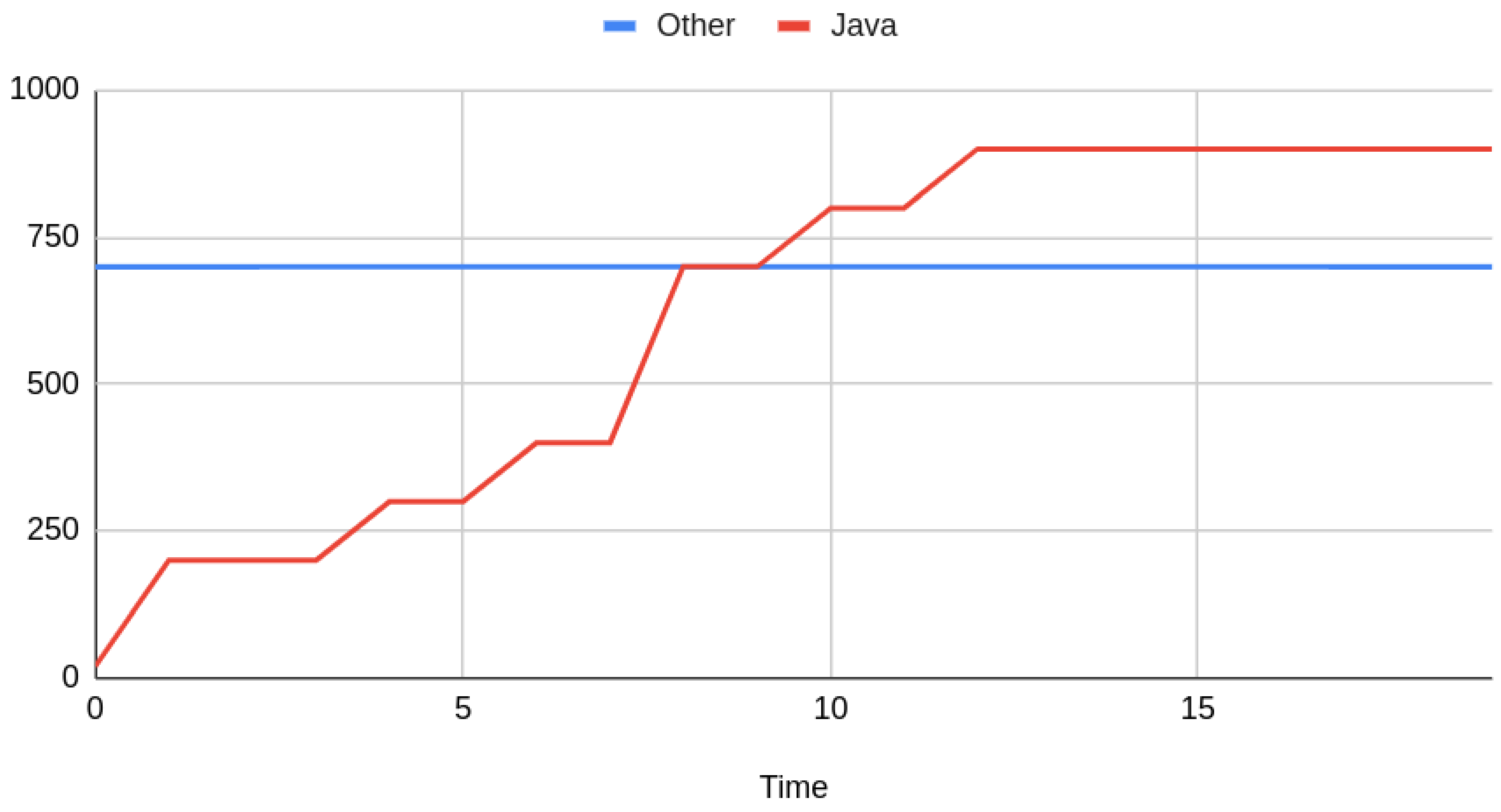
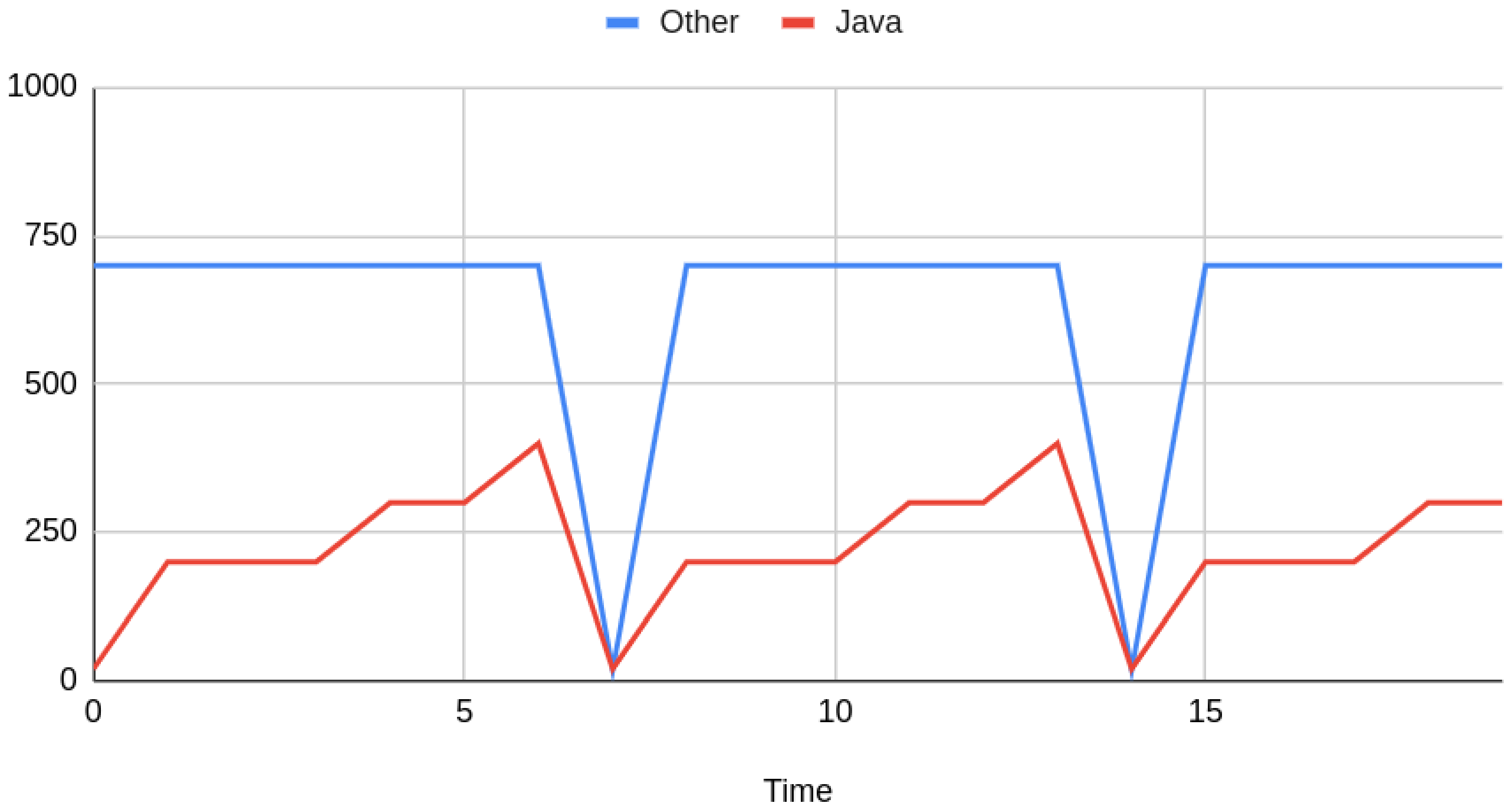
SUPERSONIC. SUBATOMIC. JAVA.

