
Quarkus 3
The Road to Loom for Cheaper, Faster, and Easier Concurrent Apps
Project Loom

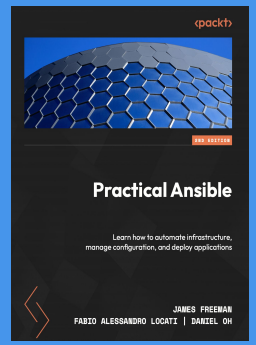
Daniel Oh

- Developer Advocate at Red Hat
  - Cloud Native Runtimes
  - Serverless, Service Mesh, and GitOps
- JAVA Champion
- Cloud Native Computing Foundation (CNCF) Ambassador
- Advisory Board Member of Global Skill Development Council
- Opensource.com Correspondents
- Public Speaker & Published Author

@danieloh30

@danieloh30

danieloh30



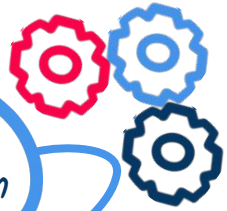
A  about threads

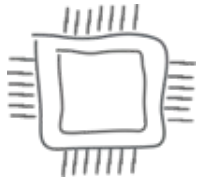
Your program runs inside a **process**

Executed code runs on a **process thread**

Your program / framework can **spawn** new process threads

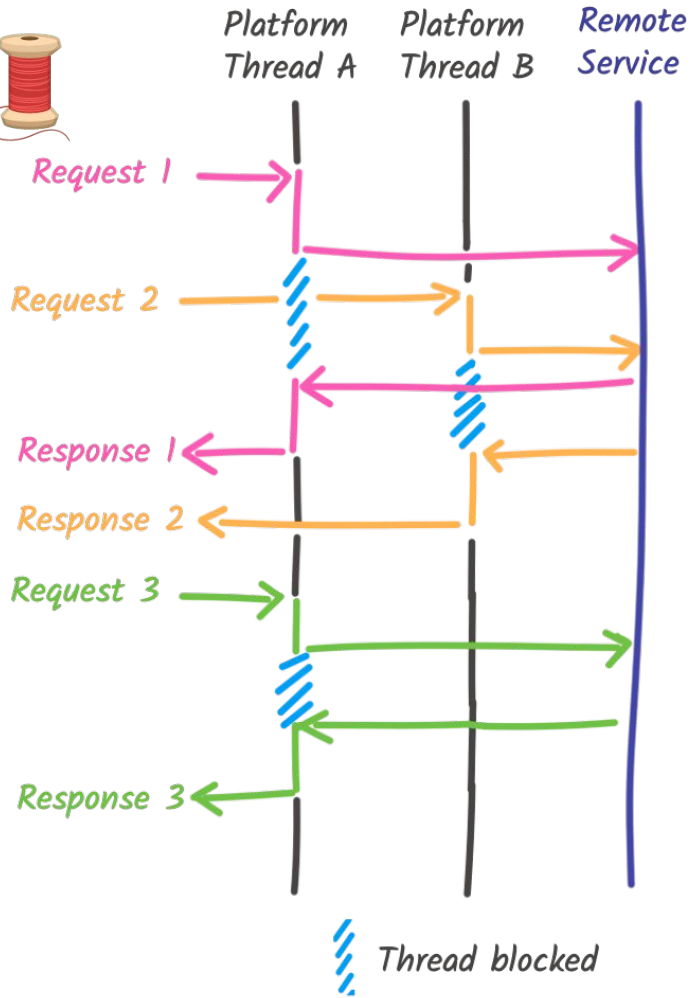
The OS **schedules** threads to run them on **CPU**





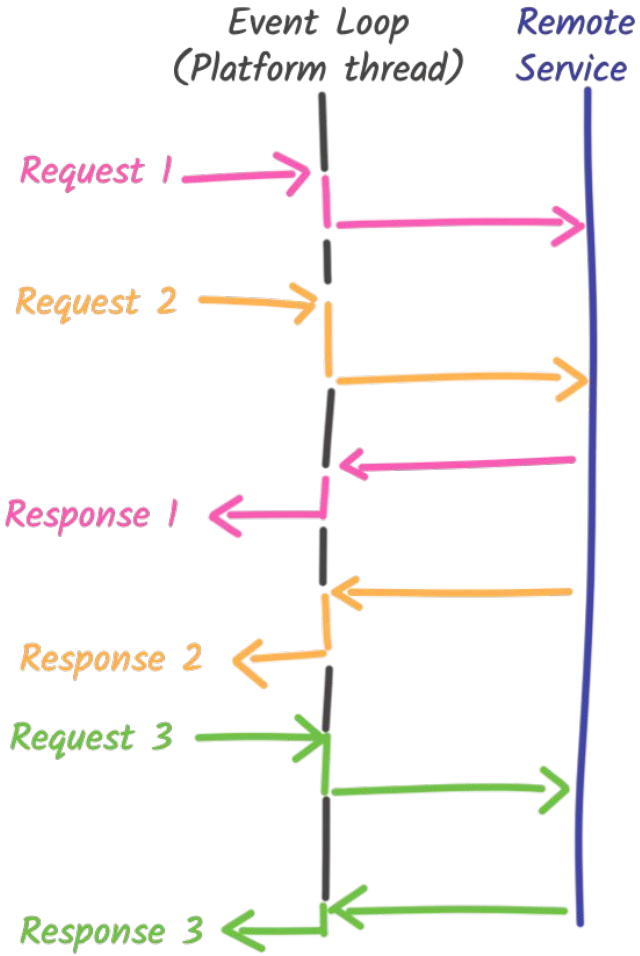
Imperative

```java
@Path("/greetings")
public class ImperativeApp {

  @RestClient RemoteService service;

  @GET
  public String process() {
    // Runs on a worker (platform) thread because the
    // method uses a synchronous signature

    // `service` is a rest client, it executes an I/O operation
    // (send an HTTP request and waits for the response)
    var response = service.greetings(); // Blocking, it waits for the response
                                        // The OS thread is also blocked

    return response.toUpperCase();
  }

}
```



Reactive

```java
@Path("/greetings")
public class ReactiveApp {

  @RestClient RemoteService service;

  @GET
  public Uni<String> process() {
    // Runs on an event loop (platform) thread because the
    // method uses a asynchronous signature

    // `service` is a rest client, it executes an I/O operation
    // but this time it returns a Uni<String>, so it's not blocking
    return service.asyncGreetings() // Non-blocking
        .map(resp -> {
          // This is the continuation, the code executed once the
          // response is received
          return resp.toUpperCase();
        });
    }
}
```

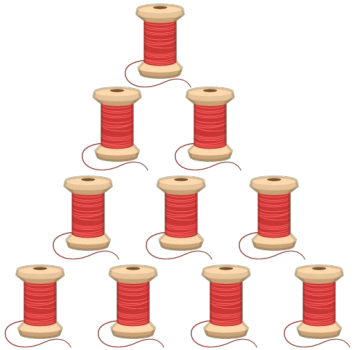


Reactive Golden Rule:

> Do not block the I/O thread.

