Software Transactional Memory and Microservices: A Match Made in Heaven

Mark Little, VP Red Hat UK Ltd

Michael Musgrove, Red Hat UK Ltd

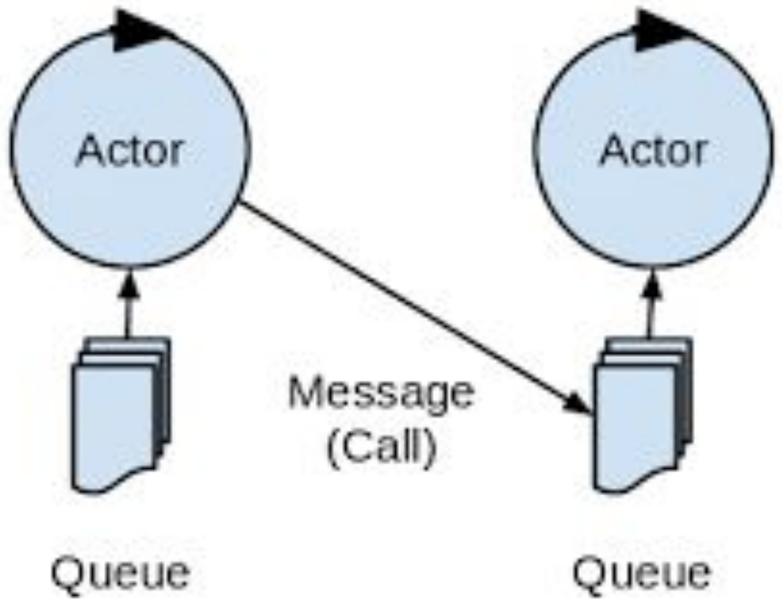
17th Sept, 2019

Agenda

- The Actor Model
- Distributed Transactions and Software Transactional Memory (STM)
- Microservices
- Why do these all belong together?
- Technology (Quarkus, Vert.x, Narayana STM)
- Coding demo

The Actor Based Programming Model

- Actors and CSP have been around for decades
  - CSP from Hoare, 1985
  - Actor model from Hewitt et al, 1973
- But popular ways to model primitives for concurrent computations
  - Embodies Processing, Storage and Communication
- Distributed computations communicate via message passing

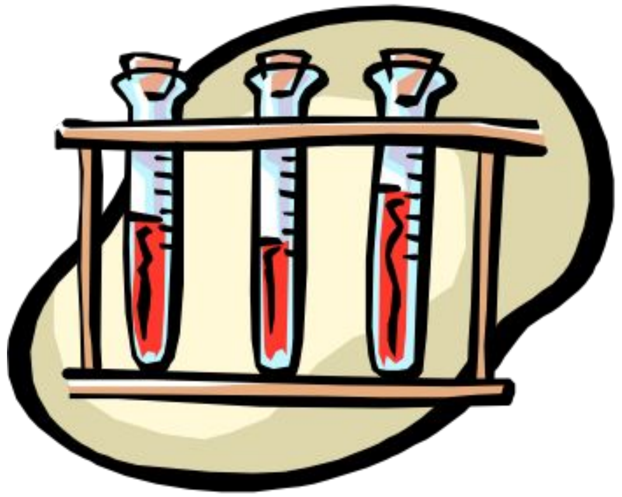


Nice Features of the Actor Model

- Fixed message sets (i.e., no hidden or unexpected interactions)
- Simplified data management (state is internal to the actor)
- Location transparency (because other actors only see the address)
- Loose coupling
- Asynchronous message passing
  - Has much in common with the reactive manifesto (Responsive, Elastic, Asynchronous and, with extensions, Resilience).

