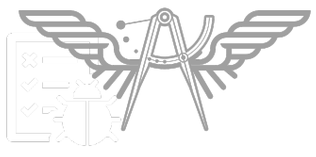
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Apache Kafka



Somkiat Puisungnoen

Somkiat | Home

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

Post | Photo/Video | Live Video | Life Event

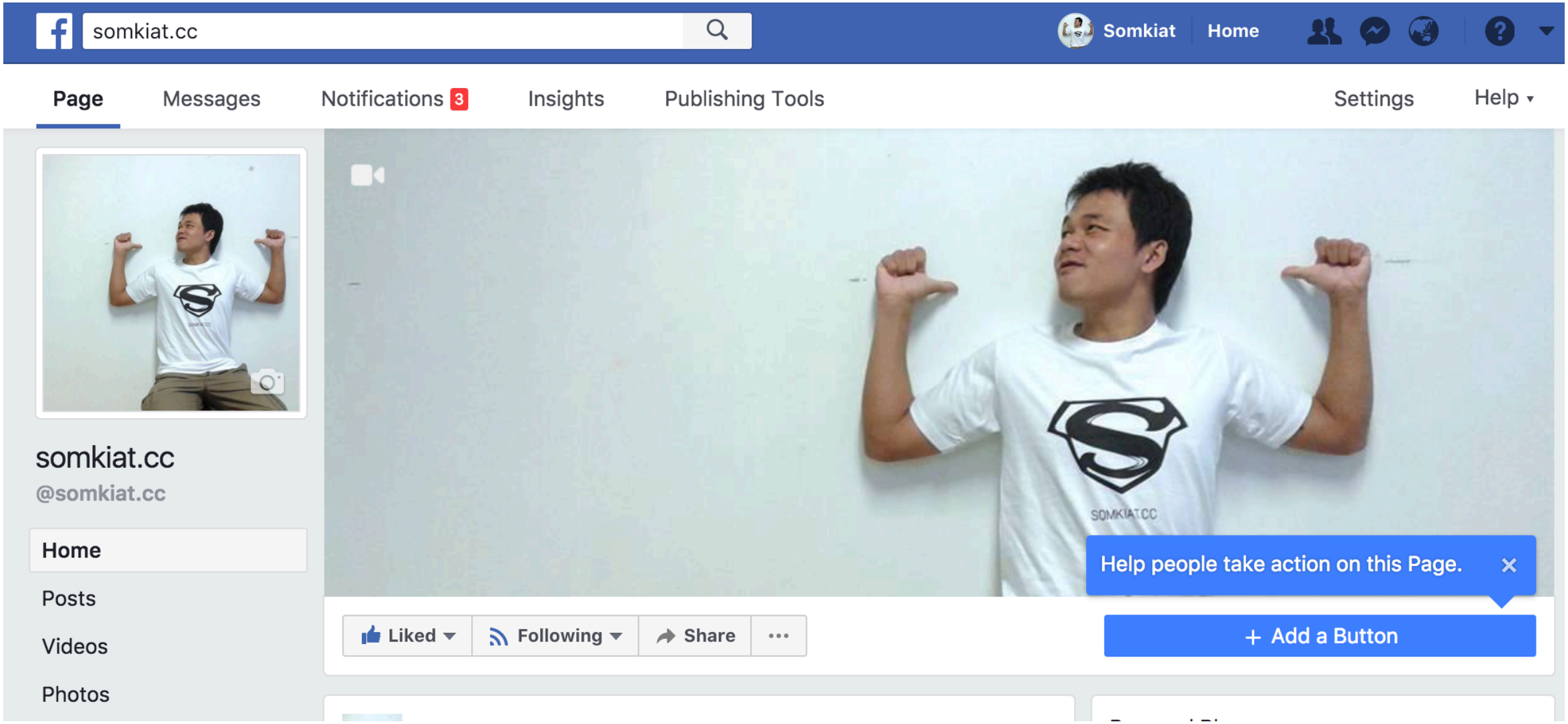
What's on your mind?

Public | Post

Somkiat Puisungnoen

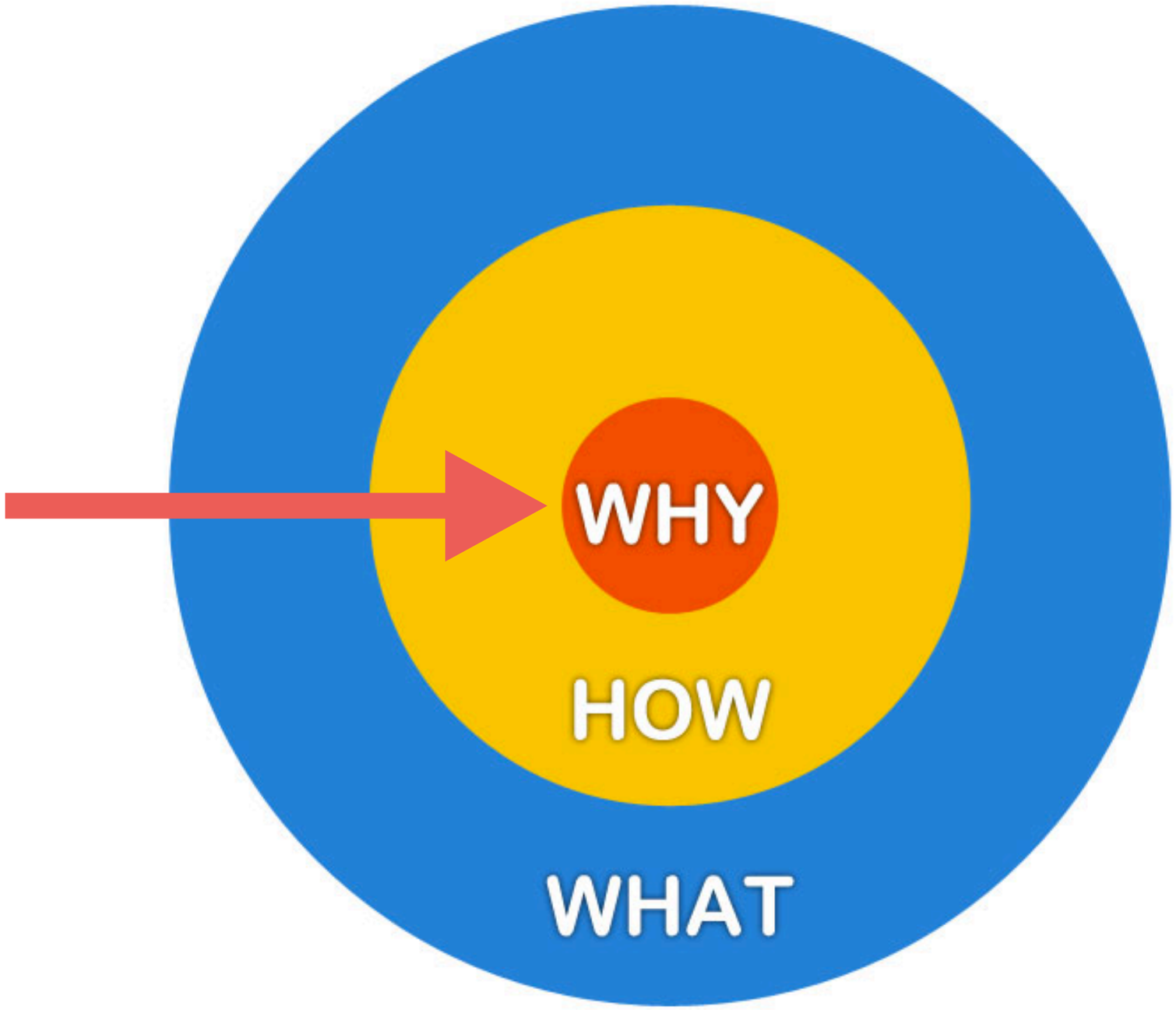
15 mins · Bangkok ·

Java and Bigdata



Apache Kafka

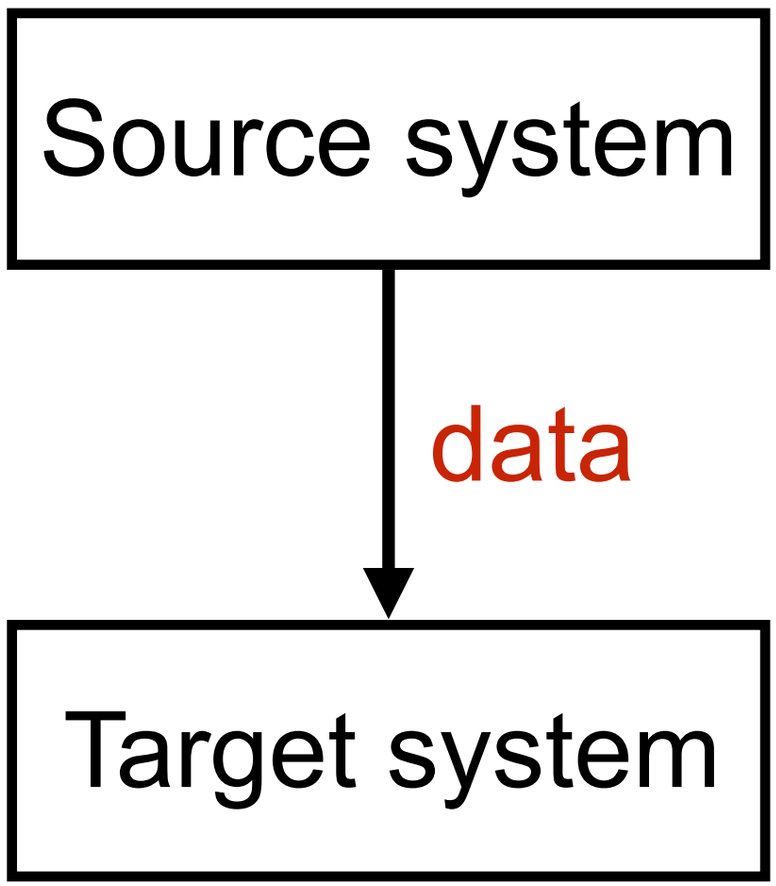
- Why Apache Kafka ?
- What is Apache Kafka ?
- Architecture
- Topologies and Tools
- Workshop



Apache Kafka

Why

Starting with simple

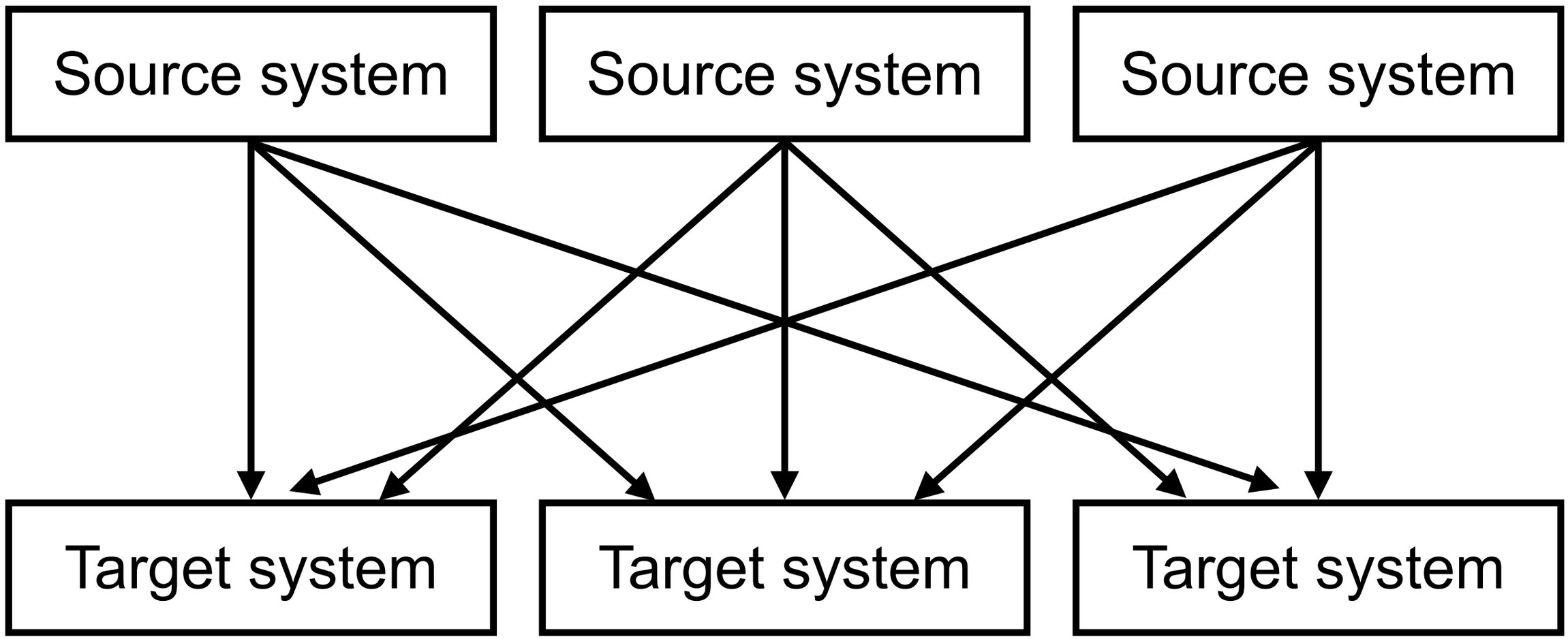


Why

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

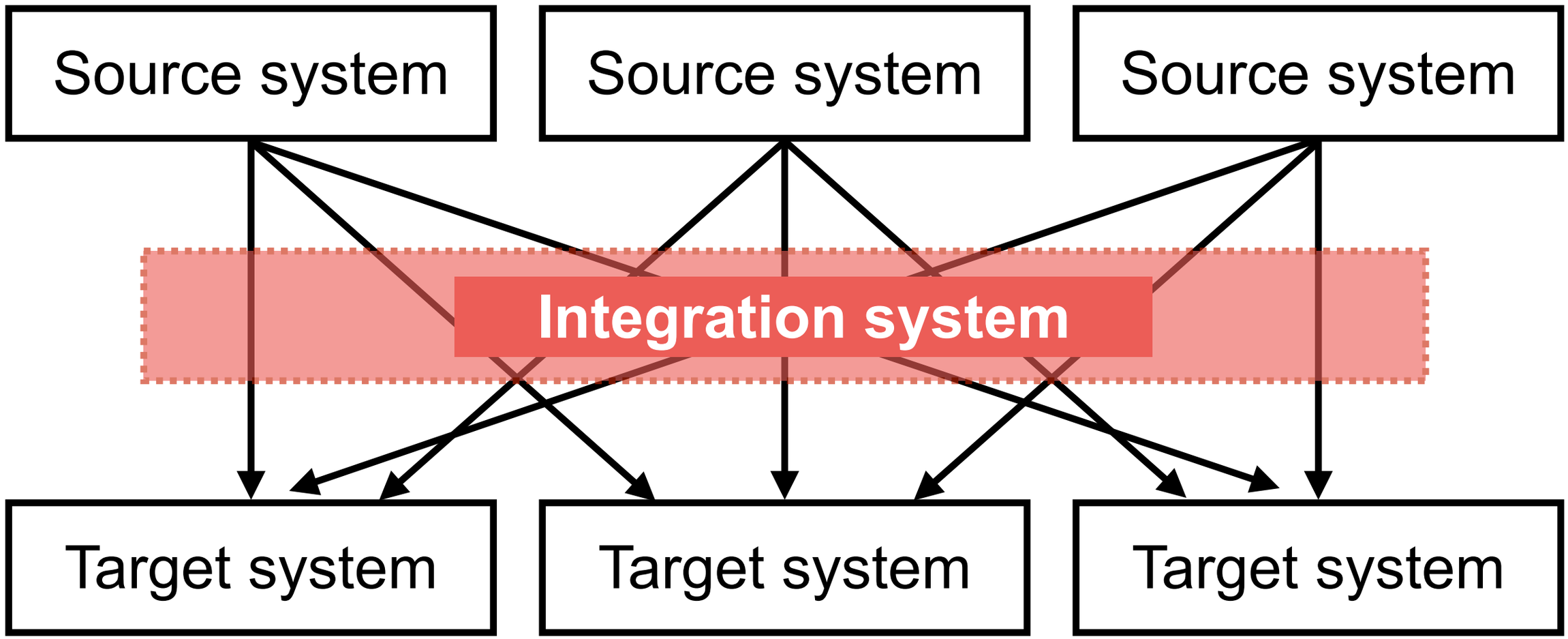
Why

Complex system !!

