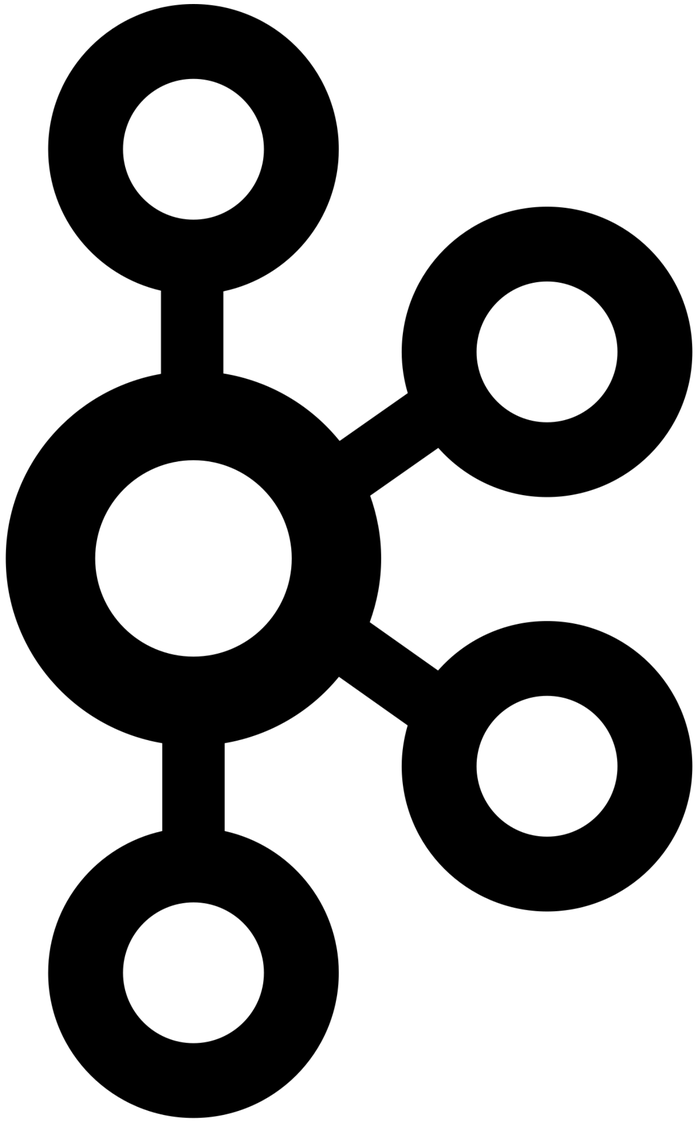


Kafka for Beginners

Dilip Sundarraj

About Me

- Dilip
- Building Software’s since 2008
- Teaching Online Since 2016

Whats Covered?

- Introduction to Kafka and internals of Kafka
- Learn to build Kafka Producers/Consumers using Java
- Covers advanced Kafka Producer and Consumer concepts
- Hands on Oriented course

Targeted Audience

- Kafka Beginners and Advanced
- Interested in building java applications using producer and consumer API
- Interested in learning advanced Kafka Producer and Consumer operations

Source Code

Thank You !

Sending Messages using Producer API

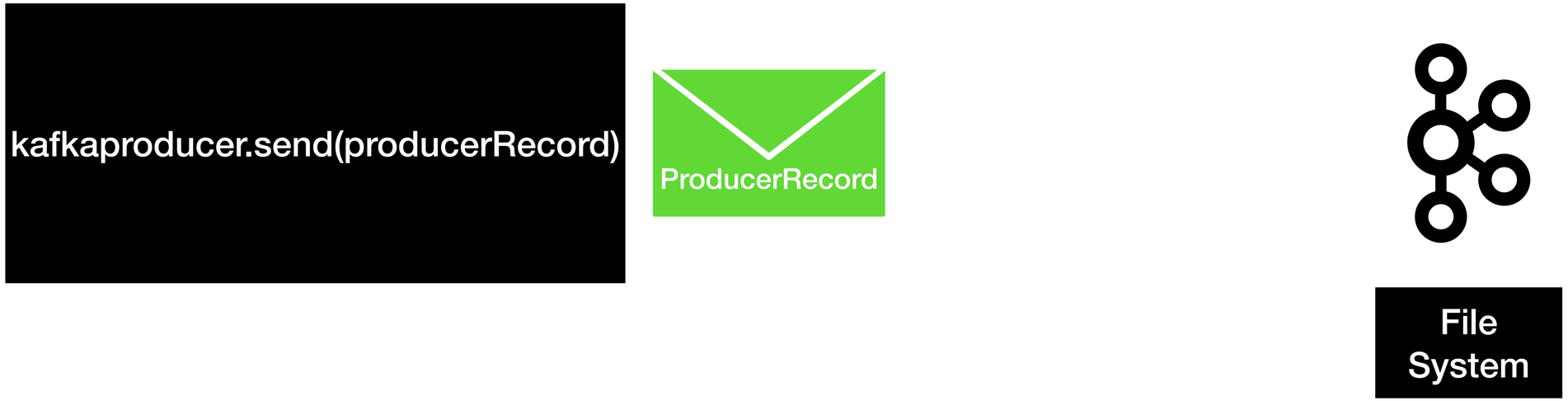
Producer API

- KafkaProducer
  - Class through which we interact with Kafka to produce new Records
  - Producer Properties

```
bootstrap.servers – “localhost:9092, localhost:9093, localhost:9094”
key.serializer – org.apache.kafka.common.serialization.StringSerializer
value.serializer – org.apache.kafka.common.serialization.StringSerializer
```

KafkaProducer.send()

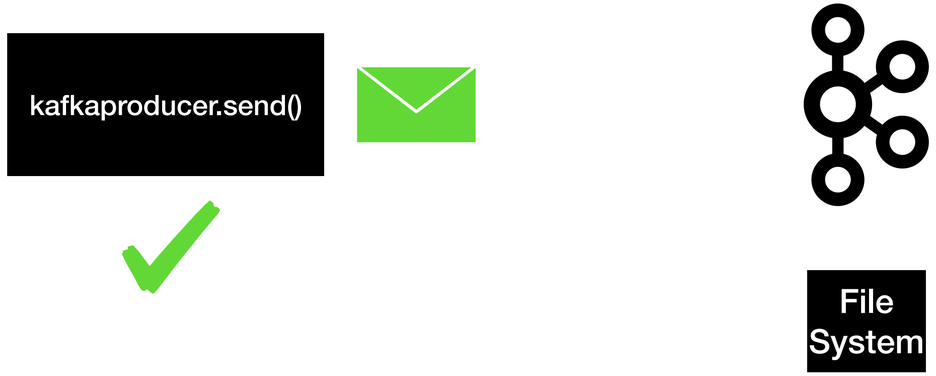
- KafkaProducer uses the **send()** method to produce the record to Kafka
- **ProducerRecord** is the data container:
  - Key and Value



KafkaProducer.send()

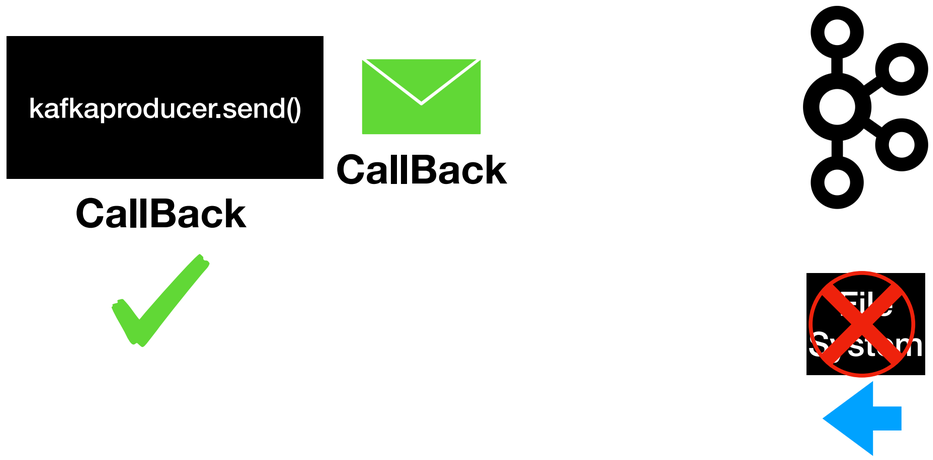
Synchronous

- The **send()** call waits until the messages is published and persisted in to the File System and replicas



Asynchronous

- The **send()** does not wait for the message to the published and persisted in to the file system and replicas



Logging using Logback

Why Logger?

