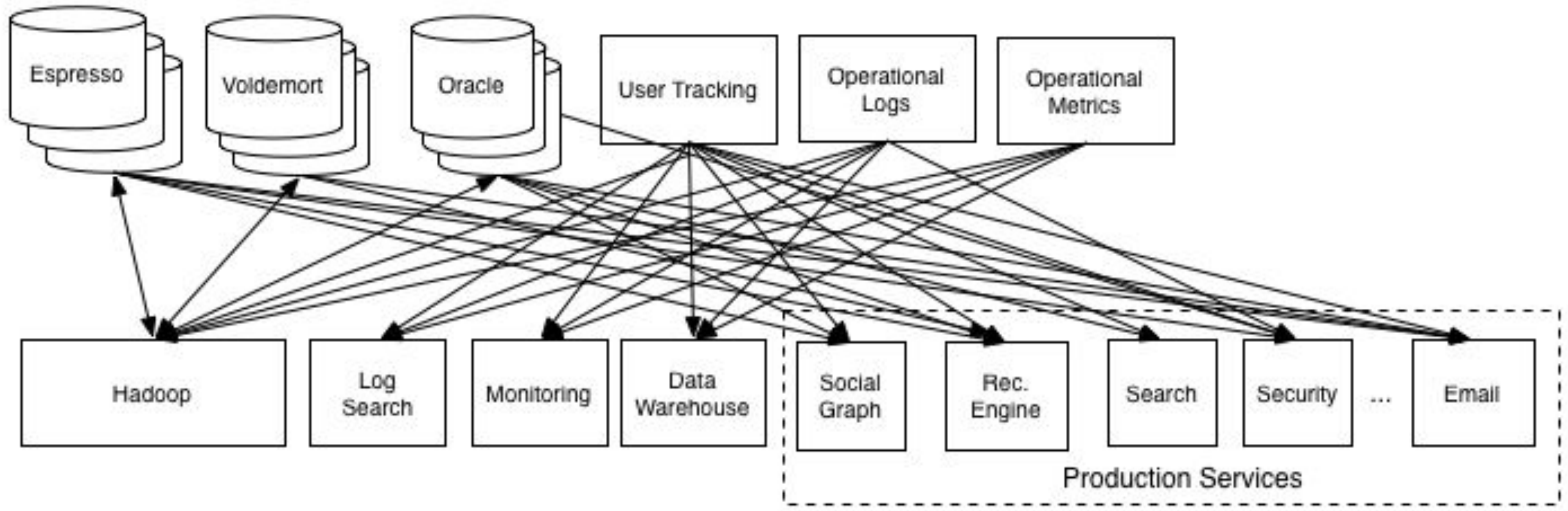
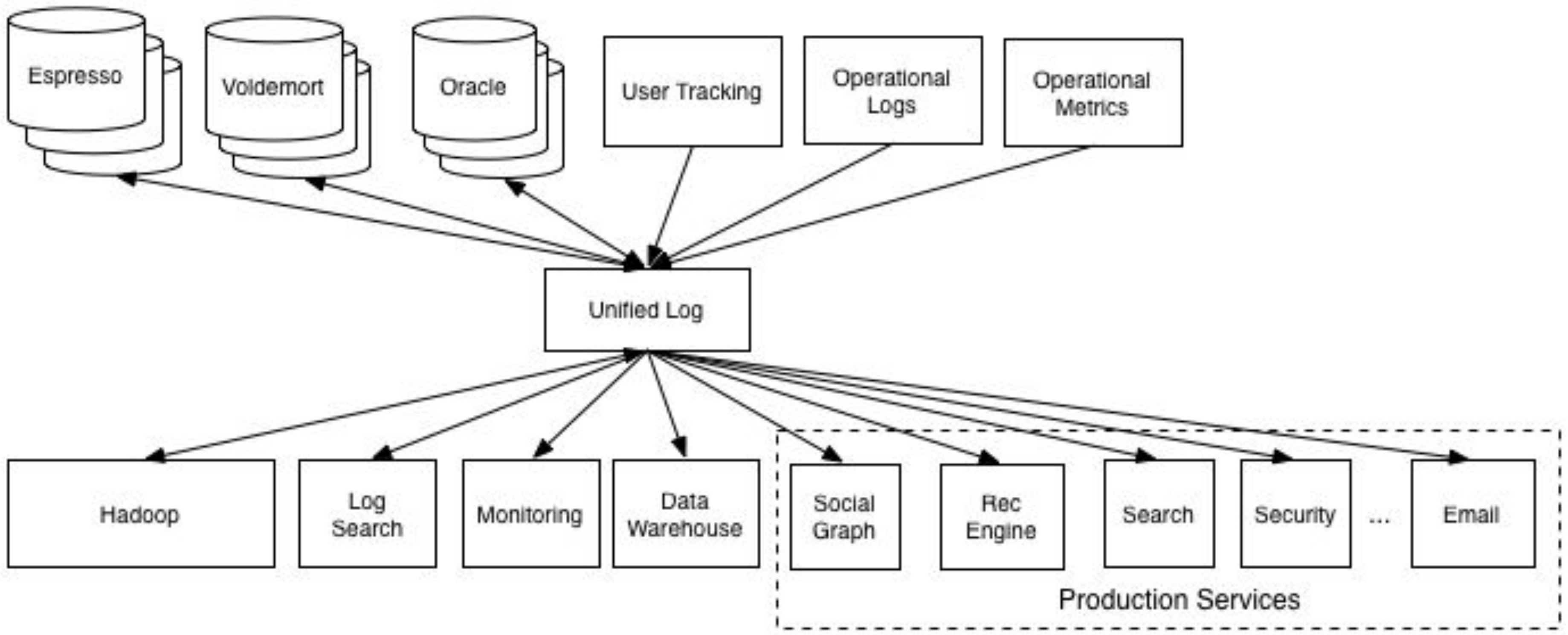
Why Kafka ?

Kafka Motivation



[1]https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin

Kafka at Linkedin



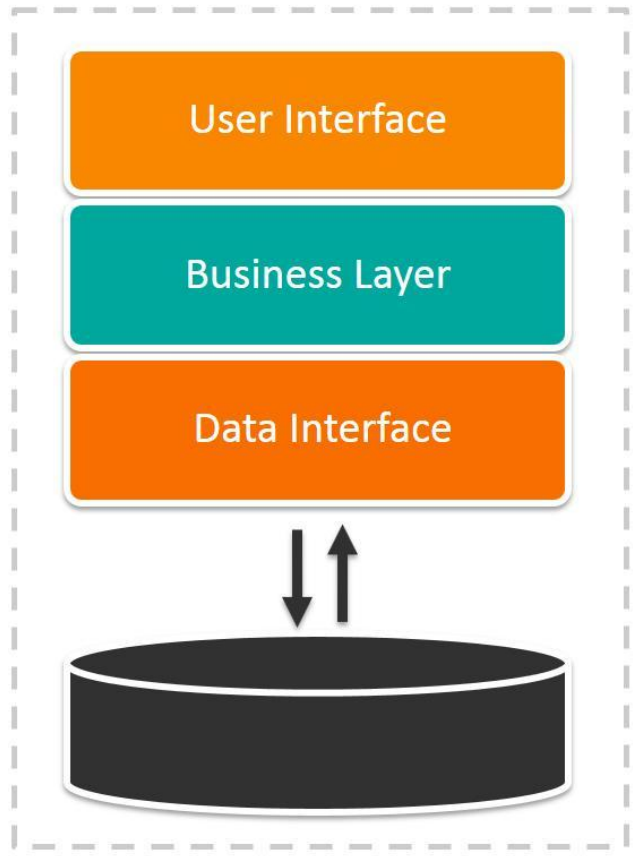
[1]https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin

Kafka at Linkedin

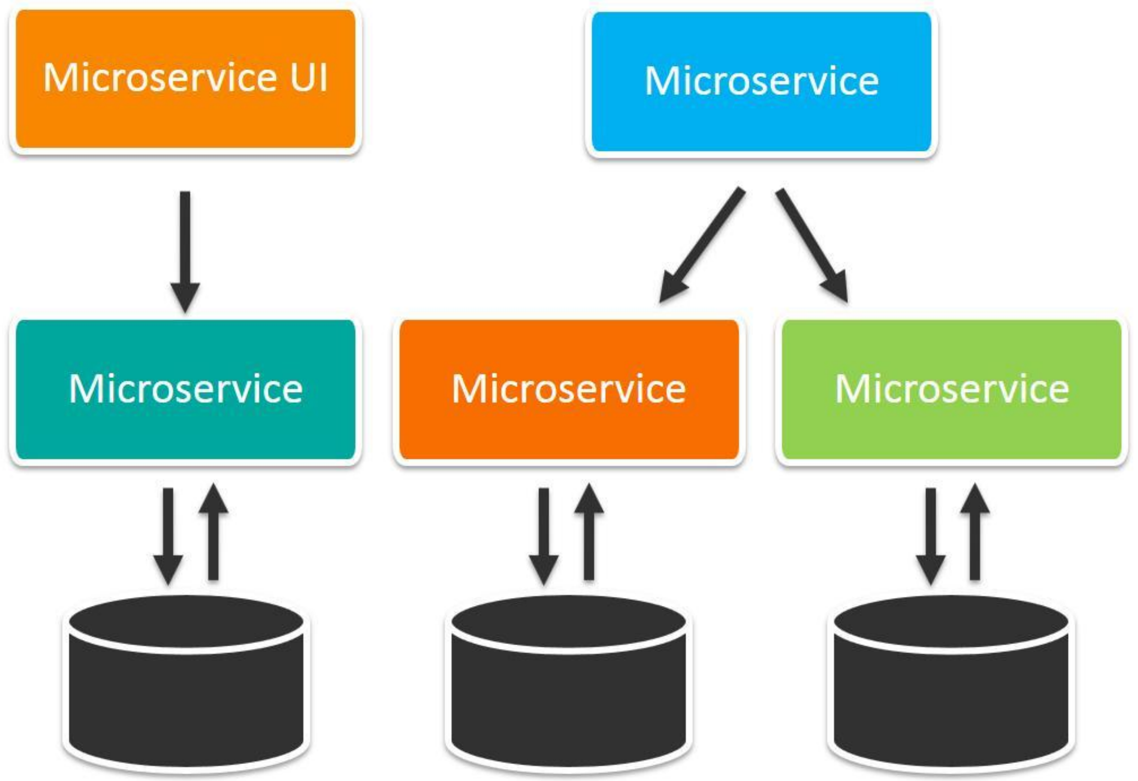
- **1400 brokers**
- **1.4 trillion messages/day**
- **AVRO encoded records**
- **Multiple uses:**
  - Message queue/data bus
  - Database replication
  - Metrics
  - Logging
  - Web app tracking data
  - Real-time Aggregation

