

Apache Kafka with Java





Apache Kafka

- Why Apache Kafka ?
- What is Apache Kafka ?
- Architecture
- Topologies and Tools
- Workshop



Working with Java

- Working with Spring Boot
- Producer
- Consumer
- Consumer group
- Message key



Apache Kafka

<https://kafka.apache.org/>

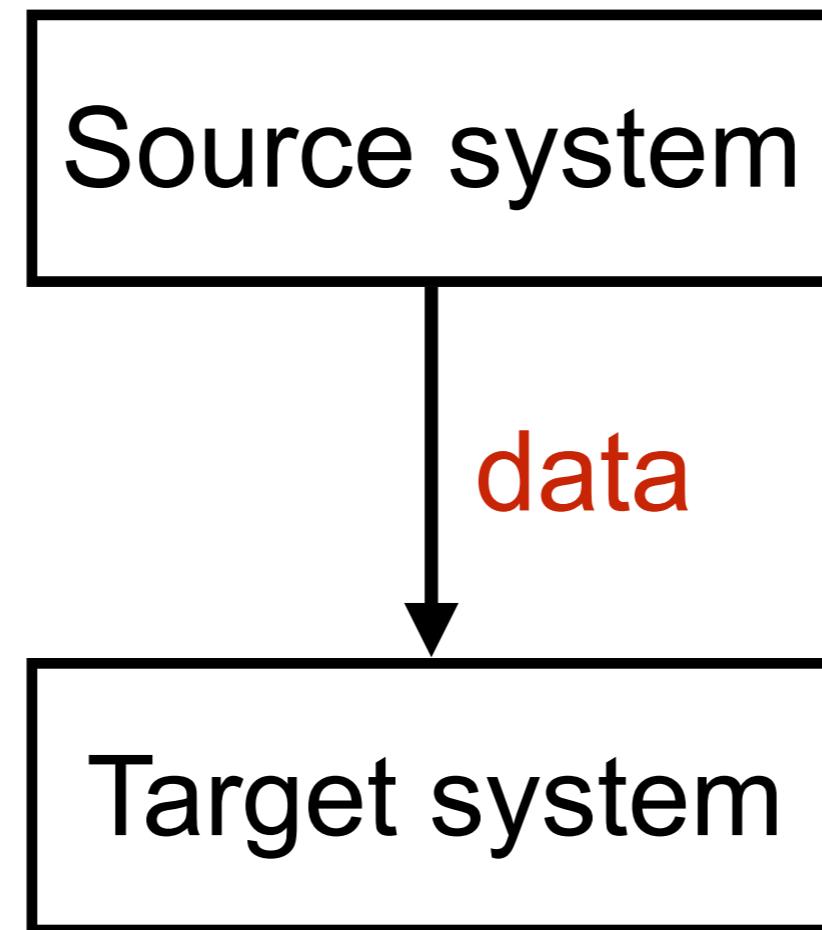


Kafka with Java

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

Why

Starting with simple



Why

More source systems

More target systems

More protocols

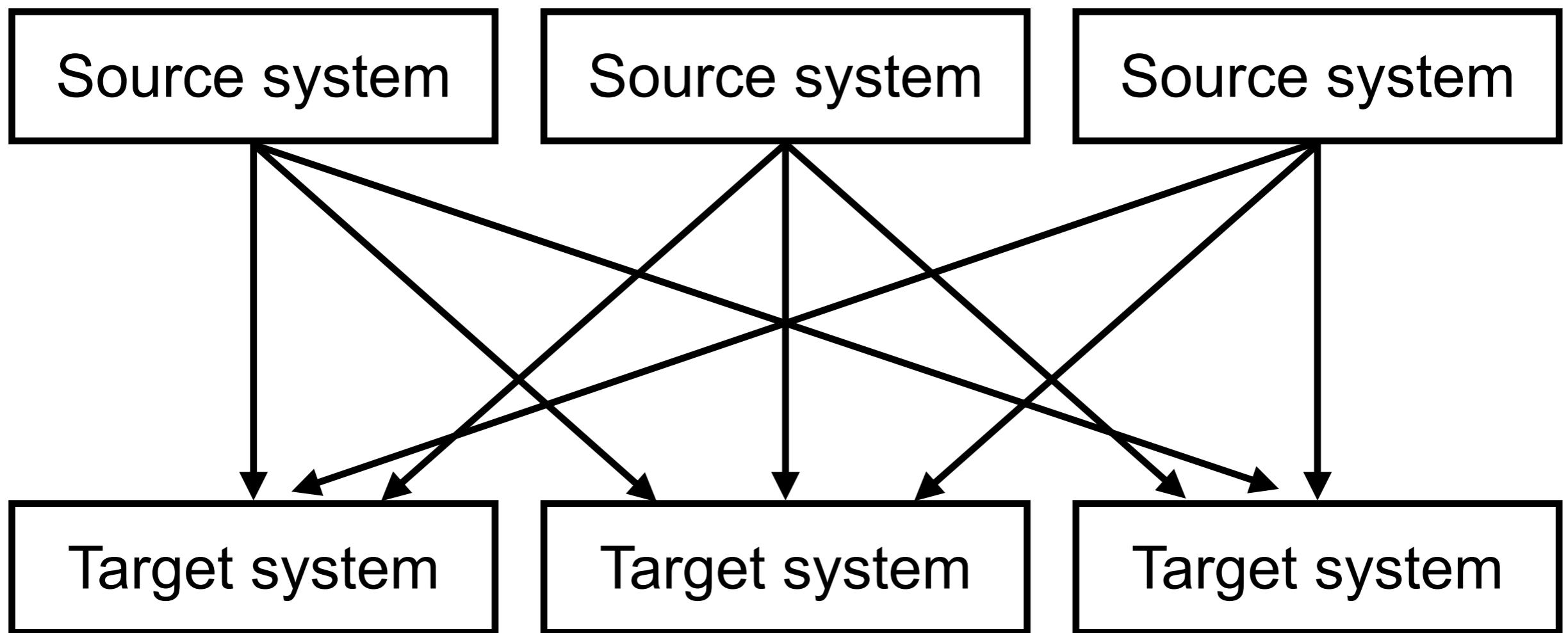
More data formats and schemas

More loads/traffics



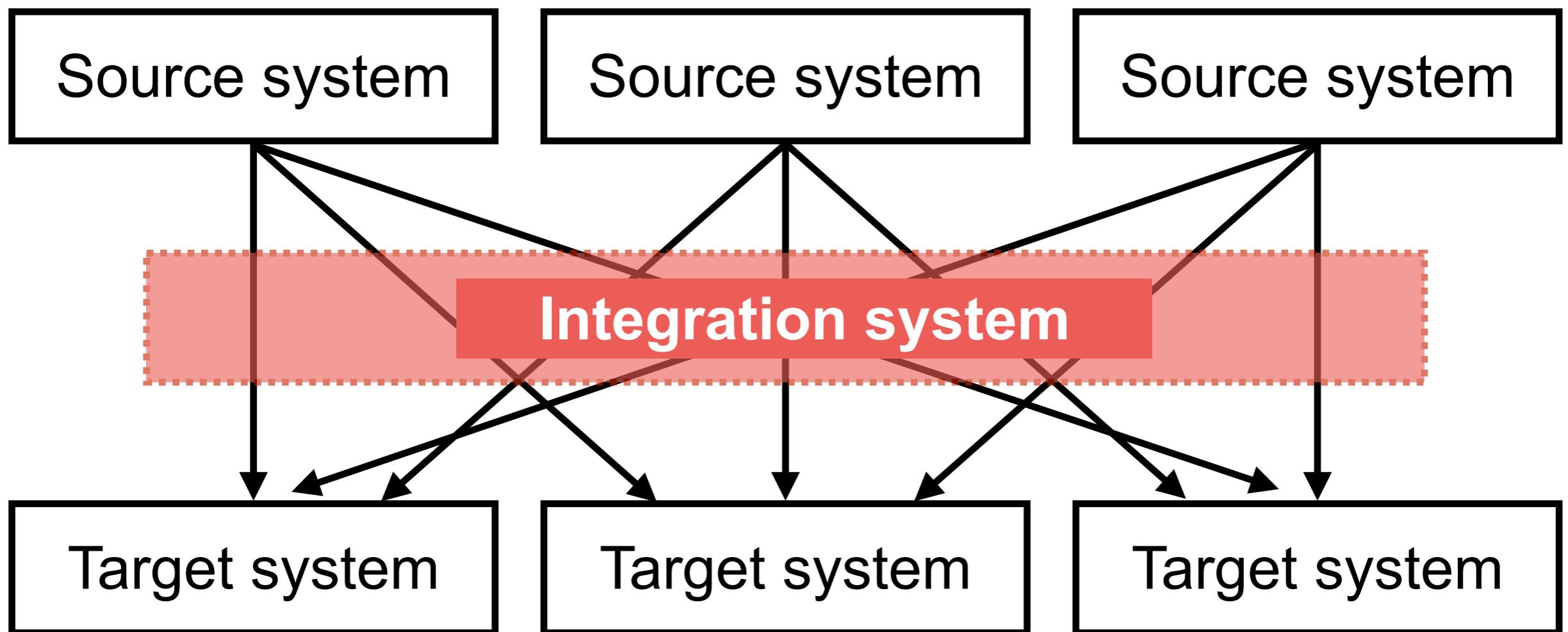
Why

Complex system !!

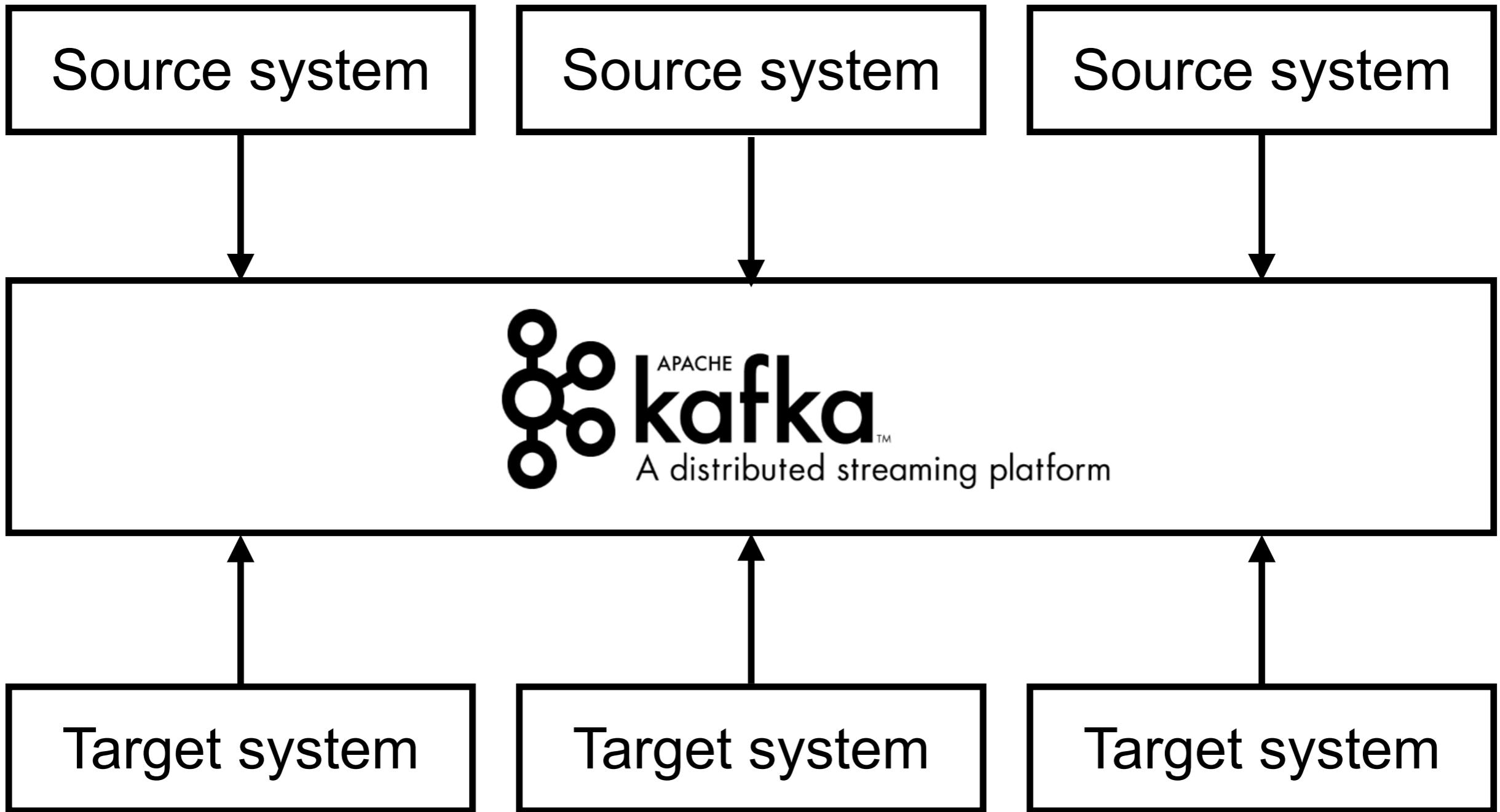


Why

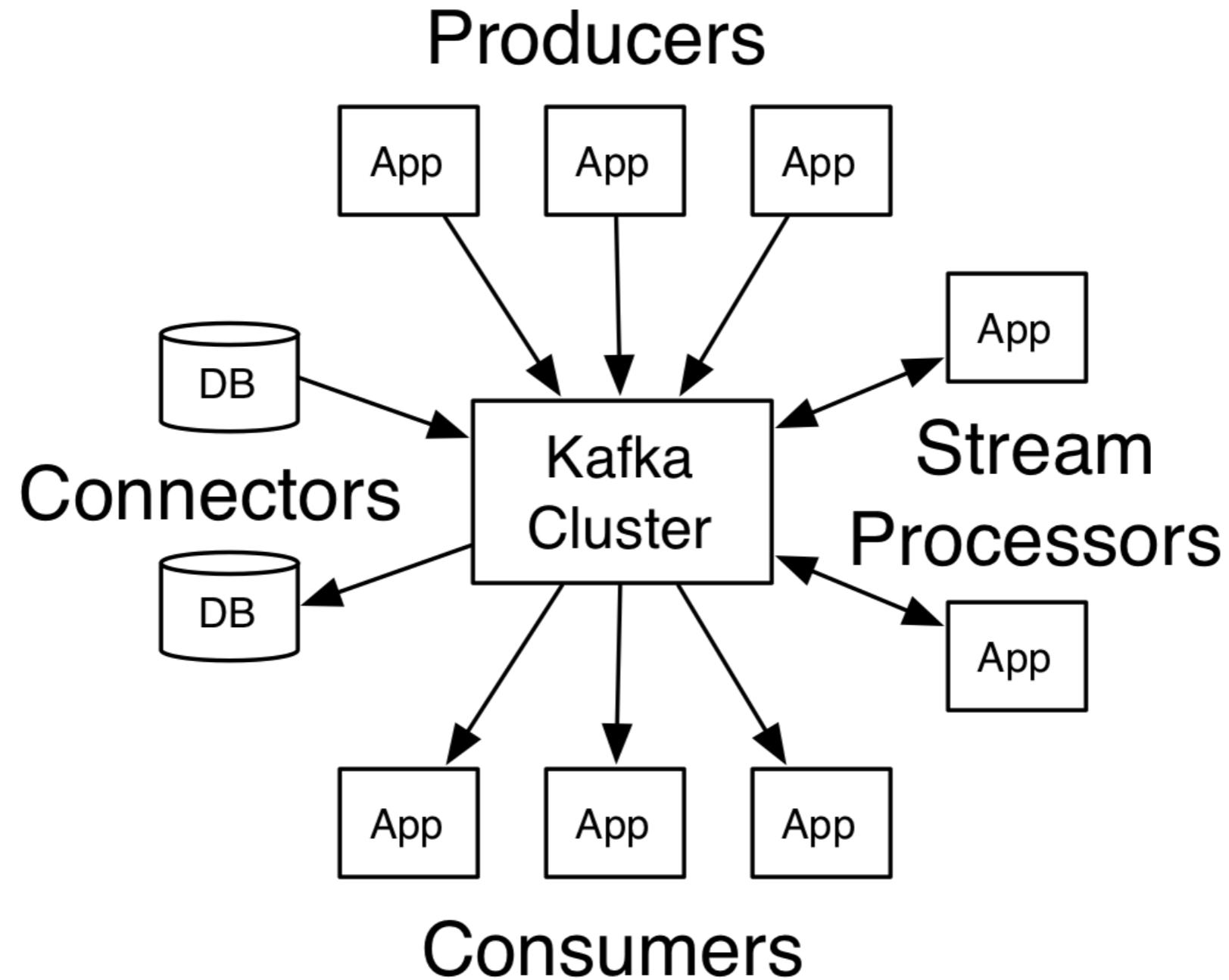
Need integration system



Apache Kafka



Apache Kafka



<https://kafka.apache.org/intro.html>



Why Apache Kafka ?

