Quarkus Concurrency In Practice

Mike Hepburn

Programming models

Characteristic	Imperative approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.
State changes	Important.
Order of execution	Important.
Primary flow control	Loops, conditionals, and function (method) calls.
Primary manipulation unit	Instances of structures or classes.

everyday code ... ah yup ...

Imperative - a developer writes code that describes in exacting detail the steps that the computer must take to accomplish the goal. Order of execution easy to follow as it follows the lines of code. Synchronous.

think - Java, C++, C#

Enter the Coffee shop, Queue, Order, Pay, Get Coffee

Reactive - based on events. Asynchronous, the bit of code in control is called back when processing is done. Order of execution does not rely on ordered lines of code.

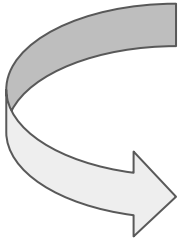
think - Javascript, RxJava

Order a Coffee. Go do something else. Hey - your Order is ready !

Imperative (sync) vs Reactive (async)

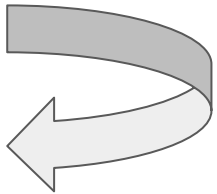
CompletableFuture in Java == Promises in Javascript

Traditional - Imperative - Synchronous
Blocking I/O



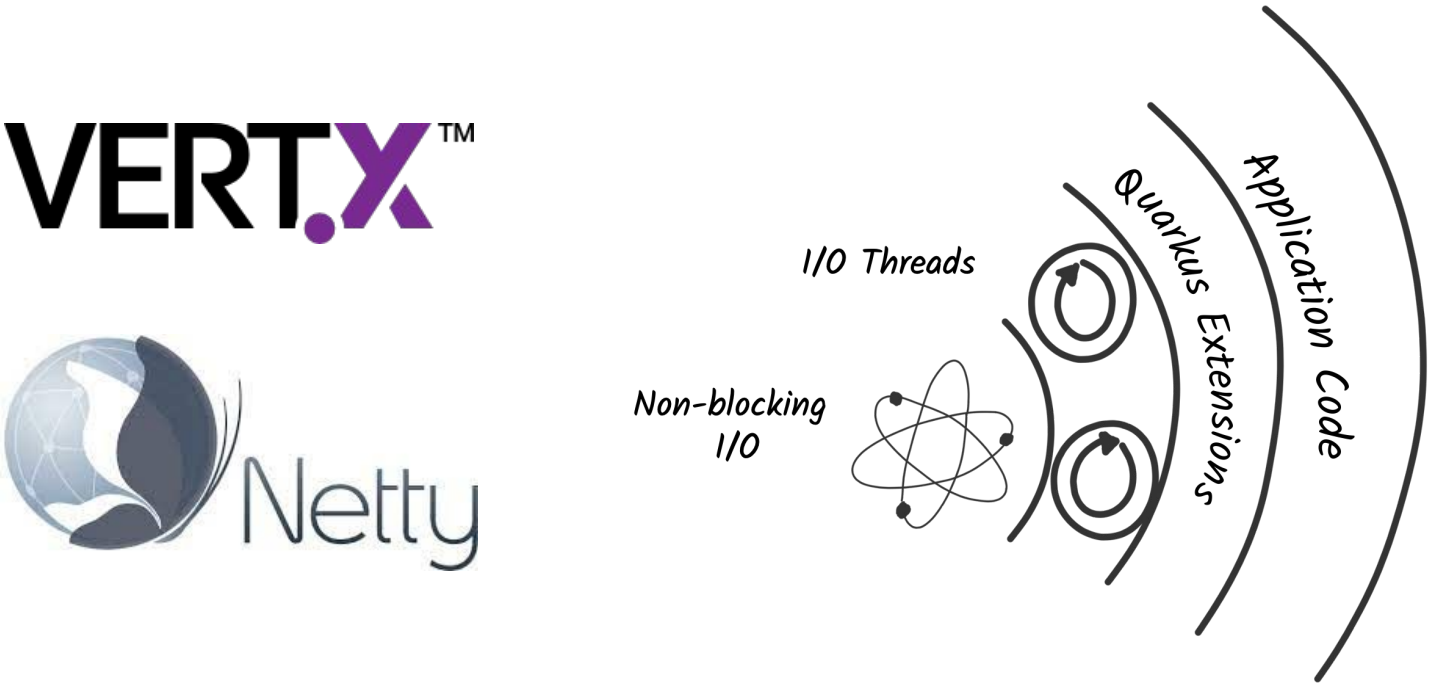
```java
@Path("/hello")
public class ReactiveGreetingResource {

    @GET
    @Path("imperative")
    @Produces(MediaType.TEXT_PLAIN)
    public String imperativeHello(@PathParam("name") String name) {
        System.out.println(Thread.currentThread().getName());
        return greeting(name);
    }

    @GET
    @Path("reactive")
    @Produces(MediaType.TEXT_PLAIN)
    public CompletionStage<String> reactiveHello(@PathParam("name") String name) {
        System.out.println(Thread.currentThread().getName());
        return CompletableFuture.supplyAsync(() -> greeting(name)).minimalCompletionStage();
    }

    private String greeting (String name) {
        return "Hello " + (null == name || name.isBlank() ? "World !" : name);
    }
}
```

Reactive - Asynchronous
Non-Blocking I/O



I/O - What does this mean though ? How are Threads (Parallel Processing) handled differently ?

