upGrad

Real-Time Data Streaming with Apache Kafka

upGrad

About

upGrad

upGrad

Course: Data Engineering - II

Lecture On: Real-Time Data Streaming with Apache Kafka

Instructor: **Vishwa Mohan**

TOPICS

01 Topics are organised collections of data. They exist to fulfil a particular purpose.

02 Each topic has a unique name. Kafka can have multiple topics depending on the storage capacity.

03 Each topic is further divided into partitions. The topics are then stored in brokers.

04 Each partition of a topic will be divided into different brokers for fault tolerance. A single topic can have multiple consumers.

PARTITIONS

01 Topics are divided into partitions.

02 Topics are stored in broker. Each partition is stored in a different broker for fault tolerance.

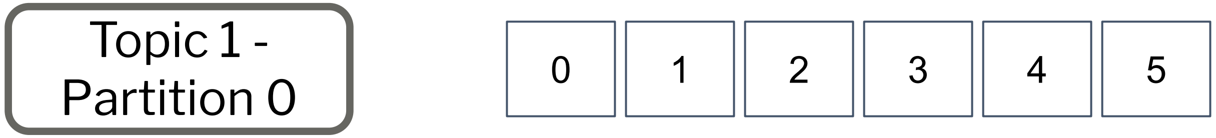
03 Each partition can only be read in the order in which the message was added to it.

04 Increasing the number of partitions increases parallelism. Multiple consumers read from partitions in-parallel.

PARTITIONS

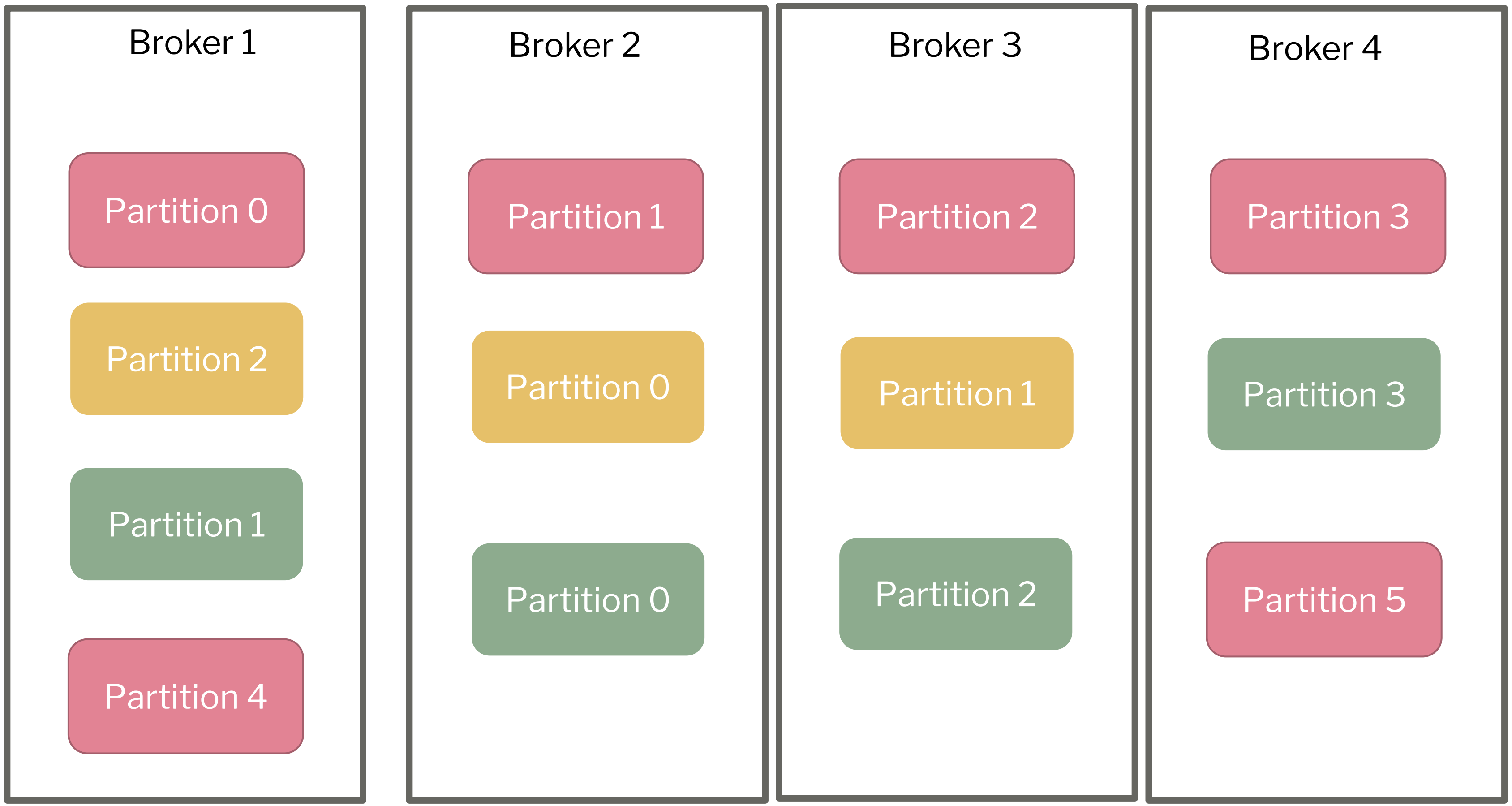
01 Partitions have messages ordered by offset. A particular message can only be in one partition.

