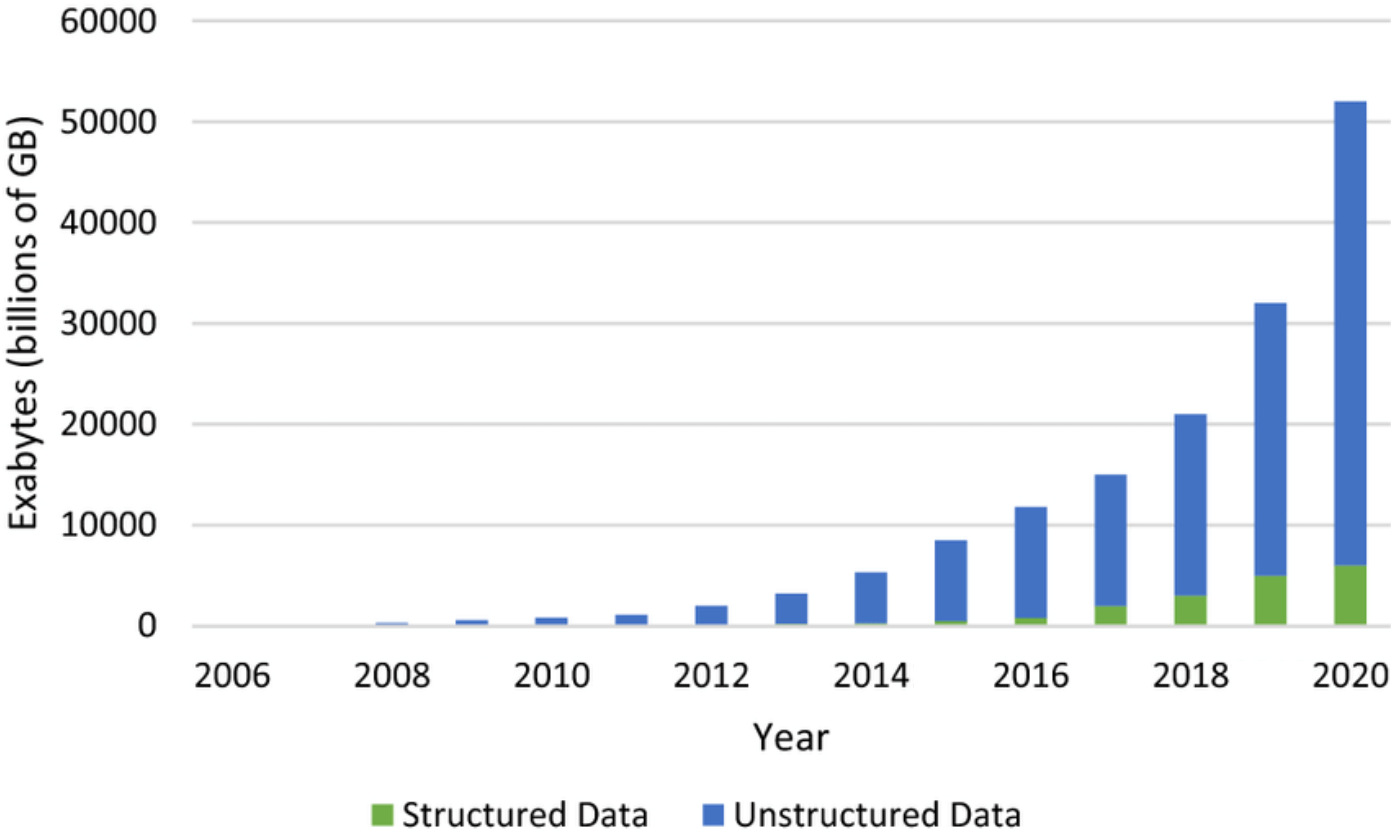
Reacting to an Event-Driven World

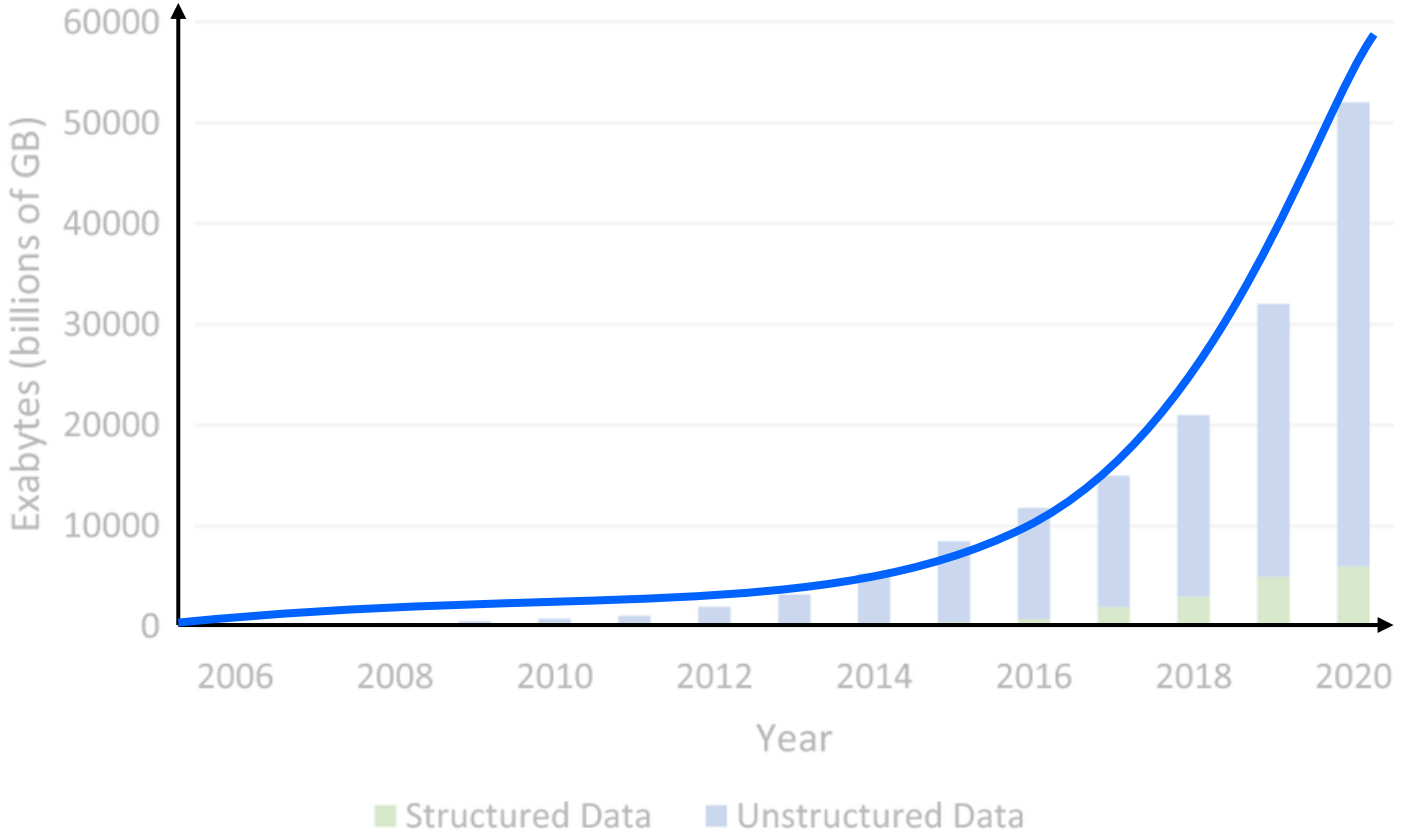
Kate Stanley and Grace Jansen

The never-ending growth of data



https://www.researchgate.net/figure/The-growth-of-structured-versus-unstructured-data-over-the-past-decade-41_fig1_335927263

The never-ending growth of data



https://www.researchgate.net/figure/The-growth-of-structured-versus-unstructured-data-over-the-past-decade-41_fig1_335927263

Customer demand for responsive, real-time applications

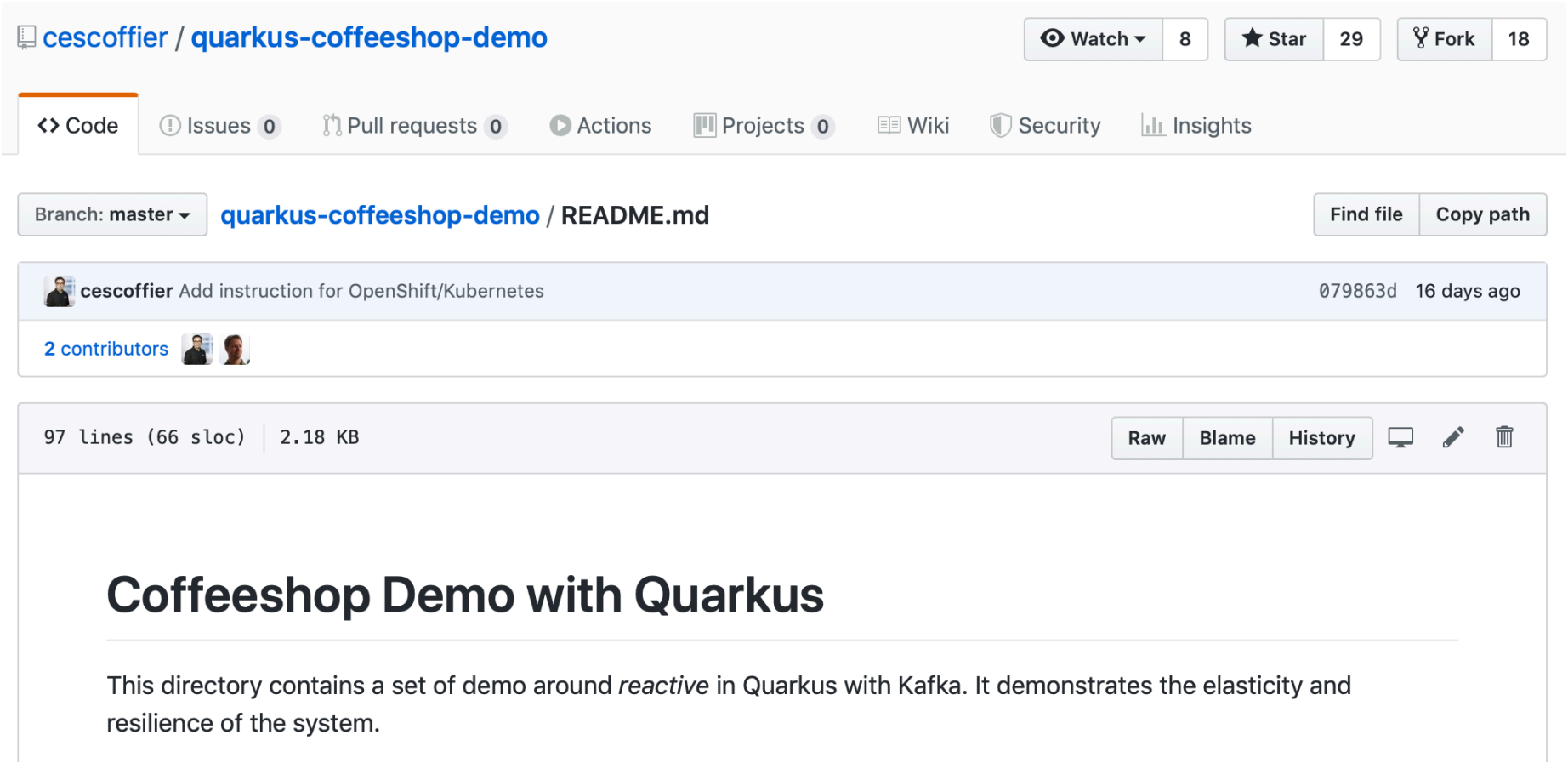
Apps must **react** to events in real time, as they happen

Apps must deliver **responsive** customer experiences

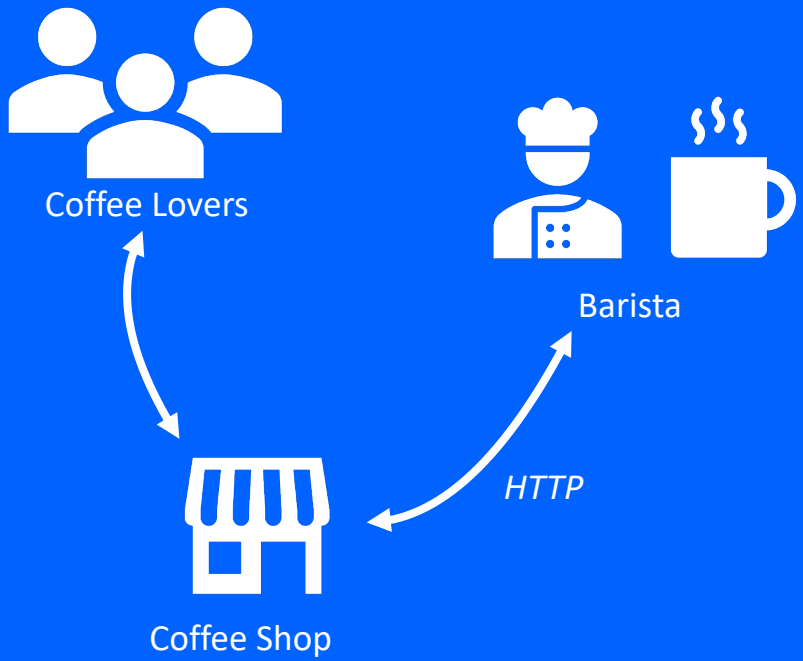
Apps enhanced with real-time **intelligence**

Now for an example, let’s get some coffee!

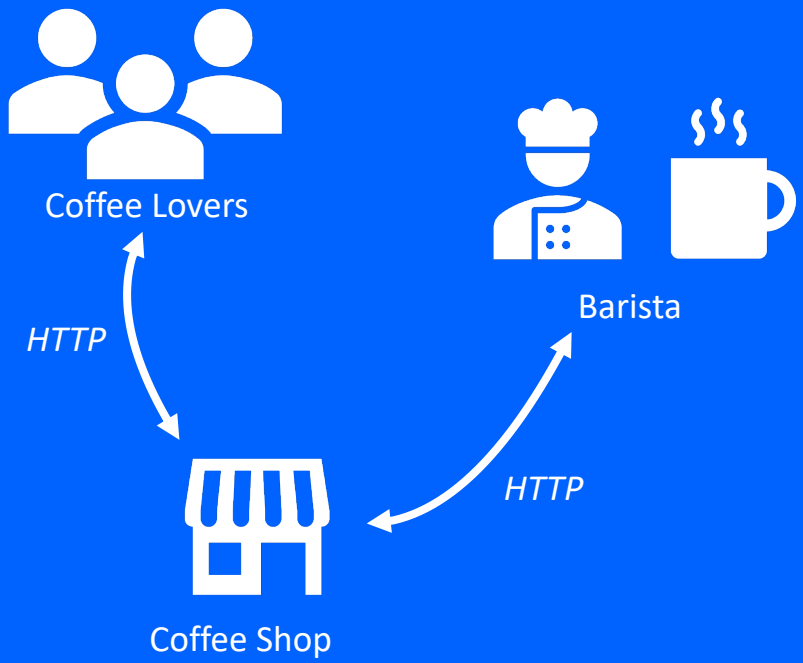
https://github.com/cescoffier/quarkus-coffeeshop-demo

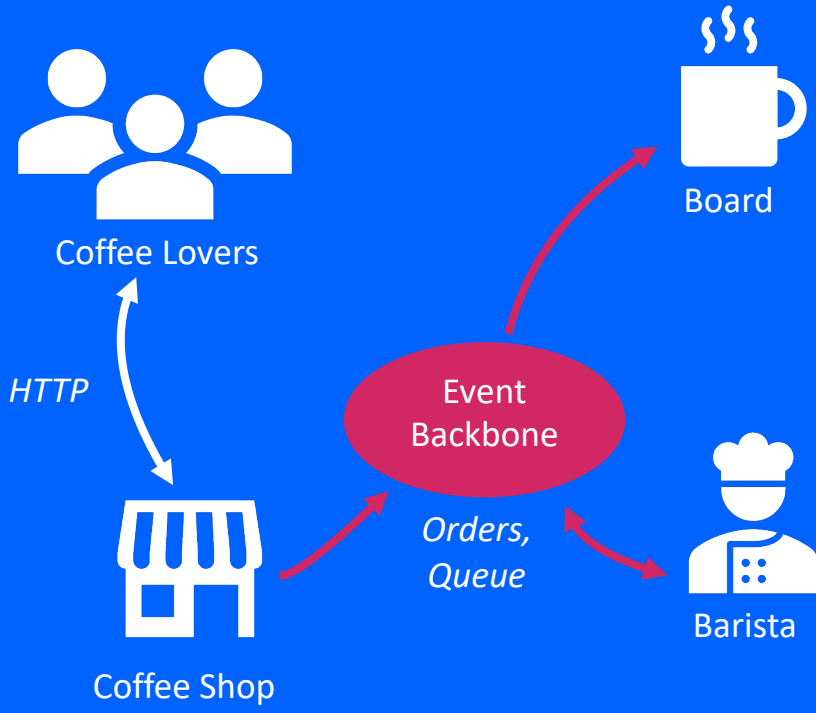


Barista Example:



Barista Example:

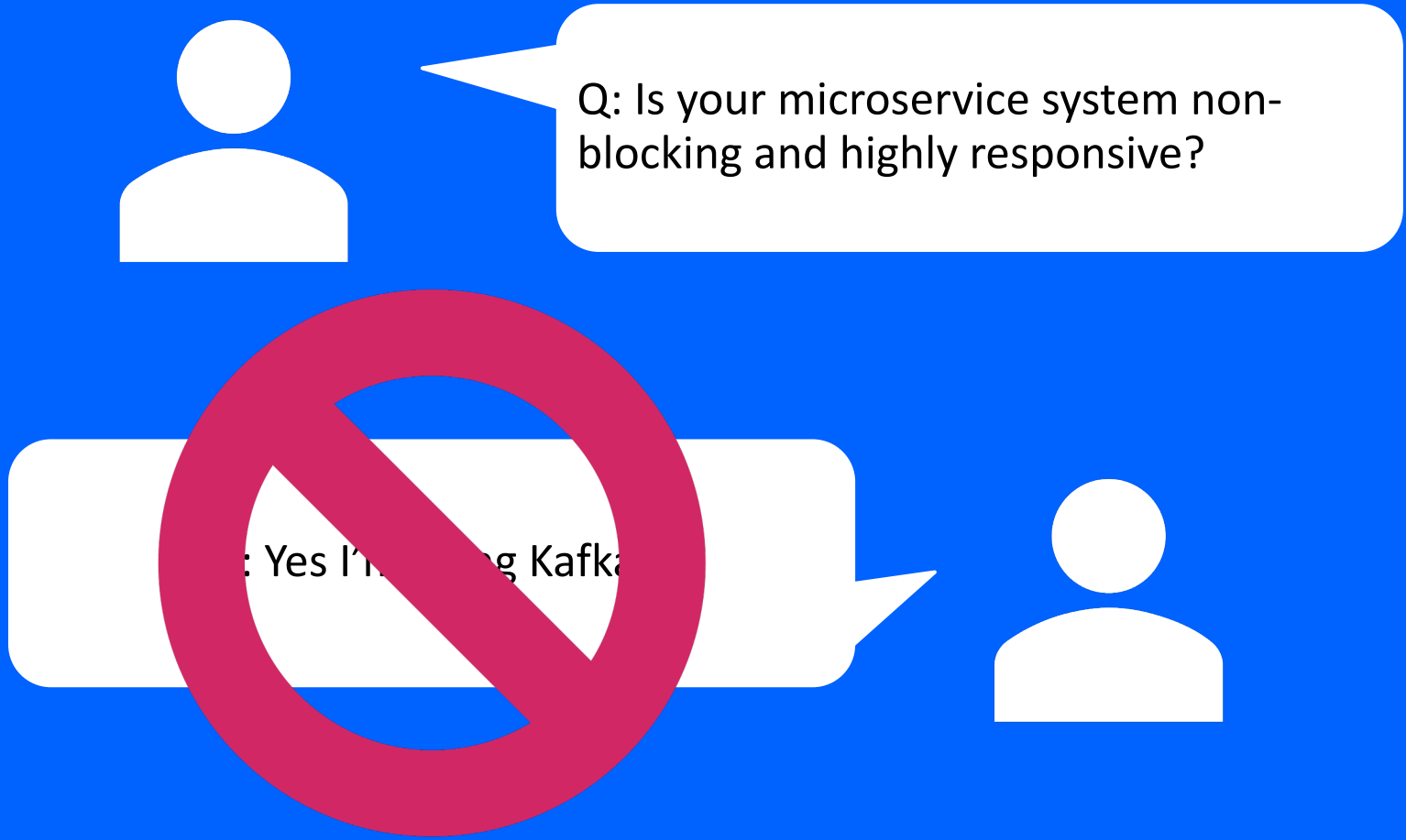




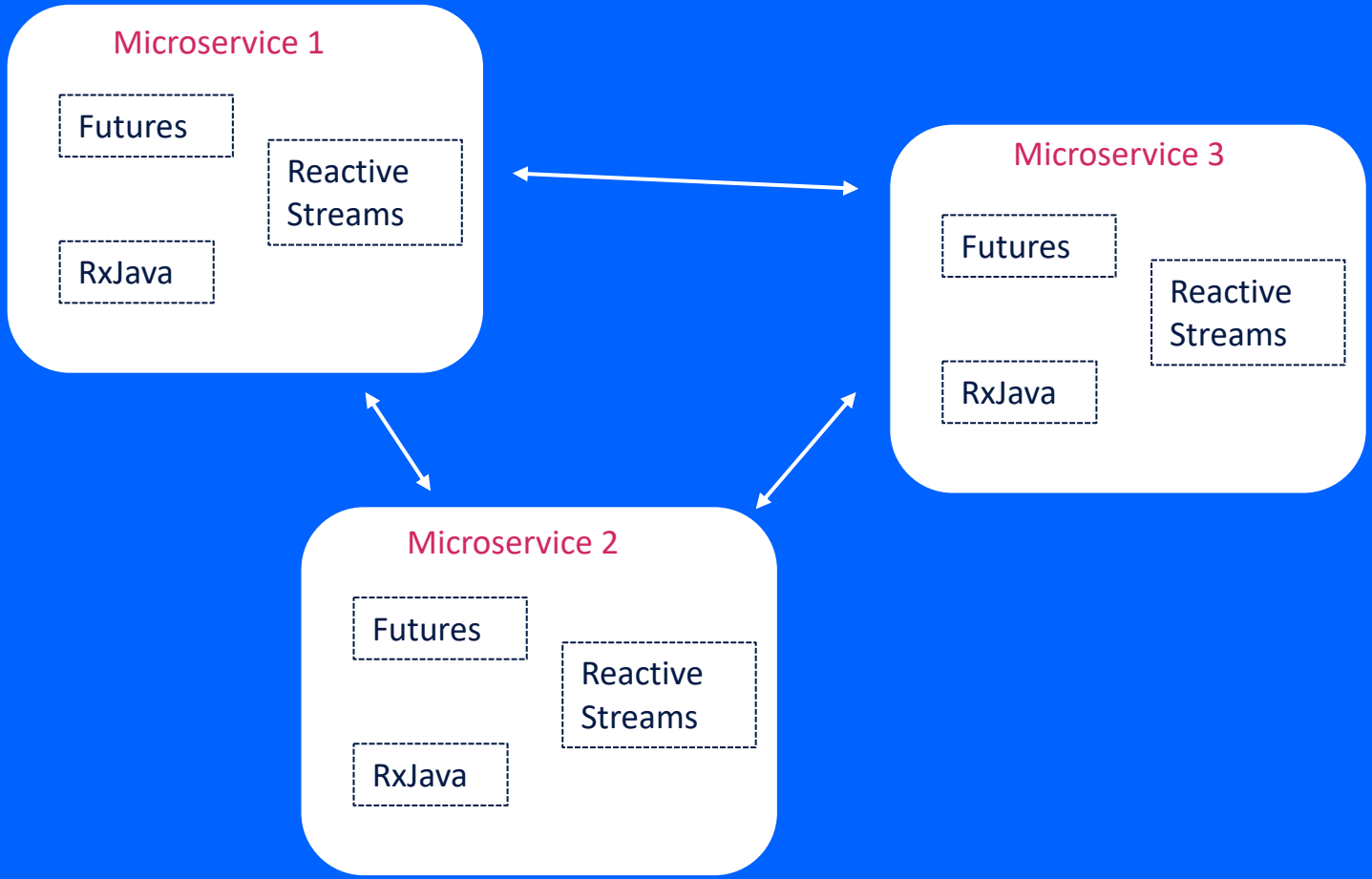
Q: Is your microservice system non-blocking and highly responsive?

Q: Is your microservice system non-blocking and highly responsive?

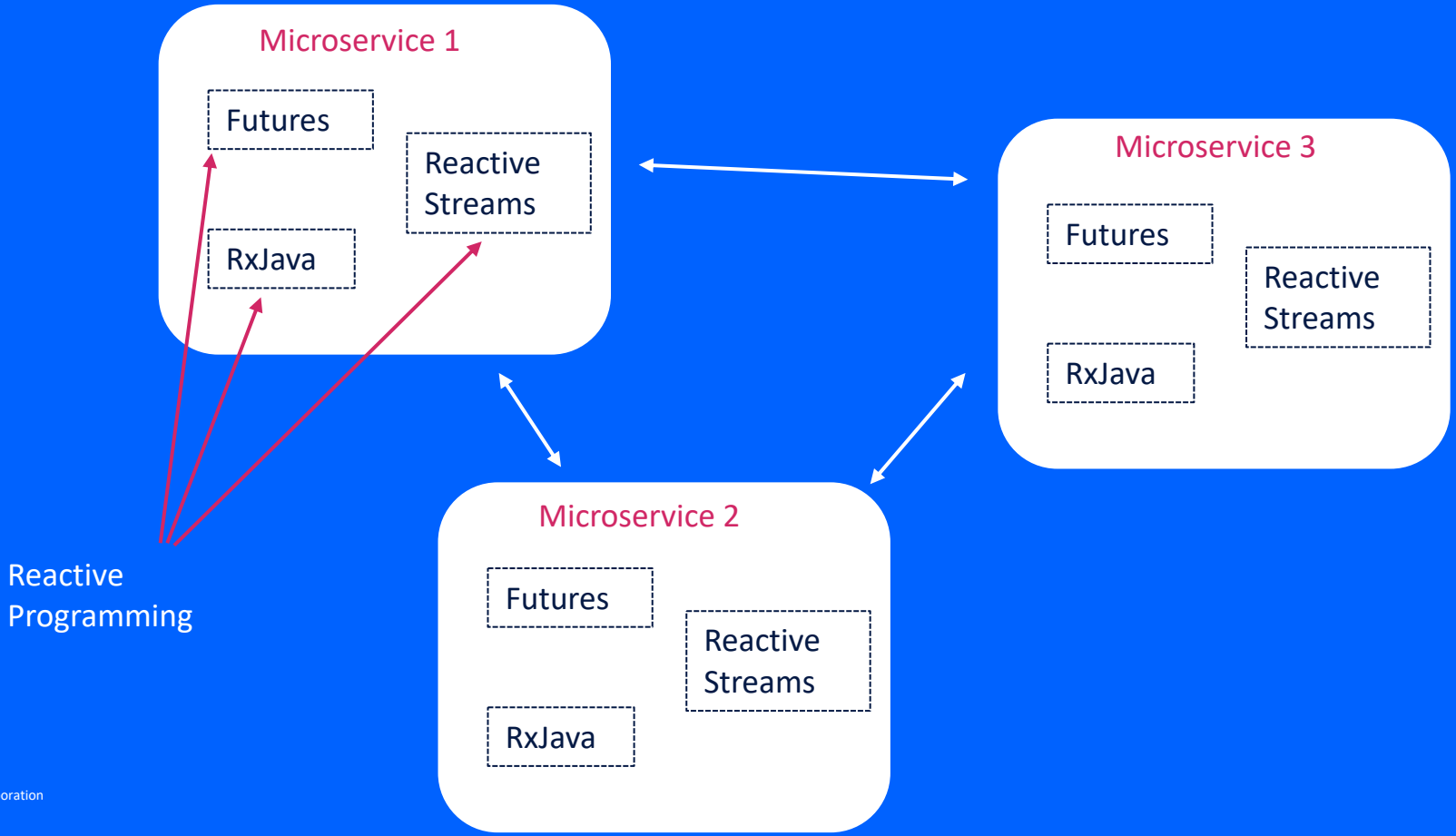
A: Yes I’m using Kafka!



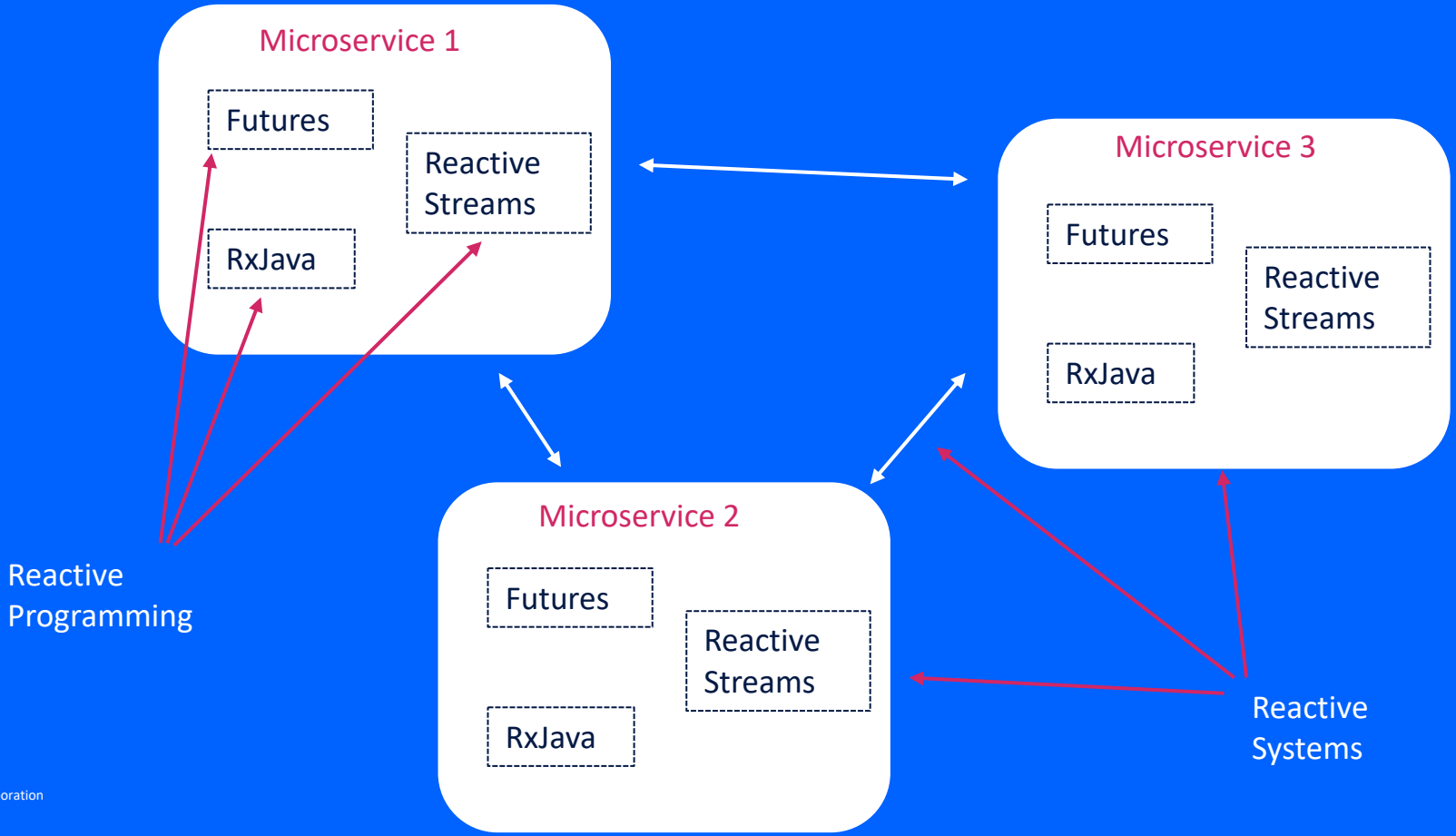
How do we make a highly responsive app?



How do we make a highly responsive app?



How do we make a highly responsive app?



Reactive Programming

A subset of **asynchronous programming** and a paradigm where the **availability of new information drives the logic forward** rather than having control flow driven by a thread-of-execution.

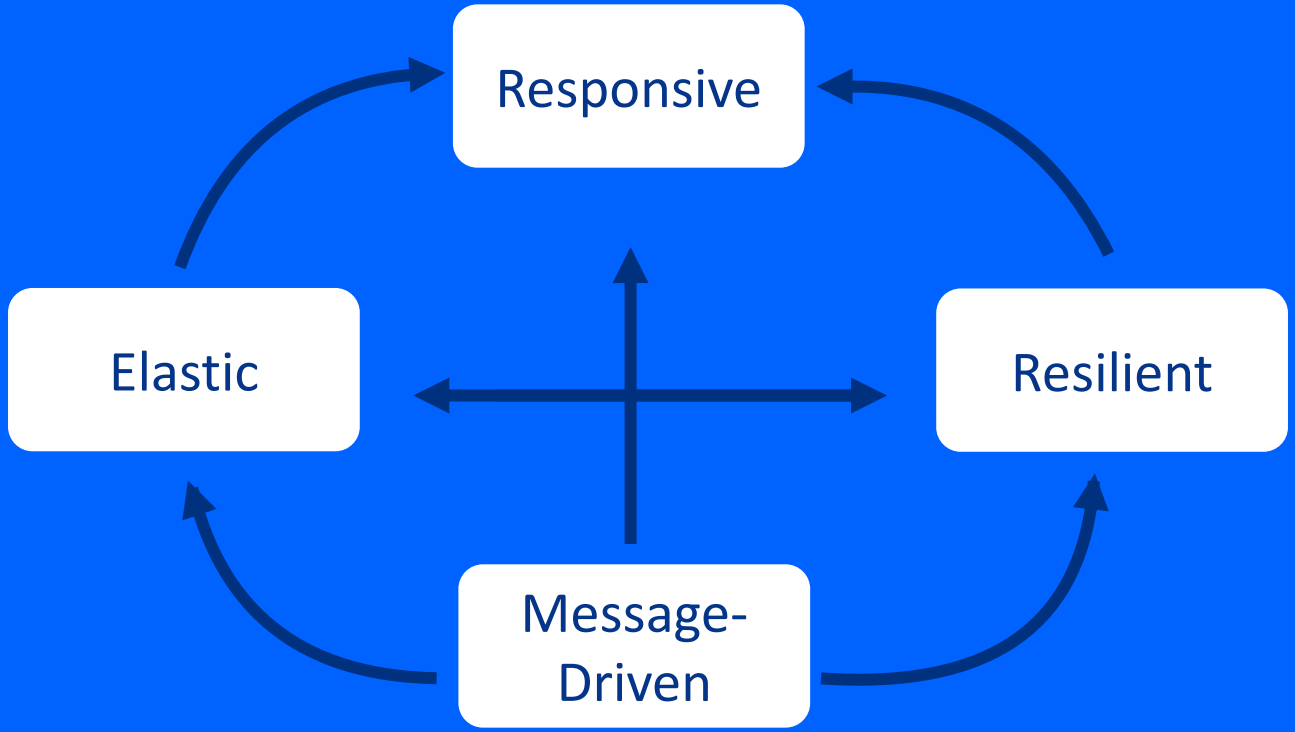
Reactive Programming Patterns

Futures: a **promise** to hold the result of some operation once that operation completes

RxJava: a Java VM implementation of **ReactiveX** (Reactive Extensions): a library for composing asynchronous and event-based programs by using observable sequences.

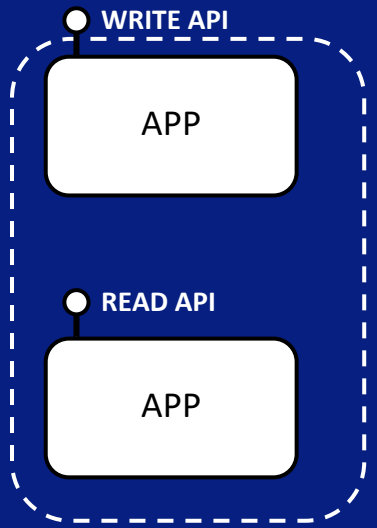
Reactive Streams: a programming concept for **handling asynchronous data streams** in a non-blocking manner while providing backpressure to stream publishers

Reactive Manifesto

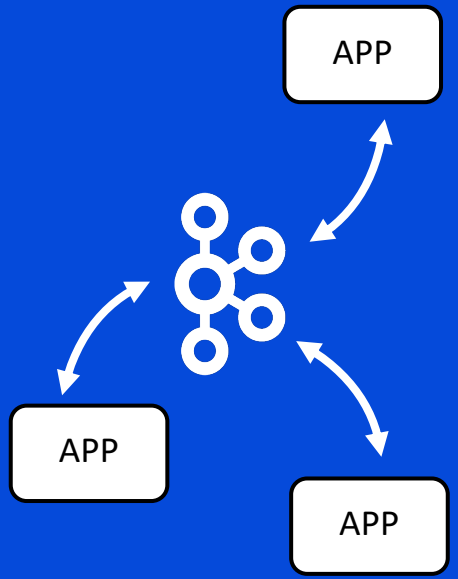


Reactive Architecture design patterns

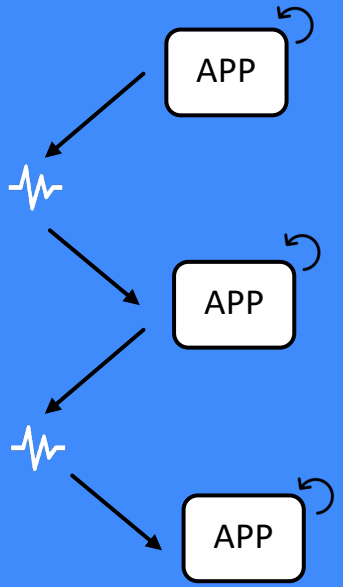
CQRS



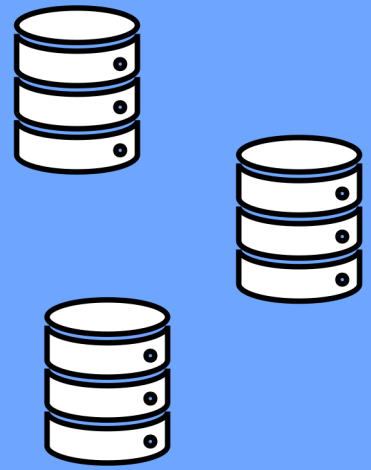
Event Sourcing



Saga



Sharding



How do I configure Kafka for a reactive system?

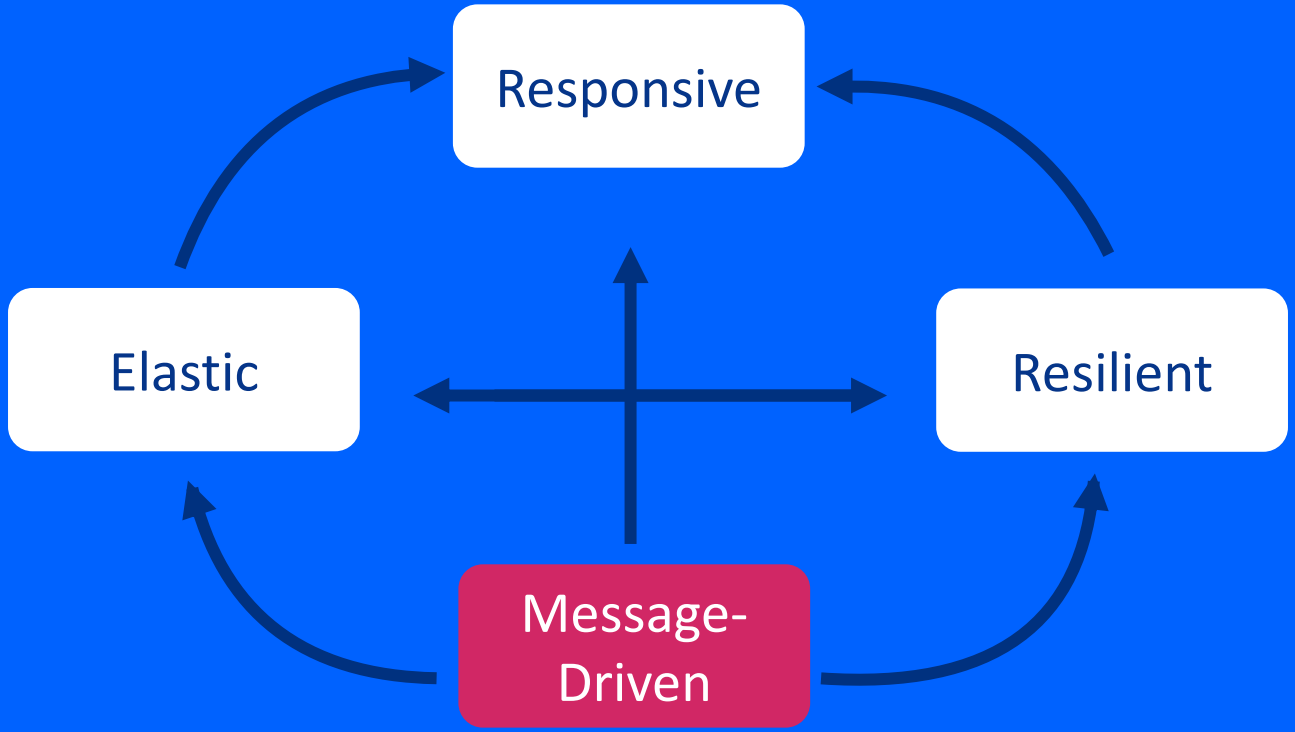
Apache Kafka is an **open source, distributed streaming platform**

Publish and subscribe to streams of records

Store records in durable way

Process streams of records as they occur

Event-Driven vs Message-Driven



Reactive Systems rely on asynchronous message-passing

Messages

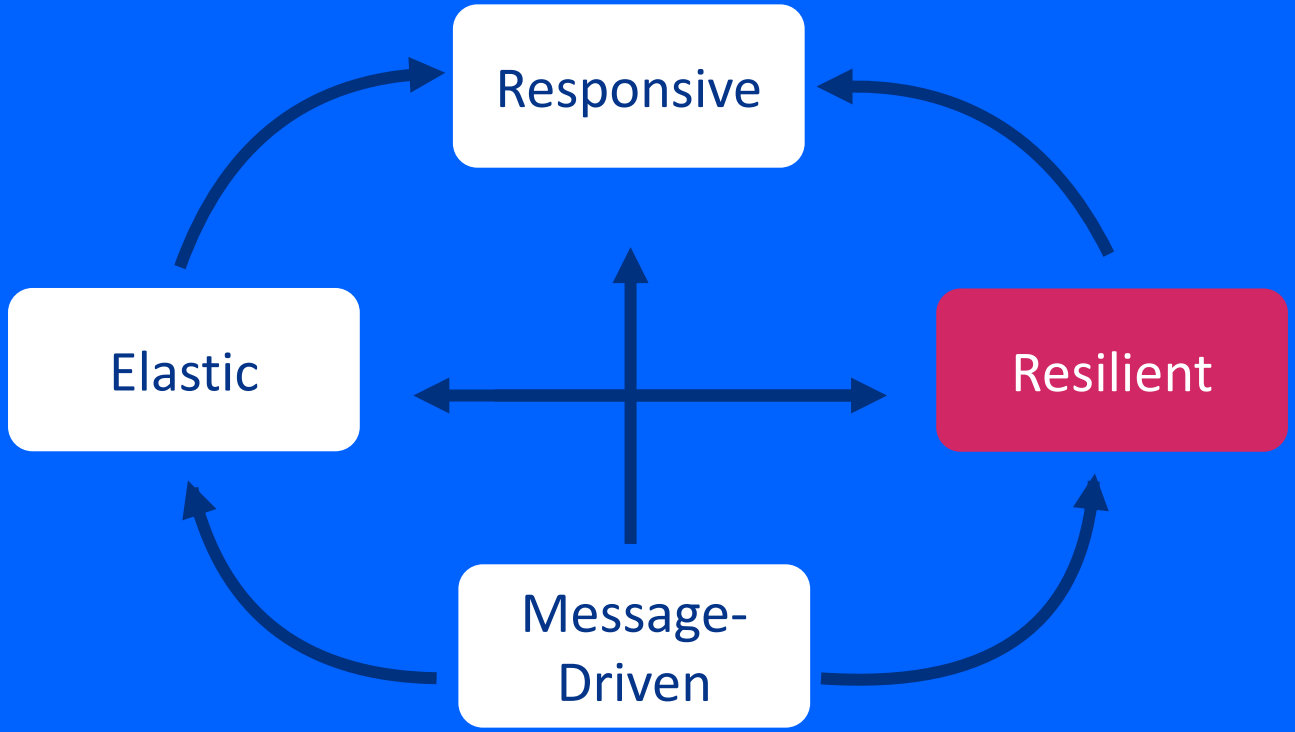
“An item of data sent to a specific location.”

A message can contain an encoded event in its payload.

