Kafka Fundamentals

An Introduction to Apache Kafka for Java Developers

Luis Castillo

04-05-2023

Agenda

1. Overview of Kafka
   - Intro
   - Use cases
   - Architecture and componentes
2. Kafka Fundamentals
   - Topics
   - Partitions and offsets
   - Producers
   - Messages
   - Consumers
   - Consumer Groups
   - Delivery Semantics
3. Kafka APIs for Java
   - Producer API
   - Consumer API
   - Streams API
   - Connect API

Section 1

Overview

Why Apache Kafka?

- Created by LinkedIn, now Open-Source Project mainly maintained by Confluent, IBM, Cloudera
- Distributed, resilient architecture, fault tolerant
- Horizontal scalability:
  - Can scale to 100s of brokers
  - Can scale to millions of messages per second
- High performance (latency less than 10ms) - real time
- Used by the 2000+ firms, 80% of the Fortune 100

Uses Cases

- Messaging System
- Activity Tracking
- Gather metrics form many different locations
- Application Logs gathering
- Stream processing (with the Kafka Streams API)
- De-coupling of system dependencies
- Integration with Spark, Flink, Storm, Hadoop, and my other Big Data technologies
- Micro-services pub/sub

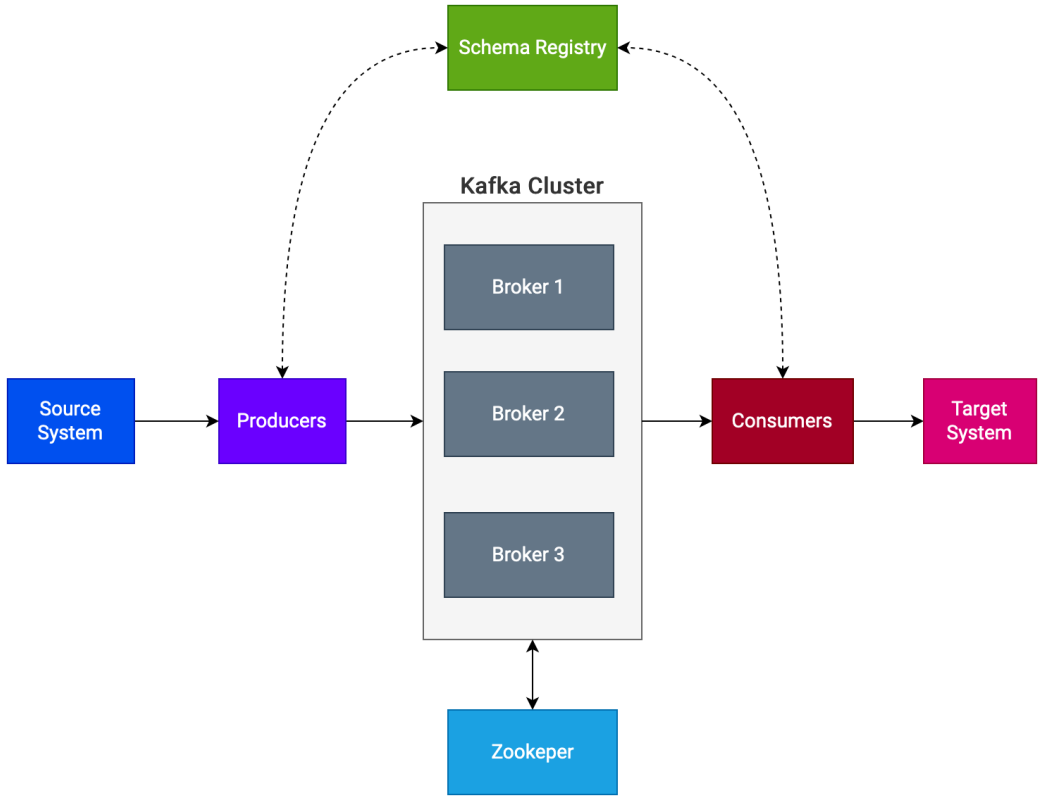
Real world examples

- **Netflix** uses Kafka to apply recommendations in real-time while you are watching TV
- **Uber** uses Kafka to gather, user, driver and trip data in real-rime to compute and forecast demand, and compute surge pricing in real-rime
- **LinkedIn** uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real time

Note: Kafka is only used as a transportation mechanism.

