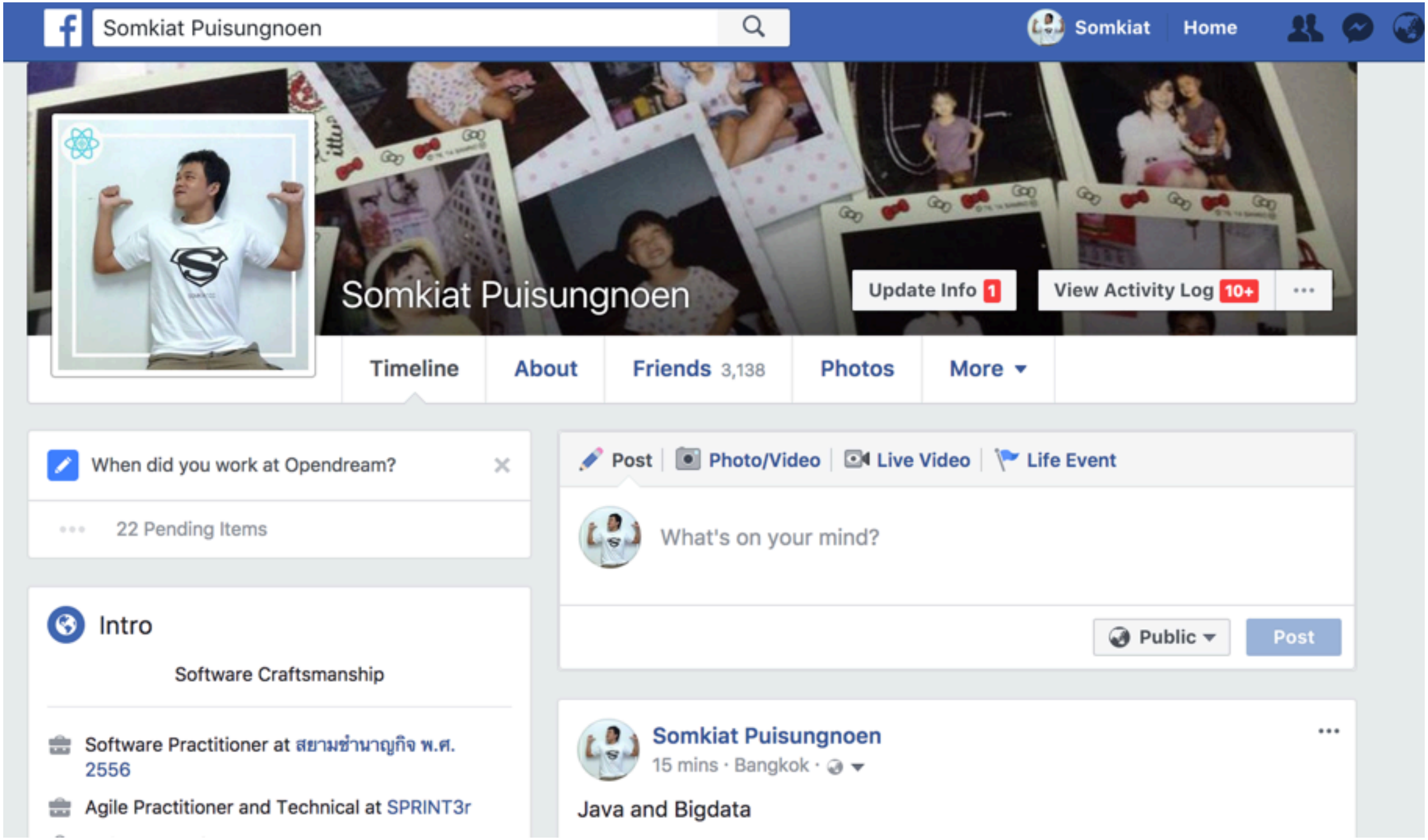
Apache Kafka
Kibana
Data pipeline

สยามชำนาญกิจ

SIAM CHAMNAN KIT

SHU HA RI

SOFTWARE DEVELOPMENT SCHOOL



Somkiat Puisungnoen

Somkiat Puisungnoen

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 สยามชำนาญกิจ พ.ศ. 2556

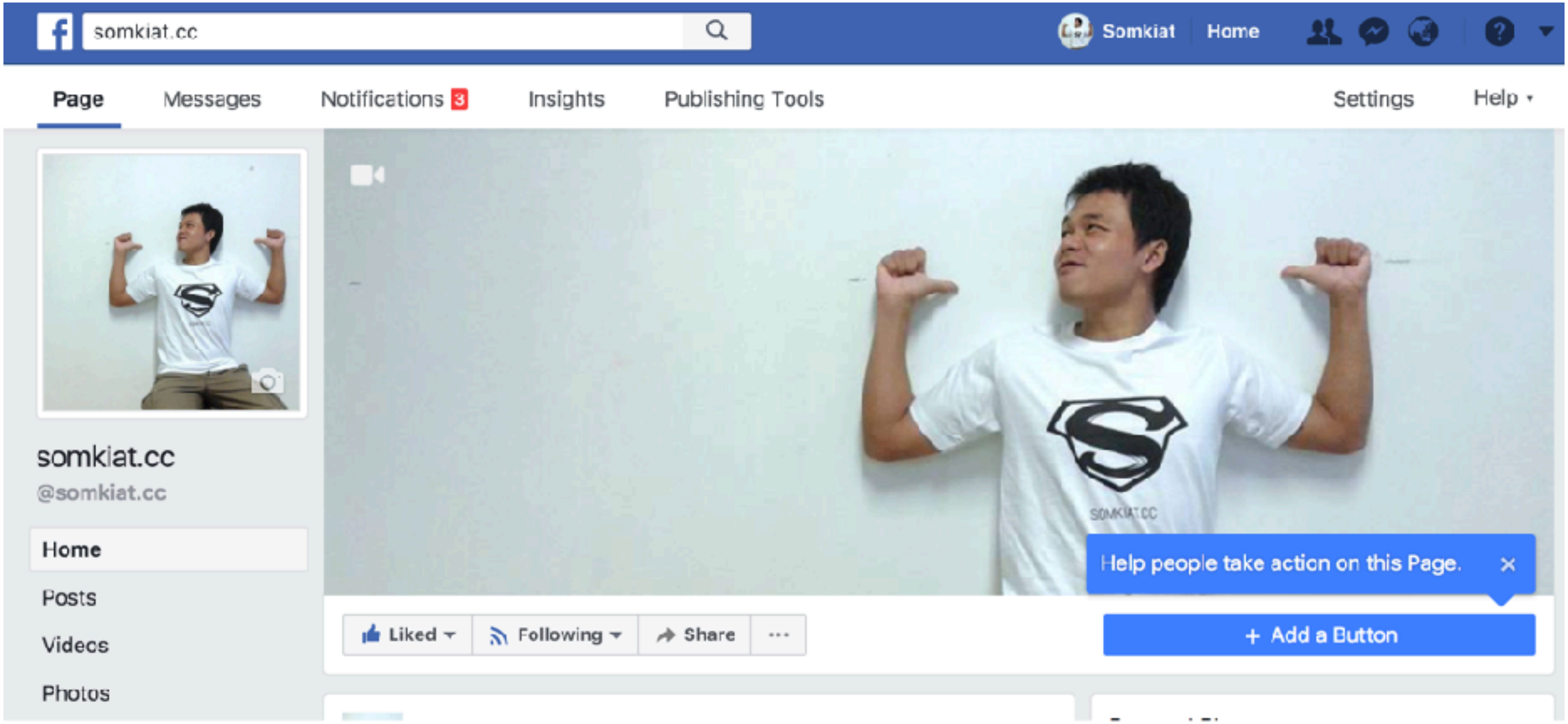
Agile Practitioner and Technical at SPRINT3r

What's on your mind?

Somkiat Puisungnoen

15 mins · Bangkok ·

Java and Bigdata

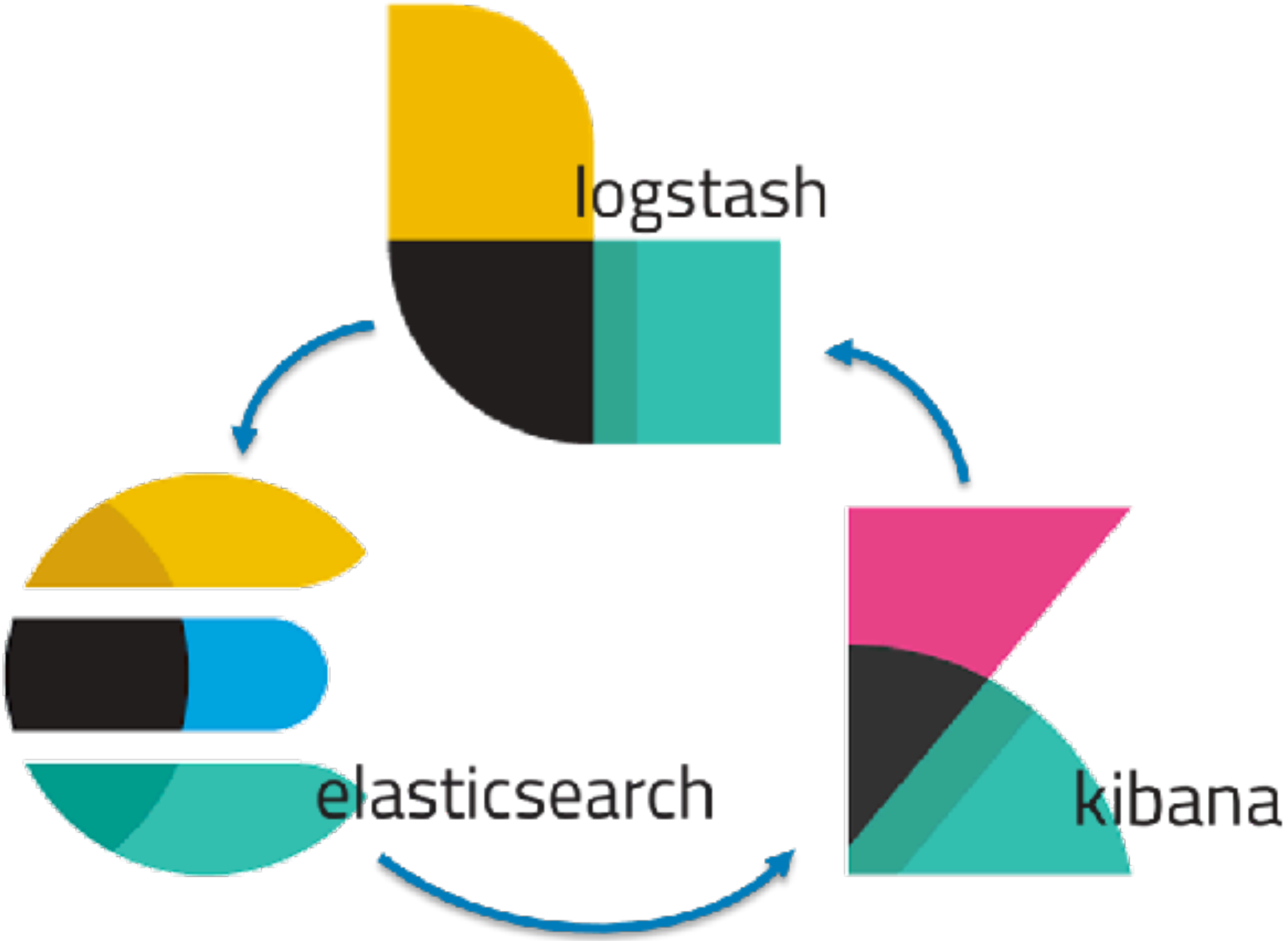


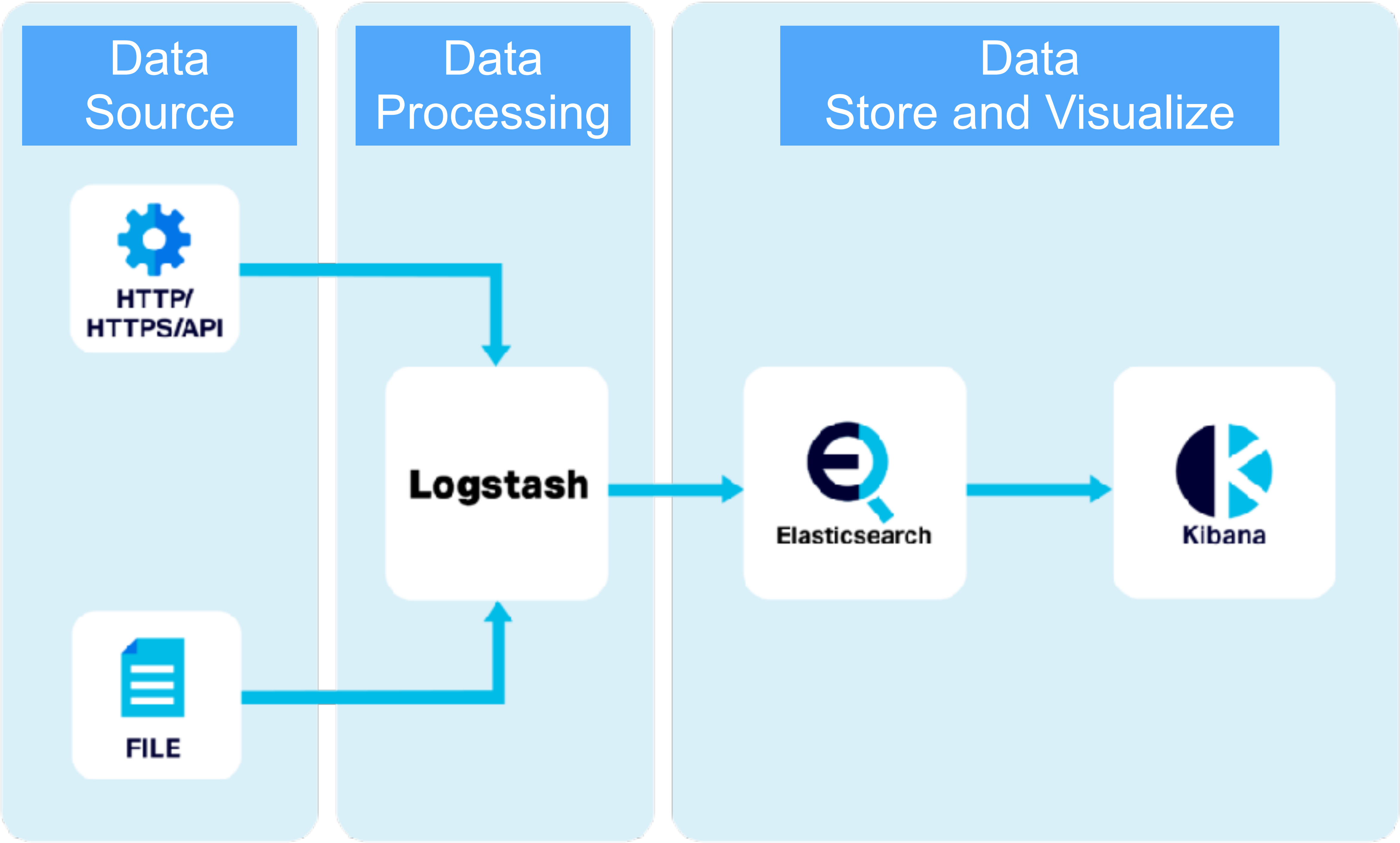
somkiat.cc
Somkiat
Home
Page
Messages
Notifications 3
Insights
Publishing Tools
Settings
Help
somkiat.cc
@somkiat.cc
Home
Posts
Videos
Photos
SOMKIAT.CC
Help people take action on this Page.
Liked
Following
Share
+ Add a Button

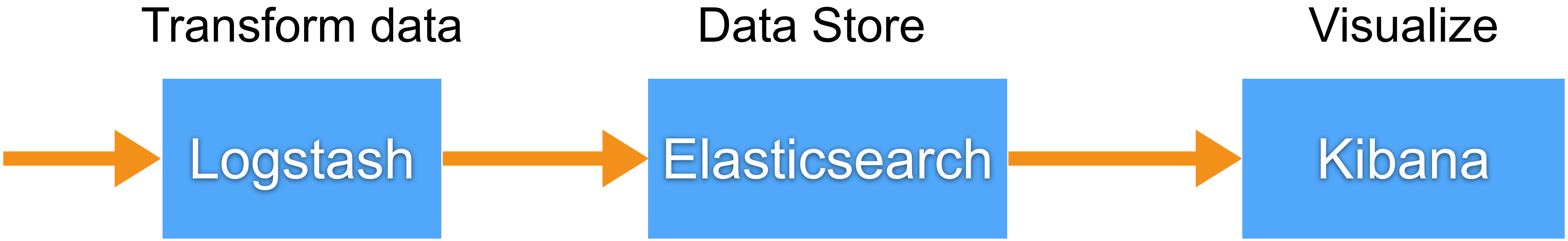
Agenda

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

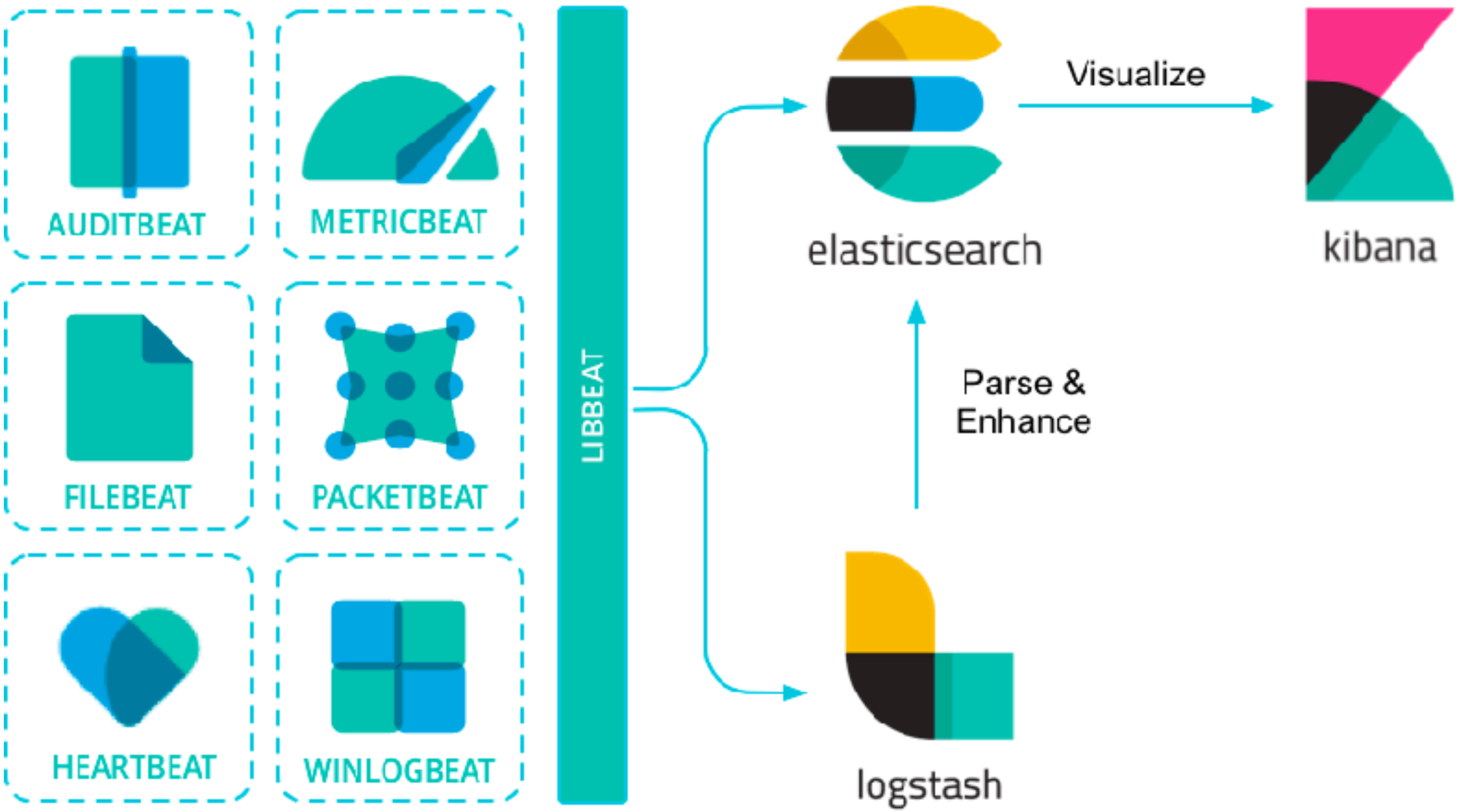
ELK stack




Data
Source
Data
Processing
Data
Store and Visualize
HTTP/
HTTPS/API
FILE
Logstash
Elasticsearch
Kibana

