

Transactional Actors using STM and Eclipse Vert.x

Michael Musgrove, Red Hat

28th Jan, 2018

Agenda

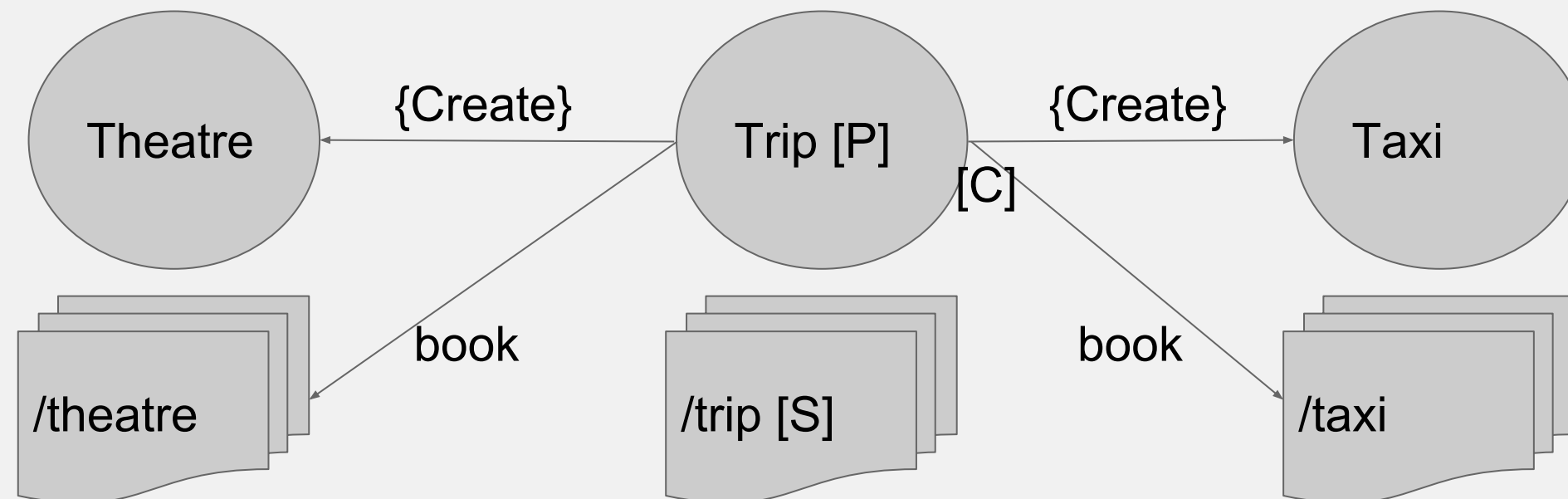
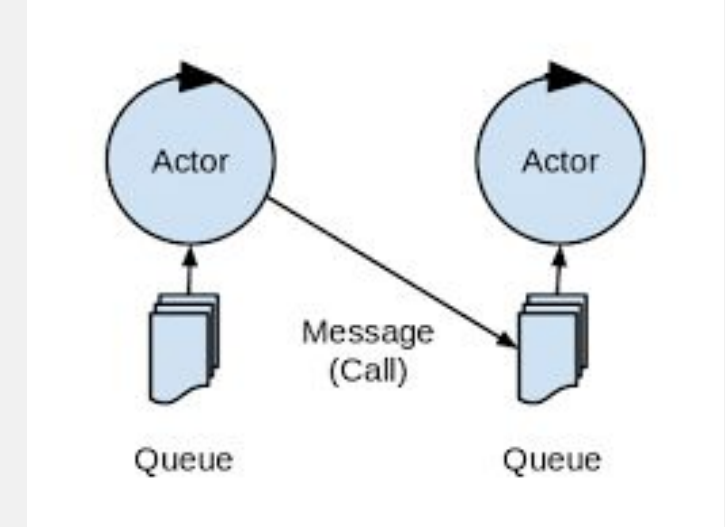
- The Actor Model
- Distributed Transactions and Software Transactional Memory (STM)
- Live coding demo

