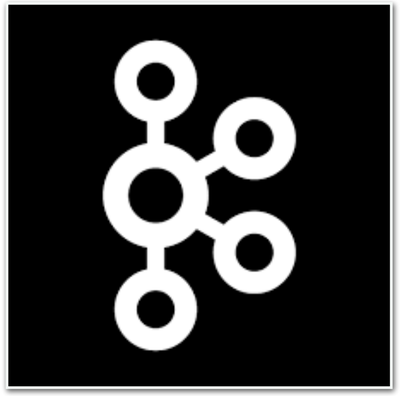


Kafka For Developers Using Spring Boot

Dilip Sundarraj

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

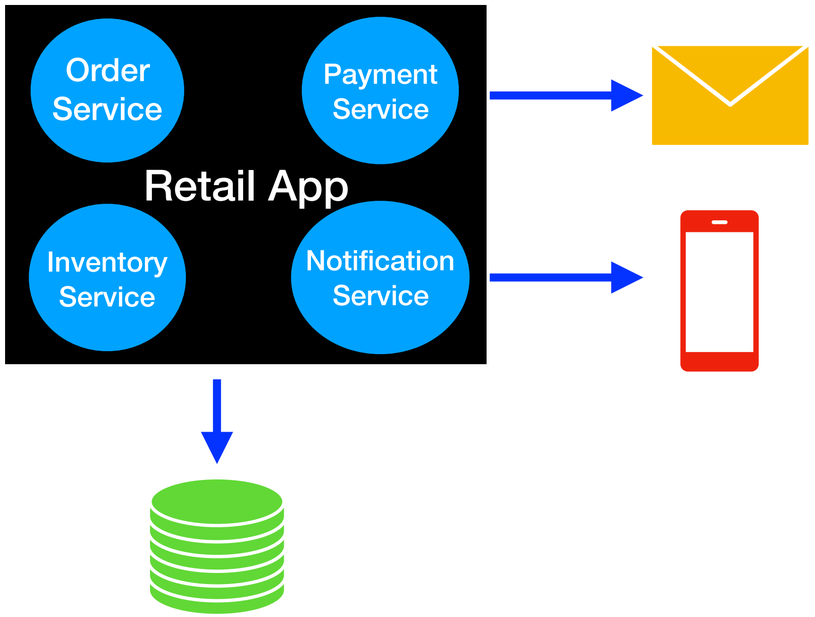
Prerequisites

Course Prerequisites

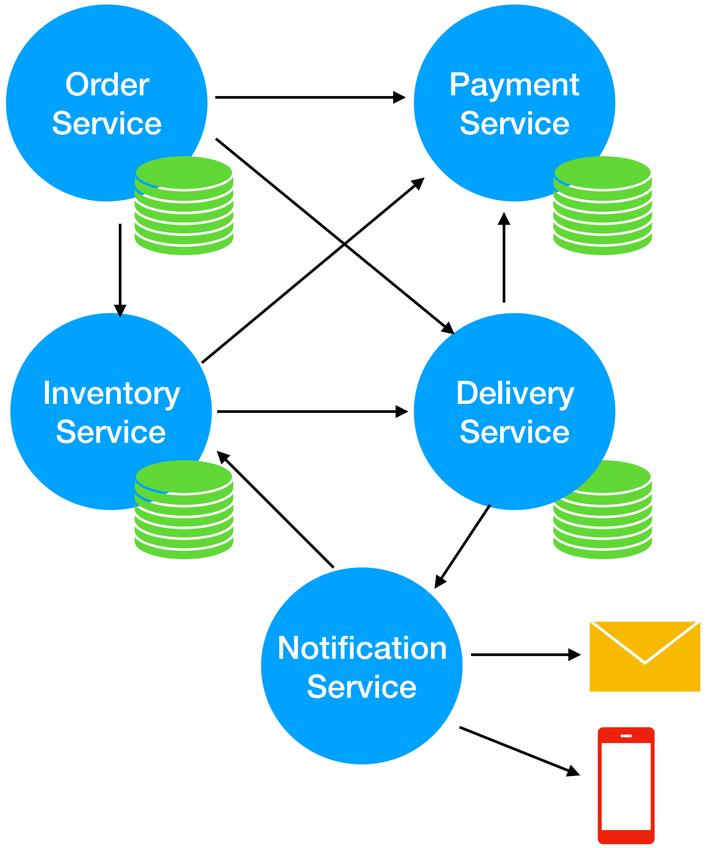
- Prior Knowledge or Working Experience with **Spring Boot/Framework**
- Knowledge about building **Kafka Clients** using Producer and Consumer API
- Knowledge about building **RESTFUL APIs** using Spring Boot
- Experience working with **Spring Data JPA**
- Automated tests using **JUnit**
- Experience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

Software Development

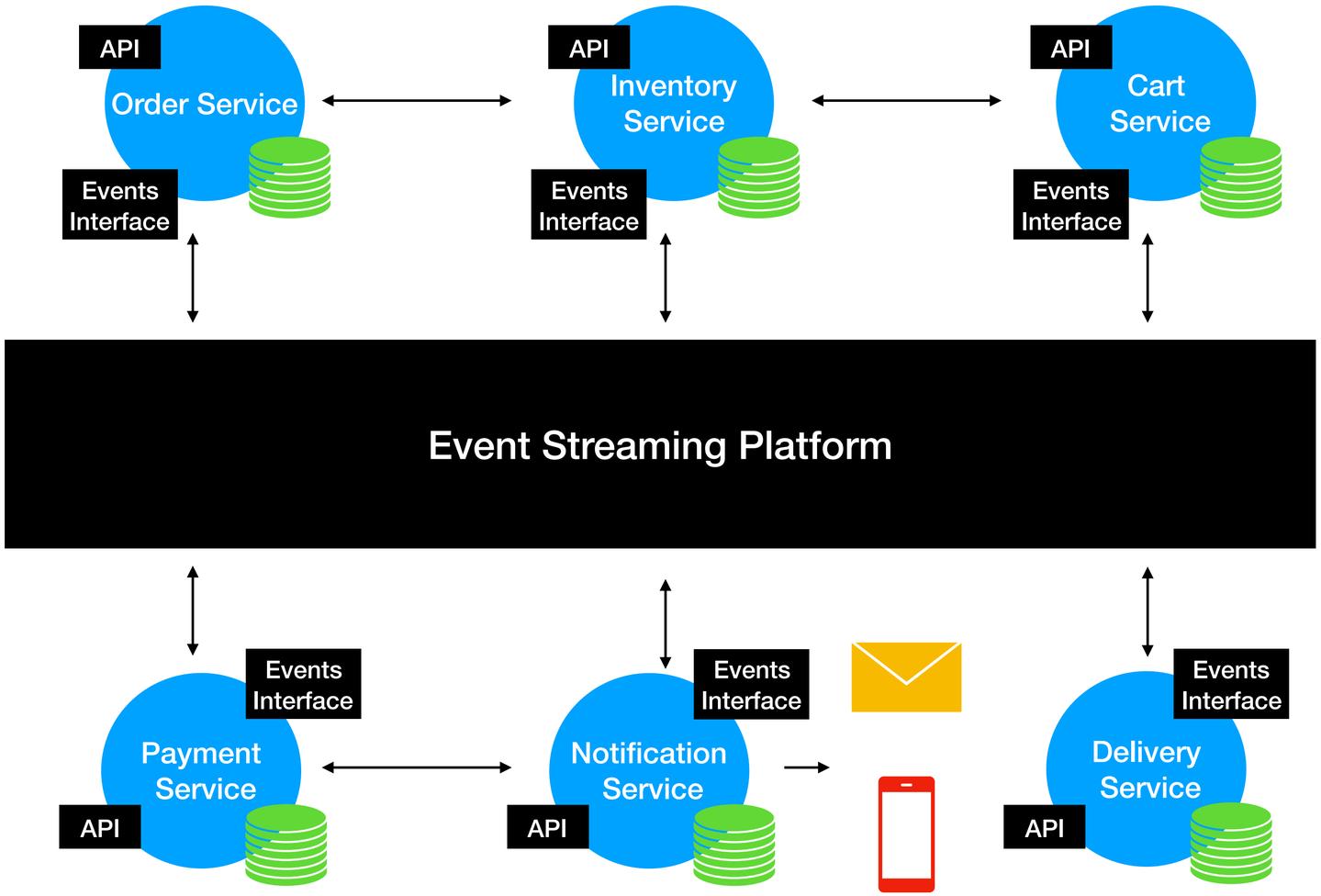
Past



Current



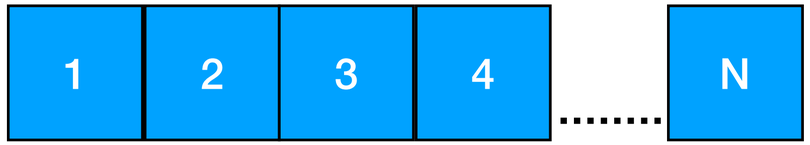
MicroServices Architecture



What is an Event Streaming Platform?

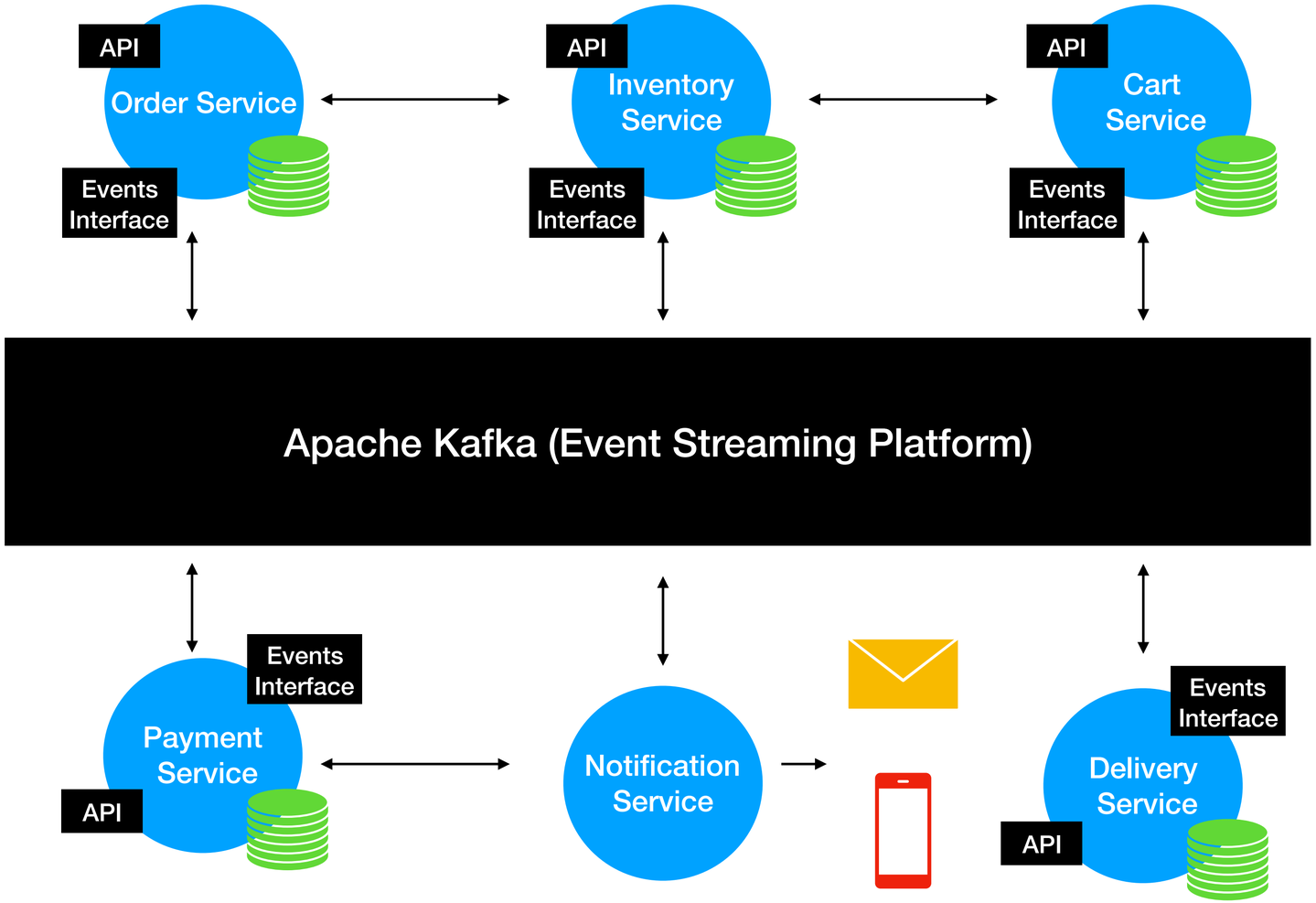
- Producers and Consumers subscribe to a stream of records

- Store stream of Events



- Analyze and Process Events as they occur

Apache Kafka (Event Streaming Platform)



Traditional Messaging System

- Transient Message Persistance
- 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

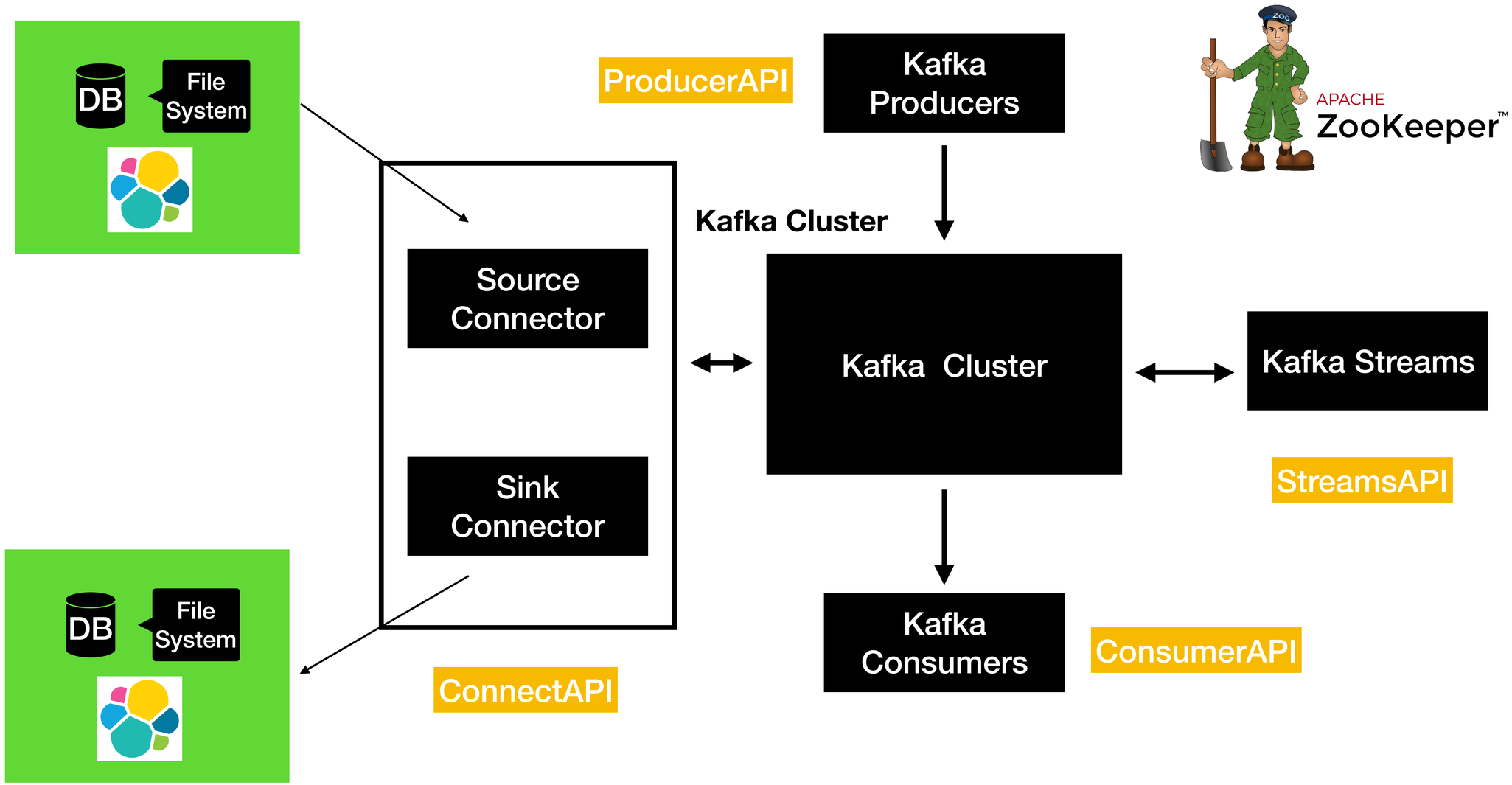
Tracking Online Order Deliveries

Banking

Fraud Transactions

New Feature/Product notifications

Kafka Terminology & Client APIs



Download Kafka

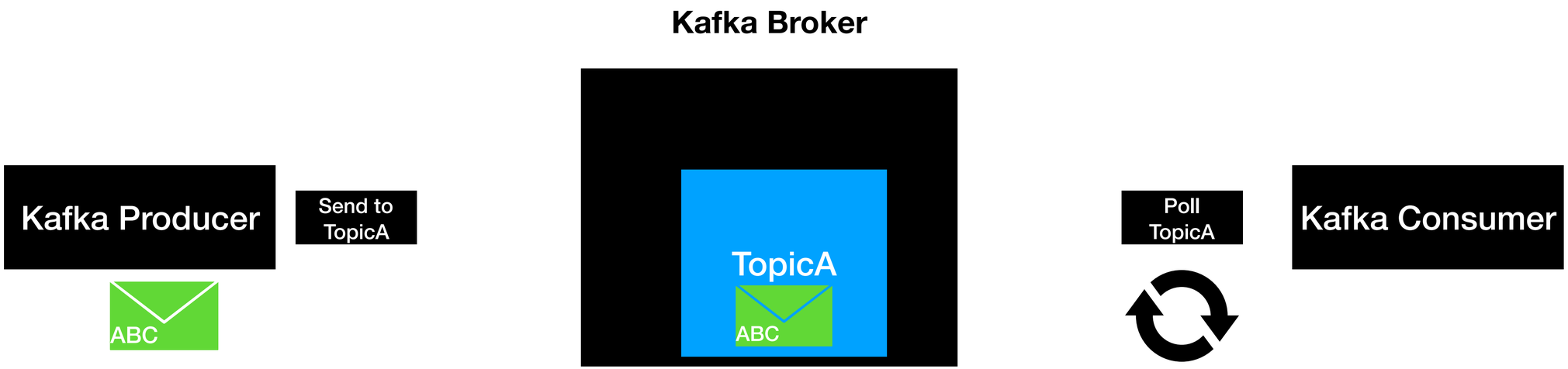
Kafka Topics
&
Partitions

Kafka Topics

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

Kafka Topics

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

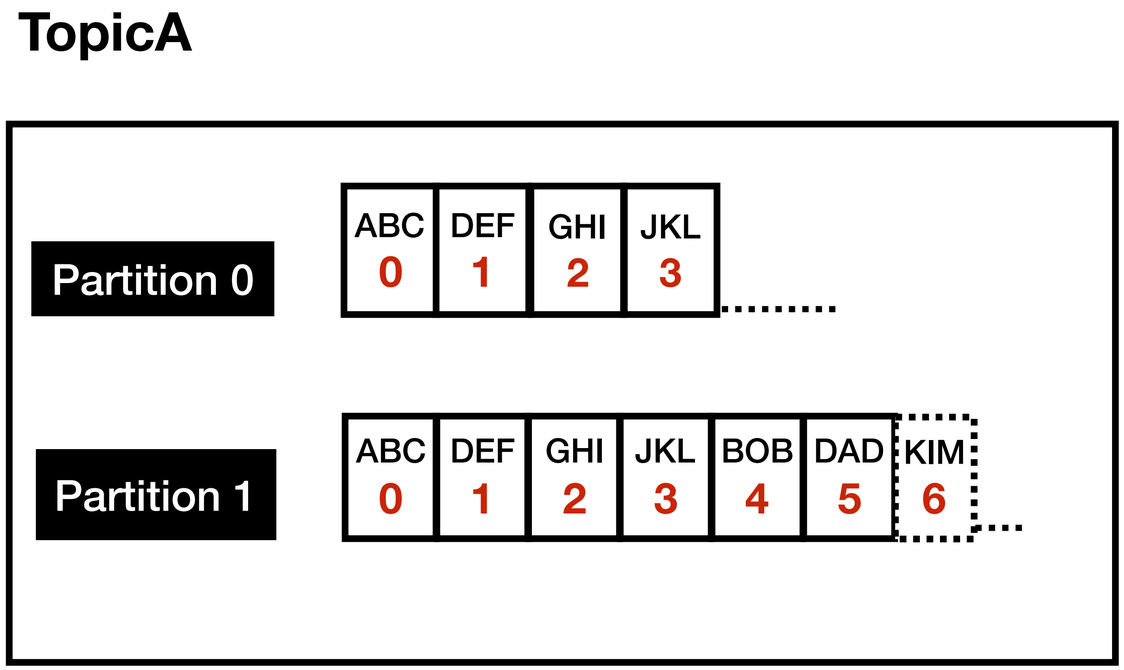


Topic and Partitions

- Partition is where the message lives inside the topic

- Each Topic will be create with one or more partitions

Topic and Partitions



- Each Partition is an ordered , immutable sequence of records

- Each record is assigned a sequential number called ***offset***

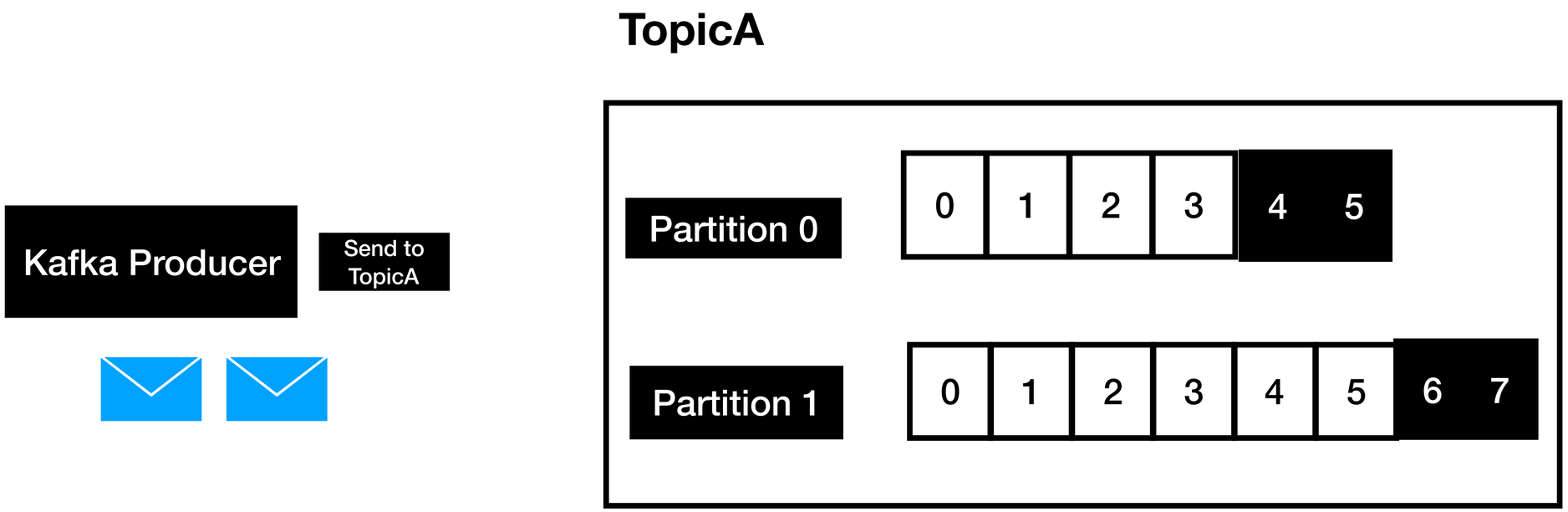
- Each partition is independent of each other

- 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

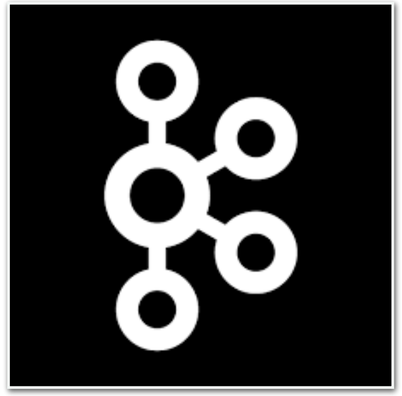
Topics and Partitions



Setting up Zookeeper & Kafka Broker

Setting up Kafka in Local

APACHE
ZooKeeper™



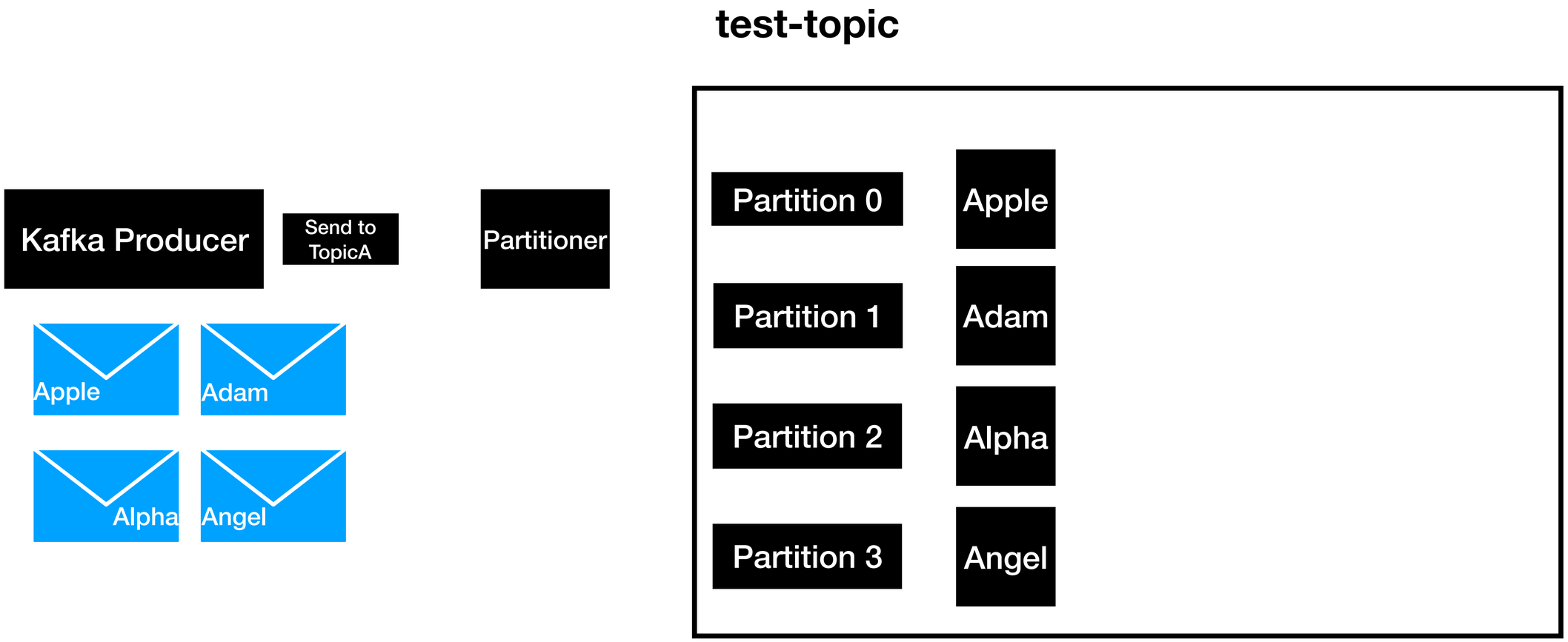
Broker registered with zookeeper

Sending Kafka Messages With Key and Value

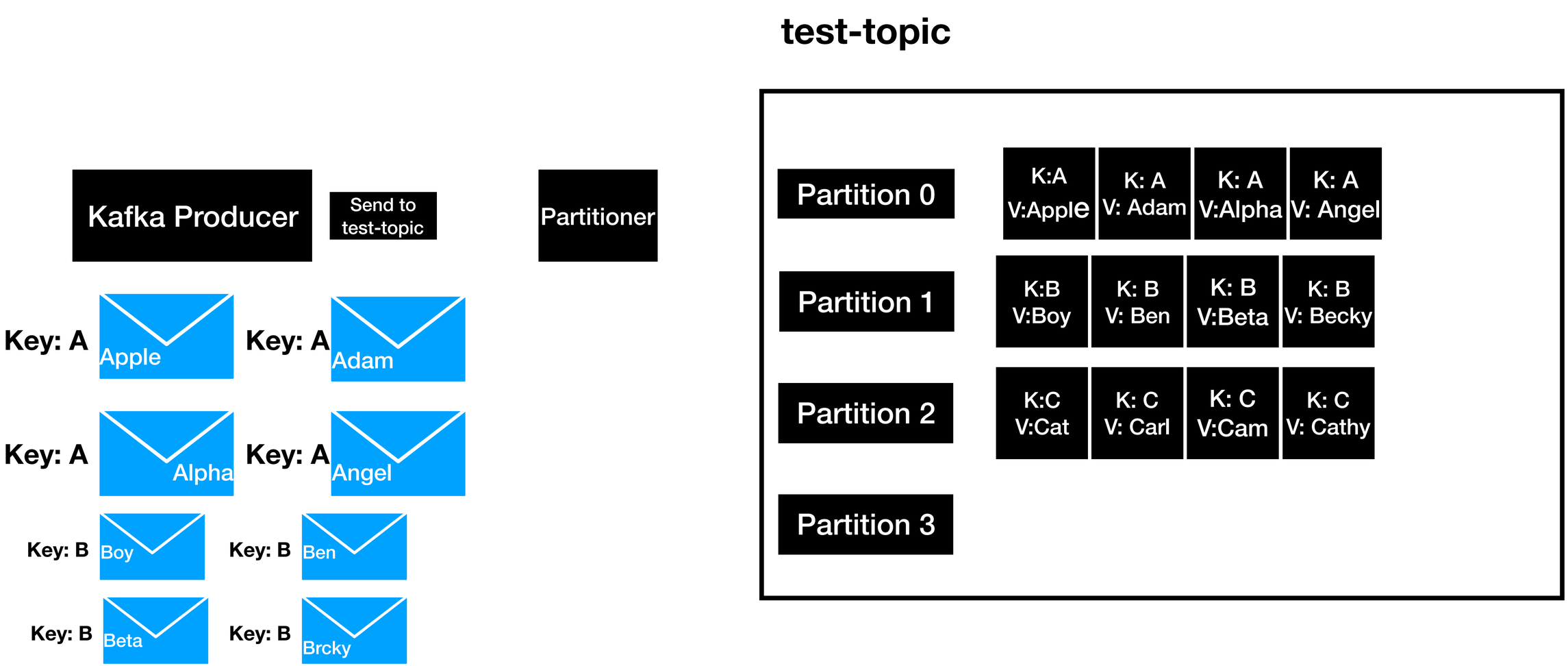
Kafka Message

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

Sending Message Without Key



Sending Message With Key



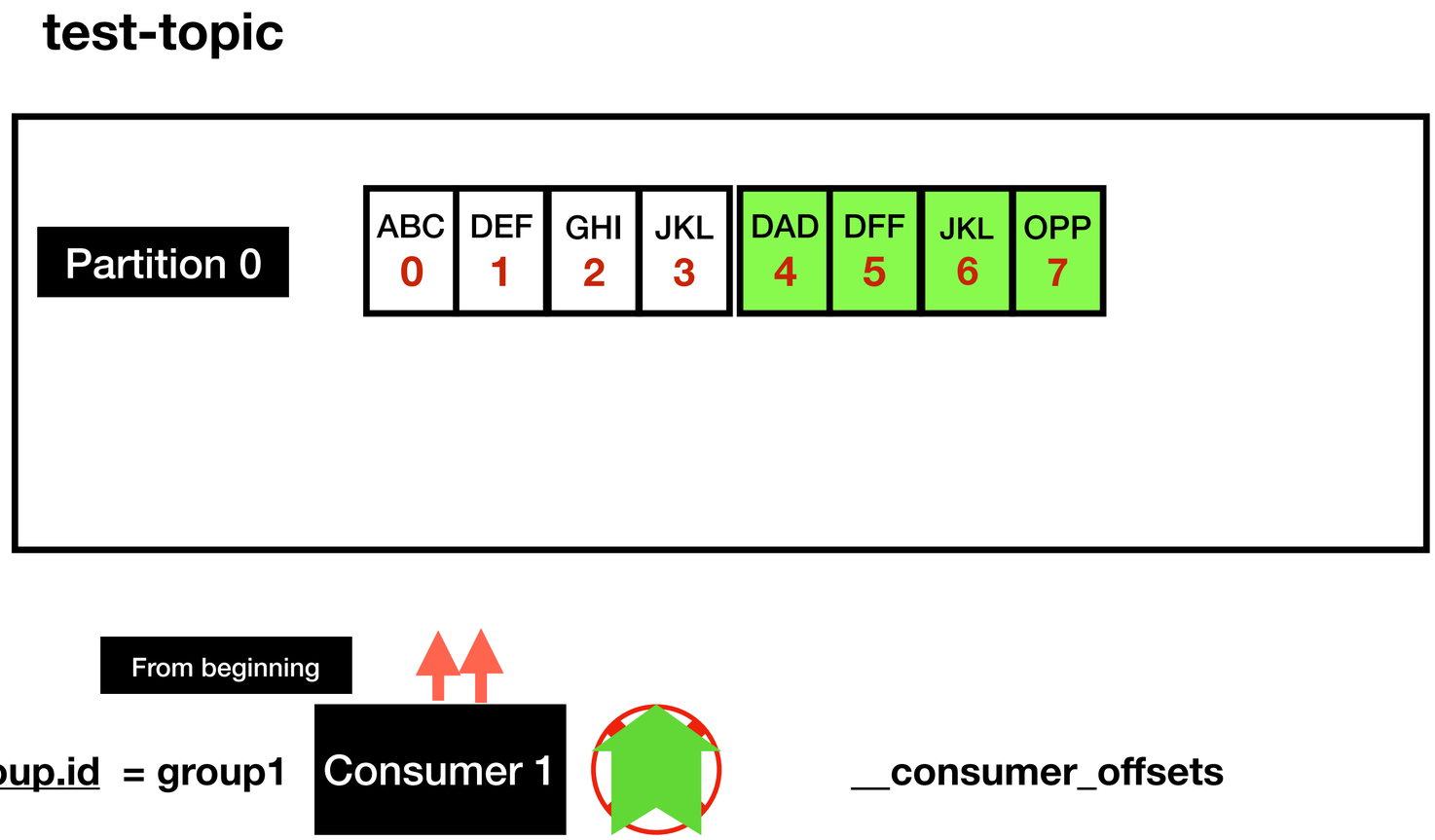
Consumer Offsets

Consumer Offsets

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

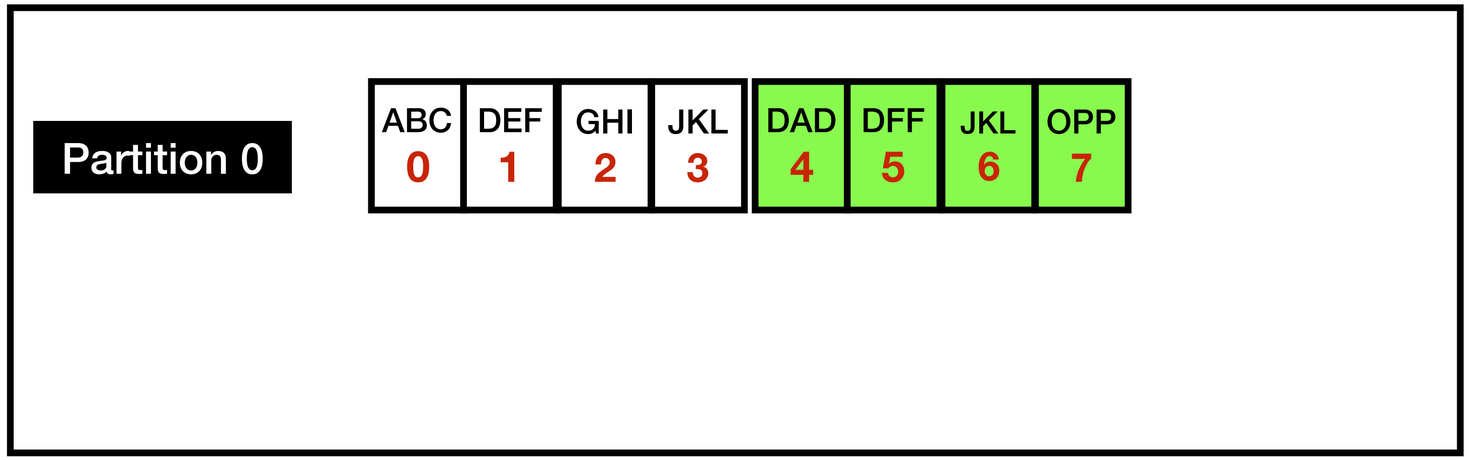
s option will be done through only programmatic wa

