Kafka 101 & Developer Best Practices

Agenda

- Kafka Overview
- Kafka 101
- Best Practices for Writing to Kafka: A tour of the Producer
- Best Practices for Reading from Kafka: The Consumer
- General Considerations

ETL/Data Integration

−	+
Batch	**High Throughput**
Expensive	**Durable**
Time Consuming	**Persistent**
	Maintains Order

Messaging

+	−
Fast (Low Latency)	Difficult to Scale
	No Persistence
	Data Loss
	No Replay

ETL/Data Integration

−

Batch

Expensive

Time Consuming

+

High Throughput

Durable

Persistent

Maintains Order

Messaging

+

Fast (Low Latency)

−

Difficult to Scale

No Persistence

Data Loss

No Replay

ETL/Data Integration

−	+
Batch	**High Throughput**
Expensive	**Durable**
Time Consuming	**Persistent**
	Maintains Order

Stored records

Messaging

+	−
Fast (Low Latency)	Difficult to Scale
	No Persistence
	Data Loss
	No Replay

Transient Messages

Both of these are a complete mismatch to how your business works.

ETL/Data Integration

−

Batch

Expensive

Time Consuming

Stored records

Event Streaming Paradigm

\+

High Throughput

Durable

Persistent

Maintains Order

\+

Fast (Low Latency)

Messaging

−

Difficult to Scale

No Persistence

Data Loss

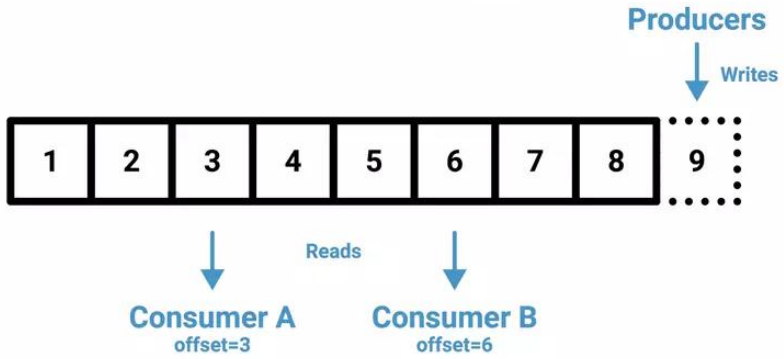
No Replay

Transient Messages

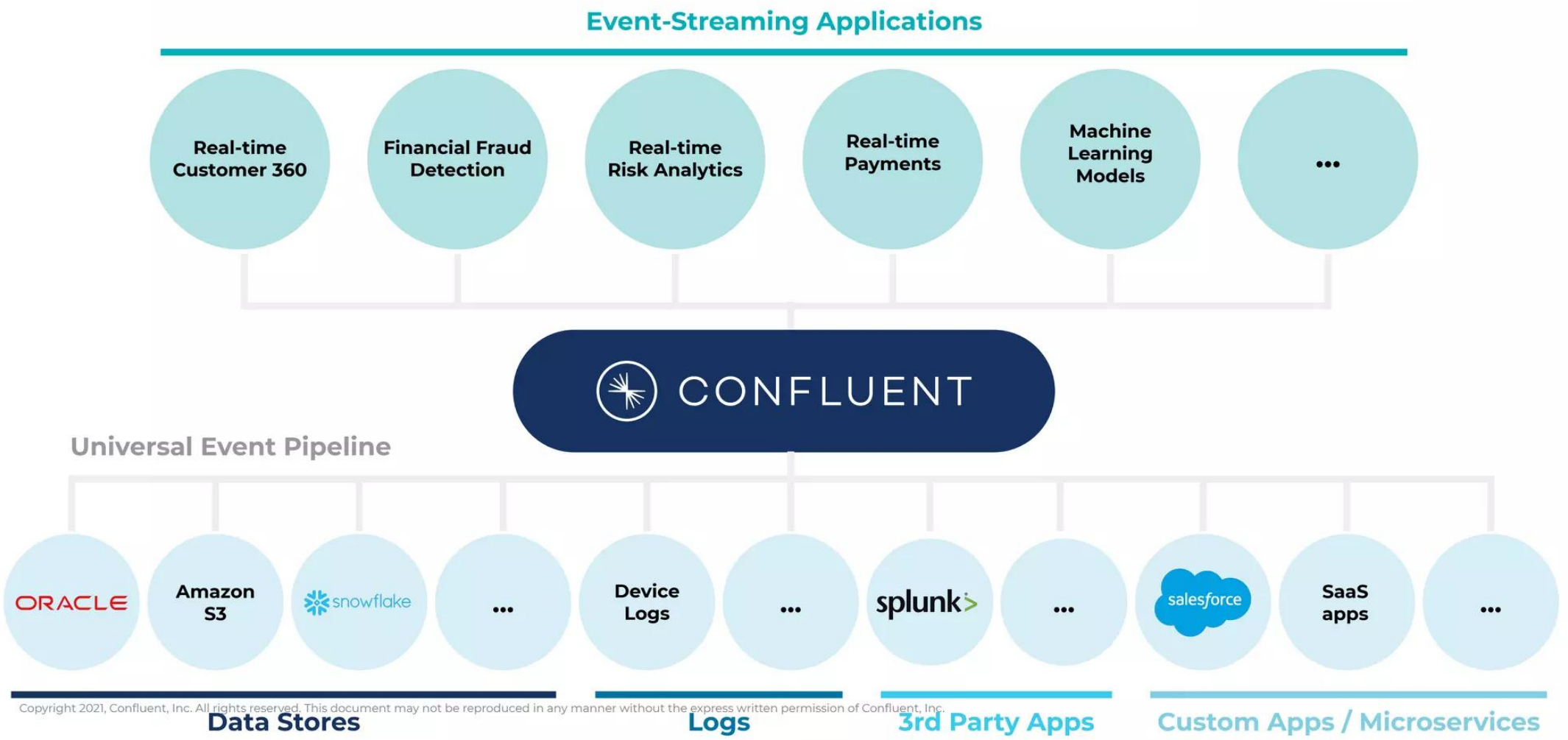
Event Streaming Paradigm

To rethink data as not stored records or transient messages, but instead as a continually updating stream of events

Event Streaming Paradigm



Confluent: Central Nervous System For Enterprise



Confluent uniquely enables Event Streaming success

Confluent founders are original creators of Kafka

Confluent team wrote 80% of Kafka commits and has over 1M hours technical experience with Kafka

Confluent Platform extends Apache Kafka to be a secure, enterprise-ready platform

Confluent helps enterprises successfully deploy event streaming at scale and accelerate time to market

AWARDS

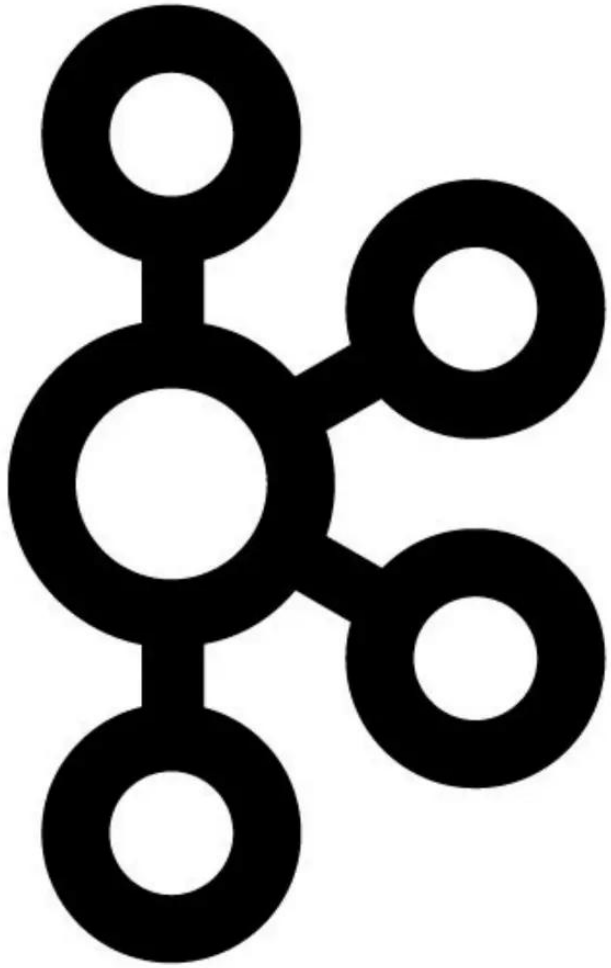
Partner of the Year
Google Cloud

Morgan Stanley
CTO Innovation Award Winner
2019

JPMorgan Chase & Co.
Hall of Innovation

BANK OF AMERICA
Enterprise Technology Innovation

Kafka 101



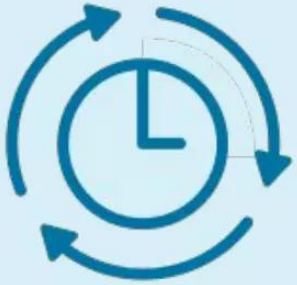
APACHE kafka

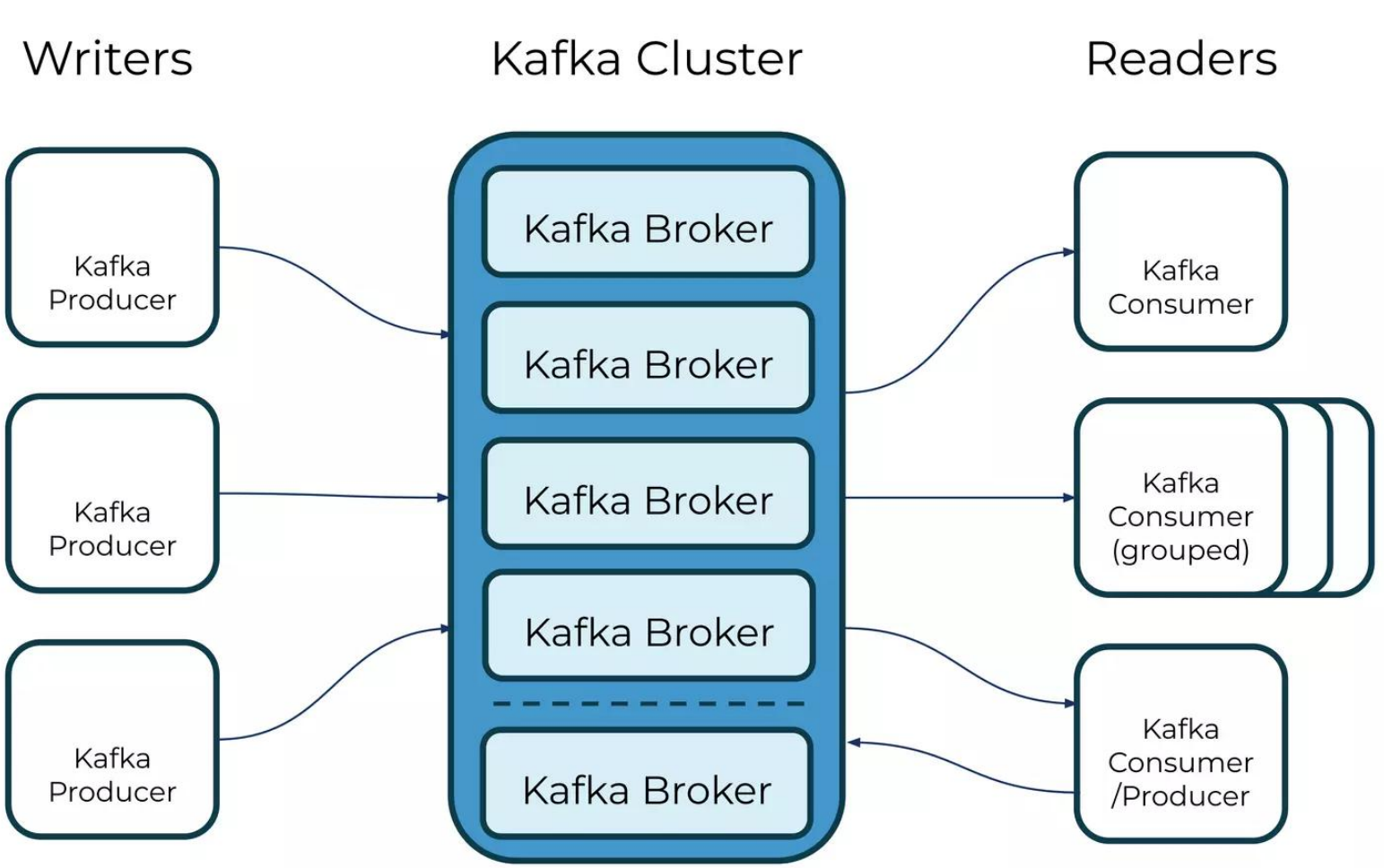
Scalability of a Filesystem

Guarantees of a Database

Distributed By Design

Rewind and Replay

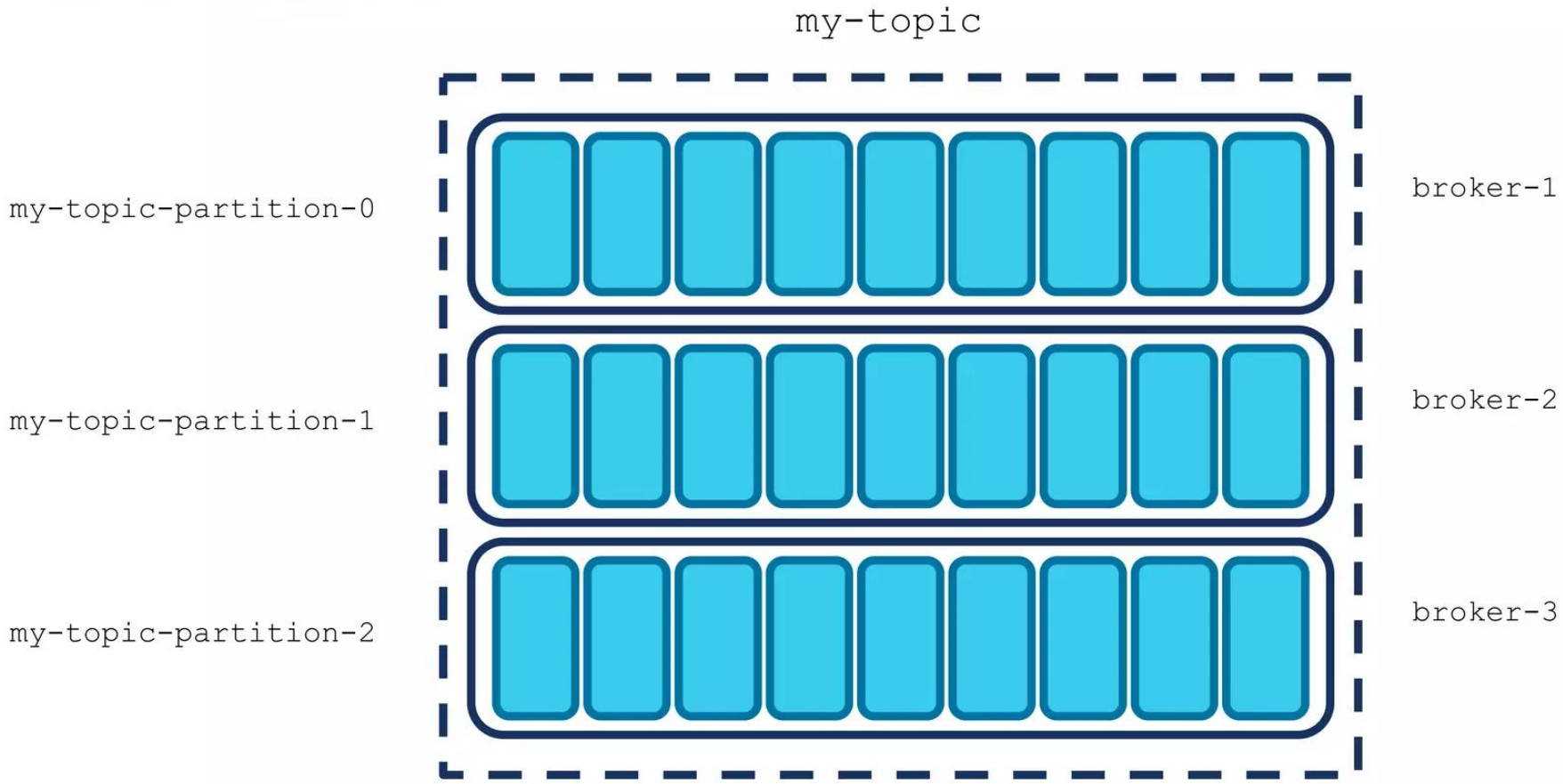



Writers
Kafka Cluster
Readers
Kafka Producer
Kafka Producer
Kafka Producer
Kafka Broker
Kafka Broker
Kafka Broker
Kafka Broker
Kafka Broker
Kafka Consumer
Kafka Consumer (grouped)
Kafka Consumer /Producer