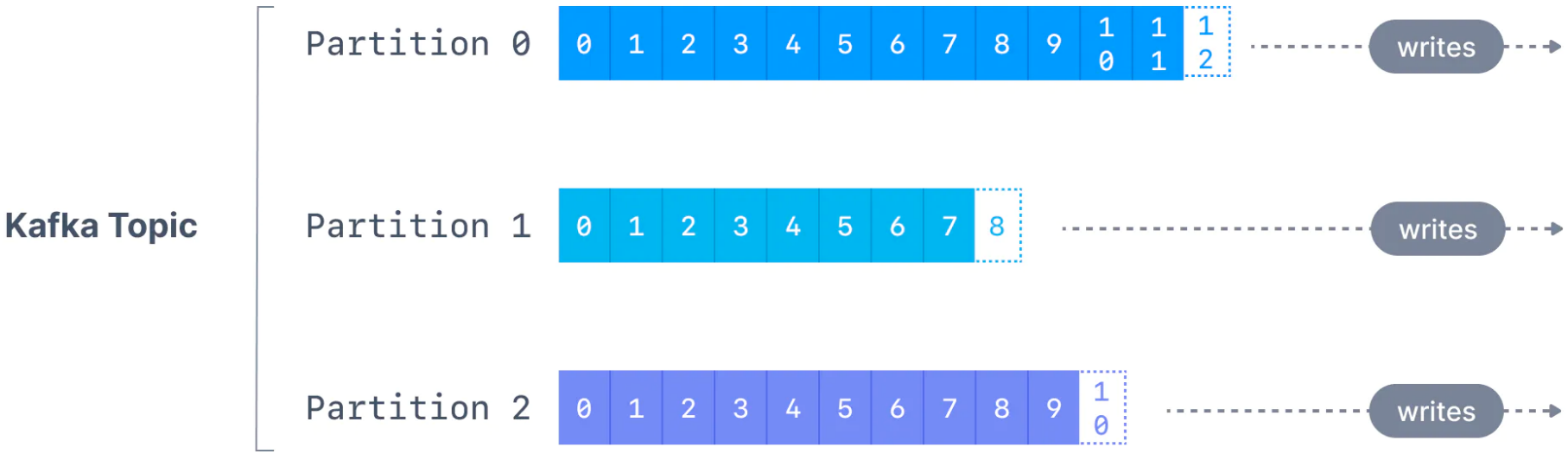
Architecture and Components

- **Broker**: Kafka server that stores and manages messages
- **Producer**: Application that sends messages to Kafka
- **Consumer**: Application that reads messages from Kafka
- **Zookeeper**: Coordination service for Kafka cluster
- **Schema Registry**: Stores and manages Avro schema versions



Kafka concepts

- **Topic**: A stream of records, categorized by name
- **Partition**: A single, ordered, immutable sequence of records in a topic
- **Offset**: Unique identifier for each record within a partition



https://www.conduktor.io/kafka/kafka-topics/

APIs for Java

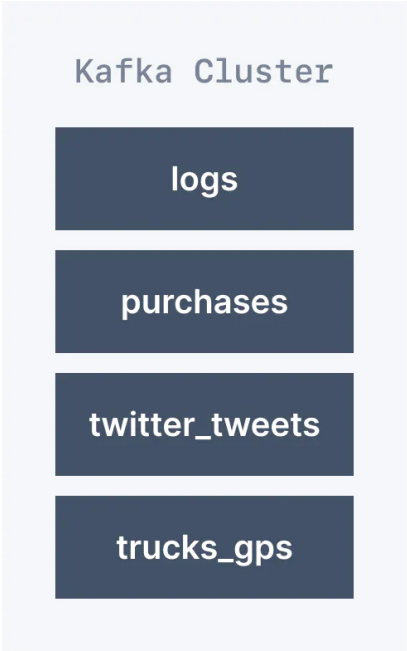
- **Producer API**: Send messages to Kafka
- **Consumer API**: Read messages from Kafka
- **Streams API**: Process streams of records in real-time
- **Connect API**: Integrate Kafka with other systems