Consumer Offsets



Consumer Offsets

test-topic



From beginning

group.id = group1

Consumer 1

__consumer_offsets

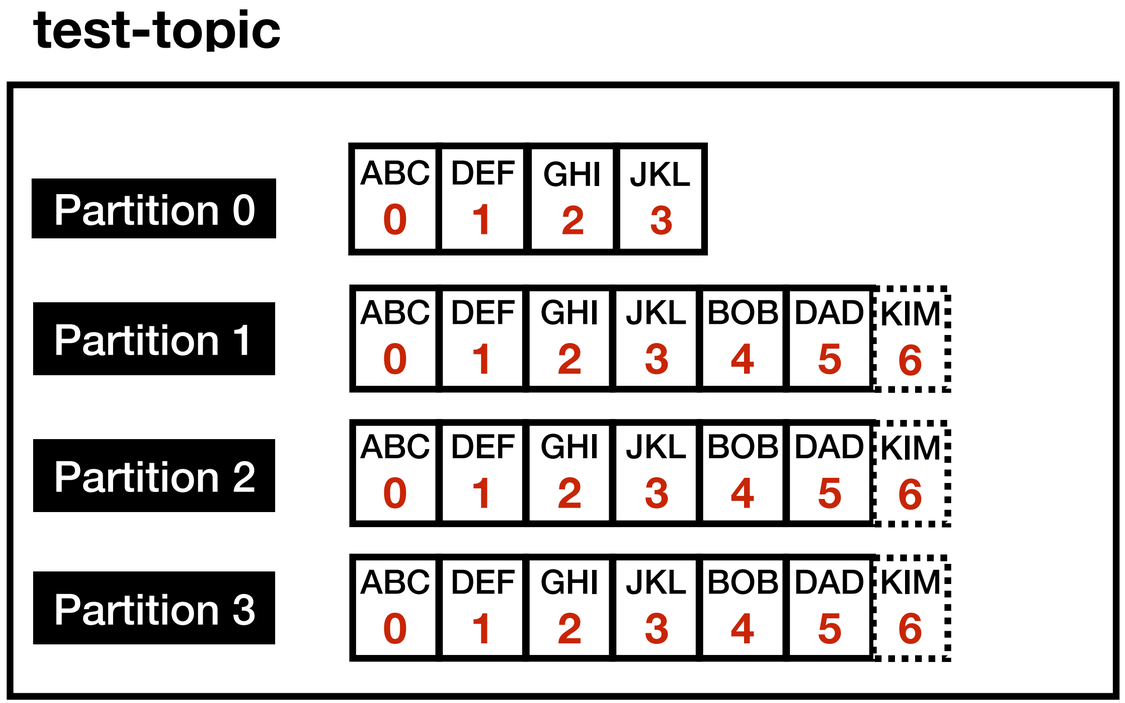
Consumer Offset

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

Consumer Groups

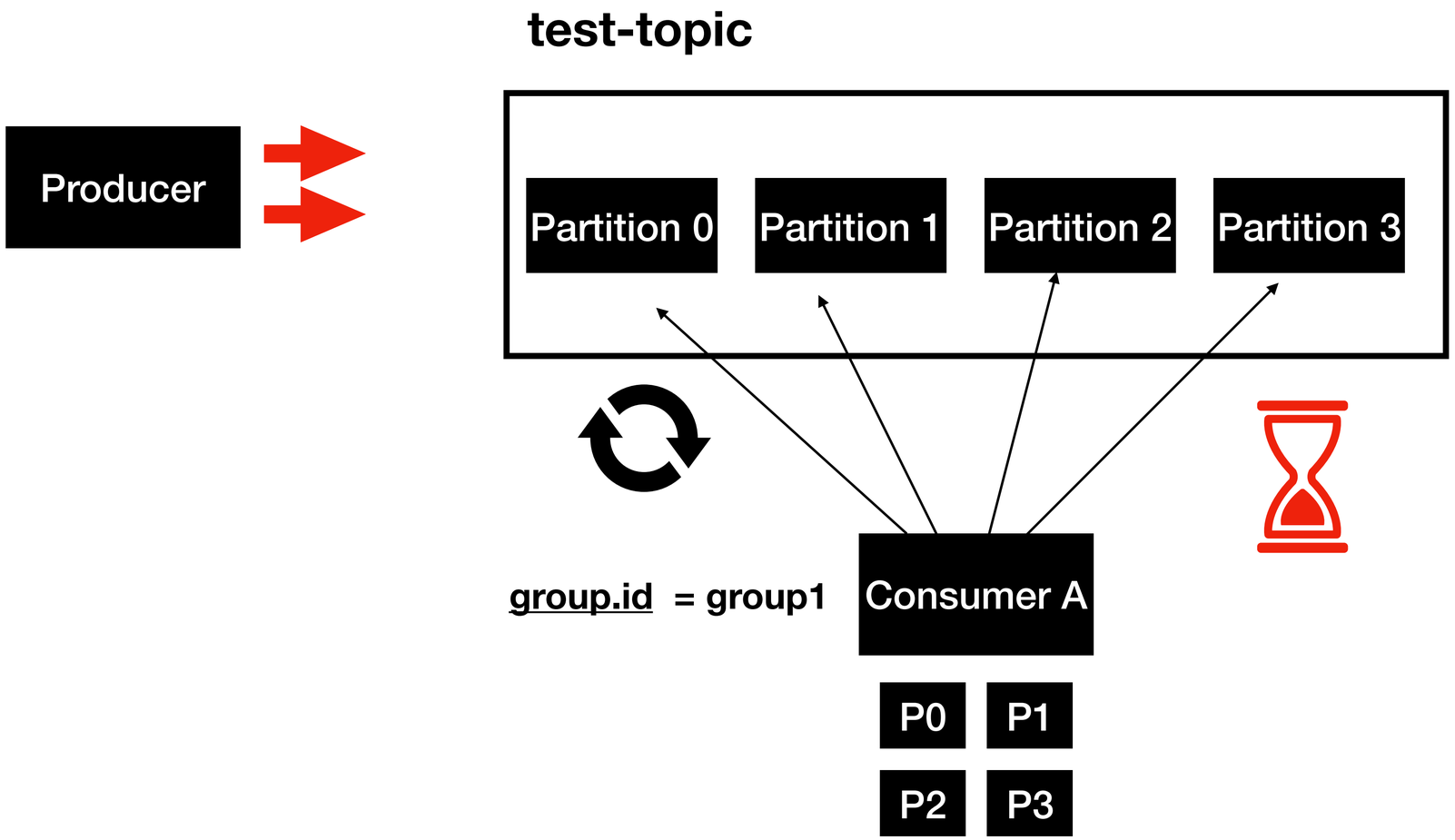
Consumer Groups

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

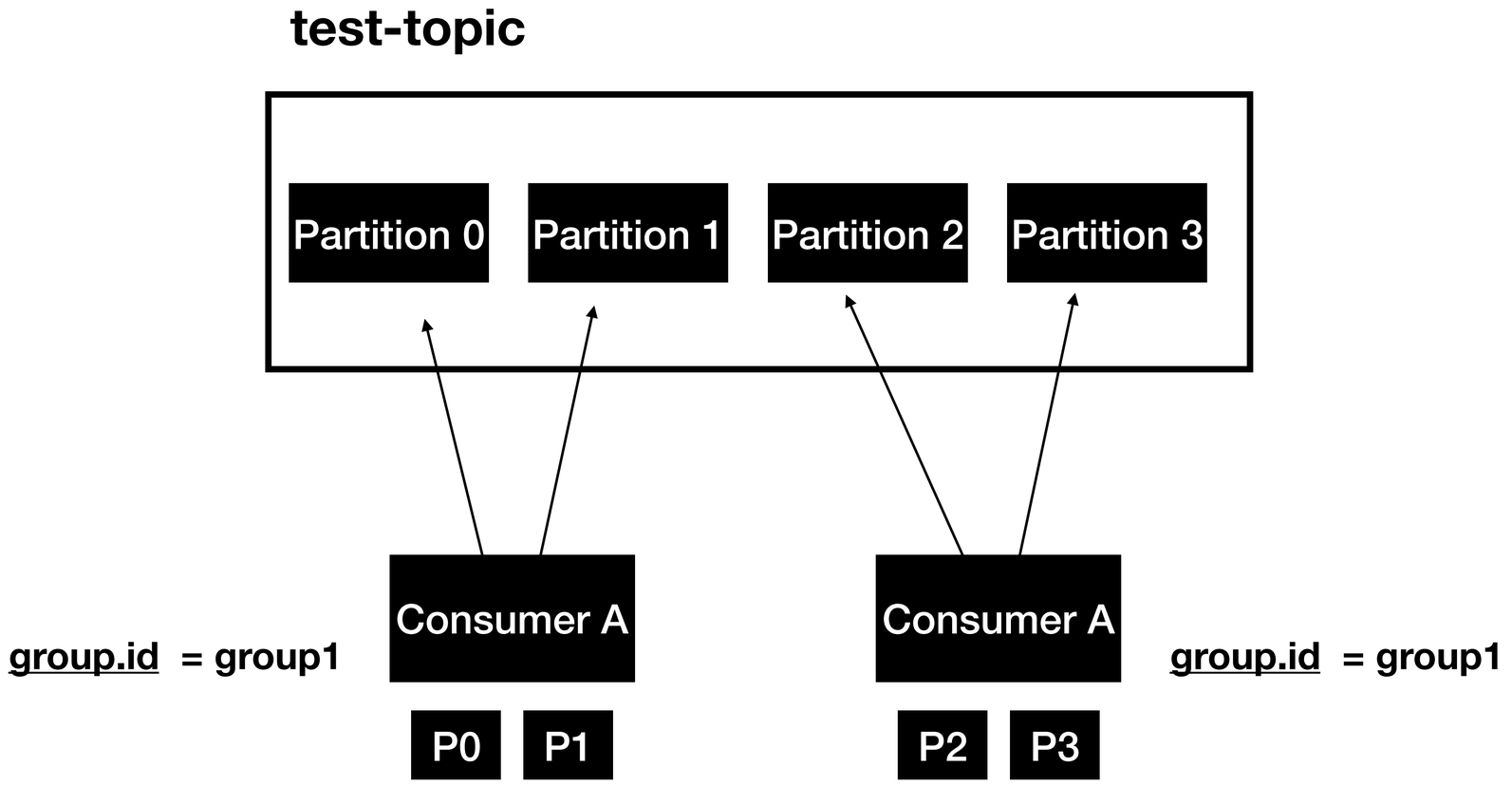


group.id = group1 Consumer 1

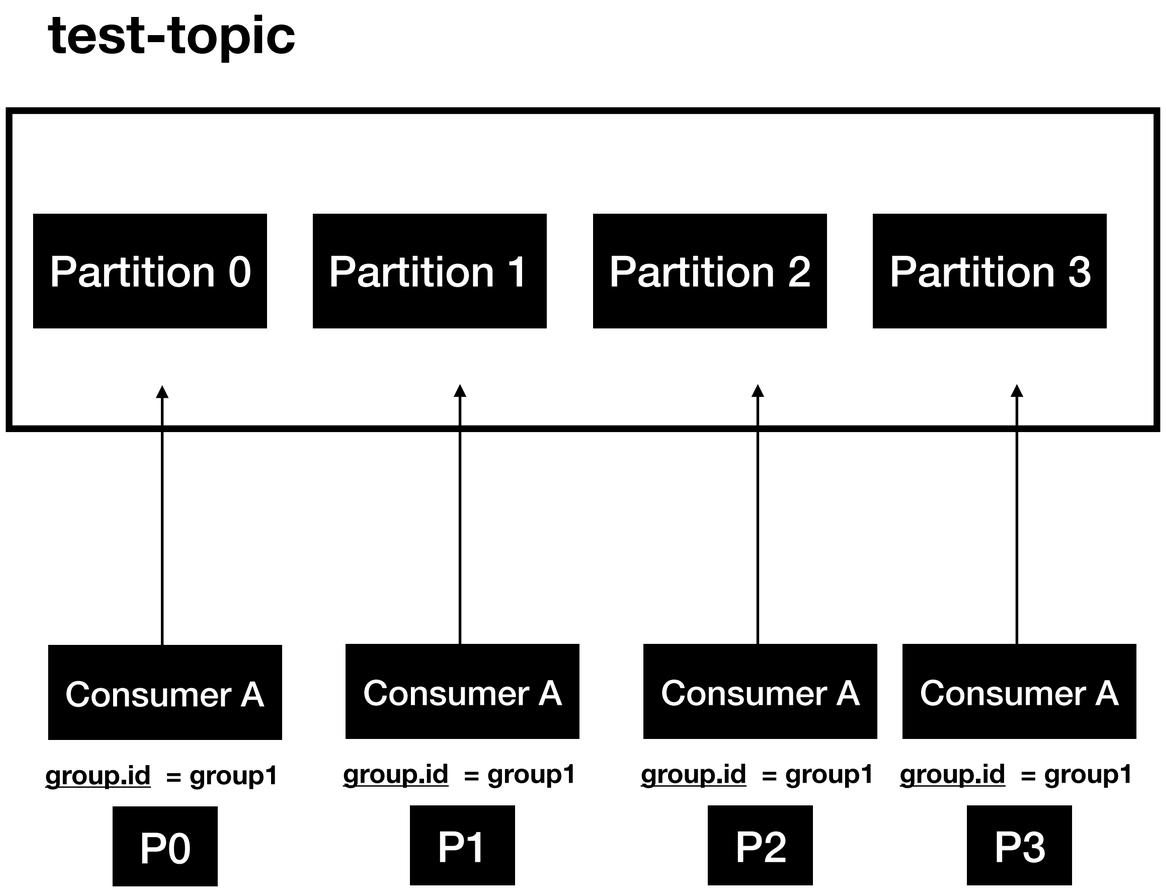
Consumer Groups



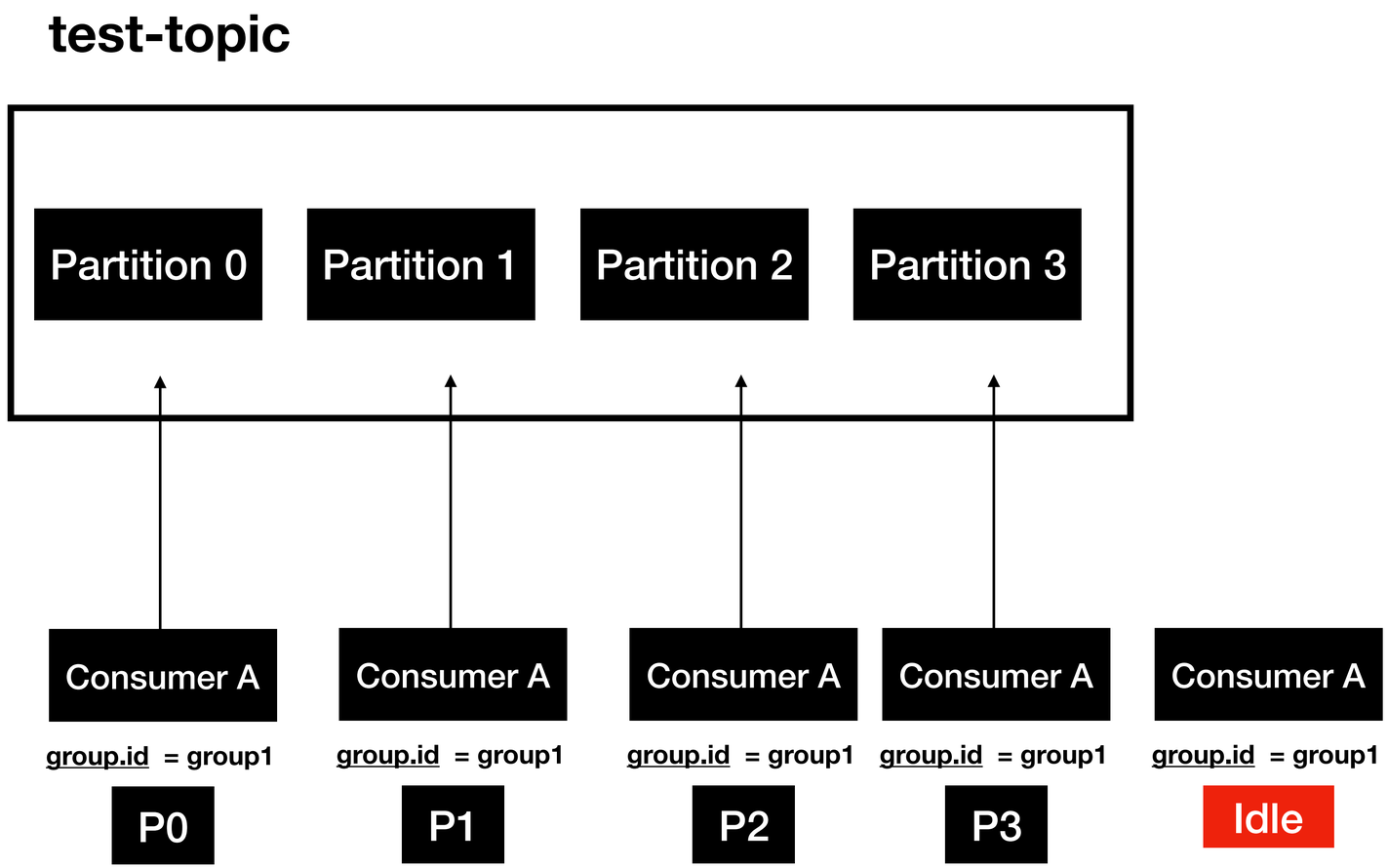
Consumer Groups



Consumer Groups



Consumer Groups

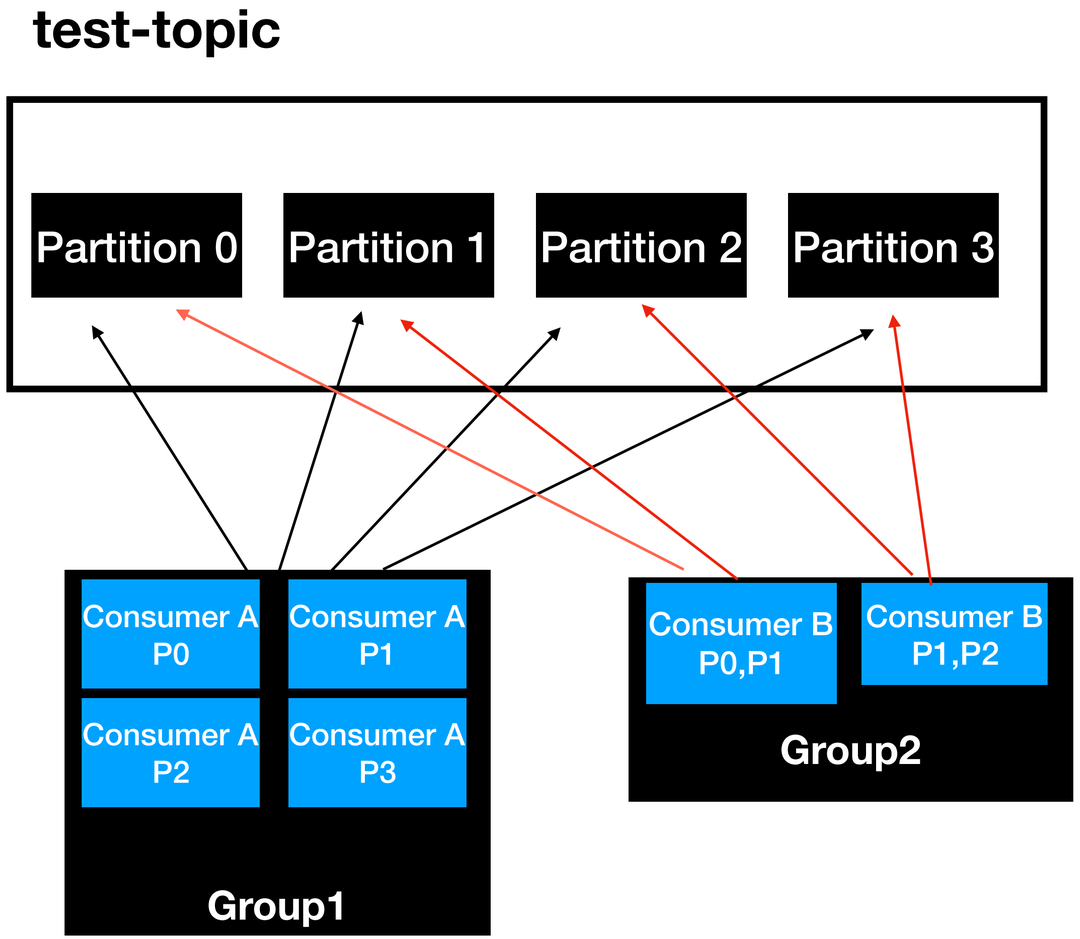


Consumer Groups

In order to test this scenario, open two consumer group window pointing to same consumer group.

Start the console producer to produce message. In this case, your message will go to respective consumer group and based on the number of partitions.

since, we have created two console consumer pointing to same consumer group, each consumer group will have two partitions.



Consumer Groups : Summary

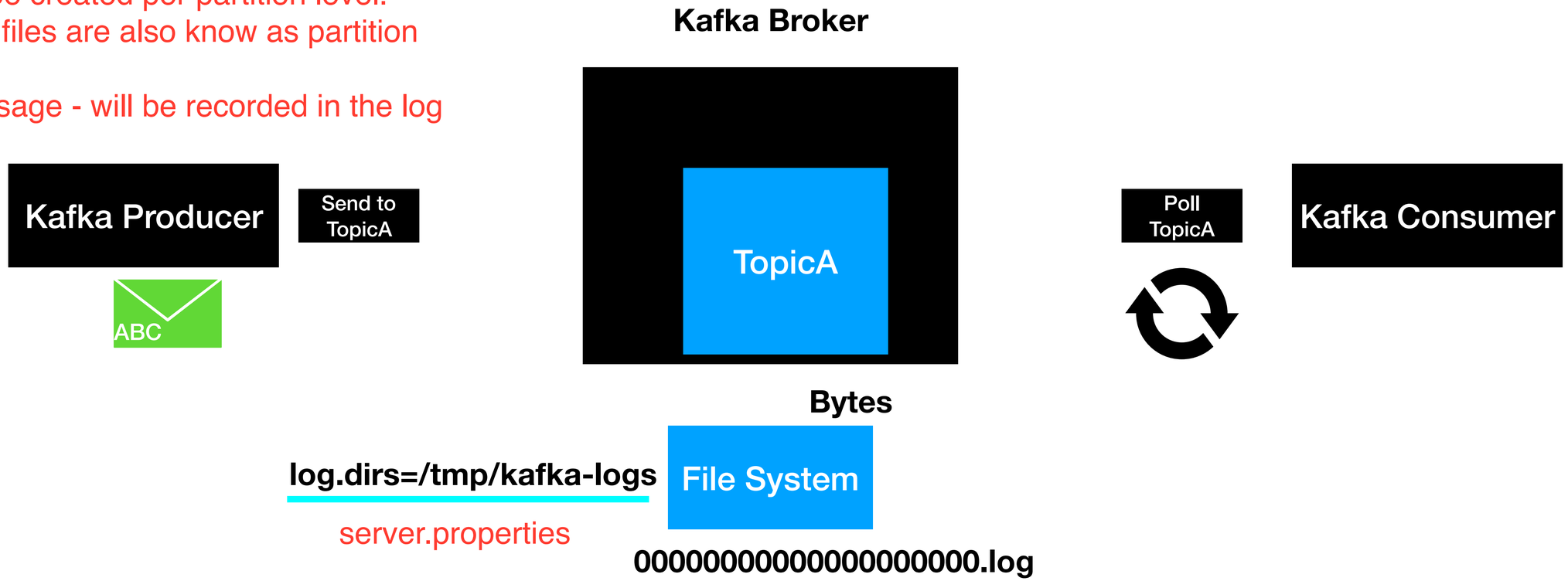
- Consumer Groups are used for scalable message consumption
- Each different application will have a unique consumer group
- Who manages the consumer group?
  - Kafka Broker manages the consumer-groups
  - Kafka Broker acts as a Group Co-ordinator

Commit Log
&
Retention Policy

Commit Log

Producer:
* Kafka Producer will produce a message.
* Each message will send to a Topic in the Kafka Broker.
* Once message reached to respective topic in Broker, message will be stored in the file system in a binary format (bytes)
* It is called as Commit Log
* If you have more than one partition, then log file will be created per partition level.
* These log files are also know as partition commit log
* Each message - will be recorded in the log file

Consumer:
* When the consumer who is polling for new records can only see the records that are committed to the file system.
* As the new records are published on the topic then the records get appended to the log file.



Navigate to /bin directory and execute this command: To see more detailed log information.
./kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --files /tmp/kafka-logs/test-topic-0/00000000000000000000.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)

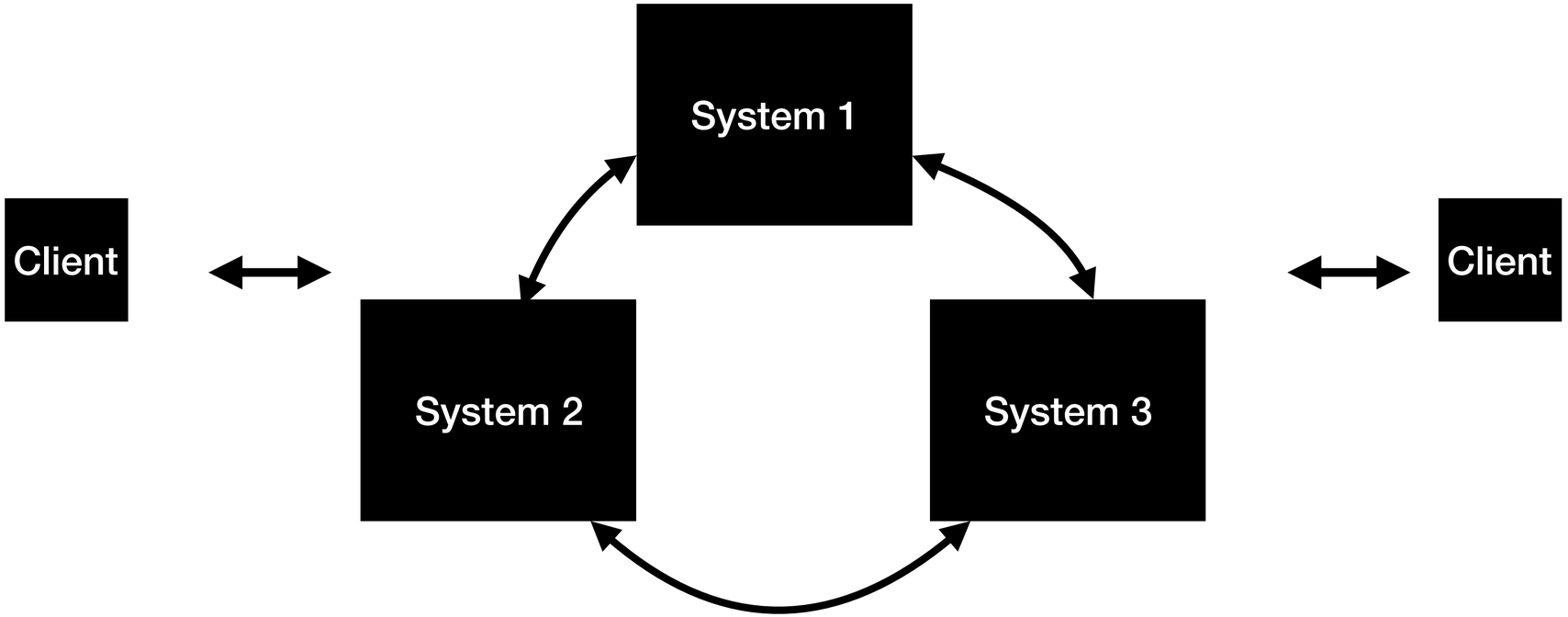
Kafka
as a
Distributed Streaming System

Apache Kafka® is *a distributed streaming platform*

Notes:
What is a distributed system and its characteristics?
Distributed system in general are a collection of systems work and interact together in order to deliver the functionality.

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

  Let's say one of the system is down.
  Still this won't impact the overall availability of the systems.
  Even in this case the client request will be handled gracefully.

- Reliable Work Distribution

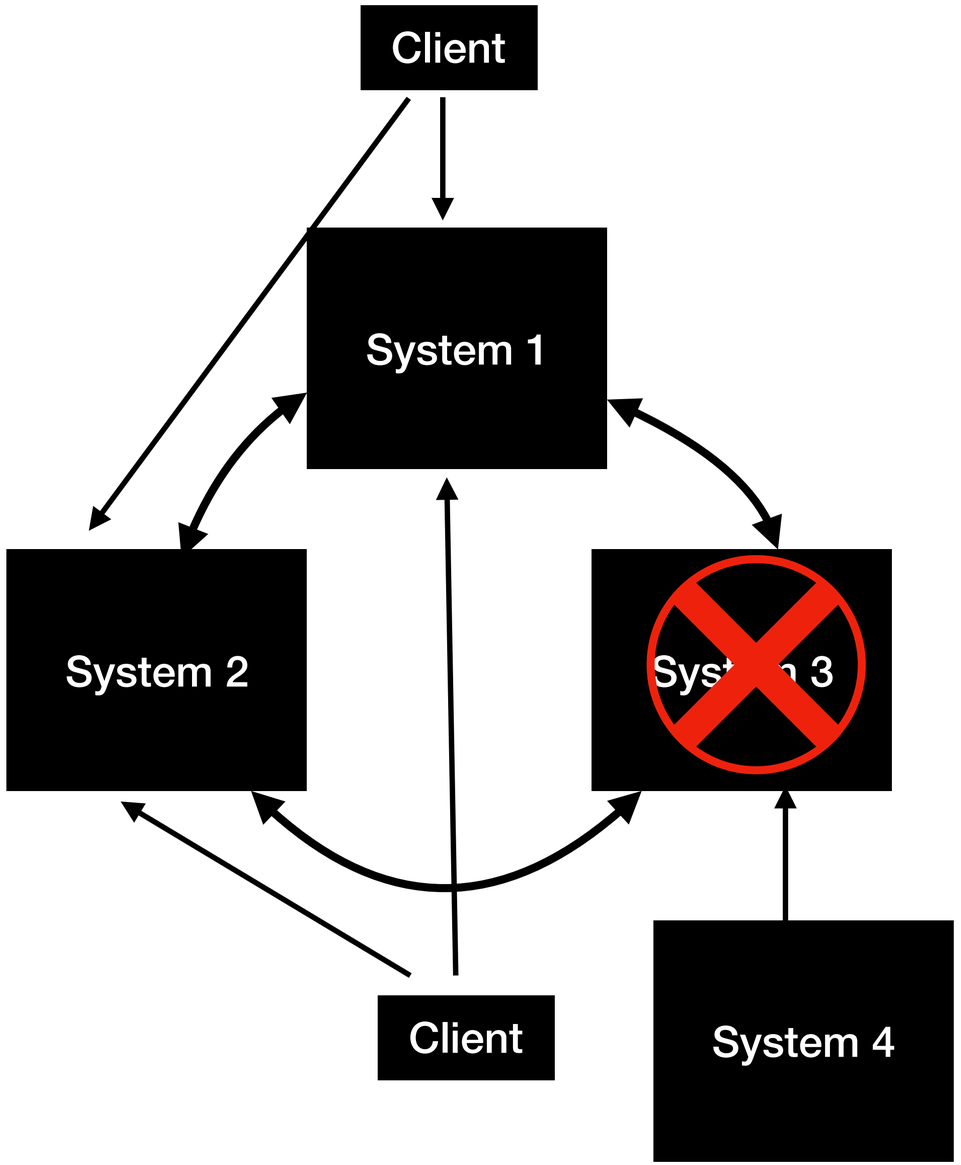
  The requests that received from the clients in general are equally distributed between the available systems.

- Easily Scalable

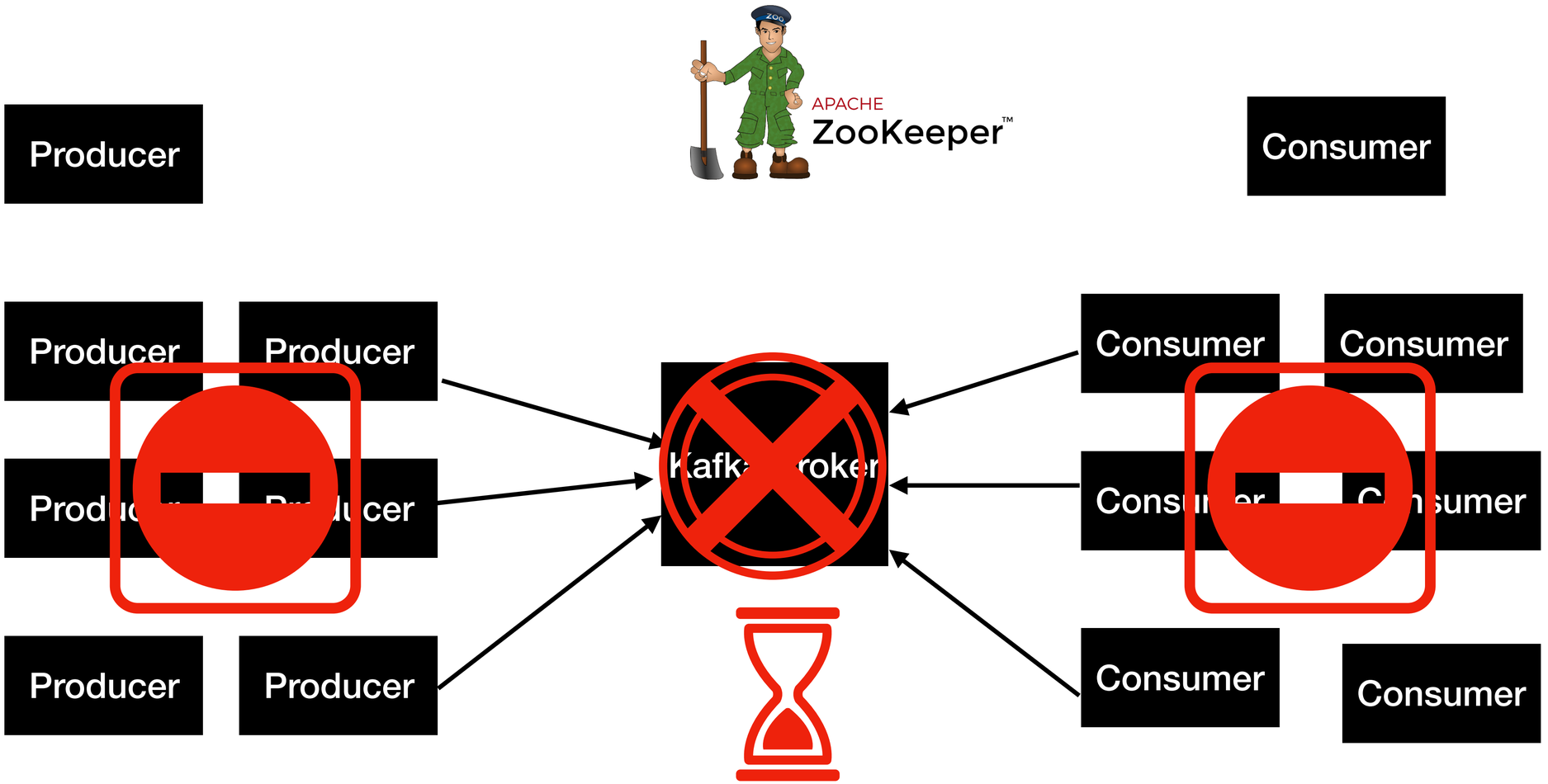
  Adding new systems to the existing setup is really easy.

- Handling Concurrency is fairly easy

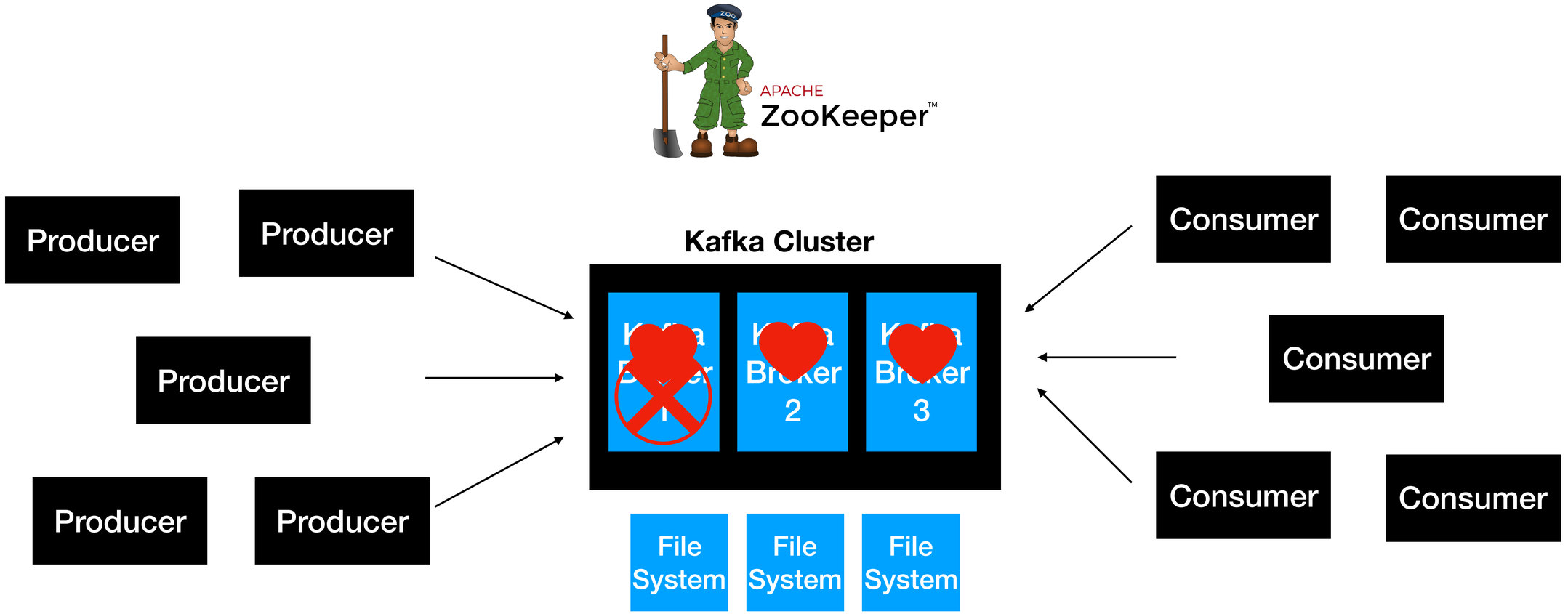
  Which can be handled fairly easy but distributed systems.