02 Offset is an incremental ID: Messages are given offsets when they are written to a partition.



03 An order is bounded to a partition.

04 Data is read in a sequential manner from each partition. Messages stored in a partition are immutable.


Broker 1
Partition 0
Partition 2
Partition 1
Partition 4
Broker 2
Partition 1
Partition 0
Partition 0
Broker 3
Partition 2
Partition 1
Partition 2
Broker 4
Partition 3
Partition 3
Partition 5

PRODUCERS

01 Producers are the ones who write data to topics.

02 They use bootstrap servers to connect to Kafka.

03 Messages are the smallest components when it comes to Kafka.

04 Messages contain Keys and Values.

PRODUCERS

05

If a key is null, then a round robin is used to write to partitions:

- Load balance
- Not to overload any partition in a topic

06

Keys are used to group data that belong together:

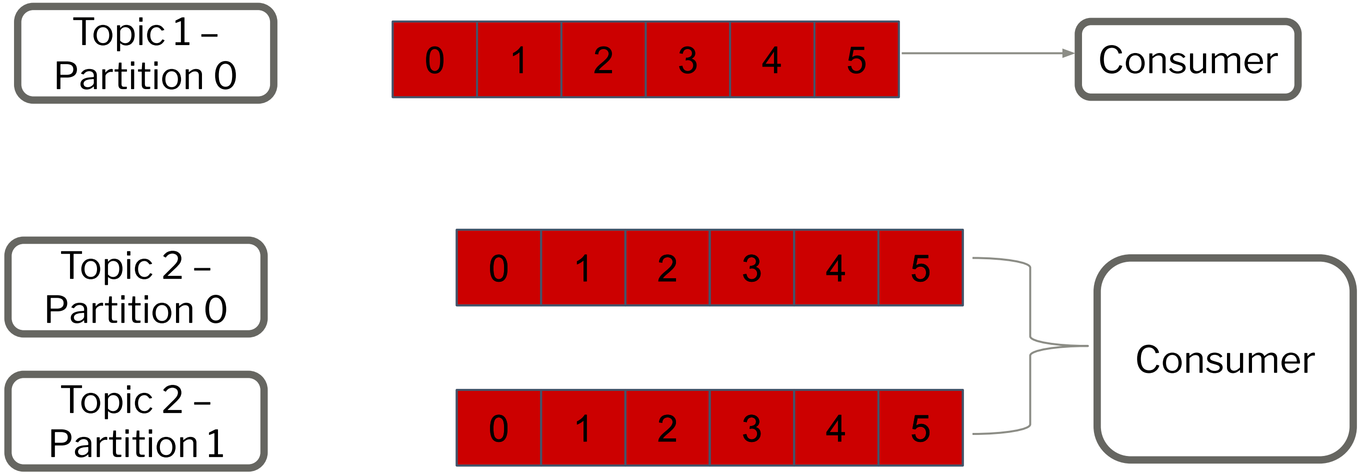
- Data with the same key are written to the same partition
- Example: If you want to pull analytics for each car in Google Maps, then you can use car id as key

