A step-by-step guide from traditional Java EE to reactive microservice design

Ondrej Mihályi

@OndroMih https://ondro.inginea.eu

Ondro Mihályi

Web:
ondro.inginea.eu
www.payara.fish

Twitter:
@OMihalyi

- Engineer at Payara
- Java EE developer, lecturer
- MicroProfile member
- Czech JUG leader

Agenda

- Reactive improvements
- Separation to micro-svc
- Deploy with Docker

https://github.com/OndrejM-demonstrations/ReactiveWay-cargotracker-ext

Majority of the enterprise code

Request started

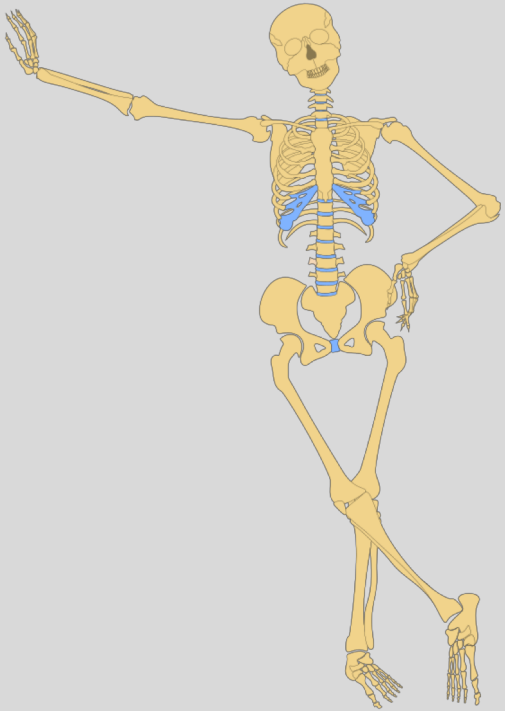
→ resource required (DB, WS, files)

→ request the resource

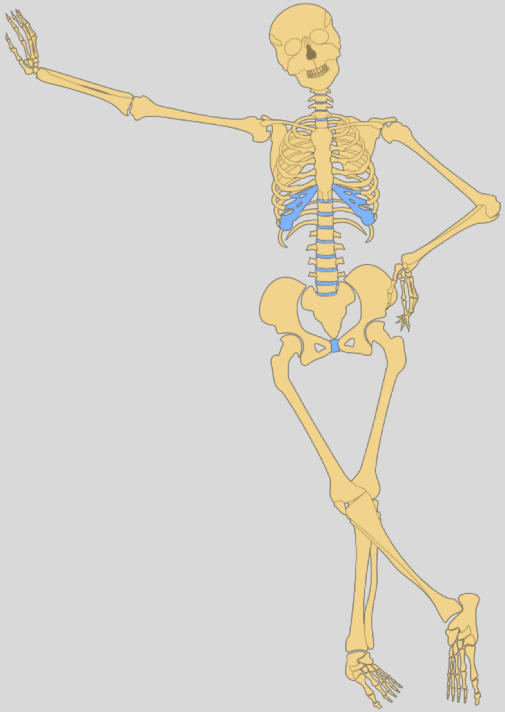
→ the resource is slow

WHAT DO WE DO? **(hint: it's quite natural)**

We simply...



We simply...



...WAIT

- No shame, we all do it
  - Simple
  - Thread-safe
  - Enough in most cases

But what if...