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)
```

Similarly create “server-2” properties which contains broker id=2 and port 9094 and different log directory

Start all 3 Kafka brokers

Once all the brokers are started, start create topic

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

here, “—replication-factor 3” is the number which is <= number of clusters. In ours case we have 3 borkers.

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
```

1

Create topic command issue to zookeeper

APACHE ZooKeeper™

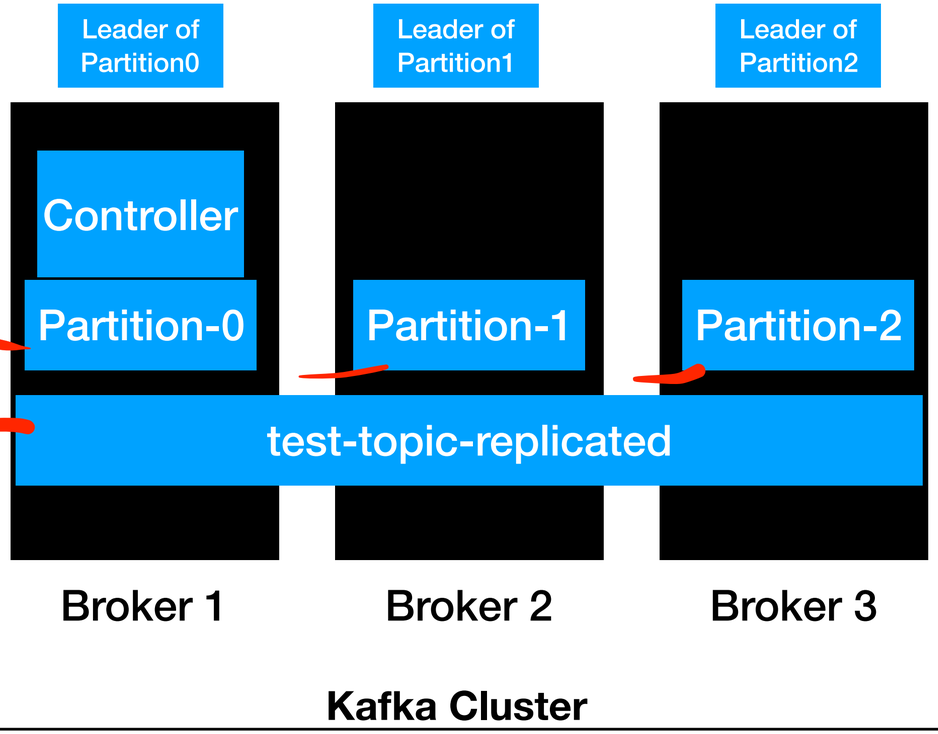
2

The role of this controller is to distribute the ownership of the partitions to the available broker.

controller

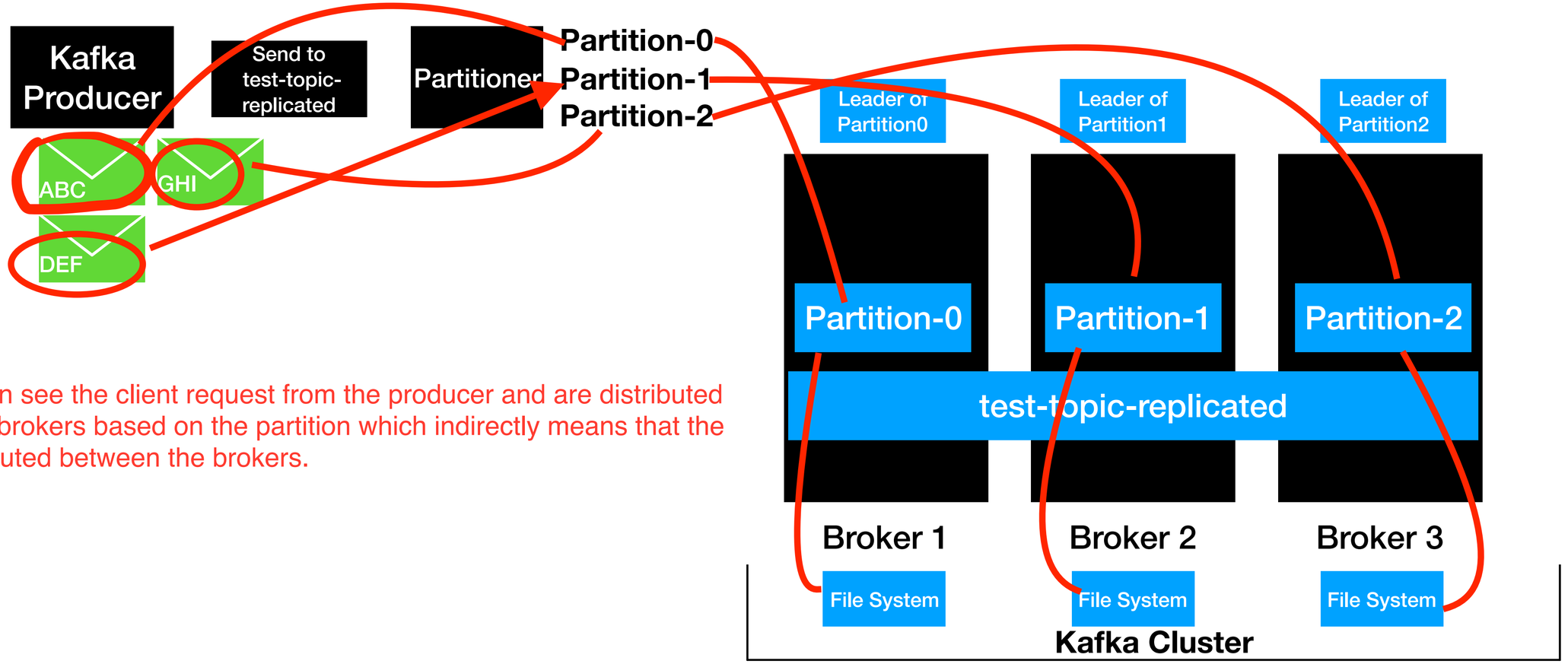
3

distribute



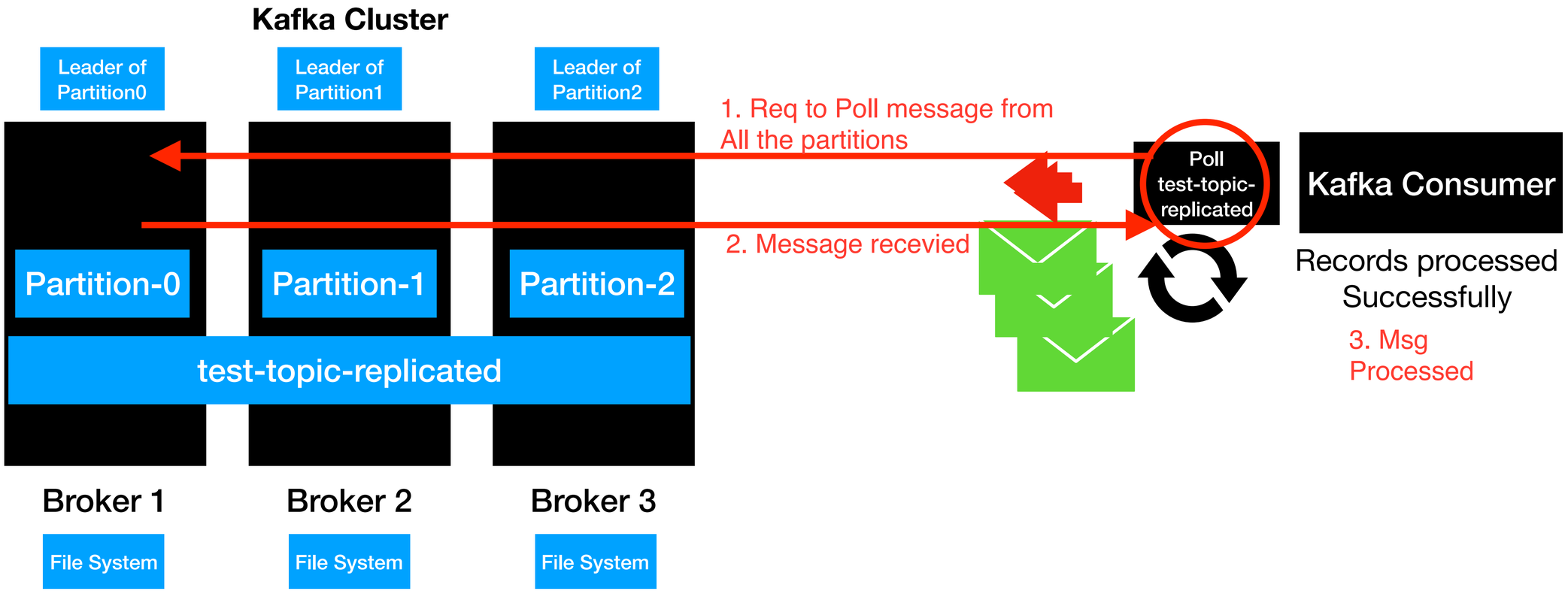
In distributed systems the concept of distributing partitions to the brokers is called a leader assignment.

How Kafka Distributes Client Requests?
Kafka Producer

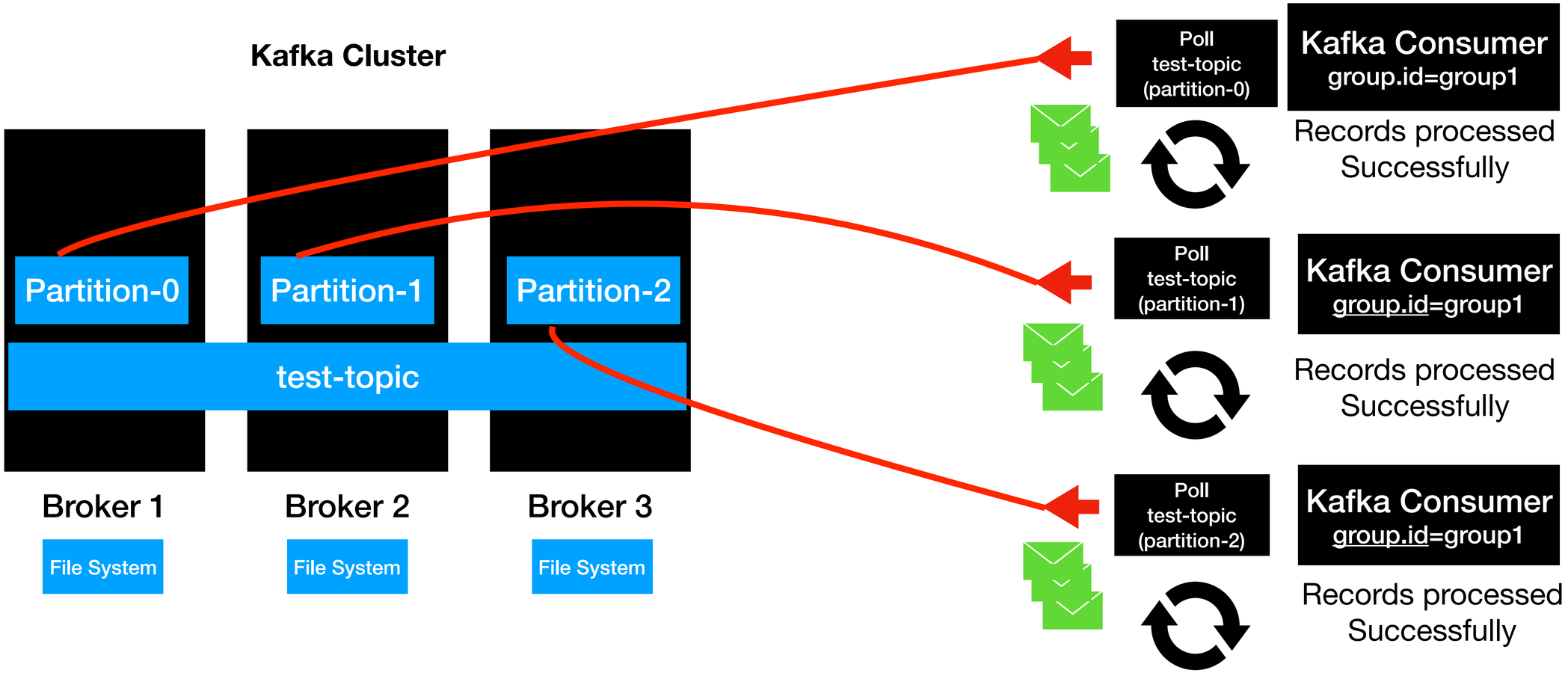


So as you can see the client request from the producer and are distributed between the brokers based on the partition which indirectly means that the load is distributed between the brokers.

How Kafka Distributes Client Requests? Kafka Consumer



How Kafka Distributes Client Requests?
Kafka Consumer Groups

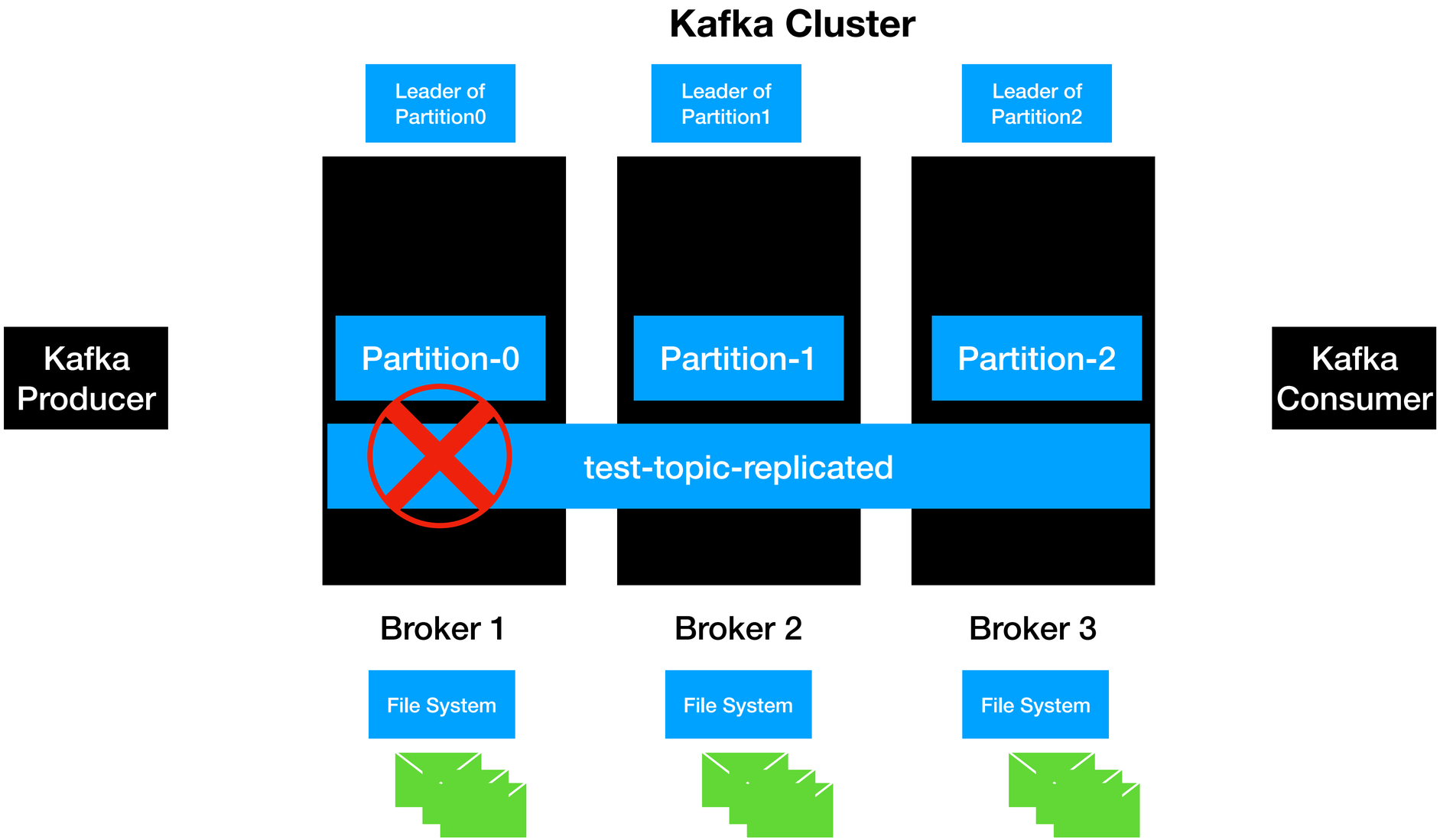


Summary : How Kafka Distributes the Client Requests?

- 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?

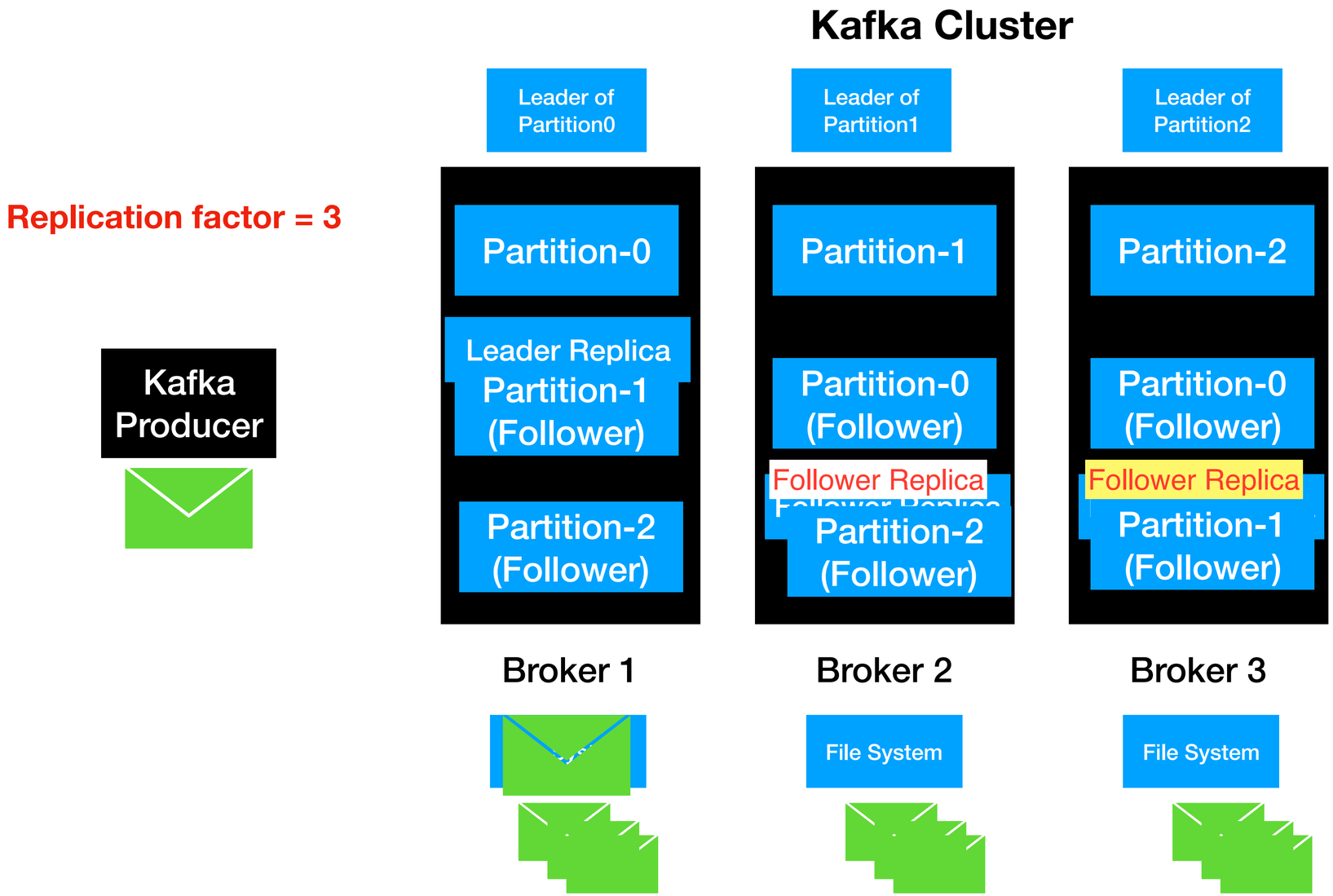


Replication

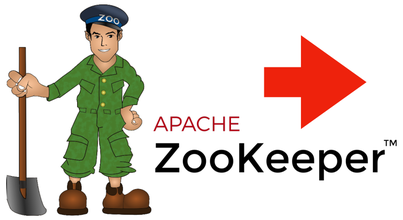
```
./kafka-topics.sh -
-create --topic test-topic-replicated
-zookeeper localhost:2181
--replication-factor 3
 --partitions 3
```

So replication factors equal to number of copies of the same message.

Replication



Replication



Kafka Cluster

Leader of Partition0

Leader of Partition 0 & 1

Leader of Partition2

Partition-0

Partition-1

Partition-2

Partition-0 (Follower)

Partition-2 (Follower)

Partition-0 (Follower)

Partition-2 (Follower)

Partition-1 (Follower)

Kafka Producer

Kafka Consumer

If broker1 is down. In this case, ZooKeeper will get a notification and it will assign Broker2 as a leader for both partition-0 and partition-1

Broker 1

Broker 2

Broker 3

File System

File System

File System

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

ISR: will be use to sync the data across all the available broker. Suppose you have 3 brokers, namely 0,1,2. If 2nd broker is down, the In-Sync Replica will pointing to other brokers automatically based on the signal provided by ZooKeeper.

./kafka-topics.sh --zookeeper localhost:2181 --describe --topic test-topic-replicated

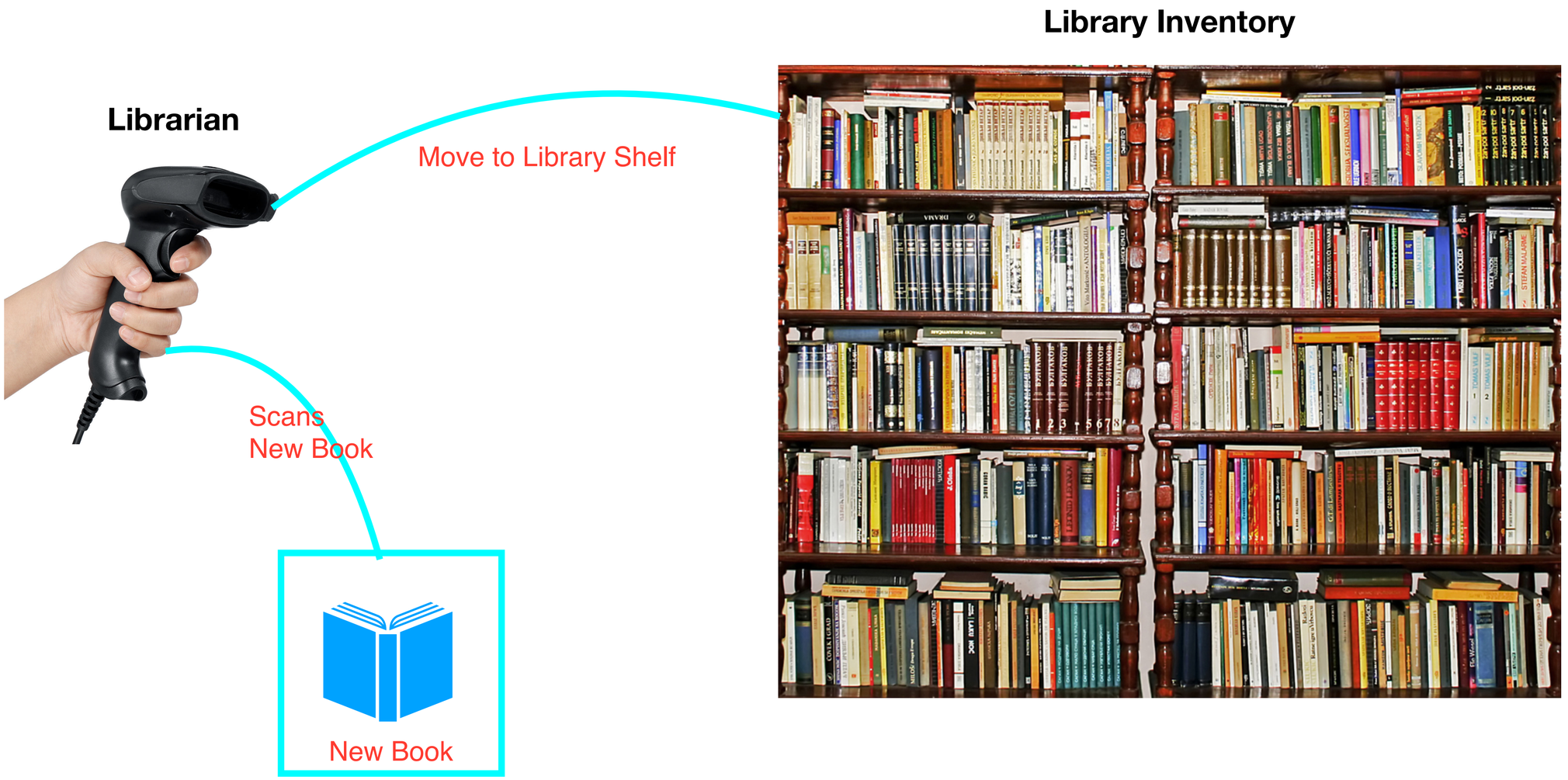
Fault Tolerance
&
Robustness

When you have more than one Kafka broker and start producing messages, if consumers connected to your Kafka broker it will start consume messages.
Suppose, one of the borker is down, it will not impact the consumers. Consumers still can receive the messages. (Robustness)
Kafka takes care the error handling. From the client perspective you no need to do anything.

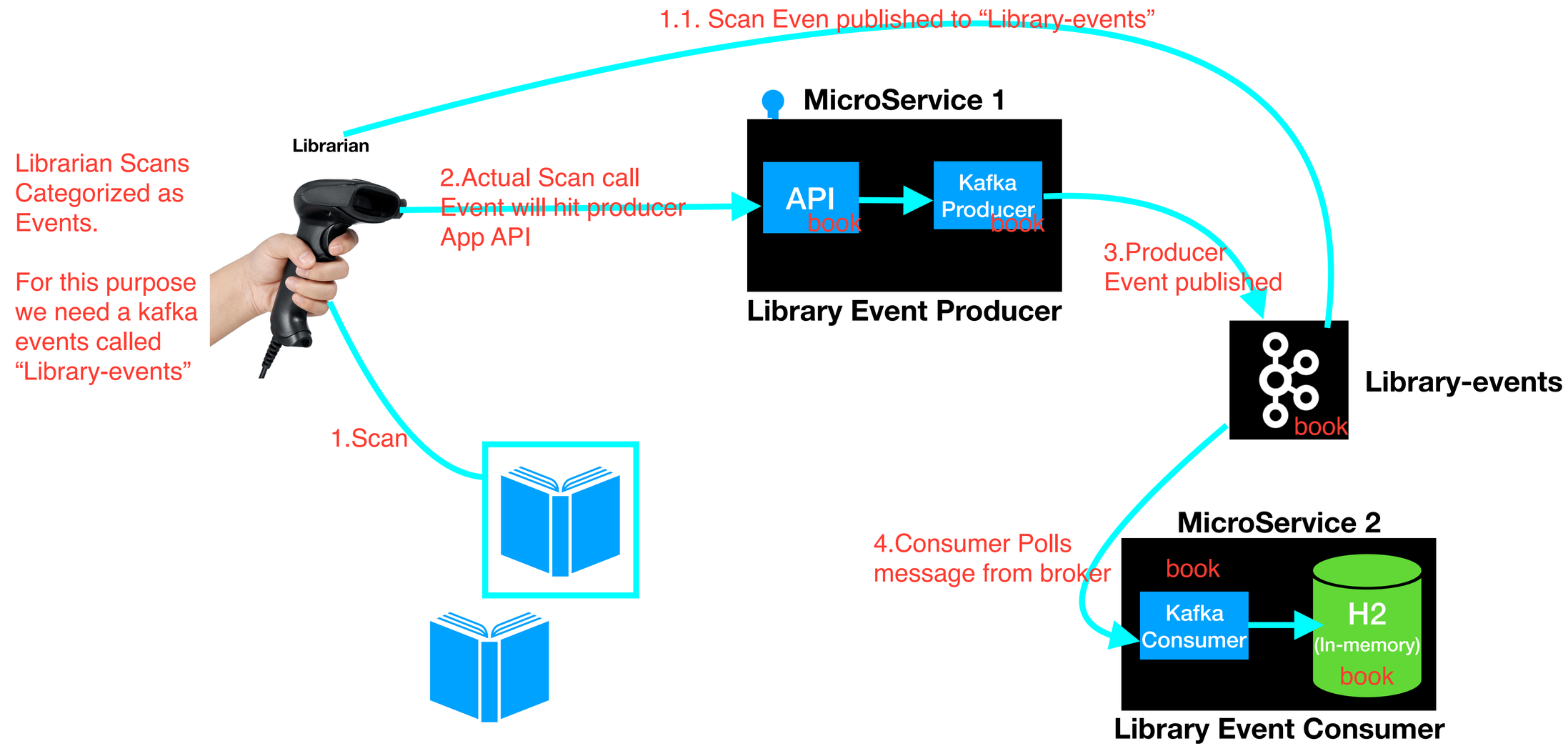
Application Overview

Library Inventory

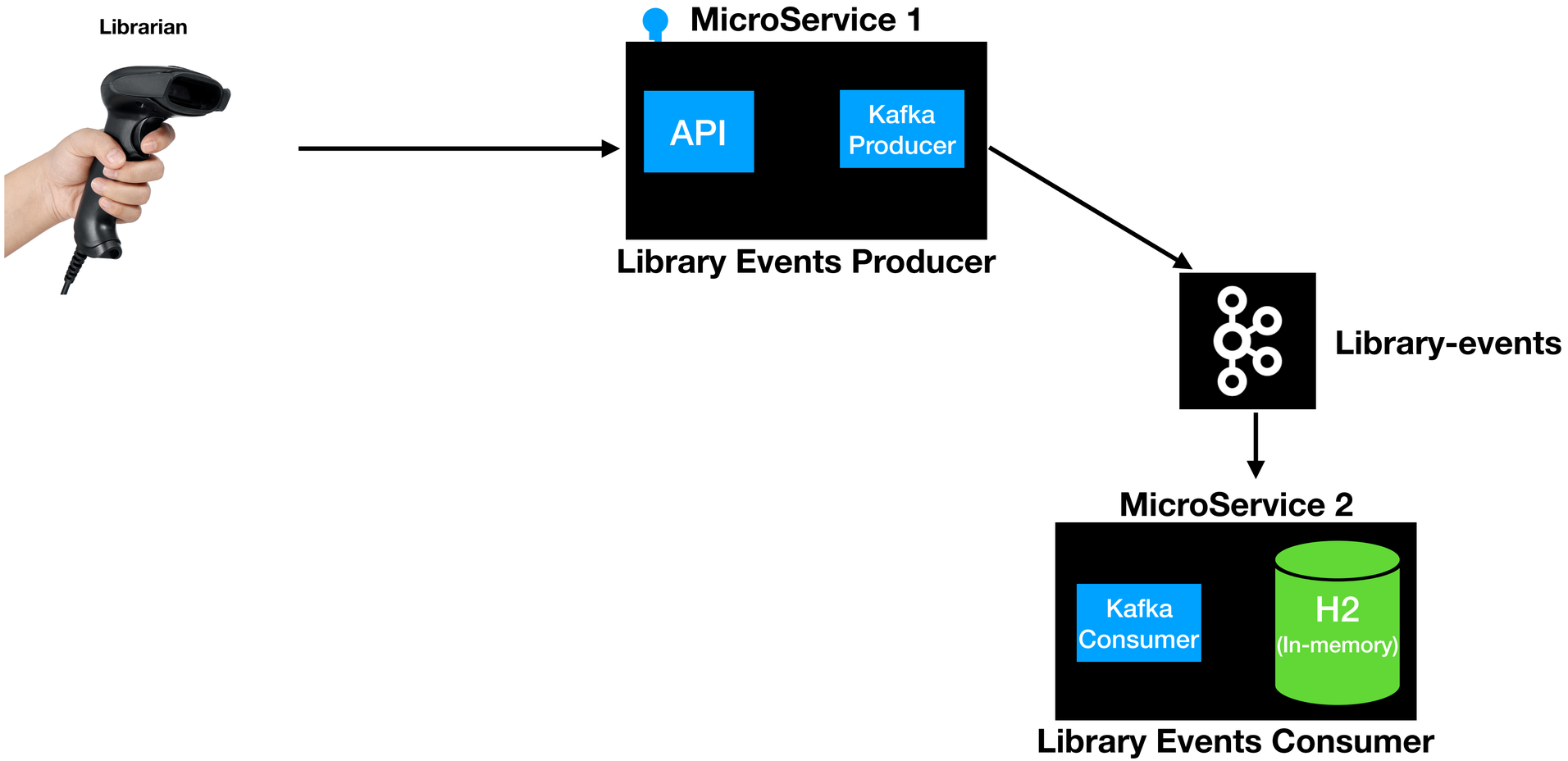
Library Inventory Flow



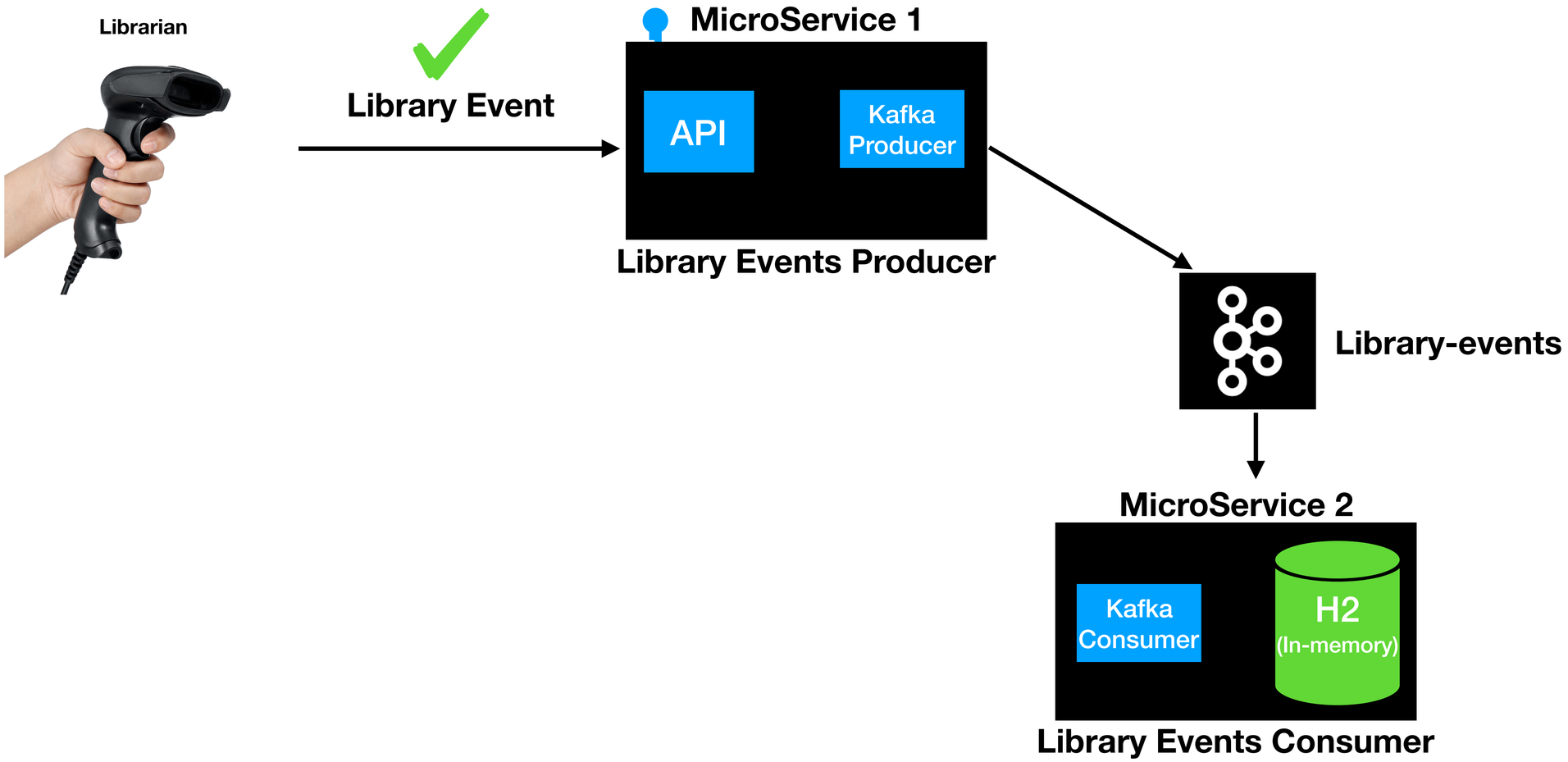
Library Inventory Architecture



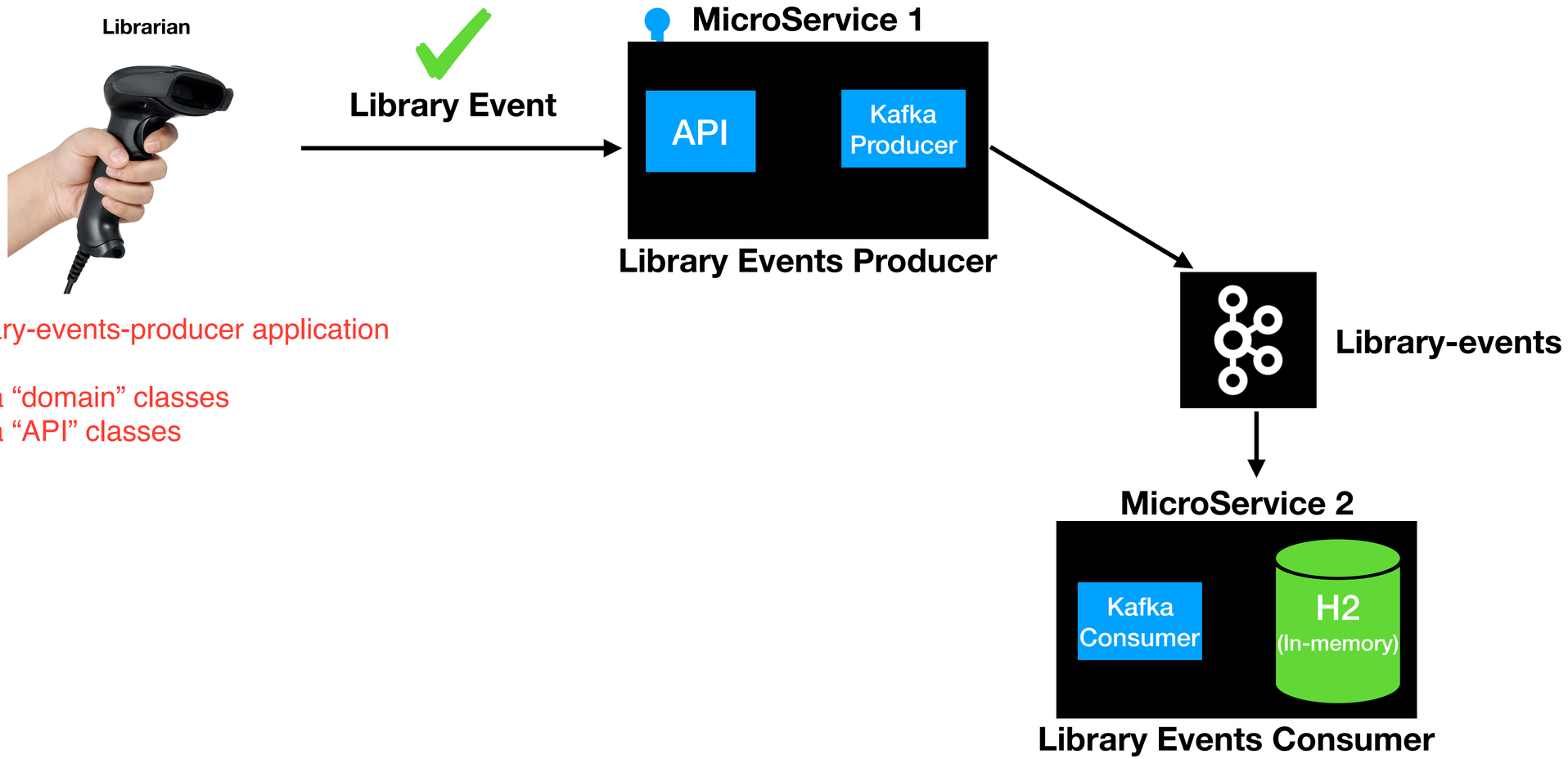
Library Inventory Architecture



Library Event Domain



Library Event Domain



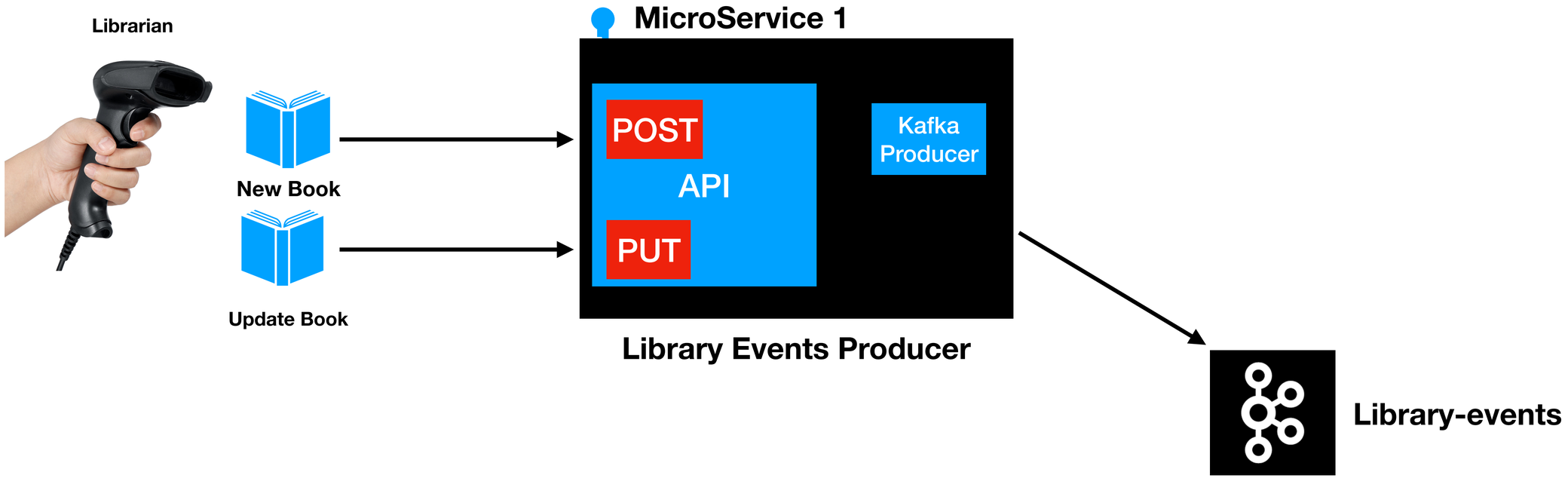
Refer: library-events-producer application

1. Create a “domain” classes
2. Create a “API” classes

Library Events Producer API

POST WITH-OUT STATUS

```
curl -i \
-d '{"libraryEventId":null,"book":{"bookId":456,"bookName":"Kafka Using Spring Boot","bookAuthor":"Dilip"}}' \
-H "Content-Type: application/json" \
-X POST http://localhost:8080/v1/library-event
```



KafkaTemplate

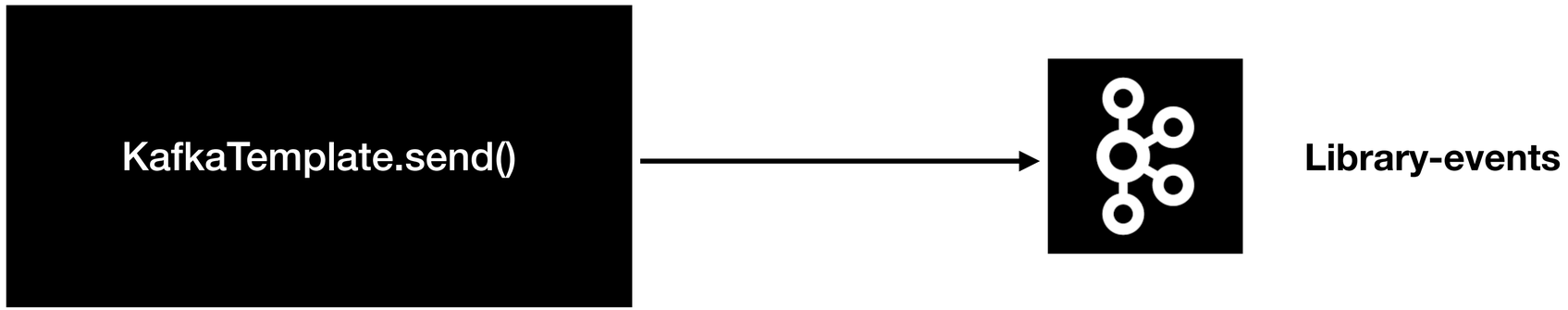
Kafka Producer in Spring

KafkaTemplate

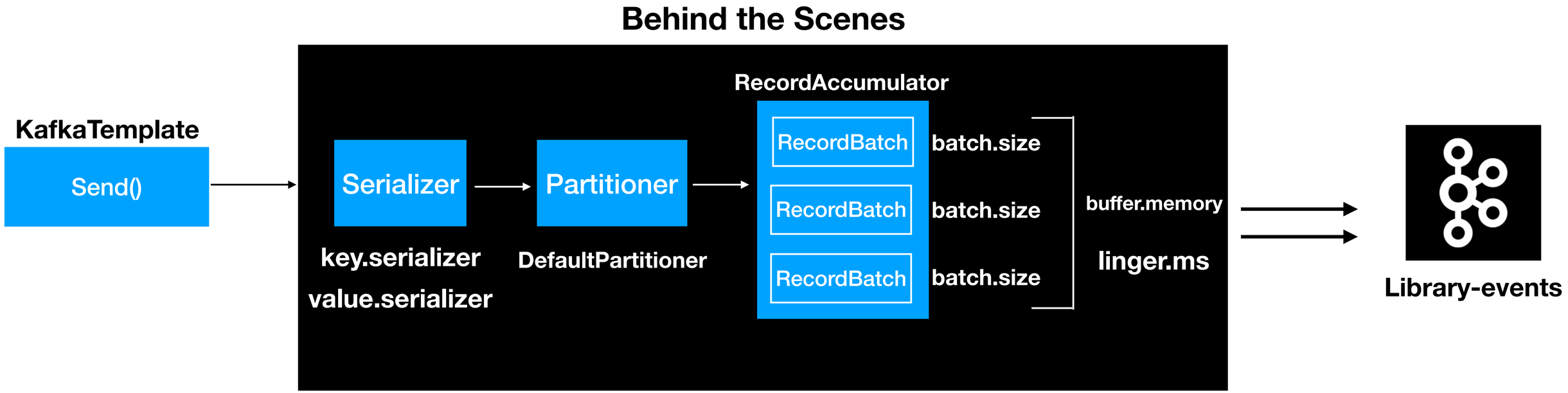
- Produce records in to Kafka Topic
  - Similar to JDBCTemplate for DB

Reference: https://docs.spring.io/spring-kafka/reference/html/#kafka-template

How KafkaTemplate Works ?



KafkaTemplate.send()



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
```

Why IntegerSerializer for key is because we are trying to send message as LibraryEvent which contains “libraryEventId” as Integer type.

The actual payload value type is String type. Hence we used StringSerializer

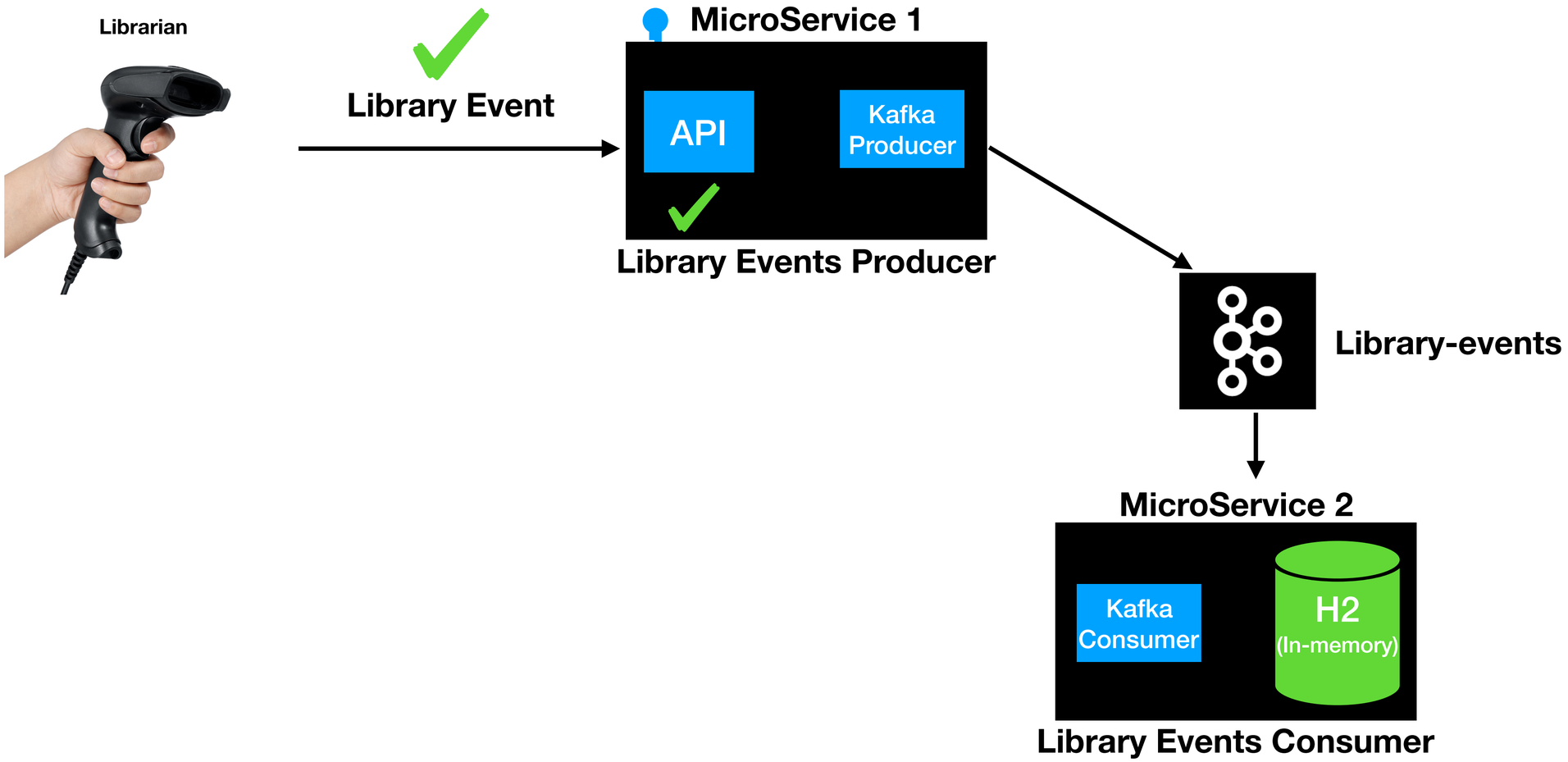
How SpringBoot AutoConfigure the Kafka producer? Refer: spring.factories, KafkaAutoConfiguration.java, KafkaProperties.java

KafkaTemplate AutoConfiguration

application.yml

```yaml
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

Not recommended

- 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

Refer: “AutoCreateConfig.java” to create a topic programmatically. After this step, run your Spring Boot Application.

Note: Before start your application, start your ZooKeeper, Broker0,1,2 (server, server-1, server-2)

Once you run, Kafka ZooKeeper would have created a topic “library-events”. To verify this, execute this command

```
$bin>./kafka-topics.sh --zookeeper localhost:2181 --list
```

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 todays software development
- Easy to capture bugs
- Types of Automated Tests:
    - UnitTest
    - Integration Tests
    - End to End Tests

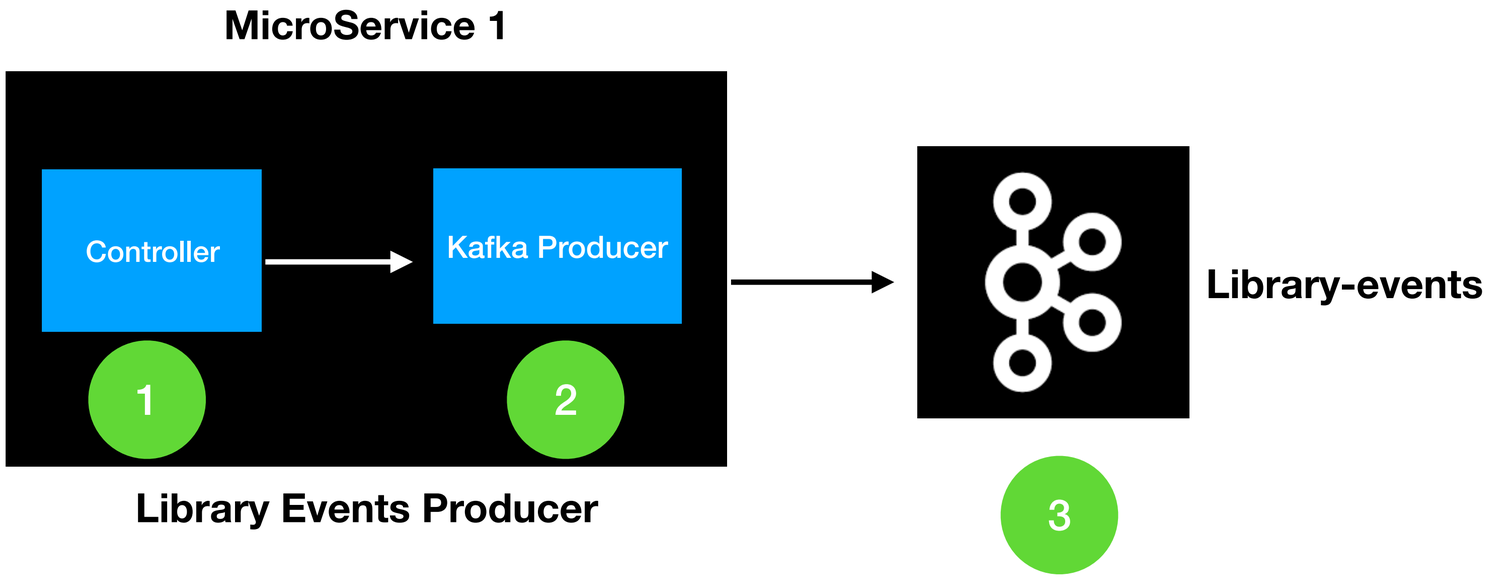
Tools for Automated

- JUnit
- Spock

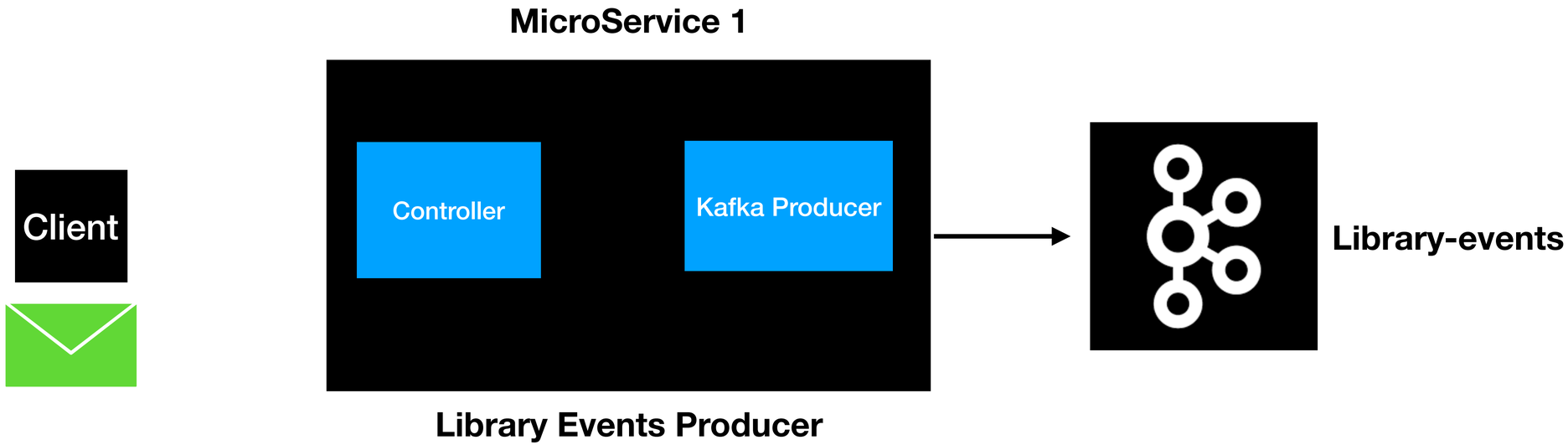
Integration Tests Using JUnit5

What is Integration Test?

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



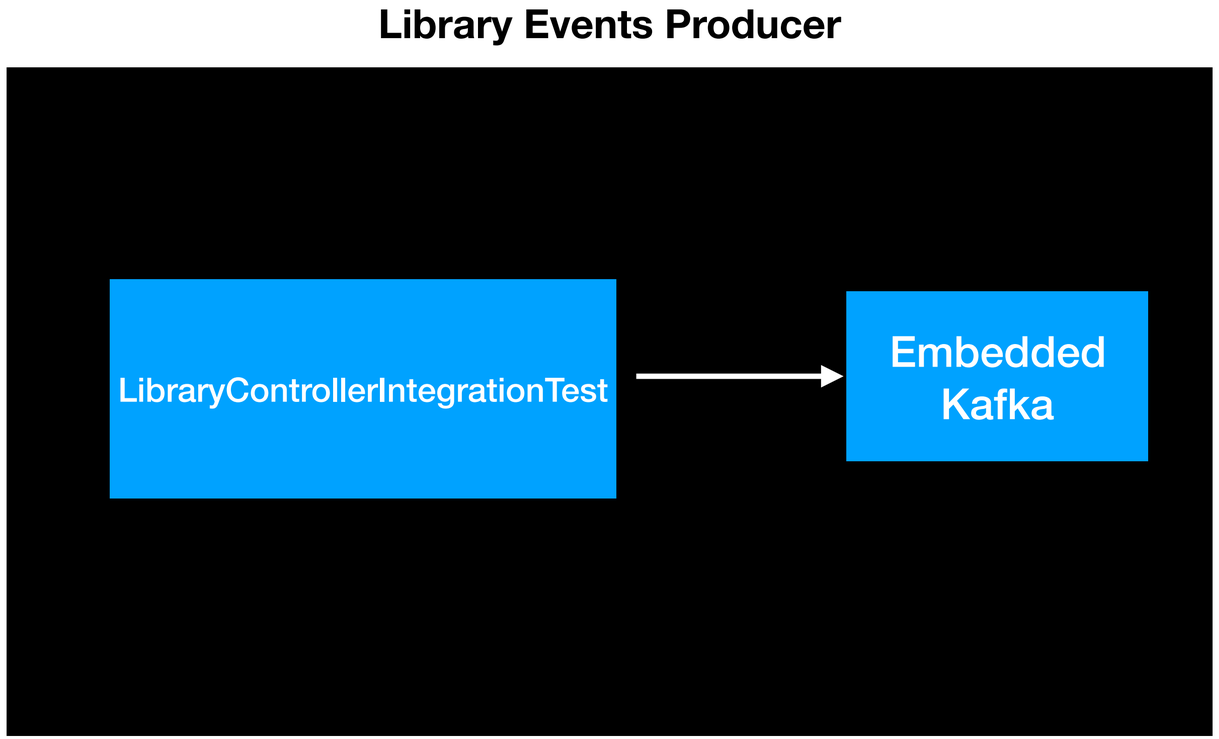
Integration Test



Embedded Kafka

What is EmbeddedKafka?

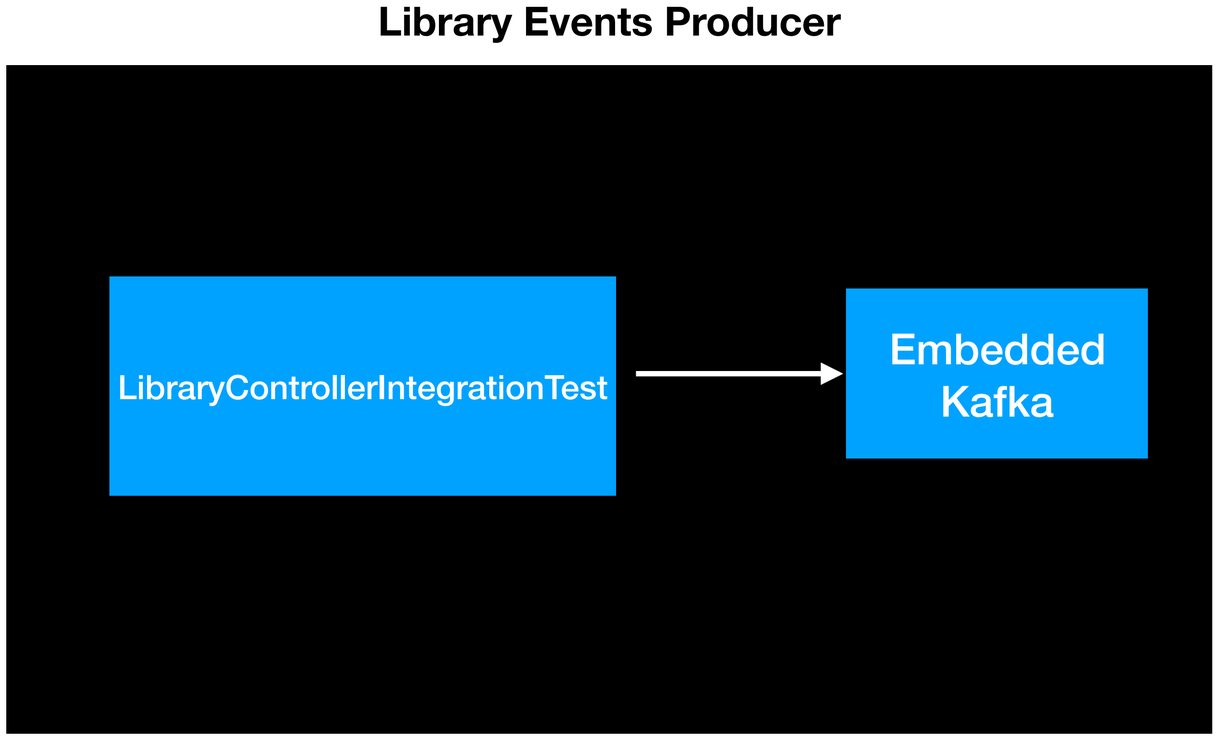
- In-Memory Kafka
- Integration Tests can interact with EmbeddedKafka



Why Embedded Kafka ?

- Easy to write Integration Tests

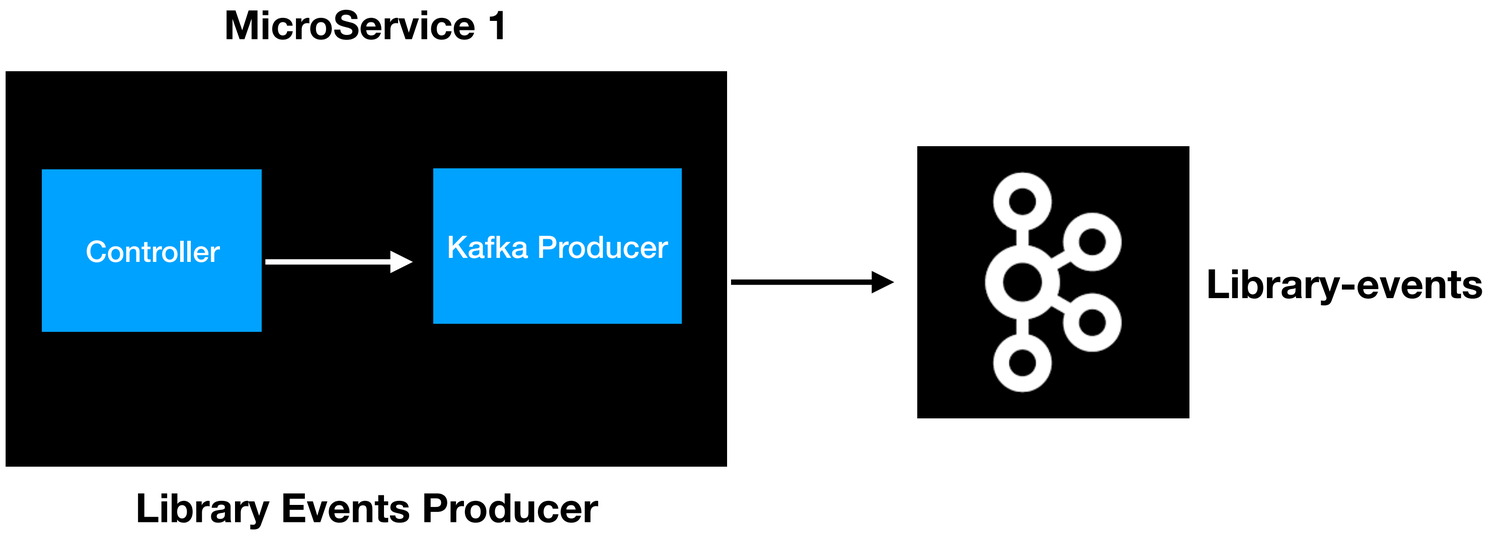
- Test all the code as like you interact with Kafka



Unit Tests
Using
JUnit5

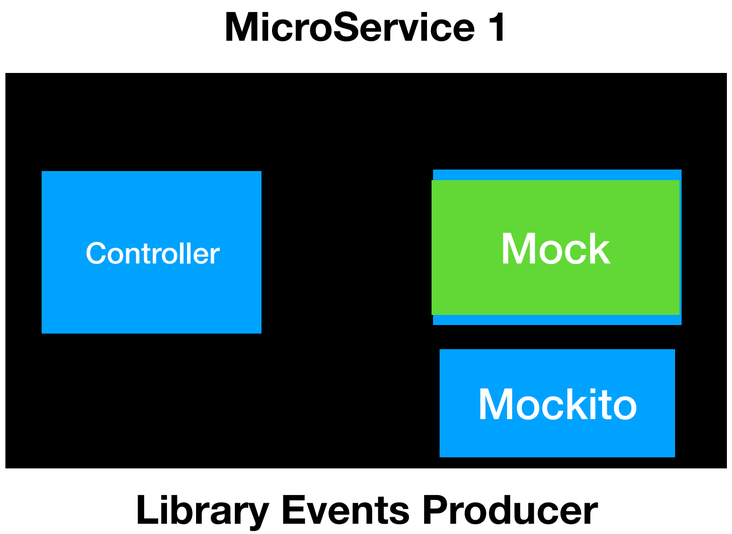
What is Unit Test?

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



What is Unit Test?

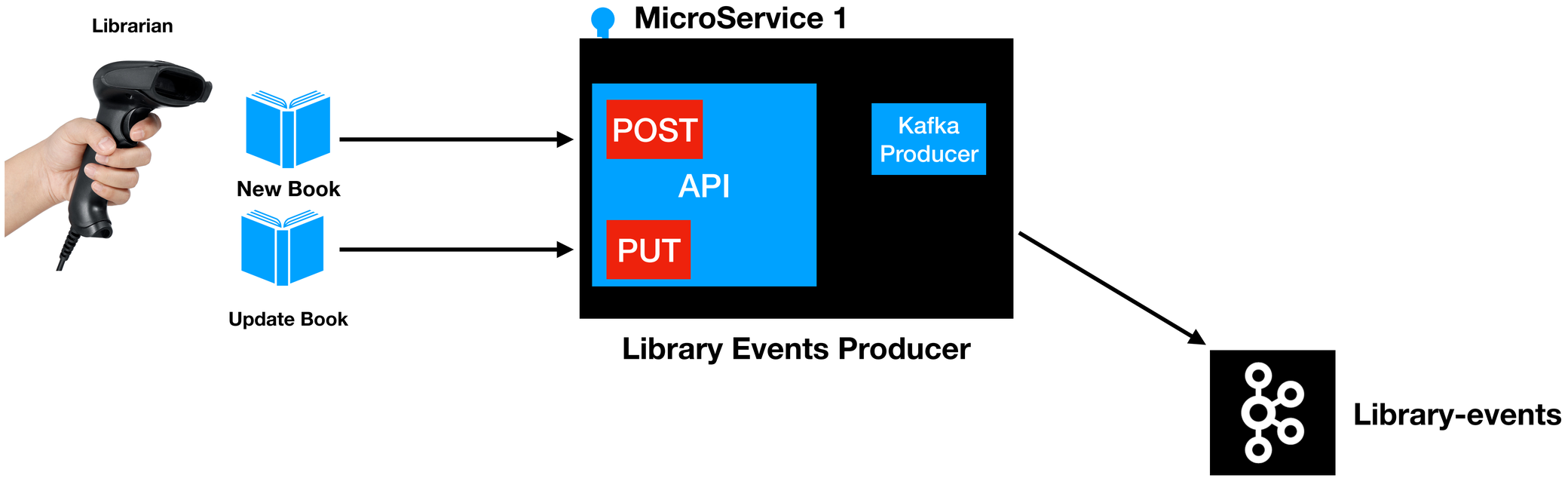
- Test the just focuses on a single unit (method)
- Mocks the external dependencies



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 which is the key configuration for reliable data delivery.
  - acks = 0, 1 and all
    - acks = 1 -> guarantees message is written to a leader (Default) 
    - acks = all -> guarantees message is written to a leader and to all the replicas 
    - acks=0 -> no guarantee (Not Recommended) 

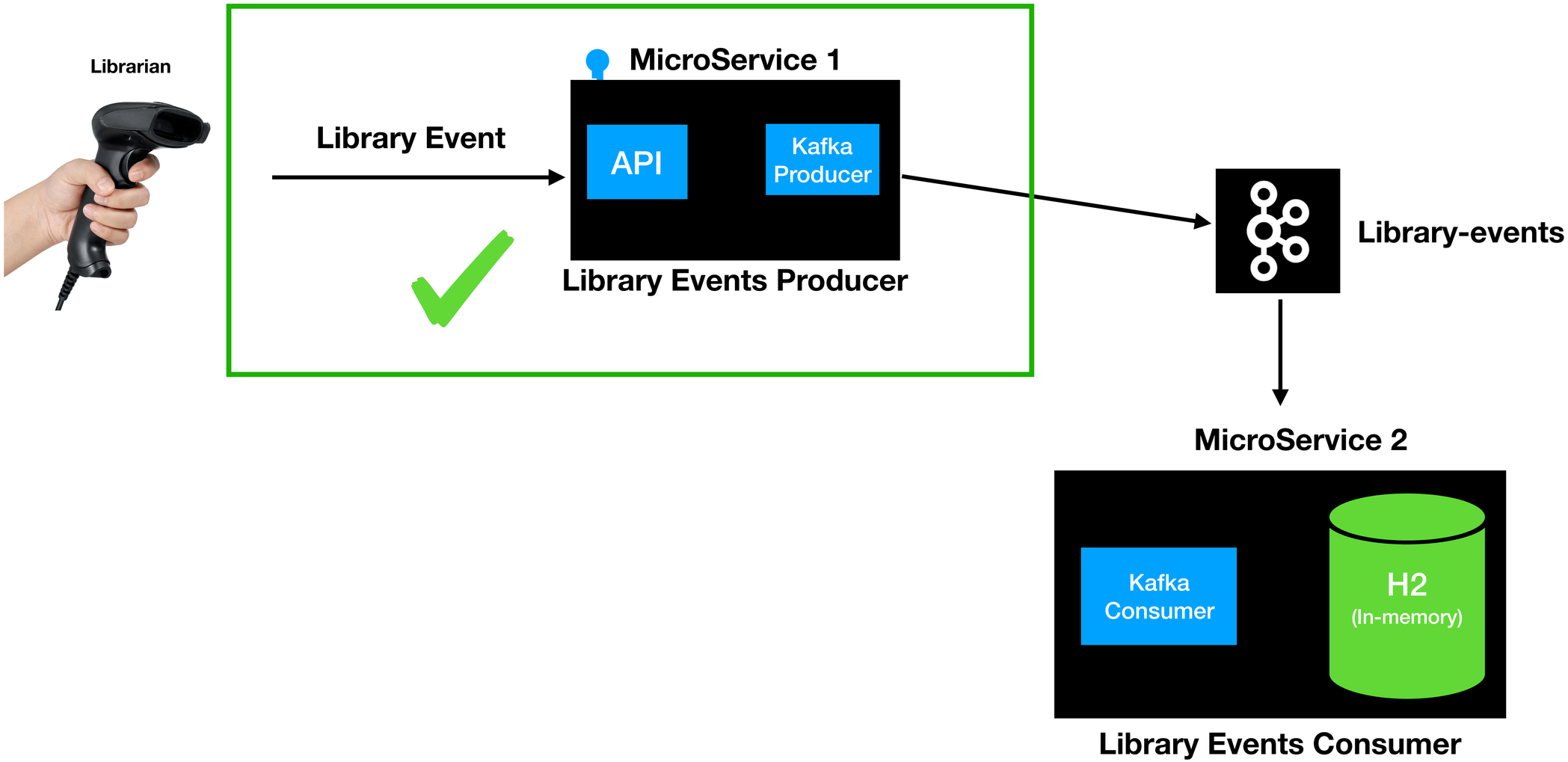
To see these values, execute the integration test and it will display all the producer config related values.

Kafka Producer Configurations

https://kafka.apache.org/documentation/#producerconfigs

- 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

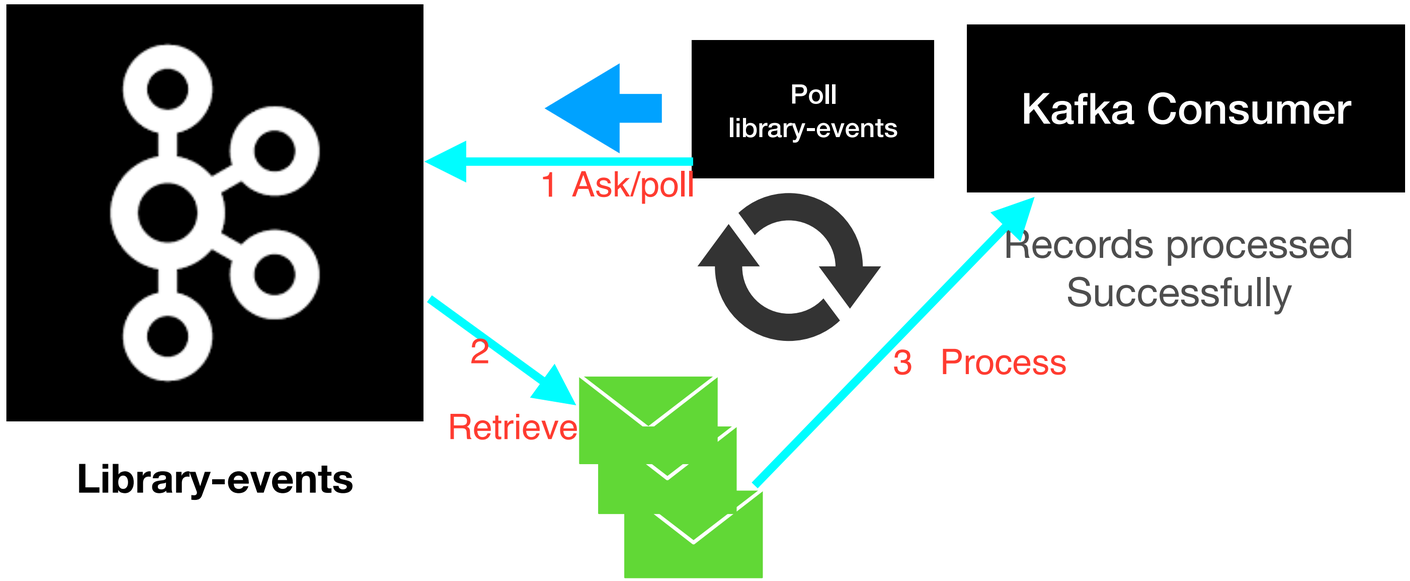
Library Events Consumer



Spring Kafka Consumer

Kafka Consumer

TIP: Kafka Consumer, can poll records from multiple topics



Spring Kafka Consumer

- MessageListenerContainer Interface

  Implementation

  - 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

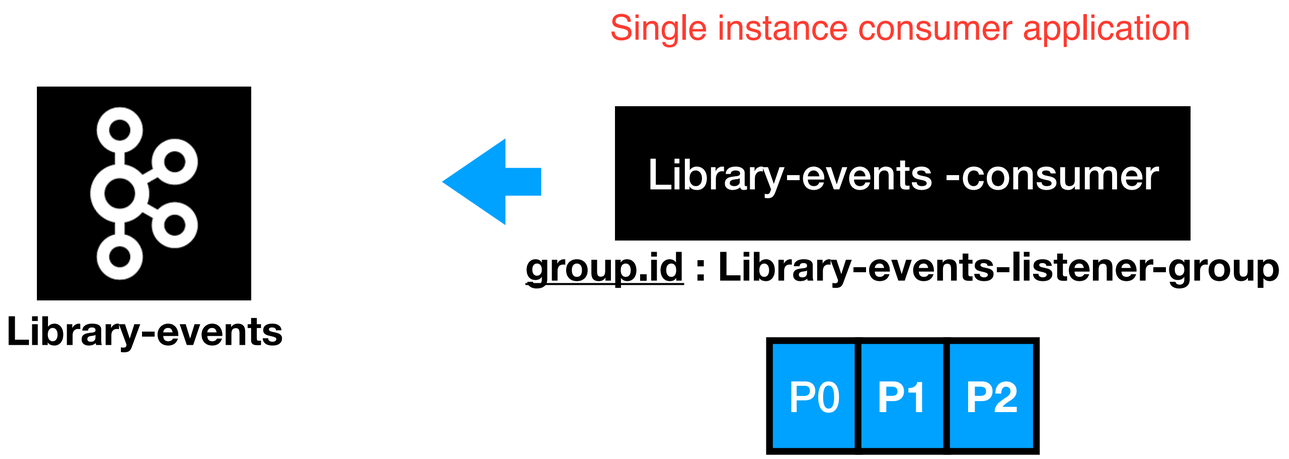
Consumer Groups

Consumer Groups are the foundation for a scalable message consumption

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

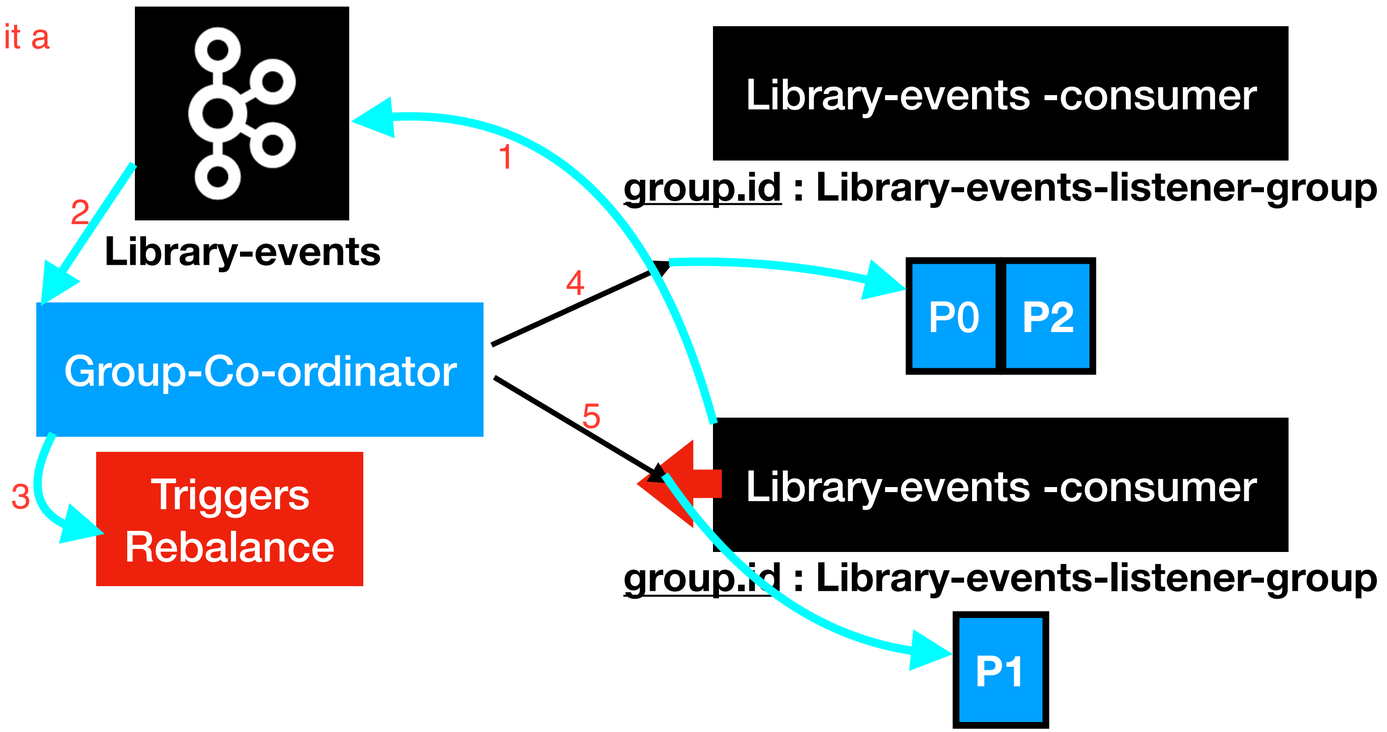
Here we have two instances of the same application running on 8081 and 8082

1. Call the Kafka Broker which intern connect to “Group-Co-Ordinator”

2. Group Coordinator checks is it a same group id? if yes

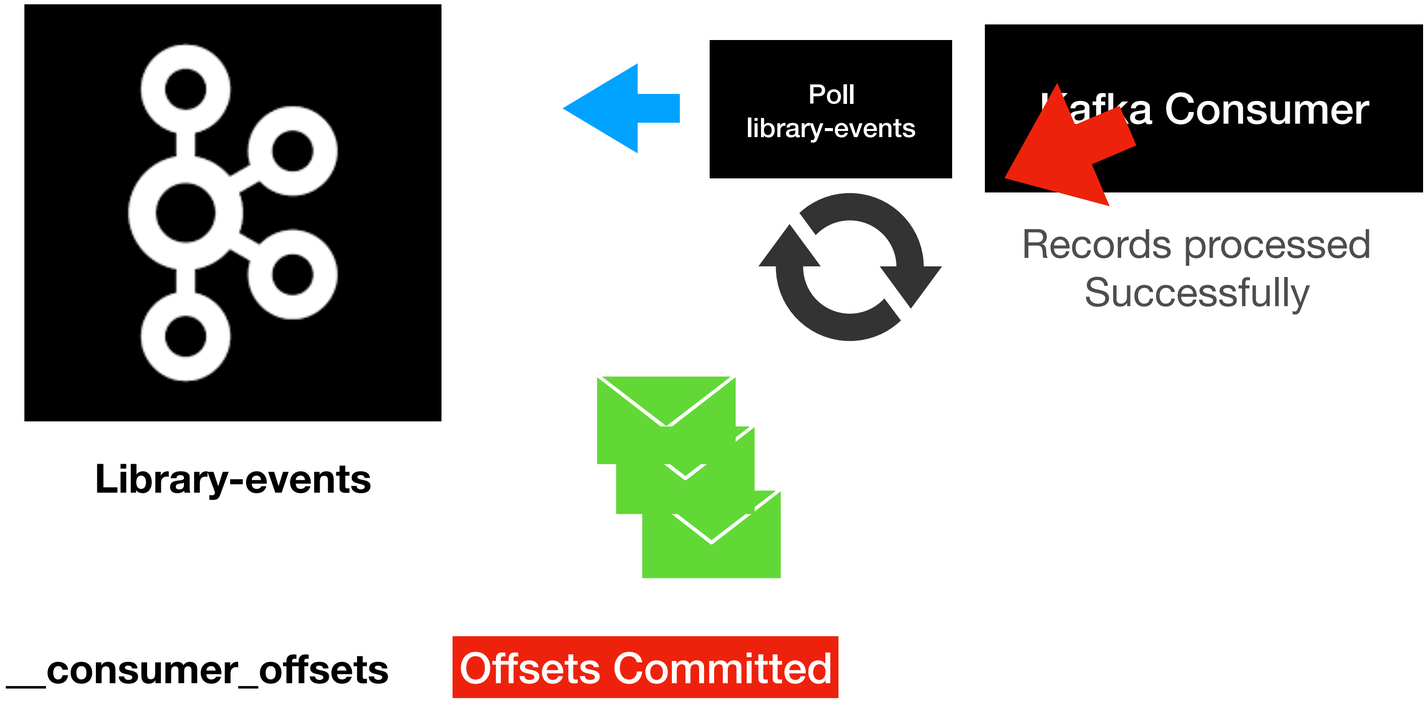
3. It will trigger a “Rebalance”. What it does?
Oh! I have one more instance of the same application in that case, i have to redistribute the partitions

4. Using “Rebalance”, Group-Coordinator will assign the odd partitions to 1st consumer (refer 4)
and event partitions to 2nd consumer (refer 5)

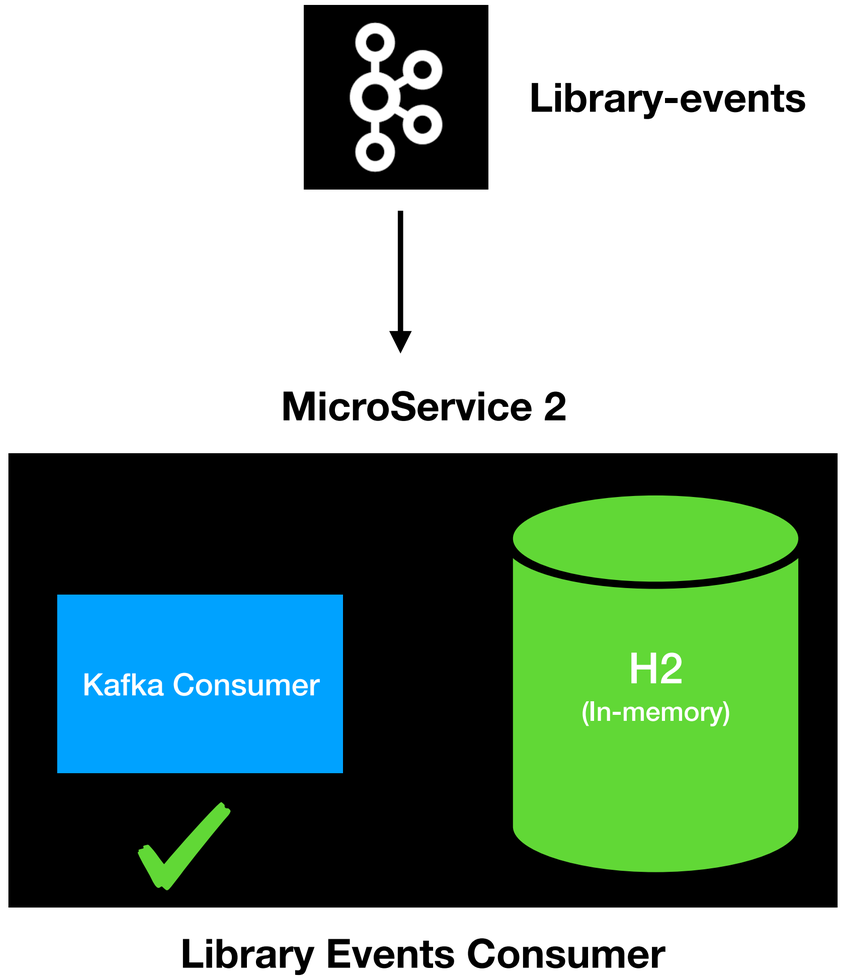


Before look at this slide,
1.run your producer application
2.run your consumer application
3.send messages through REST
4.Message reached consumer app
5.down your consumer app
6.Again start consumer app. Now it won't display the previously sent messages. why?
7.Because, once record processed successfully, behind the scene consumer commits the offsets in broker in "__consumer_offsets"

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**

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/**

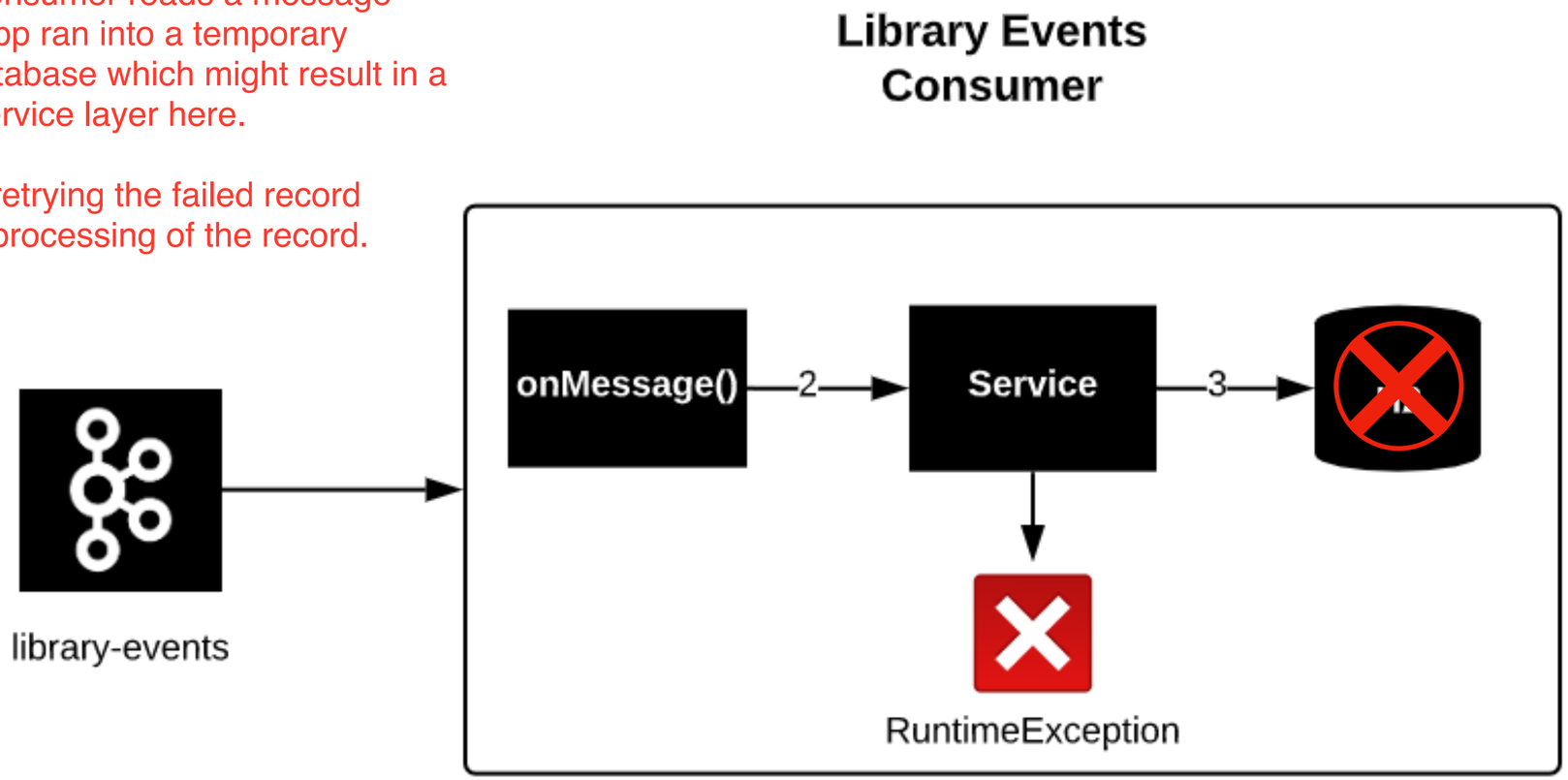
Retry in Kafka Consumer

Error in Kafka Consumer

In this lecture we will learn about the techniques to enable retry for a failed record.

In this example here the consumer reads a message and for some reason the app ran into a temporary network issues with the database which might result in a runtime exception in the service layer here.

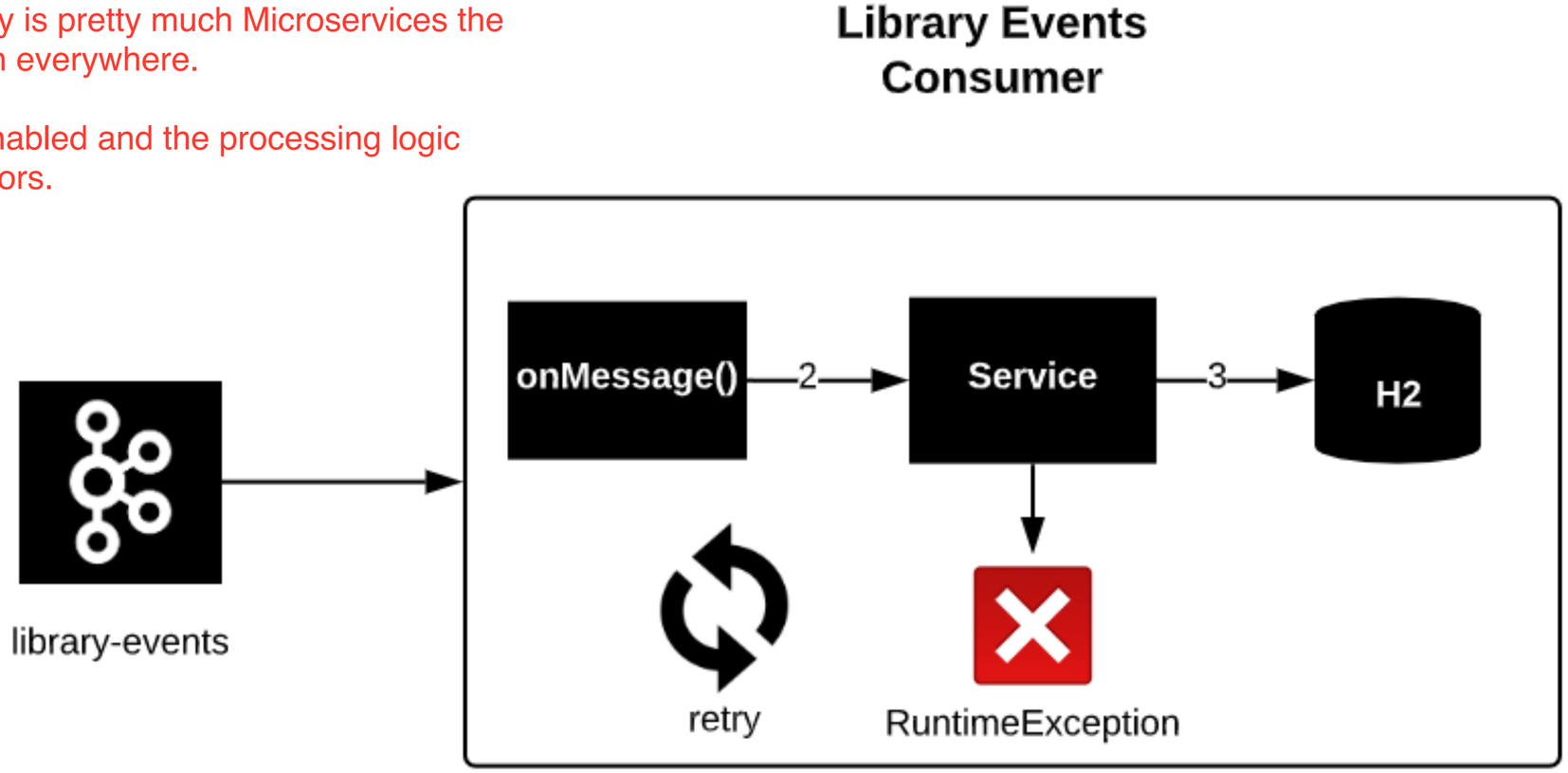
In these kind of scenarios retrying the failed record would result in successful processing of the record.



Retry in Kafka Consumer

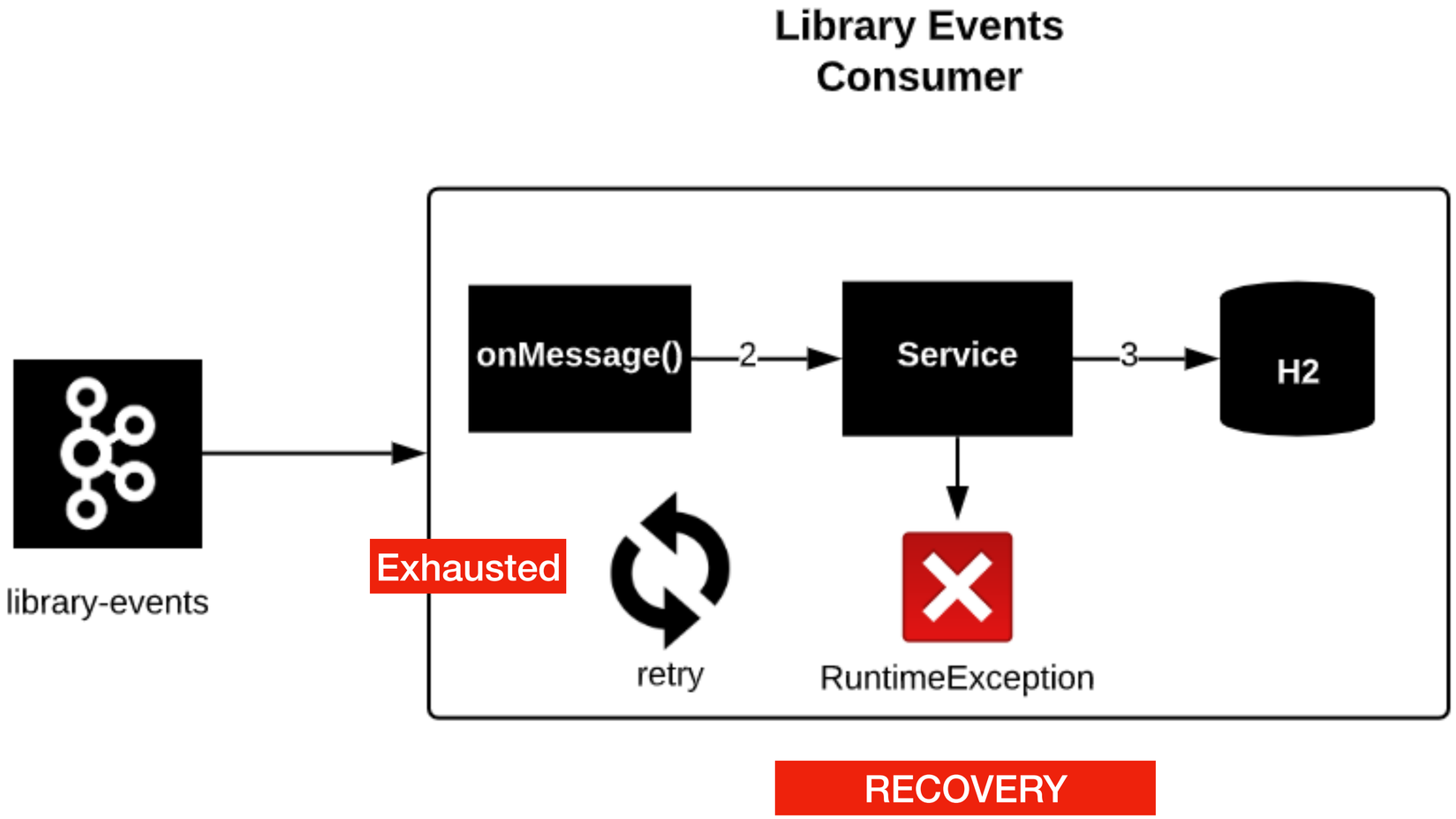
The software that we build today is pretty much Microservices the network causes are pretty much everywhere.

It's really handy to have retry enabled and the processing logic so that we can avoid a lot of errors.

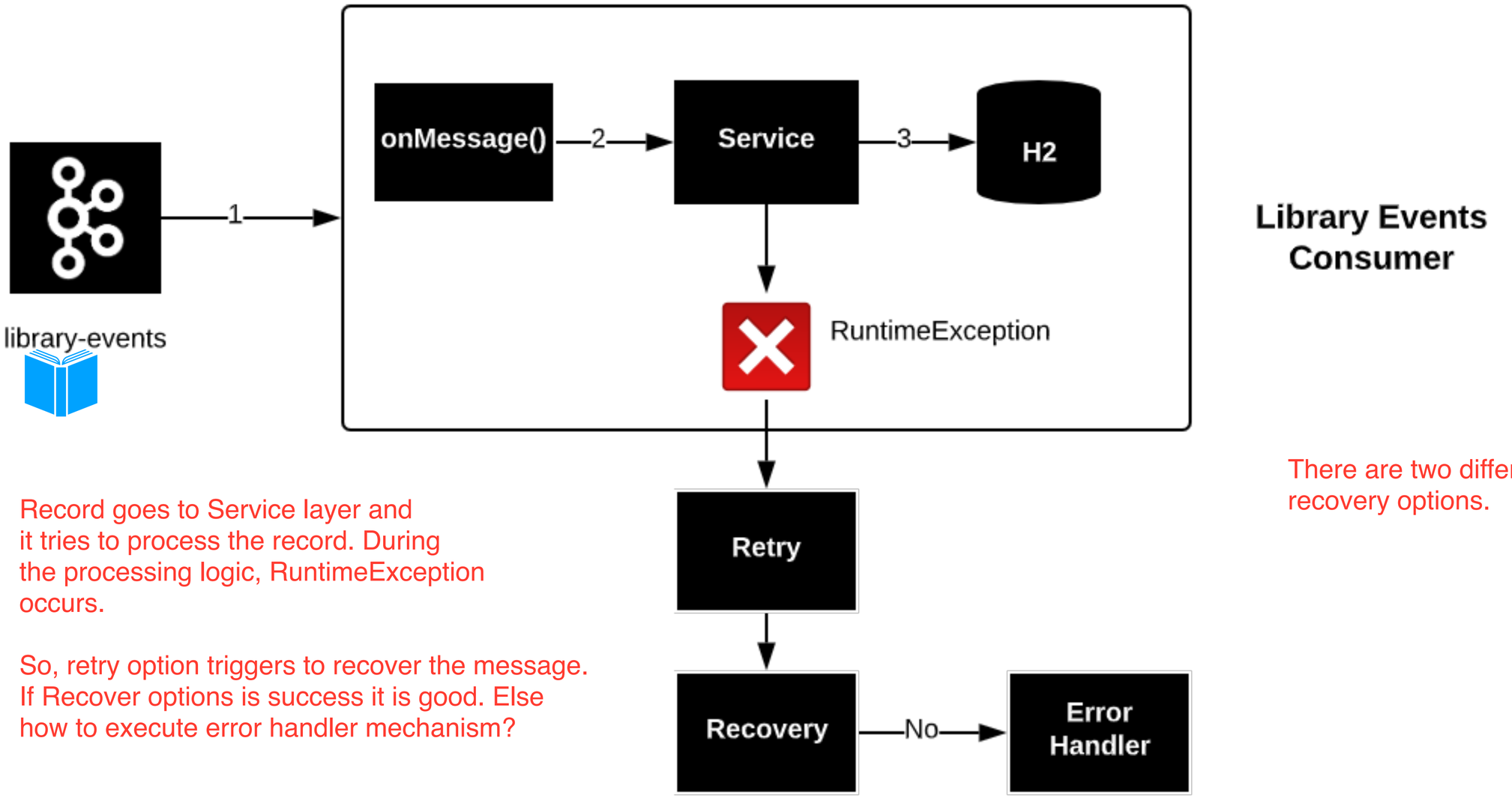


Recovery in Kafka Consumer

Recovery in Kafka Consumer



Retry and Recovery

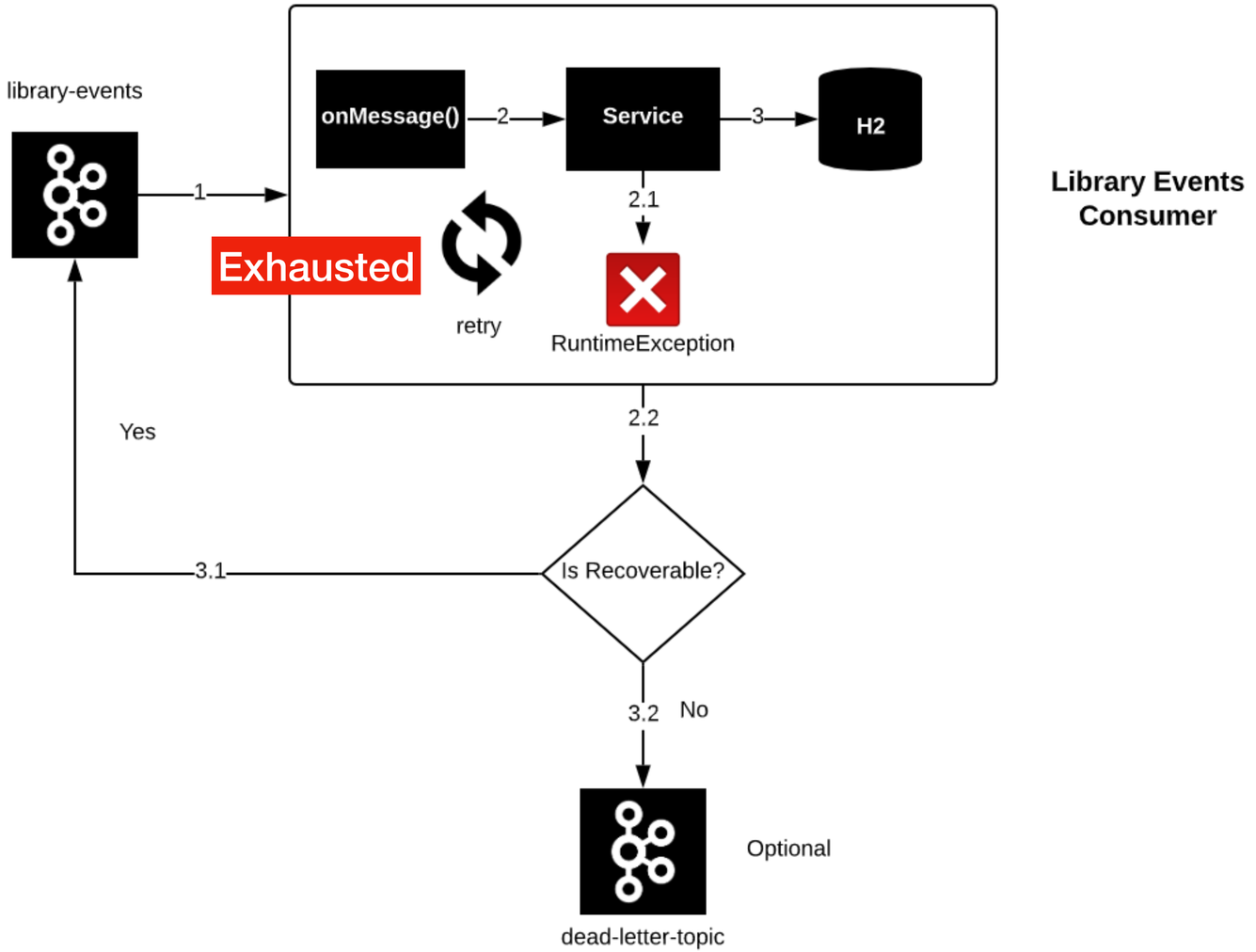


Record goes to Service layer and it tries to process the record. During the processing logic, RuntimeException occurs.

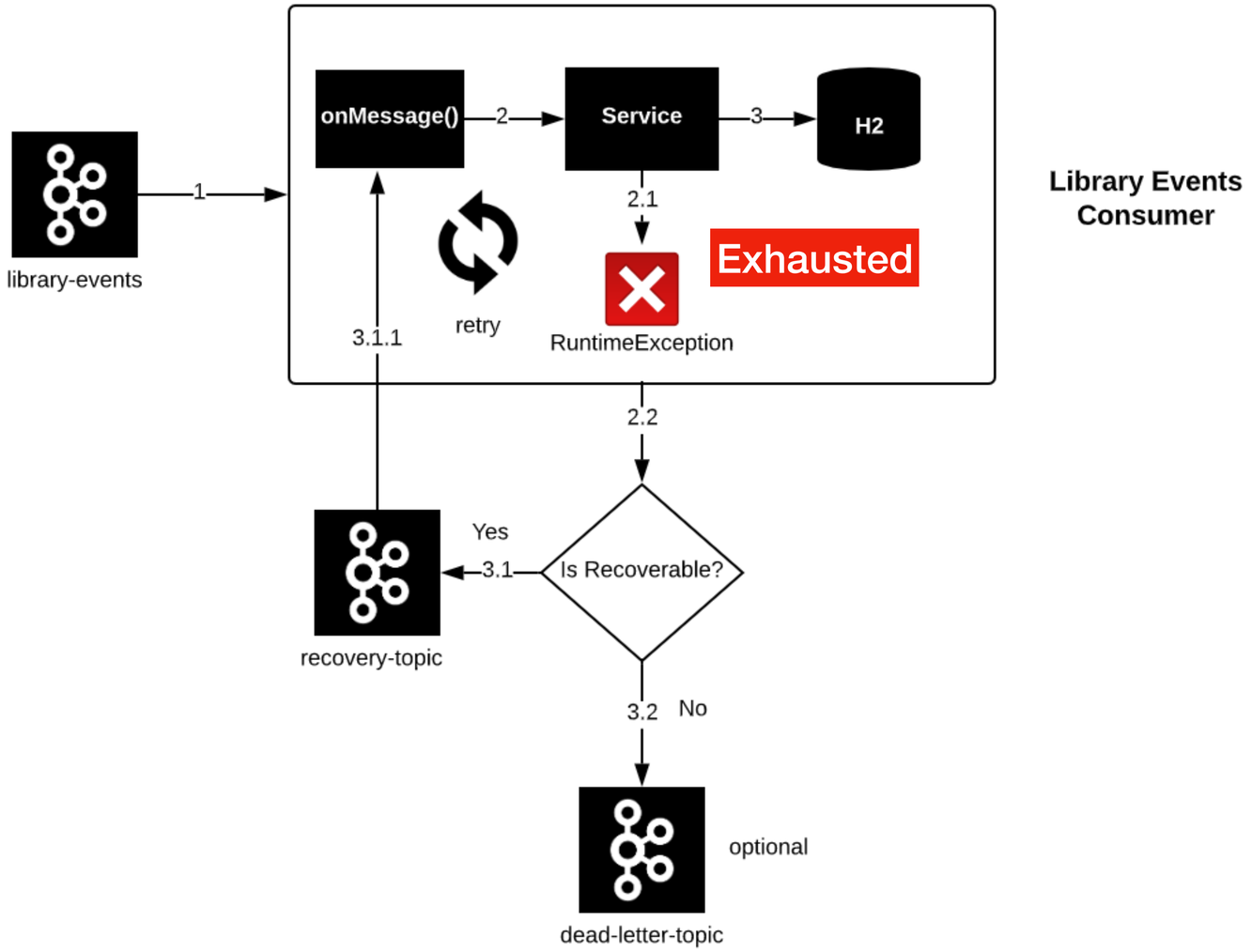
So, retry option triggers to recover the message. If Recover options is success it is good. Else how to execute error handler mechanism?

There are two different types of recovery options.

Recovery - Type 1



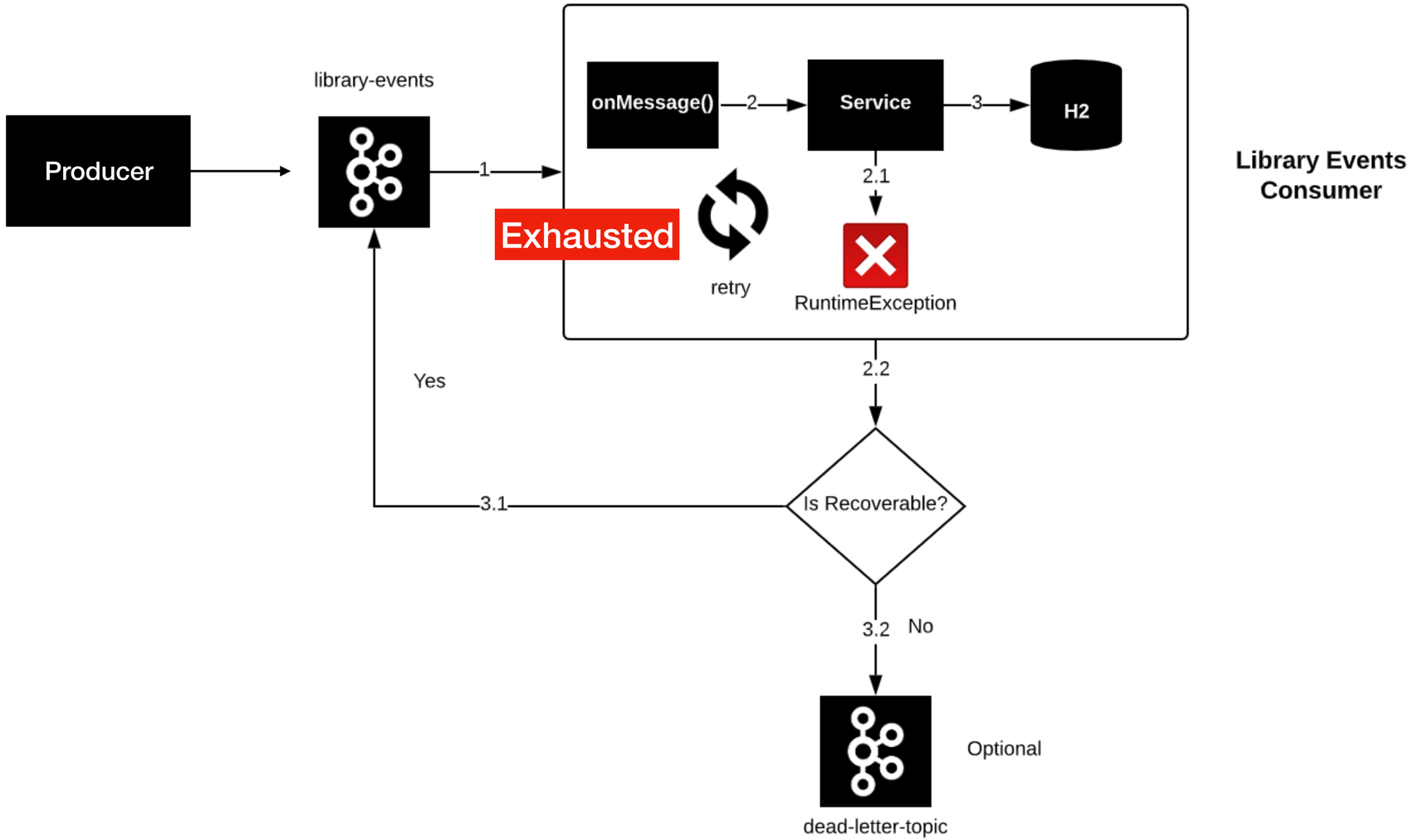
Recovery - Type 2



Issues with Recovery ?

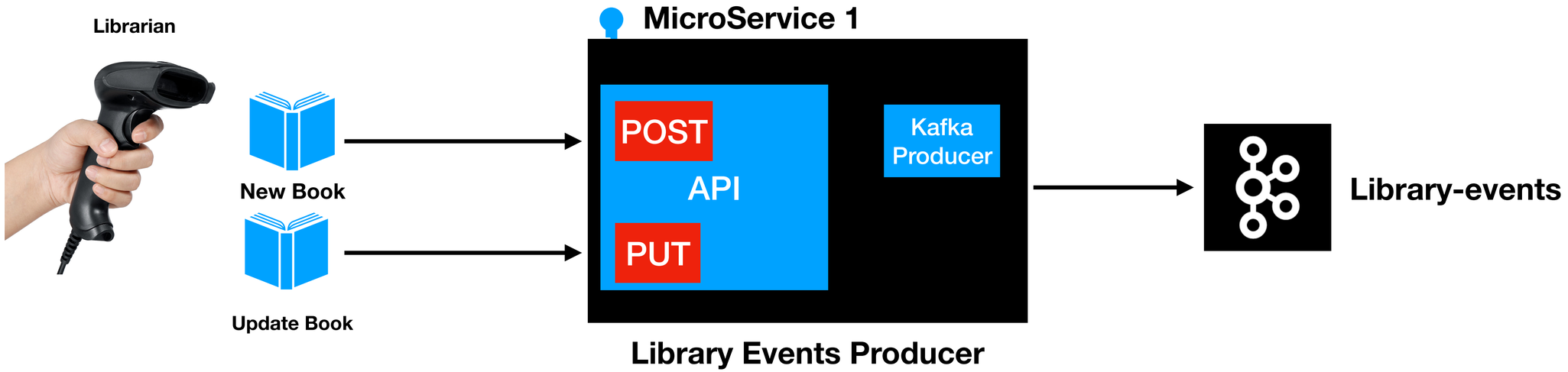
- Recovery can alter the order of events

Recovery - Type 1



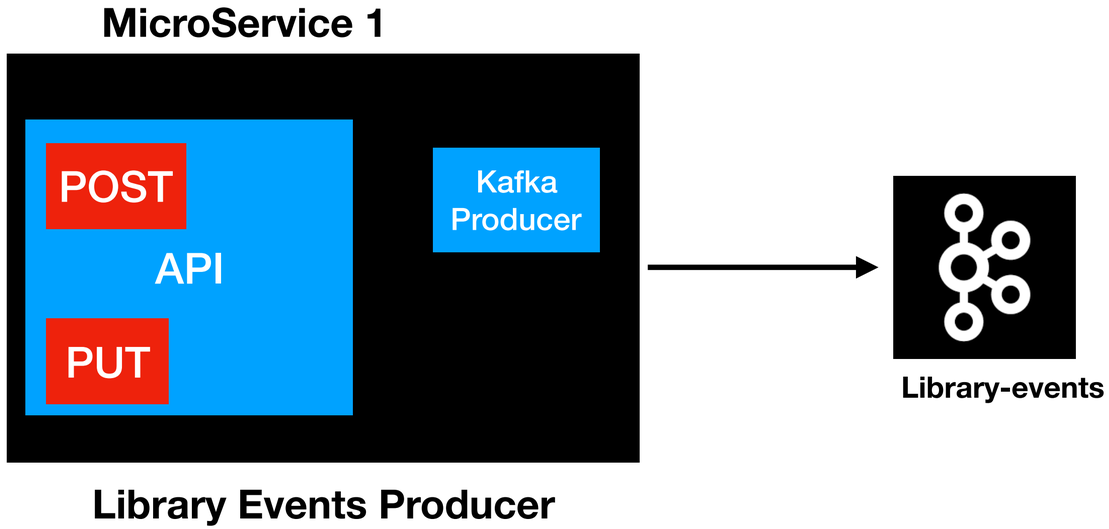
Error Handling
in
Kafka Producer

Library Events Producer API



Kafka Producer Errors

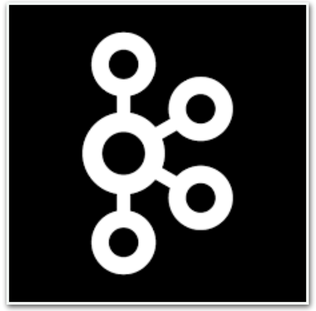
- Kafka Cluster is not available
- If **acks= all ,** some brokers are not available
- **min.insync.replicas** config
  - **Example : min.insync.replicas = 2**, But only one broker is available



min.insync.replicas

min.insync.replicas = 2

Kafka Cluster



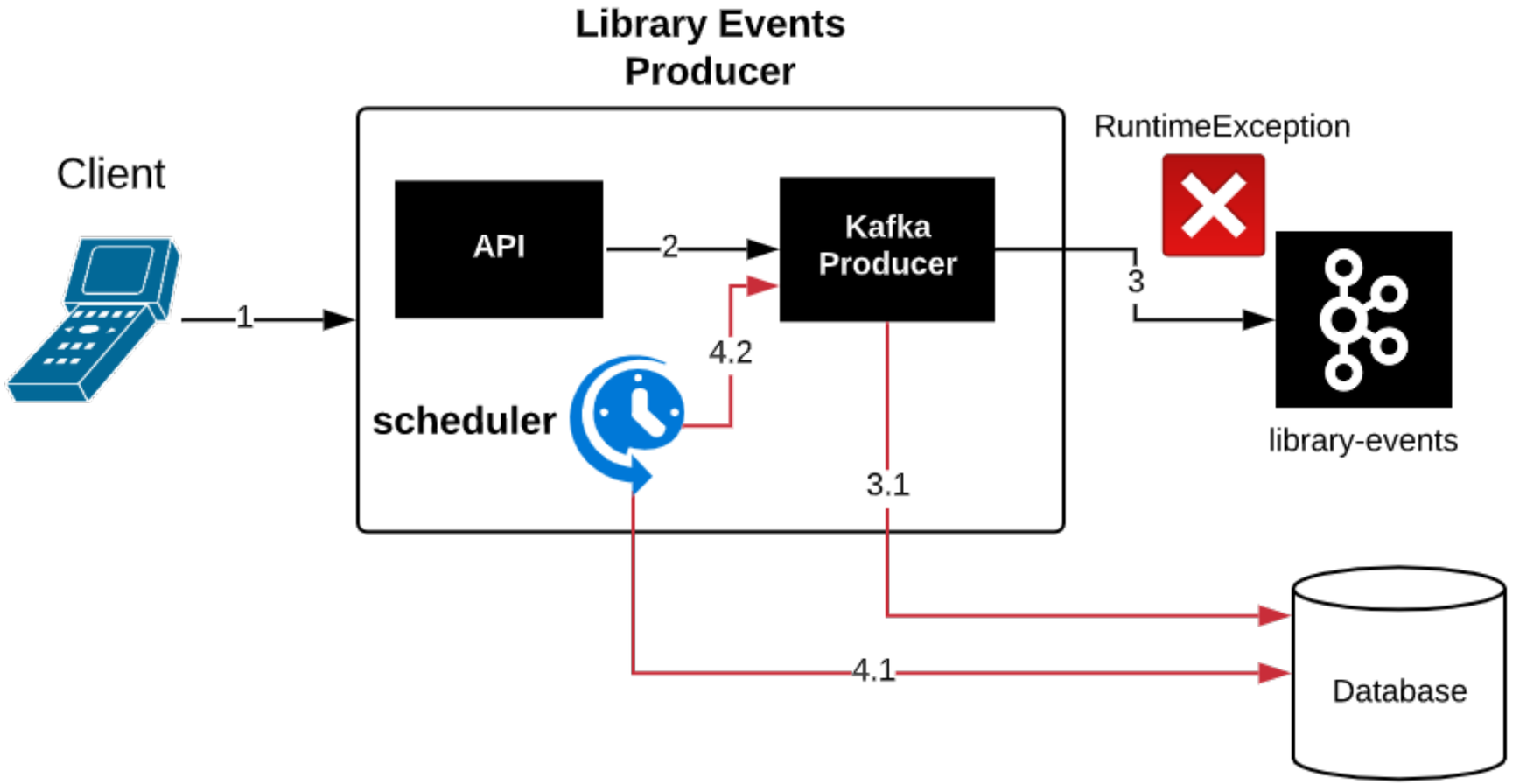
Broker 1

Broker 2

Broker 2

Retain/Recover Failed Records

Retain/Recover Failed Records



Retain/Recover Failed Records