[1]https://engineering.linkedin.com/blog/2016/04/kafka-ecosystem-at-linkedin

Monolithic Architecture

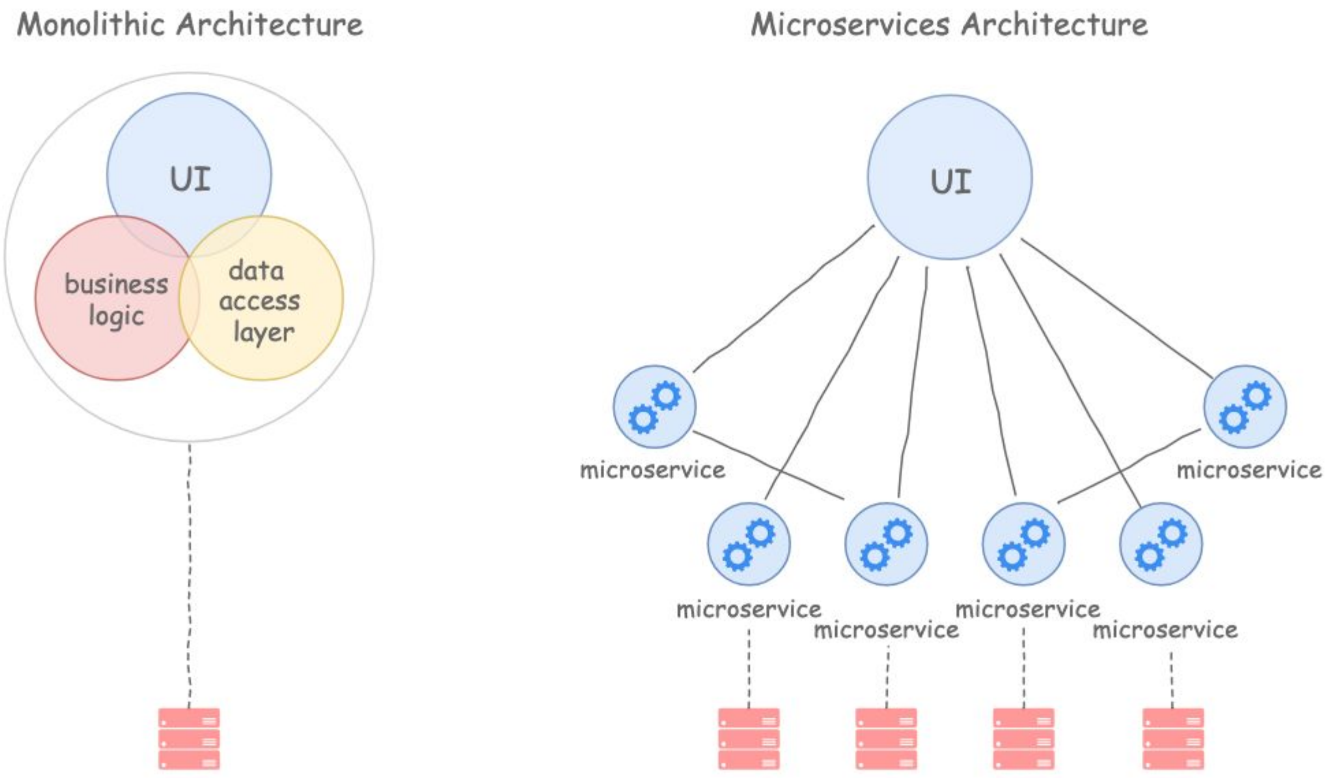


Microservices Architecture

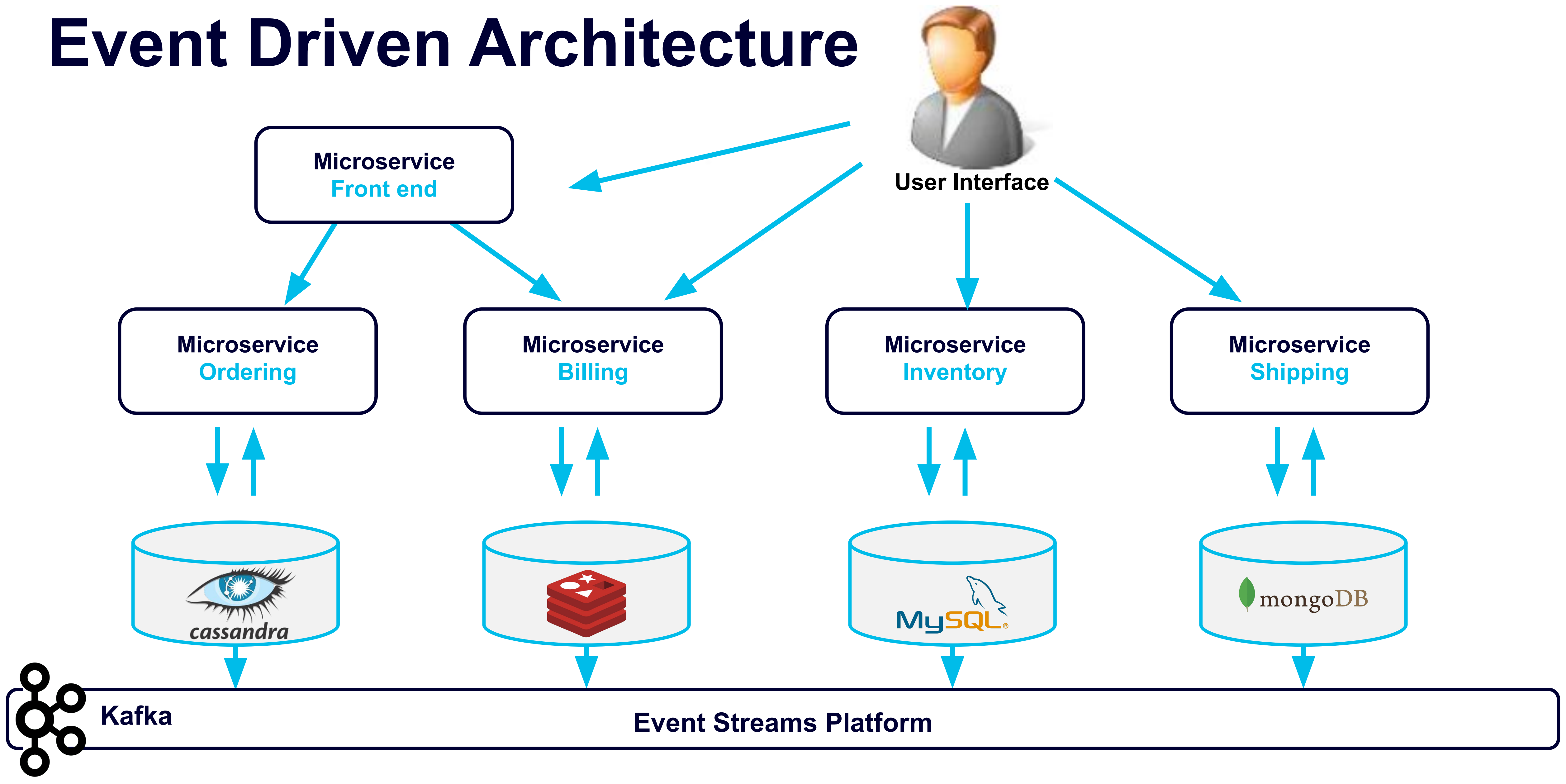


Microservices Advantages

- **Fast value delivery**
  - Fixes
  - New features
  - Experiments
  - Increased confidence
- **Language independent**
- **Fault isolation**
- **Pair well with containers**
- **Scalability**
- **Flexibility**



Event Driven Architecture



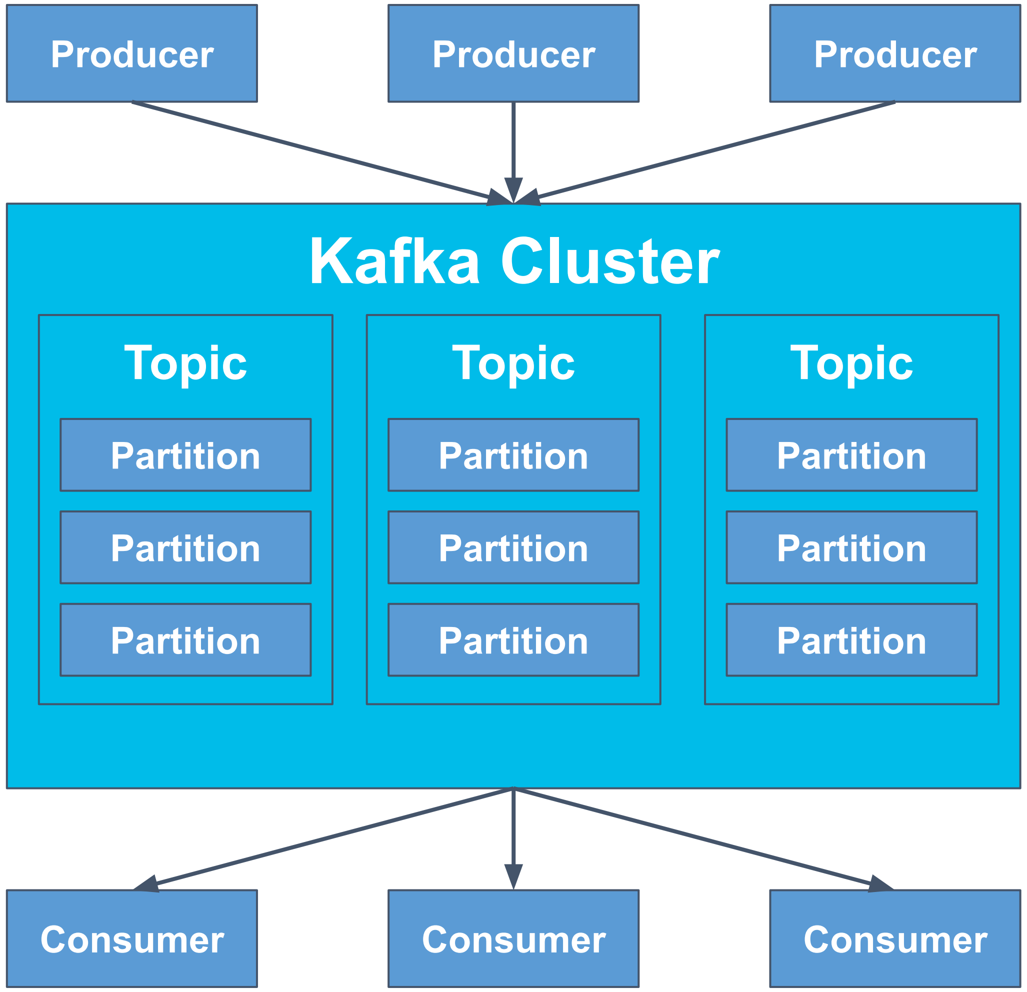
instaclustr

What Is Kafka?

