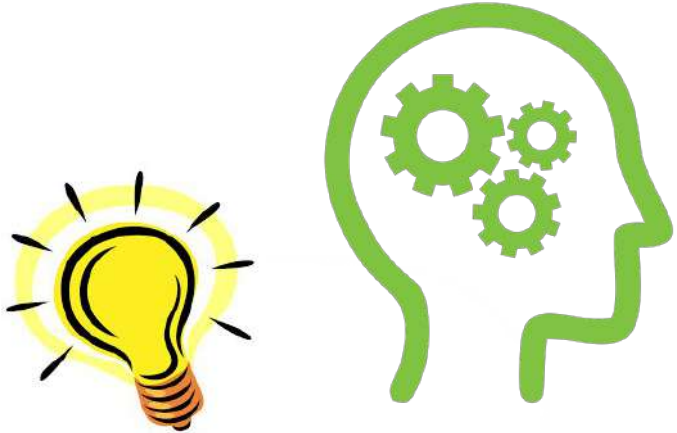
Scala vs Elixir

Ludwik Bukowski
Kacper Mentel

Erlang SOLUTIONS

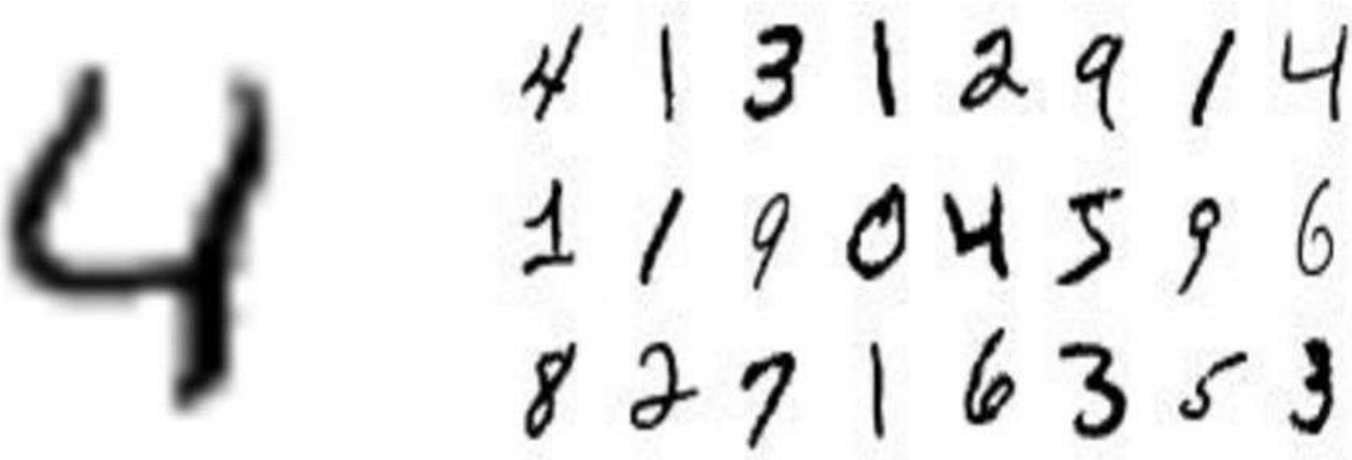
San Francisco
2020



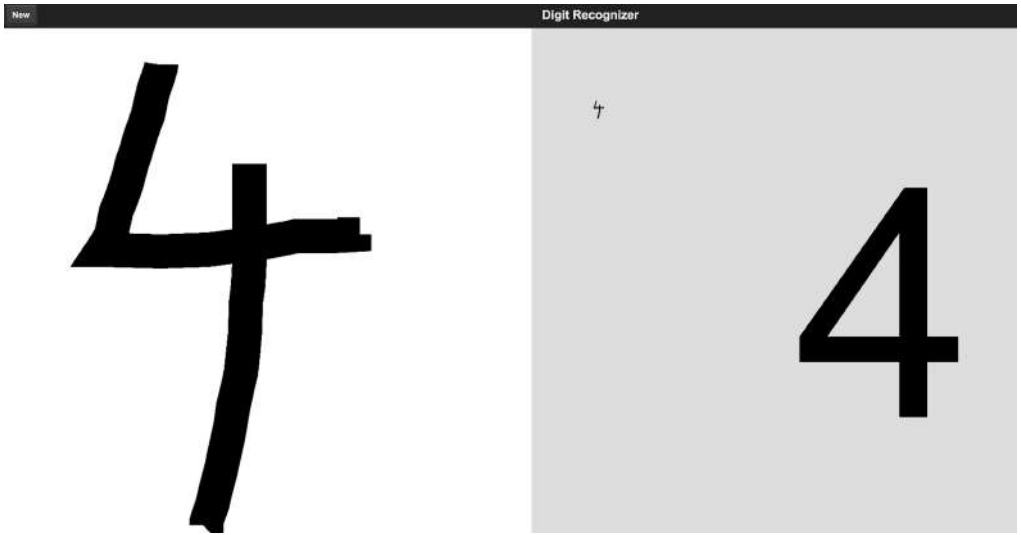
Scala

elixir

The problem

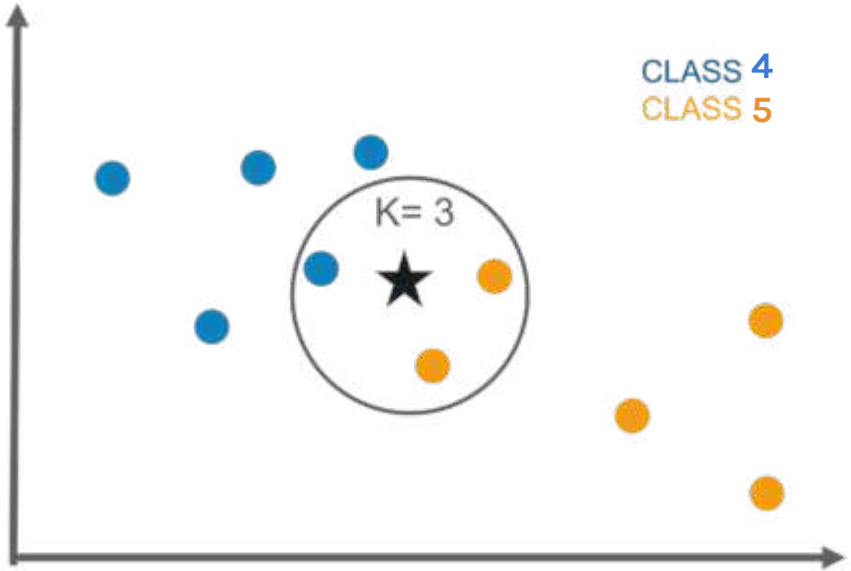


The Application



0000000000000000111100000000000
0000000000000000111111000000000
0000000000000001111110000000000
0000000000000001111110000000000
0000000000000001111110000000000
0000000000000011111000000010000
0000000000000011111000000011000
0000000000001111110000011110000
0000000000001111110000011110000
0000000000001111110000011110000
0000000001111100000001111100000
0000000001111100000001111100000
0000000011111000000001111110000
0000001111111111111111111100000
0000011111111111111111111100000
0000011111111111111111111100000
0000011111111111111111111110000
0000011111111111111111111100000000
0000001111111111111111111000000000
0000001111111100111111000000000
0000000000000000001111100000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011111000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000

The Algorithm



The Algorithm - implementation

Input vector

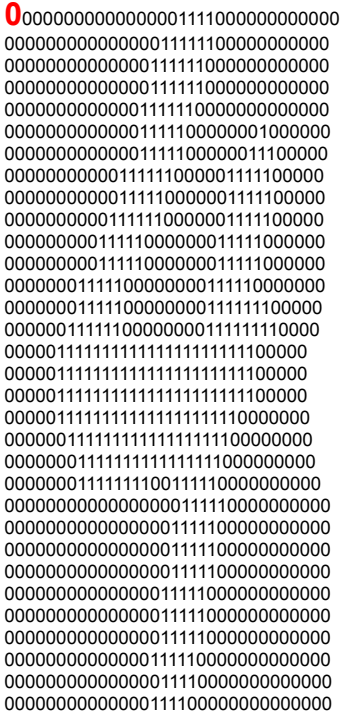
```
00000000000000000111100000000000
00000000000000000111111000000000
00000000000000011111110000000000
00000000000000011111110000000000
00000000000000111111100000000000
00000000000000111111000000010000
00000000000000111111000000111000
00000000000011111110000011110000
00000000000011111100000011111000
00000000000111111000000011111000
00000000001111110000000011111000
00000000001111110000000011111000
00000000011111100000000111110000
00000000111111000000000111111000
00000001111110000000001111111000
00000011111111111111111111100000
00000111111111111111111111100000
00000111111111111111111111100000
00001111111111111111111111100000
00001111111111111111111110000000
00001111111111111111111100000000
00000011111111111111111100000000
00000011111111100111110000000000
00000000000000000011111000000000
00000000000000000111110000000000
00000000000000000111110000000000
00000000000000001111100000000000
00000000000000001111100000000000
00000000000000001111100000000000
00000000000000011111000000000000
00000000000000011110000000000000
00000000000000111100000000000000
```

reference sample

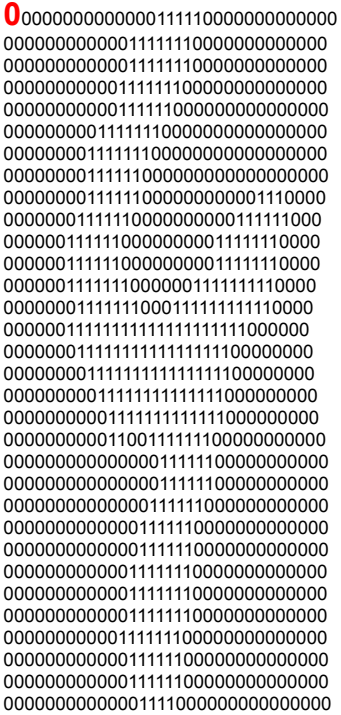
```
00000000000000111110000000000000
00000000000001111111000000000000
00000000000011111111000000000000
00000000000111111110000000000000
00000000001111111100000000000000
00000000011111111000000000000000
00000000111111100000000000000000
00000001111111000000000000000000
00000001111111000000000000001110000
00000011111100000000000111111000
00000111111000000000001111111000
00001111110000000000011111111000
00001111110000000001111111110000
00001111110000000111111111110000
00000111111111111111111111000000
00000011111111111111111100000000
00000000111111111111111000000000
00000000011001111111000000000000
00000000000000011111100000000000
00000000000000011111100000000000
00000000000000111111000000000000
00000000000001111111000000000000
00000000000001111111000000000000
00000000000001111111000000000000
00000000000011111110000000000000
00000000000011111110000000000000
00000000000111111100000000000000
00000000000111111000000000000000
```