What this talk is NOT

- Not a survey of actors
- Not a survey of transactions
- Not a tutorial on Vert.x
- Not a tutorial on Narayana
- Not a tutorial on OpenShift

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 [P]rocessing, [S]torage and [C]ommunication
- Distributed computations communicate via message passing



Nice Features of the Actor Model

- Fixed message sets (ie no hidden or unexpected interactions)
- Simplified data management (an actors state is internal to the actor)
- Location transparency (because other actors only see the address)
- Loose coupling
- Asynchronous message passing brings better scalability and resource utilisation
- Simplicity of the model means that potentially millions of them can be created (cf threads)
- Has much in common with the reactive manifesto (Responsive, Elastic, Asynchronous and, with extensions, Resilience).

Distributed Transactions

- ACID properties
 - Atomicity
 - Consistency
 - Isolation
 - Durability
- 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
 - Still an active area of research
- an approach to developing transactional applications in a highly concurrent environment:
- 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
 - typically though, the Durability property is relaxed;

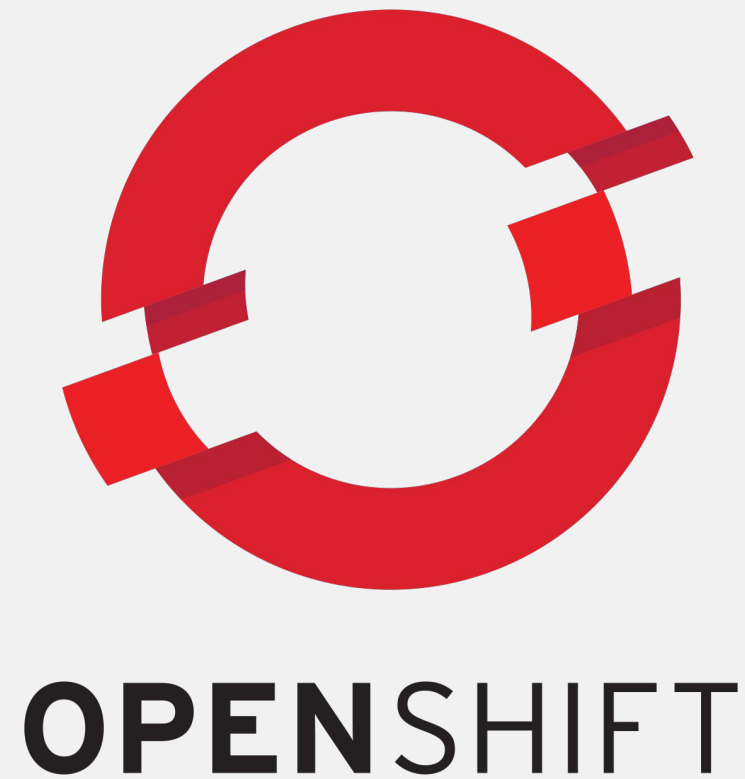
Characteristics of STM

- Isolation
- Granularity
- Concurrency Control
 - Conflict Detection
 - Conflict Resolution
- Nested Transactions
- Exceptions

Transactions and Actors

- A stateful actor may go through multiple state transitions upon receipt of a message
- An actor could be internally implemented using multiple threads (as we show in the demo later)
- 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

