Reactive Programming

— Nandakumar Purohit

Reactive Programming

Agenda

- The Reactive Manifesto
- Reactive Use Case

Reactive Programming

The Reactive Manifesto

Times have changed

<= Milliseconds of Response time

100% Availability

An exponential increase in Data Volume

Multi-platform Applications

Multi-device Client Applications

Cloud Native Apps with PaaS

https://www.reactivemanifesto.org/

- We believe that a **coherent approach** to systems architecture is needed
- We believe that all necessary **aspects** are already **recognized individually**
- We want systems that are **Responsive, Resilient, Elastic, and Message Driven**
- We call these **Reactive Systems.**
- Systems built as Reactive Systems are more **flexible, loosely coupled, and scalable**
- They meet it with **elegance** rather than **disaster**.

Reactive Programming

Why Reactive?

1. Gaining an understanding of the **central principles** of designing robust systems

2. It should be **reactive to any changes** that may affect the system's ability to respond to user requests

3. One of the first ways to achieve the primary goal is through **elasticity**

4. Building a scalable distributed system **without the ability to stay responsive** regardless of failures is a **challenge**

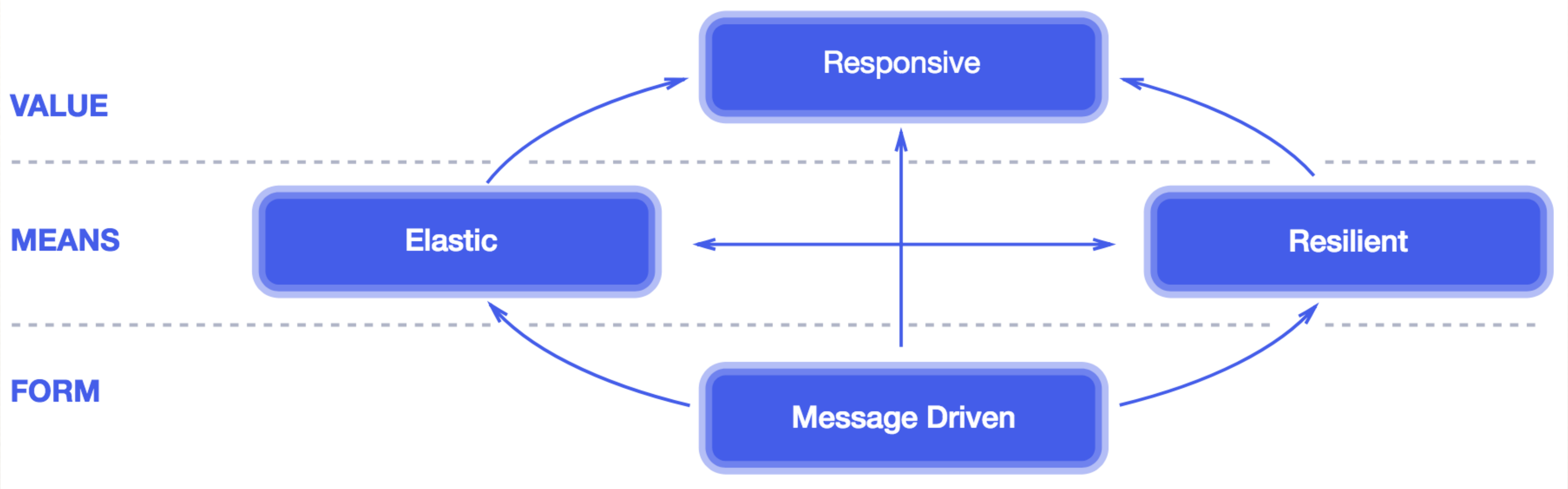
5. The acceptance criteria for the system are the ability to stay responsive under failures, or, in other words, to be **resilient**

6. Resilience can be achieved by **Isolation**

7. ONLY a combination of **Elasticity & Resilience** can make a System as **Responsive**

Reactive Programming

Why Reactive?



