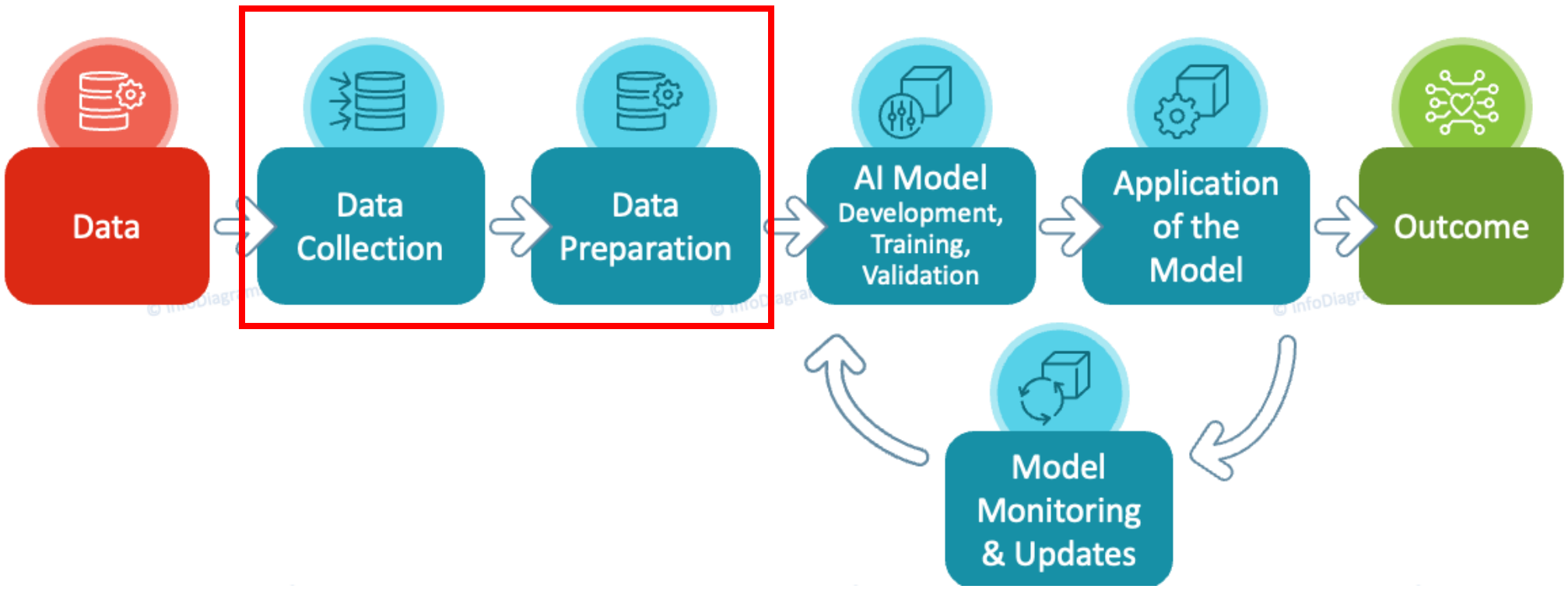
IMEN891K/IDSC501 Data Engineering

15. **Big Data IV**: Streaming

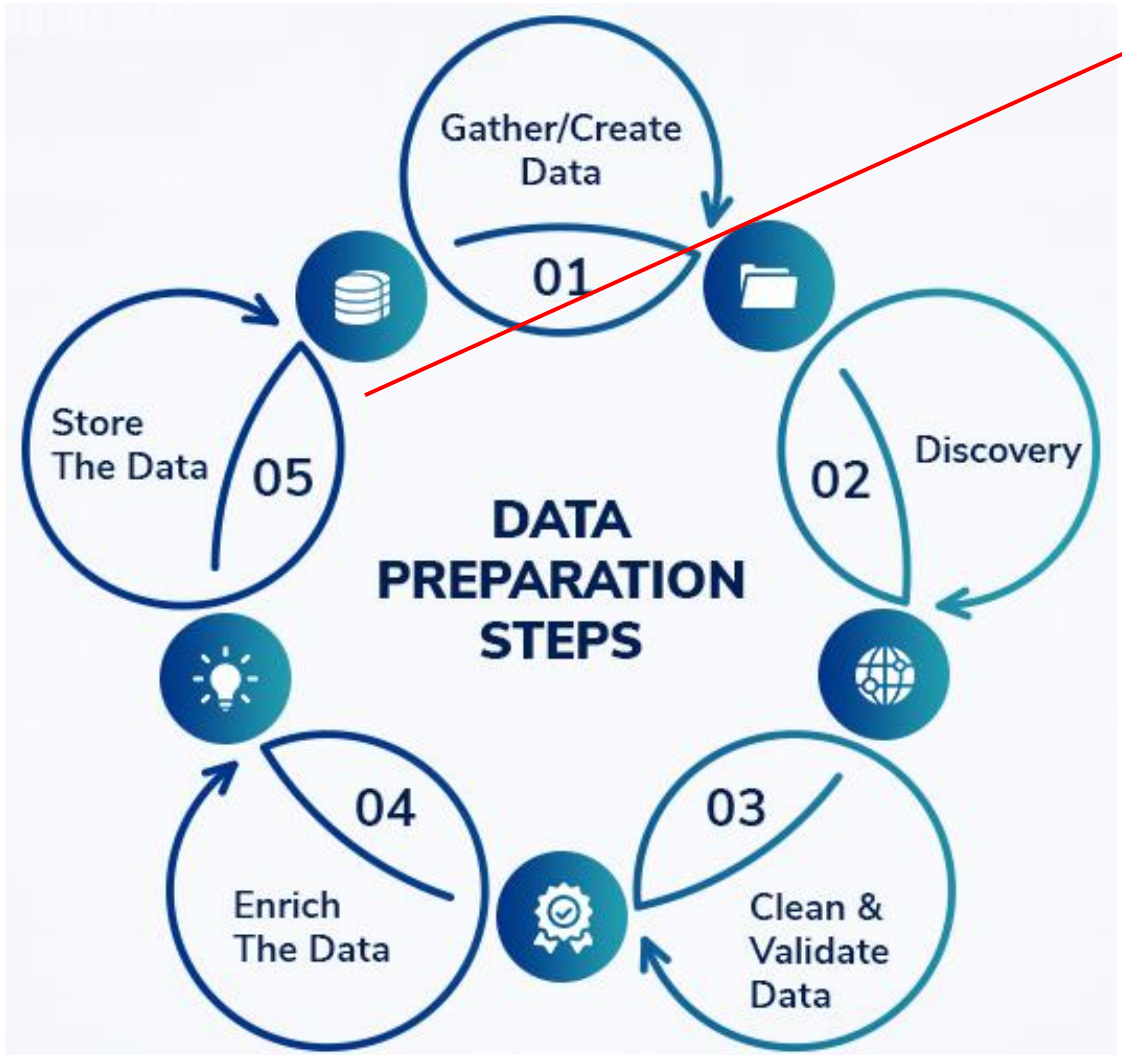
Fall 2025

Instructor: **Sangwoo Mo**

Recap: Course Overview



Recap: Course Overview



Lecture 7-15

Disclaimer

The Kafka part was heavily copy-pasted from Confluent tutorial

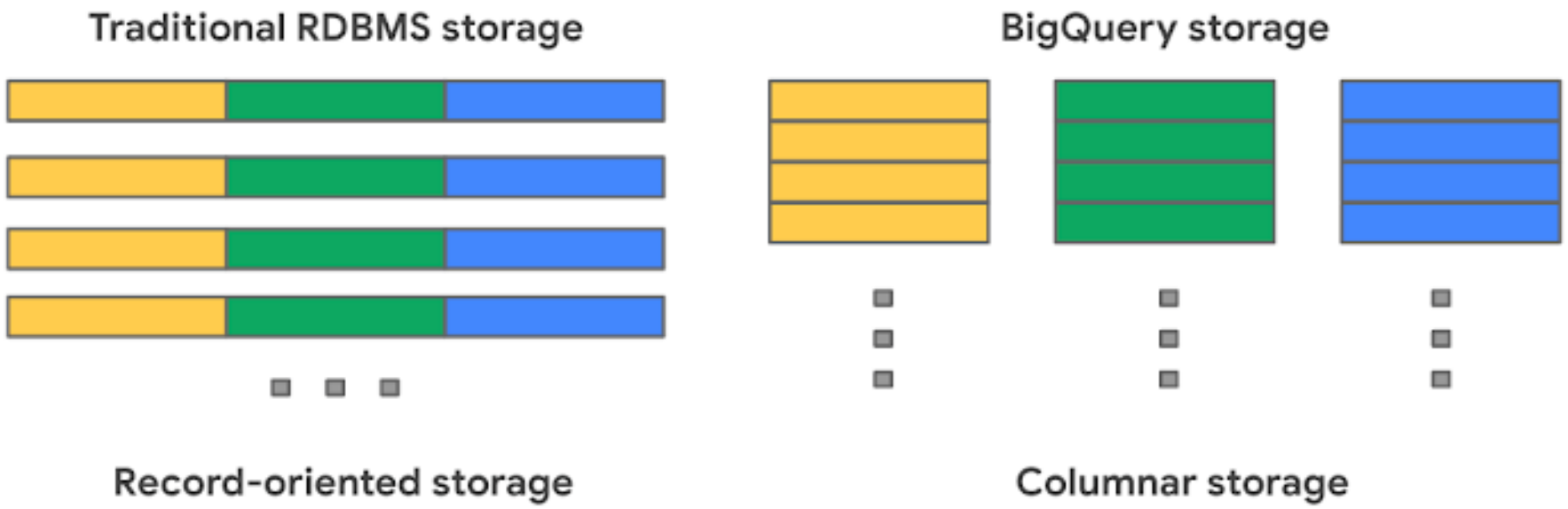


Disclaimer

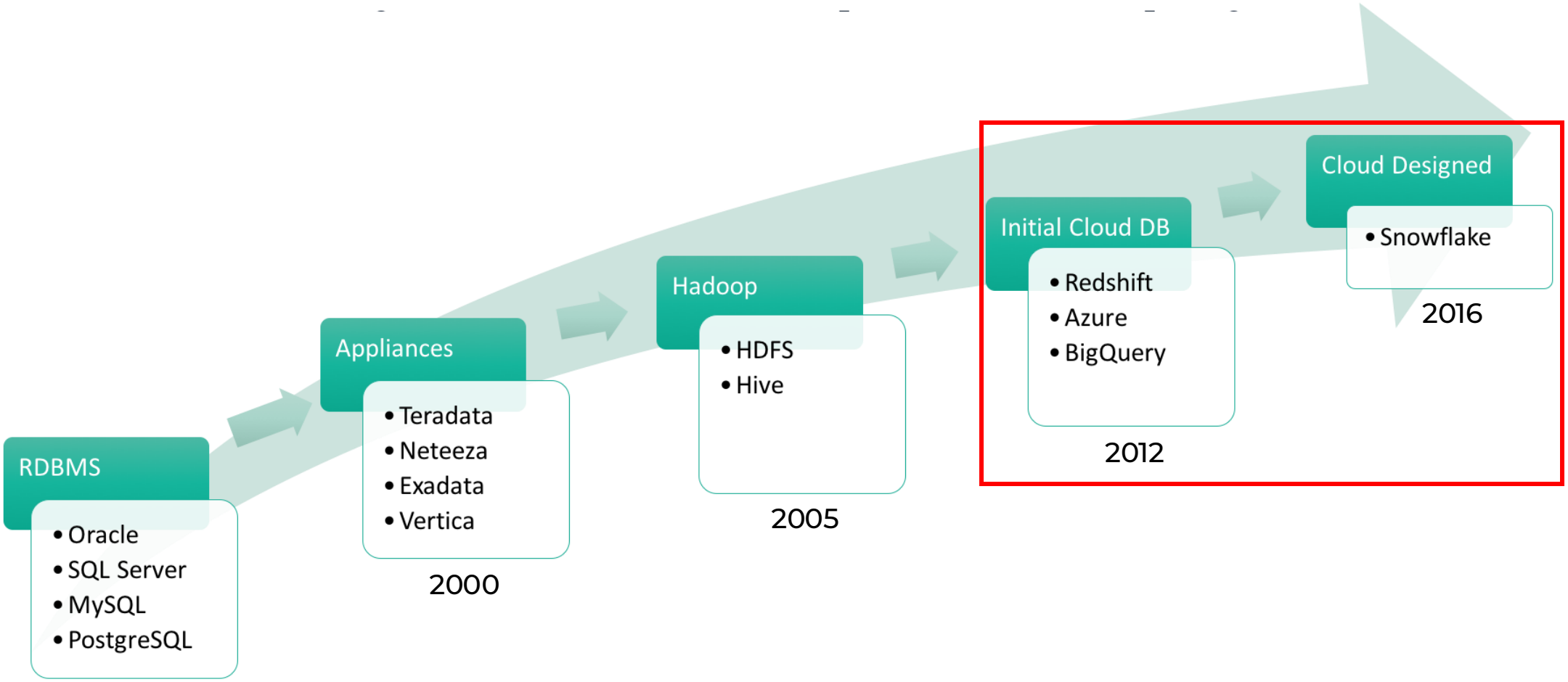
...and from Inoio tutorial



Recap: BigQuery = Serverless Data Warehouse



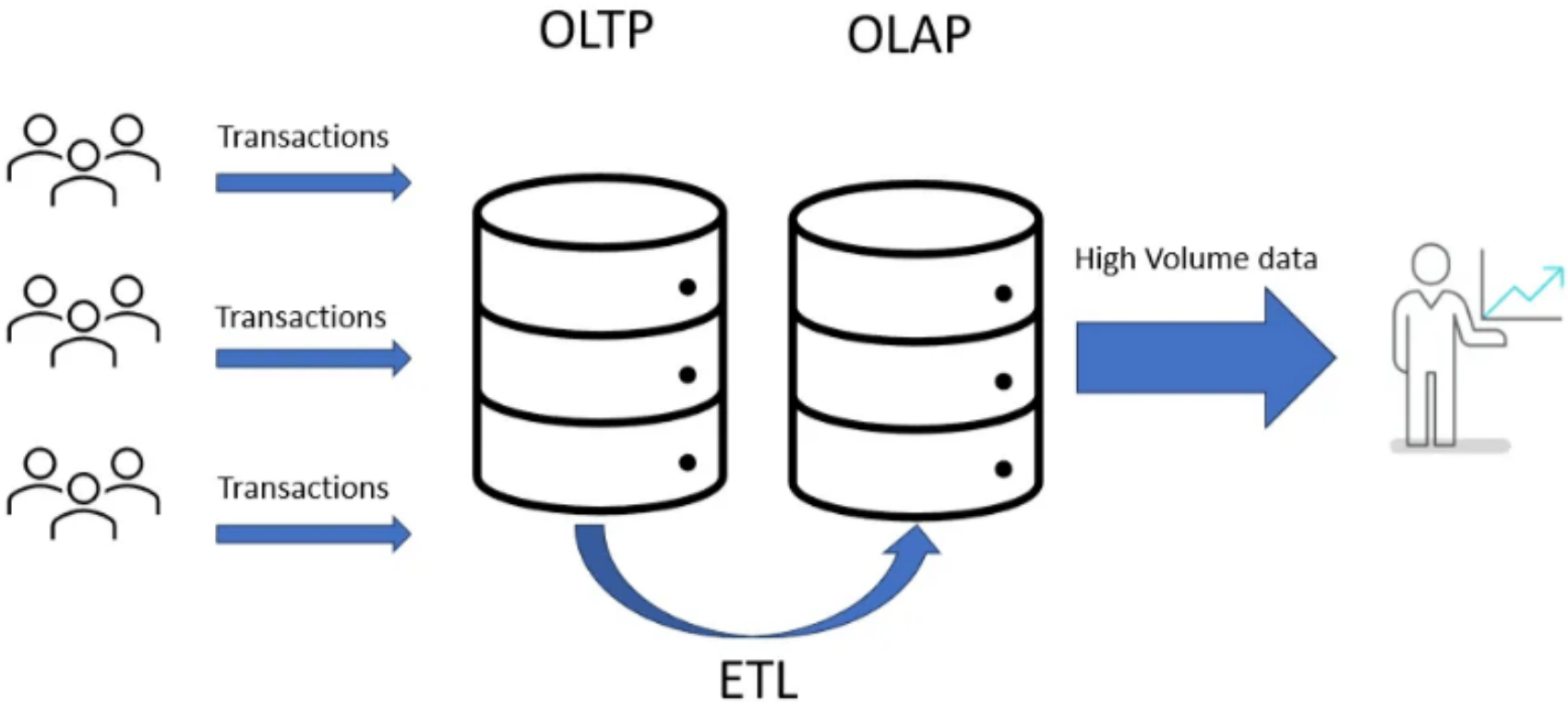
Recap: Serverless Computing



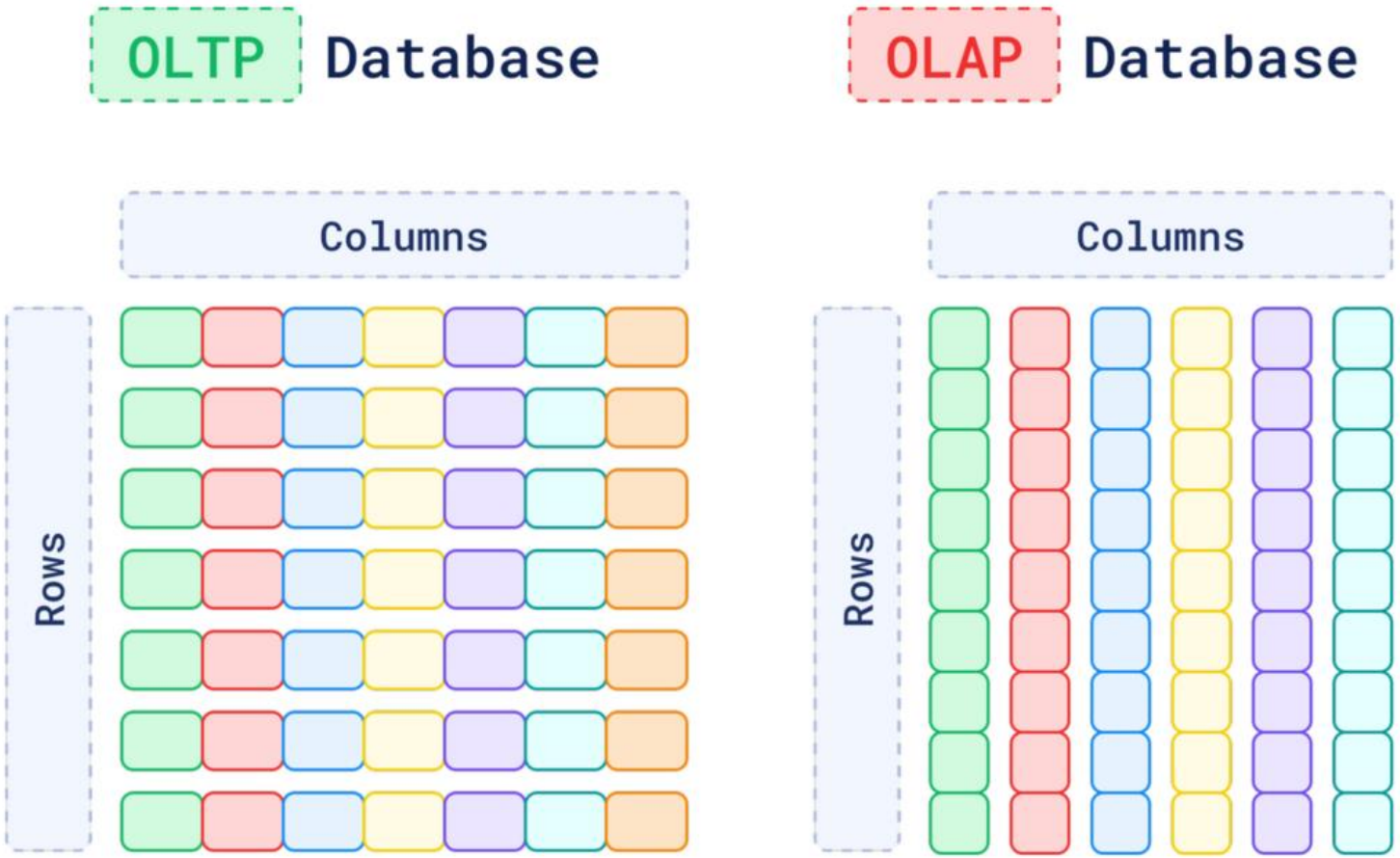
Recap: Data Warehouse

Online **Transaction** Processing (Database)

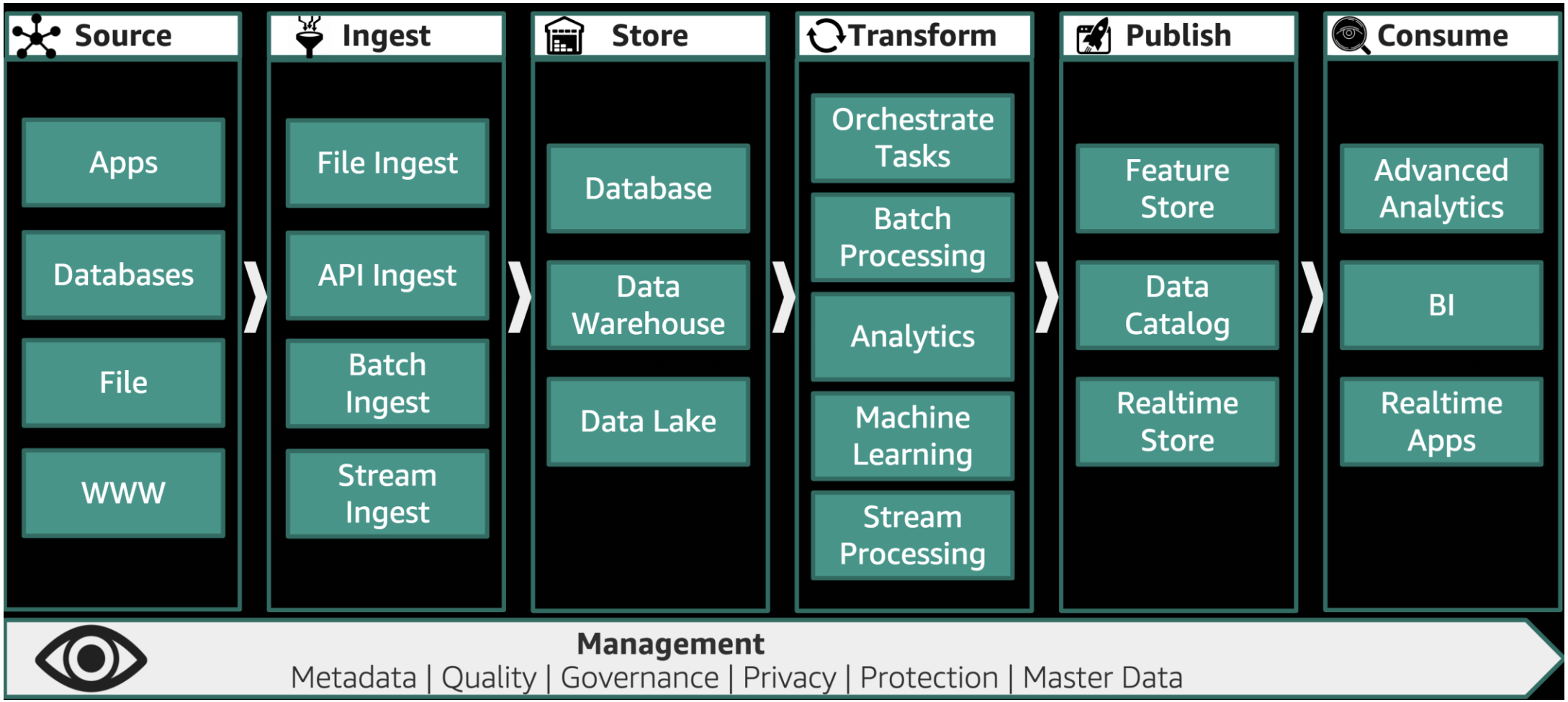
Online **Analytical** Processing (Data Warehouse)



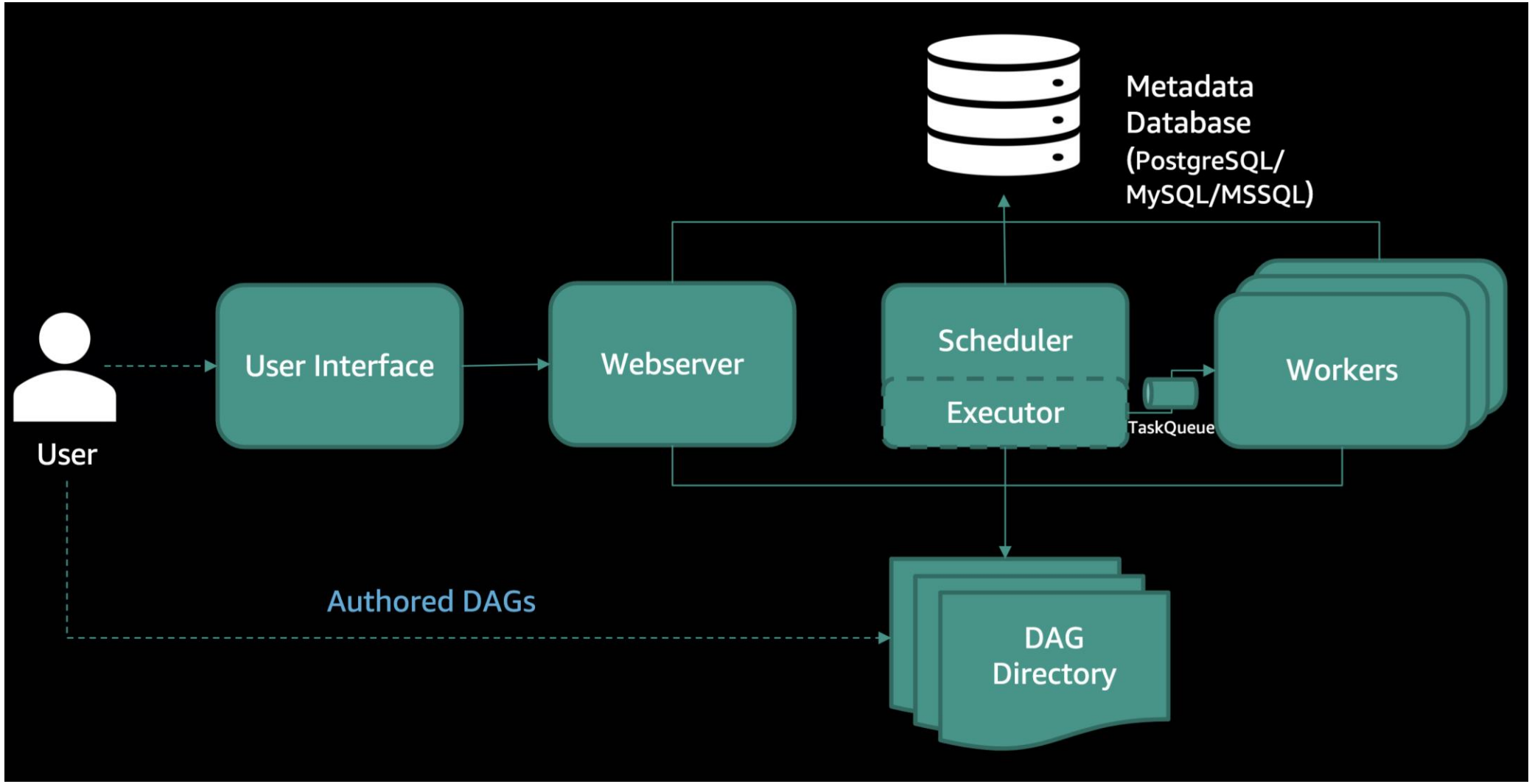
Recap: OLTP vs OLAP



Recap: Data Pipeline

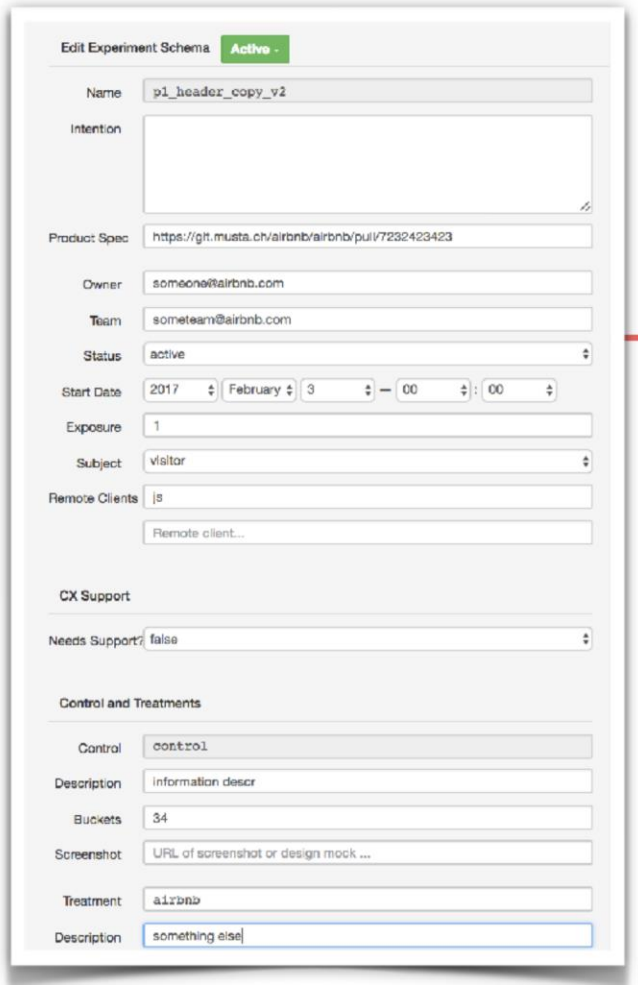


Recap: Airflow

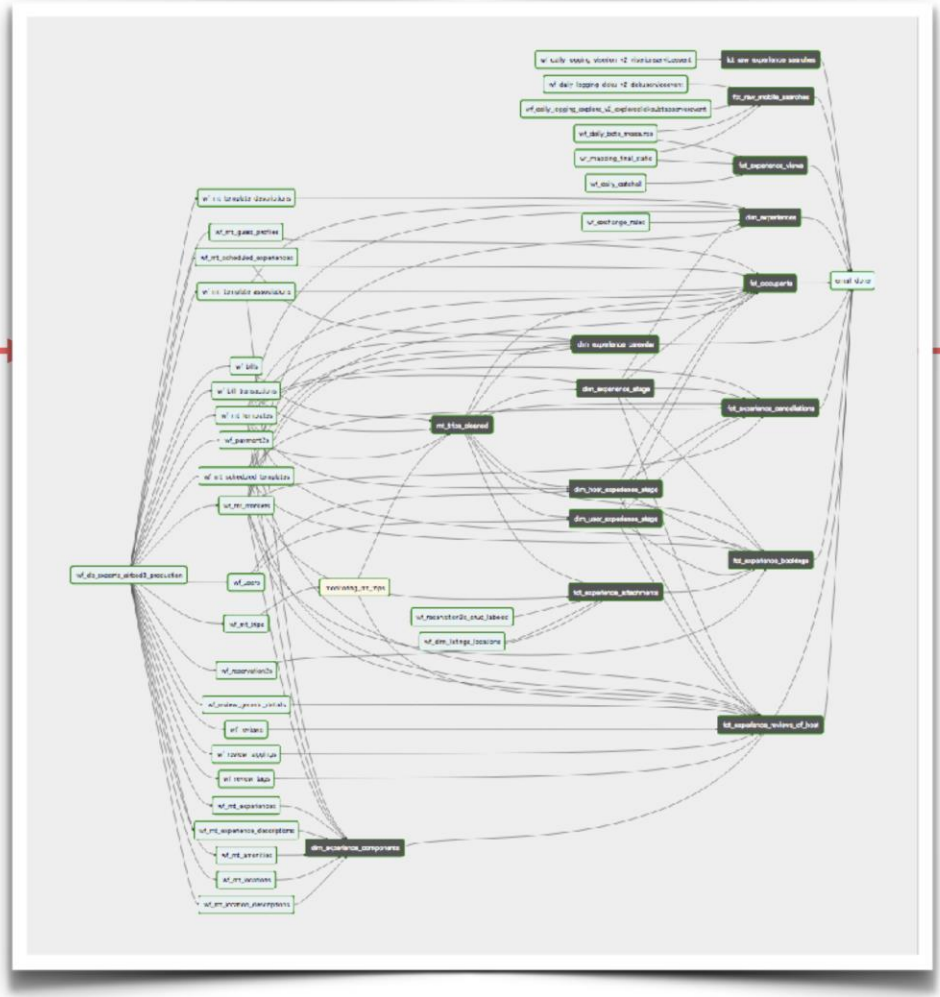


Recap: Airflow – Common Pattern

Input



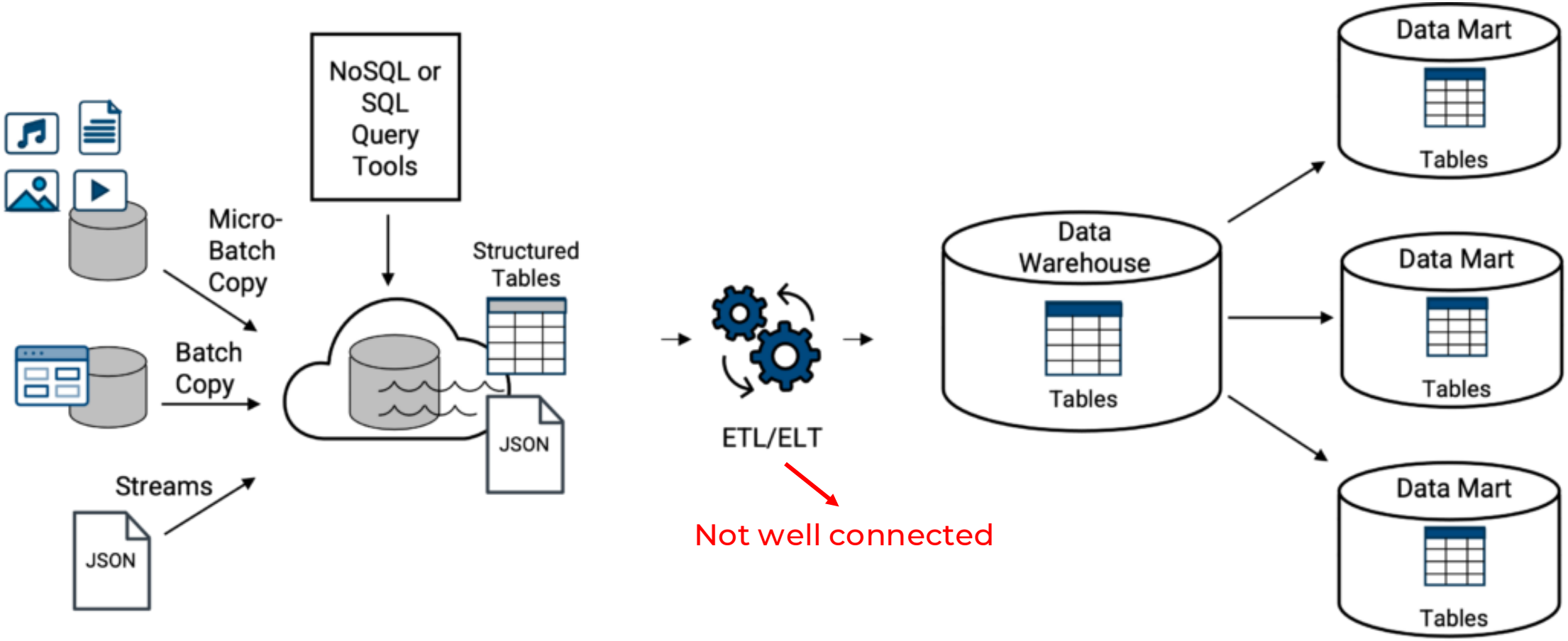
Data Processing



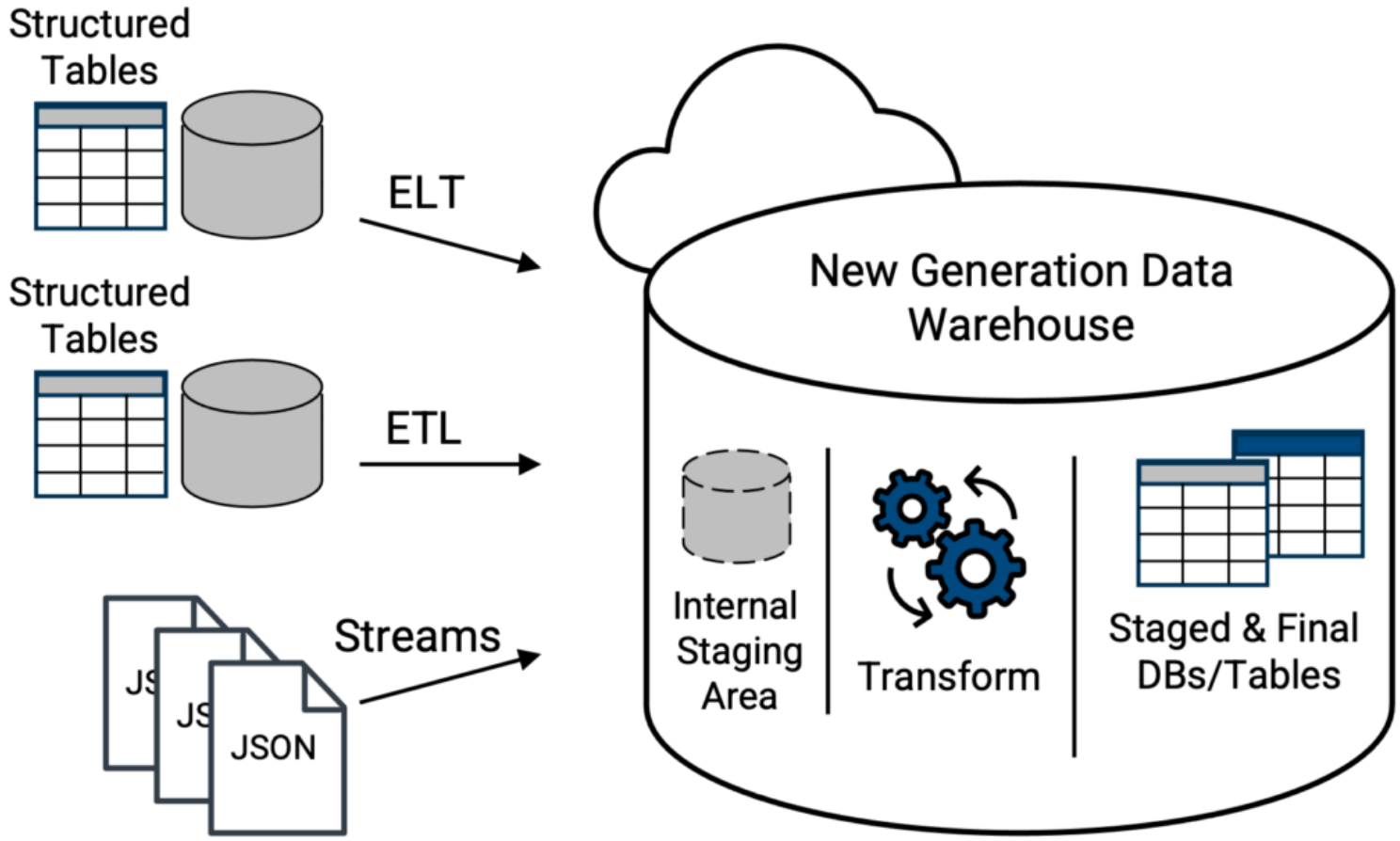
Output



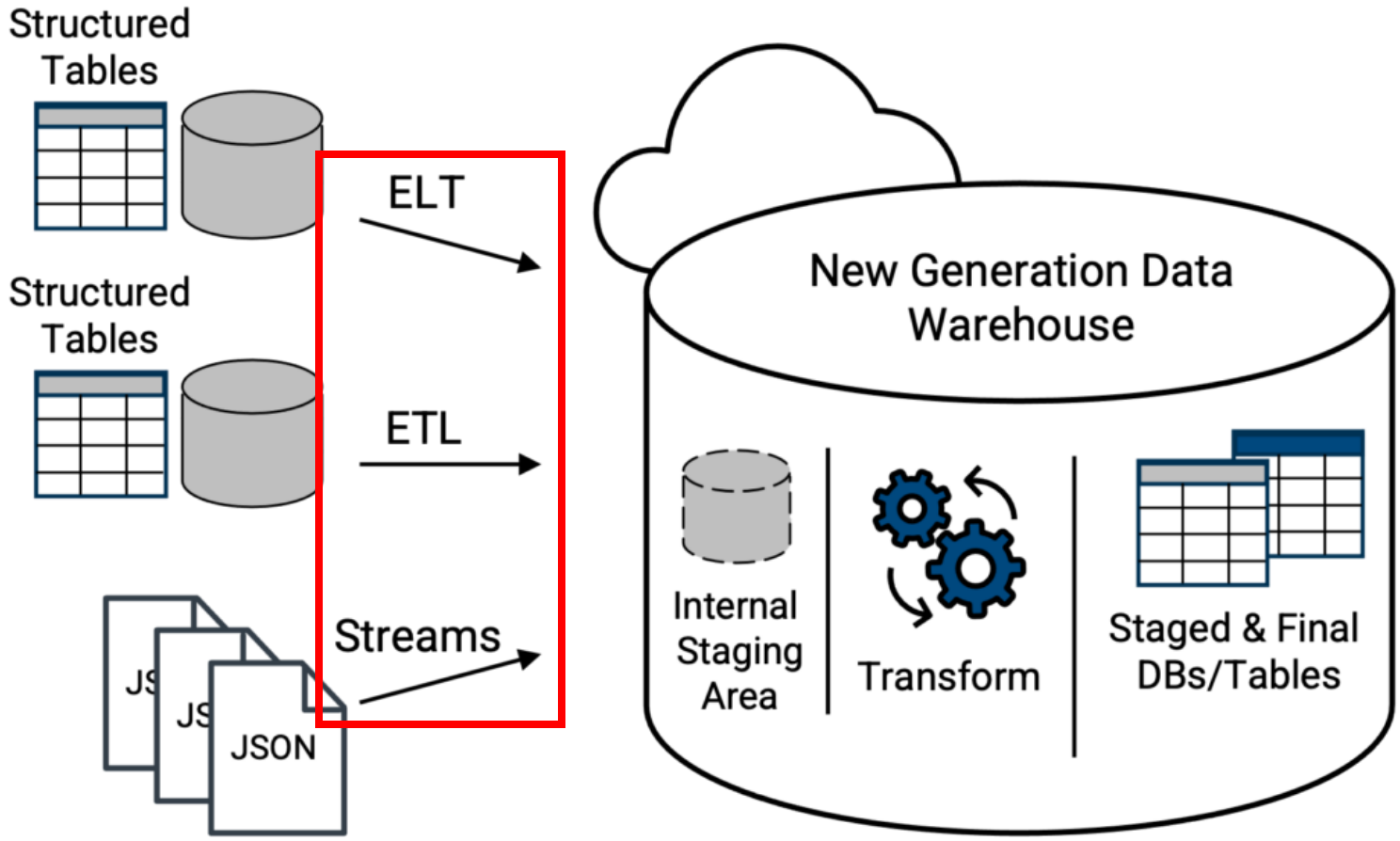
Recap: Data Lake → Warehouse



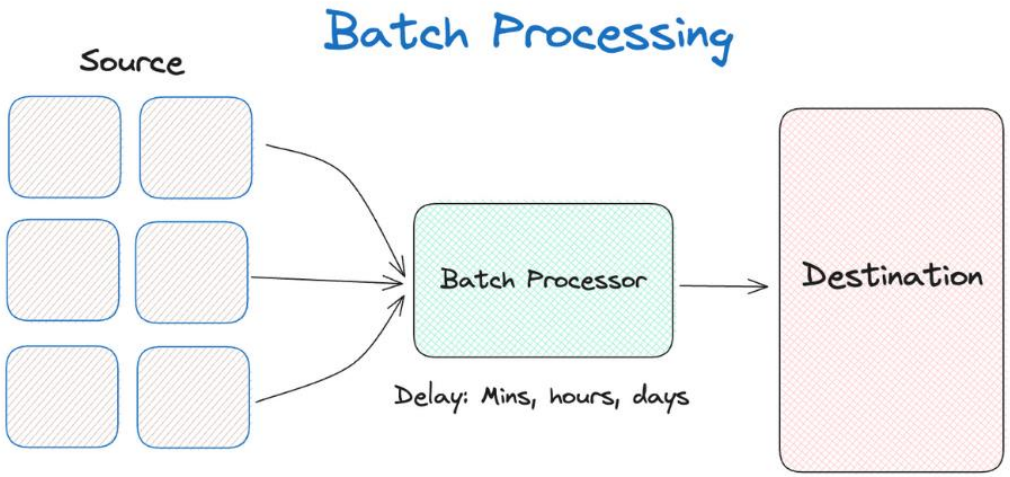
Recap: Data Lakehouse



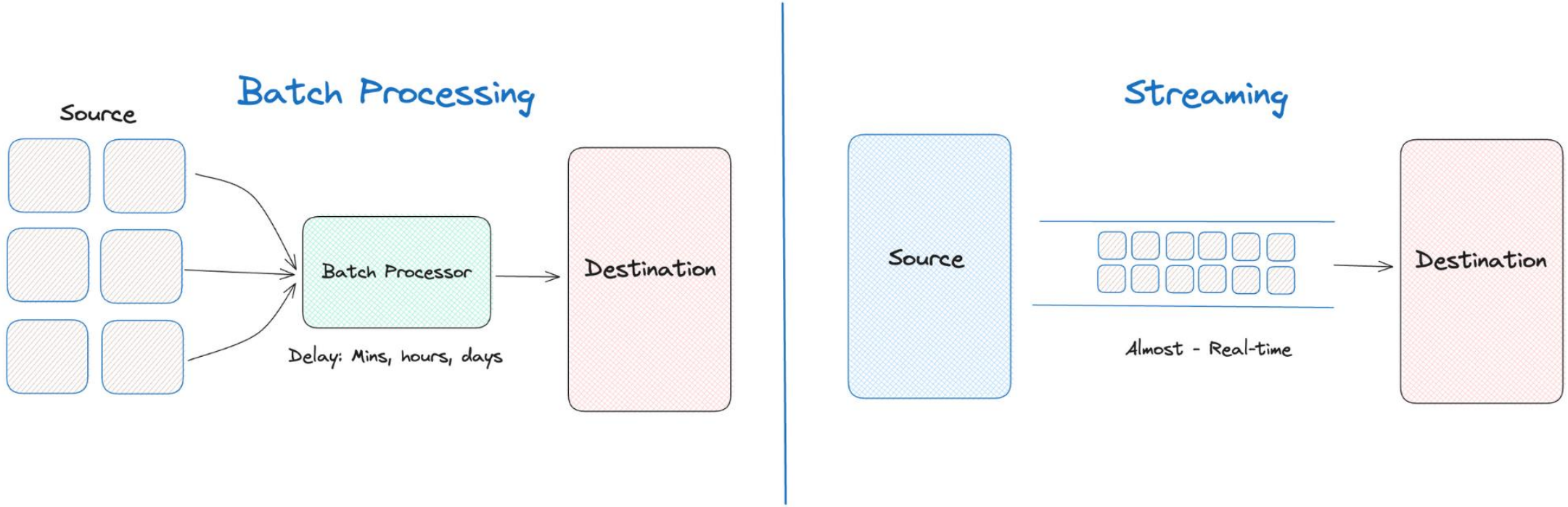
Data Streaming?



Data Streaming?



Data Streaming?



Motivation

The shift to Event-driven Systems has Already Begun …

From a static snapshot... **...to a continuous stream of events**



Occasional call to a friend



A constant feed about the activities of all your friends



Daily news reports



Real time news feeds, accessible online anytime, anywhere

Motivation

This leads us to …



A single platform to connect everyone to every event



Real-time streams of events



All events stored for historical view

Motivation

Successful Digital Businesses are Inherently Event-driven

Born cloud-native …



Social Networks
Enabling Event Sharing



Streaming Provider
On-demand Digital Content

Traditional ones that adapt …



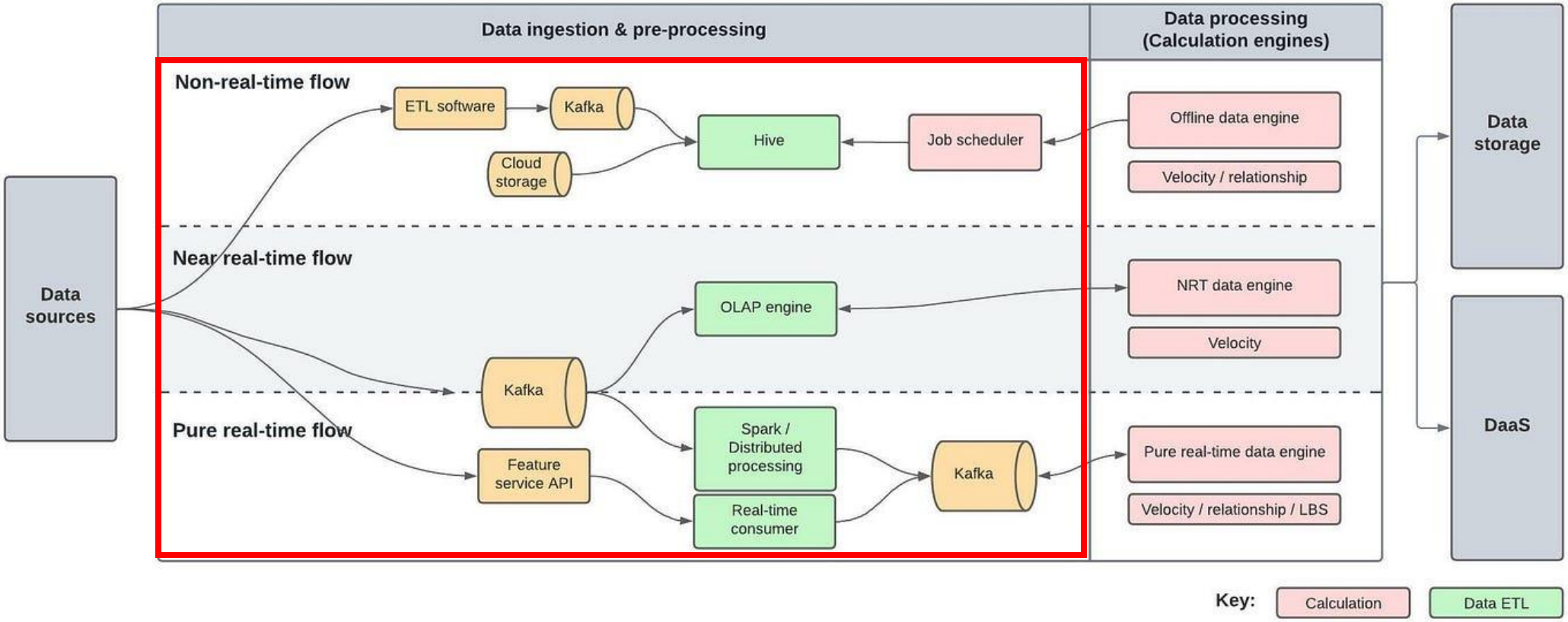
Newspaper
Provide a single Source of Truth



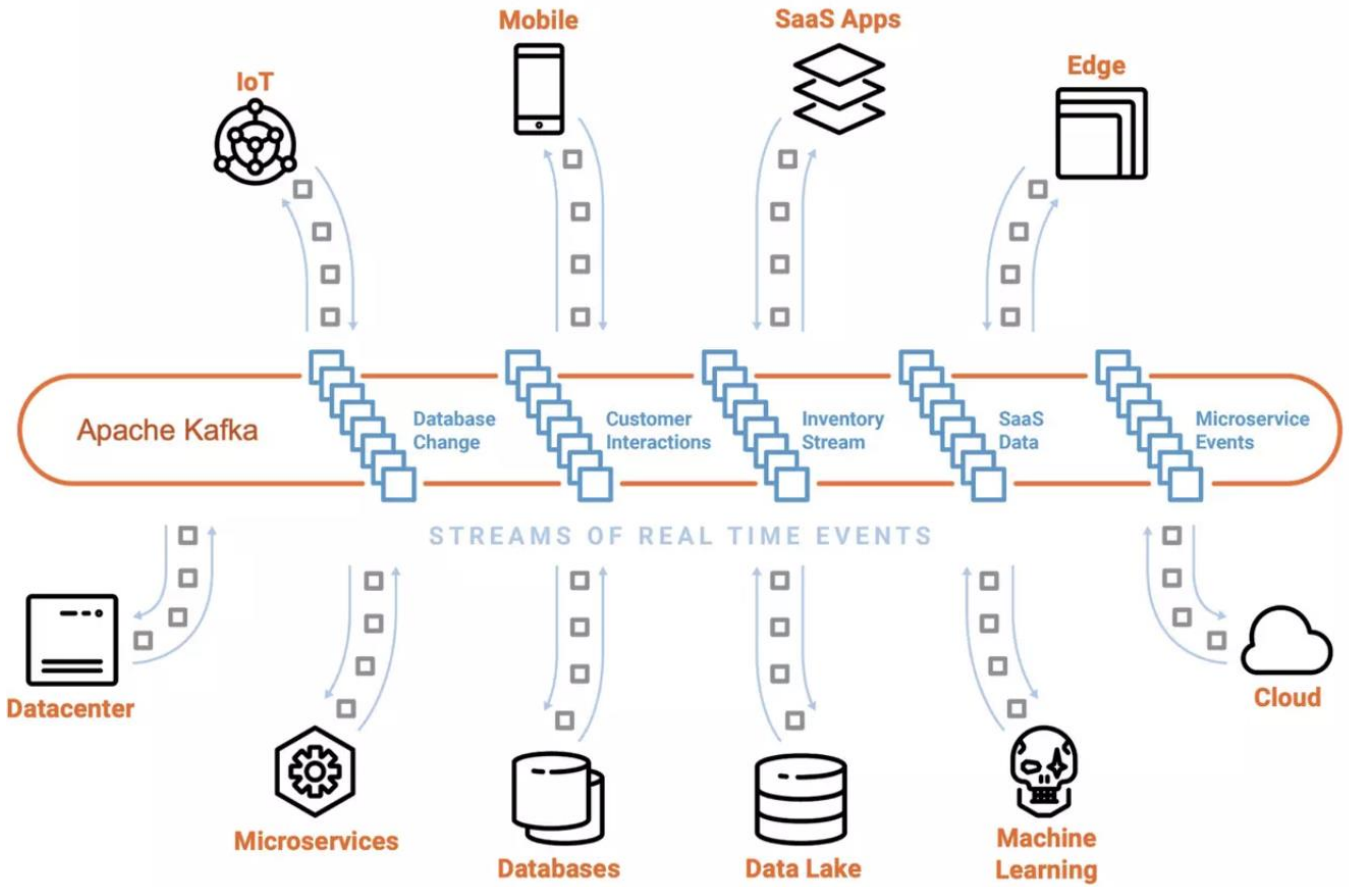
Credit Card Payments
Microservices Architecture

Motivation

coupang eats



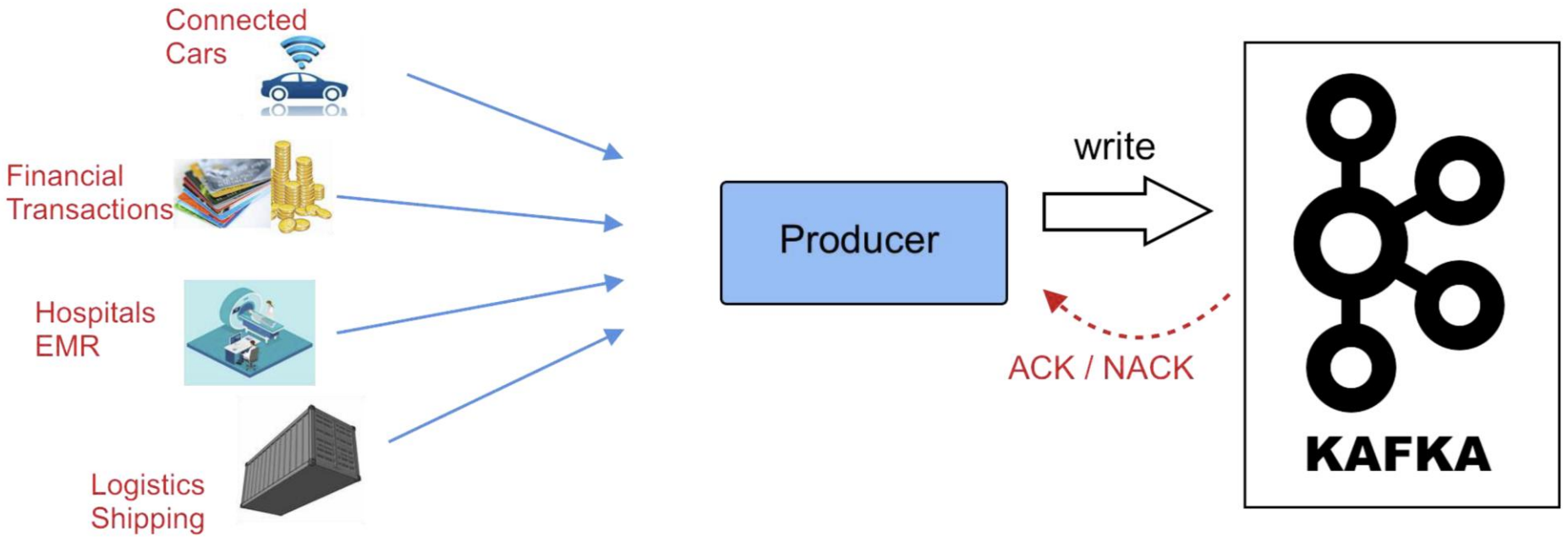
Kafka



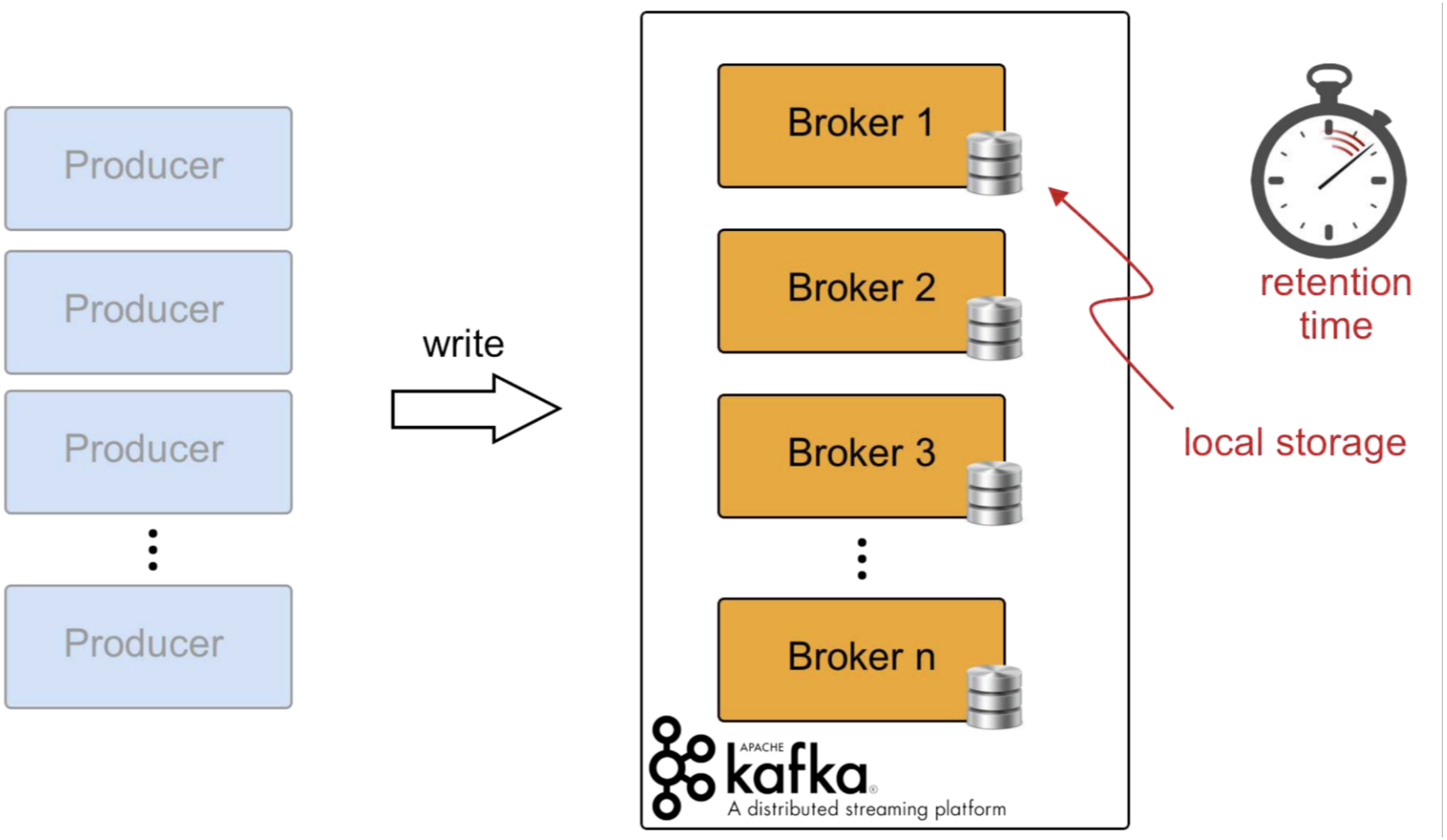
Apache Kafka: The De-facto Standard for Real-Time Event Streaming

- Global-scale
- Real-time
- Persistent Storage
- Stream Processing

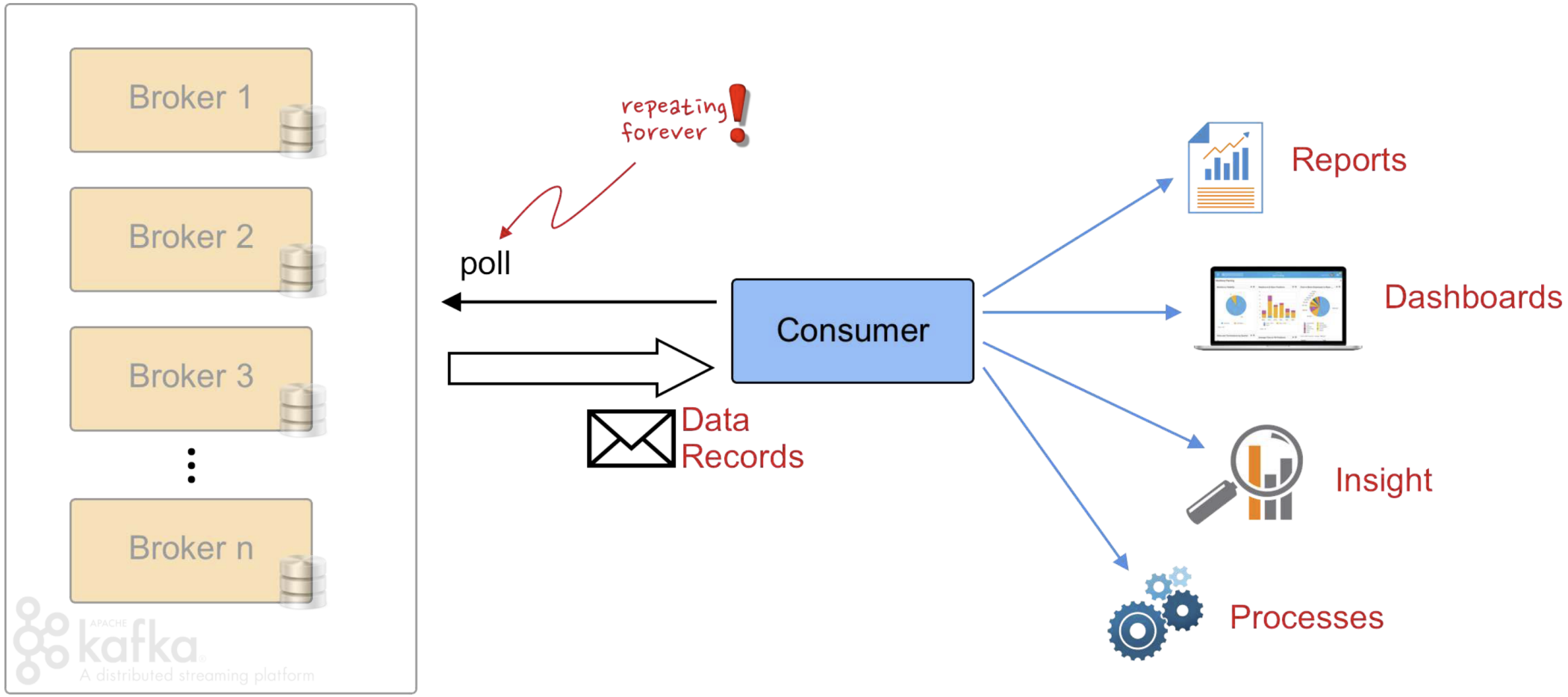
Kafka – Producers



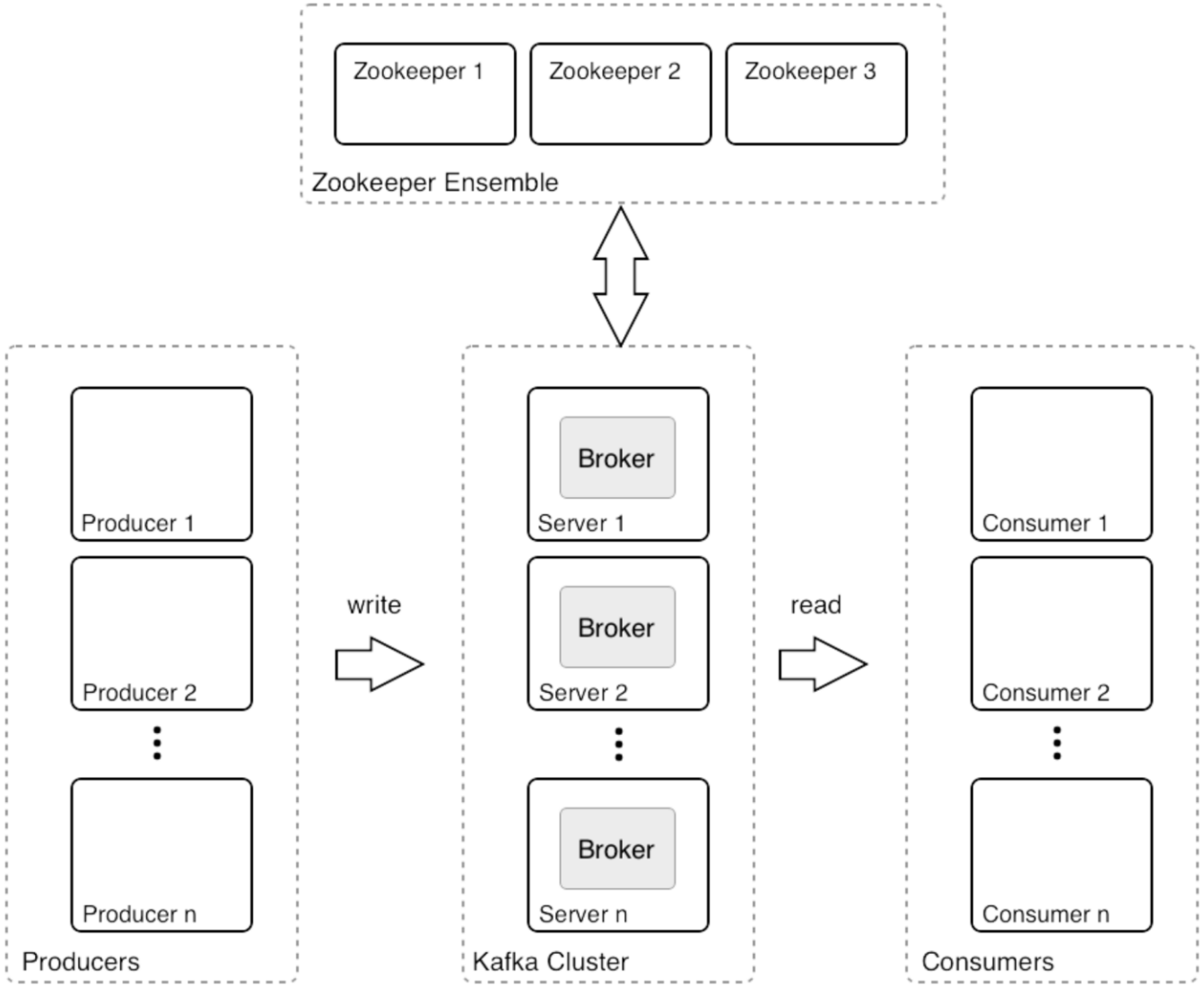
Kafka – Brokers



Kafka – Consumers

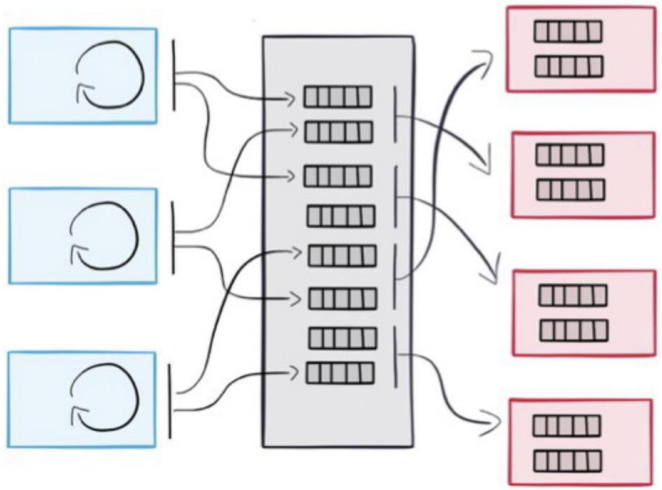


Kafka – Architecture

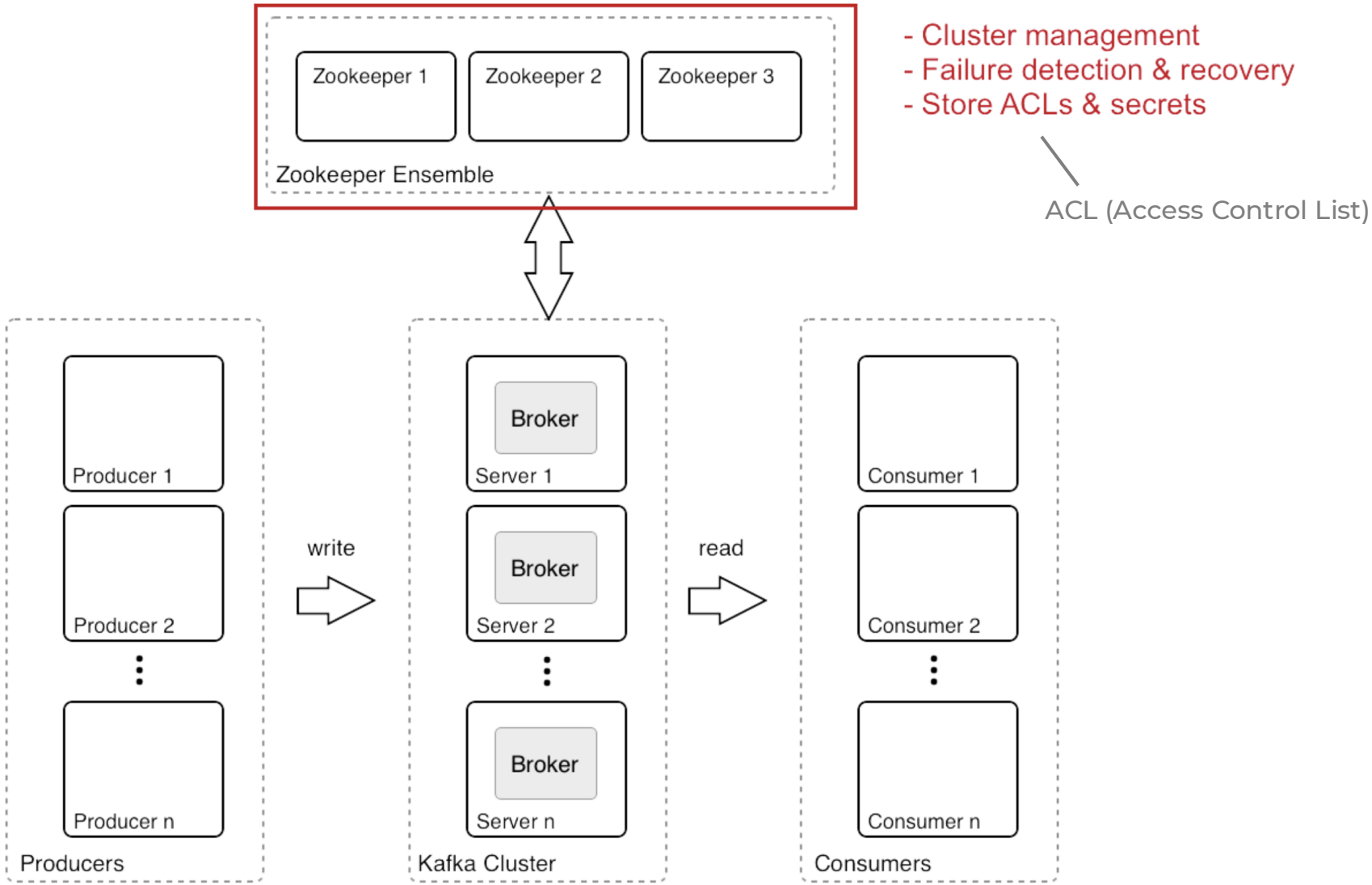


Decoupling Producers and Consumers

- Producers and Consumers are decoupled
- Slow Consumers do not affect Producers
- Add Consumers without affecting Producers
- Failure of Consumer does not affect System



Kafka – Zookeeper

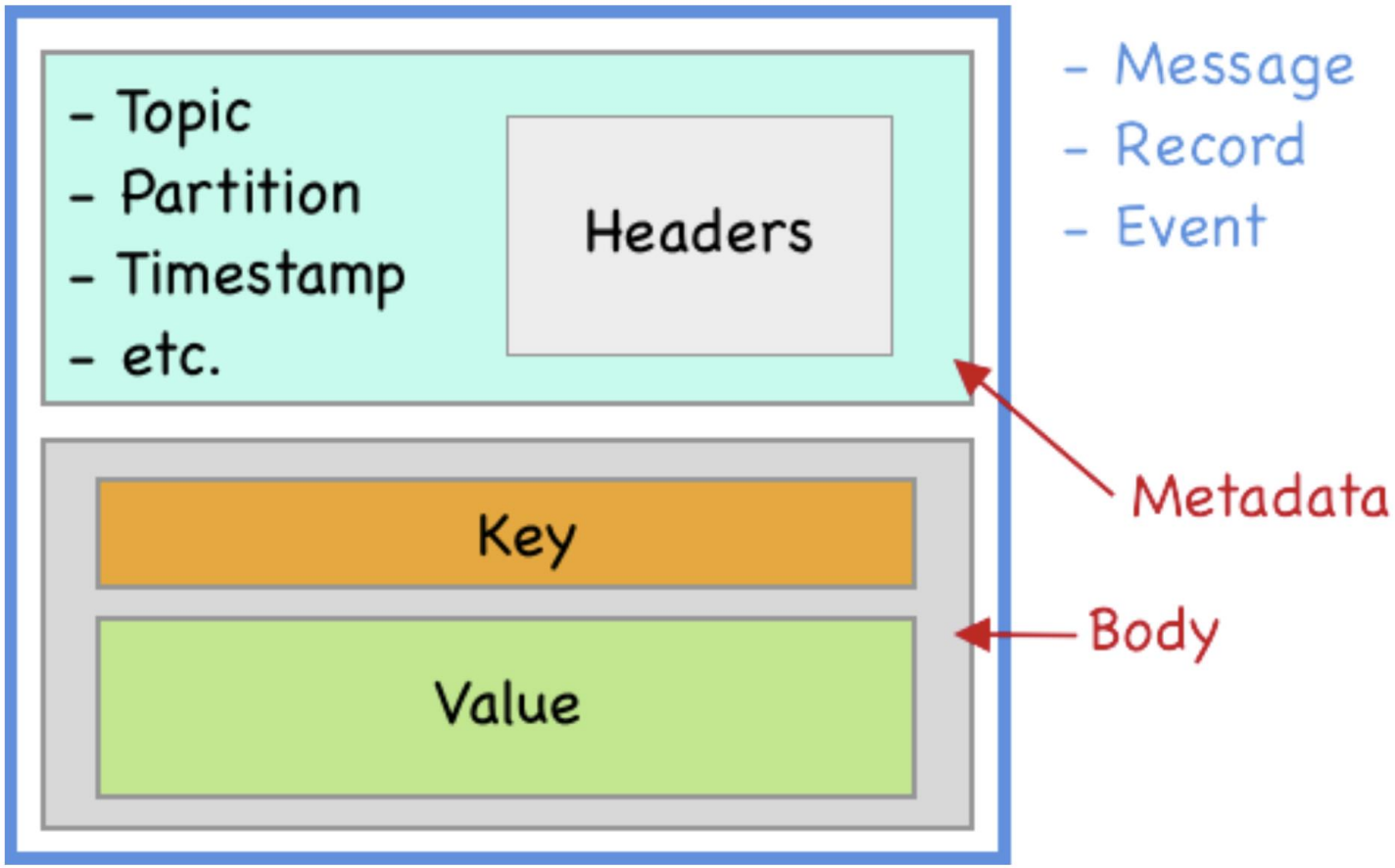


Kafka – Zookeeper



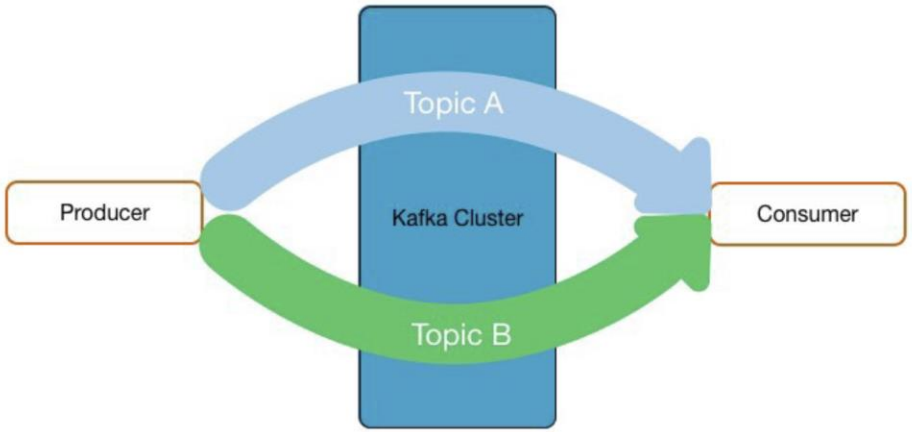
- **Open Source** Apache Project
- Distributed **Key Value Store**
- Maintains **configuration information**
- Stores **ACLs** and **Secrets**
- Enables highly reliable **distributed coordination**
- Provides **distributed synchronization**
- Three or five servers form an **ensemble**

Data Elements

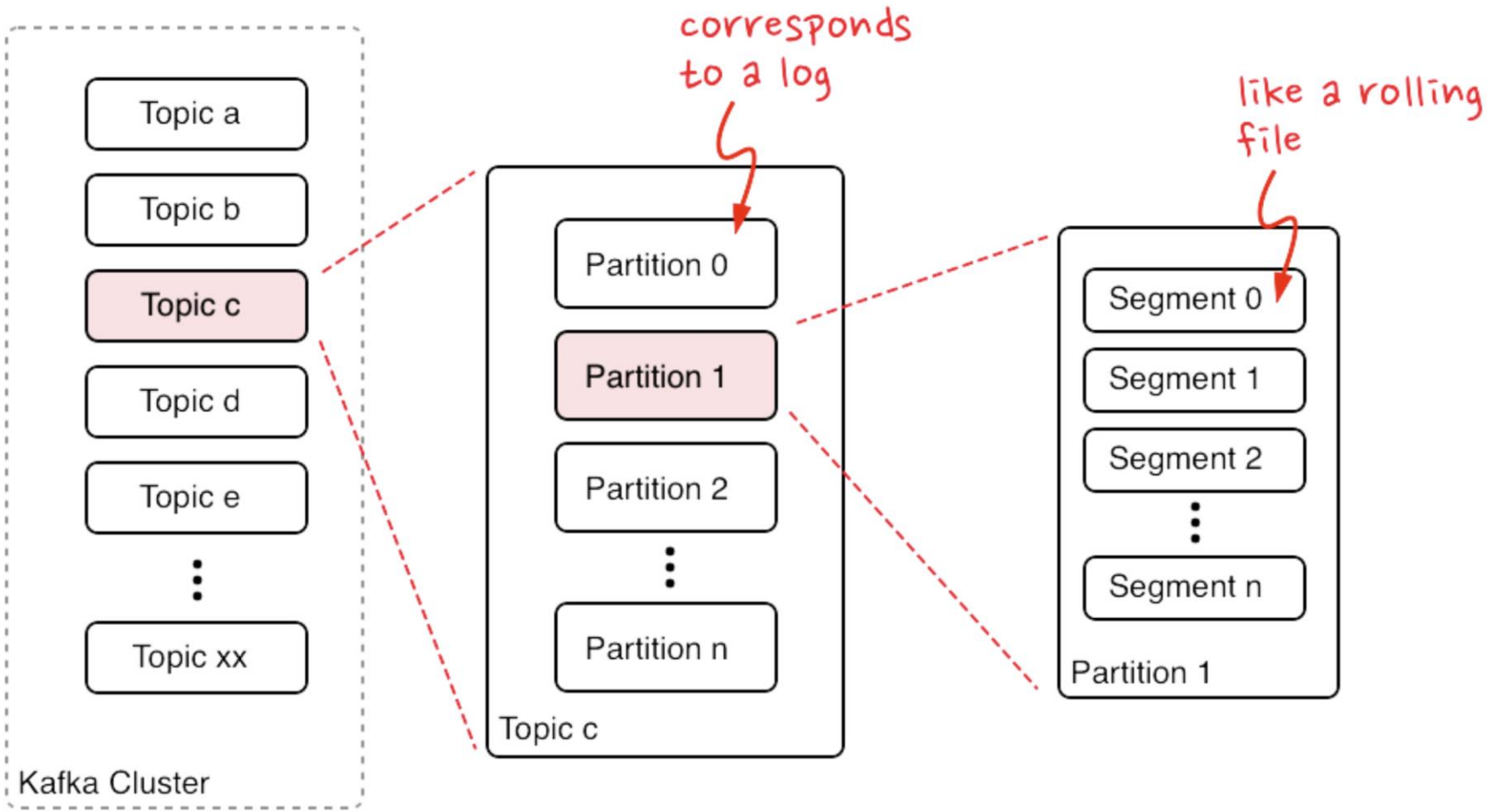


Topics

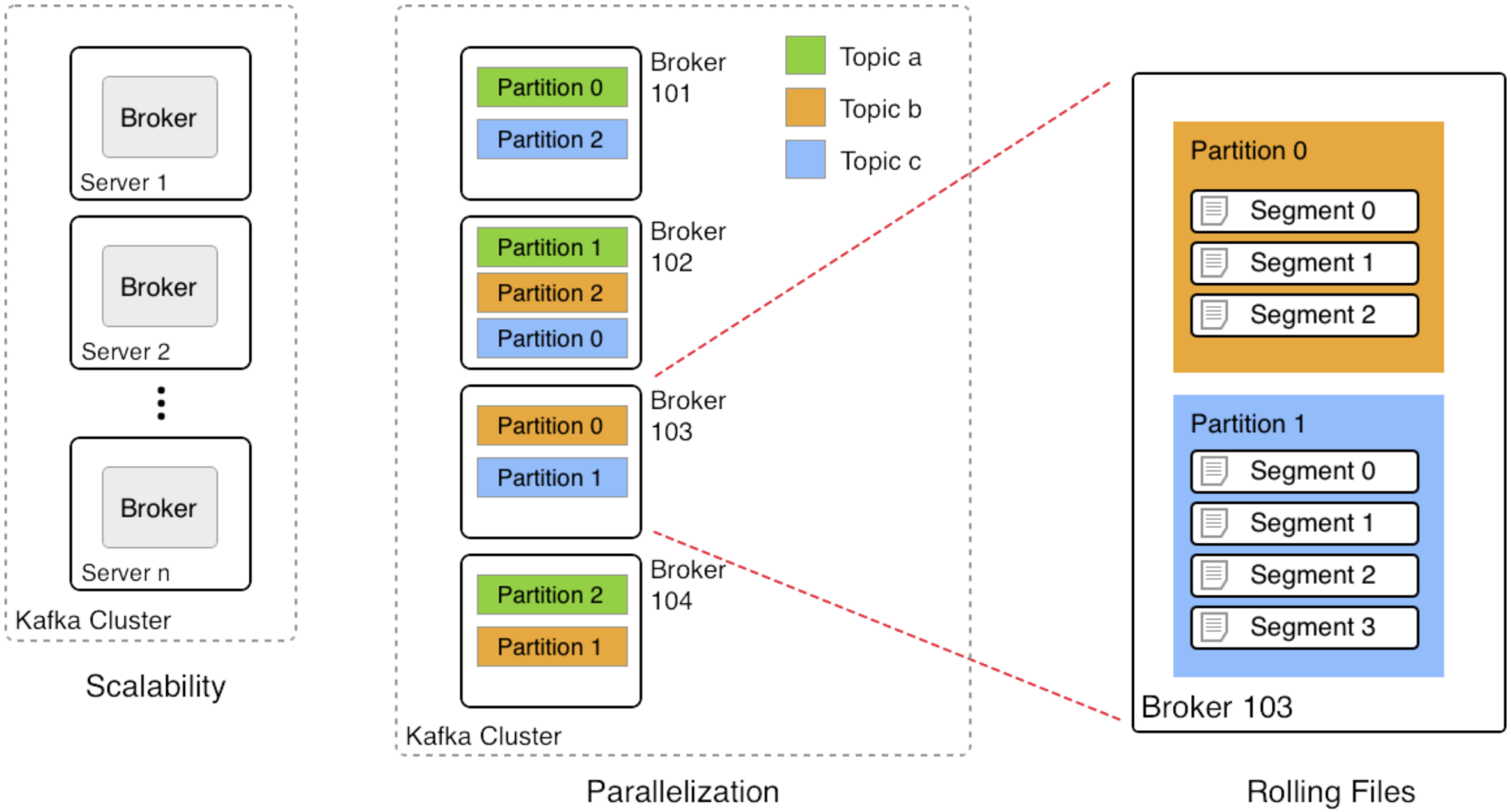
- **Topics**: Streams of “related” Messages in Kafka
  - Is a Logical Representation
  - Categorizes Messages into Groups
- Developers define Topics
- Producer ⟷ Topic: N to N Relation
- Unlimited Number of Topics



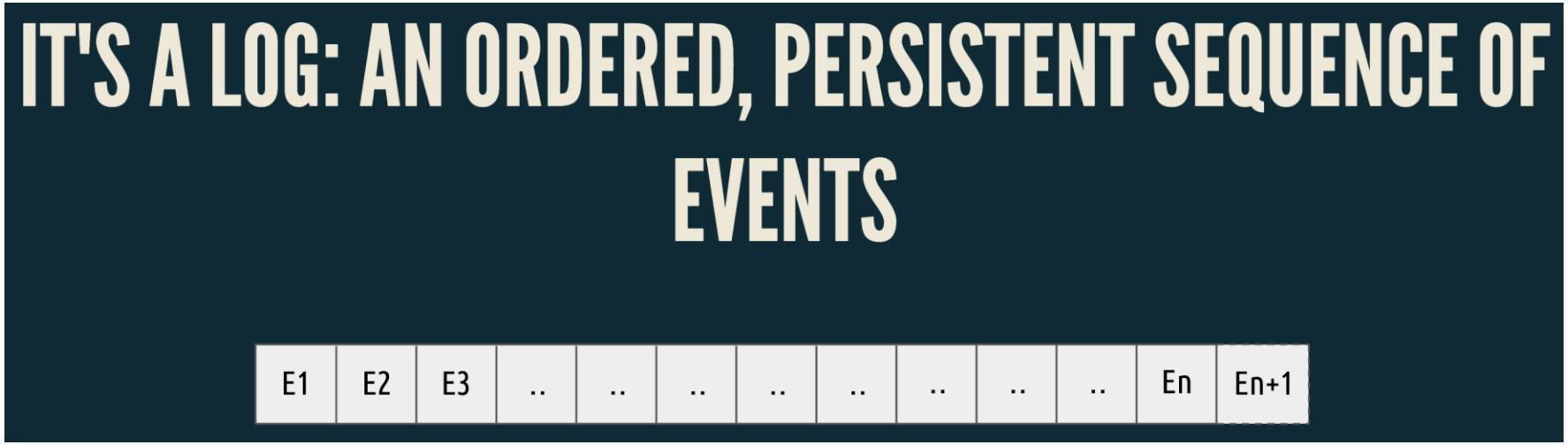
Topics, Partitions, and Segments



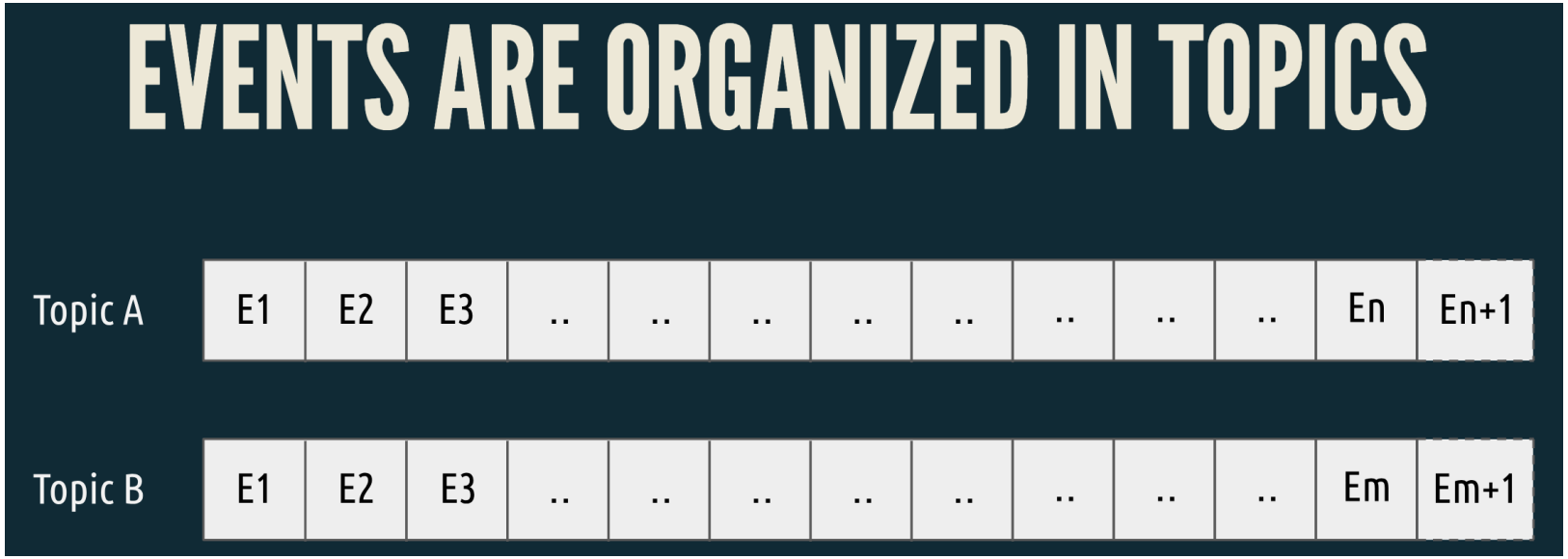
Topics, Partitions, and Segments



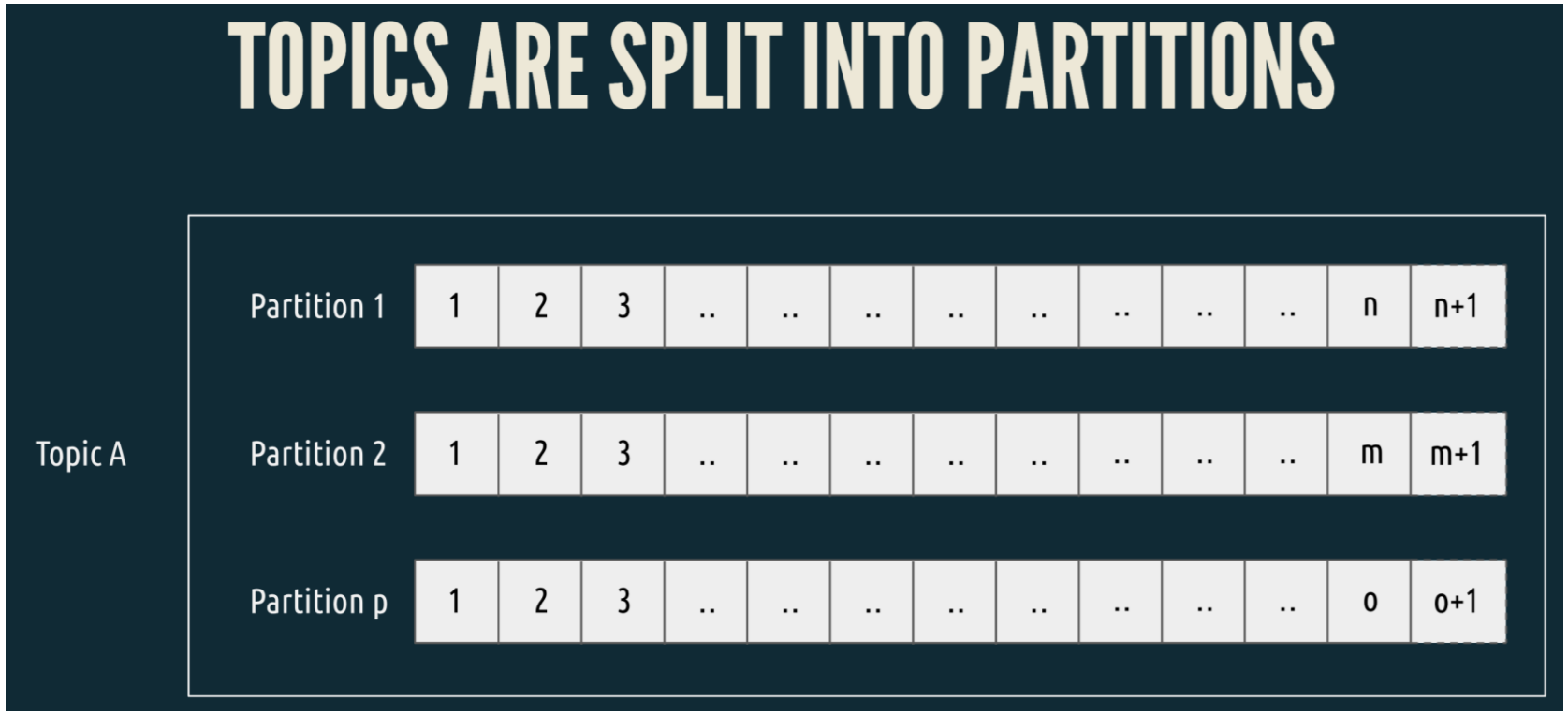
Topics, Partitions, and Segments



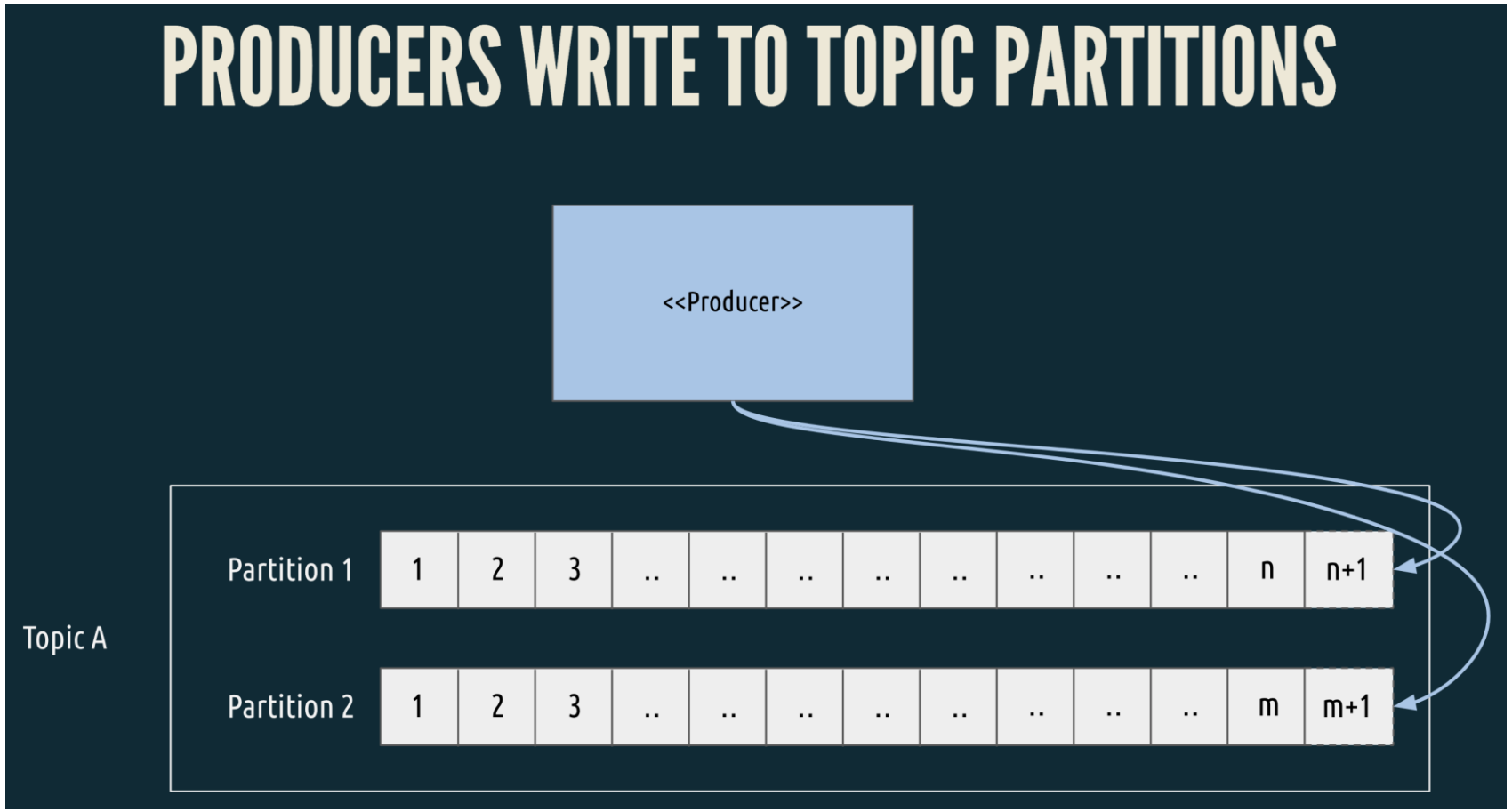
Topics, Partitions, and Segments



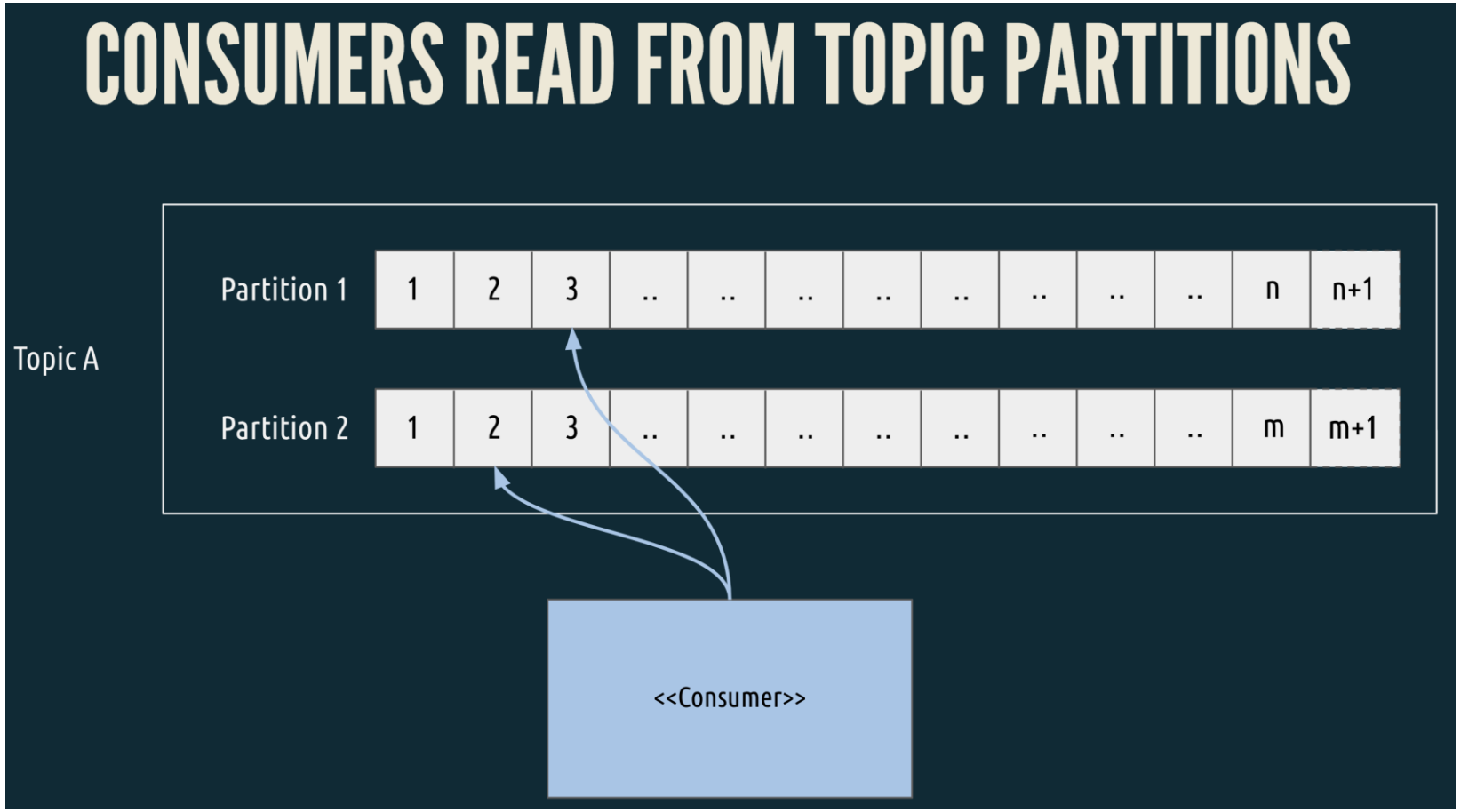
Topics, Partitions, and Segments



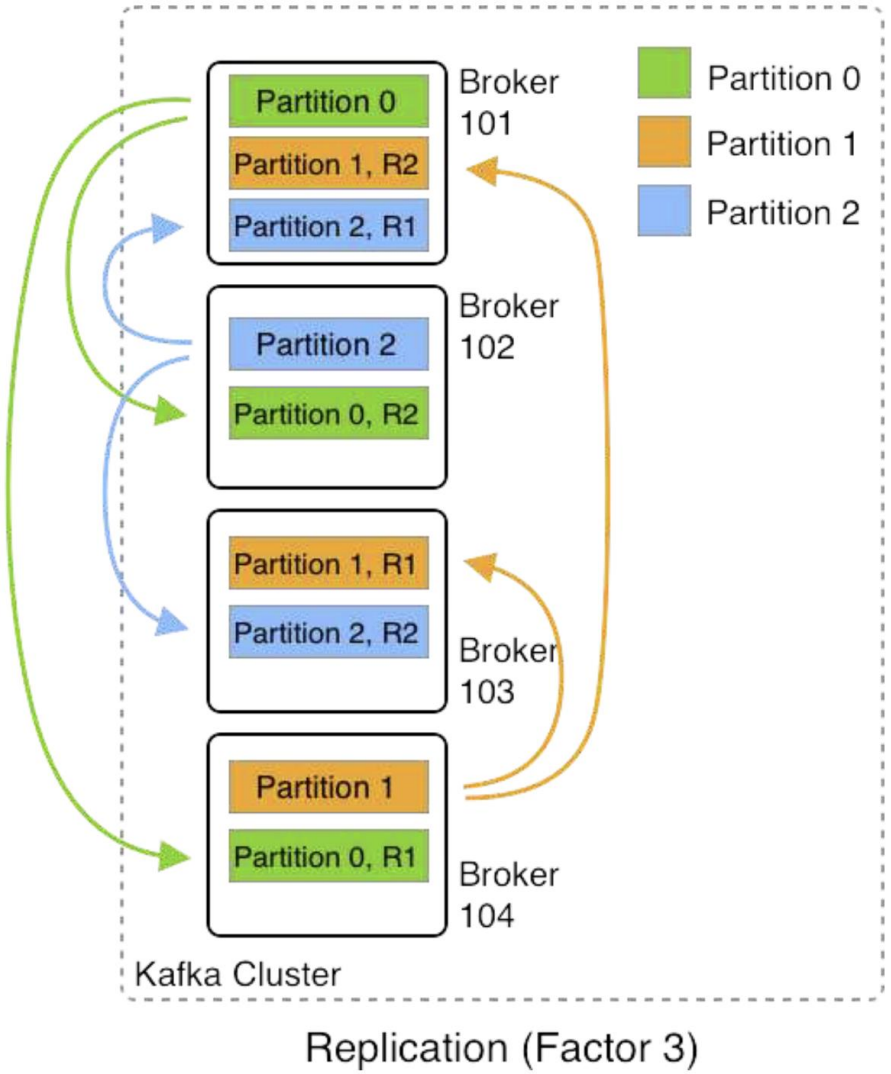
Topics, Partitions, and Segments



Topics, Partitions, and Segments

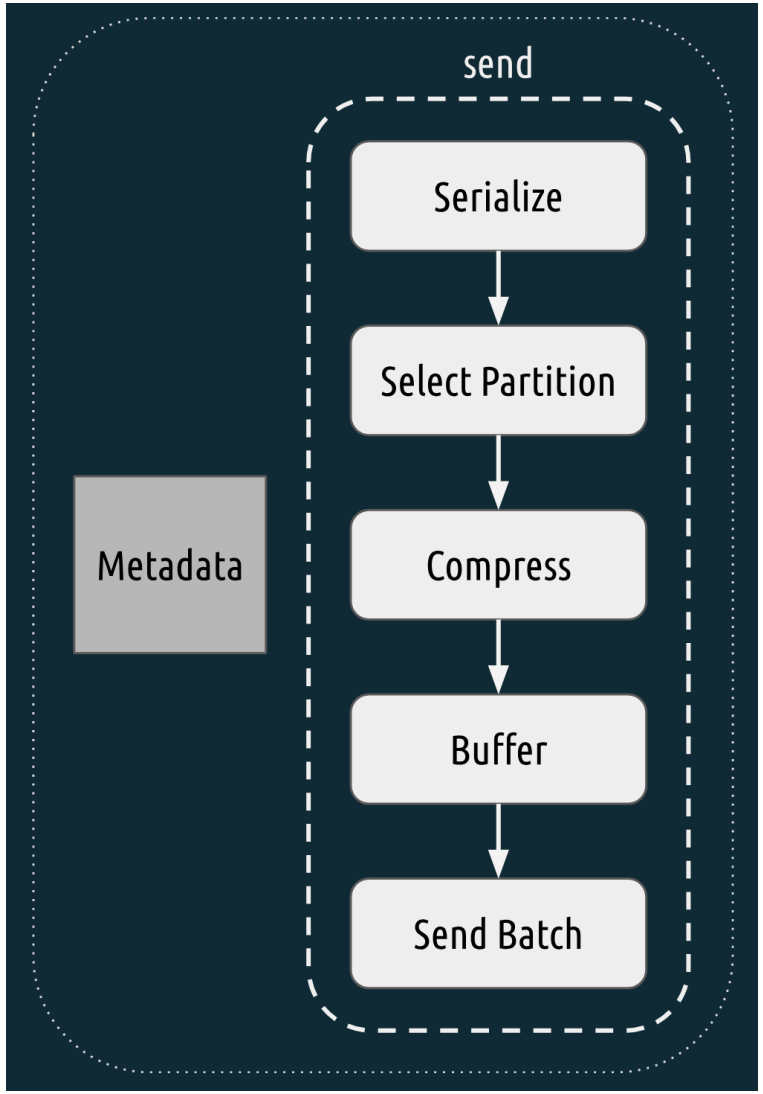


Broker Replication: Fault Tolerance



Replication (Factor 3)

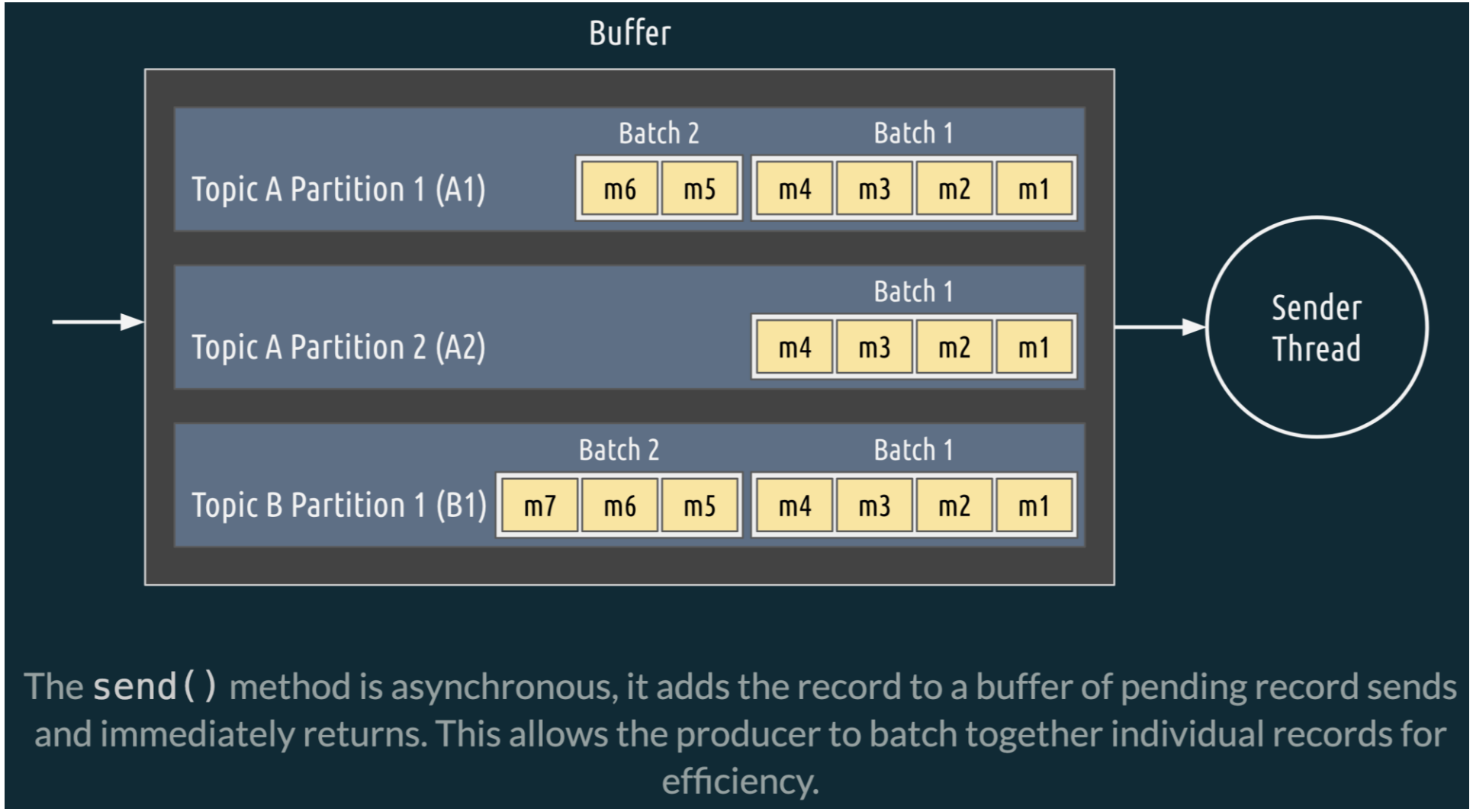
Producer


send
Serialize
Select Partition
Compress
Buffer
Send Batch
Metadata

Producer – Partitioning Strategy

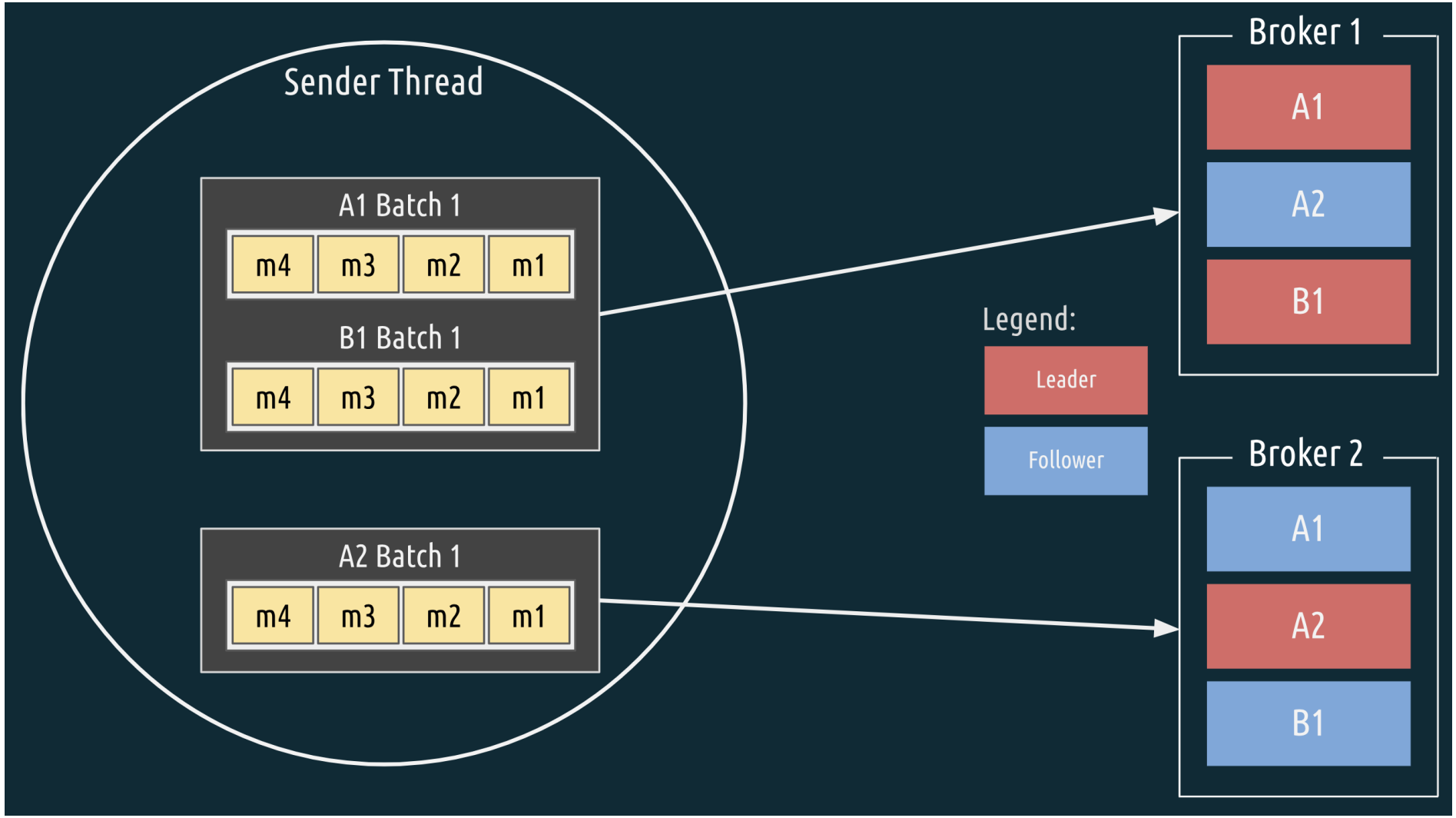
- Producers use a Partitioning Strategy to assign each message to a Partition
- Two Purposes:
  - Load Balancing
  - Semantic Partitioning
- Partitioning Strategy specified by Producer
  - Default Strategy: `hash(key) % number_of_partitions`
  - No Key → Round-Robin
- Custom Partitioner possible

Producer – Accumulate Messages



The `send()` method is asynchronous, it adds the record to a buffer of pending record sends and immediately returns. This allows the producer to batch together individual records for efficiency.

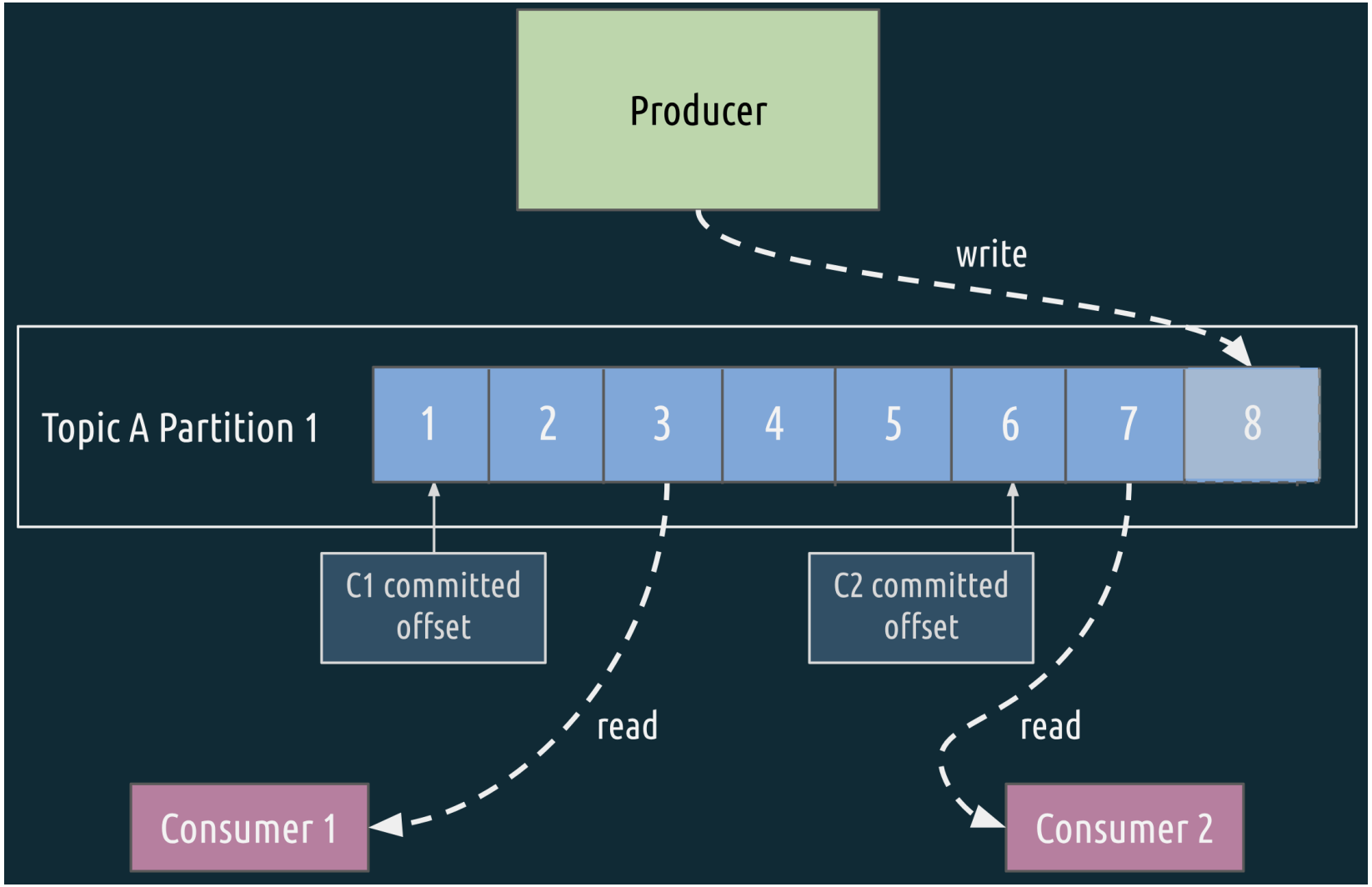
Producer – Send Message Batches



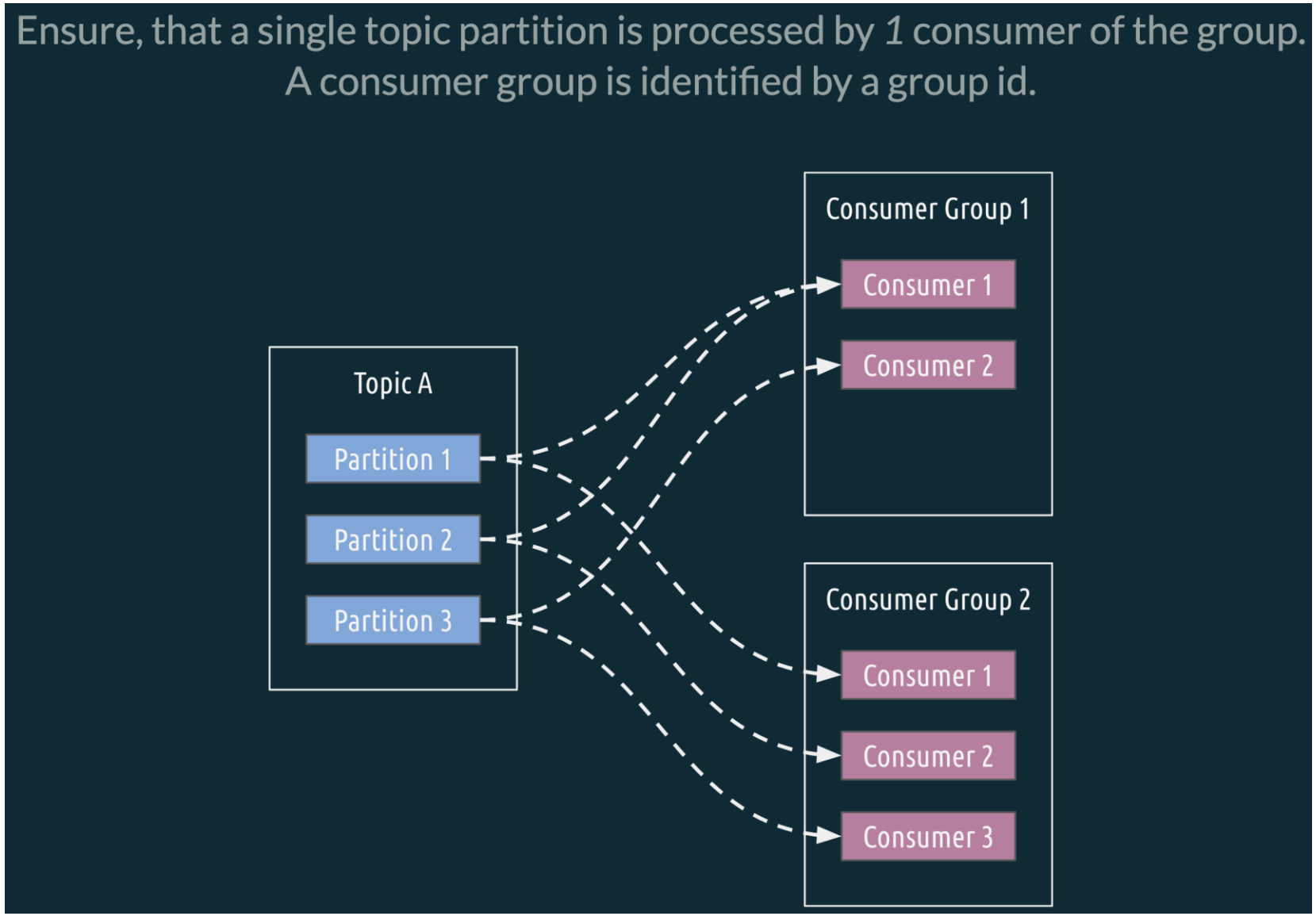
Consumer

- connect to broker(s) and cache metadata
- subscribe to topic(s)
- join consumer group
- continuously: poll records from broker(s) and commit offsets

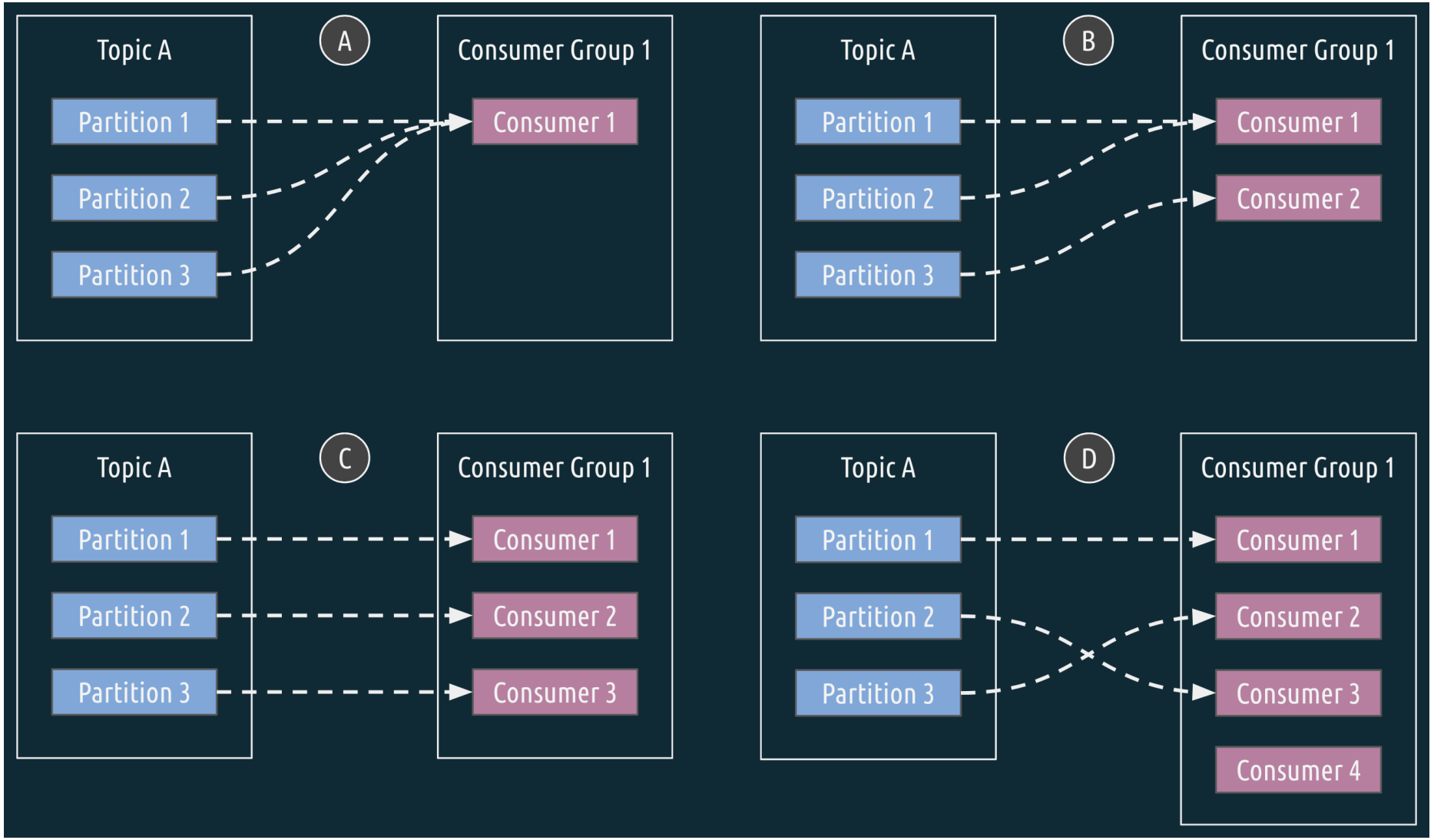
Consumer – Offsets



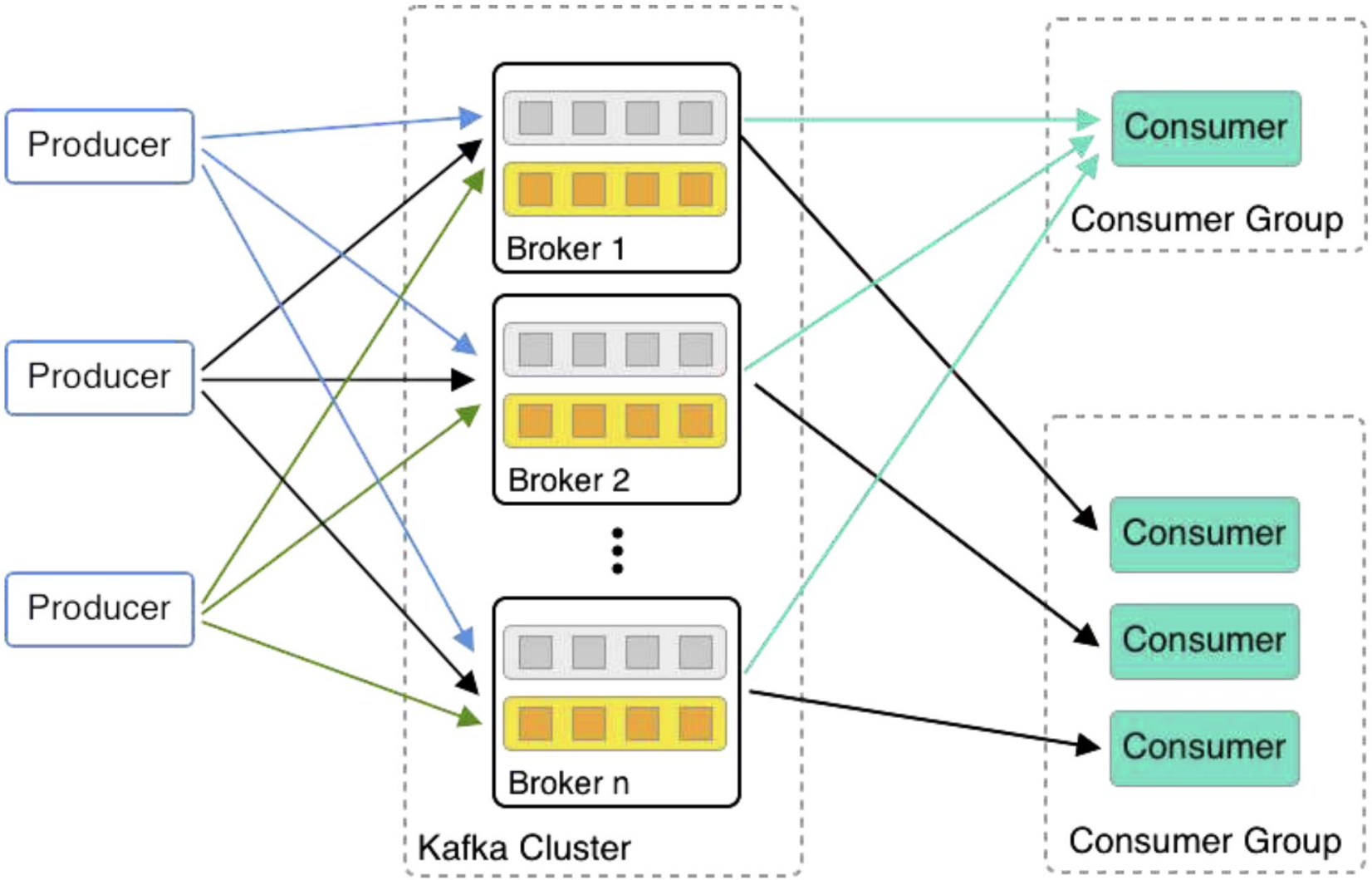
Consumer – Groups



Consumer – Group Rebalancing



Conclusion: Scalable Data Pipeline



Programming in Kafka

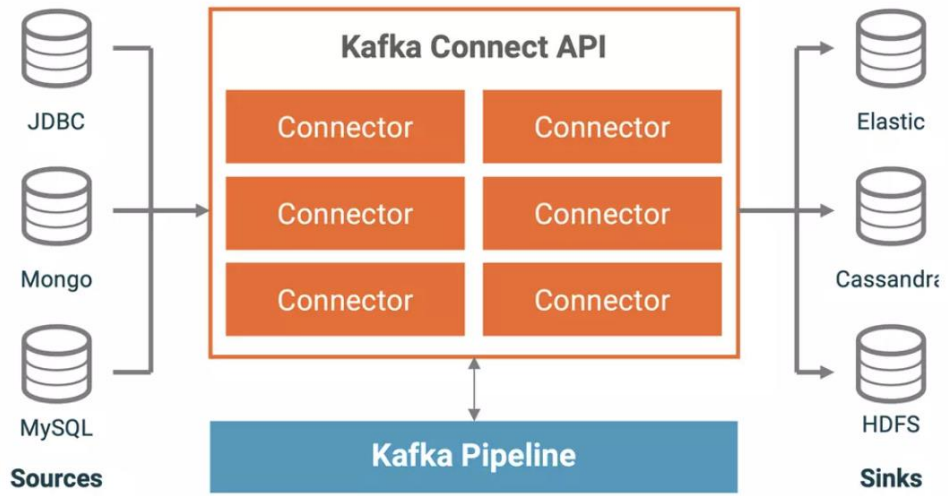


How do I get streams of data into and out of Kafka?

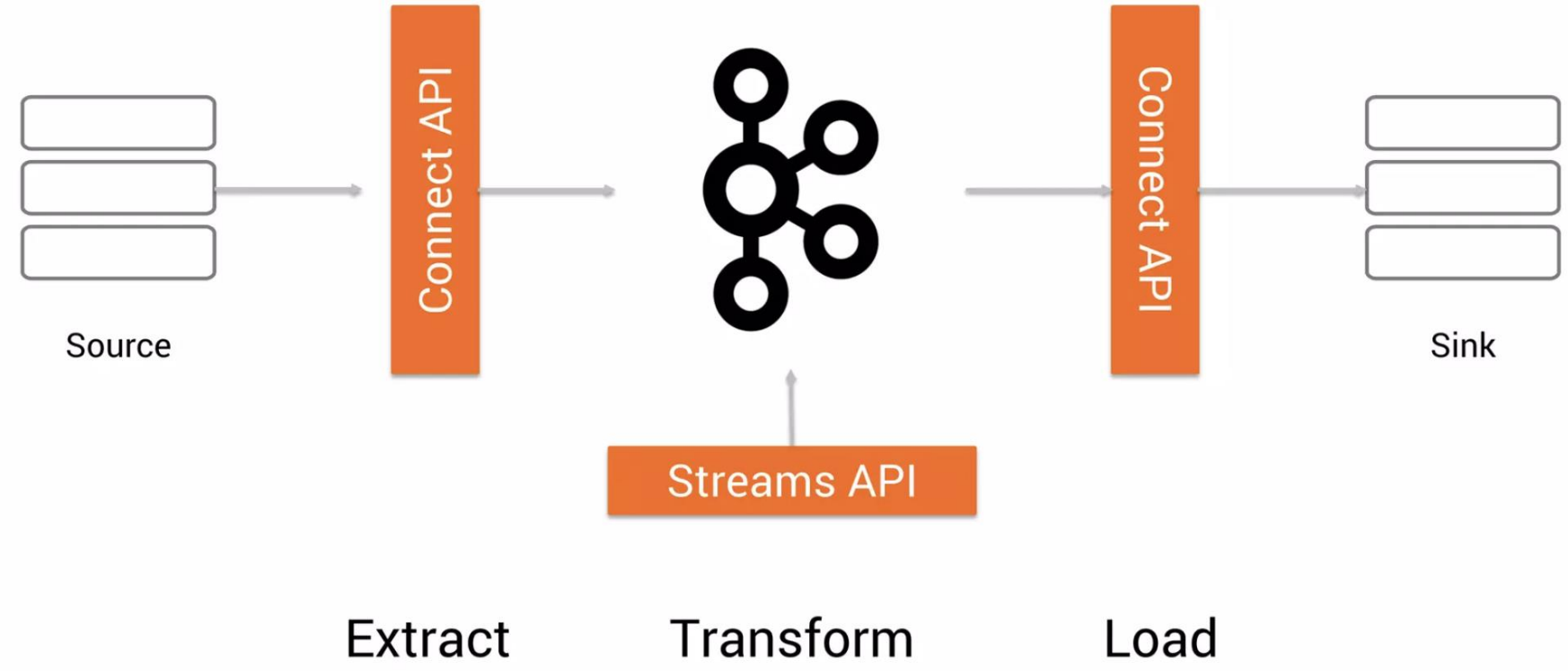
Kafka Connect

Import and Export Data In & Out of Kafka

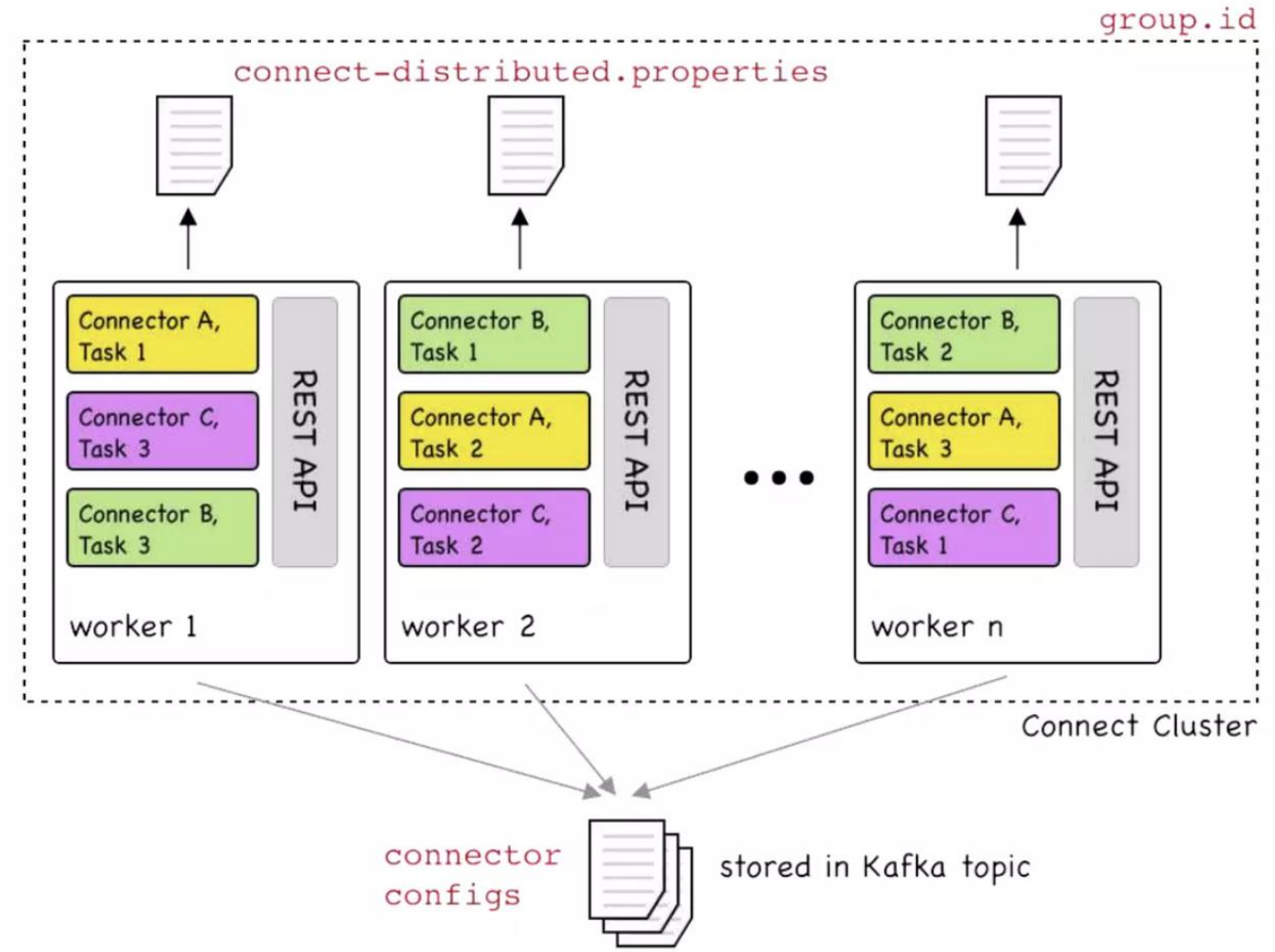
- Fault Tolerant
- Manage hundreds of data sources and sinks
- Preserves data schema
- Integrated within Confluent Control Center



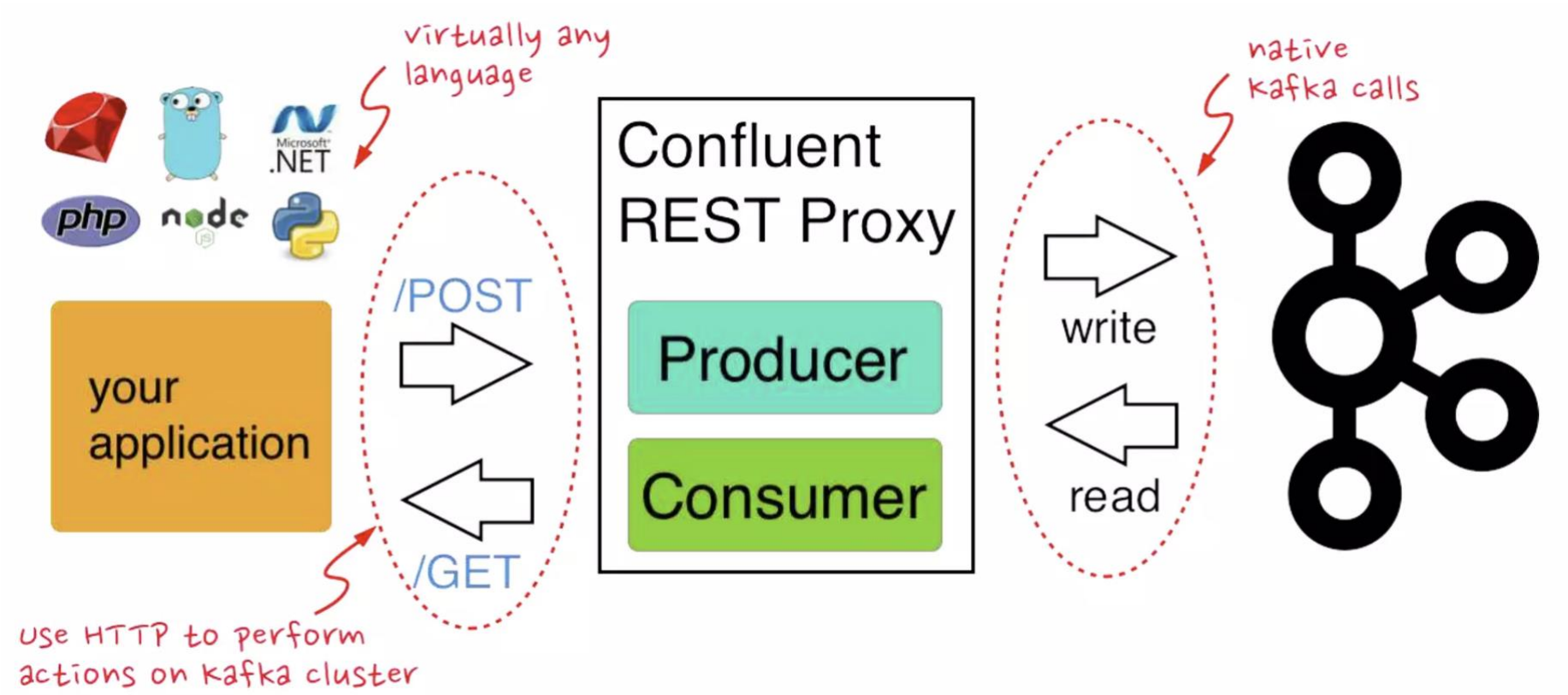
Kafka Connect



Kafka Connect



Kafka Connect – REST Proxy

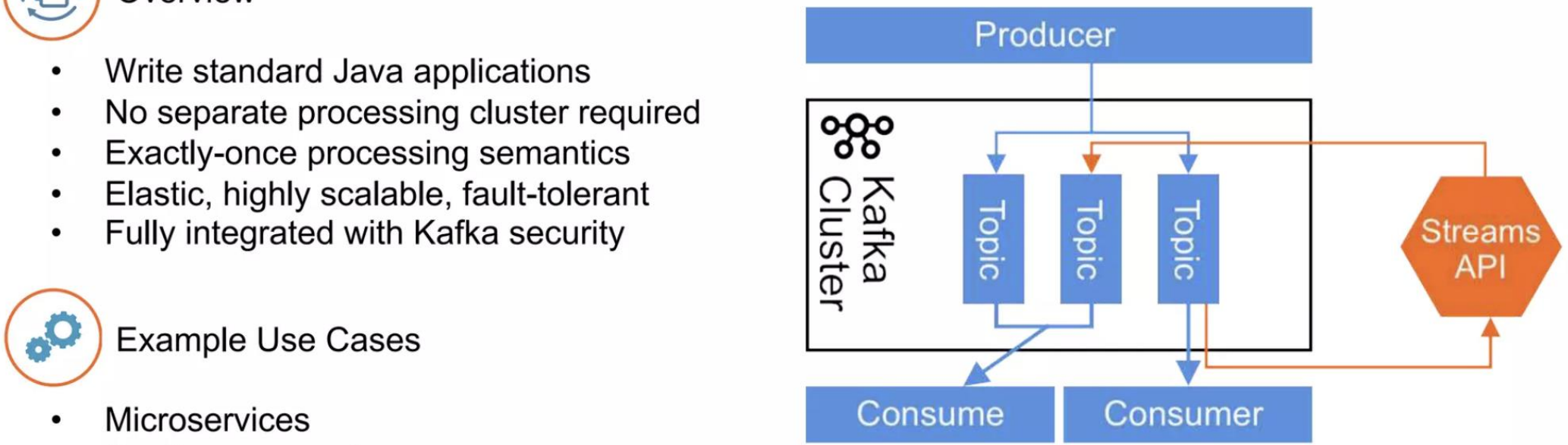


Kafka Streams

Transform Data with Real-Time Applications

Overview

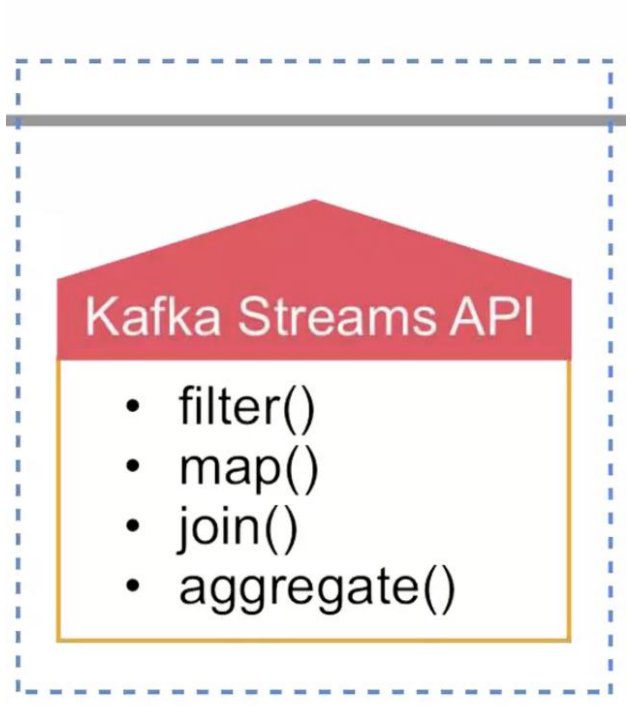
- Write standard Java applications
- No separate processing cluster required
- Exactly-once processing semantics
- Elastic, highly scalable, fault-tolerant
- Fully integrated with Kafka security



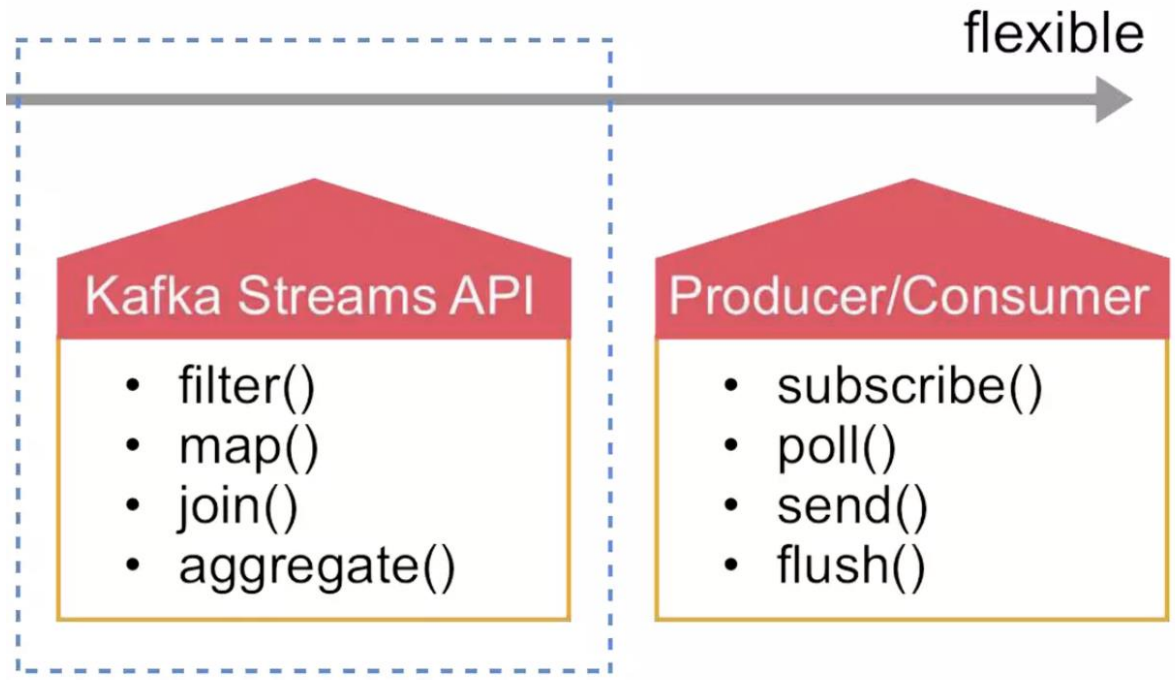
Example Use Cases

- Microservices
- Continuous queries
- Continuous transformations

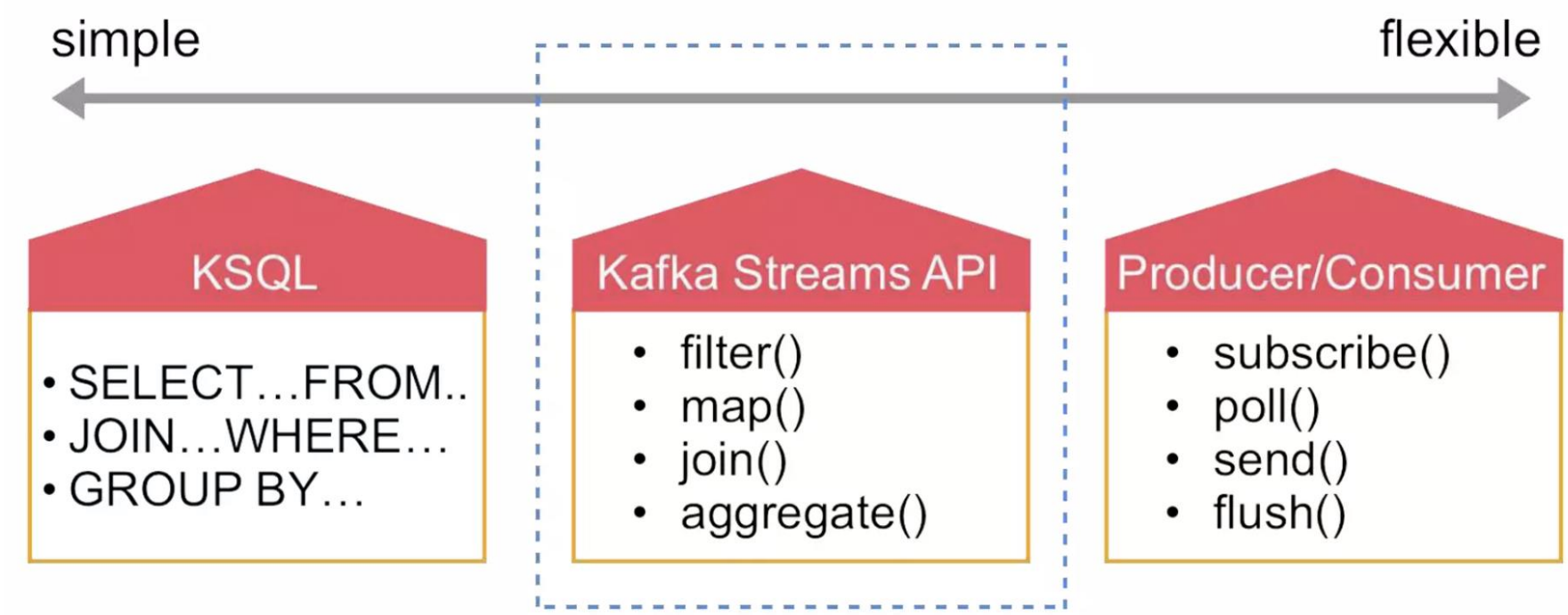
Kafka Streams



Kafka Streams



Kafka Streams



Confluent KSQL

Develop real-time stream processing apps writing only SQL!

No Java, Python, or other boilerplate to wrap around it.

```
ksql> CREATE STREAM vip_actions AS
      SELECT userid, page, action
      FROM clickstream c
      LEFT JOIN users u ON c.userid = u.user_id
      WHERE u.level = 'Platinum';
```

Example – Create Topic

So before you can write your first events, you must create a topic. Open another terminal session and run:

```
$ bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhos
```

All of Kafka's command line tools have additional options: run the `kafka-topics.sh` command without any arguments to display usage information. For example, it can also show you details such as the partition count of the new topic:

```
$ bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localh
Topic: quickstart-events        TopicId: NPmZHyhbR9y00wMglMH2sg PartitionCount: 1
Topic: quickstart-events Partition: 0    Leader: 0   Replicas: 0 Isr: 0
```

Example – Add Events to the Topic

A Kafka client communicates with the Kafka brokers via the network for writing (or reading) events. Once received, the brokers will store the events in a durable and fault-tolerant manner for as long as you need—even forever.

Run the console producer client to write a few events into your topic. By default, each line you enter will result in a separate event being written to the topic.

```
$ bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhc
>This is my first event
>This is my second event
```

Example – Process Events with Kafka Streams

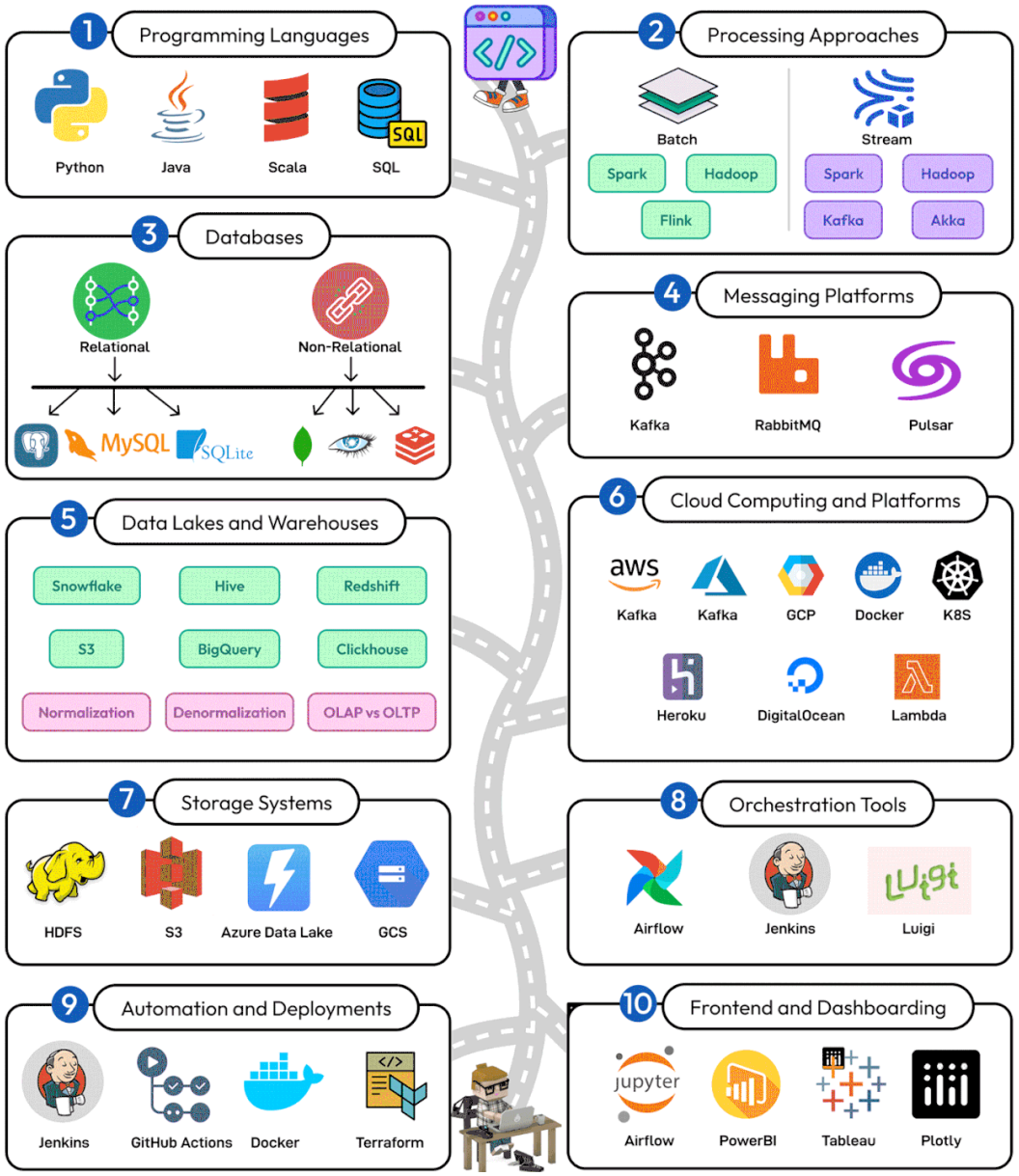
To give you a first taste, here's how one would implement the popular `WordCount` algorithm:

```
KStream<String, String> textLines = builder.stream("quickstart-events");

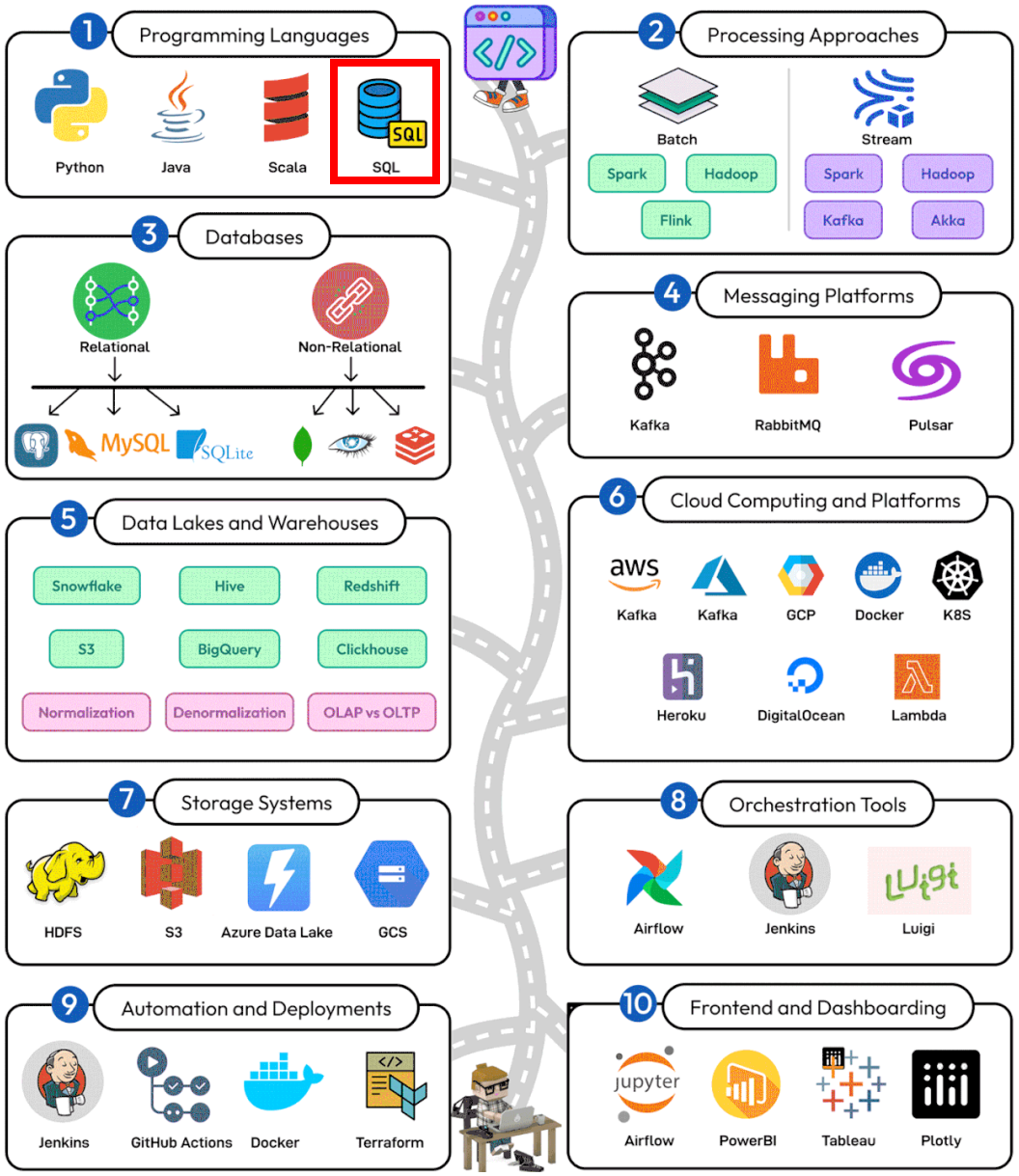
KTable<String, Long> wordCounts = textLines
            .flatMapValues(line -> Arrays.asList(line.toLowerCase().split(" ")))
            .groupBy((keyIgnored, word) -> word)
            .count();

wordCounts.toStream().to("output-topic", Produced.with(Serdes.String(), Serdes.
```

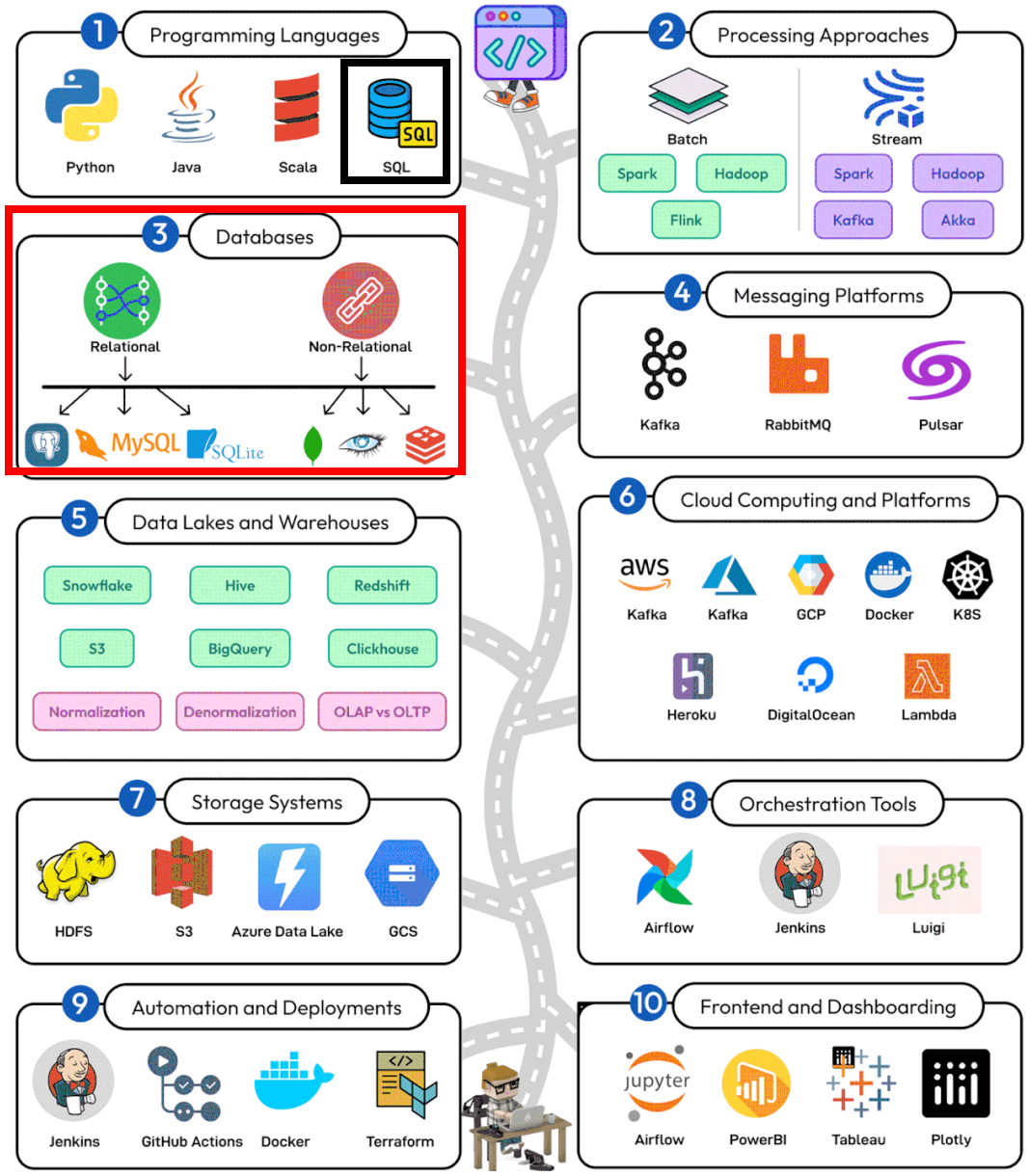
Summary: Full Stack Data Engineer! 🙌



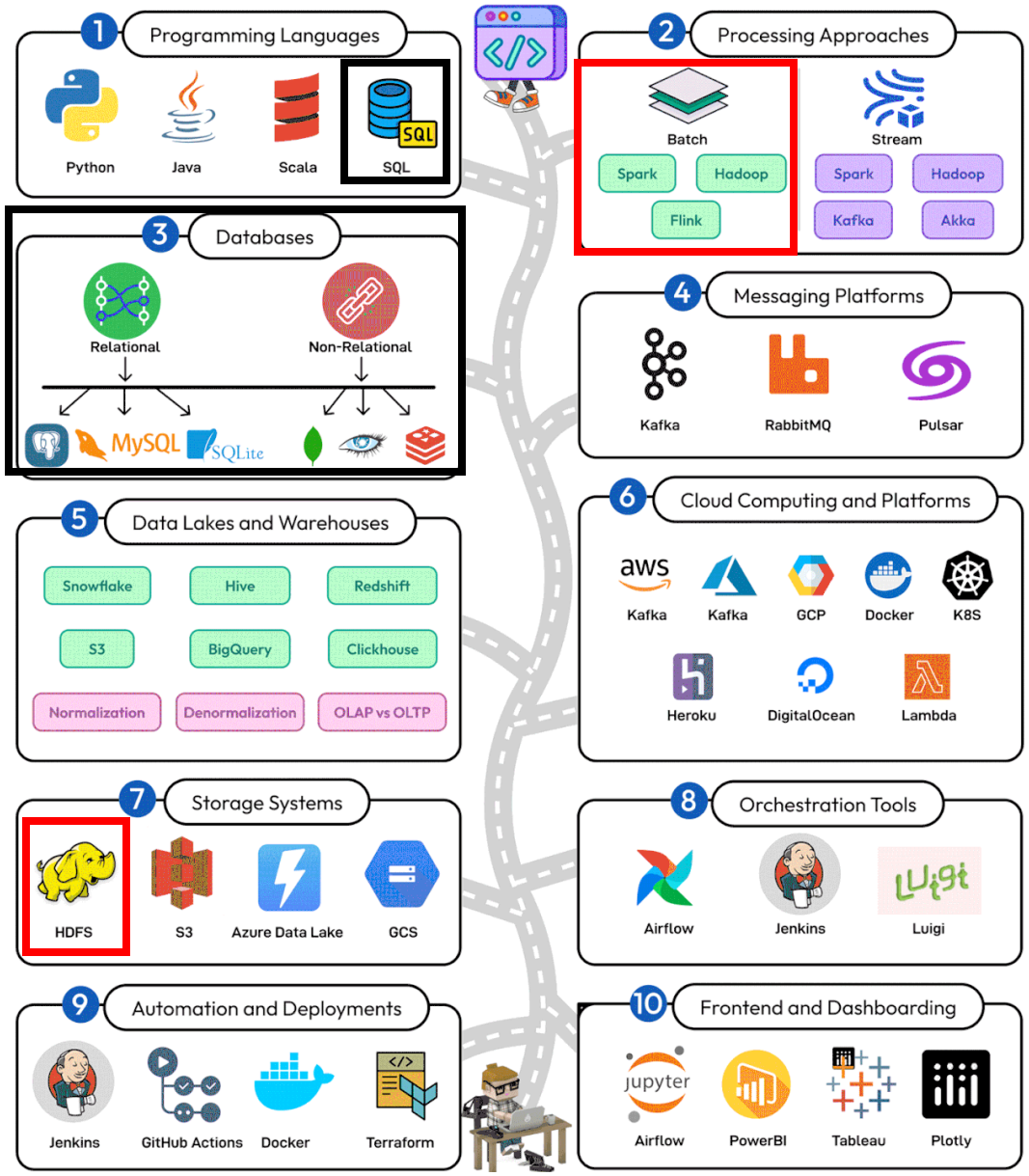
Summary: Full Stack Data Engineer! 🙌



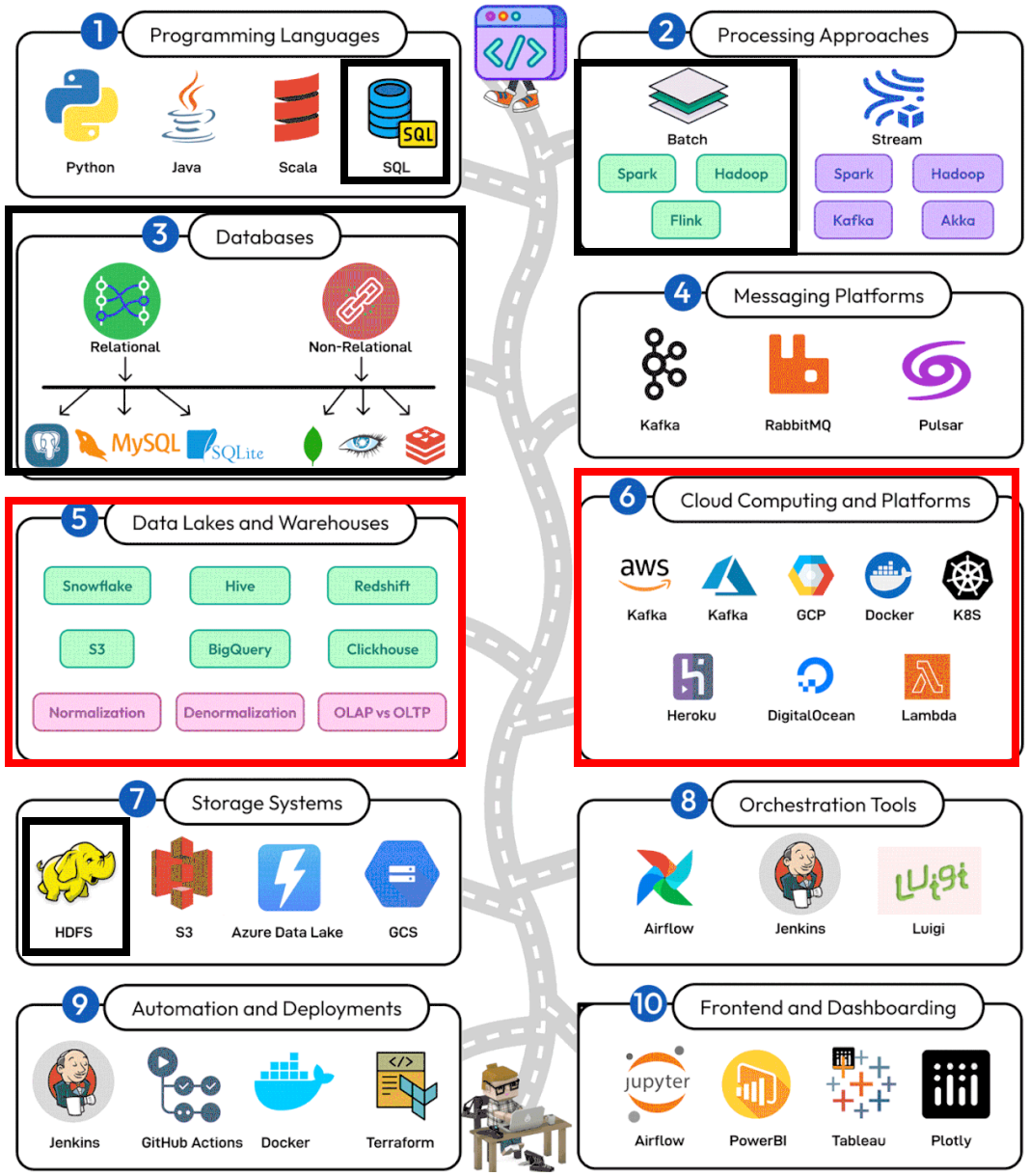
Summary: Full Stack Data Engineer! 🙌



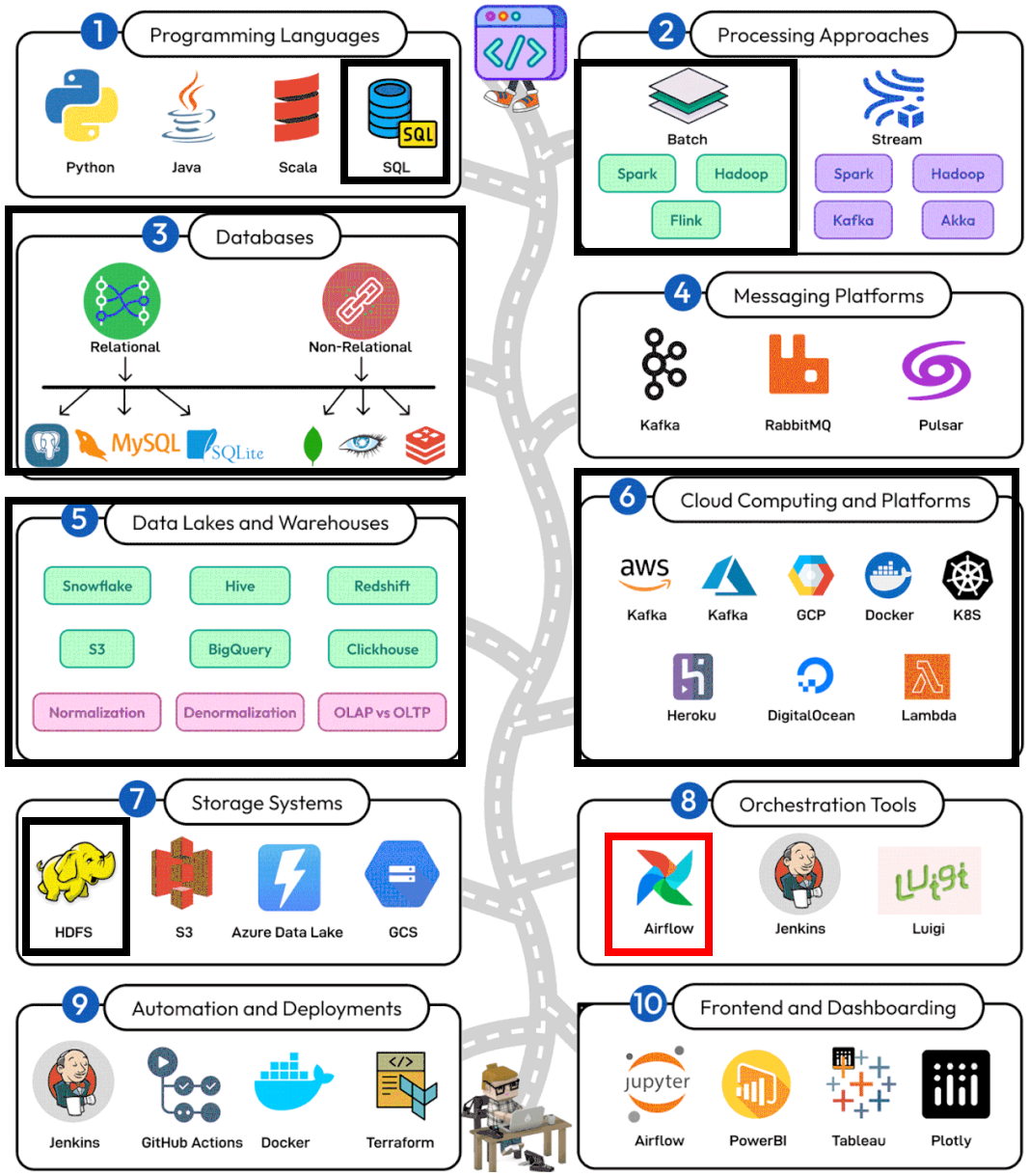
Summary: Full Stack Data Engineer! 🙌



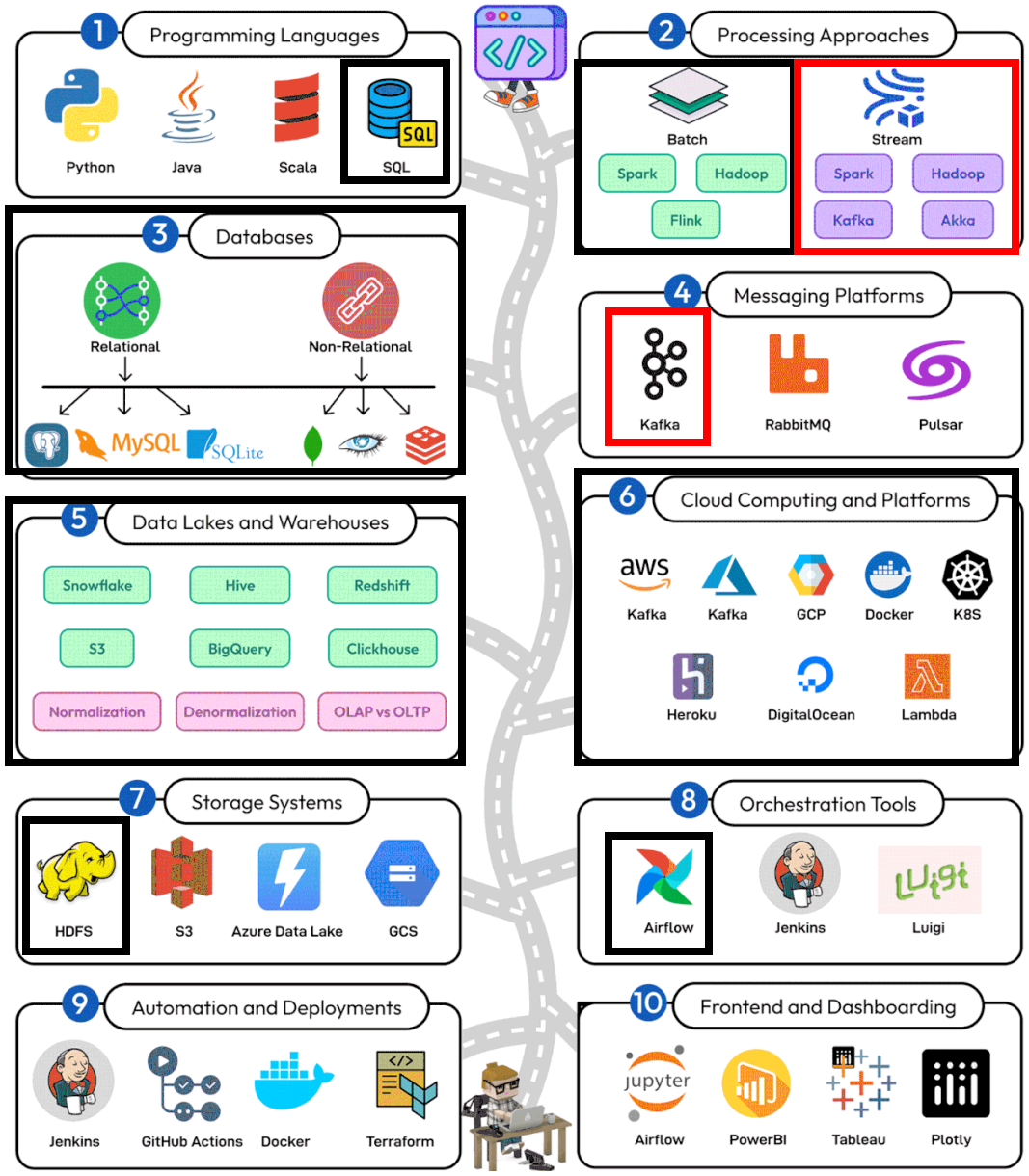
Summary: Full Stack Data Engineer! 🙌



Summary: Full Stack Data Engineer! 🙌



Summary: Full Stack Data Engineer! 🙌



Next: ML Systems

