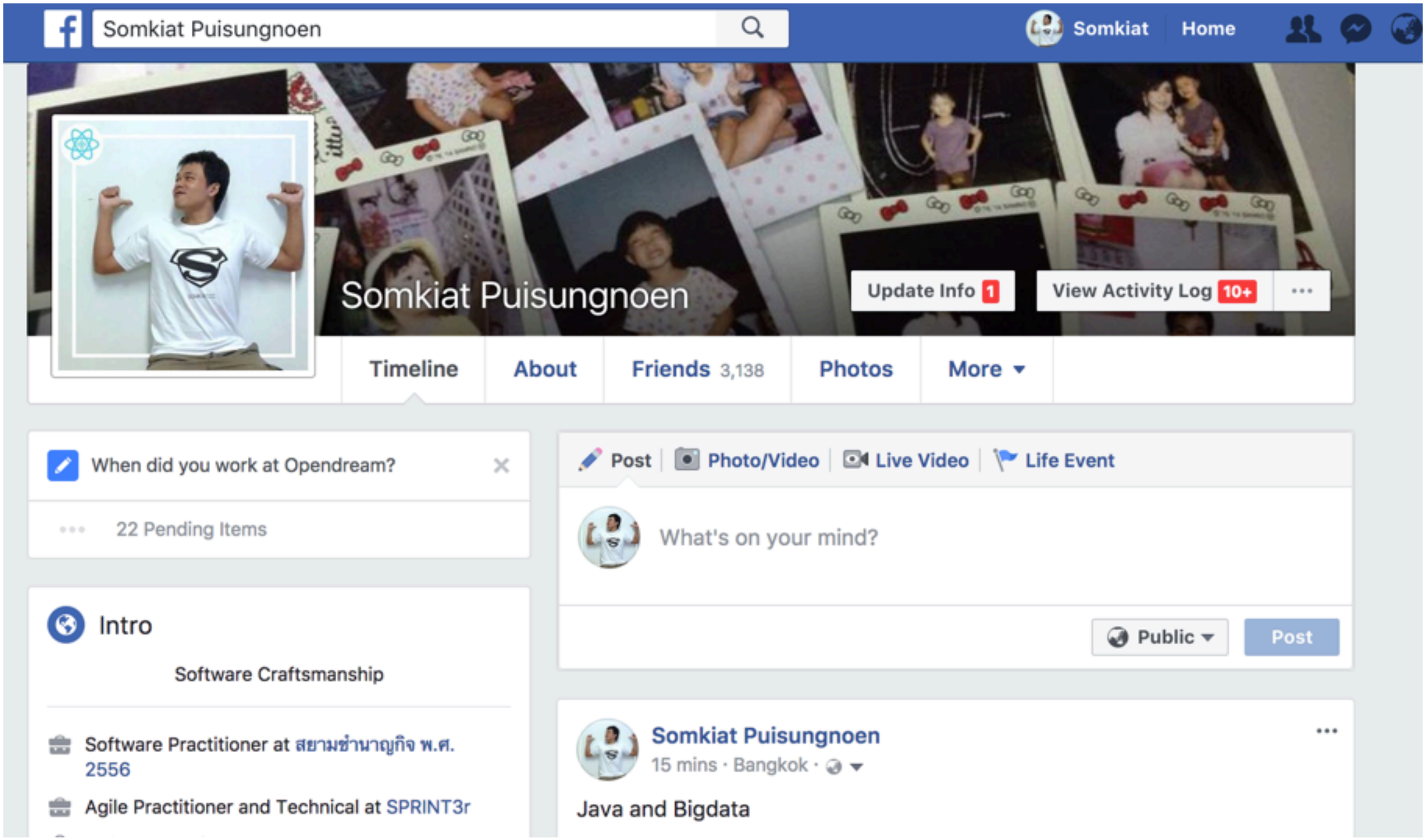
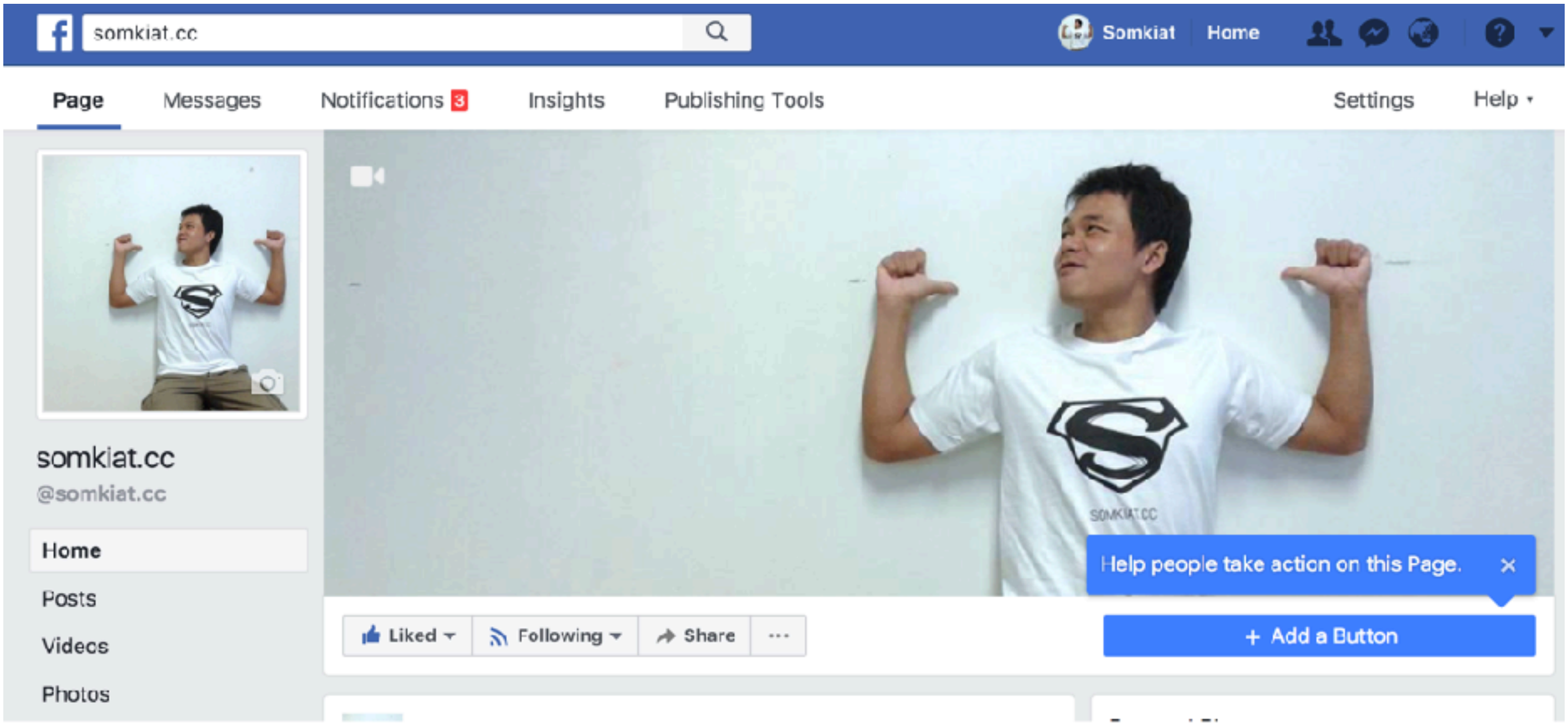
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Kafka workshop 2025





https://github.com/up1/course-kafka-2025

Topics (1)

- Why Apache Kafka ?
- What is Apache Kafka ?
- Architecture
- Topologies and Tools
- Use cases

Topics (2)

- Working with Kafka cluster
- Broker, Producer, Consumer
- Monitoring
- Perfomance testing

Topics (3)

- Event-driven with Kafka
- Develop with C#
- Testing
- Monitoring system

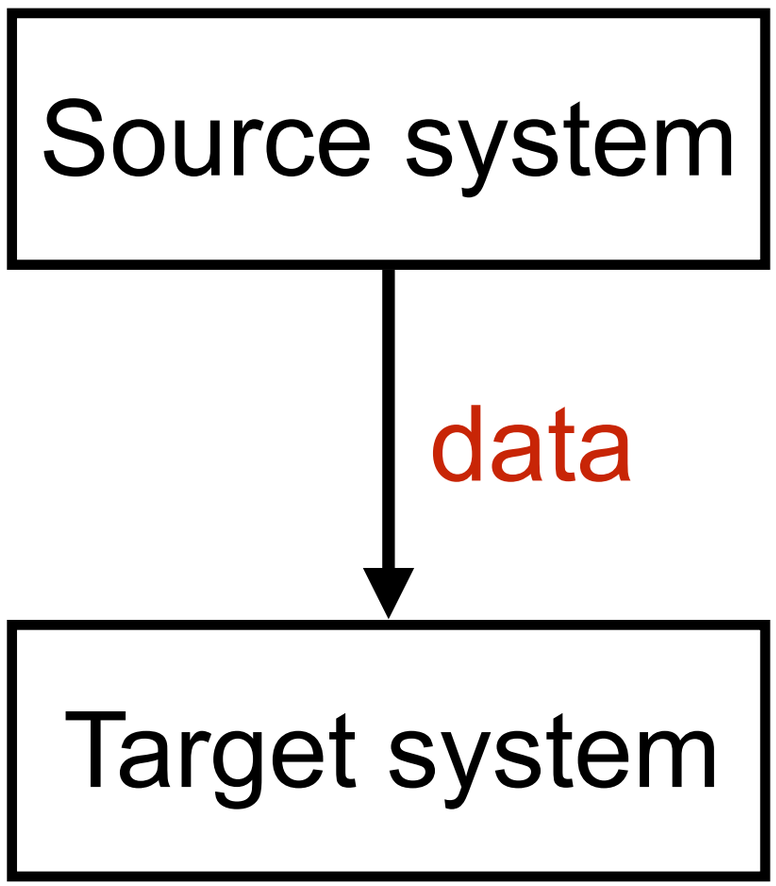
Learning paths

Part 1	Part 2	Part 3
Why, What ?	Configuration	Event-driven
Starting Kafka	Producers	Develop with C#
Architecture	Consumers	
Install and config	Monitoring	

https://kafka.apache.org/

Why ?

Starting with simple

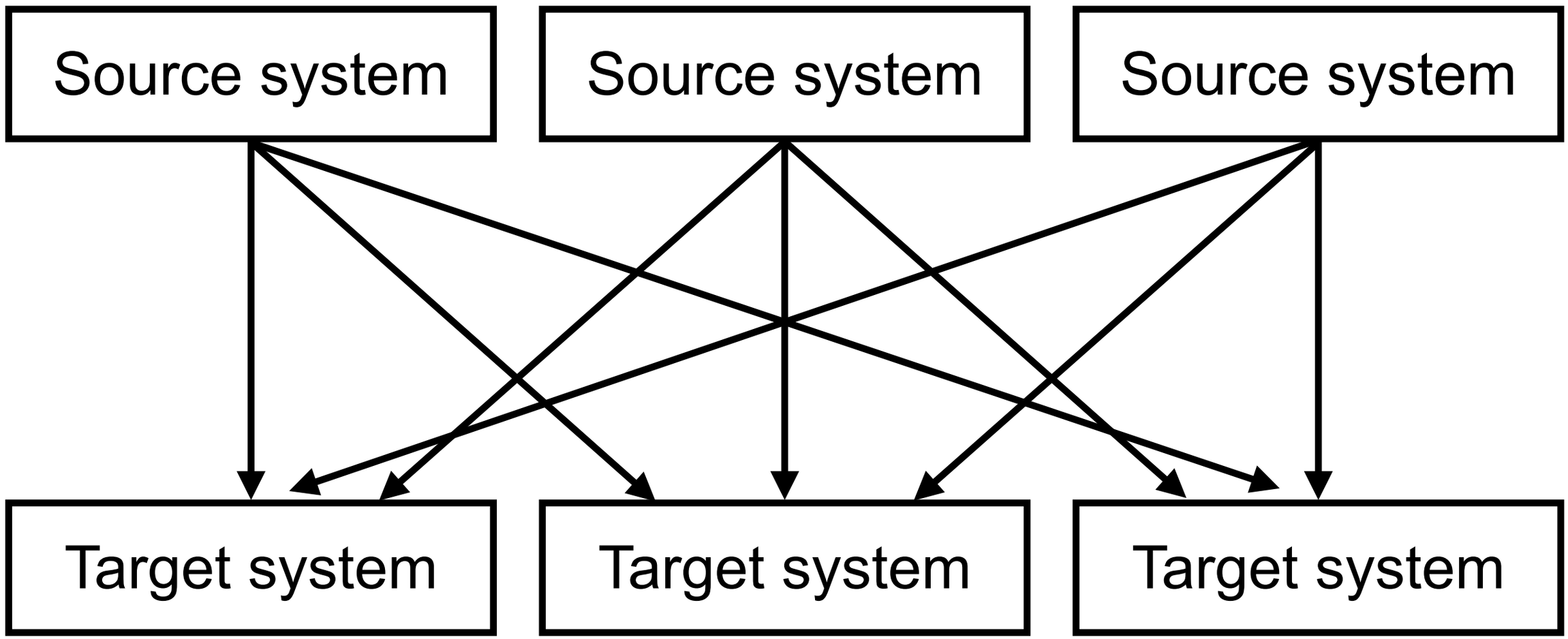


Why ?

More source systems
More target systems
More protocols
More data formats and schemas
More loads/traffics

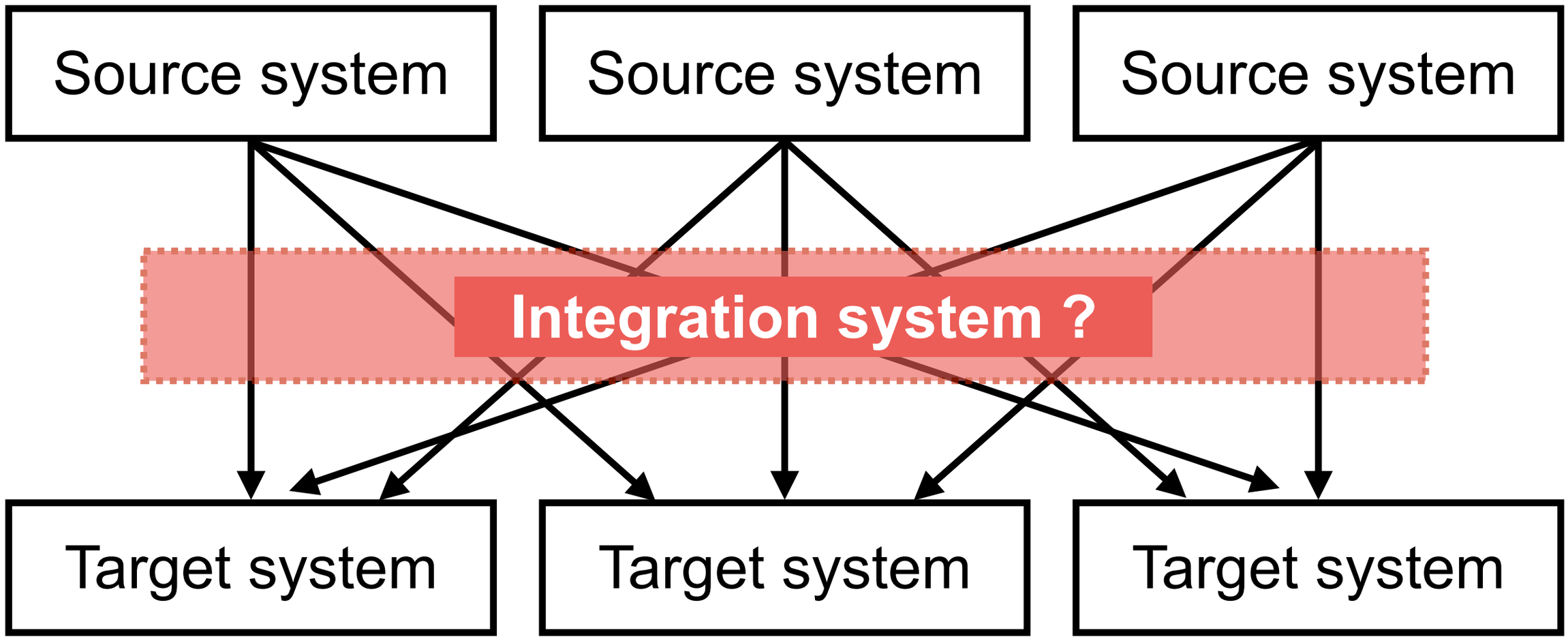
Why ?

Complex system and coupling !!



Why ?

Need integration system

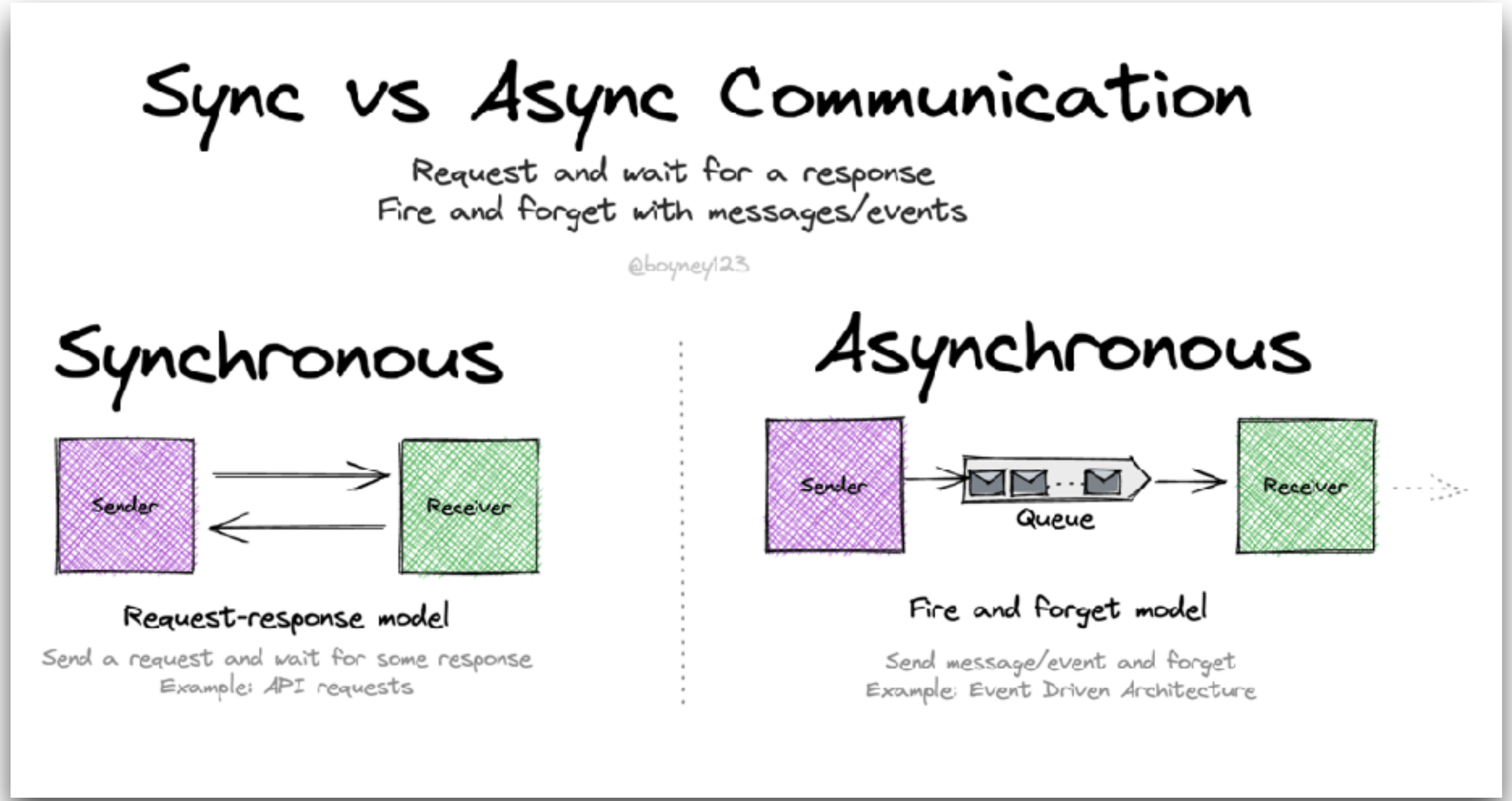


Integration ?

Protocols
Data format
Data schema and evolution

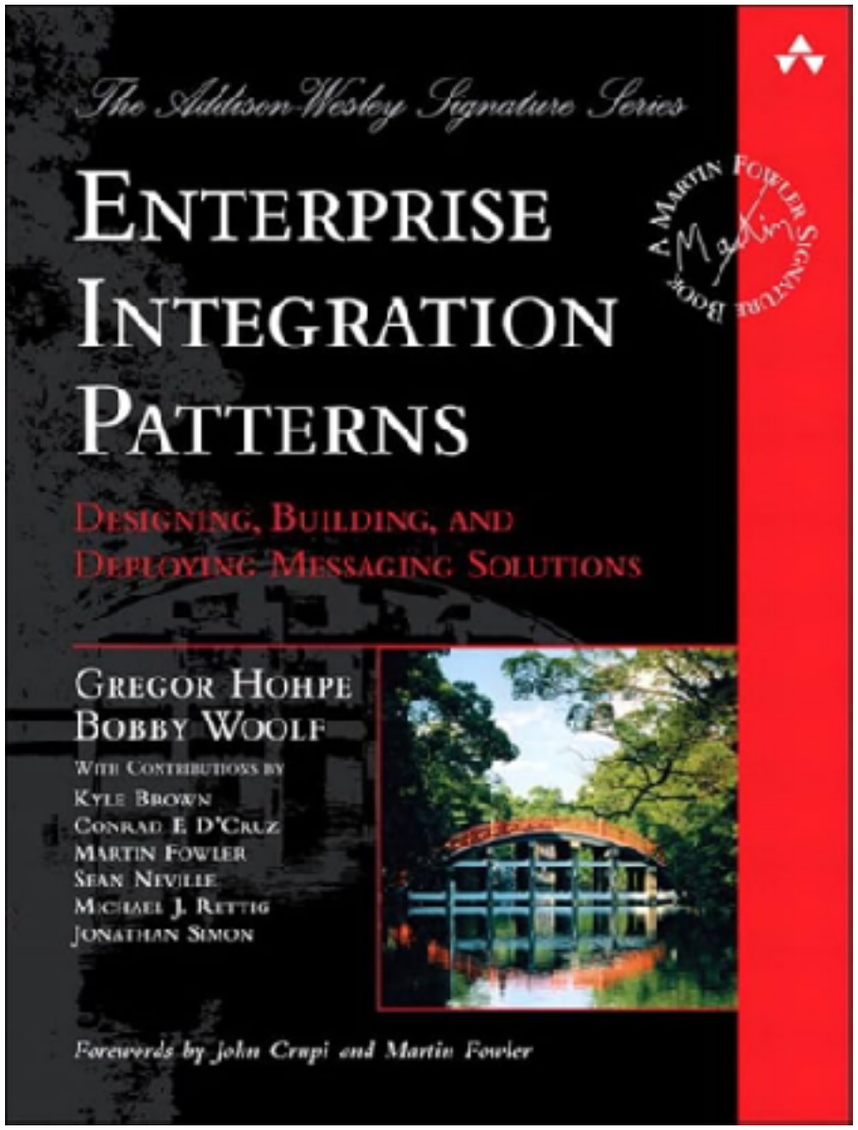
TCP HTTP REST JDBC	CSV JSON Avro Binary

Communication ?



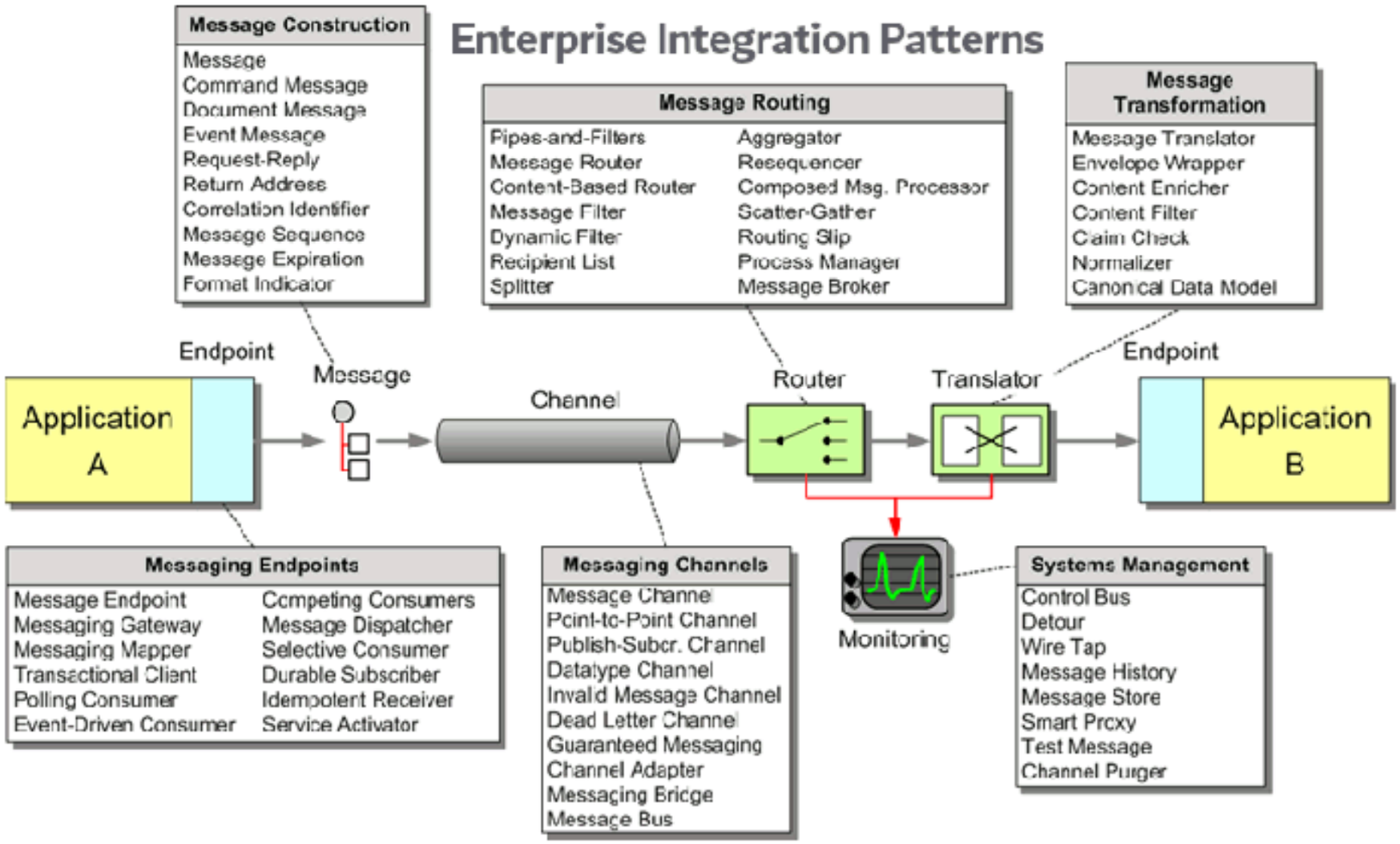
https://eda-visuals.boyney.io/visuals/sync-vs-async

Enterprise Integration Patterns



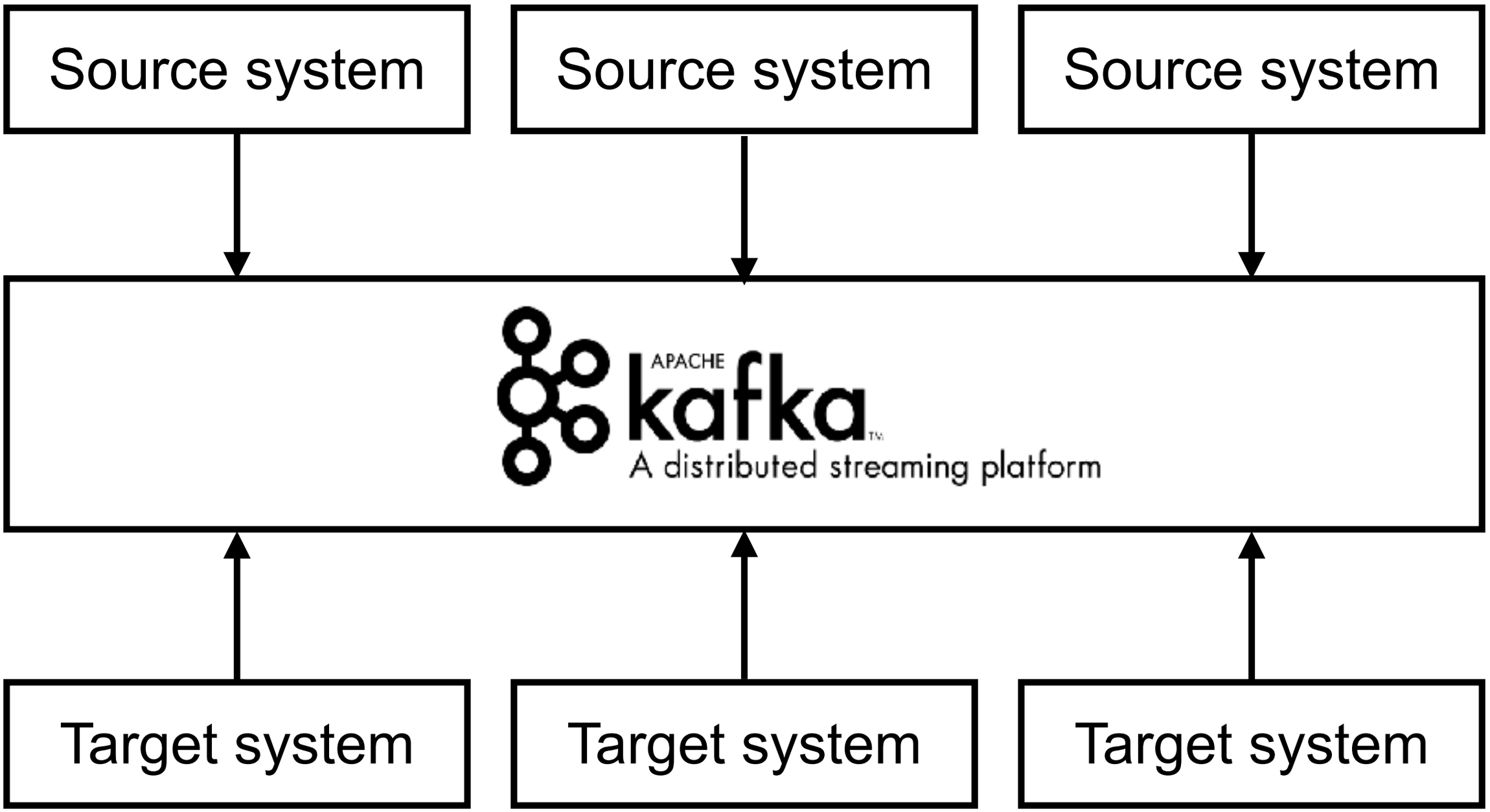
https://www.enterpriseintegrationpatterns.com/

