What's new in Java >17

A lot!

Java 18

UTF-8 is default character set

meaning: when no explicit charset is specified, UTF-8 is used

UTF-8 default character set

before locale was OS dependent

```
java.io.FileReader(“hello.txt”) -> “こんにちは” (macOS)
java.io.FileReader(“hello.txt”) -> “ã?“ã,“ã?«ã?¡ã?  ” (Windows (de-AT))
java.io.FileReader(“hello.txt”) -> “縺ォ縺。縺ッ” (Windows (ja-JP)
```

UTF-8 default character set

or specified via JVM properties

```
-Dfile.encoding=UTF-8
```

UTF-8 default character set

Can produce problems with non UTF-8 files -> test your apps!

See JEP 400 for more details

Java 18 - JEP 413: Code Snippets in Java API Documentation

neue Möglichkeit, Code Snippets in JavaDoc zu verwenden

- entweder Inline oder
- in externen Dateien

Java 18 - Inline Code Snippets

new inline tag, `{@snippet ...}`

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet :
 *   if (v.isPresent()) {
 *       System.out.println("v: " + v.get());
 *   }
 * }
 */
```

Java 18 - externe Code Snippets

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet file="ShowOptional.java" region="example"}
 */
```

Java 18 - externe Code Snippets

In Unterordner `snippet-files` -> `ShowOptional.java`

```java
public class ShowOptional {
    void show(Optional<String> v) {
        // @start region="example"
        if (v.isPresent()) {
            System.out.println("v: " + v.get());
        }
        // @end
    }
}
```

Java 18: jwebserver

einfacher Webserver um einfach statische Dateien zu serven.

```
$JAVA_HOME/bin/jwebserver - port 8000
$JAVA_HOME/bin/jwebserver -p 9000
$JAVA_HOME/bin/jwebserver -b 192.168.123.40 -p 9000
$JAVA_HOME/bin/jwebserver -d src/
```

Java 19 + 20: Only incubating + previews

we don't cover them, as they are already outdated

Java 21 - LTS

Released on September 19th 2023

EOL September 30th 2031

Latest version 21.0.1 (October 17th 2023)