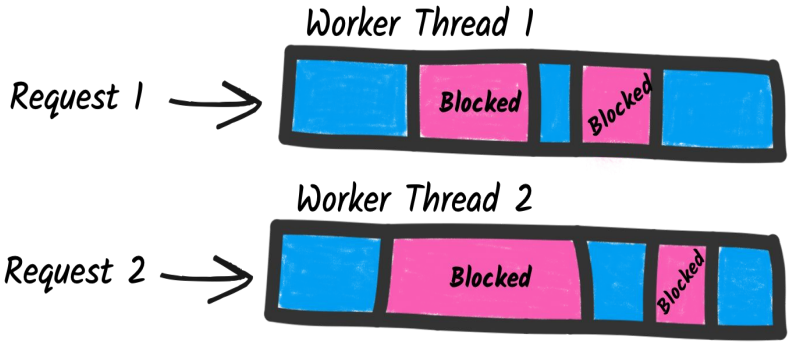
Under The Hood of Quarkus - Vertx.io, Netty.io



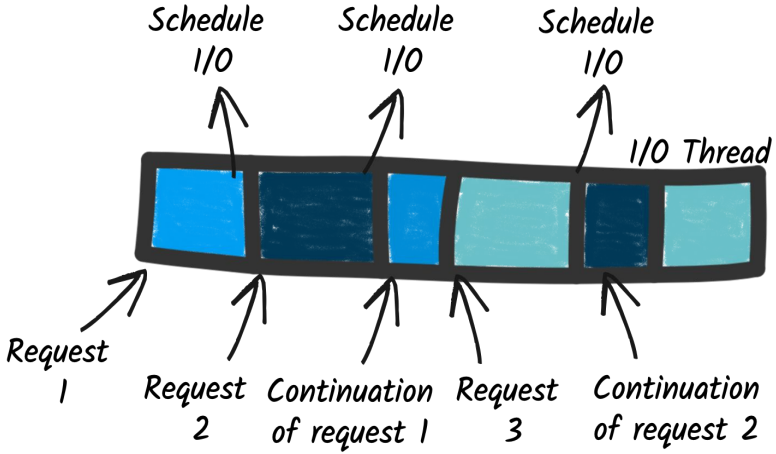
https://quarkus.io/guides/quarkus-reactive-architecture