Enterprise Integration Patterns

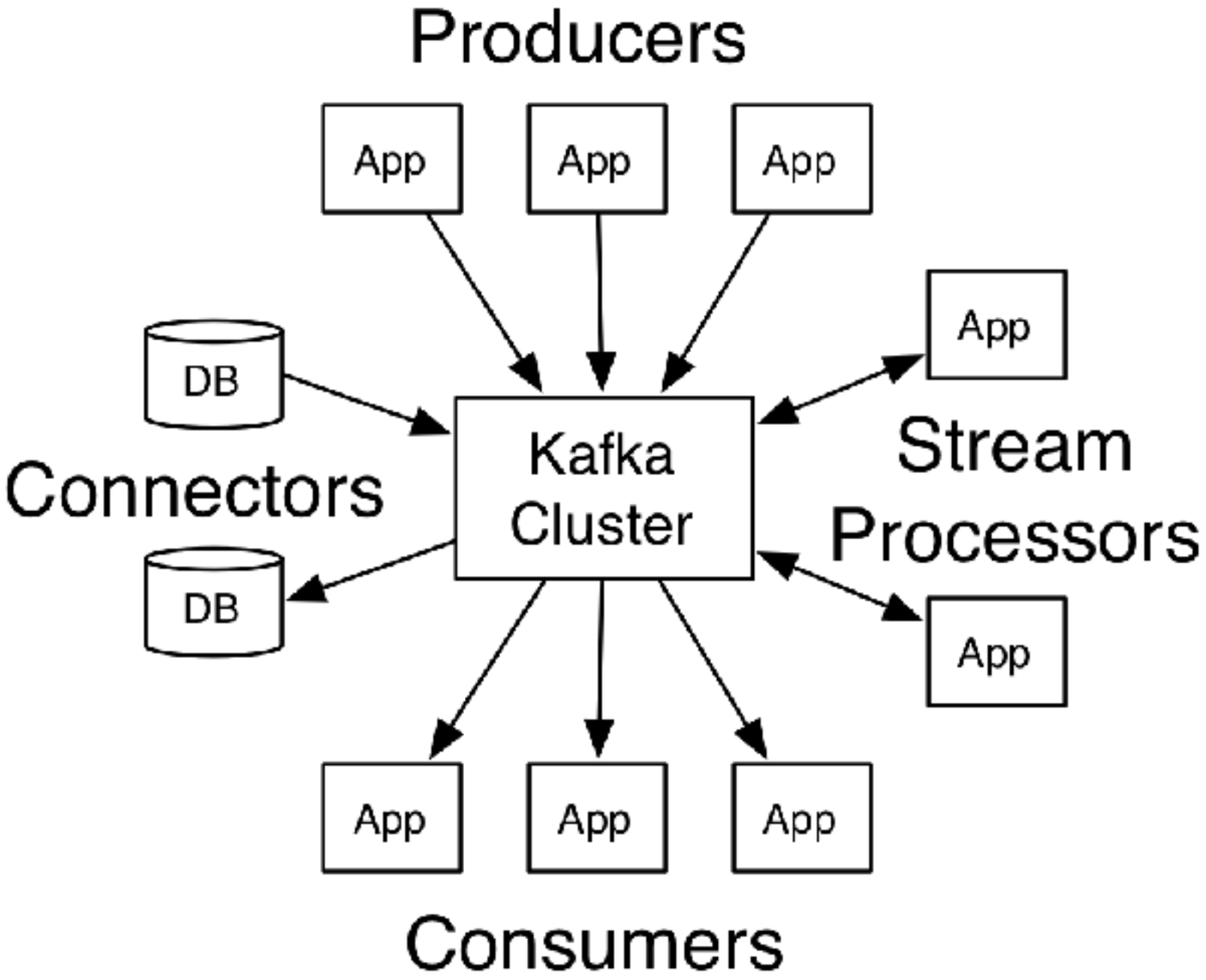


https://www.enterpriseintegrationpatterns.com/

Apache Kafka

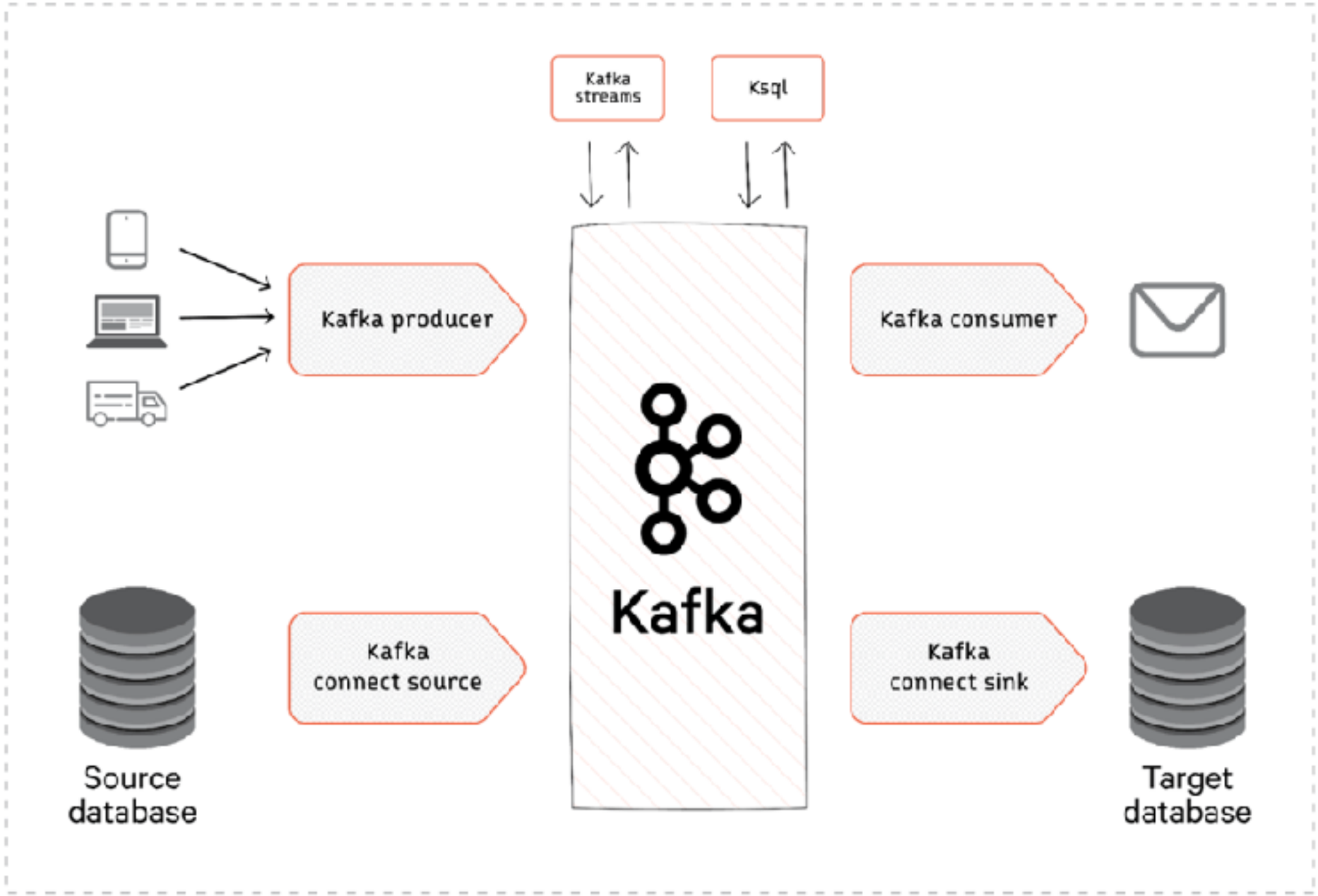


Apache Kafka



https://kafka.apache.org/intro.html

Apache Kafka



https://www.redpanda.com/guides/kafka-alternatives-kafka-api

What Apache Kafka ?

Created by LinkedIn
Open source project (maintain by Confluent)
Distributed system
Resilient architecture, **fault tolerant**
Horizontal scalability

High performance/ low latency

High throughput

Low latency

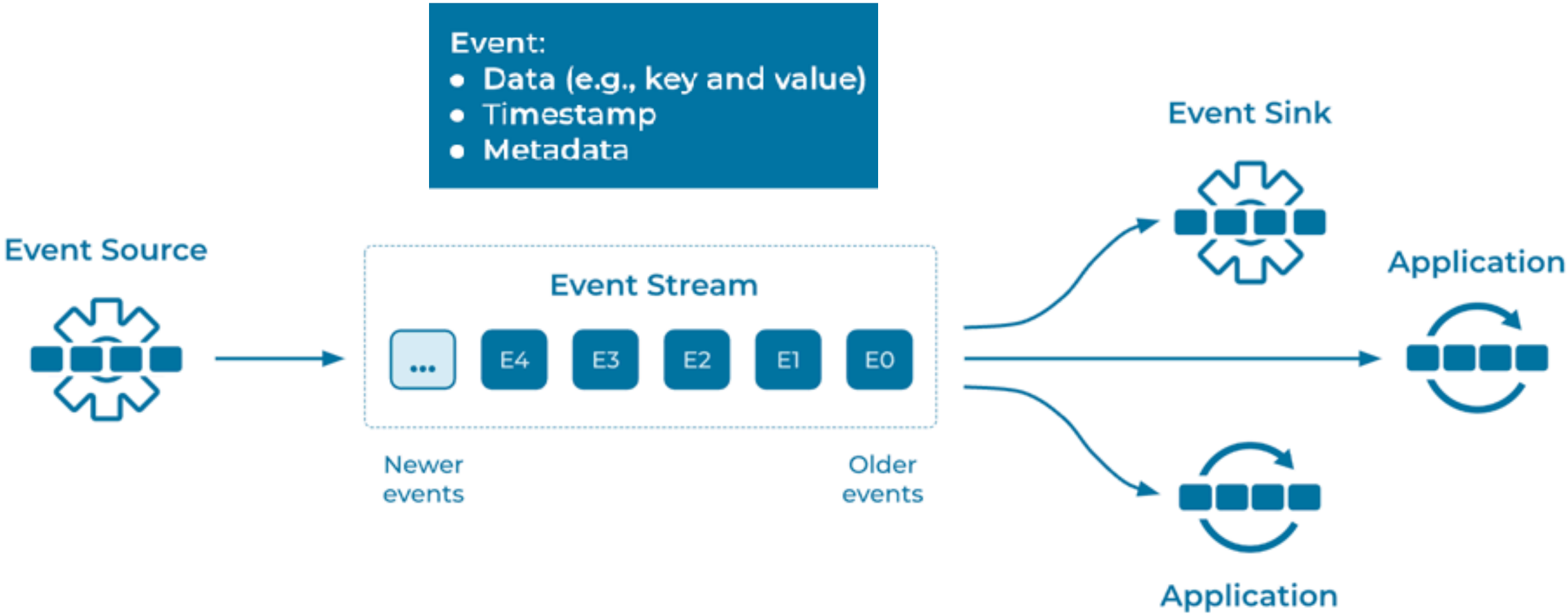
Goals ?

Data Integration

Decoupled system

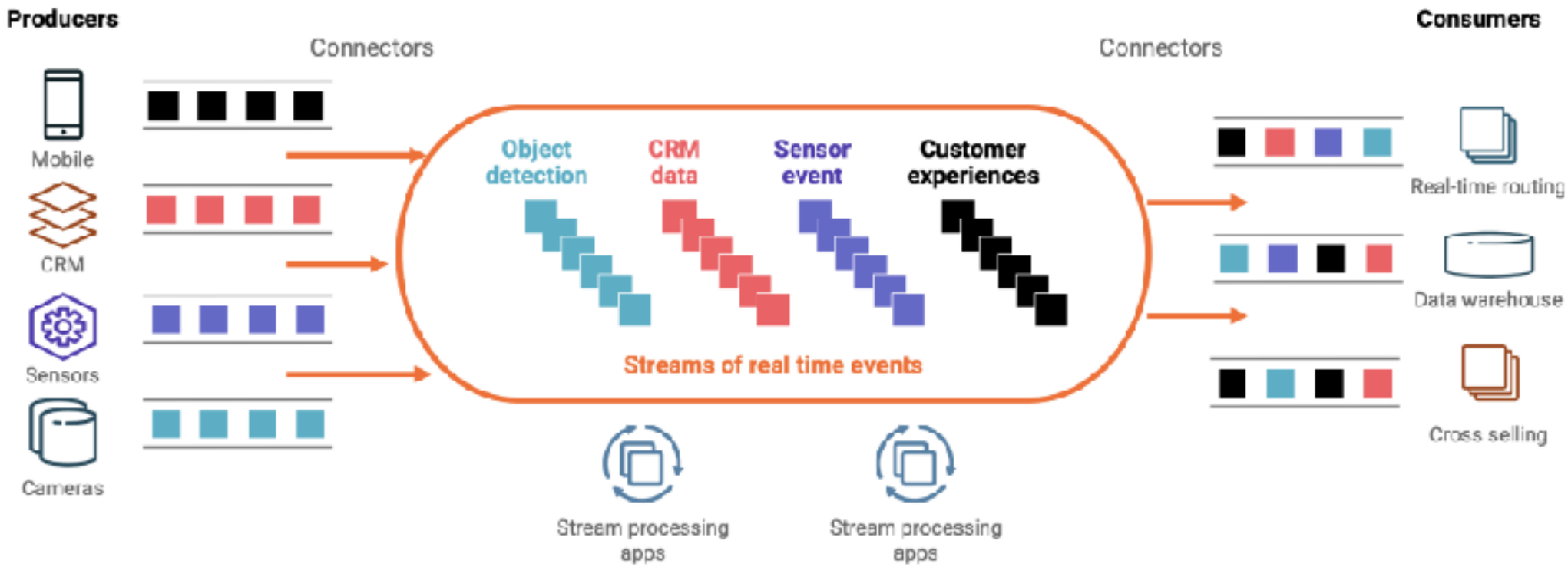
Open-source
distributed **event streaming** platform

Event Streaming



https://developer.confluent.io/patterns/event-stream/event-stream/

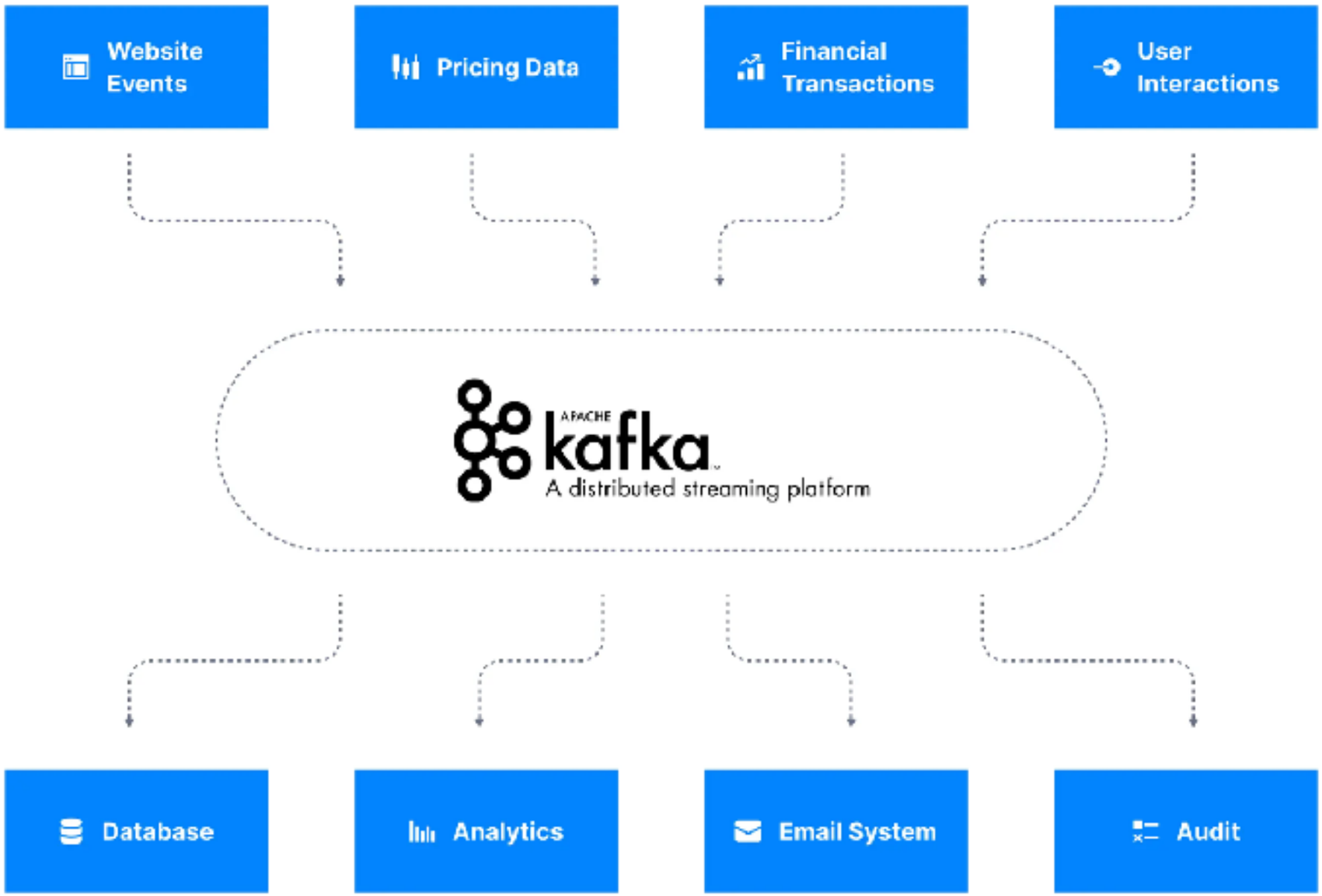
Event Streaming



Use cases

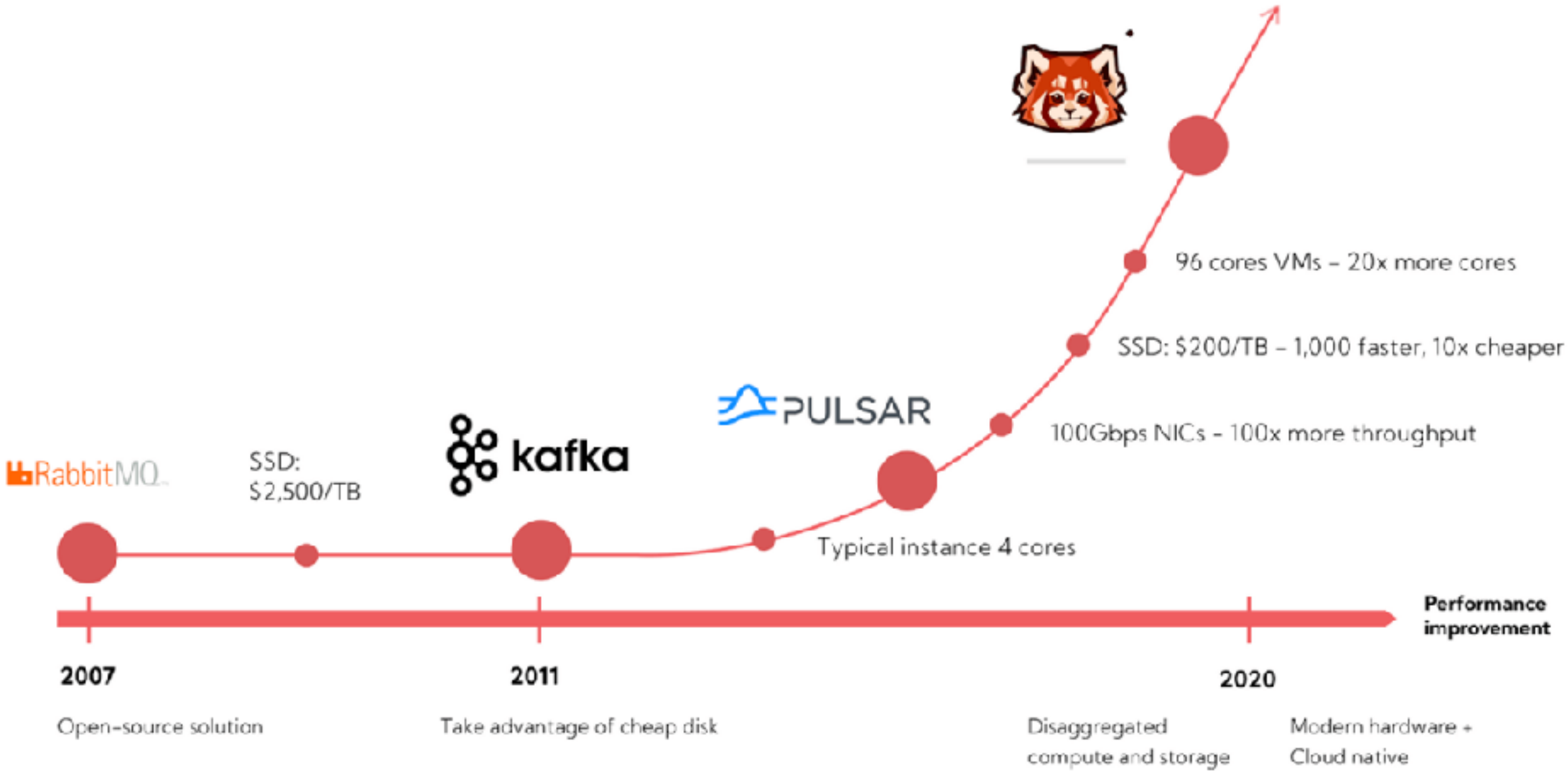
Messaging system
Activity tracking
Gather metric from different locations
Gather application logs
Stream processing
Decoupling of system dependencies
Integration with others system/technologies

Use cases



https://learn.conduktor.io/kafka/what-is-apache-kafka/

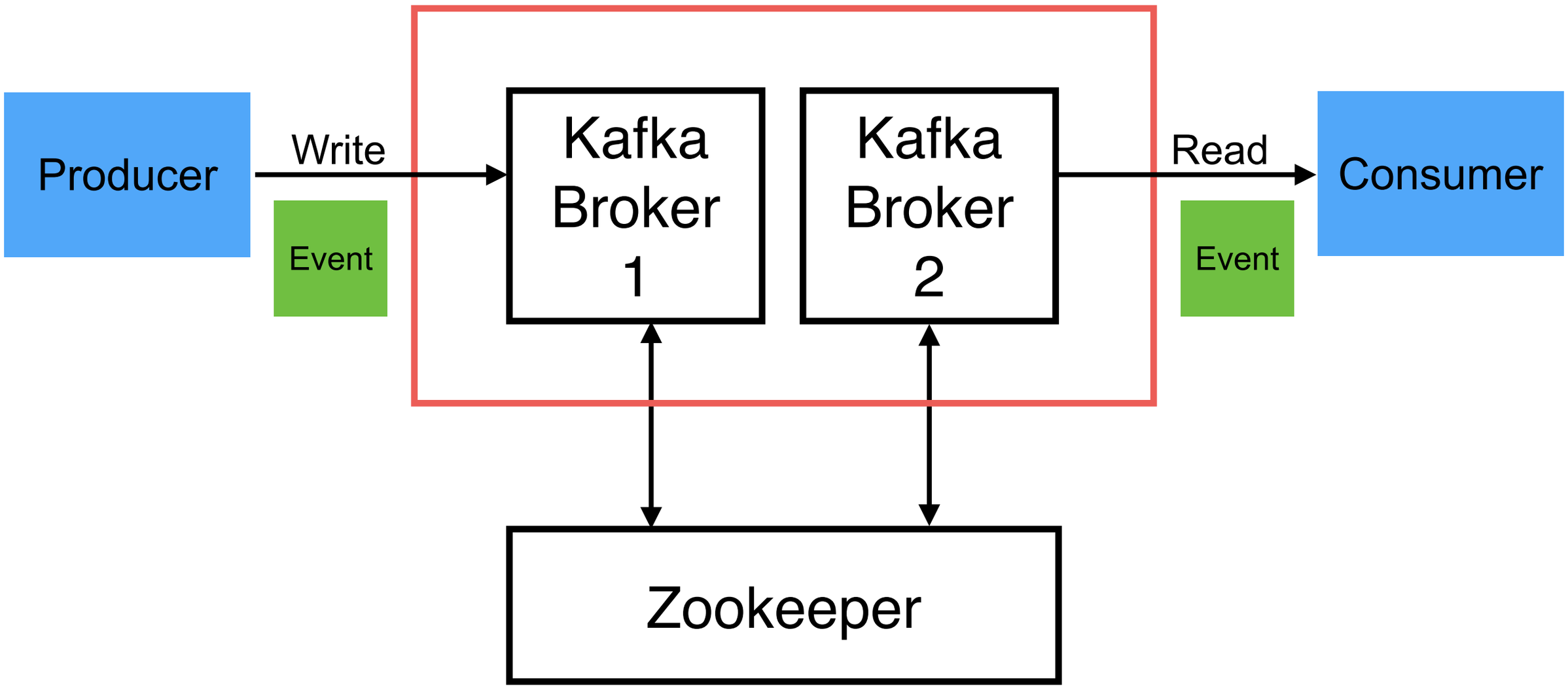
Alternative for Kafka



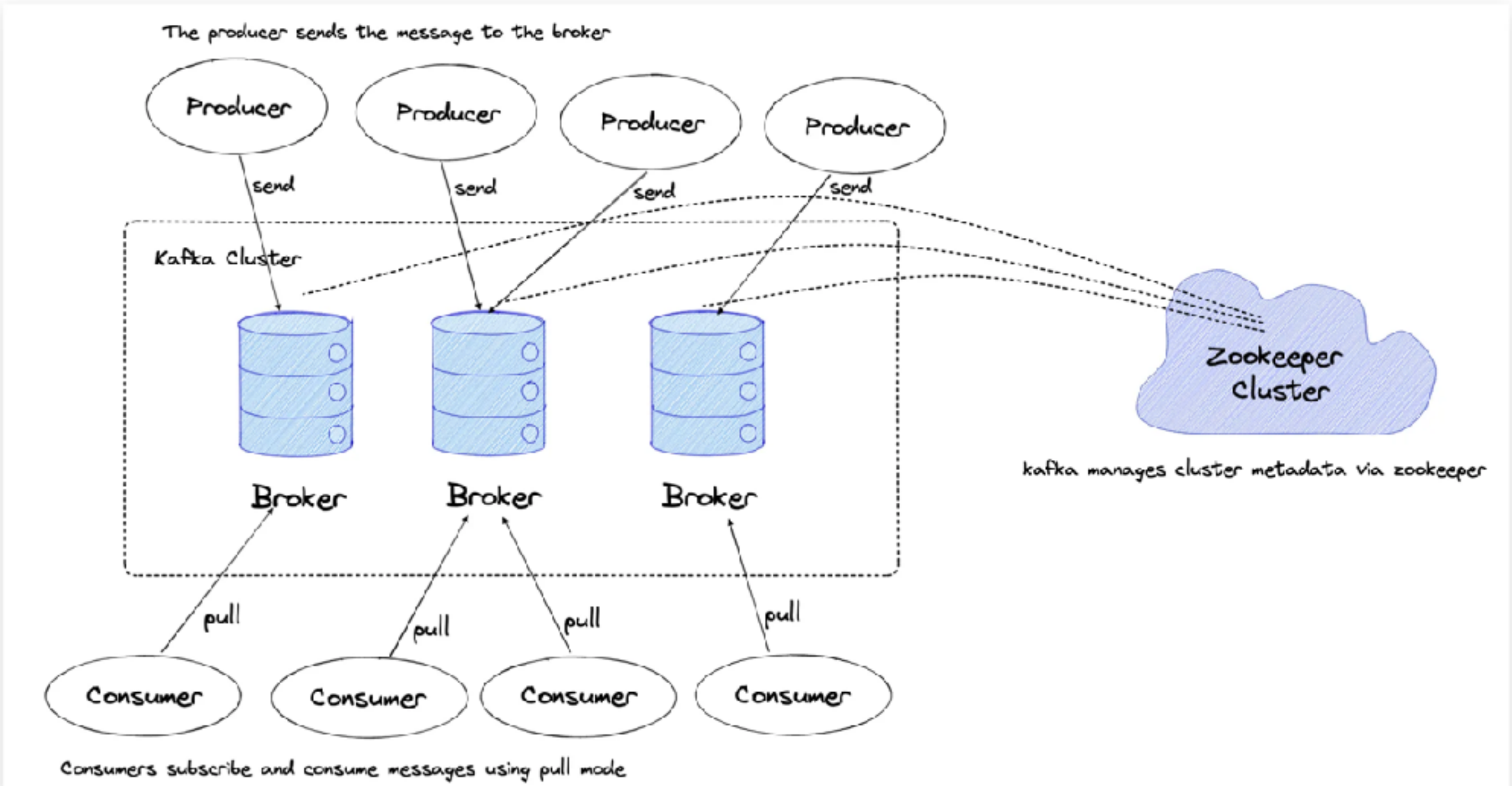
https://www.redpanda.com/guides/kafka-alternatives-kafka-api

Architecture of Kafka

Architecture (<4)



