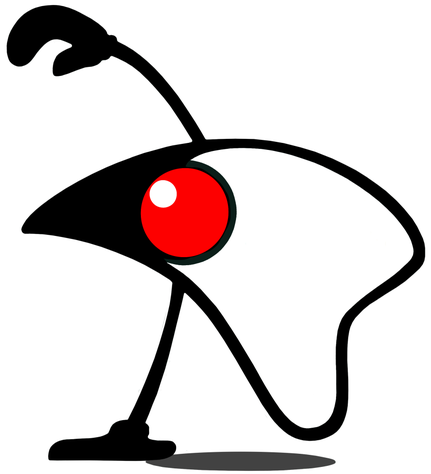
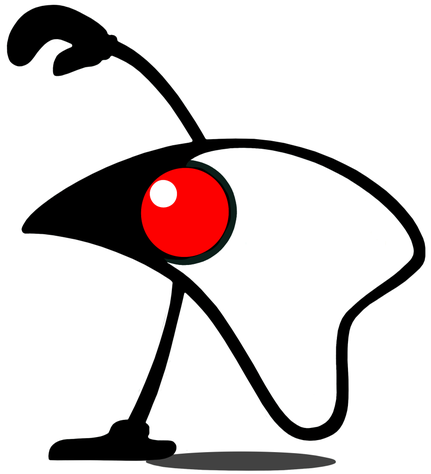


The Java Sessions

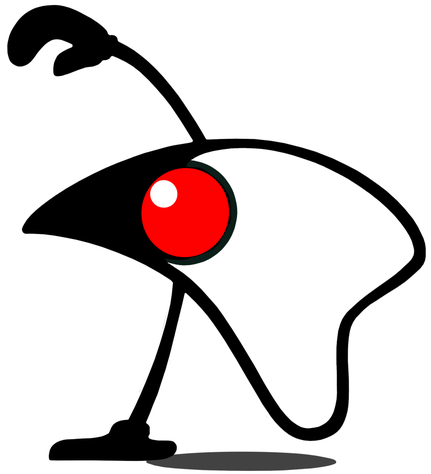
Reactive API

Daniel Hinojosa
@dhinojosa

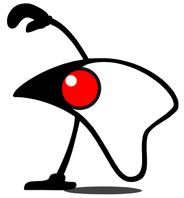


Slides and Code Available

https://github.com/dhinojosa/
java-sessions-reactive

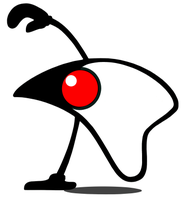


About Flow API



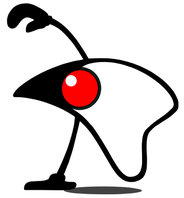
Flow API

- JEP 266 (http://openjdk.java.net/jeps/266)
- Minimal set of interfaces that capture the heart of asynchronous publication and subscription
- Interfaces contain that of what is in reactive-streams.org (Though they are not the same)



Components of Reactive API

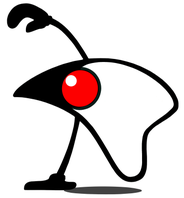
- Publisher
- Subscriber
- Processor
- Subscription



Publisher

- The `Publisher` publishes the stream of data items to the registered subscribers.
- It publishes items to the subscriber asynchronously, normally using an `Executor`.
- Publishers ensure that Subscriber method invocations for each subscription are strictly ordered.

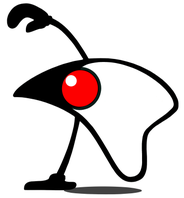
https://community.oracle.com/docs/DOC-1006738



Subscriber

- The Subscriber subscribes to the `Publisher` for the callbacks.
- Data items are not pushed to the `Subscriber` unless requested, but multiple items may be requested.
- `Subscriber` method invocations for a given Subscription are strictly ordered.
- The application can react to the following callbacks, which are available on the subscriber - `onNext`, `onError`, `onComplete`

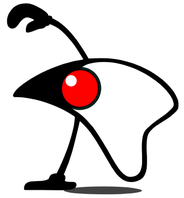
https://community.oracle.com/docs/DOC-1006738



Subscription

- Links a `Flow.Publisher` and `Flow.Subscriber`.
- `Subscribers` receive items only when requested, and may cancel at any time, via the `Subscription`

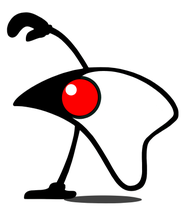
https://community.oracle.com/docs/DOC-1006738



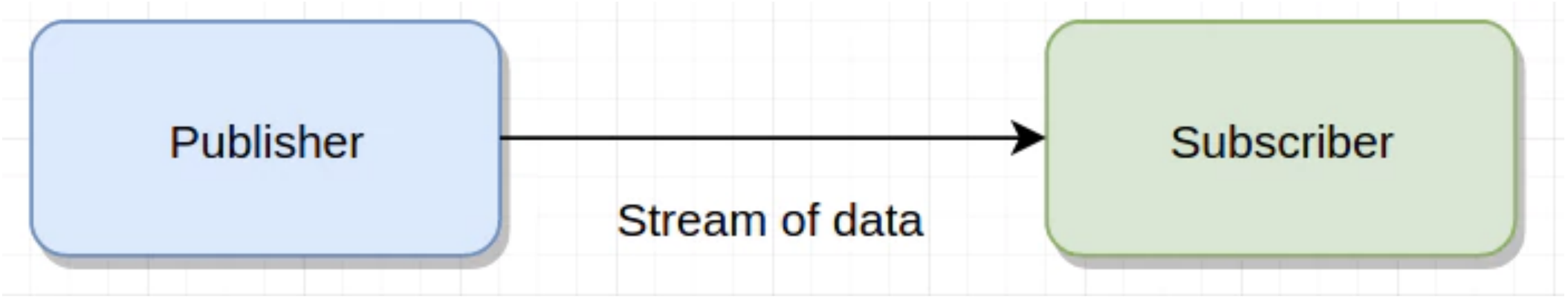
Processor

- A component that acts as both a `Subscriber` and `Publisher`.
- The processor sits between the `Publisher` and `Subscriber`, and transforms one stream to another.
- There could be one or more processor chained together, and the result of the final processor in the chain, is processed by the `Subscriber`.
- The JDK does not provide any concrete `Processors` so it is left up to the individual to write whatever processor one requires.

