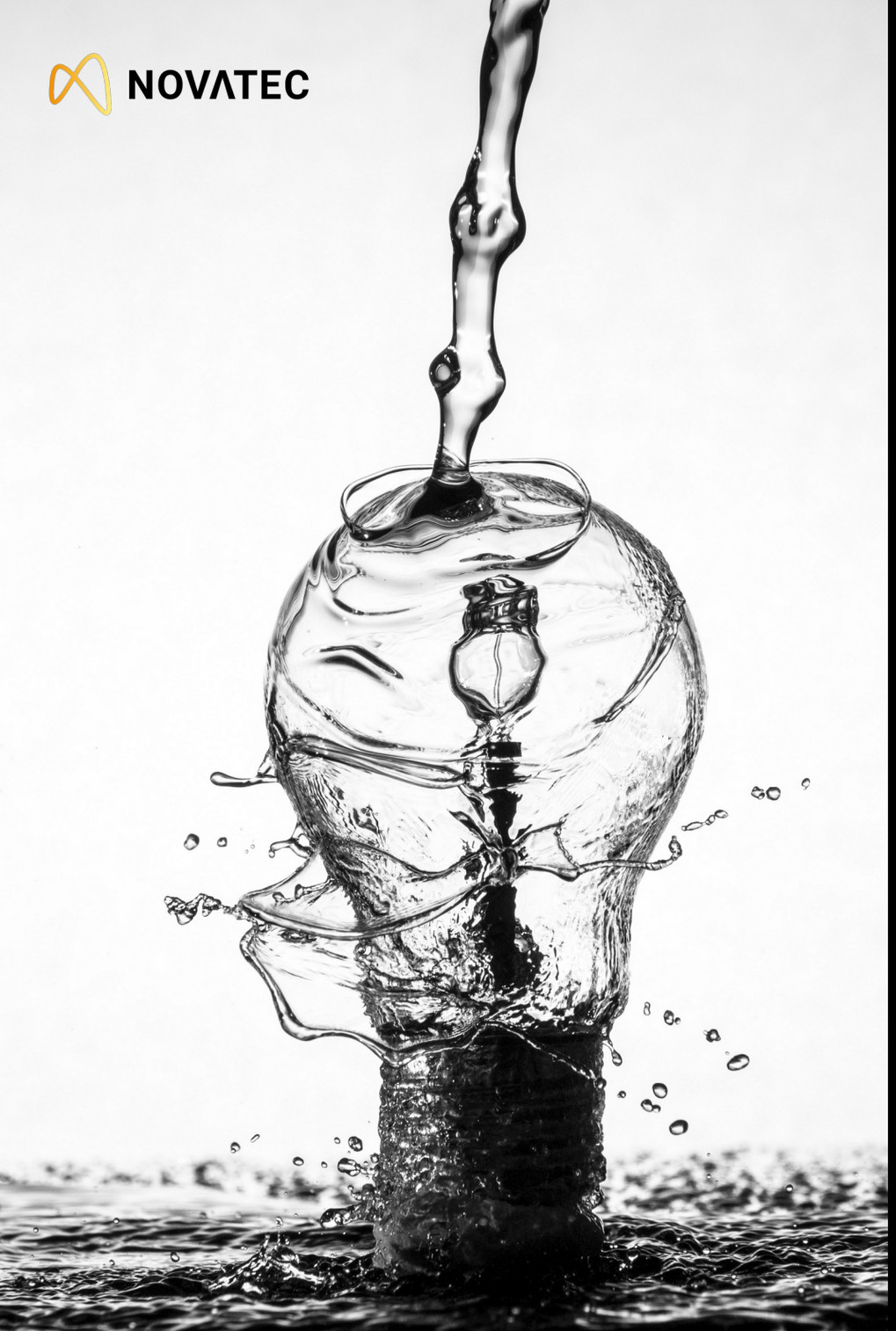


NOVATEC

# Introduction to Spring Native

## Novatec Summit 2021

Christian Schaible

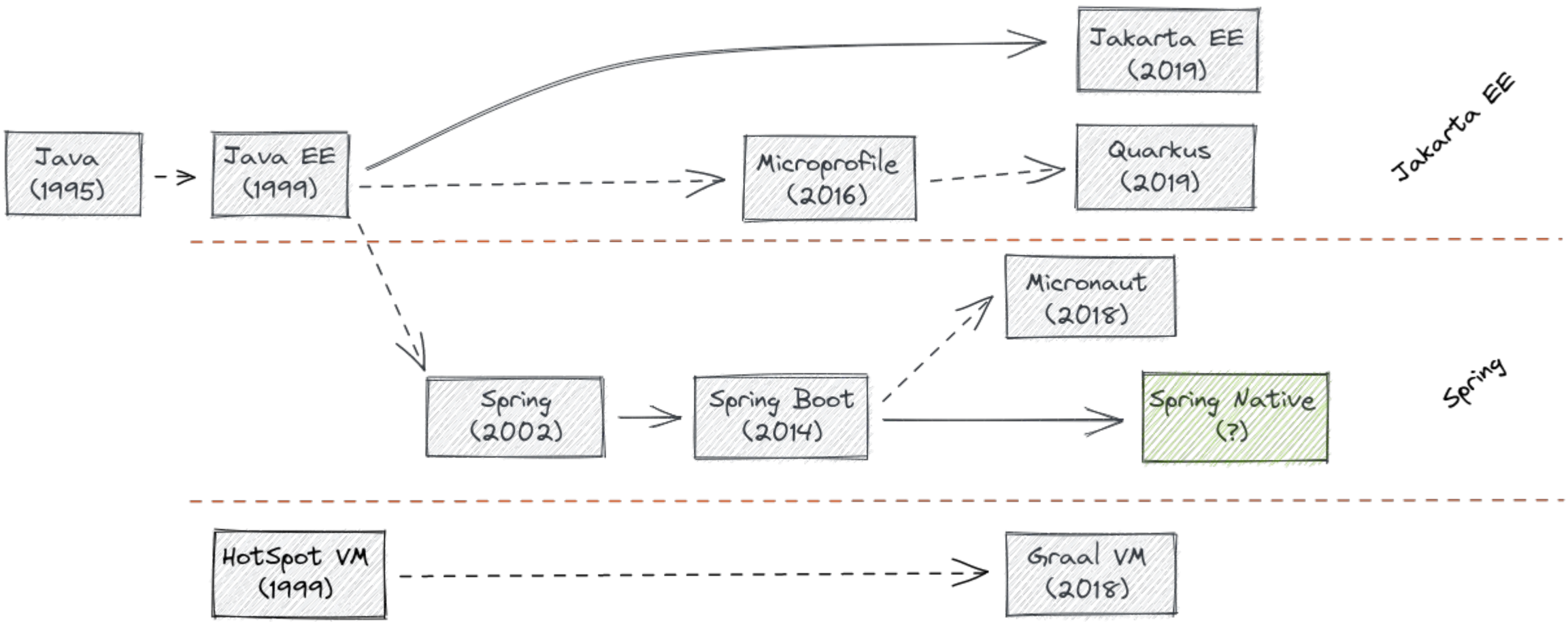
# What to expect?

