



# QUARKUS

---

## Container First

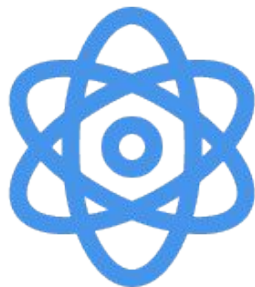
Emmanuel Bernard  
Distinguished Engineer  
Red Hat  
[@emmanuelbernard](https://twitter.com/emmanuelbernard)



# An Open Source stack to write Java apps



Cloud Native,



Microservices,



Serverless



# DEMO



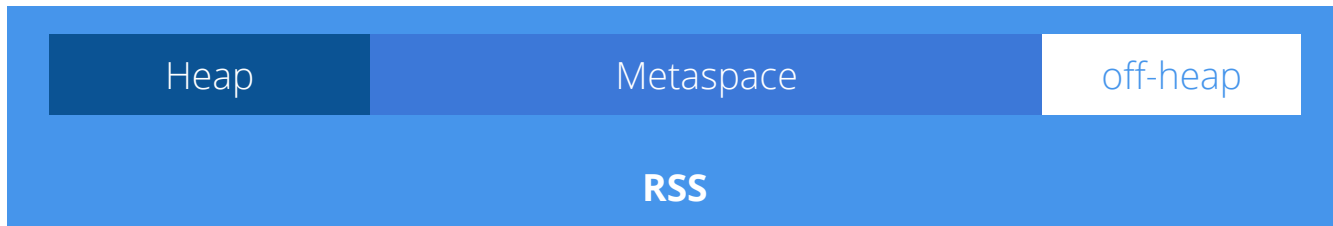
# WHY QUARKUS?