Notable technology used in the demo

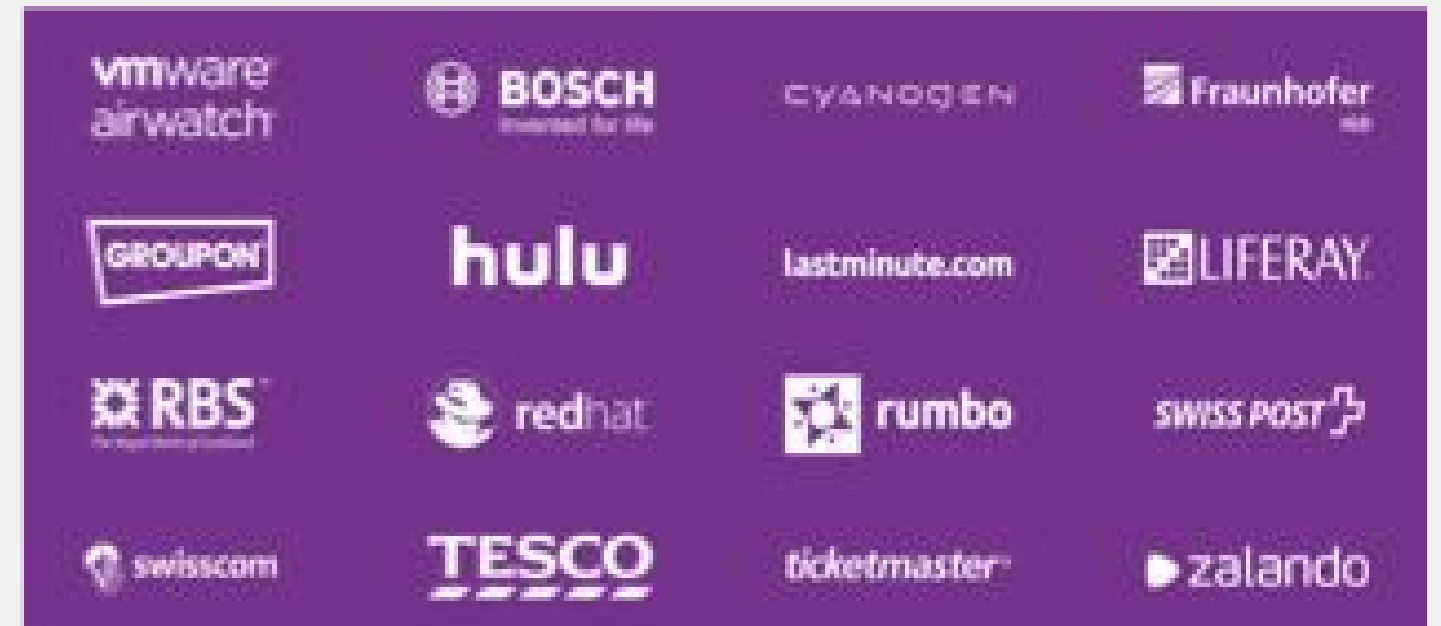


VERT.X

@narayana

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
- Adding transactions and STM
- Large user base and community of contributors