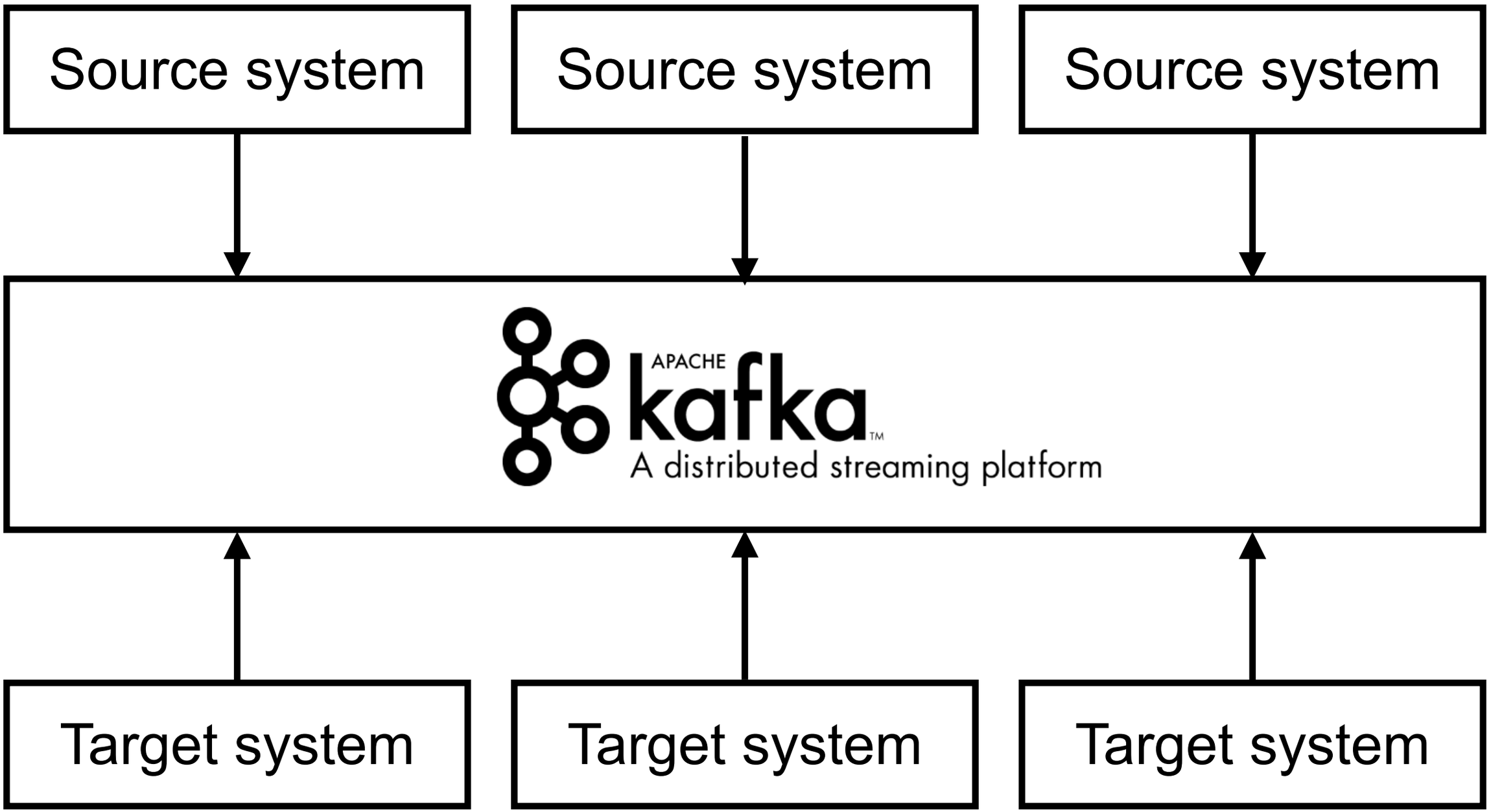


Why

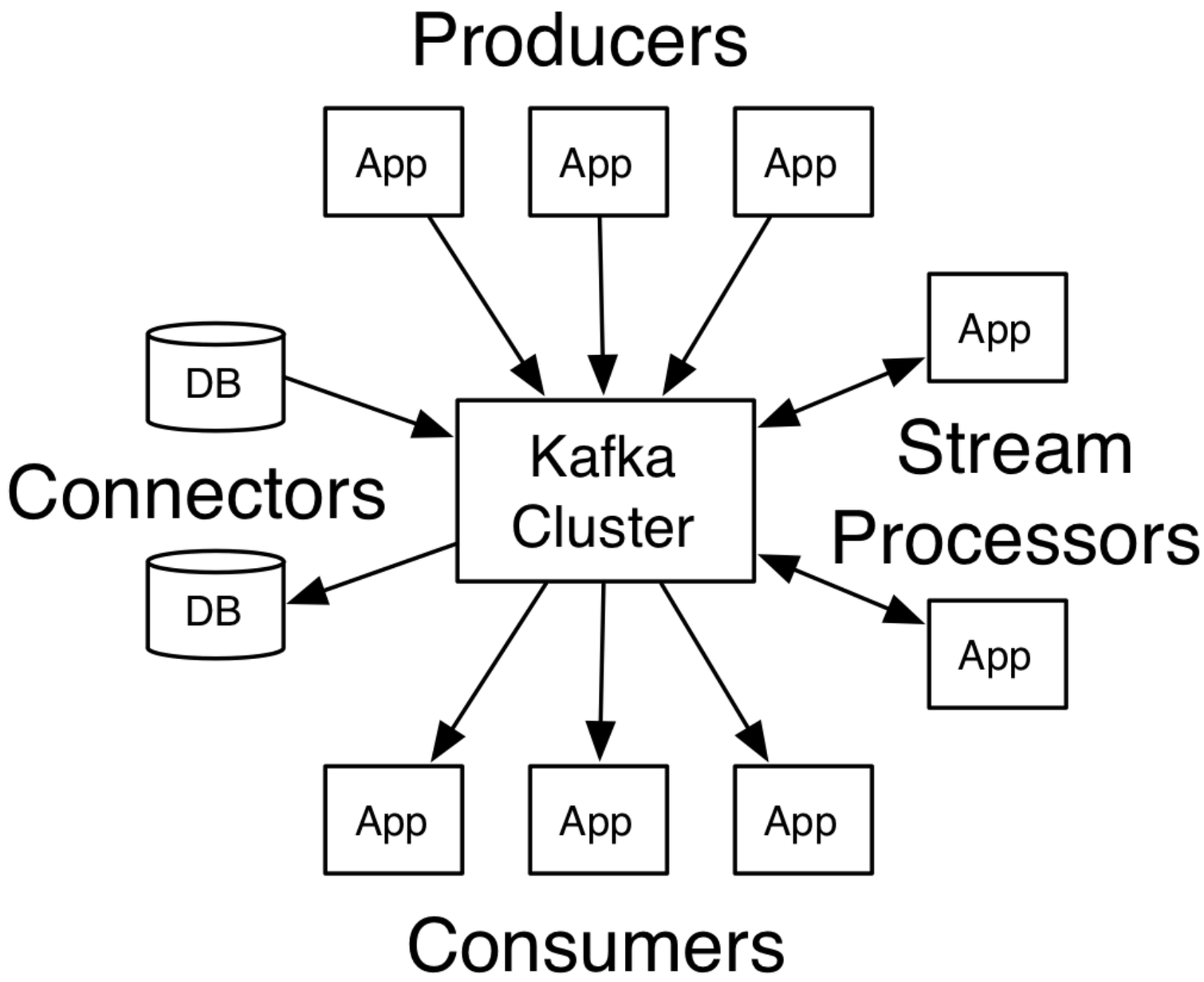
Need integration system



Apache Kafka



Apache Kafka



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

Why Apache Kafka ?

Created by LinkedIn
Open source project (maintain by Confluent)
Distributed system
Resilient architecture, fault tolerant
Horizontal scalability
High performance/ low latency

Used by

airbnb

Uber

LinkedIn

NETFLIX

Walmart

https://cwiki.apache.org/confluence/display/KAFKA/Powered+By

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

Kafka terminologies

Producers
Consumers
Topics
Partitions
Broker
Consumer groups
Cluster/Replicate
Offset

Learning paths

Part 1

- Kafka theory
- Starting Kafka
- Kafka CLI
- Kafka with Java

Part 2

- Configuration
- Producers
- Consumers
- Monitoring

Kafka theory

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**

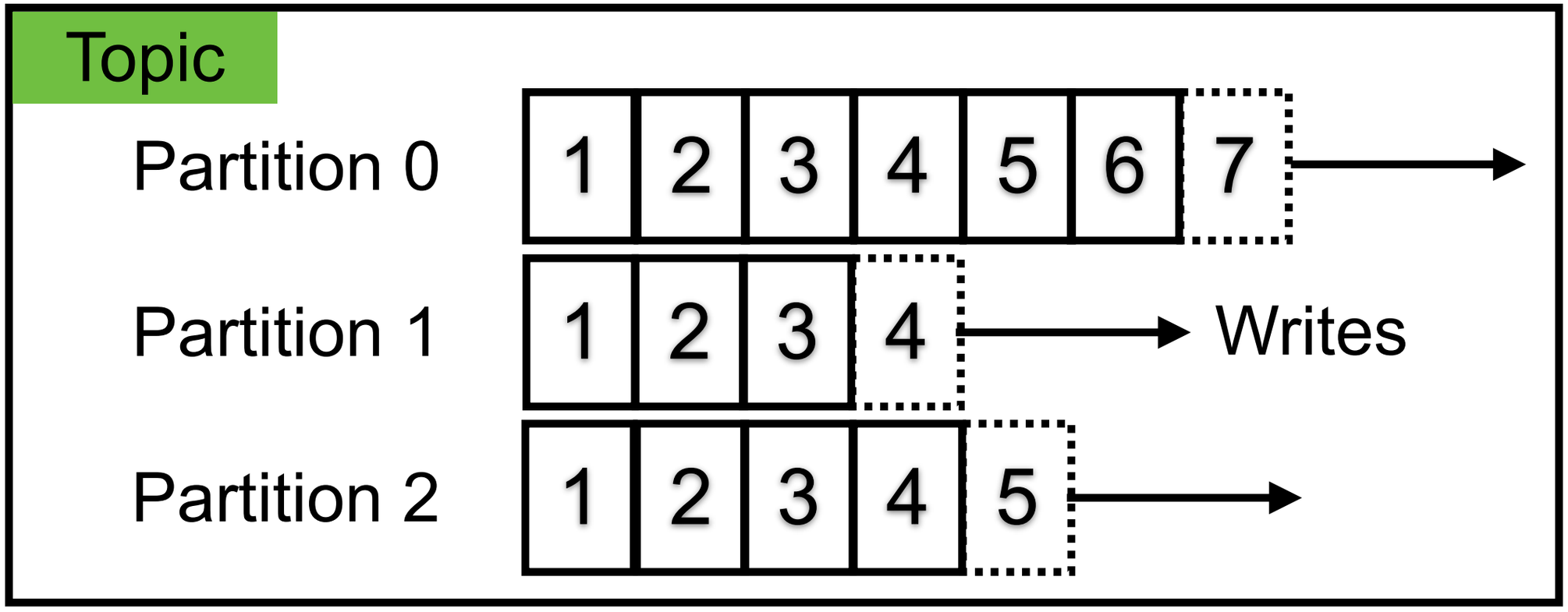
Topics, partitions and offsets

Partitions

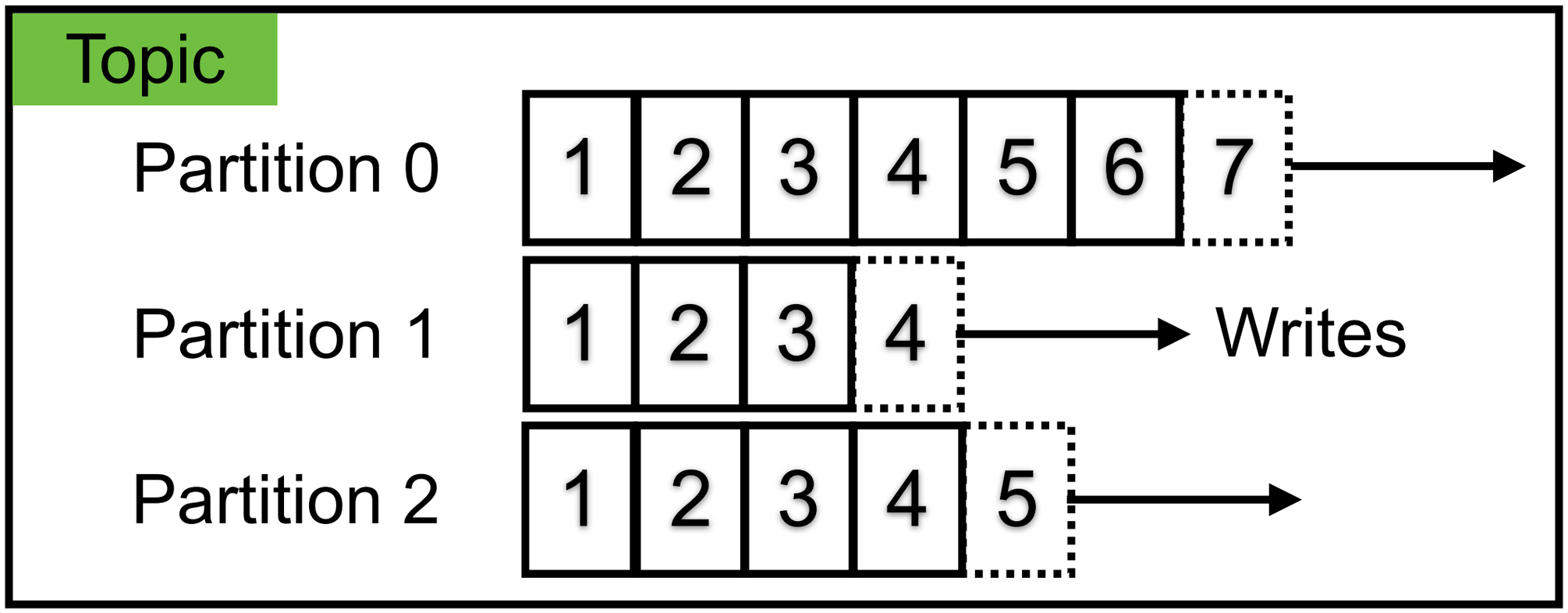
Topics are split in partitions

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**)

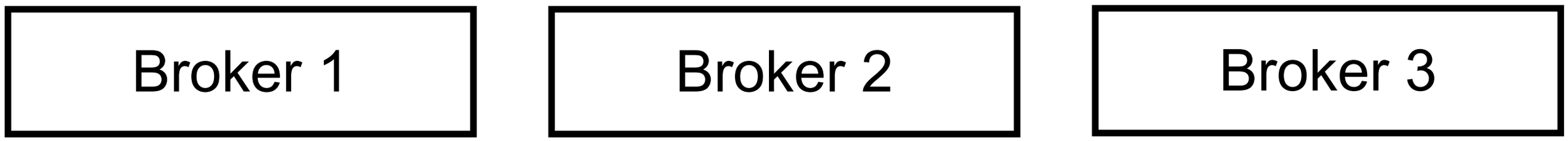
Data is **immutable** (can't be changed)

Data is assigned **randomly** to a partition

Brokers and topics

Brokers

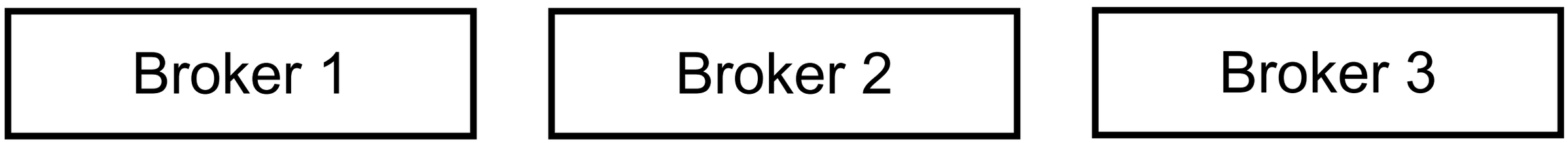
Kafka cluster is composed of multiple brokers (servers)
Each broker is identified by ID (integer)
Each broker contains certain topic partitions
After connecting any broker (bootstrap broker), you will connected to the entire cluster