Distributed Transactions

- ACID properties
- Two-phase commit
  - Required when there is more than one resource (RM)
  - Managed by the transaction manager (TM)
  - Uses a familiar two-phase technique (2PC)



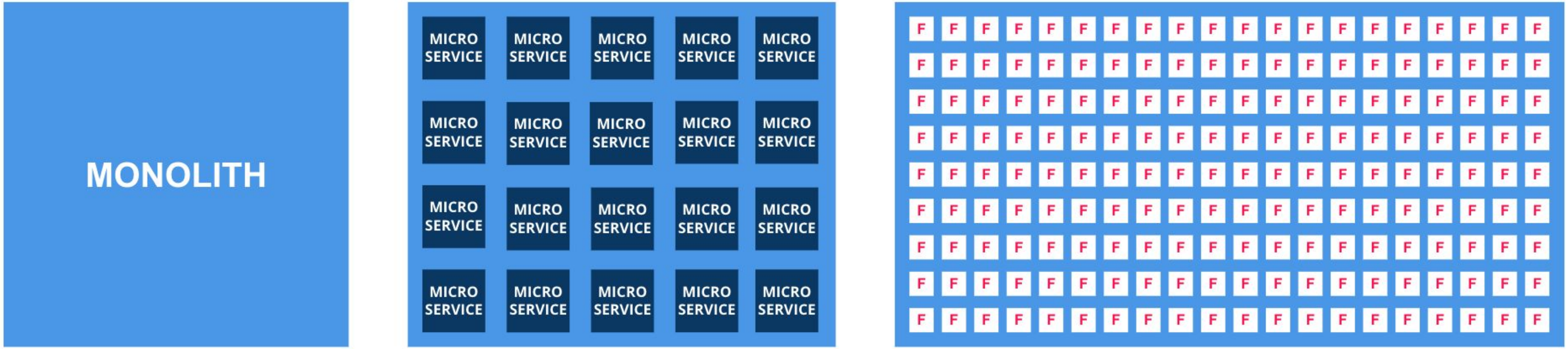
Software Transactional Memory

- Software Transactional Memory (STM) proposed in 1995
- STM is about ease of use and reliability
  - Access shared state, either for reading or writing, occurs within atomic blocks
  - All code inside an atomic block executes as if it were single threaded
  - Less error prone (the atomic block is the protection) than traditional concurrency primitives or placing composite operations behind an API
  - Some implementations can be lock free (optimistic vs pessimistic, timestamp)
  - Has some of the same characteristics of ACID transactions

Transactions and Actors

- An actor may go through multiple state transitions upon receipt of a message
- An actor could be internally implemented using multiple threads
- Computational failures may occur
- Hardware and software failures may occur
- Consistency of state important
- Composition of actors
- The combination of STM and Actors is fairly natural