- We have used **System.out.println()** until now
- SysOut does not provide more visibility on what's happening behind the scenes
- Pretty common for applications to have logger
  - Debugging
  - Exception Logging

Logback

- **Logback** is the successor of **log4j**
- **Logback** is pretty popular today when it comes to logging
- XML/Groovy based configuration

How to configure Logback ?

- Add logback dependency in the build.gradle file

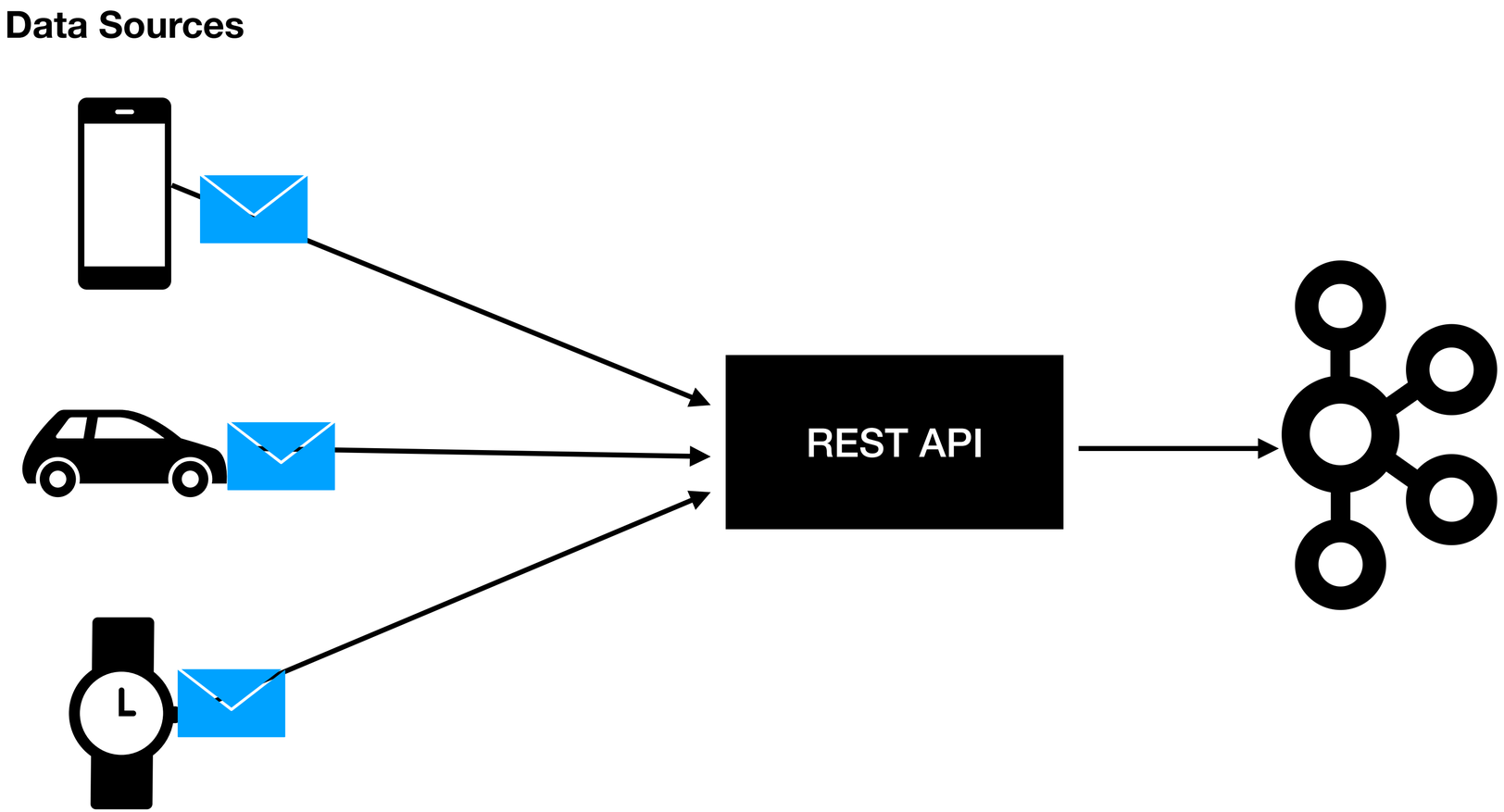
```
implementation group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'
```