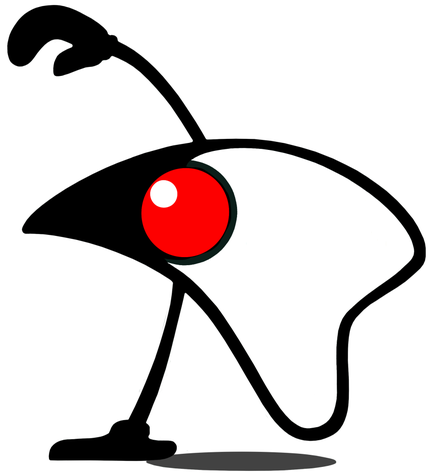
https://community.oracle.com/docs/DOC-1006738



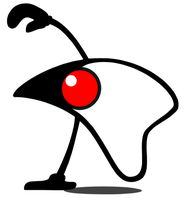
Java 9 Flow Push Model



https://community.oracle.com/docs/DOC-1006738

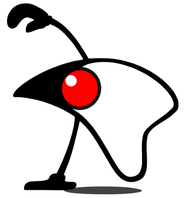


About Backpressure



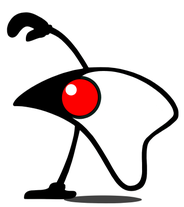
Backpressure Definition (Pipes)

Back pressure refers to pressure opposed to the desired flow of a fluid in a confined place such as a pipe. It is often caused by obstructions or tight bends in the confinement vessel along which it is moving, such as piping or air vents.

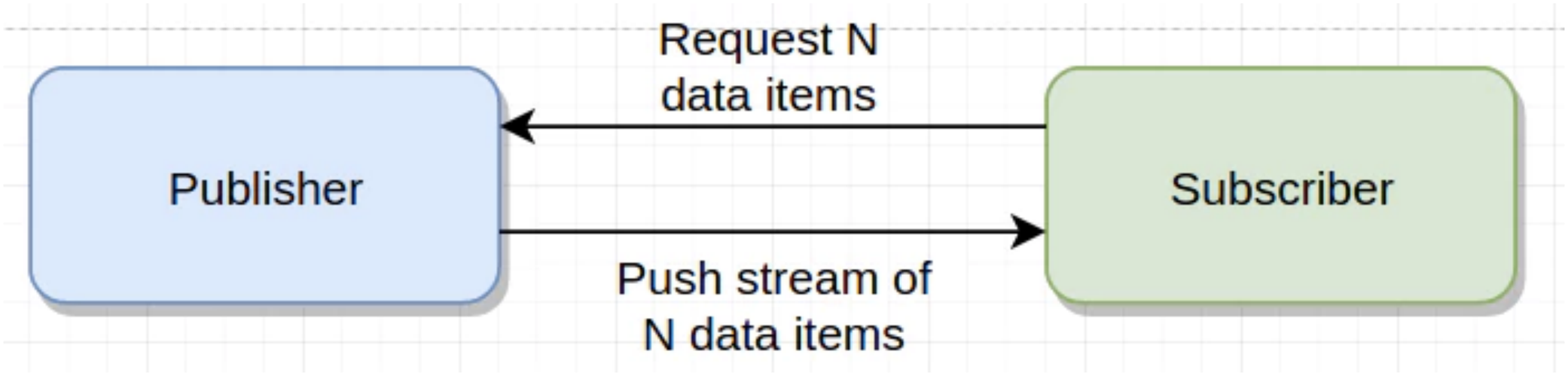


Backpressure Definition (Software)

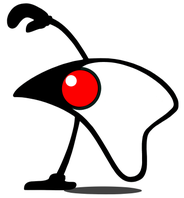
Back pressure is when an asynchronous source is emitting items more rapidly than an operator or subscriber can consume them. This presents the problem of what to do with such a growing backlog of unconsumed items.



Java 9 Flow Push Model with Backpressure

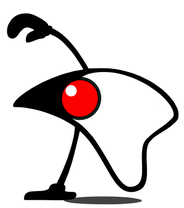


https://community.oracle.com/docs/DOC-1006738

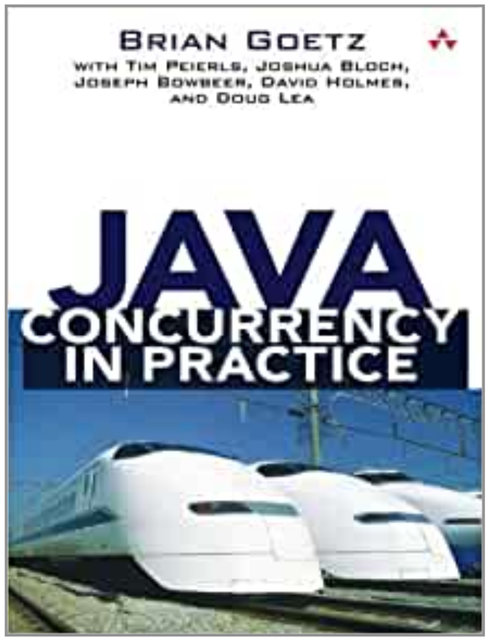


Flow Backpressure

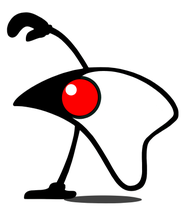
The Flow API does not provide any APIs to signal or deal with back pressure as such, but there could be various strategies one could implement by oneself to deal with backpressure