- Talk about concepts, configuration and tooling
- No demos
- Many slides

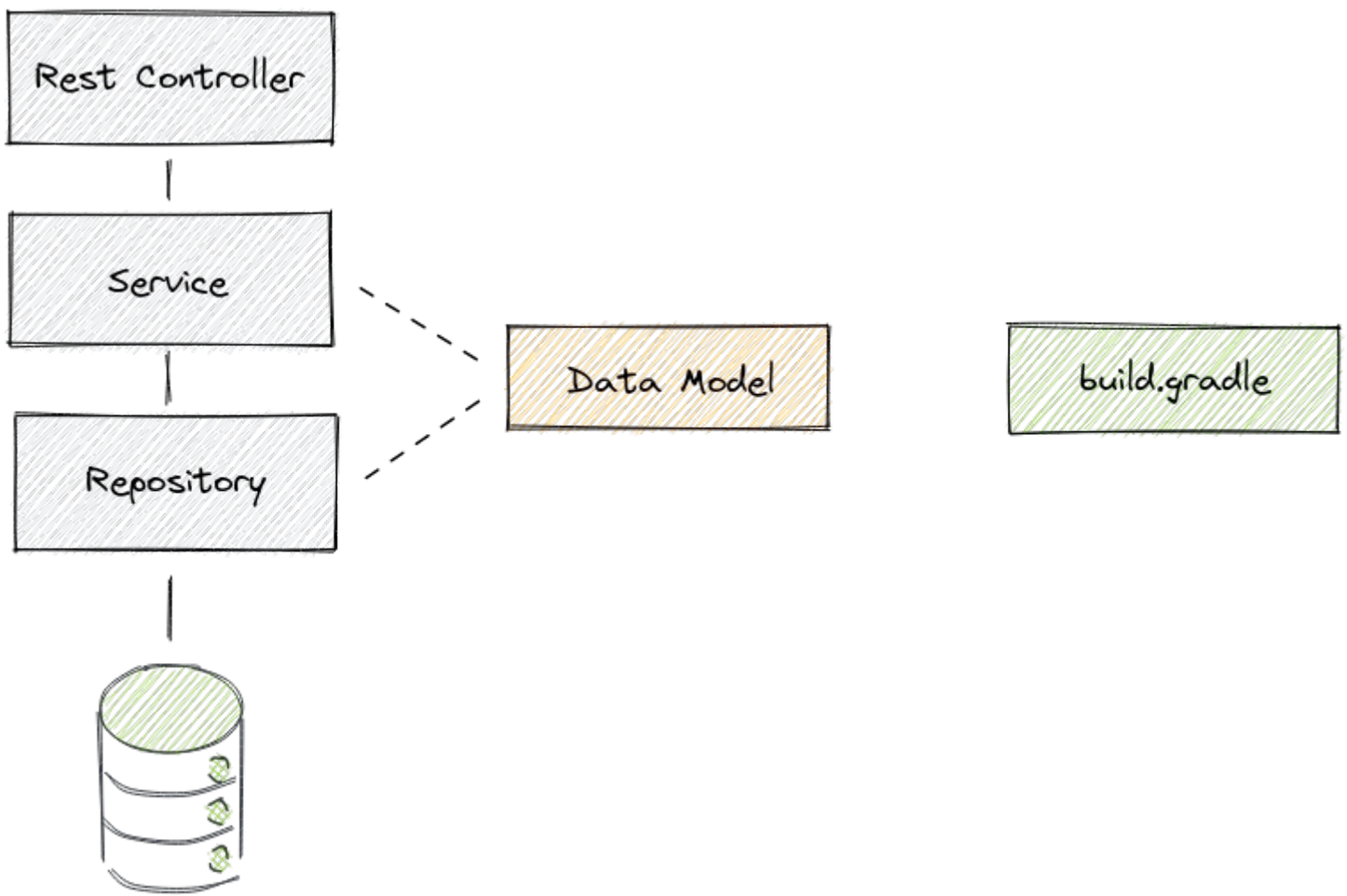
# Terminology

- Binary <-> Native Image
- ...

# Technology landscape



# Spring Boot


Rest Controller
Service
Repository
Data Model
build.gradle

# Spring Native

# Spring Native


**settings.gradle**

```
pluginManagement {
    repositories {
        mavenCentral()
        maven { url = uri("https://repo.spring.io/release") }
        // …
    }
}
```

Plugin Repository

**build.gradle**

```
plugins {
    id("org.springframework.boot") version "2.5.6"
    id("org.springframework.experimental.aot") version "0.10.5"
    id("org.graalvm.buildtools.native") version "0.9.7.1"
    // …
}

nativeBuild {
    classpath(tasks.named("processAotResources").get().outputs,
              tasks.named("compileAotJava").get().outputs)
}

repositories {
    mavenCentral()
    maven { url = uri("https://repo.spring.io/release") }
    // ...
}
```

Plugins

Task Configuration

Repository

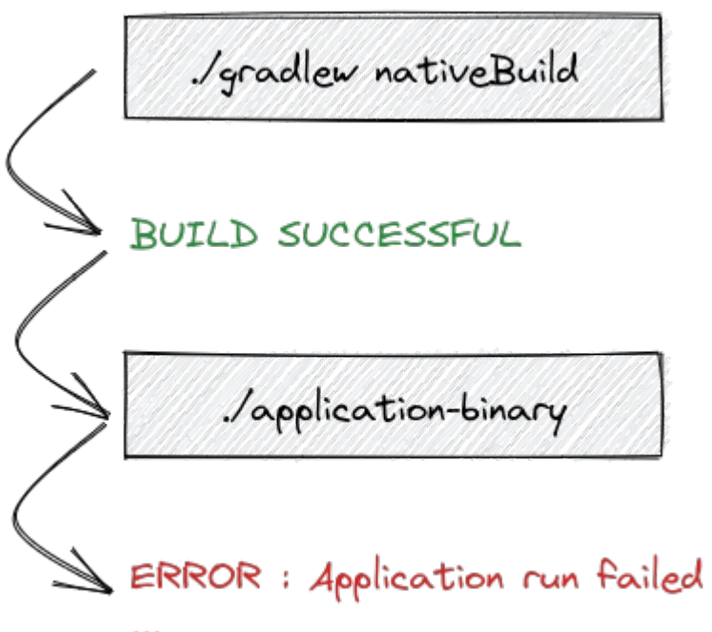
# Spring Native

```
$./gradlew nativeBuild

Starting a Gradle Daemon, 1 busy Daemon could not be reused, use --status for details

> Task :kaptKotlin

Note: Hibernate JPA 2 Static-Metamodel Generator 5.4.32.Final

> Task :compileJava

Note: /home/user/csc.native.sample/src/main/java/io/nvtc/csc/sample/ScannerSubstitutions.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

> Task :generateAot

Failed verification check: this type was requested to be added to configuration but is not resolvable:
org.springframework.security.config.annotation.web.configuration.AutowiredWebSecurityConfigurersIgnoreParents it will be skipped
Failed verification check: this type was requested to be added to configuration but is not resolvable:
org.springframework.boot.test.autoconfigure.jdbc.JdbcTest it will be skipped
…

> Task :compileAotJava

warning: unknown enum constant When.MAYBE
  reason: class file for javax.annotation.meta.When not found
Note: Some input files use or override a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: /home/user/csc.native.sample/build/generated/sources/aot/org/springframework/aot/StaticSpringFactories.java uses unchecked
or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
```