Kafka Fundamentals

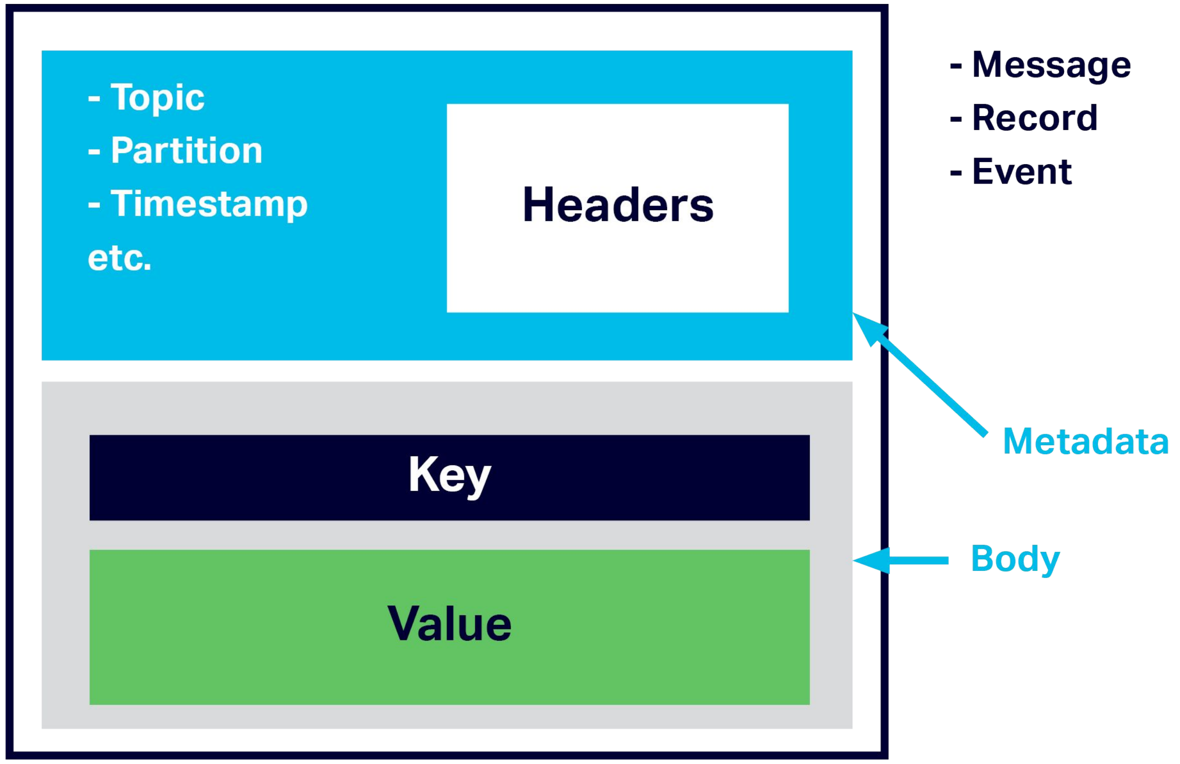
- **Records/Messages** → have a key, value, and timestamp
- **Topic** → a stream of records (“/orders”, “/user-signups”), feed name
  - **Log → topic storage on disk**
  - **Partition → topics are divided into a fixed number of partitions, amongst which the records are divided.**
- **Producer** → produces a stream of records
- **Consumer** → consumes a stream of records
- **Broker** → Kafka server that runs in a Cluster
- **Cluster:** A group of Brokers. This is also called a Kafka Cluster
- **ZooKeeper**: Coordinates brokers within the cluster.

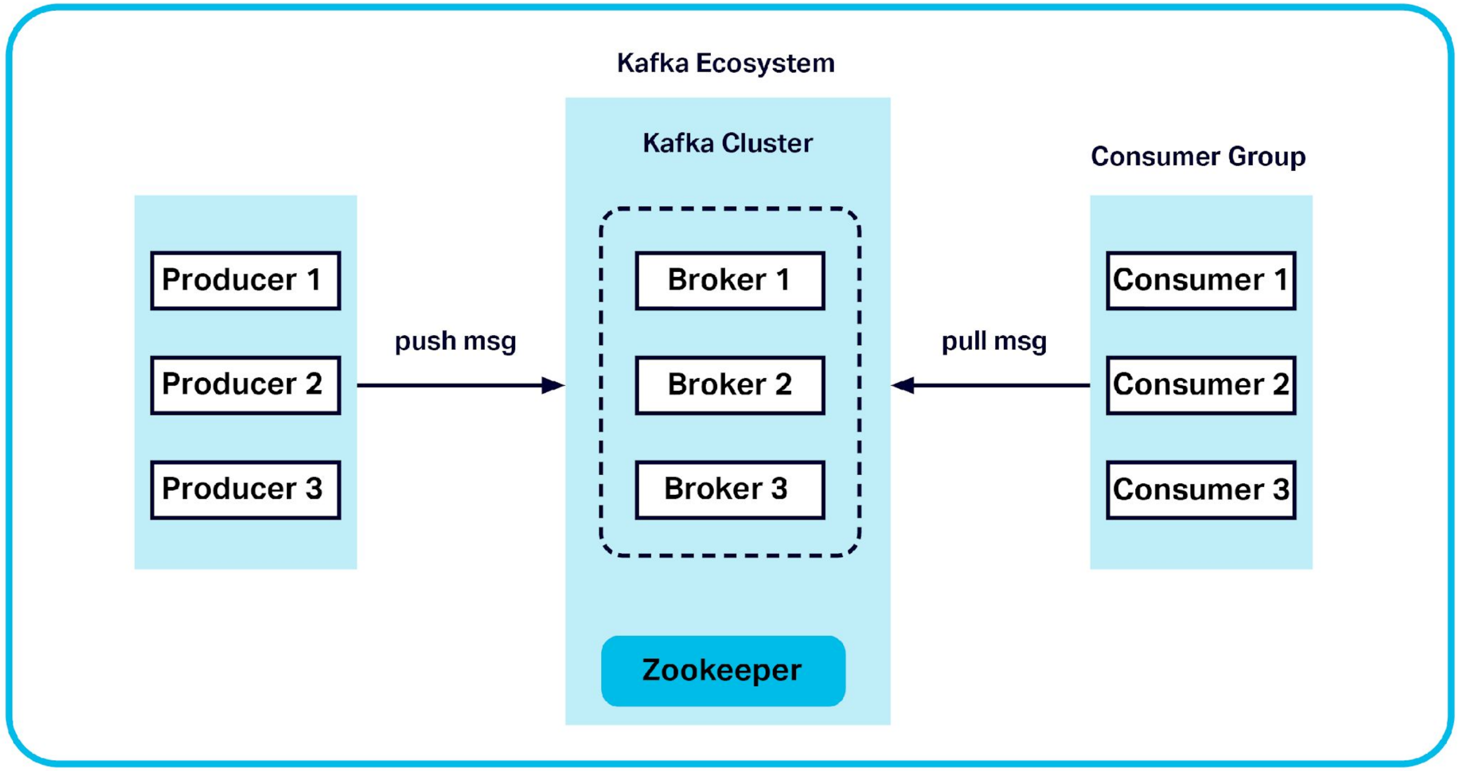
Kafka Components: Producers, Consumers, Topics, etc



Kafka Messages

- **The basic unit of data in Kafka is a message also known as record**
- **A message is a key-value pair**
  - All data is stored in Kafka as byte arrays
  - Producer provides serializers to convert the key and value to byte arrays
  - Key and value can be any data type




Kafka Ecosystem
Kafka Cluster
Consumer Group
Producer 1
Producer 2
Producer 3
push msg
Broker 1
Broker 2
Broker 3
pull msg
Consumer 1
Consumer 2
Consumer 3
Zookeeper

instaclustr

Data Order in Kafka

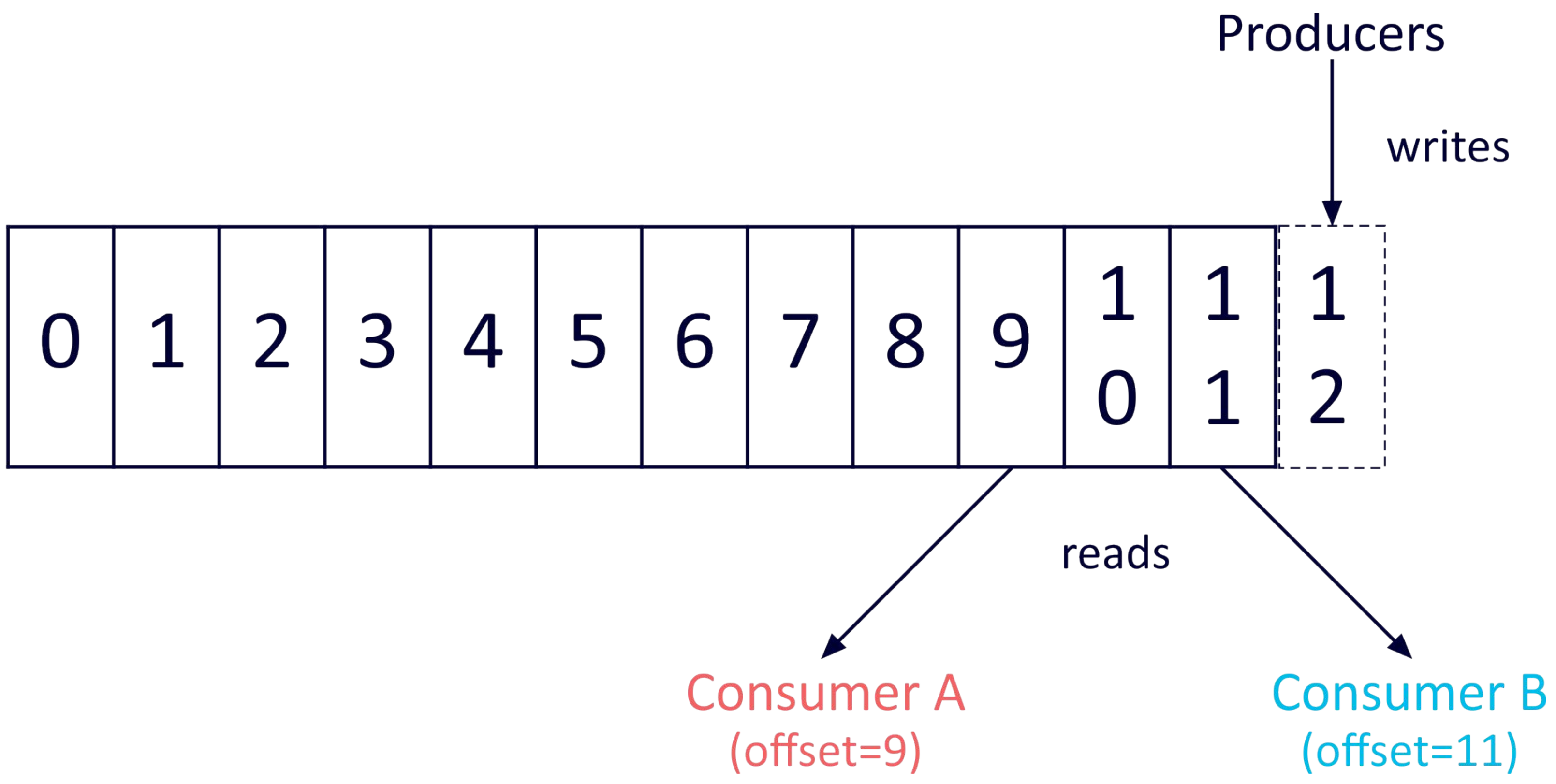
What Kafka Guarantees

Messages sent by a producer to a particular topic partition will be appended in the order they are sent.