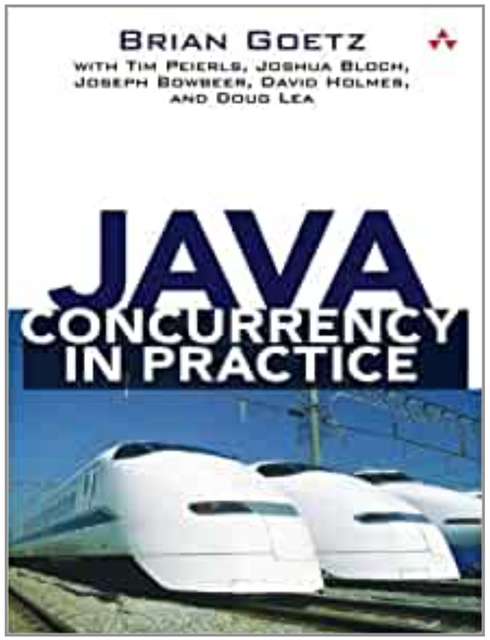
Backpressure by Brian Goetz



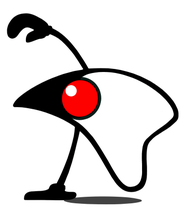
“"Backpressure" is just a fancy new term for the age-old concept of bounding resource utilization by stalling or refusing excessive requests. Examples include:”



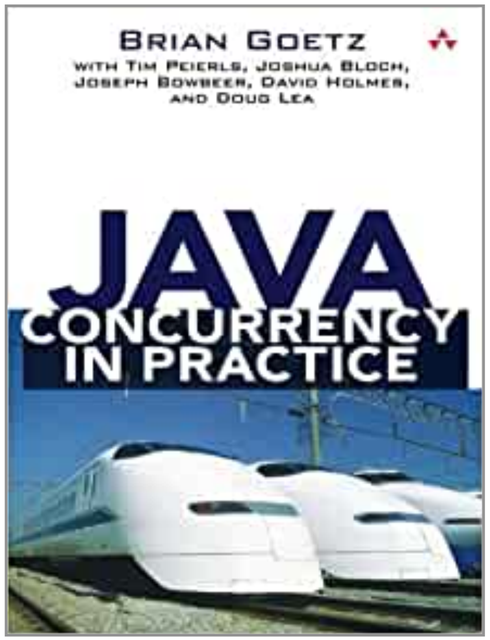
Backpressure by Brian Goetz



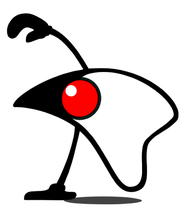
“thread pools -- limit the number of concurrently executing tasks”



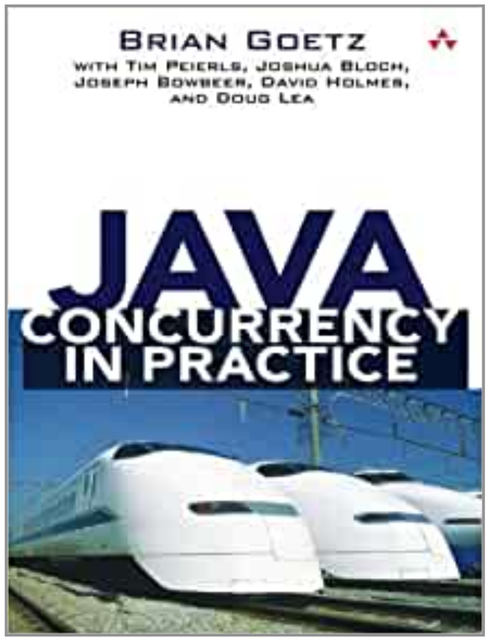
Backpressure by Brian Goetz



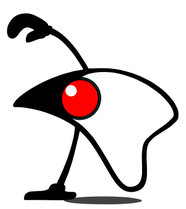
“semaphores -- limit the number of a critical resource (socket connections, open files, etc), by stalling incremental requests until the resource becomes free”



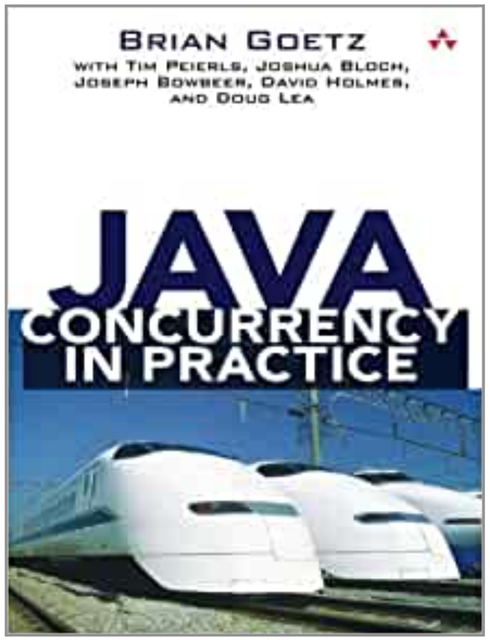
Backpressure by Brian Goetz



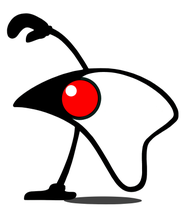
“producer/consumer with blocking queues -- when consumers are overloaded, stall the producers”



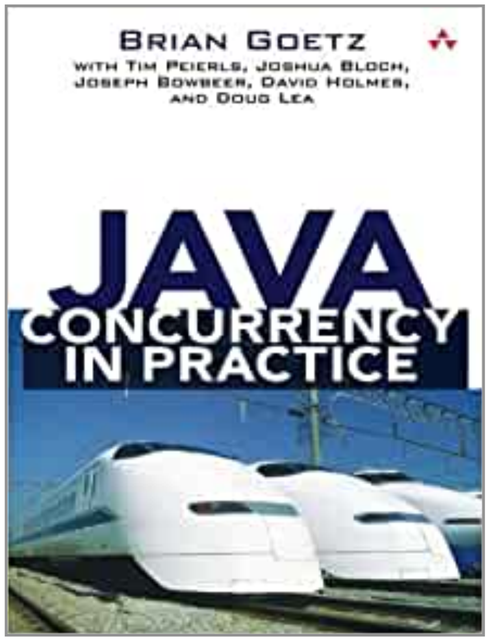
Backpressure by Brian Goetz



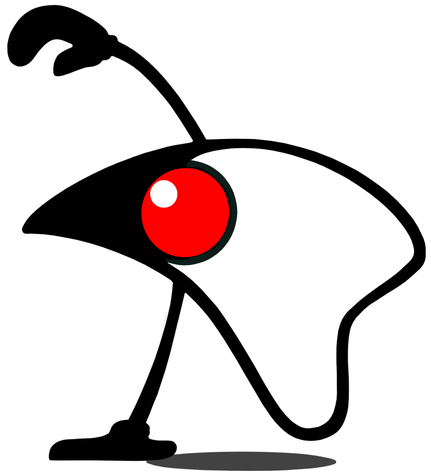
“queuing of socket connect requests in the OS, and refusing additional requests after the queue gets too long”



