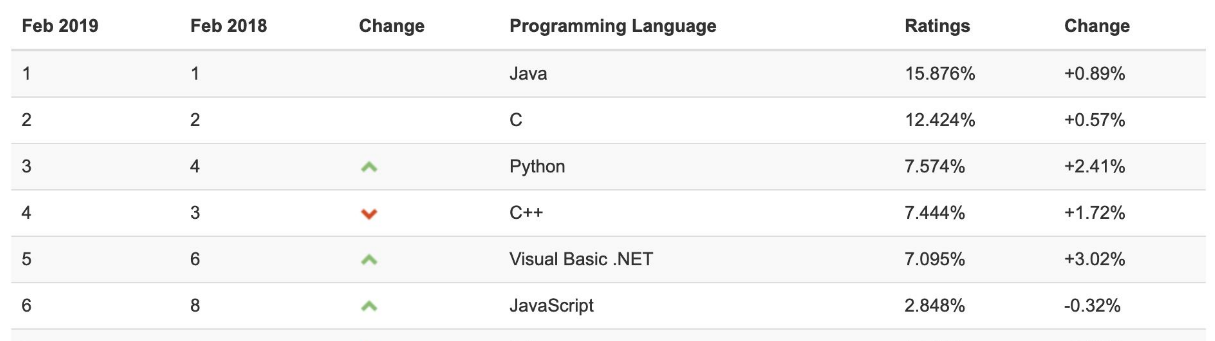
Introduction to **Quarkus:**

A Container-First Cloud Native Framework

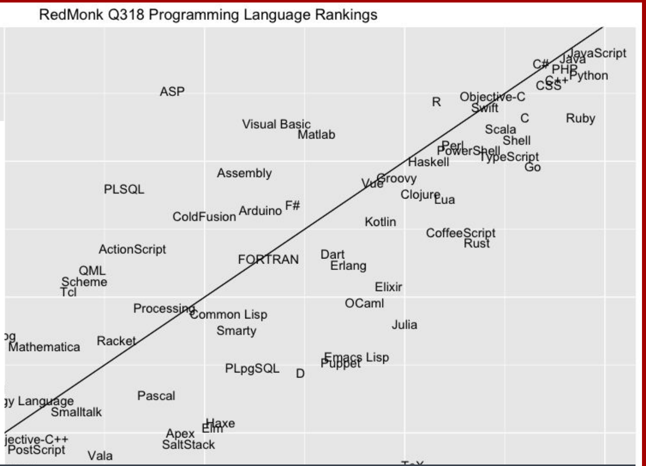
Georgios Andrianakis, Red Hat

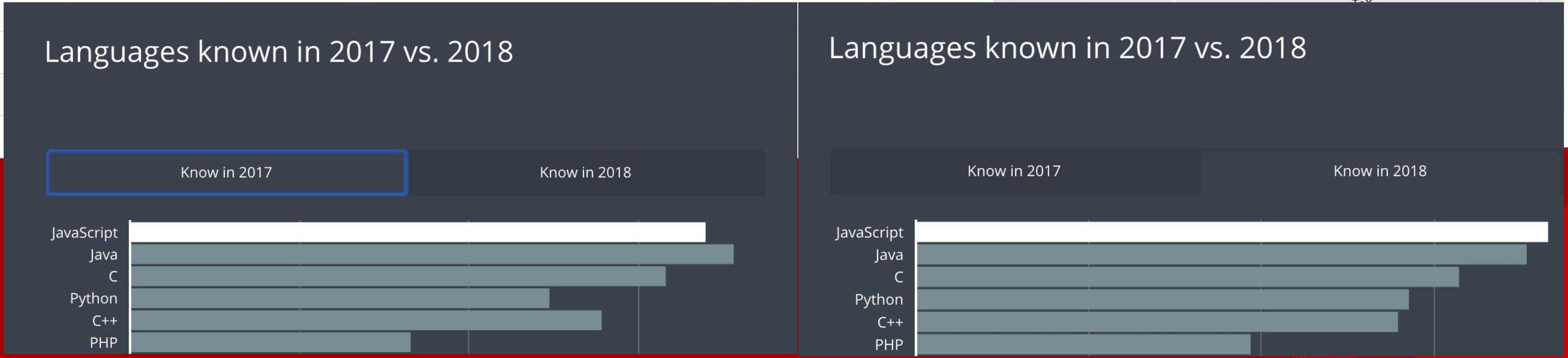
RedMonk

Tiobe Index



Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.876%	+0.89%
2	2		C	12.424%	+0.57%
3	4	▲	Python	7.574%	+2.41%
4	3	▼	C++	7.444%	+1.72%
5	6	▲	Visual Basic .NET	7.095%	+3.02%
6	8	▲	JavaScript	2.848%	-0.32%
7					
8					
9					
10					