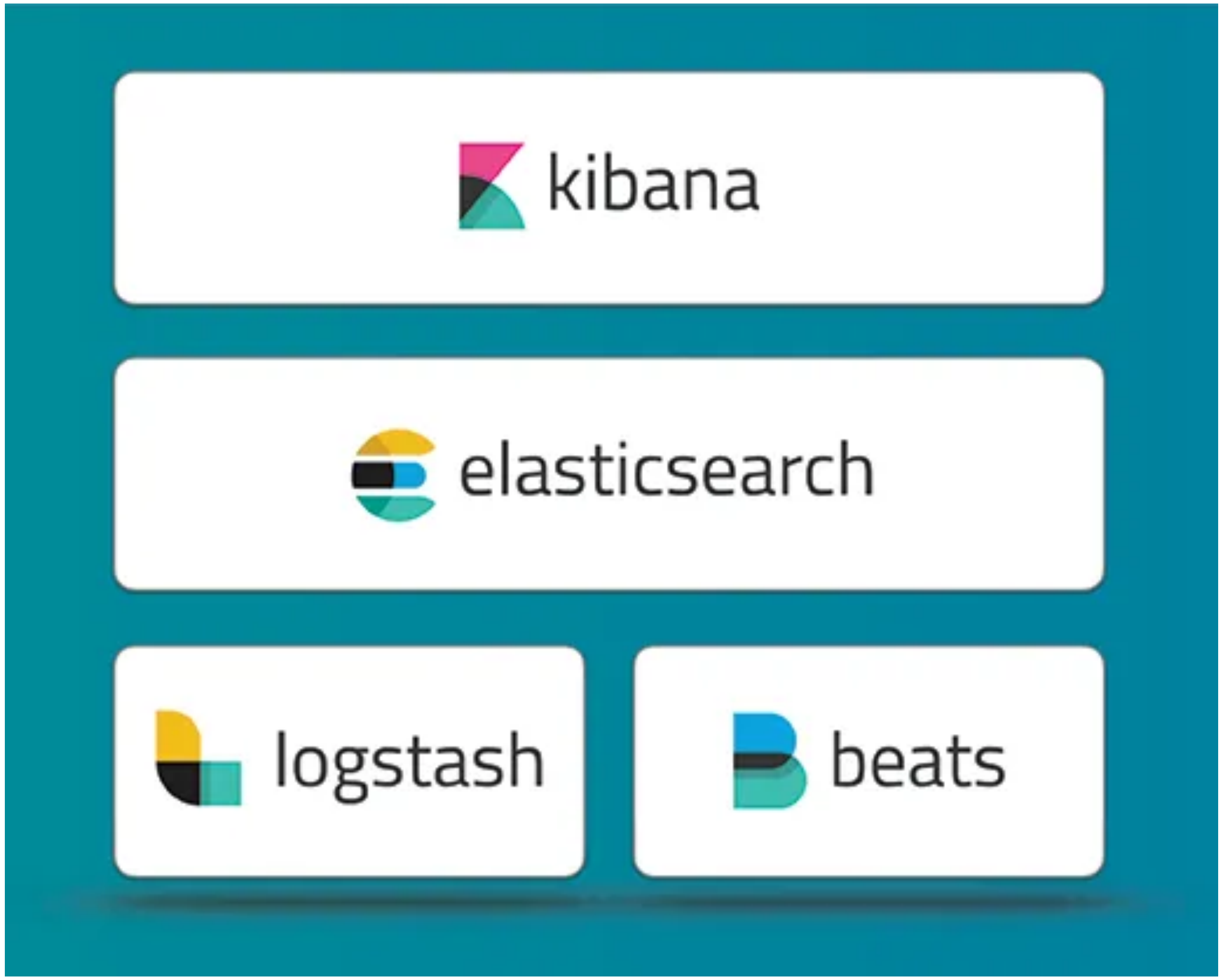

Transform data
Data Store
Visualize
Logstash
Elasticsearch
Kibana

Beat



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

ELK stack



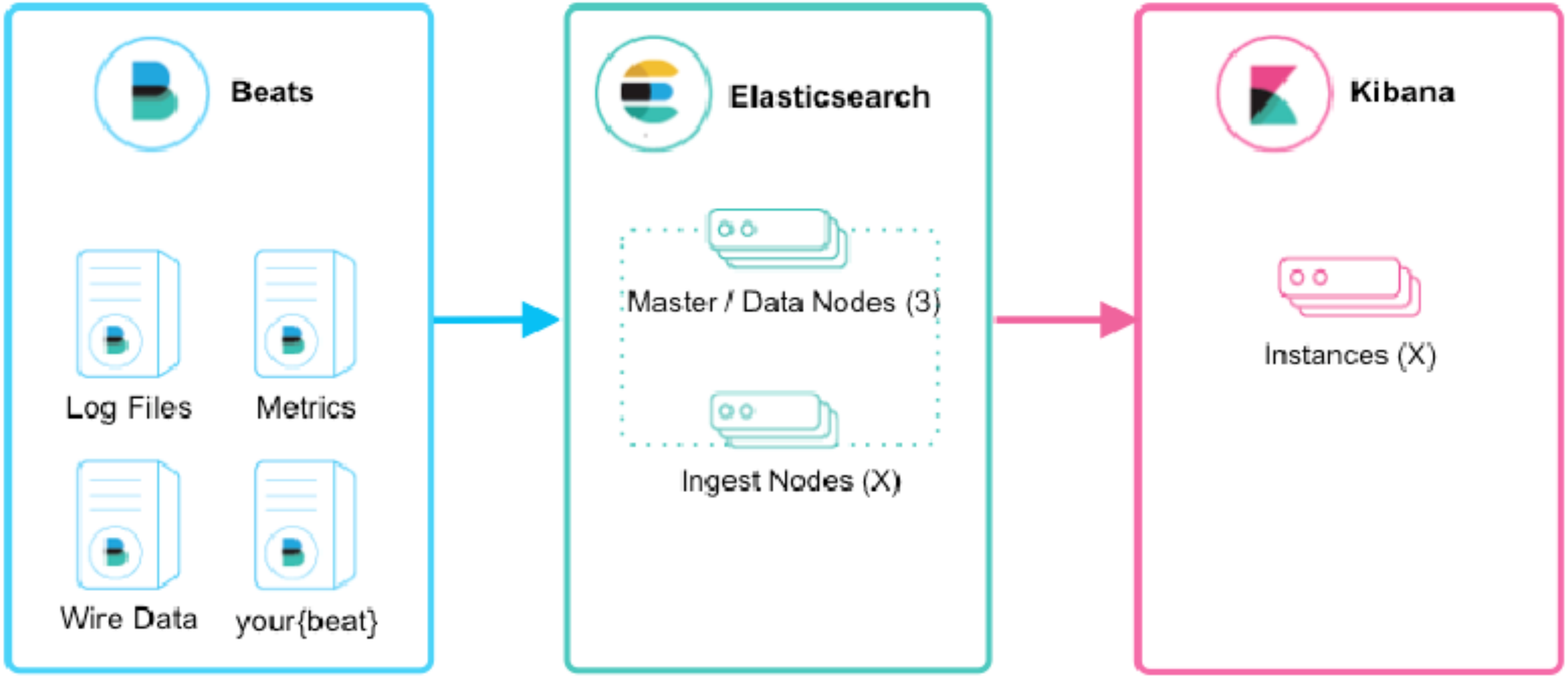
Beat

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

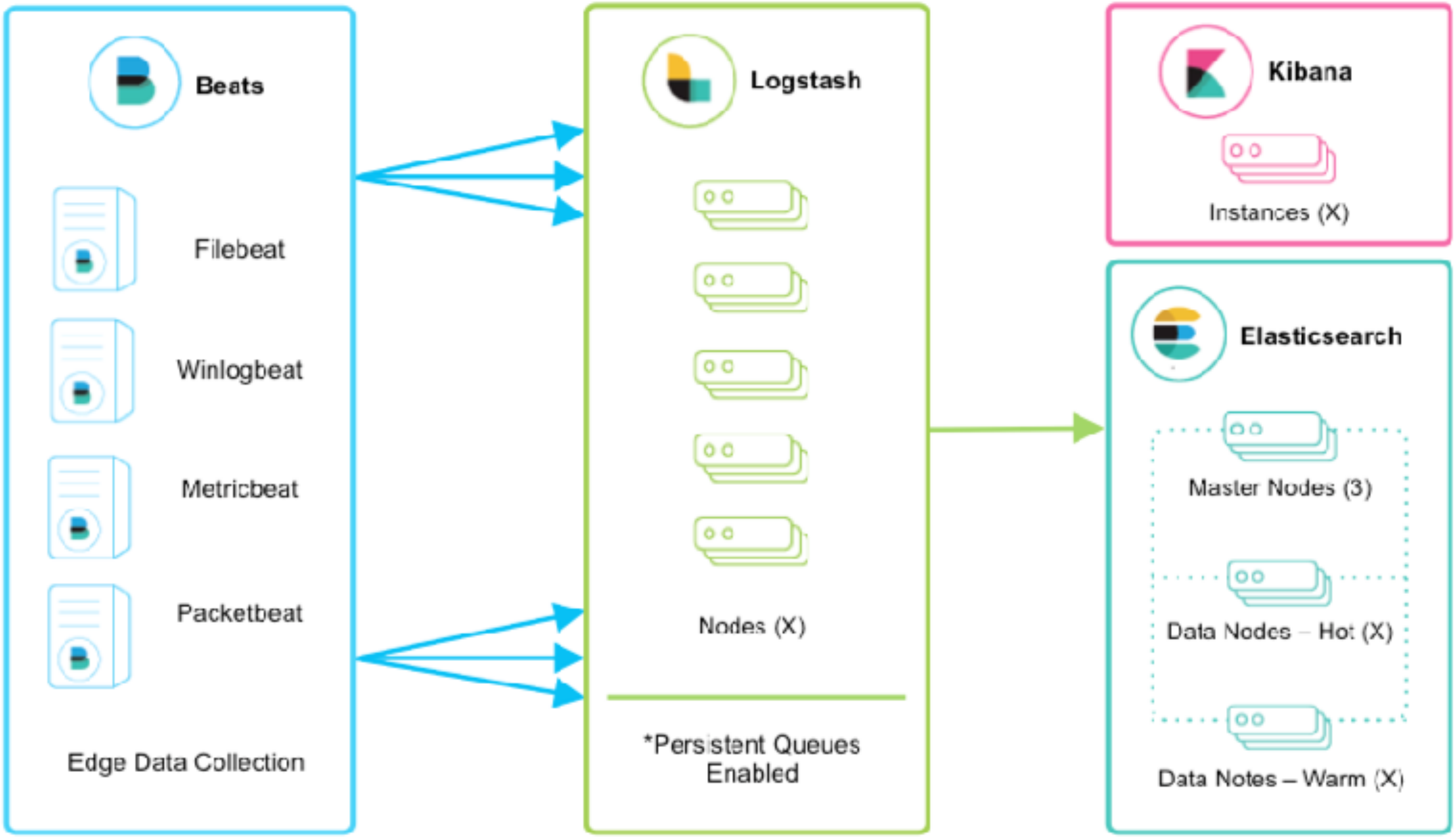
Scaling ELK Stack

Basic



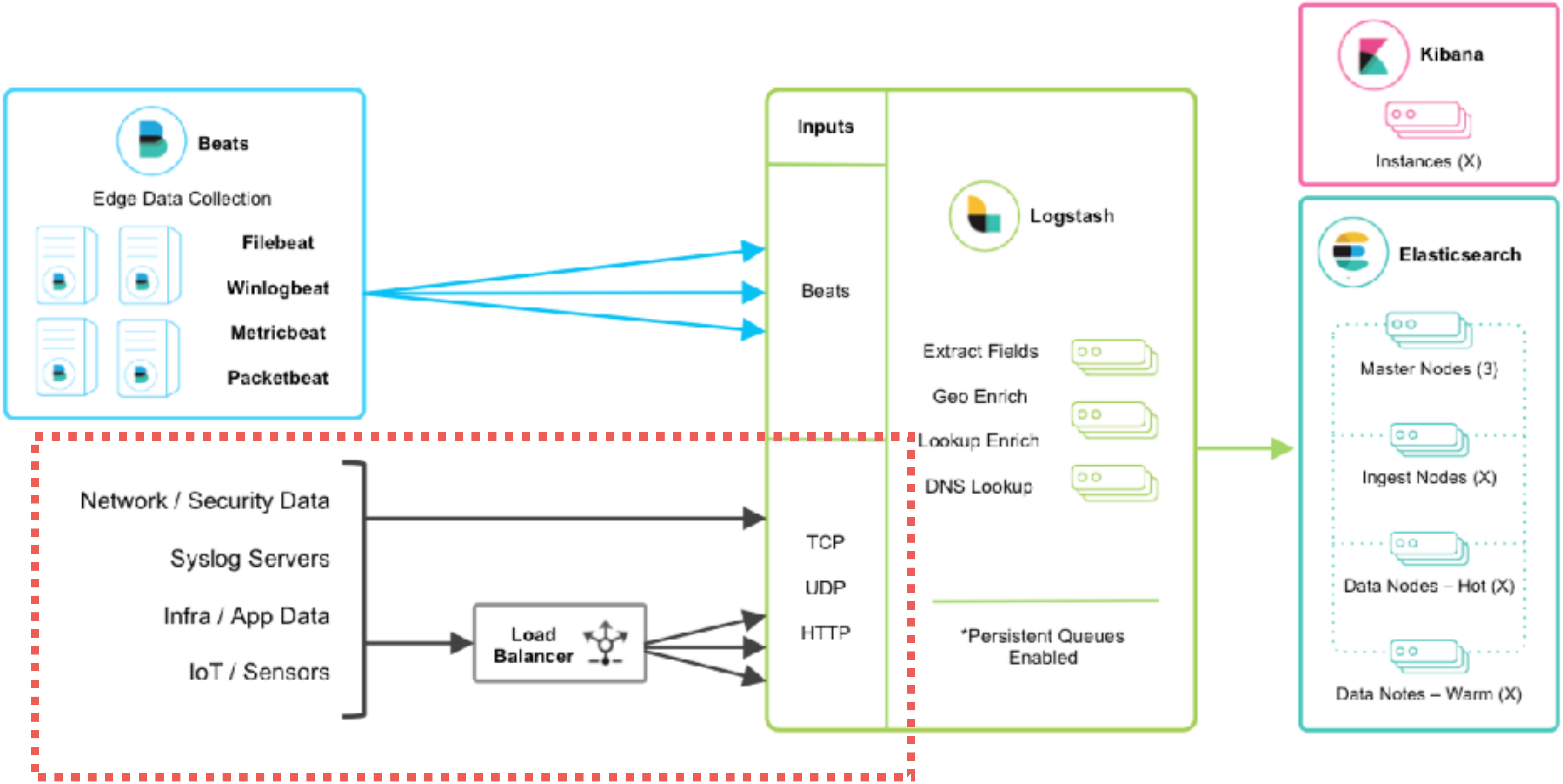
https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash

Scaling Ingest (Add more nodes)



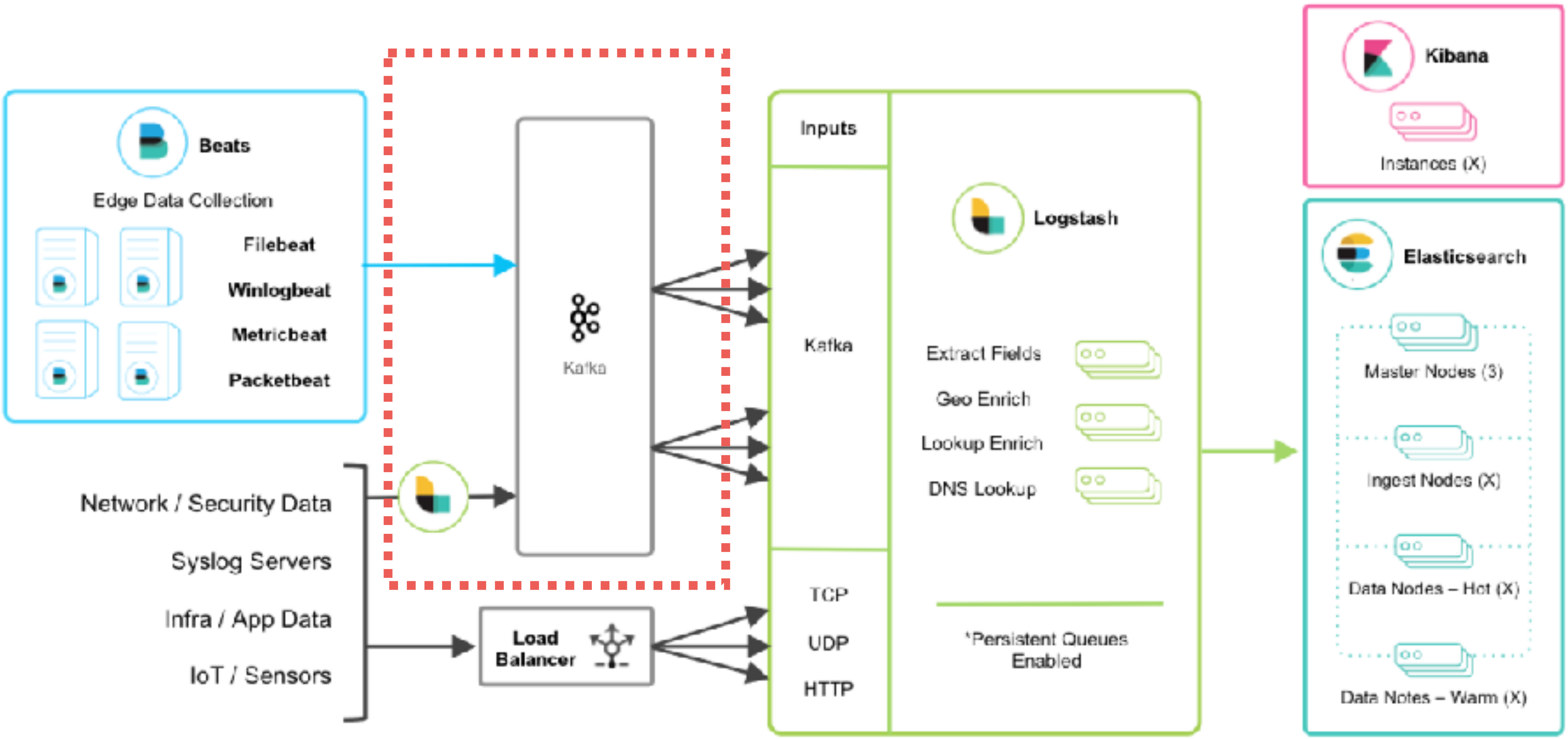
https://www.elastic.co/docs/reference/logstash/deploying-scaling-logstash

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 !!

RabbitMQ

Cloud Pub/Sub

kafka

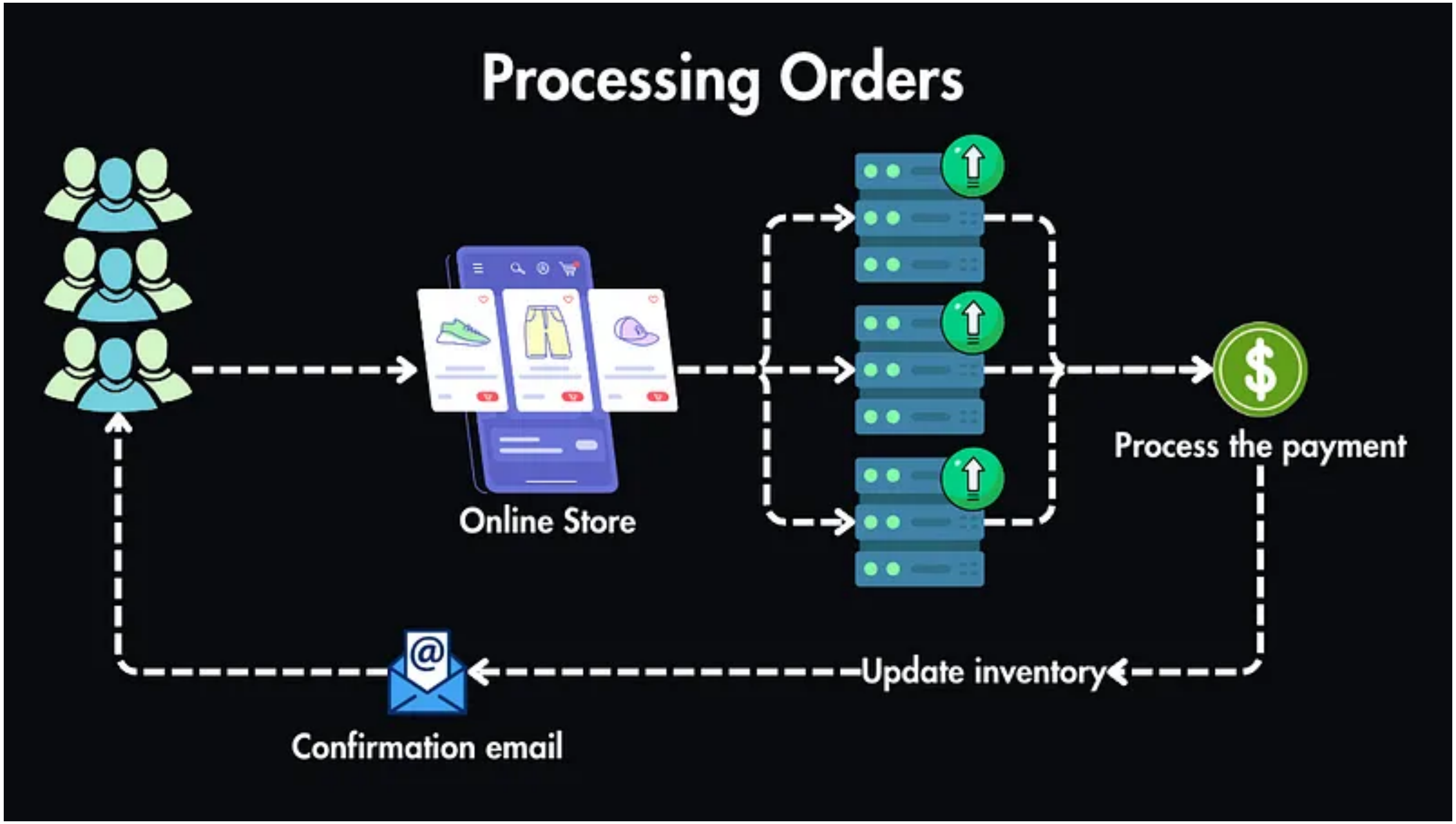
redis

amazon
SQS

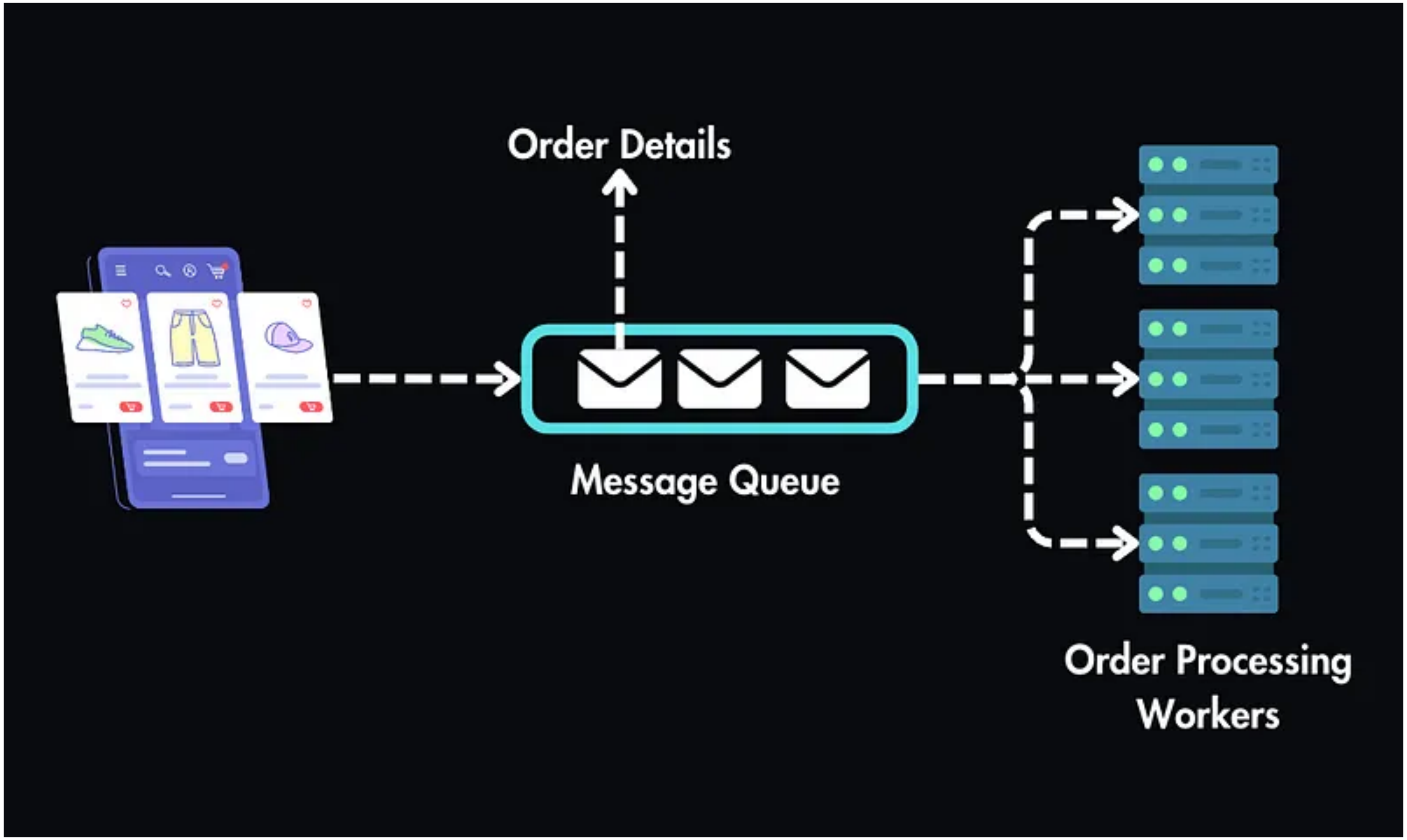
Why need Messaging ?

Decoupling between systems
Buffering
Back pressure problem

Processing Orders

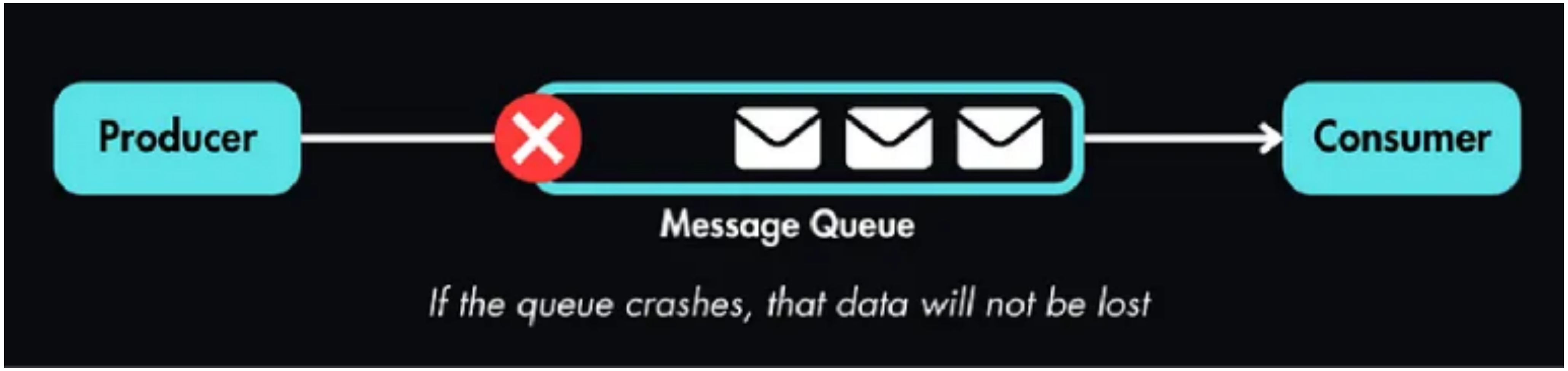


Order processing with messaging



Durable

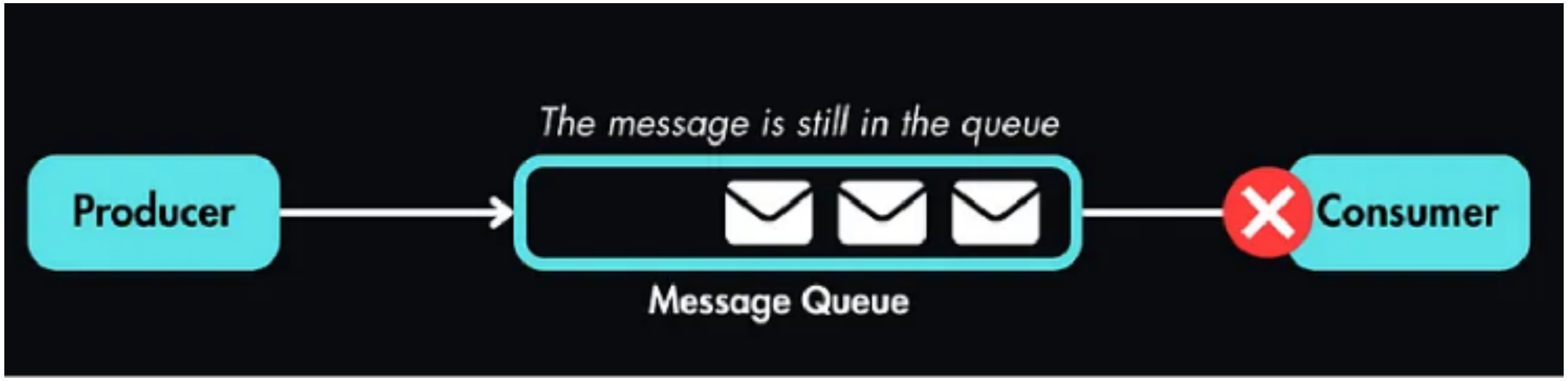
If the queue crashed, that data will not lost



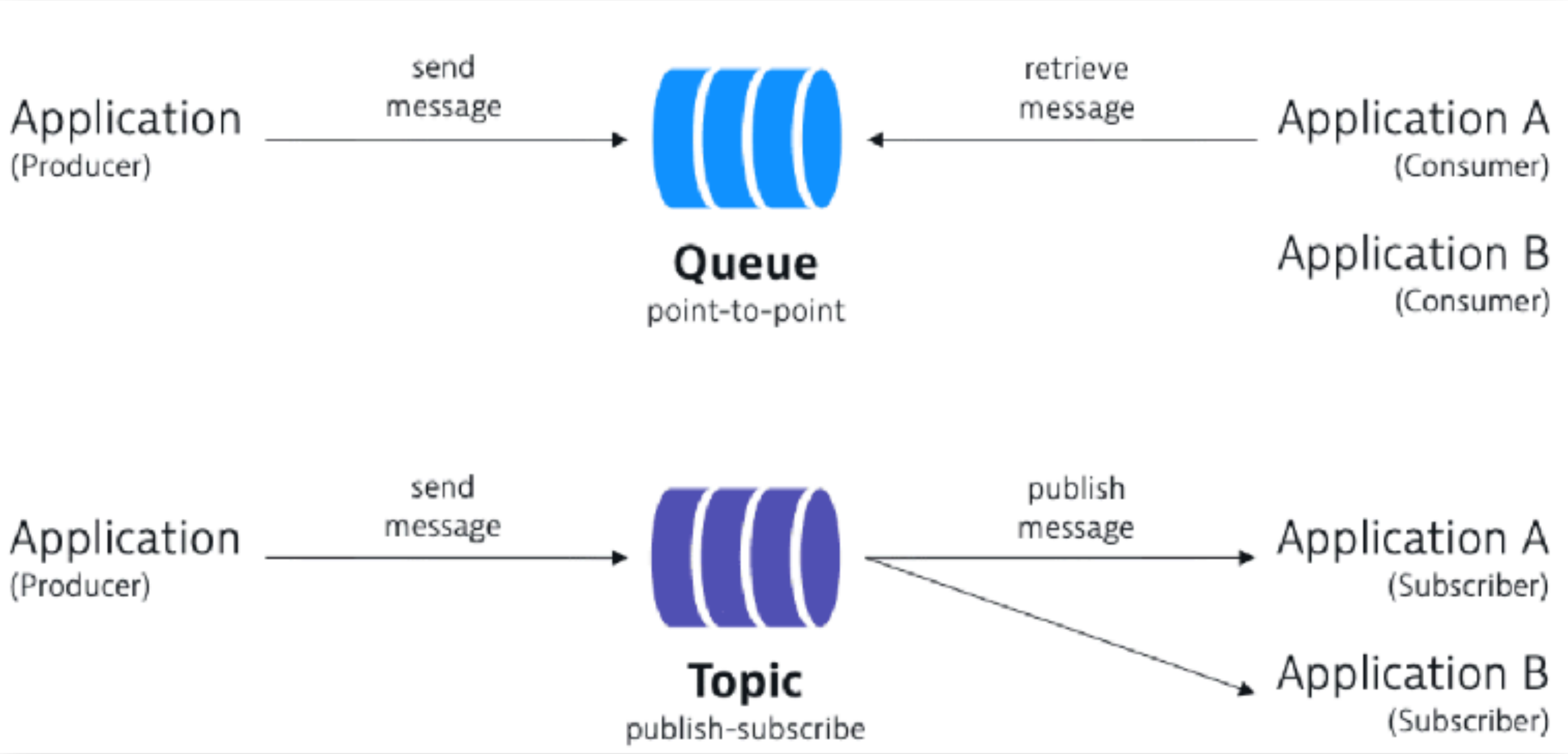
If the queue crashes, that data will not be lost

Reliable

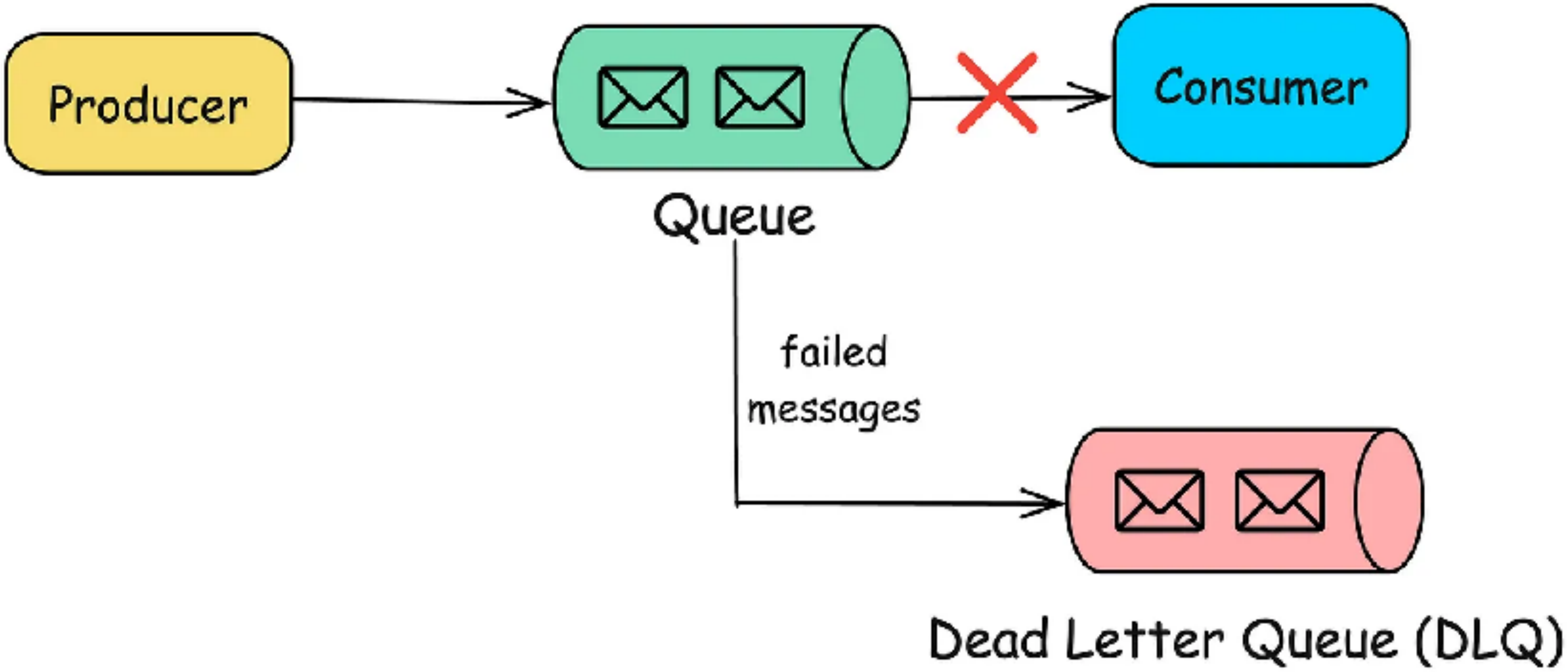
If consumer/worker failed, message still in queue



Queue vs Topic



Dead Letter Queue



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

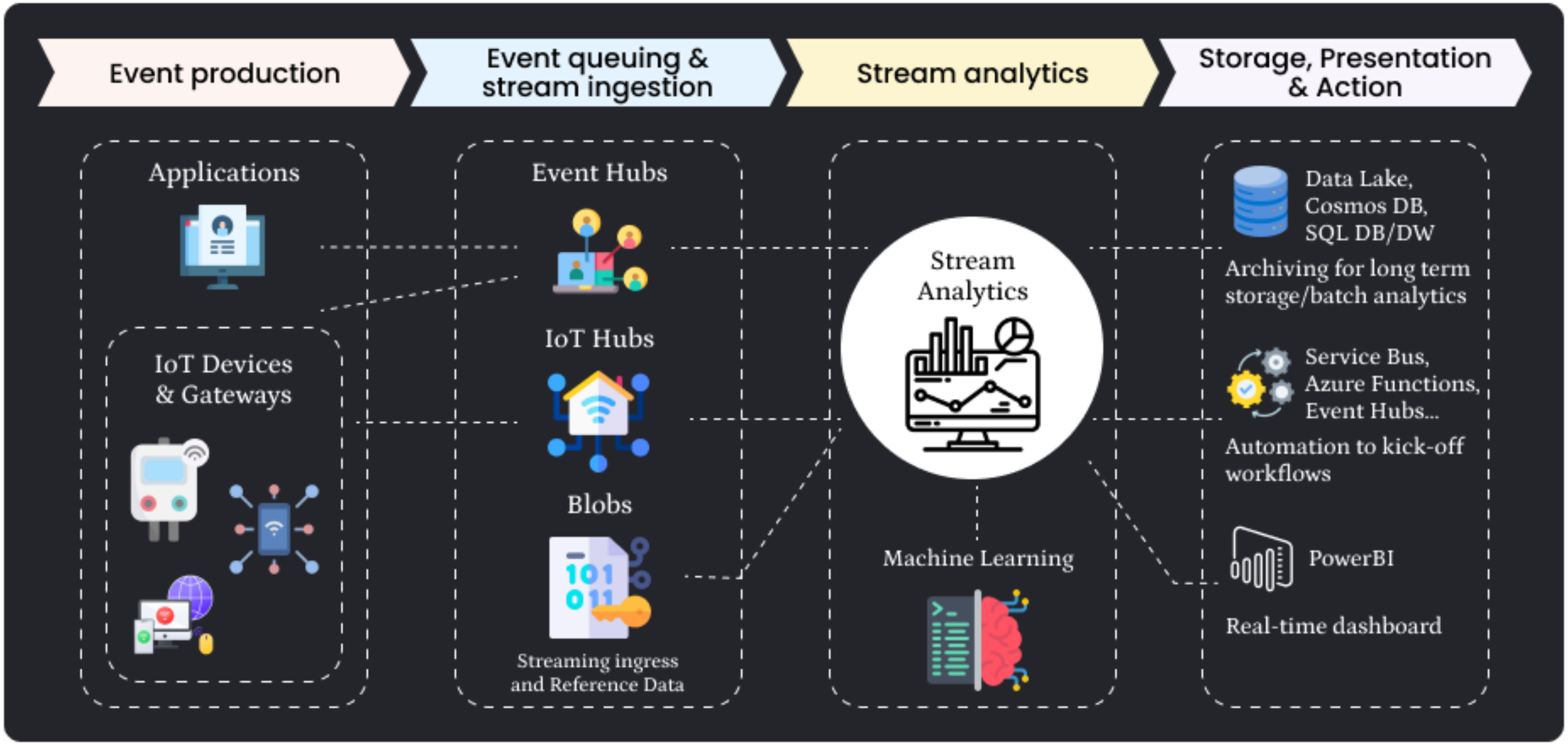
Apache Kafka

https://kafka.apache.org/

Kafka Use Cases

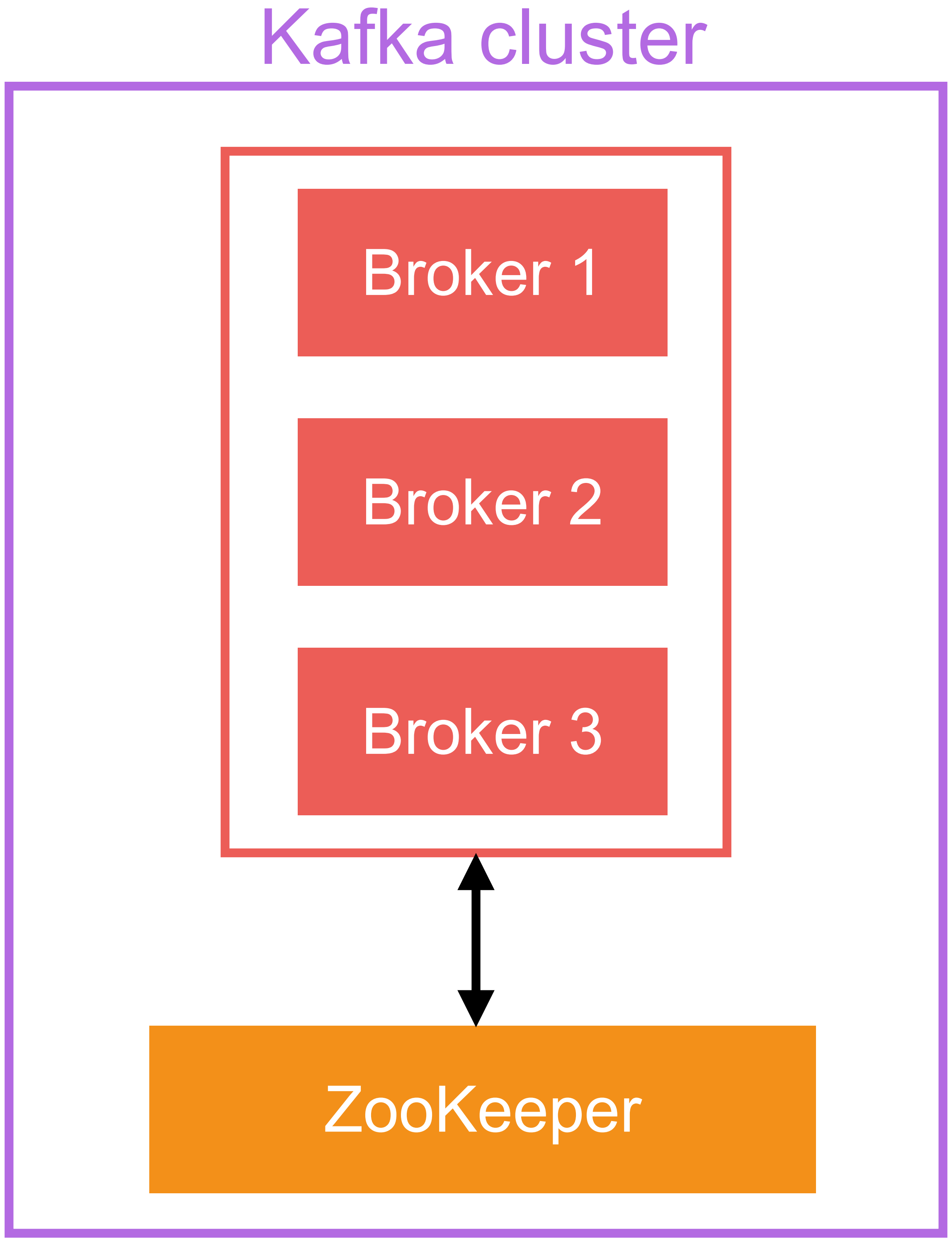
Realtime data streaming
Log aggregation
Metrics collection
Event sourcing
Streaming processing