- Add **logback.xml** file in the classpath

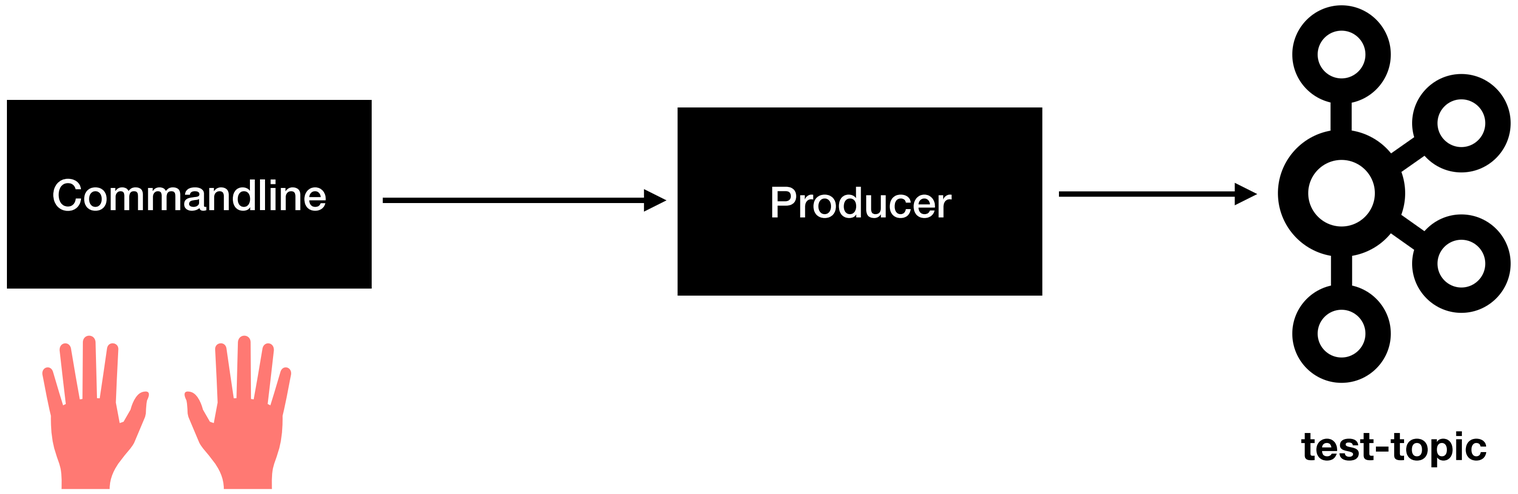
```
<configuration>

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

How Data Flows into Kafka?

How Data Flows into Kafka?

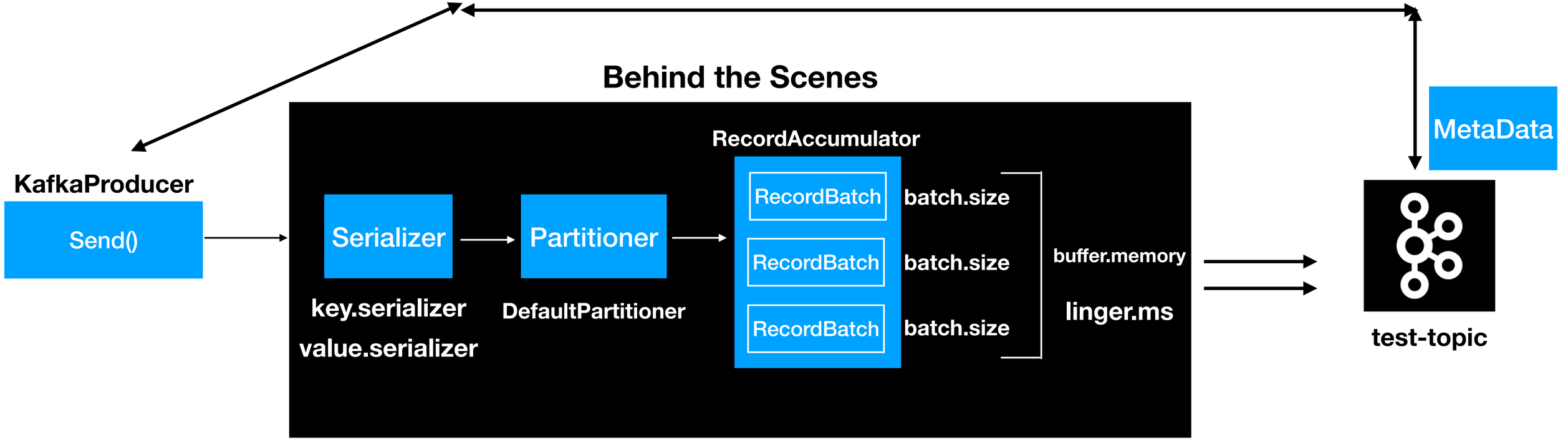


Command Line to Publish New Records



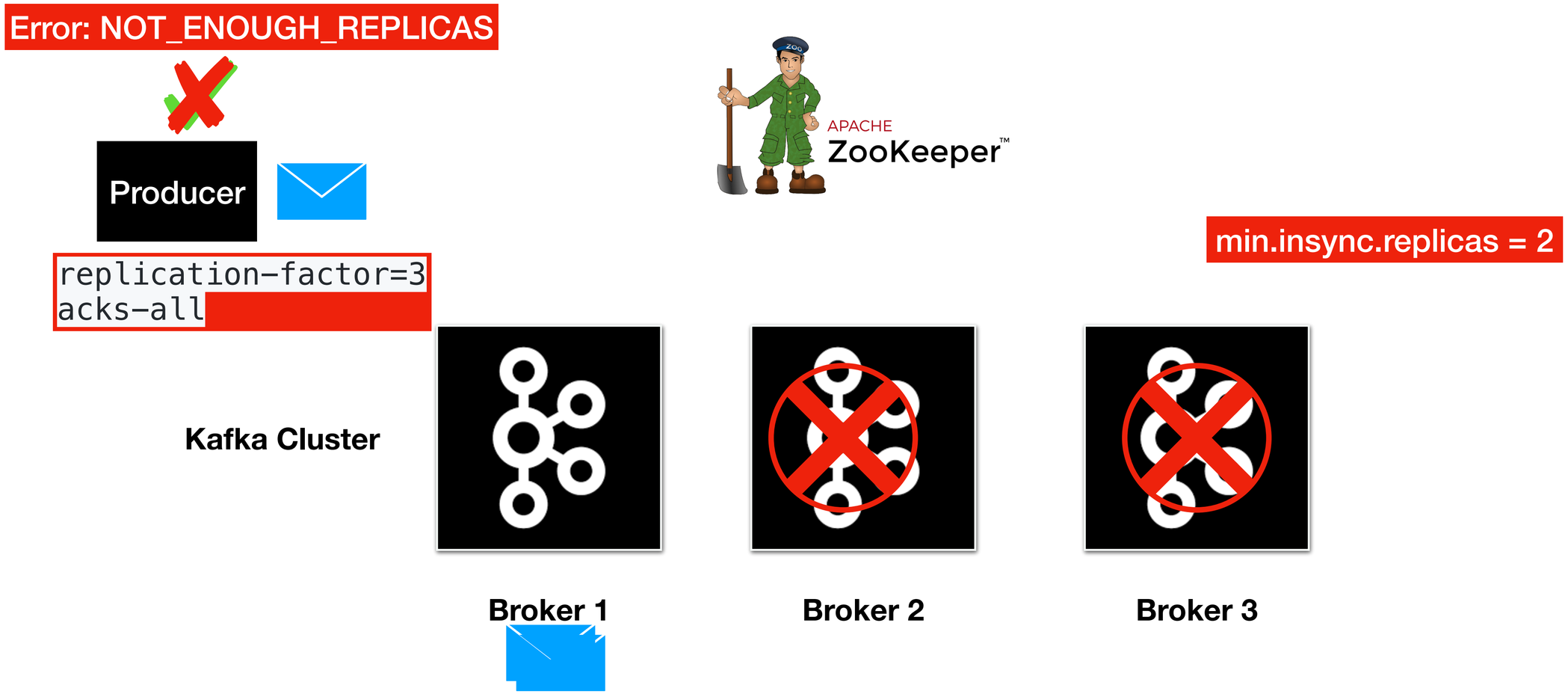
Producer API
(Behind the Scenes)

KafkaProducer.send()



Configuring acks & min.insync.replicas

min.insync.replicas



What does it guarantee?

- Guarantees always a replica of the record is available
- No Dataloss

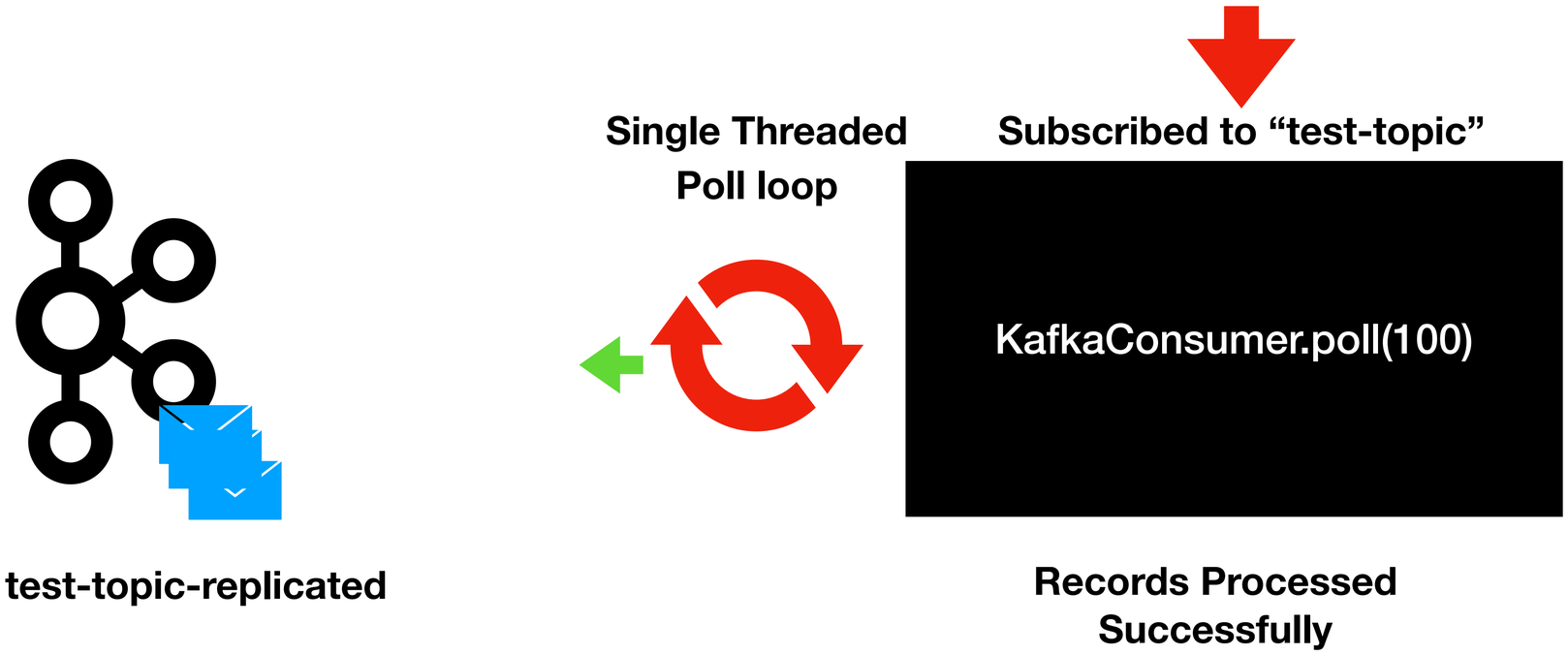
Consuming Messages using Consumer API

Consumer API

- KafkaConsumer
  - Class through which we can read messages from Kafka
  - **Consumer Properties**:

```
bootstrap.servers  – “localhost:9092, localhost:9093, localhost:9094”
key.deserializer   – org.apache.kafka.common.serialization.StringDeserializer
value.deserializer – org.apache.kafka.common.serialization.StringDeserializer
group.id           – test-consumer
```

poll() loop- Consumer API



auto.offset.reset

auto.offset.reset

- **auto.offset.reset** - Property is used instruct the Kafka consumer to read either from the beginning offset or the latest offset of the topic with the given group.id when the consumer makes the connection to the kafka topic for the very first time
    - beginning offset of the topic
        - auto.offset.reset = **earliest**
    - latest offset of the topic (**Default**)
        - auto.offset.reset = **latest**

Kafka Consumer Configurations

Consumer Configurations

- **auto.offset.reset** - Property is used instruct the Kafka consumer to read either from the beginning offset or the latest offset of the topic with the given group.id when the consumer makes the connection to the kafka topic for the very first time
    - beginning offset of the topic
        - auto.offset.reset = earliest
    - latest offset of the topic (**Default**)
        - auto.offset.reset = latest

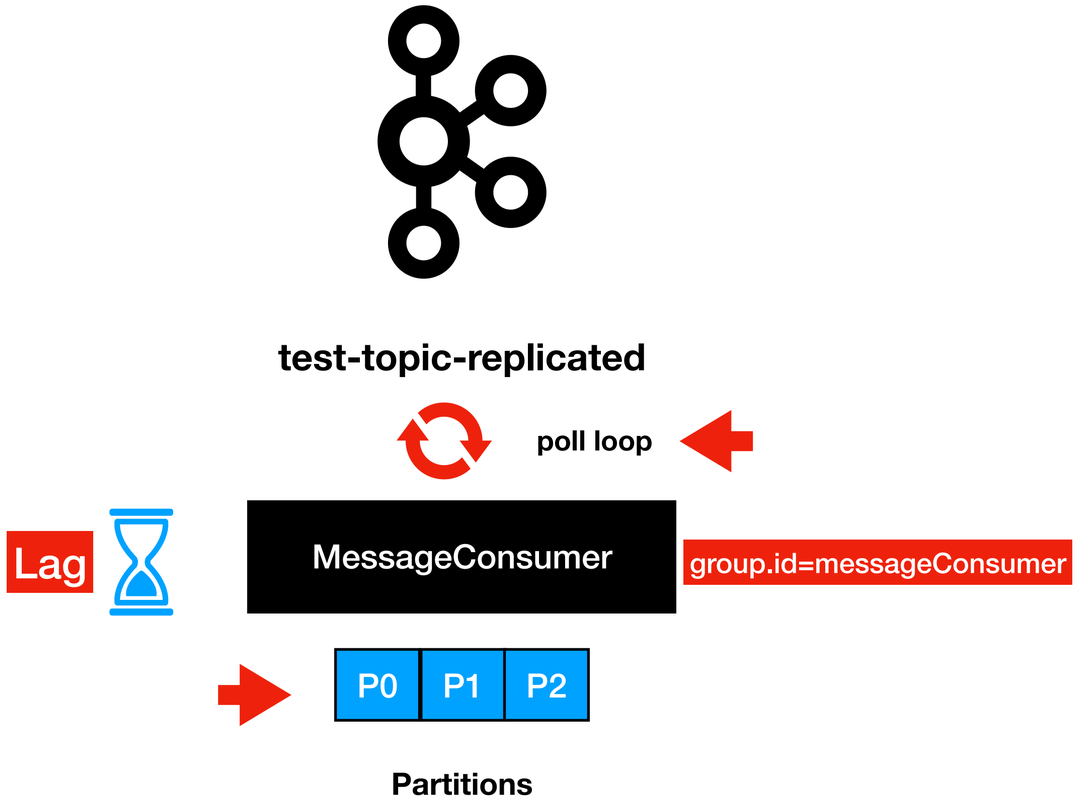
Consumer Configurations

- **max.poll.interval.ms -** The maximum delay between poll calls from the consumer

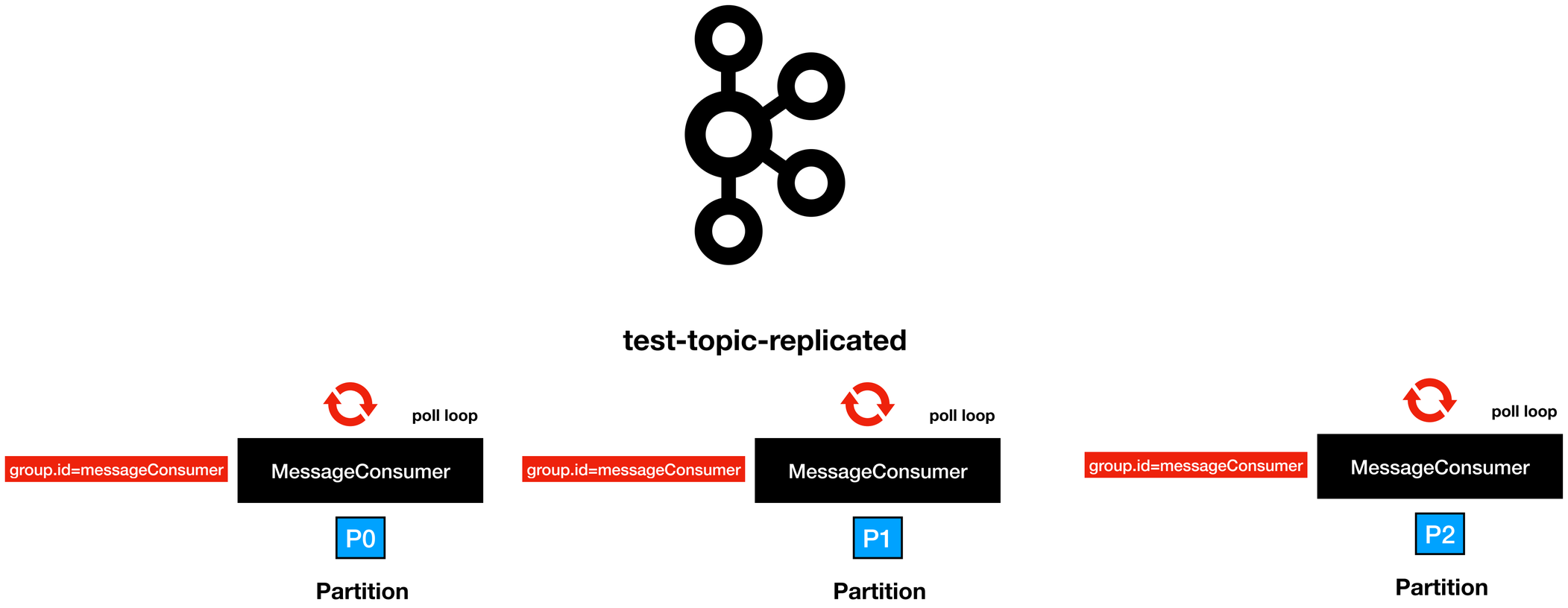
Consumer Groups

Consumer Groups

- Consumer Groups is the only way to scale the message consumption



Consumer Groups

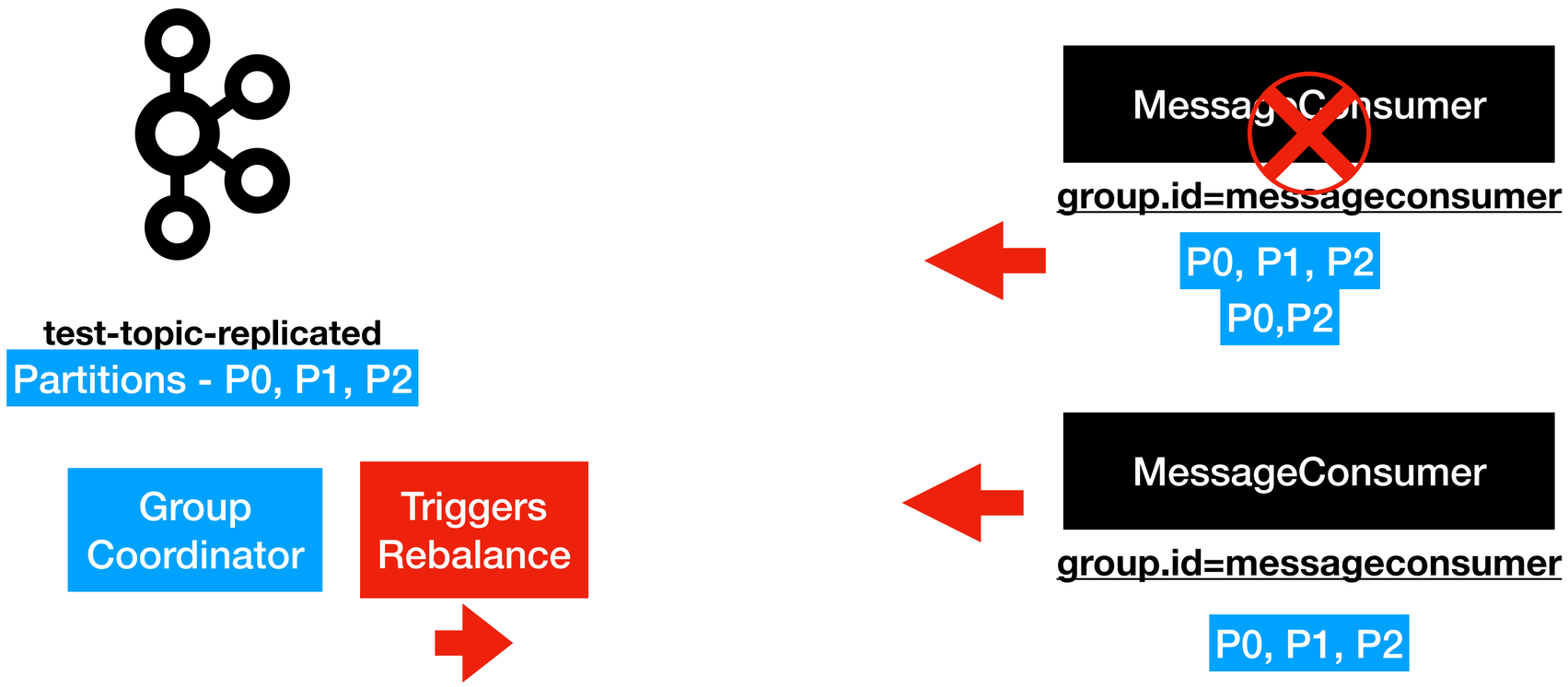


Consumer Rebalance

What is Consumer Rebalance?

- Consumer Rebalance is the concept of moving the partition ownership from one consumer to another
- Consumer Rebalance is important because it promises **scalability** and **availability**

Consumer Rebalance



max.poll.interval.ms

max.poll.interval.ms

- max.poll.interval.ms
  - The maximum delay between the poll() invocations from the consumer when using the consumer groups