Microservices 101



Nizar S.
@natewave

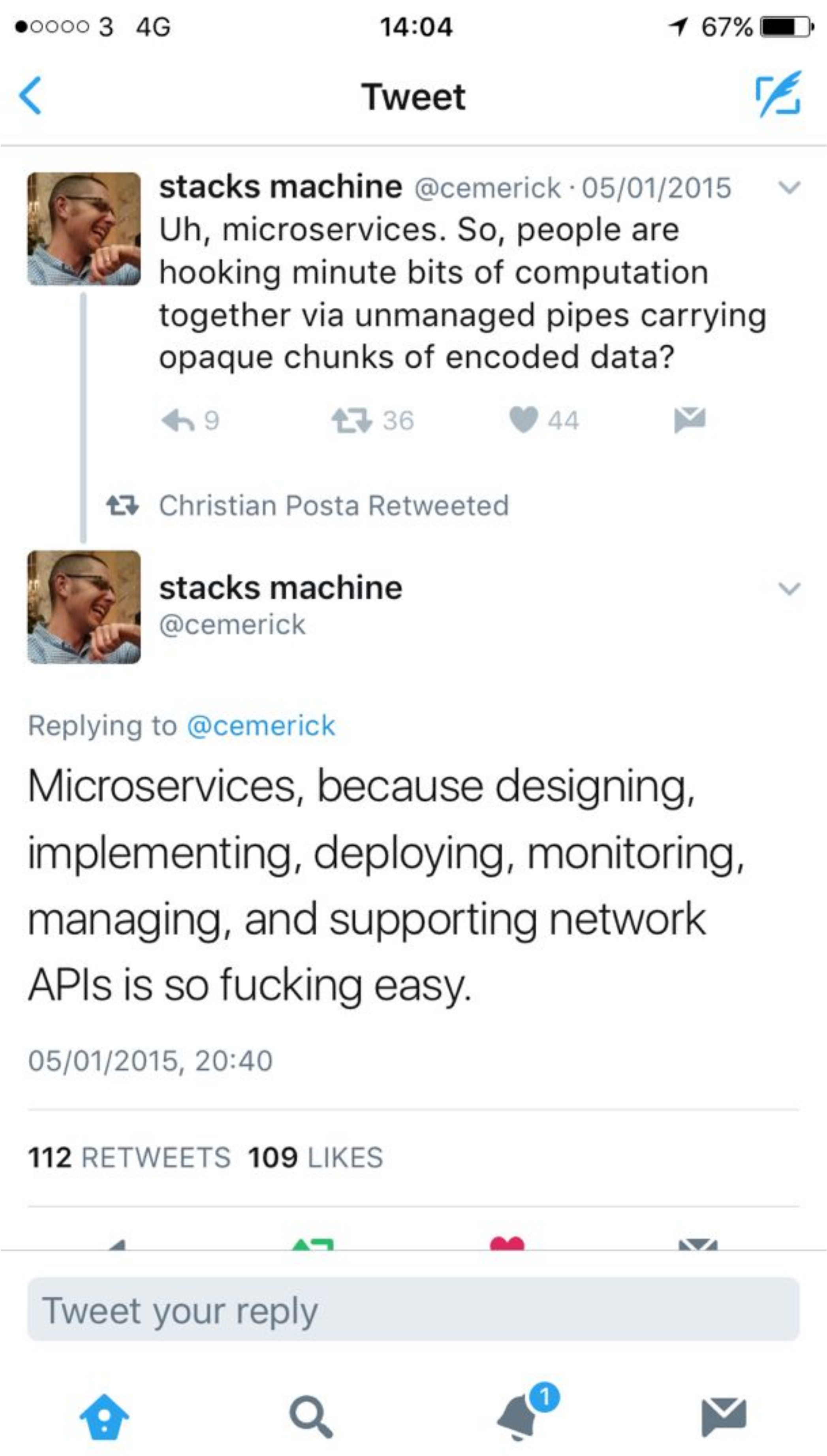
Follow

Finally, thanks to microservices, my dream of being a detective has come true. Every bug is more like a murder mystery.

RETWEETS	LIKES
4	6

1:18 PM - 11 Jun 2016

4 6


●○○○○ 3 4G
14:04
67%
Tweet
stacks machine @cemerick · 05/01/2015
Uh, microservices. So, people are hooking minute bits of computation together via unmanaged pipes carrying opaque chunks of encoded data?
9
36
44
Christian Posta Retweeted
stacks machine
@cemerick
Replying to @cemerick
Microservices, because designing, implementing, deploying, monitoring, managing, and supporting network APIs is so fucking easy.
05/01/2015, 20:40
112 RETWEETS 109 LIKES
Tweet your reply
1

OK so WHAT are they then?!

- Microservices pushes us (back) to distributed systems (circa 1996)
  - Services inherently in separate machines (physical or virtual)
  - Communication between distributed systems slower than within same address space
  - Failures can happen independently
  - Cascading failures can happen
  - Consistency concerns become even more of a challenge
  - Sh*t happens!
  - Throwing developers into distributed systems 101 is not a good way to be agile!

