




# Apache Kafka Kibana Data pipeline














Somkiat


Home









Update Info 1
View Activity Log 10+
...


Timeline
About
Friends 3,138
Photos
More ▾



When did you work at Opendream?
×





...
22 Pending Items



Intro


Software Craftsmanship


Software Practitioner at สยามชำนานุกิจ พ.ศ. 2556


Agile Practitioner and Technical at SPRINT3r



Post

Photo/Video

Live Video

Life Event


What's on your mind?


Public ▾

Post



**Somkiat Puisungnoen**
15 mins · Bangkok ·  ▾

Java and Bigdata

...



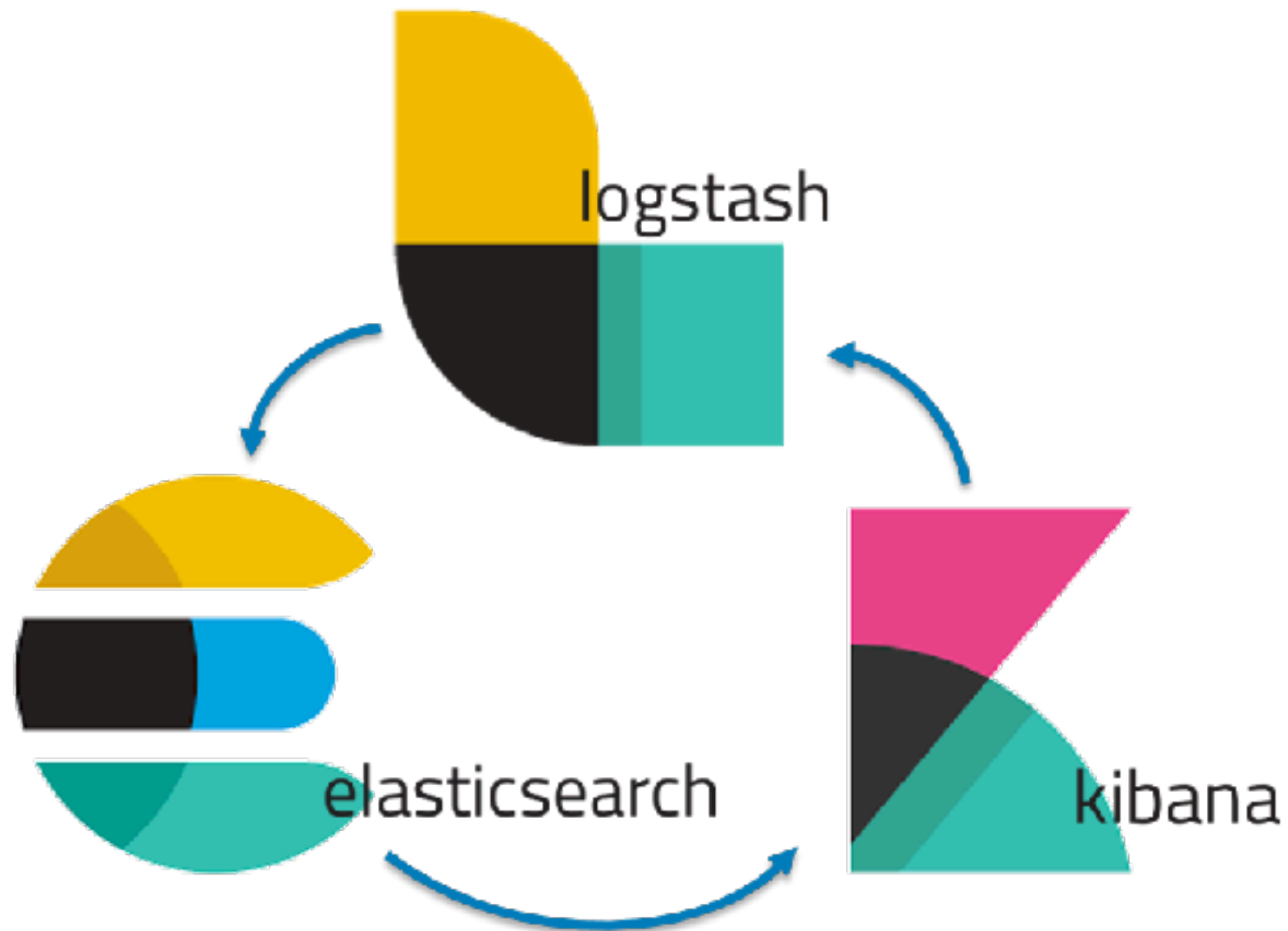


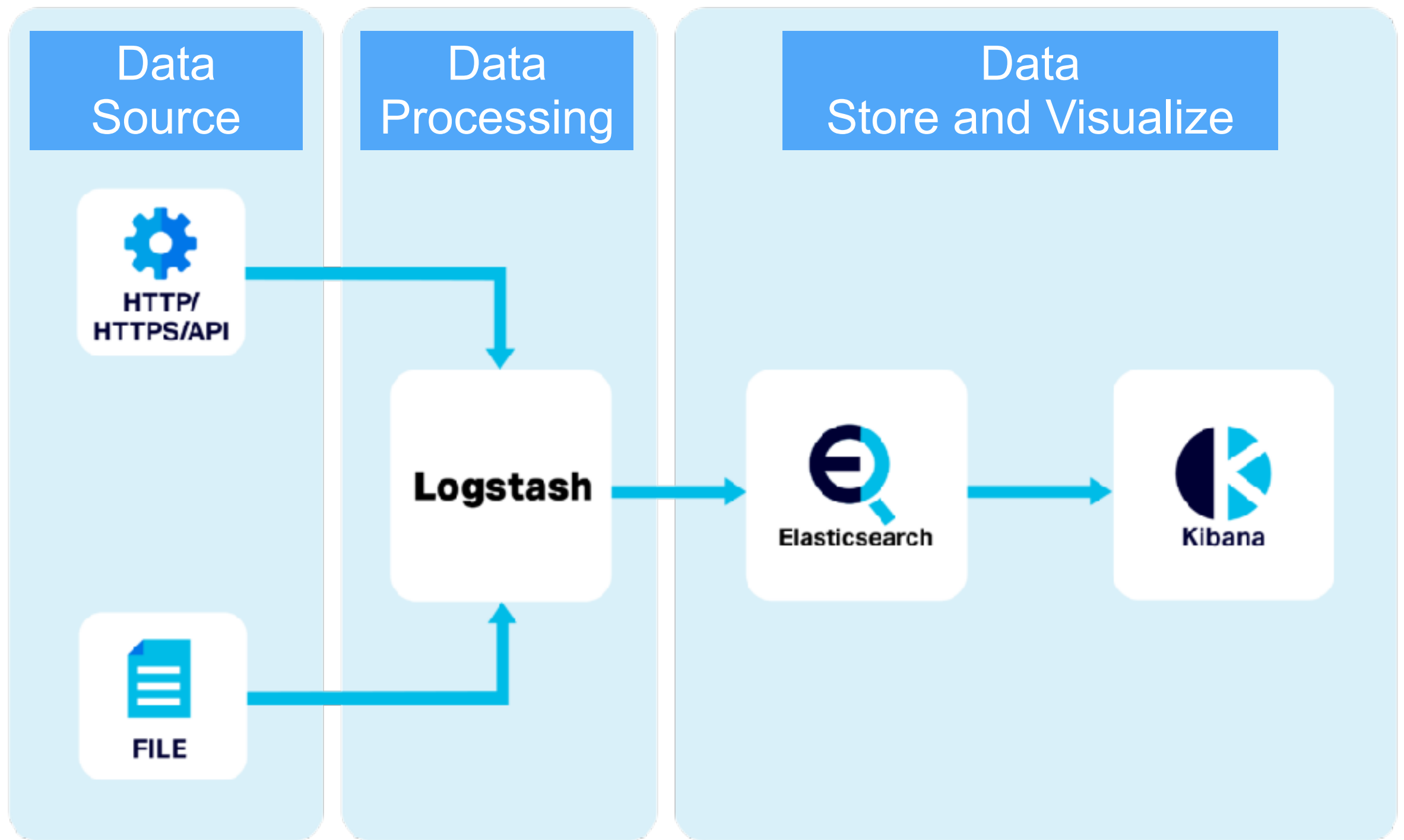
# Agenda

- Recap ELK stack
- Apache Kafka
- Working with Kibana
- Data pipeline
- Demo



# ELK stack





Transform data

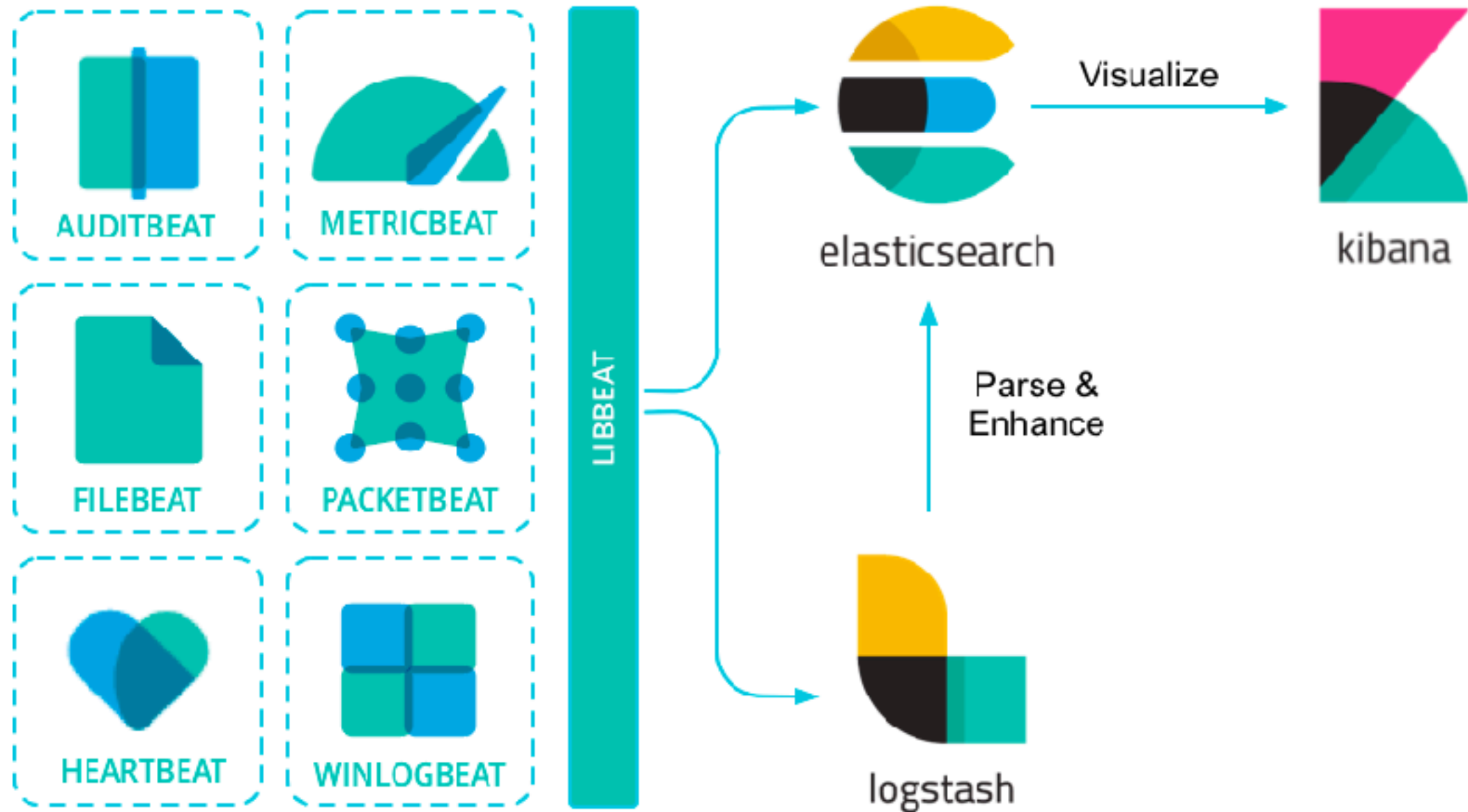
Data Store

Visualize





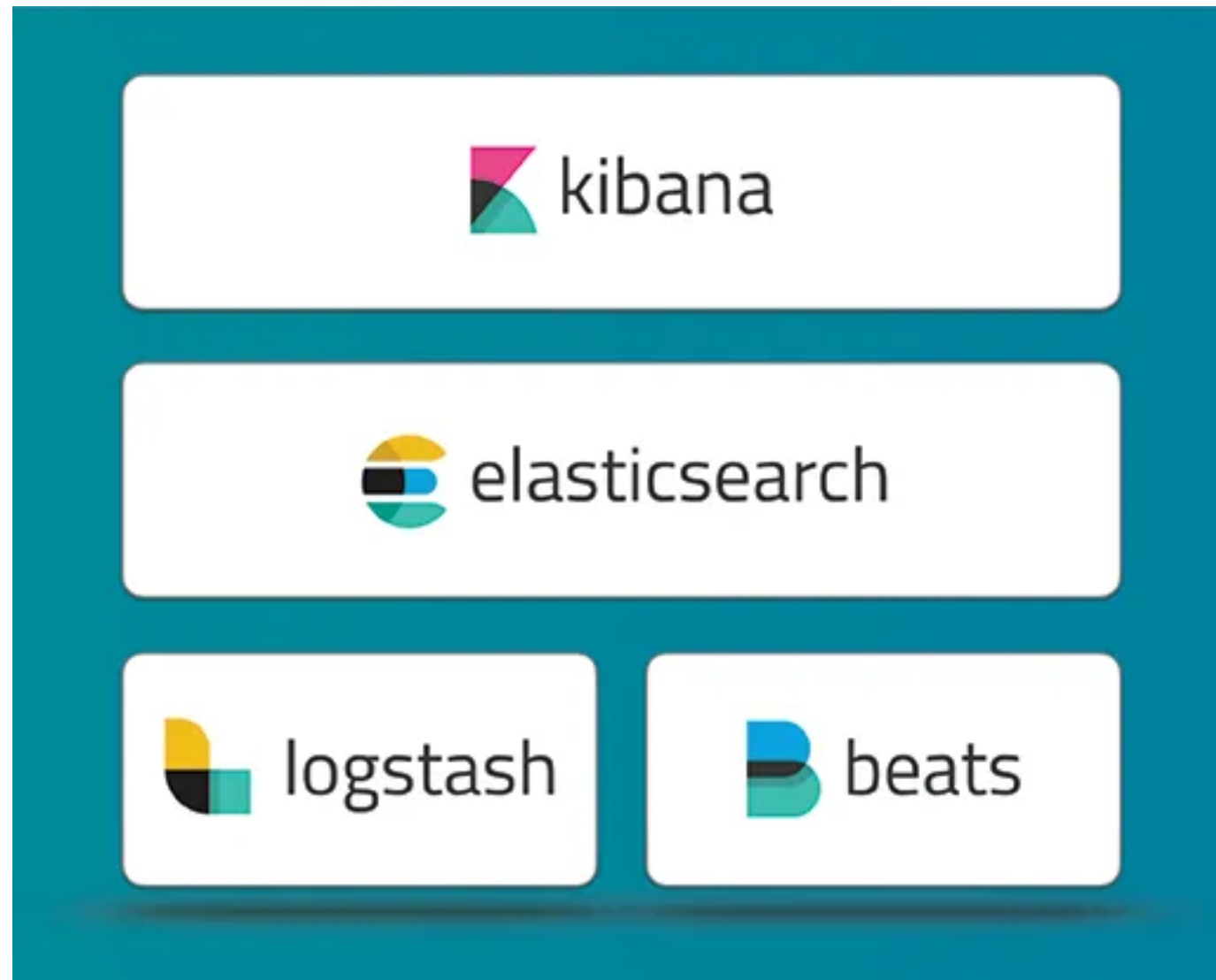
# Beat



<https://www.elastic.co/guide/en/beats/libbeat/current/index.html>



# ELK stack



# Beat

Purpose	Library
Audit data	Auditbeat
Log files	Filebeat
Cloud data	Functionbeat
Availability	Heartbeat
Metrics	Metricbeat
Network traffic	Packetbeat
Windows event logs	Winlogbeat

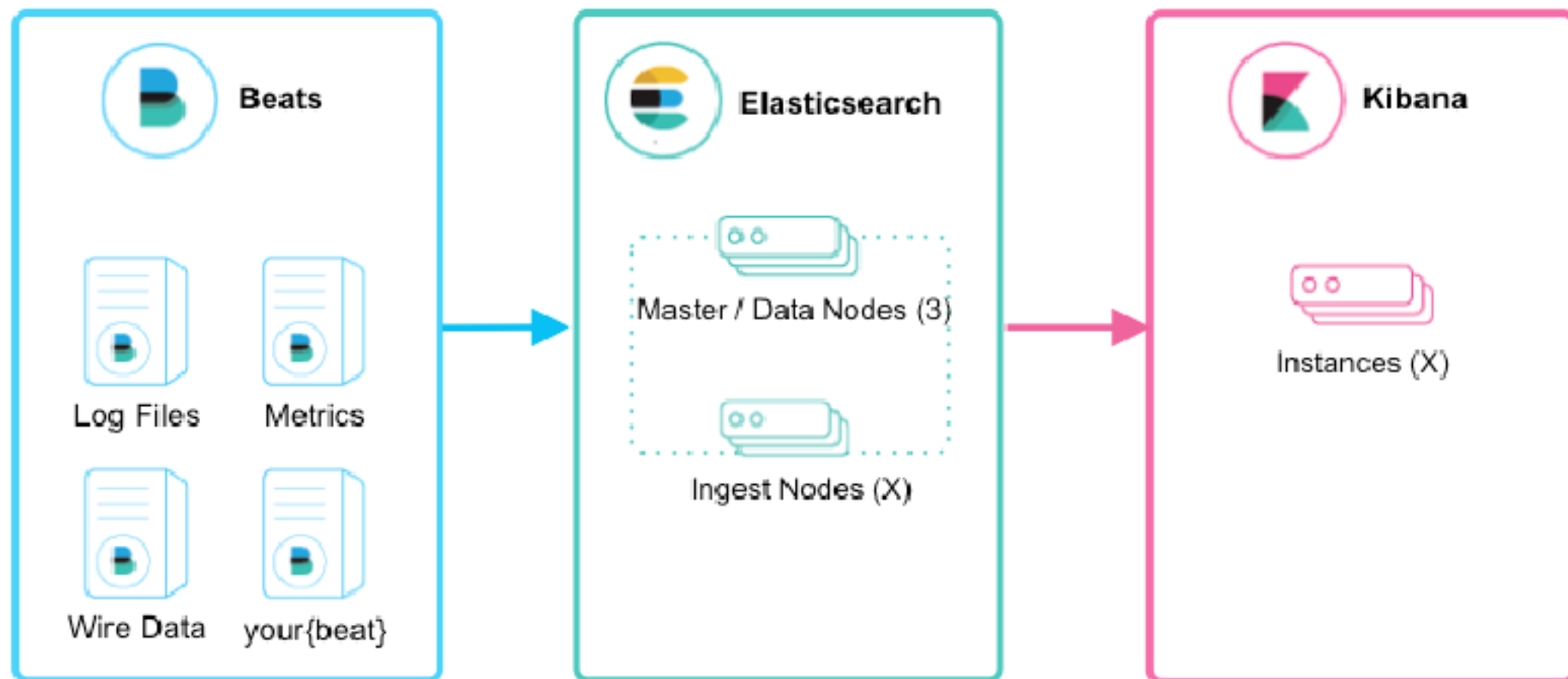
<https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html>



