

# Kafka For Developers Using Spring Boot



**Dilip Sundarraaj**

# About Me

- Dilip
- Building Software's since 2008
- Teaching in **UDEMY** Since 2016

# Whats Covered?

- Introduction to Kafka and internals of Kafka
- Building Enterprise standard Kafka Clients using **Spring-Kafka/SpringBoot**
- Resilient Kafka Client applications using **Error-Handling/Retry/Recovery**
- Writing Unit/Integration tests using **JUnit**

# Targeted Audience

- Focused for developers
- Interested in learning the internals of Kafka
- Interested in building Kafka Clients using Spring Boot
- Interested in building Enterprise standard Kafka client applications using Spring boot

# Source Code

**Thank You !**

# **Introduction to Apache Kafka**

# Prerequisites

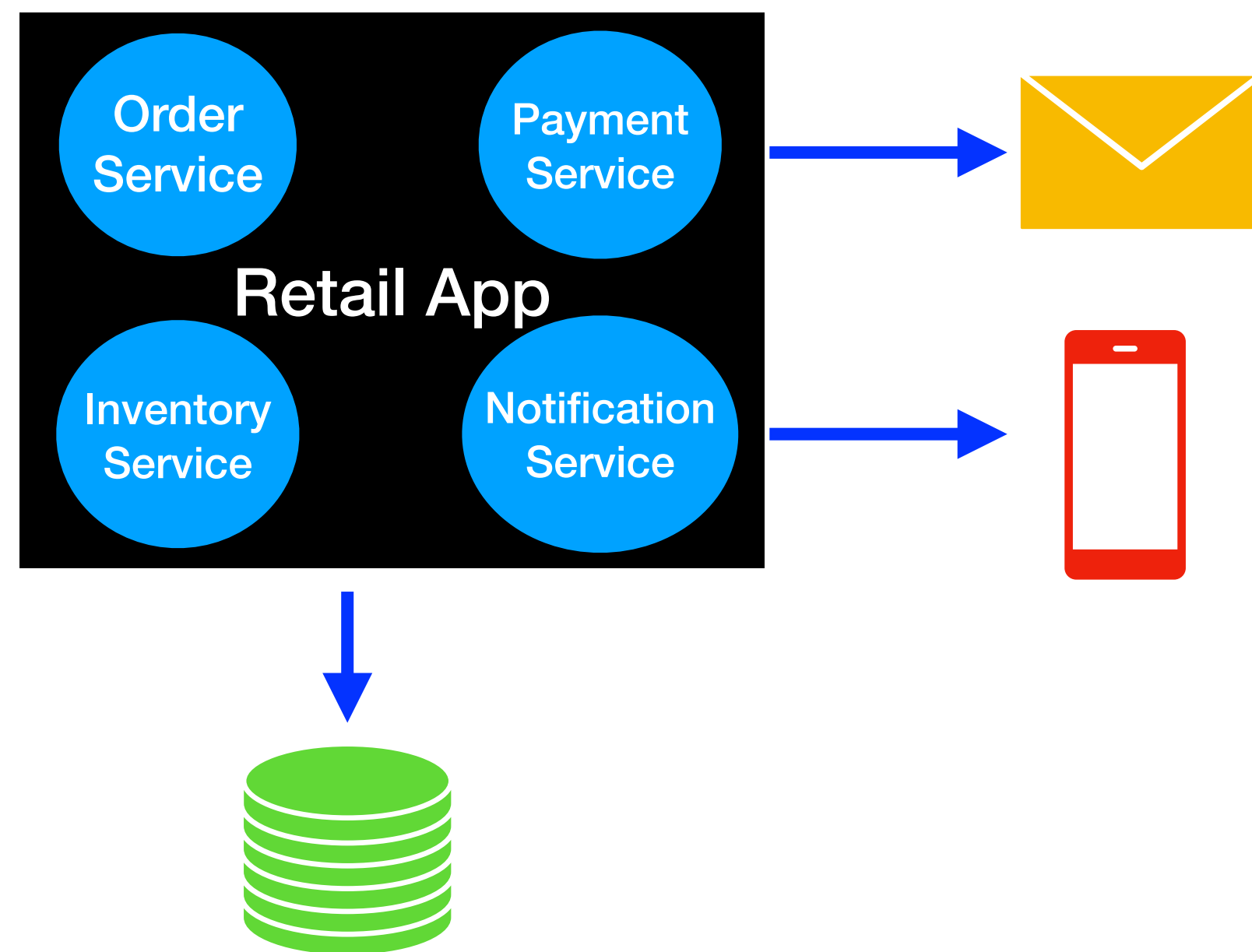


# Course Prerequisites

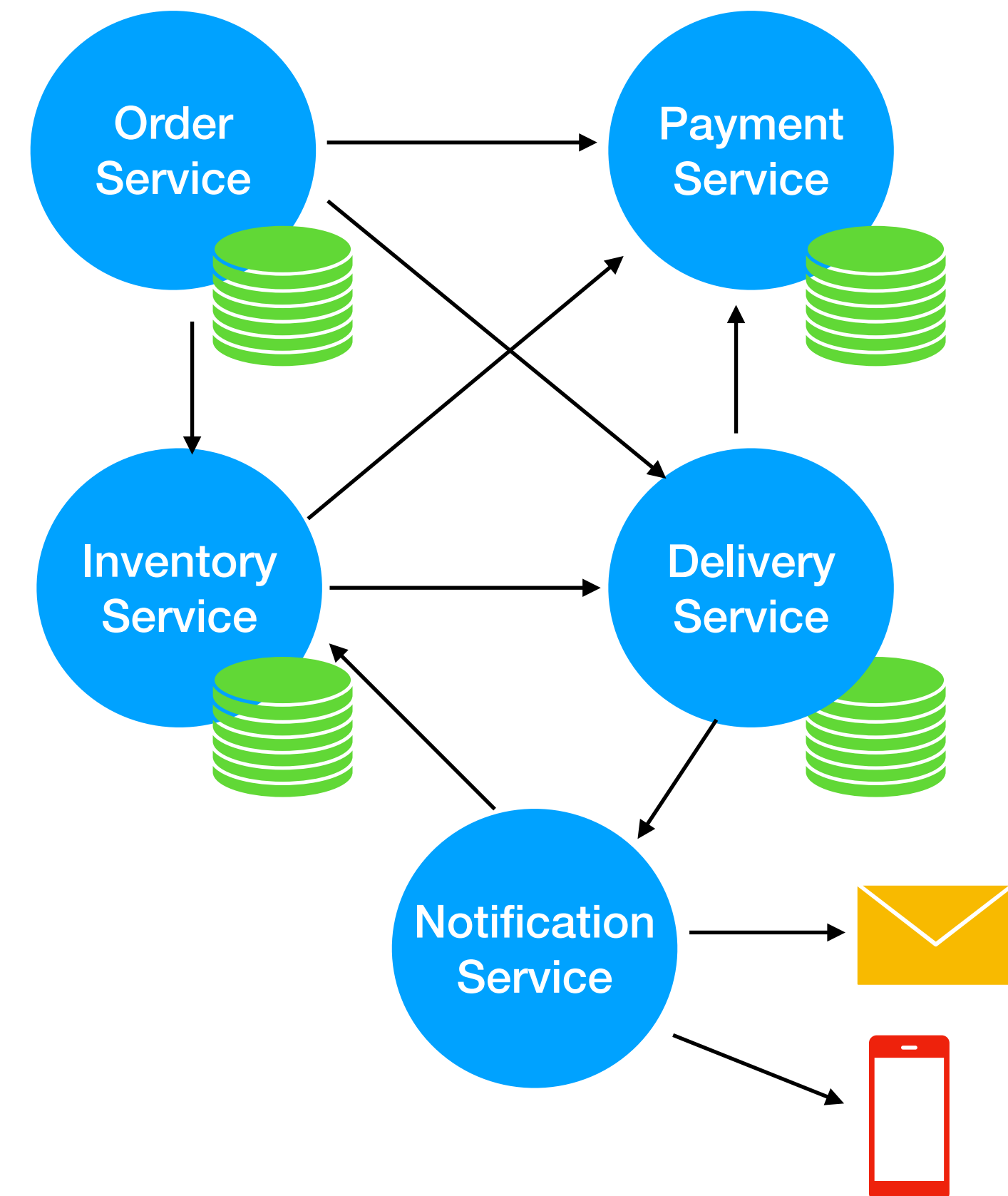
- Prior Knowledge or Working Experience with **Spring Boot/Framework**
- Knowledge about building **Kafka Clients** using Producer and Consumer API
- Knowledge about building **RESTFUL APIs** using Spring Boot
- Experience working with **Spring Data JPA**
- Automated tests using **JUnit**
- Experience Working with **Mockito**
- **Java 11 or Higher** is needed
- **Intellij , Eclipse** or any other IDE is needed

# Software Development

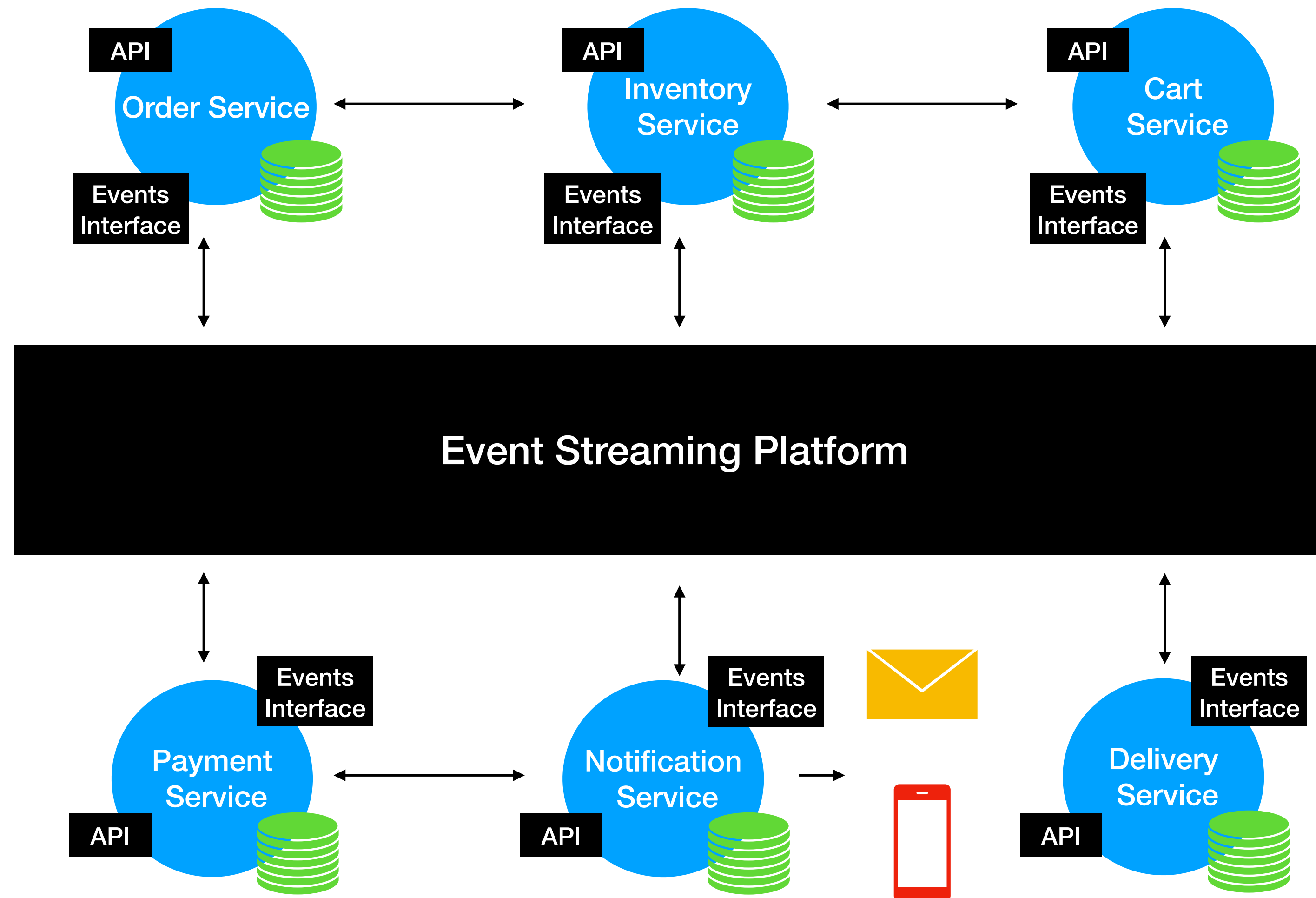
Past



Current

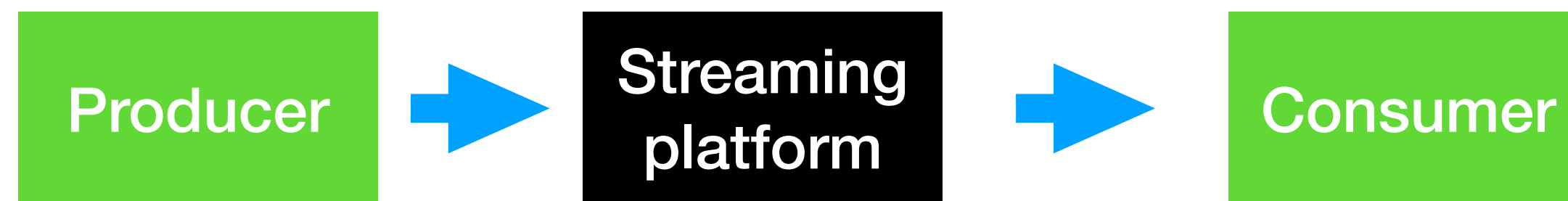


# MicroServices Architecture

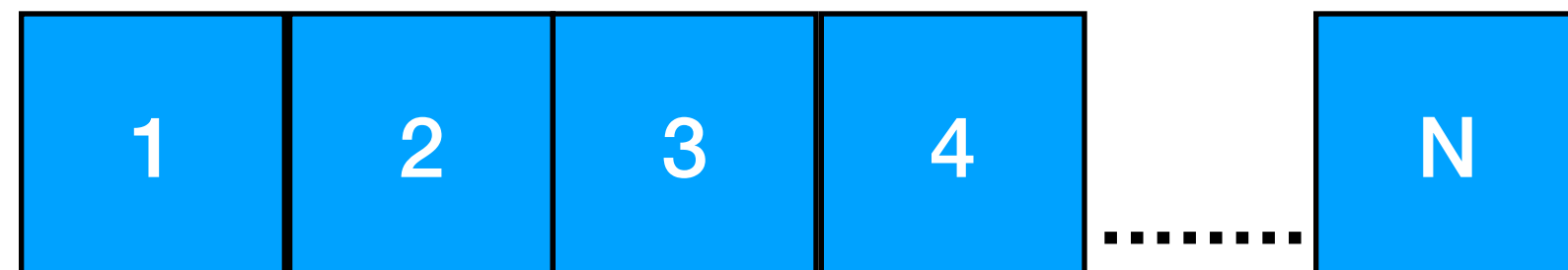


# What is an Event Streaming Platform?

- Producers and Consumers subscribe to a stream of records

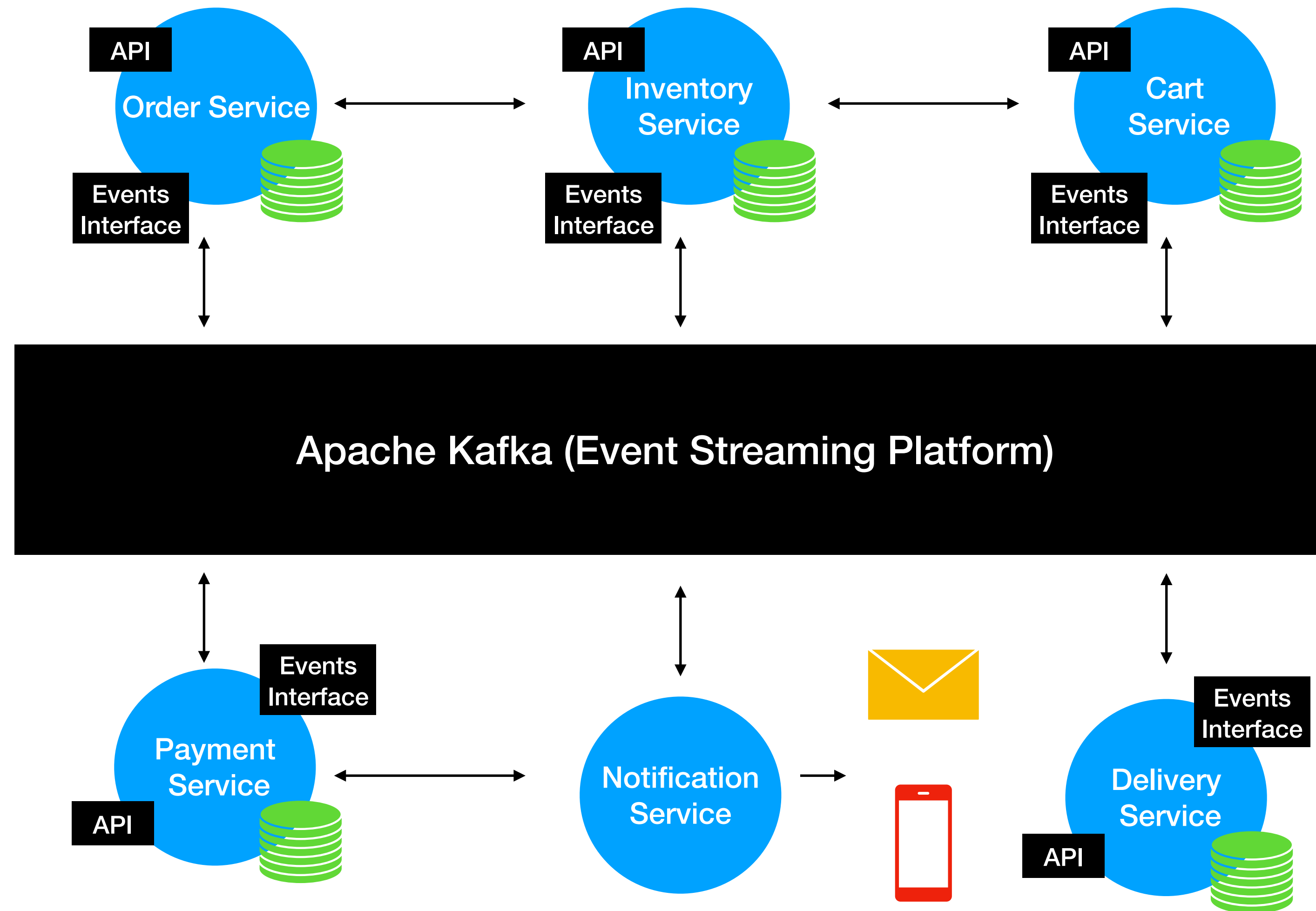


- Store stream of Events



- Analyze and Process Events as they occur

# Apache Kafka (Event Streaming Platform)



# Traditional Messaging System

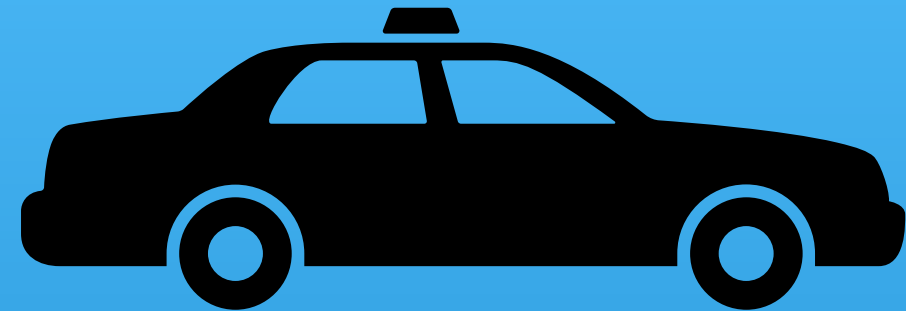
- Transient Message Persistence
- Brokers responsibility to keep track of consumed messages
- Target a specific Consumer
- Not a distributed system

# Kafka Streaming Platform

- Stores events based on a retention time. Events are Immutable
- Consumers Responsibility to keep track of consumed messages
- Any Consumer can access a message from the broker
- It's a distributed streaming system

# Kafka Use Cases

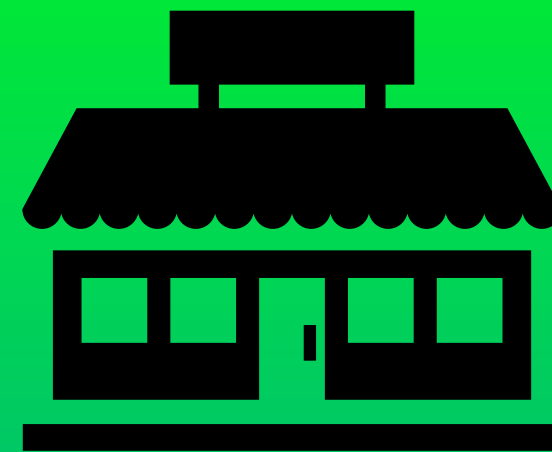
## Transportation



**Driver-Rider Notifications**

**Food Delivery Notifications**

## Retail



**Sale Notifications**

**RealTime Purchase  
recommendations**

**Tracking Online Order  
Deliveries**

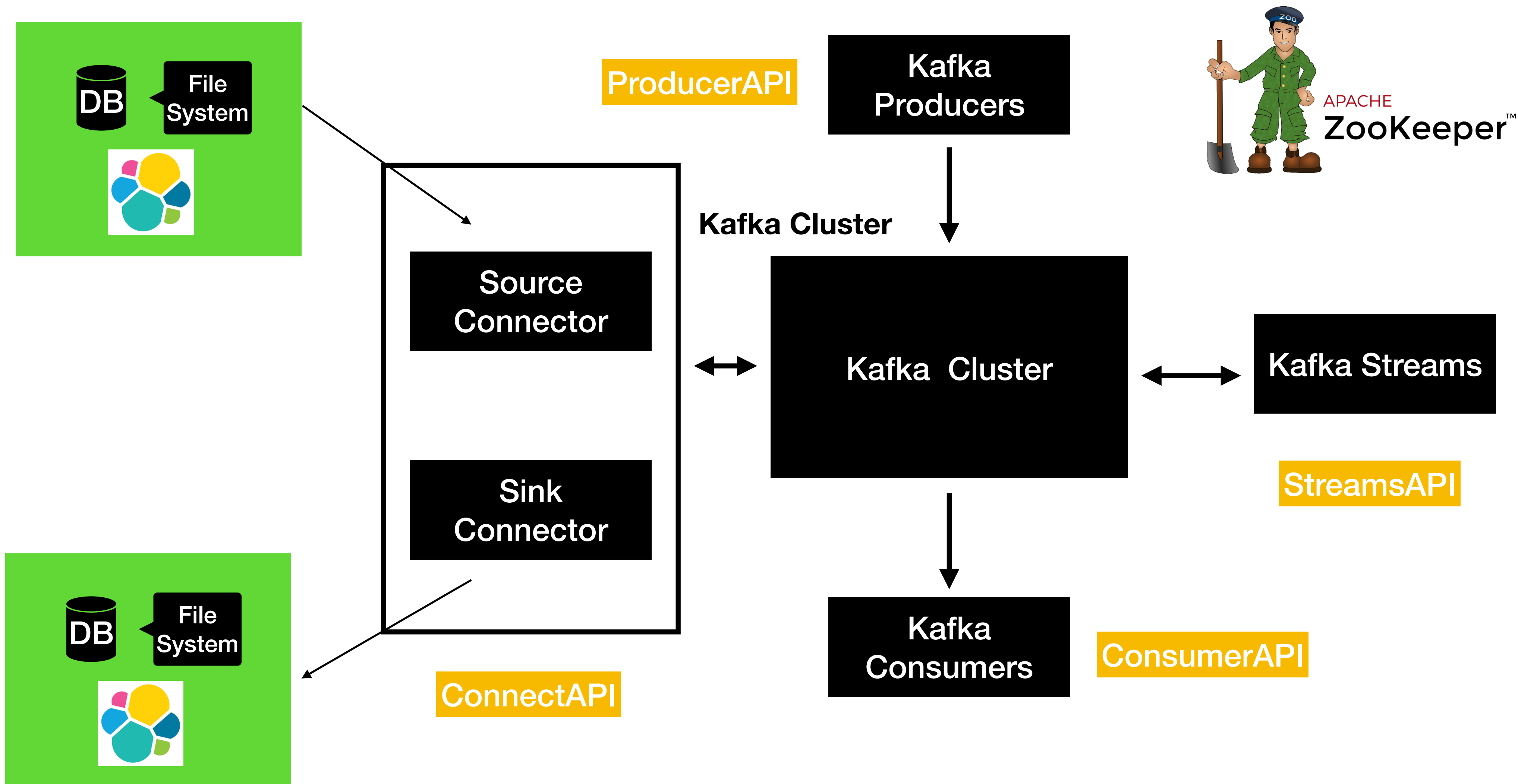
## Banking



**Fraud Transactions**

**New Feature/Product  
notifications**

# Kafka Terminology & Client APIs





# Download Kafka

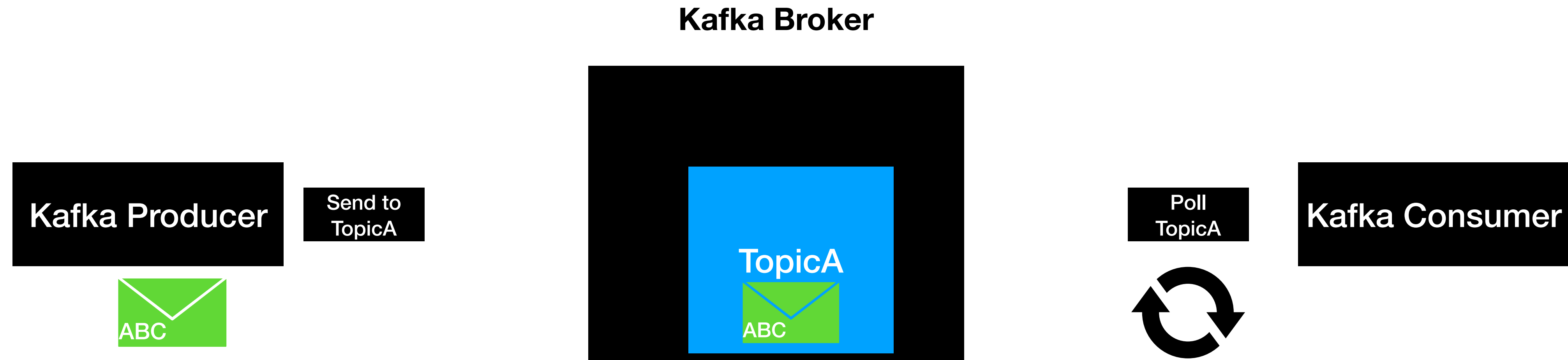
# **Kafka Topics & Partitions**

# Kafka Topics

- Topic is an **Entity** in Kafka with a name

# Kafka Topics

- Topic is an **Entity** in Kafka with a name

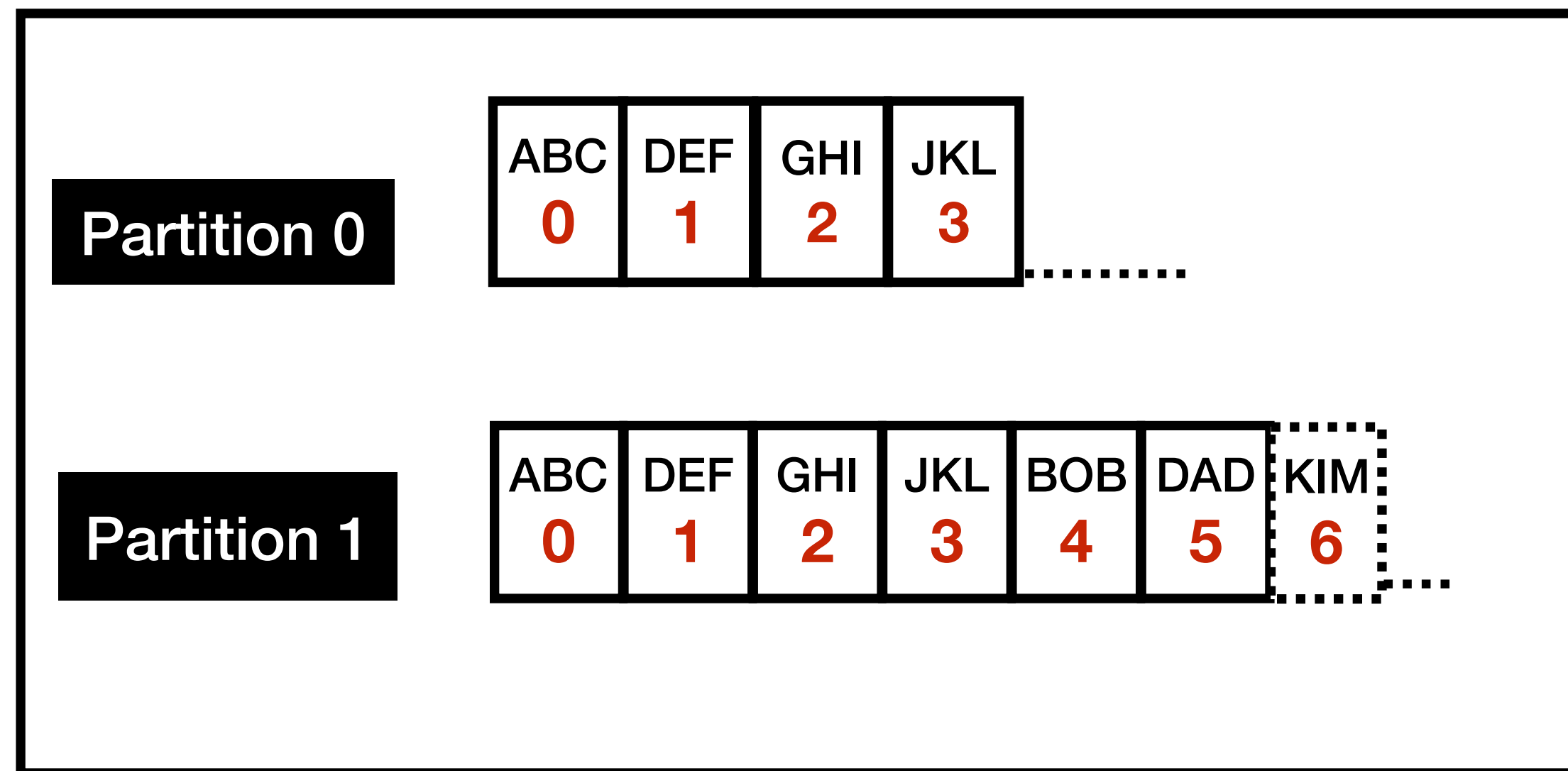


# Topic and Partitions

- Partition is where the message lives inside the topic
- Each Topic will be create with one or more partitions

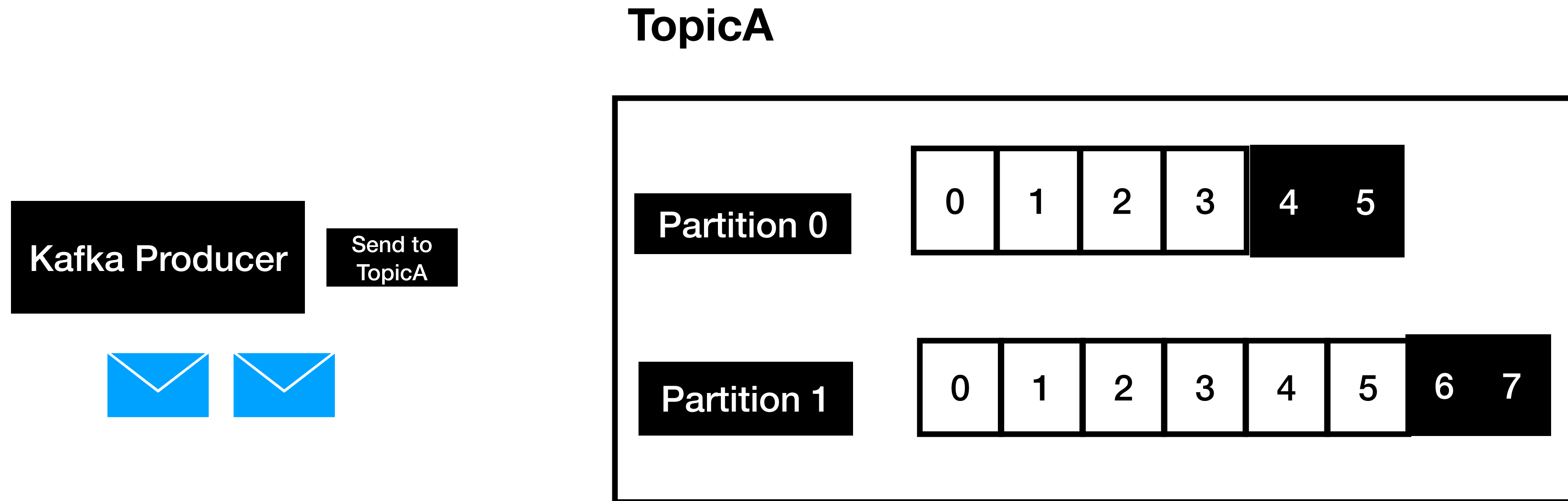
# Topic and Partitions

## TopicA



- Each Partition is an ordered , immutable sequence of records
- Each record is assigned a sequential number called **offset**
- Each partition is independent of each other
- Ordering is guaranteed only at the partition level
- Partition continuously grows as new records are produced
- All the records are persisted in a commit log in the file system where Kafka is installed

