

# Scala vs Elixir

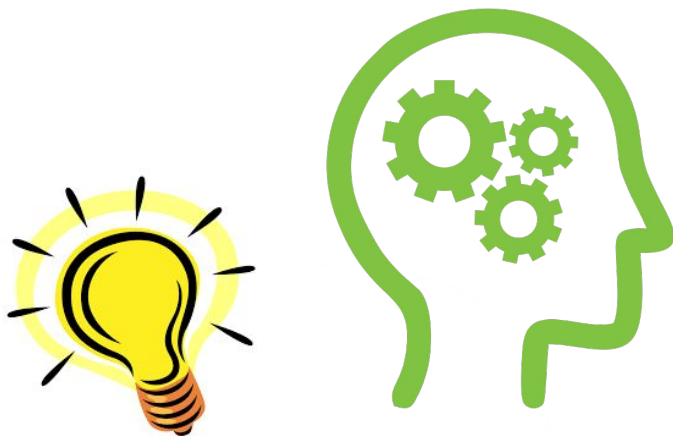


Ludwik Bukowski  
Kacper Mentel

# Erlang

SOLUTIONS

Kraków  
2019



 **Scala**



 **elixir**

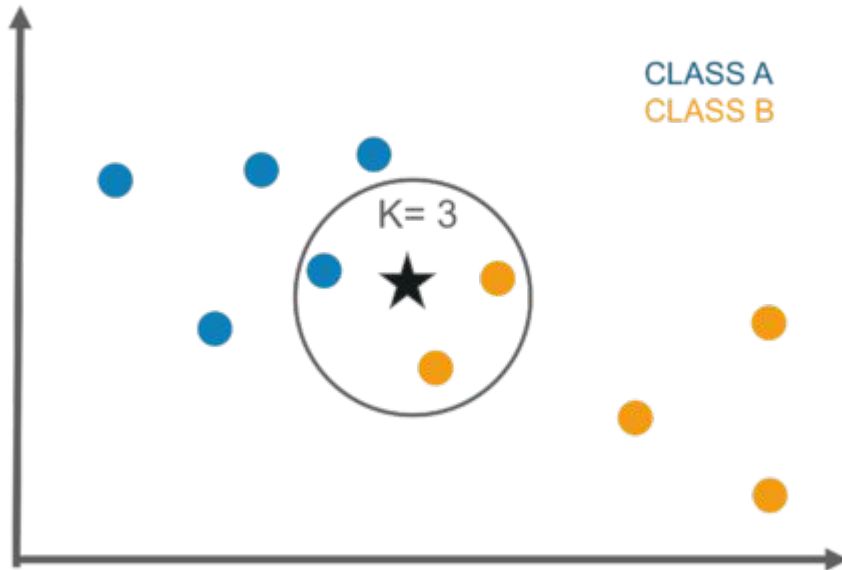
## The problem

4

4	1	3	1	2	9	1	4
1	1	9	0	4	5	9	6
8	2	7	1	6	3	5	3



# The Algorithm















# The Algorithm - implementation

Input vector

```
000000000000000111100000000000
00000000000000011111100000000000
00000000000000011111100000000000
00000000000000011111100000000000
00000000000000011111100000000000
000000000000000111100000001000000
0000000000000001111100000011100000
00000000000000011111000001111100000
00000000000000011111000001111100000
00000000000000011111000001111100000
00000000011110000000111100000000
00000000011110000000111100000000
00000000011110000000111100000000
00000001111100000001111100000000
0000001111110000000011111100000000
0000011111110000000011111100000000
0000011111110000001111100000000000
0000001111100000011111000000000000
0000001111100000111110000000000000
0000001111100000111100000000000000
0000001111100000111100000000000000
0000001111110011110000000000000000
00000000000000000111110000000000000
000000000000000001111100000000000000
000000000000000011111000000000000000
000000000000000011111000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
000000000000000111110000000000000000
```

reference  
sample

```
00000000000000000110000000000000
0000000000000000011000000000000000
0000000000000000011100000000000000
0000000000000000011100000000000000
0000000000000000011100000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
0000000000000000011110000000000000
```