Created by LinkedIn

Open source project (maintain by Confluent)

Distributed system

Resilient architecture, fault tolerant

Horizontal scalability

High performance/ low latency



Used by



Uber



NETFLIX



<https://cwiki.apache.org/confluence/display/KAFKA/Powered+By>



Kafka with Java

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

Use cases

Messaging system

Activity tracking

Gather metric from different locations

Gather application logs

Stream processing

Decoupling of system dependencies

Integration with others system/technologies



Kafka terminologies

Producers

Consumers

Topics

Partitions

Broker

Consumer groups

Cluster/Replicate

Offset



Learning paths

Part 1

Kafka theory

Starting Kafka

Kafka CLI

Kafka with Java

Part 2

Configuration

Producers

Consumers

Monitoring



Kafka theory



Topics, partitions and offsets

Topics

Stream of data

Similar to a table in database

You can have many topics as you want

A topic is identified by **name**

Topic



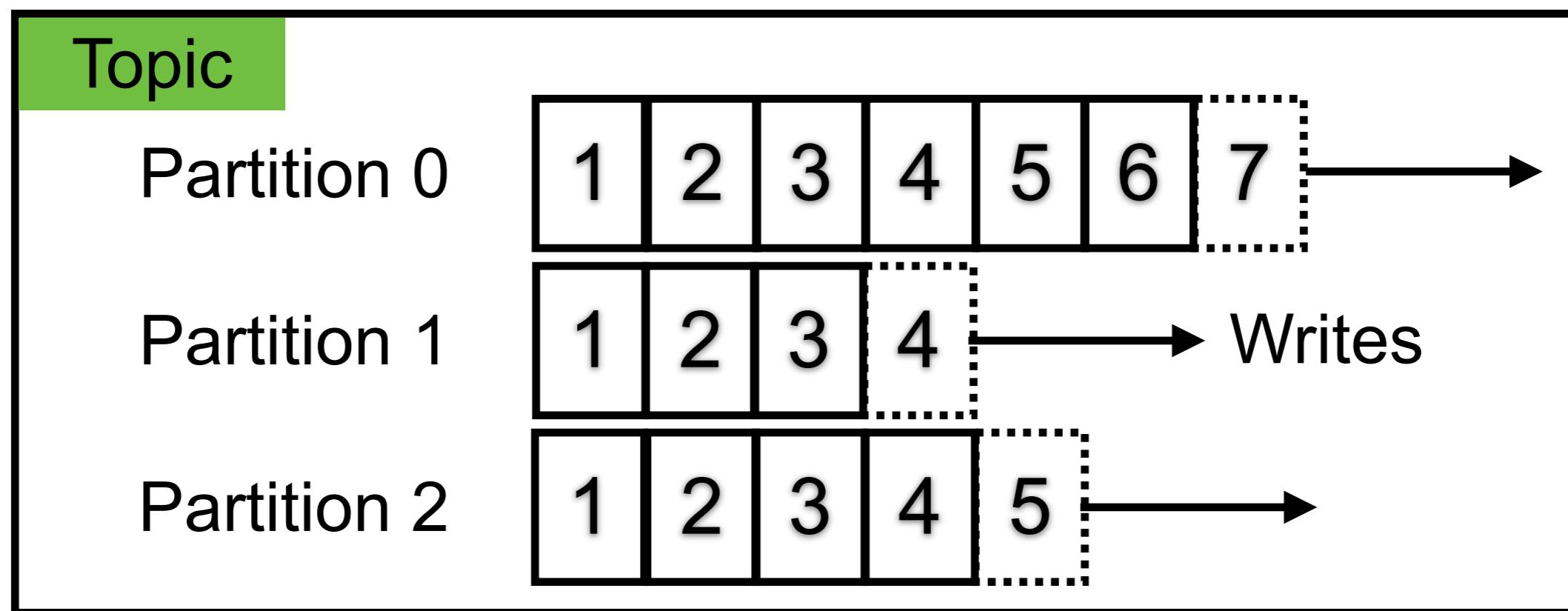
Topics, partitions and offsets

Partitions

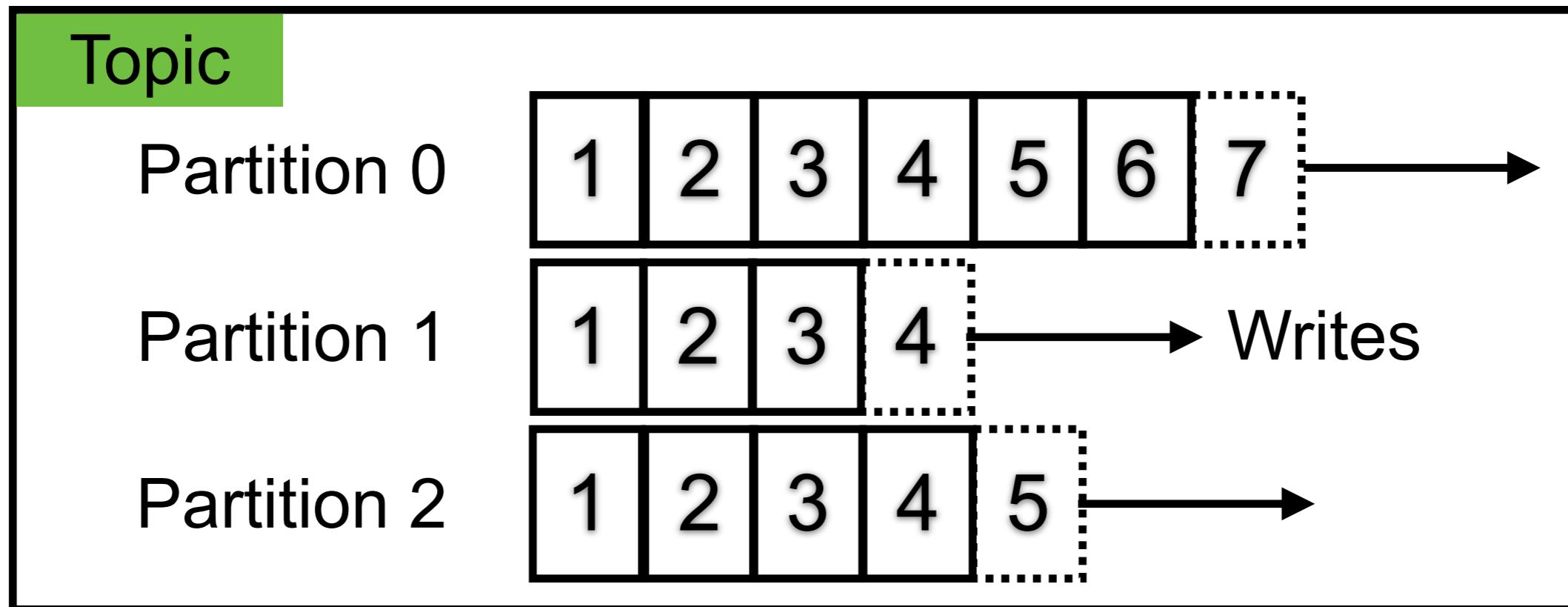
Topics are split in partitions

Each partition is **ordered**

Each message in partition get incremental id (**offset**)



Topics, partitions and offsets



Order is guaranteed only within partition
Data is keep in limited time (**Default = 1 week**)
Data is **immutable** (can't be changed)
Data is assigned **randomly** to a partition



Brokers and topics

Brokers

Kafka cluster is composed of multiple brokers (servers)

Each broker is identified by ID (integer)

Each broker contains certain topic partitions

After connecting any broker (bootstrap broker), you will connected to the entire cluster

Broker 1

Broker 2

Broker 3



Brokers and topics

Brokers

Good number to start id 3 brokers

Broker 1

Broker 2

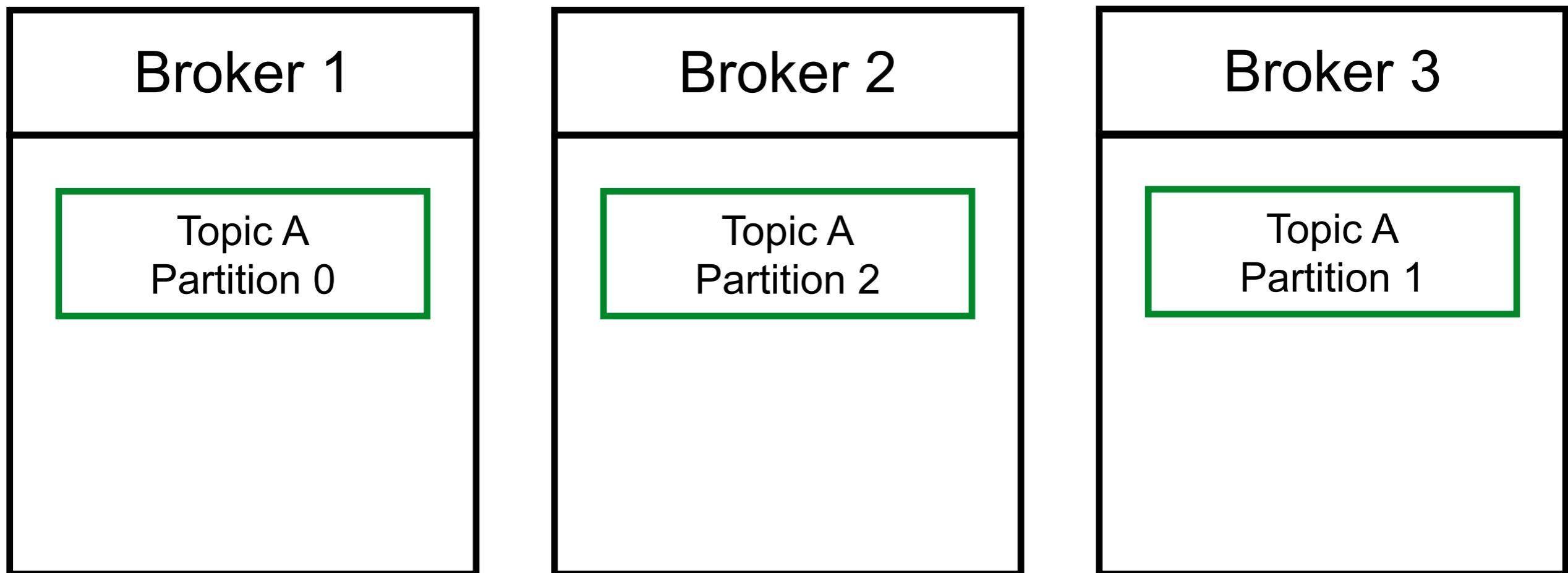
Broker 3



Brokers and topics

Example

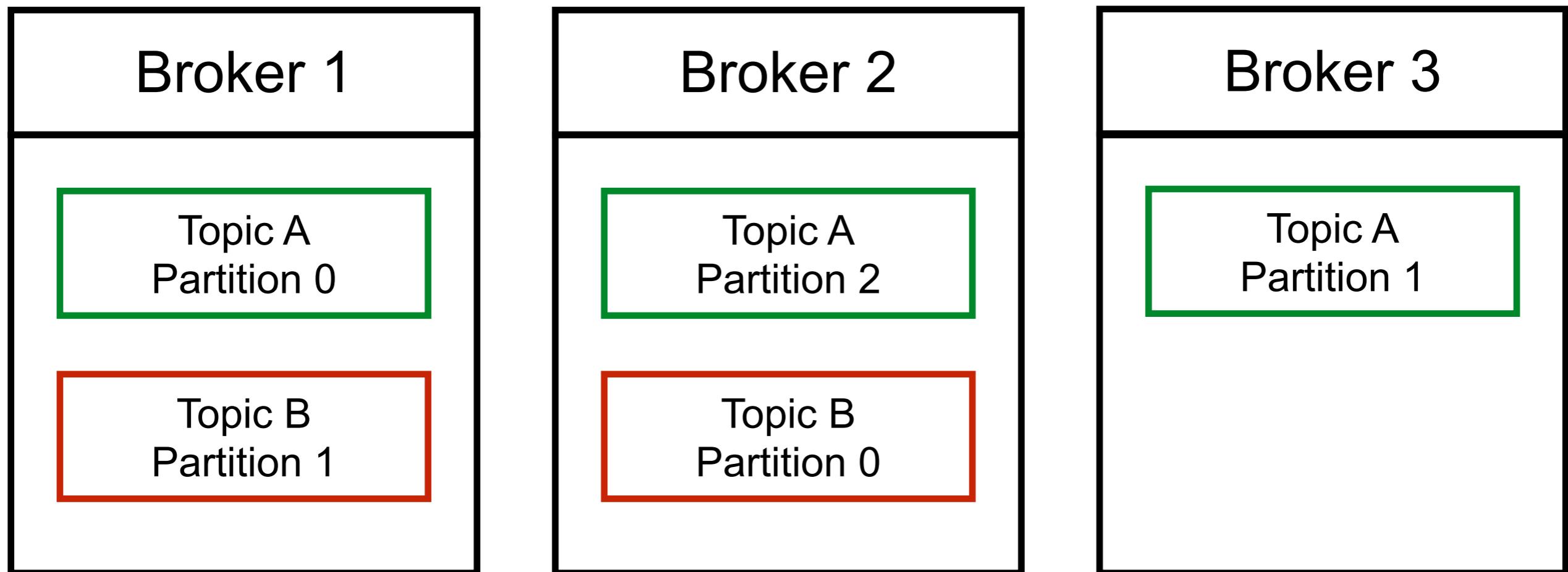
Topic A with 3 partitions



Brokers and topics

Example

Topic B with 2 partitions



Topic with replication factor

Topic should have a replica factor > 1 (2-3)

replica factor $<$ no. of brokers

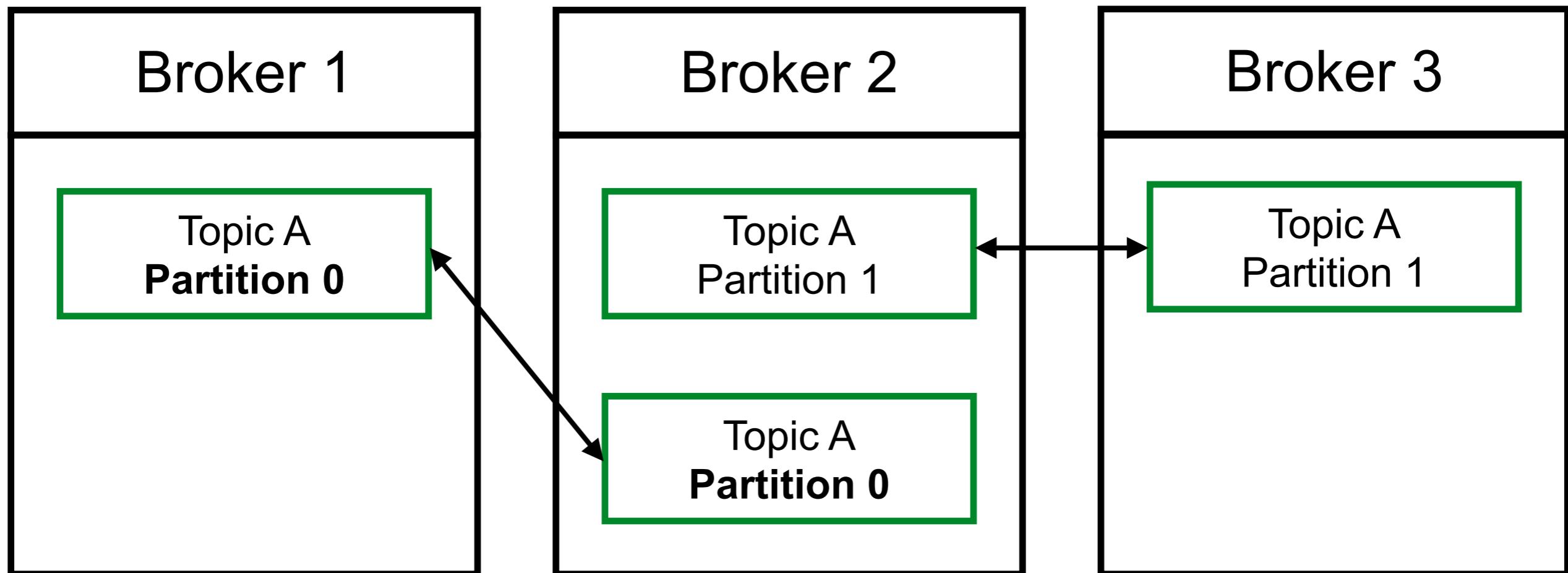
This way if a broker down, another broker can serve the data



Topic with replication factor

Example

Topic A with 2 partitions and replication factor = 2



Leader for a partition

At any time only one Broker can be leader for partition

Only leader partition can receive and serve data for a partition

Other brokers will synchronize the data

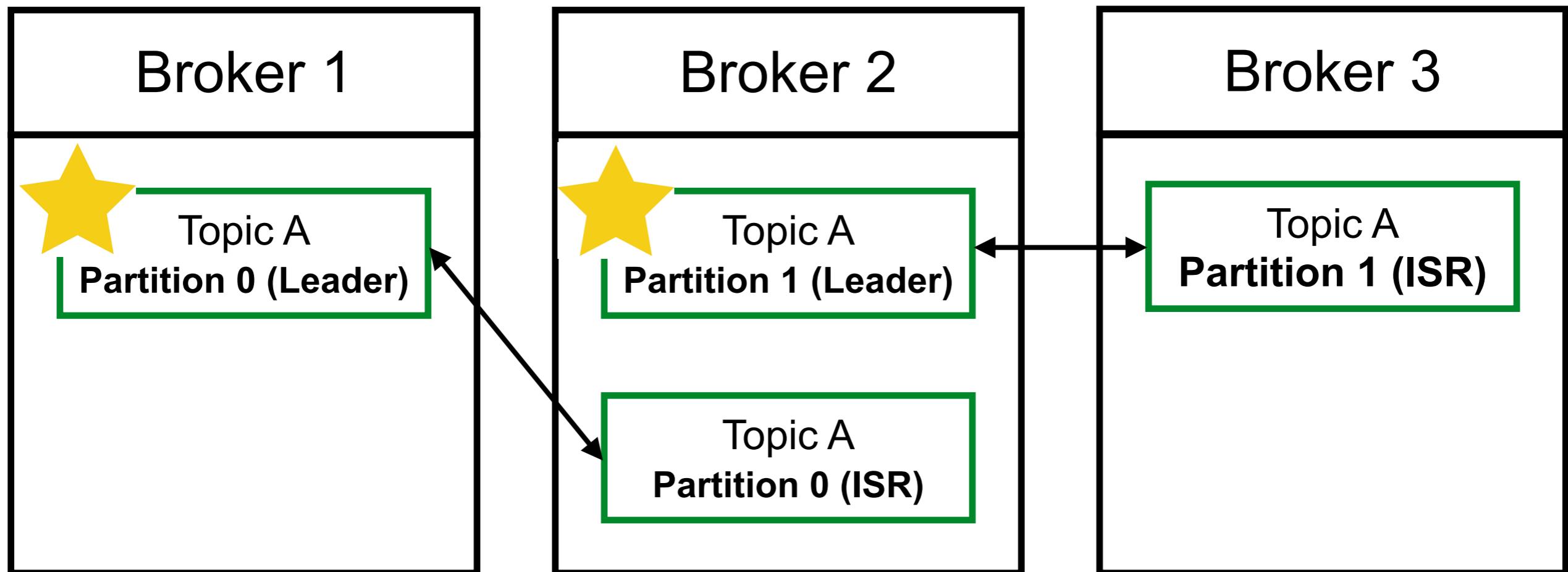
Other partition called **in-sync replica (ISR)**