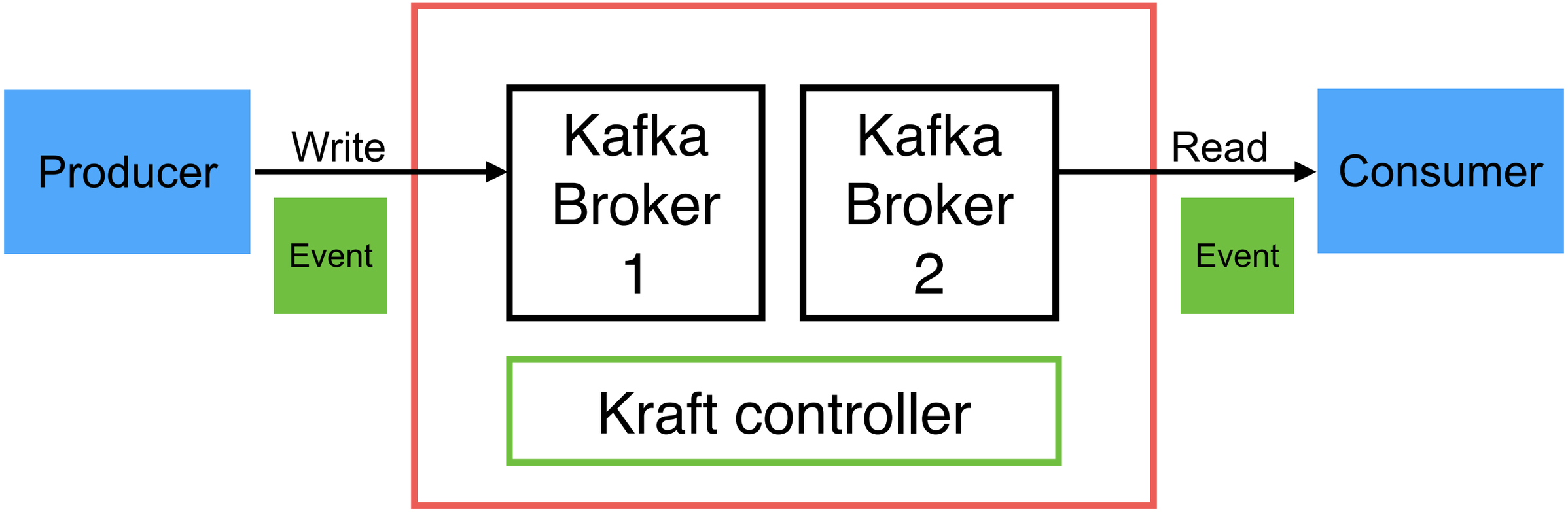
Architecture (<4)



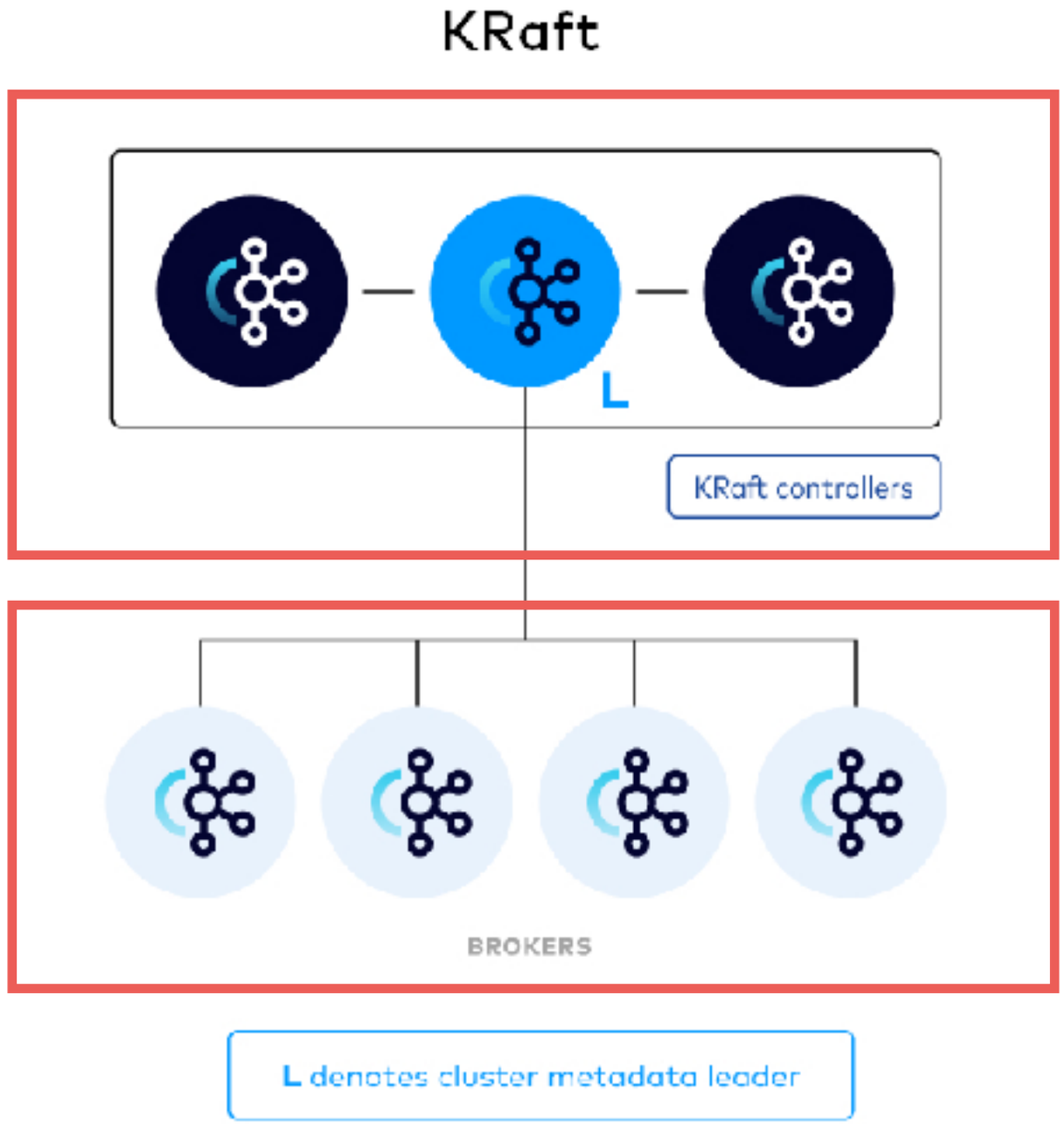
https://api7.ai/blog/why-kafka-needs-an-api-gateway

Architecture (4+)

KRaft (Kafka Raft Metadata Mode)
Remove Zookeeper
Simplify deployment, management and scaling



Kraft architecture



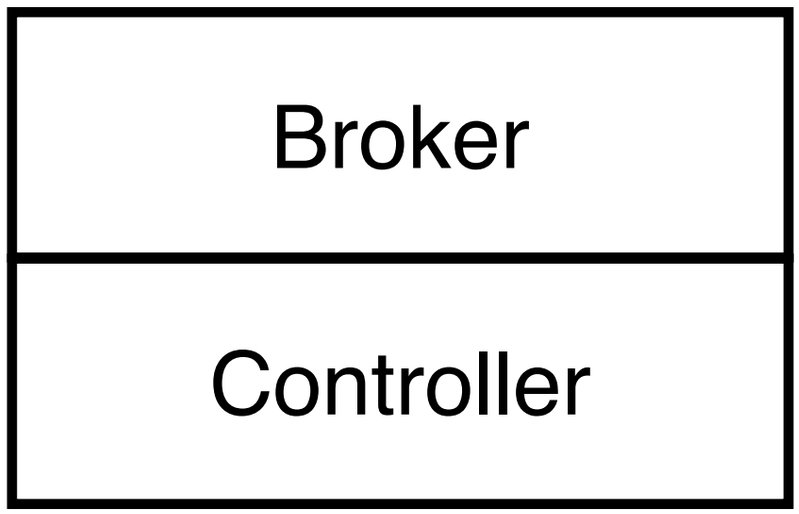
Controller role

Broker role

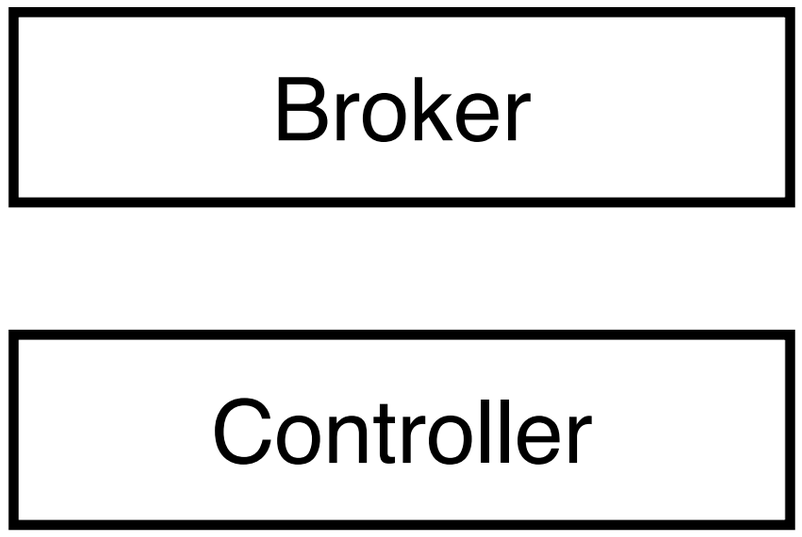
https://docs.confluent.io/platform/current/kafka-metadata/kraft.html

Kraft architecture

Combined mode

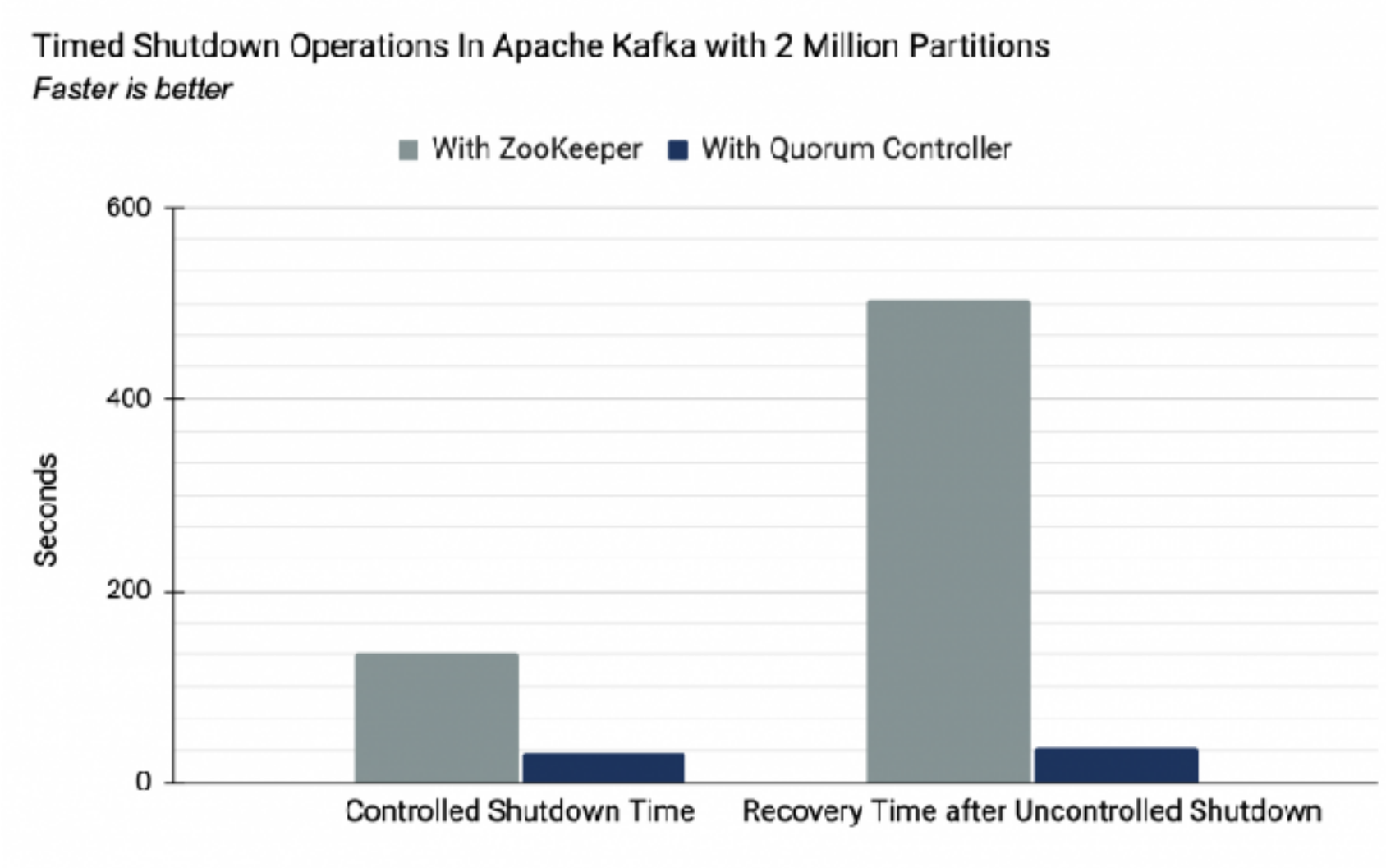


Isolated mode



https://docs.confluent.io/platform/current/kafka-metadata/kraft.html

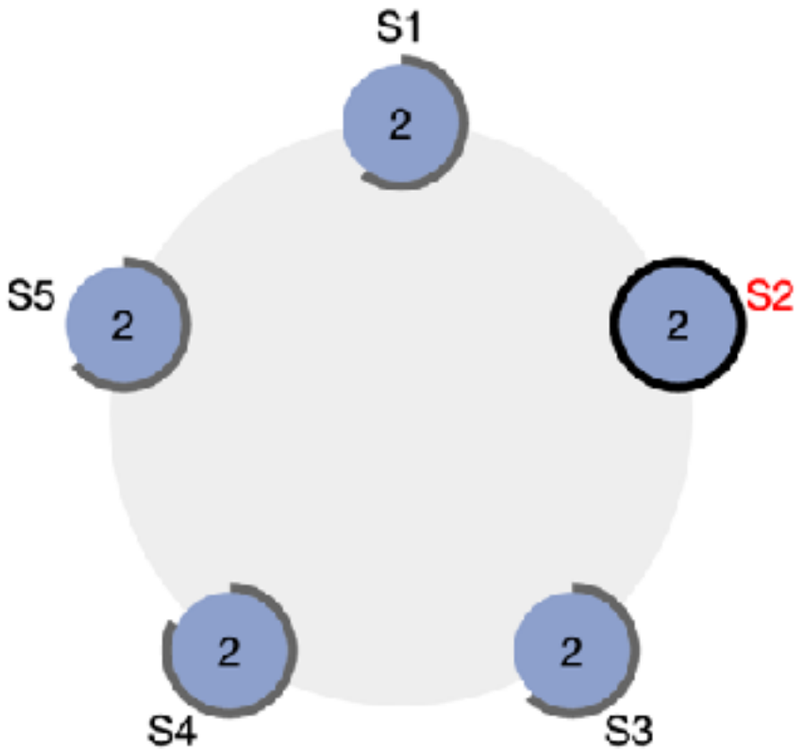
Performance !!

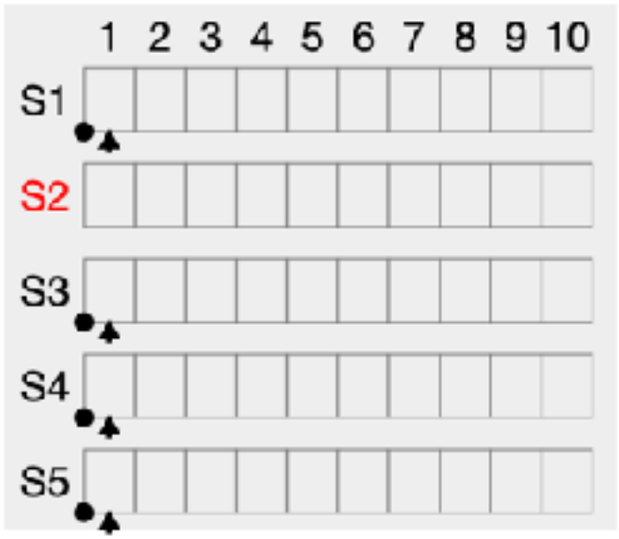


https://docs.confluent.io/platform/current/kafka-metadata/kraft.html

Raft Consensus algorithm

Manage state machine replication in distributed system
Maintain consistent state of cluster





https://raft.github.io/

Key point of Raft

Minimum server = 3

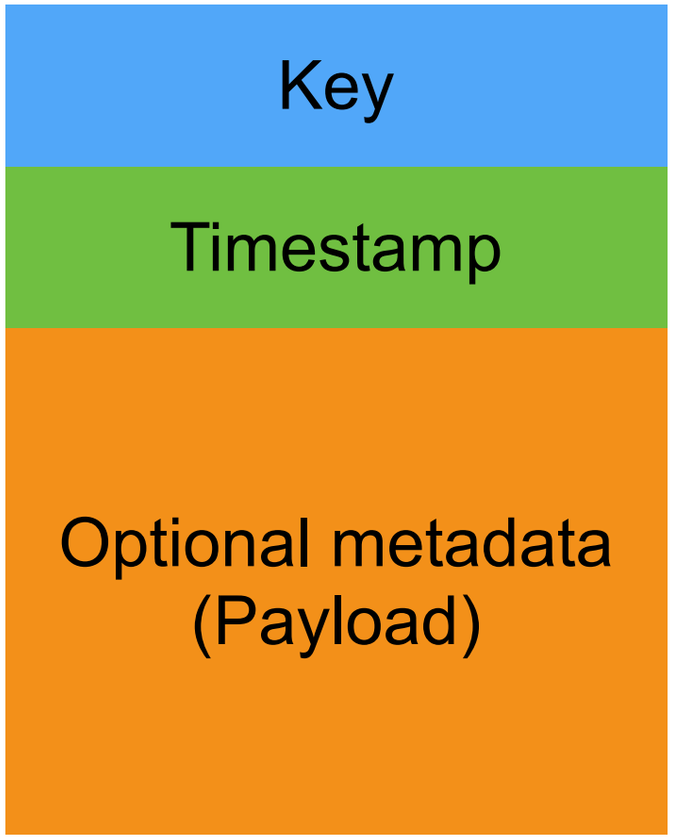
Odd number of servers

Kafka Fundamentals

Kafka Fundamentals

Event/record/message
Producers
Broker
Topic
Partition and offset
Consumer
Consumer groups
Controller

Event



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

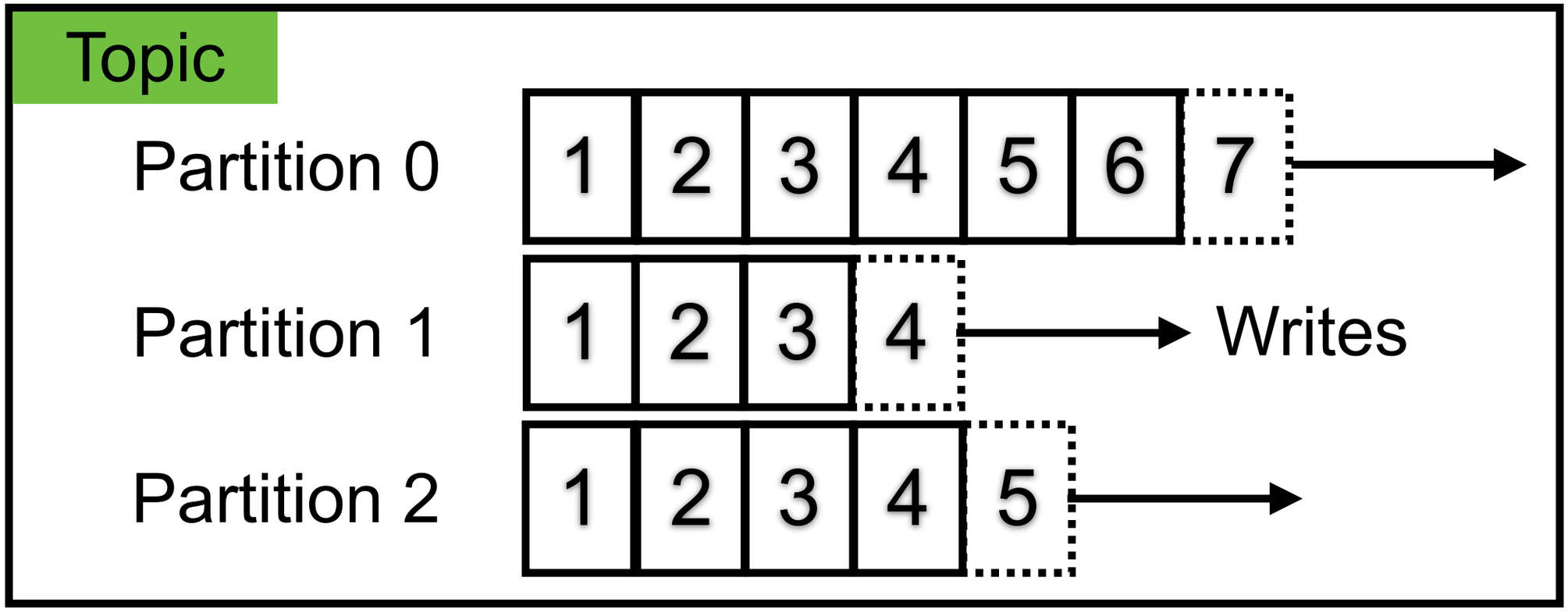
Topics

Stream of data
Similar to a table in database
You can have many topics as you want
A topic is identified by **name**

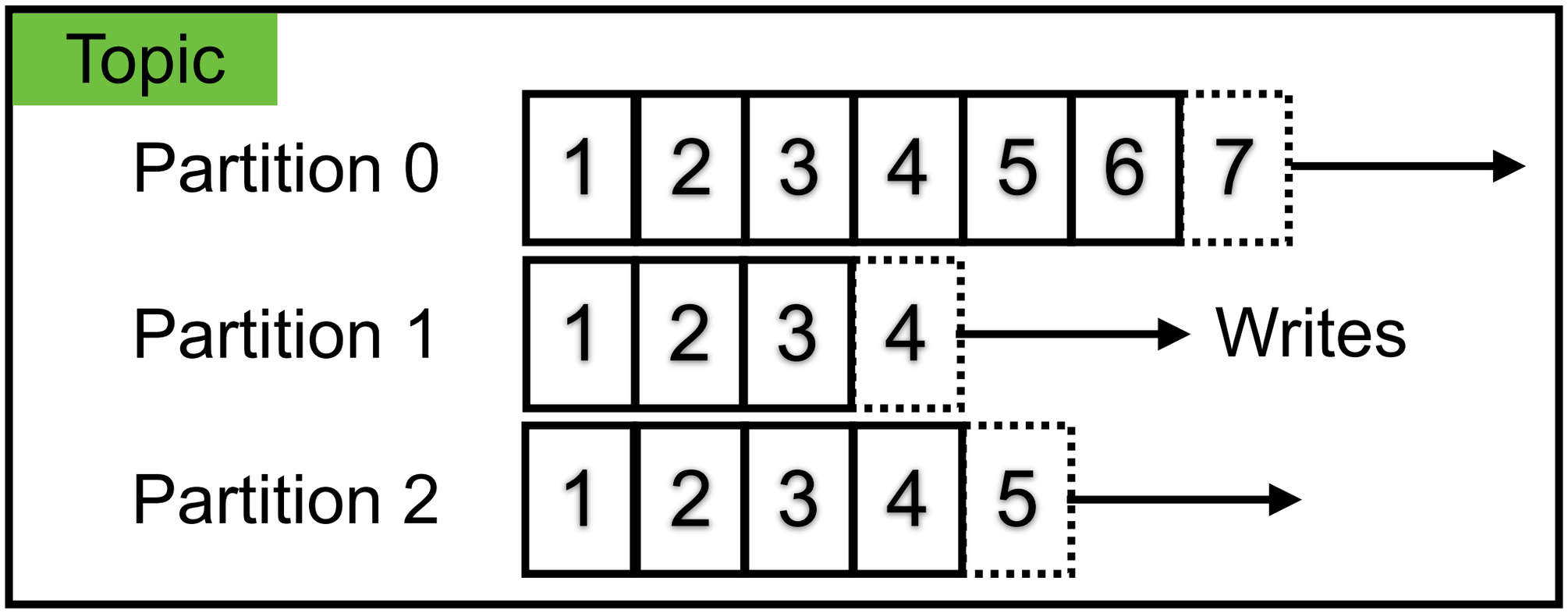
Topics, partitions and offsets

Partitions

Topics are split in partitions
Each partition is **ordered**
Each message in partition get incremental id (**offset**)



Topics, partitions and offsets



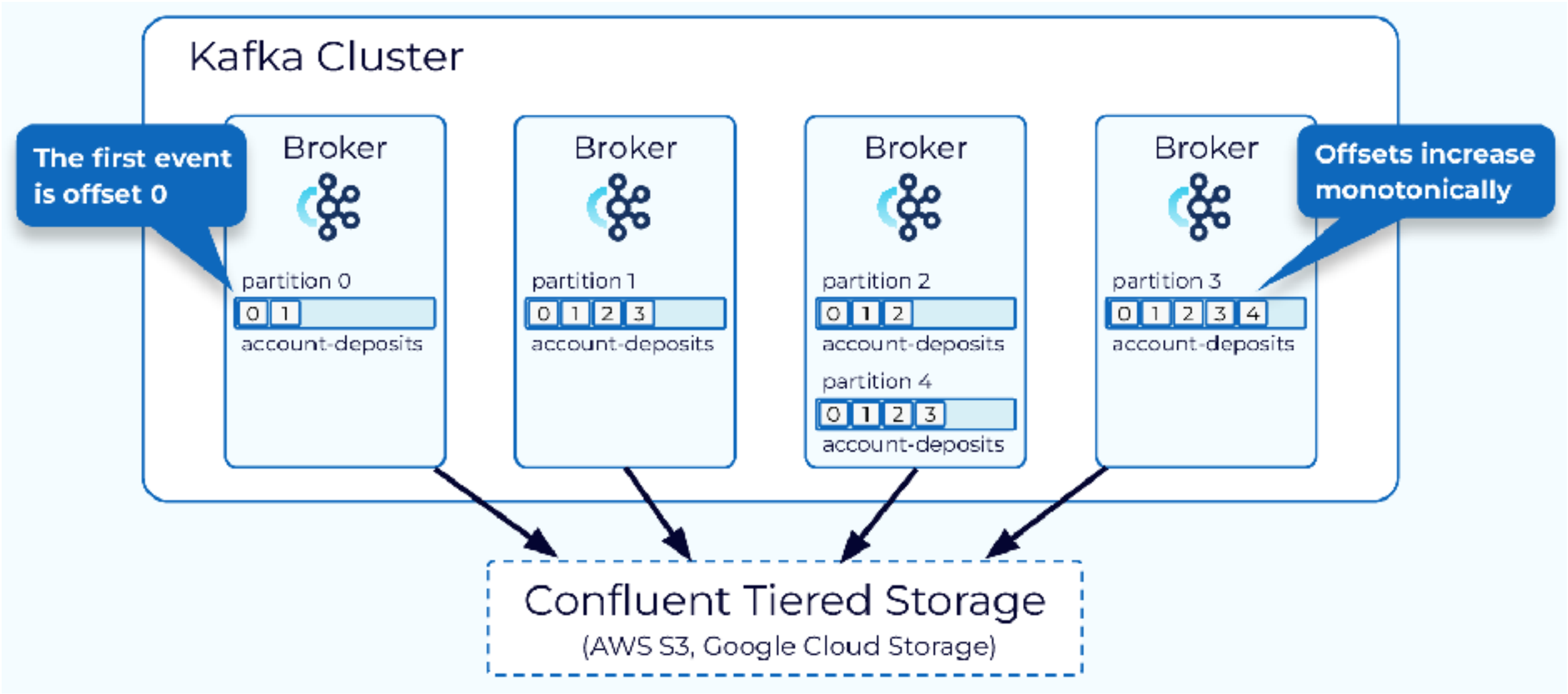
Order is guaranteed only within partition

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

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

Data is assigned **randomly** to a partition

Topics, partitions and offsets



https://developer.confluent.io/courses/architecture/get-started/

More partitions ?

Higher is your data throughput
More open files
Longer downtime
More RAM is consumed by clients

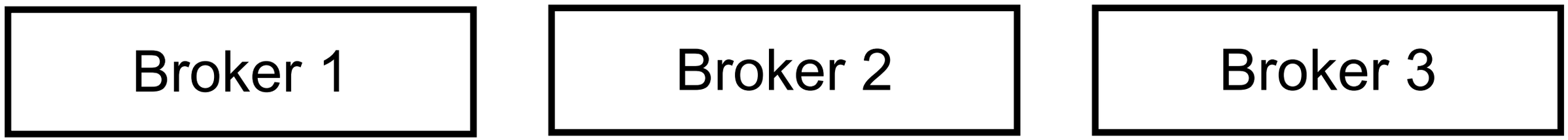
4000 per broker	200,000 per cluster	50 brokers per cluster

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

Brokers and topics

Brokers

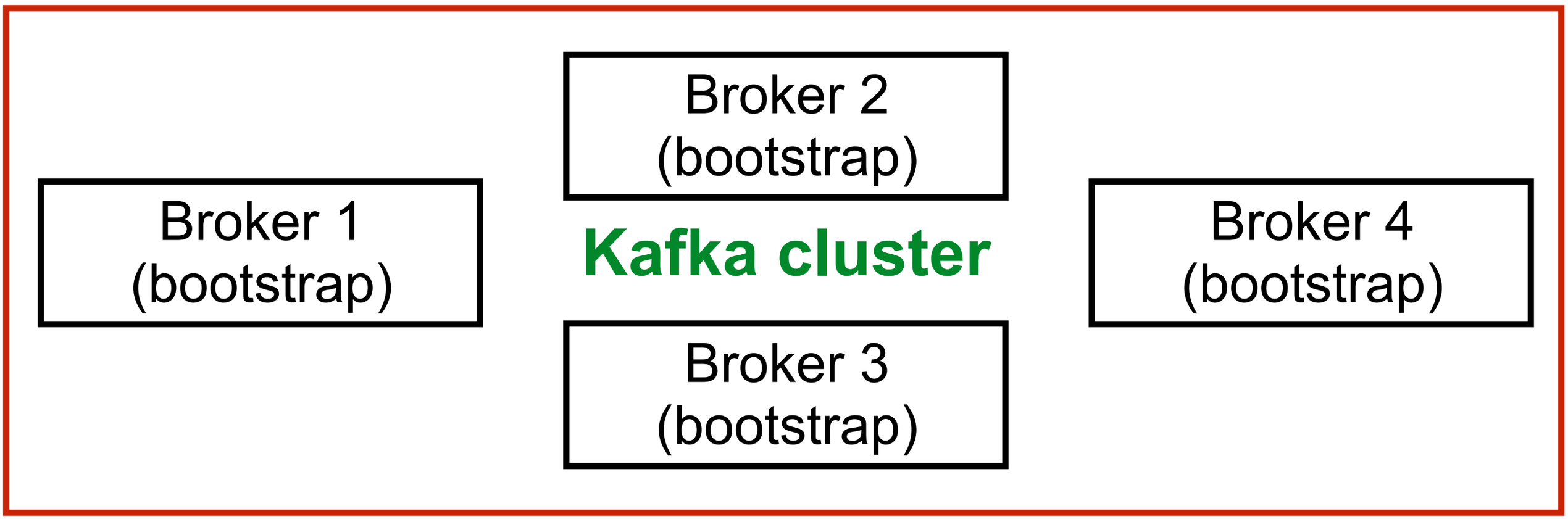
Kafka cluster is composed of multiple brokers (servers)
Each broker is identified by ID (integer)
Each broker contains certain topic partitions
After connecting any broker (**bootstrap broker**), you will connected to the entire cluster



Kafka broker discovery

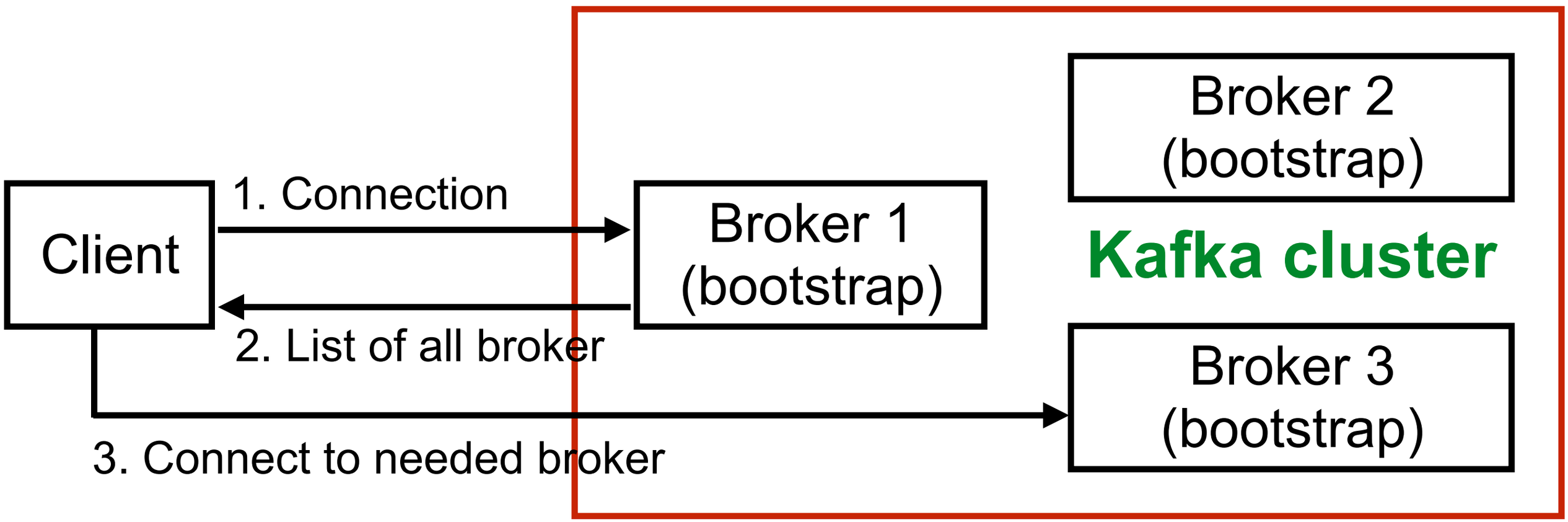
Every Kafka broker is called **“bootstrap server”**

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 can not work without Zookeeper !!