Kafka Use Cases

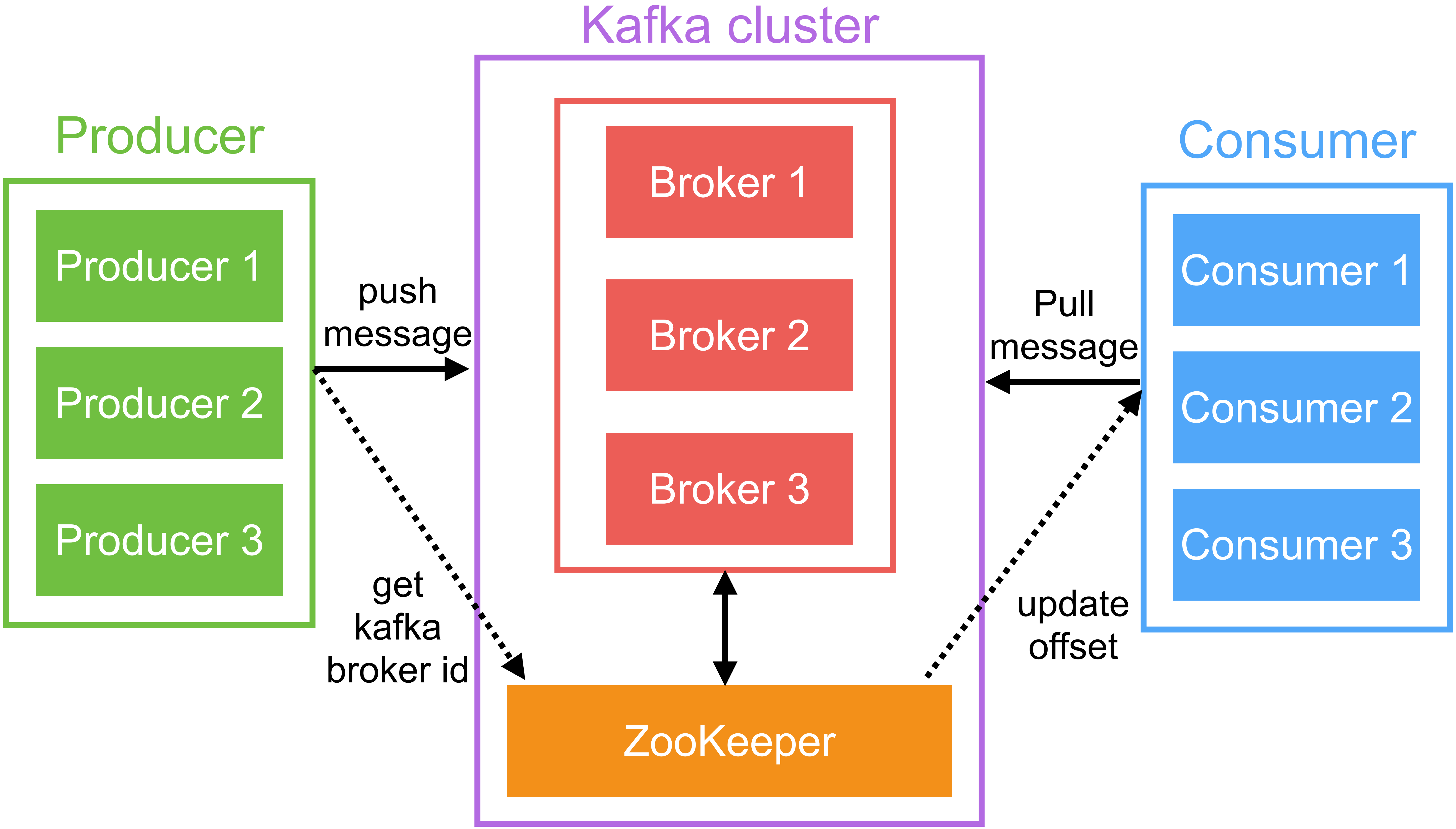

Event production
Event queuing & stream ingestion
Stream analytics
Storage, Presentation & Action
Applications
IoT Devices & Gateways
Event Hubs
IoT Hubs
Blobs
Streaming ingress and Reference Data
Stream Analytics
Machine Learning
Data Lake, Cosmos DB, SQL DB/DW
Archiving for long term storage/batch analytics
Service Bus, Azure Functions, Event Hubs...
Automation to kick-off workflows
PowerBI
Real-time dashboard

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

Kafka Architecture



Kafka Architecture



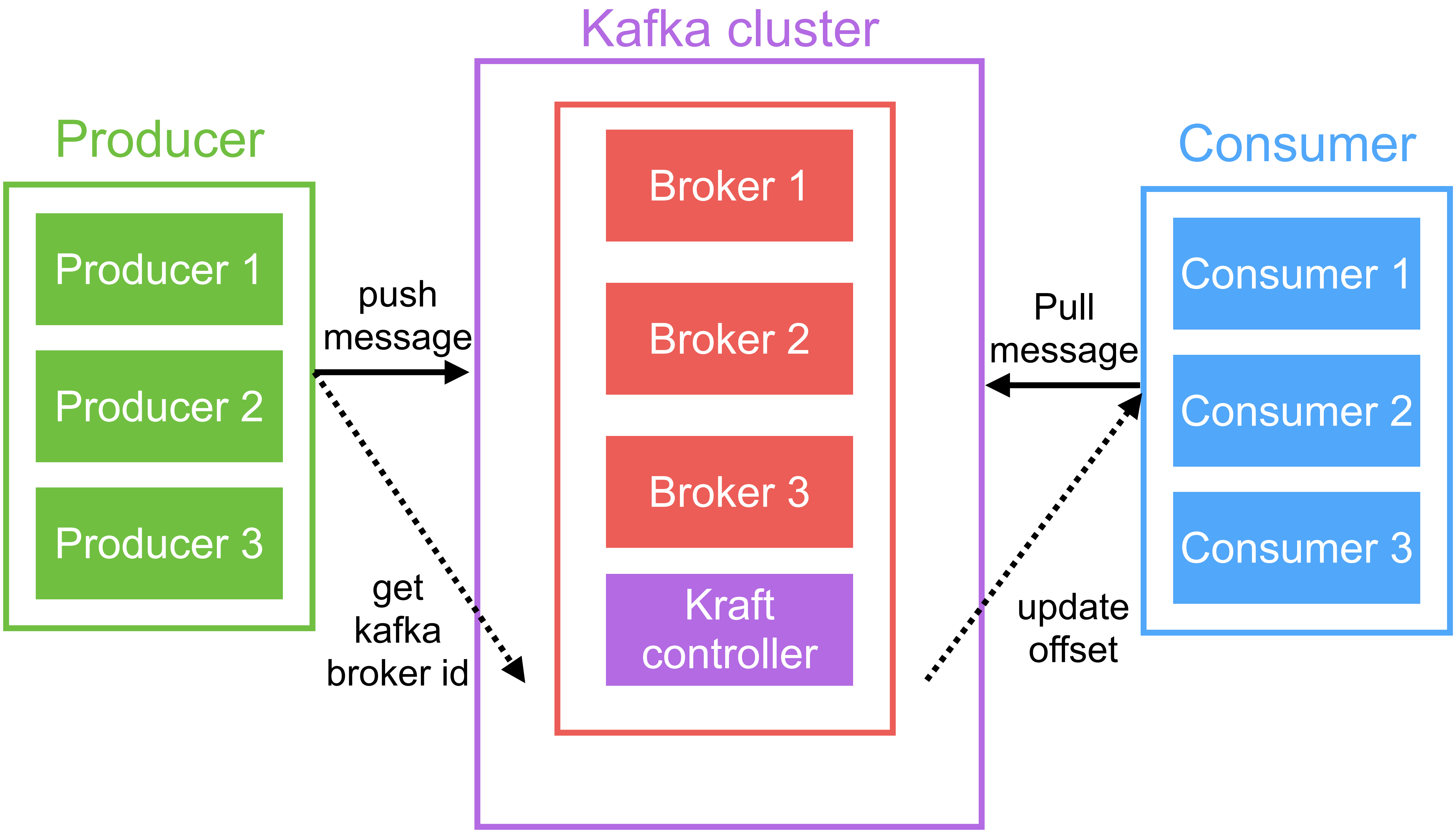
2025/04/18

Kafka 4.0

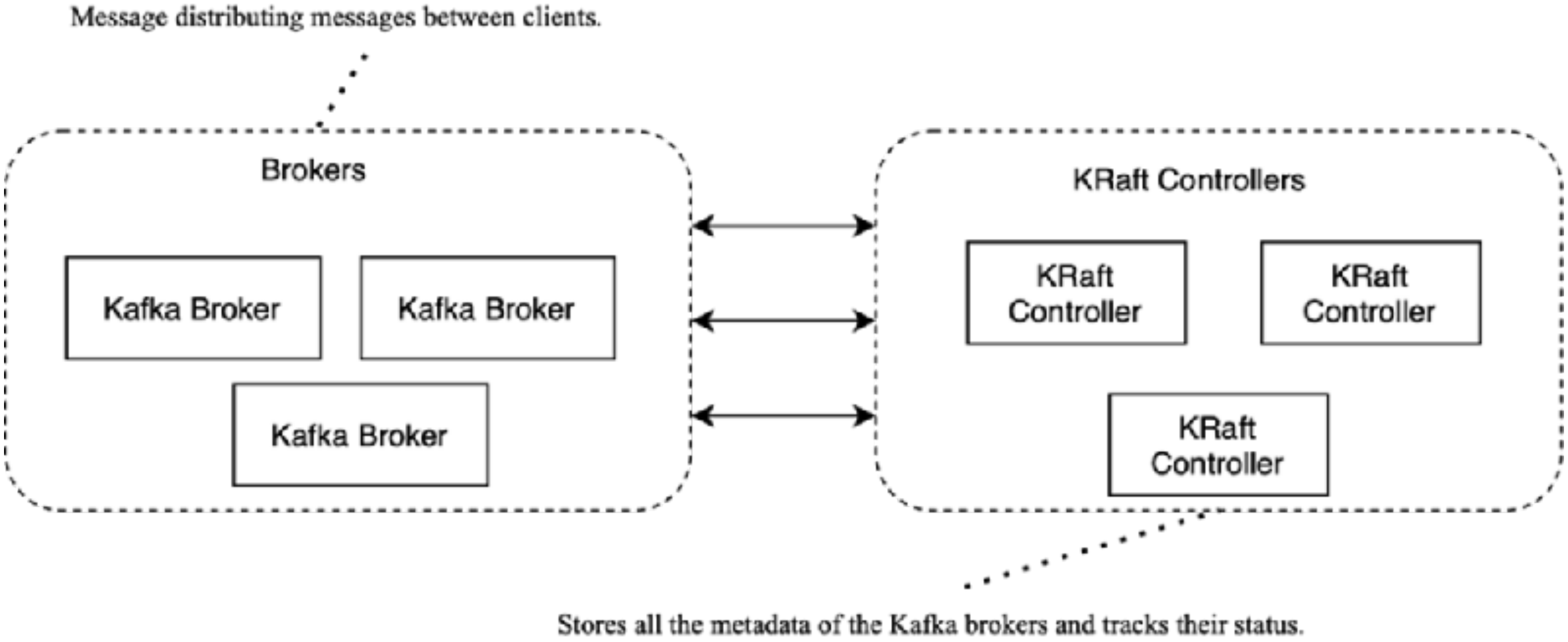
Remove Apache Zookeeper, use Kraft

https://zookeeper.apache.org/

Kafka Architecture 4.0



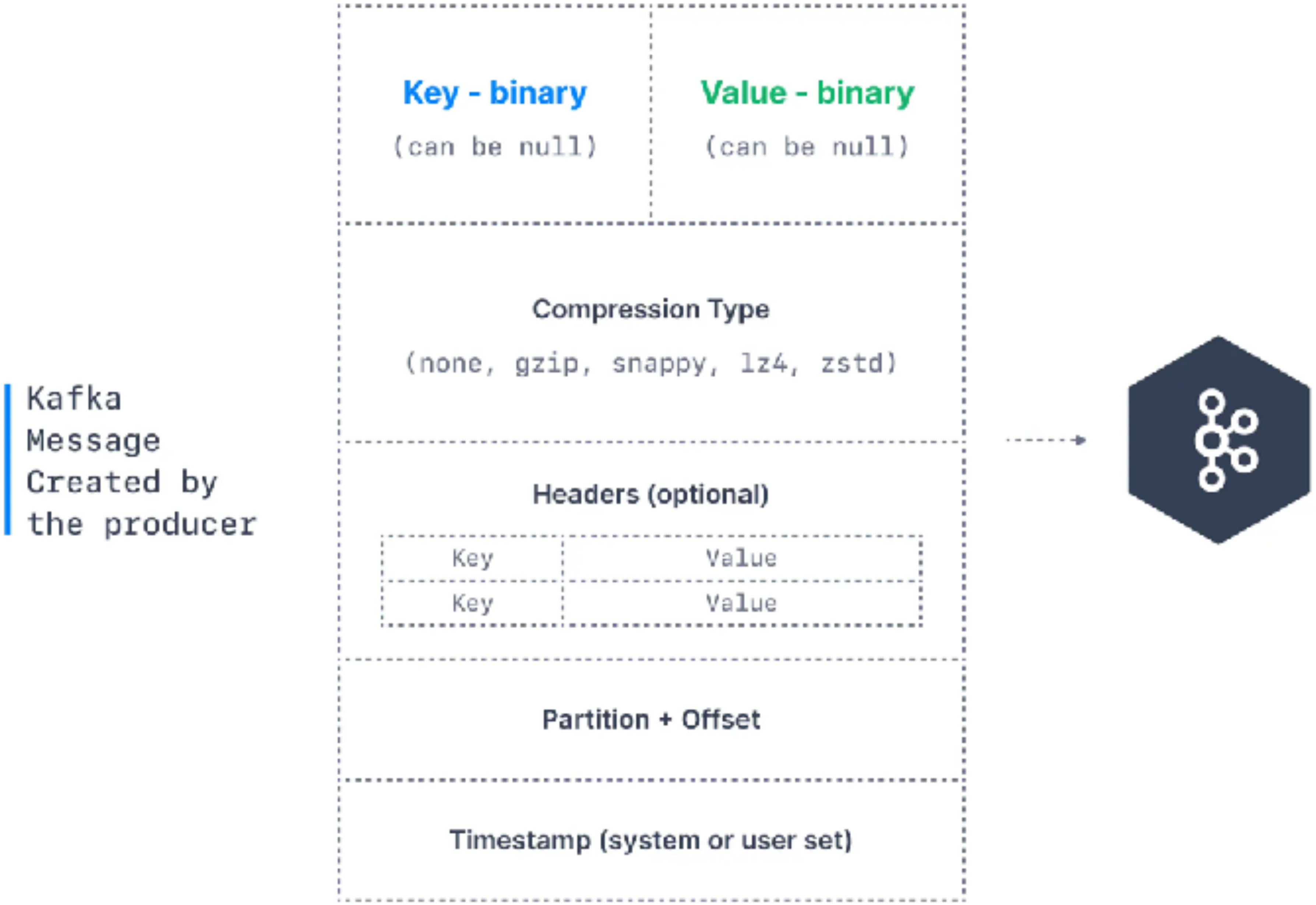
Kafka broker and Kraft controller


Message distributing messages between clients.
Brokers
Kafka Broker
Kafka Broker
Kafka Broker
KRaft Controllers
KRaft Controller
KRaft Controller
KRaft Controller
Stores all the metadata of the Kafka brokers and tracks their status.

Kafka Fundamentals

Event/record/message
Producers
Consumers
Topics
Partitions
Broker
Consumer groups
Cluster/Replicate
Offset (read and write)

Event Anatomy



Kafka Message Created by the producer

Key - binary
(can be null)

Value - binary
(can be null)

Compression Type
(none, gzip, snappy, lz4, zstd)

Headers (optional)

Key	Value
Key	Value

Partition + Offset

Timestamp (system or user set)

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**

Topic

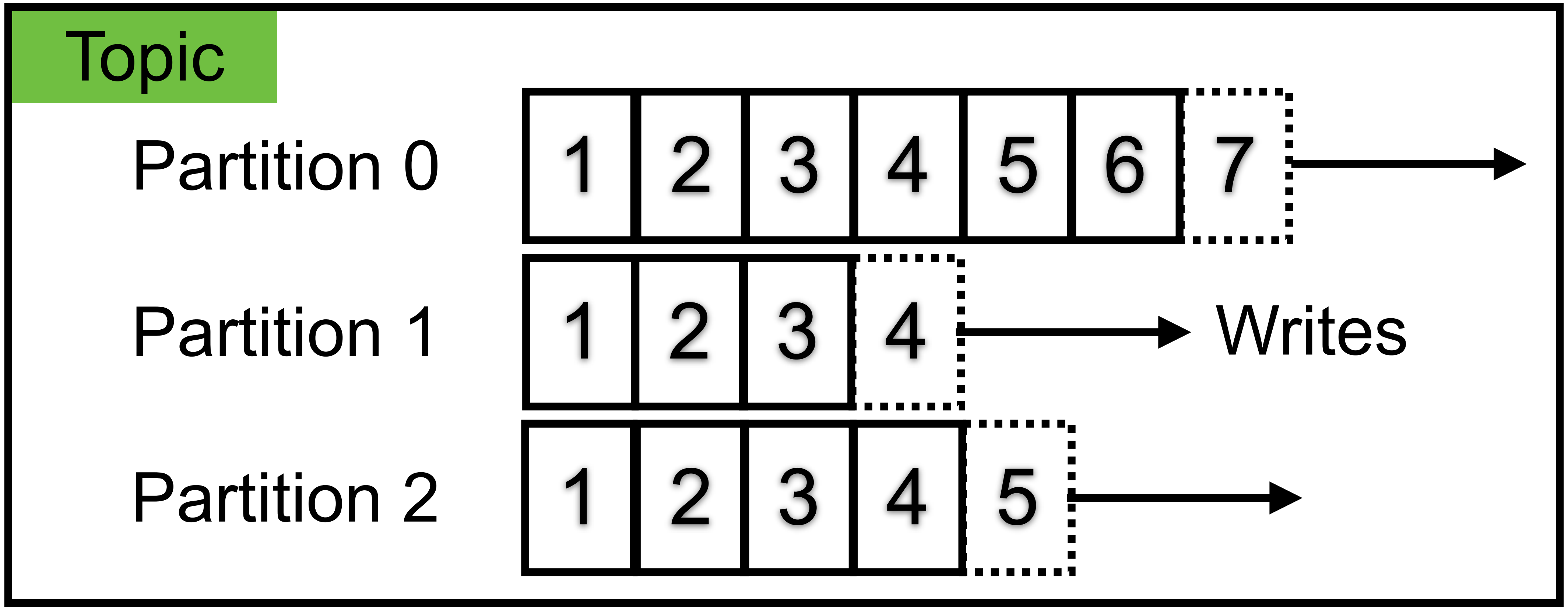
Topics, partitions and offsets

Partitions

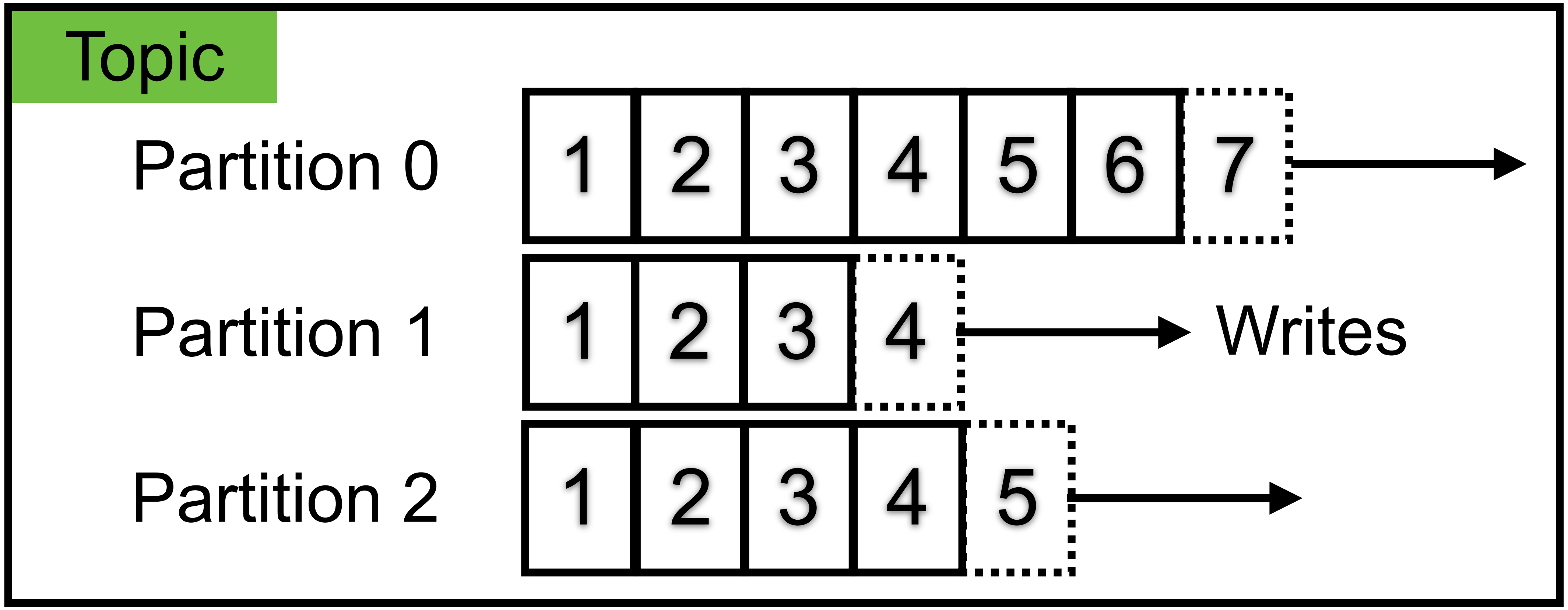
Topics are split in partitions

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

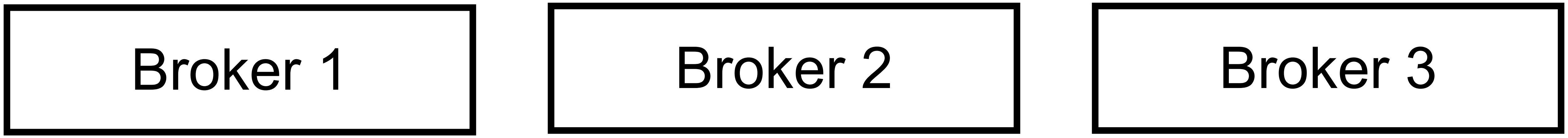
Brokers

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 connected to the entire cluster



Broker 1

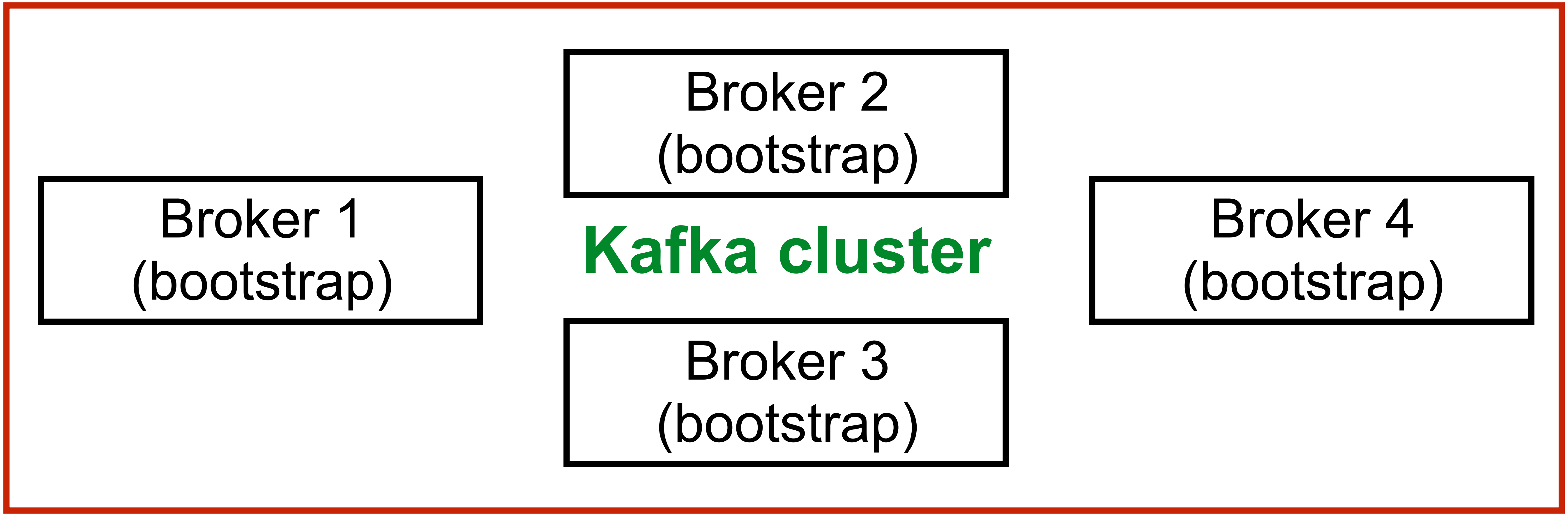
Broker 2

Broker 3

Kafka broker discovery

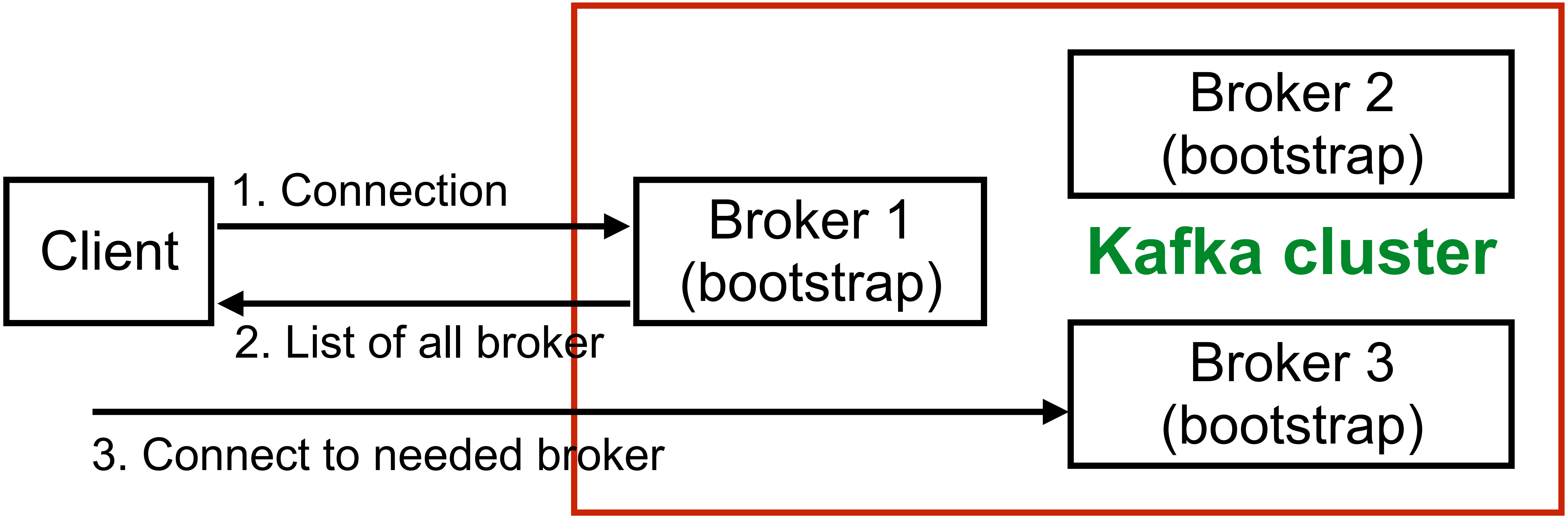
Every Kafka broker is called **“bootstrap server”**

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



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)

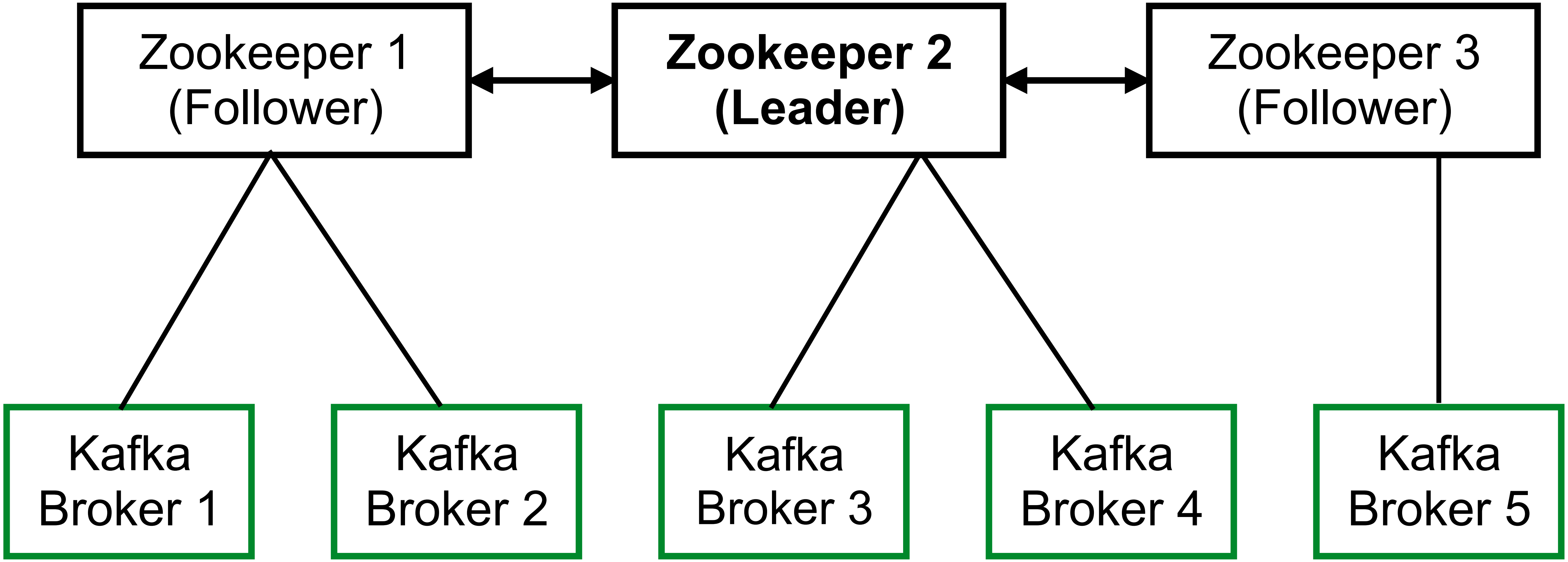
Data in Zookeeper

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



Zookeeper 1
(Follower)

Zookeeper 2
(Leader)

Zookeeper 3
(Follower)

Kafka
Broker 1

Kafka
Broker 2

Kafka
Broker 3

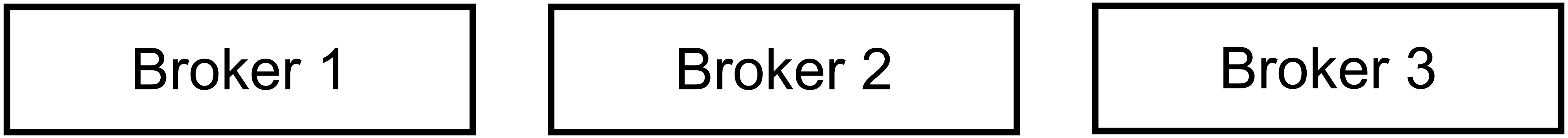
Kafka
Broker 4

Kafka
Broker 5

Brokers and topics

Brokers

Good number to start id **3 brokers**



Broker 1

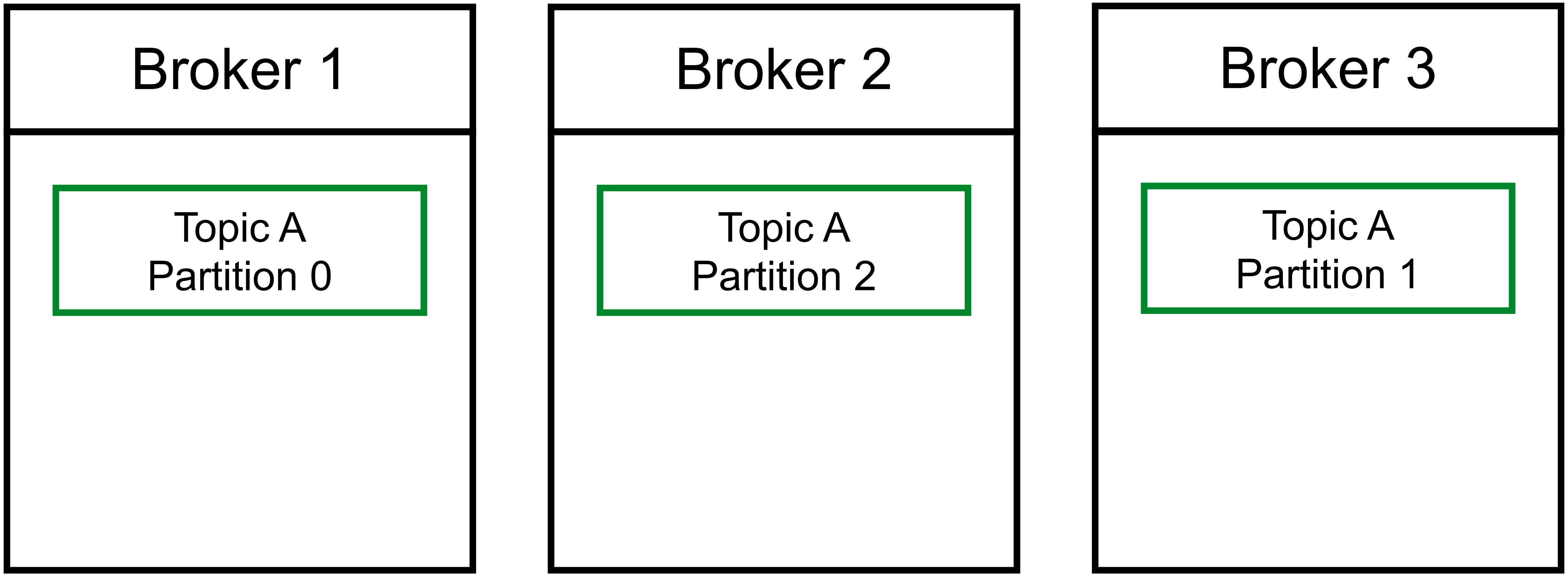
Broker 2

Broker 3

Brokers and topics

Example

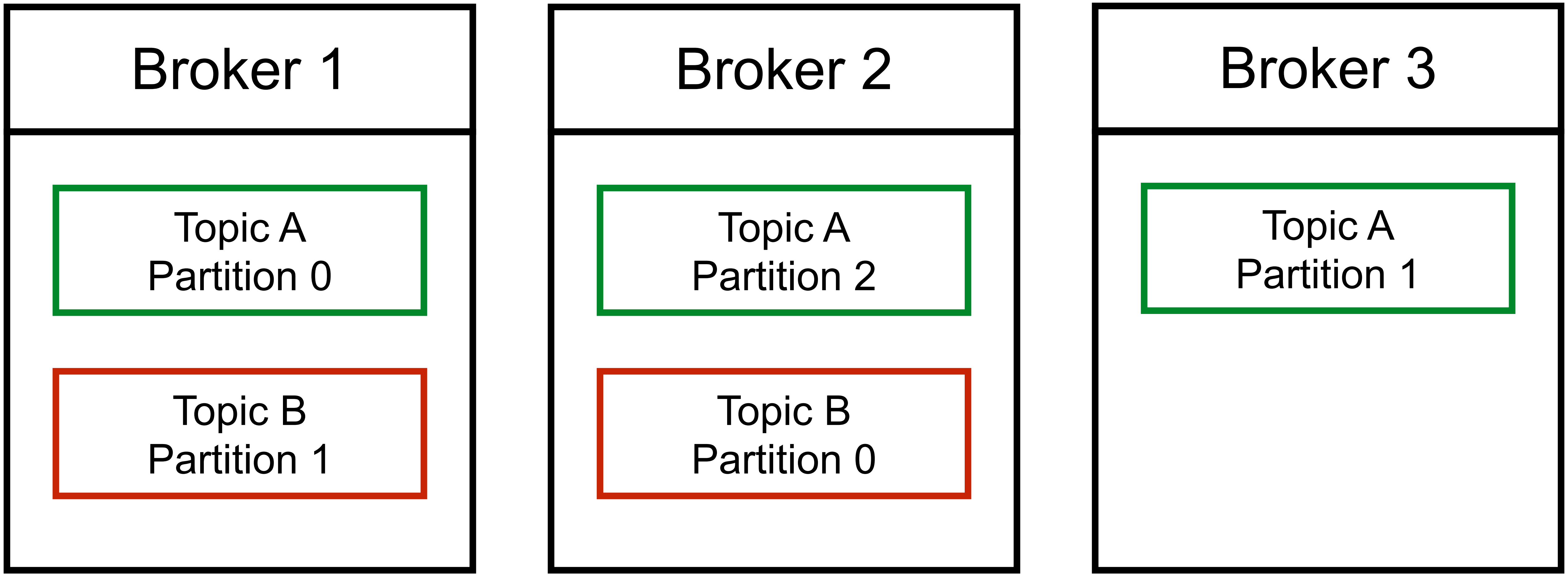
Topic A with 3 partitions



Brokers and topics

Example

Topic B with 2 partitions



Broker 1

Topic A
Partition 0

Topic B
Partition 1

Broker 2

Topic A
Partition 2

Topic B
Partition 0

Broker 3

Topic A
Partition 1

Topic with replication factor

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

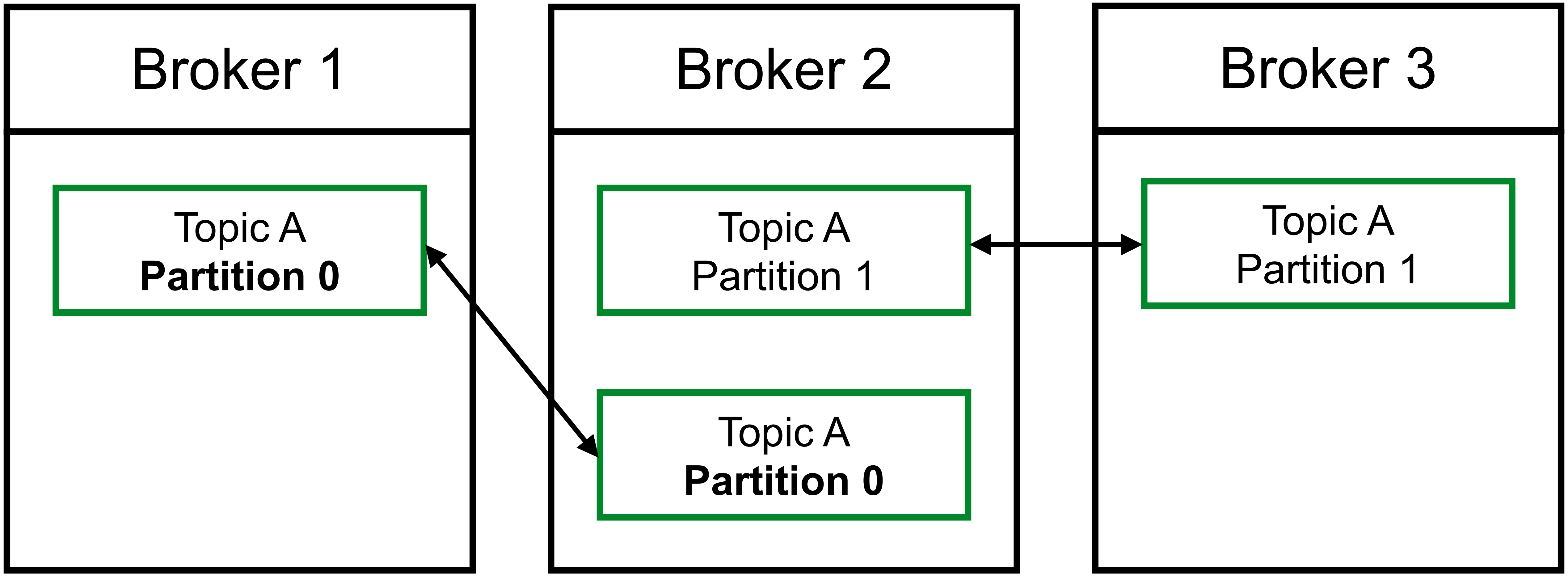
replica factor < no. of brokers

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

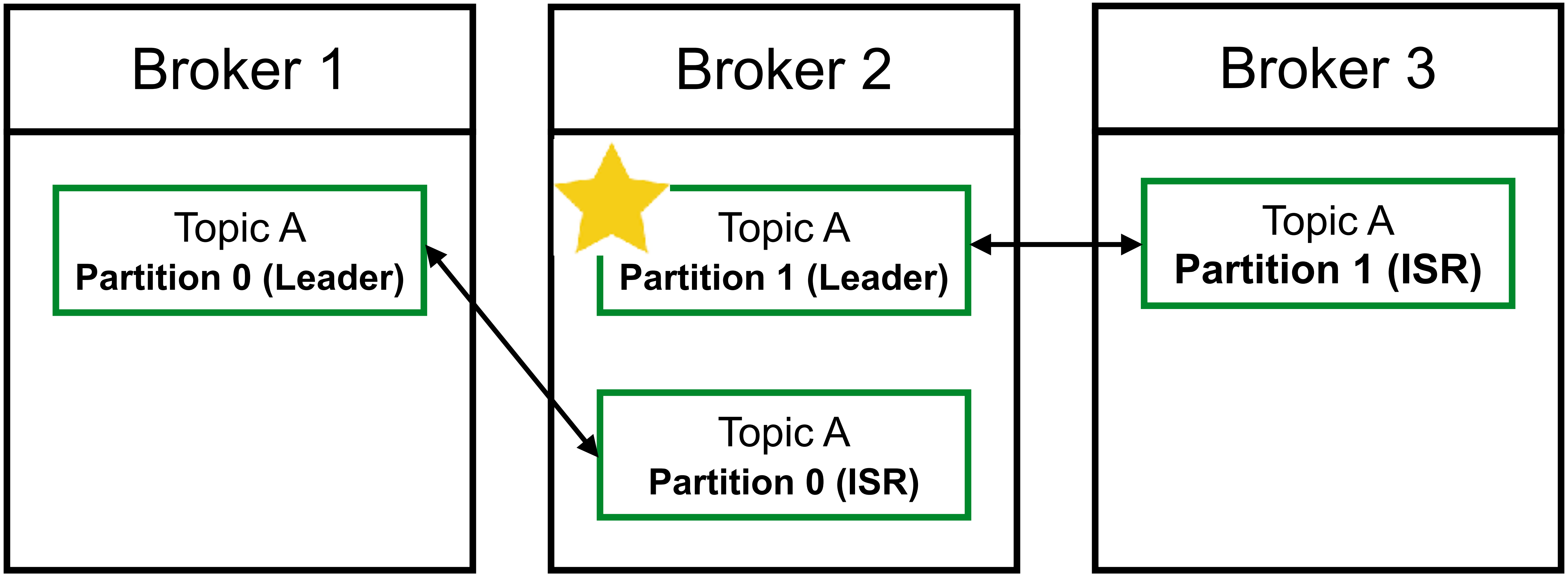
Other brokers will synchronize the data

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

Leader for a partition

Example

Topic A with 2 partitions and replication factor = 2



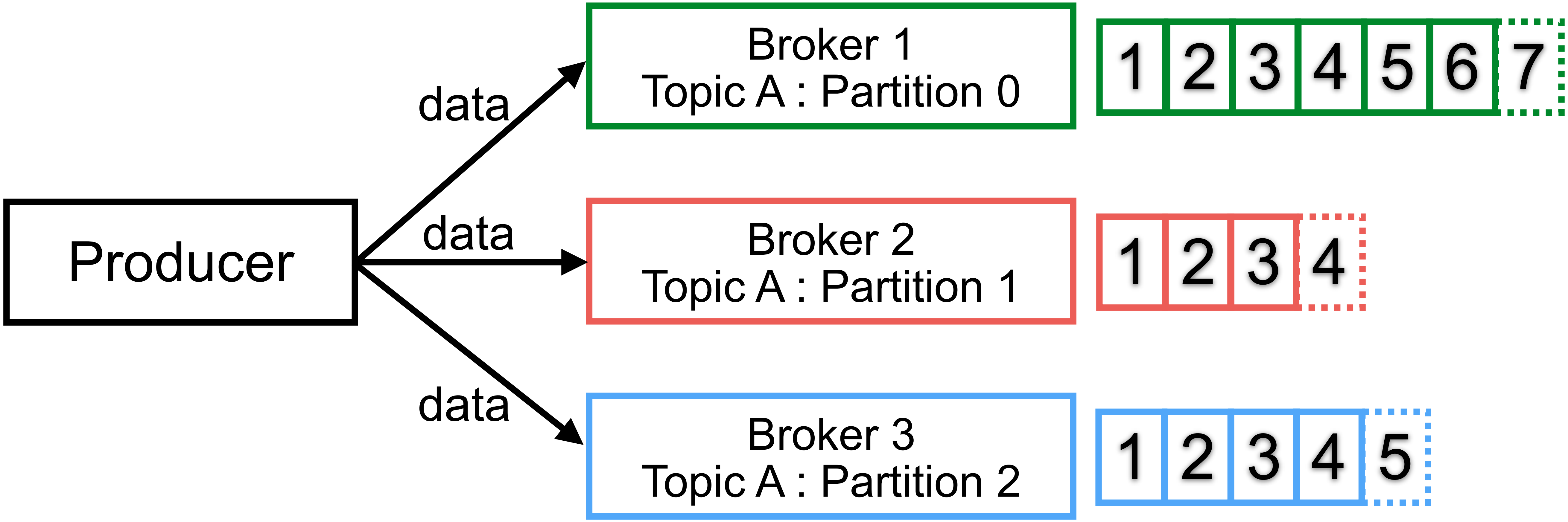
Producers

Producers write data to topics

Automatically know to broker and partition to write

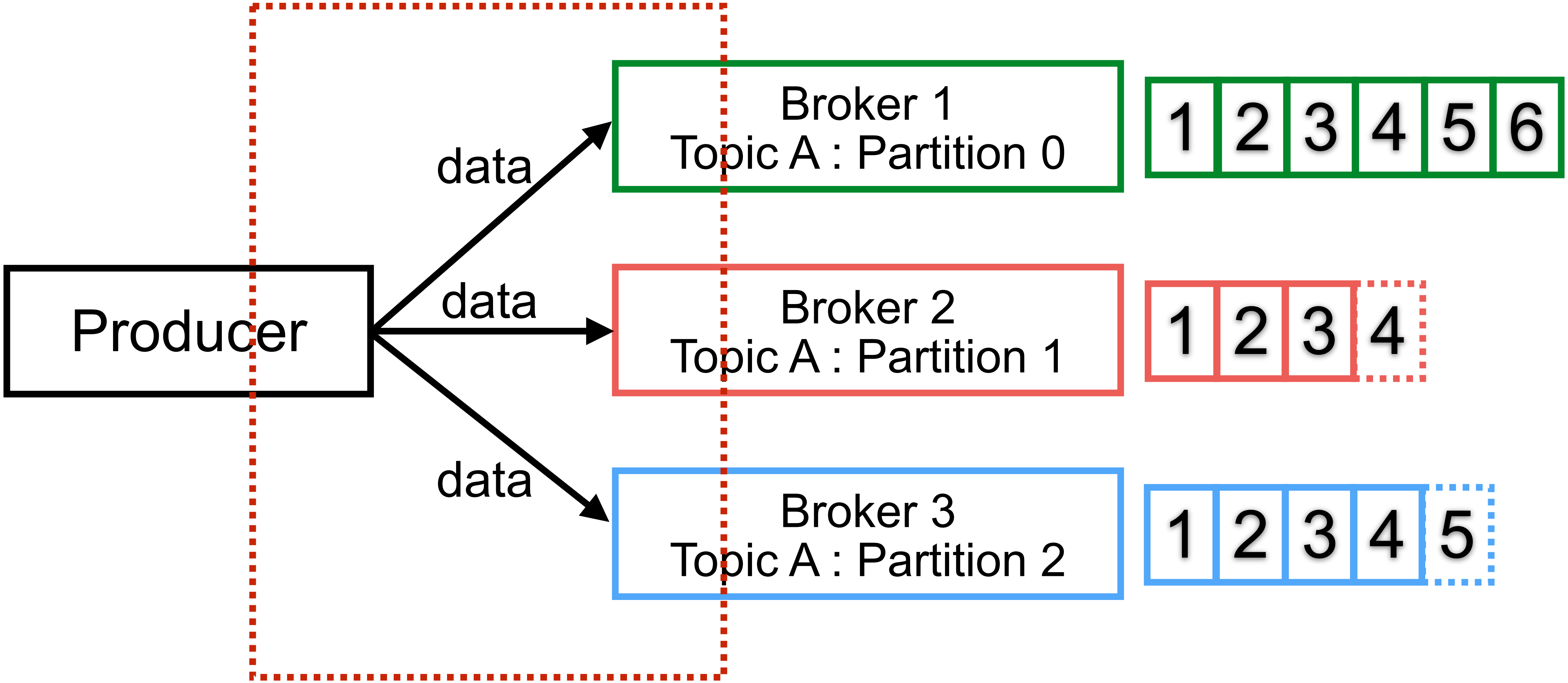
When broker failures, producers will automatically recover

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

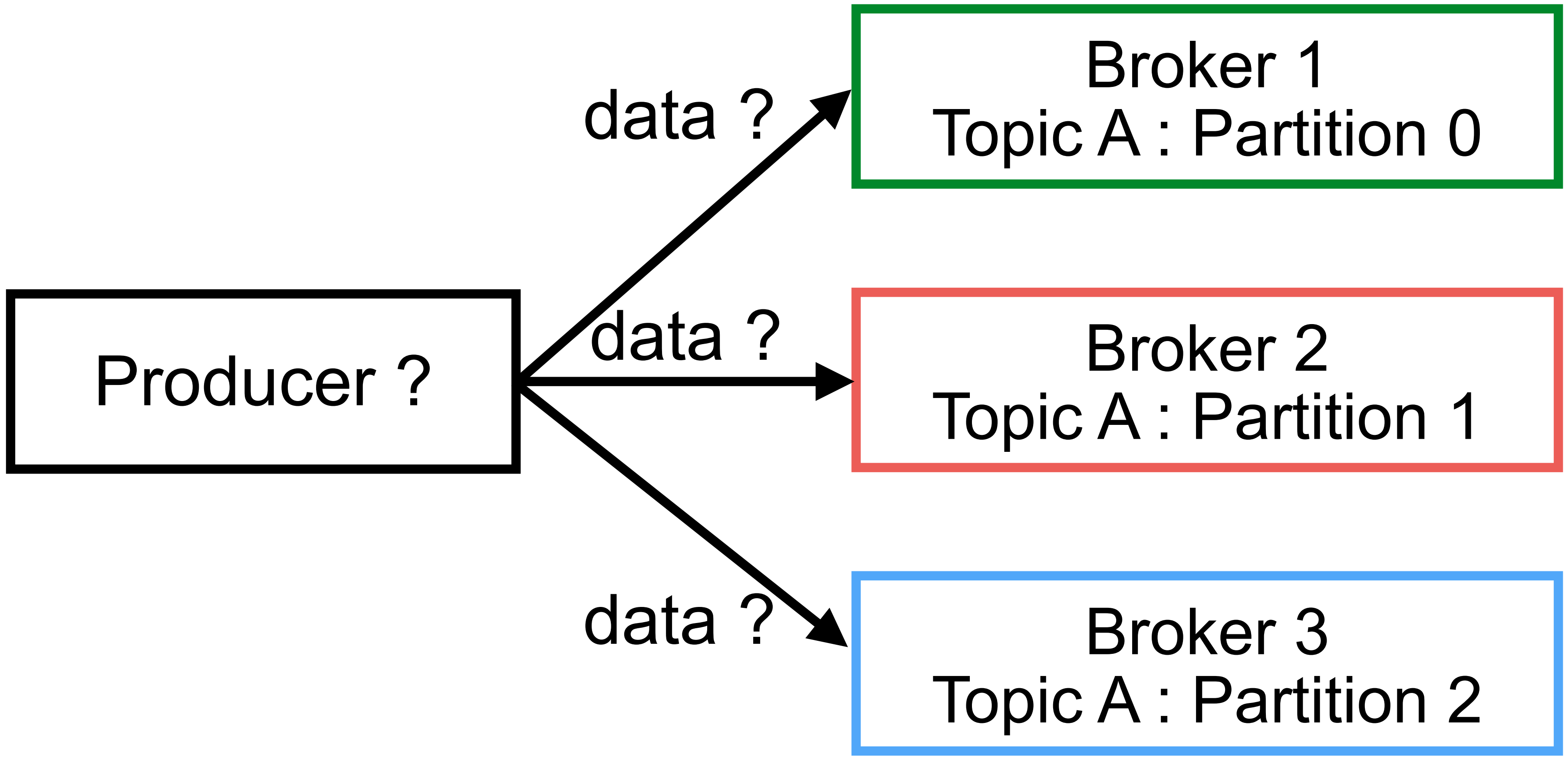
Customize (AbstractPartitionAssignor)

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

Producers with message keys

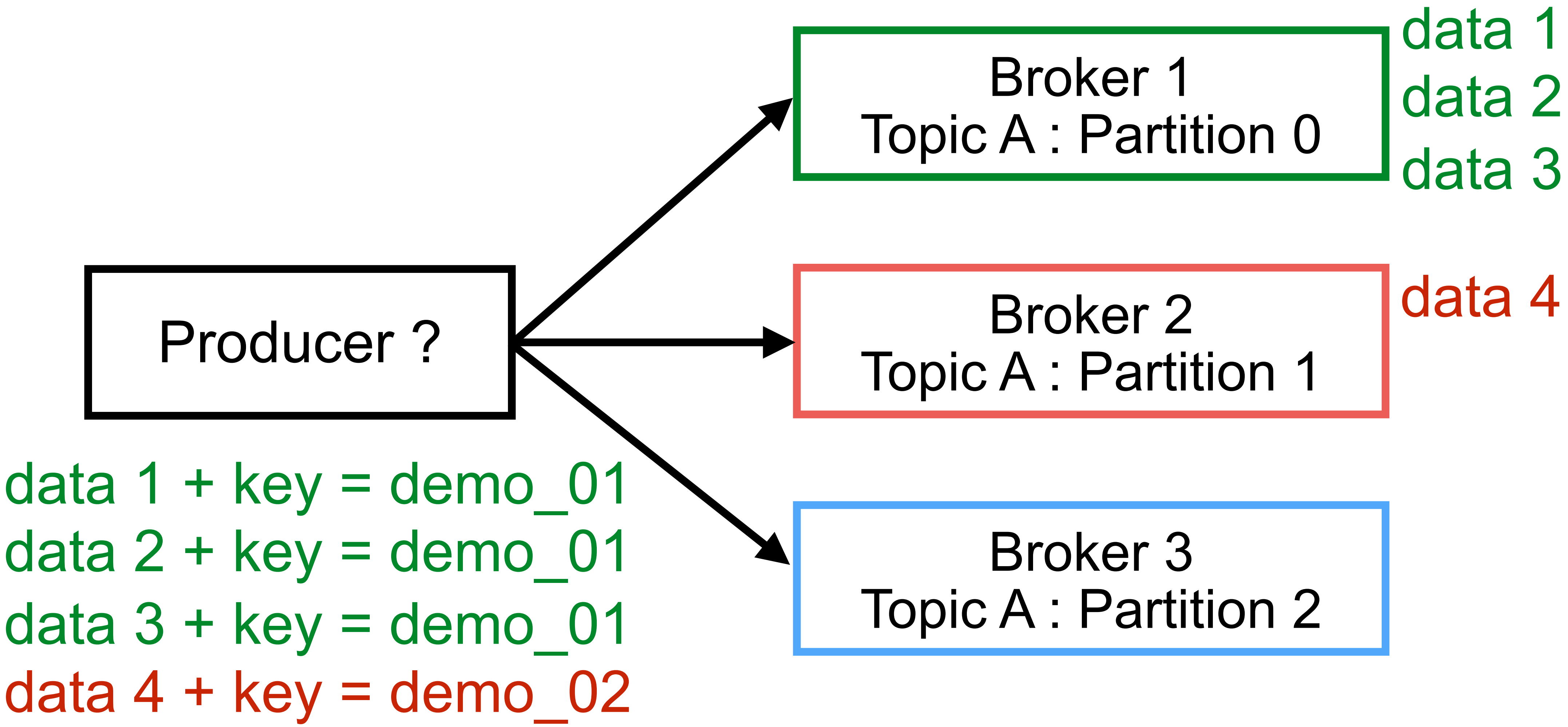
Producers can choose to sent a **key** with data

Key = null, data is sent **Sticky partitioner**



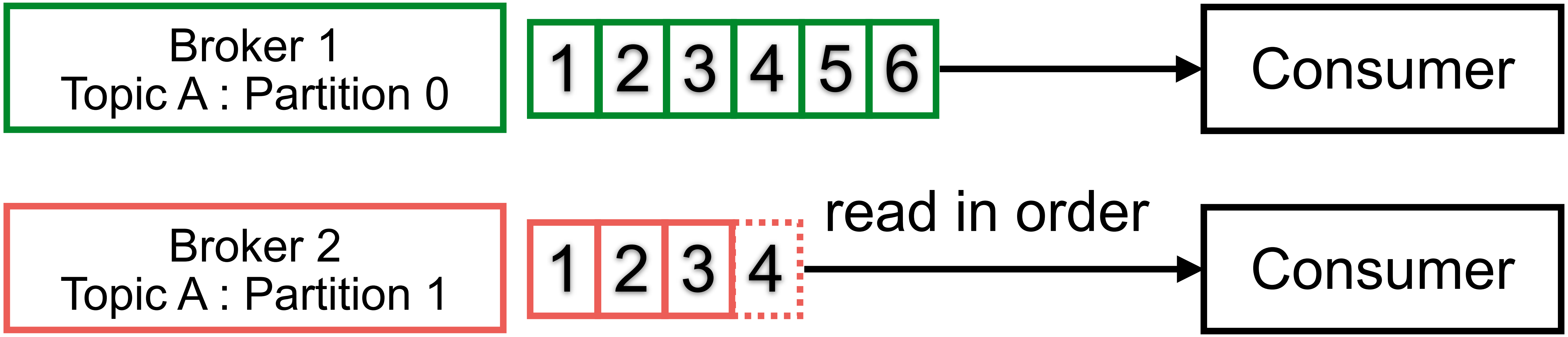
Why use message keys ?

You need message ordering



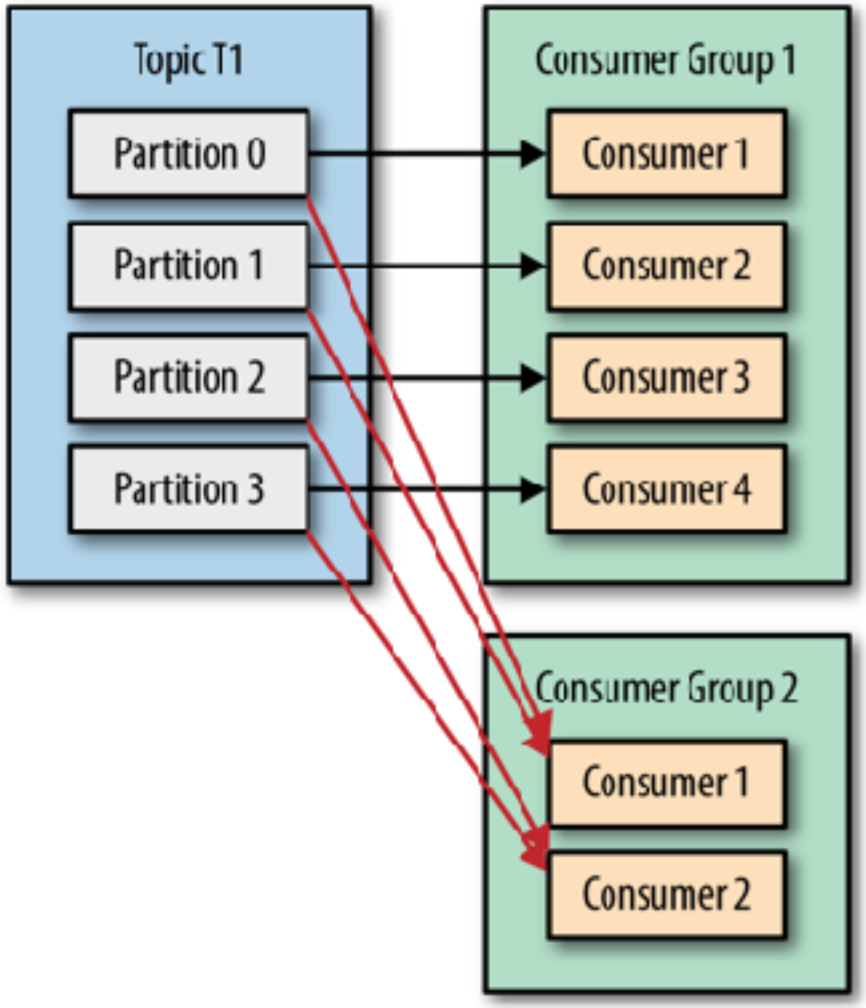
Consumers

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

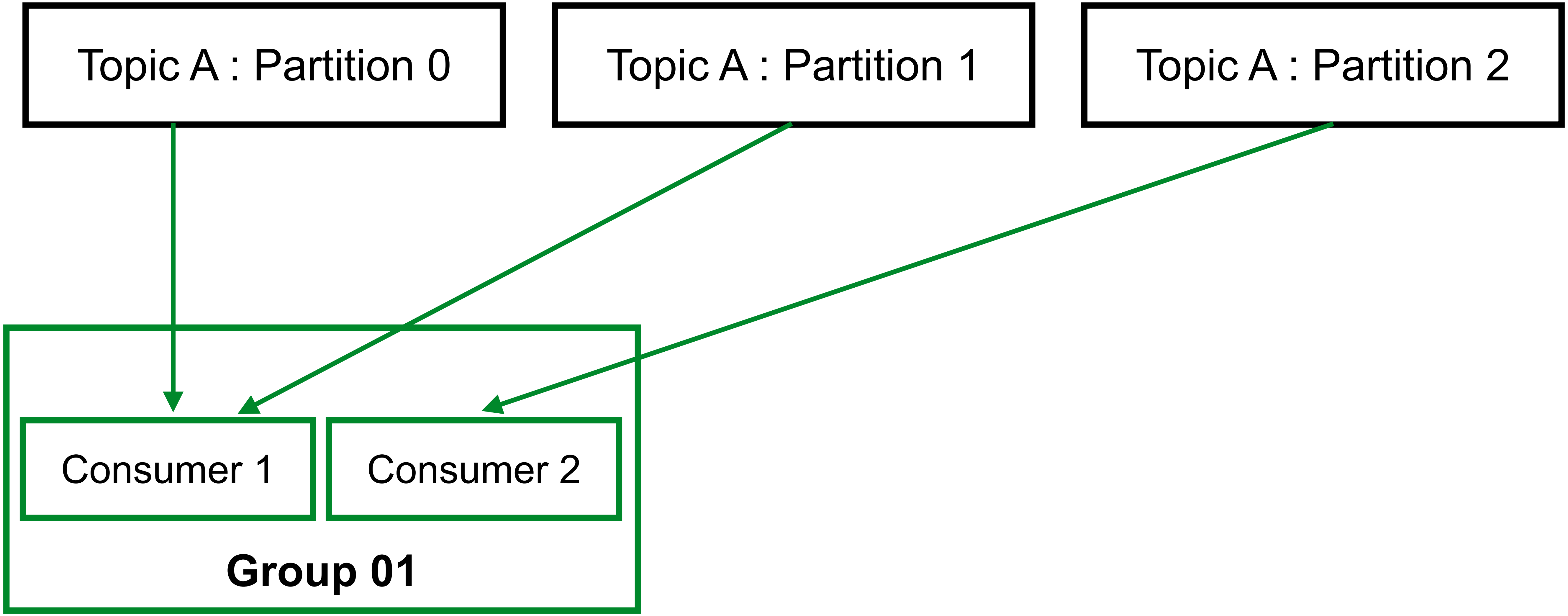


Consumer groups

Consumers read data in consumer groups
Each consumer in a group read from exclusive partitions

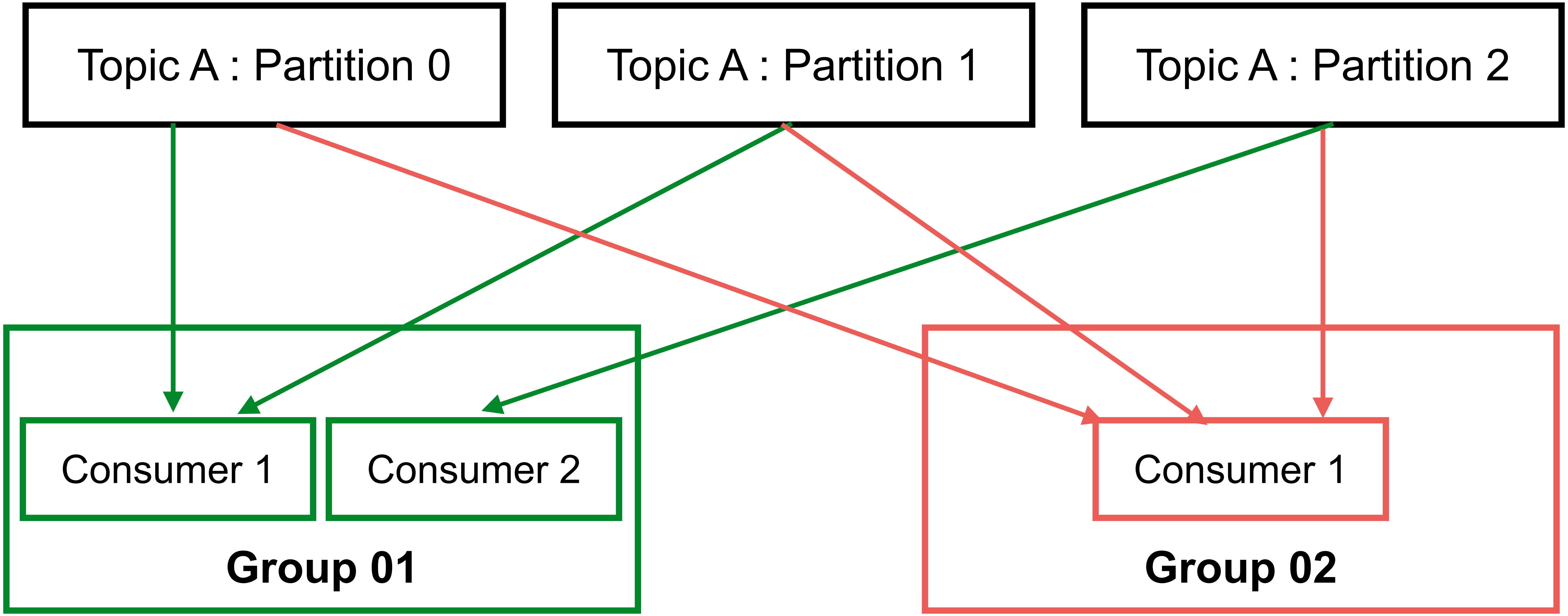


Consumer groups



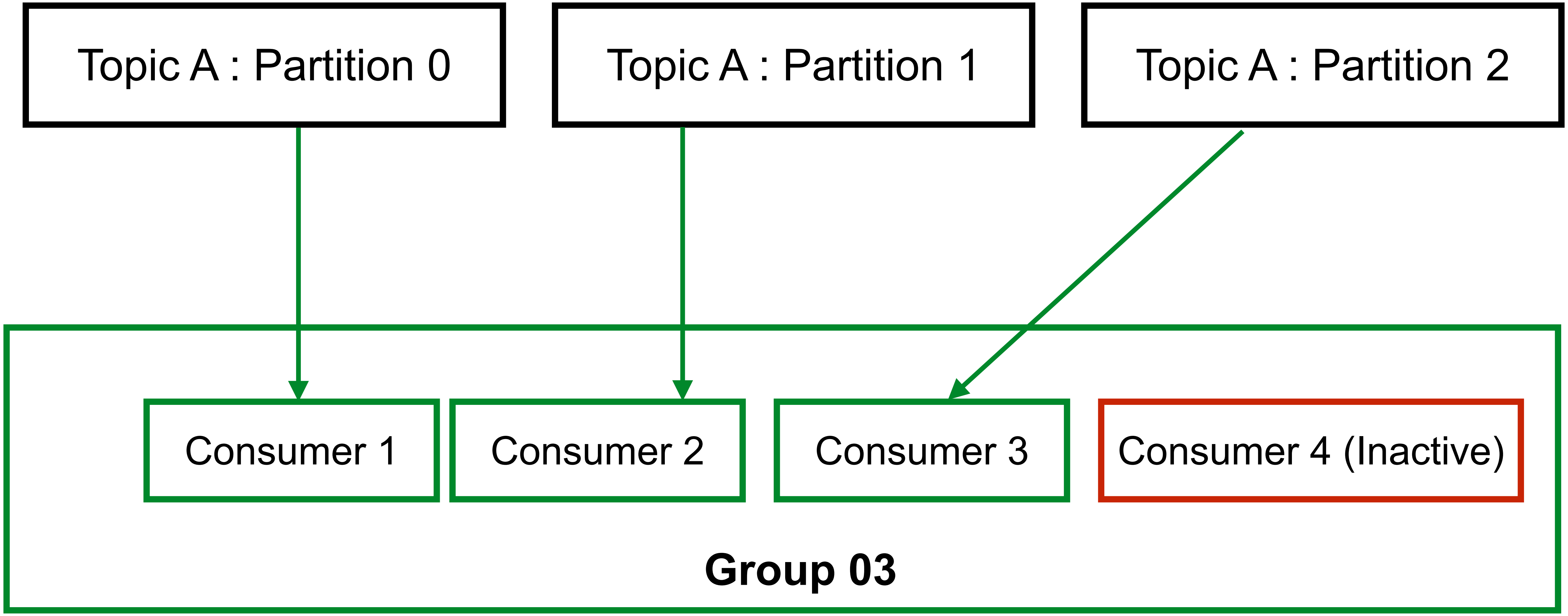
Consumer will automatically use a GroupCoordinator ConsumerCoordinator to assign a consumer to a partition.

Consumer groups

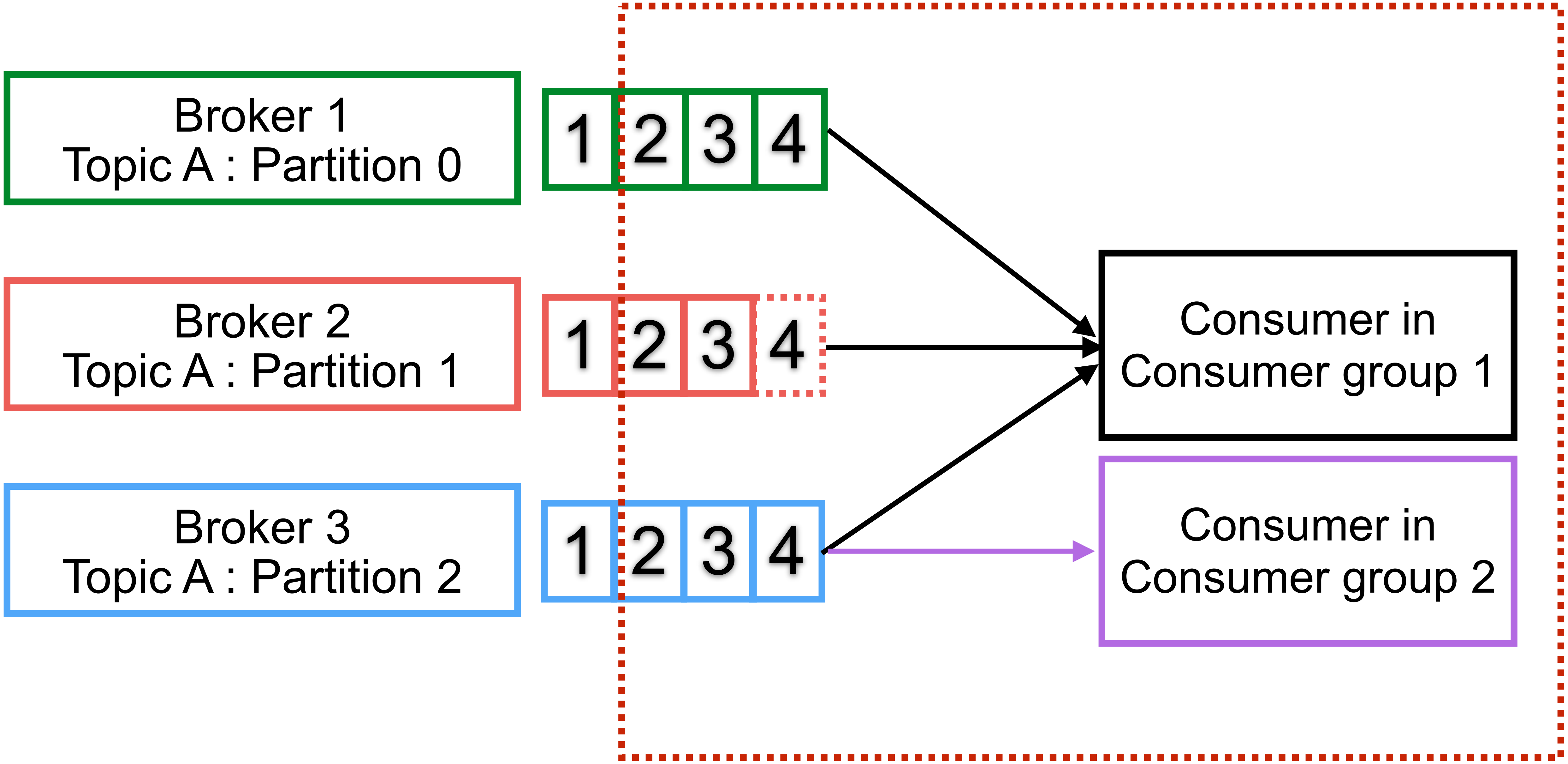


IF Consumers > partitions ?

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 lease once (preferred)
Exactly once

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

1. At most once

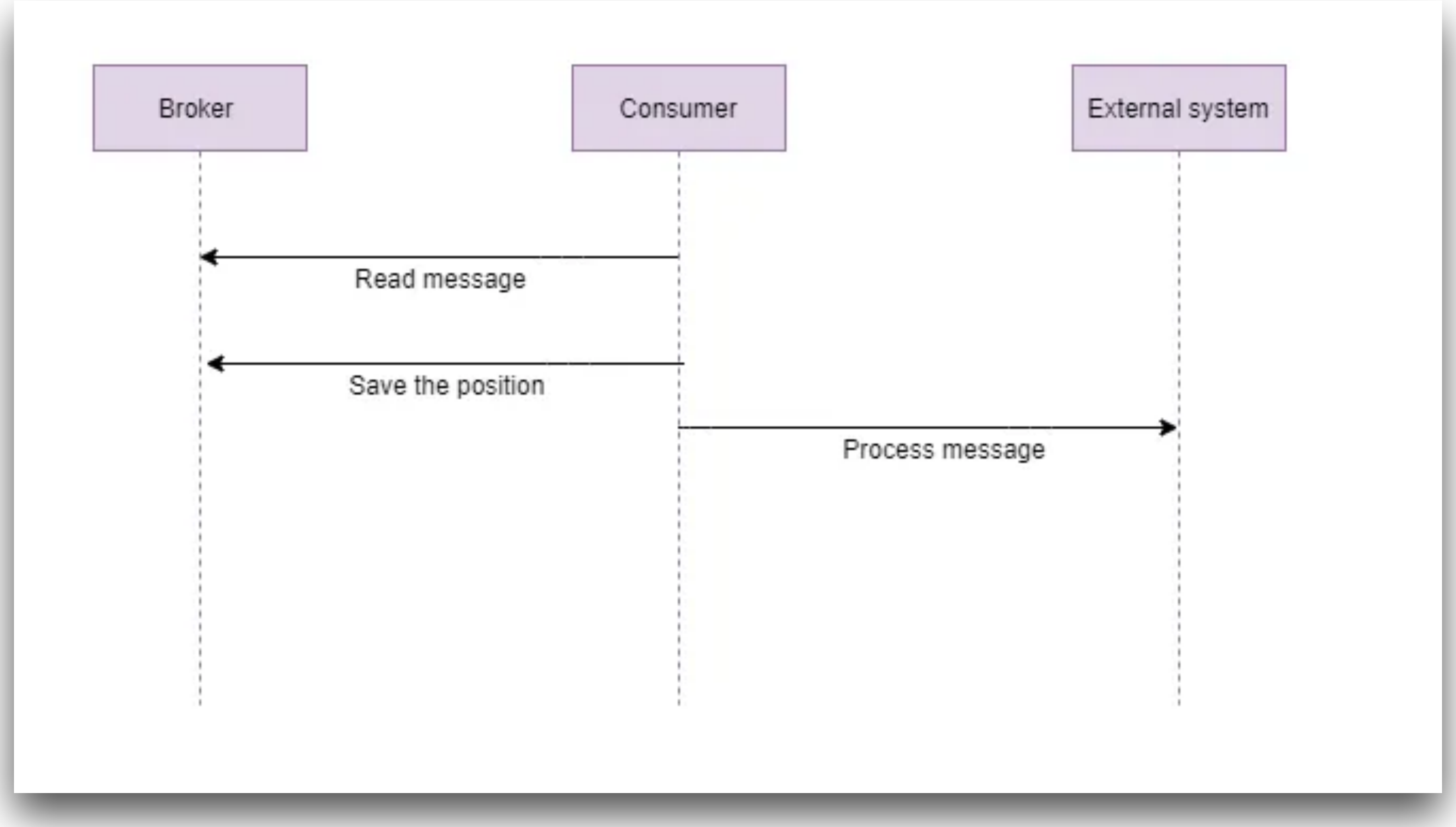
Offsets are committed as soon as the message is received

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

Fire and forgot pattern

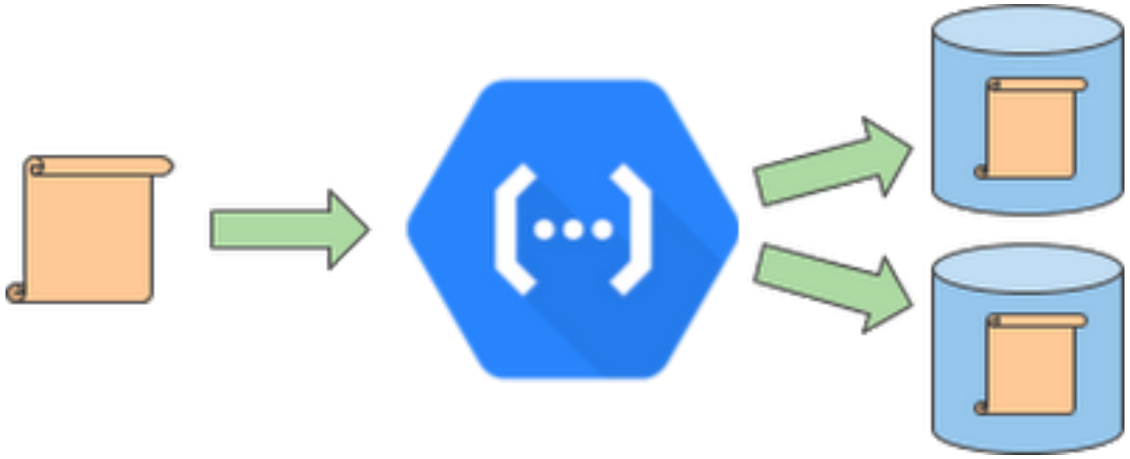
Lowest latency

At most once


Broker
Consumer
External system
Read message
Save the position
Process message

2. At lease once (1)

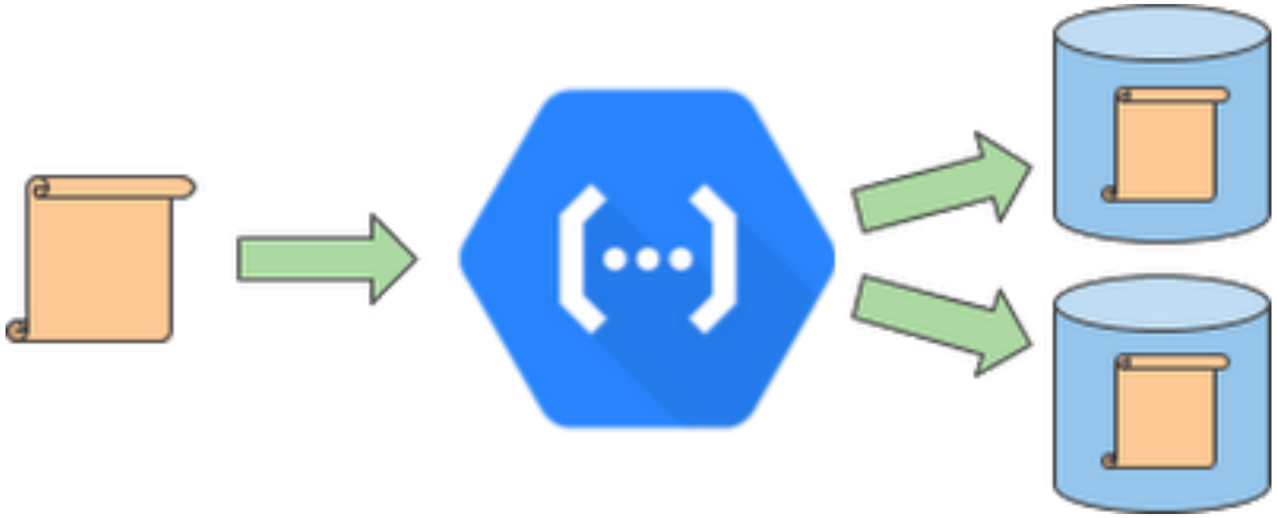
Offsets are committed after the message is processed
Messages are never lost but may be **redelivered**
If processing go wrong, the message will be **read again**



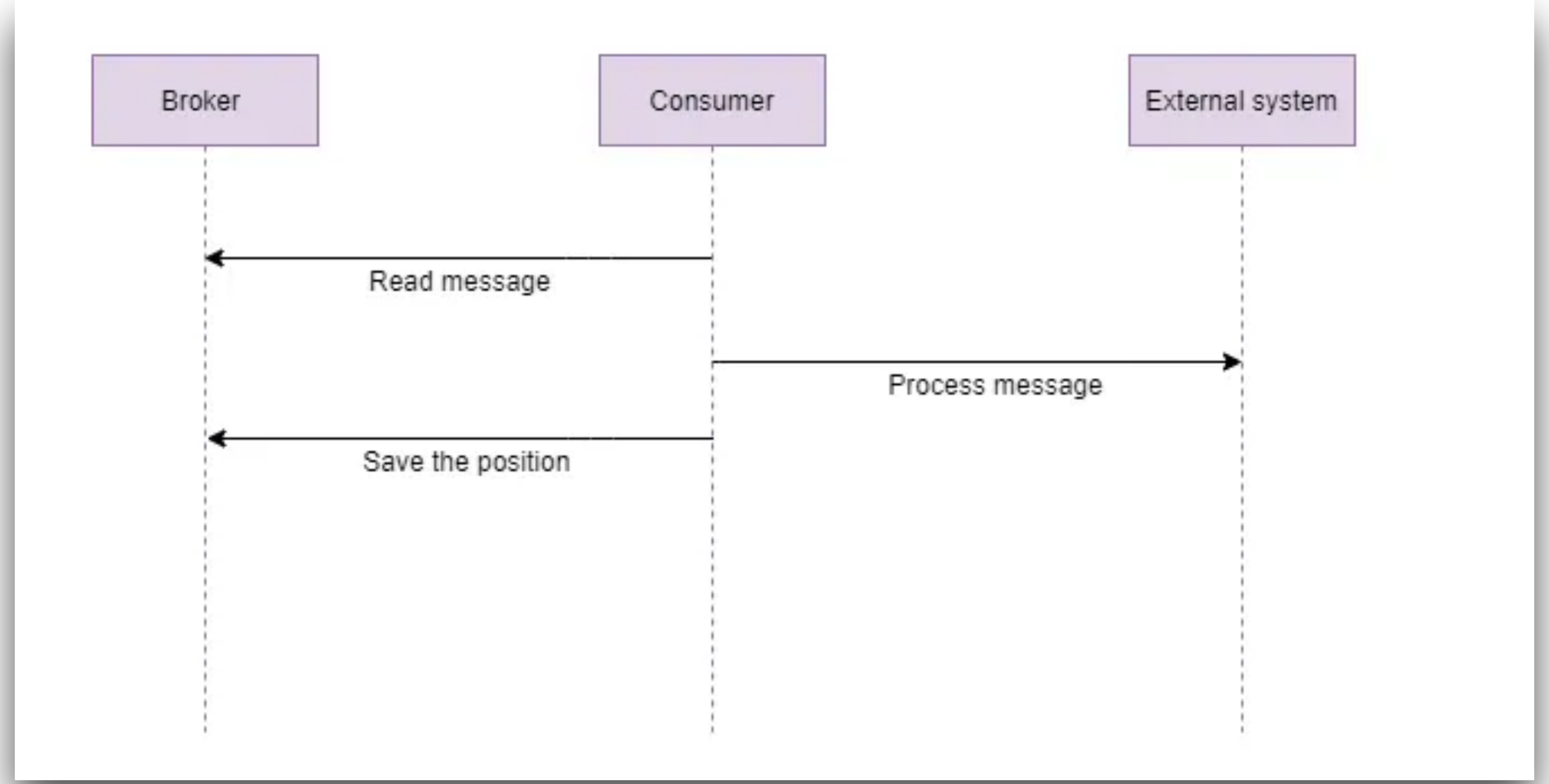
2. At lease once (2)

Make sure your processing is **idempotent**

Processing again the message
not impact to your system !!



At lease once

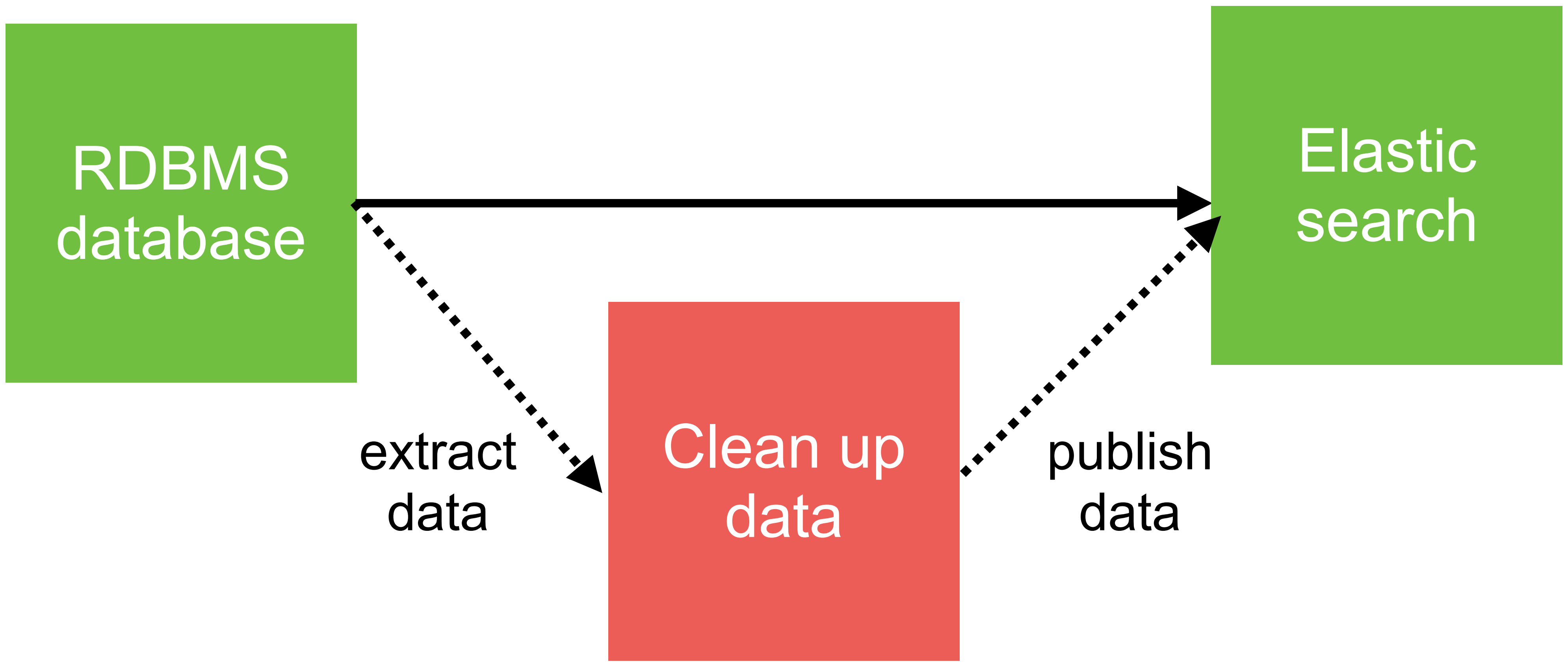


3. Exactly once

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

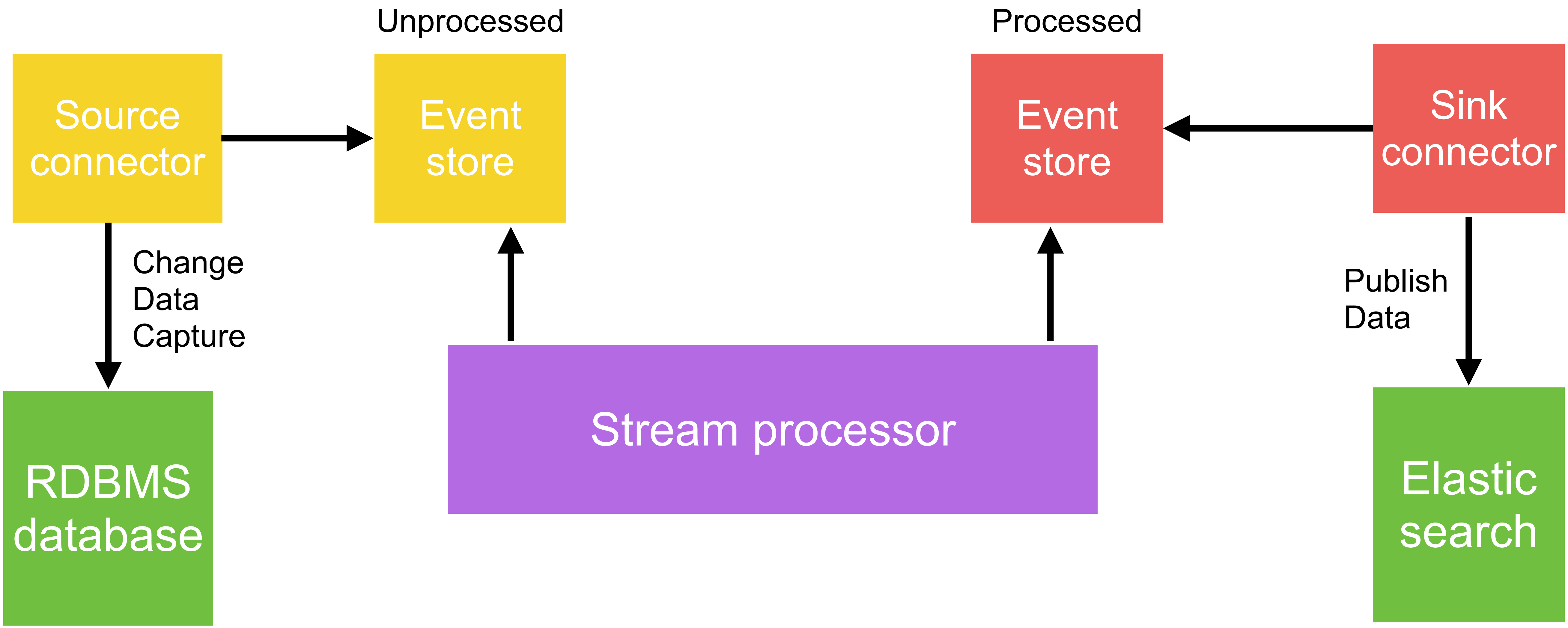
Data Pipeline

Batching process



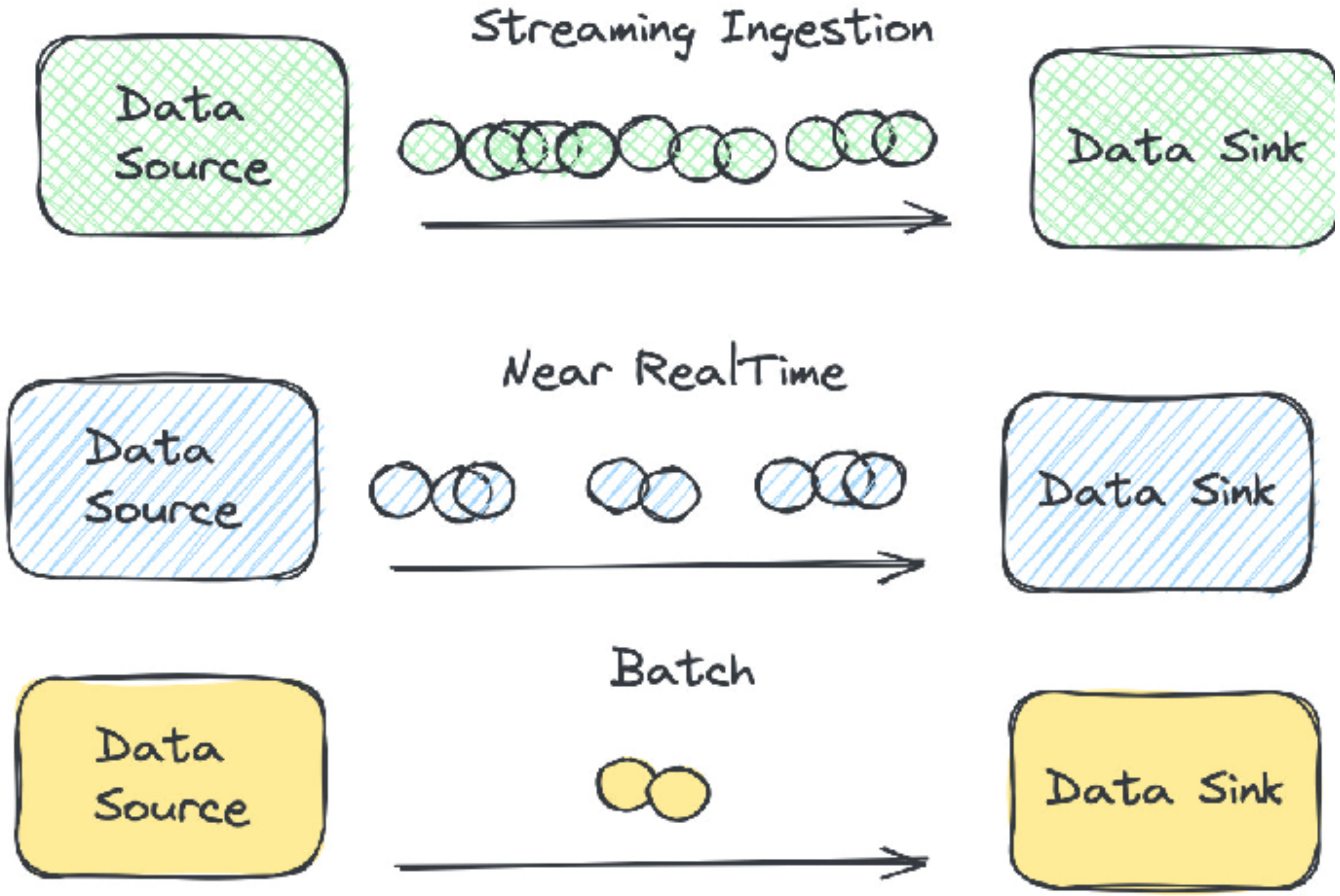
Schedule task run every 2 am

Streaming data pipeline



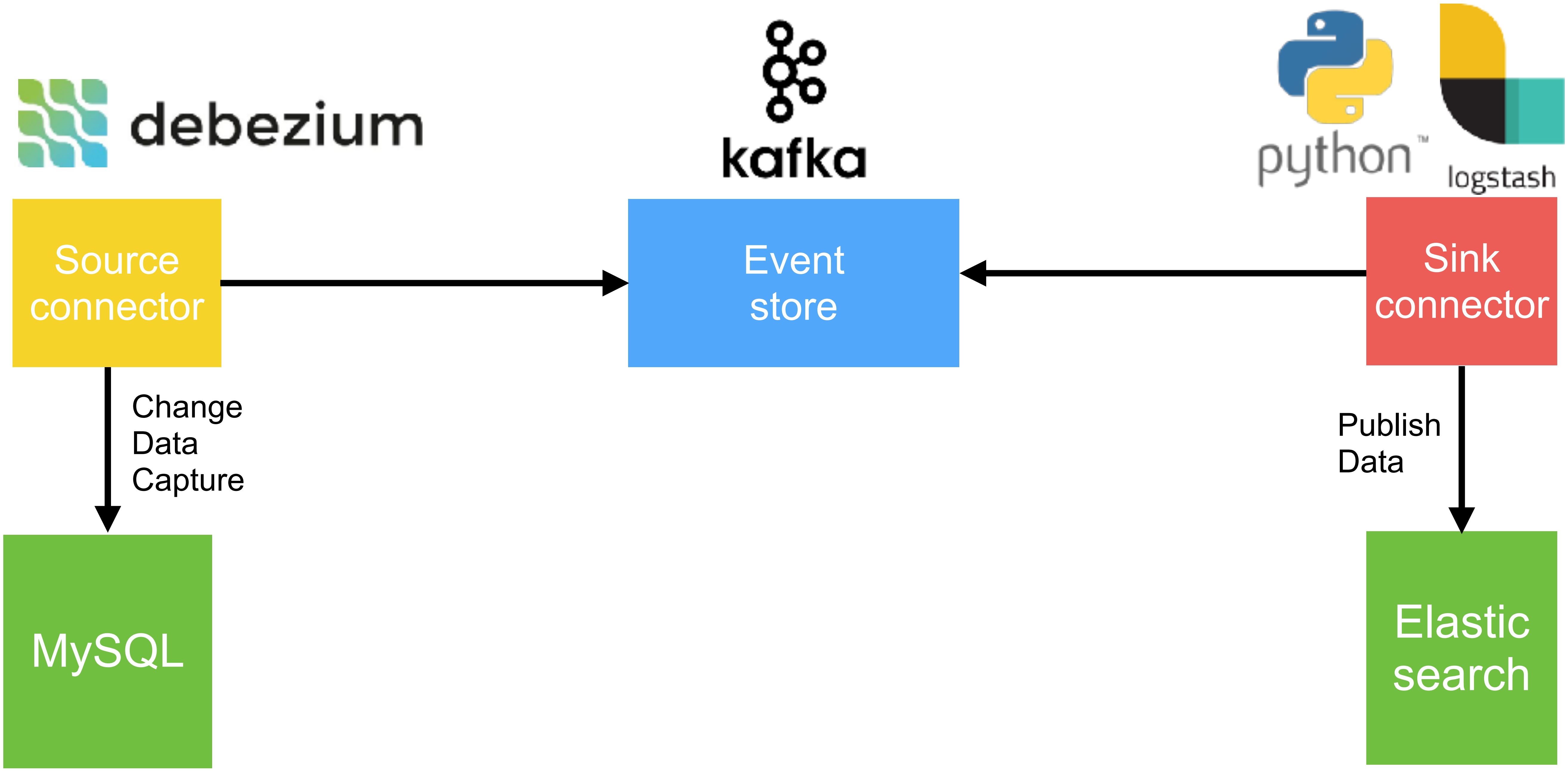
Real-time(NRT) process when data changed

Process !!



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

Example



debezium

kafka

python

logstash

Source connector

Event store

Sink connector

Change Data Capture

Publish Data

MySQL

Elastic search

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

Q/A