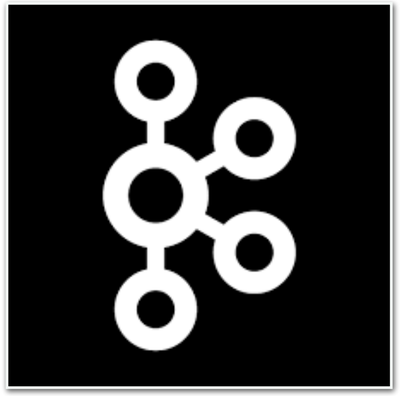
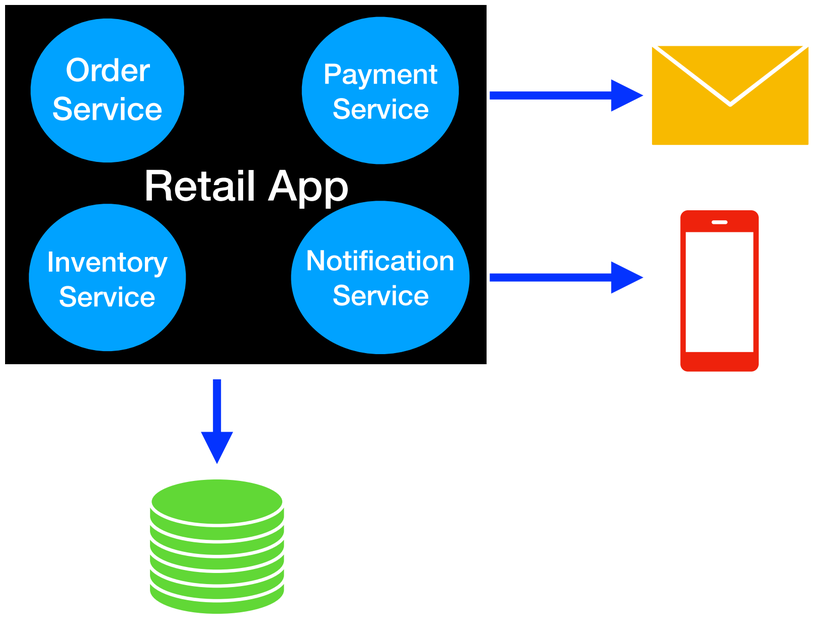


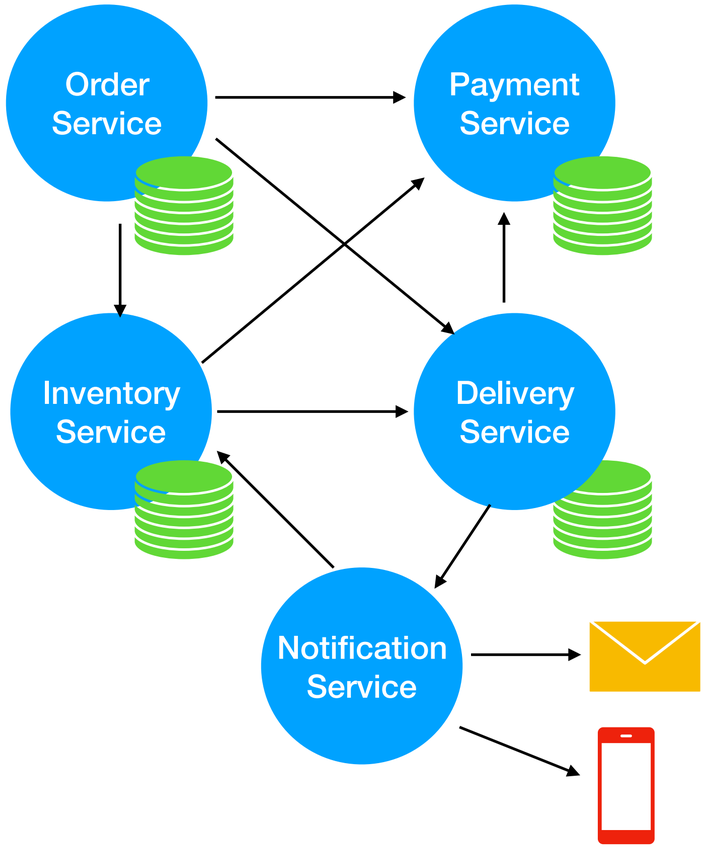
Kafka For Developers Using Spring Boot

Software Development

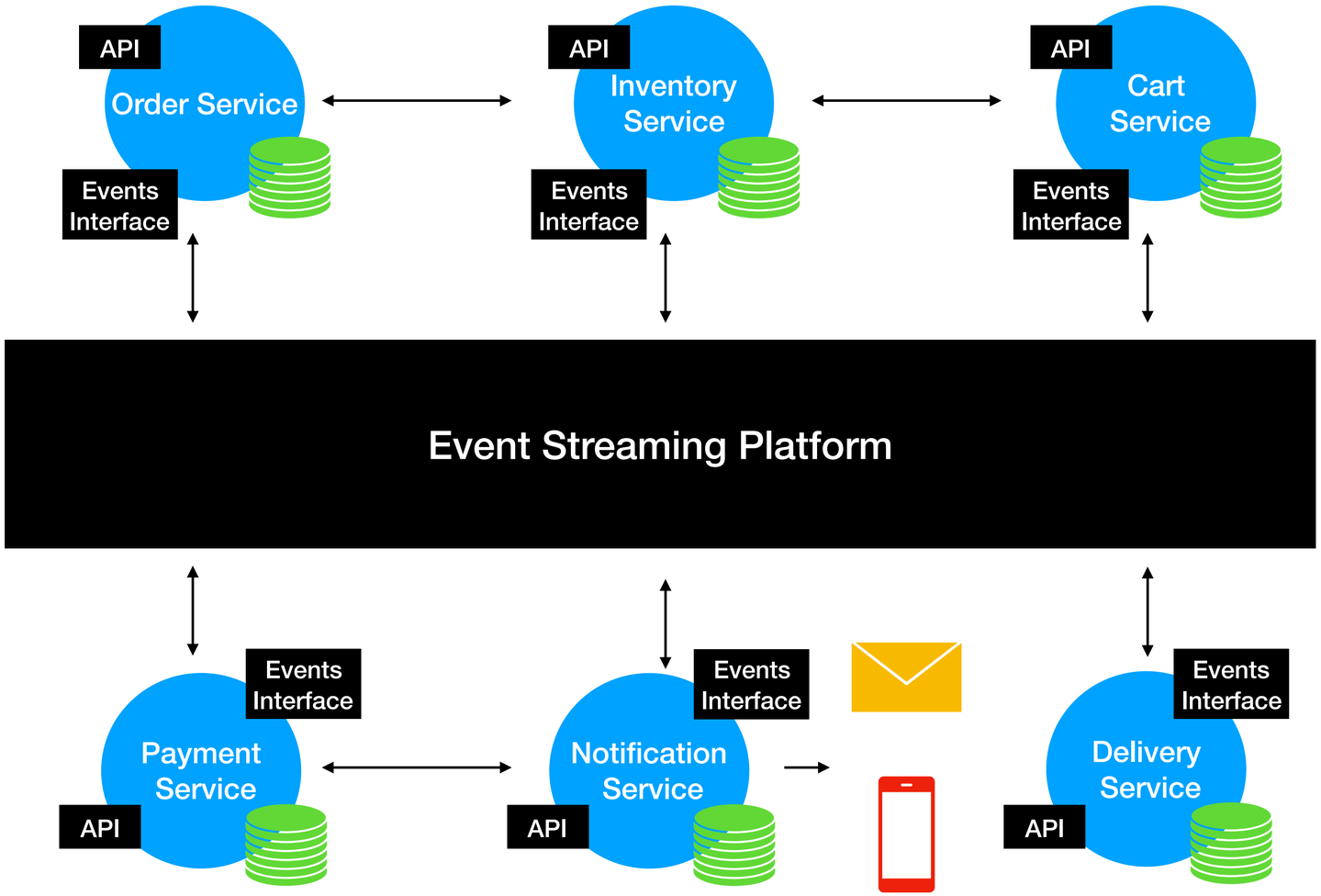
Past



Current



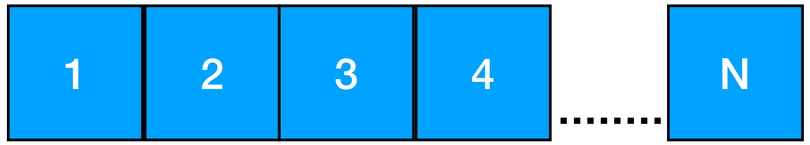
MicroServices Architecture



What is an Event Streaming Platform?

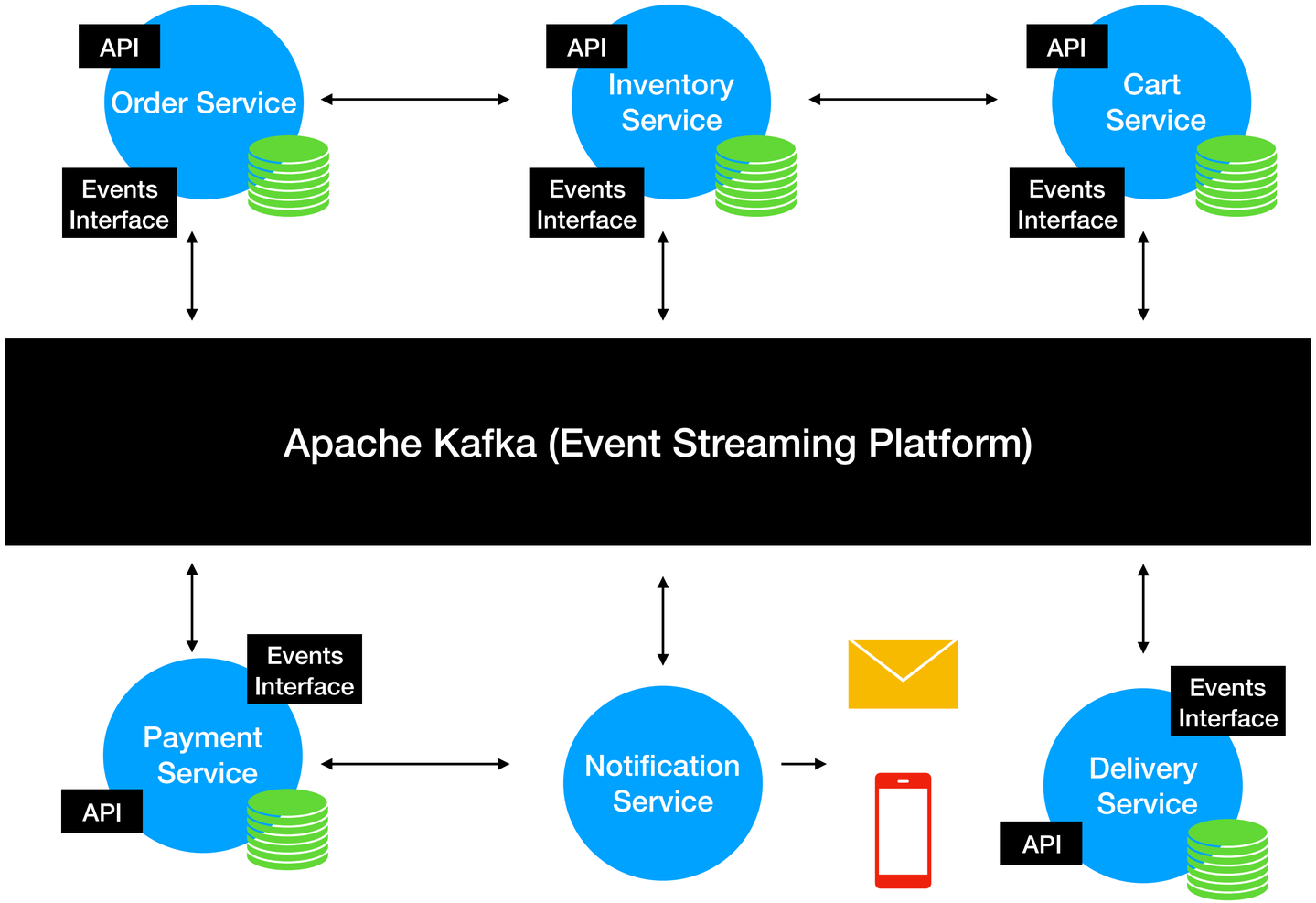
- Producers and Consumers subscribe to a stream of records

