Reactive Streams in the Web

Florian Stefan | eBay Classifieds Group | JUG Saxony 2017

Who am I?

- Florian Stefan
- mobile.de (eBay Classifieds Group)
- https://ebaytech.berlin/
- fstefan@ebay.com | @f_s_t_e_f_a_n
- https://github.com/florian-stefan/

Contents

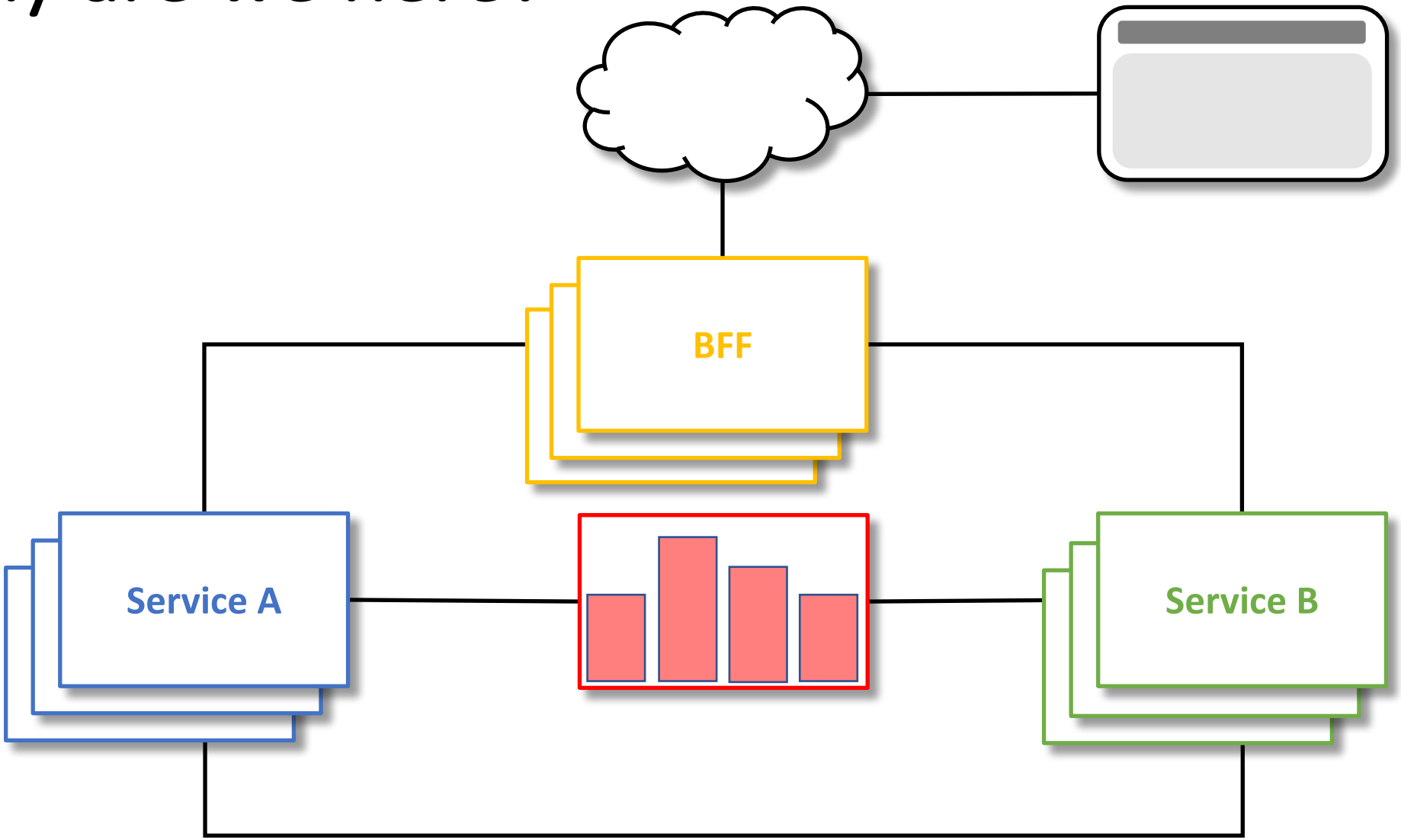
1) Motivation: Distributed Systems
2) Reactive Streams with Reactor
3) Progressive HTML Rendering with Spring 5

It‘s about concepts rather than details!

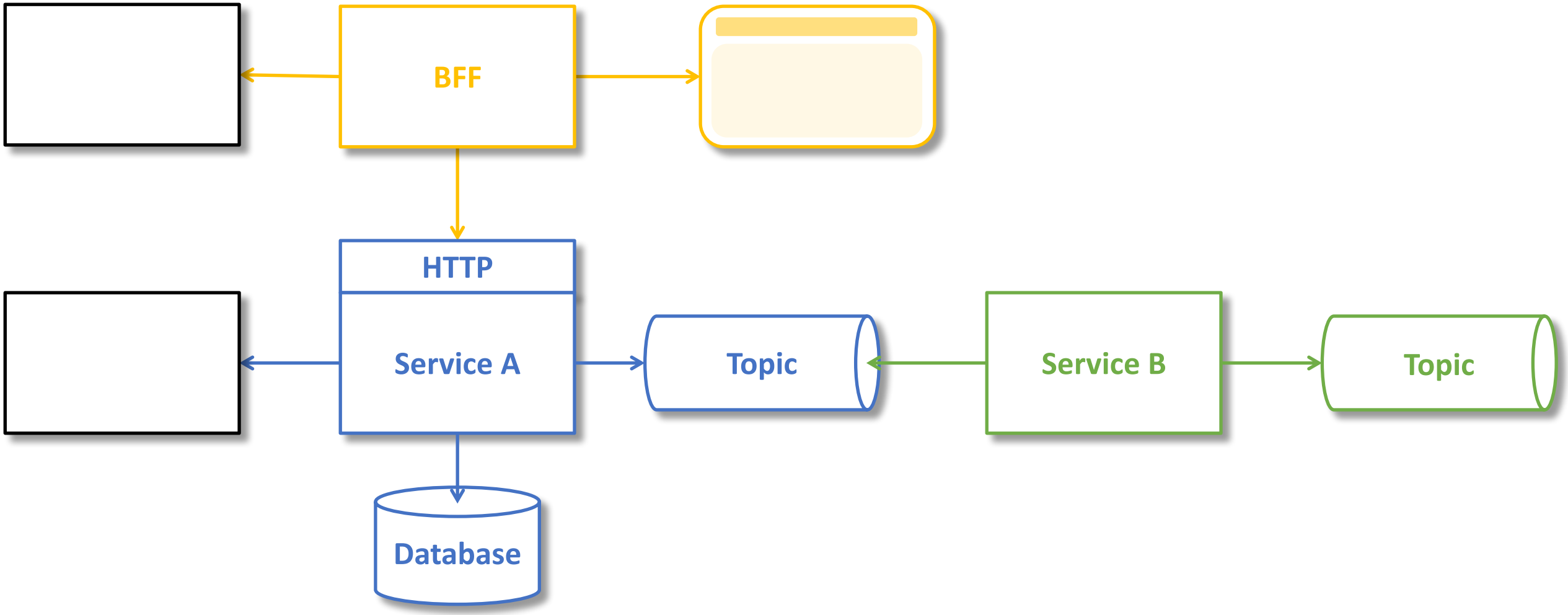
Why are we here?

It’s a distributed world!

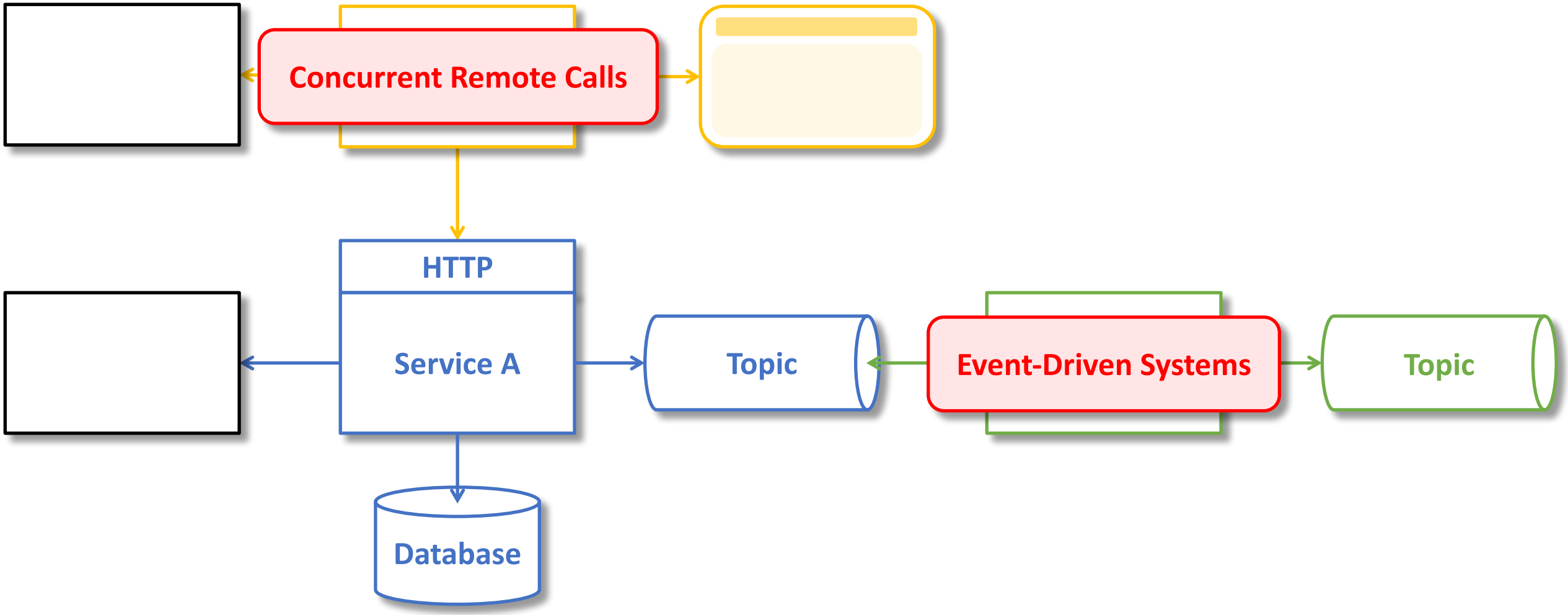
Why are we here?



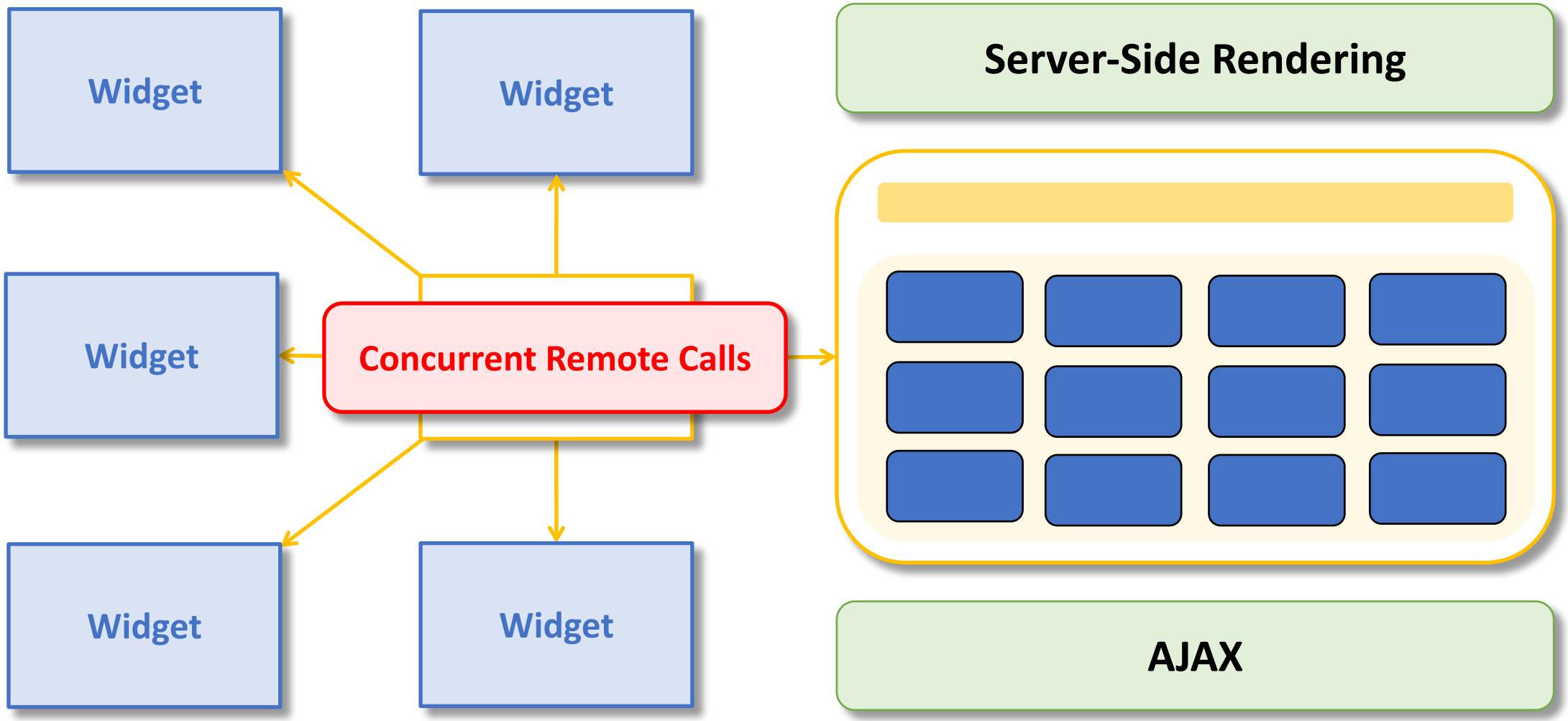
How are we doing this?



How are we doing this?