Blocking vs Non-Blocking Execution Model

Traditional - Blocking I/O

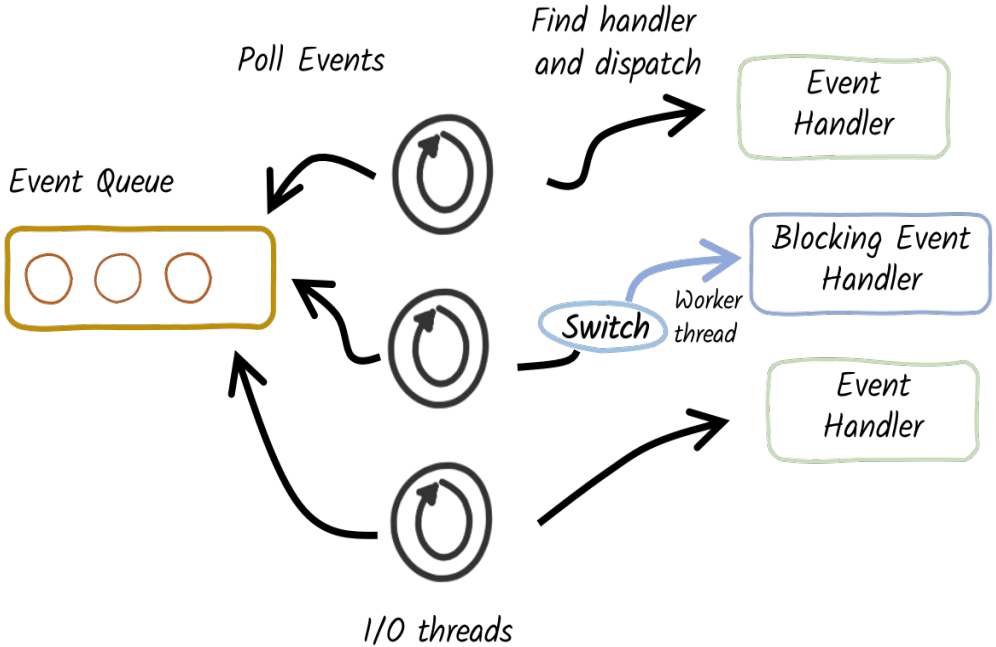


Reactive - Non-Blocking I/O

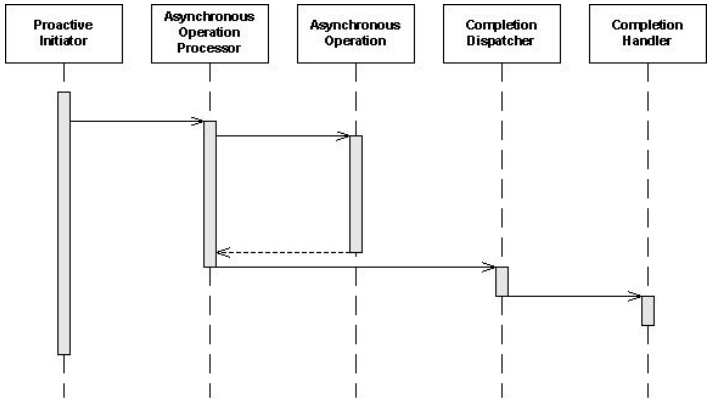


https://quarkus.io/guides/quarkus-reactive-architecture

Unification of Reactive and Imperative

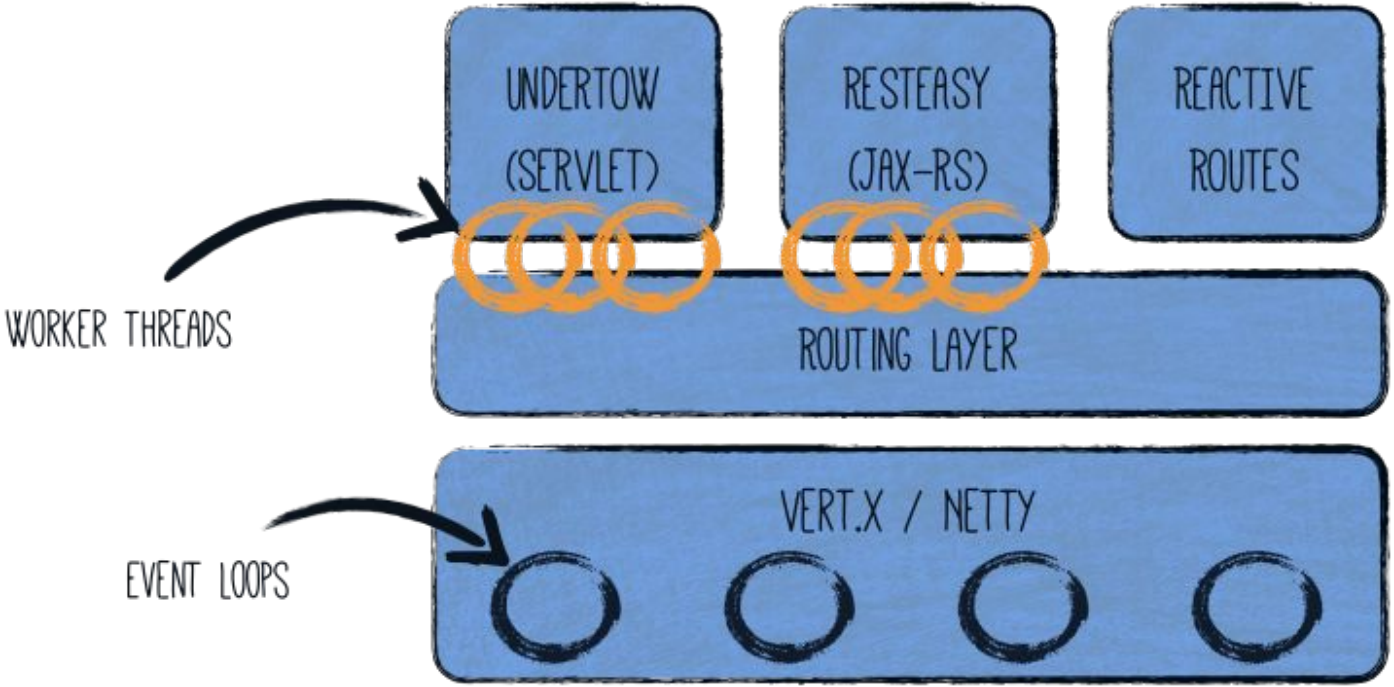


The Switch from reactive to imperative is done using the Protractor pattern
https://en.wikipedia.org/wiki/Proactor_pattern



https://quarkus.io/guides/quarkus-reactive-architecture

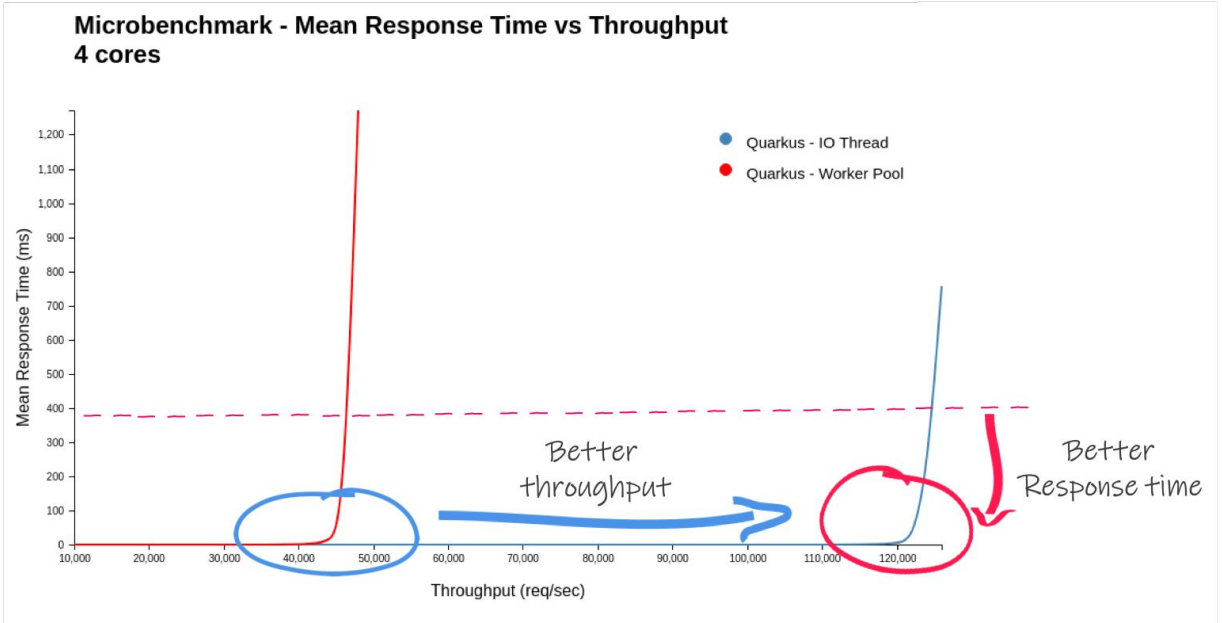
I/O Threads (Event Loops), Worker Threads



https://quarkus.io/guides/reactive-routes

Why Does it matter which Thread ?

- **Faster** Startup
- **Smaller** RSS
- **Higher** Throughput
- **Faster** Response Time