APACHE 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)

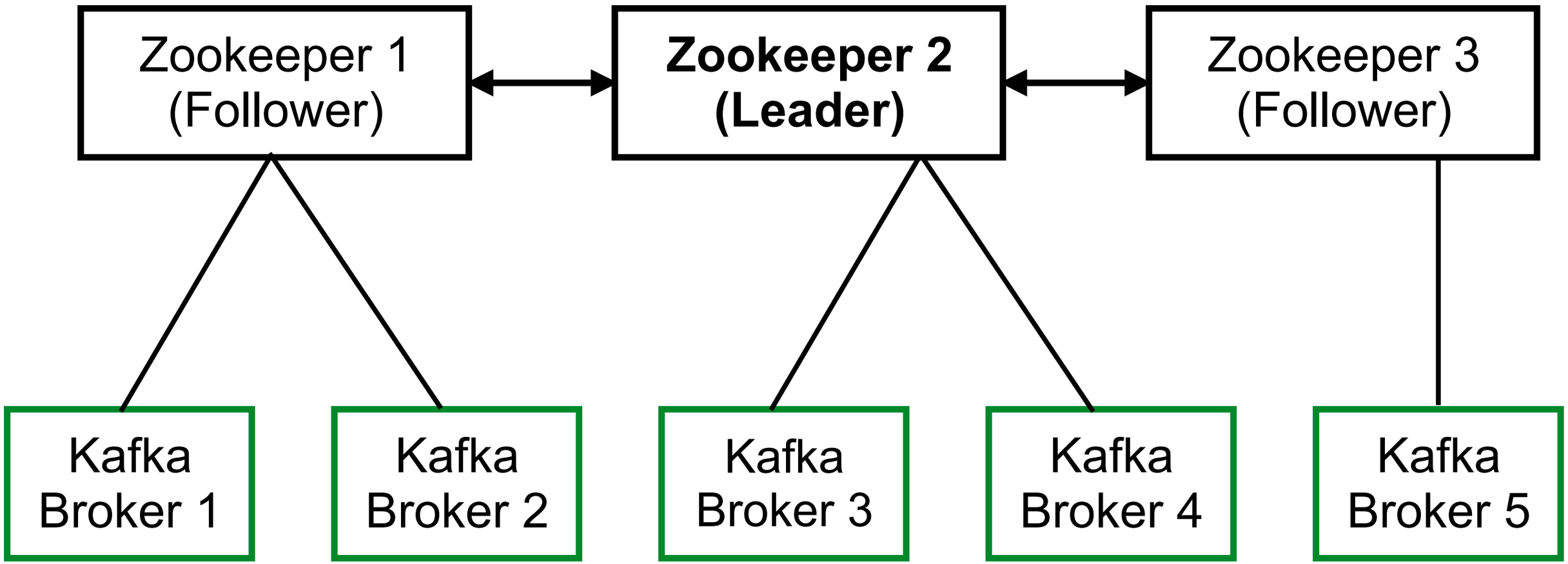
Data in Zookeeper

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

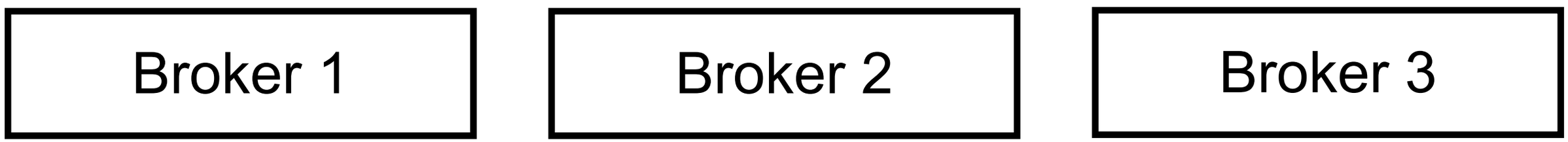
Zookeeper architecture



Brokers and topics

Brokers

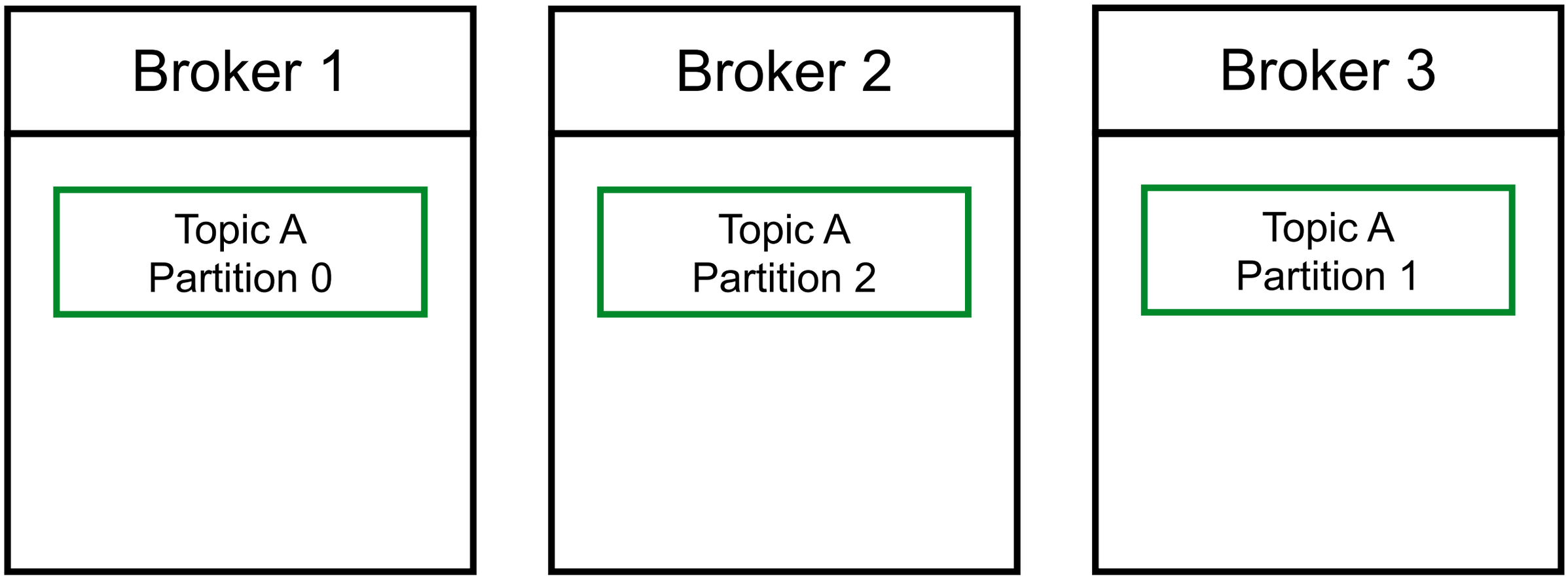
Good number to start id **3 brokers**



Brokers and topics

Example

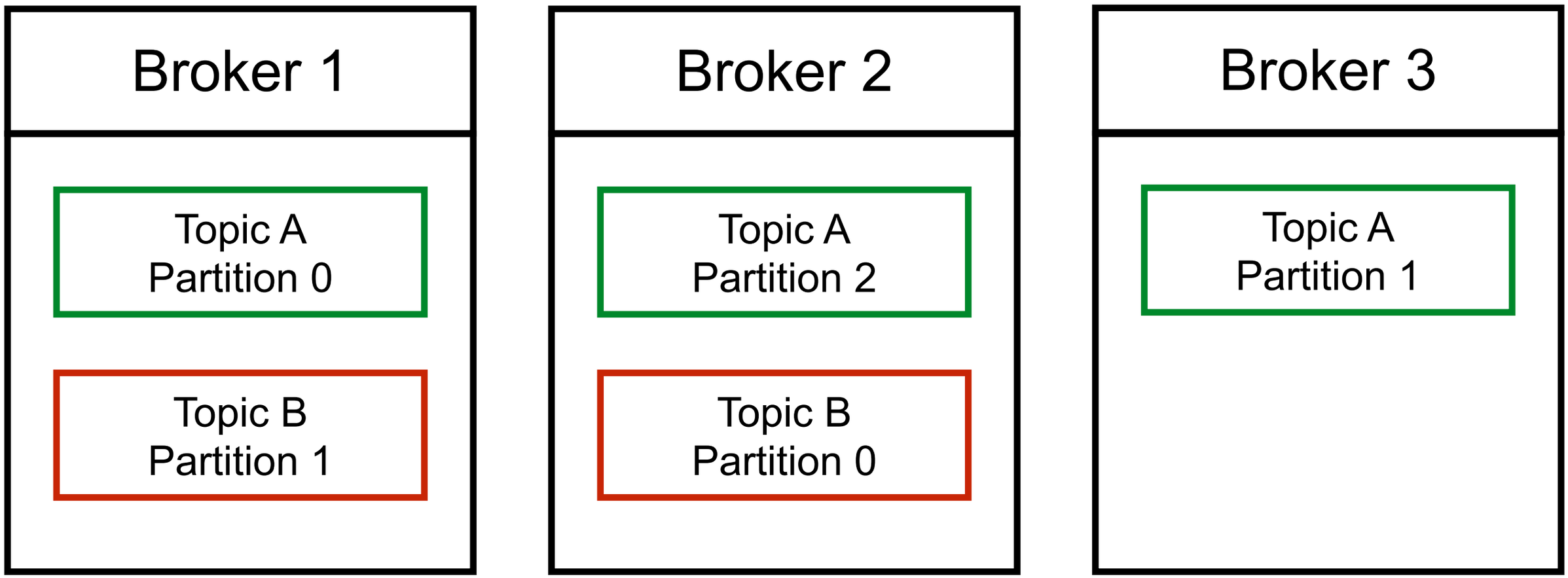
Topic A with 3 partitions



Brokers and topics

Example

Topic B with 2 partitions



Topic with replication factor

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

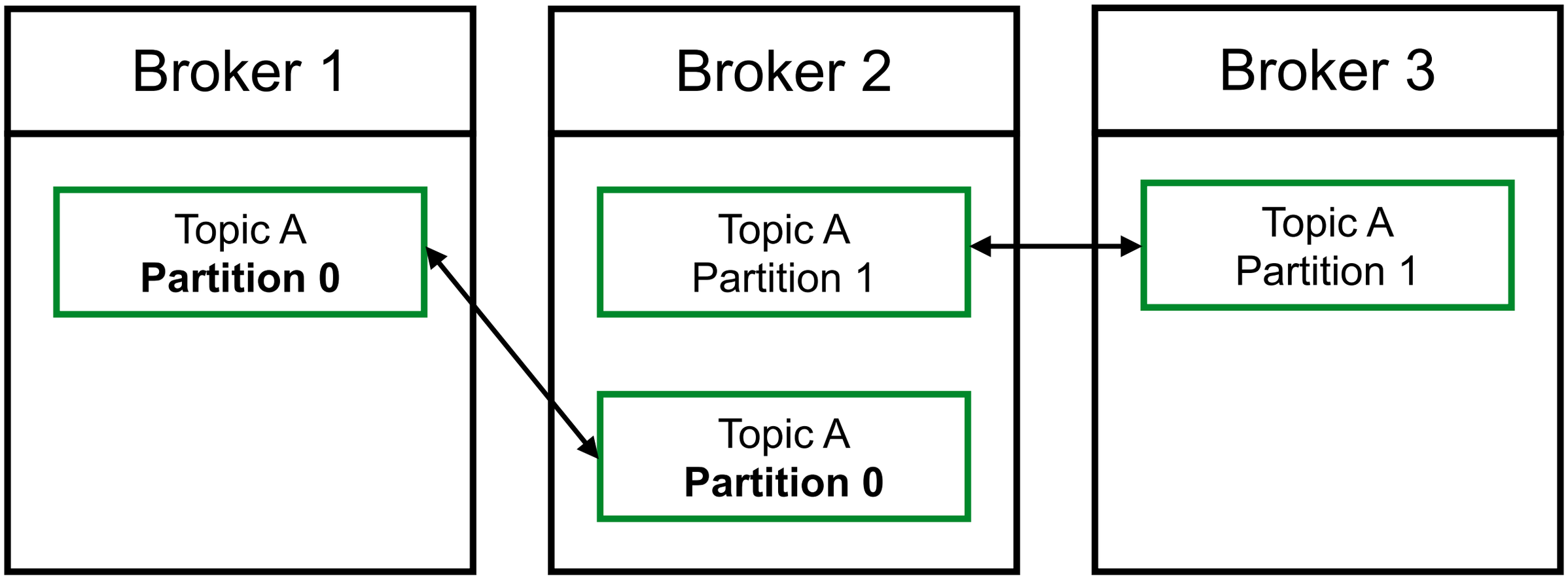
replica factor < no. of brokers

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

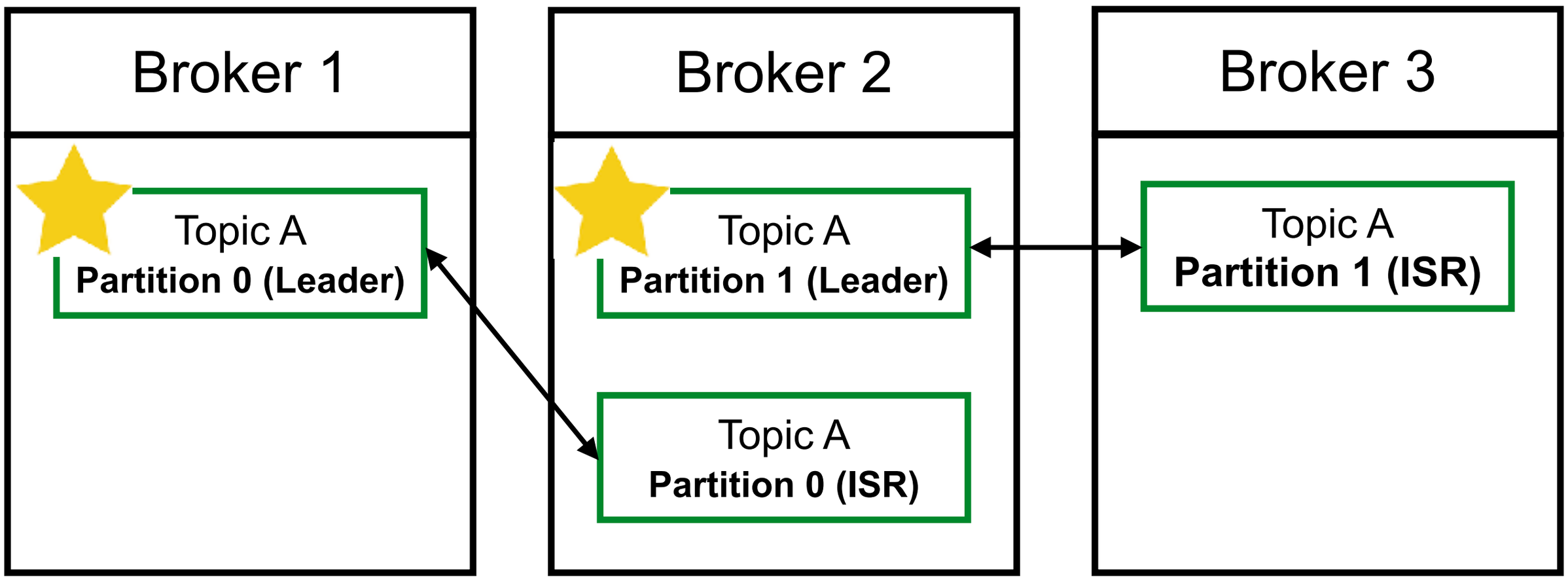
Other brokers will synchronize the data

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

Leader for a partition

Example

Topic A with 2 partitions and replication factor = 2



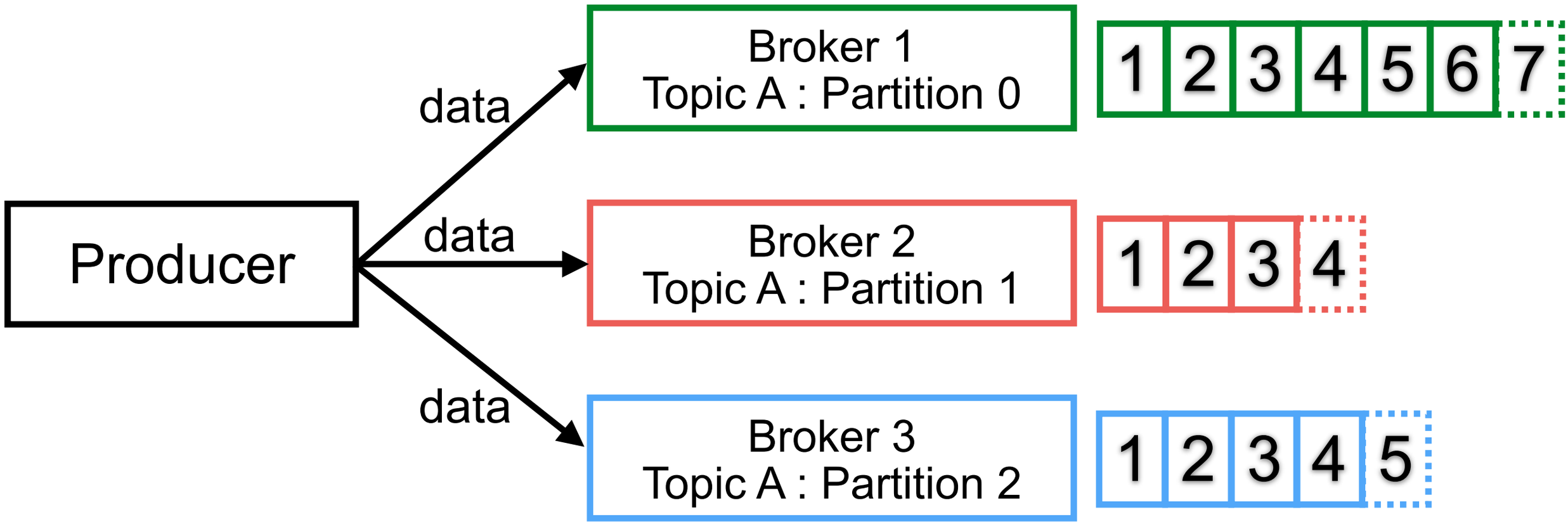
Producers

Producers write data to topics

Automatically know to broker and partition to write

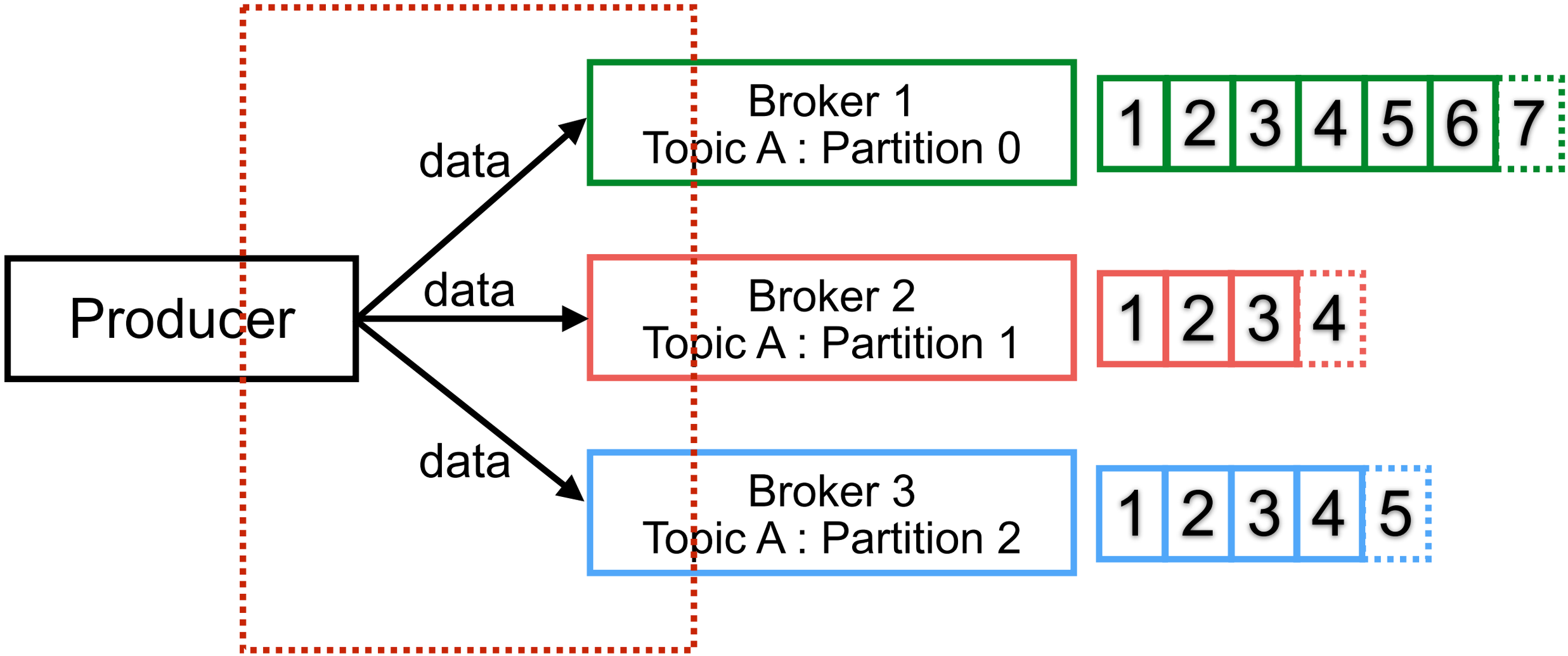
When broker failures, producers will automatically recover

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

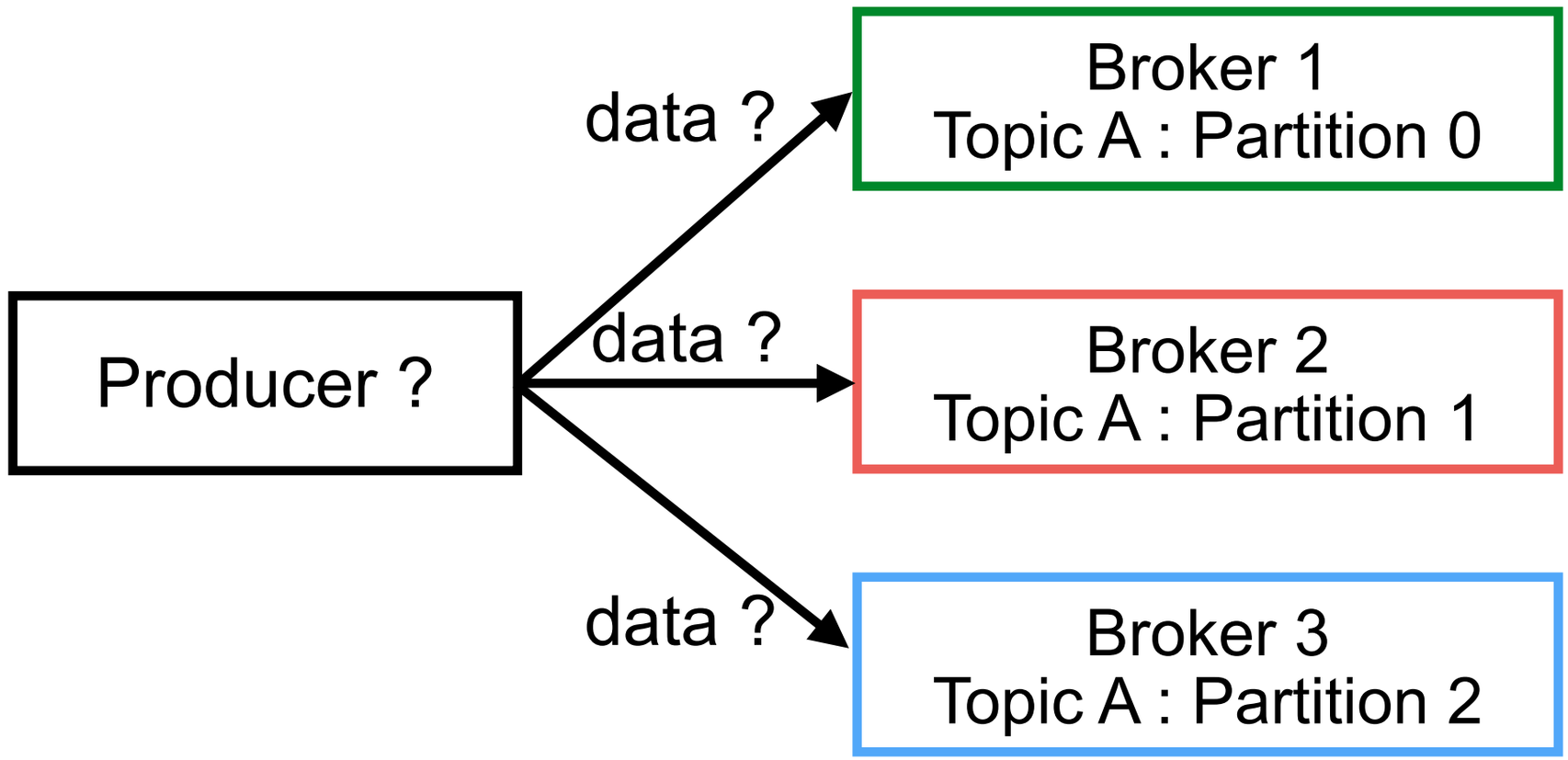
Producer will wait for **leader + ISR** acknowledgment

No data loss

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

Producers with message keys

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



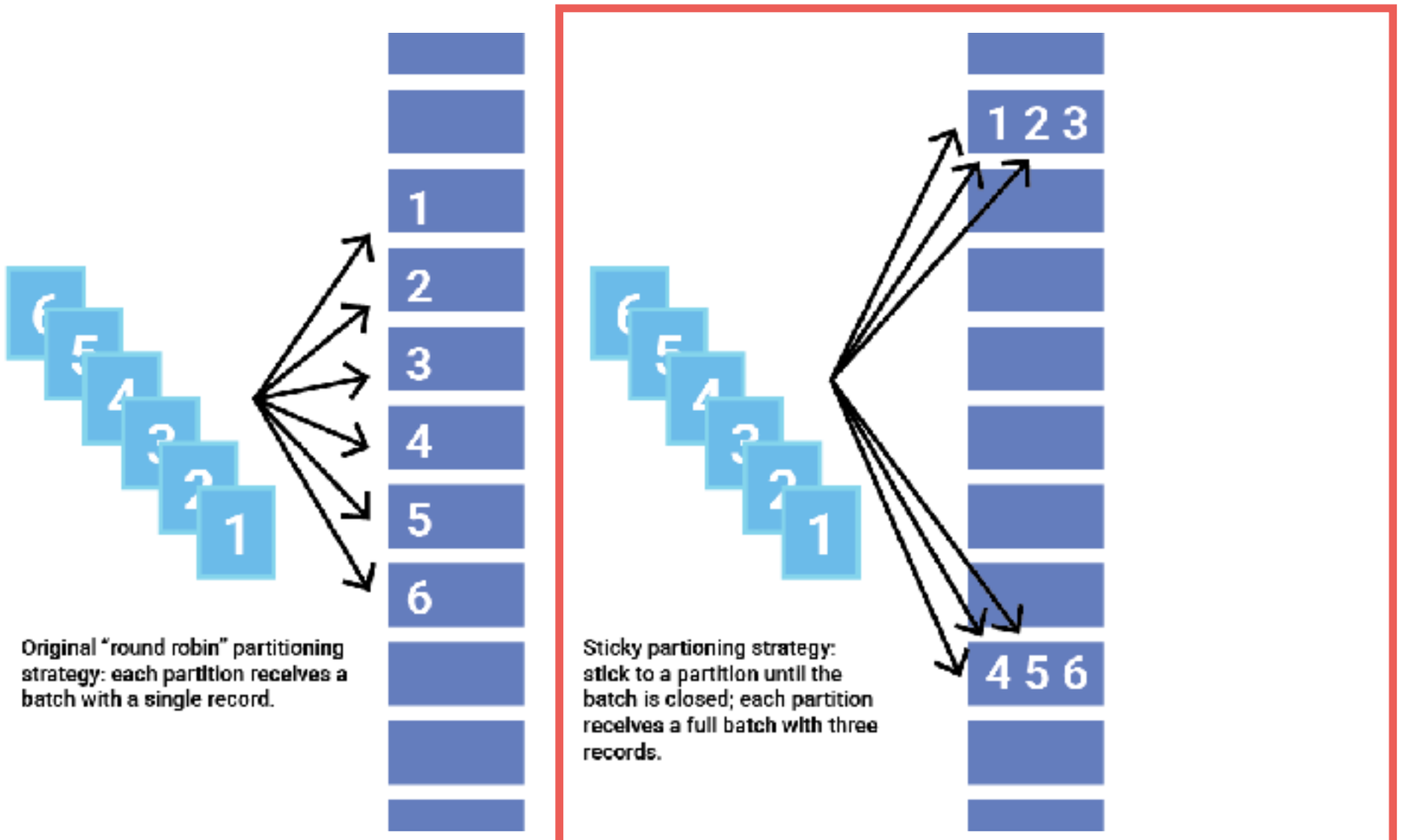
How to assign to partition ?

```
/**
 * Compute the partition for the given record.
 *
 * @param topic The topic name
 * @param numPartitions The number of partitions of the given {@code topic}
 * @param key The key to partition on (or null if no key)
 * @param keyBytes serialized key to partition on (or null if no key)
 * @param value The value to partition on or null
 * @param valueBytes serialized value to partition on or null
 * @param cluster The current cluster metadata
 */
public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster,
                     int numPartitions) {
    if (keyBytes == null) {
        return stickyPartitionCache.partition(topic, cluster);
    }
    return BuiltInPartitioner.partitionForKey(keyBytes, numPartitions);
}

public void close() {}
```