CONSUMERS

01 Consumers read data from topics.

02 They use bootstrap servers to connect to Kafka.

03 They read data from each partition in a sequential manner.


Topic 1 – Partition 0
0
1
2
3
4
5
Consumer
Topic 2 – Partition 0
0
1
2
3
4
5
Topic 2 – Partition 1
0
1
2
3
4
5
Consumer

CHALLENGES

01 1 consumer reading from 2 partitions reads some messages from one of these partitions and some messages from the other partition. No parallelism is achieved with a single consumer.

02 If there are multiple consumers, then how do we know which consumer is reading from which partition?

03 How do we make sure a set of consumers belong to a particular application?

04 If one consumer goes down, then how does a new consumer pick up from where the previous consumer left off?

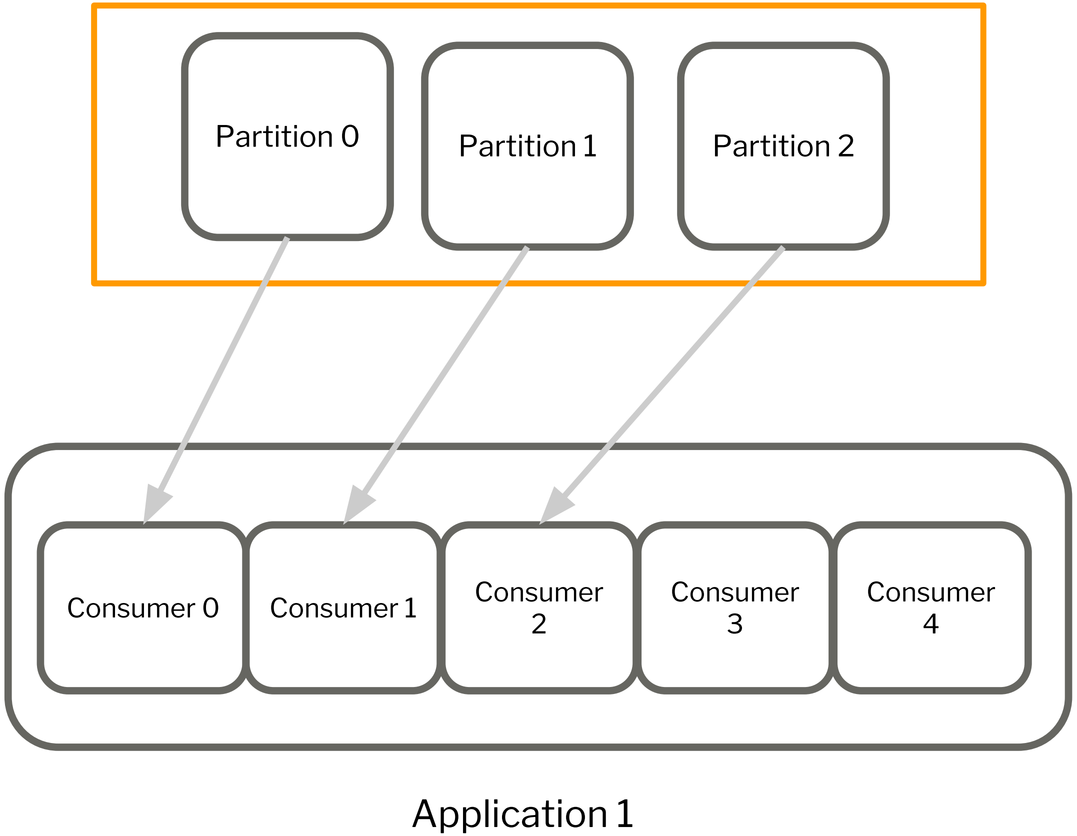
05 How to introduce parallelism?