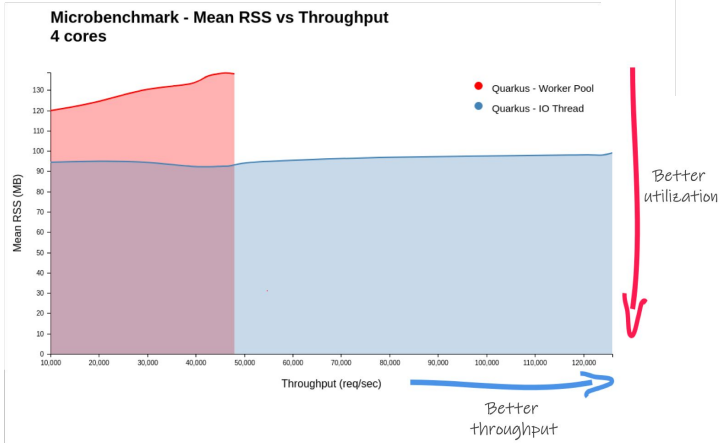
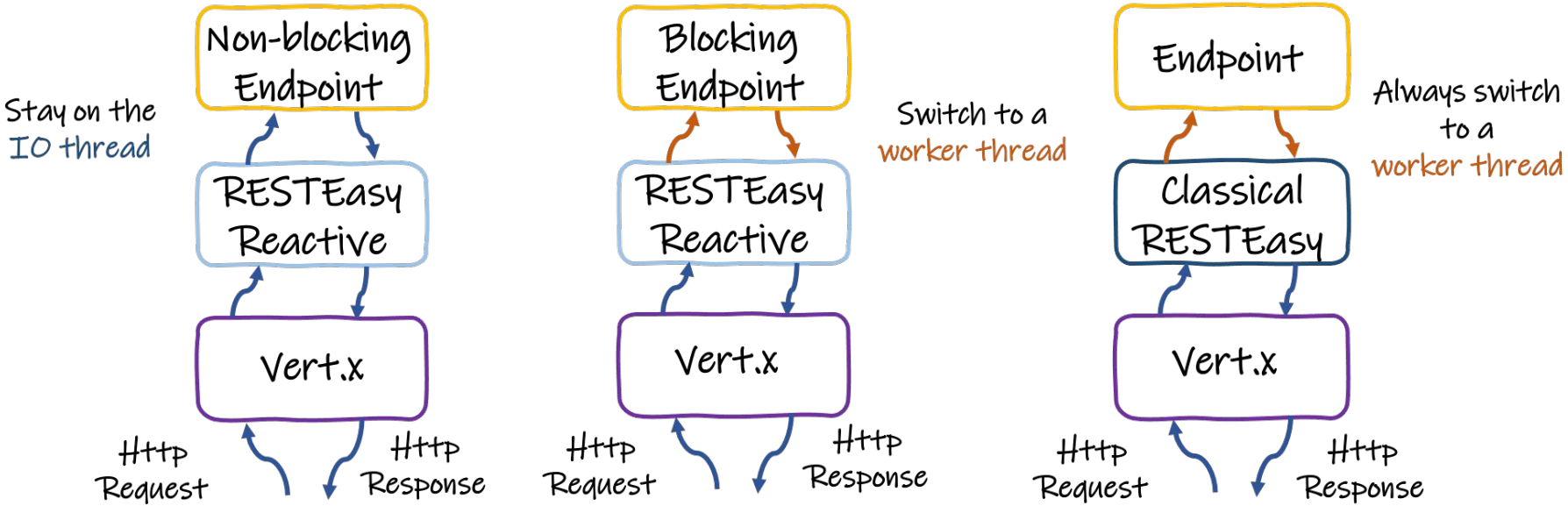


Table 1. Microbenchmark results comparing Quarkus dispatching to a worker thread vs running purely on the IO thread

Quarkus - 1.3.1.Final - 4 CPU's	Worker thread	IO thread	Ratio
Mean Start Time to First Request (ms) [1]	993.9	868.3	87.4%
Max RSS (MB)	138.8	97.9	70.5%
Max Throughput (req/sec)	46,172.2	123,520.4	267.5%
Max Req/Sec/MB	332.7	1,262.1	379.4%

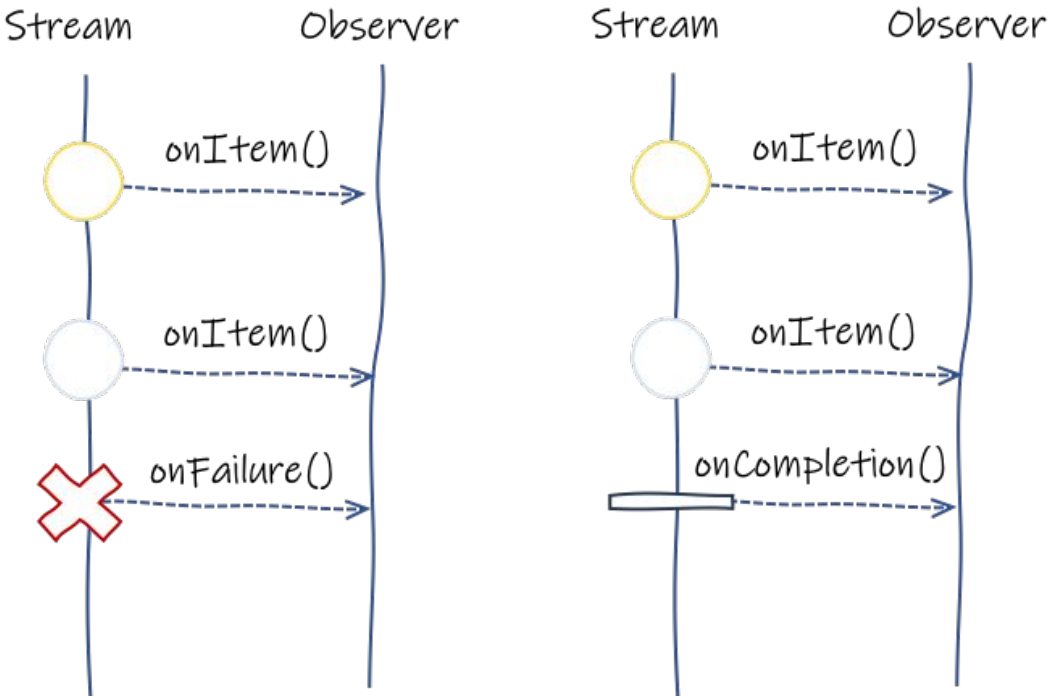
https://quarkus.io/blog/io-thread-benchmark

RESTEasy - Reactive (@NonBlocking, @Blocking) + Classic



https://quarkus.io/blog/resteasy-reactive-faq

Mutiny - Reactive Programming



With reactive programming:

- you observe streams of events
- and implement side-effects

when something flows in the event stream.

