QUARKUS

Supersonic. Subatomic. Java.

Stuart Douglas, Red Hat
@stuartwdouglas

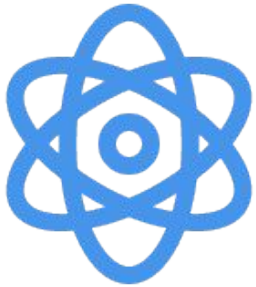
About Me

Stuart Douglas

- Senior Principal Engineer at Red Hat
- Quarkus Co-founder
- Servlet Specification Co-lead
- Founder of the Undertow Project
- Core Contributor to WildFly

An Open Source stack to write Java apps

Cloud Native,



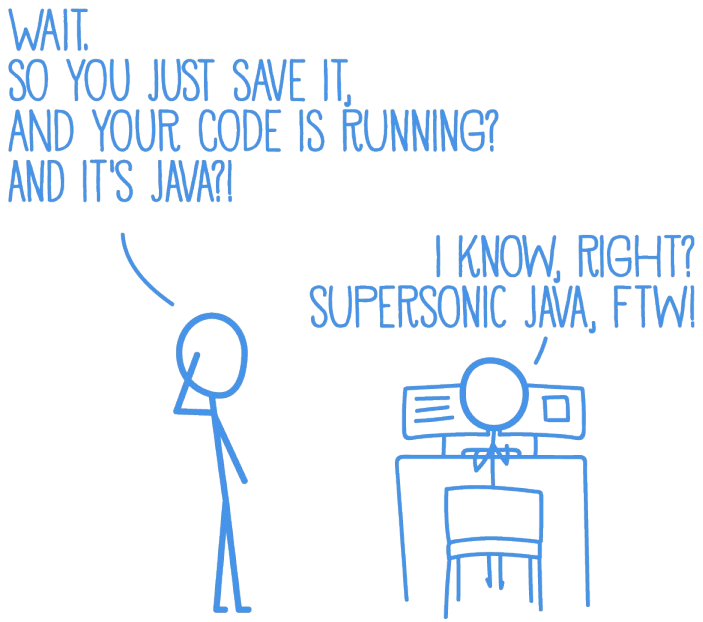
Microservices,

Serverless

Benefit No. 1: Developer Joy

A cohesive platform for optimized developer joy:

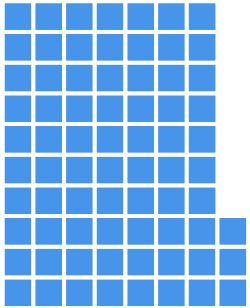
- Zero config, live reload in the blink of an eye
- Based on standards, but not limited
- Unified configuration
- Streamlined code for the 80% common usages, flexible for the 20%
- No hassle native executable generation



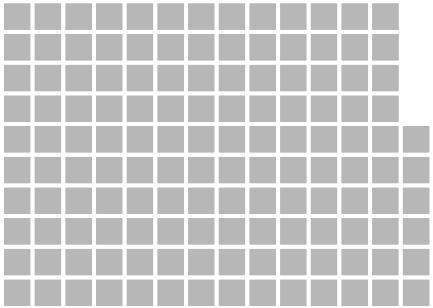
Benefit No. 2: Supersonic Subatomic Java

REST*

Quarkus + Native
(via GraalVM)
12 MB



Quarkus + JVM
(via OpenJDK)
73 MB

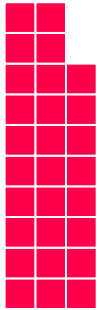


Traditional
Cloud-Native Stack
136 MB

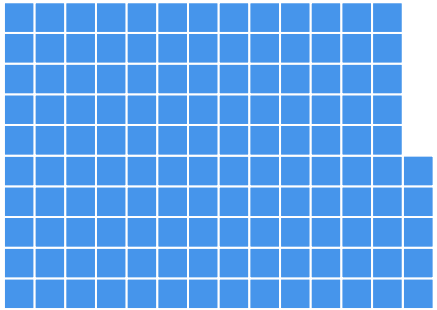
*Memory (RSS) in Megabytes, tested on a single-core machine