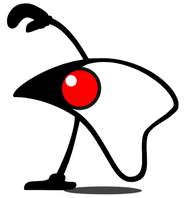
Backpressure by Brian Goetz



“various networking flow control protocols using send credits, sliding windows, etc (xmodem, TCP)”

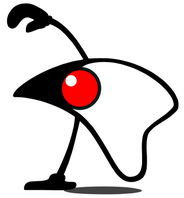


Reactive Programming?



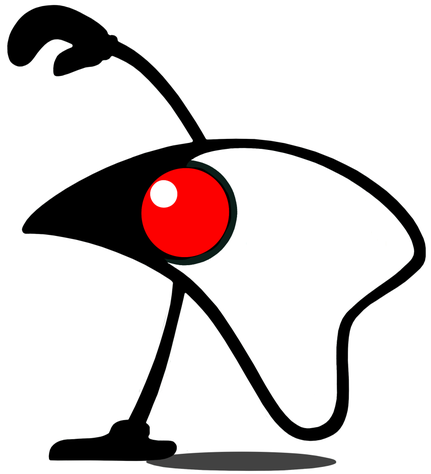
“Reactive Programming”

“Reacting to changes in data or events using functional abstractions”



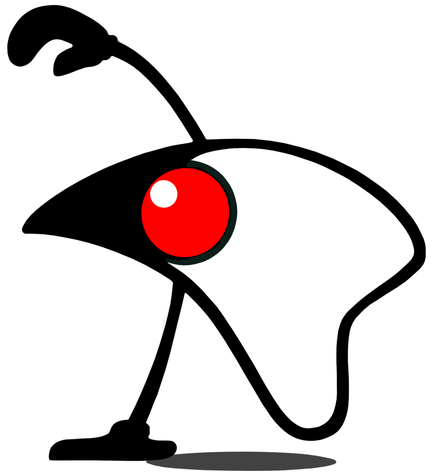
“Reactive Programming”

Responding to:
Mouse Clicks, Keyboard Events, Temperature Sensors, Stock Trades, Website Clicks, GPS Signals, HTTP Responses, MQ Messages, Kafka Messages, etc.

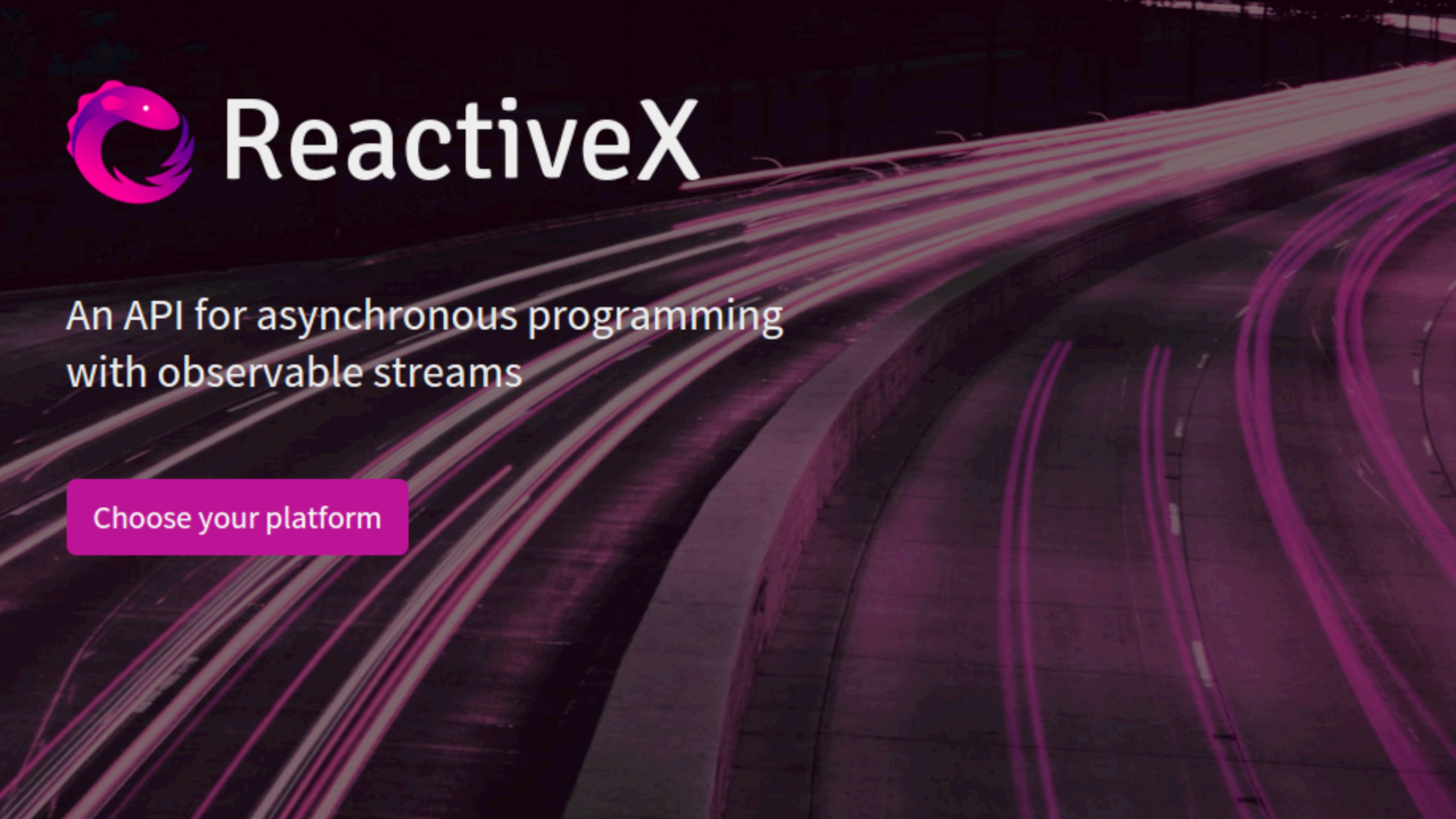


Demo & Challenges

Using the Flow API

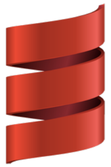


RXJava

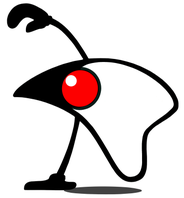

ReactiveX
An API for asynchronous programming
with observable streams
Choose your platform