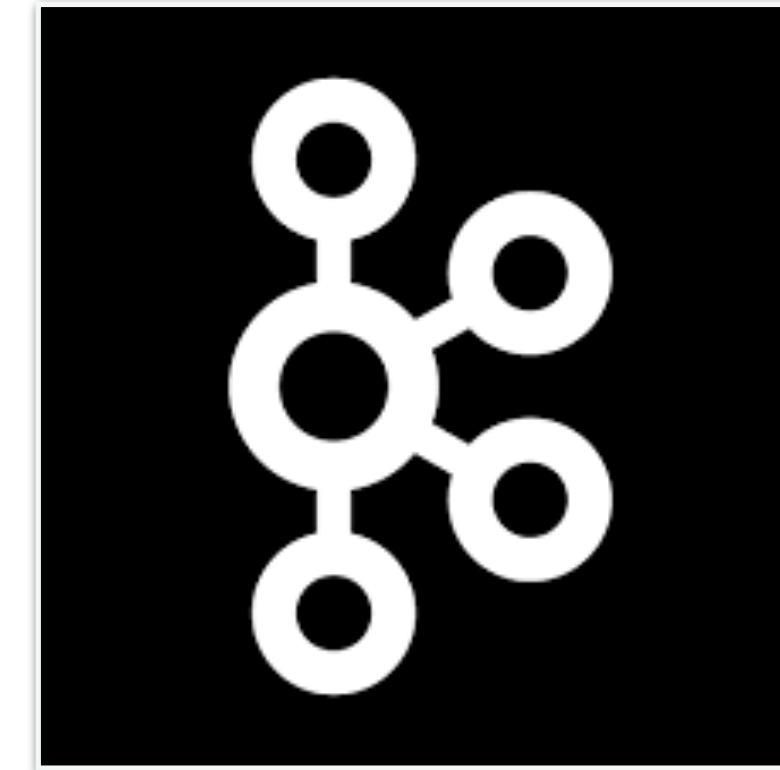
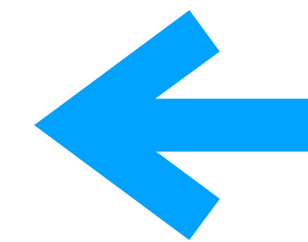
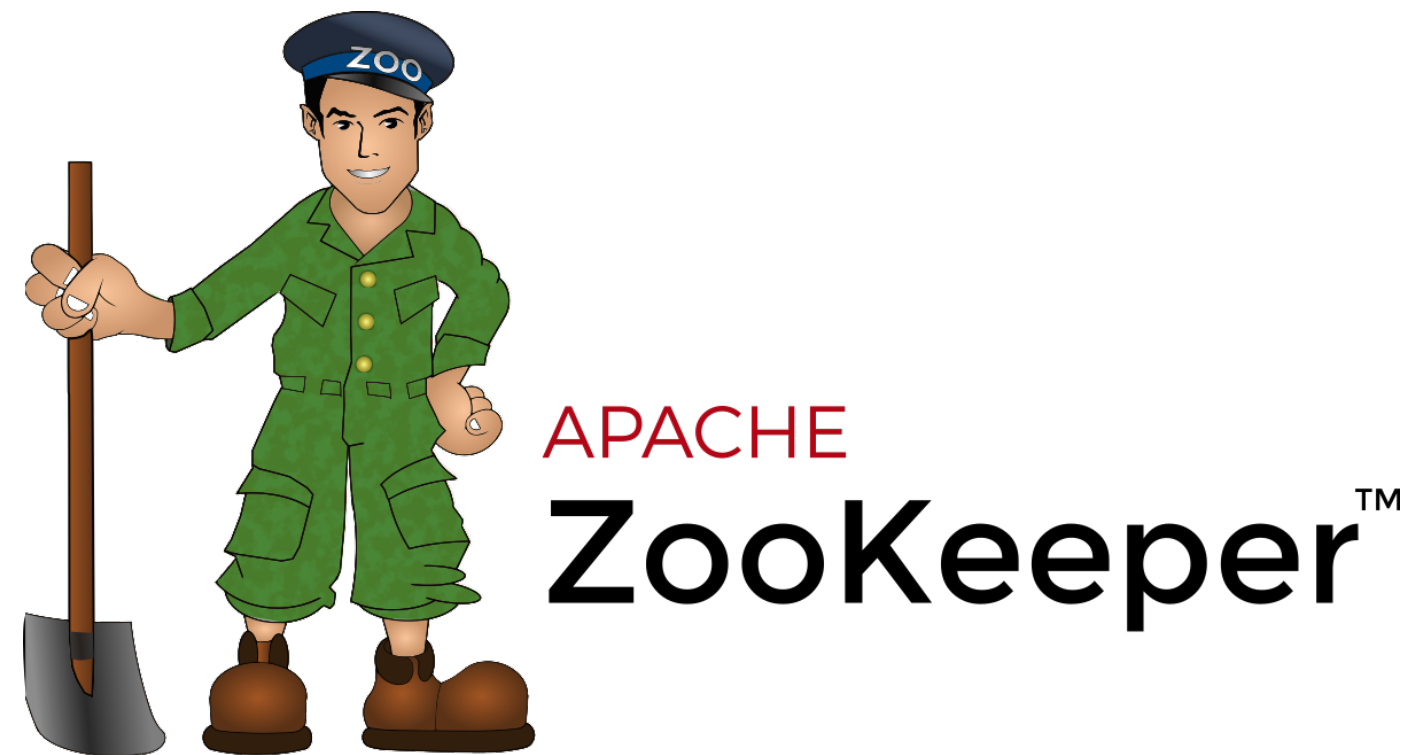
# Topics and Partitions



# Setting up Zookeeper & Kafka Broker



# Setting up Kafka in Local



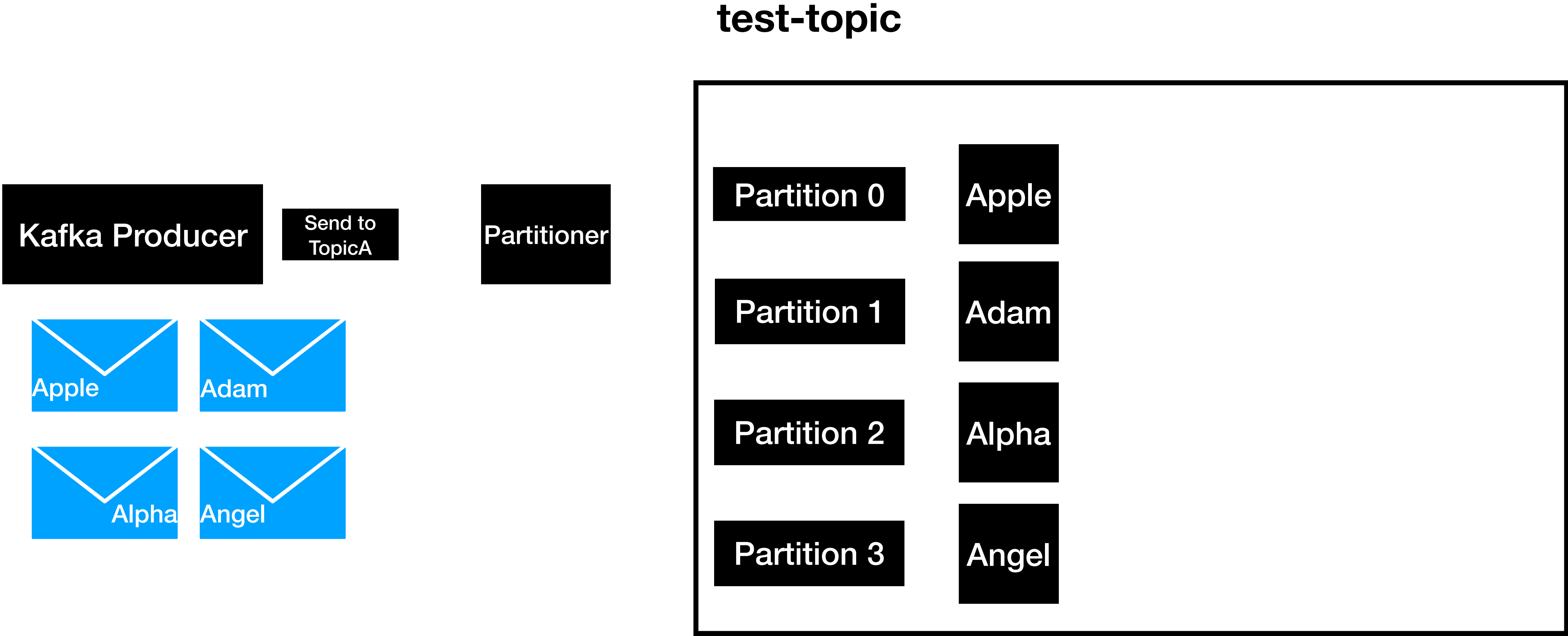
Broker registered  
with zookeeper

# **Sending Kafka Messages With Key and Value**

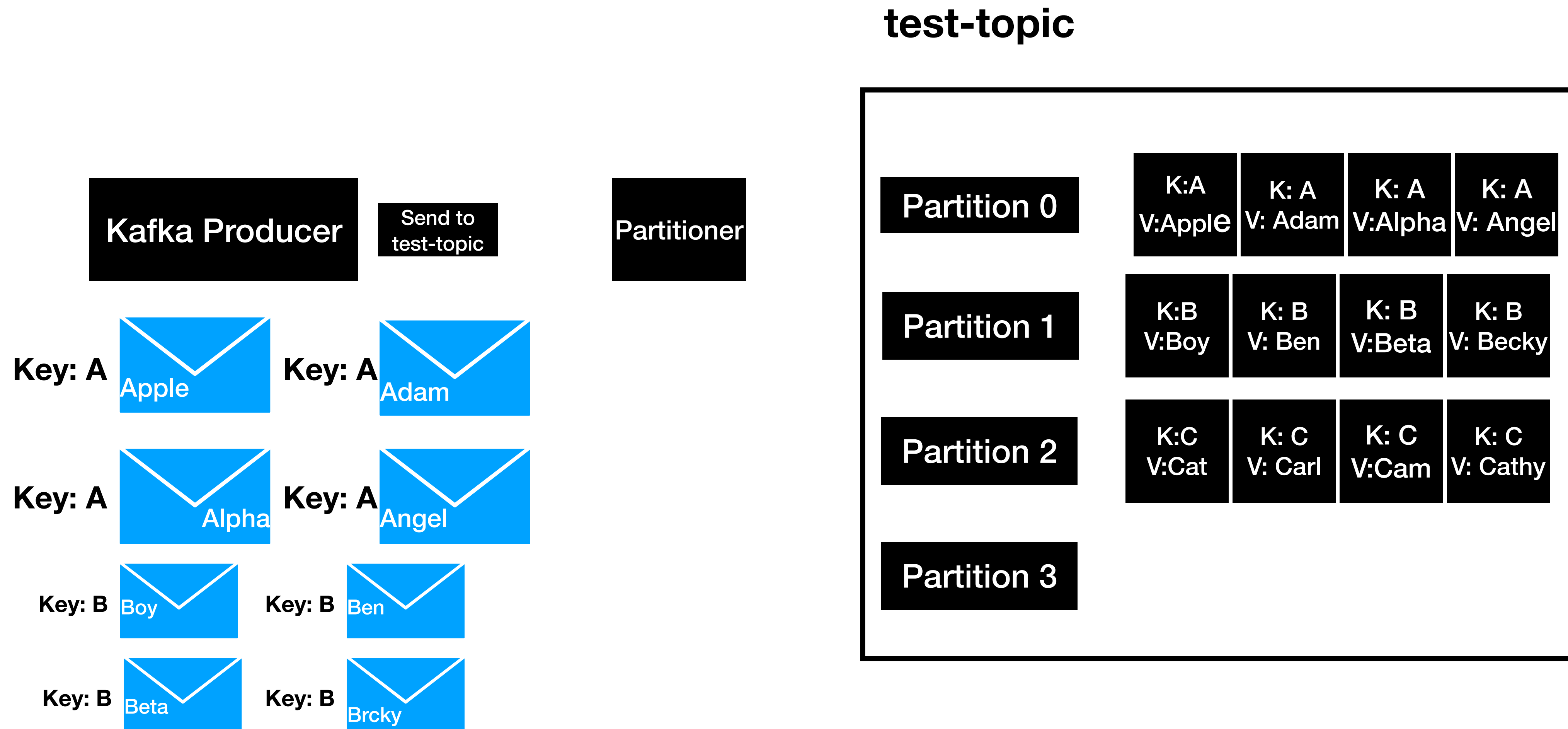
# Kafka Message

- Kafka Message these sent from producer has two properties
  - Key (optional)
  - Value

# Sending Message Without Key



# Sending Message With Key

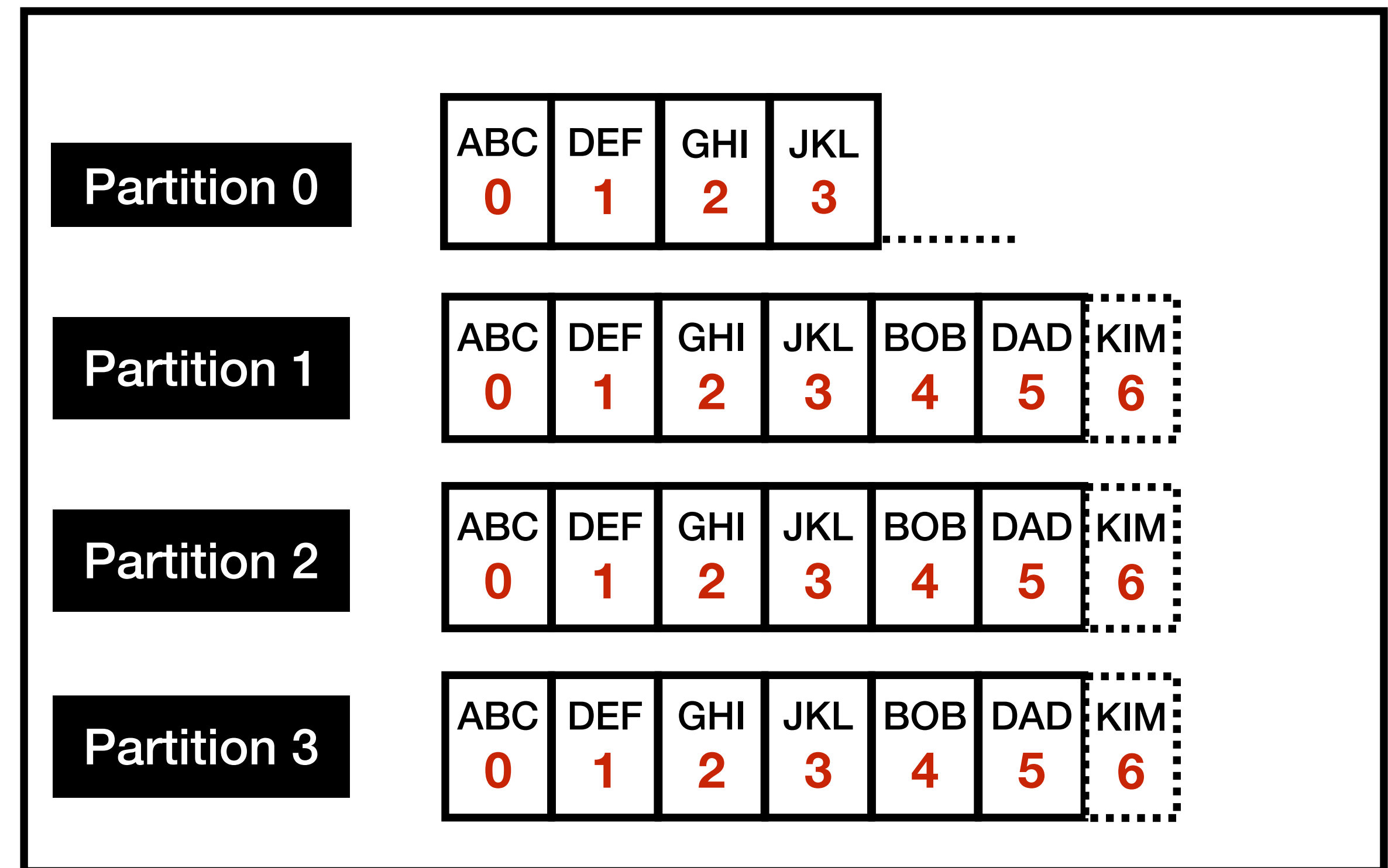


# Consumer Offsets

# Consumer Offsets

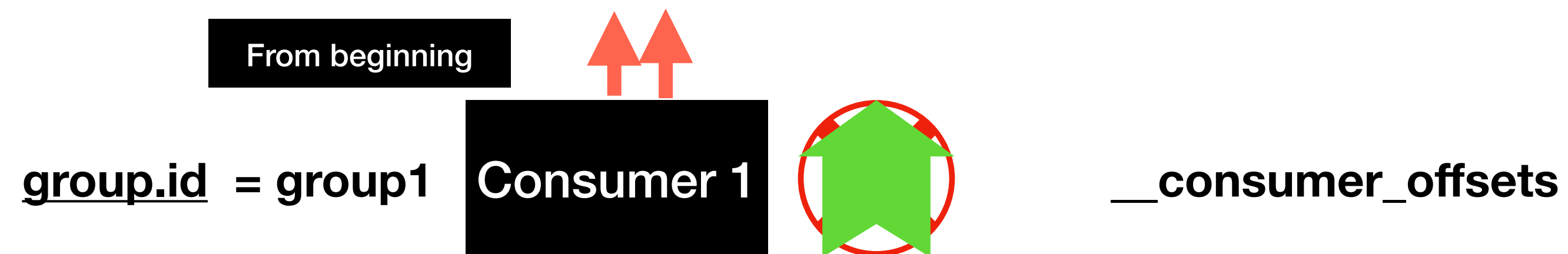
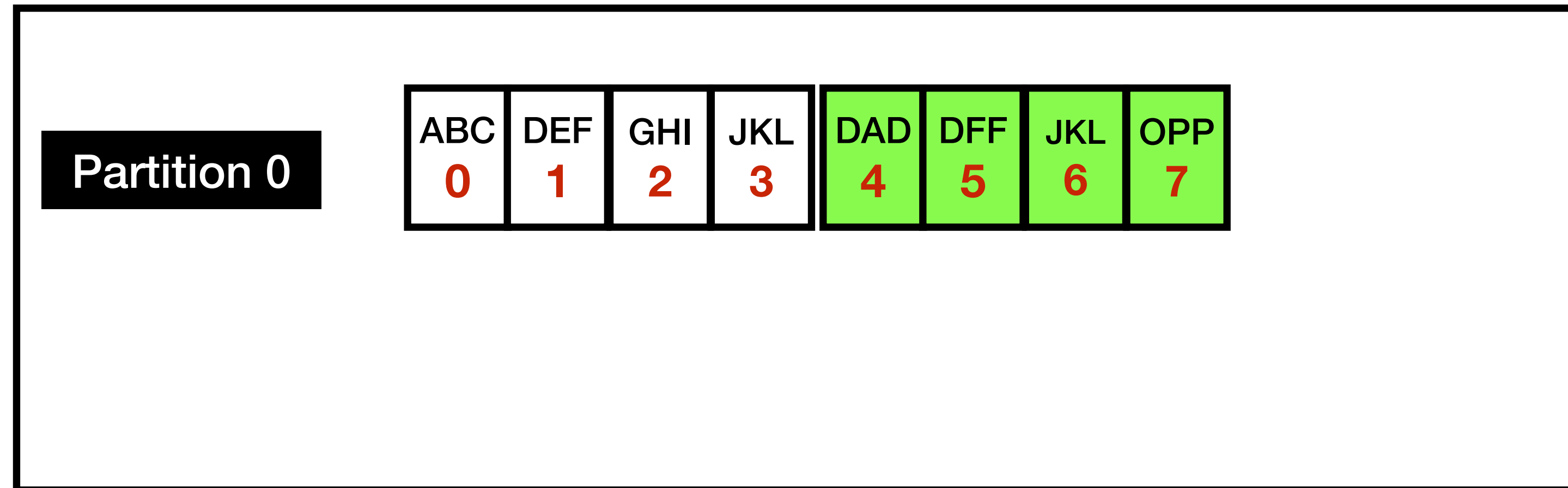
- Consumer have three options to read
  - from-beginning
  - latest
  - specific offset

**test-topic**



# Consumer Offsets

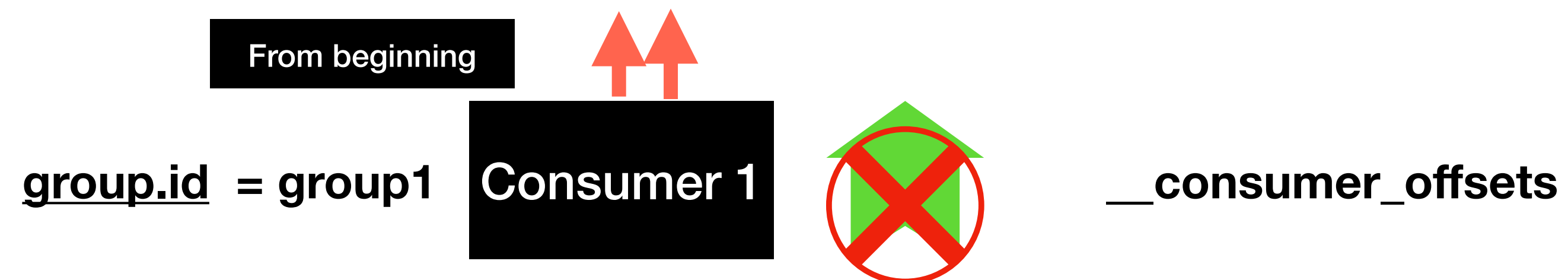
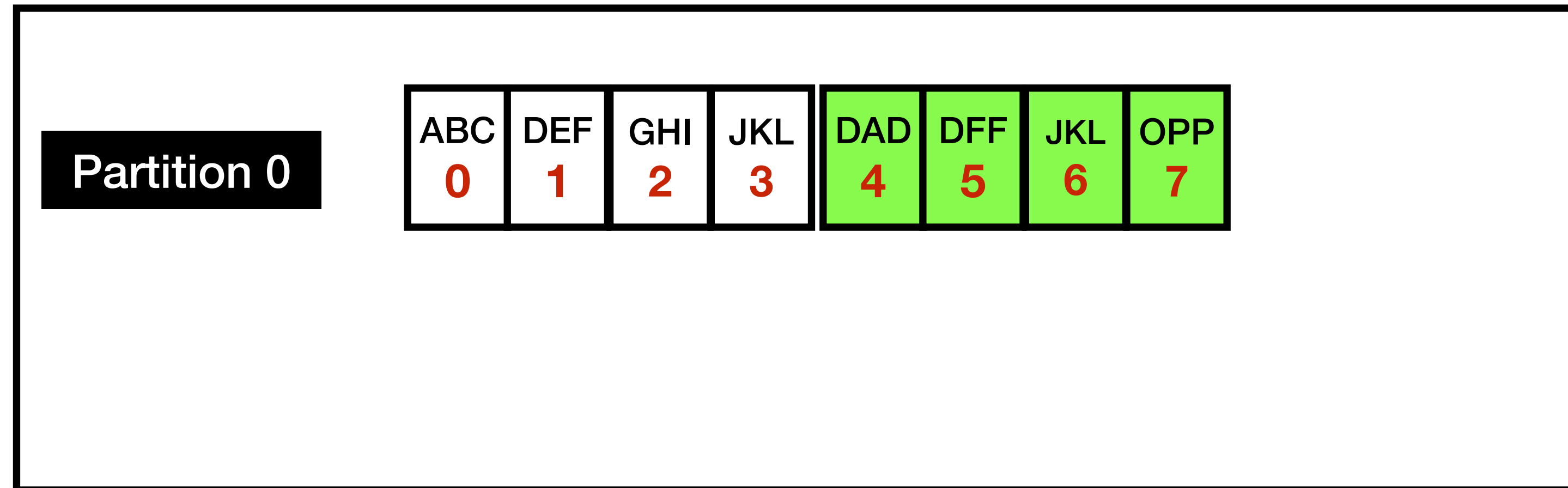
test-topic





# Consumer Offsets

test-topic



# Consumer Offset

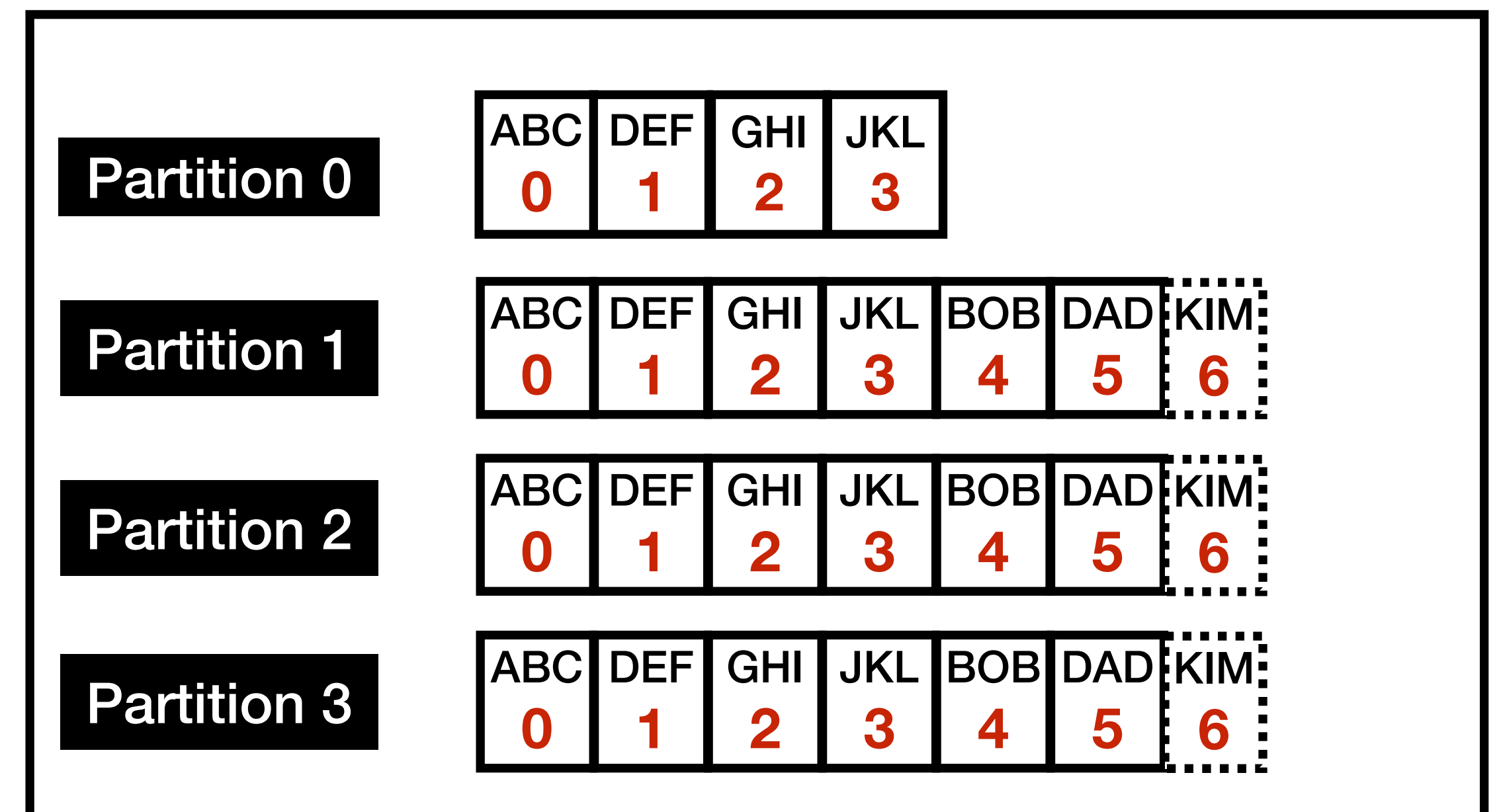
- Consumer offsets behaves like a bookmark for the consumer to start reading the messages from the point it left off.

# Consumer Groups

# Consumer Groups

- group.id is mandatory
- group.id plays a major role when it comes to scalable message consumption.

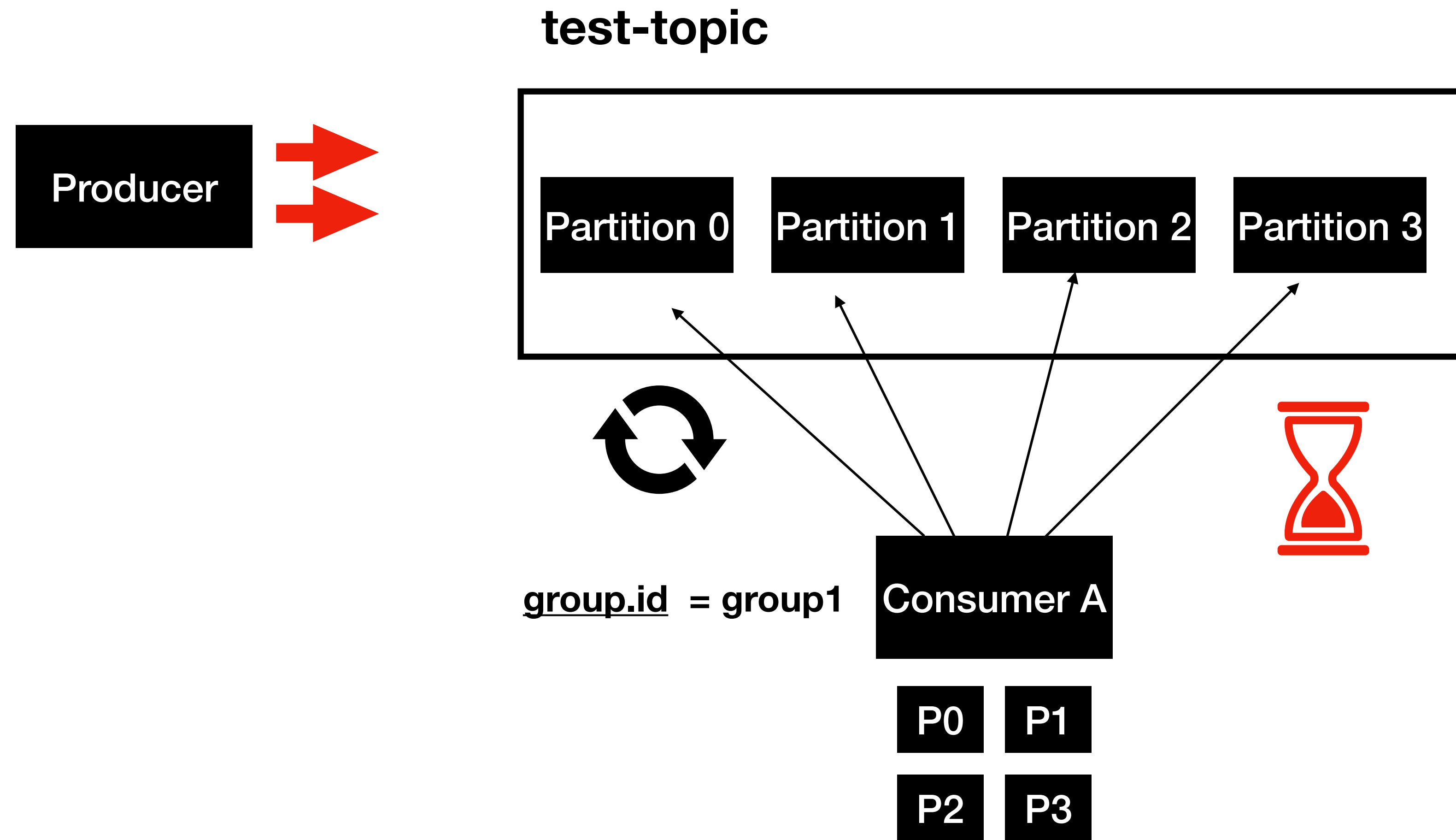
test-topic



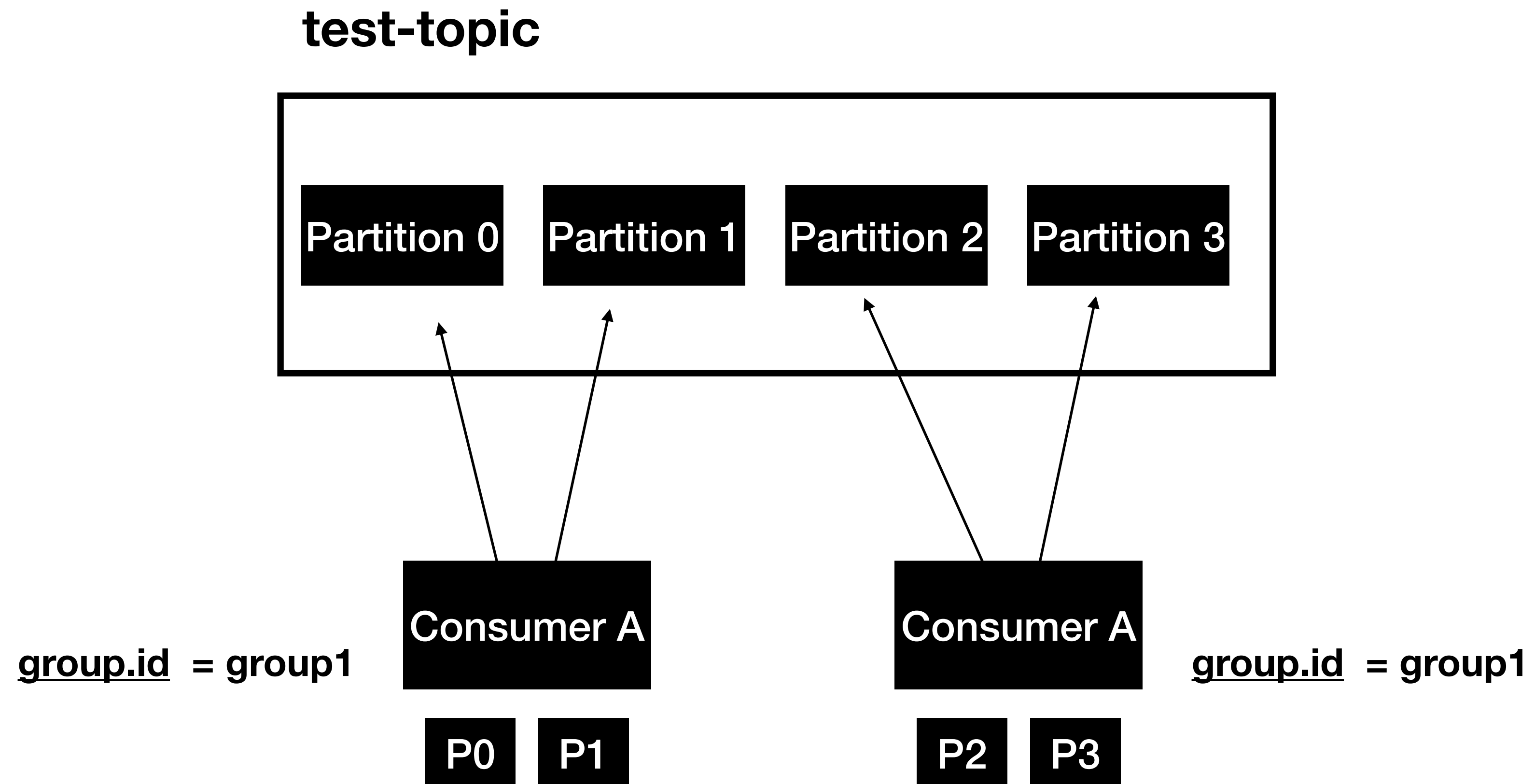
group.id = group1

Consumer 1

# Consumer Groups

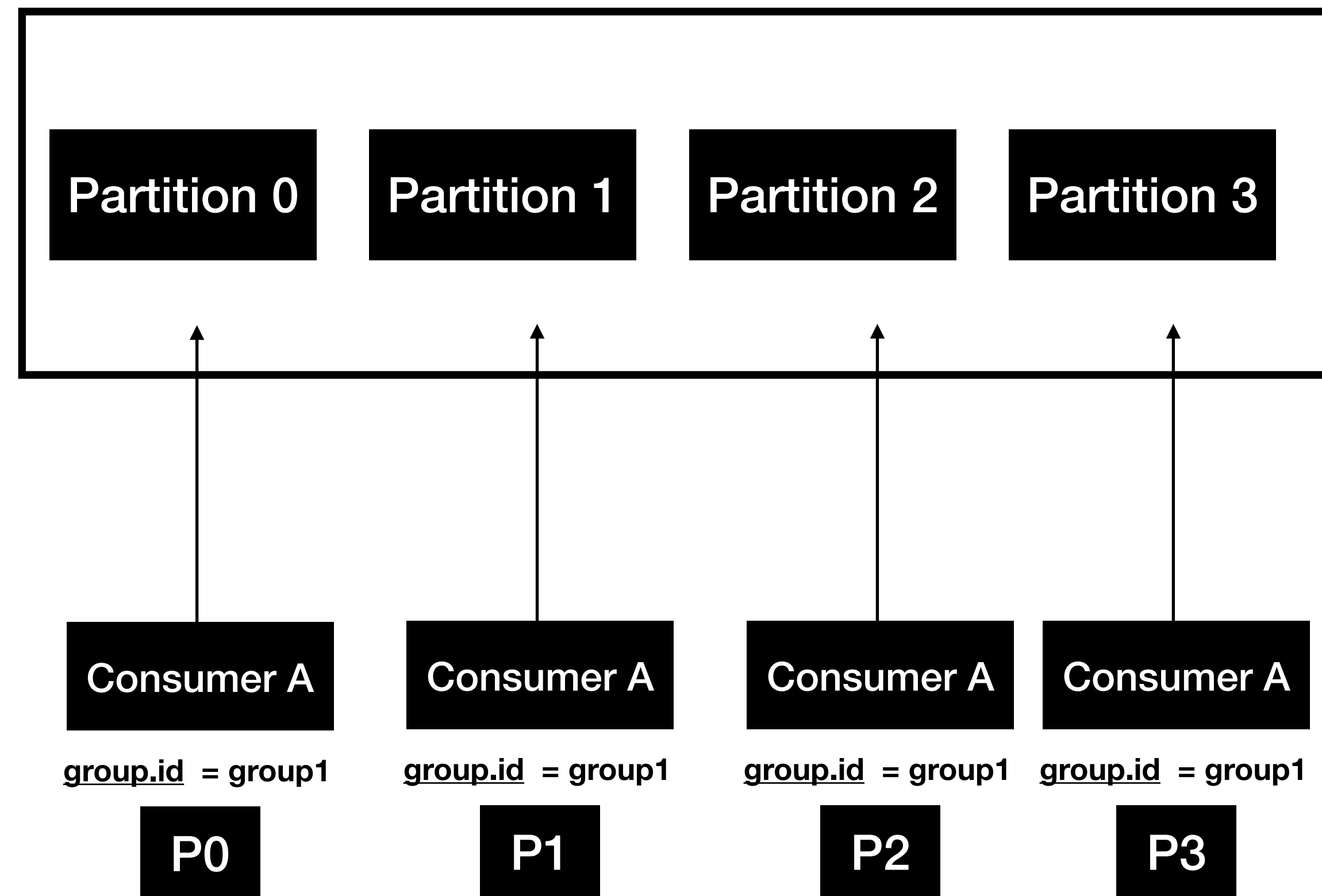


# Consumer Groups



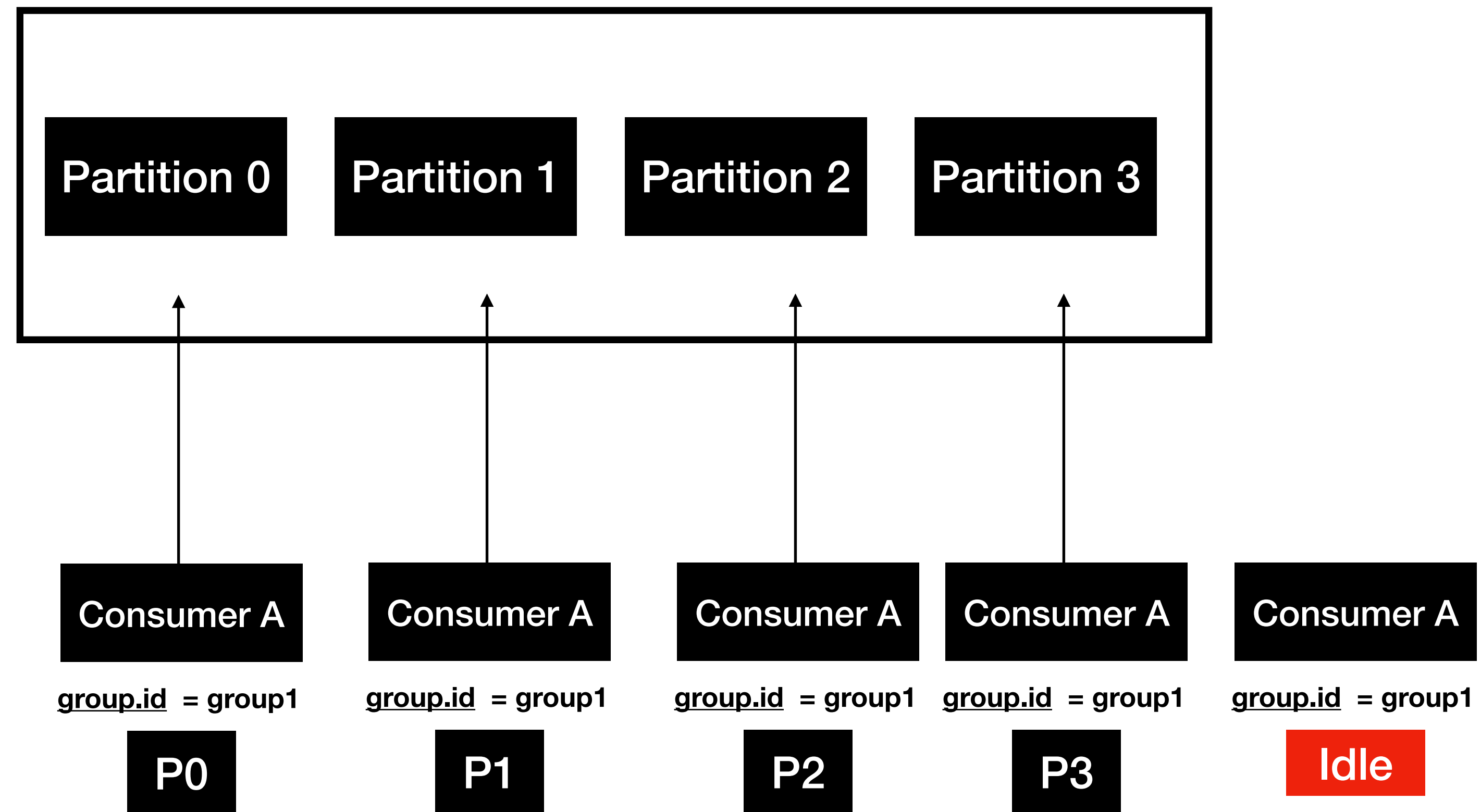
# Consumer Groups

test-topic



# Consumer Groups

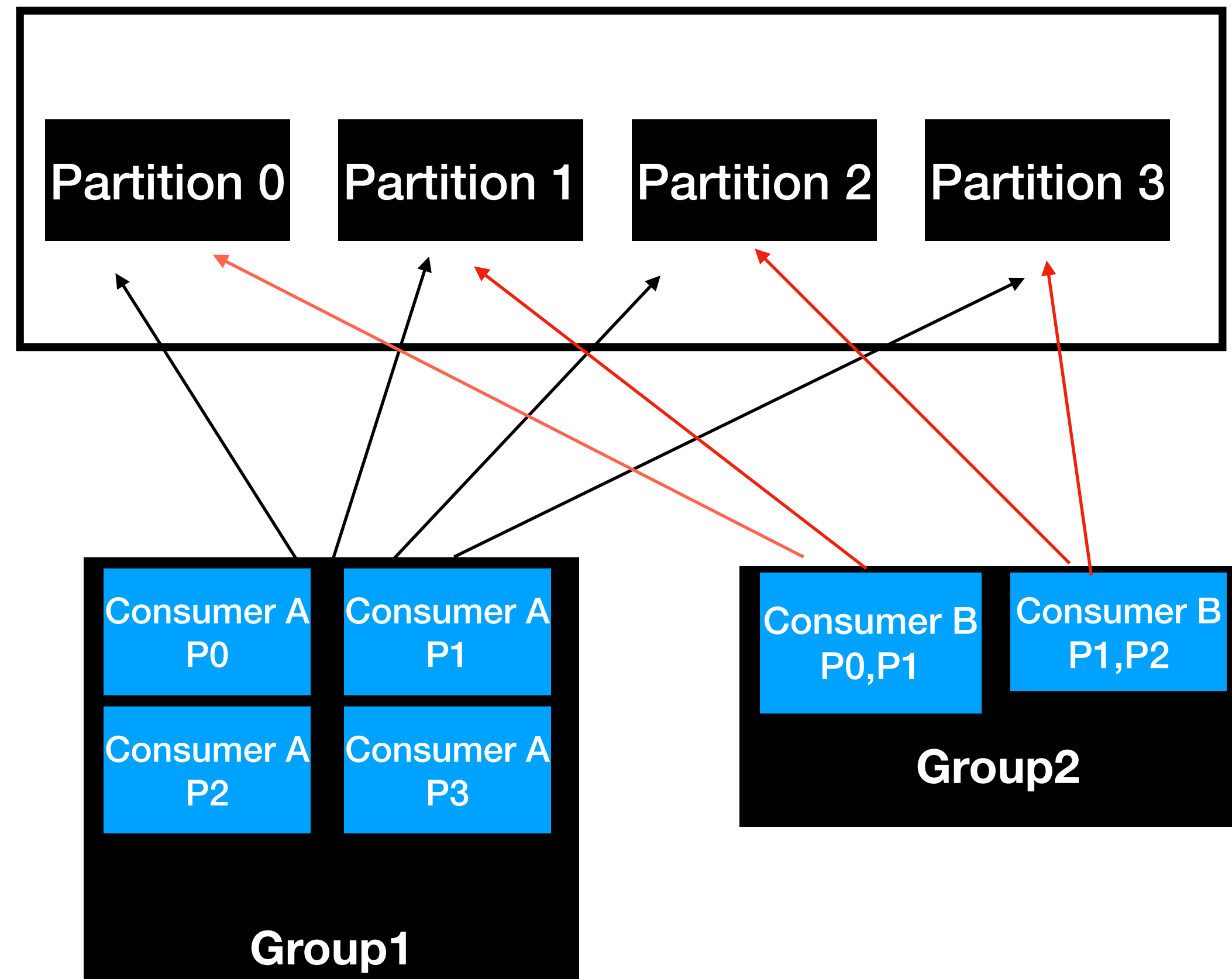
test-topic





# Consumer Groups

test-topic

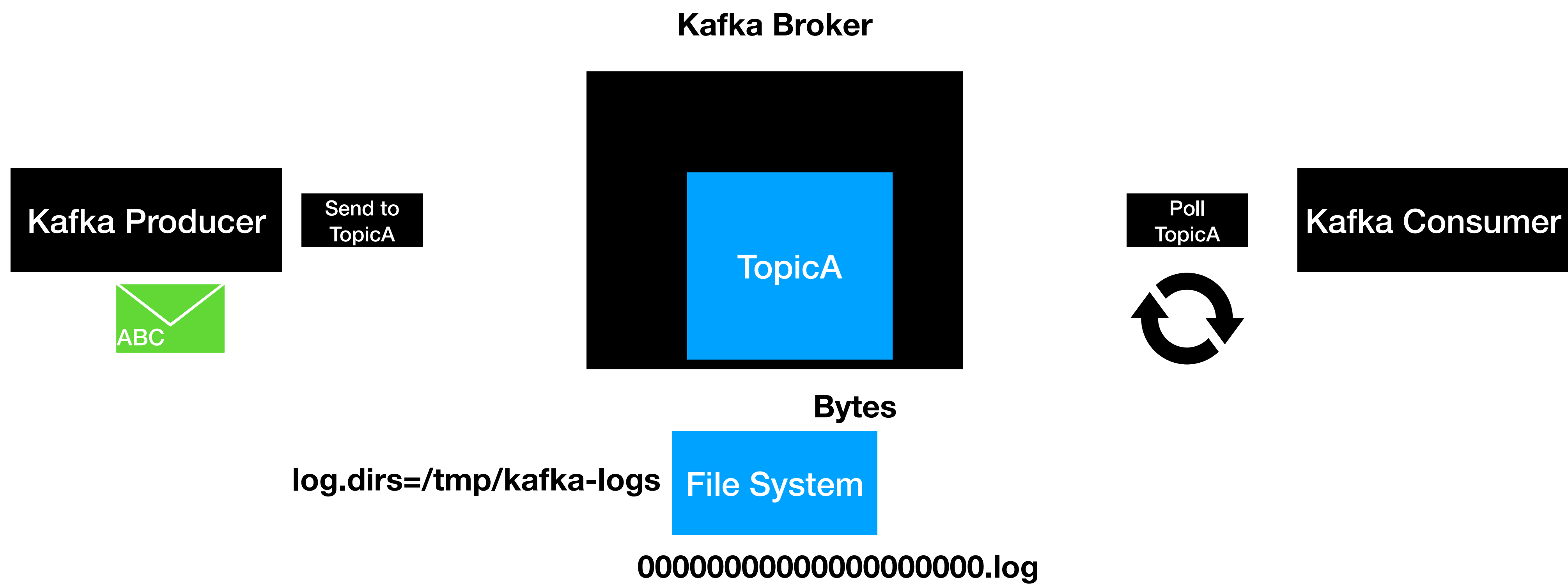


# Consumer Groups : Summary

- Consumer Groups are used for scalable message consumption
- Each different application will have a unique consumer group
- Who manages the consumer group?
  - Kafka Broker manages the consumer-groups
  - Kafka Broker acts as a Group Co-ordinator

# **Commit Log & Retention Policy**

# Commit Log



# Retention Policy

- Determines how long the message is retained ?
- Configured using the property **log.retention.hours** in **server.properties** file
- Default retention period is **168 hours** (7 days)

# **Kafka**

**as a**

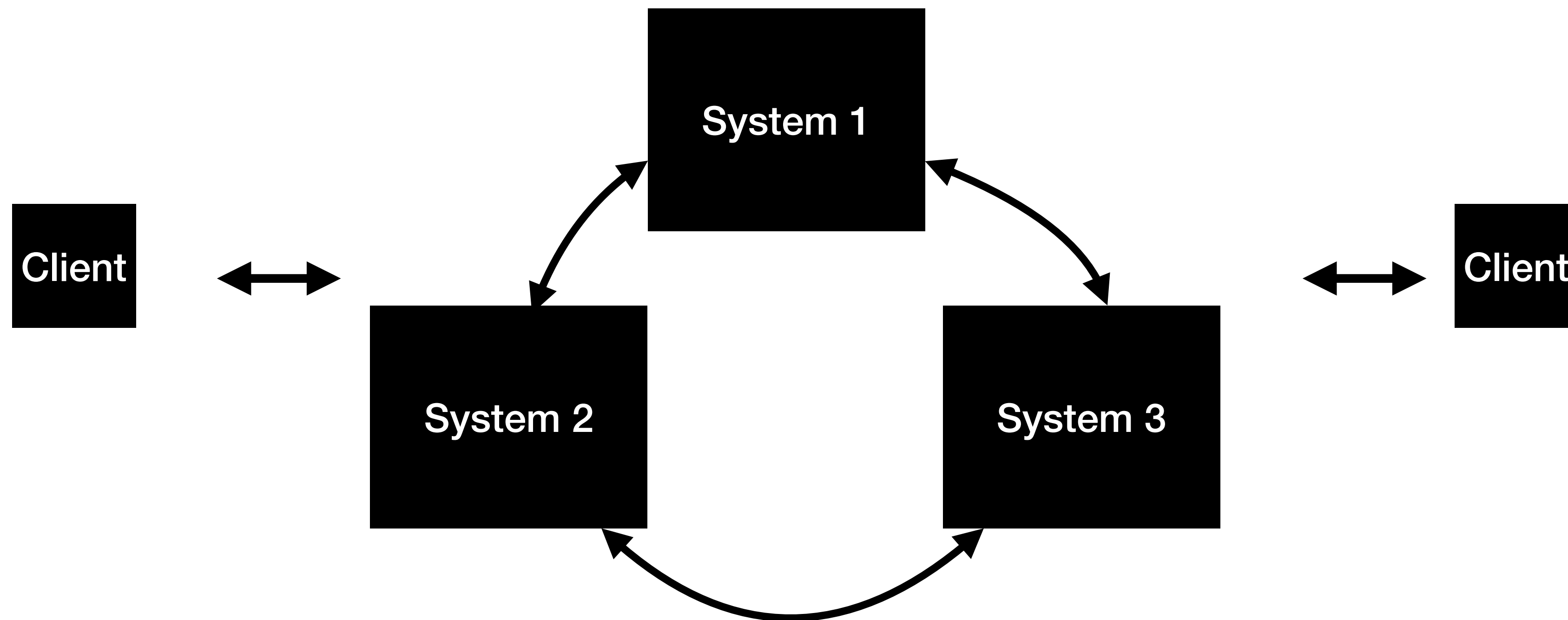
# **Distributed Streaming**

# **System**

**Apache Kafka® is *a distributed streaming platform***

# What is a Distributed System?

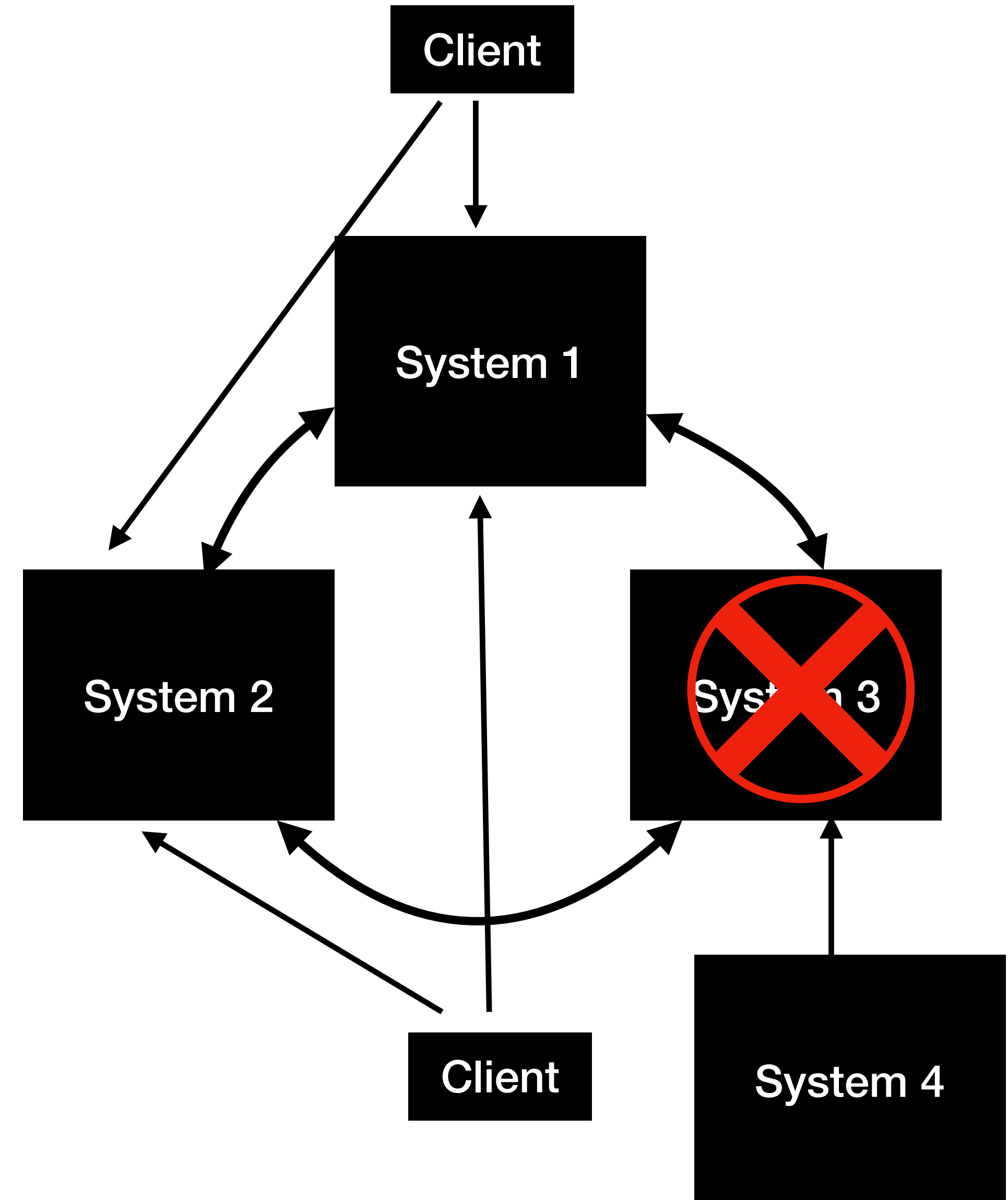
- Distributed systems are a collection of systems working together to deliver a value



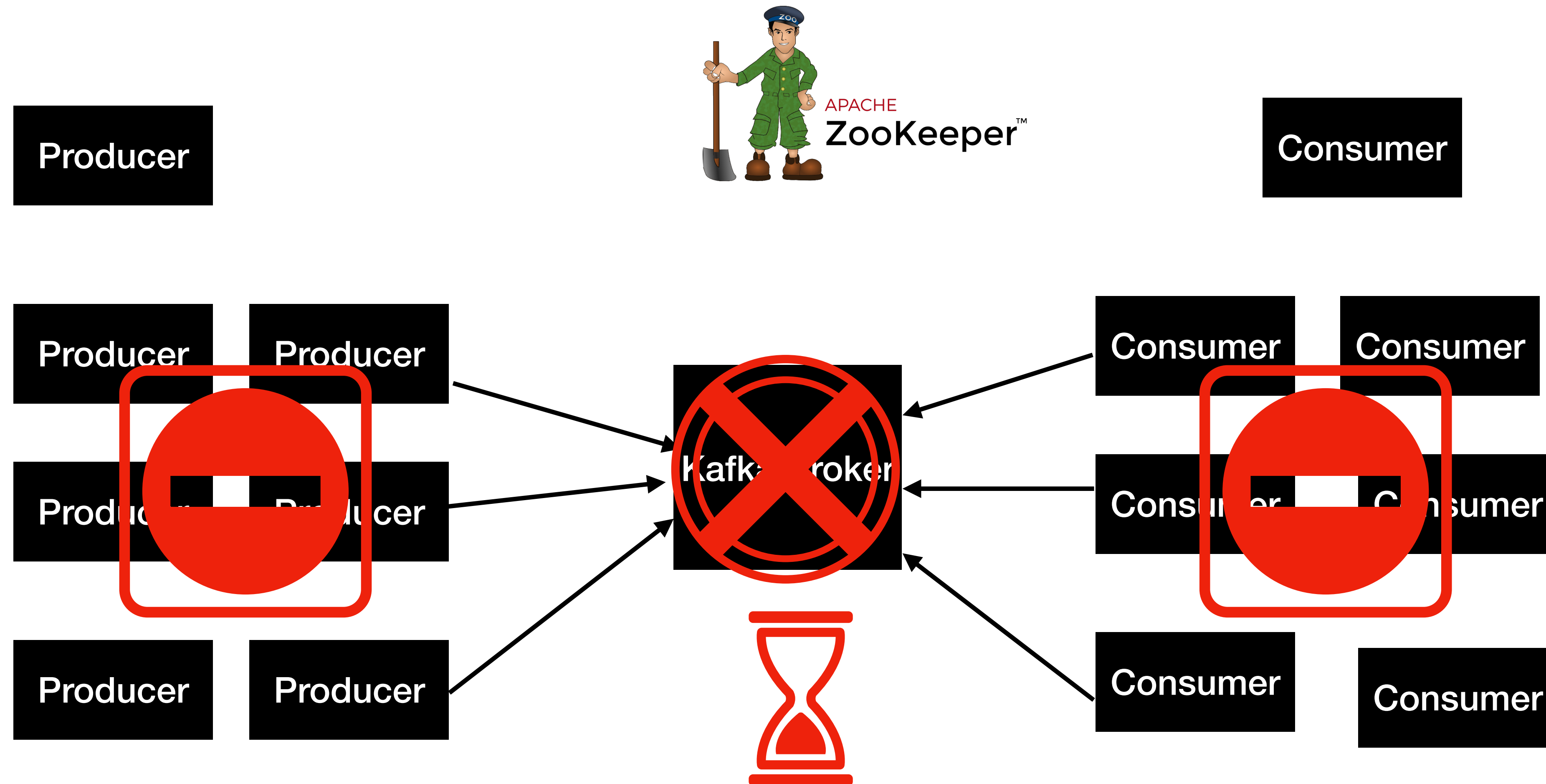


# Characteristics of Distributed System

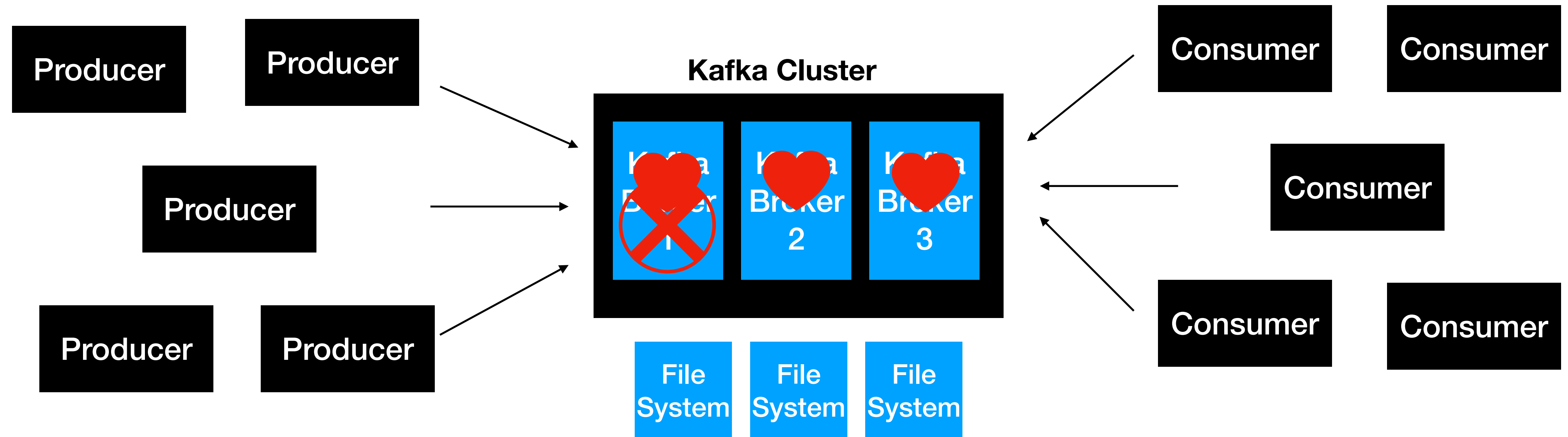
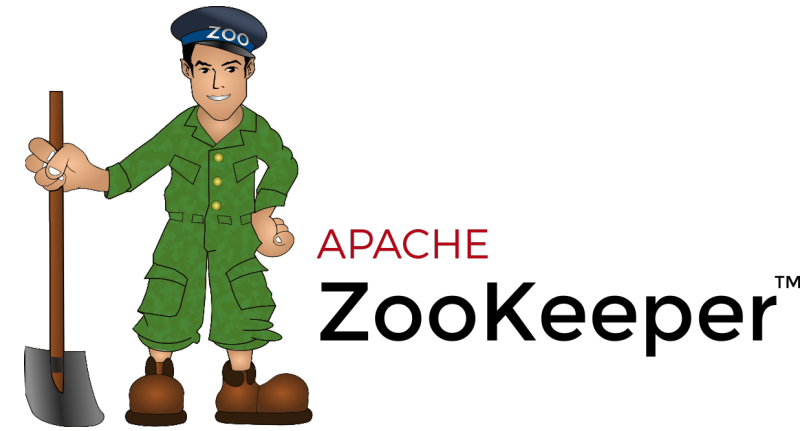
- Availability and Fault Tolerance
- Reliable Work Distribution
- Easily Scalable
- Handling Concurrency is fairly easy



# Kafka as a Distributed System



# Kafka as a Distributed System



- Client requests are distributed between brokers
- Easy to scale by adding more brokers based on the need
- Handles data loss using Replication

# **SetUp Kafka Cluster Using Three Brokers**

# Start Kafka Broker

```
./kafka-server-start.sh ../config/server.properties
```

# Setting up Kafka Cluster

- New **server.properties** files with the new broker details.

```
broker.id=<unique-broker-d>  
listeners=PLAINTEXT://localhost:<unique-port>  
log.dirs=/tmp/<unique-kafka-folder>  
auto.create.topics.enable=false(optional)
```

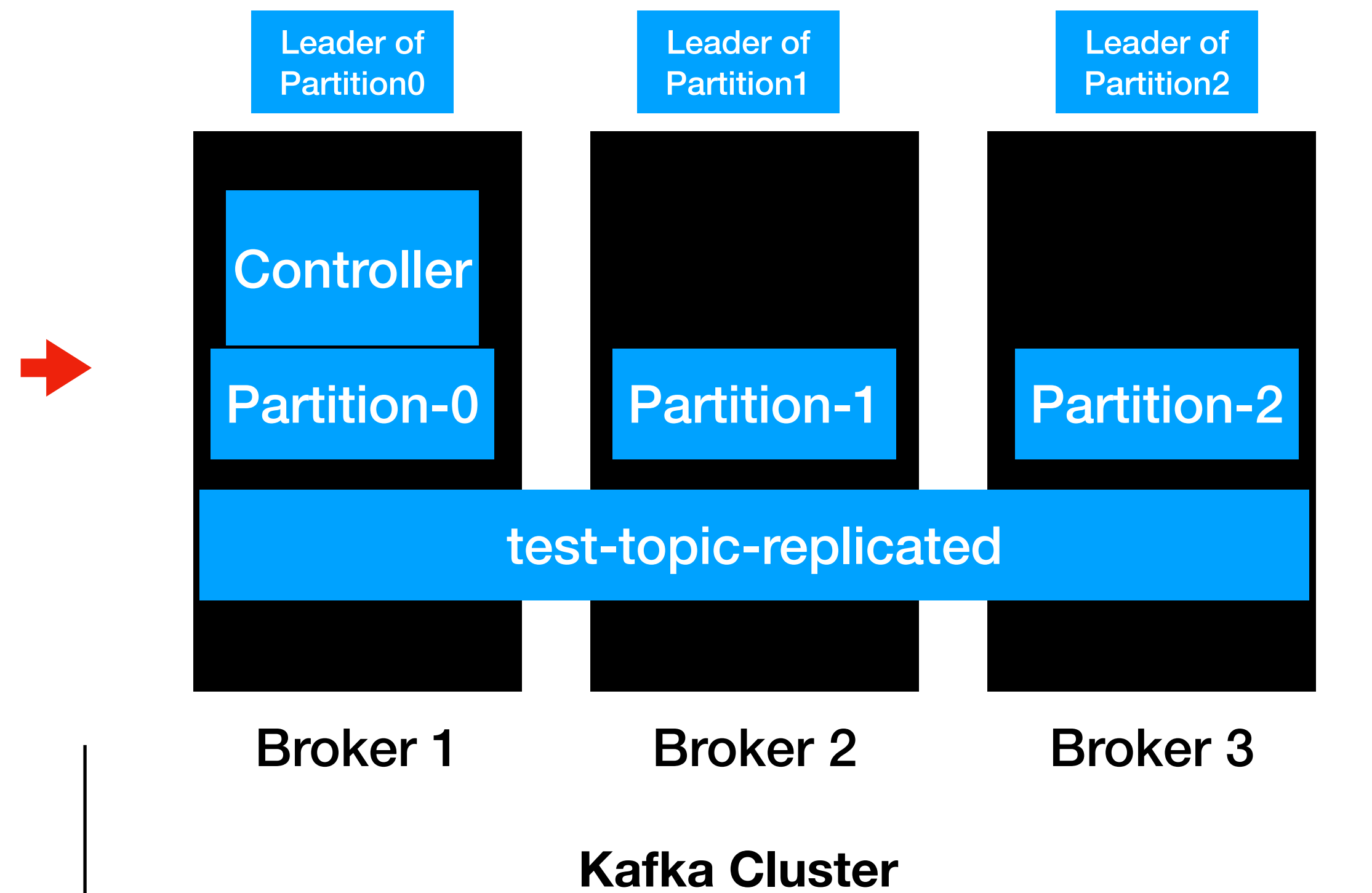
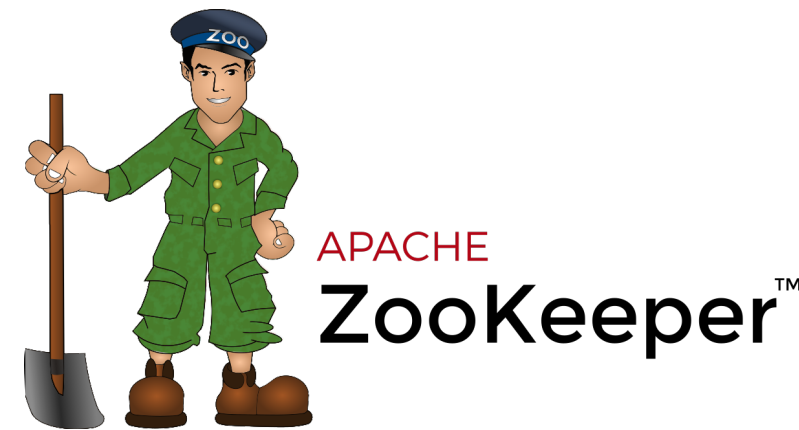
**Example:** **server-1.properties**

```
broker.id=1  
listeners=PLAINTEXT://localhost:9093  
log.dirs=/tmp/kafka-logs-1  
auto.create.topics.enable=false(optional)
```

# **How Kafka Distributes the Client Requests?**

# How Topics are distributed?

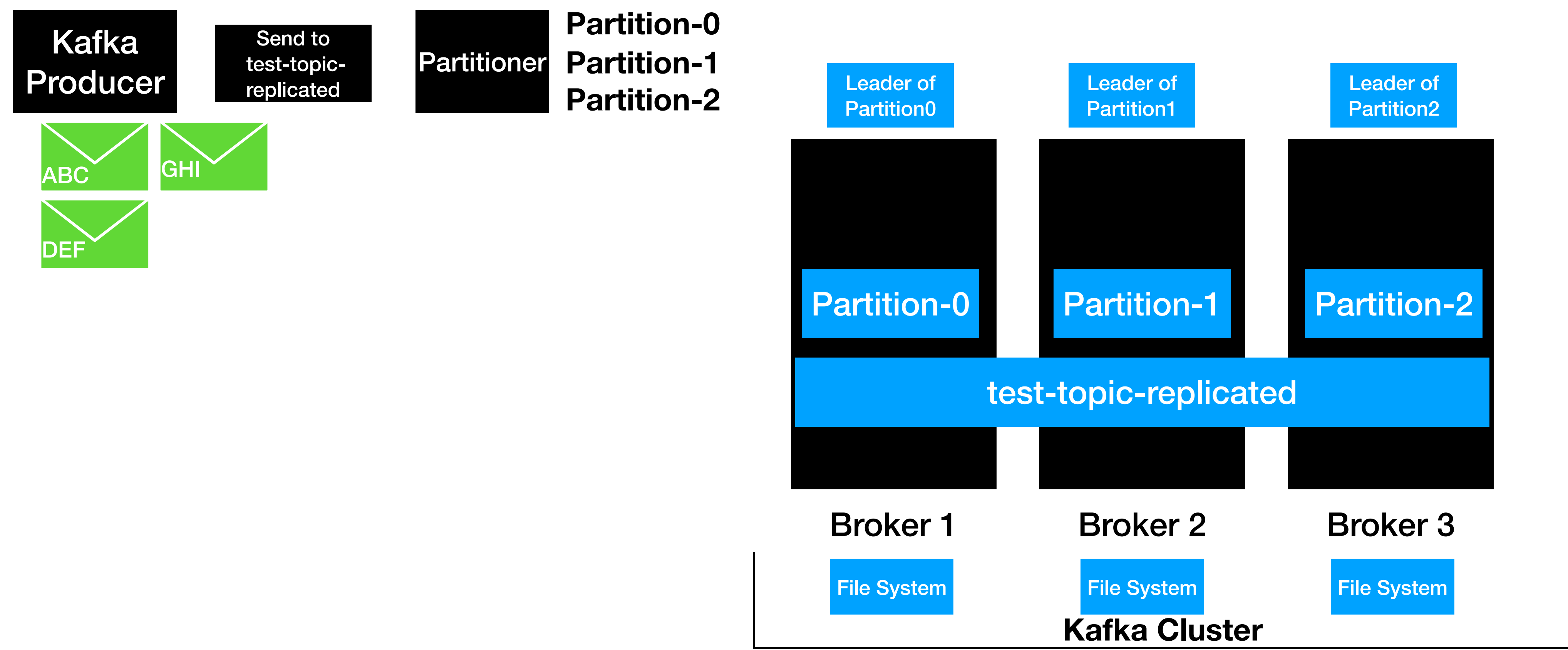
```
./kafka-topics.sh -  
-create --topic test-topic-replicated  
-zookeeper localhost:2181  
--replication-factor 3  
--partitions 3
```





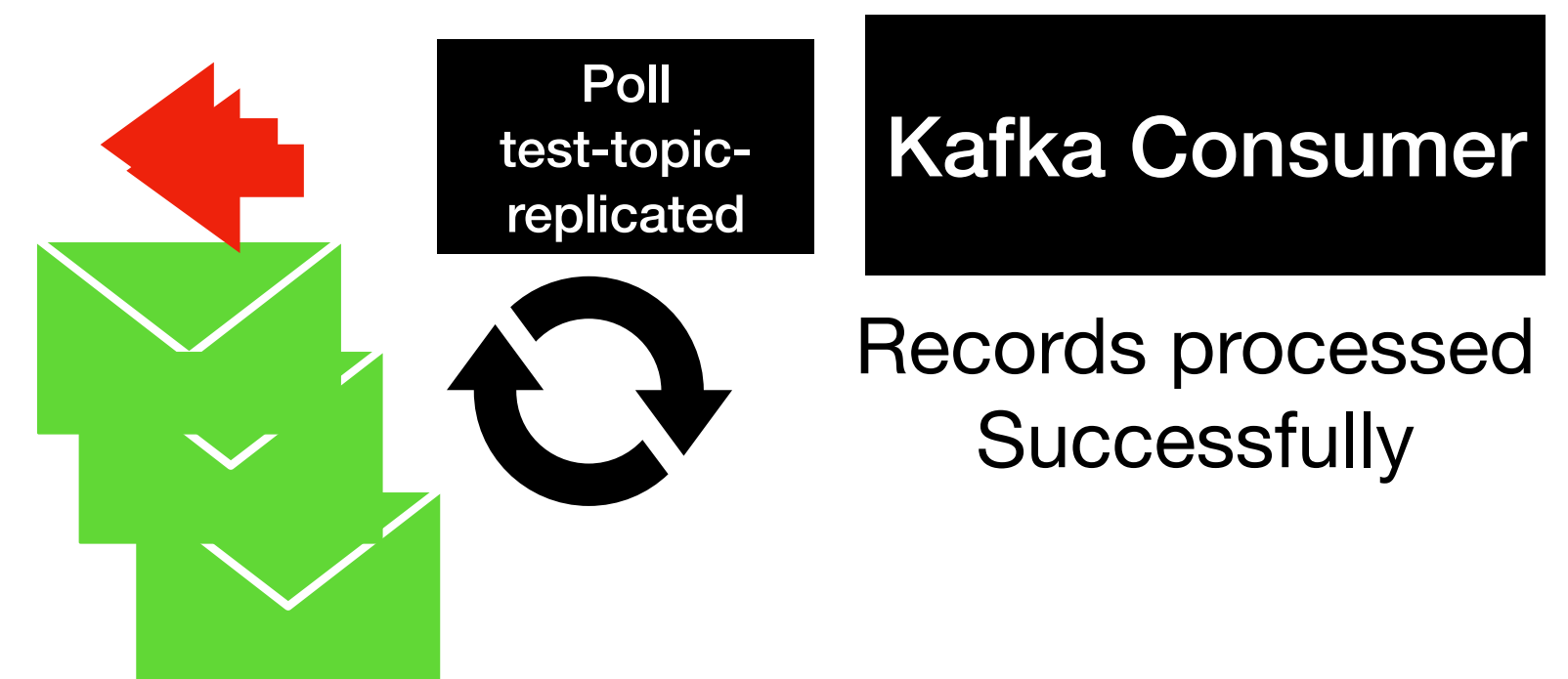
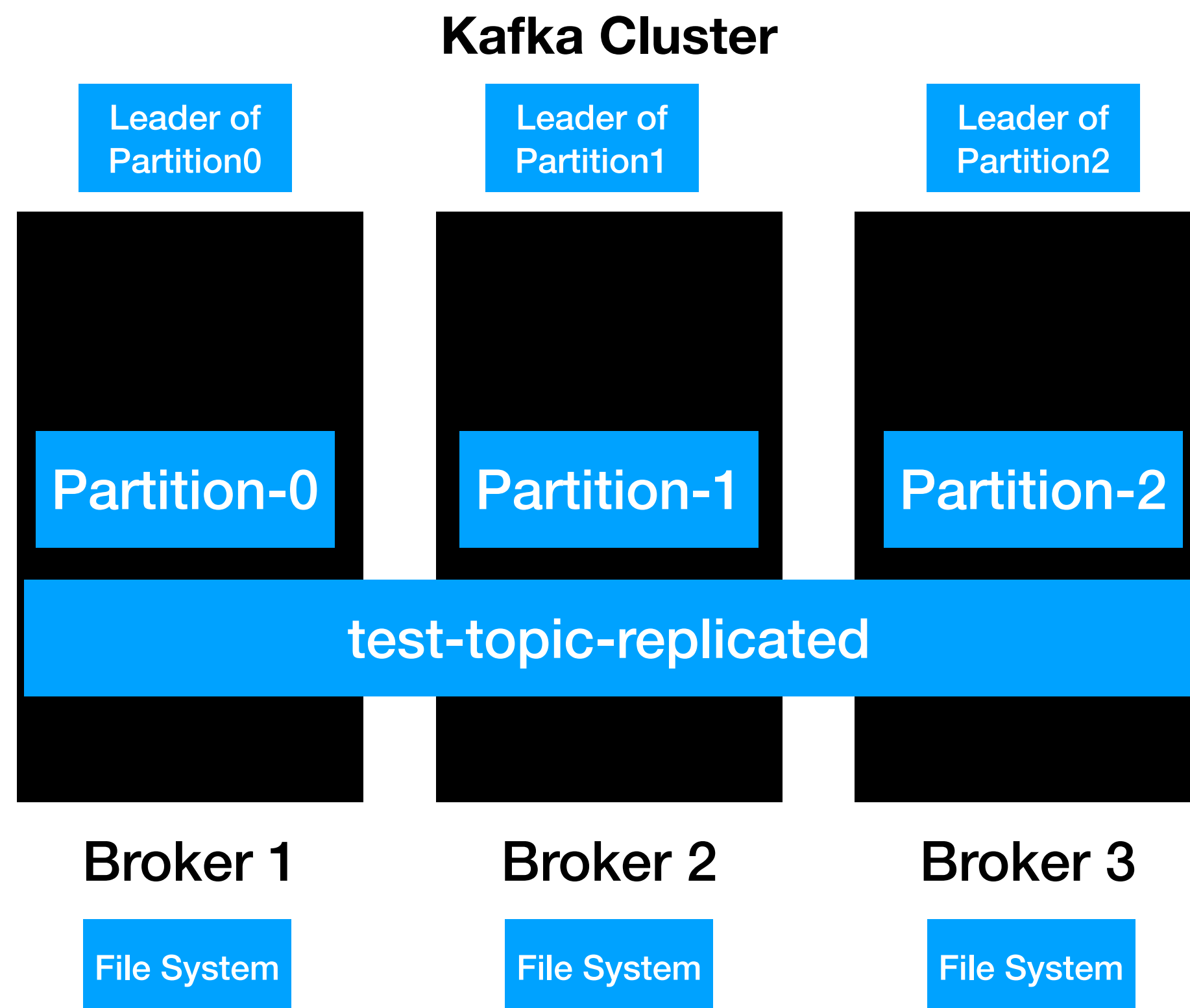
# How Kafka Distributes Client Requests?

## Kafka Producer



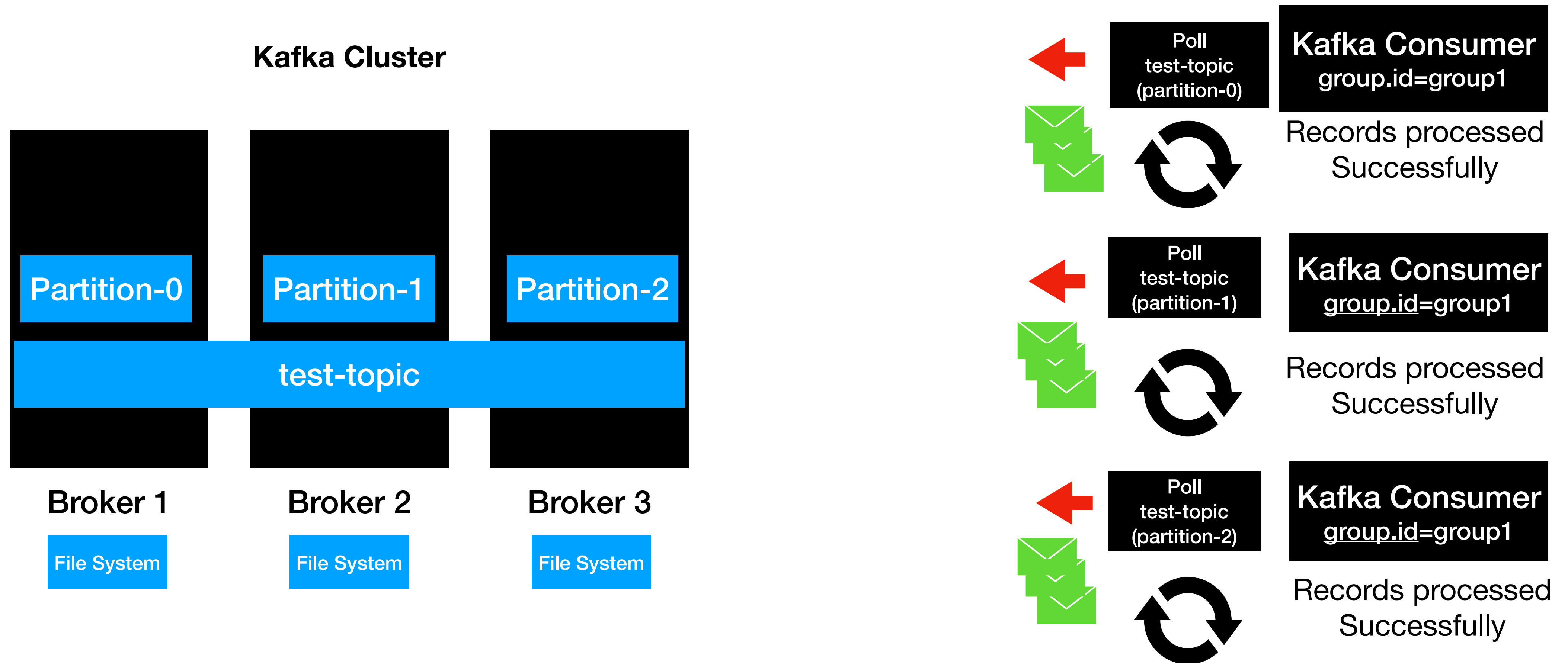
# How Kafka Distributes Client Requests?

## Kafka Consumer



# How Kafka Distributes Client Requests?

## Kafka Consumer Groups

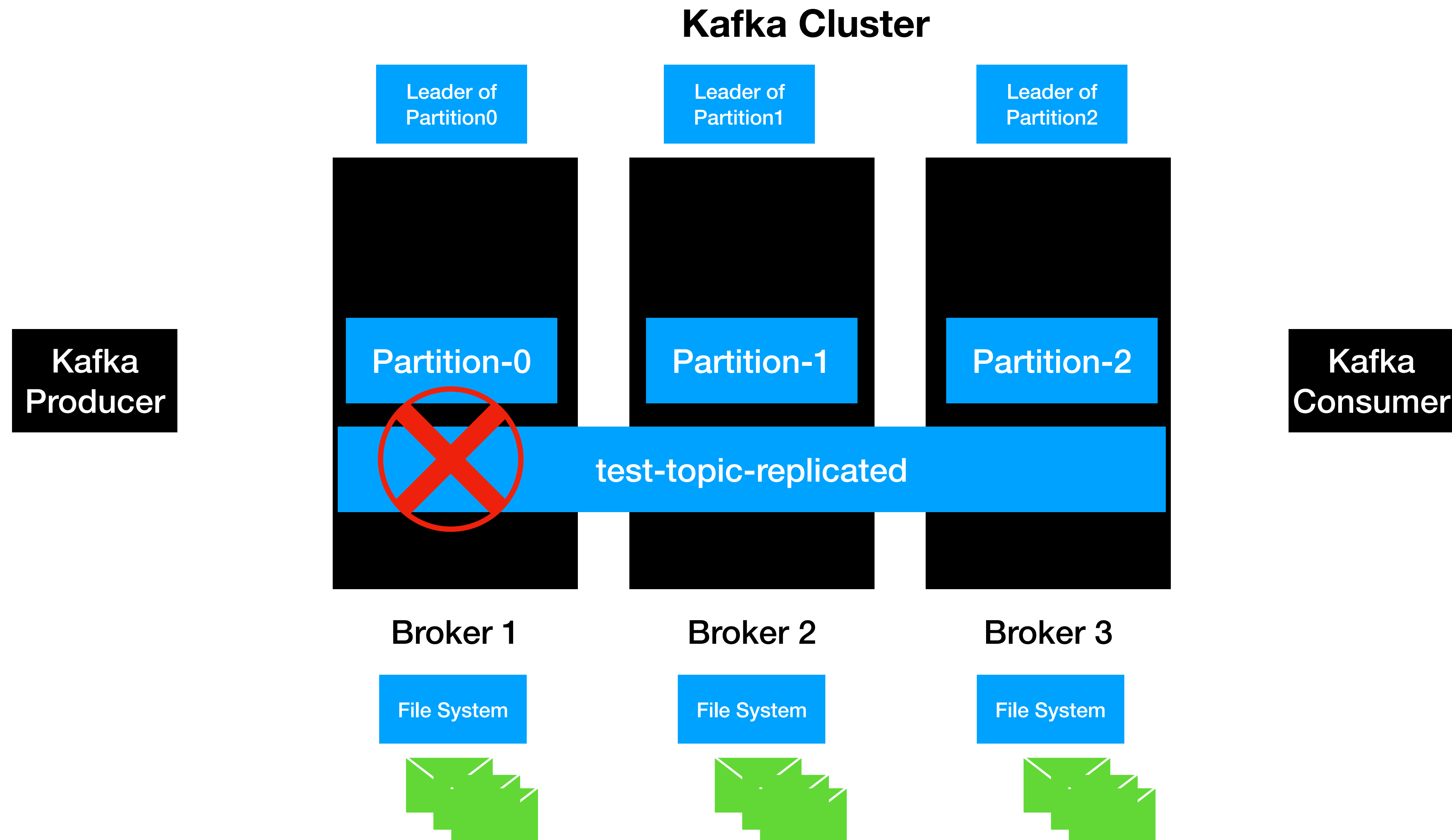


# Summary : How Kafka Distributes the Client Requests?

- Partition leaders are assigned during topic Creation
- Clients will only invoke leader of the partition to produce and consume data
  - Load is evenly distributed between the brokers

# **How Kafka handles Data Loss ?**

# How Kafka handles Data loss?



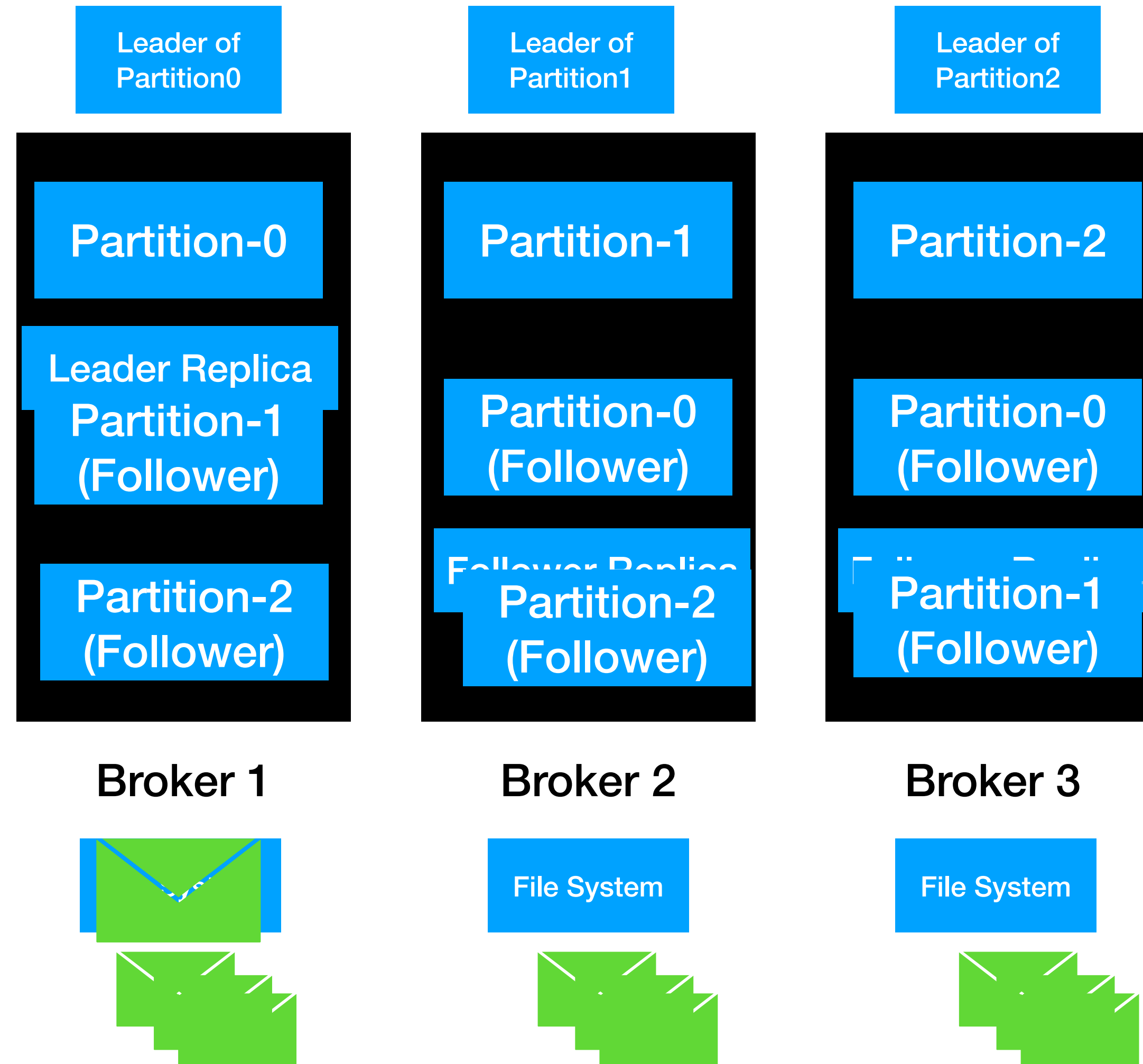
# Replication

```
./kafka-topics.sh -  
-create --topic test-topic-replicated  
-zookeeper localhost:2181  
--replication-factor 3  
--partitions 3
```

# Replication

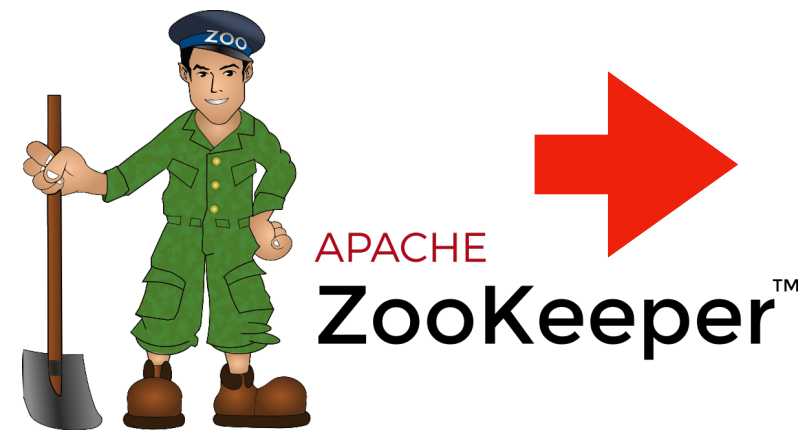
## Kafka Cluster

Replication factor = 3



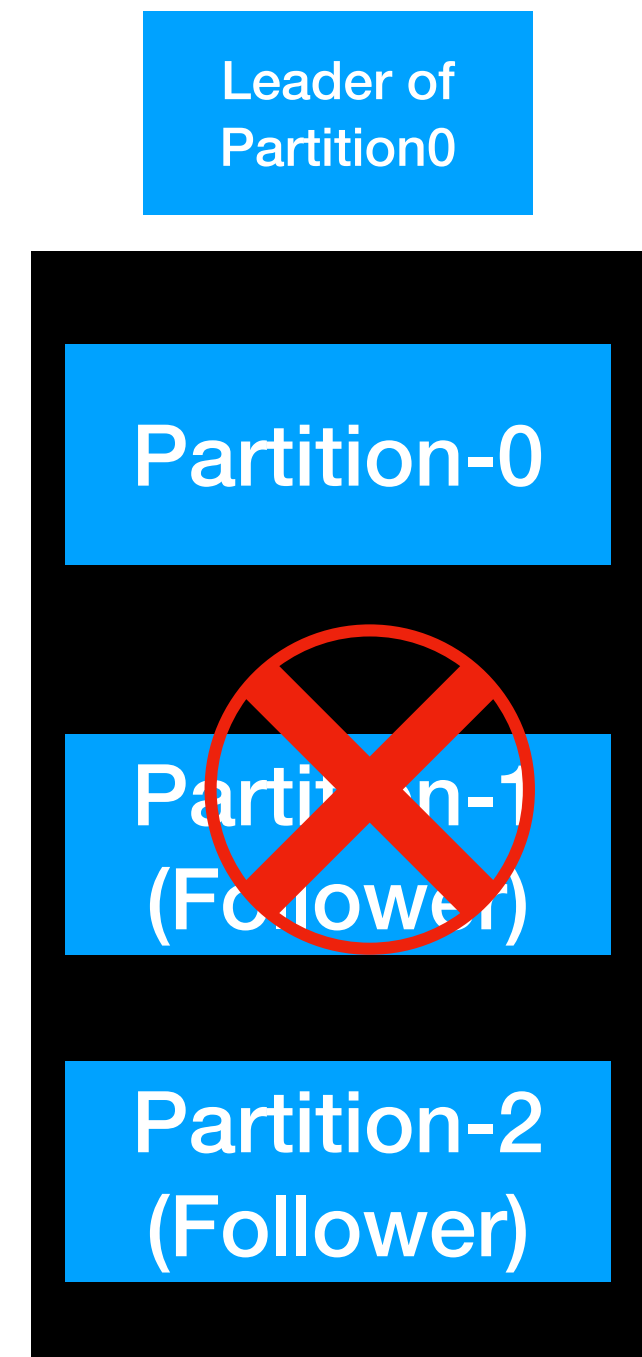
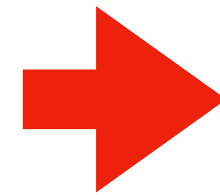


# Replication



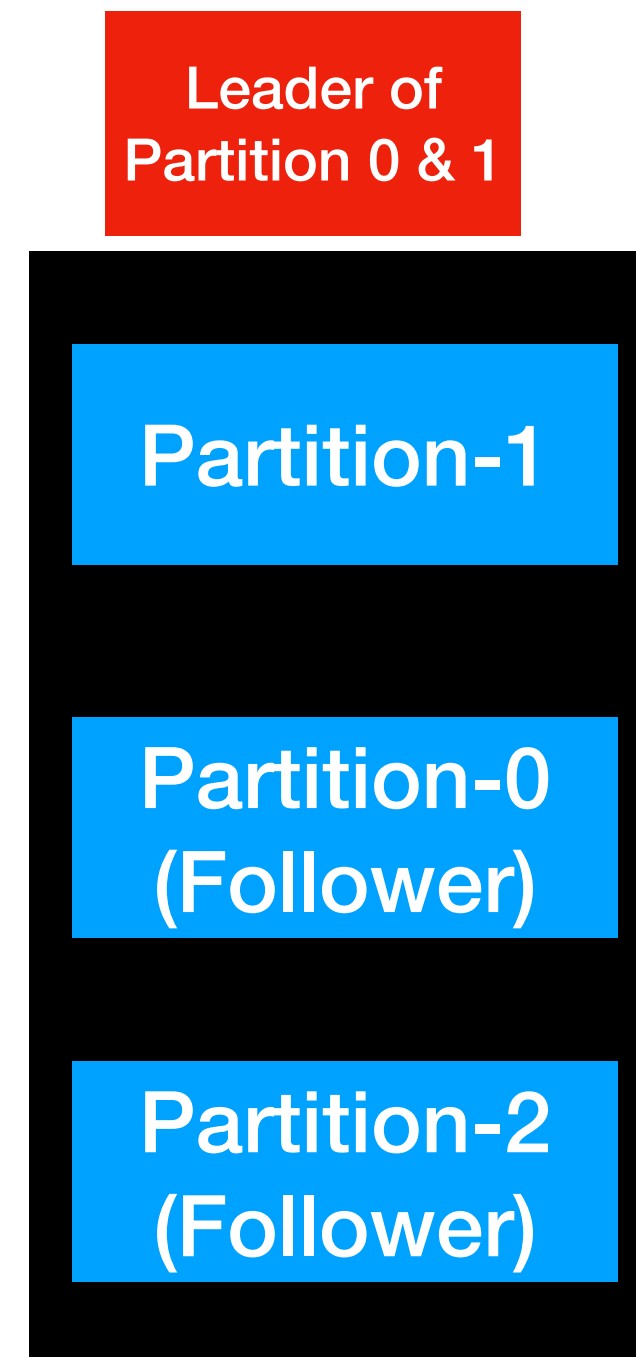
## Kafka Cluster

Kafka  
Producer



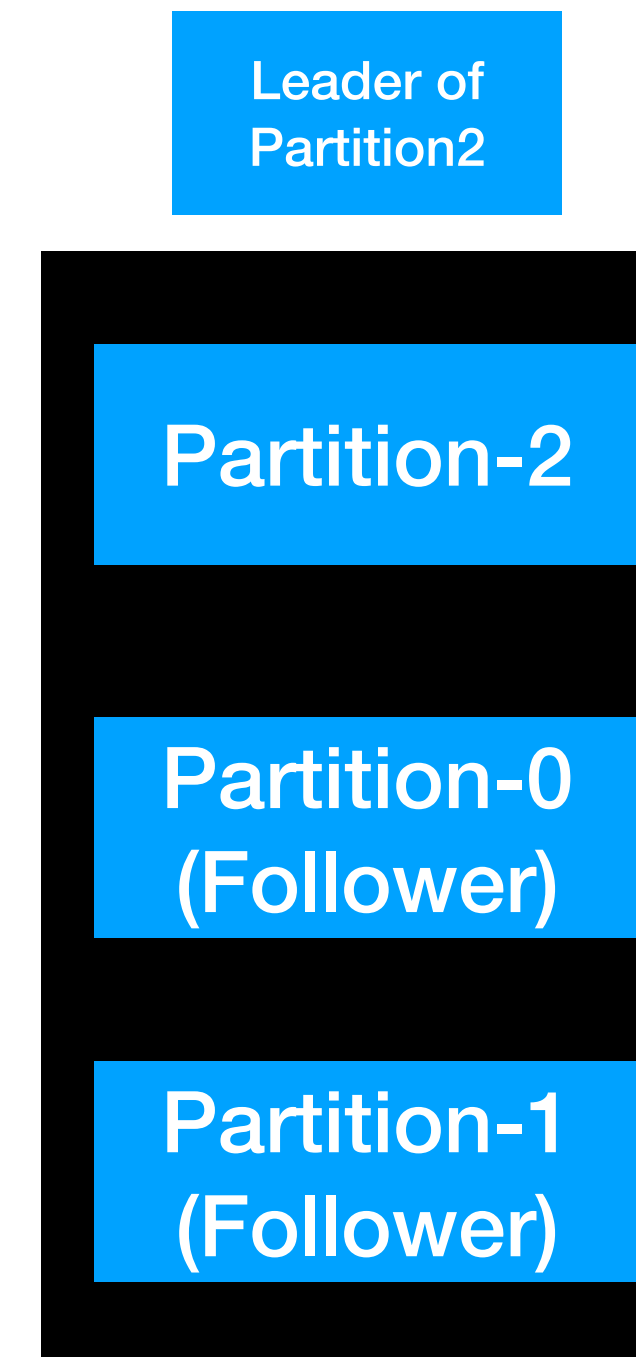
Broker 1

File System



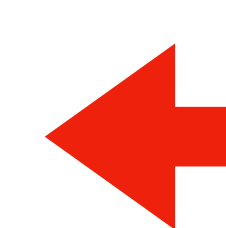
Broker 2

File System



Broker 3

File System



Kafka  
Consumer

# In-Sync Replica(ISR)

- Represents the number of replica in sync with each other in the cluster
  - Includes both **leader** and **follower** replica
- Recommended value is always greater than 1
- Ideal value is **ISR == Replication Factor**
- This can be controlled by **min.insync.replicas** property
  - It can be set at the **broker** or **topic** level

# **Fault Tolerance & Robustness**

# Application Overview



# Library Inventory





# Library Inventory Flow

Librarian

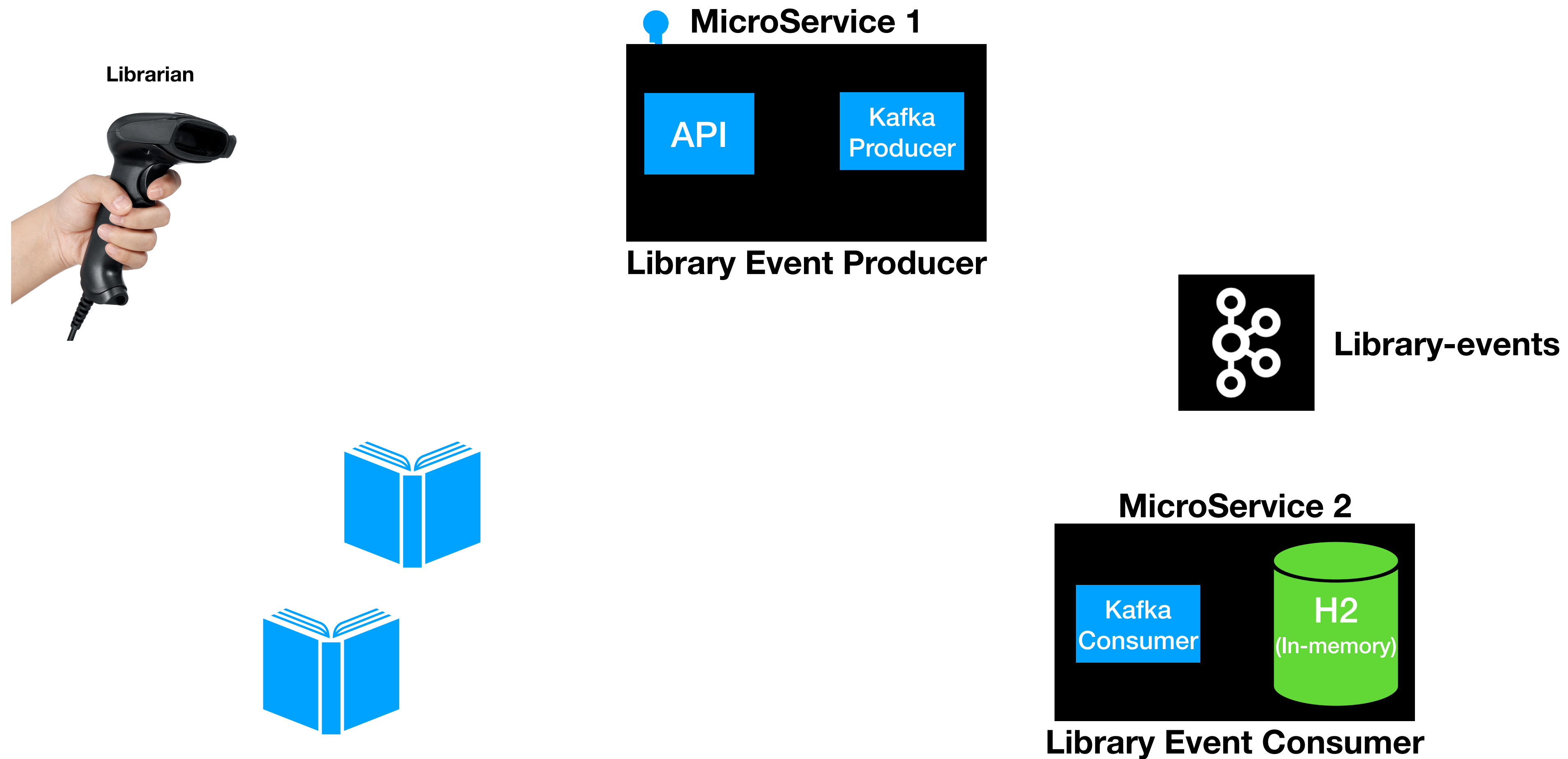


Library Inventory

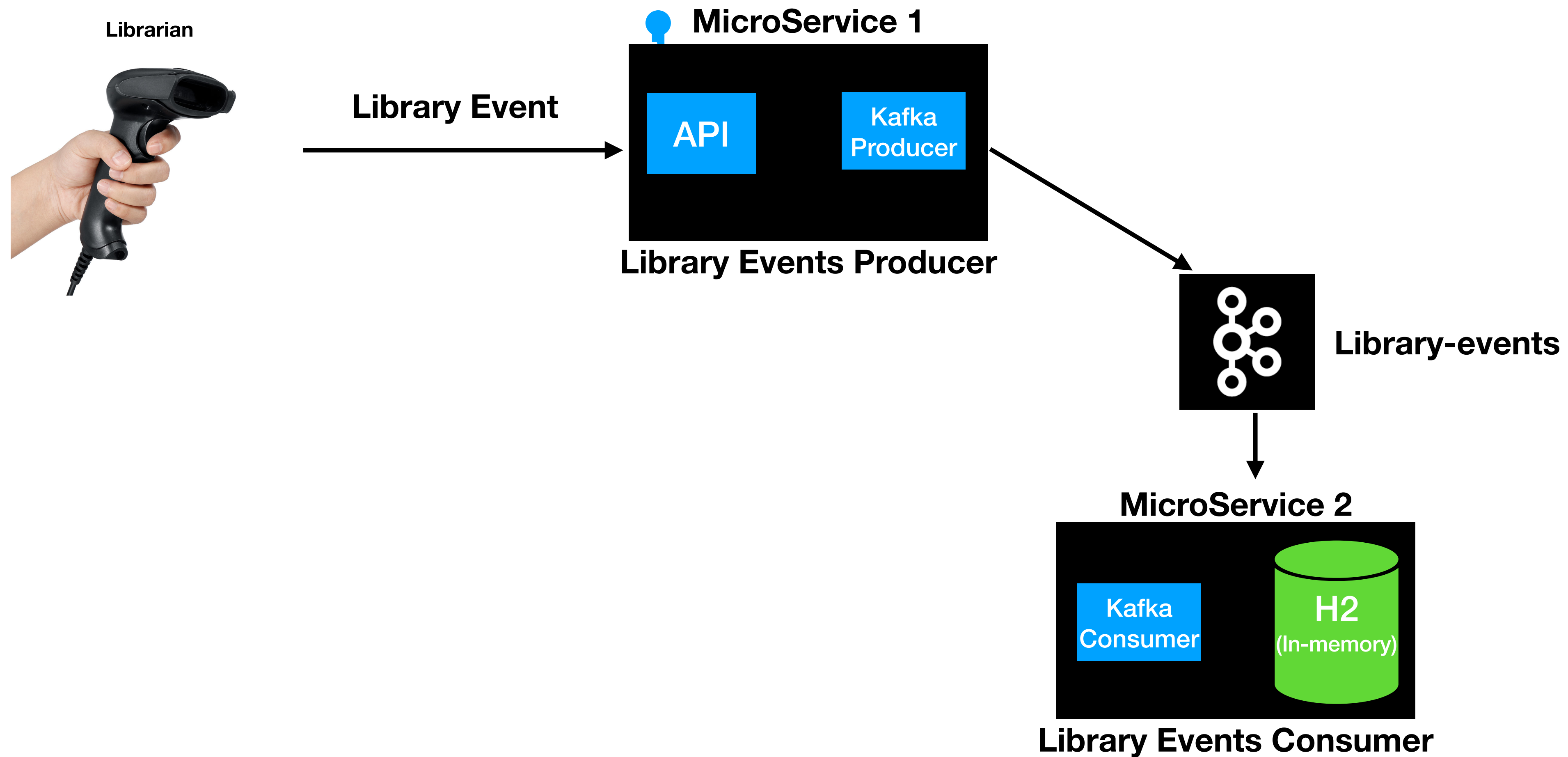




# Library Inventory Architecture

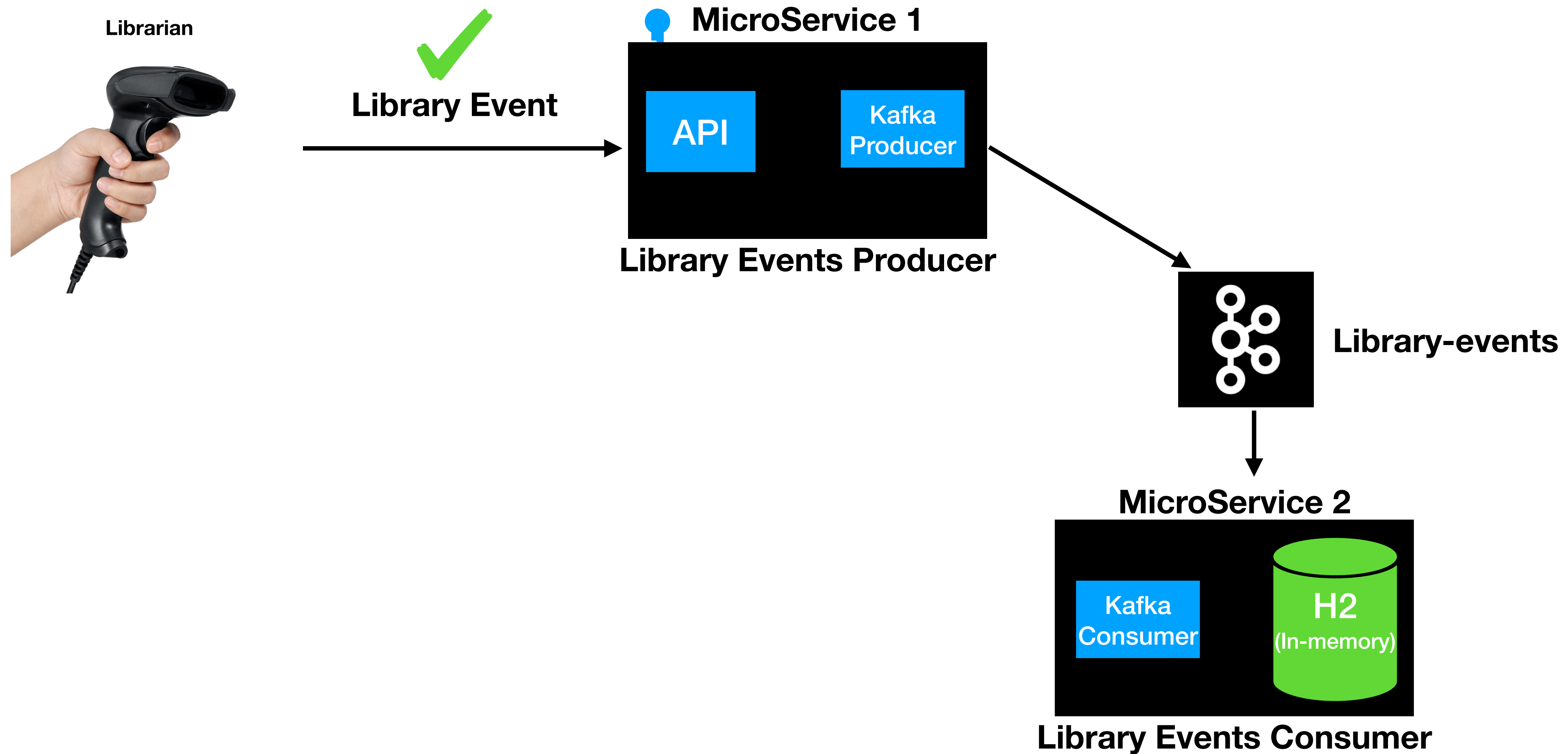


# Library Inventory Domain

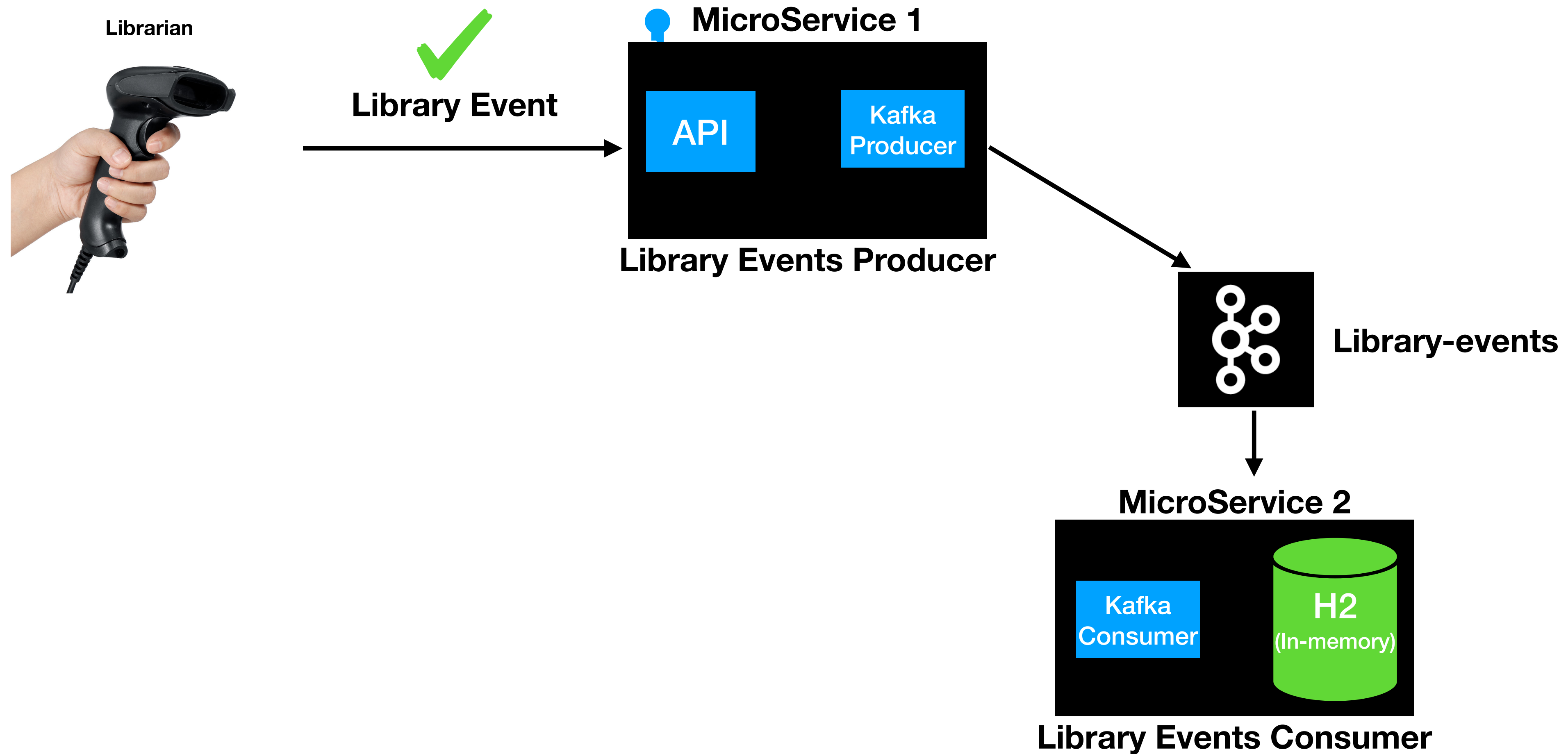




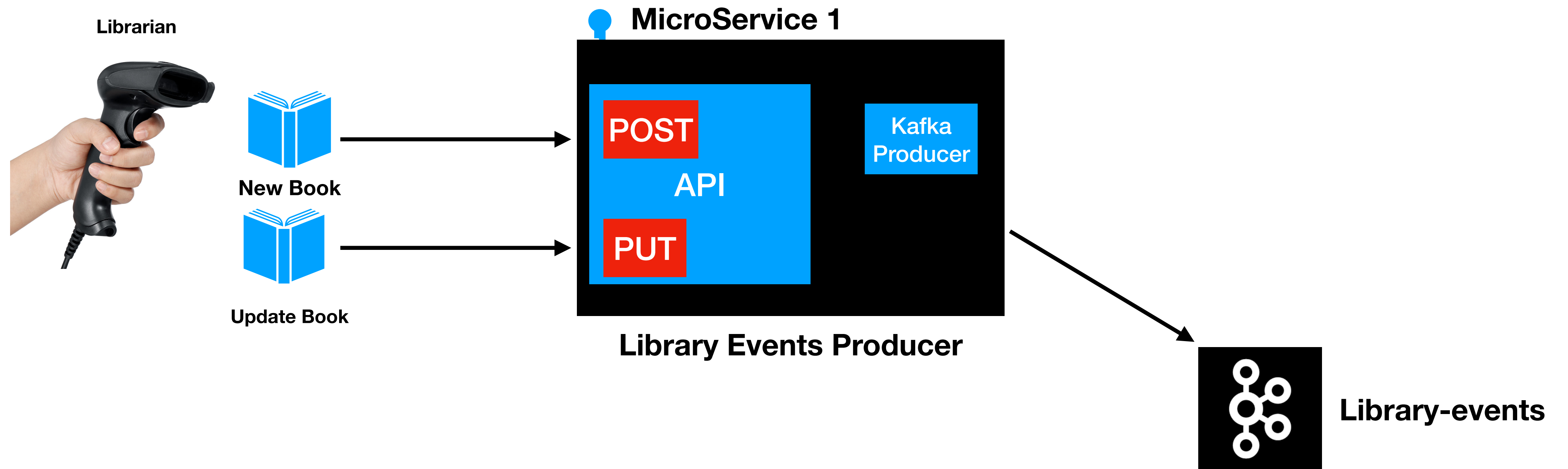
# Library Event Domain



# Library Event Domain



# Library Events Producer API



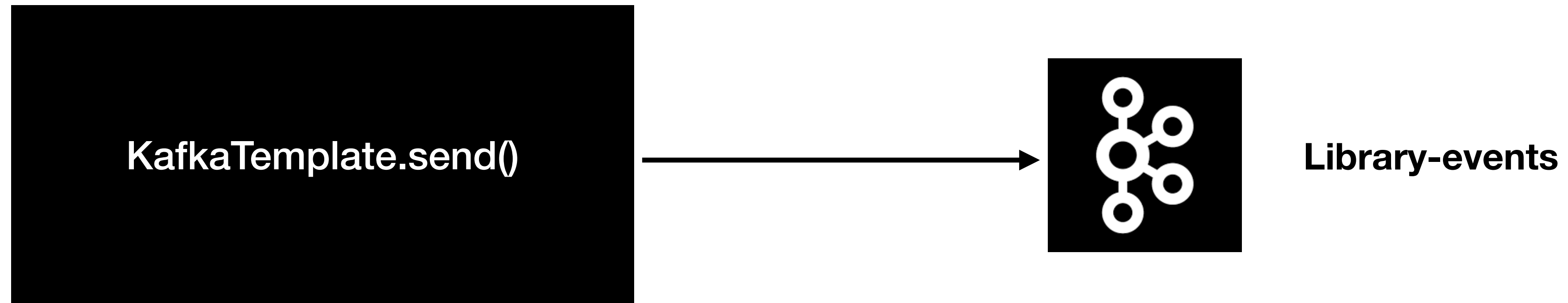
# KafkaTemplate

Kafka Producer in Spring

# KafkaTemplate

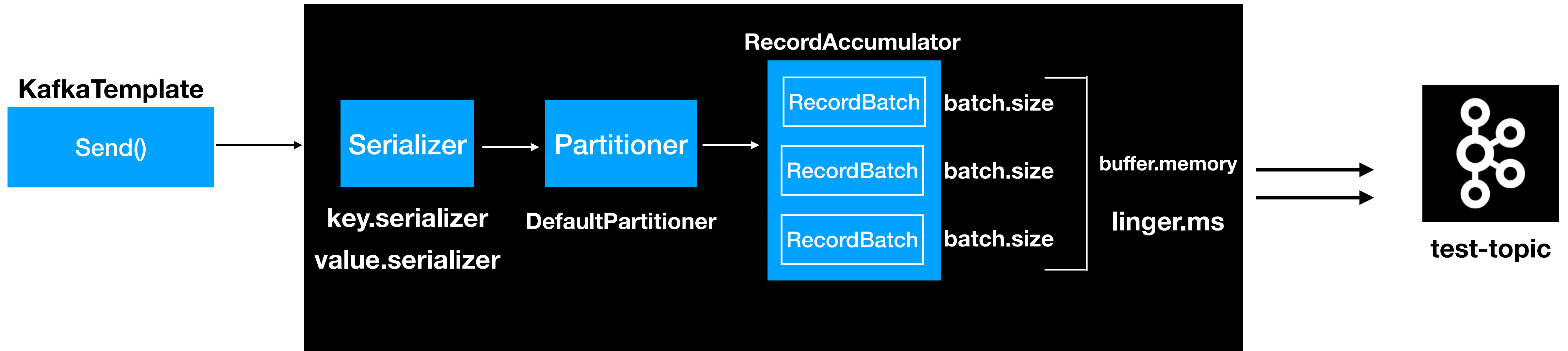
- Produce records in to Kafka Topic
  - Similar to JdbcTemplate for DB

# How KafkaTemplate Works ?



# KafkaTemplate.send()

## Behind the Scenes



# Configuring KafkaTemplate

## Mandatory Values:

**bootstrap-servers:** localhost:9092,localhost:9093,localhost:9094

**key-serializer:** org.apache.kafka.common.serialization.IntegerSerializer

**value-serializer:** org.apache.kafka.common.serialization.StringSerializer

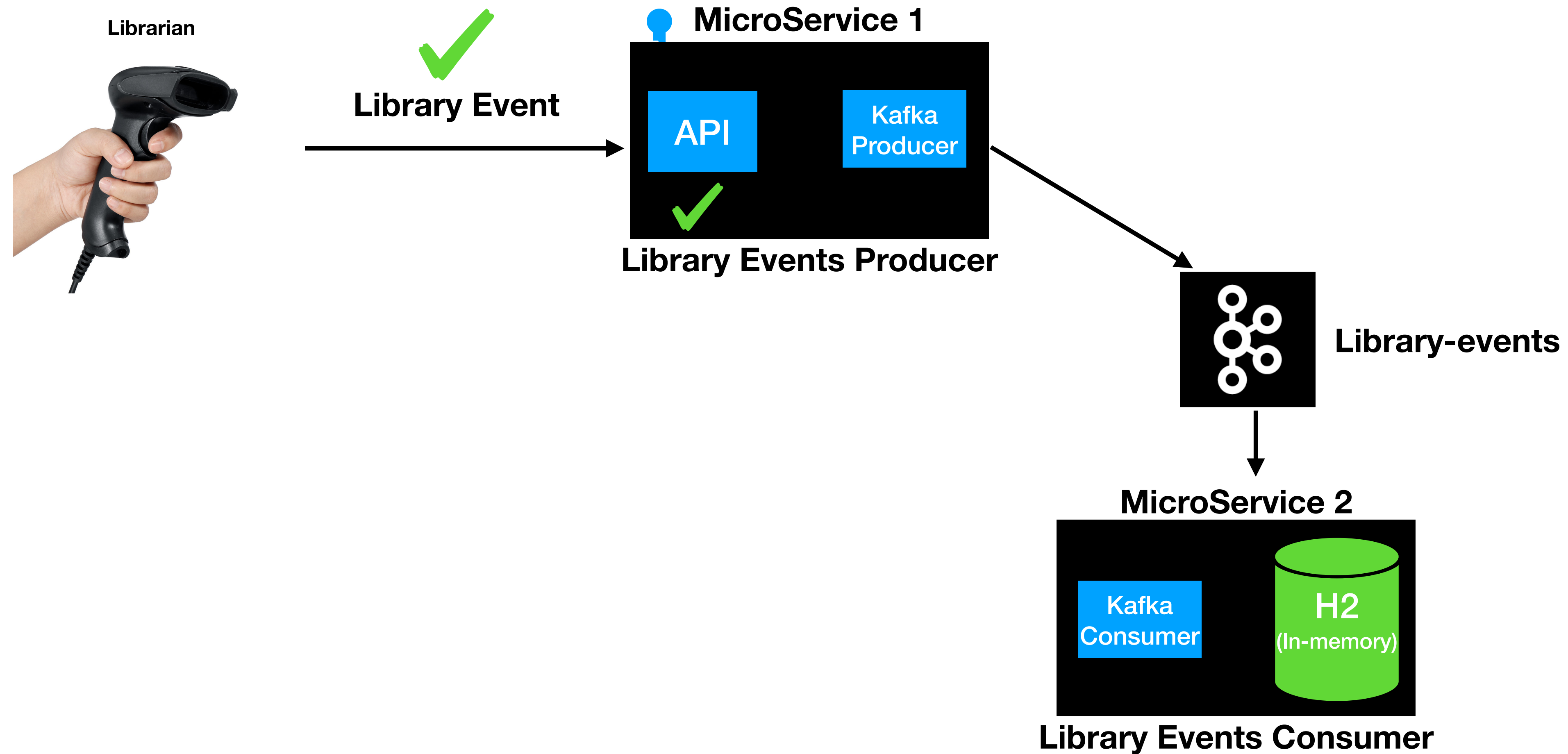


# KafkaTemplate AutoConfiguration

application.yml

```
spring:  
  profiles: local  
  kafka:  
    producer:  
      bootstrap-servers: localhost:9092,localhost:9093,localhost:9094  
      key-serializer: org.apache.kafka.common.serialization.IntegerSerializer  
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

# Library Inventory Architecture



# KafkaAdmin

- Create topics Programmatically
- Part of the **SpringKafka**
- How to Create a topic from Code?
  - Create a Bean of type **KafkaAdmin** in SpringConfiguration
  - Create a Bean of type **NewTopic** in SpringConfiguration

# **Introduction To Automated Tests**

# Why Automated Tests ?

- Manual testing is time consuming
- Manual testing slows down the development
- Adding new changes are error prone

# What are Automated Tests?

- Automated Tests run against your code base
- Automated Tests run as part of the build
- This is a requirement for today's software development
- Easy to capture bugs
- Types of Automated Tests:
  - UnitTest
  - Integration Tests
  - End to End Tests

# Tools for Automated

- JUnit
- Spock

# **Integration Tests**

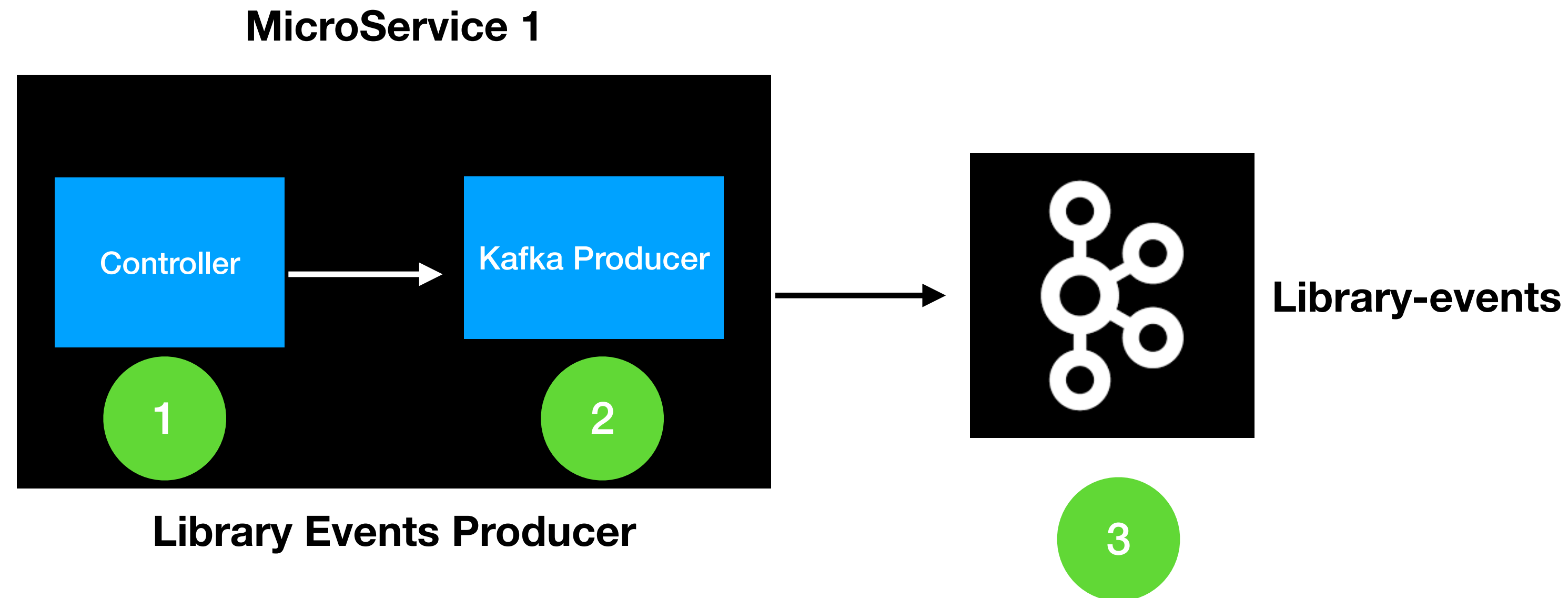
## **Using**

## **JUnit5**

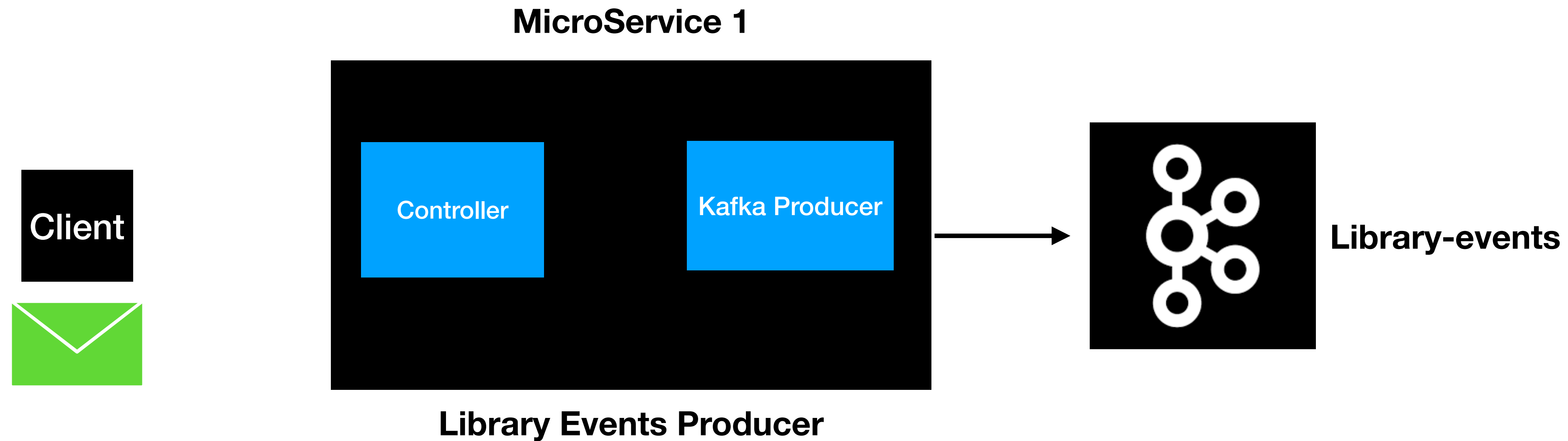


# What is Integration Test?

- Test combines the different layers of the code and verify the behavior is working as expected.



# Integration Test

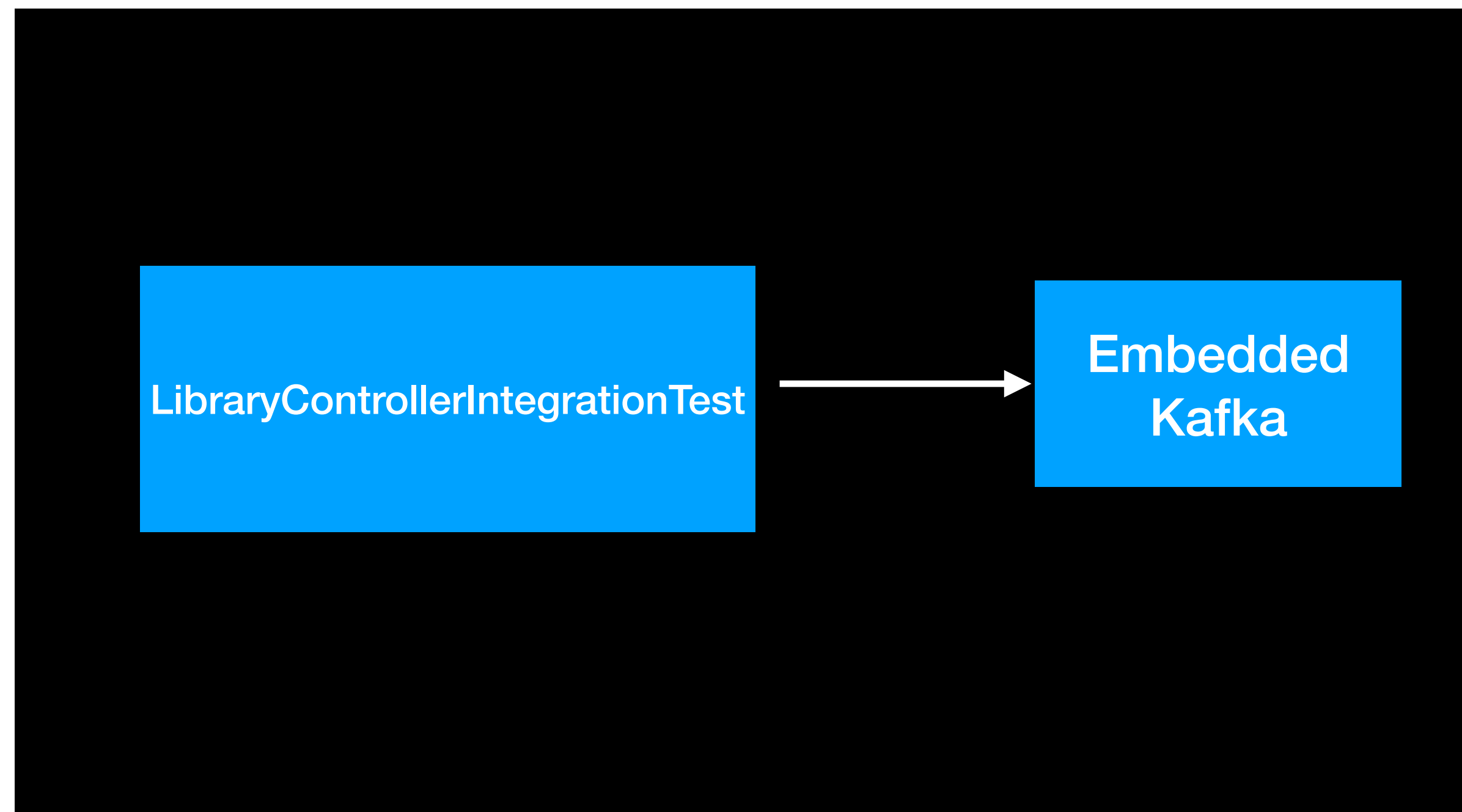


# Embedded Kafka

# What is EmbeddedKafka?

- In-Memory Kafka
- Integration Tests can interact with EmbeddedKafka

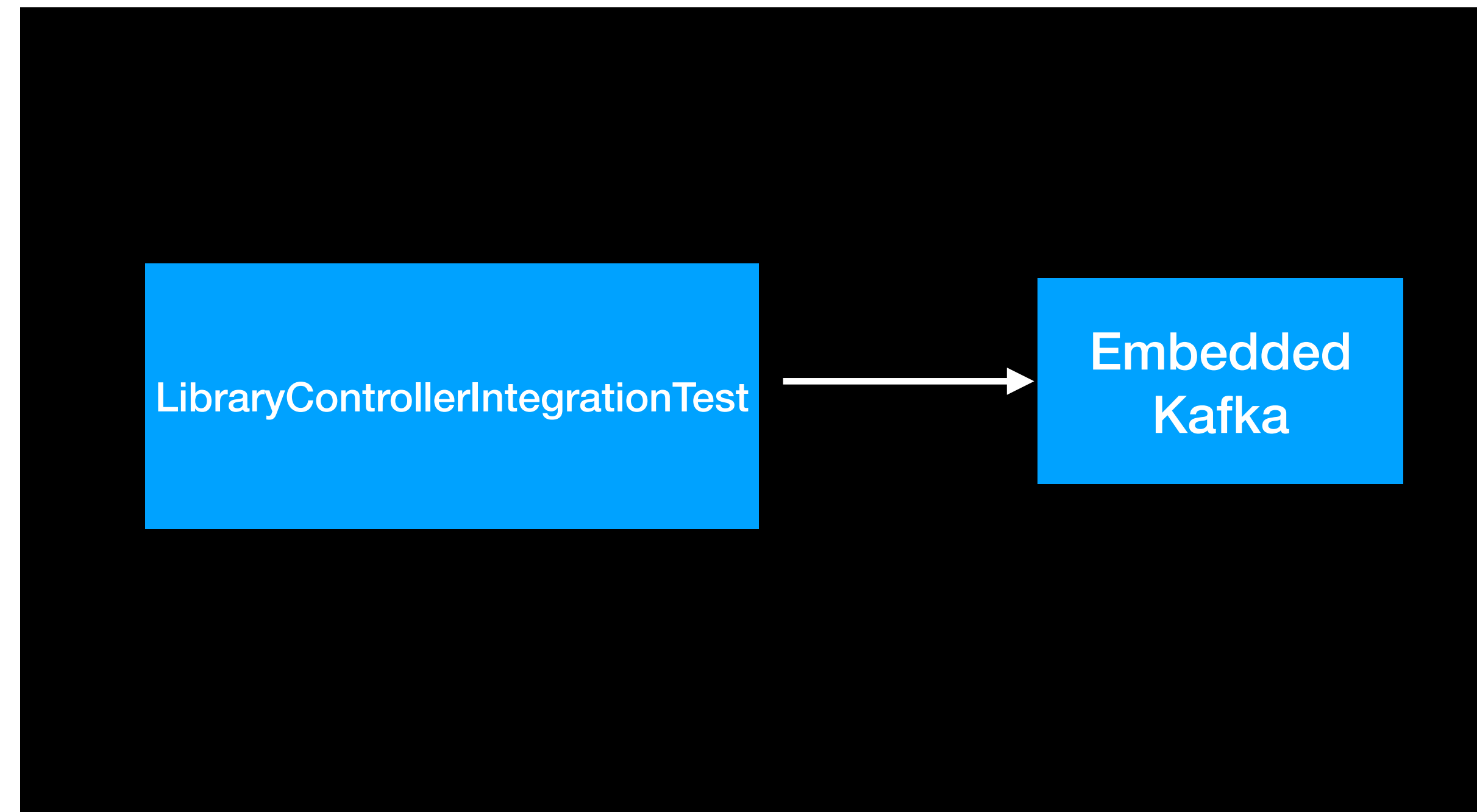
Library Events Producer



# Why Embedded Kafka ?

- Easy to write Integration Tests
- Test all the code as like you interact with Kafka

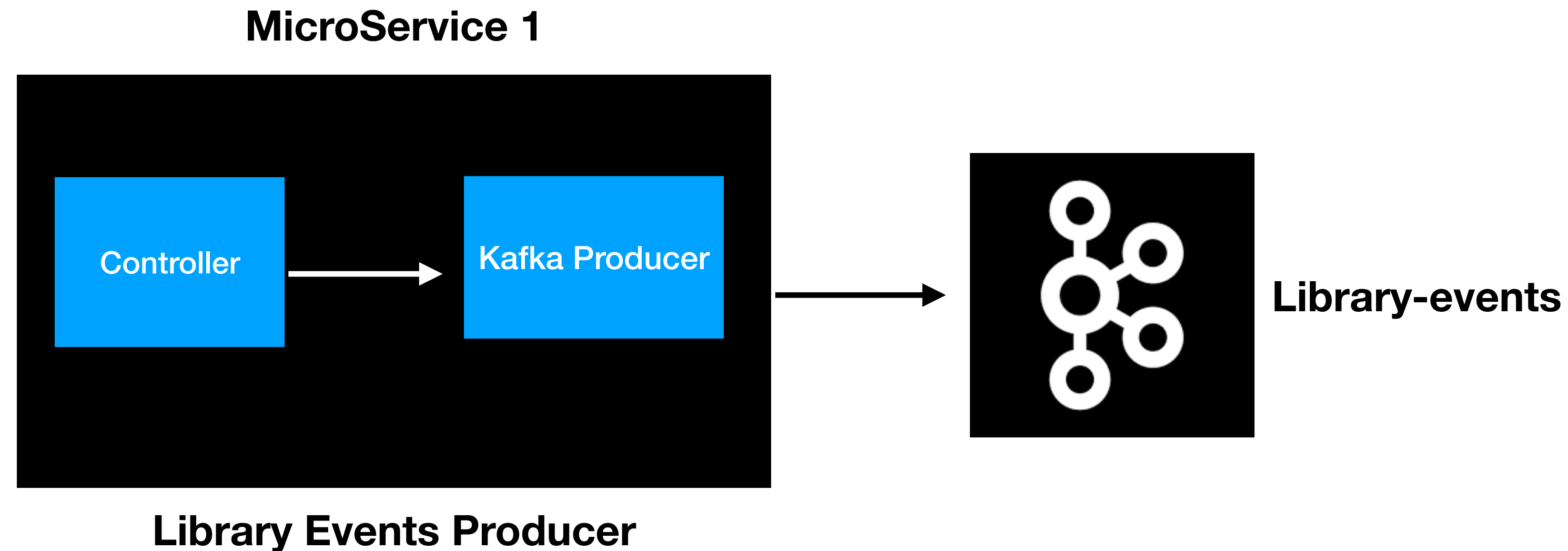
**Library Events Producer**



# **Unit Tests Using JUnit5**

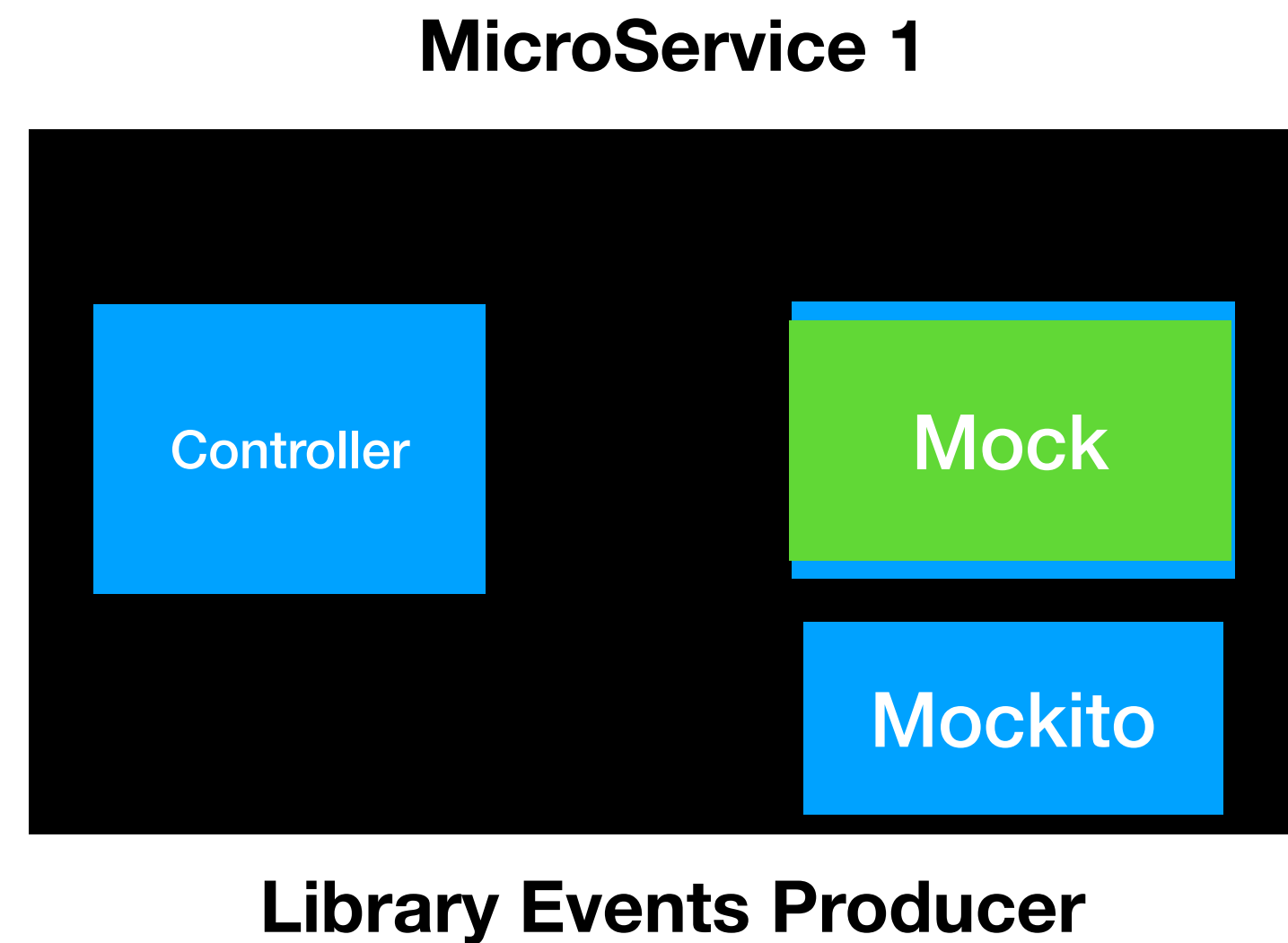
# What is Unit Test?

- Test the just focuses on a single unit (method)
- Mocks the external dependencies



# What is Unit Test?

- Test the just focuses on a single unit (method)
- Mocks the external dependencies

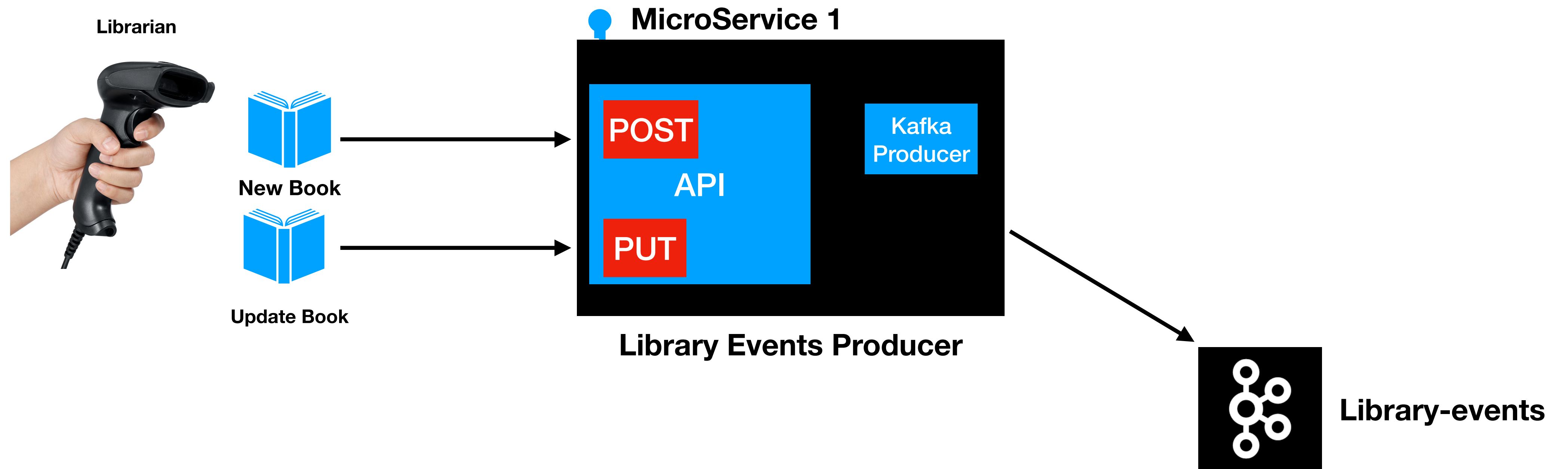




# Why Unit Test?

- Unit Tests are handy to mock external dependencies
- Unit Tests are faster compared to Integration tests
- Unit Tests cover scenarios that's not possible with Integration tests

# Library Events Producer API






# PUT - “/v1/libraryevent”

- libraryEventId is a mandatory field

```
{  
  "libraryEventId": 123,  
  "eventStatus": null,  
  "book": {  
    "bookId": 456,  
    "bookName": "Kafka Using Spring Boot",  
    "bookAuthor": "Dilip"  
  }  
}
```

# Kafka Producer Configurations

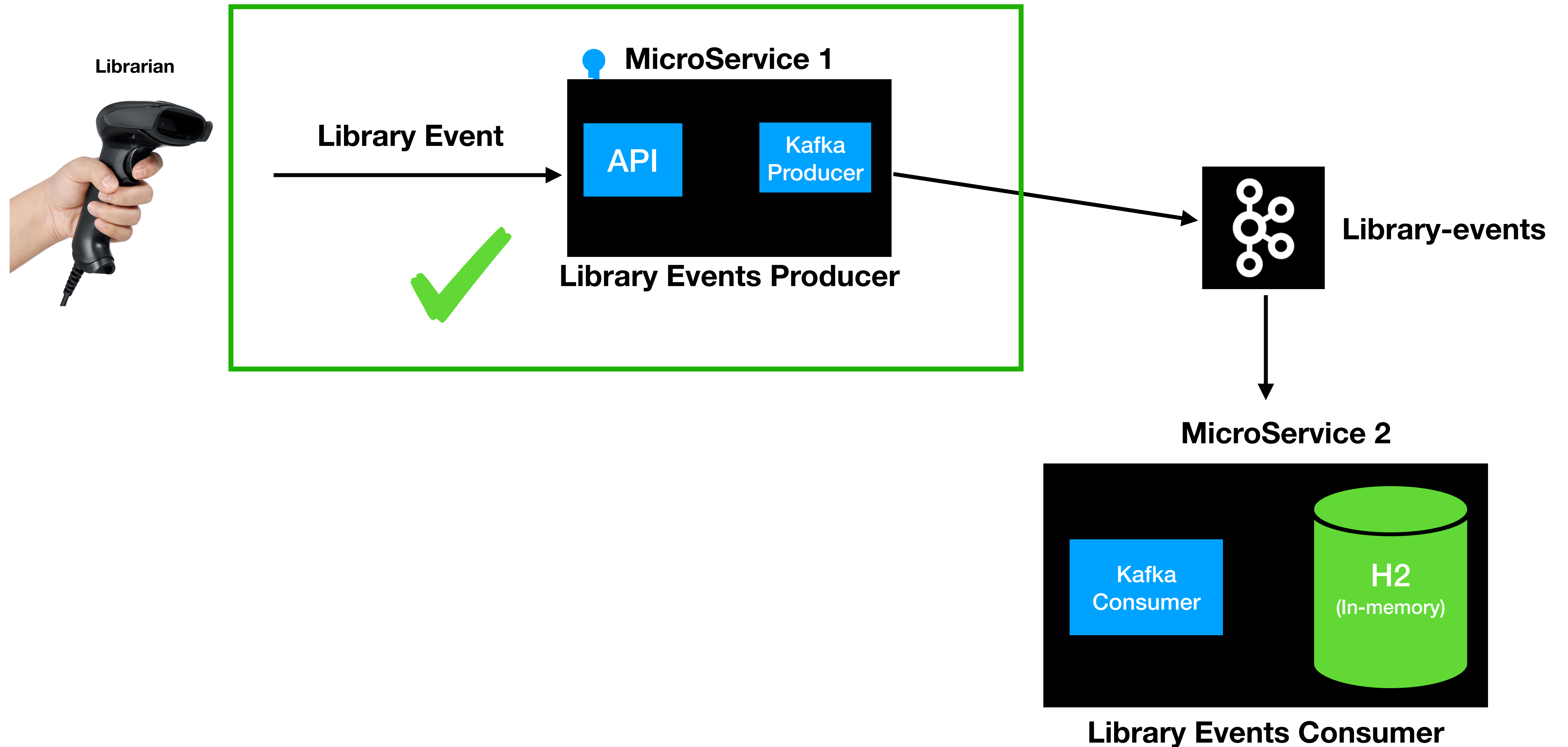
# Kafka Producer Configurations

- acks
  - acks = 0, 1 and -1(all)
  - acks = 1 -> guarantees message is written to a leader 
  - acks = -1(all) -> guarantees message is written to a leader and to all the replicas ( Default) 
  - acks=0 -> no guarantee (Not Recommended) 

# Kafka Producer Configurations

- retries
  - Integer value = [0 - 2147483647]
  - In Spring Kafka, the default value is -> **2147483647**
- retry.backoff.ms
  - Integer value represented in milliseconds
  - Default value is 100ms

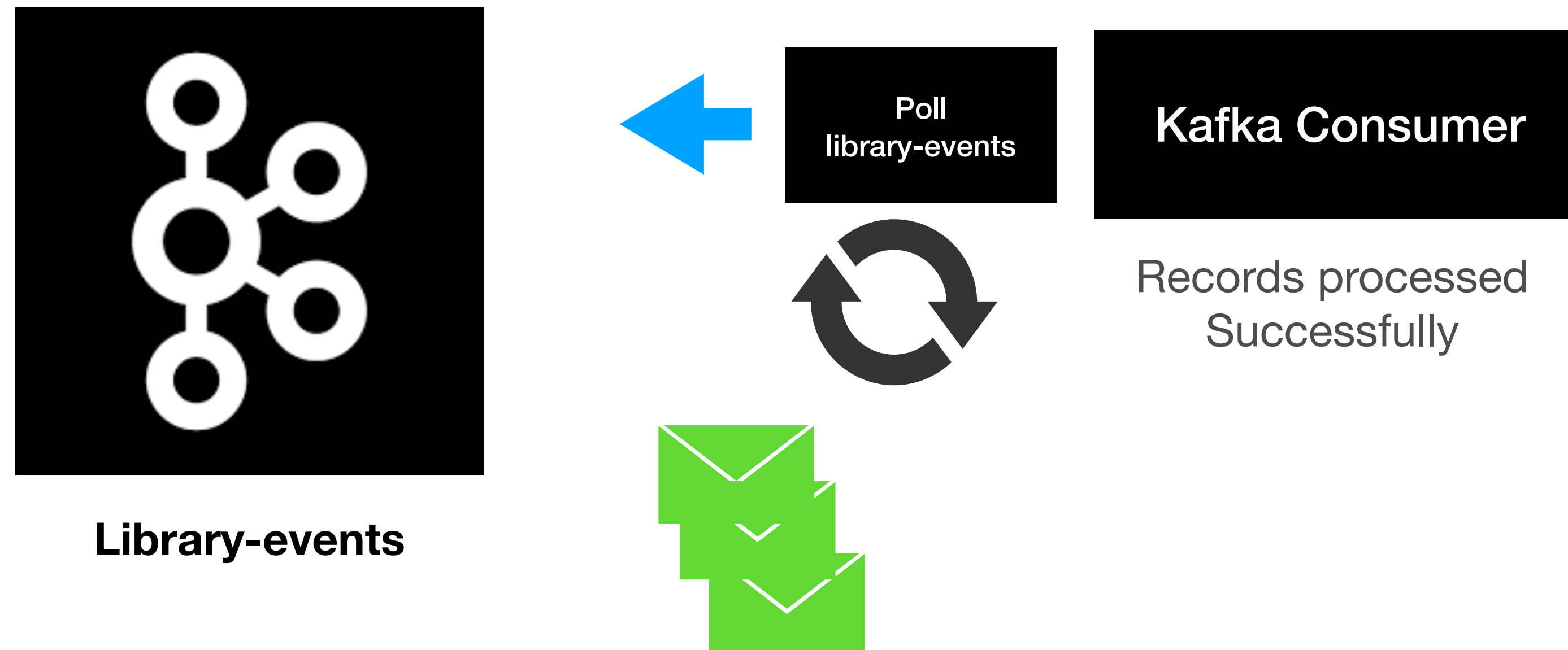
# Library Events Consumer



# Spring Kafka Consumer



# Kafka Consumer



# Spring Kafka Consumer

- MessageListenerContainer
  - KafkaMessageListenerContainer
  - ConcurrentMessageListenerContainer
- **@KafkaLisener** Annotation
  - Uses ConcurrentMessageListenerContainer behind the scenes

# KafkaMessageListenerContainer

- Implementation of MessageListenerContainer
- Polls the records
- Commits the Offsets
- Single Threaded

# ConcurrentMessageListenerContainer

Represents multiple **KafkaMessageListenerContainer**

# @KafkaListener

- This is the easiest way to build Kafka Consumer
- KafkaListener Sample Code

```
@KafkaListener(topics = {"${spring.kafka.topic}"})  
public void onMessage(ConsumerRecord<Integer, String> consumerRecord) {  
    log.info("OnMessage Record : {}", consumerRecord);  
}
```

- Configuration Sample Code

```
@Configuration  
@EnableKafka  
@Slf4j  
public class LibraryEventsConsumerConfig {
```

# KafkaConsumer Config

```
key-deserializer: org.apache.kafka.common.serialization.IntegerDeserializer  
value-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
group-id: library-events-listener-group
```

# Consumer Groups & Rebalance

# Consumer Groups

Multiple instances of the same application with the same group id.



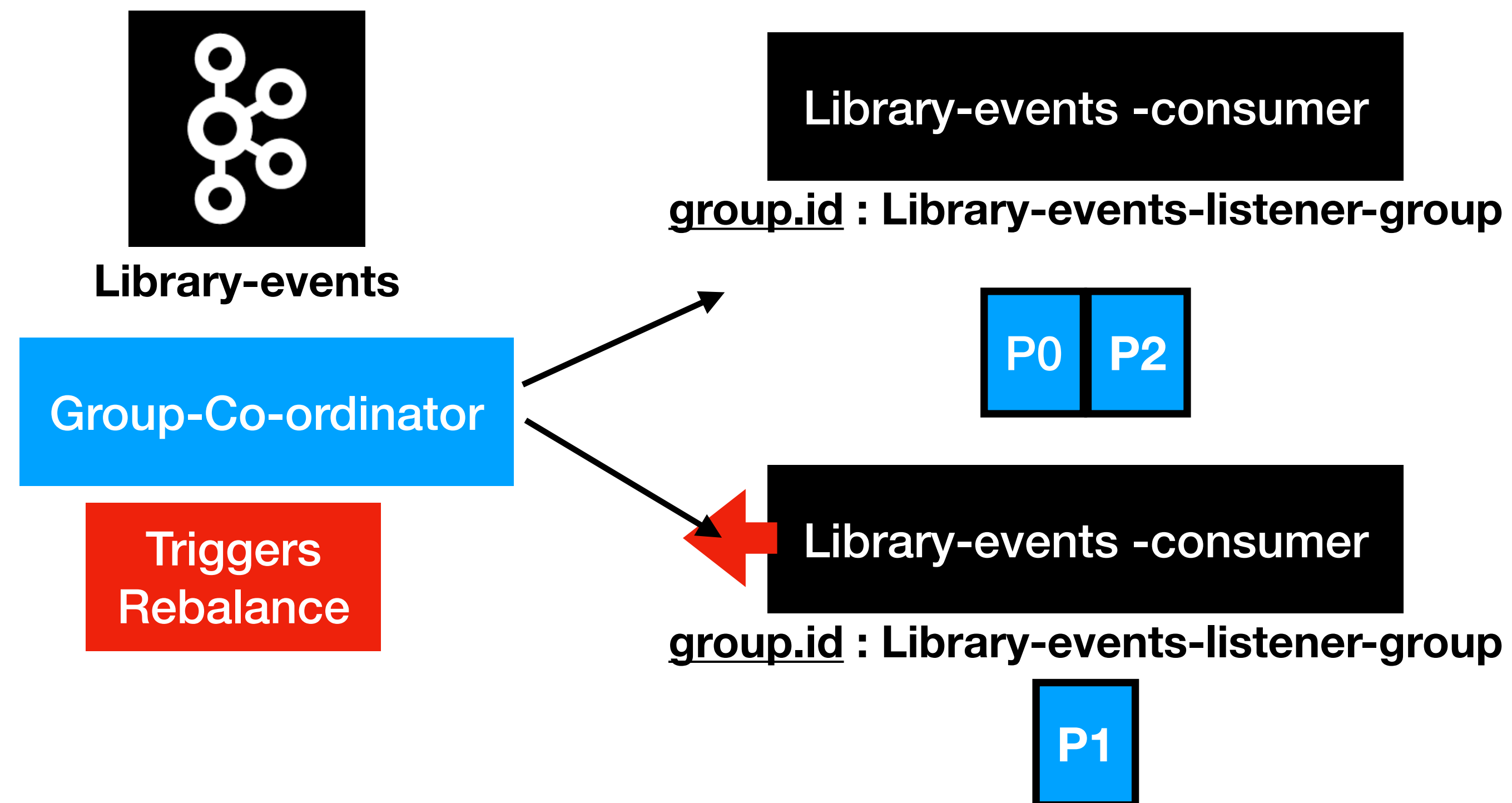
# Rebalance

- Changing the partition ownership from one consumer to another



# Rebalance

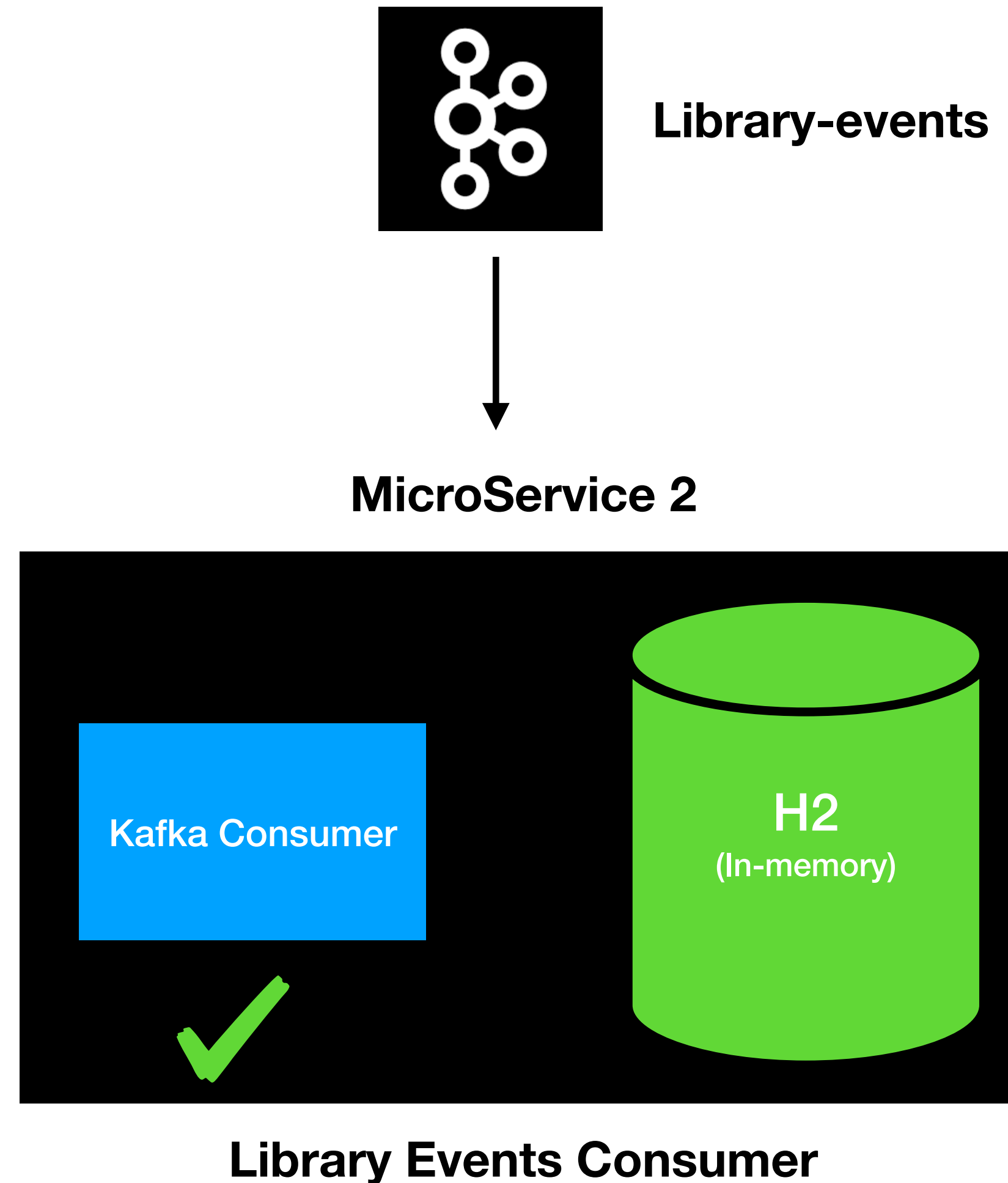
- Changing the partition ownership from one consumer to another



# Committing Offsets



# Library Events Consumer



# Integration Testing For Real DataBases

# Integration Testing using Real Databases

- Different aspects of writing unit and integration testing
- Integration testing using **TestContainers**

SpringOne Platform by Pivotal

## Testing Spring Boot Applications

Andy Wilkinson  
@ankinson

SpringOne Platform by Pivotal

October 7-10 / Austin, TX P

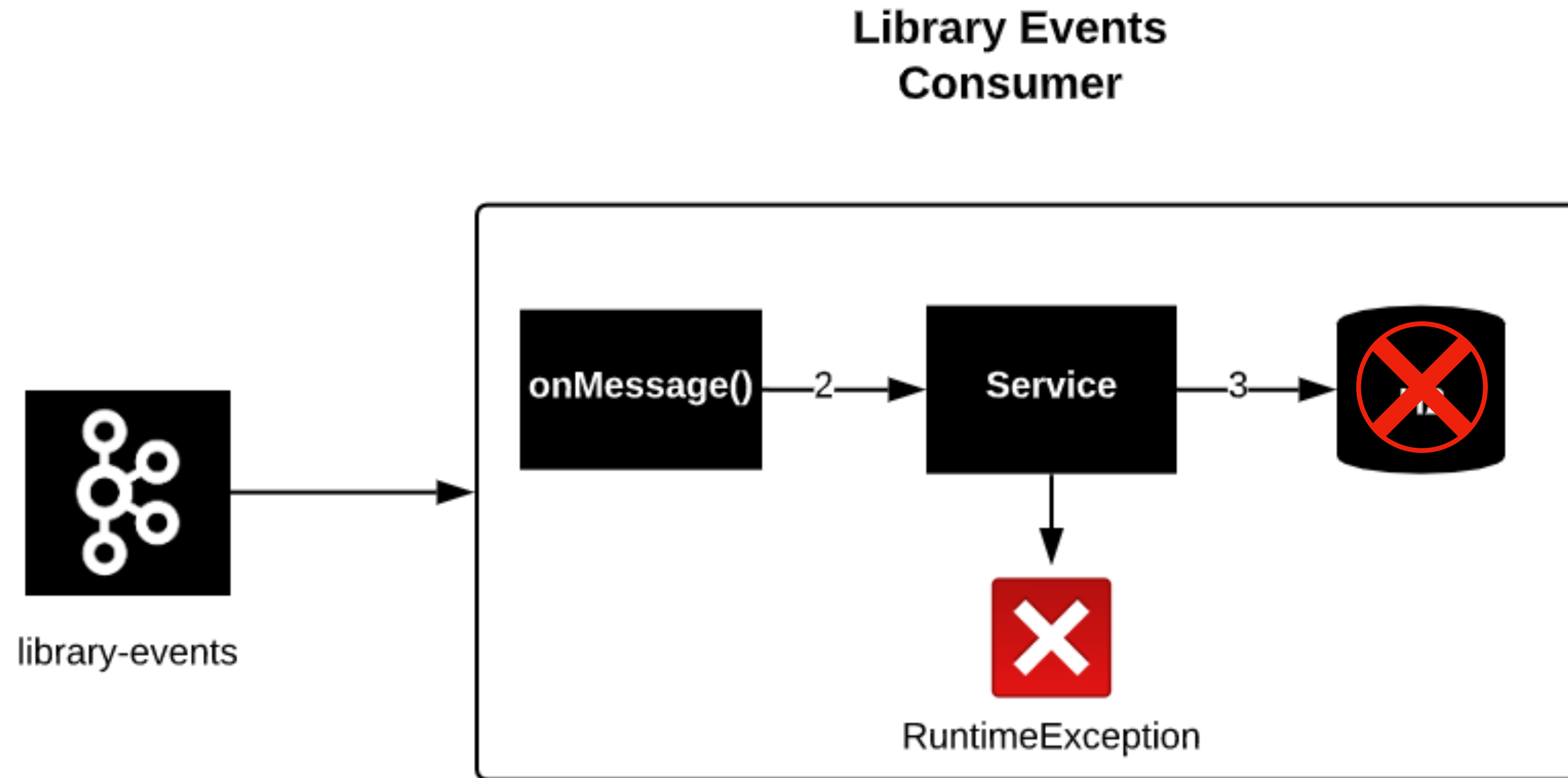
# TestContainers

- What are TestContainers?
  - Testcontainers is a Java library that supports JUnit tests, providing lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a **Docker** container.
- More Info about TestContainers - <https://www.testcontainers.org/>

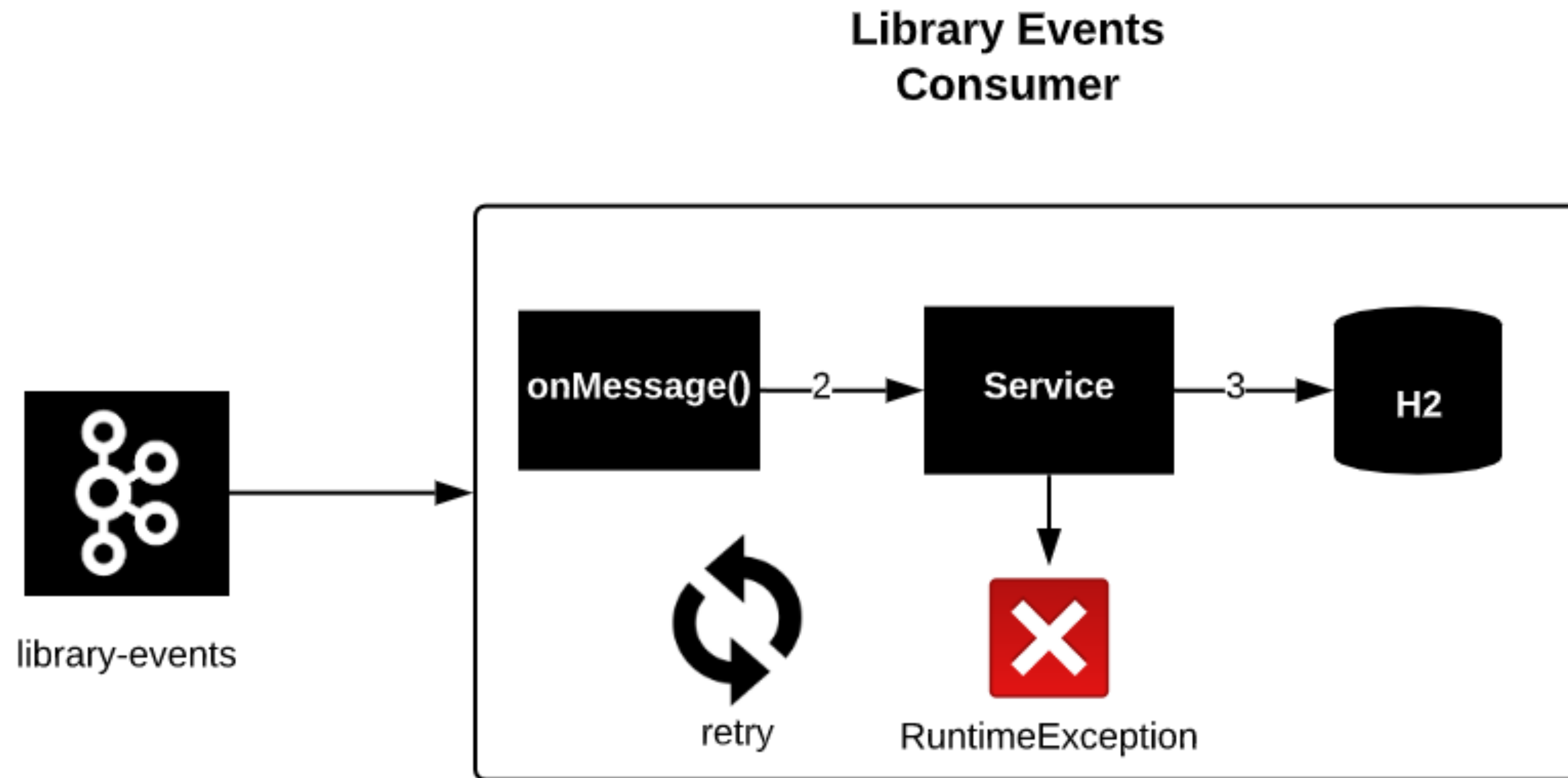
# Retry in Kafka Consumer



# Error in Kafka Consumer

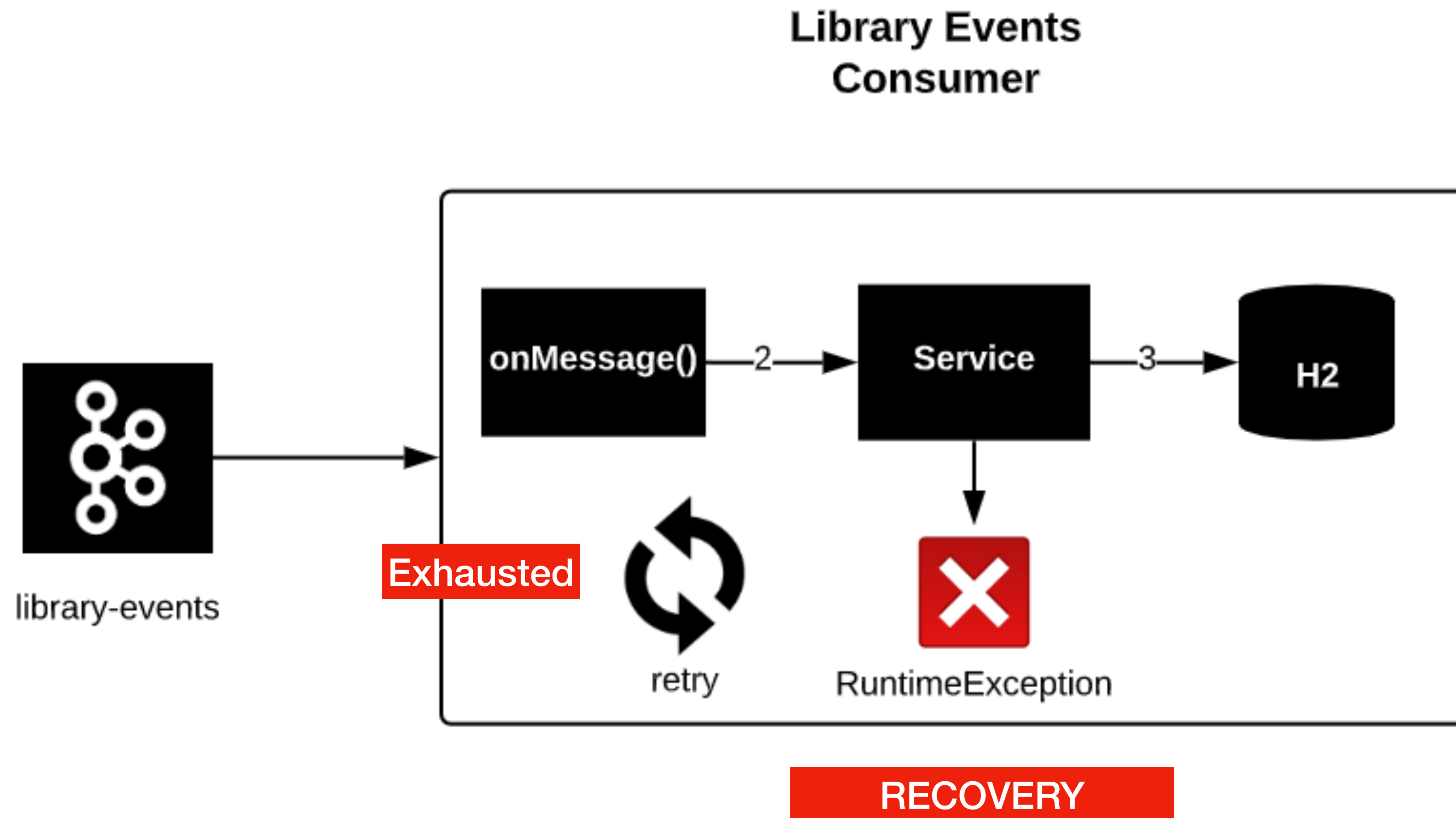


# Retry in Kafka Consumer

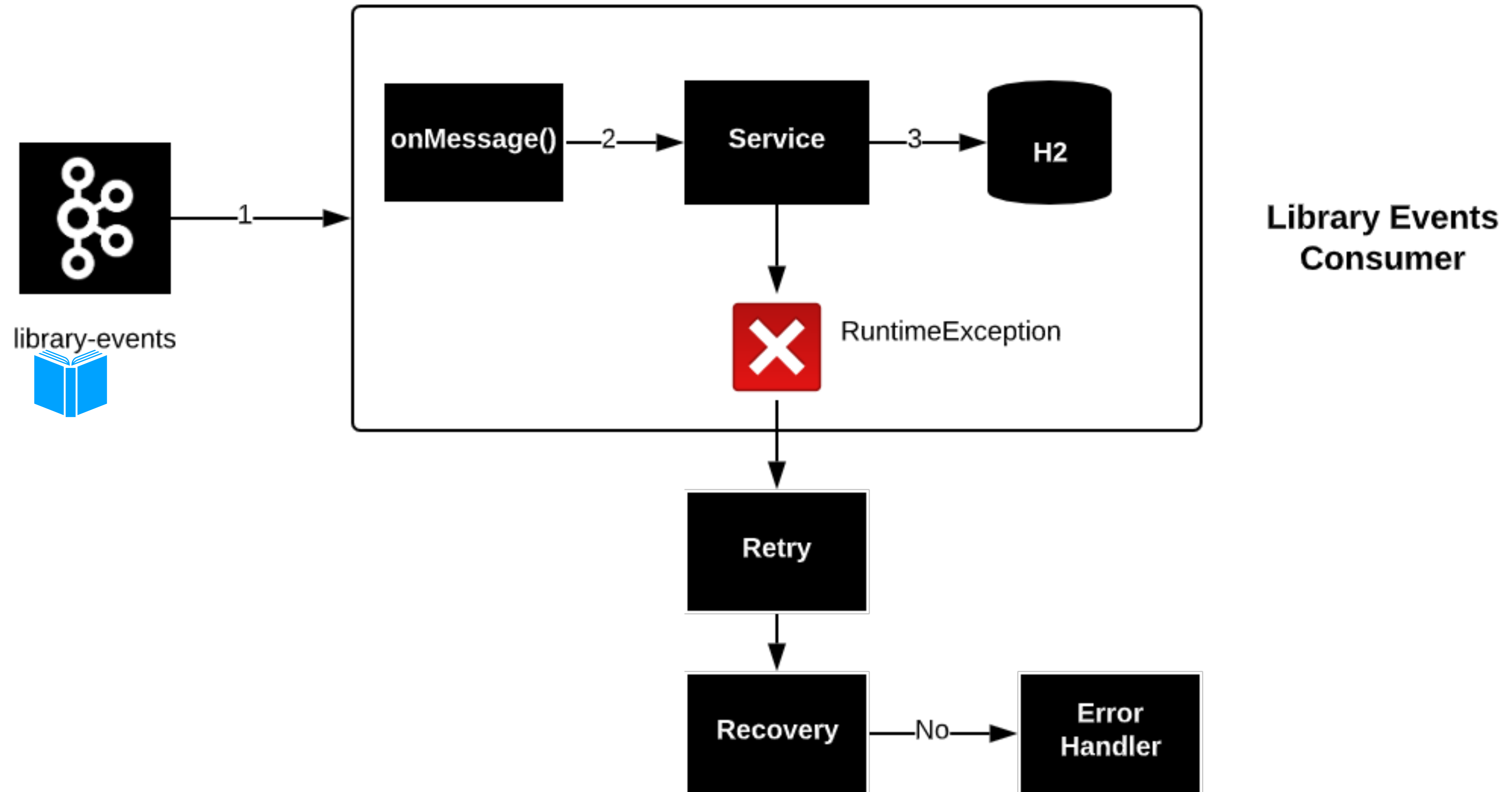


# Recovery in Kafka Consumer

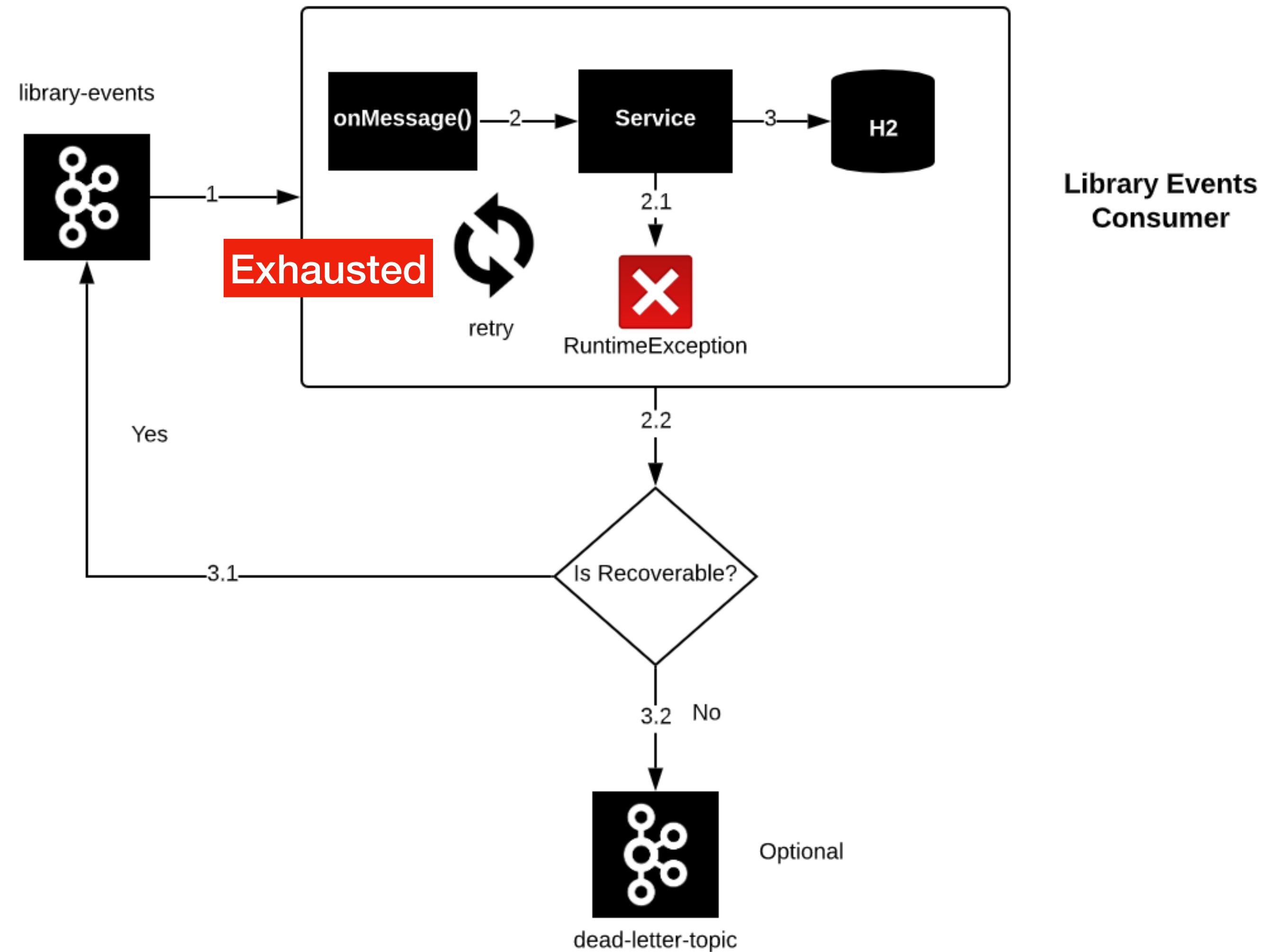
# Recovery in Kafka Consumer



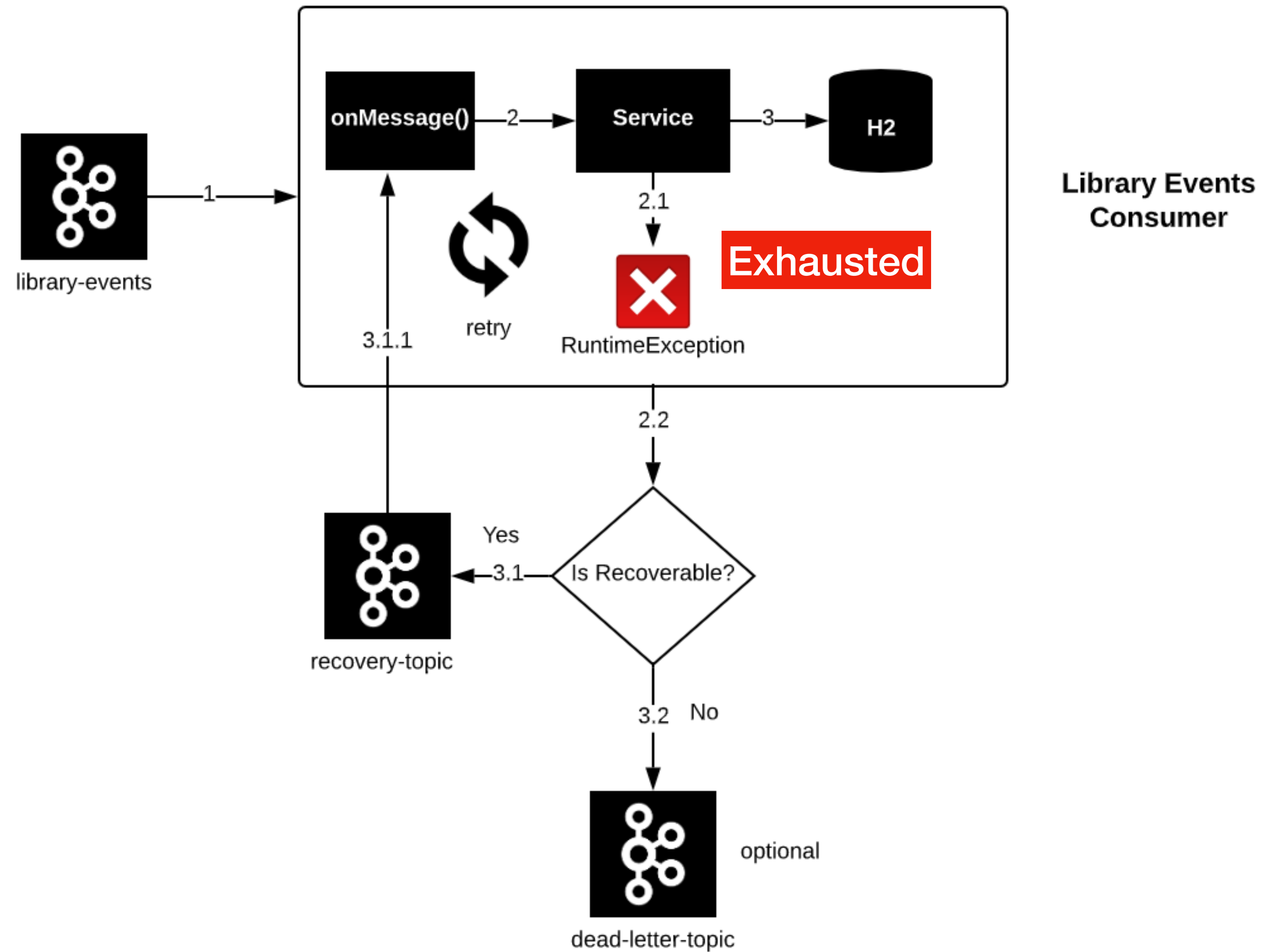
# Retry and Recovery



# Recovery - Type 1



# Recovery - Type 2

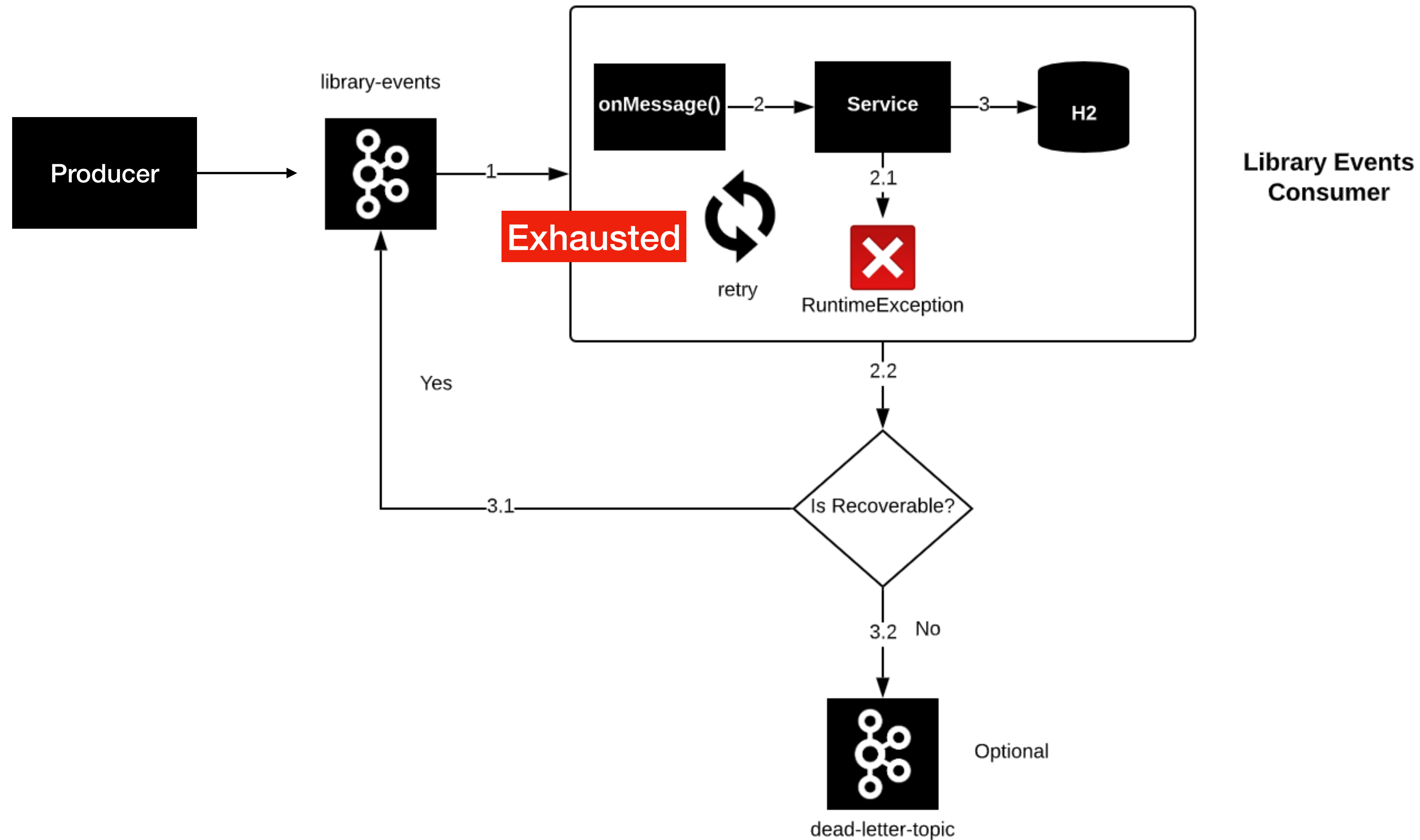


# Issues with Recovery ?

- Recovery can alter the order of events

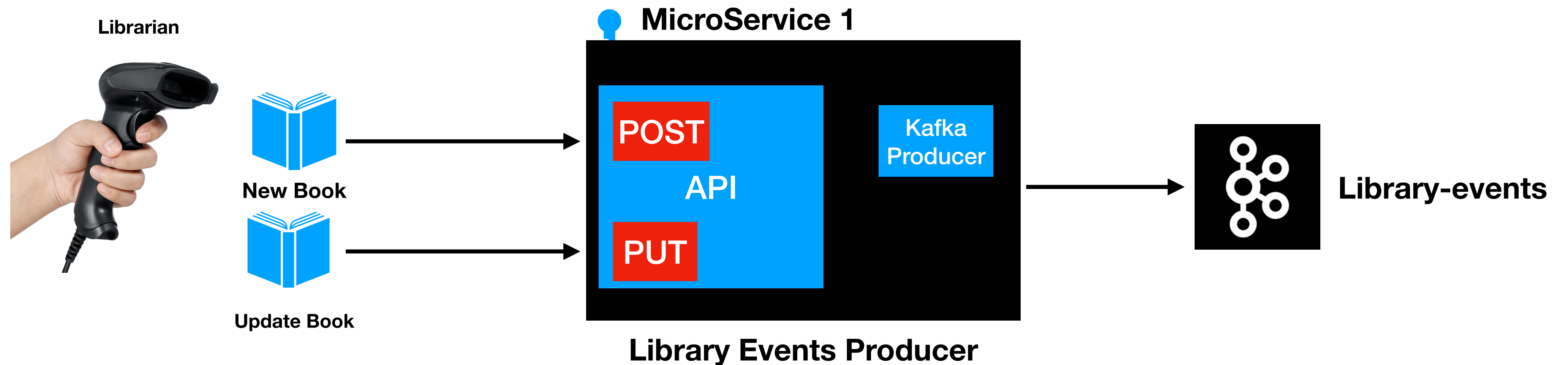


# Recovery - Type 1



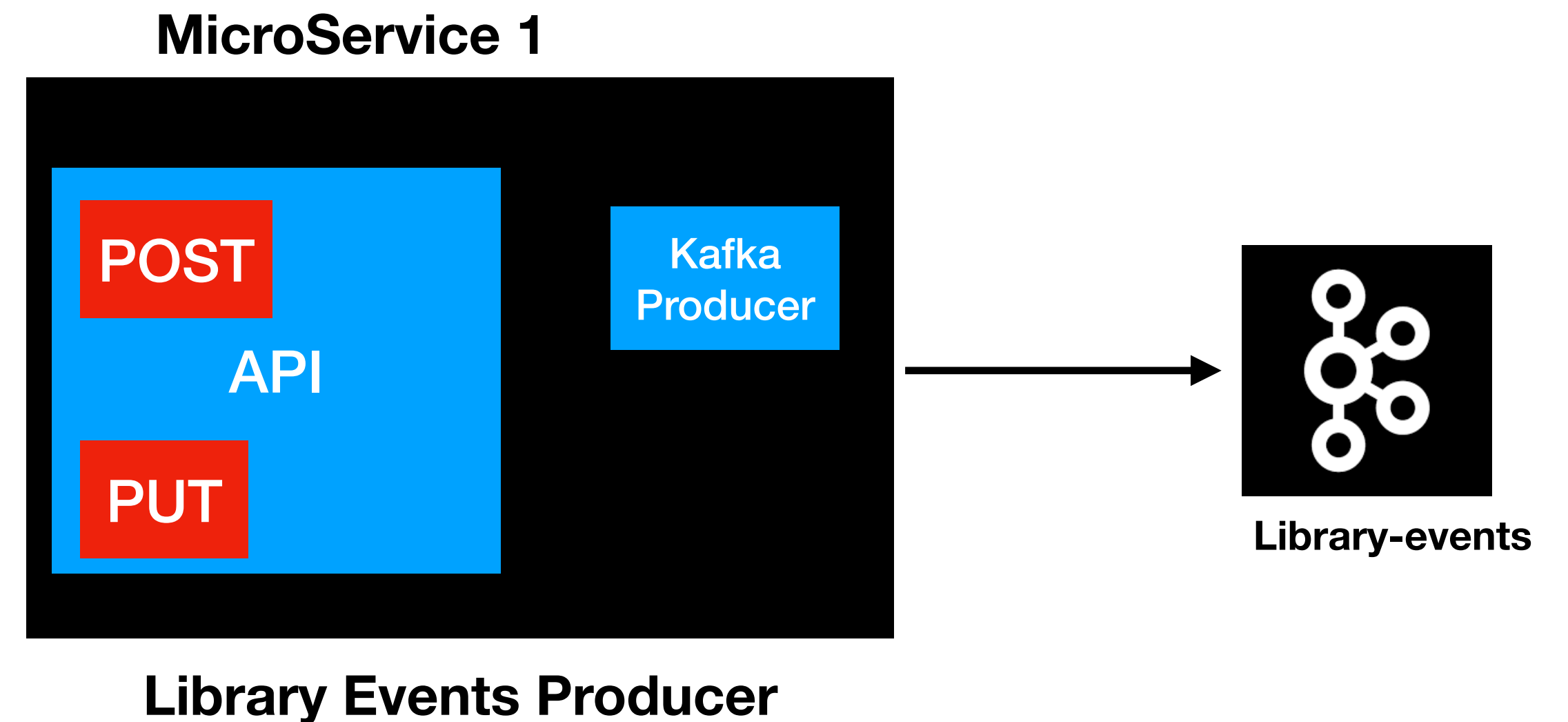
# Error Handling in Kafka Producer

# Library Events Producer API



# Kafka Producer Errors

- Kafka Cluster is not available
- If **acks= all** , some brokers are not available
- **min.insync.replicas** config
  - **Example : min.insync.replicas = 2**, But only one broker is available



# min.insync.replicas



**min.insync.replicas = 2**

**Kafka Cluster**



**Broker 1**



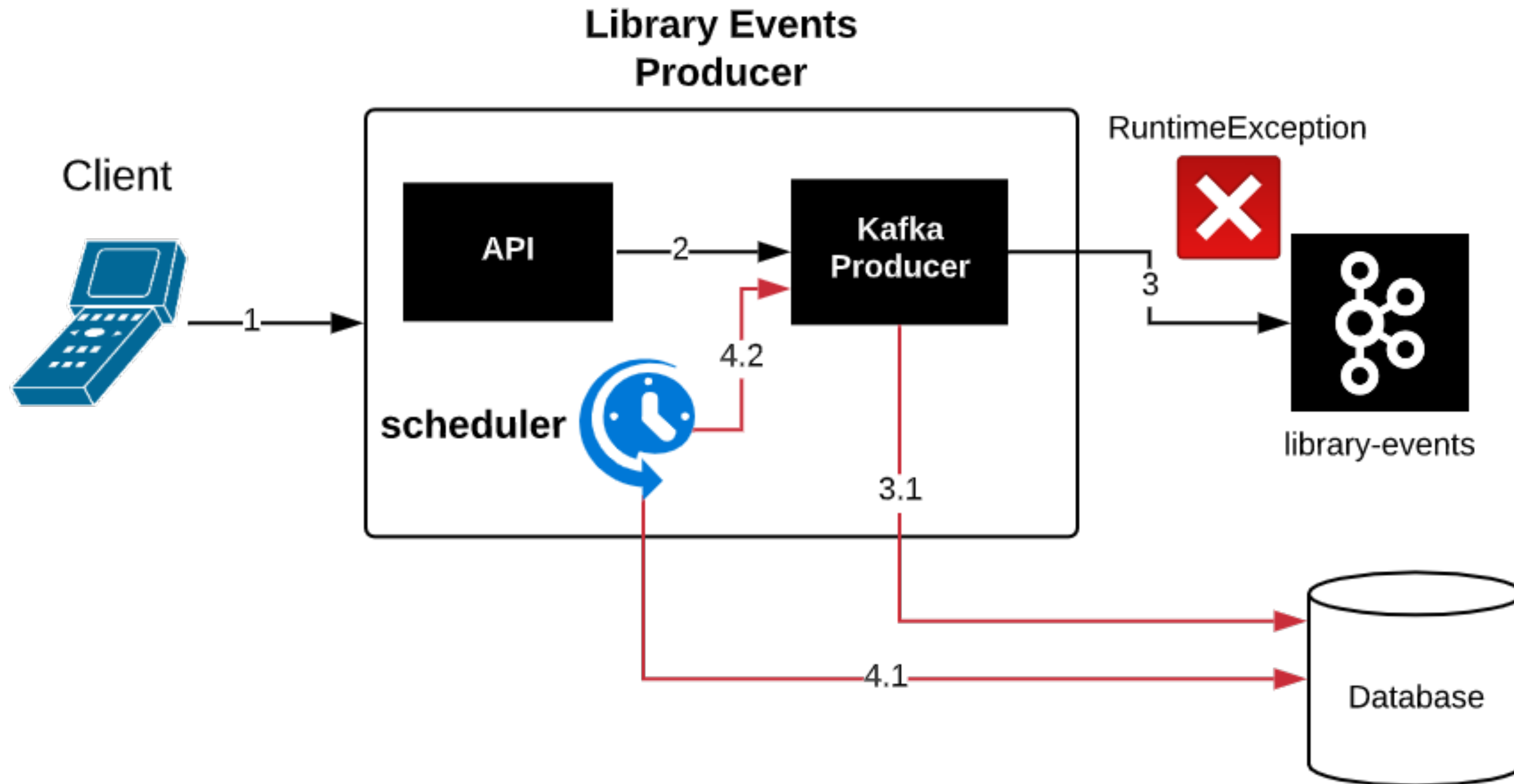
**Broker 2**



**Broker 2**

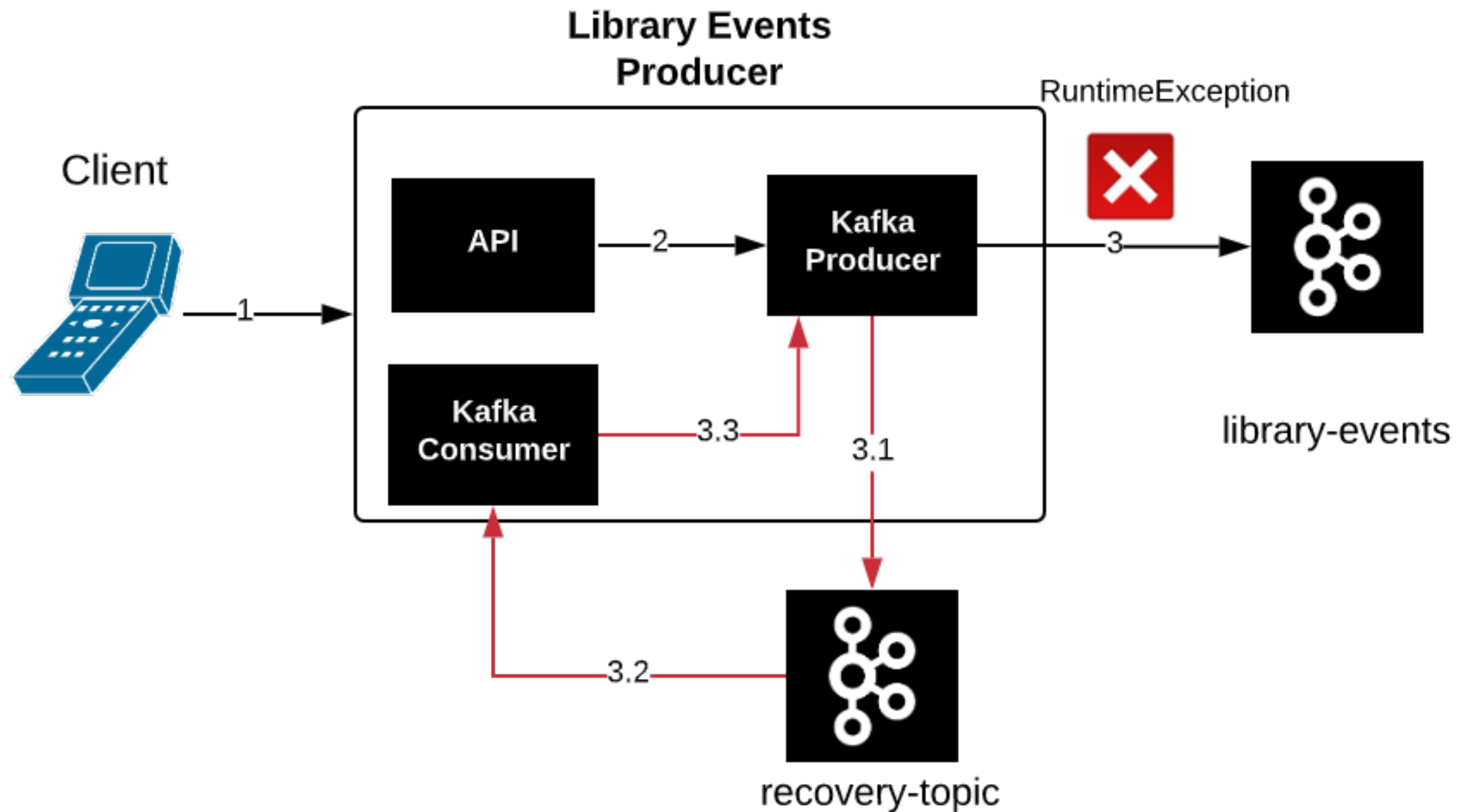
**Retain/Recover  
Failed  
Records**

# Retain/Recover Failed Records



# Retain/Recover Failed Records

## Producer Misconfiguration - Option 2





# Kafka Security

# Kafka Security

- Kafka is used in variety of business:
  - Banking
  - Retail
  - Insurance
- Confidential Information:
  - Credit Card Details
  - Customer Information and etc.,

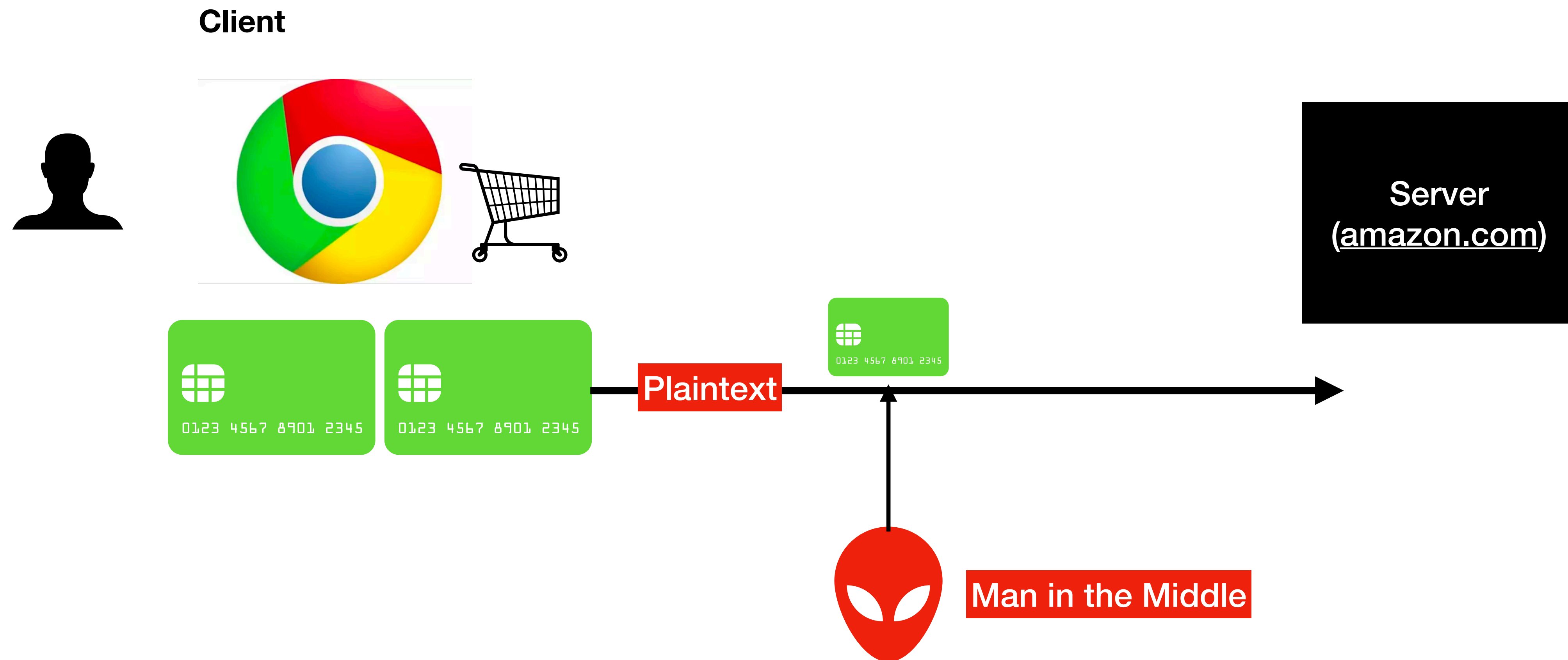
# What Kafka Security Offers?

- Kafka supports two popular protocols:
  - **SSL** (Secured Sockets Layer), which is **TLS** (Transport Layer Security) now
  - **SASL** (Simple Authentication and Security Layer)

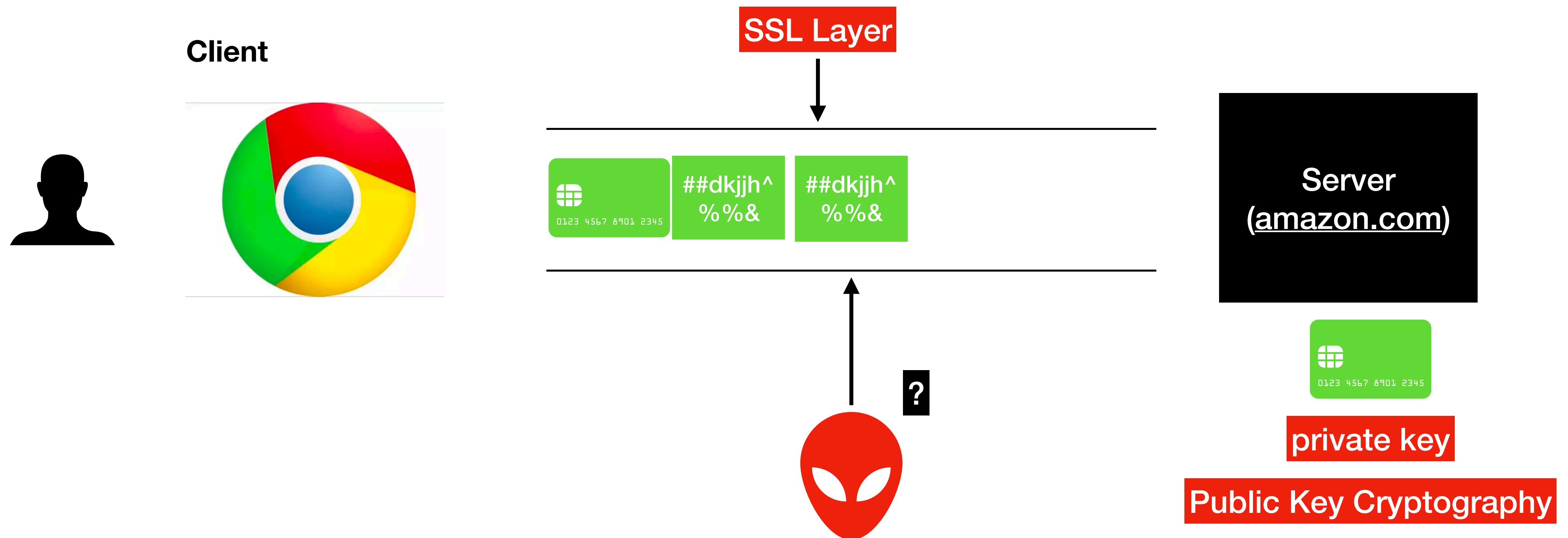
# How SSL works?

- SSL used for two things:
  - Encryption
  - Authentication

# Without SSL Encryption

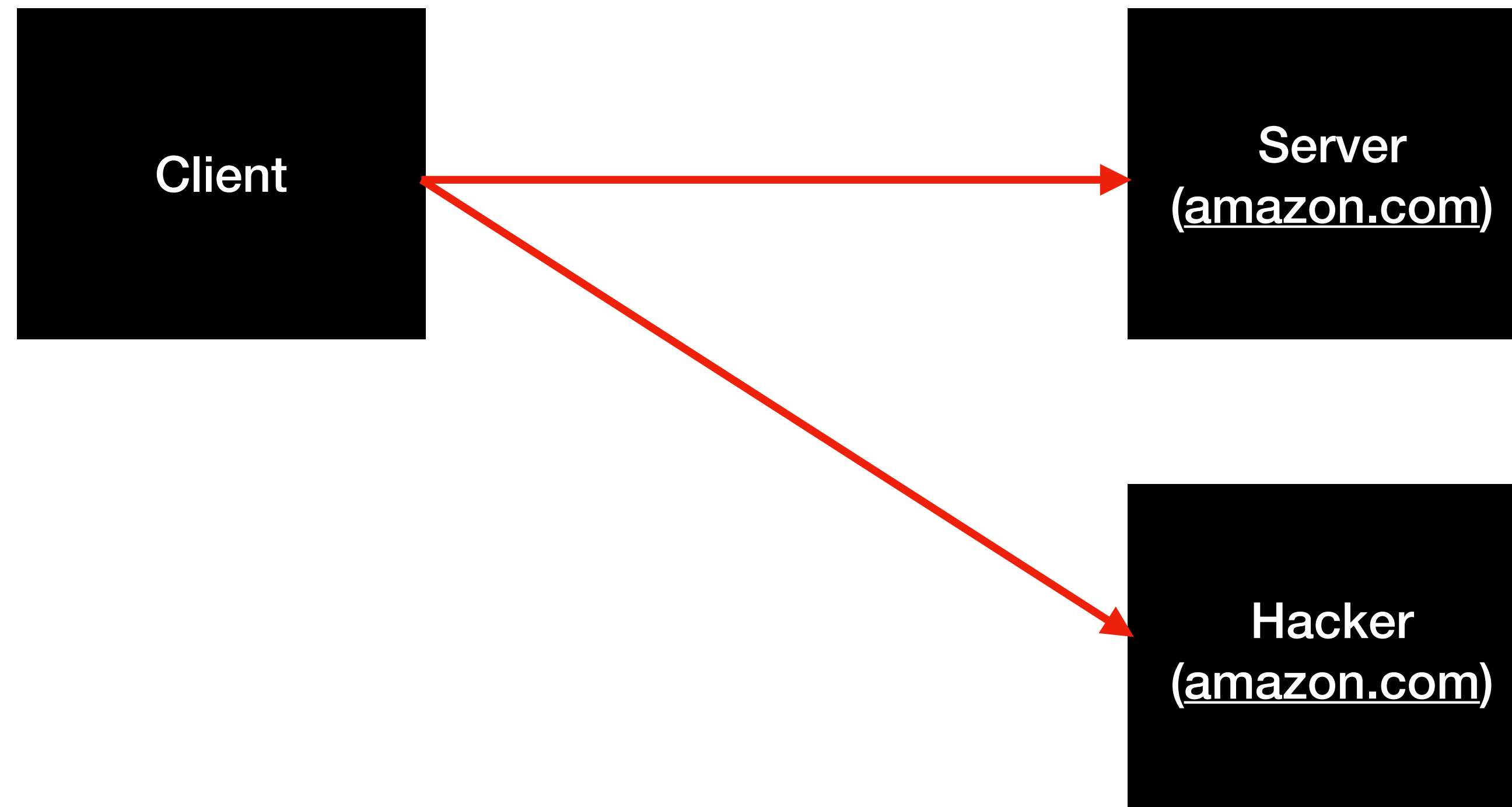


# With SSL Encryption

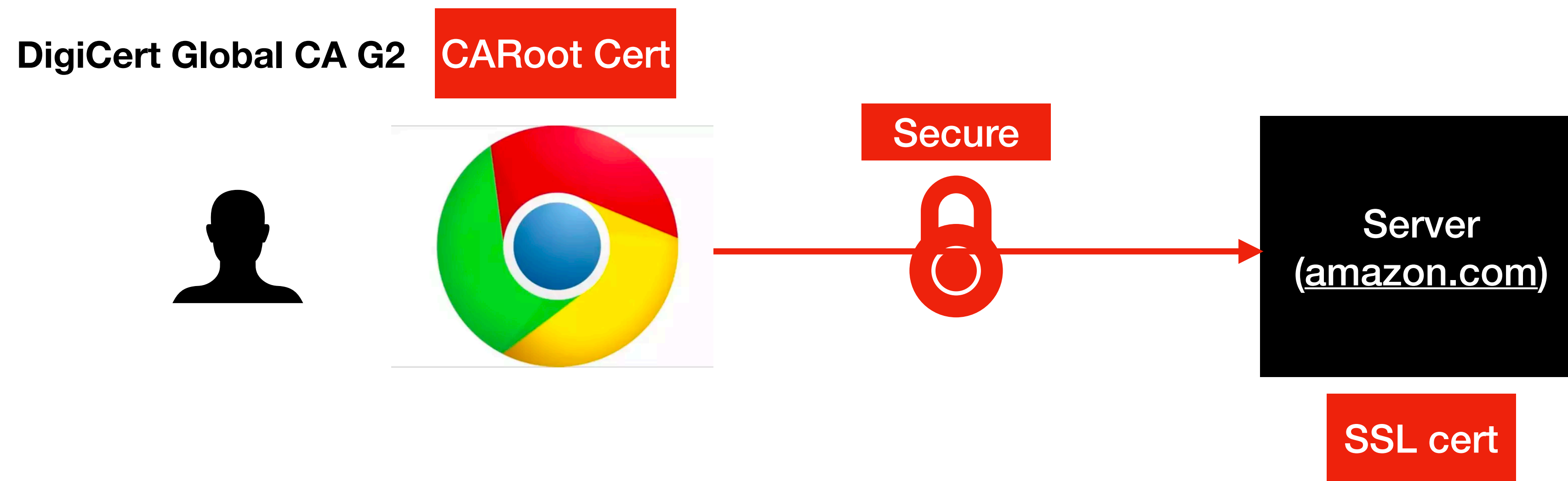


# SSL Authentication

- SSL Authentication is to make sure you are talking to the correct server

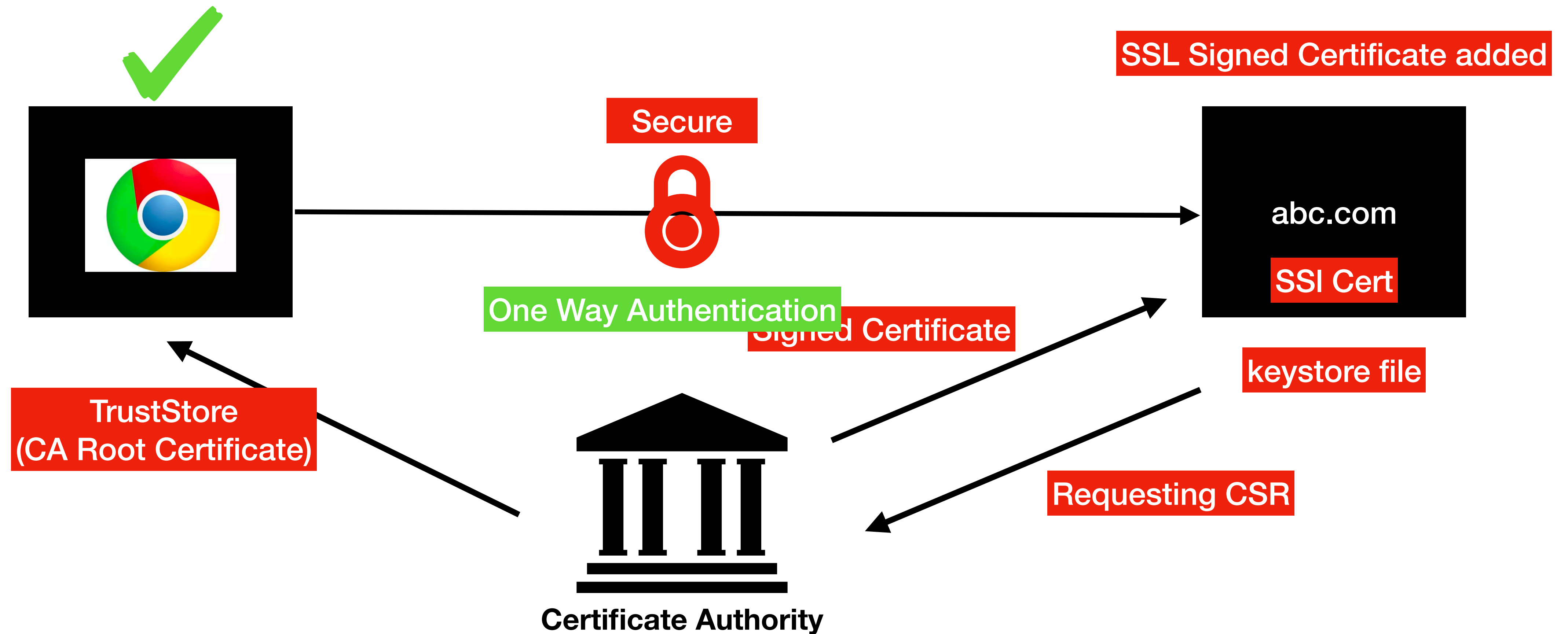


# How SSL Authentication works ?

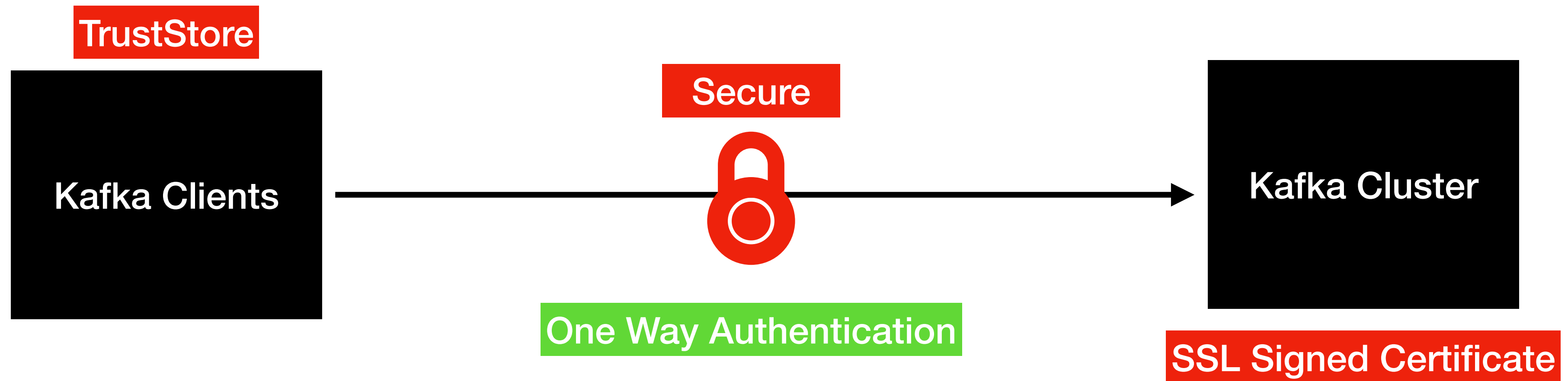




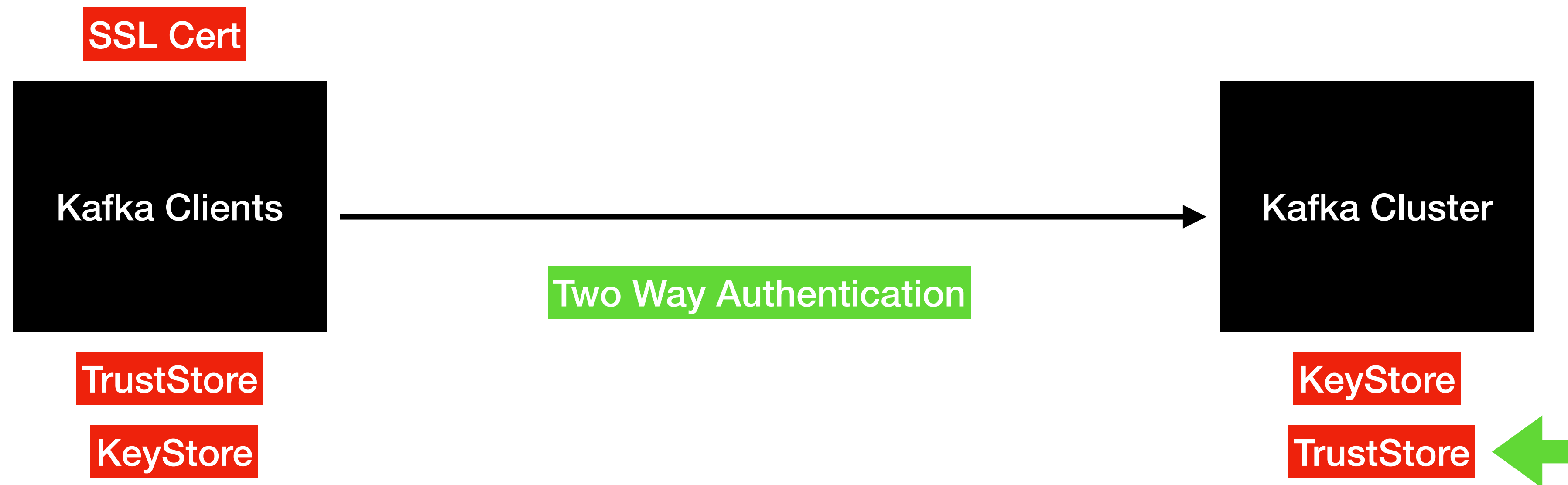
# SSL Requests by Enterprise



# SSL Authentication



# 2 Way SSL Authentication



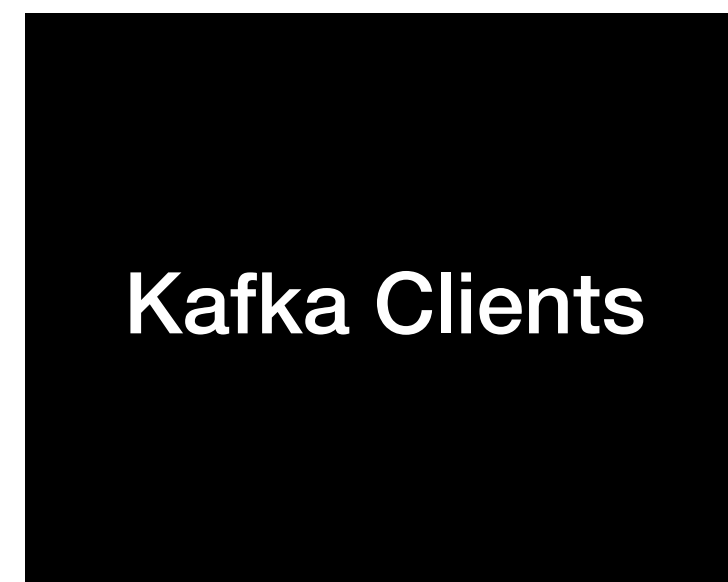
# Kafka SSL SetUp

# SSL Set Up Steps

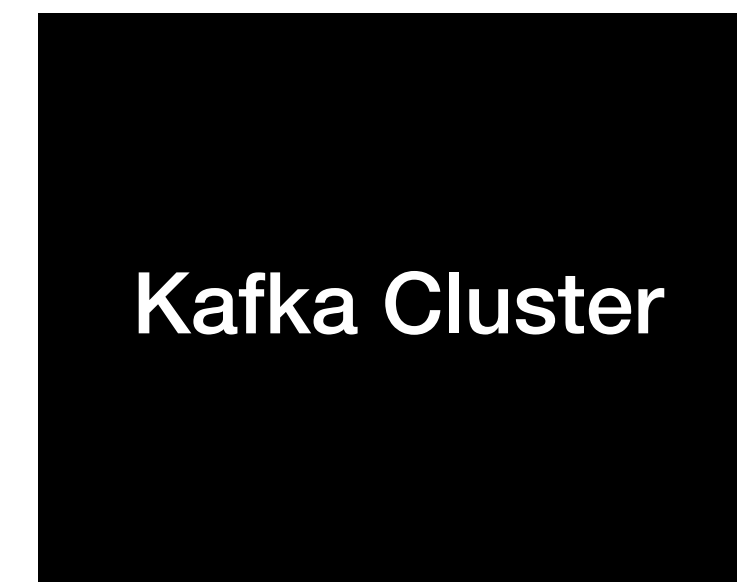
1. Generate **server.keystore.jks**
2. SetUp Local Certificate Authority
3. Create CSR(Certificate Signing Request)
4. Sign the SSL Certificate
5. Add the Signed SSL Certificate to **server.keystore** file
6. Configure the **SSL cert** in our Kafka Broker
7. Create **client.truststore.jks** for the client



# Kafka SSL SetUp



`client.truststore`



SSL Signed Certificate



Local Certificate Authority