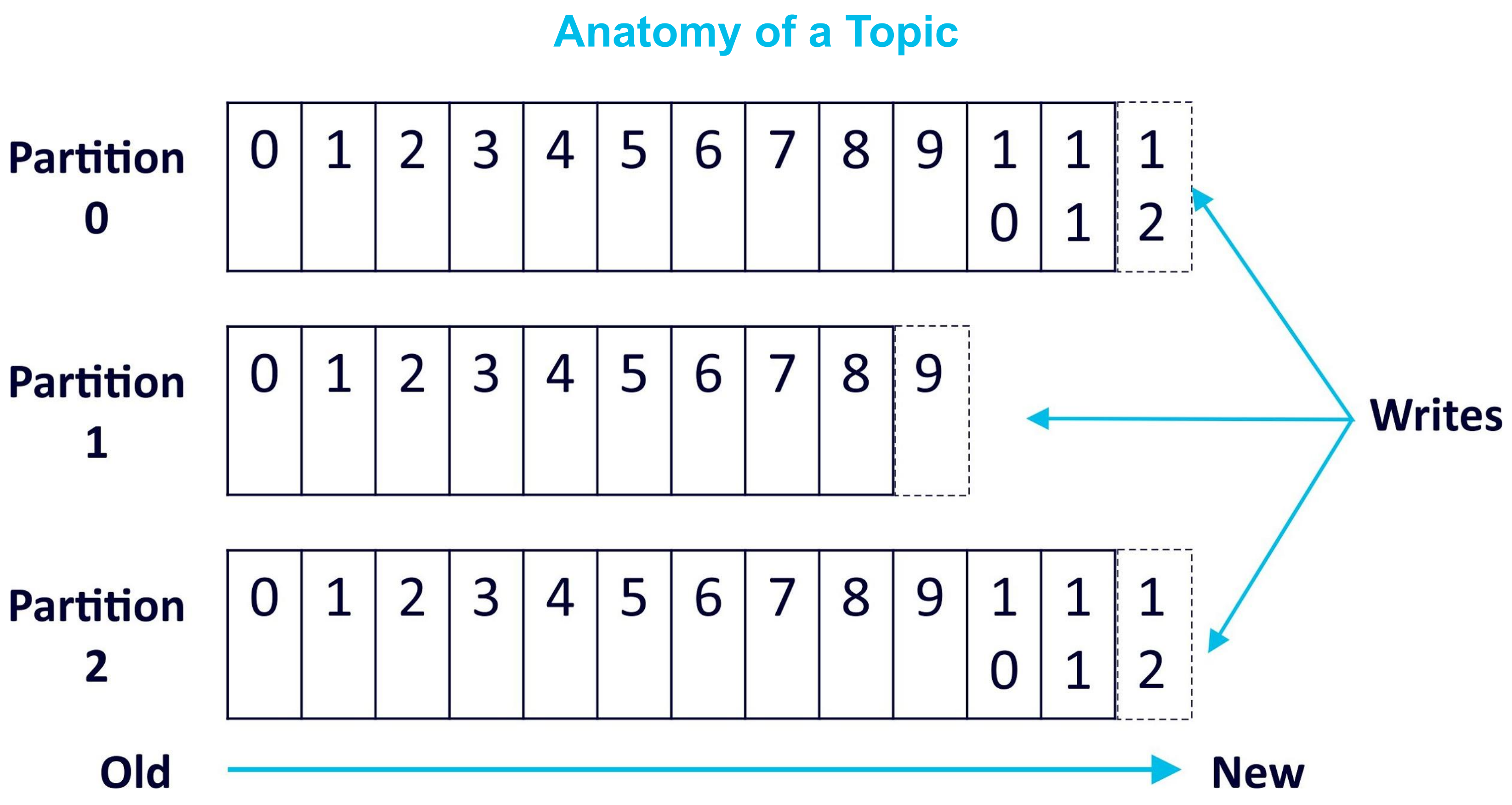
A consumer instance sees records in the order they are stored in the log.

For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.

What Kafka Guarantees



What Kafka Guarantees



What happens if we send data to many partitions?

10 Partitions

```
./kafka-topics.sh --create \
 --zookeeper localhost:2181/kafka \
 --replication-factor 1 --partitions 10 \
 --topic my-topic
```

Send Some Data

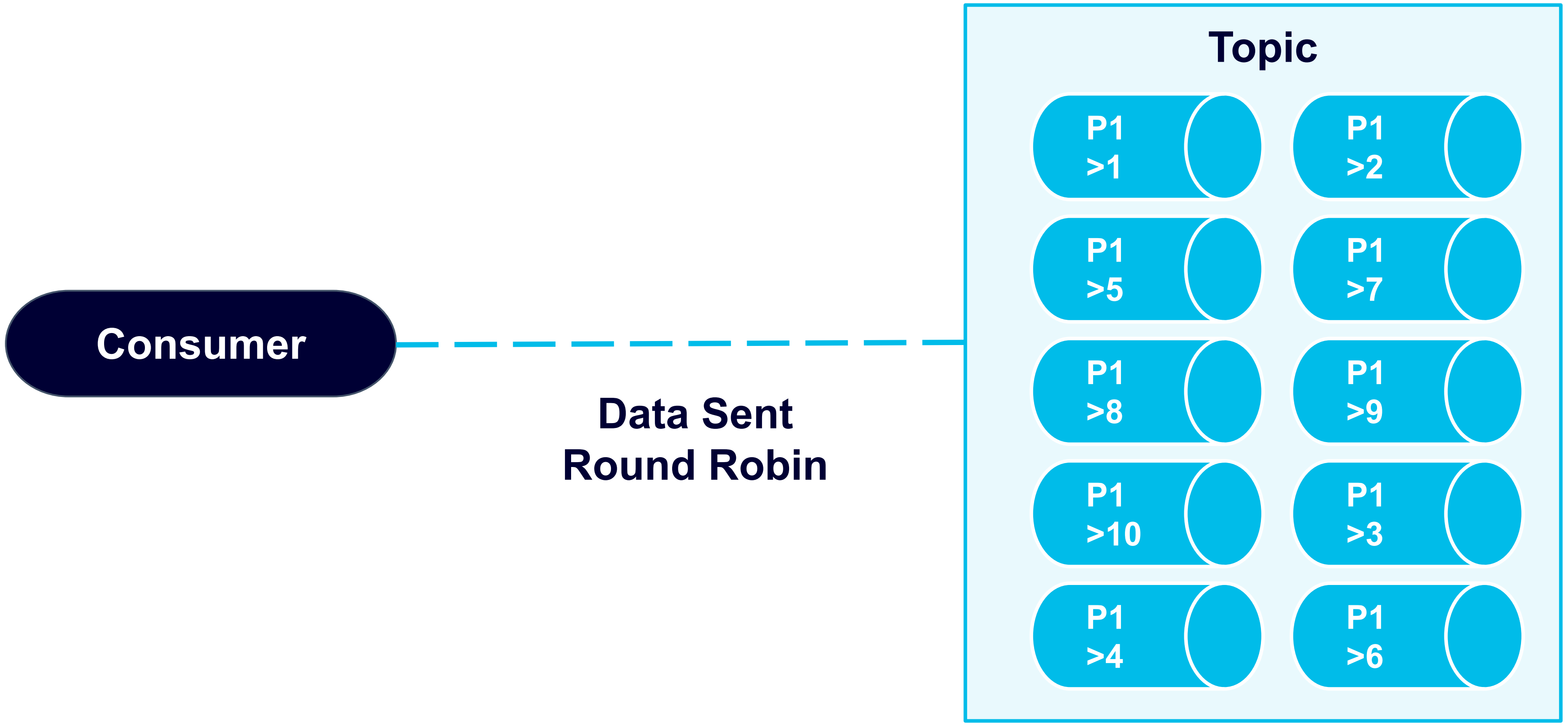
```
./kafka-console-producer.sh \
 --broker-list localhost:9092 \
 --topic my-topic

> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10
```

Receive the Same Data

```
./kafka-console-consumer.sh \
 --bootstrap-server localhost:9092 \
 --topic my-topic \
 --from-beginning
```

```
> 2
> 5
> 9
> 3
> 4
> 7
> 6
> 10
> 8
> 1
```

How did this happen?



What happens if we send data to one partition?

1 Partition

```
./kafka-topics.sh --create \
 --zookeeper localhost:2181/kafka \
 --replication-factor 1 --partitions 1 \
 --topic my-topic
```

Send Some Data

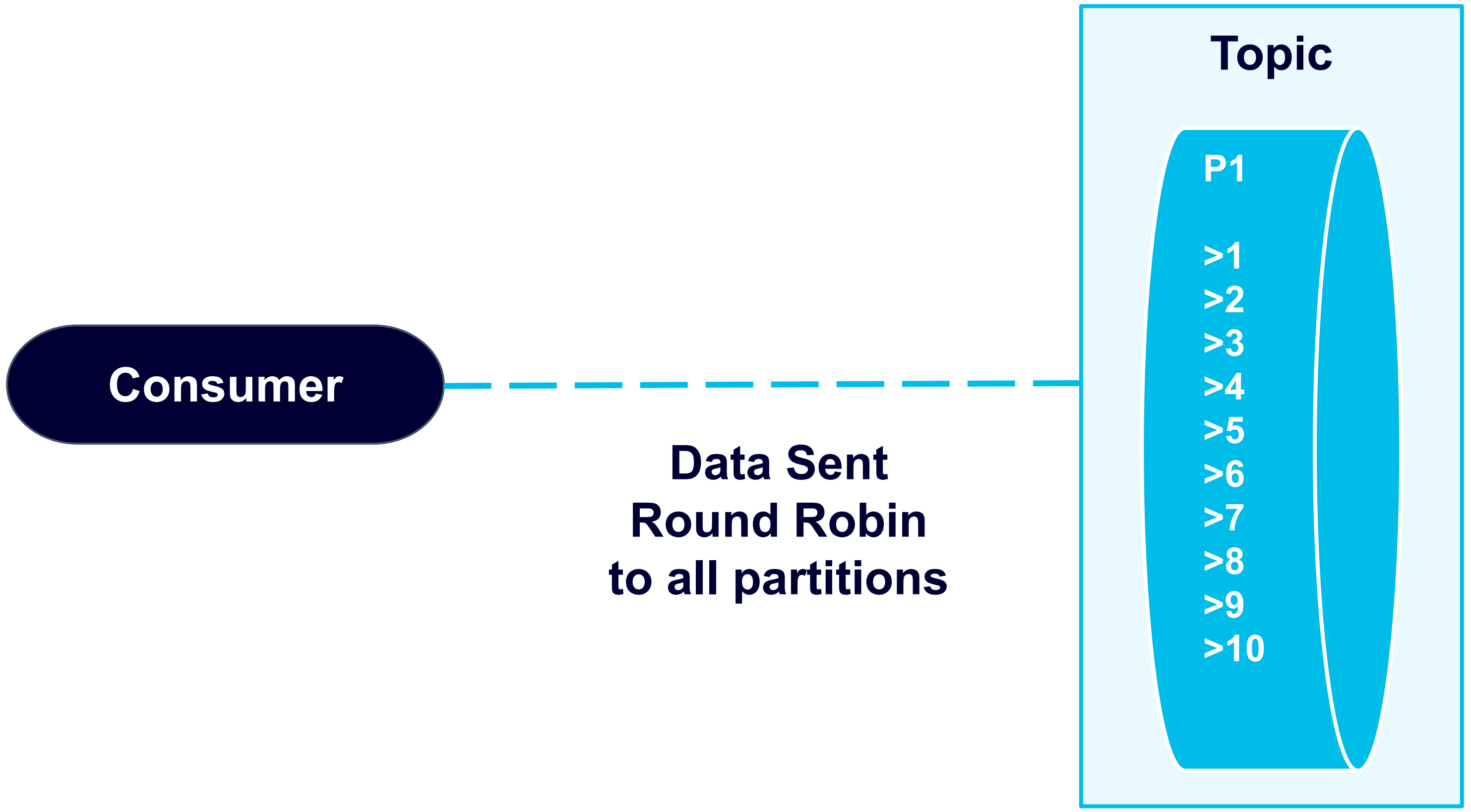
```
./kafka-console-producer.sh \
 --broker-list localhost:9092 \
 --topic my-topic

> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10
```

Receive the Same Data

```
./kafka-console-consumer.sh \
 --bootstrap-server localhost:9092 \
 --topic my-topic \
 --from-beginning
```

```
> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10
```

How did this happen?