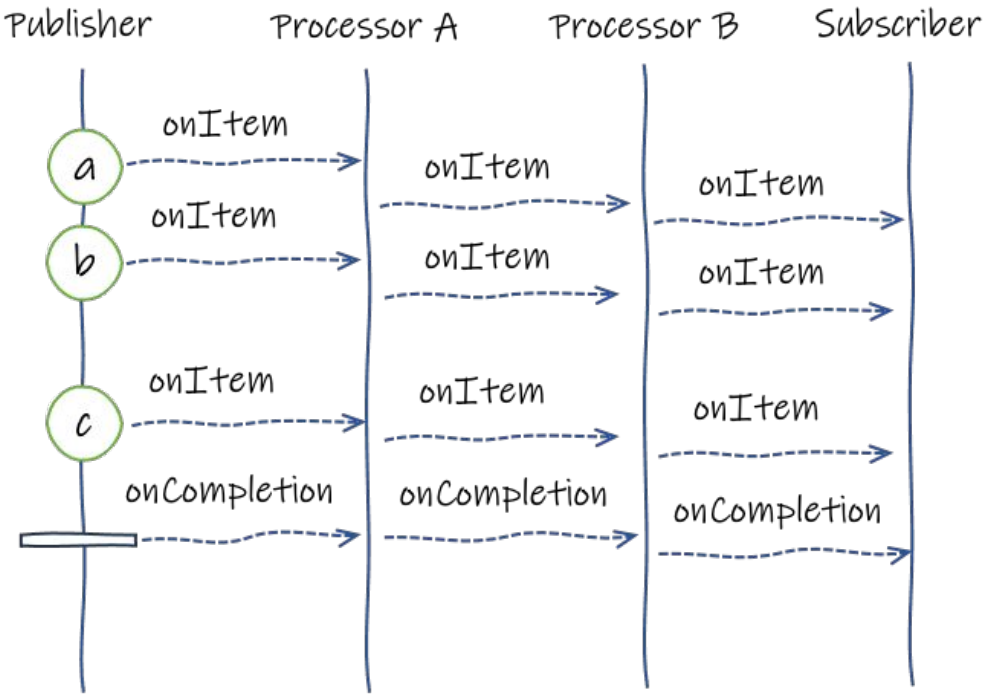
Mutiny provides **only** two types:

Multi - represents streams of 0..* items (potentially unbounded)

Uni - represents streams receiving either an item or a failure

https://smallrye.io/smallrye-mutiny/pages/philosophy
https://quarkus.io/guides/mutiny-primer

Event Pipelines



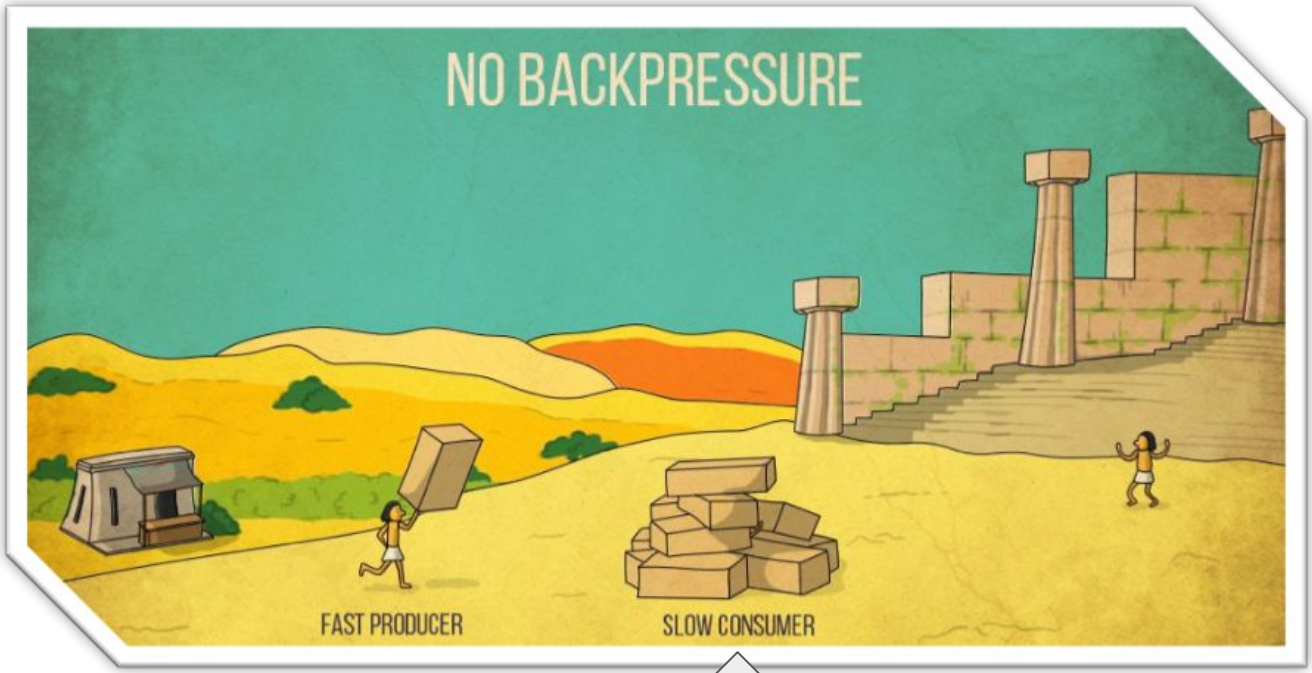
- You design a pipeline of actions into which the events flow
- From upstream (source) to downstream (sinks)
- Events going from upstream to downstream are published by `Publishers` and consumed by (downstream) `Subscribers`

https://smallrye.io/smallrye-mutiny/pages/philosophy

What is Back Pressure ?

https://quarkus.io/blog/mutiny-back-pressure

What happens when you do not deal with Back Pressure?