CONSUMER GROUPS

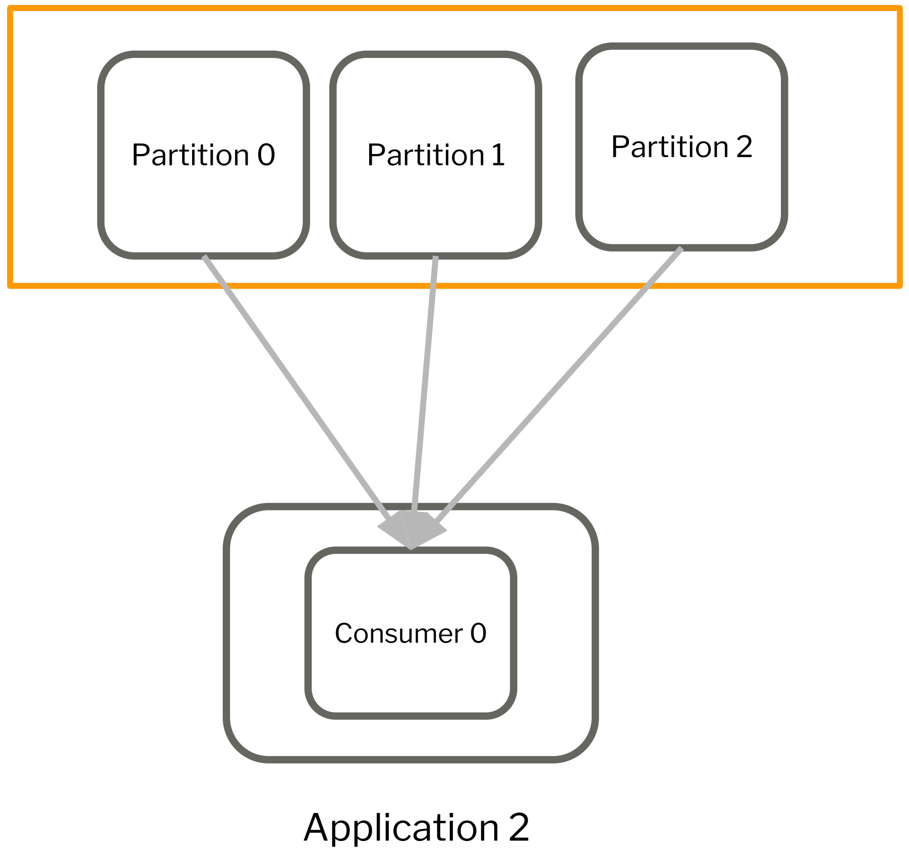
01 A set of consumers who are grouped together and are identified by group id

