آشنایی ، استفاده و مدیریت آپاچی کافکا

فهرست مطالب

فهرست مطالب

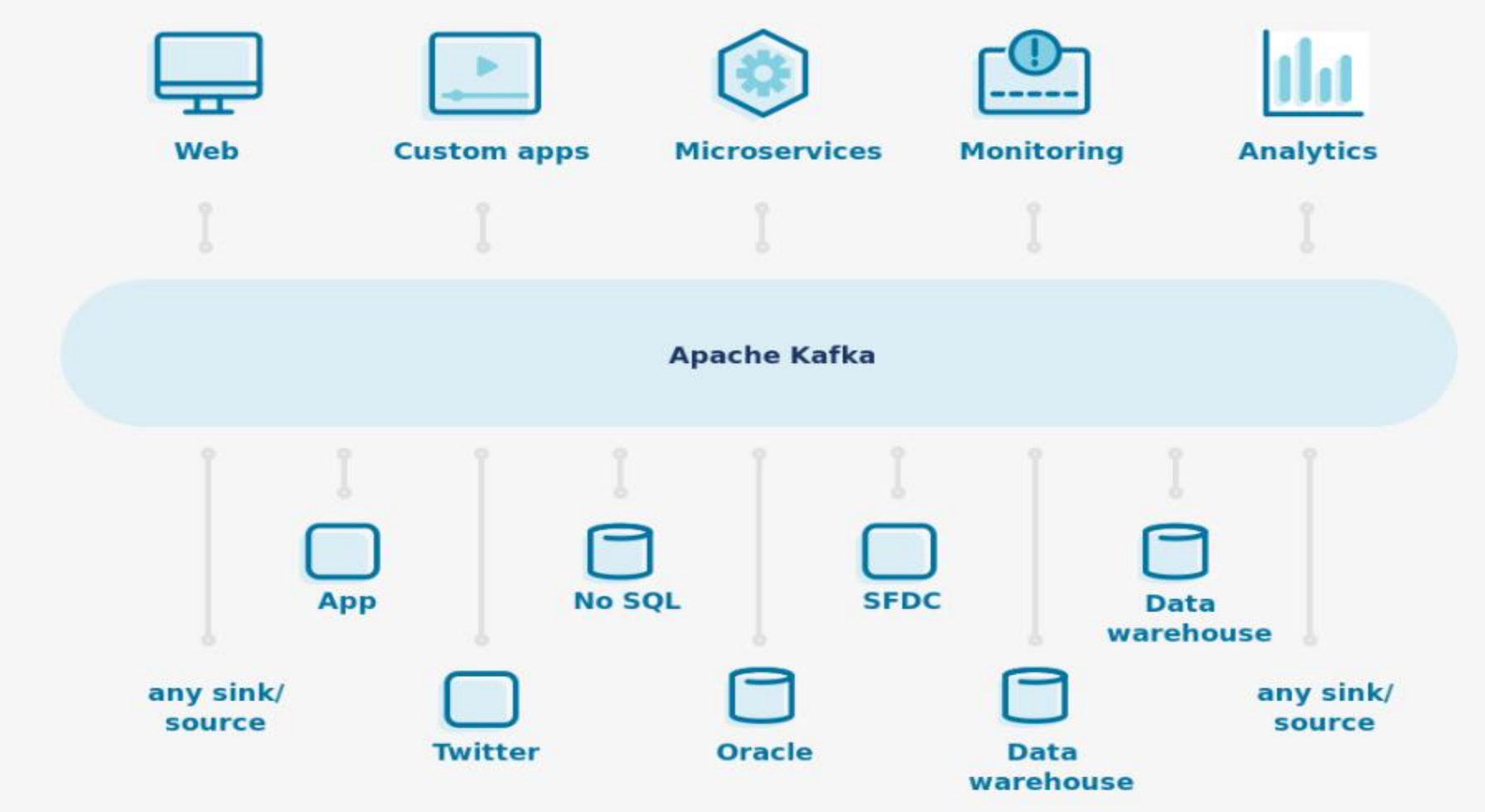
1 - Introduction - Architecture and Concept

Introduction

- Community
- Distributed event streaming platform
- Capable of handling trillions of events a day
- Based on an abstraction of a distributed commit log
- Created and open sourced by LinkedIn in 2011

Introduction

Where Apache Kafka fits in



Usage

Thousands of companies are built on Kafka

How can Kafka help you

Publish + Subscribe

Store

Process

Architecture

How does Kafka work?

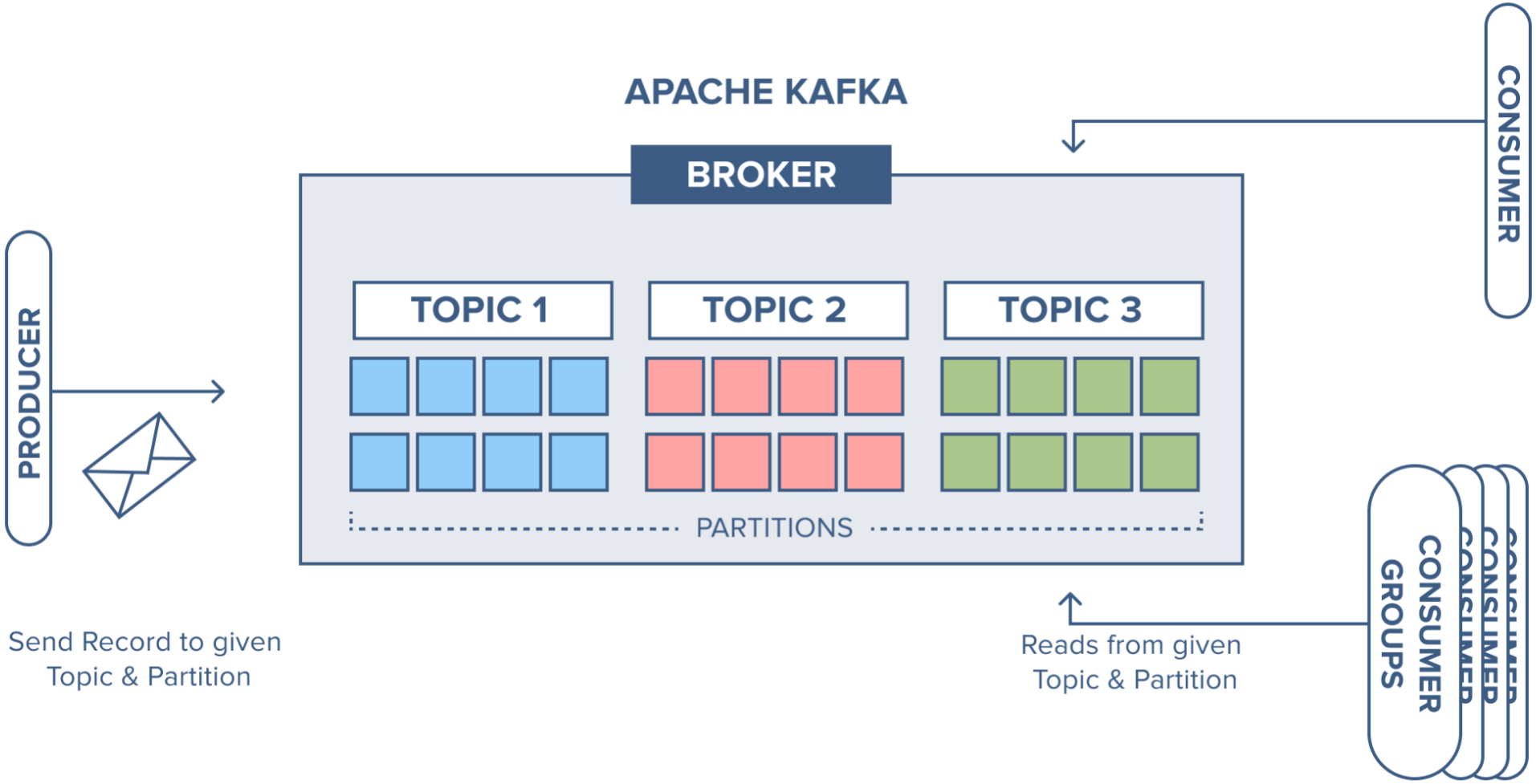
- Kafka is a distributed system
- Consisting of servers and clients
- Communicate via a high-performance TCP

Kafka Client

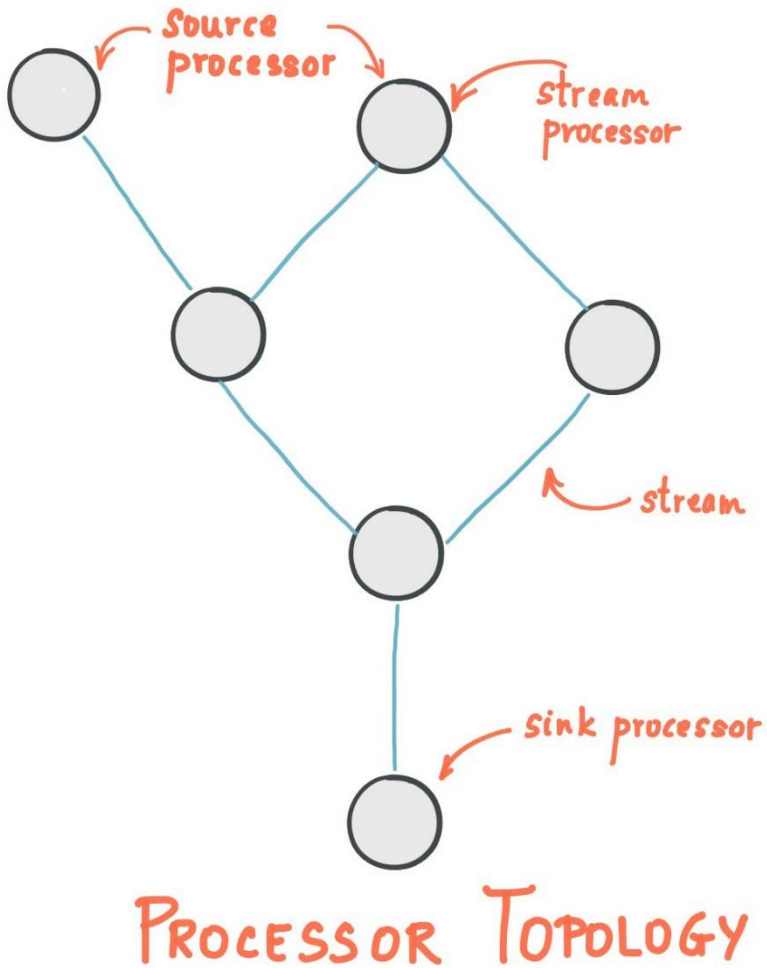
- Java
- Scala
- C/C++
- Python
- Go (AKA golang)
- Erlang
- .NET
- Clojure
- Ruby
- Node.js
- Proxy(Http Rest)
- Perl
- Rust
- Swift
- PHP

2 - Primary Elements

Main Concepts & Terminology

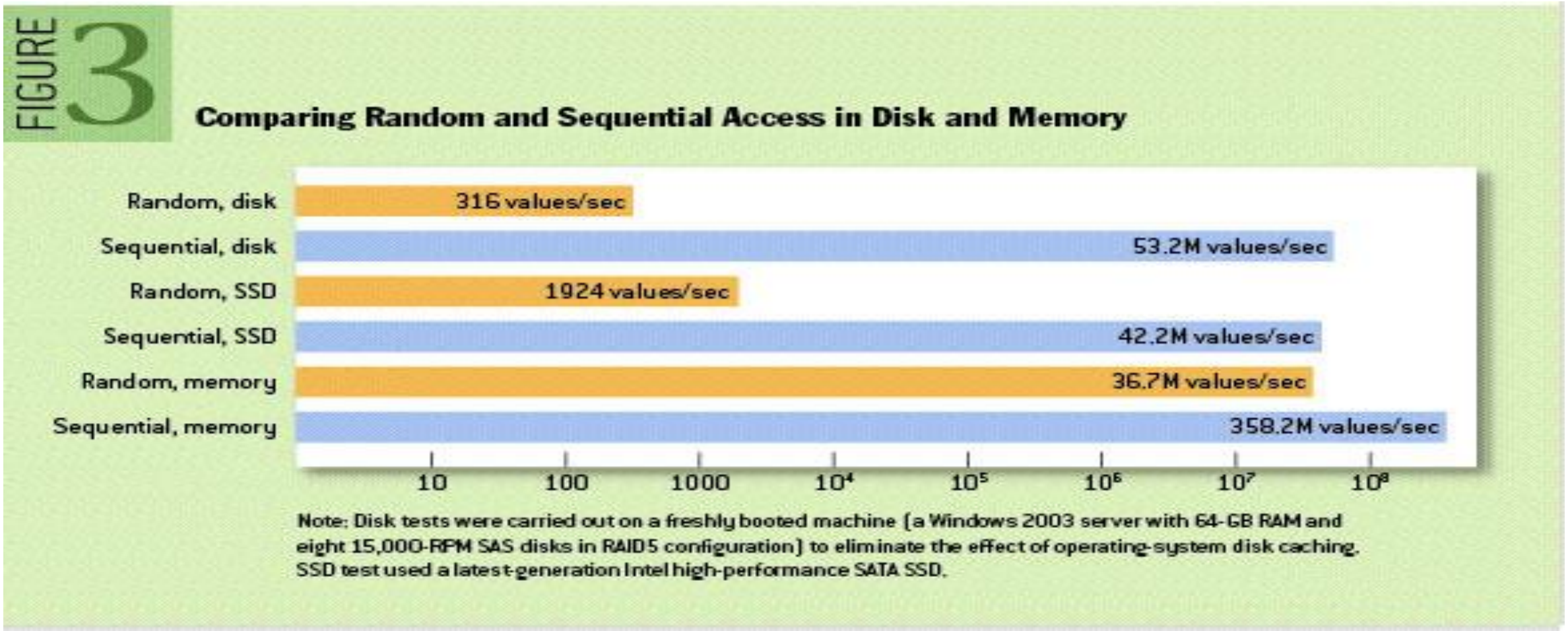


Stream Processing Topology



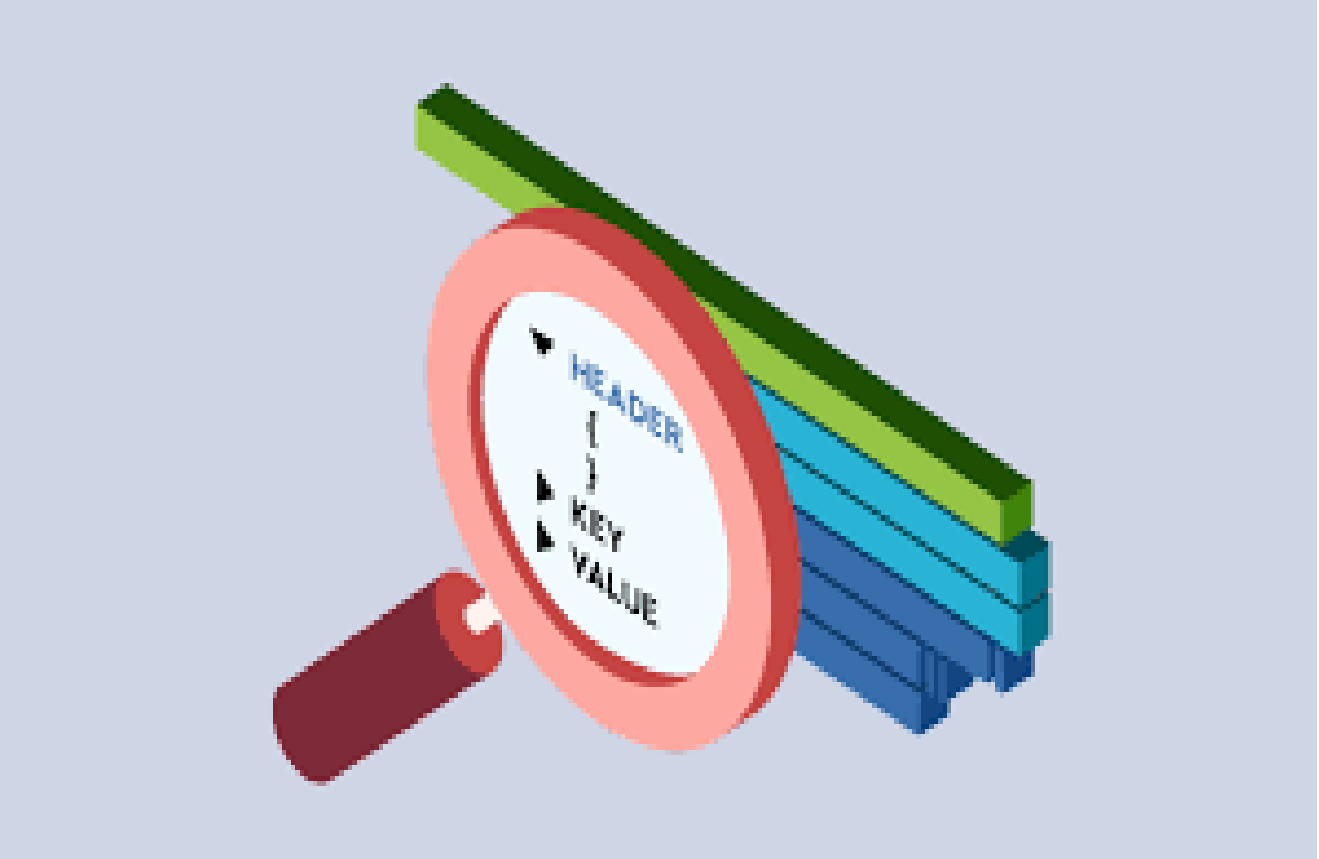
Persistence

Don’t fear the file system

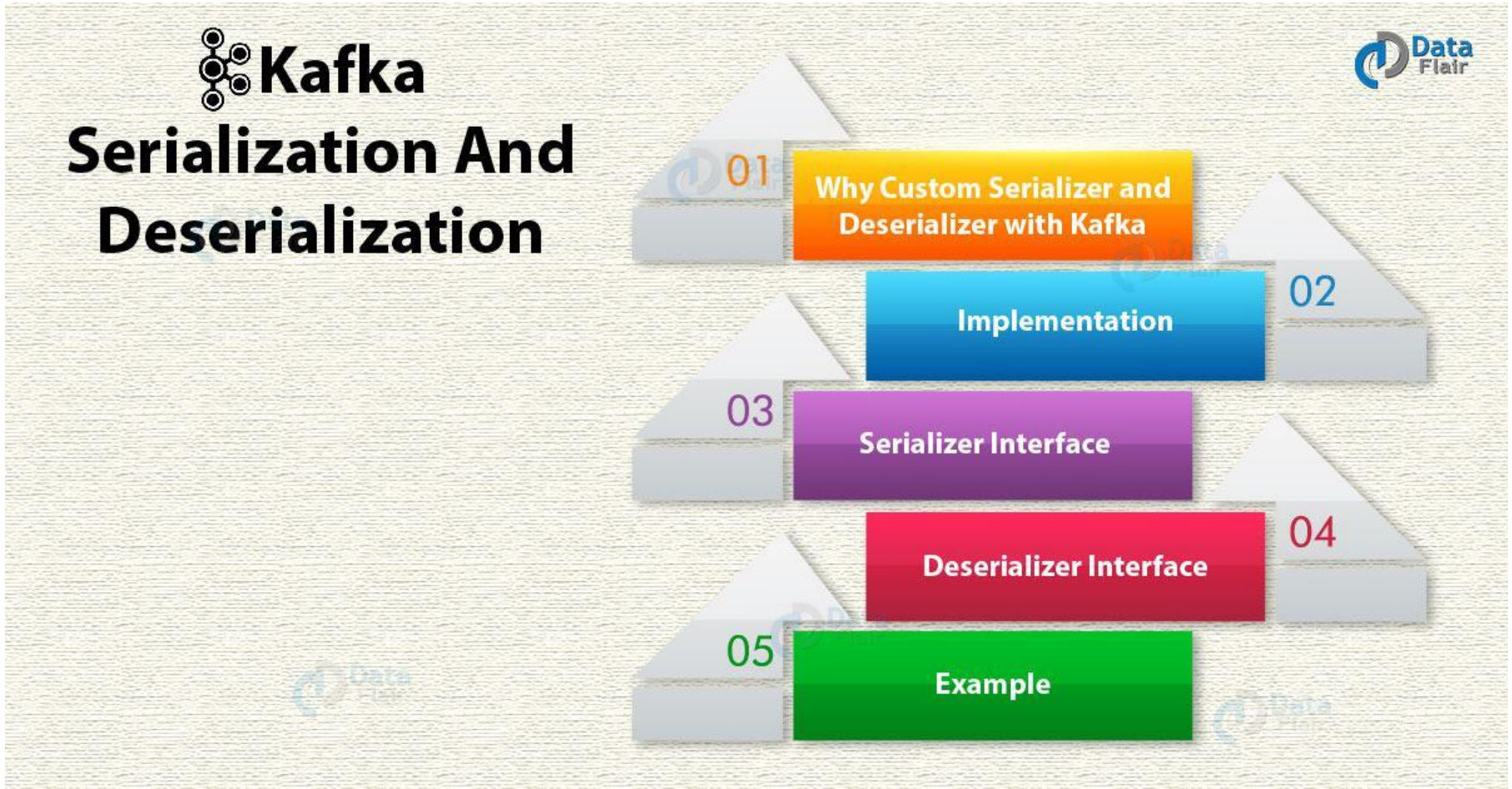


3 - Messages – Metadata- Schema

Use the power of record headers



Message Schema



Message Serializer & Deserializer

Serializer Interface

```
public interface Serializer extends Closeable {

    void configure(Map<String, ?> var1, boolean var2);

    byte[] serialize(String var1, T var2);

    void close();

}
```

Deserializer Interface

```
public interface Deserializer extends Closeable {

    void configure(Map<String, ?> var1, boolean var2);

    T deserialize(String var1, byte[] var2);

    void close();

}
```

Example

```
public class User {
    private String name;
    private int age;

    // Constructor

    // Getter & Setter
}
```

Example

```
public class UserSerializer implements Serializer {

	@Override public void configure(Map<String, ?> map, boolean b) {
	}

	@Override public byte[] serialize(String arg0, User arg1) {
		return new ObjectMapper().writeValueAsString(arg1).getBytes();
	}

	@Override public void close() {
	}

}
```

Example

```
public class UserDeserializer implements Deserializer {

	@Override public void close() {
	}

	@Override public void configure(Map<String, ?> arg0, boolean arg1) {
	}

	@Override public User deserialize(String arg0, byte[] arg1) {
		return new ObjectMapper().readValue(arg1, User.class);
	}

}
```

Producer and Consumer Configuration

```
// Producer side
props.put("value.serializer",
          “com.isc.nps.npsd.kafka.serializers.UserSerializer");
```

```
//Consumer side
props.put("value.deserializer",
          "com.isc.nps.npsd.kafka.deserializer.UserDeserializer");
```

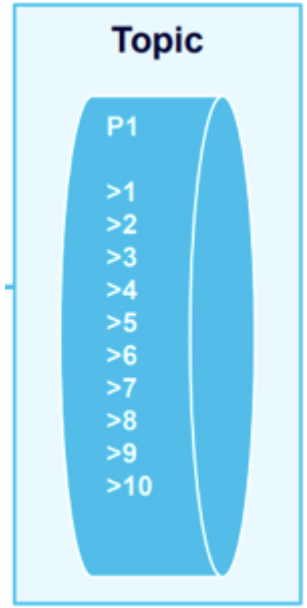
4 - Messages - Order and Delivery

Message Order - Single Partitioning

```
./kafka-topics.sh --create \
 --zookeeper localhost:2181/kafka \
 --replication-factor 1 --partitions 1 \
 --topic my-topic
```

```
./kafka-console-producer.sh \
 --broker-list localhost:9092 \
 --topic my-topic
```

```
> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10
```

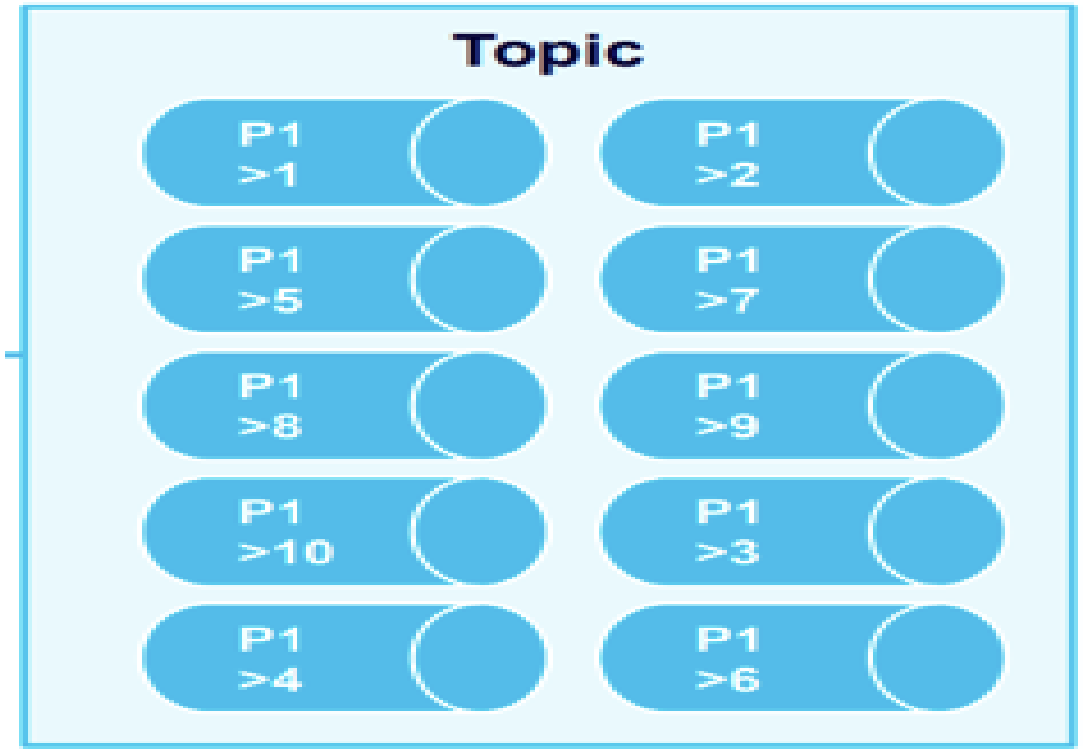


Message Order - Multiple Partitioning

```
./kafka-topics.sh --create \
 --zookeeper localhost:2181/kafka \
 --replication-factor 1 --partitions 10 \
 --topic my-topic
```

```
./kafka-console-producer.sh \
 --broker-list localhost:9092 \
 --topic my-topic
```

```
> 1
> 2
> 3
> 4
> 5
> 6
> 7
> 8
> 9
> 10
```



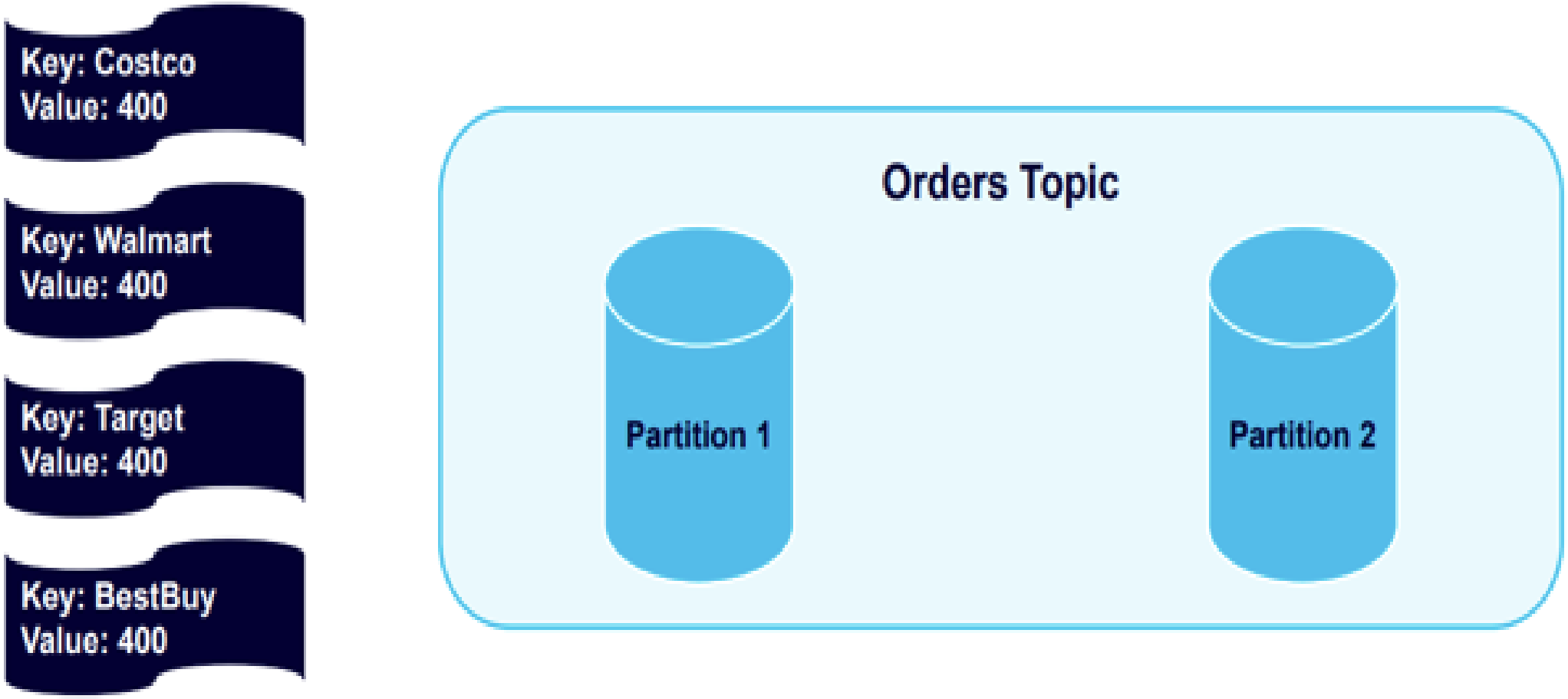

```
./kafka-console-consumer.sh \
 --bootstrap-server localhost:9092 \
 --topic my-topic \
 --from-beginning
```

```
> 2
> 5
> 9
> 3
> 4
> 7
> 6
> 10
> 8
> 1
```

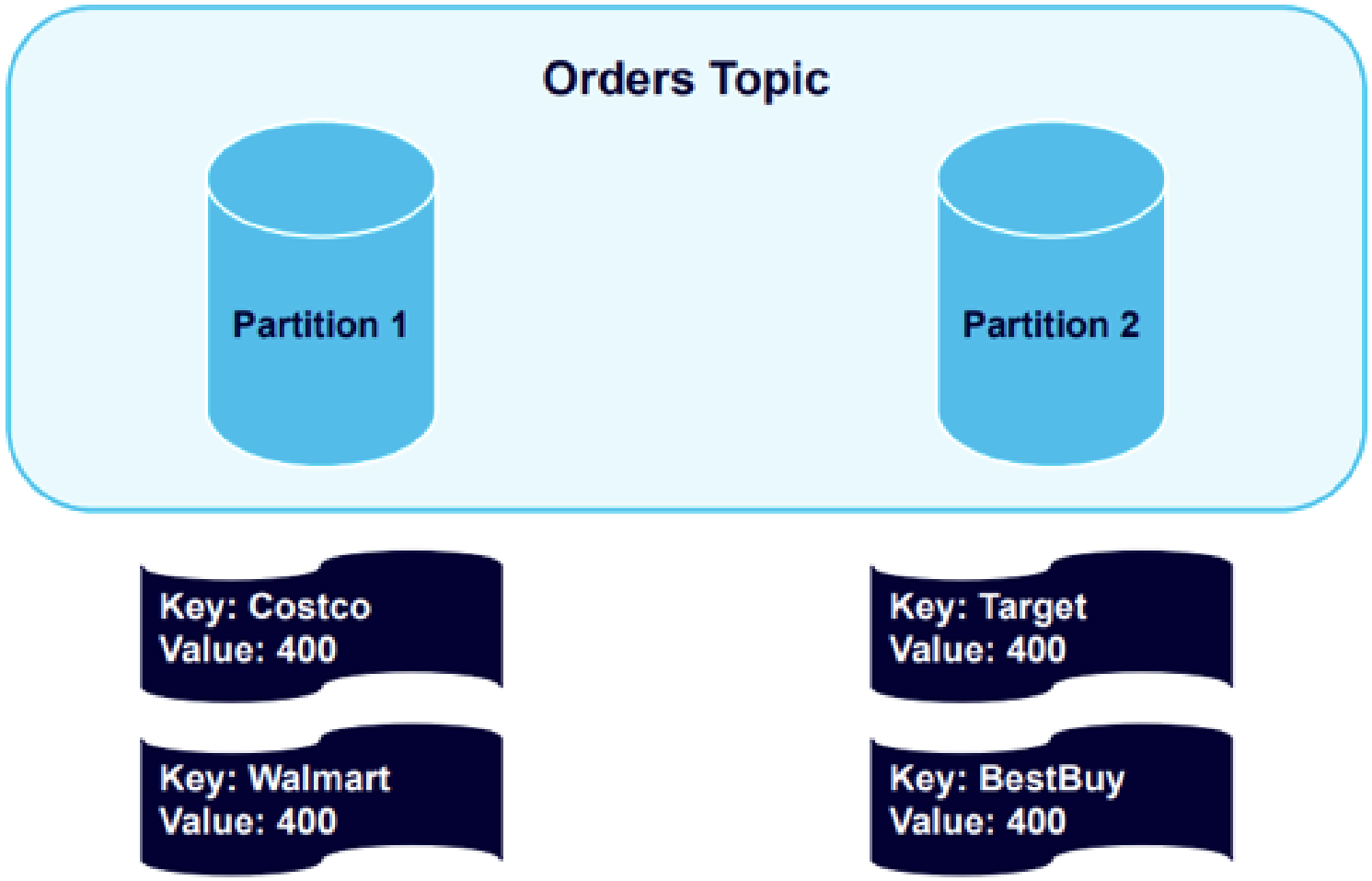
How to achieve strict ordering?

- Most businesses require things to be in the correct order
  - You must use only one partition
- Prefer messages to be ordered based on a certain property in the message
  - Use multiple partitions and still reach the result you wish