The primary **value** for any business implemented with a distributed system is responsiveness.

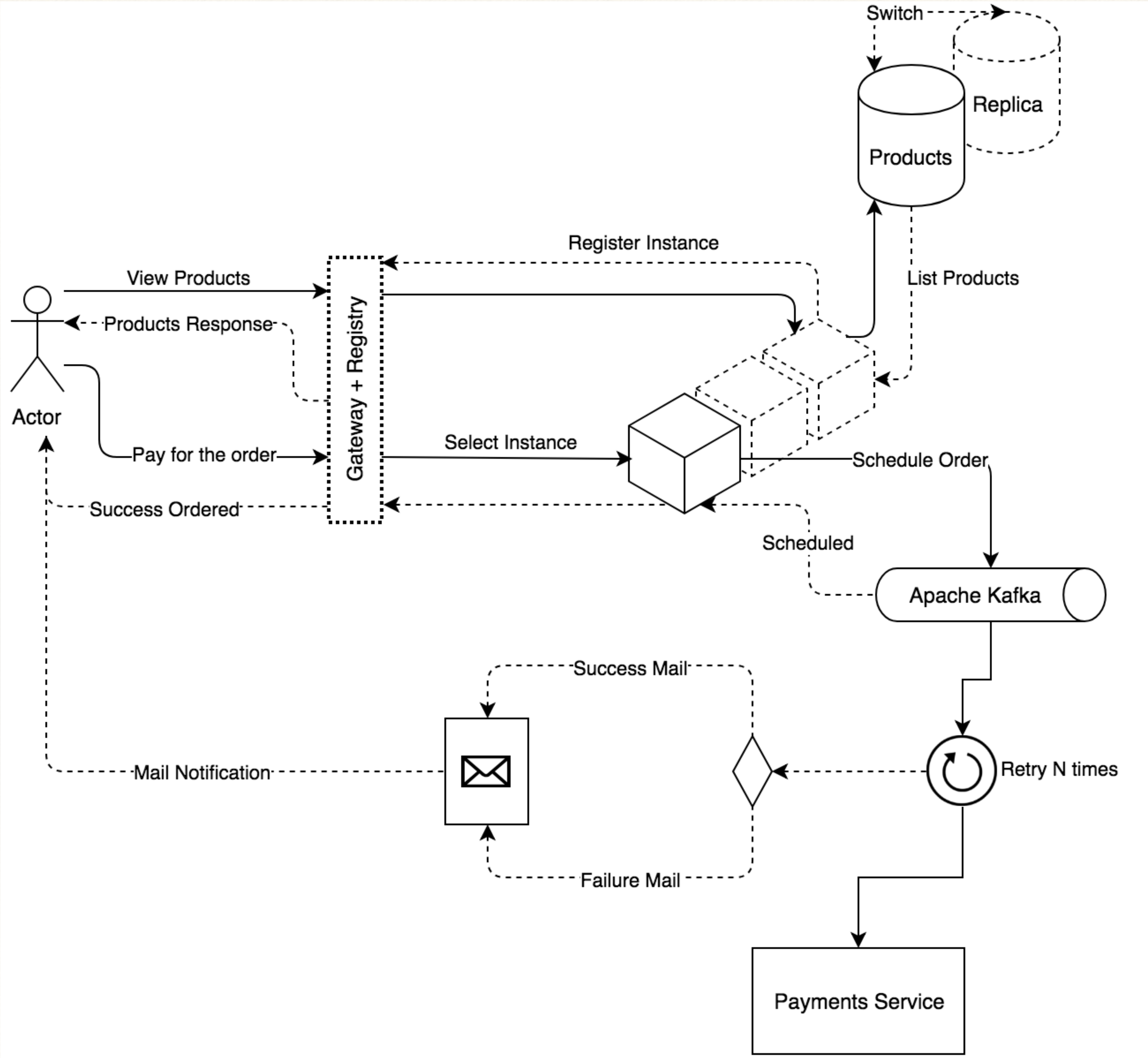
Achieving a responsive system **means** following fundamental techniques such as elasticity and resilience.

Finally, one of the fundamental **ways (form)** to attain a responsive, elastic, and resilient system is by employing message-driven communication.

In addition, systems built following such principles are highly maintainable and extensible, since all components in the system are independent and properly isolated.

Reactive Programming

Reactivity Real-time Use Case - Web Store



Reactive Programming

Reactivity Real-time Use Case - Web Store

- Here, we improved our small web store by applying modern **microservice patterns**.
- In that case, we use an **API Gateway pattern** for achieving location transparency.
- It provides the identification of a specific resource with **no knowledge about particular services** that are responsible for handling requests.
- In turn, the responsibility for keeping information about available services up to date is implemented using the **service registry pattern** and achieved with the support of the client-side discovery pattern.
- Additionally, the **high responsiveness** of the system is achieved by applying **replication to the service**.
- On the other hand, **failure tolerance** is attained by properly employed **message-driven communication** using **Apache Kafka** and the independent **Payment Proxy Service** which is responsible for **redelivering payment** in the case of **unavailability of the external system**.
- Also, we use **database replication** to stay **resilient** in the case of the **outage of one of the replicas**.
- To stay responsive, we return a **response about an accepted order immediately** and **asynchronously** process and send the user payment to the **payments service**.
- A **final notification** will be delivered later by one of the supported **channels**, for example, via **email**.

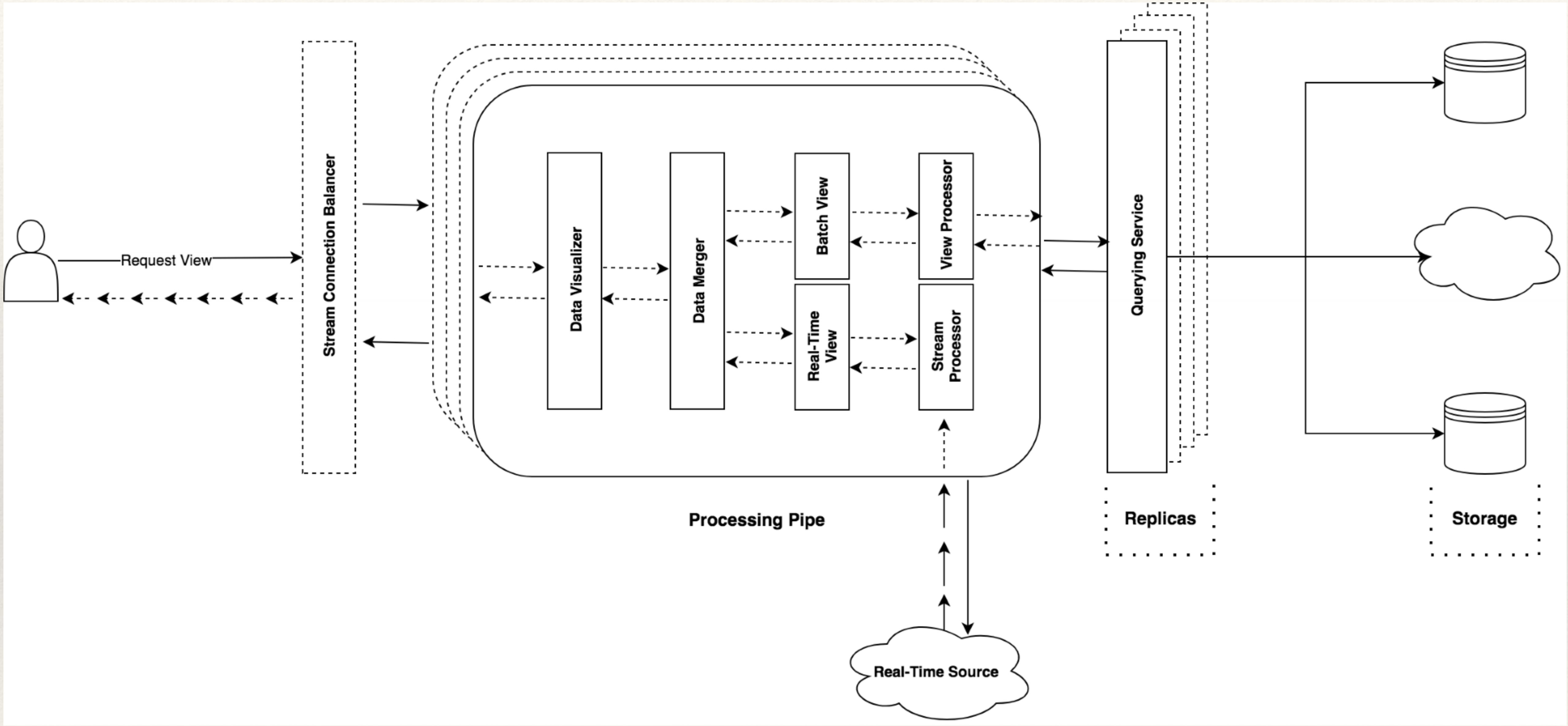
Reactive Programming

Reactivity Real-time Use Case - Analytics

- Suppose we are designing a system for monitoring a telecommunication network based on cell site data.
- Due to the latest statistic report of the number of cell towers, in 2016 there were 308,334 active sites in the USA.
- To design this system, we may follow one of the efficient architectural techniques called streaming.

Reactive Programming

Reactivity Real-time Use Case - Analytics



Reactive Programming

Reactivity Real-time Use Case - Analytics

- Streaming architecture is about the construction of the **flow of data processing and transformation**.
- In general, such a system is characterized by **low latency and high throughput**.
- In turn, the **ability to respond** or simply **deliver analyzed updates** of the telecommunication network state is therefore crucial.
- Thus, to build such a highly-available system, we have to rely on **fundamental principles**.
- For example, achieving **resilience** might be done by enabling **backpressure support**.
- **Backpressure** refers to a sophisticated mechanism of **workload management between processing stages** in such a way that ensures we do not overwhelm another.
- **Efficient workload management** may be achieved by using **message-driven communication** over a **reliable message broker**, which may persist messages internally and send messages on demand.
- Alternatively, the stream of data may be **processed in a batch** in the databases, or partially processed in real-time by applying **windowing or machine-learning techniques**.
- Anyways, all **fundamental principles** offered by the **Reactive Manifesto** are valid here, regardless of the overall domain or business idea.

Reactive Programming

Why Reactive Spring?

- In the JVM world, the most commonly known frameworks for building a reactive system has been **Akka and Vert.x ecosystems**.
- On one hand, **Akka** is a popular framework with a **huge list of features and a big community**.
- However, at the very beginning, **Akka was built as part of the Scala ecosystem** and for a long time, it **showed its power only within solutions written in Scala**.
- Despite the fact that **Scala is** a JVM-based language, it is **noticeably different from Java**.
- A few years ago, **Akka provided direct support for Java**, but for some reason, it was not as popular in the Java world as it was in Scala.
- On the other hand, there is the **Vert.x framework** which is also a powerful solution for building an efficient reactive system.
- **Vert.x** was designed as a **non-blocking, event-driven** alternative to **Node.js** that runs on the **JVM**.
- However, **Vert.x started being competitive only a few years ago** and during the last 15 years, the market for frameworks for **flexible robust application development has been held by the Spring Framework.**
- The **Spring Framework** provides wide possibilities for building a web application using a **developer-friendly programming model**.
- **However, for a long time, Spring had some limitations in building a robust reactive system.**

Reactive Programming

Reactive Programming in Spring

- Java 8 does not support Reactive Programming
- We will use **Reactive Streams, Reactor & Spring WebFlux**

"Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure. This encompasses efforts aimed at runtime environments (JVM and JavaScript) as well as network protocols."	Reactive streams aim to define a **minimal set of interfaces**, methods, and protocols to enable reactive programming.
	Reactive streams aim to be a **language-neutral approach** with implementation in the Java (JVM-based) and JavaScript languages.
	Multiple transport streams (TCP, UDP, HTTP, and WebSockets) are supported.

Reactive Programming

Reactive Streams

Interfaces in reactive-streams

```
public interface Subscriber<T> {
      public void onSubscribe(Subscription s);
      public void onNext(T t);
      public void onError(Throwable t);
      public void onComplete();
    }

   public interface Publisher<T> {
     public void subscribe(Subscriber<? super T> s);
   }

   public interface Subscription {
     public void request(long n);
     public void cancel();
  }
```

Reactive Programming

Reactive Streams

Interface Publisher	• Publisher provides a **stream of elements** in response to the demand received from its subscribers. • A publisher can serve **any number of subscribers**. The subscriber count might vary with time.
Interface Subscriber	• **Subscriber registers** to listen to the stream of events. • Subscribing is a **two-step process**. • The first step is calling **Publisher.subscribe (Subscriber)**. • The second step involves making a call to **Subscription.request (long)**. • Once these steps are completed, the subscriber can start processing notifications using the **onNext(T t)** method. • The **onComplete()** method signals the end of the notifications.
Interface Subscription	• Subscription represents the **link between one Subscriber and its Publisher**. • A subscriber can request **more data using request(long n).** • It can cancel the subscription to notifications using the **cancel() method.**

Reactive Programming

Reactive Programming in Spring

Exploring Reactor Framework

```xml
<dependency>
    <groupId>io.projectreactor</groupId>
    <artifactId>reactor-core</artifactId>
</dependency>

<dependency>
<groupId>io.projectreactor.addons</groupId>
    <artifactId>reactor-test</artifactId>
    <version>3.0.6.RELEASE</version>
</dependency>
```

- Reactor is a Reactive Framework from **Spring Pivotal** Team
- Spring 5 uses Reactive Framework to enable Reactive Web features
- Reactor adds important APIs on top of Reactive Streams - **Flux & Mono**

Flux	Flux represents a Reactive stream that **emits 0 to n elements**.
Mono	Mono represents a Reactive stream that emits either **no elements or one element**.

Reactive Programming

Observer Use Case - Stock Price change

The use case we want to build is a stock price page that notifies all the subscribers about the price change.

There are 2 Observers – Trader & Trading Agent

Each Observer must register with the Subject to get change notification.

Both of these observers make different decisions based on the stock price change.

Reactive Programming

Reactive Use Case - A Stock Price Page

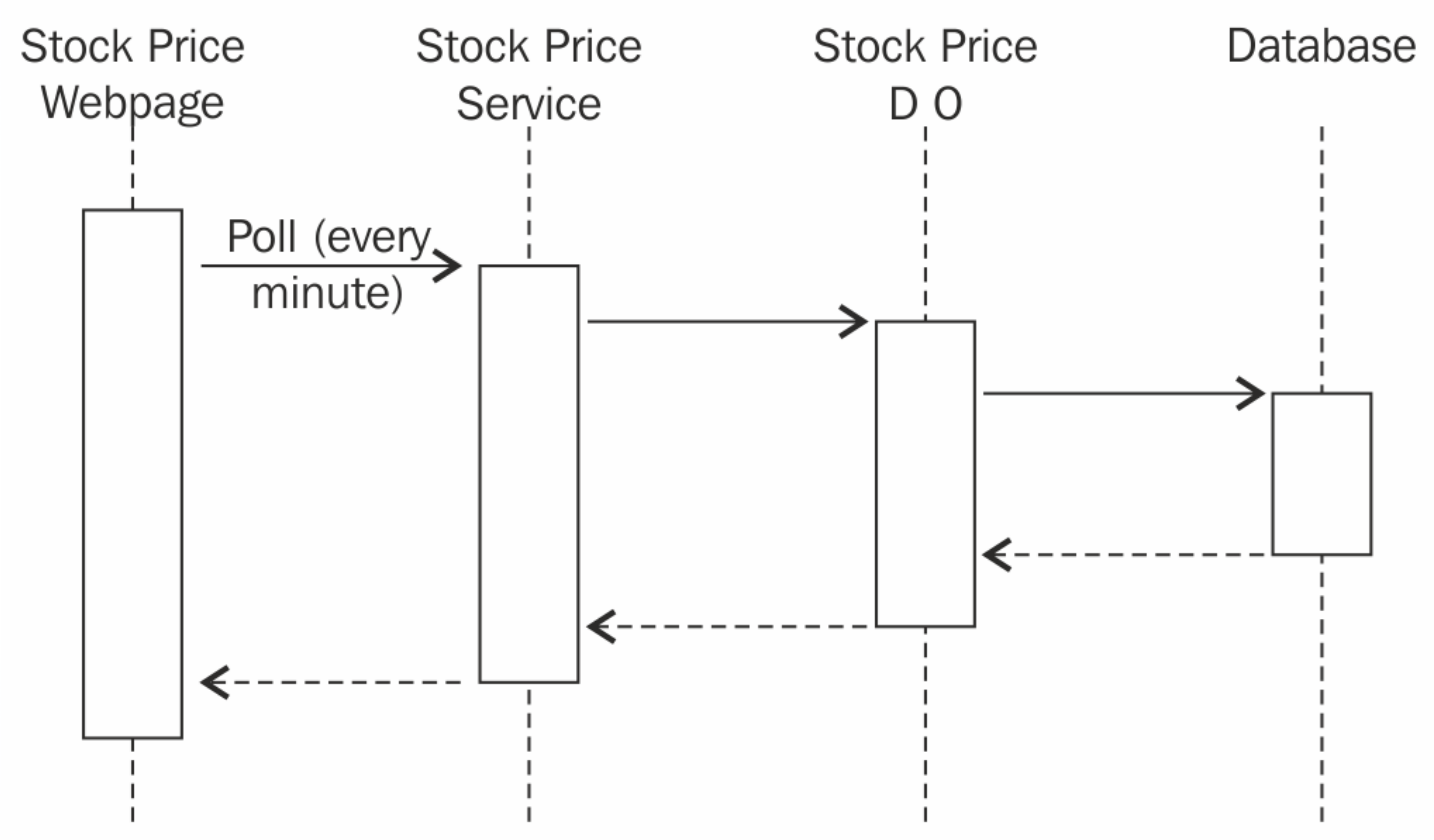
The use case we want to build is a stock price page that displays the price of a specific stock.

As long as the page remains open, we want to update the latest price of the stock on the page.

Reactive Programming

Reactive Use Case - A Stock Price Page

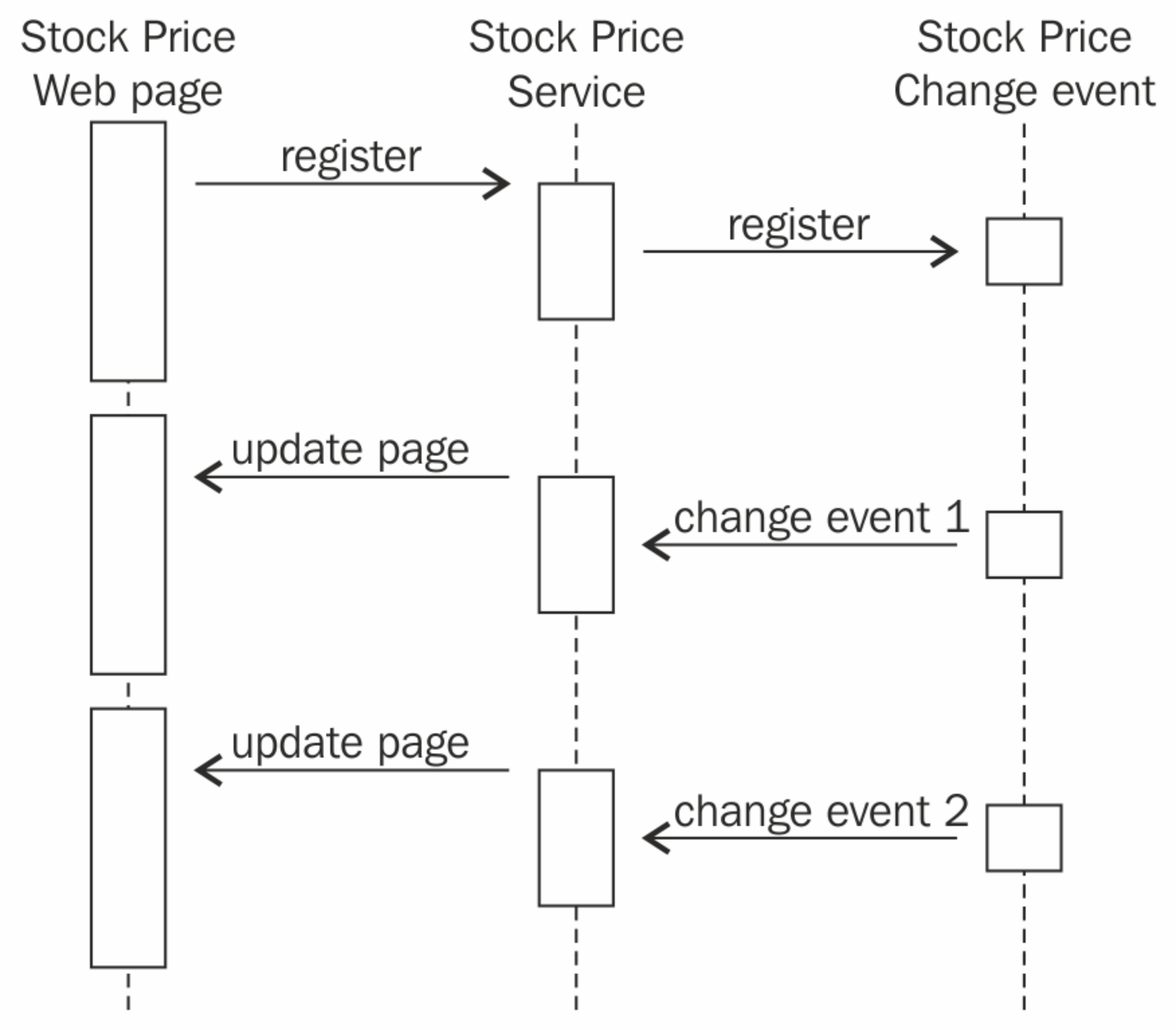
A quick look at the traditional approach



Reactive Programming

Reactive Use Case - A Stock Price Page

How is the Reactive approach different?



Reactive Programming

Reactive Use Case - A Stock Price Page

Steps for Reactive Approach

- Subscribing to Events
- The occurrence of Events
- Unregistering

How Stock Price Page works?

- **Subscribe** to Stock price change **event** on load of webpage
- The way you subscribe is different based on reactive framework
- When the stock price change **event occurs** for a stock, the event is triggered for all the subscribers
- The **listener** within web page updates the latest data
- Unregister request is sent once the browser is closed – Invokes **cancel()** method

Reactive Programming

Reactive Use Case - A Stock Price Page

Traditional Approach

- **POLL** for changes
- Polling is irrespective of change
- The lifetime of threads is longer
- All resources used by threads are locked
- High chances of resource contention
- Scaling up is inevitable

Reactive Approach

- Implements Reactive Subscribe & Event Chain
- More complex if Event Chain involves Message Broker
- The sequence is triggered only when the stock price changes
- Threads live for a short time and hence less resource contention
- Reactive infrastructure can handle more users