Distance: 257







## Concurrency Scalability

OOP + FE

statically typed

Runs on JVM

FE

dynamically typed

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. “

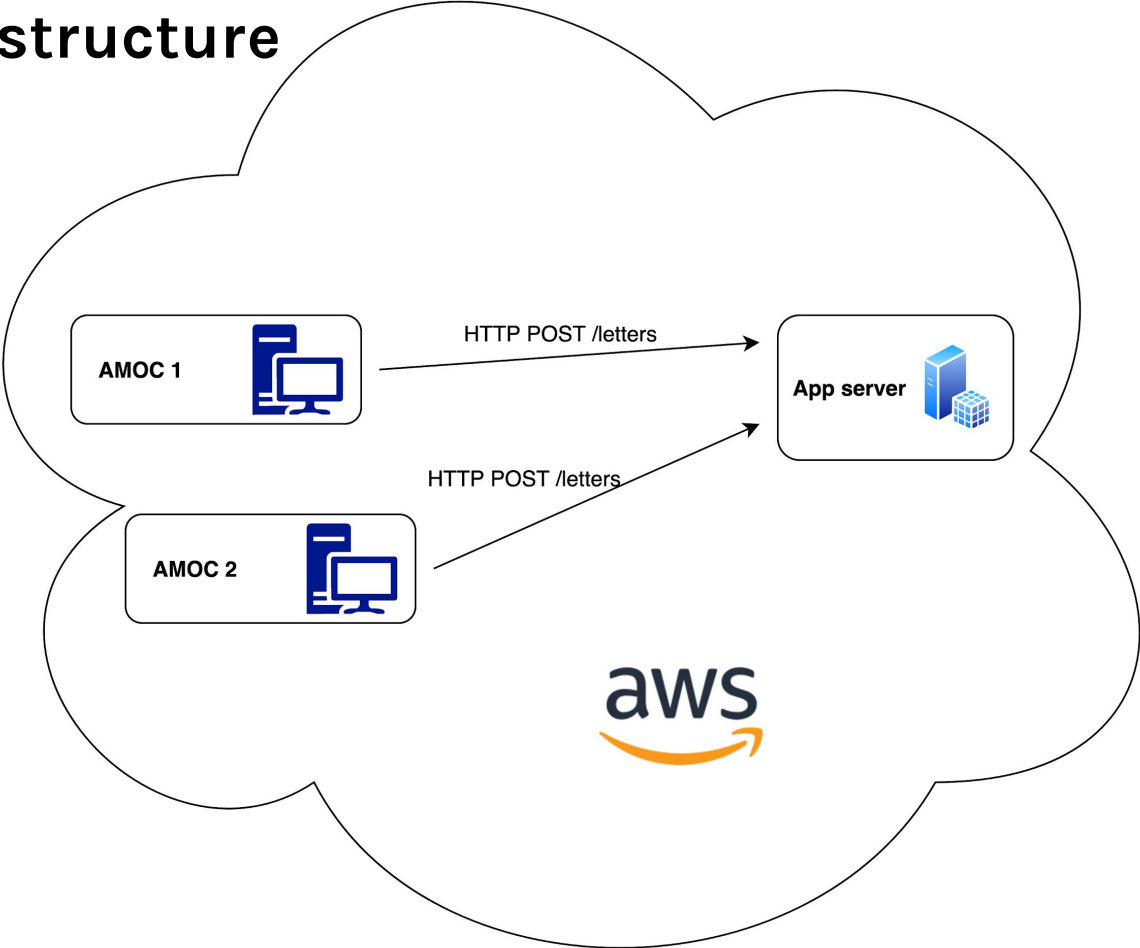
~ Scala documentation: <https://docs.scala-lang.org/overviews/core/futures.html>