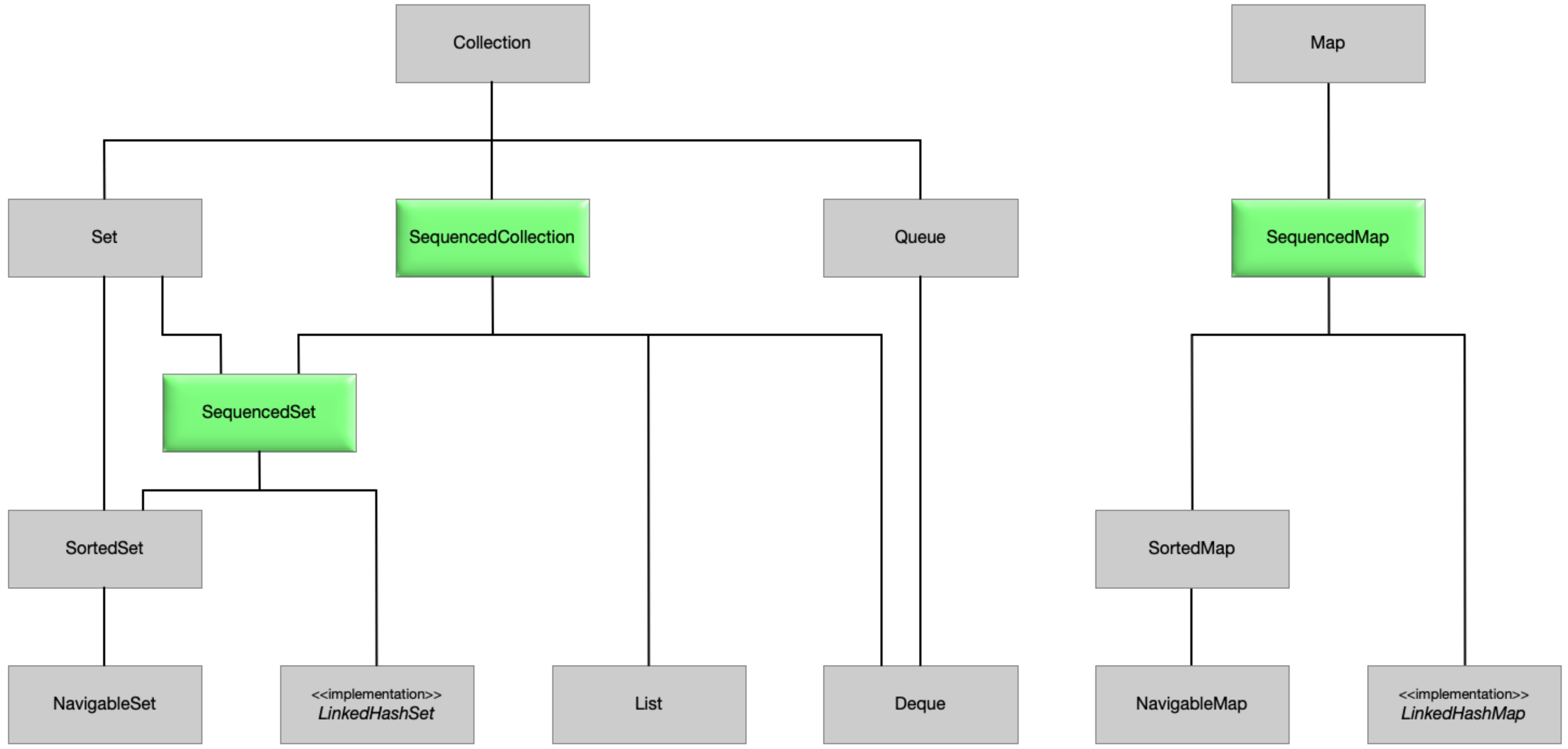
String Templates

Finally there (as preview)!

- JEP 430: String Templates
- `STR` always imported, `RAW` and `FMT` have to be imported.
- custom Processors (e.g. `SQL`, `JSON`, `URL`) only a matter of time (IMHO)

Sequenced Collections

- JEP 431: Sequenced Collections
- Integrated in the default collection hierarchy
- Can create problems with custom collection implementations


Collection
Map
Set
SequencedCollection
Queue
SequencedMap
SequencedSet
SortedSet
SortedMap
NavigableSet
<<implementation>> LinkedHashSet
List
Deque
NavigableMap
<<implementation>> LinkedHashMap

SequencedCollection<E>

```java
public interface SequencedCollection<E> extends Collection<E> {
    SequencedCollection<E> reversed();
    default void addFirst(E e);
    default void addLast(E e);
    default E getFirst();
    default E getLast();
    default E removeFirst();
    default E removeLast();
}
```

SequencedMap<K, V>

```
public interface SequencedMap<K, V> extends Map<K, V> {
    SequencedMap<K, V> reversed();
    default Map.Entry<K,V> firstEntry();
    default Map.Entry<K,V> lastEntry();
    default Map.Entry<K,V> pollFirstEntry();
    default Map.Entry<K,V> pollLastEntry();
    default V putFirst(K k, V v);
    default V putLast(K k, V v);
    default SequencedSet<K> sequencedKeySet();
    default SequencedCollection<V> sequencedValues();
    default SequencedSet<Map.Entry<K, V>> sequencedEntrySet();
}
```

SequencedSet<E>

```java
public interface SequencedSet<E> extends SequencedCollection<E>, Set<E> {
    SequencedSet<E> reversed();
}
```

Sequenced Collections (summary)

- List now has SequencedCollection as its immediate superinterface,
- Deque now has SequencedCollection as its immediate superinterface,
- LinkedHashSet additionally implements SequencedSet,
- SortedSet now has SequencedSet as its immediate superinterface,