**QUARK:** elementary particle / **US:** hardest thing in computer science

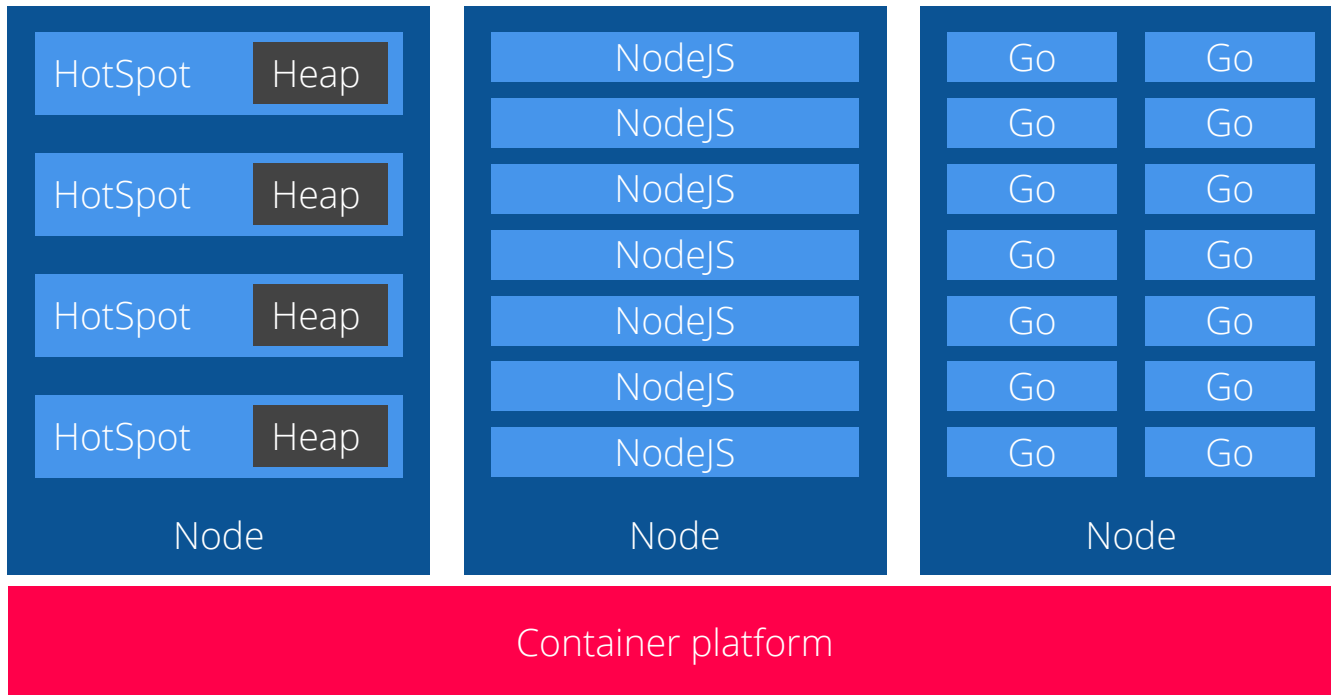


# The hidden truth about Java + containers

- Designed for requests/s
- Startup overhead
  - # of classes, bytecode, JIT
- Memory overhead
  - # of classes, metadata, compilation



# The hidden truth about Java + containers



# WHAT IS QUARKUS?

**QUARK:** elementary particle / **US:** hardest thing in computer science

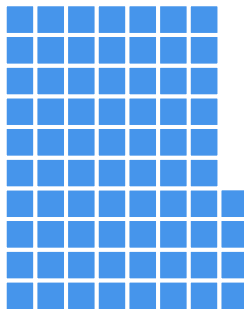


## Benefit No. 2: Supersonic Subatomic Java

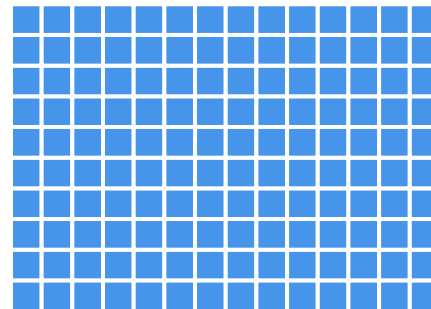
REST



Quarkus + AOT  
**19 MB**



Quarkus + OpenJDK (JIT)  
**77 MB**



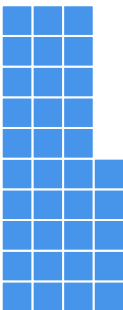
Traditional Cloud-Native Stack  
**140 MB**



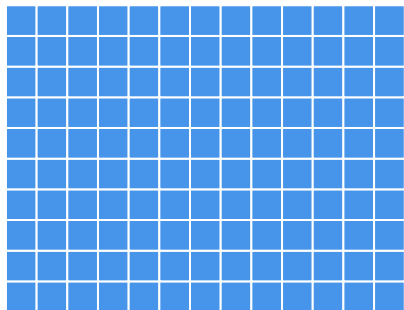


## Benefit No. 2: Supersonic Subatomic Java

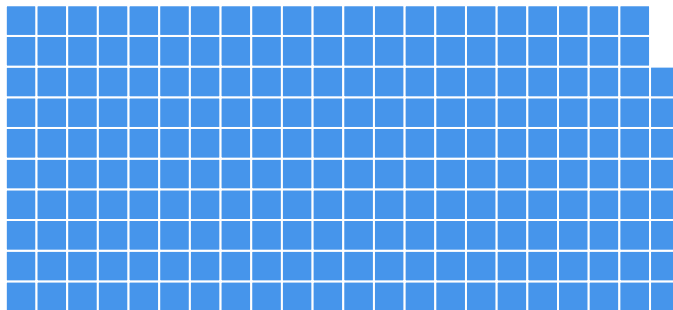
REST + CRUD



Quarkus + AOT  
**35 MB**



Quarkus + OpenJDK (JIT)  
**130 MB**

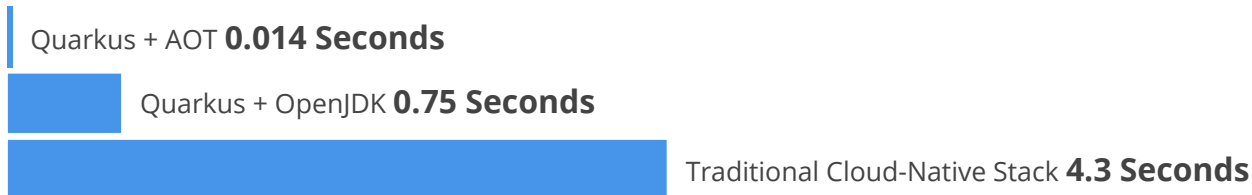


Traditional Cloud-Native Stack  
**218 MB**

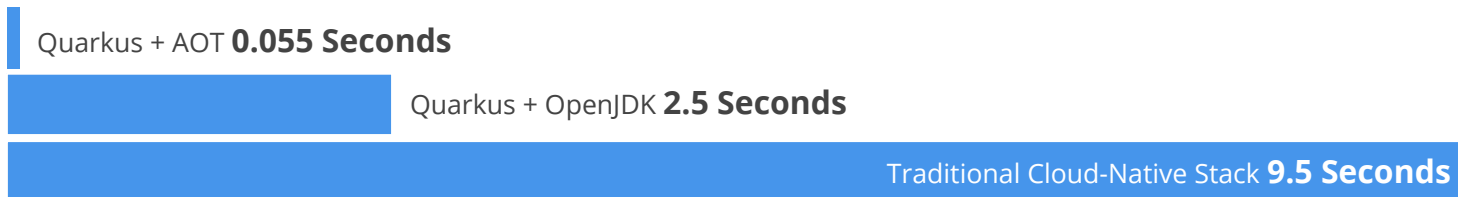


# Benefit No. 2: Supersonic Subatomic Java

REST



REST + CRUD



Time to first response





The Enabler

# GraalVM

You keep using that word.  
I do not think it means what you think it means.

# HOW QUARKUS WORKS



# Move startup time to build time

## What does a framework do at startup time

- Parse config files
- Classpath & classes scanning
  - for annotations, getters or other metadata
- Build framework metamodel objects
- Prepare reflection and build proxies
- *Start and open IO, threads etc*



# Build time benefits

Do the work once, not at each start

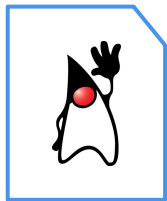
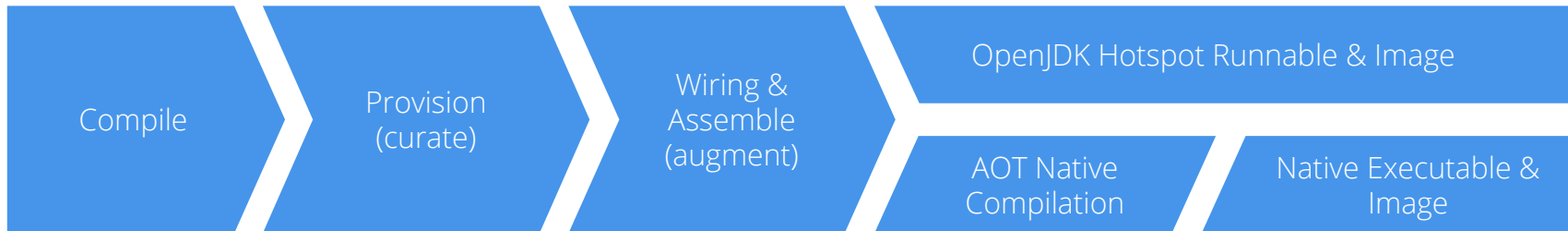
All the bootstrap classes are no longer loaded

