Efficient communication with QUARKUS and gRPC

Michał Szynkiewicz, Quarkus team
@mszynkiewicz

Who am I

- Writing Java for living since 2007
- gRPC and REST Clients in Quarkus
- SmallRye Stork

In the past, e.g.

- MicroProfile/SmallRye Fault Tolerance
- PayU, Pointpack (parcels in Żabka)

Who are you?

- Have you tried Quarkus?

QUARKUS

Who are you?

- Who has tried Quarkus or has seen any presentations on Quarkus?

QUARKUS

Who are you?

- Have you worked on a production system using Quarkus?

QUARKUS

Long long time ago, there was SOAP

SOAP

- From 1998
- Text-based, through HTTP
- Easy integration with services that use different technologies/languages
- Standardized description language/schema

SOAP message format

- XML
- Not really human-readable
- Not human-writeable

```
<?xml version="1.0"?>

<soap:Envelope
       xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding" >

   <soap:Header>
       ...
   </soap:Header>

   <soap:Body>
       ...
       <soap:Fault>
           ...
       </soap:Fault>
   </soap:Body>

</soap:Envelope>
```

SOAP WSDL

```xml
<definitions name = "HelloService"
             targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
             xmlns = "http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
             xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>

   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name = "Hello_Binding" type = "tns:Hello_PortType">
      <soap:binding style = "rpc"
                    transport = "http://schemas.xmlsoap.org/soap/http"/>
      <operation name = "sayHello">
         <soap:operation soapAction = "sayHello"/>
         <input>
            <soap:body
                   encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                   namespace = "urn:examples:helloservice"
                   use = "encoded"/>
         </input>

         <output>
            <soap:body
                   encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
                   namespace = "urn:examples:helloservice"
                   use = "encoded"/>
         </output>
      </operation>
   </binding>

   <service name = "Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding = "tns:Hello_Binding" name = "Hello_Port">
         <soap:address
```

SOAP

- Inefficient
- Using only POST method
- Difficult testing

Then came REST

REST - Representational State Transfer

- REST is easy!
- Text-based
- Over HTTP
- Using HTTP methods

REST+XML

- Human readable and writable
- Simple
- Less boilerplate

REST+JSON

- Compact messages
- Relatively fast ser- and deserialization

```
{
 "message": "Hello, World!"
}
```

REST (+JSON)

- Easily testable
- Plays well with Web technologies
- Standardized API description - OpenApi (FKA Swagger)

REST is awesome!

- And that’s why I work on the Quarkus REST clients :)

REST is awesome!

- But has some inconveniences

REST+JSON can perform well

- Especially with Quarkus and RESTEasy Reactive

REST+JSON can perform well but

- Well thought binary serialization and deserialization should perform better than text format
- It’s mostly request-response, a lot of HTTP boilerplate on each message

Not all operations fit HTTP methods

- What if you need to triggering report generation

REST doesn’t do streaming (much)

- Okay, okay, there are Server Sent Events
- But no client-side streaming at all
- Alternative: WebSockets
  - No (widely used) standard for passing messages, etc
  - No defined error handling

REST is Code-first

- OpenApi is readable, not very writeable

```
paths:
 /hello:
   get:
     tags:
     - Reactive Greeting Resource
     responses:
       "200":
         description: OK
         content:
           text/plain:
             schema:
               type: string
```

Here comes gRPC

gRPC: better serialization performance

- Binary format
- Protocol Buffers for serialization and deserialization
  - More efficient CPU-wise (mobile friendly)
  - More efficient size-wise