```
while (true) {
    ConsumerRecords<String, String> consumerRecords = kafkaConsumer.poll(timeOutDuration);
    consumerRecords.forEach((record) -> {
        logger.info("Consumer Record Key is {} and the value is {} and the partion {}",
                record.key(), record.value(), record.partition());
    });
}
```

Default value of max.poll.interval.ms = 300000(5 mins)

max.poll.interval.ms

- What does this property have to do with Consumer Rebalance?
  - If two subsequent poll invocations take more than 5 mins then the Group Co-Ordinator triggers a **Rebalance**

Committing Consumer Offsets

Consumer Offsets

- What is an **Offset** ?
  - An offset is a sequence number that's represents a unique number for each record in a Kafka topic
- What are **Consumer Offsets**?
  - Consumer offsets provides tracking of records that are read by the consumer for a given group id
  - These offsets are present in the **__consumer_offsets** topic

Committing Consumer Offsets

- Consumers should commit offsets to the **__consumer_offsets** to keep track of the records read by them
- Separate process from **poll()** loop
- Whats the **Benefits of Committing Offsets** ?
  - Avoids **duplicate** processing of the same record
  - In the event of a **consumer crash**, the consumer knows what was the last read message and the consumer picks it up from where it left off once it up

Options to Committing Offsets

- Options for committing consumer offsets
  - **Option 1 - Auto Committing Offsets (Default)**
    - Committing offsets is automatically taken care for you by the consumer
    - No code needed
  - **Option 2 - Manually Committing Offsets (Default)**
    - Committing offsets explicitly from the code.
    - Two approaches to commit offsets
      - Commit offsets **Synchronously**
      - Commit Offsets **Asynchronously**

Option 1 - Auto Committing Offsets

- This is default option
- What configuration in Consumer enables this behavior ?
    - **enable.auto.commit = true**
    - **auto.commit.interval.ms = 5000**

Option 1 - Auto Committing Offsets

- Does this option work for all scenarios?
    - No
    - Consumer Rebalance within the 5 seconds before committing the offsets might reprocess the same message again.

Manually Committing Offsets

Manually Committing Offsets

Synchronous Commit

- commitSync()
- Application is blocked until the response is received from Kafka
- Any failure will be retried

Asynchronous Commit

- commitAsync()
- Application is not blocked because the commit invocation from the code -is asynchronous
- Any failure will not be retried

Rebalance Listeners

Rebalance Listeners

- This concept is related to Kafka Consumers Rebalance
- Consumer Rebalance occurs in the below scenarios:
  - Consumer goes down
  - New Consumer in to the consumer group
  - No **poll()** invocation within the **max.poll.interval.ms** config

Why Rebalance Listeners?

- **RebalanceListeners** is mainly used to perform some clean up work before partitions are revoked from the consumer instance
  - Committing Offsets
  - Closing DB Connections
- **RebalanceListeners** can also be used during partition assignment
  - Perform some initialization tasks
  - Seek to a specific offset , rather than just reading from the beginning or latest.

Coding Rebalance Listeners

- ConsumerRebalanceListener (**Interface**)