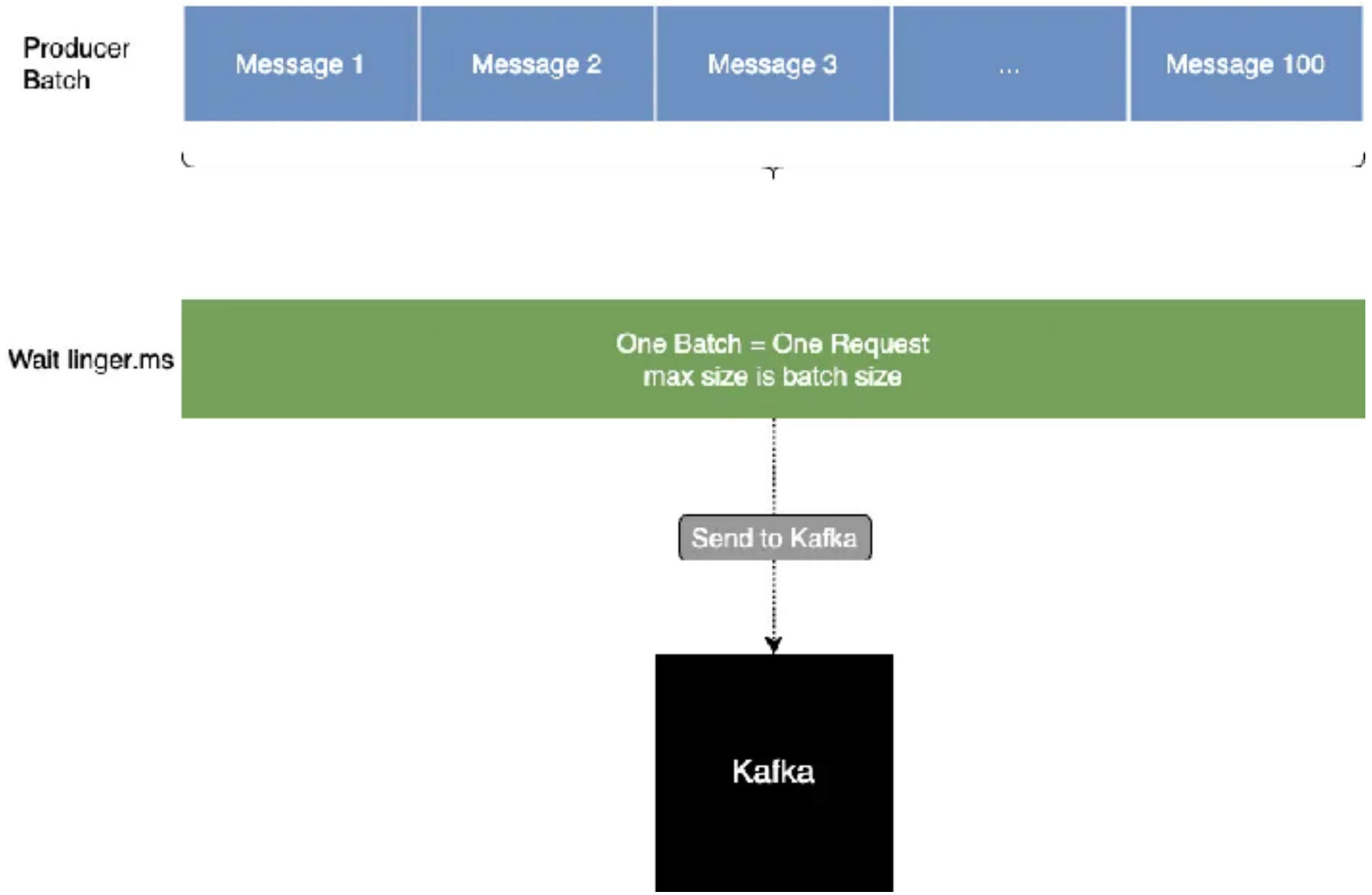
https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/producer/internals/DefaultPartitioner.java

How to assign to partition ?



batch.size and linger.ms

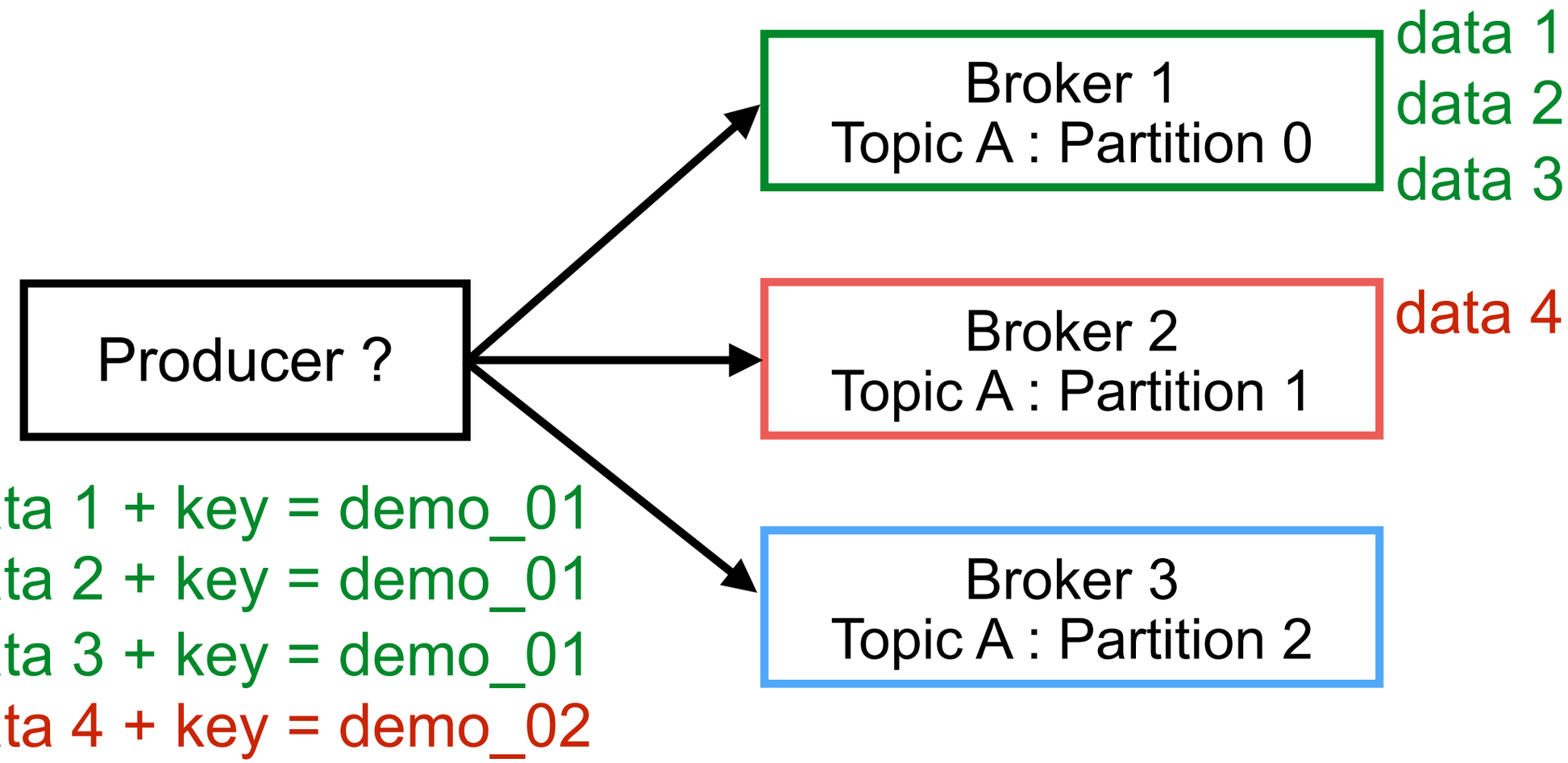
Batch size and linger.ms



https://docs.confluent.io/platform/current/installation/configuration/producer-configs.html

Why use message keys ?

You need message ordering



How to assign to partition ?

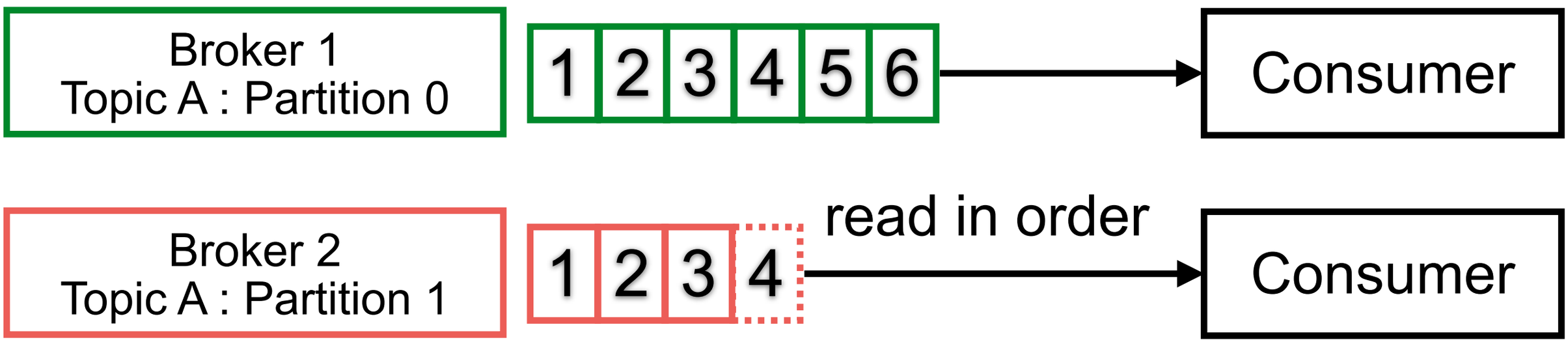
```
326         /*
327          * Default hashing function to choose a partition from the serialized key bytes
328          */
329         public static int partitionForKey(final byte[] serializedKey, final int numPartitions) {
330             return Utils.toPositive(Utils.murmur2(serializedKey)) % numPartitions;
331         }
332
```

Murmur2 algorithm (hashing function)

https://github.com/apache/kafka/blob/trunk/clients/src/main/java/org/apache/kafka/clients/producer/internals/BuiltInPartitioner.java

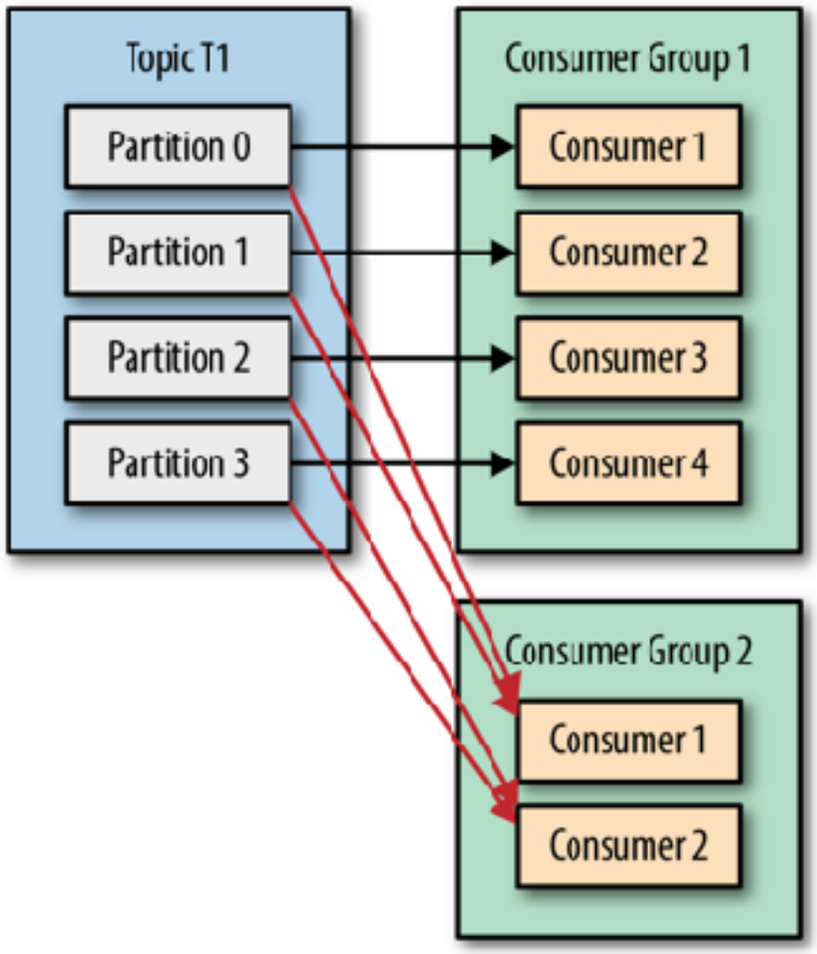
Consumers

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

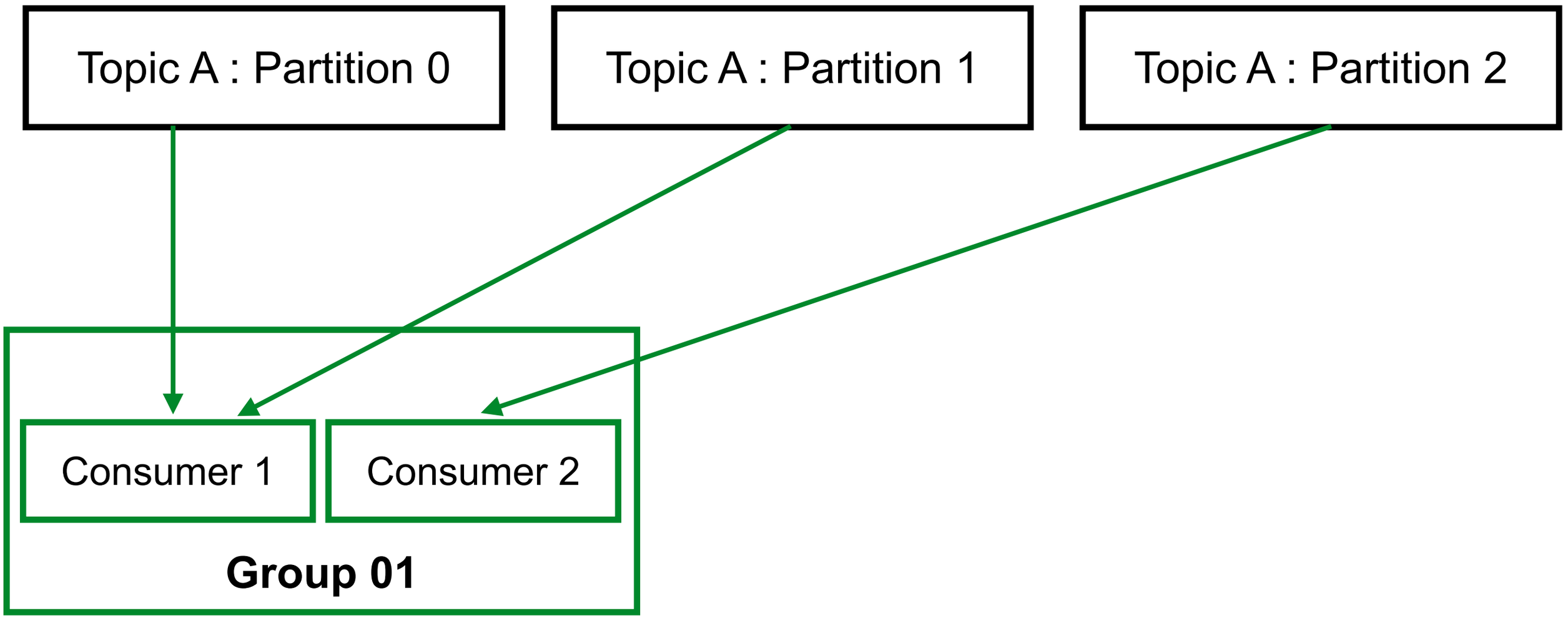


Consumer groups

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

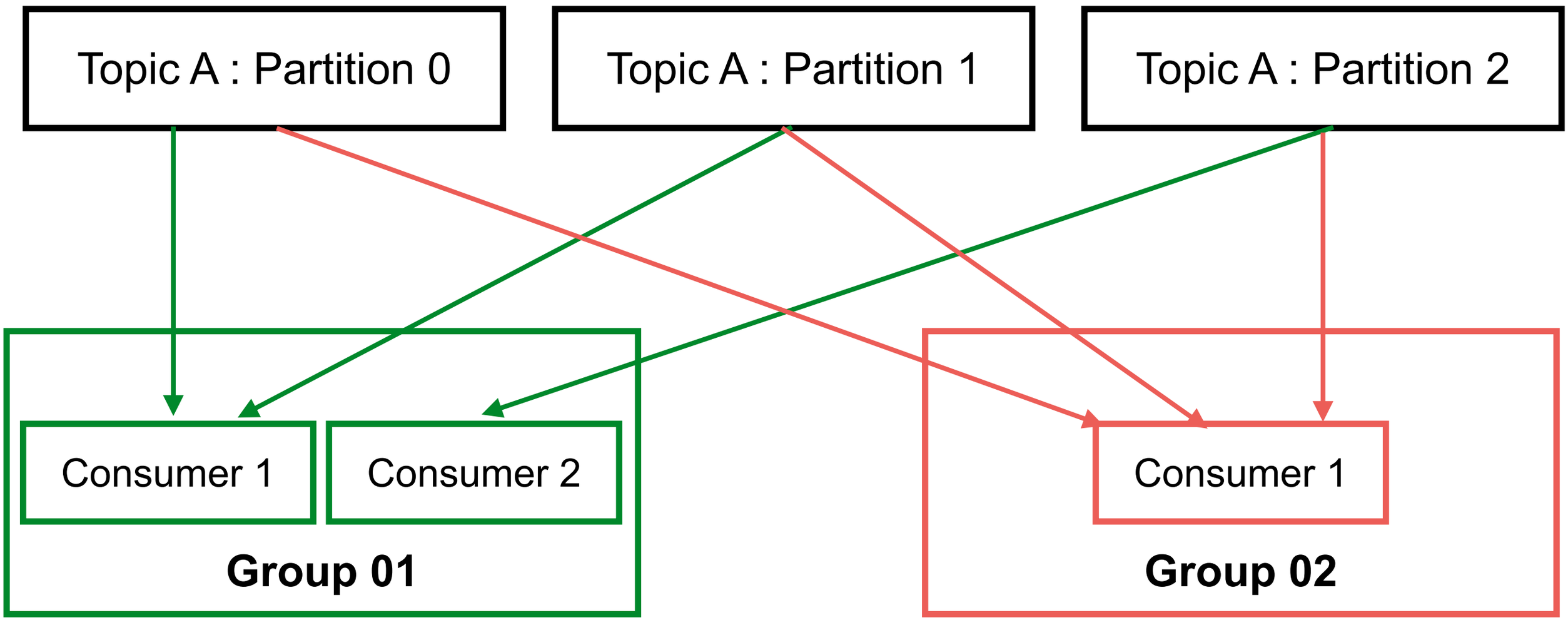


Consumer groups



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

Consumer groups



Partition assignment strategies

Range (default)

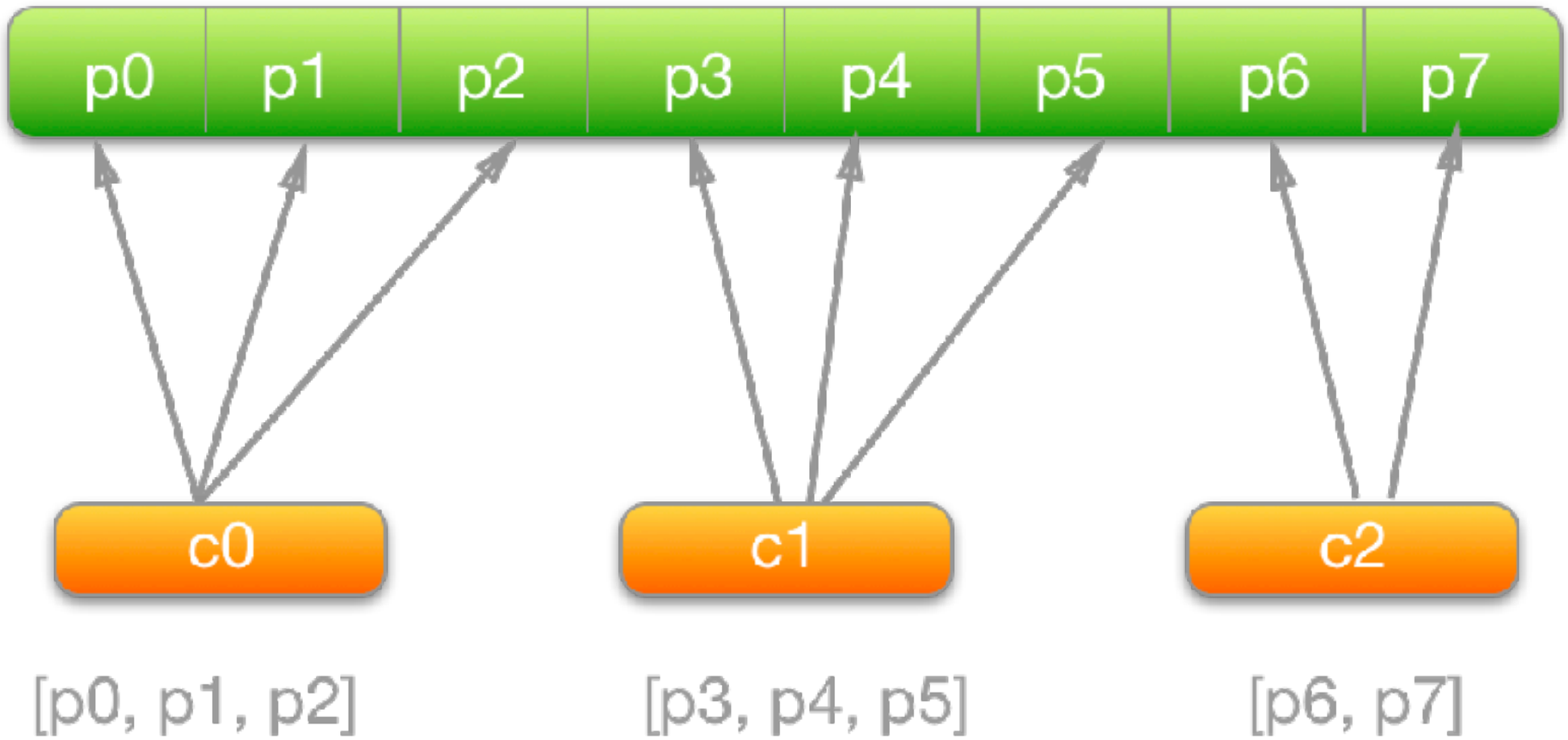
Round Robin

Sticky

Customize (AbstractPartitionAssignor)

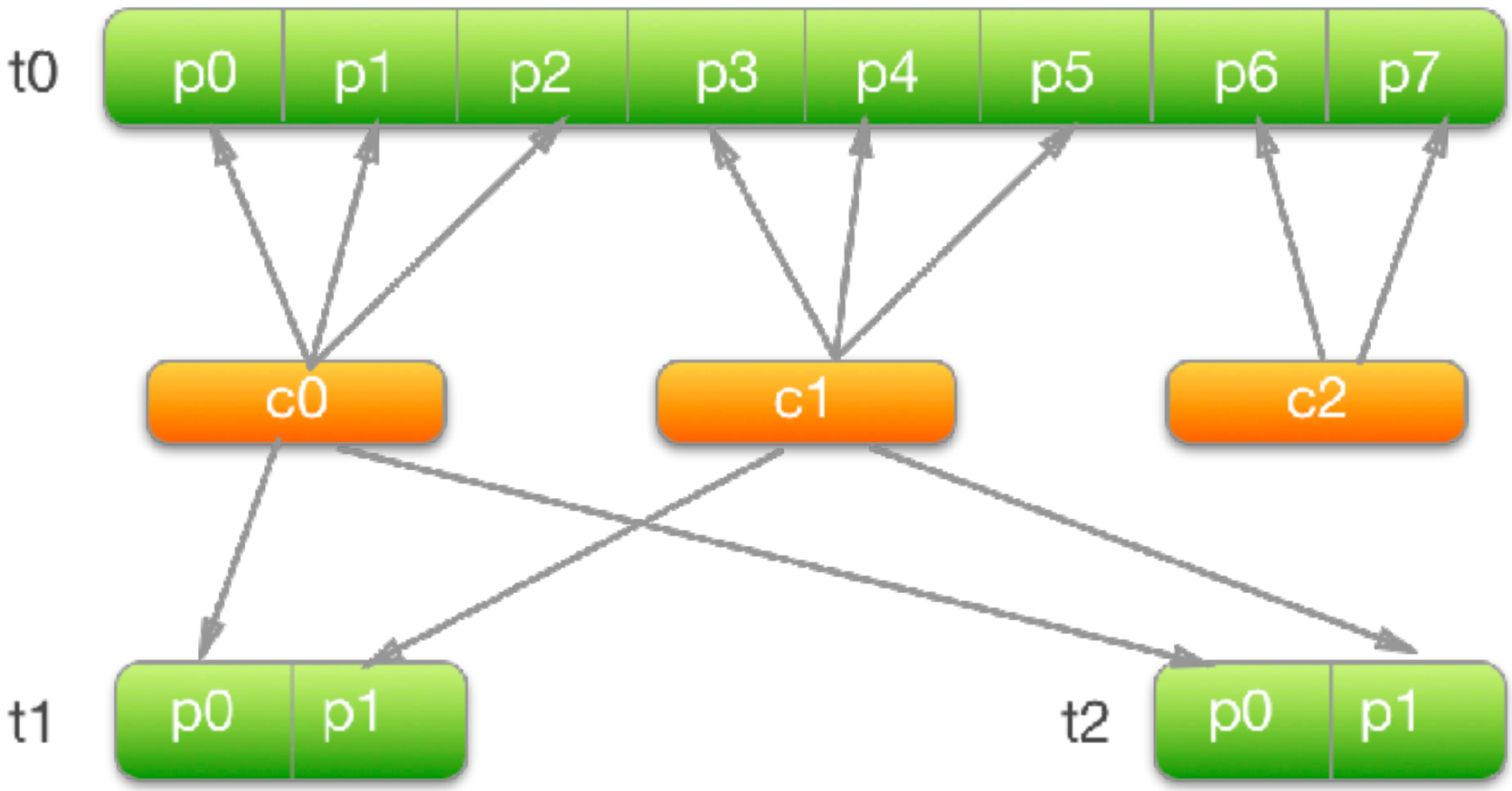
https://docs.confluent.io/platform/current/installation/configuration/consumer-configs.html#partition-assignment-strategy

Range assignor



org.apache.kafka.clients.consumer.RangeAssignor

Range assignor

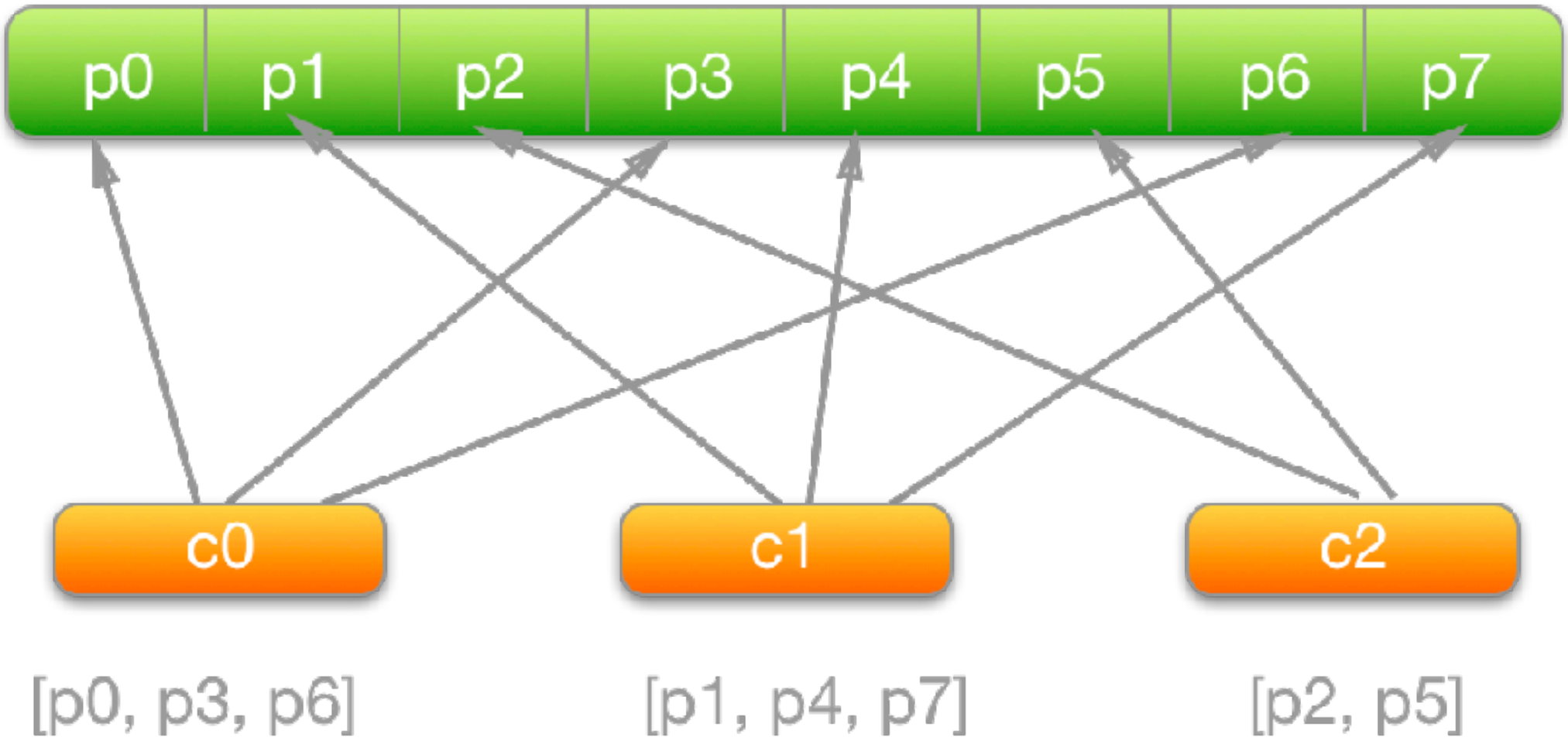


c0 -> [t0p0, t0p1, t0p2, t1p0, t2p0]

c1 -> [t0p3, t0p4, t0p5, t1p1, t2p1]

c2 -> [t0p6, t0p7]