# Scaling ELK Stack



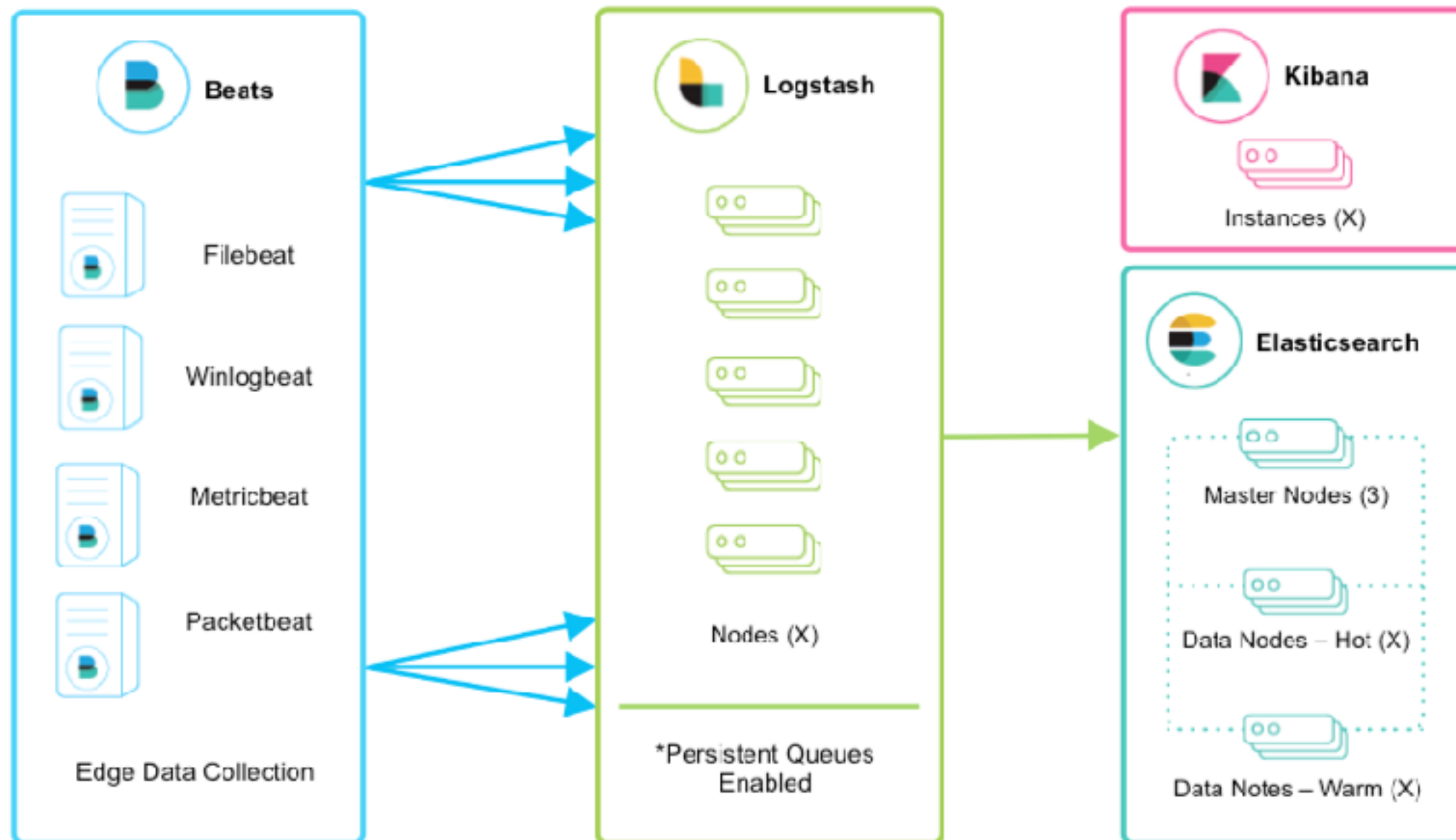
# Basic



<https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash>



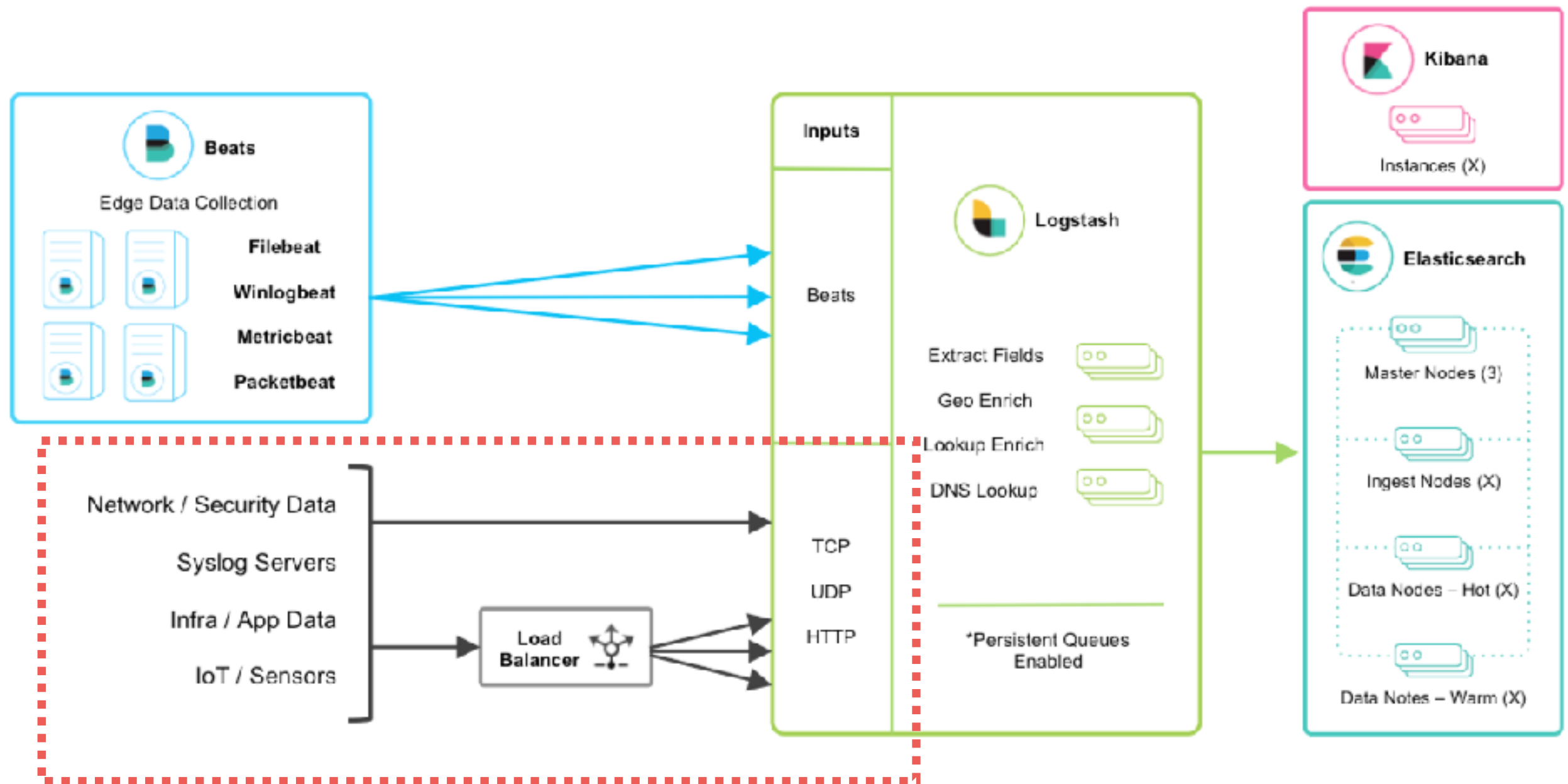
# Scaling Ingest (Add more nodes)



<https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash>



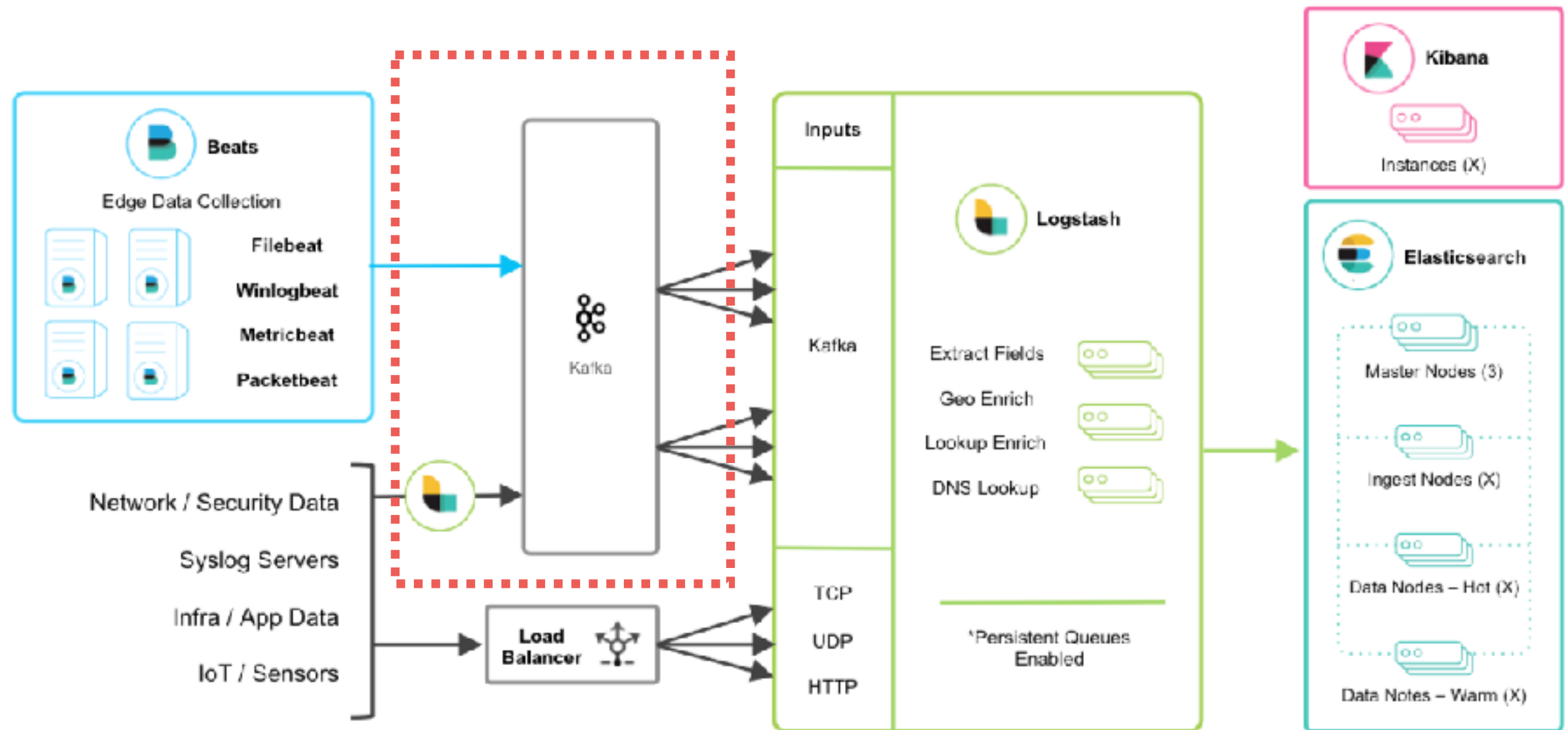
# Integrate with More data source



<https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash>



# Integrate with Messaging



<https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash>





# Messaging !!





Cloud Pub/Sub



# Why need Messaging ?

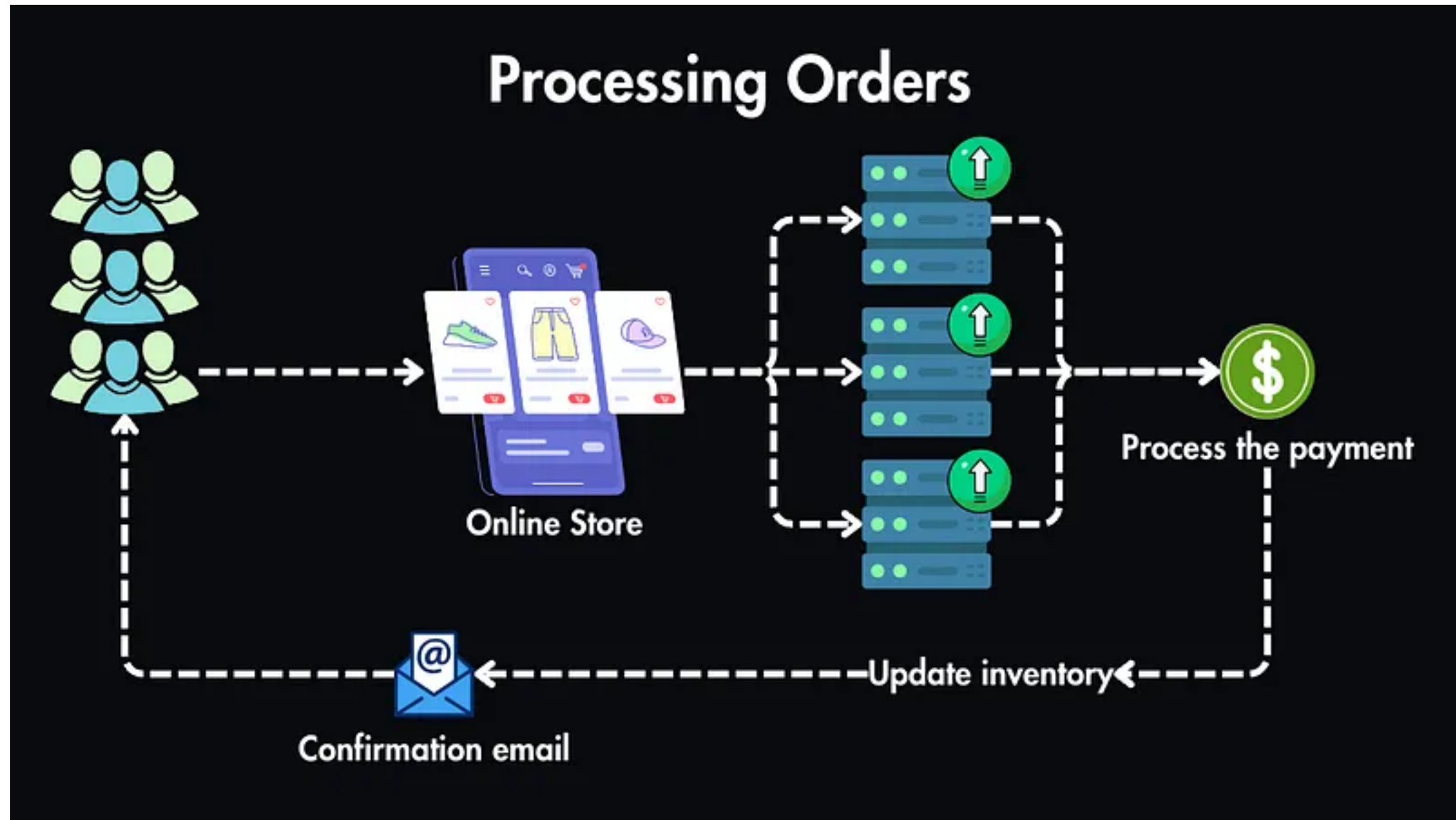
Decoupling between systems

Buffering

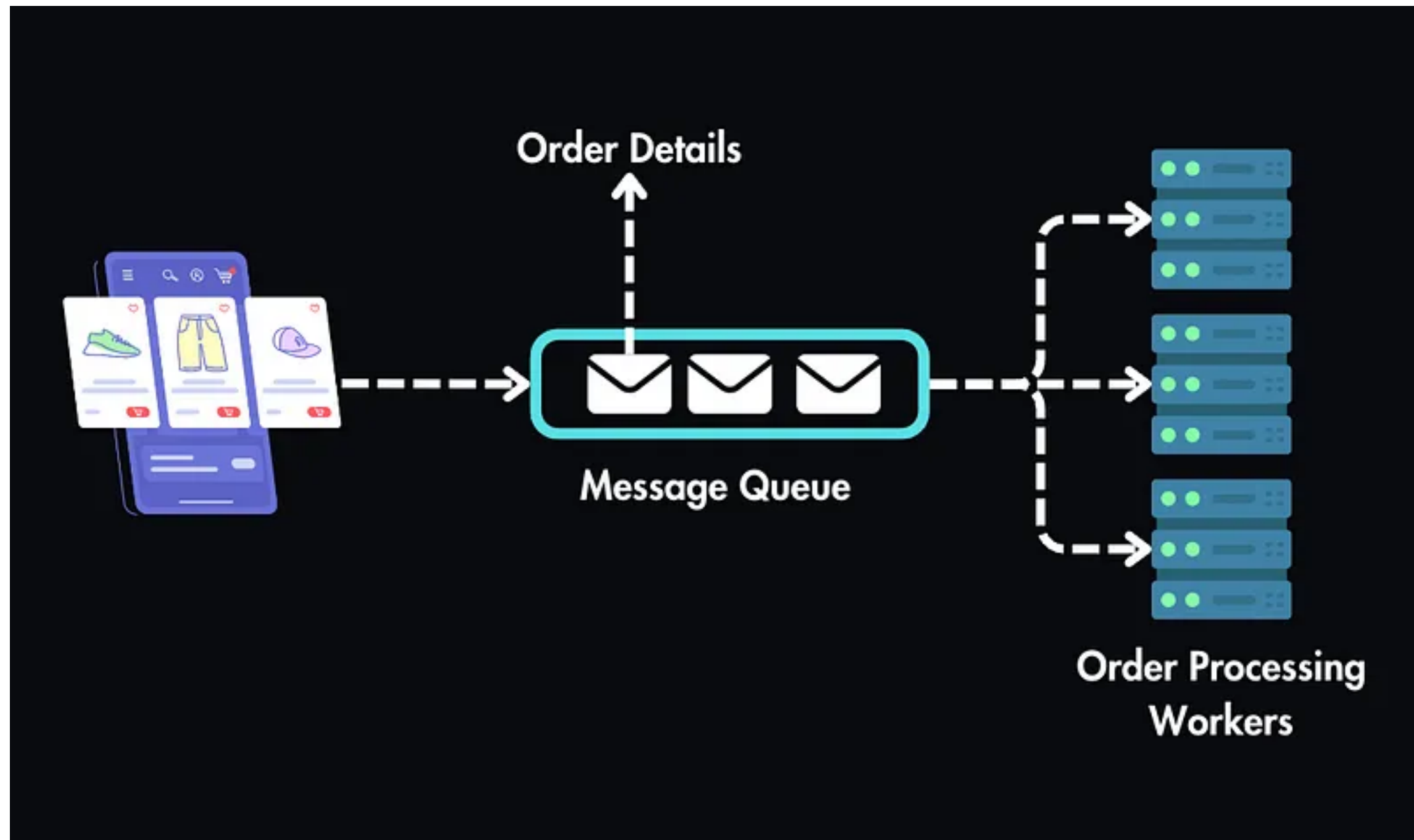
Back pressure problem



# Processing Orders



# Order processing with messaging



# Durable

If the queue crashed, that data will not lost

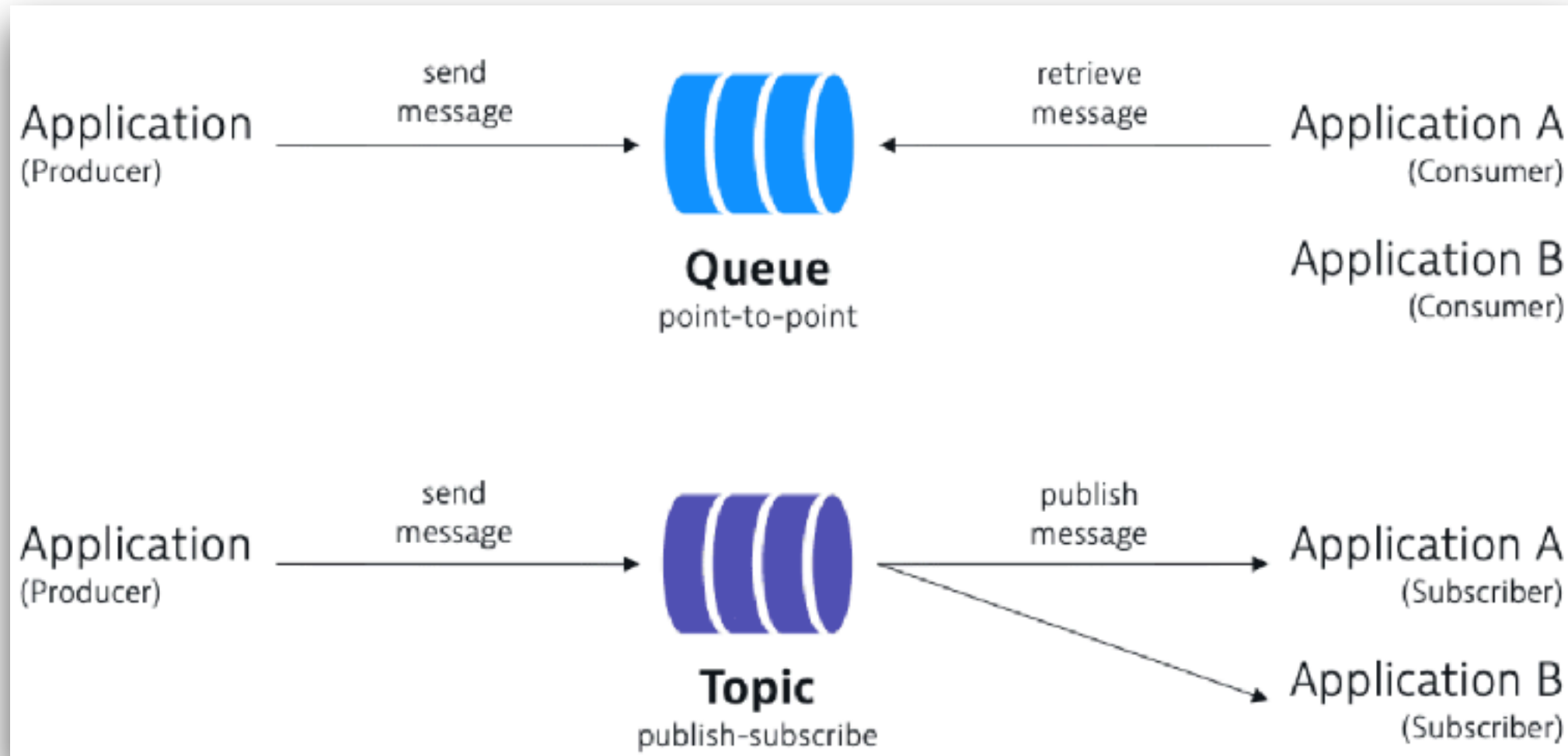


# Reliable

If consumer/worker failed, message still in queue

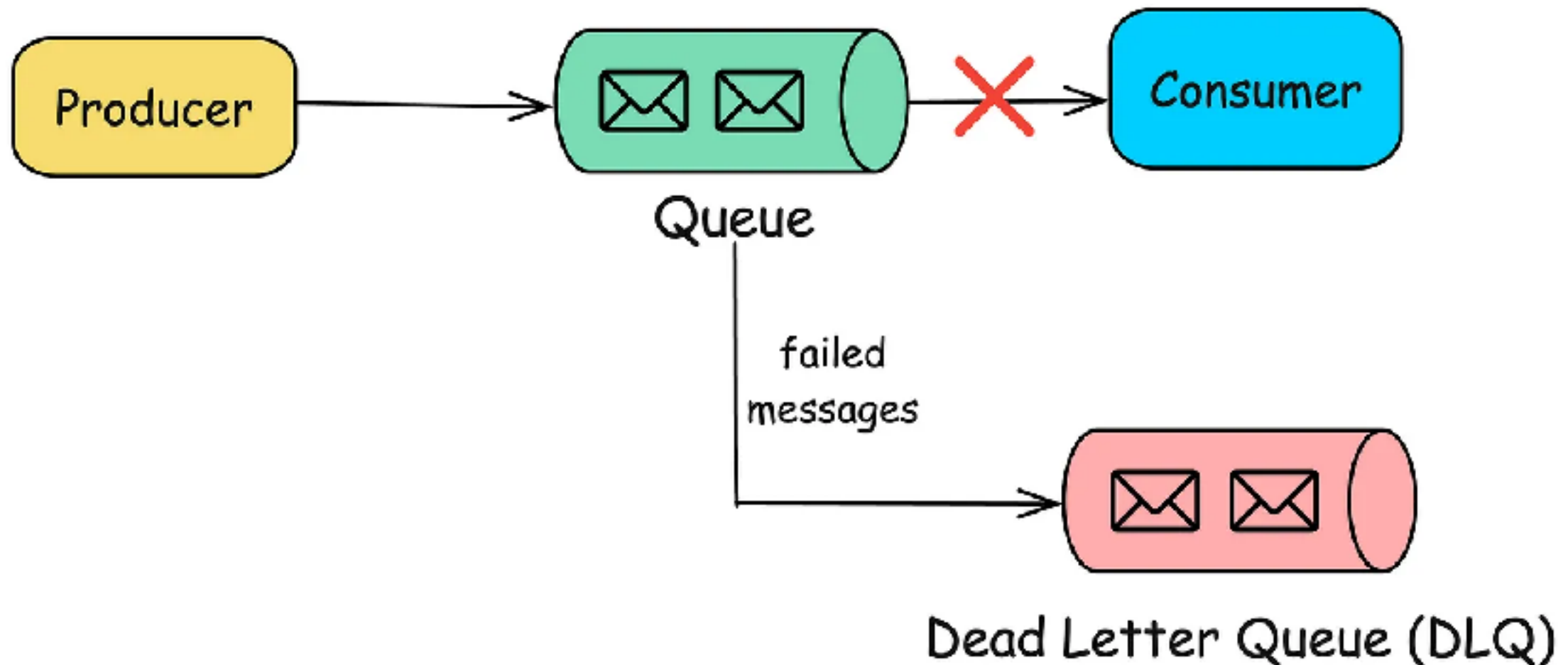


# Queue vs Topic





# Dead Letter Queue



<https://blog.algomaster.io/p/message-queues>



# Apache Kafka

<https://kafka.apache.org/>



# Kafka Use Cases

Realtime data streaming

Log aggregation

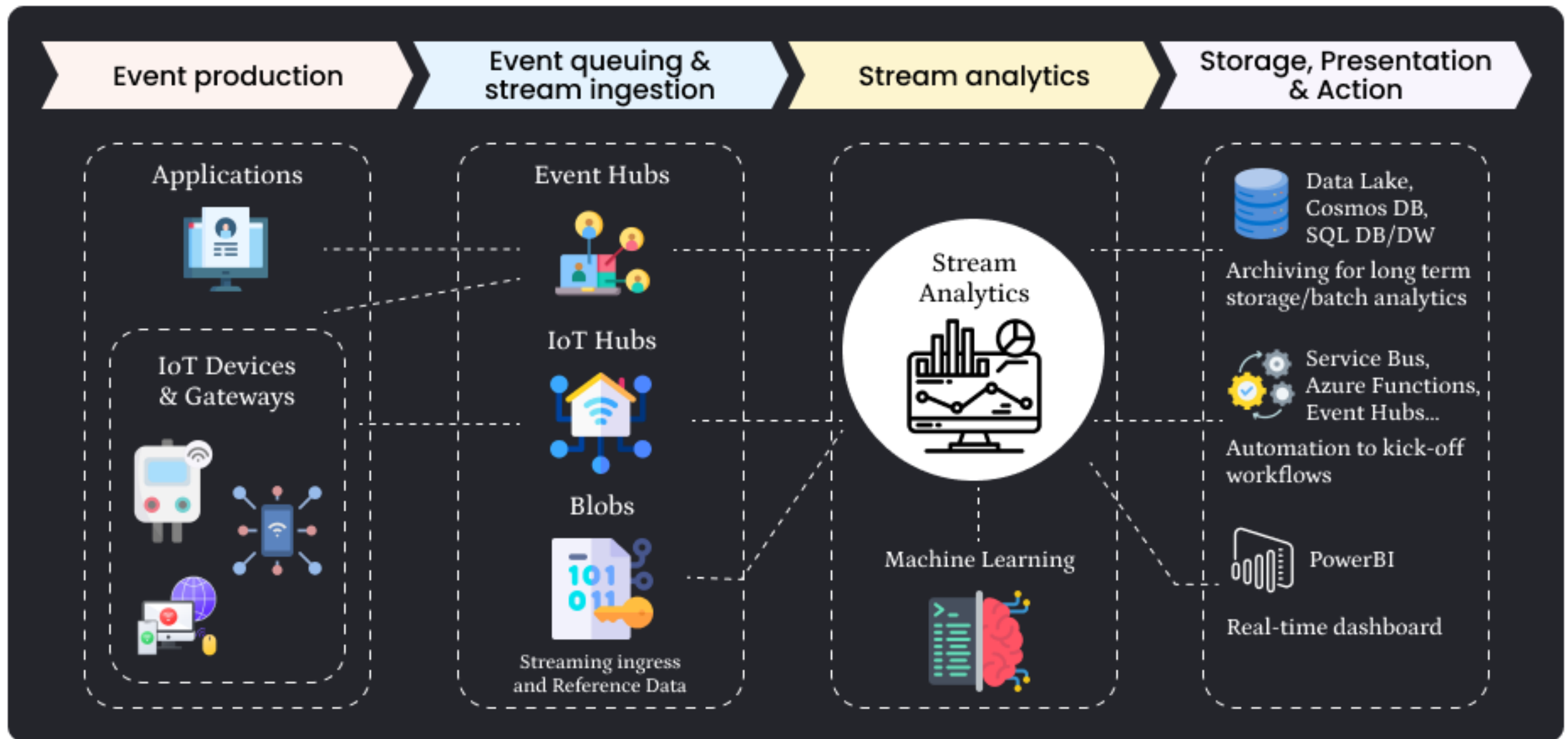
Metrics collection

Event sourcing

Streaming processing



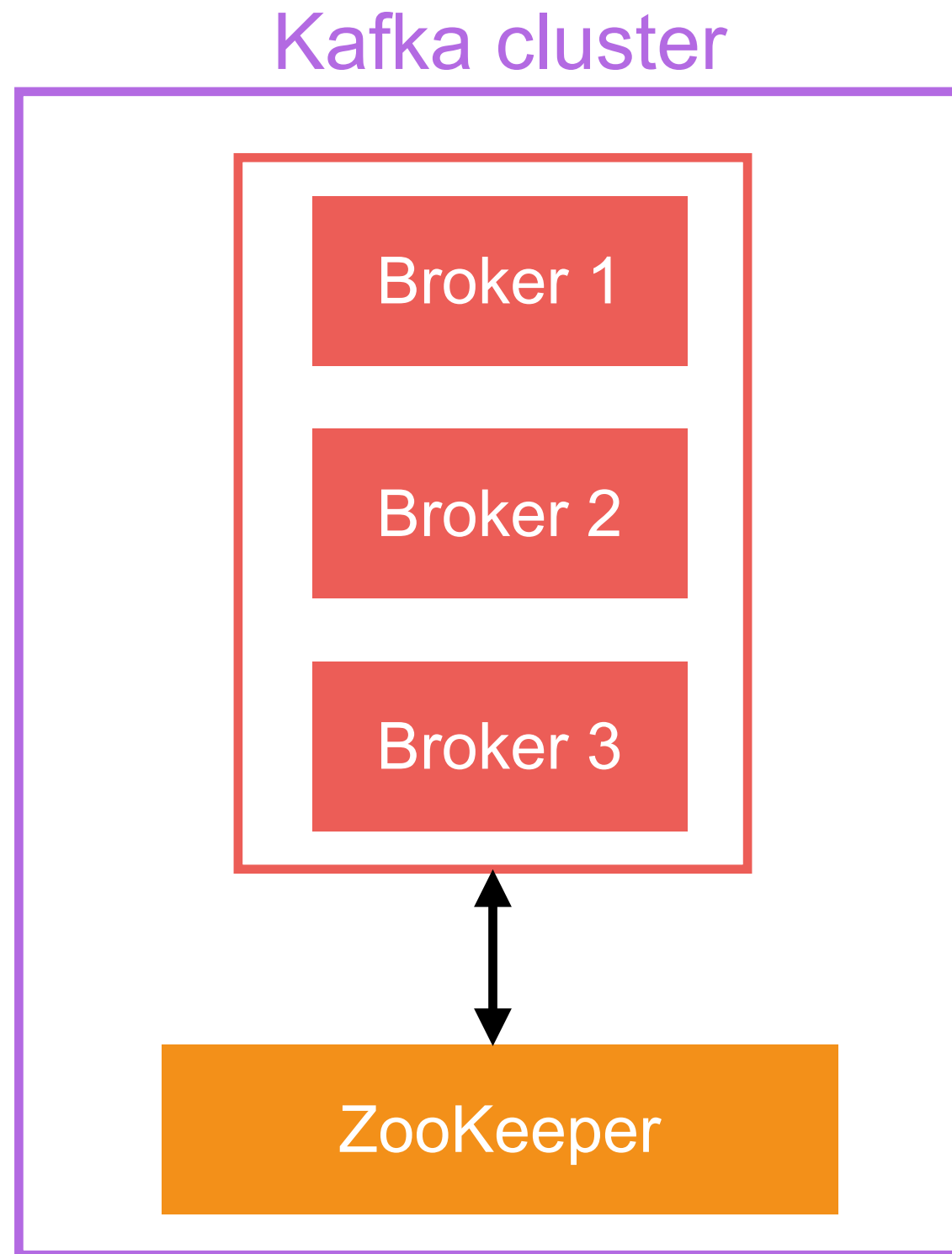
# Kafka Use Cases



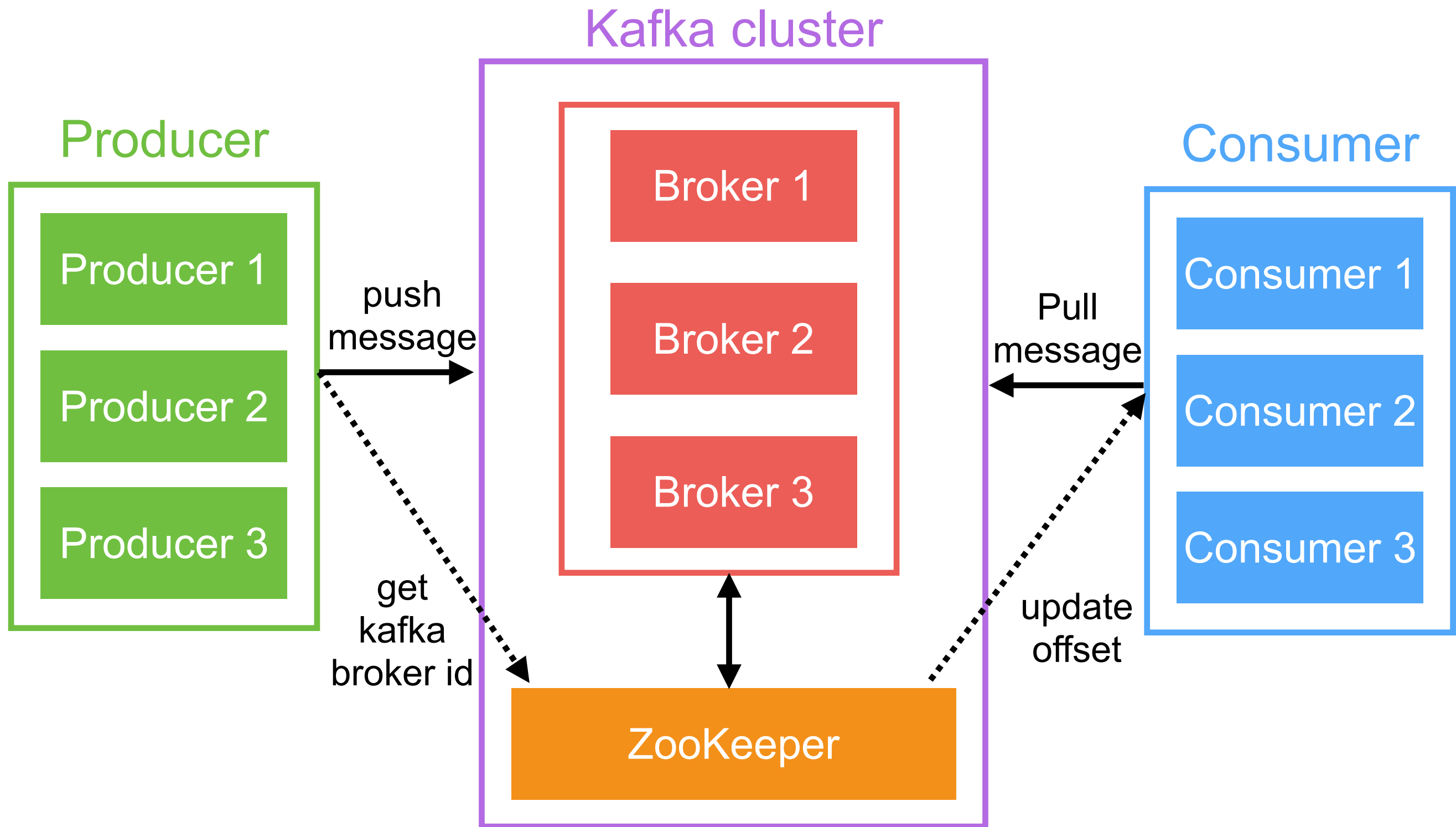
<https://www.softqubes.com/blog/apache-kafka-a-comprehensive-guide-to-real-time-data-streaming-and-processing/>