@SanneGrinovero

LONG RUNNING SERVER, PERFORMANCE



WHAT IF... CONTINUOUS DELIVERY



BLACK FRIDAY: OUR WORST NIGHTMARE?

SAVE UP TO
50%
BLACK
FRIDAY
NOW IT'S TIME TO SAVE BIG

WE HAVE A PROBLEM?

Long warmup times are no longer acceptable

ENEMIES OF SLOW STARTUP

Continuous Delivery

Elasticity, scale on cloud: trends, people, reality

I'M SANNE GRINOVERO

Dutch, Italian, living in London.

- Red Hat, middleware engineering
    - R&D
    - Hibernate team lead
    - Quarkus, founding team member
    - Architect, Sr. Principal Software Engineer
- Passionate about all OSS, Java & performance

SUPERSONIC ?

FAST BOOT is now essential

How Quarkus achieves it

S U B A T O M I C ?

LOW MEMORY, high density
How Quarkus achieves it

JAVA ?

Enable use of existing know-how
Leverage all great existing libraries
And yet enable strong innovation

WHAT IS QUARKUS

TOOLKIT

and

FRAMEWORK

for writing Java applications

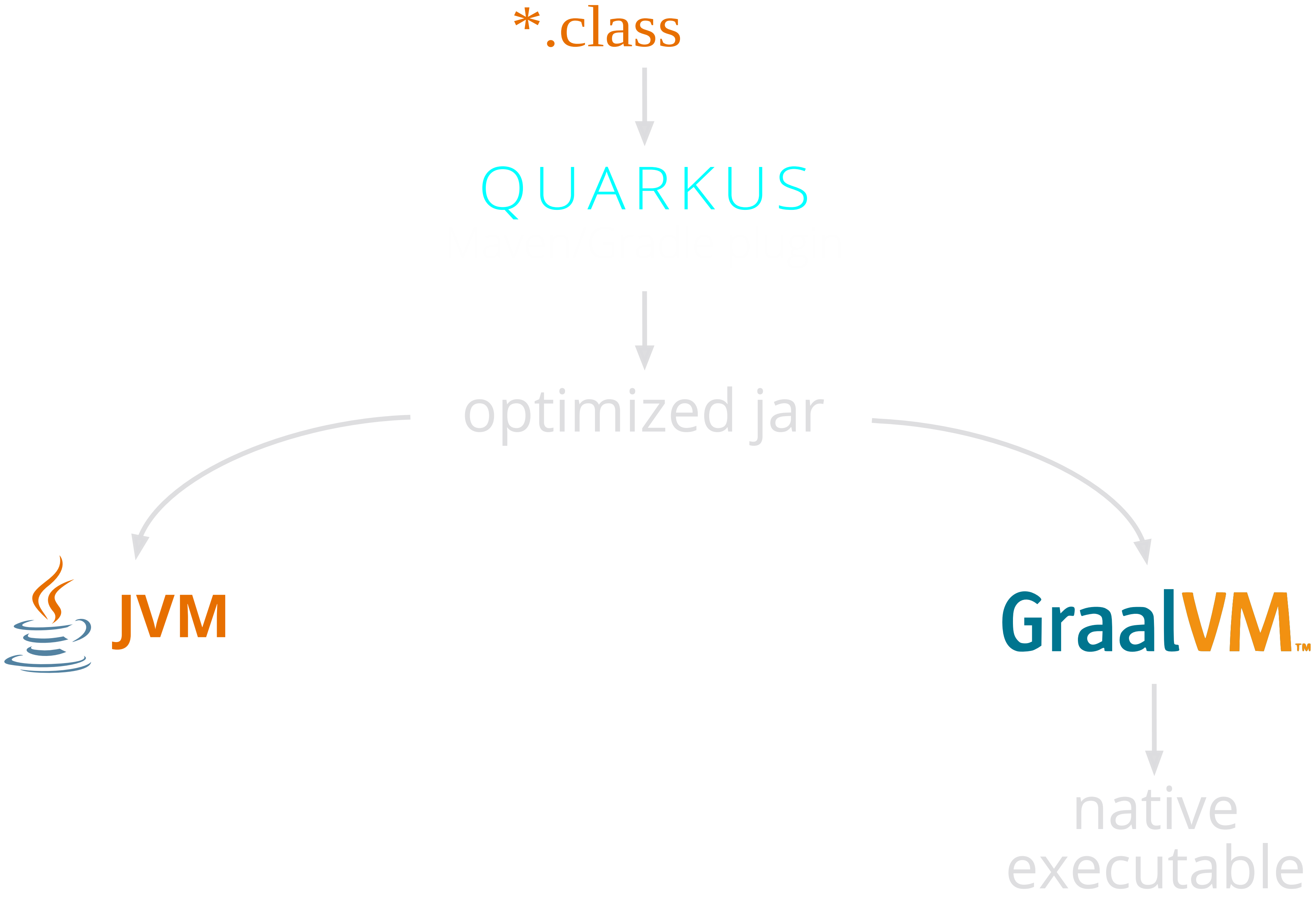
LIGHT, CLOUD FRIENDLY, DESIGNED FOR GRAALVM

Helps overcome limitations of GraalVM

LIGHT, CLOUD FRIENDLY, DESIGNED FOR GRAALVM

Helps overcome limitations of GraalVM

Embrace these limitations, we love them!