Reactive Extensions for Async Programming



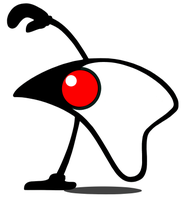



RXJava



What is RXJava?

A library for composing asynchronous and event-based programs by using observable sequences.



Unbiased

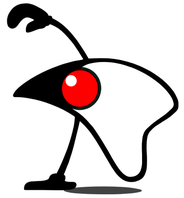
The source for any `Observable/Flowable` can from anywhere. Convert your arrays, futures, threads, into `Observables/Flowable`

formerly of...

Microsoft

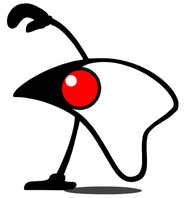
“The mouse is a database”

NETFLIX



More about RXJava

- Push Model (Reactive)
- Pull Model (Interactive, Subscription Based)
- Lazy
- Can be Asynchronous or Synchronous
- Combination Observer + Iterator

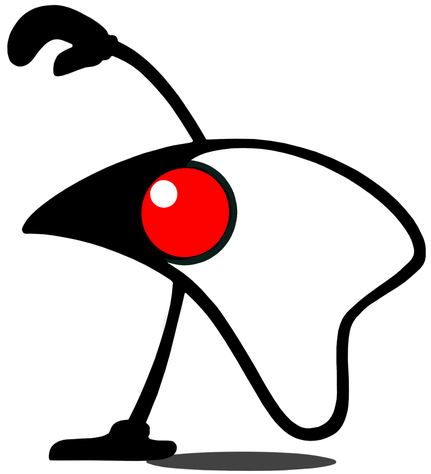


Where can you use Reactive Libraries?

Anywhere. It is a processing asynchronous glue in applications.

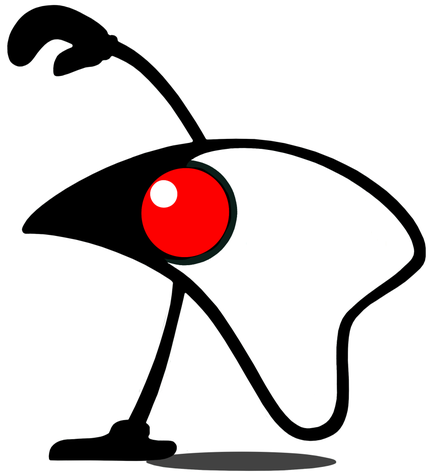
Find all your favorite operators at:

http://reactivex.io/documentation/operators



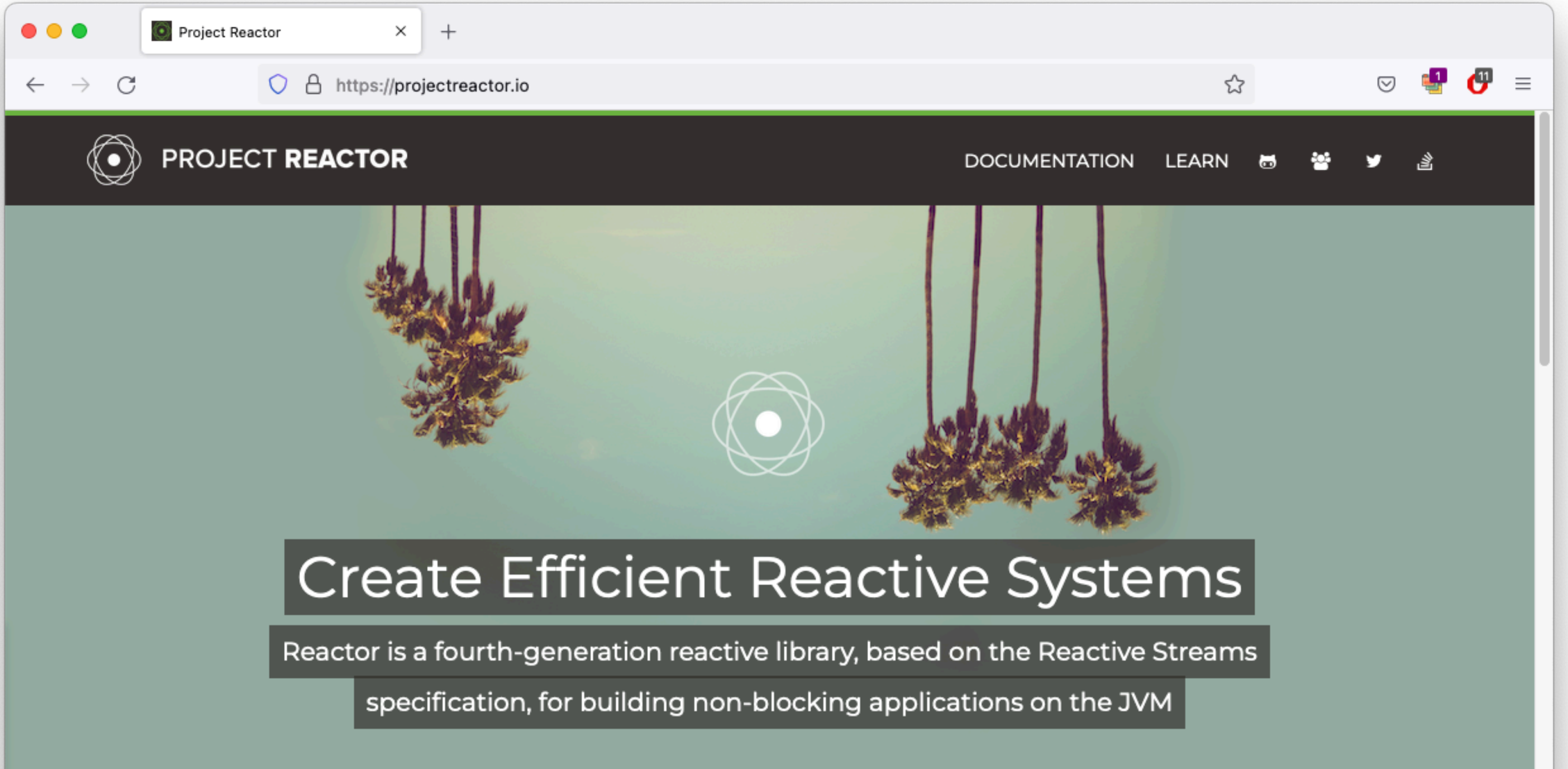
Demo & Challenges

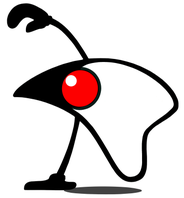
Using RXJava



Project Reactor

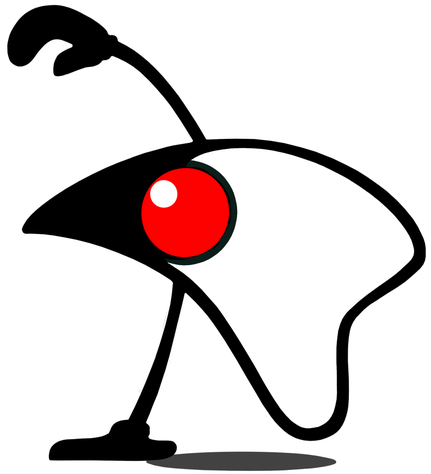
https://projectreactor.io/





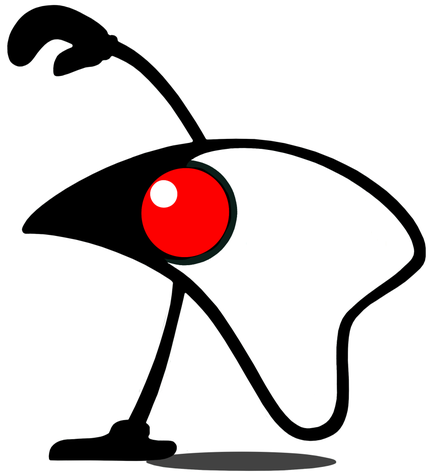
More about Project Reactor

- Push Model (Reactive)
- Pull Model (Interactive, Subscription Based)
- Lazy
- Can be Asynchronous or Synchronous
- Combination Observer + Iterator
- Integrates well with Java 8 API
- Included with Spring 5