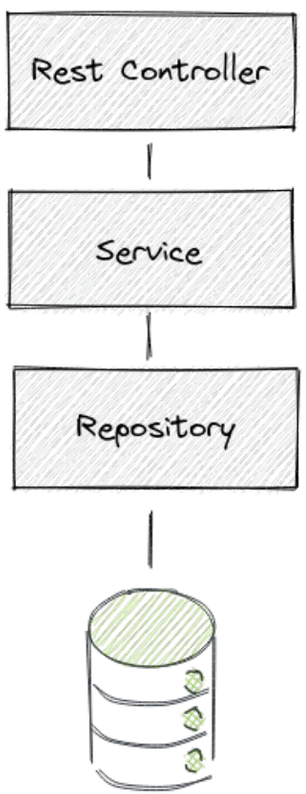
# Spring Native

```
> Task :generateResourcesConfigFile

[native-image-plugin] Resources configuration written into
/home/user/csc.native.sample/build/native/generated/generateResourcesConfigFile/resource-config.json

> Task :nativeBuild

[csc.native.sample:58862]    classlist:   7,664.26 ms,  1.69 GB
[csc.native.sample:58862]        (cap):     733.75 ms,  1.69 GB

Warning: Could not resolve com.google.gson.FieldNamingPolicy for reflection configuration. Reason:
java.lang.ClassNotFoundException: com.google.gson.FieldNamingPolicy.
Warning: Could not resolve com.google.gson.Gson for reflection configuration. Reason: java.lang.ClassNotFoundException:
com.google.gson.Gson.
Warning: Could not resolve com.google.gson.GsonBuilder for reflection configuration. Reason: java.lang.ClassNotFoundException:
com.google.gson.GsonBuilder.
…

[csc.native.sample:58862]     (clinit):   2,557.71 ms,  8.32 GB
[csc.native.sample:58862]   (typeflow):  36,898.83 ms,  8.32 GB
[csc.native.sample:58862]    (objects): 172,290.27 ms,  8.32 GB
[csc.native.sample:58862]   (features):  36,691.43 ms,  8.32 GB
[csc.native.sample:58862]     analysis: 257,058.39 ms,  8.32 GB
[csc.native.sample:58862]     universe:  11,592.94 ms,  8.29 GB
[csc.native.sample:58862]      (parse):   6,618.42 ms,  8.00 GB
[csc.native.sample:58862]     (inline):  20,724.55 ms,  7.24 GB
[csc.native.sample:58862]    (compile): 112,557.17 ms,  7.04 GB
[csc.native.sample:58862]      compile: 148,415.82 ms,  7.04 GB
[csc.native.sample:58862]        image:  13,002.30 ms,  7.13 GB
[csc.native.sample:58862]        write:   1,680.15 ms,  7.13 GB
[csc.native.sample:58862]      [total]: 443,604.64 ms,  7.13 GB

BUILD SUCCESSFUL in 7m 57s

16 actionable tasks: 16 executed
```

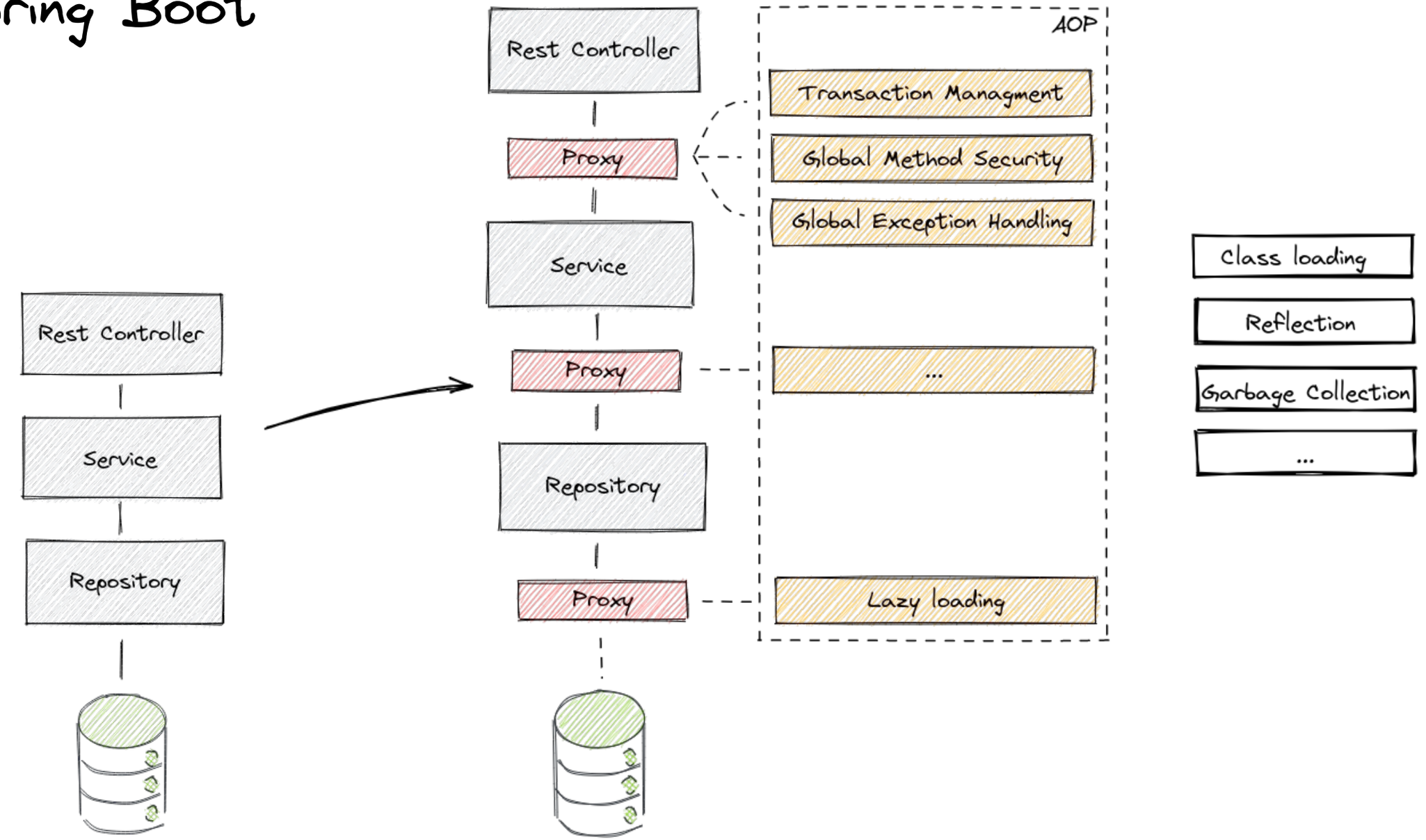
# Spring Native



# Spring Boot



# Spring Boot


Rest Controller
Service
Repository
Rest Controller
Proxy
Service
Proxy
Repository
Proxy
AOP
Transaction Managment
Global Method Security
Global Exception Handling
...
Lazy loading
Class loading
Reflection
Garbage Collection
...