*.class
QUARKUS
Maven/Gradle plugin
optimized jar
JVM
GraalVM™
native executable

EXTENSIONS

For each Java framework, a Quarkus extension

Makes it compatible with GraalVM native-images

And makes it much lighter to run on JVM

LIBRARIES YOU ALREADY KNOW

REST Easy

JAEGER

kafka

Infinispan

Unifies

IMPERATIVE and REACTIVE

```java
@Inject
SayService say;

@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return say.hello();
}
```

```java
@Inject @Stream("kafka")
Publisher<String> reactiveSay

@GET
@Produces(MediaType.SERVER_SE
public Publisher<String> stre
    return reactiveSay;
}
```

CONTAINER FIRST

CONTAINER FIRST

💾 Small size on disk

✓ Small container images

CONTAINER FIRST

Small size on disk	✓ Small container images
Fast boot time	✓ Instant scale up

CONTAINER FIRST

Small size on disk	✓ Small container images
Fast boot time	✓ Instant scale up
Low RSS[1] memory	✓ More containers with the same RAM

1) Resident Set Size

MEASURING MEMORY

RSS = all actual RAM consumed by the process
There's more than heap sizes!

```
$ ps -o pid,rss,command -p $(pgrep quarkus)
  PID   RSS COMMAND
11229 12628 ./target/quarkus-hello
```

```
java -XX:MaxRAM=50m -Xmx15m -Xss228k -jar app.jar
```

See also:
developers.redhat.com/blog/2017/04/04/openjdk-and-containers/

MEMORY (RSS)

Quarkus + GraalVM	Quarkus + OpenJDK	Best of traditio

MEMORY (RSS)

	Quarkus + GraalVM	Quarkus + OpenJDK	Best of traditio
REST	13 MB	74 MB	140 M

MEMORY (RSS)

	Quarkus + GraalVM	Quarkus + OpenJDK	Best of traditio
REST	13 MB	74 MB	140 M
REST+JPA	35 MB	130 MB	218 M

STARTUP TIME

Often frameworks use lazy initialization

"started" reported too early

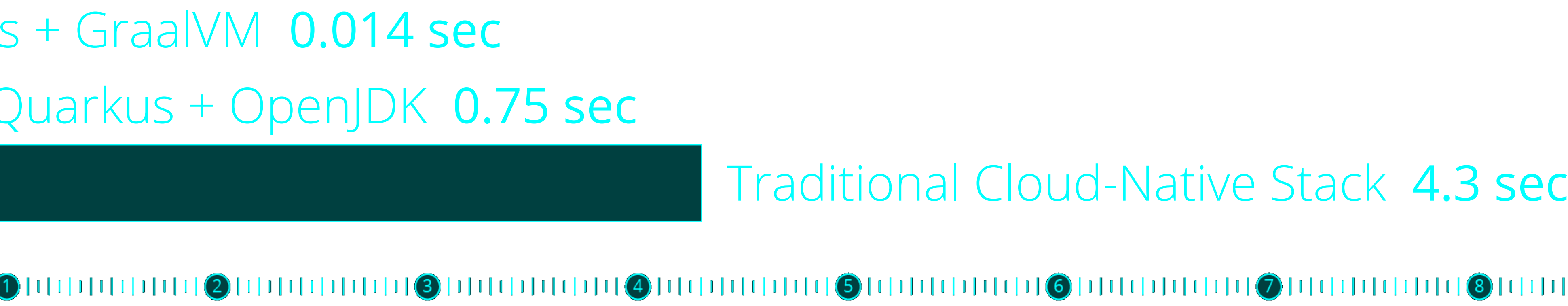
STARTUP TIME

Often frameworks use lazy initialization

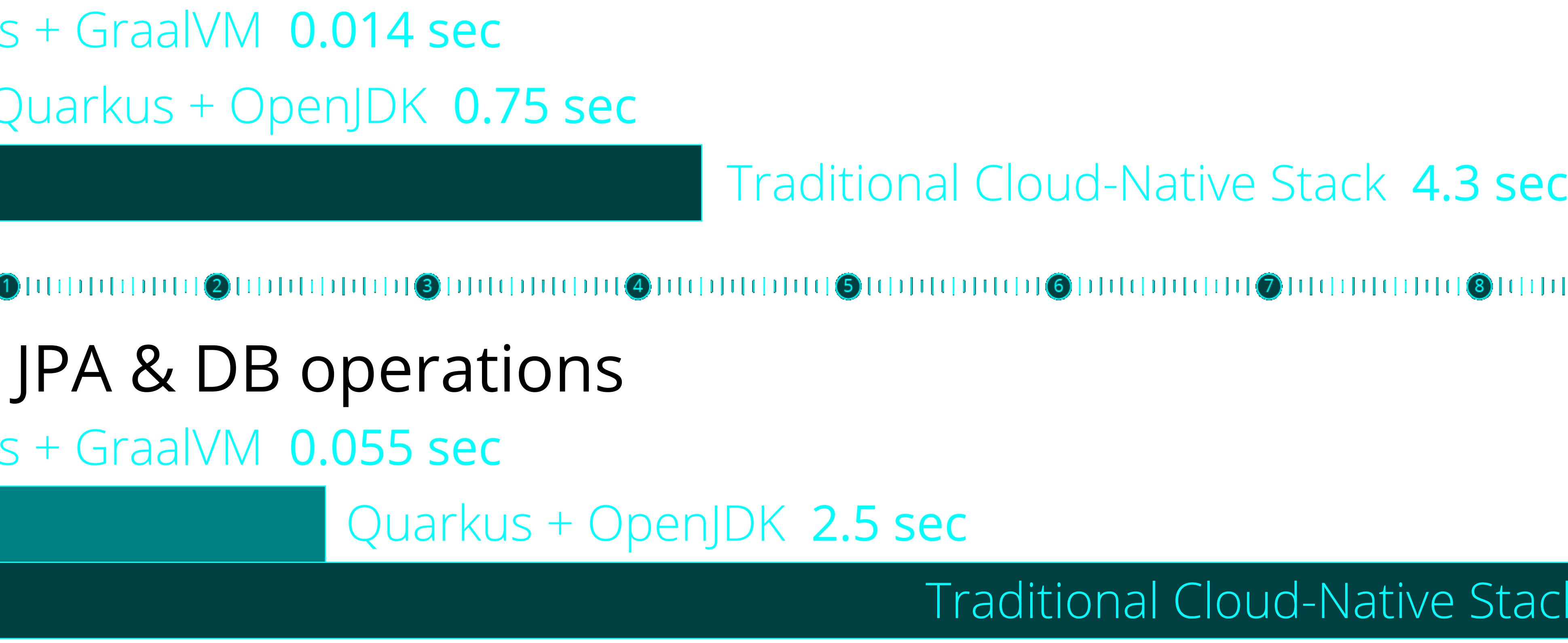
"started" reported too early

Measure time to first request

TIME TO FIRST REQUEST



TIME TO FIRST REQUEST

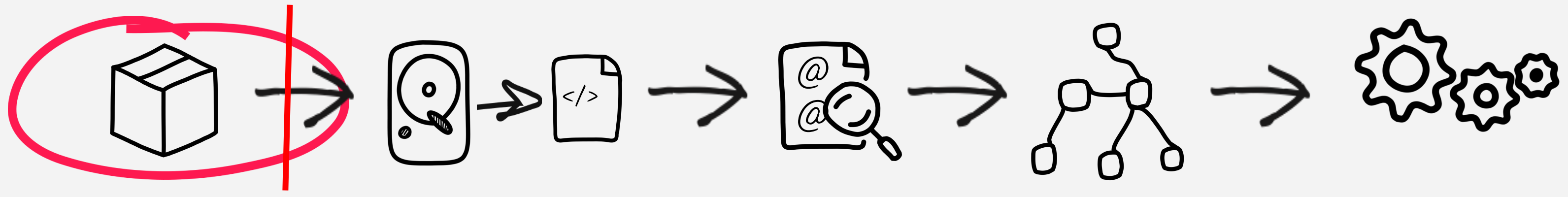


S H O W U S ?

Show me! REST / CRUD demo

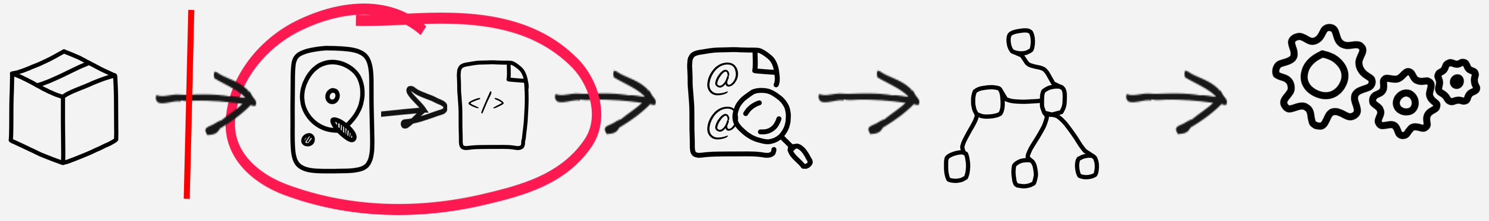
HOW IT WORKS

HOW A TRADITIONAL STACK WORKS



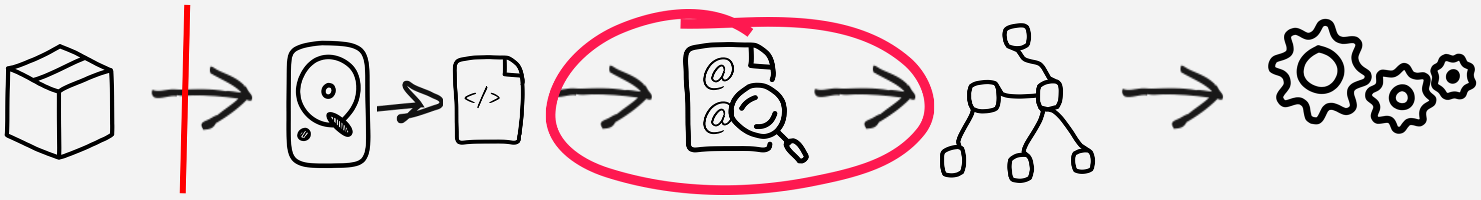
Built Java archive
/ deployment

HOW A TRADITIONAL STACK WORKS



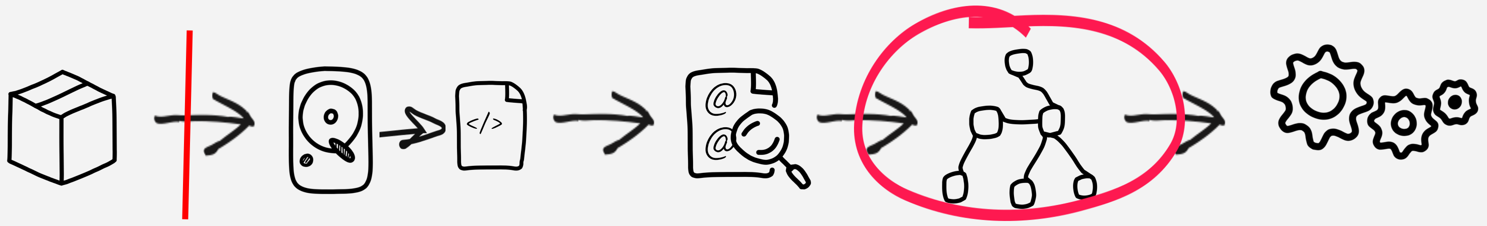
Search for configuration files,
Parse them

HOW A TRADITIONAL STACK WORKS



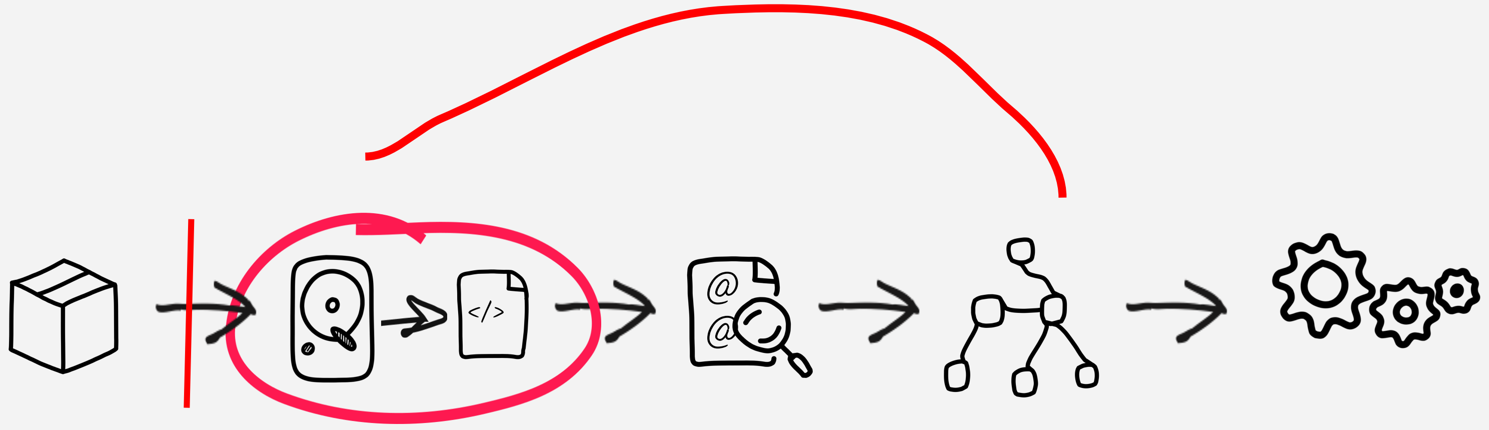
Classpath scanning to find
annotated classes.
Discover extension points,
plugins, optional features

HOW A TRADITIONAL STACK WORKS



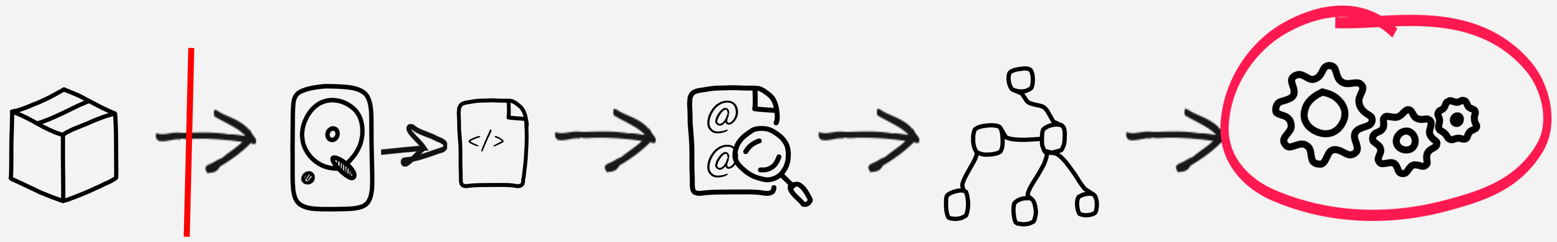
Build the metamodel,
Prepare injection points,
Generate proxies,
Enhance classes,
Validate the world

HOW A TRADITIONAL STACK WORKS



Search for configuration files,
Parse them

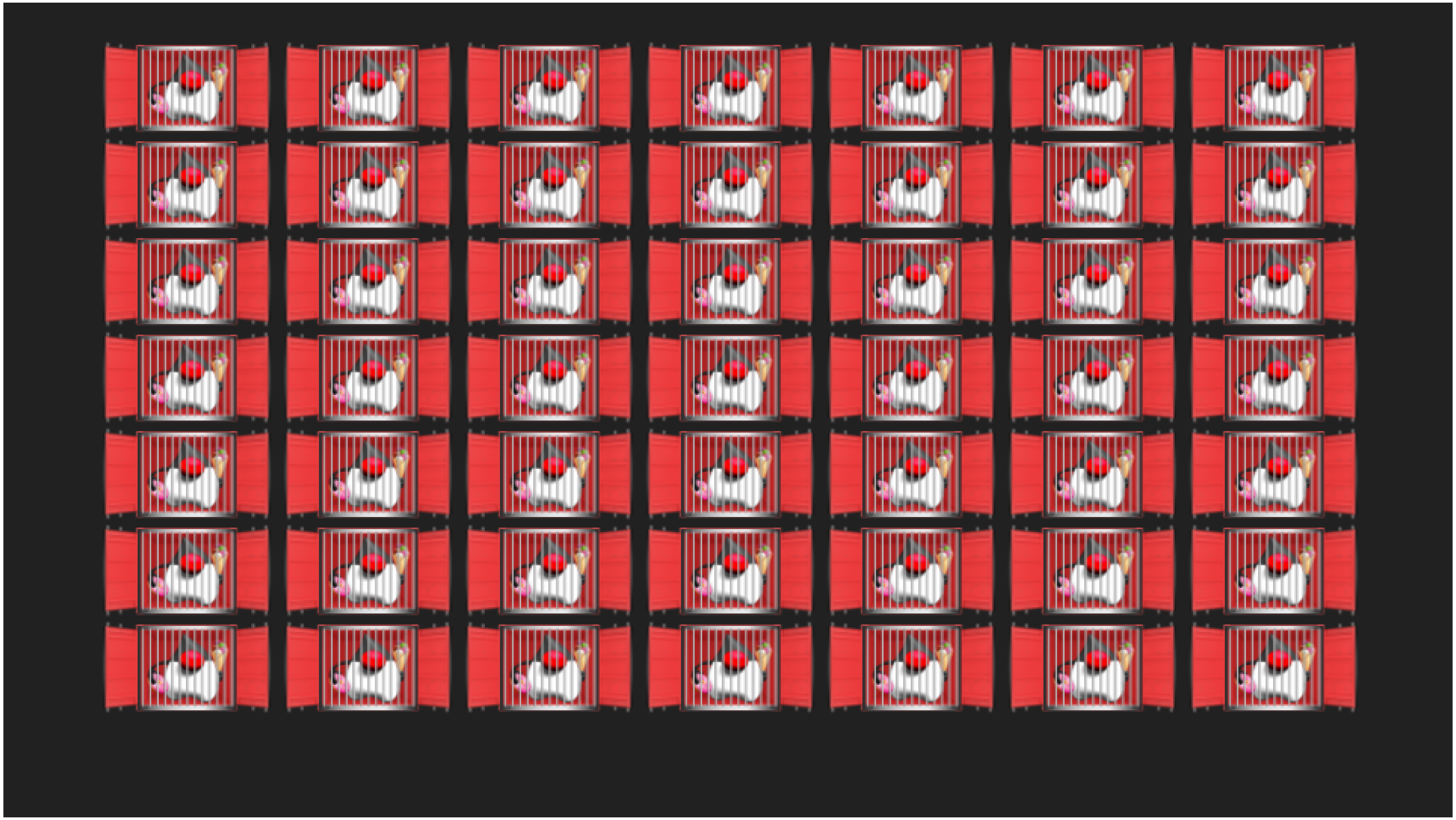
HOW A TRADITIONAL STACK WORKS



Ready to process!

THE OVERHEAD IS HIGH

PAY FOR IT *N* TIMES



WHILE IN QUARKUS:

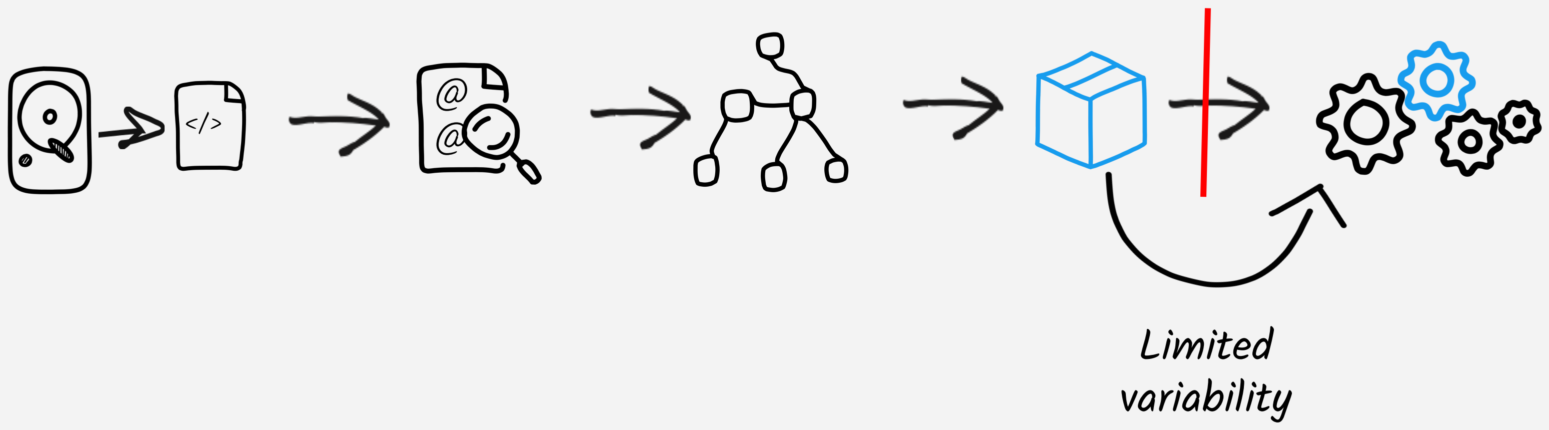
BUILD TIME BOOT

As much work as possible done at build time

Output: recorded wiring bytecode

Heap & state can be captured by the GraalVM native-image compiler

WHILE IN QUARKUS



EXTENSIONS MODEL

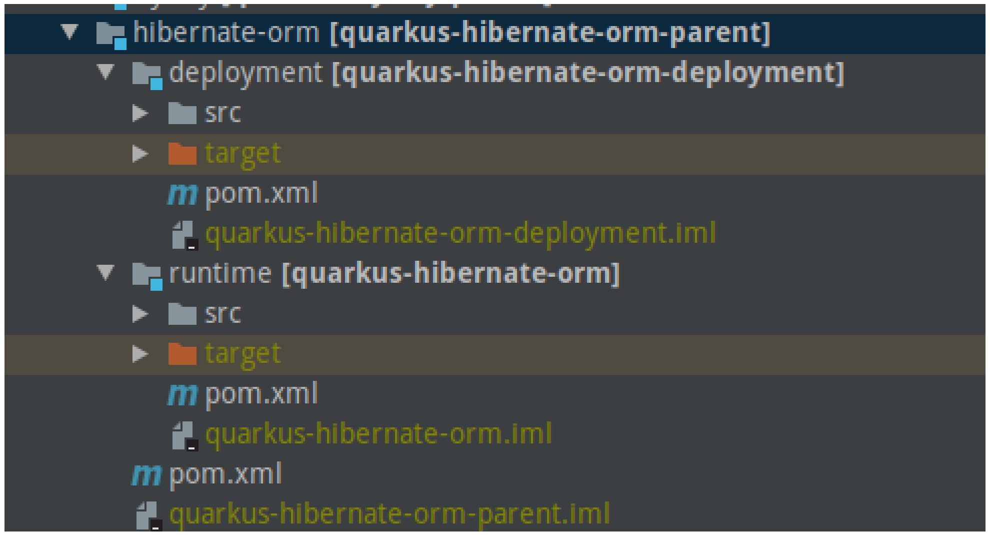
Each framework/library needs an extension to apply these benefits

Can physically avoid shipping some bootstrap-preparation only code

Is Quarkus a meta-build tool?

EXTENSIONS MODEL

Can physically avoid shipping some code



```
▼ hibernate-orm [quarkus-hibernate-orm-parent]
  ▼ deployment [quarkus-hibernate-orm-deployment]
    ▶ src
    ▶ target
      pom.xml
      quarkus-hibernate-orm-deployment.iml
  ▼ runtime [quarkus-hibernate-orm]
    ▶ src
    ▶ target
      pom.xml
      quarkus-hibernate-orm.iml
    pom.xml
    quarkus-hibernate-orm-parent.iml
```

JANDEX

High performance classpath scanner & indexer:
avoids any class initialization

A R C

CDI based dependency injection, at build time

G I Z M O

Bytecode generation library, used by extensions to generate all infrastructure

DESIGN CONSEQUENCES

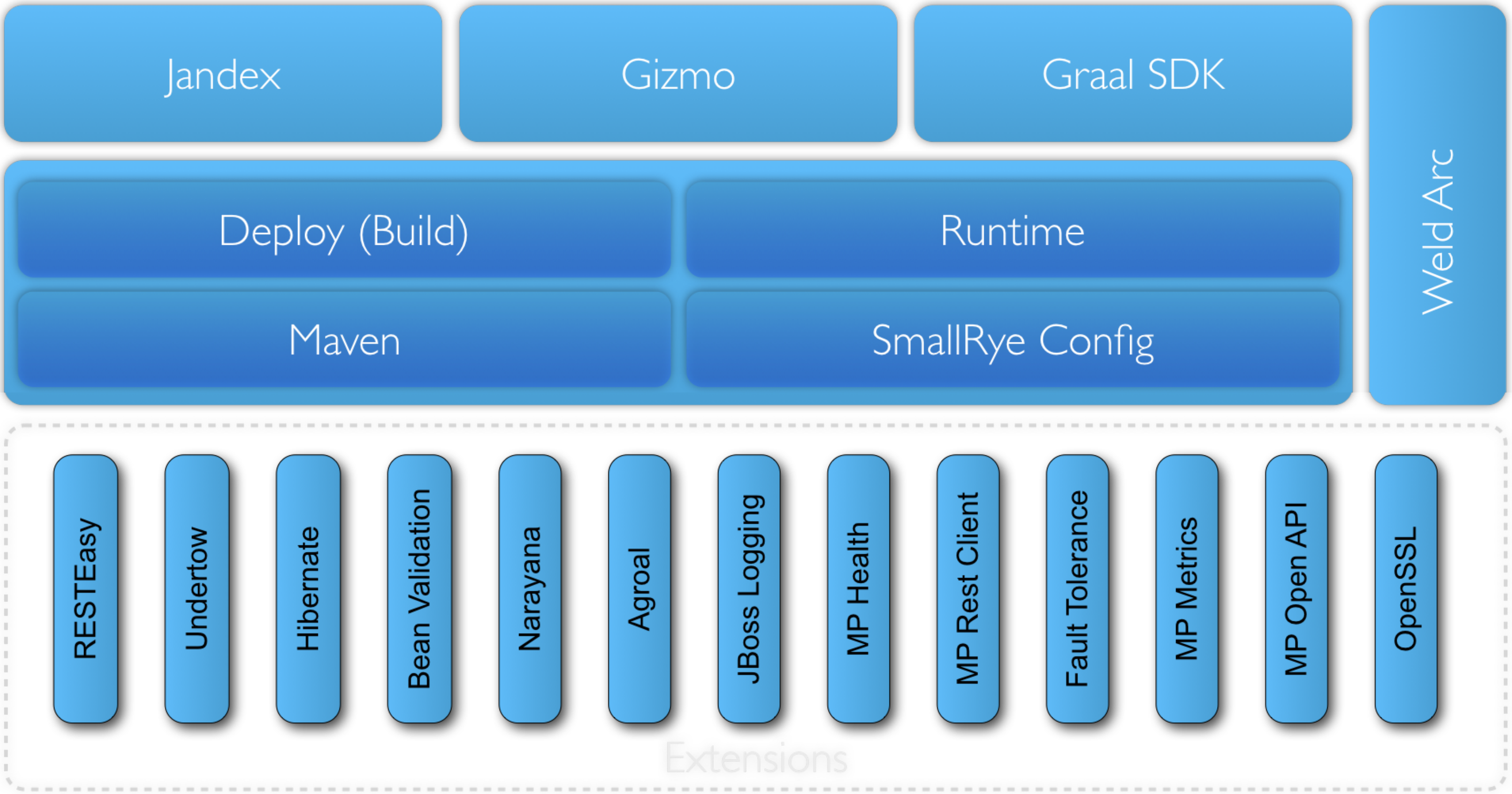
Less classes are loaded

Can physically avoid shipping some bootstrap-preparation only code

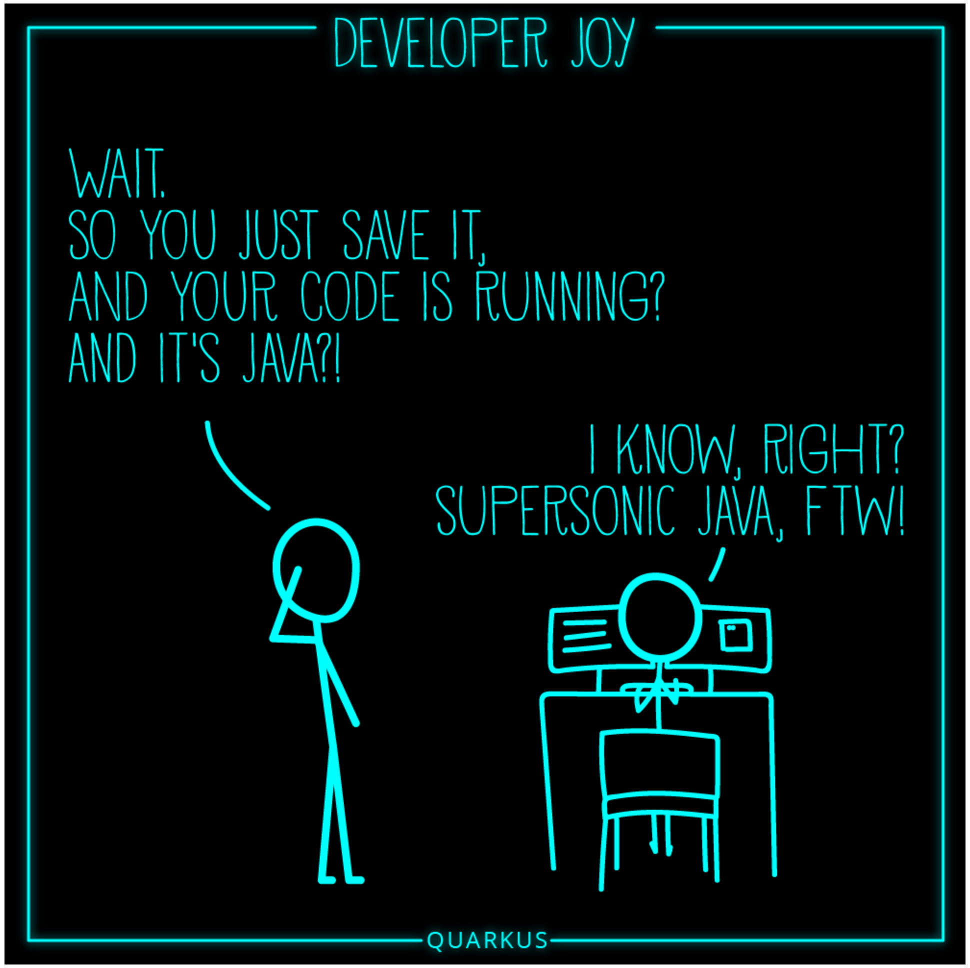
Overhead not repeated on each container boot

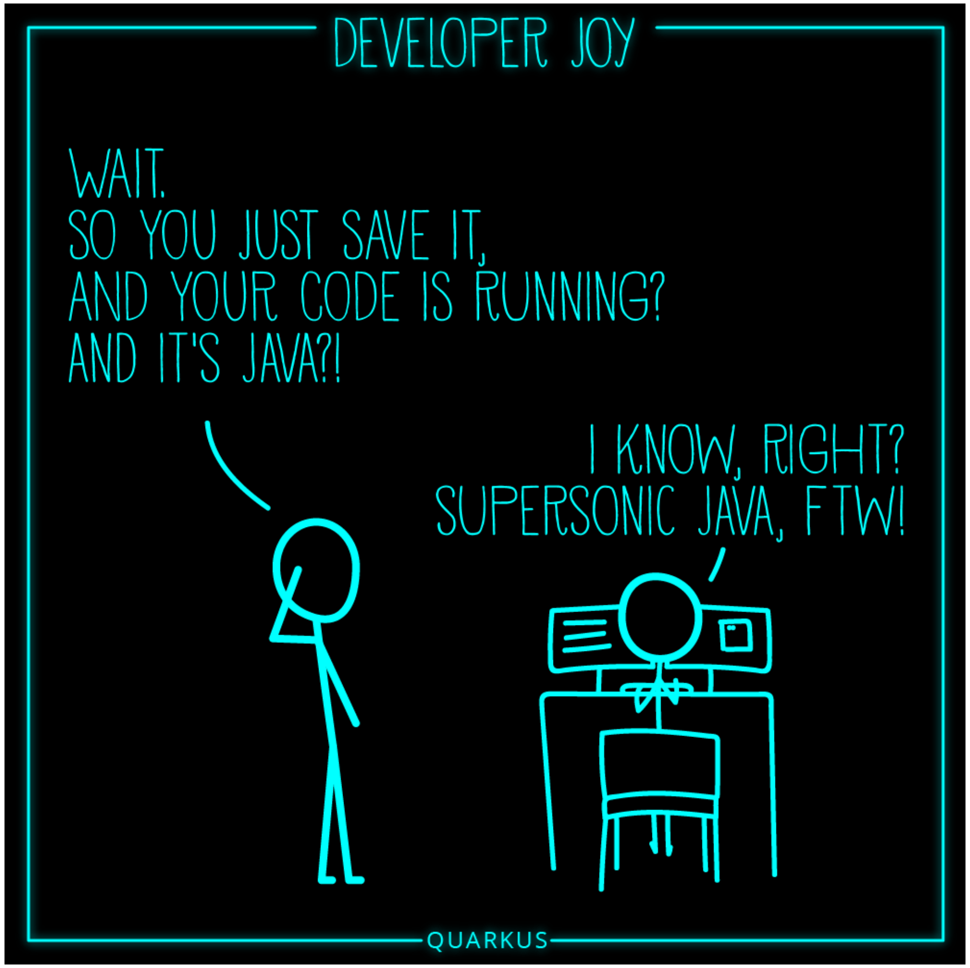
Far easier to get working in GraalVM native images - and better optimised code!