# Kafka Architecture



# Kafka Architecture



# APACHE KAFKA

# 4.0

2025/04/18



# Kafka 4.0

Remove Apache Zookeeper, use Kraft



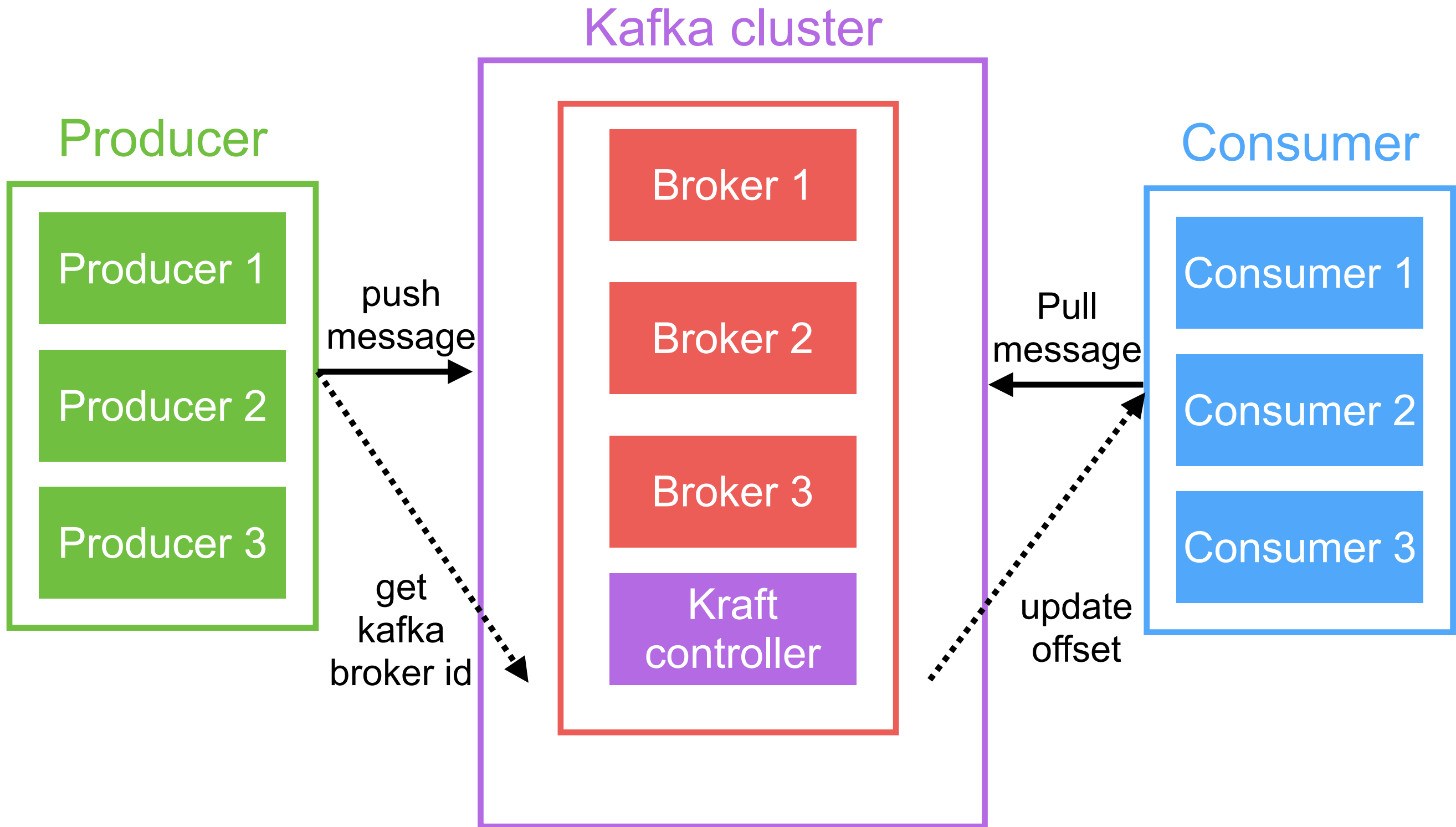
APACHE  
**ZooKeeper**<sup>TM</sup>

<https://zookeeper.apache.org/>

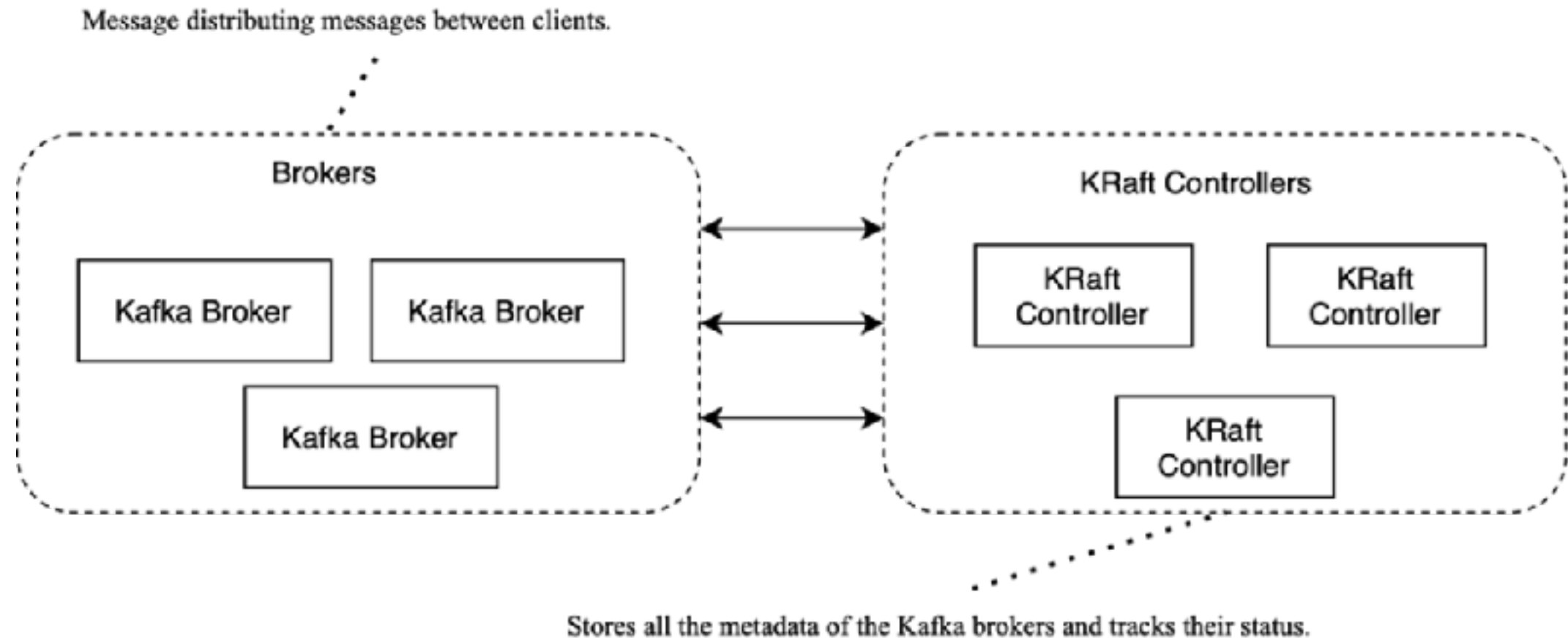




# Kafka Architecture 4.0



# Kafka broker and KRaft controller



# Kafka Fundamentals



# Kafka Fundamentals

Event/record/message

Producers

Consumers

Topics

Partitions

Broker

Consumer groups

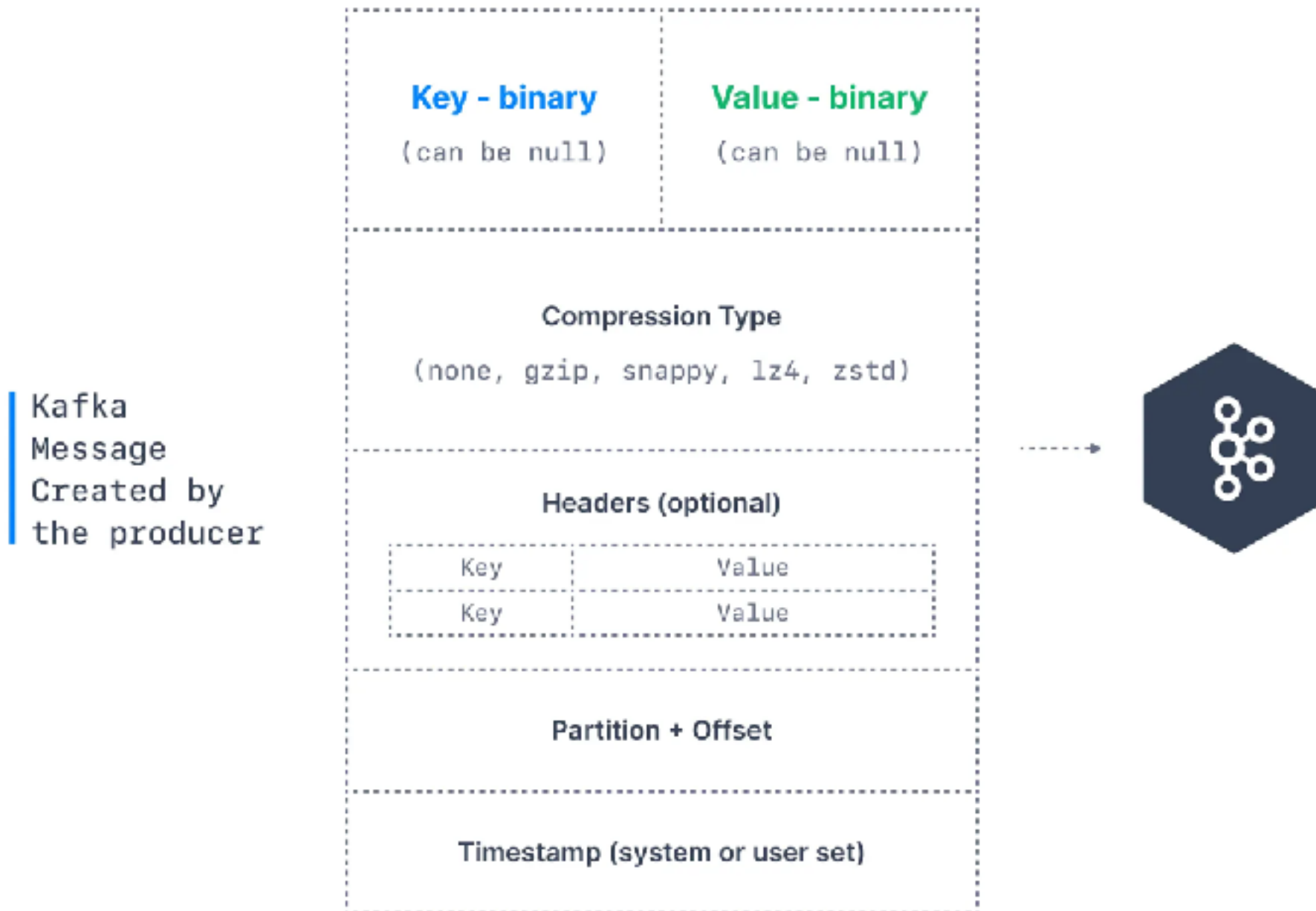
Cluster/Replicate

Offset (read and write)

**Kafka 4.0 :: Queue => share group (Early access)**



# Event Anatomy



# Topics, partitions and offsets

## Topics

Stream of data

Similar to a table in database

You can have many topics as you want

A topic is identified by **name**



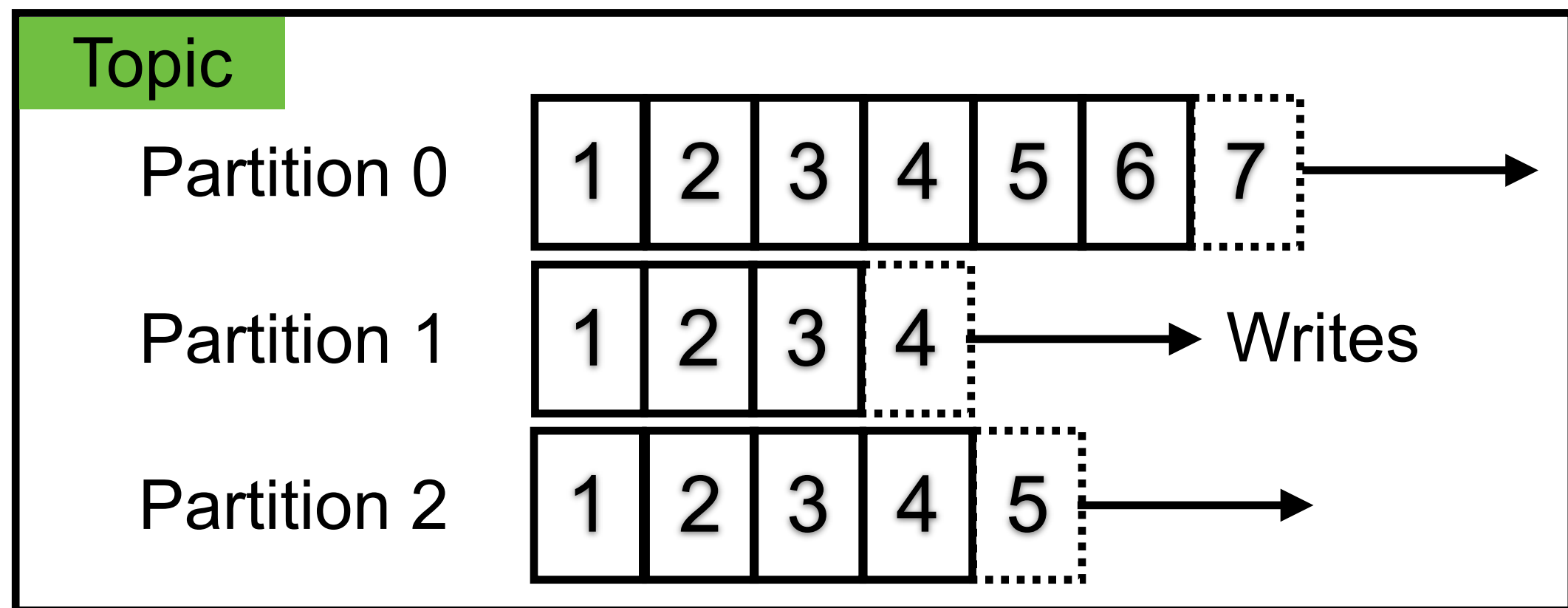
# Topics, partitions and offsets

## Partitions

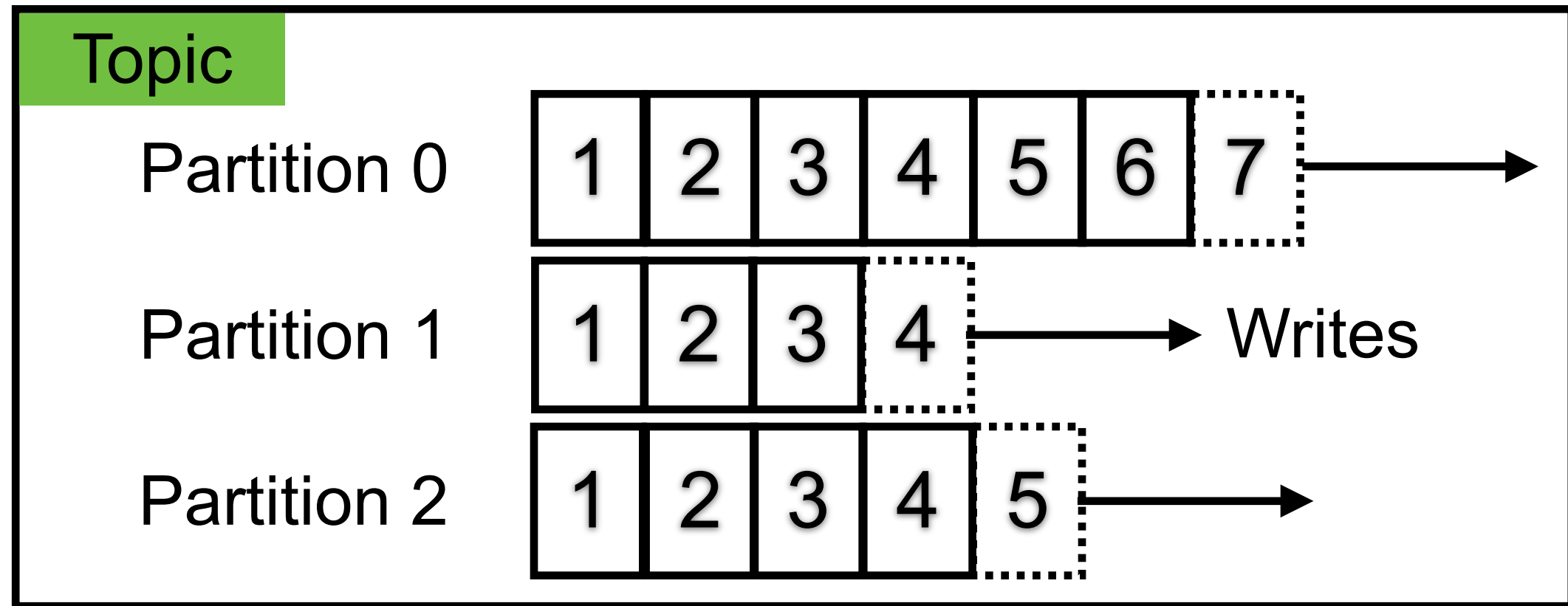
Topics are split in partitions

Each partition is **ordered**

Each message in partition get incremental id (**offset**)



# Topics, partitions and offsets



Order is guaranteed only within partition

Data is keep in limited time (**Default = 1 week**)

Data is **immutable** (can't be changed)

Data is assigned **randomly** to a partition





# More partitions ?

Higher is your data throughput

More open files

Longer downtime

More RAM is consumed by clients

4000  
per broker

200,000  
per cluster

50 brokers  
per cluster

<https://api7.ai/blog/why-kafka-needs-an-api-gateway>



# Brokers and topics

## Brokers

Kafka cluster is composed of multiple brokers (servers)

Each broker is identified by ID (integer)

Each broker contains certain topic partitions

After connecting any broker (**bootstrap broker**),  
you will be connected to the entire cluster

Broker 1

Broker 2

Broker 3



# Kafka broker discovery

Every Kafka broker is called “**bootstrap server**”

You only need to connect to one broker, and you will be connected to the entire cluster

Broker 1  
(bootstrap)

Broker 2  
(bootstrap)

**Kafka cluster**

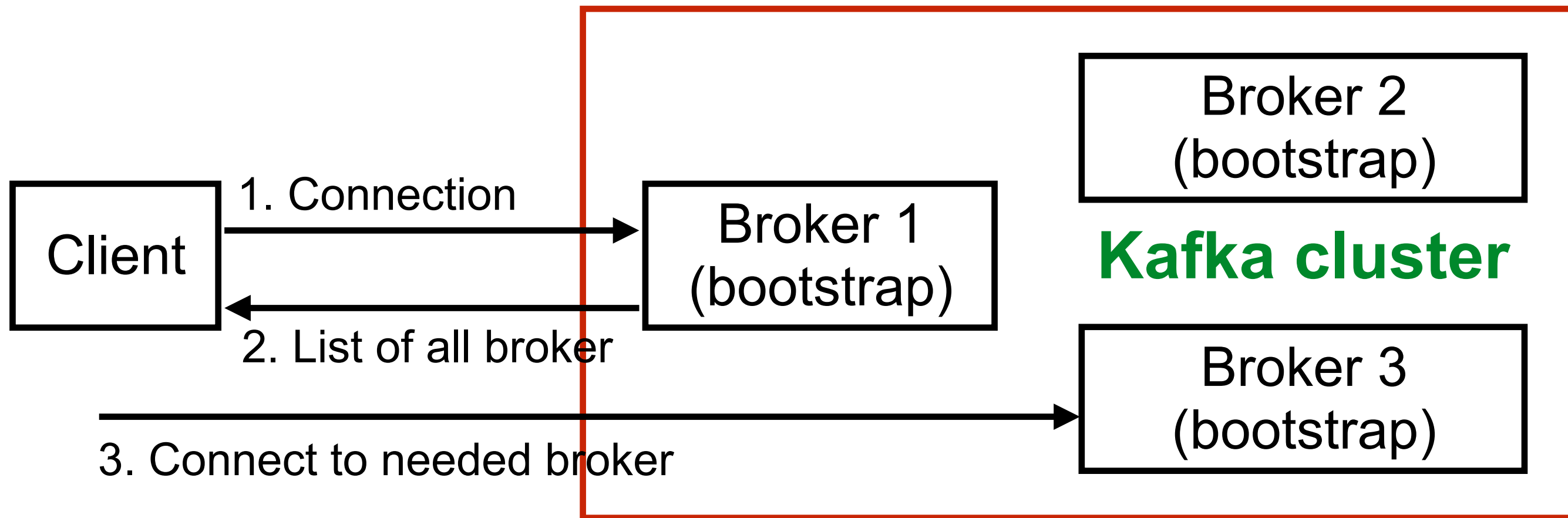
Broker 3  
(bootstrap)

Broker 4  
(bootstrap)



# Kafka broker discovery

Each broker knows about all brokers, topics and partitions (**metadata**)



# Apache Zookeeper

Kafka < 4.0 can not work without Zookeeper !!



<https://zookeeper.apache.org/>



# Apache Zookeeper

Zookeeper **manages** brokers

Zookeeper help in performing **leading election** for partitions

Zookeeper sends **notifications** to Kafka in case of changes (new topic, broker die)

Zookeeper by design operates with a odd number of servers (1, 3, 5, 7)



# Data in Zookeeper

Kafka's data operations	Format
Broker metadata	ID, hostname
Topic metadata	Topic name, partition count, replica count
Partition assignment	Leader partition
Consumer group metadata	Consumer group name
Cluster metadata	Active broker, list of topics, partitions
Leader election	Select leader of partition



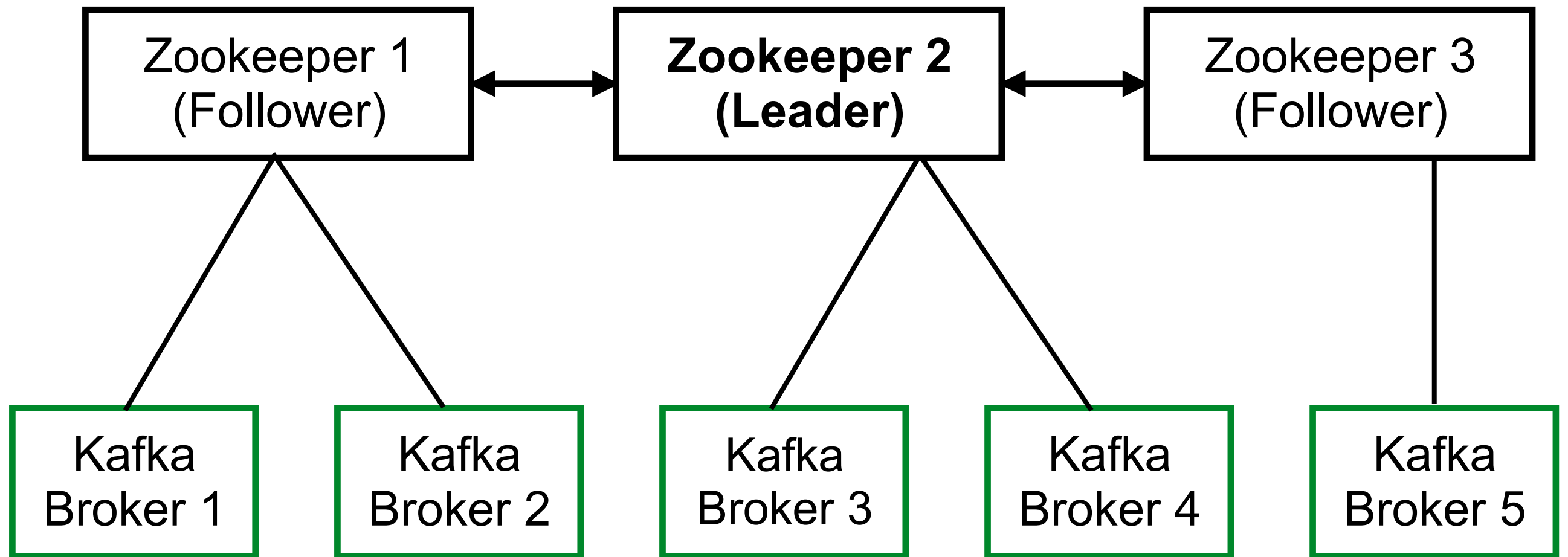
# Zookeeper architecture

Leader for write  
Follows for read





# Zookeeper architecture



# Brokers and topics

## Brokers

Good number to start id **3 brokers**

Broker 1

Broker 2

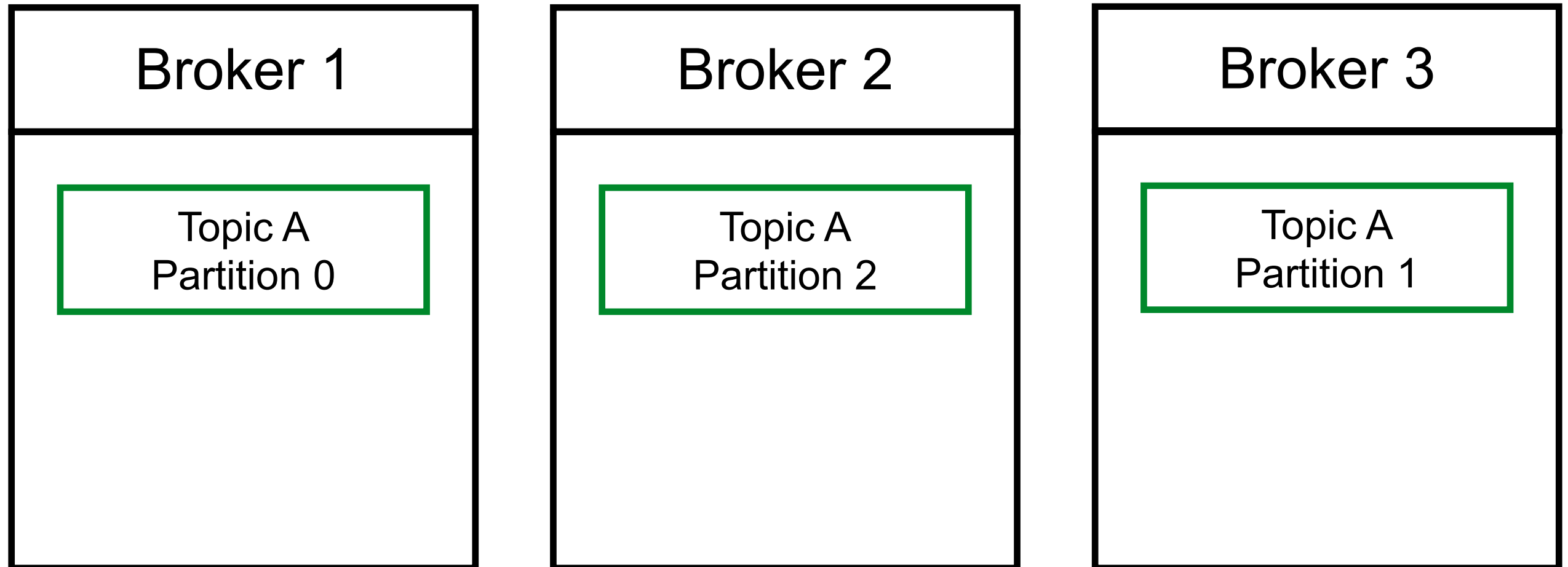
Broker 3



# Brokers and topics

## Example

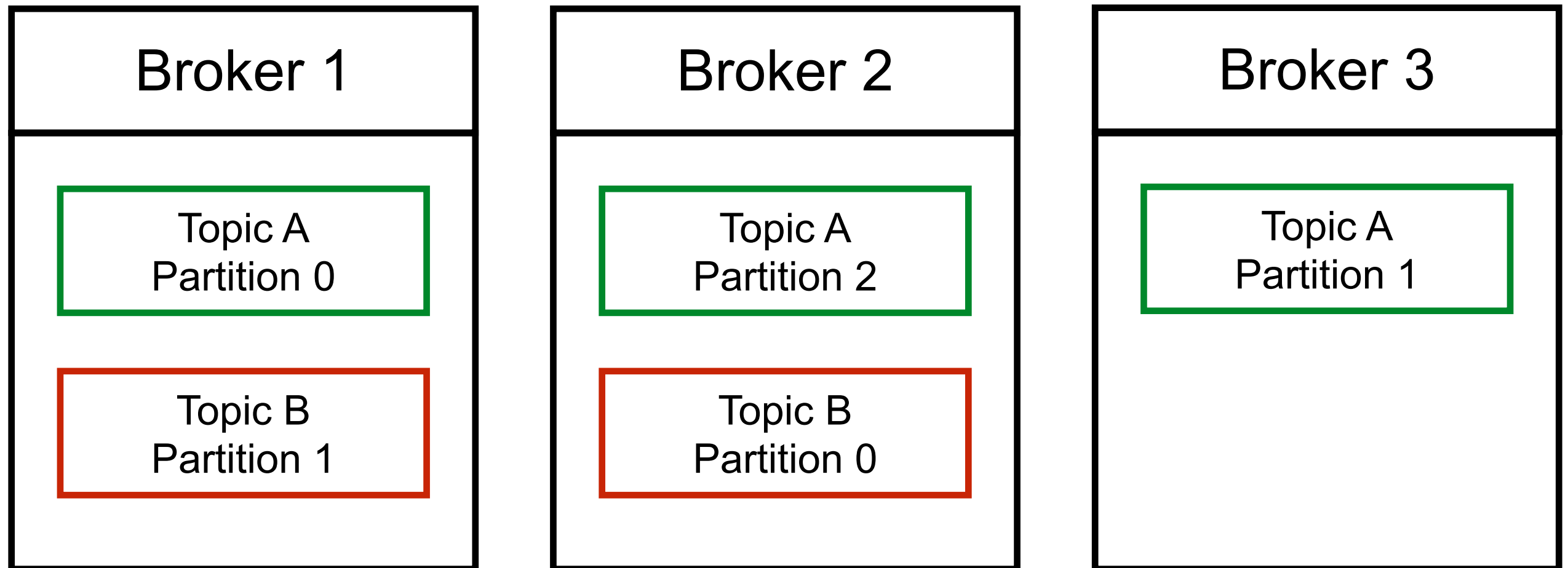
Topic A with 3 partitions



# Brokers and topics

## Example

Topic B with 2 partitions



# Topic with replication factor

Topic should have a replica factor  $> 1$  (2-3)

**replica factor  $<$  no. of brokers**

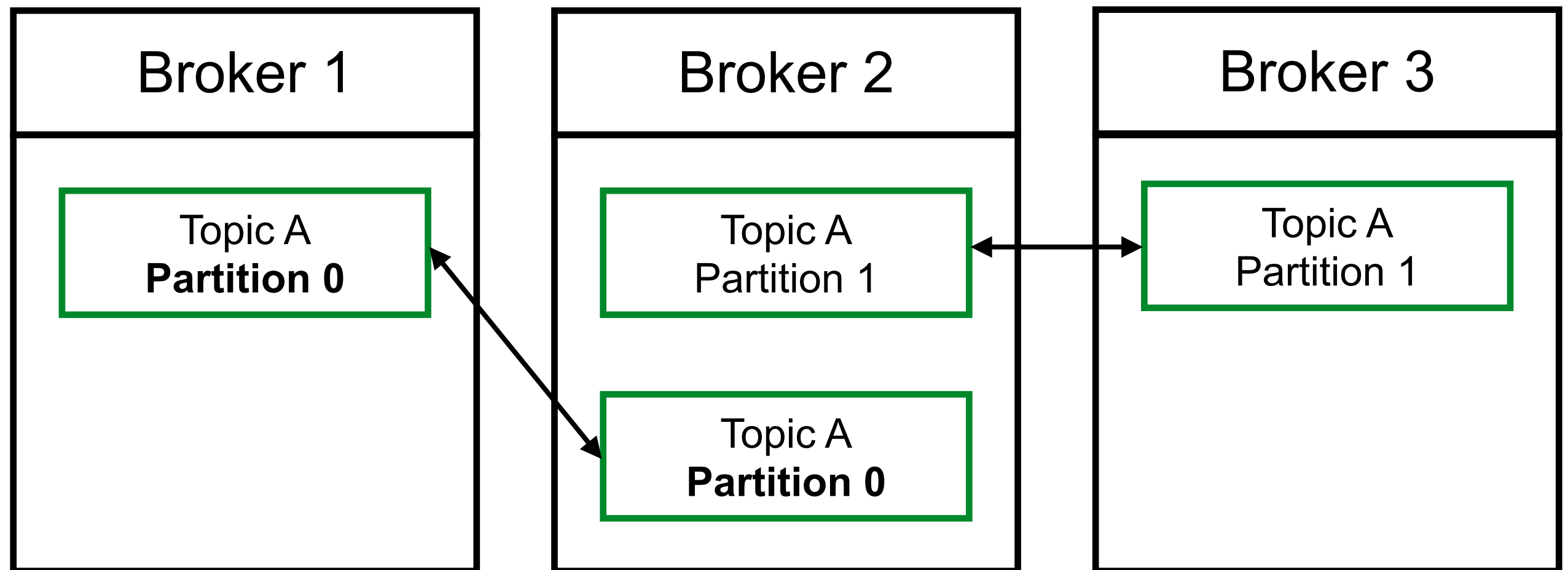
This way if a broker down, another broker can serve the data



# Topic with replication factor

## Example

Topic A with 2 partitions and replication factor = 2



# Leader for a partition

At any time only **one Broker** can be leader for partition

**Only leader partition** can receive and serve data for a partition

Other brokers will synchronize the data

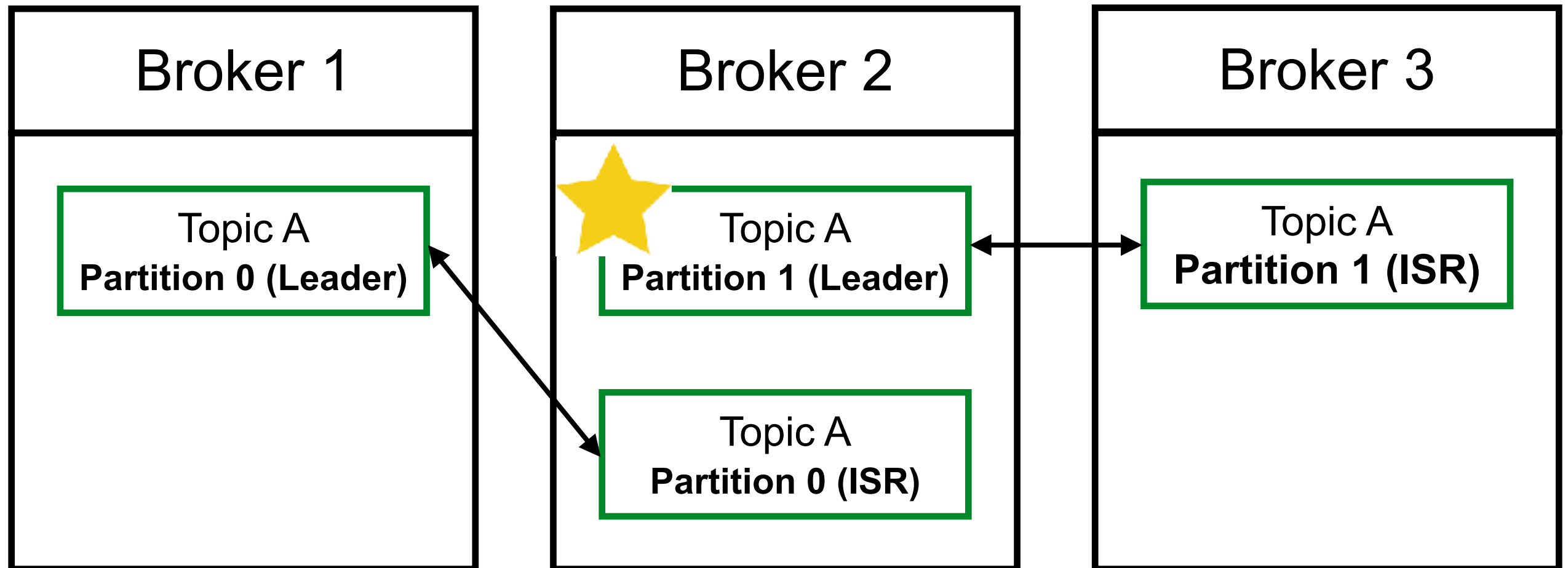
Other partition called **in-sync replica (ISR)**



# Leader for a partition

## Example

Topic A with 2 partitions and replication factor = 2





# Producers

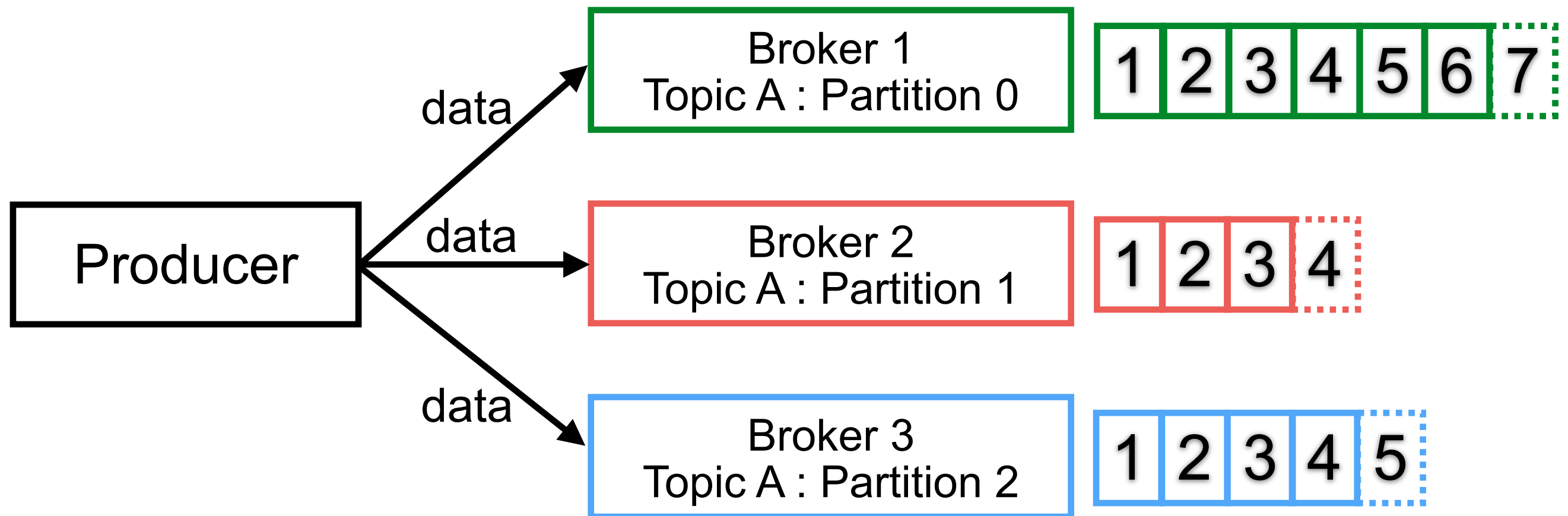
Producers write data to topics

Automatically know to broker and partition to write

When broker failures, producers will automatically recover



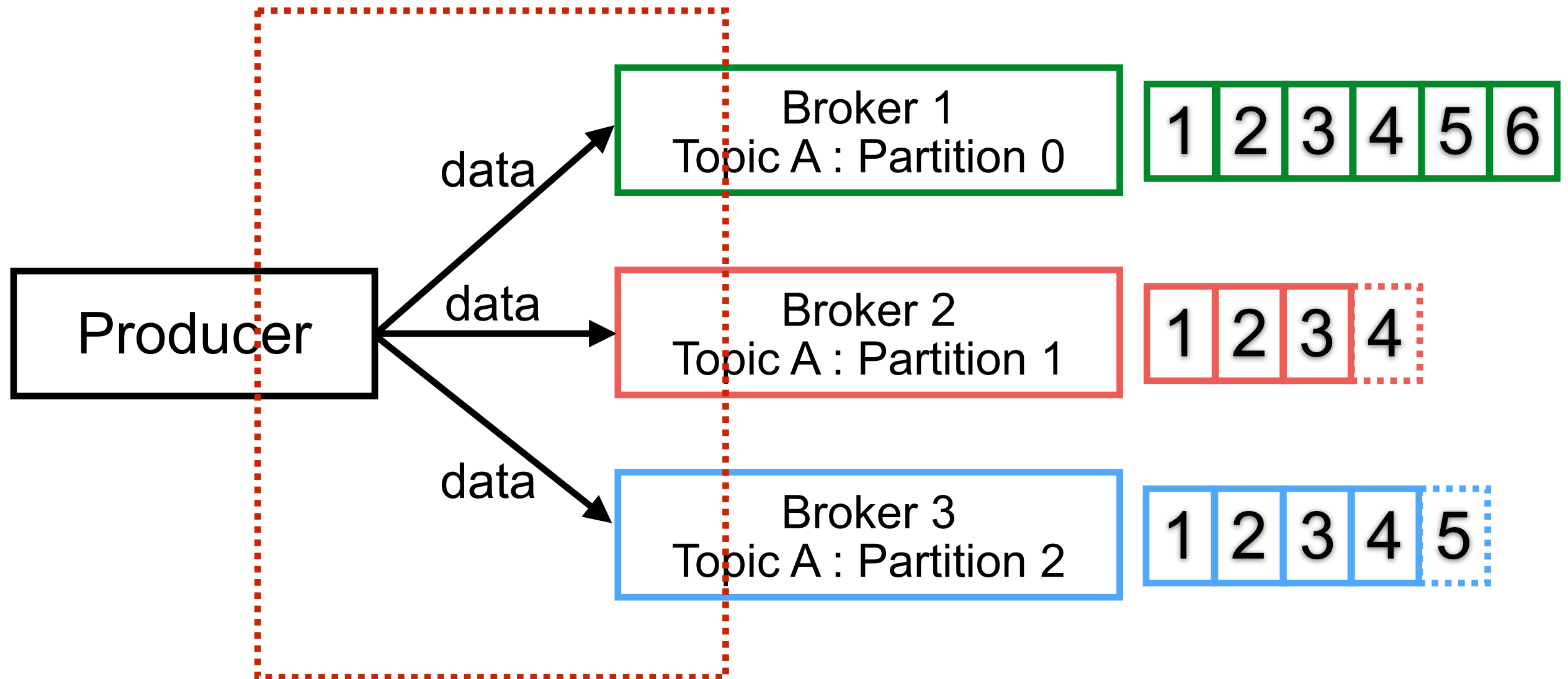
# Producers



\*\*\* *The load is balance to many brokers (no. of partitions)* \*\*\*



# Producers issue to write data !!



# Producers with acknowledgment

## **acks=0**

Producer not wait for acknowledgment

Possible data loss

## **acks=1**

Producer will wait for **leader** acknowledgment

Limited data loss

## **acks=all**

Producer will wait for **leader + ISR** acknowledgment

No data loss

<https://docs.confluent.io/platform/current/clients/producer.html>



# Partition assignment strategies

Range

Round Robin

Sticky

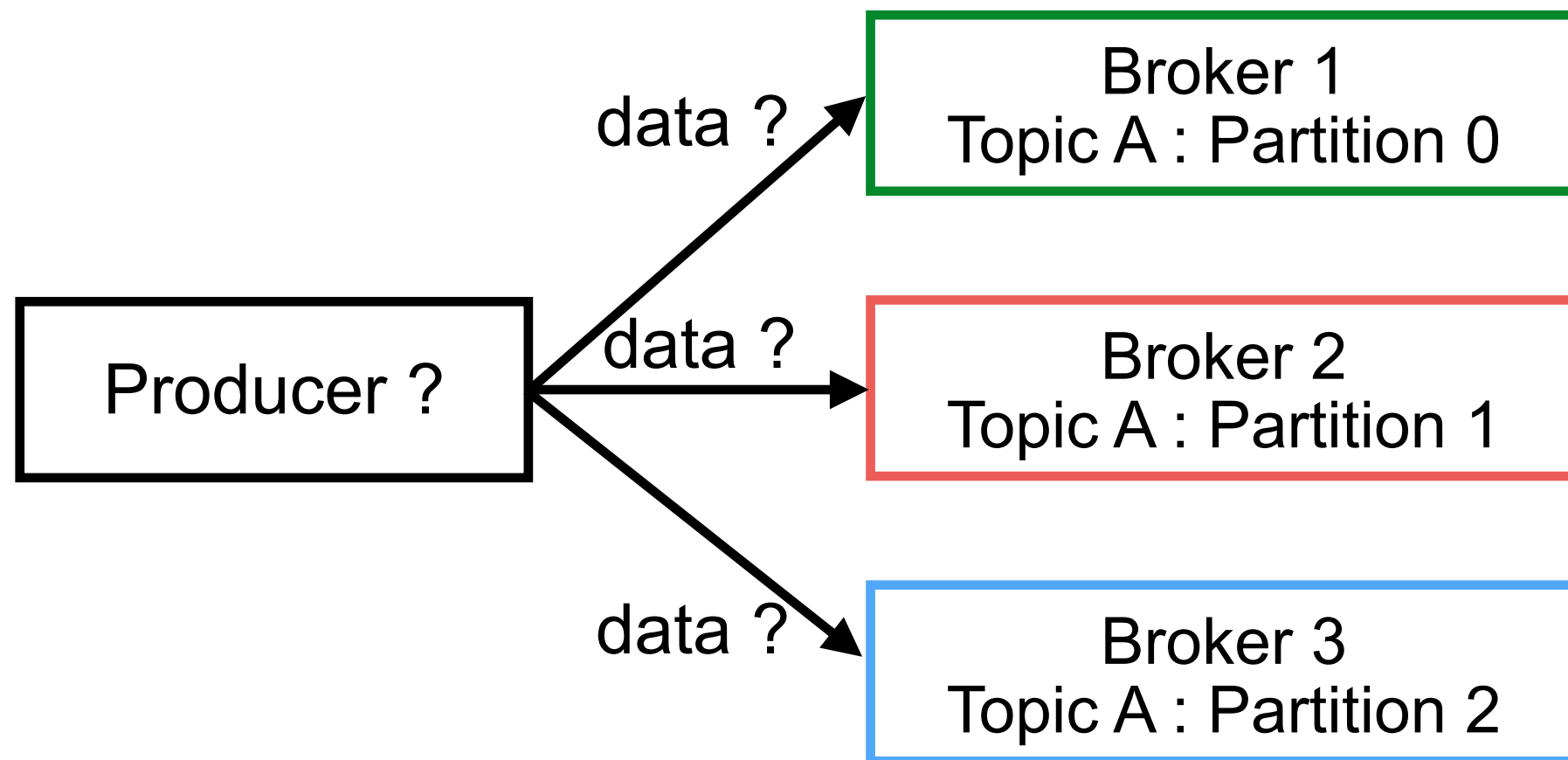
Customize (AbstractPartitionAssignor)

<https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html#partition-assignment-strategy>



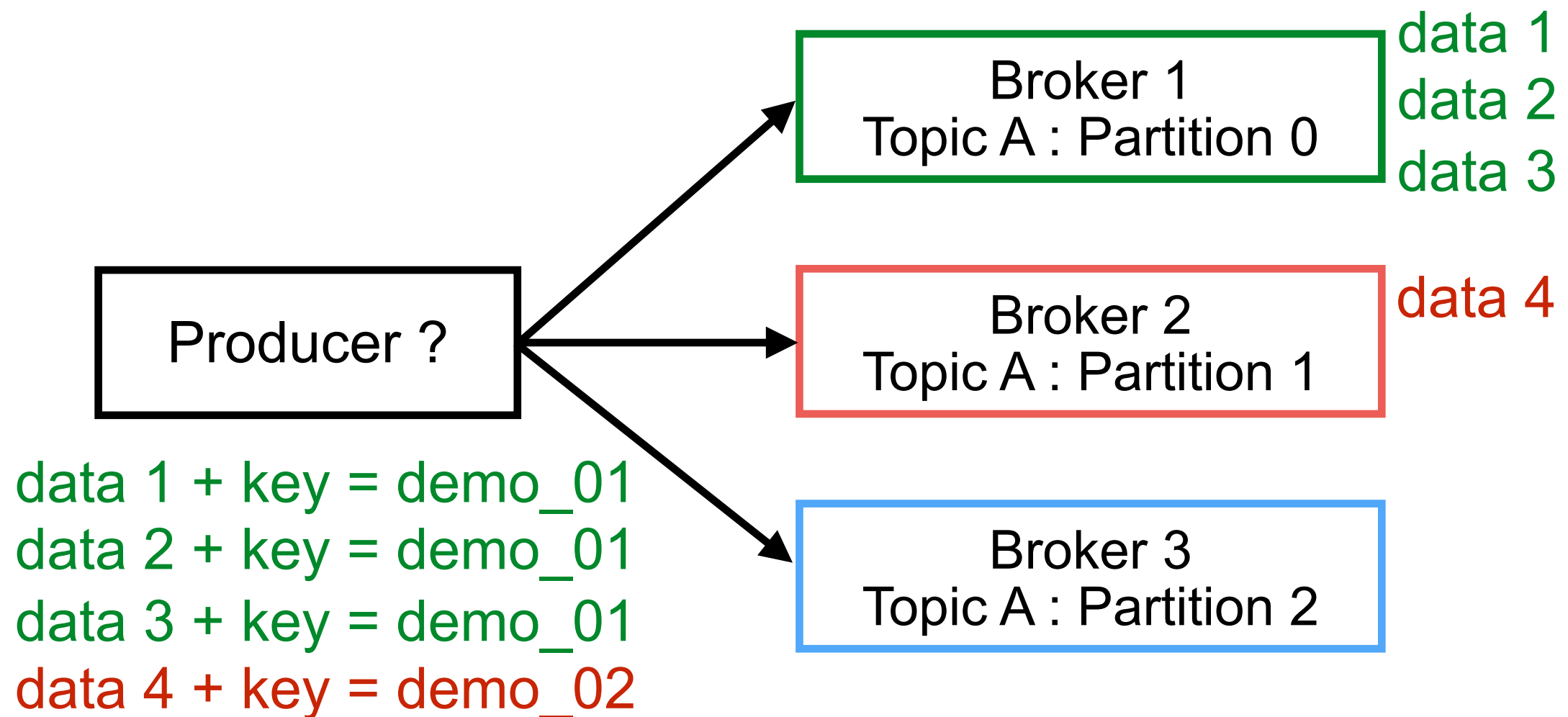
# Producers with message keys

Producers can choose to send a **key** with data  
**Key = null**, data is sent **Sticky partitioner**



# Why use message keys ?

You need message ordering



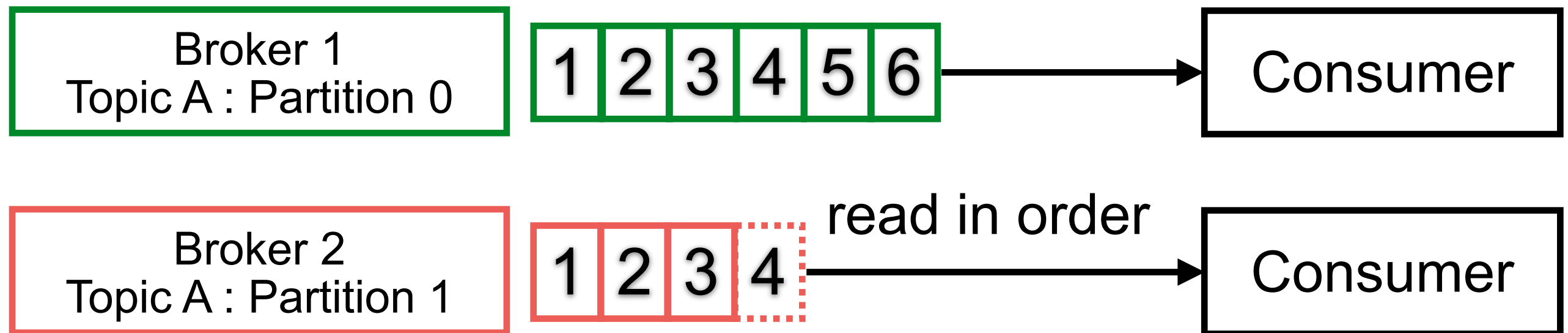
# Consumers

Consumers read data from topic

Consumers know which broker to read from

Data will read in order **within each partition**

When broker **failures**, consumer know how to recover

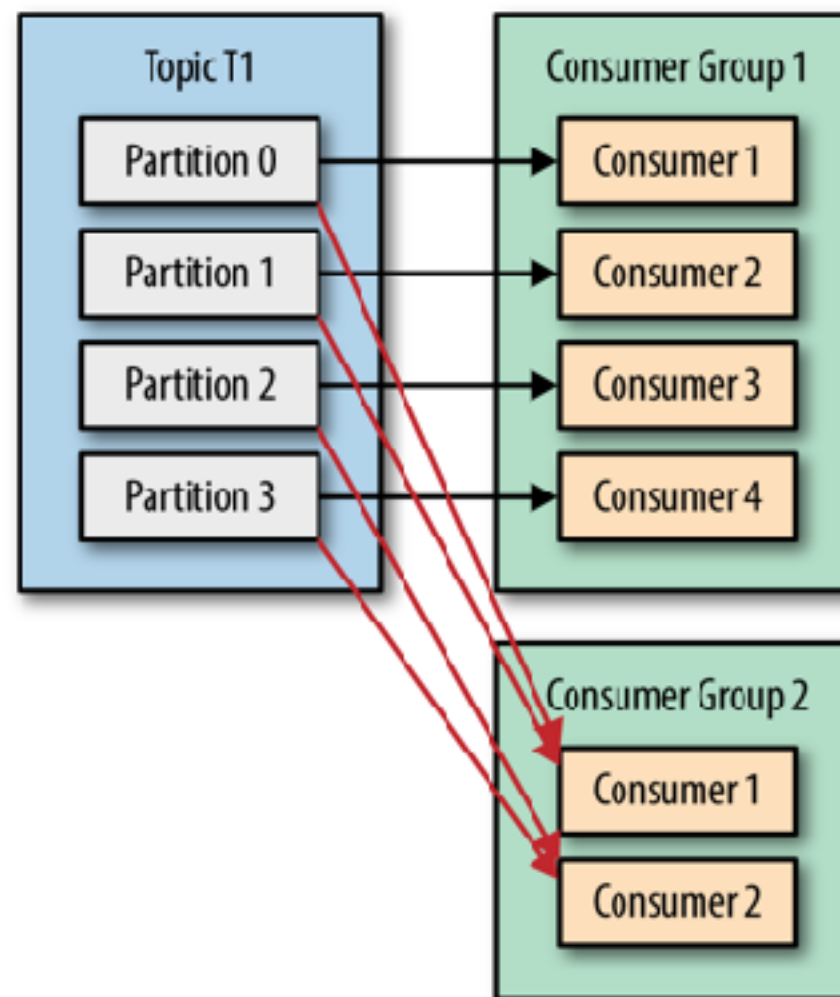




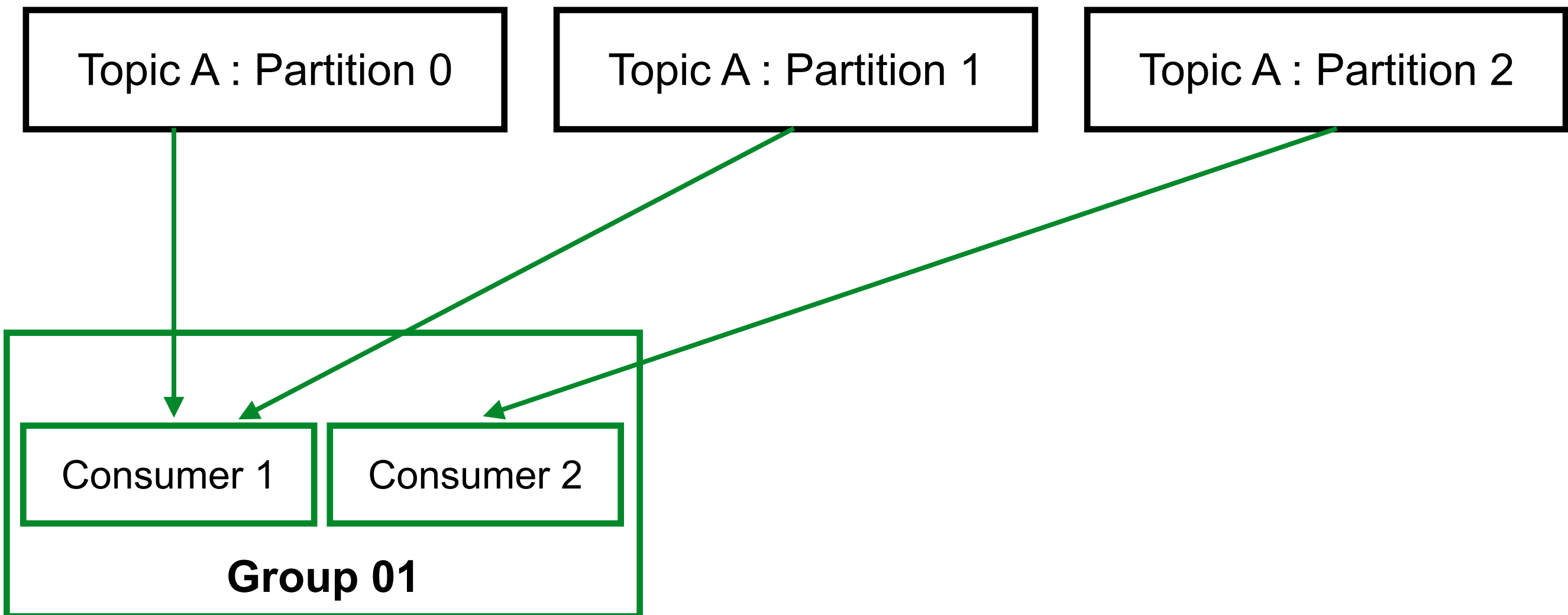
# Consumer groups

Consumers read data in consumer groups

Each consumer in a group read from exclusive partitions



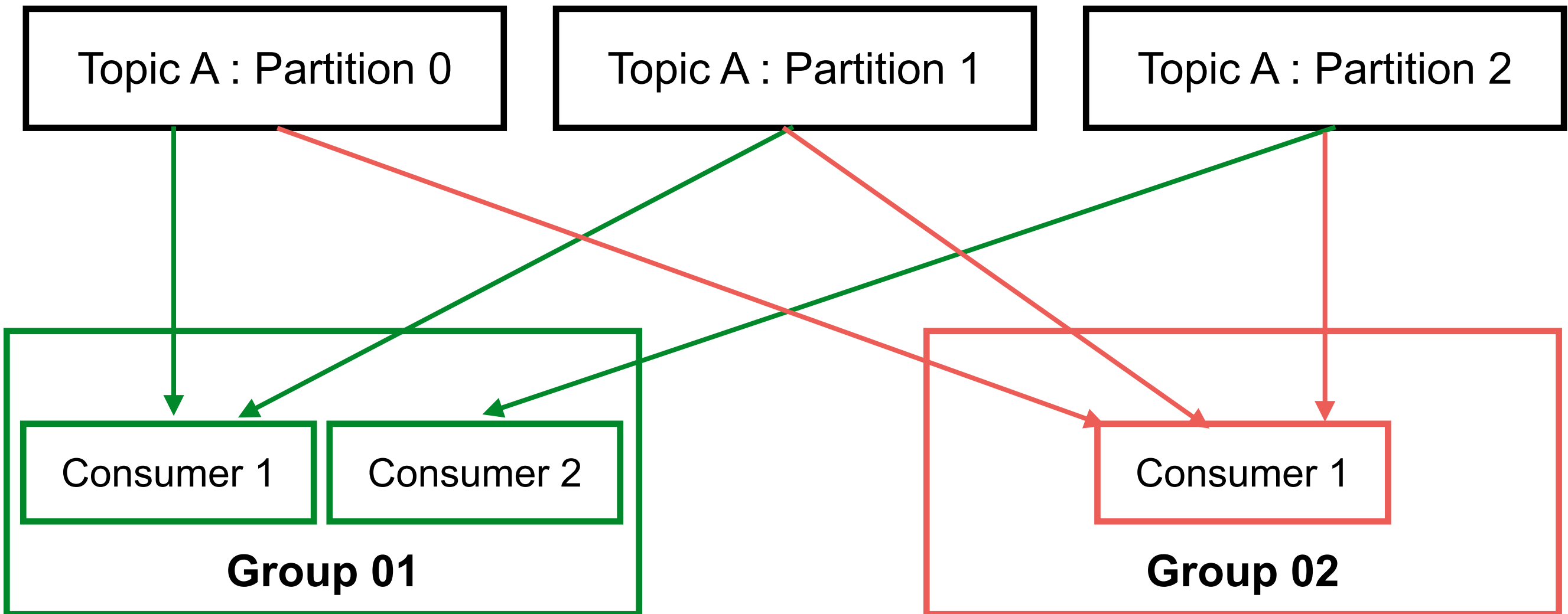
# Consumer groups



Consumer will automatically use a GroupCoordinator  
ConsumerCoordinator to assign a consumer to a partition.

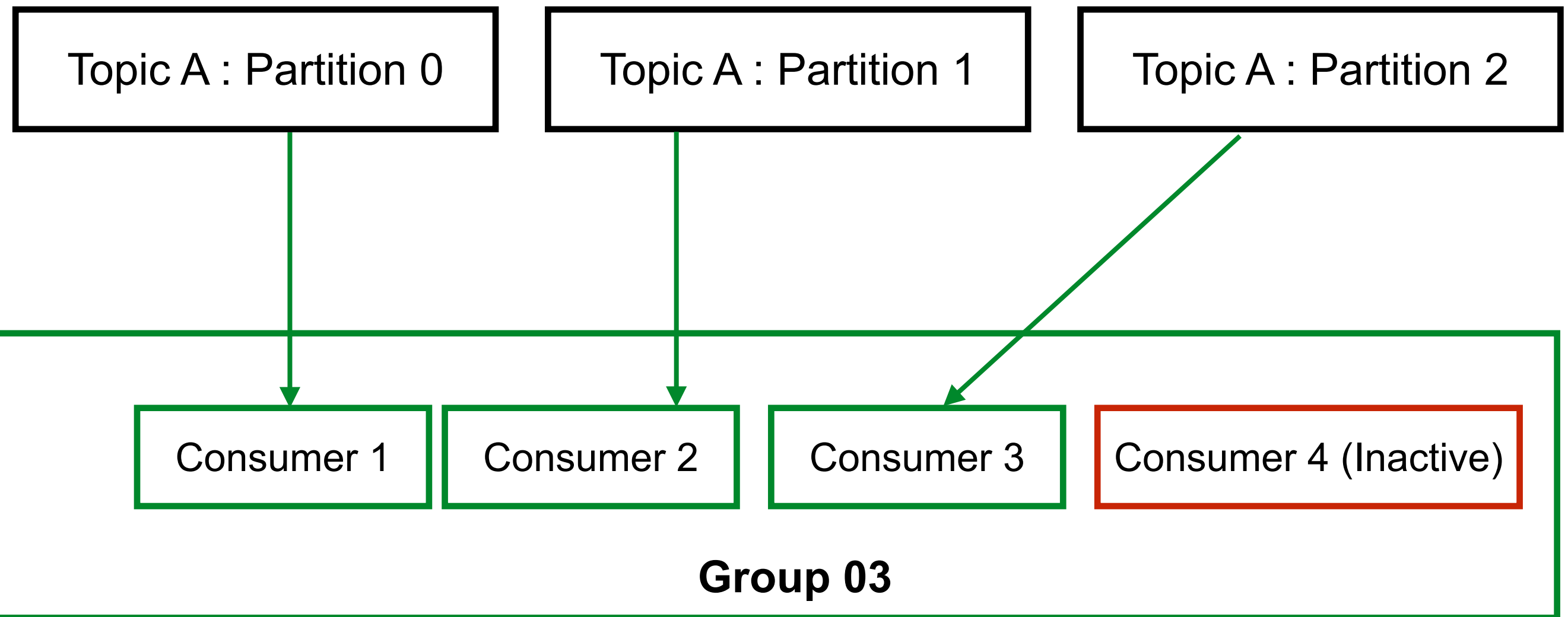


# Consumer groups

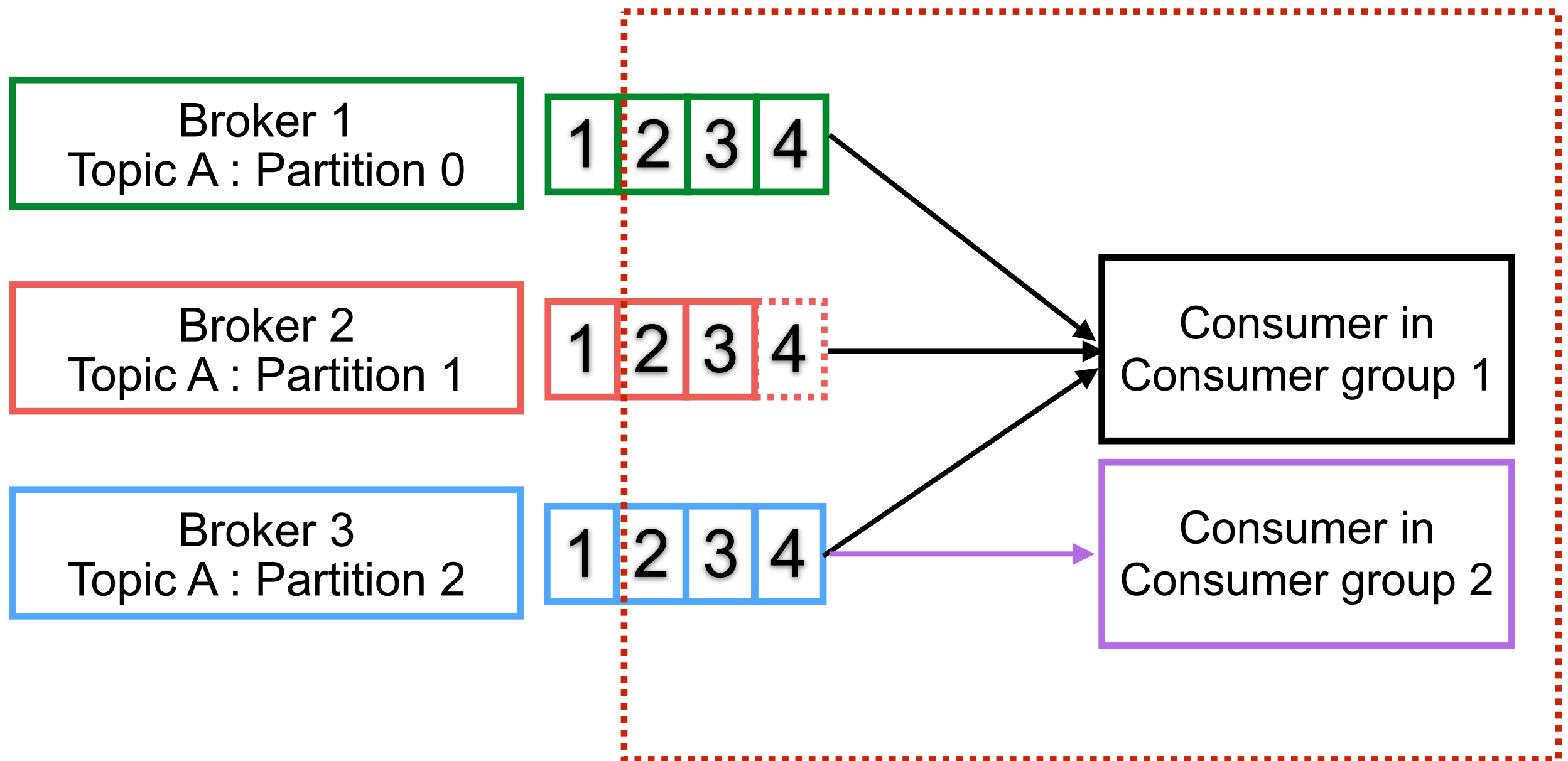


# IF Consumers > partitions ?

Some consumers will **inactive**



# Consumers offsets



# Consumers offsets

**Kafka** store the offset at which a consumer group has been reading

The offsets committed live in **topic** named “**\_\_consumer\_offsets**”

When consumer in a group has processed data received from Kafka,  
it should be **committing the offsets**



# When to commit the offset ?



# Delivery semantics for consumer

At most once

At least once (preferred)

Exactly once

<https://docs.confluent.io/kafka/design/delivery-semantics.html>





# 1. At most once

Offsets are committed as soon as the message is received

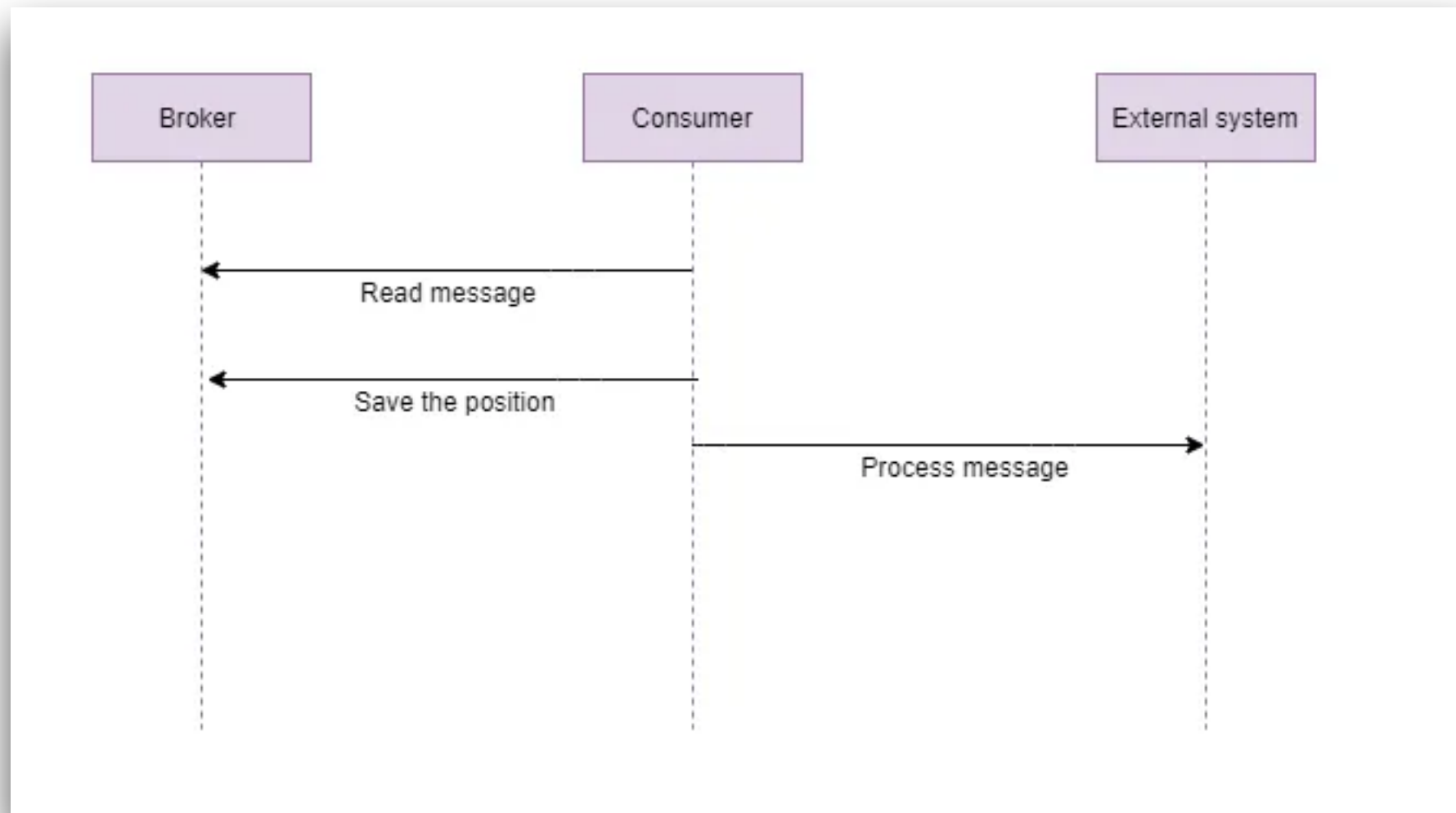
If processing go wrong, the message will be **loss!!**

**Fire and forgot pattern**

Lowest latency



# At most once



## 2. At lease once (1)

Offsets are committed after the message is processed

Messages are never lost but may be **redelivered**

If processing go wrong, the message will be **read again**



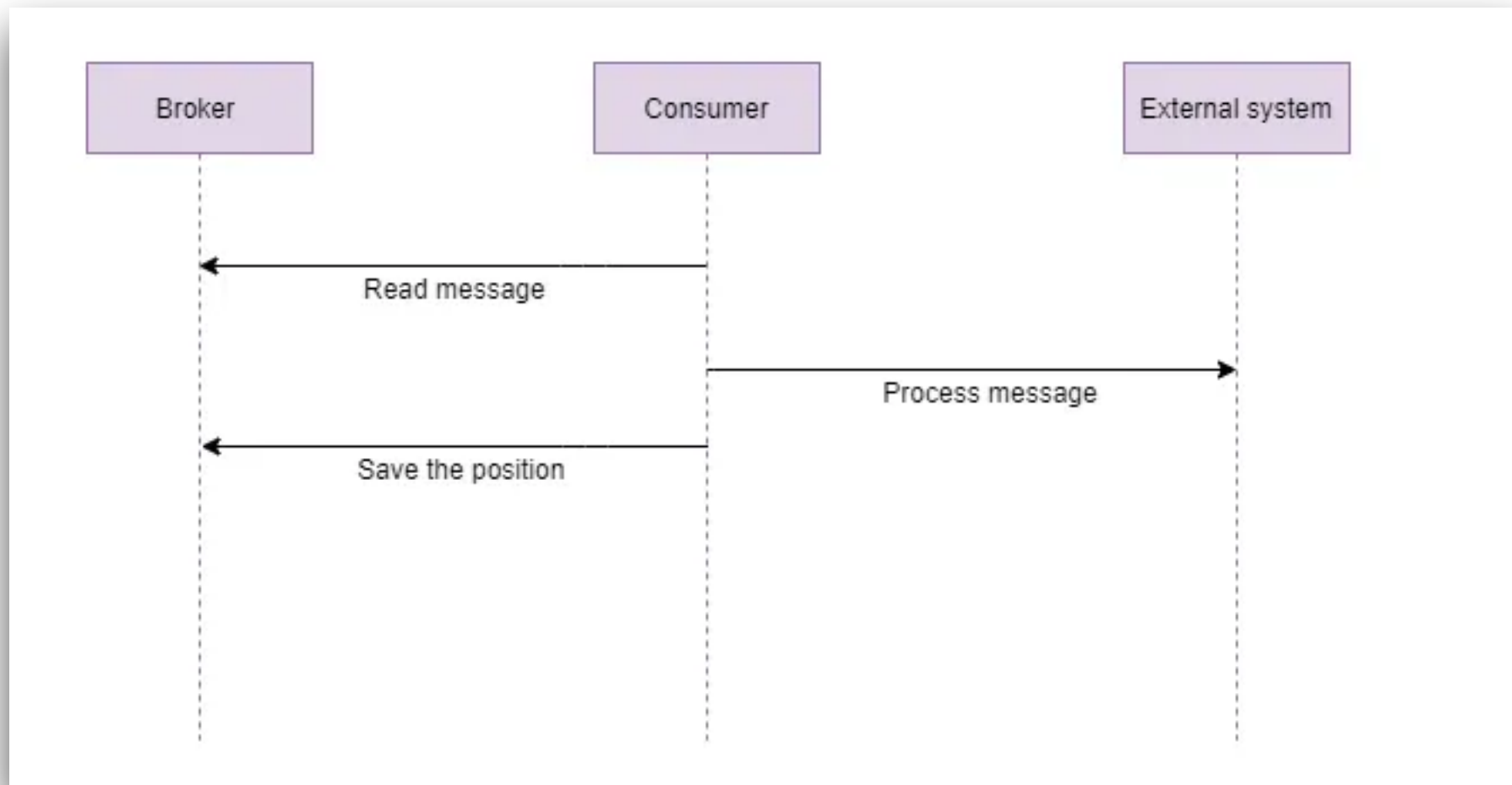
## 2. At lease once (2)

Make sure your processing is **idempotent**

*Processing again the message  
not impact to your system !!*



# At lease once

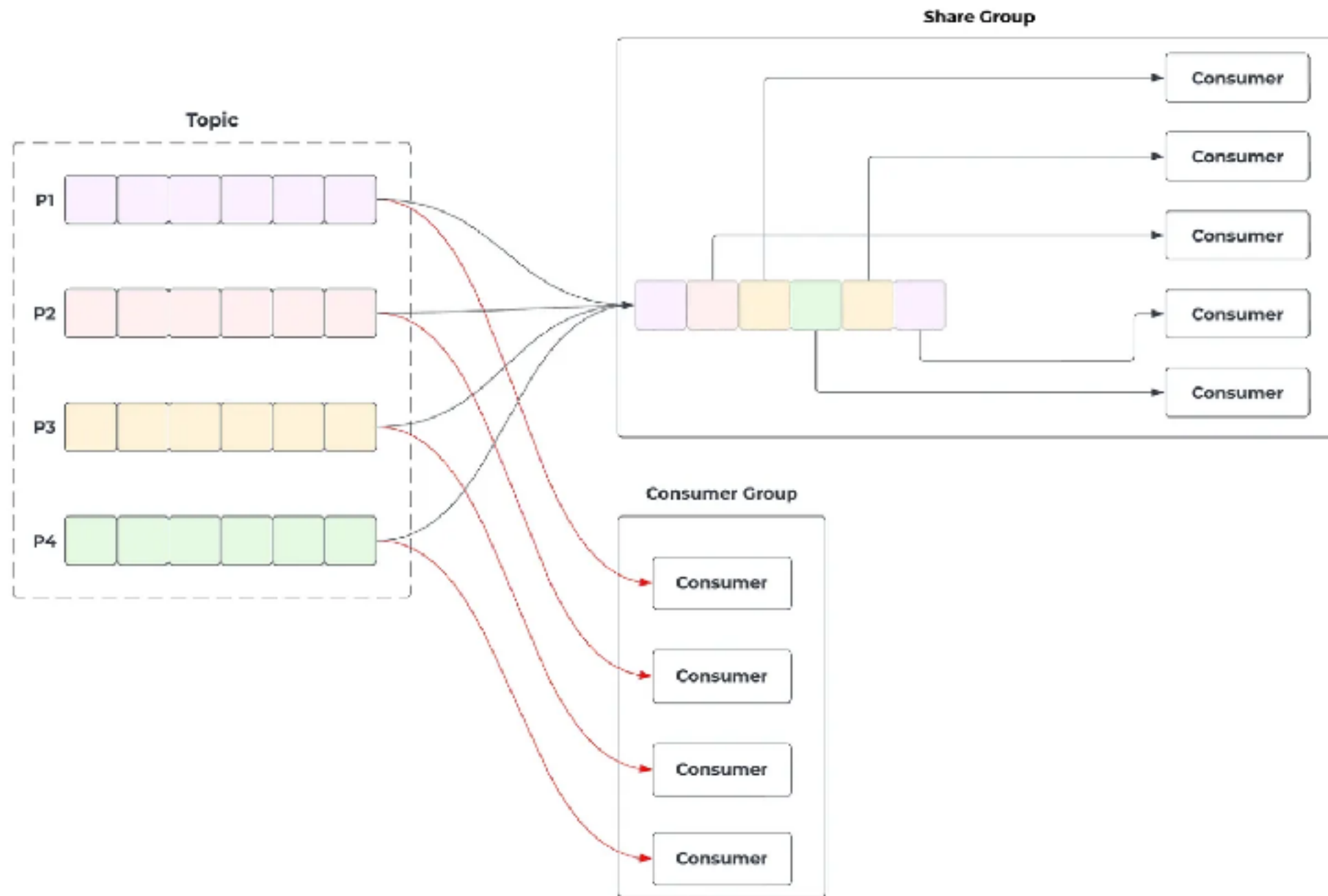


# 3. Exactly once

Each message is delivered once and only once  
Can be achieved for Kafka (Workflow, Stream API)  
Transactional delivery  
Resent with **idempotent**



# Share Group in Kafka 4.0



<https://cwiki.apache.org/confluence/display/KAFKA/KIP-932%3A+Queues+for+Kafka>

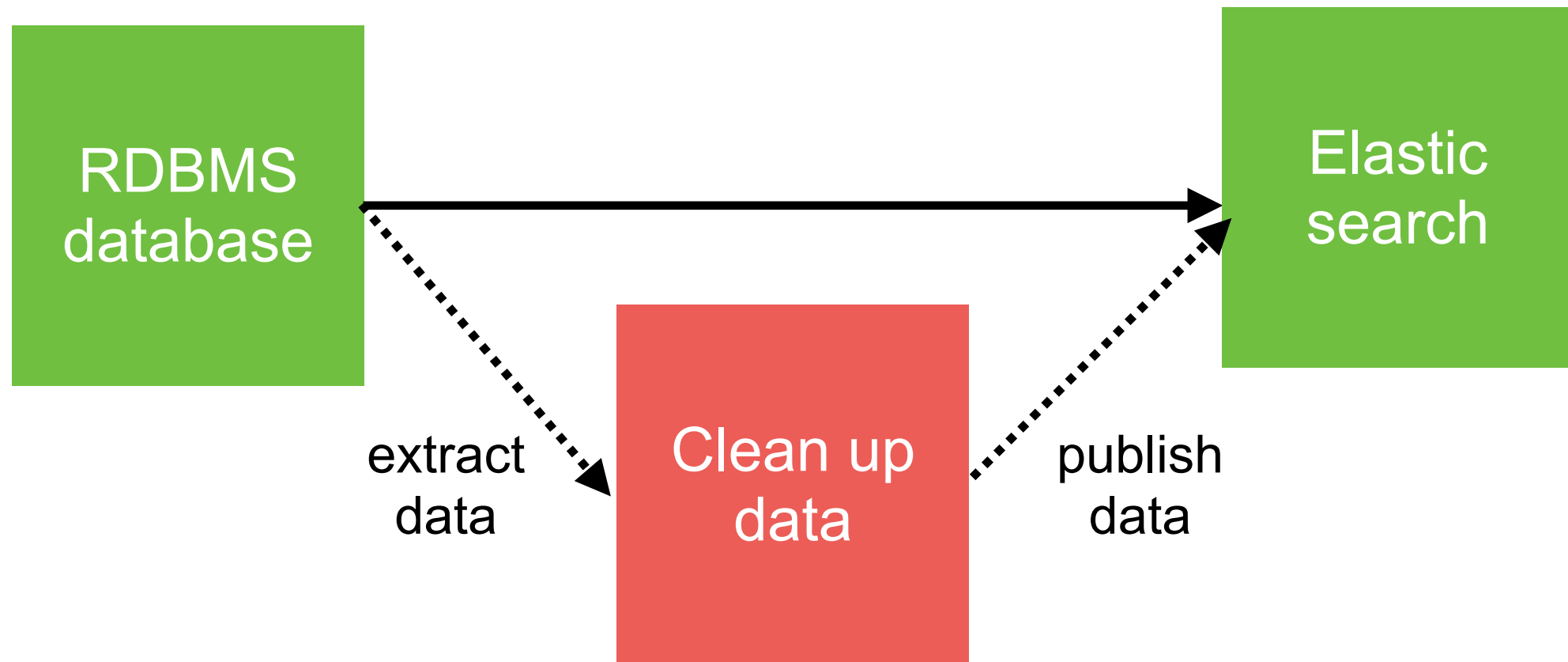


# Data Pipeline





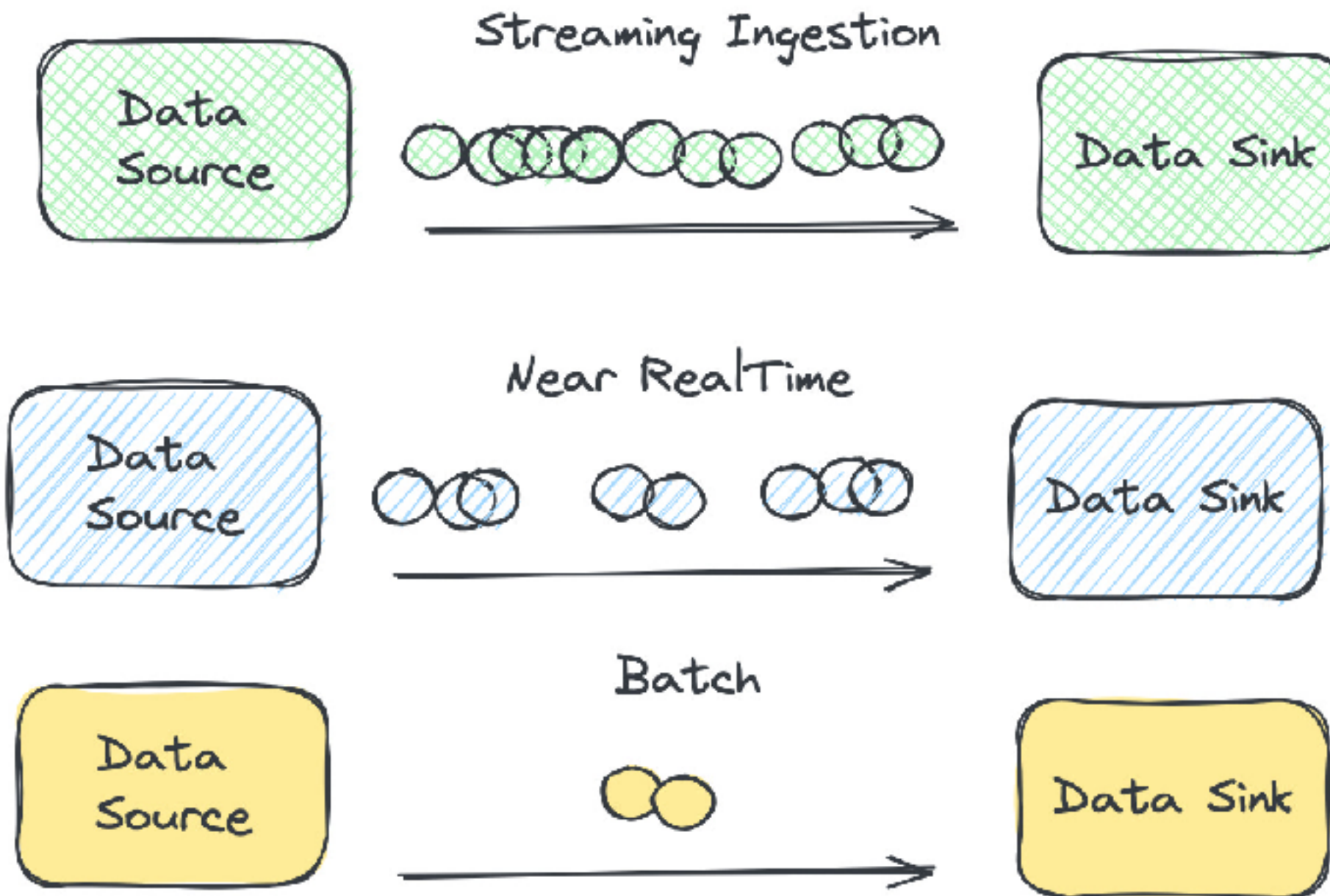
# Batching process



Schedule task run every 2 am



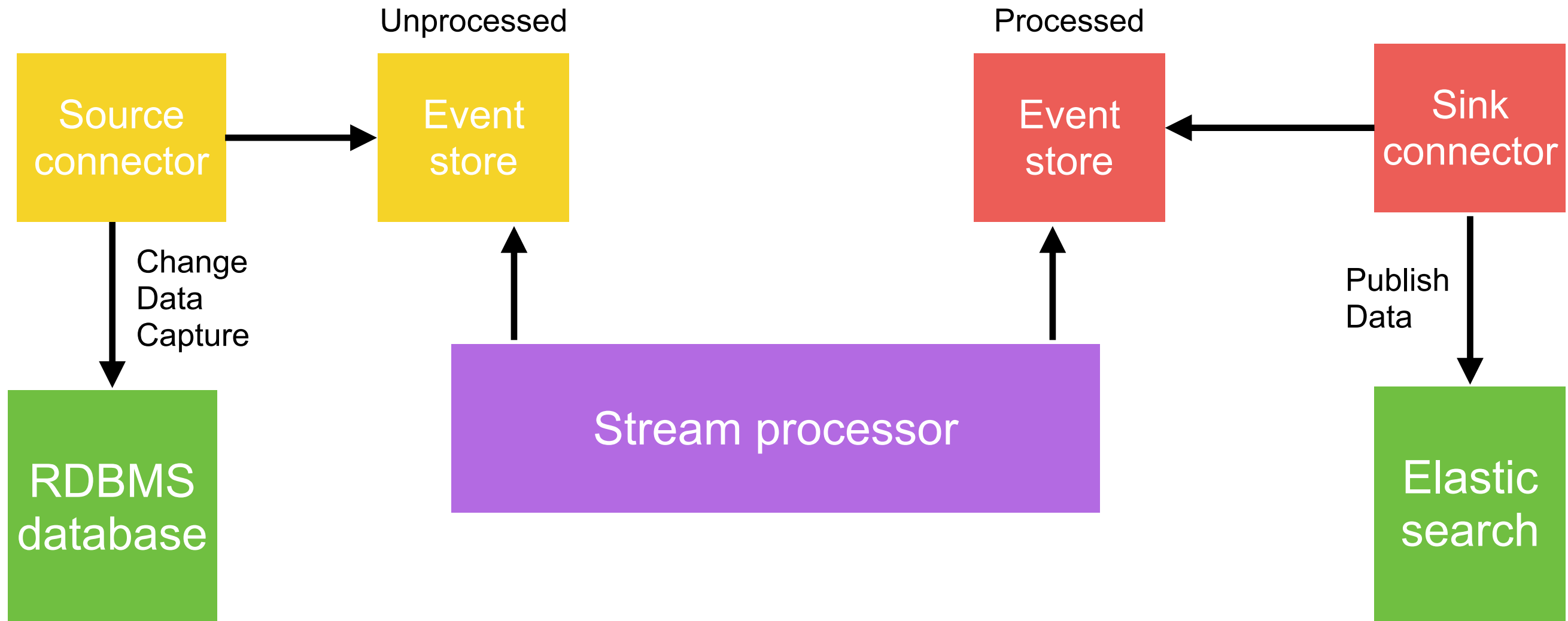
# Process !!



<https://dataengineeringcentral.substack.com/p/batch-vs-near-realtime-vs-streaming>



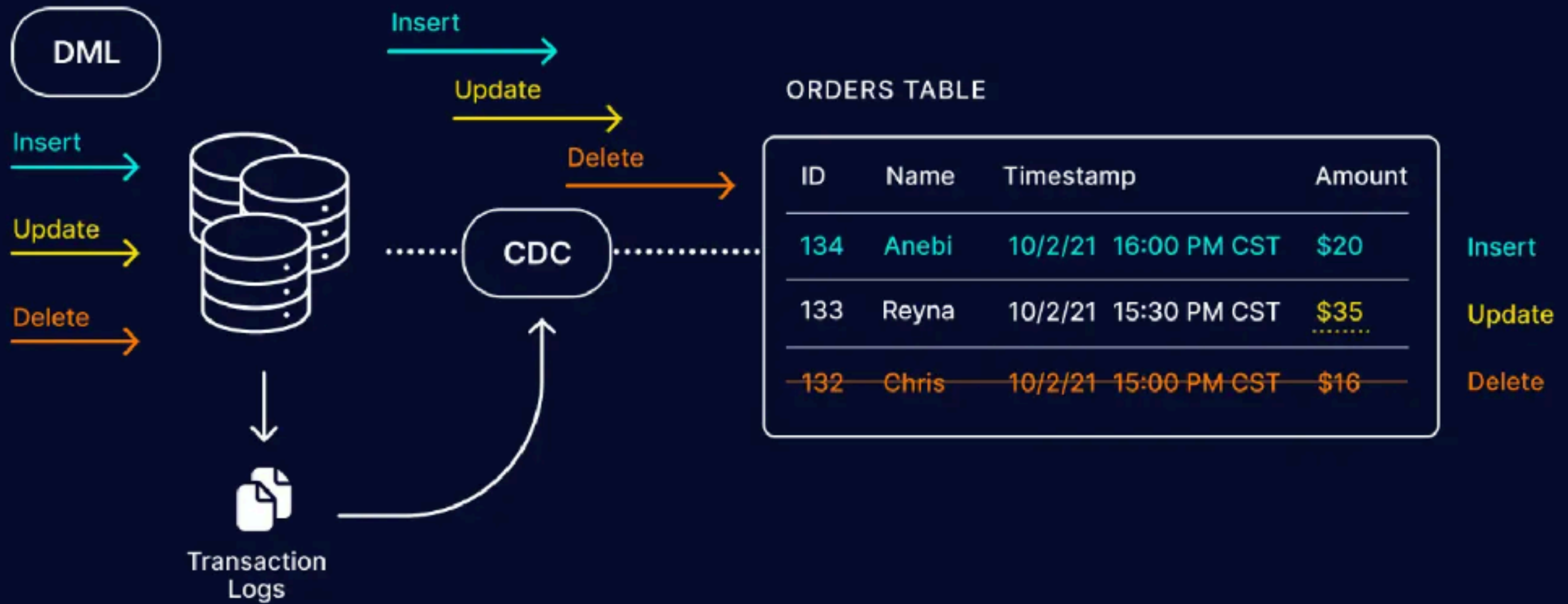
# Streaming data pipeline



Real-time(NRT) process when data changed



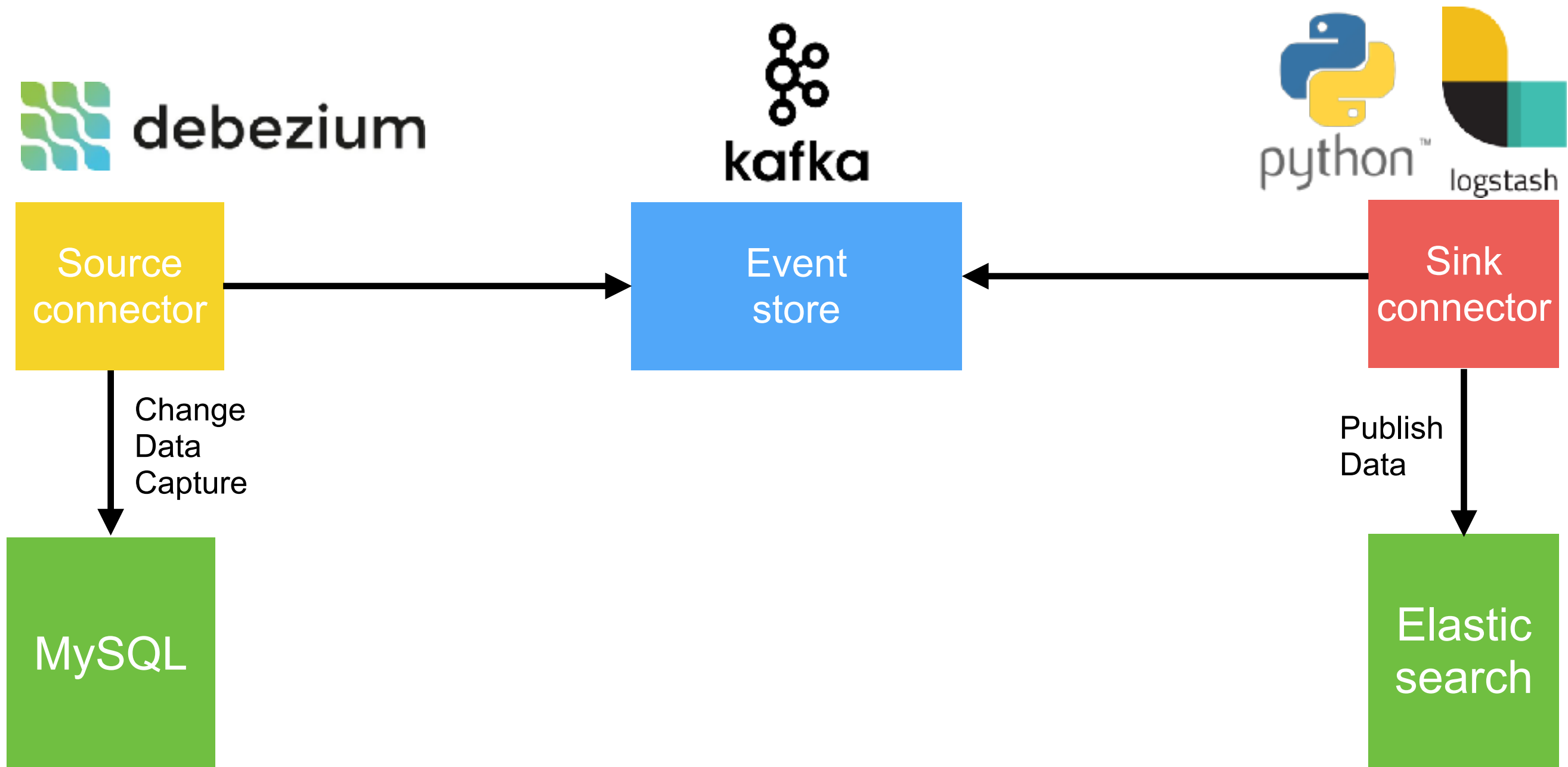
# Change Data Capture (CDC)



<https://www.striim.com/blog/change-data-capture-cdc-what-it-is-and-how-it-works/>



# Example



<https://github.com/up1/workshop-kafka-2025/tree/main/kafka-cdc>



# Q/A