Section 2

Kafka Basics

Topics

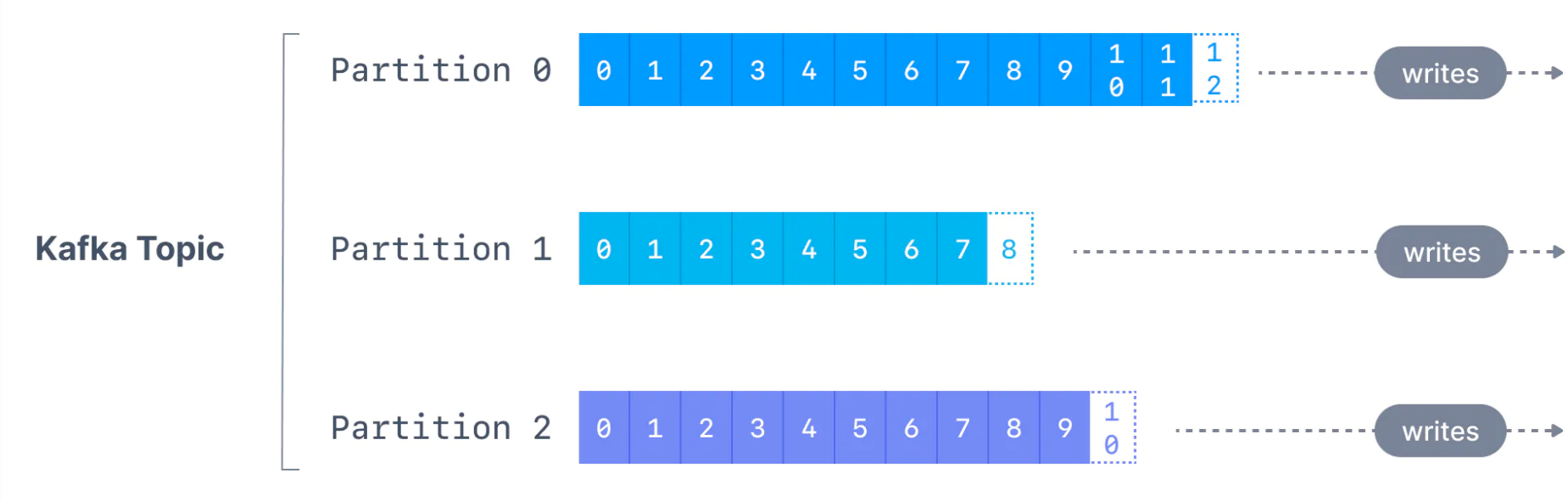
- Kafka uses the concept of topic to organize related messages
- A topic is identified by its name
- Topics can contain any kind of message in any format
- Sequence of messages in a topic is called a **data stream**



https://www.conduktor.io/kafka/kafka-topics/

Partitions and offsets

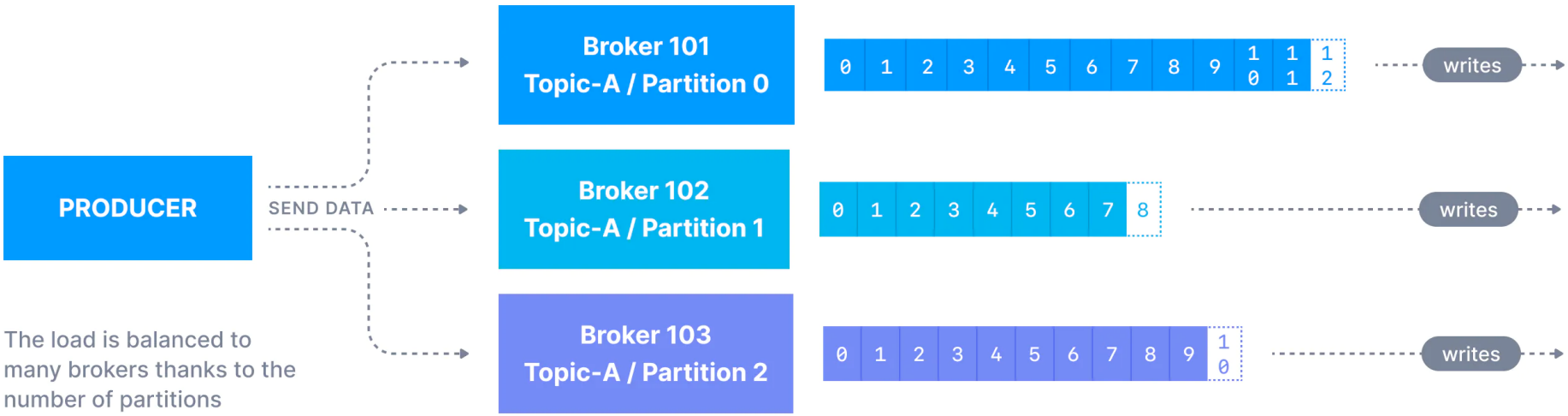
- Topics are split in **partitions**
  - Messages within each partition are ordered
  - Each message within a partition gets an incremental id, called offset
  - Partitions enable parallelism and provide fault tolerance
- An **offset** is a unique id for each record within a partition
- Kafka topics are **immutable**: once data is written to a partition, it cannot be changed.