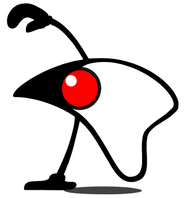


Demo & Challenges

Using Project Reactor

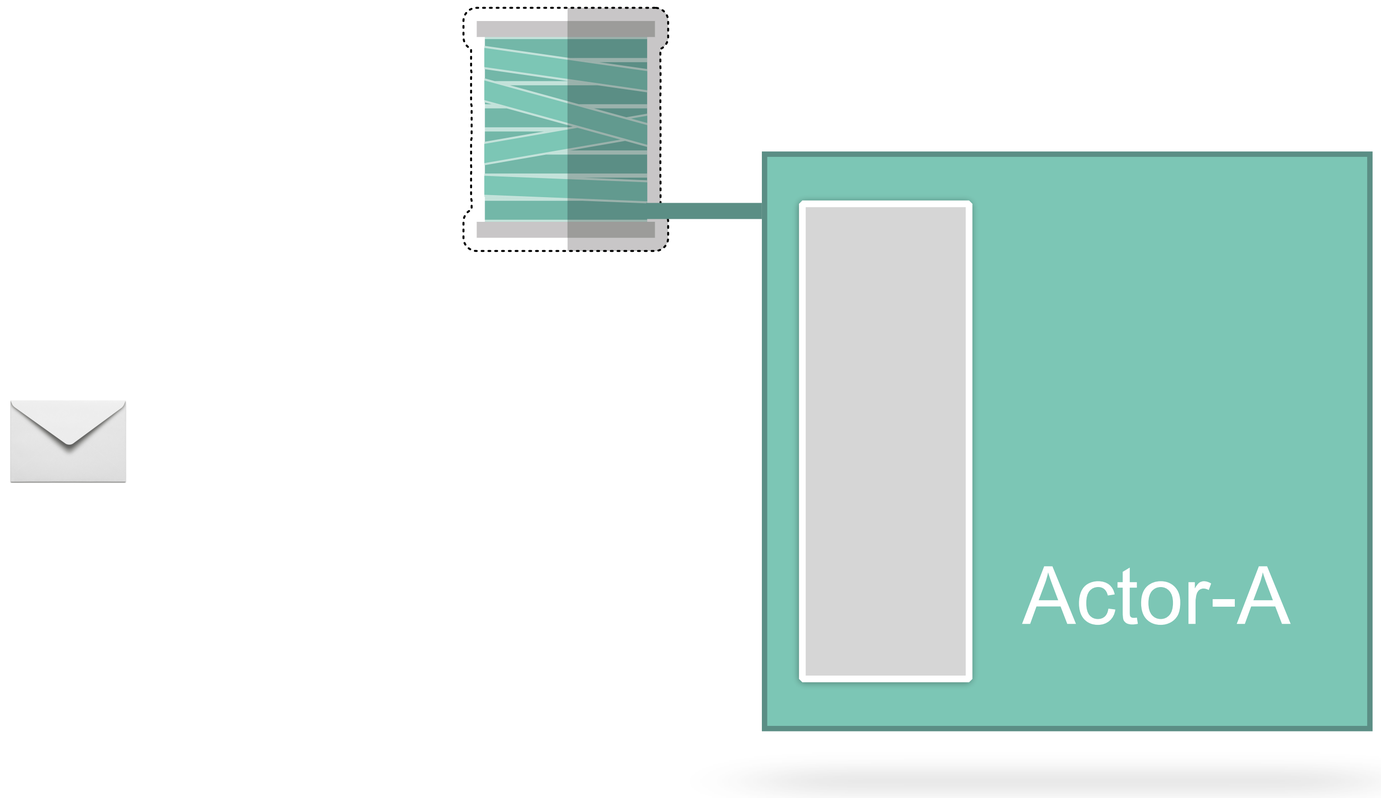


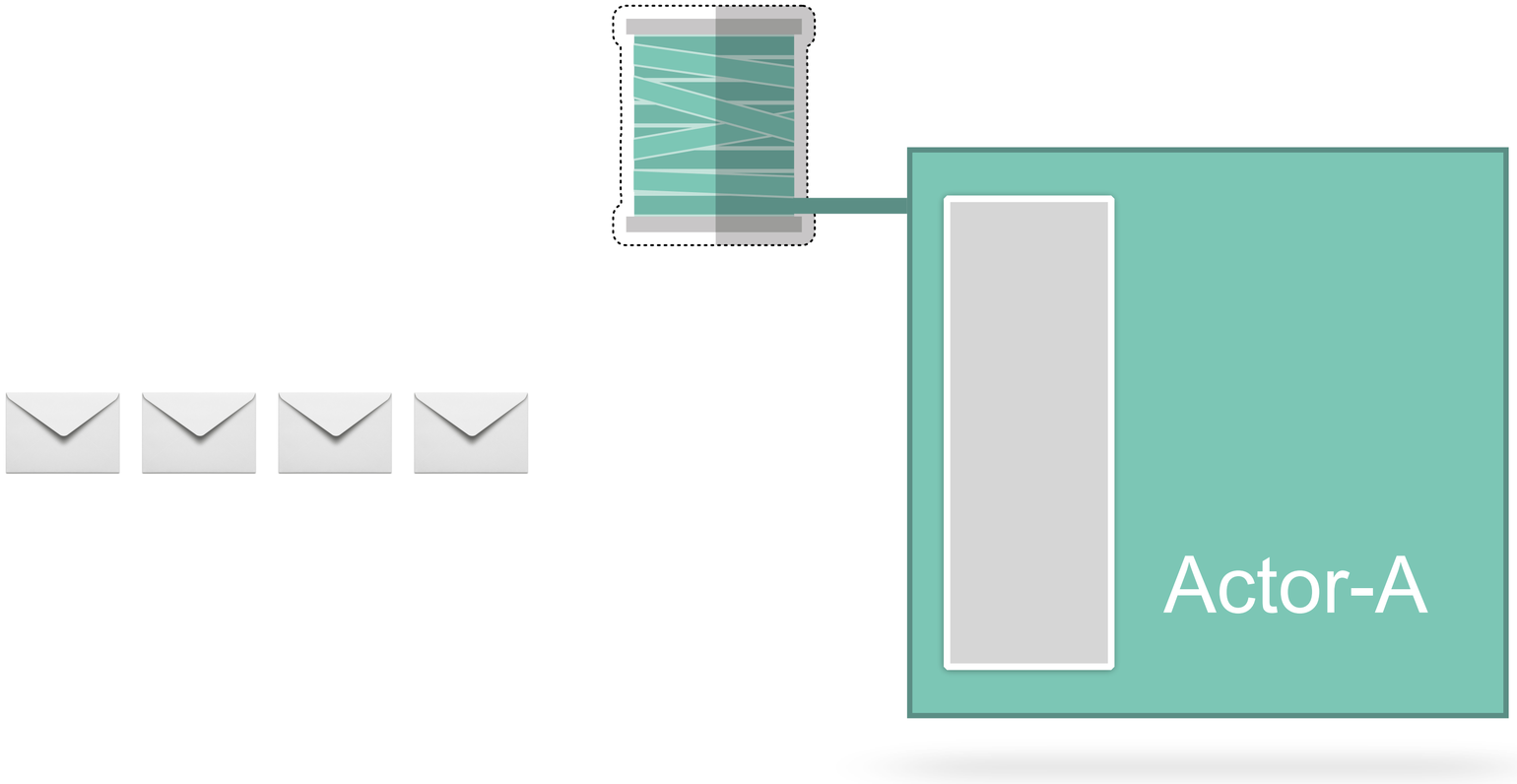
Akka Streams

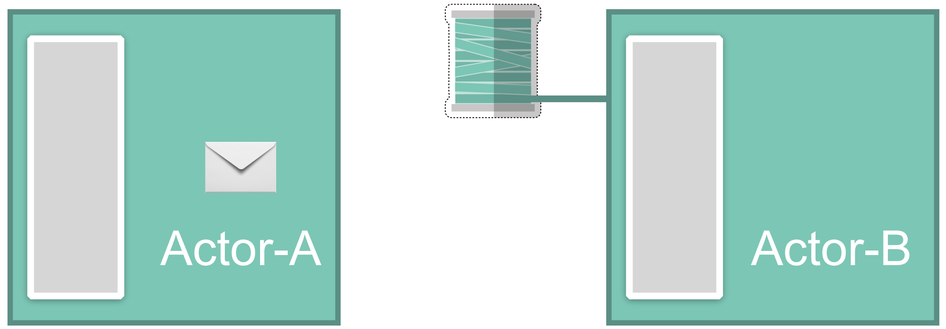


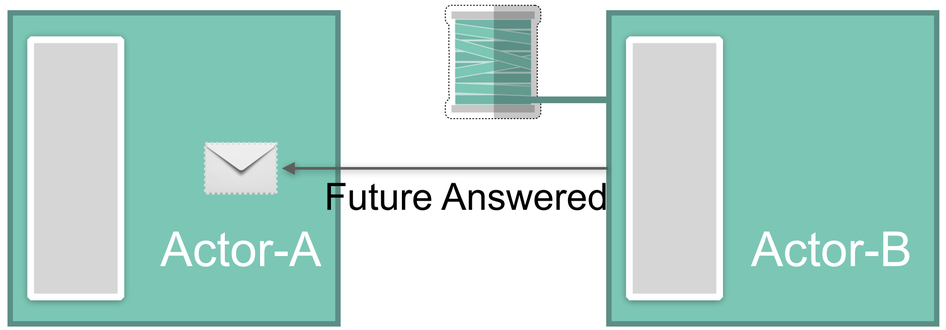
About Akka Streams

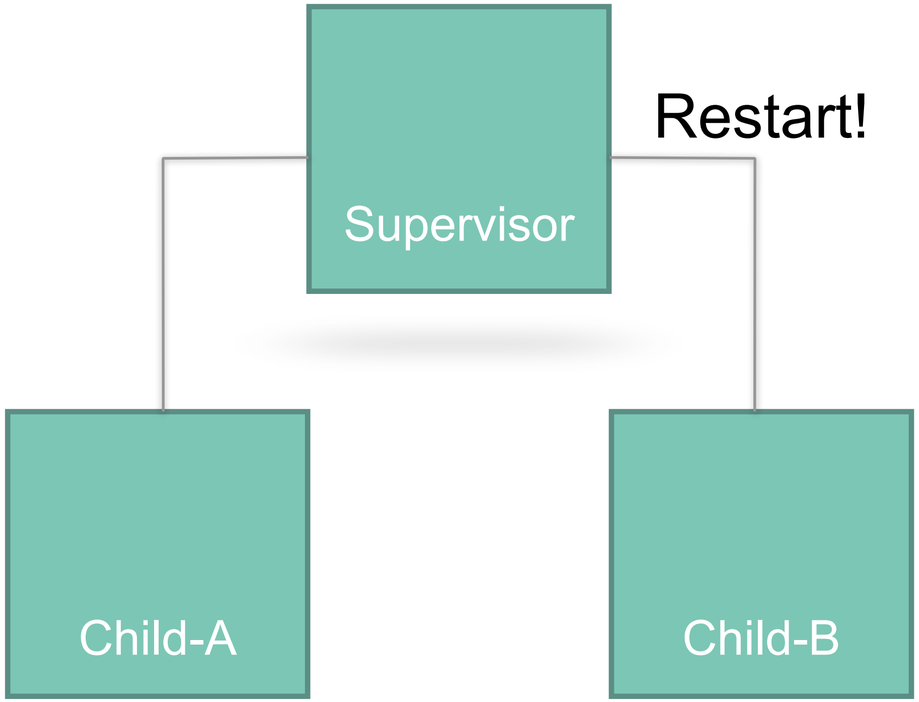
- Backed by Actor Systems and Model
- Stream processors much like their competitors RXJava and Project Reactor
- Builds on the idea of flows and flow graphs that define how a stream is processed
- Streams are built with reusable pieces
  - Prepackaged components
  - Custom made composites