The Algorithm - implementation

Input vector



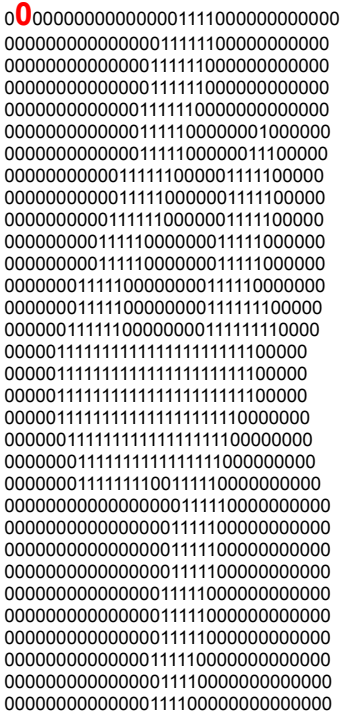
reference sample



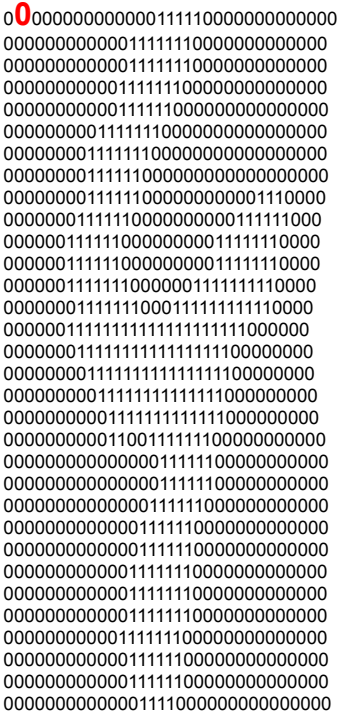
Distance: 0

The Algorithm - implementation

Input vector



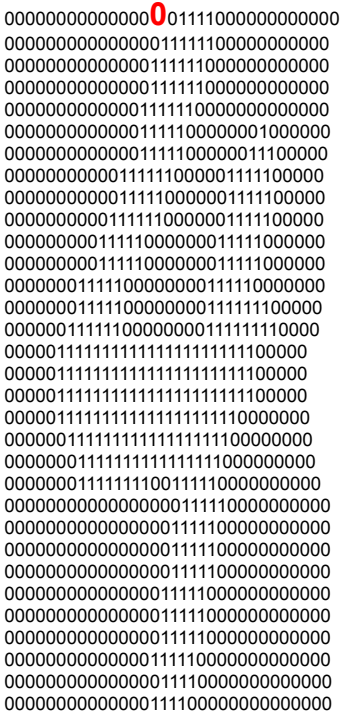
reference sample



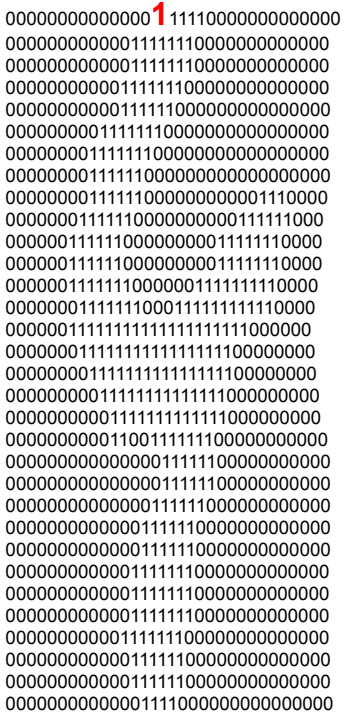
Distance: 0 + 0 = 0

The Algorithm - implementation

Input vector

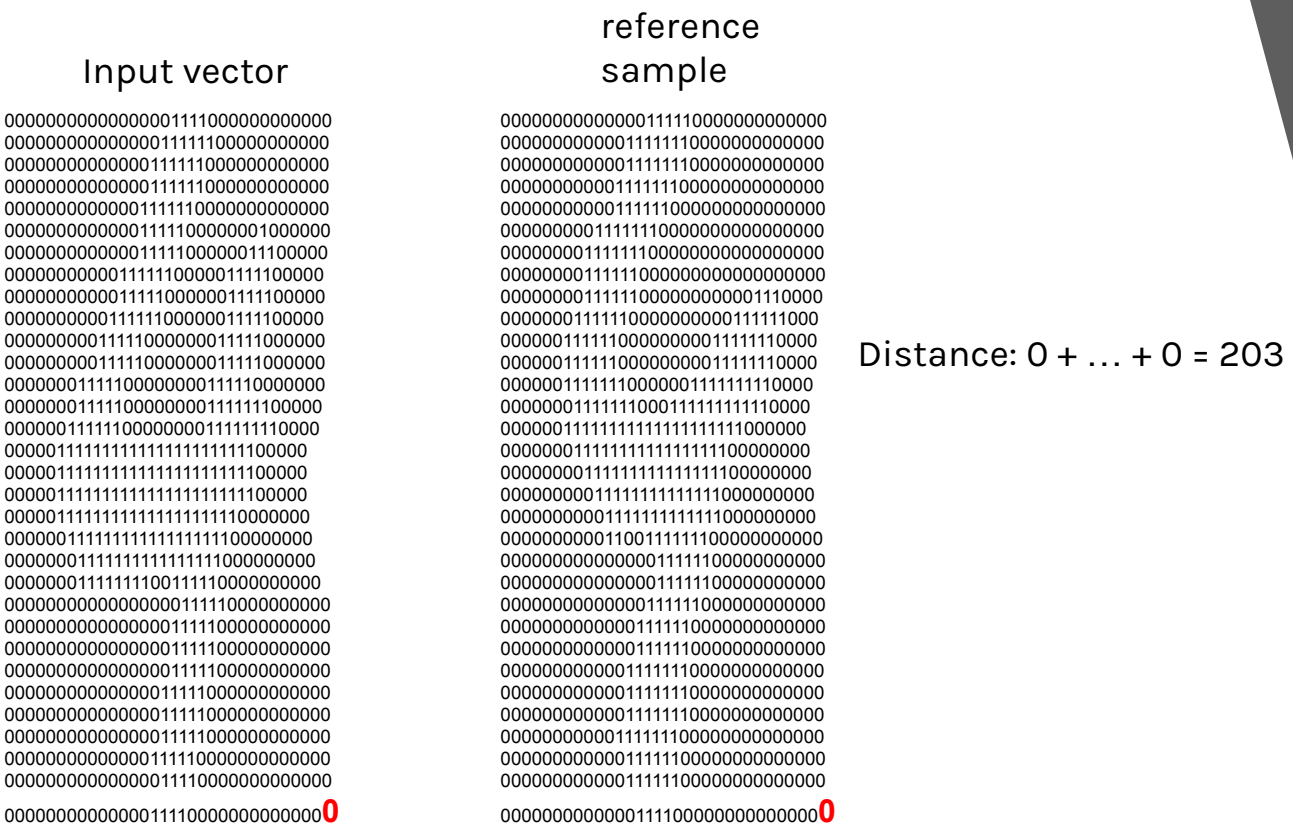


reference sample



Distance: 0 + 0 + … + 1 = 1

The Algorithm - implementation



The Algorithm - implementation

Input vector

```
00000000000000000111100000000000
00000000000000000111111000000000
00000000000000011111110000000000
00000000000000011111110000000000
00000000000000111111100000000000
00000000000000111111000000001000000
00000000000000111111000000011100000
00000000000011111110000011111100000
00000000000011111100000011111100000
00000000001111110000001111100000
00000000011111100000001111100000
00000000111111000000001111110000000
00000001111110000000011111110000
00000011111110000000011111111100000
00000111111111111111111111100000
00000111111111111111111111100000
00000111111111111111111111110000000
00000111111111111111111111100000000
00000011111111111111111111000000000
00000001111111100111110000000000
00000000000000000011111000000000000
00000000000000000111110000000000000
00000000000000000111110000000000000
00000000000000000111110000000000000
00000000000000001111100000000000000
00000000000000001111100000000000000
00000000000000001111100000000000000
00000000000000011111000000000000000
00000000000000011111000000000000000
00000000000000111110000000000000000
```

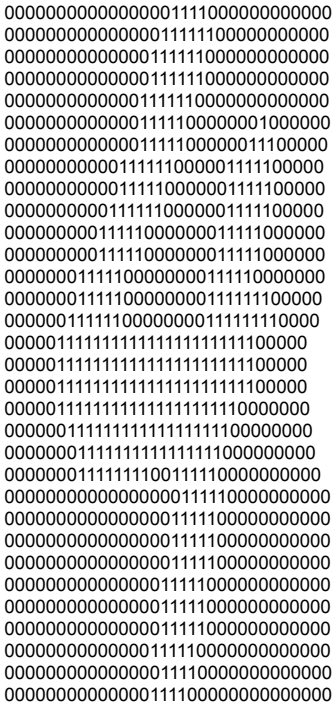
reference sample

```
00000000000000000110000000000000
00000000000000000111000000000000
00000000000000000111100000000000
00000000000000000111100000000000
00000000000000000111100000000000
00000000000000001111000000000000
00000000000000001111000000000000
00000000000000011110000000000000
00000000000000011110000000000000
00000000000001111100000000000000
00000000000001111100000000000000
00000000000001111100000000000000
00000000000001111100000000000000
00000000000011111000000000000000
00000000000011111000000000000000
00000000000011111000000000000000
00000000000011110000010000000000
00000000011111000011100000000000
00000000111110000011110000000000
00000001111100000111100000000000
00000011111100000111110000000000
00000111111100000111111000000000
00001111111110011111111000000000
00001111111111111111111100000000
00001111111111111111111100000000
00001111111111111111111000000000
00001111111111111111111000000000
00001111111111111111111000000000
00000111111111111111111000000000
00000011111111111111111100000000
00000000000000000111111000000000
00000000000000000111111000000000
00000000000000000111111000000000
00000000000000000100000000000000
```

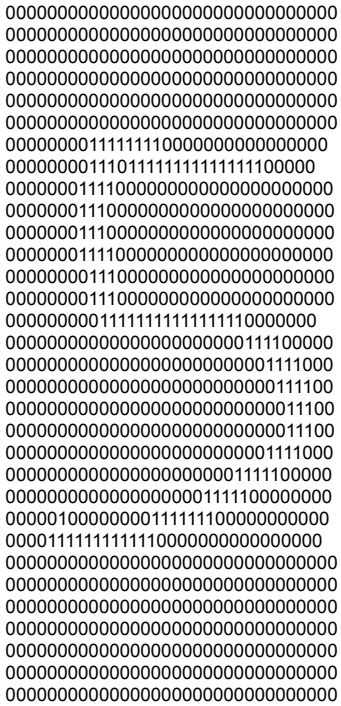
Distance: 257

The Algorithm - implementation

Input vector



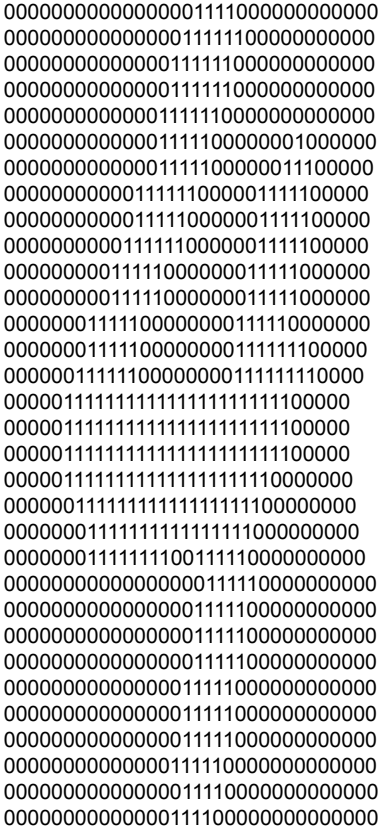
reference sample



Distance: 335

The Algorithm - implementation

- Each training sample contains 1024 binary values
- 2,339 reference samples
- *K = 15*



Concurrency
Scalability

OOP + FE	FE
statically typed	dynamically typed
Runs on JVM	Runs on BEAM

- *Akka-http* version 10.1.9 on Scala 2.13.1
- *SprayJson* parser
- Implemented based on Futures -> Java Thread Pool -> POSIX Thread Pool