Brokers and topics

Brokers

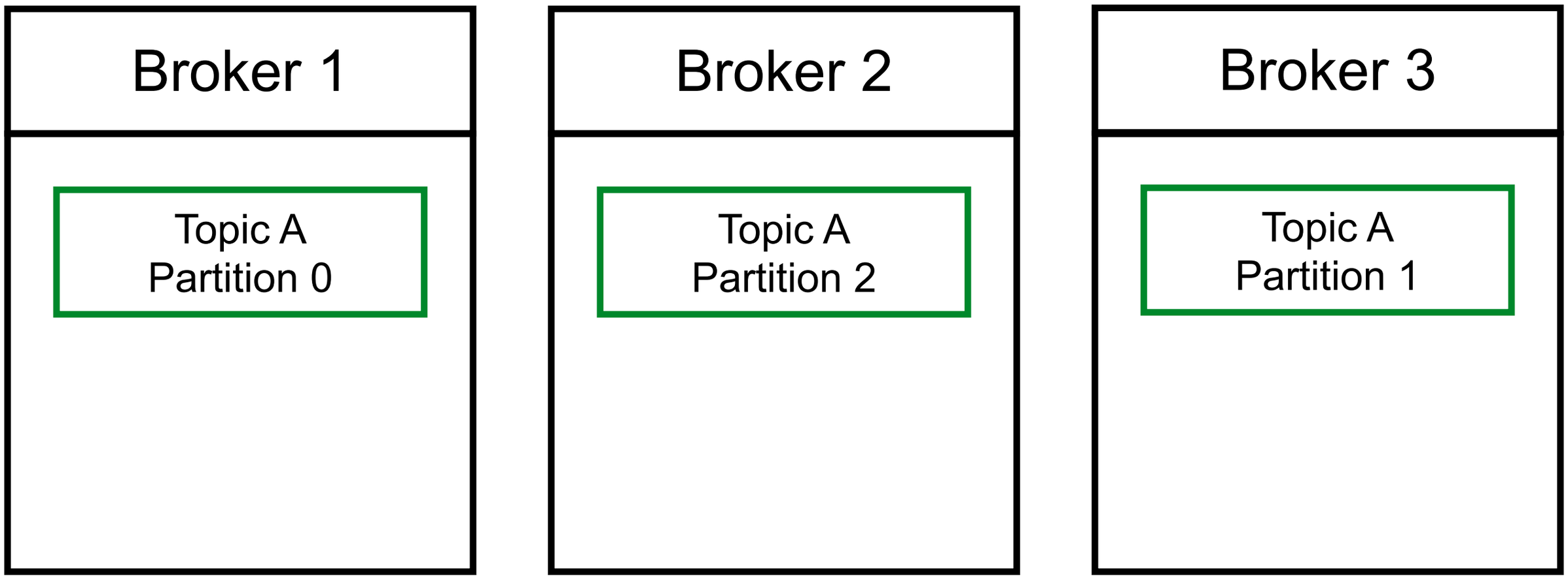
Good number to start id **3 brokers**



Brokers and topics

Example

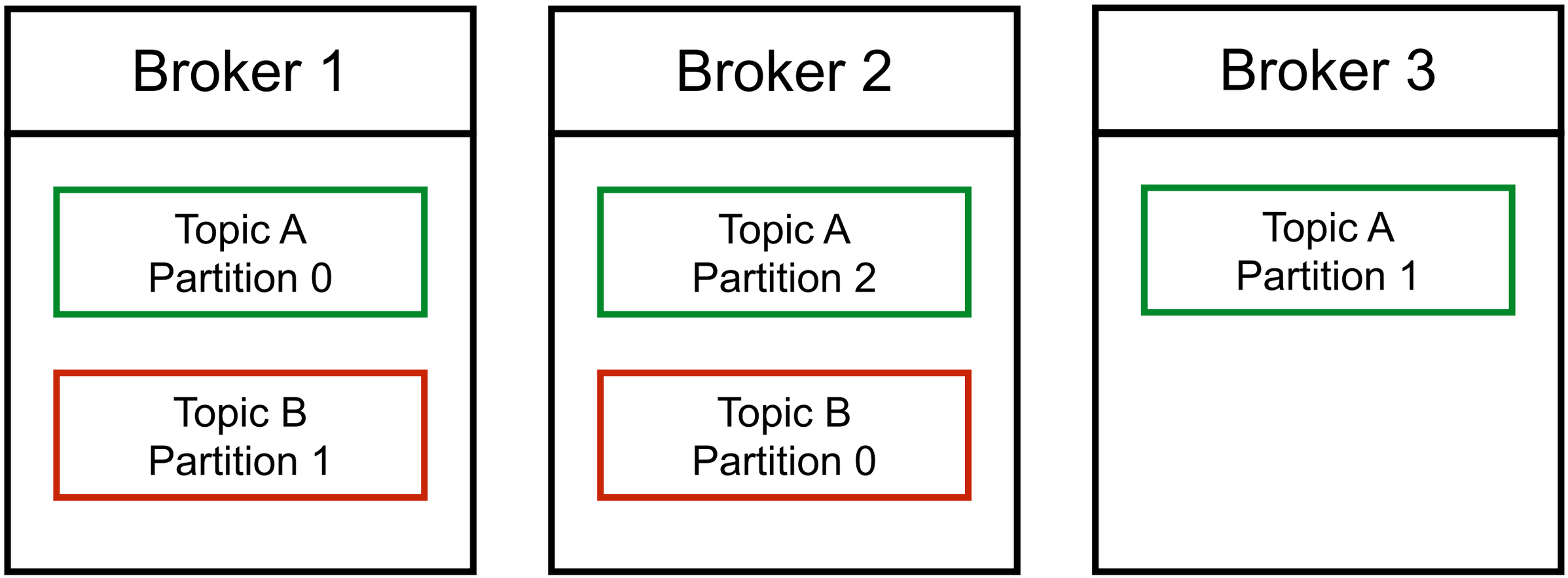
Topic A with 3 partitions



Brokers and topics

Example

Topic B with 2 partitions



Topic with replication factor

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

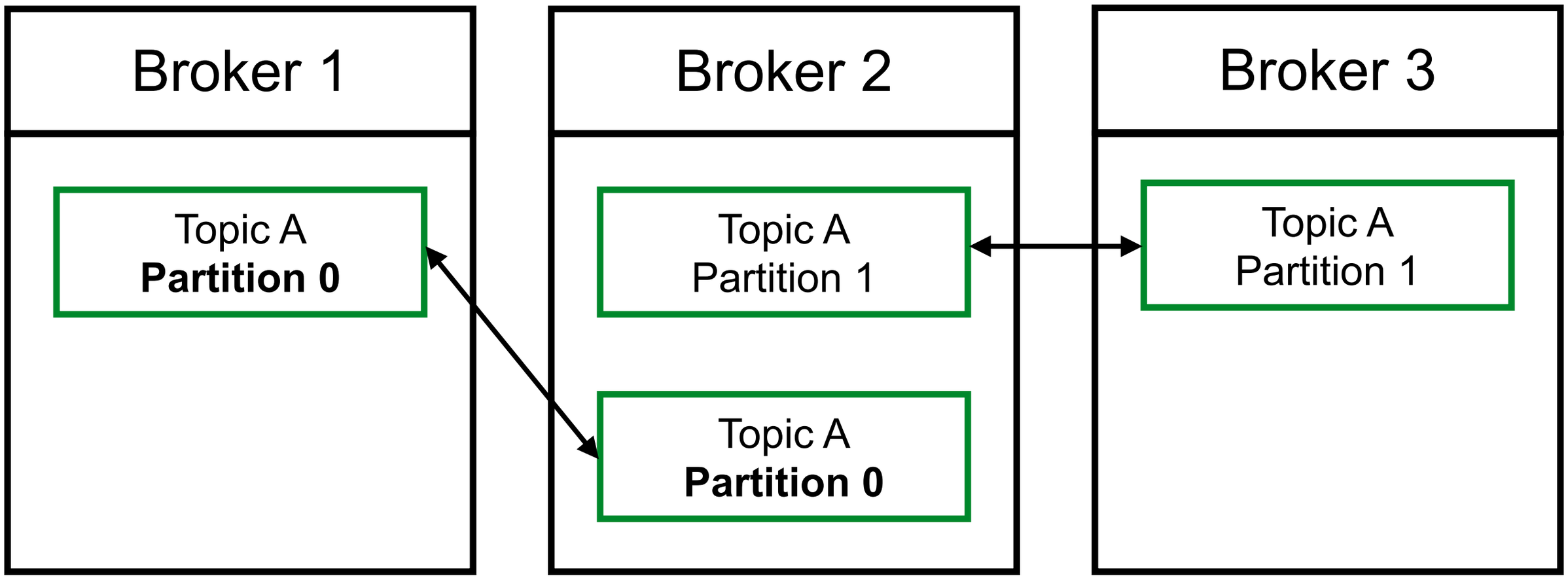
replica factor < no. of brokers

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

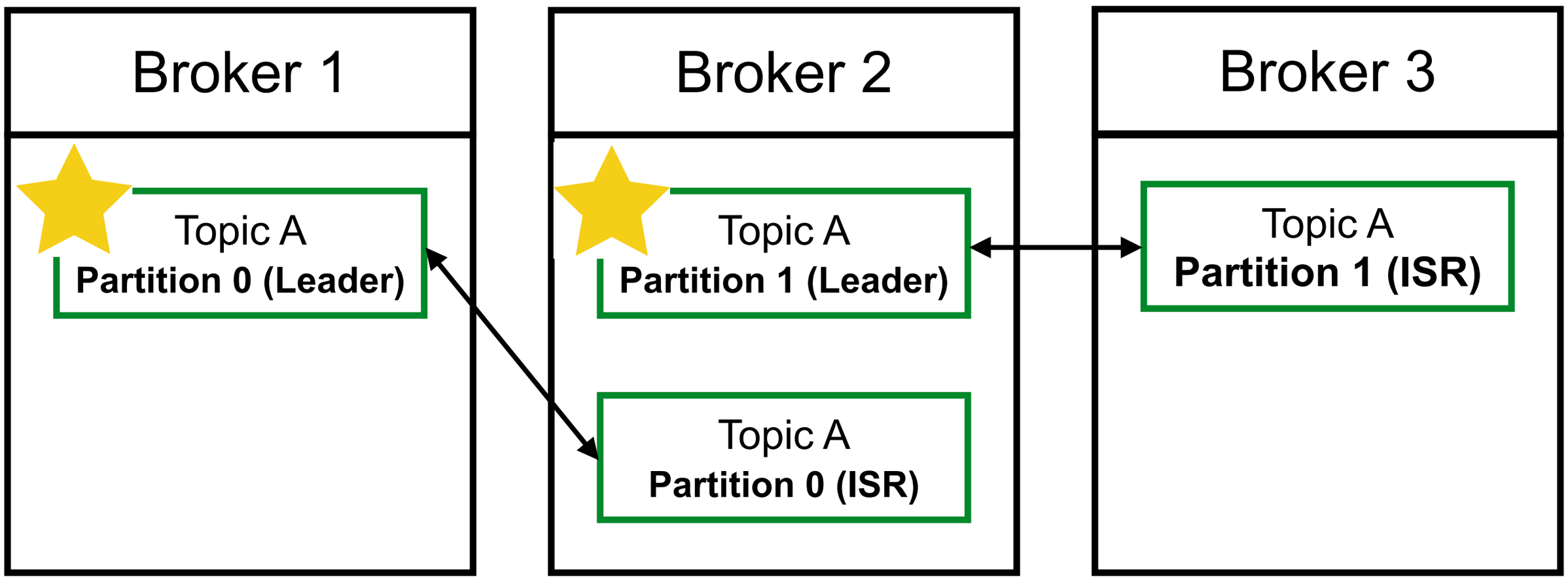
Other brokers will synchronize the data

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

Leader for a partition

Example

Topic A with 2 partitions and replication factor = 2



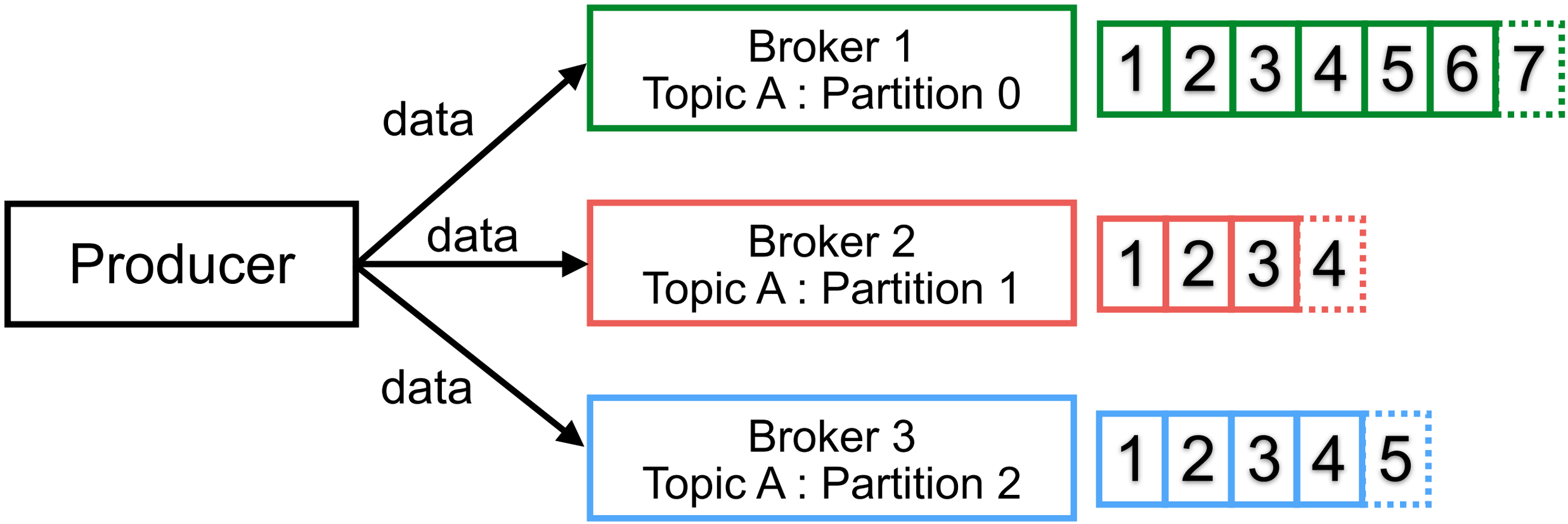
Producers

Producers write data to topics

Producers automatically know to broker and partition to write

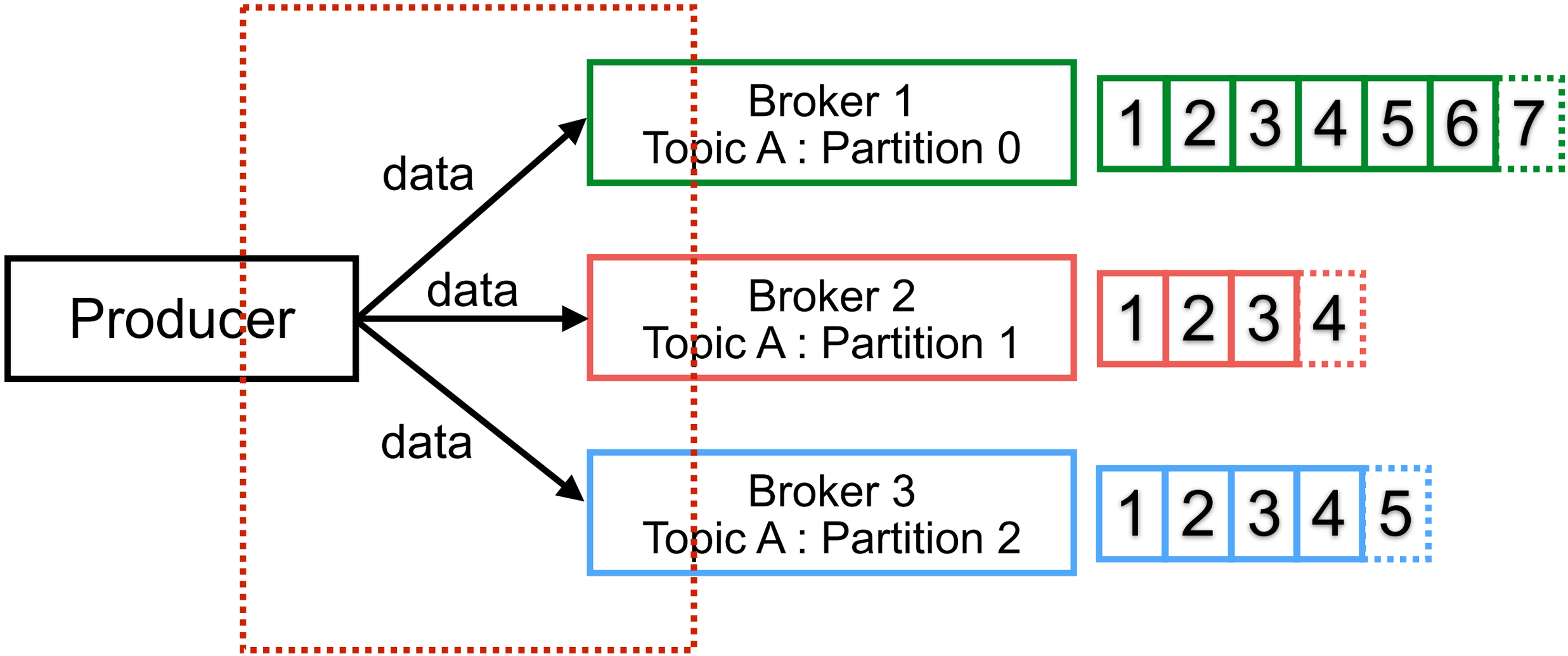
When broker failures, producers will automatically recover