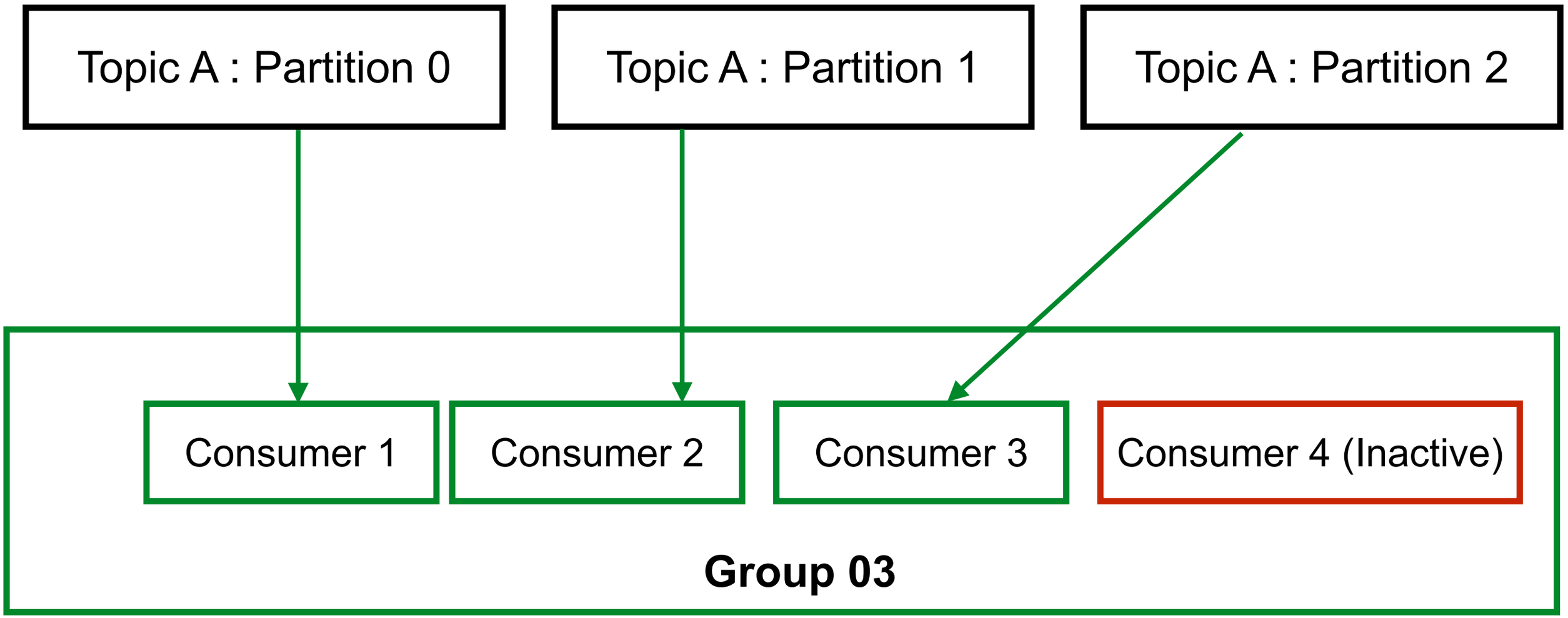
Round Robin assignor



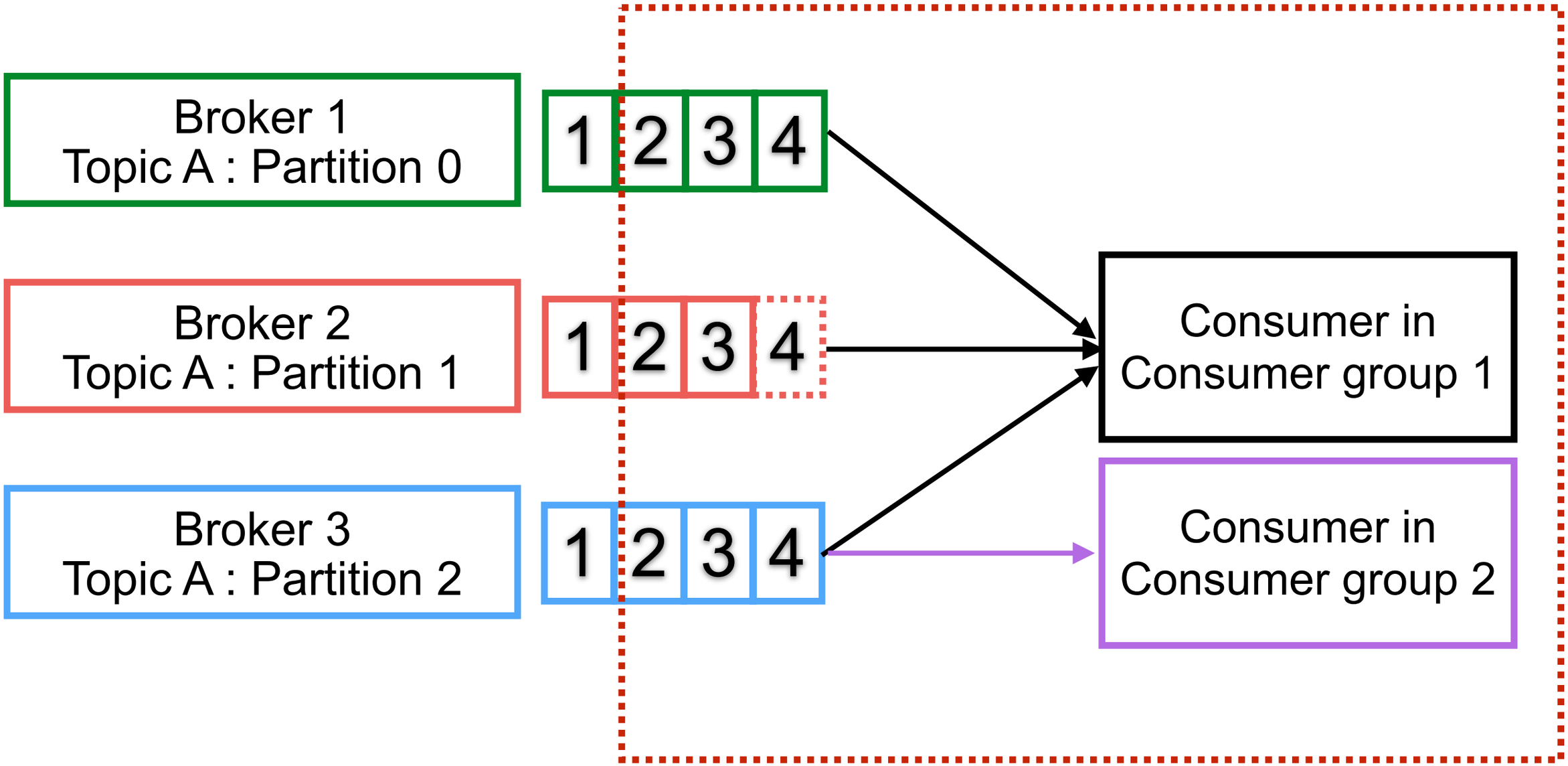
org.apache.kafka.clients.consumer.RoundRobinAssignor

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

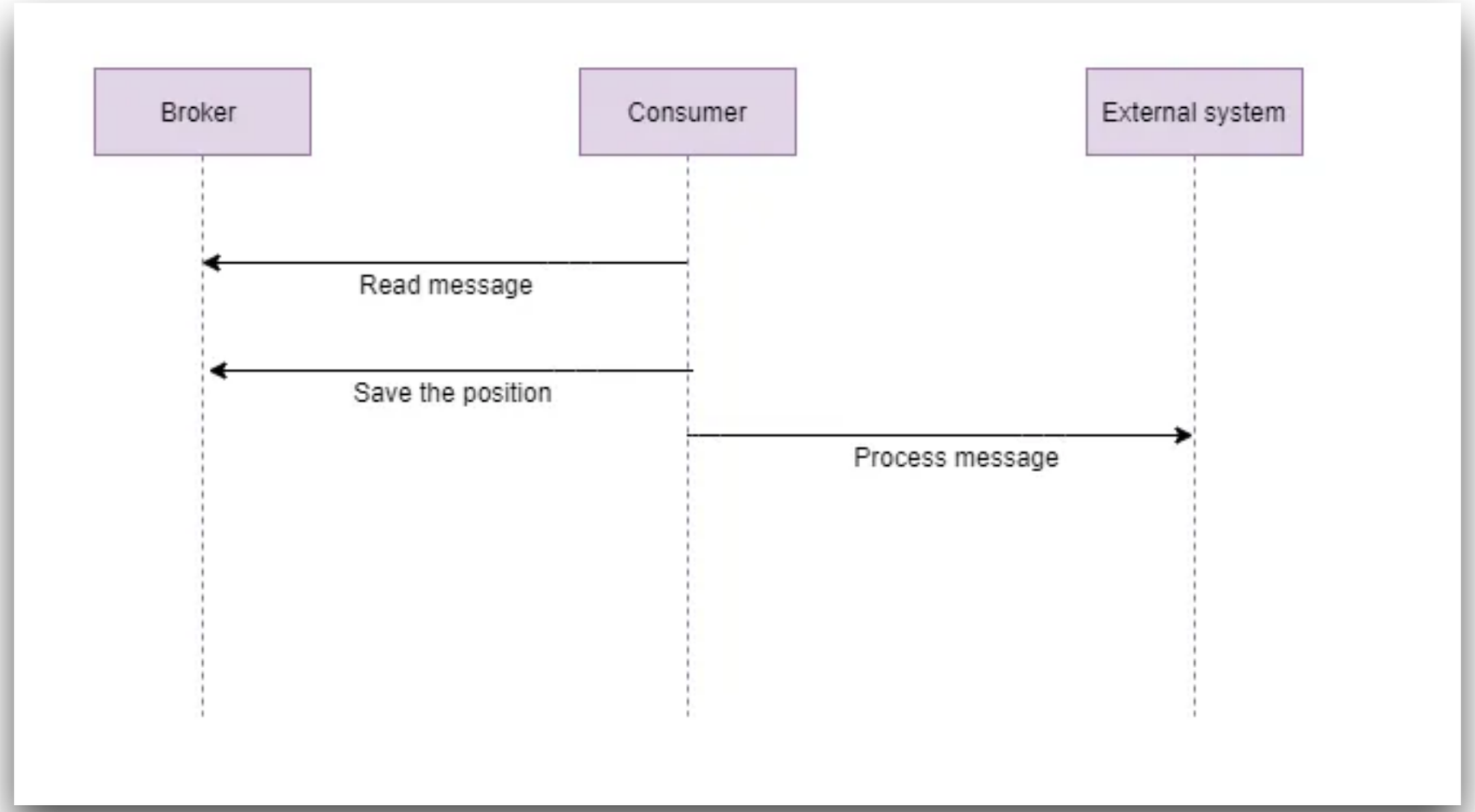
Offsets are committed as soon as the message is received

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

Fire and forgot pattern

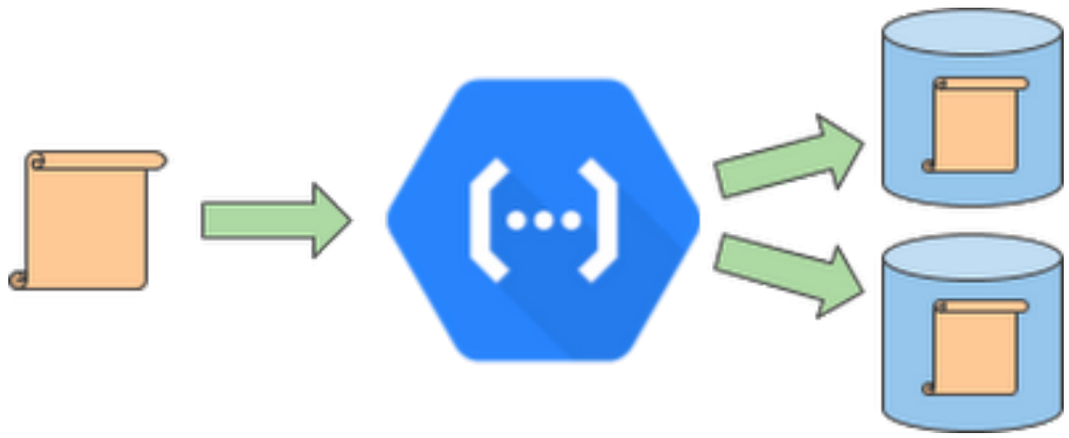
Lowest latency

At most once



2. At lease once (1)

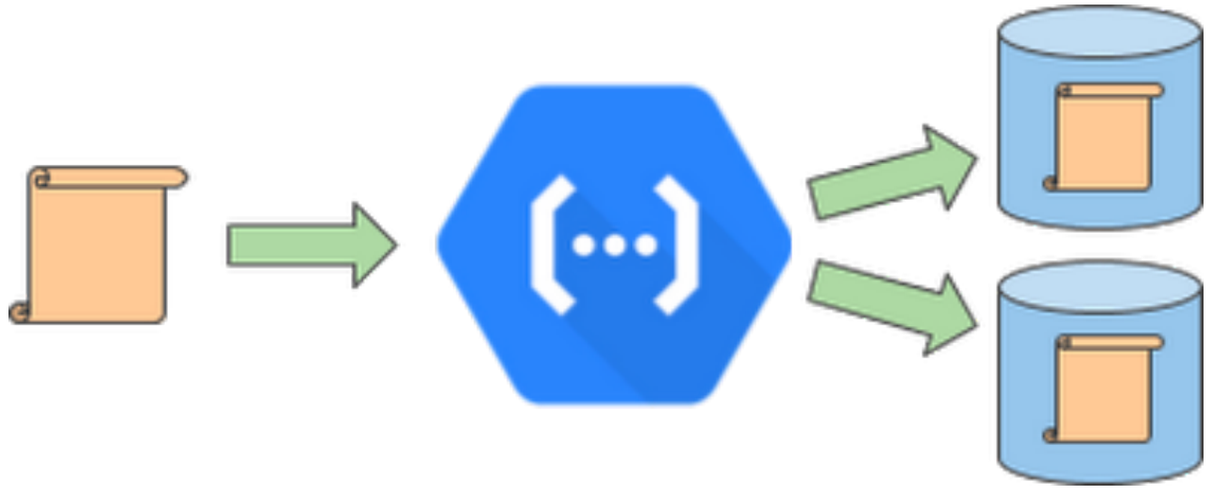
Offsets are committed after the message is processed
Messages are never lost but may be **redelivered**
If processing go wrong, the message will be **read again**



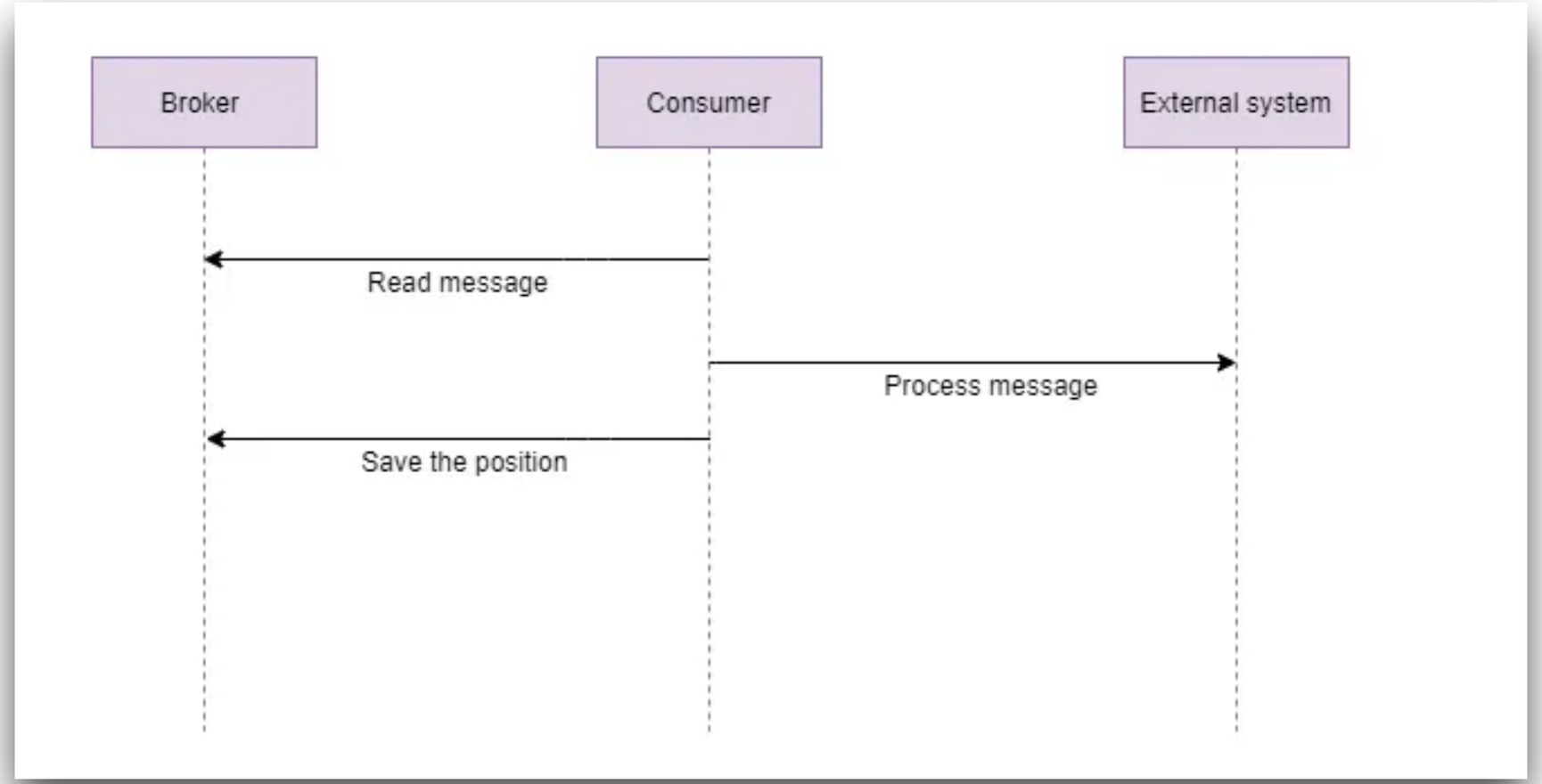
2. At lease once (2)

Make sure your processing is **idempotent**

Processing again the message not impact to your system !!



At lease once

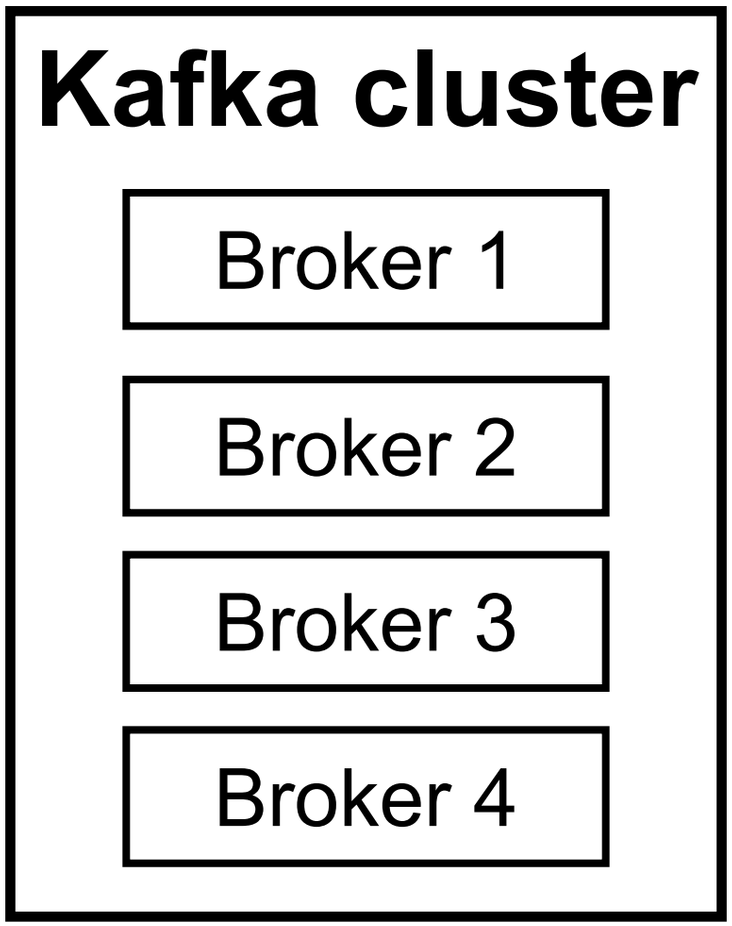


3. Exactly once

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

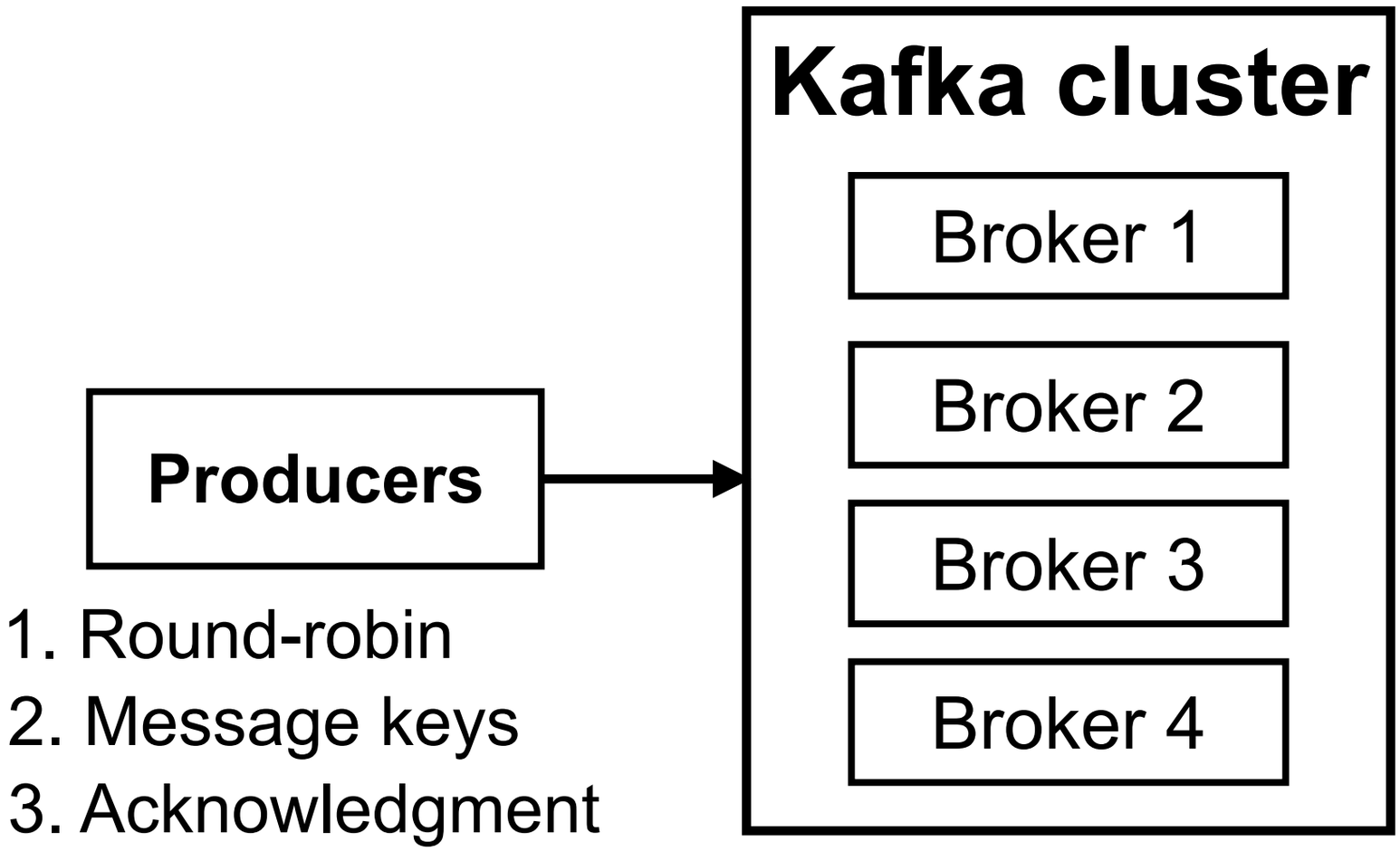
Summary of Kafka

Kafka concepts



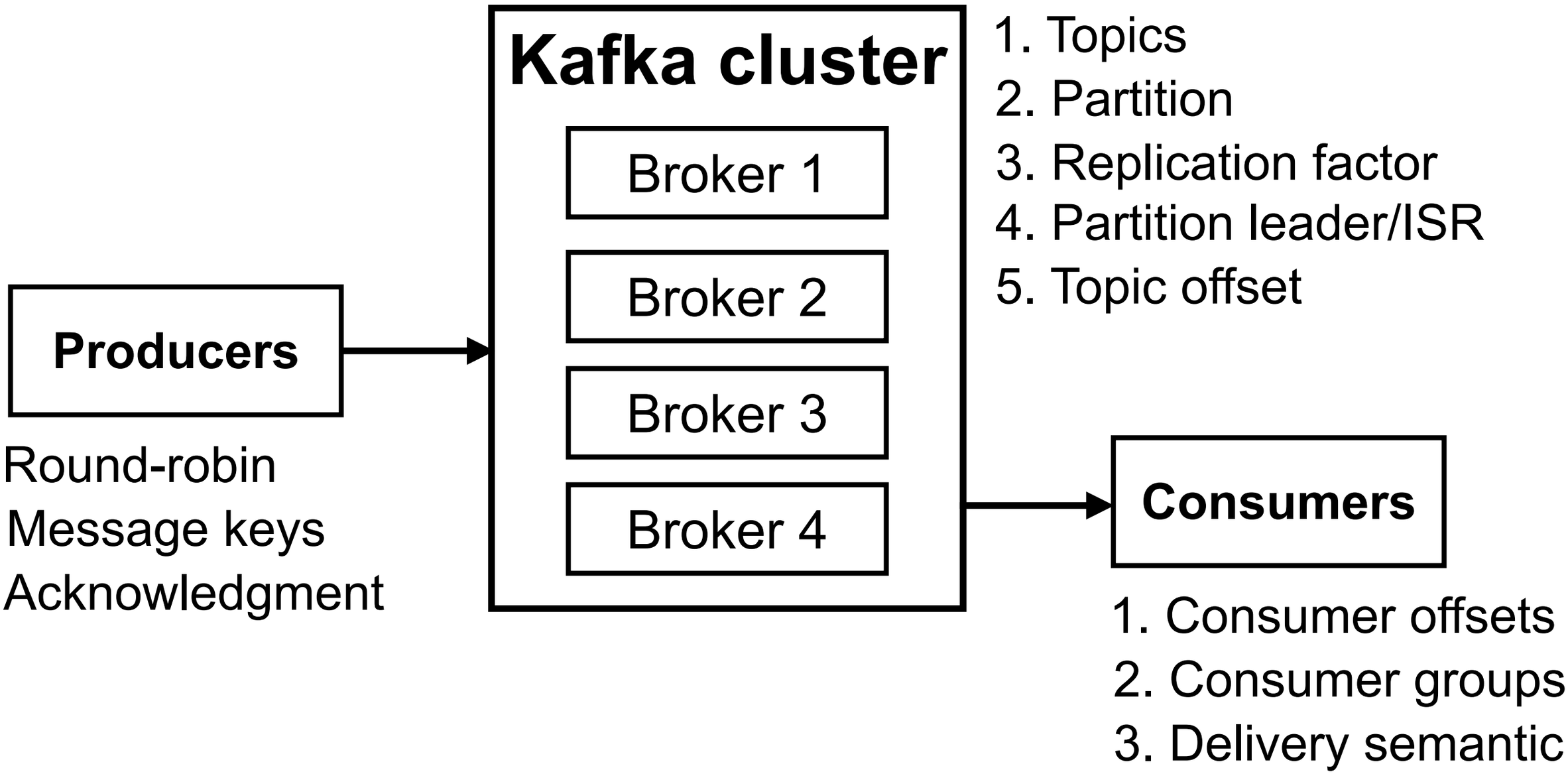
1. Topics
2. Partition
3. Replication factor
4. Partition leader/ISR
5. Topic offset

