Reactive Streams

because parallelism matters

@humbertostreb

Humberto Streb

- software developer
- falling in love with distributed systems
- half-assed goalkeeper ;(

fb.com/humbertostreb

twitter.com/humbertostreb

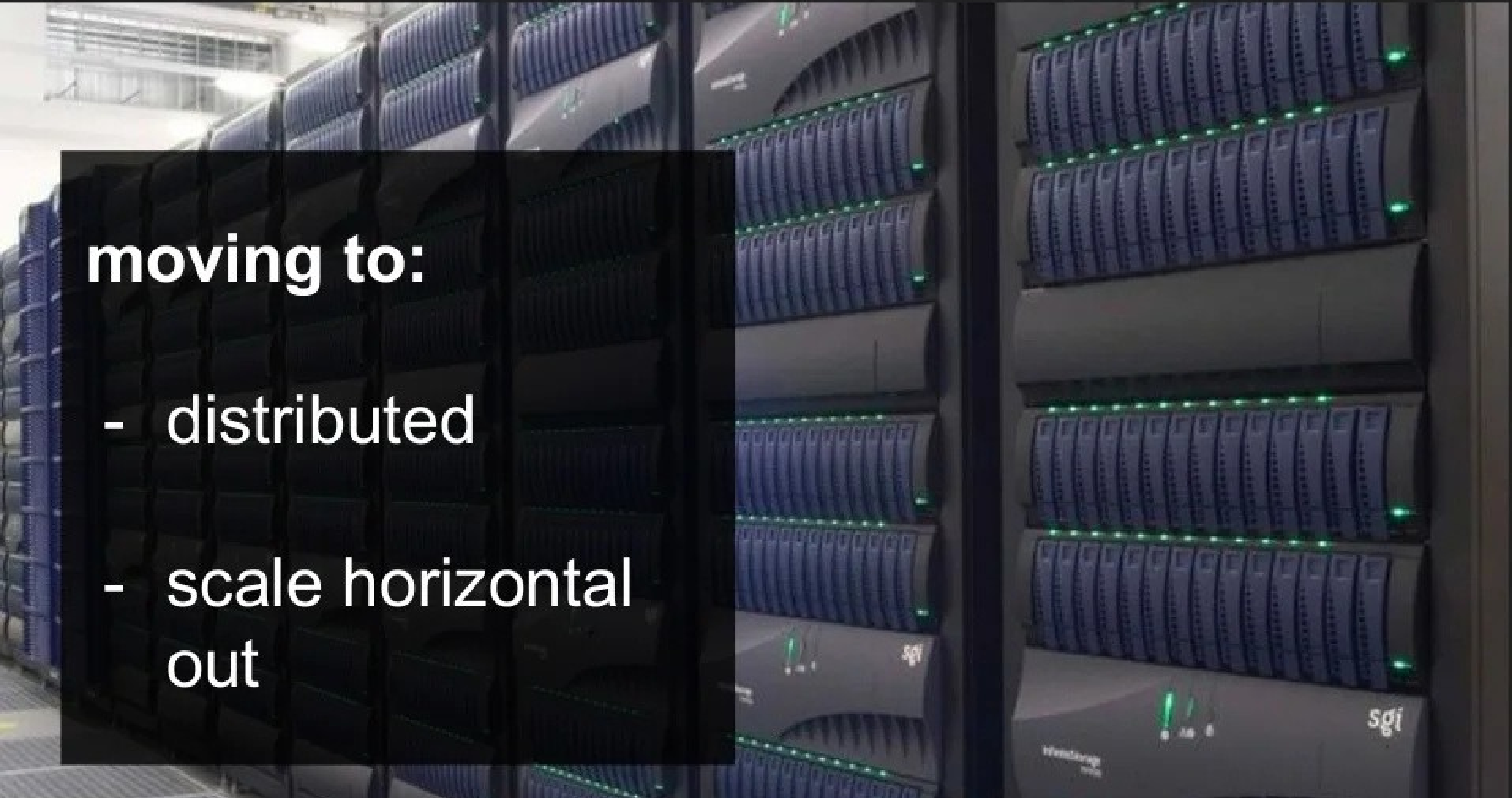
github.com/hstreb

history

From:

- single node

- vertical scale up



moving to:

- distributed
- scale horizontal out

From:

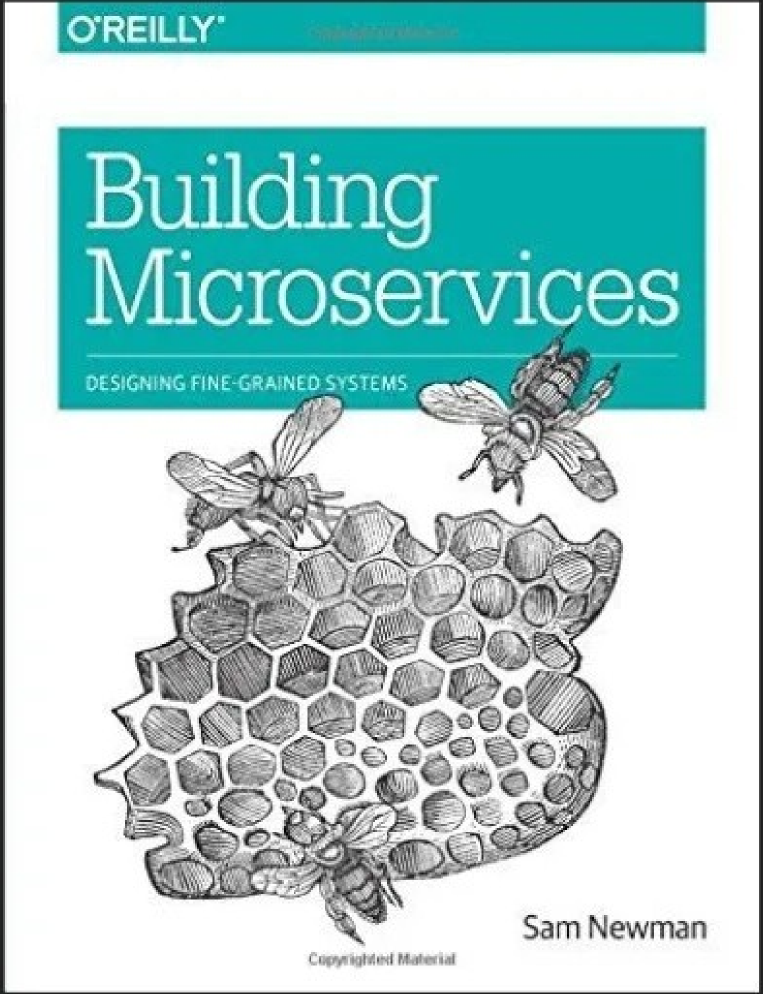
Applications Servers

JBoss

GlassFish

moving to:

Microservices



O'REILLY

Building Microservices

DESIGNING FINE-GRAINED SYSTEMS

Sam Newman

What I need to change in my codebase?

Async - Threads

Threads are powerful, but:

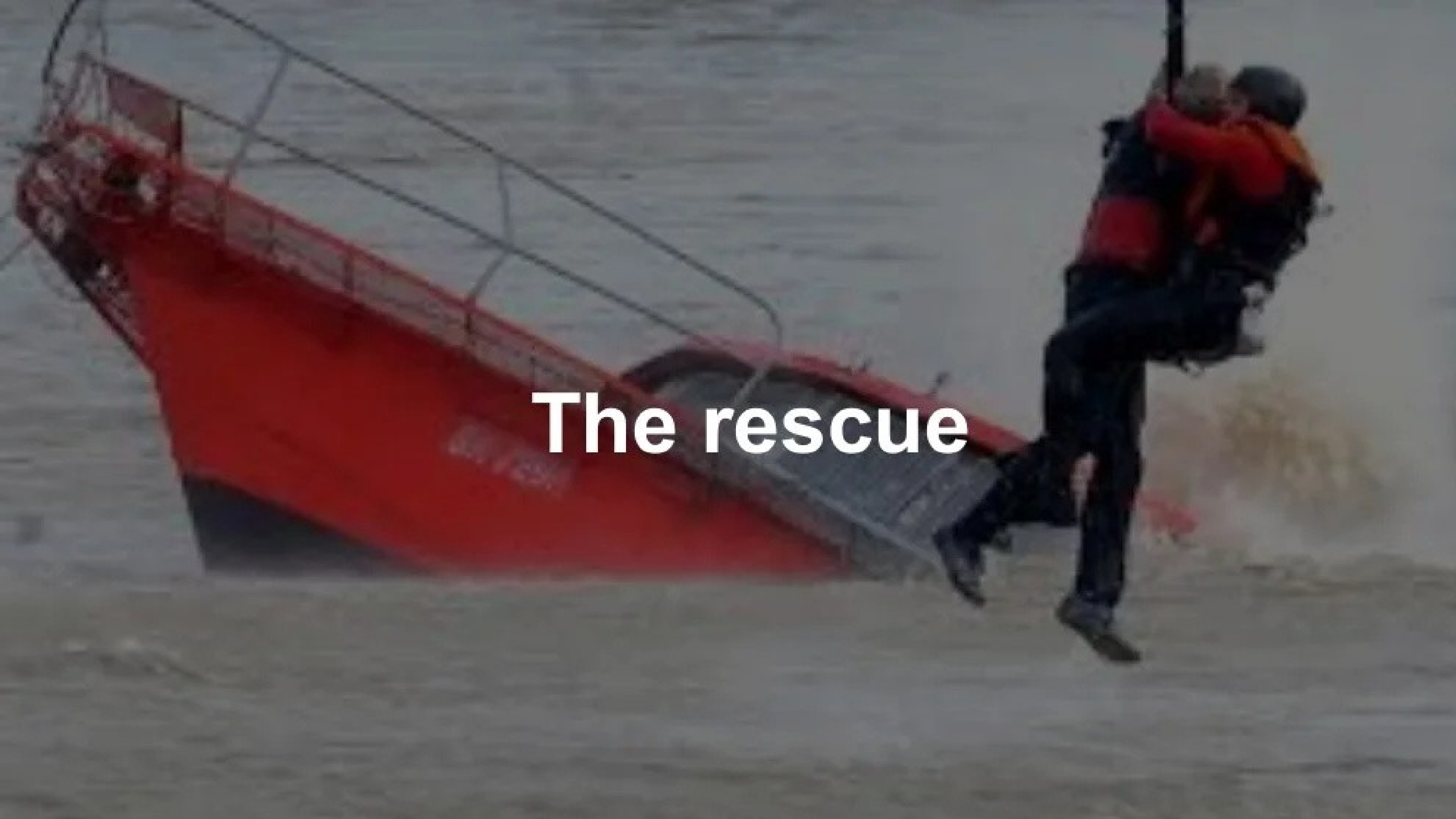
- low level abstraction
- imperative style
- demand more knowledge from the java API (lock, synchronized, ...)

lock

```
final ReentrantReadWriteLock lock = new ReentrantReadWriteLock();
lock.writeLock().lock();
try {
    variable.complexAlgorithm();
} finally {
    lock.writeLock().unlock();
}
```

synchronized

```
synchronized(variable) {
    variable.complexAlgorithm();
}
...
sychronized(a.variable) {
    consume(a.variable);
}
```



The rescue

Reactive Streams

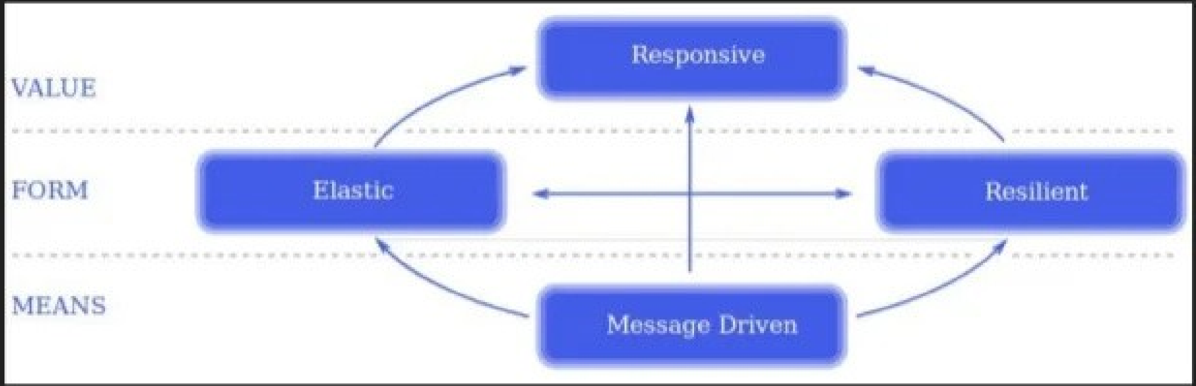
Reactive Streams

- asynchronous stream processing
- non-blocking IO
- back pressure

http://www.reactive-streams.org/

Reactive Manifesto

- responsive
- resilient
- elastic
- message driven



https://reactivemanifesto.org

From Imperative to Reactive Programming

- Composability and readability
- Data as a flow manipulated with a rich vocabulary of operators
- Nothing happens until you subscribe
- Backpressure
- High level but high value abstraction that is concurrency-agnostic

Implementations

- akka streams
  - based in actors
  - rich environment (akka)
    - cluster
    - TCP
    - akka persistence
    - Alpakka

Implementations

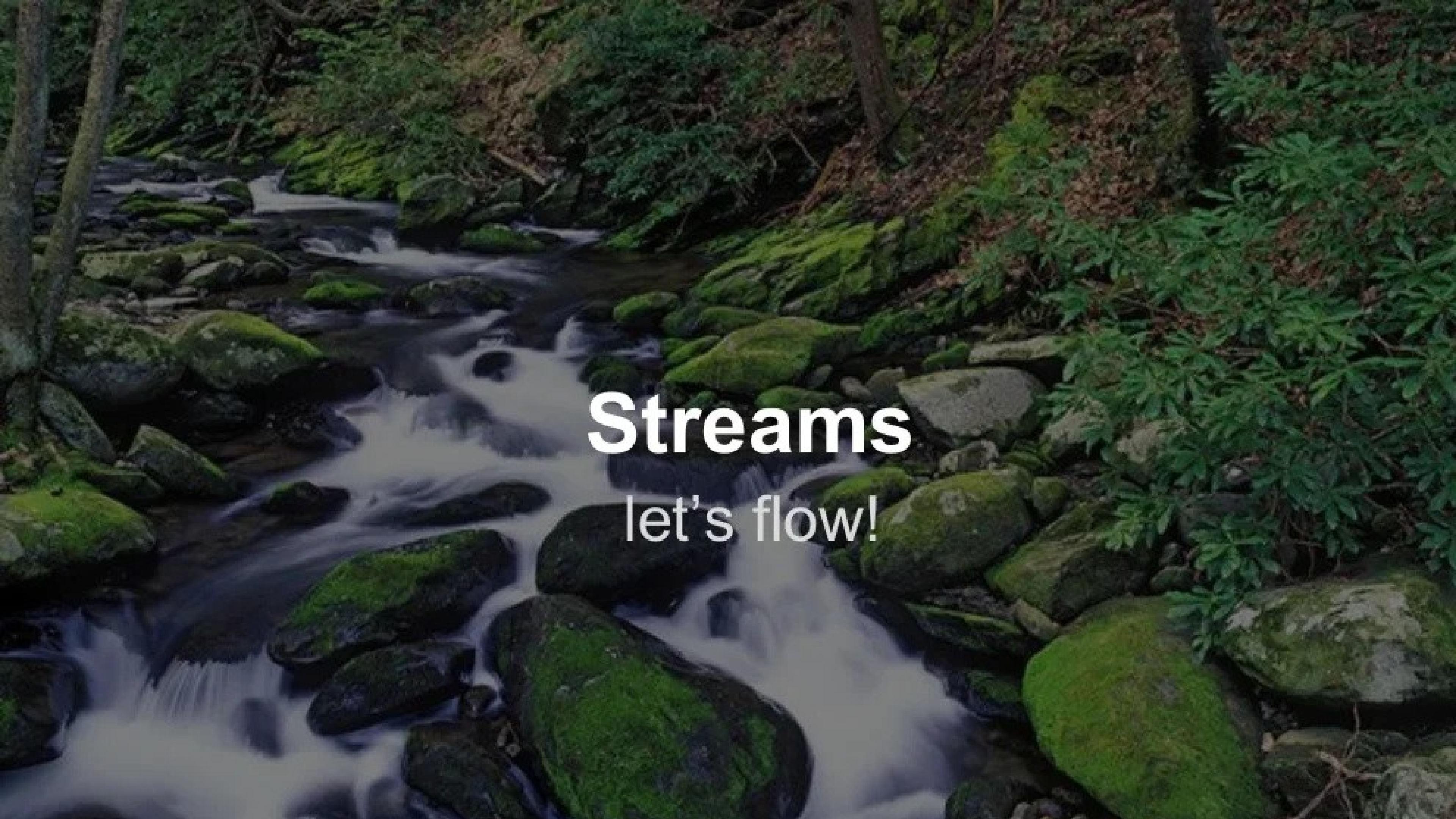
- RxJava
  - netflix oss

Implementations

- Reactor
  - Spring support

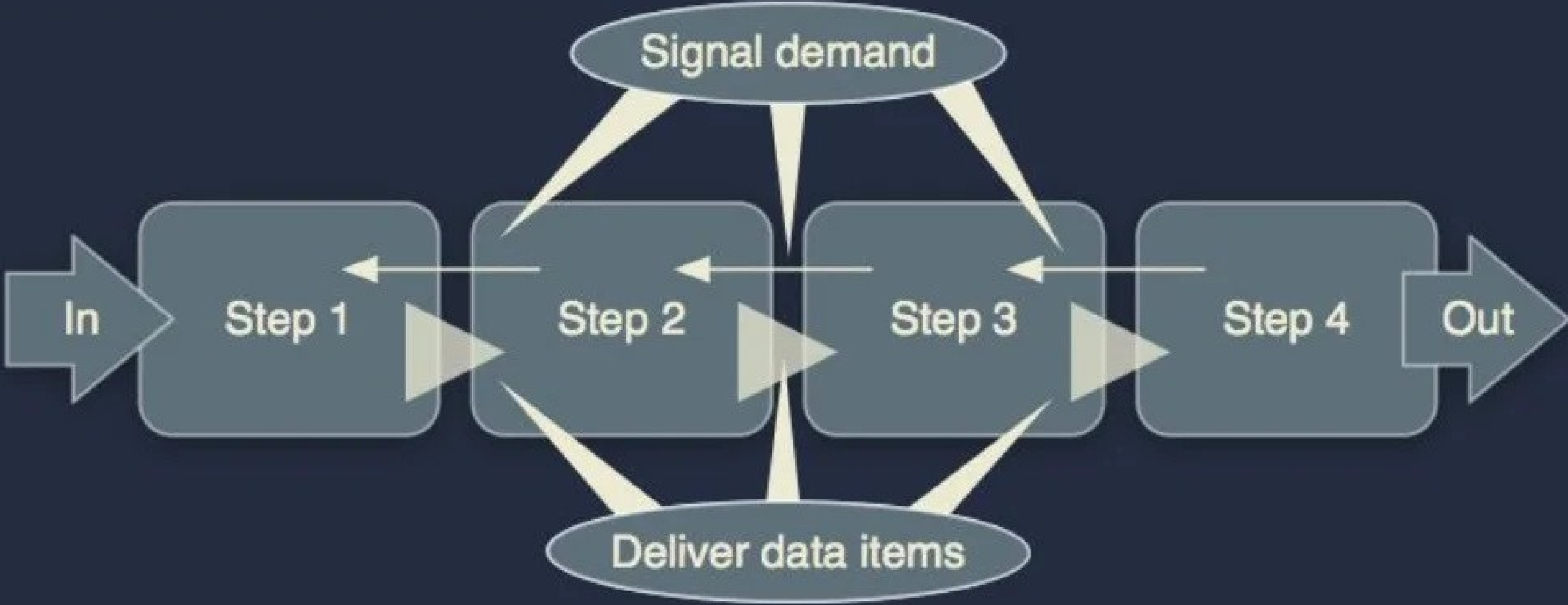
Implementations

- Java 9



Streams

let’s flow!


Signal demand
In
Step 1
Step 2
Step 3
Step 4
Out
Deliver data items

When use Reactive Streams

- it’s not about be fast, it’s about be efficient
- racionalize resources consumption
- support high demand

Project Reactor

Reactor

- **Mono<T>** an Asynchronous 0-1 Result
- **Flux<T>** an Asynchronous Sequence of 0-n Items

Creating

```
Mono<Integer> mono = Mono.just(1);
Mono<Integer> monoEmpty = Mono.empty();

Flux<Integer> flux = Flux.range(0, 10);
Flux<String> foo = Flux.just("foo", "bar");
Flux<String> fromIterable = Flux.fromIterable(iterable);
```

Extensive number of functions

- map()
- filter()
- repeat()
- replay()
- retry()
- skip(), skipUntil(), skipeWhile() ...
- switchIfEmpty()
- take(), takeUntil(), takeWhile() ...
- zip()

...

Async

```
Flux.defer(() -> Flux.fromIterable(repository.findAll()))
            .subscribeOn(Schedulers.elastic());


...


flux.publishOn(Schedulers.parallel())
            .doOnNext(repository::save)
            .then();
```

Back pressure

```java
Flux<Long> flux = Flux.interval(Duration.ofMillis(100))
        .take(100);

flux
        .buffer(Duration.ofMillis(10))
        .toIterable()
        .forEach(System.out::println);

flux
        .buffer(Duration.ofMillis(200))
        .toIterable()
        .forEach(System.out::println);
```

```
// faster consumer
[0]
[1]
[2]
…

// slow consumer
[1, 2]
[3, 4]
[5, 6]
...
```

Testing

```
@Test
public void testAppendBoomError() {
  Flux<String> source = Flux.just("foo", "bar");

  StepVerifier.create(
    appendBoomError(source))
    .expectNext("foo")
    .expectNext("bar")
    .expectErrorMessage("boom")
    .verify();
}
```

More

- Spring 5.0 with reactive support

  https://www.brighttalk.com/webcast/14893/277207

- Deal with JDBC blocking

  https://dzone.com/articles/spring-5-webflux-and-jdbc-to-block-or-not-to-block

links

- https://www.reactivemanifesto.org/
- https://blog.redelastic.com/a-journey-into-reactive-streams-5ee2a9cd7e29
- https://www.infoq.com/articles/reactor-by-example
- https://spring.io/blog/2016/07/28/reactive-programming-with-spring-5-0-m1
- http://musigma.org/java/2016/11/21/reactor.html

Thanks

Reactive Streams

because parallelism metters

@humbertostreb