# Spring for Apache Kafka


SoftUni Team

Technical Trainers

Software University

https://softuni.bg

# Have a Question?

Software University

sli.do

# #java-web

# Table of Contents




# Introduction to Kafka

# What is Kafka?

- **Kafka** is a distributed streaming platform
  - **High** Scalable (partition)
  - **Fault** Tolerant (replication)
  - Allow **high level** of **parallelism** and **decoupling** between data producers and data consumers

# What is Kafka?

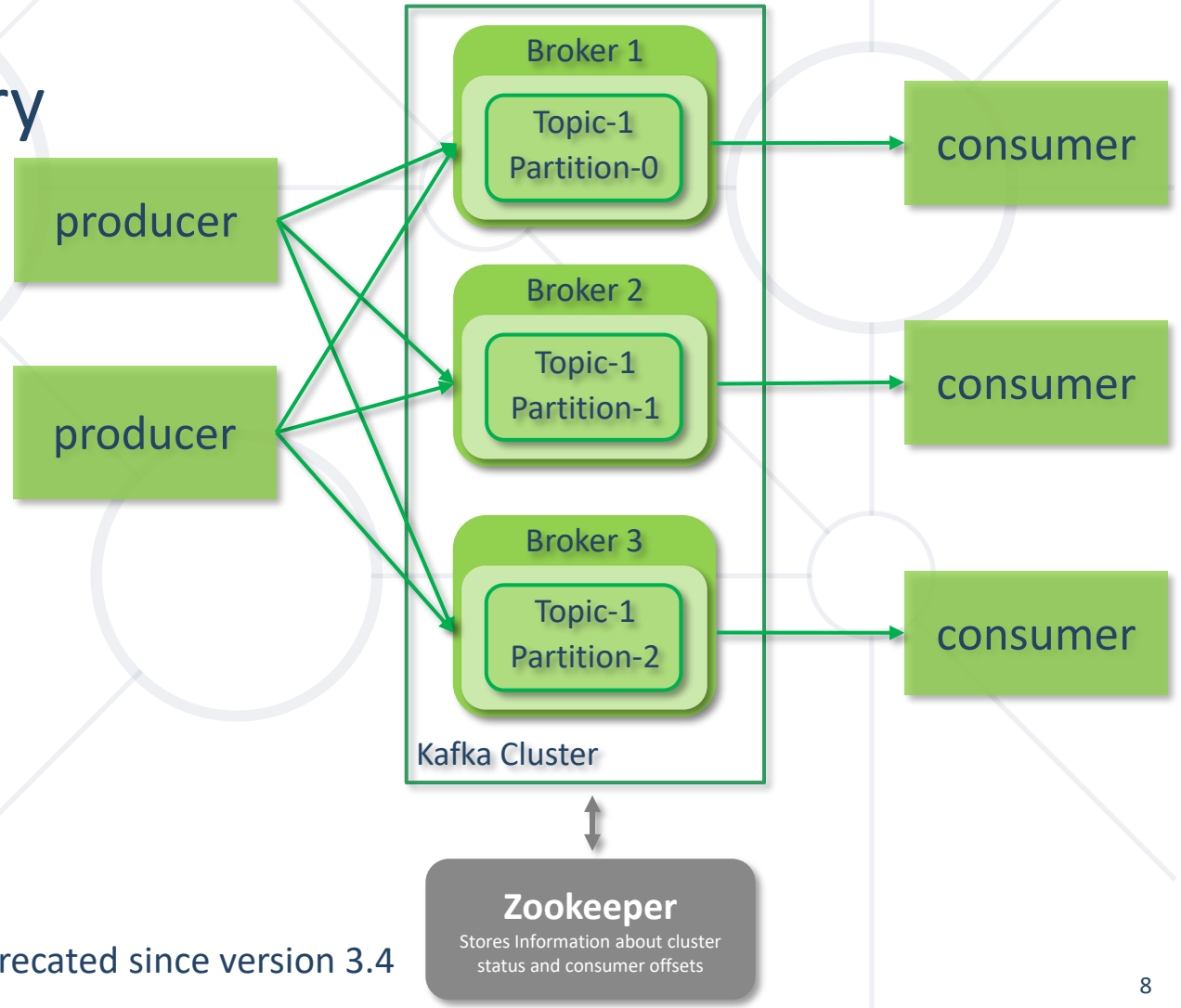
- **Purpose:**
  - **Apache** Kafka was originally developed by **LinkedIn**, open sourced in early 2011
  - Designed to **handle** large volumes of data streams and provide a **reliable**, low-latency, and highly available platform for real-time data processing.