gRPC uses HTTP/2

- Still network/router friendly
- Faster on the wire

gRPC: a standard

- A CNCF incubating project
- Open source, Apache Software License 2

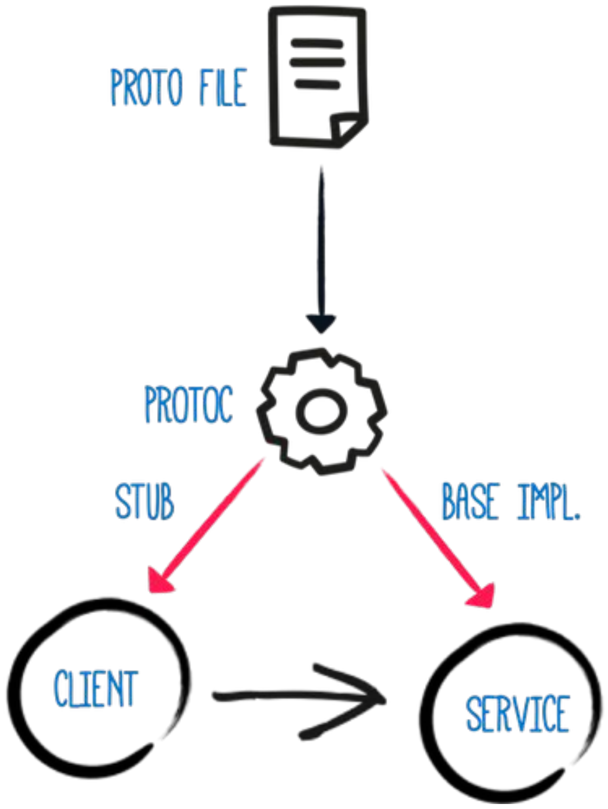
gRPC: a standard

- Used by Google, Netflix, and many, many other companies
    - E.g. seems to be the backbone of Lightbend’s Kalix

gRPC: a standard

- Even though it’s binary, you can use it with most (sane) languages you can think of
    - Java, Rust, Go, JavaScript, PHP, Python, …

gRPC is contract-first



gRPC has a simple concise contract

```
service Hello {
   rpc SayHello (HelloRequest) returns (HelloReply);
}
message HelloRequest {
   string name = 1;
}
message HelloReply {
   string message = 1;
}
```

gRPC: contract is backward and forward compatible

```
service Hello {
   rpc SayHello (HelloRequest) returns (HelloReply);
}
message HelloRequest {
   string name = 1;    int32 age = 2;
}
message HelloReply {
   string message = 1;
}
```

gRPC: remote procedure call

- More like SOAP than REST
  - More freedom/power
  - Less standardized API

gRPC communication flavors

- Request-response
- Server-side streaming
- Client-side streaming
- Bidirectional streaming