# GraalVM – Hints

- proxy-config.json

```
[
  [
    "org.springframework.transaction.annotation.Transactional",
    "org.springframework.core.annotation.SynthesizedAnnotation"
    …
  ],
  …
]
```

- reflect-config.json

```
[
  {
    "name": "ch.qos.logback.classic.pattern.DateConverter",
    "allDeclaredFields": true,
    …
  },
  …
]
```

- resource-config.json

```
[ … ]
```

- serialization-config.json

```
[ … ]
```

- ...

Put them into:

src/main/resources/
META-INF/native-image/

# Spring AOT – Typesafe Hints

Config types:

- proxy-config.json
  `@AotProxyHint` // Class needs a proxy
  `@JdkProxyHint`
- reflect-config.json
  `@TypeHint`
  `@FieldHint` // reflection access on a field
  `@MethodHint` // reflection access of a method
- resource-config.json
  `@ResourceHint` // Resource file should be packaged into the native image
- serialization-config.json
  `@SerializationHint` // Type should be serializable

Container-Types:

- `@NativeHint`

Initialization Types:

- `@InitializationHint` // Initialization at build/runtime

# Spring AOT – Typesafe Hints

**src/main/kotlin/.../Application.kt**

```
@NativeHints(
  NativeHint(
    trigger = Driver::class,
    types = [ TypeHint(
                types = [Options::class],
                fields =  [ FieldHint(name = "user"), ... ]
              ) ],
    resources = [ ResourceHint(patterns = ["mariadb.properties"]) ]
  )
)
@SpringBootApplication
class Application { ... }
```

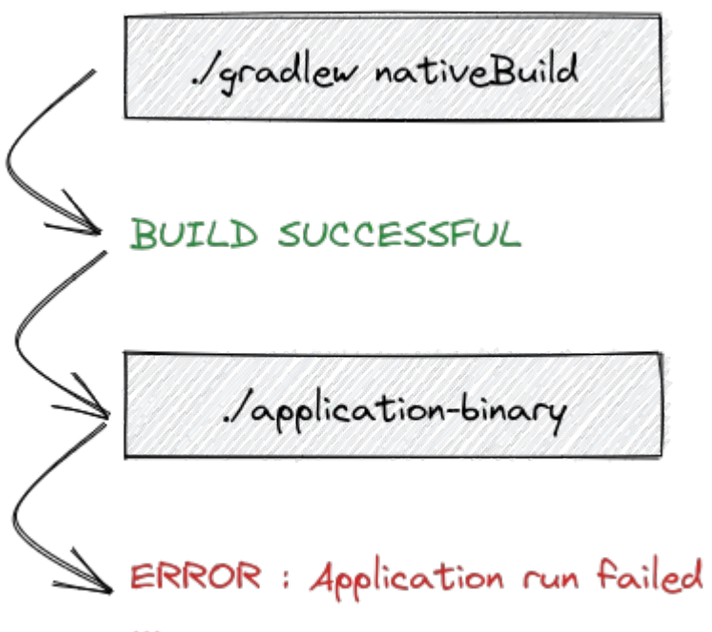
**build/resources/aot/META-INF/native-image/org.springframework.aot/spring-aot/reflect-config.json**

```
[
  {
    "name": "org.mariadb.jdbc.util.Options",
    "fields": [ { "name": "user" }, … ]
  },
  …
]
```

**build/resources/aot/META-INF/native-image/org.springframework.aot/spring-aot/resource-config.json**

```
{
  "resources": {
    "includes": [ { "pattern": "mariadb.properties" }, ... ]
  },
  …
}
```

# Spring Native



# Spring Native – Typical Errors

```
Failed to instantiate [org.springframework.kafka.core.DefaultKafkaConsumerFactory]:
    Factory method 'kafkaConsumerFactory' threw exception;
    nested exception is org.apache.kafka.common.config.ConfigException:
    Invalid value io.confluent.kafka.serializers.subject.RecordNameStrategy
    for configuration key.subject.name.strategy:
    Class io.confluent.kafka.serializers.subject.RecordNameStrategy could not be found.
…
    at org.apache.kafka.common.config.ConfigDef.parseType(ConfigDef.java:744) ~[na:na]
```

**//ConfigDev.java**

```
public static Object parseType(String name, Object value, Type type) {
    …
    String trimmed = null;
    if (value instanceof String)
        trimmed = ((String) value).trim();
    …
    else if (value instanceof String) {
        ClassLoader contextOrKafkaClassLoader = Utils.getContextOrKafkaClassLoader();
        Class<?> klass = contextOrKafkaClassLoader.loadClass(trimmed);
```

## Solution:

```
@TypeHint(types = [RecordNameStrategy::class])
```

# Spring Native – Typical Errors

```
java.lang.NoSuchFieldException: password
    at java.lang.Class.getField(DynamicHub.java:1139)
    at org.mariadb.jdbc.util.DefaultOptions.parse(DefaultOptions.java:877)
    at org.mariadb.jdbc.util.DefaultOptions.parse(DefaultOptions.java:861)
    at org.mariadb.jdbc.UrlParser.defineUrlParserParameters(UrlParser.java:263)
    at org.mariadb.jdbc.UrlParser.parseInternal(UrlParser.java:225)
    at org.mariadb.jdbc.UrlParser.parse(UrlParser.java:180)
    at org.mariadb.jdbc.Driver.connect(Driver.java:85)
    at com.zaxxer.hikari.util.DriverDataSource.getConnection(DriverDataSource.java:121)
```

```
// DefaultOptions.java

private static Options parse(…) {
    final Options options = paramOptions != null ? paramOptions : new Options();
    …
    for (final Object keyObj : properties.keySet()) {
        final Object propertyValue = properties.get(keyObj);
        final DefaultOptions o = OptionUtils.OPTIONS_MAP.get(keyObj);
        if (o != null && propertyValue != null) {
            final Field field = Options.class.getField(o.optionName);
            …
…
```

## Solution:

```
@TypeHint(types = [Options::class], fields =  [ FieldHint(name = "password"), ... ])
```

# Spring Native – Typical Errors

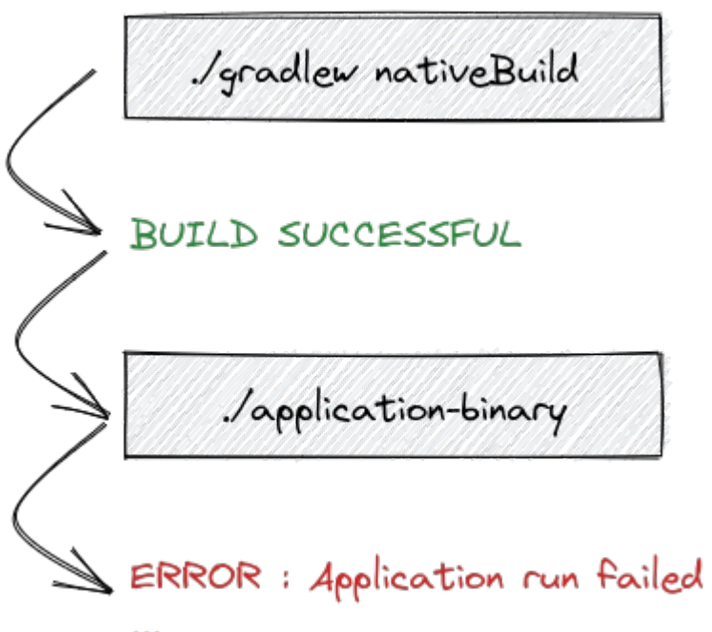
```
property file 'mariadb.properties' not found in the classpath
```

```
// VersionFactory.java

public static Version getInstance() {
    …
    try (InputStream inputStream =
            Version.class.getClassLoader().getResourceAsStream("mariadb.properties")) {
                if (inputStream == null) {
                    System.out.println("property file 'mariadb.properties' not found in the classpath");
                }
            …
…
```

## Solution:

```
@ResourceHint(patterns = "mariadb.properties")
```

# Spring Native



# GraalVM Tracing Agent

Run the app

```
$java -DspringAot=true \
   -agentlib:native-image-agent=config-output-dir=src/main/resources/META-INF/native-image
   -jar build/libs/app.jar

→ Do all kind of representative actions in the app...

<Application runs normally until you press Ctrl+C>
```

Option 1:

- Compile to native and check if the configuration works
- Reduce configuration
- Repeat until the minimum working set is reached

Option 2:

- Sort the configuration generated by the tracing agent in src/main/resources/META-INF/...
- Sort the configuration generated by the native build in build/resources/aot/META-INF/...
- Compare the json files

```
{
    "name": "com.fasterxml.jackson.module.kotlin.SingletonSupport"
},
{
    "allDeclaredConstructors": true,
    "name": "com.fasterxml.jackson.module.paramnames.ParameterNamesModule"
```

```
574  {
575      "name": "com.fasterxml.jackson.module.kotlin.SingletonSupport"
576  },
577  {
578+     "allDeclaredFields": true,
579+     "name": "com.fasterxml.jackson.module.paramnames.ParameterNamesModule",
580+     "queriedMethods": [
581+         {
582+             "name": "setupModule",
583+             "parameterTypes": [
584+                 "com.fasterxml.jackson.databind.Module$SetupContext"
585+             ]
586+         }
587+     ],
588+     "queryAllDeclaredMethods": true
589+ },
```

<- Focus on types not details

# GraalVM native image Features

- Intercept native image generation
- Modify native images dynamically at build time
  - e.g. resolve list of embedded resources in a jar at build time and add them as a static list of resources into the native image

```
@AutomaticFeature
public final class FlywayFeature implements Feature {

  …

  @Override
  public void beforeAnalysis(BeforeAnalysisAccess access) {
    …
  }
}
```

# Substitutions

- Classes, Constructors, Methods and Fields can be replaced at build time
- Existing fields can be referenced

```
@TargetClass(className = "org.flywaydb.core.internal.scanner.Scanner")
public final class ScannerSubstitutions {

  …

  @Alias
  private List<LoadableResource> resources = new ArrayList<>();

  @Substitute
  public void scan(…) {
    var resource = resources.get(0); // Get elements from original resources field in Scanner.java
    …
    doScan(); // Call original doScan() method in Scanner.java
  }

  @Alias
  private Result doScan() {
    return null;
  }
}
```

# Summary

- Static hint annotations
- GraalVM Features / Substitutions
- Tracing Agent
- Code reading

# What does it mean for ...

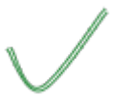
- Logging
- Metrics
- Tracing
- DB migrations
- Garbage collection
- ...

# Logging

- Logback / Slf4j ✓
- Logback.xml customization ✓
- Logging profiles -> Configuration not compilation

```
---
spring:
  config:
    activate:
      on-profile: "log-requests"

logging:
  level:
    io.nvtc.csc.logging.service: debug
```

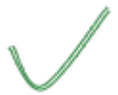
# Metrics

- Micrometer 

```
springAot {
    removeJmxSupport.set(false)
}
```

- JVM Metrics (e.g. jvm.memory.used) 

# Tracing

- Code based tracing (sleuth, zipkin, opentracing, ...) 

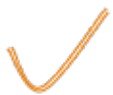
- Java agent based tracing

  -> Java agents are not officially supported 

  -> Datadog agent doesn't work (at the moment) 

```
java -javaagent:/datadog/dd-java-agent.jar ...
```

# Database migrations

- Flyway 
  -> Manual configuration
- Liquibase 
  -> Manual configuration

# Garbage Collection

- Serial (default)
  `--gc=serial`
- Epsilon
  `--gc=epsilon`
- G1 (GraalVM Enterprise)
  `--gc=G1`

# Build times / resources

- High memory consumption at build time
  - Simple app (spring web + kafka + jdbc)
    - JVM: 1.5 GB, < 1m
    - Native: 10 GB, > 5m
- No incremental compiler / No compilation cache
- No cross compilation yet

# Licensing

- LGPL static vs dynamical linking?

  If you distribute a Java application that imports LGPL libraries, it's easy to comply with the LGPL. Your application's license needs to **allow users to modify the library**, and reverse engineer your code to debug these modifications.

  https://www.gnu.org/licenses/lgpl-java.html

  Ask a lawyer!

# Performance

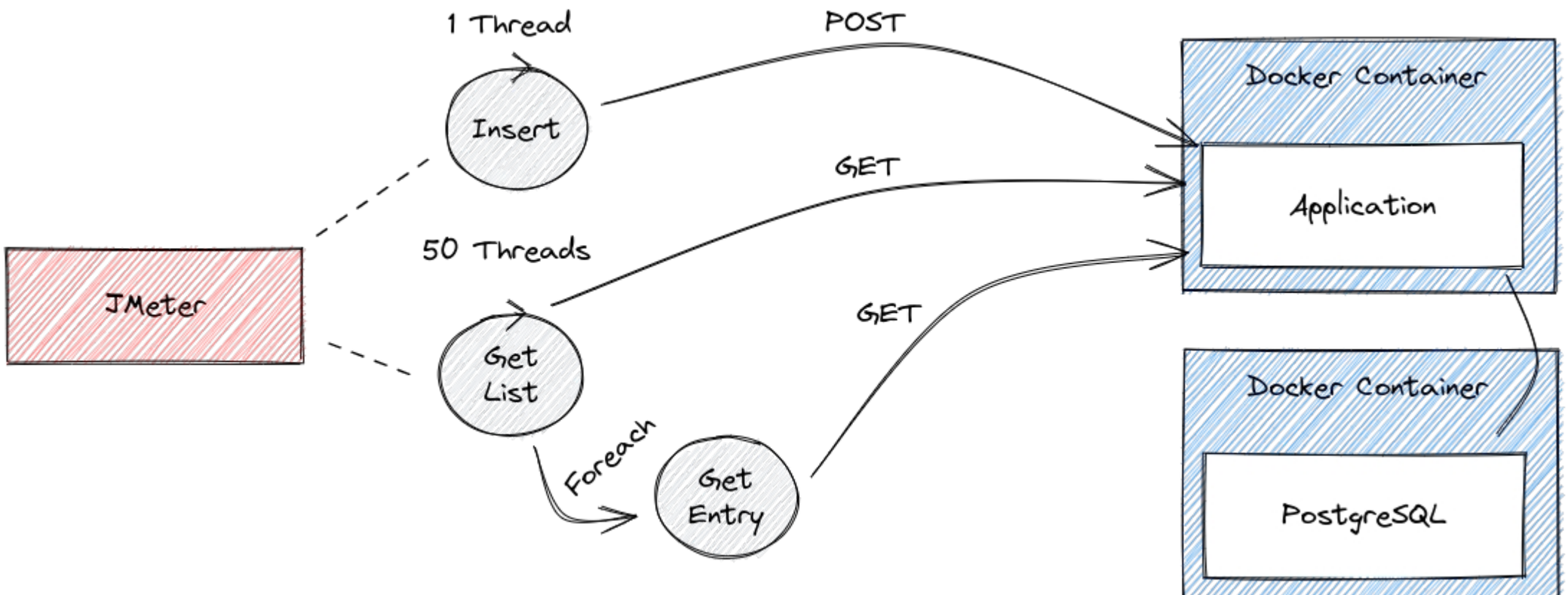
- No optimizations at run time
- Profile Guided Optimization (GraalVM Enterprise)

Rule of thumb

- Startup time
- Peak performance
- Memory Footprint

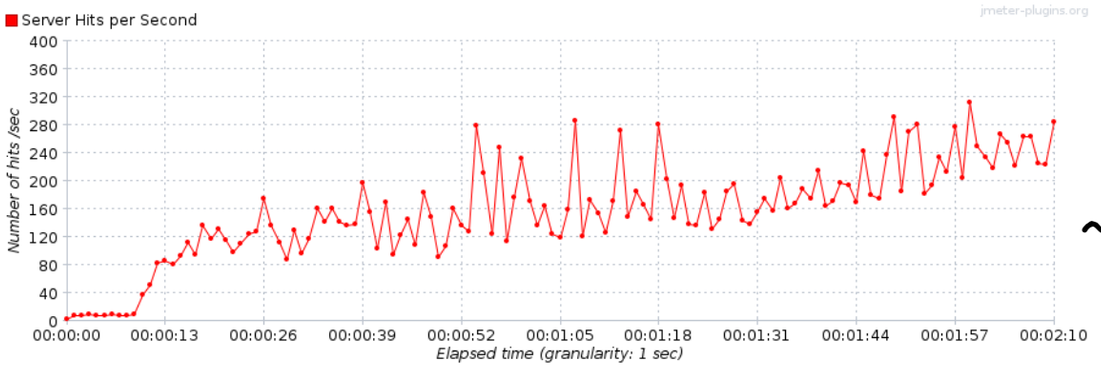
# Performance

- 1 CPU CORE
  Max 512 MB RAM (JVM) / 384 MB RAM (Native)
  Max 15 DB Connections

