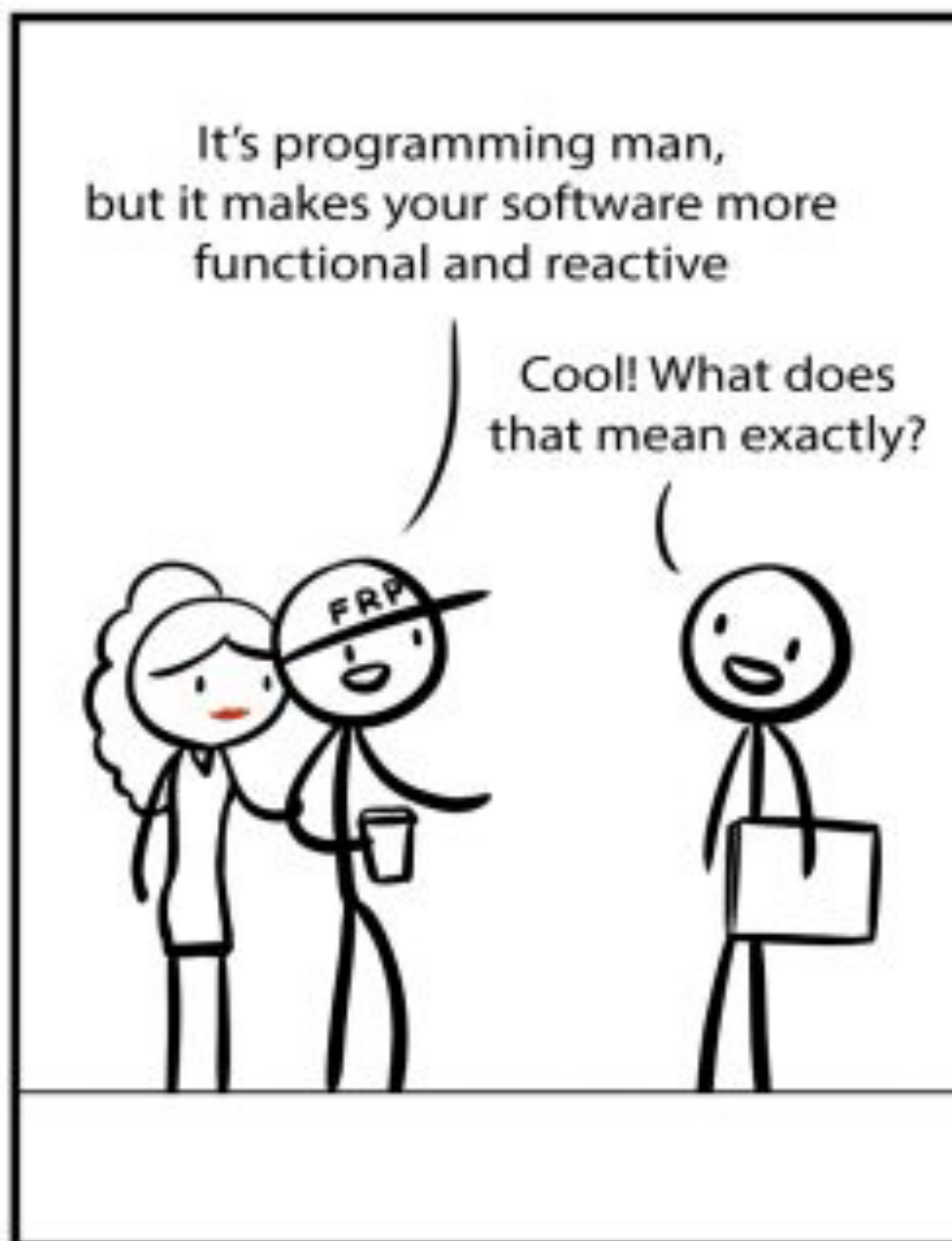
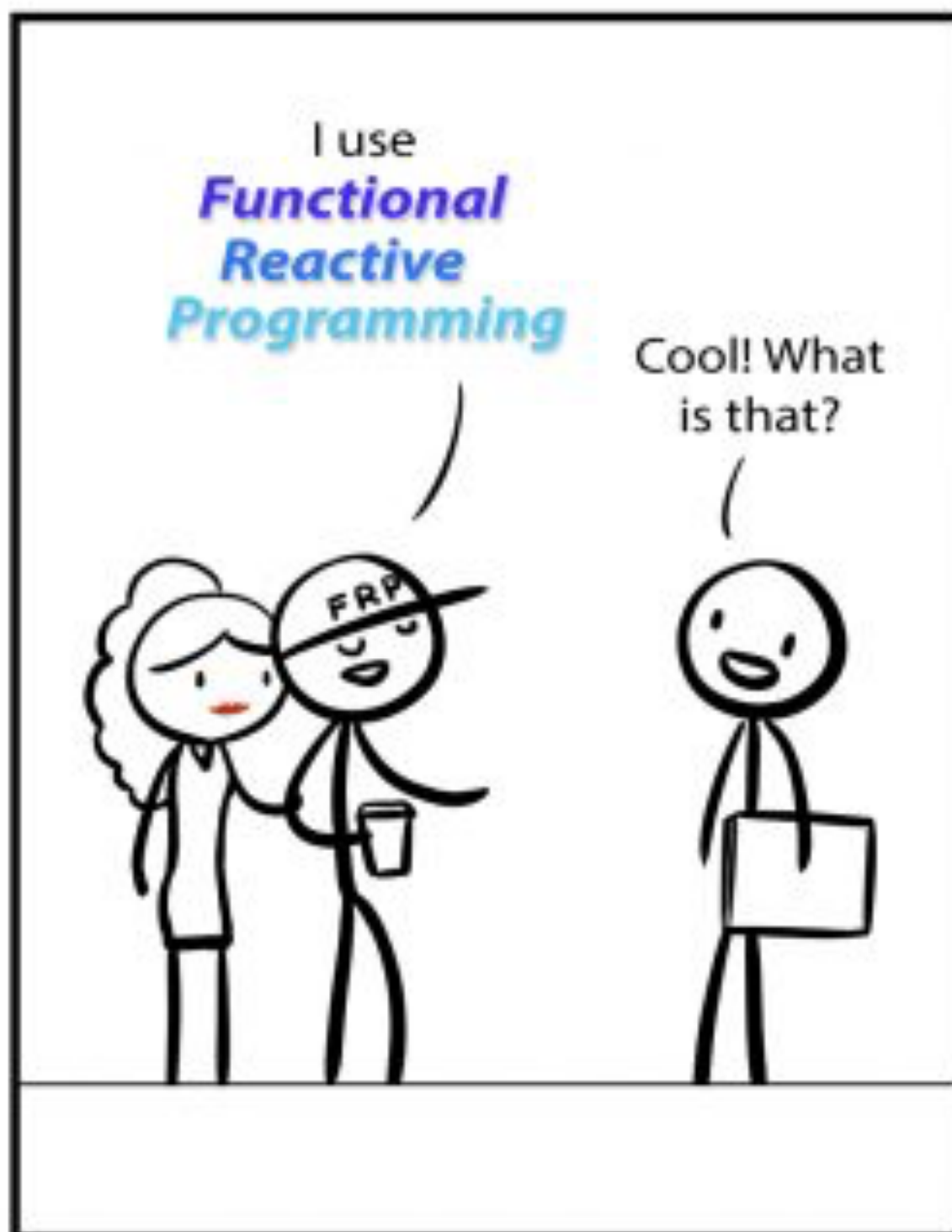


Webflux Functional Endpoints

Functional Reactive Programming

08/09/2021

**What is Functional Reactive
Programming (FRP) ?**



Functional Reactive Programming = Functional Programming + Reactive Programming

Functional Reactive Programming

- Programming Styles
 - Imperative
 - Declarative
- Programming Paradigms
 - Object-Oriented
 - Functional
 - Reactive

Imperative Style

- Java is primarily an imperative language
- Each step of the program has to be detailed
- Imperative Languages
 - C, C#, Java, JavaScript, etc.

```
if (chassisRepository.findByName(name).isEmpty()) {  
    throw new EntityNotFoundException("Chassis not found with name : "+name);  
}  
return chassisRepository.findByName(name);
```

Declarative Style

- Tells the program what to do, not how to do it.
- This style fits in perfectly with functional programming paradigm.
- Declarative Languages
 - Domain-Specific Languages
 - SQL, CSS, XML, Groovy, etc.

```
return chassisService.searchChassisByName(name);
```

Imperative vs Declarative

Imperative

```
// Imperative

let arr = [1, 2, 3, 4, 5], arr2 = [];

for (let i = 0; i < arr.length; i++) {
  arr2[i] = arr[i] * 2;
}

return arr2;
```

VS

Declarative

```
// Declarative

let arr = [1,2,3,4,5];

return arr.map(v => v * 2)
```


Imperative vs Declarative

Imperative

Explicit Instructions

The system is stupid,
you are smart

Declarative

Describe the Outcome

The system is smart,
you don't care

Object-Oriented Paradigm

- Everything is an object
- Object contains data (fields/attributes/properties) and code (methods)
- Class-based
- Usually imperative and procedural programming

Functional Paradigm

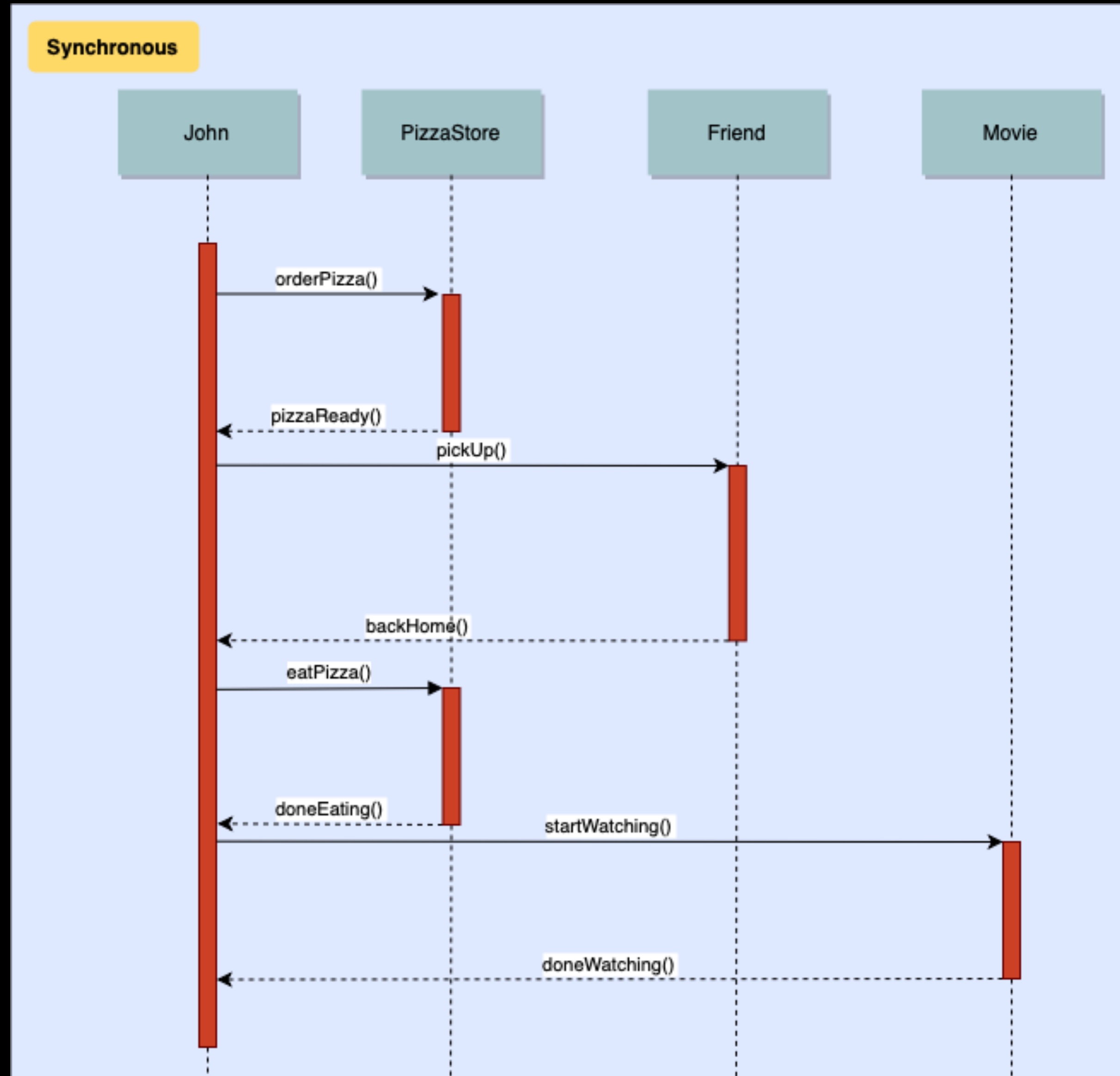
- Having functions does not make your code functional
- Use Java 8 functional API also does not make your code functional
- Functional should tell what to do, not how to do it (declarative)
- To be functional a set of rules must to be obeyed
 - Idempotent, pure functions, immutability, closure, high-order functions, etc.
- Functions should avoid side-effects

Reactive Paradigm

- Asynchronous data streams
- Non-Blocking
- Event-Driven
- Push and pull model
- Changed, created, combined on the fly
- Unordered execution
- Back-pressure out of the box support

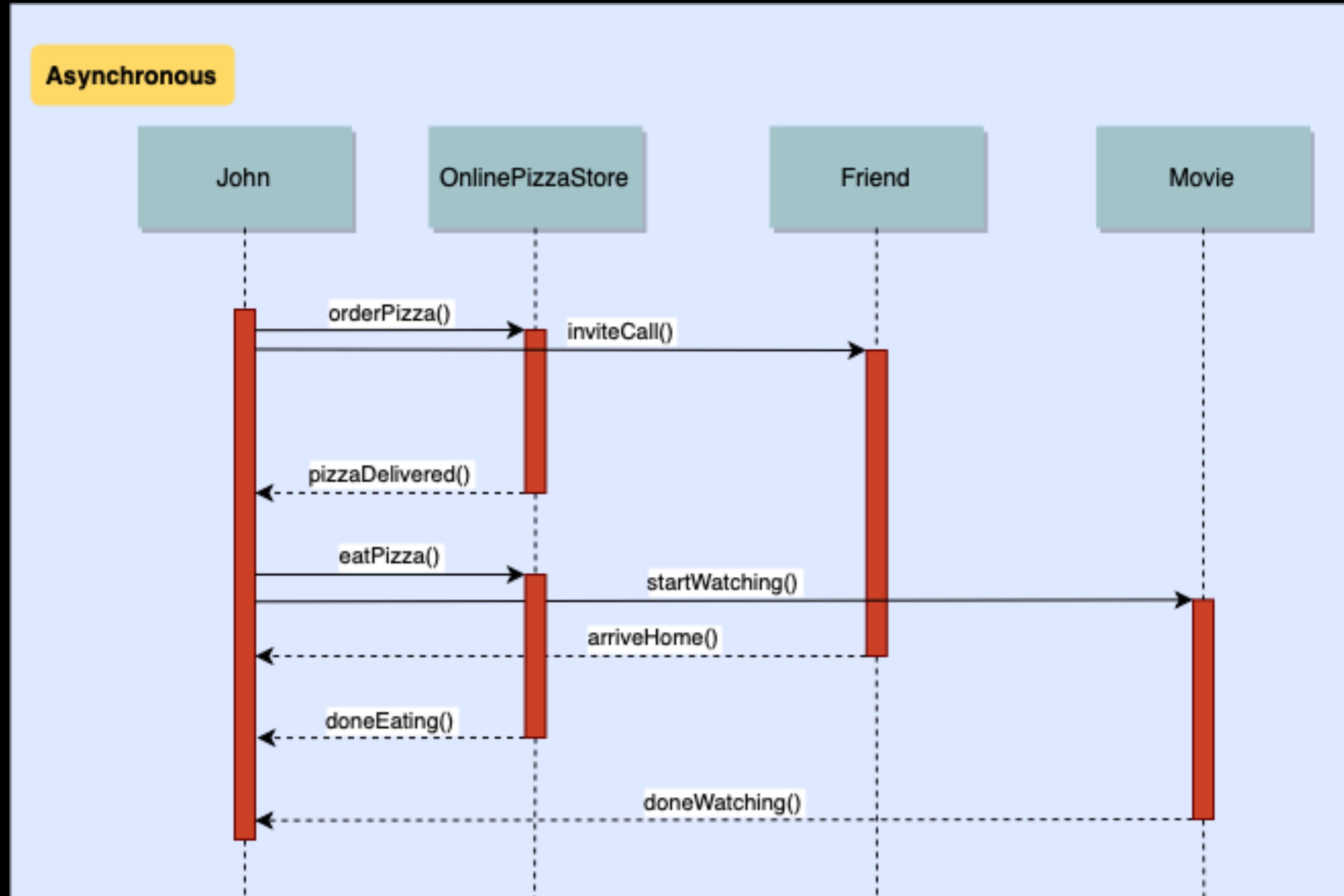
Synchronous Calls

The Bad



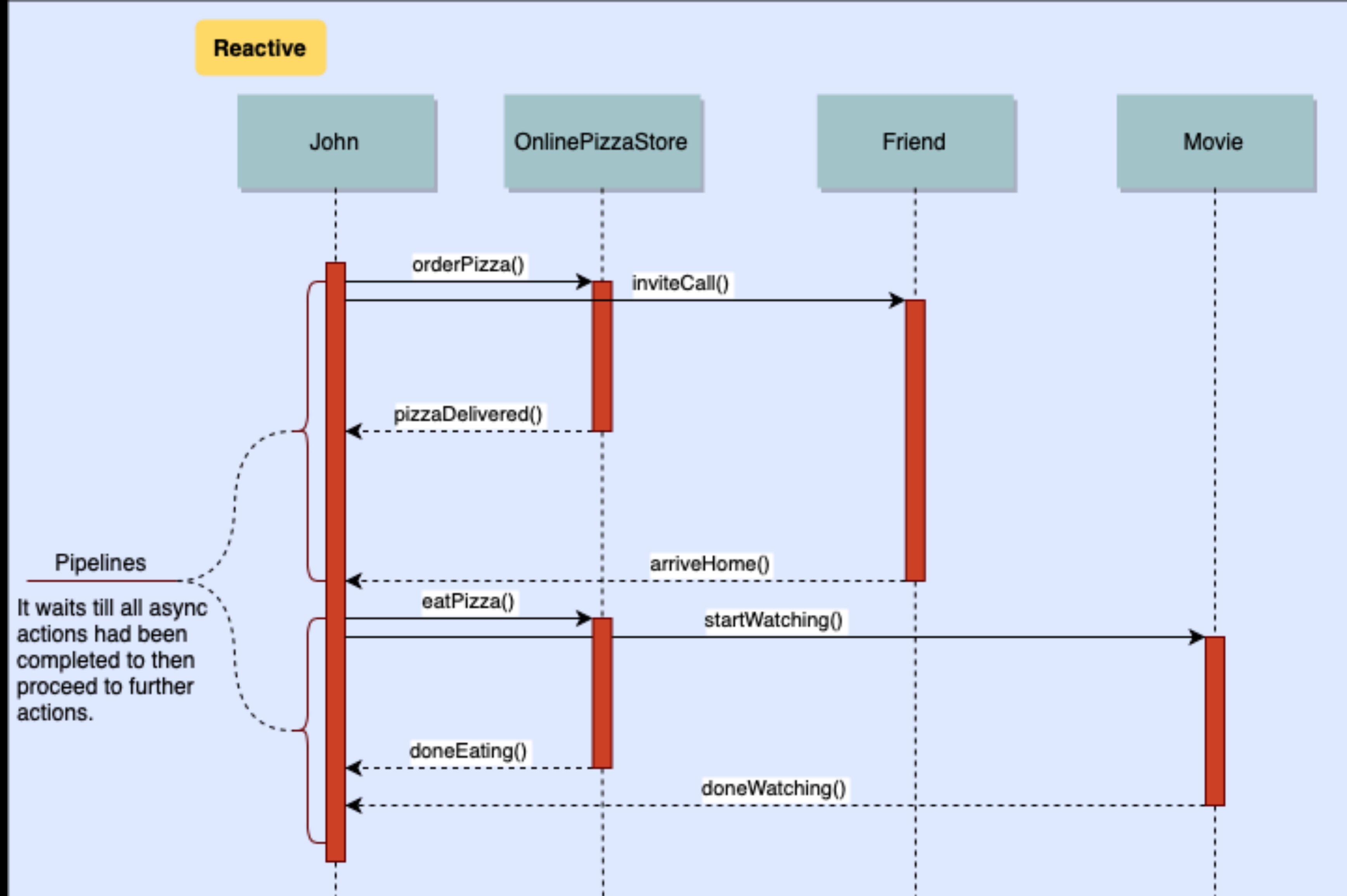
Asynchronous Calls

The Ugly

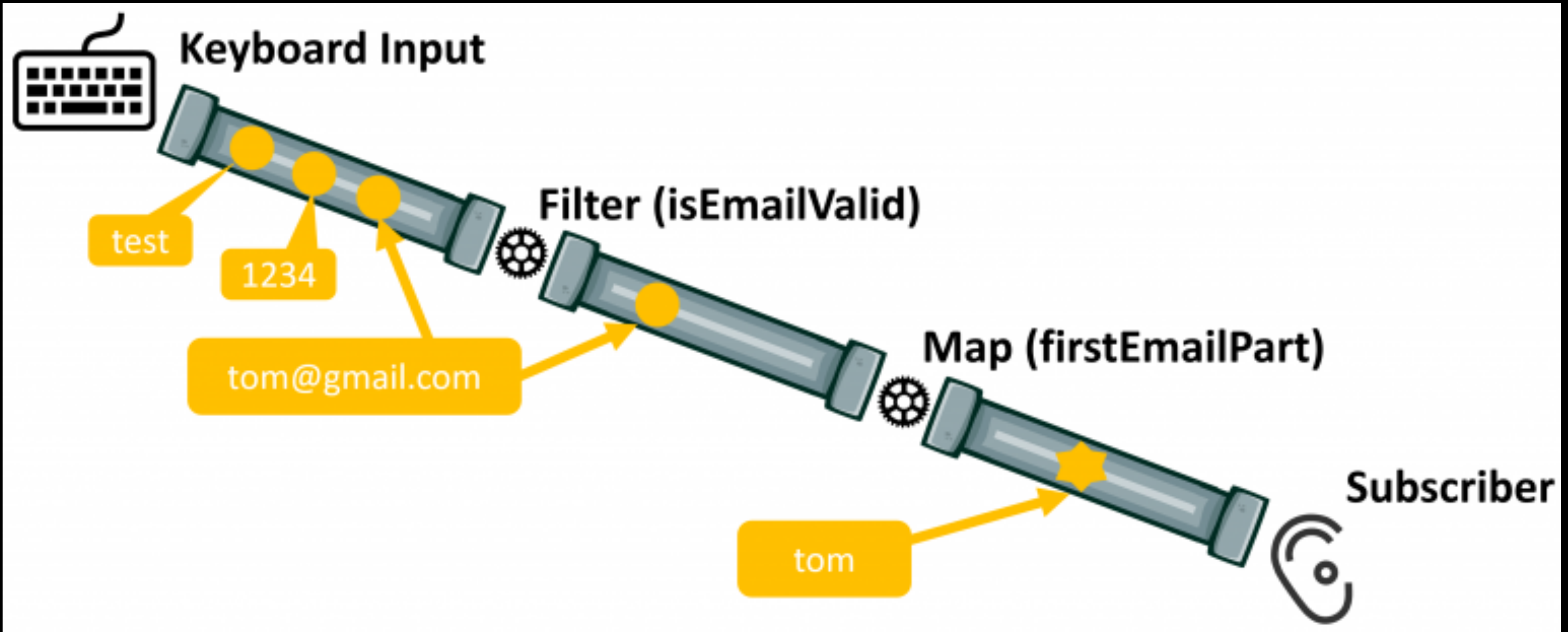


Reactive Calls

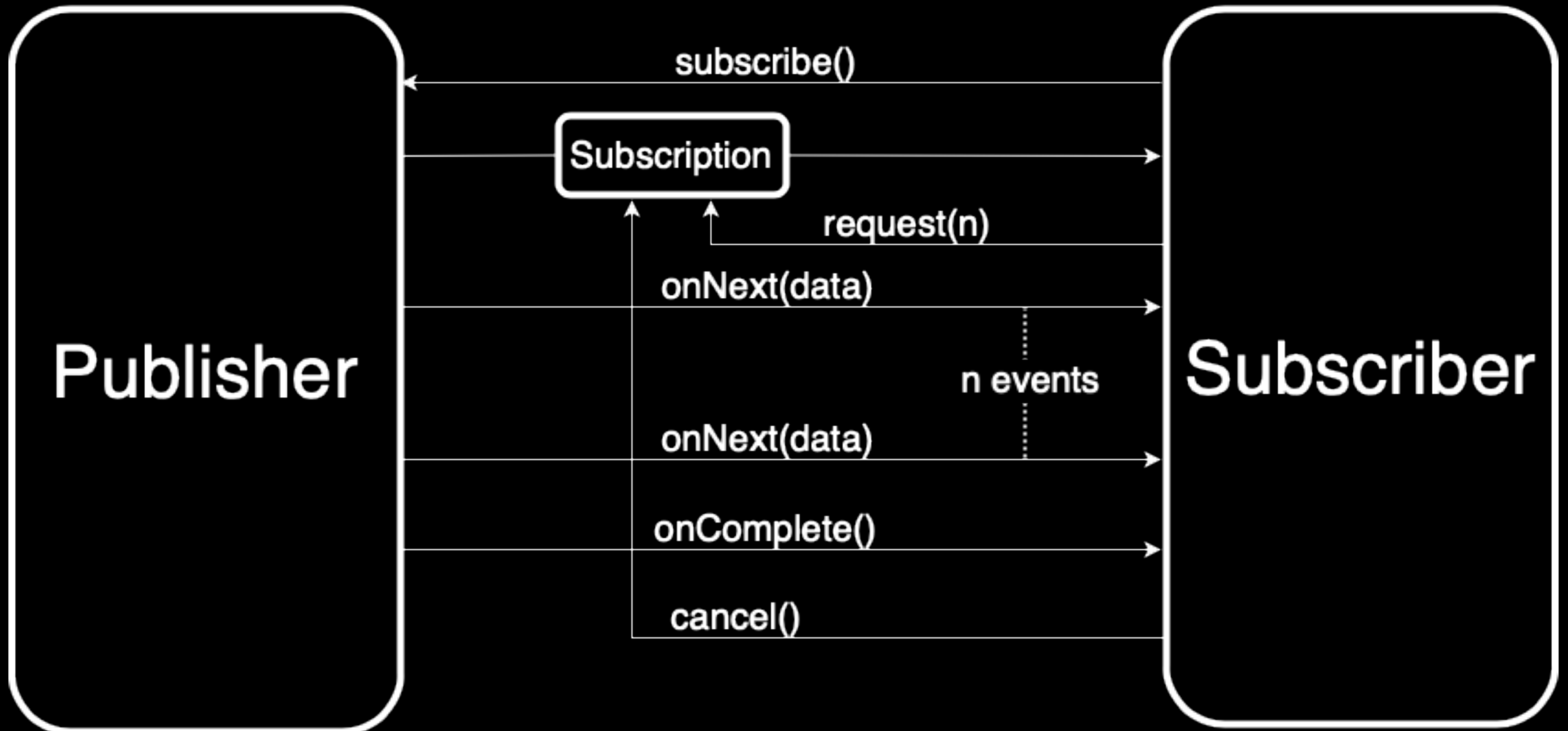
The Good



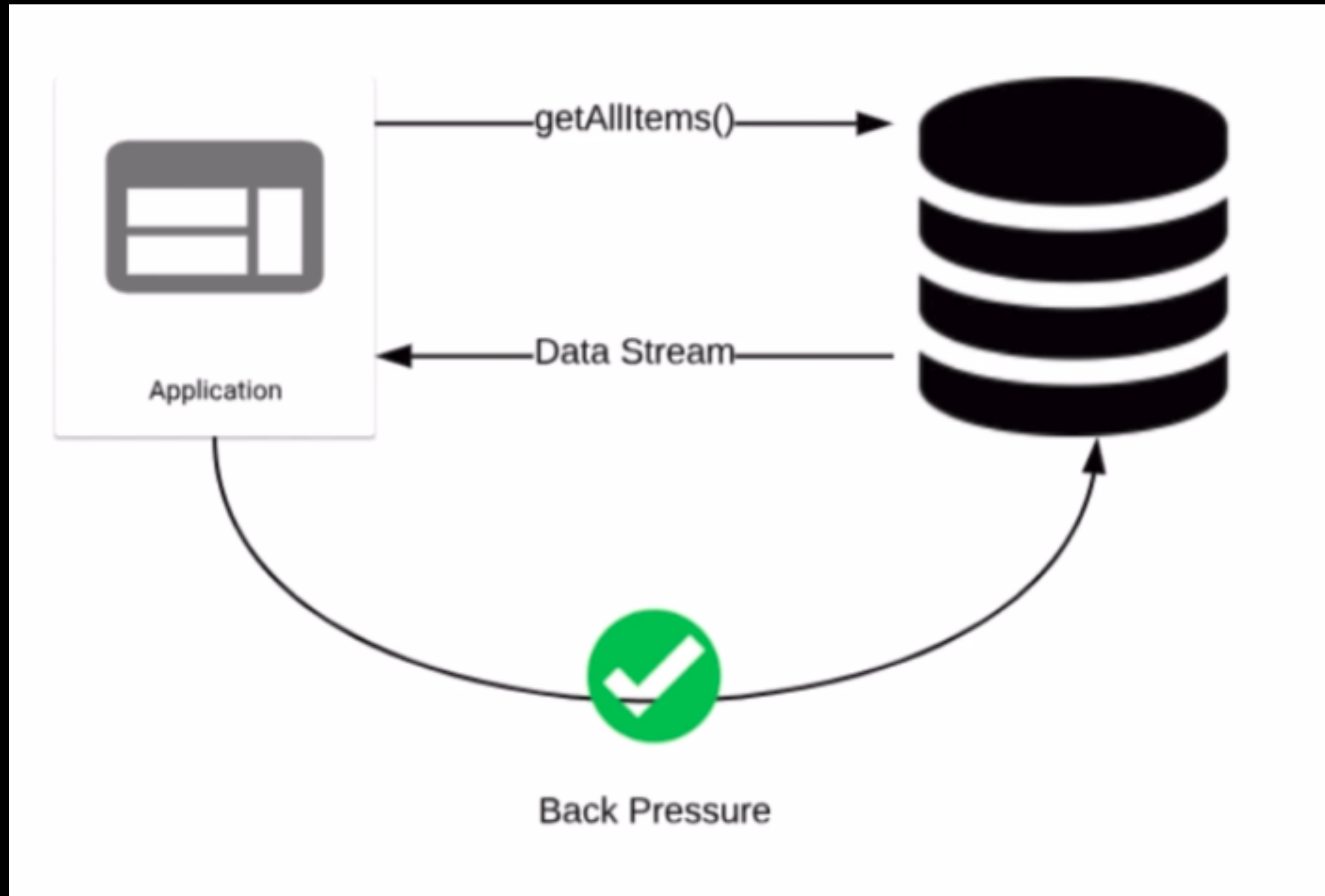
Reactive Paradigm



Reactive Flow



Back-Pressure



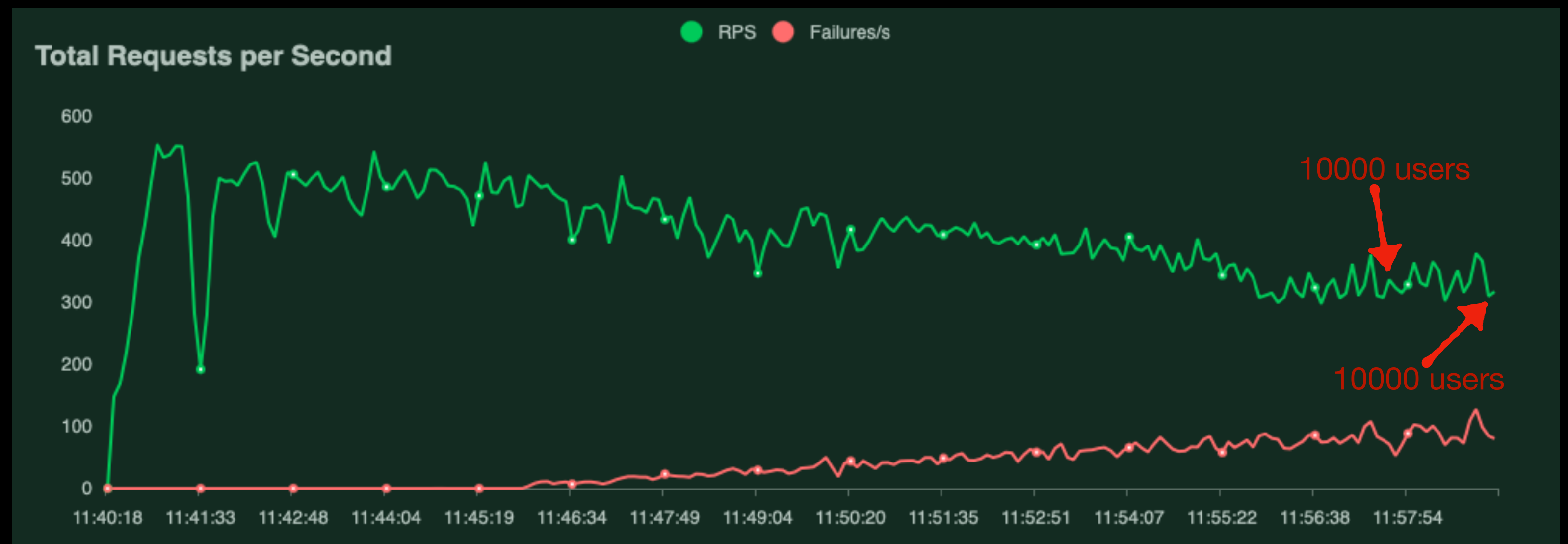
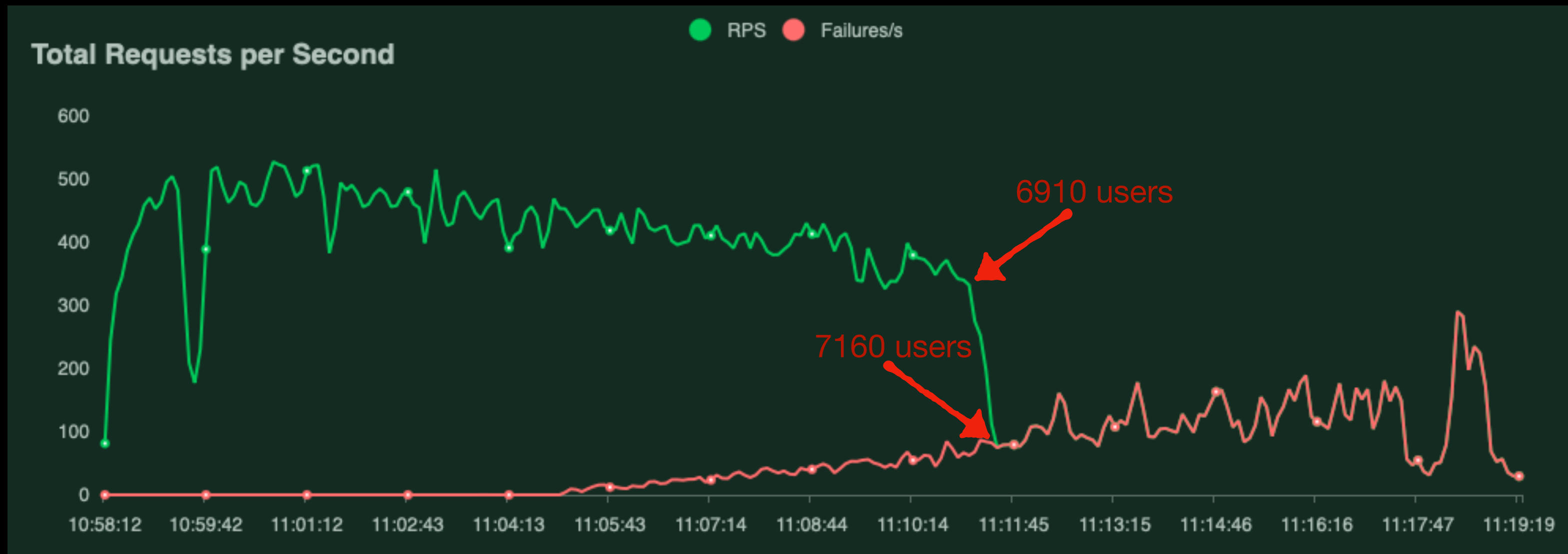
Push/Pull Model

- The publisher starts pushing data, as soon as, the subscription is made
- The subscriber controls how much data it would like to pull
- The subscriber decides when cancel the subscription

Reactive Stream Specification

- Java 9 Reactive Stream SPI Support in the JDK
- Implementations
 - ReactiveX (RxJava)
 - Akka Streams
 - Project Reactor (Spring Webflux)

Why Reactive?



References

<https://www.reactive-streams.org>

<https://github.com/reactive-streams/reactive-streams-jvm>

<https://projectreactor.io>

<https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html>

<http://reactivex.io>

<https://github.com/ReactiveX/RxJava>

<https://doc.akka.io/docs/akka/current/stream/index.html>

“Talk is cheap. Show me the code.”

— Linus Torvalds