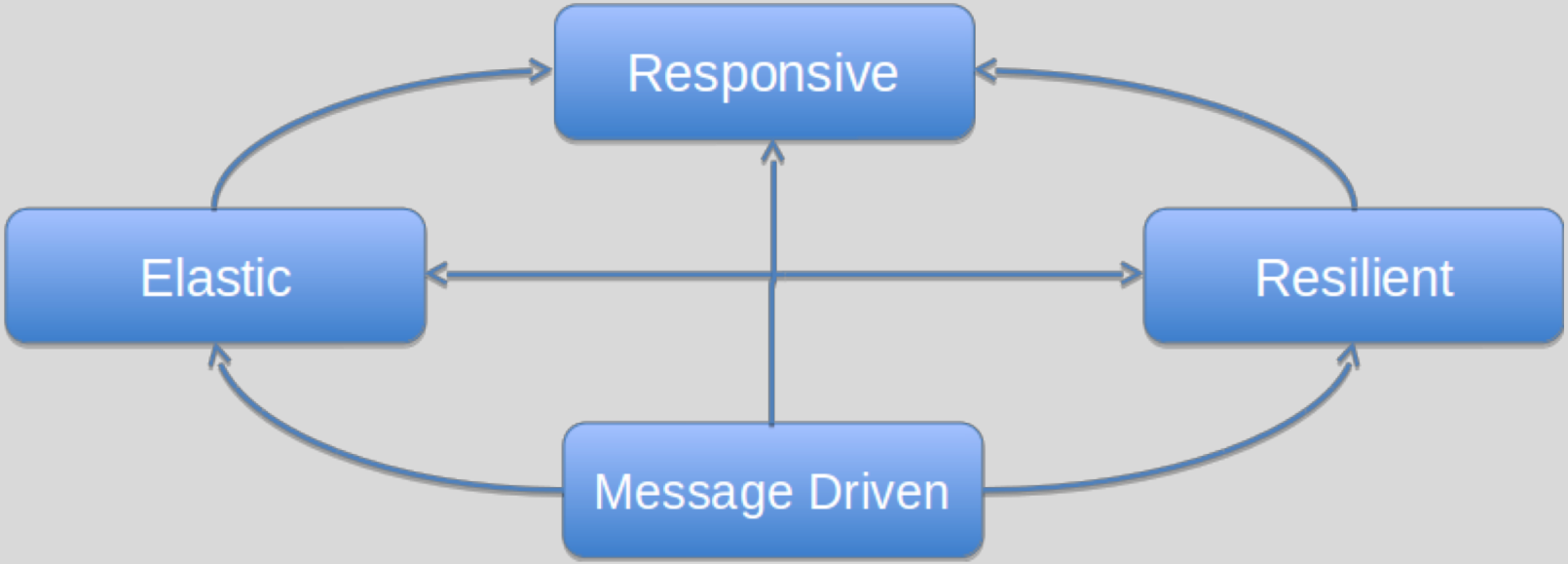
If waiting is too long...

- Thread resources are **wasted**
- Processing may **slow** down or **halt** if all threads waiting
- Users don’t receive a timely **feedback**
- Failures lead to more **failures**

If waiting is too long...

- Thread resources are **wasted**
- Processing may **slow** down or **halt** if all threads waiting
- Users don’t receive a timely **feedback**
- Failures lead to more **failures**

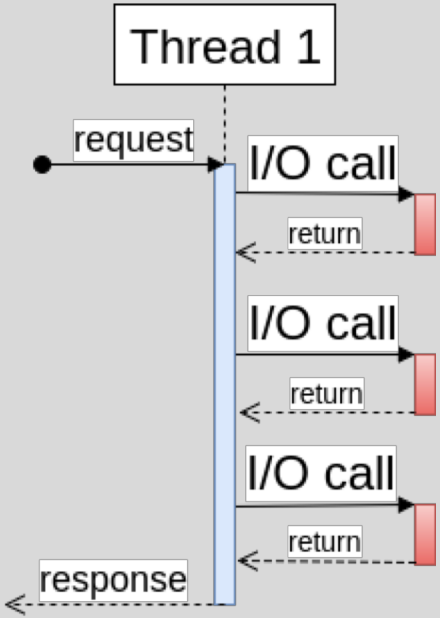
Reactive manifesto



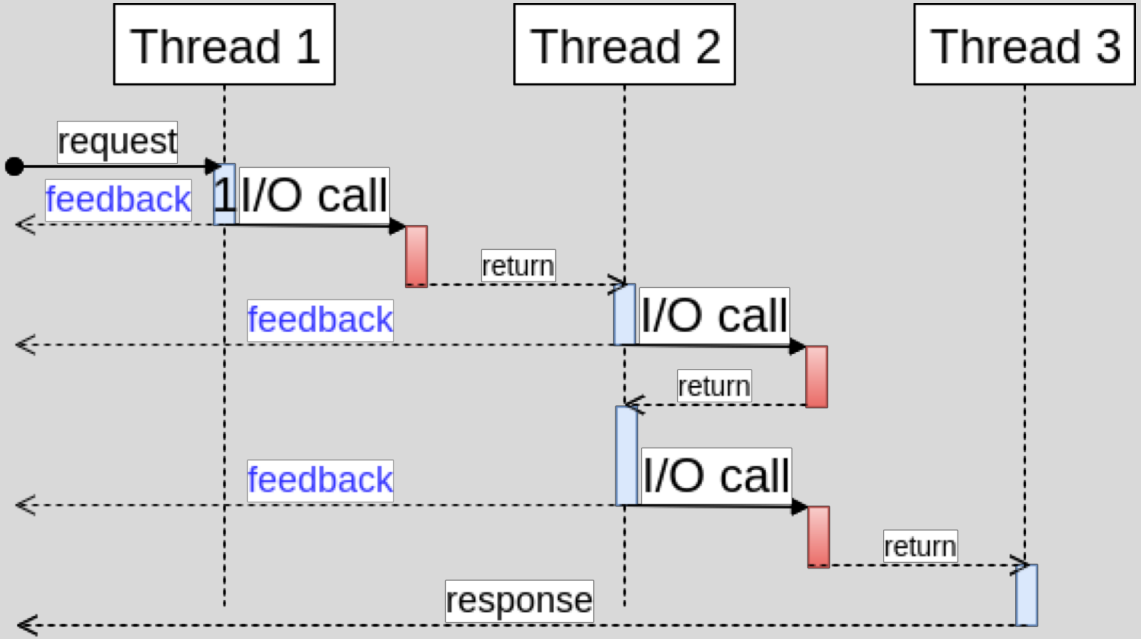
http://www.reactivemanifesto.org/

Reactive idea in a nutshell

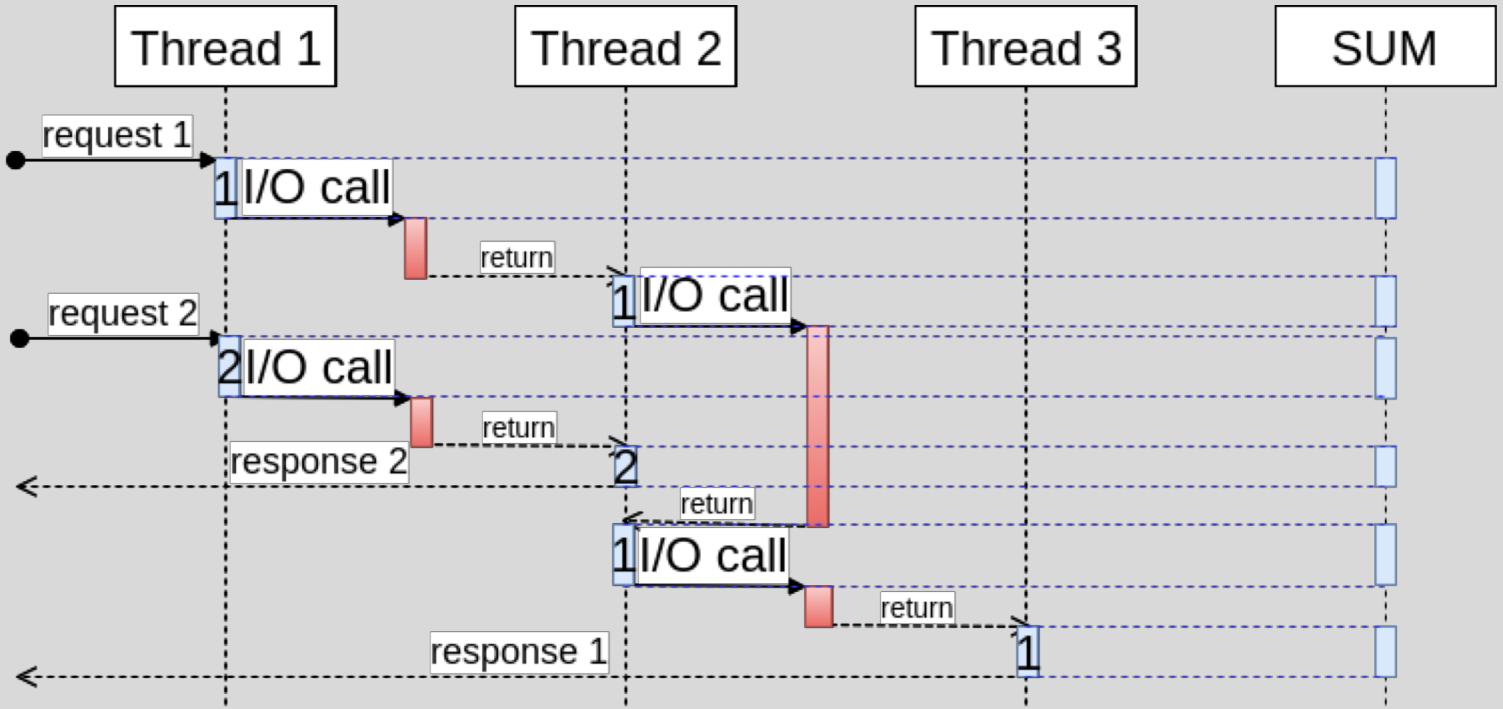