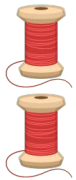
Imperative

- Simple development model
- Expensive: 1Mb / thread + 10 µs
- Limited concurrency



Reactive

- More complex development model
- Resource efficient
- High **concurrency**



Virtual Threads (Loom): The pros without the cons!

So, what about Virtual Threads (Loom)?

Imperative

- Simple development model
- Expensive: 1Mb / thread + 10 µs
- Limit concurrency

Virtual Threads

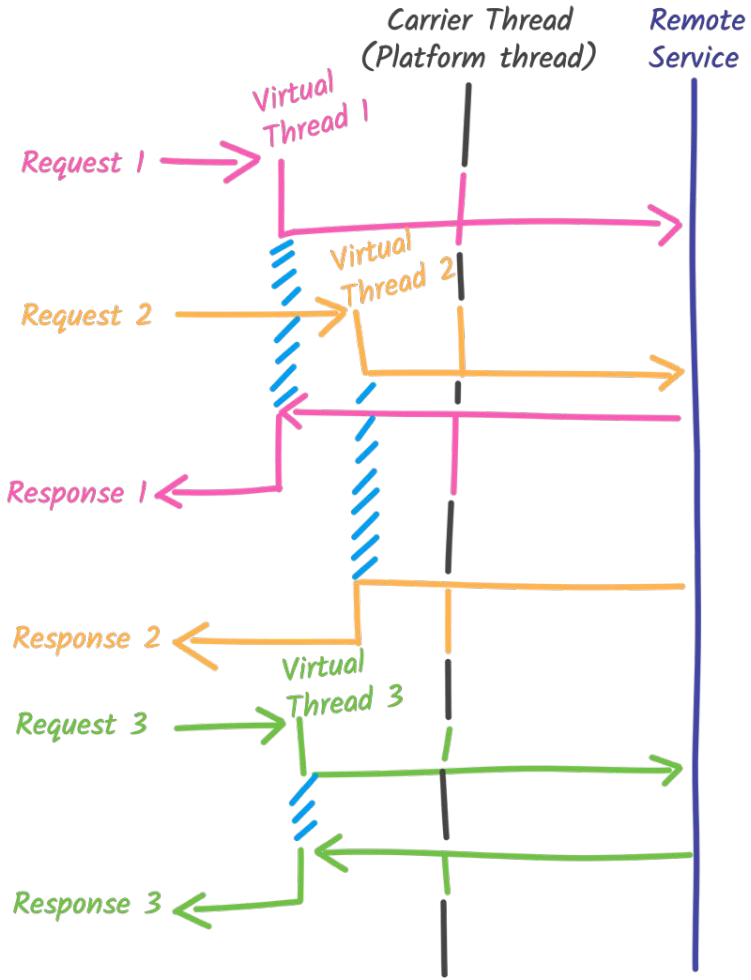
- Simple development model
- Resource efficient
- High concurrency

Reactive

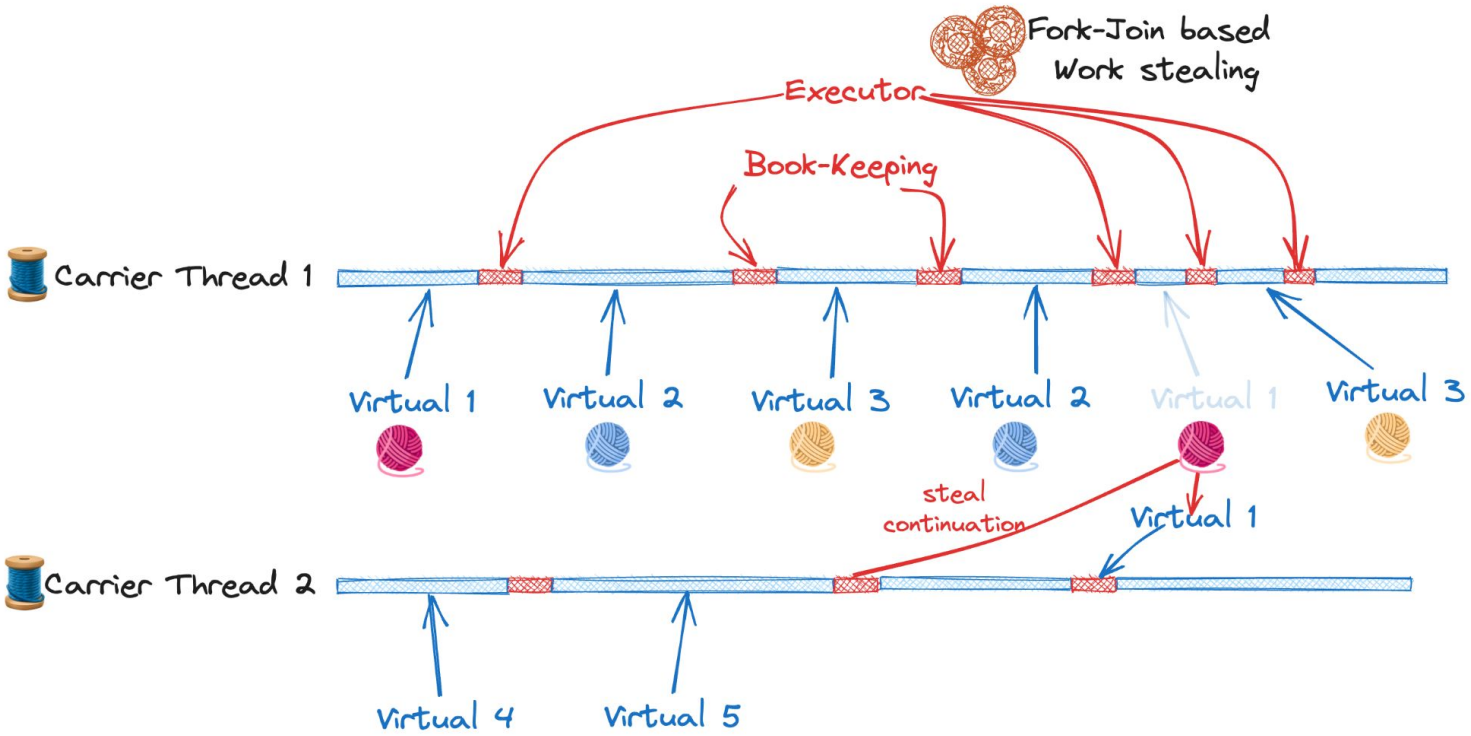
- More complex development model
- Resource efficient
- High concurrency