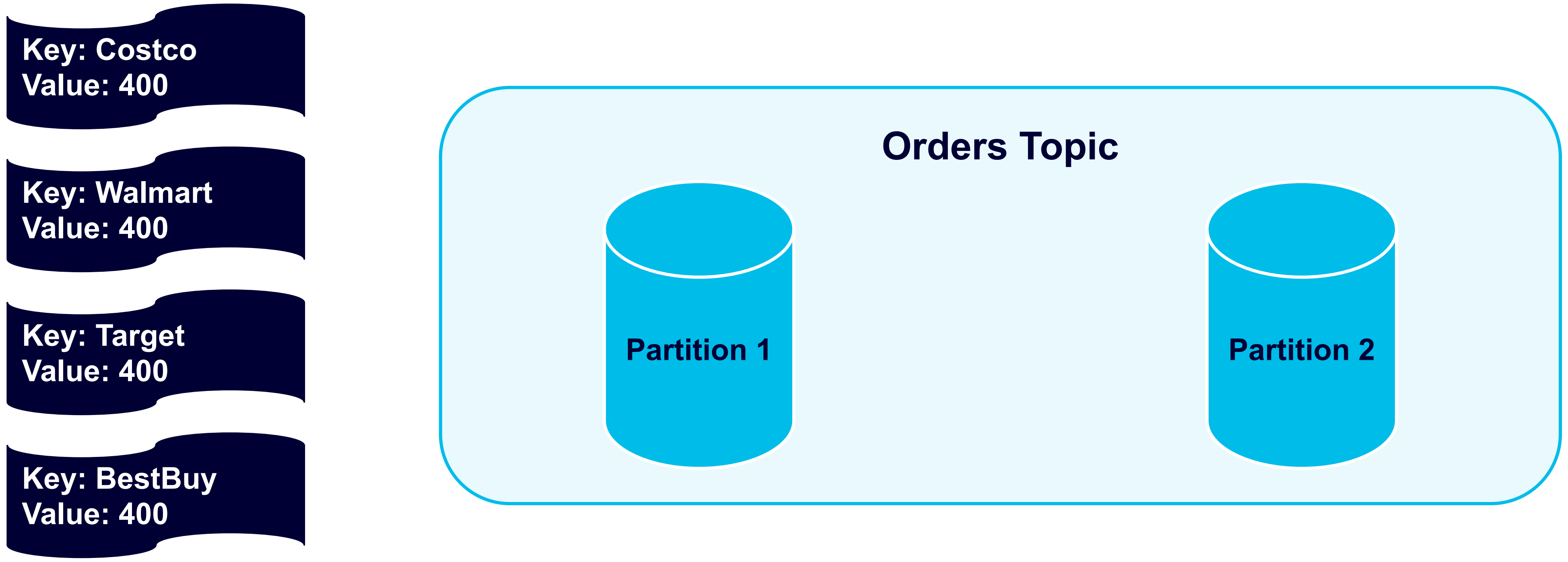


What happens if we introduce keying?

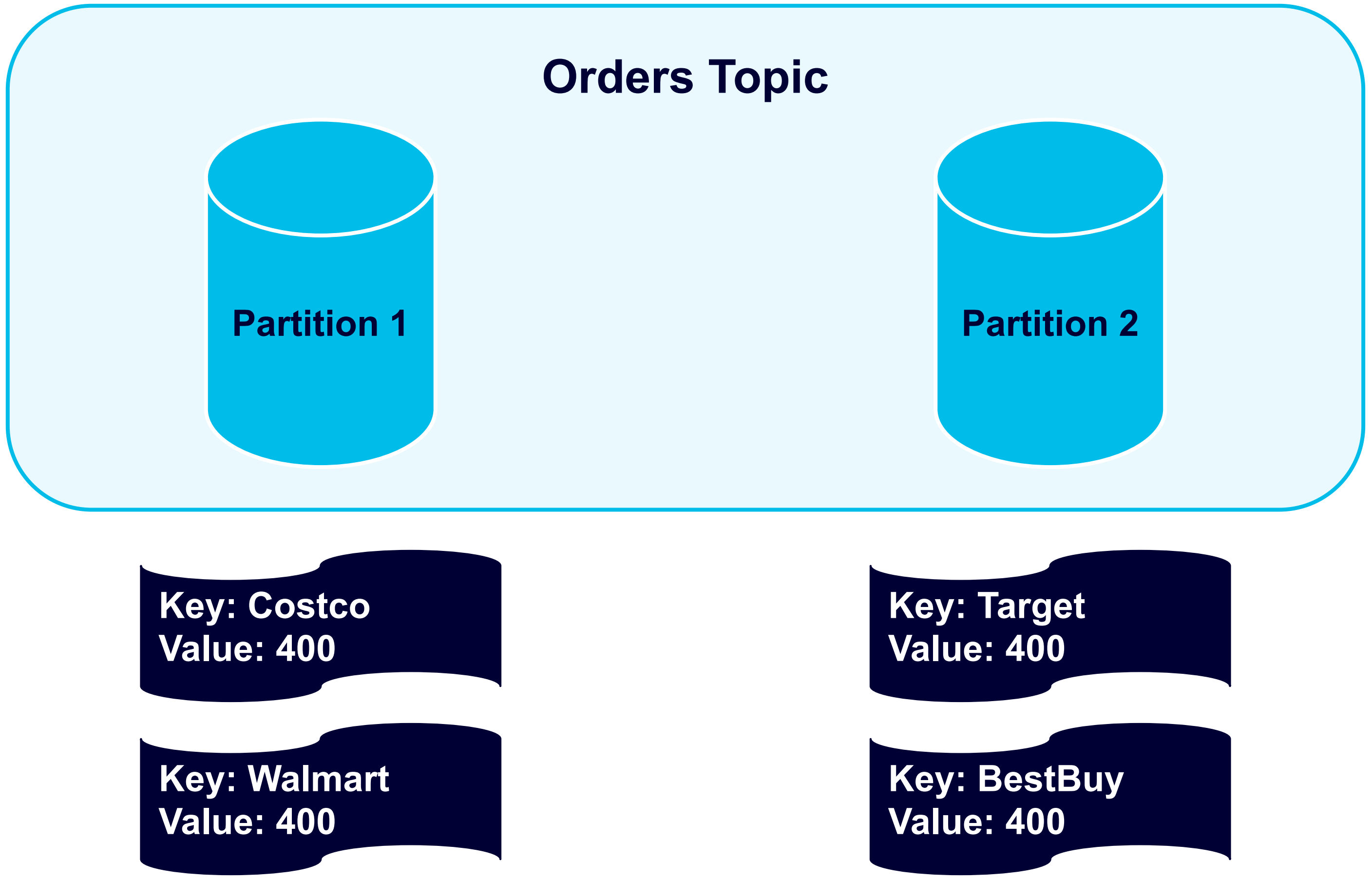
Kafka messages with a key

All Key messages with a Key will go to the same partition

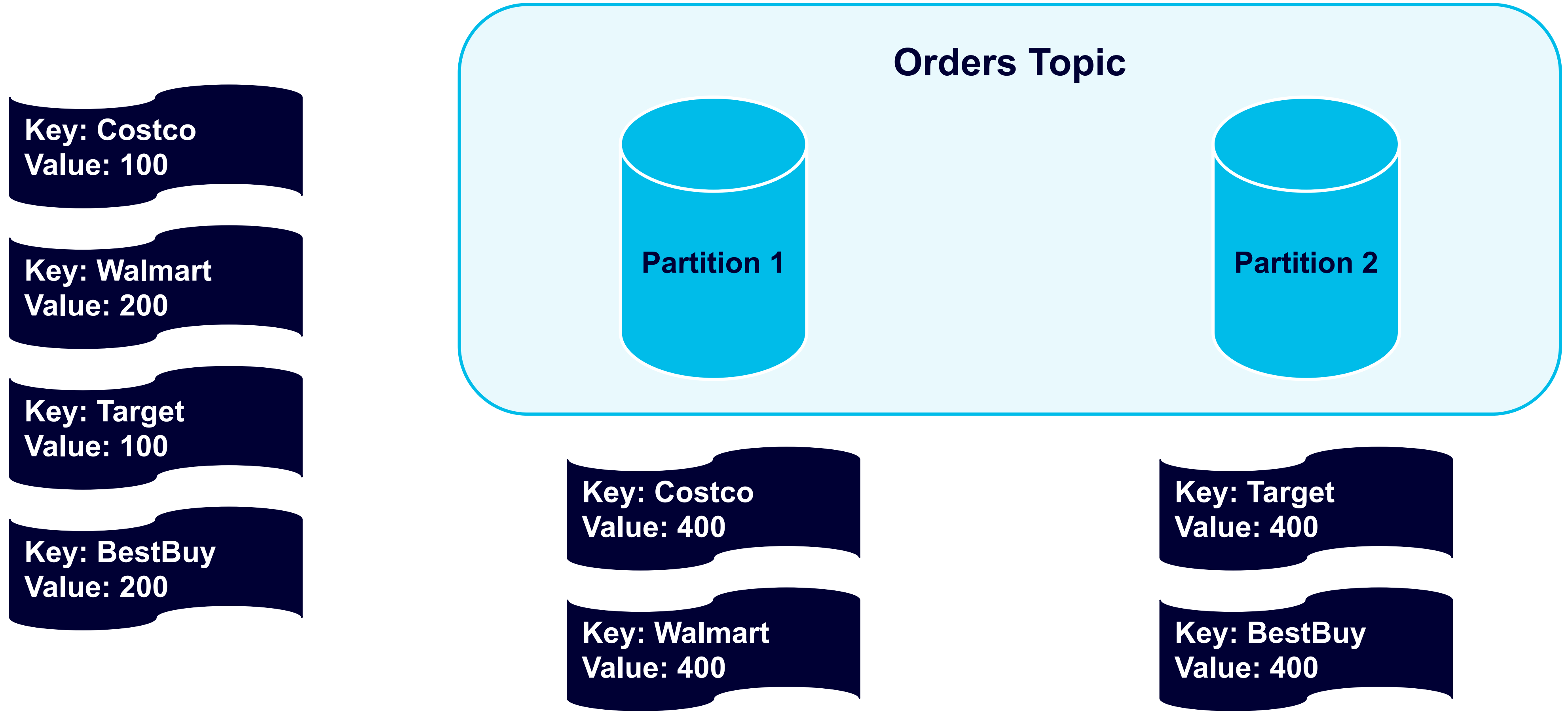
Suppose we want to send 4 messages to a Kafka topic with 2 partitions



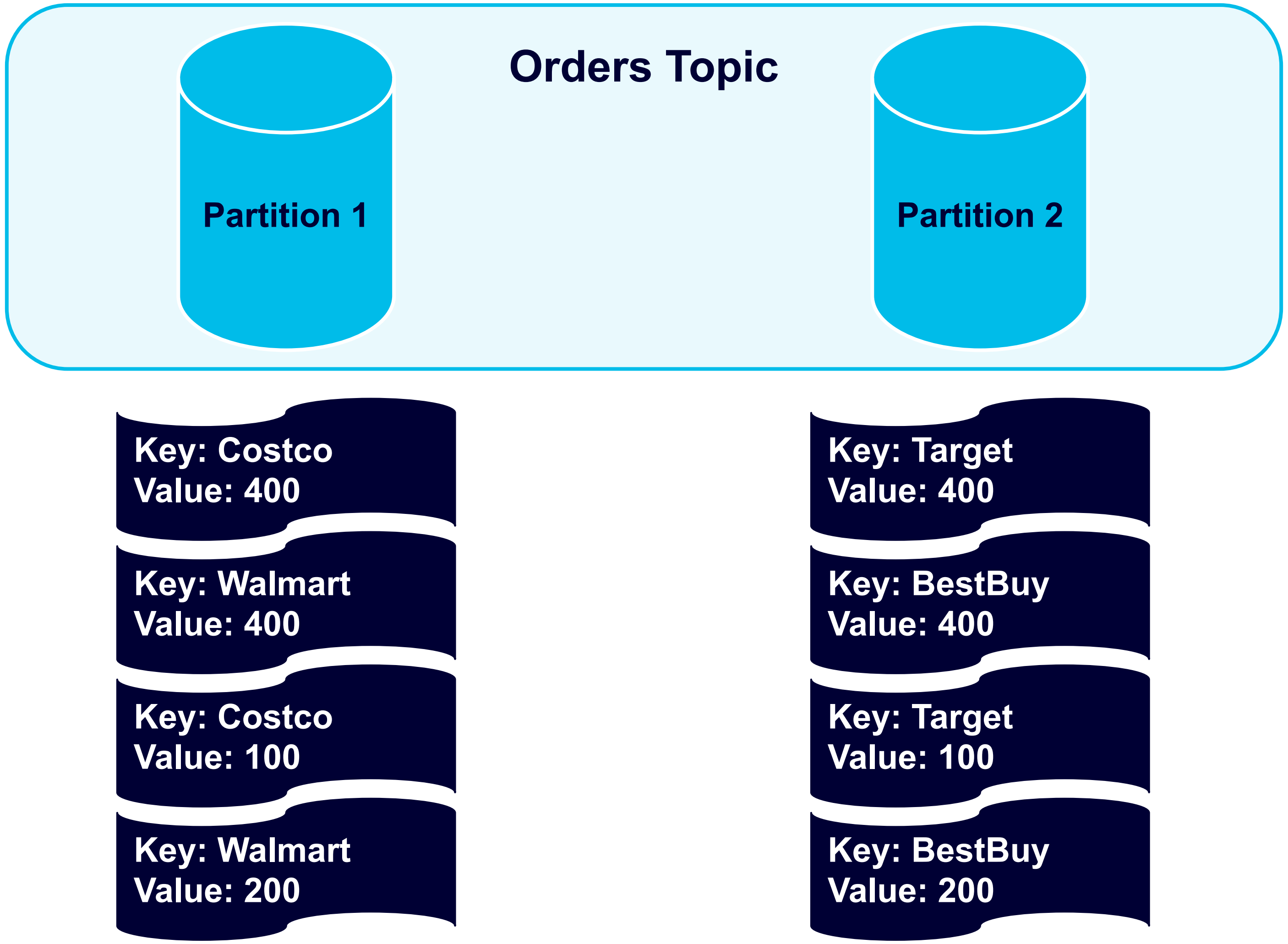
Keys are hashed and distributed across the cluster



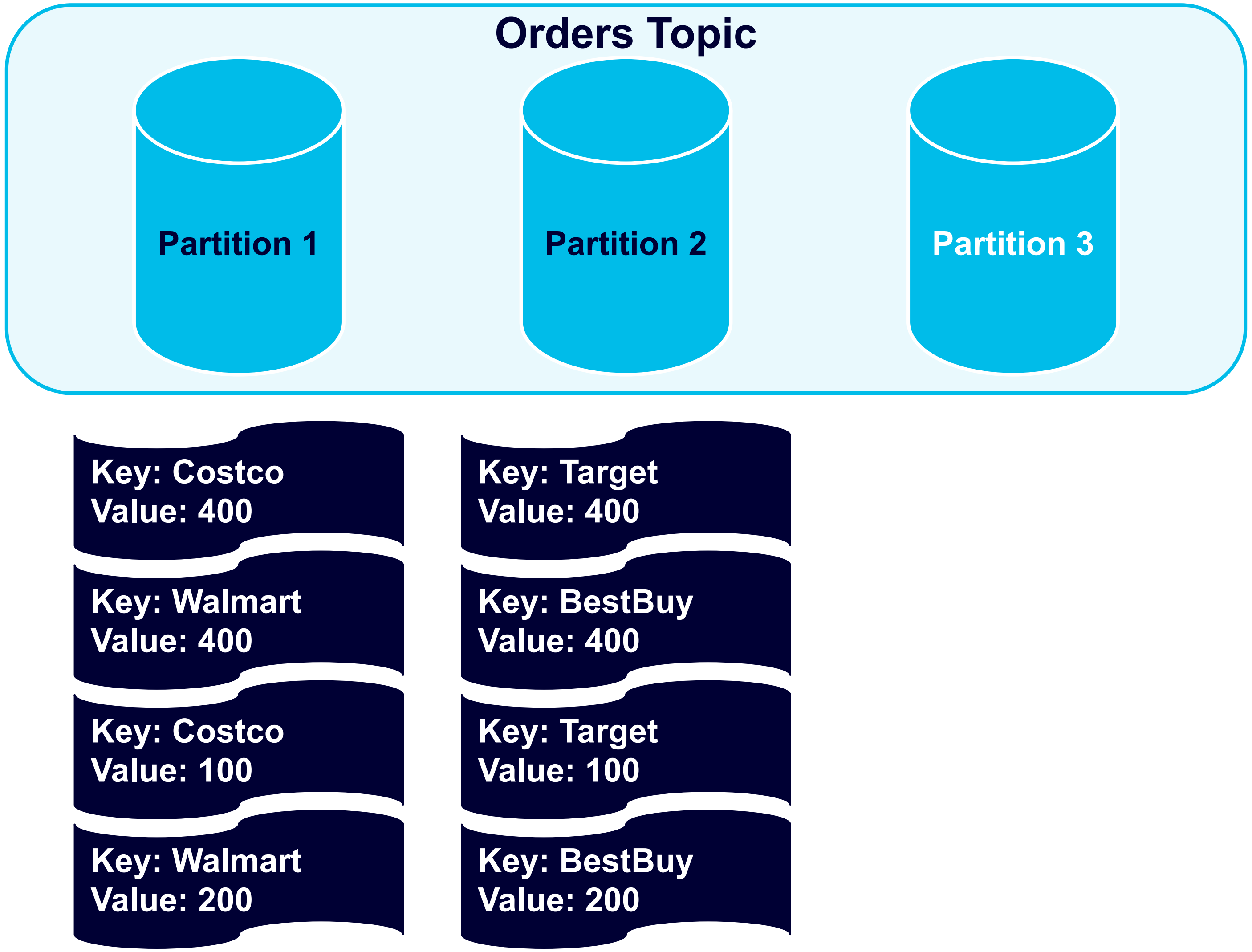
Suppose we want to send 4 MORE messages to a Kafka topic with 2 partitions



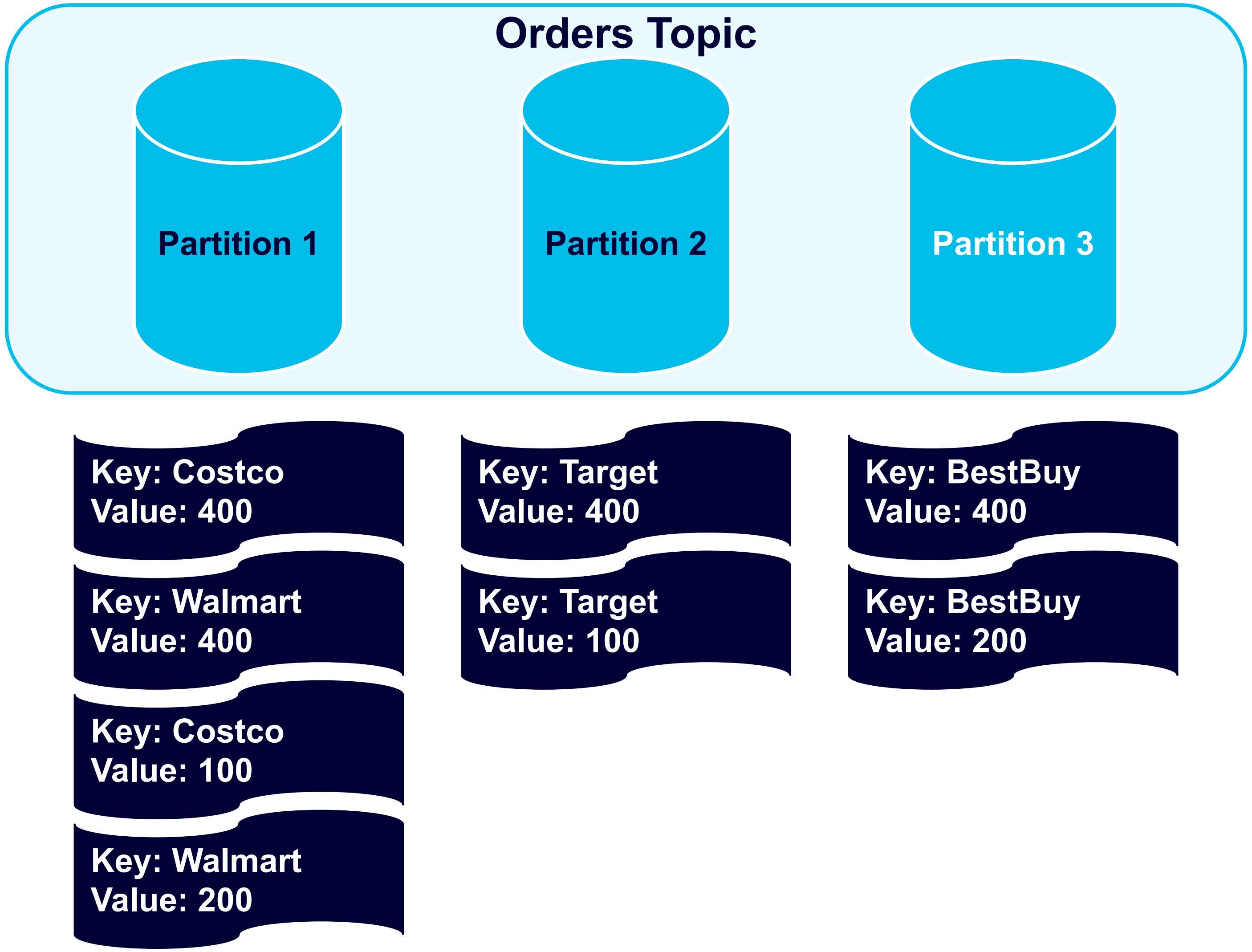
Messages are sent to the same partition using the existing key