Single-threaded



Reactive - not bound to a thread



Threads reused between calls



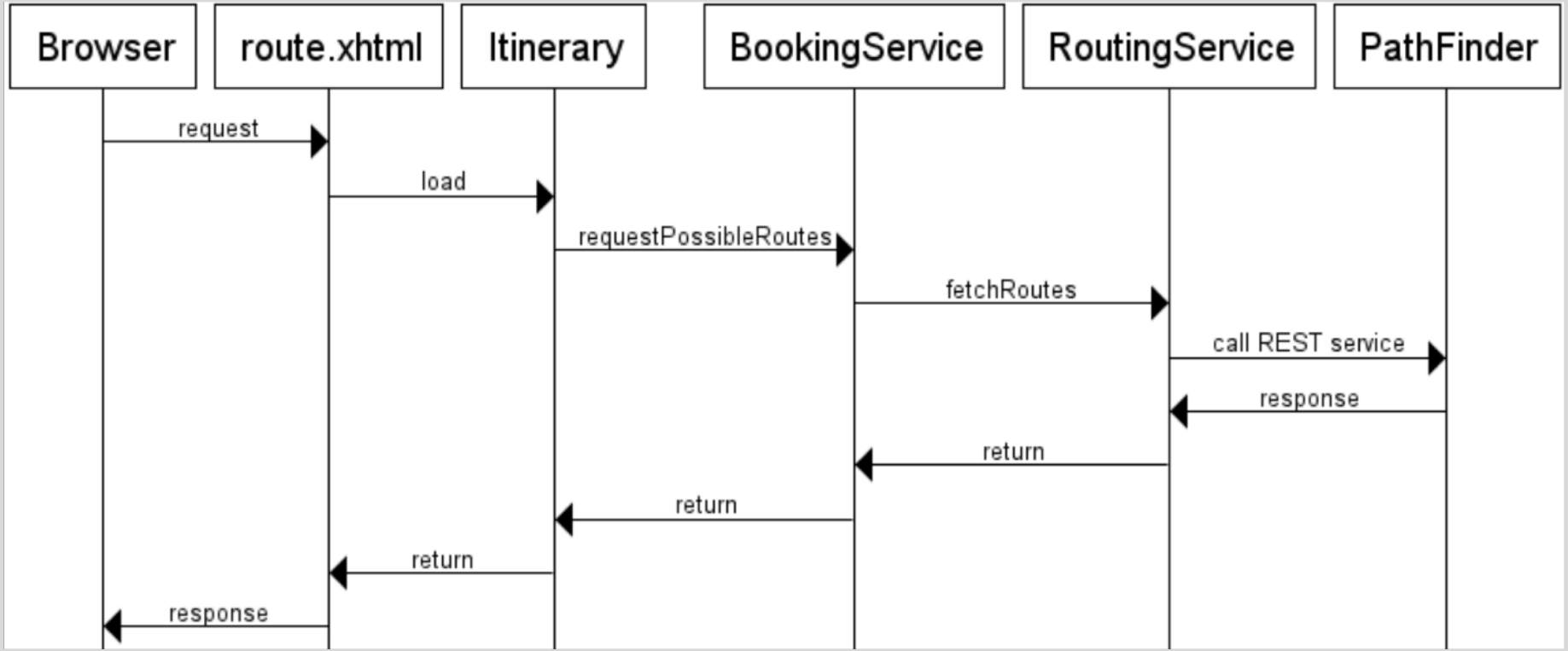
Example traditional application

We’re going to improve it reactively