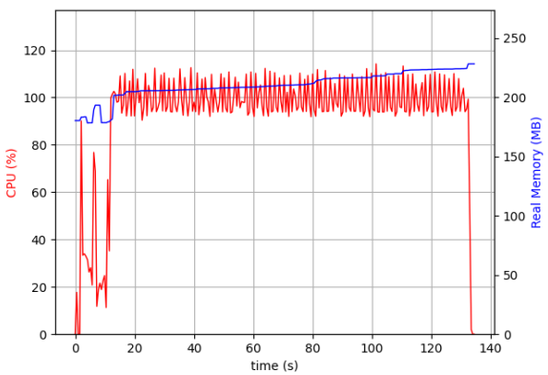


# Performance

- Spring Boot – JVM

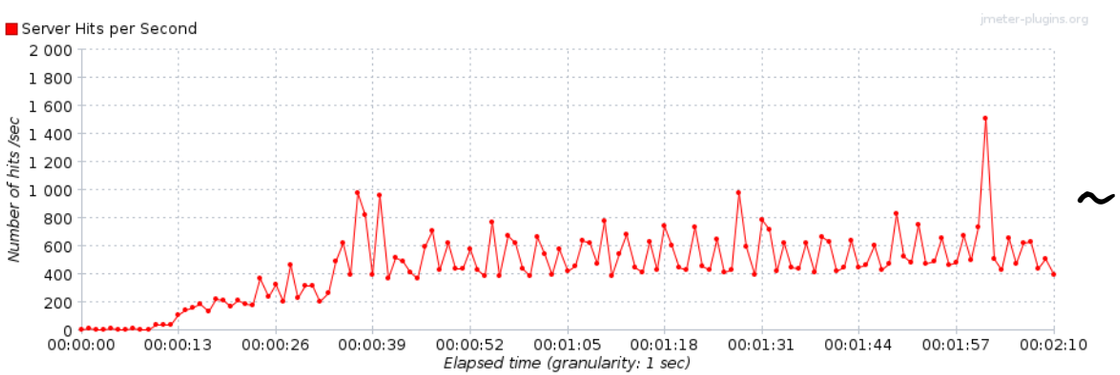



~ 180 R/s

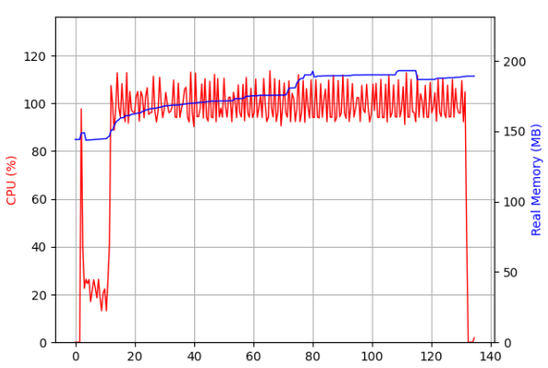



~ 225 MB

- Quarkus – JVM



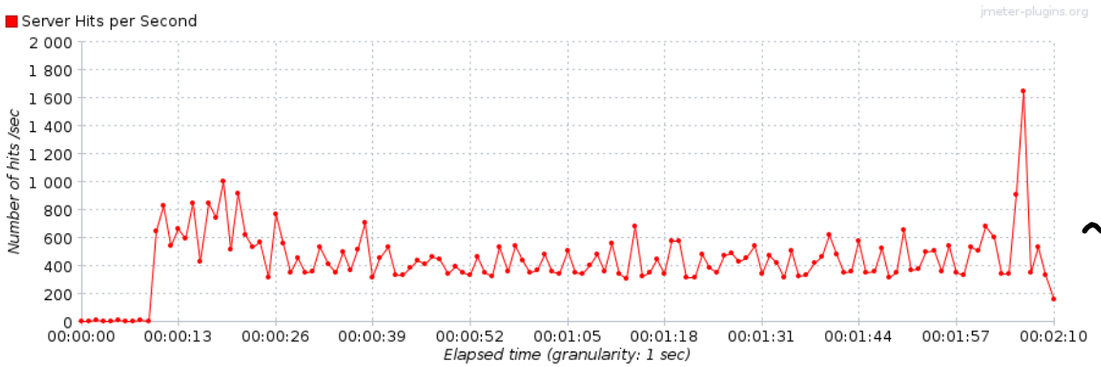

~ 550 R/s



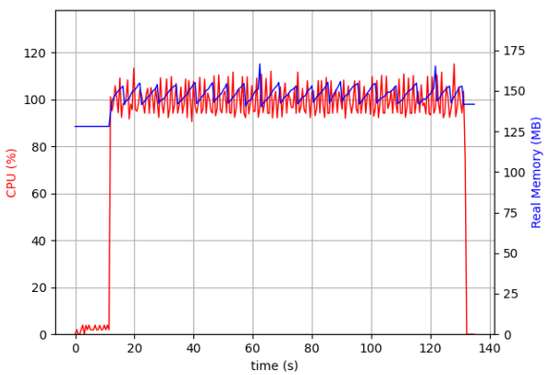

~ 175 MB

# Performance

- Spring Boot – Native

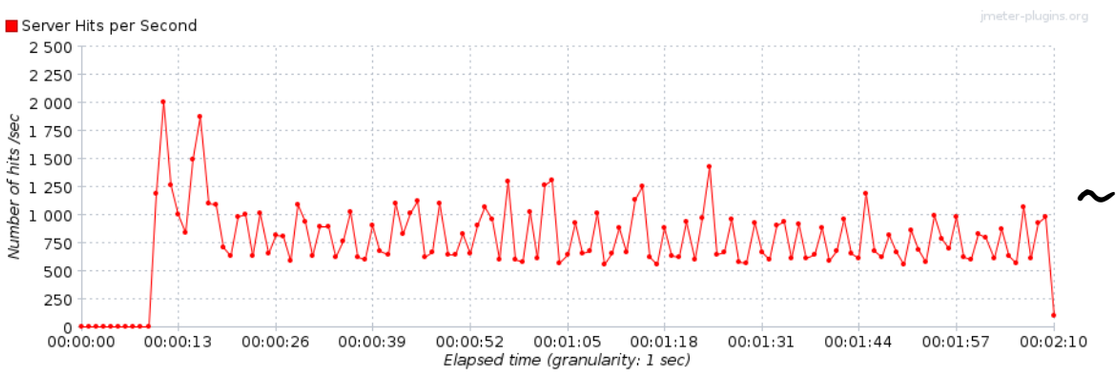



~ 425 R/s

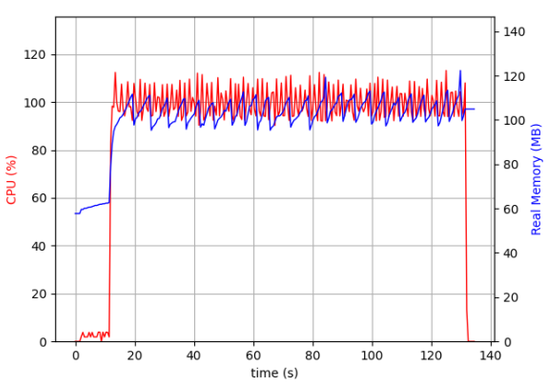



~ 150 MB

- Quarkus – Native



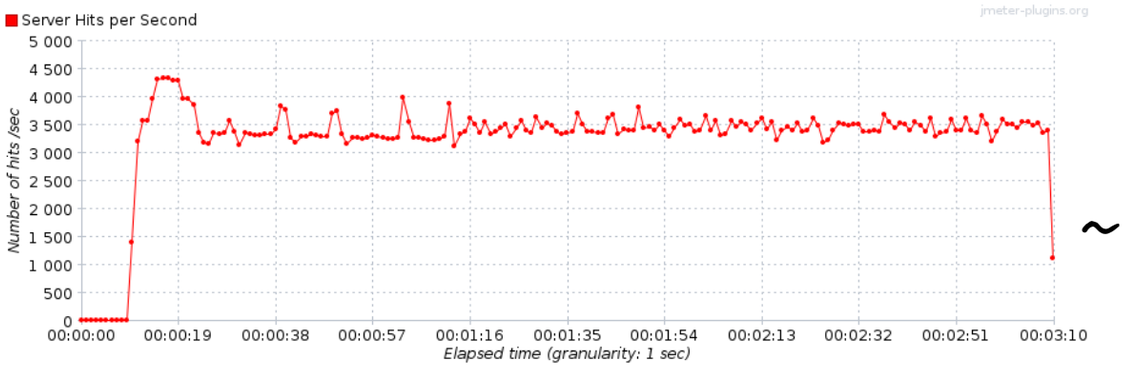

~ 800 R/s



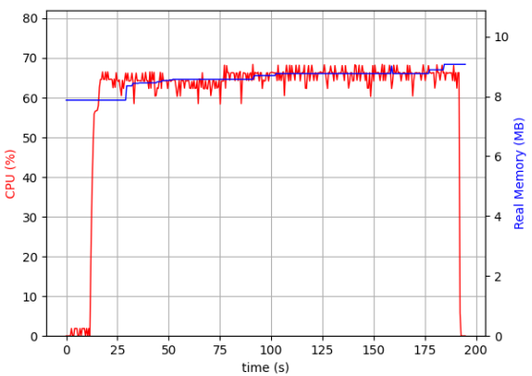

~ 105 MB

# Performance

- Rocket (Rust)