Record Patterns

-> see the code

Records & Switch

switch syntax changed slightly, from && to `when`

-> see code samples

(preview) JEP 443: Unnamed Patterns and Variables

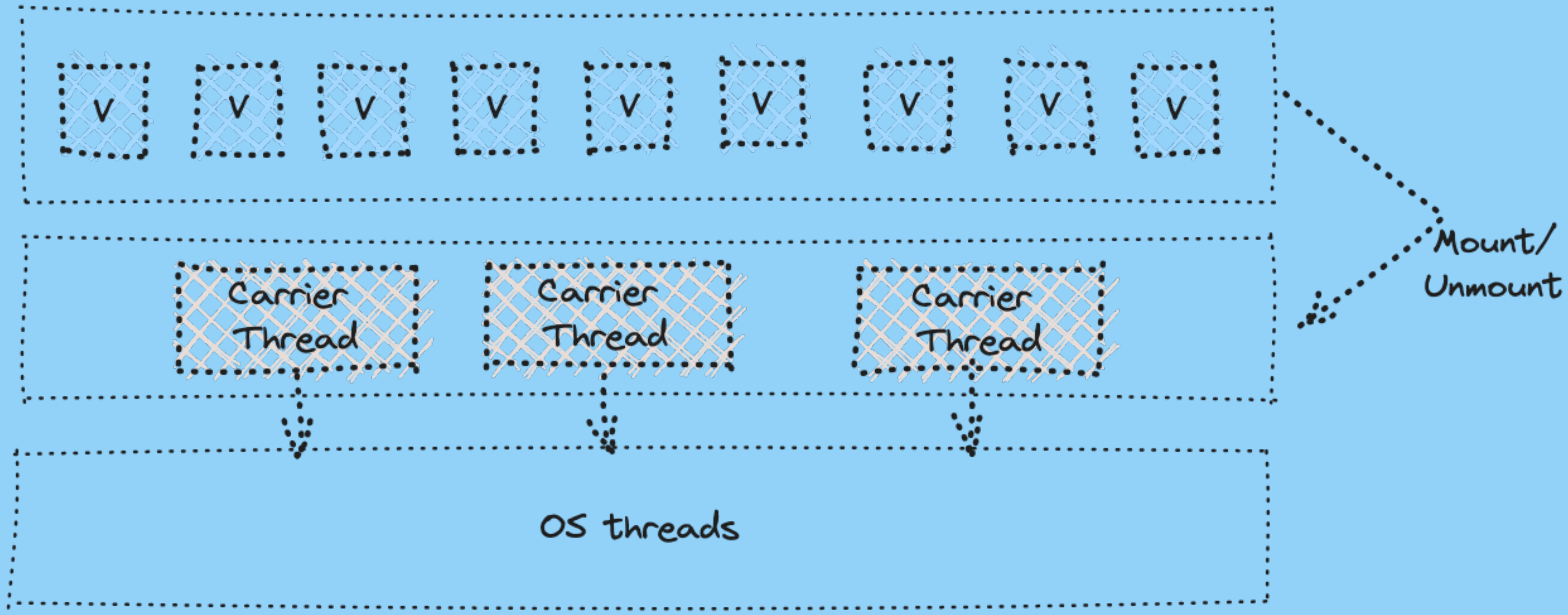
- TLDR: _ is now used for stuff you don't care about
- see code samples

Virtual Threads / Loom

- klassisch: 1 OS Thread per JVM Thread
  - hohe Kosten für OS Threads
  - Umgehungskonstrukte:
    - Thread-Pools
    - Reactive-Programming: CompletableFuture, RxJava, Project Reactor, Akka, Vert.x, ...

Virtual Threads / Loom

- neu: Carrier Threads
  - 1 OS Thread per Carrier Thread
  - Scheduler zwischen Carrier Threads und virtuellen Threads (Green Threads, Fibers)
  - Scheduler erkennt blockierende Aufrufe und wechselt zu anderem virtuellen Thread


V
V
V
V
V
V
V
V
V
Mount/
Unmount
Carrier Thread
Carrier Thread
Carrier Thread
OS threads

Virtual Threads / Loom

- Vorteile:
  - weniger OS Threads notwendig
  - bessere Auslastung bei blockierenden Aufrufen (z.B. IO, Thread.sleep(), ...)
  - weniger Overhead bei "Thread"-Wechsel

Virtual Threads / Loom

- Nachteile:
  - kann nur bei blockierenden Aufrufen wechseln
  - SpinLoops blockieren auch virtuelle Threads -> Carrier Threads

```
while(true) {
// do nothing
}
```

Virtual Threads / Loom

Workarounds

```java
while(true) {
  Thread.sleep(1); // blockiert (nicht wirklich), aber gibt anderen virtuellen Thread die Chance
}
```