Kafka concepts

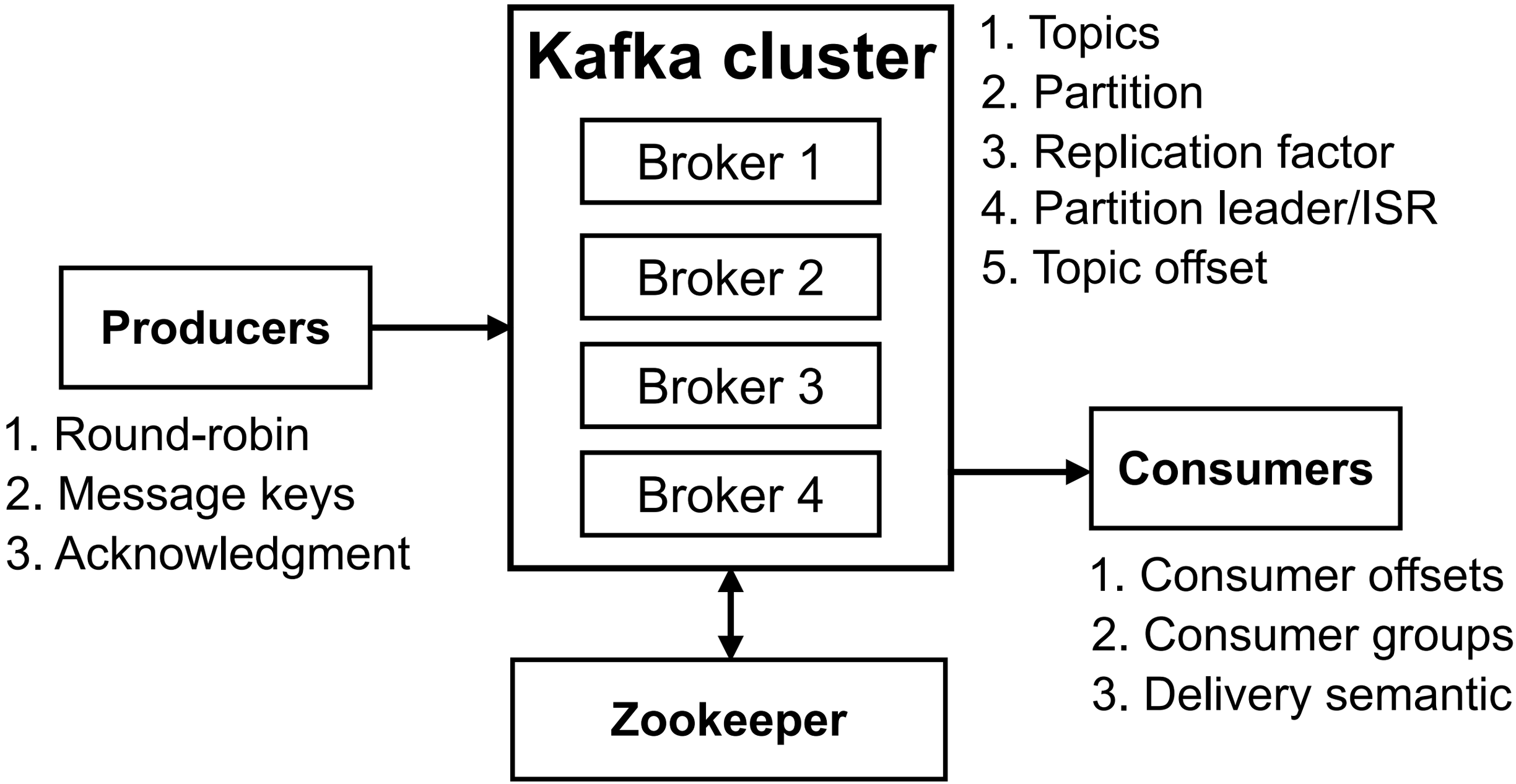


1. Topics
2. Partition
3. Replication factor
4. Partition leader/ISR
5. Topic offset

Kafka concepts



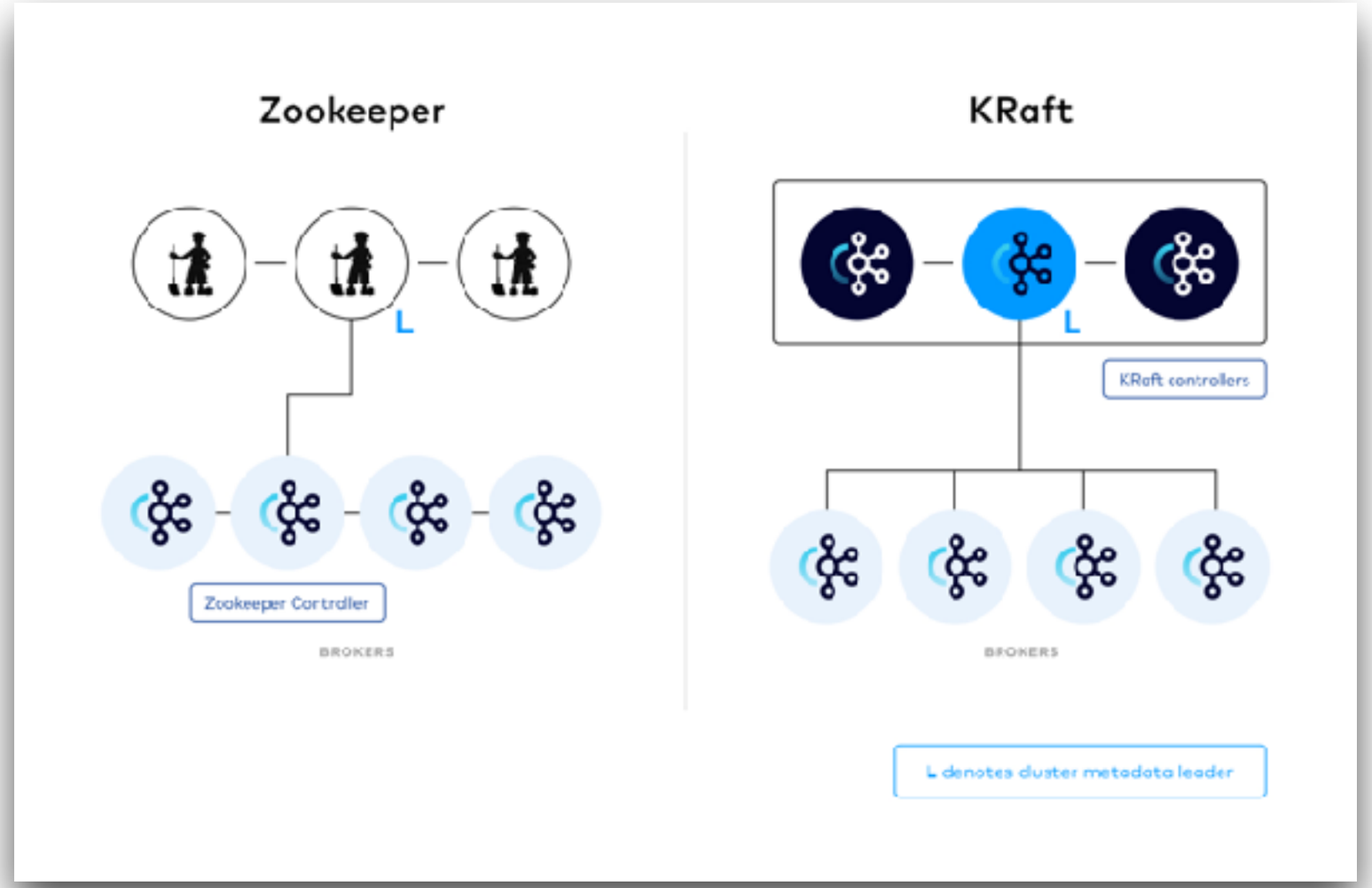
Kafka concepts



Kraft

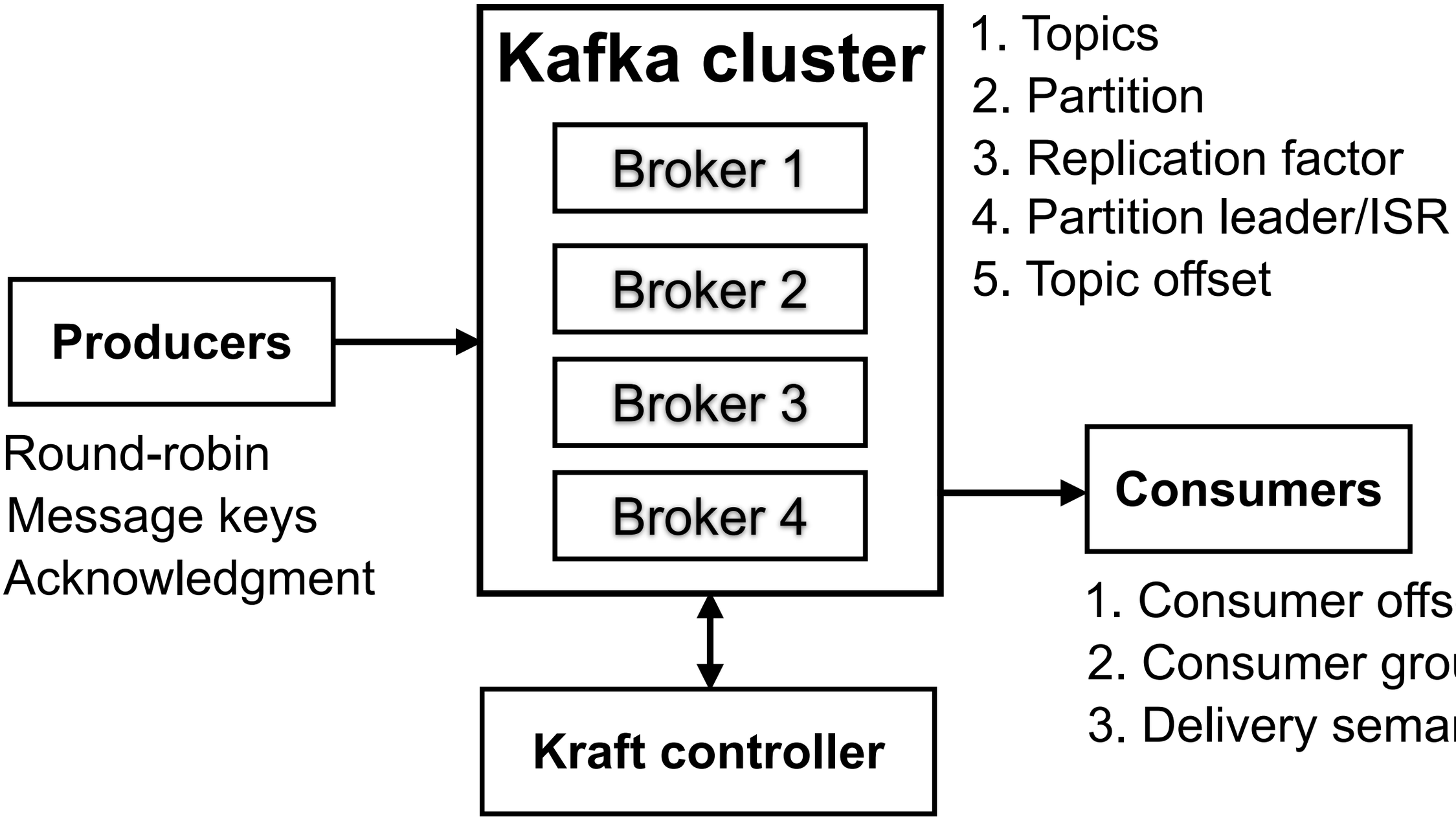
Kafka without Zookeeper (Kraft)

Kafka Raft



https://developer.confluent.io/learn/kraft/

Kraft

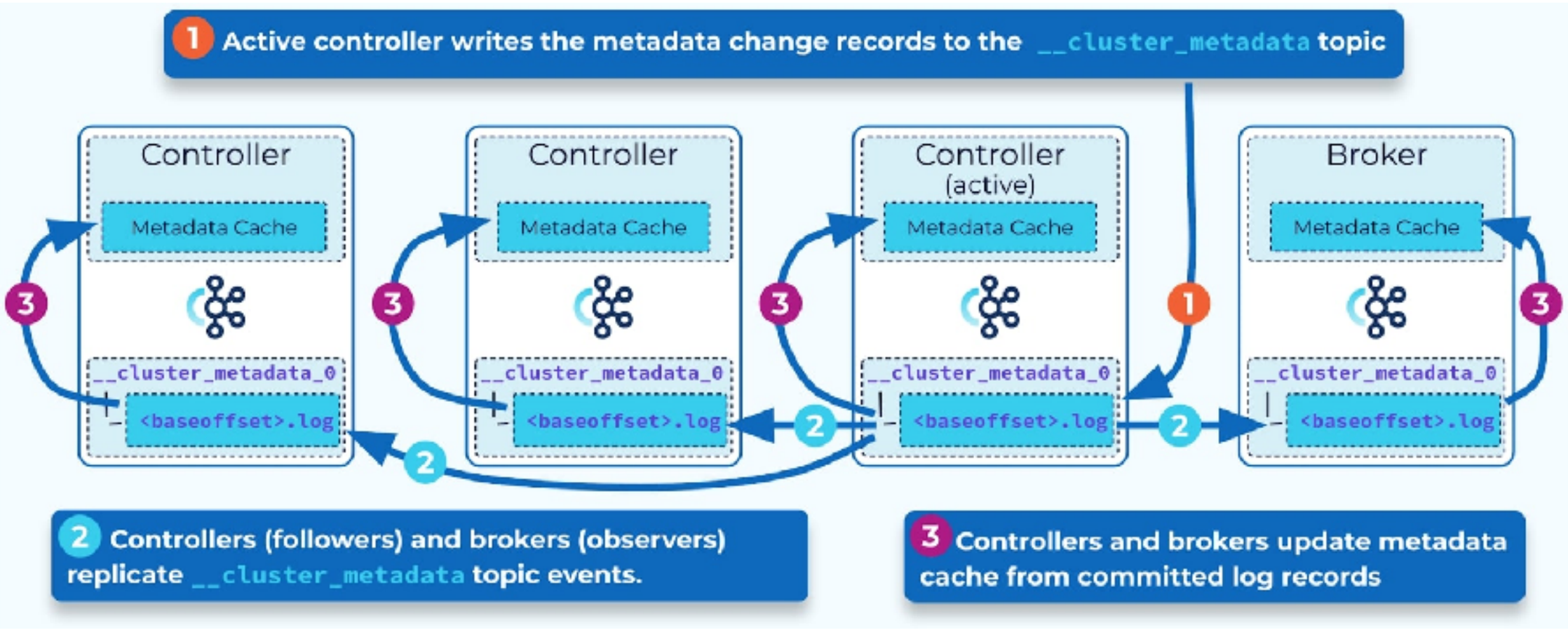


1. Round-robin
2. Message keys
3. Acknowledgment

1. Topics
2. Partition
3. Replication factor
4. Partition leader/ISR
5. Topic offset

1. Consumer offsets
2. Consumer groups
3. Delivery semantic

Kraft



https://www.confluent.io/fr-fr/blog/what-is-kraft-and-how-do-you-use-it/

Working with Schema Registry

Schema Registry

Server process external to Kafka brokers

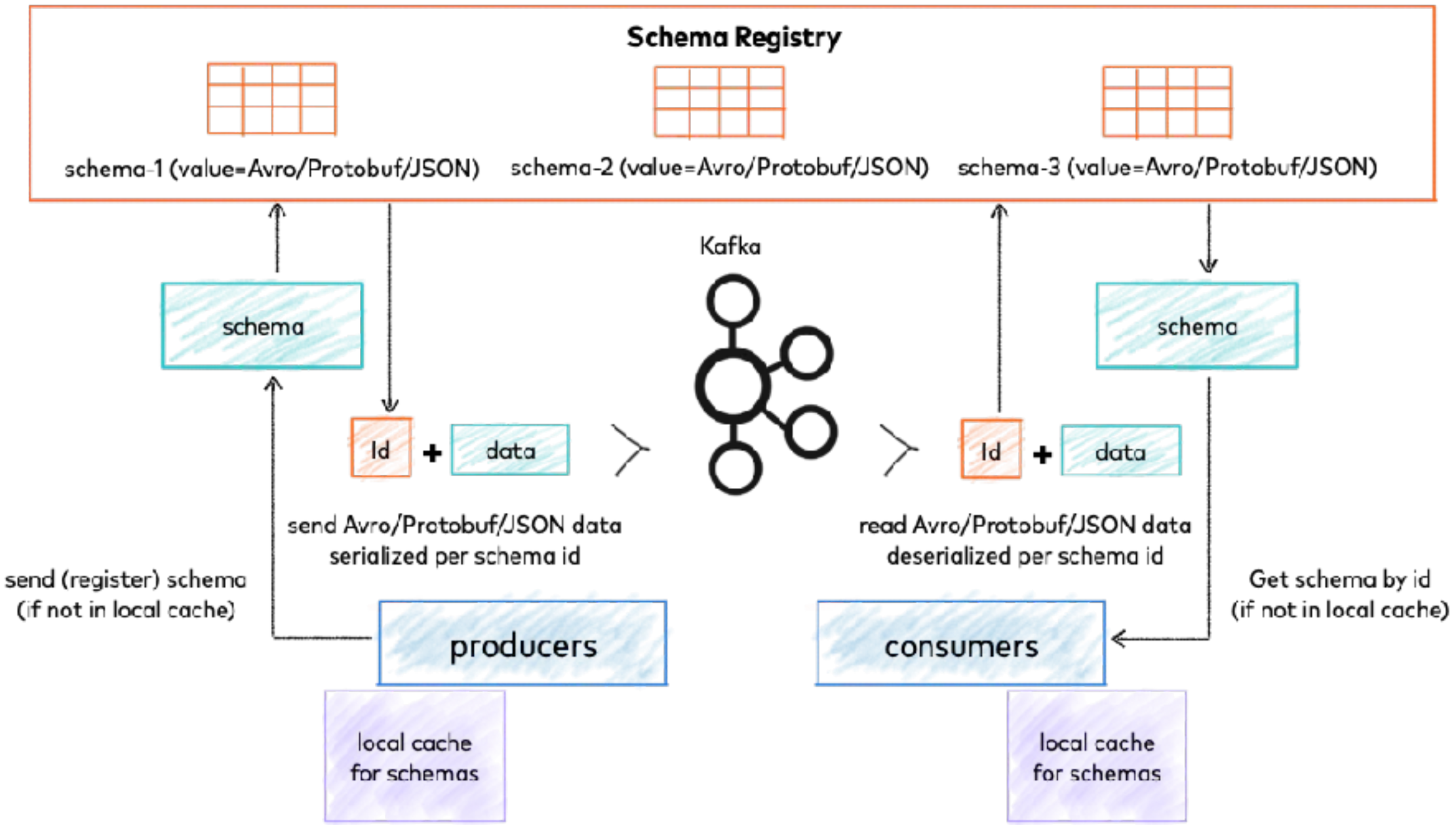
Maintain a database of schemas

Provider APIs for producer and consumer

Compatible of message in producer and consumer

Schema change !!
Validation …

Schema Registry



https://docs.confluent.io/platform/current/schema-registry/

Steps in Schema Registry

Define schema compatibility rules per topic
Producer API prevents incompatible messages
Consumer API prevents incompatible messages

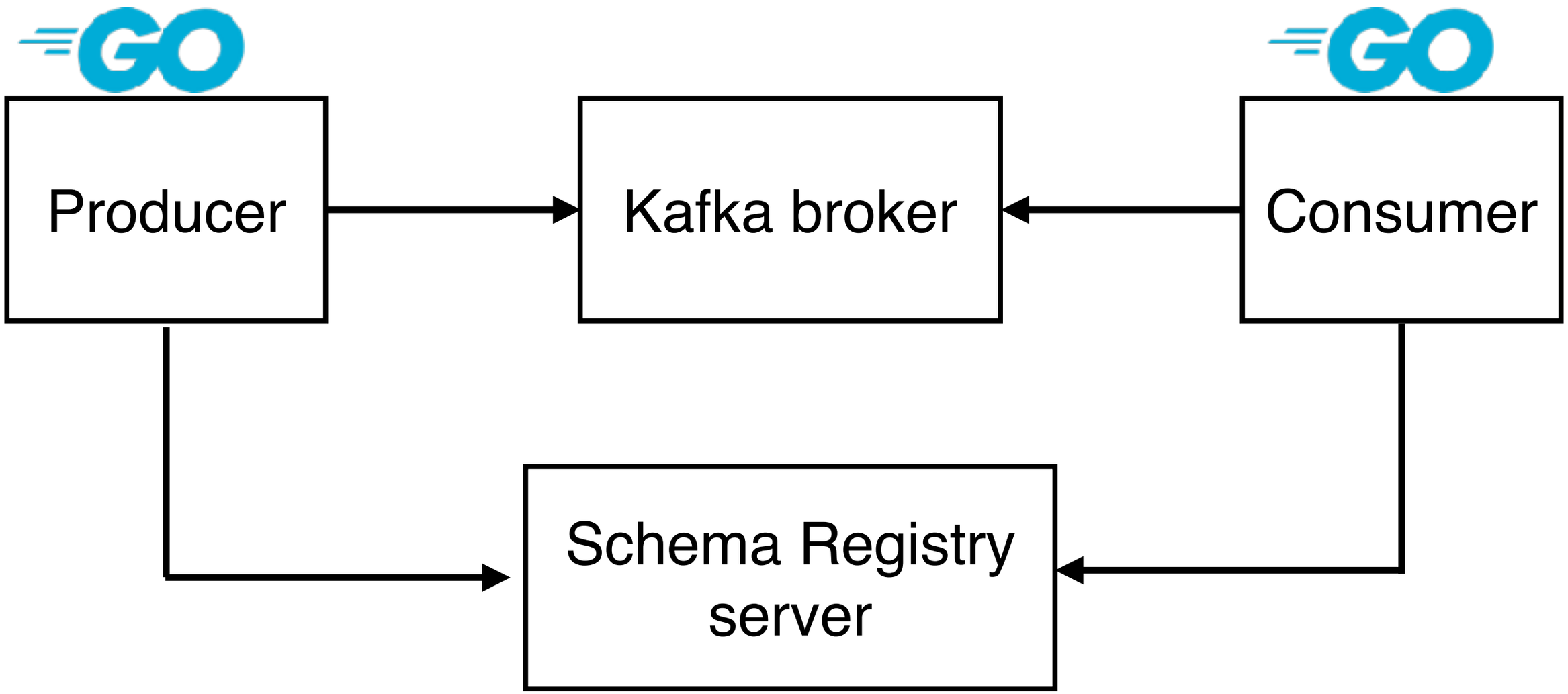
Supported formats

JSON Schema

Avro

Protocol Buffers

Workshop

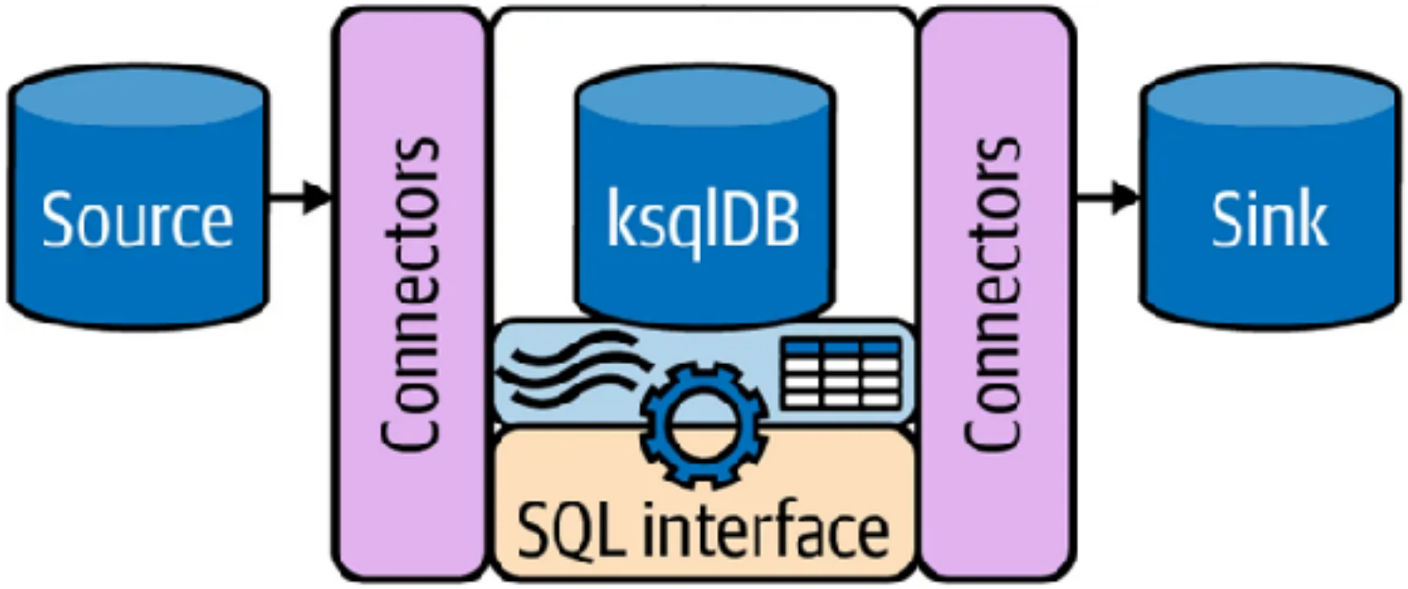


https://github.com/up1/course-kafka-2024/tree/main/workshop/basic

ksqlDB

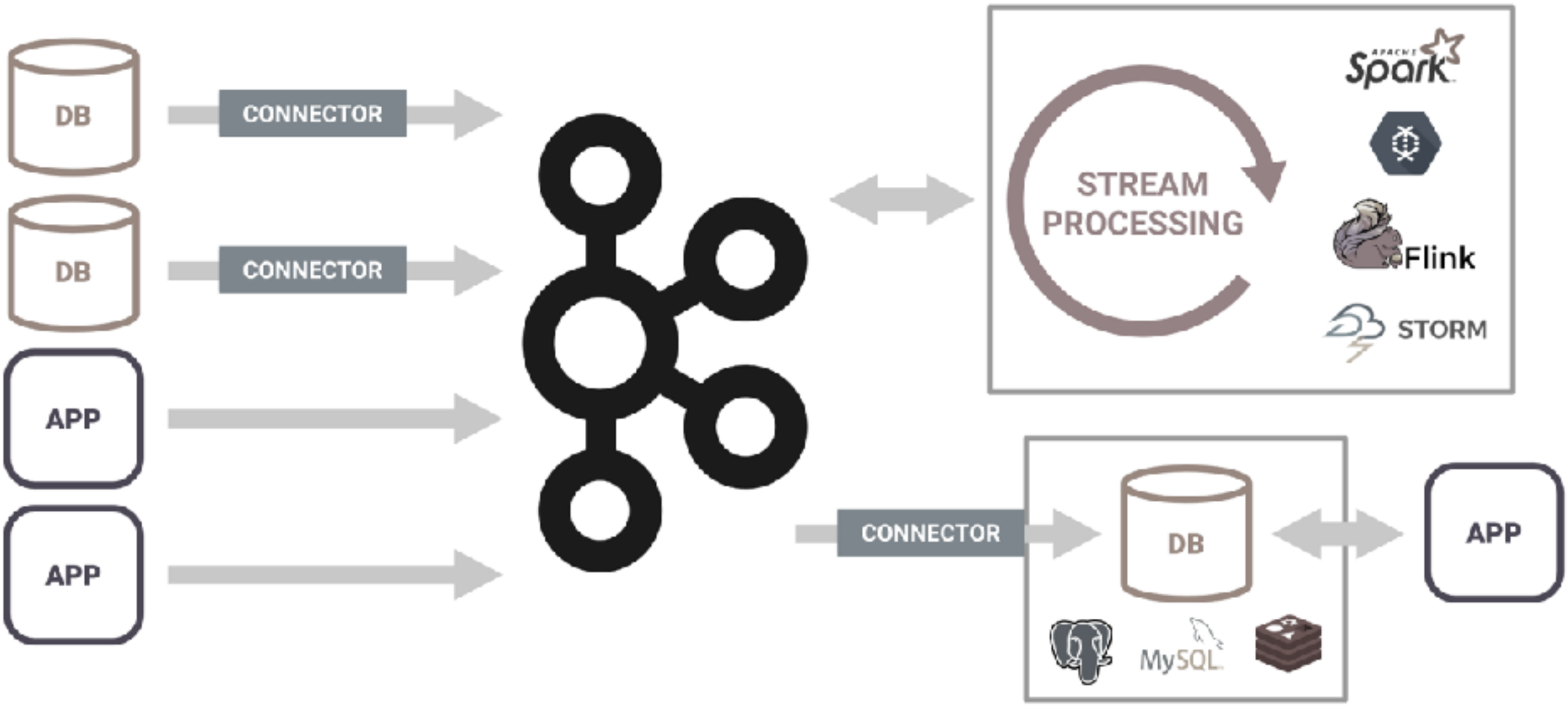
ksqlDB

Kafka native database for **stream processing**
Used like SQL
Working with Kafka connect



https://www.confluent.io/product/ksqldb/

Kafka

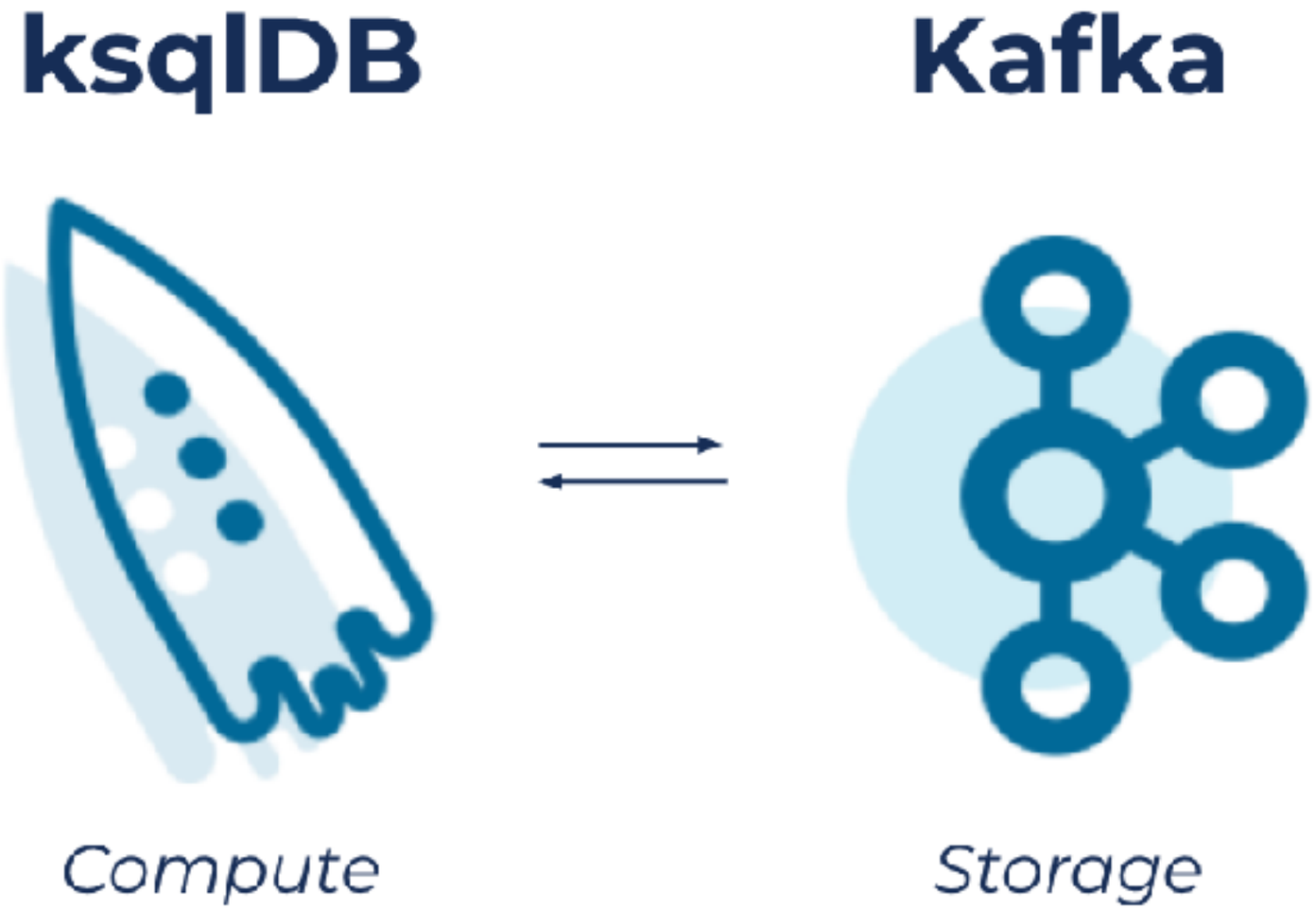


ksqlDB

https://developer.confluent.io/courses/ksqldb/intro/

ksqlDB

Don’t use for primary database



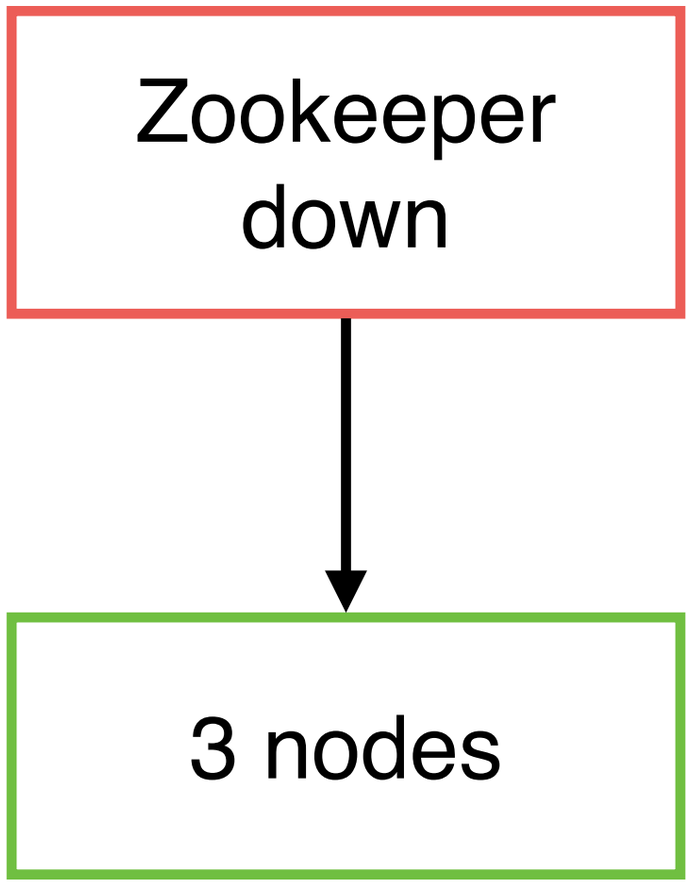
Plan for Failures of Kafka

Plan for Failures of Kafka

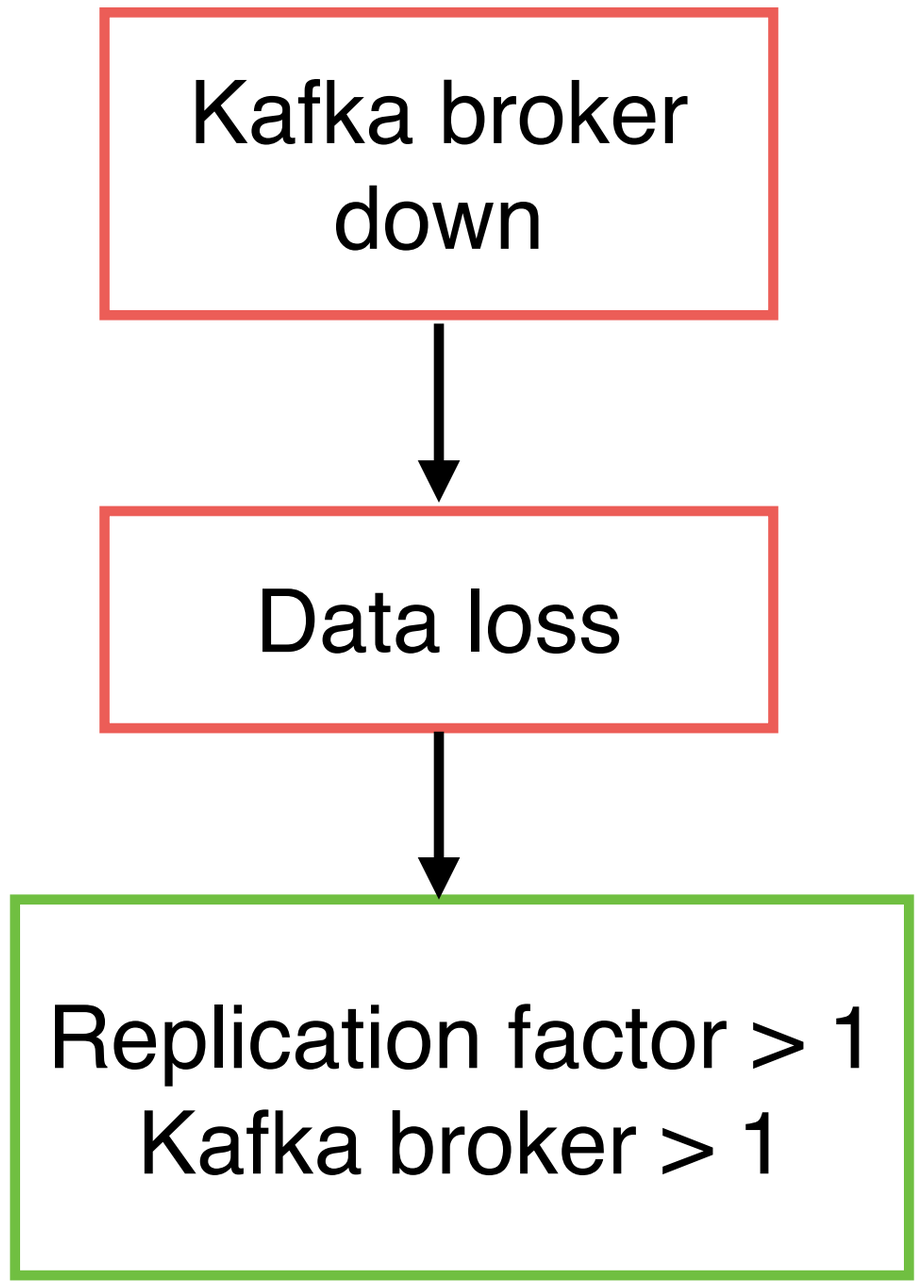
Zookeeper

Kafka broker

Zookeeper



Kafka broker



Rule of Thumb

Replication factor (RF) should typically be less than or equal to the **number of brokers**

Recommended in production = 3

High Availability

Fault tolerance
With low overhead

Ratio guideline

Number of Brokers	Recommended Replication Factor (RF)	Broker-to-RF Ratio
3	3	1:1
4	3	4:3
5	3	5:3
6	3	2:1
10	3-5	2:1 to 10:5
15	3-5	3:1 to 15:5

Key Considerations (1)

More RF, more storage/network overhead

RF = 3
Overhead = 3 * # of partitions

Disk

Memory

Network

Key Considerations (2)

Broker failure

RF = N + 1
N = number of broker

Key Considerations (3)

Partition distribution

More partition, more RF
==
more overhead

Key Considerations (4)

Write latency of producer

Ack = 0
Ack = 1
Ack = all

Improve write latency of producer

Increase # of partition
Reduce batch size and linker
Increase buffer size (reduce network call)
Use compression

Key Considerations (5)

Read latency of consumer

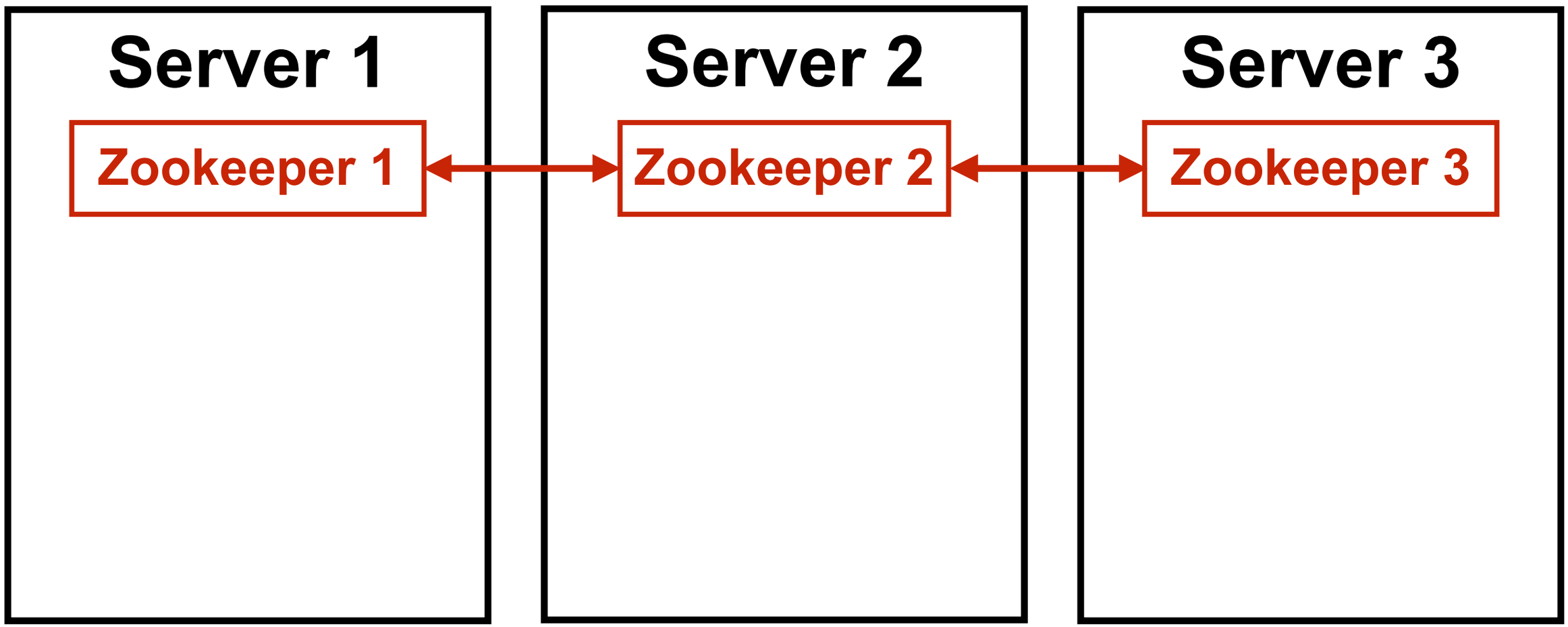
Fetch min

Fetch max

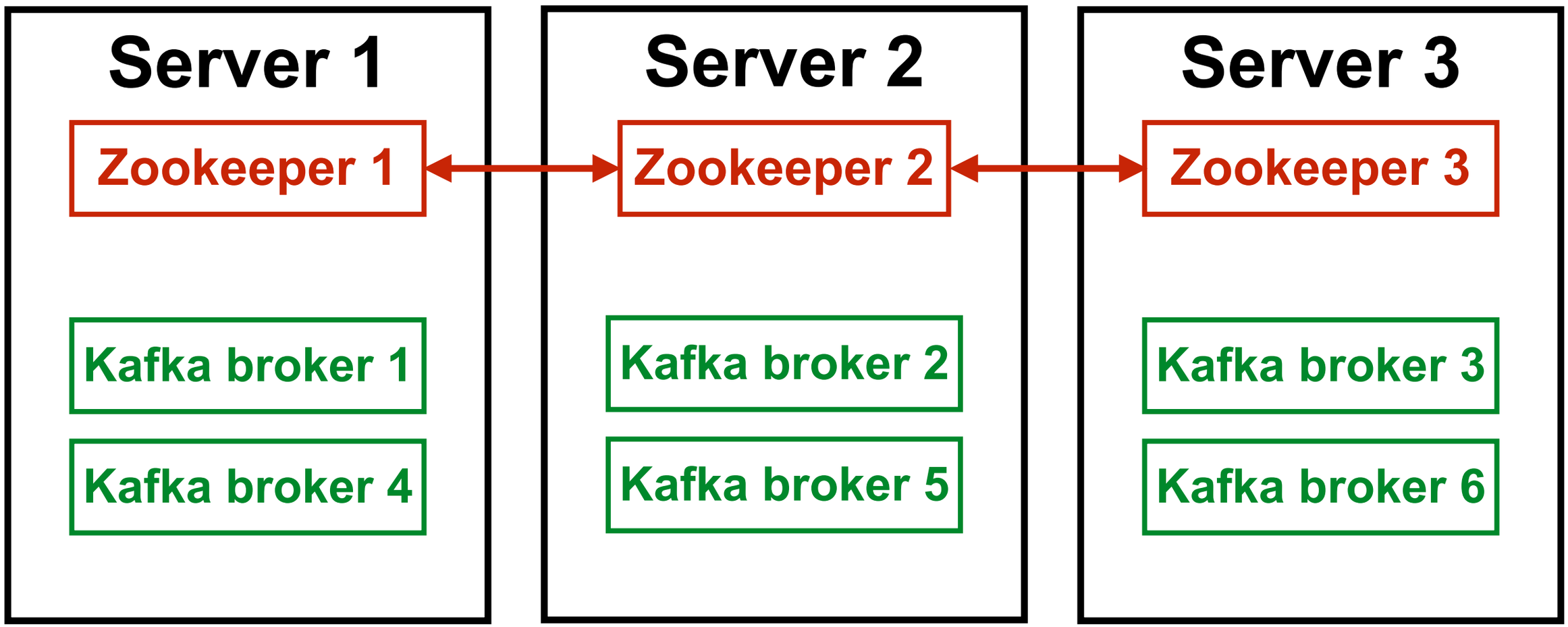
Max poll record

Monitoring Kafka

Kafka cluster



Kafka cluster



Kafka cluster

It’s not easy to setup a cluster

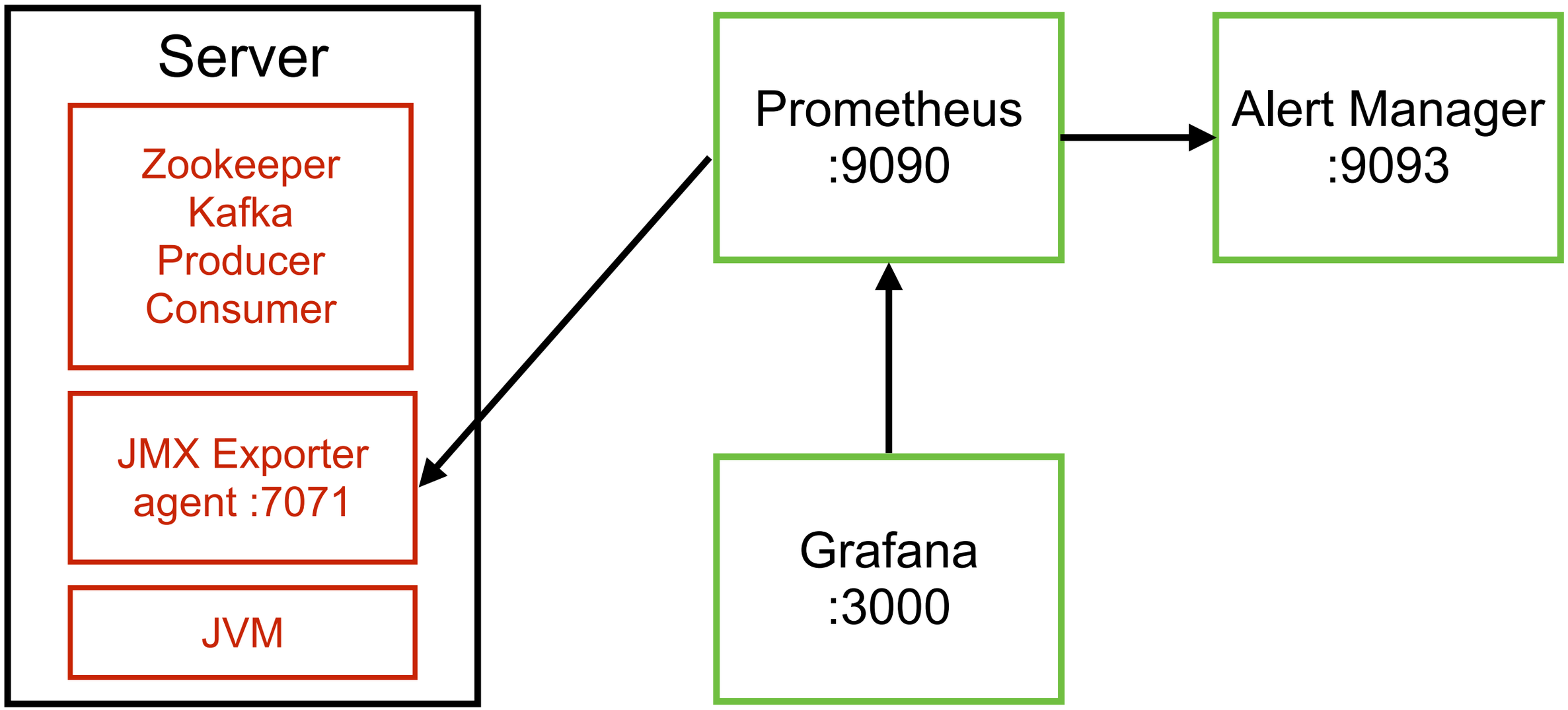
Need to isolate each Zookeeper and broker on separate servers

Need to implement a **Monitoring system**

Need skill of **operation** teams

Need a good **Kafka Admin**

Kafka monitoring



https://github.com/up1/course-kafka-2024/tree/main/workshop/monitoring

Tips #1

Please monitor Your Kafka !!

Monitoring ?

Broker health

Message delivery

Performance

Capacity

Kafka monitoring

Kafka exposes **metrics** through **JMX**
These metrics are very important for monitoring

Kafka monitoring

JConsole

JMX/Metrics integration

Burrow

Keep Kafka metrics ?

ELK stack
Prometheus
Confluent control center
Datadog
NewRelic
etc…

Important metrics

Under replicated partitions

of partitions are have problem with the **ISR**

May indicate a high load on the system

Request handlers

Utilization of threads for I/O, network

Utilization of Kafka broker

Important metrics

Request timing

How long it takes to reply to requests

Low is better, latency will improved

https://kafka.apache.org/documentation/#monitoring

https://docs.confluent.io/current/kafka/monitoring.html

https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/

Tips #2

Have a dashboard that lets you know
“Everything is OK ?”
(in one look)

Dashboard

Dashboard

Tips #3

Don’t watch the dashboard

Alert on what is important

Only restart if you know why this will fix the issue !!

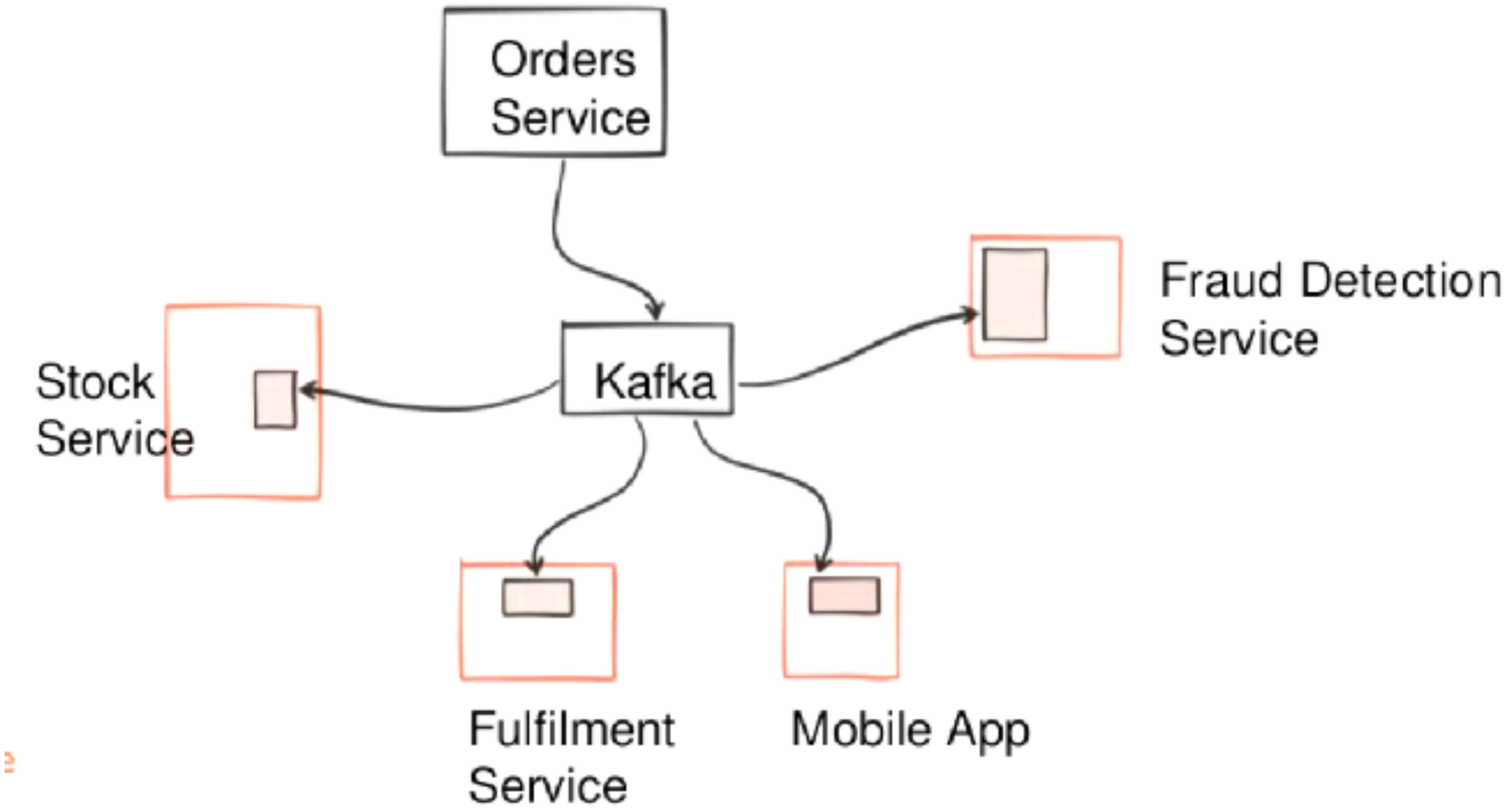
Capacity planning

CPU
Network and thread pool usages
Request latency
Network utilization
Disk utilization

Tips #4

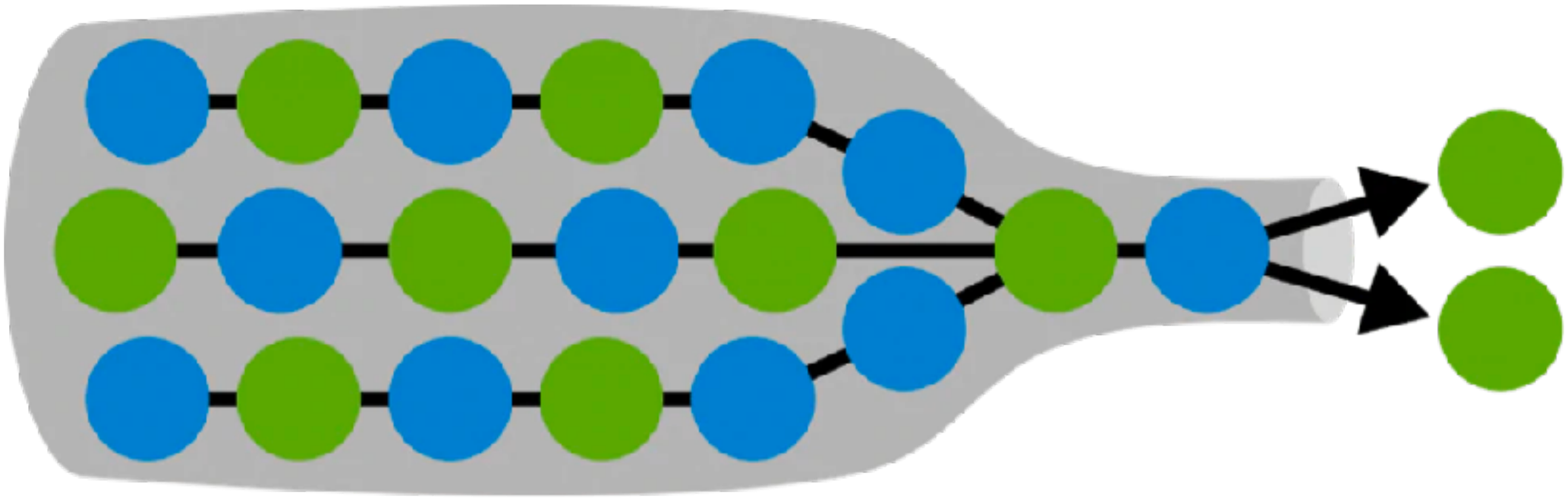
Monitoring brokers isn’t enough

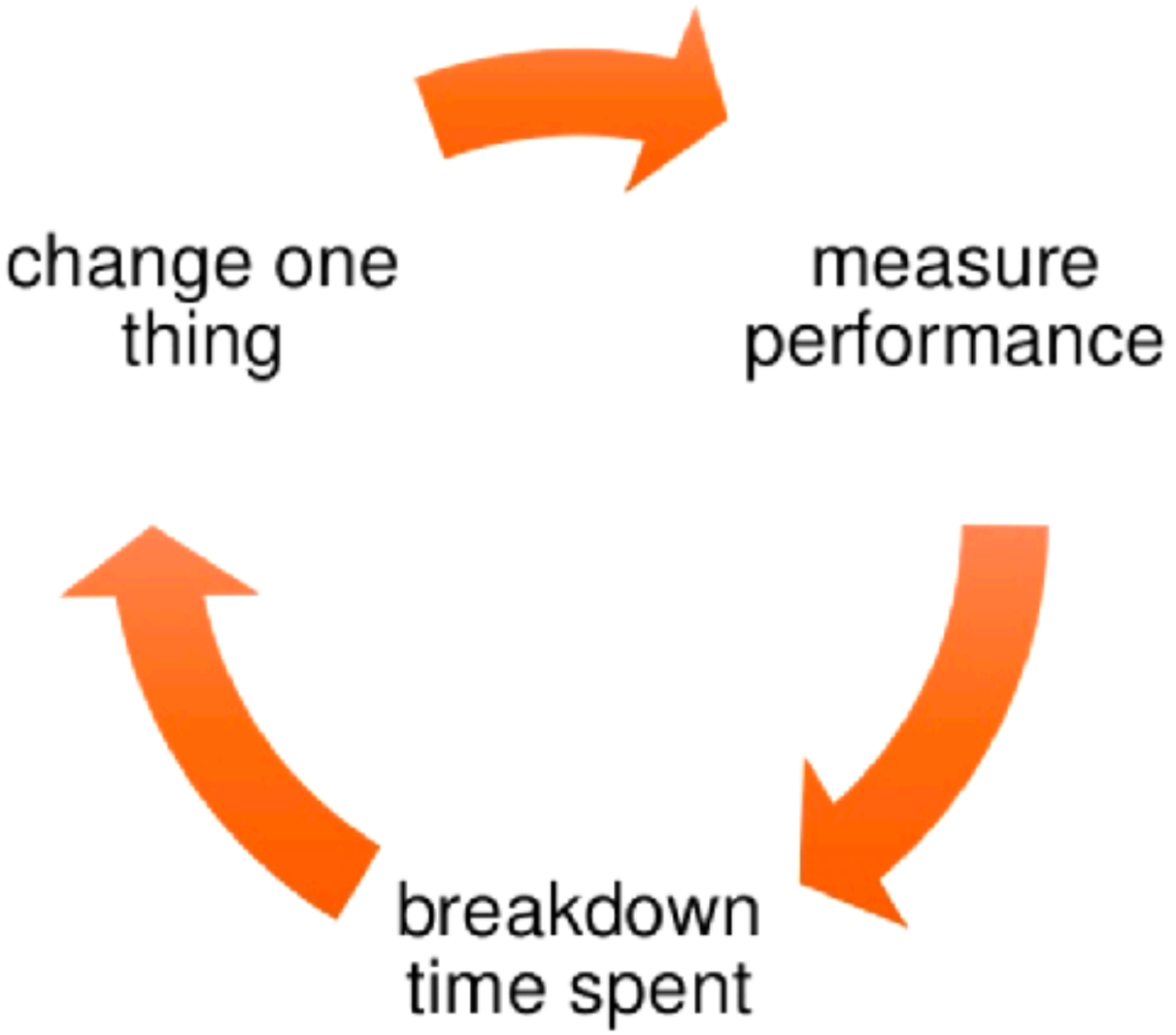
Need to monitor events



Tips #4

Tuning the bottlenecks





Life cycle of request

Client send request to broker
Network thread get request and put on queue
I/O thread/handler pick up request and process
(Read and write from/to disk)
Waiting for other brokers to ack messages
Put response on queue
Network thread send response to client

Kafka for operation

Kafka operations

Rolling restart of brokers
Update configurations
Rebalancing partitions
Increase replication factor
Add/replace/remove a broker
Upgrade a Kafka cluster with zero downtime

Security in Kafka

Security in Kafka

Authentication (SSL/SASL)
Authorization with ACL (Access Control List)
Data encryption

Plain text vs SSL