Enterprise microservices

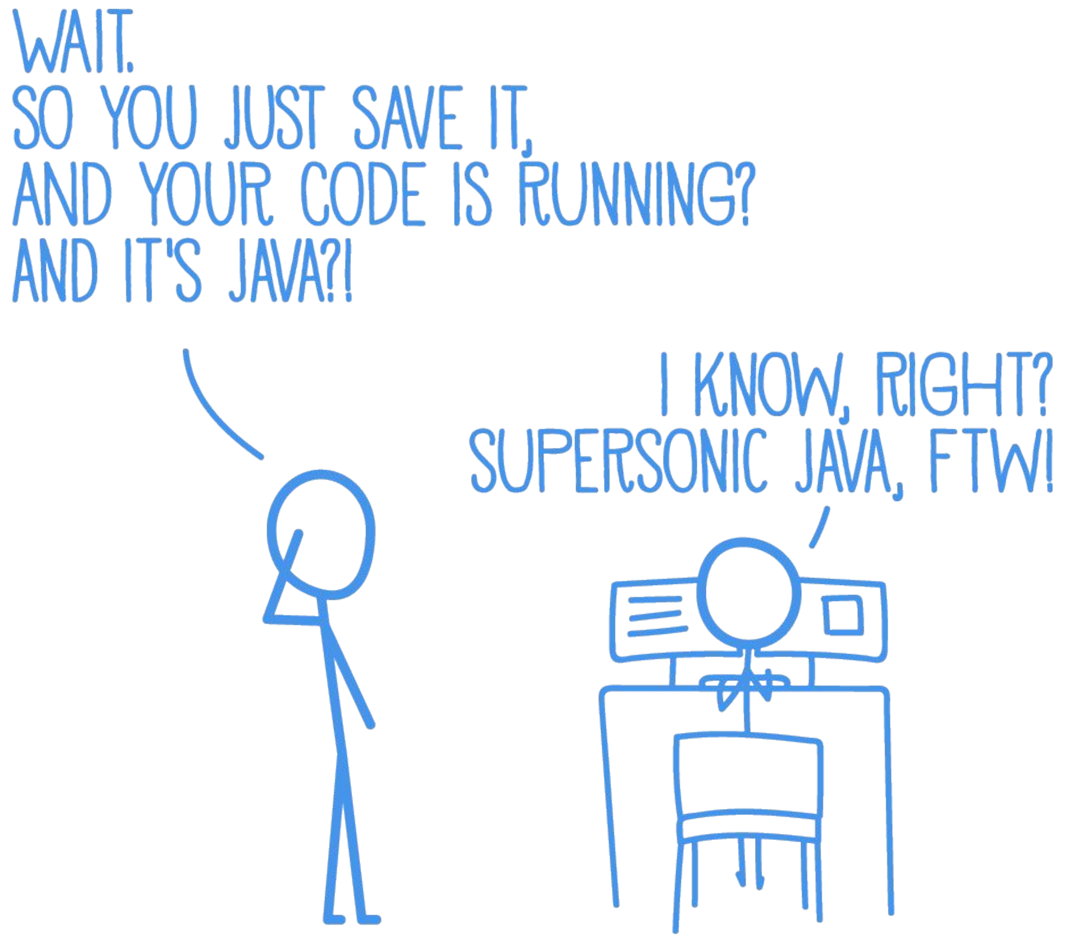
- Microservices distributed systems present challenges
  - The need for transactions, reliable messaging etc. doesn’t go away
  - If anything it becomes more important to developers
  - Application containers breaking into pieces
  - Independently deployable (Linux container based) services
  - Available to different language clients using REST/HTTP
  - And other protocols?
  - Still A LOT slower than in-memory!