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

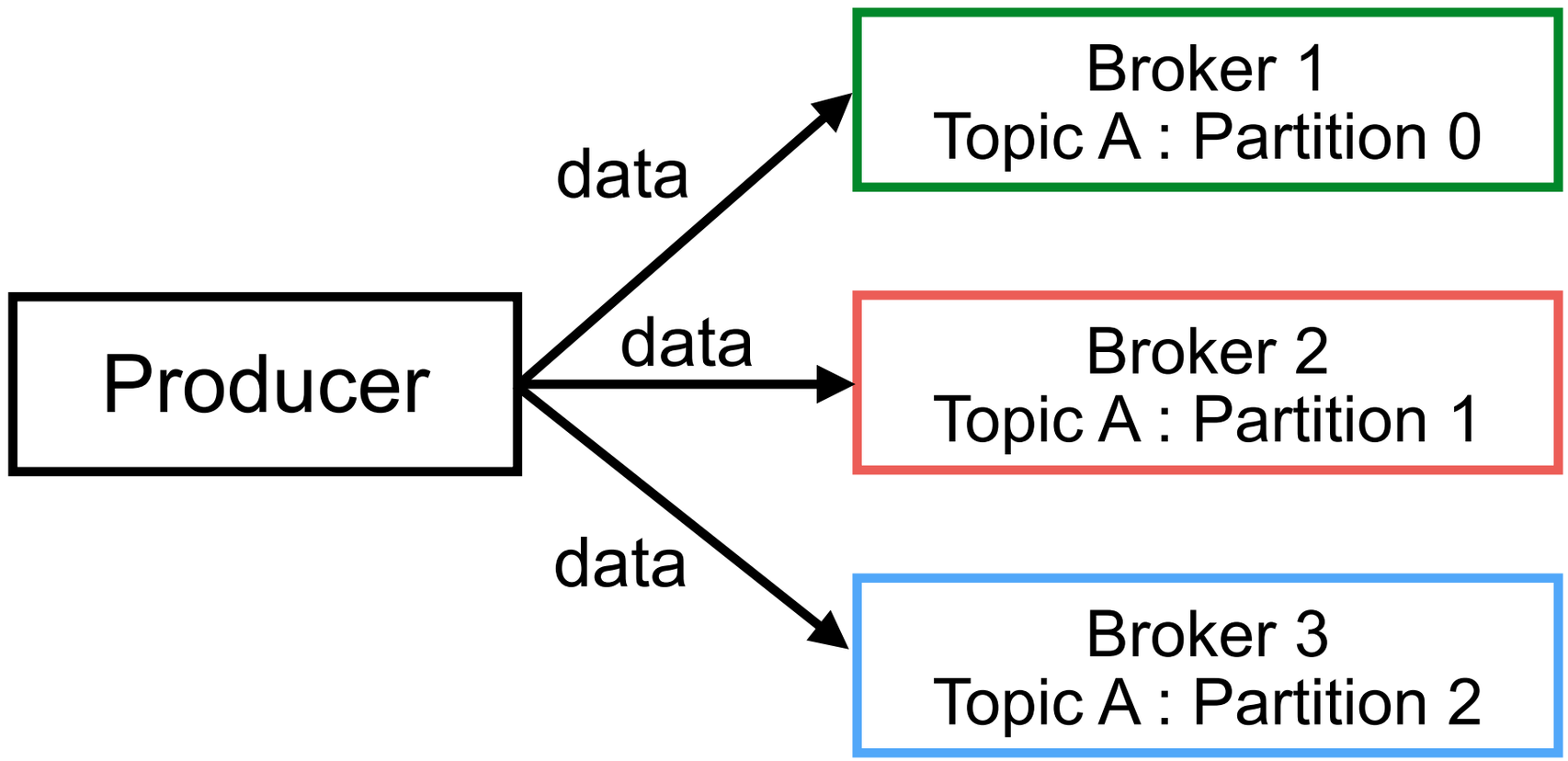
Producer will wait for **leader + ISR** acknowledgment

No data loss

Producers with message keys

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

Key = null, data is sent **round-robin**



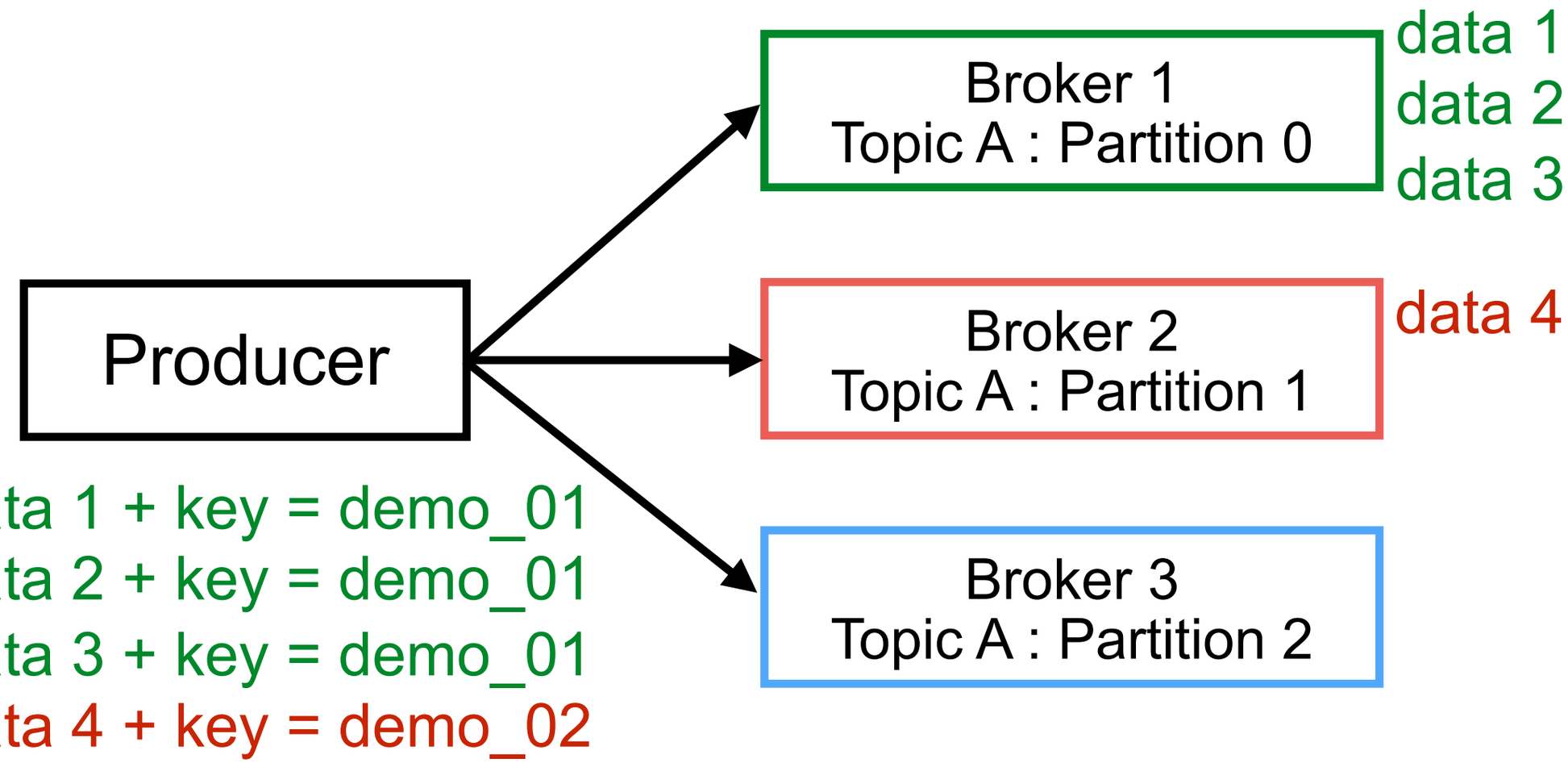
Producers with message keys

```
public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[]
    if (keyBytes == null) {
        return stickyPartitionCache.partition(topic, cluster);
    }
    List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
    int numPartitions = partitions.size();
    // hash the keyBytes to choose a partition
    return Utils.toPositive(Utils.murmur2(keyBytes)) % numPartitions;
}
```

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

Why use message keys ?

You need message ordering



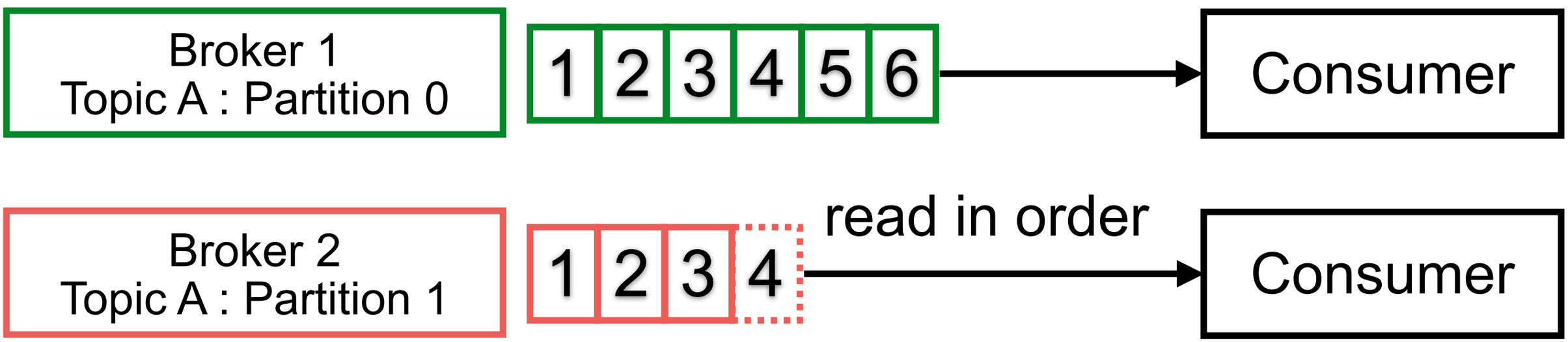
Consumers

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

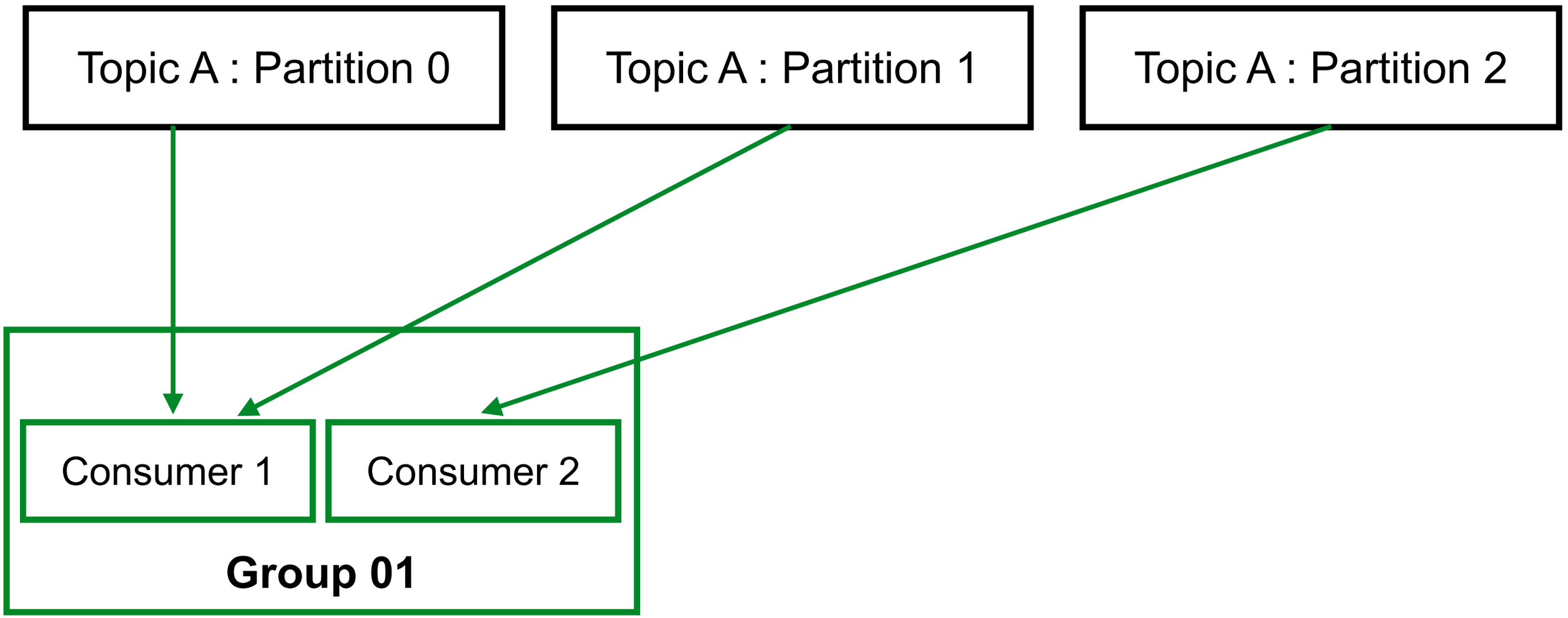


Consumer groups

Consumers read data in consumer groups

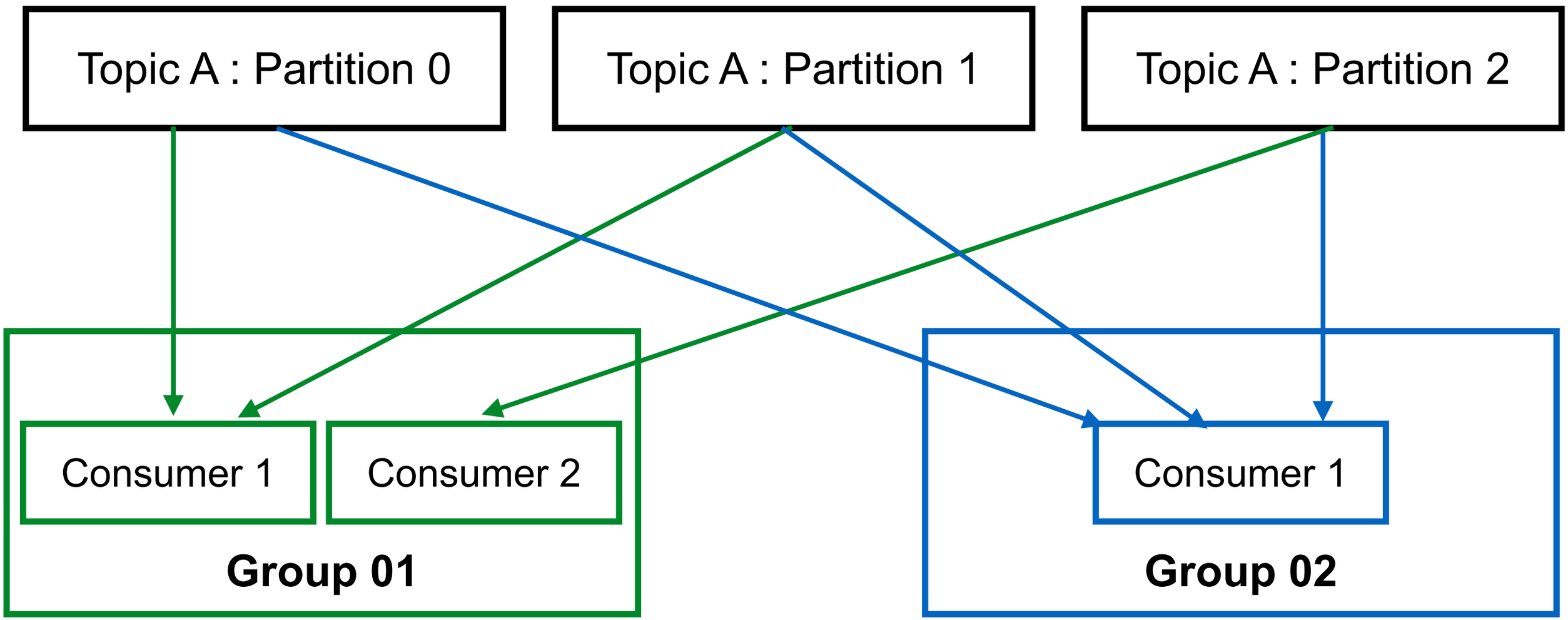
Each consumer in a group read from exclusive partitions