```
void onPartitionsRevoked(Collection<TopicPartition> partitions);
```

Clean Up Tasks

```
void onPartitionsAssigned(Collection<TopicPartition> partitions);
```

Initialization Tasks

Is this Mandatory for Kafka Consumer ?

- No

- Implement this only if its applicable for your consumer application

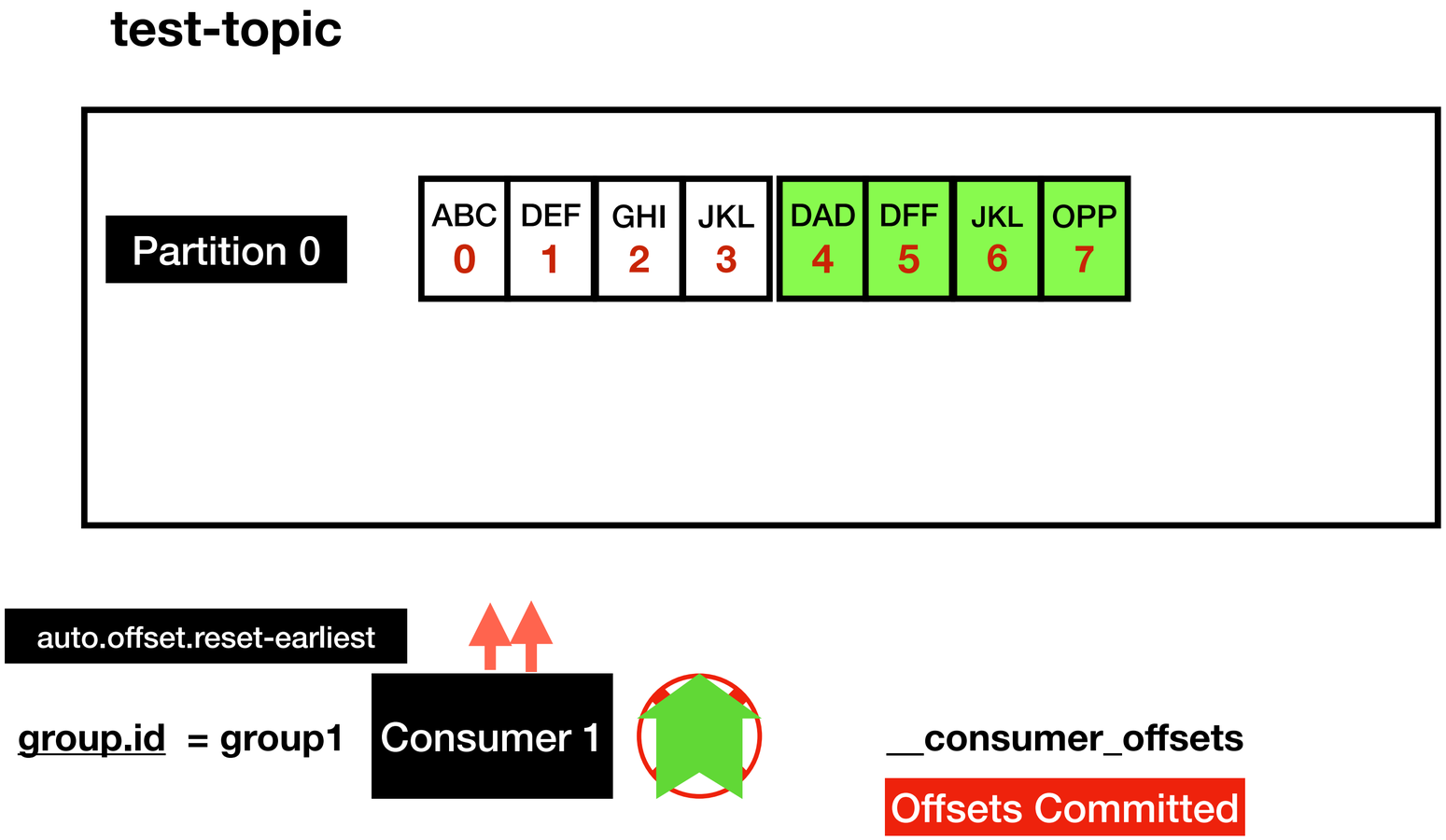
seekToBeginning()
&
seekToEnd()

seekToBeginning() & seekToEnd()

- Part of the KafkaConsumer class
- seekToBeginning()
  - Consumers always **seek** to read the records from beginning offset of the topic
- seekToEnd()
  - Consumer always **seek** to read the records from latest offset of the topic

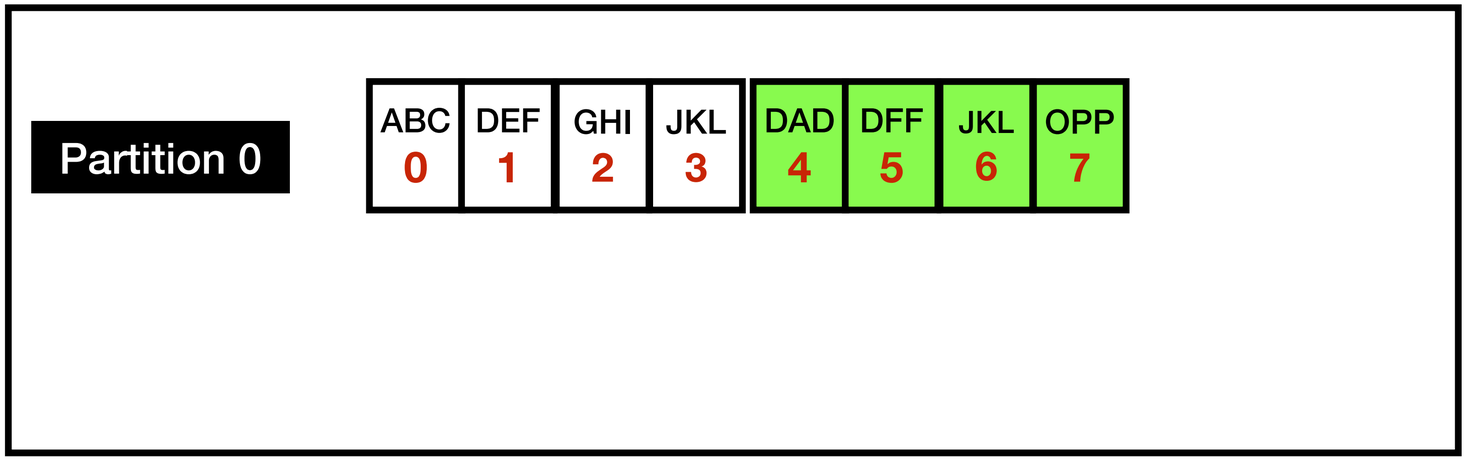
Consumer Offset Tracking is not applicable

Current Consumer Read Behavior



seekToBeginning()

test-topic



auto.offset.reset-earliest

<u>group.id</u> = group1

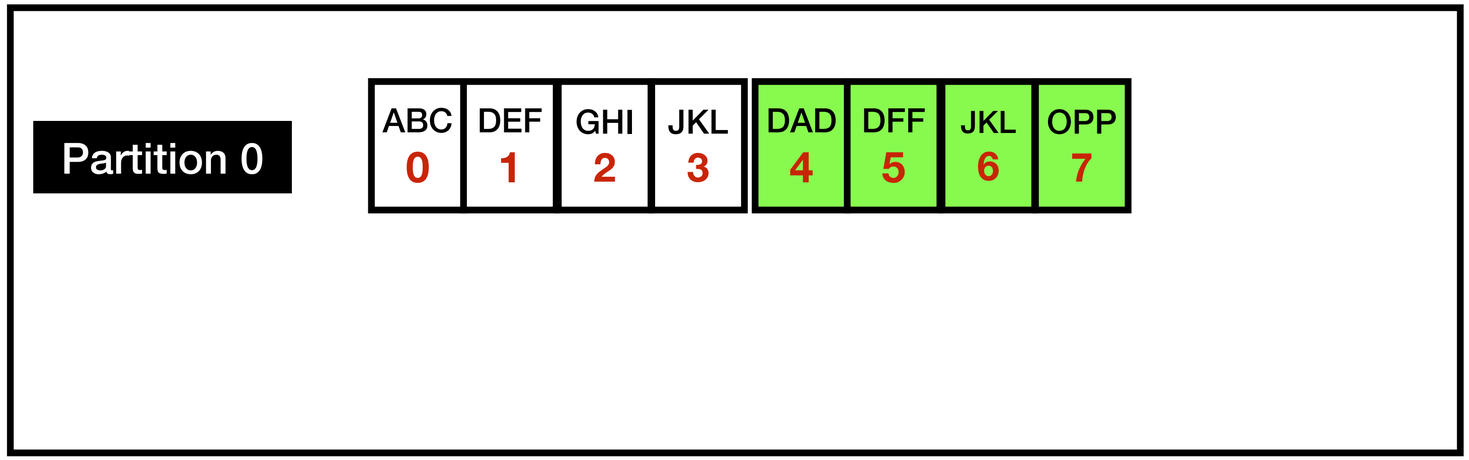
Consumer 1

When to use seekToBeginning?

- **seekToBeginning()**
  - Use-Case to read records from the beginning of the topic all the time
    - Example : Using Kafka as a DataStore for reference data(Compacted Topic)

seekToEnd()

test-topic



auto.offset.reset-earliest

group.id = group1

Consumer 1

When to use seekToEnd?

- **seekToEnd()**
    - Use-Case to read only the new records every time after the consumer is brought up

Seek
to a
Specific Offset

seek()

- KafkaConsumer class has a method **seek()** using which we can seek to a specific offset in the Topic