Core + Extensions



DEVELOPER'S JOY?


DEVELOPER JOY
WAIT. SO YOU JUST SAVE IT, AND YOUR CODE IS RUNNING? AND IT'S JAVA?!
I KNOW, RIGHT? SUPERSONIC JAVA, FTW!
QUARKUS


DEVELOPER JOY
WAIT. SO YOU JUST SAVE IT, AND YOUR CODE IS RUNNING? AND IT'S JAVA?!
I KNOW, RIGHT? SUPERSONIC JAVA, FTW!
QUARKUS

Show me! Demo #2

QUARKUS EXTENSIONS

Required for frameworks that hit GraalVM limitations
Opportunity to highly optimise also for JVM
Code strictly separates build time analysis and runtime: extremely lean output!

WHAT CAN AN EXTENSION DO?

Invoke Quarkus helpers to dynamically

Interact with the GraalVM compiler needs
Generate "Bootstrap at build" initializers
Much much more... and evolving

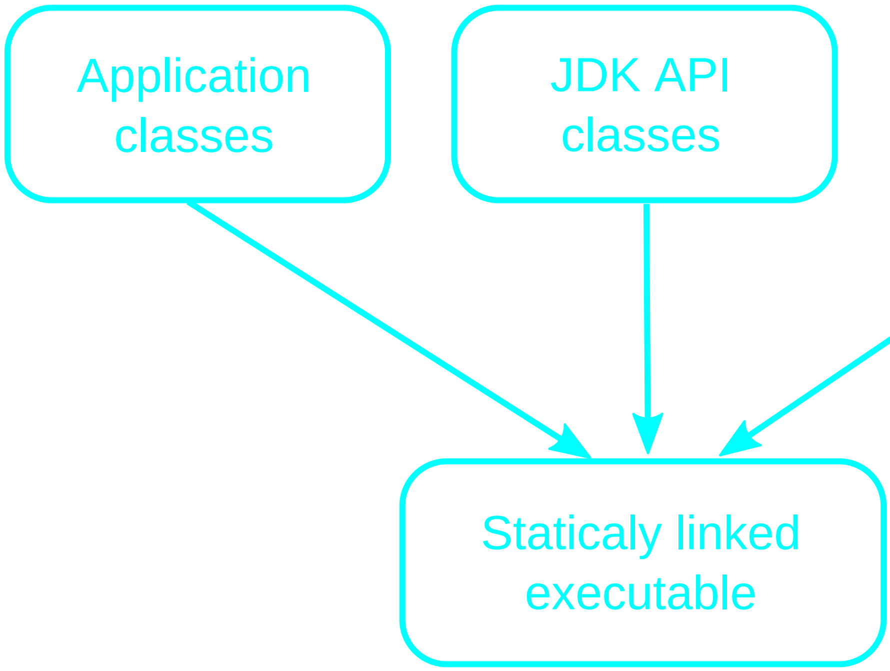
SO, WHERE'S THE CATCH?

NO PERFORMANCE COMPROMISES

&

GraalVM™

AoT compilation with GraalVM



AoT compilation with GraalVM

Static analysis
Closed world assumption
Aggressive dead code elimination

GraalVM™

LIMITATIONS

OF GRAALVM NATIVE IMAGES

GraalVM™

DYNAMIC CLASSLOADING

X unsupported

GraalVM

DYNAMIC CLASSLOADING

X unsupported

Deloying jars, wars, etc. at runtime impossible

GraalVM™

JVMTI, JMX

+ other native VM interfaces

X unsupported

No agents

JRebel, Byteman, profilers, tracers, ...

No Java Debugger

GraalVM™

REFLECTION

⚠ limited!

Requires registration via `native-image` CLI/API

GraalVM™

MORE...

limited!

Need to register in advance also:

- Dynamic proxies
- Resources being loaded
- JNI, Unsafe Memory Access, ...

GraalVM™ Very special

STATIC INIT

Attempts to *run* them at build time

Resolve classes, run "safe" static initializers
Take a snapshot of the produced instances - prune the unreachable ones
Include needed state in the executable

GraalVM™ Very special

STATIC INIT

not allowed: file handles, sockets, threads

careful with other state: timestamps, system dependent constants, capturing environment variables, etc..

HOW DO YOU DISABLE A FEATURE ANYWAY?

HOW DO YOU DISABLE A FEATURE ANYWAY?

```java
boolean jmxEnabled = parseConfiguration(...);

if (jmxEnabled) {
    registerJMX();
}
```

HOW DO YOU DISABLE A FEATURE ANYWAY?

```
boolean jmxEnabled = parseConfiguration(...);

if (jmxEnabled) {
    registerJMX();
}
```

```
static final JMX_ENABLED = false;

if (JMX_ENABLED) {
    registerJMX();
}
```

THINK TWICE BEFORE STARTING...

THINK TWICE BEFORE STARTING...

All your dependencies need to get compiled too!

THINK TWICE BEFORE STARTING...

All your dependencies need to get compiled too!

ALL REACHABLE CODE

THINK TWICE BEFORE STARTING...

All your dependencies need to get compiled too!

ALL REACHABLE CODE

ALL DEPENDENCIES

THINK TWICE BEFORE STARTING...

Might be wiser to contribute to an open community of per-dependency extensions?

All Quarkus code is Apache License v.2

QUARKUS WRAP UP

- ✓ Good old Java
- ✓ More fun, less weight
- ✓ Can go small as Go, works great on JVM too

Java suited for clouds and containers!

NATIVE IMAGE PERFORMANCE

Slightly lower than JVM

Yet a winner in some conditions:

NATIVE IMAGE PERFORMANCE

Slightly lower than JVM

Yet a winner in some conditions:

high memory density

NATIVE IMAGE PERFORMANCE

Slightly lower than JVM

Yet a winner in some conditions:

high memory density

no warmup needed!

NATIVE IMAGE PERFORMANCE

Slightly lower than JVM

Yet a winner in some conditions:

high memory density

no warmup needed!

instant elastic response / lambda support

NATIVE IMAGE PERFORMANCE

Slightly lower than JVM

Yet a winner in some conditions:

high memory density

no warmup needed!

instant elastic response / lambda support

Bonus: you don't have to make a choice upfront.

THANK YOU!

Q & A

- ✓ Docs & guides: quarkus.io
- ✓ Chat: quarkusio.zulipchat.com
- ✓ Quickstarts: github.com/quarkusio/quarkus-quick
- ✓ Stack Overflow tag: quarkus
- ✓ Twitter: @quarkusio