Virtual Threads: 1 , n

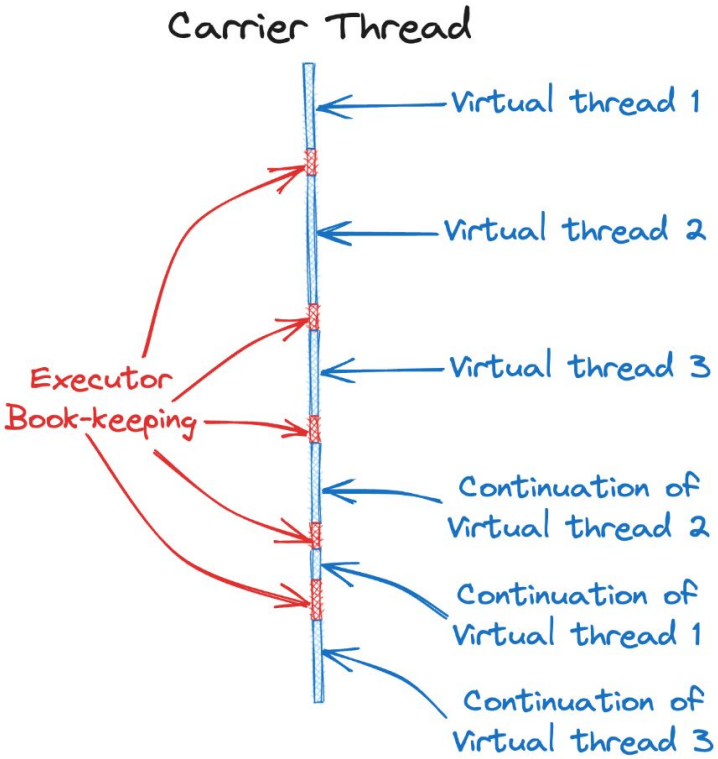
```java
@Path("/greetings")
public class VirtualThreadApp {

  @RestClient RemoteService service;

  @GET
  @RunOnVirtualThread
  public String process() {
    // Runs on a virtual thread because the
    // method uses the @RunOnVirtualThread annotation.

    // `service` is a rest client, it executes an I/O operation
    var response = service.greetings(); // Blocking, but this time, it
                                        // does neither block the carrier thread
                                        // nor the OS thread.
                                        // Only the virtual thread is blocked.
        return response.toUpperCase();
  }

}
```

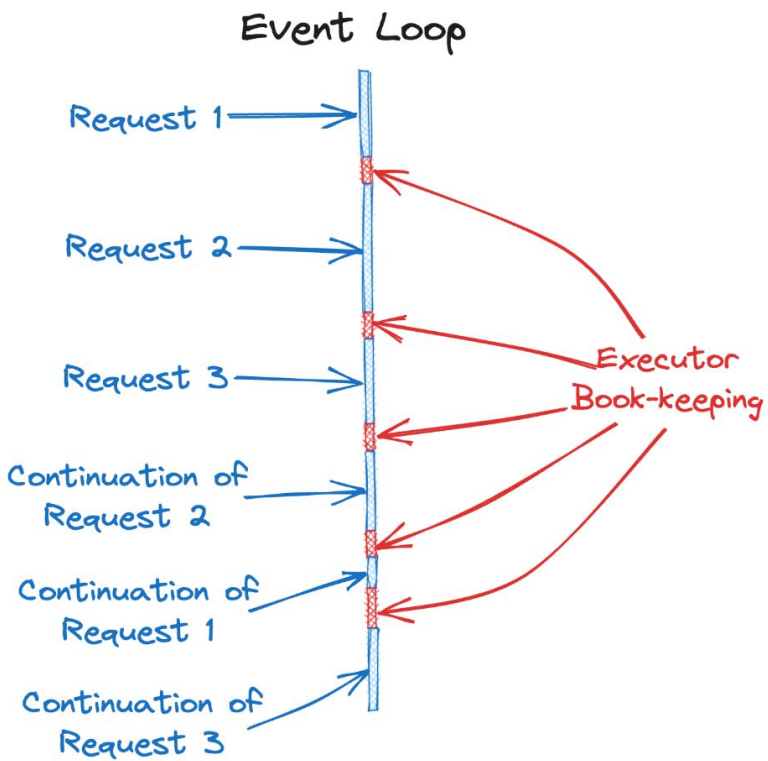


Virtual Threads: 1 , n: Scheduling made in JVM

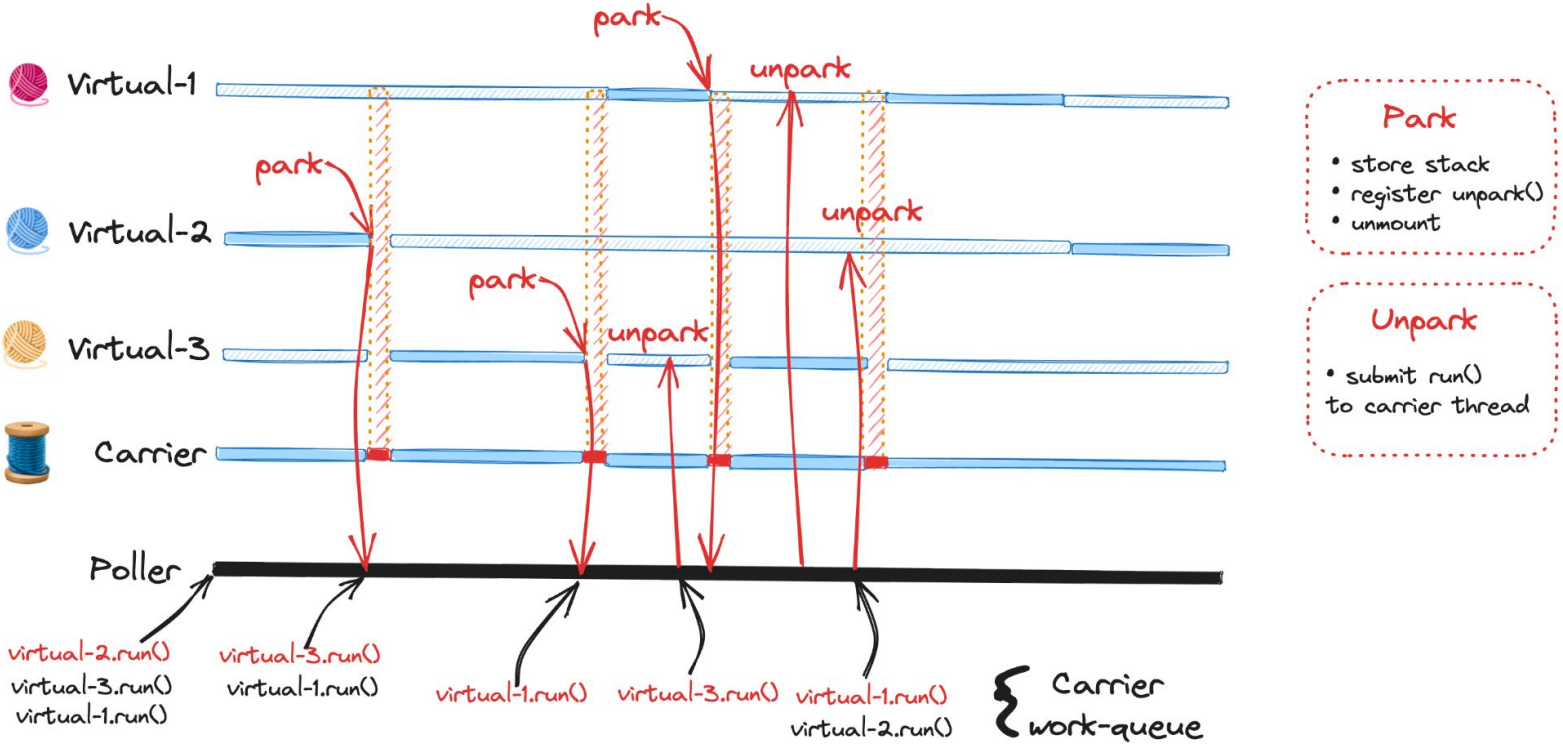


1 🧵, n 🧶 ≋ Reactive?