```
void seek(TopicPartition partition, long offset);

void seek(TopicPartition partition, OffsetAndMetadata offsetAndMetadata);
```

Why would you use seek() ?

Poll Loop

```
while (true) {
    ConsumerRecords<String, String> consumerRecords = kafkaConsumer.poll(timeOutDuration);
    consumerRecords.forEach((record) -> {
        logger.info("Consumer Record Key is {} and the value is {} and the partion {}",
                record.key(), record.value(), record.partition());
        // Invoke Some API
        // Persist the Record in DB
    });

    if(consumerRecords.count()>0){
        kafkaConsumer.commitSync(); // the last record offset returned by the poll
        logger.info("Offset Committed!");
    }
}
```

Duplicate Processing of the Record

Consumer Rebalance

How to avoid this ?

Poll Loop

Approach 1 - Using seek()

```
while (true) {
    ConsumerRecords<String, String> consumerRecords = kafkaConsumer.poll(timeOutDuration);
    consumerRecords.forEach((record) -> {
        logger.info("Consumer Record Key is {} and the value is {} and the partion {}",
                record.key(), record.value(), record.partition());
        // Invoke Some API
        // Persist the Record in DB
        // Persist the Consumer Offsets in DB
    });
}
```

} Transaction

With this approach we need to use seek() method to seek to a specific offset from the consumer end

Consumer reads the offset from the external system(DB) and then seek to the point where it left off

How to avoid this ?

Poll Loop

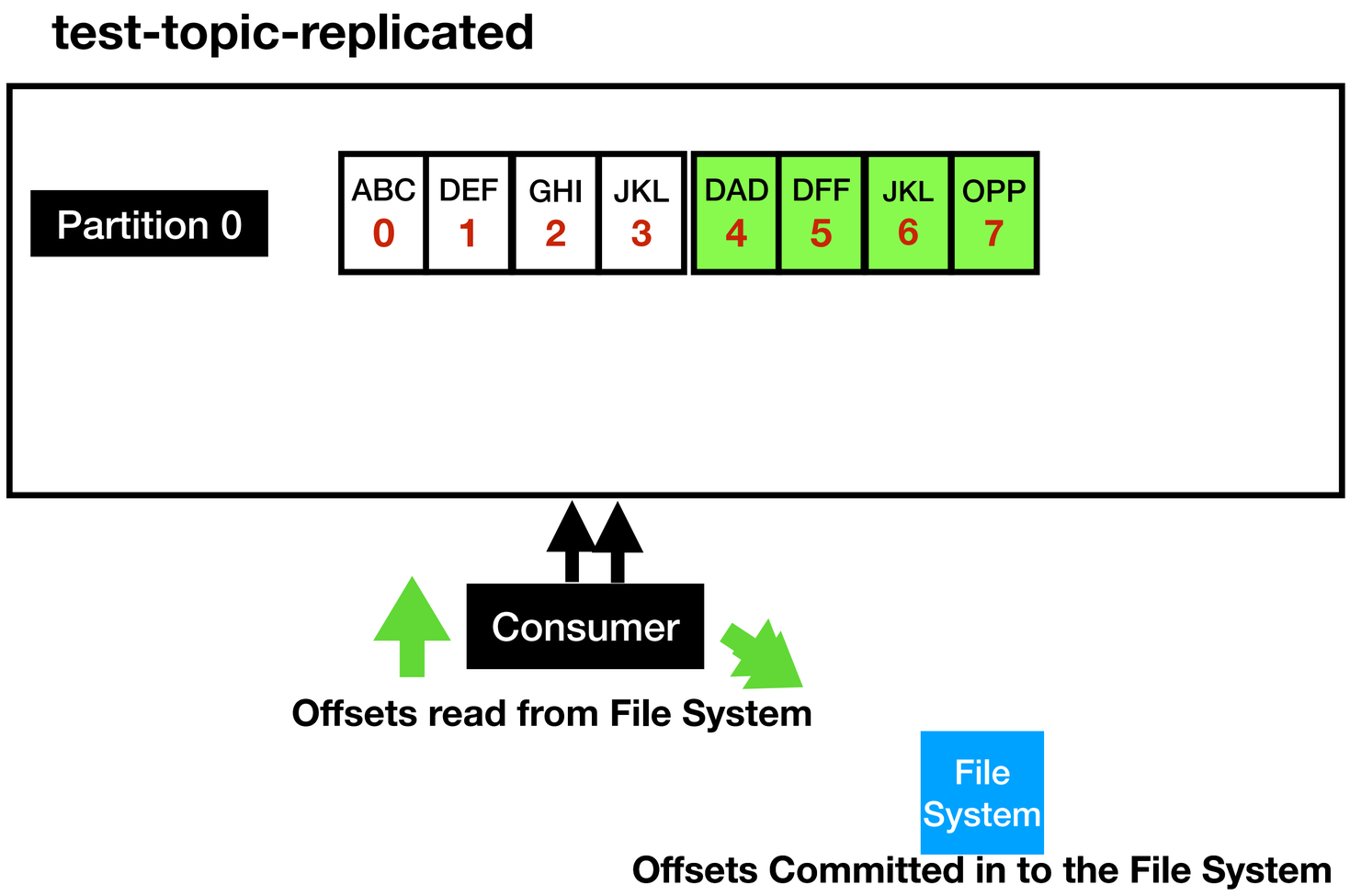
Approach2 - Perform Duplicate Check

```
while (true) {
    ConsumerRecords<String, String> consumerRecords = kafkaConsumer.poll(timeOutDuration);
    consumerRecords.forEach((record) -> {
        logger.info("Consumer Record Key is {} and the value is {} and the partion {}",
                record.key(), record.value(), record.partition());

        // Perform Duplicate Check
        // Invoke Some API
        // Persist the Record in DB
   });
}
```

Implement

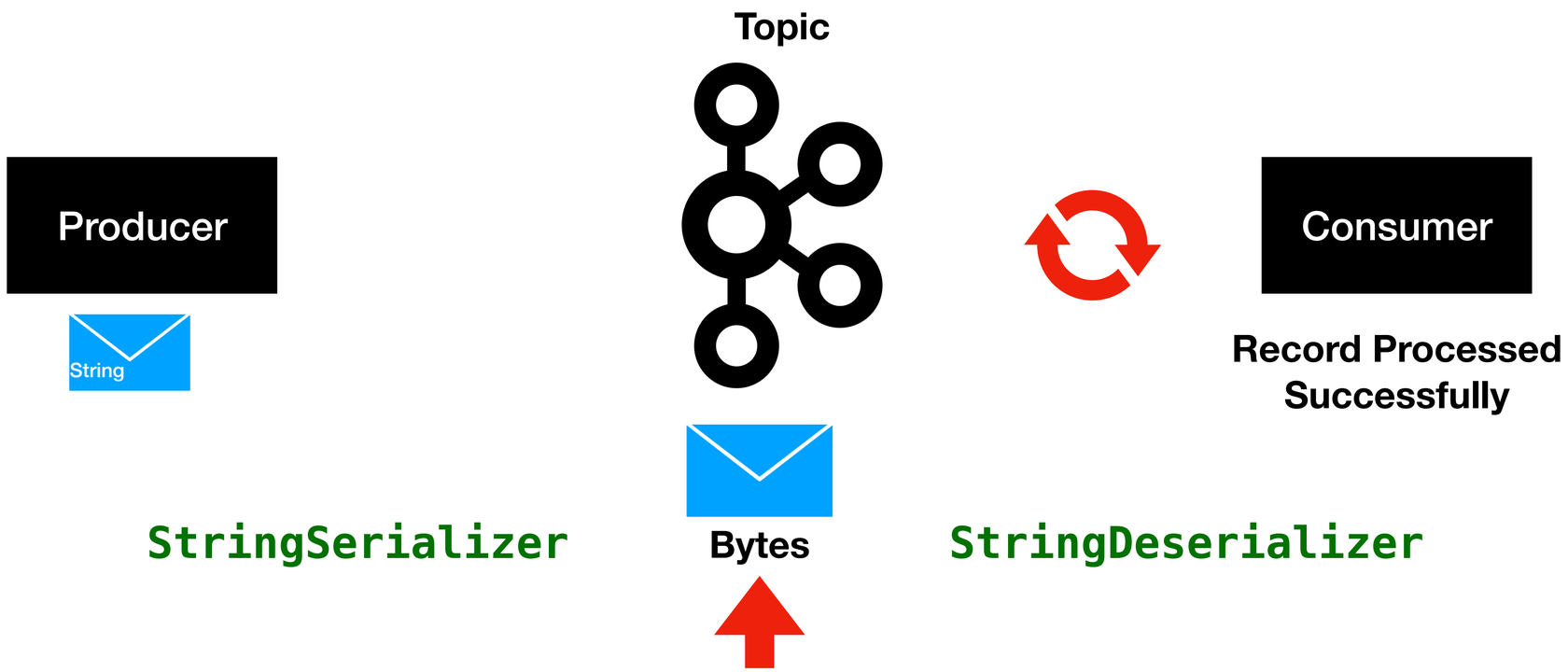
seek(TopicPartition partition, OffsetAndMetadata offsetAndMetadata)

Implement seek()



Custom Kafka Serializer & Deserializer

What do we have until now ?

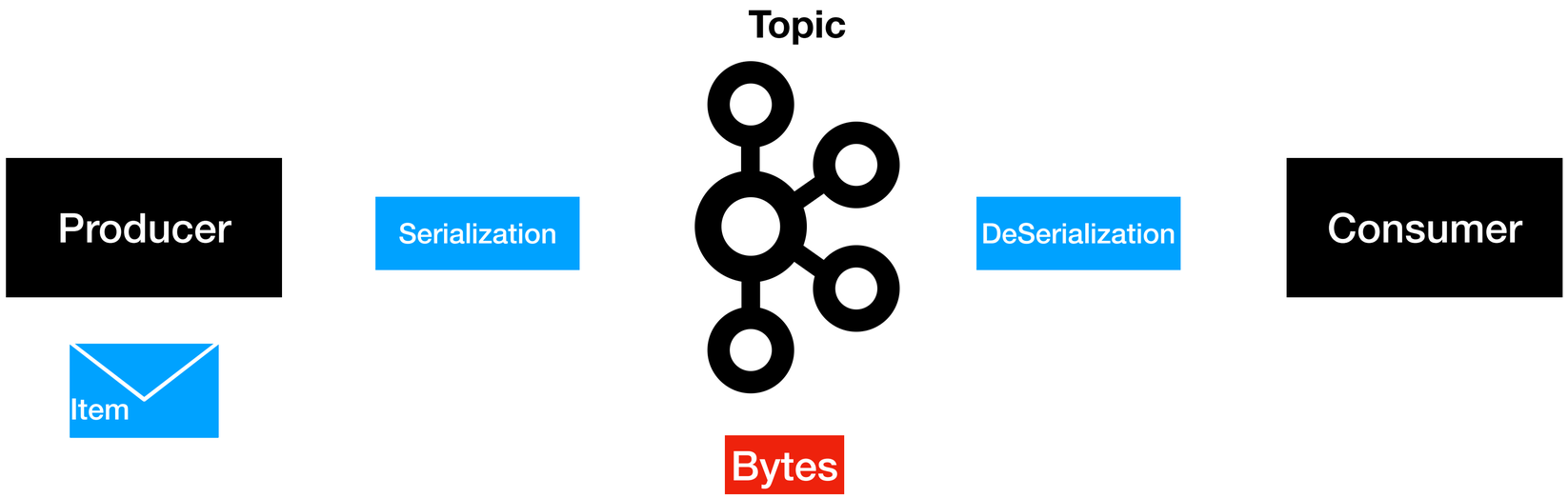


Use Kafka in Enterprise

- Retail
  - Item, Order , Cart etc.,
- Banking
  - Customer, Account , Transaction etc.,

Lets take Retail for example

```
public class Item implements Serializable{

    private static final long serialVersionUID = 1969906832571875737L;
    private Integer id;
    private String itemName;
    private Double price;
}
```



Serialize/DeSerialize Custom Objects

- Option 1 - **Build Custom Serializer/Deserializer**

- Option 2 - **Use Existing Serializer/Deserializer**
  - StringSerializer/Deserializer
  - IntegerSerializer/Deserializer

Build Custom Kafka Serializer

Build Custom Kafka Serializer

- Item Domain Class

- ItemSerializer

Build Custom Kafka DeSerializer

Build Custom Kafka DeSerializer

- Item Domain Class ✓

- ItemDeSerializer