KAFKA

A MODERN, DISTRIBUTED PLATFORM FOR DATA STREAMS

Kafka Topics

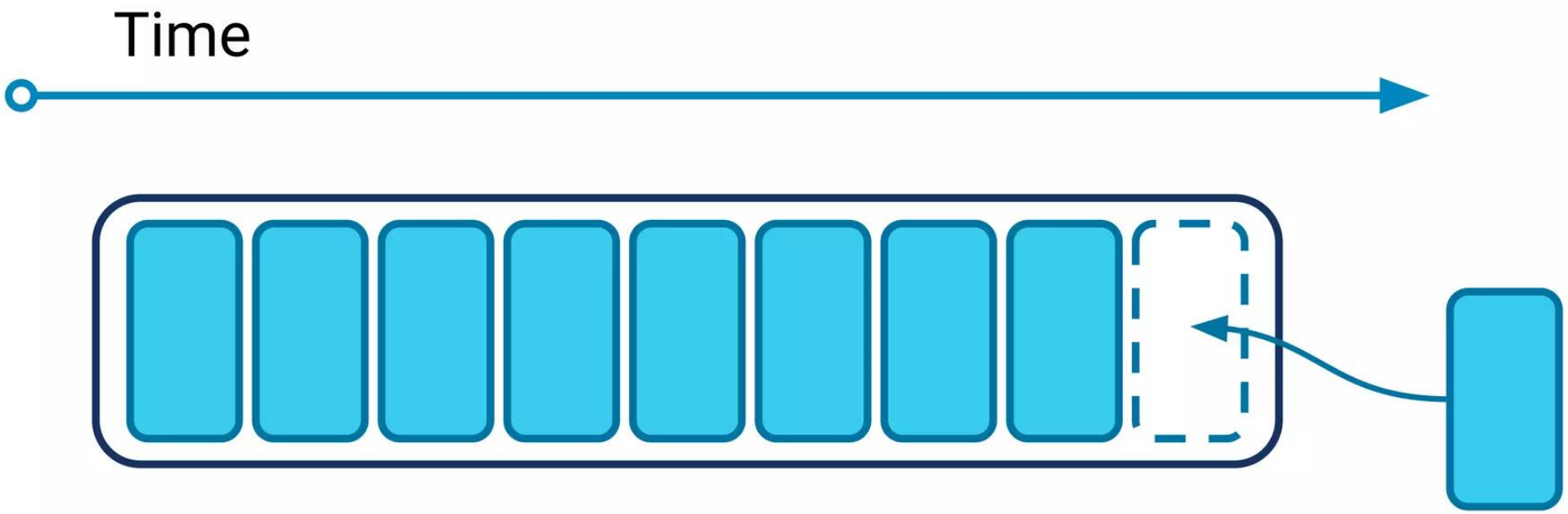


Creating a topic

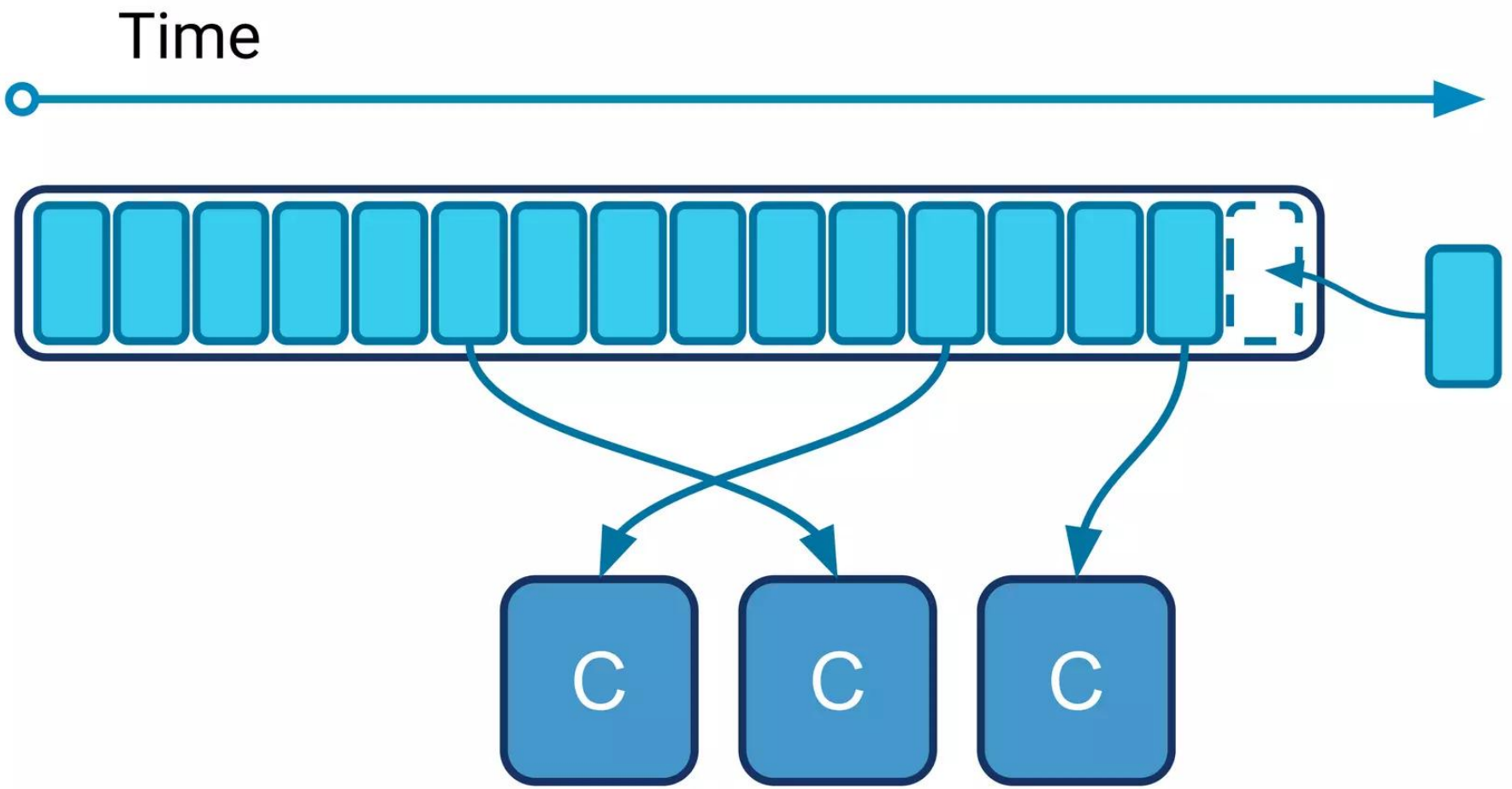
```
$ kafka-topics --zookeeper zk:2181\
    --create \
    --topic my-topic \
    --replication-factor 3 \
    --partitions 3
```

Or use the `AdminClient` API!

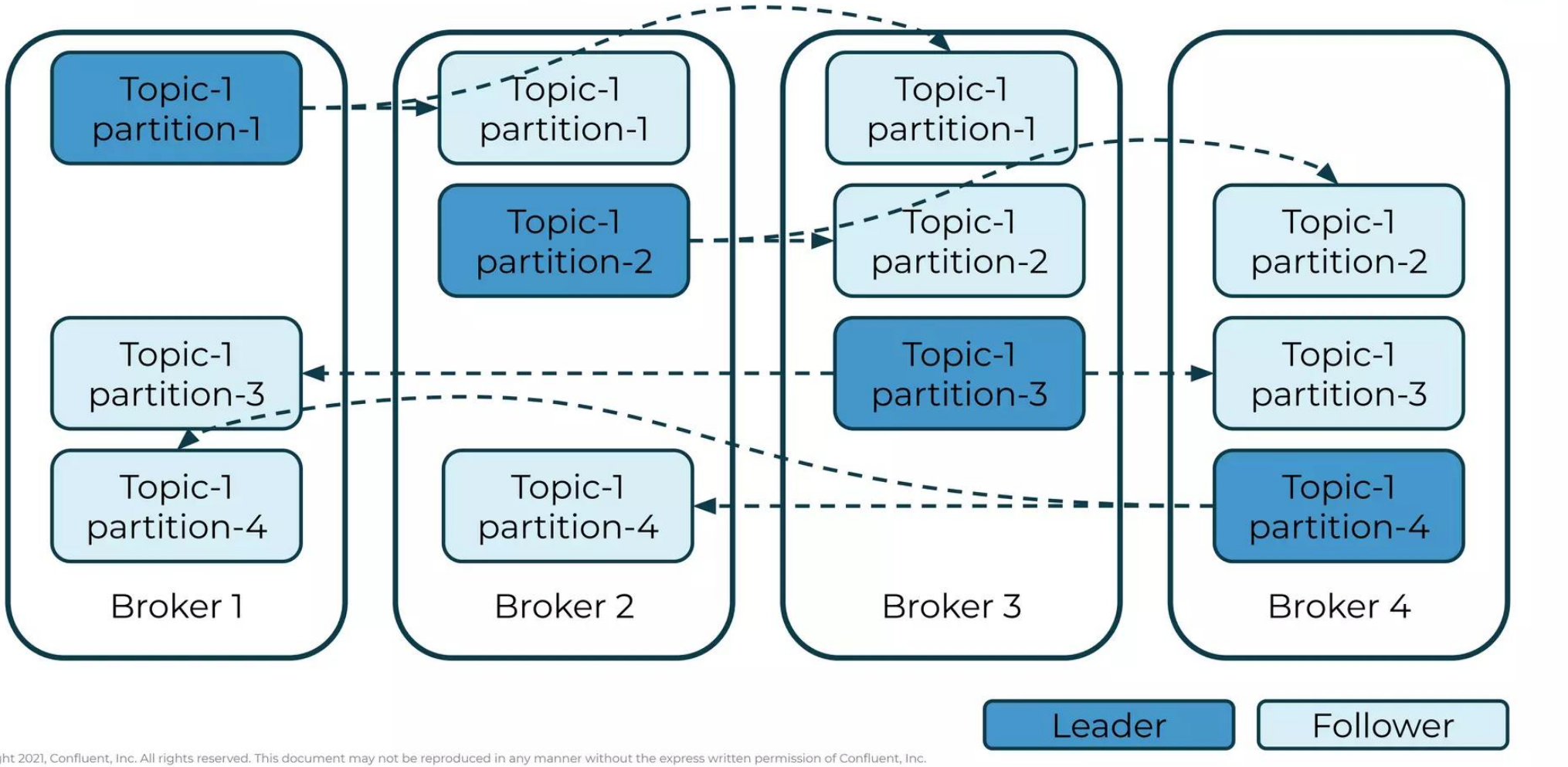
Producing to Kafka



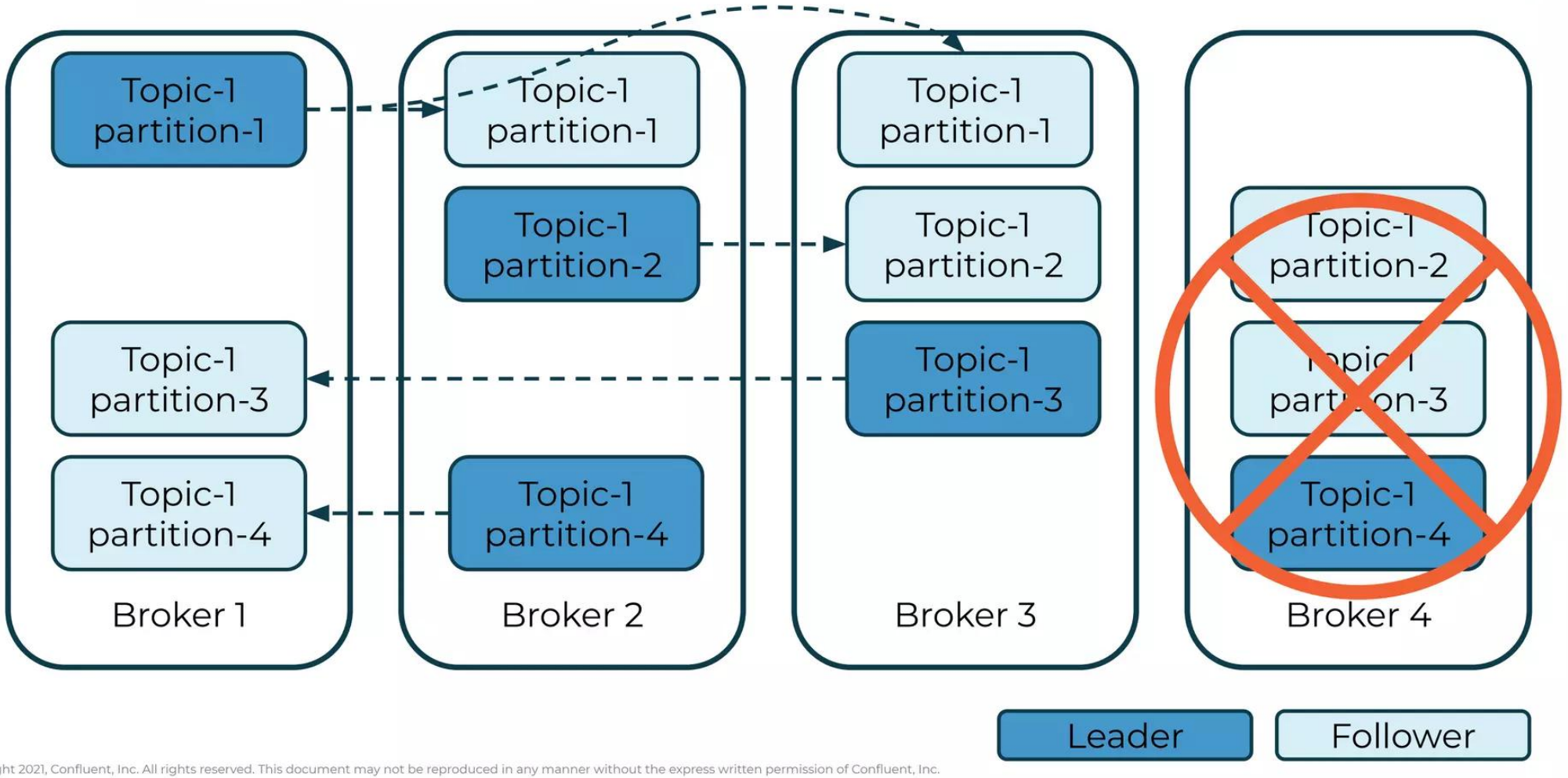
Producing to Kafka



Kafka's distributed nature

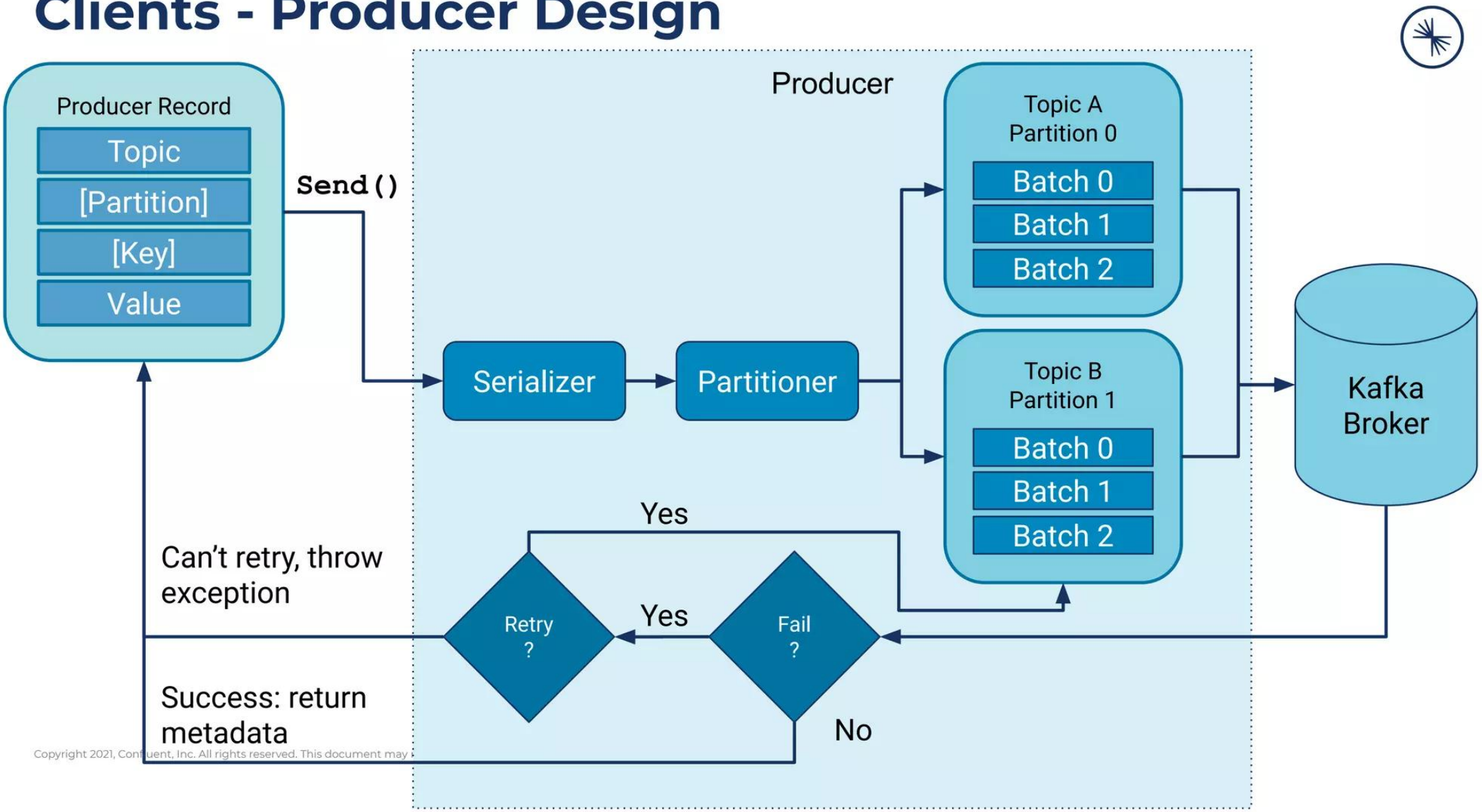


Kafka's distributed nature



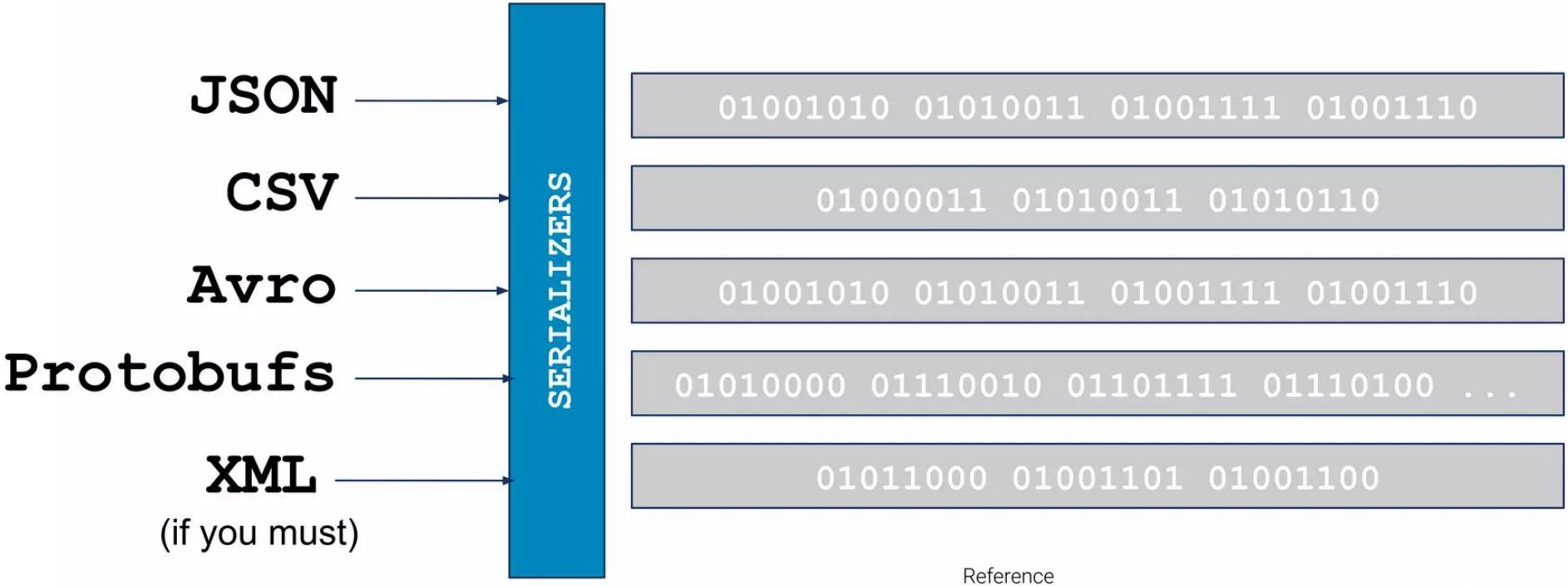
Producing to Kafka

Clients - Producer Design



The Serializer

Kafka doesn't care about what you send to it as long as it's been converted to a byte stream beforehand.



Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

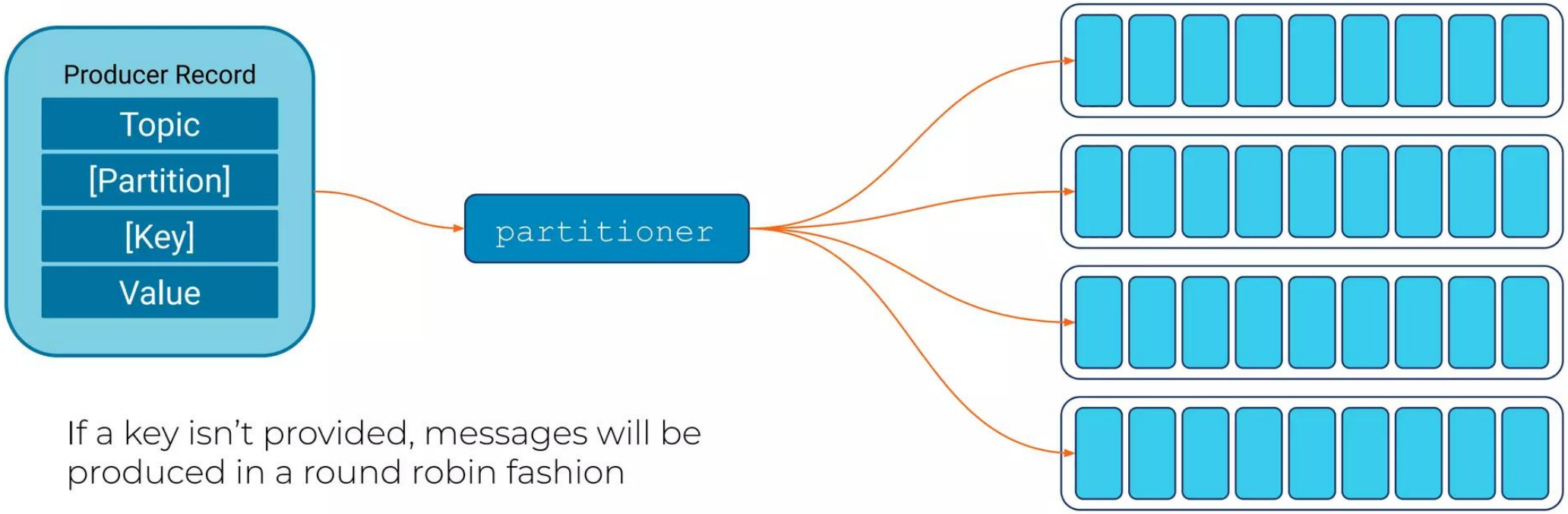
The Serializer

```
private Properties kafkaProps = new Properties();

kafkaProps.put("bootstrap.servers", "broker1:9092,broker2:9092");
kafkaProps.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
kafkaProps.put("value.serializer", "io.confluent.kafka.serializers.KafkaAvroSerializer");
kafkaProps.put("schema.registry.url", "https://schema-registry:8083");

producer = new KafkaProducer<String, SpecificRecord>(kafkaProps);
```

Reference
https://kafka.apache.org/10/documentation/streams/developer-guide/datatypes.html

Record Keys and why they're important - Ordering

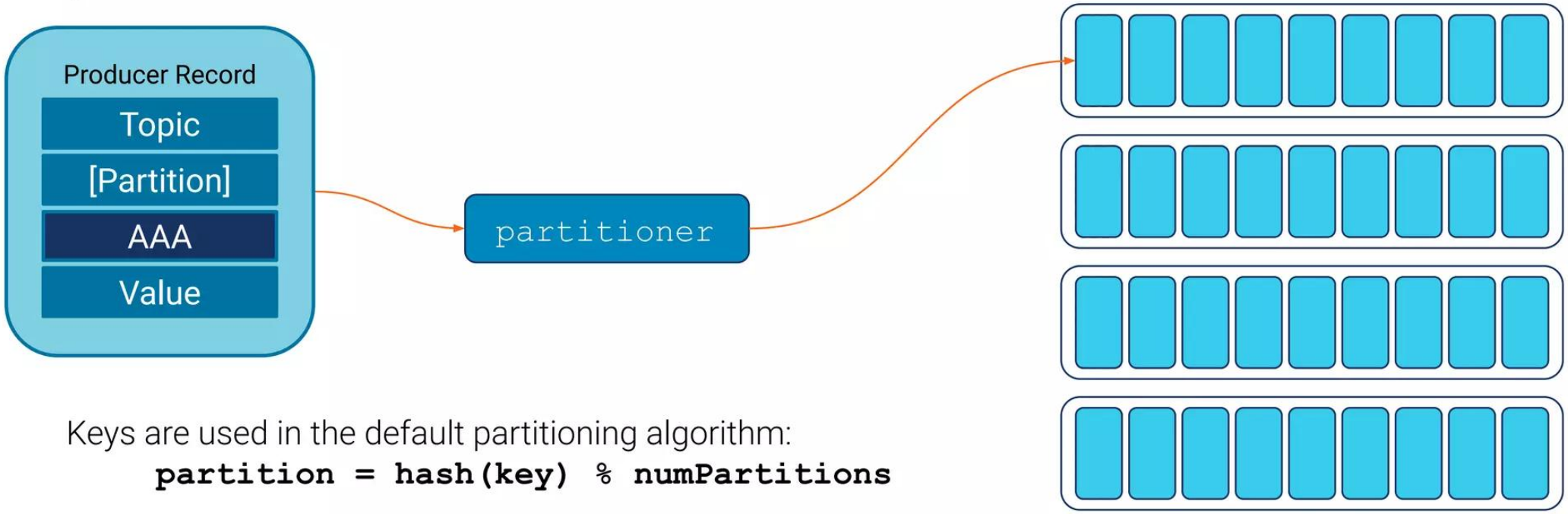
Record keys determine the partition with the default kafka partitioner



If a key isn't provided, messages will be produced in a round robin fashion

Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner

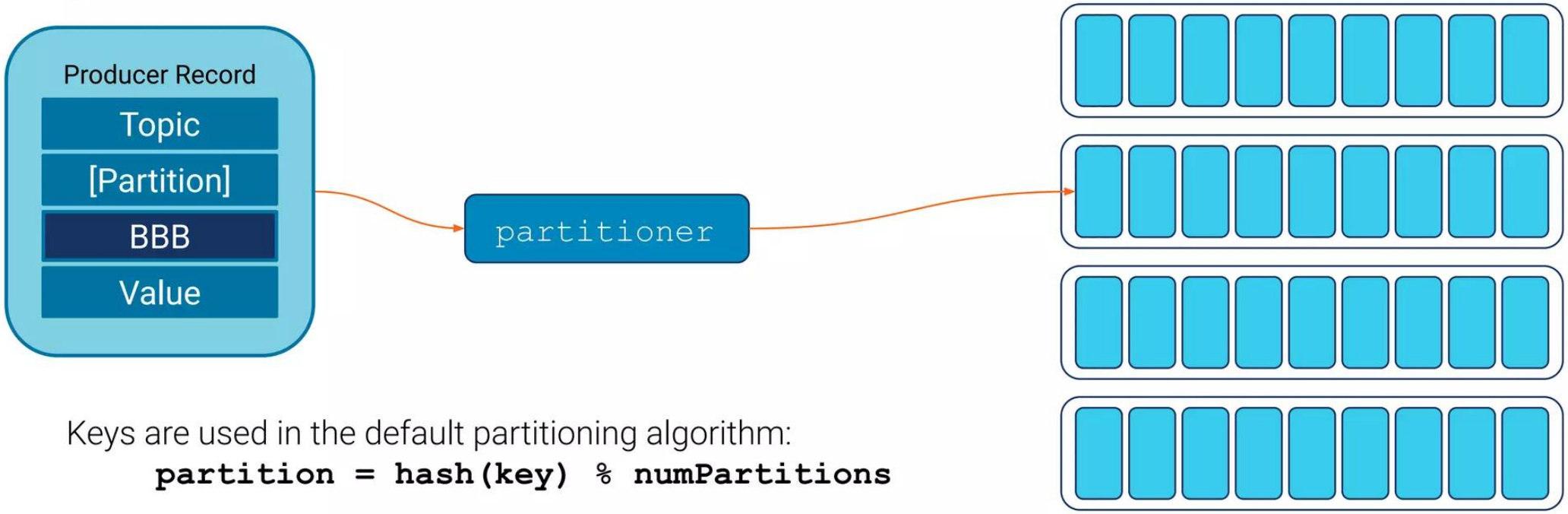


Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner

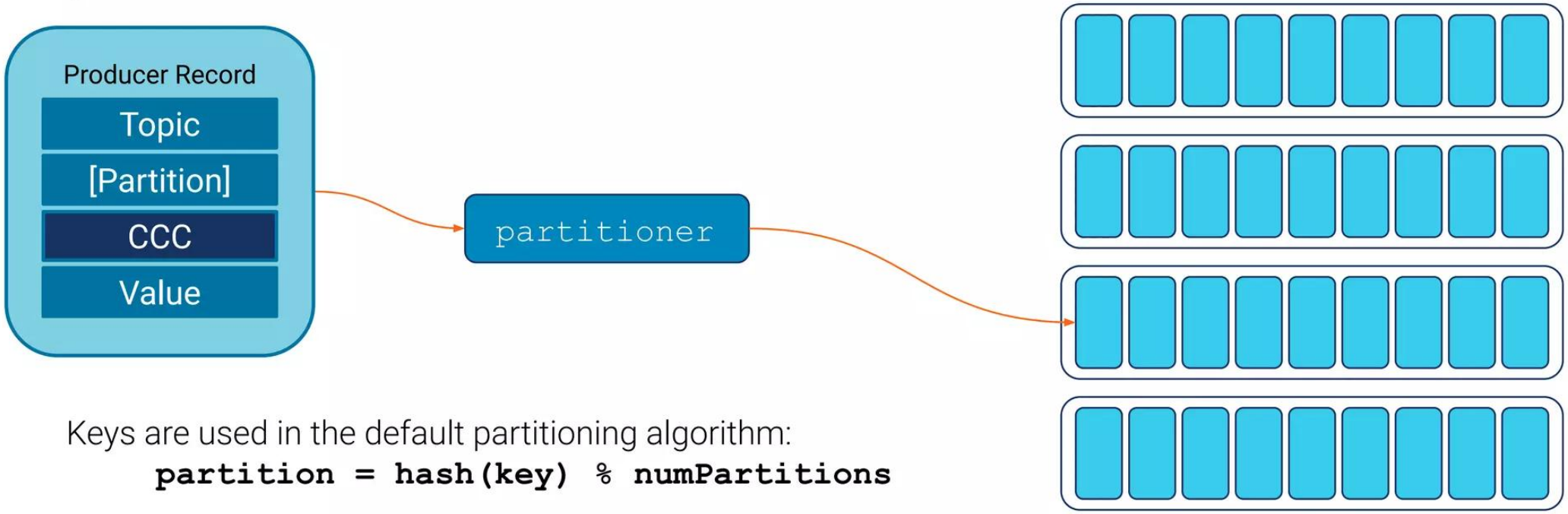


Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner

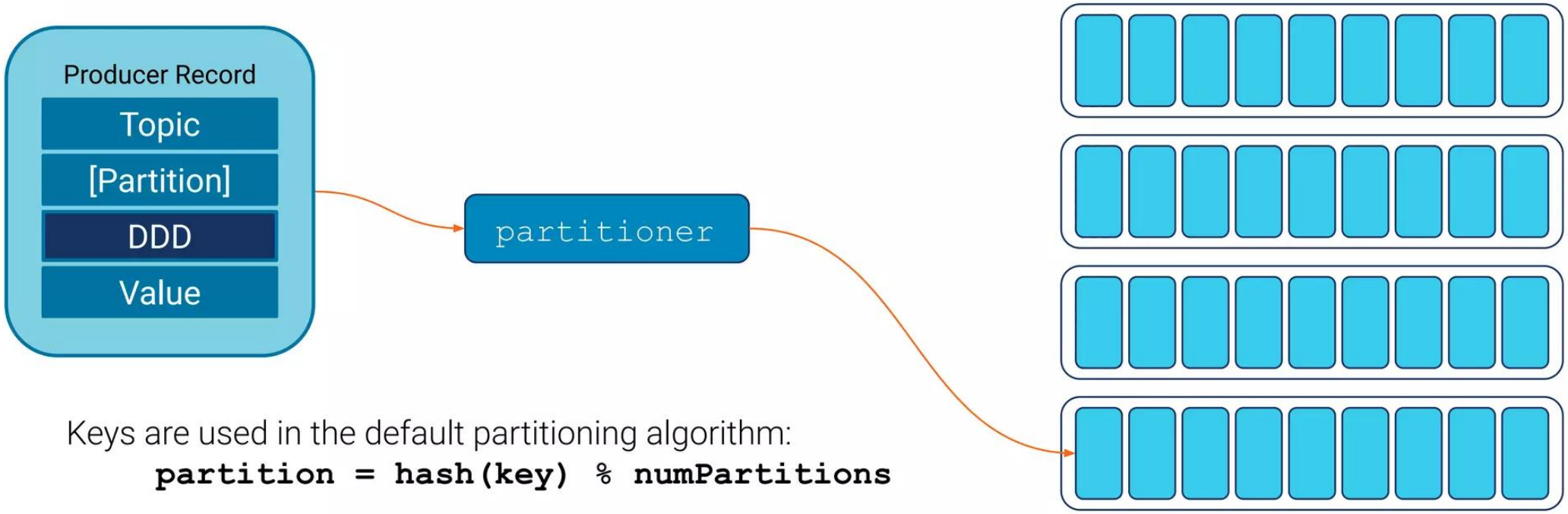


Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Ordering

Record keys determine the partition with the default kafka partitioner



Keys are used in the default partitioning algorithm:

```
partition = hash(key) % numPartitions
```

Record Keys and why they're important - Key

Car·di·nal·i·ty

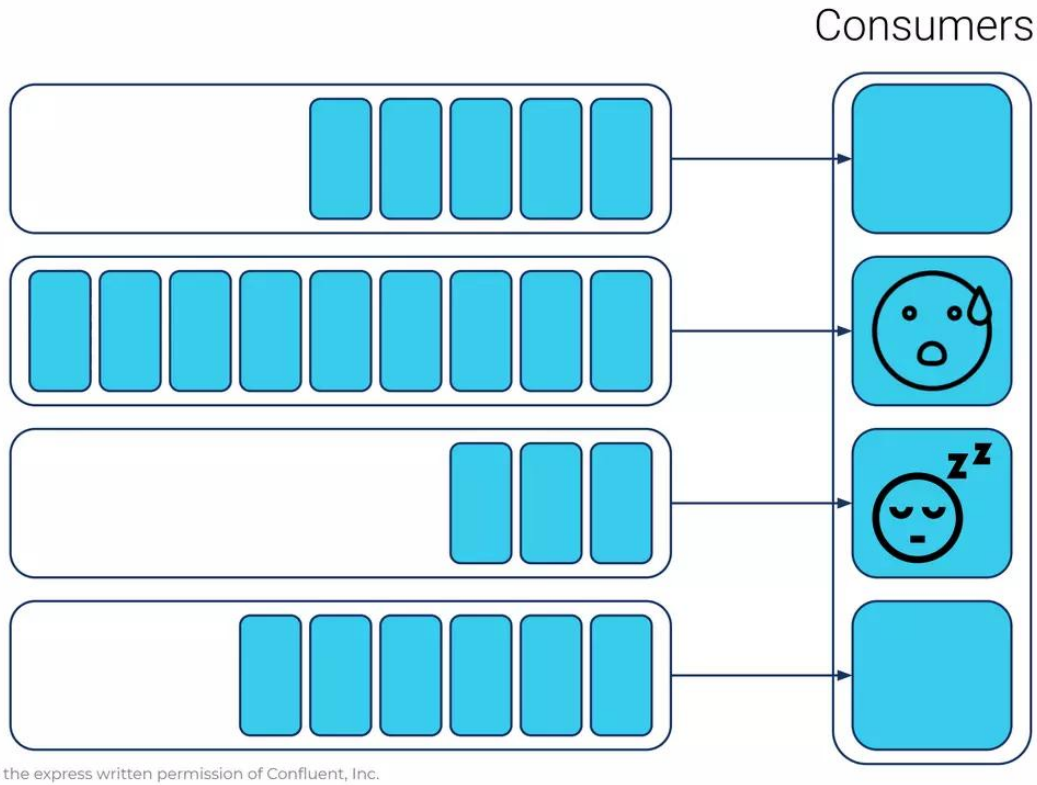
/ˌkärdəˈnalədē/

Noun

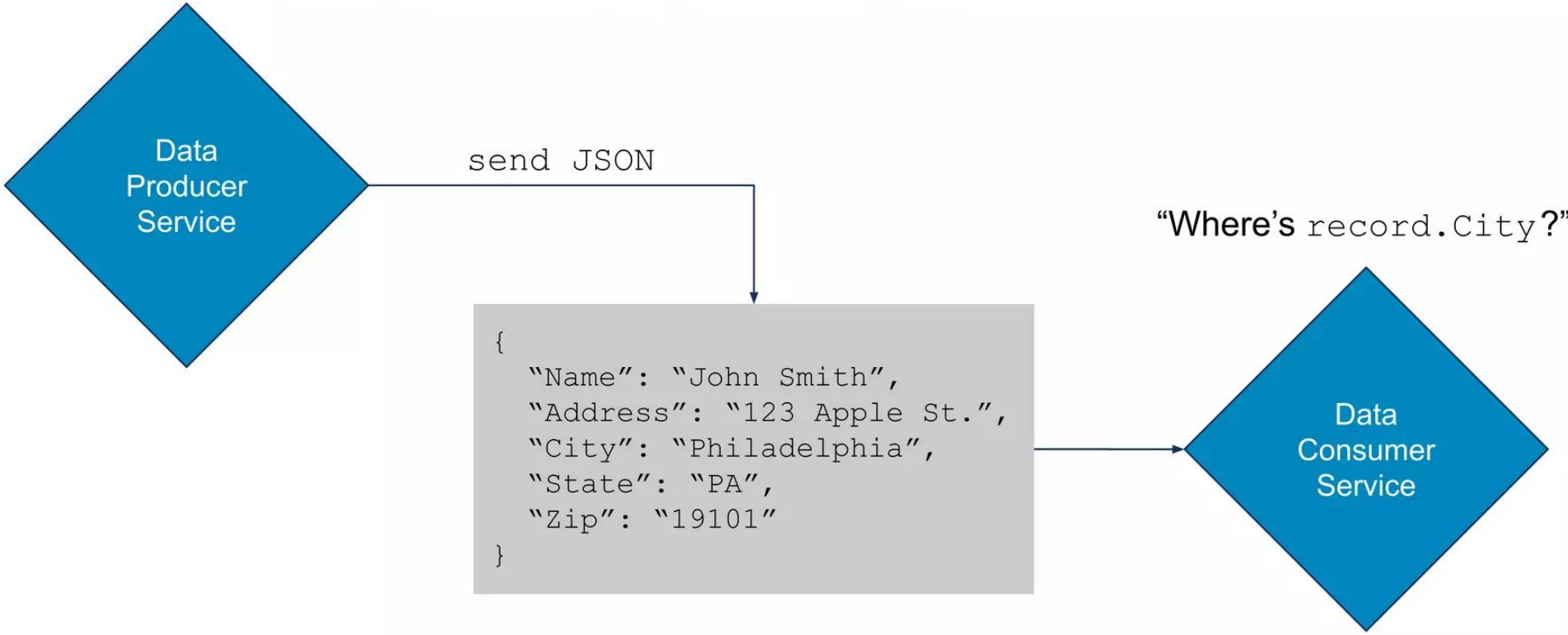
the number of elements in a set or other grouping, as a property of that grouping.

Key cardinality affects the amount of work done by the individual consumers in a group. Poor key choice can lead to uneven workloads.

Keys in Kafka don't have to be primitives, like strings or ints. Like values, they can be be anything: JSON, Avro, etc… So create a key that will evenly distribute groups of records around the partitions.



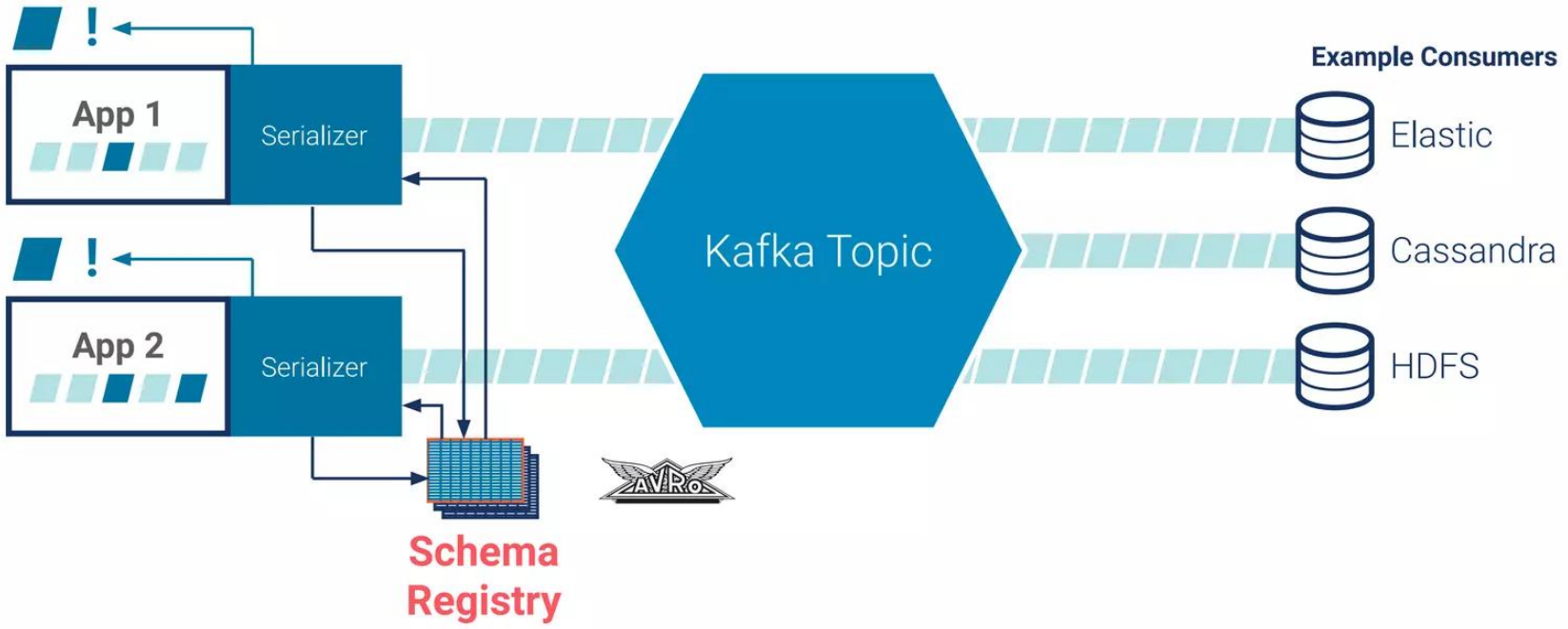
You don’t have to but… use a Schema!