RedMonk Q318 Programming Language Rankings

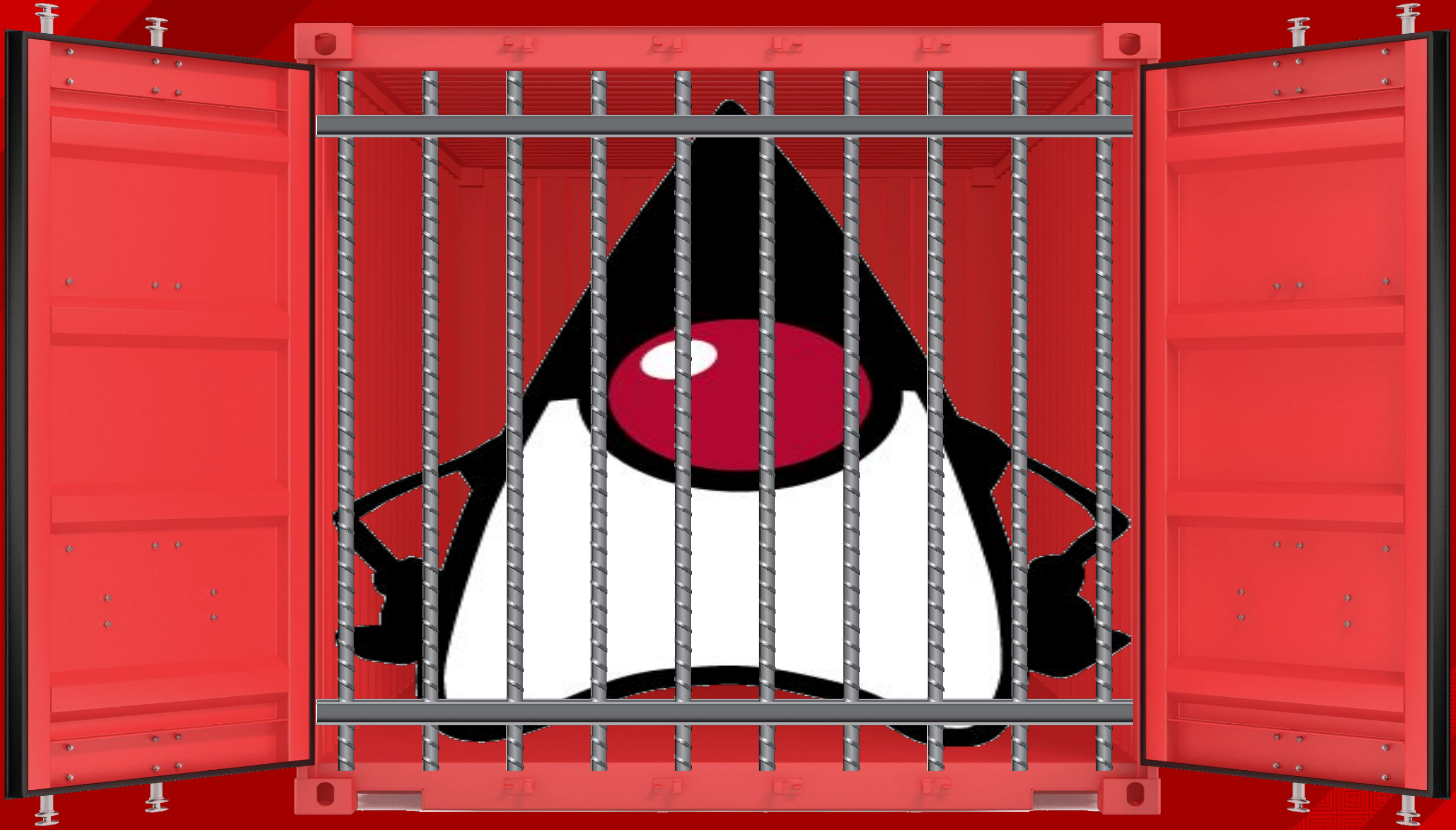


HackerRank

But Java has been showing its age…

Memory Hog

🐷

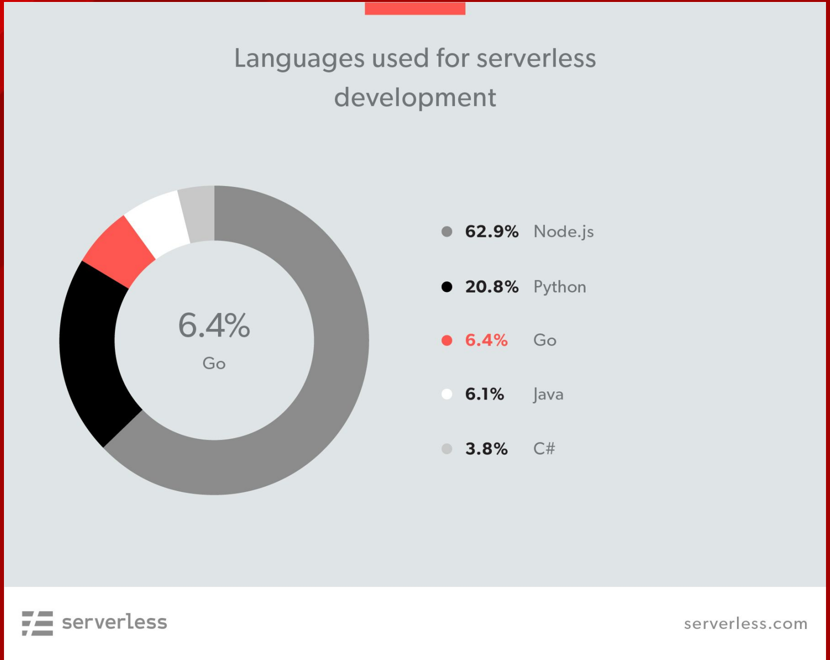


BOOM!

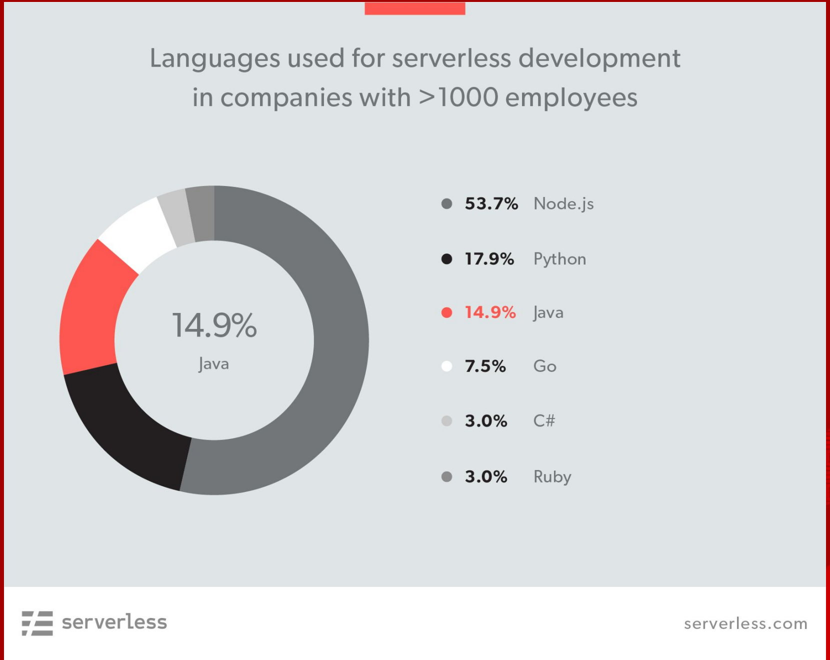
Slow Startup

🐍

Serverless Adoption



Languages used for serverless development



Languages used for serverless development in companies with >1000 employees

https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage

QUARKUS

Quarkus Features

No compromises ✨

📦 Fat Jars and Native Executables
☯️ Optimized for JAX-RS & JPA

Developer Joy ⌨️

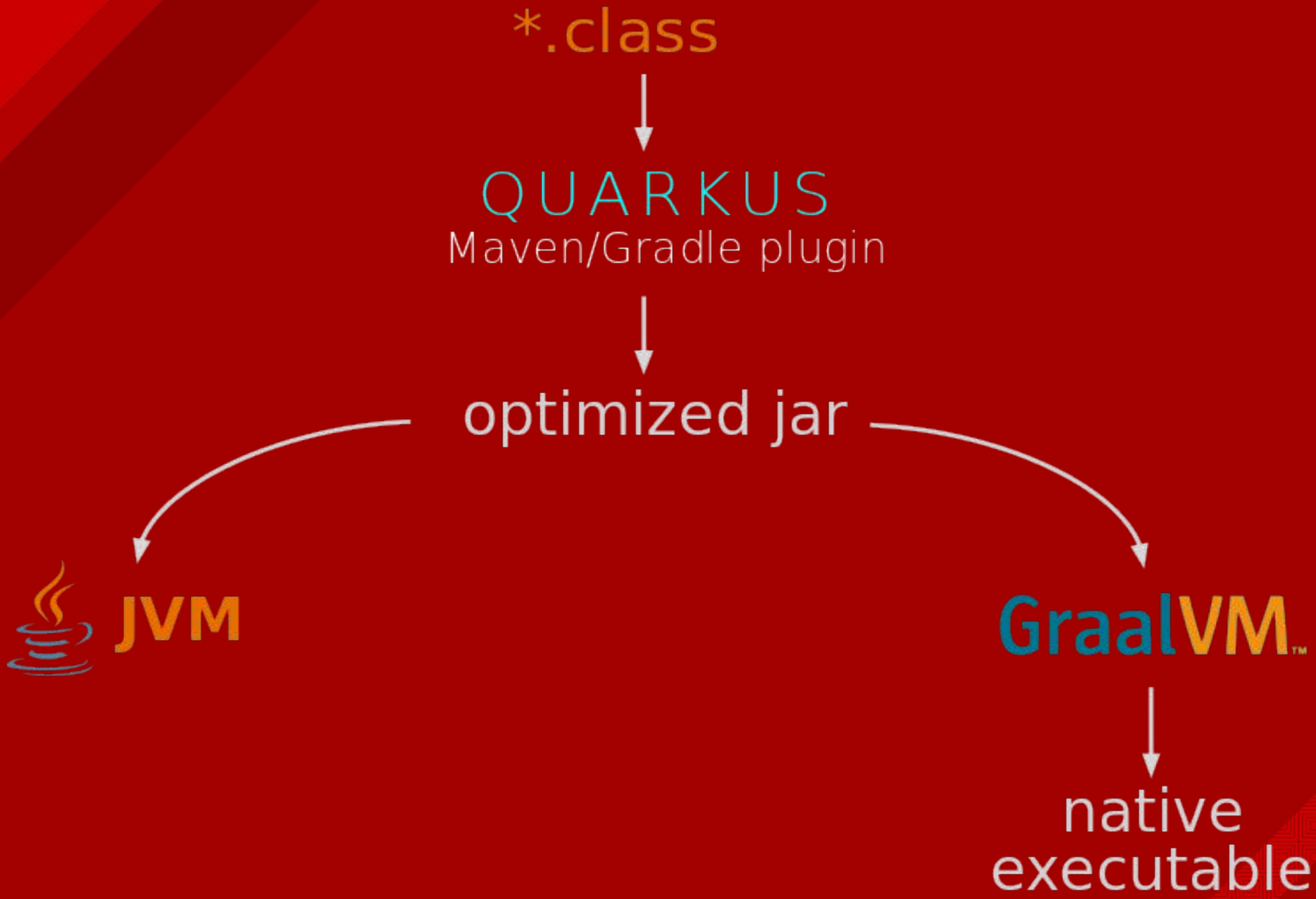
🔥 Live Reload
😍 Imperative and Reactive
⚛️ Serverless and Microservices

Optimized for the Cloud 🚀

💾 Lower memory usage
🚤 Faster startup
⏱️ Optimized for short-lived processes
⎈ Kubernetes Native

Supported Libraries and Standards

VERT.X
REST Easy
OPENSHIFT
JAEGER
kafka
Infinispan



*.class

QUARKUS

Maven/Gradle plugin

optimized jar

JVM

GraalVM™

native executable

Traditional Java apps (both app server and fat-jar stacks)

- Tons of classes loaded during boot time
  - Unused later on
  - Occupy a lot of memory
- Reflection used extensively

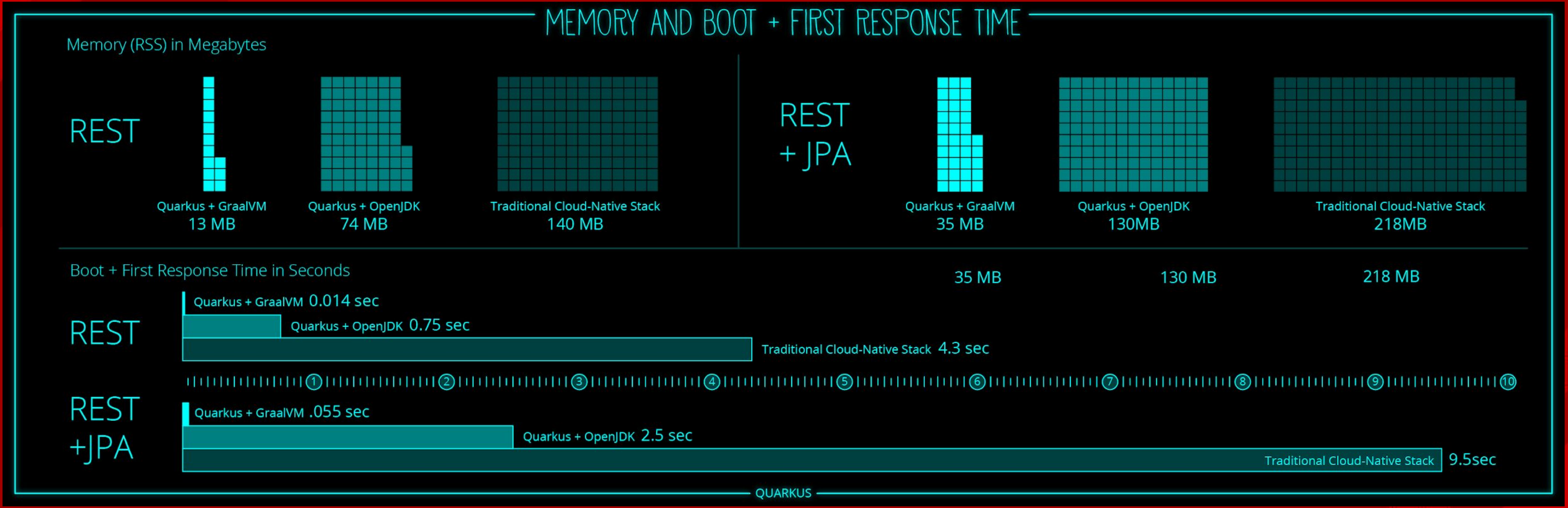
XML Parsers, Annotation models….

Quarkus

Do as much as possible at **build** time

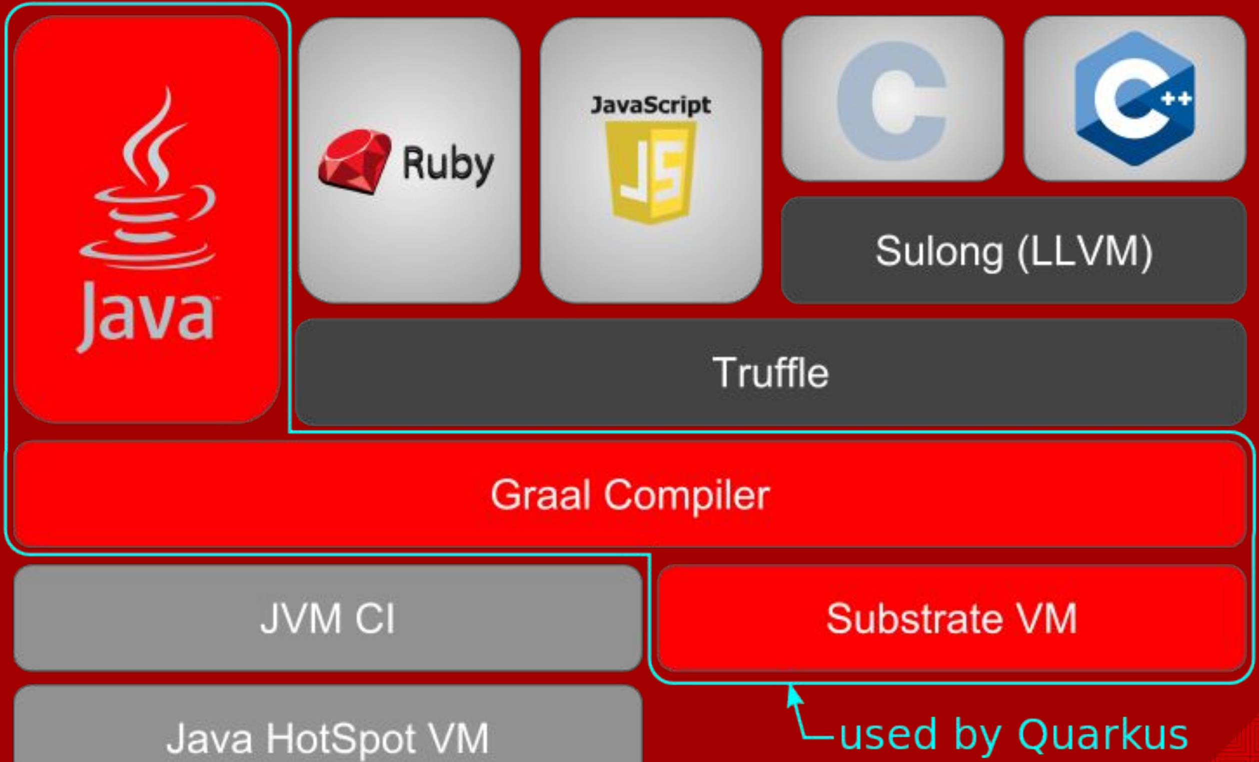
- Annotation processing
- Configuration parsing
- Throw away all classes that are not needed at runtime
- Avoid runtime reflection as much as possible