# Kafka Messages

- Kafka operates on the principle of **message-based** data exchange
- Messages are the **fundamental units** of data in Kafka.
- Messages can represent **events**, **records**, or any data **payload**.
- They are typically in a **key-value** format but can be structured in various ways (e.g., JSON, Avro, or binary)

# Kafka Basic Concepts

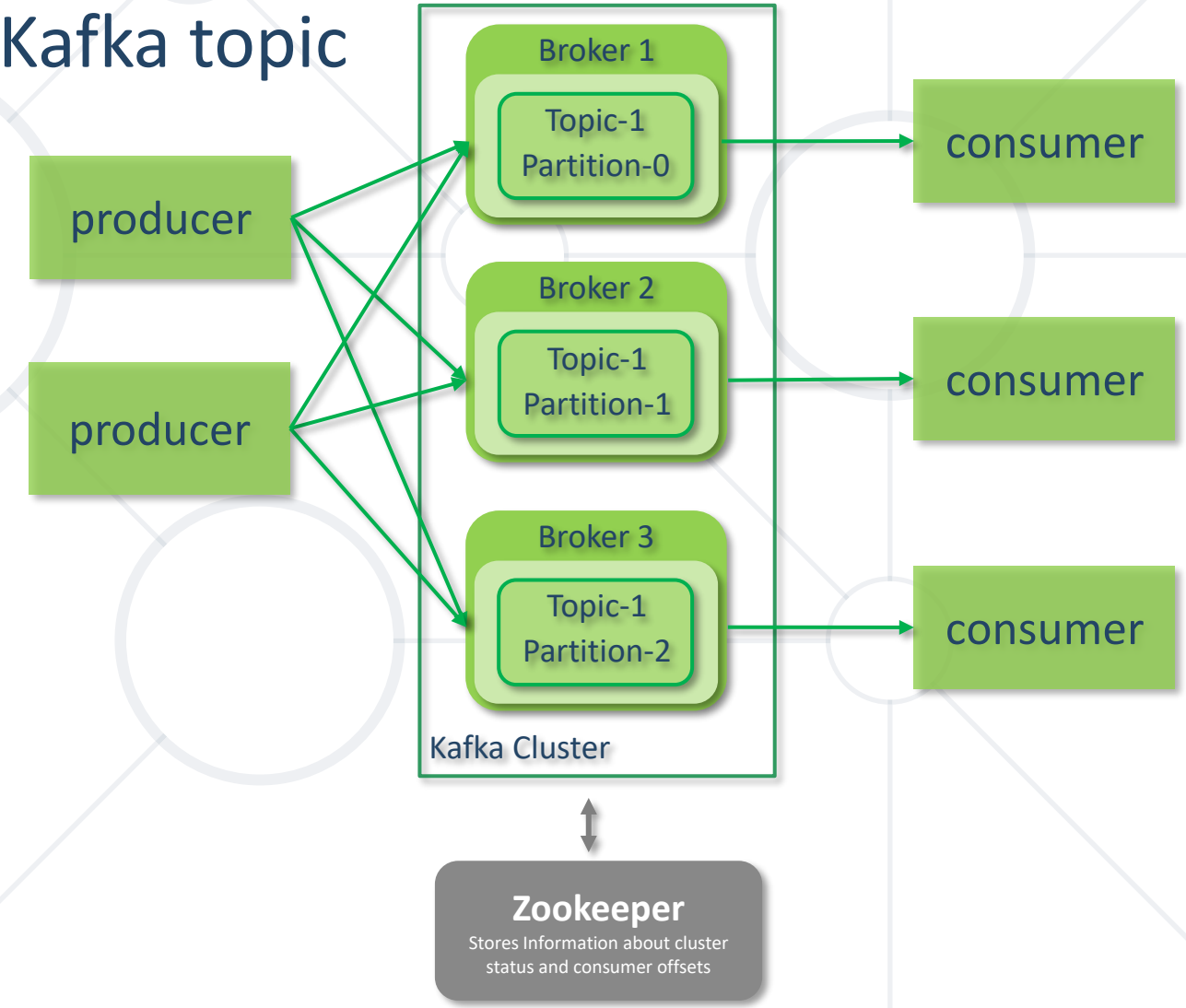
- Broker: **Kafka node** on the cluster
- Topics: **Stream** of records category
  - Multiple **writers** and **readers**
  - **Partitioned**
  - **Replicated**
- Consumer: **pulls messages off** of a Kafka topic