Leader for a partition

Example

Topic A with 2 partitions and replication factor = 2



Producers

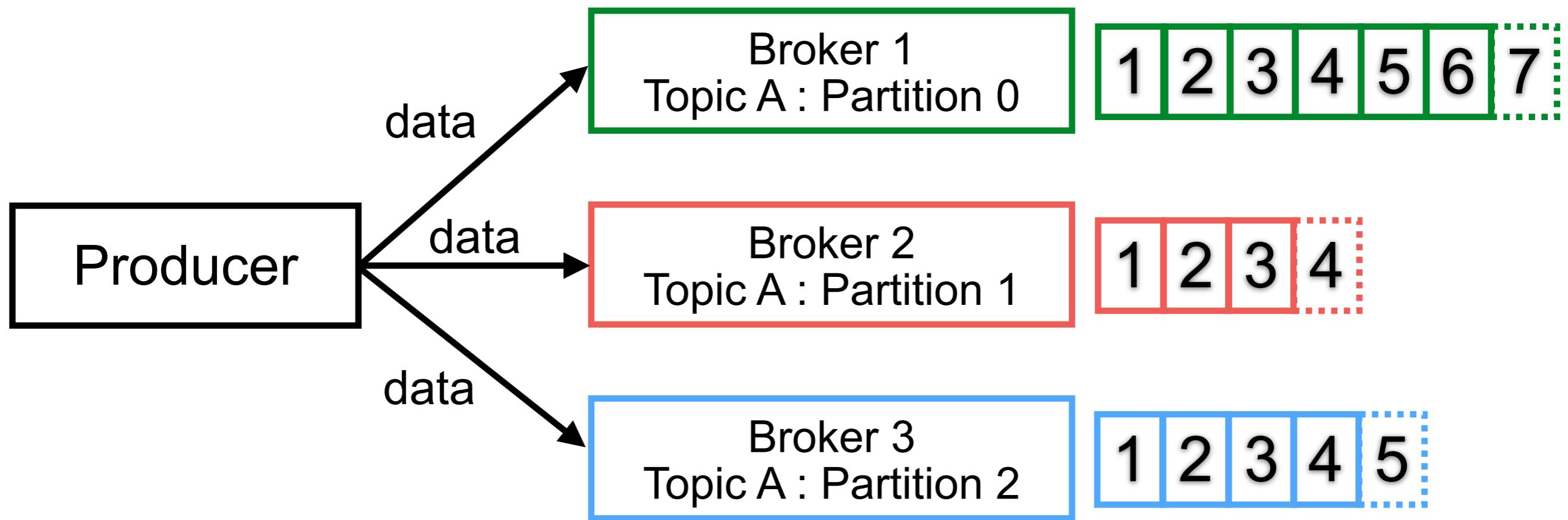
Producers write data to topics

Producers automatically know to broker and partition to write

When broker failures, producers will automatically recover



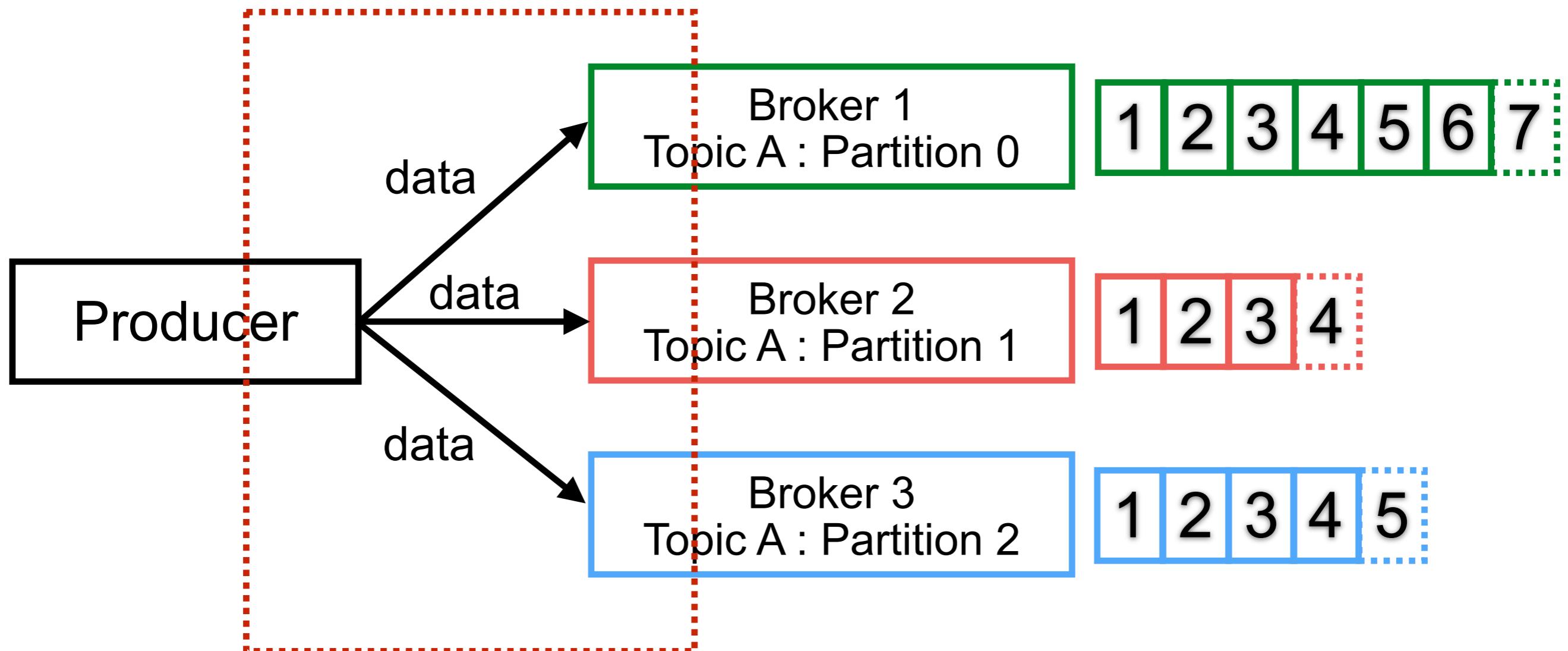
Producers



*** *The load is balanced to many brokers (no. of partitions)* ***



Producers issue to write data !!



Producers with acknowledgment

acks=0

Producer not wait for acknowledgment

Possible data loss

acks=1

Producer will wait for **leader** acknowledgment

Limited data loss

acks=all

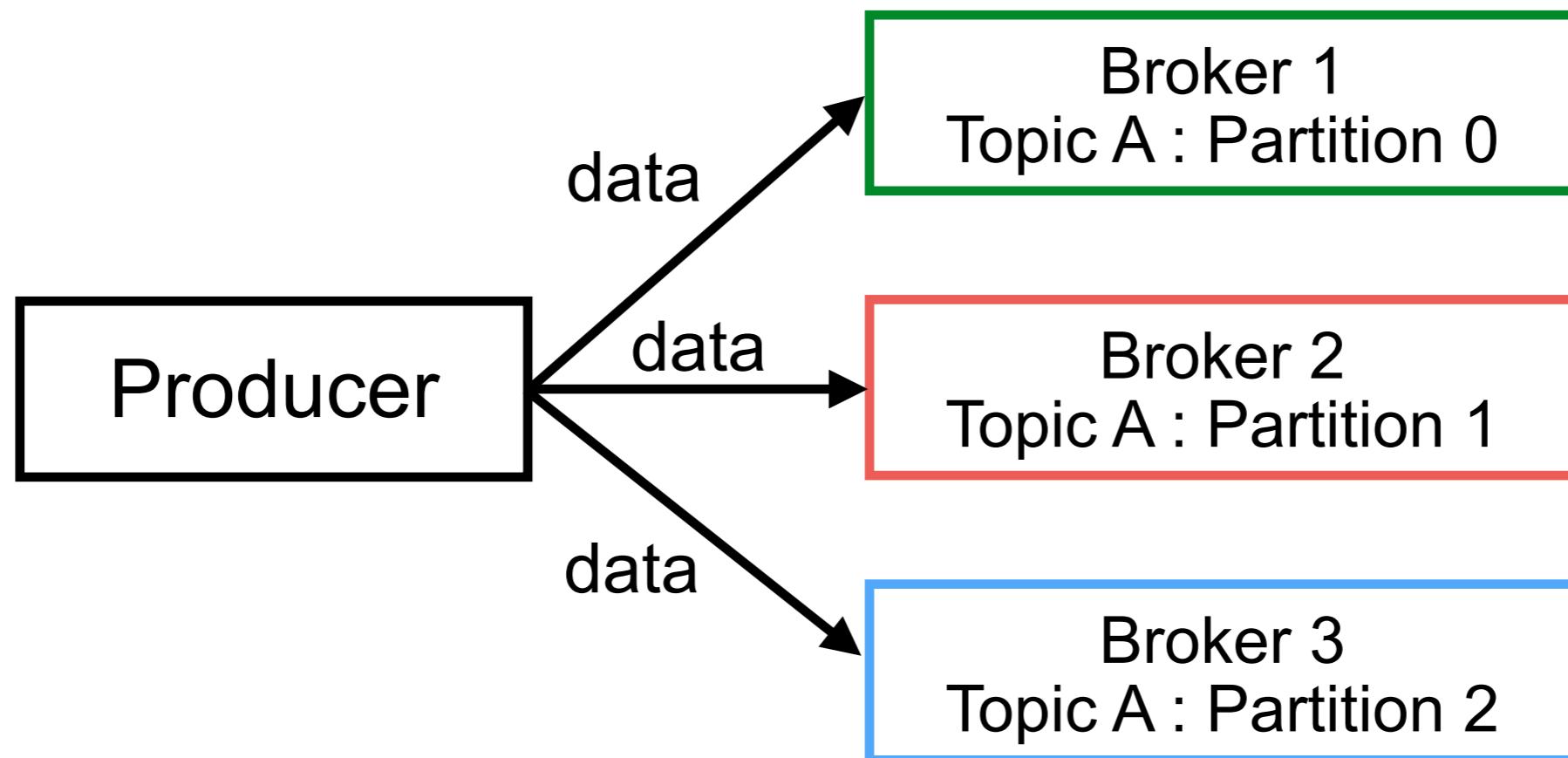
Producer will wait for **leader + ISR** acknowledgment

No data loss



Producers with message keys

Producers can choose to sent a **key** with data
Key = null, data is sent **round-robin**



Producers with message keys

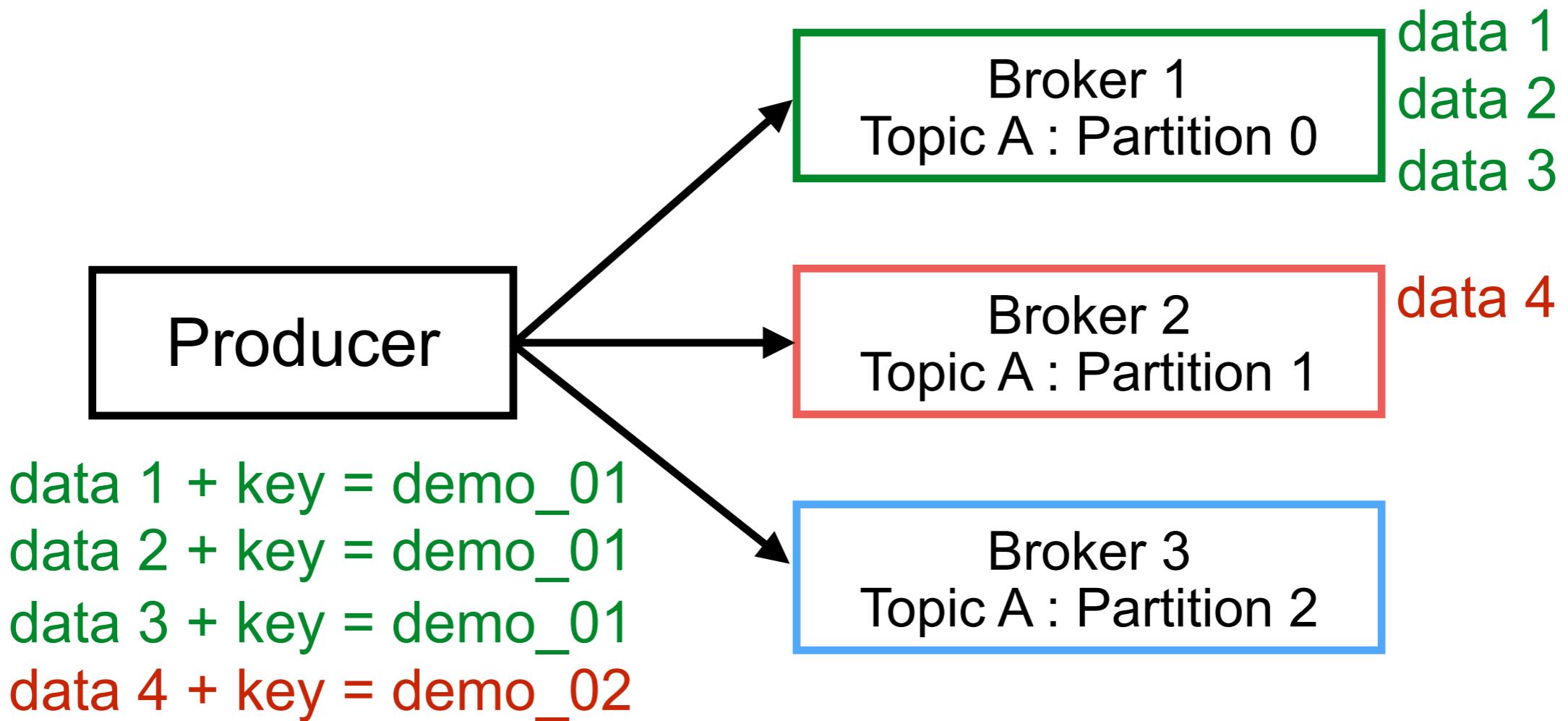
```
public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[]  
... if (keyBytes == null) {  
...     return stickyPartitionCache.partition(topic, cluster);  
... }  
... List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);  
... int numPartitions = partitions.size();  
... // hash the keyBytes to choose a partition  
... return Utils.toPositive(Utils.murmur2(keyBytes)) % numPartitions;  
}
```

<https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/producer/internals/DefaultPartitioner.java>



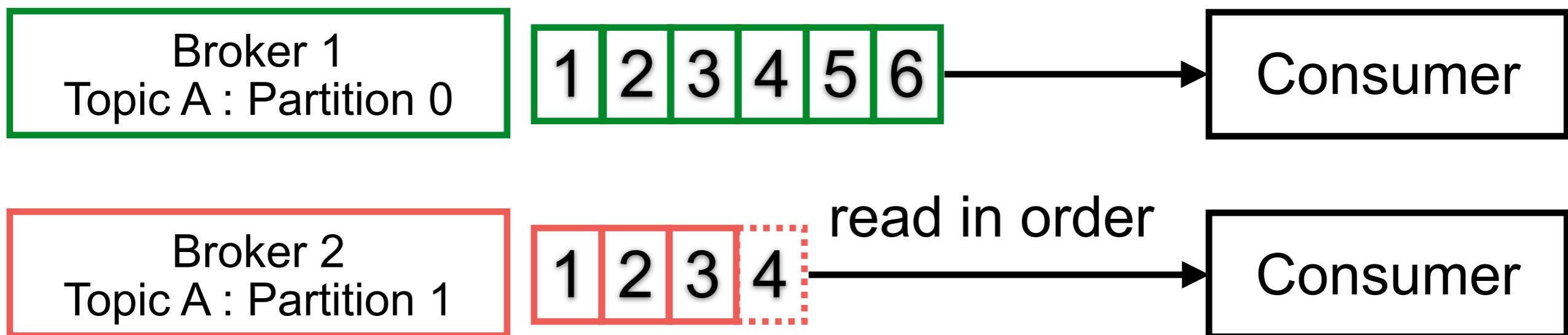
Why use message keys ?