Actor-A


Actor-A

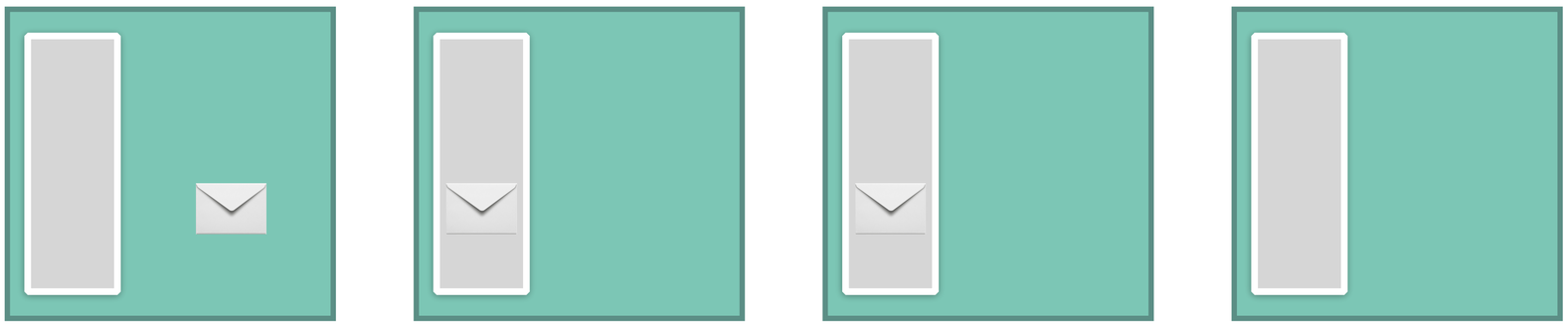

Actor-A
Actor-B

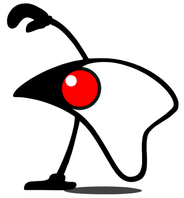

Future Answered
Actor-A
Actor-B



`IllegalStateException`

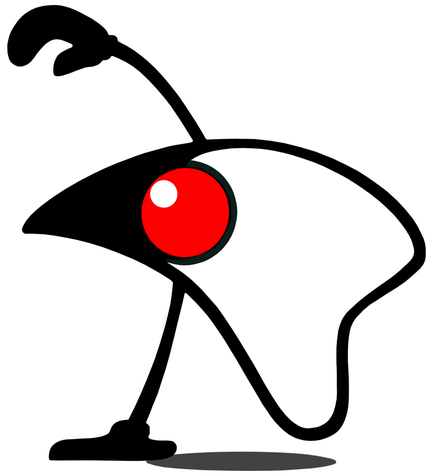
Source[Int] .map(...) filter(...) to(Sink)





About Akka Alpakka

- Stream-aware and reactive integration pipelines for Java and Scala
- List of systems that integrate with Akka Streams
- Reactive Streams and JDK 9+ `java.util.concurrent.Flow`-compliant
- https://doc.akka.io/docs/alpakka/current/



Demo

Using Akka Streams

Akka Alpakka

https://doc.akka.io/docs/alpakka/current/