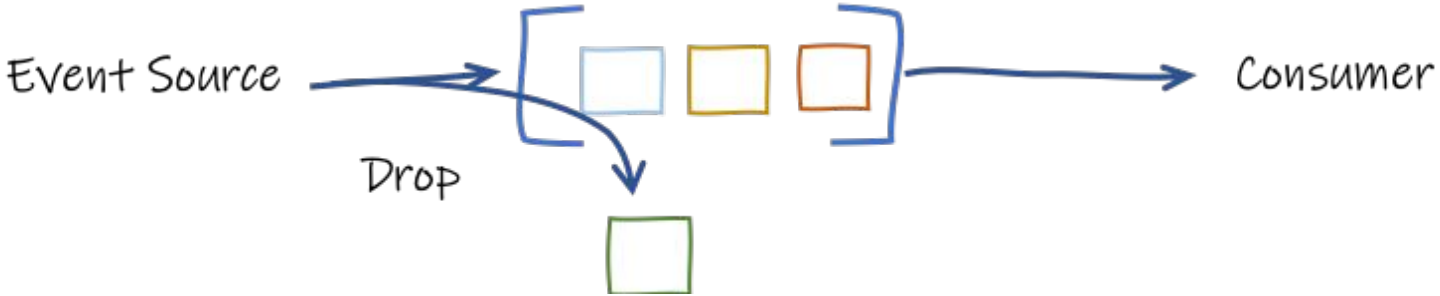


https://quarkus.io/blog/mutiny-back-pressure

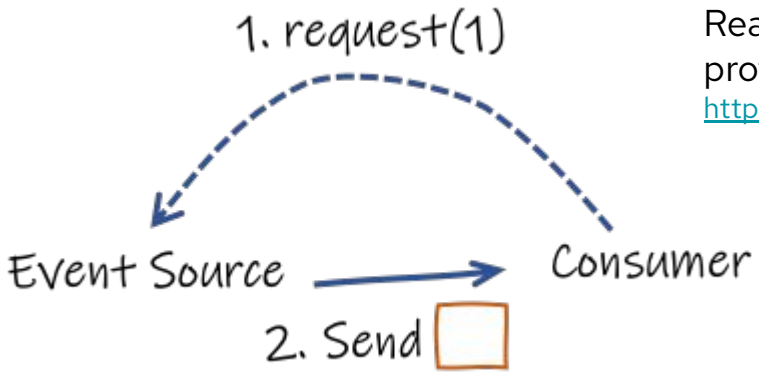
😵 Oh no, did we kill the consumer ?!

Back Pressure - Solutions

Just Drop It !
(or buffer then drop)



Consumer controls the Flow



Reactive Streams a back pressure protocol
https://www.reactive-streams.org

https://quarkus.io/blog/mutiny-back-pressure

Back Pressure - Reactive Streams - Mutiny does it for you, built in

https://quarkus.io/blog/mutiny-back-pressure

Back pressure .. built in handling
we can use invoke() to see what's going on under the hood

```java
@Path("/handleit")
public class BackPressure {

    @GET
    public void handleIt() {
        Multi.createFrom().range(0, 10)
                .onSubscription().invoke(sub -> System.out.println("Received subscription: " + sub))
                .onRequest().invoke(req -> System.out.println("Got a request: " + req))
                .select().where((i -> i % 2 == 0))
                .onItem().transform(i -> i * 100);
    }

}
```

Q. What number(s) get printed ?

Err.. this code does nothing ... Why not ???!

DON'T FORGET TO SUBSCRIBE

If no subscriber *subscribes*, no items would be emitted. More importantly, nothing will ever happen. If your program does not do anything, check that it subscribes, it's a very common error.

REACTIVE STREAMS

Mutiny's back-pressure is based on Reactive Streams.

```
@Path("/handleit")
public class BackPressure {

    @GET
    public void handleIt() {
        Multi.createFrom().range(0, 10)
                .onSubscription().invoke(sub -> System.out.println("Received subscription: " + sub))
                .onRequest().invoke(req -> System.out.println("Got a request: " + req))
                .select().where((i -> i % 2 == 0))
                .onItem().transform(i -> i * 100)
                .subscribe().with(
                        i -> System.out.println("i: " + i)
                );
    }
}
```

the reactive stream **Subscriber** requests ***one more***

```
Received subscription: io.smallrye.mutiny.operators
.multi.builders.IterableBasedMulti$IteratorSubscrip
tion@45f343fc
Got a request: 9223372036854775807
i: 0
i: 200
i: 400
i: 600
i: 800
```

Quarkus 2.2.0 - New Smart Dispatching Strategy

Method signature	Dispatching strategy
`T method(…)`	Worker thread
`Uni<T> method(…)`	I/O thread
`CompletionStage<T> method(…)`	I/O thread
`Multi<T> method(…)`	I/O thread
`Publisher<T> method(…)`	I/O thread
`@Transactional CompletionStage<T> method(…)`	Worker thread

- Classic/Imperative - **synchronous** methods default to **worker threads**
- Reactive - **asynchronous** methods default to **I/O threads**, except if explicitly stated otherwise i.e. @Blocking, @NonBlocking, @Transactional

https://quarkus.io/blog/resteasy-reactive-smart-dispatch

Q. Mutiny - What will this print in the log ??

```java
@Path("/hello")
public class ReactiveGreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        System.out.println(Thread.currentThread().getName());
        return "Hello RESTEasy Reactive";
    }
}
```

```
2021-09-29 14:29:32,537 INFO  [io.quarkus] (Quarkus Main Thread) quarku
us 2.2.3.Final) started in 1.498s. Listening on: http://localhost:8080
2021-09-29 14:29:32,552 INFO  [io.quarkus] (Quarkus Main Thread) Profil
2021-09-29 14:29:32,553 INFO  [io.quarkus] (Quarkus Main Thread) Instal
xt-propagation]
executor-thread-0
executor-thread-0
executor-thread-0
```

Worker Thread

```java
@Path("/hello")
public class ReactiveGreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Uni<String> hello() {
        System.out.println(Thread.currentThread().getName());
        return Uni.createFrom().item("Hello RESTEasy Reactive");
    }
}
```

```
2021-09-29 14:30:40,364 INFO  [io.quarkus] (Quarkus Main Thread) quarkus-
us 2.2.3.Final) started in 0.255s. Listening on: http://localhost:8080
2021-09-29 14:30:40,365 INFO  [io.quarkus] (Quarkus Main Thread) Profile
2021-09-29 14:30:40,365 INFO  [io.quarkus] (Quarkus Main Thread) Installe
xt-propagation]
2021-09-29 14:30:40,366 INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (
4s
vert.x-eventloop-thread-1
vert.x-eventloop-thread-10
vert.x-eventloop-thread-0
```

I/O Thread

Q. And now what happens ??

```java
@Path("/hello")
public class ReactiveGreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Blocking
    public Uni<String> hello() {
        System.out.println(Thread.currentThread().getName());
        return Uni.createFrom().item("Hello RESTEasy Reactive");
    }
}
```

```
2021-09-29 14:29:32,537 INFO  [io.quarkus] (Quarkus Main Thread) quarku
us 2.2.3.Final) started in 1.498s. Listening on: http://localhost:8080
2021-09-29 14:29:32,552 INFO  [io.quarkus] (Quarkus Main Thread) Profil
2021-09-29 14:29:32,553 INFO  [io.quarkus] (Quarkus Main Thread) Instal
xt-propagation]
executor-thread-0
executor-thread-0
executor-thread-0
```

Worker Thread

Other Reactive Extensions (not an exhaustive list)