*Deprecated since version 3.4

# Kafka Basic Concepts

- Producer: **push** messages into a Kafka topic
- Data Retention:
  - Based on **time** or **size**
- Zookeeper: Stores Kafka **Metadata**




*Deprecated since version 3.4

# Topics

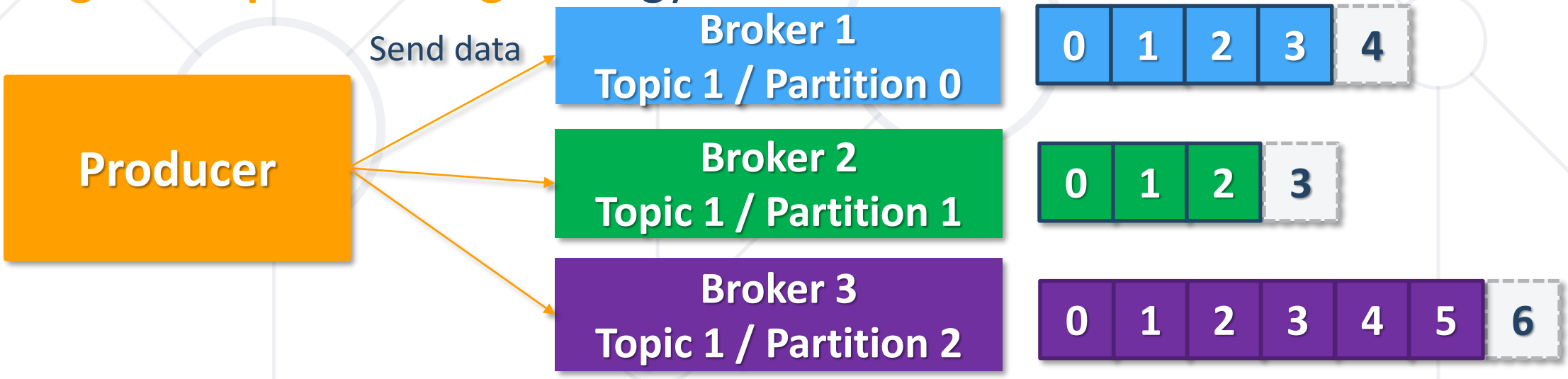
- A **topic** is an ordered log of events
- Topics in Kafka represent the **subject** or **category** to which messages are published
- Topics are identified by their **names**, which are strings
- Kafka allows **multiple** producers to write data to the same topic **concurrently**

# Topics

- Topics are the central concept in Kafka that **decouples** producers and consumers
- A consumer **pulls messages off** of a Kafka topic while producers **push messages into** a Kafka topic
- A topic can have **many** producers and many consumers

# Producers

- Entities responsible for **sending** messages to Kafka topics
- They **publish** data to Kafka topics **based** on the topic's name
- Producers can be **any** software component, application, device, or system that **generates** data
- Producers **decide** to which topic and partition a message is sent, using a **configurable partitioning** strategy