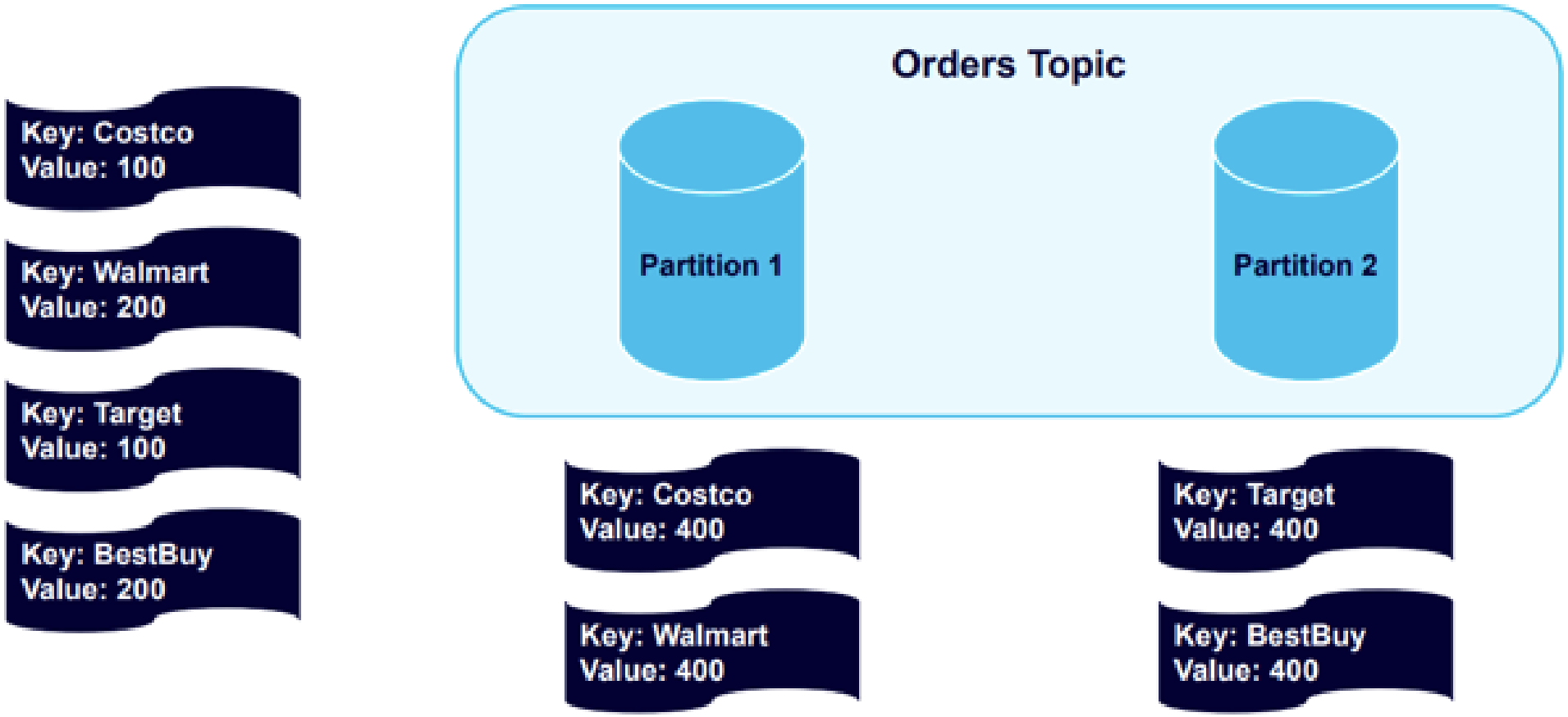
Message Order - Keying



Message Order - Keying

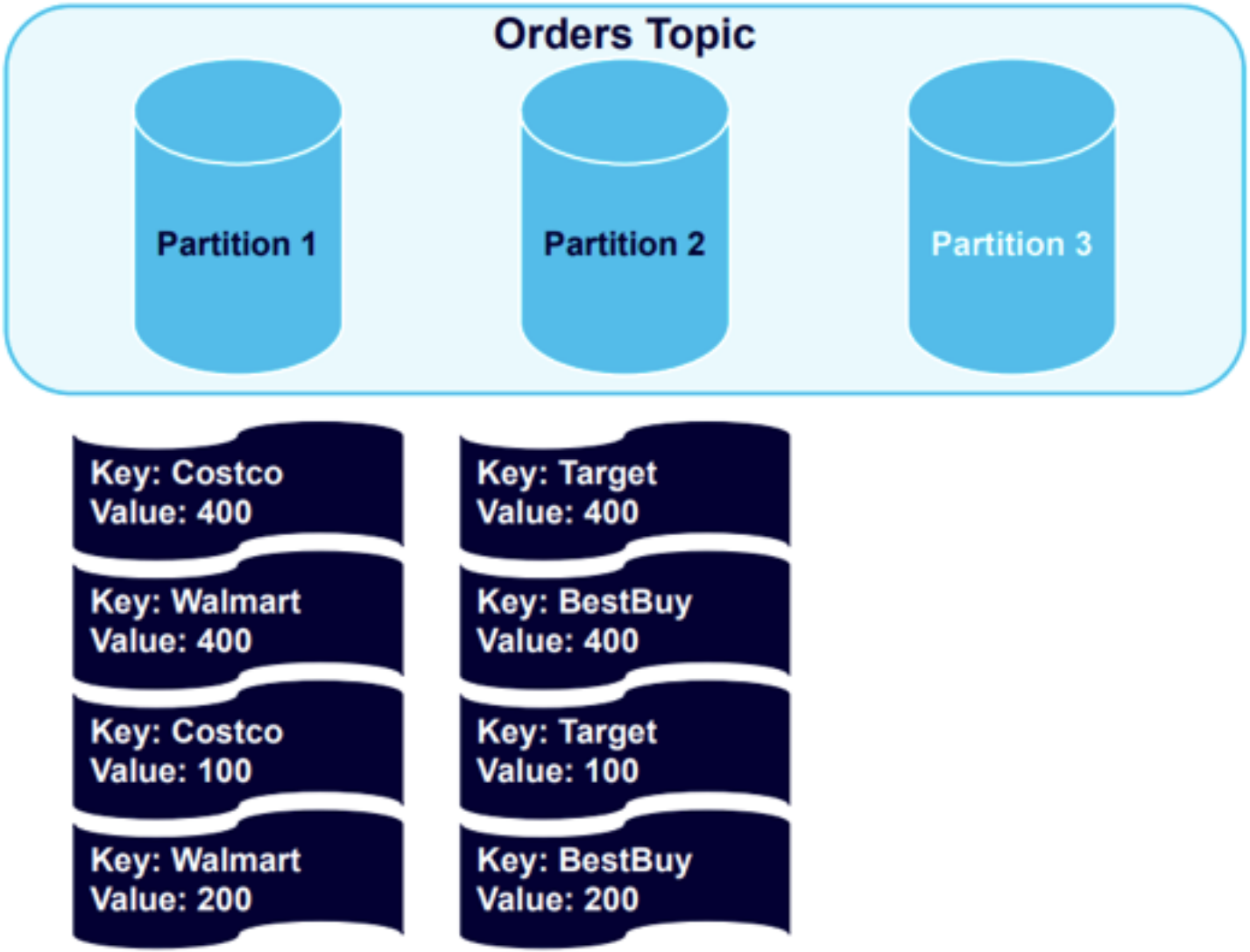


Message Order - Keying

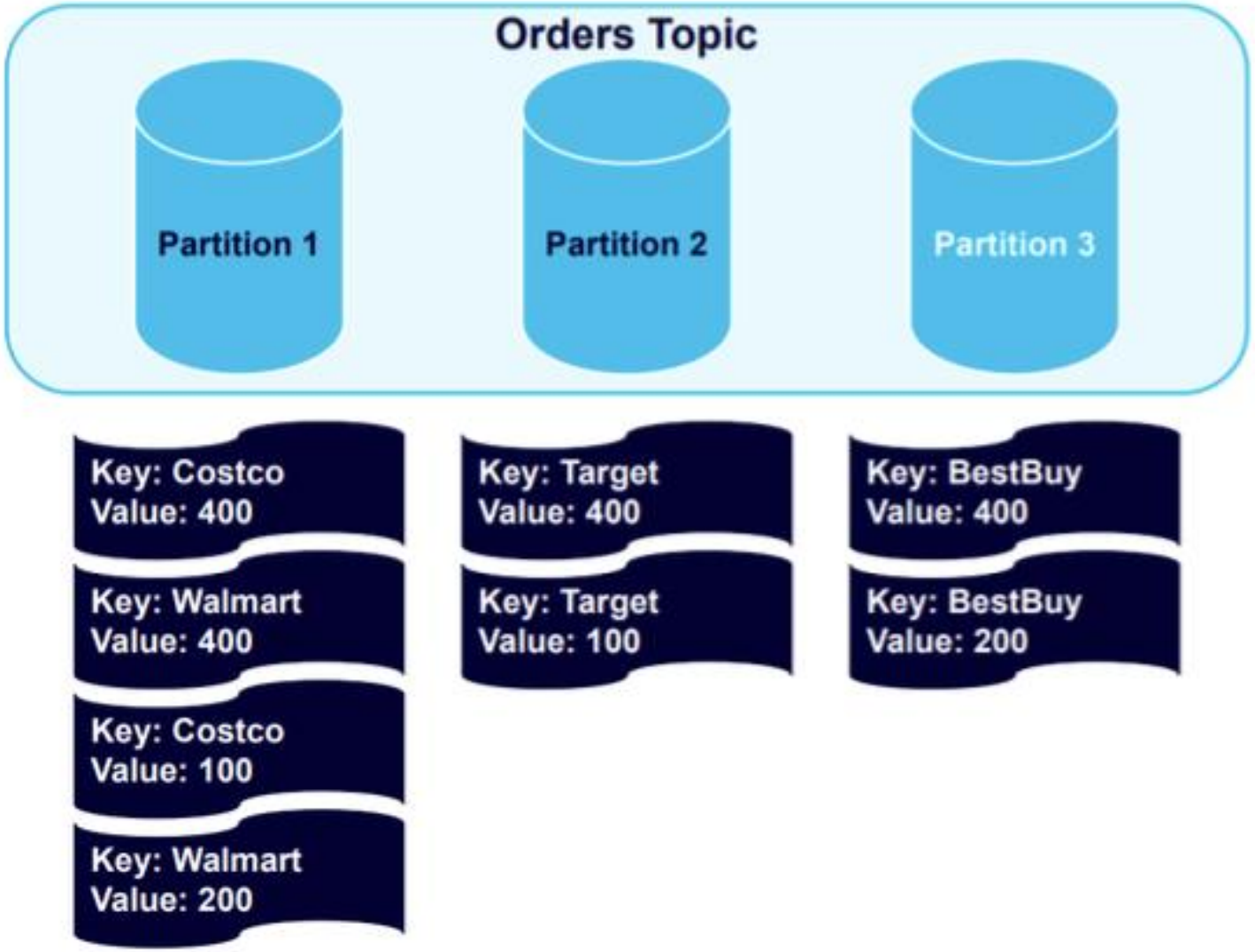


Message Order - Keying

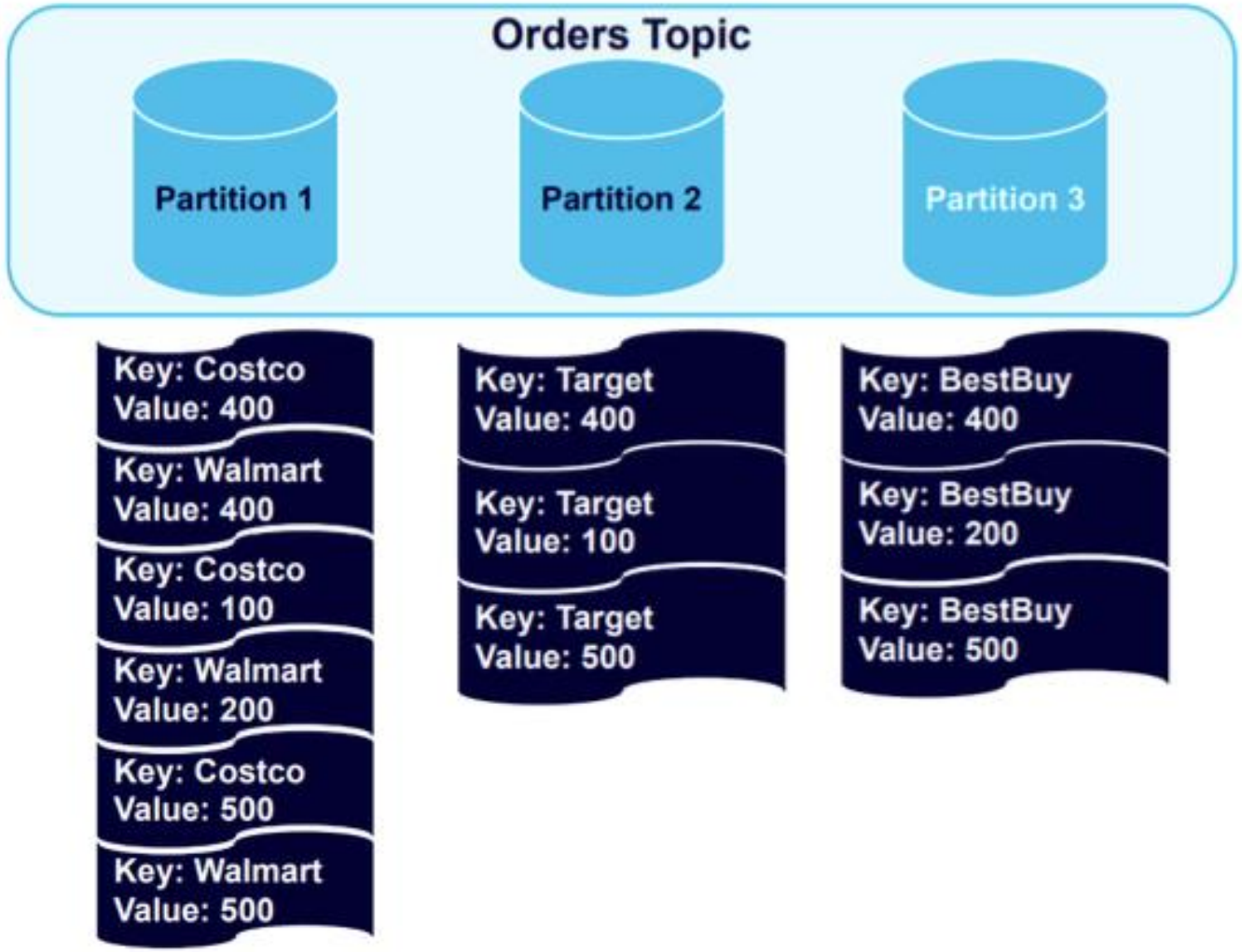
Message Order - Keying and Partition Rebalancing



Message Order - Keying and Partition Rebalancing

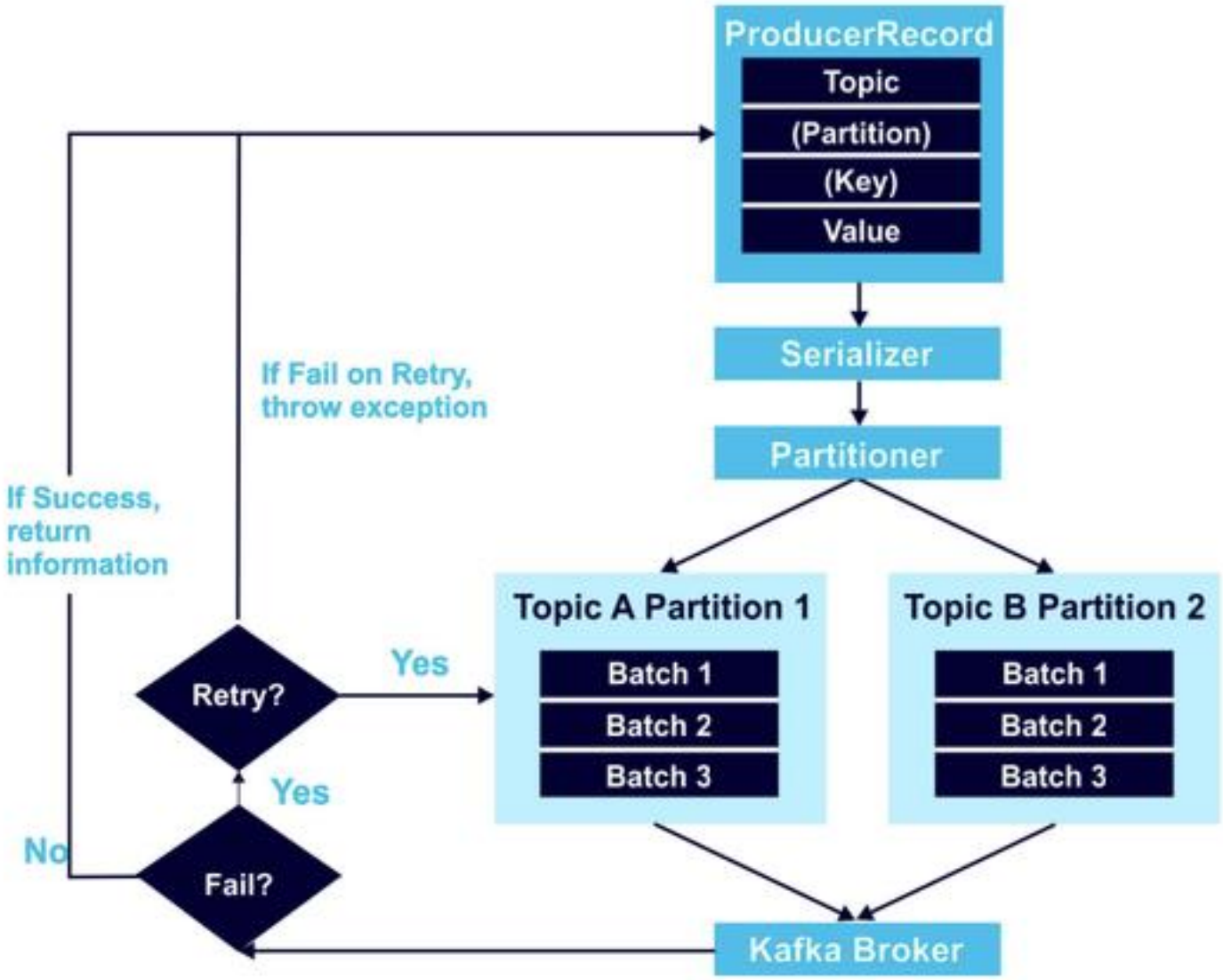


Message Order - Keying and Partition Rebalancing



Message Order - Keying and Partition Rebalancing

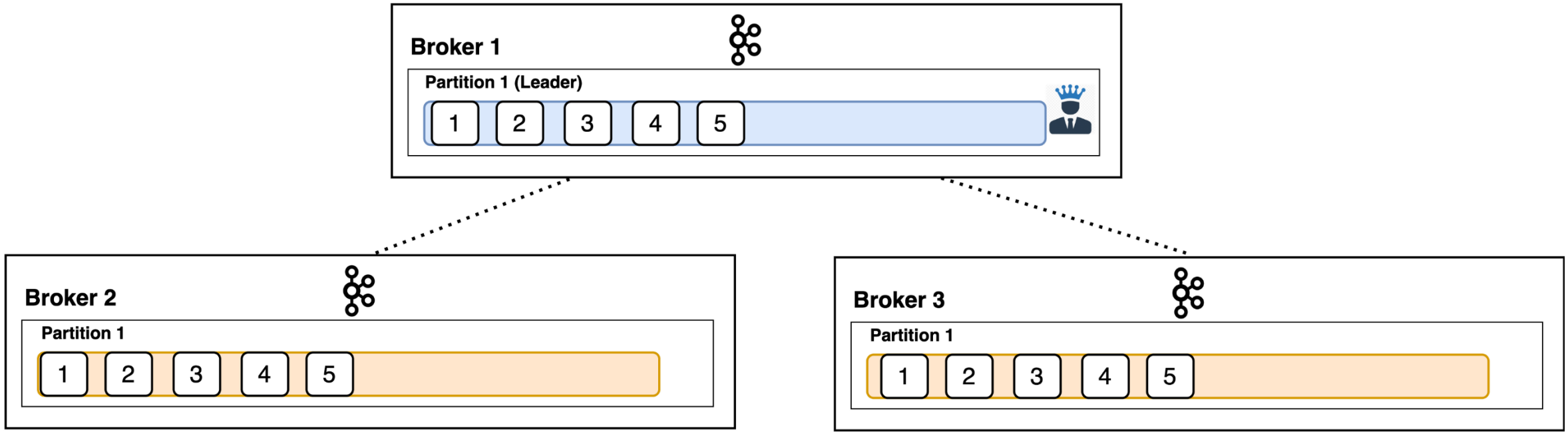
Message Order - Ensuring that Data is Always Sent in Order



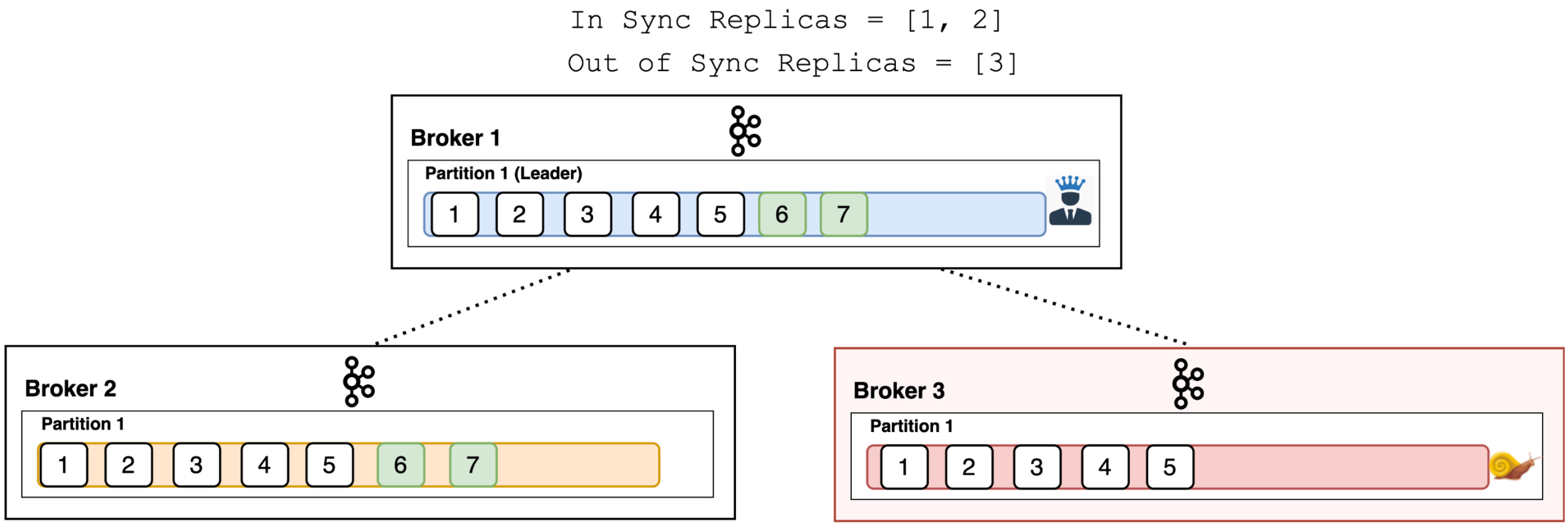
Message Delivery and Durability

- For Data Durability
  - Producer configuration setting “acks”
- “acks” specifies how many acknowledgement receives
  - To consider record / event delivered to the broker.
- “acks” options choose
  - none
  - One
  - all

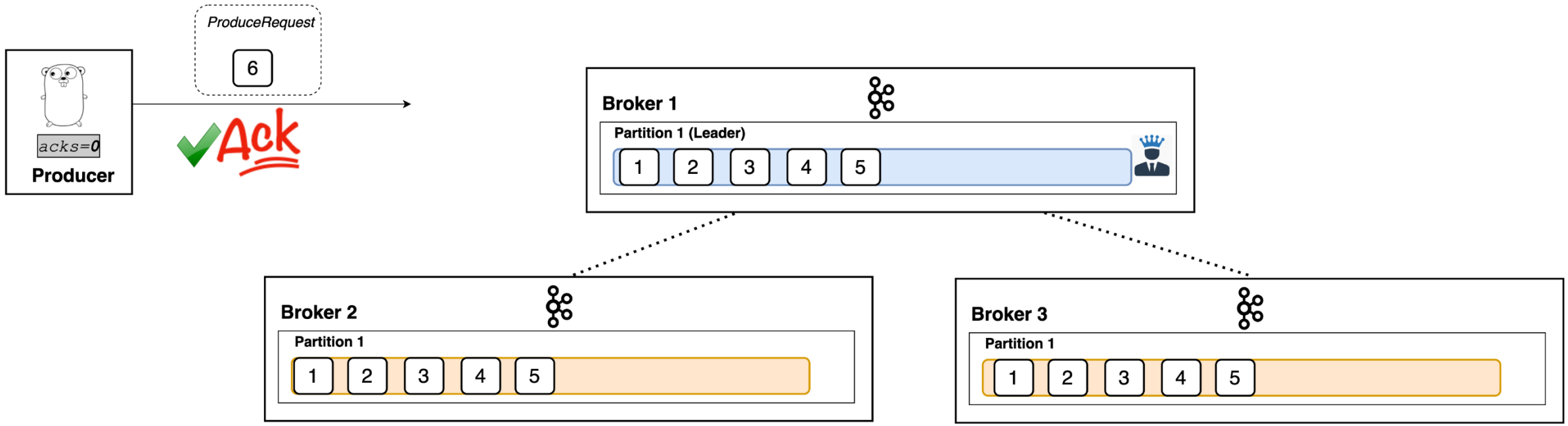
Message Delivery and Durability – acks - replication



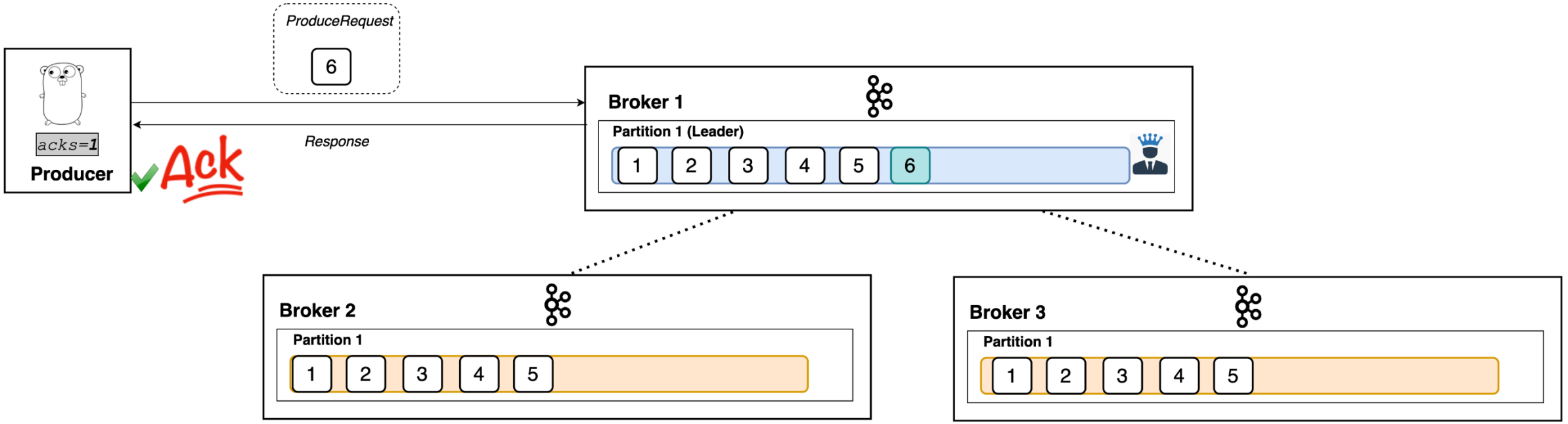
Message Delivery and Durability – acks – isr



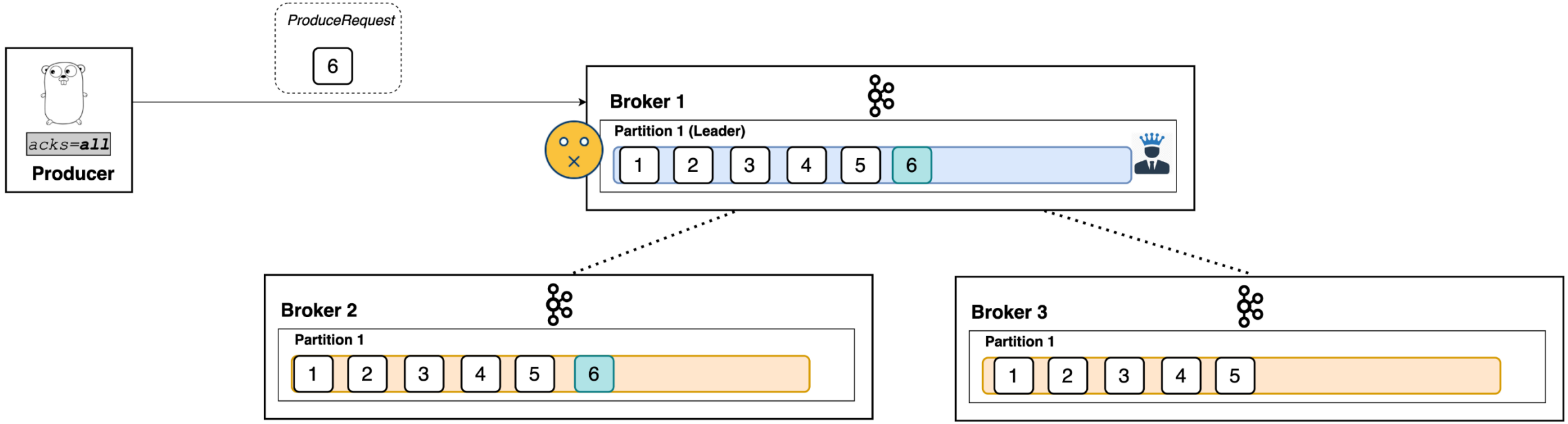
Message Delivery and Durability – acks = 0



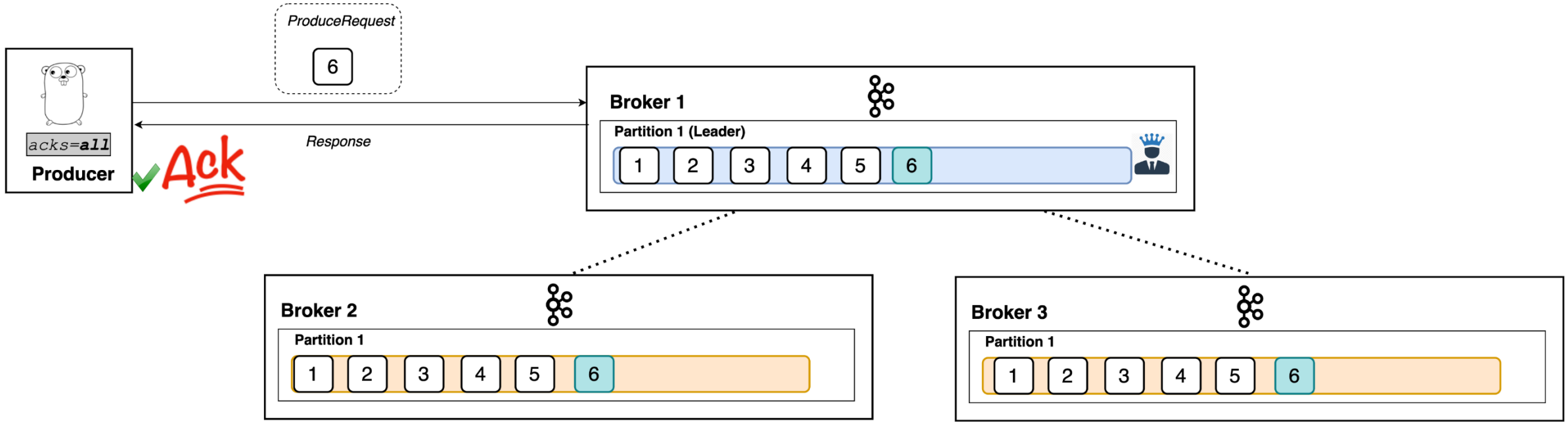
Message Delivery and Durability – acks = 1



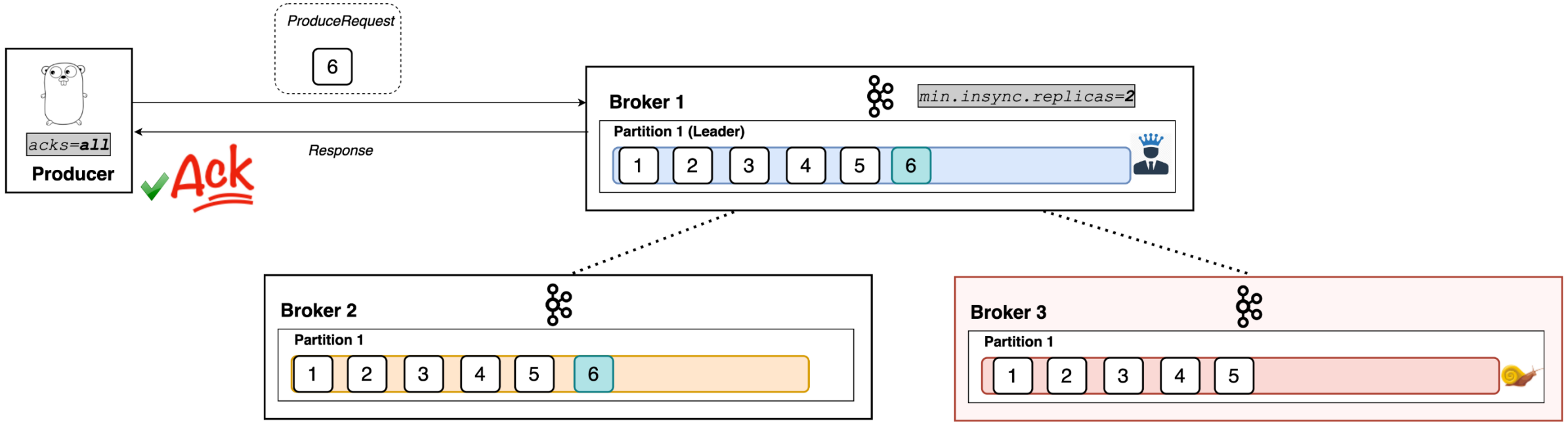
Message Delivery and Durability – acks = all



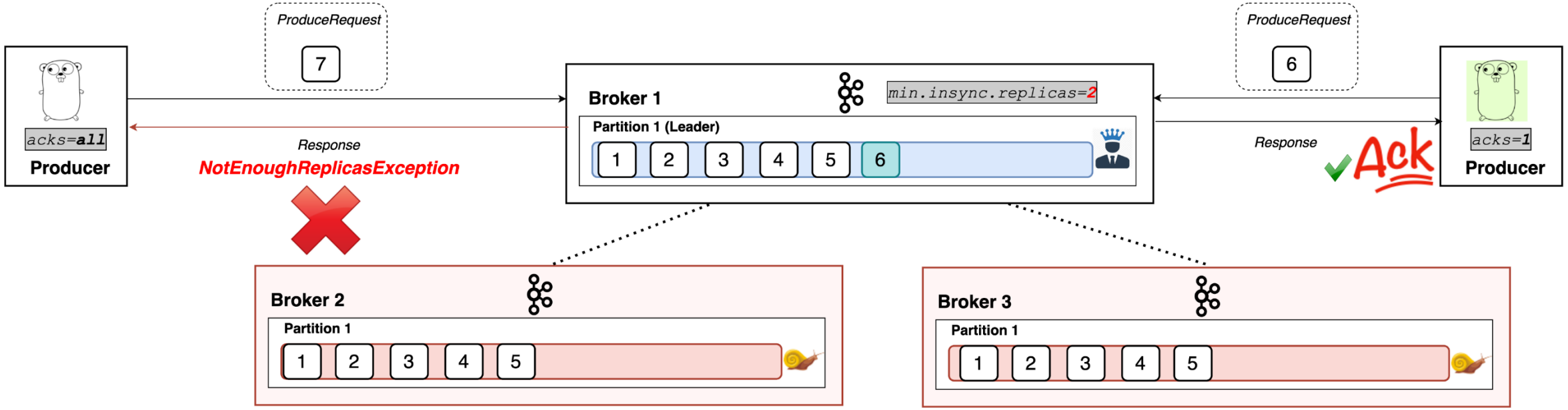
Message Delivery and Durability – acks = all



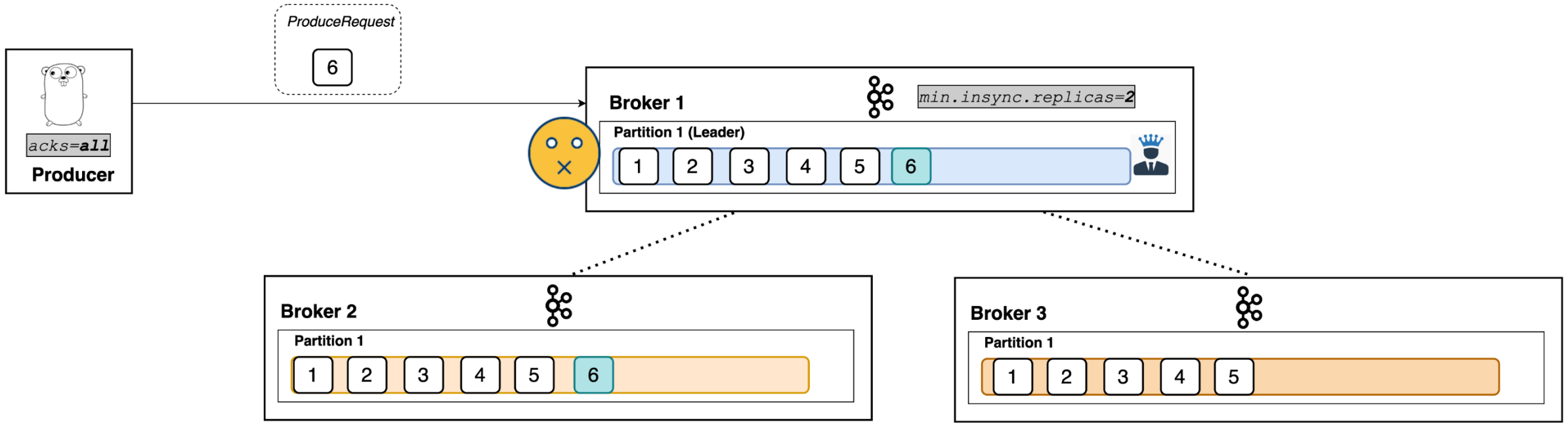
Message Delivery and Durability – Minimum In-Sync Replica



Message Delivery and Durability – Minimum In-Sync Replica



Message Delivery and Durability – Minimum In-Sync Replica

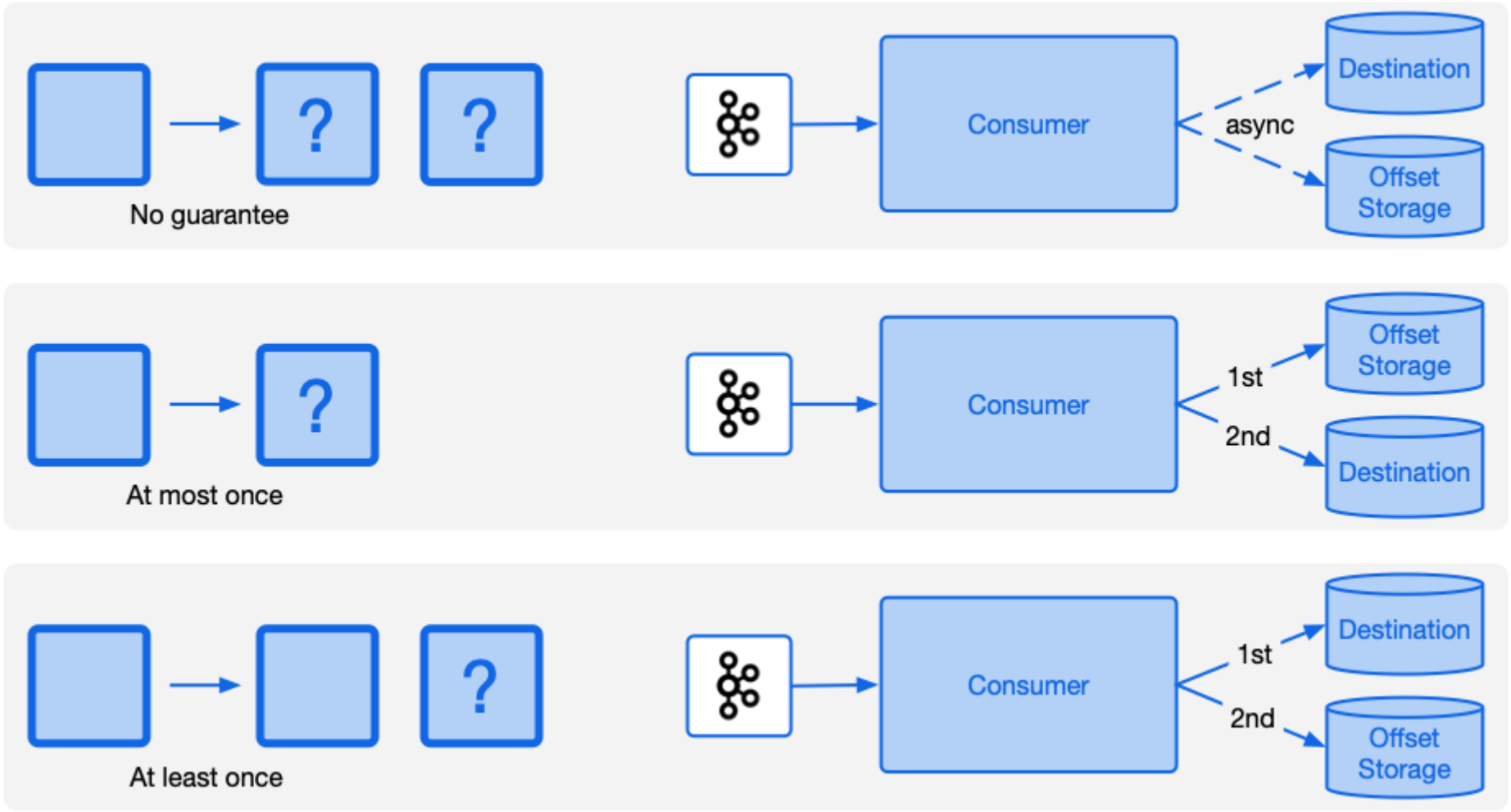


Processing Guarantees – Issues

- How do we guarantee all messages are processed?

- How do we avoid or handle duplicate messages?

Processing Guarantees – Concept



5 - Consumers

Consumer & Consumer Group

- Topic
- Producer - Tail
- Consumer – Own Place
- Scale - Partition
- Consumer Group
- Old - Zookeeper for Group Management
- New – Group Coordination Protocol

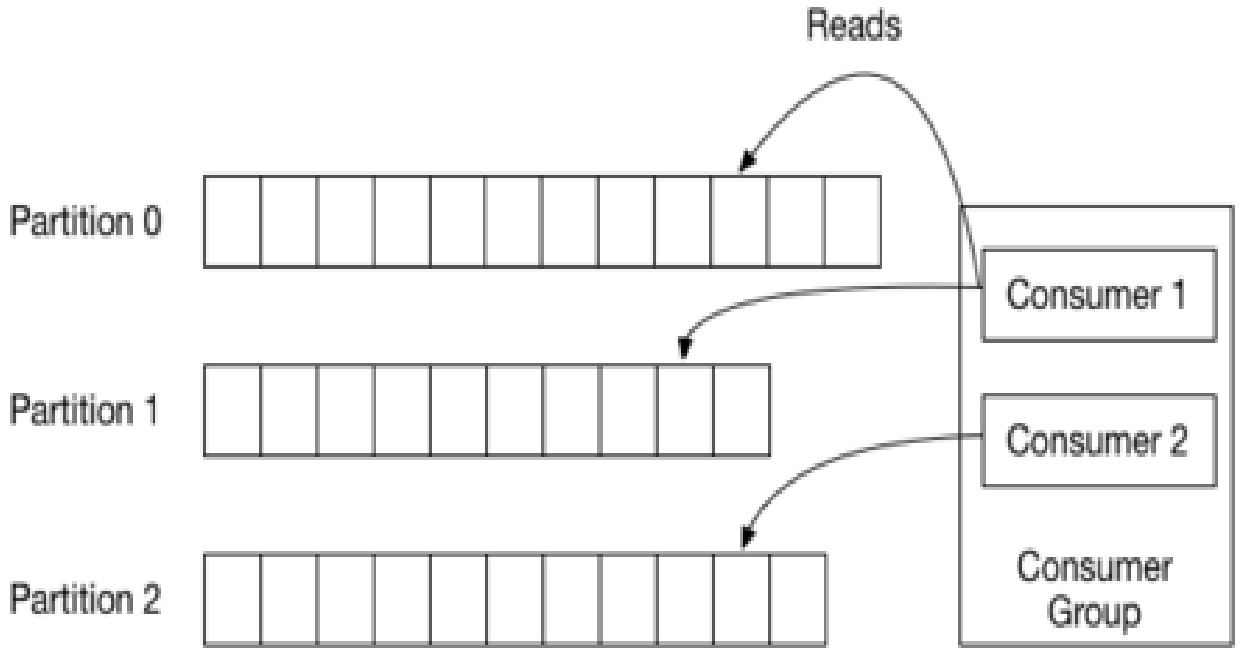
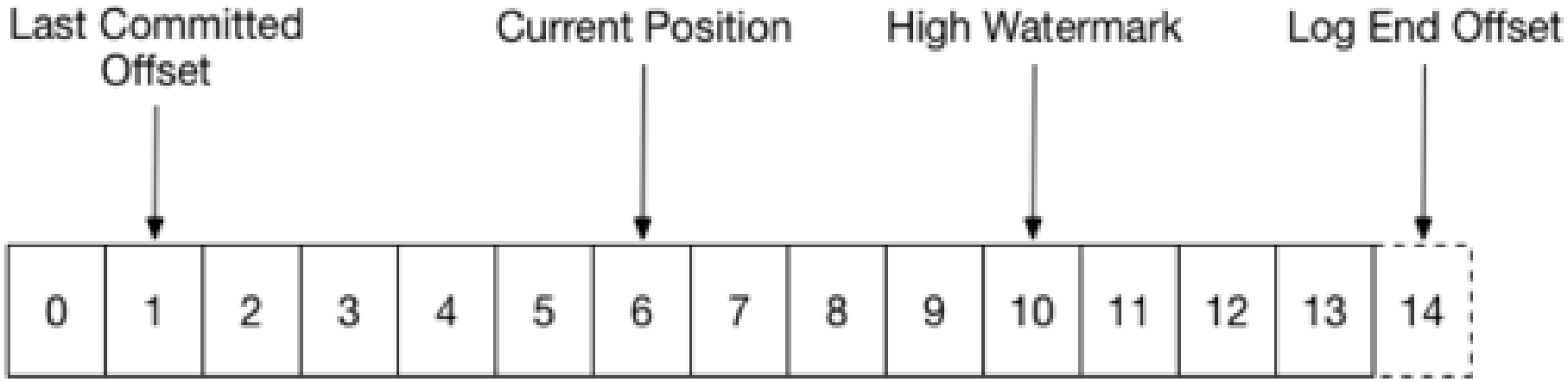
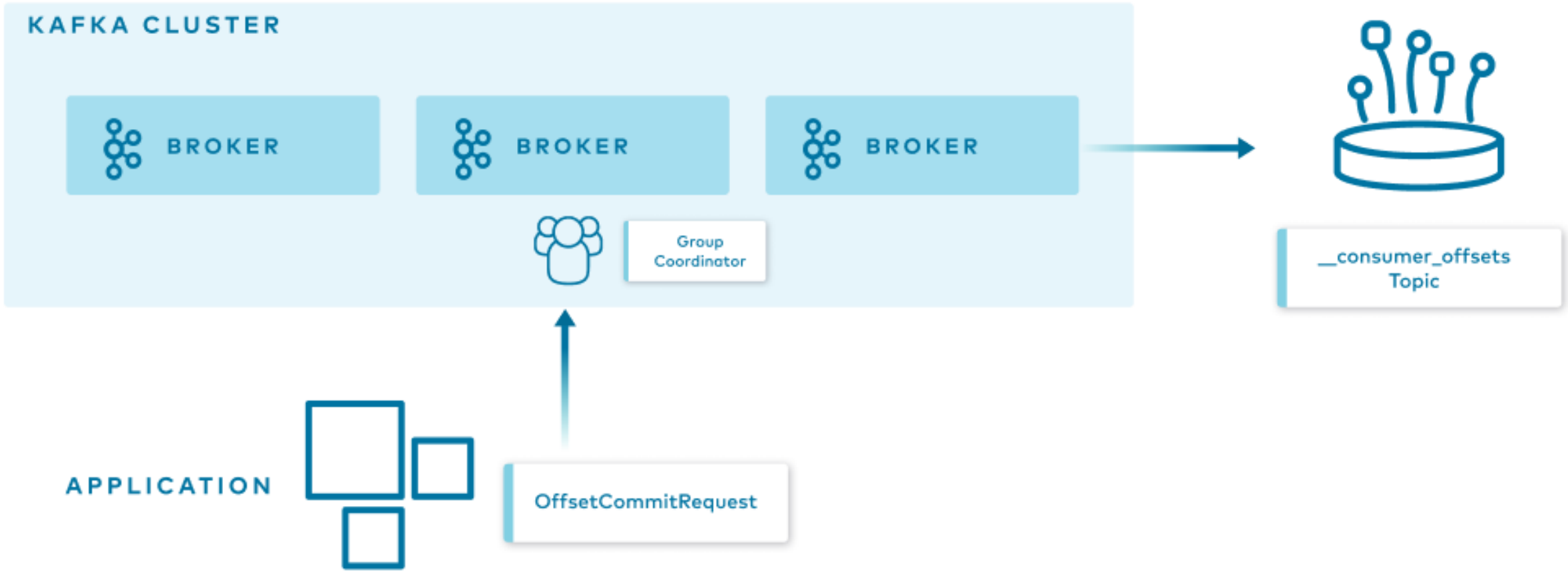


Figure 1: Consumer Group

Consumer Position



Consumer – Offset Storage



Consumer – Describe using CLI

```
./kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-group

TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG     CONSUMER-ID                                     HOST            CLIENT-ID
my-topic       0          2               4               2       consumer-1-029af89c-873c-4751-a720-cefd41a669d6  /127.0.0.1      consumer-1
my-topic       1          2               3               1       consumer-1-029af89c-873c-4751-a720-cefd41a669d6  /127.0.0.1      consumer-1
my-topic       2          2               3               1       consumer-2-42c1abd4-e3b2-425d-a8bb-e1ea49b29bb2  /127.0.0.1      consumer-2
```

Consumer – Lag

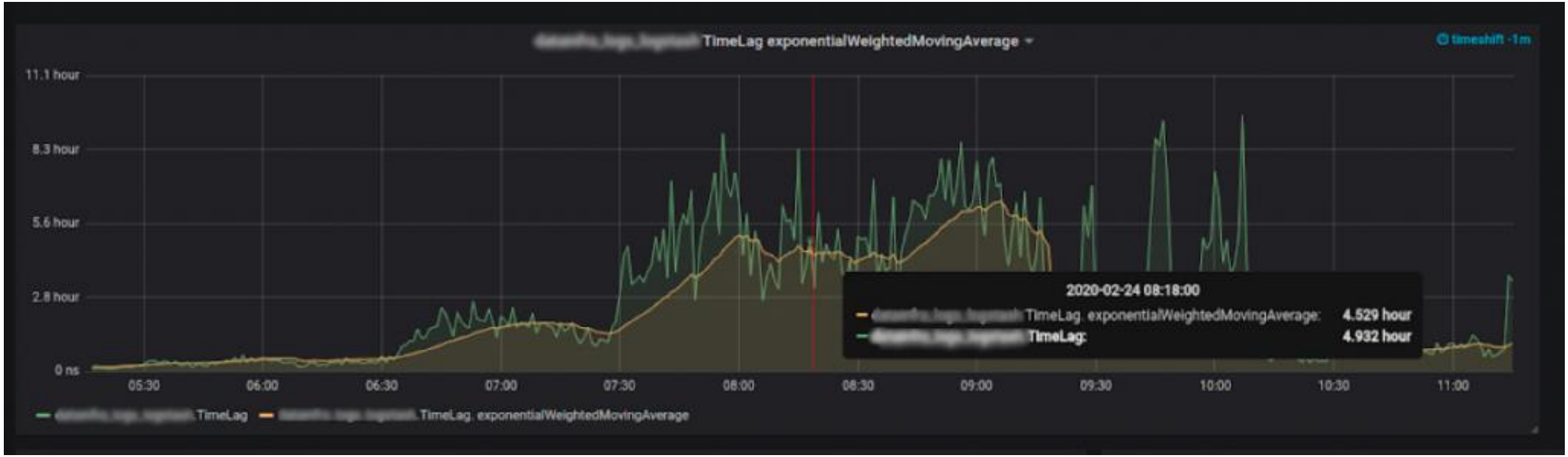
- Key Performance Indicator - KPI
- Delta between last committed to last produces
- Keep lag to minimum
  - Kafka Persistence is based on retention
  - Lag persists
  - Lose data at some point in time

Consumer Lag Explore

- Kafka Lag Exporter
  - native Kubernetes support
  - contained time-based lag monitoring
- Burrow
  - Burrow is an active LinkedIn project
  - Has an active community
- Remora
  - Was created after Zalando spent some time using Burrow.
  - Has Datadog and CloudWatch integration

Burrow dashboards

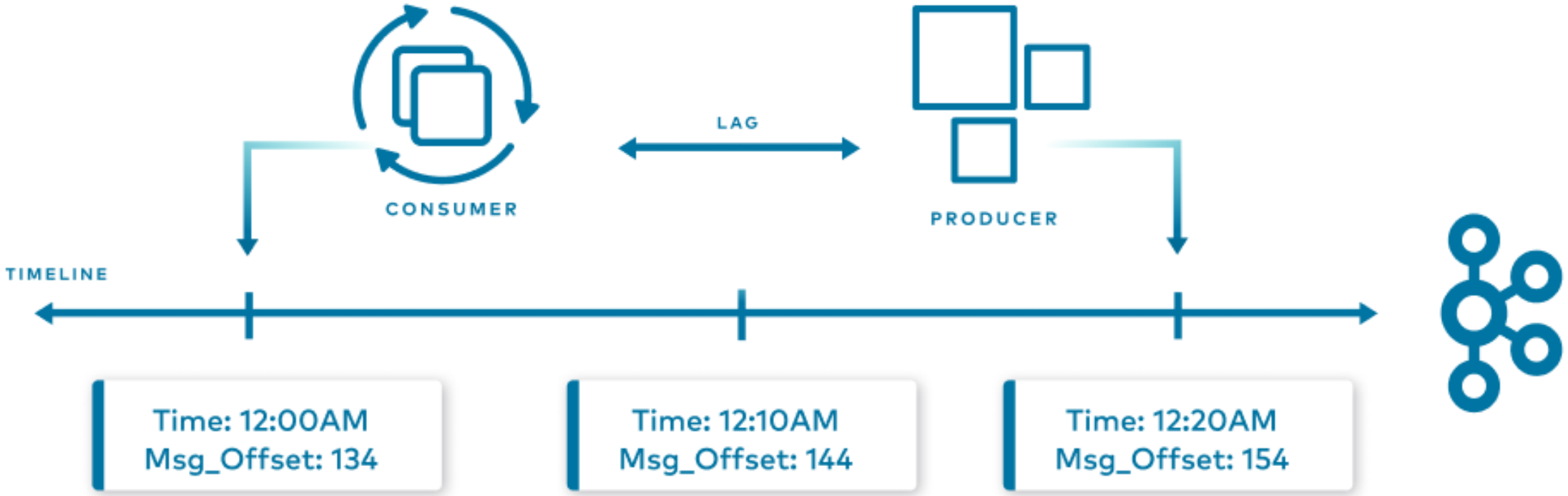
More Important – Time-Based



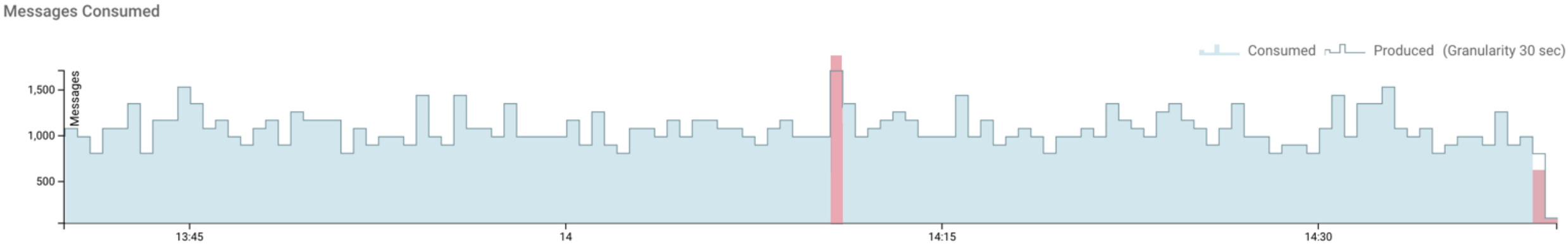
Consumer Lag – Time Lag

$$\frac{\text{Diff (Last\_Consumed, Last\_Produced)}}{\text{Producer Rate}}$$

Consumer Lag – Time Lag - Visual

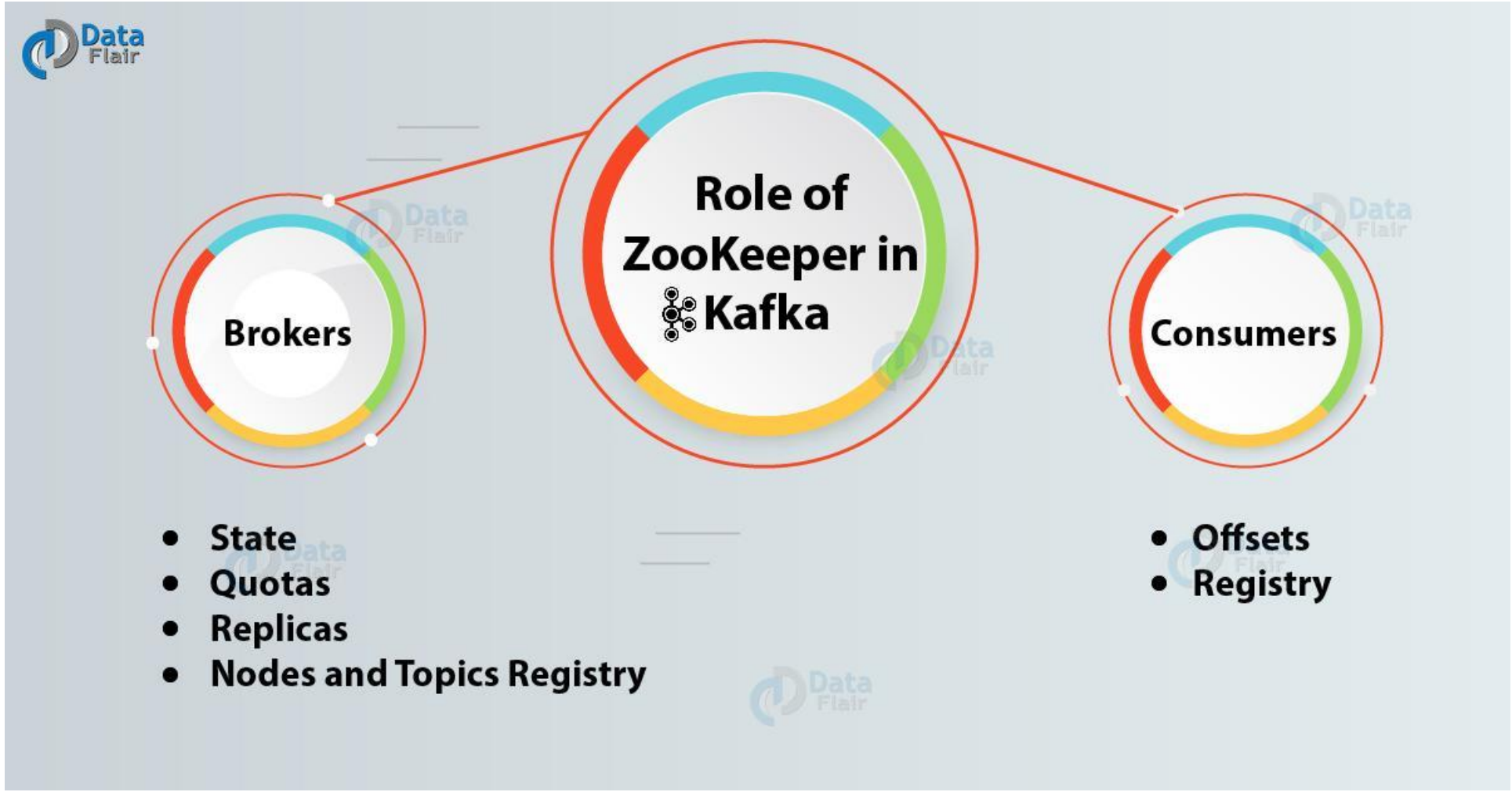


Consumer - Over and Under Consumption



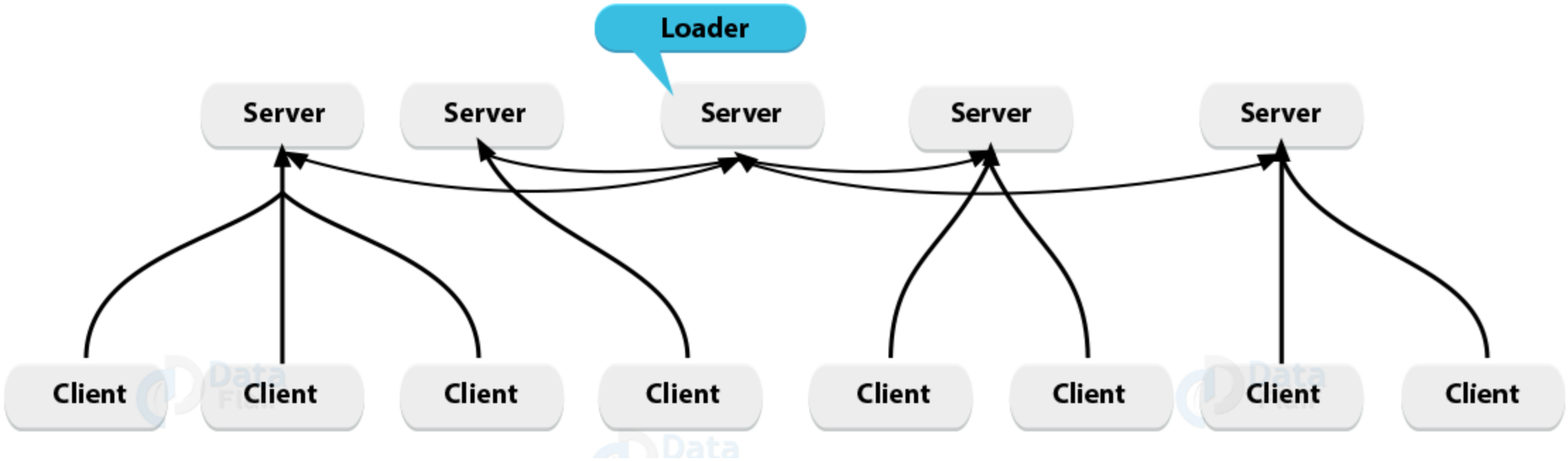
6 - Configuration and Dependency

Dependency - Zookeeper

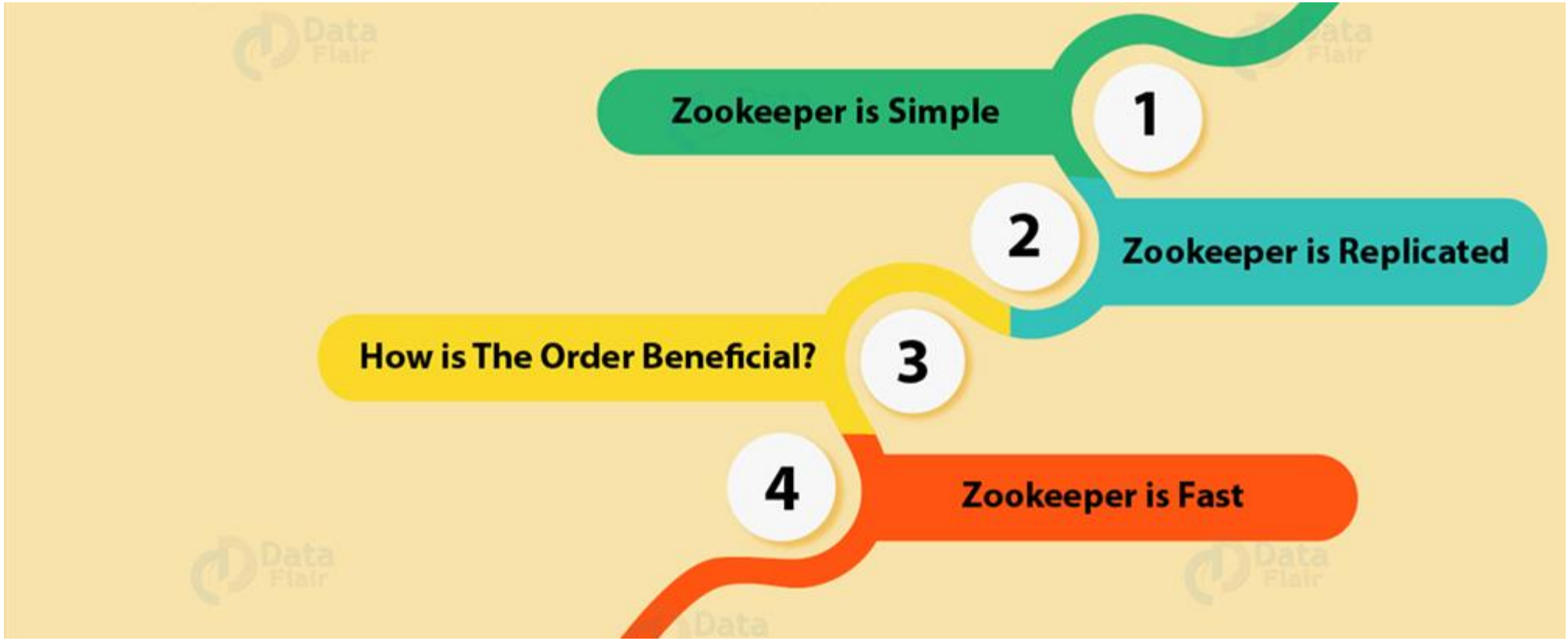


Zookeeper Architecture

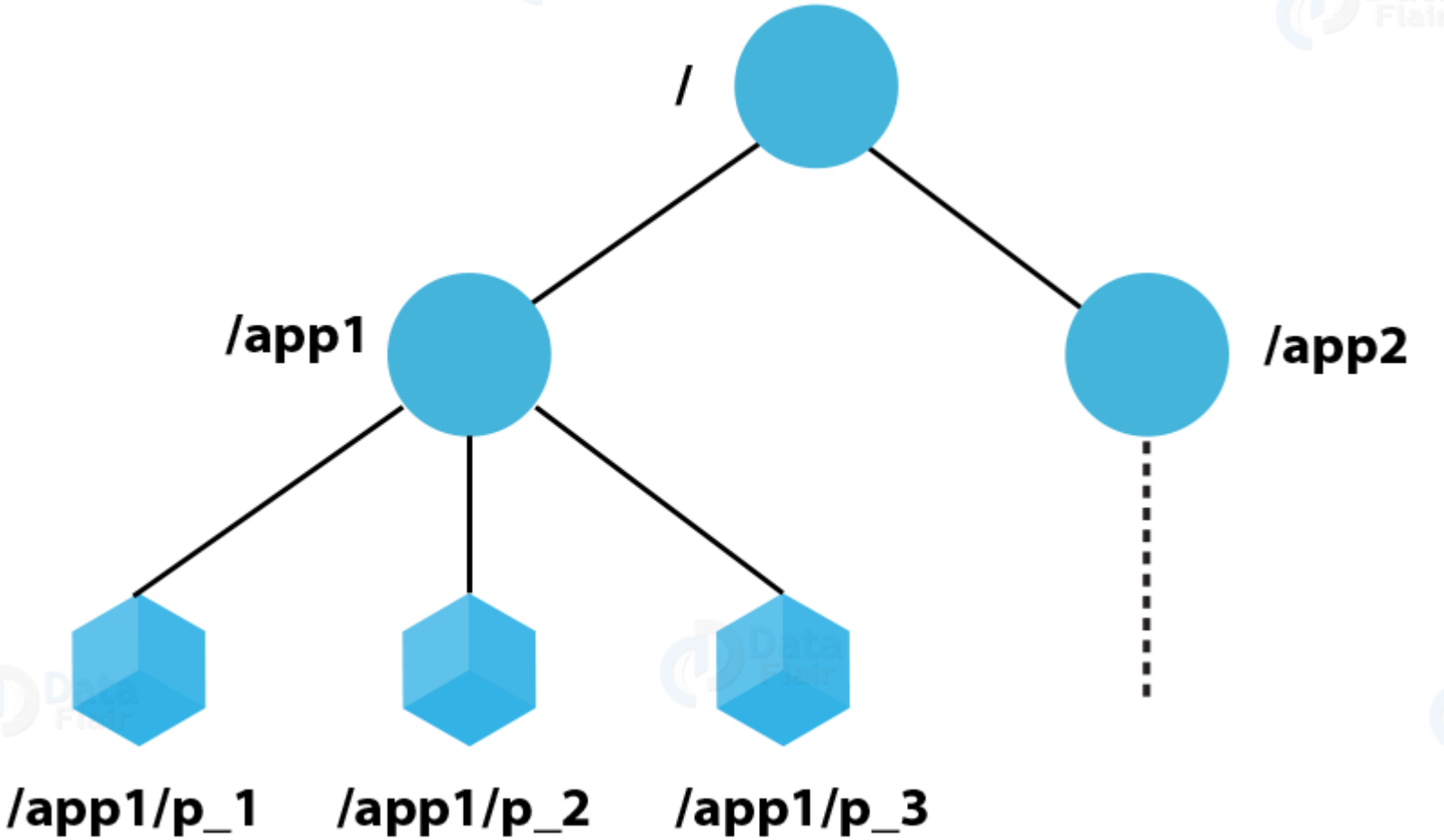
Zookeeper – Client / Server Architecture



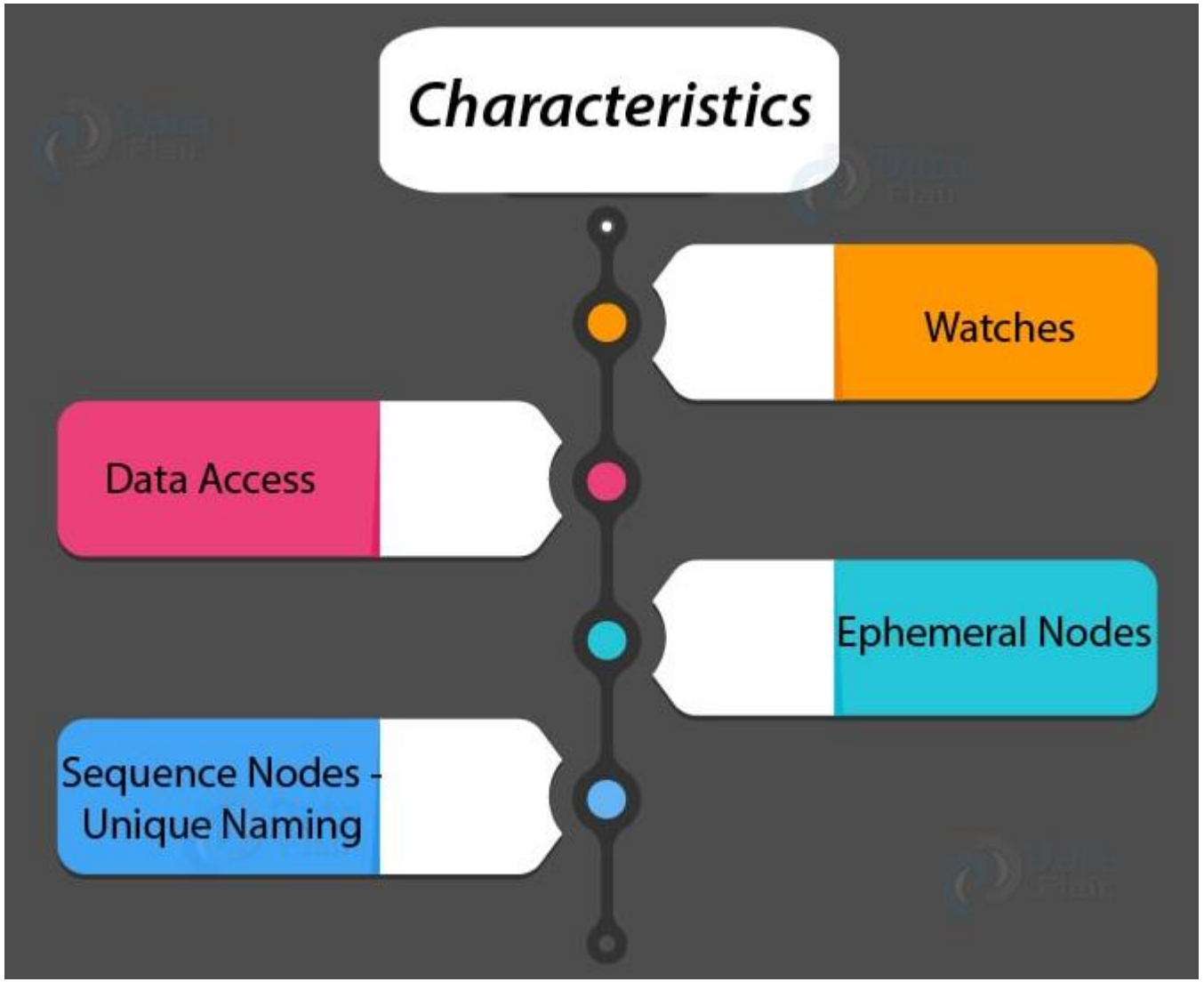
Design Goals of Zookeeper Architecture



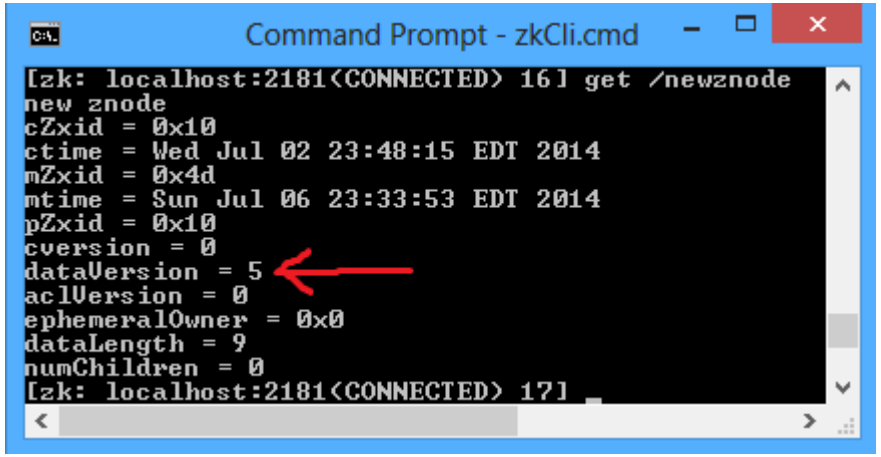
Data Model in ZooKeeper

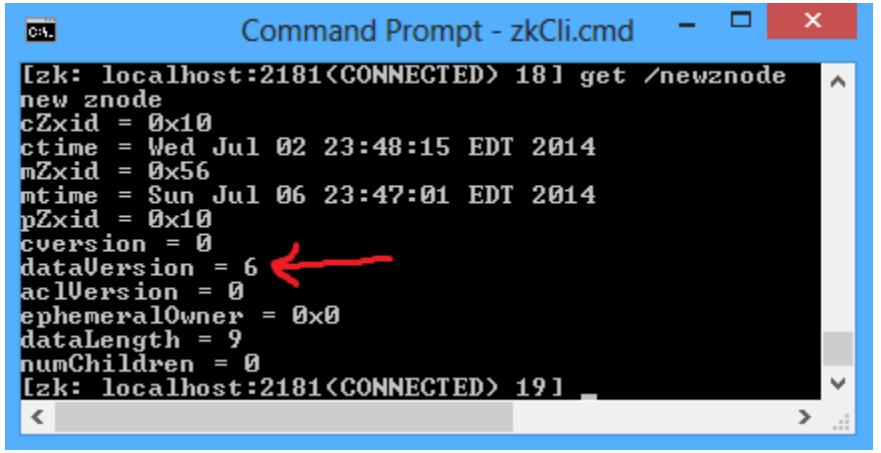


Zookeeper ZNodes



Zookeeper – Versions





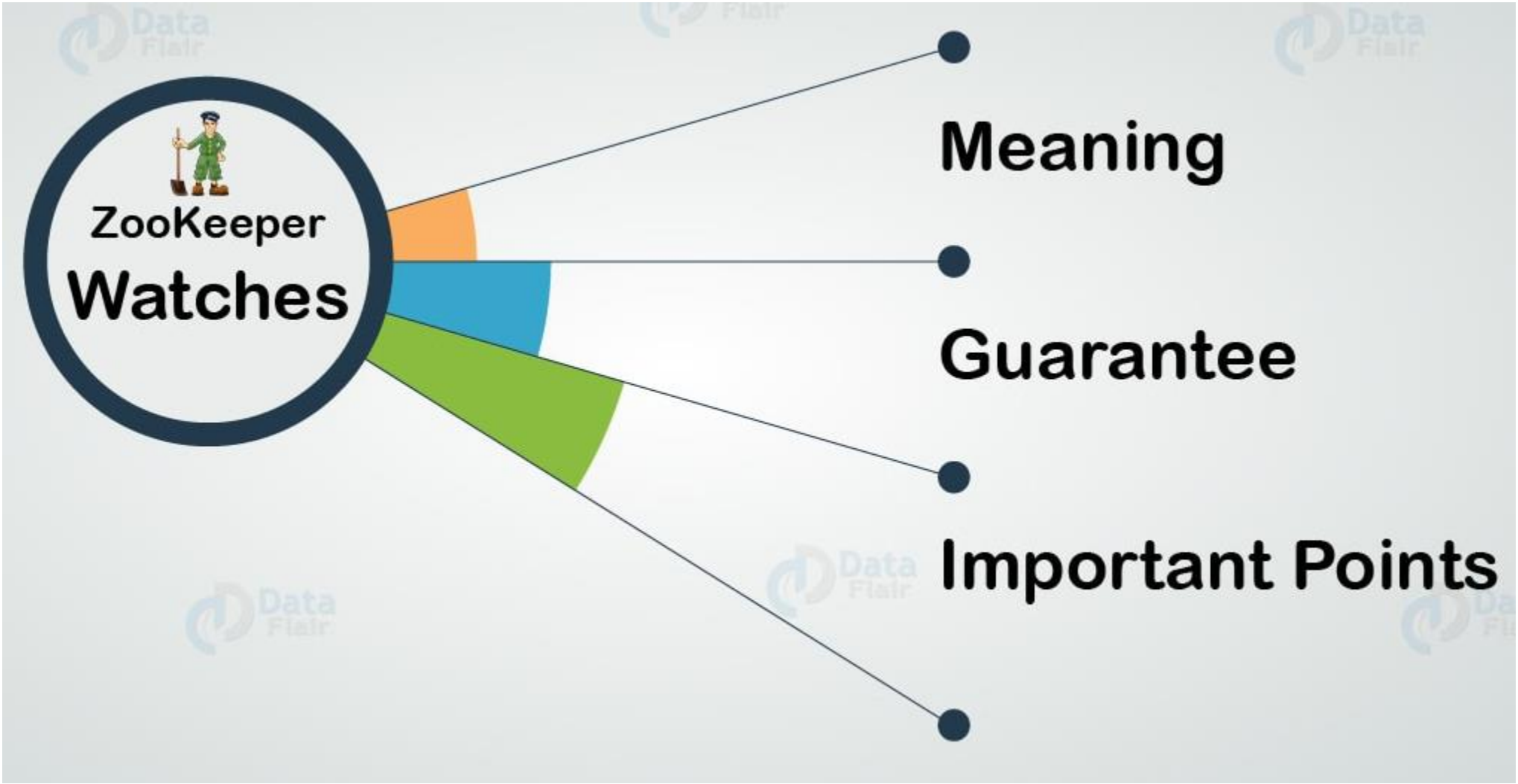

```
26          zkc.connect("localhost");
27          ZooKeeper zk = zkc.getZooKeeper();
28          Stat stat = zk.exists("/newznode", true);
29          zk.setData("/newznode", "new znode".getBytes(), 8);
```

Markers Properties Data Source Explorer Snippets Console

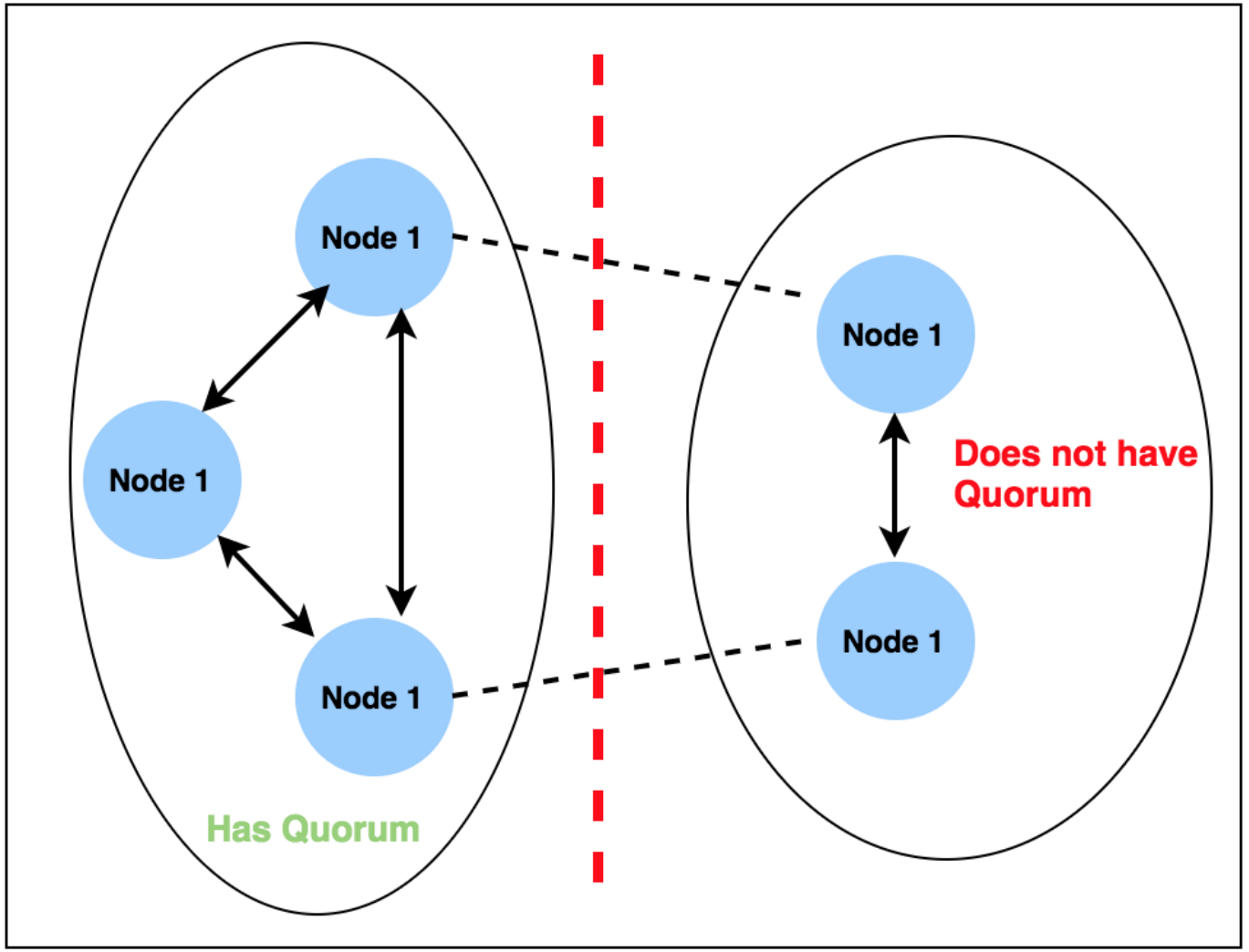
<terminated> App [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jul 6, 2014, 11:51:50 PM)