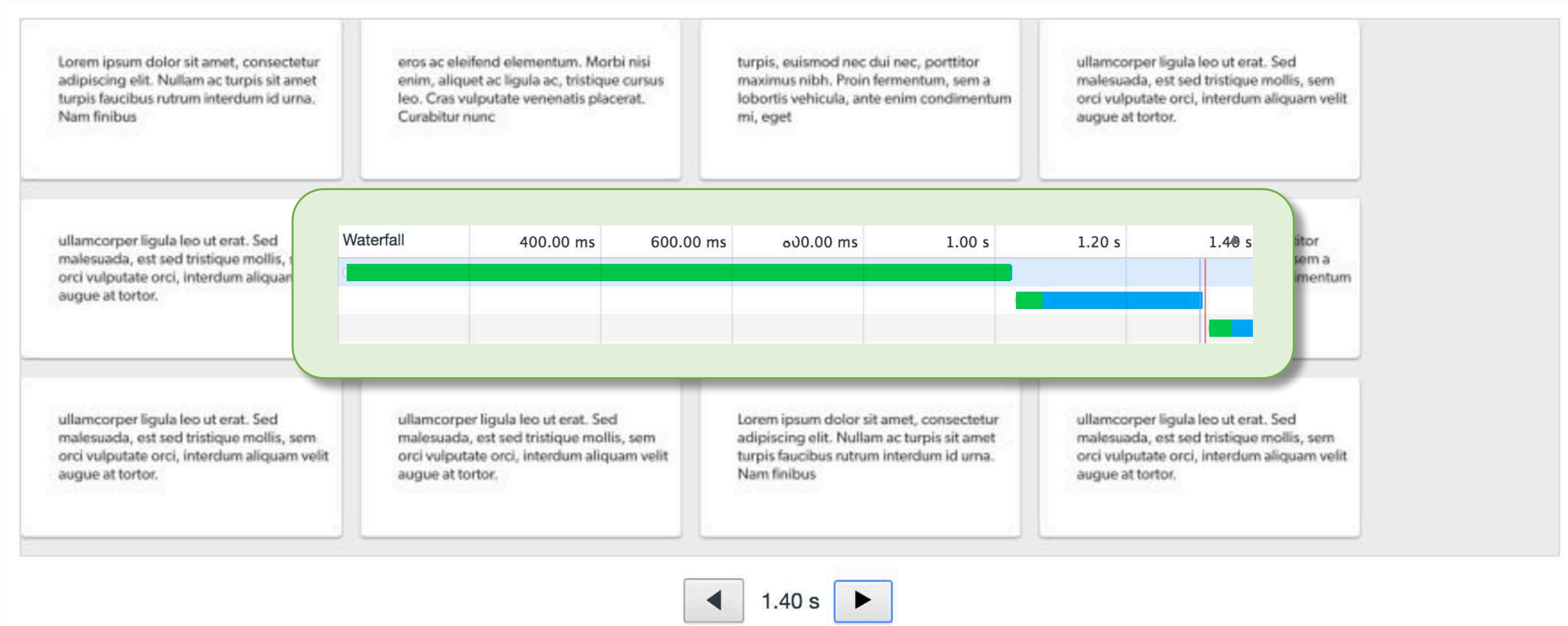
HTML Rendering



HTML Rendering

Server-Side Rendering

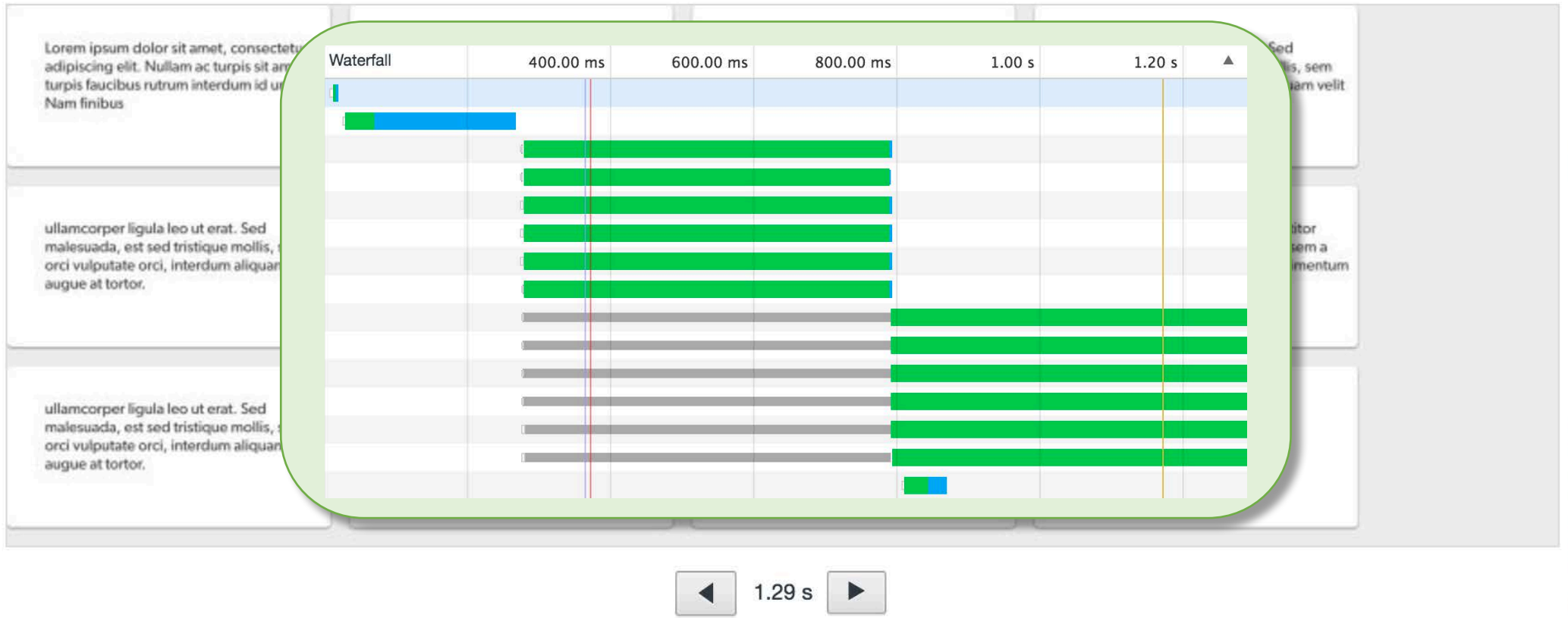
- Time To First Byte
- Prerendering / Caching



HTML Rendering

AJAX

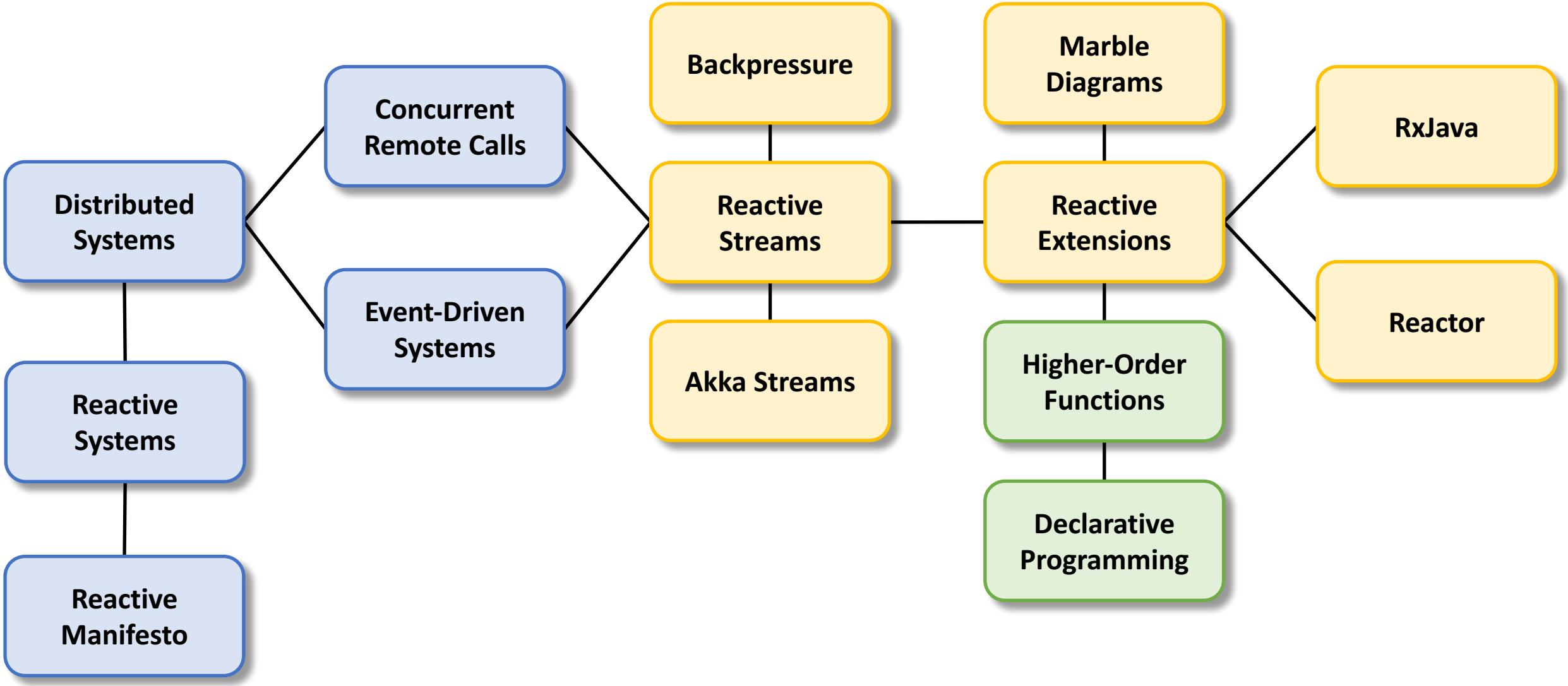
- Head-of-line blocking
- Multiplexing with HTTP/2

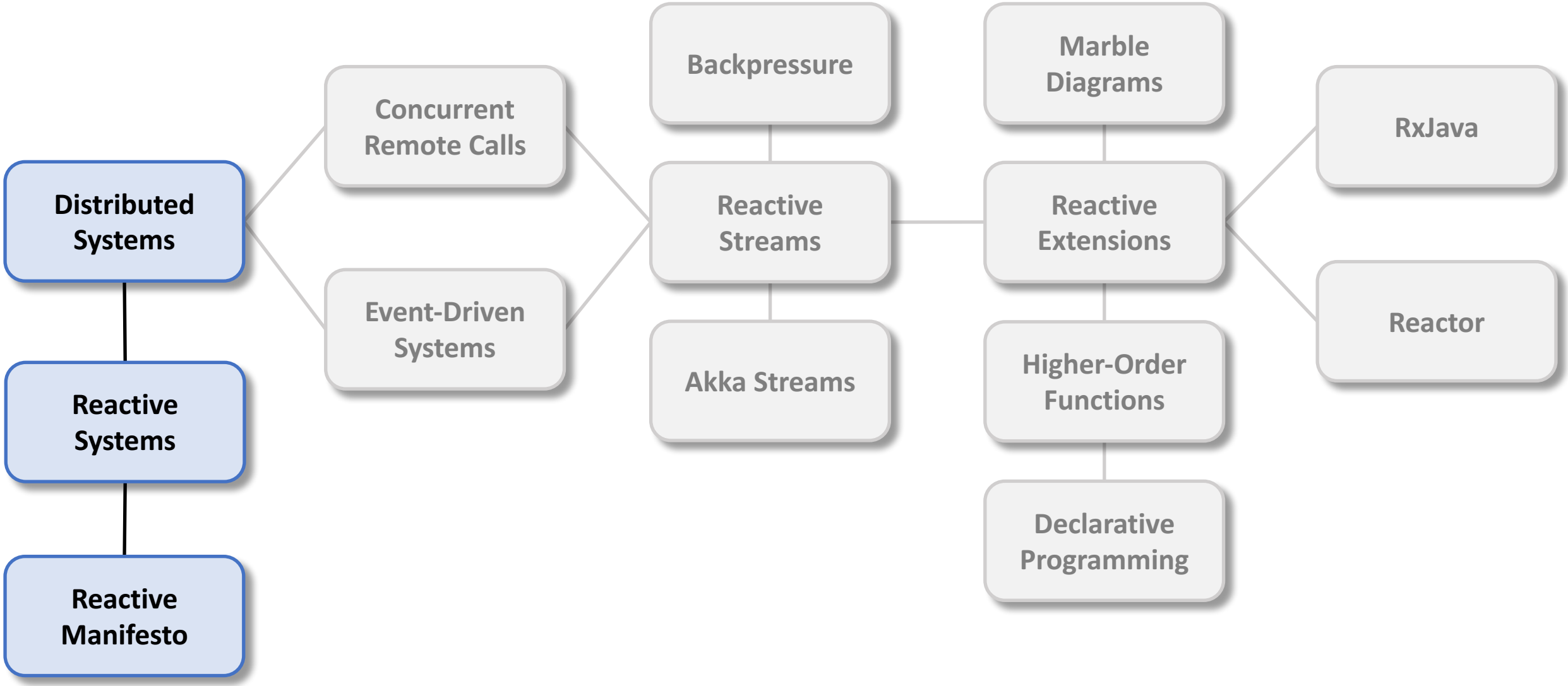


Can we do better?

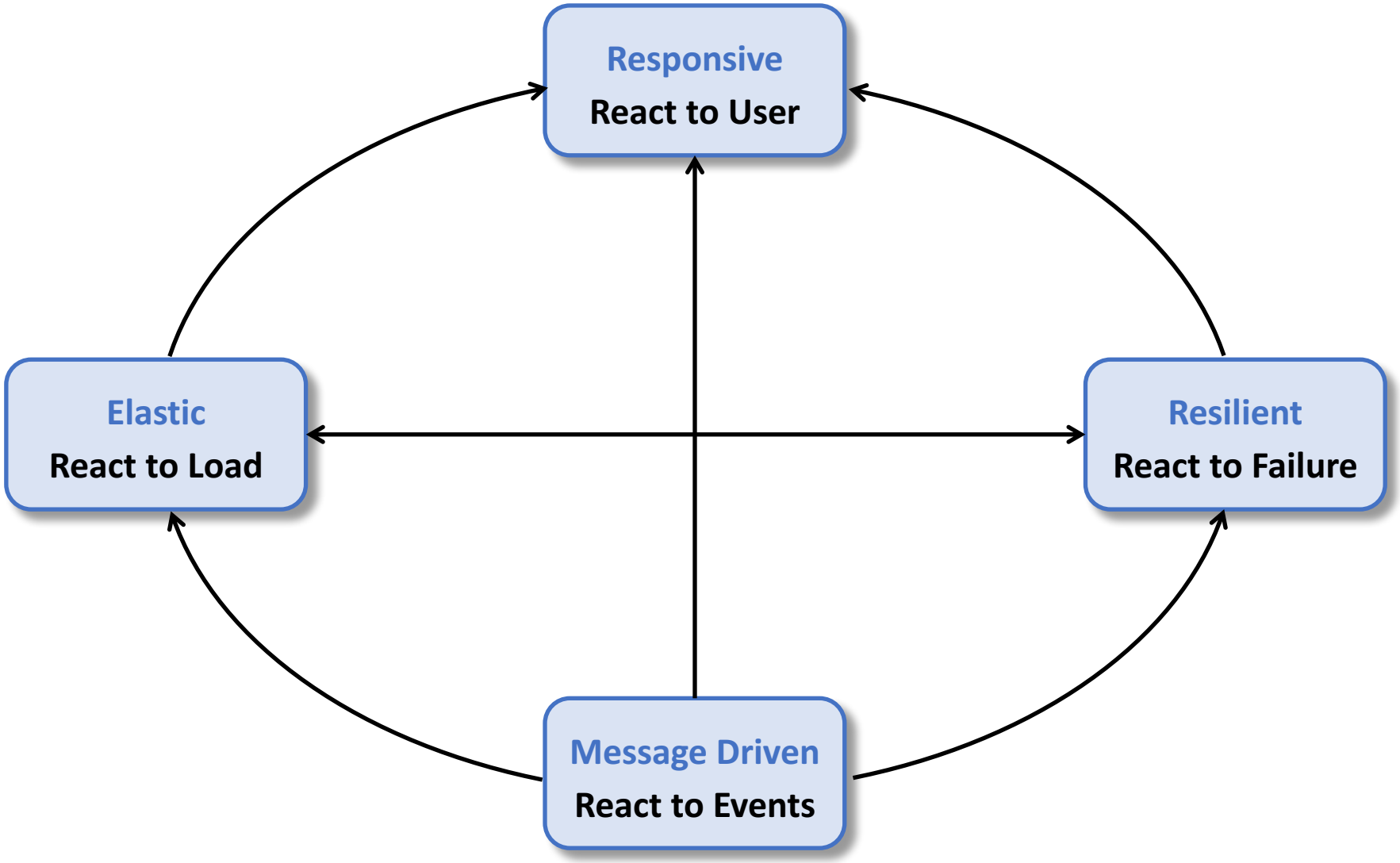
How expensive will it be?

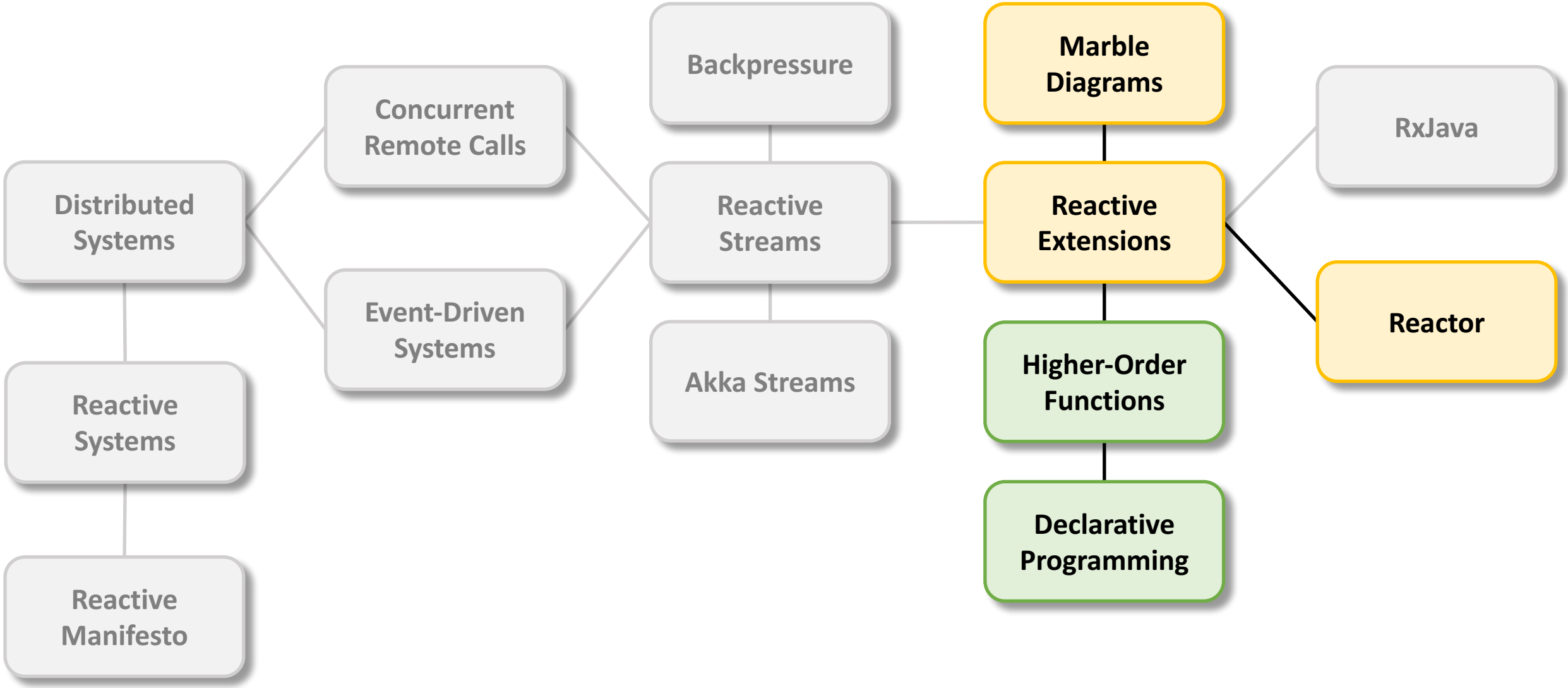
The Reactive Landscape


Distributed Systems
Concurrent Remote Calls
Event-Driven Systems
Reactive Systems
Reactive Manifesto
Backpressure
Reactive Streams
Akka Streams
Marble Diagrams
Reactive Extensions
Higher-Order Functions
Declarative Programming
RxJava
Reactor

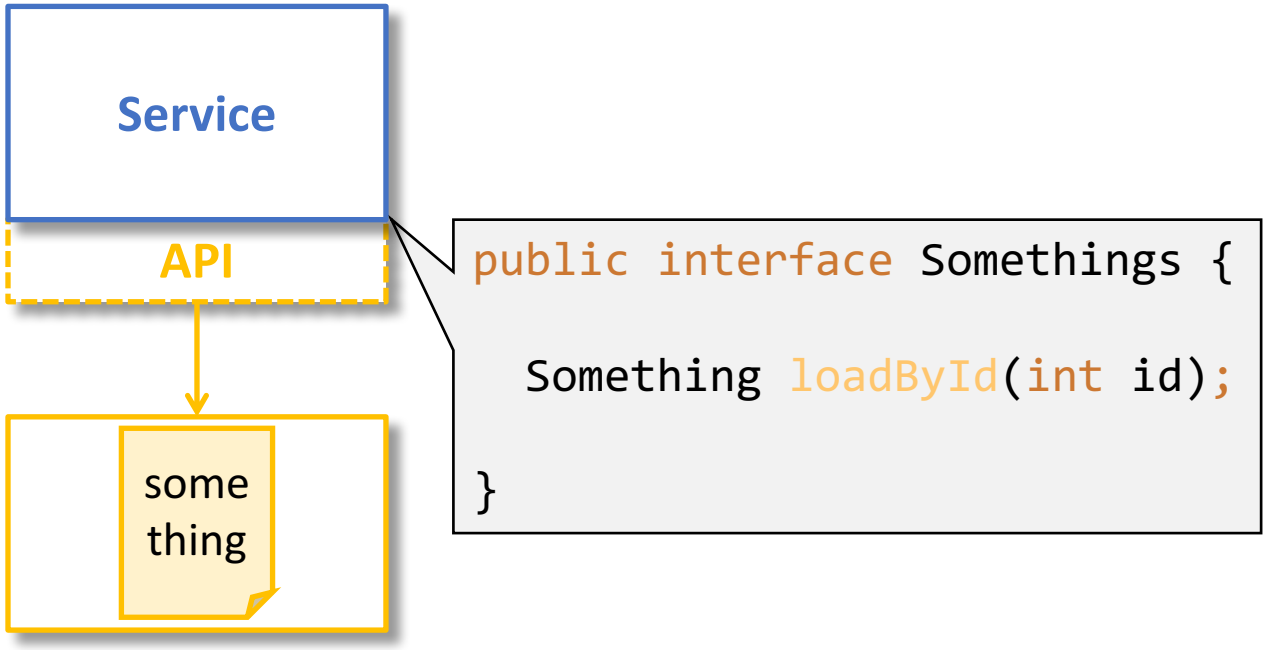

Distributed Systems
Reactive Systems
Reactive Manifesto
Concurrent Remote Calls
Event-Driven Systems
Backpressure
Reactive Streams
Akka Streams
Marble Diagrams
Reactive Extensions
Higher-Order Functions
Declarative Programming
RxJava
Reactor

The Reactive Manifesto

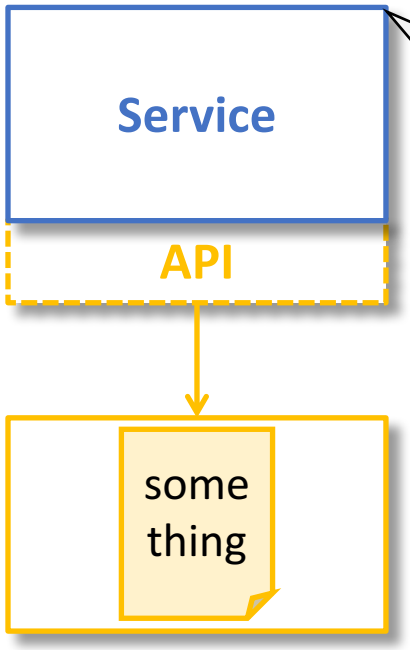

Responsive
React to User
Elastic
React to Load
Resilient
React to Failure
Message Driven
React to Events


Distributed Systems
Concurrent Remote Calls
Event-Driven Systems
Backpressure
Reactive Streams
Akka Streams
Marble Diagrams
Reactive Extensions
Higher-Order Functions
Declarative Programming
RxJava
Reactor
Reactive Systems
Reactive Manifesto

Reactive Extensions

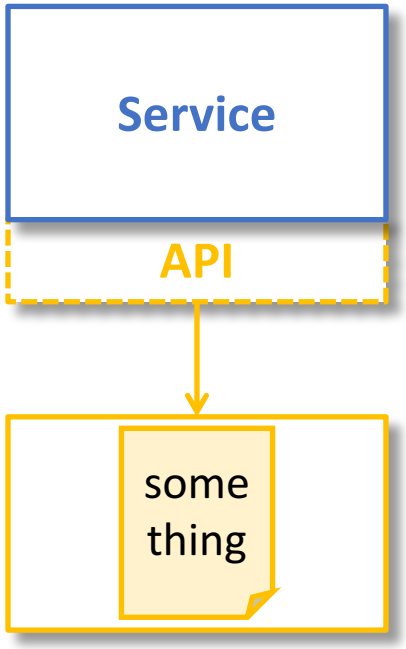


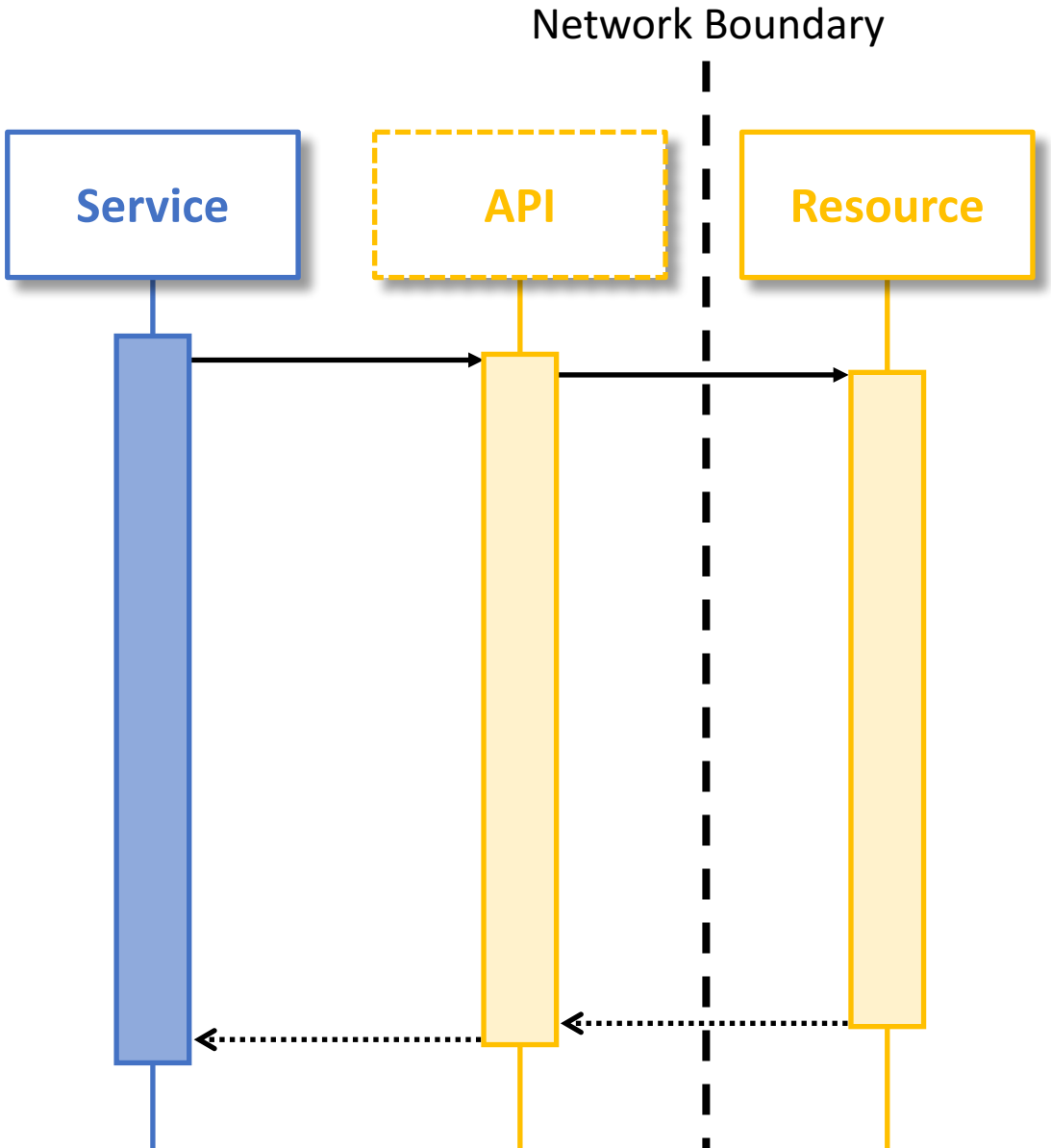
Reactive Extensions



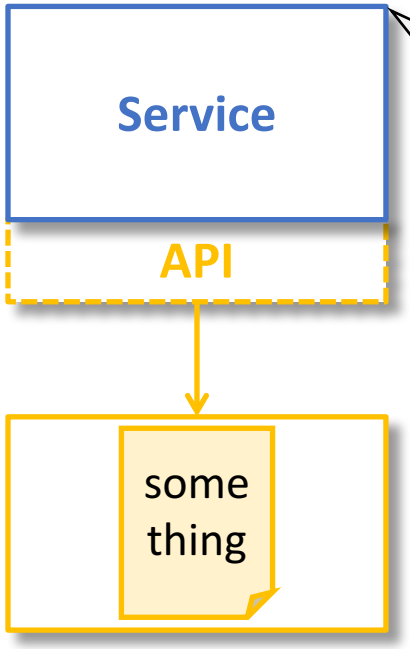

```
public Something loadById(int id) {
  return somethings.loadById(id);
}
```

Reactive Extensions



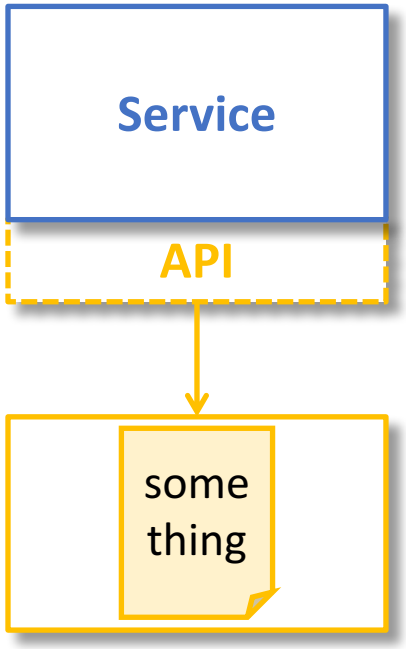


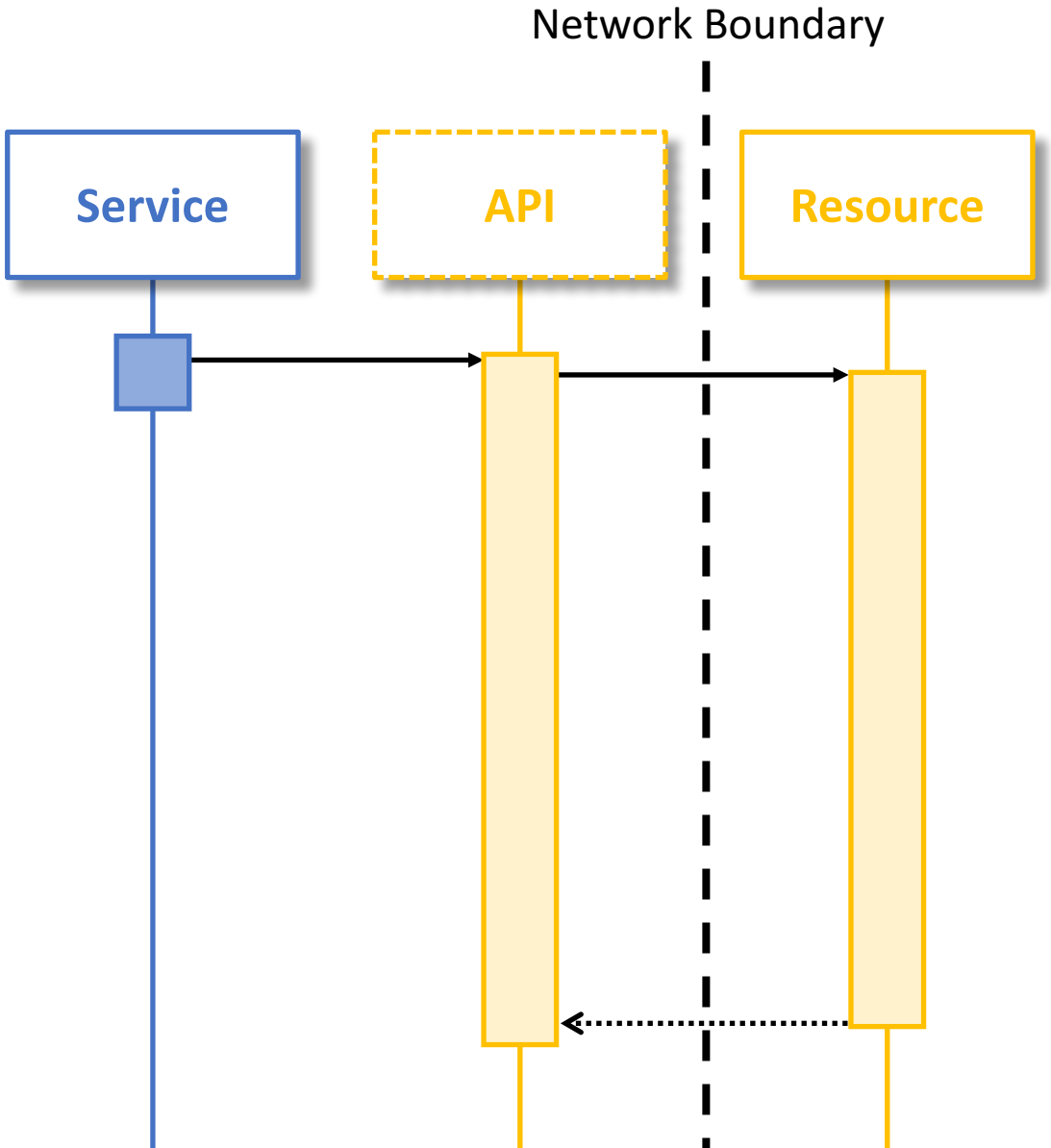
Reactive Extensions



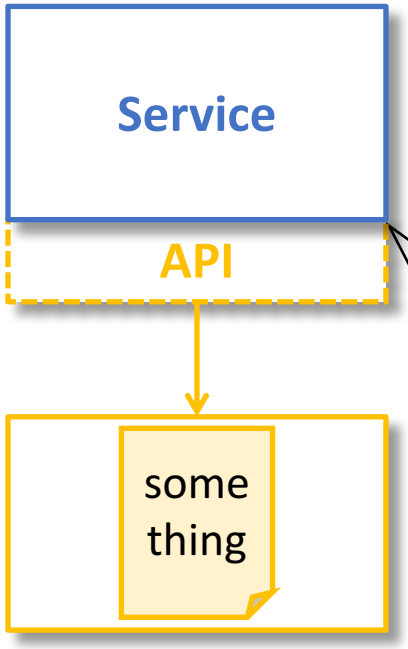

```
public CompletableFuture<Something> loadById(int id) {
  return supplyAsync(() -> somethings.loadById(id), executor);
}
```

Reactive Extensions



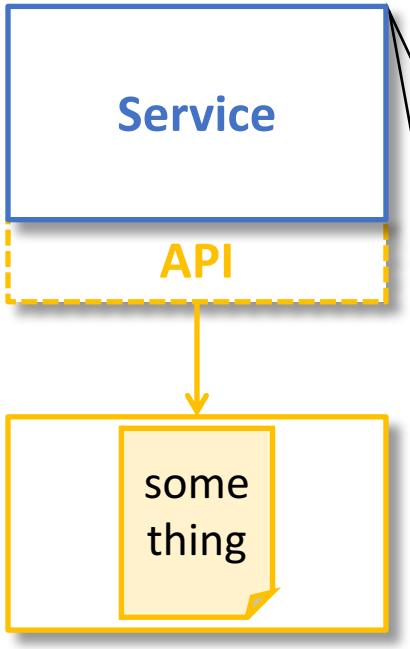


Reactive Extensions



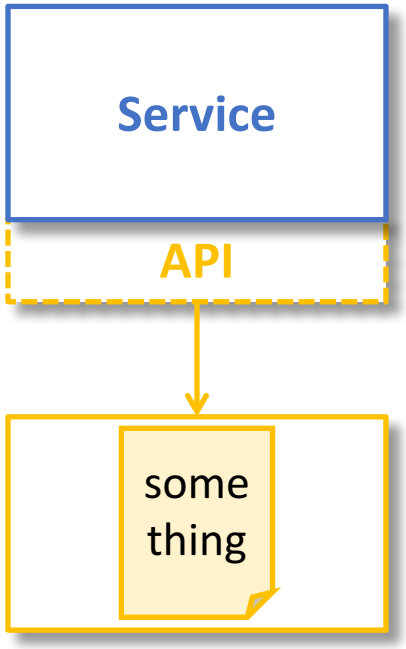

```
public interface Somethings {

  void loadById(int id, BiConsumer<Exception, Something> callback);

}
```