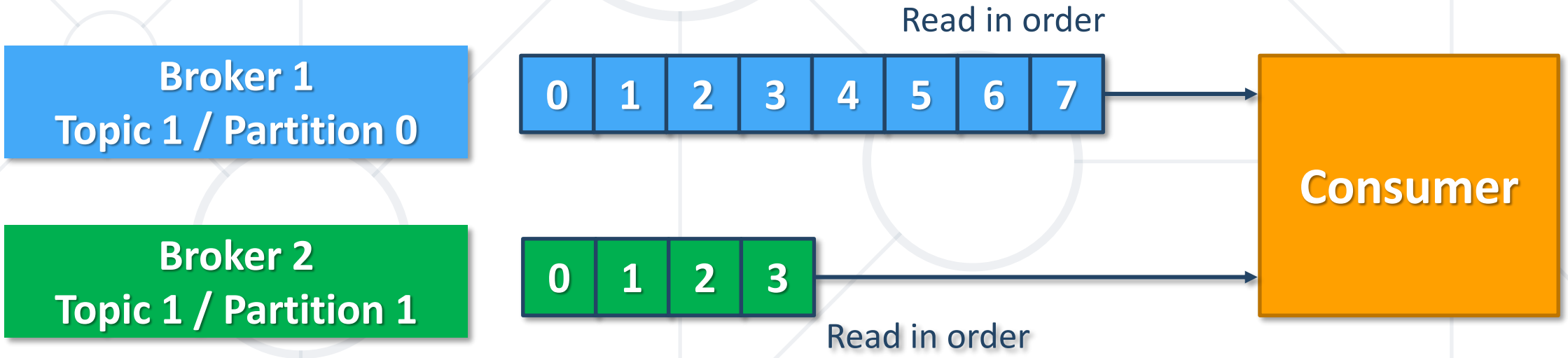


# Consumers

- Entities that **subscribe** to Kafka topics to **consume** and process messages
- They can be:
  - **Applications**
  - **Services**
  - **Components** that need access to the data published in Kafka

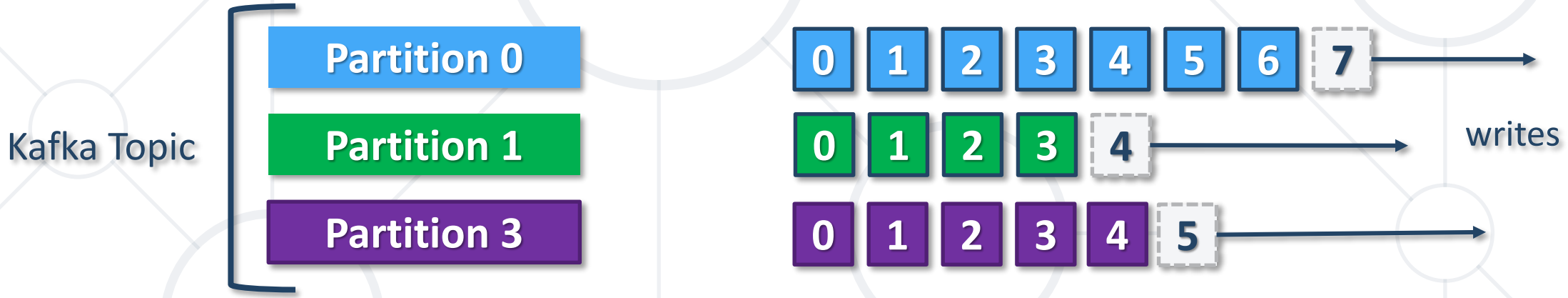
# Consumers

- Consumers **read** messages from Kafka partitions in **real-time**
- Kafka **ensures** that each message is delivered to **all** consumers in a topic, enabling parallel processing



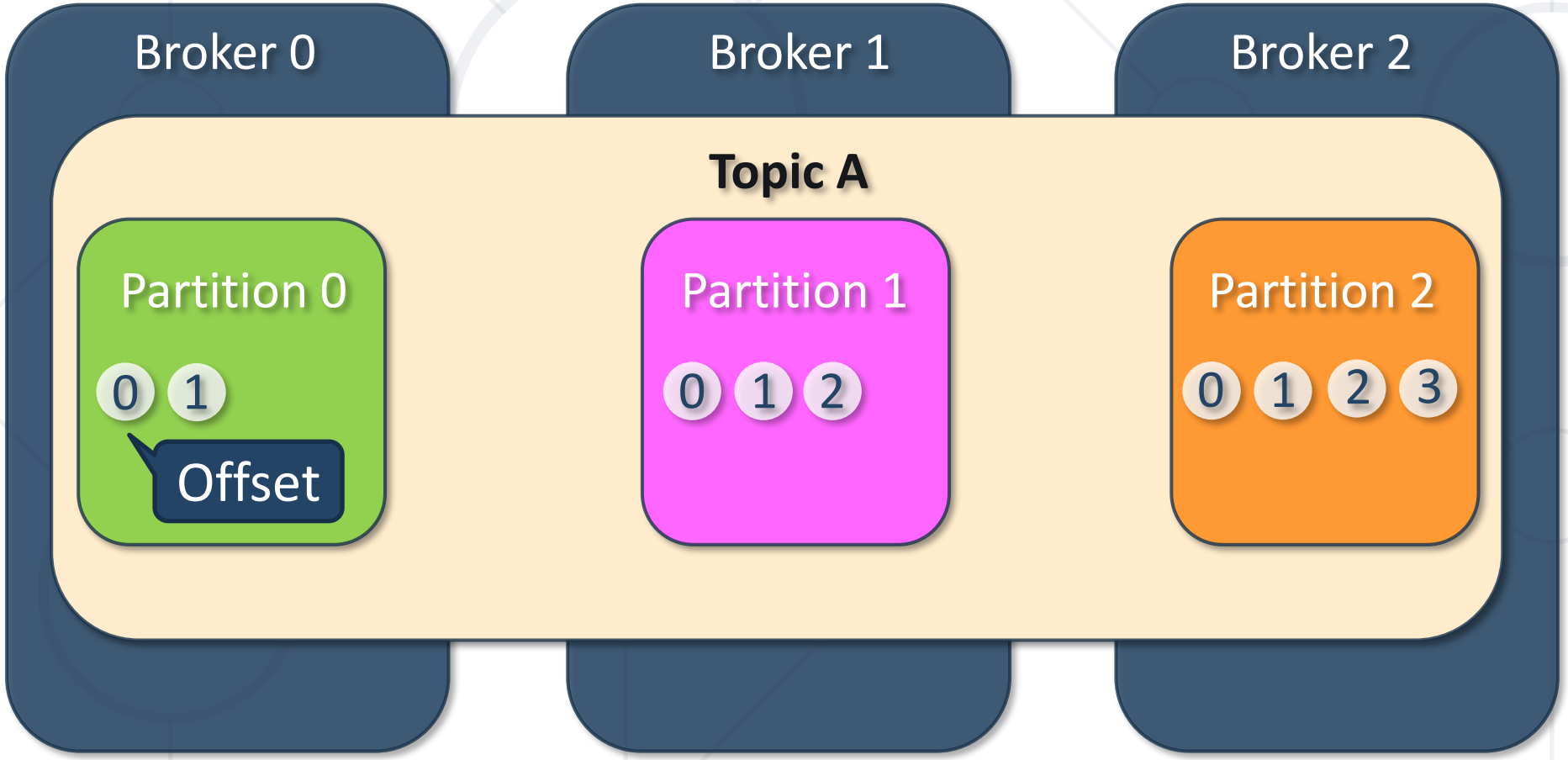
# Partitions

- Topics are split in **partitions**
  - Each partition is **ordered**
  - Each message within a partition gets an incremental id, called **offset**




- Partitions are the way that Kafka provides **scalability**
- Partitions **enable** Kafka to distribute the data and processing load across multiple brokers and consumers

# Producing messages - distribution across partitions



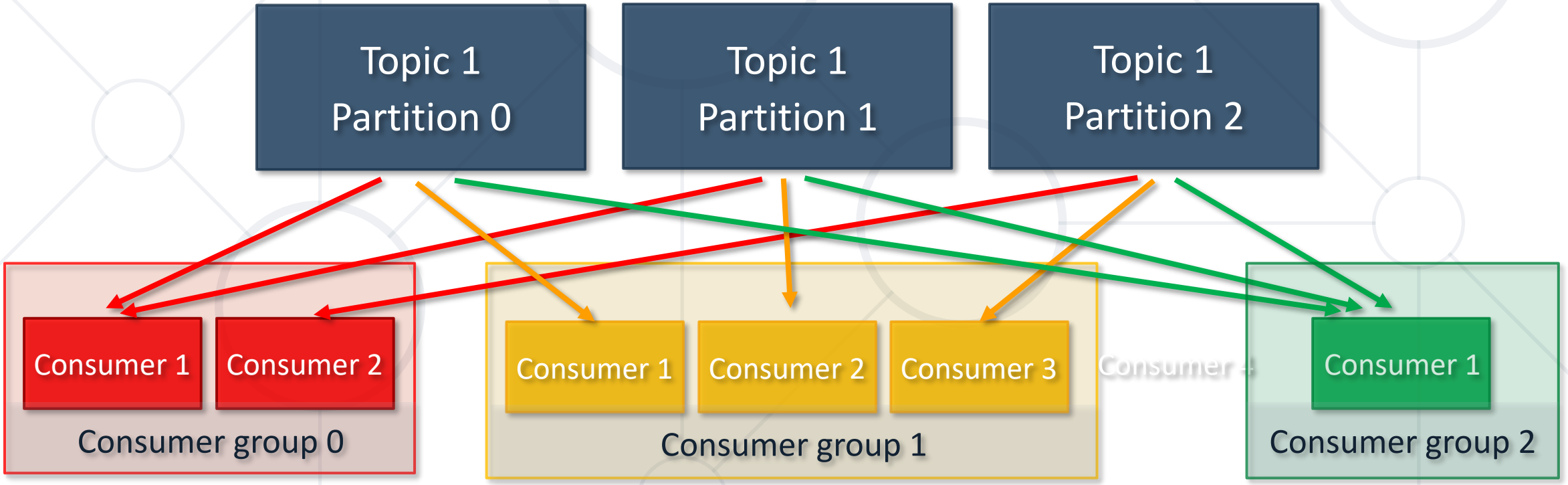
# Consuming messages - offsets, order of consumption



- In **Kafka**, every message within a partition is assigned a unique identifier known as an **offset**
  - Offsets are used to **keep** track of the consumer's **progress** in a partition
- Within a single Kafka partition, messages are **strictly** ordered based on their offsets
  - A consumer **reading** from a partition processes messages in the order they were **written**.
  - However, across different partitions of the same topic, **there is no guarantee** of global message order

# Consumer groups

- Kafka consumers can be organized into consumer **groups**
  - Each consumer within a group **reads** from **exclusive** partitions



# Consumer offsets

- In order to **"checkpoint"** how far a consumer has been reading into a topic partition, the consumer will regularly **commit** the latest processed message, also known as **consumer offset**
- If a consumer process **dies**, it will be able to read back from where it left off thanks to consumer **offsets**


# Spring for Apache Kafka

# Setup

- Add the *spring-kafka* dependency to our *pom.xml/build.gradle*:

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
```

```
implementation 'org.springframework.kafka:spring-kafka'
```

- To create a single custom topic with non-default settings, we can use the NewTopic:

```
@Bean
public NewTopic topic1() {
    return TopicBuilder.name("thing1")
            .partitions(10)
            .replicas(3)
            .compact()
            .build();
}
```

# Producing messages

- First we need to configure a **ProducerFactory**
- Then we need a **KafkaTemplate**, which wraps a Producer instance

# Producing messages

```
@Configuration
public class KafkaProducerConfig {

    @Bean
    public ProducerFactory<String, String> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, String> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

# Publishing messages

- We can send messages using the **KafkaTemplate** class:

```
@Autowired
private KafkaTemplate<String, String> kafkaTemplate;

public void sendMessage(String msg) {
    kafkaTemplate.send(topicName, msg);
}
```

# Consuming messages

- For consuming messages, we need to configure:
  - **ConsumerFactory**

```
@EnableKafka
@Configuration
public class KafkaConsumerConfig {

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(props);
    }
. . .
```

# Consuming messages

- And **KafkaListenerContainerFactory**

```
. . .

 @Bean
    public ConcurrentKafkaListenerContainerFactory<String, String>
      kafkaListenerContainerFactory() {

        ConcurrentKafkaListenerContainerFactory<String, String> factory =
          new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}
```