gRPC

- Metadata attached to requests and responses
  - E.g. security headers

gRPC: Standardized error handling

- Errors translated to Java Exceptions
- Java exceptions translated to errors

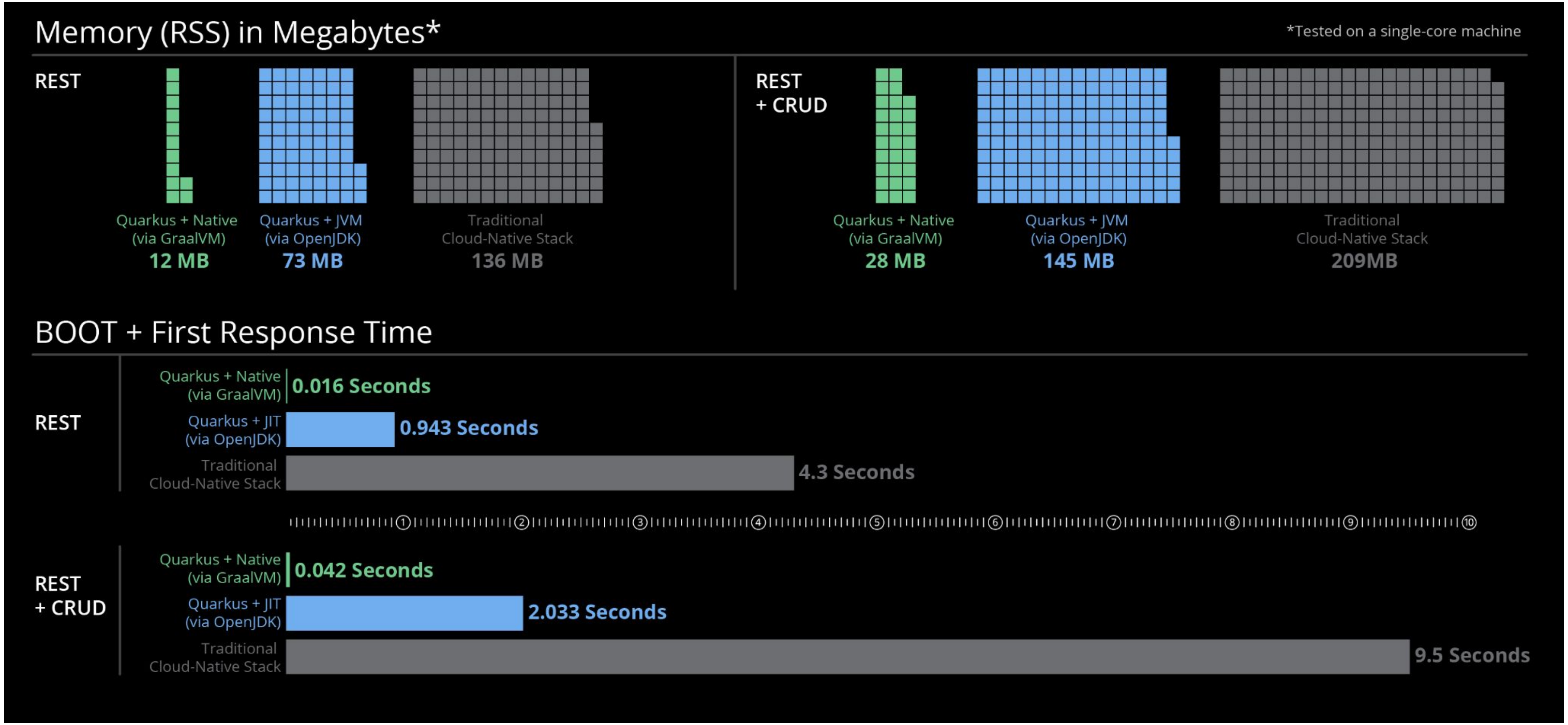
Quarkus

AMQP
Liquibase
Hibernate
Kotlin
Keycloak
Qute templating
Elasticsearch
Hibernate Validator
MicroProfile
OIDC
Mutiny
Reactive
AWS
GraphQL
GCP
Micrometer
Kafka
Narayana
SmallRye Stork
Kind
Kogito
gRPC
Azure
GraalVM
OpenApi
Redis
Spring APIs
Hazelcast
Vert.x
RESTEasy
Kubernetes
Cassandra
OpenTracing
Prometheus
Infinispan
Camel
REST Client
Jackson
RabbitMQ
MQTT
Jacoco
OpenShift
MongoDB
Picocli
SmallRye Fault Tolerance
Quartz
WebSockets
Flyway

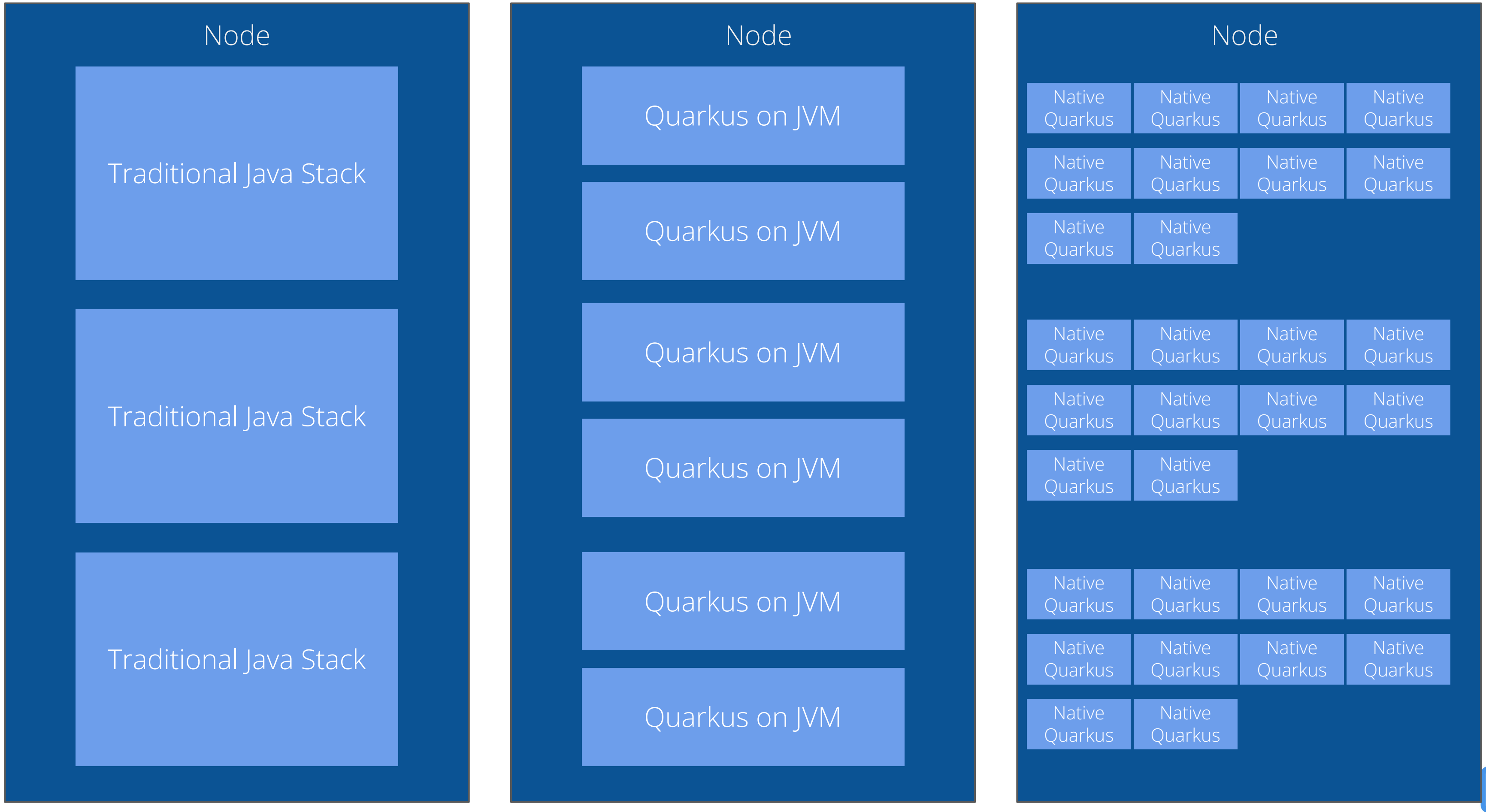
More:
https://code.quarkus.io

Two major features

QUARKUS



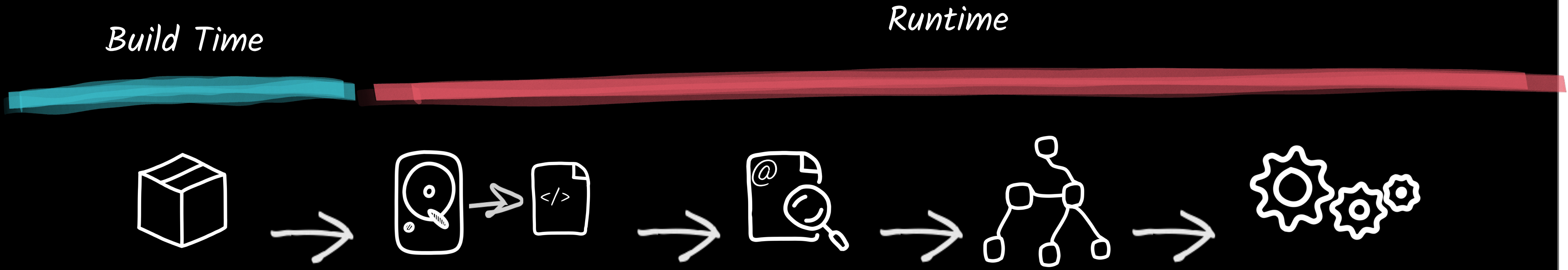
Disclaimer: the numbers will differ depending on the Quarkus and GraalVM versions.


Node
Traditional Java Stack
Traditional Java Stack
Traditional Java Stack
Node
Quarkus on JVM
Quarkus on JVM
Quarkus on JVM
Quarkus on JVM
Quarkus on JVM
Quarkus on JVM
Node
Native Quarkus

QUARKUS

But how?

How Does a Framework Start?



Packaging (maven, gradle)

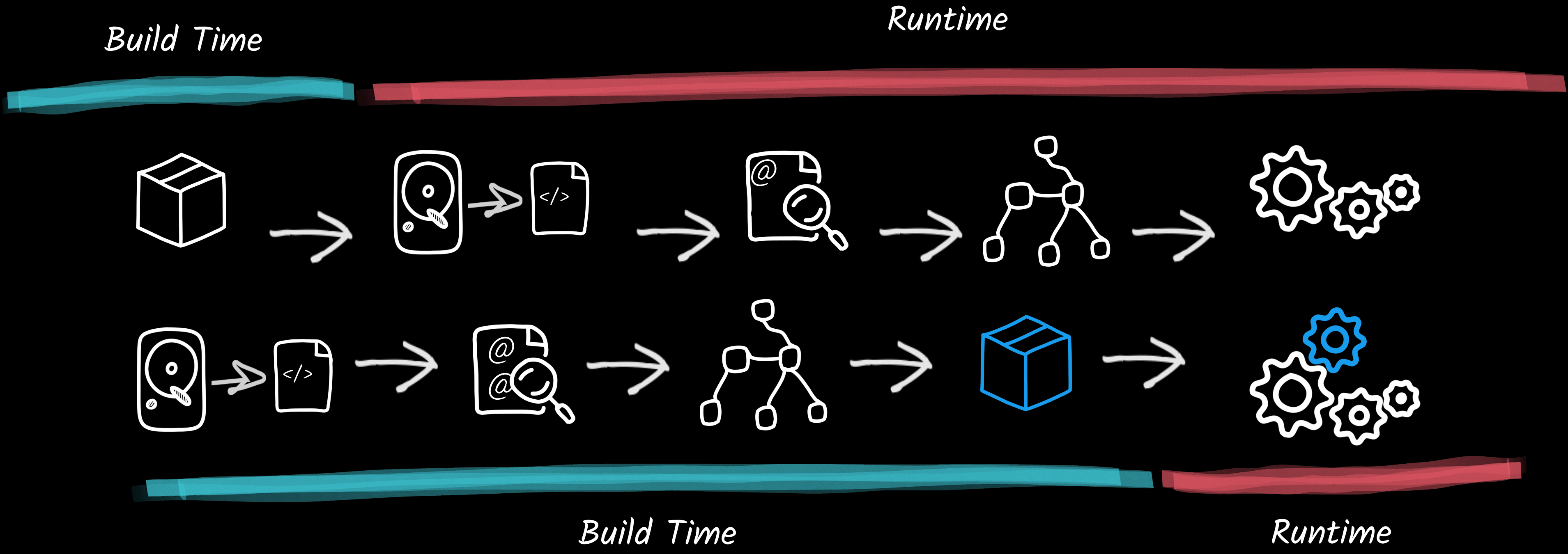
Load config file from file system Parse it

Classpath scanning to find annotated classes Attempt to load class to enable/disable features

Build its model of the world.

Start the management (thread, pool...)

The Quarkus Way



Quarkus - a great fit for native compilation

- Need registration
  - Reflection
  - Dynamic proxies
- Not supported
  - Dynamic classloading
  - Method handles
  - Invoke dynamic
- Different
  - Static initialization (can be done) on build-time

So performance, and

Developer Joy

- Dev reload
  - Just code and test, no restart needed
    - Code, config and more
  - Scripting language like experience

Developer Joy

- Dev Services
    - Automatically spinned up containers for:
        - Databases
        - Keycloak
        - Kafka
        - And many more
    - Also works for tests

QUARKUS

Developer Joy

- Dev UI
  - Look under the hood of your application
  - Adjust configuration
  - Play with your application

QUARKUS

Developer Joy

- Continuous testing

QUARKUS

Quarkus + gRPC

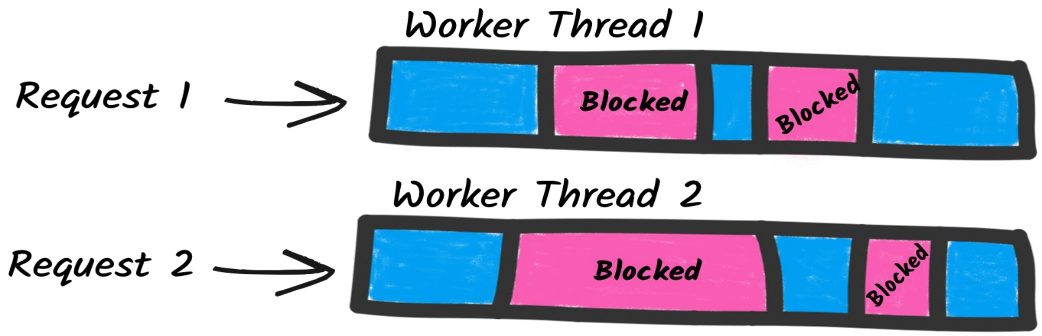
Let’s start with some code

- Code starts
  - Proto
  - Interface
  - Tests

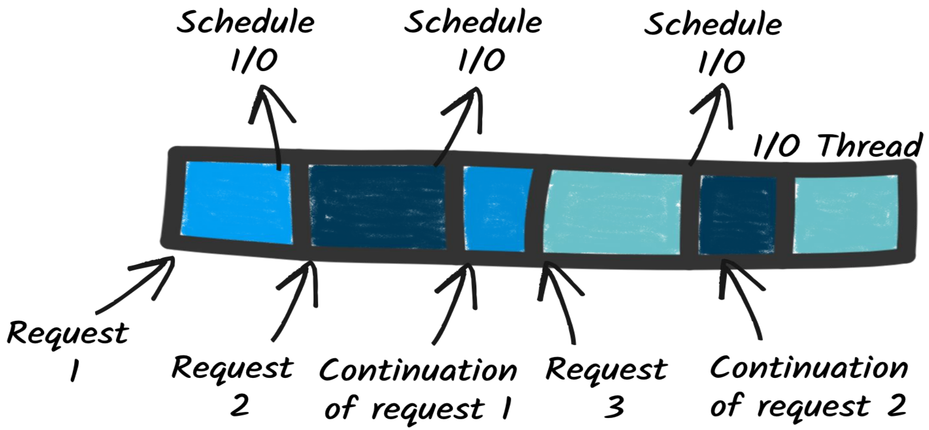
QUARKUS

Before we go further: reactive programming

BLOCKING

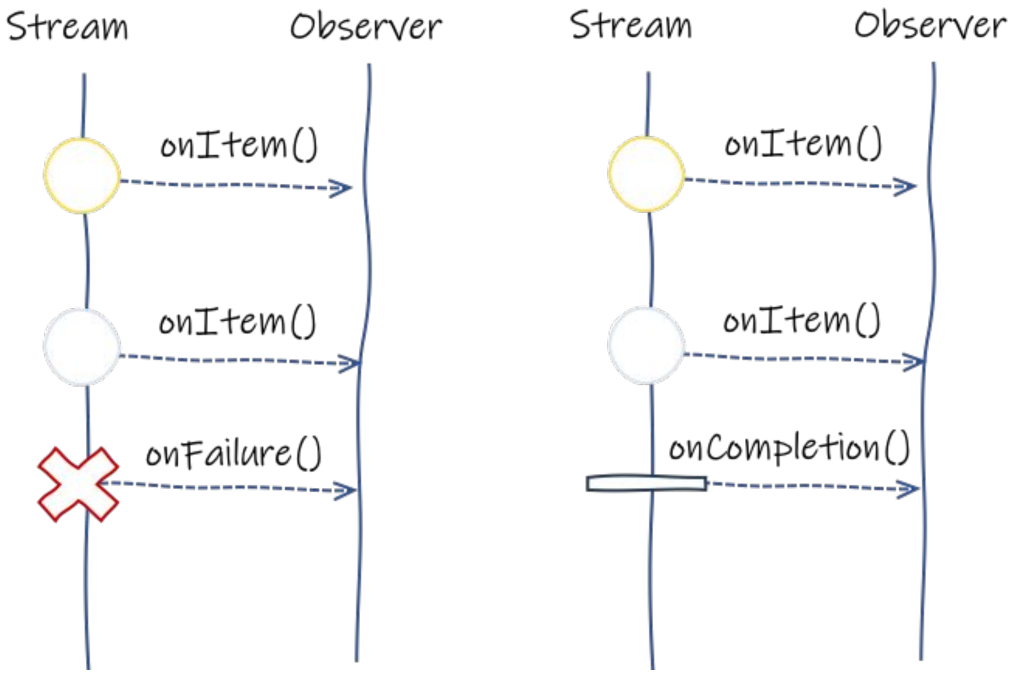


EVENT LOOP



QUARKUS

Before we go further: reactive programming



Before we go further: reactive programming

- Multi
  - 0..* items (potentially unbounded)
  - and/or failure
- Uni
  - Item or failure

MUTINY!

Before we go further: reactive programming

```
Multi.createFrom().items("a", "b", "c")
  .onItem().transform(String::toUpperCase)
  .subscribe().with(
    item -> System.out.println("Received: " + item),
    failure -> System.out.println("Failed with " + failure)
);

Uni.createFrom().item("a")
  .onItem().transform(String::toUpperCase)
  .subscribe().with(
    item -> System.out.println("Received: " + item),
    failure -> System.out.println("Failed with " + failure)
);
```

MUTINY!

Let's get back to code

- Quiz service

QUARKUS

Developer experience

- Dev UI:
  - Looking into the internals of the application
  - Playing with your gRPC services manually
    - Similar with GraphQL
  - And more

QUARKUS

Developer experience

- Integrated code generation
  - With dev reload

Developer experience

- Preconfigured @GrpcClient for tests

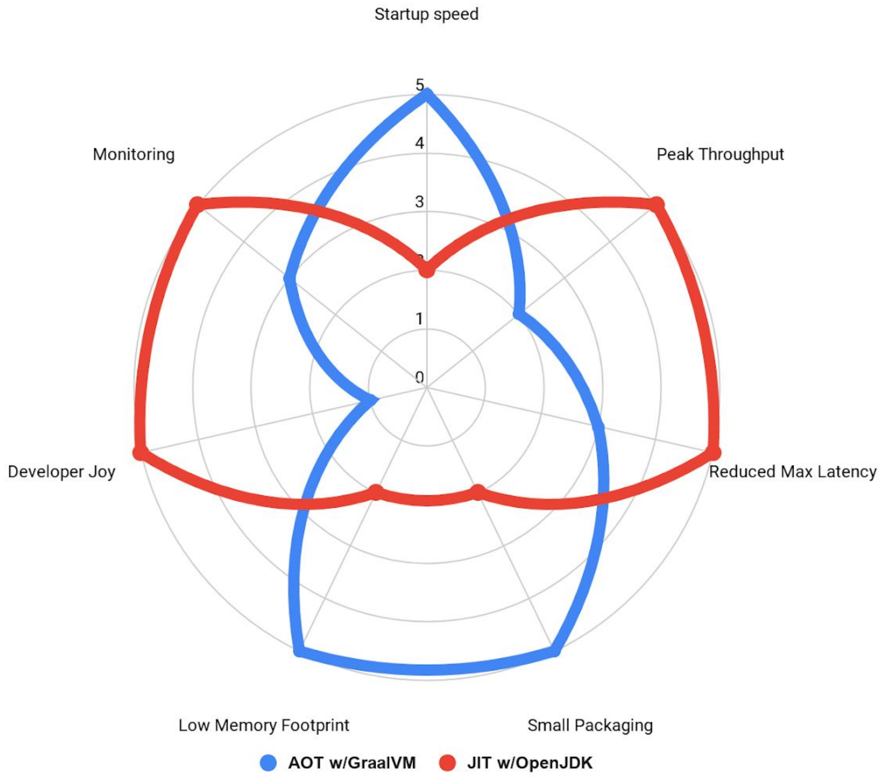
QUARKUS

Let’s play!

QUARKUS

Questions?

When to use native compilation?



QUARKUS

Links

- Zulip: https://quarkusio.zulipchat.com/
- GitHub Discussions: https://github.com/quarkusio/quarkus/discussions
- Quarkus page: https://quarkus.io
  - Code generator
  - Guides
- My twitter: @mszynkiewicz

QUARKUS

Thank you!