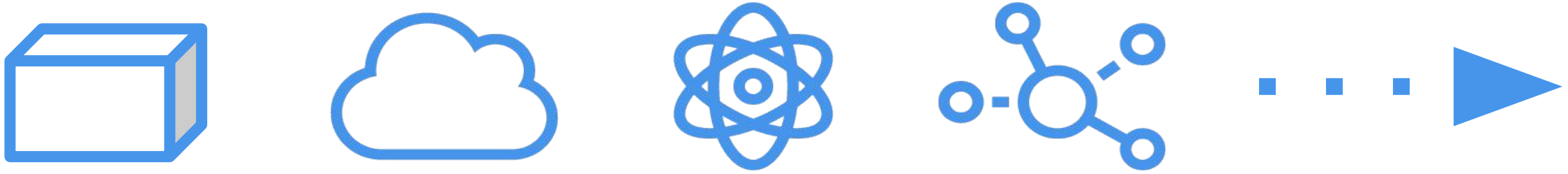
Developing with Quarkus: Supersonic Subatomic Java

Hands-On Workshop

James Falkner
Application Runtimes Technical Director, Red Hat

Growth in Application Architecture Choices



Monolith

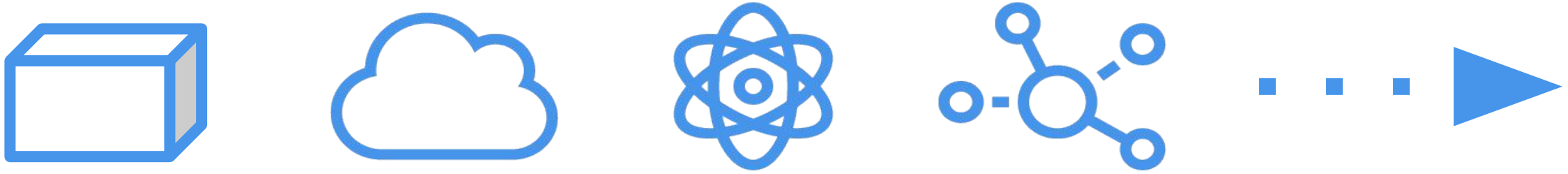
Cloud Native

Microservices

Serverless

Event-Driven Architecture

Common Deployment Platform



Monolith

Cloud Native

Microservices

Serverless

Event-Driven Architecture

kubernetes

Istio

Knative

Historical Enterprise Java Stack

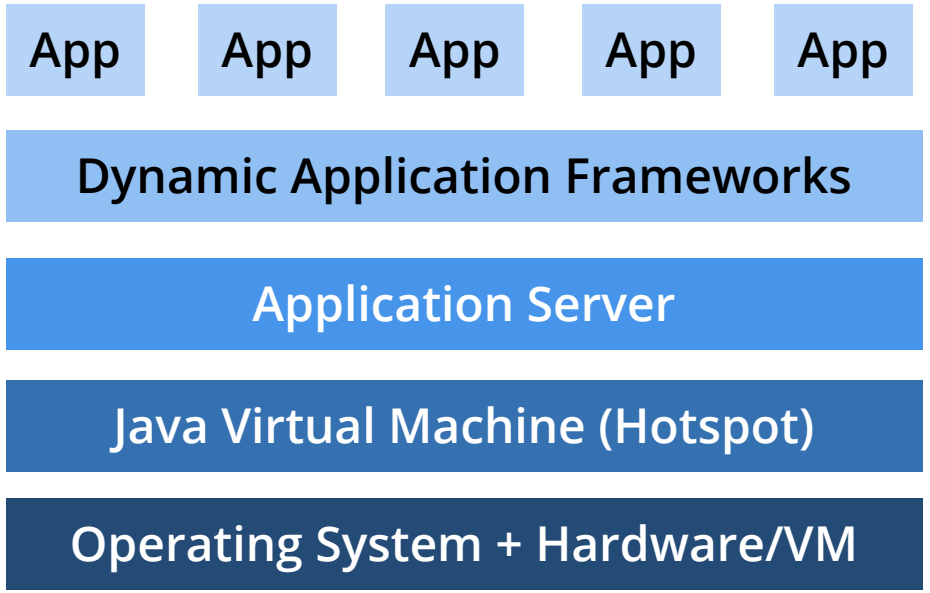
Architecture: Monoliths

Deployment: multi-app, appserver

App Lifecycle: Months

Memory: 1GB+ RAM

Startup Time: 10s of sec



Modern Enterprise Java Stack

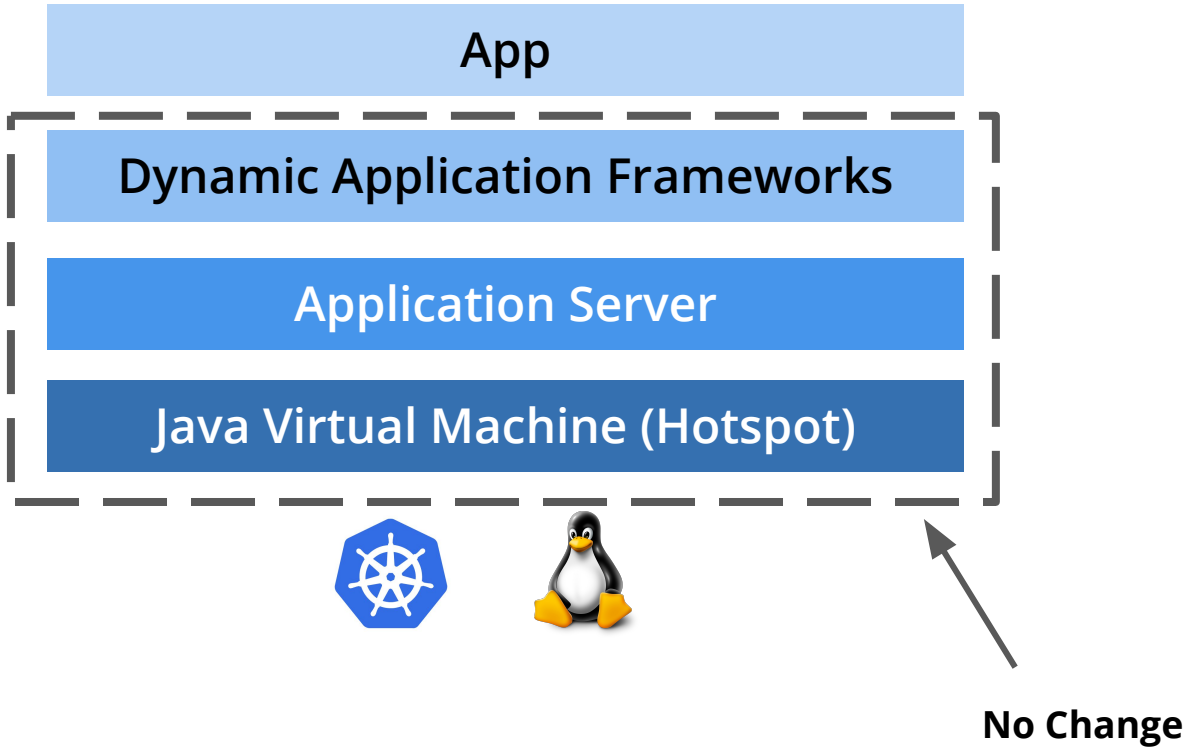
Architecture: Microservices

Deployment: Single App

App Lifecycle: Days

Memory: 100MBs+ RAM

Startup Time: Seconds



~~Java Serverless Stack~~

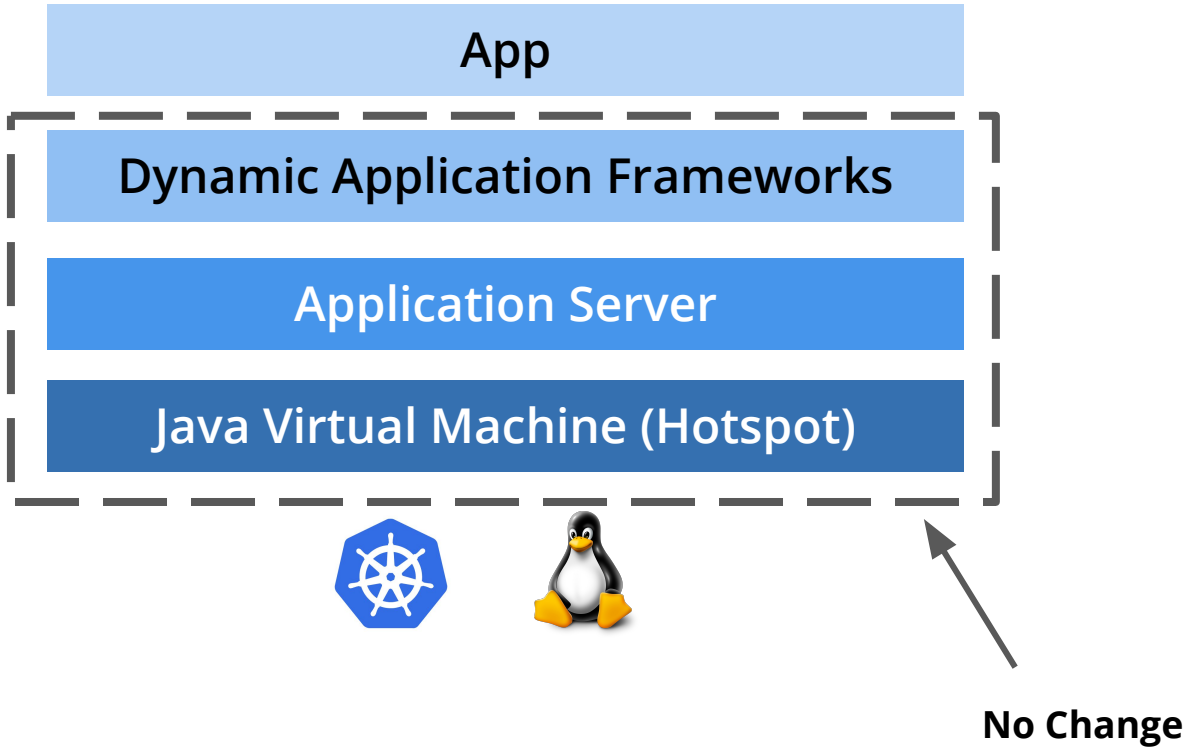
Architecture: Microservices

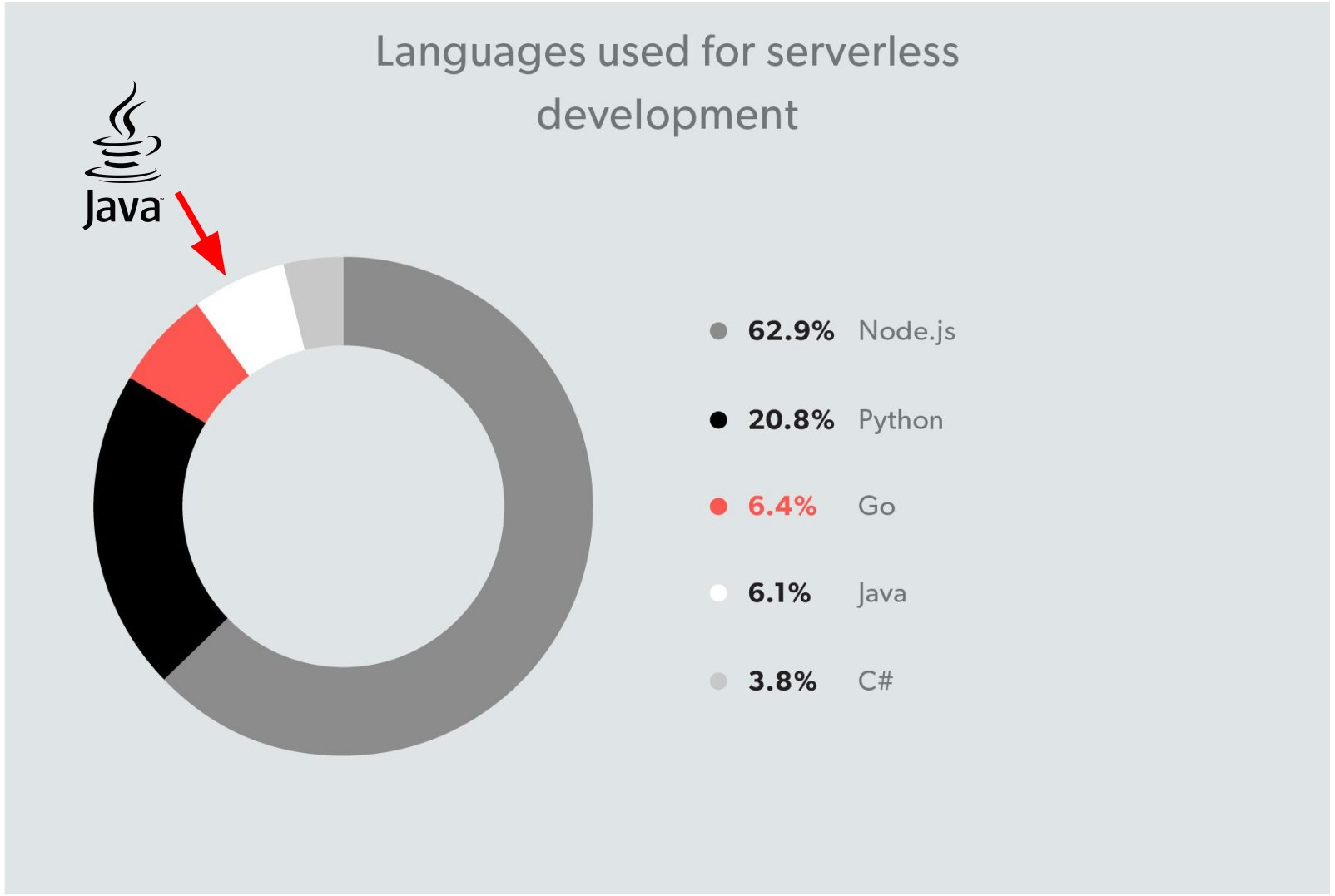
Deployment: Single App

App Lifecycle: Days

Memory: 100MBs+ RAM

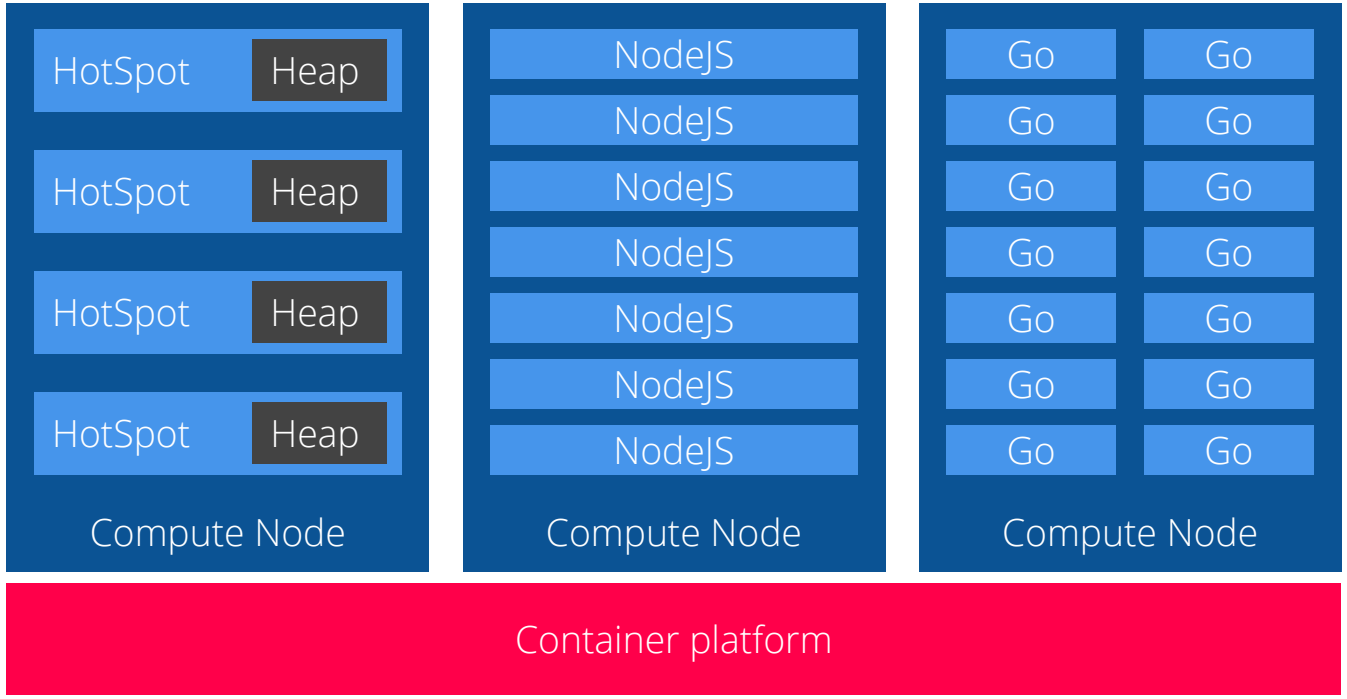
Startup Time: Seconds





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

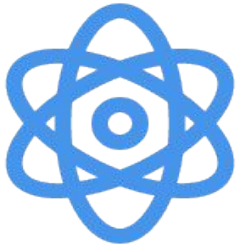
The Hidden Truth About Java + Containers



Quarkus - Kubernetes Native Java

Monolith

Cloud Native



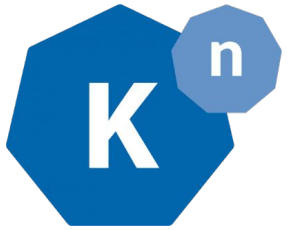
Microservices

Serverless

Event-Driven Architecture

kubernetes

Istio



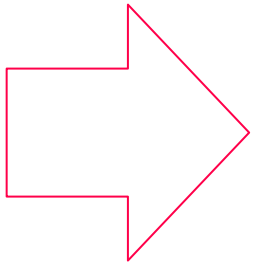
Knative

Moving to Compile-Time Boot

A dynamic runtime on immutable infrastructure is unnecessary overhead

What does a framework do at startup time

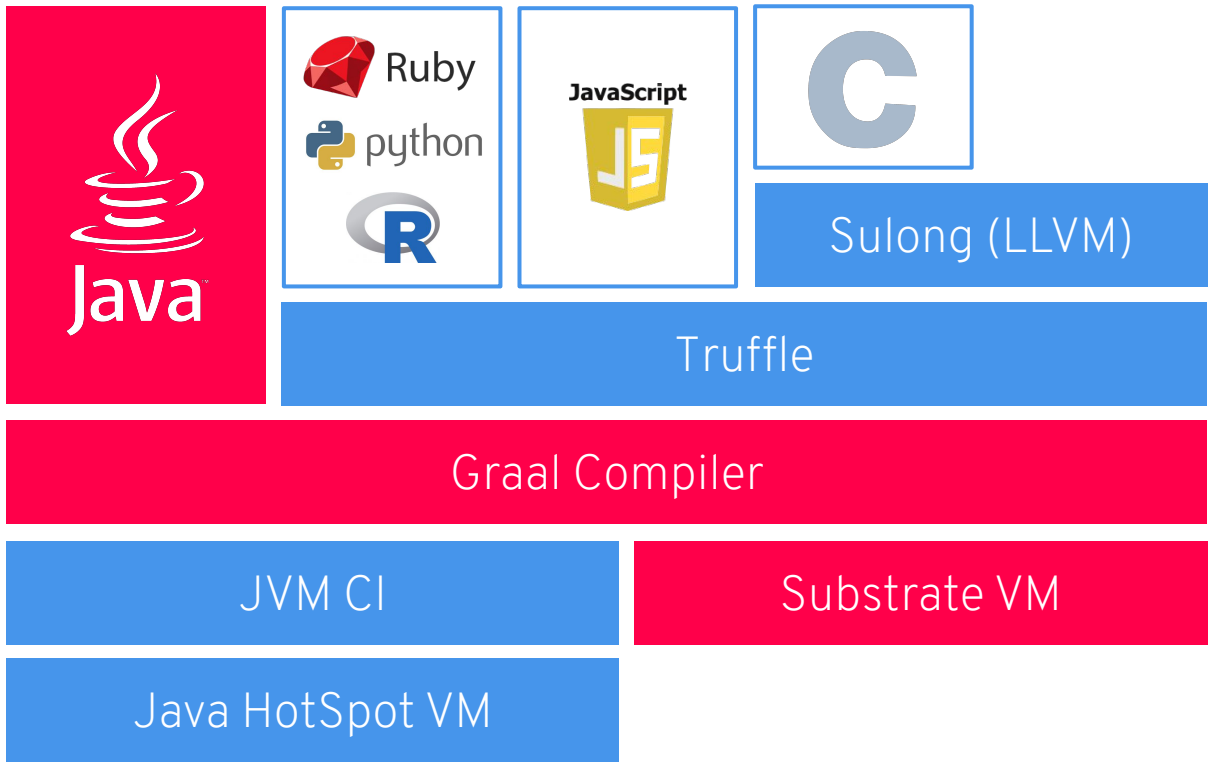
- Parse config files
- Classpath & classes scanning
  - for annotations, getters or other metadata
- Build framework metamodel objects
- Prepare reflection and build proxies
- *Start and open IO, threads etc*



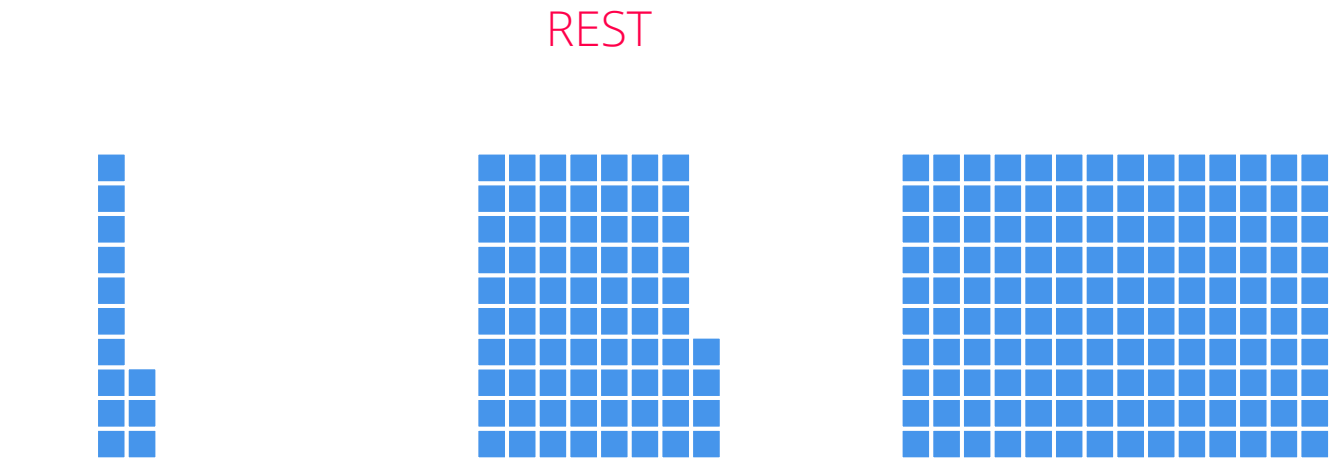
Framework Optimizations

- Moved as much as possible to build phase
- Minimized runtime dependencies
- Maximize dead code elimination
- Introduced clear metadata contracts
- Spectrum of optimization levels (all → some → no runtime reflection)

GraalVM

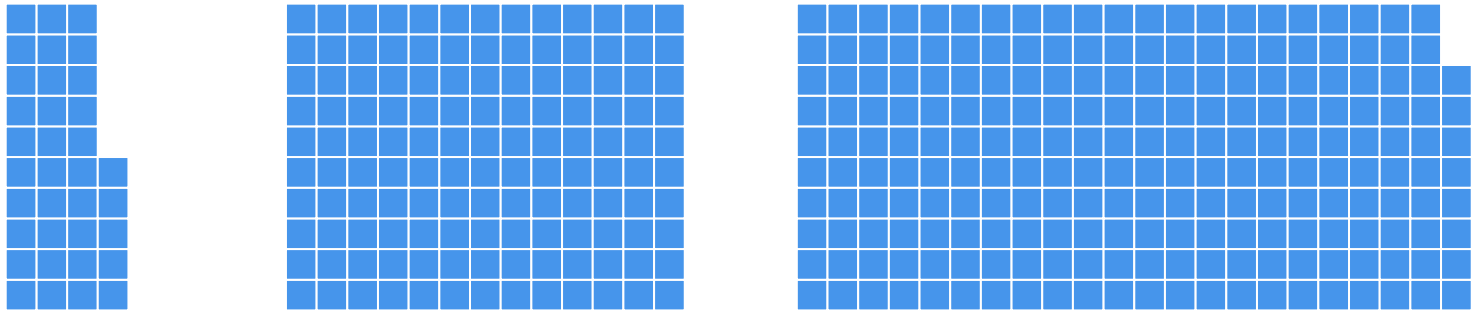

Java
Ruby
python
R
JavaScript
C
Sulong (LLVM)
Truffle
Graal Compiler
JVM CI
Substrate VM
Java HotSpot VM

Quarkus Reduces Memory Utilization



Quarkus Reduces Memory Utilization

REST + CRUD

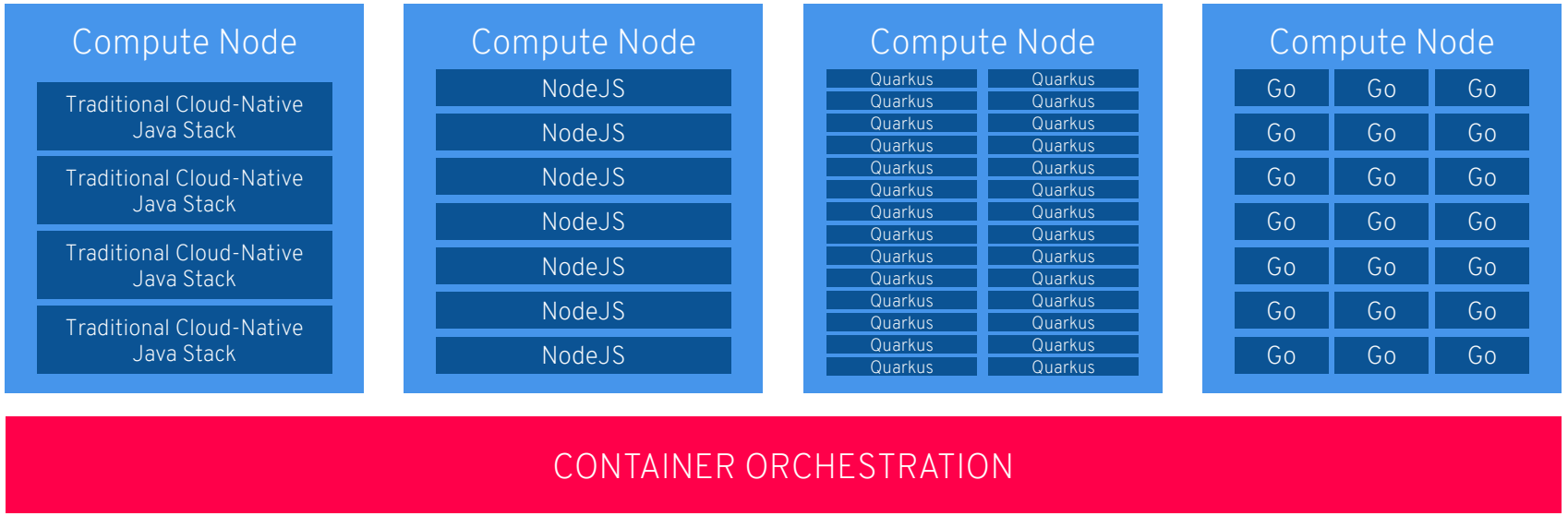


Quarkus + GraalVM
35 MB

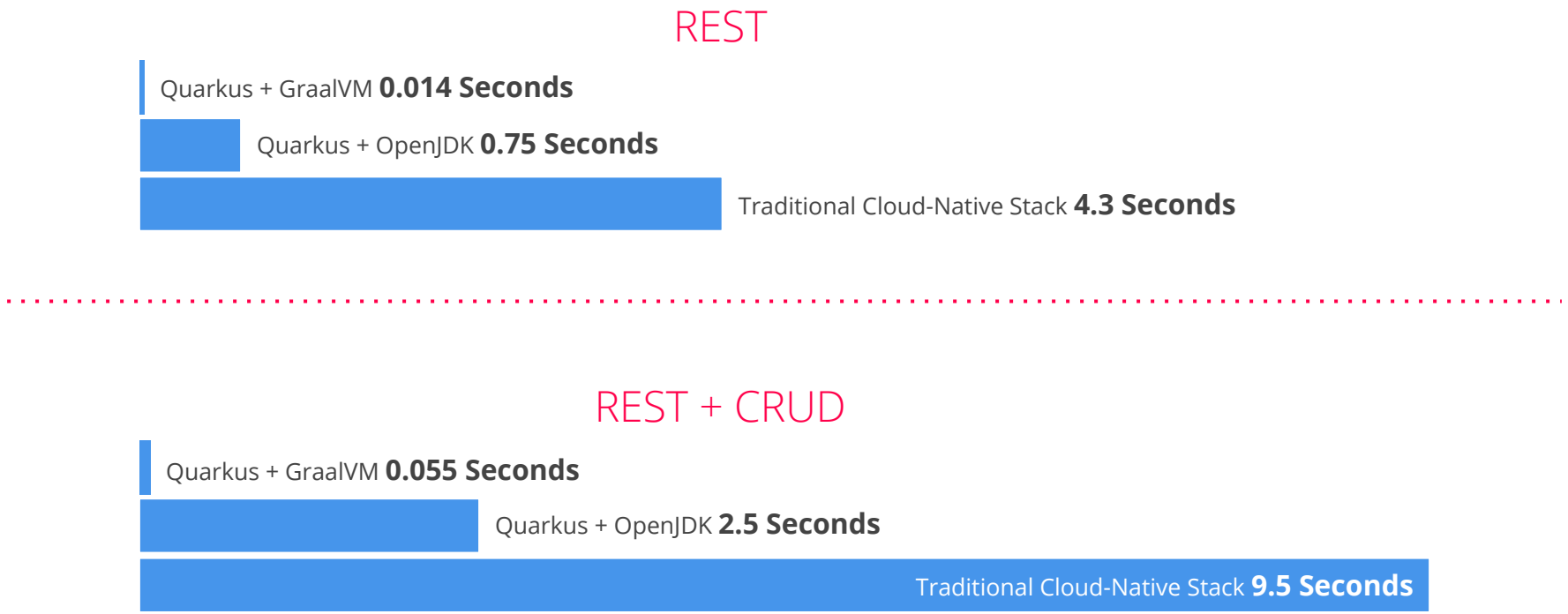
Quarkus + OpenJDK
130 MB

Traditional Cloud-Native Stack
218 MB

The New Truth about Java + Containers



Quarkus Improve Startup Time



Quarkus

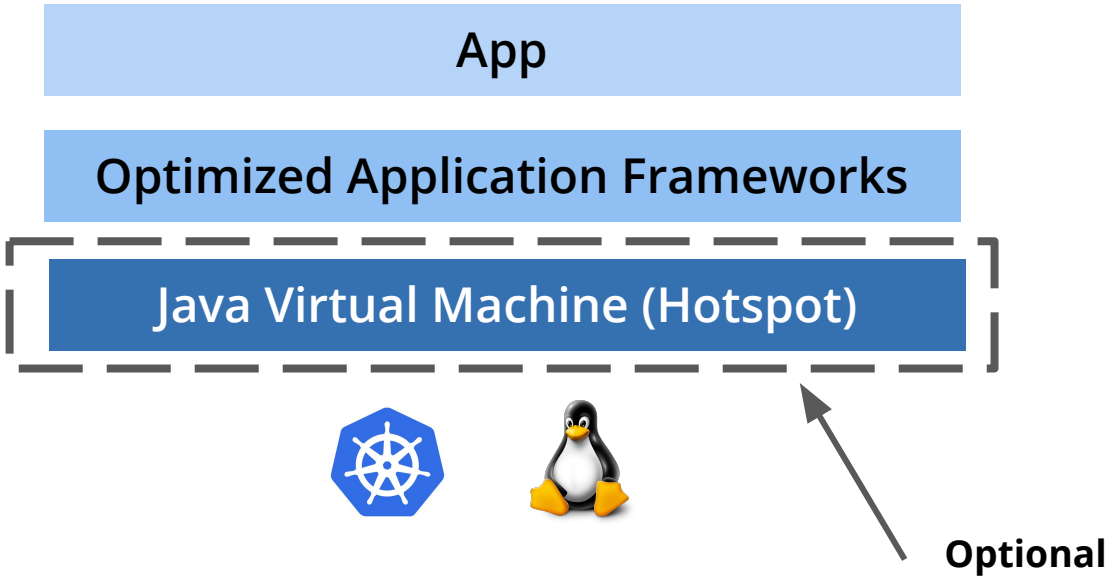
Architecture: Microservices, Serverless

Deployment: Single App

App Lifecycle: Seconds to Days

Memory: 10MBs+ RAM

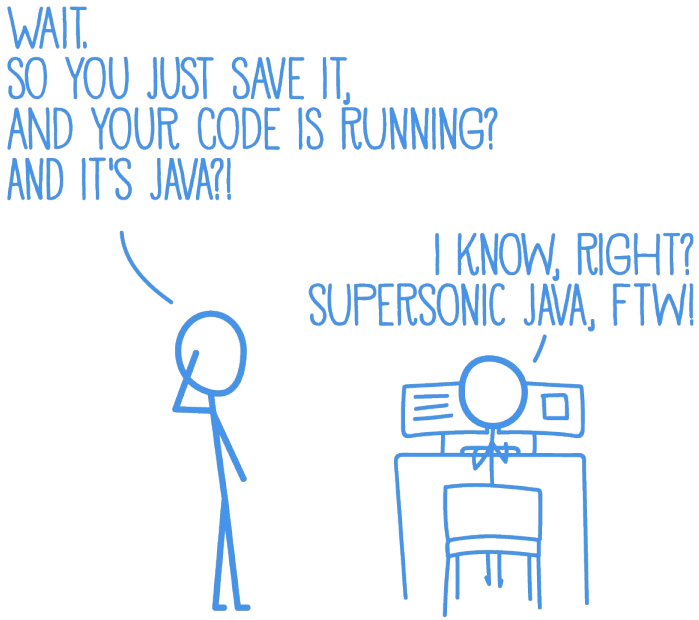
Startup Time: Milliseconds



Developer Joy

A cohesive platform for optimized developer joy:

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



Unifies Imperative and Reactive

```
@Inject
SayService say;

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

```
@Inject @Stream("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
- Inject the EventBus or the Vertx context
- Use the technology that fits your use-case

Best of Breed Frameworks and Standards

Eclipse Vert.x

Hibernate

RESTEasy

Apache Camel

Eclipse MicroProfile

Netty

Kubernetes

OpenShift

Jaeger

Prometheus

Apache Kafka

Infinispan

“Feels familiar and new at the same time”