You need message ordering



Consumers

Consumers read data from topic
Consumers know which broker to read from
Data will read in order **within each partition**
When broker **failures**, consumer know how to recover



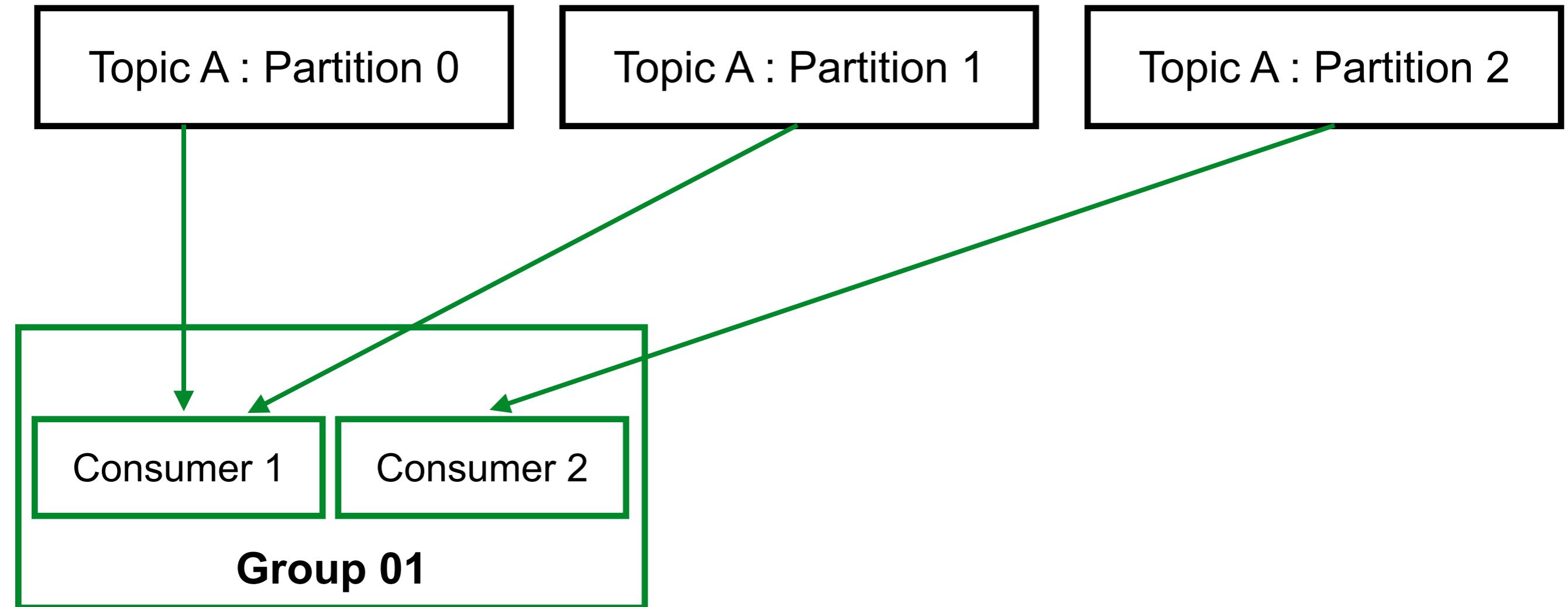
Consumer groups

Consumers read data in consumer groups

Each consumer in a group read from exclusive partitions



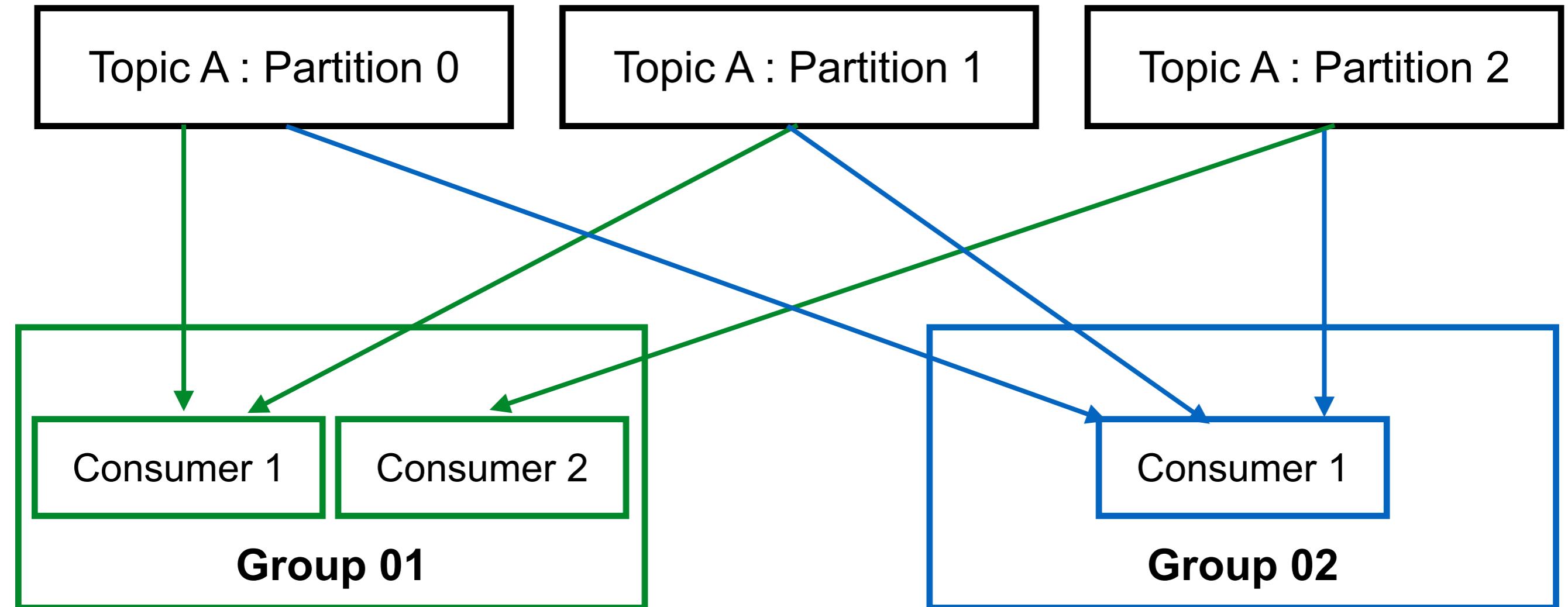
Consumer groups



Consumer will automatically use a GroupCoordinator and ConsumerCoordinator to assign a consumer to a partition.



Consumer groups



Partition assignment strategies

Range (default)

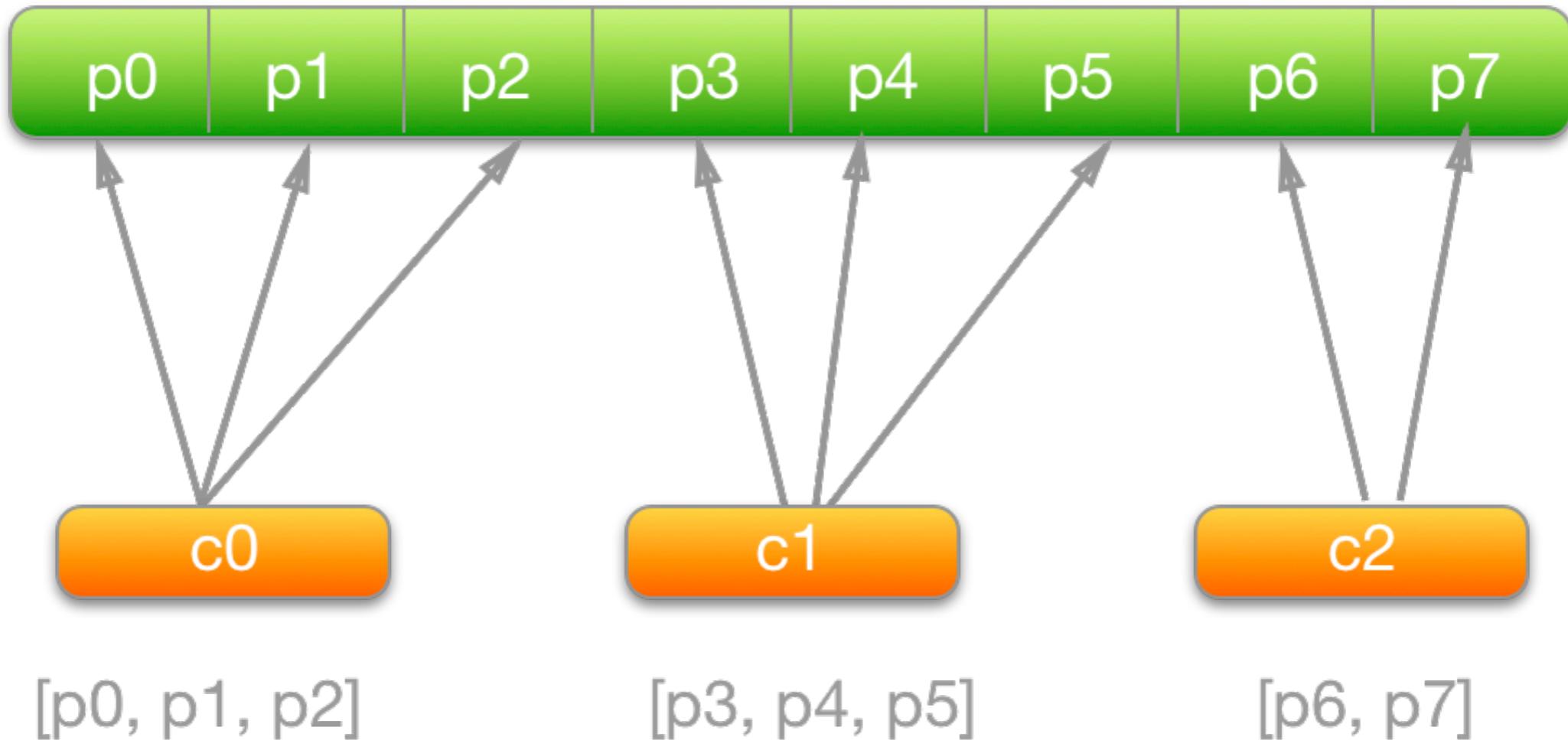
Round Robin

Sticky

Customize (AbstractPartitionAssignor)



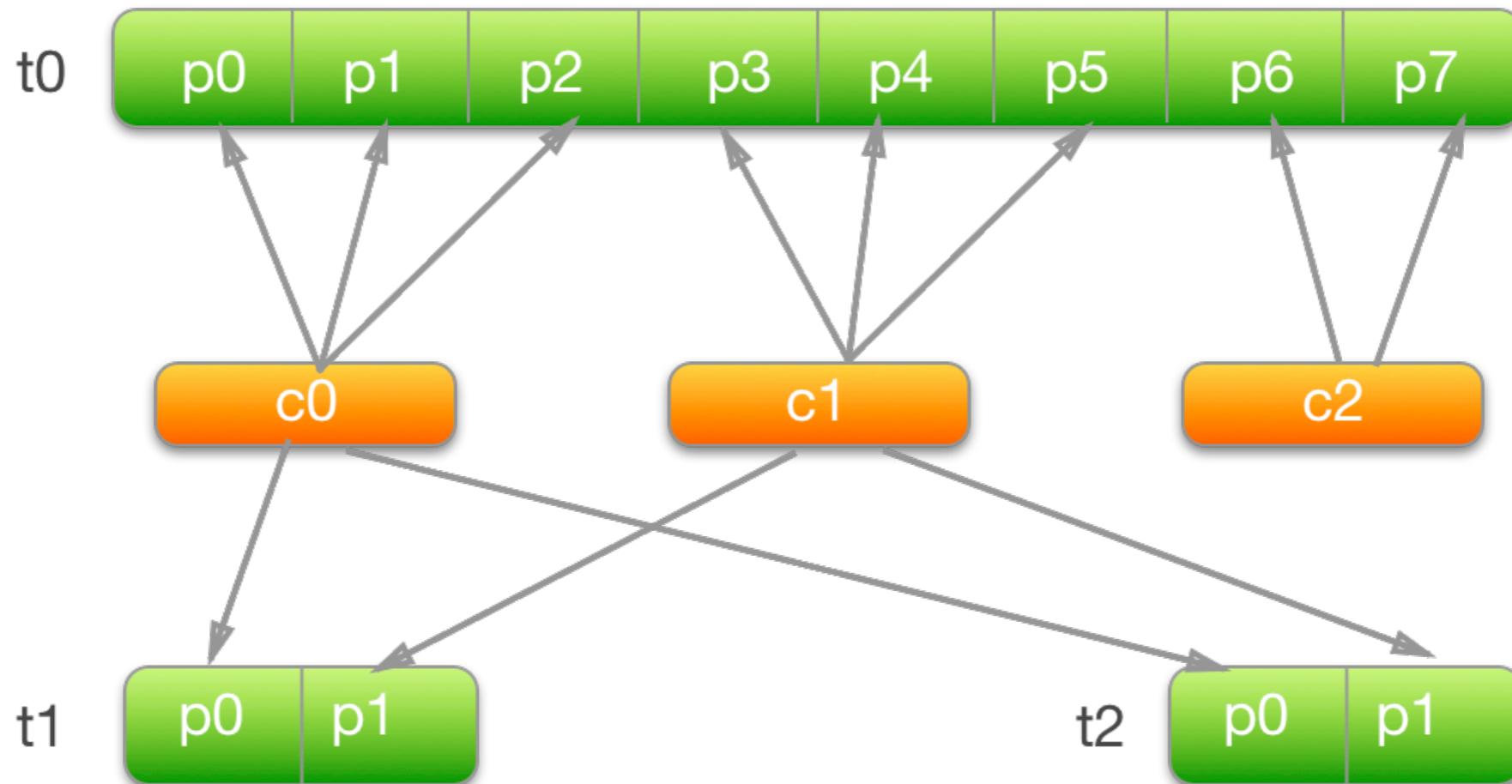
Range assignor



org.apache.kafka.clients.consumer.RangeAssignor



Range assignor



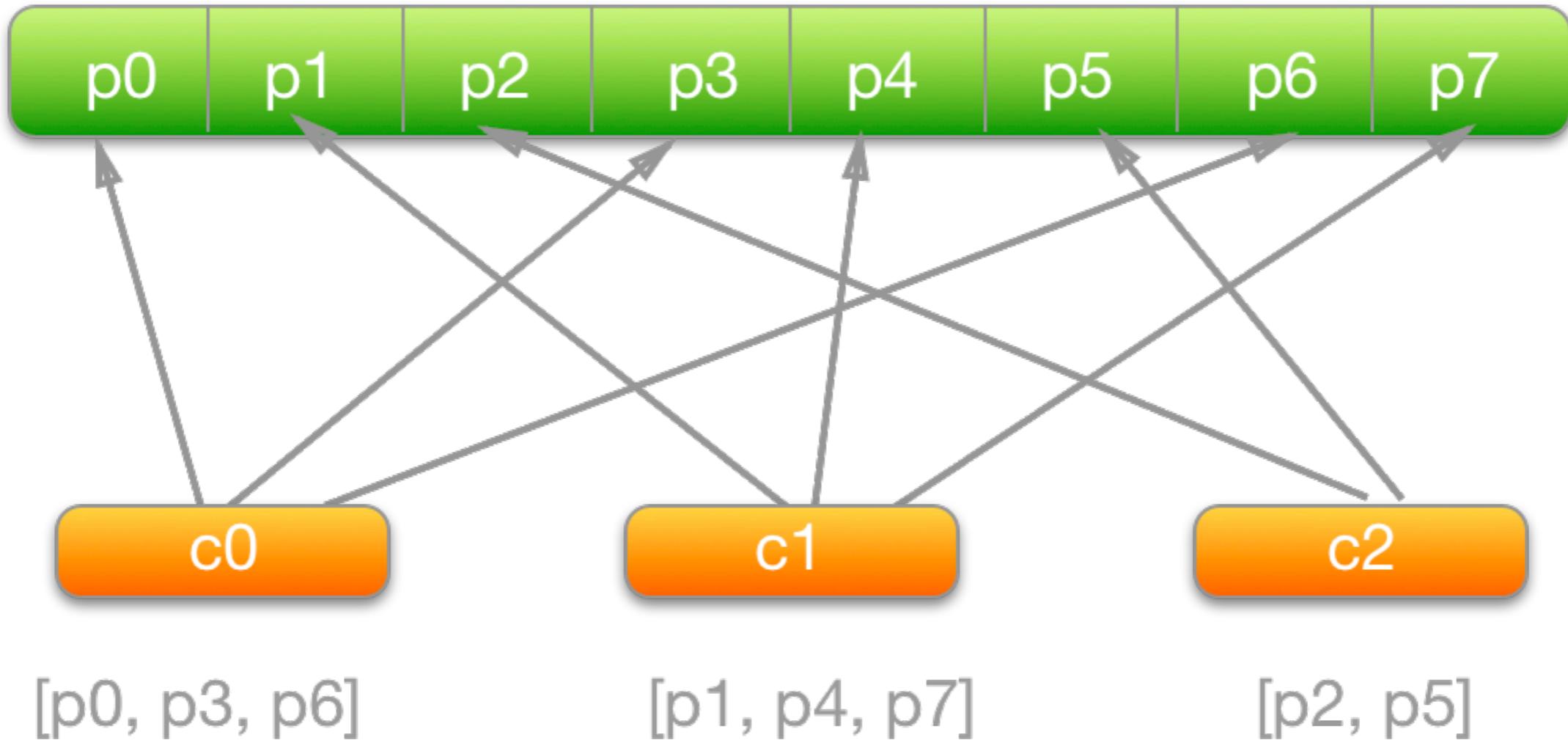
c0 -> [t0p0, t0p1, t0p2, t1p0, t2p0]

c1 -> [t0p3, t0p4, t0p5, t1p1, t2p1]

c2 -> [t0p6, t0p7]



Round Robin assignor

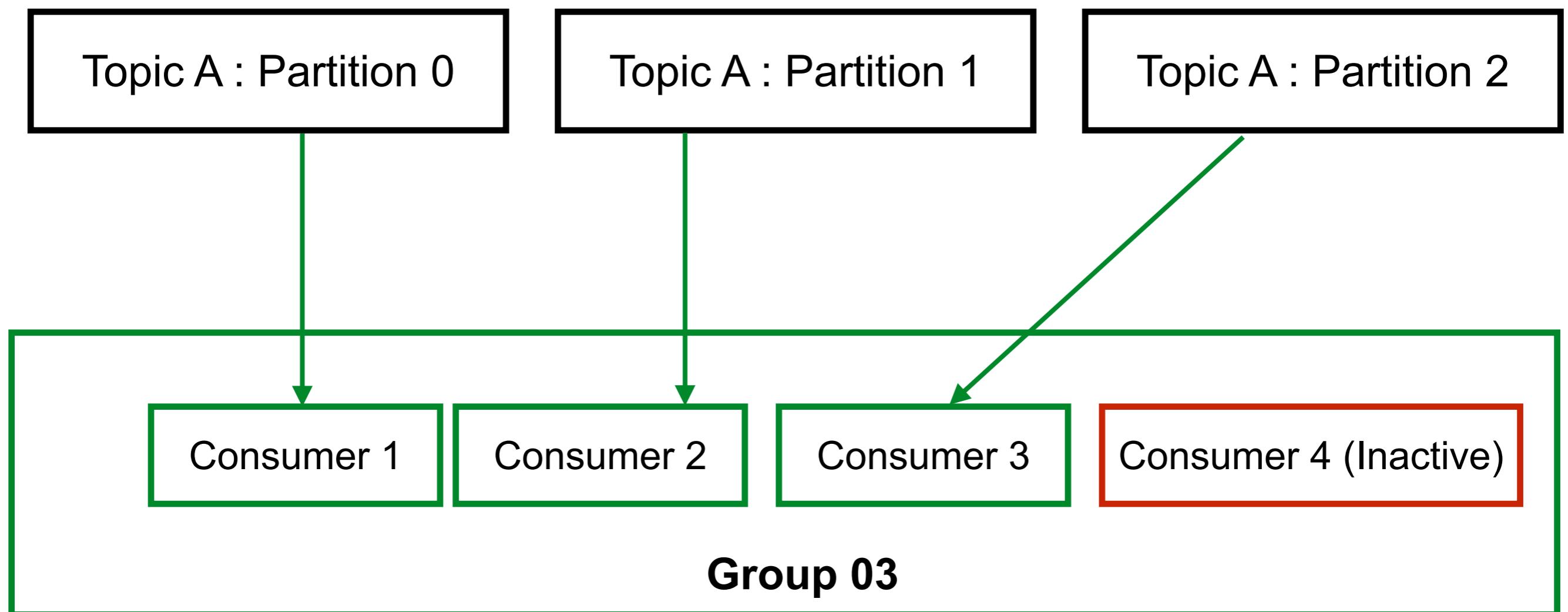


org.apache.kafka.clients.consumer.RoundRobinAssignor

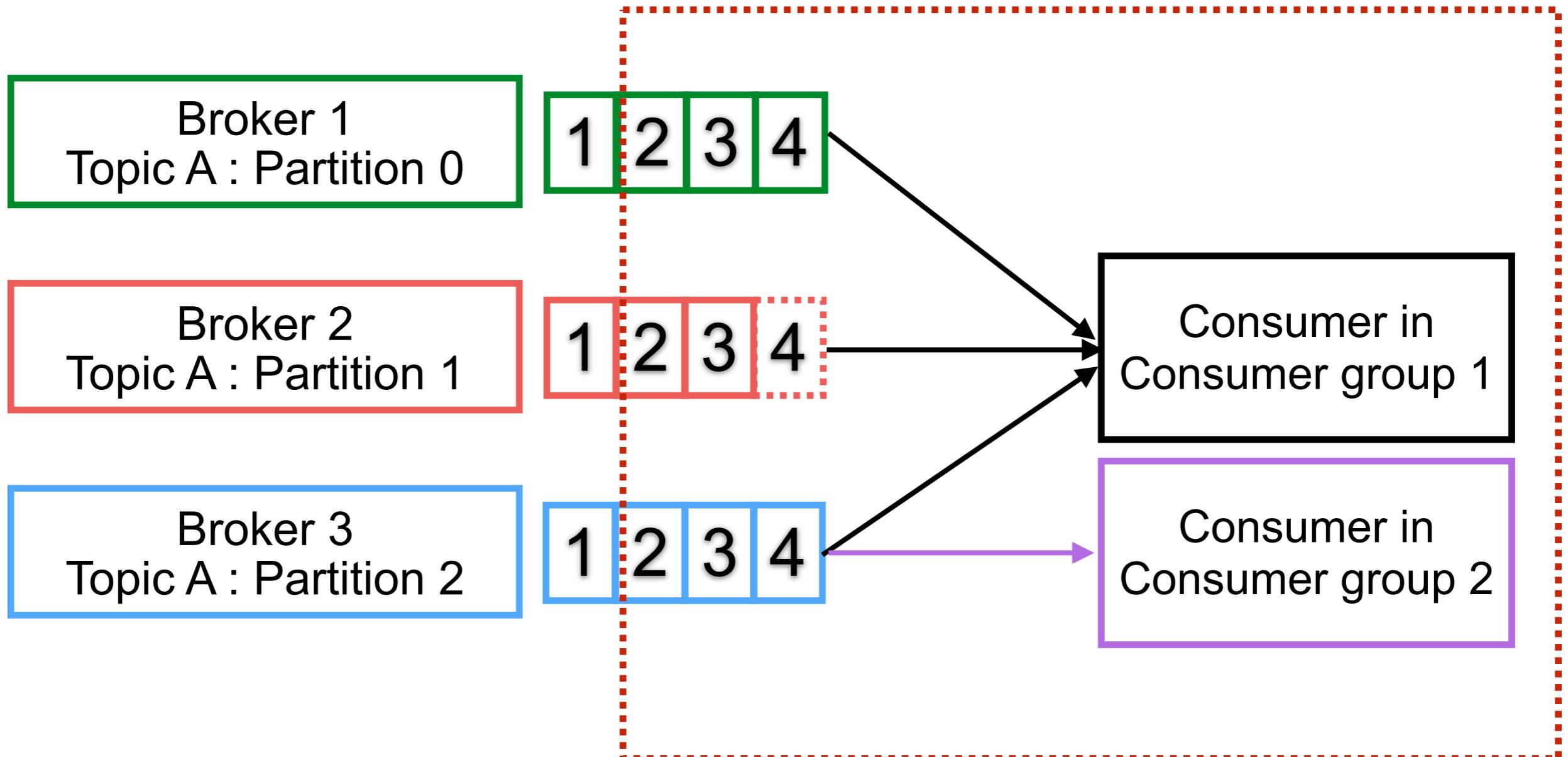


Consumers > partitions ?

Some consumers will **inactive**



Consumers offsets



Consumers offsets

Kafka store the offset at which a consumer group has been reading

The offsets committed live in topic named
“**__consumer_offsets**”

When consumer in a group has processed data received from Kafka,
it should be **committing the offsets**



When to commit the offset ?



Delivery semantics for consumer

At most once

At lease once (preferred)

Exactly once



1. At most once

Offsets are committed as soon as the message is received

If processing go wrong, the message will be loss!!

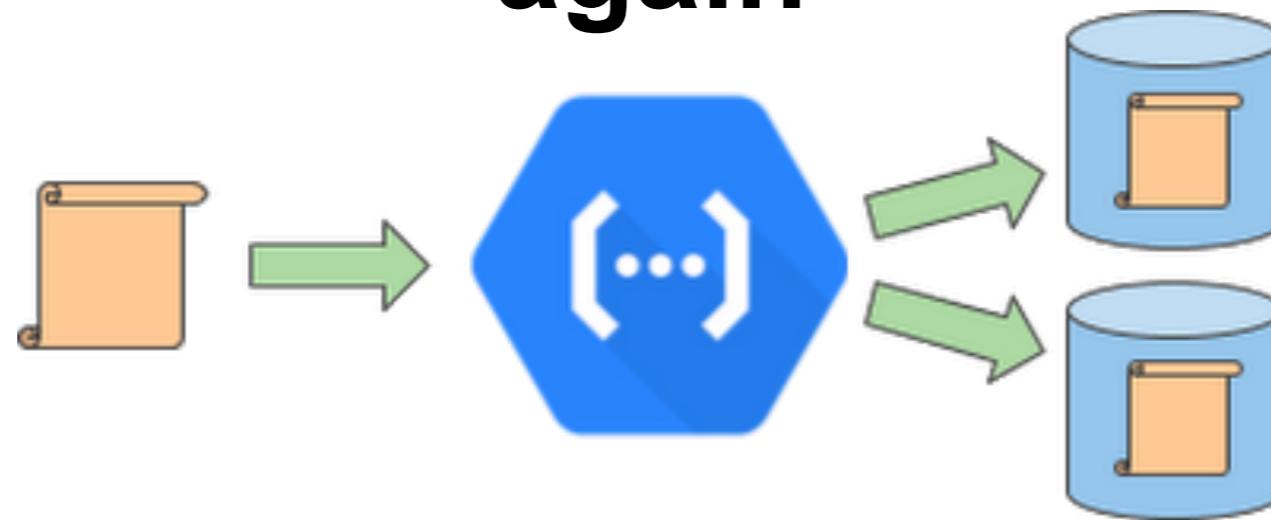


2. At lease once

Offsets are committed after the message is processed

Messages are never lost but may be **redelivered**

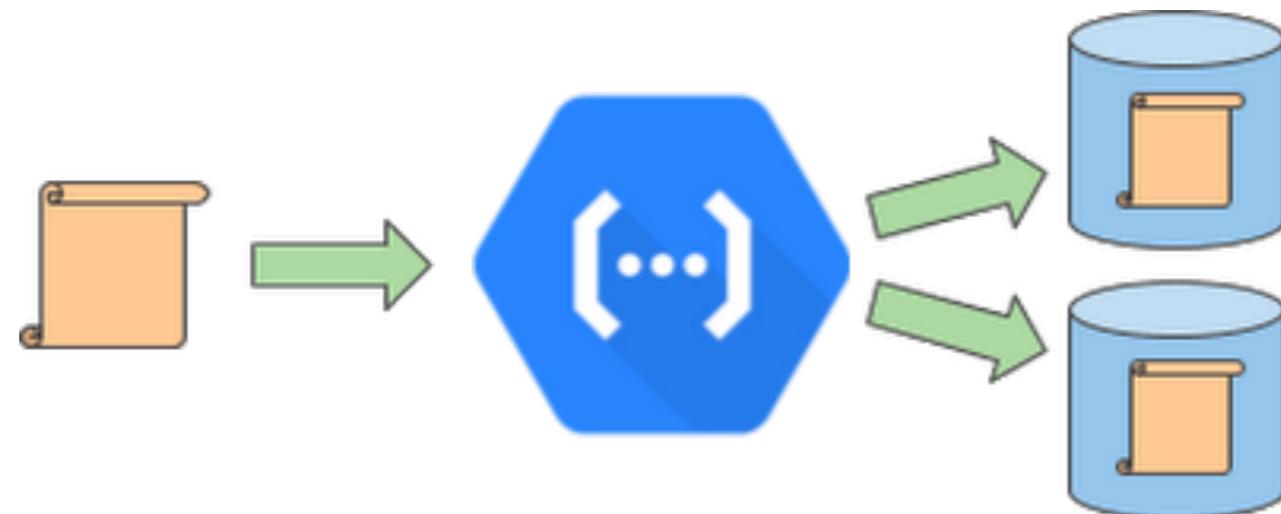
If processing go wrong, the message will be **read again**



2. At lease once

Make sure your processing is **idempotent**

Processing again the message not impact to your system !!



3. Exactly once

Each message is delivered once and only once
Can be achieved for Kafka (Workflow, Stream API)



Kafka broker discovery

Every Kafka broker is called “bootstrap server”
You only need to connect to one broker, and you
will connected to the entire cluster

Broker 1
(bootstrap)

Broker 2
(bootstrap)

Kafka cluster

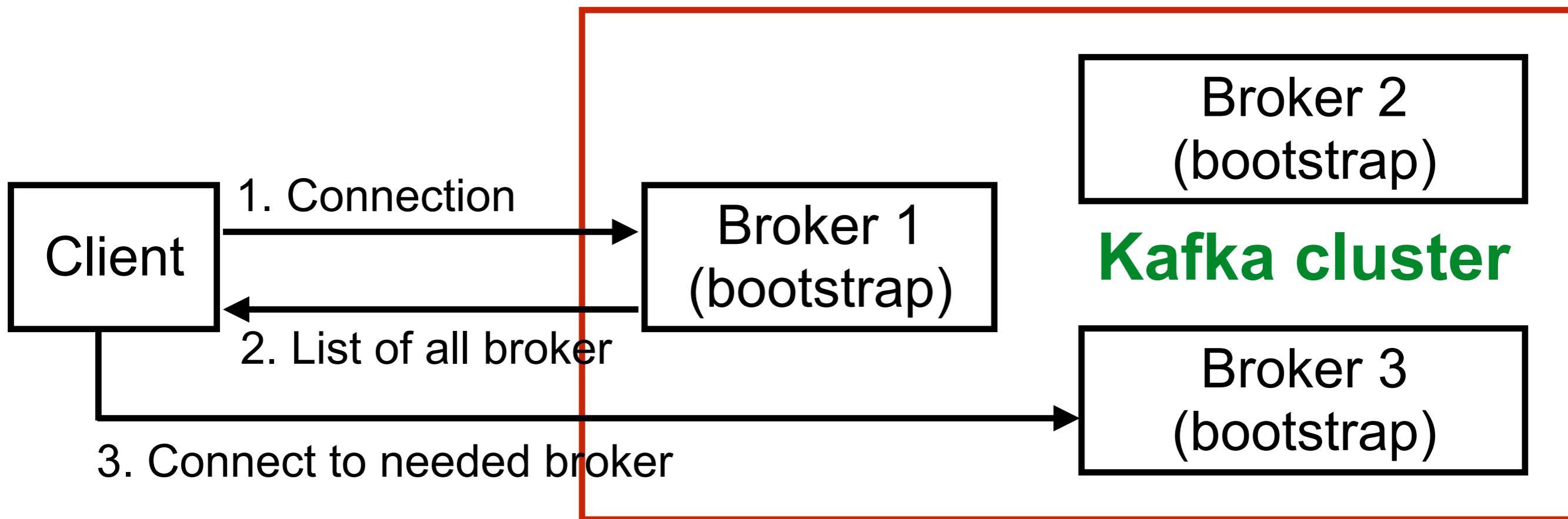
Broker 3
(bootstrap)

Broker 4
(bootstrap)



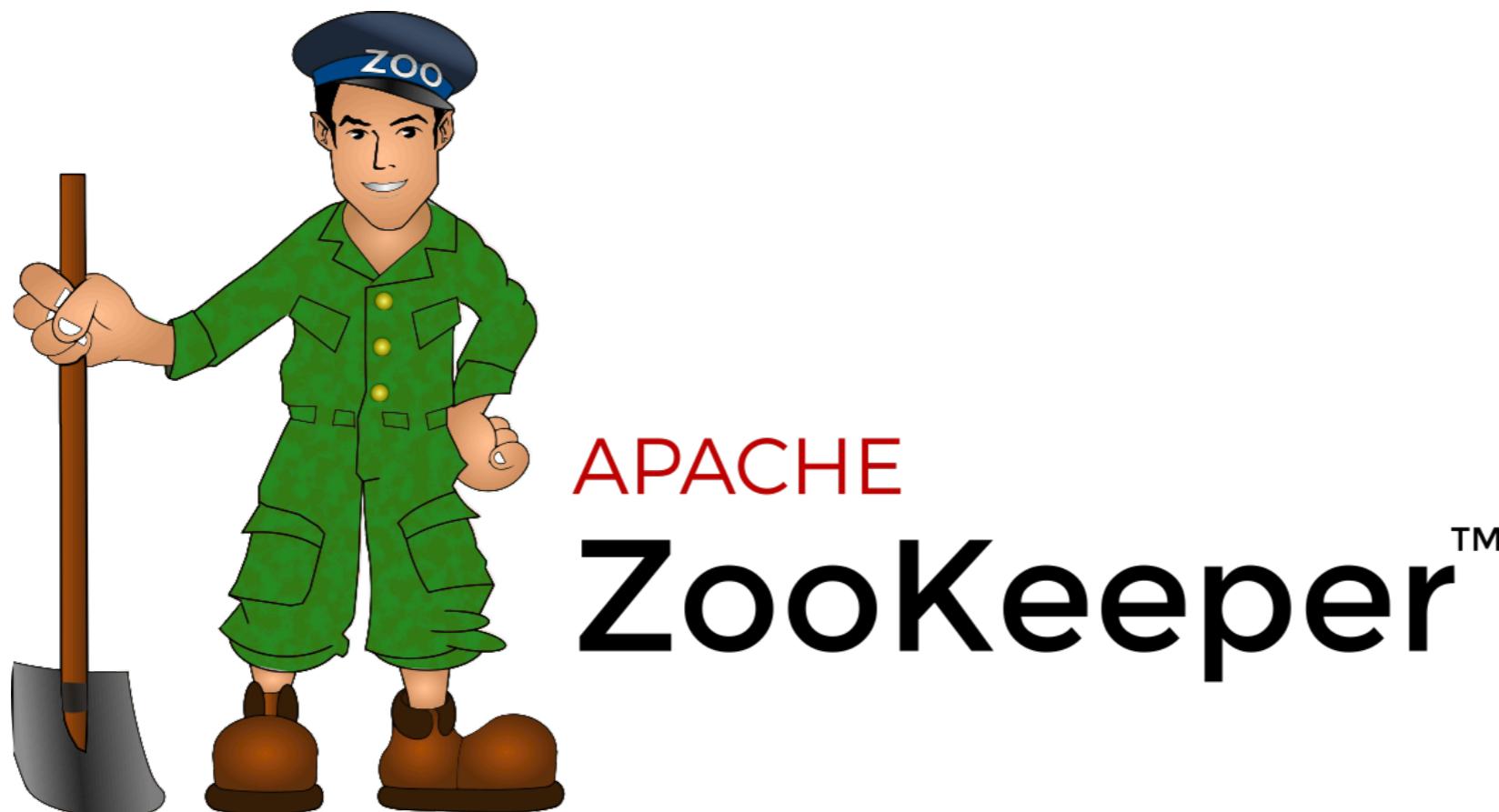
Kafka broker discovery

Each broker knows about all brokers, topics and partitions (metadata)



Apache Zookeeper

Kafka can not work without Zookeeper !!