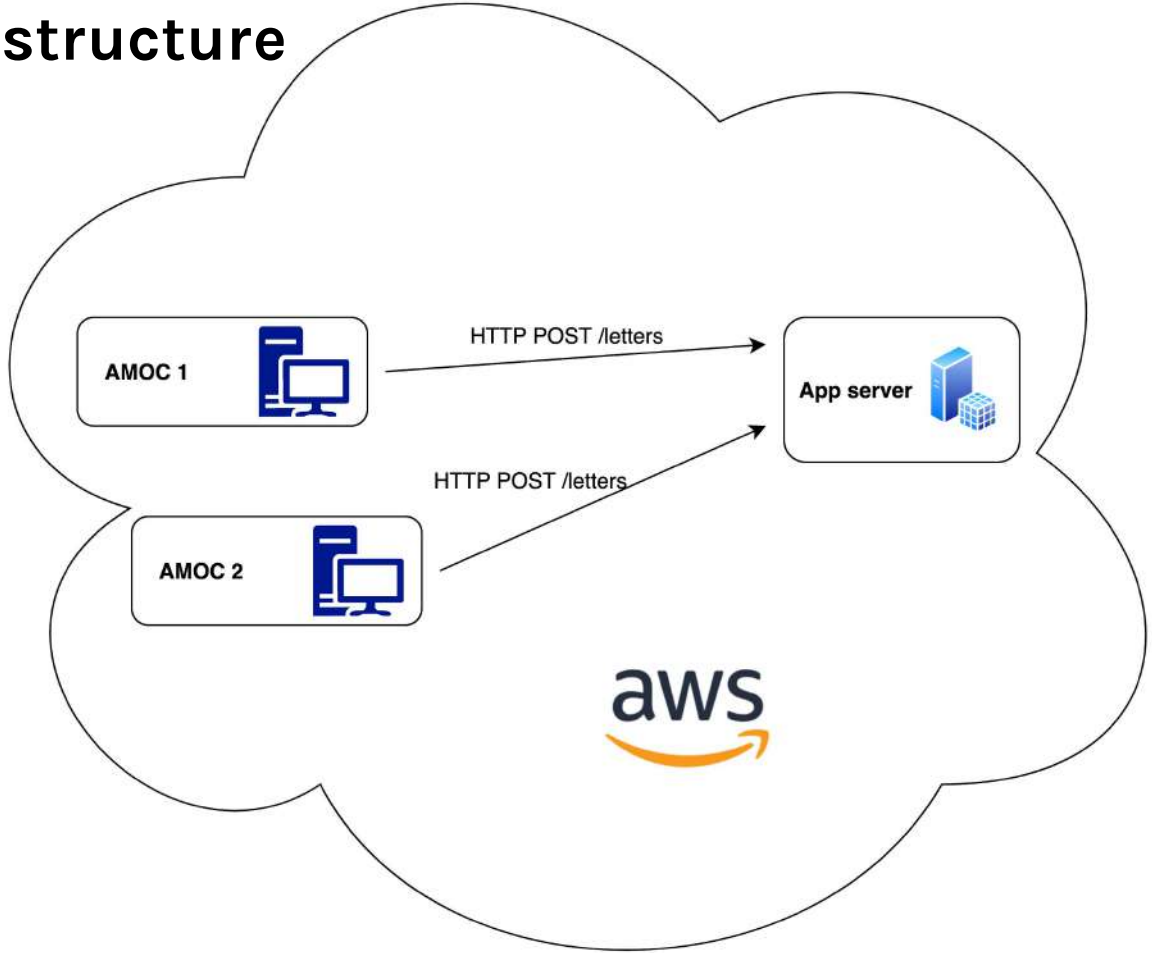
“(...) By default the ExecutionContext.global sets the parallelism level of its underlying fork-join pool to the number of available processors (Runtime.availableProcessors). (...) ”

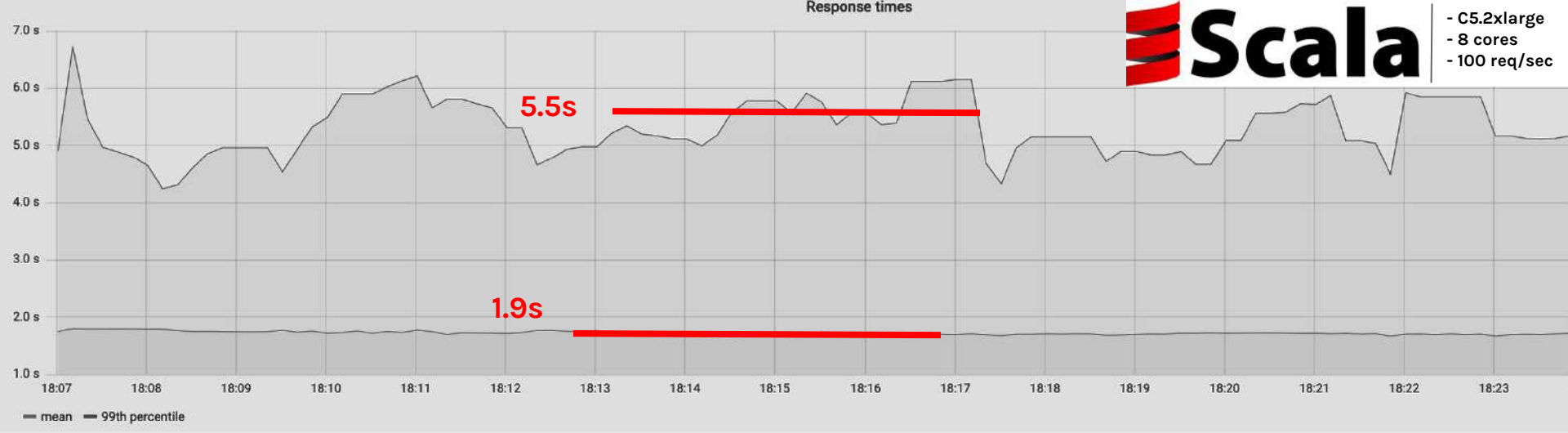
“(...) there is no guarantee it will be called by the thread that completed the future or the thread which created the callback. Instead, the **callback is executed by some thread**, at some time after the future object is completed. “

~ Scala documentation: https://docs.scala-lang.org/overviews/core/futures.html

- *Phoenix* version 1.4.9 on *Elixir* 1.9
- *Jason* parser
- Implemented using *Task.async_stream/1* -> Separate process for each distance calculation between two samples
- Implementation with separate fixed-size process pool ***slightly*** improved performance

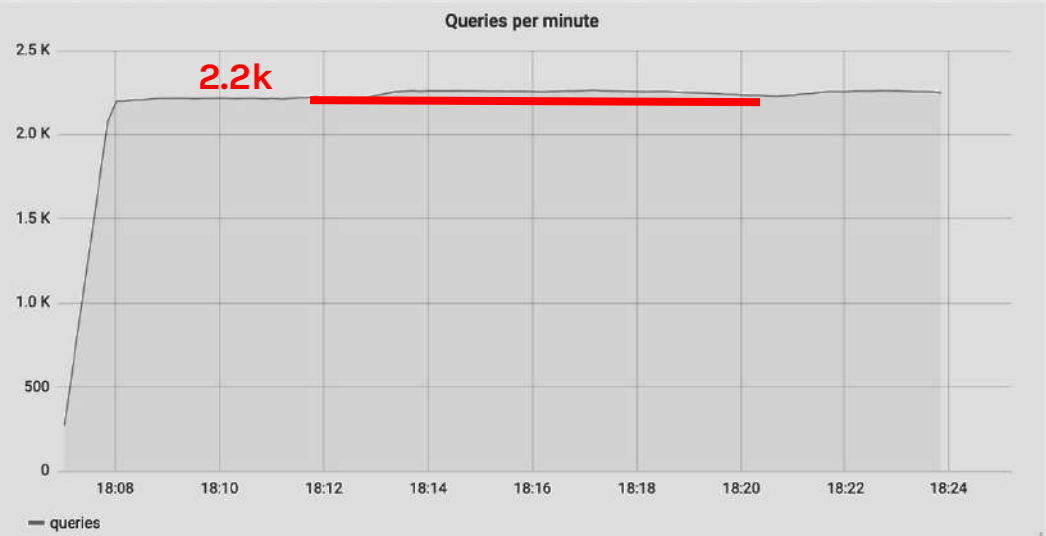
Infrastructure

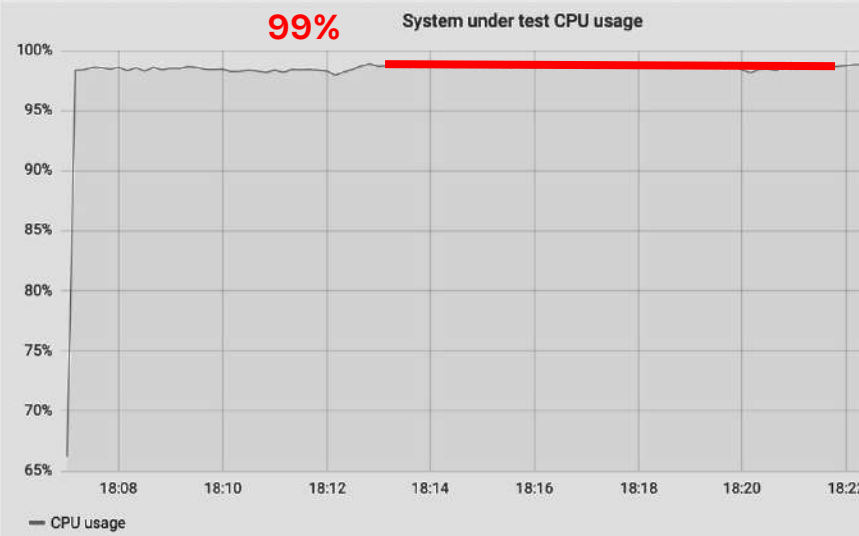


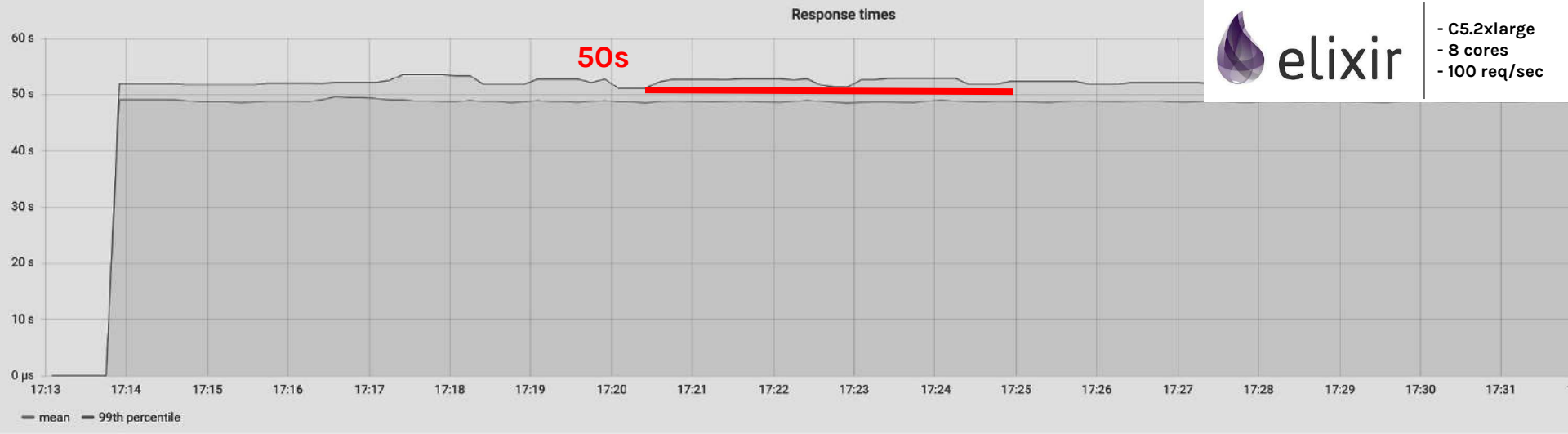


Scala

- C5.2xlarge
- 8 cores
- 100 req/sec

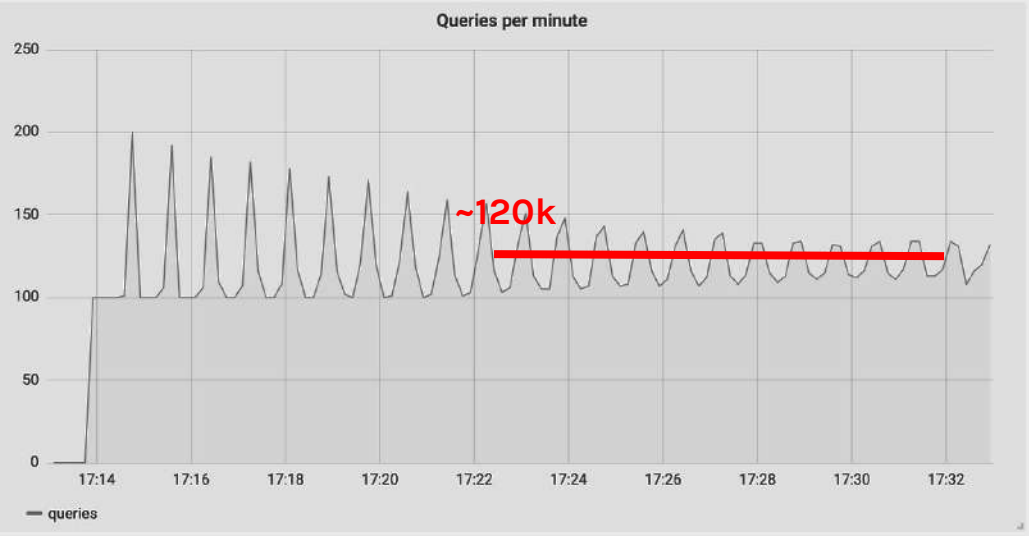


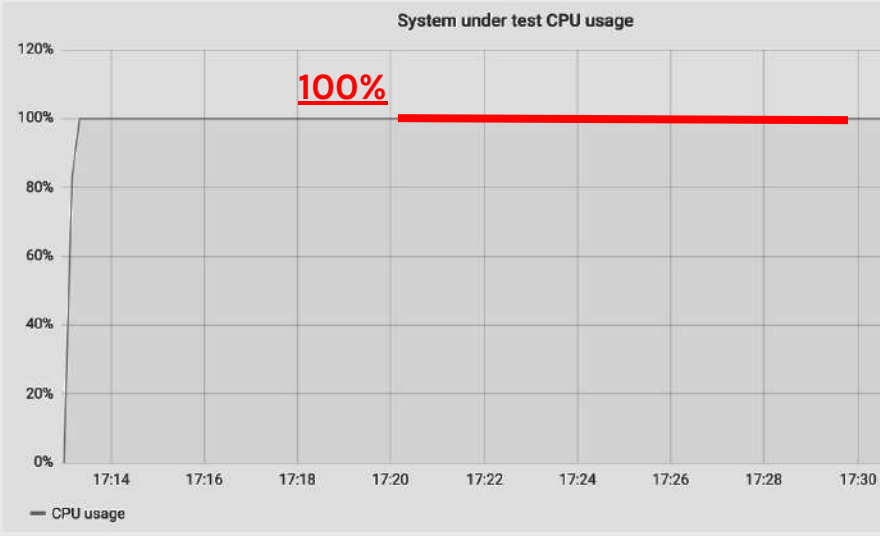




elixir

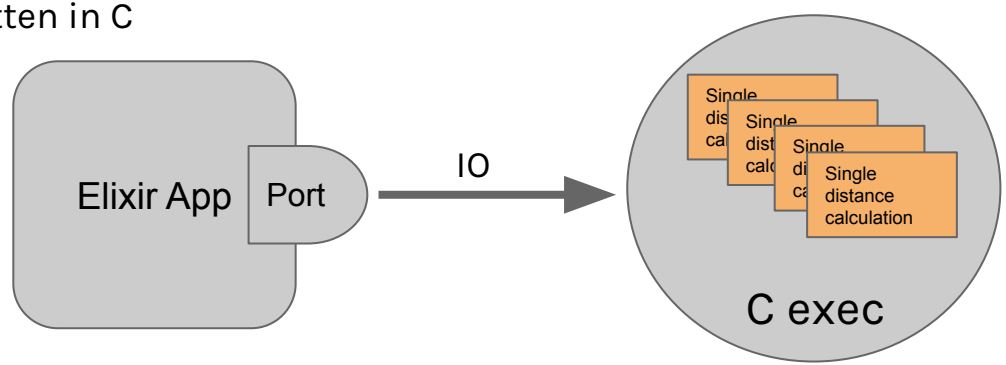
- C5.2xlarge
- 8 cores
- 100 req/sec

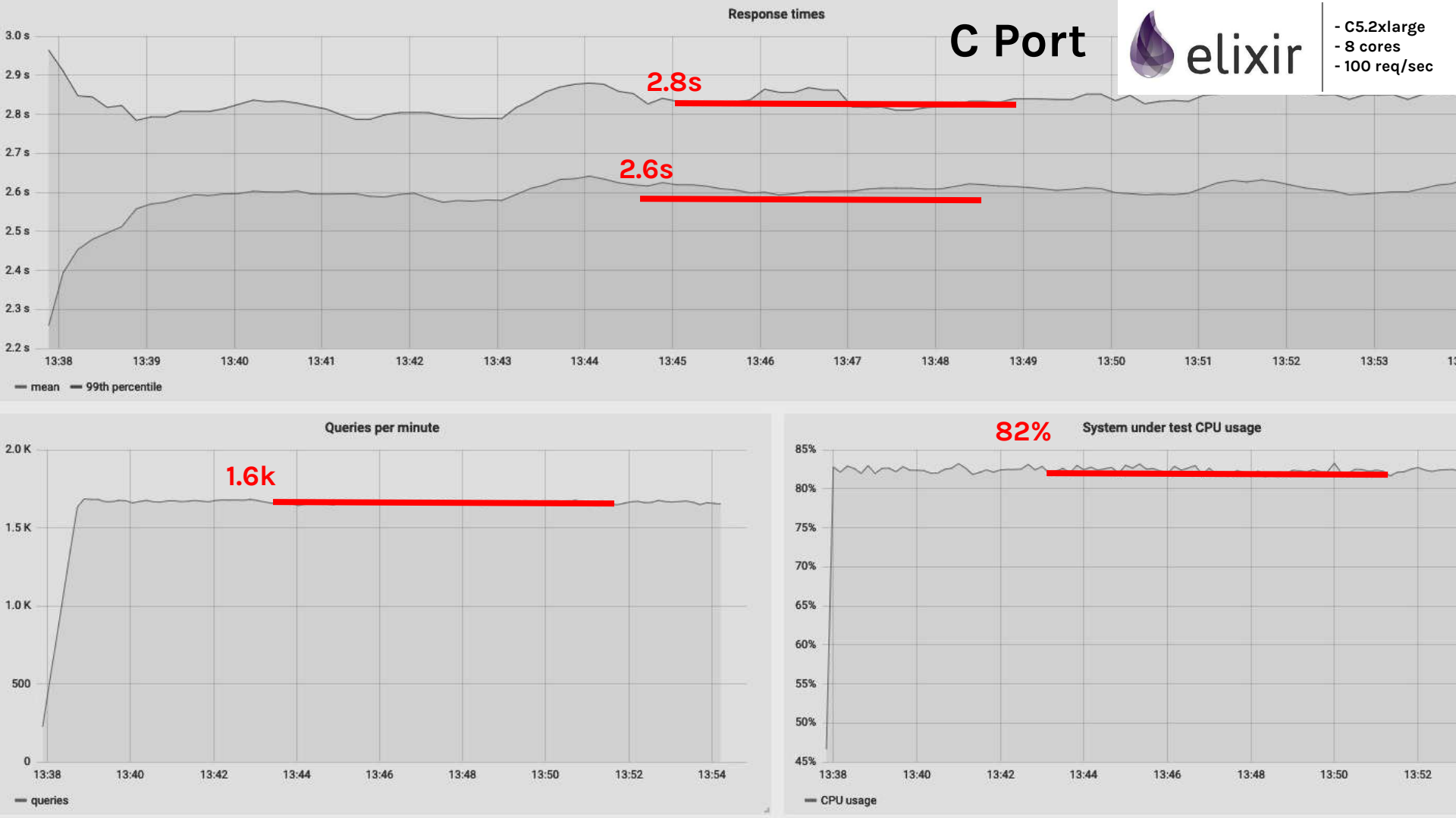




C Port

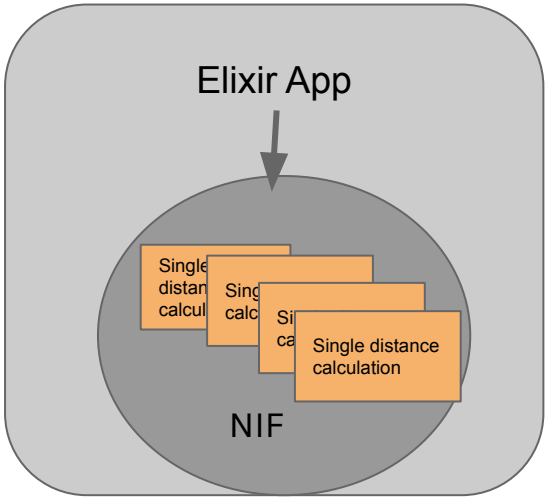
- Spawn external program outside of the BEAM
- Communication through I/O operations
- Safe
- Pooling
- Distance calculation between two samples delegated to external program written in C

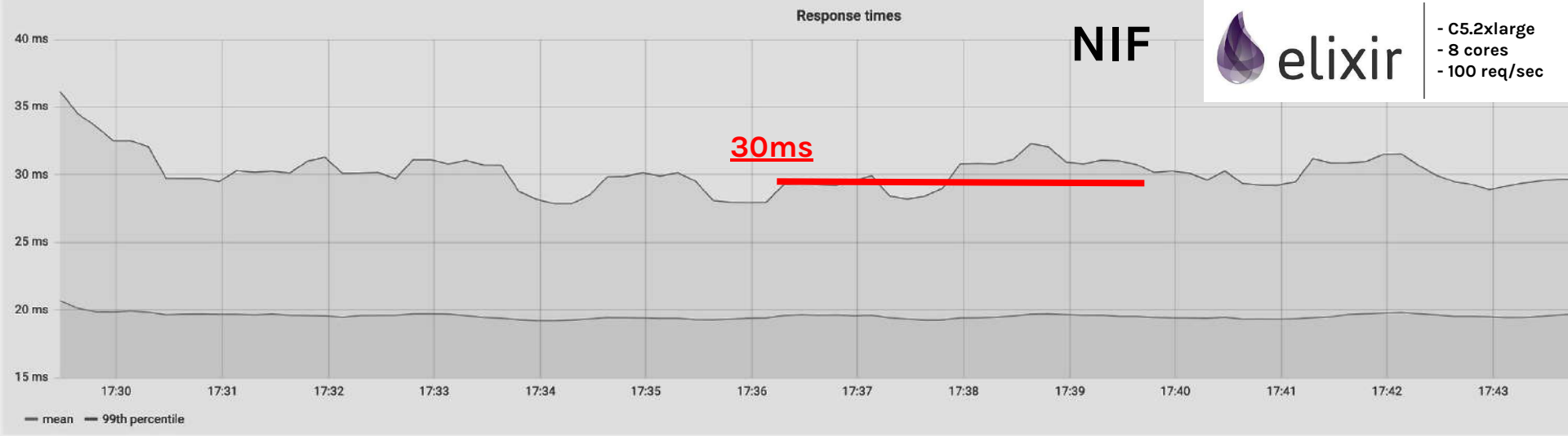


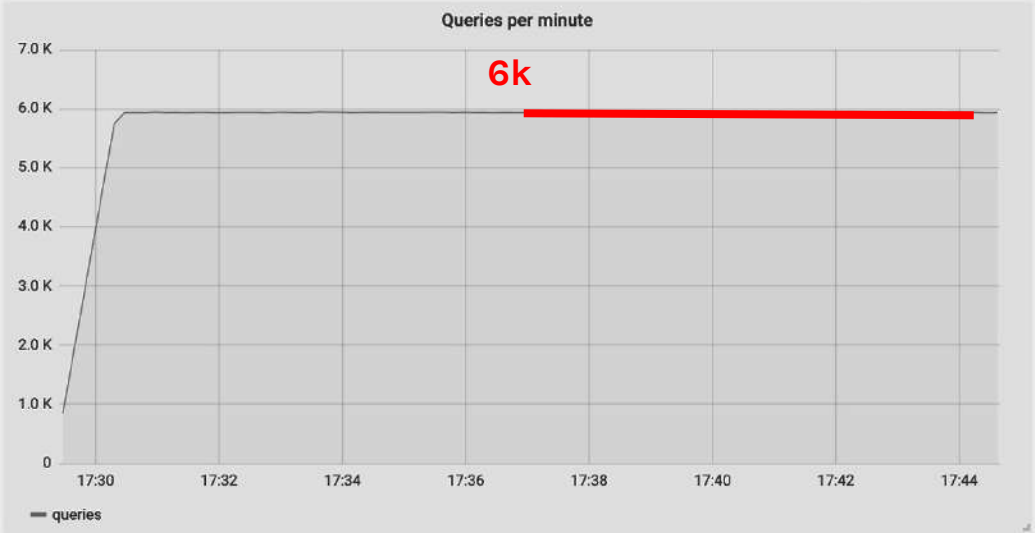

Response times
C Port
elixir
- C5.2xlarge
- 8 cores
- 100 req/sec
2.8s
2.6s
3.0 s
2.9 s
2.8 s
2.7 s
2.6 s
2.5 s
2.4 s
2.3 s
2.2 s
13:38
13:39
13:40
13:41
13:42
13:43
13:44
13:45
13:46
13:47
13:48
13:49
13:50
13:51
13:52
13:53
mean
99th percentile
Queries per minute
1.6k
2.0 K
1.5 K
1.0 K
500
0
13:38
13:40
13:42
13:44
13:46
13:48
13:50
13:52
13:54
queries
82%
System under test CPU usage
85%
80%
75%
70%
65%
60%
55%
50%
45%
13:38
13:40
13:42
13:44
13:46
13:48
13:50
13:52
CPU usage

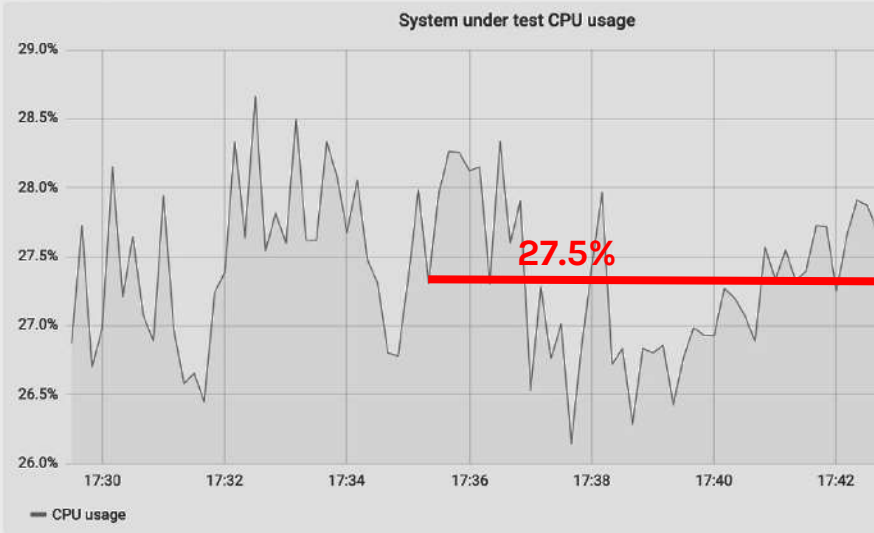
NIF

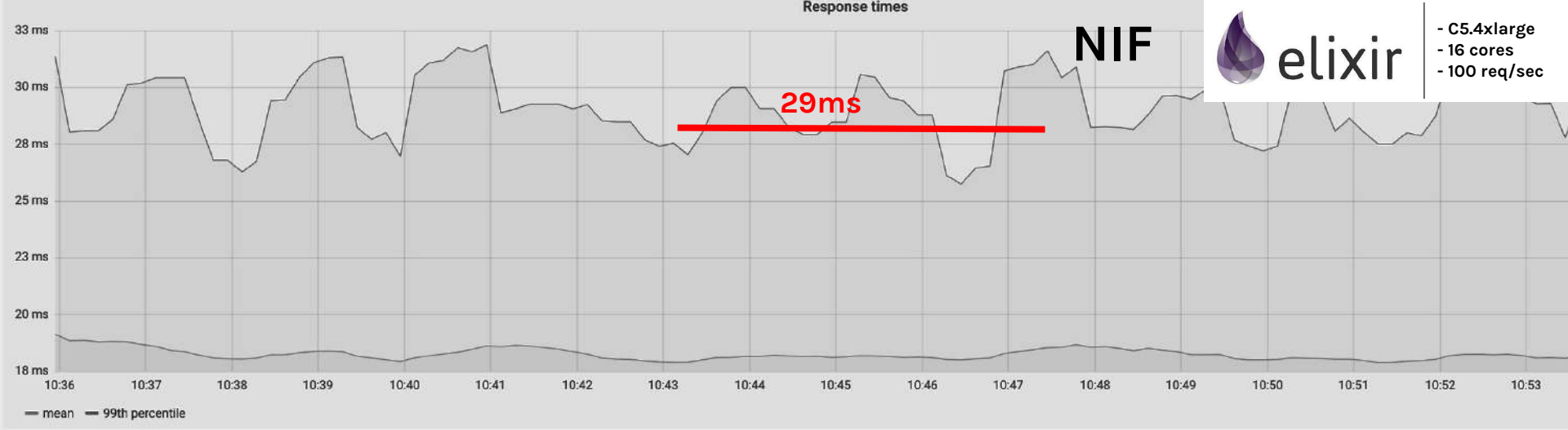
- Call the C code in context of *BEAM* like regular *Erlang* function
- Faster than port
- Unsafe!




Response times
NIF
elixir
- C5.2xlarge
- 8 cores
- 100 req/sec
30ms
40 ms
35 ms
30 ms
25 ms
20 ms
15 ms
17:30
17:31
17:32
17:33
17:34
17:35
17:36
17:37
17:38
17:39
17:40
17:41
17:42
17:43
mean
99th percentile

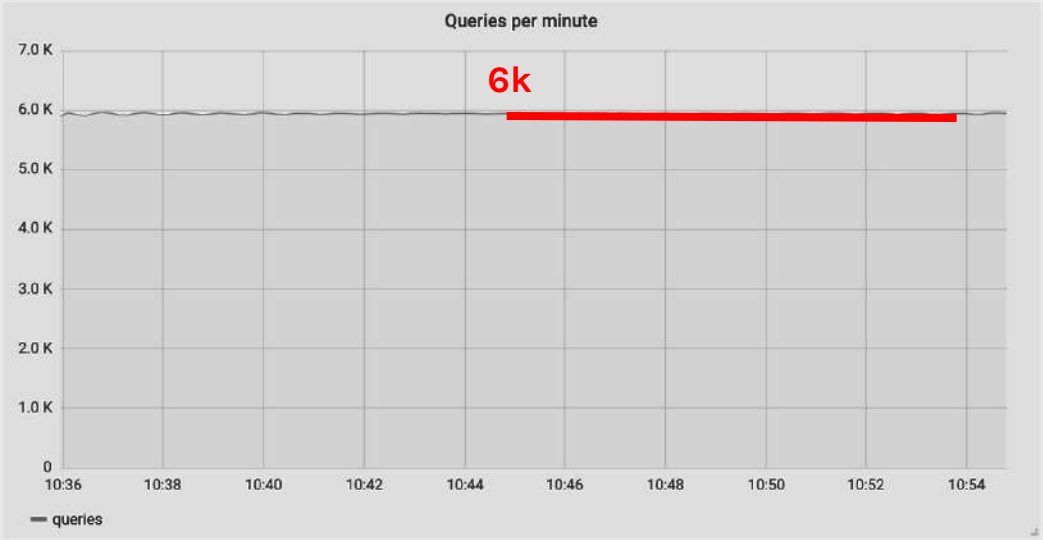

Queries per minute
6k
7.0 K
6.0 K
5.0 K
4.0 K
3.0 K
2.0 K
1.0 K
0
17:30
17:32
17:34
17:36
17:38
17:40
17:42
17:44
queries

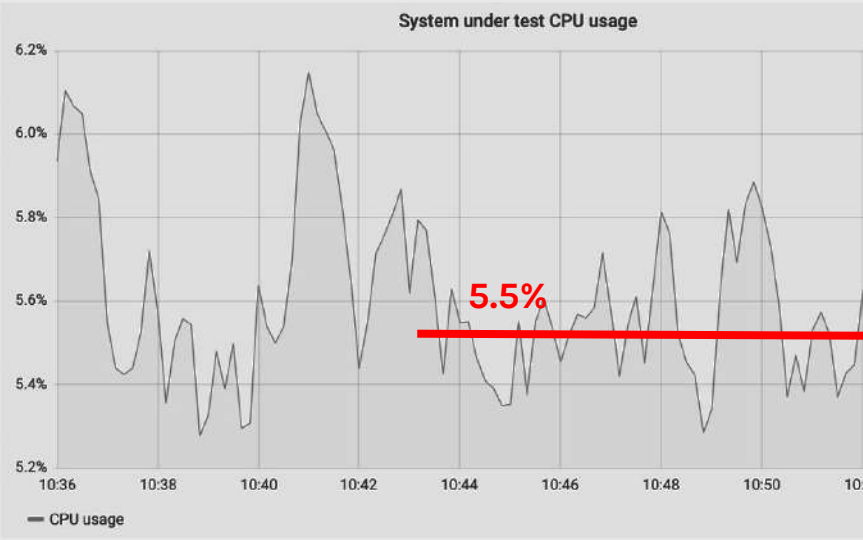

System under test CPU usage
27.5%
29.0%
28.5%
28.0%
27.5%
27.0%
26.5%
26.0%
17:30
17:32
17:34
17:36
17:38
17:40
17:42
CPU usage