Notable Technology Used in the Demonstration

Quarkus

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



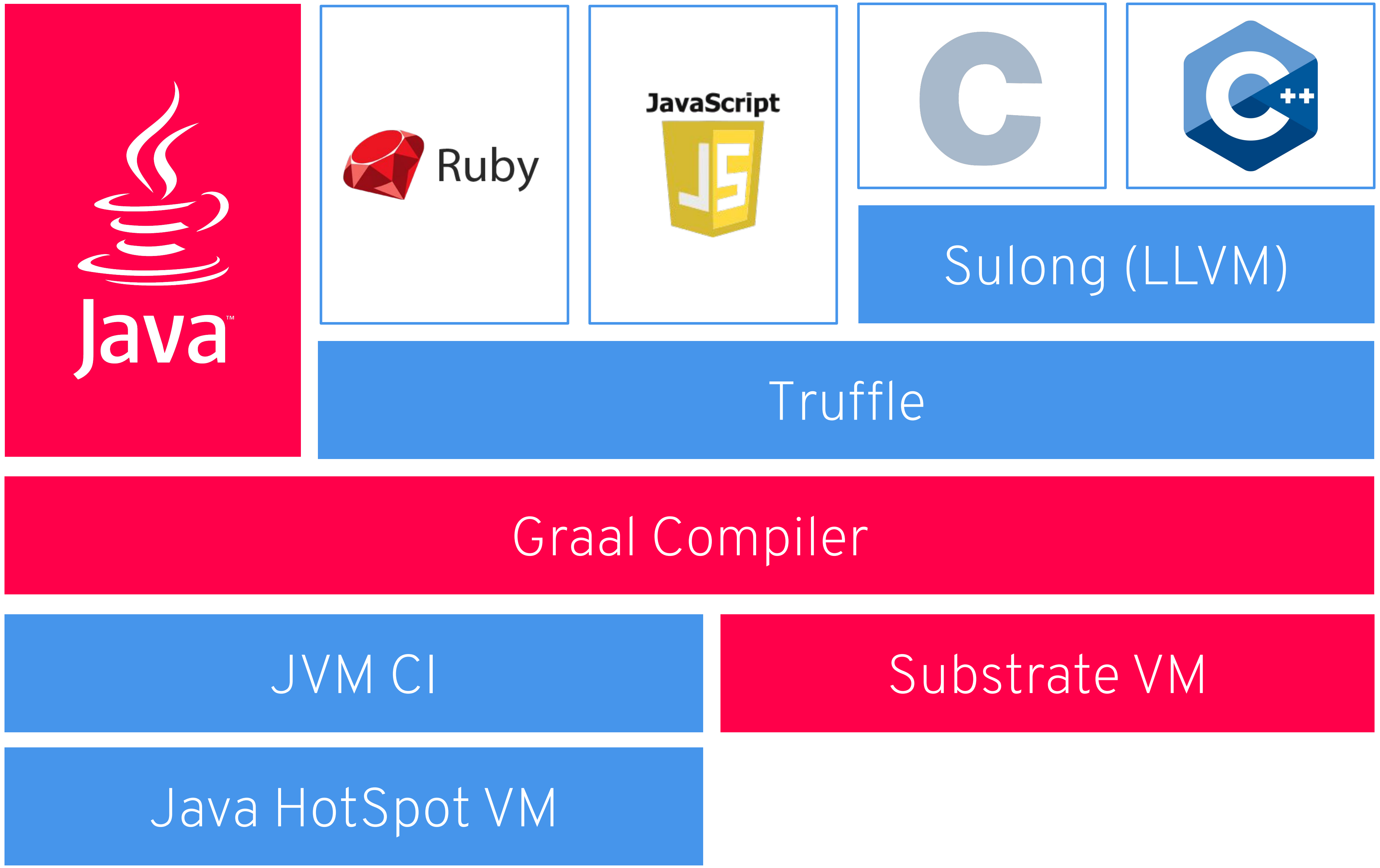
Boot time to compile time

What does a framework do at startup time?