HTTP
- RESTEasy Reactive - JAX-RS
- Reactive Routes - Vert.x routes
- Reactive Rest Client - consuming reactive endpoints
- Qute Templating - server side template rendering

DATA
- Hibernate Reactive - ORM
- Hibernate Reactive with Panache - record & repository support
- Reactive PostgreSQL, MySQL, MongoDB clients
- Mongo with Panache
- Cassandra client
- Redis

EDA
- Reactive Messaging
- Kafka Connector for Reactive Messaging
- AMQP 1.0 Connector for Reactive Message

NETWORK
- gRPC
- GraphQL
- Fault Tolerance

ENGINE
- Vert.x
- Context Propagation

https://quarkus.io/guides/quarkus-reactive-architecture#quarkus-extensions-enabling-reactive

Let’s Run our own - hello world - microbenchmark …

Worker Thread

```
virt:~$ hey -c 1000 -n 100000 http://localhost:8080/hello

Summary:
  Total:        56.5157 secs
  Slowest:      1.4144 secs
  Fastest:      0.0008 secs
  Average:      0.5549 secs
  Requests/sec: 1769.4206

  Total data:   2298574 bytes
  Size/request: 23 bytes

Response time histogram:
  0.001 [1]     |
  0.142 [1107]  |■■
  0.284 [6621]  |■■■■■■■■■■■
  0.425 [21818] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.566 [22846] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.708 [24851] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.849 [14919] |■■■■■■■■■■■■■■■■■■■■■■■■
  0.990 [5833]  |■■■■■■■■■
  1.132 [1427]  |■■
  1.273 [424]   |■
  1.414 [91]    |


Latency distribution:
  10% in 0.2958 secs
  25% in 0.3851 secs
  50% in 0.5521 secs
  75% in 0.6933 secs
  90% in 0.8202 secs
  95% in 0.9052 secs
  99% in 1.0725 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0003 secs, 0.0008 secs, 1.4144 secs
  DNS-lookup:   0.0001 secs, 0.0000 secs, 0.0371 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0360 secs
  resp wait:    0.5543 secs, 0.0008 secs, 1.4144 secs
  resp read:    0.0000 secs, 0.0000 secs, 0.0502 secs

Status code distribution:
  [200] 99938 responses

Error distribution:
  [42]  Get "http://localhost:8080/hello": dial tcp 127.0.0.1:8080: socket: too many open files
  [4]   Get "http://localhost:8080/hello": dial tcp [::1]:8080: connect: connection refused
  [16]  Get "http://localhost:8080/hello": dial tcp [::1]:8080: socket: too many open files
```

I/O Thread

```
virt:~$ hey -c 1000 -n 100000 http://localhost:8080/hello

Summary:
  Total:        31.0738 secs
  Slowest:      0.9173 secs
  Fastest:      0.0006 secs
  Average:      0.2880 secs
  Requests/sec: 3218.1414

  Total data:   2300000 bytes
  Size/request: 23 bytes

Response time histogram:
  0.001 [1]     |
  0.092 [3396]  |■■■
  0.184 [2632]  |■■
  0.276 [42791] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.367 [42351] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.459 [2457]  |■■
  0.551 [2278]  |■■
  0.642 [3039]  |■■■
  0.734 [928]   |■
  0.826 [99]    |
  0.917 [28]    |


Latency distribution:
  10% in 0.2198 secs
  25% in 0.2549 secs
  50% in 0.2766 secs
  75% in 0.3046 secs
  90% in 0.3546 secs
  95% in 0.5273 secs
  99% in 0.6490 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0004 secs, 0.0006 secs, 0.9173 secs
  DNS-lookup:   0.0002 secs, 0.0000 secs, 0.0659 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0152 secs
  resp wait:    0.2874 secs, 0.0005 secs, 0.9172 secs
  resp read:    0.0000 secs, 0.0000 secs, 0.0044 secs

Status code distribution:
  [200] 100000 responses
```

82% more throughput
55 % faster clock time
52% faster mean response
61 % faster @99%
I/O Thread - No Errors

You know you messed up ... when ...

```
You have attempted to perform a blocking operation on a IO thread. This is not allowed, as blocking the IO thread
will cause major performance issues with your application. If you want to perform blocking EntityManager
operations make sure you are doing it from a worker thread.: java.lang.IllegalStateException: You have attempted
to perform a blocking operation on a IO thread. This is not allowed, as blocking the IO thread will cause major
performance issues with your application. If you want to perform blocking EntityManager operations make sure you
are doing it from a worker thread.
```

Blocking I/O threads (considered Harmful) ...

```java
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Uni<String> hello() throws InterruptedException {
        System.out.println(Thread.currentThread().getName());
        Thread.sleep( 10000);
        return Uni.createFrom().item("Hello RESTEasy Reactive");
    }
}
```

... Boom !

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2021-09-29 15:21:25,077 INFO  [io.quarkus] (Quarkus Main Thread) quarkus-concurrency 1.0.0-SNAPSHOT on JVM (powered by Quarkus
2.2.3.Final) started in 0.298s. Listening on: http://localhost:8080
2021-09-29 15:21:25,077 INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
2021-09-29 15:21:25,077 INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, resteasy-reactive, smallrye-context-
propagation]
2021-09-29 15:21:25,078 INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (vert.x-worker-thread-1) Live reload total time: 0.344s
vert.x-eventloop-thread-4
2021-09-29 15:21:28,639 WARN  [io.ver.cor.imp.BlockedThreadChecker] (vertx-blocked-thread-checker) Thread Thread[vert.x-eventlo
op-thread-4,5,main] has been blocked for 3559 ms, time limit is 2000 ms: io.vertx.core.VertxException: Thread blocked
        at java.base@16.0.2/java.lang.Thread.sleep(Native Method)
        at org.acme.ReactiveGreetingResource.hello(ReactiveGreetingResource.java:18)
        at org.acme.ReactiveGreetingResource_Subclass.hello$$superforward1(ReactiveGreetingResource_Subclass.zig:89)
        at org.acme.ReactiveGreetingResource_Subclass$$function$$15.apply(ReactiveGreetingResource_Subclass$$function$$15.zig:2
4)
        at io.quarkus.arc.impl.AroundInvokeInvocationContext.proceed(AroundInvokeInvocationContext.java:54)
        at io.quarkus.arc.runtime.devconsole.InvocationInterceptor.proceed(InvocationInterceptor.java:62)
        at io.quarkus.arc.runtime.devconsole.InvocationInterceptor.monitor(InvocationInterceptor.java:49)
```

Non-Blocking sleep ...