Reference
https://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one/

Schema Registry: Make Data Backwards Compatible and Future-Proof

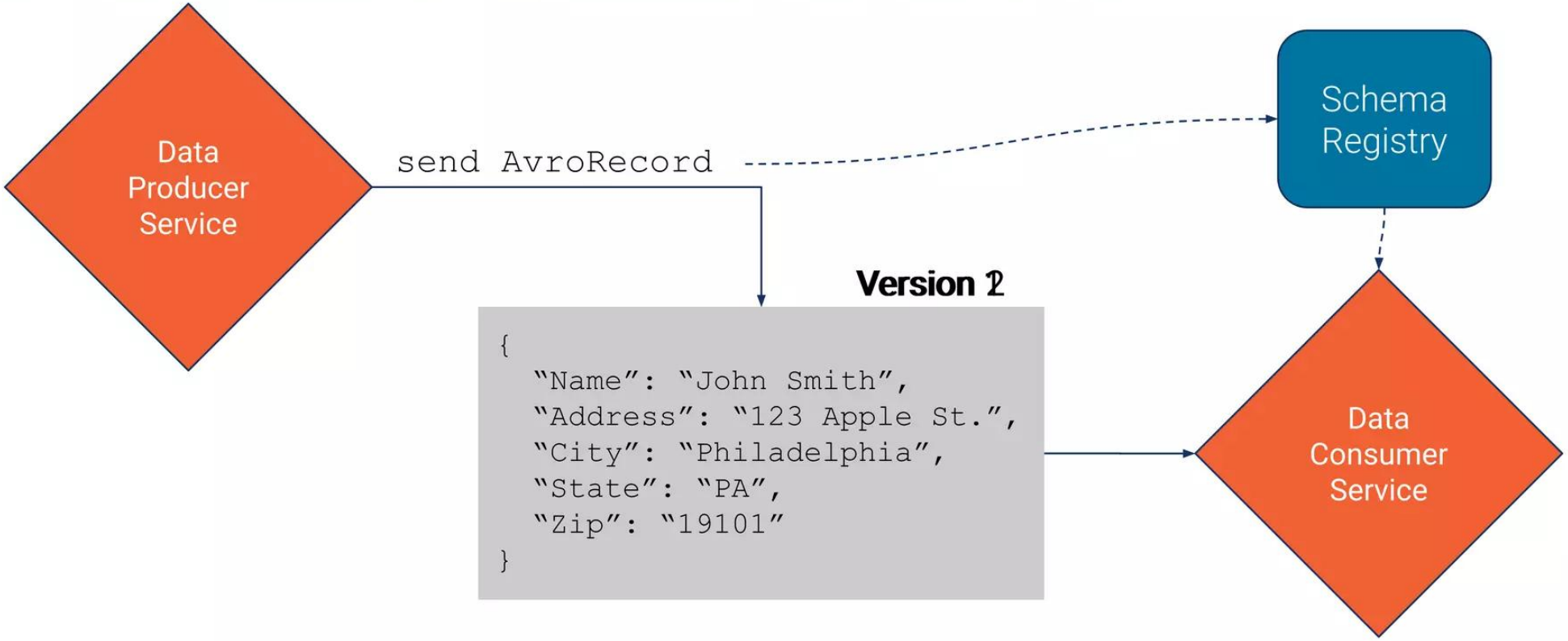
Open Source Feature



- **Define** the expected fields for each Kafka topic
- Automatically **handle** schema changes (e.g. new fields)

- **Prevent** backwards incompatible changes
- **Support** multi-data center environments

Avro allows for evolution of schemas



Reference
https://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one/

Developing with Confluent Schema Registry

We provide several Maven plugins for developing with the Confluent Schema Registry

- **`download`** - download a subject's schema to your project
- **`register`** - register a new schema to the schema registry from your development env
- **`test-compatibility`** - test changes made to a schema against compatibility rules set by the schema registry

```
<plugin>
    <groupId>io.confluent</groupId>
<artifactId>kafka-schema-registry-maven-plug
    <version>5.0.0</version>
    <configuration>
        <schemaRegistryUrls>
            <param>http://192.168.99.100:808
        </schemaRegistryUrls>
        <outputDirectory>src/main/avro</outp
        <subjectPatterns>
            <param>^TestSubject000-(key|valu
        </subjectPatterns>
    </configuration>
</plugin>
```

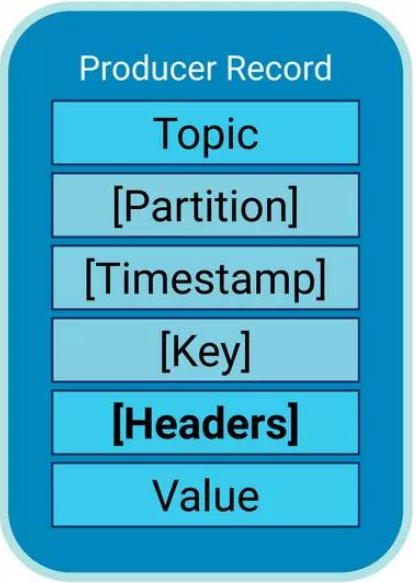
Reference
https://docs.confluent.io/current/schema-registry/docs/maven-plugin.html

Use Kafka's Headers

Kafka Headers are simply an interface that requires a key of type `String`, and a value of type `byte[]`, the headers are stored in an iterator in the `ProducerRecord`.

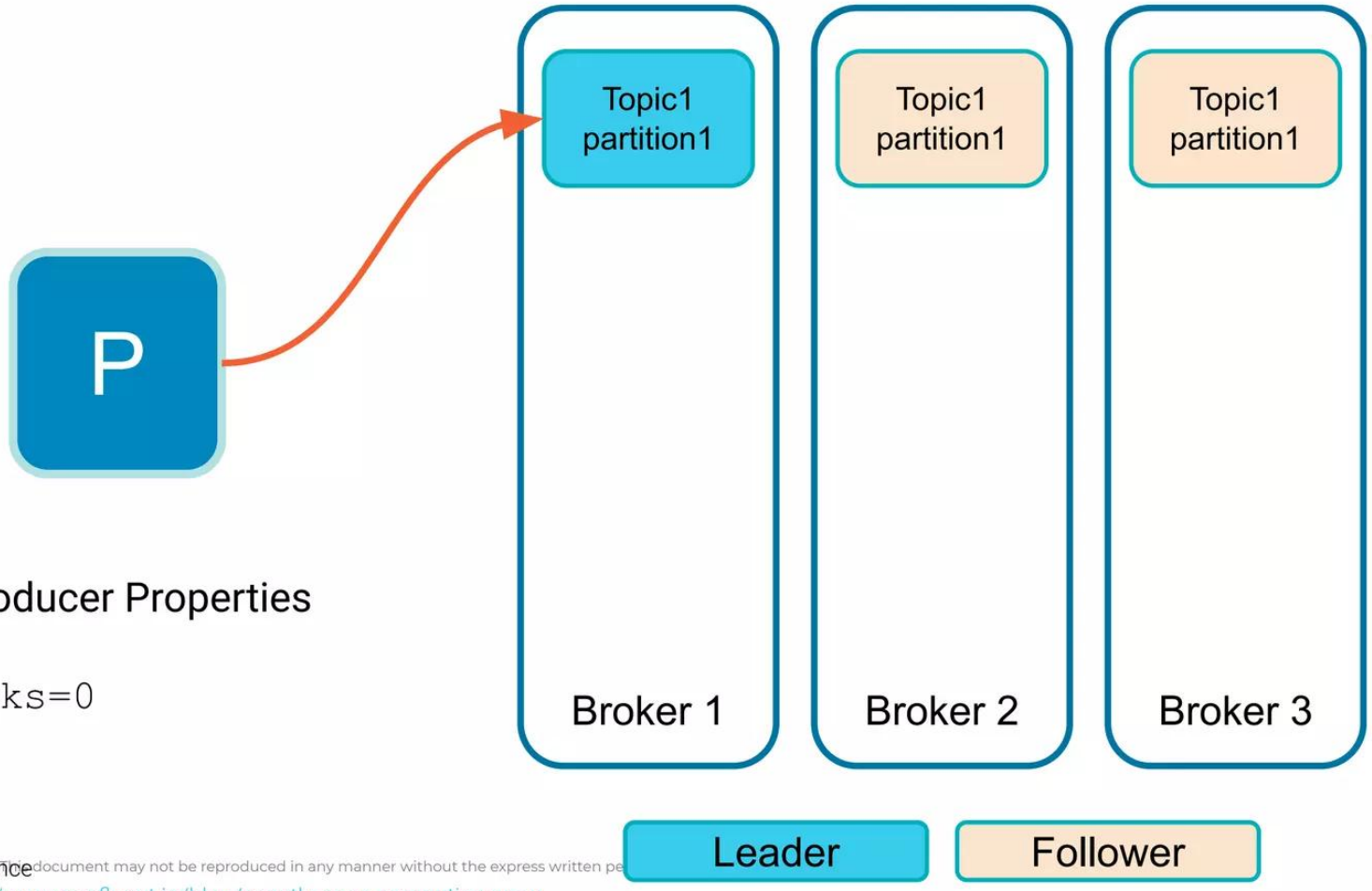
Example Use Cases

- Data lineage: reference previous topic partition/offsets
- Producing host/application/owner
- Message routing
- Encryption metadata (which key pair was this message payload encrypted with?)



Reference
https://cwiki.apache.org/confluence/display/KAFKA/KIP-82+-+Add+Record+Headers

Producer Guarantees



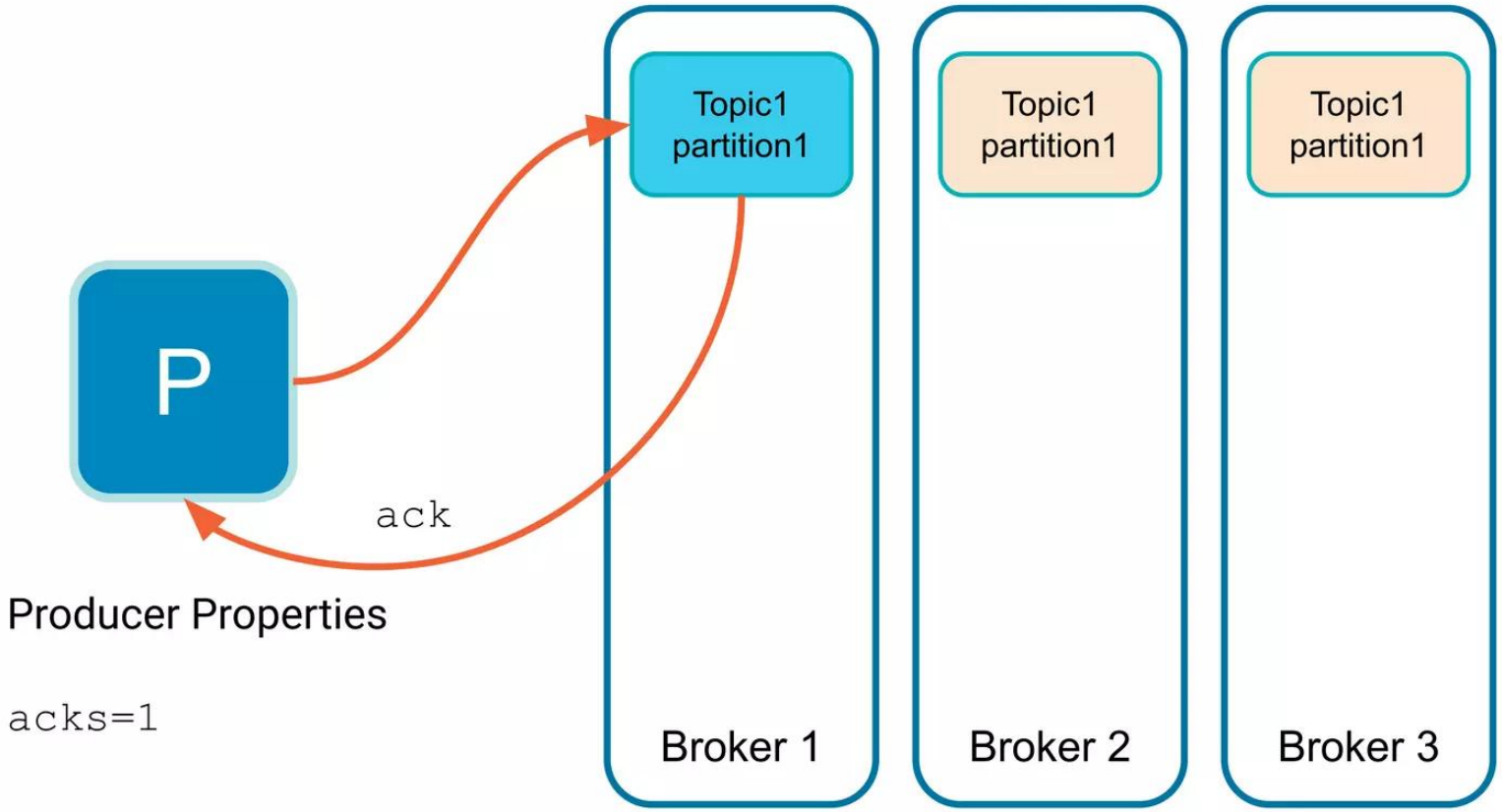
Producer Properties

```
acks=0
```


Reference

https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

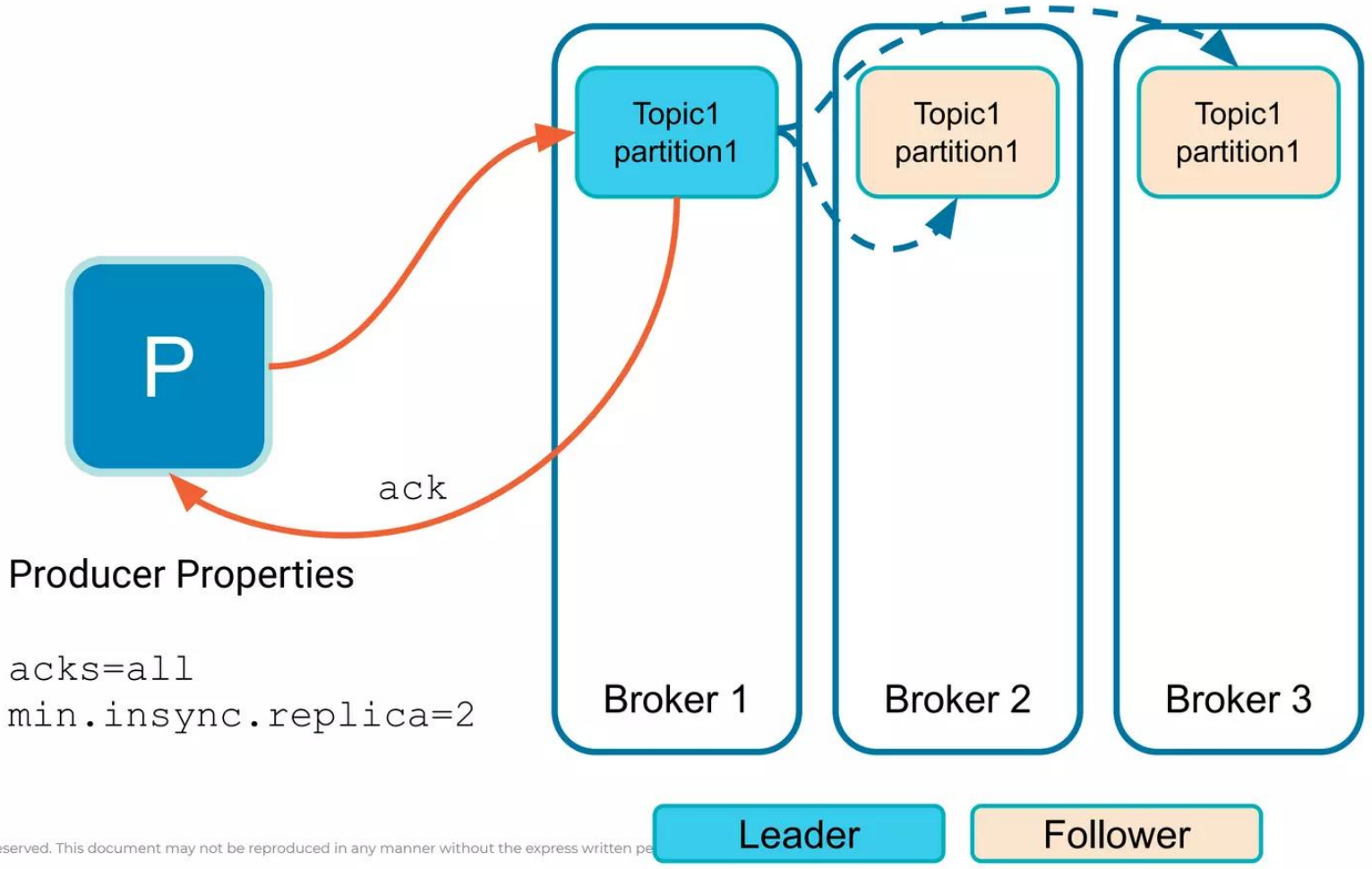
Producer Guarantees



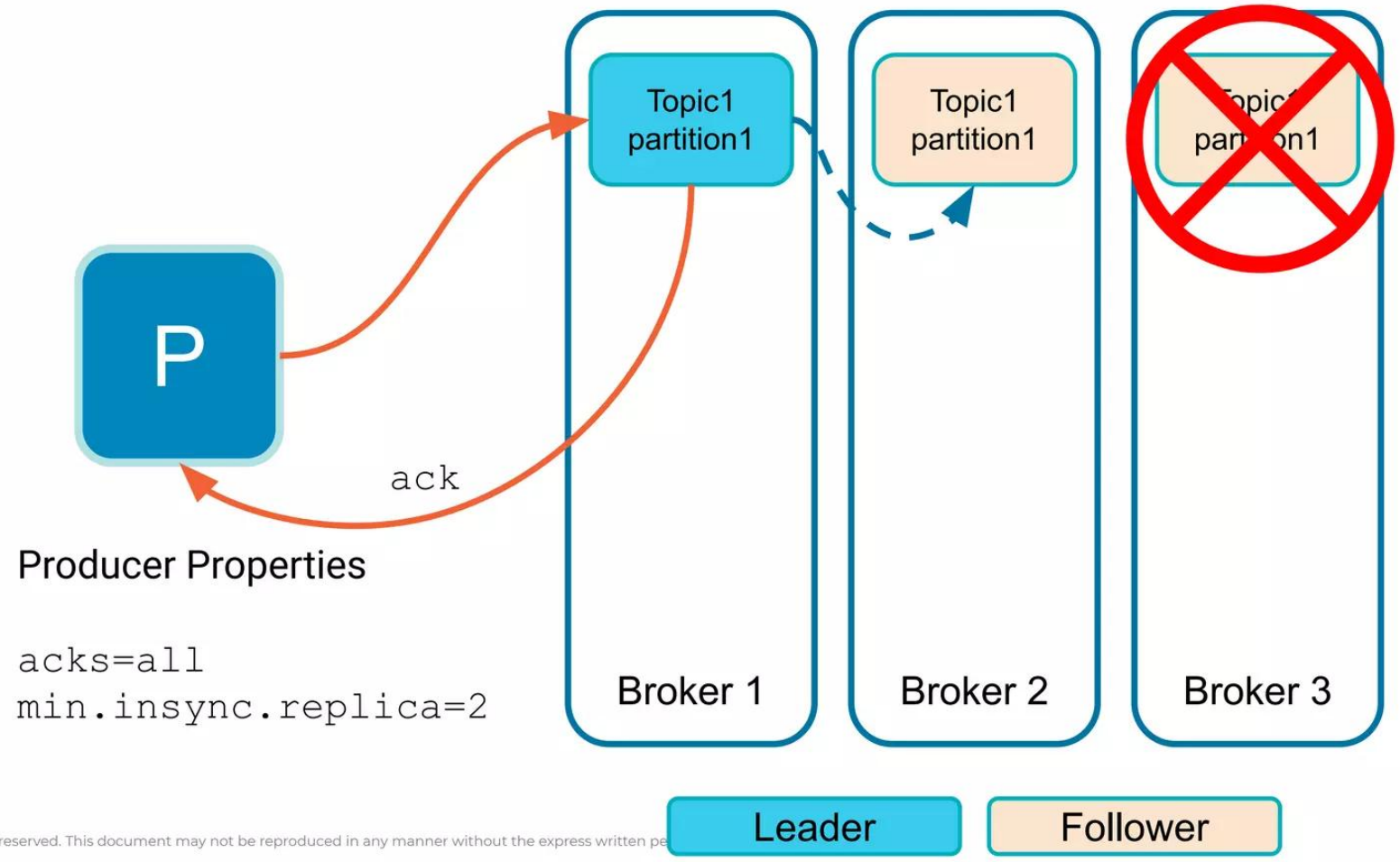
Reference

https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

Producer Guarantees

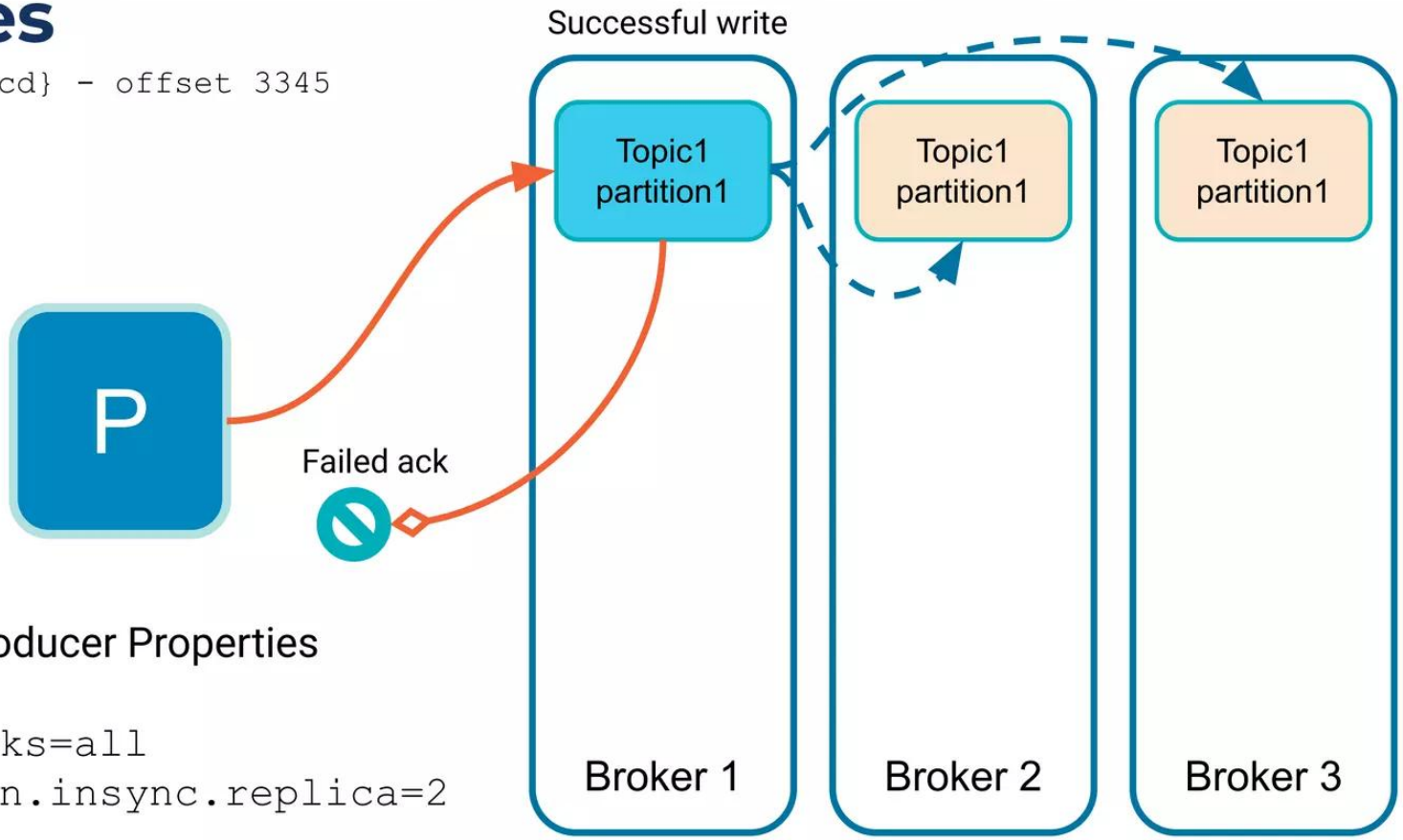


Producer Guarantees



Producer Guarantees - without exactly once guarantees

{key: 1234 data: abcd} - offset 3345



Producer Properties

```
acks=all
min.insync.replica=2
```

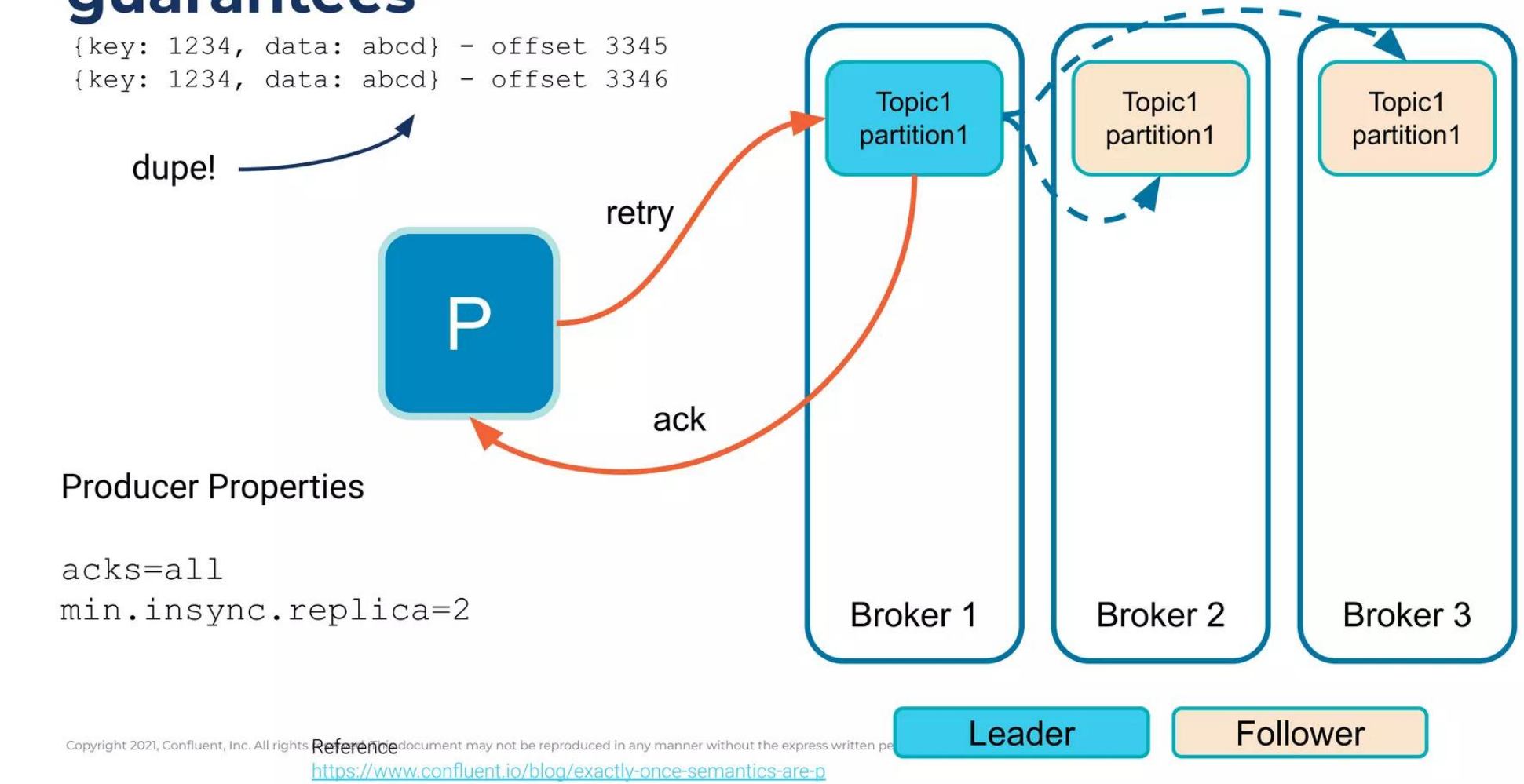
Leader | Follower

Reference

https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

Producer Guarantees - without exactly once guarantees


```
{key: 1234, data: abcd} - offset 3345
{key: 1234, data: abcd} - offset 3346
```



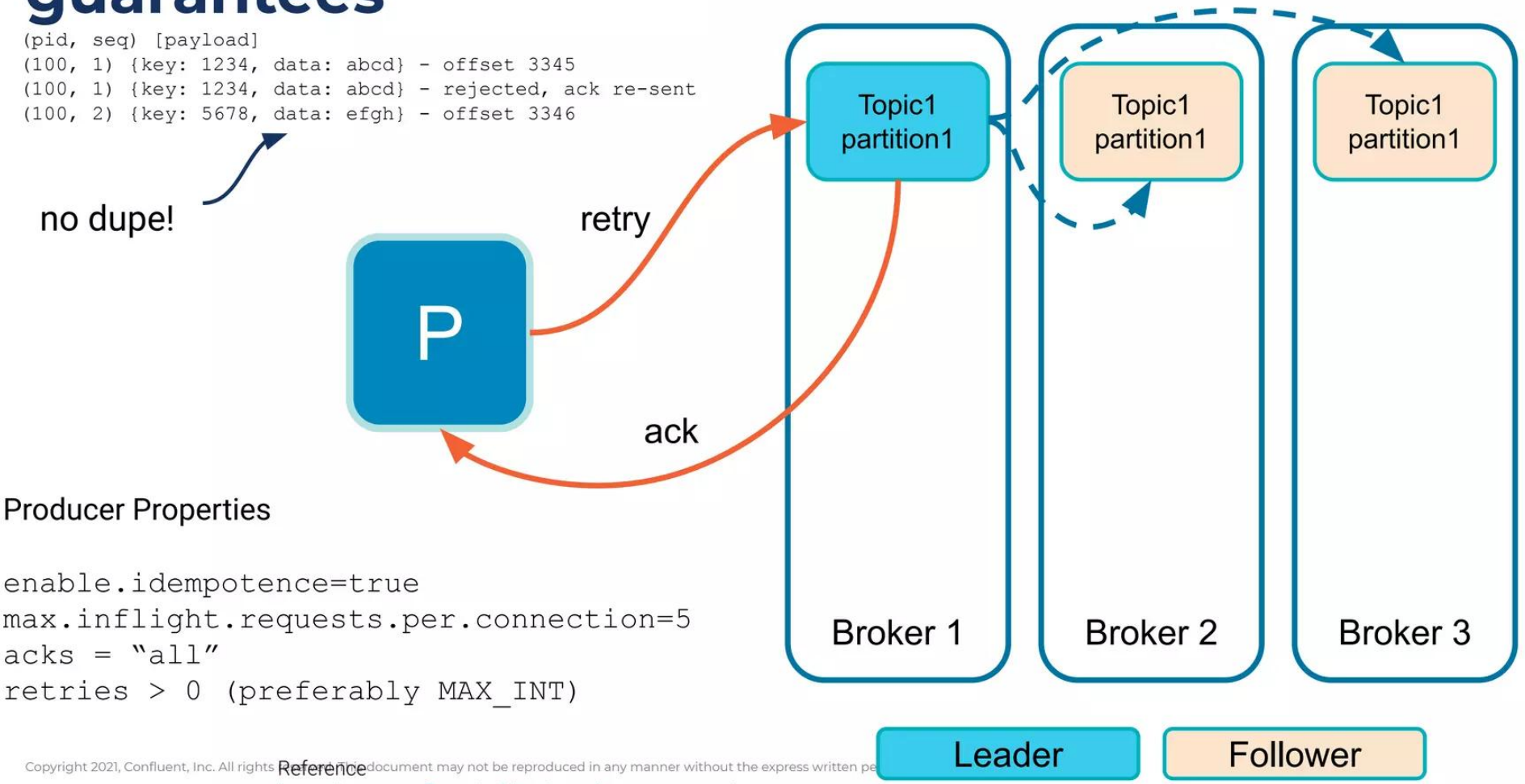
Producer Properties

```
acks=all
min.insync.replica=2
```


Reference

https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

Producer Guarantees - with exactly once guarantees

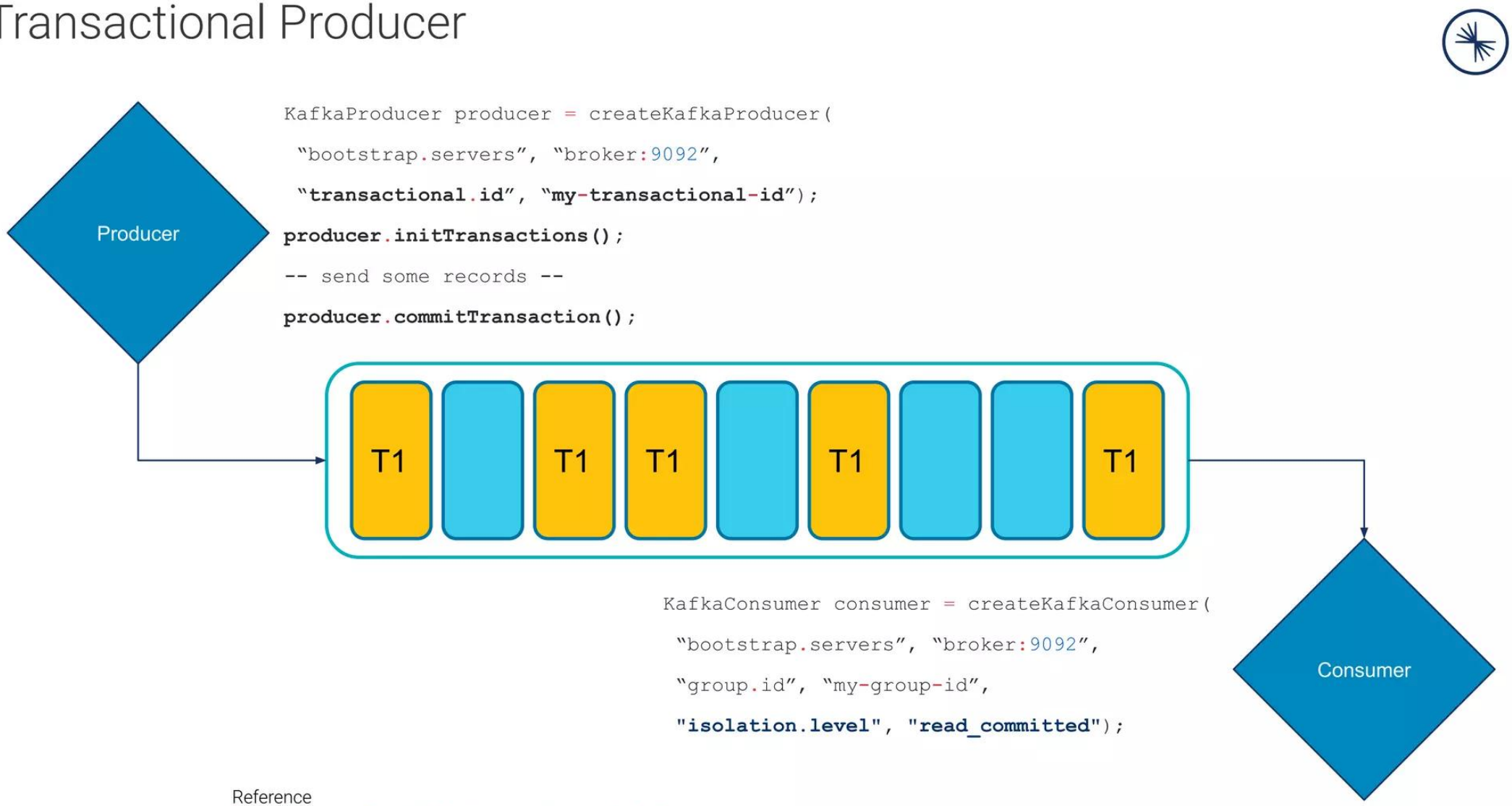


Producer Properties

```
enable.idempotence=true
max.inflight.requests.per.connection=5
acks = "all"
retries > 0 (preferably MAX_INT)
```


Reference
https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

Transactional Producer



Reference
https://www.confluent.io/blog/transactions-apache-kafka/

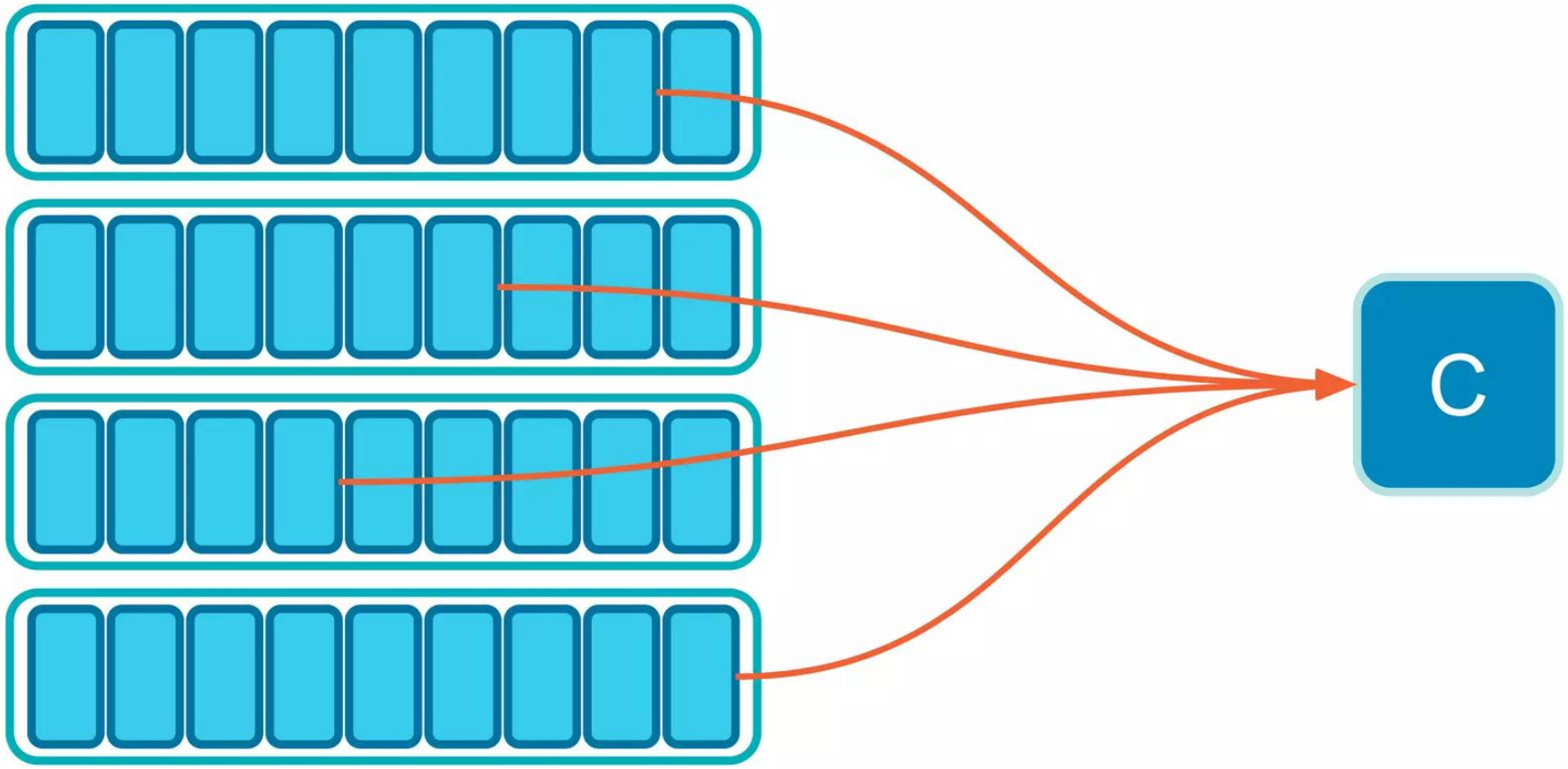
CONFLUENT

Consuming from Kafka

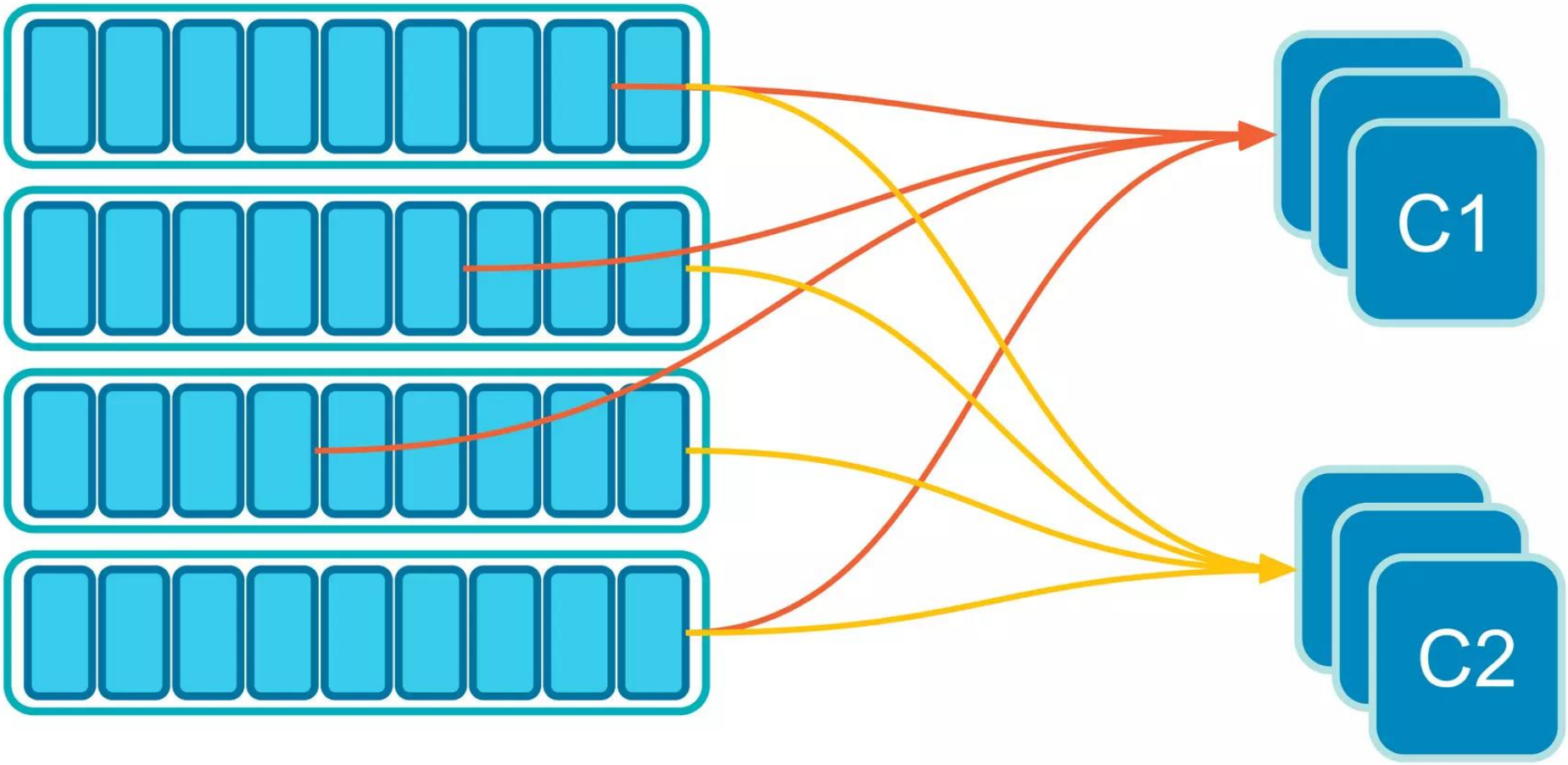
A basic Java Consumer

```
  final Consumer<String, String> consumer = new KafkaConsumer<String,
String>(props);
  consumer.subscribe(Arrays.asList(topic));
  try {
    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records) {
        -- Do Some Work --
      }
    }
  } finally {
    consumer.close();
  }
}
```

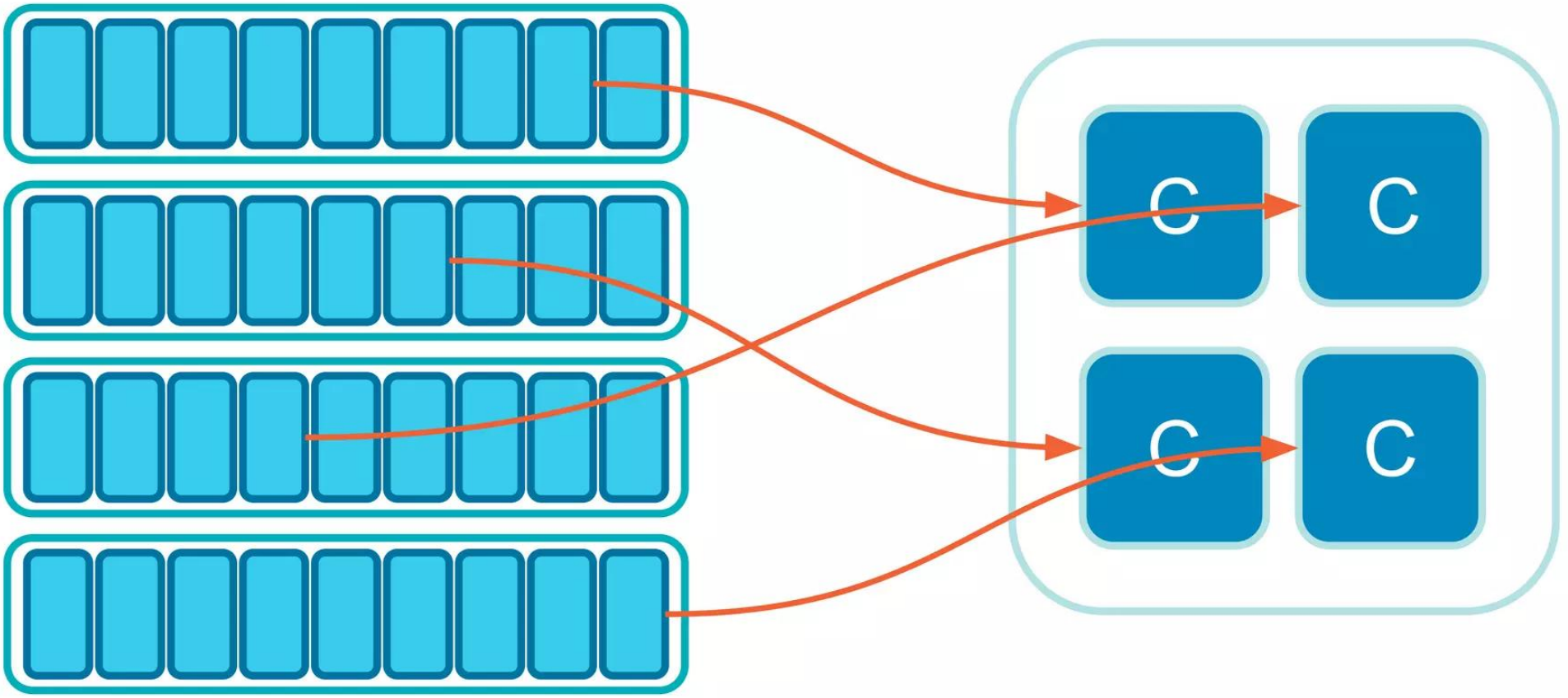
Consuming From Kafka - Single Consumer



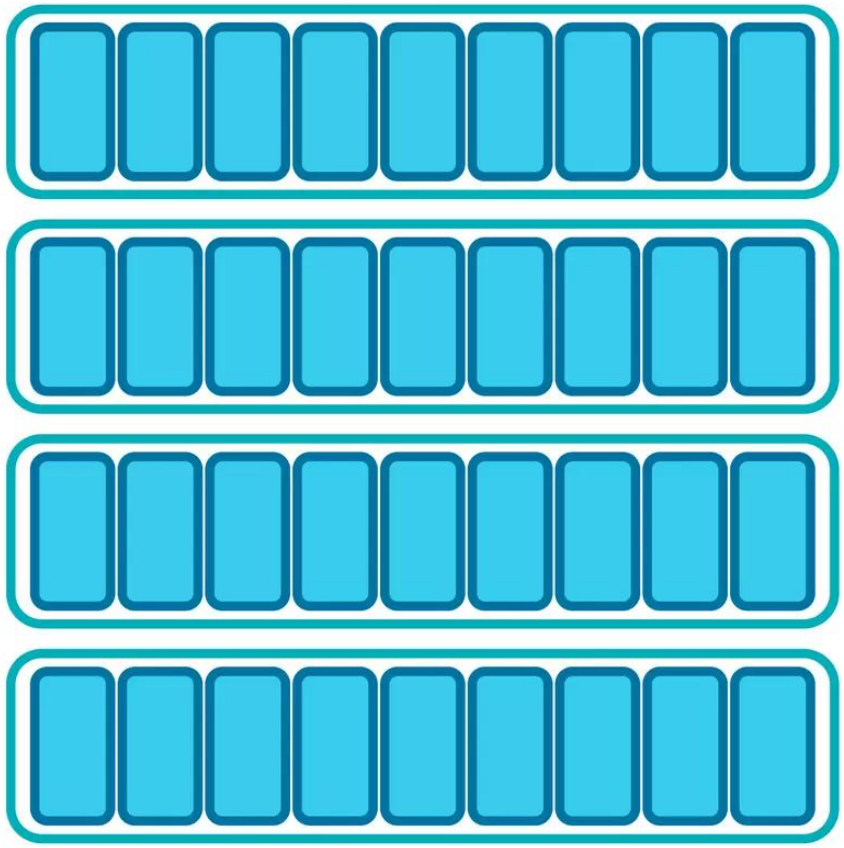
Consuming From Kafka - Grouped Consumers