- 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

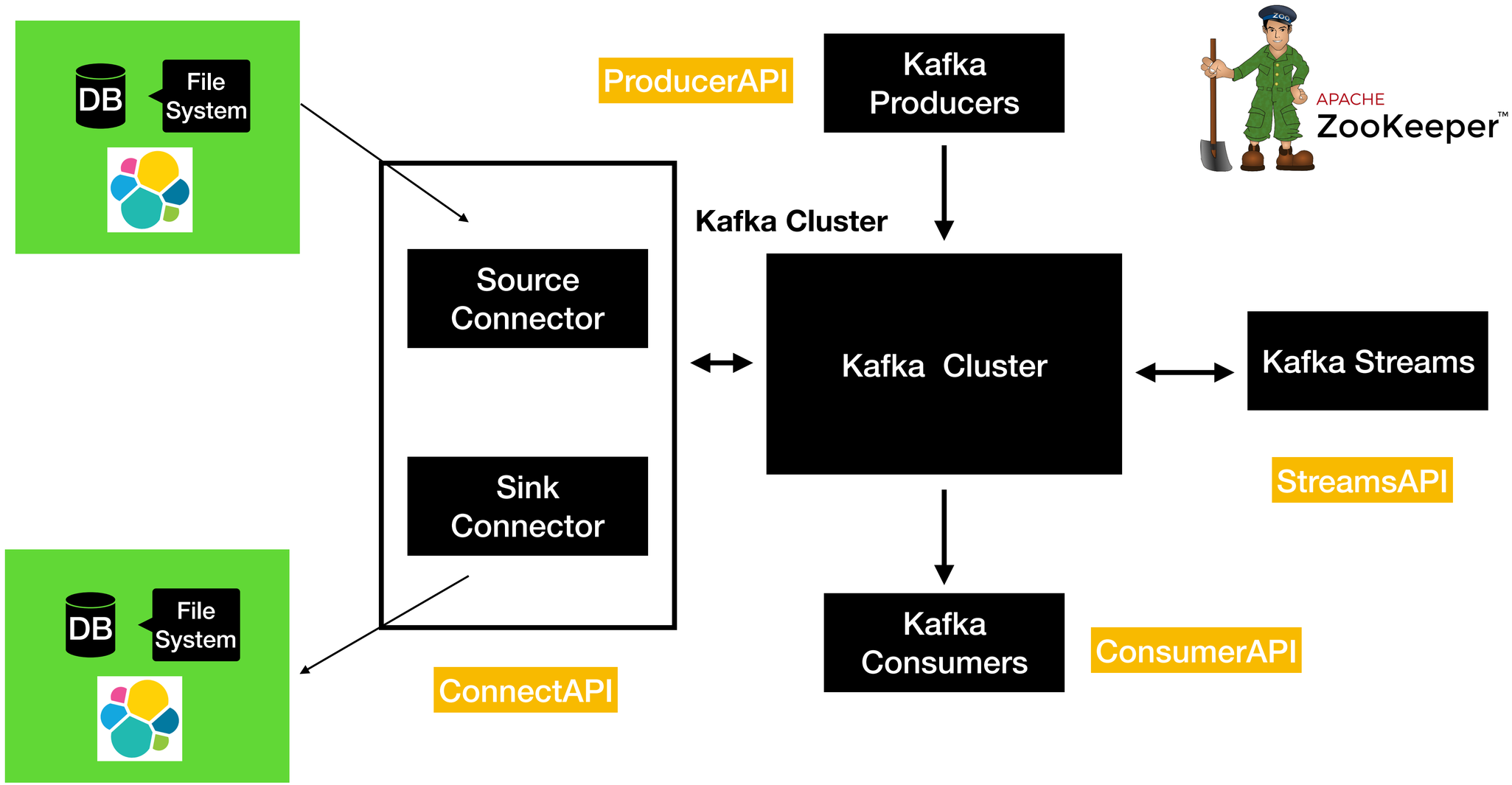
Tracking Online Order Deliveries

Banking

Fraud Transactions

New Feature/Product notifications

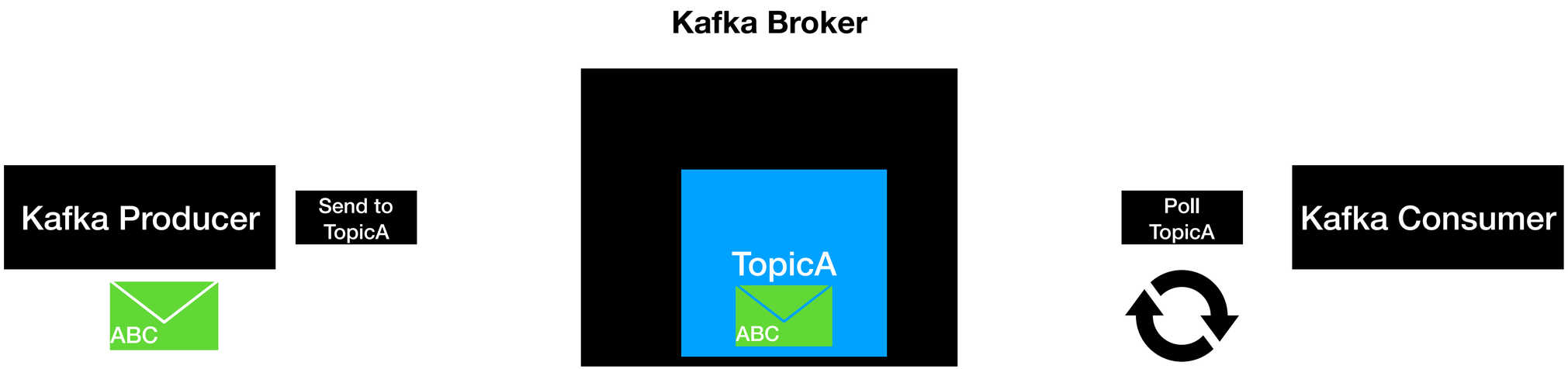
Kafka Terminology & Client APIs



Kafka Topics
&
Partitions

Kafka Topics

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

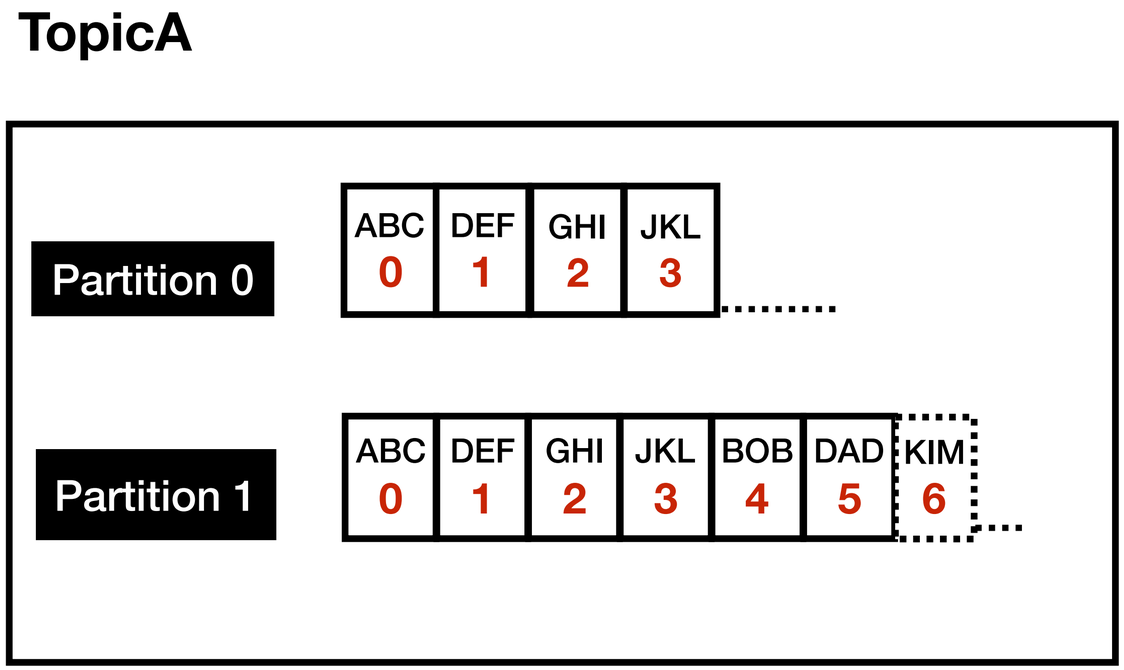


Topic and Partitions

- 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***

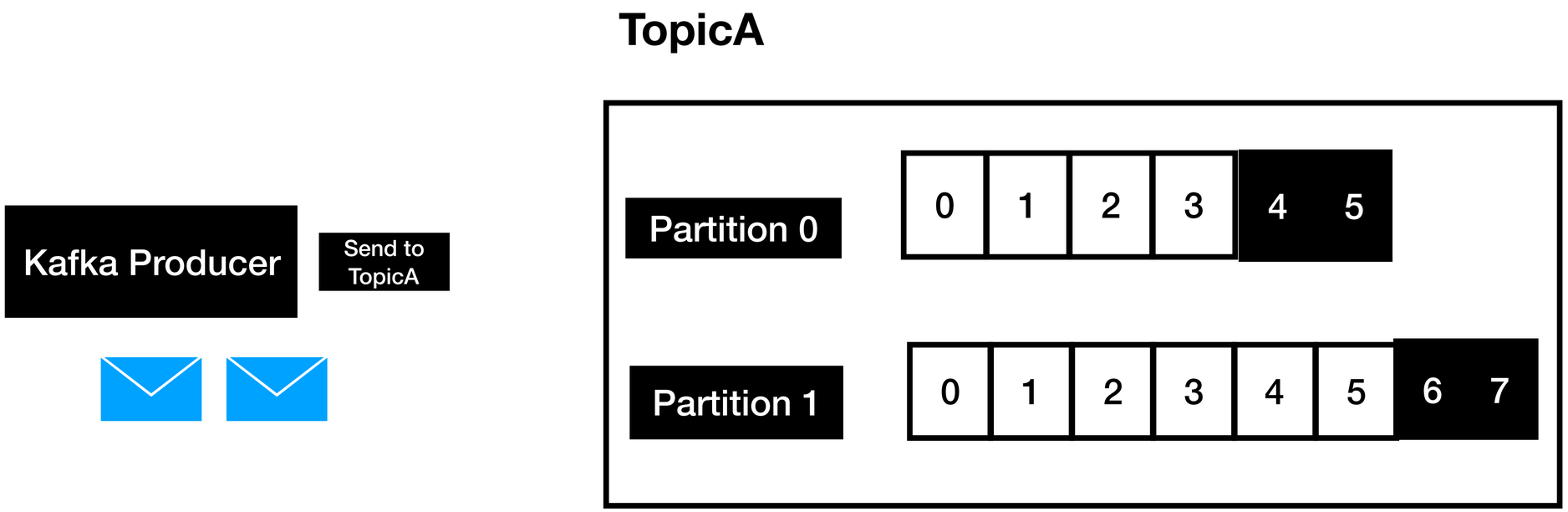
- Each partition is independent of each other

- Ordering is guaranteed only at the partition level

- Partition continuously grows as new records are produced

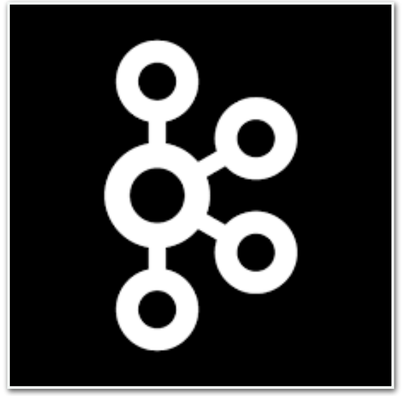
- All the records are persisted in a commit log in the file system where Kafka is installed