02 Each group id is an application

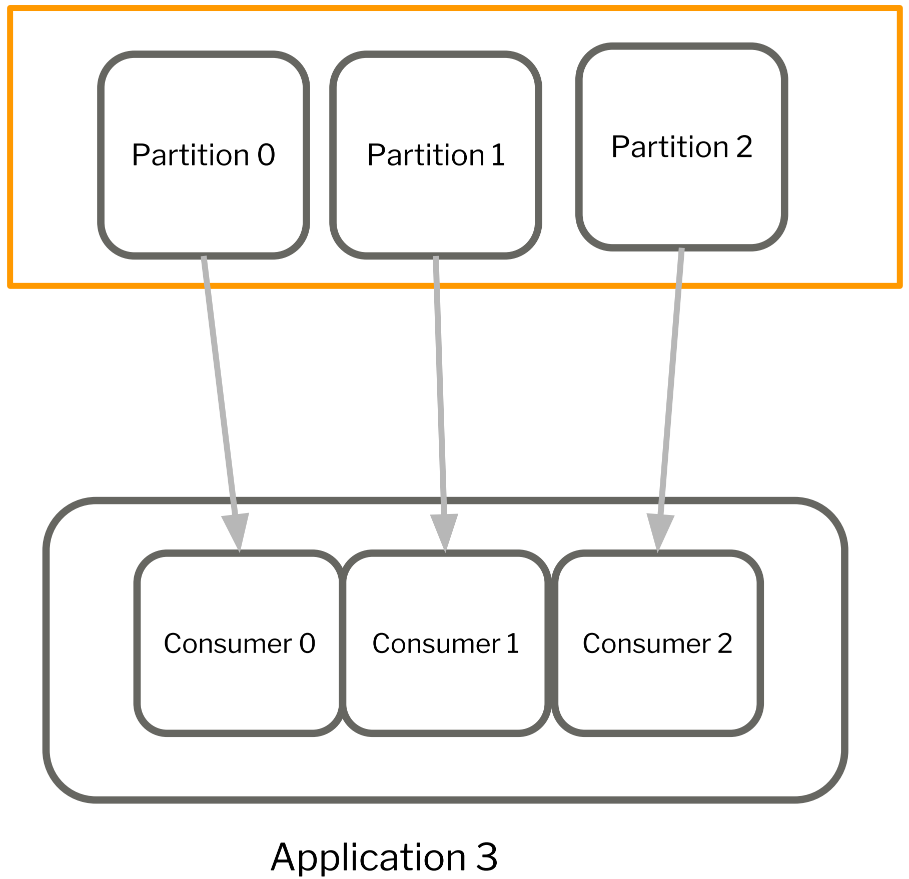
MORE CONSUMERS THAN PARTITIONS



LESS CONSUMERS THAN PARTITIONS



EQUAL NUMBER OF CONSUMERS AND PARTITION



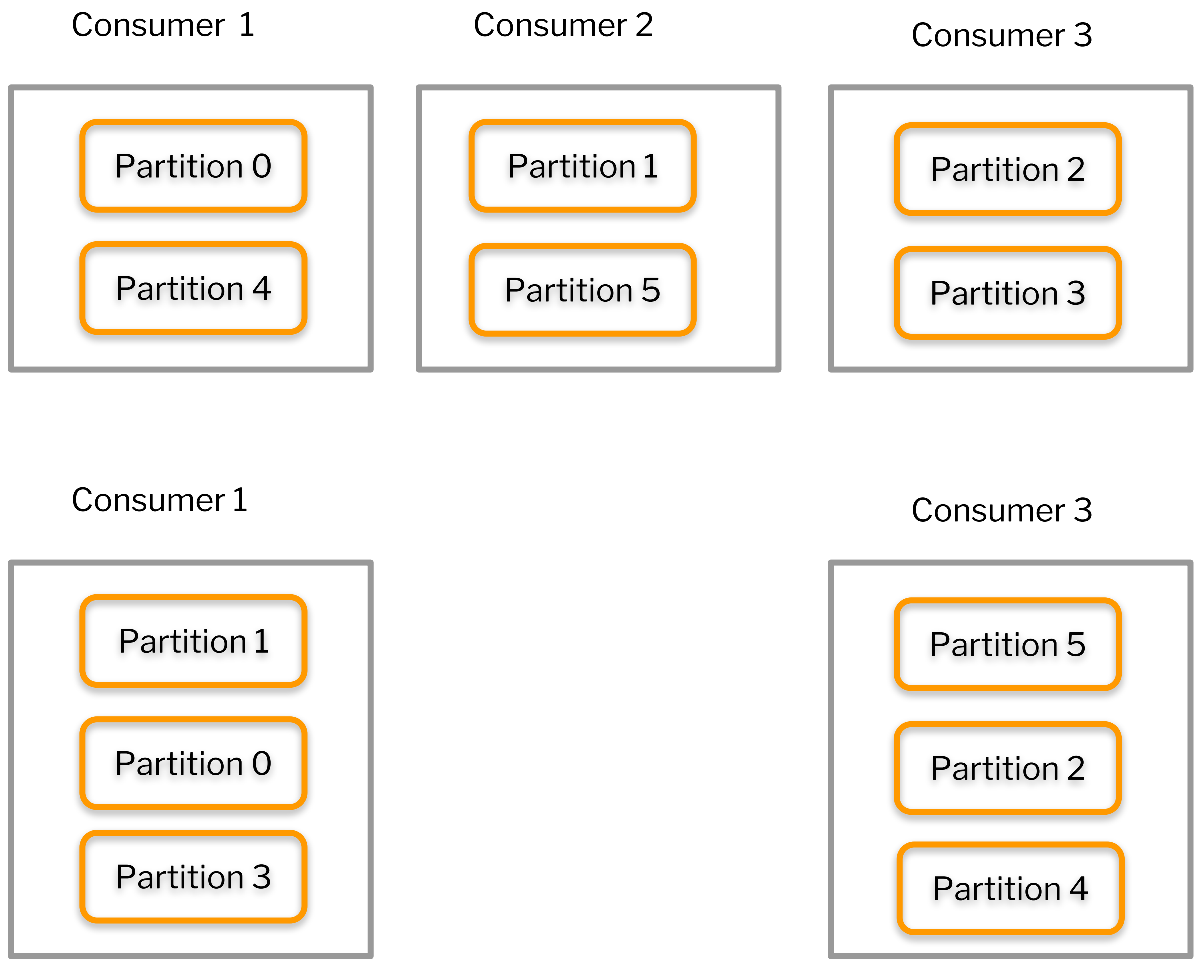
REBALANCING

01 Rebalancing happens in two cases:

- A consumer leaves a group
- A consumer gets added to a group

02 Rebalancing happens only as a group.

03 All partitions are re-assigned different consumers in rebalancing.


Consumer 1
Consumer 2
Consumer 3
Partition 0
Partition 4
Partition 1
Partition 5
Partition 2
Partition 3
Consumer 1
Consumer 3
Partition 1
Partition 0
Partition 3
Partition 5
Partition 2
Partition 4

TOPIC REPLICATION

01 It is crucial in distributed architecture.

02 It ensures the following qualities:

- High reliability
- Fault tolerance
- Mitigation of data loss

03 Replication factor decides the number of copies of a particular topic that will be created.

TOPIC REPLICATION

01

Problems with replication:

- Sync issues

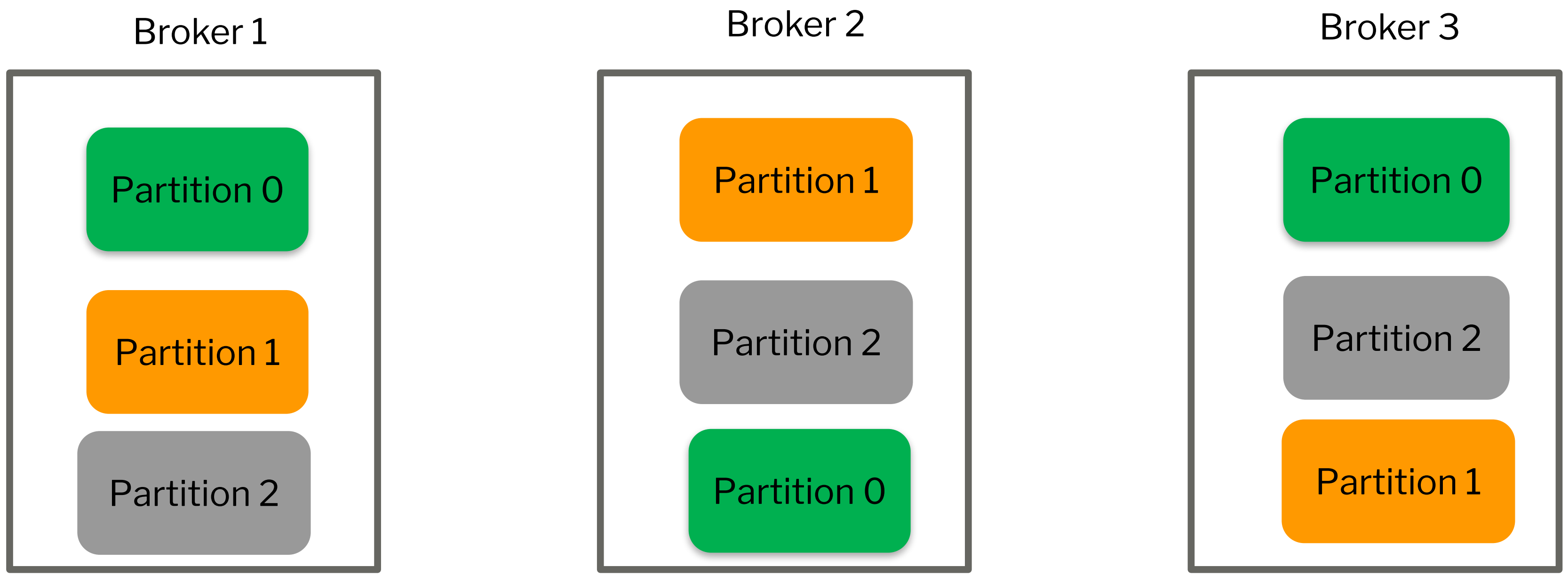
02

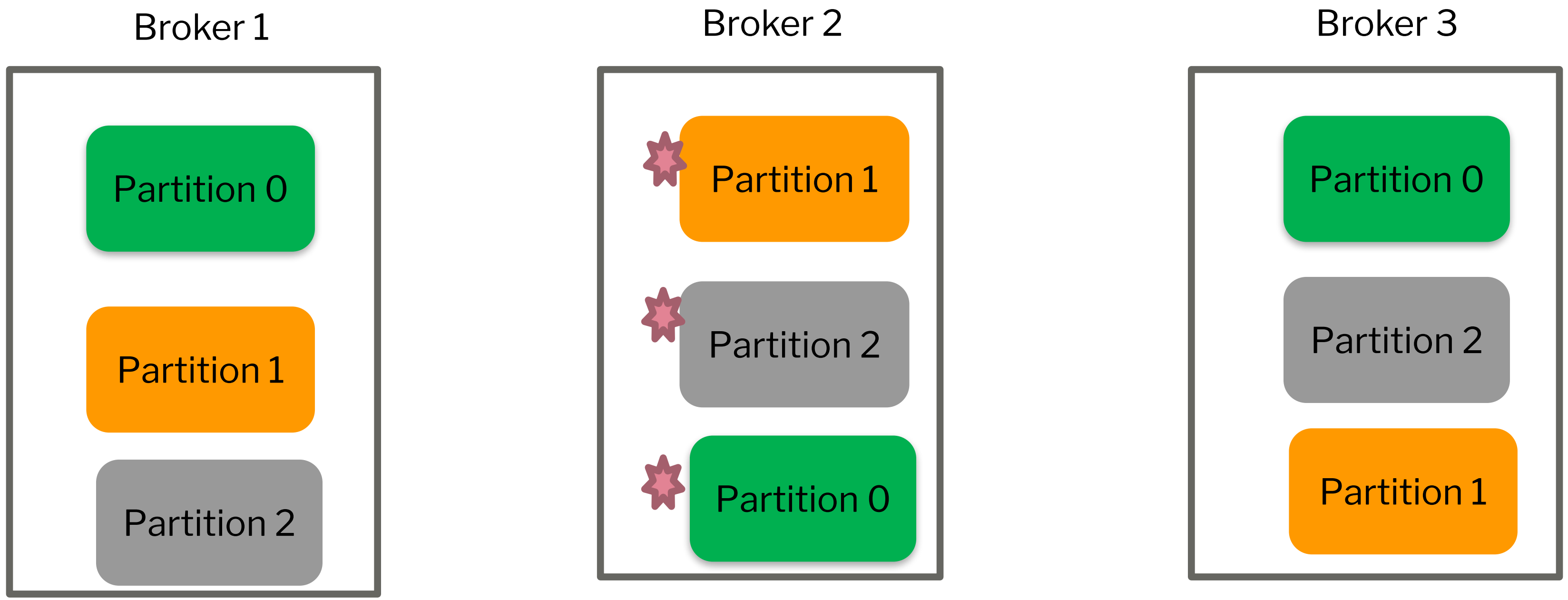
Solves it using the Leader approach:

- 1 Broker partition acts as the Leader for all the replicated partitions
    - Takes reads and writes

03

Zookeeper elects a new Broker as Leader if a Leader goes down


Broker 1
Partition 0
Partition 1
Partition 2
Broker 2
Partition 1
Partition 2
Partition 0
Broker 3
Partition 0
Partition 2
Partition 1


Broker 1
Partition 0
Partition 1
Partition 2
Broker 2
Partition 1
Partition 2
Partition 0
Broker 3
Partition 0
Partition 2
Partition 1

upGrad

Thank You