Carrier Thread
Virtual thread 1
Virtual thread 2
Virtual thread 3
Executor
Book-keeping
Continuation of
Virtual thread 2
Continuation of
Virtual thread 1
Continuation of
Virtual thread 3


Event Loop
Request 1
Request 2
Request 3
Executor
Book-keeping
Continuation of
Request 2
Continuation of
Request 1
Continuation of
Request 3

Suspending threads, how to?



Bored with the theory,
let's see some code!

Integration of Virtual Threads in Quarkus

Imperative / Blocking

```
@GET
public List<Todo> getAll() {
  return
Todo.listAll(Sort.by("order"));
}
```

Virtual Threads

```
@GET
@RunOnVirtualThread
public List<Todo> getAll() {
  return
Todo.listAll(Sort.by("order")
);
}
```

Reactive / Non-Blocking

```
@GET
public Uni<List<Todo>>
getAll() {
  return
Panache.withTransaction(()
->
Todo.findAll(Sort.by("order"
)).list());
}
```

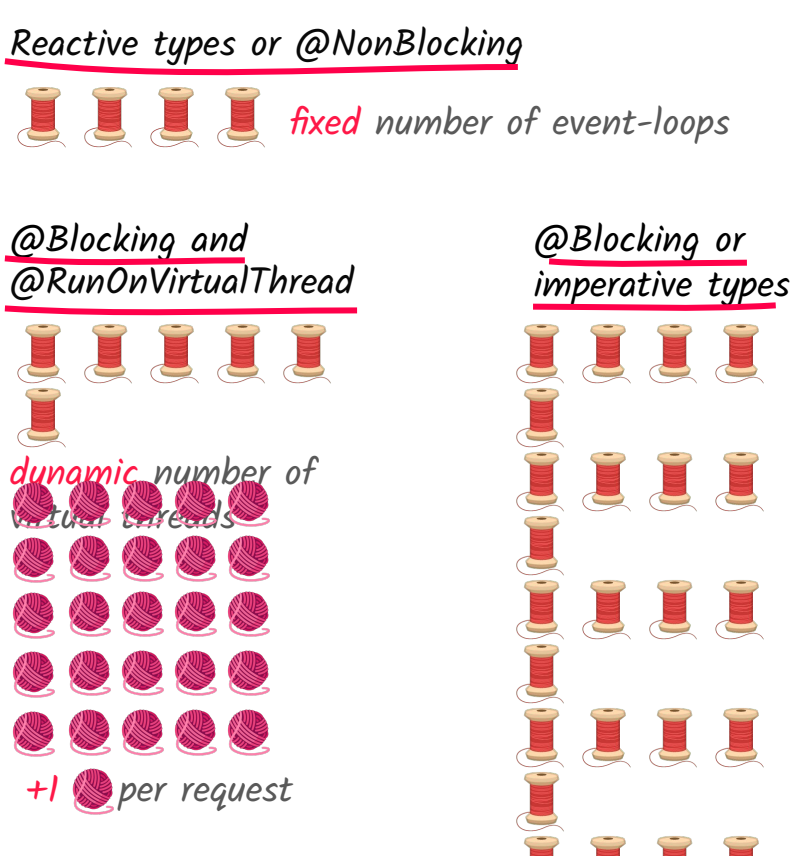
https://github.com/danieloh30/quarkus-todo-app-main

Under the hood

At build time, Quarkus analyzes the endpoints:

- Return types (reactive vs. imperative types)

- Method annotations
  - **@RunOnVirtualThread**
  - @Blocking
  - @NonBlocking



DEMO

https://github.com/danieloh30/quarkus-todo-app-main

One annotation and everything is great?

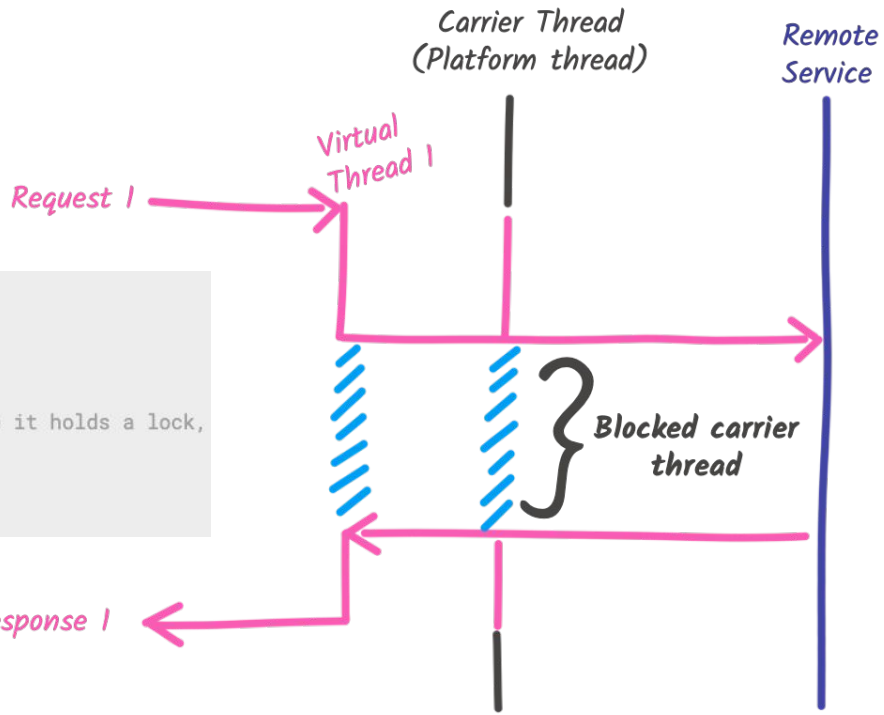
Why not run everything on virtual threads?

