Funkcyjne serwery w Javie

Ratpack

What???

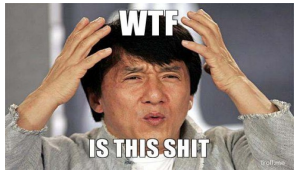
- Why?
- Basic Ratpack
- Non blocking architecture
- But why?
- More complex scenario
- Alternatives

This is not Ratpack manual

Why

- JavaEE/Spring are standards
- Super easy to create REST Servers
  - @RequestMapping("/hello")
- But this is magic
  - Beans, proxys, thread locals, classpath scanning etc.
- Sometime magic stops working
- Problems with testability,
- Boring code (getters and setters)

Not convinced



Think about this Spring annotation @NoRepositoryBean

There are alternatives

- Scala/Akka-http
- NodeJS/Express
- Kotlin/Ktor
- Java/Spring5 – web flux!!!
- Java/Ratkpack *

Code:
https://github.com/javaFunAgain/ratpackSchool

Hello world!

- git checkout STEP_HELLO
- build.gradle
- MyServer
- curl http://localhost:5050/jdd_webinar

Simply stupid

More 1

- git checkout STEP_ADD1
- serverConfig,
- prefix,
- get
- curl http://localhost:5050/add/7

More 2

- git checkout STEP_INC
- prefix,
- get,
- post
- curl -d "" http://localhost:5050/counter/inc
- git checkout STEP_INCCLEAN

Threads ?

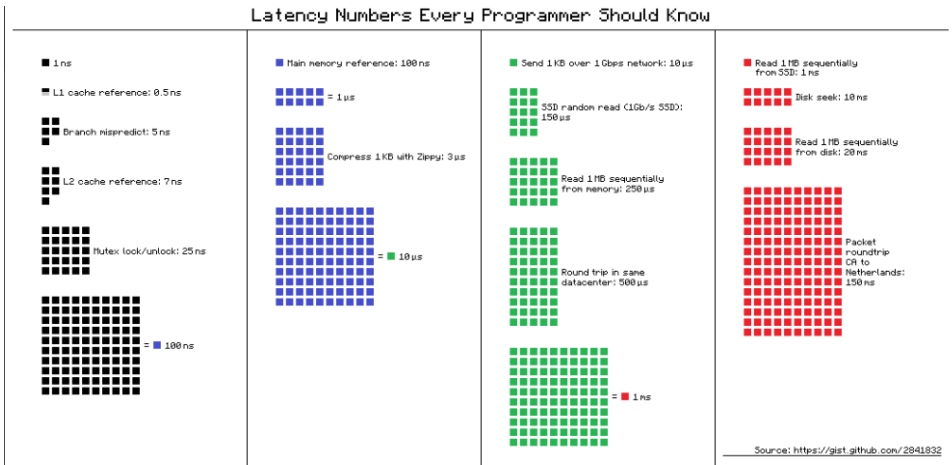
Sleep well my thread

- git checkout STEP_SLEEP
- Server config/ port/ threads
- curl -d "" http://localhost:8080/counter/inc
- ab -m POST -n 10 -c 5 http://localhost:8080/counter/inc

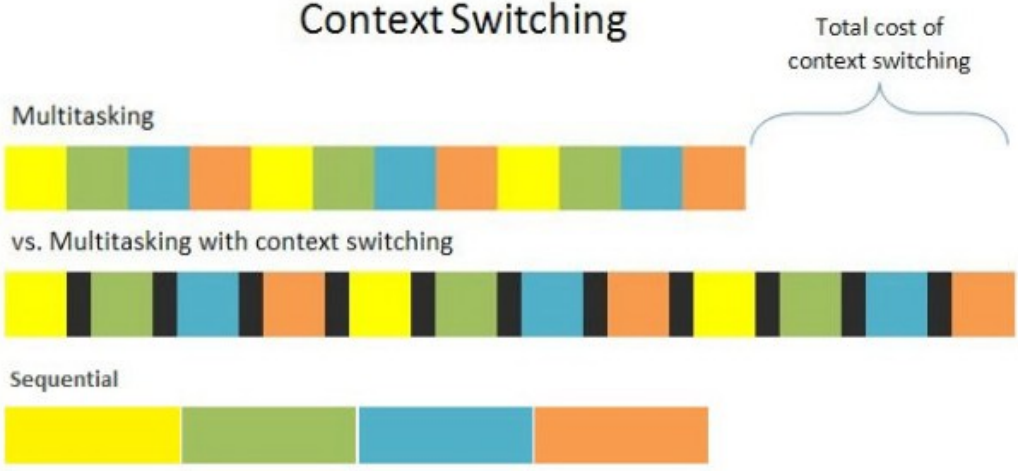
```
return cfg
        .development(true)
        .port(8080)
        .threads(1);
```

More threads -> better?

Latencies


Latency Numbers Every Programmer Should Know
1 ns
L1 cache reference: 0.5 ns
Branch mispredict: 5 ns
L2 cache reference: 7 ns
Mutex lock/unlock: 25 ns
= 100 ns
Main memory reference: 100 ns
= 1 µs
Compress 1K bytes with Zippy: 3 µs
= 10 µs
Send 1K bytes over 1 Gbps network: 10 µs
SSD random read (1Gb/s SSD): 150 µs
Read 1 MB sequentially from memory: 250 µs
Round trip in same datacenter: 500 µs
= 1 ms
Read 1 MB sequentially from SSD: 1 ms
Disk seek: 10 ms
Read 1 MB sequentially from disk: 20 ms
Packet roundtrip CA to Netherlands: 150 ms
Source: https://gist.github.com/2841832

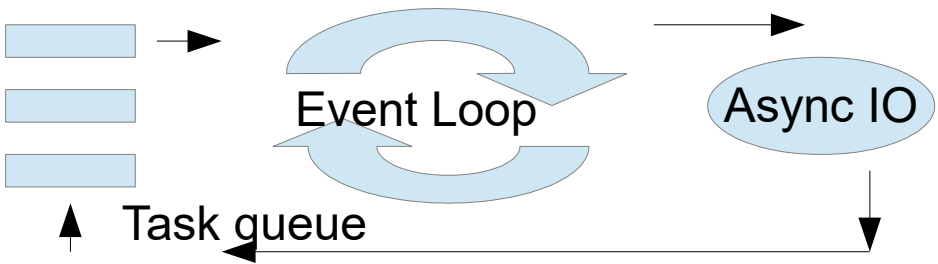
Context Switching



Essence of non blocking

Non blocking

- Tasks/events queue
- Executors
- Small number of threads
- Effective use of CPU caches



The third most useless fibbonacci implementation You will ever see

Fibbonacci

- git checkout STEP_FIBB
- ab -n 100 -c 5 http://localhost:8080/fibb/10
- !!! add Spring alternative

Hardest lines

```
Promise<Long> fibb1 = httpClient
  .get(new URI("http://localhost:8080/fibb/" + (n – 1)))
  .map(response ->Long.parseLong(response.getBody().getText())).fork();
Promise<Long> fibb2 = httpClient
  .get(new URI("http://localhost:8080/fibb/" + (n – 2)))
  . map(response -> Long.parseLong(response.getBody().getText())).fork();

Promise<Long> result = fibb1.flatMap( n1 -> fibb2.map( n2-> n1+n2));
```


IT'S A
TRAP

Monad

- A container
- map : Optional<String> -> Optional<Long>
- flatMap : (Optional<Optional<String>> -> Optional<String>

Monads: lets encapsulate mutability, state and all dirty stuff behind
pure functional code

Non blocking

- Async IO !
- Effective use of CPU (L1, L2)
- Not instantly faster... -> but tunable

Testing!

- git checkout STEP_TEST
- JUNIT 5
- build.gradle (:/)
- test
- Test HTTP ! How fast?

Let's end with this philosophy

Reality – DB + JSON

Persistence

- git checkout STEP_JSON
- JOOQ
- build.gradle
- Persistence class
- VAVR

JSON

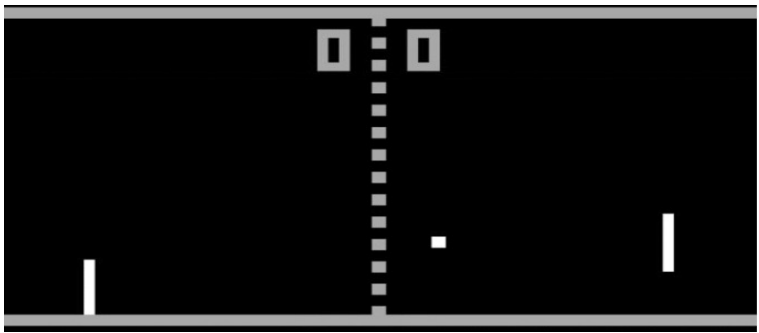
- Mappers
  - .registerModule(new ParameterNamesModule())
  - .registerModule(new Jdk8Module())
  - .registerModule(new JavaTimeModule())
  - .registerModule(new VavrModule());
- ItemIn class
- Parsing
- Rendering
- curl -d '{"points" :2, "info": "webinar przygotowanie" }' -H "Content-Type: application/json" http://localhost:8080/life

Blocking code

- Blocking.get
- Executors

Full sample - game
https://github.com/javaFunAgain/ratpong

Pong check (1973)



https://github.com/javaFunAgain/ratpong

Summary

- Hipster in Java
- Normal elsewhere
- Immutablity, clean / functional code
- Testable
- Non blocking
- Spring 5 can make approach popular
- Try on small sample first
- Kotlin ?