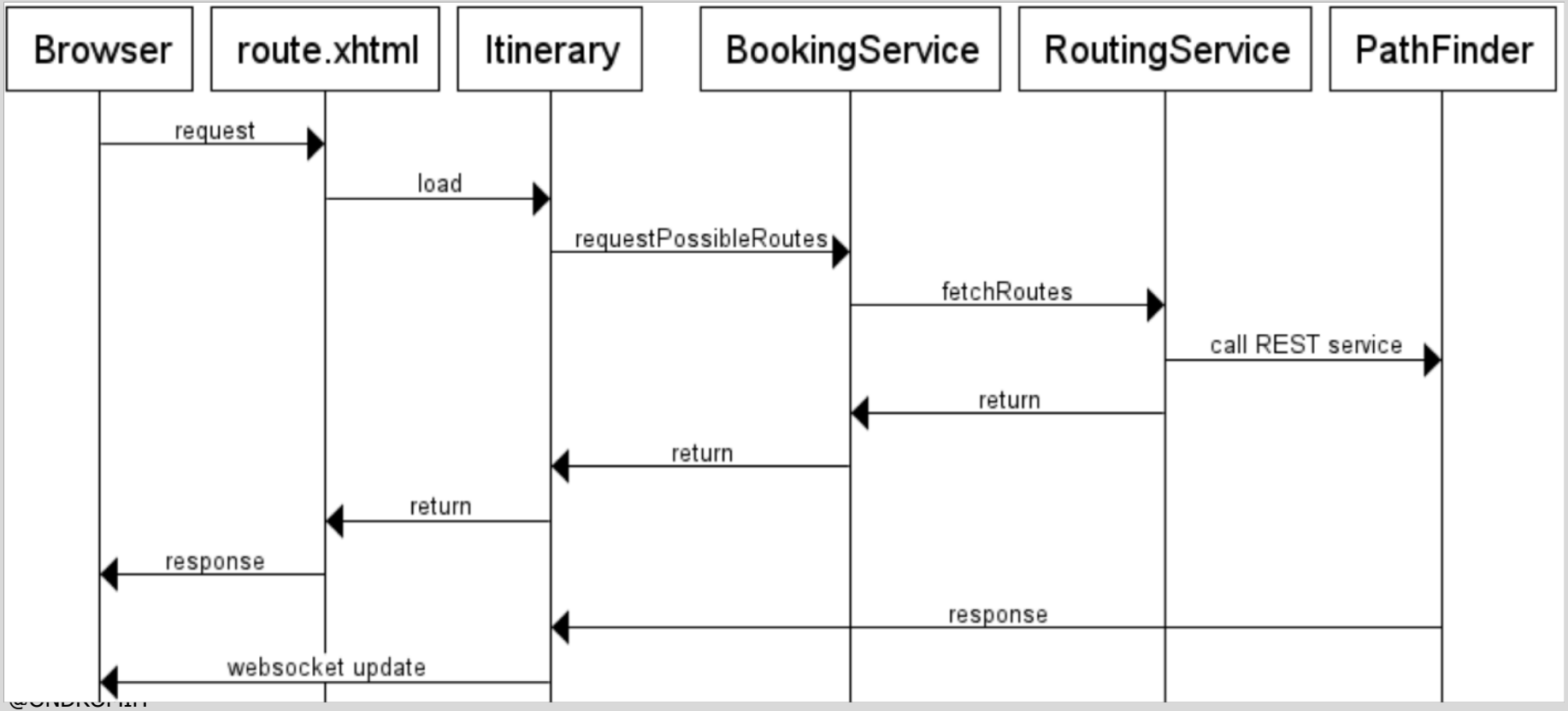
Original project:
https://github.com/javaee/cargotracker

Reactive improvements:
https://github.com/OndrejM-demonstrations/ReactiveWay-cargotracker-ext