5 things you need to know on virtual threads

Pinning

- Synchronized block or method
- Native method or Foreign function

```
Object monitor = new Object();
//...
public void aMethodThatPinTheCarrierThread() throws Exception {
  synchronized(monitor) {
    Thread.sleep(1000); // The virtual thread cannot be unmounted because it holds a lock,
                        // so the carrier thread is blocked.
  }
}
```

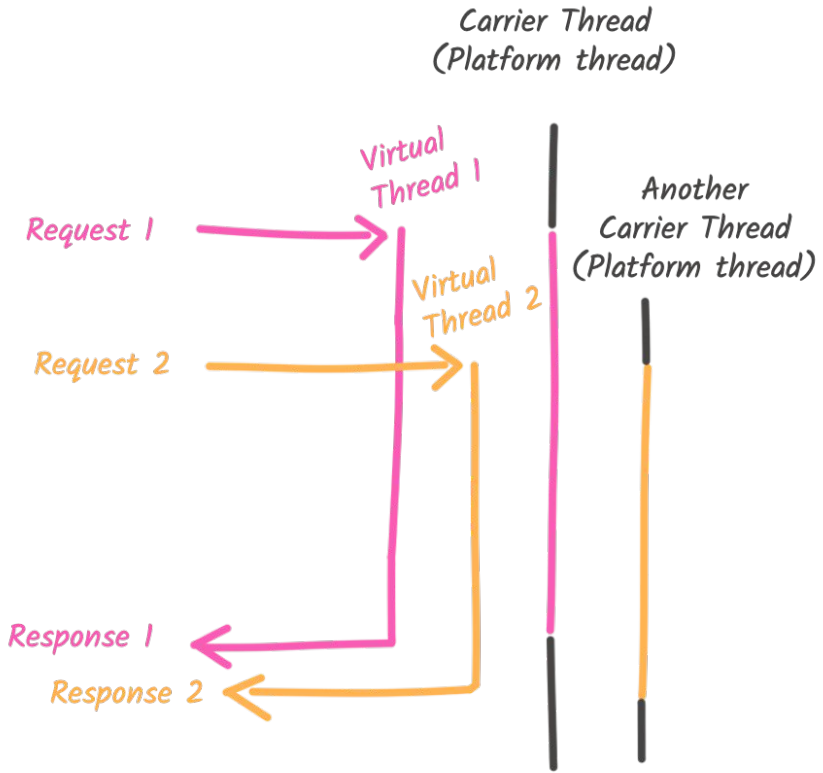


https://quarkus.io/blog/virtual-thread-1/

5 things you need to know on virtual threads

Monopolization

- Lead to high-memory usage
- Avoid with using a dedicated platform thread pool



https://quarkus.io/blog/virtual-thread-1/

5 things you need to know on virtual threads

Stressing thread safety

- Ensure to develop thread-safe code
- Lots of libraries and frameworks do not allow concurrent access

```java
@GET
@RunOnVirtualThread
public String structuredConcurrencyExample() throws InterruptedException, ExecutionException {
    var someState = ... // Must be thread-safe, as multiple virtual thread will access
                        // it concurrently
    try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
        var task1 = scope.fork(() -> {
          // Run in another virtual thread
          return someState.touch();
        });
        var task2 = scope.fork(() -> {
          // Run in another virtual thread
          return someState.touch();
        });

        scope.join().throwIfFailed();

        return task1.get() + "/" + task2.get();
    }
}
```

https://quarkus.io/blog/virtual-thread-1/

5 things you need to know on virtual threads 

Object pooling

- Using ThreadLocal (e.g., Jackson or Netty)
- Limited number of threads (recycled thread pool, unrelated but sequential tasks)

https://quarkus.io/blog/virtual-thread-1/

5 things you need to know on virtual threads 

Carrier thread pool elasticity

- Avoid having too many unscheduled virtual threads
- If you don't, in a container with memory constraints, the application can be killed

 https://quarkus.io/blog/virtual-thread-1/

A lot of the Java ecosystem would need a rewrite...

More Java ecosystems

- JMS, Rest Client, Mailer, Event Bus, Scheduled Methods, and so on

What about numbers...

The application

Techempower

- Multiple scenarios
- Run on bare metal and Cloud
- Multi languages, Multi frameworks

The *Todo scenario**

1. Fetch all rows from a DB (quotes)
2. Add one quote in the returned list
3. Sort the list
4. *Return the list as JSON**

https://github.com/danieloh30/quarkus-todo-app-main

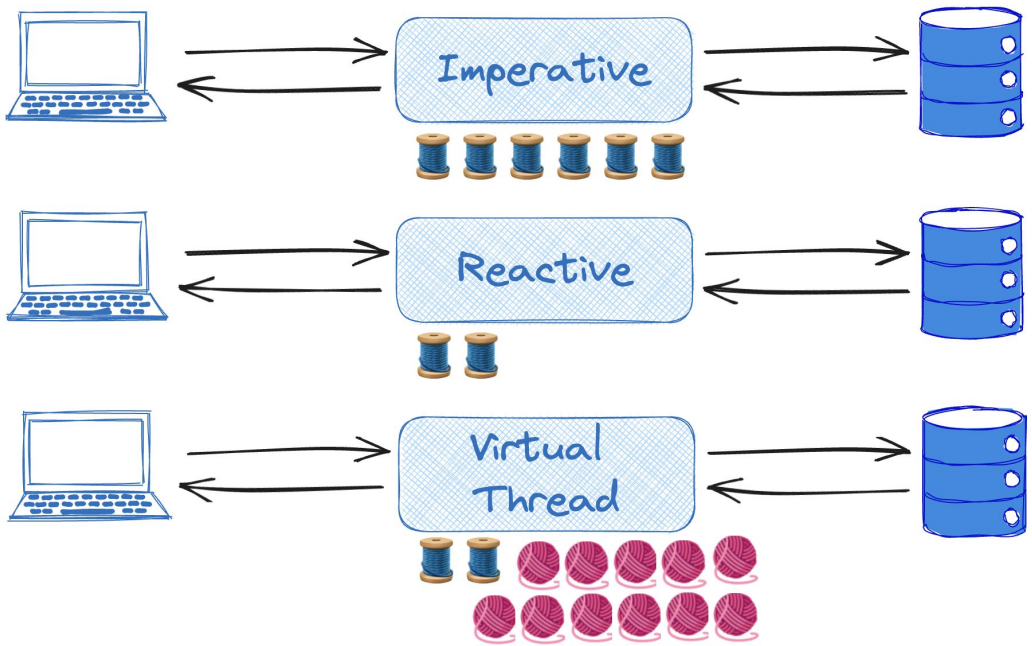
The benchmark

Variants

- Blocking,
- Non-Blocking (Reactive)
- @RunOnVirtualThread (Loom)

Environments

- Container (0.5vCPU, 512Mb Ram)



https://github.com/danieloh30/quarkus-todo-app-main

The benchmark

Concurrency

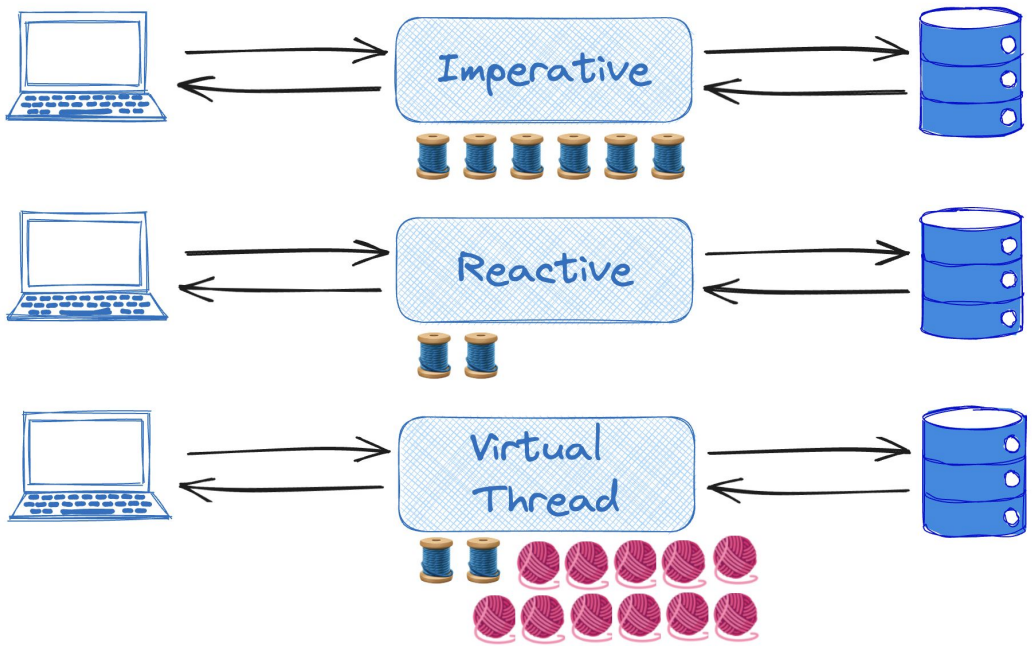
- 1200 => 4400 req/sec

Latency

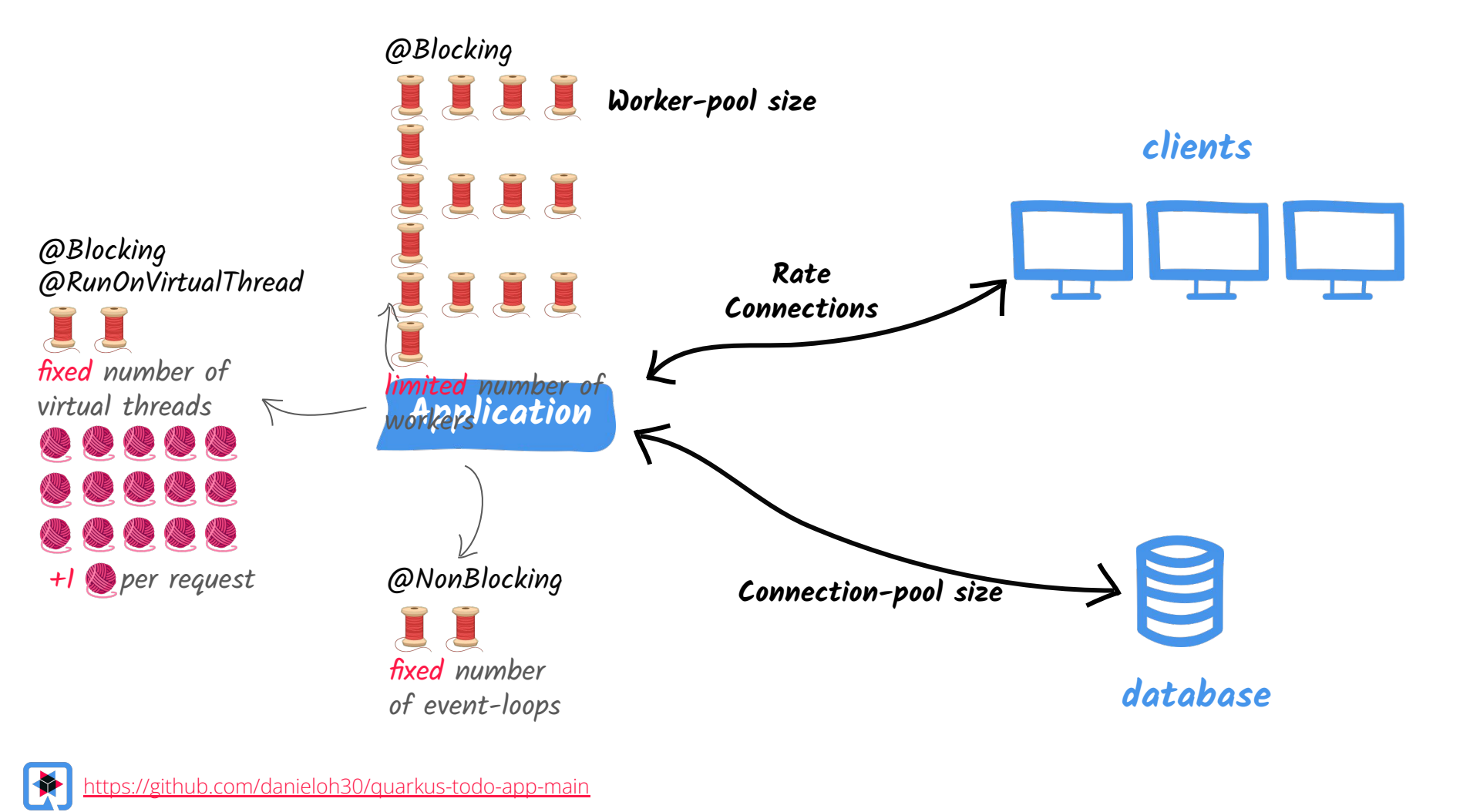
- 0.2s of latency between applicatio and DB

Measures

- Response time (max)
- Throughput
- Resident State Size (RSS)

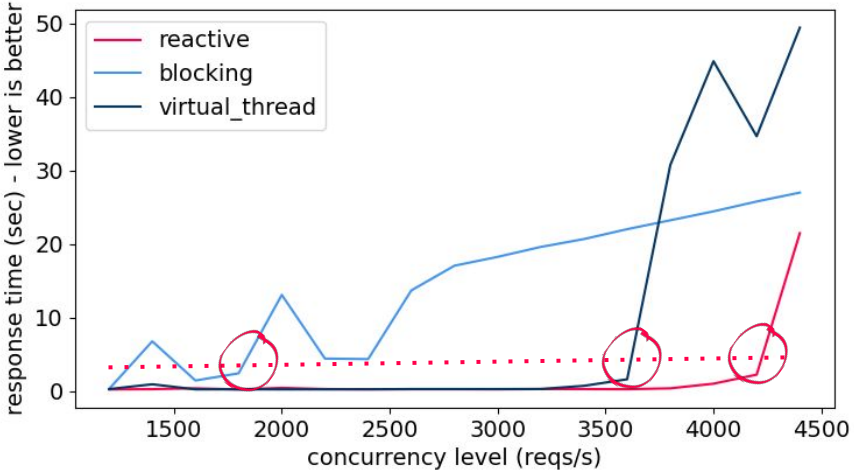


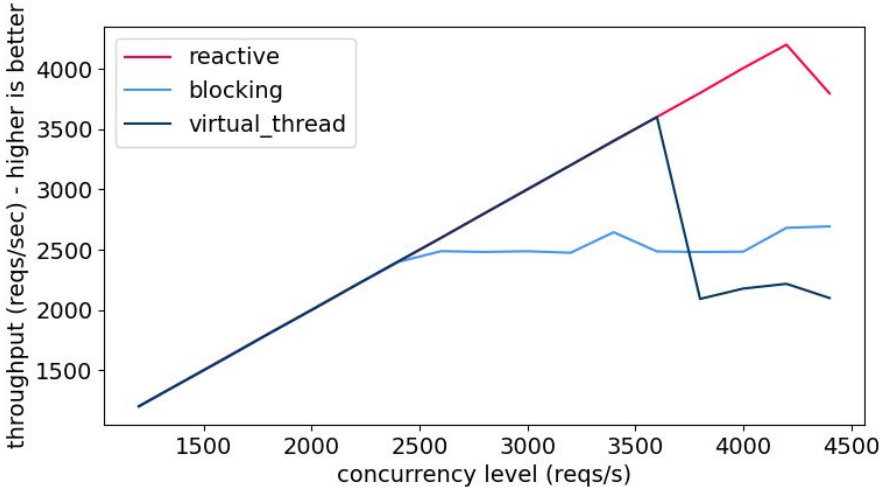
https://github.com/danieloh30/quarkus-todo-app-main



https://github.com/danieloh30/quarkus-todo-app-main

Response time and Throughput

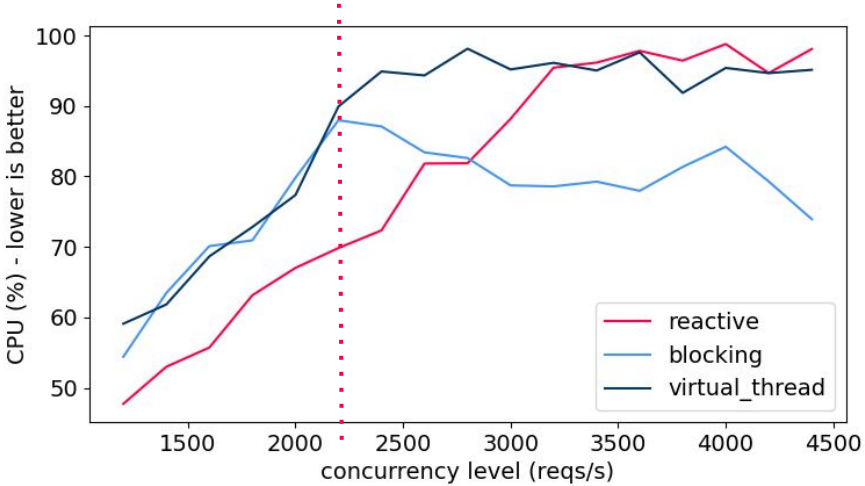


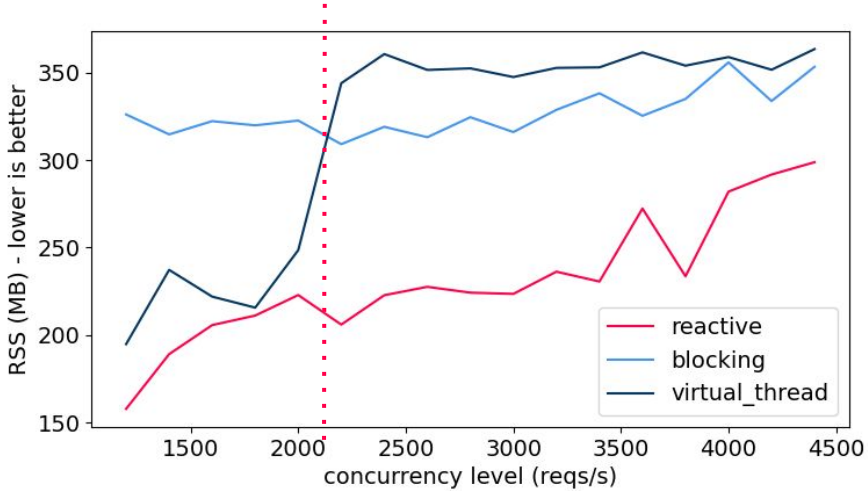


Scale better than worker threads,
but not as good as reactive.

https://github.com/danieloh30/quarkus-todo-app-main

Resource Usage (CPU and RSS)

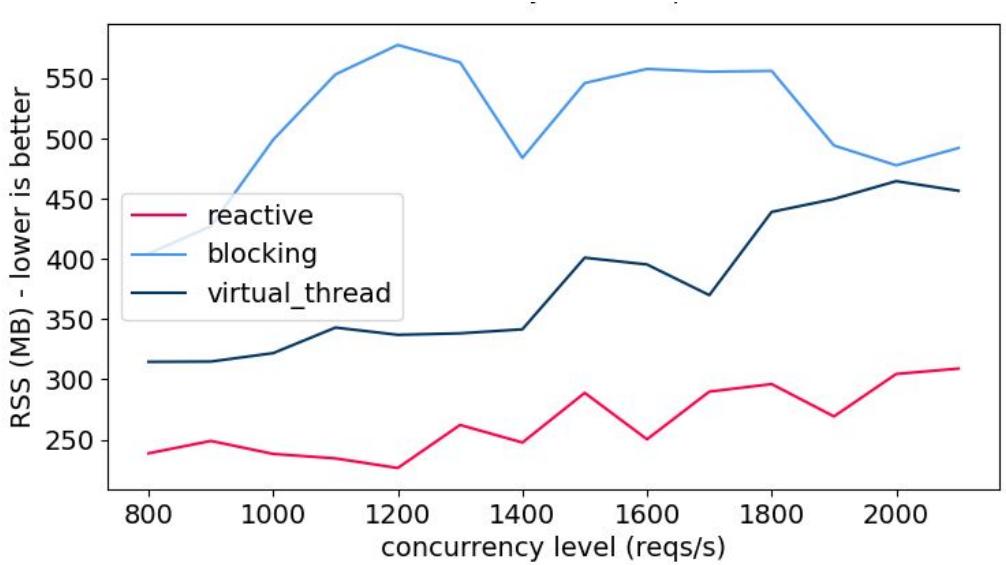




High memory/CPU usage
(higher than worker threads)

https://github.com/danieloh30/quarkus-todo-app-main

Memory Usage - Container



High memory usage
(higher than reactive)

https://github.com/danieloh30/quarkus-todo-app-main

So, what about Virtual Threads?

Imperative

- Simple development model
- Expensive: 1Mb / thread + 10 µs
- Limit concurrency

Virtual Threads

- Simple development model 👌
- Resource efficient 🤞
- High concurrency 👍

Reactive

- More complex development model
- Resource efficient
- High concurrency

Key Takeaways

Virtual Threads in Quarkus

Every reactive APIs can be used

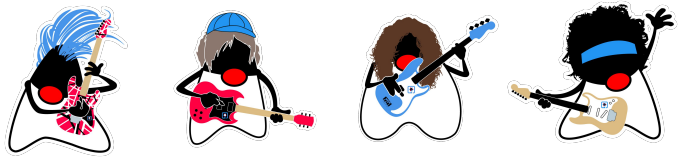
- Databases
- Network client (Rest client, gRPC, GraphQL)
- Kafka, Pulsar,AMQP
- Template engine, Email...