<https://zookeeper.apache.org/>



Apache Zookeeper

Zookeeper **manages** brokers

Zookeeper help in performing **leading election** for partitions

Zookeeper sends **notifications** to Kafka in case of changes (new topic, broker die)

Zookeeper by design operates with a odd number of servers (3, 5, 7)

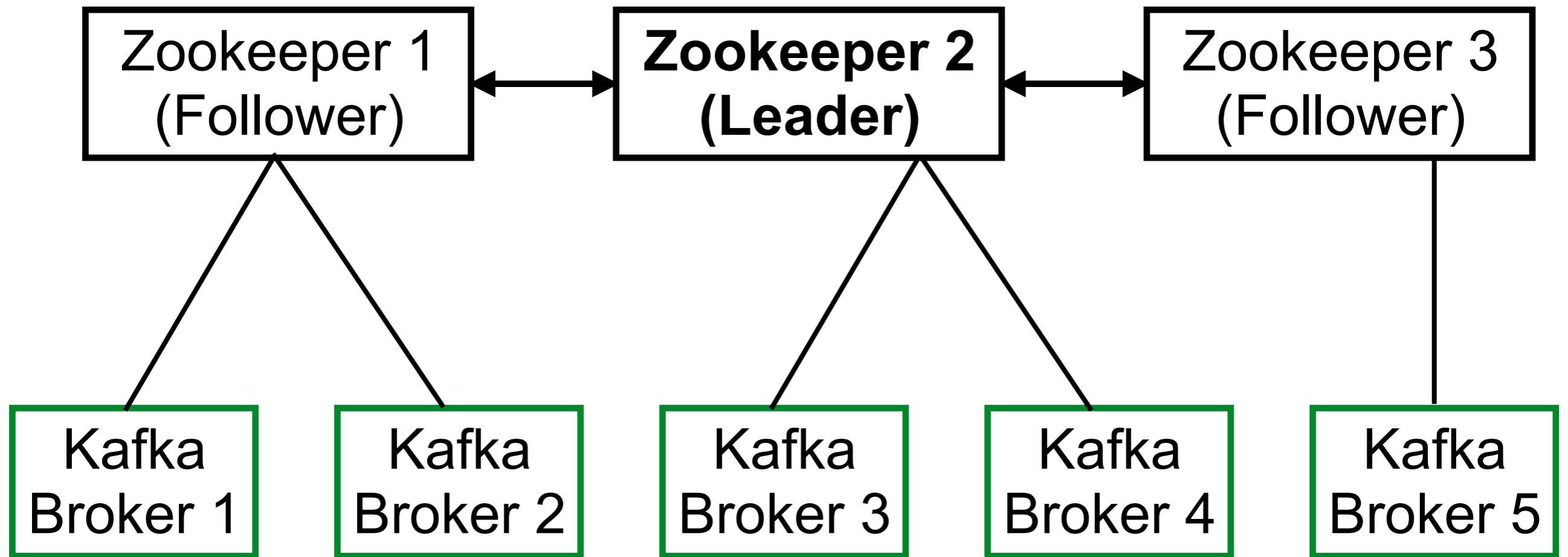


Zookeeper architecture

Leader for write
Follows for read



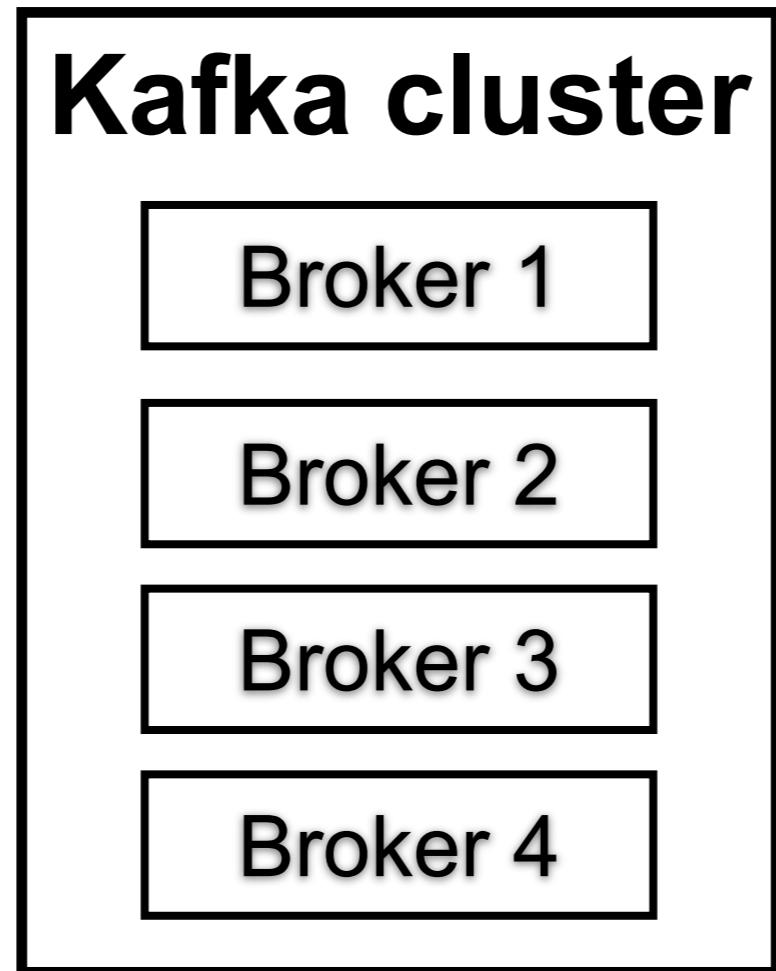
Zookeeper architecture



Summary of Kafka



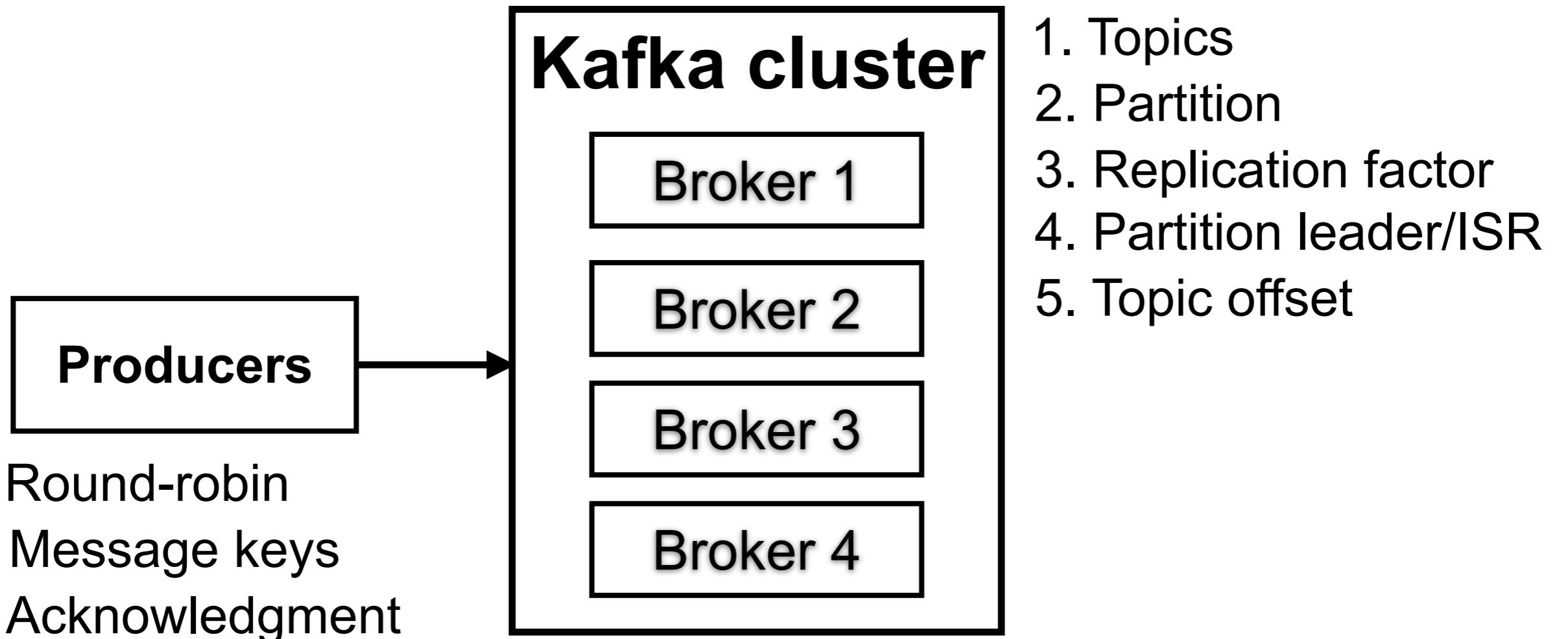
Kafka concepts



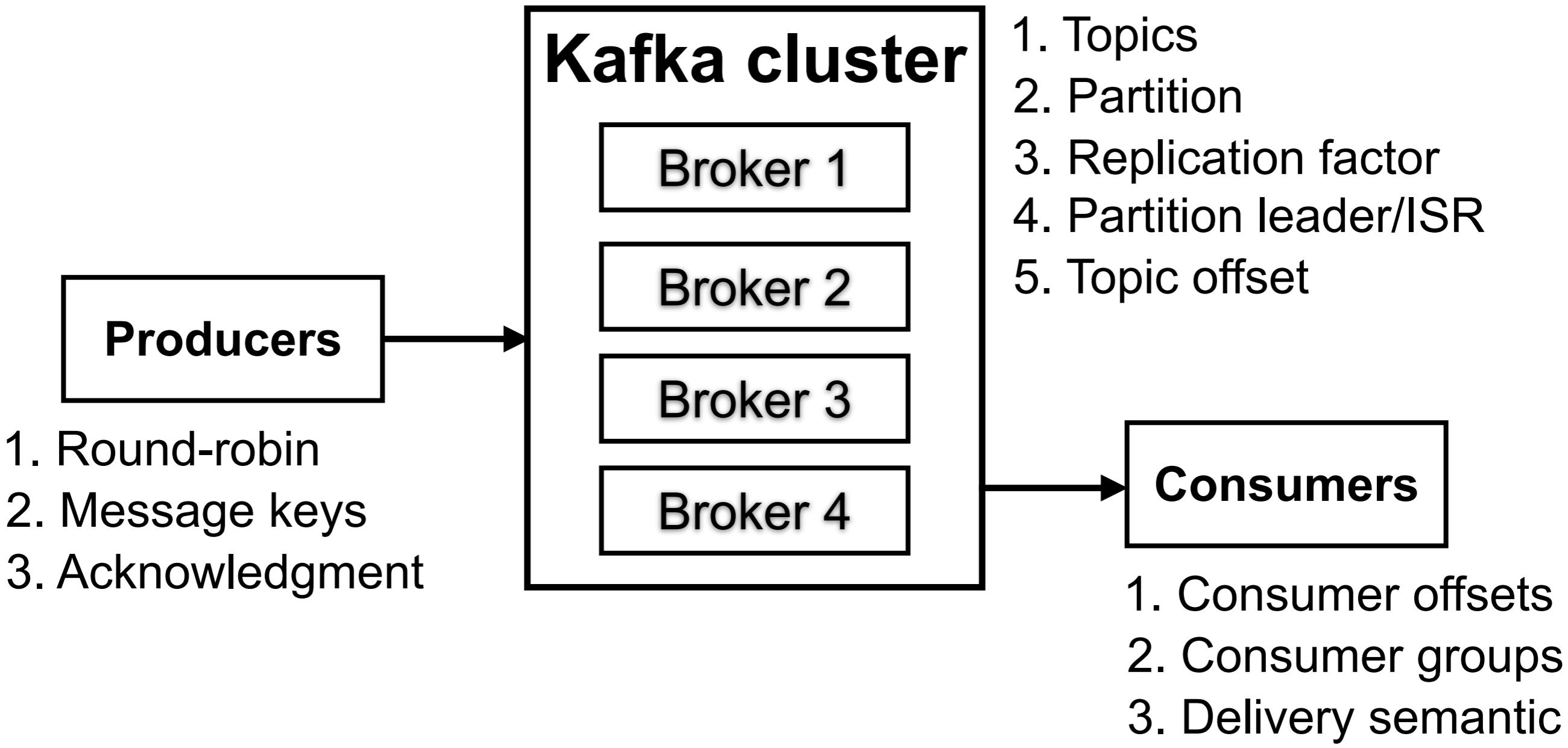
1. Topics
2. Partition
3. Replication factor
4. Partition leader/ISR
5. Topic offset



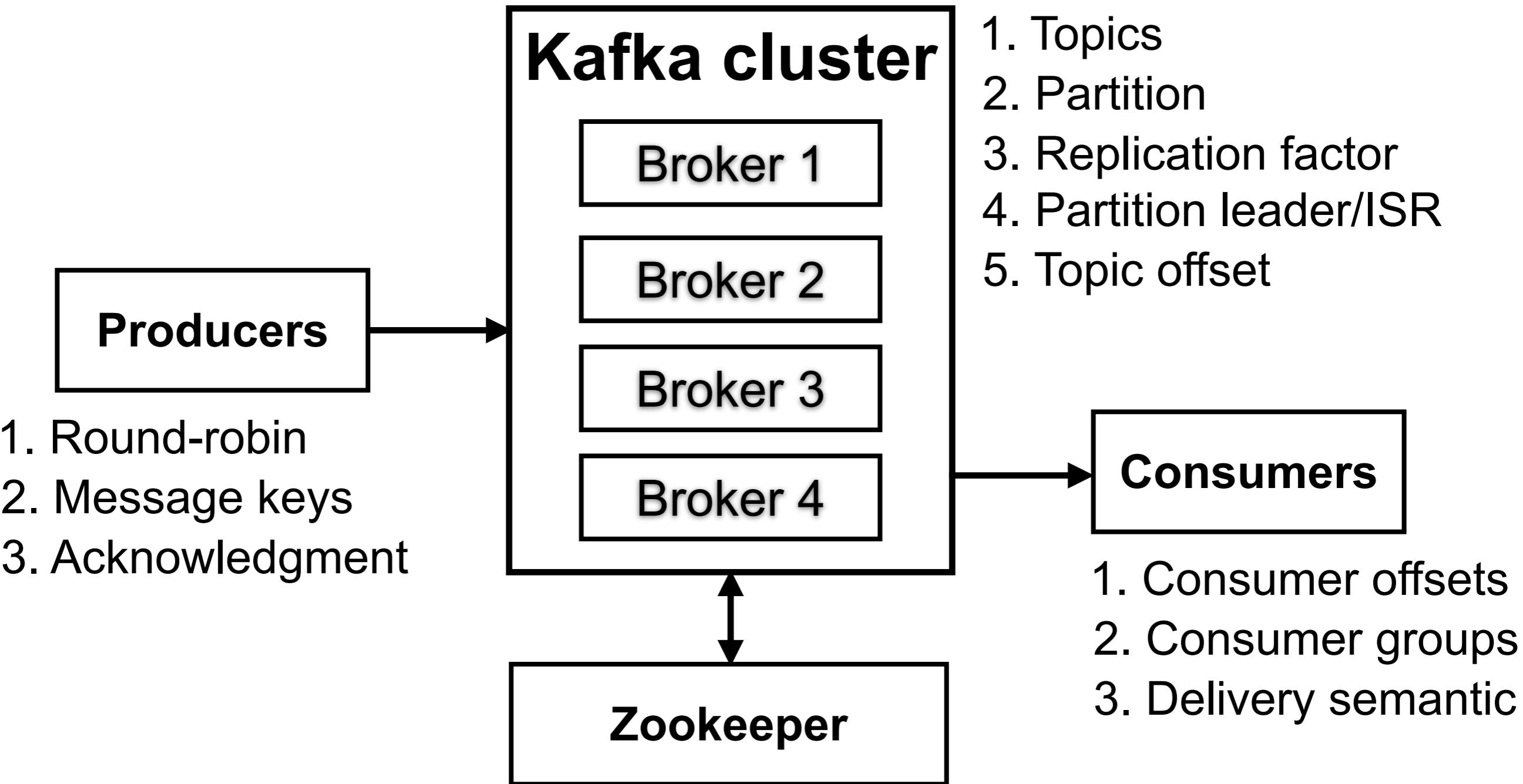
Kafka concepts



Kafka concepts



Kafka concepts



Kafka workshop



Steps

Start Zookeeper

Start Kafka broker

Create topic

Produce message to topic

Consume message from topic



Kafka + Java



Workshop



Steps

Create project

Message for publish and read from Topic

Configuration

Create producer

Create consumer

Create REST controller



Create project



Project
 Maven Project Gradle Project

Language
 Java Kotlin Groovy

Spring Boot
 3.0.0 (SNAPSHOT) 3.0.0 (M4) 2.7.3 (SNAPSHOT) 2.7.2
 2.6.11 (SNAPSHOT) 2.6.10

Project Metadata

Group	com.example
Artifact	kafka
Name	kafka
Description	Demo project for Spring Boot
Package name	com.example.kafka
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 18 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

Dependencies

[ADD DEPENDENCIES... ⌘ + B](#)

Spring for Apache Kafka MESSAGING

Publish, subscribe, store, and process streams of records.

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

<https://start.spring.io/>



Kafka with Java

© 2017 - 2018 Siam Chamnankit Company Limited. All rights reserved.

Spring for Apache Kafka

KafkaTemplate

KafkaMessageListnerContainer

@KafkaListener

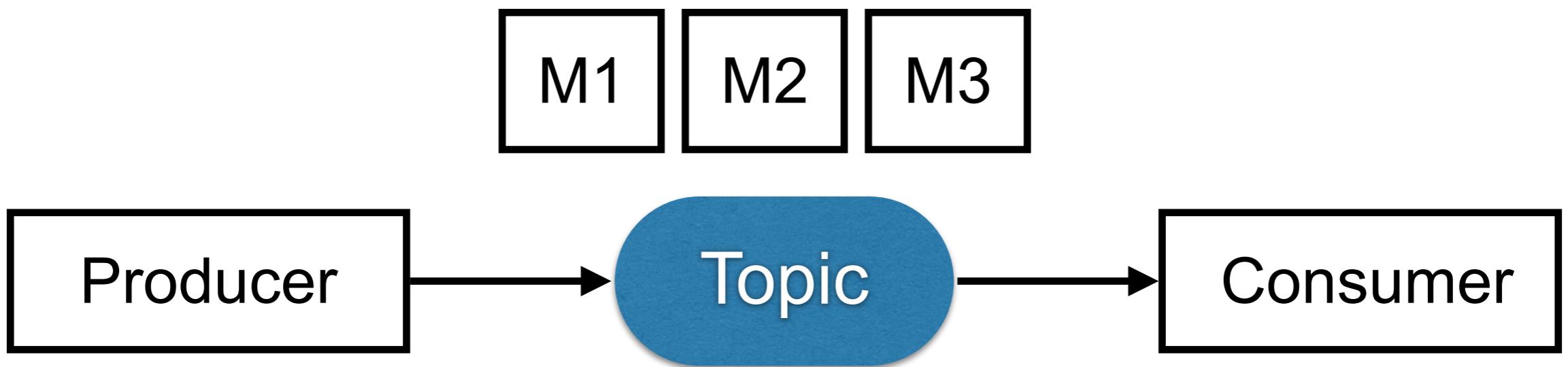
KafkaTransactionManager

spring-kafka-test (embedded Kafka server)

<https://spring.io/projects/spring-kafka>

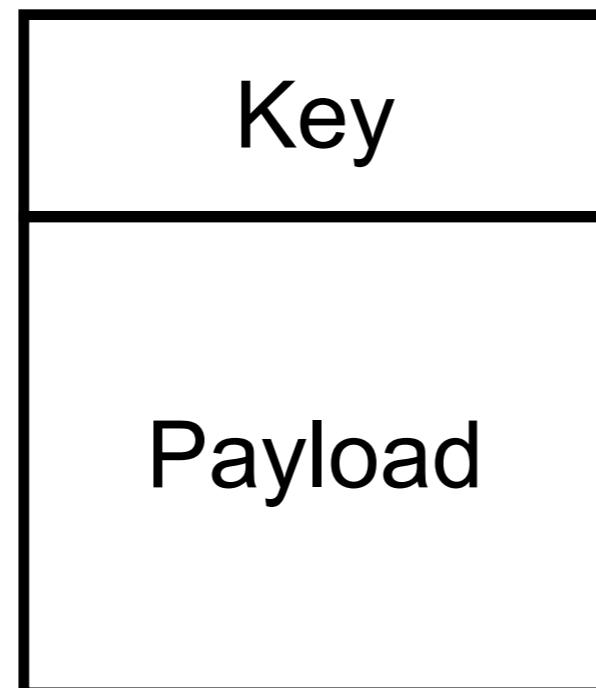


Message



Message format

Text
Binary



Working with JSON

```
{  
  "id": 1,  
  "name": "User 01",  
  "age": 30  
}
```

```
public class NewUser{  
  
  private String name;  
  private int id;  
  private int age;  
  
  ...  
  
}
```