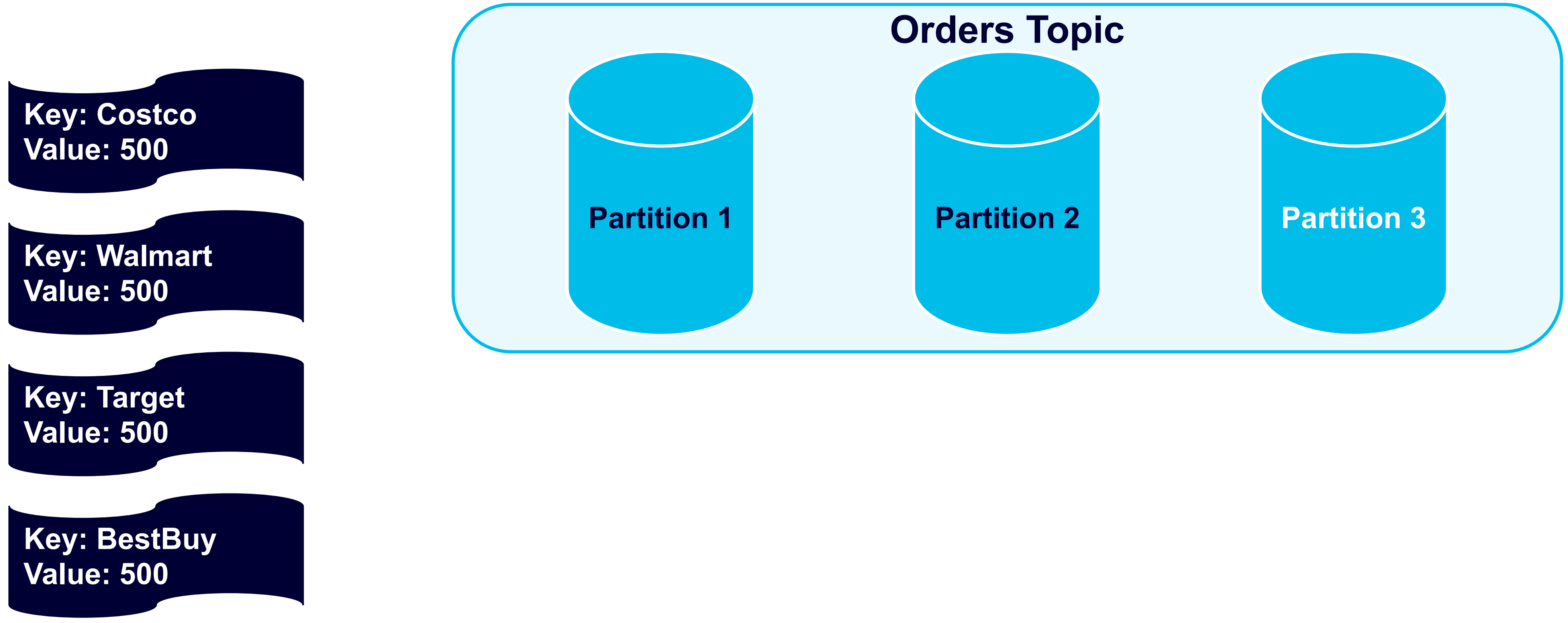
Suppose we add more partitions to the cluster



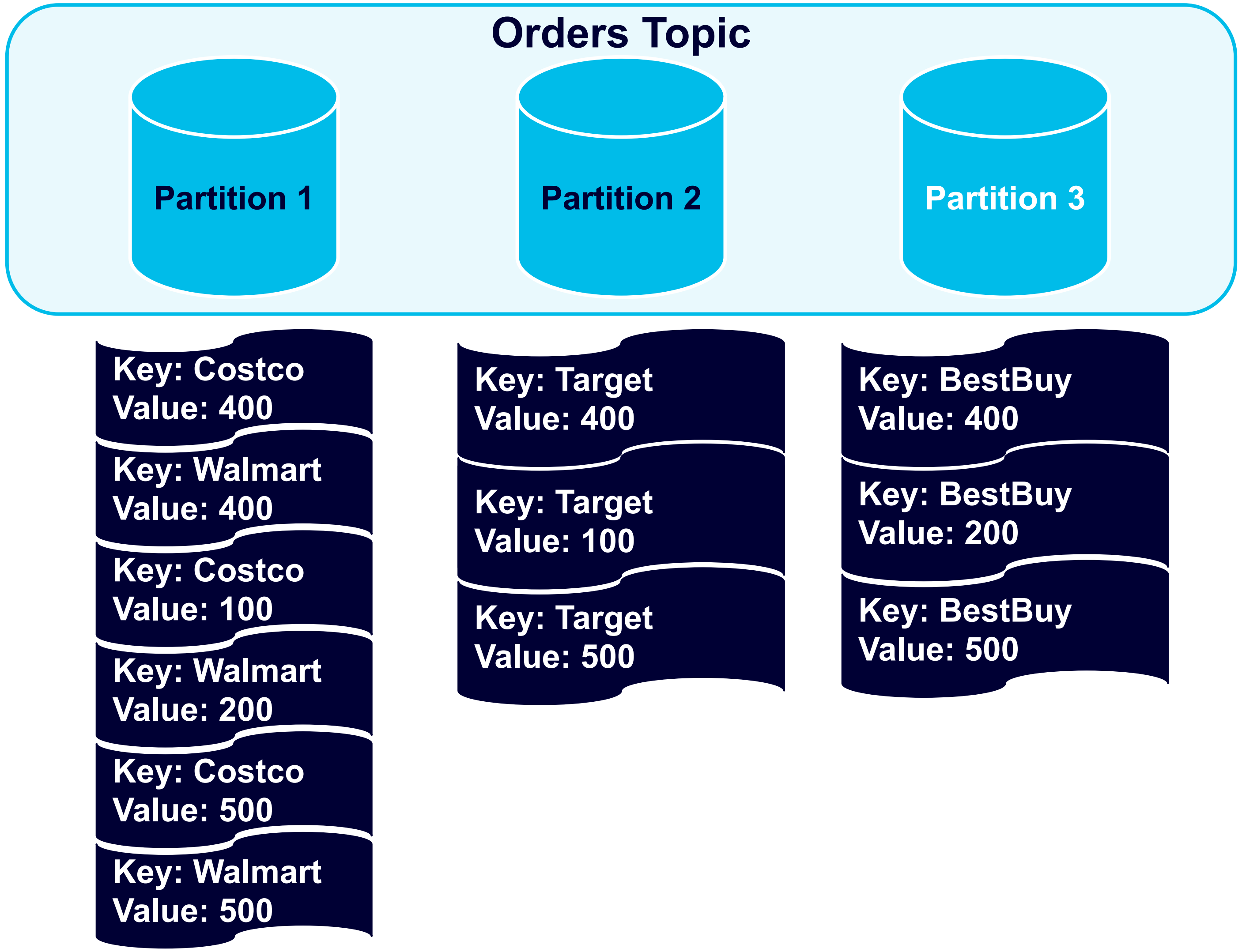
Suppose we then decide to rebalance the partitions



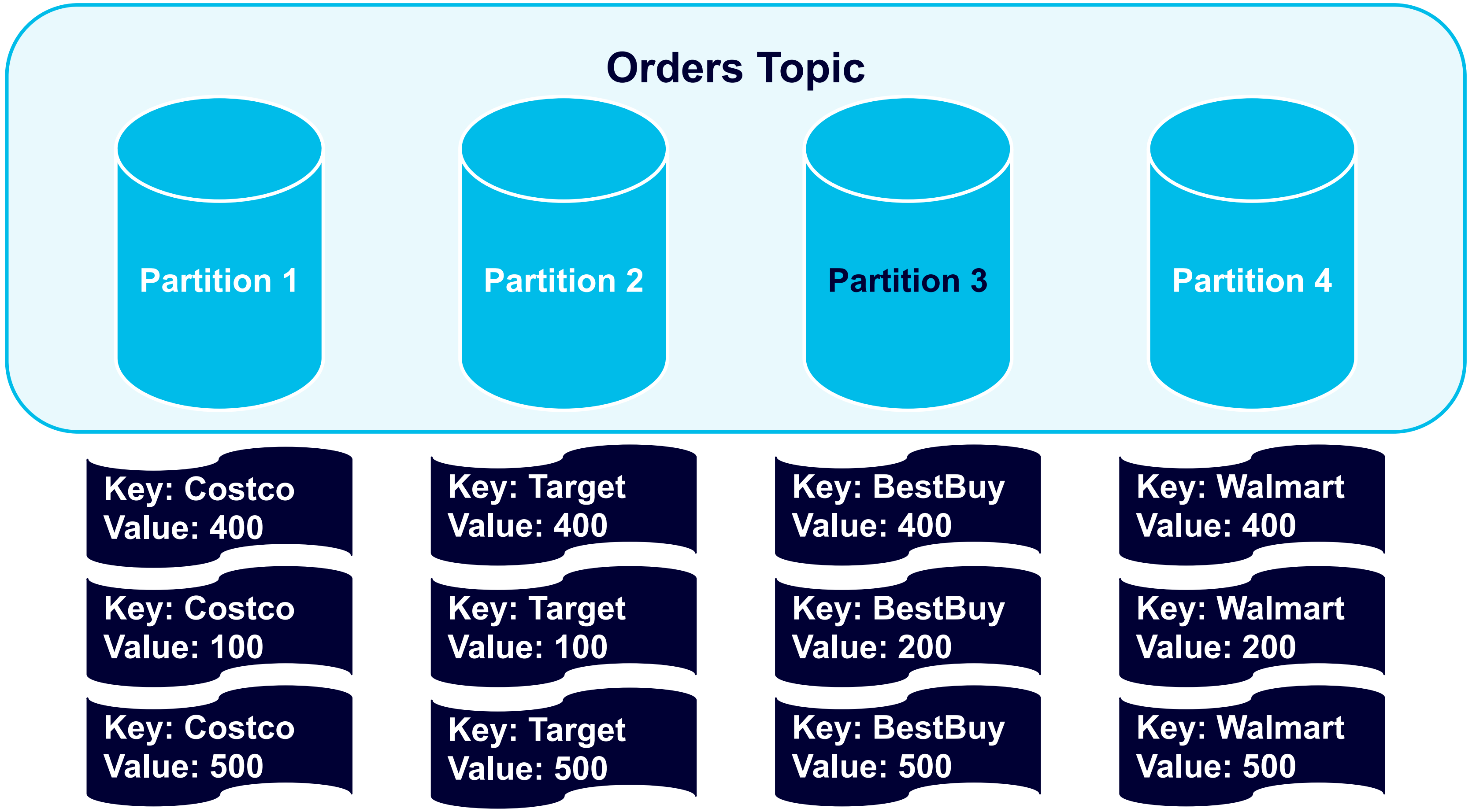
Suppose we want to send 4 MORE messages to a kafka topic with 3 partitions



Suppose we then decide to rebalance the partitions again

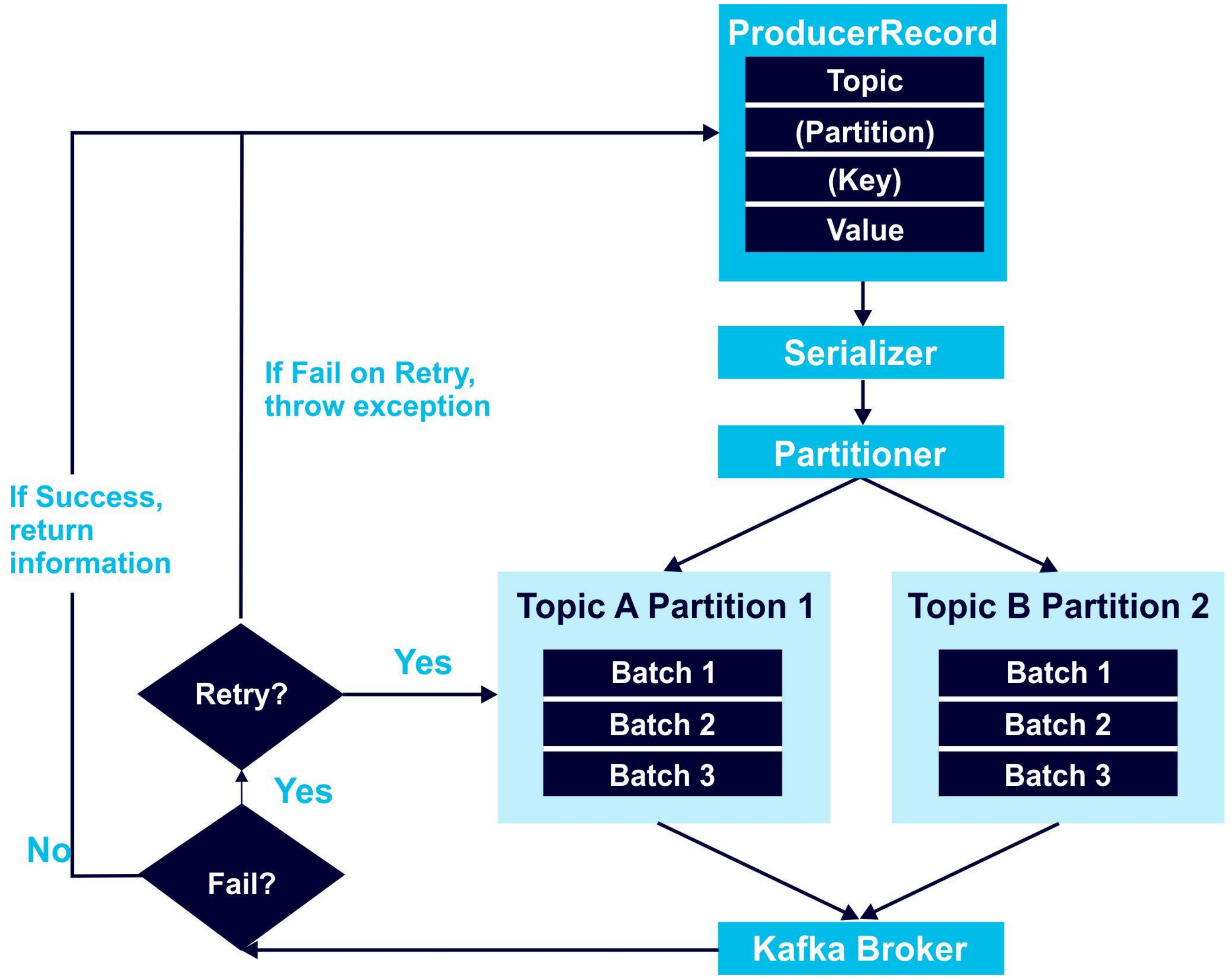


The result looks like Balanced Partitions



How do make sure the data is always sent in order?

Kafka Producer Overview



max.in.flight.requests.per.connection

Setting the retries parameter to nonzero and the **max.in.flight.requests.per.connection** to more than one means that it is possible that the broker will fail to write the first batch of messages, succeed to write the second (which was already in-flight), and then retry the first batch and succeed, thereby reversing the order.

max.in.flight.requests.per.connection

Usually, setting the number of retries to zero is not an option in a reliable system, so if guaranteeing order is critical, we recommend setting **in.flight.requests.per.session = 1** to make sure that while a batch of messages is retrying, additional messages will not be sent (because this has the potential to reverse the correct order).

This will severely limit the throughput of the producer, so only use this when order is important.

Kafka Delivery Guarantees

At Once

Guarantees that a particular message will always be delivered.

At Least Once

Guarantees that a particular message will always be delivered.

Exactly Once

Guarantees that all messages will always be delivered exactly once.

Exactly Once Delivery

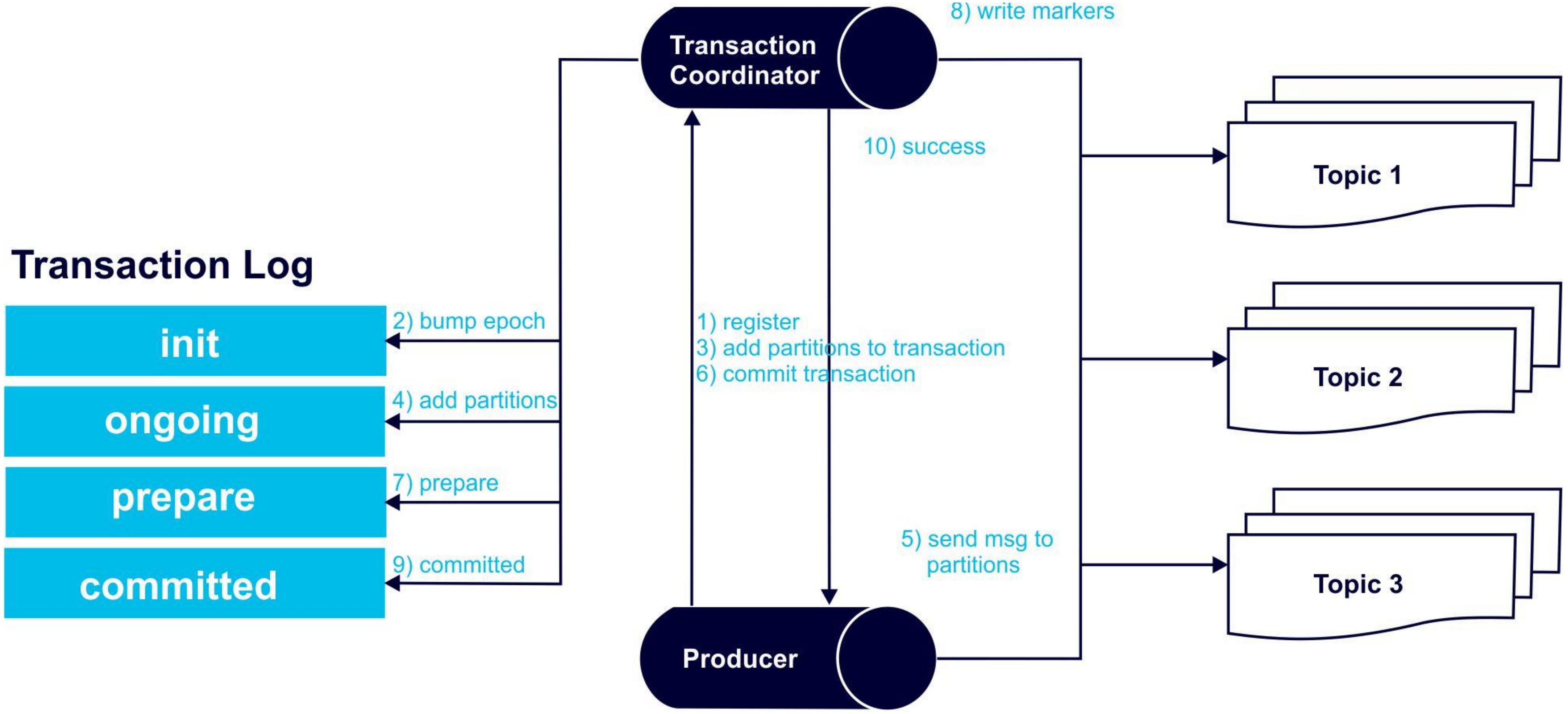
Idempotent Producer

Transactions Across Partitions

Transactional Consumer

instaclustr

Kafka Transaction Example Workflow



Exactly Once Semantics Diagram

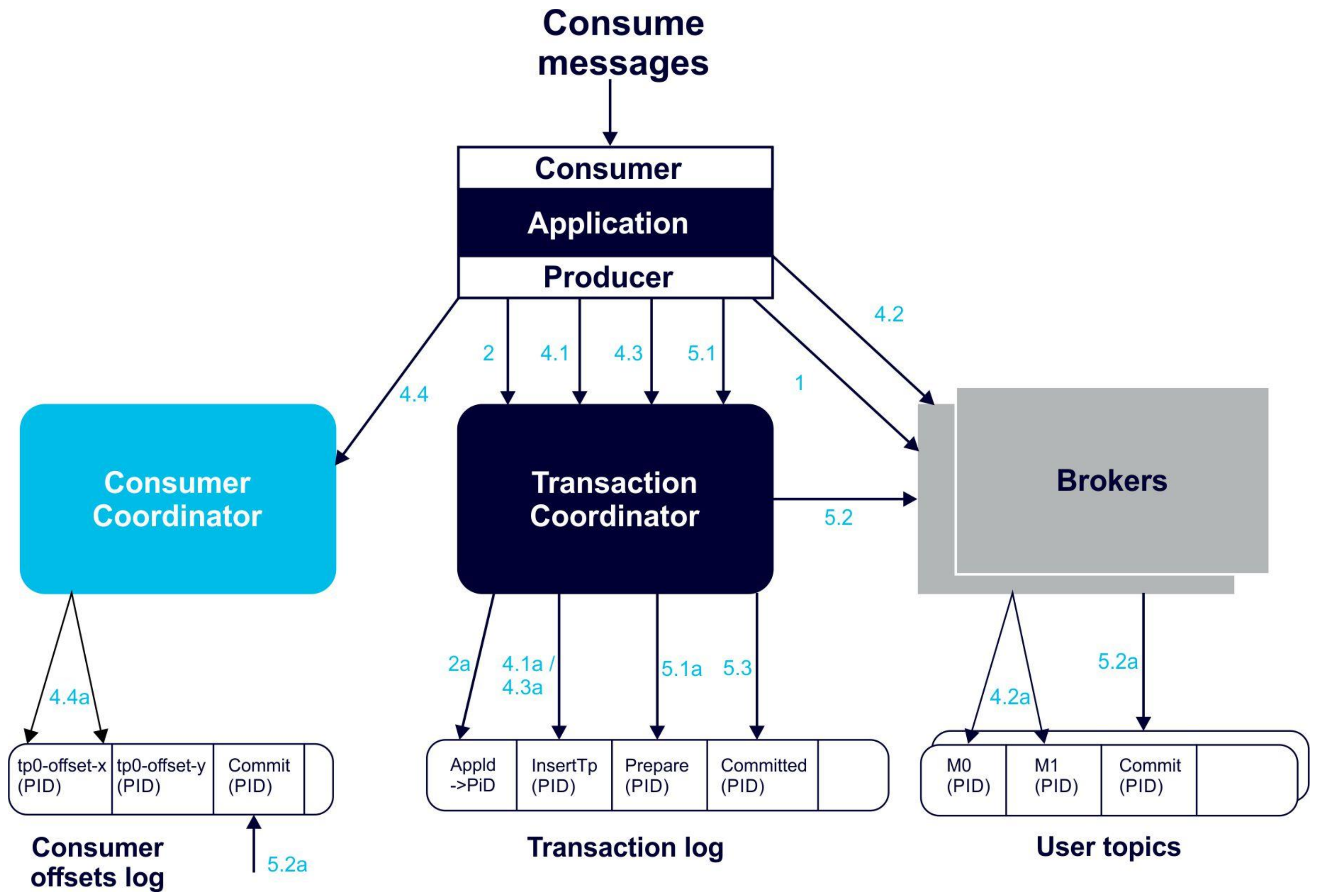


Diagram Originally produced Here

info@instaclustr.com

instaclustr