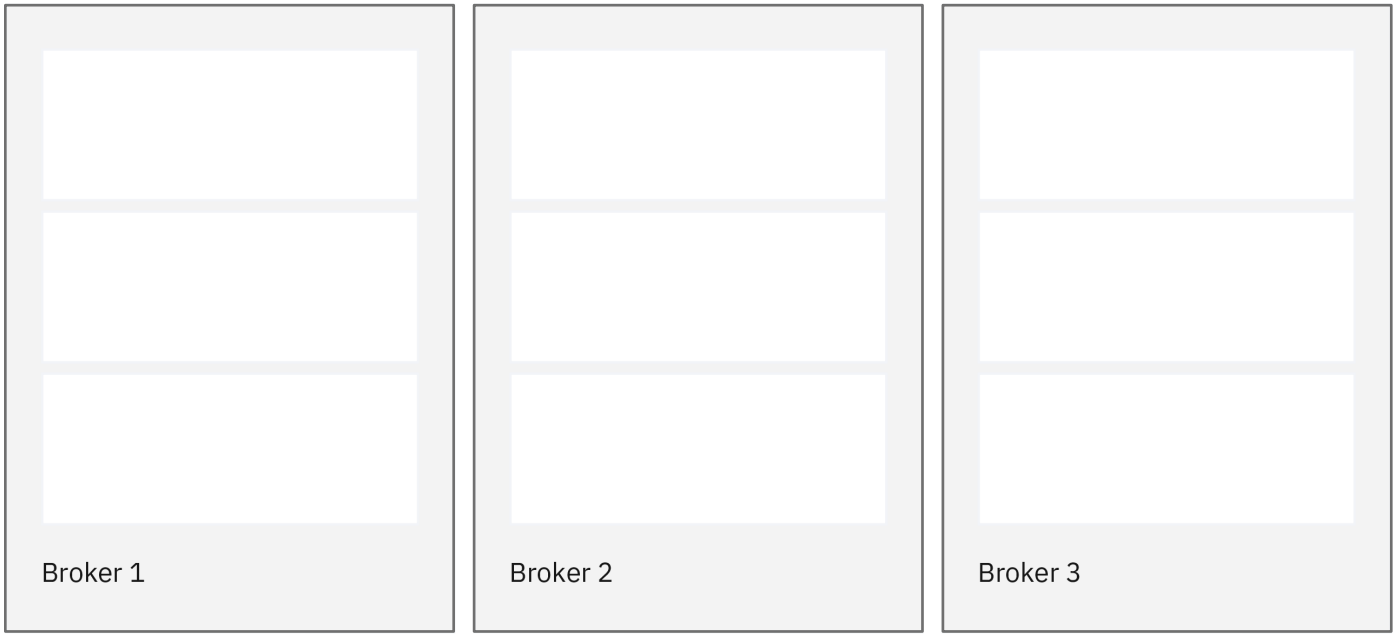
Events

“A signal emitted by a component upon reaching a given state.”

Resiliency in Kafka

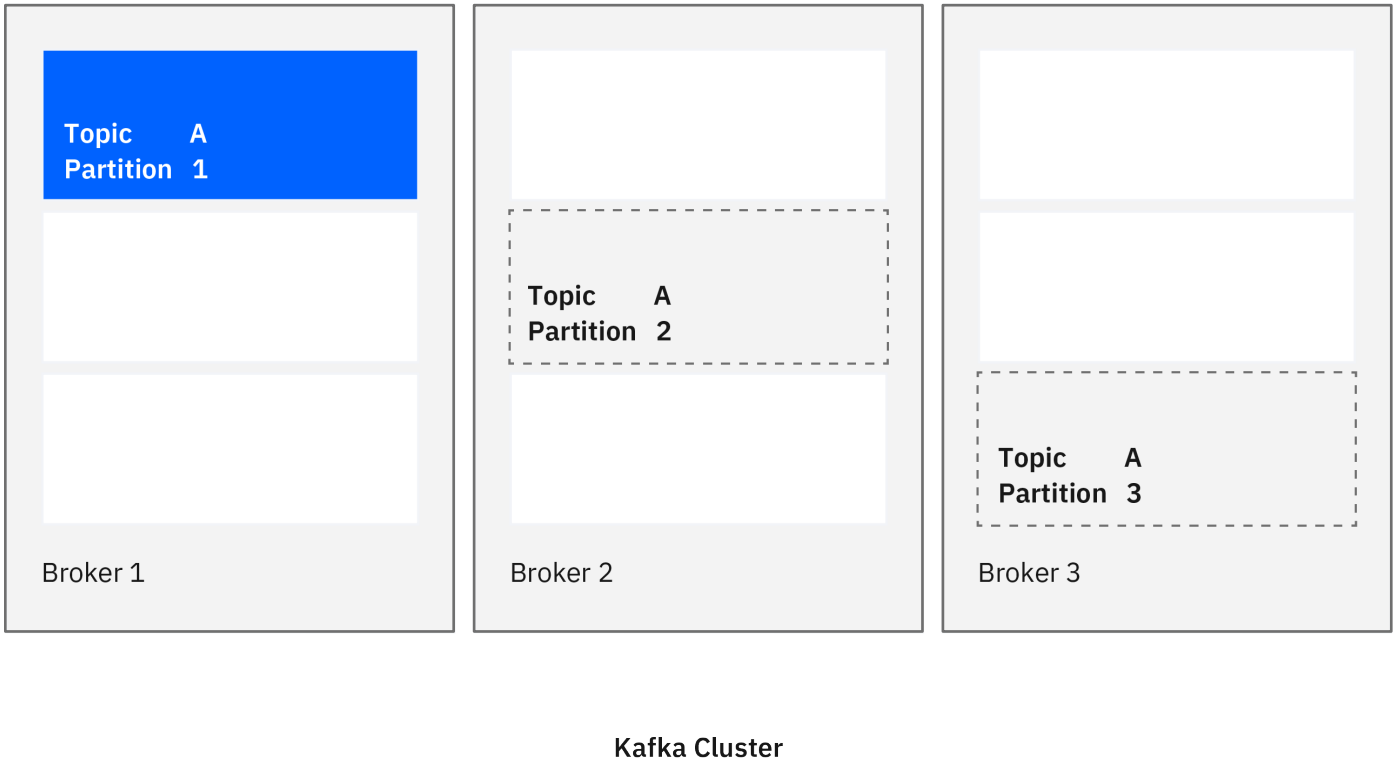


Resilient

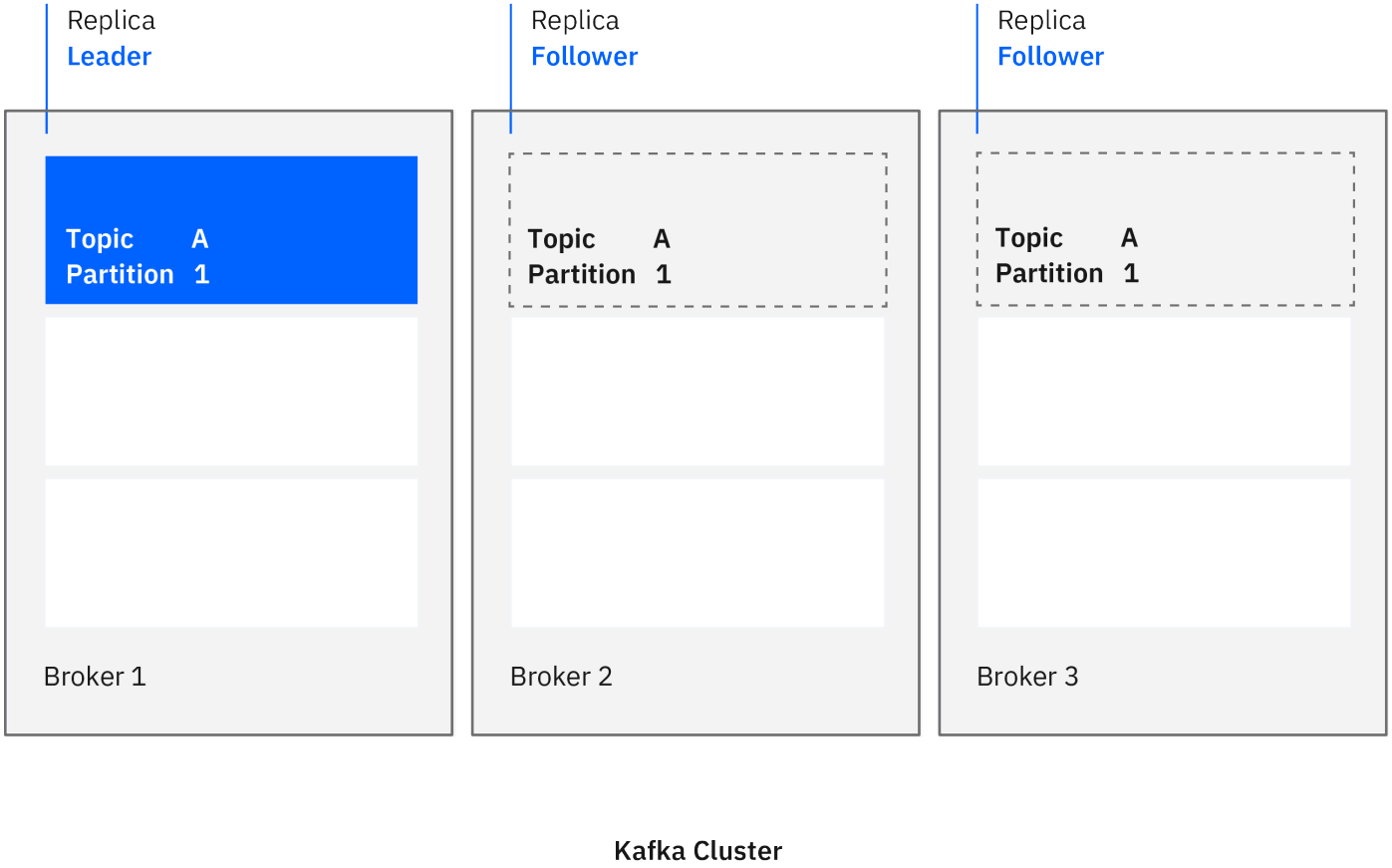


Kafka Cluster

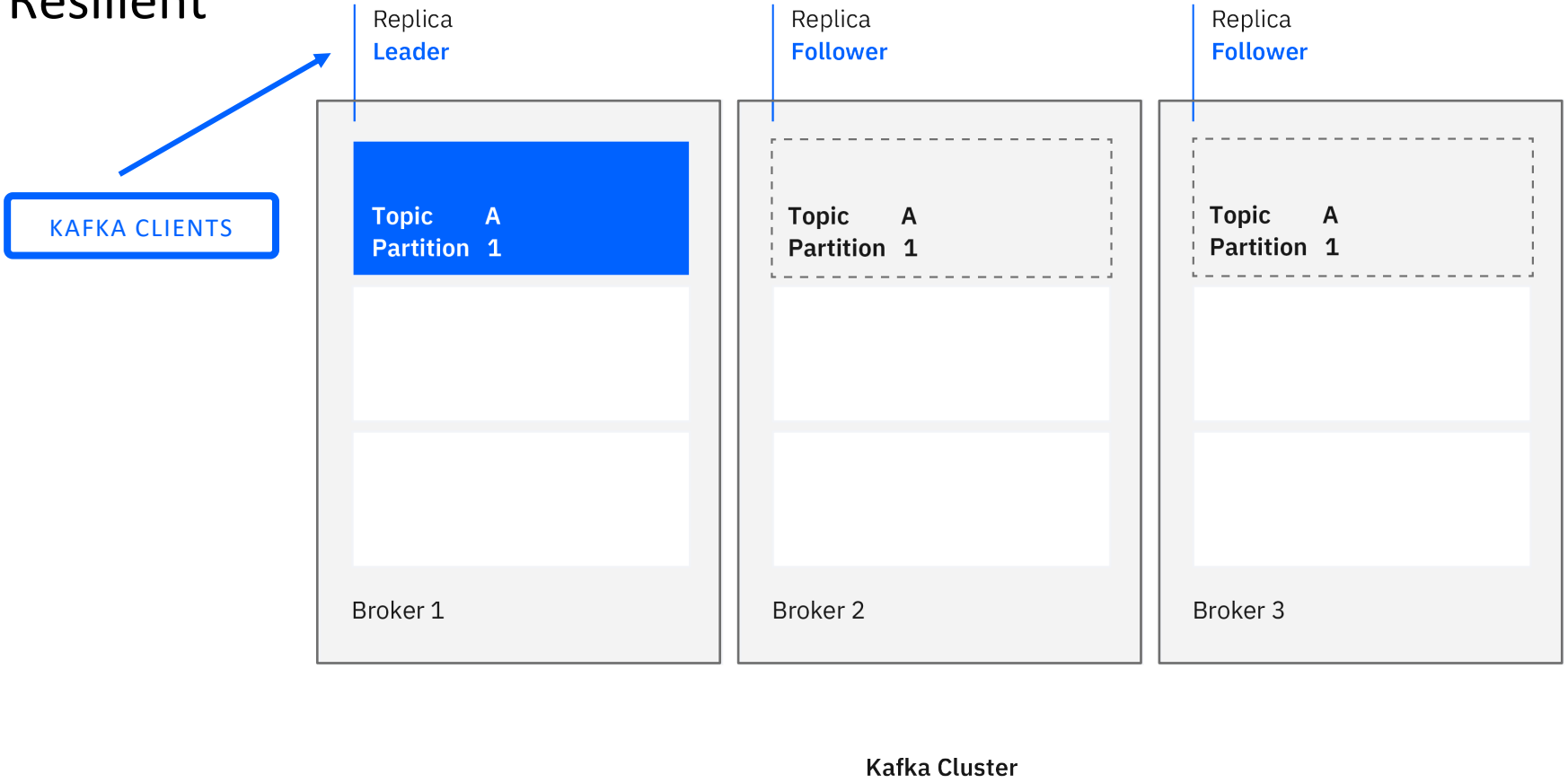
Resilient



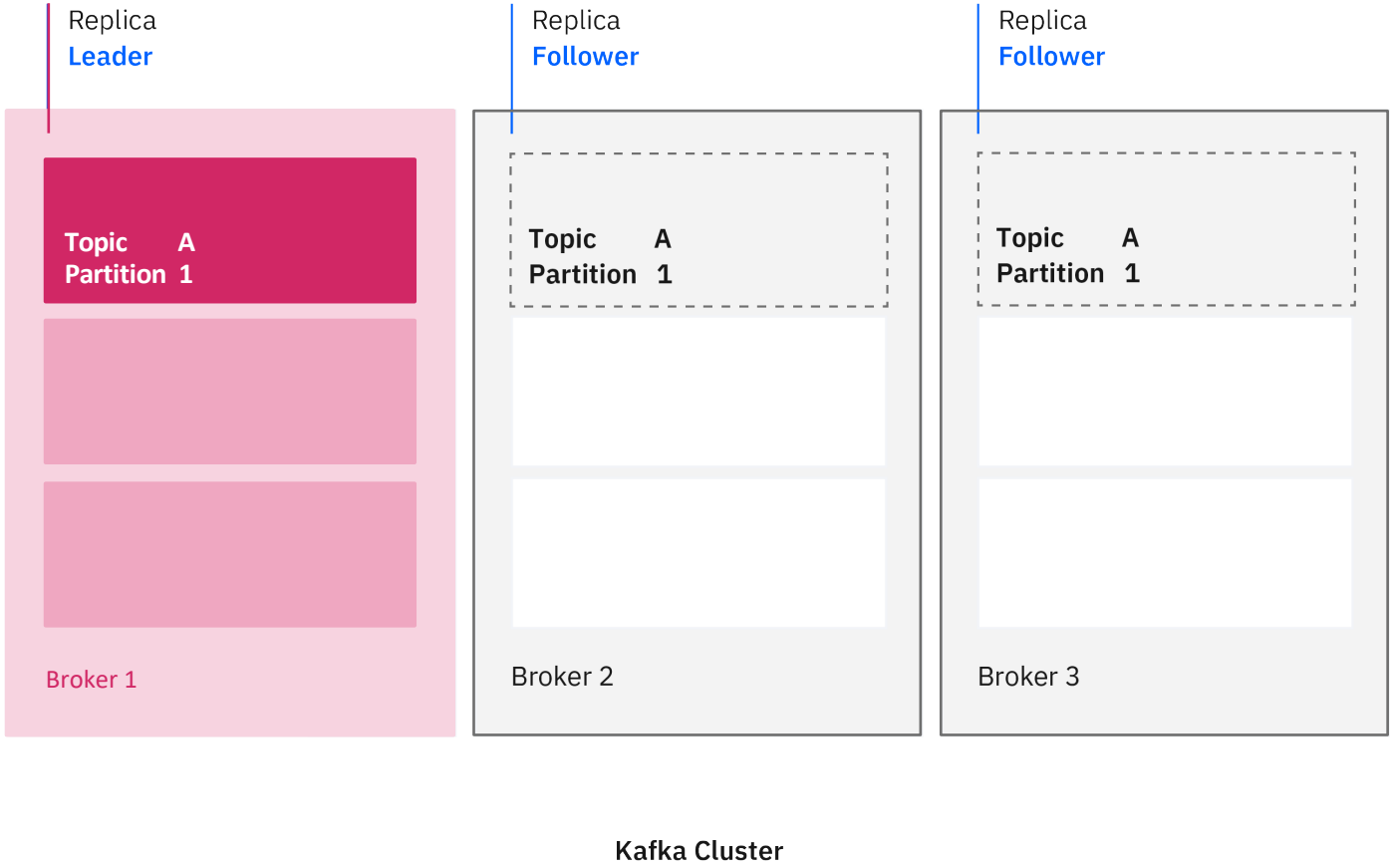
Resilient



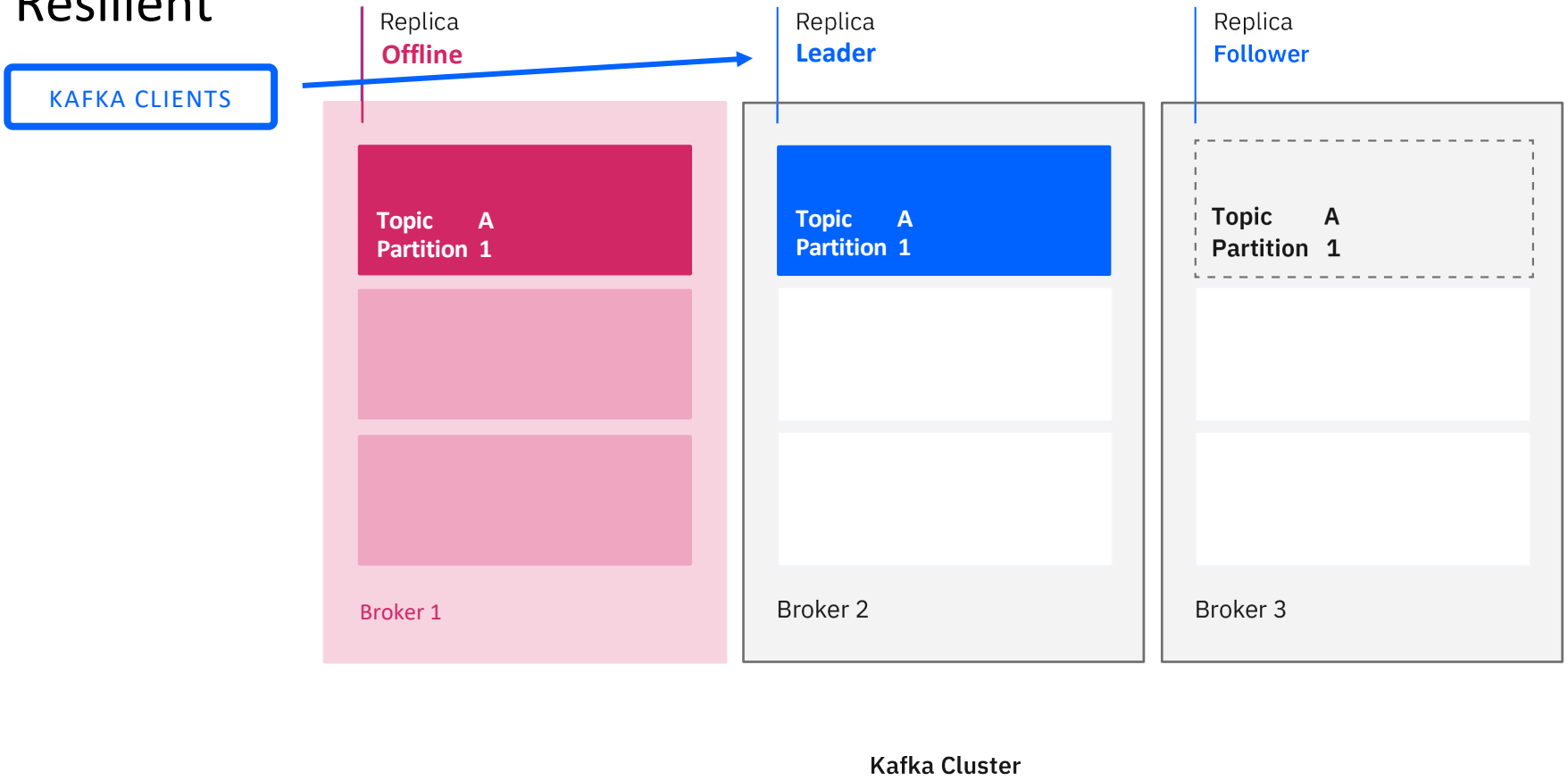
Resilient



Resilient



Resilient



Resilient producers

The producer doing a

“fire-and-forget”

isn’t good enough

Guarantees in Kafka:

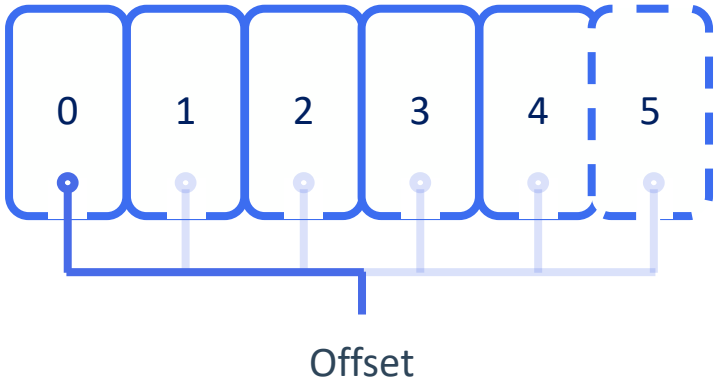
- At most once
- At least once

Configuration in producers

- Acks
- Retries

Resilient consumers

If a consumer dies it needs to know where to pick up from again



Consumer offsets

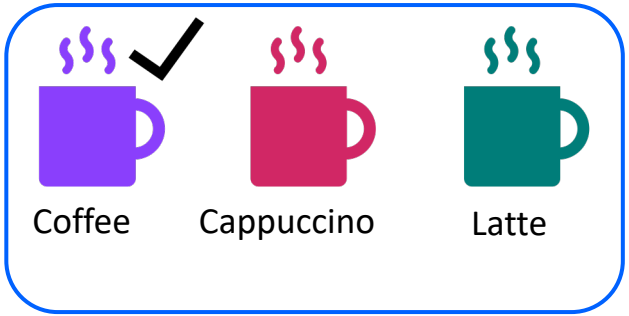
Commit offset:

Manually or automatically

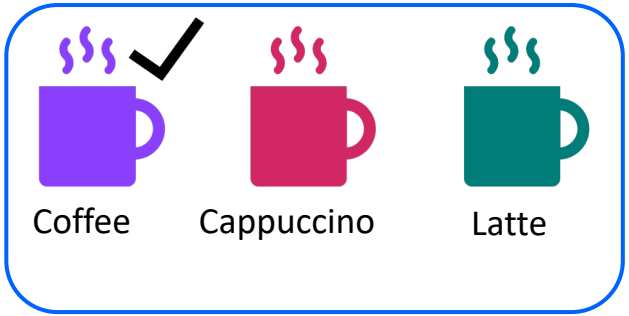
Baristas with auto commit

Baristas with auto commit

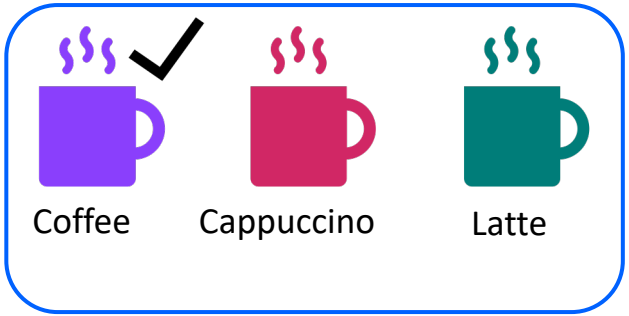
Baristas with auto commit



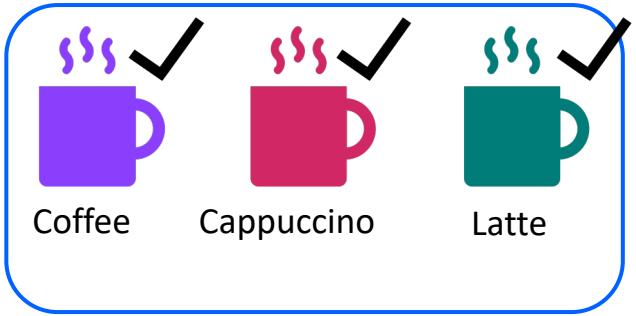
Baristas with auto commit



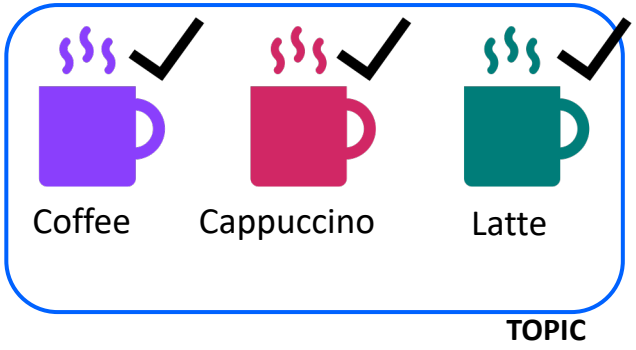
Baristas with auto commit



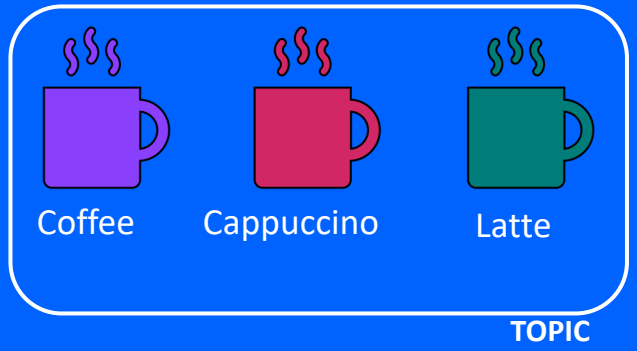
Baristas with auto commit



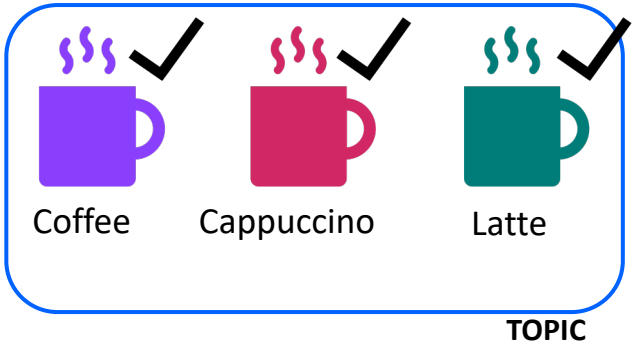
Baristas with auto commit



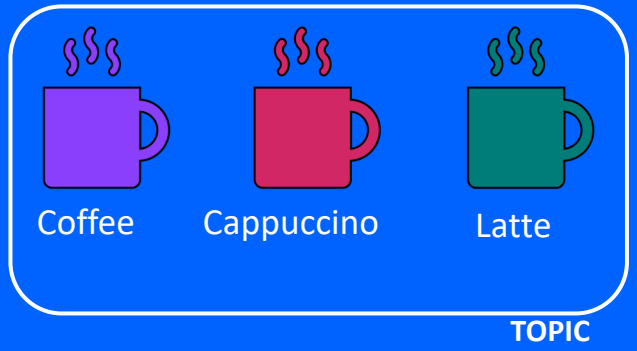
Baristas with manual commit



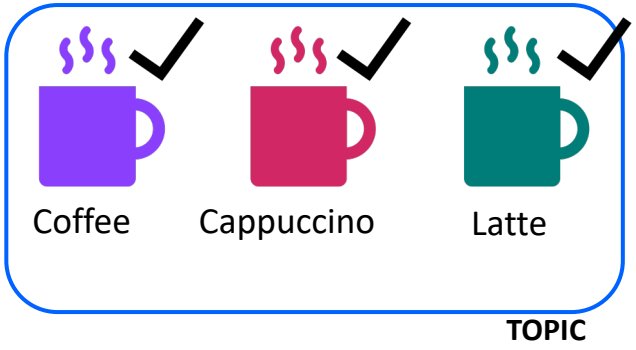
Baristas with auto commit



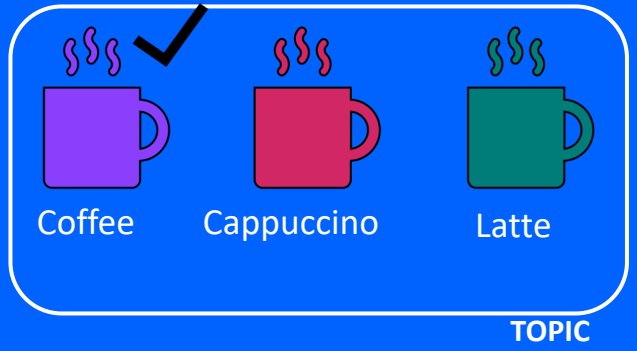
Baristas with manual commit



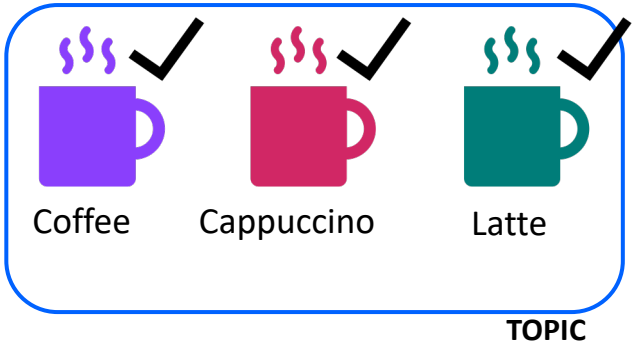
Baristas with auto commit



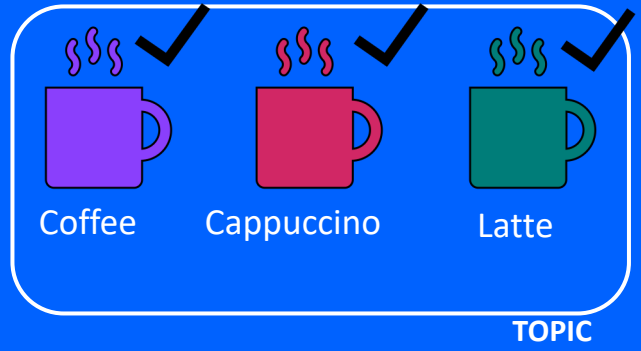
Baristas with manual commit



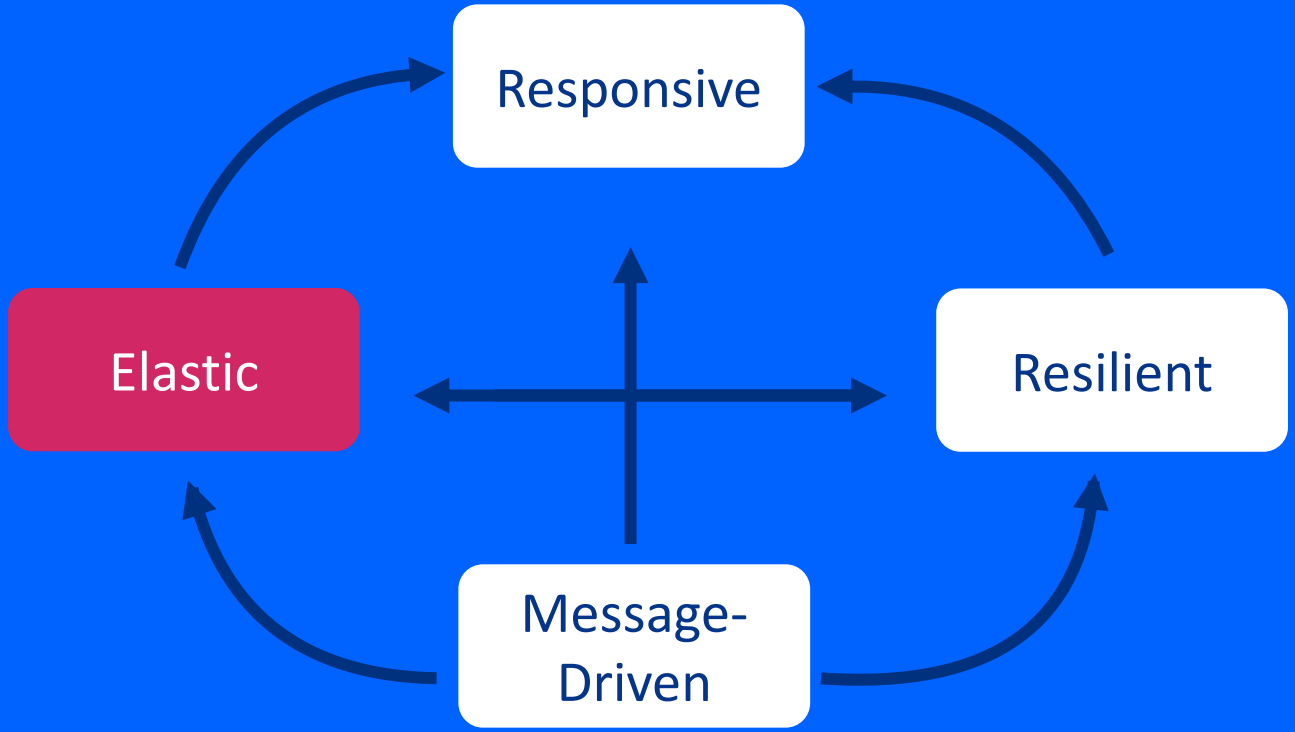
Baristas with auto commit



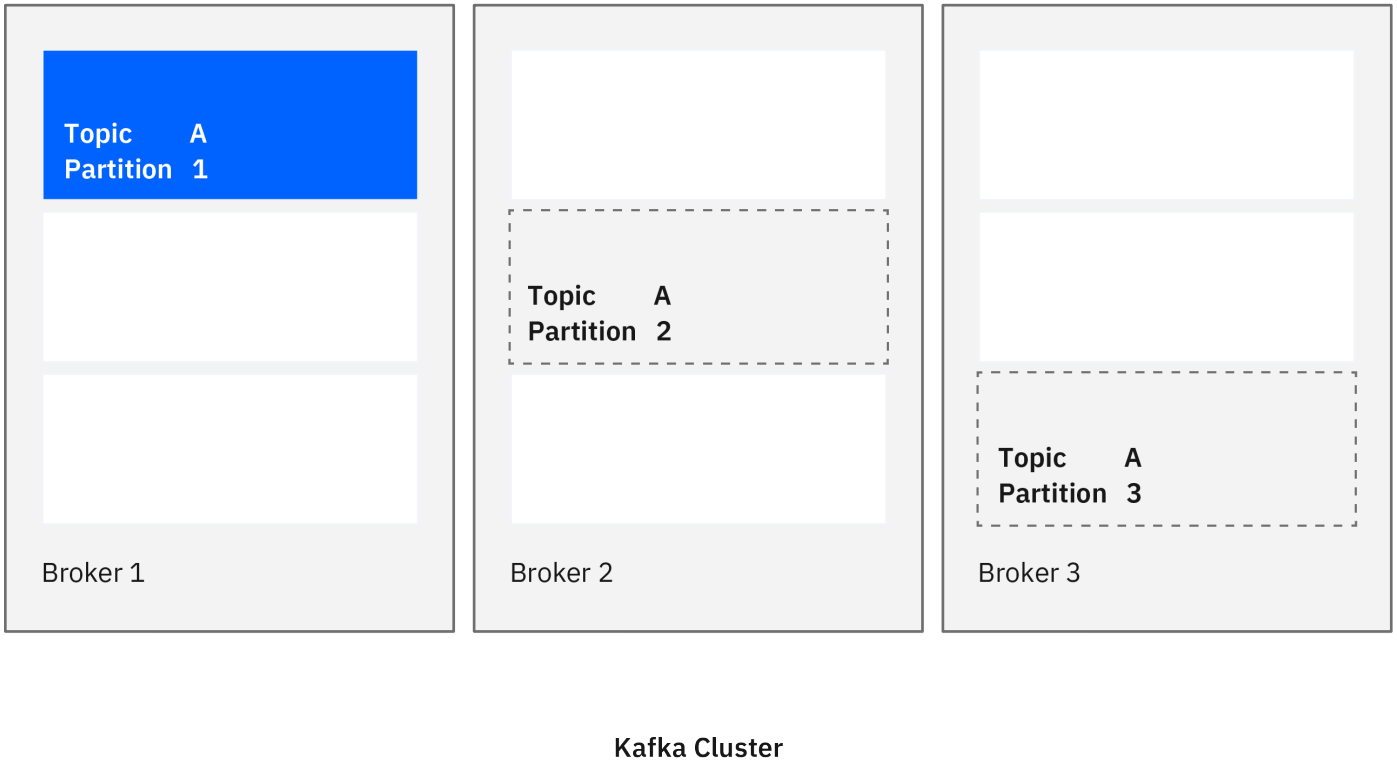
Baristas with manual commit



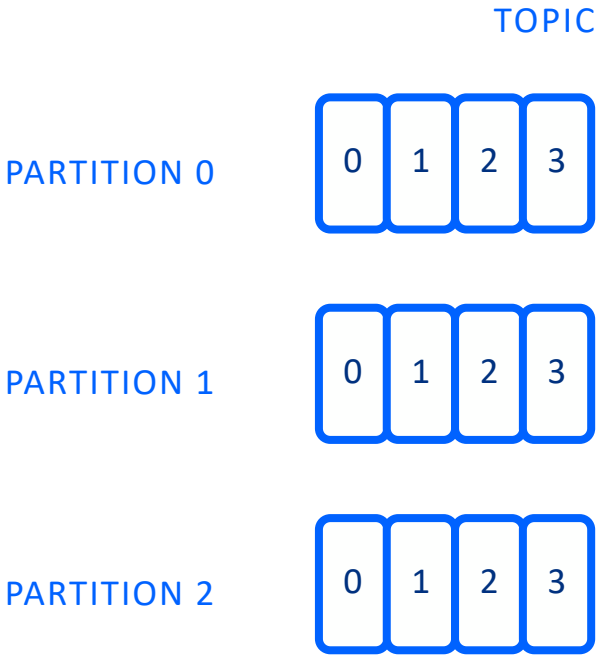
Scalability in Kafka



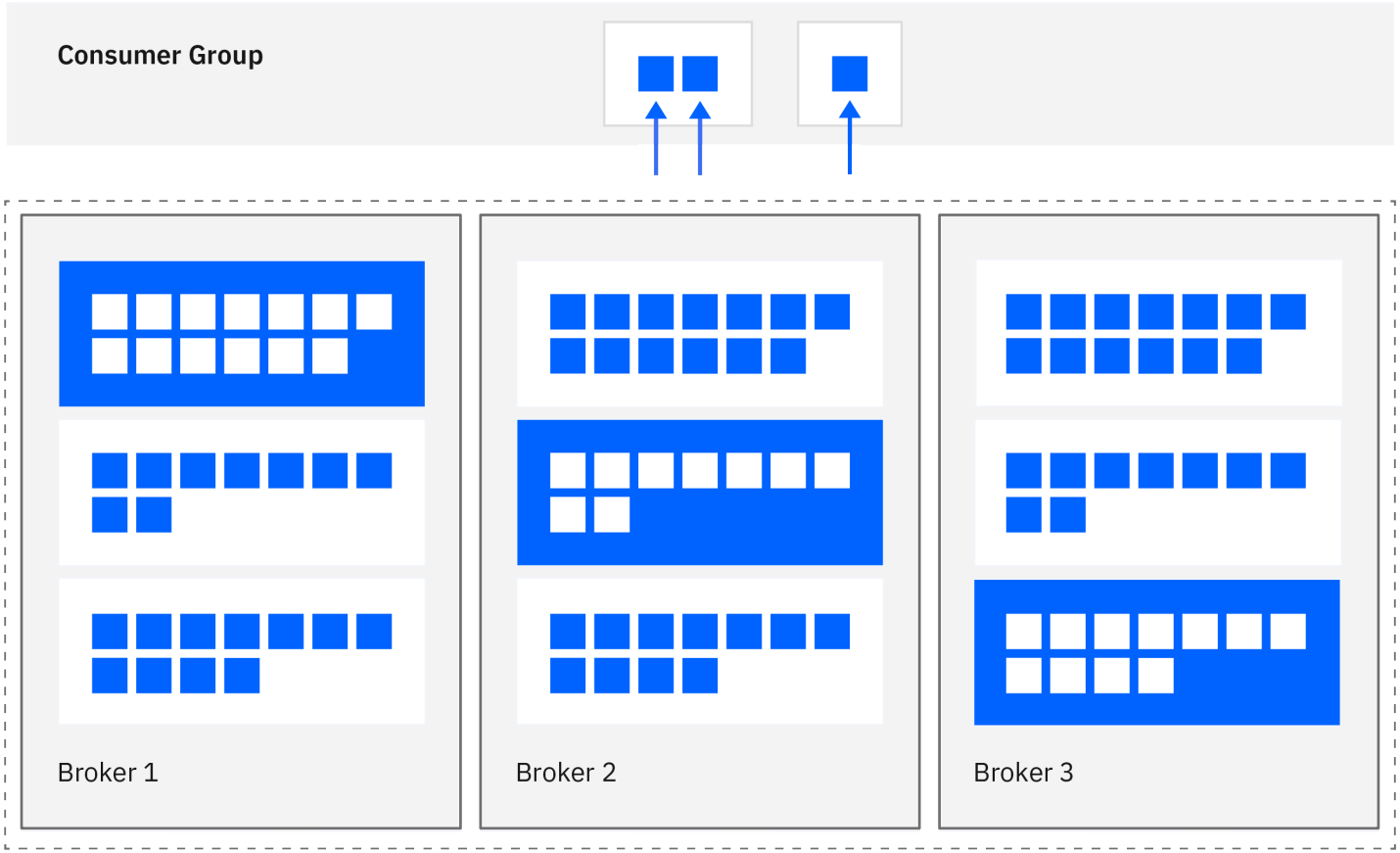
Partitions



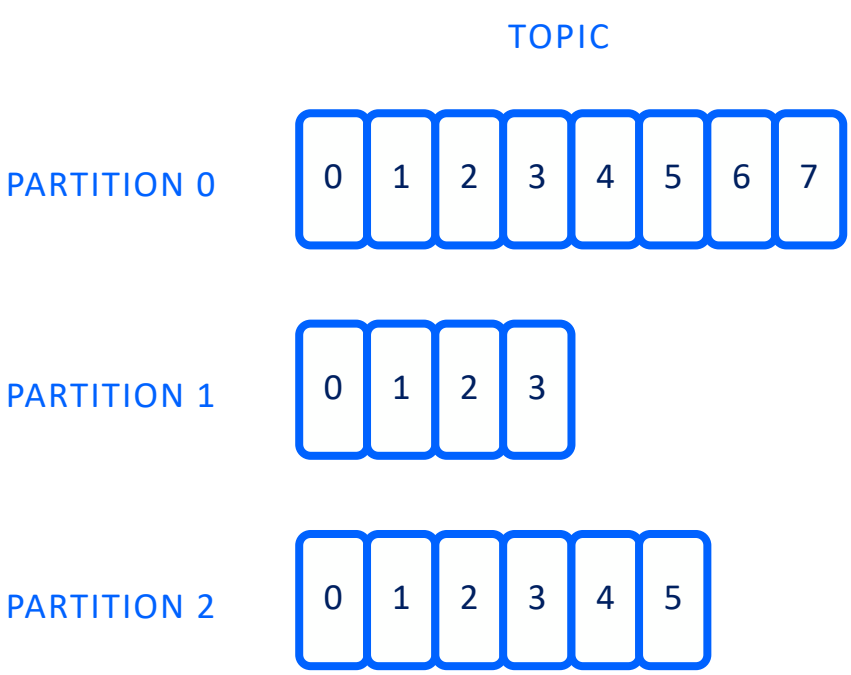
Topics and Keys

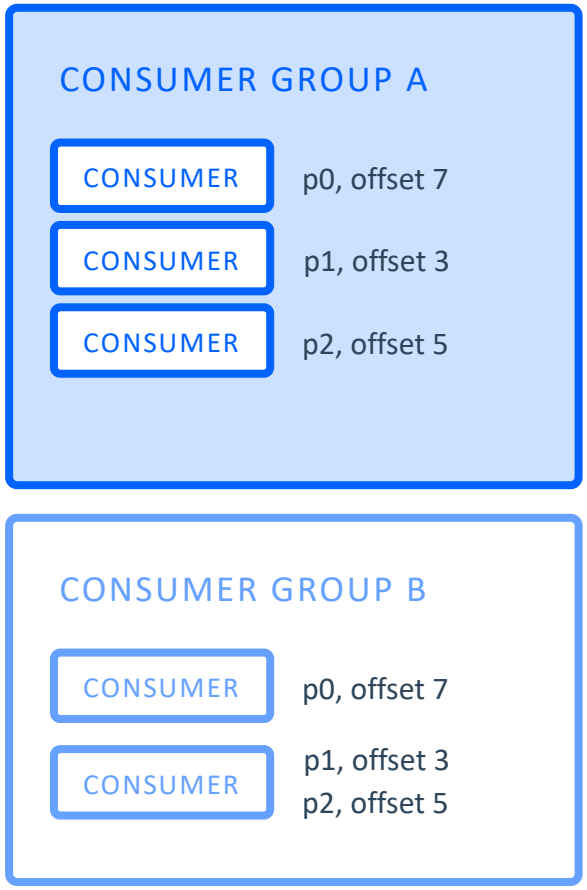


Consumer Groups

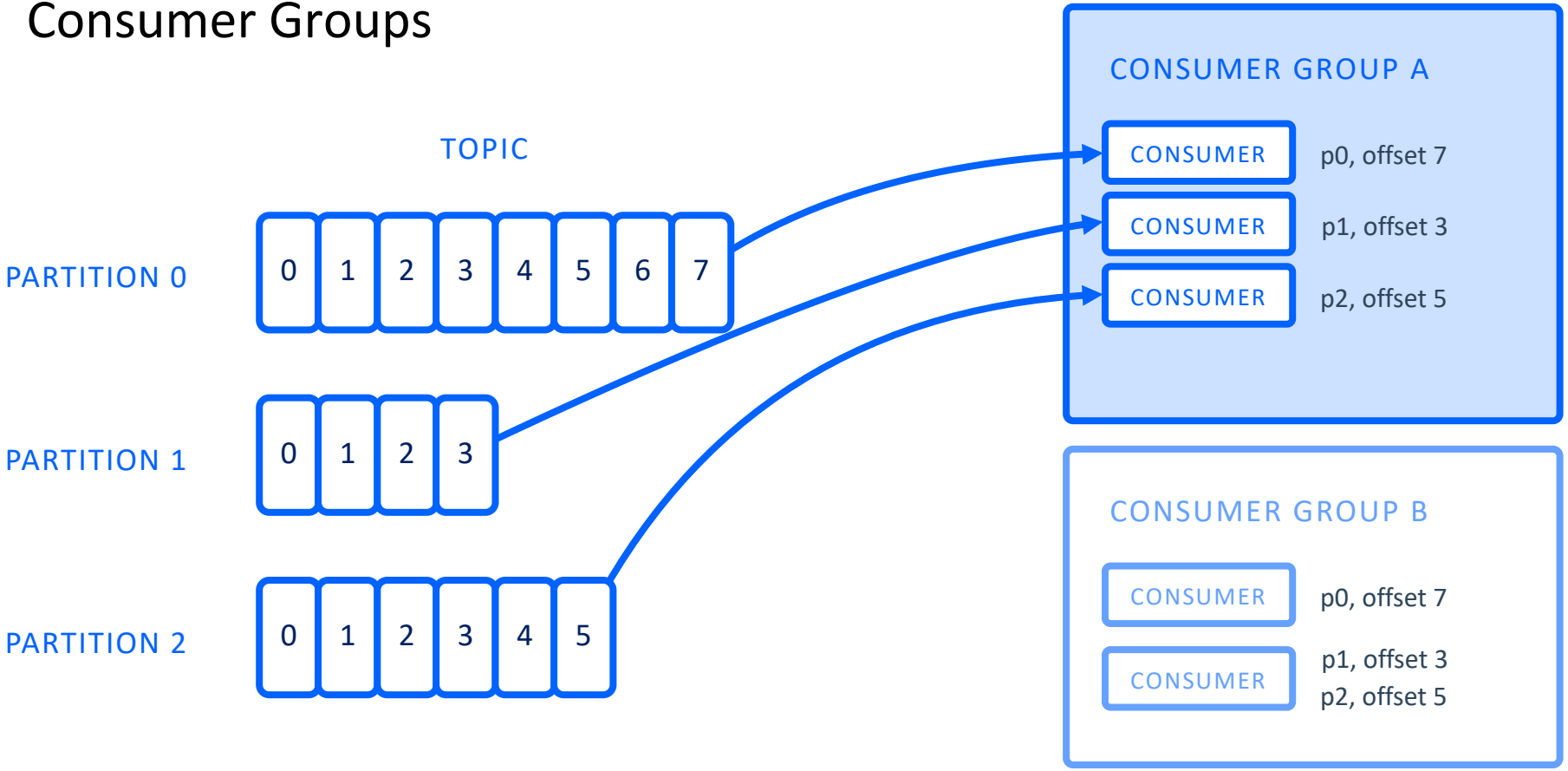


Consumer Groups

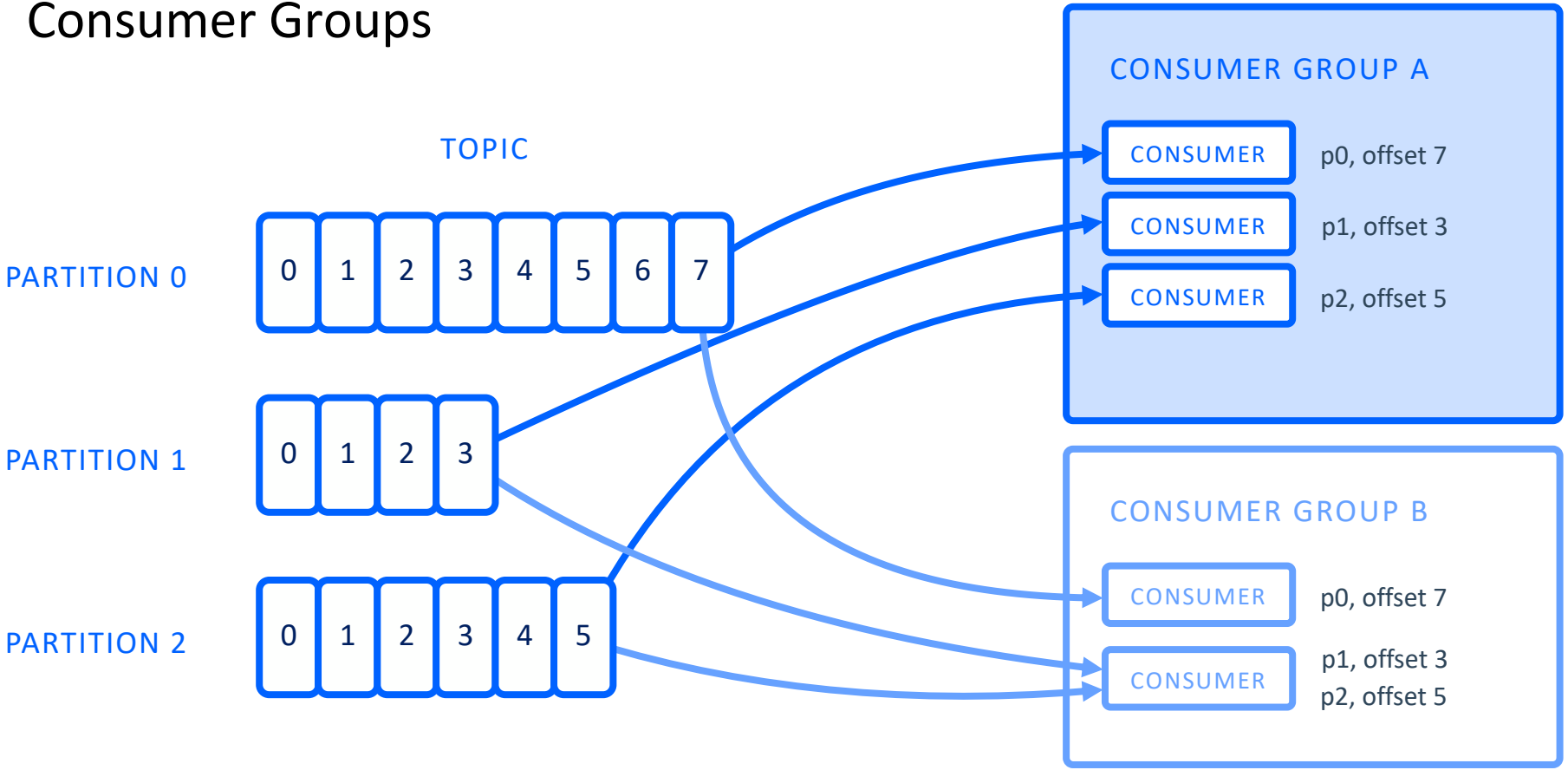




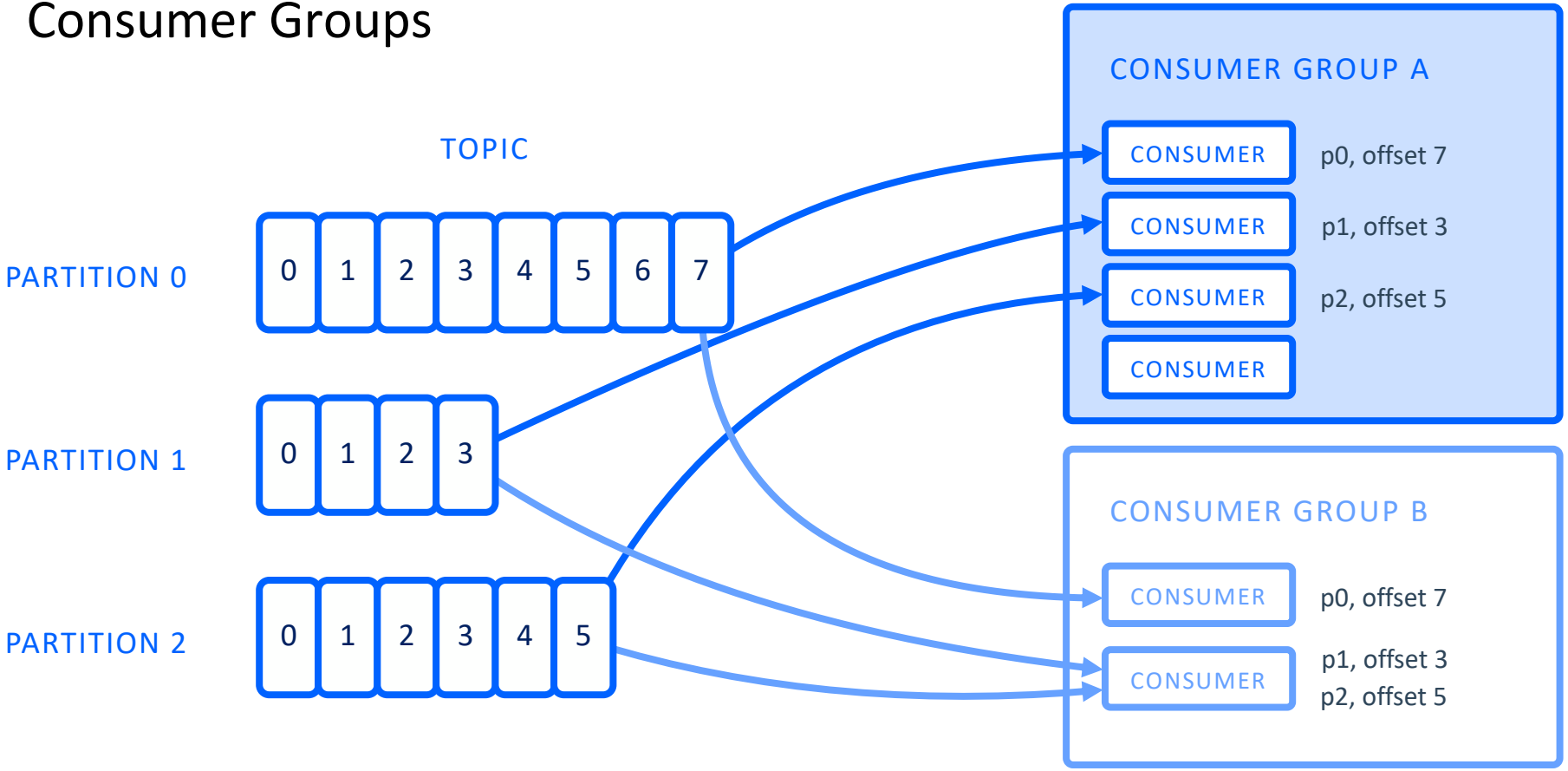
Consumer Groups



Consumer Groups



Consumer Groups



How do we get started?

Quickstart

This tutorial assumes you are starting fresh and have no existing Kafka or ZooKeeper data. Since Kafka console scripts are different for Unix-based and Windows platforms, on Windows platforms use `bin\windows\` instead of `bin/`, and change the script extension to `.bat`.

Step 1: Download the code

Download the 2.3.0 release and un-tar it.

```
> tar -xzf kafka_2.12-2.3.0.tgz
> cd kafka_2.12-2.3.0
```

Step 2: Start the server

Kafka uses ZooKeeper so you need to first start a ZooKeeper server if you don't already have one. You can use the convenience script packaged with kafka to get a quick-and-dirty single-node ZooKeeper instance.

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
[2013-04-22 15:01:37,495] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.ser
...
```

Now start the Kafka server:

```
> bin/kafka-server-start.sh config/server.properties
[2013-04-22 15:01:47,028] INFO Verifying properties (kafka.utils.VerifiableProperties)
[2013-04-22 15:01:47,051] INFO Property socket.send.buffer.bytes is overridden to 1048576 (kafka.utils.Verifiabl
...
```


https://kafka.apache.org/quickstart

org.apache.kafka.clients.producer

Class KafkaProducer<K,V>

java.lang.Object
    org.apache.kafka.clients.producer.KafkaProducer<K,V>

All Implemented Interfaces:

`java.io.Closeable, java.lang.AutoCloseable, Pro`

```
public class KafkaProducer<K,V>
extends java.lang.Object
implements Producer<K,V>
```

A Kafka client that publishes records to the Kafka cluster.

The producer is *thread safe* and sharing a single producer instar

Here is a simple example of using the producer to send records

```
 Properties props = new Properties();
 props.put("bootstrap.servers", "localhost:9092
 props.put("acks", "all");
 props.put("retries", 0);
 props.put("batch.size", 16384);
 props.put("linger.ms", 1);
 props.put("buffer.memory", 33554432);
 props.put("key.serializer", "org.apache.kafka.
 props.put("value.serializer", "org.apache.kafk

 Producer<String, String> producer = new KafkaP
```

org.apache.kafka.clients.consumer

Class KafkaConsumer<K,V>

java.lang.Object
    org.apache.kafka.clients.consumer.KafkaConsumer<K,V>

All Implemented Interfaces:

`java.io.Closeable, java.lang.AutoCloseable, Consumer<K,V>`

```
public class KafkaConsumer<K,V>
extends java.lang.Object
implements Consumer<K,V>
```

A client that consumes records from a Kafka cluster.

This client transparently handles the failure of Kafka brokers, and transparently adapts as topic partitions it fetches migrate with broker to allow groups of consumers to load balance consumption using consumer groups.

The consumer maintains TCP connections to the necessary brokers to fetch data. Failure to close the consumer after use will leak safe. See Multi-threaded Processing for more details.

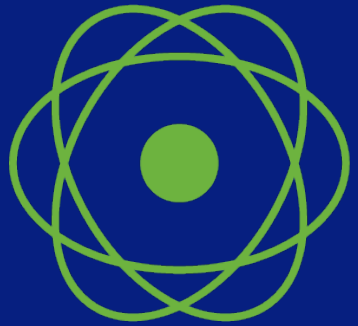
Cross-Version Compatibility

This client can communicate with brokers that are version 0.10.0 or newer. Older or newer brokers may not support certain featu offsetsForTimes, because this feature was added in version 0.10.1. You will receive an `UnsupportedVersionException` when running broker version.

Offsets and Consumer Position

Reactive Frameworks for Kafka

Reactor Kafka



MicroProfile Reactive Messaging

Alpakka Kafka Connector

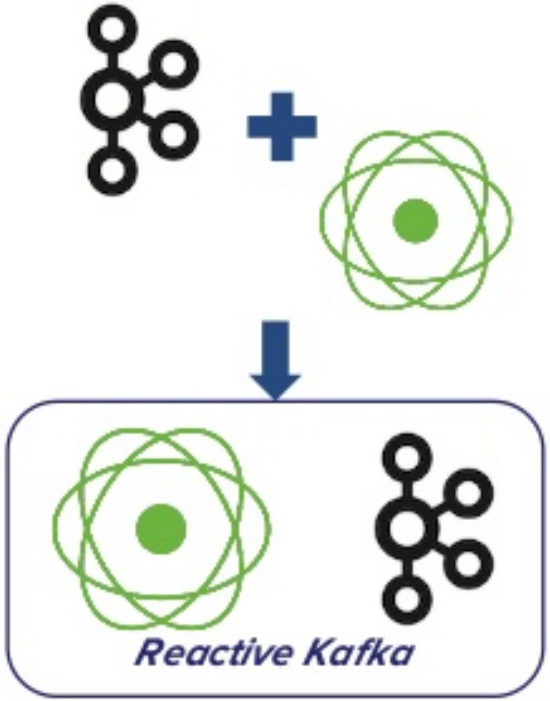
Vert.x Kafka Client

Reactor Kafka

Reactor Kafka is a thin layer of reactive streams over Kafka client

Hides the complexity of Kafka

Built-in backpressure support

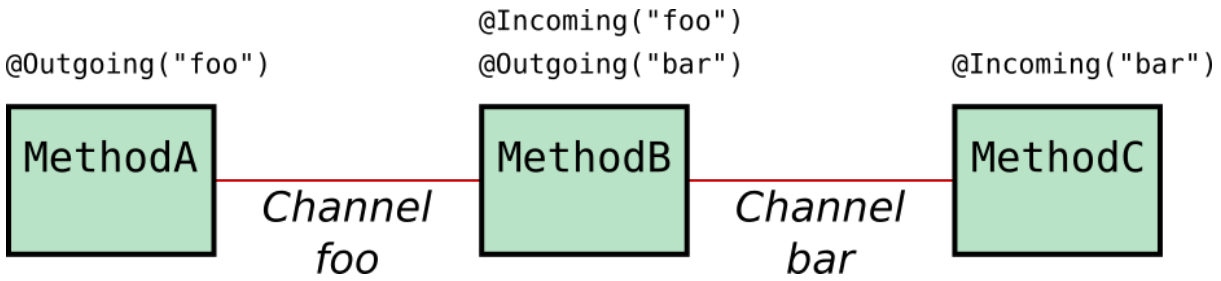


MicroProfile Reactive Messaging

This specification provides asynchronous messaging support based on Reactive Streams for MicroProfile.
Implementations include:

- SmallRye Reactive Messaging 1.0.0
- Open Liberty 19.0.0.9 (via SmallRye)
- Quarkus 1.0.0.Final (via SmallRye)
- WebSphere Liberty 19.0.0.9 (via SmallRye)

Reactive Messaging uses a model of annotated methods which are connected by named *channels*.



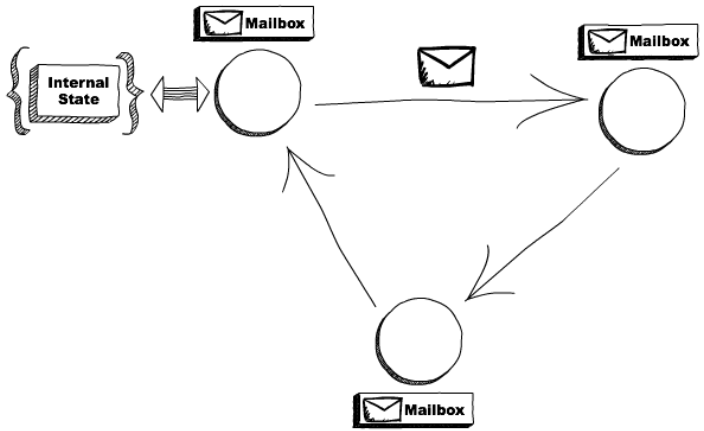
Alpakka Kafka Connector

Enables connection between Apache Kafka and Akka Streams.

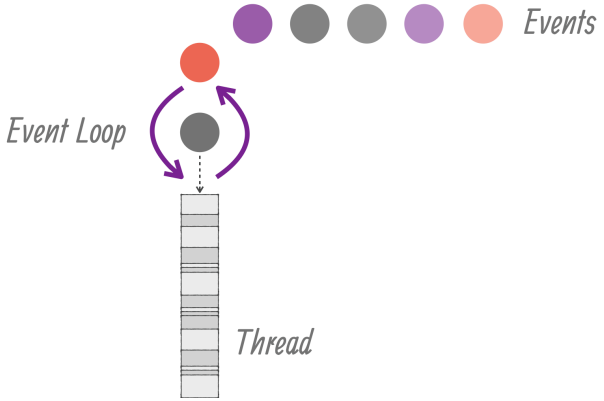
Hides complexity of Kafka

Built-in backpressure support

Akka Streams uses Akka actors as its foundation

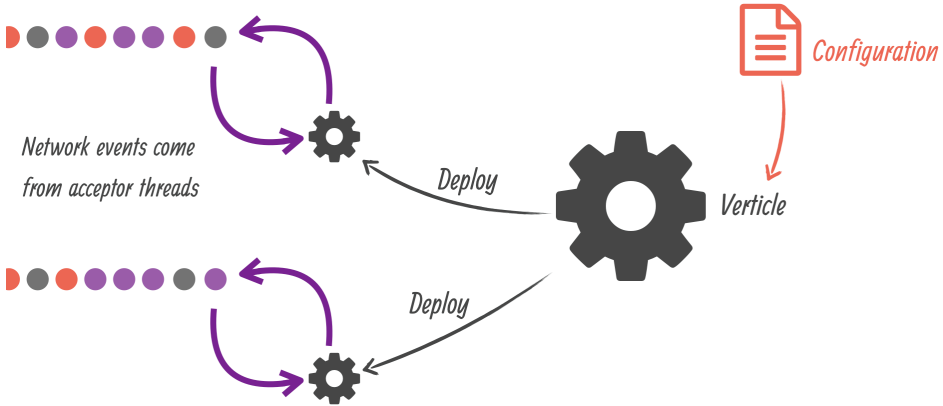


What is Vert.x?

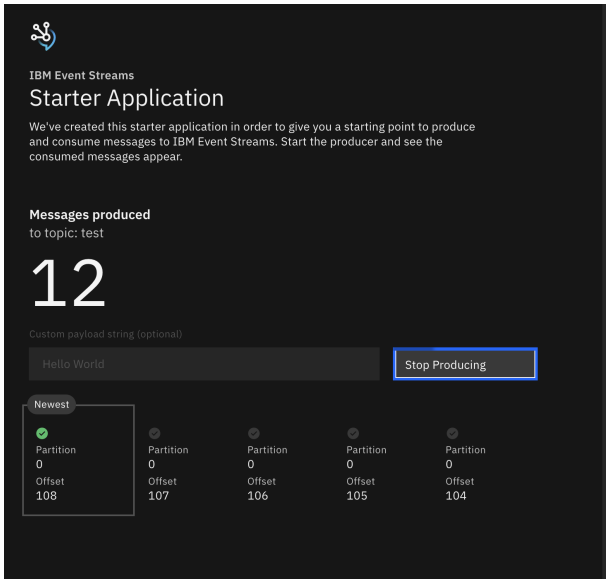


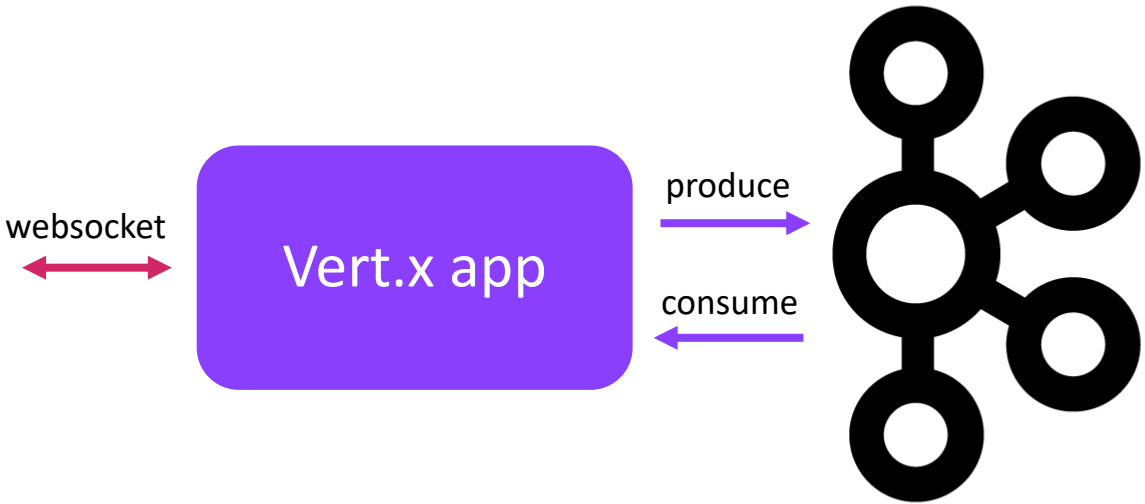
VERT.X

- Polyglot Framework
- Based on Reactor pattern
- Runs on the JVM
- Non-blocking
- Event-driven
- Includes distributed event-bus
- Code is single-threaded



Demo app





https://github.com/ibm-messaging/kafka-java-vertx-starter

org.apache.kafka.clients.producer.KafkaProducer<K, V>

```
for (int i = 0; i < 100; i++) {

    ProducerRecord<String, String> producerRecord =
            new ProducerRecord<>(“my-topic”, Integer.toString(i));

    producer.send(producerRecord);

}
```

Vert.x Kafka Client - Producer

```
for (int i = 0; i < 100; i++) {

  KafkaProducerRecord<String, String> producerRecord =
          KafkaProducerRecord.create(“my-topic”, Integer.toString(i));

  producer.write(producerRecord);

}
```

org.apache.kafka.clients.consumer.KafkaConsumer<K, V>

```
while (true) {

   ConsumerRecords<String, String> records = consumer.poll(100);

   for (ConsumerRecord<String, String> record : records) {
      processRecord(record);
      consumer.commit();
   }

}
```

Vert.x Kafka Client - Consumer

```
consumer.handler(record -> {

    processRecord(record, processResult -> {

      consumer.commit(record.offset(), commitResult -> {
          // handle outcome
      });

    });

});
```

ibm-messaging / **kafka-java-vertx-starter**

Watch 2 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Security | Insights

Starter Java app for testing connection to Apache Kafka with Vert.x

1 commit | **2** branches | **0** packages | **0** releases | **1** contributor | Apache-2.0

Branch: **master** | New pull request | Find file | Clone or download

katheris feat: Version 0.0.1 of the app ... Latest commit `bc00def` 1 hour ago

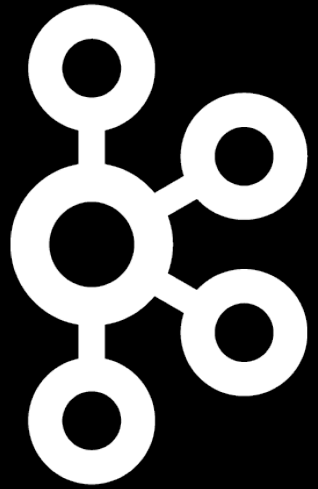
src/main	feat: Version 0.0.1 of the app	1 hour ago
.gitignore	feat: Version 0.0.1 of the app	1 hour ago
LICENSE	feat: Version 0.0.1 of the app	1 hour ago
README.md	feat: Version 0.0.1 of the app	1 hour ago
pom.xml	feat: Version 0.0.1 of the app	1 hour ago

Summary

Building reactive can help you!

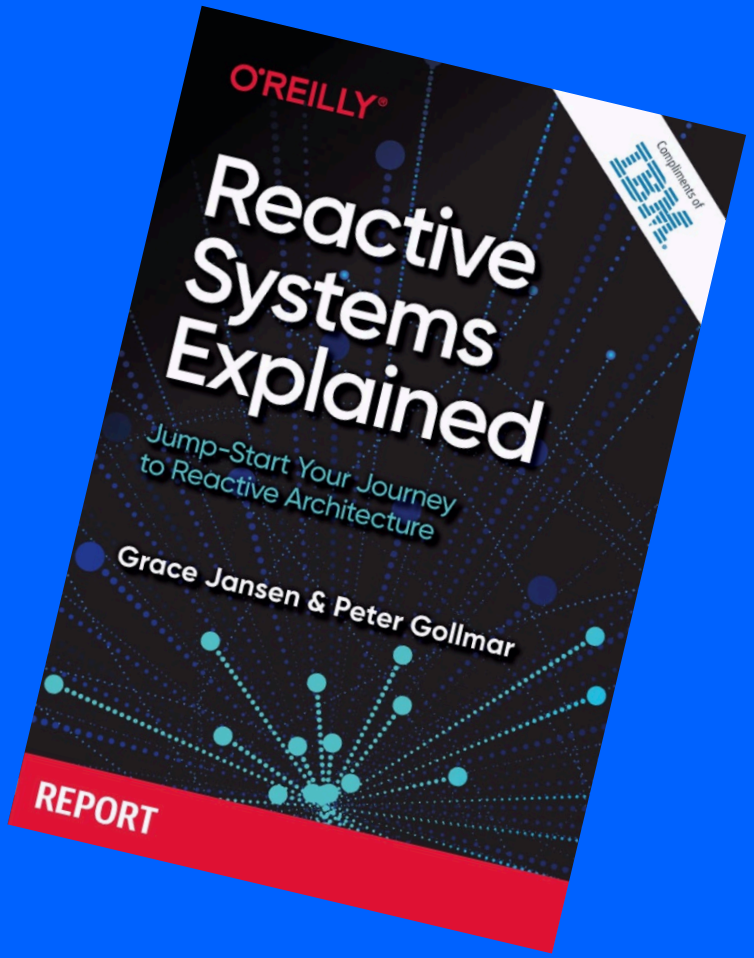
Kafka is useful, but think carefully when designing your system

You are not alone! Lots of reactive frameworks to help you along the way



Reactive Systems Explained

ibm.biz/ReactiveReport



Thank you

Kate Stanley | @katestanley91
Grace Jansen | @gracejansen27

Reactive Manifesto:

https://www.reactivemanifesto.org

Getting started with Kafka:

https://kafka.apache.org/quickstart

https://strimzi.io

Reactive Kafka libraries

https://vertx.io/docs/vertx-kafka-client/java/

https://github.com/eclipse/microprofile-reactive-messaging

https://projectreactor.io/docs/kafka

https://github.com/akka/alpakka-kafka

Starter app

https://github.com/ibm-messaging/kafka-java-vertx-starter

IBM Event Streams: ibm.biz/aboutEventStreams