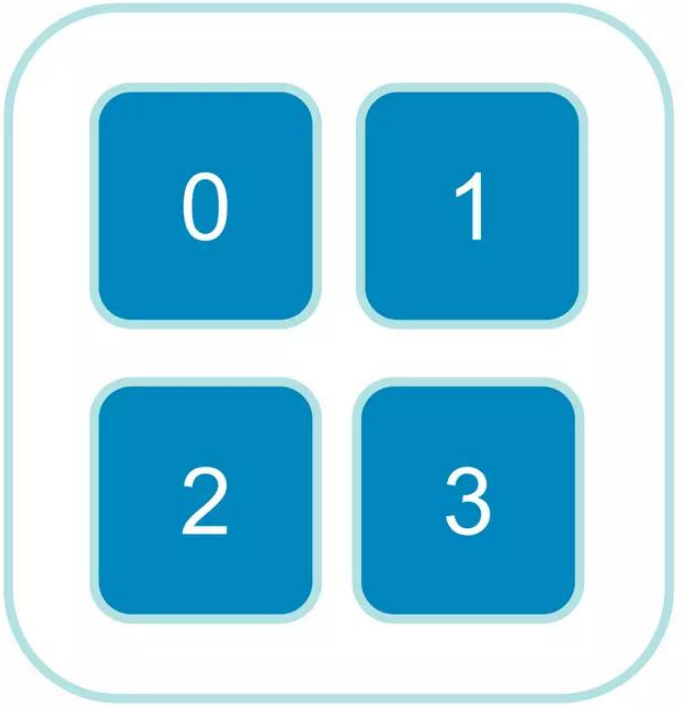


Consuming From Kafka - Grouped Consumers

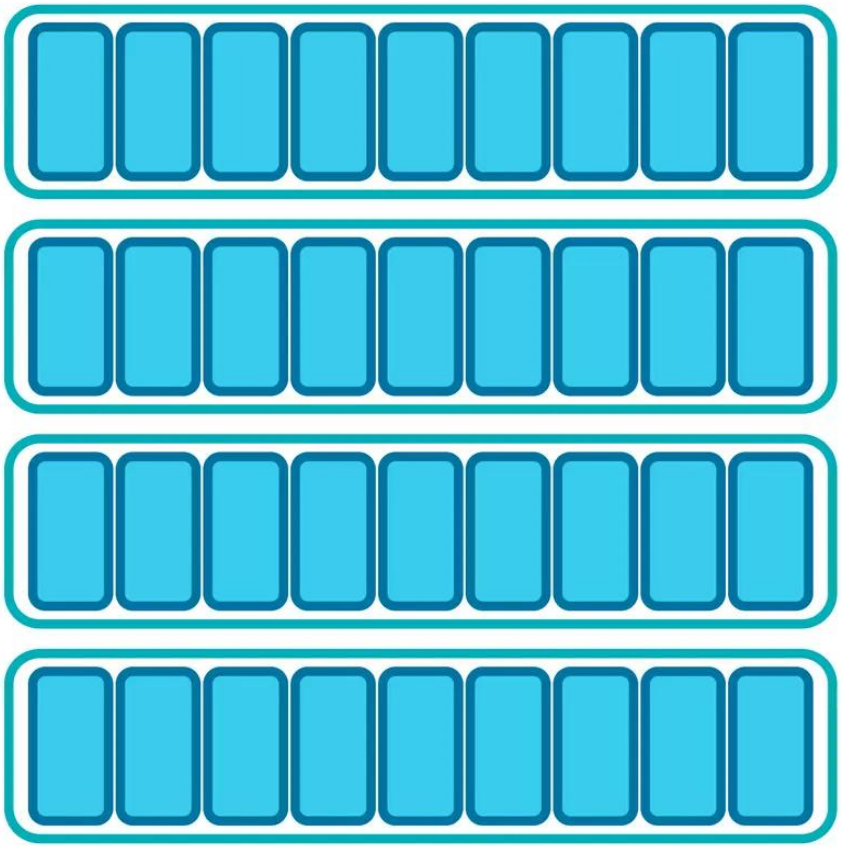


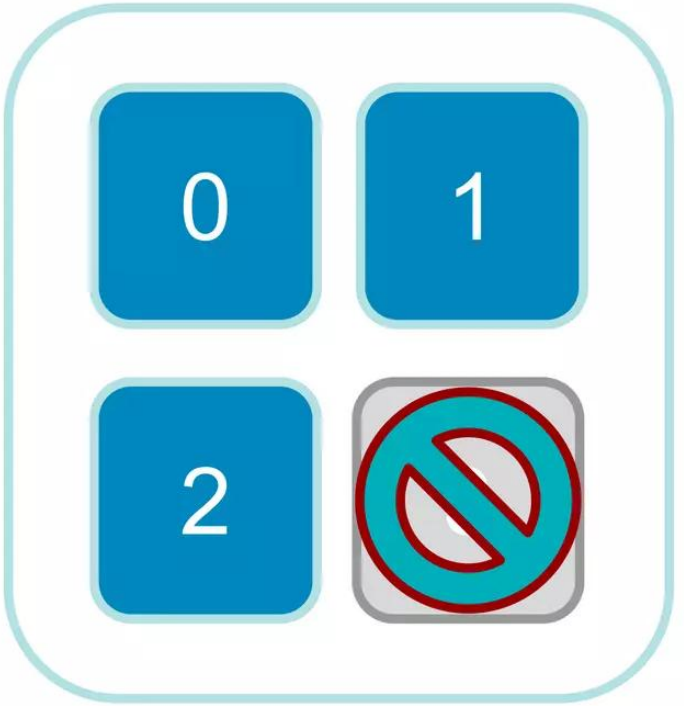
Consuming From Kafka - Grouped Consumers



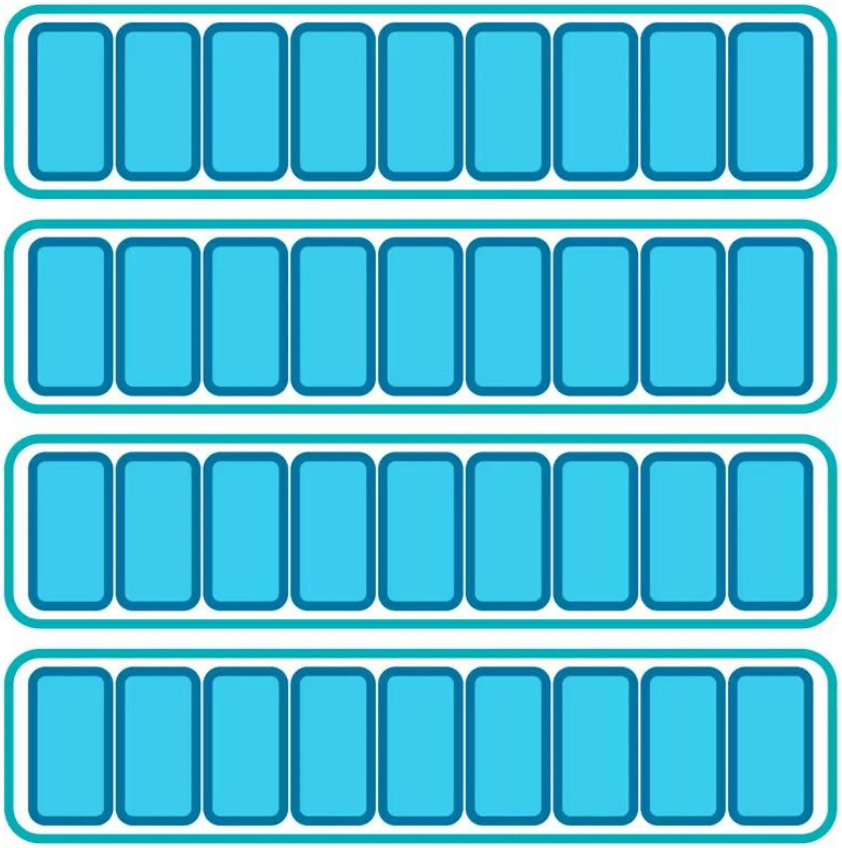


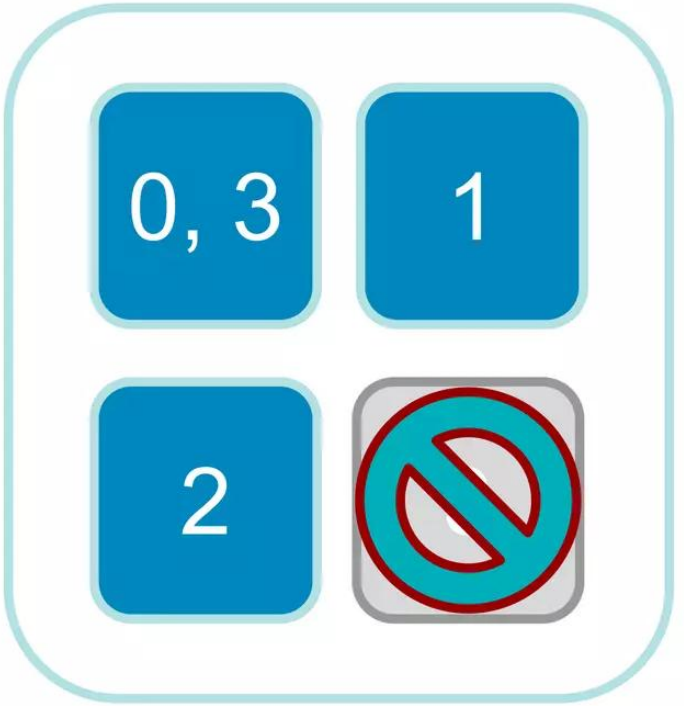
Consuming From Kafka - Grouped Consumers





Consuming From Kafka - Grouped Consumers





Kafka's Interceptors

ProducerInterceptor

onSend(ProducerRecord<K, V> record)

Returns `ProducerRecord<K, V>`. Called from `send()` before key and value get serialized and partition is assigned. This method is allowed to modify the record.

onAcknowledgement(RecordMetadata metadata, java.lang.Exception exception)

This method is called when the record sent to the server has been acknowledged, or when sending the record fails before it gets sent to the server.

Used for observability and reporting.

Reference
https://kafka.apache.org/20/javadoc/org/apache/kafka/clients/producer/ProducerInterceptor.html

ConsumerInterceptor

onConsume(ConsumerRecords<K,V> records)

Called just before the records are returned by `KafkaConsumer.poll()`

This method is allowed to modify consumer records, in which case the new records will be returned.

onCommit(Map<TopicPartition,OffsetAndMetadata> offsets)

This is called when offsets get committed.

Used for observability and reporting

Reference
https://kafka.apache.org/20/javadoc/org/apache/kafka/clients/consumer/ConsumerInterceptor.html

Should I pool connections?

NO!

Since Kafka connections are long-lived, there is no reason to pool connections. It’s common to keep one connection per thread.

Use a good client!

Clients

- Java/Scala - default clients, comes with Kafka
- C/C++ - https://github.com/edenhill/librdkafka
- C#/.Net - https://github.com/confluentinc/confluent-kafka-dotnet
- Python - https://github.com/confluentinc/confluent-kafka-python
- Golang - https://github.com/confluentinc/confluent-kafka-go
- Node/JavaScript - https://github.com/Blizzard/node-rdkafka (not supported by Confluent!)

New Kafka features will only be available to modern, updated clients!

Resources

Free E-Books from Confluent!

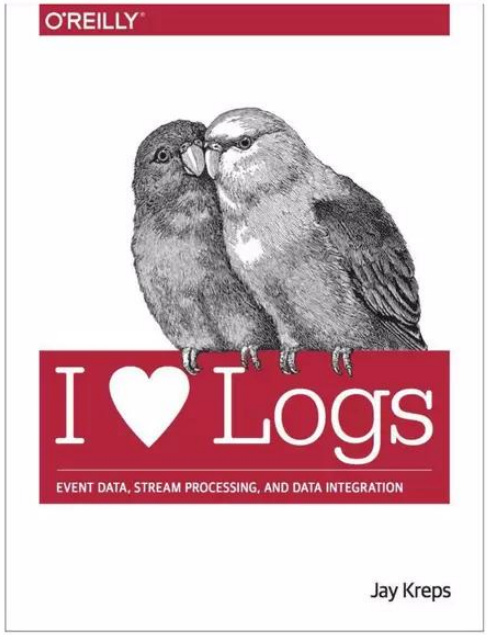
I Heart Logs:
https://www.confluent.io/ebook/i-heart-logs-event-data-stream-processing-and-data-integration/

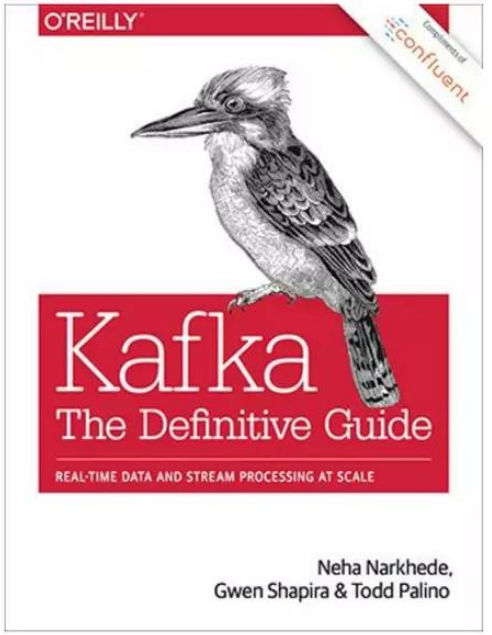
Kafka: The Definitive Guide: https://www.confluent.io/resources/kafka-the-definitive-guide/

Designing Event Driven Systems:
https://www.confluent.io/designing-event-driven-systems/

Confluent Blog: https://www.confluent.io/blog

Thank You!





CONFLUENT

Thank you!

pascal@confluent.io