Less time to start, less memory used

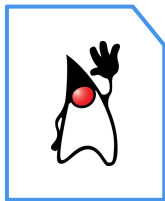
Less or no reflection nor dynamic proxy



# An ahead-of-time, build-time, runtime



app.jar



frameworks



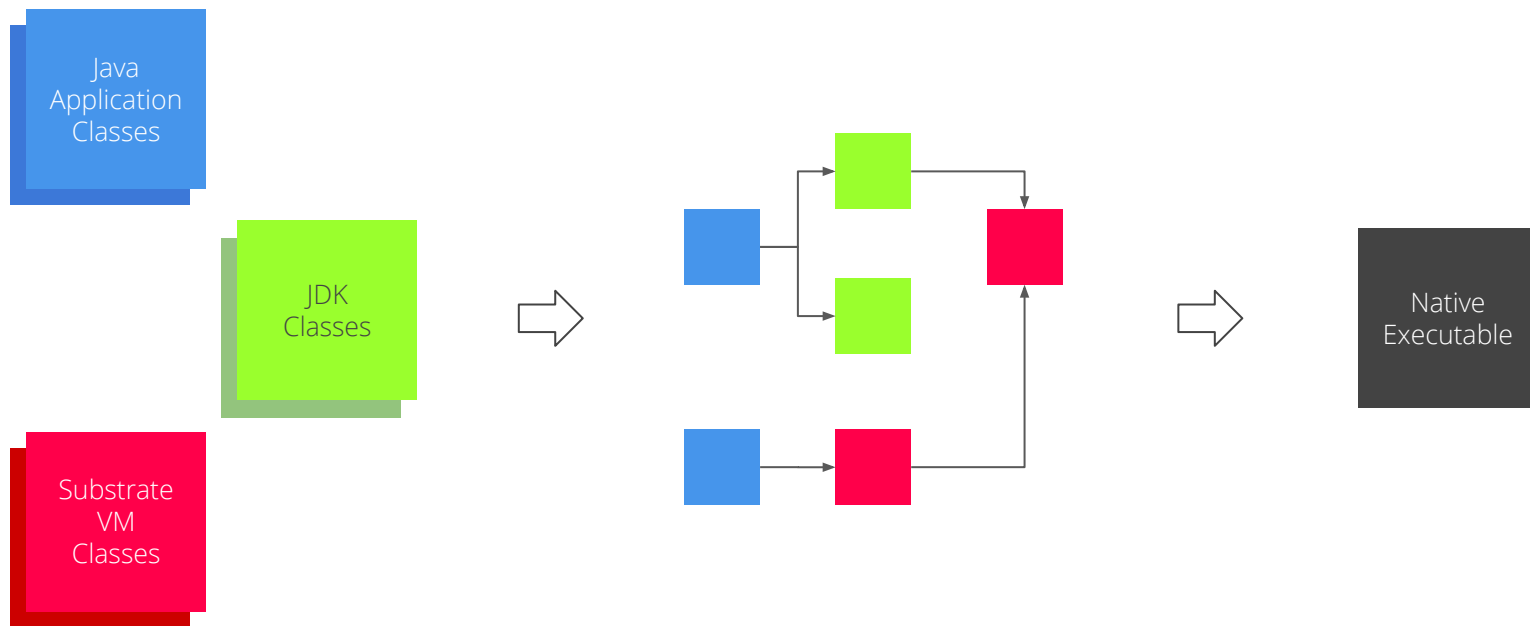
Runnable java app



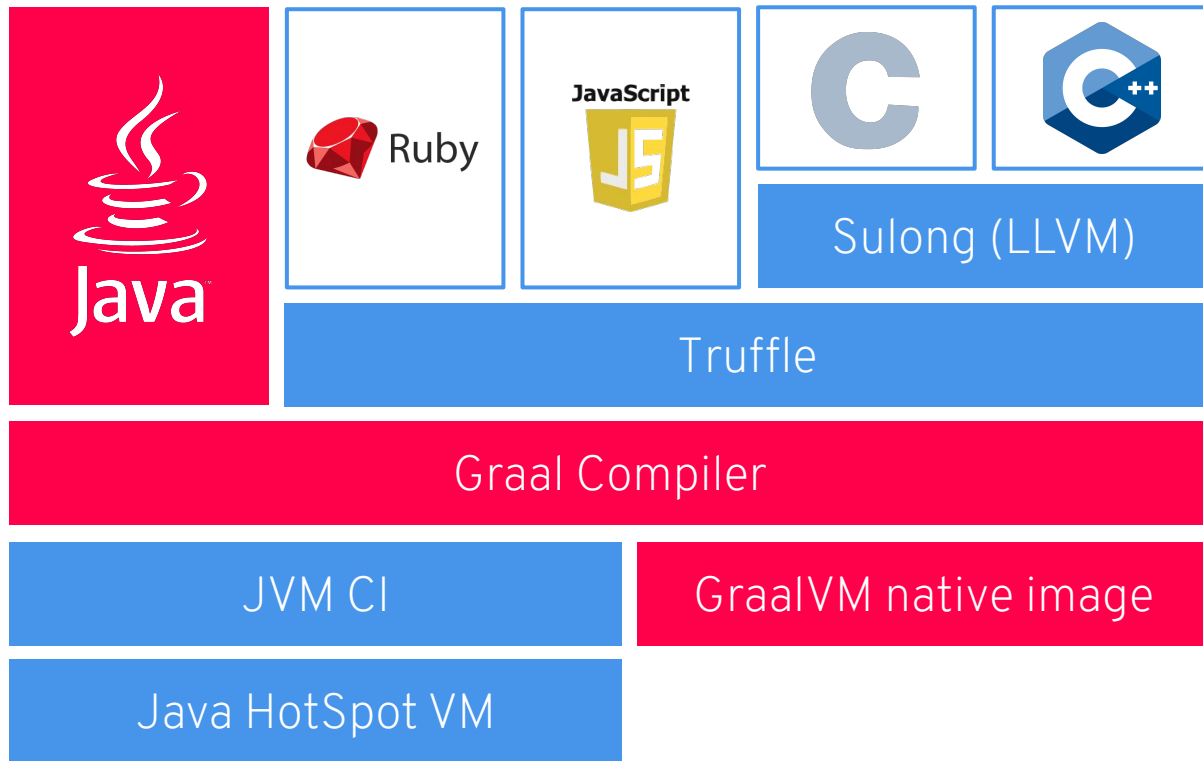
native-app

# Dead code elimination

Closed-world assumption







# 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)
- References (similar)



# GraalVM specific benefits

Drives the gathering of metadata needed by GraalVM

- based on framework knowledge
- Classes using reflection, resources, etc

Minimize dependencies

Help dead code elimination



# 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



## 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

Jandex

Gizmo

Graal SDK

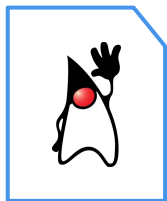
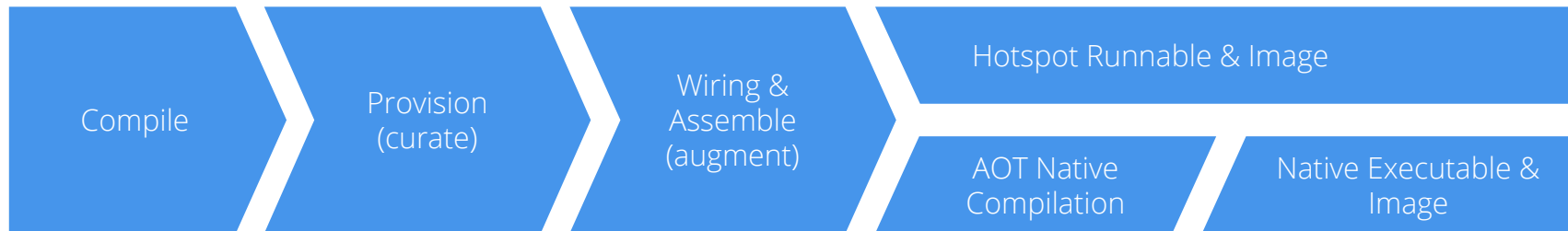
Arc (DI)

HotSpot (Any JDK)

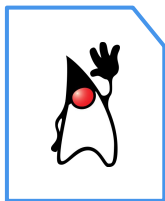
GraalVM Native Image



# Build Process



app.jar



frameworks



Runnable java app



native-app

# Quarkus Benefits

Developer Joy

Supersonic Subatomic Java

Unifies

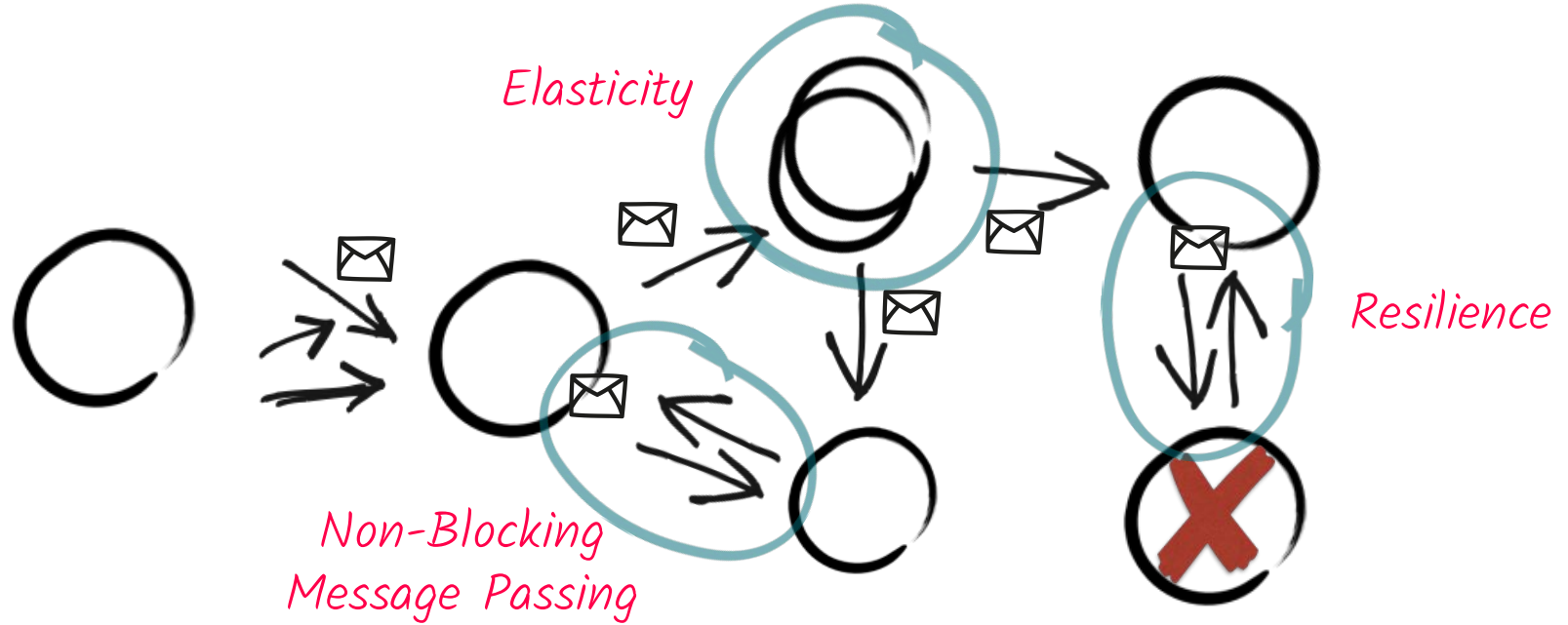
imperative and reactive

Best of breed

libraries and standards



# THE BENEFITS OF MESSAGING





Thank you.



# QUARKUS



<https://quarkus.io>



<https://quarkusio.zulipchat.com>



[@quarkusio](#)