Virtual Threads / Loom

Workarounds

```
while(true) {
  Thread.onSpinWait(); // signalisiert, dass hier ein wechsel möglich ist
}
```

Virtual Threads / Loom

Threads erstellen - klassisch

klassisch:

```
new Thread(runnable).start();
```

Virtual Threads / Loom

Threads erstellen - mit `ThreadBuilder`

```java
Thread.Builder threadBuilder = Thread.ofPlatform();
```

Virtual Threads / Loom

Threads erstellen - mit `ThreadBuilder`

für virtuelle Threads:

```java
Thread.Builder threadBuilder = Thread.ofVirtual();
```

Virtual Threads / Loom

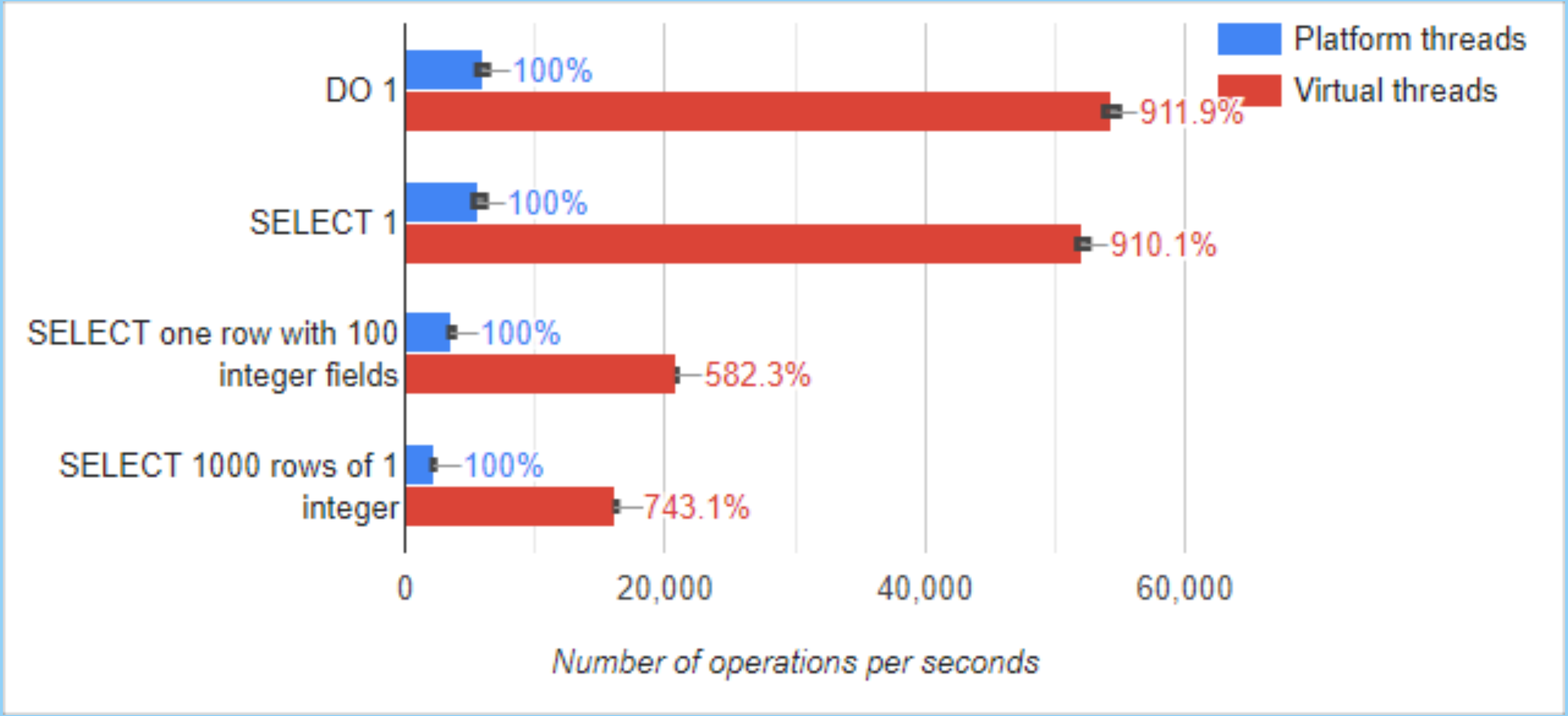
Payload ausführen

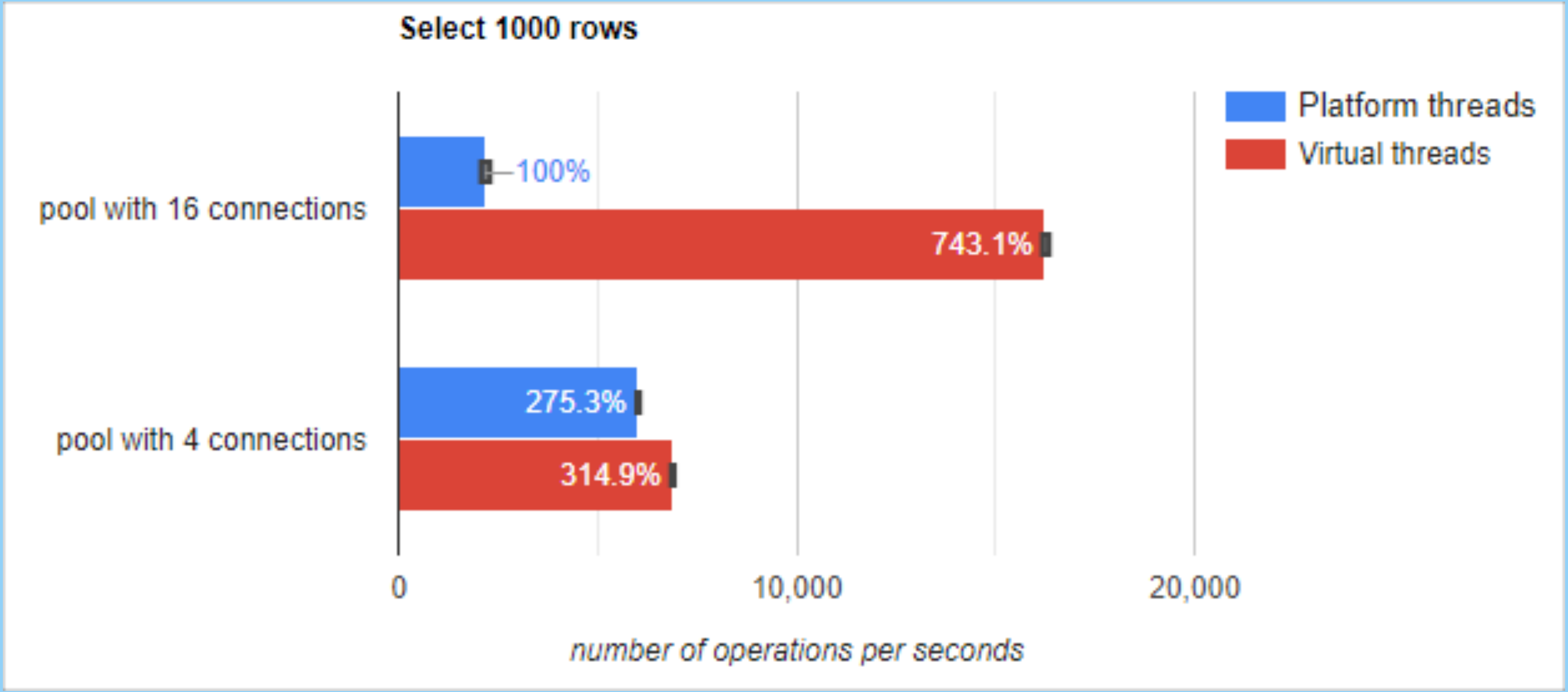
```
builder.start(() -> {
System.out.println( "bin ich virtuell?: " + Thread.currentThread().isVirtual());
});
```