Benefit No. 2: Supersonic Subatomic Java

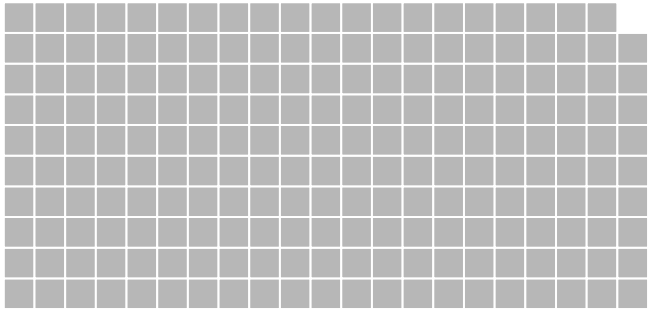
REST + CRUD*



Quarkus + Native
(via GraalVM)
28 MB



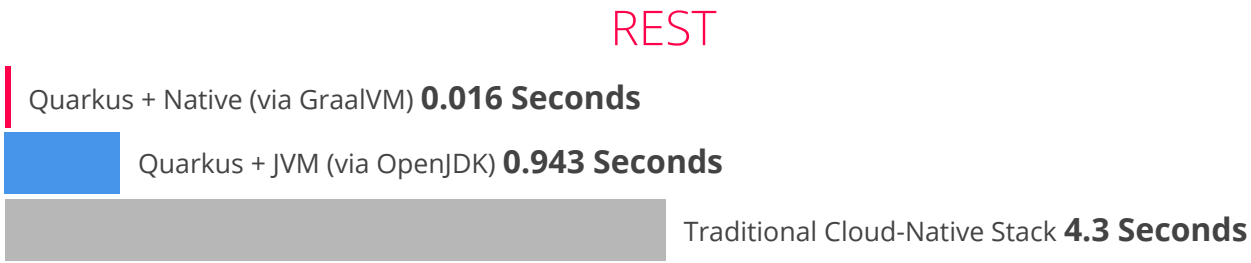
Quarkus + JVM
(via OpenJDK)
145 MB

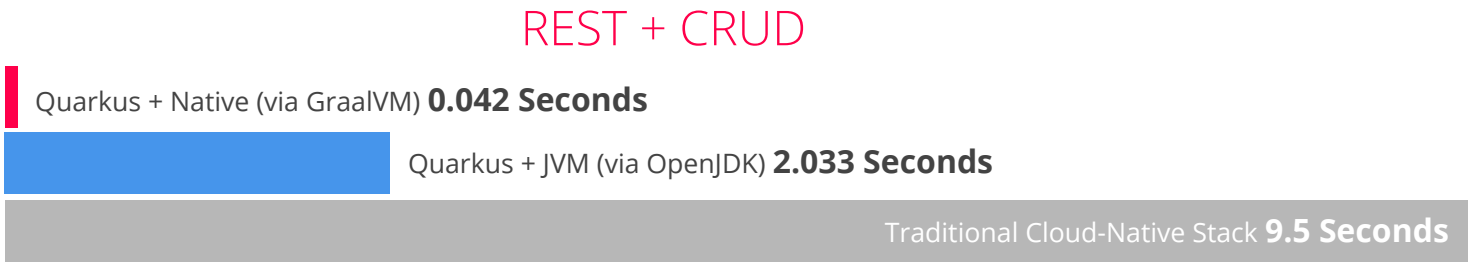


Traditional
Cloud-Native Stack
209 MB

*Memory (RSS) in Megabytes, tested on a single-core machine

Benefit No. 2: Supersonic Subatomic Java





Time to first response

Benefit No. 3: Unifies Imperative and Reactive

```
@Inject
SayService say;

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

```
@Inject @Channel("kafka")
Publisher<String> reactiveSay;

@GET
@Produces(MediaType.SERVER_SENT_EVENTS)
public Publisher<String> stream() {
      return reactiveSay;
}
```

- Combine both Reactive and imperative development in the same application
- Use the technology that fits your use-case
- Key for reactive systems based on event driven apps

Benefit No. 4: Best of Breed Frameworks & Standards

Quarkus provides a cohesive, fun to use, full-stack framework by leveraging a growing list of over fifty best-of-breed libraries that you love and use. All wired on a standard backbone.

Quarkus Benefits

Developer Joy

Supersonic Subatomic Java

Unifies

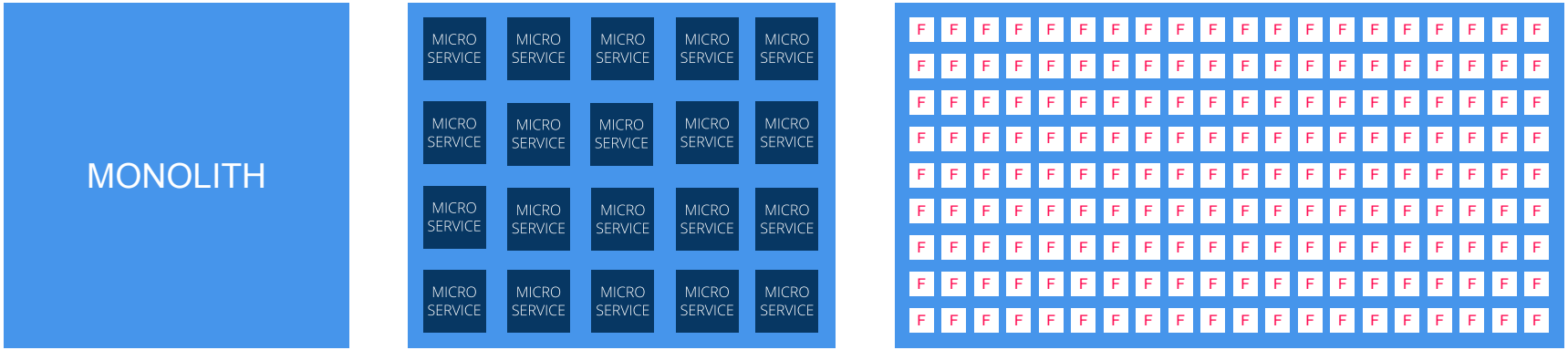
imperative and reactive

Best of breed

libraries and standards

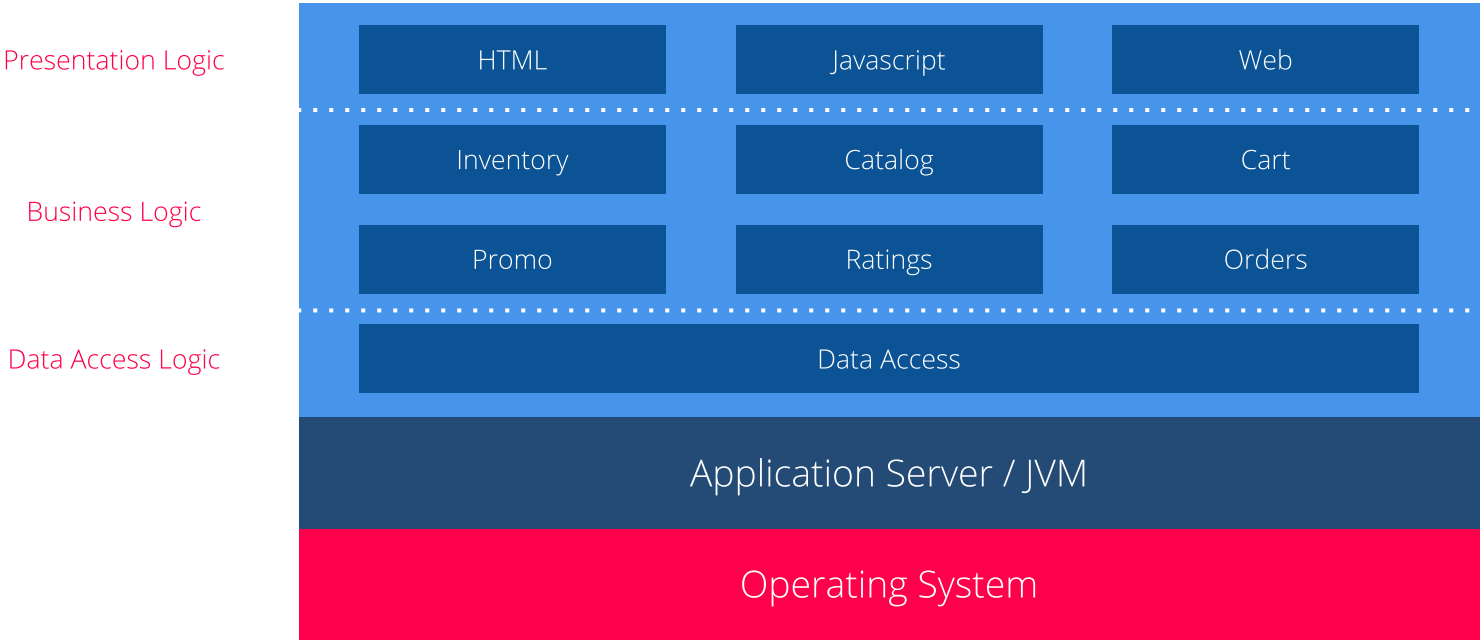
DEMO

From monolith to...

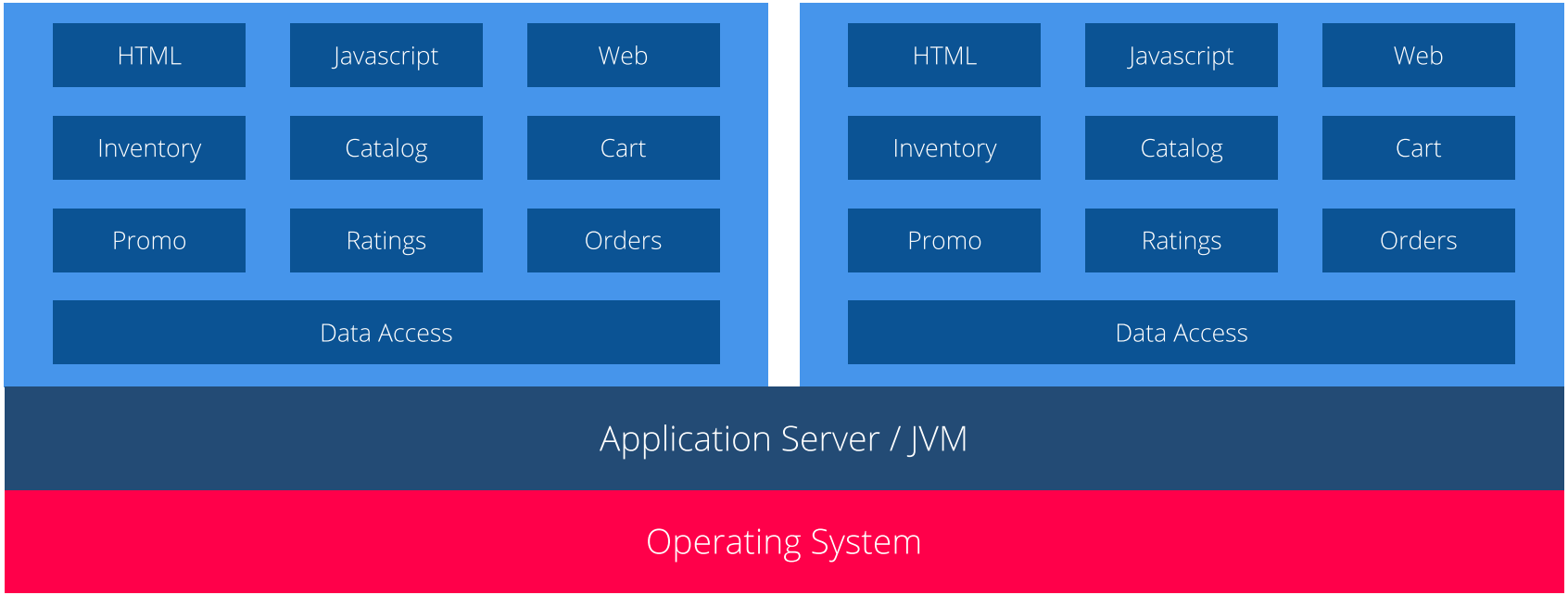


- 1 monolith ≈ 20 microservices ≈ 200 functions
- Scale to 1 vs scale to 0
- Communication pattern

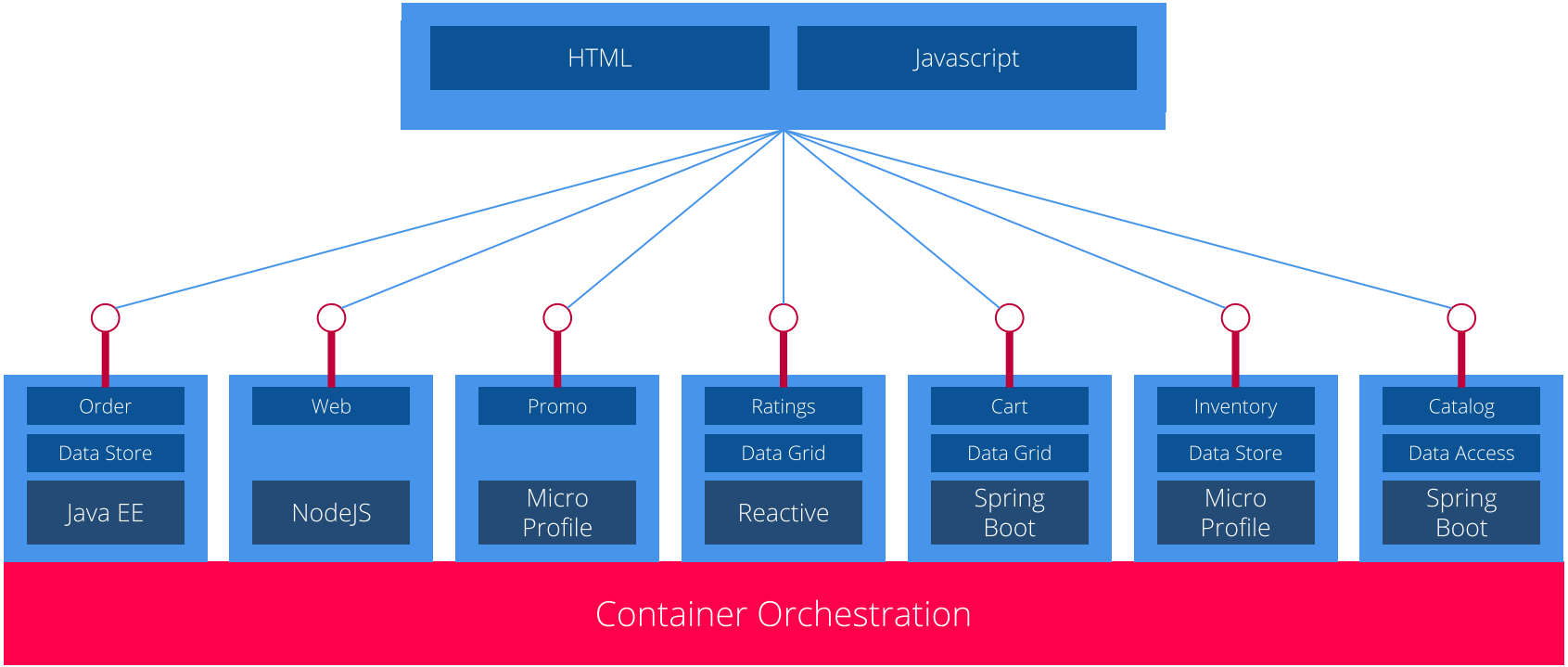
Enterprise Java was Designed for 3-Tier Architecture



How Application Services Deployed Java Apps



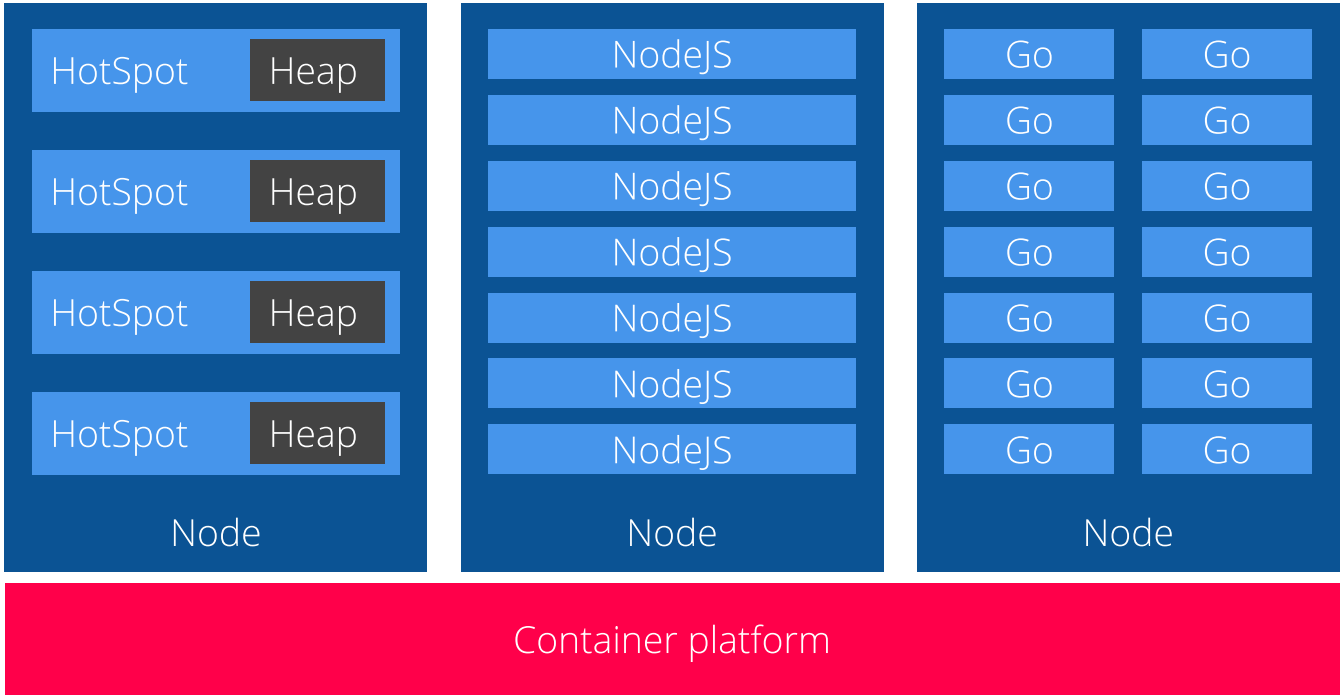
Microservices Changed How We Deploy Apps



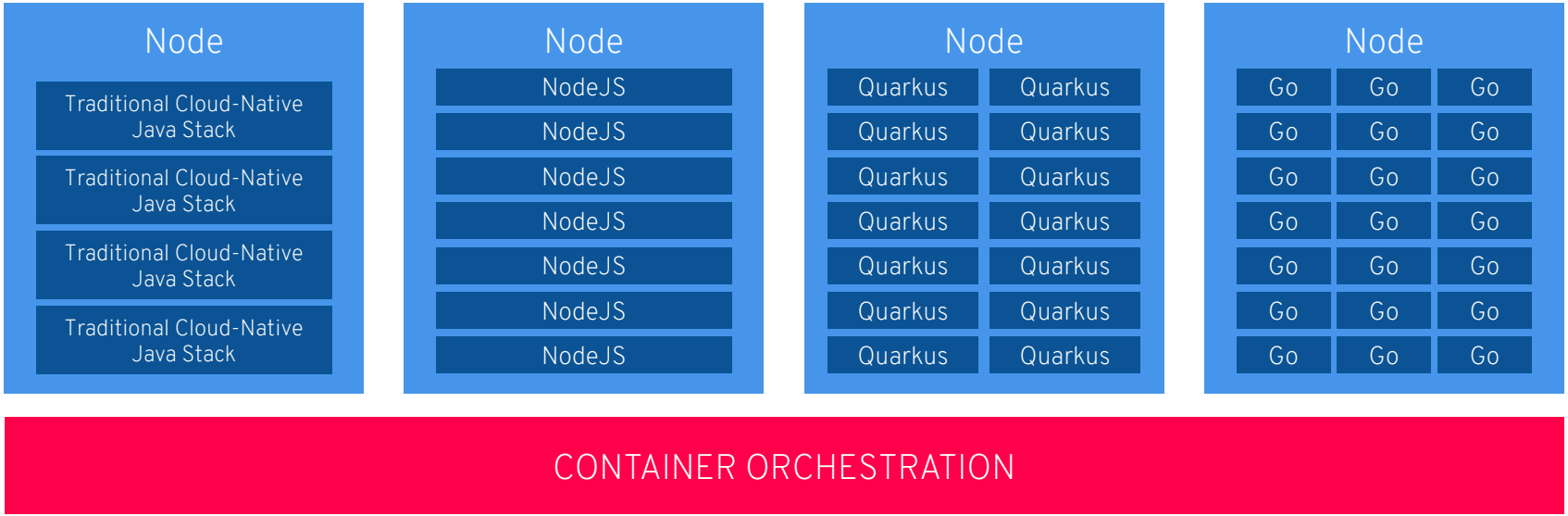
The hidden truth about Java + containers

- Startup overhead
  - # of classes, bytecode, JIT
- Memory overhead
  - # of classes, metadata, compilation

The hidden truth about Java + containers



The hidden truth about Java + containers



https://developers.redhat.com/blog/2017/03/14/java-inside-docker/

Java on cloud native: a Red Hat journey

Standards

- Java EE streamlining
- Eclipse MicroProfile

Runtimes

- WildFly on OpenShift

Hardware architectures

- Raspberry Pi and Plug Computer
- Android

Ahead of time compilation

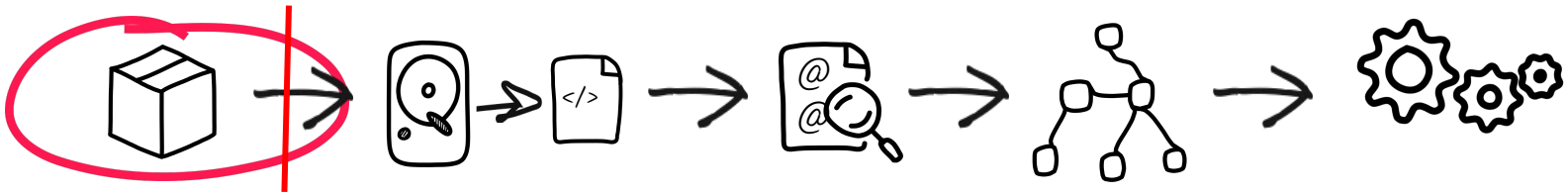
- Lead gcj
- Looked at Dalvik, Avian, Excelsior JET

OpenJDK

- Container ergonomics
- JVM metadata reduction

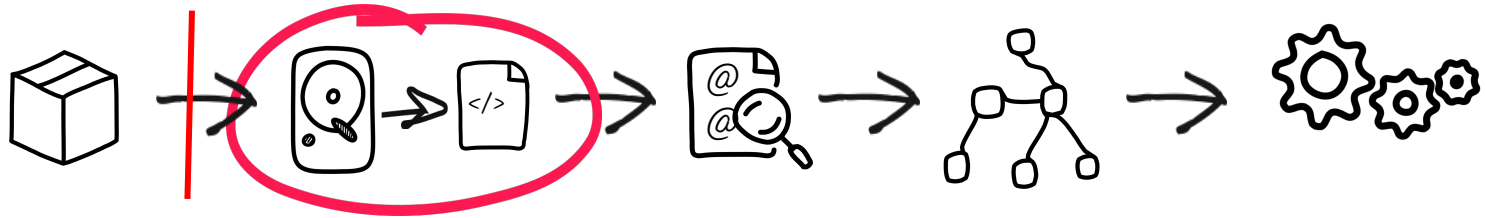
HOW QUARKUS WORKS

How does a *framework* start?



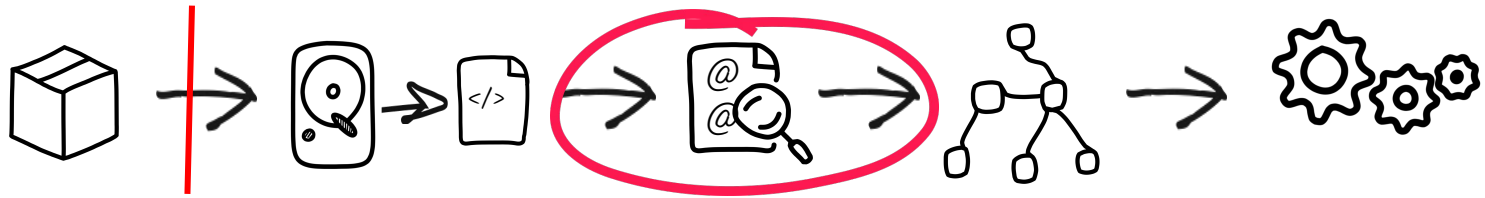
Build time
(maven, gradle...)

How does a *framework* start?



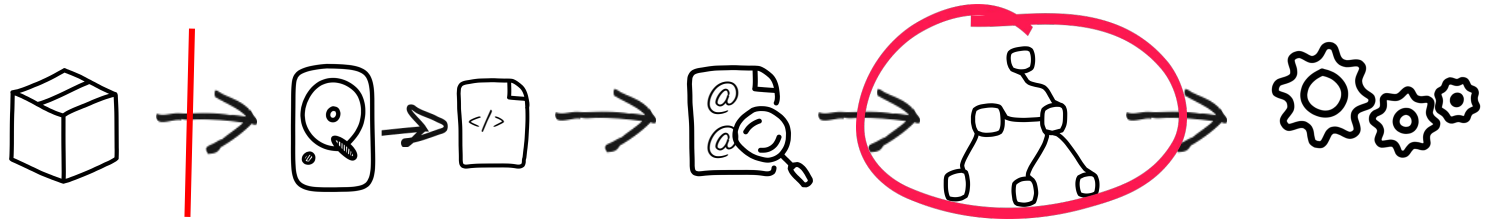
Load config file from file system
Parse it

How does a *framework* start?

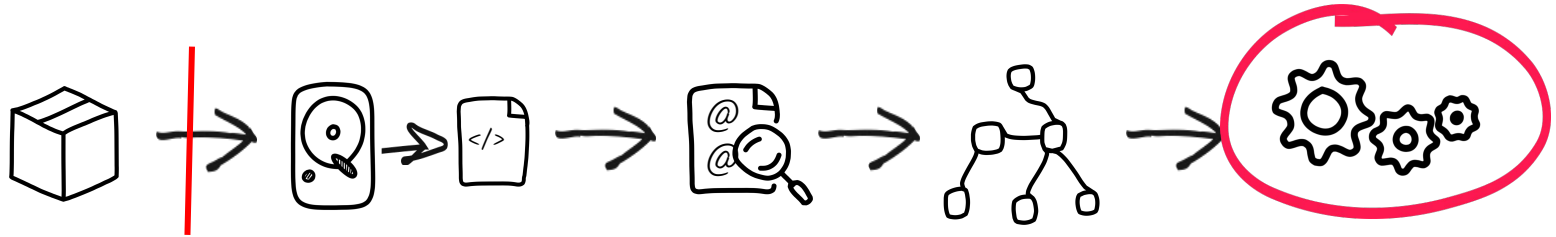


Classpath scanning to find annotated classes
Attempt to load class to enable/disable features

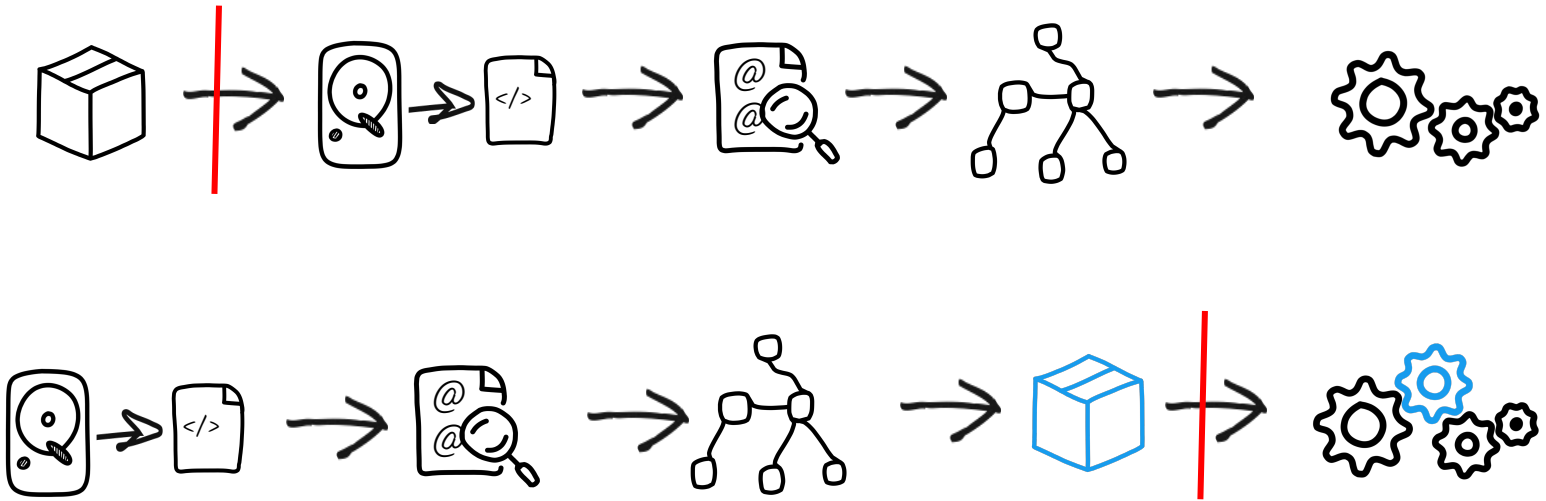
How does a *framework* start?



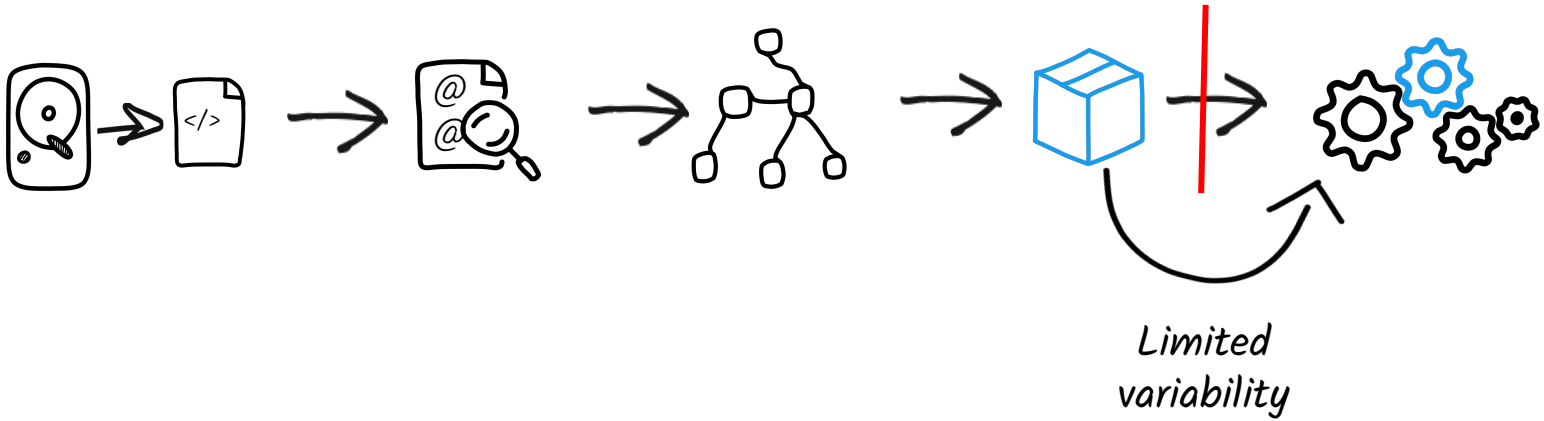
How does a *framework* start?



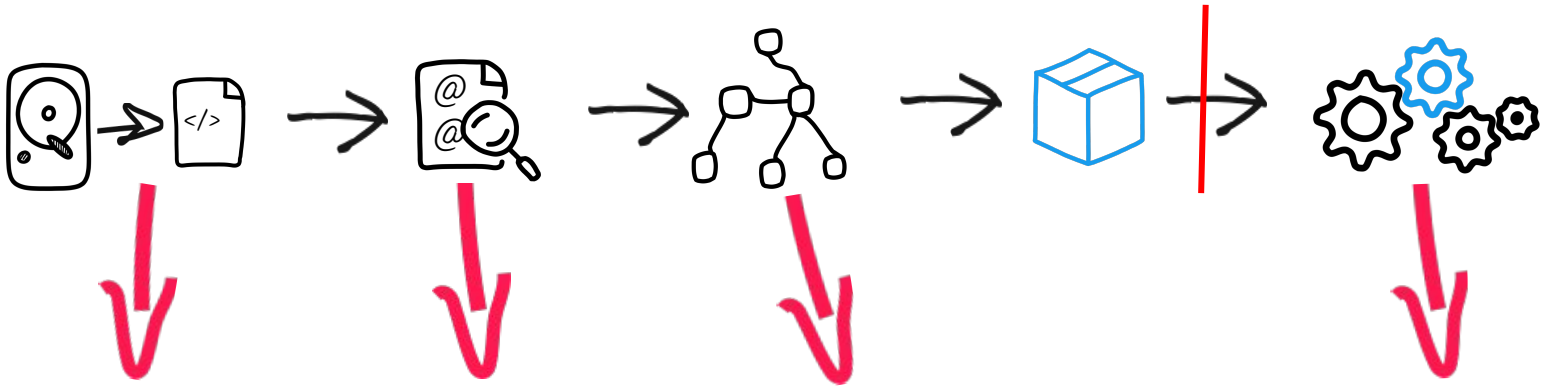
The Quarkus way



The Quarkus way - Closed-World Assumption



The Quarkus way - Benefits

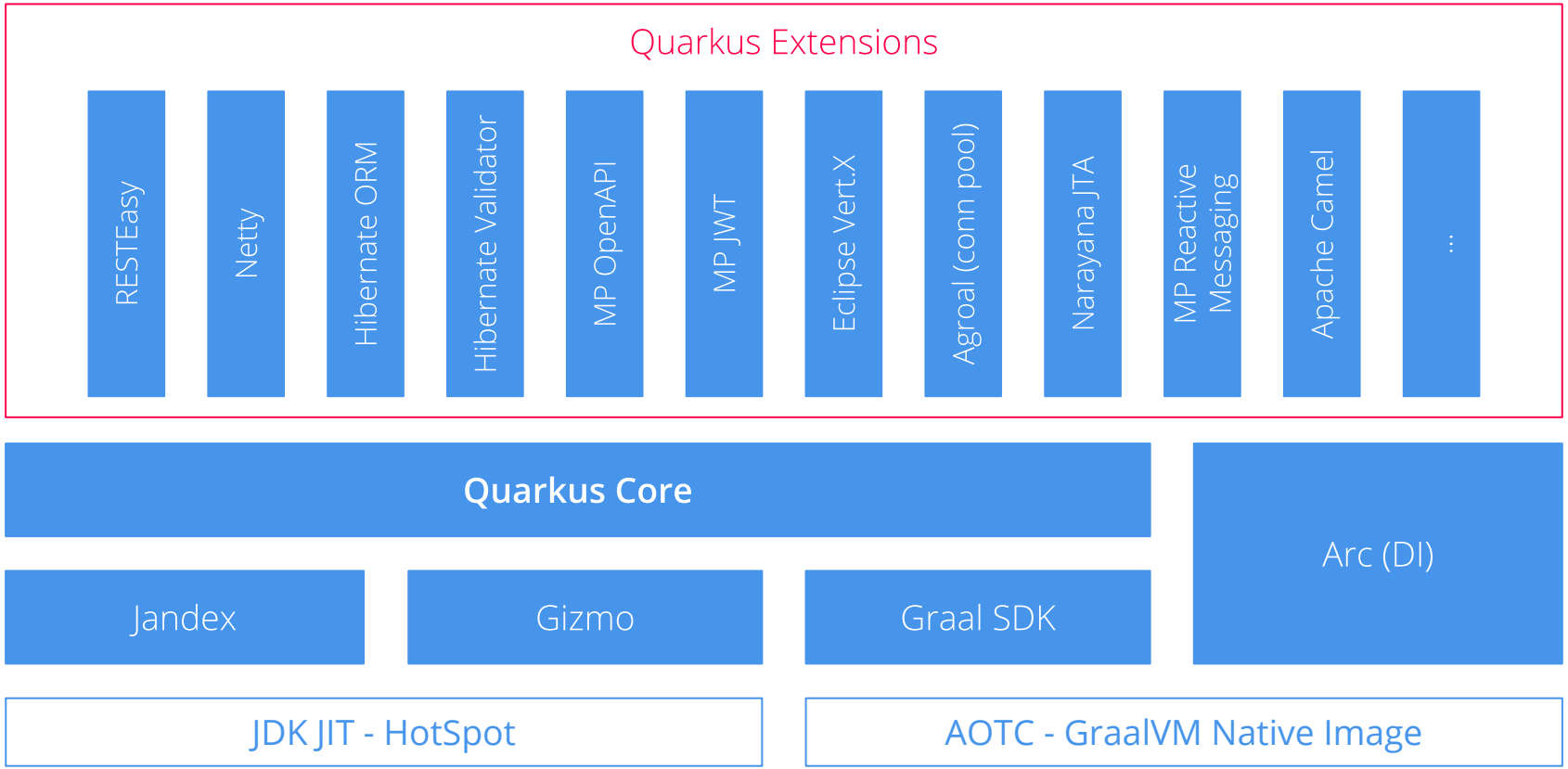


No FS access at runtime
No need for a parser at runtime

No scanning at runtime
Less reflection and classloading at runtime

Optimized configuration
Less reflection (proxy generation)

Prepared initialization (ready to serve)


Quarkus Extensions
RESTEasy
Netty
Hibernate ORM
Hibernate Validator
MP OpenAPI
MP JWT
Eclipse Vert.X
Agroal (conn pool)
Narayana JTA
MP Reactive Messaging
Apache Camel
...
Quarkus Core
Arc (DI)
Jandex
Gizmo
Graal SDK
JDK JIT - HotSpot
AOTC - GraalVM Native Image

Conclusion

Full stack

including JPA, Transaction Manager etc

Both in live reload and native executable

Massive productivity wins

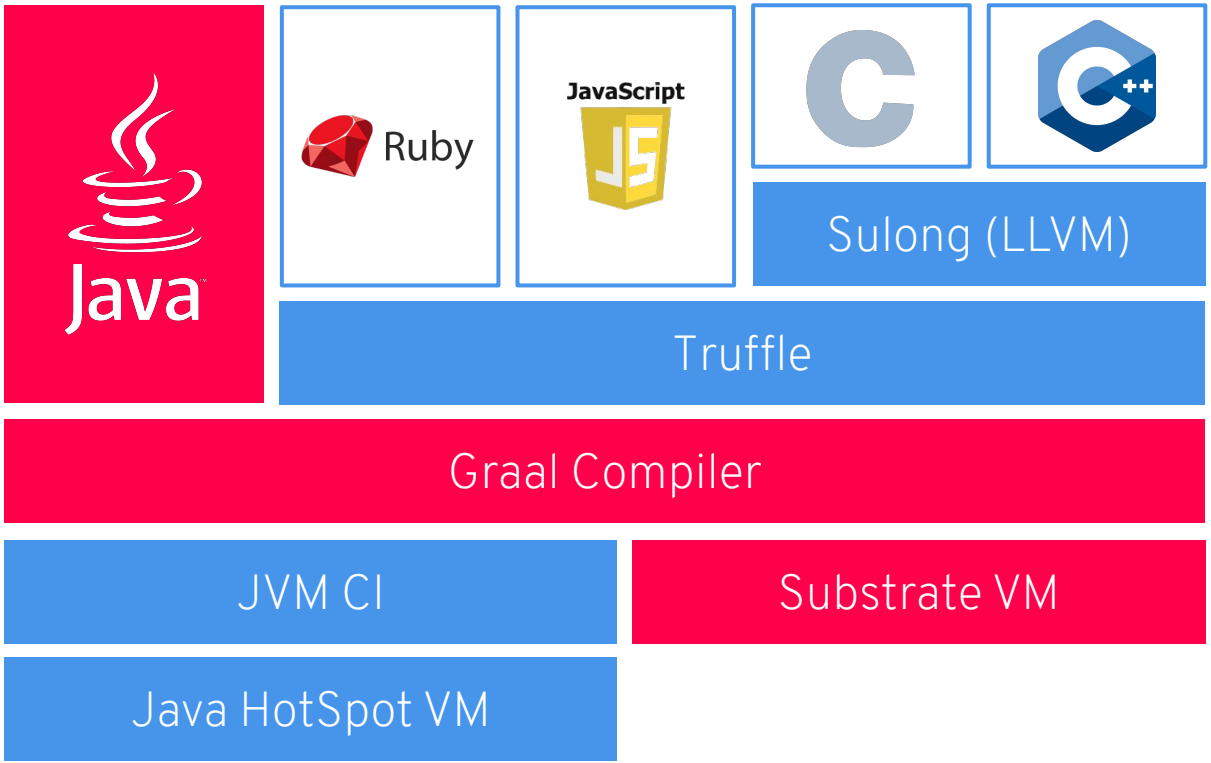
Simplified configuration

Microscopic code->test loop

80% common usages made easy

GraalVM

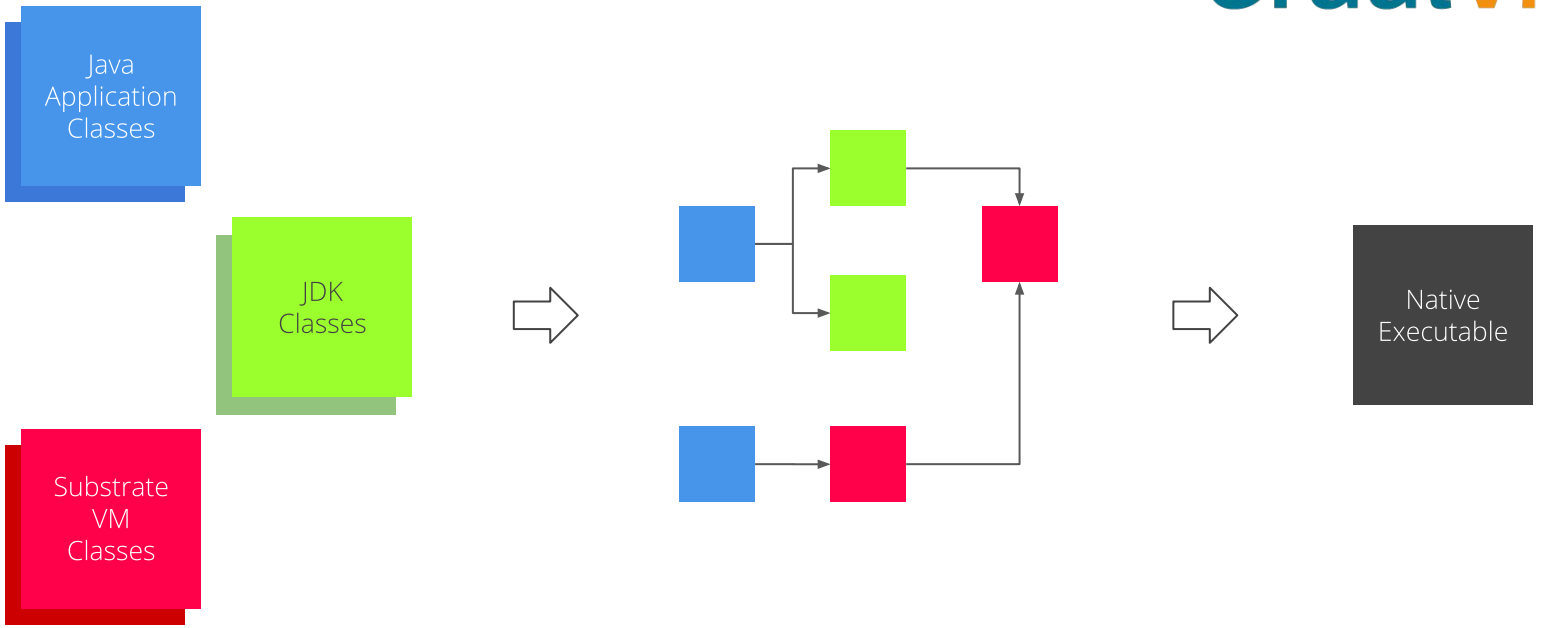
Polyglot, Native of JVM, Embeddable


Java
Ruby
JavaScript
C
C++
Sulong (LLVM)
Truffle
Graal Compiler
JVM CI
Substrate VM
Java HotSpot VM

AOTC - GraalVM native image - Dead code elimination

Closed-world assumption

GraalVM™



The Dark Side

Not supported

- Dynamic classloading
- InvokeDynamic & Method handles
- Finalizer
- Security manager
- JVMTI, JMX, native VM Interfaces

OK with caveats in usage

- Reflection (manual list)
- Dynamic proxy (manual list)
- JNI (manual list)
- Static initializers (eager)
- Lambda, Threads (OK, pfff!)
- References (similar)

The Dark Side

The interesting parts

Not supported

- Dynamic classloading
- InvokeDynamic & Method handles
- Finalizer
- Security manager
- JVMTI, JMX, native VM Interfaces

OK with caveats in usage

- Reflection (manual list)
- Dynamic proxy (manual list)
- JNI (manual list)
- Static initializers (eager)
- Lambda, Threads (OK, pfff!)
- References (similar)

When to use which VM with Quarkus

JIT - OpenJDK HotSpot

High memory density requirements
High request/s/MB
Fast startup time

Best raw performance (CPU)
Best garbage collectors
Higher heap size usage

Known monitoring tools
Compile Once, Run anywhere
Libraries that only works in standard JDK

AOT - GraalVM native image

Highest memory density requirements
Highest request/s/MB
    for low heap size usages
Faster startup time
    10s of ms for Serverless

GraalVM specific benefits

100% of the ecosystem supported on GraalVM

Drives the gathering of metadata needed by GraalVM

- based on framework knowledge
- Classes using reflection, resources, etc
- No need for agent + prerun, long JSON metadata or manual command lines

Minimize dependencies

Help dead code elimination

Can I add my dependencies?

YES

Add your own dependency

- Works on the JVM (OpenJDK)
- May work on AOT (GraalVM)

Write your own extension

- Like add your dependency plus…
- Build time startup and memory improvements
- Better dead code elimination
- Developer Joy

Thank you.

QUARKUS

https://quarkus.io

https://quarkusio.zulipchat.com

@quarkusio