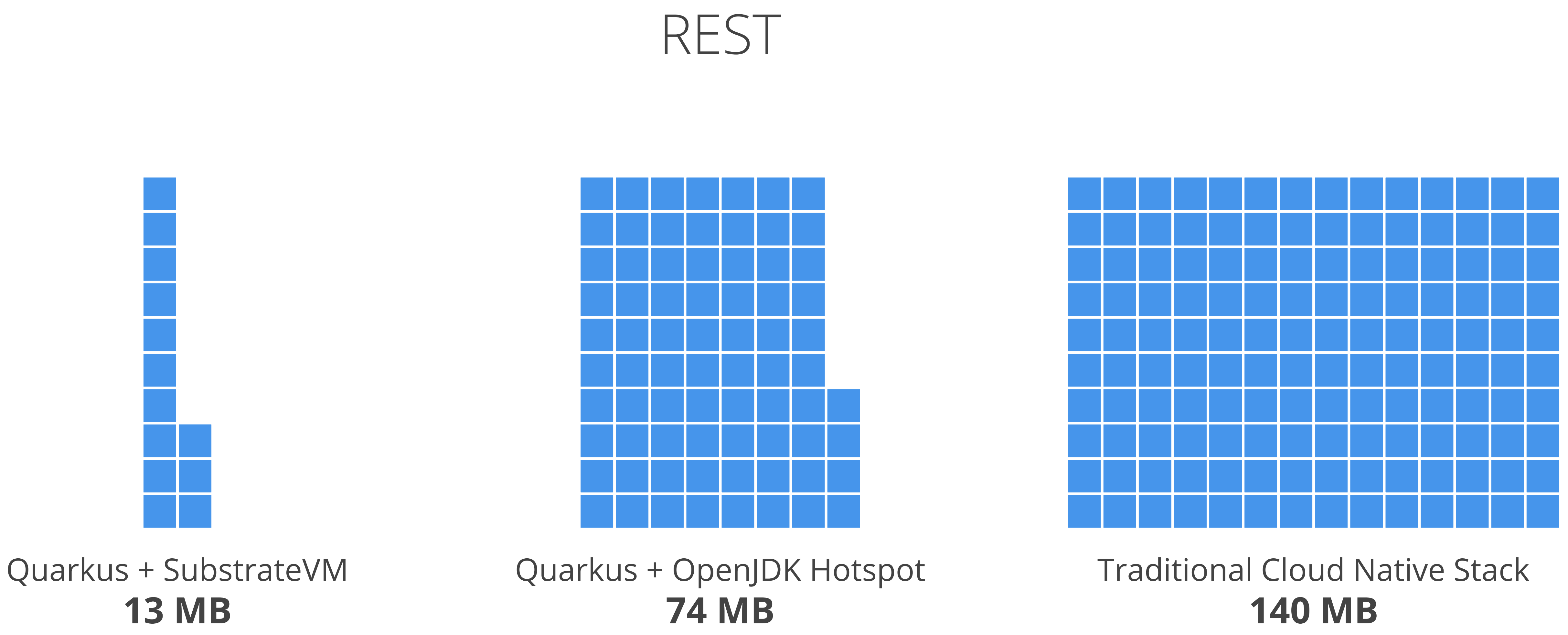
- Parse config files
- Classpath & classes scanning
    - for annotations, getters or other metadata
- Build framework metamodel objects
- Prepare reflection and build proxies
- *Start and open IO, threads etc*

Framework Optimizations

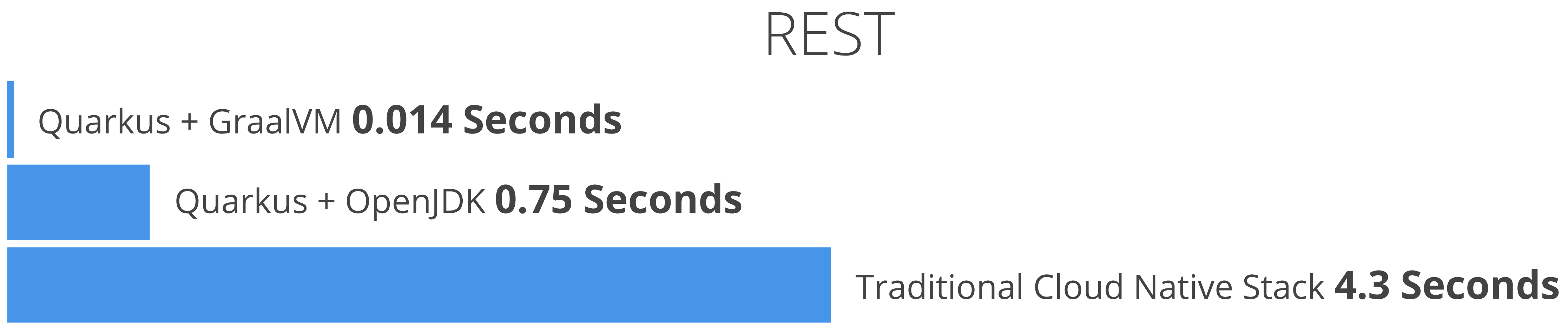
- Moved as much as possible to build phase
- Minimized runtime dependencies
- Maximize dead code elimination
- Introduced clear metadata contracts
- Spectrum of optimization levels (all → some → no runtime reflection)