- 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



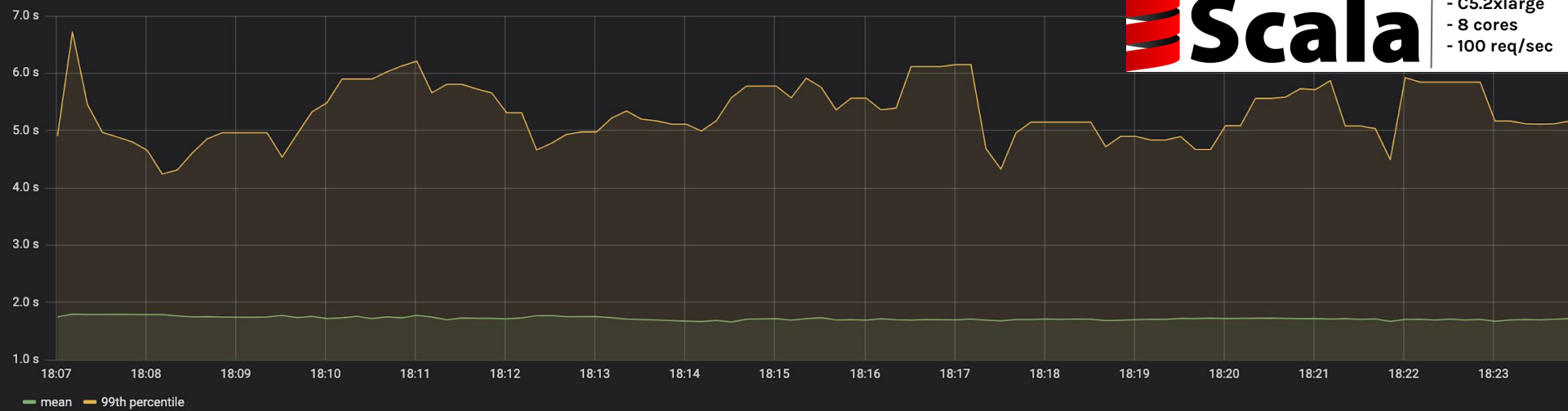
# Infrastructure





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

Response times



Queries per minute



System under test CPU usage

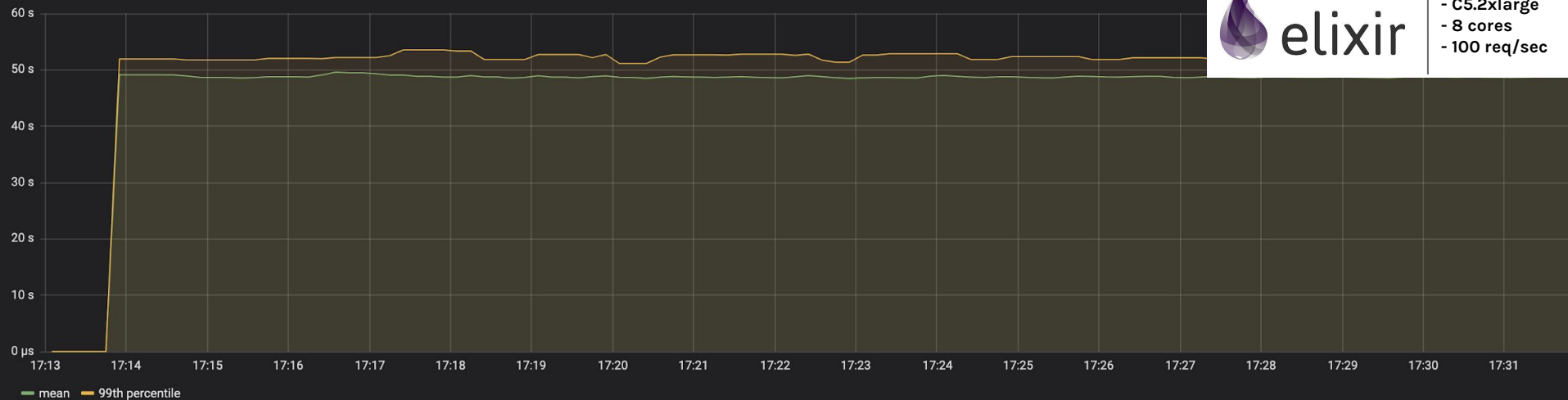


### Response times



elixir

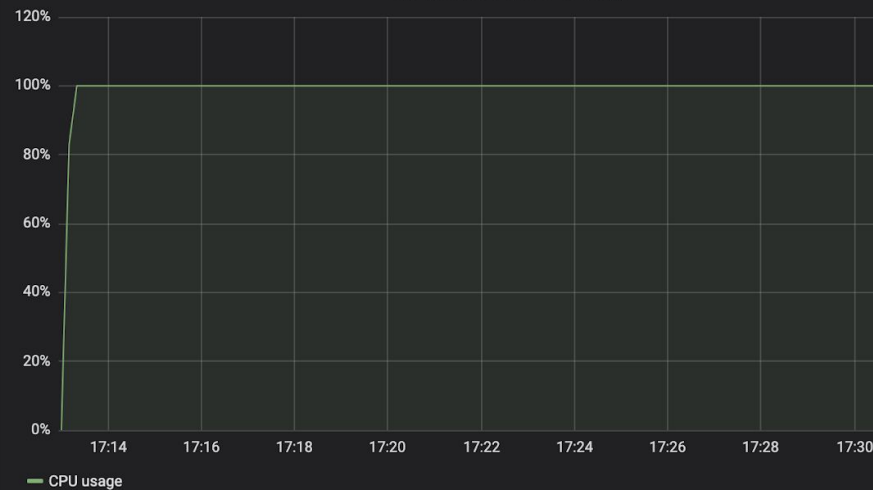
- C5.2xlarge
- 8 cores
- 100 req/sec