# Consuming messages

- **POJO-based** consumers can be configured using **@KafkaListener** annotation

```
@KafkaListener(topics = "topicName", groupId = "foo")
public void listenGroupFoo(String message) {
    System.out.println("Received Message in group foo: " + message);
}
```

- We can implement **multiple** listeners for a topic

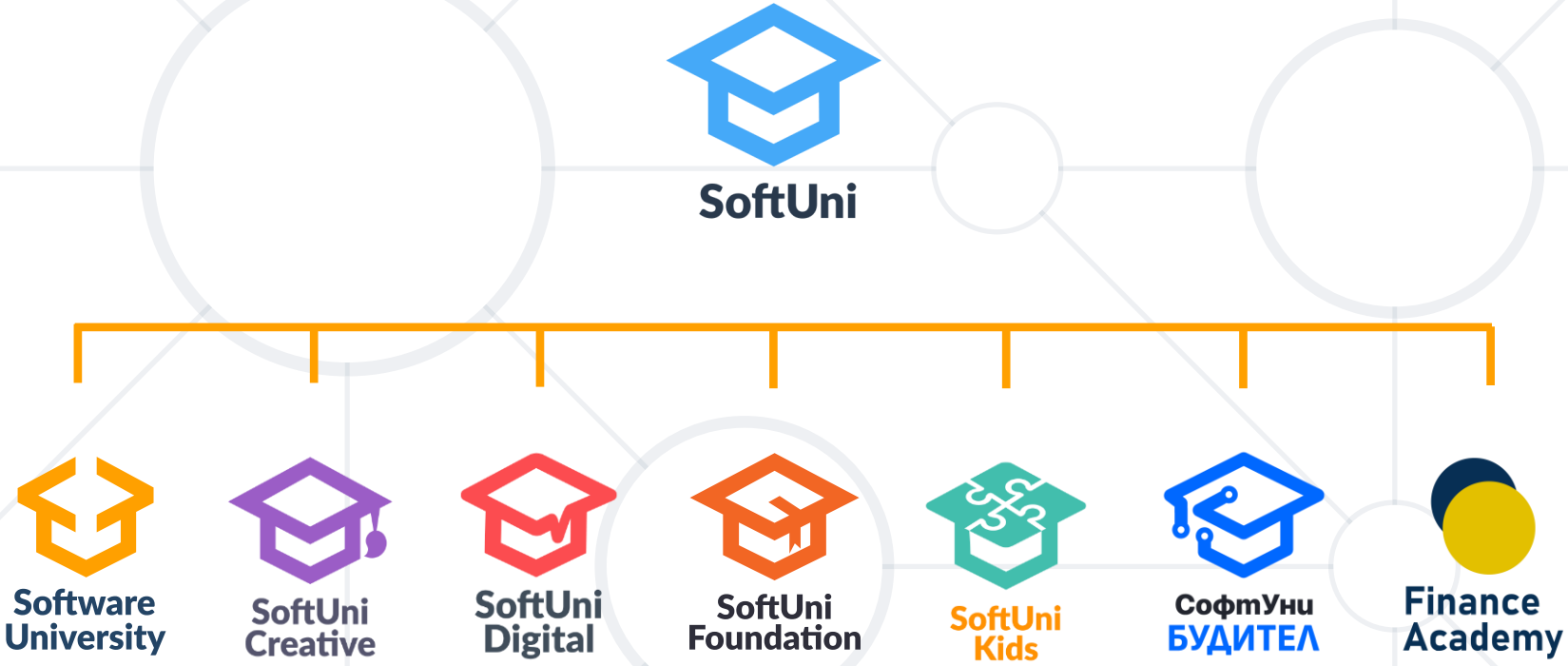
```
@KafkaListener(topics = "topic1, topic2", groupId = "foo")
```

# Summary



- Introduction to Kafka
  - What is Kafka?
  - Topics, partitions
  - Producing messages - distribution across partitions
- Spring for Apache Kafka
  - Connecting to Kafka
  - Producing messages
  - Consuming messages

# Questions?



# SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

# Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

# License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg