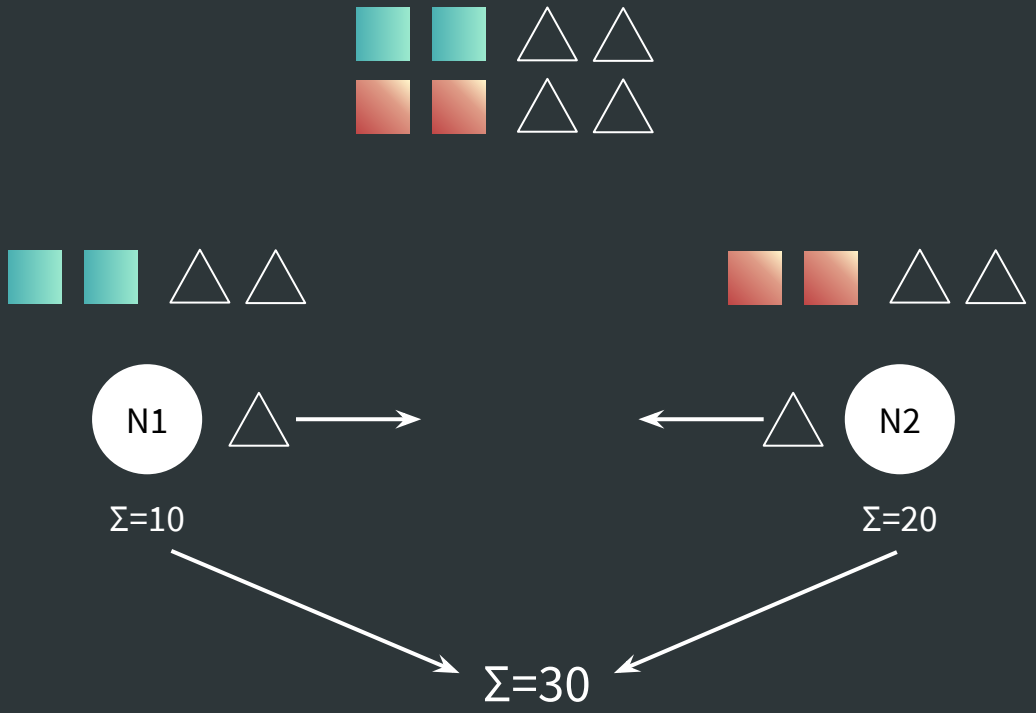
Several Nodes/no Affinity Across Data



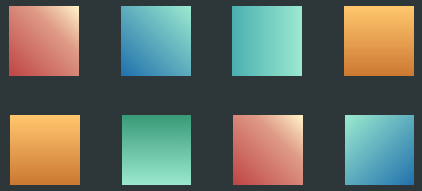
Several Nodes/no Affinity Across Data



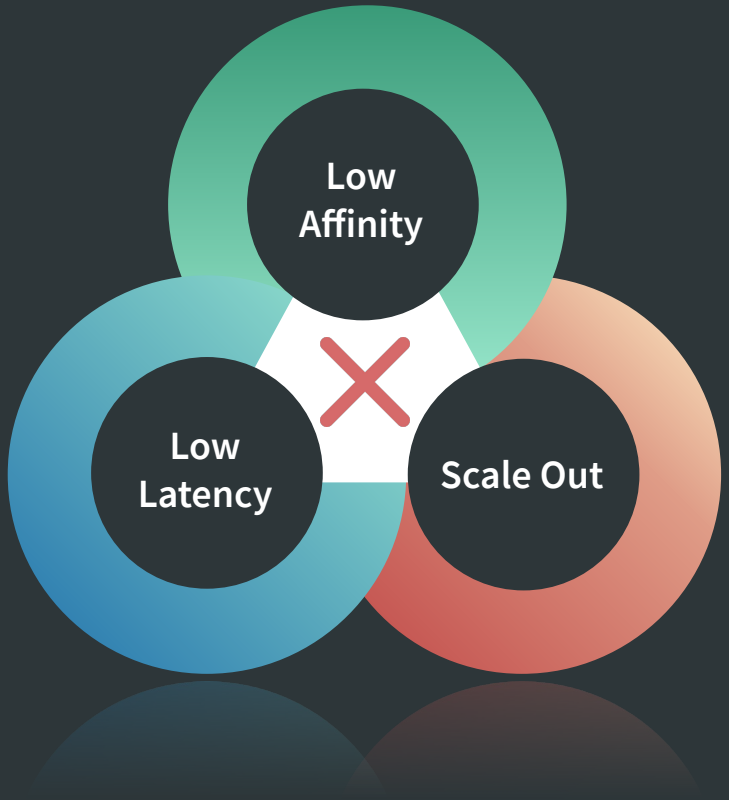
Several Nodes/no Affinity Across Data



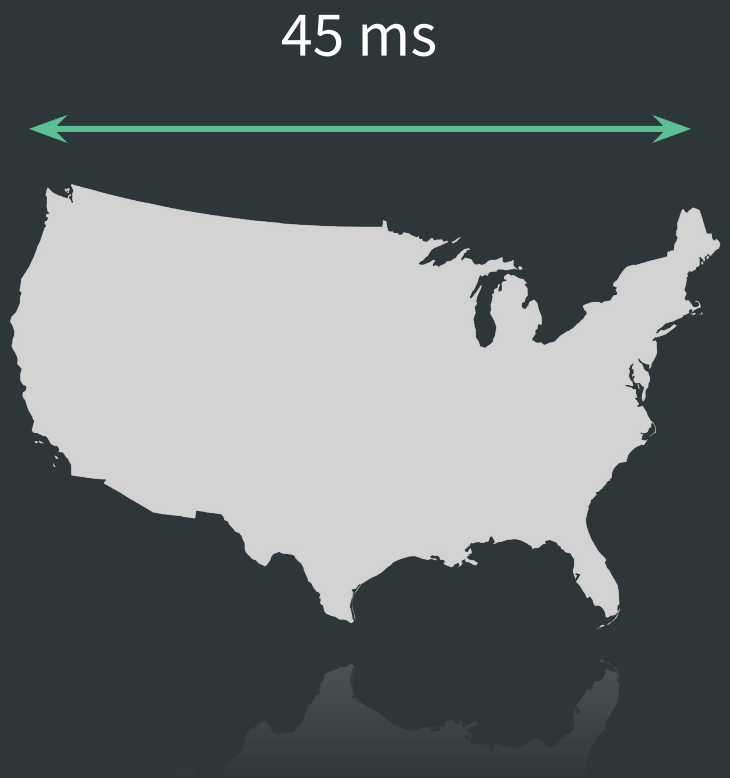
Several Object Types/Nodes/no Affinity Across Data

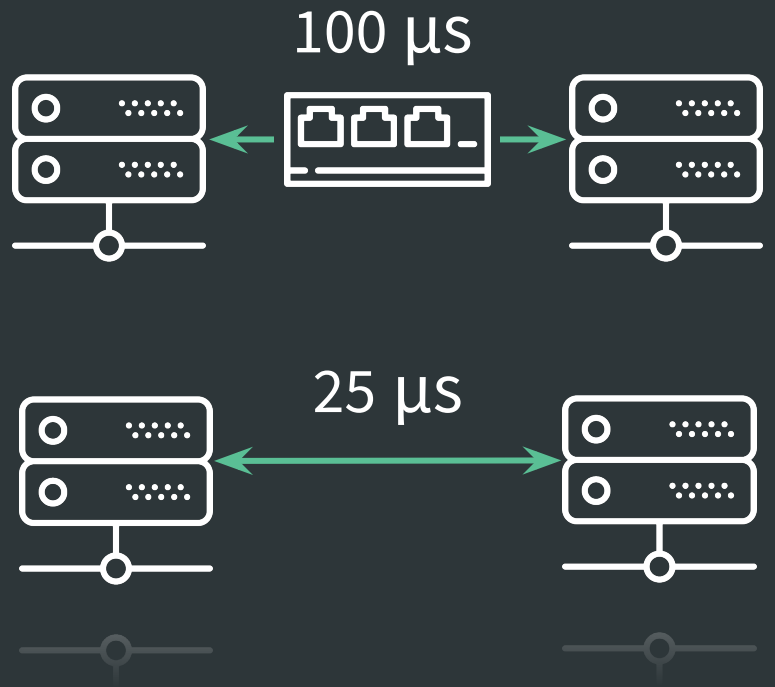


Several Nodes/no Affinity Across Data

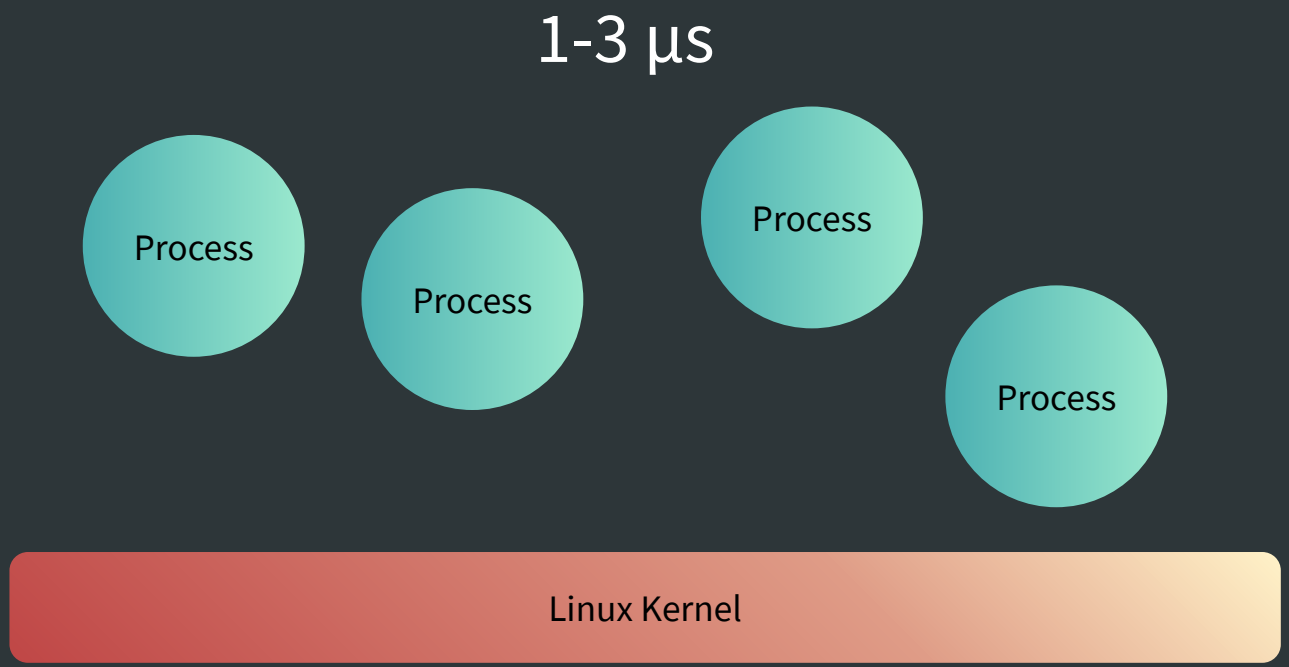


Data is Remote: Laws of Nature





Data is Remote: Operating System



Unnecessary Object Creation

1s

Unnecessary Object Creation

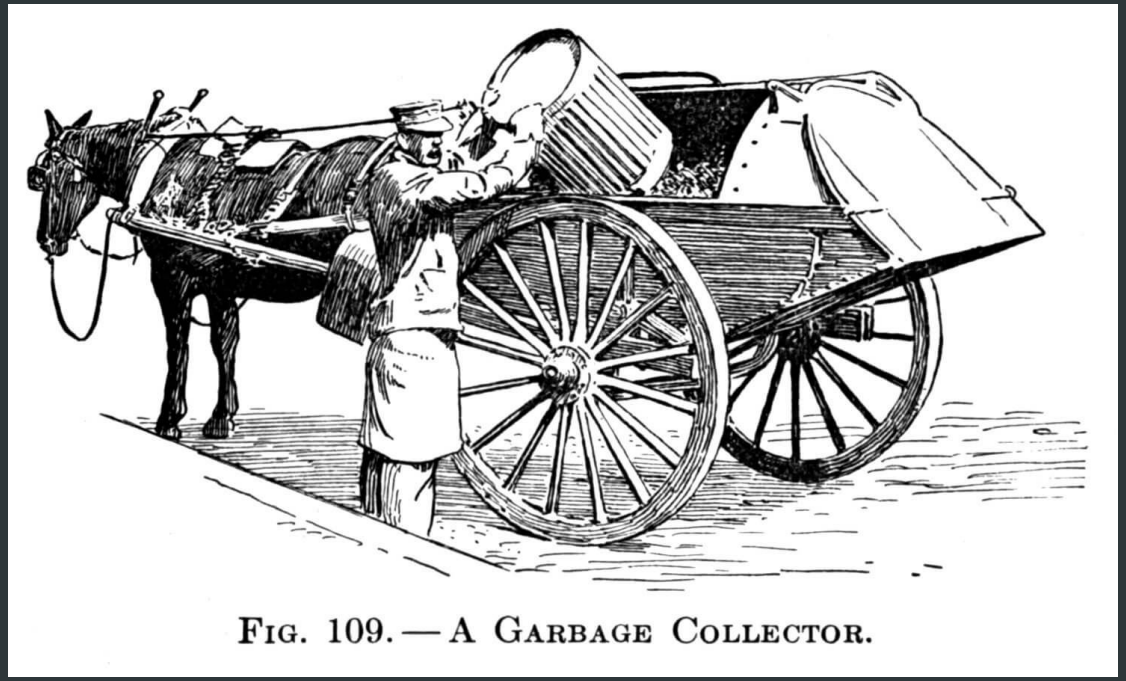


FIG. 109. — A GARBAGE COLLECTOR.

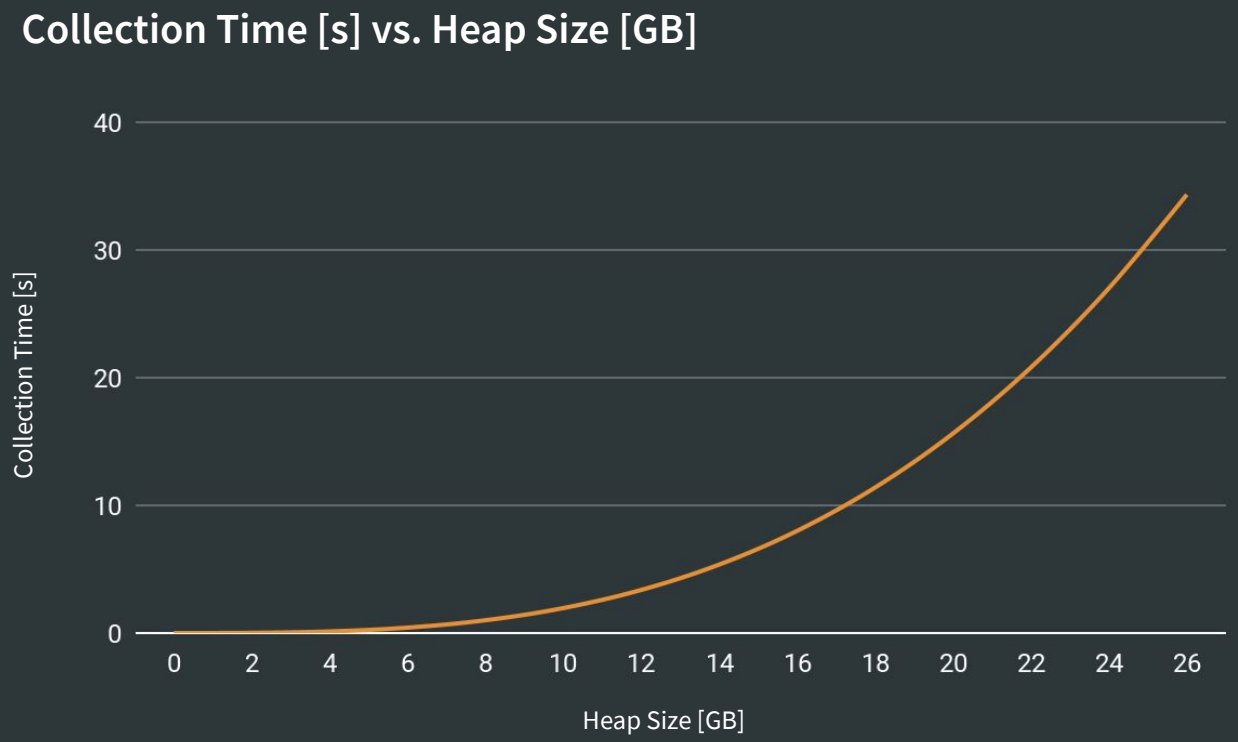
Unnecessary Object Creation

```
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



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

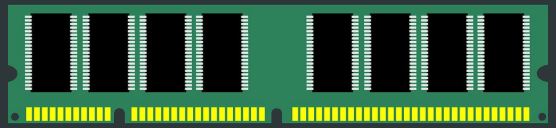
What is that...?

In-Memory vs. In-JVM-Memory

In-Memory

Data is in RAM

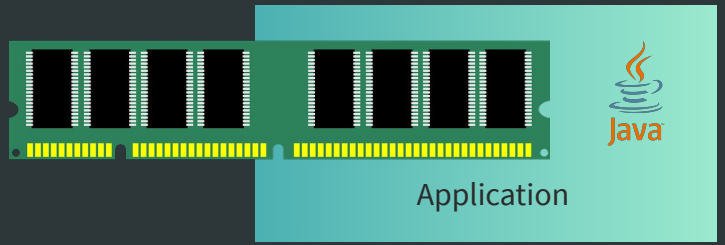
The application is remotely connected to a grid, other machine, other process



In-JVM-Memory

Data is in RAM

The application and data resides in the same JVM



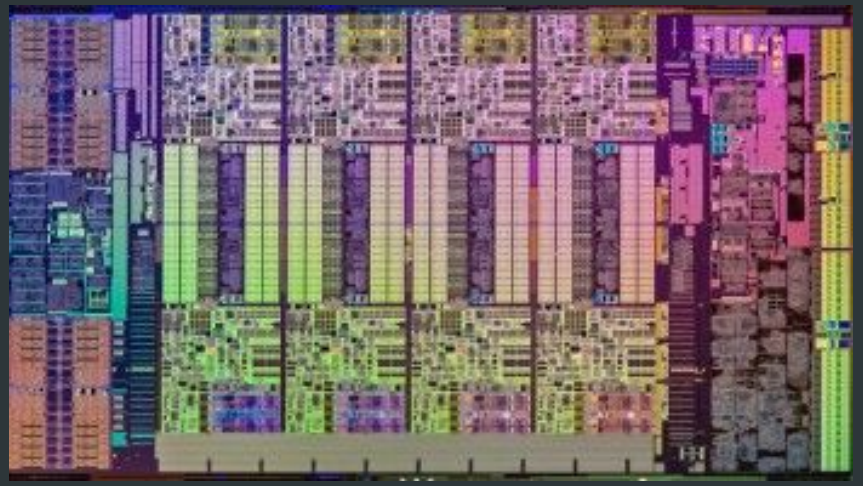
In-JVM-Memory Makes Ultra-Low Latency Possible

CPU Cache Latencies

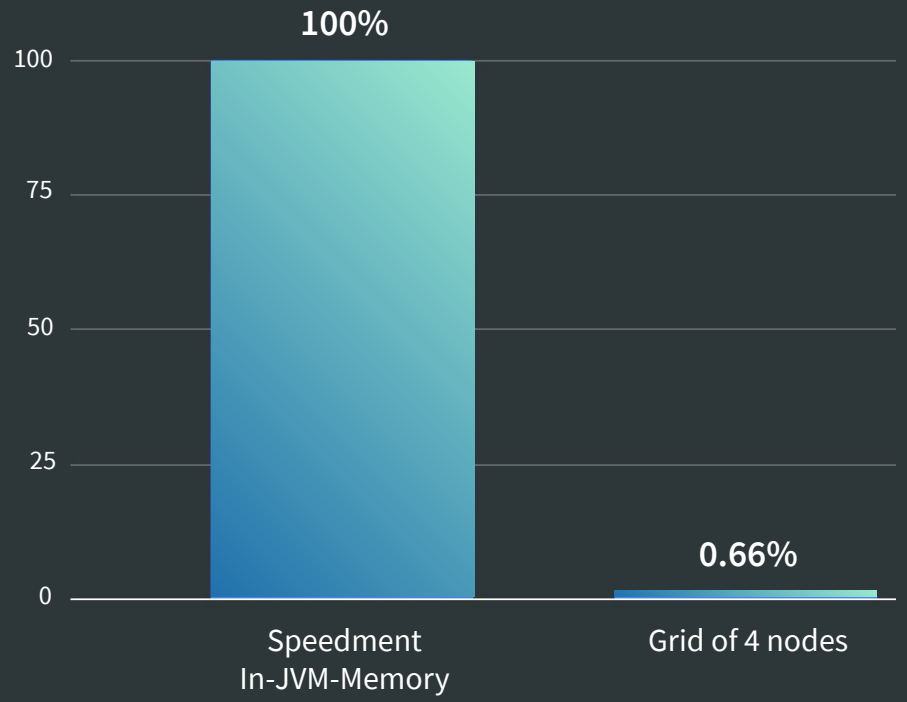
- L1 ~0.5 ns
- L2 ~7 ns
- L3 ~20 ns

64-bit Main Memory Read

- ~100 ns



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



In-JVM-Memory Scalability

Is that even possible...?


STARBUCKS
COFFEE

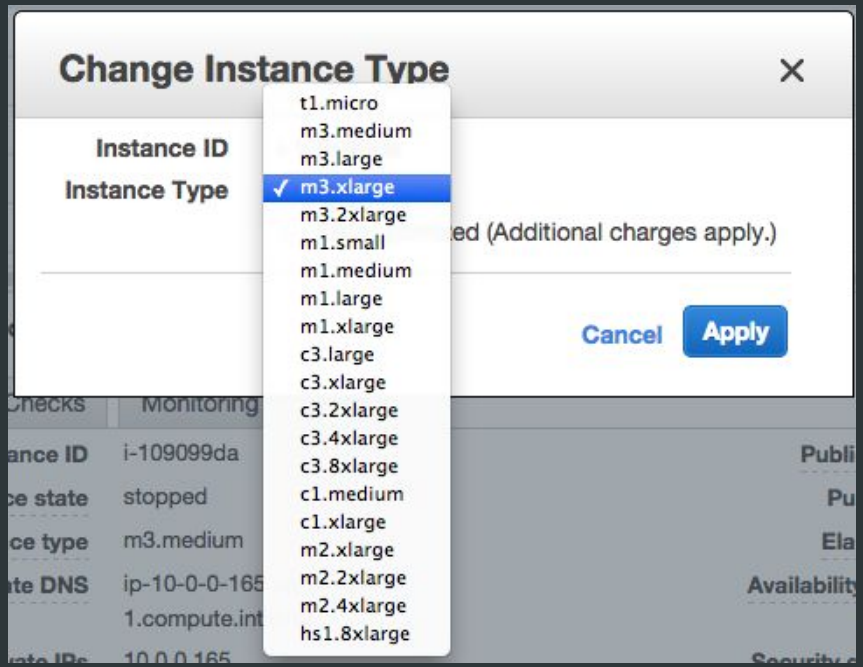
Scaling up In-JVM-Memory

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

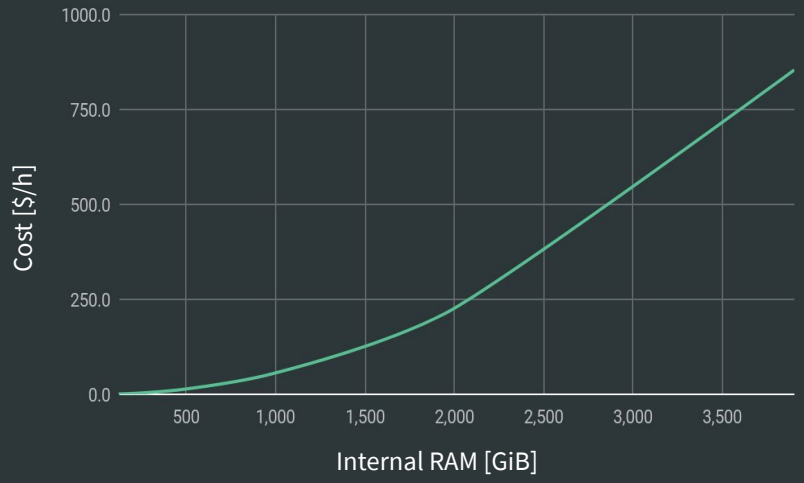
Soon: Scale up to 48 TB

Scaling up In-JVM-Memory



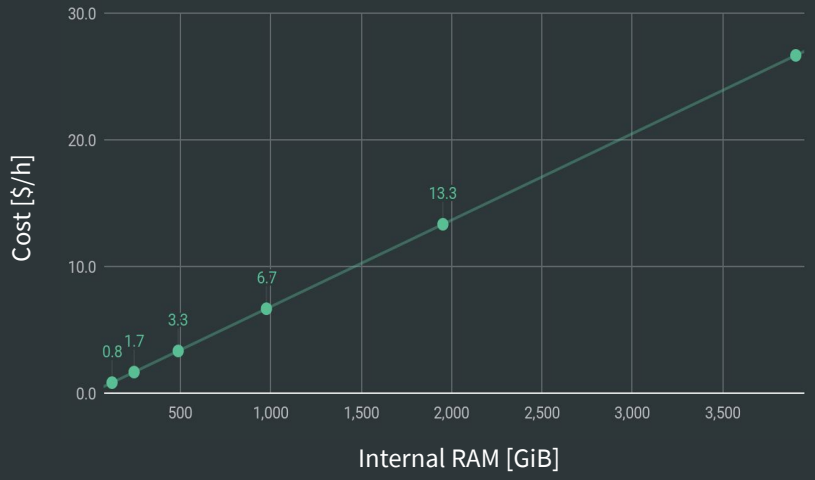
Increase Memory in the Cloud as You Grow

General Belief



Fact

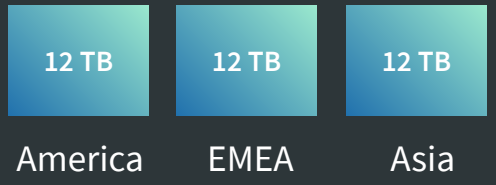
AWS “x1e.Nxlarge”



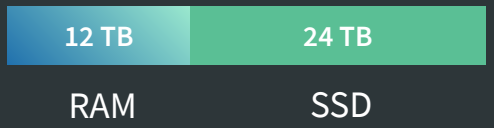
What if I Have More Than 12 TB?

High Level Sharding

Per year, region, segment

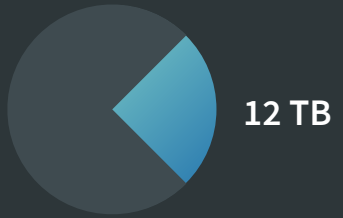


Memory Mapping (e.g. IMDT)

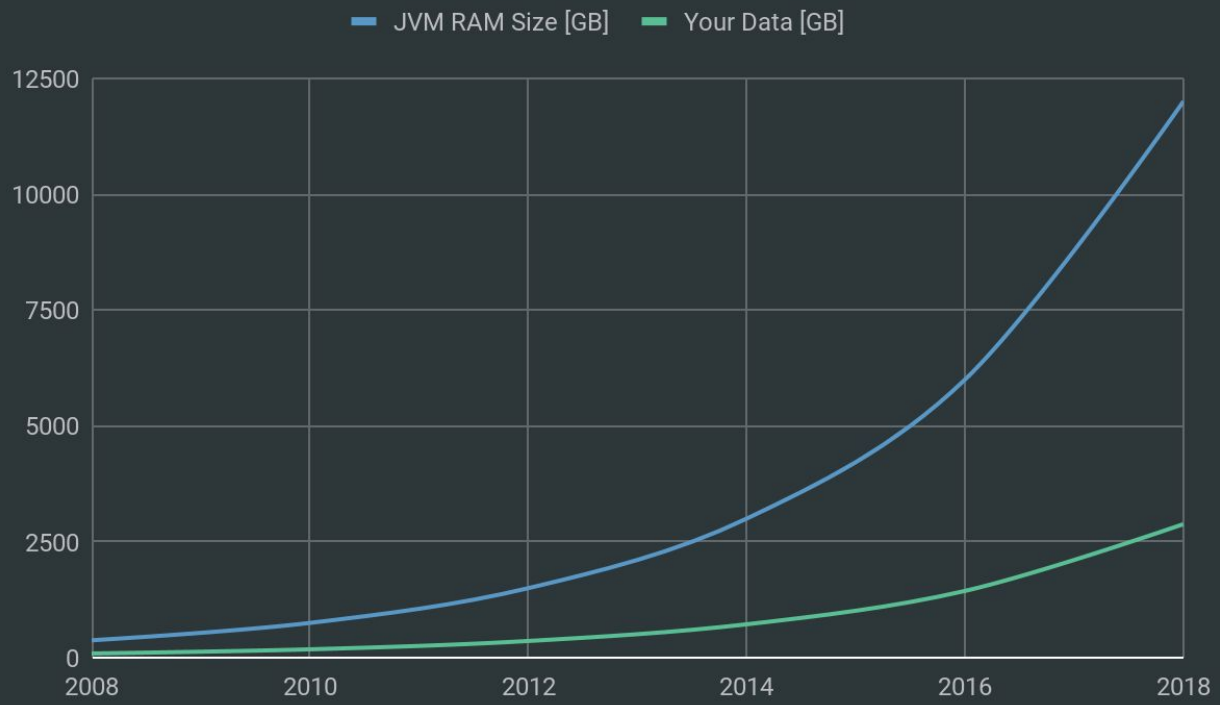


In-JVM-Memory Solution

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

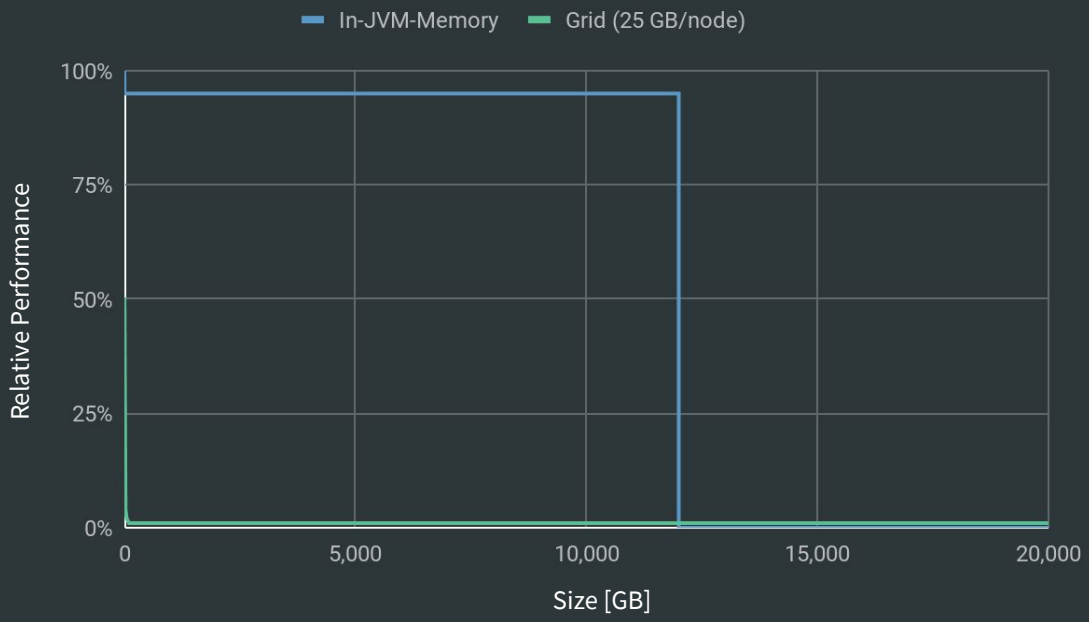


What if My Data Grows?

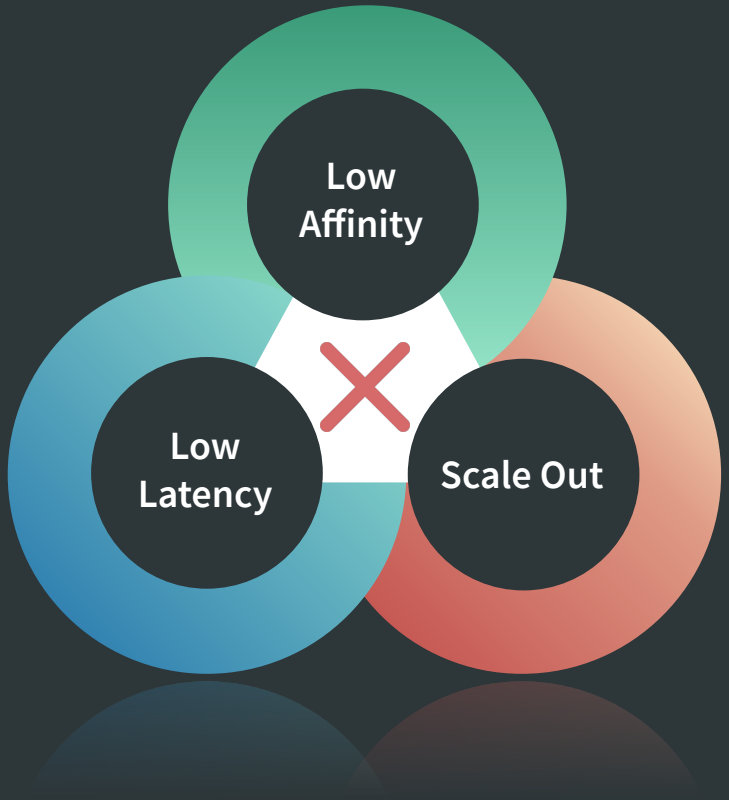


In-JVM-Memory vs. In-Memory Performance

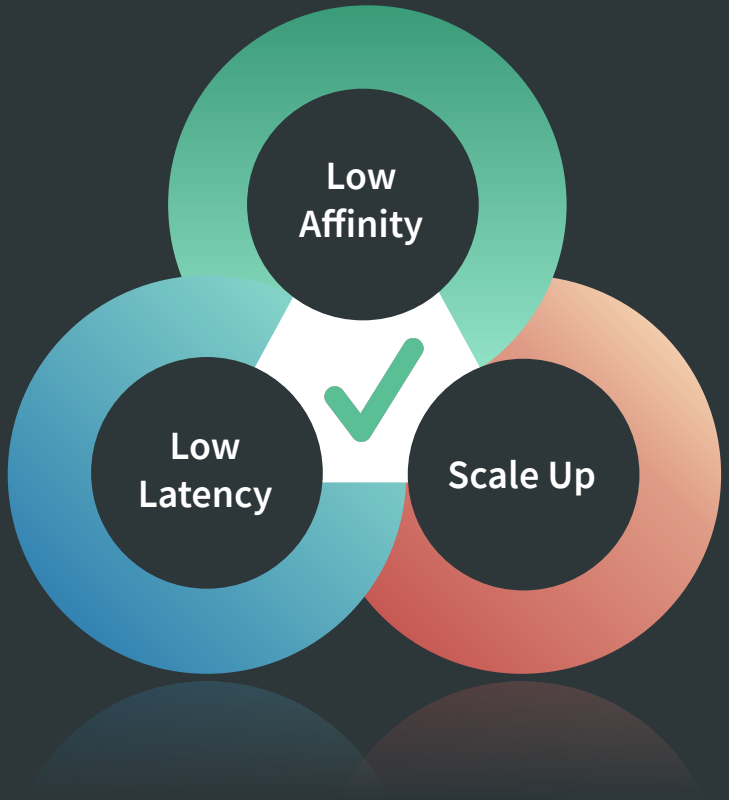
Data with 75% correlation



Recap

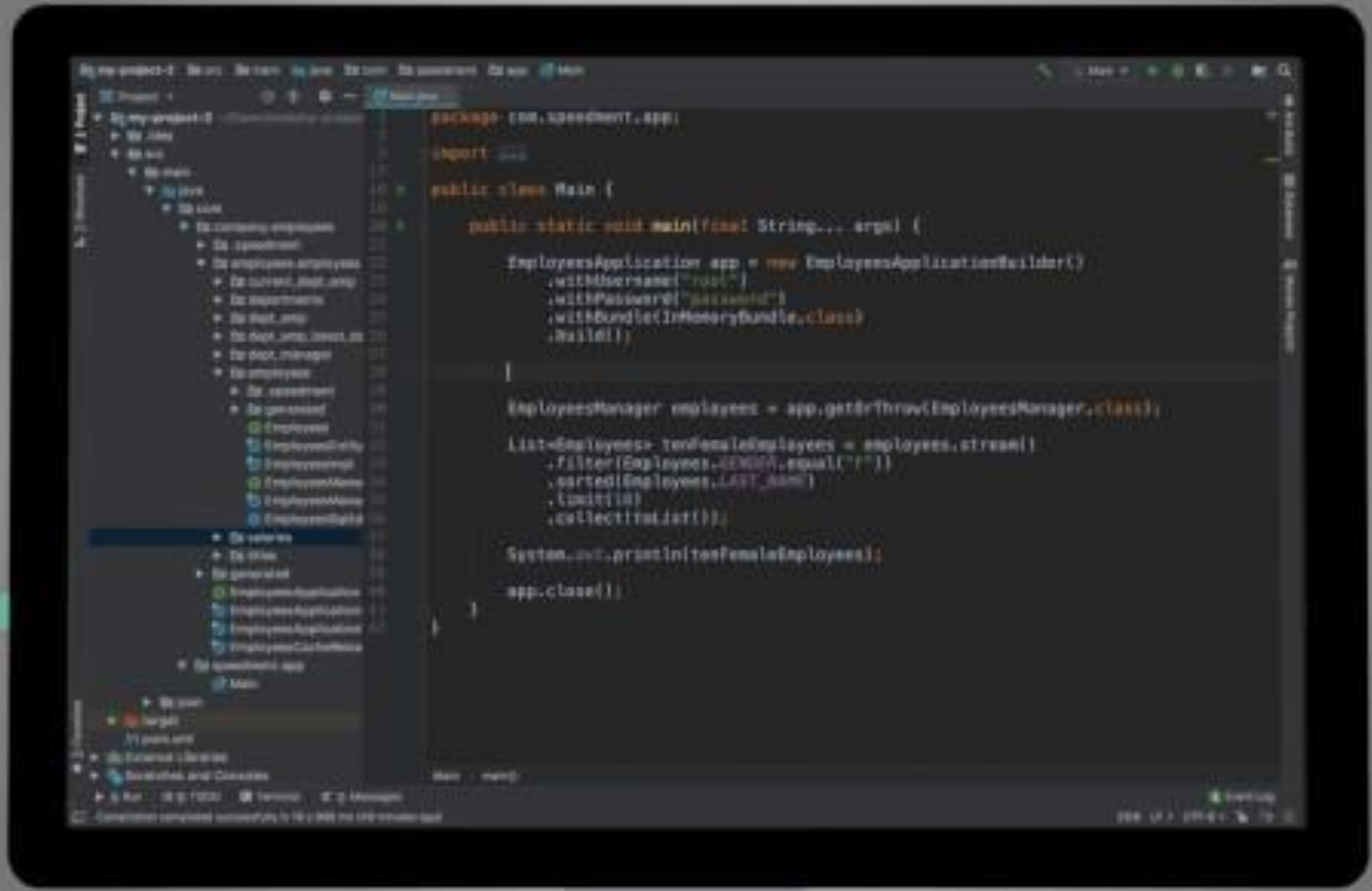


In-JVM-Memory Makes it Possible



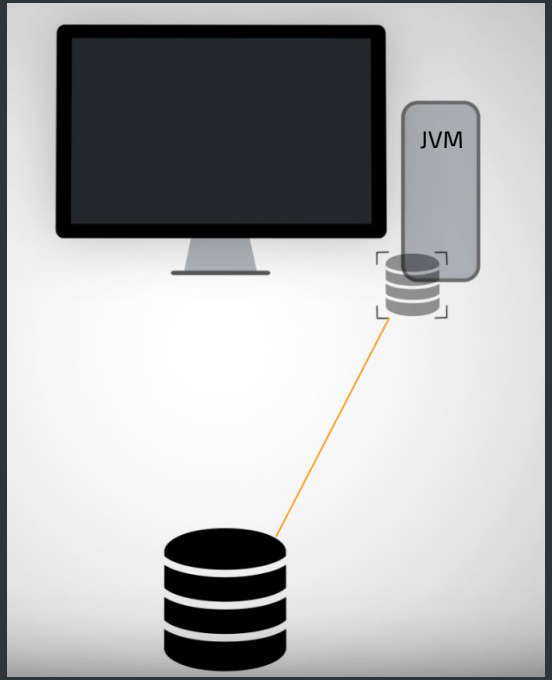
In-JVM-Memory Solution:

Speedment

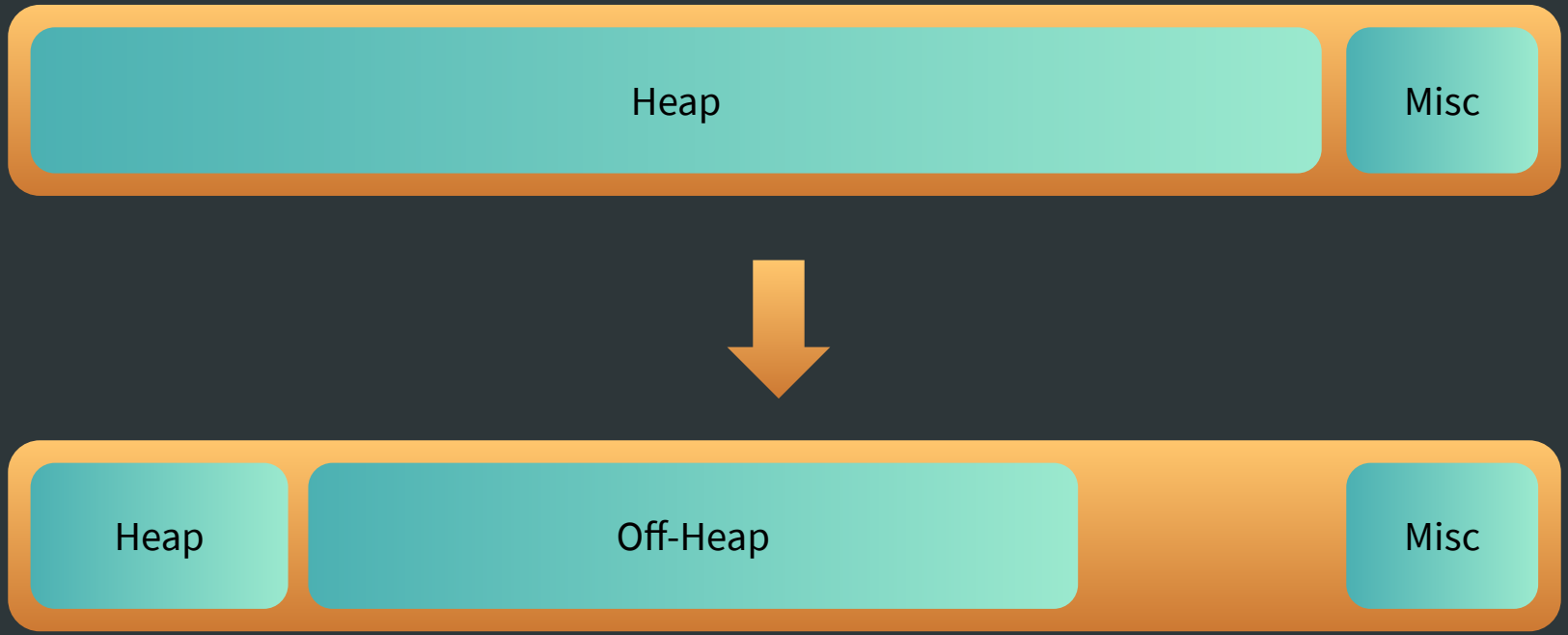

package com.speedment.app;
import ...
public class Main {
public static void main(final String... args) {
EmployeesApplication app = new EmployeesApplicationBuilder()
.withUsername("root")
.withPassword("password")
.withBundle(InMemoryBundle.class)
.build();
EmployeesManager employees = app.getOrThrow(EmployeesManager.class);
List<Employees> tenFemaleEmployees = employees.stream()
.filter(Employees.GENDER.equal("F"))
.sorted(Employees.LAST_NAME)
.limit(10)
.collect(toList());
System.out.println(tenFemaleEmployees);
app.close();
}
}

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



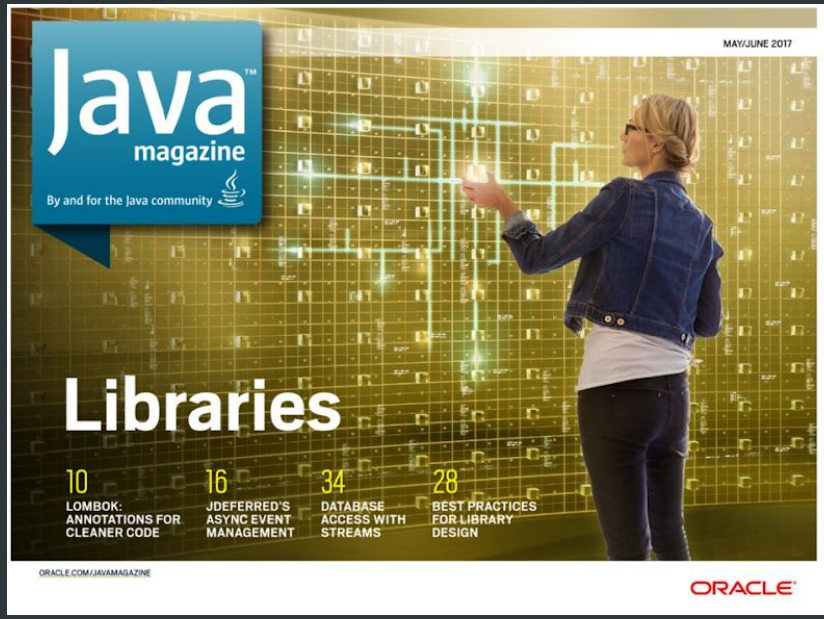
Speedment: In-JVM-Memory DataStore

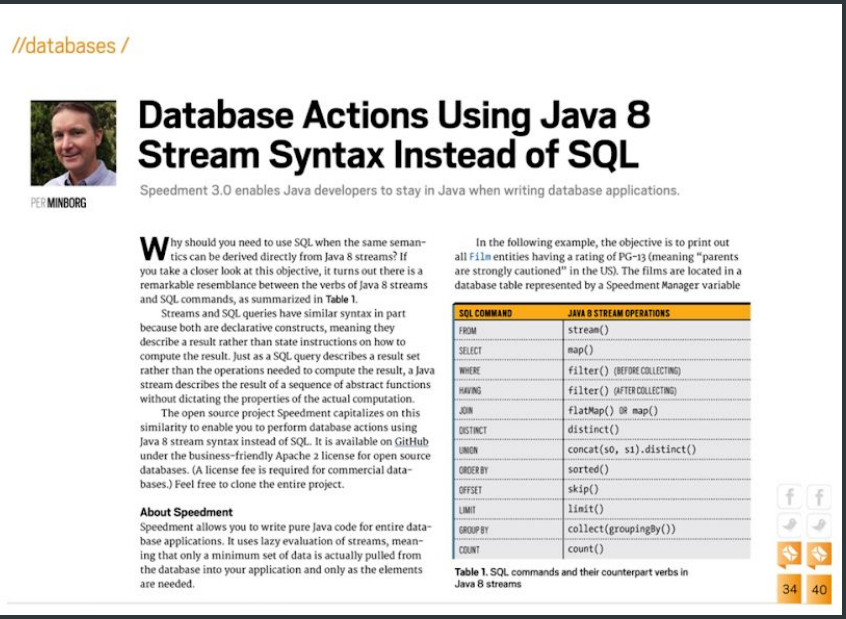


Speedment API: Java Stream ORM

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

Speedment API: Java Stream ORM





//databases /

Database Actions Using Java 8 Stream Syntax Instead of SQL

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

PER MINBORG

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

34 40

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();
```

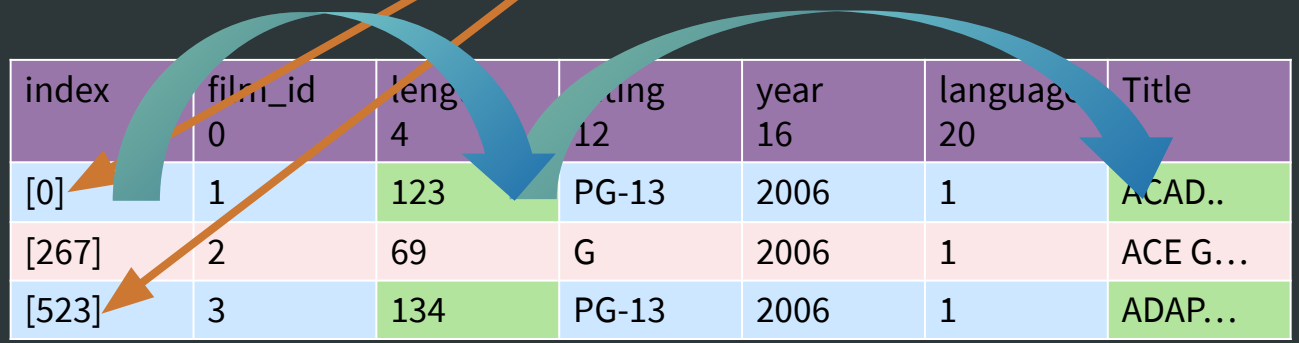
Process Data without Creating Intermediate Objects

```
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 0	leng 4	ating 12	year 16	language 20	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

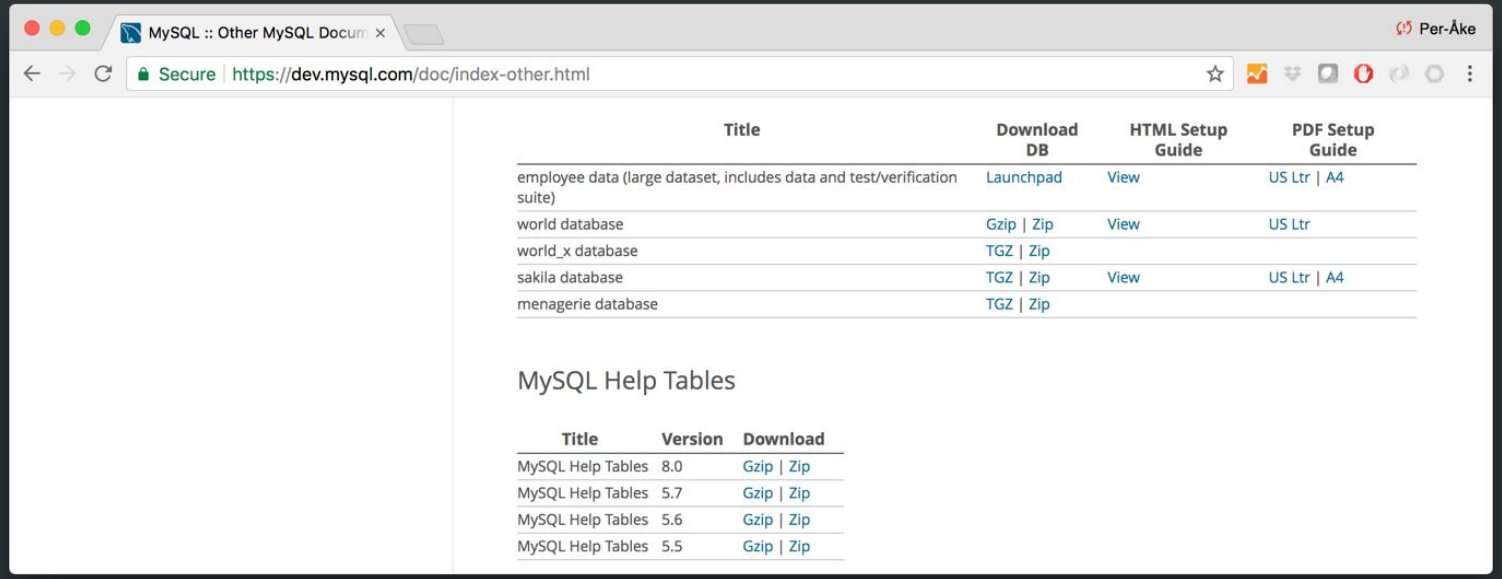
```
$ nproc –all
32

$ 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



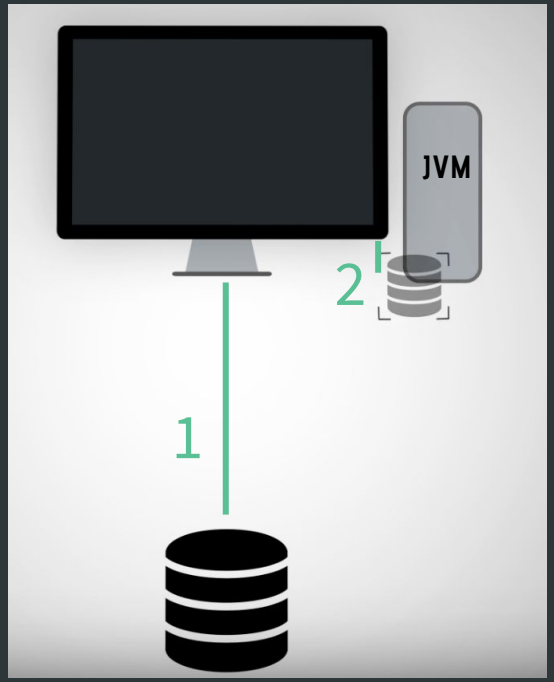
Title	Download DB	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	Launchpad	View	US Ltr | A4
world database	Gzip | Zip	View	US Ltr
world_x database	TGZ | Zip		
sakila database	TGZ | Zip	View	US Ltr | A4
menagerie database	TGZ | Zip		

MySQL Help Tables

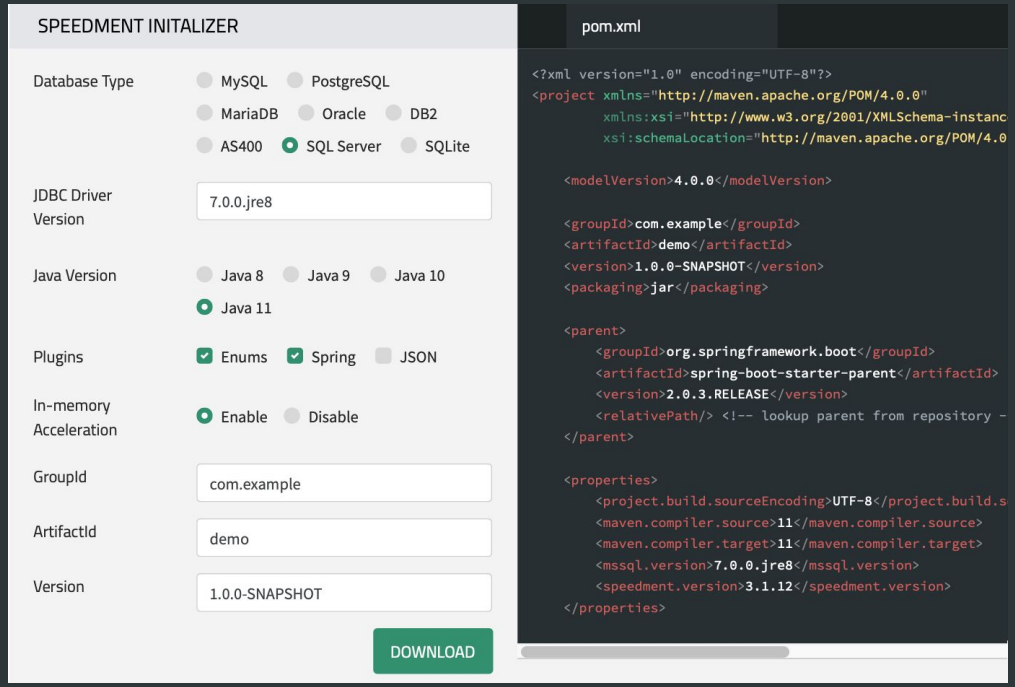
Title	Version	Download
MySQL Help Tables	8.0	Gzip | Zip
MySQL Help Tables	5.7	Gzip | Zip
MySQL Help Tables	5.6	Gzip | Zip
MySQL Help Tables	5.5	Gzip | Zip