Configuration

Application.properties or YAML
Java configuration

<https://docs.spring.io/spring-kafka/docs/current/reference/html/>



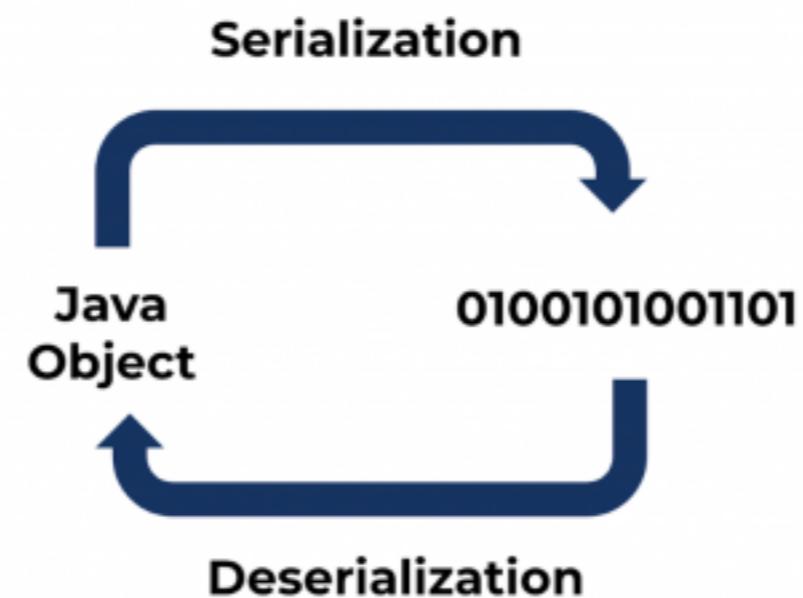
Application.yml

```
spring:  
  kafka:  
    consumer:  
      bootstrap-servers: localhost:9092  
      auto-offset-reset: earliest  
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
    producer:  
      bootstrap-servers: localhost:9092  
      key-serializer: org.apache.kafka.common.serialization.StringSerializer  
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
```

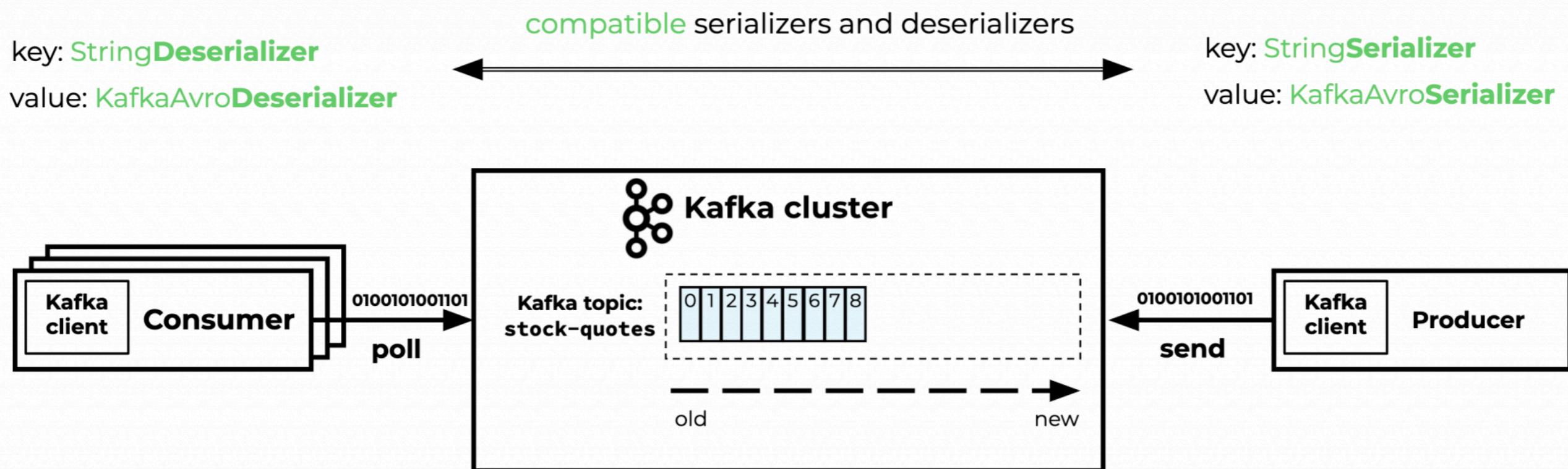
<https://kafka.apache.org/documentation/#configuration>



Serialization and Deserialization



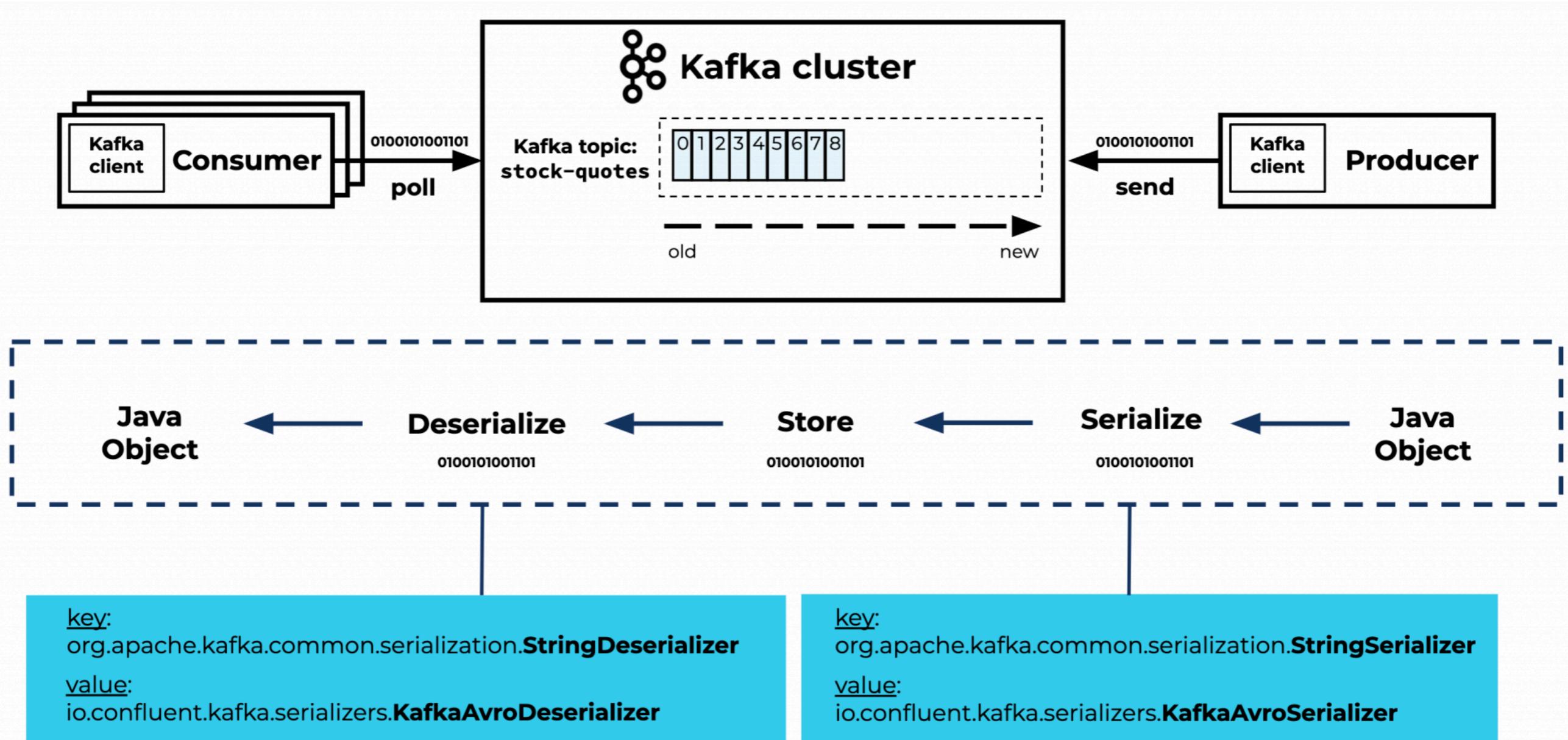
Serialization and Deserialization



<https://www.confluent.io/blog/spring-kafka-can-your-kafka-consumers-handle-a-poison-pill/>



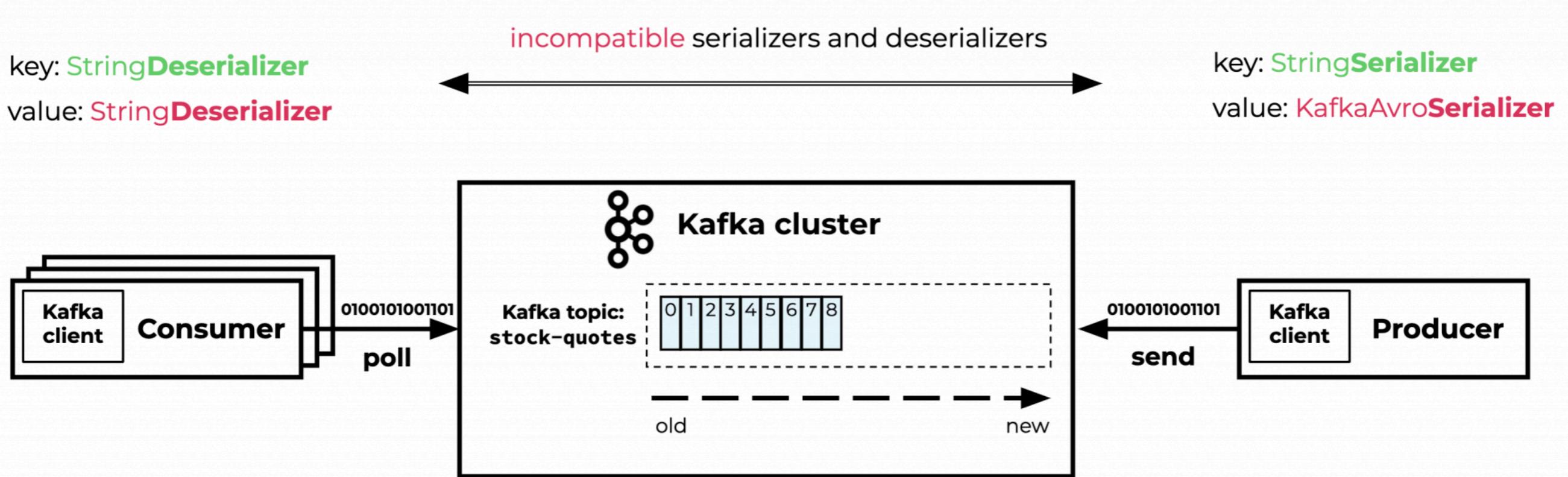
Serialization and Deserialization



<https://www.confluent.io/blog/spring-kafka-can-your-kafka-consumers-handle-a-poison-pill/>



Problem !!



Auto-offset-reset

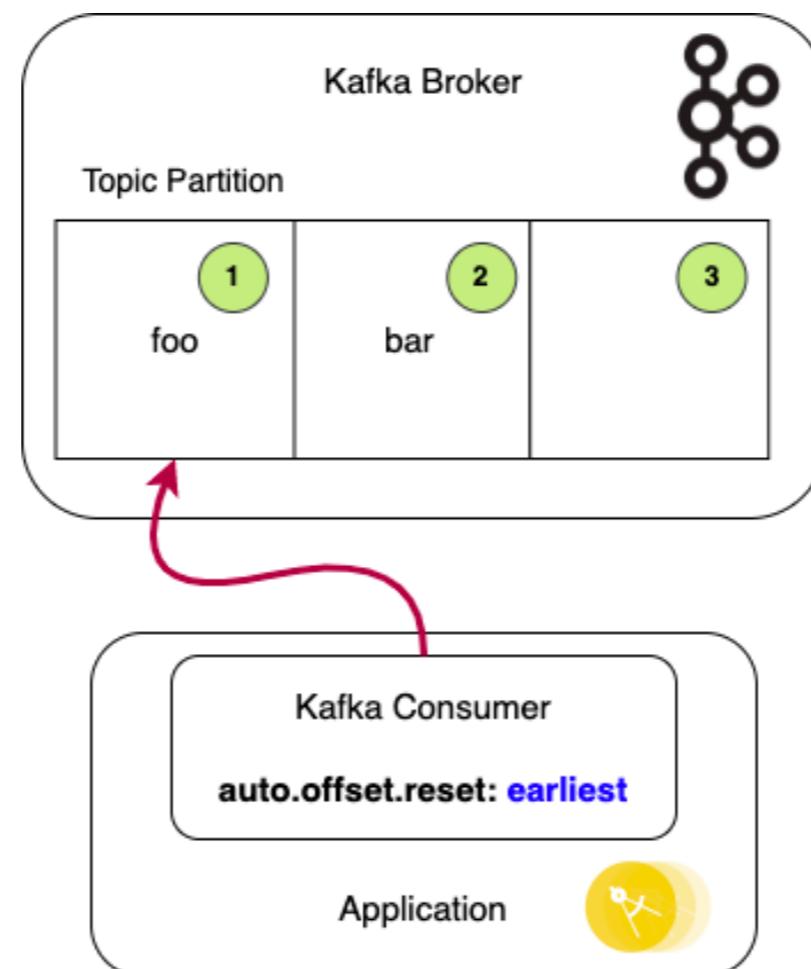
earliest
latest (default)
none

https://kafka.apache.org/documentation/#consumerconfigs_auto.offset.reset



Earliest

Consume from the beginning of topic partition



Earliest

of messages are depend on **retention.ms**

retention.ms

This configuration controls the maximum time we will retain a log before we will discard old log segments to free up space if we are using the "delete" retention policy. This represents an SLA on how soon consumers must read their data. If set to -1, no time limit is applied.

Type: long

Default: 604800000 (7 days)

Valid Values: [-1,...]

Server Default Property: log.retention.ms

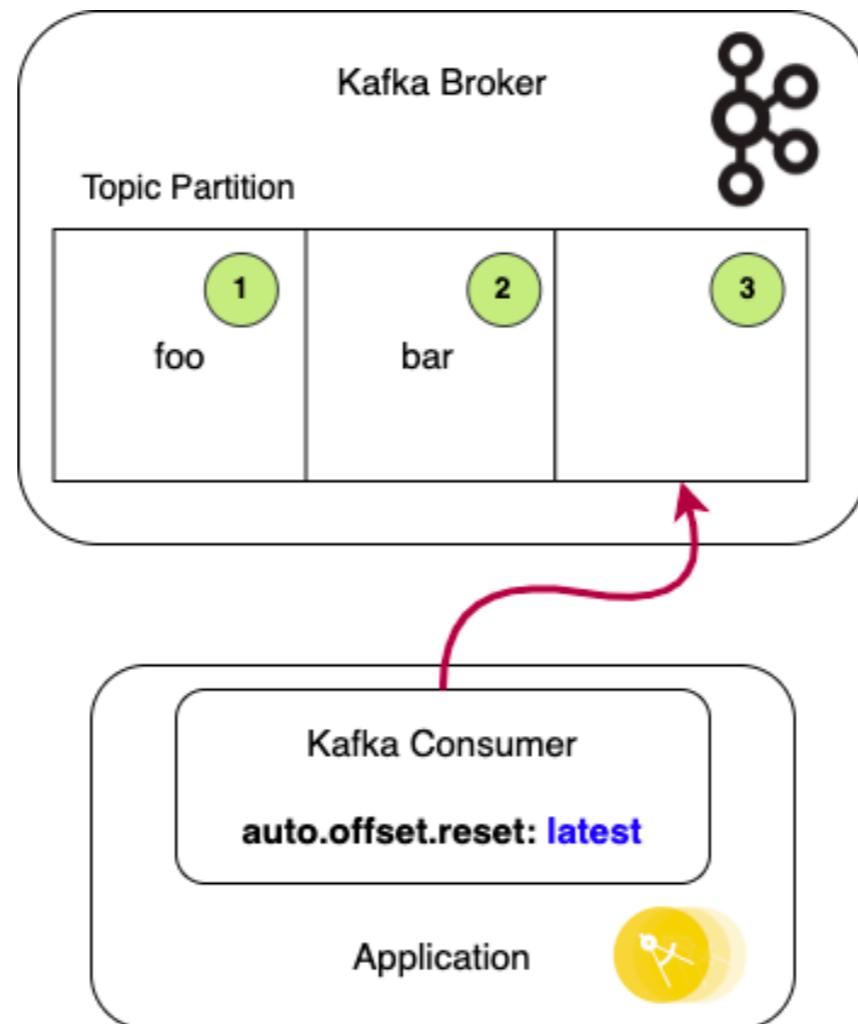
Importance: medium

https://kafka.apache.org/documentation/#topicconfigs_retention.ms



Latest

Consume from the end of topic partition



None

Throw an exception if no offset present for the consumer group



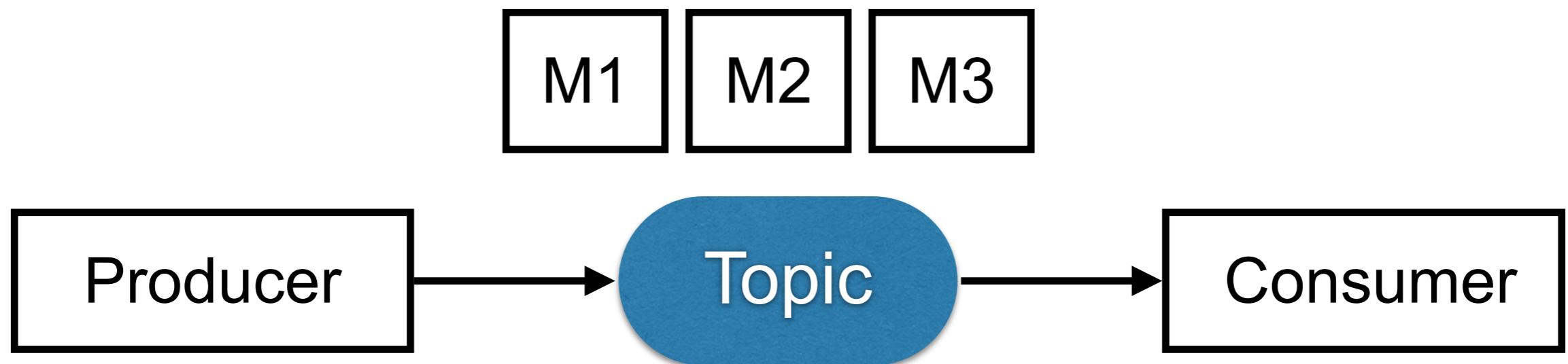
Create producer

Create a message
Send to topic in Kafka server

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



Create producer



Working with KafkaTemplate

```
@Service  
public class DemoProducer {  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    public void sendMessage(String message) {  
        this.kafkaTemplate.send("demo.topic", message);  
    }  
}
```



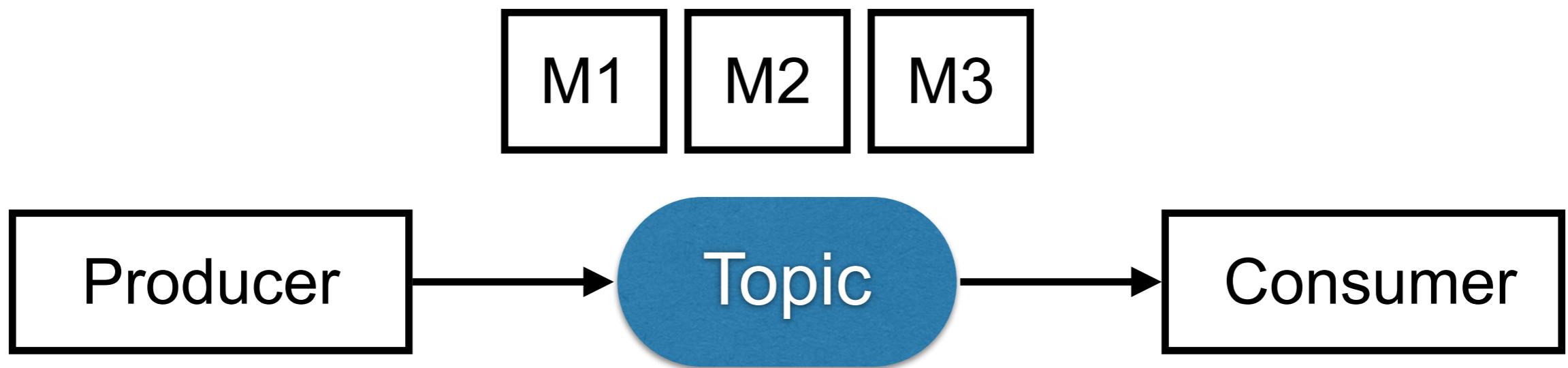
Create consumer

Read message from topic

<https://kafka.apache.org/documentation/#consumerconfigs>



Create consumer



Working with KafkaListener

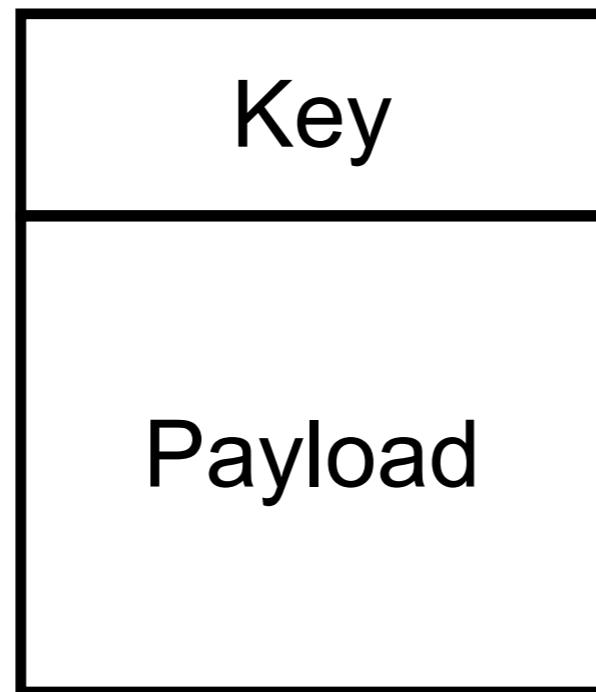
```
@Service  
public class DemoConsumer {  
  
    @KafkaListener(topics="demo.topic")  
    public void receiveData(String message) {  
        System.out.println("Receive data => " + message);  
    }  
  
}
```



Working with Message format



Message format



Use ProducerRecord

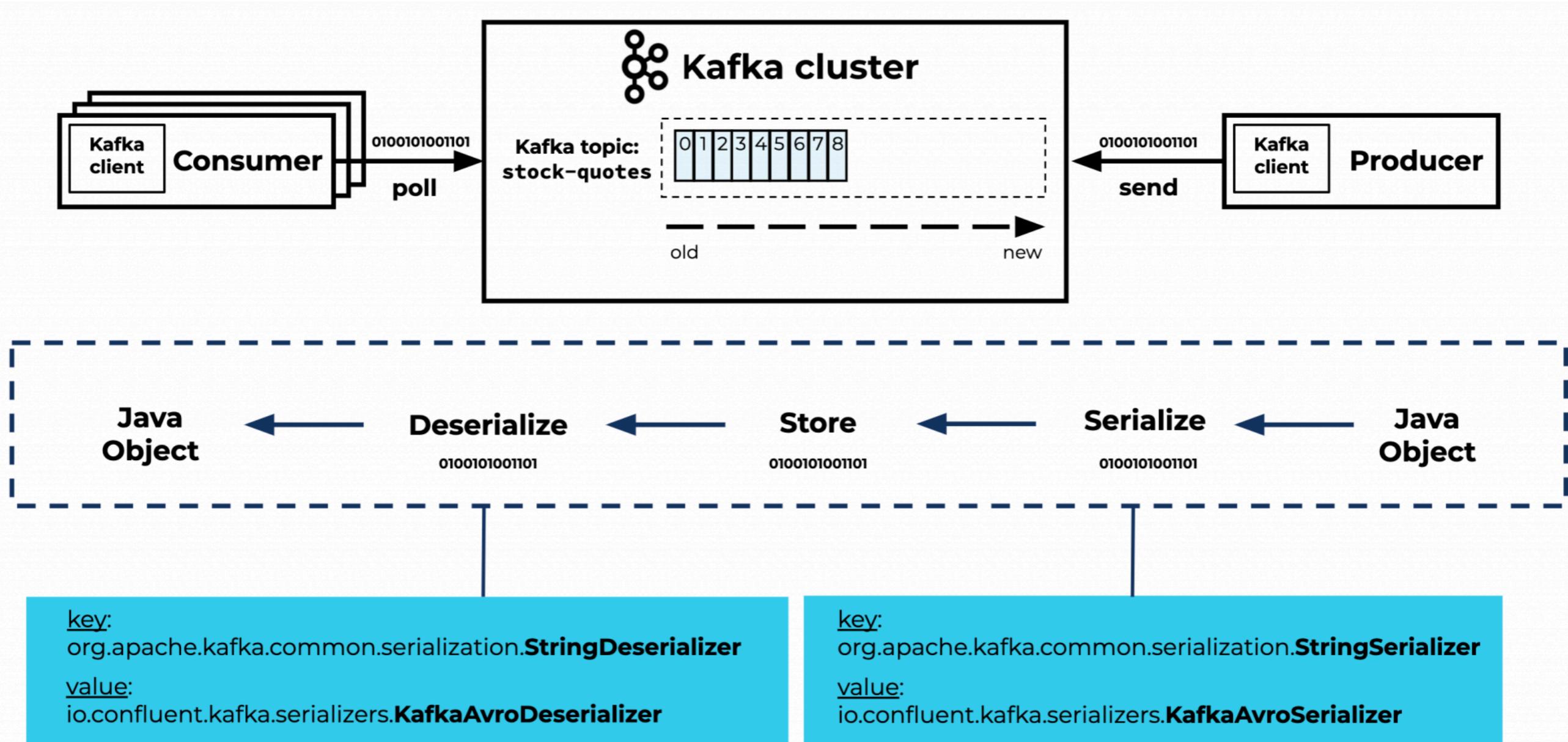
```
public SendResult<String, String> send(String key, String message) {  
  
    // Create message with key and payload  
    ProducerRecord<String, String> record  
        = new ProducerRecord<>("topic", key, message);  
  
    // Send message  
    try {  
        SendResult<String, String> result = template.send(record).get();  
        return result;  
    } catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}
```



Serialization and Deserialization



Serialization and Deserialization

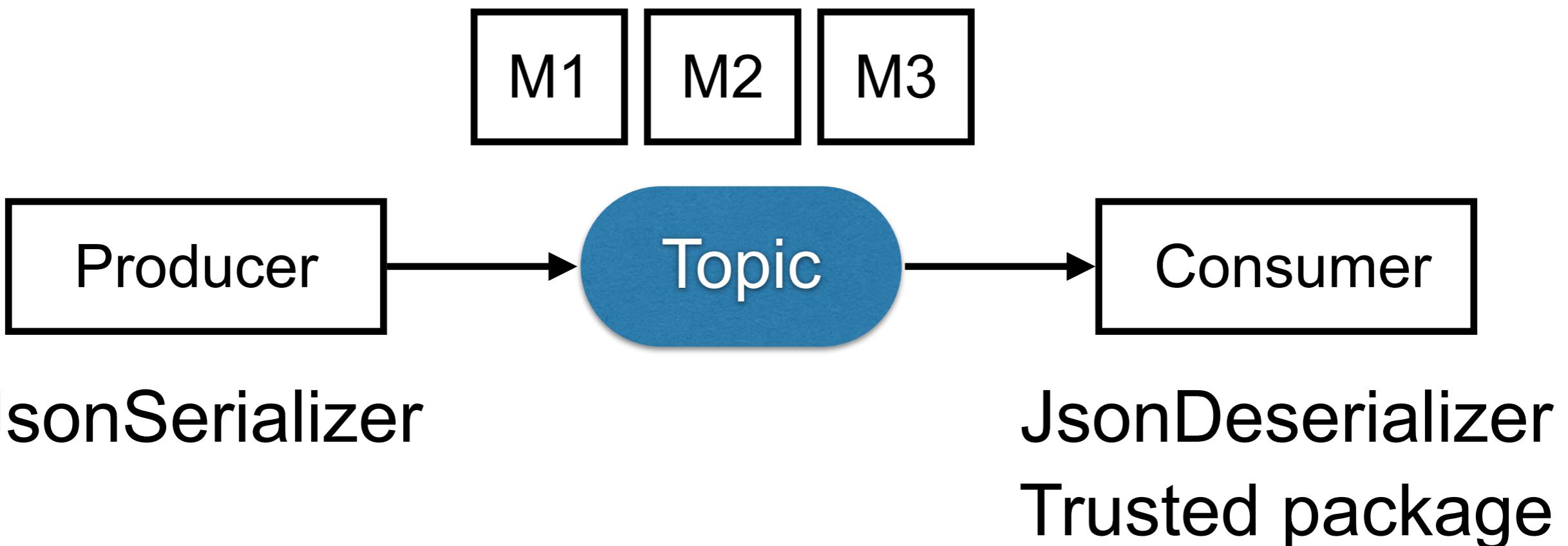


<https://www.confluent.io/blog/spring-kafka-can-your-kafka-consumers-handle-a-poison-pill/>



JSON

Use JSON format in value



JSON

Configuration in Spring project

kafka:

```
consumer:  
  bootstrap-servers: localhost:9092  
  group-id: ${spring.application.name}-group  
  auto-offset-reset: latest  #earliest/latest/none  
  value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  
properties:  
  spring:  
    json:  
      trusted:  
        packages: com.example.kafka.models  
  
producer:  
  bootstrap-servers: localhost:9092  
  value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```



JSON

Configuration in Spring project

kafka:

```
consumer:  
  bootstrap-servers: localhost:9092  
  group-id: ${spring.application.name}-group  
  auto-offset-reset: latest  #earliest/latest/none  
  value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  
properties:  
  spring:  
    json:  
      trusted:  
        packages: com.example.kafka.models  
producer:  
  bootstrap-servers: localhost:9092  
  value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
```



Apache Avro



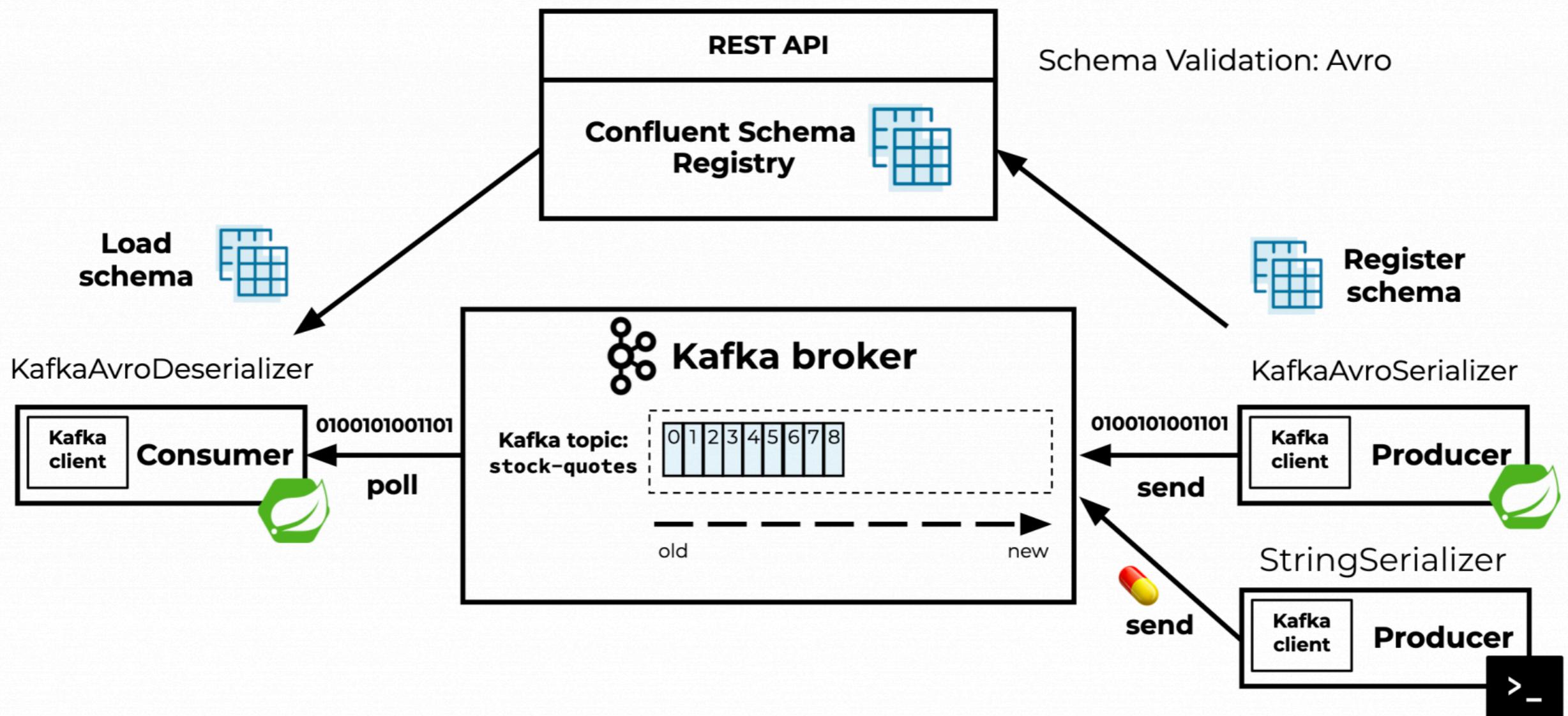
The screenshot shows the Apache Avro website. At the top, there is a dark blue header bar with the Apache Avro logo on the left, followed by the text "APACHE AVRO". To the right of the logo are four navigation links: "Project", "Blog", "Community", and "Documentation". Below the header, the main title "Apache Avro™ - a data serialization system" is displayed in large, bold, black font. Underneath the title are two buttons: a blue "Learn More" button with a white arrow icon, and an orange "Download" button with a white download icon.

<https://avro.apache.org/>



Working Schema Registry

Schema validation with Apache Avro



<https://github.com/confluentinc/schema-registry>



Apache Avro with Kafka

Add dependency and repository

```
<dependency>
    <groupId>io.confluent</groupId>
    <artifactId>kafka-avro-serializer</artifactId>
    <version>5.5.0</version>
</dependency>
```



Apache Avro with Kafka

Add dependency and repository

```
<repositories>
    <!-- Confluent maven repo, required to Confluent Kafka Avro Dependencies -->
    <repository>
        <id>confluent</id>
        <url>https://packages.confluent.io/maven/</url>
    </repository>
</repositories>
```



Generate class from Avro

```
<plugin>
  <groupId>org.apache.avro</groupId>
  <artifactId>avro-maven-plugin</artifactId>
  <version>1.8.2</version>
  <executions>
    <execution>
      <id>schemas</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>schema</goal>
        <goal>protocol</goal>
        <goal>idl-protocol</goal>
      </goals>
      <configuration>
        <sourceDirectory>${project.basedir}/src/main/resources/avro/</
sourceDirectory>
        <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```



Producer

Value serializer

kafka:

 bootstrap-servers: localhost:9092

 producer:

 key-serializer: org.apache.kafka.common.serialization.StringSerializer

 value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer

 client-id: \${spring.application.name}

 properties:

 enable.idempotence: true

 schema.registry.url: http://localhost:8081



Producer

Connect to schema registry

kafka:

 bootstrap-servers: localhost:9092

 producer:

 key-serializer: org.apache.kafka.common.serialization.StringSerializer

 value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer

 client-id: \${spring.application.name}

 properties:

 enable.idempotence: true

 schema.registry.url: http://localhost:8081



Consumer

Value deserializer

consumer:

```
key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
value-deserializer: io.confluent.kafka.serializers.KafkaAvroDeserializer
client-id: ${spring.application.name}
group-id: ${spring.application.name}-group
auto-offset-reset: earliest
properties:
    schema.registry.url: http://localhost:8081
    specific.avro.reader: true
```



Consumer

Connect to schema registry

consumer:

```
key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
value-deserializer: io.confluent.kafka.serializers.KafkaAvroDeserializer
client-id: ${spring.application.name}
group-id: ${spring.application.name}-group
auto-offset-reset: earliest
properties:
    schema.registry.url: http://localhost:8081
    specific.avro.reader: true
```



Testing



Testing

Unit test

Integration test

Component test

End-to-end test