### Queries per minute



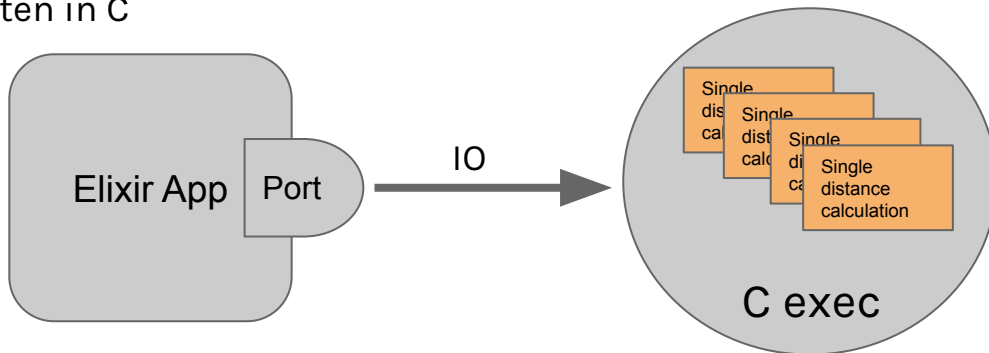
### System under test CPU usage



# 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



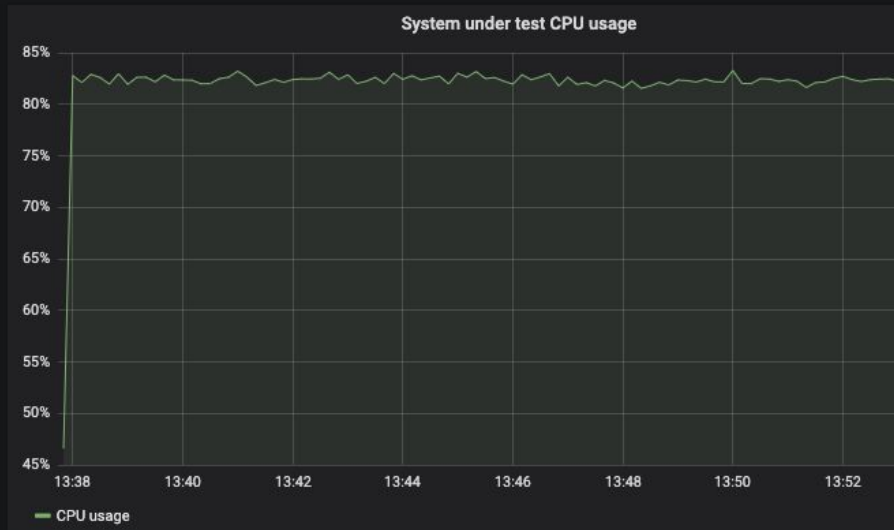
### Response times

# C Port



## elixir

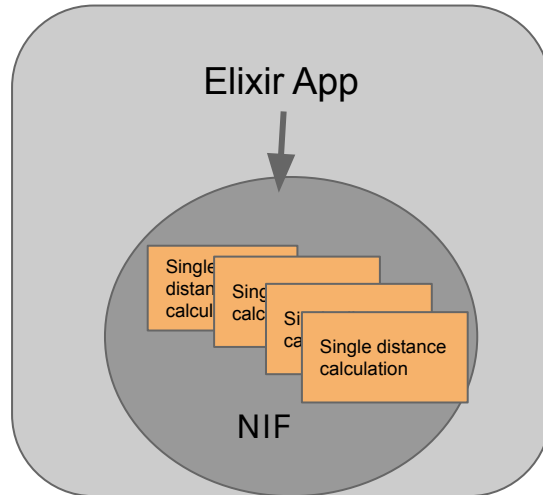
- C5.2xlarge
- 8 cores
- 100 req/sec



# NIF



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



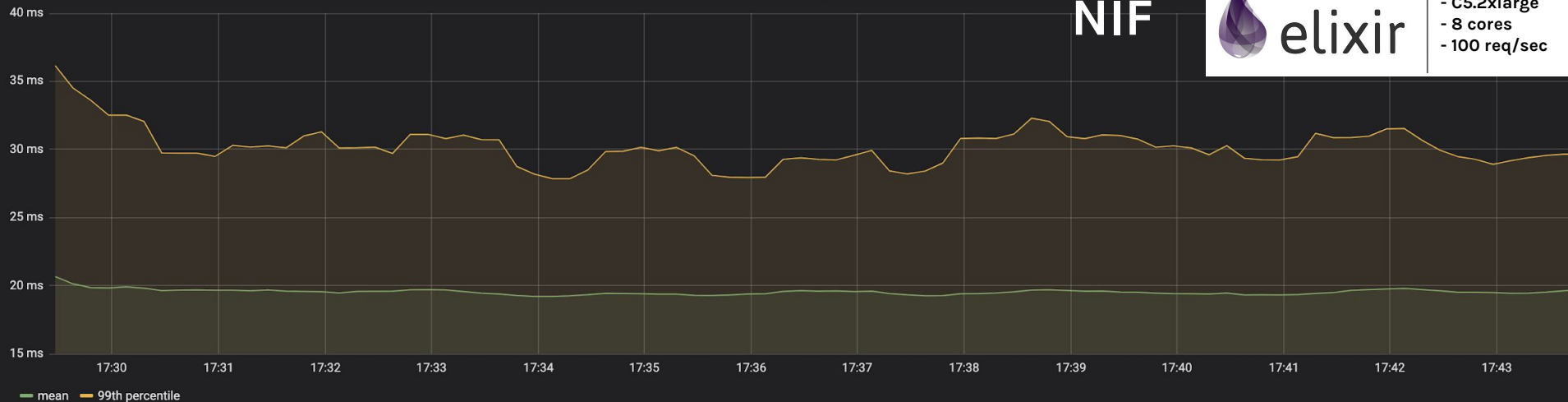
### Response times

# NIF



## elixir

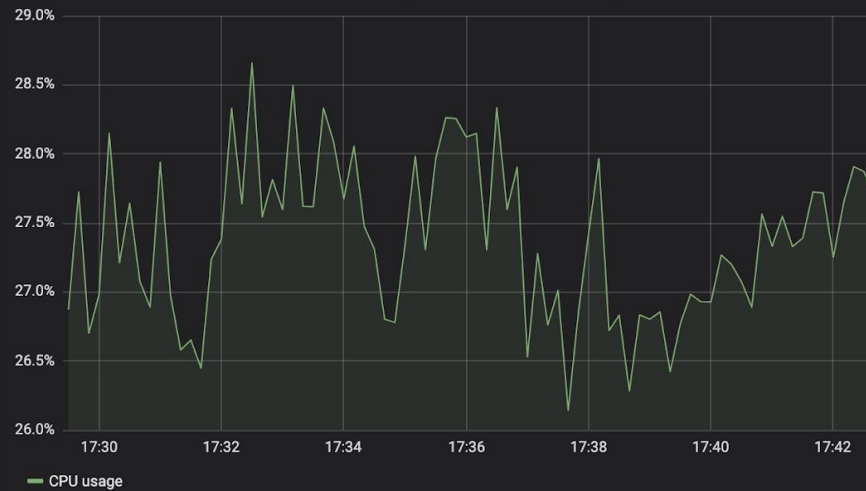
- C5.2xlarge
- 8 cores
- 100 req/sec



### Queries per minute



### System under test CPU usage



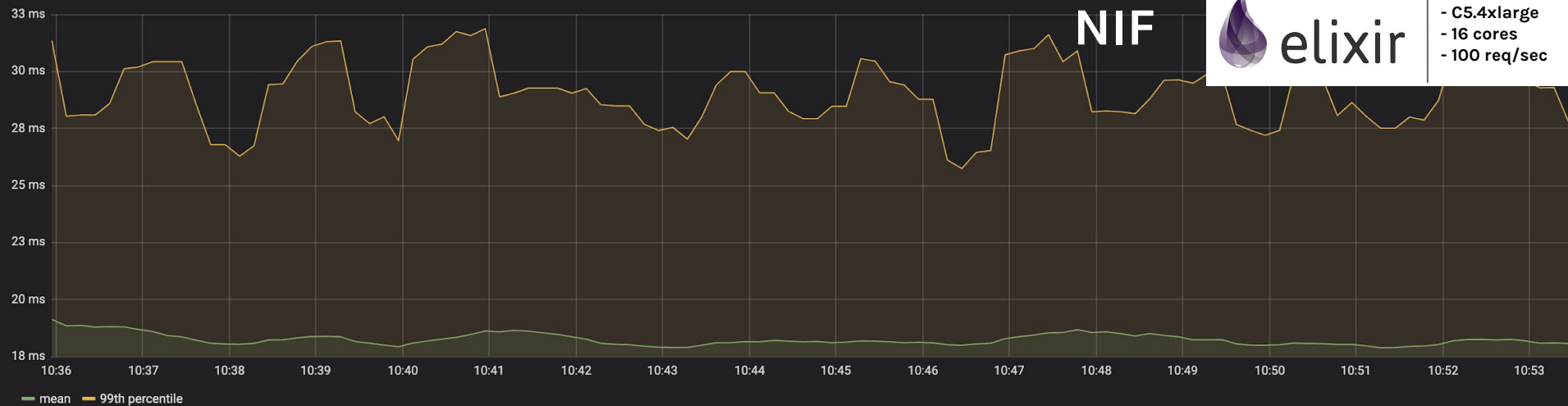
### Response times

# NIF



## elixir

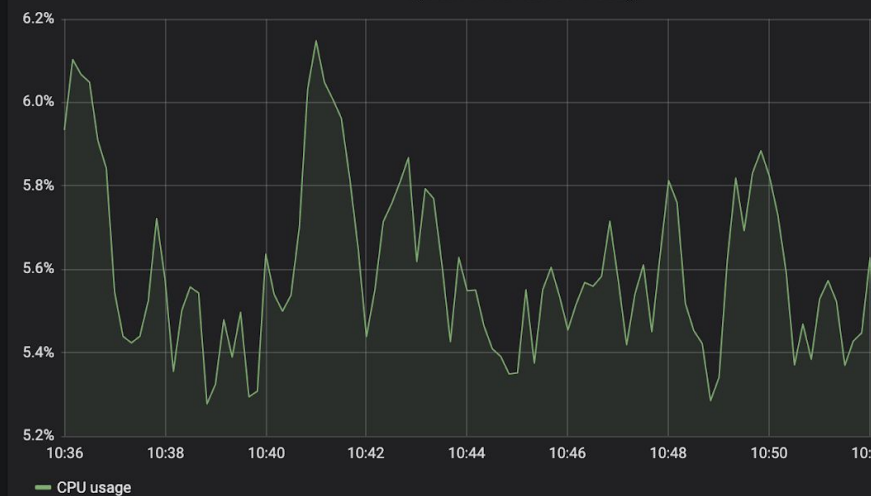
- C5.4xlarge
- 16 cores
- 100 req/sec



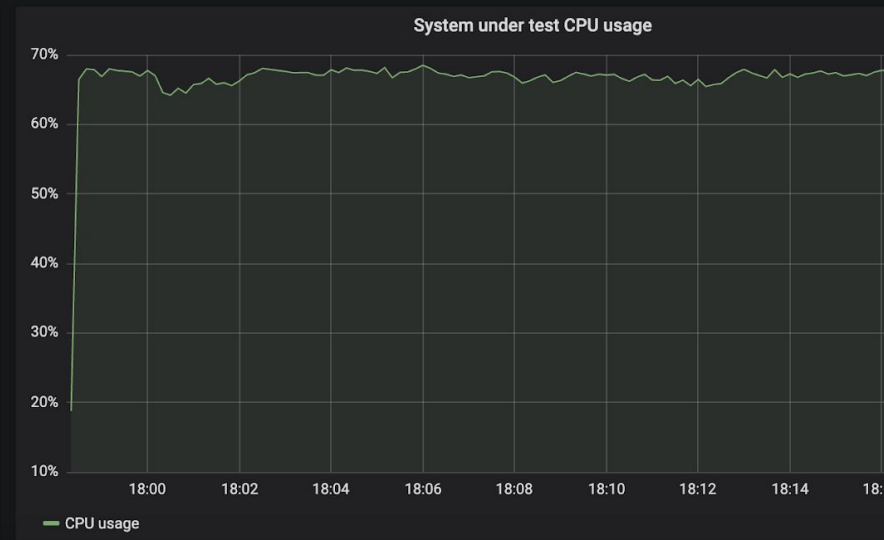
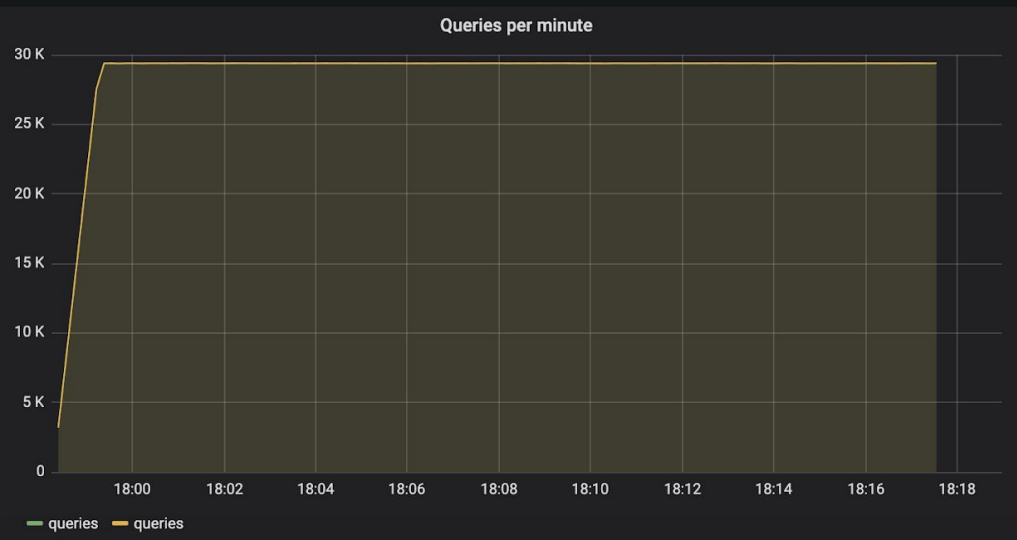
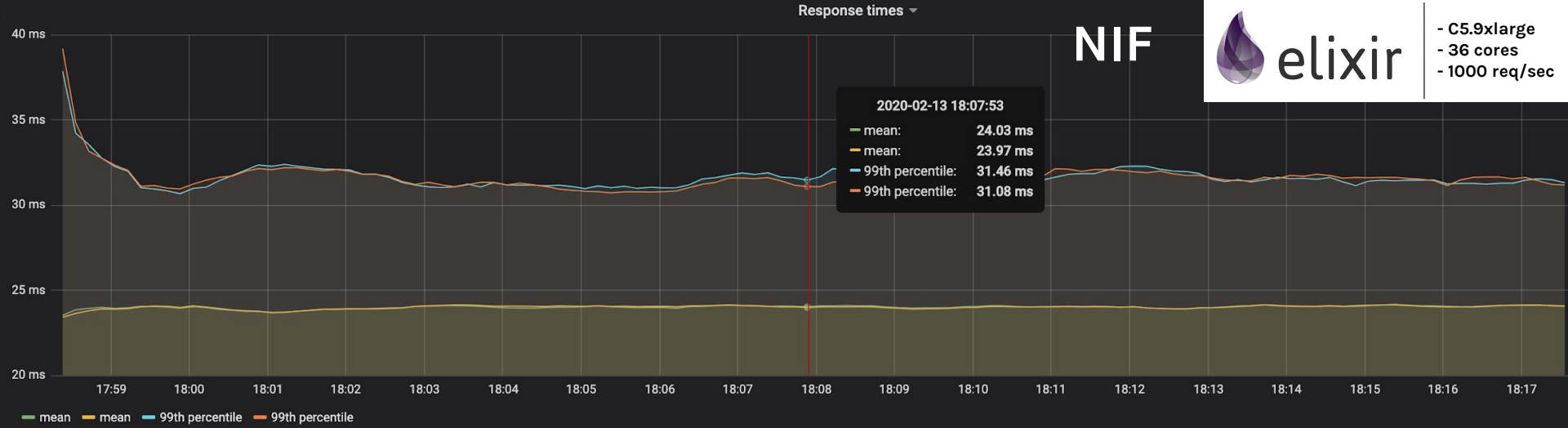
### Queries per minute



### System under test CPU usage





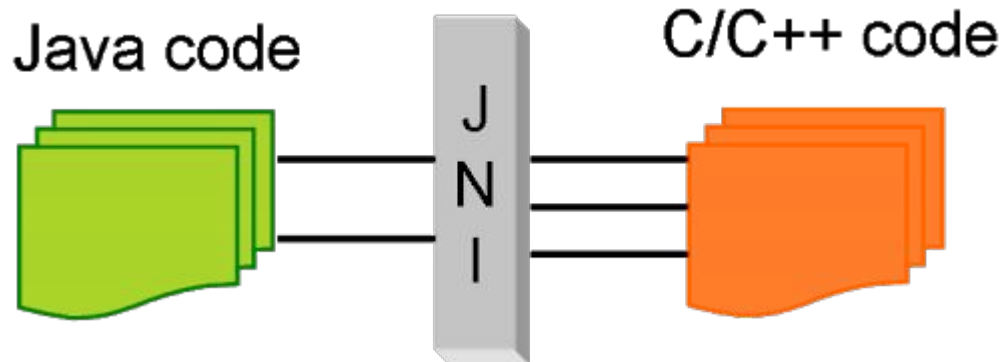


# JNI



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



```
[INFO] [02/12/2020 15:59:22.355] [HwrecHttpServer-akka.actor.default-dispatcher-15] [Routes(akka://HwrecHttpSer  
[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



```
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

[Browse files](#)

jni-2-13

mkacper committed 9 hours ago

1 parent 7a83c4d

commit 8da88f2e5a7d67a2e2bcfb663d363f81d310378

Showing 1 changed file with 2 additions and 0 deletions.

Unified

Split

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      32      + env->ReleaseByteArrayElements(input, data_arr, 0);
33      33      + env->ReleaseByteArrayElements(data, input_arr, 0);
34      34
35      35      return res;
36      36      }
```

src



Queries per minute



System under test CPU usage





Queries per minute

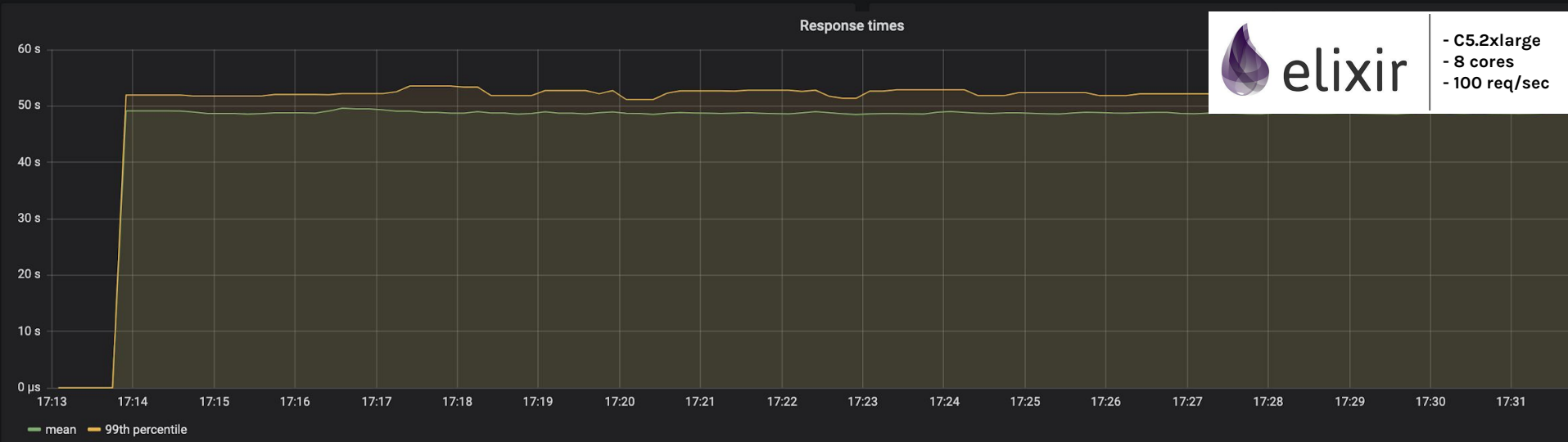
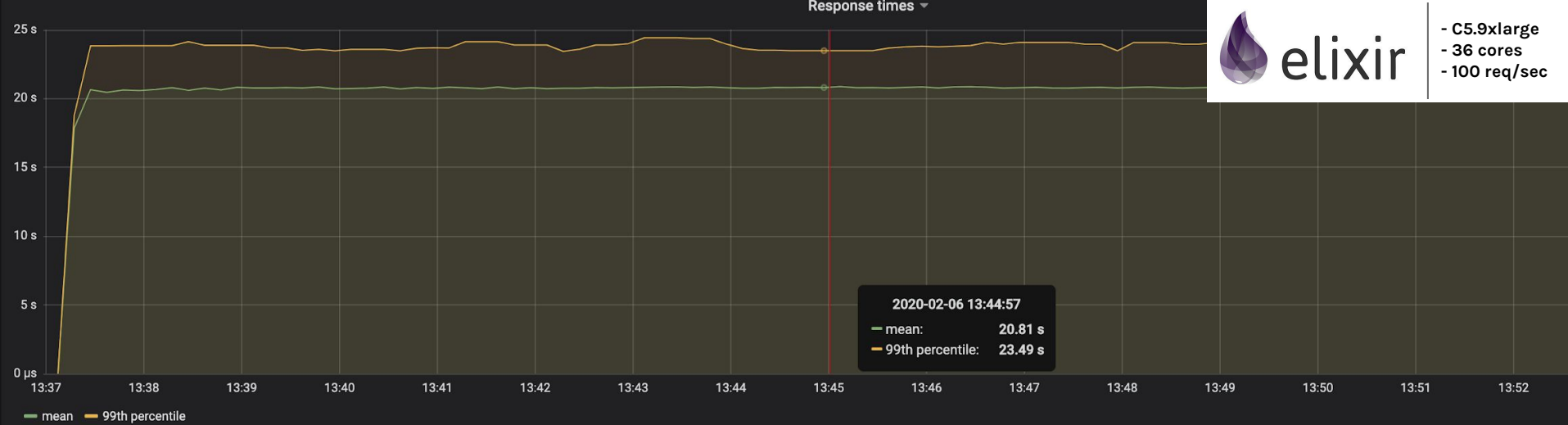


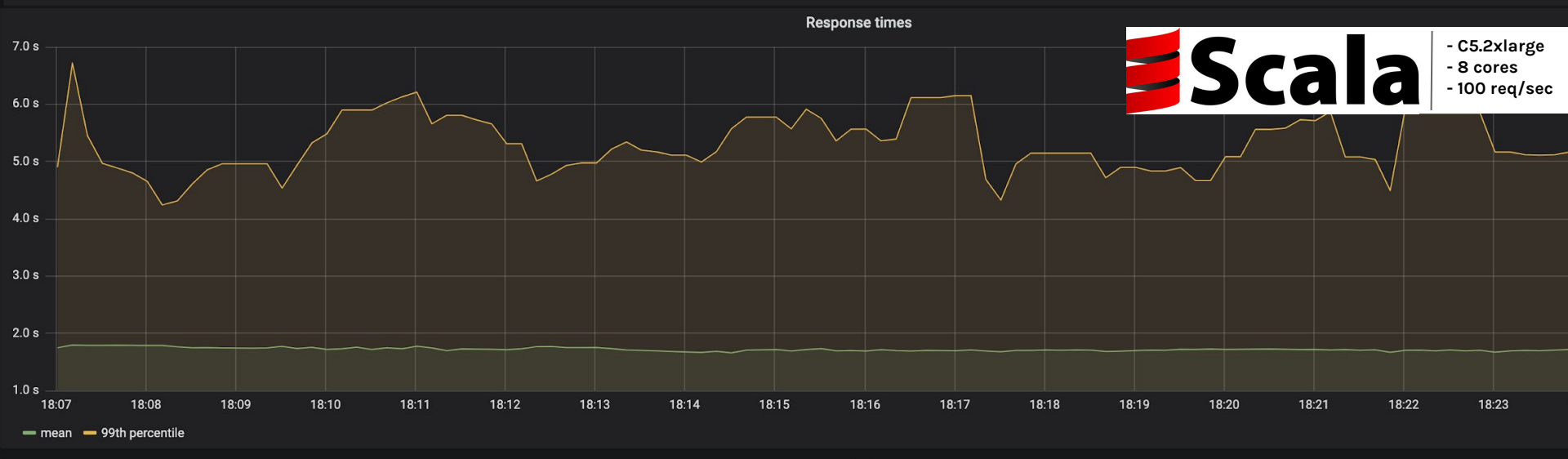
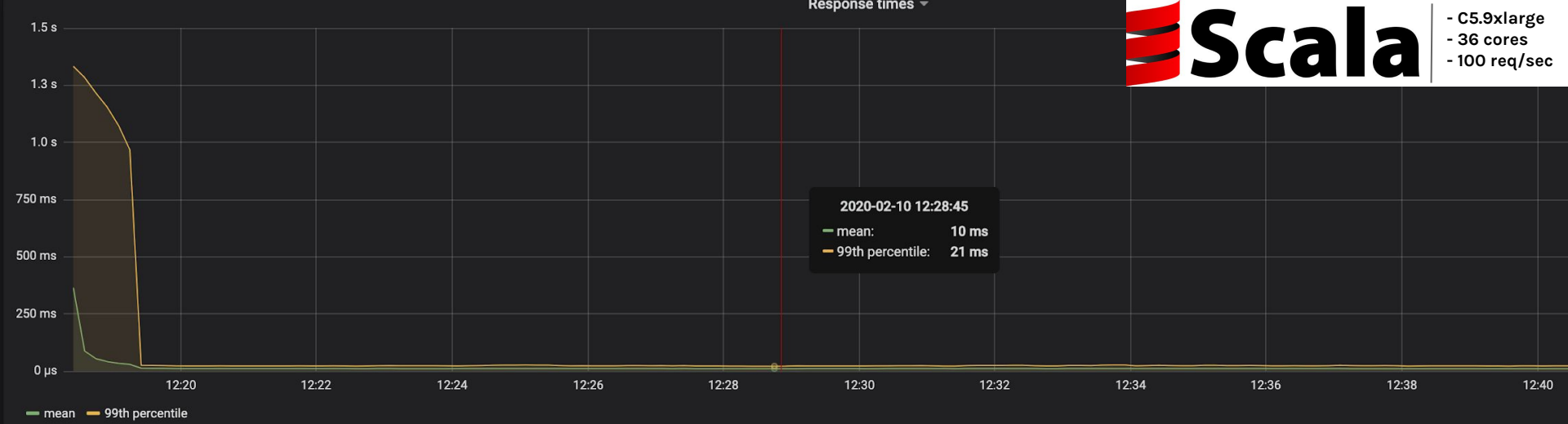
System under test CPU usage



**Couldn't we just avoid using native code?**







# 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 is faster than Elixir one
- At some point adding more cores does not improve the overall performance
- Writing native implementation is **not that** hard