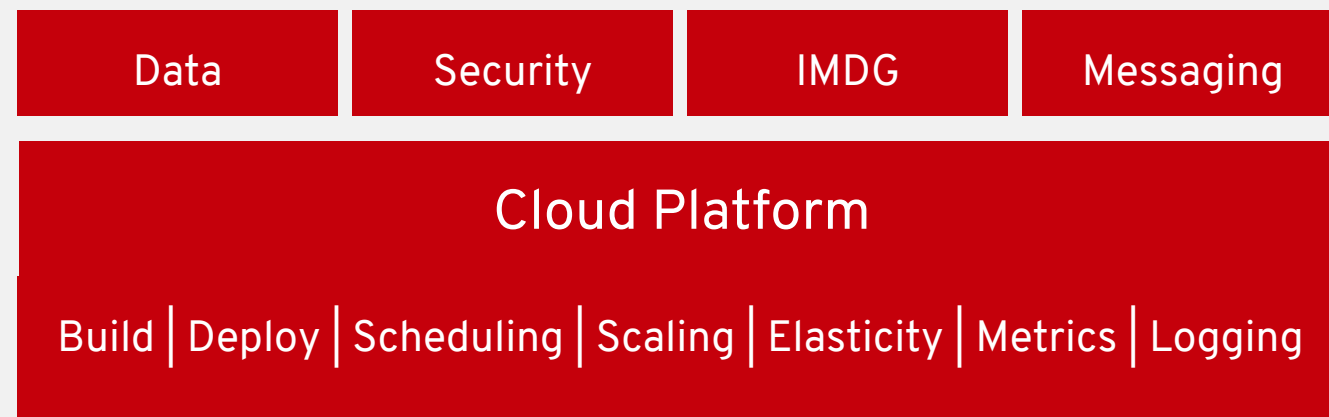
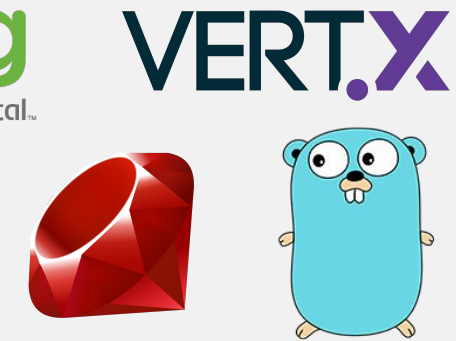
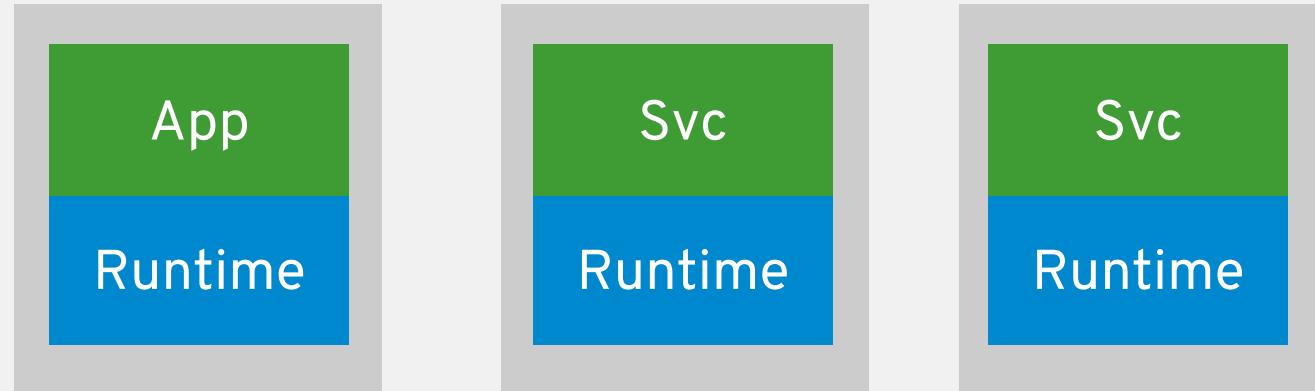
Narayana STM

- 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
 - Transaction context on threads
- Annotations

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



Microsoft Azure



Live coding demo

An STM Example: Create a new Vert.x Project

- Use the vertx plugin to create a new maven project

```
mvn io.fabric8:vertx-maven-plugin:1.0.7:setup -DvertxVersion=3.4.2 -Ddependencies=web  
Enter the GAV coordinates + verticle class name
```

- Fill in the verticle's start method
- Start the verticle: `mvn compile vertx:run`
- Test it: `curl -X POST http://localhost:8080/api`
- Add a service interface and implementation and main to deploy the verticle
- Stress test the service
- Make the service interface a volatile STM object and restress
- Turn it into a shared persistent STM object and deploy to OpenShift
- Test and then scale up and down via the OpenShift console to show persistency

Running on OpenShift

- > minishift start --vm-driver=virtualbox # or whatever hypervisor you are using
- > minishift console # opens the OpenShift web console
- > oc login -u developer -p developer
- > oc new-project stmdemo
- # Create a PVC via the OpenShift console ("stm-vertx-demo-flight-logs", RWX access)
- > mvn fabric8:deploy -Popenshift # use S2I to build and deploy the image
- > oc get routes # to find the host/port
- > curl -X POST <http://stm-vertx-demo-flight-stmdemo.192.168.99.100.nip.io/api>
- > Show sharing by scaling up (hostId field). Scale to zero and back up to show persistence.

Where can I find out more?

- More information available from narayana.io
 - Forums
 - Blogs
 - IRC
- Demo source
 - <https://github.com/mmusgrov/conferences/tree/master/devconf-brno-2018>
- 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