https://www.conduktor.io/kafka/kafka-topics/

Producers

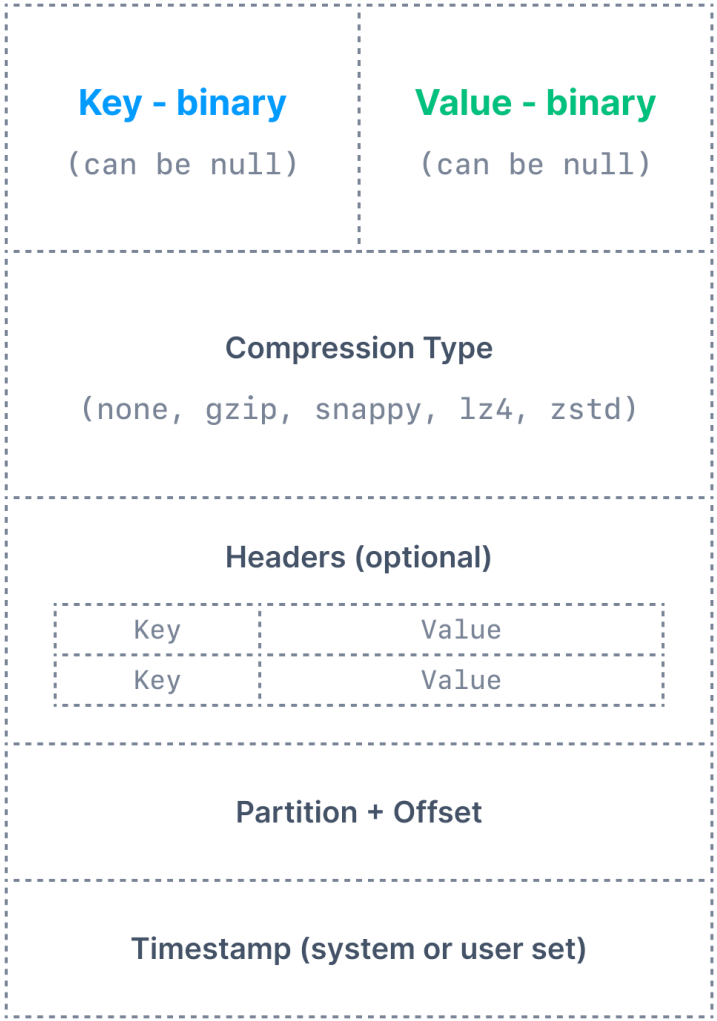
- Producers write data to topics
- Producers know (in advance) to which partition to write to
- In case of Broker failures, Producers will automatically recover



https://www.conduktor.io/kafka/kafka-topics/

Messages anatomy

Kafka Message Created by the producer



https://www.conduktor.io/kafka/kafka-producers/

Message Serializer

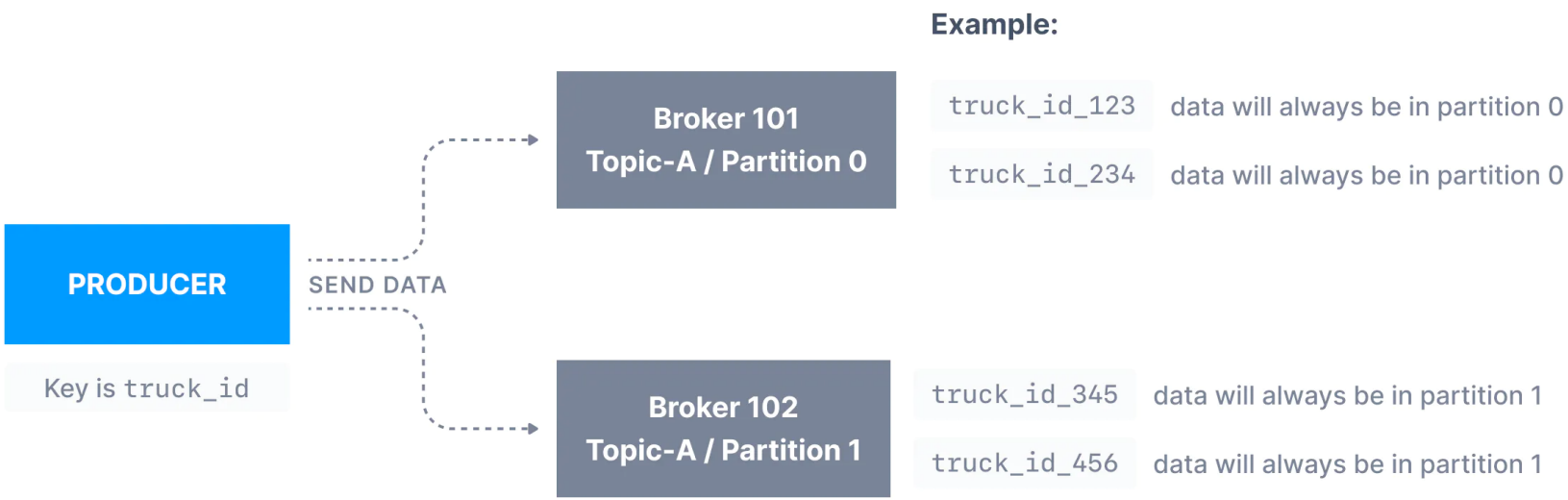
- Kafka only accept bytes as an input from producers
- They are used on the value and the key
- Common Serializers
  - String (incl. JSON)
  - int, float
  - **AVRO**
  - Protobuf

https://www.conduktor.io/kafka/kafka-producers/

Message Key Hashing

- Key Hashing is the process of determining the mapping of a key to a partition
- By default, in Kafka partitioner the keys are hashed using the **murmur2 algorithm**

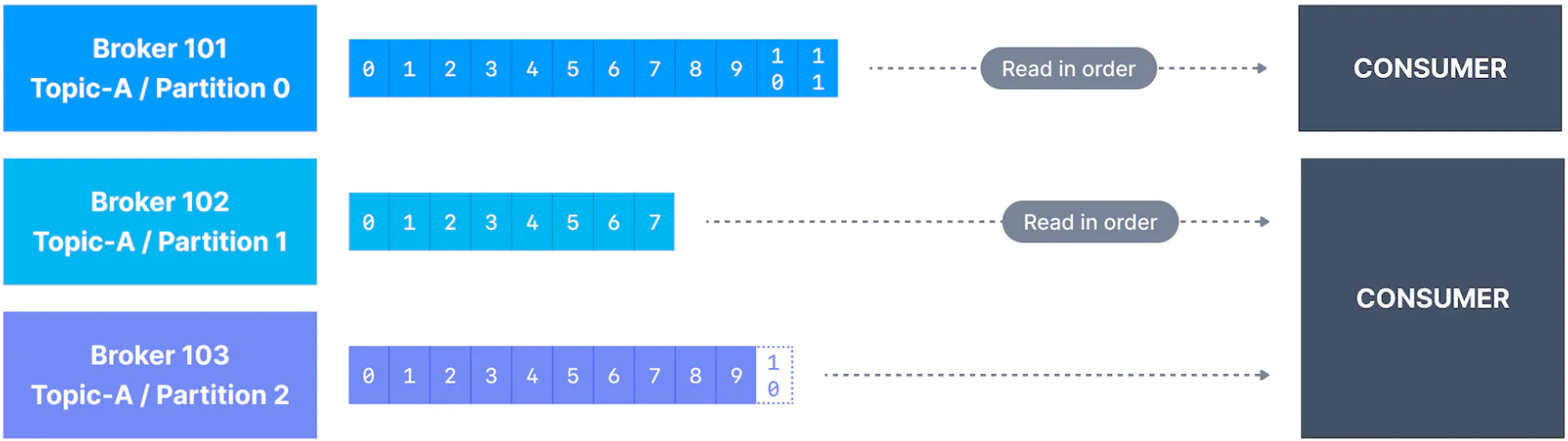
```
Math.abs(Utils.murmur2(keyBytes) % (numPartitions -1))
```