Topics and Partitions



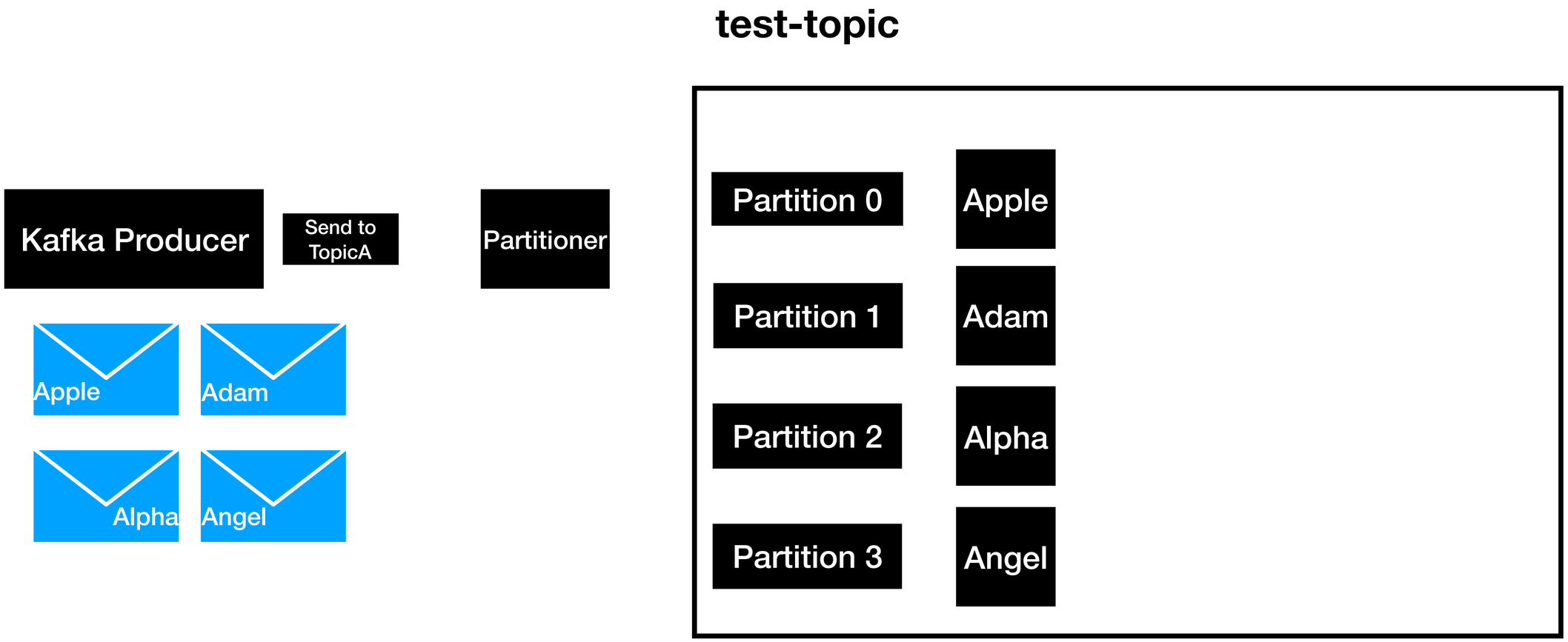
Setting up Kafka in Local

APACHE
ZooKeeper™

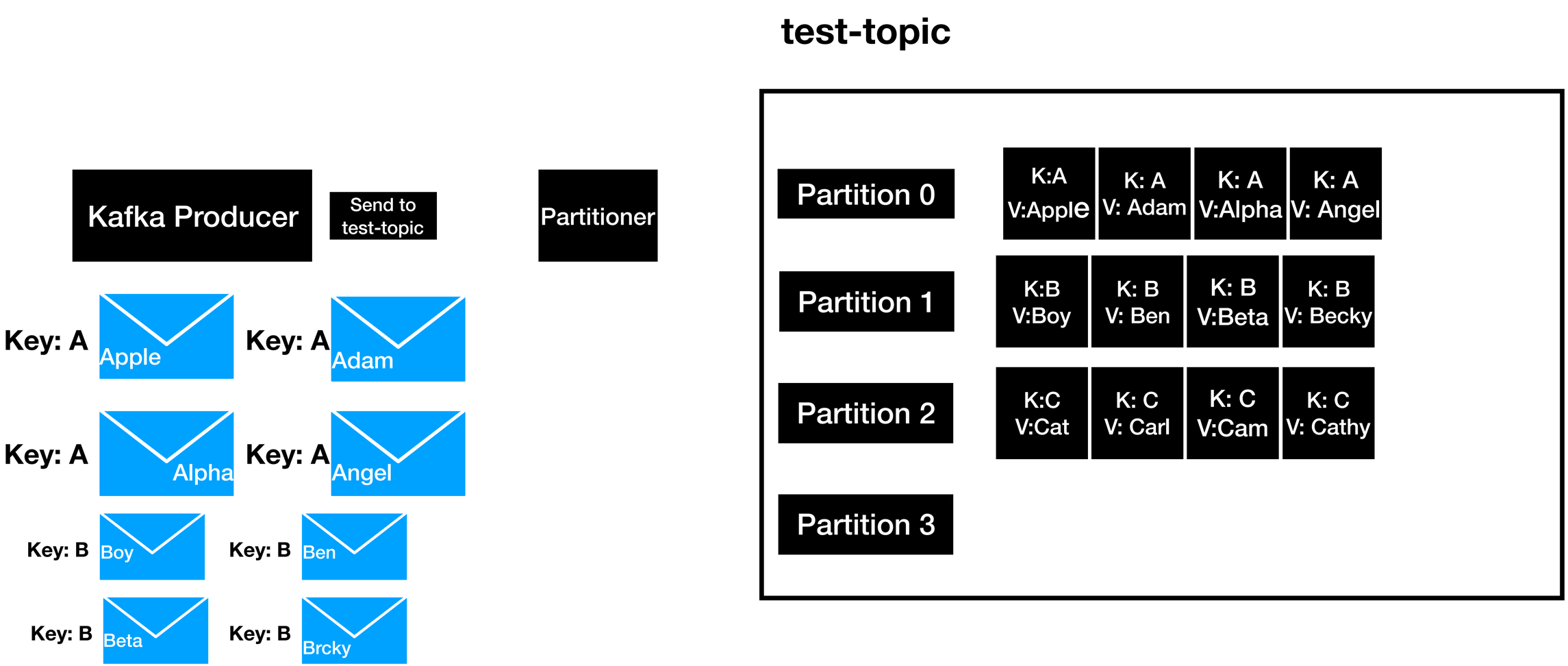


Broker registered with zookeeper

Sending Message Without Key

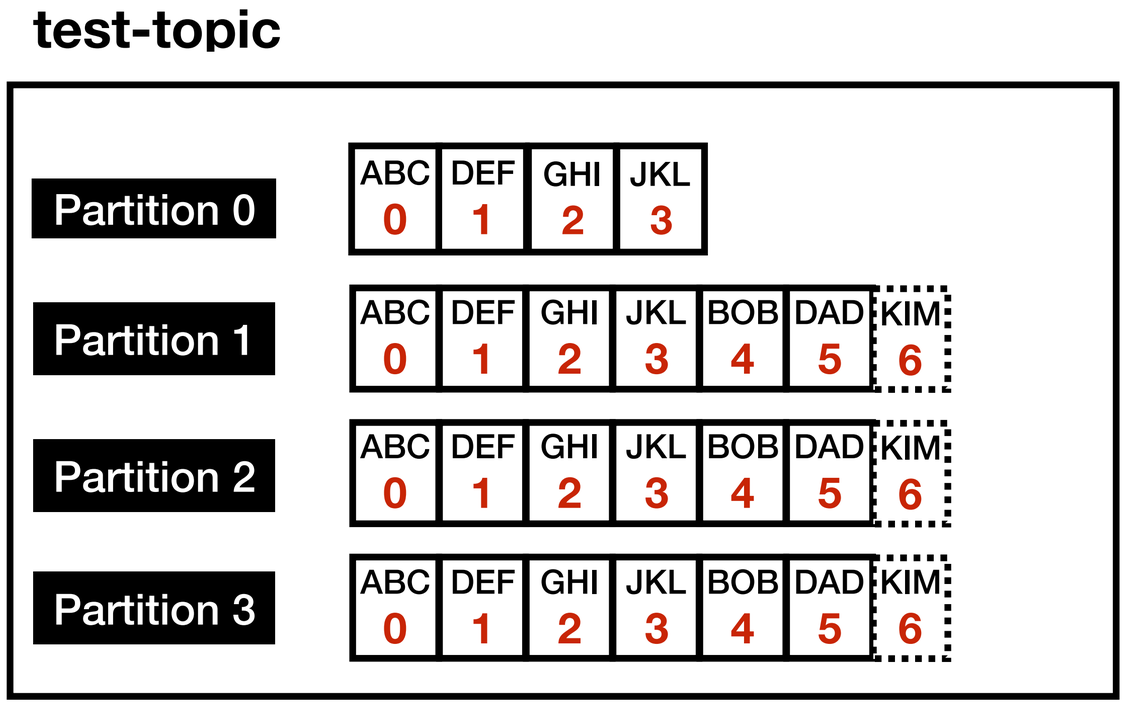


Sending Message With Key



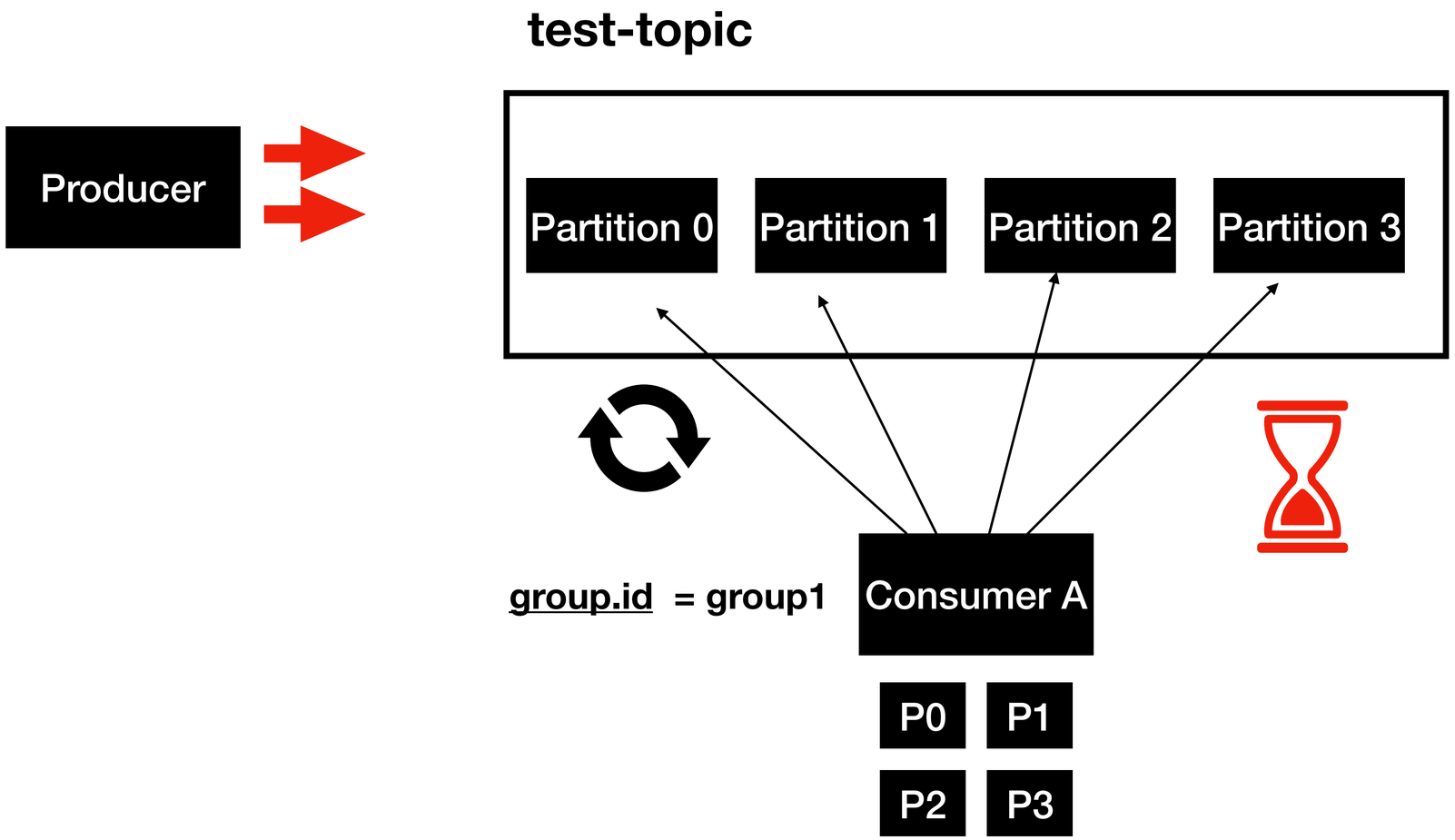
Consumer Groups

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

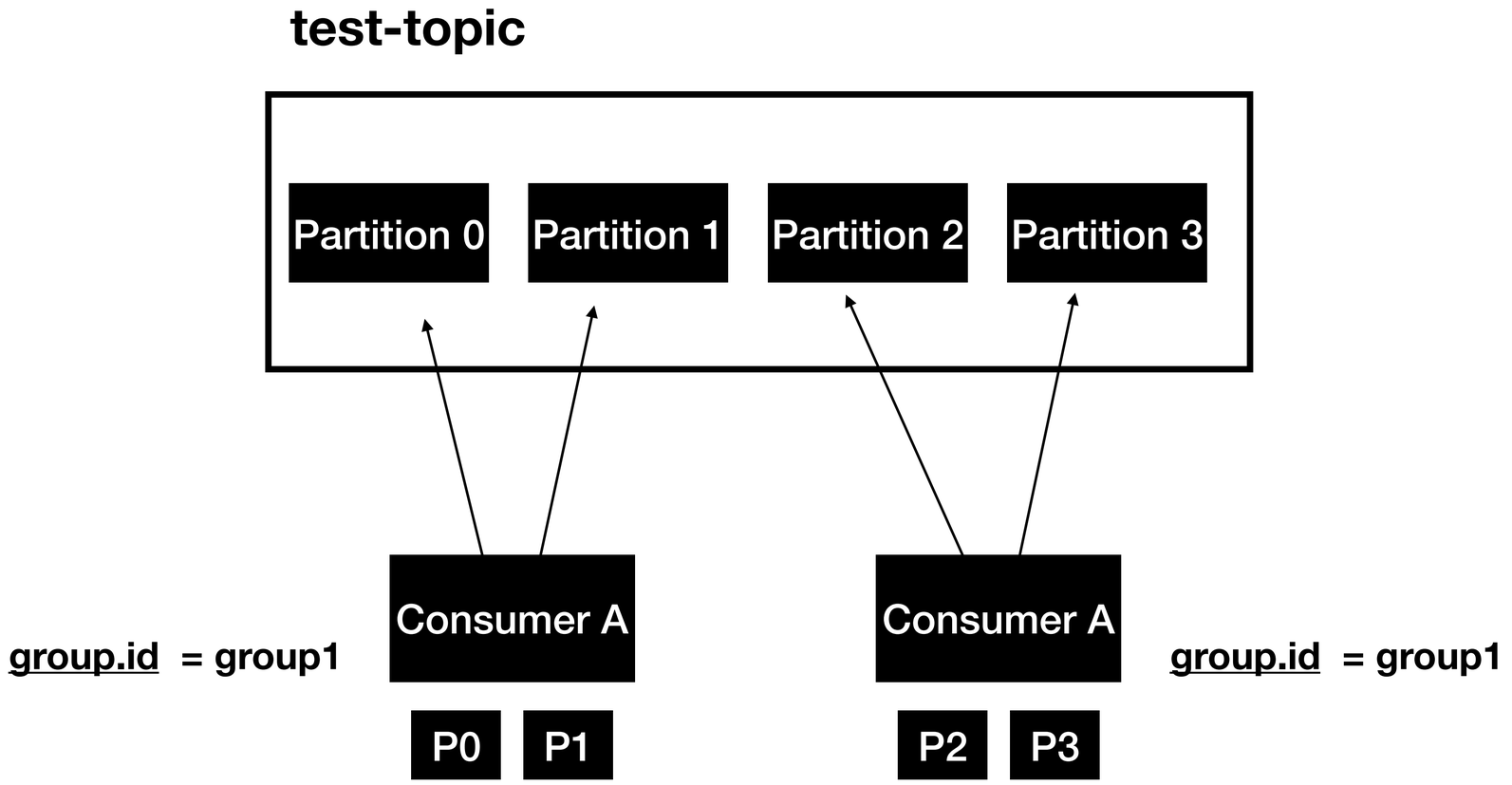


group.id = group1 Consumer 1

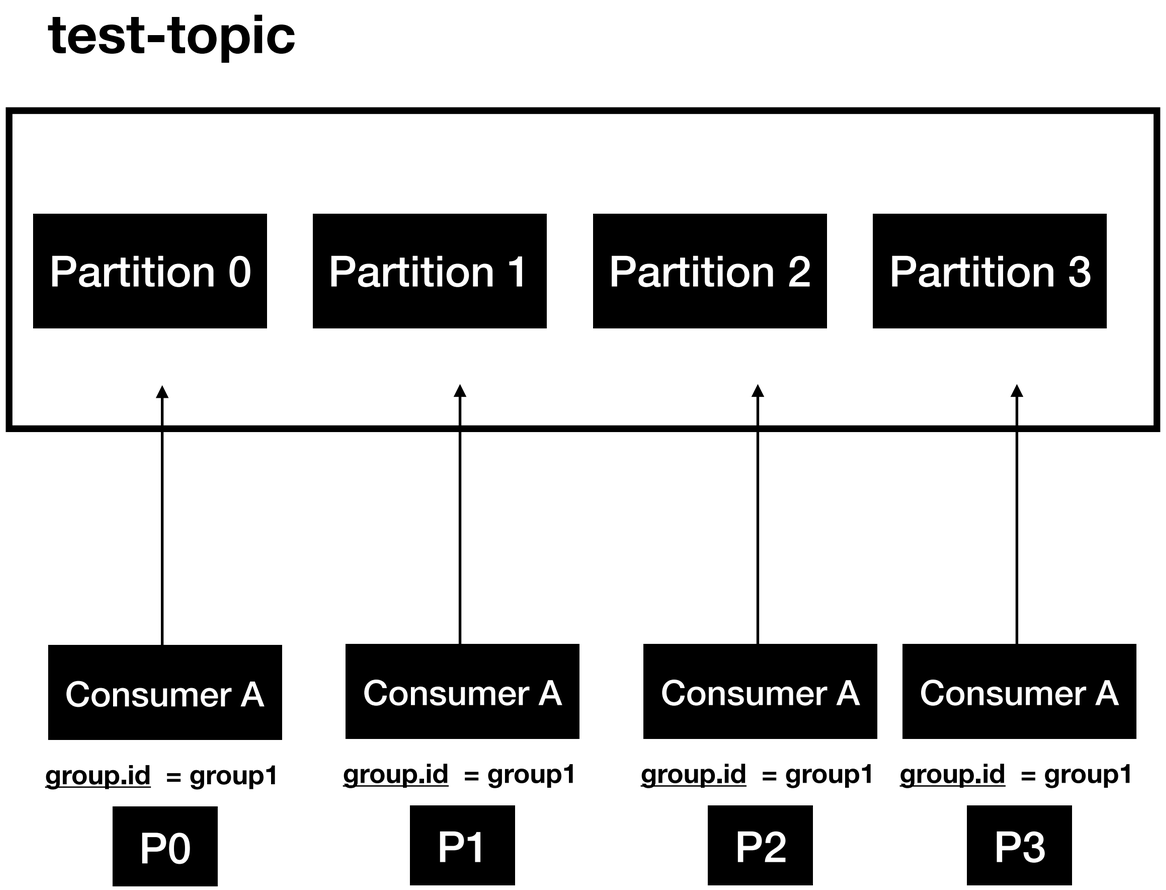
Consumer Groups



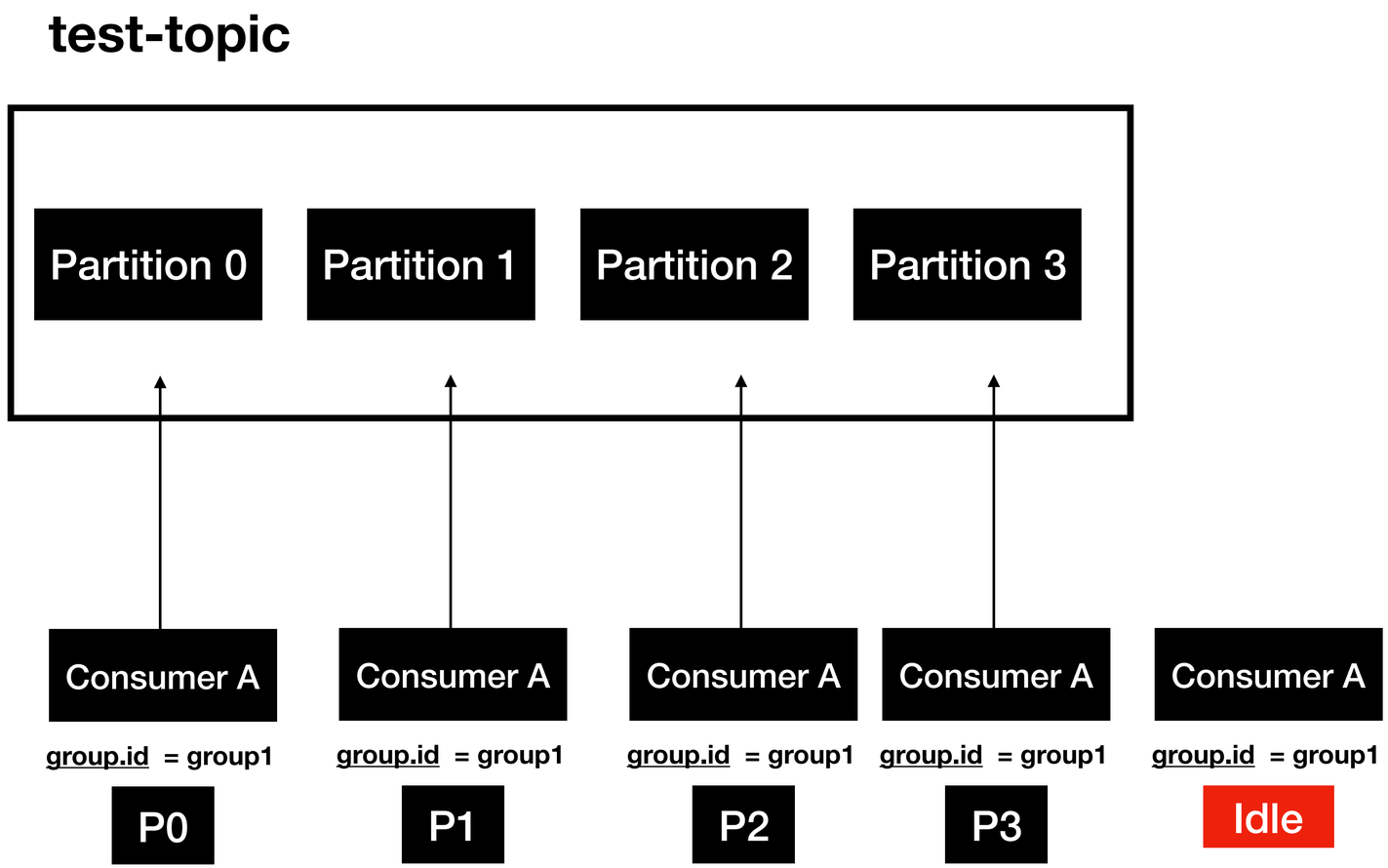
Consumer Groups



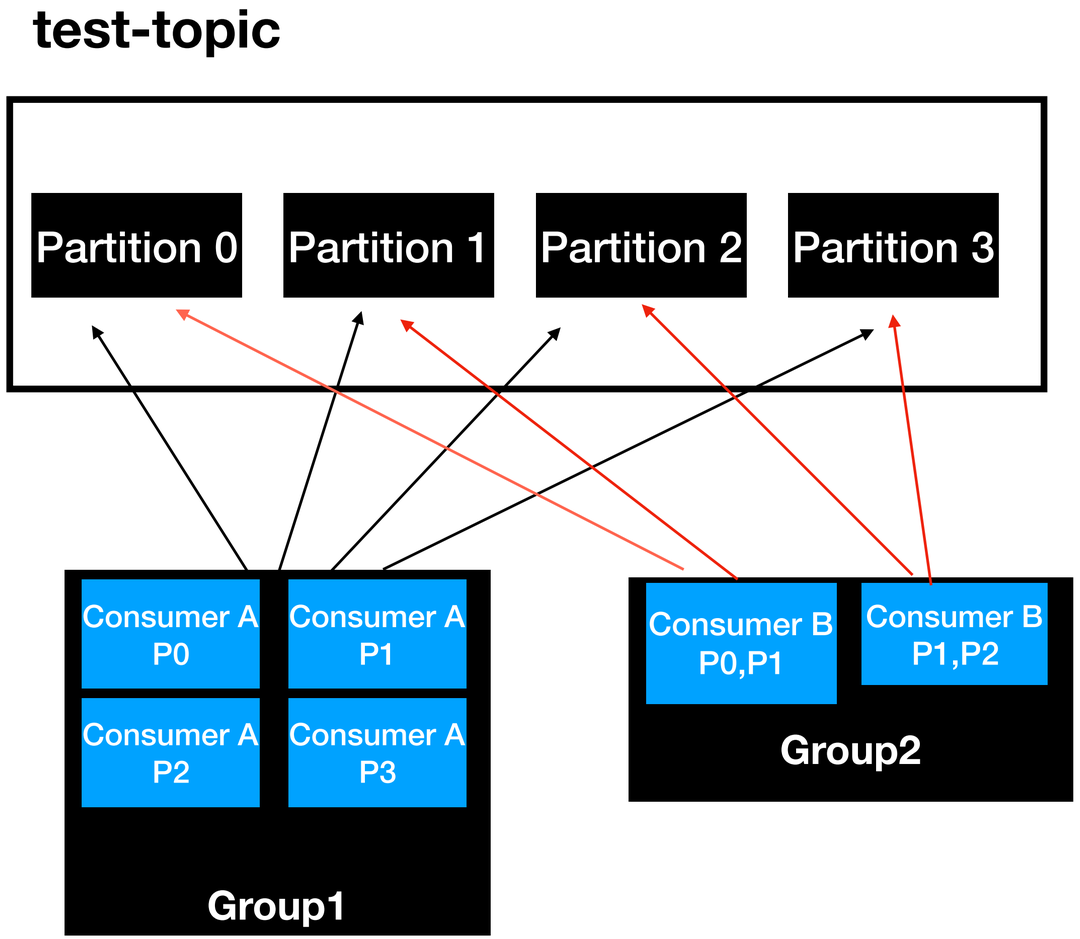
Consumer Groups



Consumer Groups



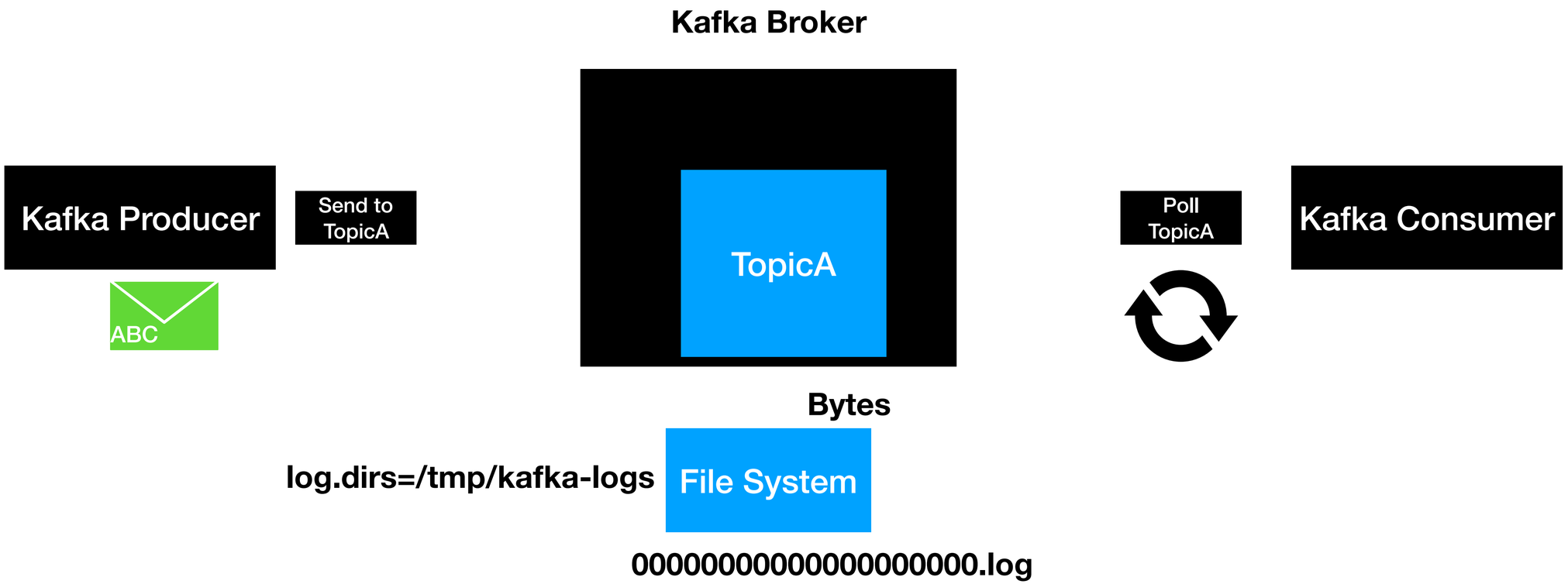