Consumer groups



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

Consumer groups



Partition assignment strategies

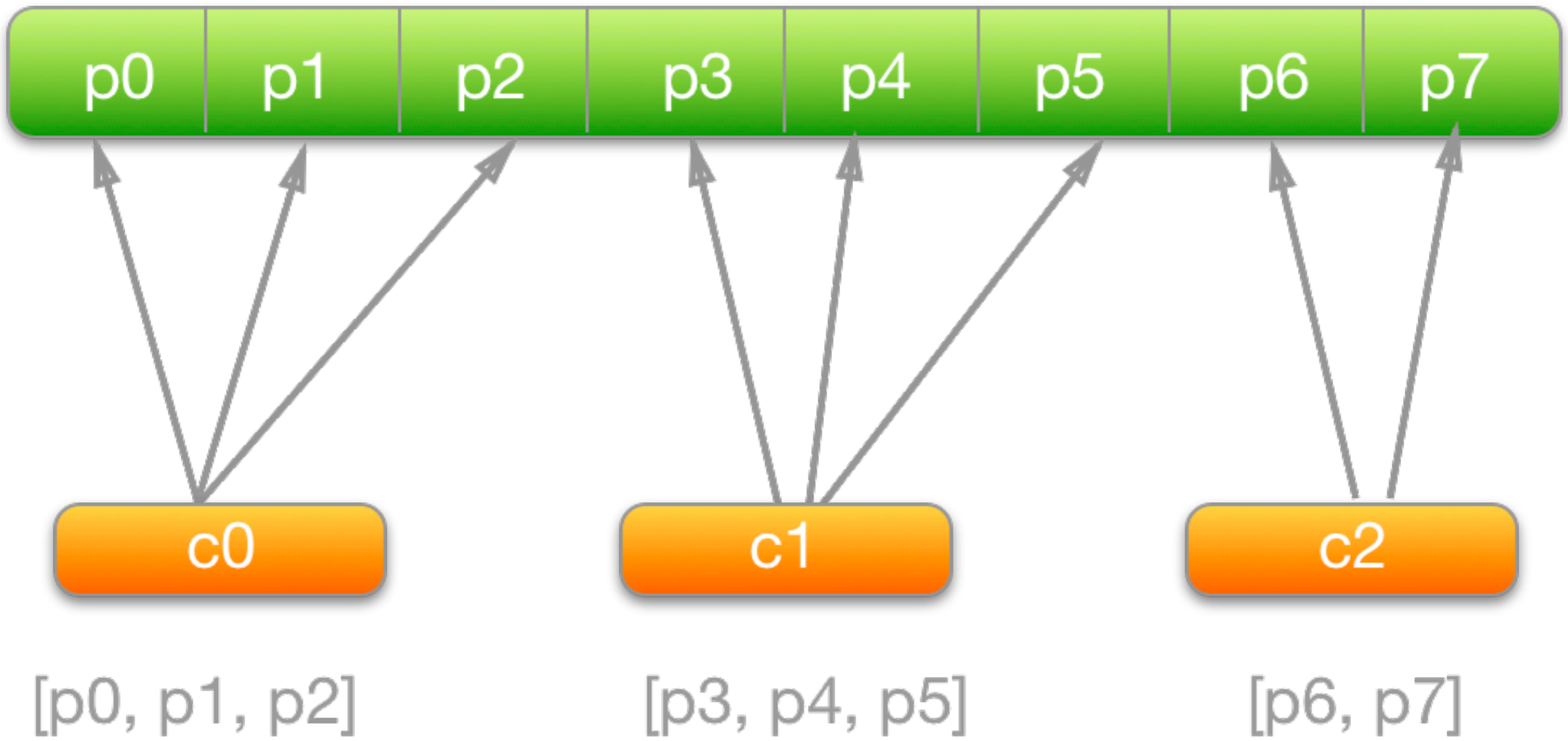
Range (default)

Round Robin

Sticky

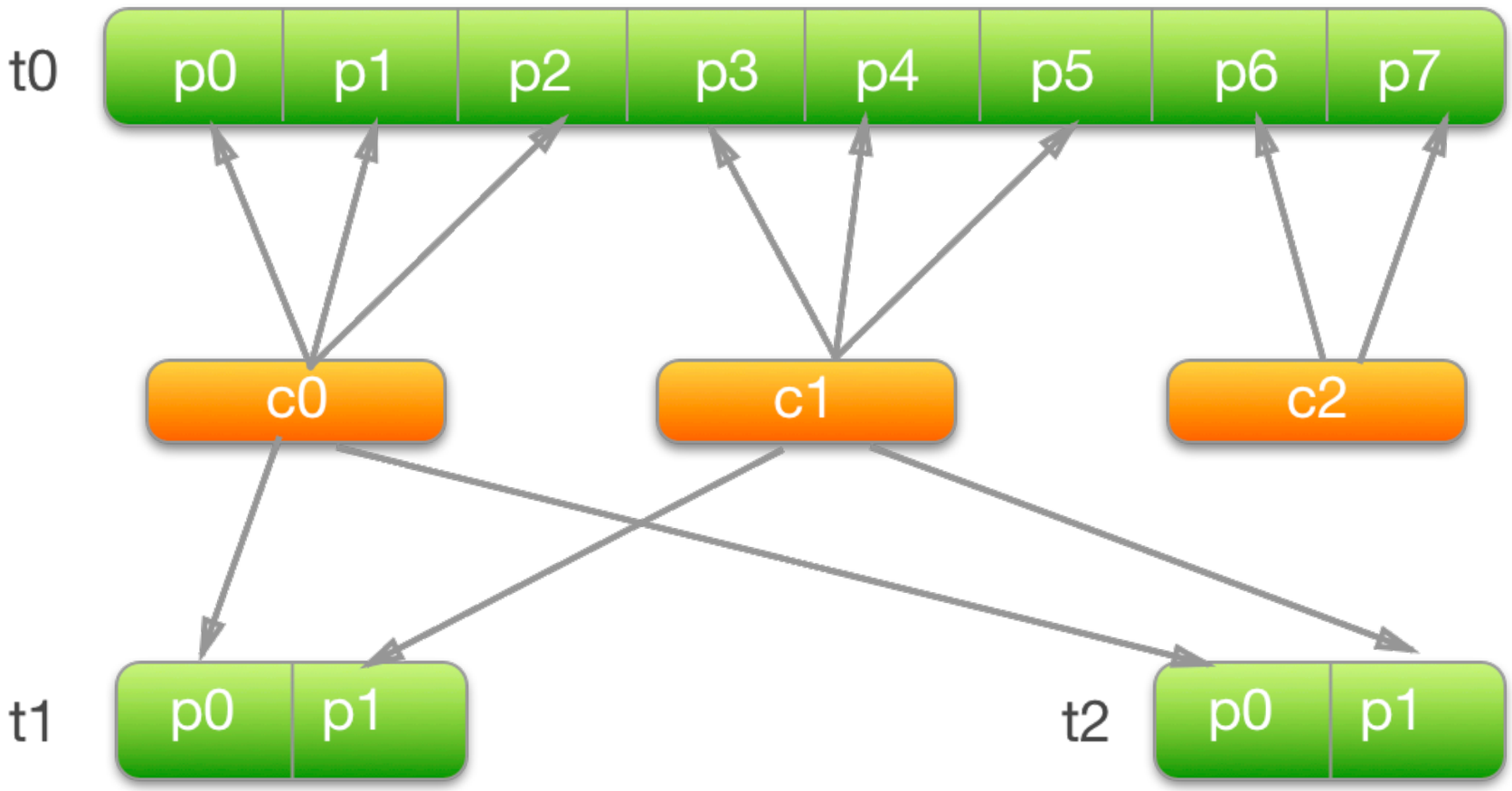
Customize (AbstractPartitionAssignor)

Range assignor



org.apache.kafka.clients.consumer.RangeAssignor

Range assignor

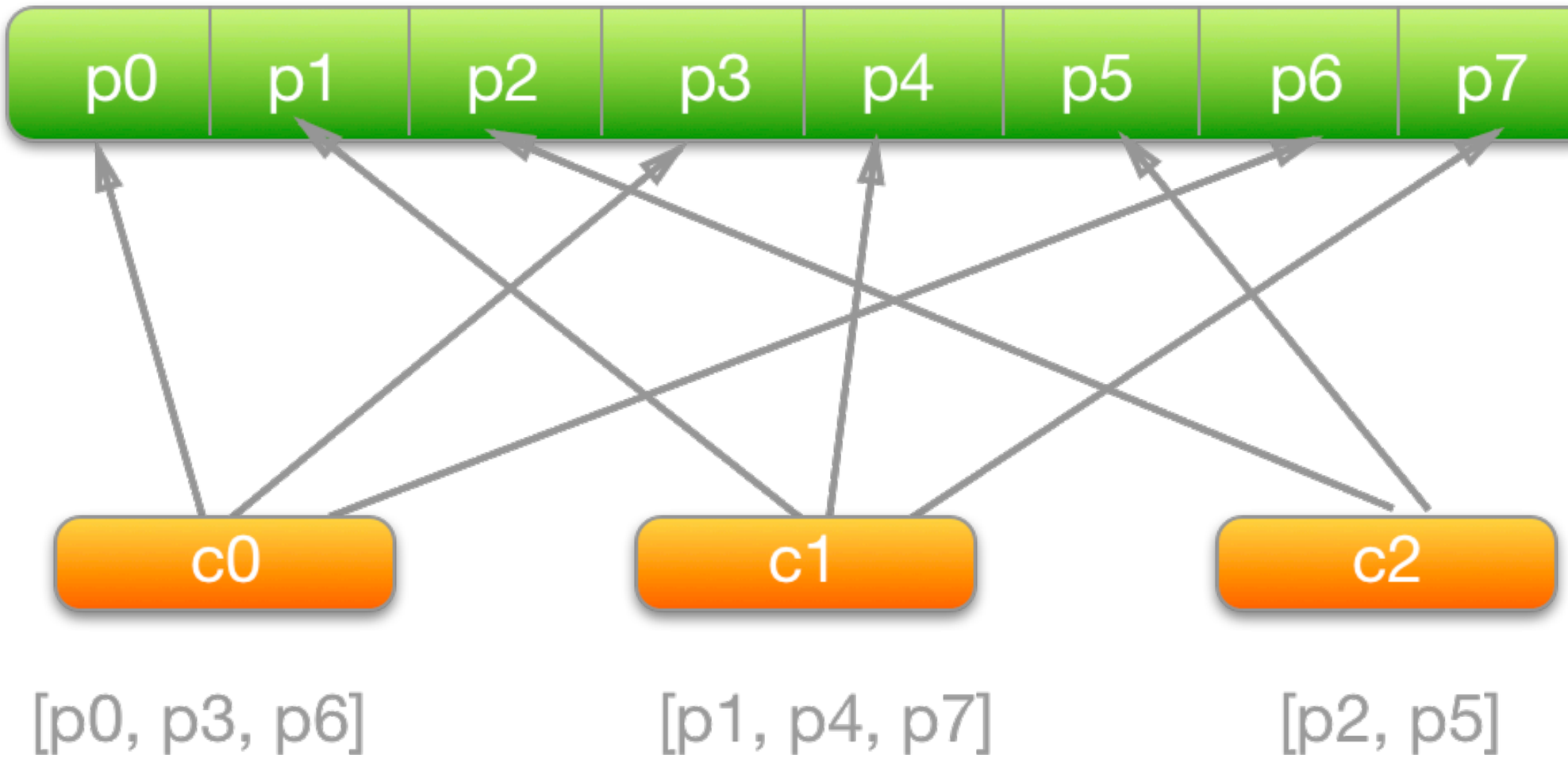


c0 -> [t0p0, t0p1, t0p2, t1p0, t2p0]

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

c2 -> [t0p6, t0p7]