https://www.conduktor.io/kafka/kafka-producers/

Consumers

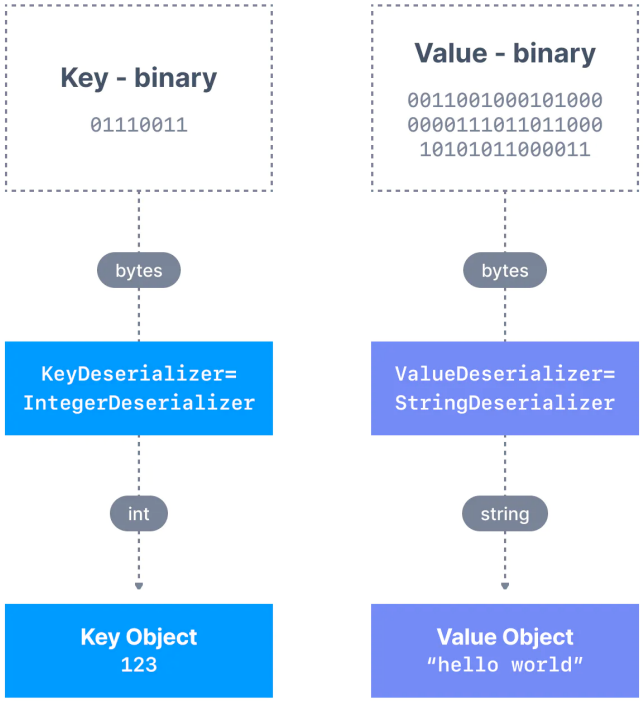
- Consumers read data from a topic (pull mode)
- Consumers automatically know which broker to read from
- In case of broker failures, consumers know how to recover
- Data is read in order from low to high offset **within each partitions**



https://www.conduktor.io/kafka/kafka-consumers/

Consumer Deserializer

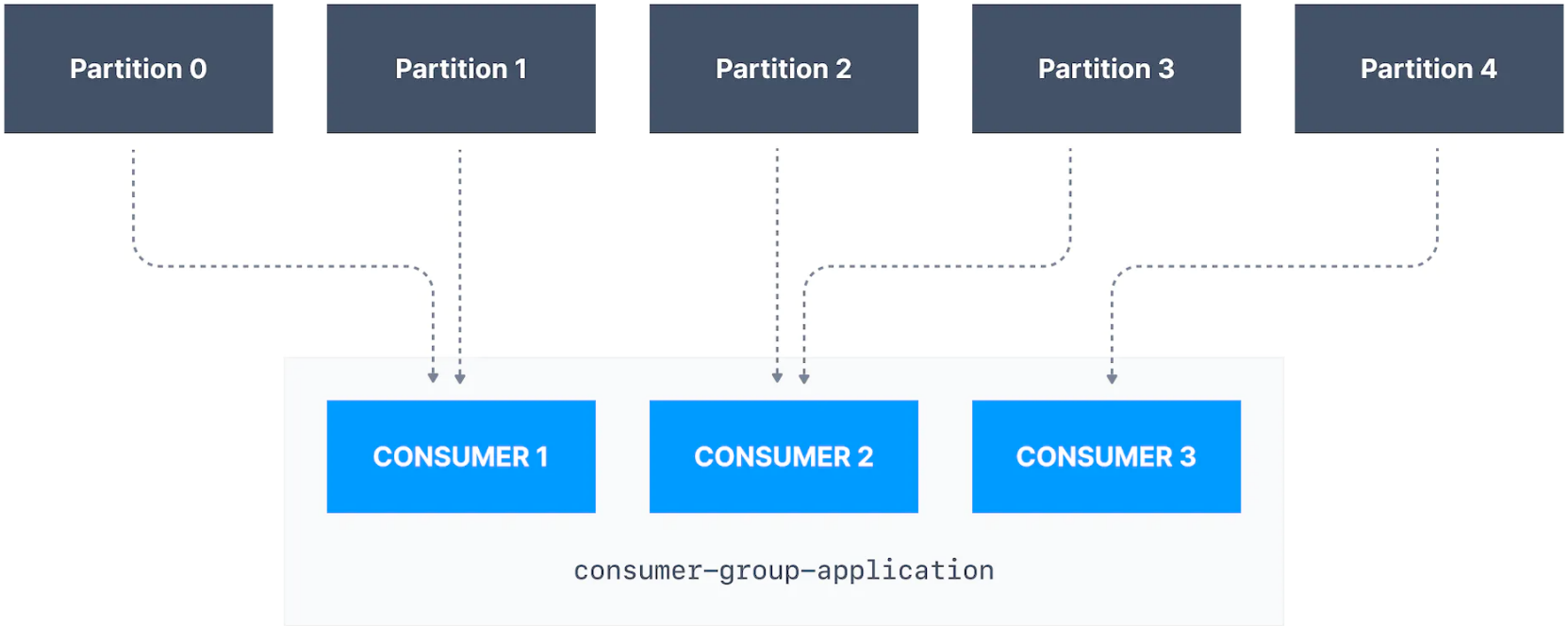
- Deserialize indicates how to transfrom bytes into objects
- They are used on the value and the key of the message
- Common Deserializers:
  - String (incl. JSON)
  - int, float
  - AVRO
  - Protobuf



https://www.conduktor.io/kafka/kafka-consumers/

Consumer Groups

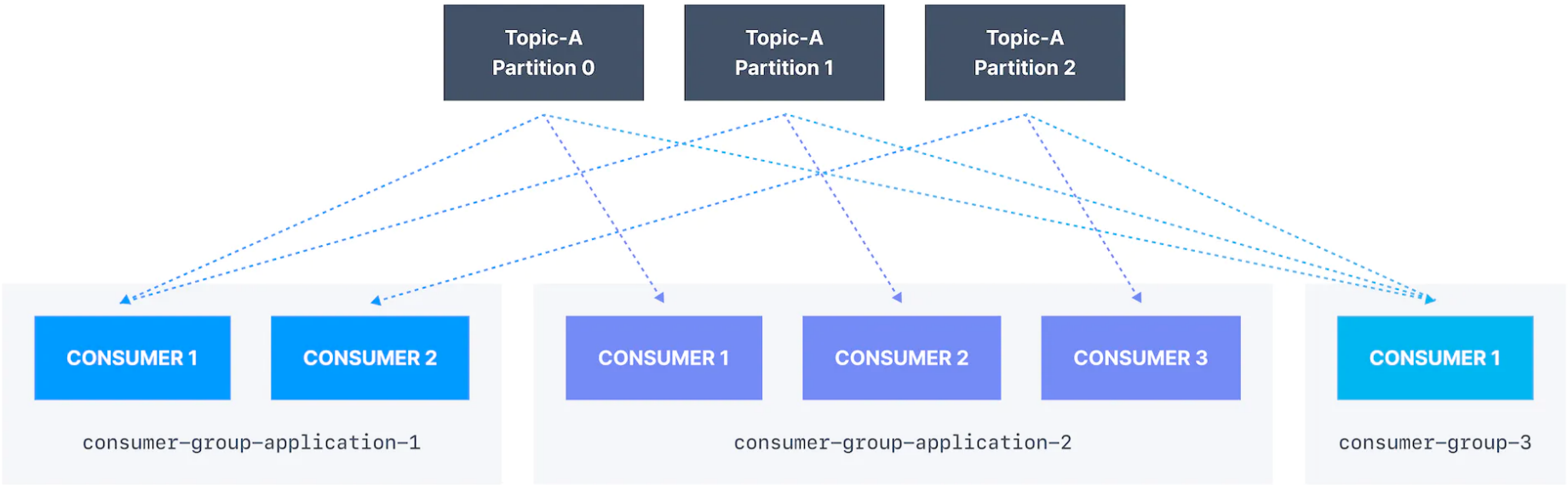
- Group of consumers working together to read messages from a topic
- Each consumer in a group reads messages from a unique partition
- Enables parallelism and load balancing



https://www.conduktor.io/kafka/kafka-consumer-groups-and-consumer-offsets/

Multiple groups on one topic

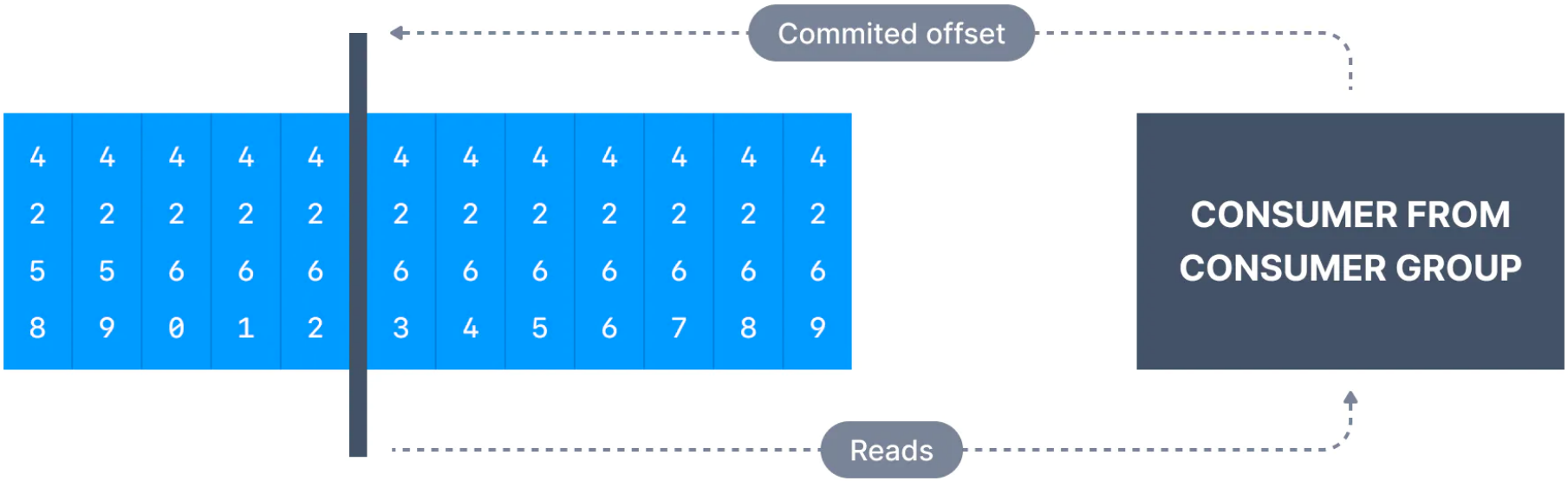
- In Kafka it is acceptable to have multiple consumer groups on the same topic



https://www.conduktor.io/kafka/kafka-consumer-groups-and-consumer-offsets/

Consumer Offsets

- Kafka stores offsets at wich consumer group has been reading
- The offsets committed are in Kafka topic named `__consumer_offsets` -When a consumer in a group has processed data received from Kafka, it should be periodically committing the offsets
- If a consumer dies, it will be able to read back from where it left off



https://www.conduktor.io/kafka/kafka-consumer-groups-and-consumer-offsets/

Delivery Semantics

- Defines the guarantee of message delivery in Kafka
- By default, Java consumers will automatically commit offsets (at least once)
- There are 3 delivery semantics if you choose to commit manually

Semantics	Characteristics
At Least Once	Possibility of duplicate messages
At Most Once	Possibility of message loss
Exactly Once	No duplicates, no message loss

Section 3

Kafka APIs

Key Feature of Producer API

- **Asynchronous** message sending
- **Batching** of messages to optimize throughput
- **Serialization** support for various data formats

Key Features of Consumer API

- **Offset** management and committing
- **Deserialization** support for various data formats
- **Rebalancing** for consumer groups

Key Features of Streams API

- **Stateful** stream processing
- **Windowing** support for time-based operations
- **Joining** streams to create complex processing topologies

Key Features of Connect API

- **Source Connectors**: Import data from external systems into Kafka
- **Sink Connectors**: Export data from Kafka to external systems
- **Configuration-based** integration with minimal coding

Questions and Discussion

- Open the floor for questions and discussions
- Share experiences, challenges, and best practices

Thank You

- Thank you for your time and attention
- Good luck with your kafka journey!

References I

1. Maarek, S. “Apache kafka series,” 2018.

2. “What is apache kafka?” n.d.

3. “Apache kafka: An introduction,” n.d.

4. “Introduction to apache kafka,” n.d.

5. “What is kafka,” n.d.

6. “Welcome to kafka! We’re glad you’re here,” n.d.

7. “Fundamentals for apache kafka,” n.d.

8. “Kafka internals fundamentals,” n.d.

References II

9. "Event streaming platform," n.d.

10. "Creating your first apache kafka producer application," n.d.

11. "KafkaProducer JavaDoc," n.d.

12. "How to implement kafka producer," n.d.

13. "Creating your first apache kafka producer application with confluent," n.d.

14. "Kafka-python documentation," n.d.