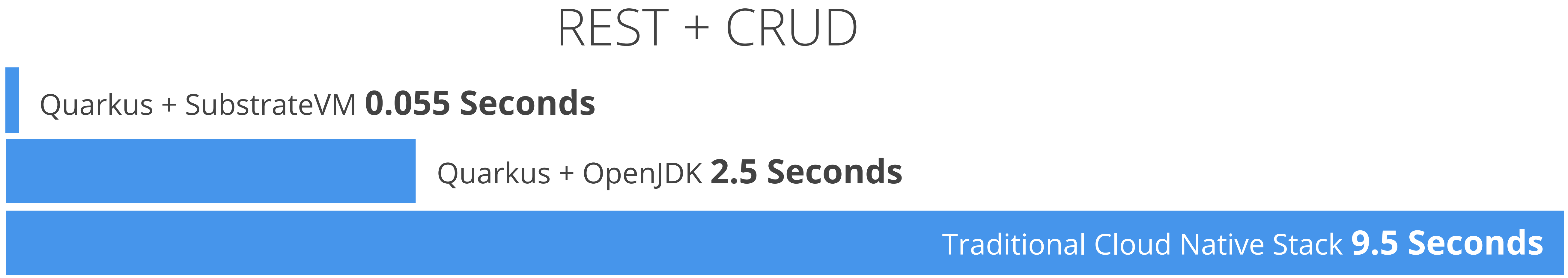

Java
Ruby
JavaScript
Sulong (LLVM)
Truffle
Graal Compiler
JVM CI
Substrate VM
Java HotSpot VM

Supersonic Subatomic Java



Supersonic Subatomic Java





Time to first response

Eclipse Vert.x

- Non-blocking (asynchronous) and non locking (concurrency is natural)
- Event bus (reactor pattern)
- Load balancing, failover, circuit breaker
- Clustering and Service Discovery
- Polyglot JVM
- Infinispan/JDG and Spring Boot
- Added AMQ and Qpid Dispatch Router
- TCP, UDP, HTTP 1 & 2 servers and clients (non-blocking), gRPC

Narayana STM

- Provides an STM implementation
- Define state (objects) which can be manipulated within transactions
    - Volatile (recoverable) and persistent (durable)
- Pessimistic and Optimistic models
- Different variants of transactions
    - Top level
    - Nested
    - Nested top level
- Modularity

STM Object Annotations

- @Transactional
  - Implementations of the interface then become container managed
- @Nested & @NestedTopLevel
  - The container will create a new transaction for each method
- @Optimistic & @Pessimistic (with @Timeout and @Retry options)
- @ReadLock & @WriteLock
- @State & @NotState
- @TransactionFree

OpenShift

App	Svc	Svc
Runtime	Runtime	Runtime

Java Compatible Enterprise Edition · spring by Pivotal · VERT.X · node js

Data	Security	IMDG	Messaging

Cloud Platform

Build | Deploy | Scheduling | Scaling | Elasticity | Metrics | Logging

OPENSHIFT by Red Hat

Cloud Provider

Microsoft Azure · amazon web services

Code demonstration

An STM example running on Quarkus

Three versions of an application running on Quarkus:

1. A standard async JAX-RS app
   a. No concurrency control
   b. Reactive/async support via Vert.x
   c. Stress test to show concurrency issues
2. Same app using volatile STM for concurrency control
   a. Make the service interface a volatile STM object and restress
3. Deploy onto Openshift
   a. Turn it into a shared persistent STM object and deploy to OpenShift
   b. Create a shared persistent volume for the STM logs
   c. Test and then scale up and down via the OpenShift console to show persistency

Where can I find out more?

- More information available from https://narayana.io:
  - Forums
  - Blogs
  - IRC
- Demo source
  - https://github.com/narayana/conferences/tree/master/codeone-2019
- STM source
  - https://github.com/jbosstm/narayana/tree/master/STM

THANK YOU
- any questions?

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos