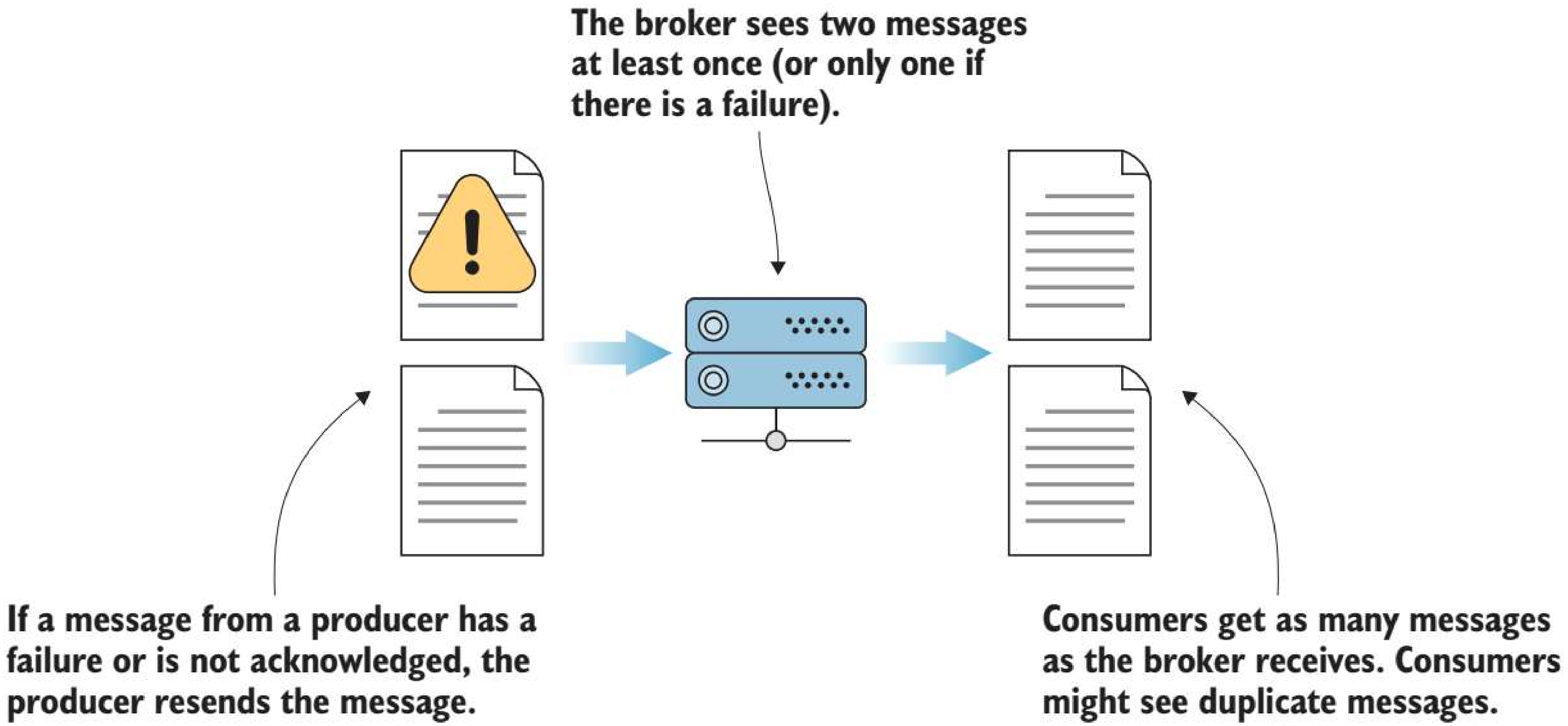
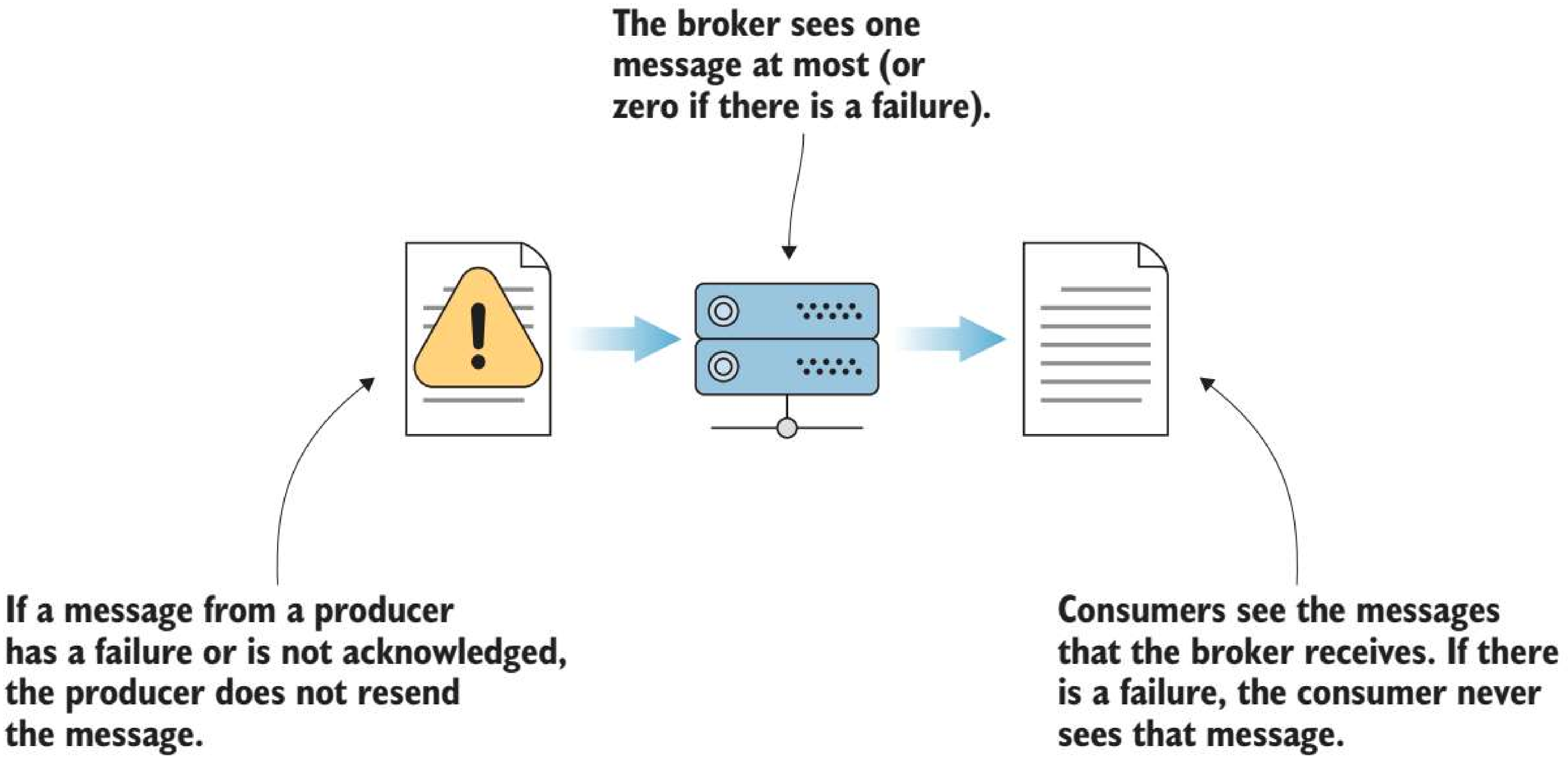
Kafka

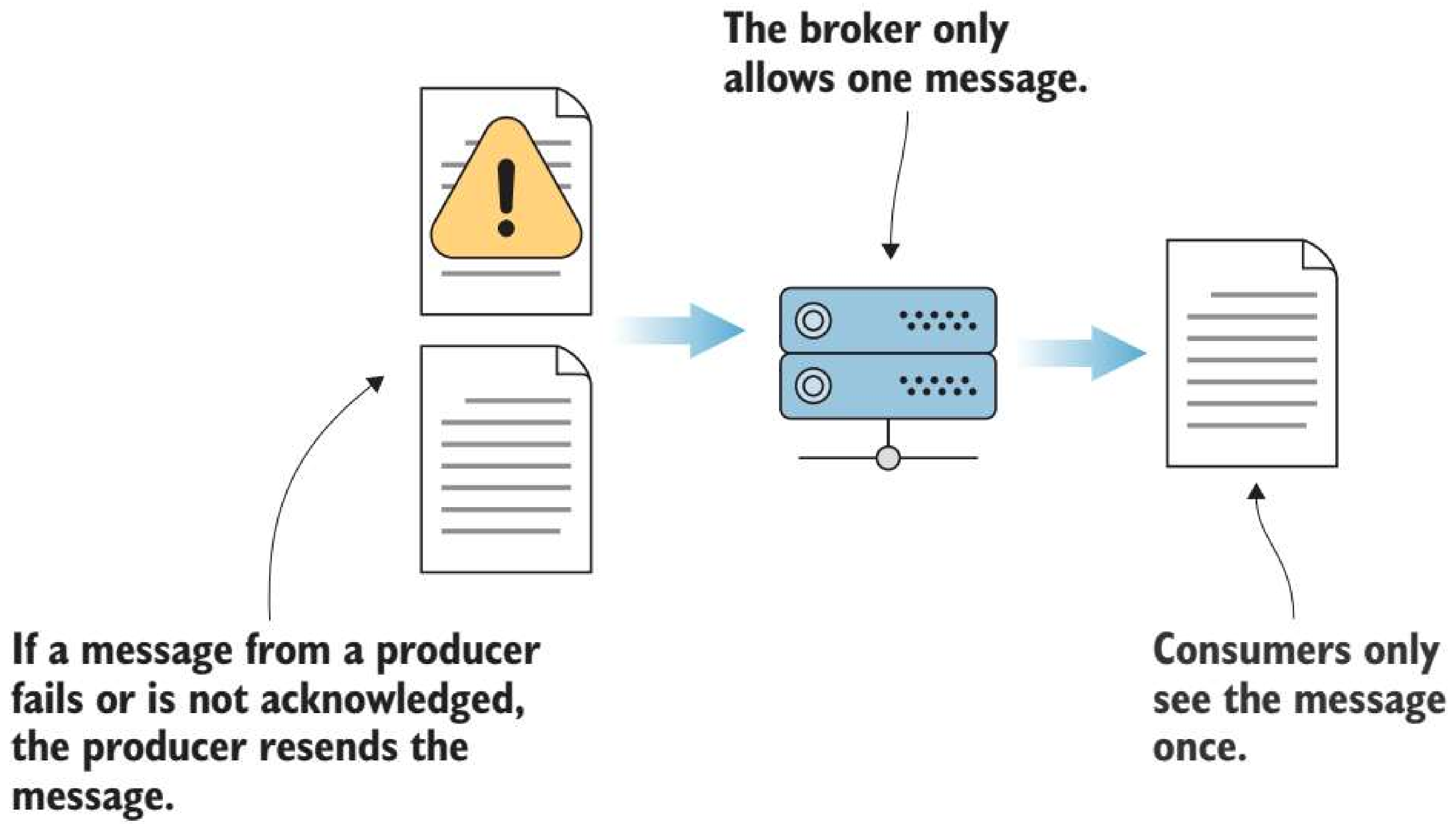
Concepts in Depth


The broker sees two messages at least once (or only one if there is a failure).
If a message from a producer has a failure or is not acknowledged, the producer resends the message.
Consumers get as many messages as the broker receives. Consumers might see duplicate messages.

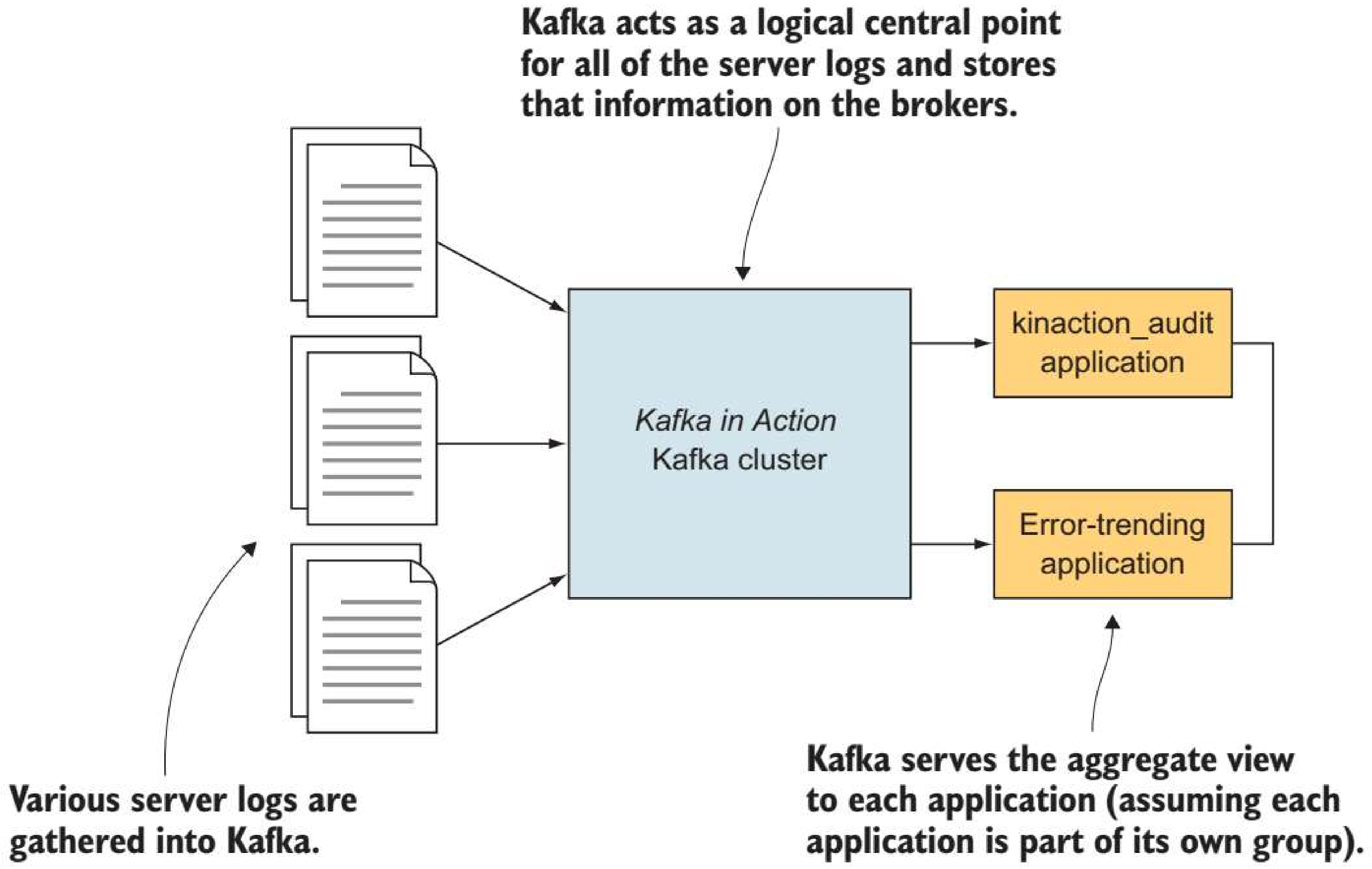
At-least-once message flow


The broker sees one message at most (or zero if there is a failure).
If a message from a producer has a failure or is not acknowledged, the producer does not resend the message.
Consumers see the messages that the broker receives. If there is a failure, the consumer never sees that message.

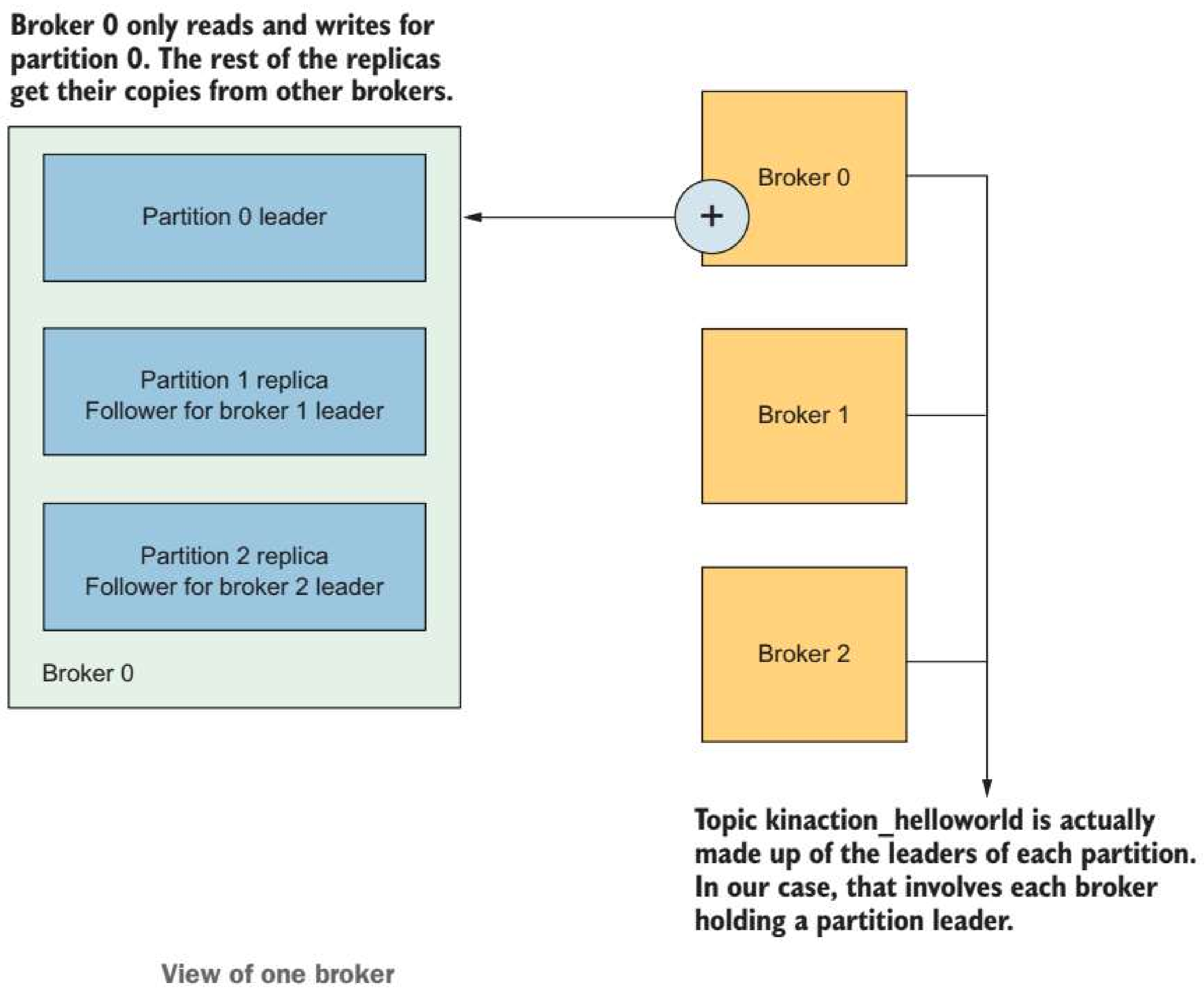
At-most-once message flow

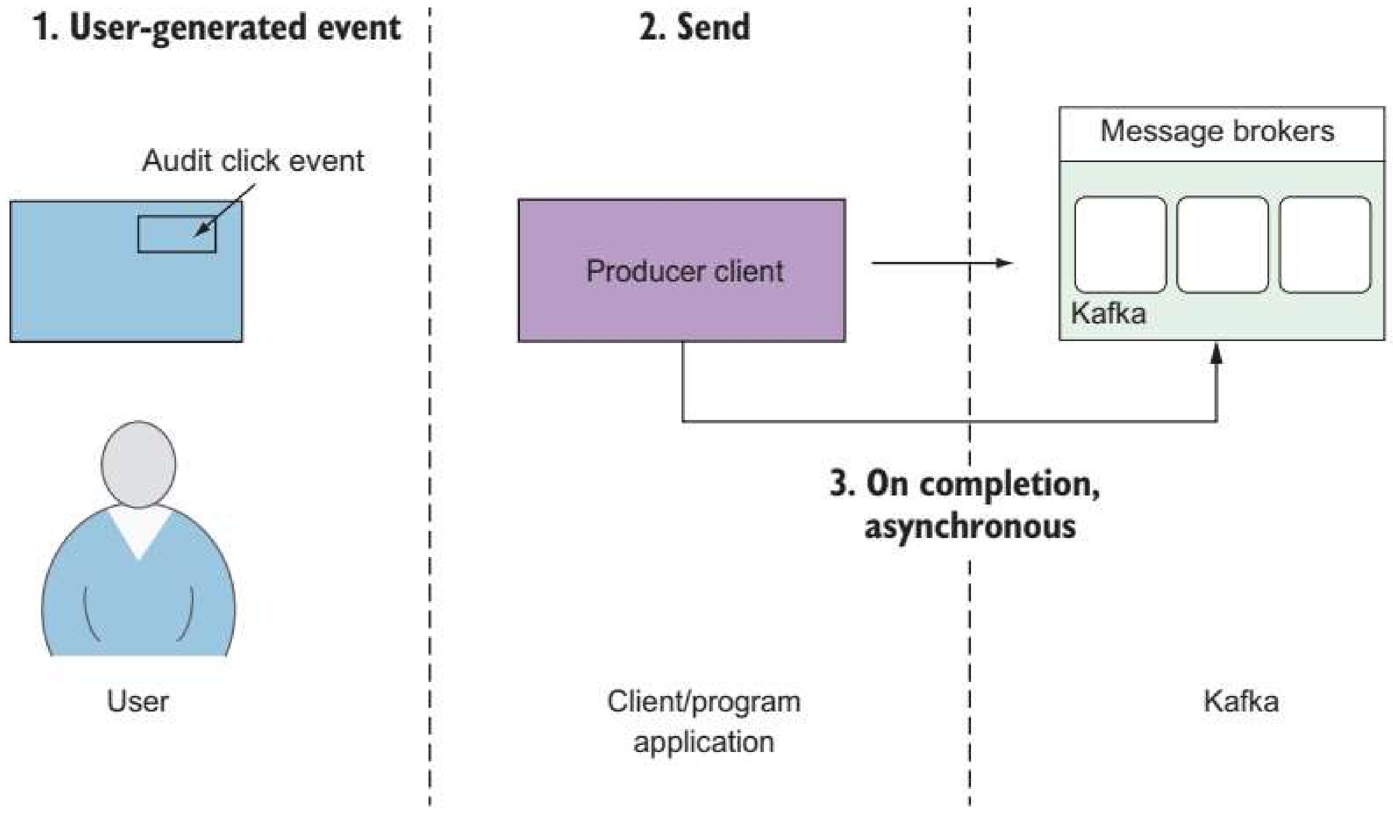

The broker only allows one message.
If a message from a producer fails or is not acknowledged, the producer resends the message.
Consumers only see the message once.

Exactly-once message flow

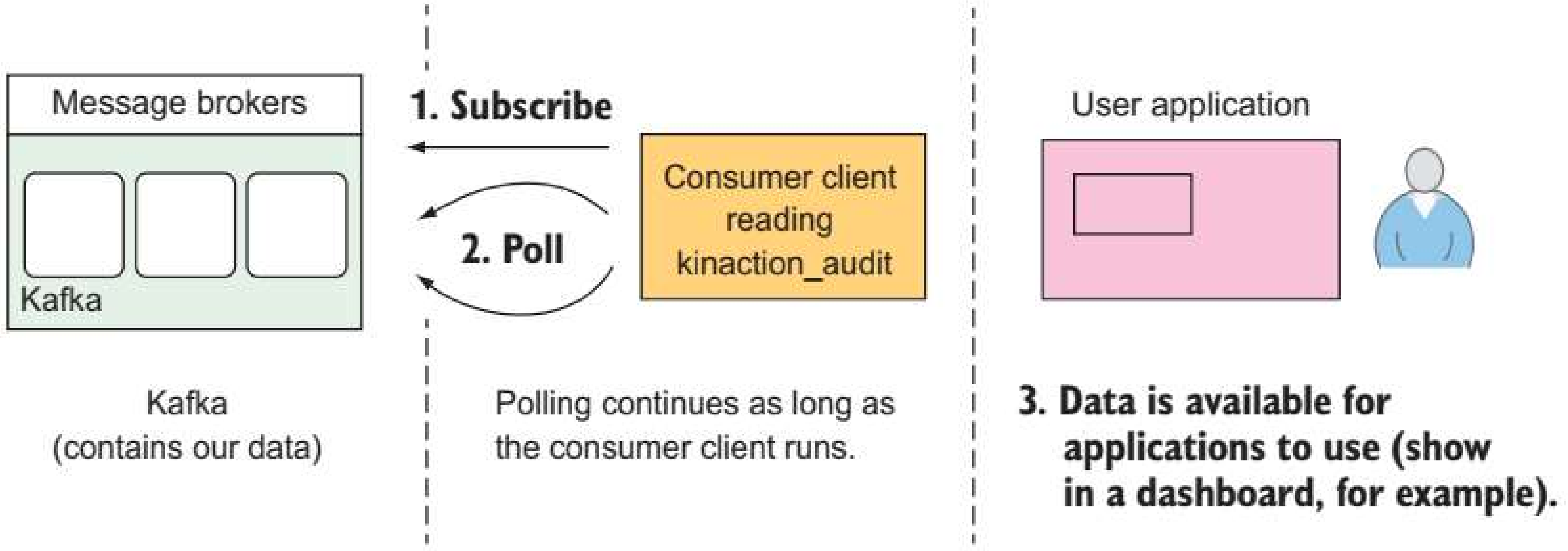

Kafka acts as a logical central point for all of the server logs and stores that information on the brokers.
Kafka in Action Kafka cluster
kinaction_audit application
Error-trending application
Various server logs are gathered into Kafka.
Kafka serves the aggregate view to each application (assuming each application is part of its own group).

Kafka log aggregation

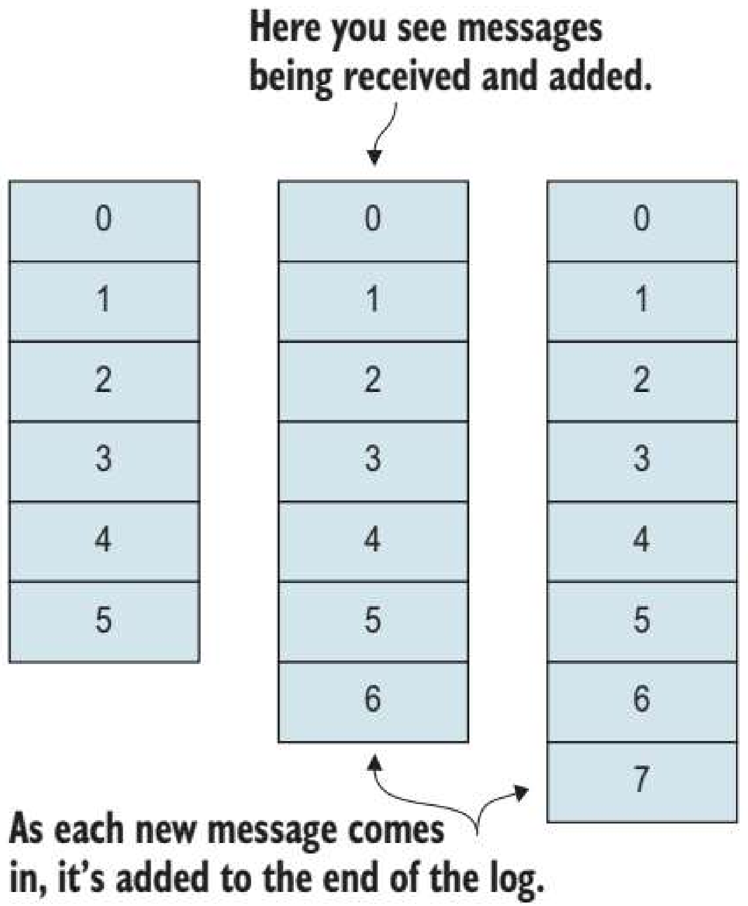

Broker 0 only reads and writes for partition 0. The rest of the replicas get their copies from other brokers.
Partition 0 leader
Partition 1 replica
Follower for broker 1 leader
Partition 2 replica
Follower for broker 2 leader
Broker 0
+
Broker 0
Broker 1
Broker 2
Topic kinaction_helloworld is actually made up of the leaders of each partition. In our case, that involves each broker holding a partition leader.
View of one broker



Producer example for user event

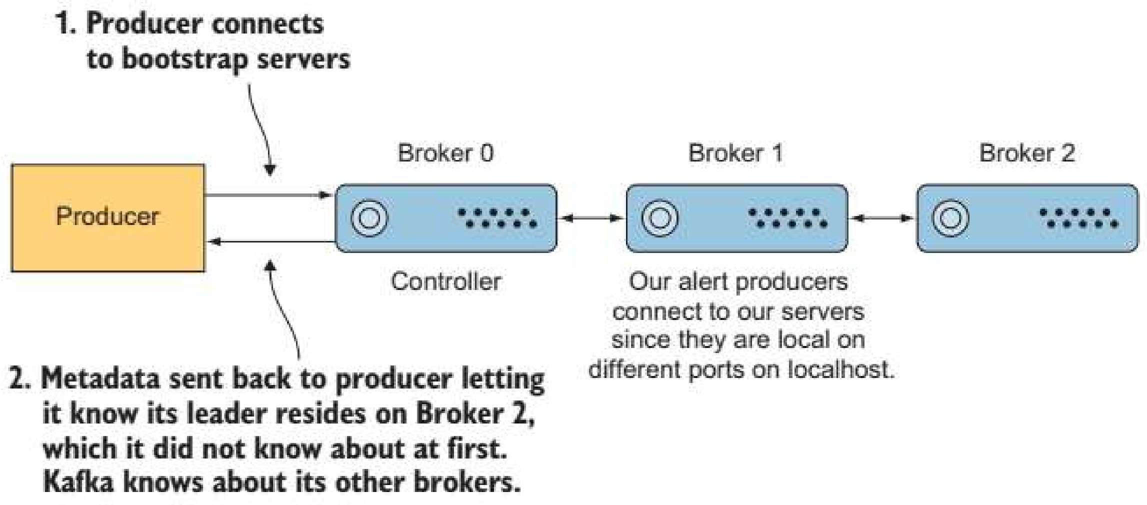

Message brokers
Kafka
1. Subscribe
2. Poll
Consumer client reading kinaction_audit
User application
Kafka (contains our data)
Polling continues as long as the consumer client runs.
3. Data is available for applications to use (show in a dashboard, for example).

Consumer example flow

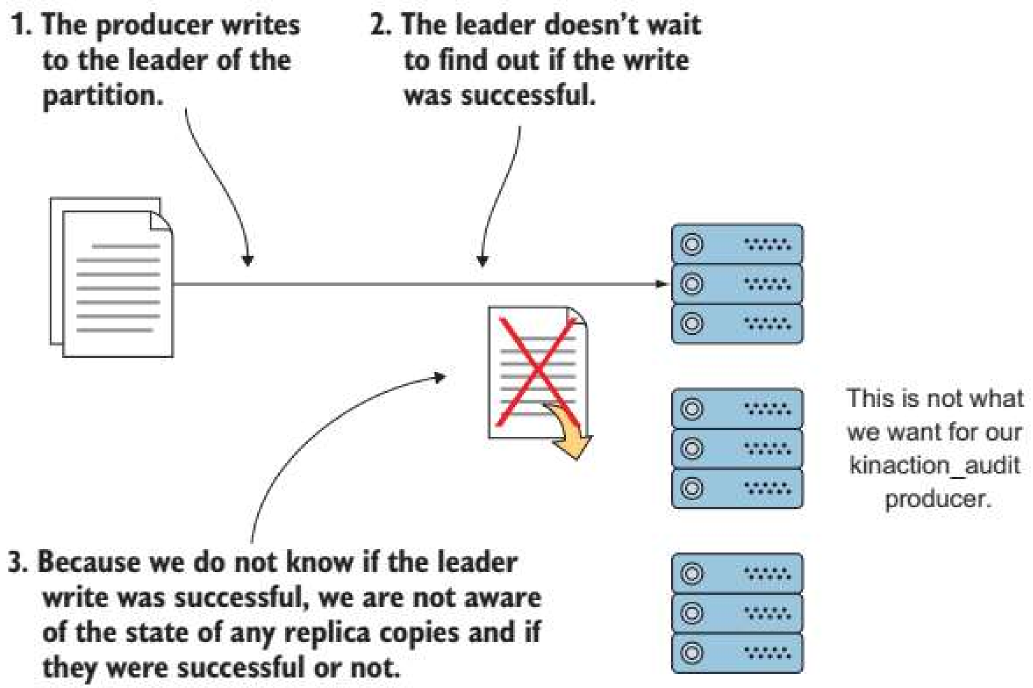

Here you see messages being received and added.
0
1
2
3
4
5
0
1
2
3
4
5
6
0
1
2
3
4
5
6
7
As each new message comes in, it's added to the end of the log.

Important producer configurations

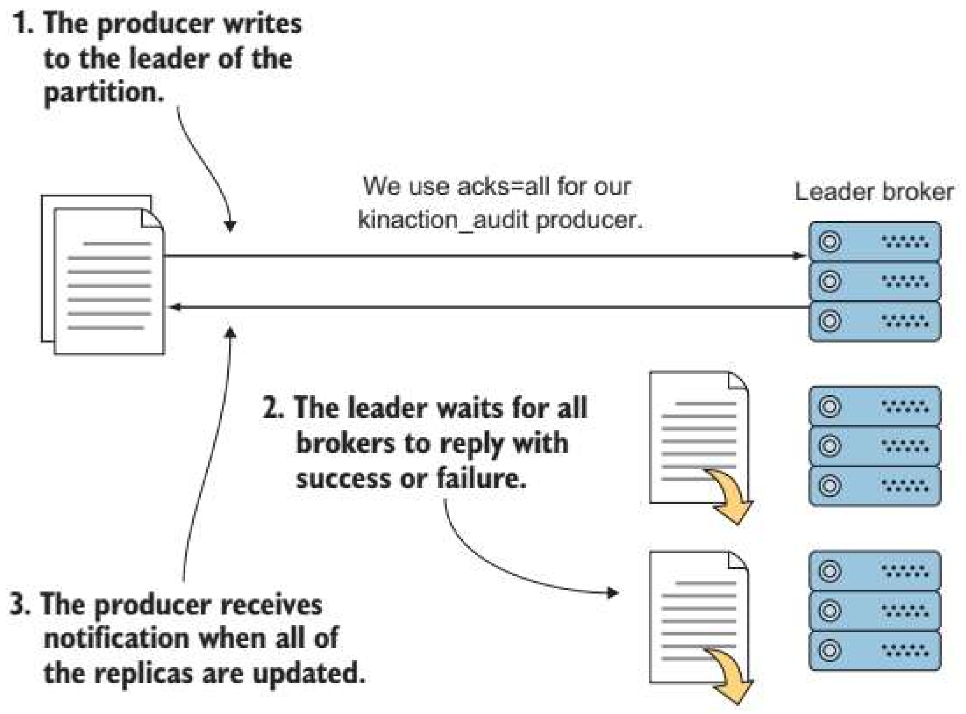
Key	Purpose
`acks`	Number of replica acknowledgments that a producer requires before success is established
`bootstrap.servers`	One or more Kafka brokers to connect for startup
`value.serializer`	The class that's used for serialization of the value
`key.serializer`	The class that's used for serialization of the key


1. Producer connects to bootstrap servers
Broker 0
Broker 1
Broker 2
Producer
Controller
Our alert producers connect to our servers since they are local on different ports on localhost.
2. Metadata sent back to producer letting it know its leader resides on Broker 2, which it did not know about at first. Kafka knows about its other brokers.

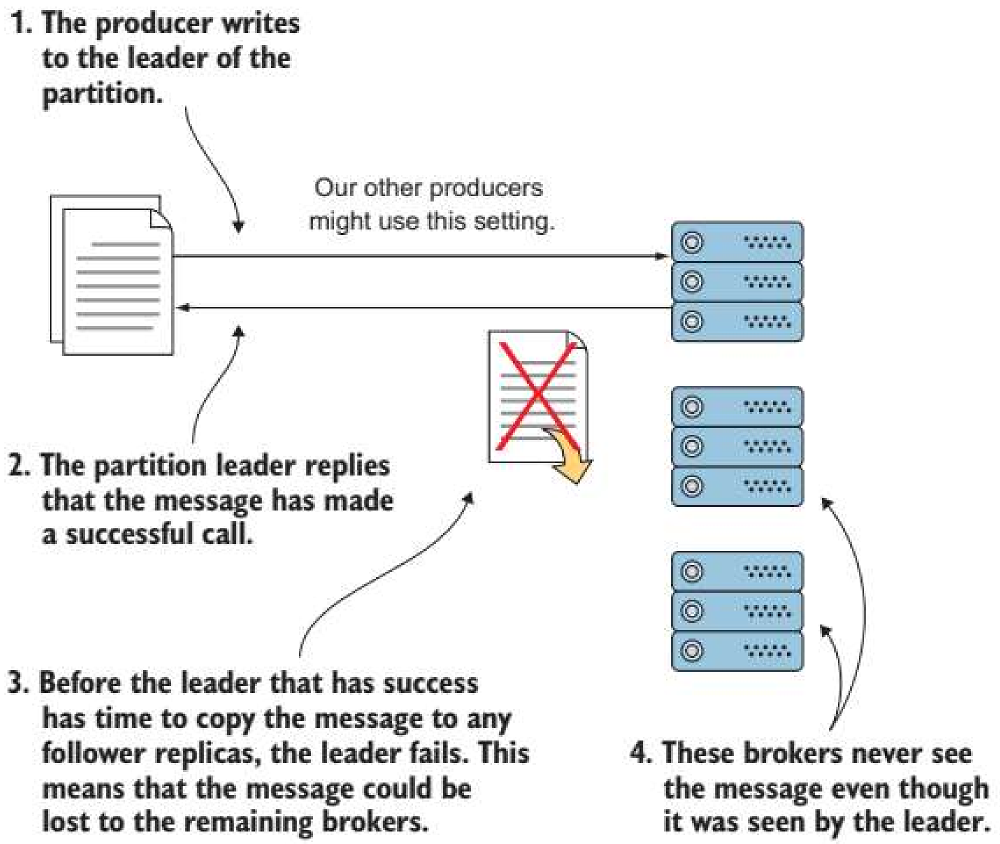
Bootstrap servers


1. The producer writes to the leader of the partition.
2. The leader doesn't wait to find out if the write was successful.
3. Because we do not know if the leader write was successful, we are not aware of the state of any replica copies and if they were successful or not.
This is not what we want for our kinaction_audit producer.

The property `acks` equals `0`.


1. The producer writes to the leader of the partition.
We use acks=all for our kinaction_audit producer.
Leader broker
2. The leader waits for all brokers to reply with success or failure.
3. The producer receives notification when all of the replicas are updated.

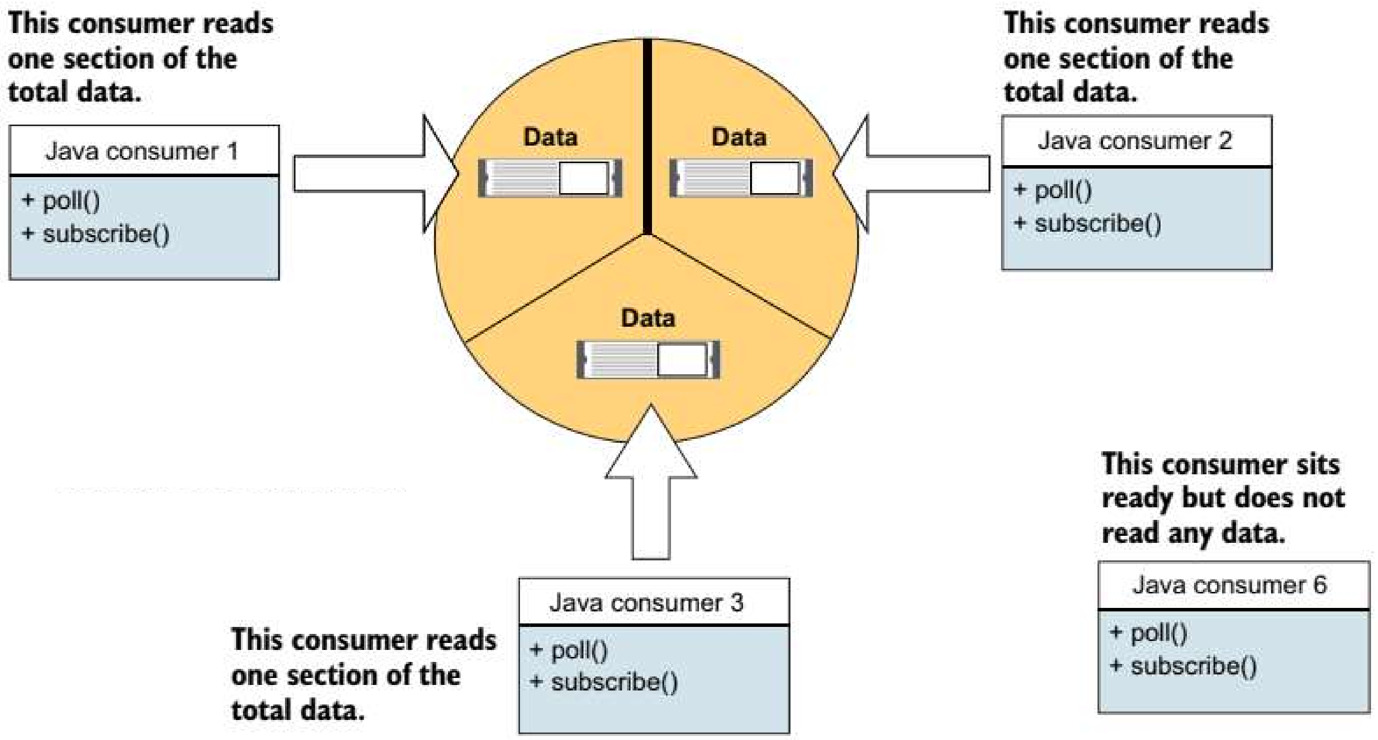
The property `acks` equals `all`.


1. The producer writes to the leader of the partition.
Our other producers might use this setting.
2. The partition leader replies that the message has made a successful call.
3. Before the leader that has success has time to copy the message to any follower replicas, the leader fails. This means that the message could be lost to the remaining brokers.
4. These brokers never see the message even though it was seen by the leader.

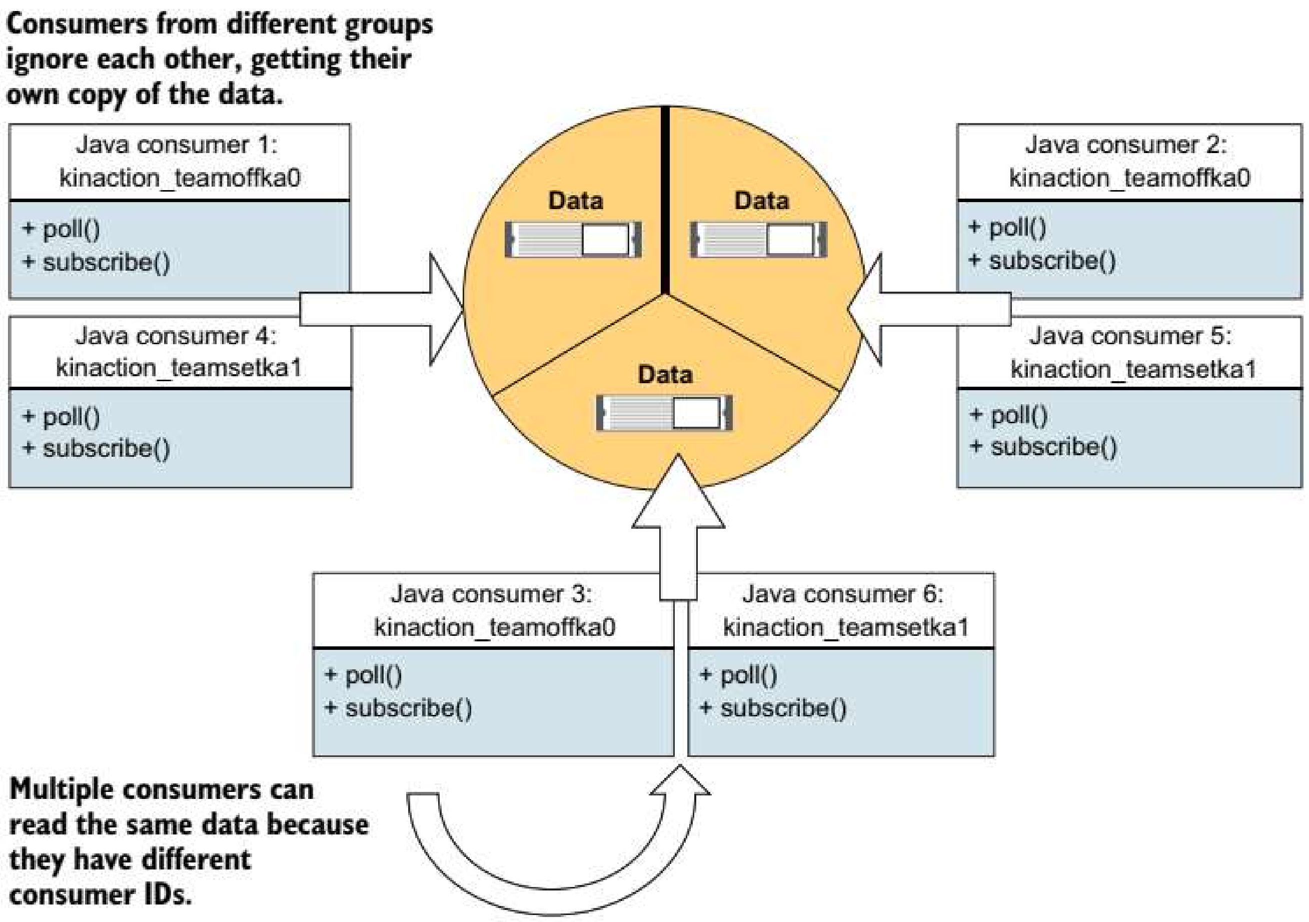
The property `acks` equals `1`.

Consumer configuration

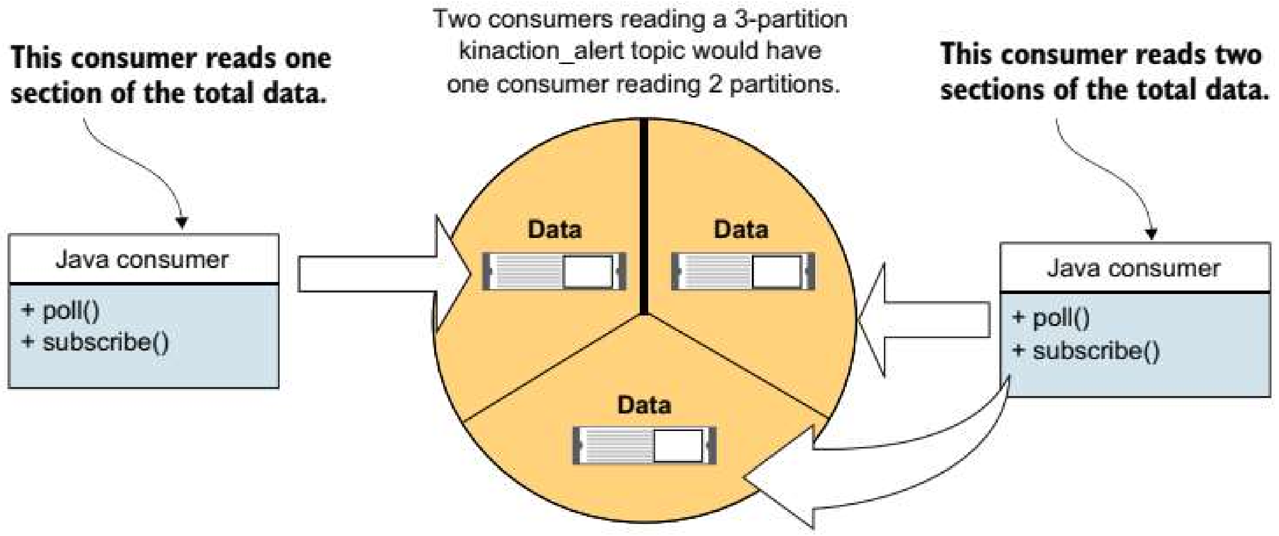
Key	Purpose
`bootstrap.servers`	One or more Kafka brokers to connect on startup
`value.deserializer`	Needed for deserialization of the value
`key.deserializer`	Needed for deserialization of the key
`group.id`	A name that's used to join a consumer group
`client.id`	An ID to identify a user
`heartbeat.interval.ms`	Interval for consumer's pings to the group coordinator


This consumer reads one section of the total data.
Java consumer 1
+ poll()
+ subscribe()
Data
Data
Data
This consumer reads one section of the total data.
Java consumer 2
+ poll()
+ subscribe()
This consumer sits ready but does not read any data.
Java consumer 6
+ poll()
+ subscribe()
This consumer reads one section of the total data.
Java consumer 3
+ poll()
+ subscribe()

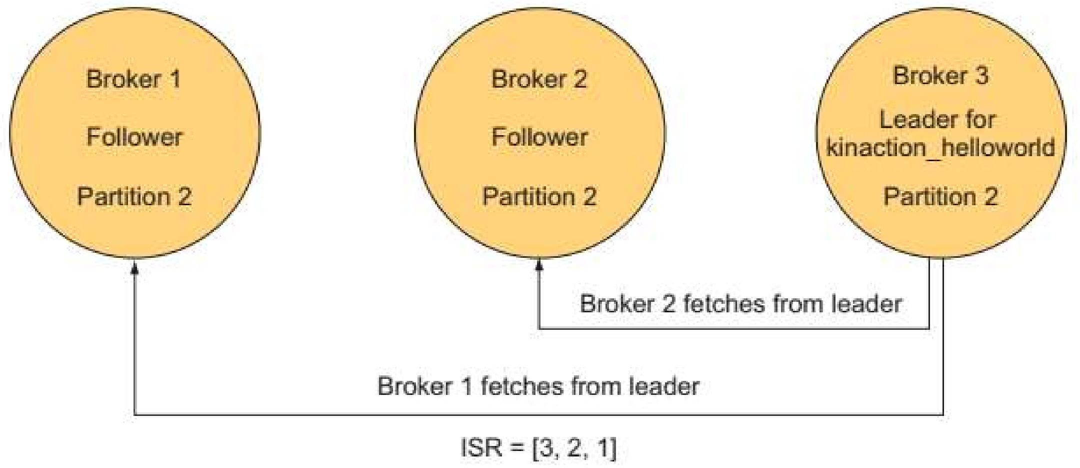
An extra Kafka consumer


Consumers from different groups ignore each other, getting their own copy of the data.
Java consumer 1: kinaction_teamoffka0
+ poll()
+ subscribe()
Java consumer 4: kinaction_teamsetka1
+ poll()
+ subscribe()
Data
Data
Data
Java consumer 2: kinaction_teamoffka0
+ poll()
+ subscribe()
Java consumer 5: kinaction_teamsetka1
+ poll()
+ subscribe()
Java consumer 3: kinaction_teamoffka0
+ poll()
+ subscribe()
Java consumer 6: kinaction_teamsetka1
+ poll()
+ subscribe()
Multiple consumers can read the same data because they have different consumer IDs.

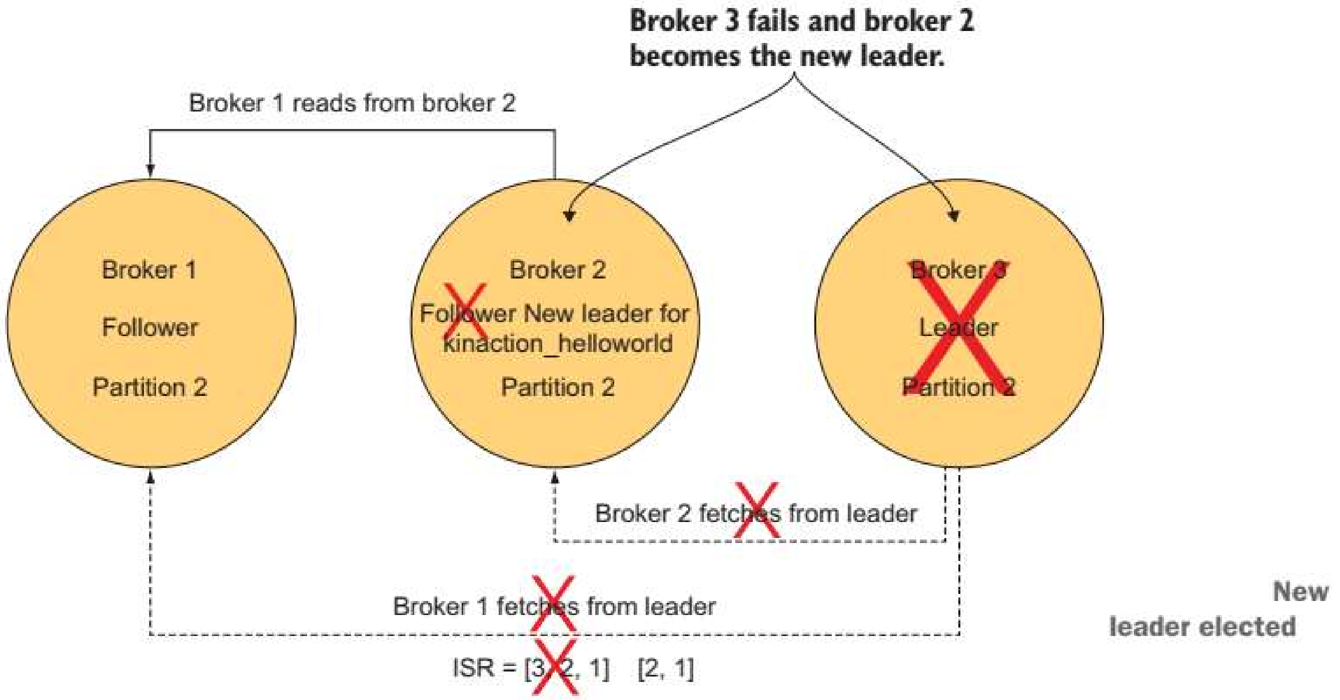
Consumers in separate groups [12]


This consumer reads one section of the total data.
Two consumers reading a 3-partition kinaction_alert topic would have one consumer reading 2 partitions.
This consumer reads two sections of the total data.
Java consumer
+ poll()
+ subscribe()
Data
Data
Data
Java consumer
+ poll()
+ subscribe()

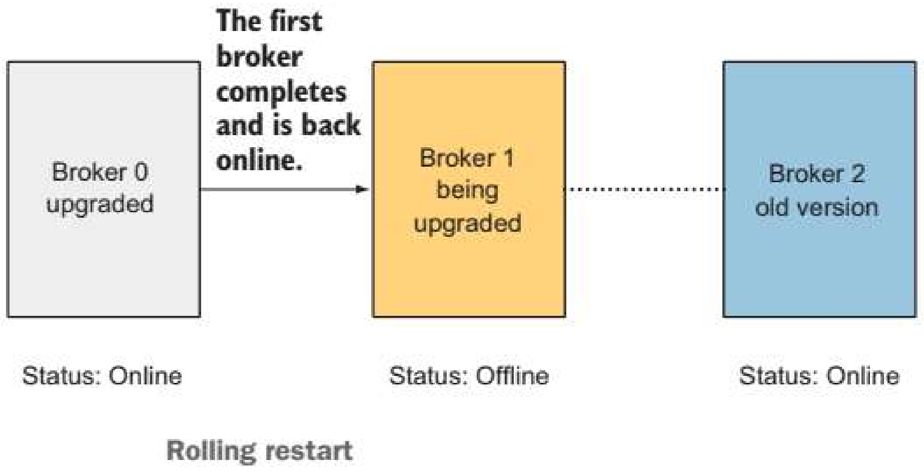
Kafka consumers in a group

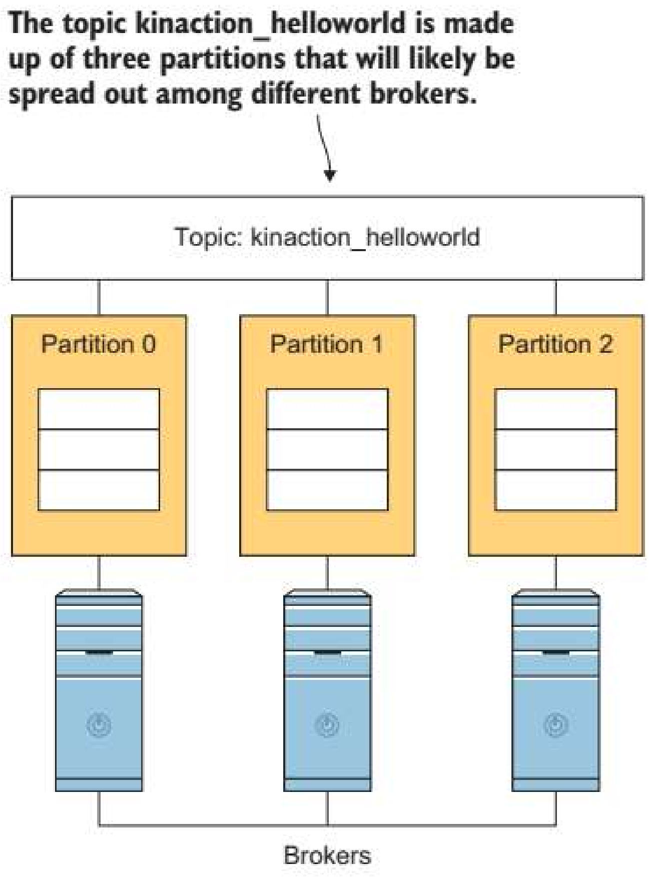

Broker 1
Follower
Partition 2
Broker 2
Follower
Partition 2
Broker 3
Leader for kinaction_helloworld
Partition 2
Broker 2 fetches from leader
Broker 1 fetches from leader
ISR = [3, 2, 1]

Leader



Broker 3 fails and broker 2 becomes the new leader.
Broker 1 reads from broker 2
Broker 1
Follower
Partition 2
Broker 2
Follower New leader for kinaction_helloworld
Partition 2
Broker 3
Leader
Partition 2
Broker 2 fetches from leader
Broker 1 fetches from leader
ISR = [3, 2, 1] [2, 1]
New leader elected


The first broker completes and is back online.
Broker 0 upgraded
Broker 1 being upgraded
Broker 2 old version
Status: Online
Status: Offline
Status: Online
Rolling restart


The topic kinaction_helloworld is made up of three partitions that will likely be spread out among different brokers.
Topic: kinaction_helloworld
Partition 0
Partition 1
Partition 2
Brokers

Example topic with partitions

Broker retention configuration

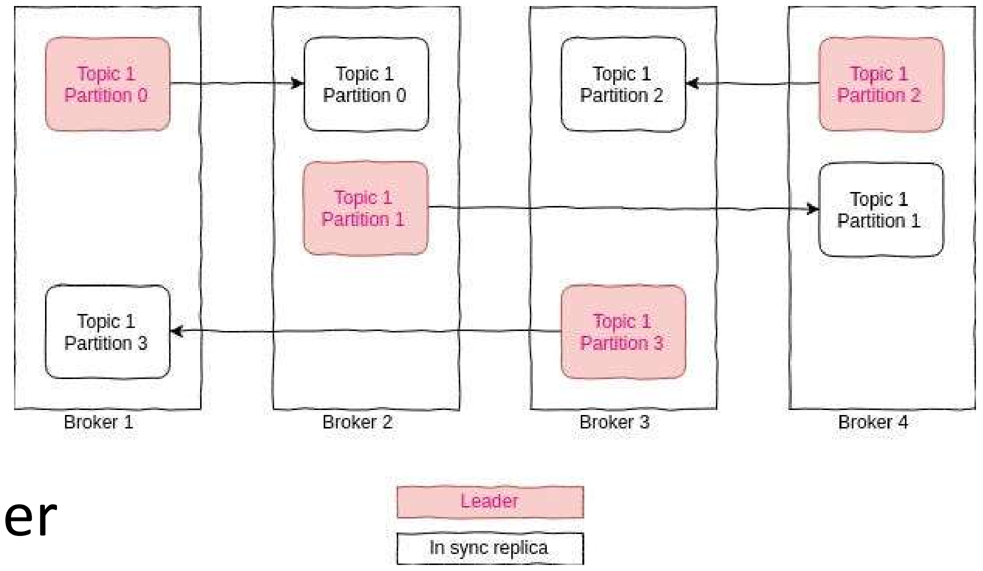
Key	Purpose
`log.retention.bytes`	The largest size threshold in bytes for deleting a log.
`log.retention.ms`	The length in milliseconds a log will be maintained before being deleted.
`log.retention.minutes`	Length before deletion in minutes. `log.retention.ms` is used as well if both are set.
`log.retention.hours`	Length before deletion in hours. `log.retention.ms` and `log.retention.minutes` would be used before this value if either of those are set.

Replication

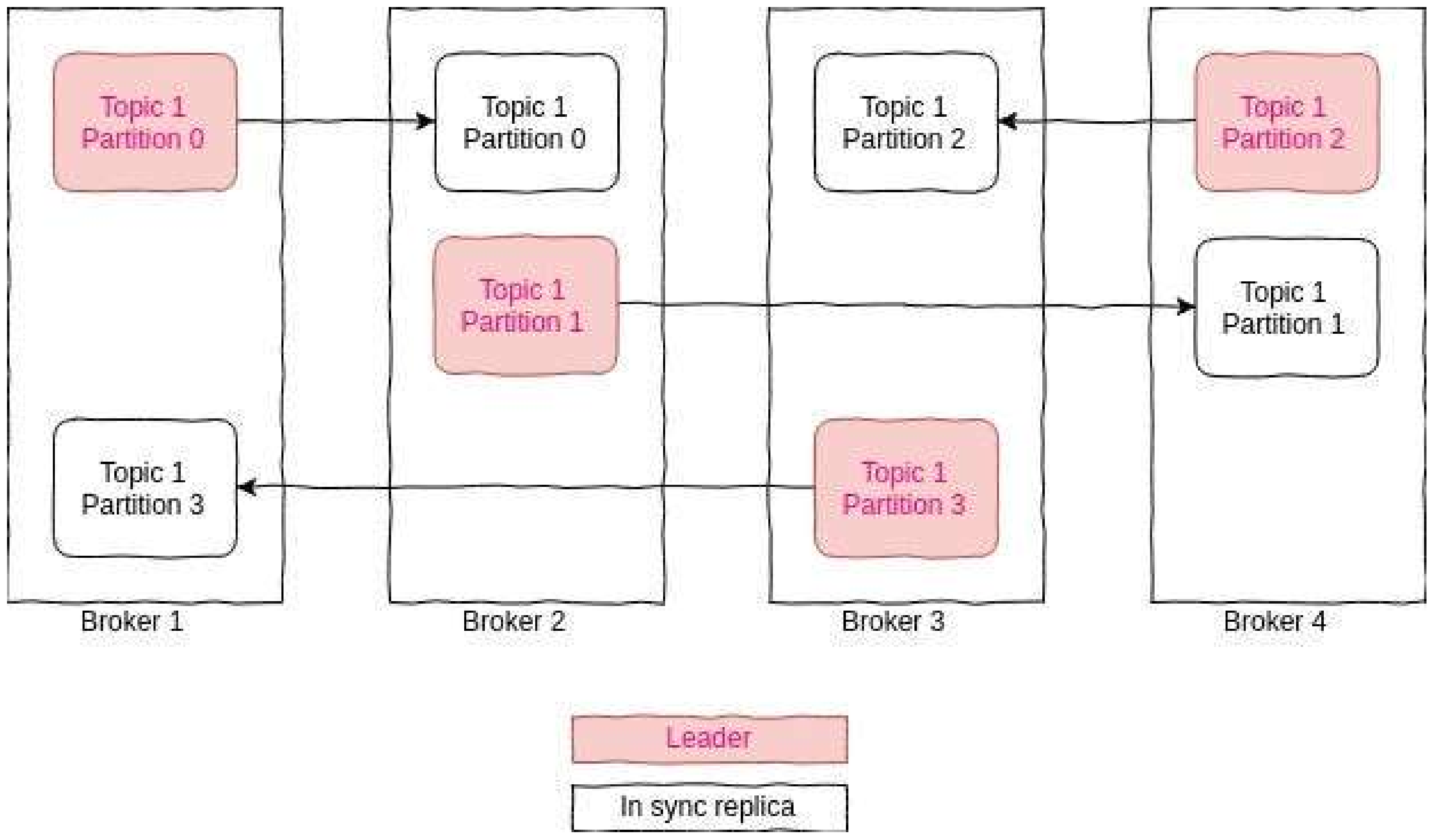
- leader replica
  - All requests are performed through a leader (ensuring consistency)

- follower replica
  - All replicas that are not leaders
  - Only copies messages from the leader

- To determine whether a replica is ISR,
  the replica makes a request to the leader
- replica.lag.time.max.ms -> 10 seconds



Replication



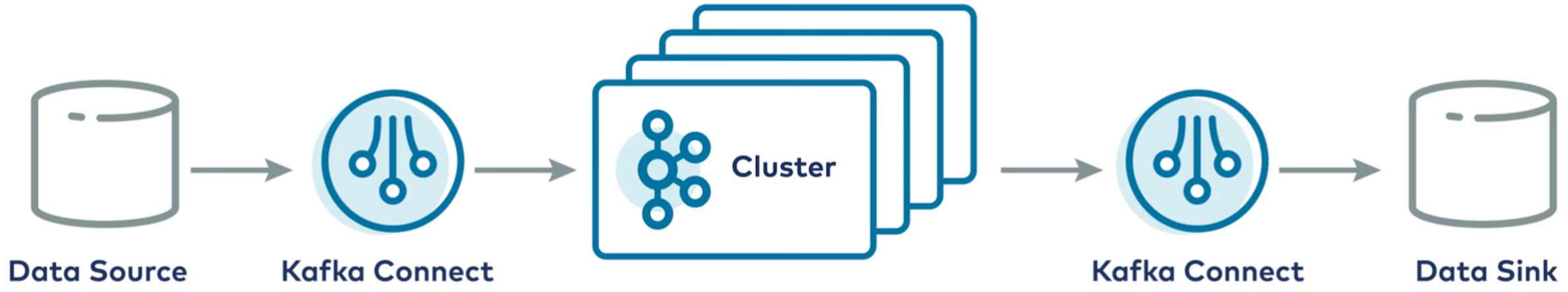
Kafka Connect

When to Use Kafka Connect

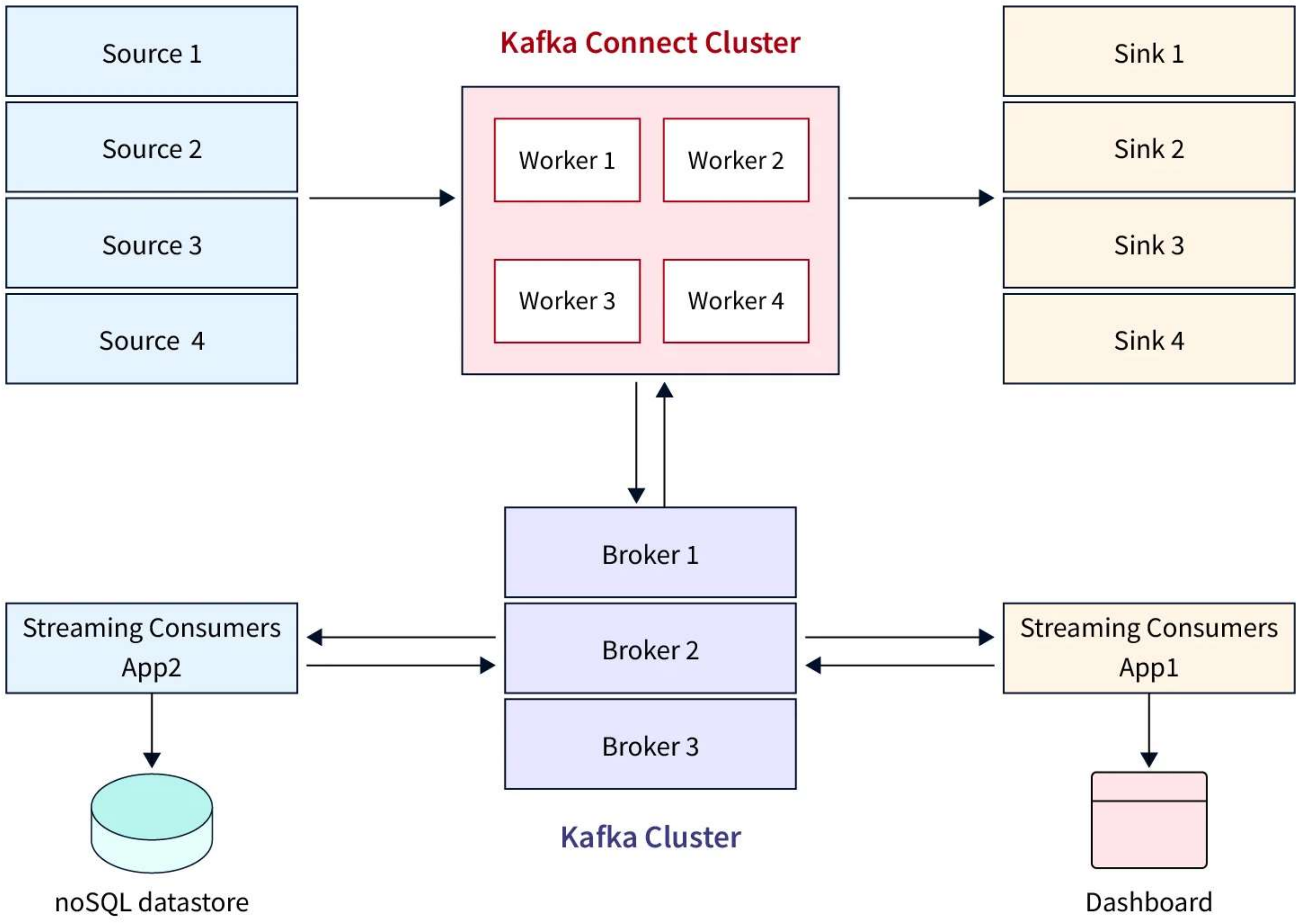
- Use Kafka clients when you can modify the code of the application
- Use Connect to
    - Connect Kafka to datastores whose code you cannot modify.
- Where a connector already exists, Connect can be used by nondevelopers, who will only need to configure the connectors.

Kafka Connect

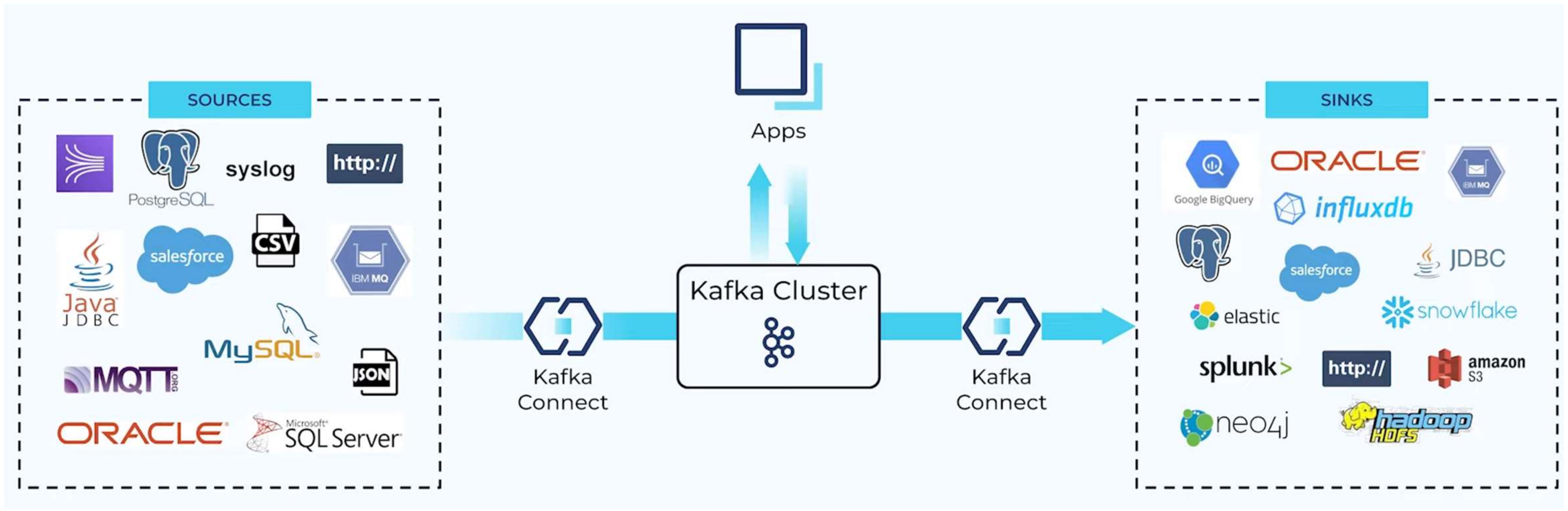
- A framework for connecting Kafka with external systems



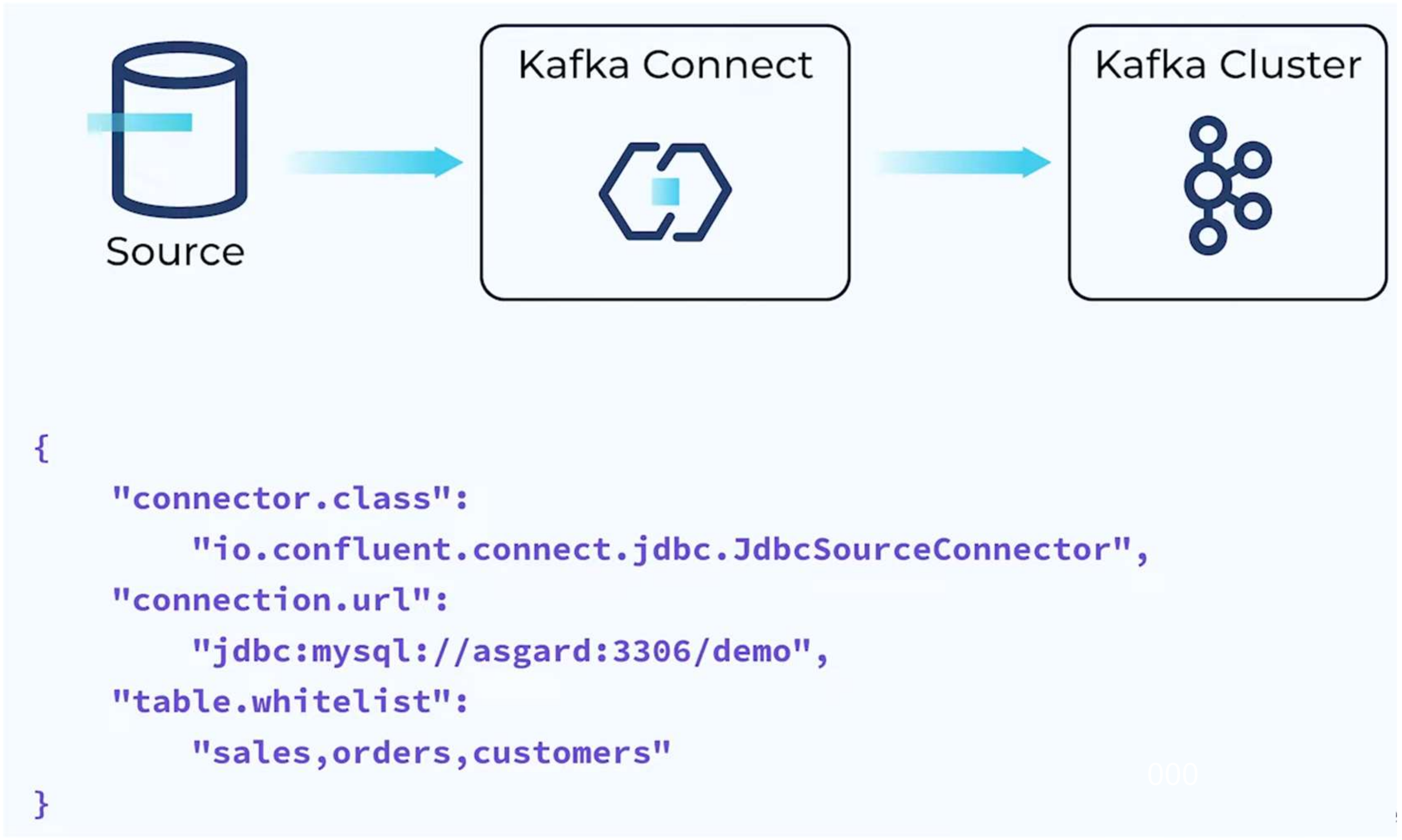
Kafka Connect



Ingest Data from Upstream Systems

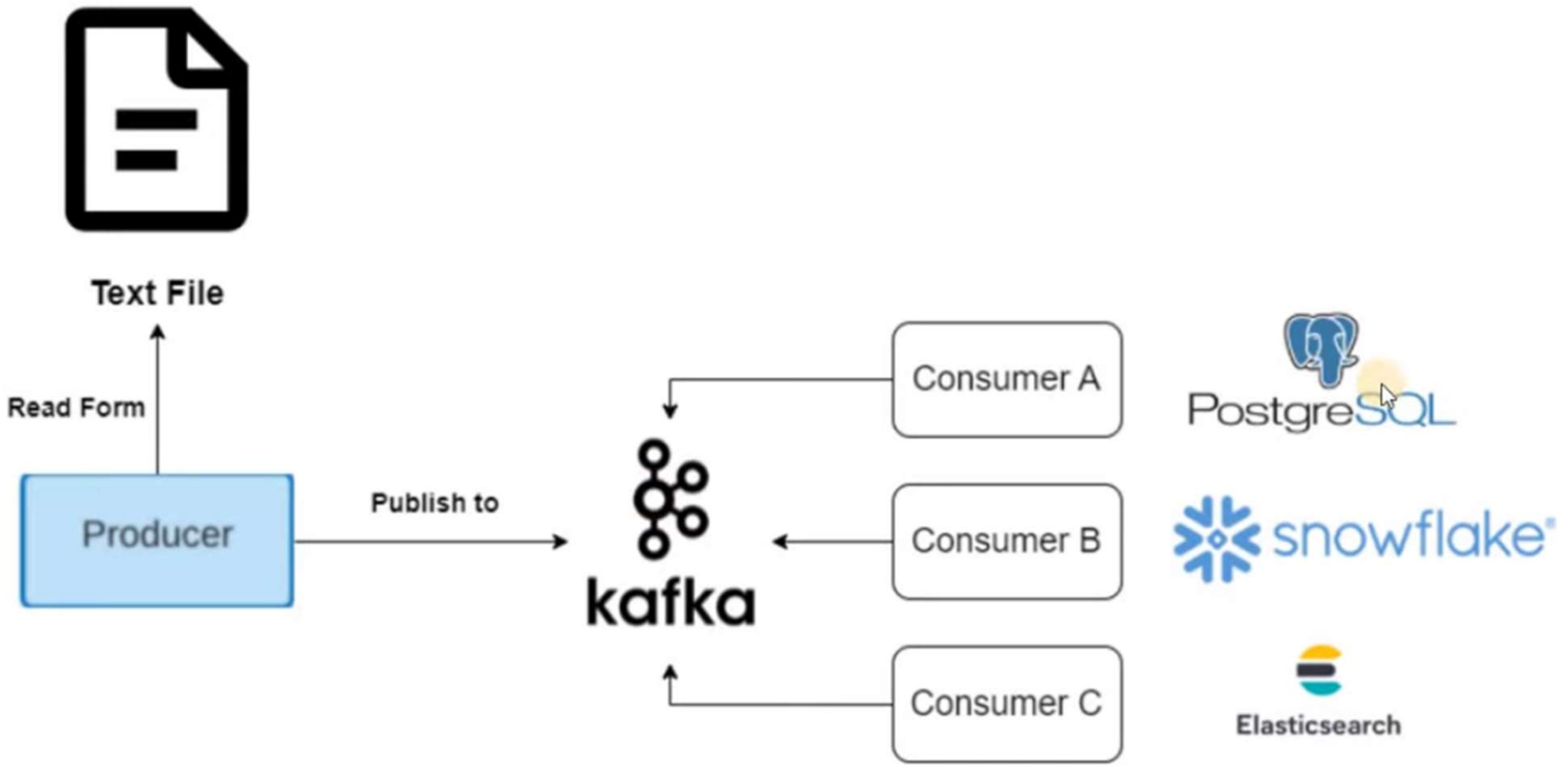


How Kafka Connect Works



```
{
    "connector.class":
        "io.confluent.connect.jdbc.JdbcSourceConnector",
    "connection.url":
        "jdbc:mysql://asgard:3306/demo",
    "table.whitelist":
        "sales,orders,customers"
}
```

Without Connect

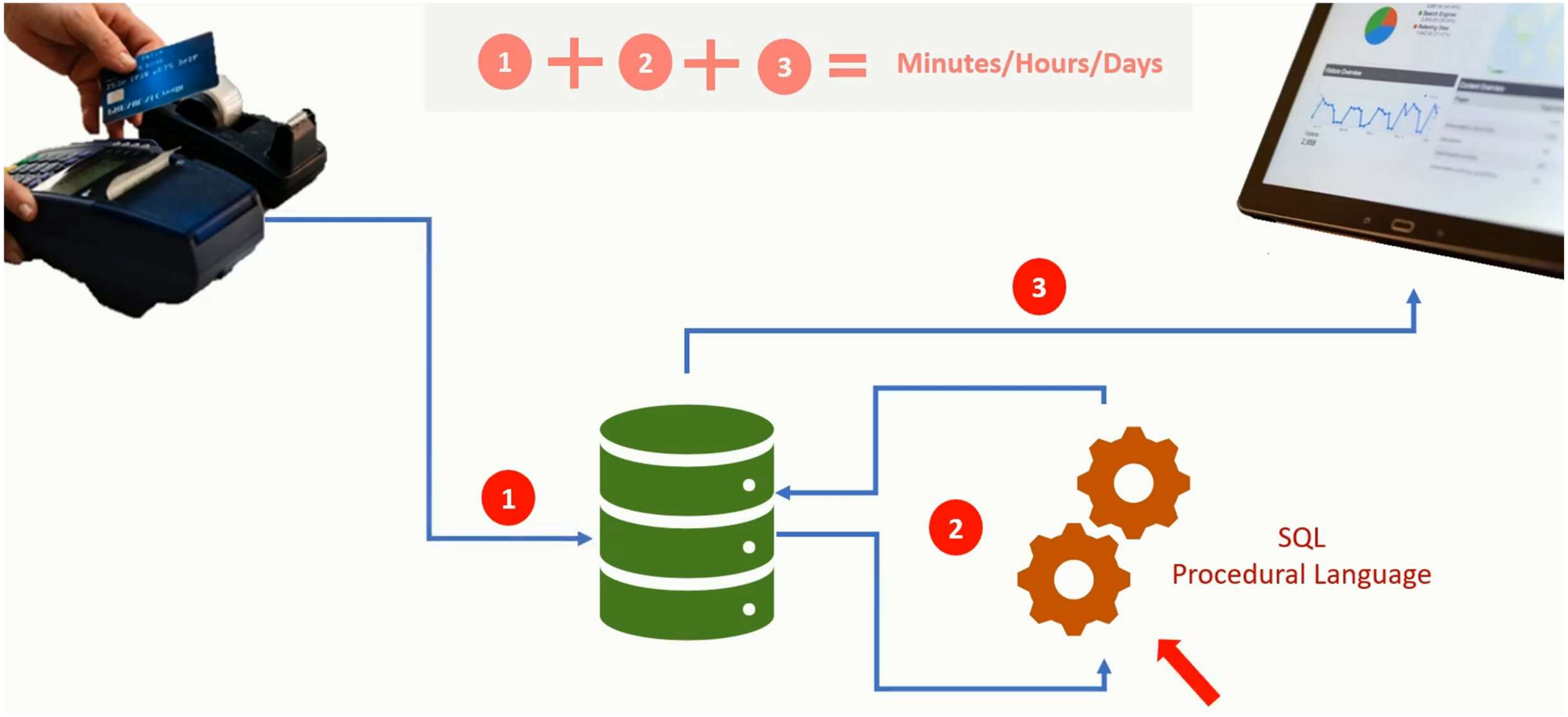


Working few famous connectors

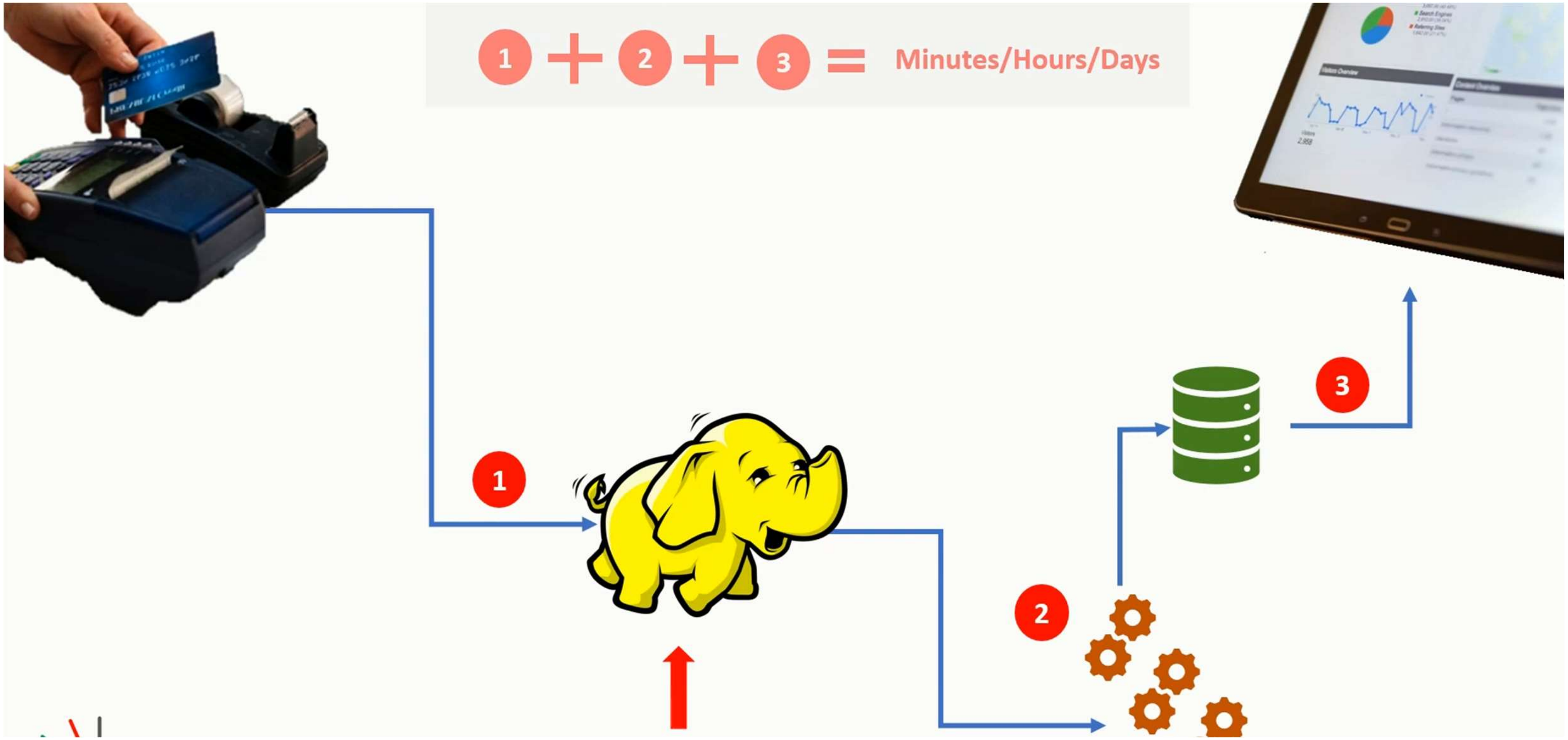
- JDBC Source and Sink Connector
  - Enables a Java application to interact with a Database
- Google BigQuery Sink Connector
  - To stream data into BigQuery Tables
- JMS Source Connector
  - For moving messages from any JMS-compliant broker into a Kafka Topic
- Elasticsearch Service Sink Connector
  - For moving data from a Kafka to Elasticsearch
- Amazon S3 Sink Connector
  - Exports data from Kafka Topics to Amazon S3
- HDFS 2 Sink Connector
  - For exporting data from any Kafka Topic to HDFS 2.x files in a variety of formats
- Replicator
  - Replicate Topics from one Kafka Cluster to another

Kafka Stream Processing

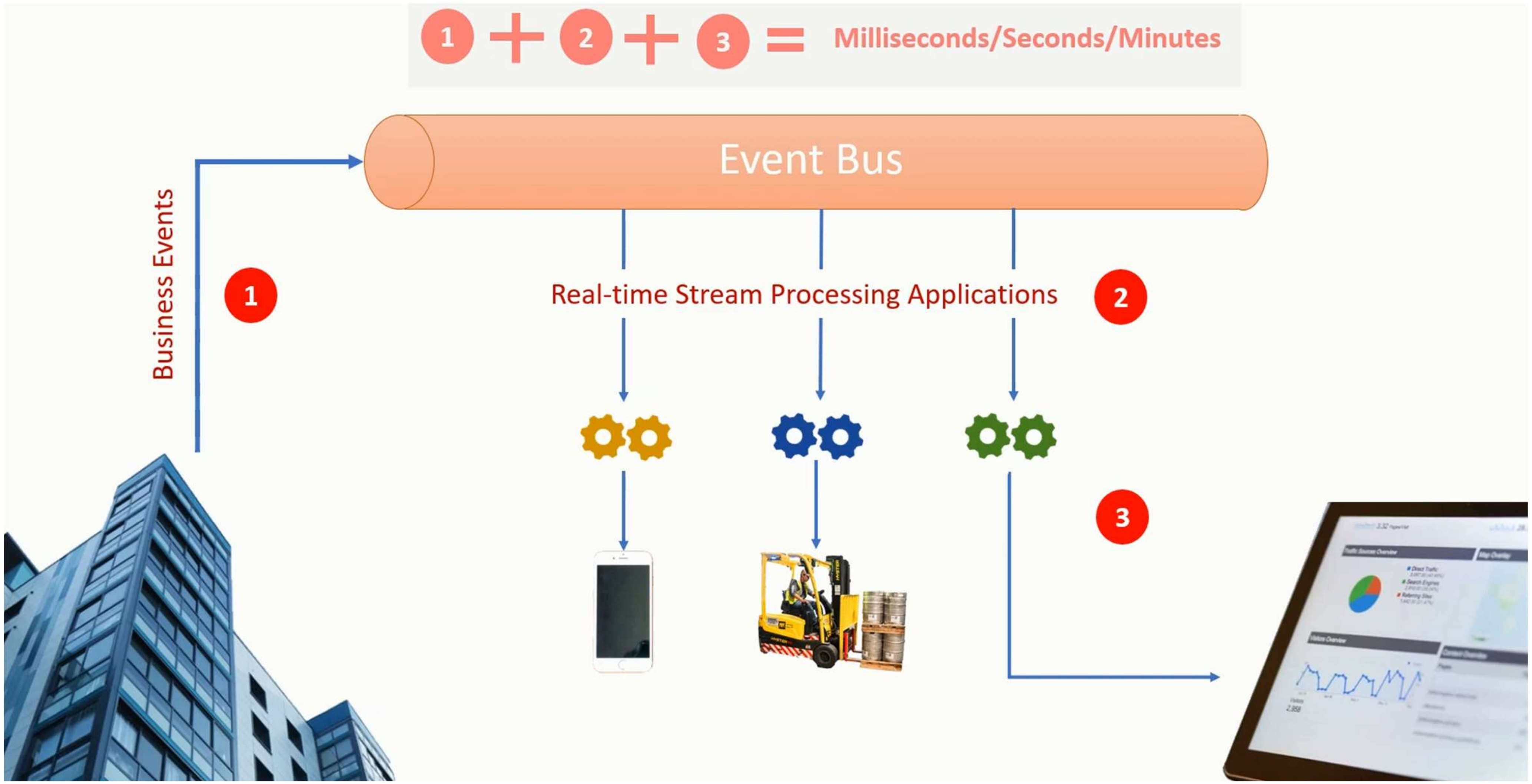
Traditional Data Processing



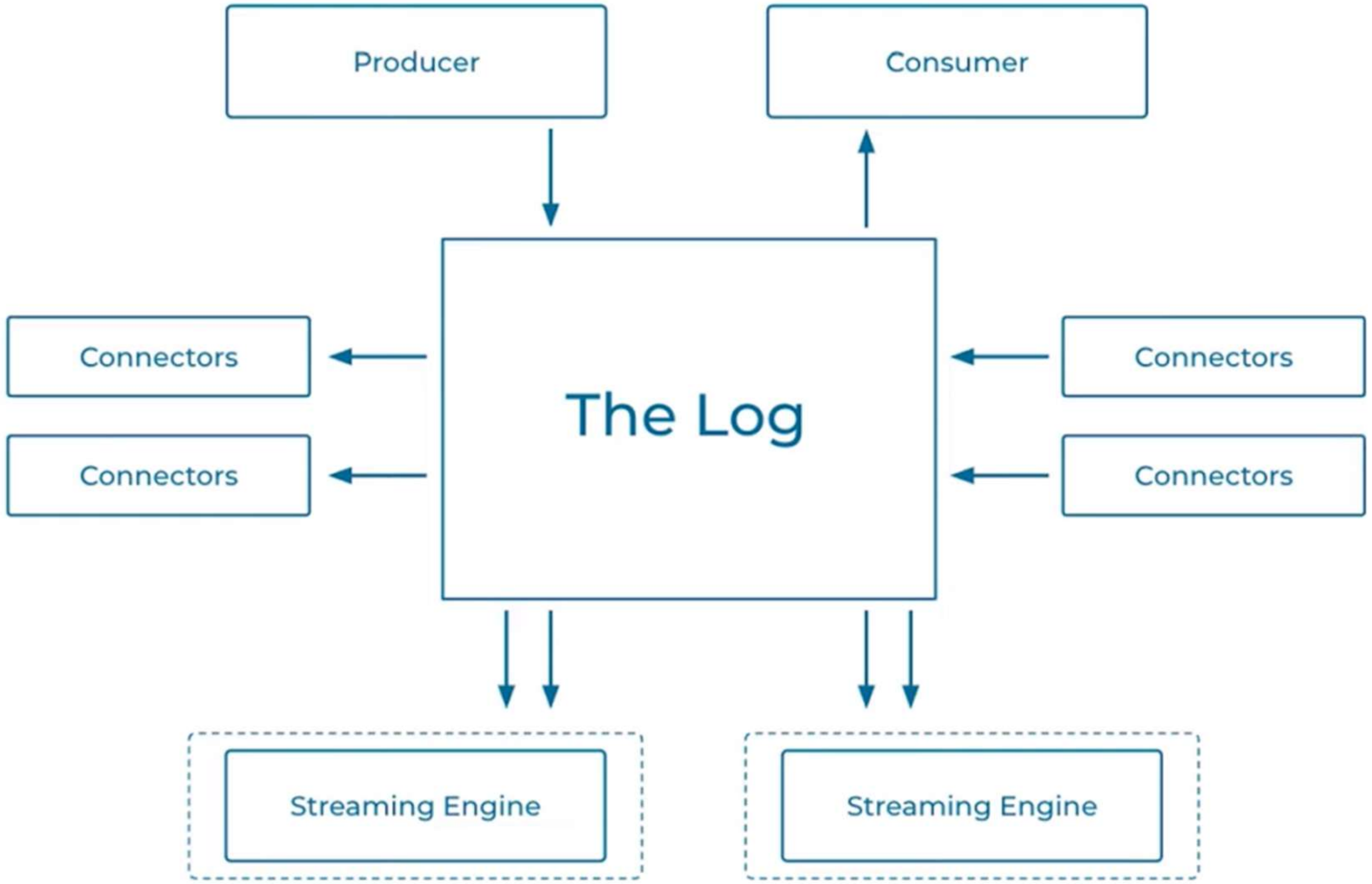
Hadoop for Bigdata processing



Stream Processing


1 + 2 + 3 = Milliseconds/Seconds/Minutes
Event Bus
Business Events
1
Real-time Stream Processing Applications
2
3

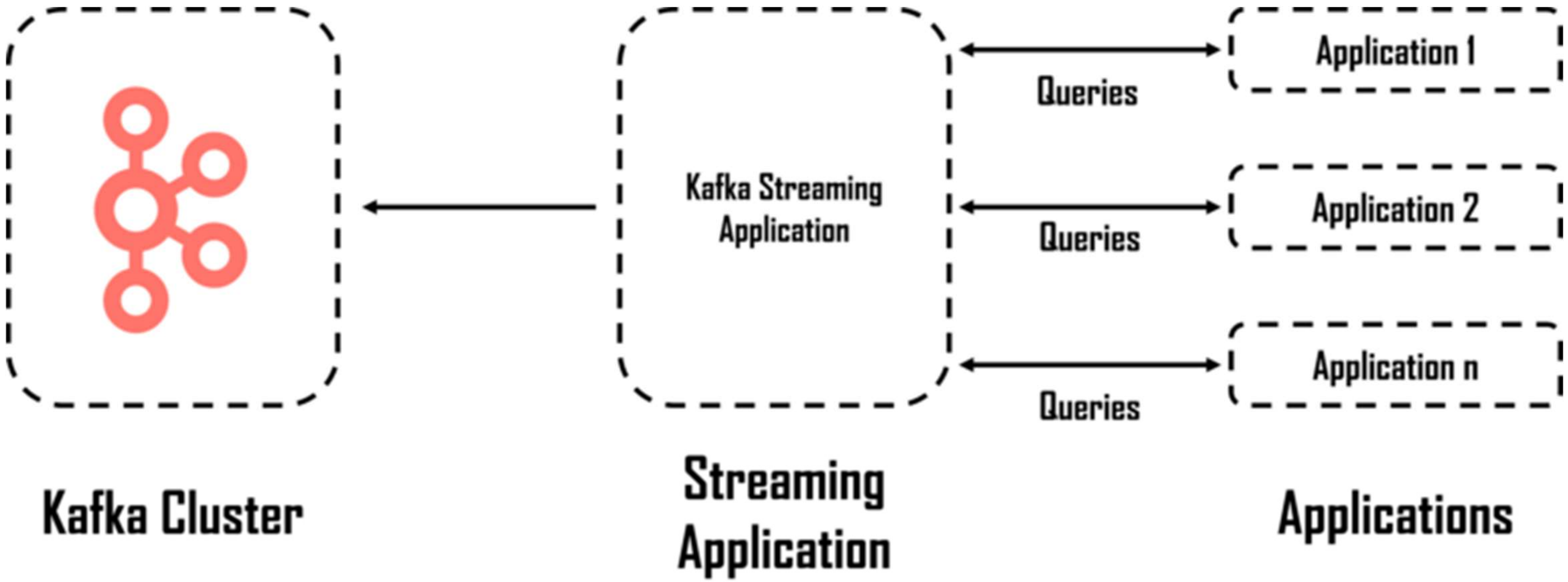
Kafka


Producer
Consumer
Connectors
Connectors
The Log
Connectors
Connectors
Streaming Engine
Streaming Engine

Kafka Streams

- Functional Java API
- Filtering, grouping, aggregating, joining, and more
- Scalable, fault-tolerant state management

Kafka Stream



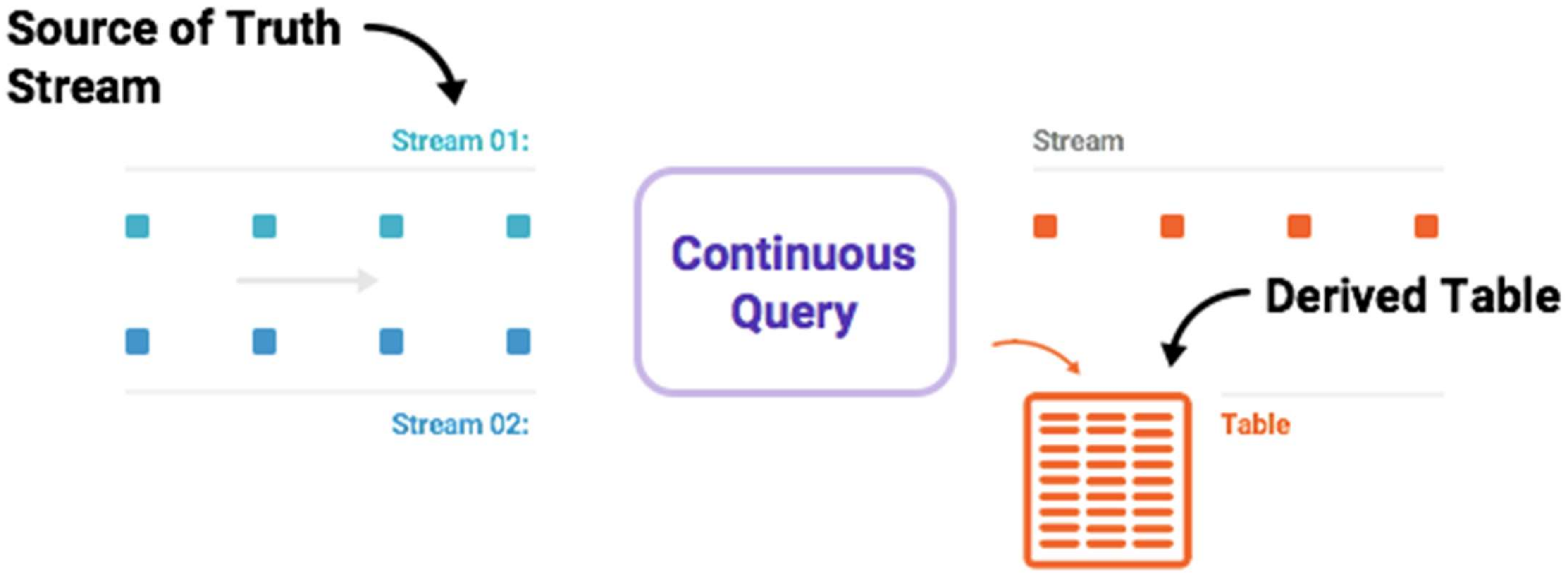
Kafka Structured Streaming

Confluent – KSQL

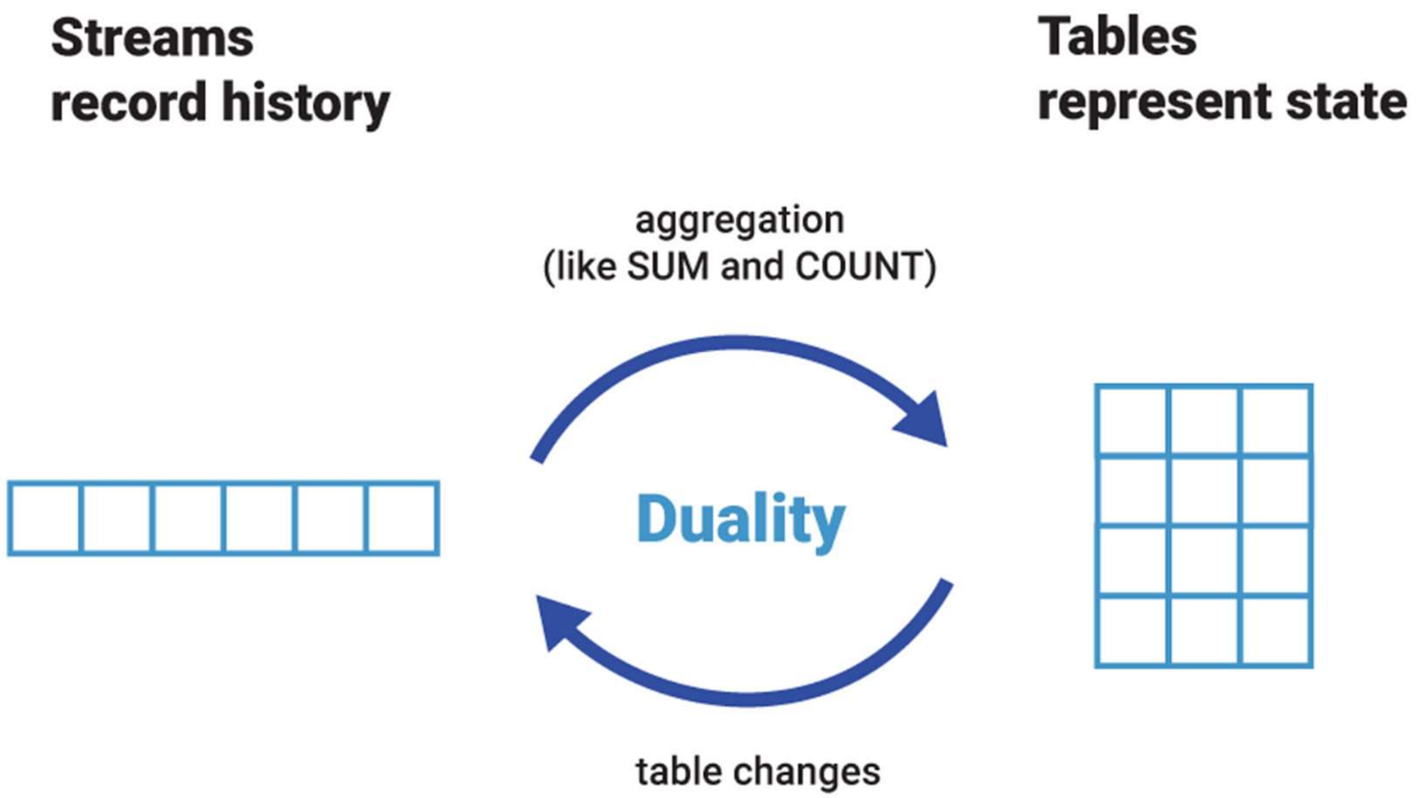
- Streaming SQL for Apache Kafka
- Provides a simple and completely interactive SQL interface for processing data in Kafka
- No longer need to write code
- Supports aggregations, joins, windowing, sessionization, and much more

- Example:
  - CREATE TABLE error_counts AS
  - SELECT error_code, count(*)FROM monitoring_stream
  - WINDOW TUMBLING (SIZE 1 MINUTE)
  - WHERE type = 'ERROR'

Confluent – KSQL



Structured Streaming – Kstreams and KTable



KSQL Joins

```
ksql> CREATE STREAM RATINGS_WITH_CUSTOMER_DATA WITH (PARTITIONS=1) AS \
SELECT R.RATING_ID, R.CHANNEL, R.STARS, R.MESSAGE, \
   C.ID, C.CLUB_STATUS, C.EMAIL, \
   C.FIRST_NAME, C.LAST_NAME \
FROM RATINGS R \
   INNER JOIN CUSTOMERS C \
     ON R.USER_ID = C.ID;
```

Thanks