Roadmap

- Web Sockets
- CDI Async Observer

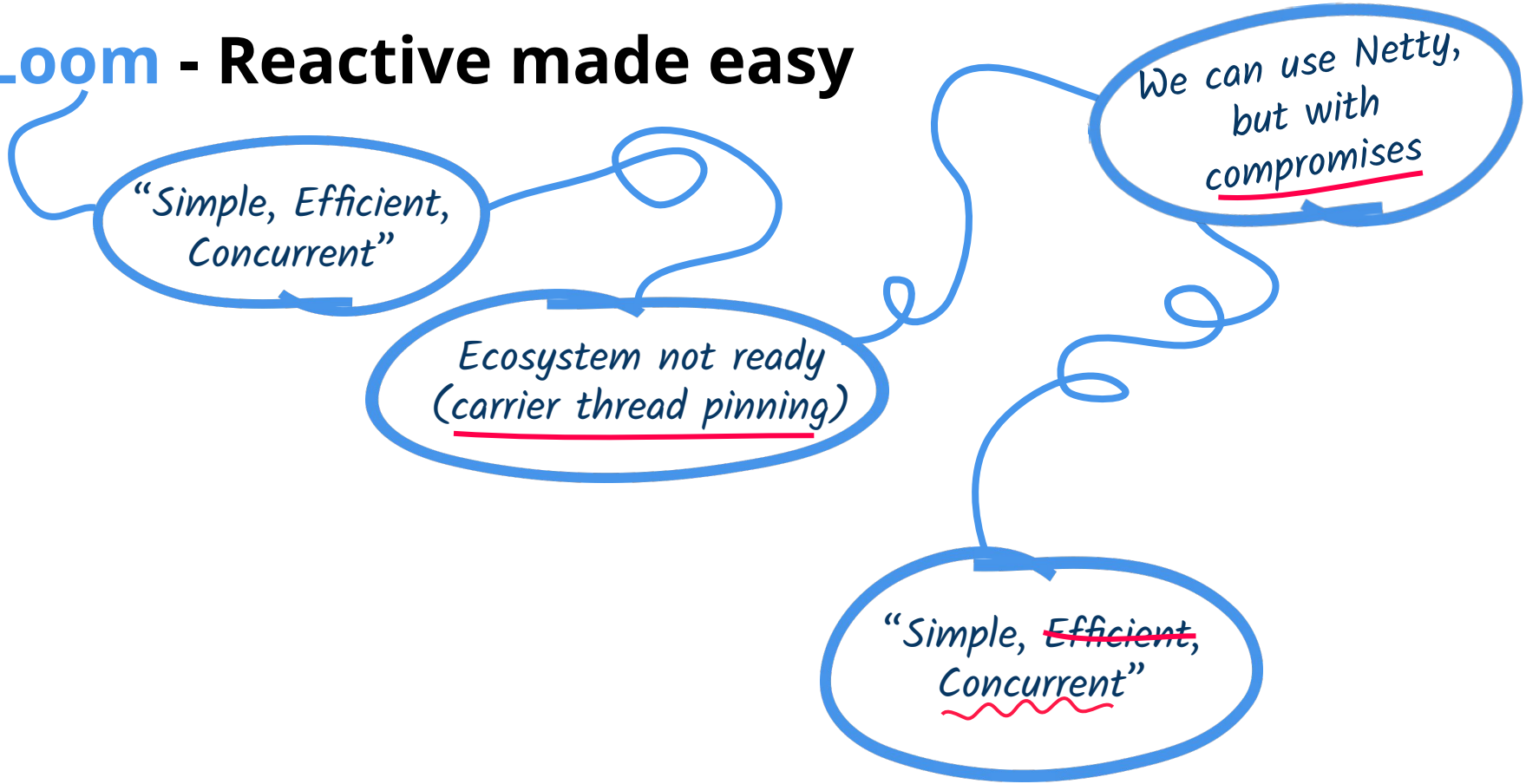
How do use Virtual Threads in Quarkus

- JDK 21
- Use **@RunOnVirtualThread**



Quarkus 3: The Road to Loom for Cheaper, Faster, and Easier Concurrent Applications?

Loom - Reactive made easy

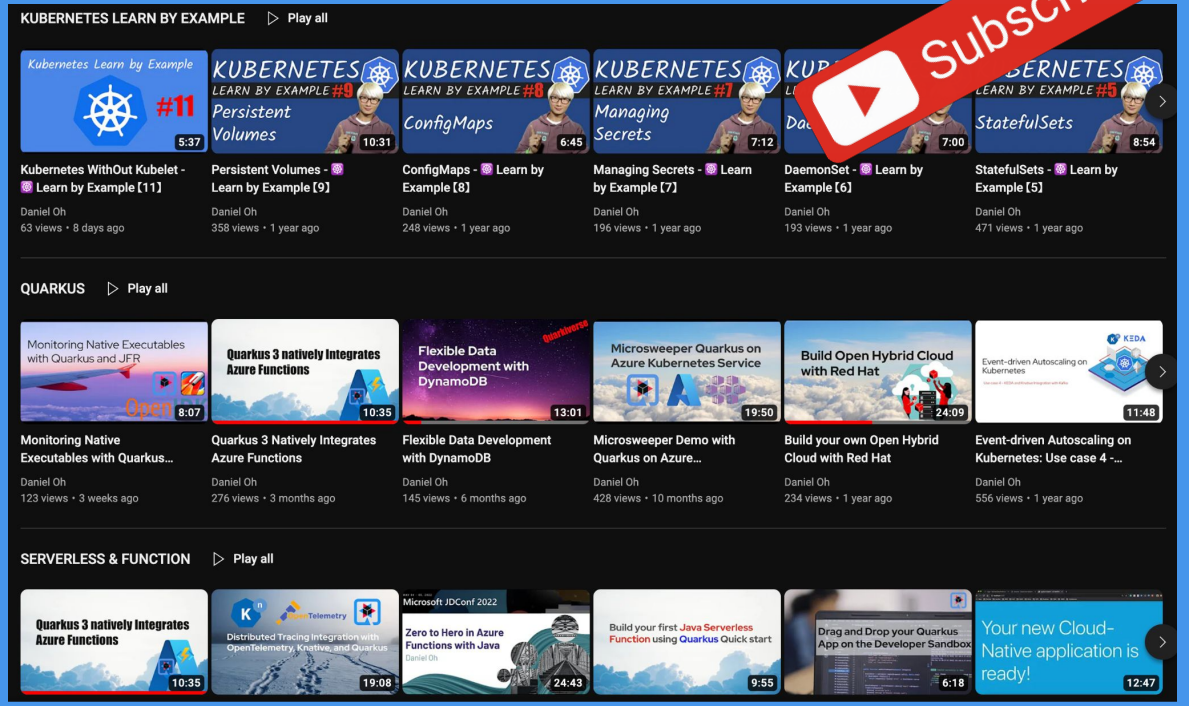


Learn More

Quarkus Virtual Threads

- REST / HTTP
- Hibernate, Bean Validation, Transactions
- Kafka, AMQP, JMS
- gRPC
- Scheduled tasks
- Event Bus
- ...

youtube.com/@danieloh30
bit.ly/danielohtv




Thank you!
Questions?
Project Loom