```
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Uni<String> hello() throws InterruptedException {
        System.out.println(Thread.currentThread().getName());
        return Uni.createFrom().item("Hello RESTEasy Reactive").onItem().delayIt().by(Duration.ofSeconds(10));
    }
}
```

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2021-09-29 15:29:39,916 INFO  [io.quarkus] (Quarkus Main Thread) quark
2.2.3.Final) started in 0.227s. Listening on: http://localhost:8080
2021-09-29 15:29:39,916 INFO  [io.quarkus] (Quarkus Main Thread) Profi
2021-09-29 15:29:39,917 INFO  [io.quarkus] (Quarkus Main Thread) Insta
propagation]
2021-09-29 15:29:39,917 INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor]
vert.x-eventloop-thread-1
```

All Good !!

Doing Blocking Work with Mutiny …

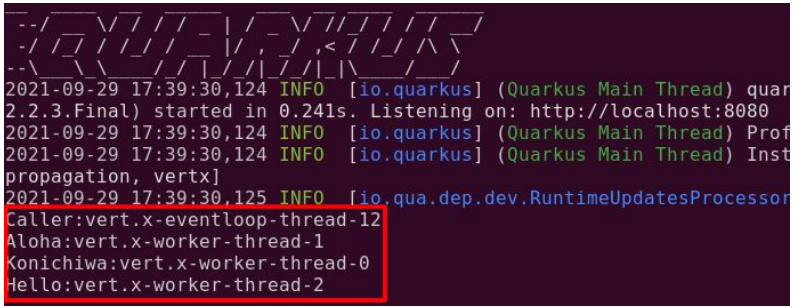
```java
@GET
@Produces(MediaType.TEXT_PLAIN)
public Uni<String> hello() {
    System.out.println(Thread.currentThread().getName());
    Uni.createFrom().item("Mutiny").emitOn(Infrastructure.getDefaultWorkerPool()).subscribe().with(
            item -> System.out.println(item + ":" + Thread.currentThread().getName())
    );
    return Uni.createFrom().item("Hello RESTEasy Reactive");
}
```

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2021-09-29 15:47:45,061 INFO  [io.quarkus] (Quarkus Main Thread) quarkus-
2.2.3.Final) started in 1.506s. Listening on: http://localhost:8080
2021-09-29 15:47:45,079 INFO  [io.quarkus] (Quarkus Main Thread) Profile
2021-09-29 15:47:45,080 INFO  [io.quarkus] (Quarkus Main Thread) Installe
propagation]
vert.x-eventloop-thread-1
Mutiny:executor-thread-0
```

Fun ways to Parallelize work

```java
@Path("/hello")
public class ReactiveGreetingResource {

    @Inject
    EventBus bus;

    @GET
    @NonBlocking
    @Produces(MediaType.TEXT_PLAIN)
    public void hello() {
        System.out.println("Caller:" + Thread.currentThread().getName());
        Arrays.asList("Hello", "Aloha", "Konichiwa").forEach(
                item -> bus.<String>request( address: "hi", item)
        );
    }

    @ConsumeEvent(value = "hi", blocking = true)
    public void doSomething(String greeting) {
        System.out.println(greeting + ":" + Thread.currentThread().getName());
    }
}
```

Notice: the vert.x-worker-thread pool



Vert.x Reactive WebClient

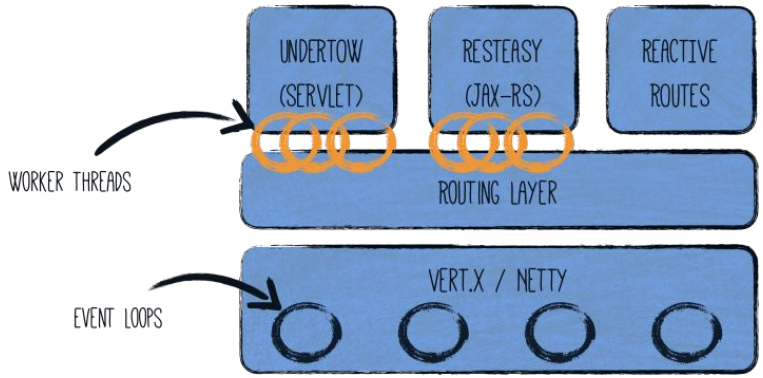
… so elegant !

```java
@Path("/joke")
public class VertxResource {

    @Inject
    Vertx vertx;

    private final String URL = "https://api.chucknorris.io/jokes/random";

    @GET
    public Uni<JsonObject> chuckNorrisJokes() {
        return WebClient.create(vertx).getAbs(URL).send()
                .onItem().transform(HttpResponse::bodyAsJsonObject);
    }

}
```

```
virt:~$ curl -s -n localhost:8080/joke | jq .
{
  "categories": [],
  "created_at": "2020-01-05 13:42:25.905626",
  "icon_url": "https://assets.chucknorris.host/img/avatar/chuck-norris.png",
  "id": "gt0TtJAWRPCmihXVYVDHaw",
  "updated_at": "2020-01-05 13:42:25.905626",
  "url": "https://api.chucknorris.io/jokes/gt0TtJAWRPCmihXVYVDHaw",
  "value": "Chuck Norris can squeeze lime juice out of a lemon."
}
```

Thread Pool Configuration Parameters

```
quarkus.vertx.worker-pool-size = 20 default

quarkus.vertx.event-loops-pool-size = 2 x CPU count

quarkus.http.io-threads = 2 x CPU count
```



- Worker threads are created lazily, so setting a high number to `quarkus.vertx.worker-pool-size` might not have any immediate effect.
- For event loop tasks, size of the thread pool is two times the CPU count by default

- If the REST services do a lot of long-running synchronous operations, then you might need to increase `quarkus.vertx.worker-pool-size` , because these threads handle the long running operations.
- If you have a high amount of HTTP requests but they don't necessarily take long to handle (or are asynchronous), you might need to increase quarkus.http.io-threads

https://quarkus.io/guides/all-config