Traditional design



More reactive design



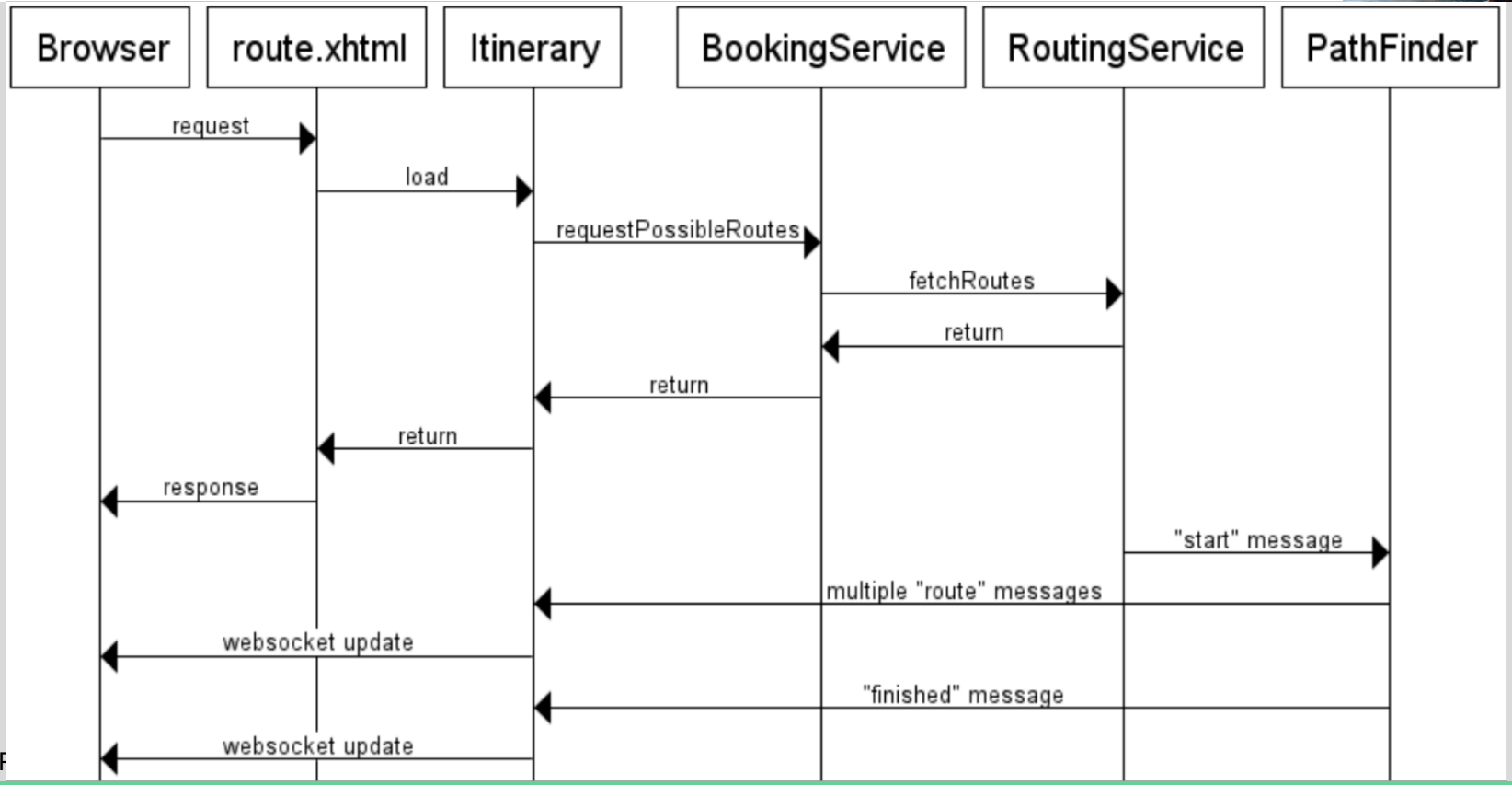
Chaining callbacks

- CompletableFuture in Java 8
  - complete, completeExceptionally
  - thenAccept, thenApply, thenCompose
  - exceptionally
- Alternatively use third-party libraries
  - RxJava observables

3. Messages

- WebSocket, Server-Sent Events
- CDI events
  - Light-weight but extensible API
  - Events don’t cross JVM boundaries
  - Can we turn them into messages?

Even more reactive design



Simple messaging and caching

```
@Inject @Outbound

Event<MyMsg> ev;

/* handle in the same
or a different JVM: */

void handle(@Observes

@Inbound MyMsg ev) {

 … }
```

Payara **CDI Event Bus**

- Async events on all nodes
- On top of CDI events
- Uses Hazelcast

JCache API

- Scalable distributed memory
- Replication and failover

Reactive programming isn’t easy

- Multiple threads, need to track origin and arrival of responses, errors may be unnoticed, new design patterns
- Don’t over-engineer, but leave space for future improvements
  - Java EE and MicroProfile allow for gradual improvements

Thank you!