Consumer Groups



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

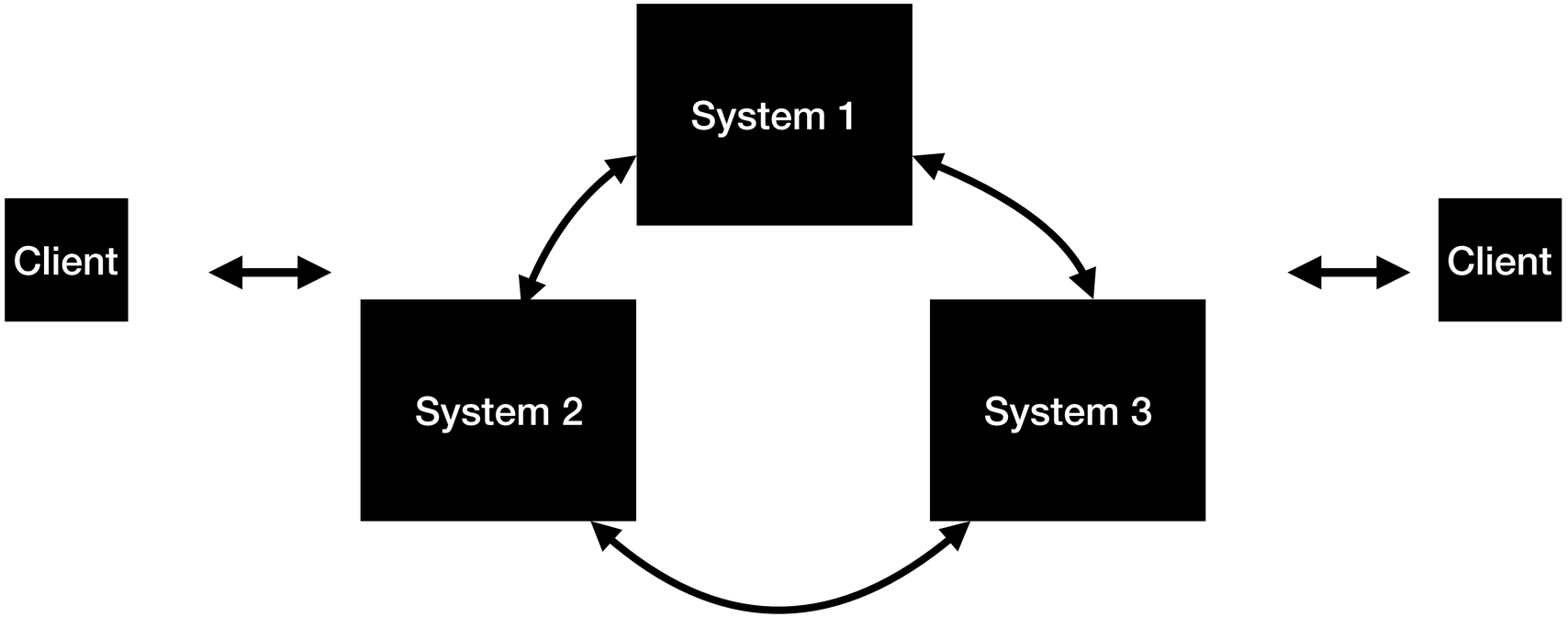

Kafka Broker
Kafka Producer
Send to TopicA
ABC
TopicA
Poll TopicA
Kafka Consumer
Bytes
log.dirs=/tmp/kafka-logs
File System
00000000000000000000.log

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)

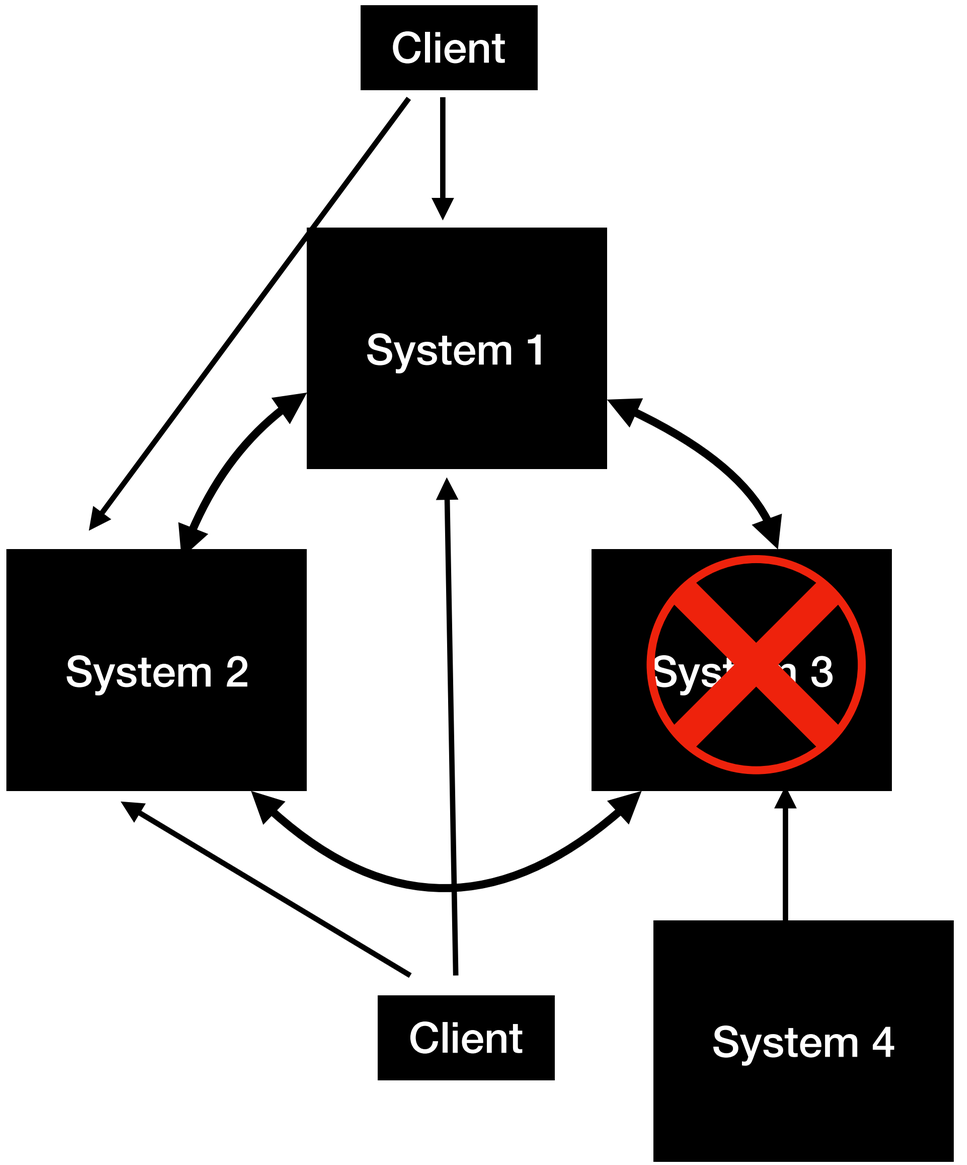
What is a Distributed System?

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

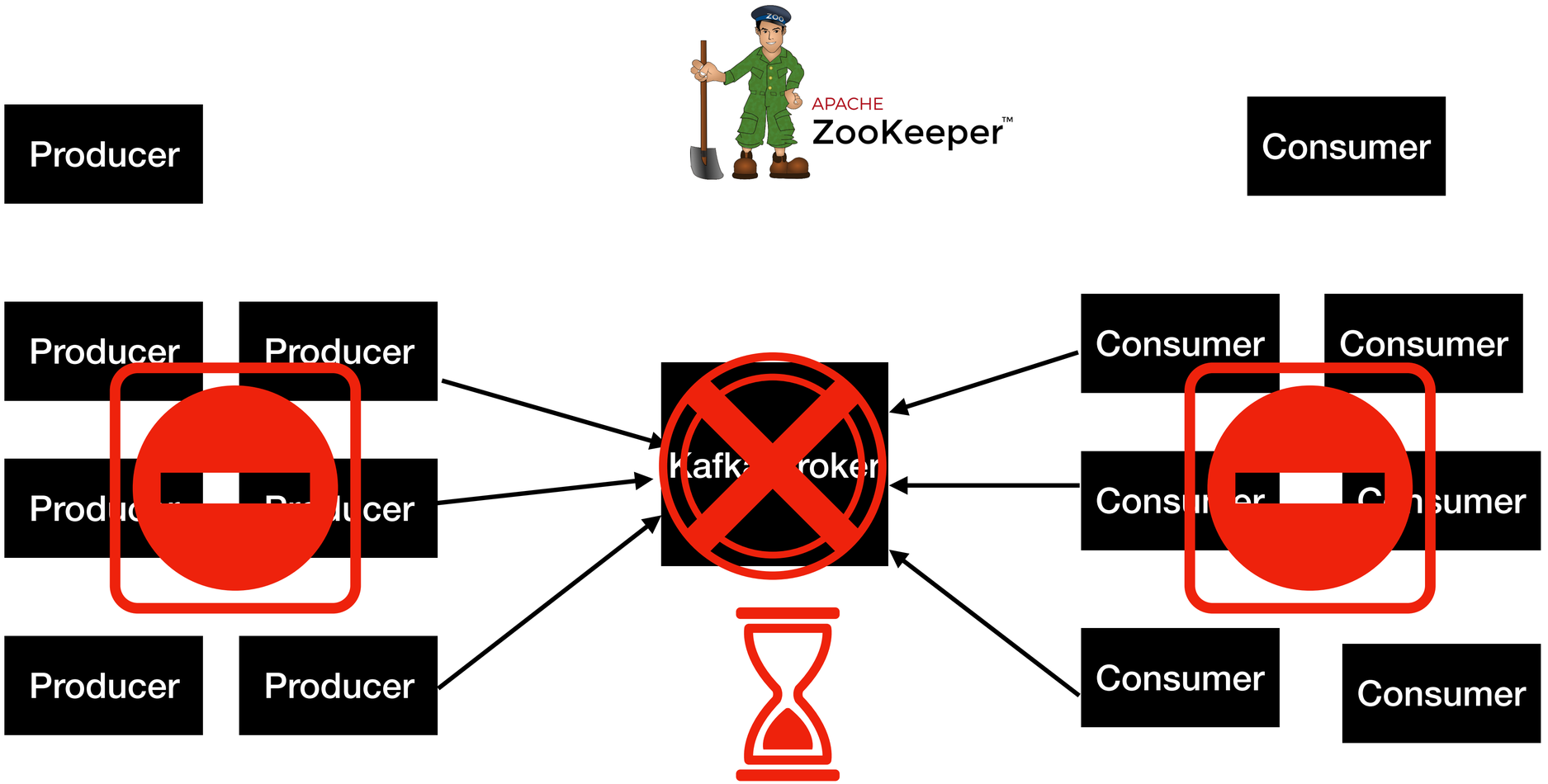


Characteristics of Distributed System

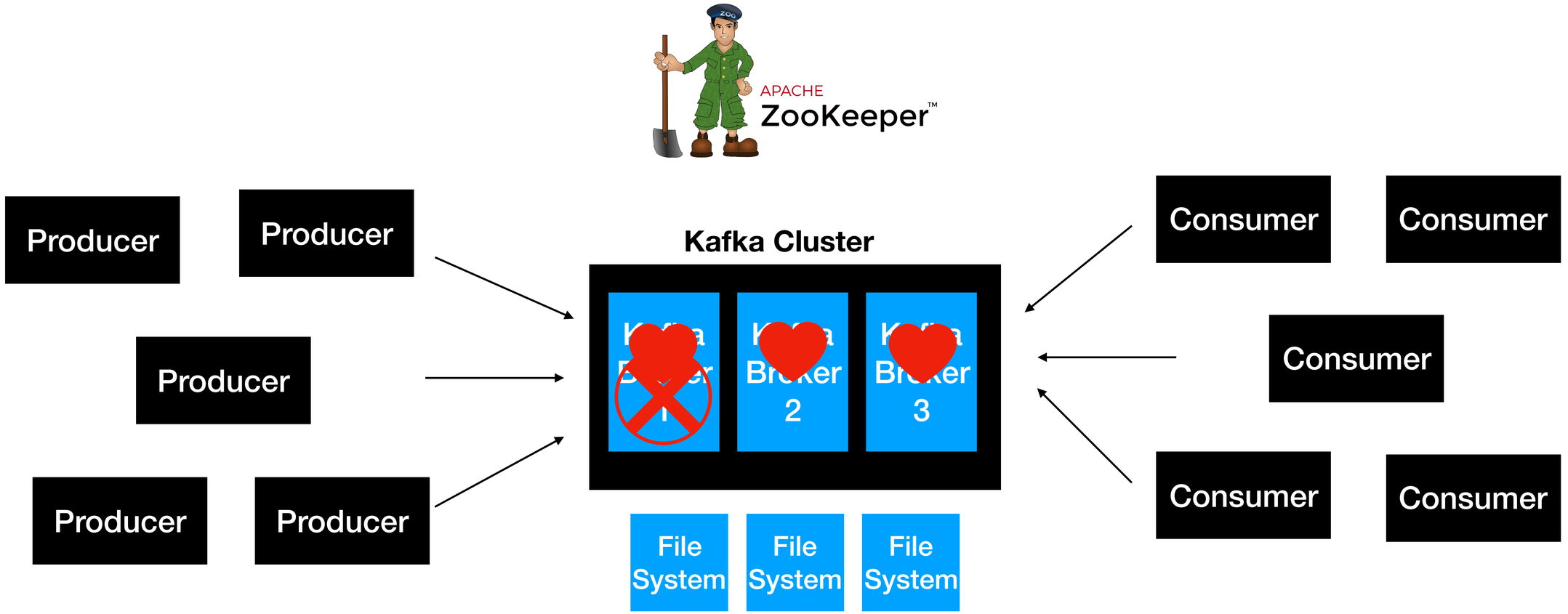
- Availability and Fault Tolerance
- Reliable Work Distribution
- Easily Scalable
- Handling Concurrency is fairly easy



