Distributed Logs

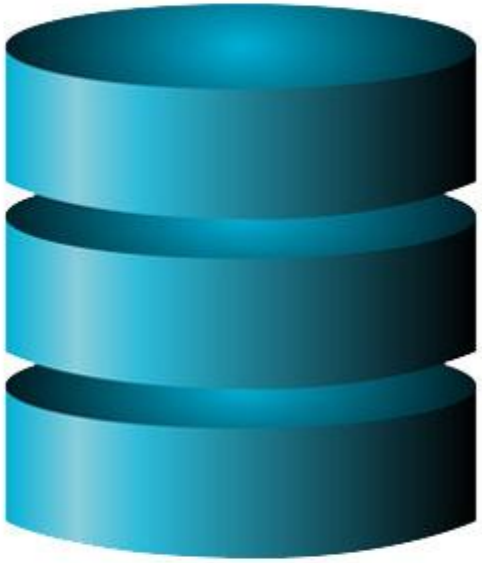
Michael Enudi

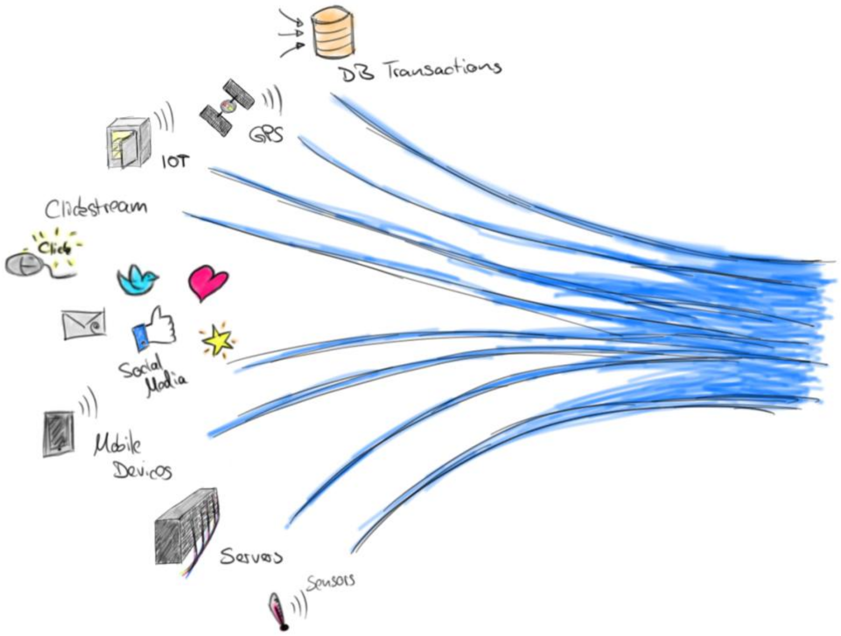
Journey through the world of databases and data engineering

Scope

- Data Architectures
- Introduction to Distributed Logs
- Use case: Apache Kafka
- Apache Zookeeper
- Stream processing
- Stream Processing II (Kafka Streams)
- Stream analytics
- Lab: NYC Taxi and Fares
- Streaming: To be or not to be

In software or data engineering, we encounter data in two general states





Data-at-rest

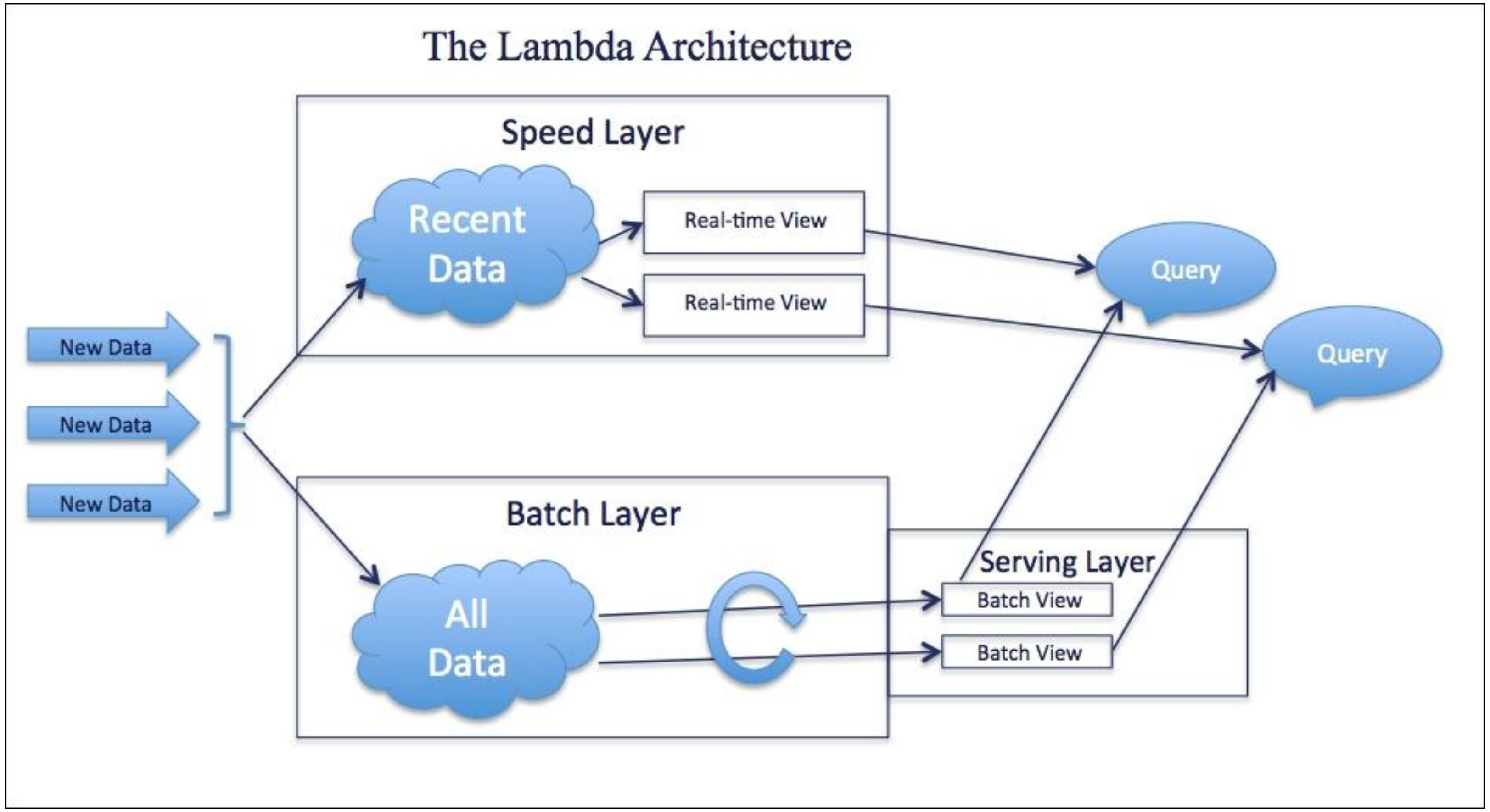
Batch

Data-in-motion

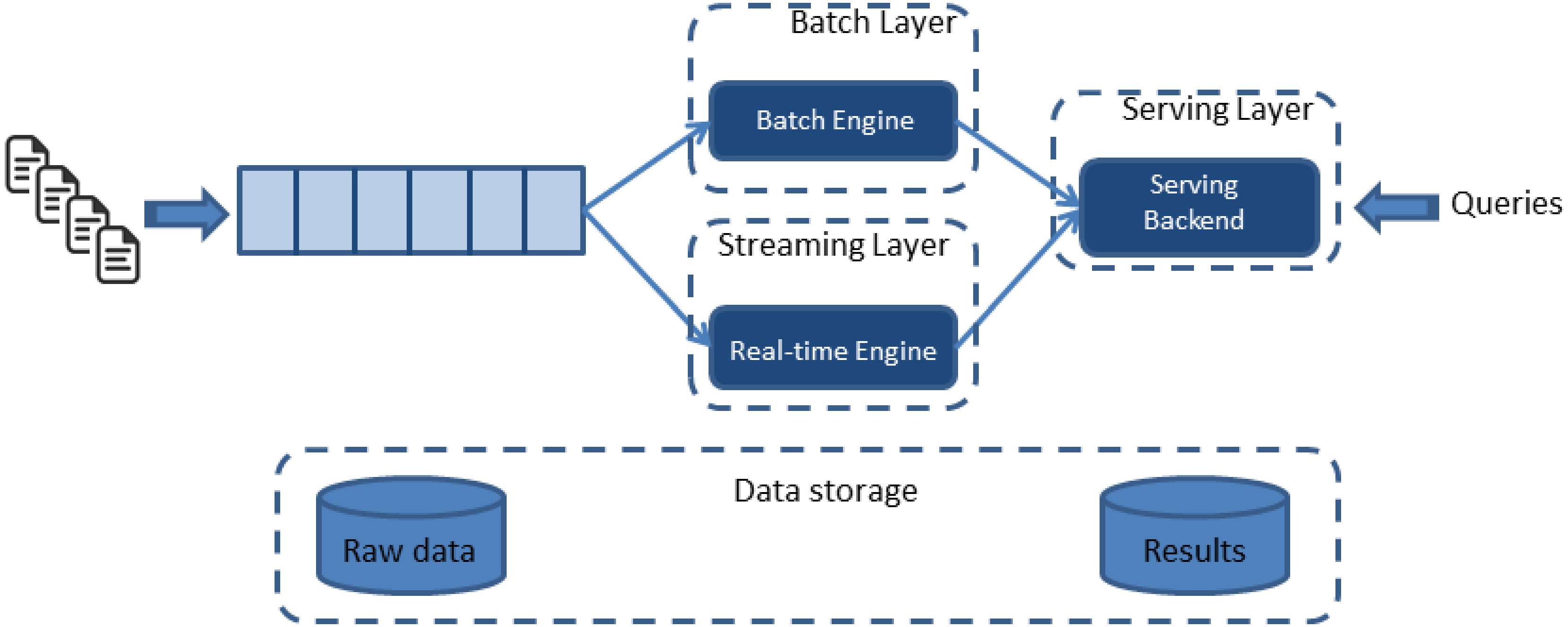
Stream/Streaming

"Data-in-Use" / "Serving"

Data architecture (Lambda)



Data architecture (Kappa)



Message Brokers

Apache ACTIVEMQ

ØMQ

HornetQ

amazon SQS

RabbitMQ

Distributed Logs

LogDevice

APACHE kafka™

DISTRIBUTEDLOG

amazon KINESIS

Azure Event Hub

APACHE kafka™

DISTRIBUTED COMMIT LOG AS A STORAGE PLATFORM

APACHE kafka™

confluent

❑ Used as a Message Broker or as a building real-time data pipelines for streaming applications

❑ Kafka started from within LinkedIn in January 2011

❑ Current version as at June 2019 is 2.3.0

❑ Written in Scala and Java

❑ Uses Zookeeper (just like HBase) for distributed coordination and metadata repository.

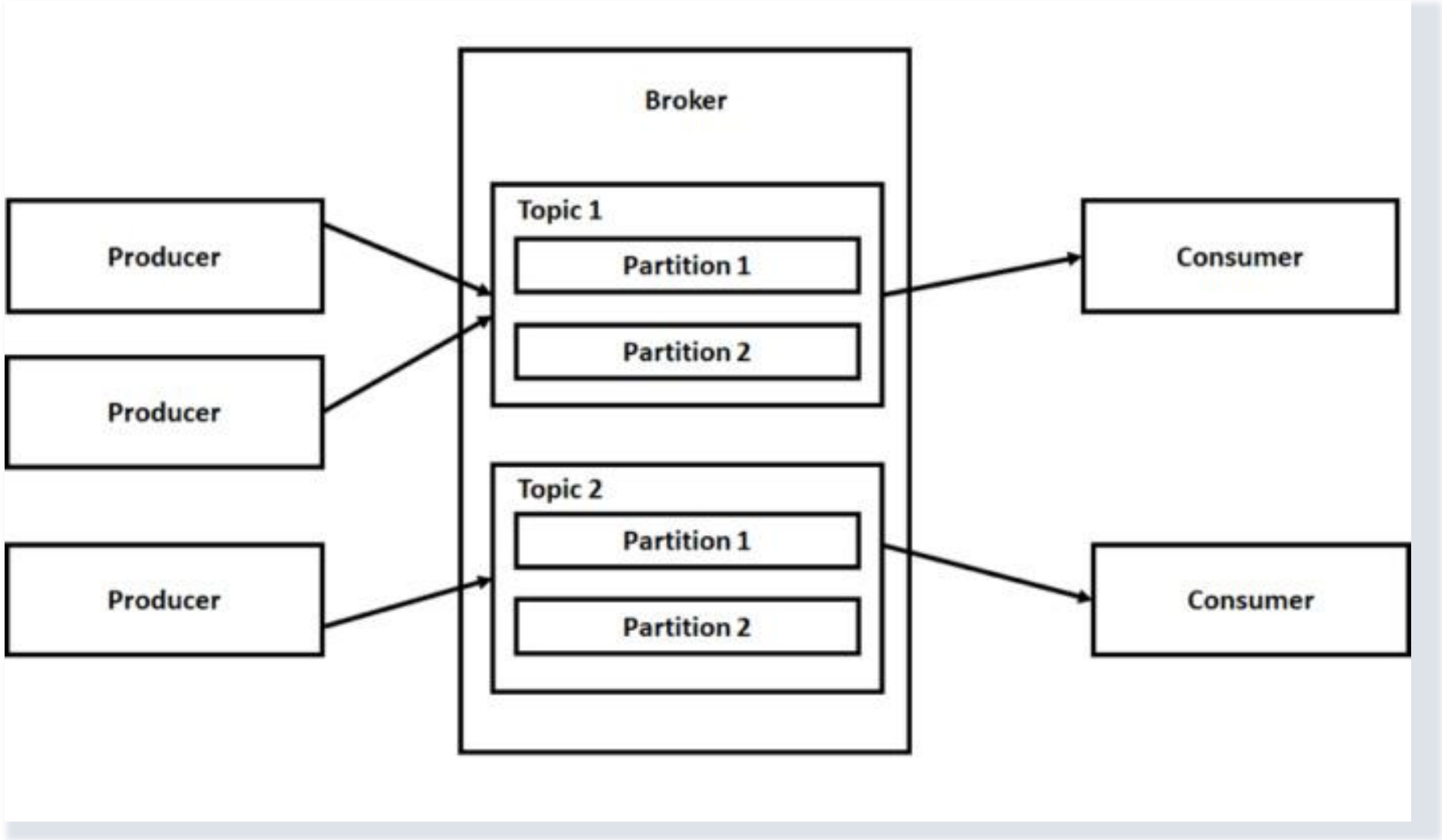
❑ Confluent currently offers Kafka as part of the Confluent Platform.

❑ Use cases include message broker, streaming application, event sourcing, commit logging, activity tracking and more.

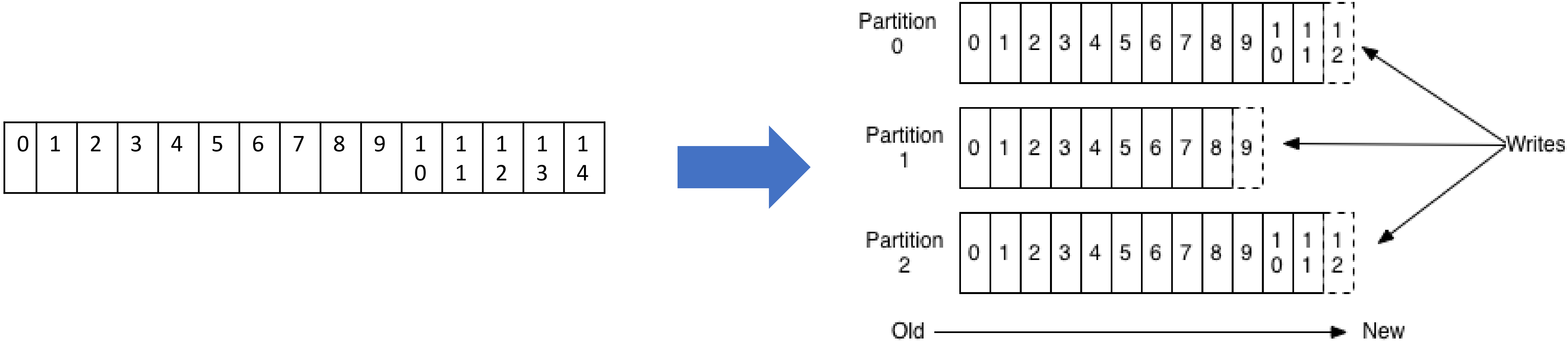
❑ High profile users of kafka include LinkedIn, Apple, eBay, Netflix, Yelp, Oracle, Uber, Citibank, Spotify and a host of others

APACHE kafka™

- Topic
- Records / Messages
- Producers
- Consumers



Kafka Topic


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Partition 0
0 1 2 3 4 5 6 7 8 9 10 11 12
Partition 1
0 1 2 3 4 5 6 7 8 9
Partition 2
0 1 2 3 4 5 6 7 8 9 10 11 12
Writes
Old
New

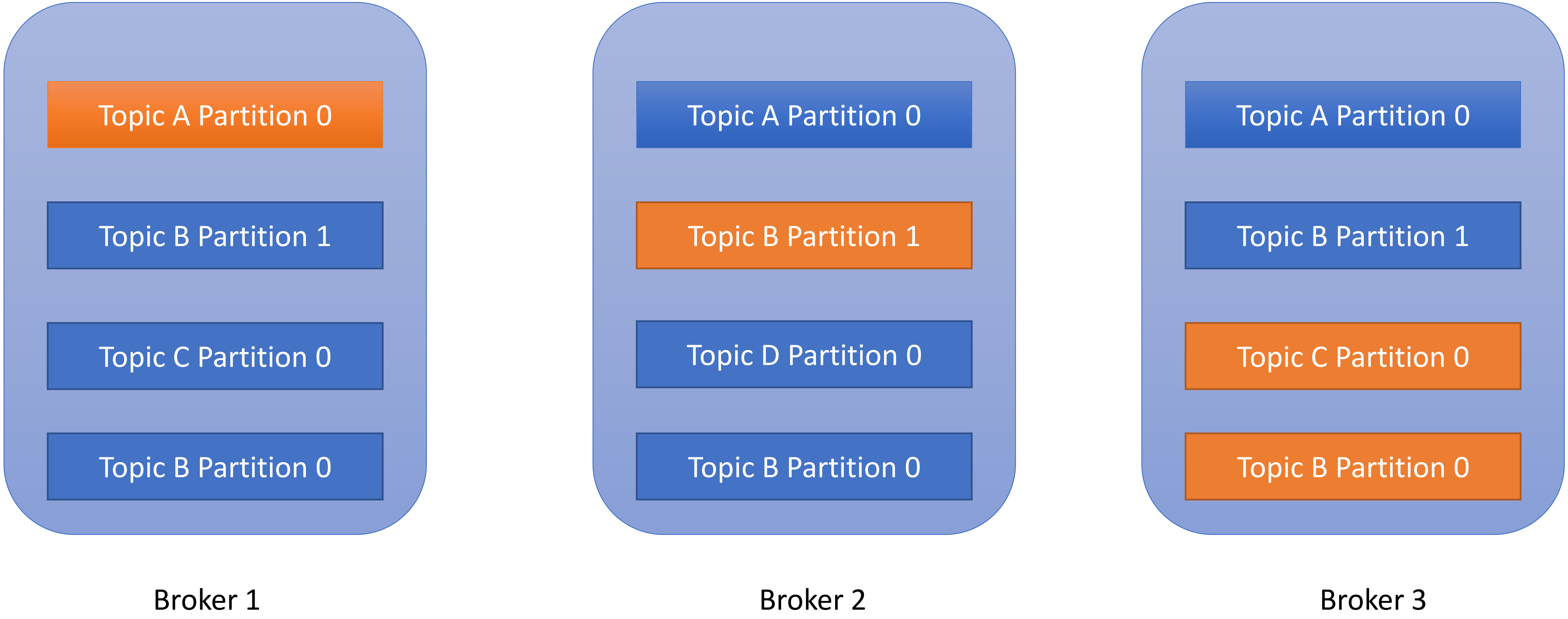
Topic Partition Replication

Topic A - 1 Partition & Replication factor of 3

Topic B - 2 Partition & Replication factor of 3

Topic C - 1 Partition & Replication factor of 2

Topic D - 1 Partition & Replication factor of 1



Topic A Partition 0

Topic B Partition 1

Topic C Partition 0

Topic B Partition 0

Broker 1

Topic A Partition 0

Topic B Partition 1

Topic D Partition 0

Topic B Partition 0

Broker 2

Topic A Partition 0

Topic B Partition 1

Topic C Partition 0

Topic B Partition 0

Broker 3

Kafka Messages

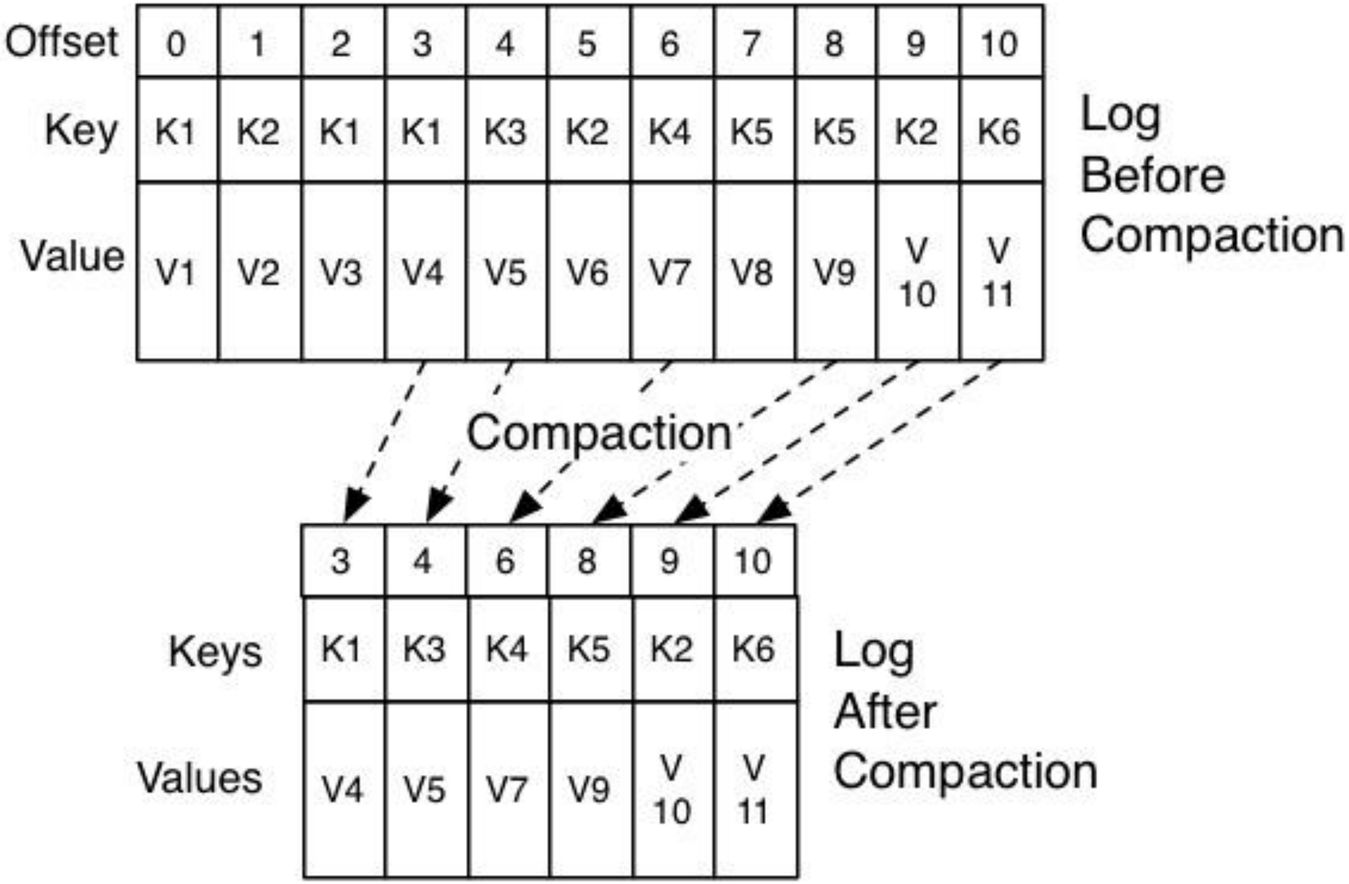
Topic

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Relational Table

0
1
2
3
4
5
6
4
7
8
9
10
11
12
13
14

Kafka Messages
Compaction Clean up policy



Offset	0	1	2	3	4	5	6	7	8	9	10
Key	K1	K2	K1	K1	K3	K2	K4	K5	K5	K2	K6
Value	V1	V2	V3	V4	V5	V6	V7	V8	V9	V 10	V 11

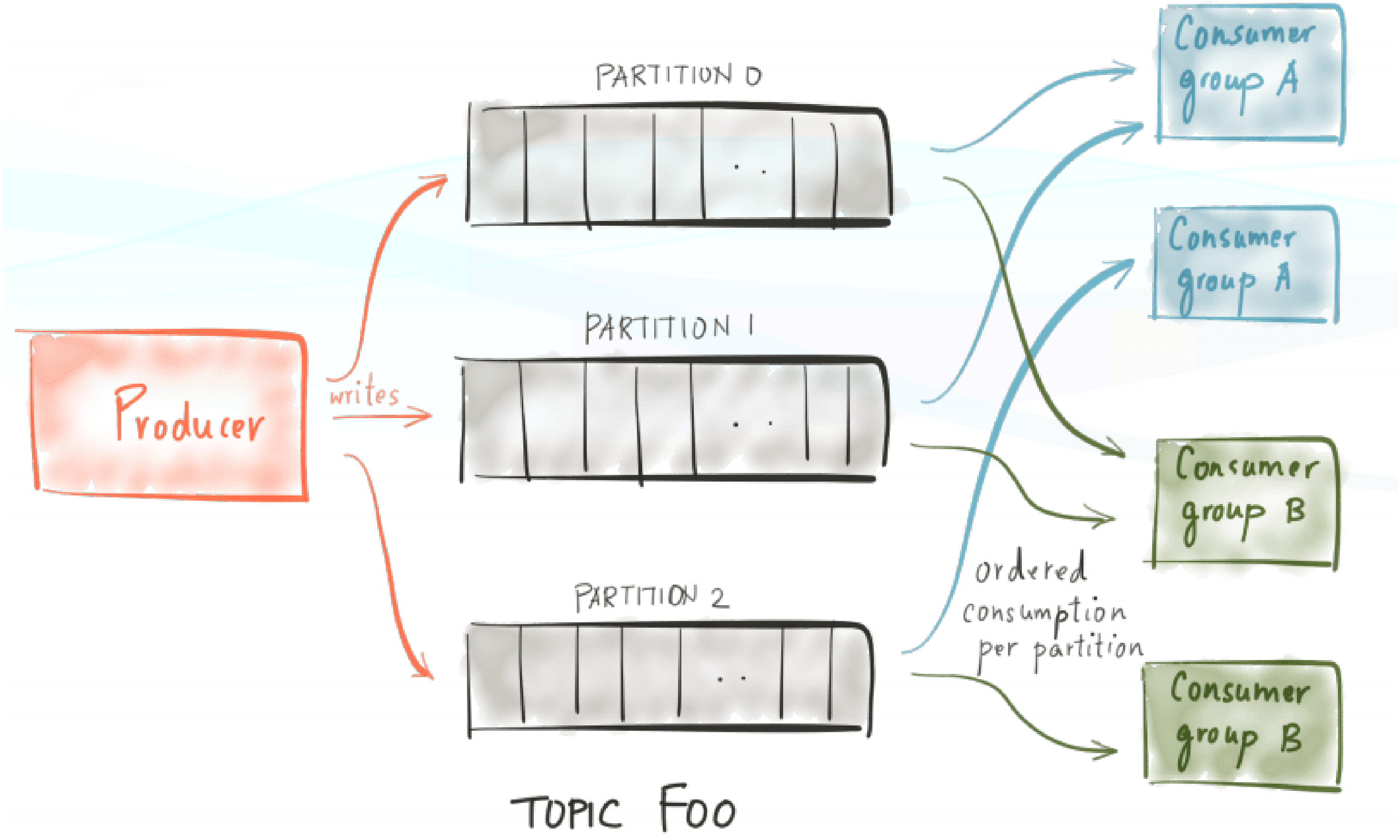
Log Before Compaction

Compaction

	3	4	6	8	9	10
Keys	K1	K3	K4	K5	K2	K6
Values	V4	V5	V7	V9	V 10	V 11

Log After Compaction

Kafka Producer



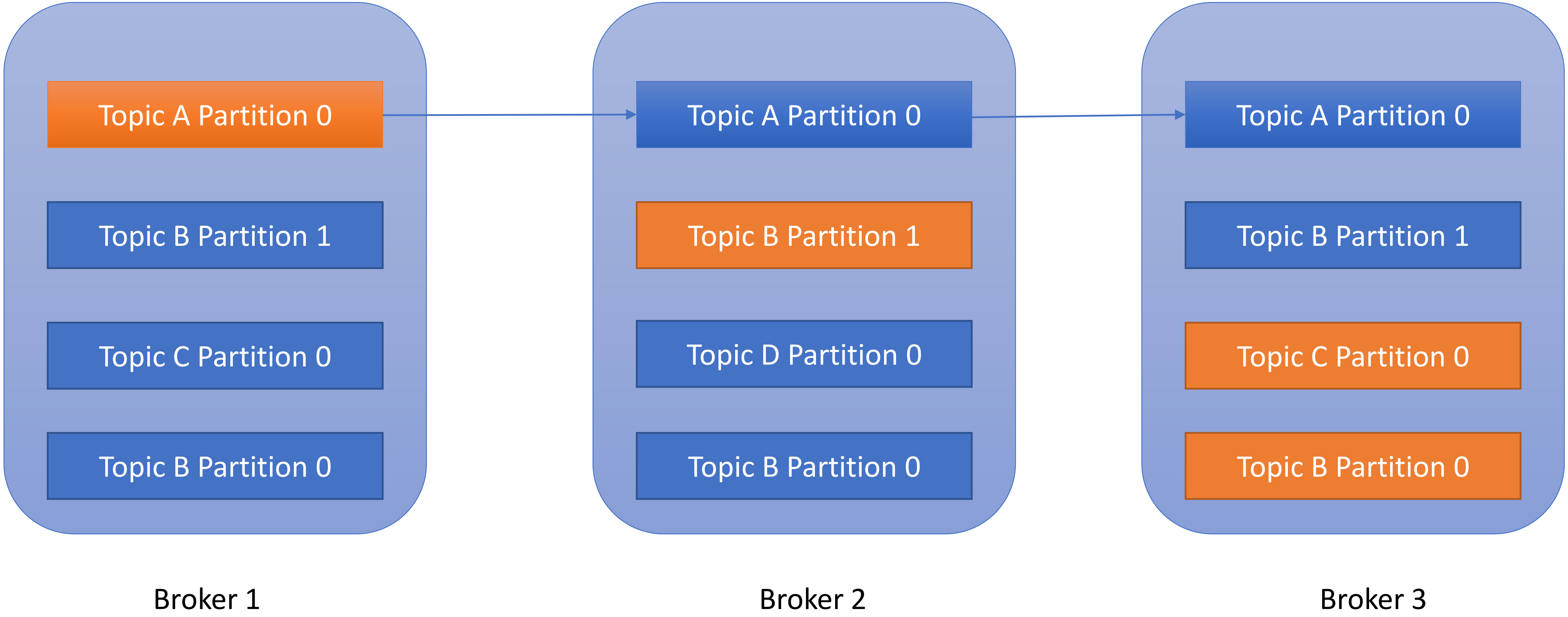
Kafka Producer Acknowledgement

Topic A - 1 Partition & Replication factor of 3

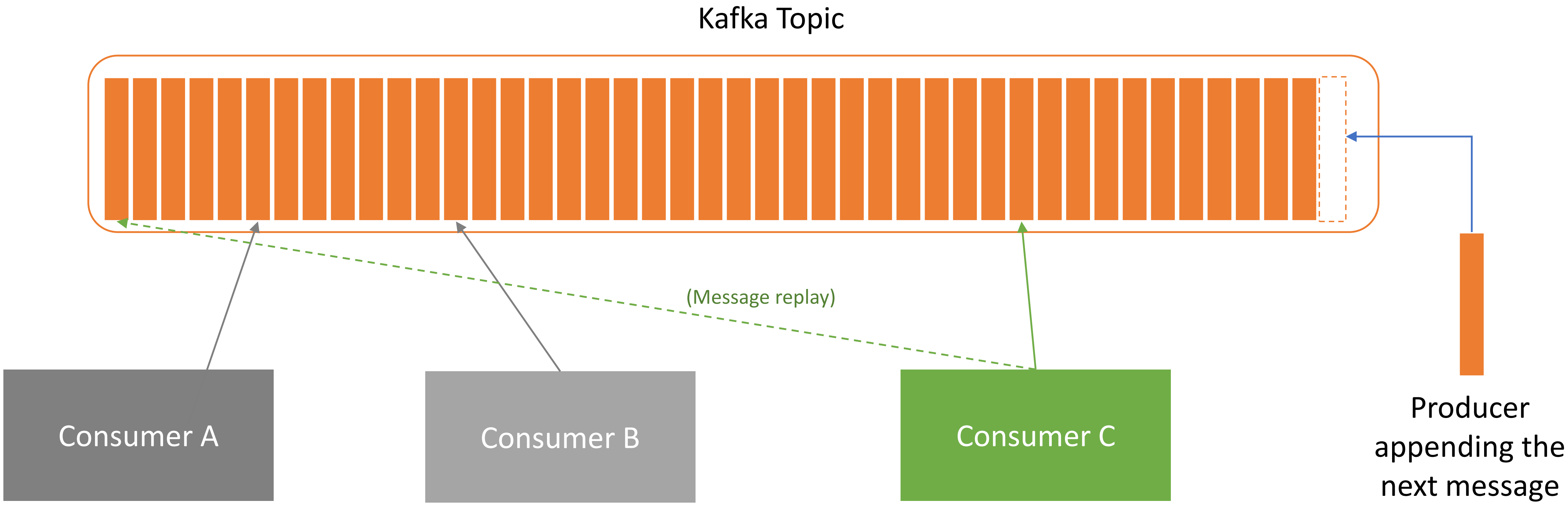
Topic B - 2 Partition & Replication factor of 3

Topic C - 1 Partition & Replication factor of 2

Topic D - 1 Partition & Replication factor of 1

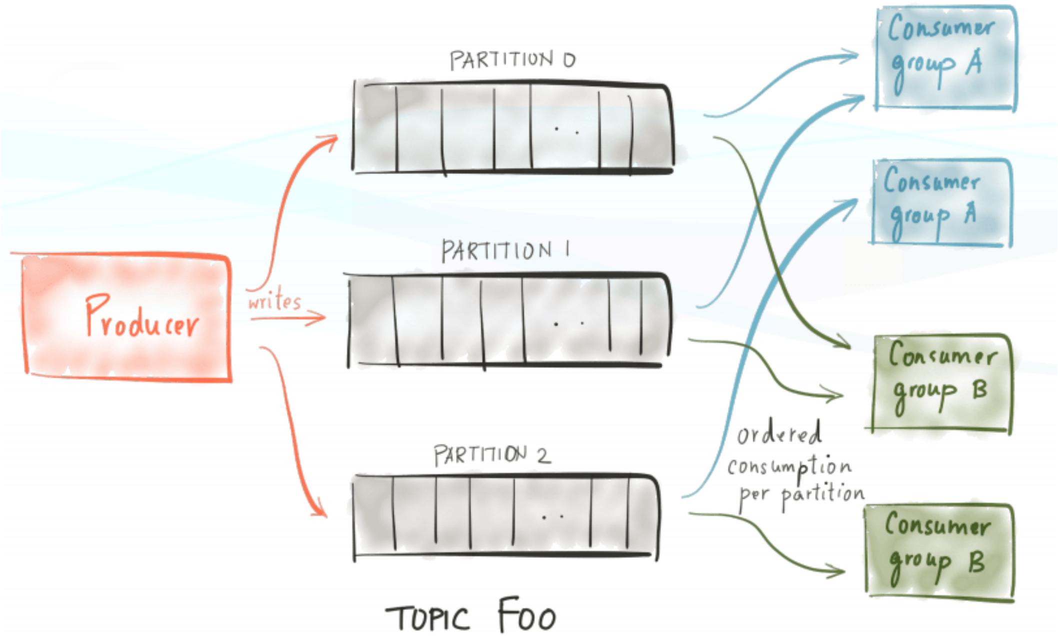


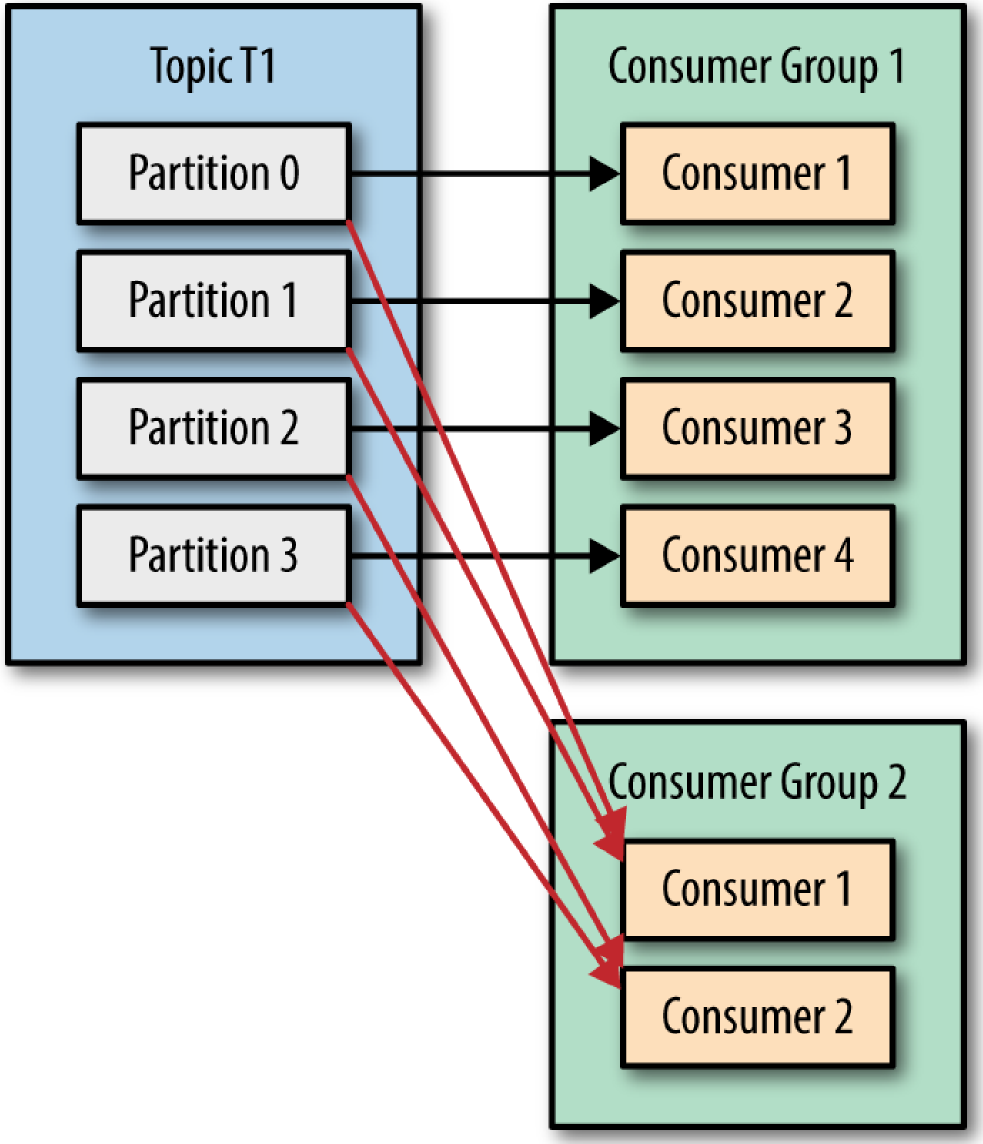
Kafka Consumers



Consumers subscribe to Topics and read from them at their various offsets

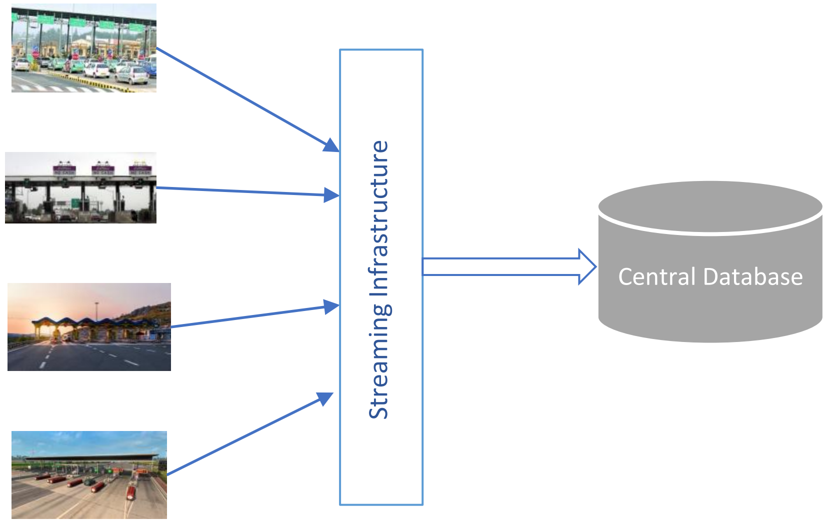
Kafka Consumer Groups



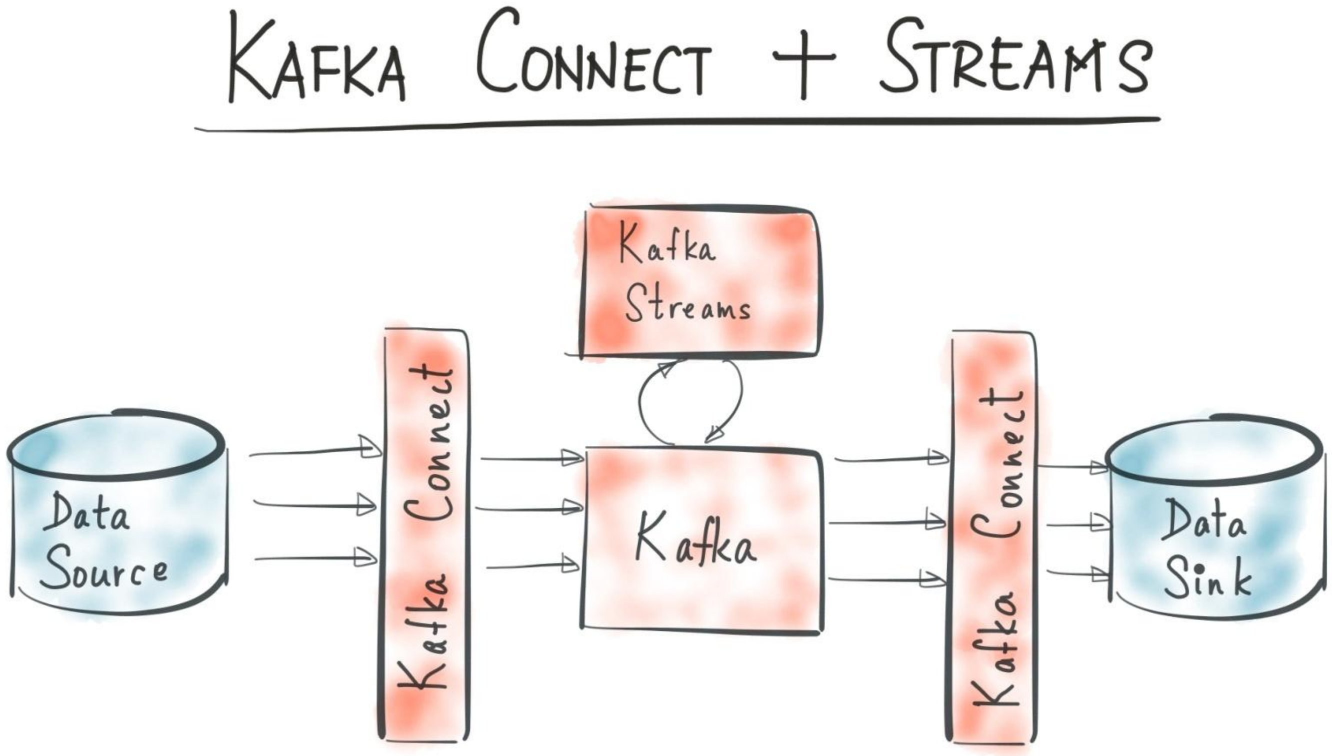


Use case

Toll payment collection and monitoring platform



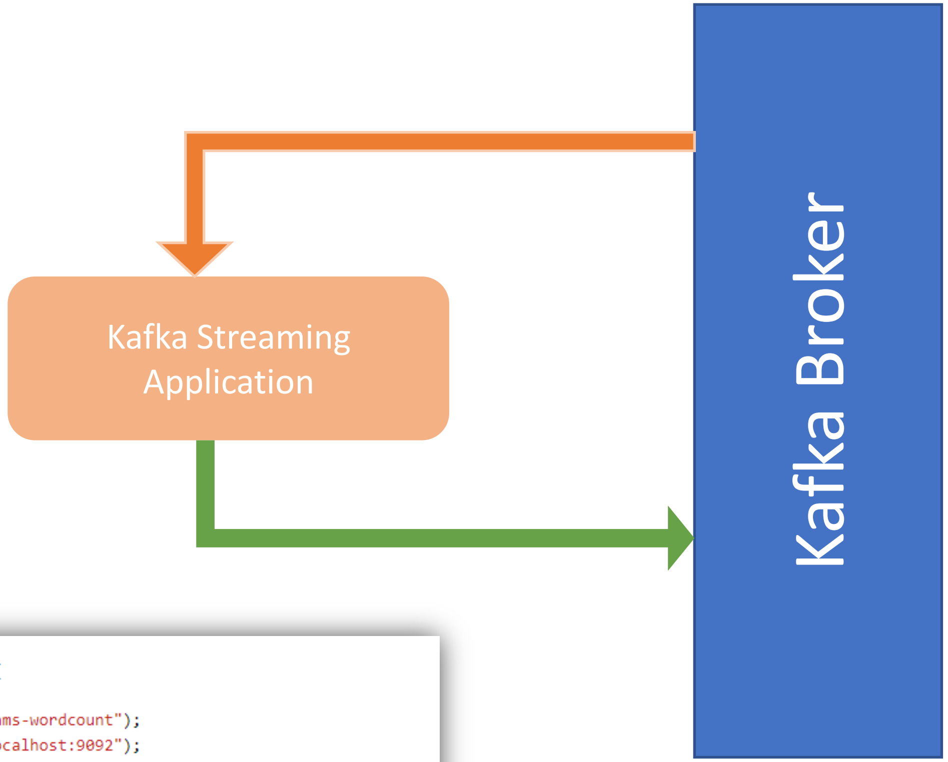
KAFKA CONNECT + KAFKA STREAMS



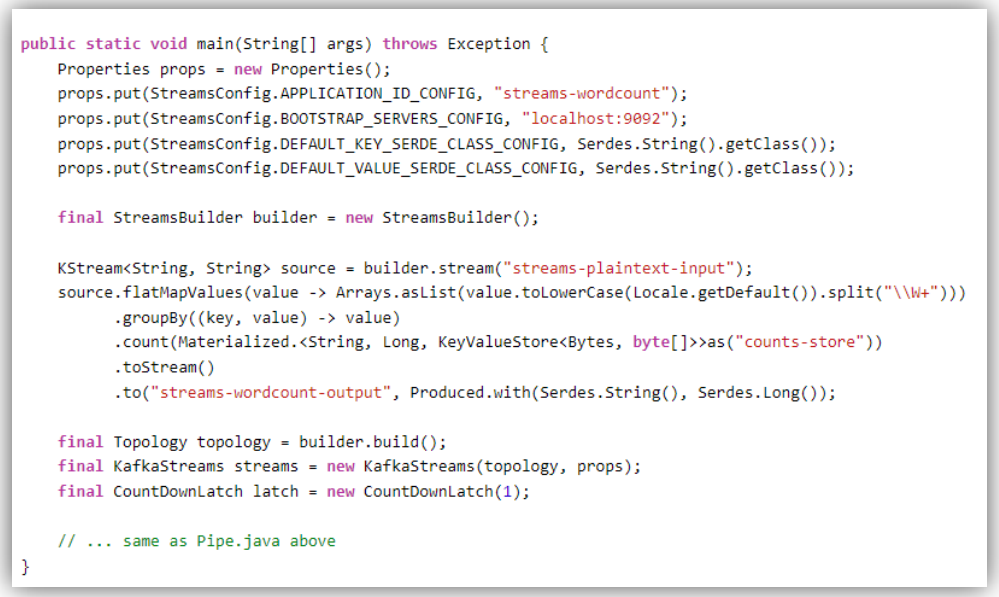
Kafka Stream

A library for implementing scalable stream processing applications that react to events in real-time in Scala and Java.

It is more than just executing a function between a consumer and a producer.



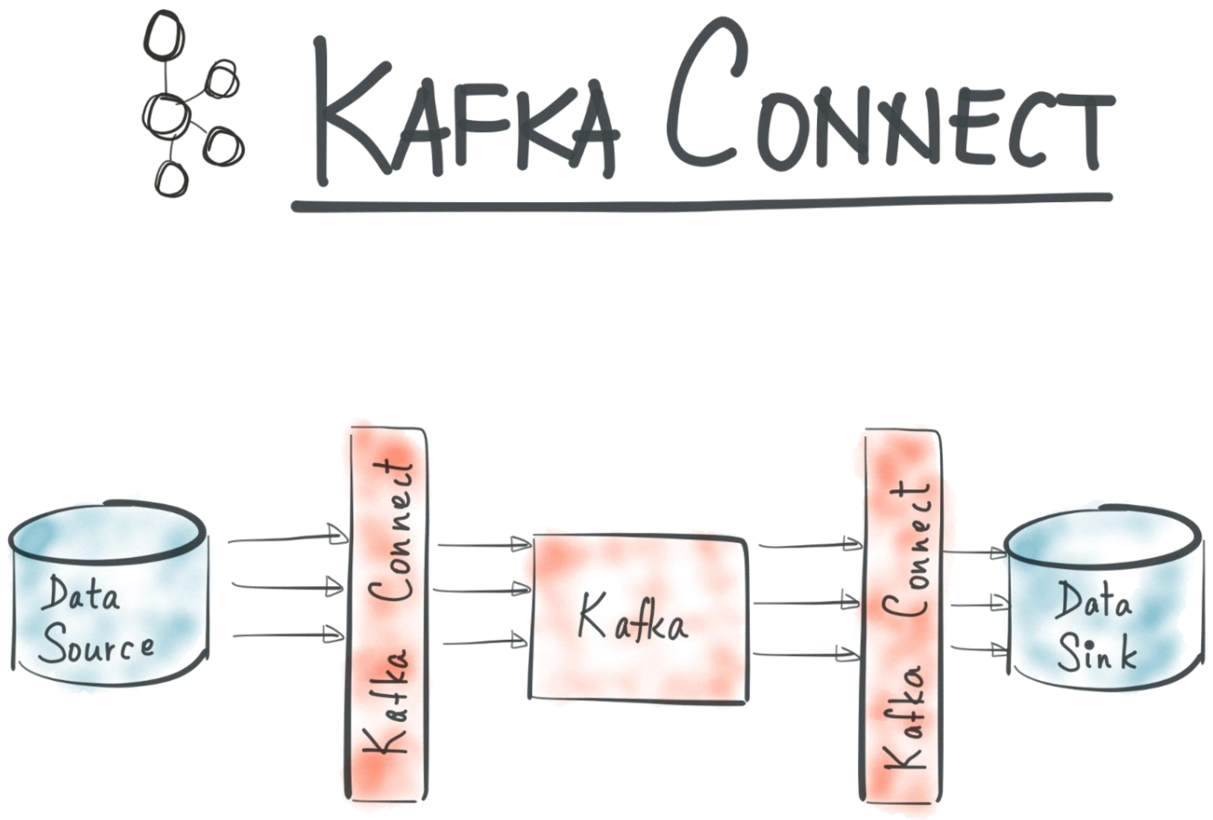
- ❑ Stream abstraction
- ❑ Streams-Tables duality
- ❑ Time
  - ❑ Windowing
- ❑ Aggregations
- ❑ State



```java
public static void main(String[] args) throws Exception {
    Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

    final StreamsBuilder builder = new StreamsBuilder();

    KStream<String, String> source = builder.stream("streams-plaintext-input");
    source.flatMapValues(value -> Arrays.asList(value.toLowerCase(Locale.getDefault()).split("\\W+")))
          .groupBy((key, value) -> value)
          .count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as("counts-store"))
          .toStream()
          .to("streams-wordcount-output", Produced.with(Serdes.String(), Serdes.Long());

    final Topology topology = builder.build();
    final KafkaStreams streams = new KafkaStreams(topology, props);
    final CountDownLatch latch = new CountDownLatch(1);

    // ... same as Pipe.java above
}
```

A library for reusable component from export data from a Kafka topic to an external destination or for import data from an external source into Kafka.

The name for this reusable component is called a connector.



KSQL

Time

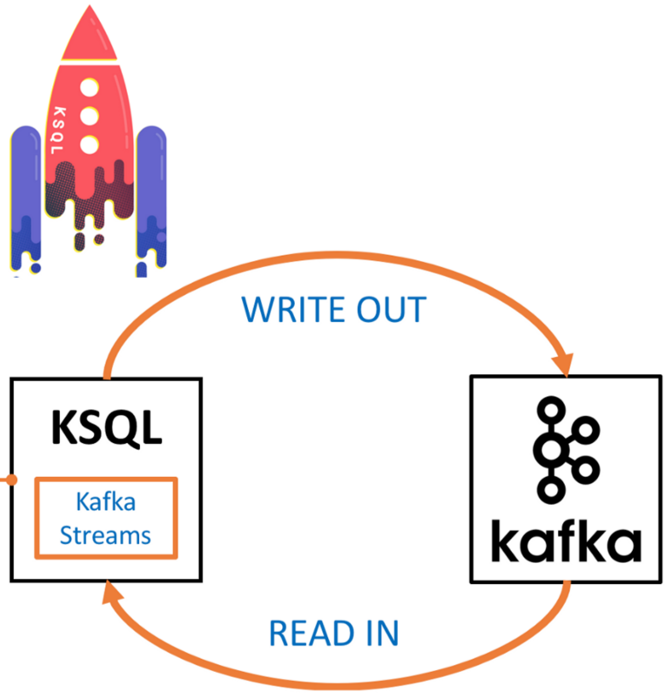
host_name	process_name
HR001	cmd.exe
HR002	cmd.exe
…	…

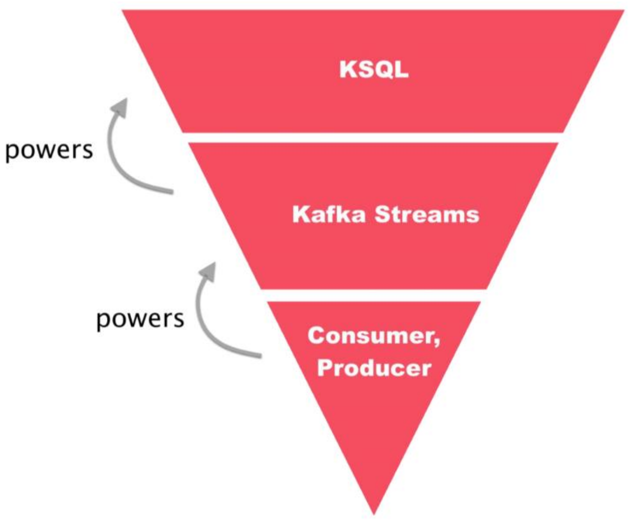
SELECT * FROM SYSMON_STREAM
WHERE process_name='cmd.exe'

SYSMON STREAM

Time

host_name	process_name
HR001	cmd.exe
HR001	powershell.exe
HR001	net.exe
…	…





```
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, …

KStream, KTable,
filter(), map(), flatMap(),
join(), aggregate(), …

subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

DISTRIBUTED LOG WRAP-UP