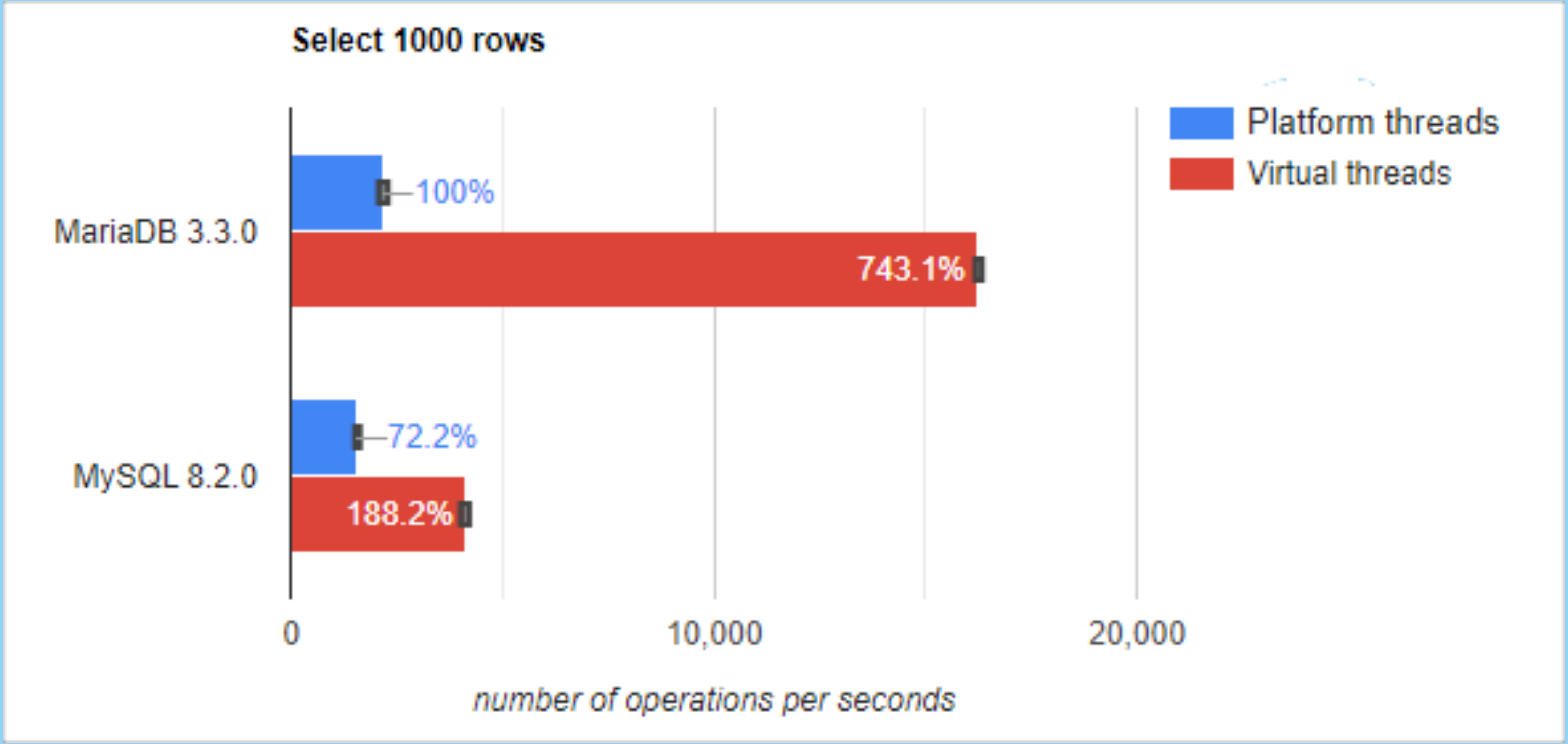
Virtual Threads / Benchmarks

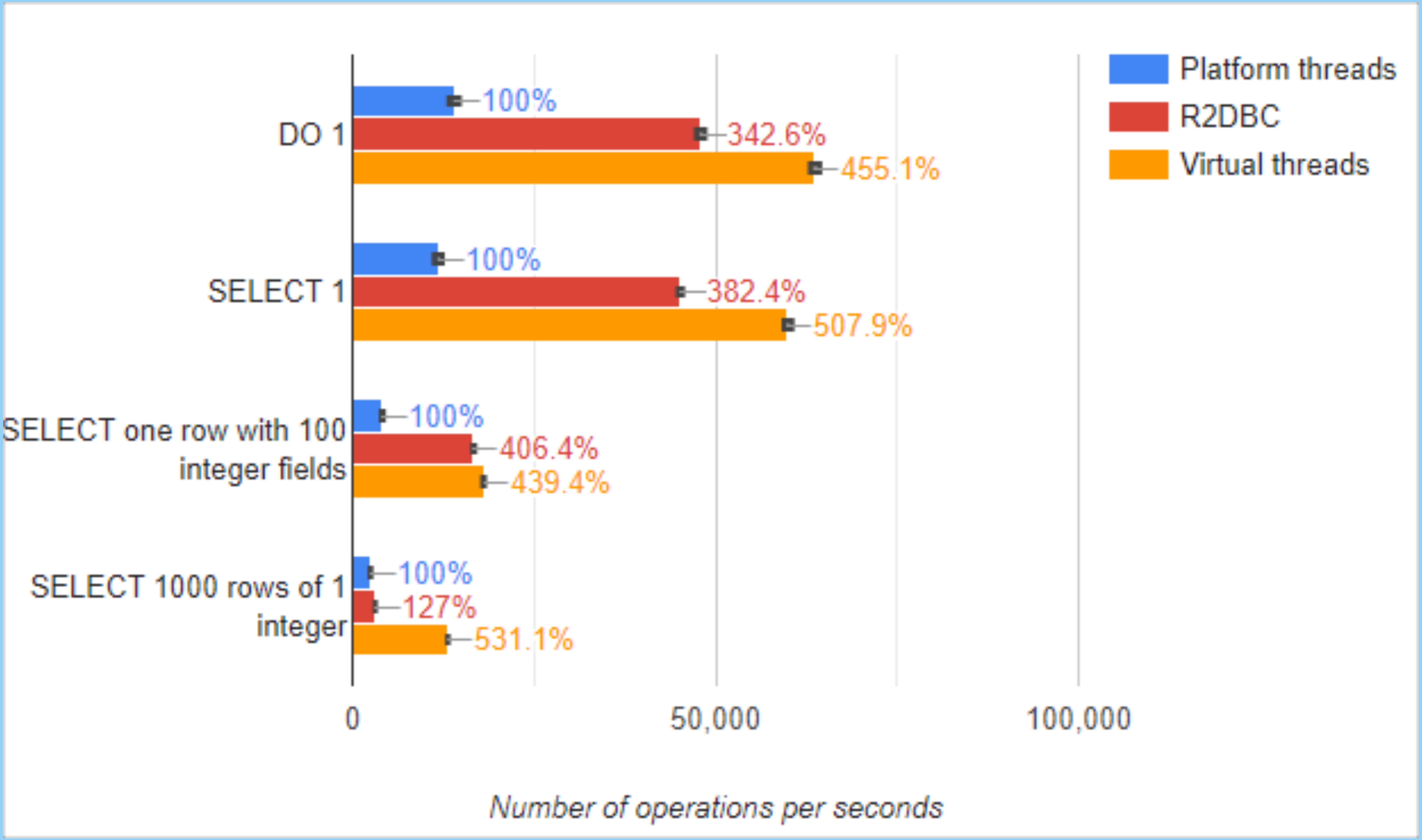
by mariadb

https://mariadb.com/resources/blog/benchmark-jdbc-connectors-and-java-21-virtual-threads/


Platform threads
Virtual threads
DO 1
SELECT 1
SELECT one row with 100 integer fields
SELECT 1000 rows of 1 integer
100%
911.9%
100%
910.1%
100%
582.3%
100%
743.1%
0
20,000
40,000
60,000
Number of operations per seconds


Select 1000 rows
Platform threads
Virtual threads
pool with 16 connections
pool with 4 connections
100%
743.1%
275.3%
314.9%
0
10,000
20,000
number of operations per seconds


Select 1000 rows
Platform threads
Virtual threads
MariaDB 3.3.0
100%
743.1%
MySQL 8.2.0
72.2%
188.2%
0
10,000
20,000
number of operations per seconds


Platform threads
R2DBC
Virtual threads
DO 1
100%
342.6%
455.1%
SELECT 1
100%
382.4%
507.9%
SELECT one row with 100 integer fields
100%
406.4%
439.4%
SELECT 1000 rows of 1 integer
100%
127%
531.1%
0
50,000
100,000
Number of operations per seconds

Structured Concurrency (Preview)

-> Demo

Sources

- Java Aktuell 01-2024
- RedHat Developers: https://developers.redhat.com/articles/2023/09/21/whats-new-developers-jdk-21