Demo: Stream

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

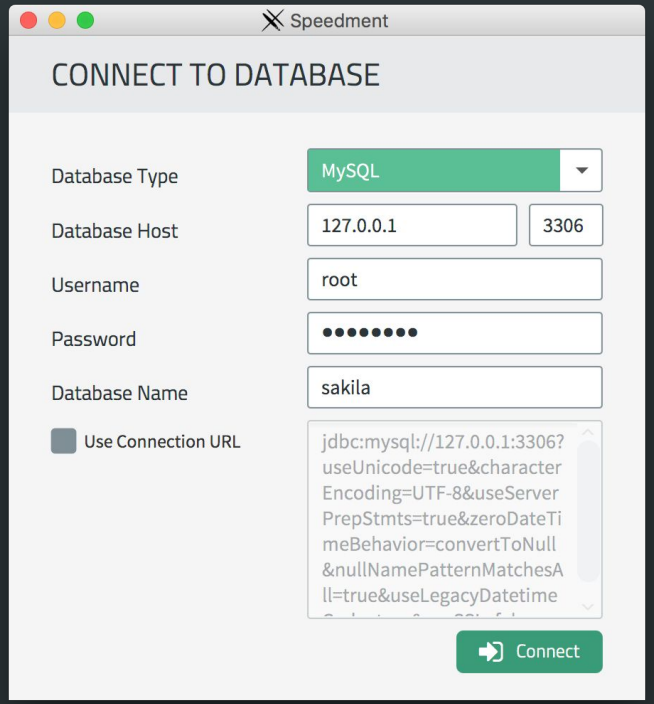


Demo: Initialize the Project

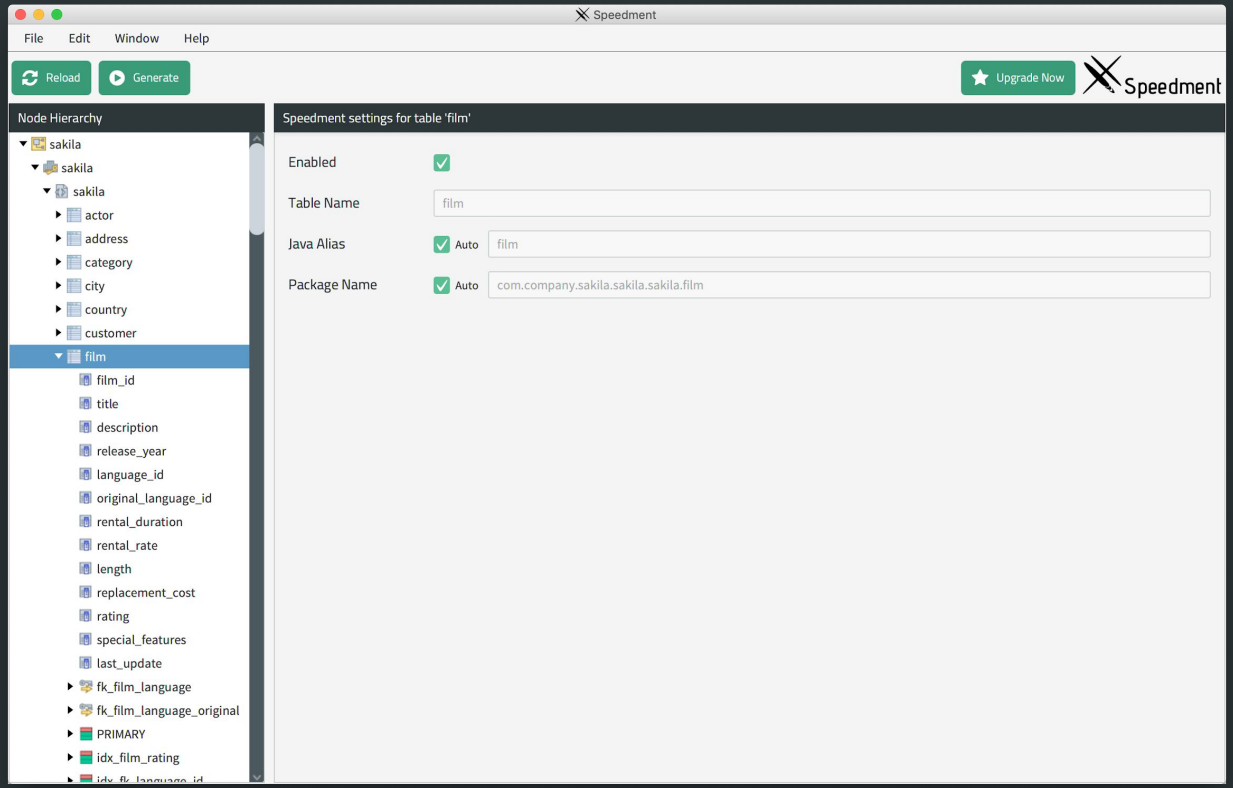


www.speedment.com/initializer

Demo: Connect to the Sakila Database

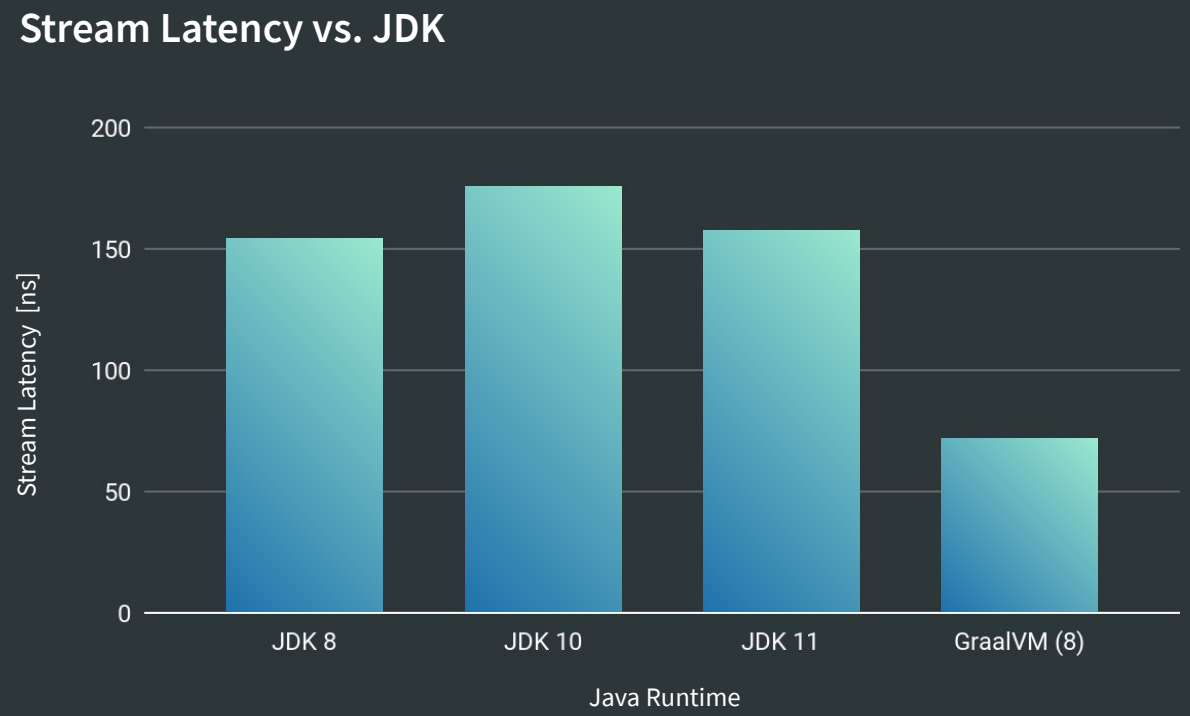


Demo: Generate the Domain Model



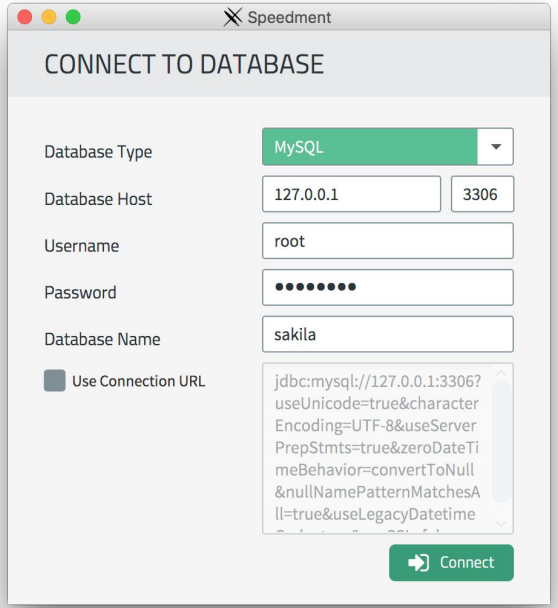
Demo

Ultra-Low Latency (Lower is Better)

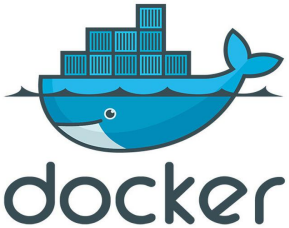


Use Existing Infrastructure

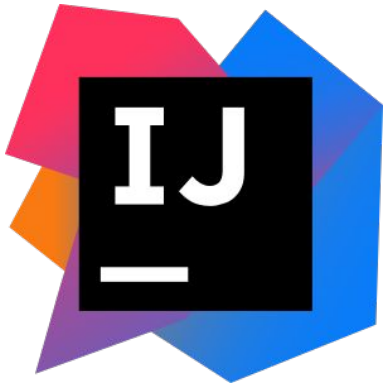
Easy Integration: Any Data Source



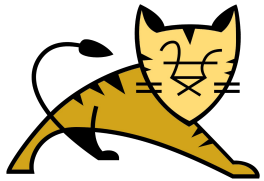
Deploy Anywhere



IDE Integration



Web Service Integration



Thanks!

E-mail
minborg@speedment.com

Free Trial:
www.speedment.com/initializer

Speedment

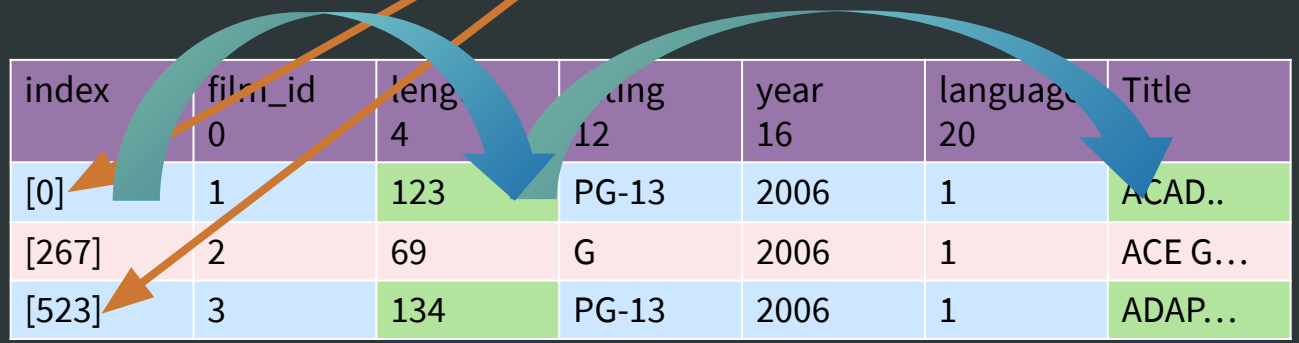
But wait...

- What if I have a transactional application?
- How to pin down my app to a specific CPU?
- How does memory management work off-heap?
- I have data in a no-SQL database or on file?
- How does the snapshot affect application startup?
- If I stream over a kazillion objects, won't my heap overflow?
- Does Speedment rely on C, C++ libraries or native code?

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 0	leng 4	ating 12	year 16	language 20	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

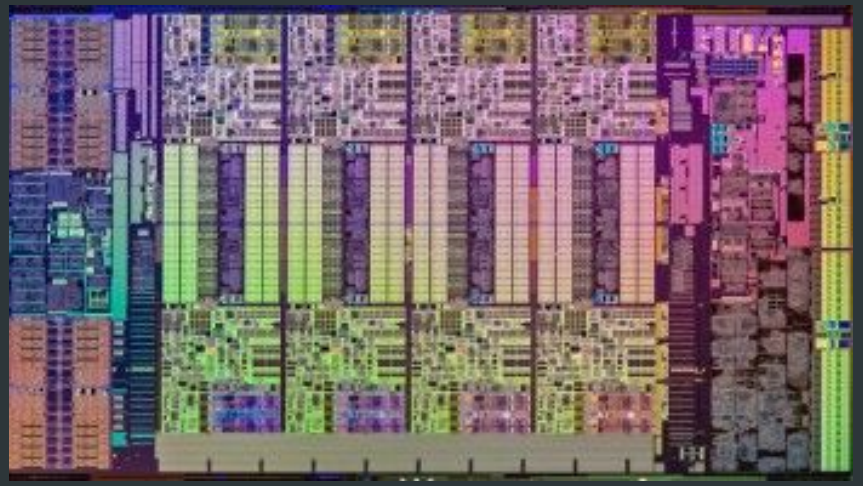
In-JVM-Memory Makes Ultra-Low Latency Possible

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();
```