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

Start Kafka Broker

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

Setting up Kafka Cluster

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

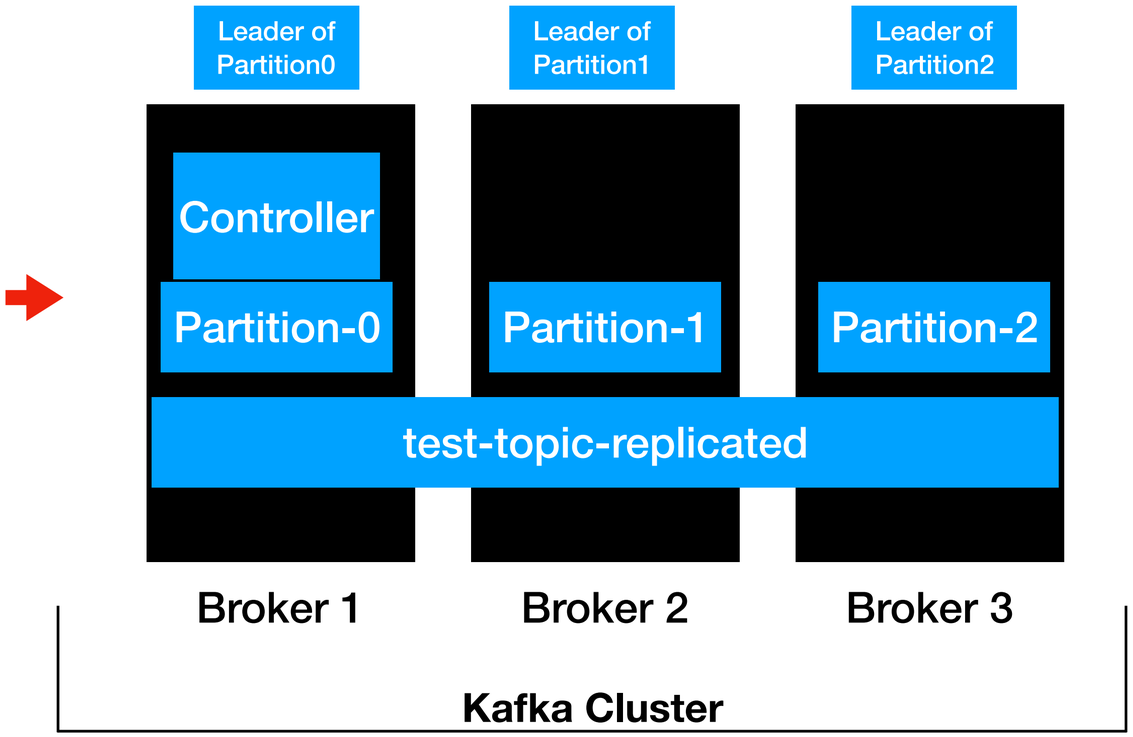
```
broker.id=<unique-broker-d>
listeners=PLAINTEXT://localhost:<unique-port>
log.dirs=/tmp/<unique-kafka-folder>
auto.create.topics.enable=false(optional)
```

Example: **server-1.properties**

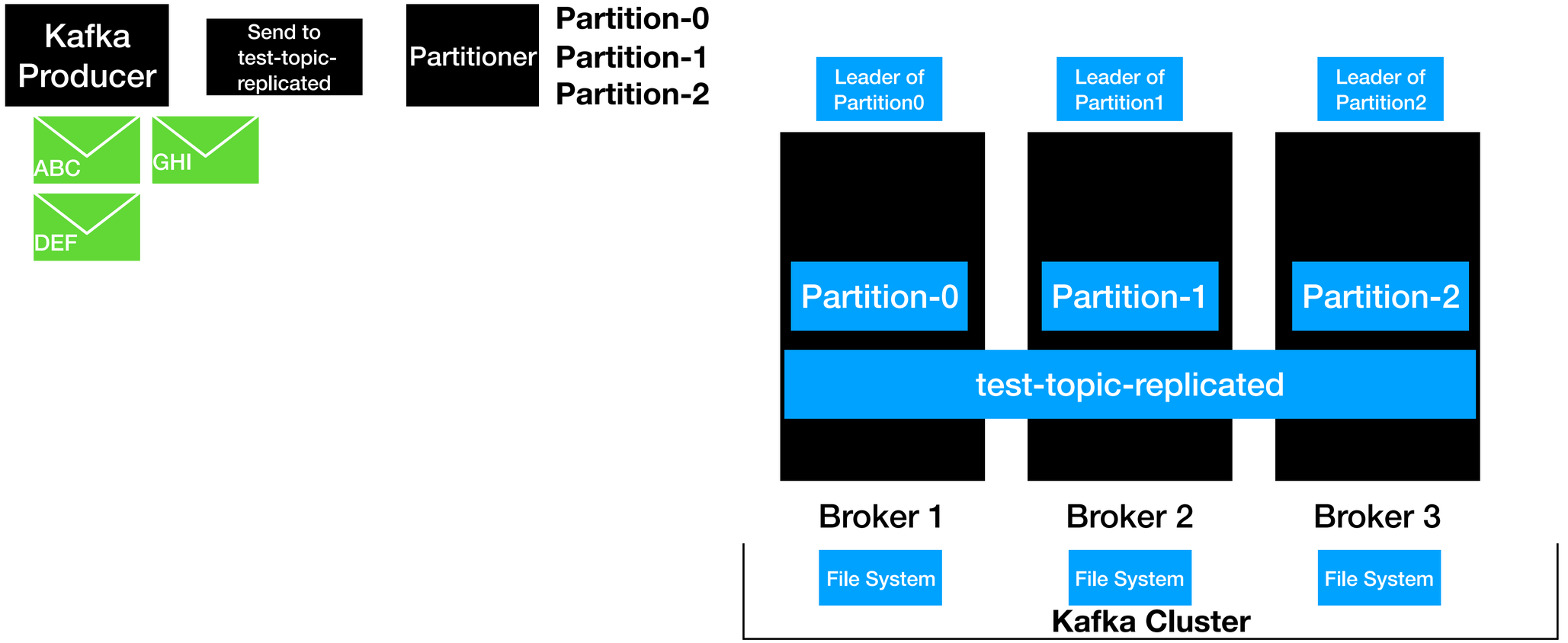
```
broker.id=1
listeners=PLAINTEXT://localhost:9093
log.dirs=/tmp/kafka-logs-1
auto.create.topics.enable=false(optional)
```

How Topics are distributed?

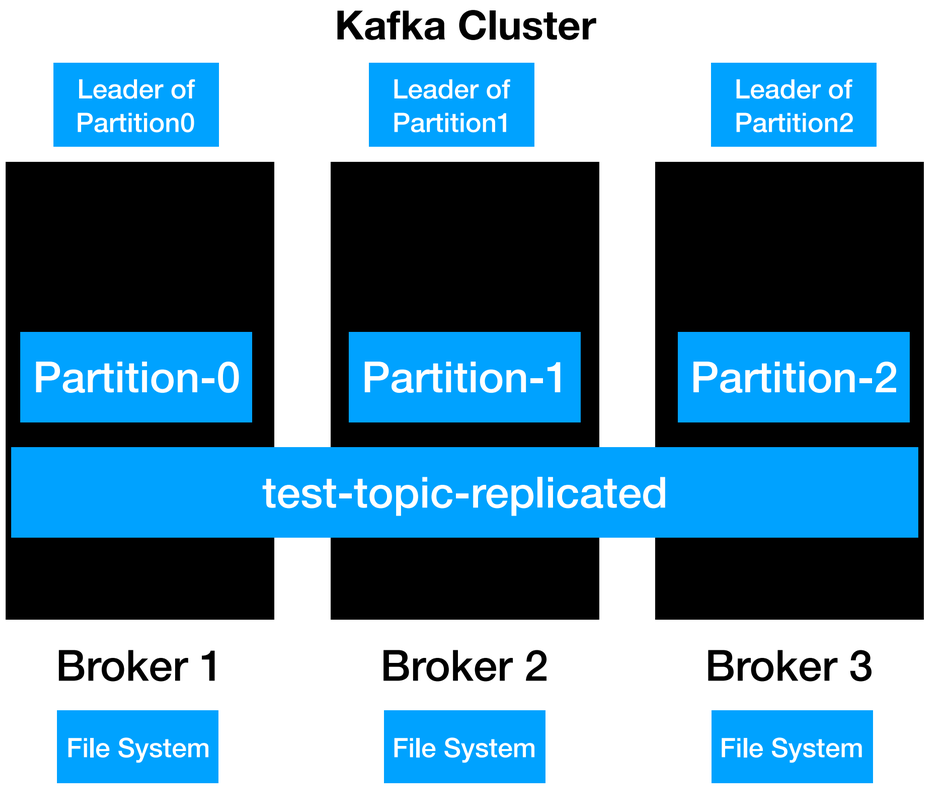
```
./kafka-topics.sh -
-create --topic test-topic-replicated
-zookeeper localhost:2181
--replication-factor 3
 --partitions 3
```

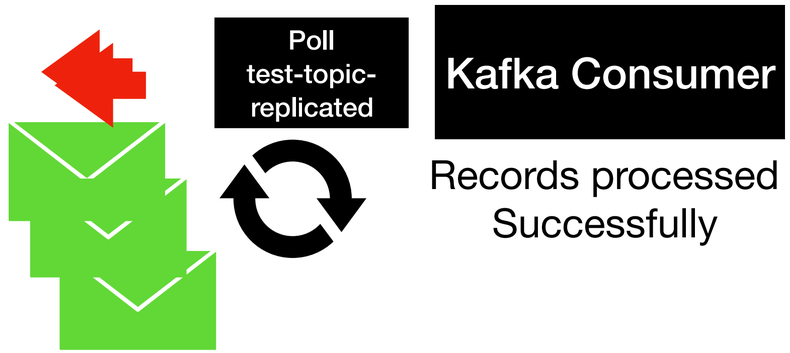


How Kafka Distributes Client Requests?
Kafka Producer

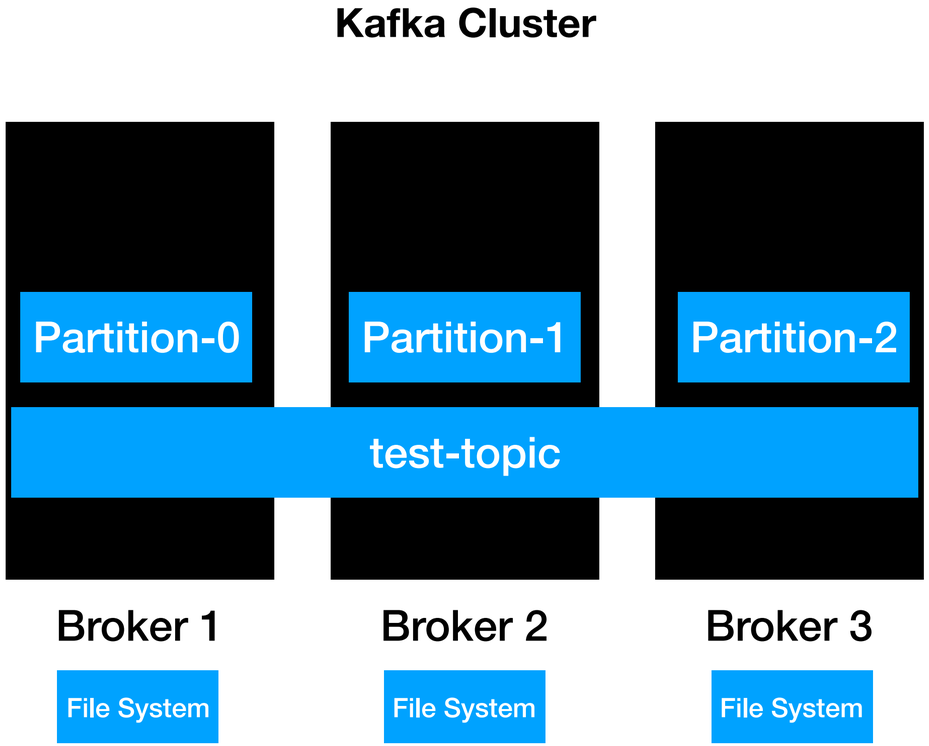


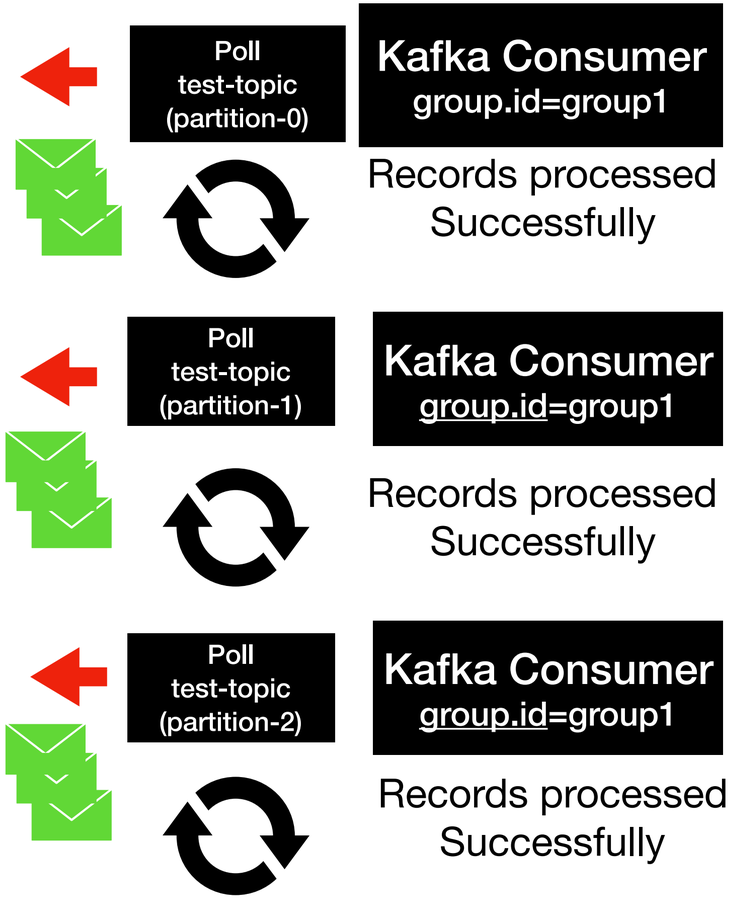
How Kafka Distributes Client Requests?
Kafka Consumer





How Kafka Distributes Client Requests?
Kafka Consumer Groups

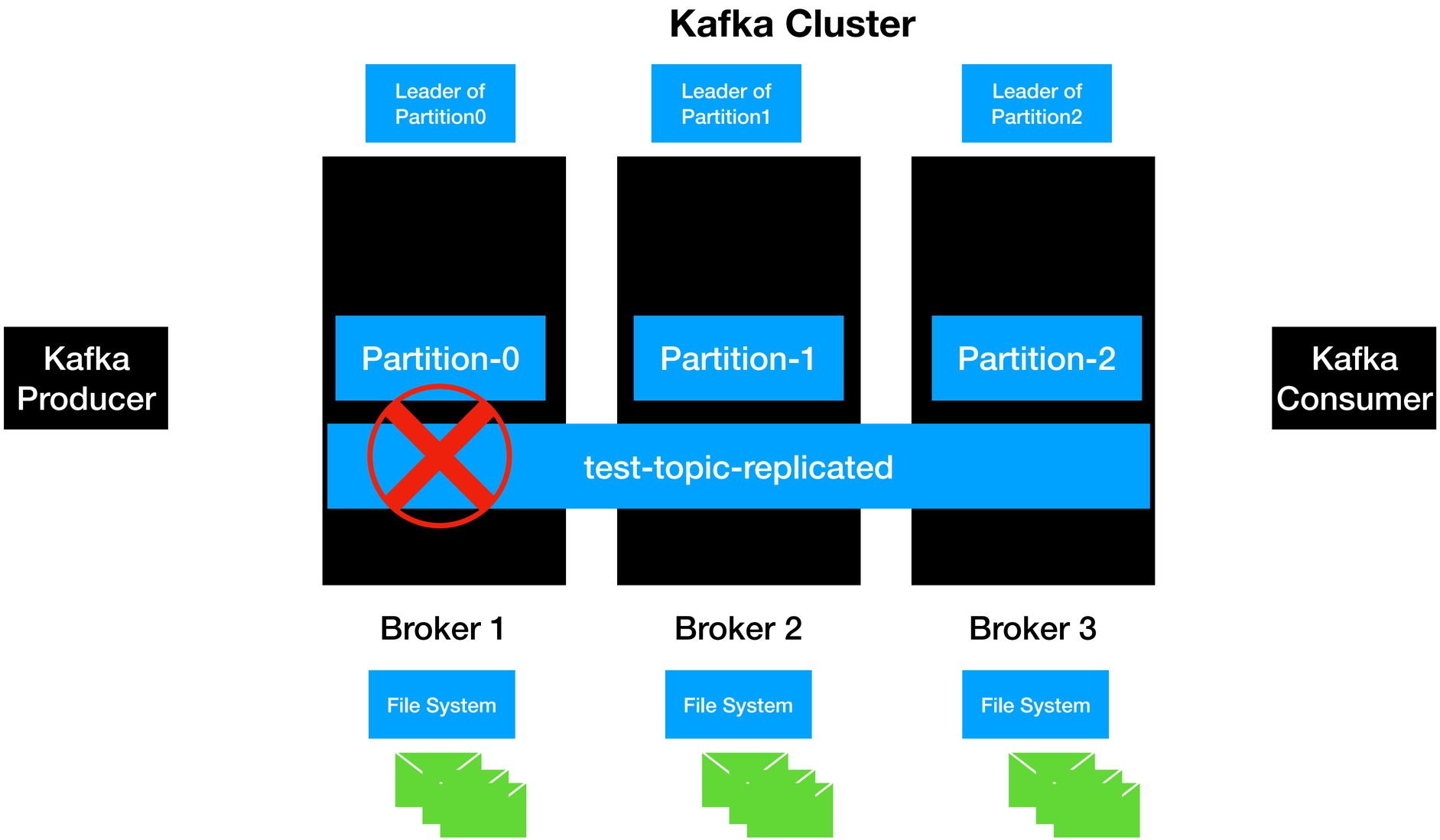




Summary : How Kafka Distributes the Client Requests?

- 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

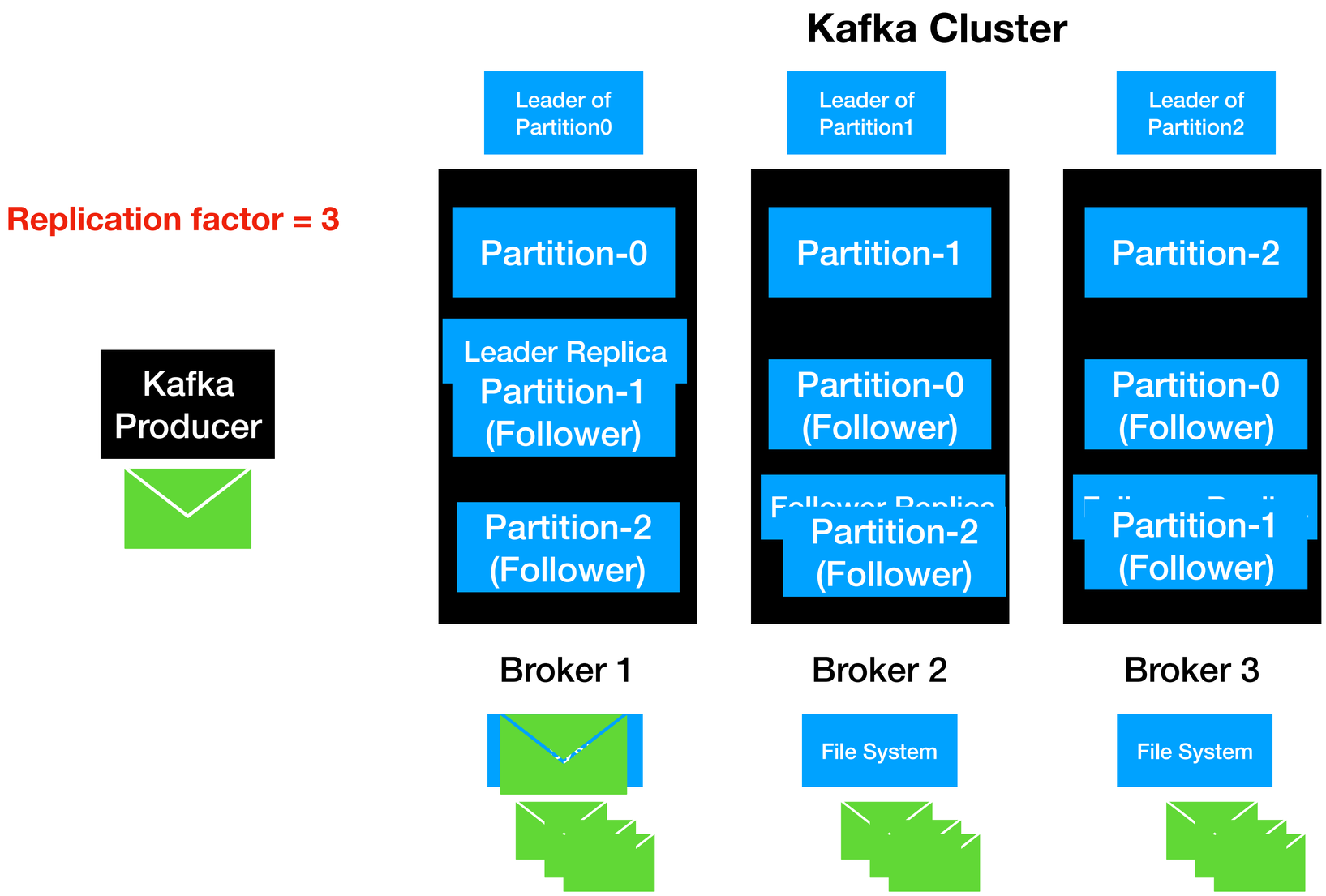
How Kafka handles Data loss?



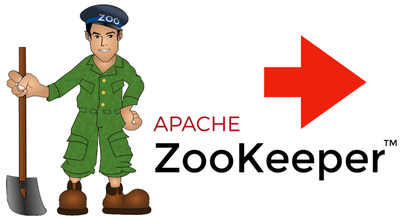
Replication

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

Replication



Replication



Kafka Cluster

Kafka Producer

Kafka Consumer

Broker 1	Broker 2	Broker 3
Leader of Partition0	Leader of Partition 0 & 1	Leader of Partition2
Partition-0	Partition-1	Partition-2
Partition-1 (Follower)	Partition-0 (Follower)	Partition-0 (Follower)
Partition-2 (Follower)	Partition-2 (Follower)	Partition-1 (Follower)
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

Library Inventory

Library Inventory Flow

Librarian

Library Inventory

Library Inventory Architecture

Librarian

MicroService 1

API

Kafka Producer

Library Event Producer

Library-events

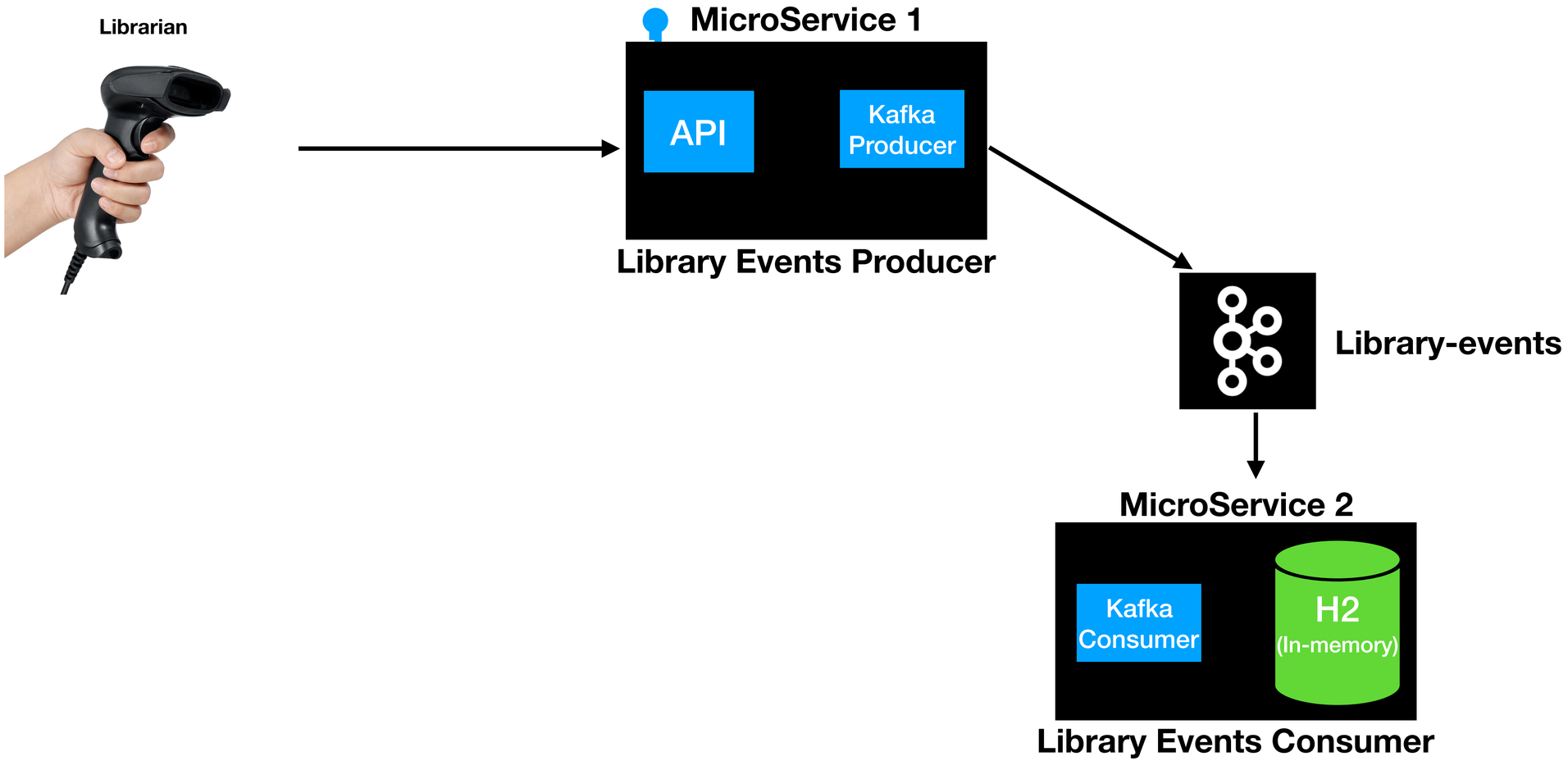
MicroService 2

Kafka Consumer

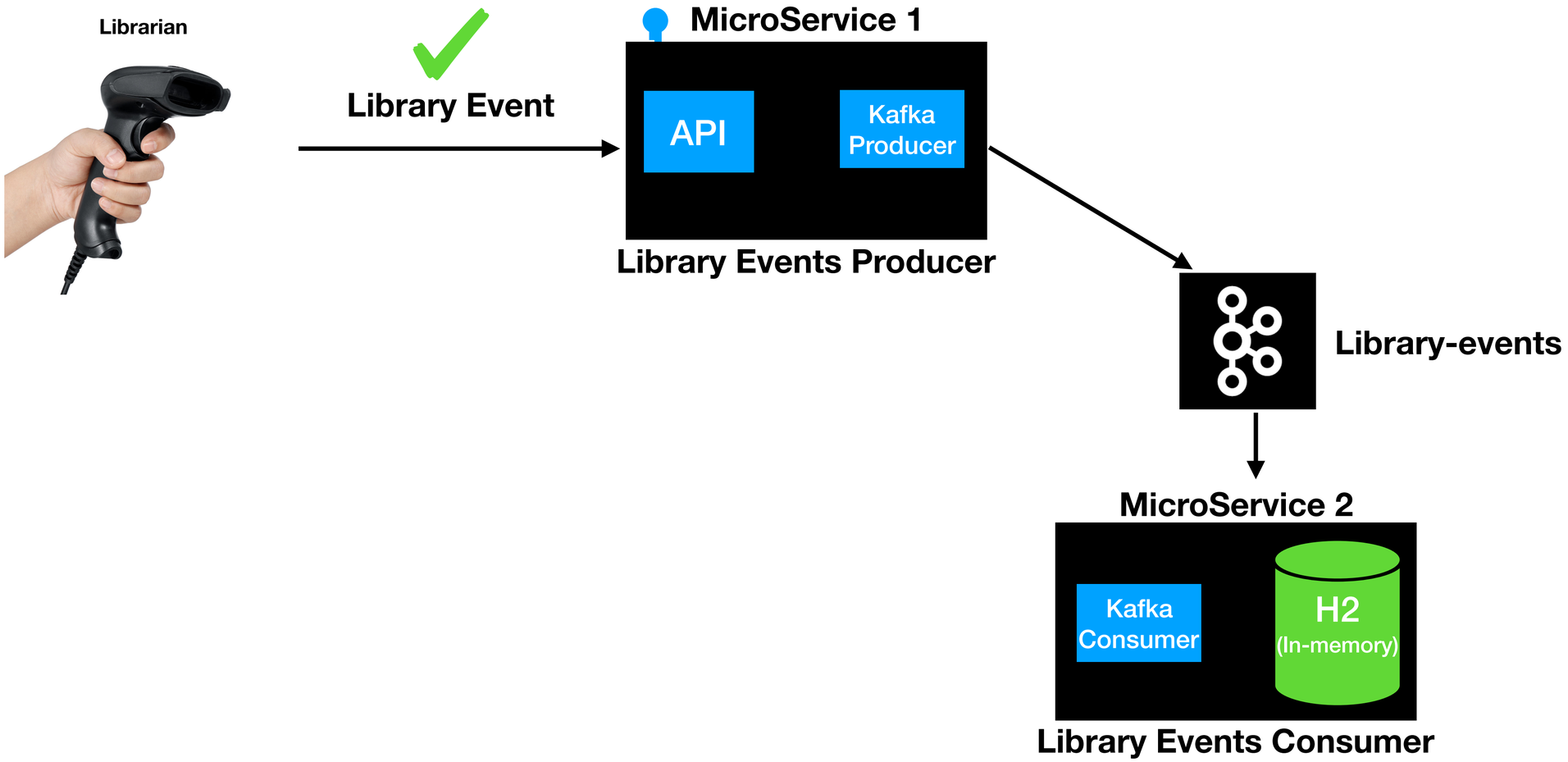
H2 (In-memory)

Library Event Consumer

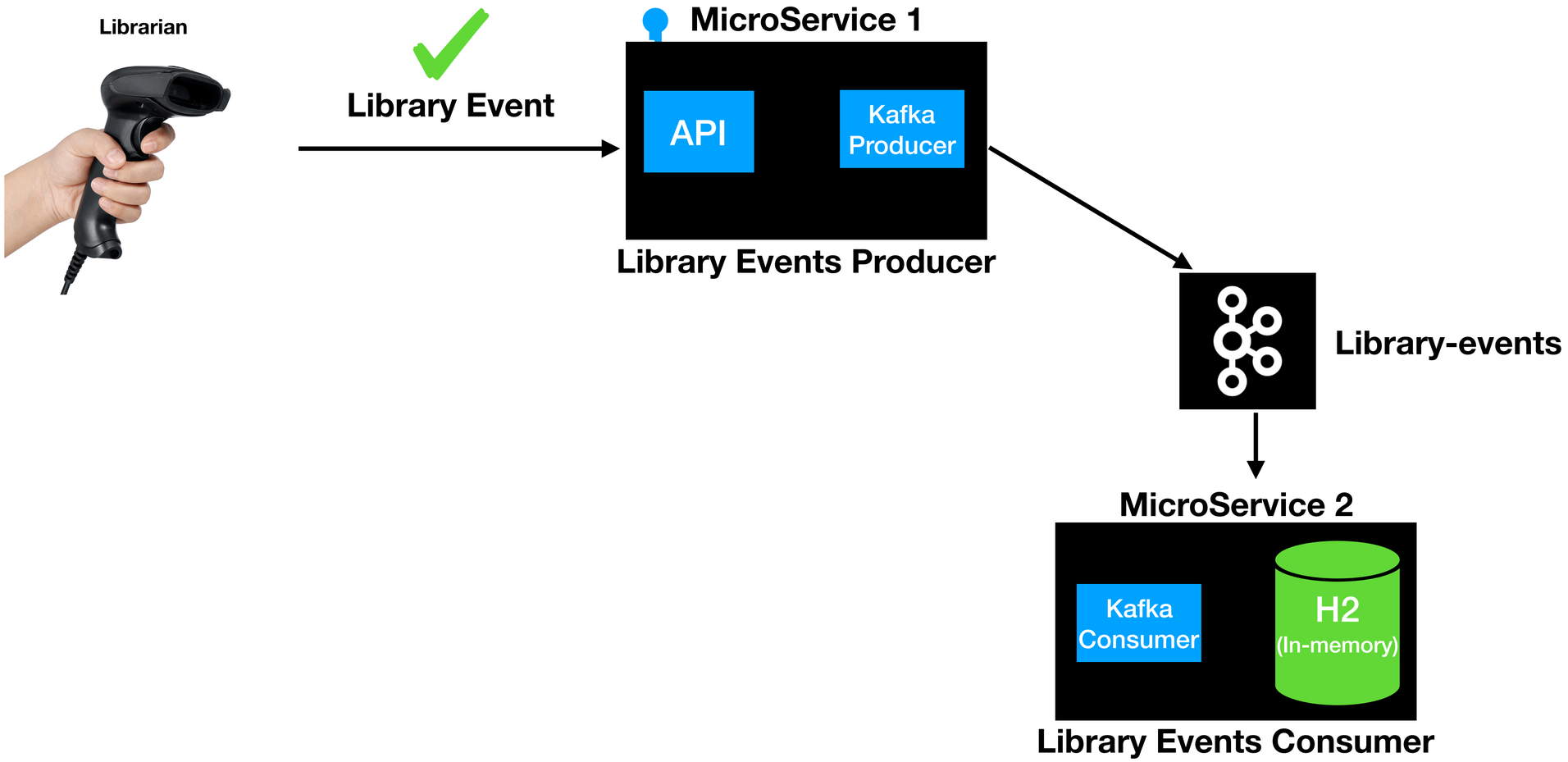
Library Inventory Architecture



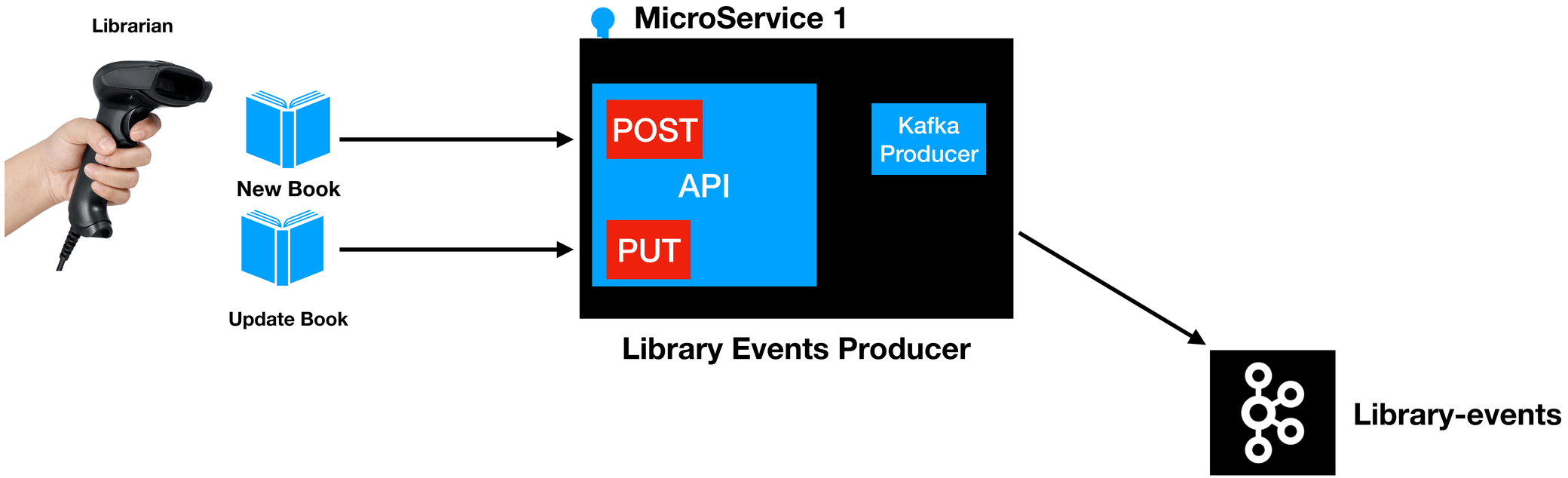
Library Event Domain



Library Event Domain



Library Events Producer API



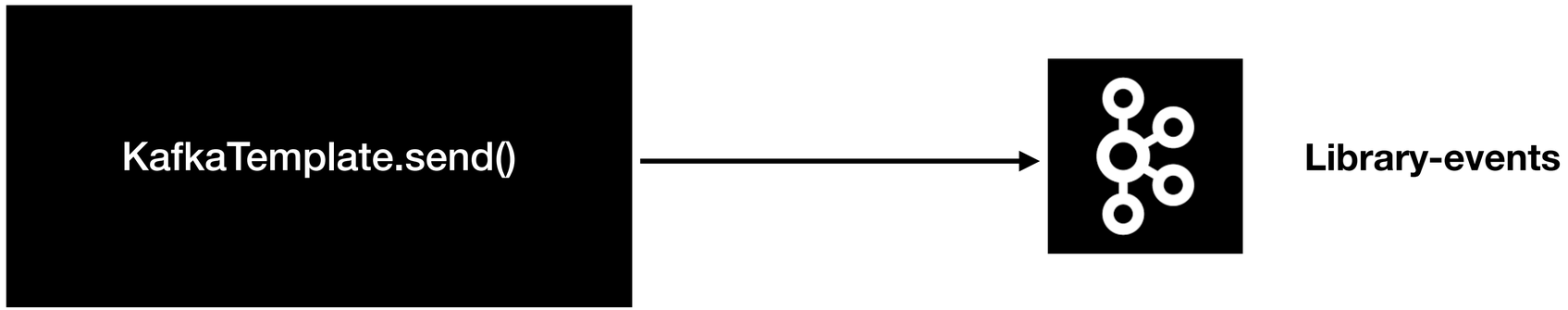
KafkaTemplate

Kafka Producer in Spring

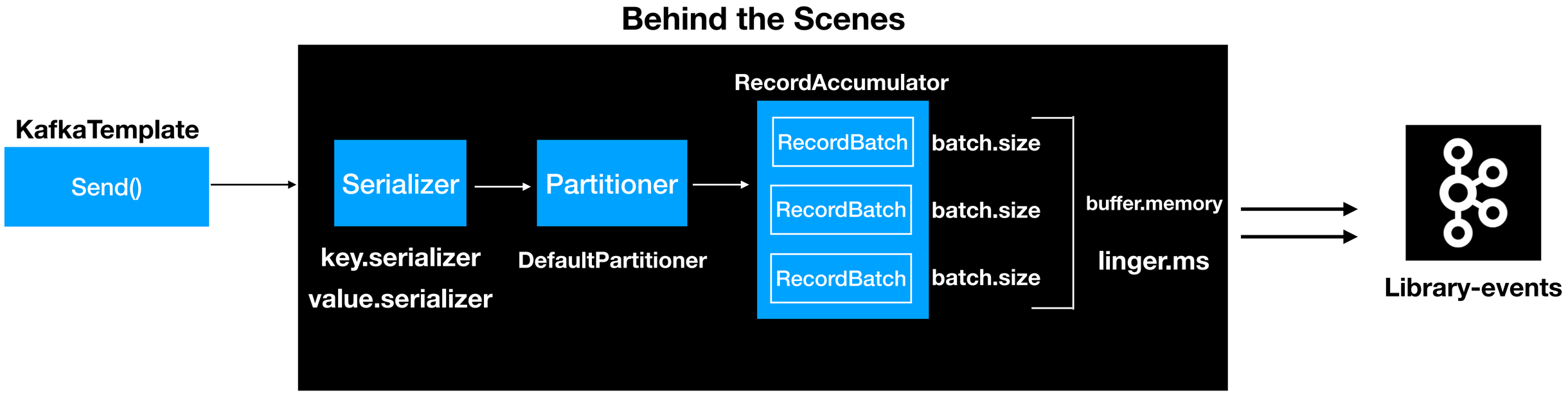
KafkaTemplate

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

How KafkaTemplate Works ?



KafkaTemplate.send()



Configuring KafkaTemplate

Mandatory Values:

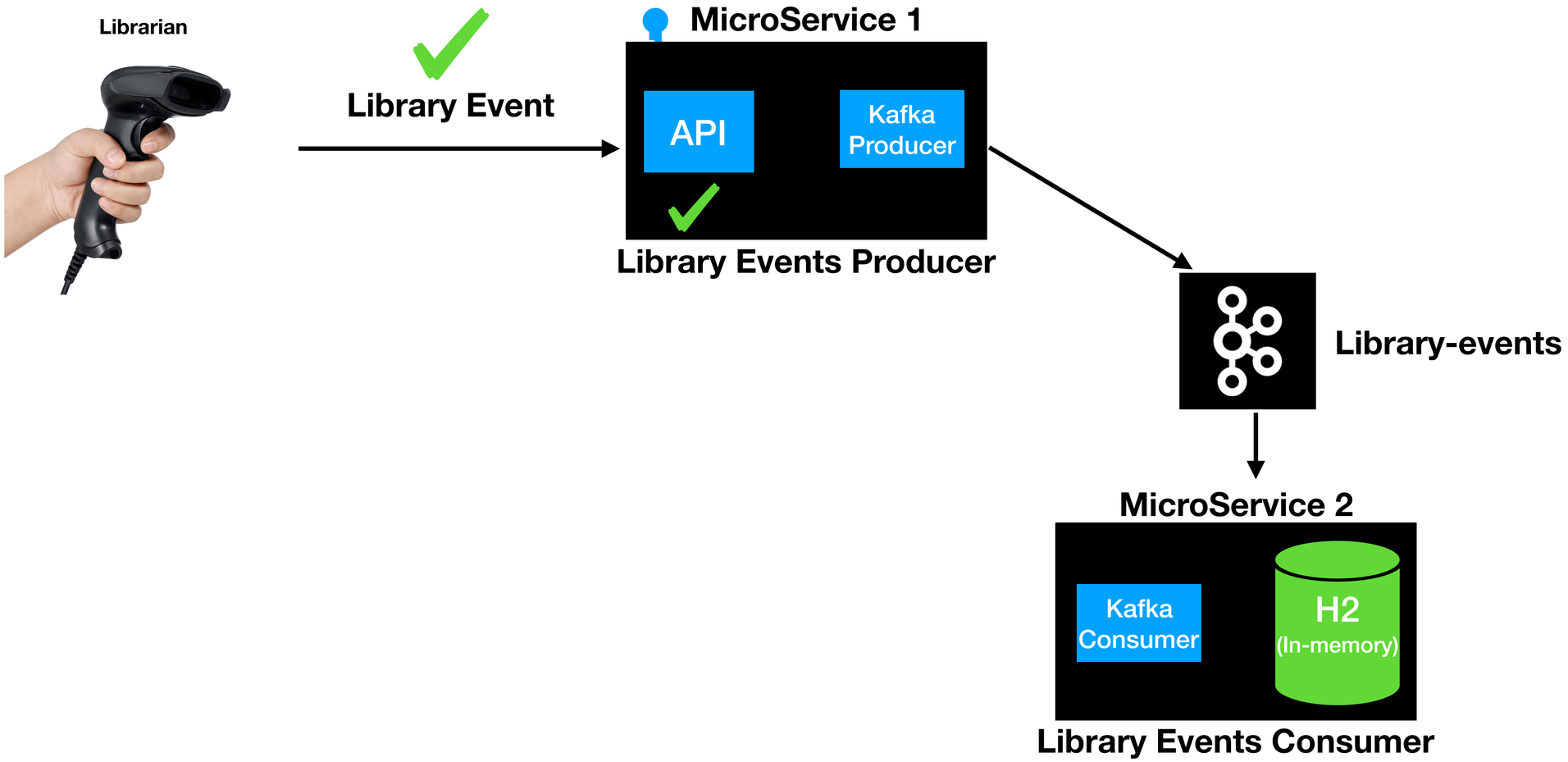
```
bootstrap-servers: localhost:9092,localhost:9093,localhost:9094
key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

KafkaTemplate AutoConfiguration

application.yml

```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
```

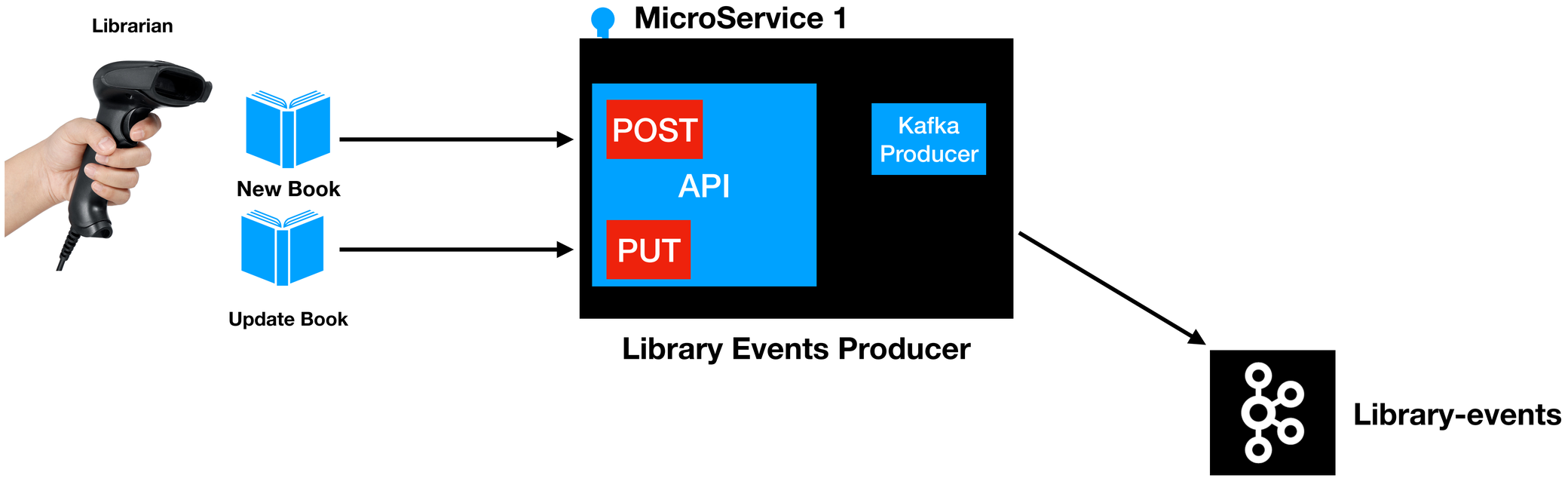
Library Inventory Architecture



KafkaAdmin

- Create topics Programmatically
- 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

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"
   }
}
```

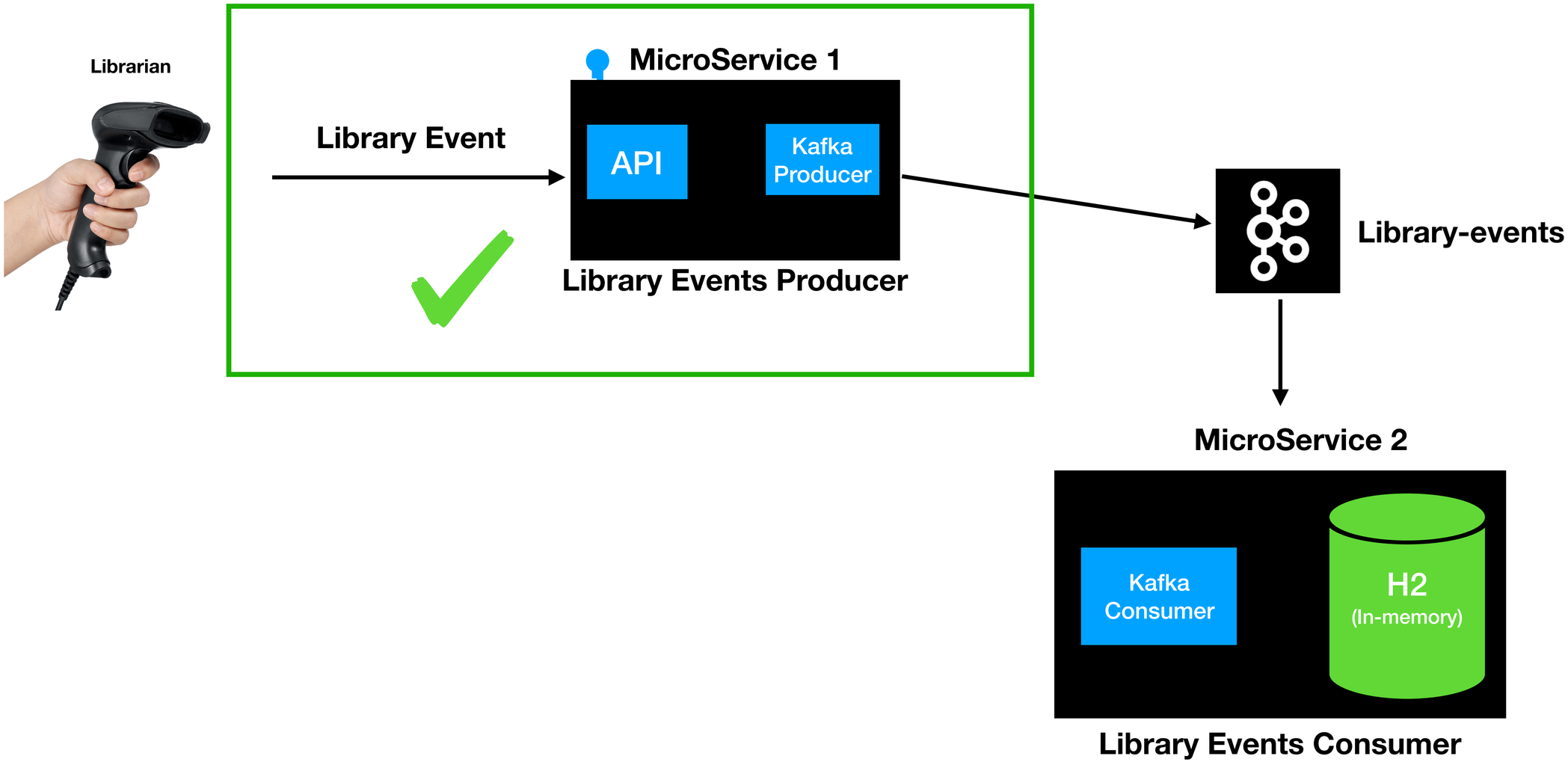
Kafka Producer Configurations

- acks
  - 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) 

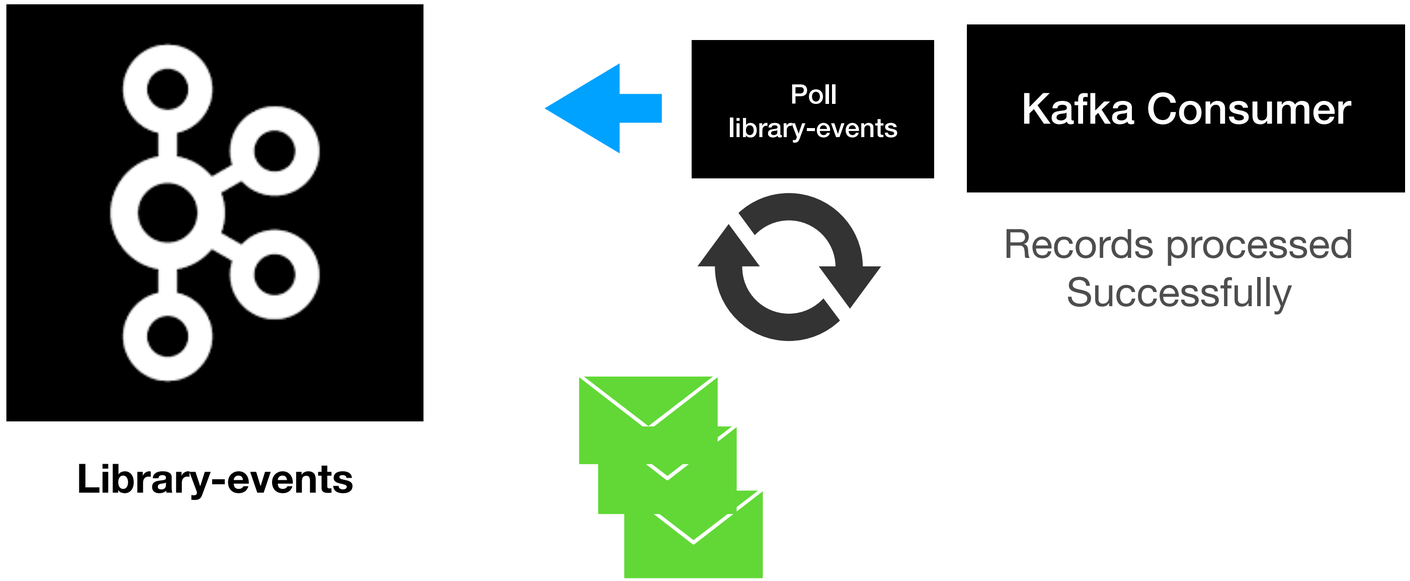
Kafka Producer Configurations

- 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



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

@KafkaListener

- This is the easiest way to build Kafka Consumer

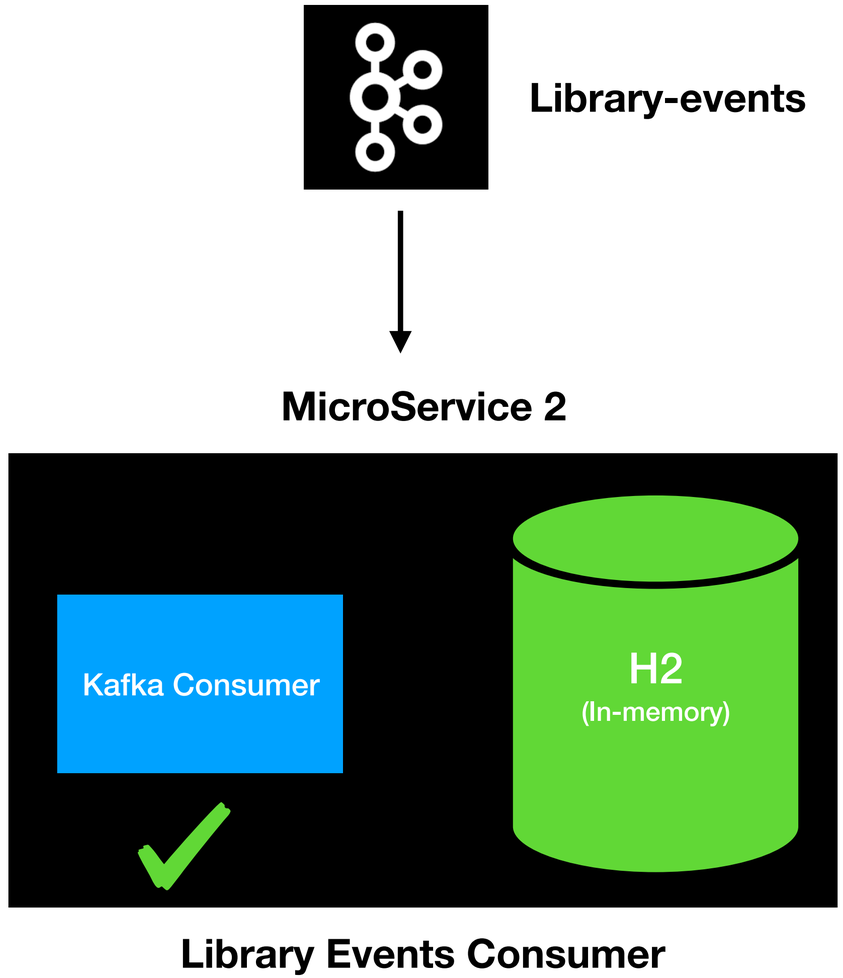
- KafkaListener Sample Code

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

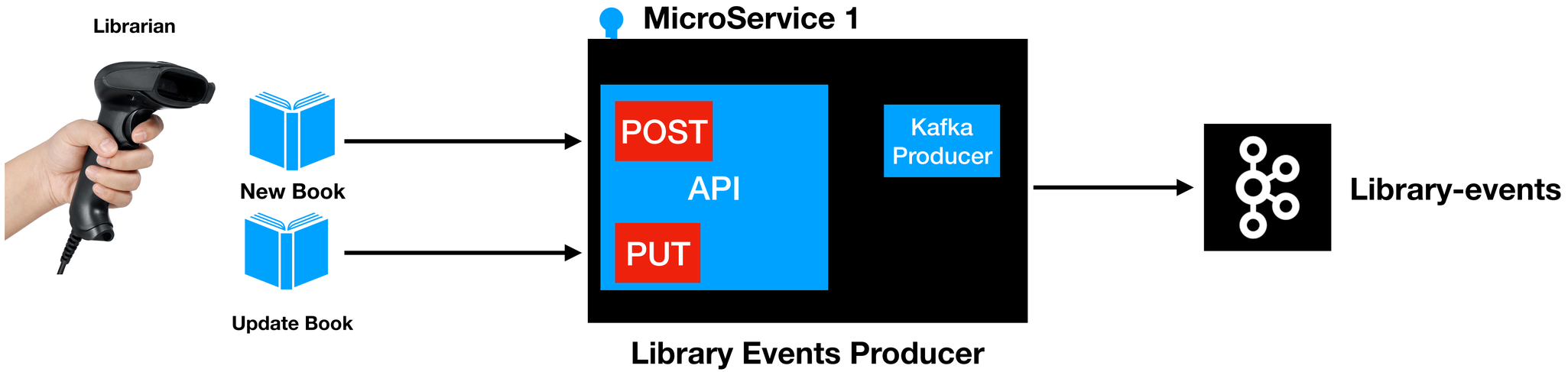
- Configuration Sample Code

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

Library Events Consumer

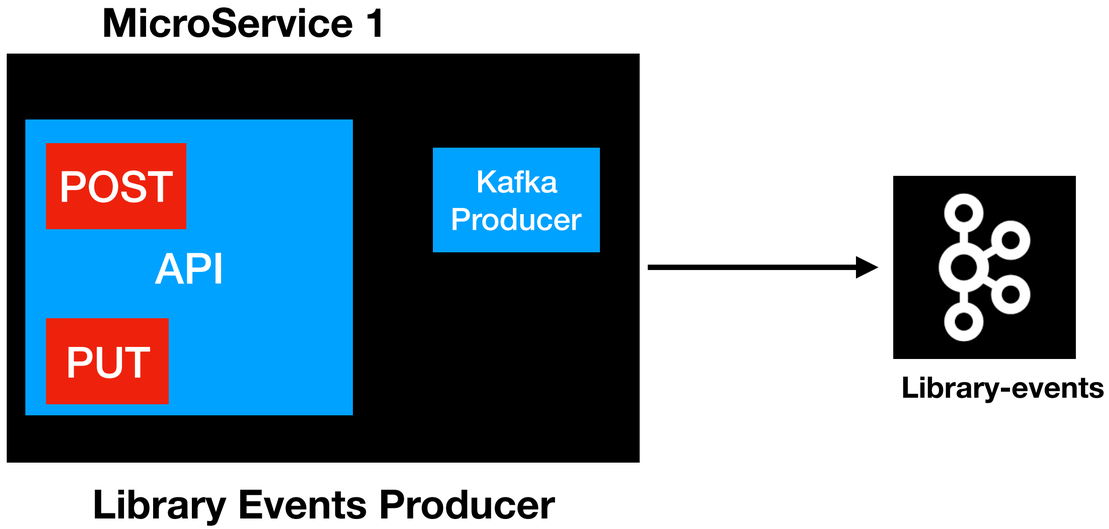


Library Events Producer API



Kafka Producer Errors

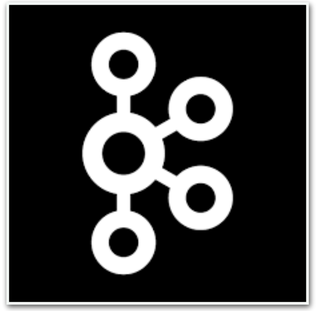
- Kafka Cluster is not available
- If **acks= all ,** some brokers are not available
- **min.insync.replicas** config
  - **Example : min.insync.replicas = 2**, But only one broker is available



min.insync.replicas

min.insync.replicas = 2

Kafka Cluster

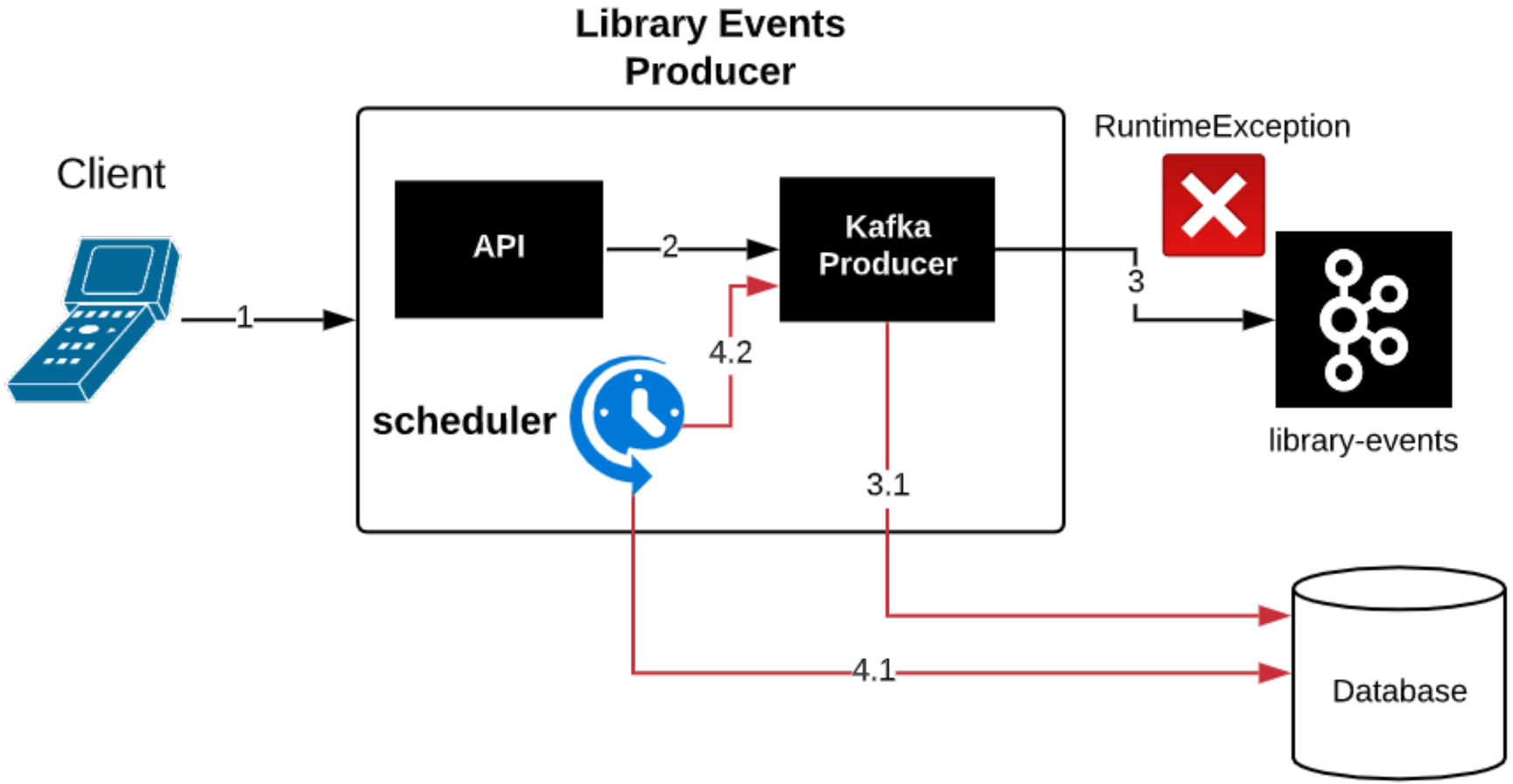


Broker 1

Broker 2

Broker 2

Retain/Recover Failed Records



Retain/Recover Failed Records