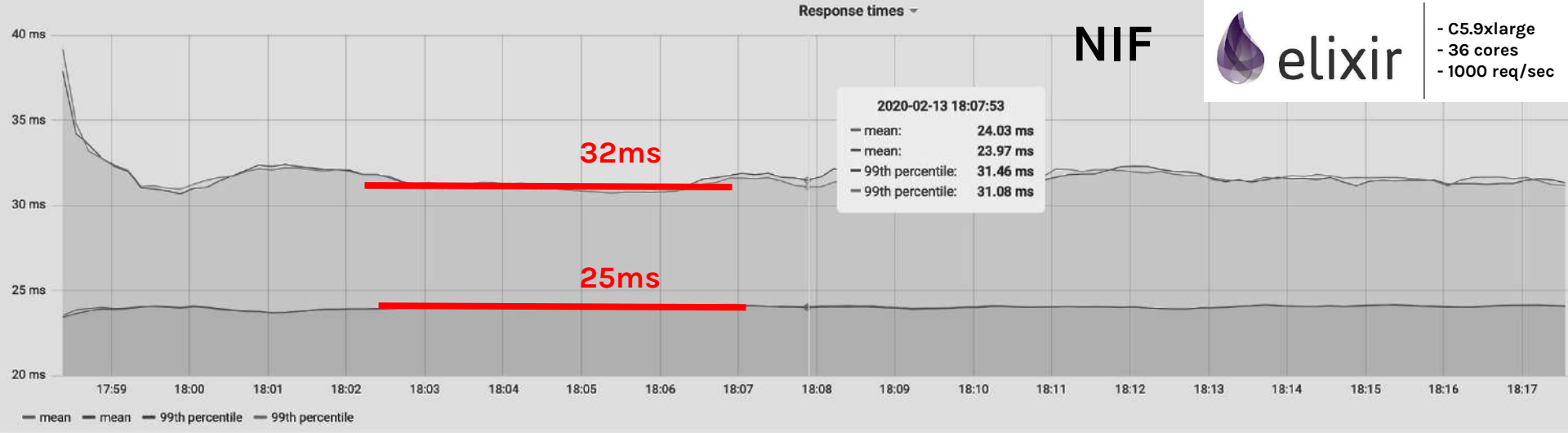


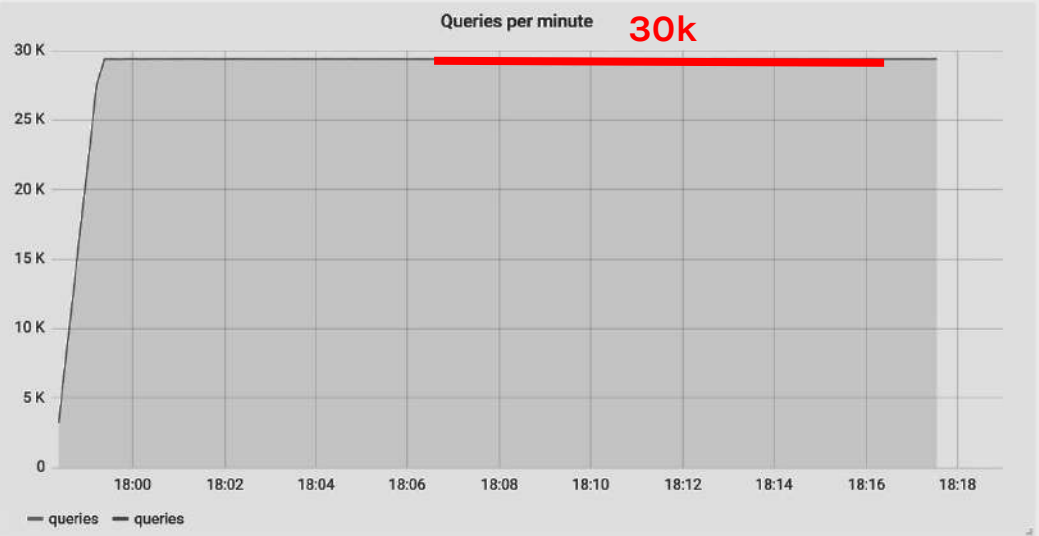
elixir

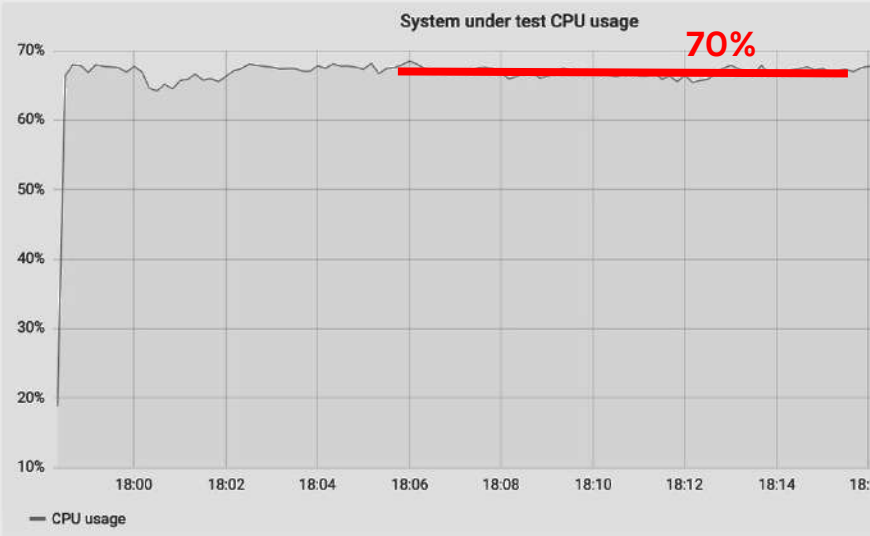
- C5.4xlarge
- 16 cores
- 100 req/sec






Response times
NIF
elixir
- C5.9xlarge
- 36 cores
- 1000 req/sec
40 ms
35 ms
30 ms
25 ms
20 ms
32ms
25ms
2020-02-13 18:07:53
mean: 24.03 ms
mean: 23.97 ms
99th percentile: 31.46 ms
99th percentile: 31.08 ms
17:59
18:00
18:01
18:02
18:03
18:04
18:05
18:06
18:07
18:08
18:09
18:10
18:11
18:12
18:13
18:14
18:15
18:16
18:17
mean
mean
99th percentile
99th percentile

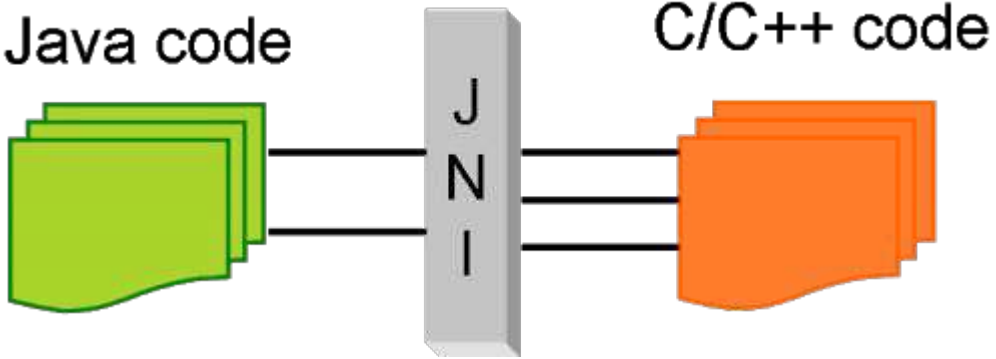

Queries per minute
30k
30 K
25 K
20 K
15 K
10 K
5 K
0
18:00
18:02
18:04
18:06
18:08
18:10
18:12
18:14
18:16
18:18
queries
queries


System under test CPU usage
70%
70%
60%
50%
40%
30%
20%
10%
18:00
18:02
18:04
18:06
18:08
18:10
18:12
18:14
CPU usage

JNI

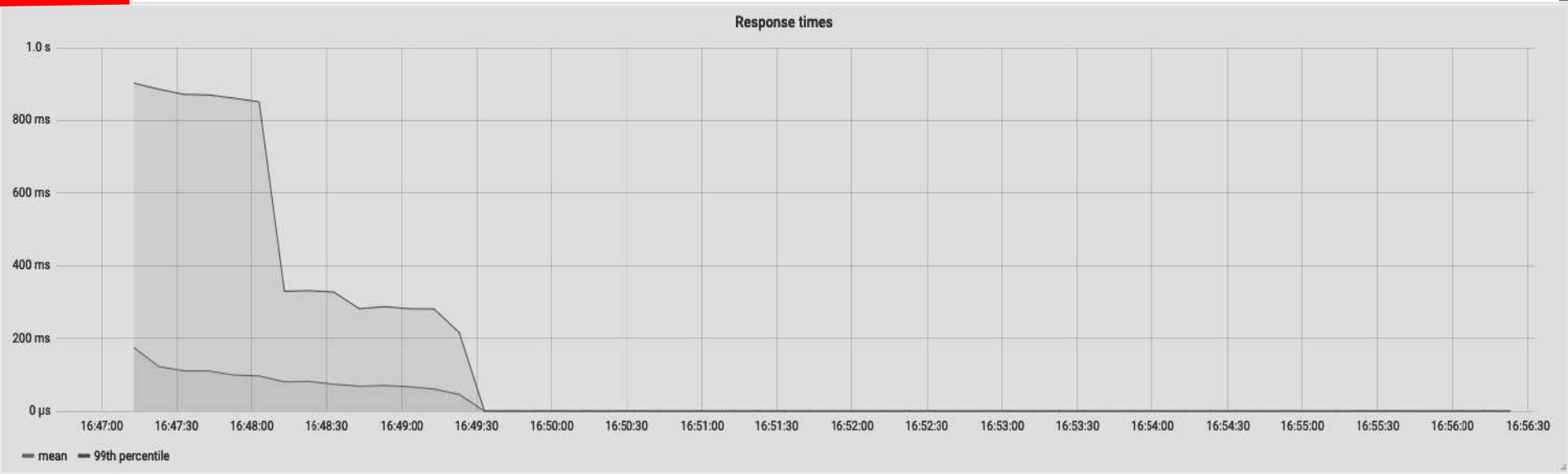
The Java Native Interface (JNI) is a foreign function interface programming framework that enables Java code running in a Java virtual machine (JVM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.

~wikipedia



- C5.2xlarge
- 8 cores
- 100 req/sec

```
[INFO] [02/12/2020 15:59:22.357] [HwrecHttpServer-akka.actor.default-dispatcher-15] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.369] [HwrecHttpServer-akka.actor.default-dispatcher-25] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.370] [HwrecHttpServer-akka.actor.default-dispatcher-25] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.371] [HwrecHttpServer-akka.actor.default-dispatcher-21] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.374] [HwrecHttpServer-akka.actor.default-dispatcher-13] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.374] [HwrecHttpServer-akka.actor.default-dispatcher-19] [Routes(akka://HwrecHttpSer
[INFO] [02/12/2020 15:59:22.374] [HwrecHttpServer-akka.actor.default-dispatcher-3] [Routes(akka://HwrecHttpServ
Killed
```



- C5.2xlarge
- 8 cores
- 100 req/sec

```
root@ip-10-2-0-202:/opt/knn/hwrec_scala# tail -f /var/log/kern.log
...
Feb 12 15:59:22 ip-10-2-0-202 kernel: [ 1857.731712] [15517]     0 15517
5632486  3857852 31735808        0             0 java
Feb 12 15:59:22 ip-10-2-0-202 kernel: [ 1857.731714] Out of memory: Kill
process 15517 (java) score 947 or sacrifice child
Feb 12 15:59:22 ip-10-2-0-202 kernel: [ 1857.735169] Killed process 15517
(java) total-vm:22529944kB, anon-rss:15431408kB, file-rss:0kB,
shmem-rss:0kB
Feb 12 15:59:23 ip-10-2-0-202 kernel: [ 1858.308338] oom_reaper: reaped
process 15517 (java), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
```

Fix OOM

jni-2-13

Browse files

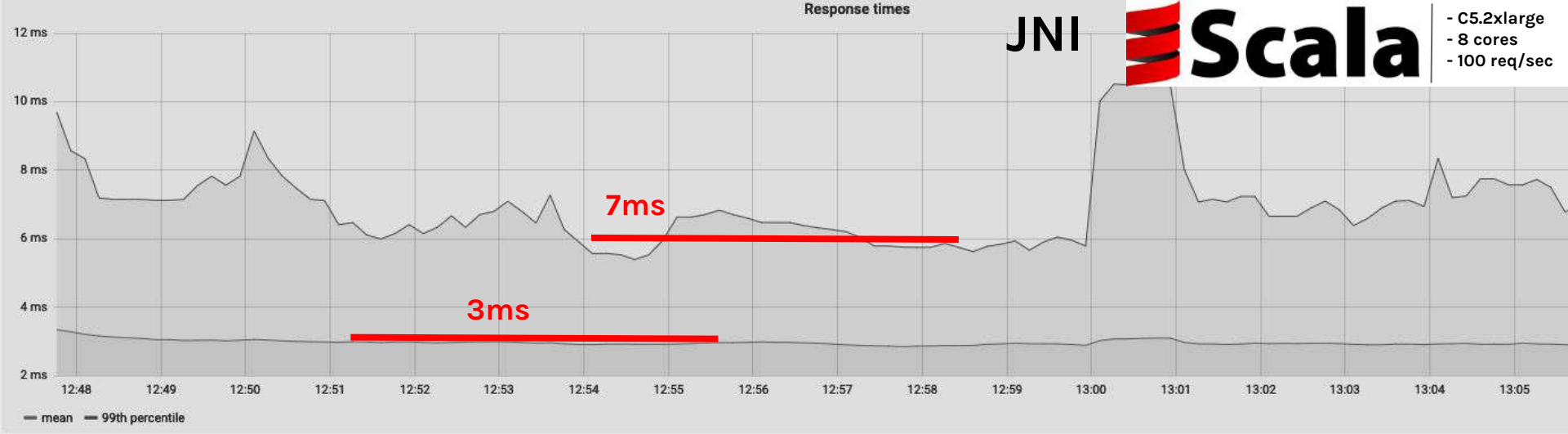
mkacper committed 9 hours ago 1 parent 7a83c4d commit 8da888f2e5a7d67a2e2bcfb663d363f81d310378

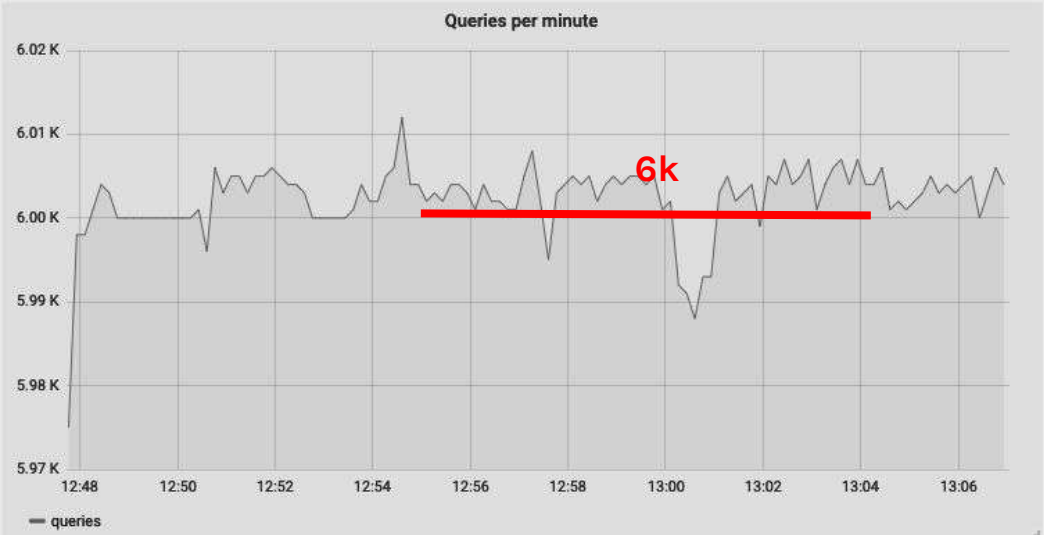
Showing 1 changed file with 2 additions and 0 deletions.

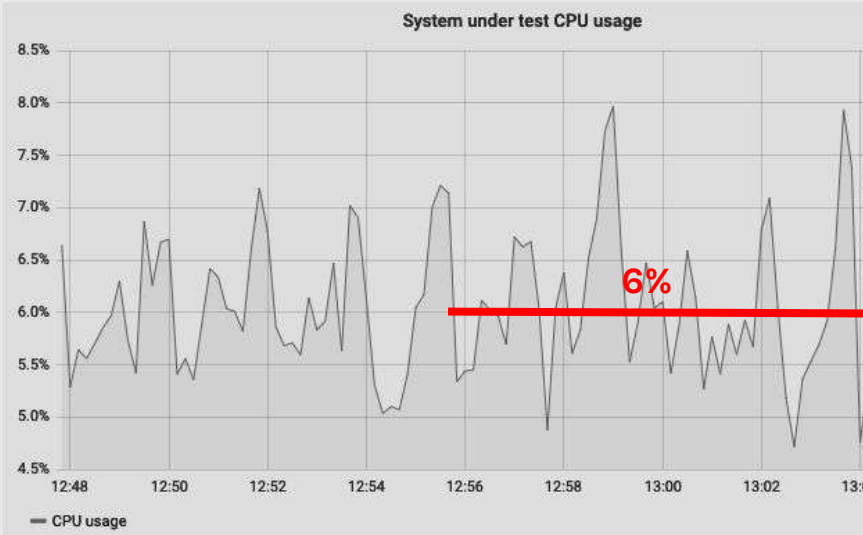
Unified | Split

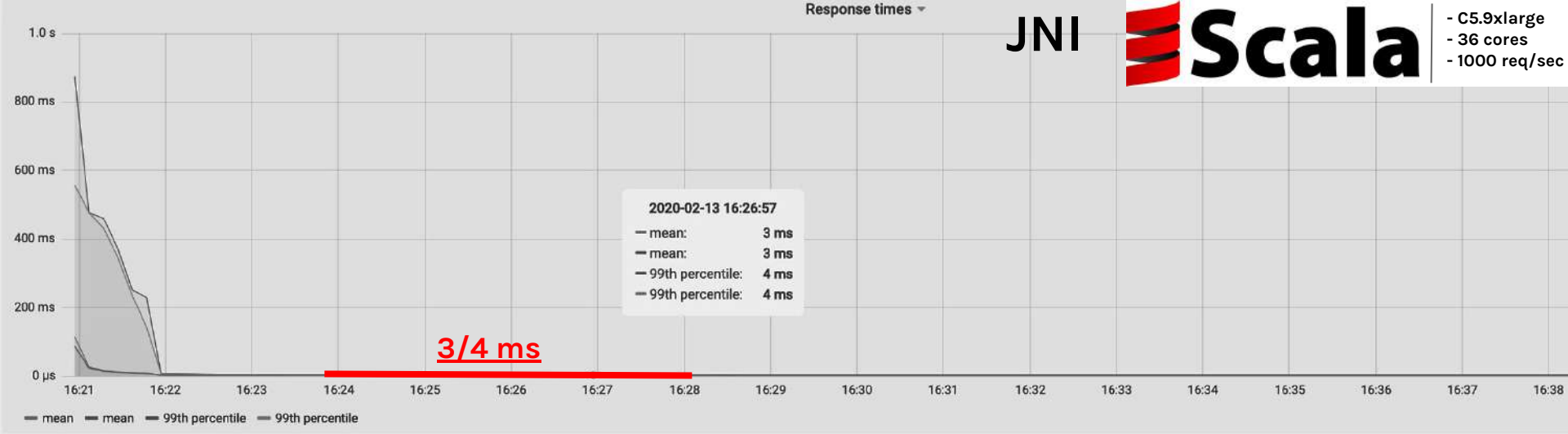
2 src/main/scala/com/hwrec/com_hwrec_JavaNativeCalculator.cpp

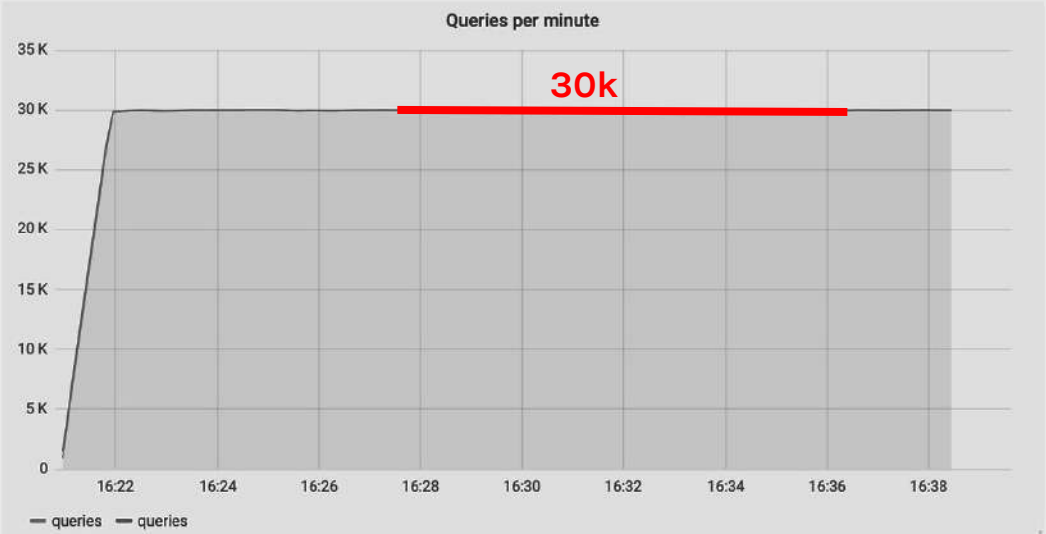
```
@@ -29,6 +29,8 @@ JNIEXPORT jdouble JNICALL Java_com_hwrec_JavaNativeCalculator_distance
29 29     for(int i = 0;i< data_len;i++){
30 30     res += abs(data_arr[i] - input_arr[i]);
31 31     }
   32 +   env->ReleaseByteArrayElements(input, data_arr, 0);
   33 +   env->ReleaseByteArrayElements(data, input_arr, 0);
32 34
33 35     return res;
34 36   }
```

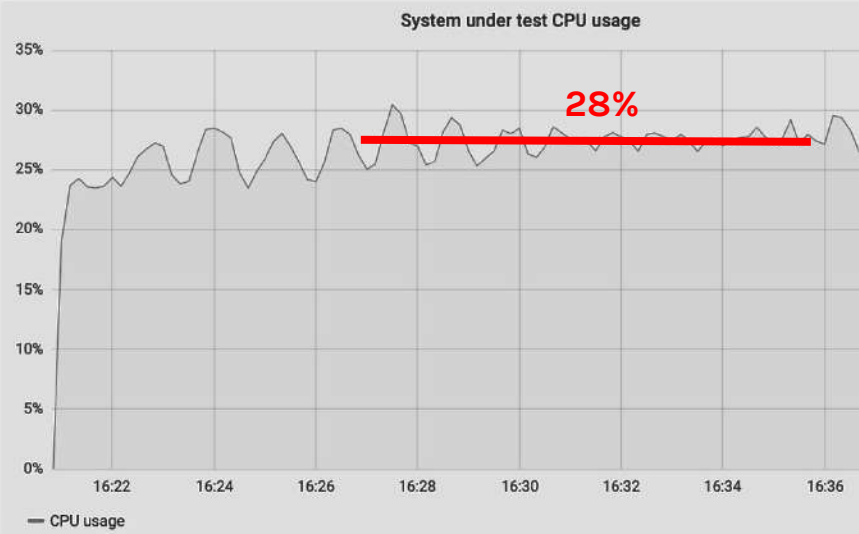

Response times
JNI
Scala
- C5.2xlarge
- 8 cores
- 100 req/sec
12 ms
10 ms
8 ms
6 ms
4 ms
2 ms
7ms
3ms
12:48
12:49
12:50
12:51
12:52
12:53
12:54
12:55
12:56
12:57
12:58
12:59
13:00
13:01
13:02
13:03
13:04
13:05
mean
99th percentile


Queries per minute
6.02 K
6.01 K
6.00 K
5.99 K
5.98 K
5.97 K
6k
12:48
12:50
12:52
12:54
12:56
12:58
13:00
13:02
13:04
13:06
queries

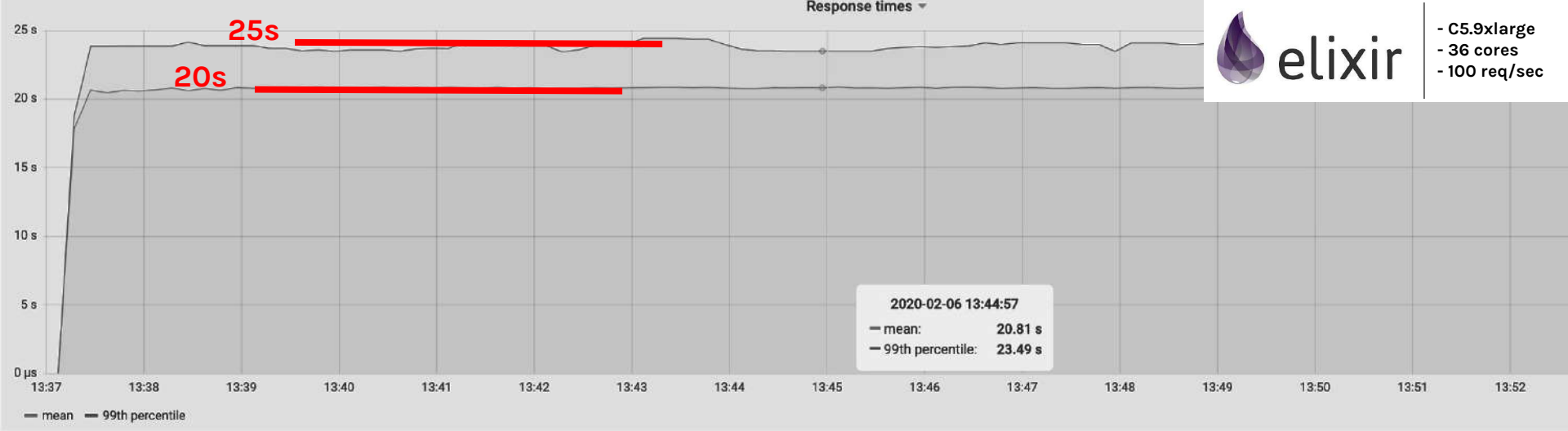

System under test CPU usage
8.5%
8.0%
7.5%
7.0%
6.5%
6.0%
5.5%
5.0%
4.5%
6%
12:48
12:50
12:52
12:54
12:56
12:58
13:00
13:02
CPU usage


Response times
JNI
Scala
- C5.9xlarge
- 36 cores
- 1000 req/sec
1.0 s
800 ms
600 ms
400 ms
200 ms
0 µs
2020-02-13 16:26:57
mean: 3 ms
mean: 3 ms
99th percentile: 4 ms
99th percentile: 4 ms
3/4 ms
16:21
16:22
16:23
16:24
16:25
16:26
16:27
16:28
16:29
16:30
16:31
16:32
16:33
16:34
16:35
16:36
16:37
16:38
mean
mean
99th percentile
99th percentile


Queries per minute
35 K
30 K
25 K
20 K
15 K
10 K
5 K
0
30k
16:22
16:24
16:26
16:28
16:30
16:32
16:34
16:36
16:38
queries
queries

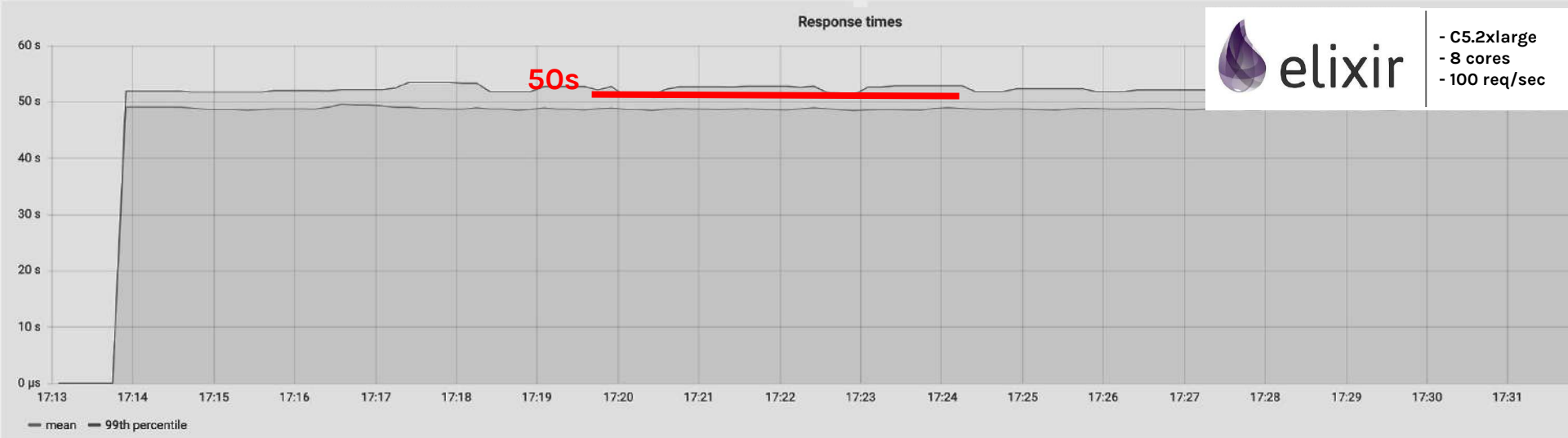

System under test CPU usage
35%
30%
25%
20%
15%
10%
5%
0%
28%
16:22
16:24
16:26
16:28
16:30
16:32
16:34
16:36
CPU usage

Couldn’t we just avoid using native code?



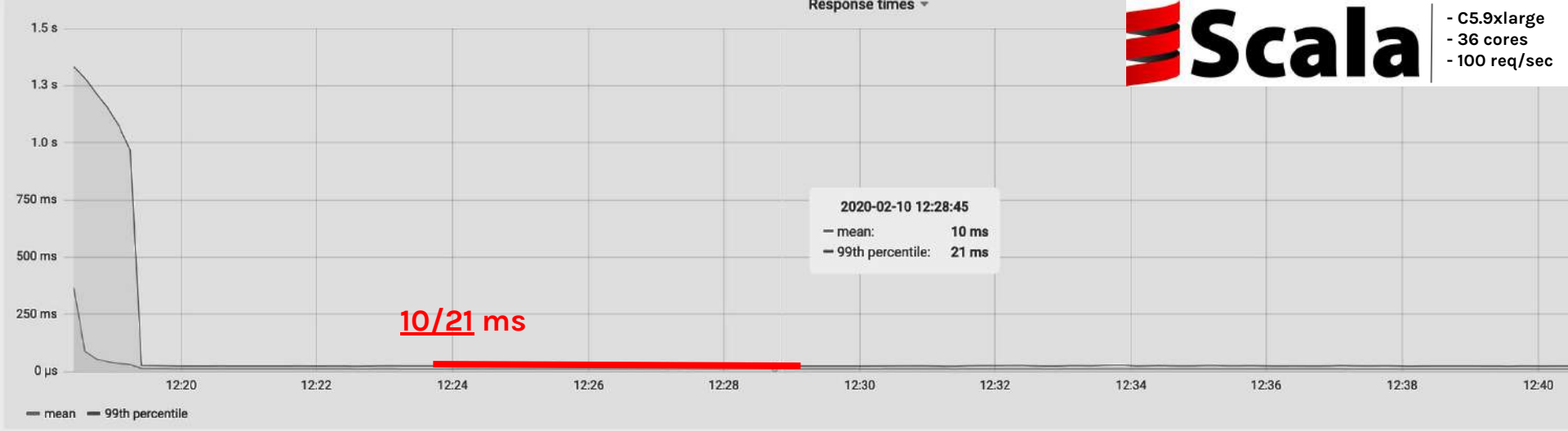
elixir

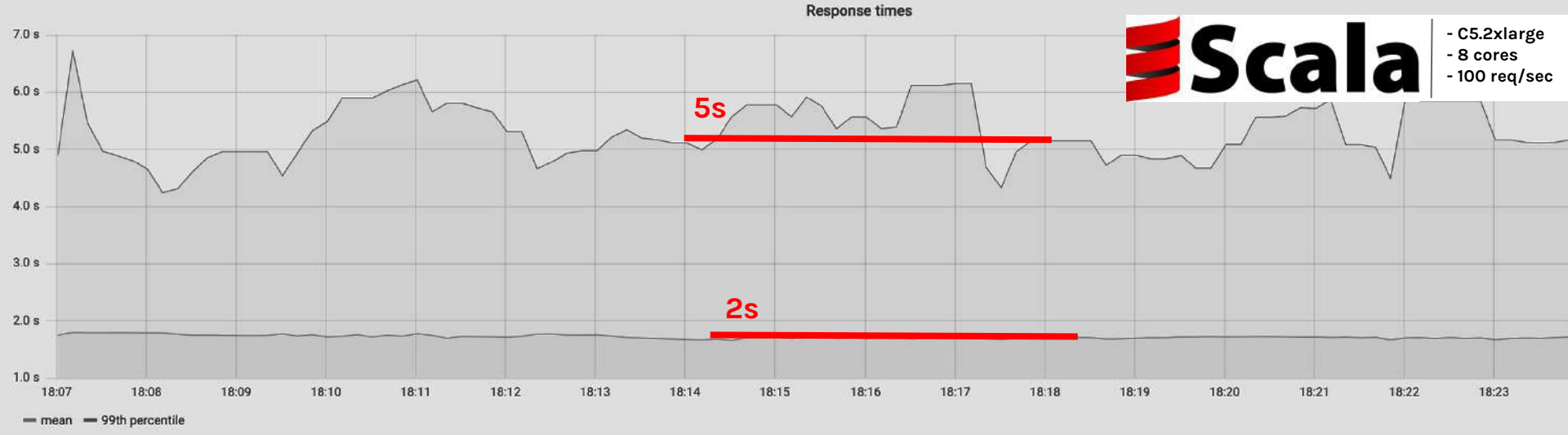
- C5.9xlarge
- 36 cores
- 100 req/sec



elixir

- C5.2xlarge
- 8 cores
- 100 req/sec


Response times
1.5 s
1.3 s
1.0 s
750 ms
500 ms
250 ms
0 µs
12:20
12:22
12:24
12:26
12:28
12:30
12:32
12:34
12:36
12:38
12:40
2020-02-10 12:28:45
mean: 10 ms
99th percentile: 21 ms
10/21 ms
mean
99th percentile
Scala
- C5.9xlarge
- 36 cores
- 100 req/sec


Response times
7.0 s
6.0 s
5.0 s
4.0 s
3.0 s
2.0 s
1.0 s
18:07
18:08
18:09
18:10
18:11
18:12
18:13
18:14
18:15
18:16
18:17
18:18
18:19
18:20
18:21
18:22
18:23
5s
2s
mean
99th percentile
Scala
- C5.2xlarge
- 8 cores
- 100 req/sec

Summary

- Both Scala and Elixir native code will improve you app performance.
- Both Scala and Elixir native code can crash your machine
- **For this particular problem our** Scala implementation have lower latencies than Elixir one
- At some point adding more cores does not improve the overall performance
- Writing native implementation is **not that** hard