Output: **generated** classes

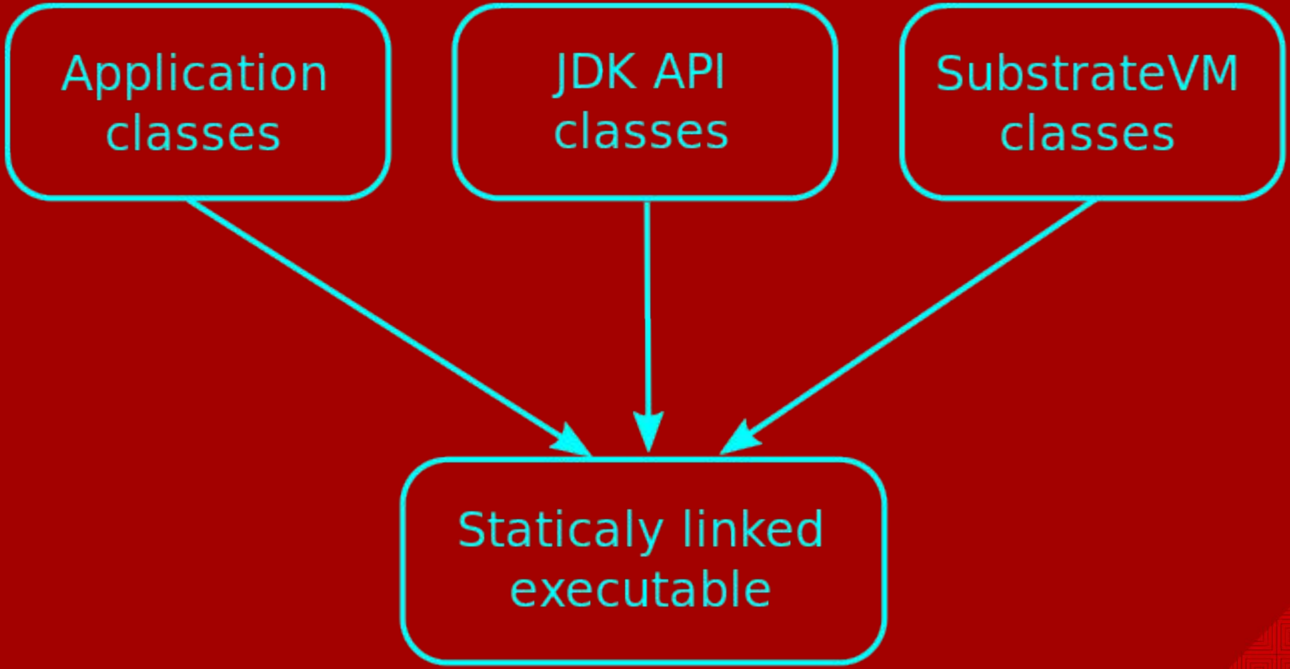

MEMORY AND BOOT + FIRST RESPONSE TIME
Memory (RSS) in Megabytes
REST
Quarkus + GraalVM
13 MB
Quarkus + OpenJDK
74 MB
Traditional Cloud-Native Stack
140 MB
REST
+ JPA
Quarkus + GraalVM
35 MB
Quarkus + OpenJDK
130MB
Traditional Cloud-Native Stack
218MB
Boot + First Response Time in Seconds
35 MB
130 MB
218 MB
REST
Quarkus + GraalVM 0.014 sec
Quarkus + OpenJDK 0.75 sec
Traditional Cloud-Native Stack 4.3 sec
1
2
3
4
5
6
7
8
9
10
REST
+JPA
Quarkus + GraalVM .055 sec
Quarkus + OpenJDK 2.5 sec
Traditional Cloud-Native Stack 9.5sec
QUARKUS

GraalVM™

GraalVM


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

AoT with GraalVM



Application classes

JDK API classes

SubstrateVM classes

Staticaly linked executable

AoT with GraalVM

- Static Analysis
- Closed world assumptions
- Dead code elimination

GraalVM Limitations

- Dynamic classloading
  - Creating, reloading classes at runtime is not possible
- JVMTI, JMX
  - No agents
- InvokeDynamic, MethodHandles
  - But lambdas are supported

GraalVM Limitations

- Reflection
  - All targets of reflection need to be known
- Dynamic Proxies
  - All classes that will get proxied at runtime need to be declared
- Classpath resources
  - All resources to be included must be declared

GraalVM Limitations

Example of manual invocation:

```
native-image -jar target/app.jar -H:ReflectionConfigurationResources=reflection_config.json
-H:Name=name
--delay-class-initialization-to-runtime=io.netty.handler.codec.http.HttpObjectEncoder
```

Quarkus shields you from all GraalVM peculiarities!

Demo time!

Thank you!

Twitter: @geoand86

Further reading

- http://quarkus.io
- https://developers.redhat.com/blog/2019/03/07/quarkus-next-generation-kubernetes-native-java-framework/
- http://in.relation.to/2019/03/08/why-quarkus/

Q&A