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Exception in thread "main" org.apache.zookeeper.KeeperException$BadVersionException: KeeperErrorCode = BadVersion for /newznode
        at org.apache.zookeeper.KeeperException.create(KeeperException.java:115)
        at org.apache.zookeeper.KeeperException.create(KeeperException.java:51)
        at org.apache.zookeeper.ZooKeeper.setData(ZooKeeper.java:1270)
        at com.zook.app.App.main(App.java:29)
```

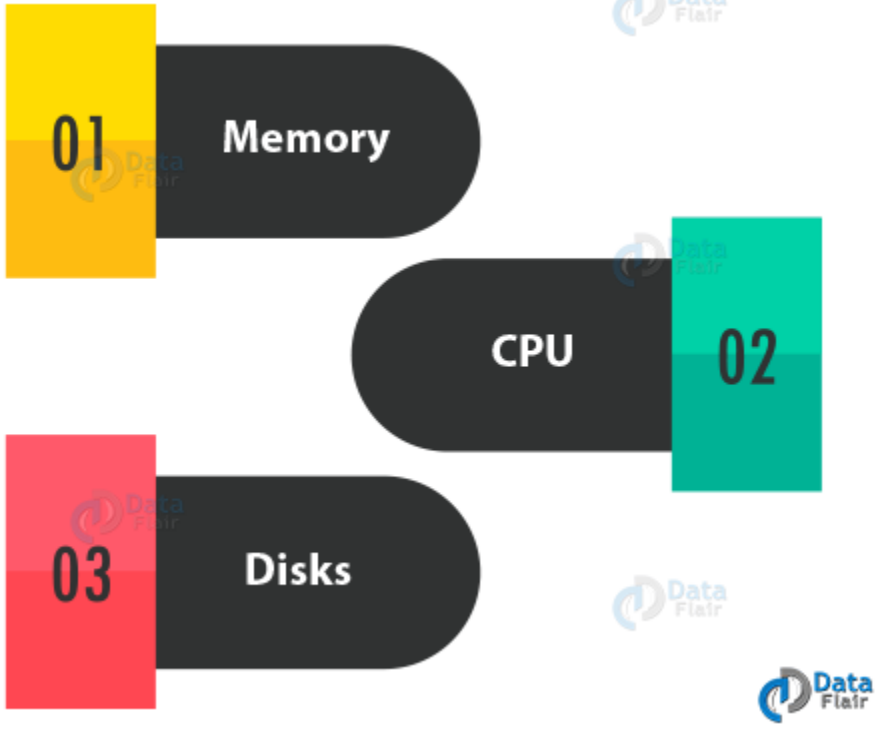
Zookeeper – Watches



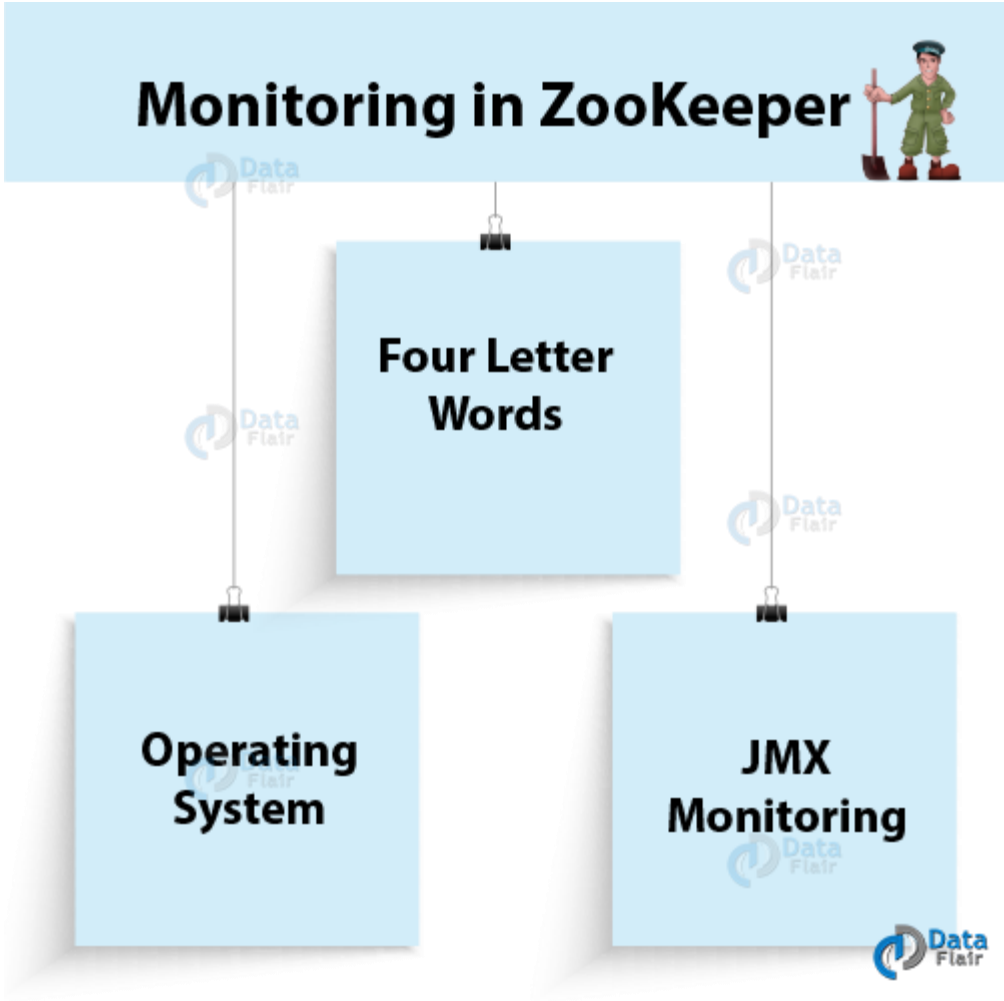
Zookeeper - Quorums



Hardware of Zookeeper



Zookeeper - Monitoring



Kafka Requirement

Kafka – Important Configuration

Level	Important Configuration
Operation System – OS	❖ File descriptor limits ❖ Max socket buffer ❖ Max number of memory map
Producer	❖ acks ❖ compression ❖ batch size
Consumer	fetch size
Java	8 and 11

Kafka – Linkedin busiest Cluster

Cluster Status

- 60 brokers
- 50k partitions (replication factor 2)
- 800k messages/sec in
- 300 MB/sec inbound
- 1 GB/sec outbound

JVM Argument

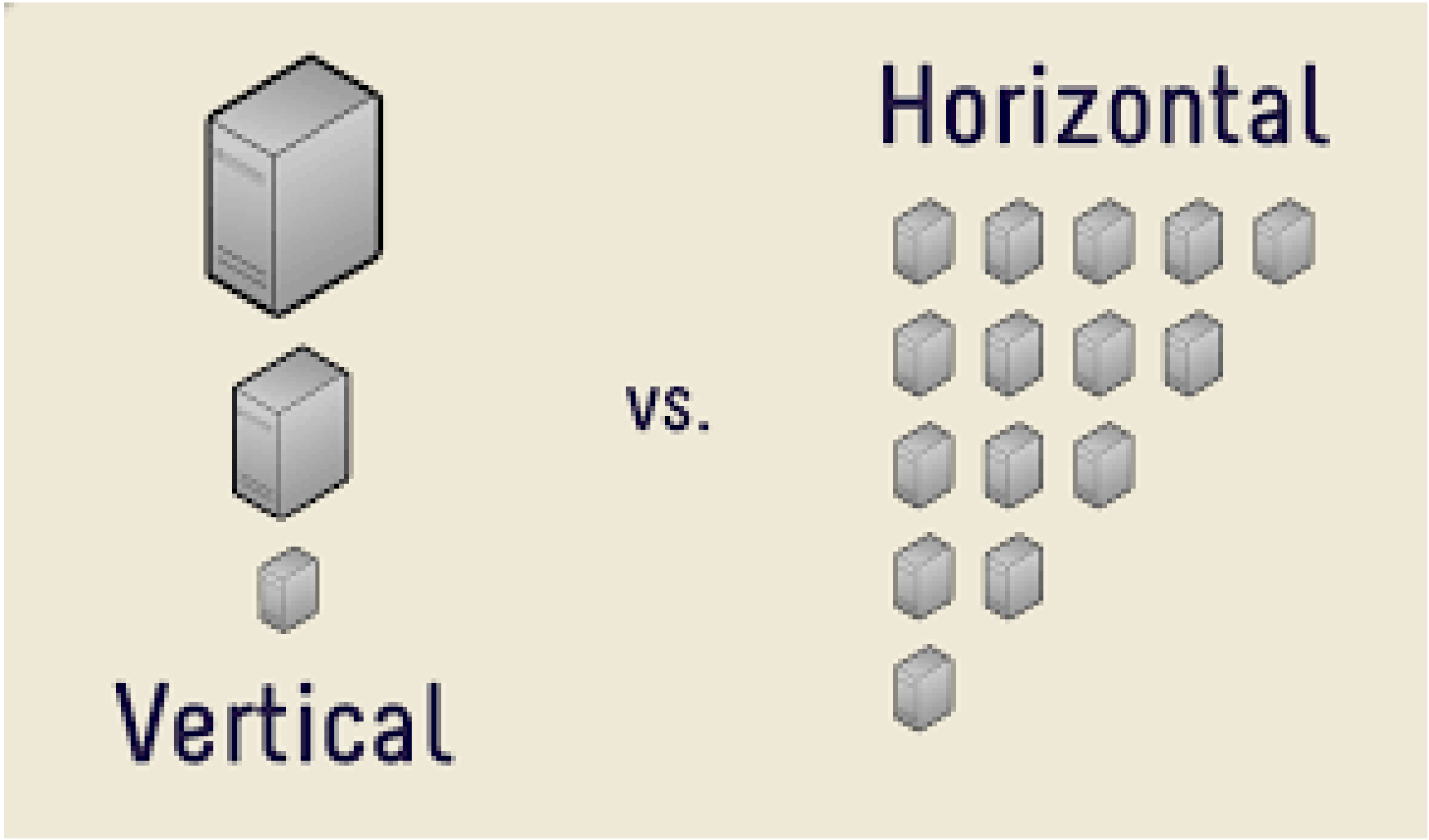
- -XX:+UseG1GC
- -XX:MaxGCPauseMillis=20
- -XX:InitiatingHeapOccupancyPercent=35
- -XX:G1HeapRegionSize=16M
- -XX:MinMetaspaceFreeRatio=50
- -XX:MaxMetaspaceFreeRatio=80
- -XX:+ExplicitGCInvokesConcurrent

GC Result

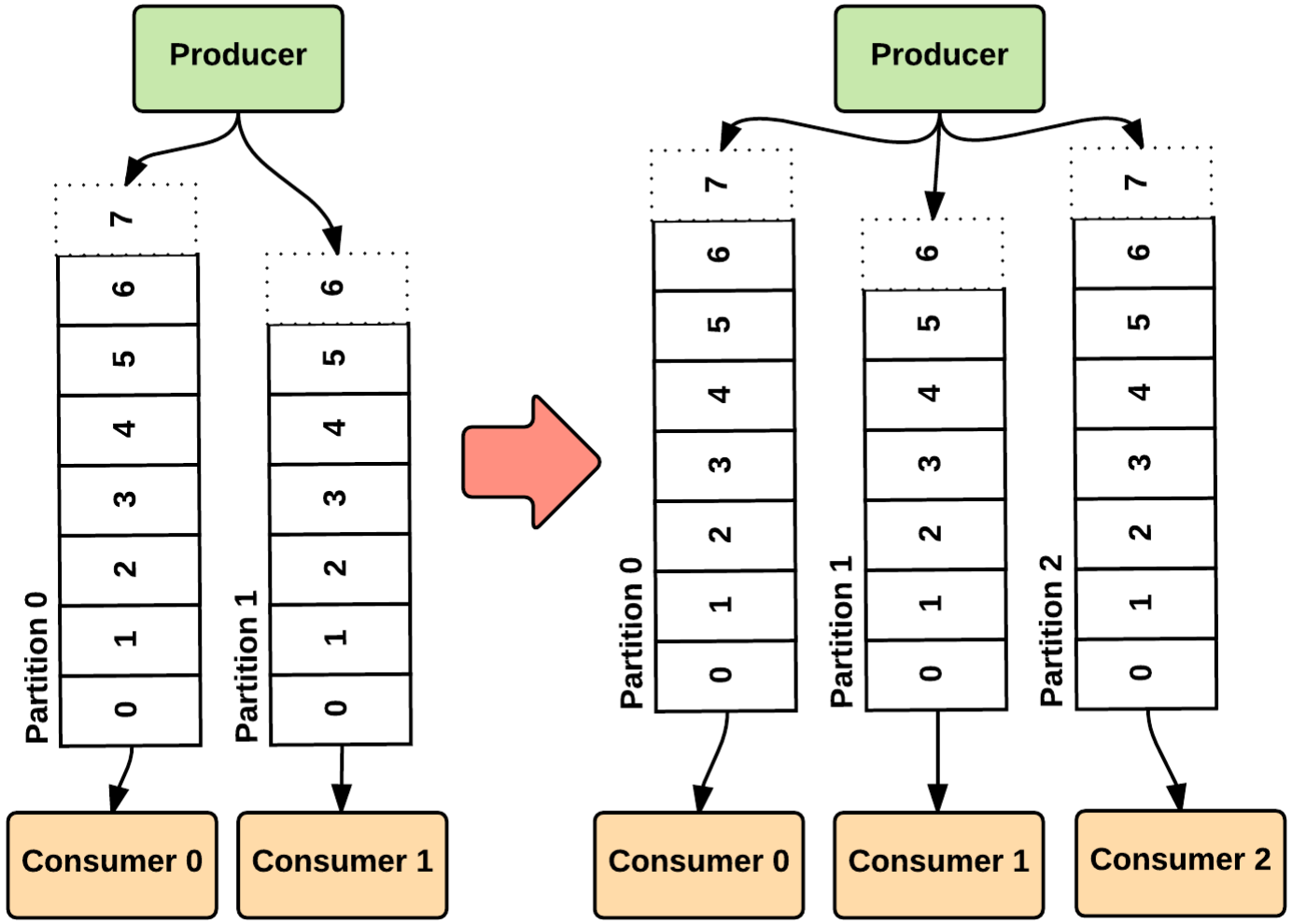
- 90% GC pause time of about 21ms
- less than 1% young GC per second

7 - Clustering – Scalability, Fault Tolerant and High Availability

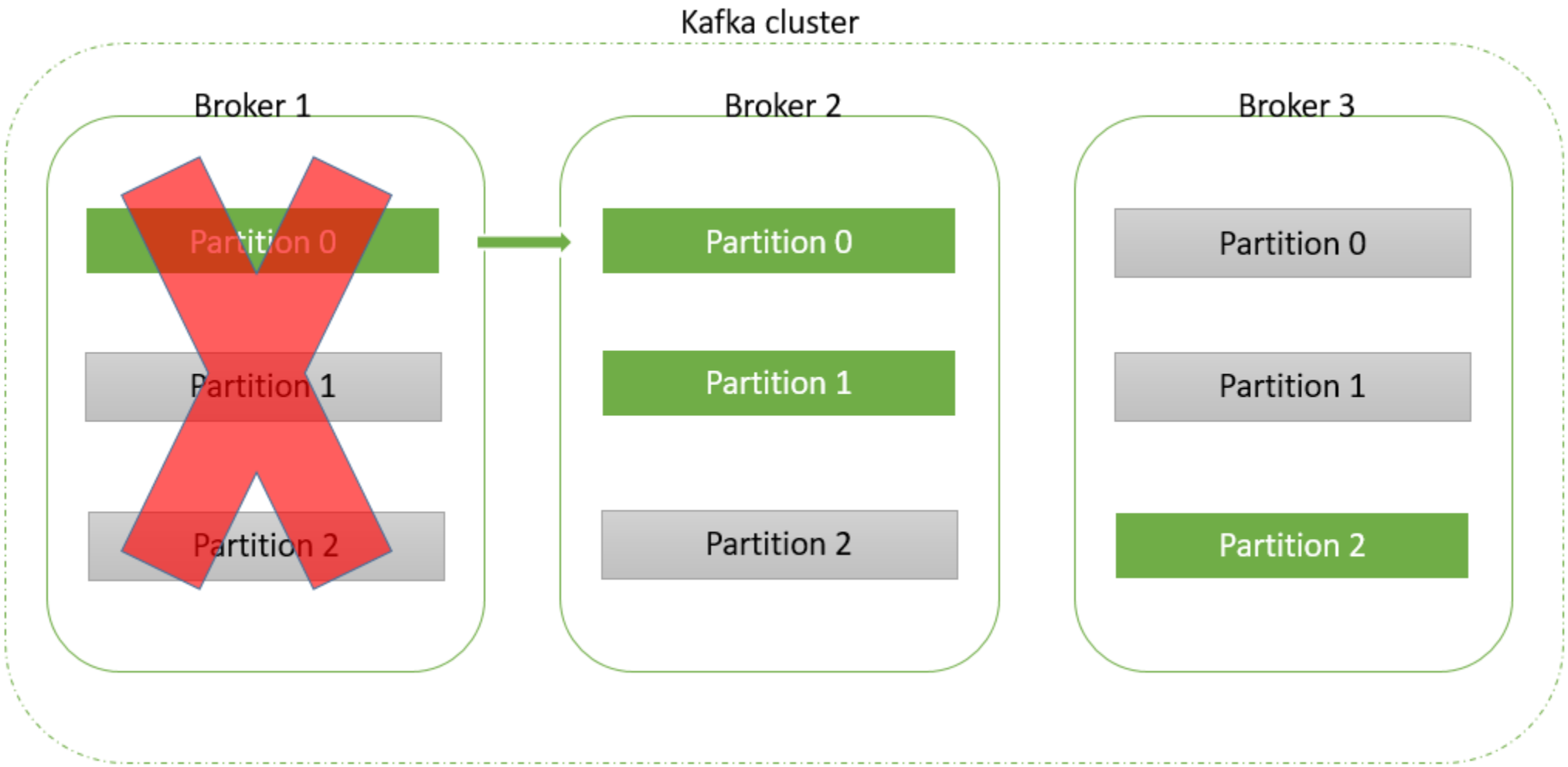
Scaling – Vertical vs Horizontal



Scale-out architecture with Kafka

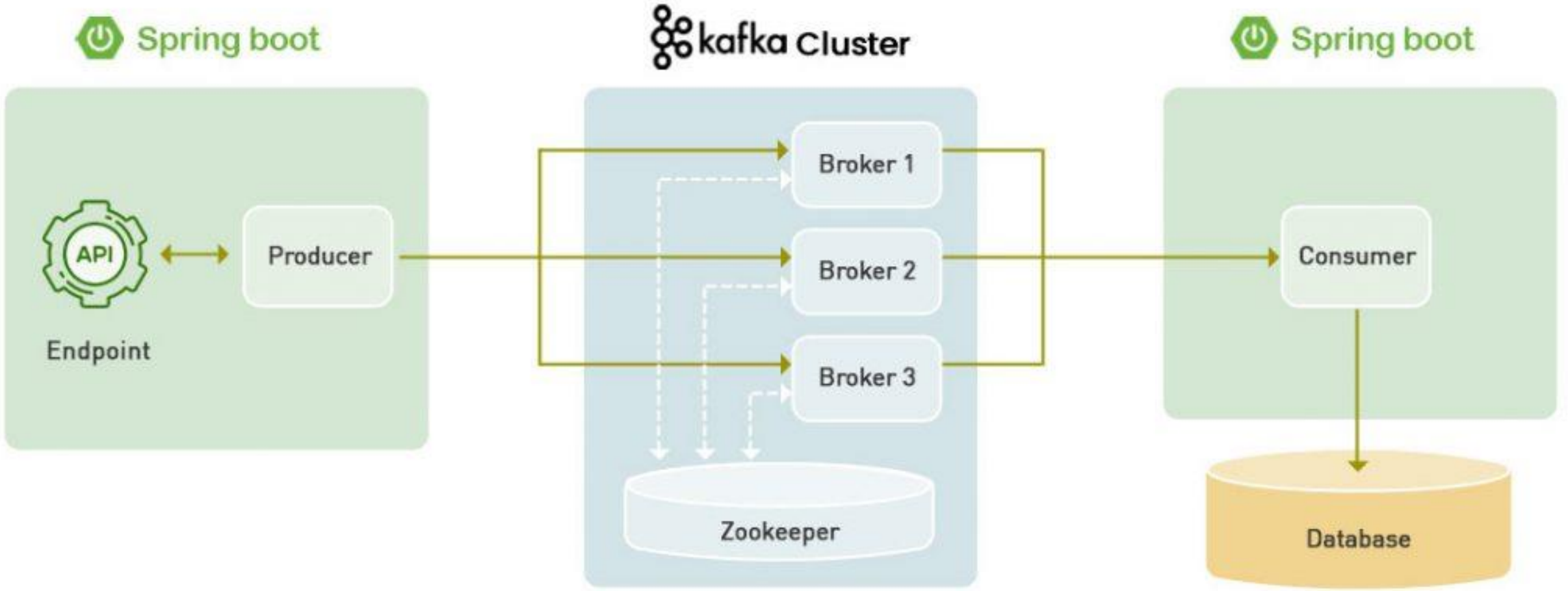


Fault Tolerance

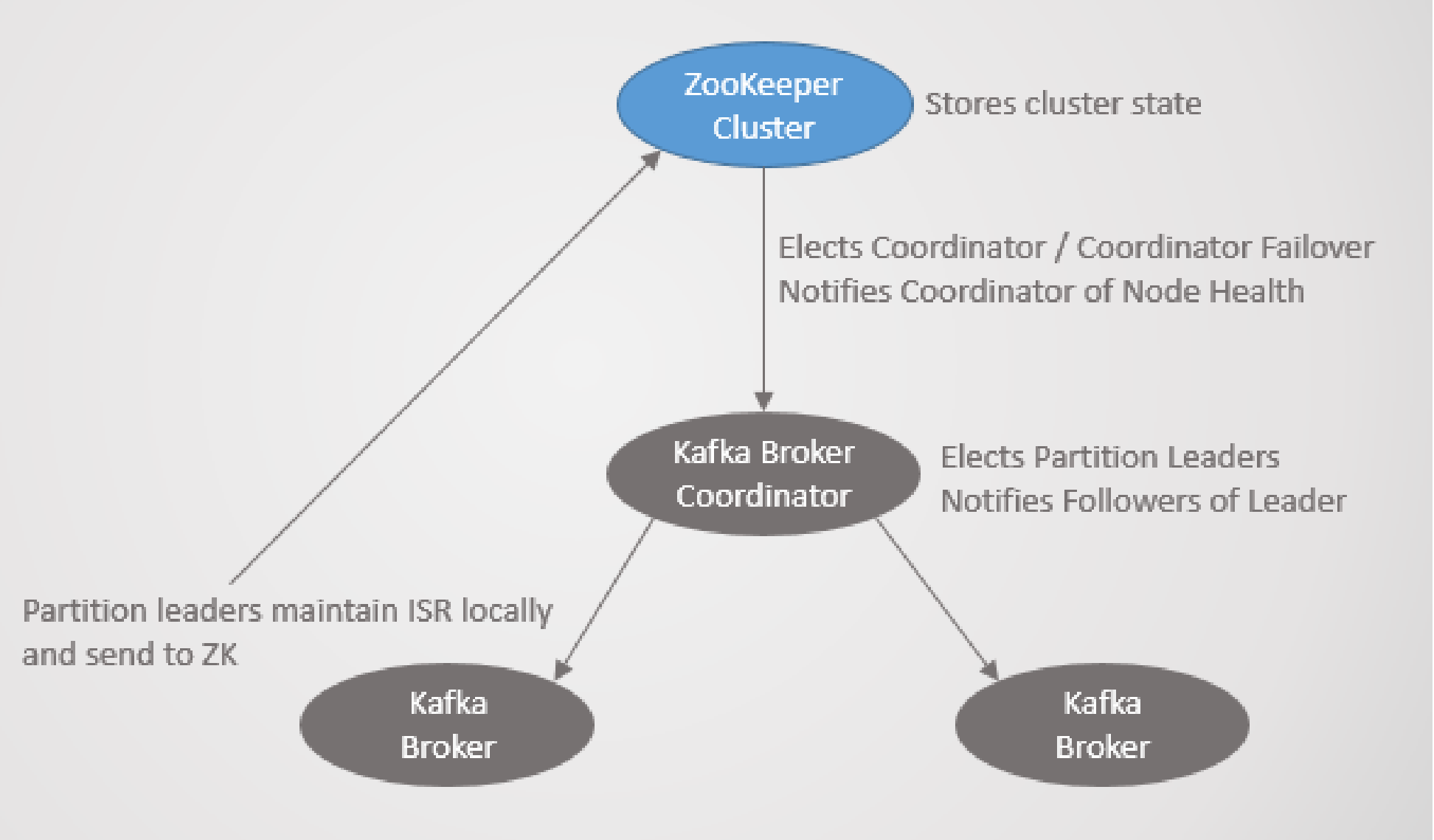


Availability

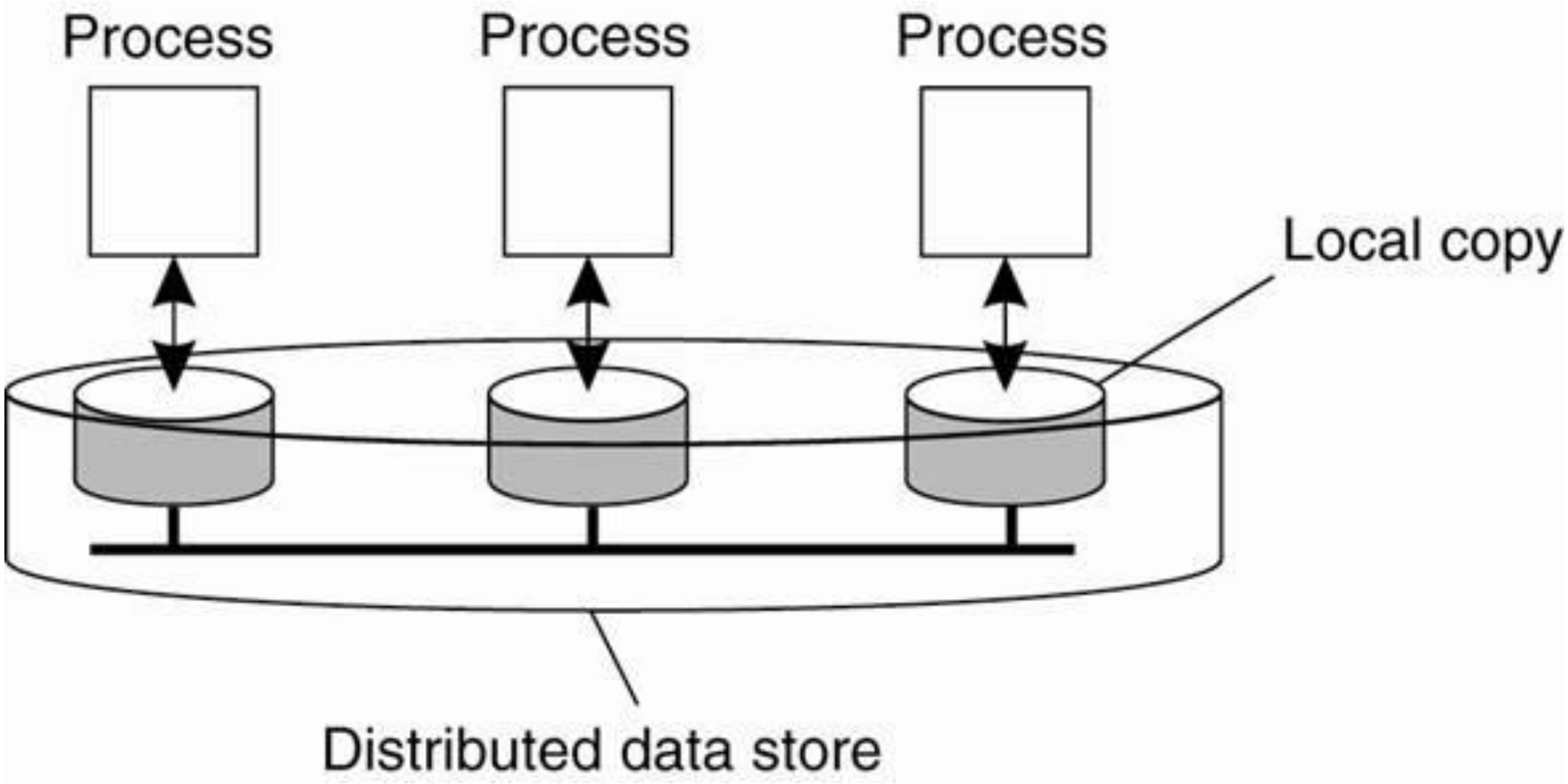
Ensuring Client Connectivity



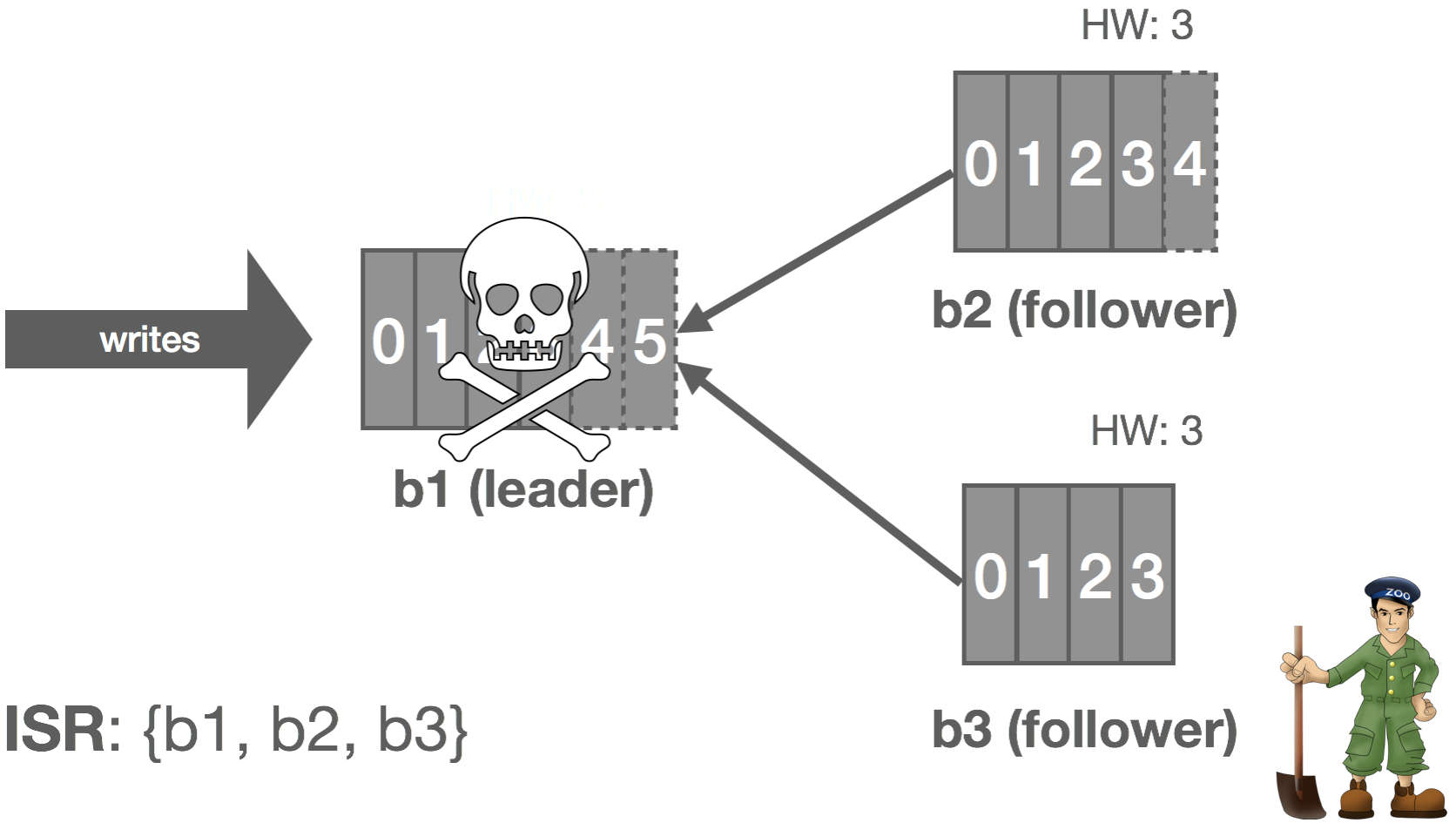
The Kafka Consensus Architecture



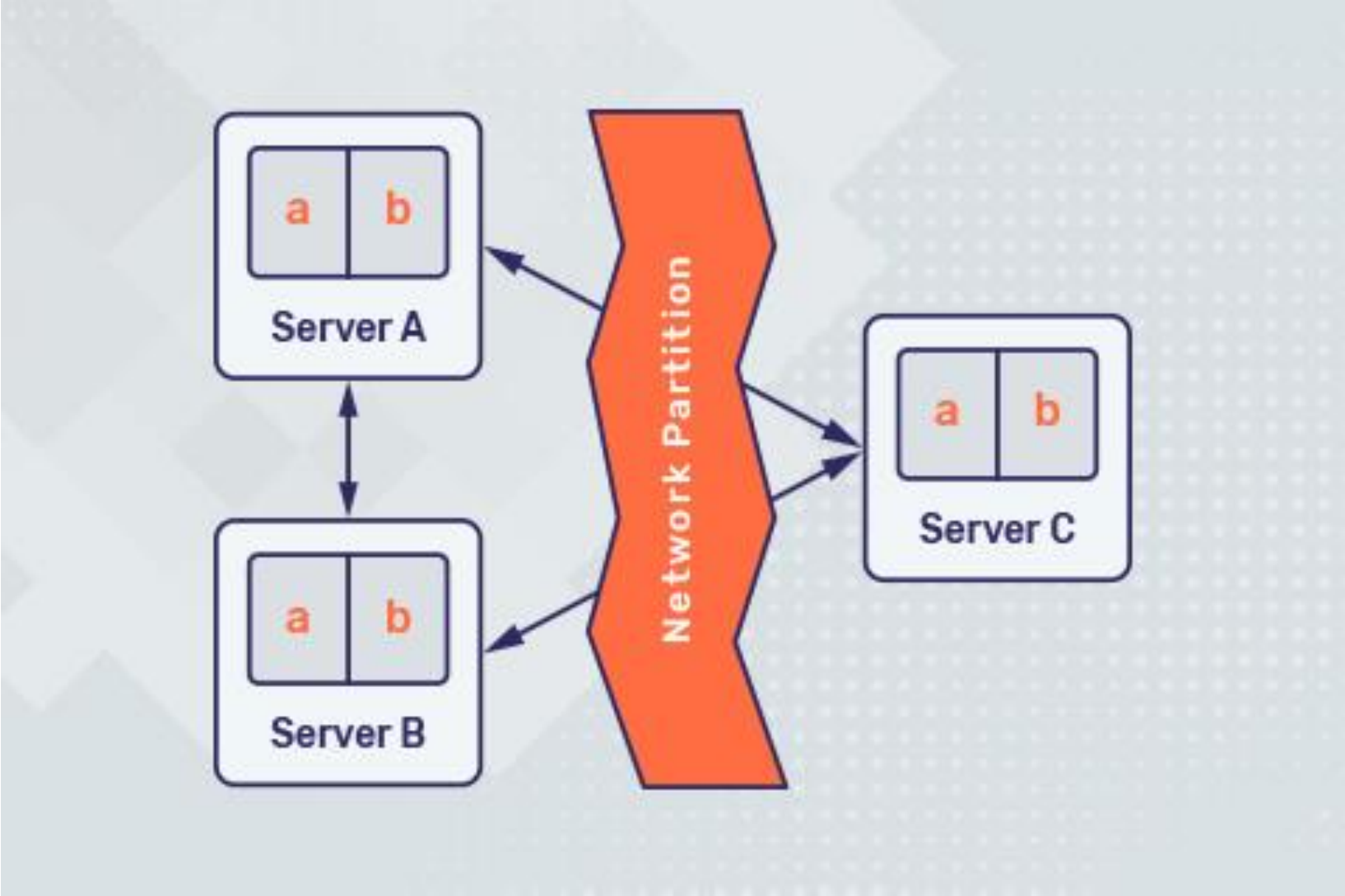
The Replication Protocol



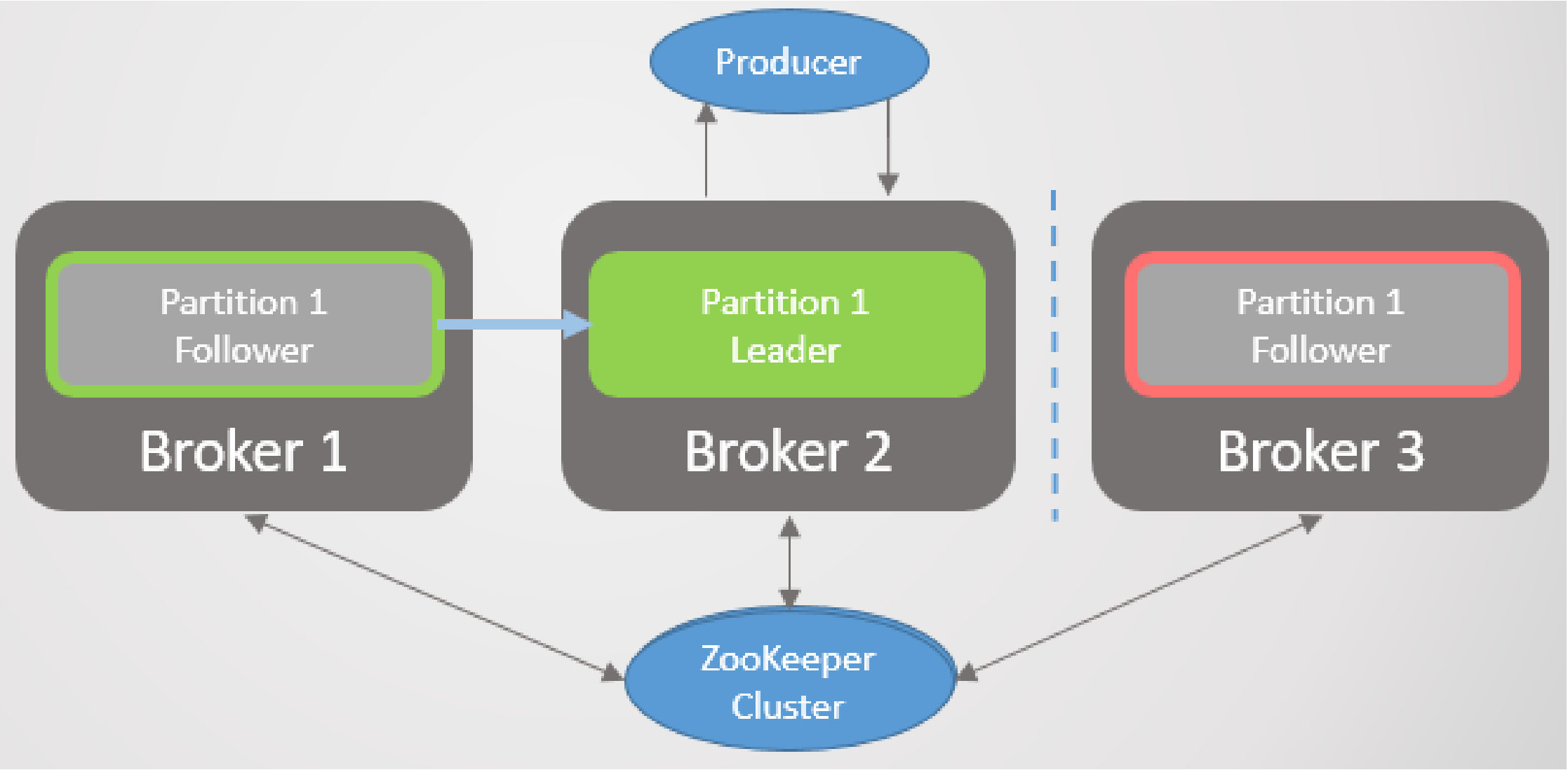
Leader Fail-Over



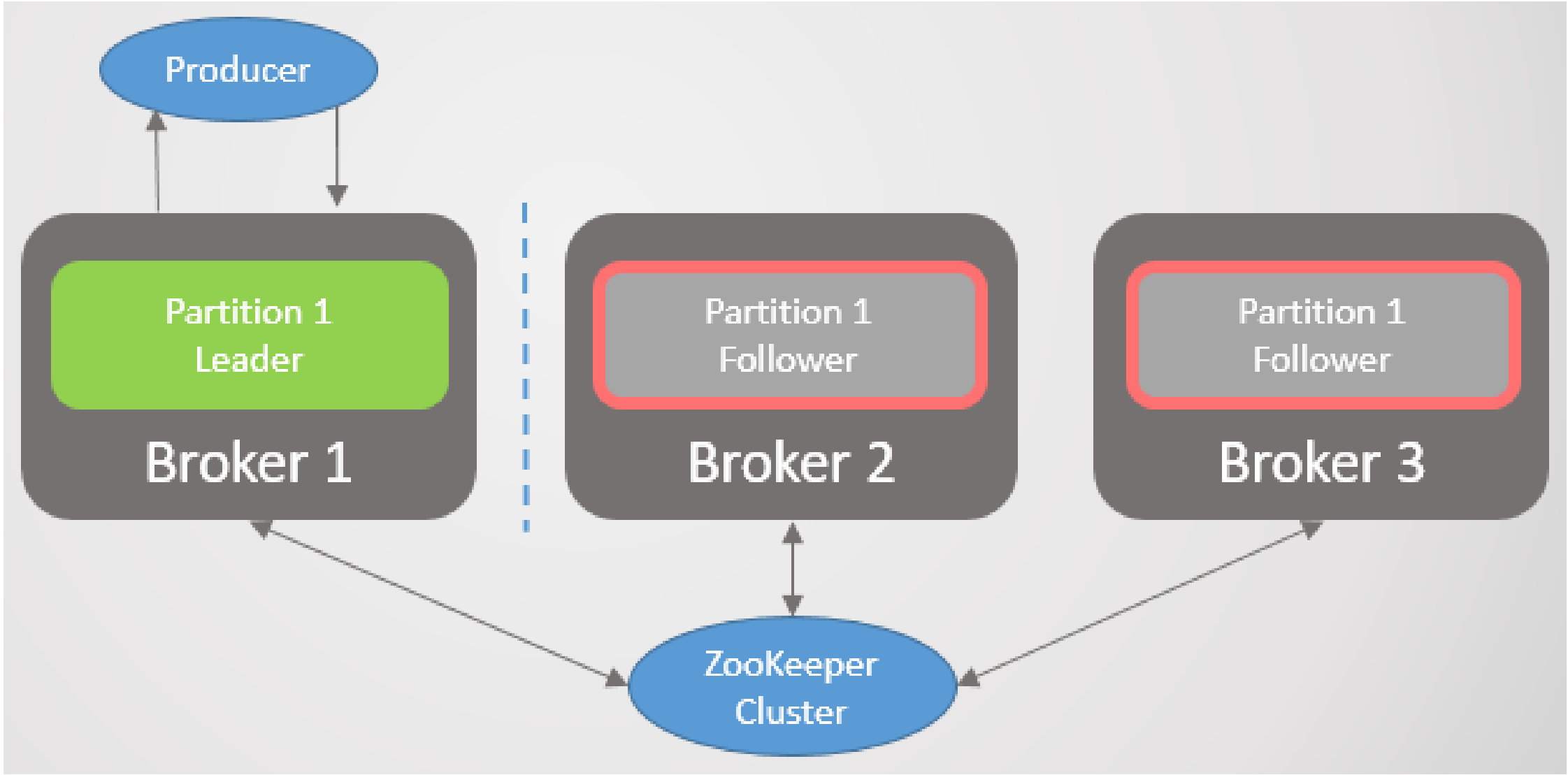
Network Partitions



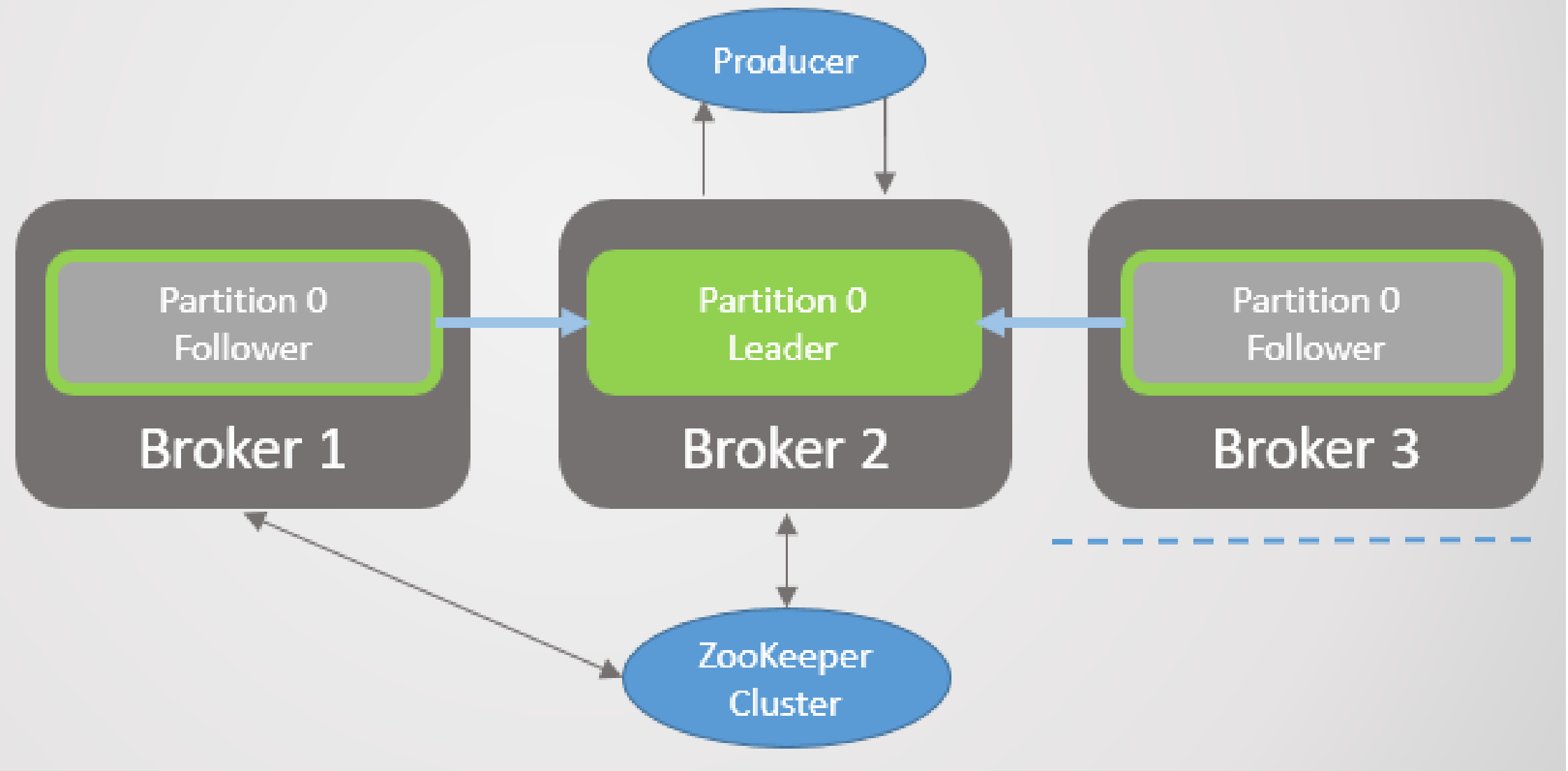
Scenario 1: A follower cannot see the leader, but can still see Zookeeper



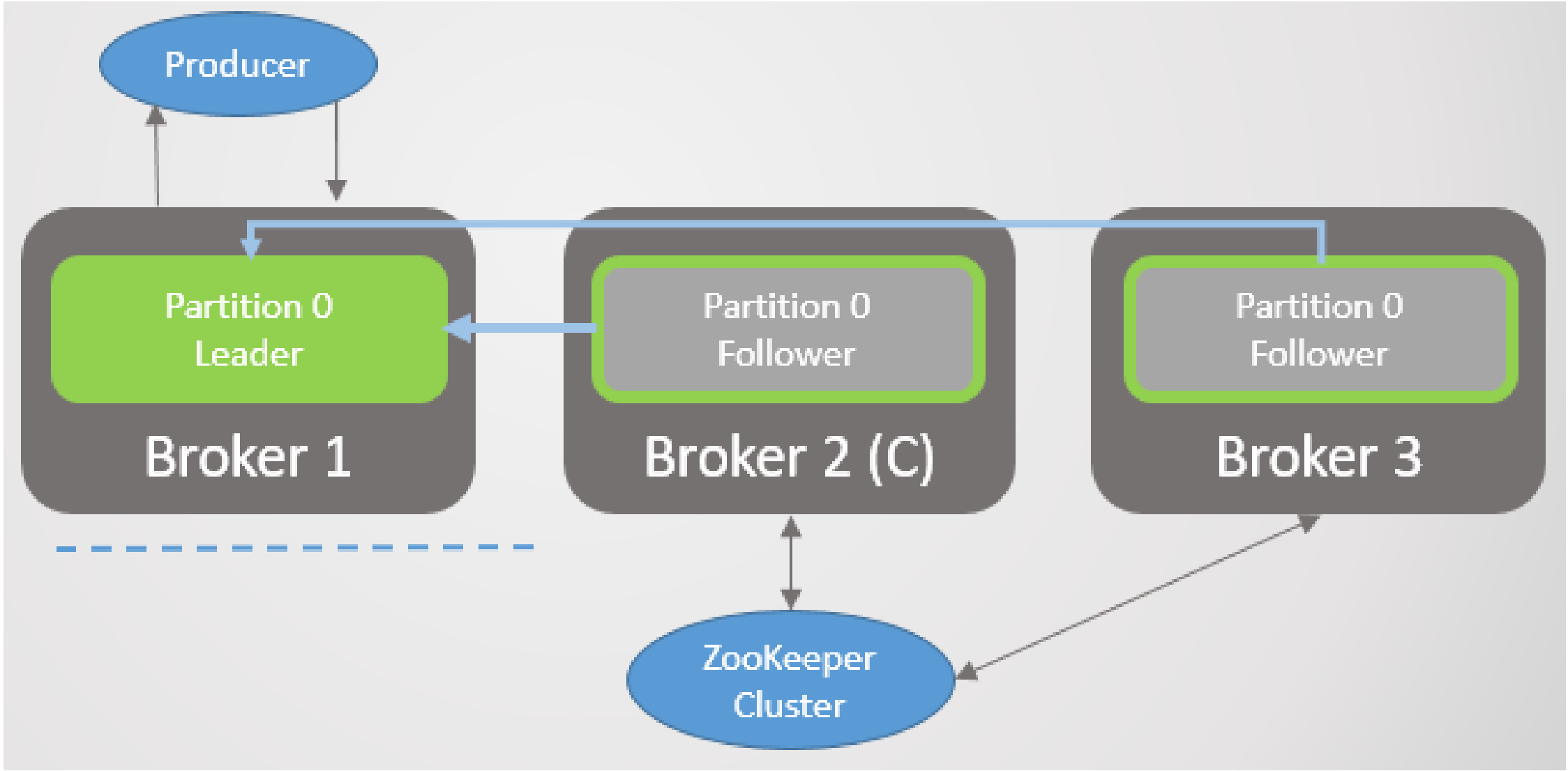
Scenario 2: A leader cannot see any of its followers, but can still see Zookeeper



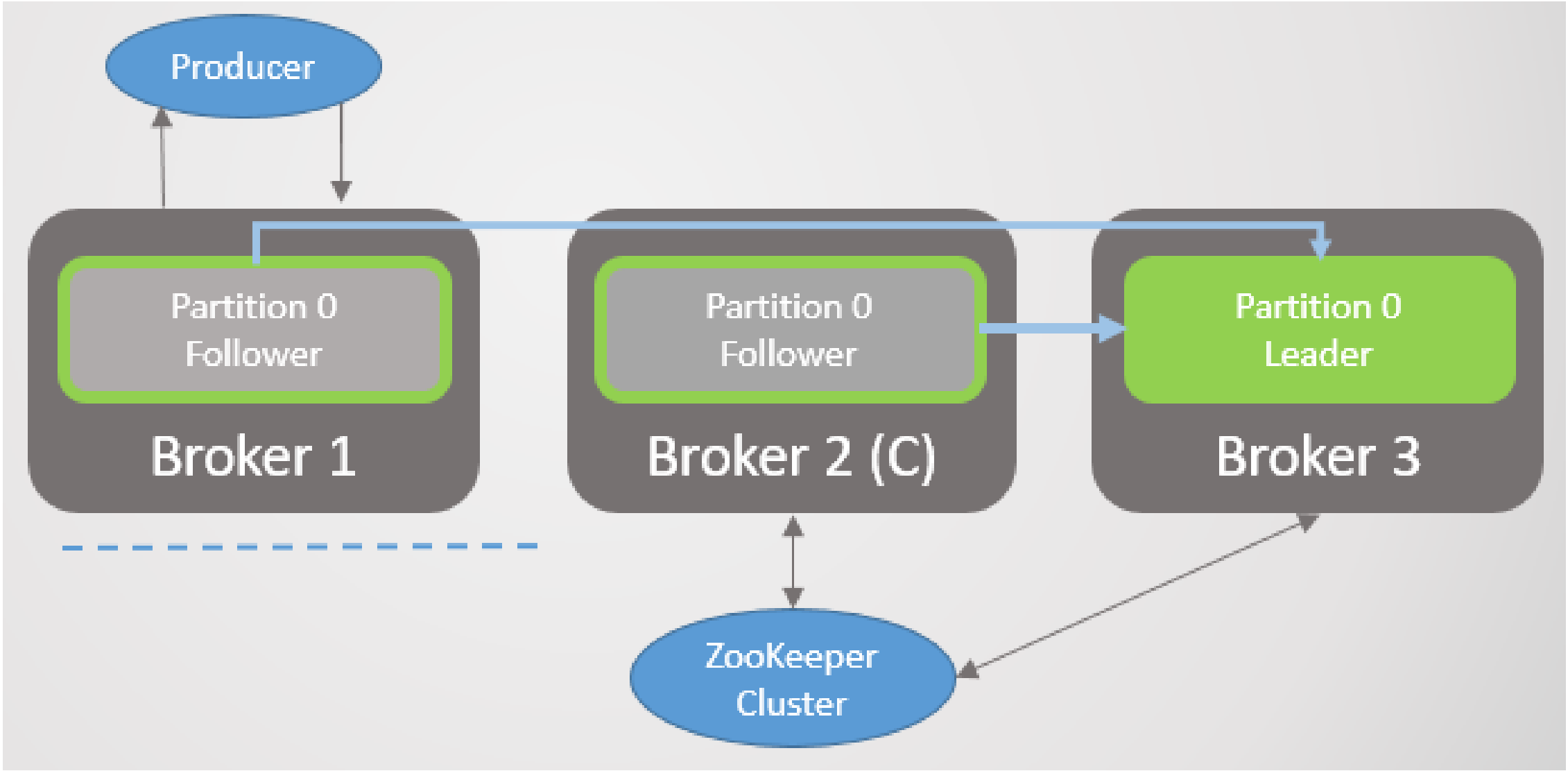
Scenario 3: A follower can see the leader, but cannot see Zookeeper



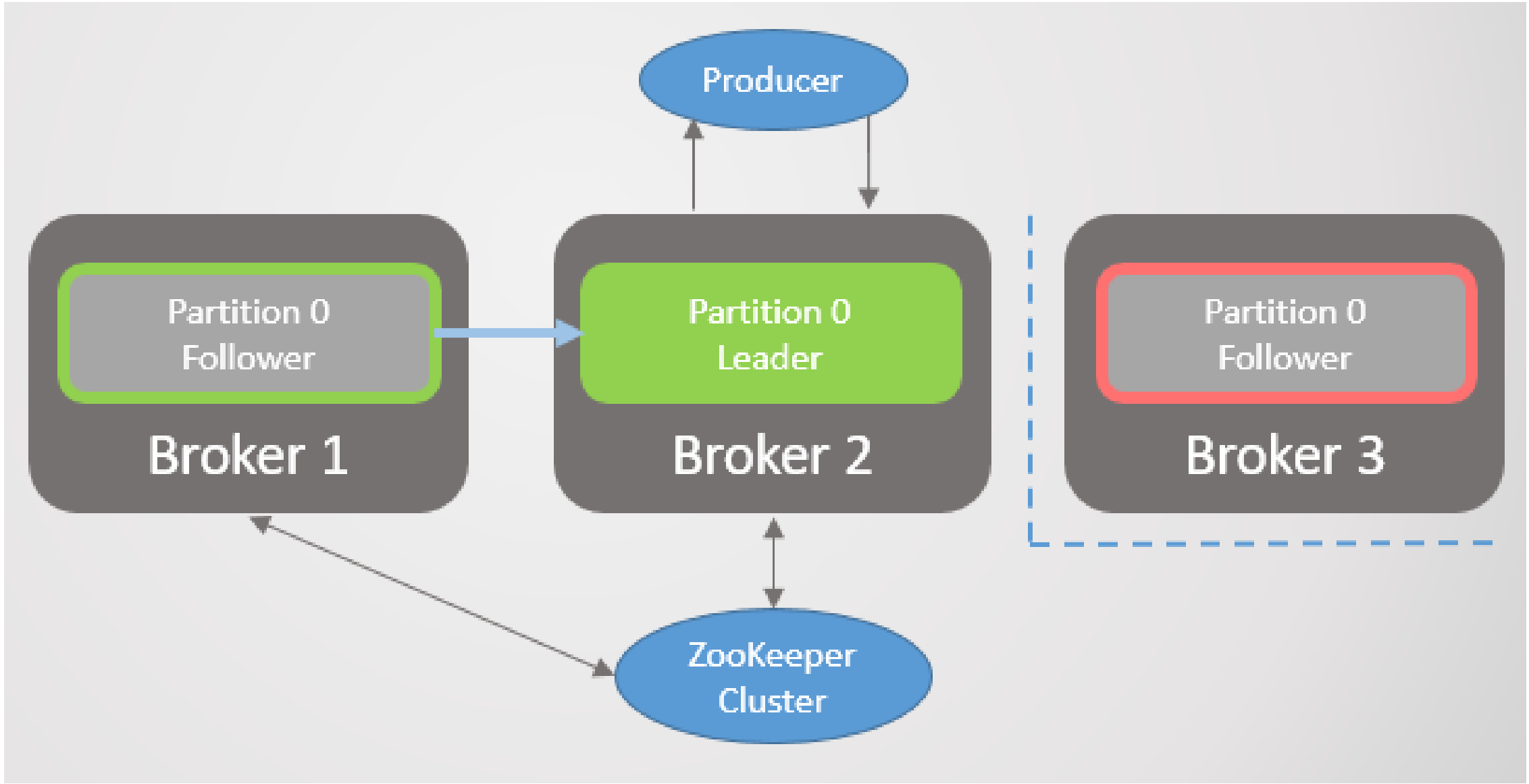
Scenario 4: A leader can see its followers, but cannot see Zookeeper



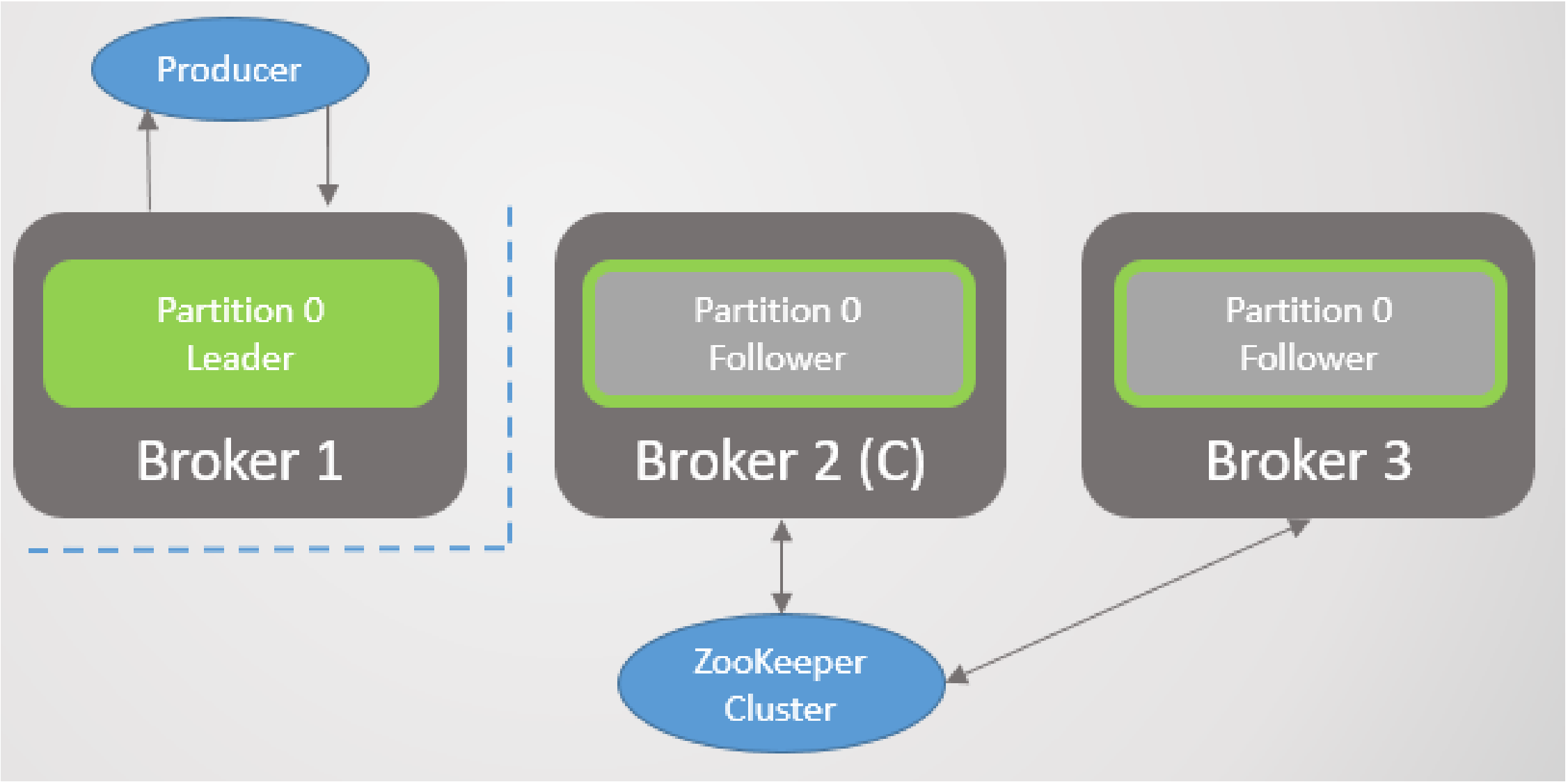
Scenario 4. The leader on Broker 1 becomes a follower after the network partition is resolved



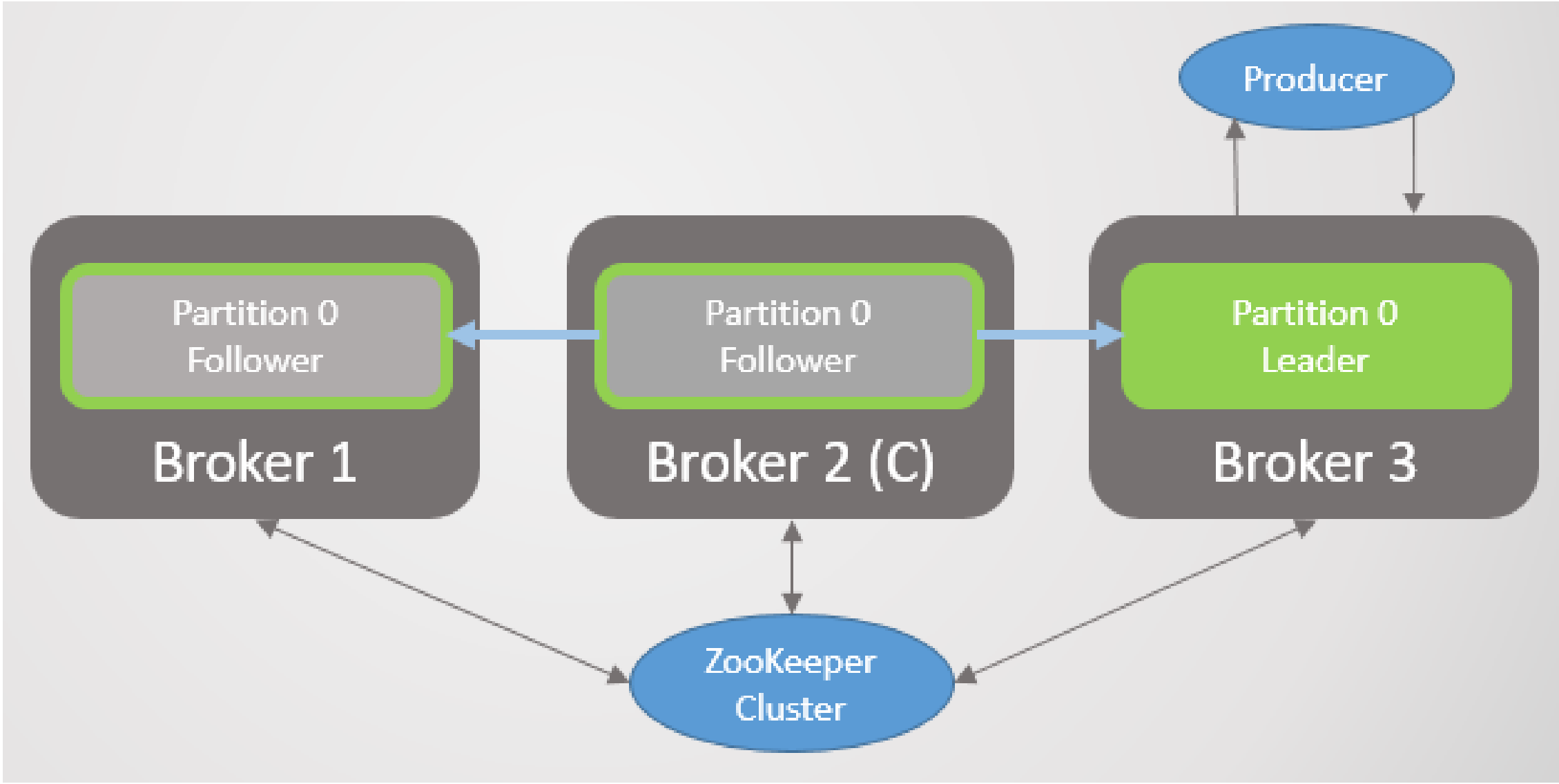
Scenario 5: A follower is completely partitioned from both the other Kafka nodes and Zookeeper



Scenario 6: A leader is completely partitioned from both the other Kafka nodes and Zookeeper



Scenario 6: The original leader becomes a follower after the network partition is resolved

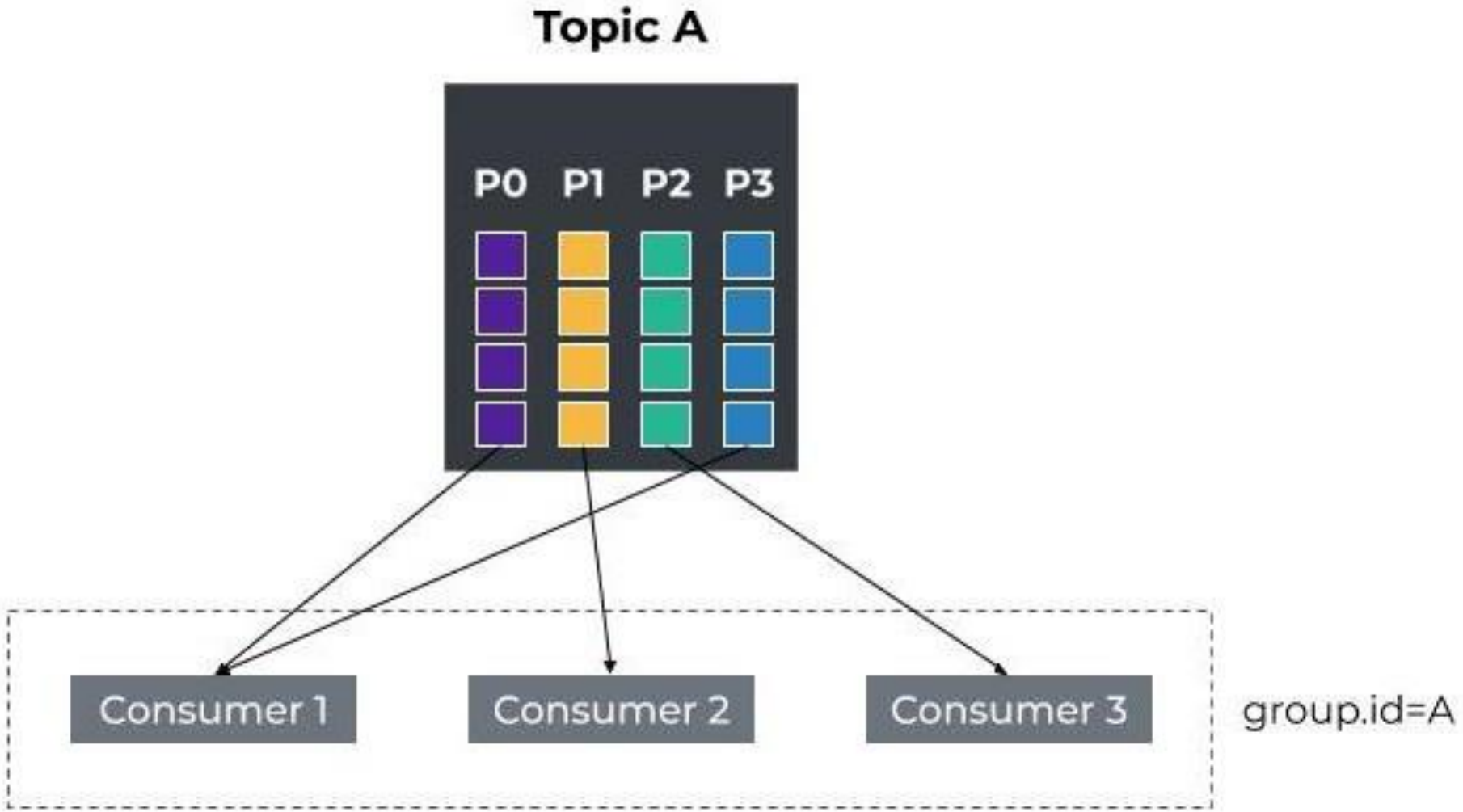


Scenarios Conclusions - Message Loss

* leader fail-over where acks=1
* unclean fail-over with acks=all
* A leader isolation from Zookeeper with acks=1
* A fully isolated leader, even acks=all if min.insync.replicas=1
* Simultaneous failures of all nodes of a partition.
    * messages are acknowledged once in memory, some messages may not yet have been written to disk.

8 - Clustering – Partition Assignment and Rebalancing

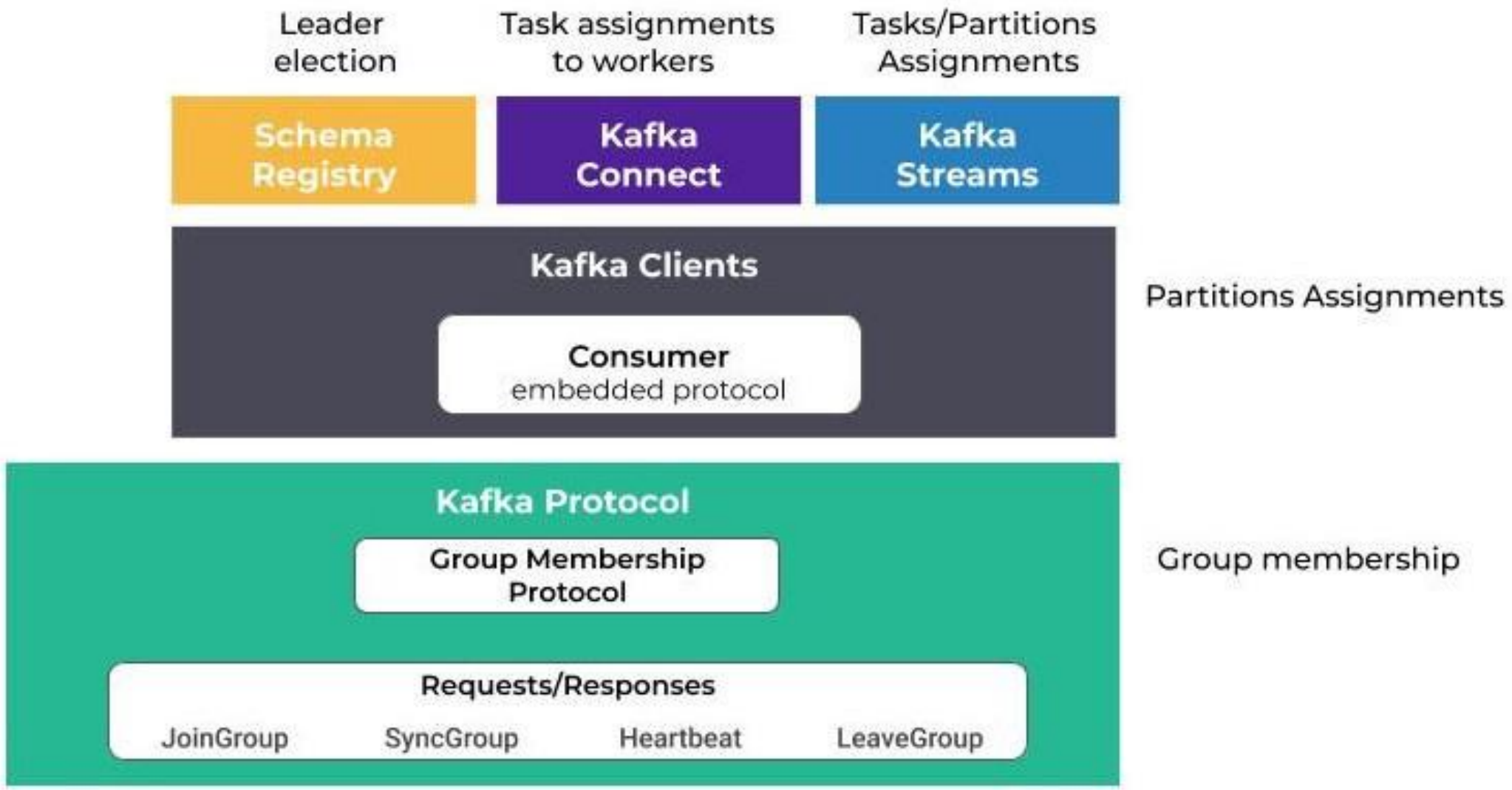
Let’s go back to some basics



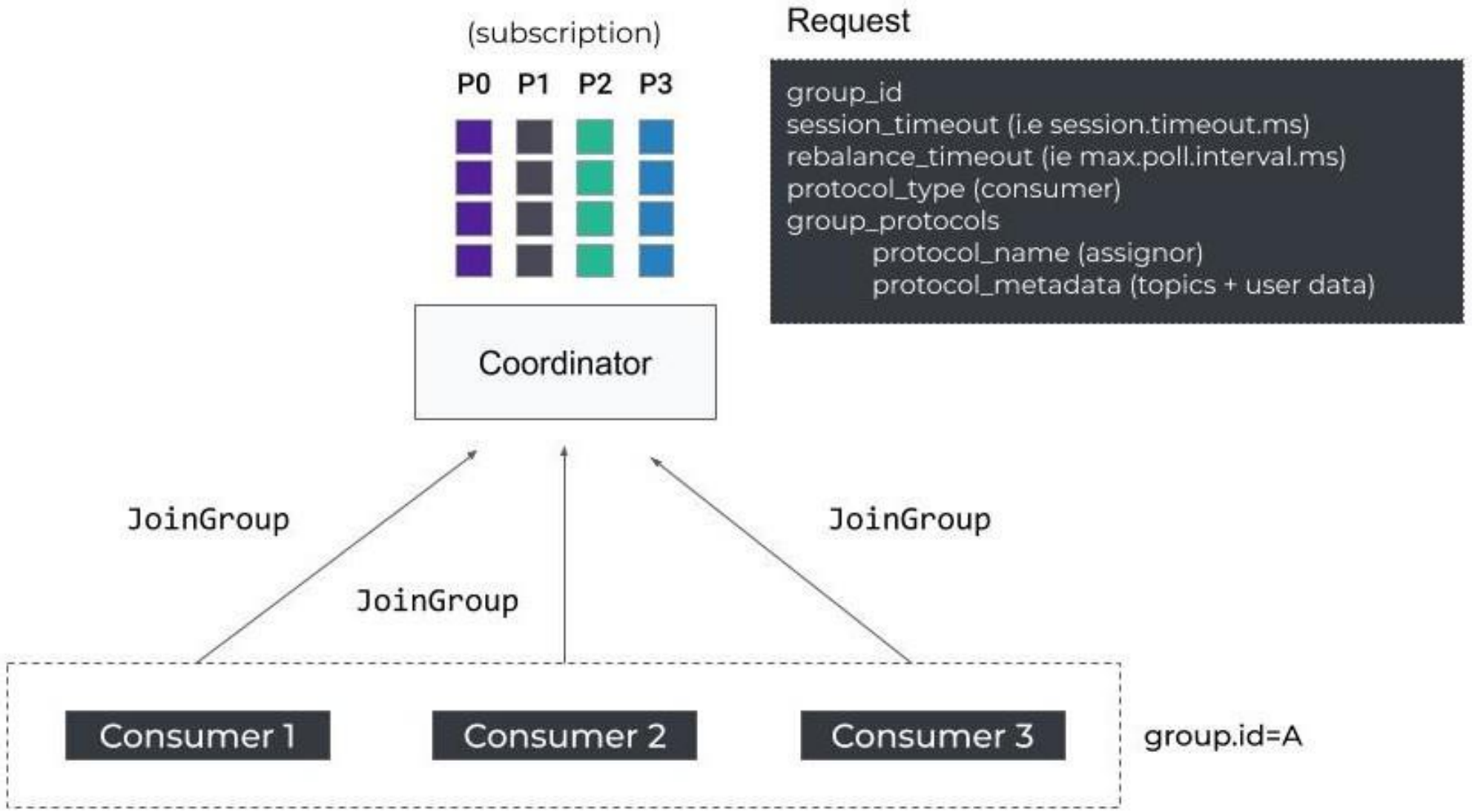
The Rebalance Protocol, in a Nutshell

Rebalance/Rebalancing: the procedure that is followed by a number of distributed processes that use Kafka clients and/or the Kafka coordinator to form a common group and distribute a set of resources among the members of the group.

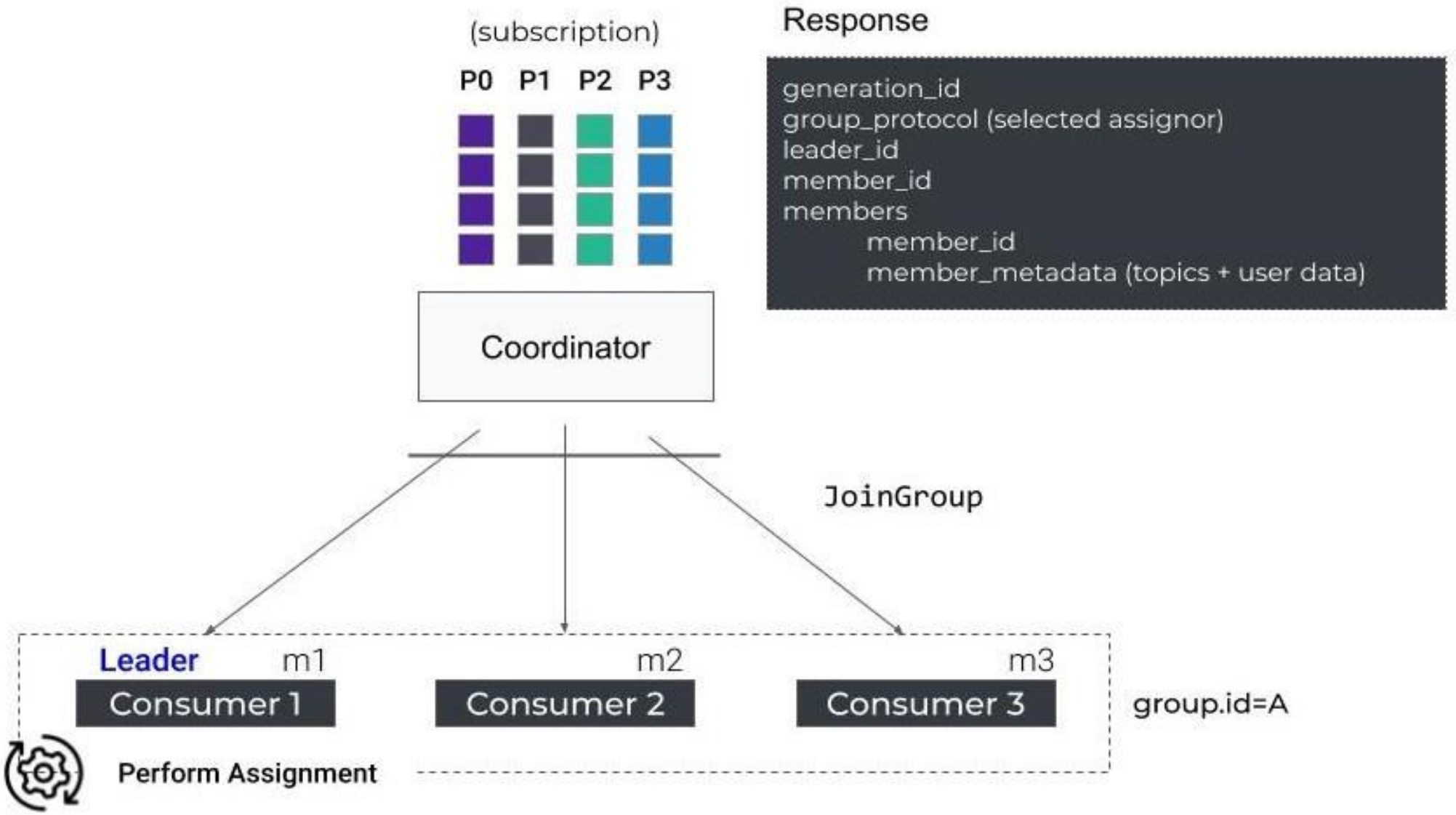
Apache Kafka Rebalance Protocol and components



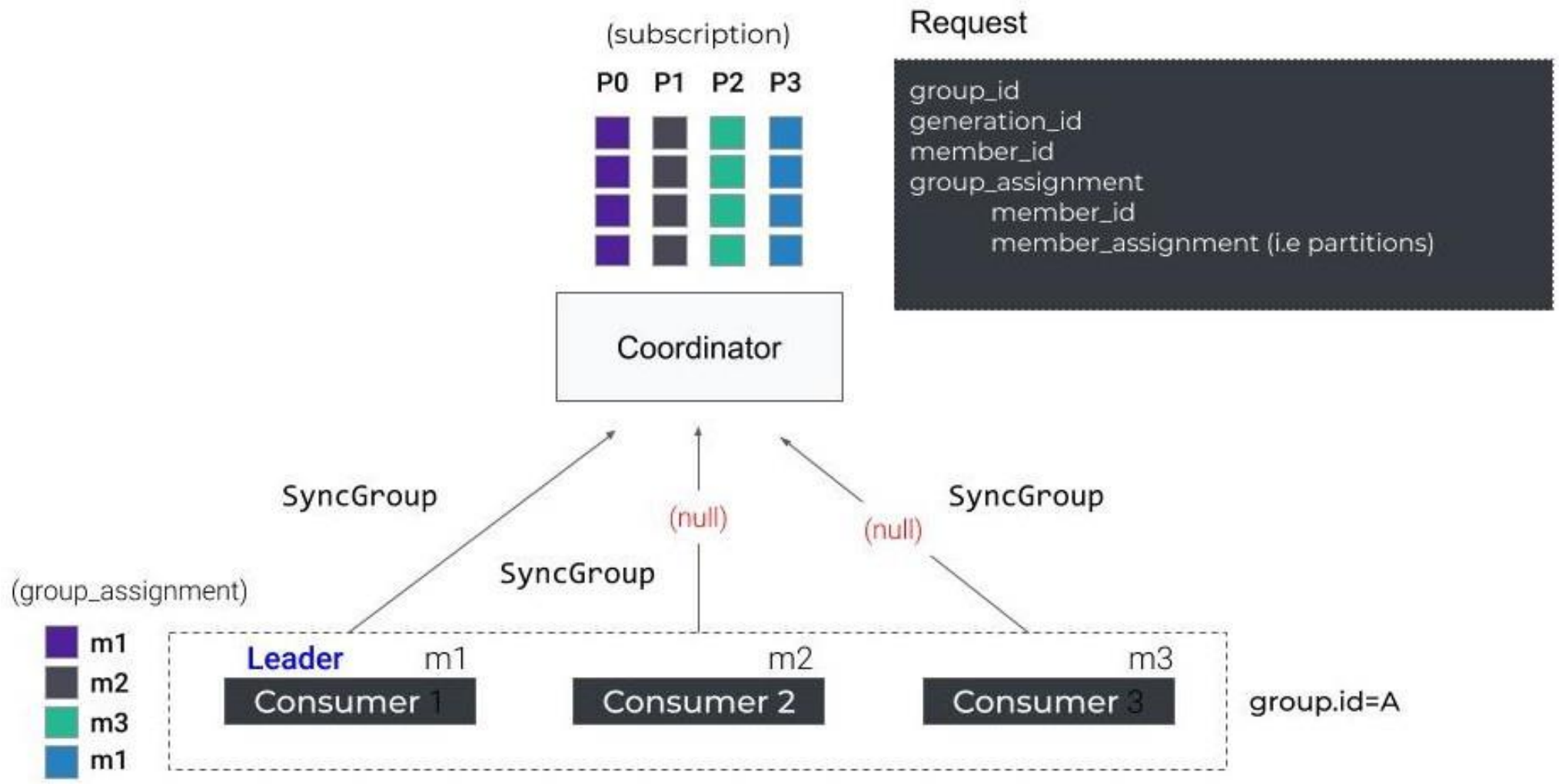
Join Group Request



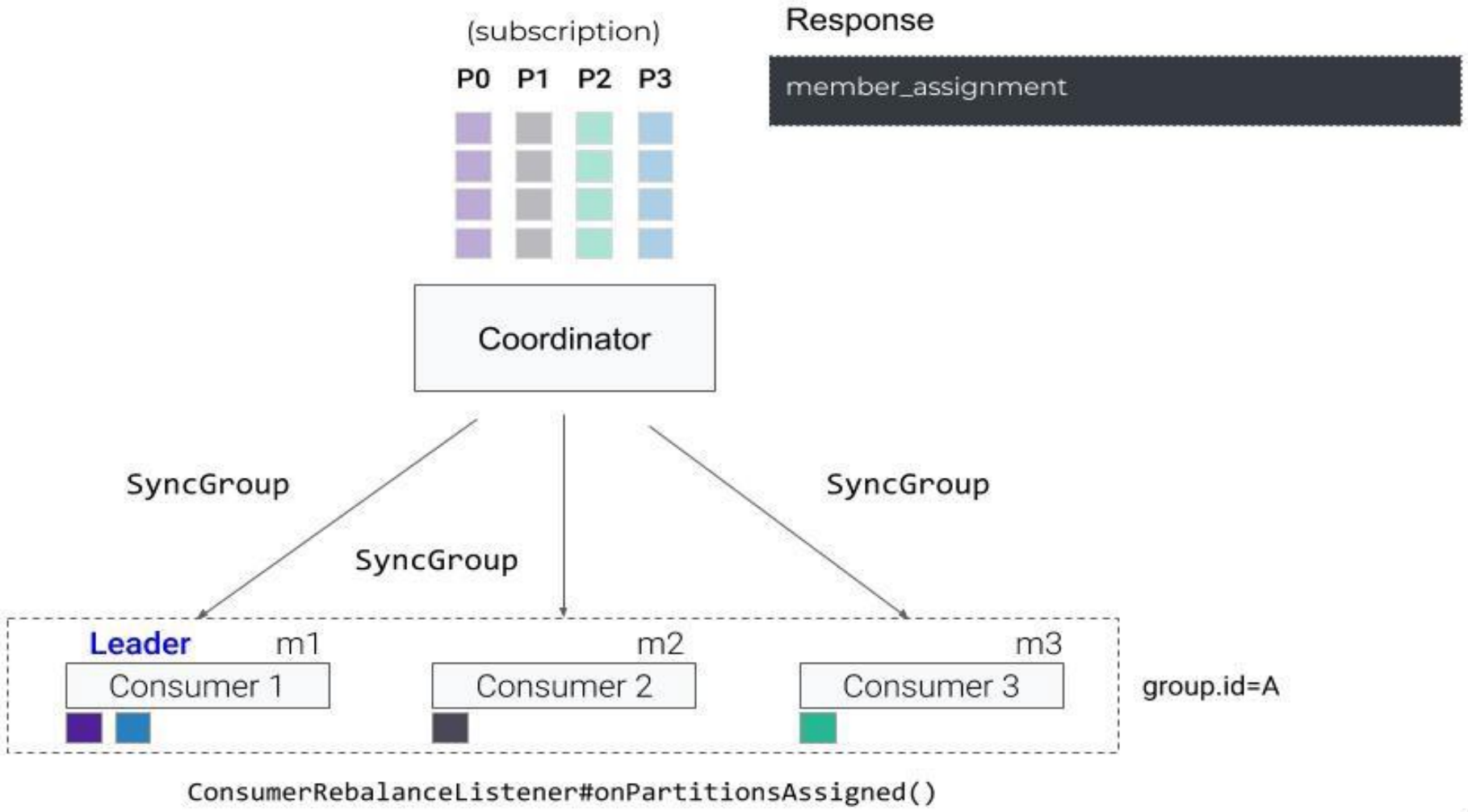
Join Group Response



Sync Group Request

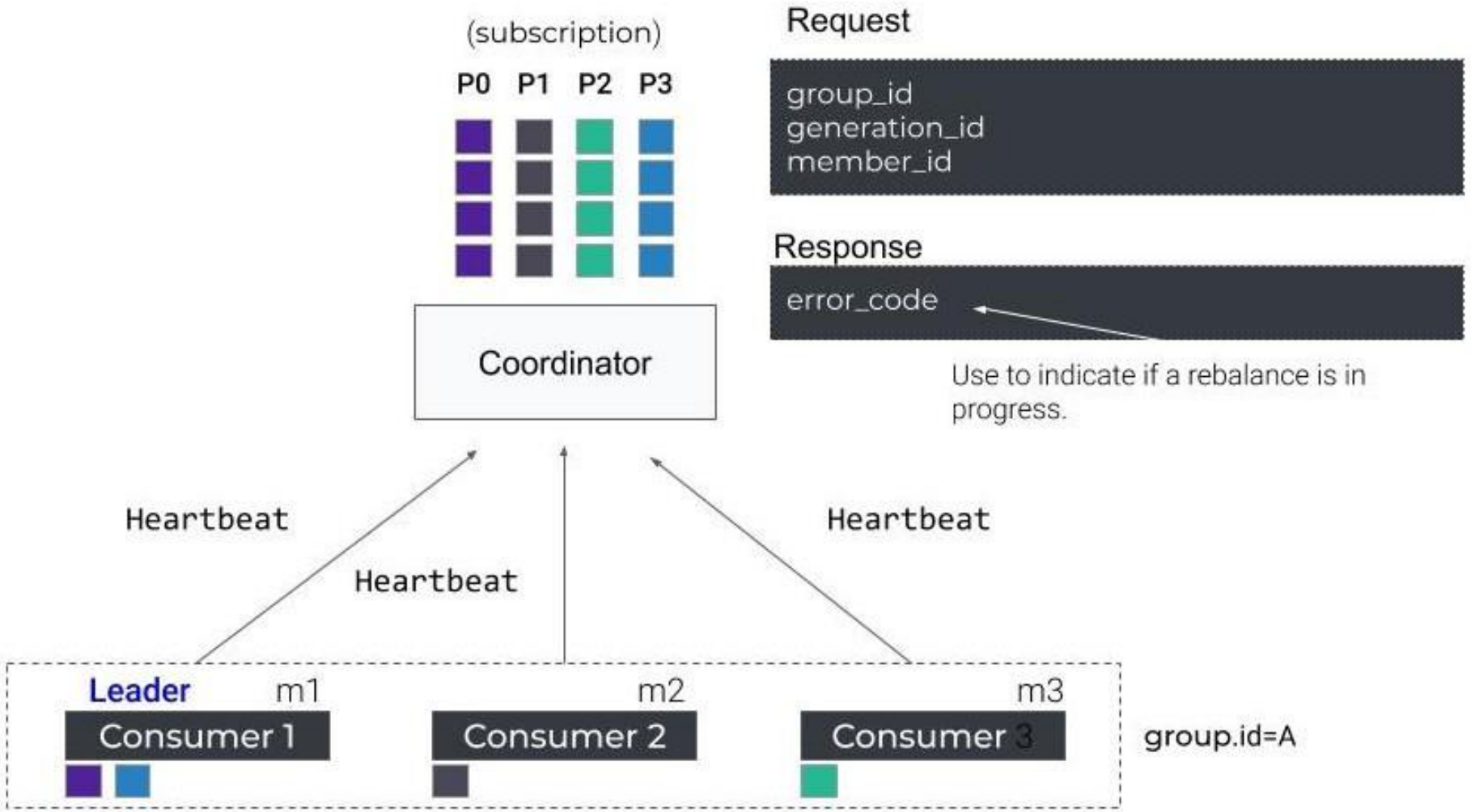


Sync Group Response

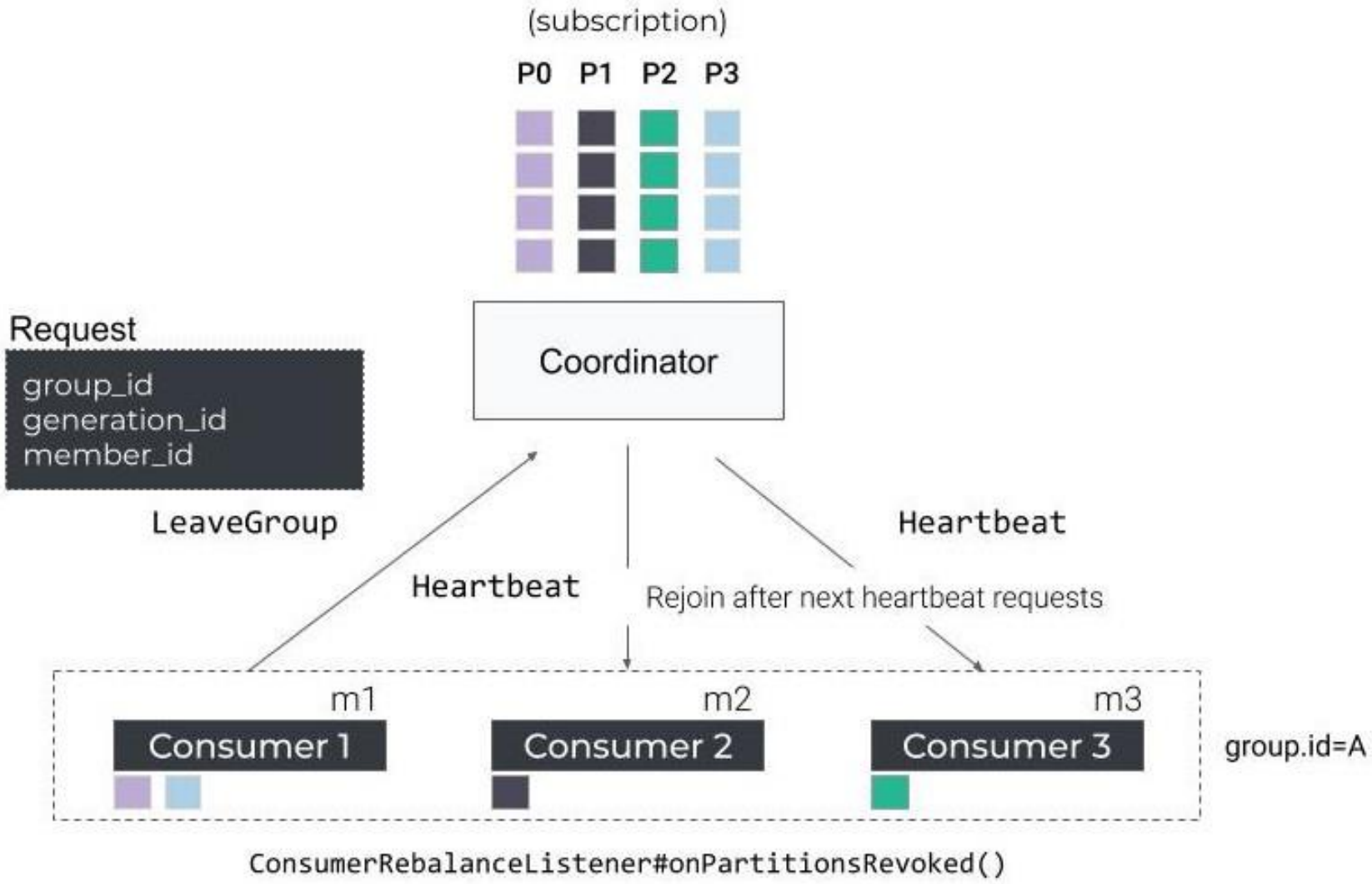


18

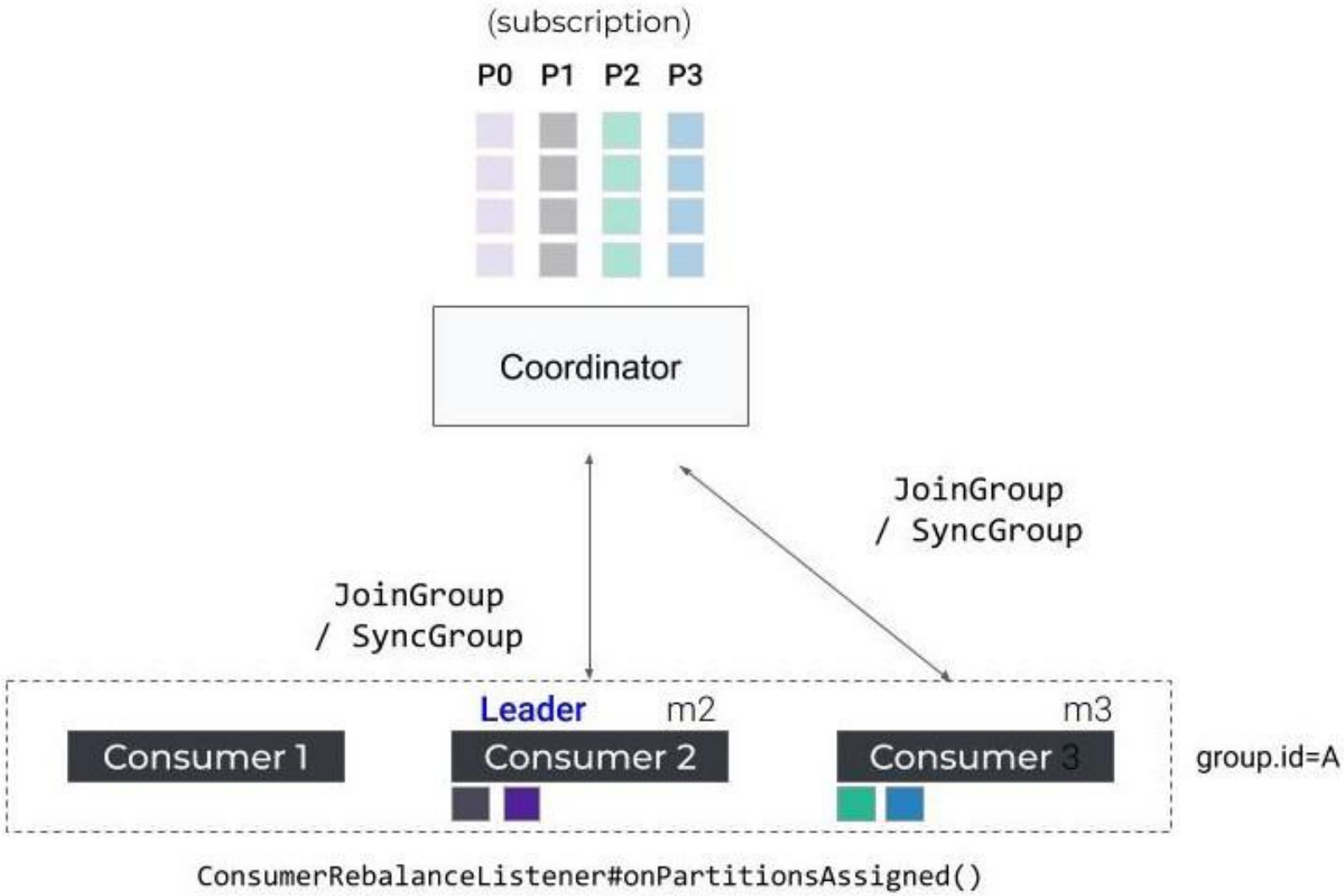
Heartbeat


(subscription)
P0 P1 P2 P3
Coordinator
Request
group_id
generation_id
member_id
Response
error_code
Use to indicate if a rebalance is in progress.
Heartbeat
Heartbeat
Heartbeat
Leader
m1
Consumer 1
m2
Consumer 2
m3
Consumer 3
group.id=A

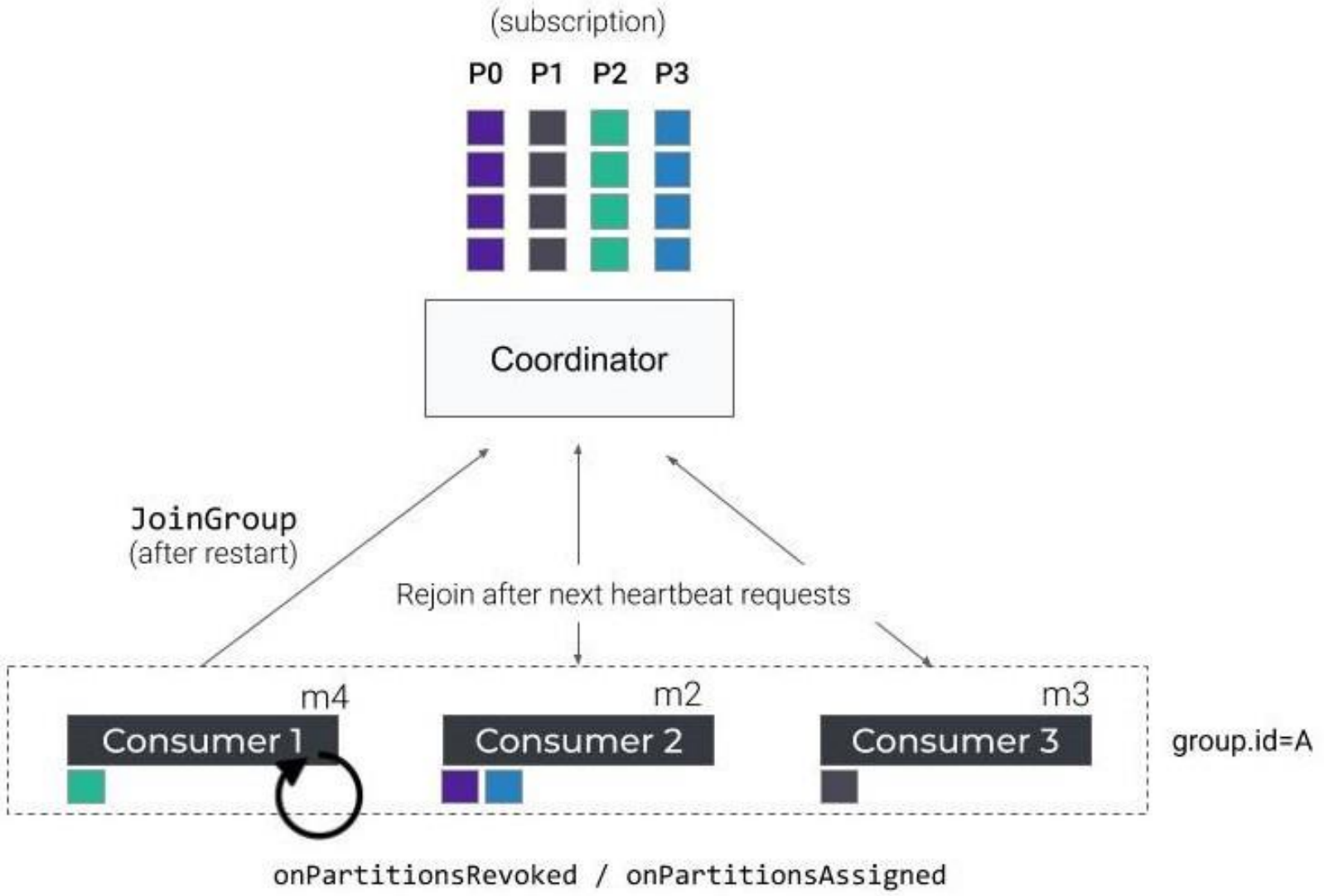
Consumer — Rebalance Protocol — Leave Group



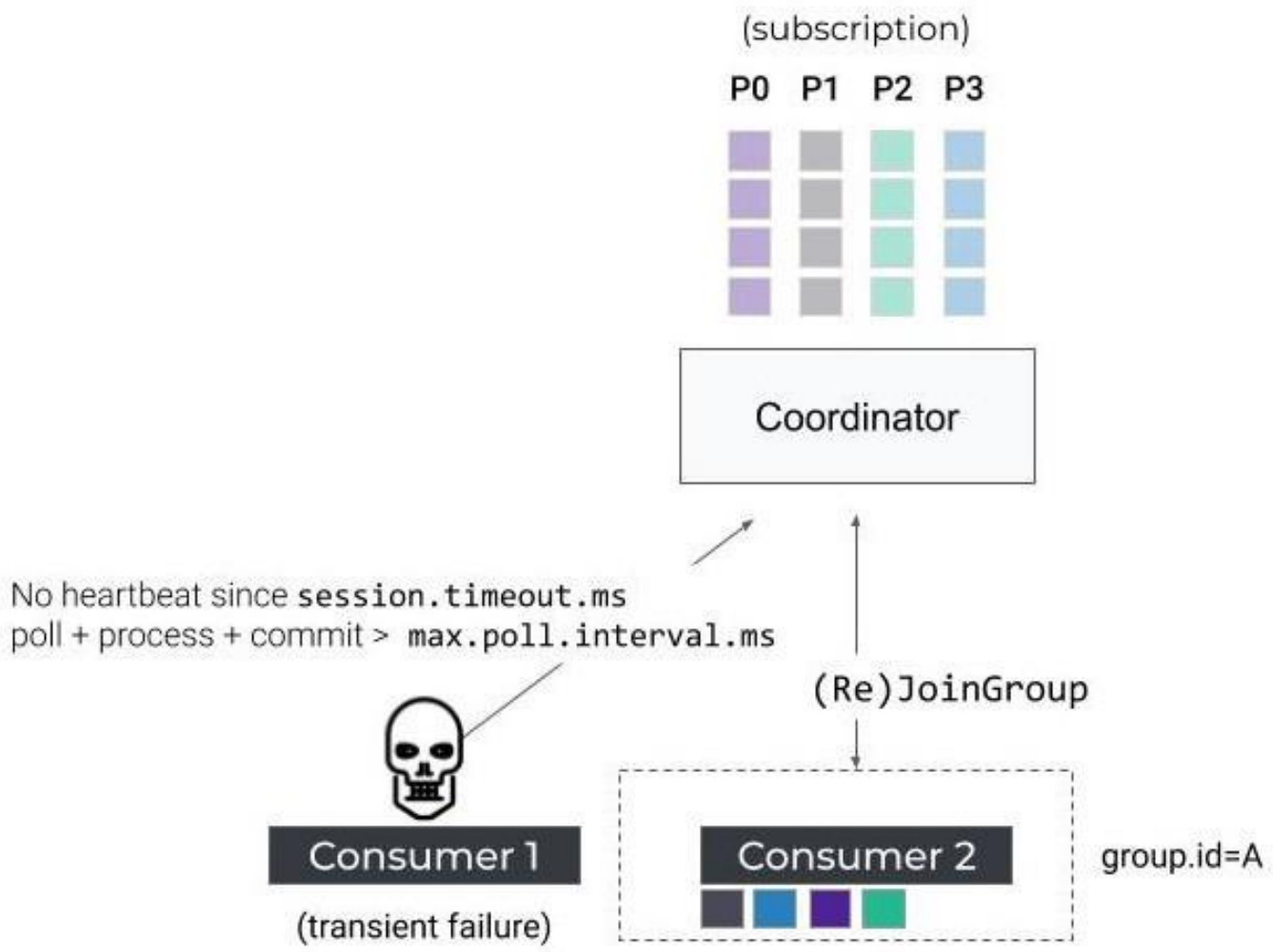
Consumer — Rebalance Protocol — Rejoin



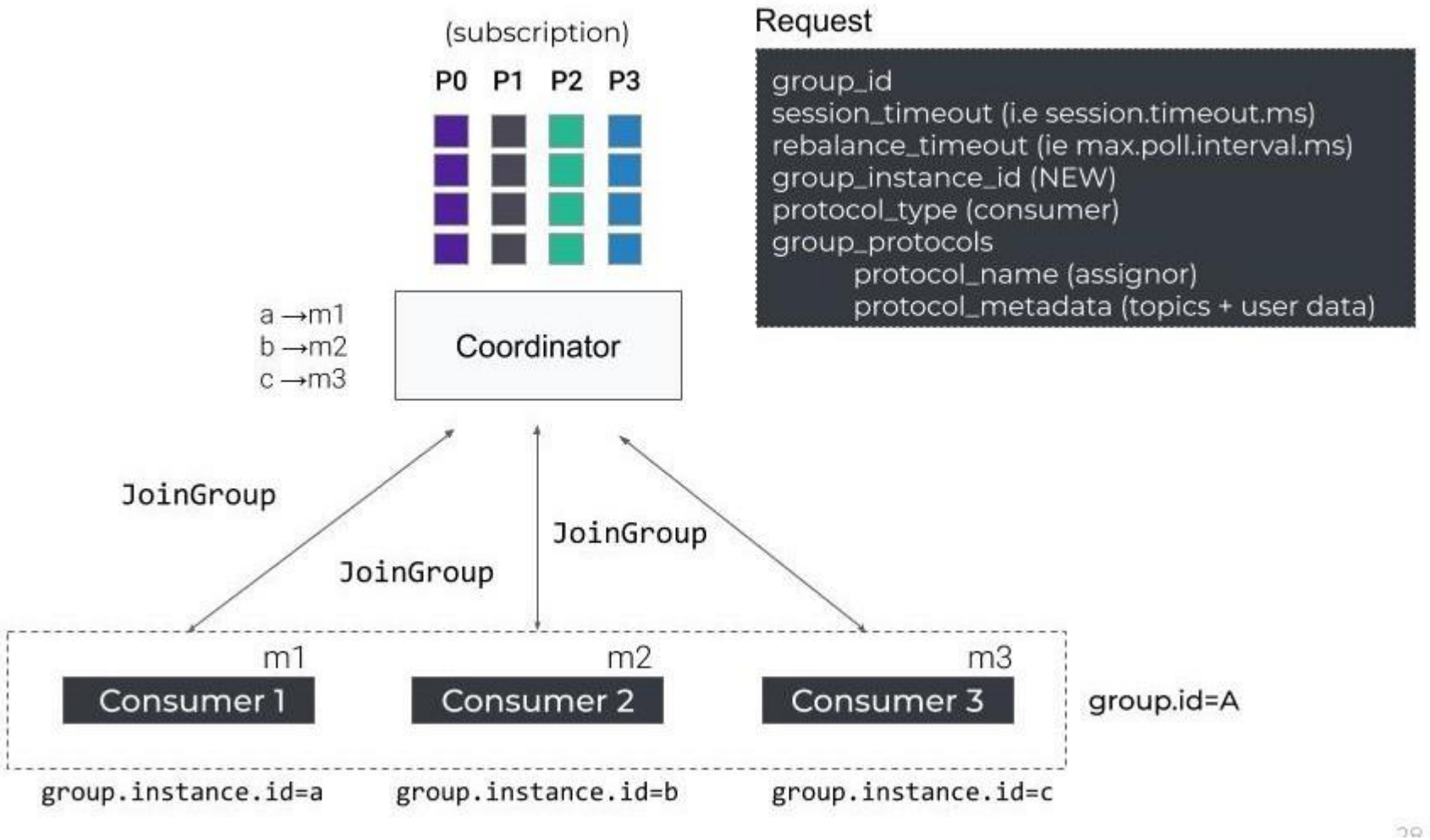
Consumer — Rebalance Protocol — Restart



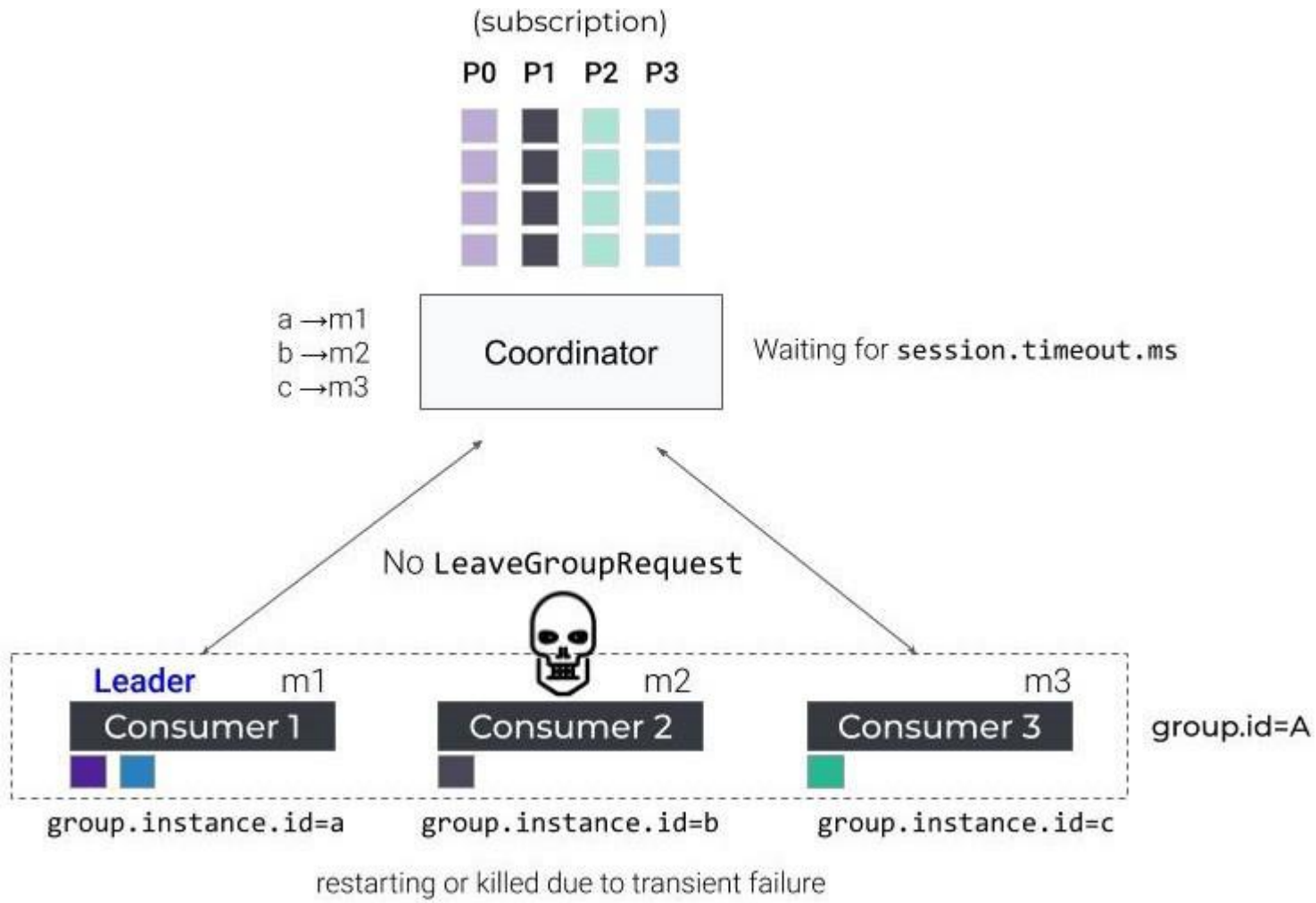
Consumer — Rebalance Protocol — Timeout



Static Membership



Static Membership - Restarting or Killed due to transient failure



Incremental Cooperative Rebalancing

* New embedded protocols
* Improve the resource availability of each member
* Minimizing stop-the-world effect
* The basic idea behind is perform rebalancing
    * Incrementally
    * In cooperation

KIP - Rebalancing Protocols

* KIP : Kafka Improvement Proposals
* KIP-345: Introduce static membership protocol to reduce consumer rebalances
    * Released: 2.4.0
    * Last modified: Sep 09 2019
* KIP-429: Kafka Consumer Incremental Rebalance Protocol
    * Released: 2.4.0
    * Last modified: Mar 16 2021

The Partition Assignor Strategies

* What is it?

  Protocol to customize how partitions are assigned to the group members.

* How to configure?

  Configurable through the consumer property

  `partition.assignment.strategy`

* Built-in
  * Range
  * RoundRobin
  * StickyAssignor

Specify a Partition Assignor - code snippet

```
Properties props = new Properties();

...

props.put(ConsumerConfig.PARTITION_ASSIGNMENT_STRATEGY_CONFIG,
          StickyAssignor.class.getName());


KafkaConsumer<String, String> consumer =
                                  new KafkaConsumer<>(props);

//...
```

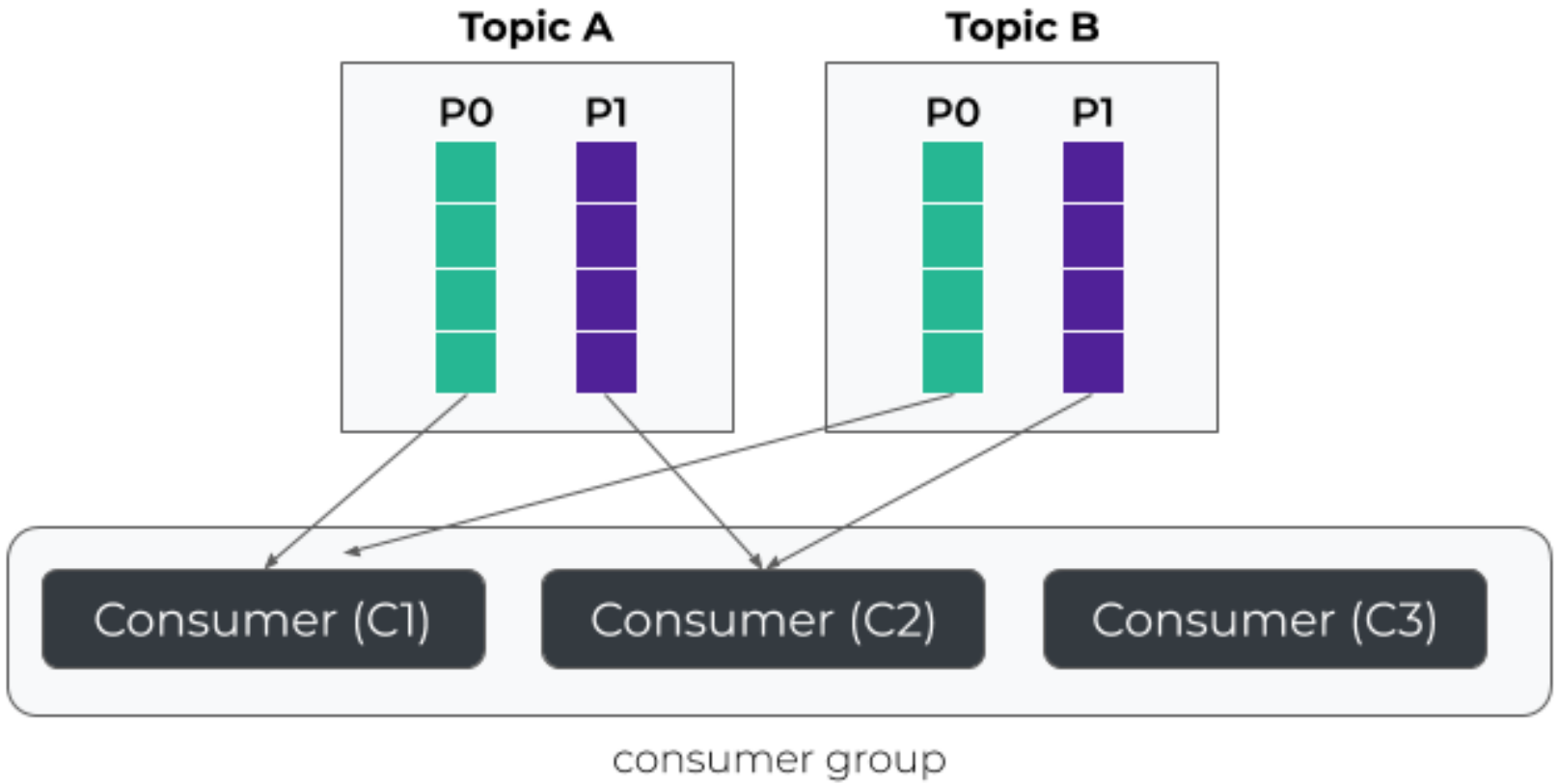
Different strategy Issue

* All consumers which belong to the same group must have one common strategy declared

* An assignment configuration inconsistent with other group members

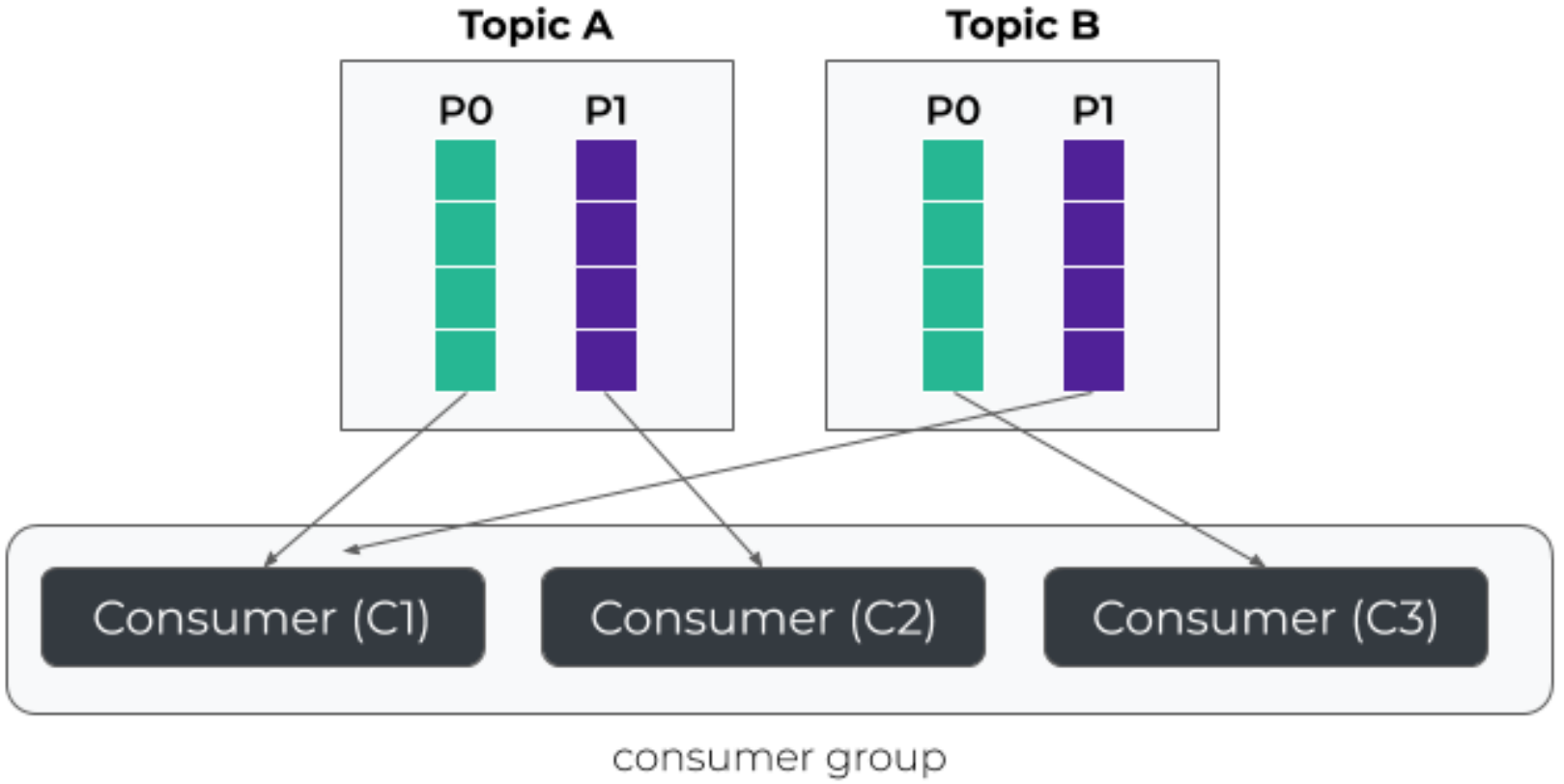
```
org.apache.kafka.common.errors.InconsistentGroupProtocolException: The
group member’s supported protocols are incompatible with those of
existing members or first group member tried to join with empty
protocol type or empty protocol list.
```

Range Assignor



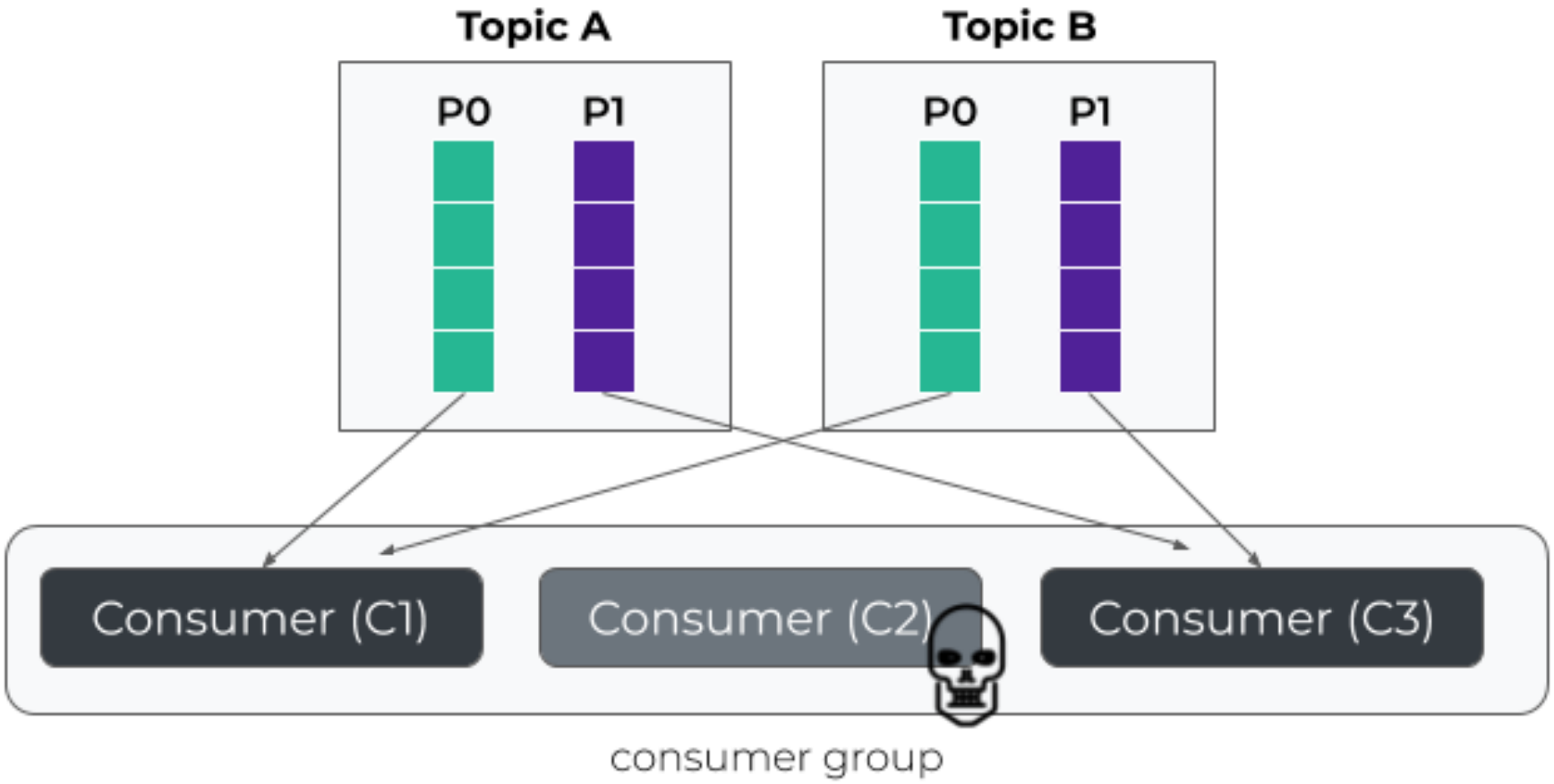
Assignment —> C1 = {A-0, B-0}, C2 = {A-1, B-1}, C3 = {}

Round Robin Assignor



Assignment —> C1 = {A-0, B-1}, C2 = {A-1}, C3 = {B-0}

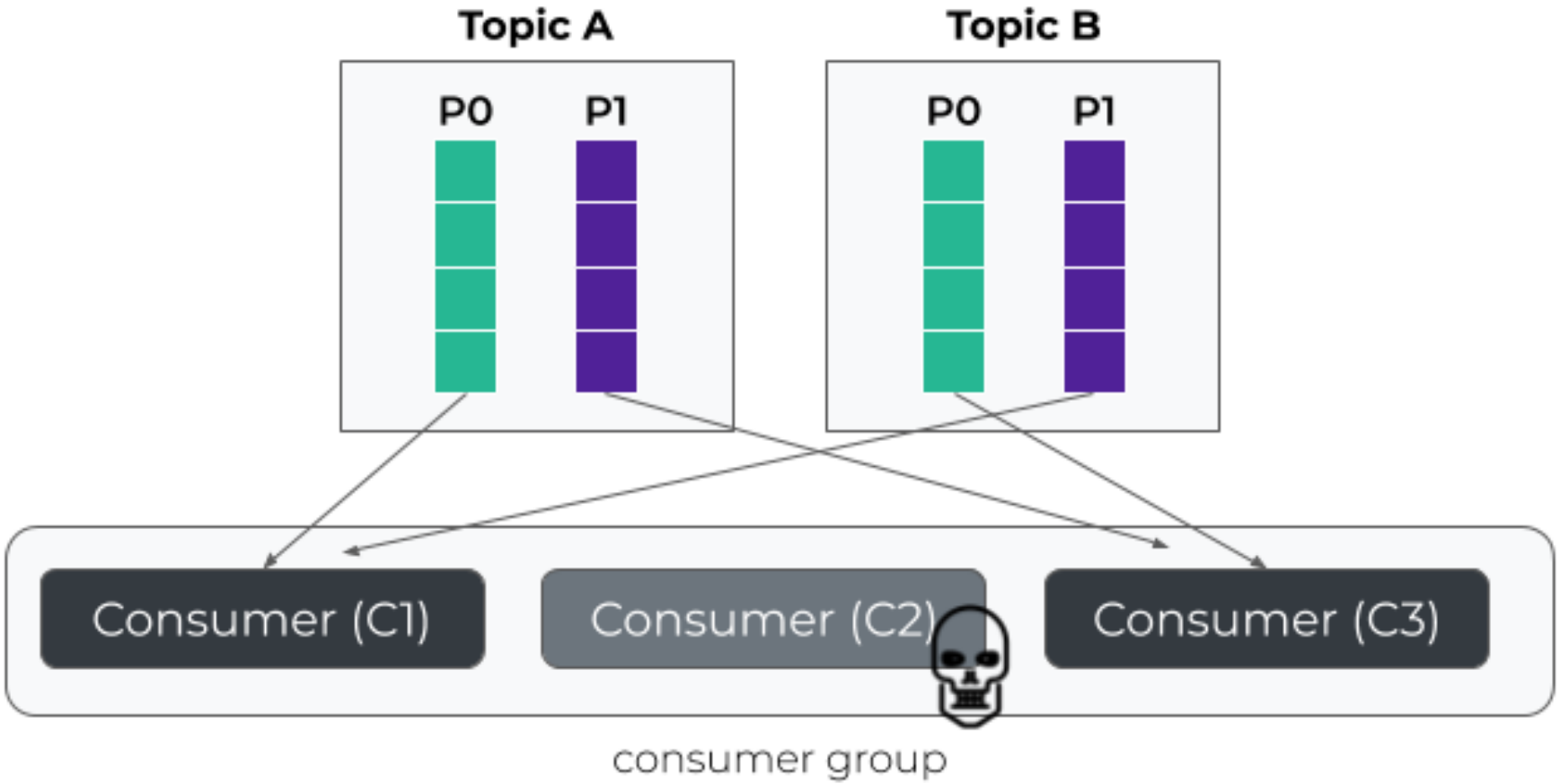
Round Robin Assignor with reassignment



before —> C1 = {A-0, B-1}, C2 = {A-1}, C3 = {B-0}

after —> C1 = {A-0, B-0}, C3 = {A-1, B-1}

Sticky Assignor

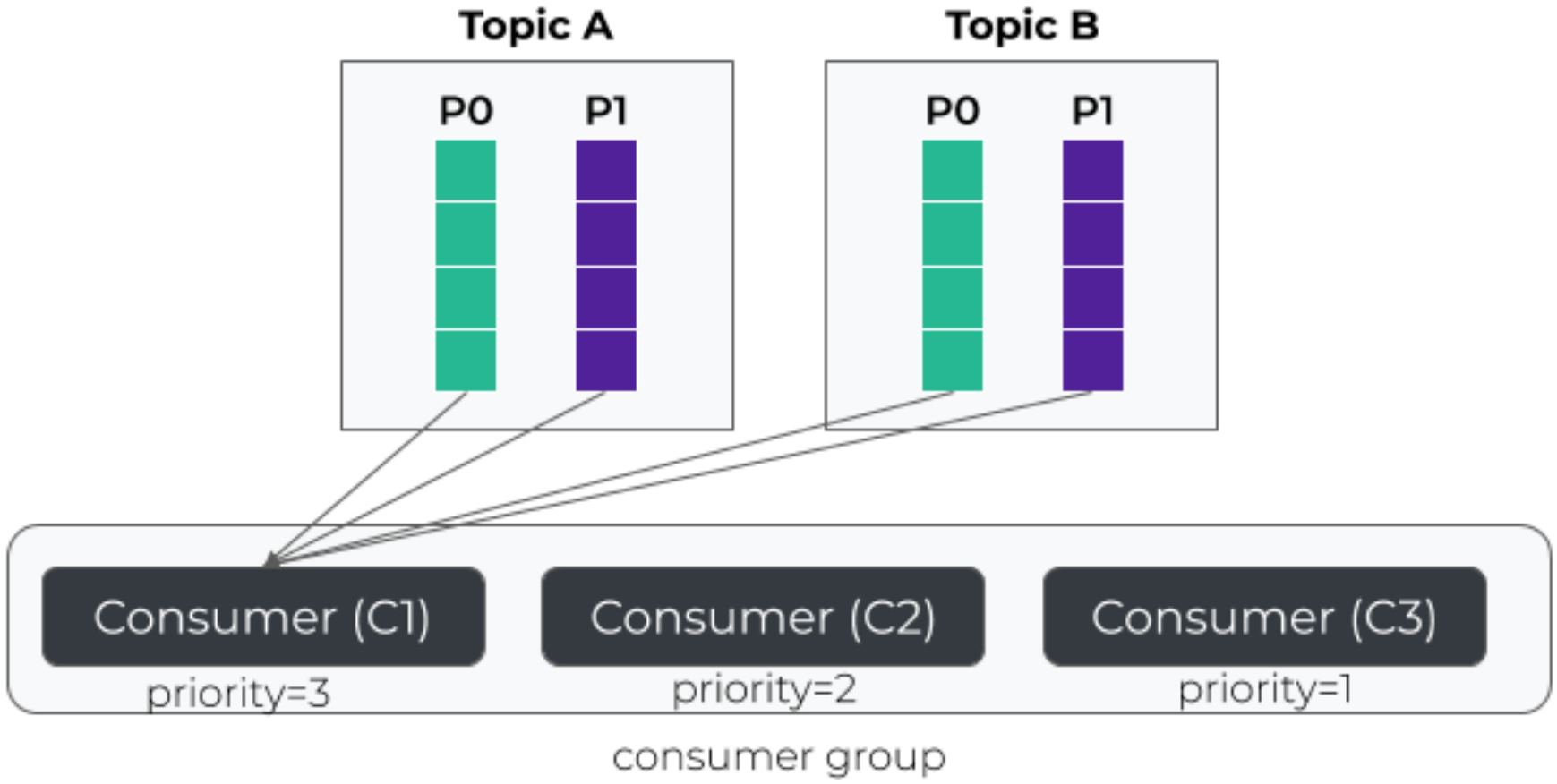


Assignment —> C1 = {A-0, B-1}, C2 = {A-1}, C3 = {B-0}

Implementing a Custom Strategy

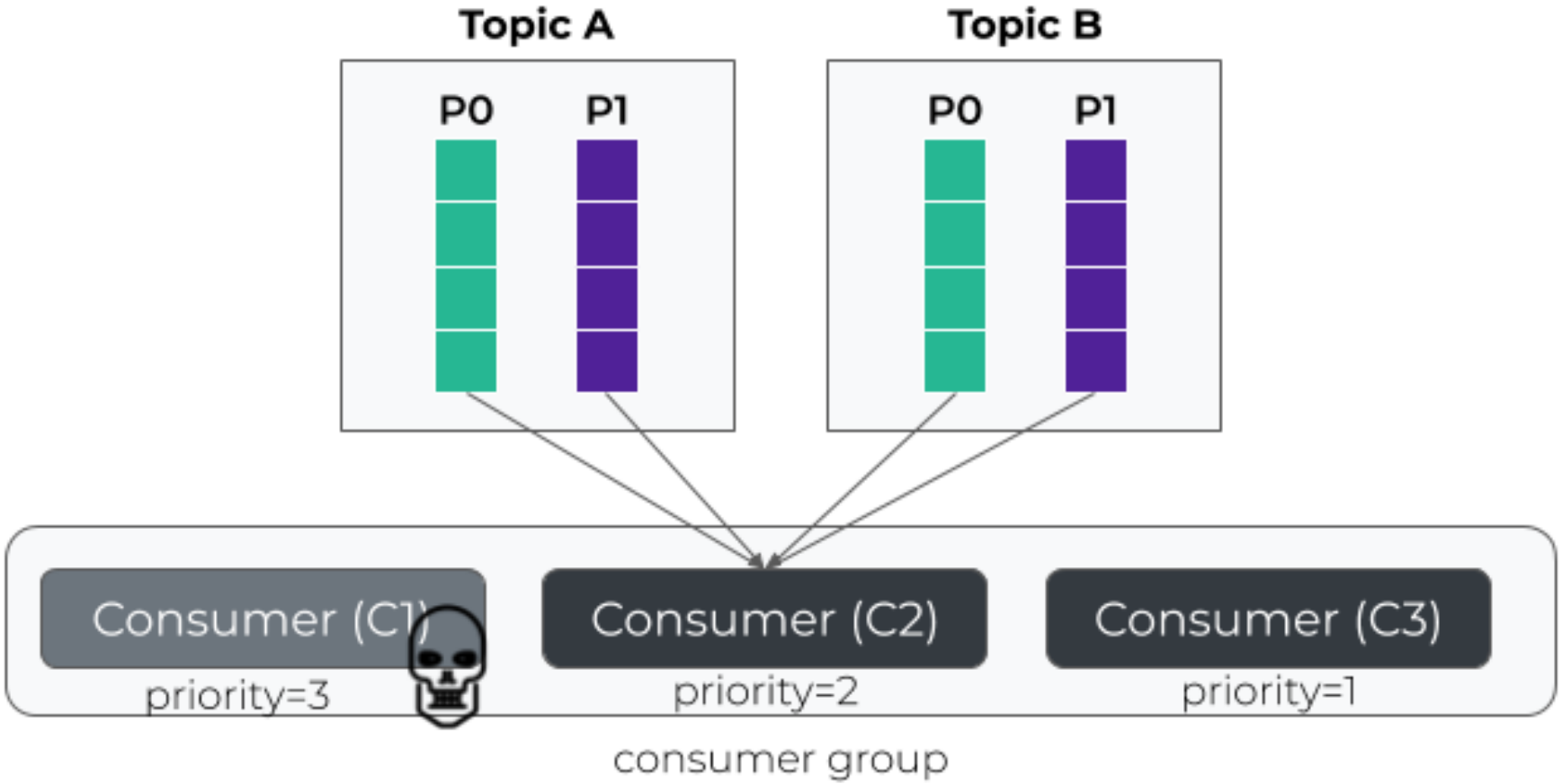
```
public interface PartitionAssignor {

    Subscription subscription(Set<String> topics);

    Map<String, Assignment> assign(
            Cluster metadata,
            Map<String, Subscription> subscriptions);

    void onAssignment(Assignment assignment);

    String name();
}
```

Simple Failover strategy



Assignment —> C1 = {A-0, A-1, B-0, B-1}, C2 = {}, C3 = {}

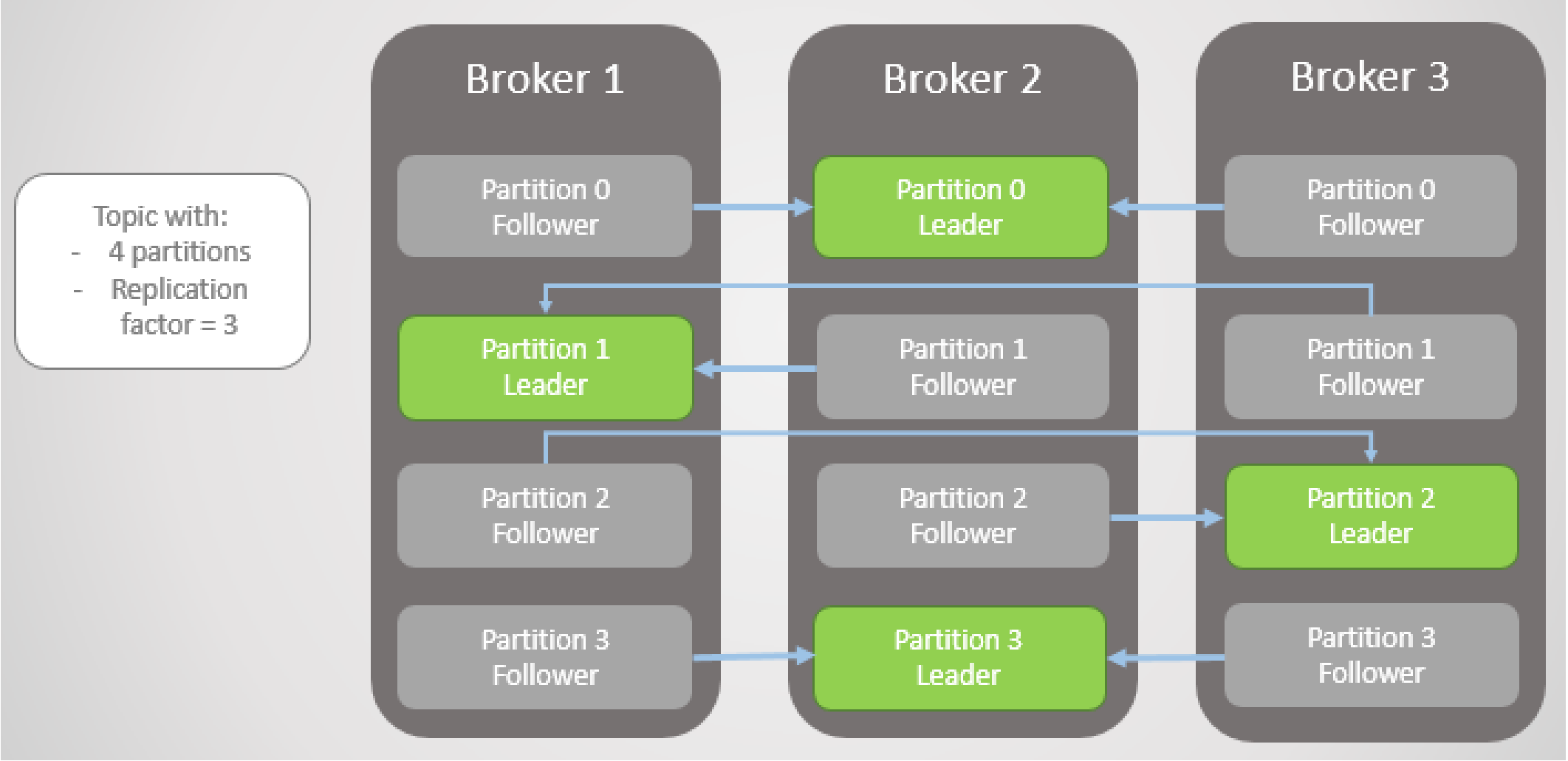
Simple Failover strategy – C1 fails



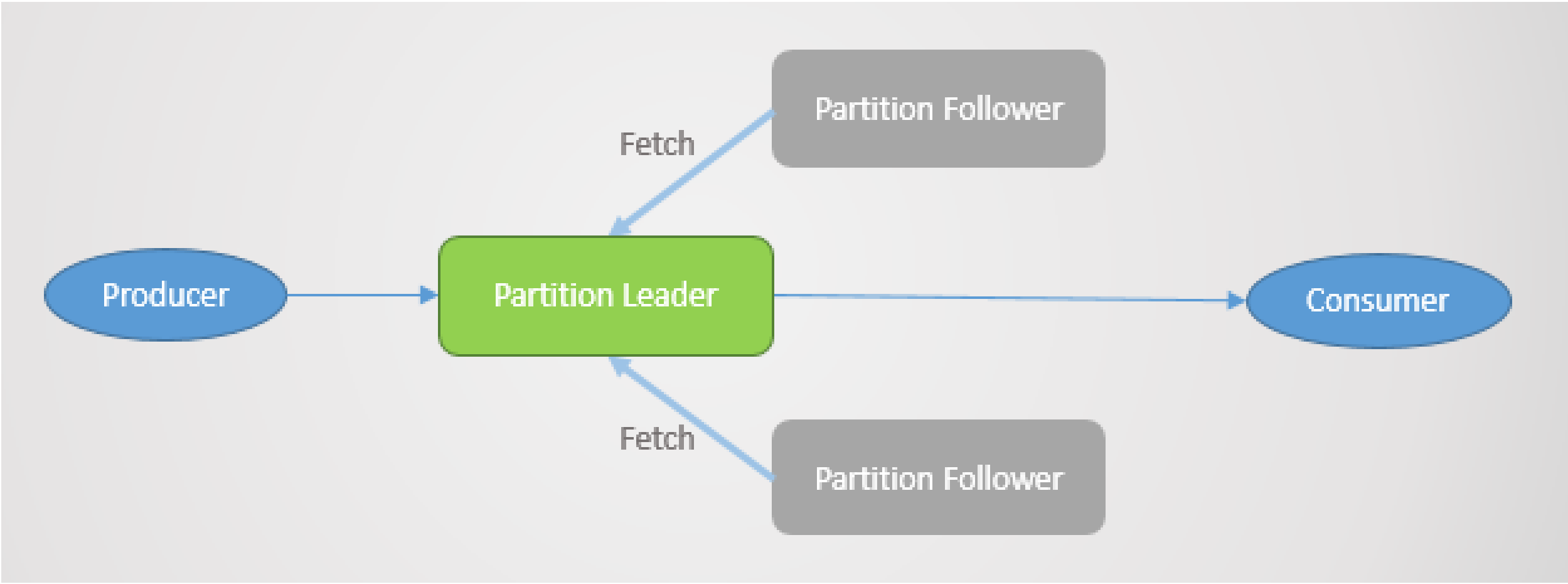
Assignment —> C2 = {A-0, A-1, B-0, B-1}, C3 = {}

9 - Clustering - Replication, Leaders and Followers

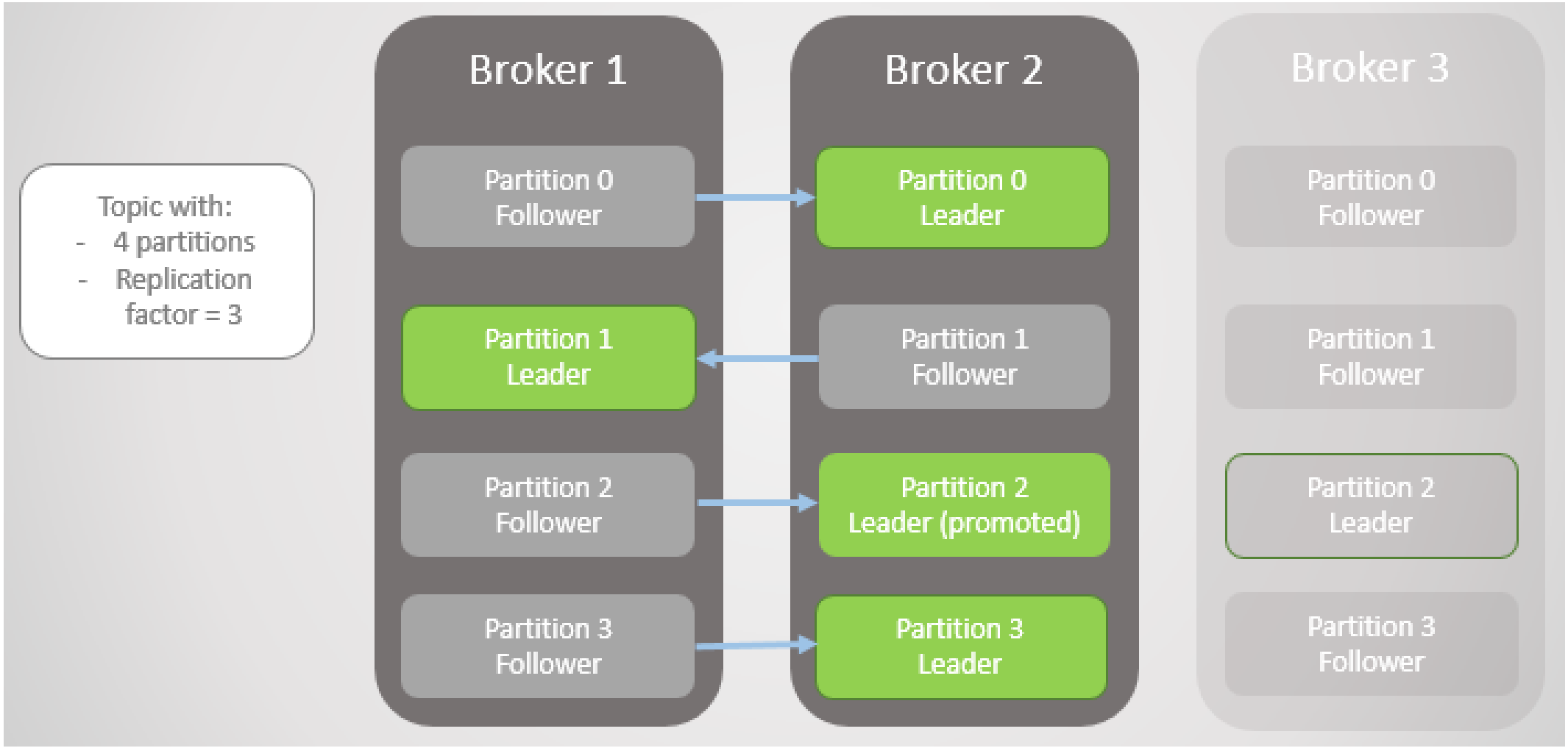
Four partitions distributed across three brokers



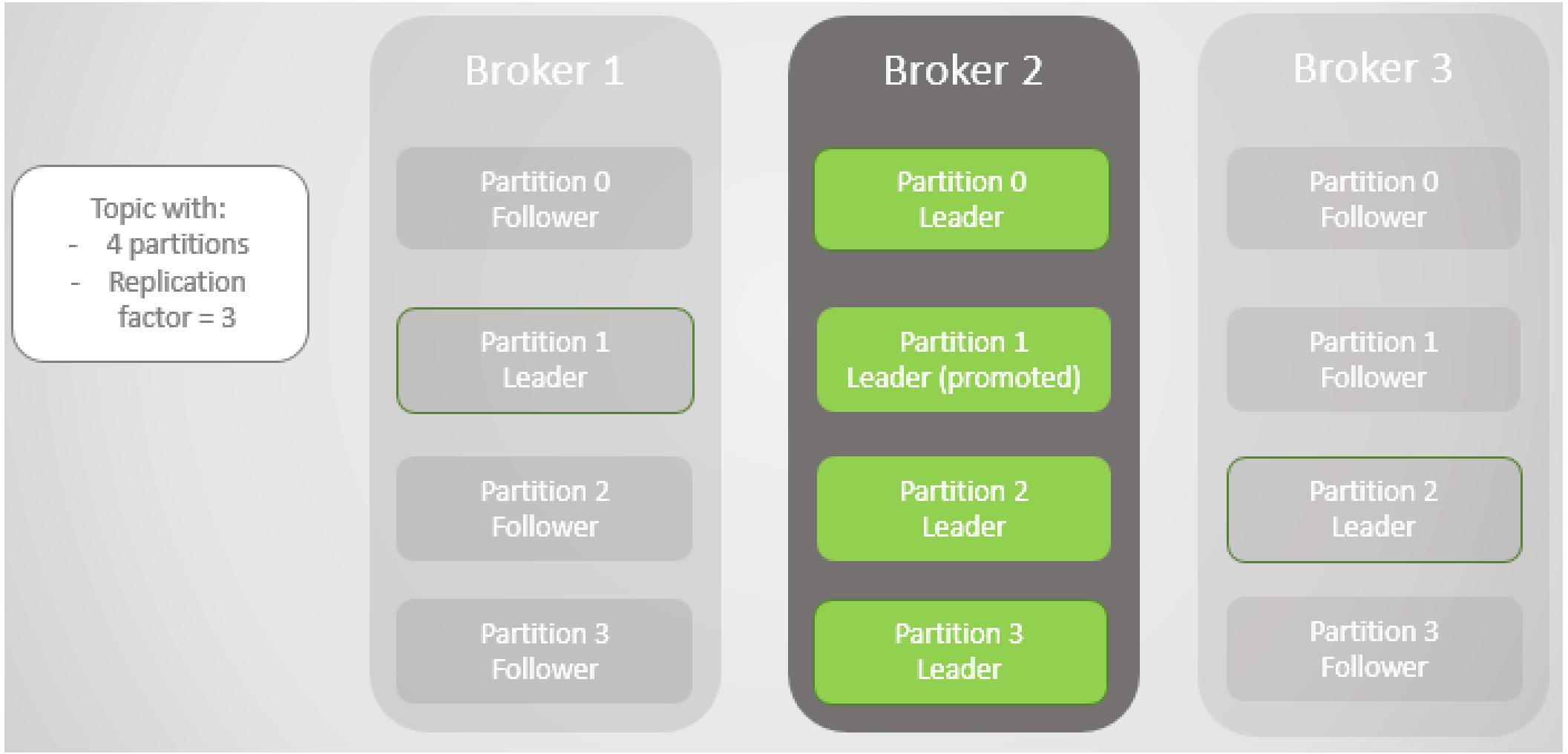
Producer, Consumer And Leader



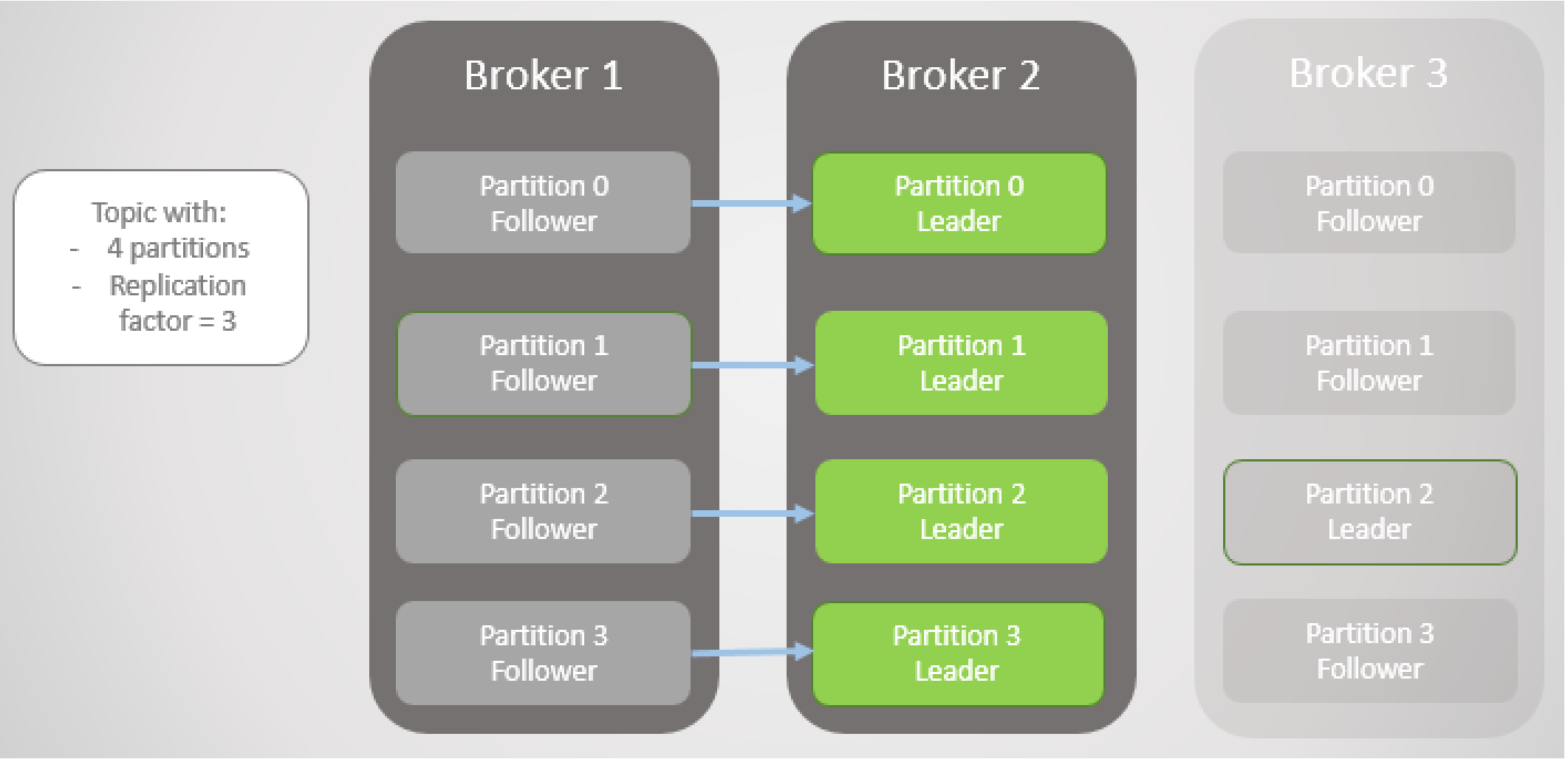
Partition Fail-Over - Broker 3 dies



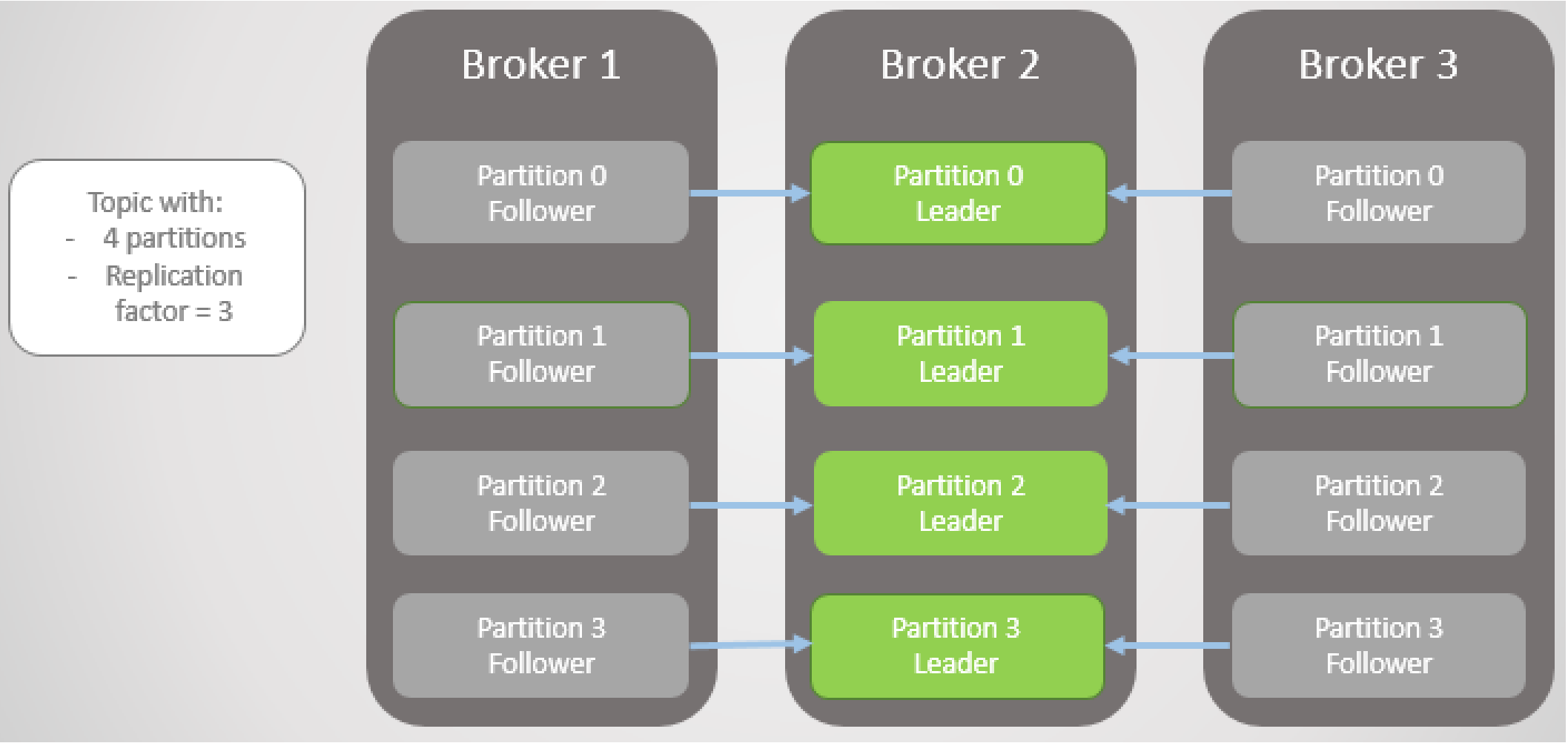
Partition Fail-Over - Broker 1 dies too



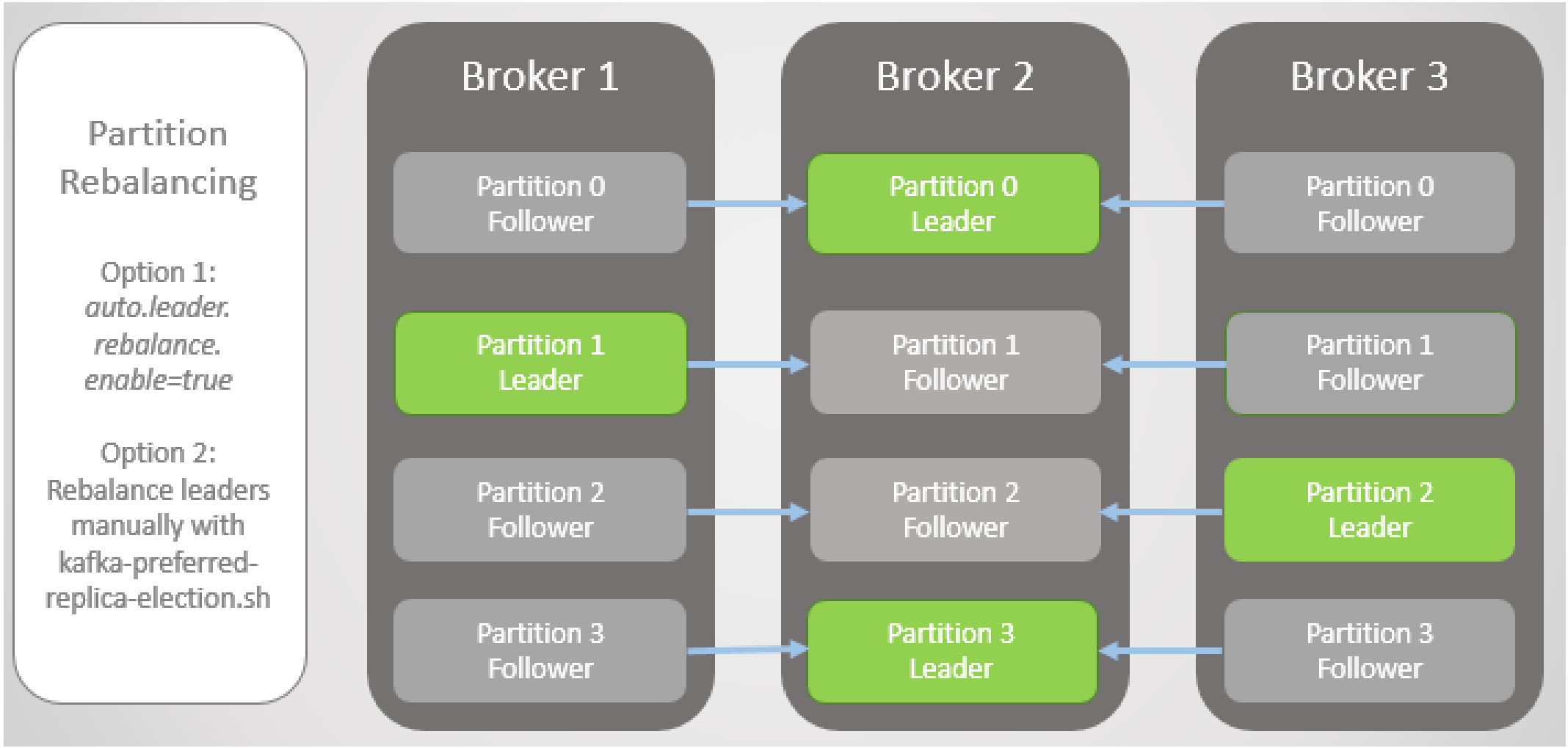
Leaders remain on broker 2



Unbalanced leaders after recovery of broker 1 and 3



Replica leaders rebalanced

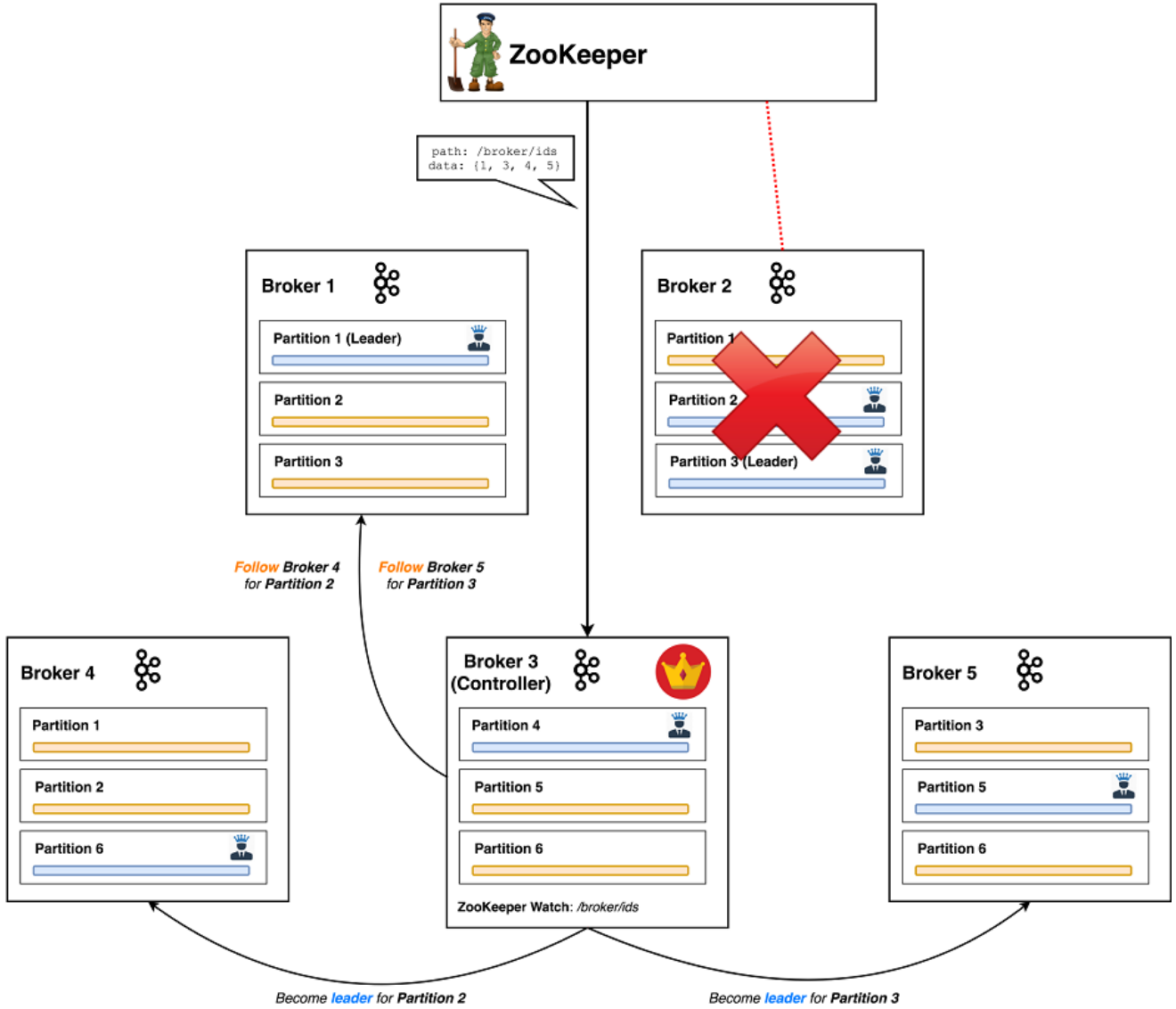


10 - Broker – Fail Detection - Imbalance

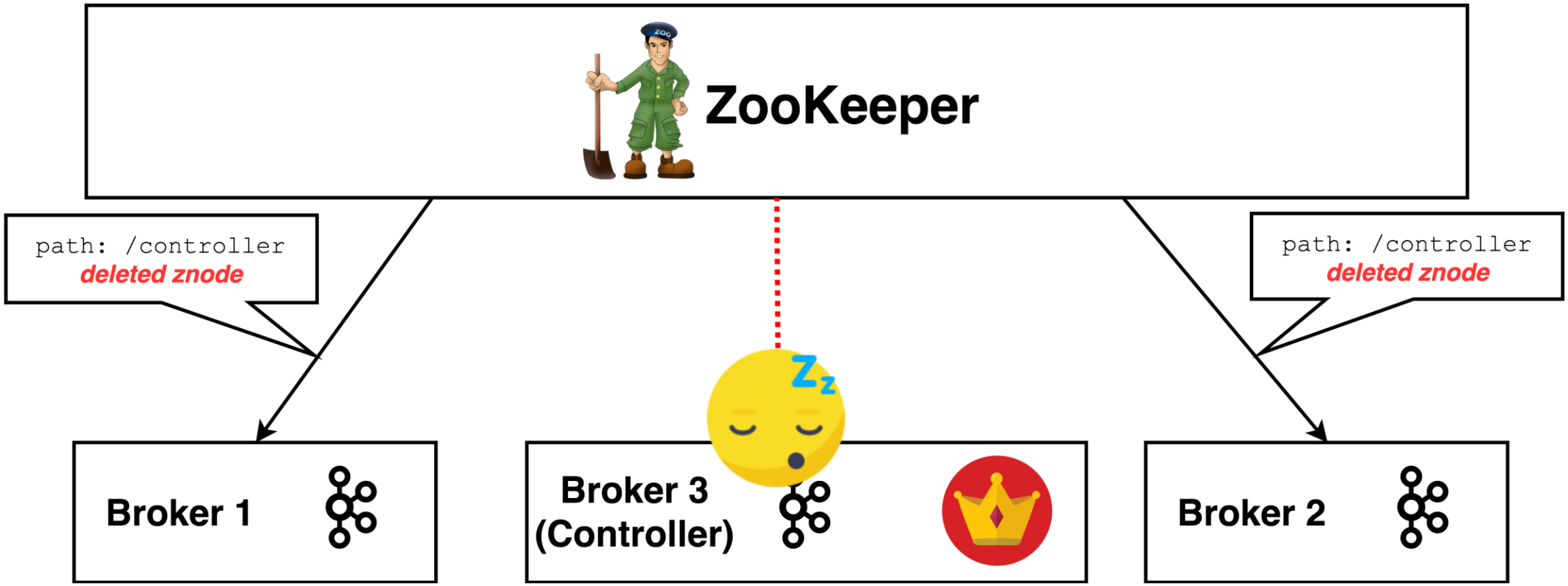
Controller Broker

* A distributed system must be coordinated.
    * Event happens & React in an organized way
* It is a normal broker that simply has additional responsibility
* Additional responsibility
    * Keeping track of nodes in the cluster
    * Handling nodes that leave, join or fail
* There is always exactly one controller broker in a Kafka cluster.

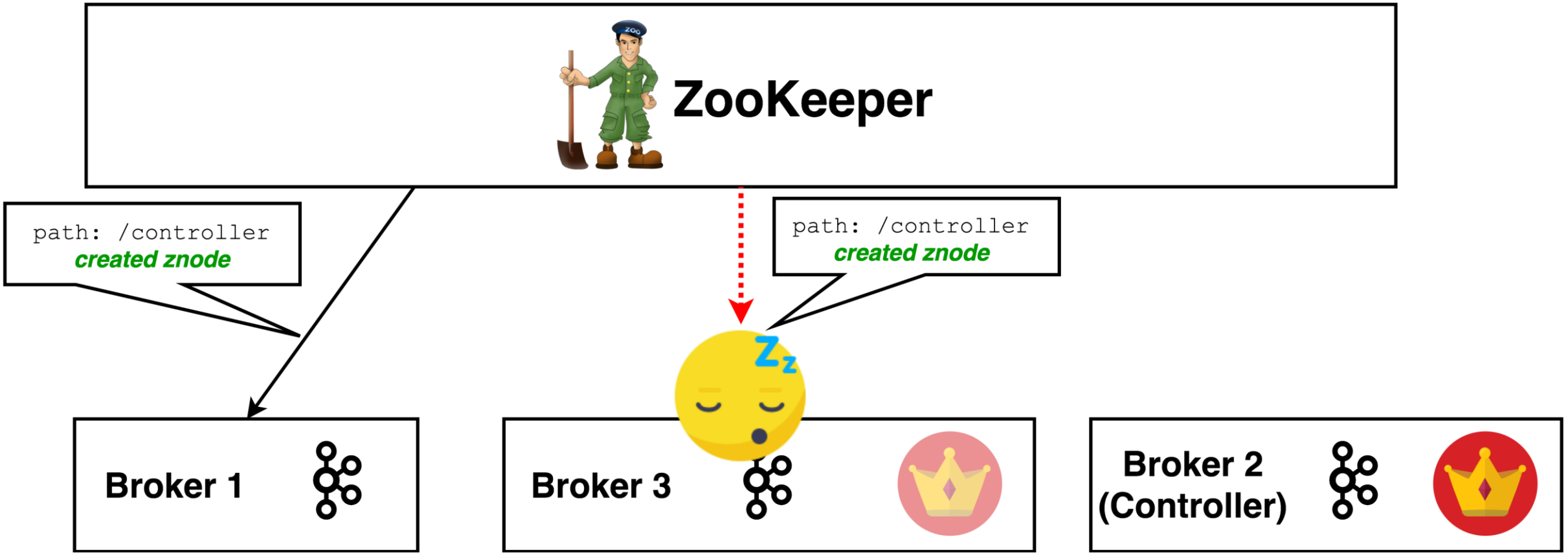
Handle a node leaving the cluster



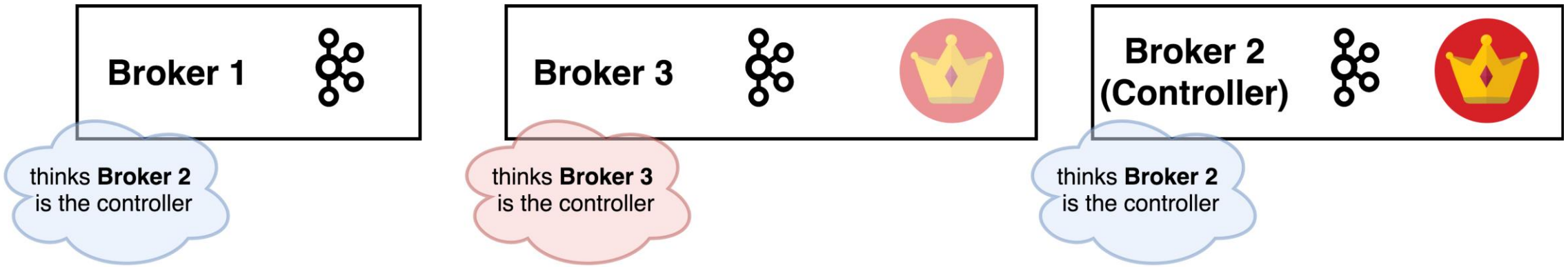
Split Brain


ZooKeeper
path: /controller
deleted znode
path: /controller
deleted znode
Broker 1
Broker 3
(Controller)
Broker 2

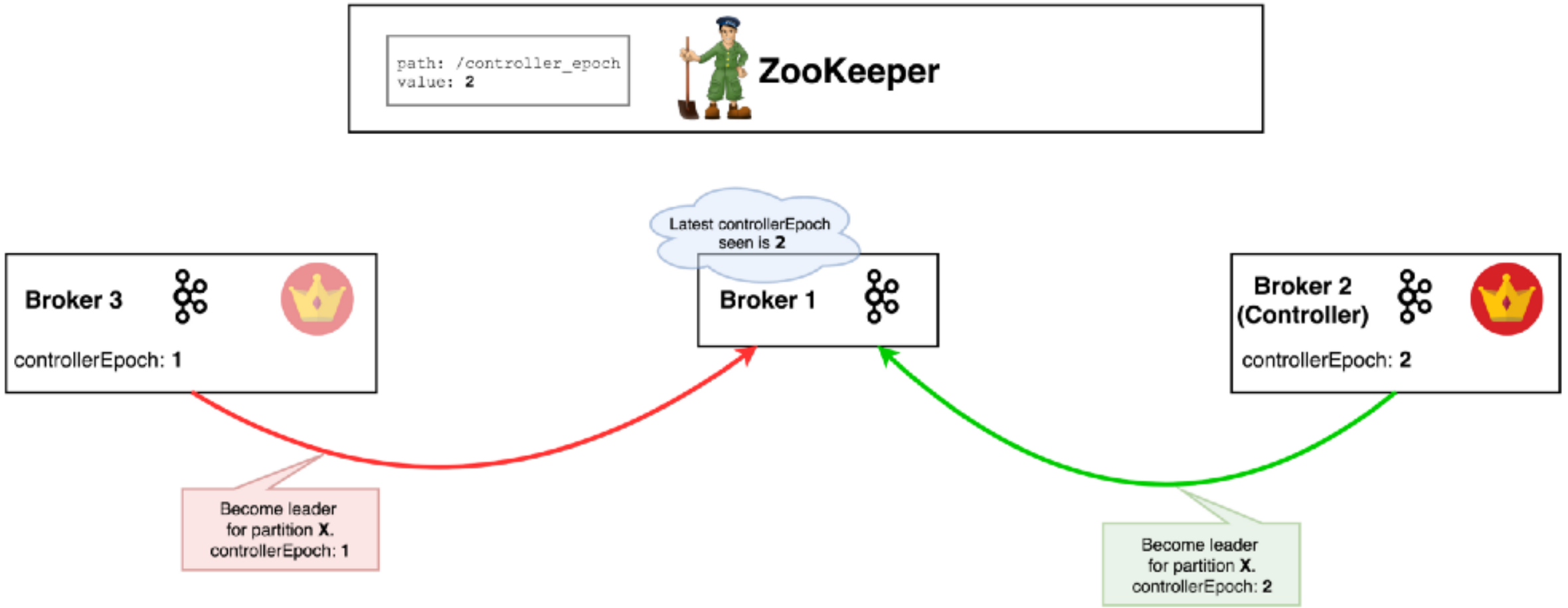
Split Brain - Broker 2 won the race



Broker 3 wake up



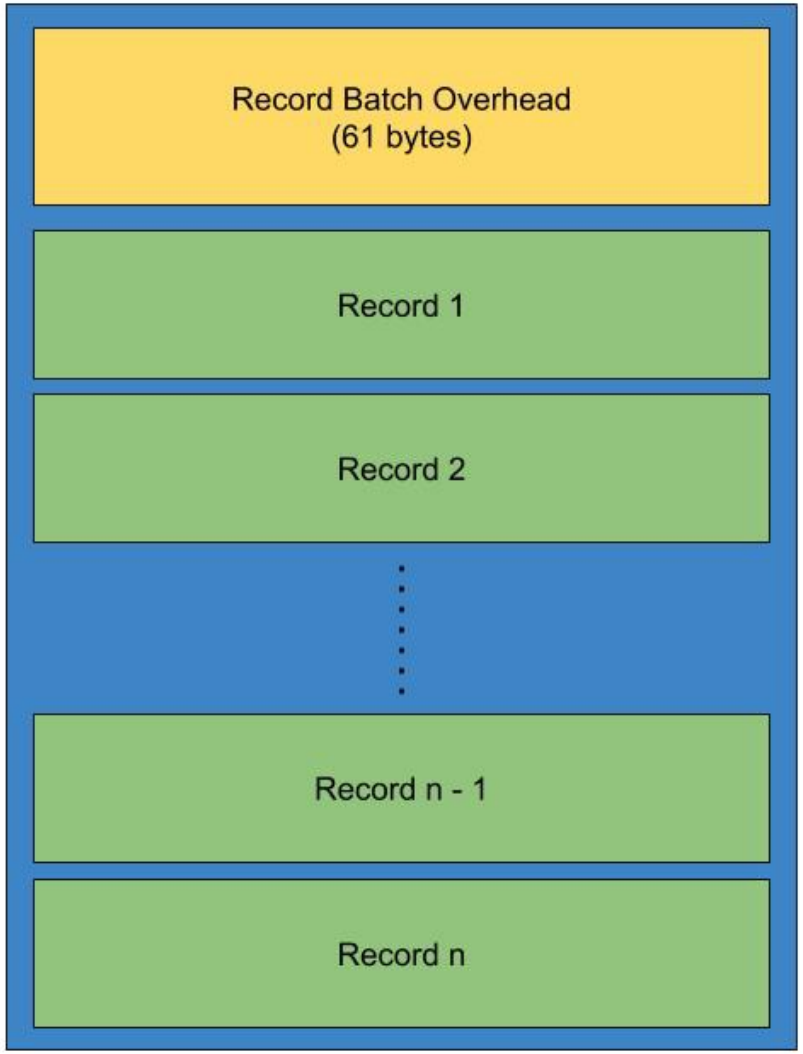
Epoch Number



11 - Logs – Compression - Retention

Why do I need to reduce message size?

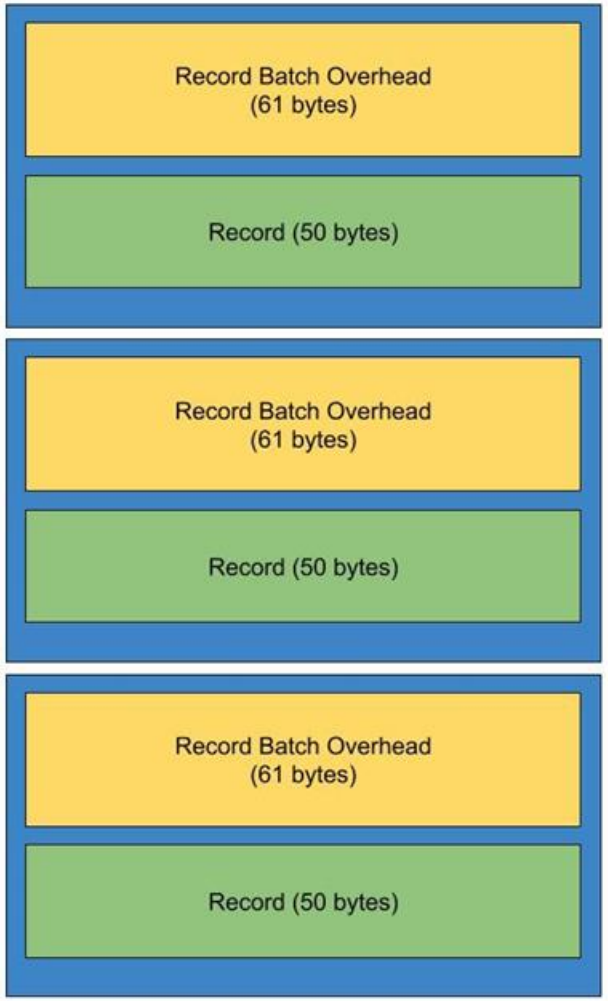
Message Format



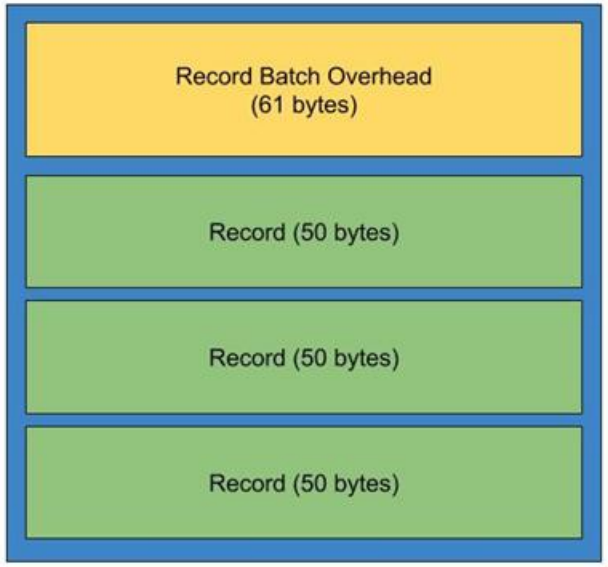
Record Batch

Lingering

Without Lingering VS Using Lingering

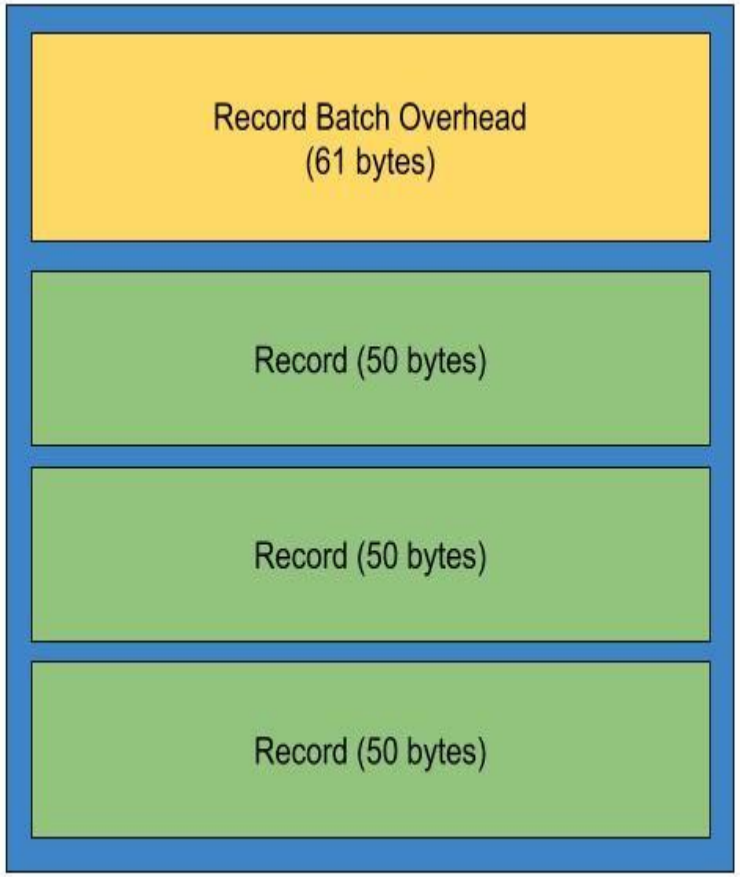


Without lingering we need 333 bytes to store 3 records

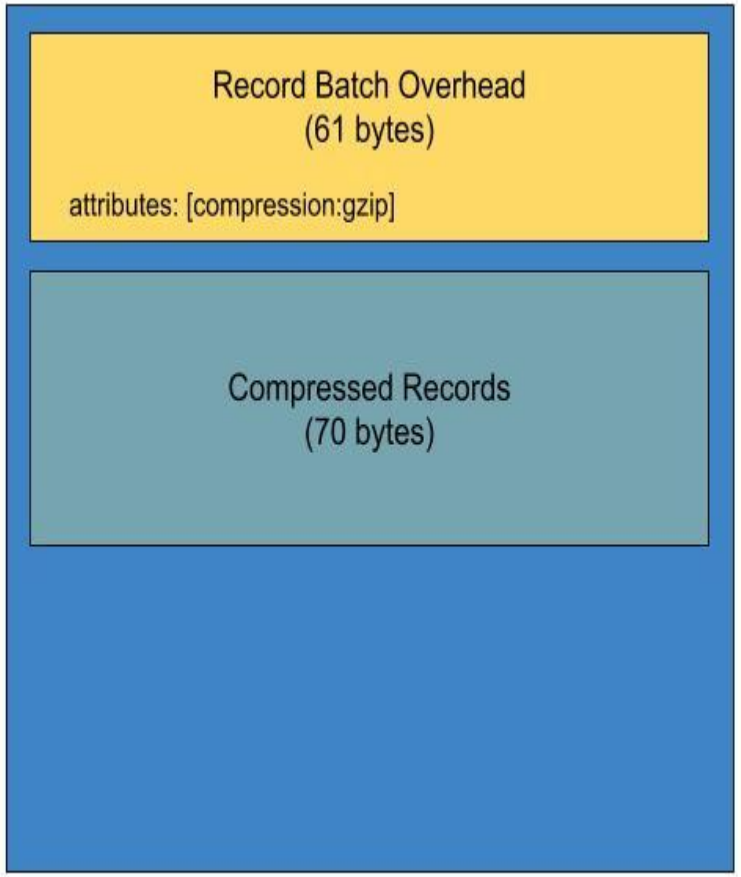


Using lingering we need 211 bytes to store 3 records

Compression



Using lingering we need 211 bytes to store 3 records

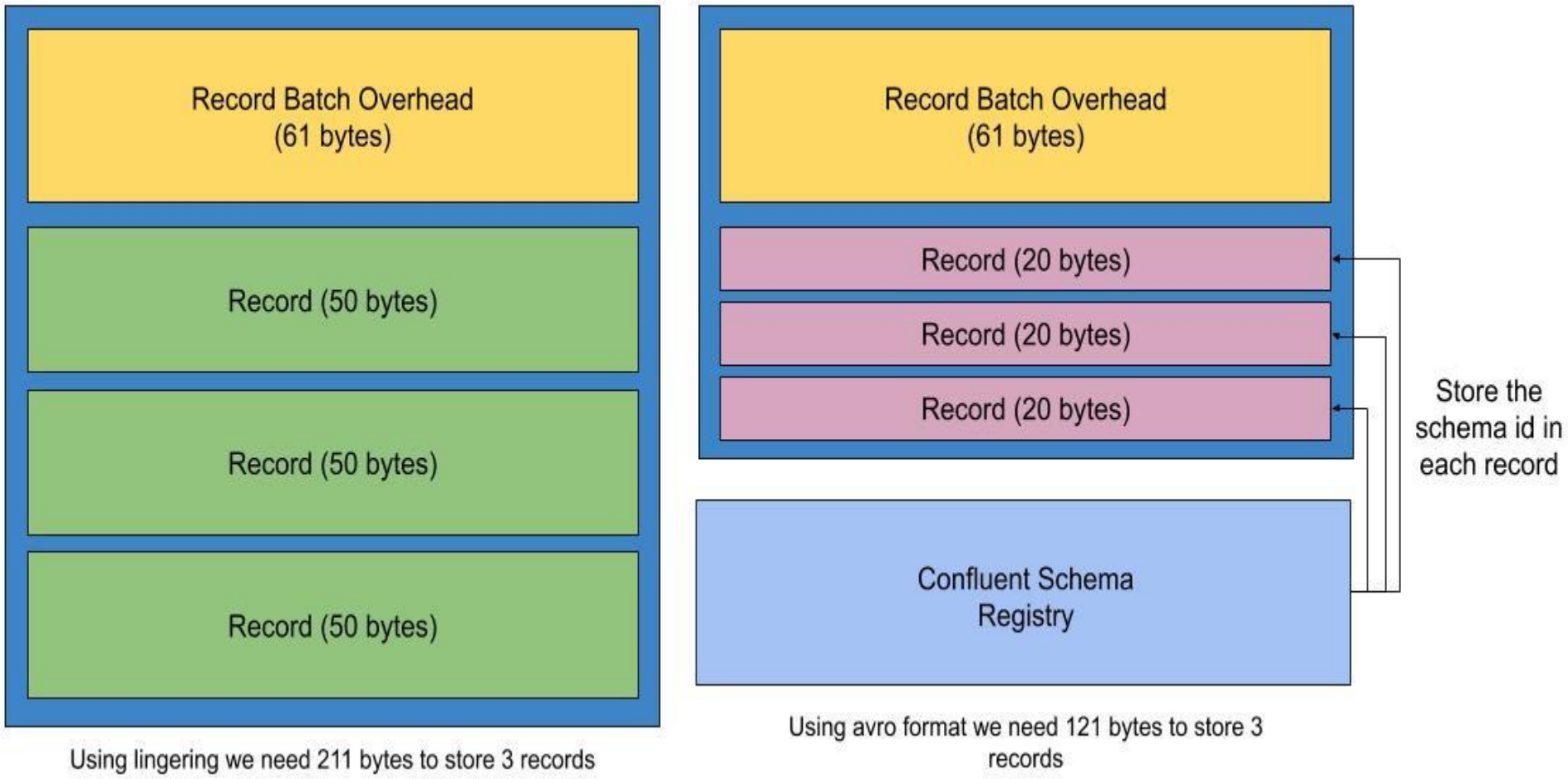


Using lingering and compression we need 131 bytes
to store 3 records

Support for Zstandard Compression (KIP-110)

	Ratio	Comp Speed MB/s	Decomp Speed MB/s
zstd cli	3.14	100.1	459.40
gzip cli	3.11	19.66	125.55

Schema


Record Batch Overhead
(61 bytes)
Record (50 bytes)
Record (50 bytes)
Record (50 bytes)
Using lingering we need 211 bytes to store 3 records
Record Batch Overhead
(61 bytes)
Record (20 bytes)
Record (20 bytes)
Record (20 bytes)
Confluent Schema
Registry
Store the
schema id in
each record
Using avro format we need 121 bytes to store 3
records

Conclusion

- Lingering needs you wait a bit more to gather more records.
- Compression uses more CPU usage but will reduce the amount of IO.
- Using Avro impose
  - Dependency on clients (consumer and producer)
  - have Confluent Schema Registry to keep the schema of the records.

Retention


RETENTION
MARKETING
SOCIAL
PLANNING
ROI
PACKAGES
EXPERIENCE
SOLUTIONS
CLUBS
SERVICES
CLIENTS
WELLNESS
SUCCESS
REFERRALS
MEMBERSHIP
LOYALTY
EARNINGS
REVIEWS
SATISFACTION
WORD OF MOUTH
VISITS
REWARDS
REPEAT VISITS
CX
REVENUES
MARKETING
SOLUTIONS
SOCIAL
REVENUES
LOYALTY
MEMBERSHIP
REFERRALS
CLIENTS
EXPERIENCE
EARNINGS
VISITS
PACKAGES
REWARDS
SUCCESS
SATISFACTION
PLANNING
REVIEWS
ROI
CLUBS
SERVICES
CX
WELLNESS
REPEAT VISITS
WORD OF MOUTH

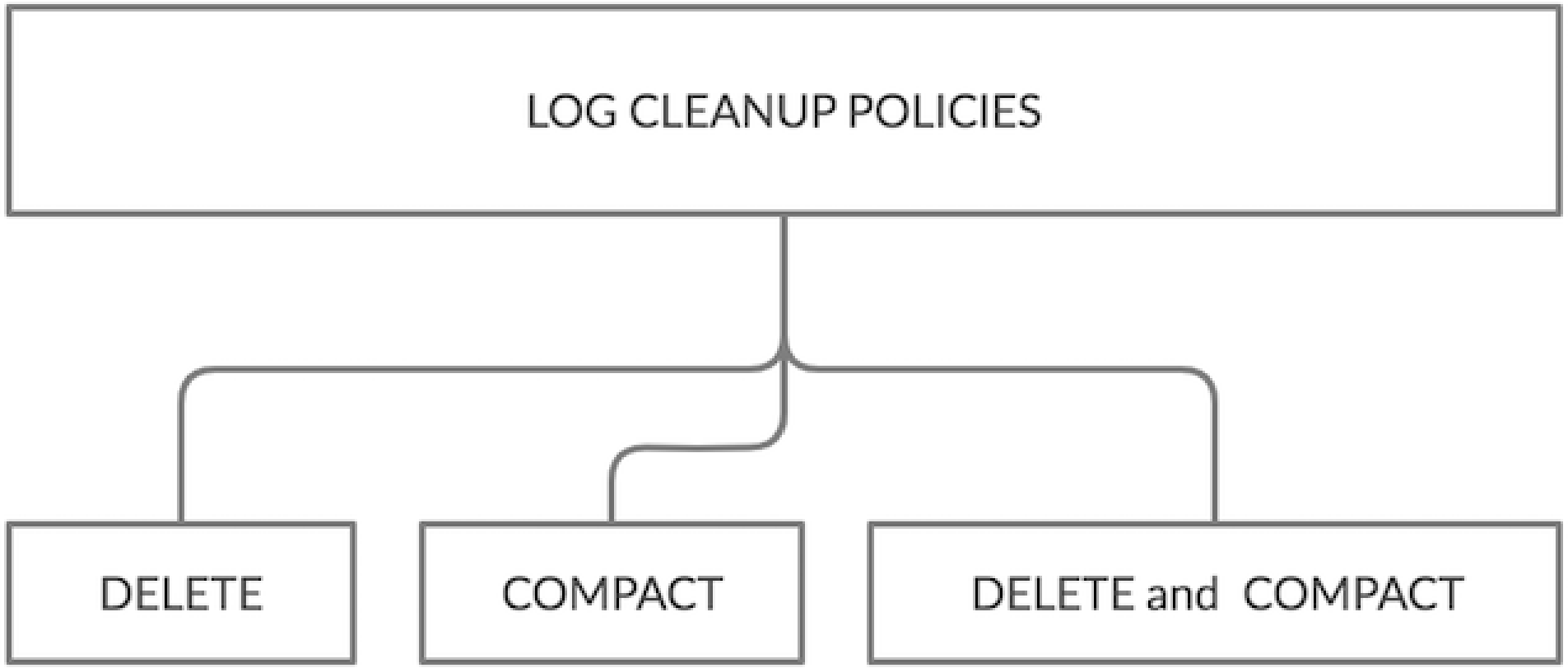
Time Based Retention

Config Name	Description	Default Value	Valid Values
log.retention.ms	The number of milliseconds to keep a log file before deleting it (in milliseconds), If not set, the value in log.retention.minutes is used	null	any long value
log.retention.minutes	The number of minutes to keep a log file before deleting it (in minutes), secondary to log.retention.ms property. If not set, the value in log.retention.hours is used	null	any long value
log.retention.hours	The number of hours to keep a log file before deleting it (in hours), tertiary to log.retention.ms property	168	any long value

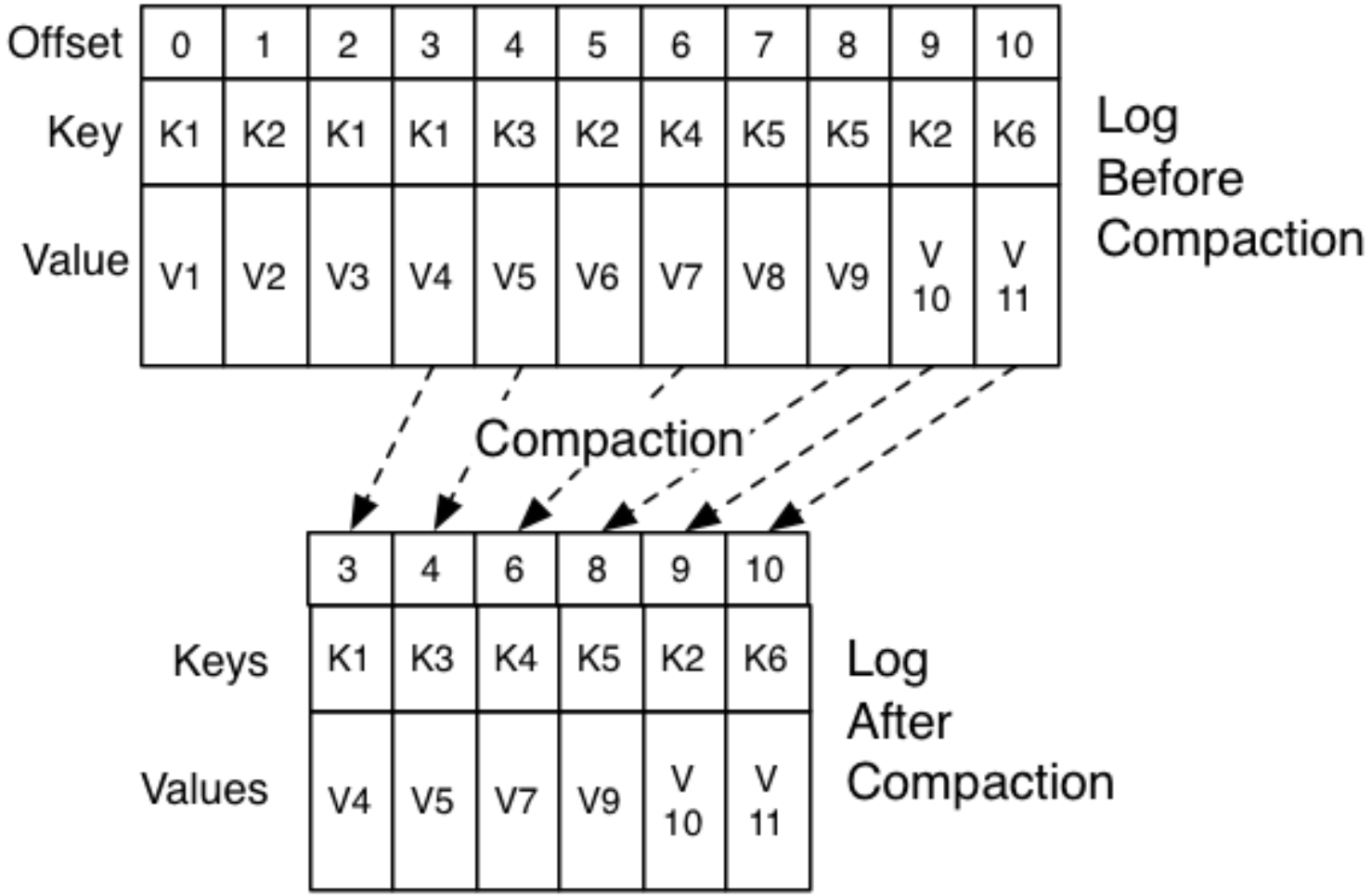
Size Based Retention

Config Name	Description	Default Value	Valid Values
log.retention.bytes	The maximum size of the log before deleting it	-1	any long value

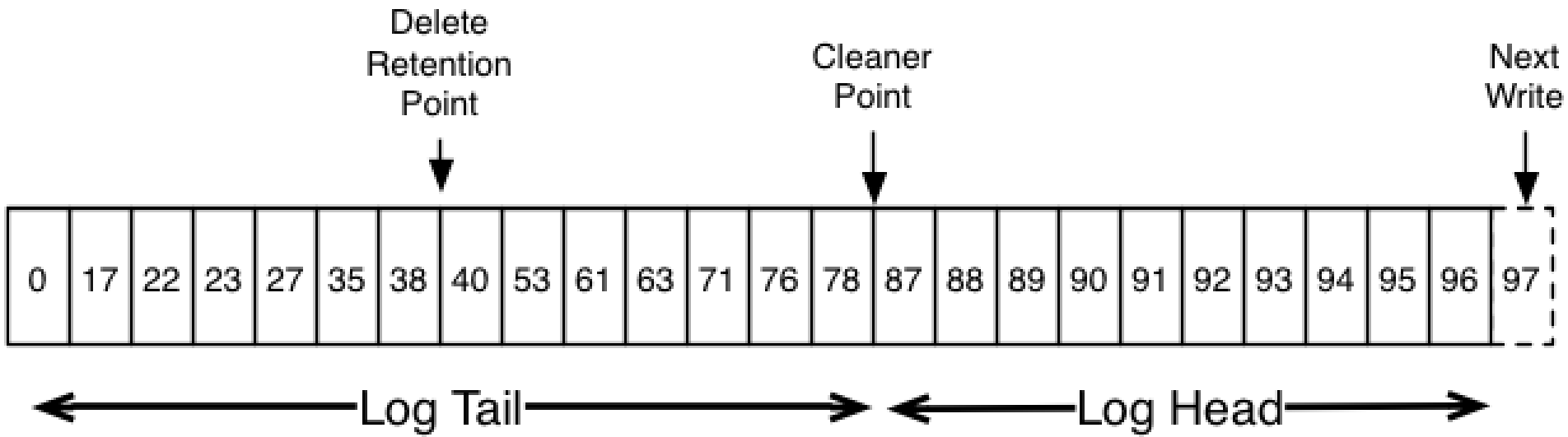
Log Cleanup Policies



Compact Policy



Log Cleaner


Delete
Retention
Point
Cleaner
Point
Next
Write
0
17
22
23
27
35
38
40
53
61
63
71
76
78
87
88
89
90
91
92
93
94
95
96
97
Log Tail
Log Head

FAQ – Log Cleaner

1. Does Log Cleaner impacts Read performance?

2. Does offset of message gets changed after compaction?

3. Does Log Cleaner also deletes Tombstone messages?

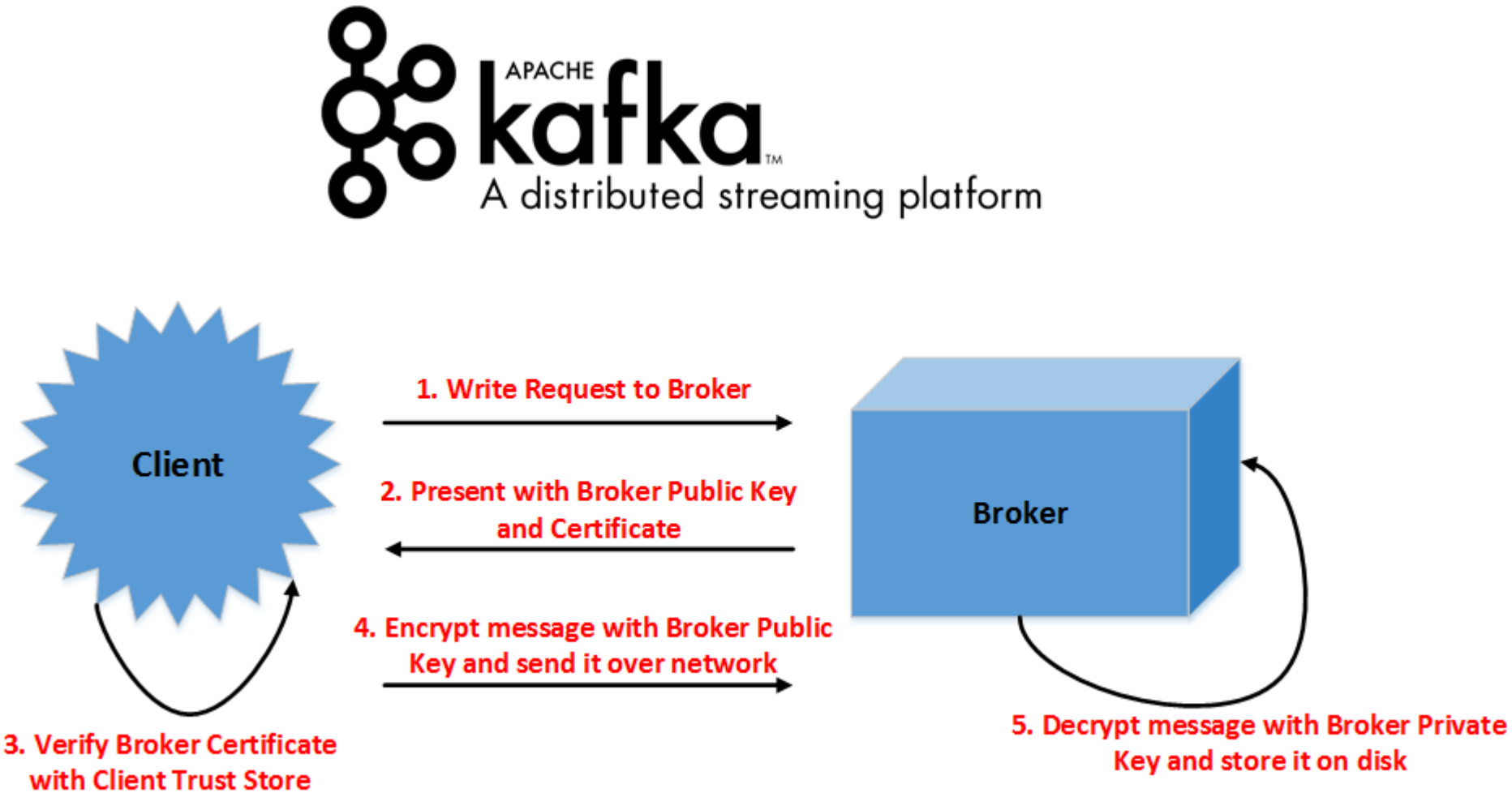
4. Does order of messages gets changed after compaction?

12 - Security – TLS – Authentication - Authorization

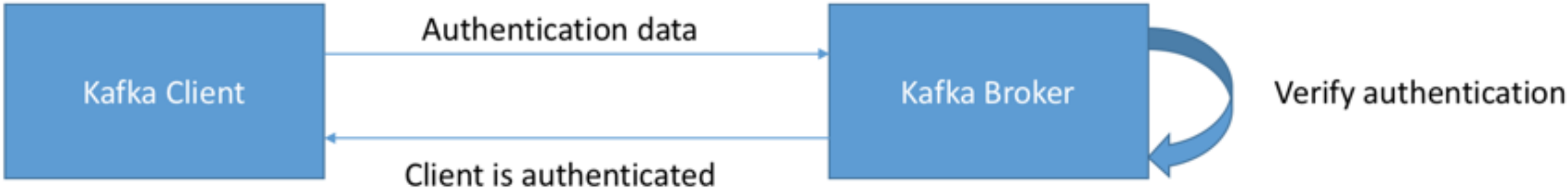
The need for security

Problems Security is solving

Encryption


APACHE
kafka
A distributed streaming platform
Client
1. Write Request to Broker
2. Present with Broker Public Key and Certificate
4. Encrypt message with Broker Public Key and send it over network
Broker
3. Verify Broker Certificate with Client Trust Store
5. Decrypt message with Broker Private Key and store it on disk

Authentication



Authorization

- "User alice can view topic finance"
- "User bob cannot view topic trucks"

بـا ســـپاس