~ 3500 R/s




~ 9 MB

# What else to know?

- JVM Memory parameters:
  `./application-binary -Xmx64m`

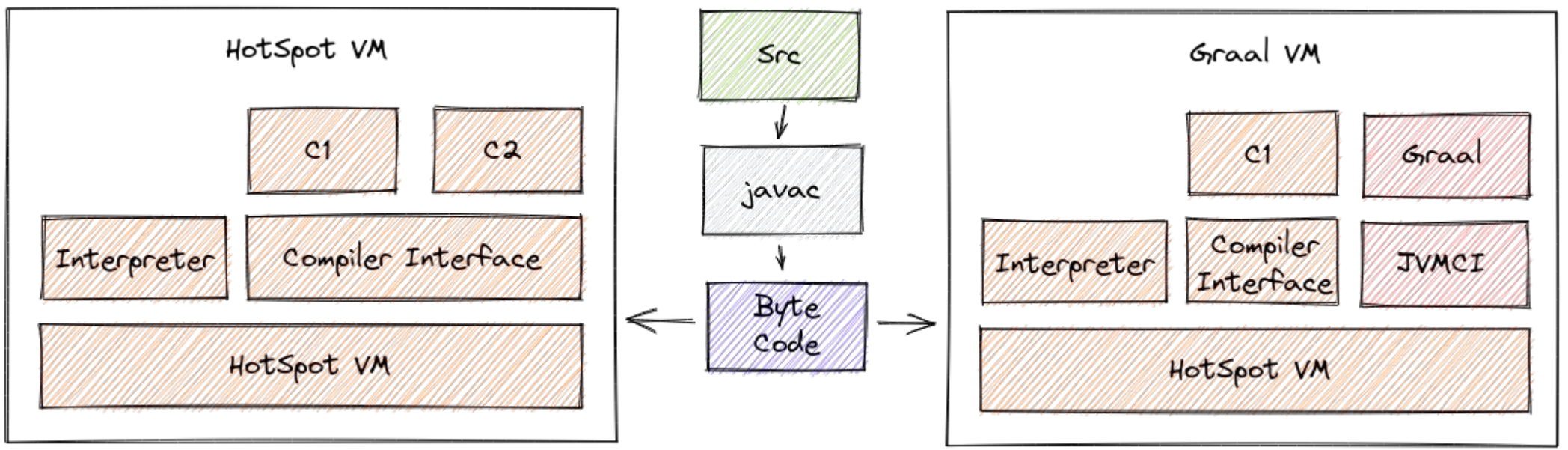
- Spring Features can be removed
  - JMX (default: true)
  - XML (default: true)
  - SPEL (default: false)
  - YAML (default: false)

- AOT Mode on JVM
  `java -DspringAot=true -jar app.jar`

# VMs & Compilers

GraalVM isn't only the native compiler!

## JIT




Tiered Compilation

| Interpreter | C1 (Client) | C2 (Server) |
|---|---|---|
| Level 0 | Level 1 - 3 | Level 4 |

-XX:+TieredCompilation (default)

-XX:-TieredCompilation

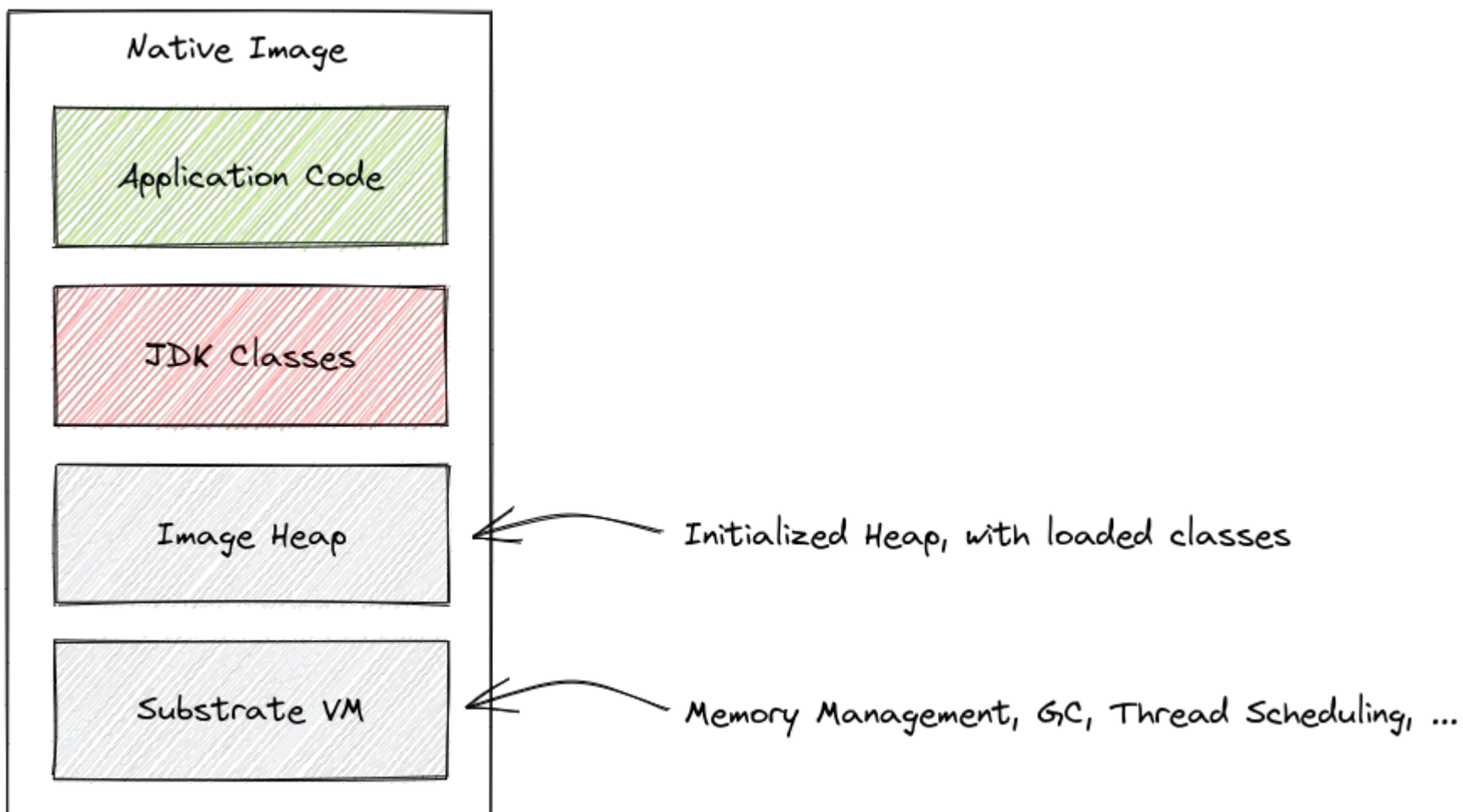
-XX:TieredStopAtLevel=X (X: 0-4)

The JVM uses multiple compilers transparently

OpenJDK 9-16: -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler

# VMs & Compilers

## Native



# Should you use it?

- New application / existing application
- Size of applications
- Initial effort / Change rate
- Maturity (do you use "unsupported" libs?)
- JVM / GraalVM / Native

-> Start small

-> Expect to fail

# Thank you

# Resources

- Spring Native Documentation
- Spring Native – Details explained by Sébastien Deleuze (Video)
- Spring Native repo of all supported libs (hints)
- My sample repo
  - csc.native.jpa (jpa, flyway)
  - csc.native.logging (logging)
  - csc.native.metrics (micrometer, kafka)

# Resources

- GraalVM Architecture / JIT / Aot
- GraalVM Native Image initialization
- GraalVM Native Image documentation
- JVM Compilers / Graal VM Compiler Performance
- JVM JIT Compiler Overview

# Images

- Slide 1: Photo by Sharon Pittaway – light bulb – unsplash.com
- Slide 42: Photo by Daniil Silantev – compass – unsplash.com

**Novatec Consulting GmbH**
Bertha-Benz-Platz 1
D-70771 Leinfelden-Echterdingen

T. +49 711 22040-700
info@novatec-gmbh.de
www.novatec-gmbh.de