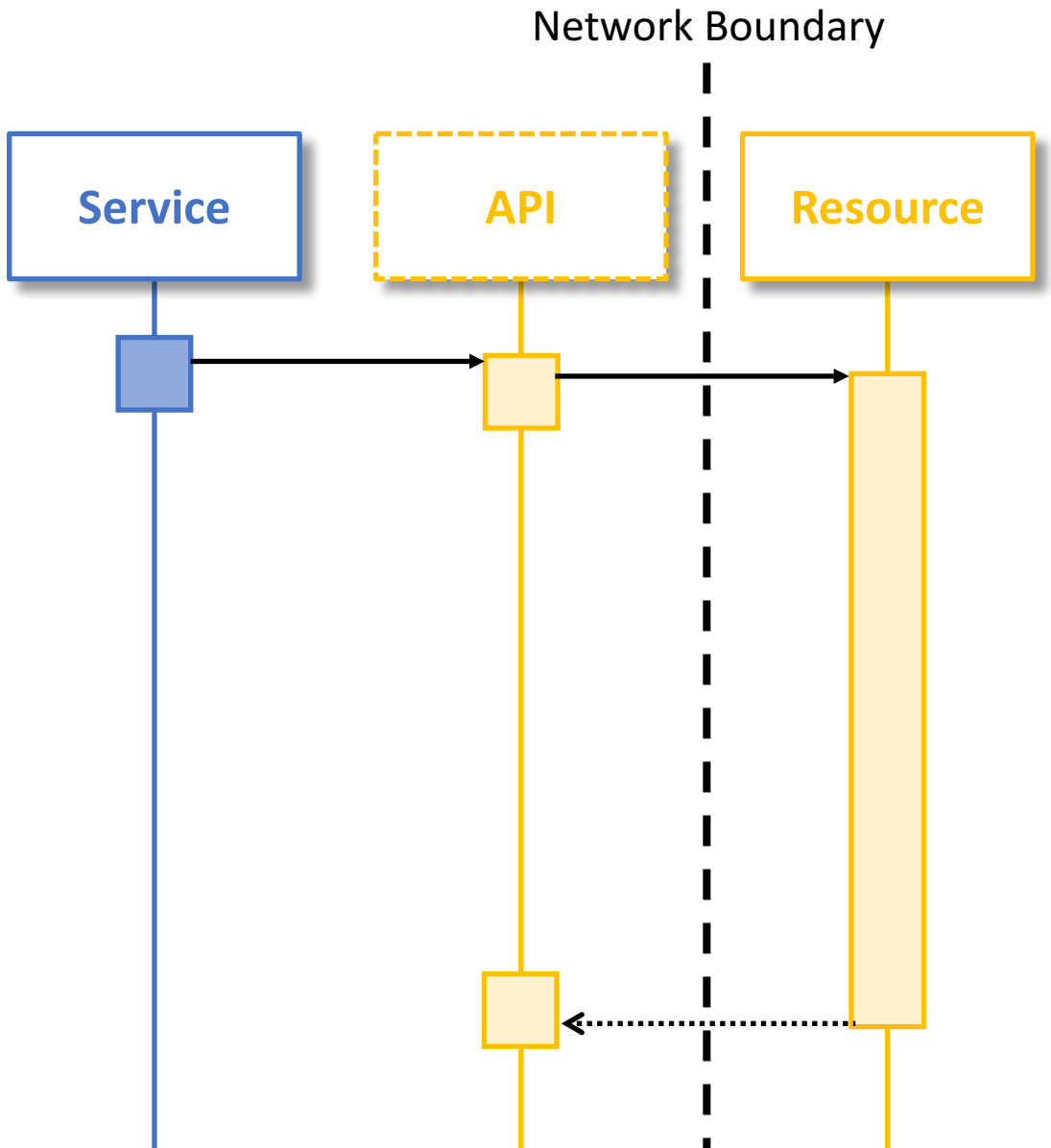
Reactive Extensions




```
public CompletableFuture<Something> loadById(int id) {
  CompletableFuture<Something> eventualSomething =
    new CompletableFuture<>();

  somethings.loadById(id, (error, something) -> {
    if (error != null) {
      eventualSomething.completeExceptionally(error);
    } else {
      eventualSomething.complete(something);
    }
  });

  return eventualSomething;
}
```

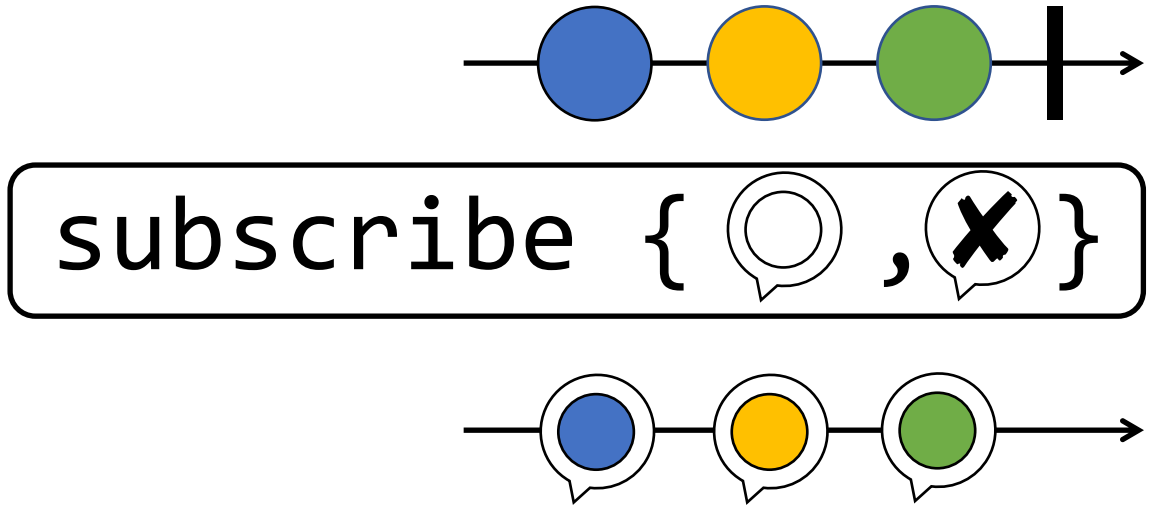
Reactive Extensions



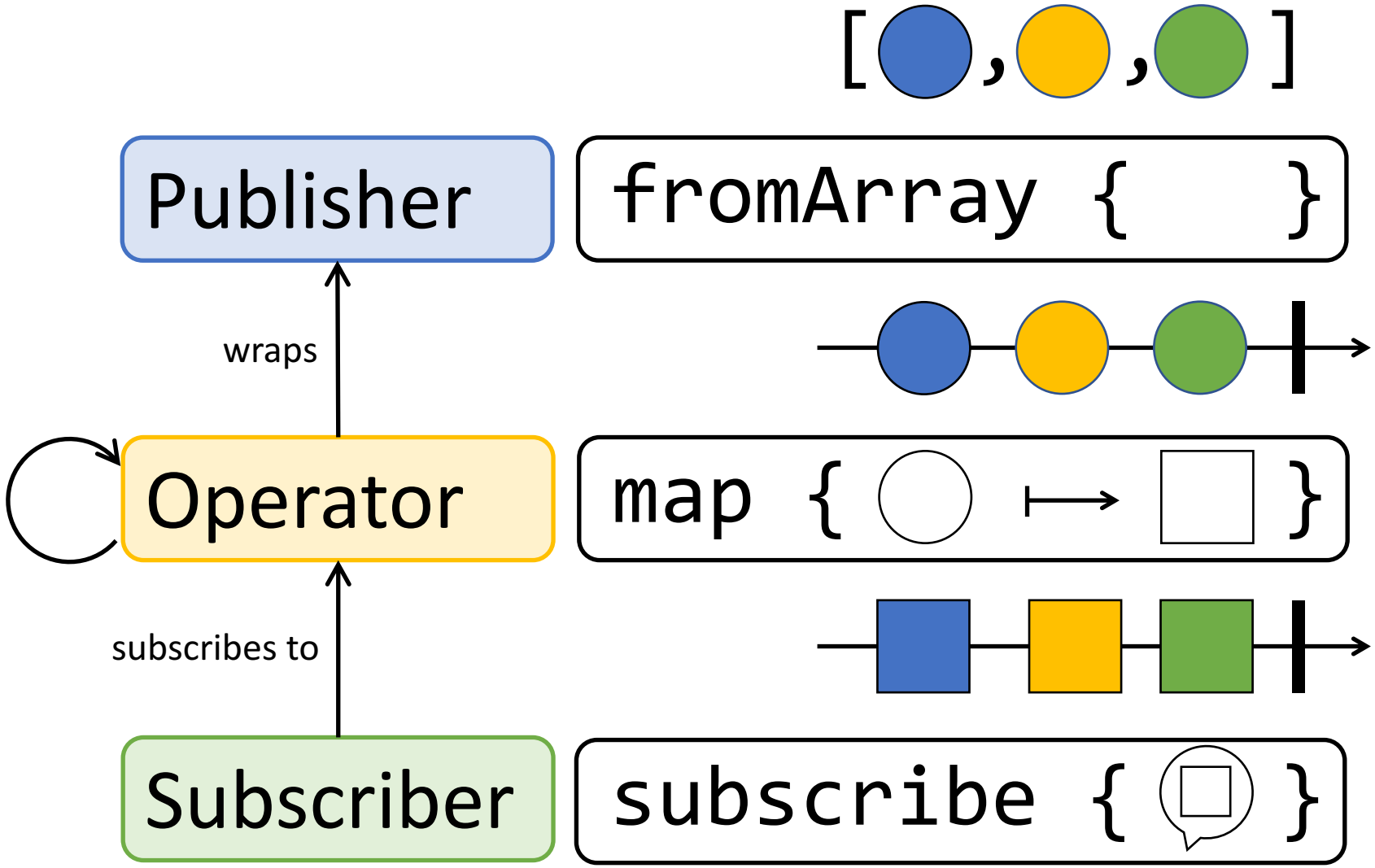


Reactive Extensions

CompletableFuture<T>	Mono<T>	Flux<T>
represents an already scheduled task	represents a building plan (caller has to subscribe)	represents a building plan (caller has to subscribe)
completes only once	emits at most one value	emits many values



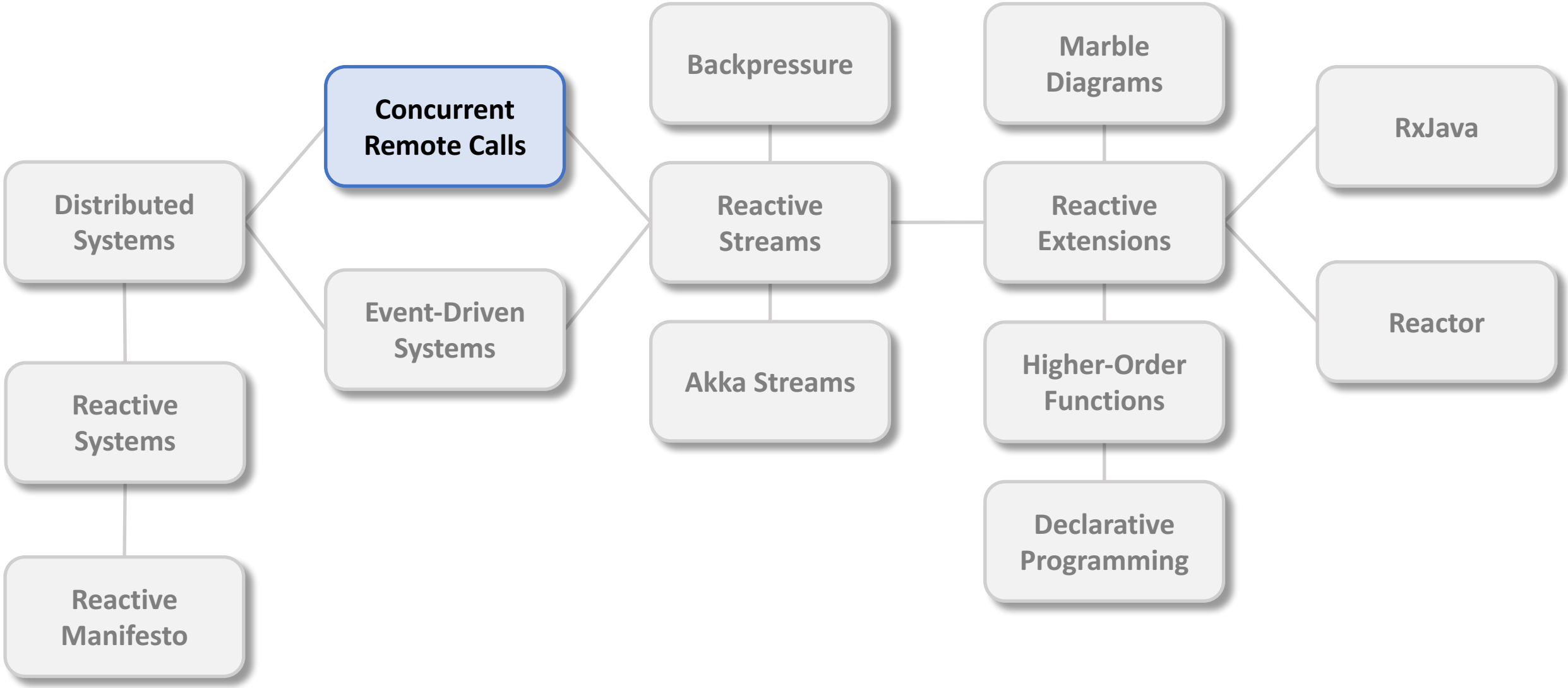
Reactive Extensions



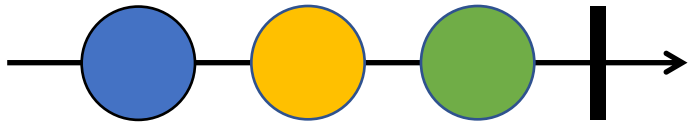
Reactive Extensions

```
Circle[] circles = { Circle.of(BLUE), ... };

Flux.fromArray(circles)

    .map(circle -> Square.of(circle.getColor()))

    .log()

    .subscribe(...);
```

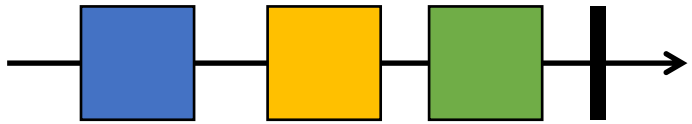
```
[main] INFO - | onSubscribe()
[main] INFO - | ?
[main] INFO - | onNext(Square(BLUE))
[main] INFO - | onNext(Square(YELLOW))
[main] INFO - | onNext(Square(GREEN))
[main] INFO - | onComplete()
```


Distributed Systems
Concurrent Remote Calls
Event-Driven Systems
Reactive Systems
Reactive Manifesto
Backpressure
Reactive Streams
Akka Streams
Marble Diagrams
Reactive Extensions
Higher-Order Functions
Declarative Programming
RxJava
Reactor

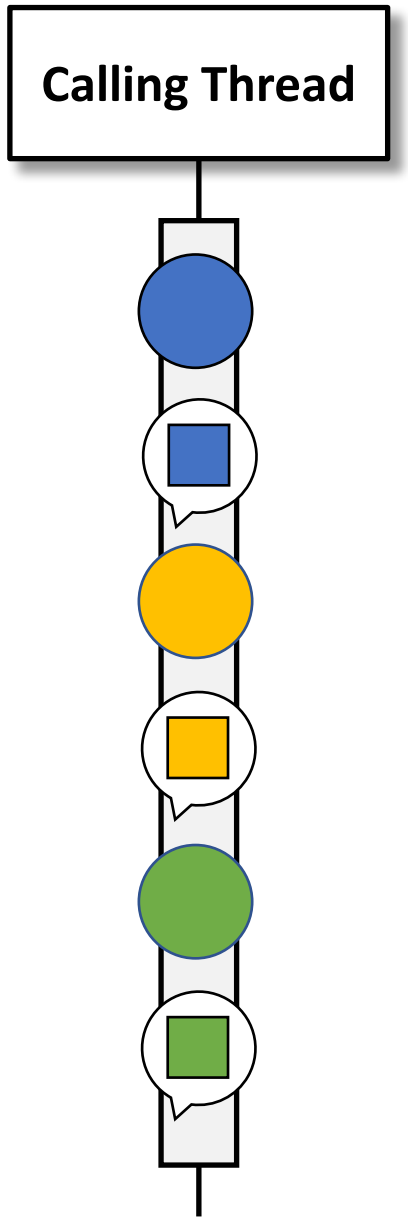
Concurrency



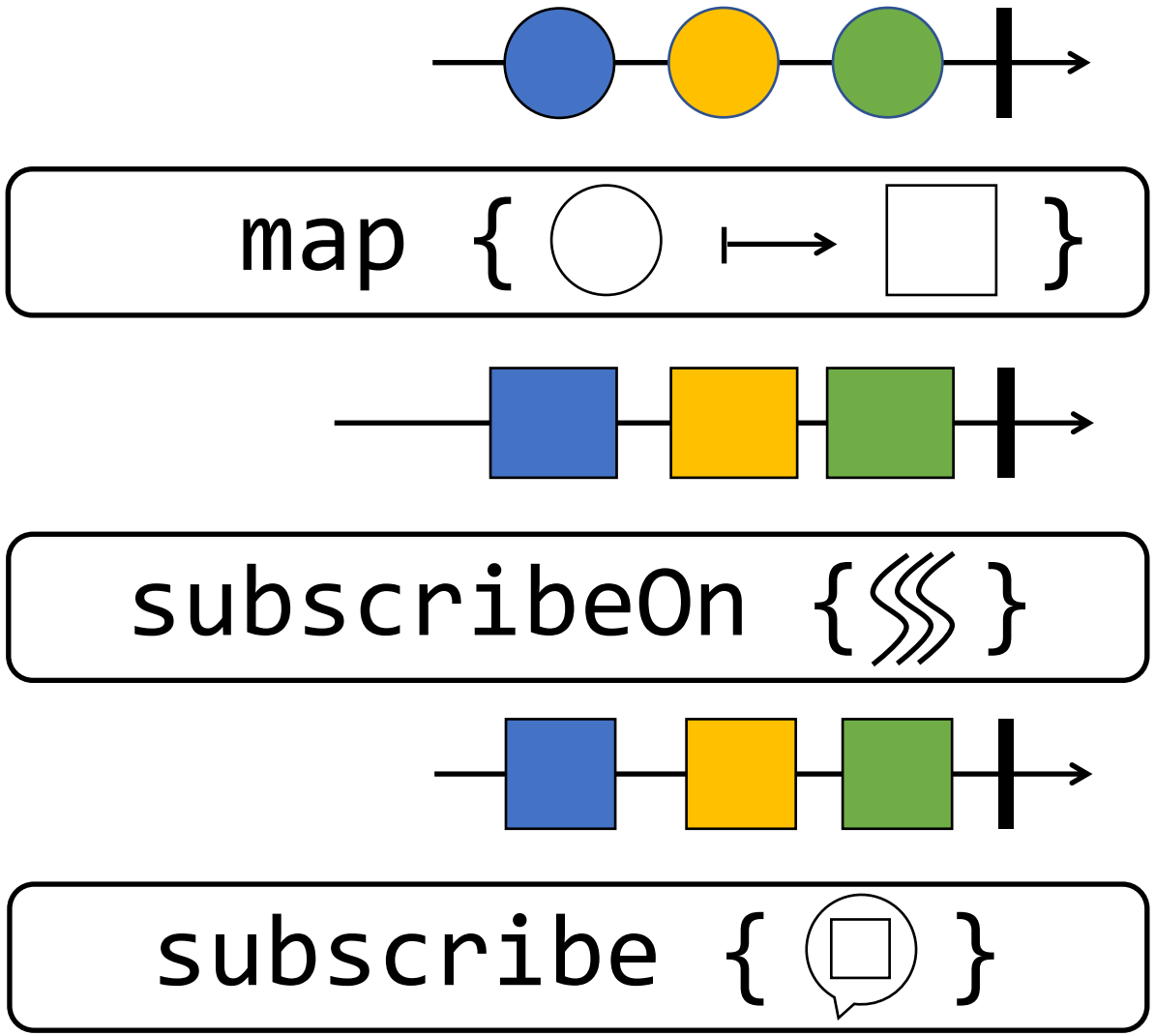
```
map { ○ ⟼ □ }
```

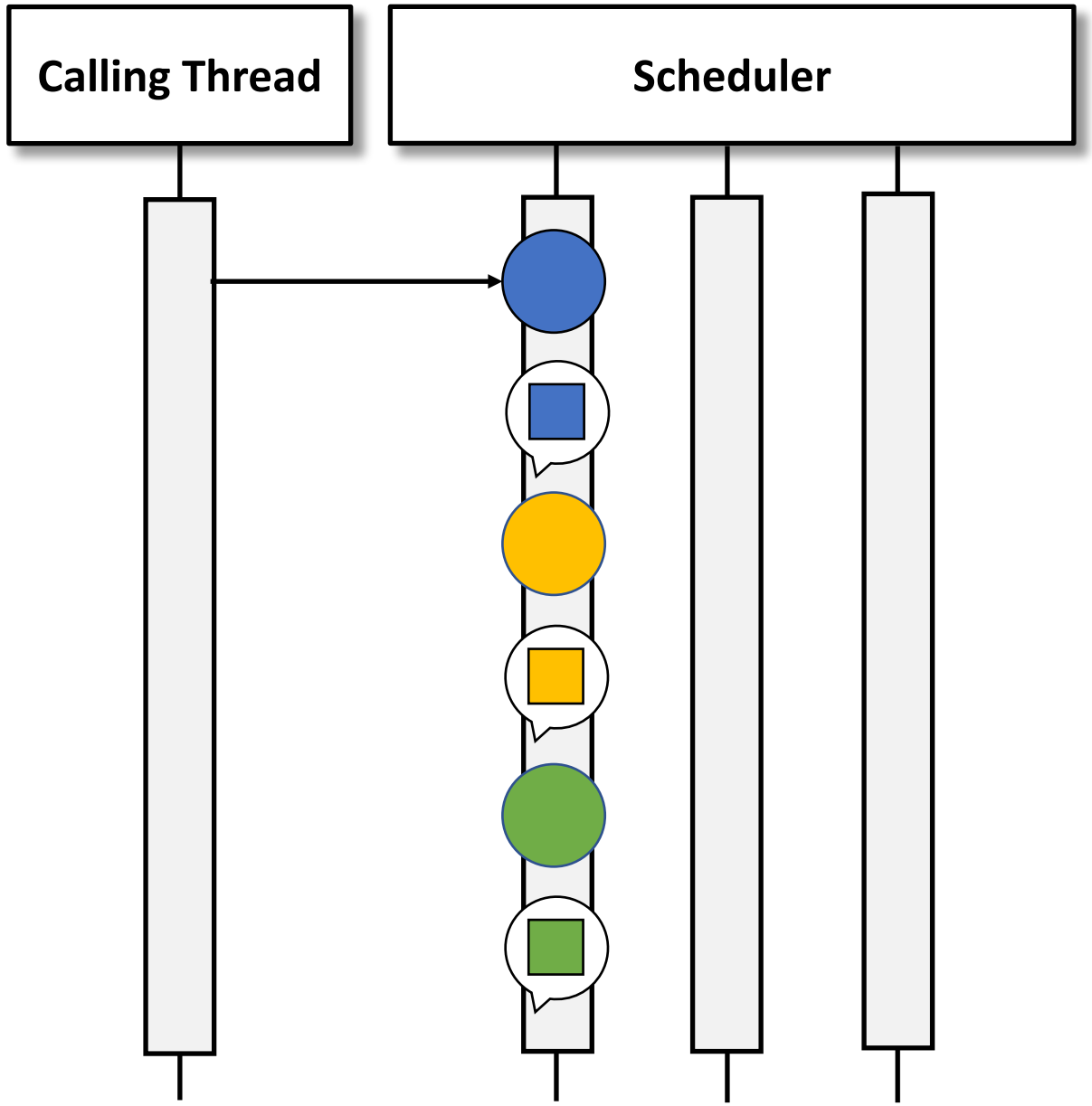


```
subscribe { □ }
```

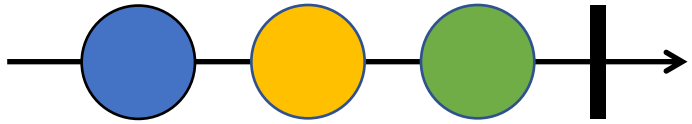


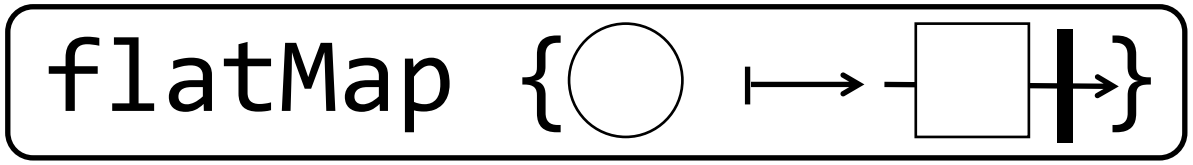
Concurrency

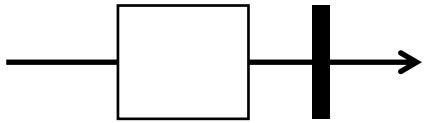


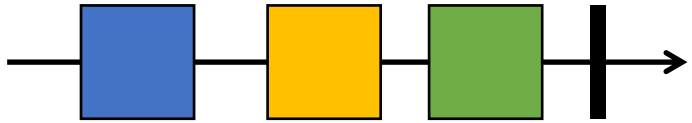


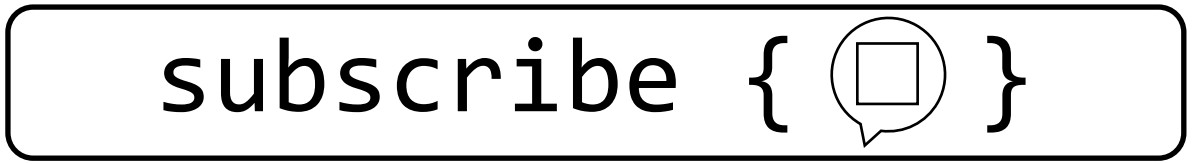
Concurrency

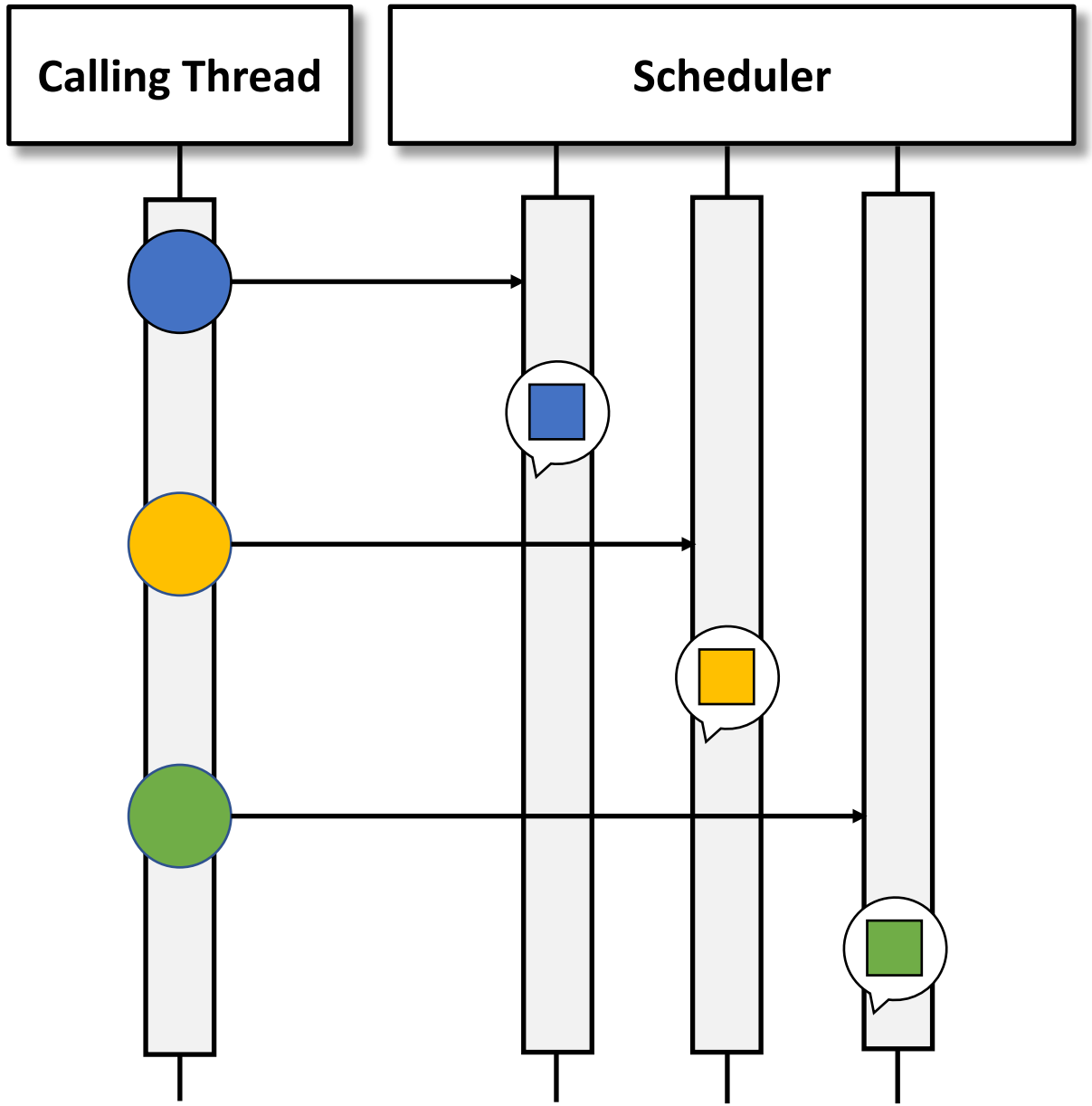








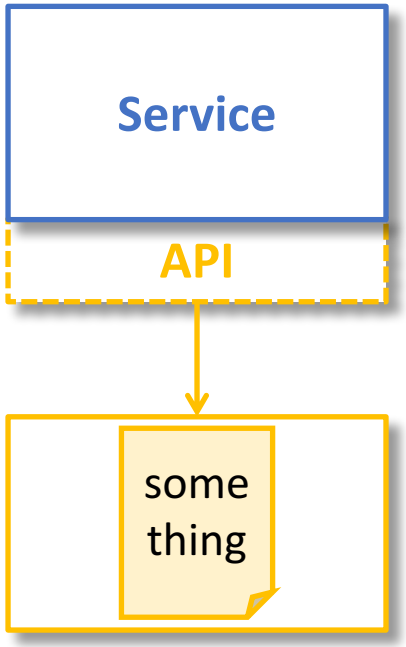




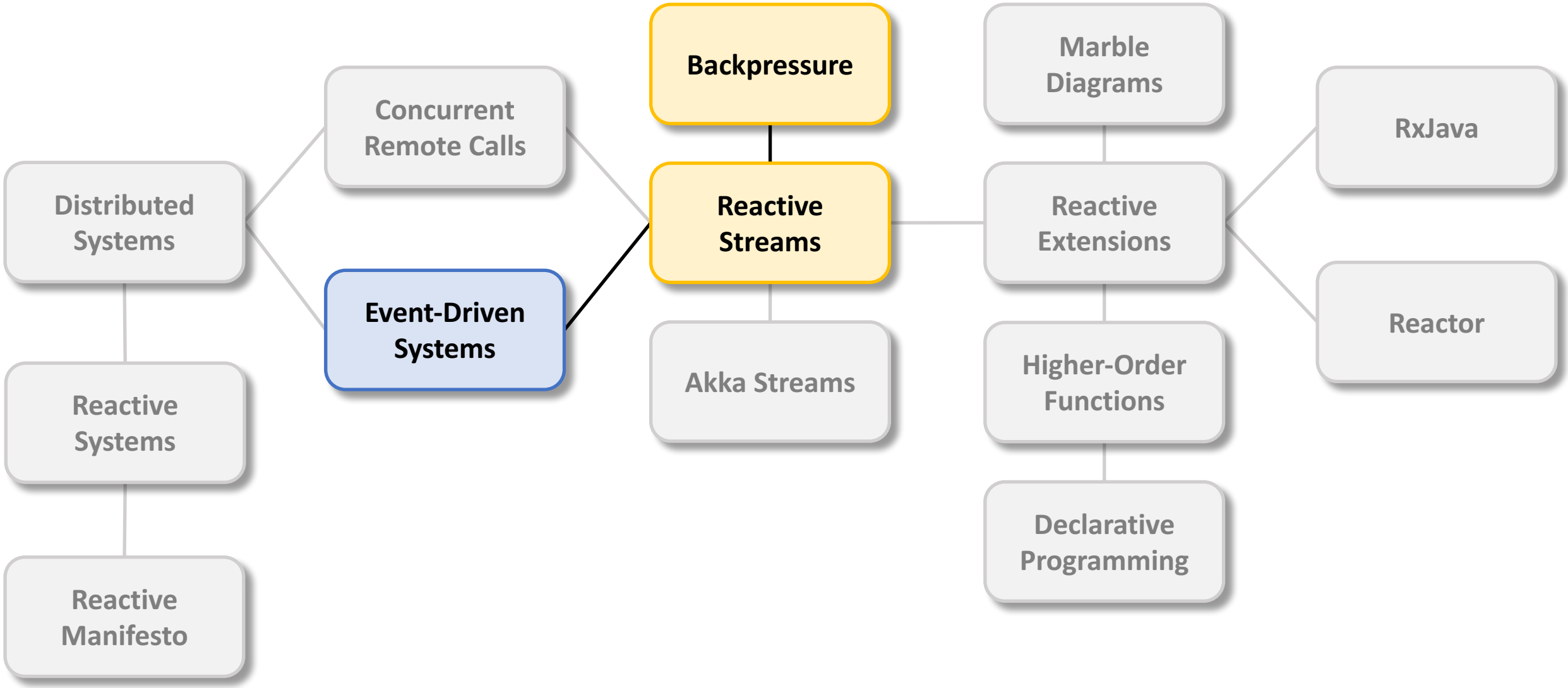
Concurrency

```
Scheduler scheduler =
    Schedulers.newParallel("pool", 8);
```

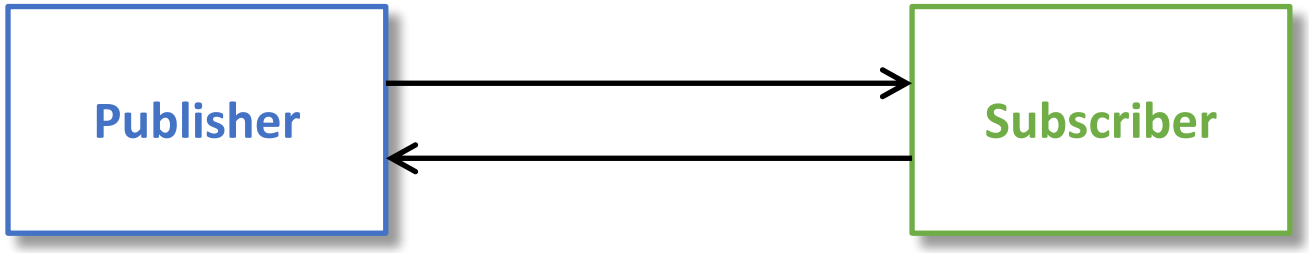
```
Flux.fromIterable(ids)

    .flatMap(id -> Mono.just(id)
        .map(somethings::loadById)
        .subscribeOn(scheduler))

    .log()

    .subscribe();
```




```
[main] INFO - onSubscribe()
[main] INFO - ?
[pool-1] INFO - onNext(Something(id=1))
[pool-2] INFO - onNext(Something(id=2))
[pool-3] INFO - onNext(Something(id=3))
[pool-3] INFO - onComplete()
```

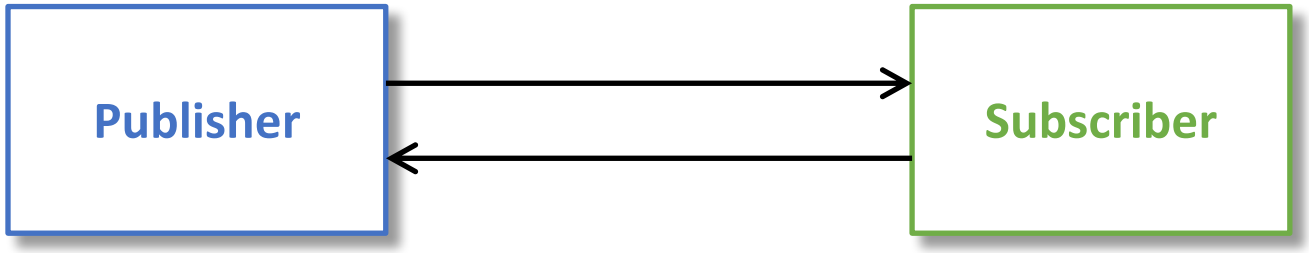

Backpressure
Concurrent Remote Calls
Distributed Systems
Reactive Streams
Event-Driven Systems
Akka Streams
Reactive Systems
Reactive Manifesto
Marble Diagrams
Reactive Extensions
RxJava
Reactor
Higher-Order Functions
Declarative Programming

Backpressure

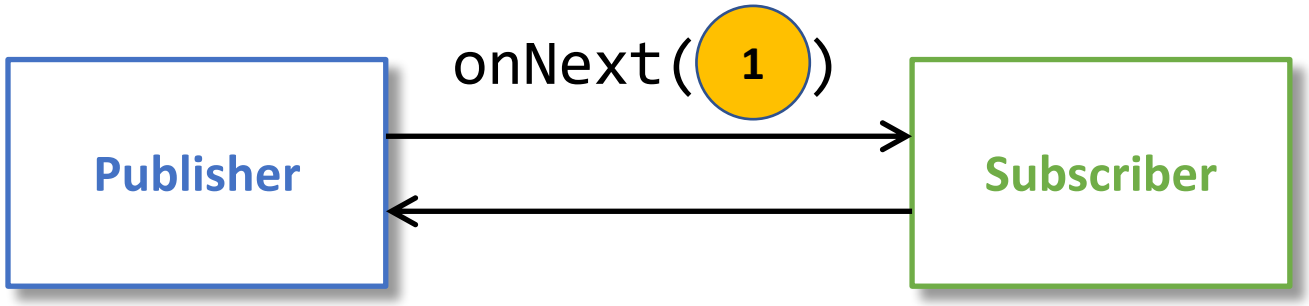


Backpressure

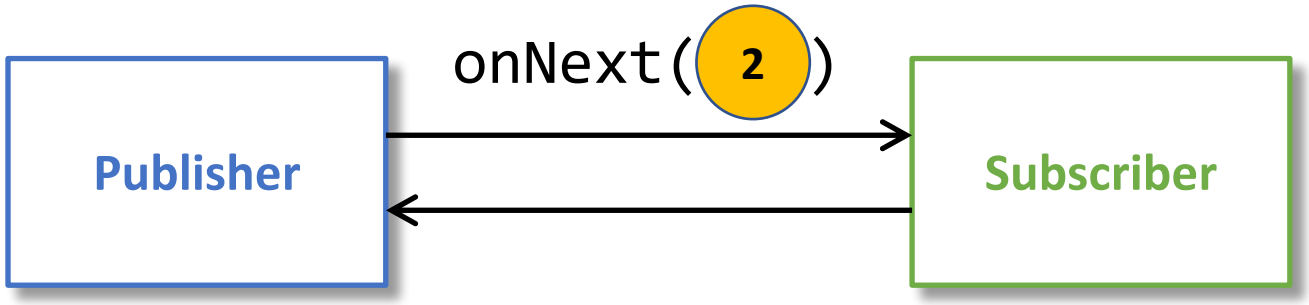
Backpressure



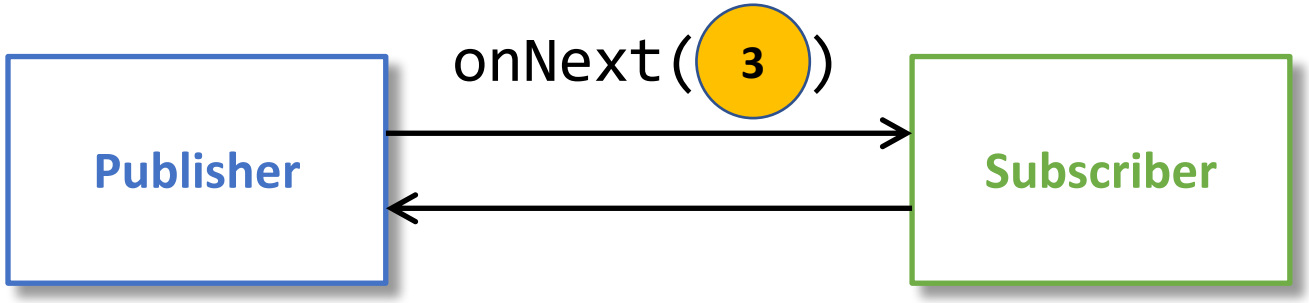
Backpressure



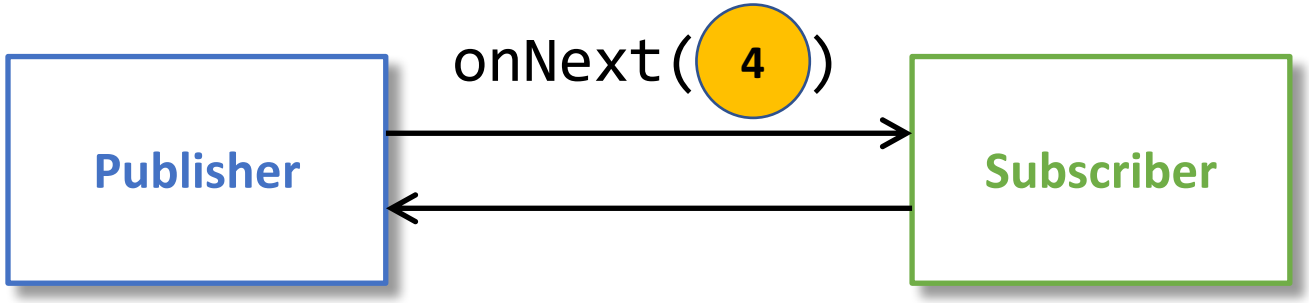
Backpressure



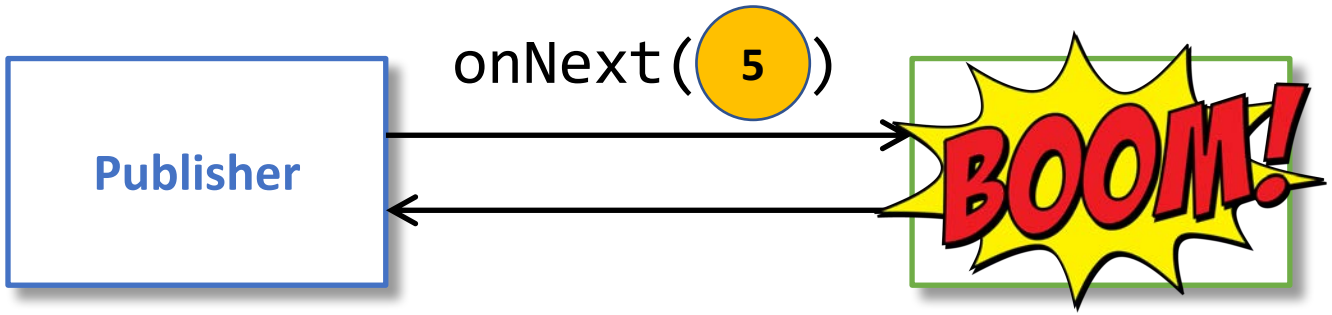
Backpressure



Backpressure

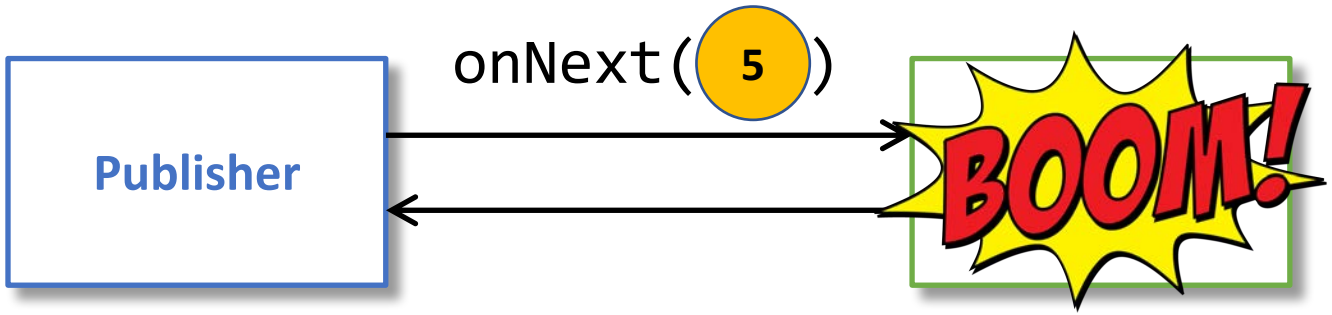


Backpressure



Backpressure

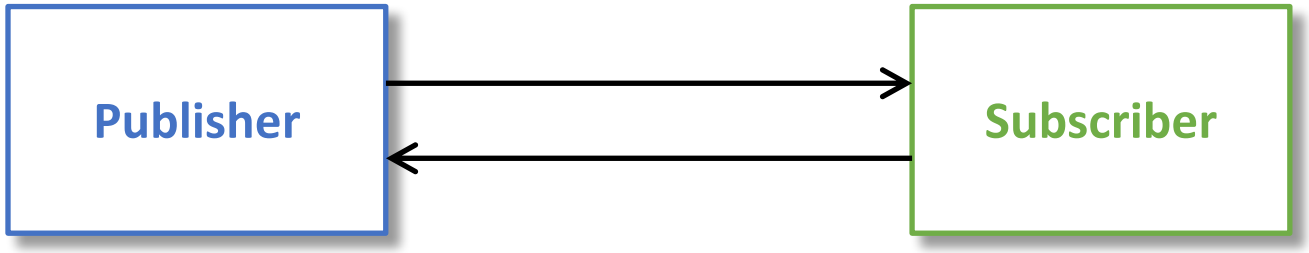
```
java.util.concurrent.RejectedExecutionException:
Task rejected from ThreadPoolExecutor [
  Running, pool size = 2, active threads = 2,
  queued tasks = 4, completed tasks = 0
]
```




```
ExecutorService executorService =
  new ThreadPoolExecutor(
      nThreads, nThreads,
      0L, TimeUnit.MILLISECONDS,
      new LinkedBlockingQueue<>(capacity)
  );
```

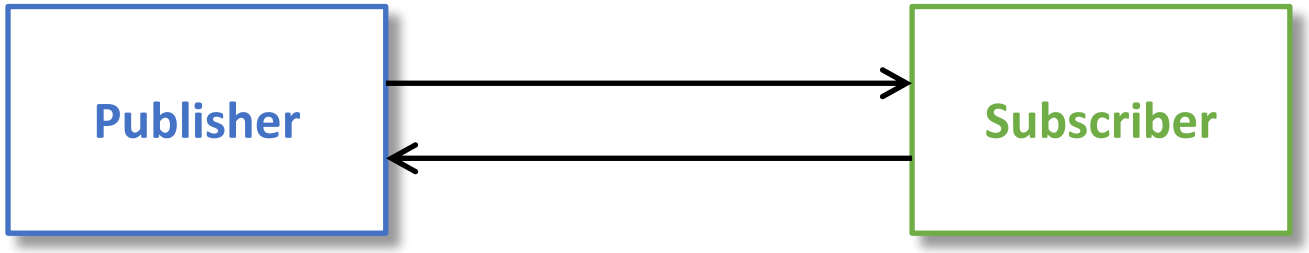
```
ids.stream()
   .map(id -> executorService
     .submit(() -> somethings.loadById(id)))
   .collect(toList())
   .stream()
   .map(waitForSomething())
   .forEach(handleSomething());
```

Backpressure

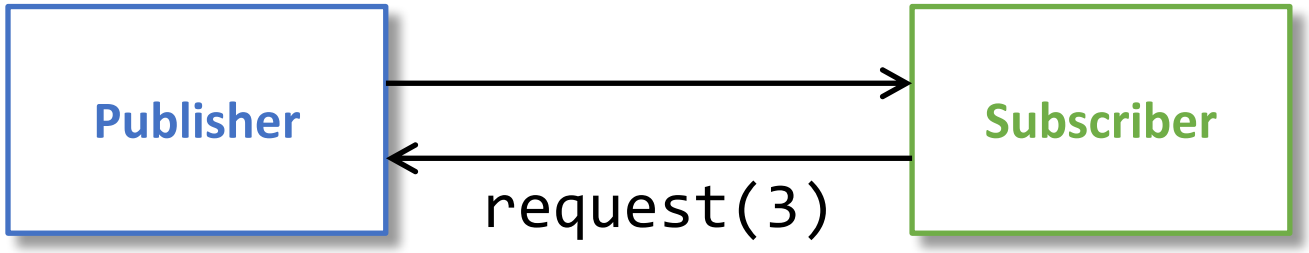


Backpressure

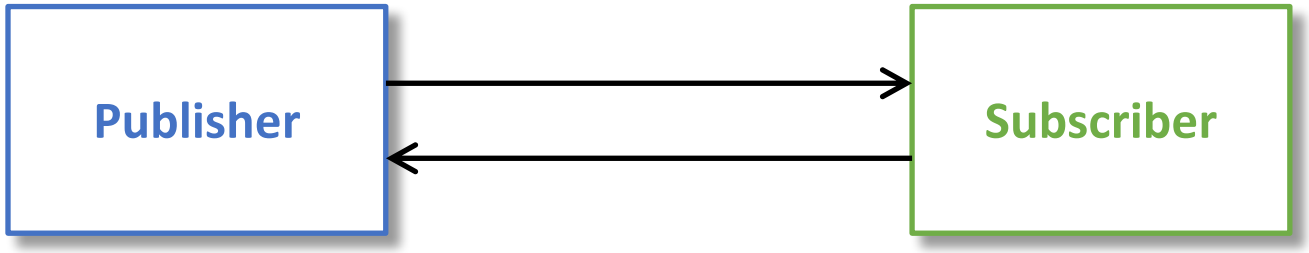
Backpressure



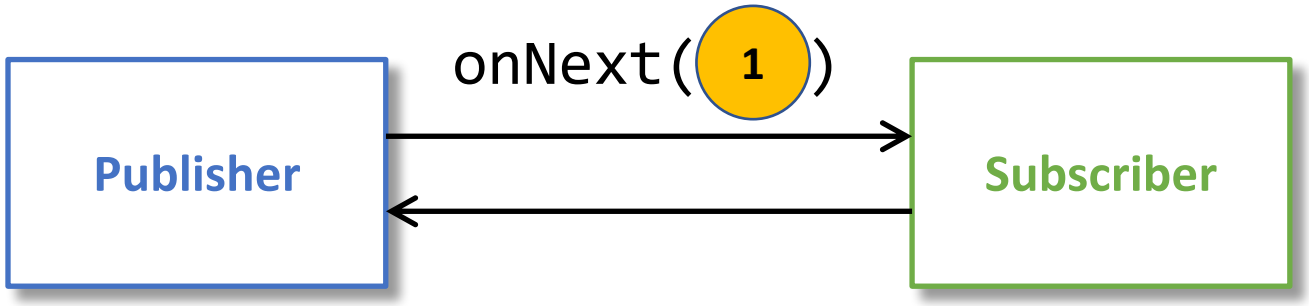
Backpressure



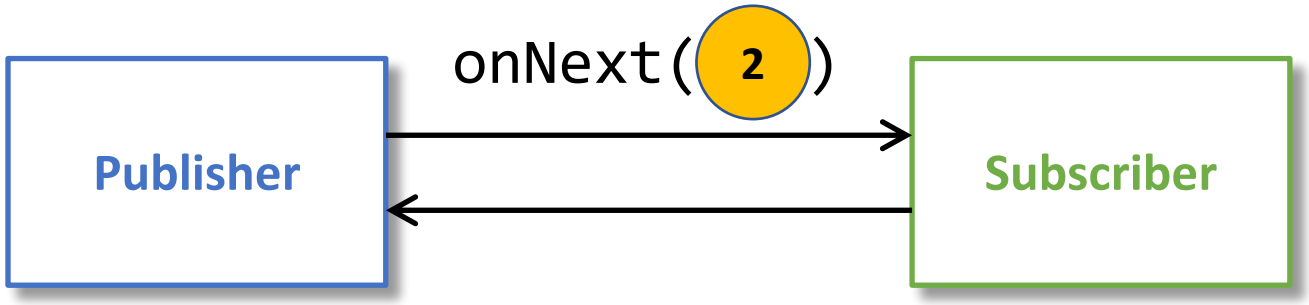
Backpressure



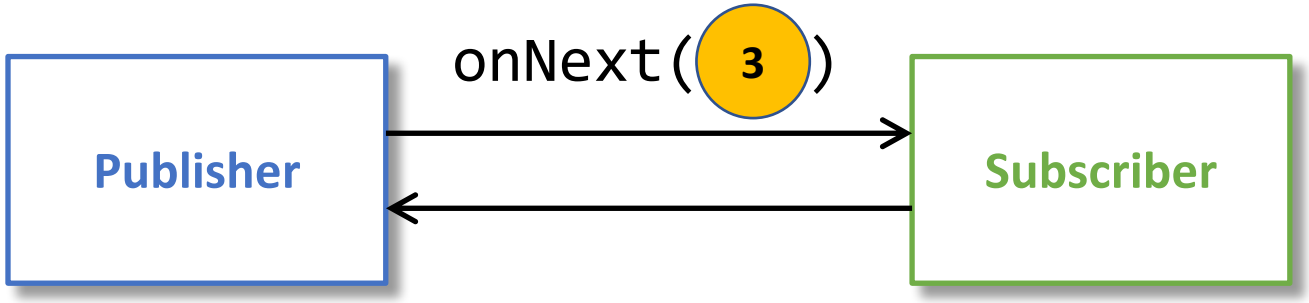
Backpressure



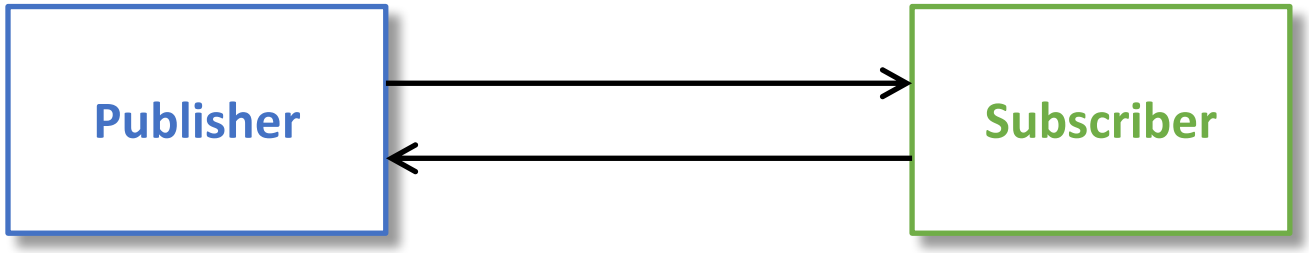
Backpressure



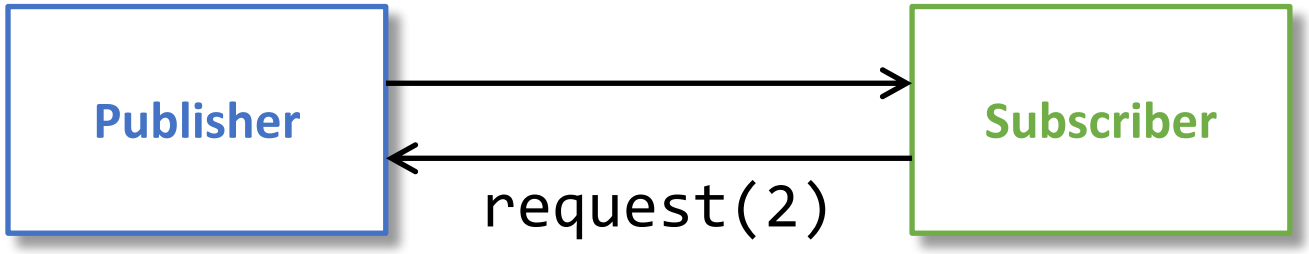
Backpressure



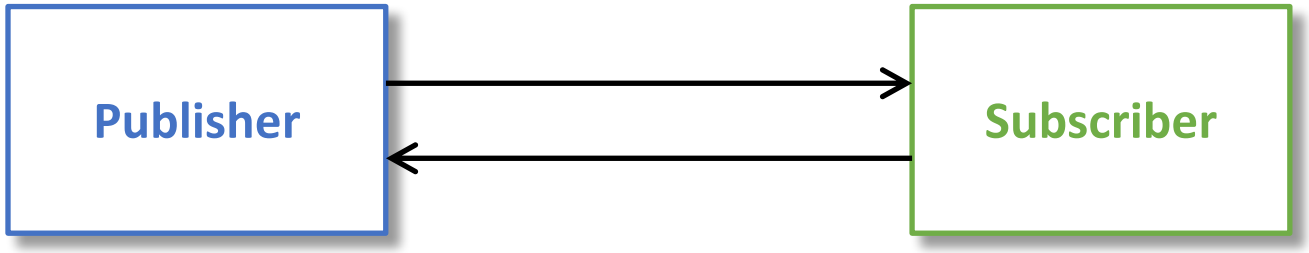
Backpressure



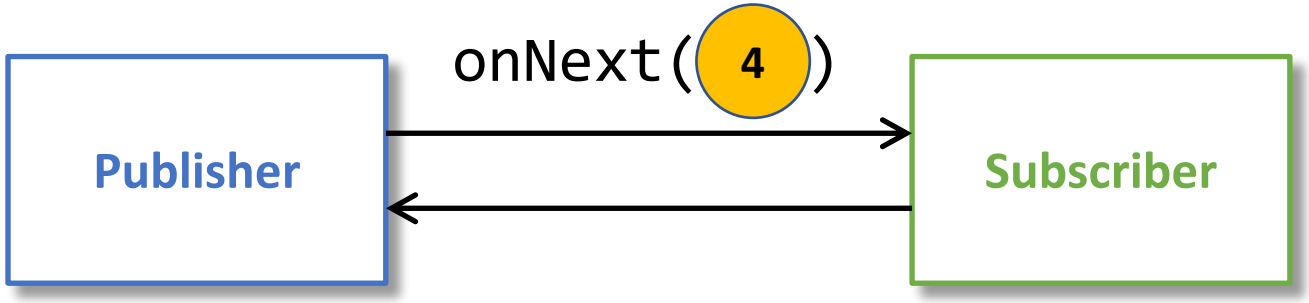
Backpressure



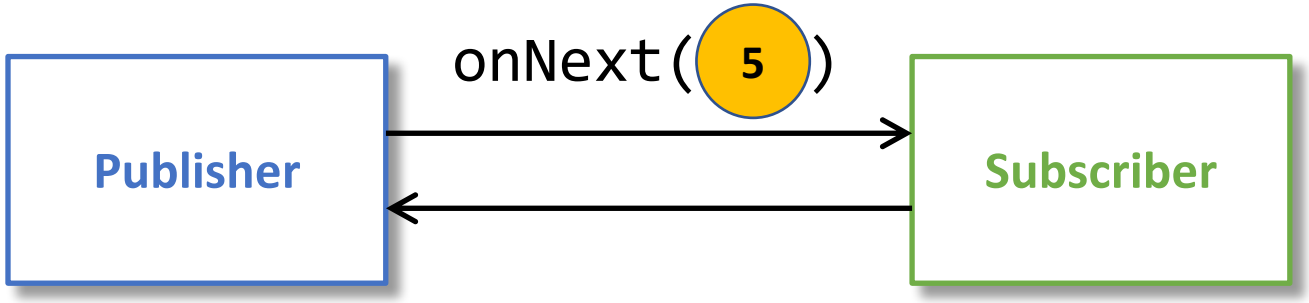
Backpressure



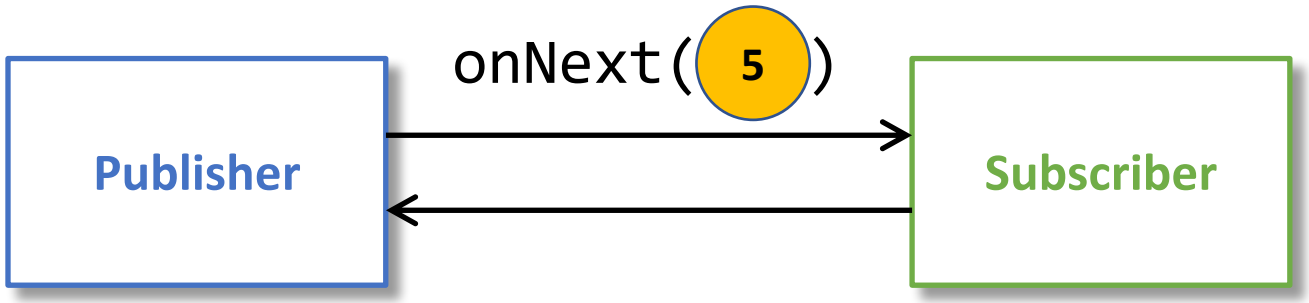
Backpressure



Backpressure



Backpressure



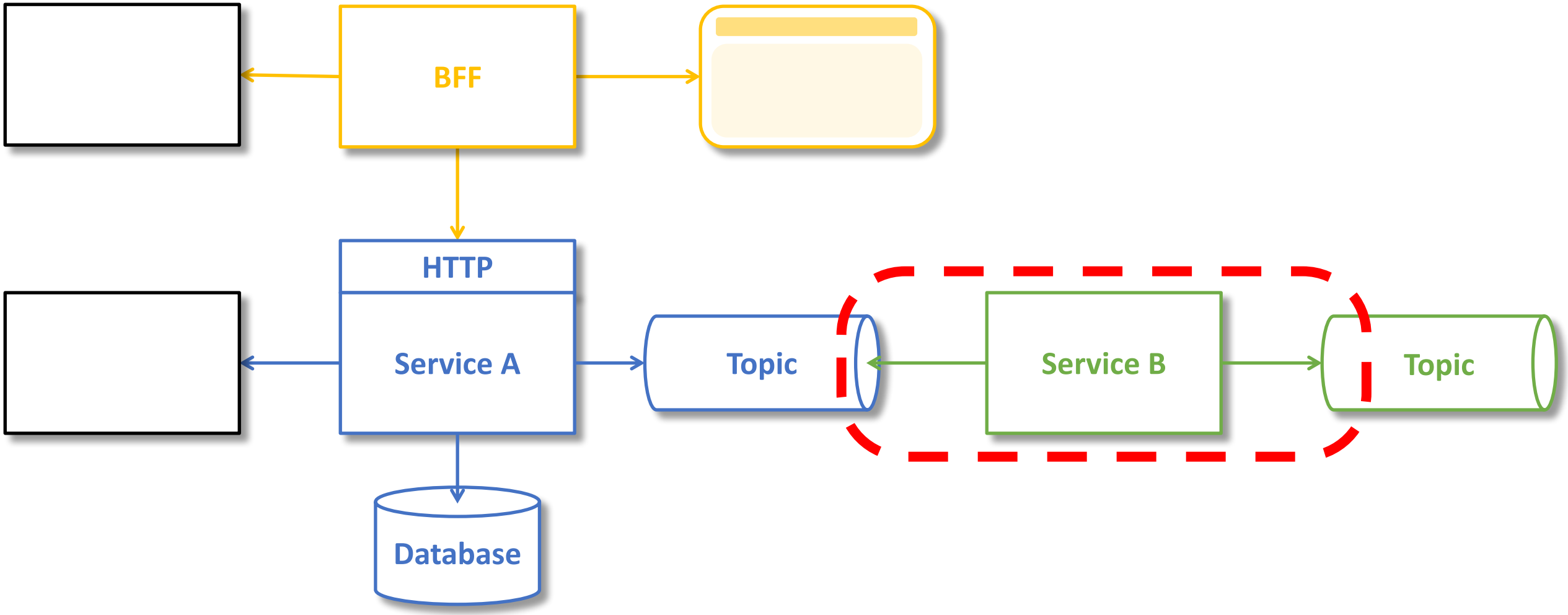

```
[main] INFO - | onSubscribe()
[main] INFO - | request(3)
[main] INFO - | onNext(1)
[main] INFO - | onNext(2)
[main] INFO - | onNext(3)
[pool-1] INFO - | request(1)
[pool-1] INFO - | onNext(4)
[pool-2] INFO - | request(1)
[pool-2] INFO - | onNext(5)
[pool-2] INFO - | onComplete()
```

```
ExecutorService executorService =
  new ThreadPoolExecutor(
      nThreads, nThreads,
      0L, TimeUnit.MILLISECONDS,
      new LinkedBlockingQueue<>(capacity)
  );
```

```
Scheduler scheduler = Schedulers
  .fromExecutorService(executorService);
```

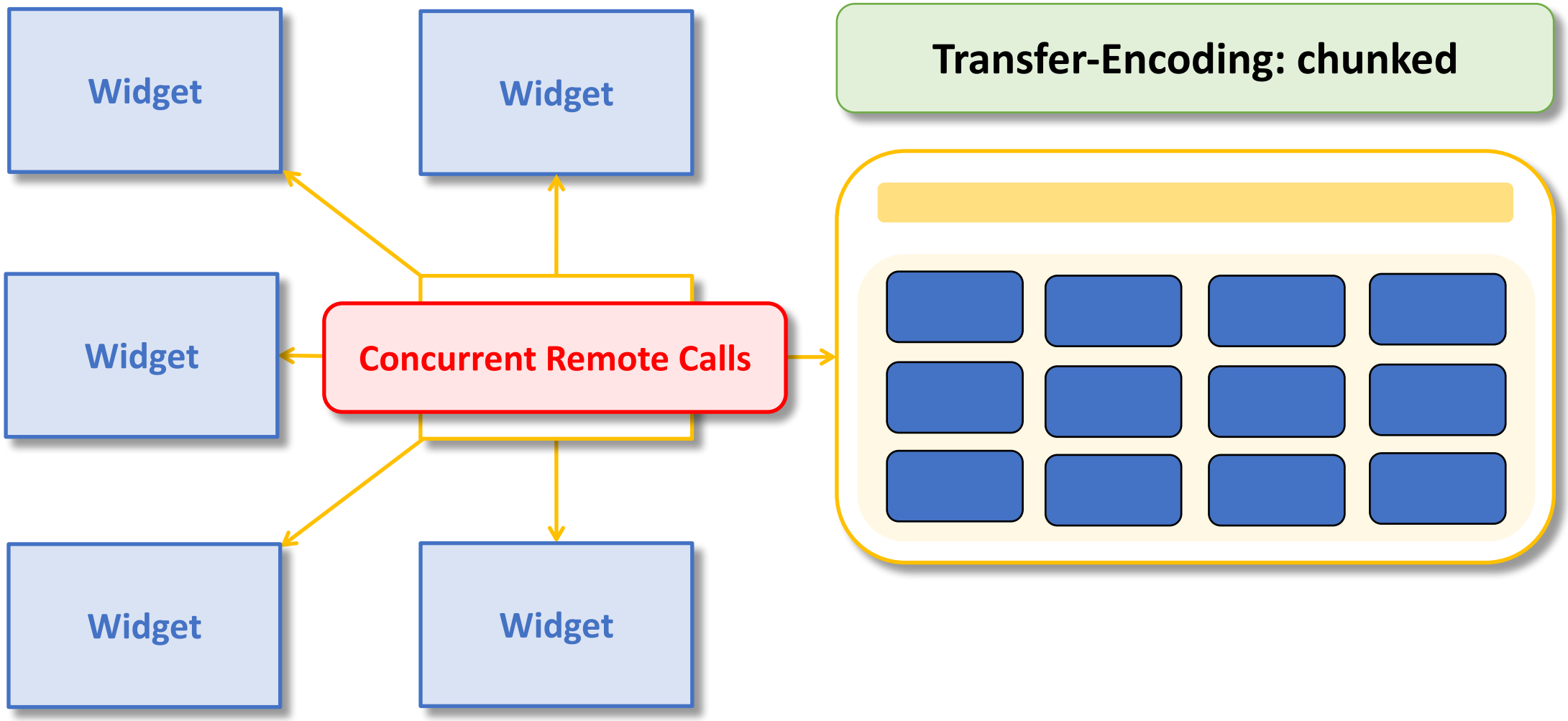
```
Flux.fromIterable(ids)
  .log()
  .flatMap(id -> Mono.just(id)
    .map(somethings::loadById)
    .subscribeOn(scheduler), 3)
  .doOnNext(handleSomething())
  .subscribe();
```

Backpressure



Progressive HTML Rendering

Progressive HTML Rendering



Progressive HTML Rendering

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

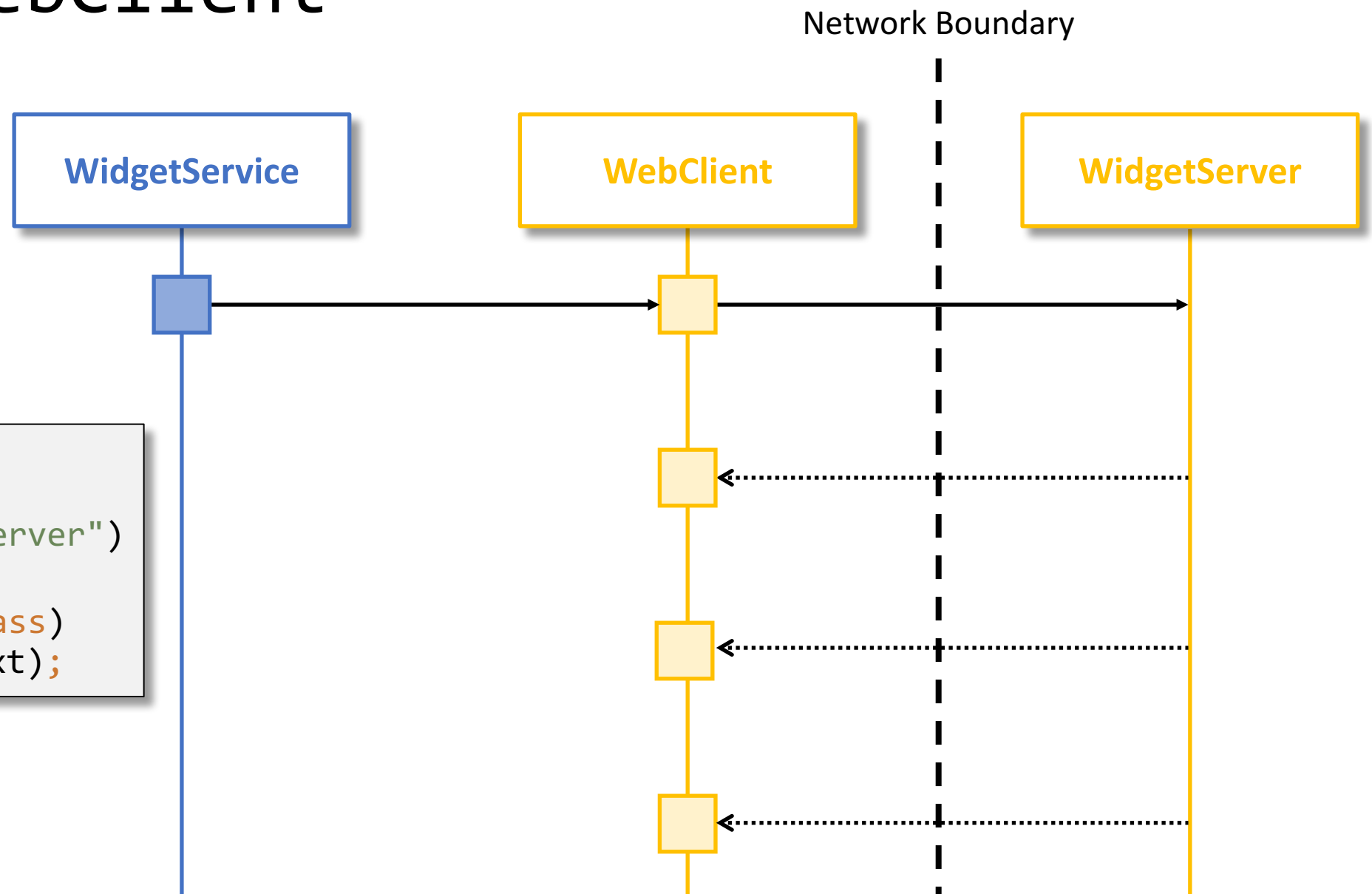
Progressive HTML Rendering

```
curl -i -N localhost:8080/hello

HTTP/1.1 200
Content-Type: text/plain
Transfer-Encoding: chunked

Hello
Hello
Hello
Hello
Hello
```

```
@SpringBootApplication
@RestController
public class Application {

  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }

  @GetMapping(value = "hello")
  public Flux<String> sayHello() {
    return Flux.interval(Duration.ofMillis(500)).map(tick -> "Hello\n");
  }

}
```

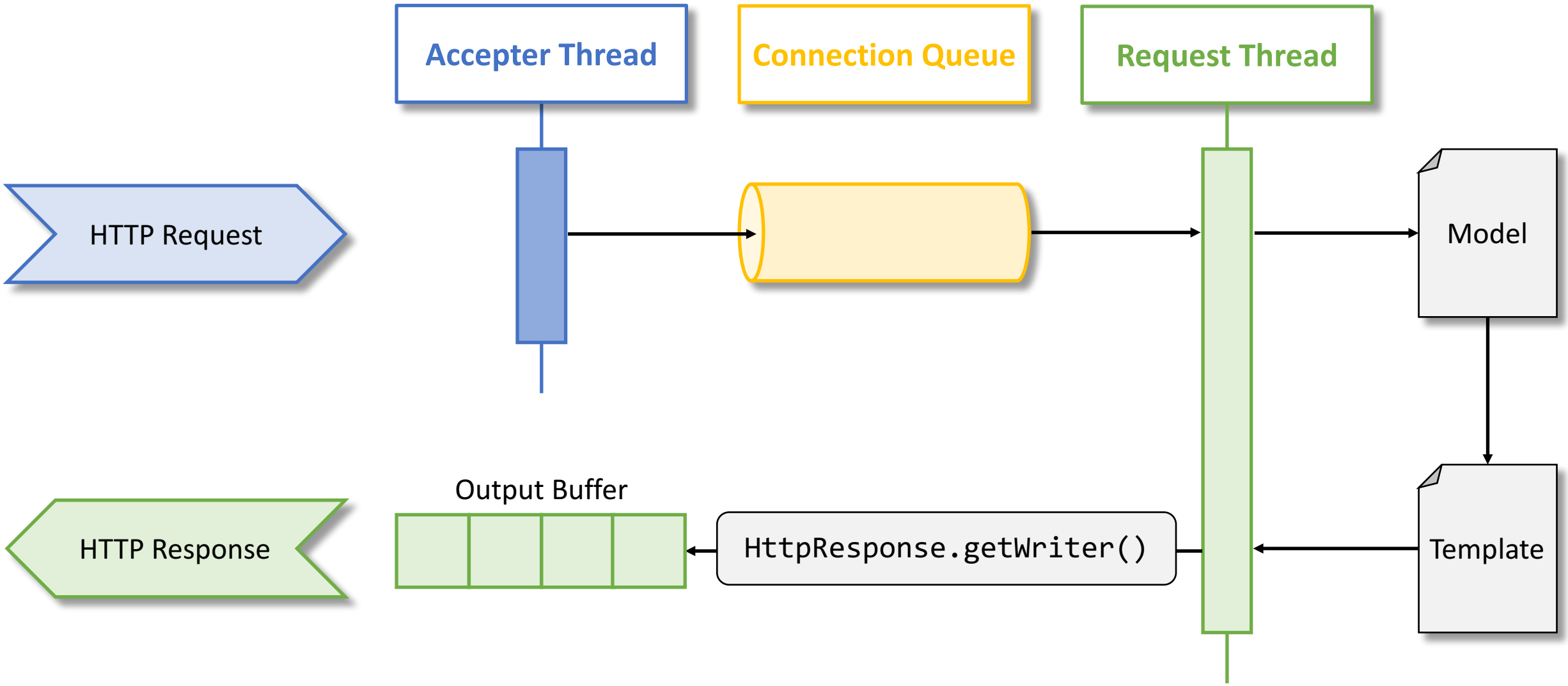
Baking an HTML stream

Spring 5 `WebClient`

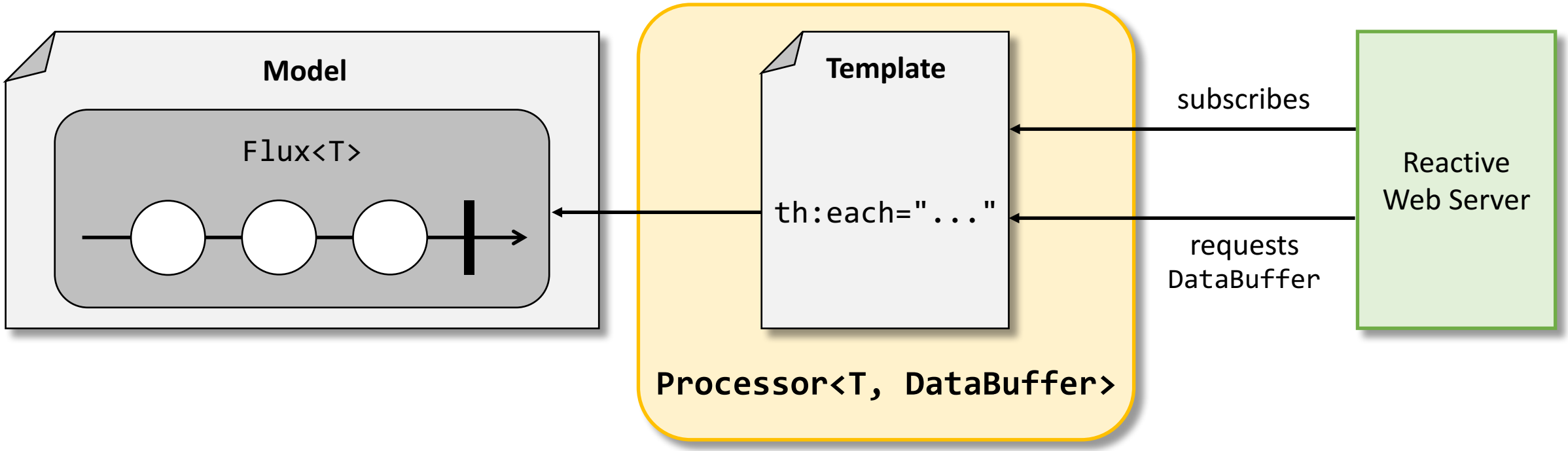



```
WebClient.create()
   .get()
   .uri("http://widget-server")
   .retrieve()
   .bodyToFlux(String.class)
   .subscribe(this::onNext);
```

HTML Rendering

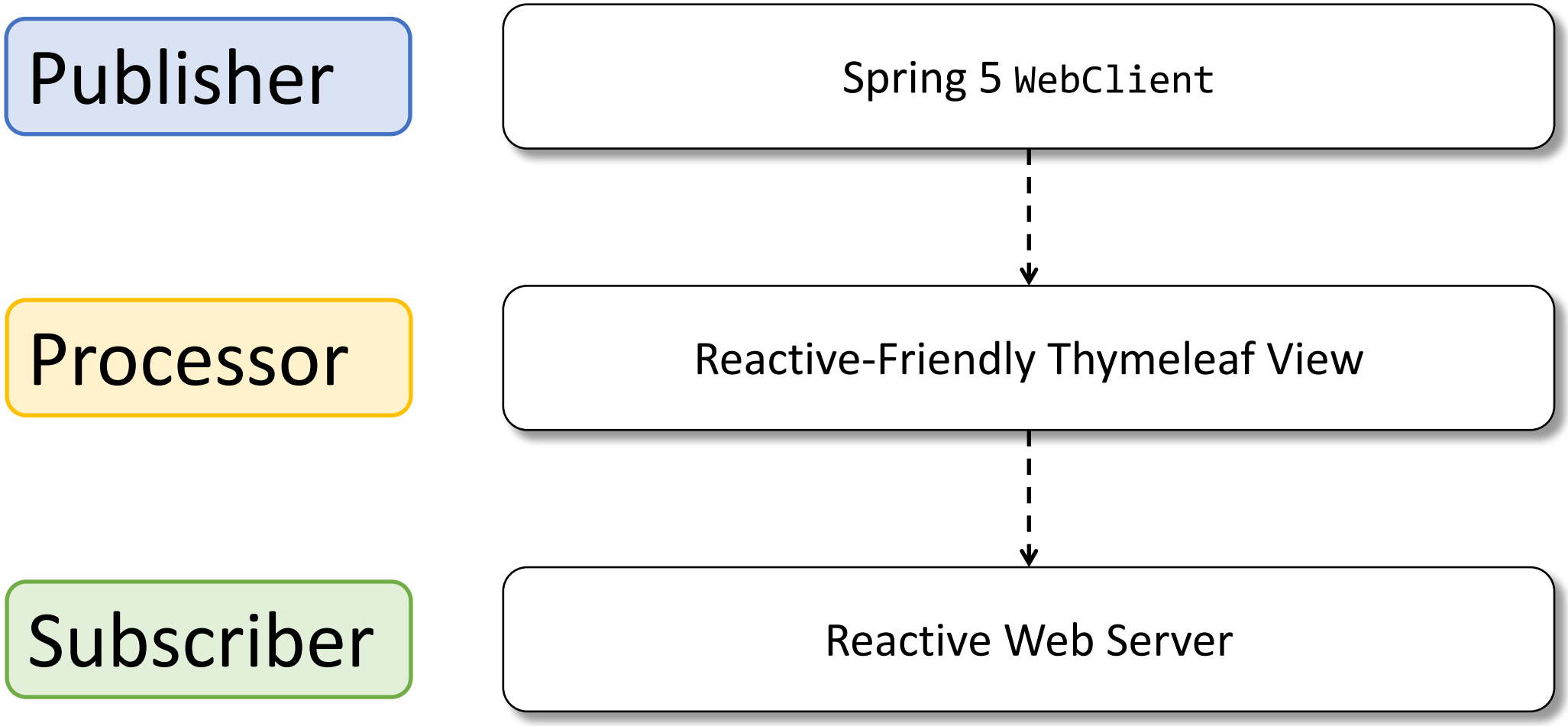


Reactive-Friendly Template Engine



HTML rendered as values are published

Progressive HTML Rendering



Progressive HTML Rendering

Publisher

Spring 5 `WebClient`

```
URI uri = ...
String name = ...

Mono<Pagelet> pageletMono = webClient.get().uri(uri).exchange()
  .flatMap(res -> res.bodyToMono(String.class))
  .map(body -> new Pagelet(name, body));
```

```
List<Mono<Pagelet>> pageletMonos = ...;

... = Flux.merge(pageletMonos);
```

Progressive HTML Rendering

Processor

Reactive Thymeleaf View

```
@GetMapping(value = "/widgets")
public String widgets(Model model) {
  List<String> pageletNames = widgetService.getPageletNames();
  Flux<Pagelet> pagelets = widgetService.loadPagelets();

  model.addAttribute("pageletNames", pageletNames);
  model.addAttribute("pagelets", new ReactiveDataDriverContextVariable(pagelets, 1));

  return "widgets";
}
```

Progressive HTML Rendering

Processor

Reactive Thymeleaf View

```
<article th:each="pageletName: ${pageletNames}" class="...">
  <section class="...">
    <div th:id="${pageletName}" class="...">
      <div class="...">...</div>
    </div>
  </section>
</article>
```

Progressive HTML Rendering

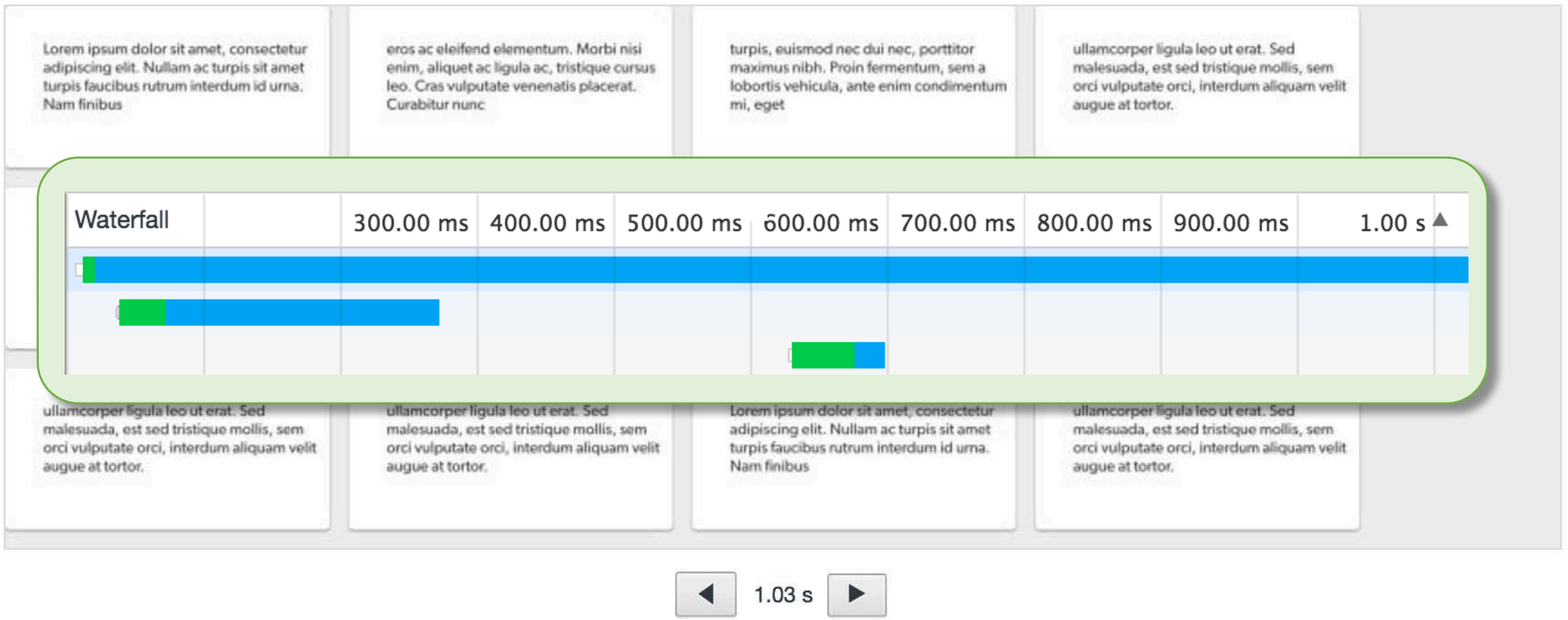
Processor

Reactive Thymeleaf View

```
<script th-inline="javascript" th:each="pagelet : ${pagelets}">
    (function() {
        var element = document.getElementById("[[${pagelet.name}]]");
        if(element) {
            element.innerHTML = "[(${pagelet.content})]";
        }
    })();
</script>
```

Does it work?

Progressive HTML Rendering



What‘s more?

- Cold Streams vs. Hot Streams
- Reactive Database Drivers
- Backpressure Strategies
- Threaded vs. Evented Server Model
- WebSockets and Server-Sent Events

Thank you!

Questions?