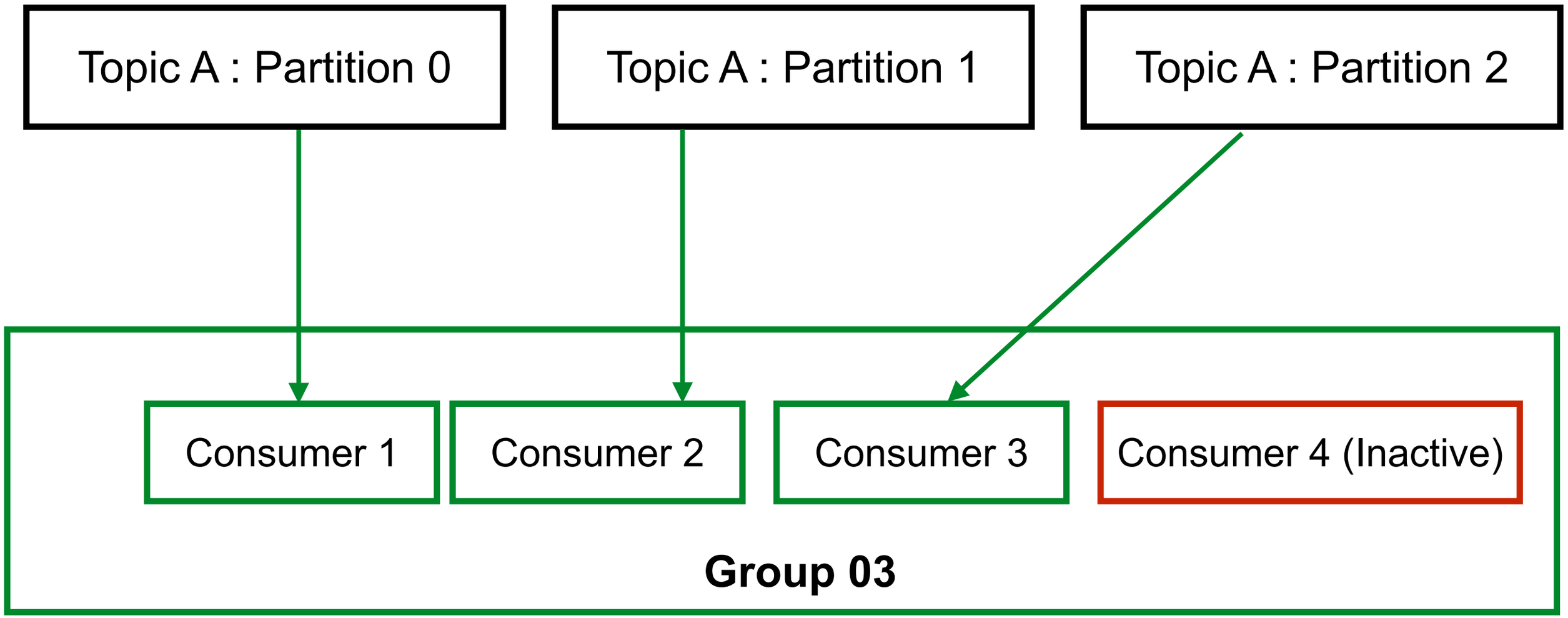
Round Robin assignor



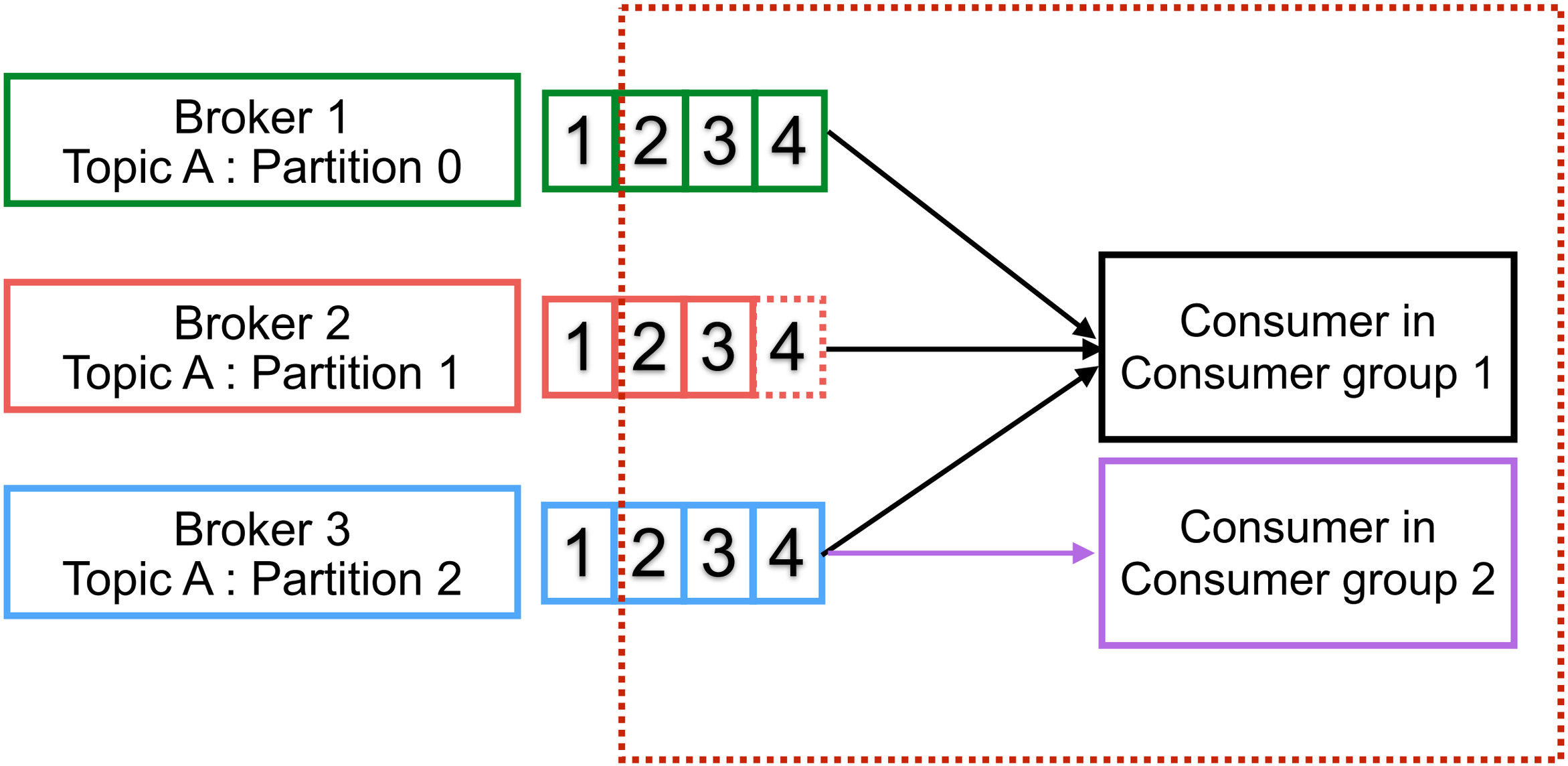
org.apache.kafka.clients.consumer.RoundRobinAssignor

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

1. At most once

Offsets are committed as soon as the message is received

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

2. At lease once

Offsets are committed after the message is processed

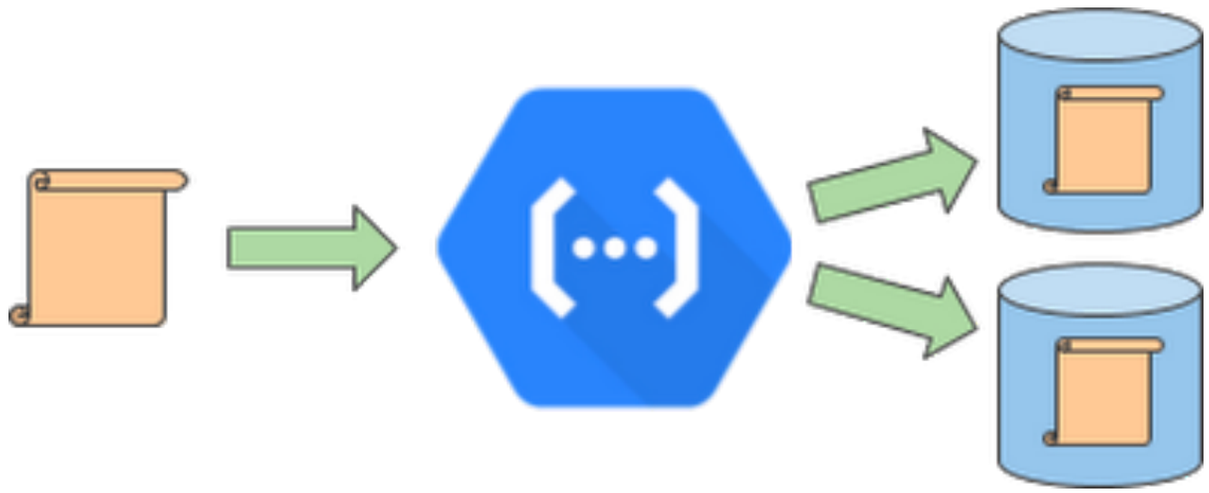
Messages are never lost but may be **redelivered**

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

2. At lease once

Make sure your processing is **idempotent**

Processing again the message not impact to your system !!



3. Exactly once

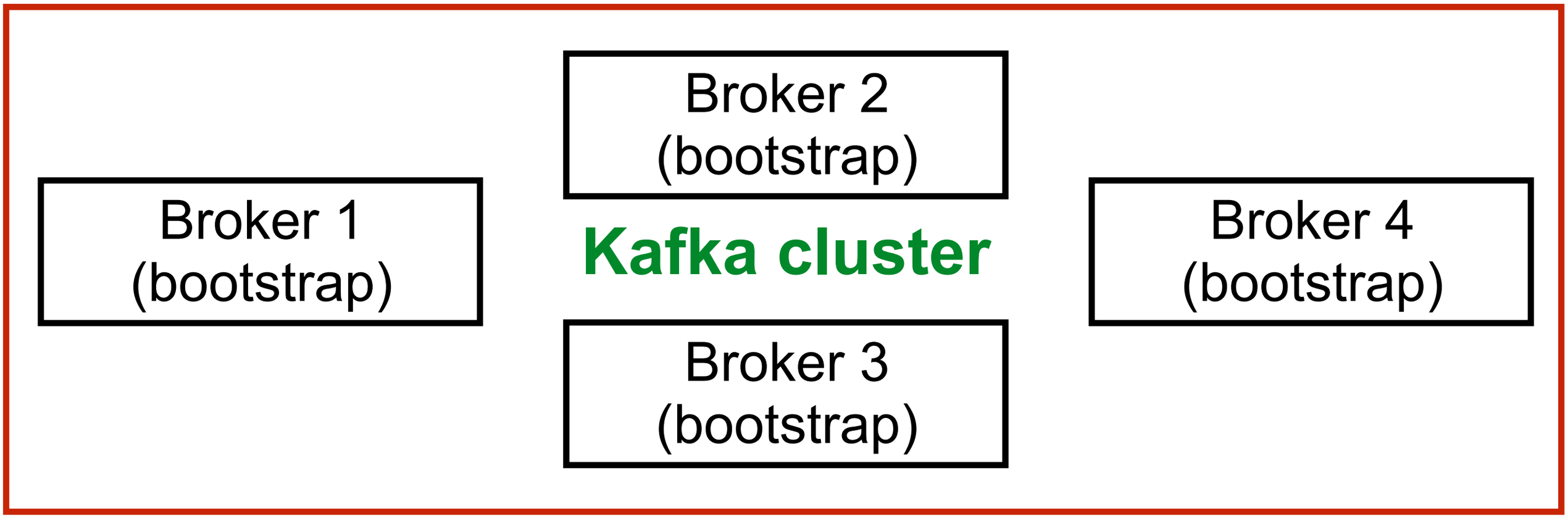
Each message is delivered once and only once

Can be achieved for Kafka (Workflow, Stream API)

Kafka broker discovery

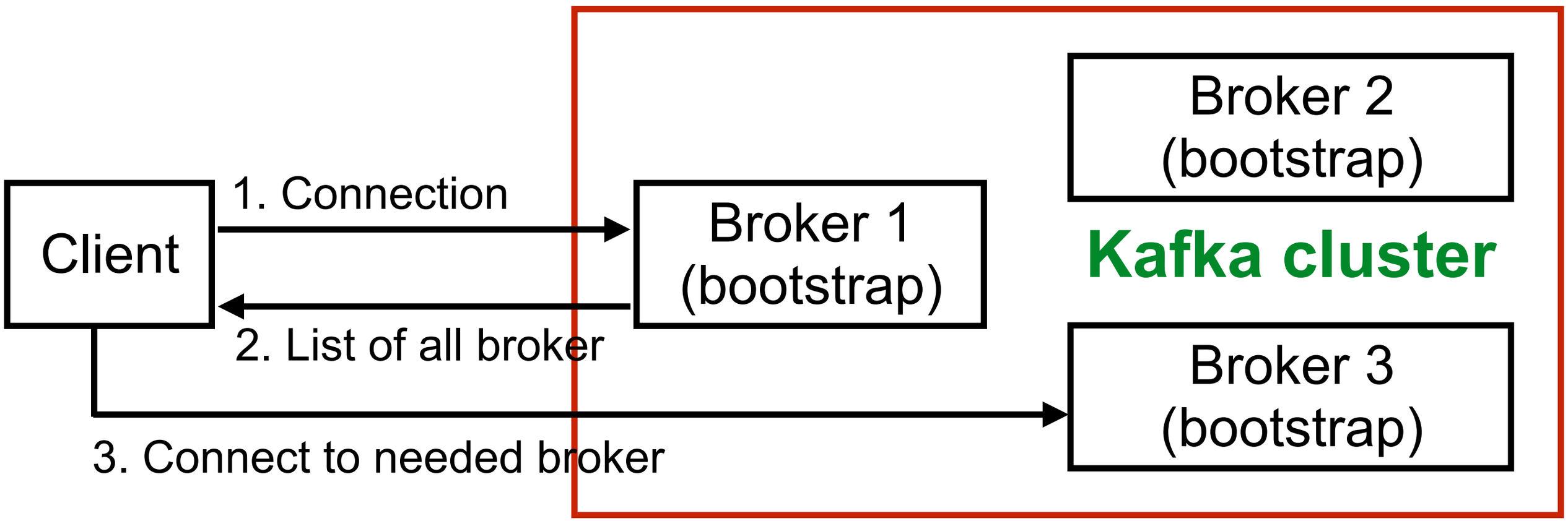
Every Kafka broker is called **“bootstrap server”**

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

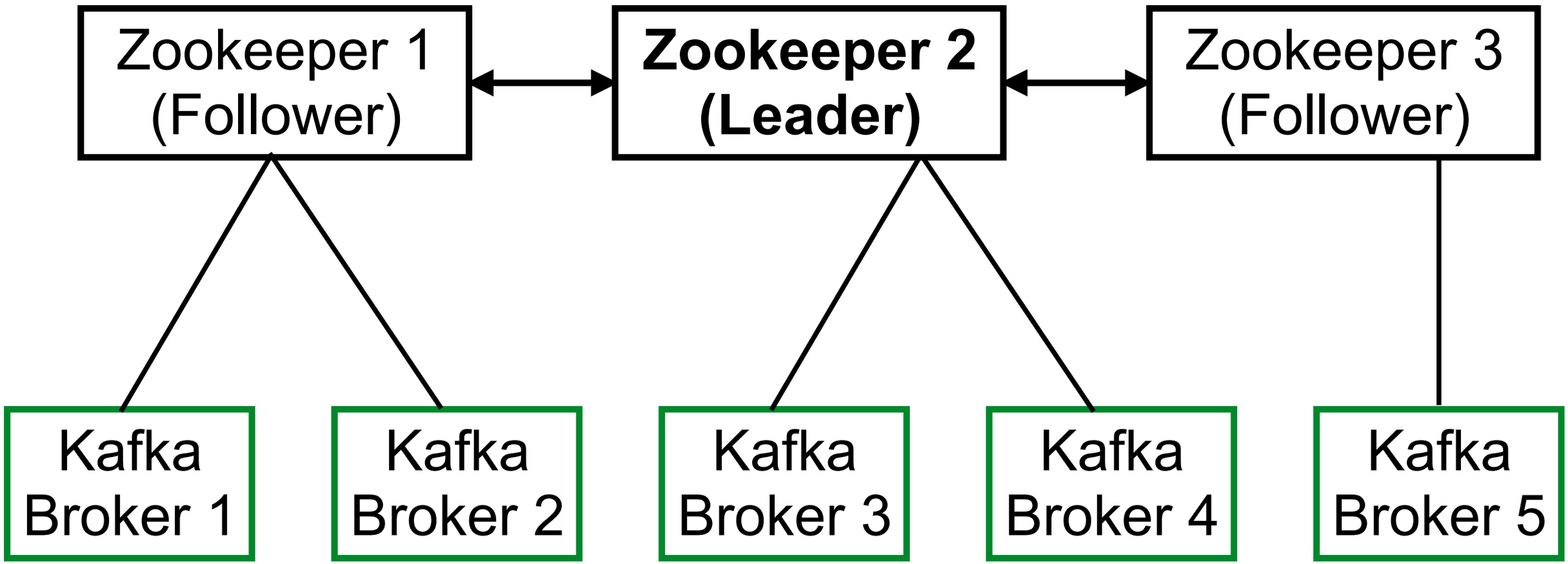
Zookeeper sends **notifications** to Kafka in case of changes (new topic, broker die)

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

Zookeeper architecture

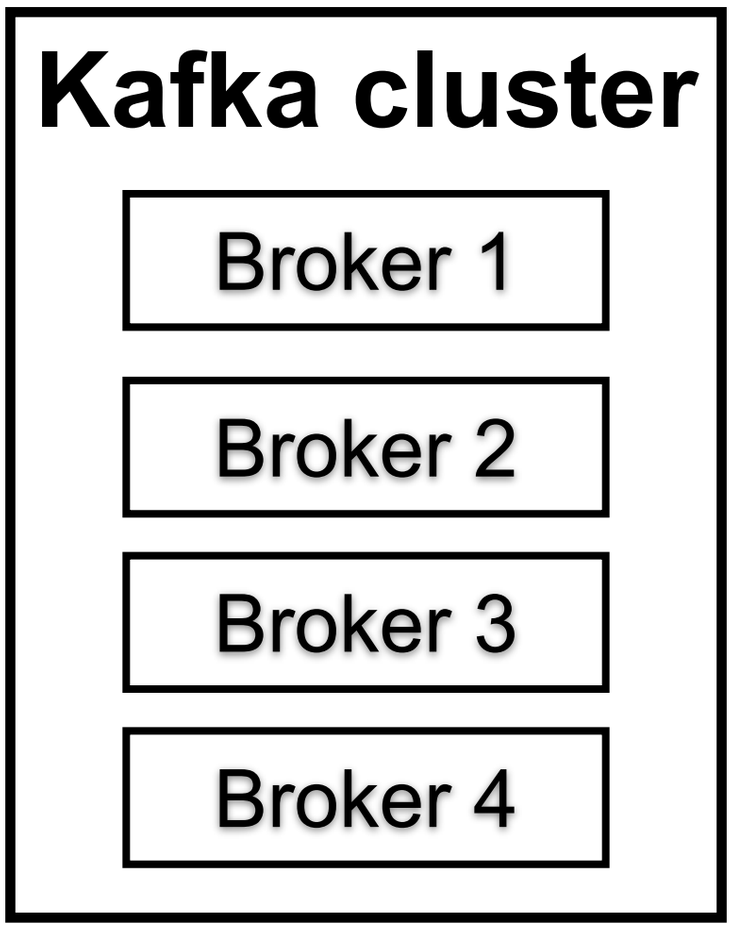
Leader for write
Follows for read

Zookeeper architecture



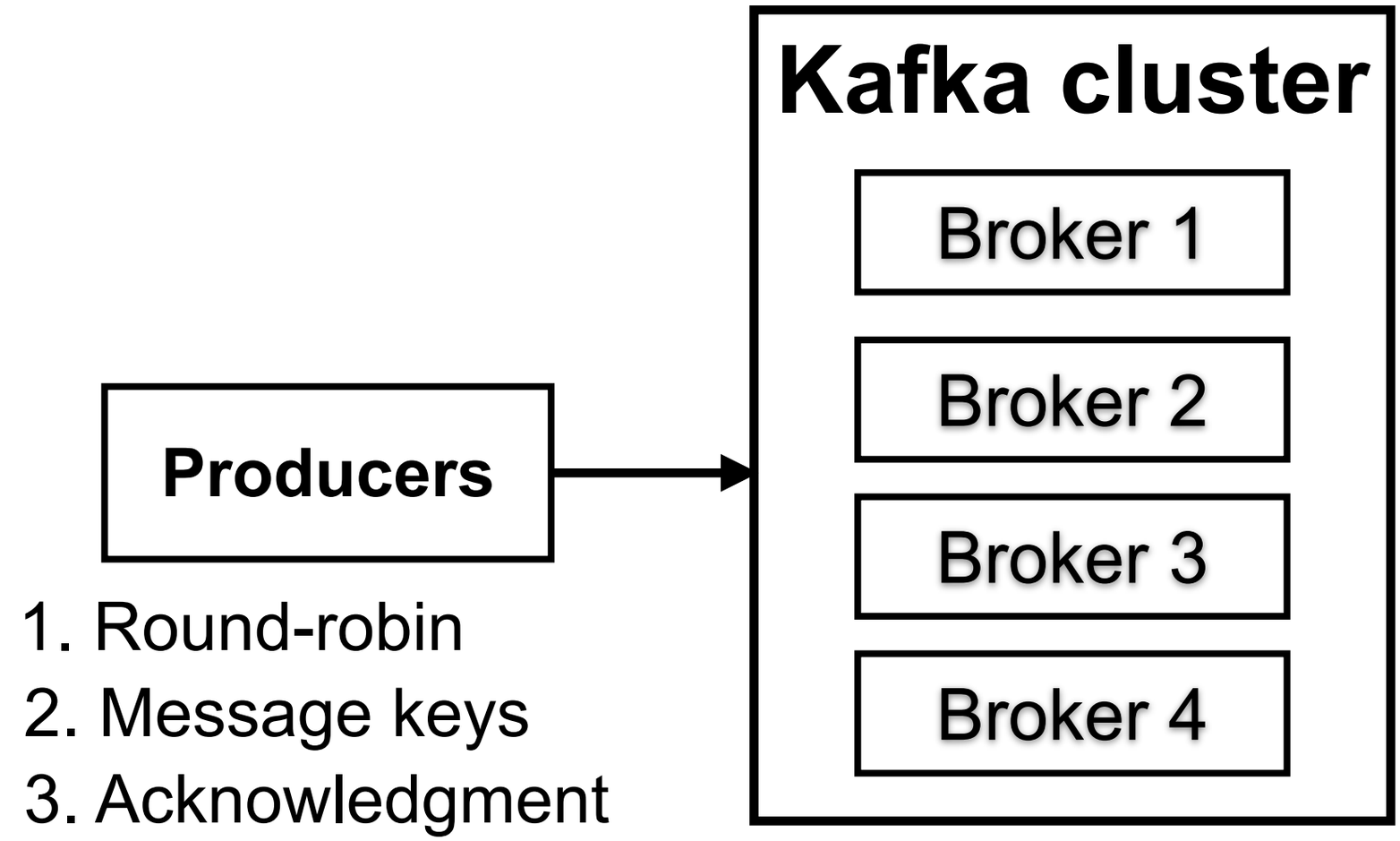
Summary of Kafka

Kafka concepts



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

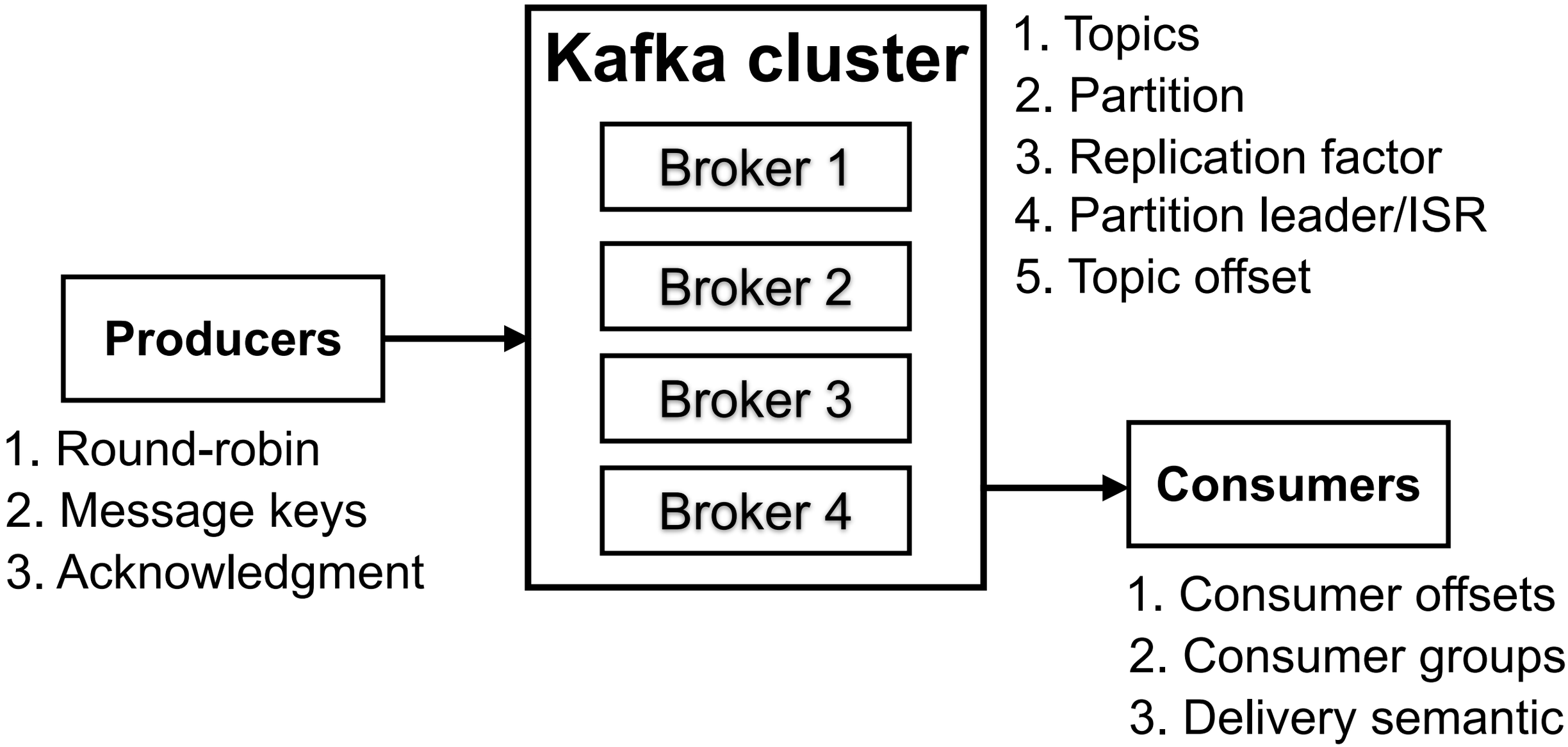
Kafka concepts



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

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

Kafka concepts

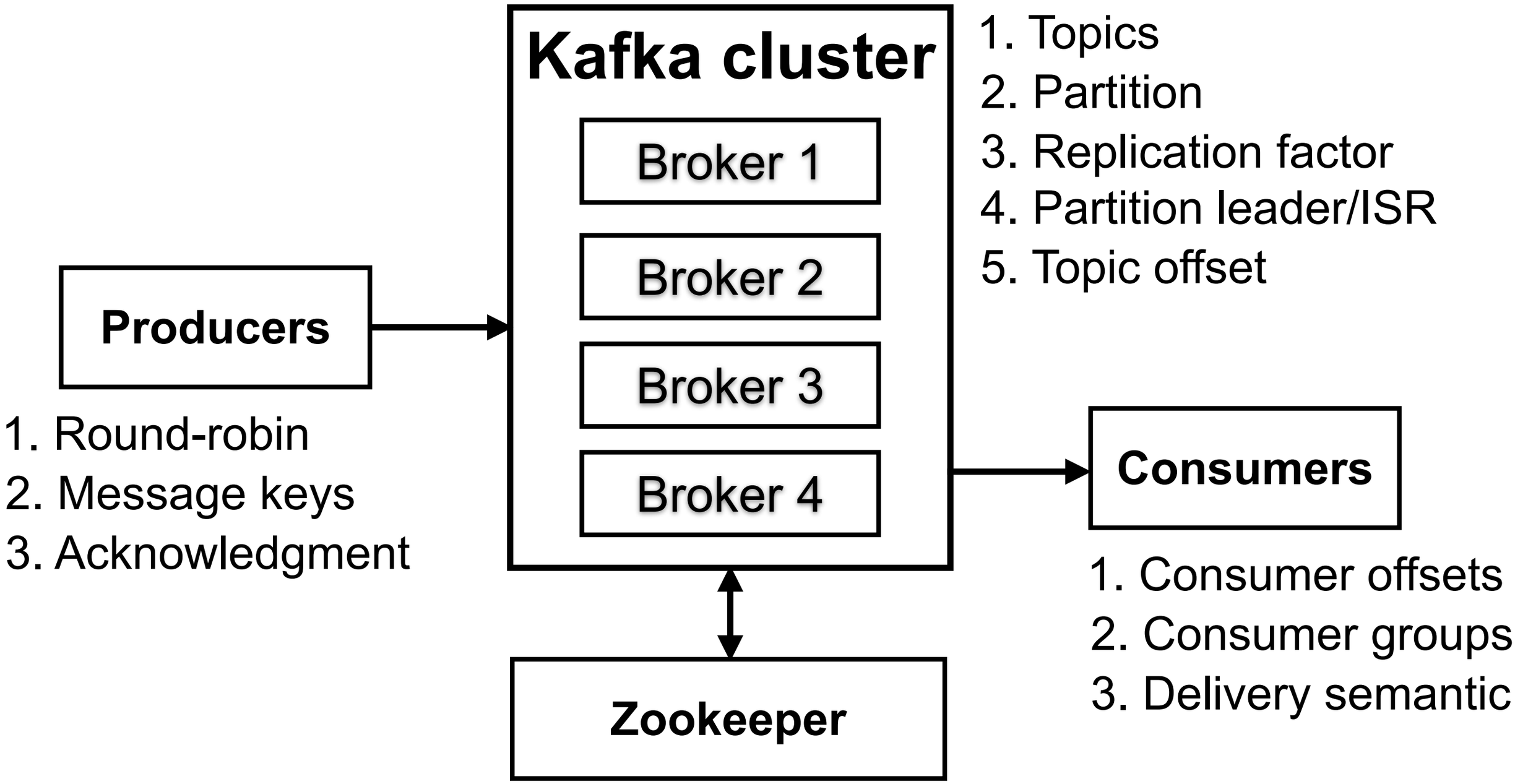


1. Topics
2. Partition
3. Replication factor
4. Partition leader/ISR
5. Topic offset

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

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

Kafka concepts



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

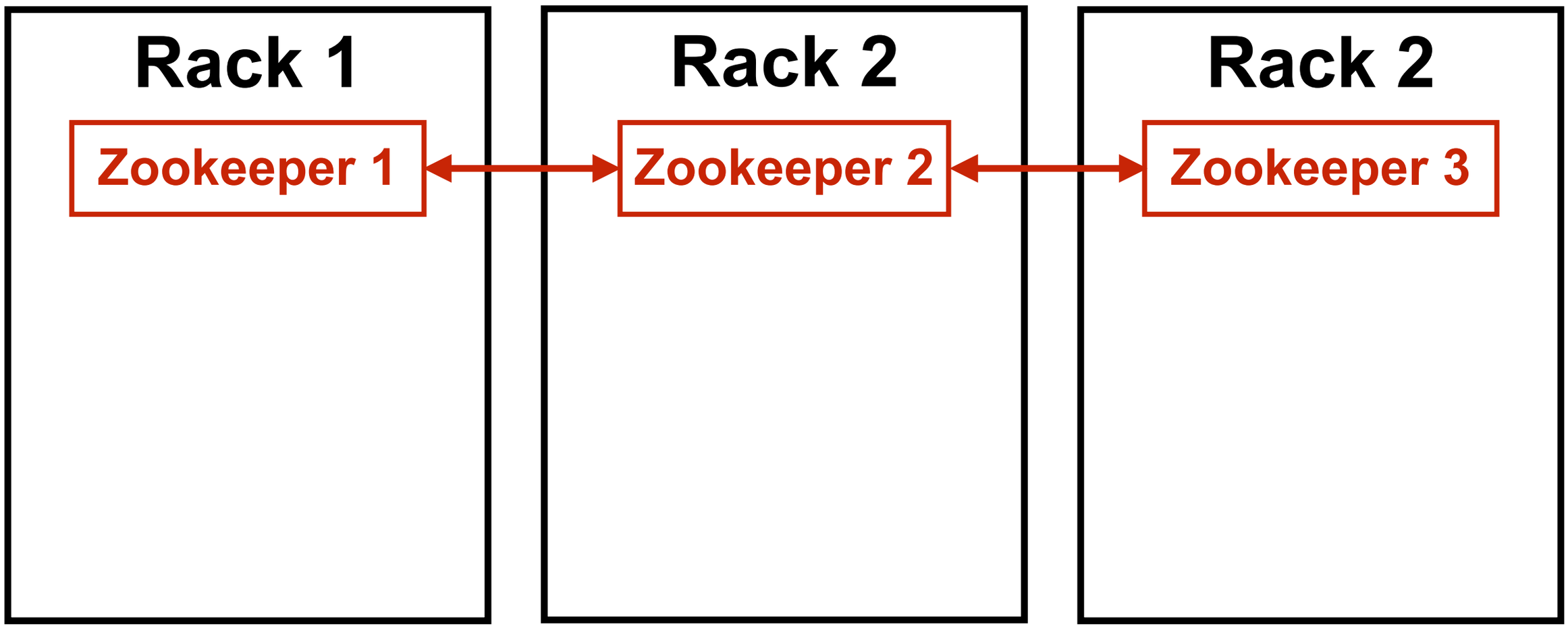
1. Round-robin
2. Message keys
3. Acknowledgment

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

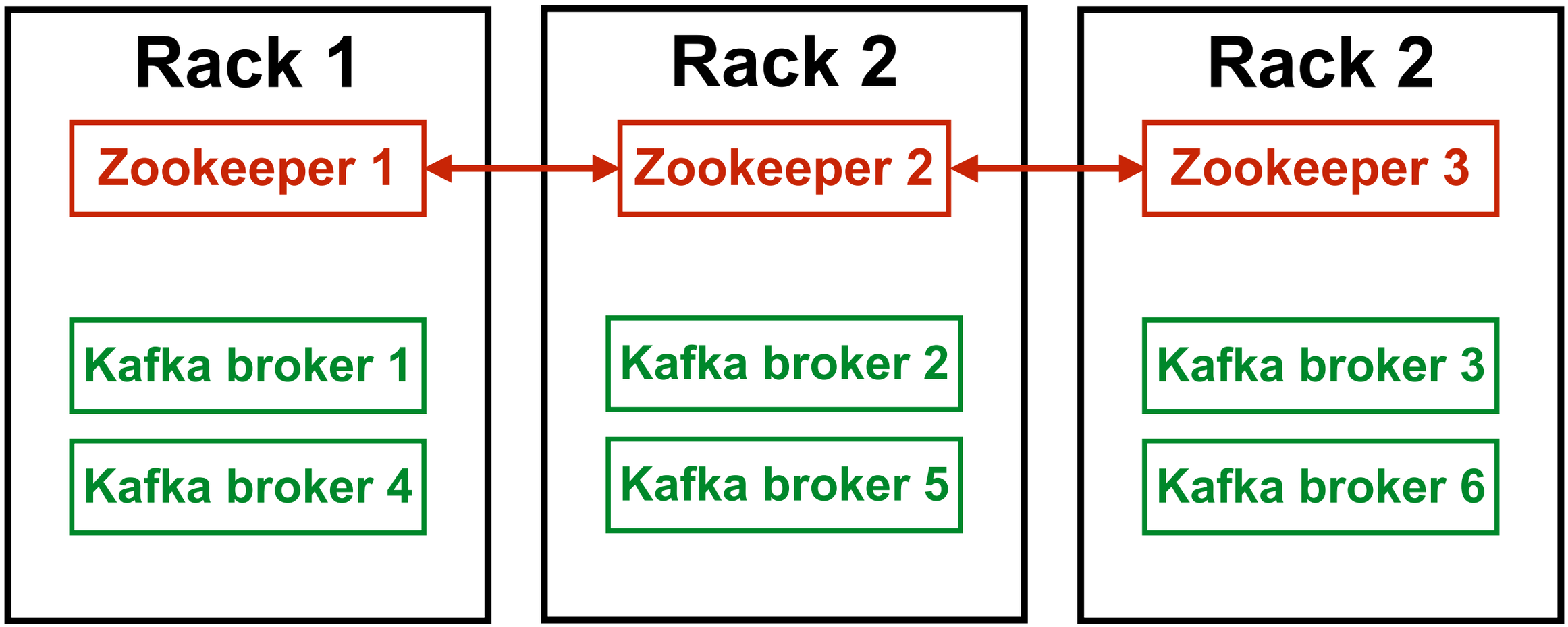
Kafka workshop

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

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

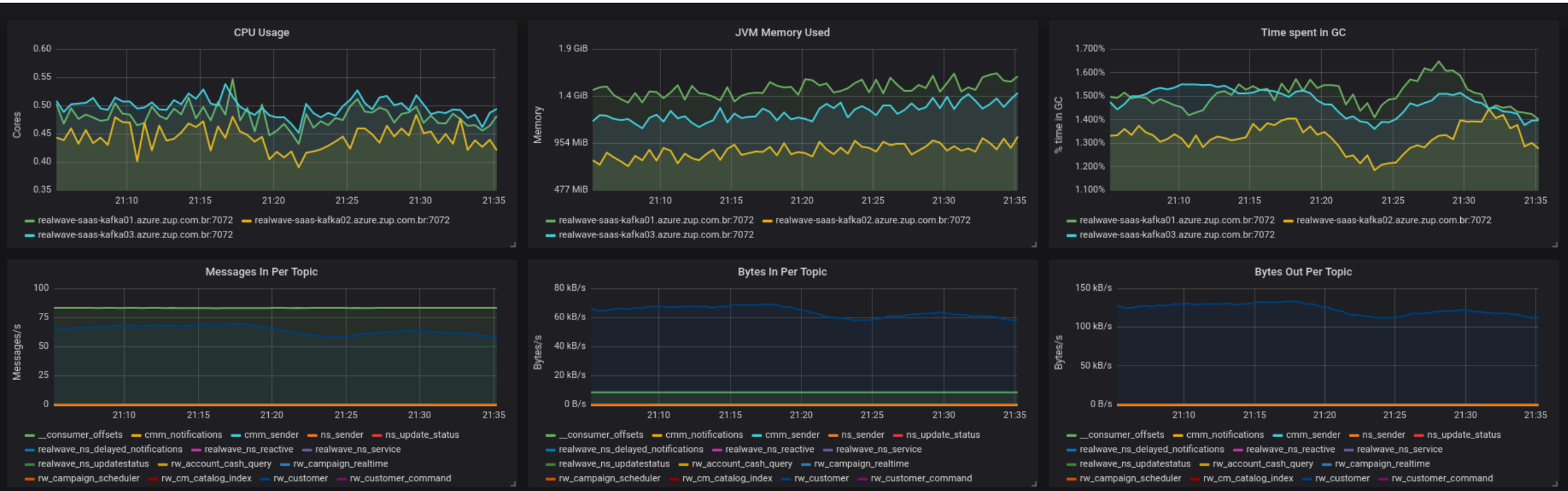
https://docs.confluent.io/current/kafka/monitoring.html

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

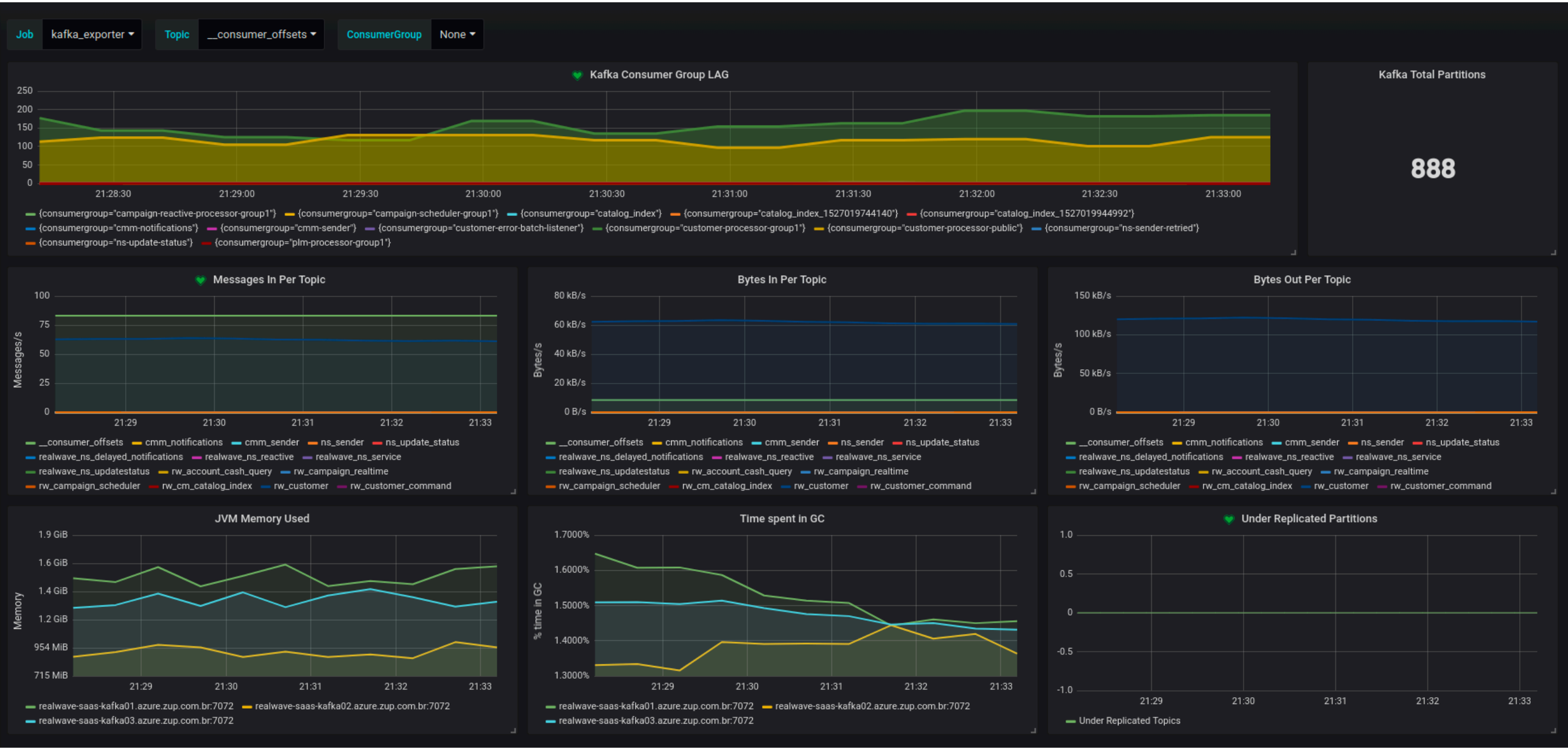
Tips #2

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 !!

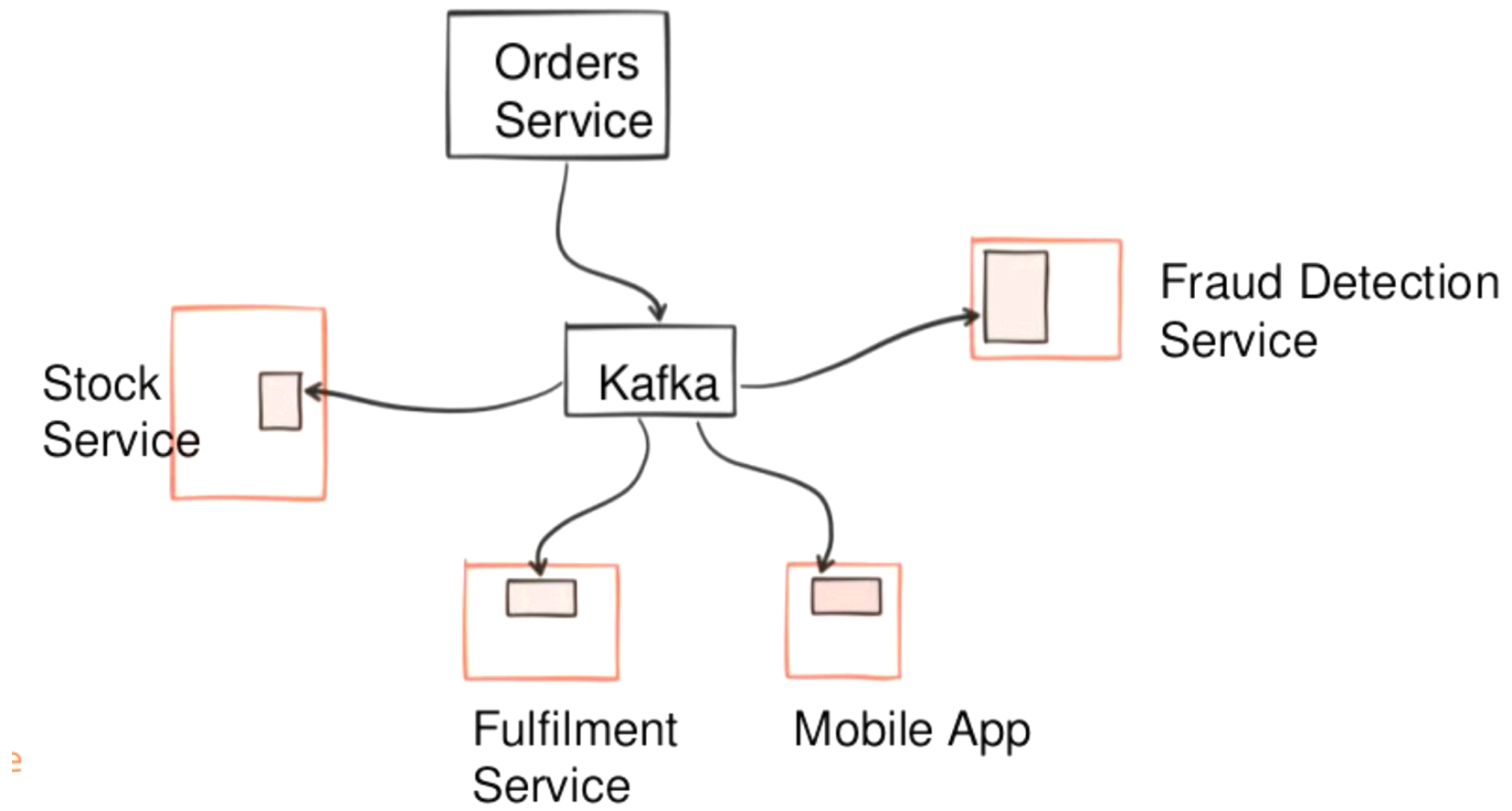
Capacity planning

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

Tips #4

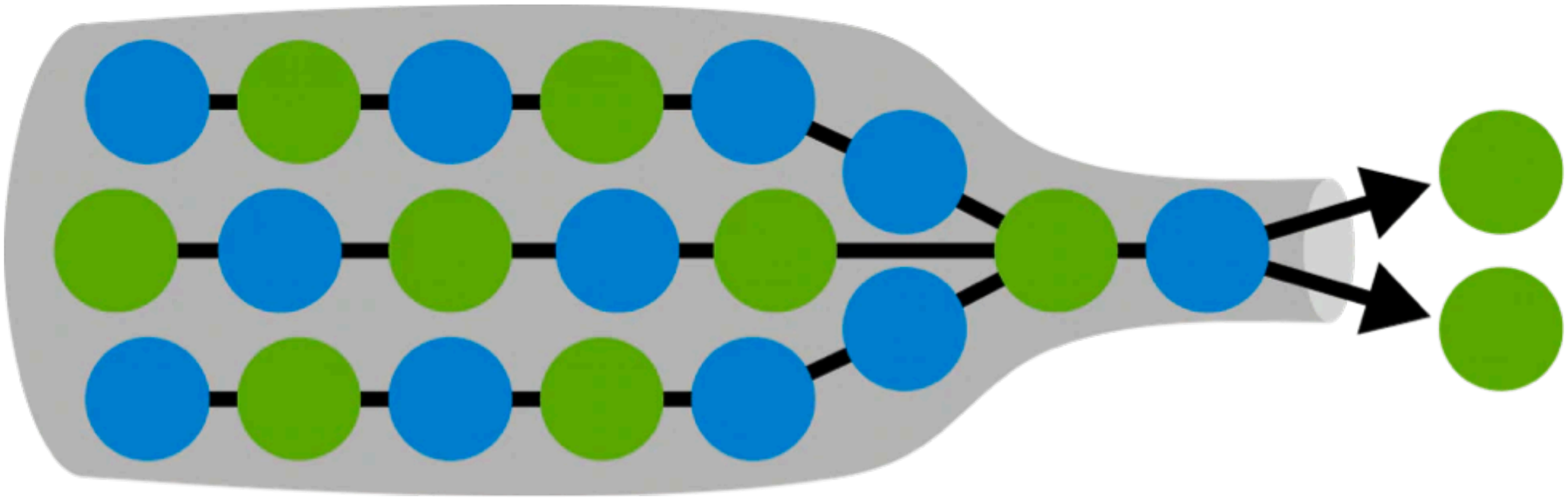
Monitoring brokers isn't enough

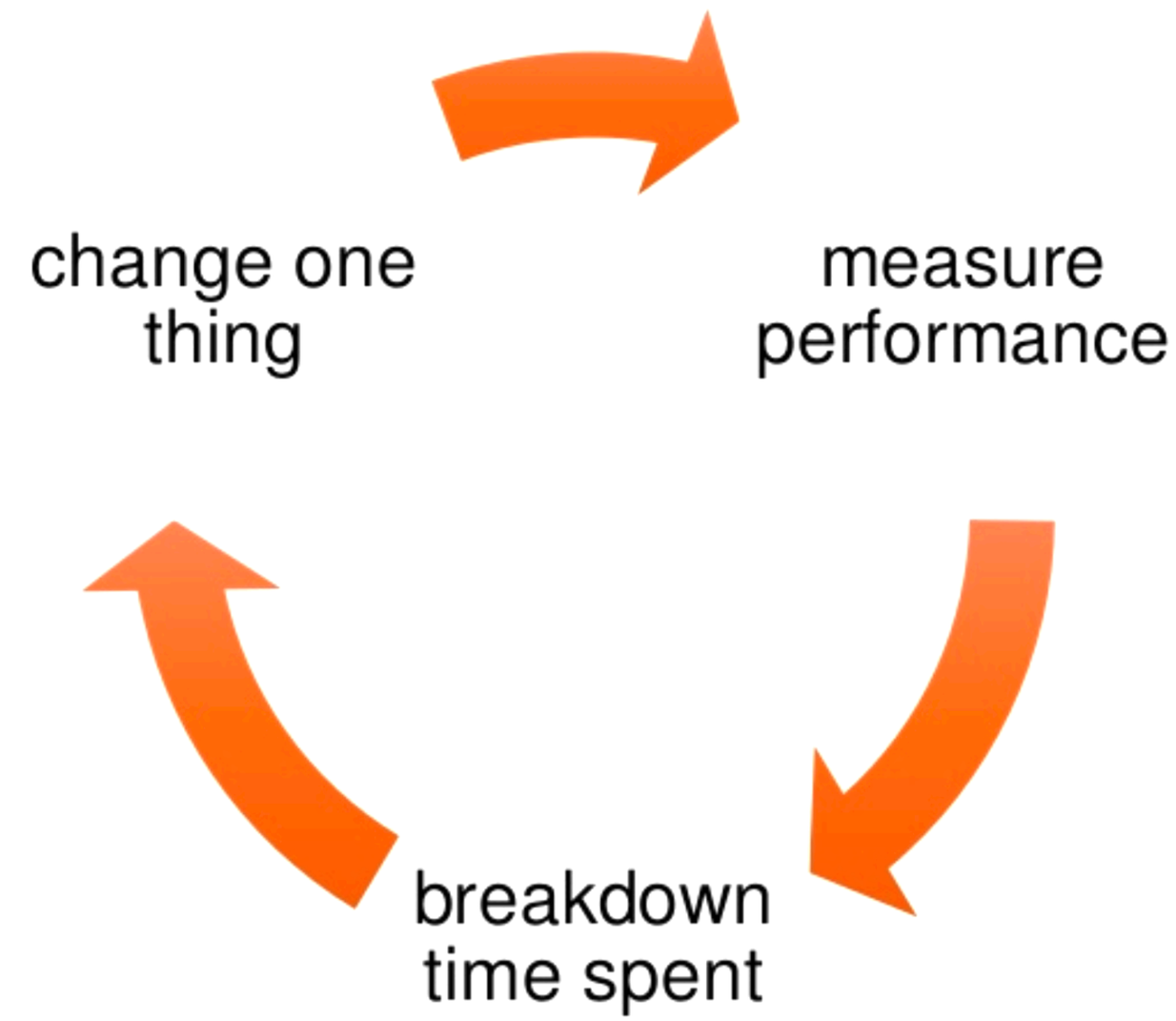
Need to monitor events



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