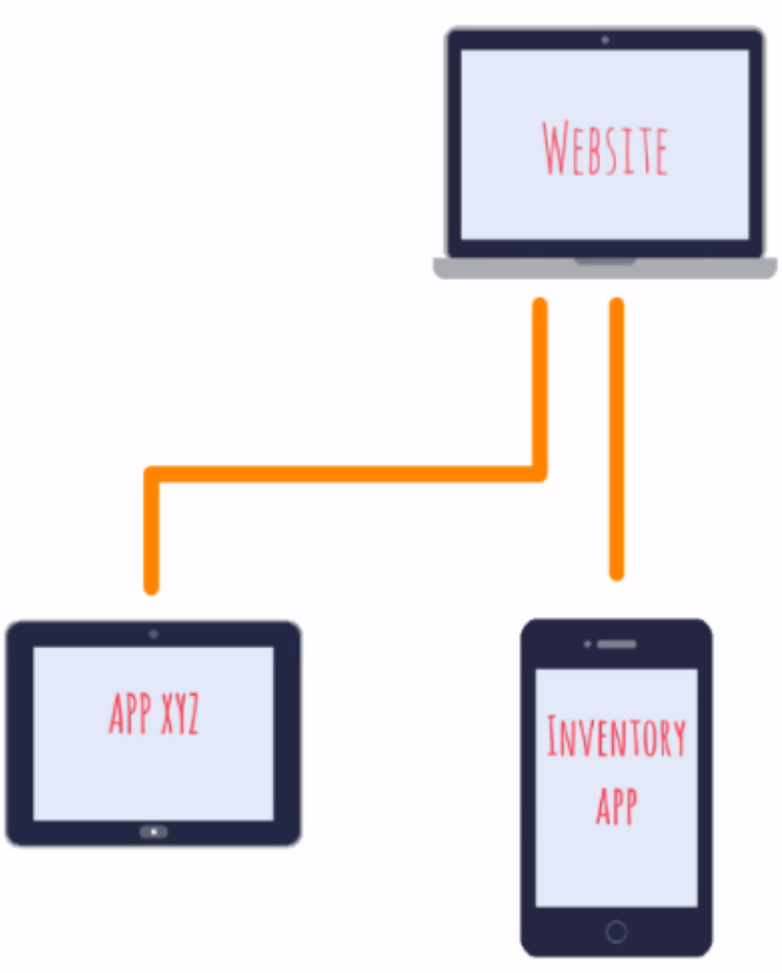
Lecture 10 –Kafka

BDAT 1002

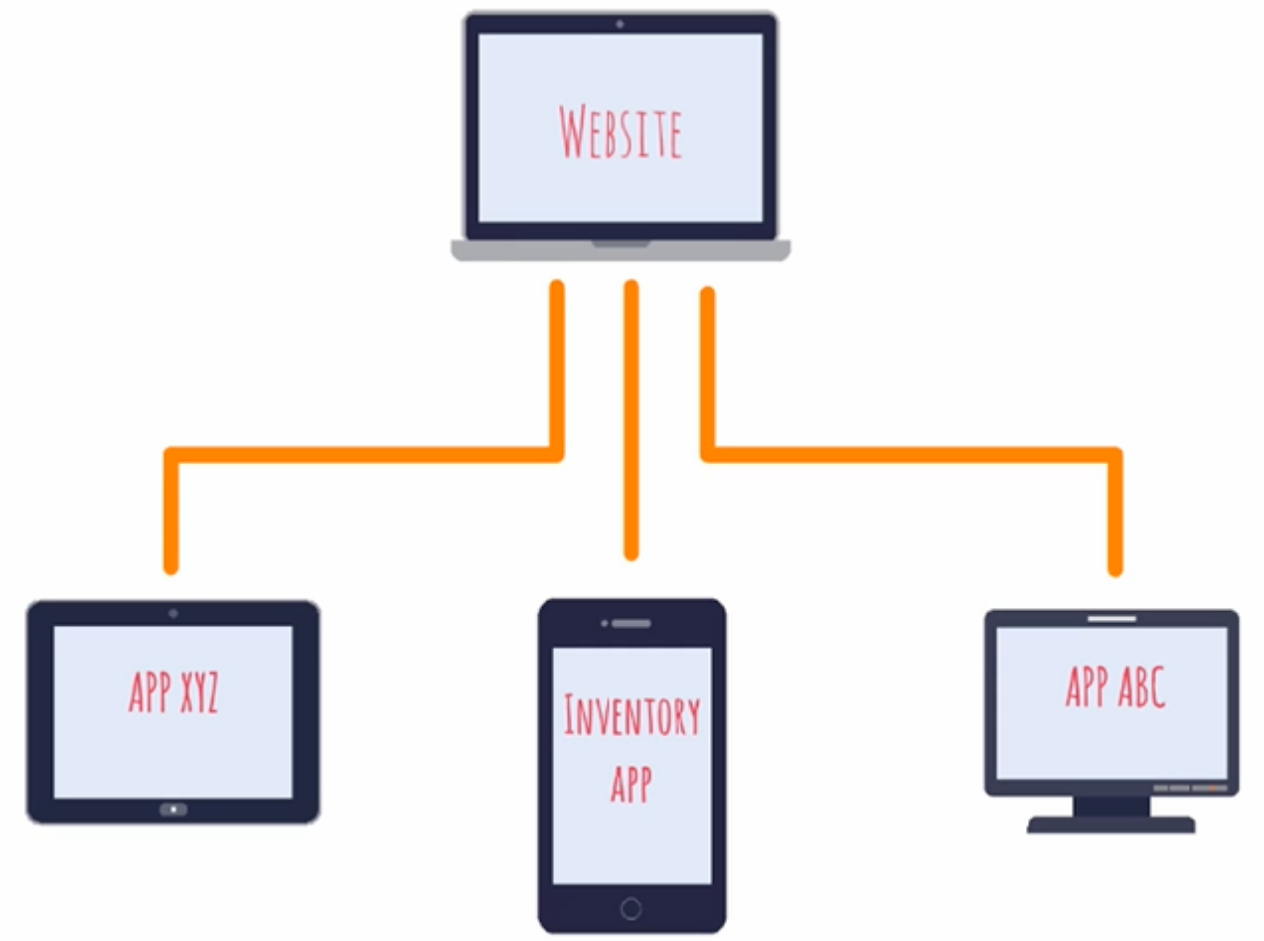
The Need for Kafka

Scenario

Scenario



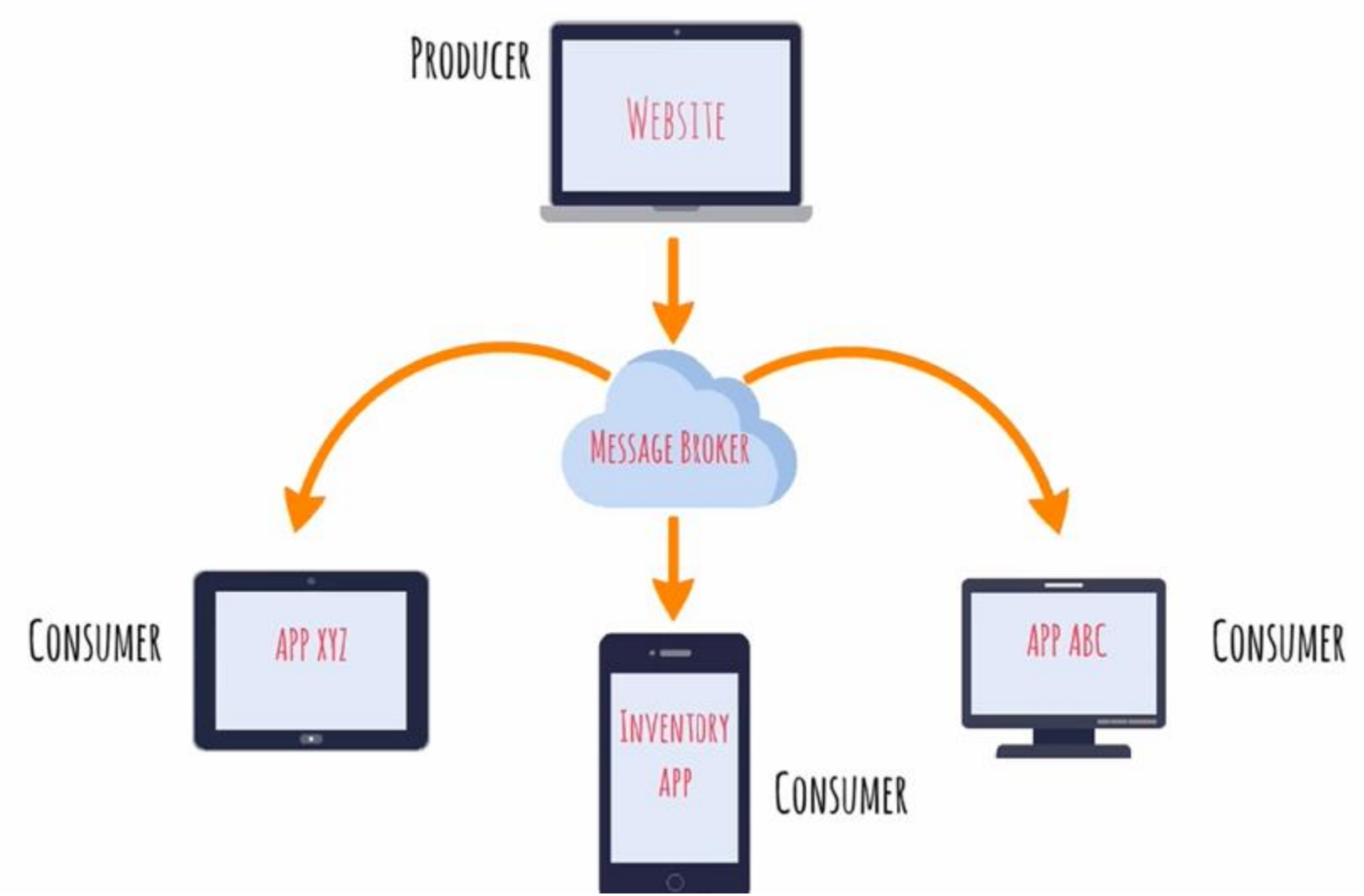
Scenario



Problems

- Two major problems
- All three applications become tightly coupled to one another
  - Changes to the website may affect the apps
  - Vice versa, if an app wants to change the way it gets information, can affect everyone else
- We have redundant code to communicate with different applications
  - Just a bit of a change for each application

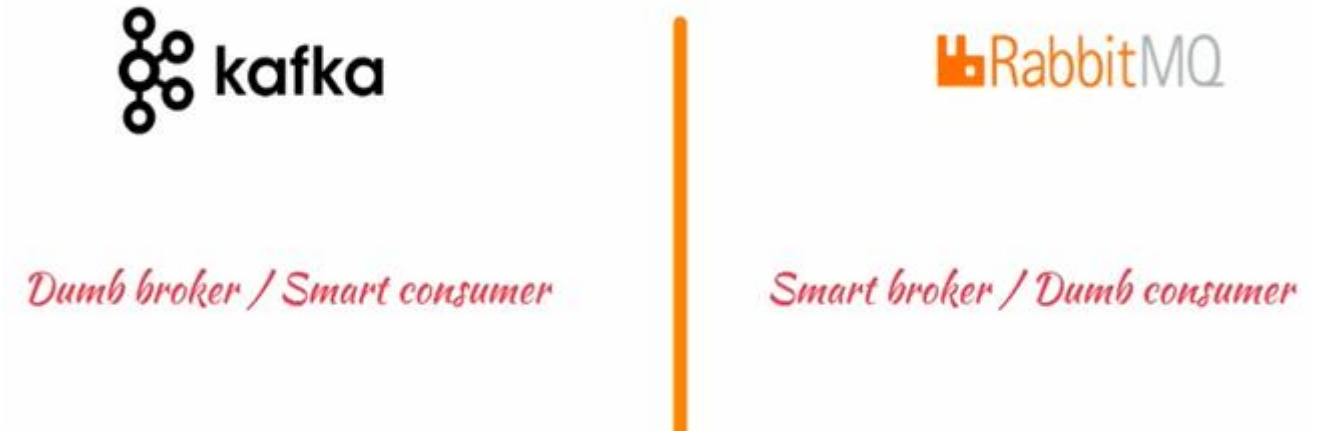
Solution: Message Broker



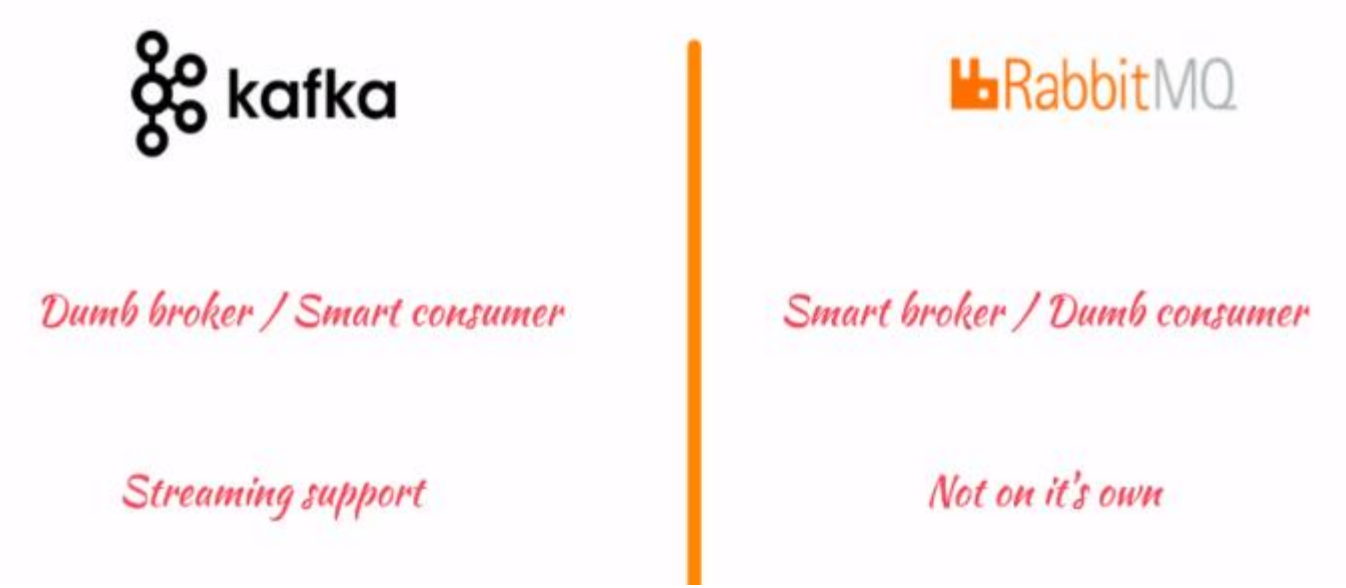
Kafka

- Kafka is open source
- Developed by LinkedIn
- Written in Scala (mostly)
- But is Kafka the first message broker ever written?

Kafka vs other MB's



Kafka vs other MB's



Kafka vs other MB's

kafka	RabbitMQ
Dumb broker / Smart consumer	Smart broker / Dumb consumer
Streaming support	Not on it's own
External tools - Zookeeper	External tools - not needed

Kafka vs other MB's

kafka	RabbitMQ
Dumb broker / Smart consumer	Smart broker / Dumb consumer
Streaming support	Not on it's own
External tools - Zookeeper	External tools - not needed
Supports few languages	Supports many languages

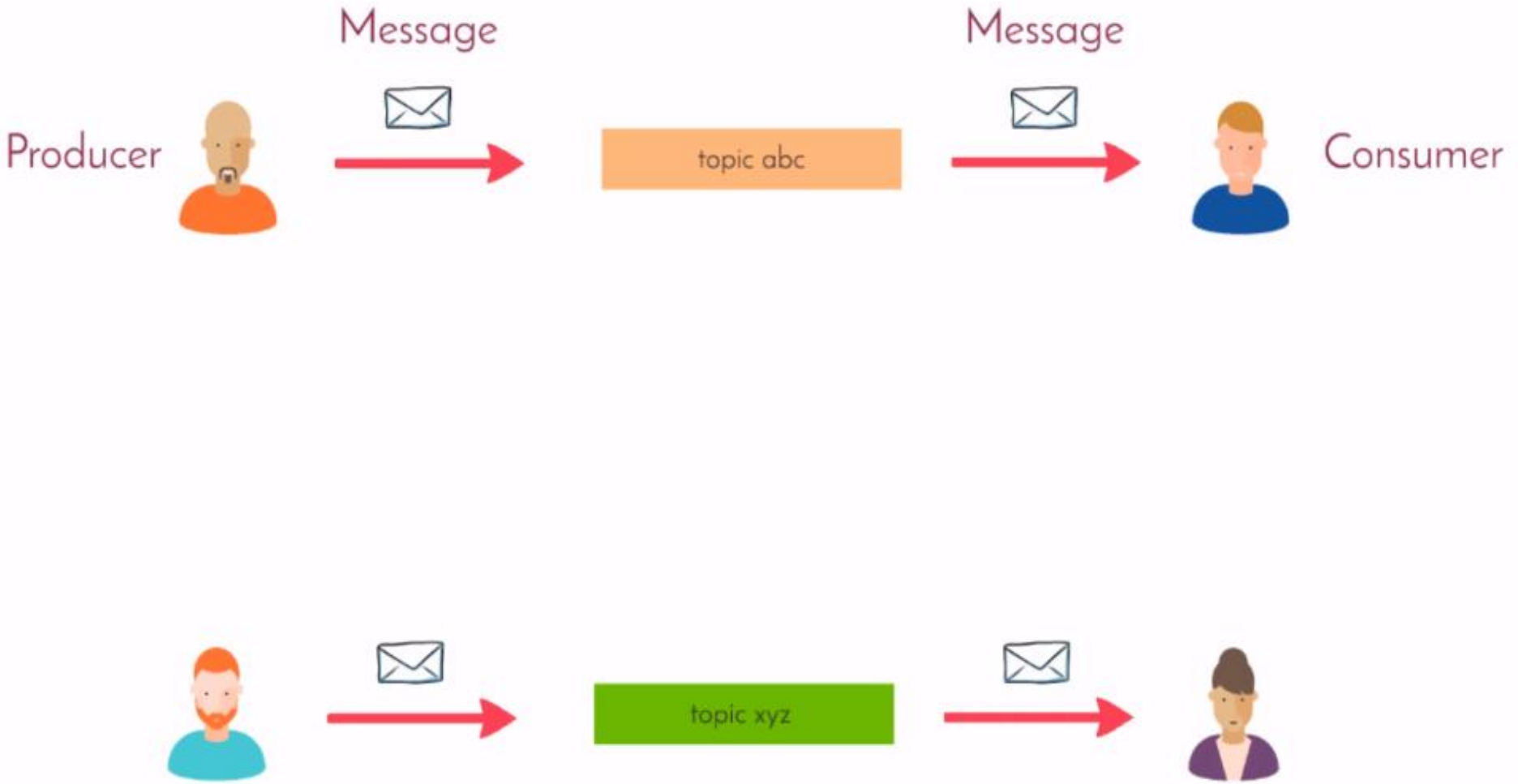
Kafka vs other MB's

kafka	RabbitMQ
Dumb broker / Smart consumer	Smart broker / Dumb consumer
Streaming support	Not on it's own
External tools - Zookeeper	External tools - not needed
Supports few languages	Supports many languages
100,000 messages per second	20,000 messages per second

Kafka Concepts

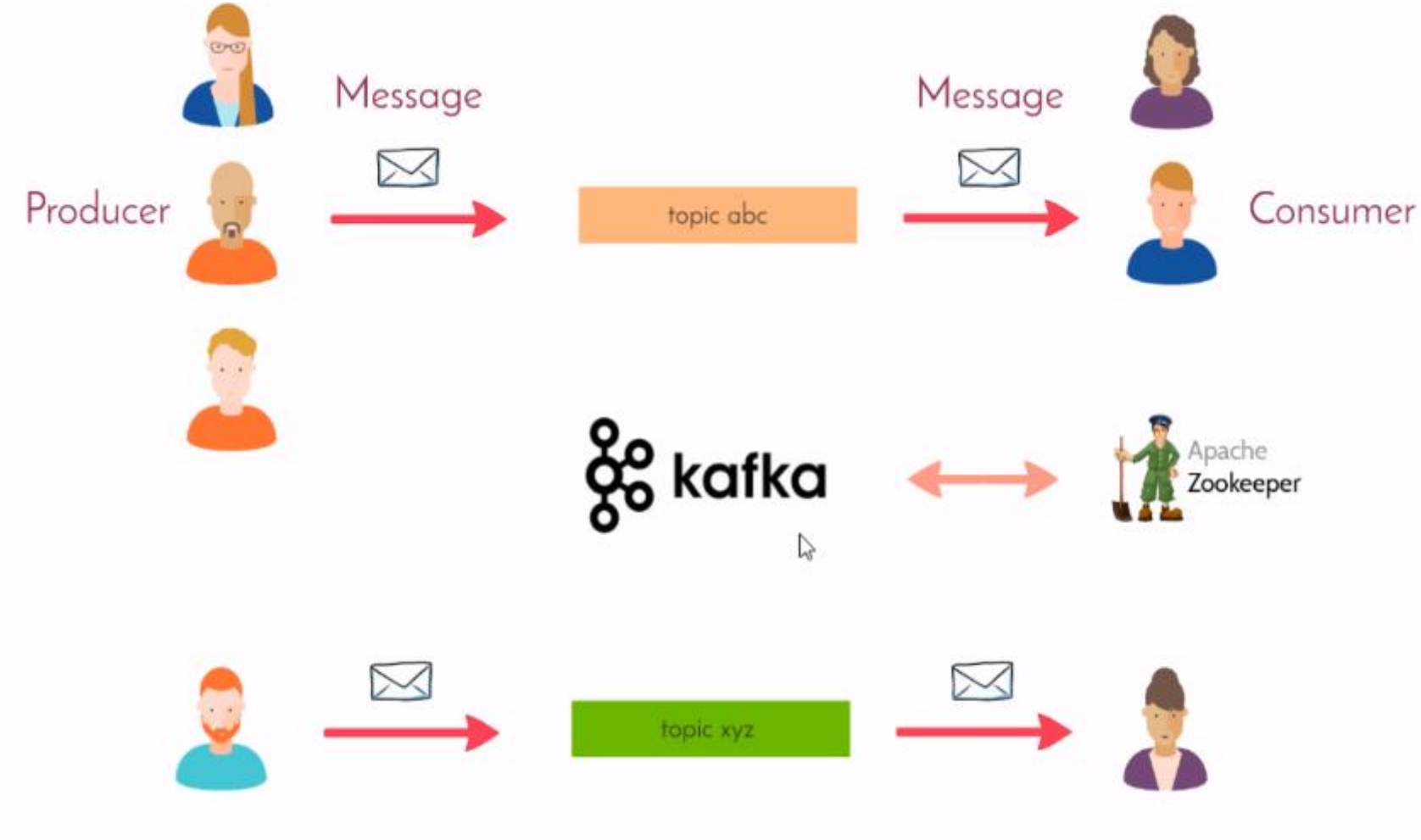
Kafka Basic Process

Kafka Basic Process



Kafka Basic Process

Kafka Basic Process



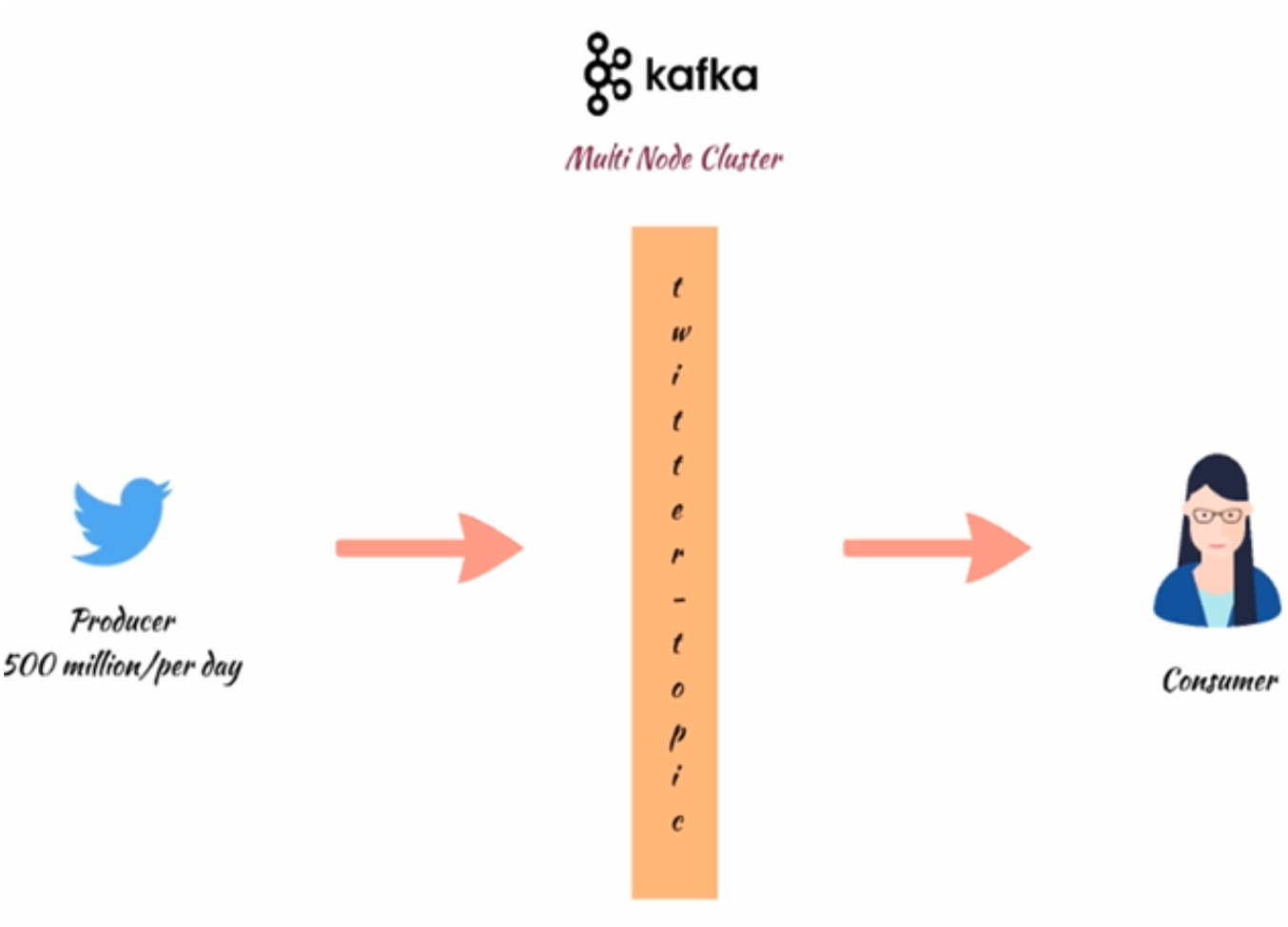
Producer, consumer, message

- A producer can be twitter producing tweets or an application or a program
  - Anything that produces a stream of message
- A consumer can be any program or application that is interested in consuming the messages produced by the producer
- A message is nothing more than a key-value pair with a time stamp
  - String, json, costum object

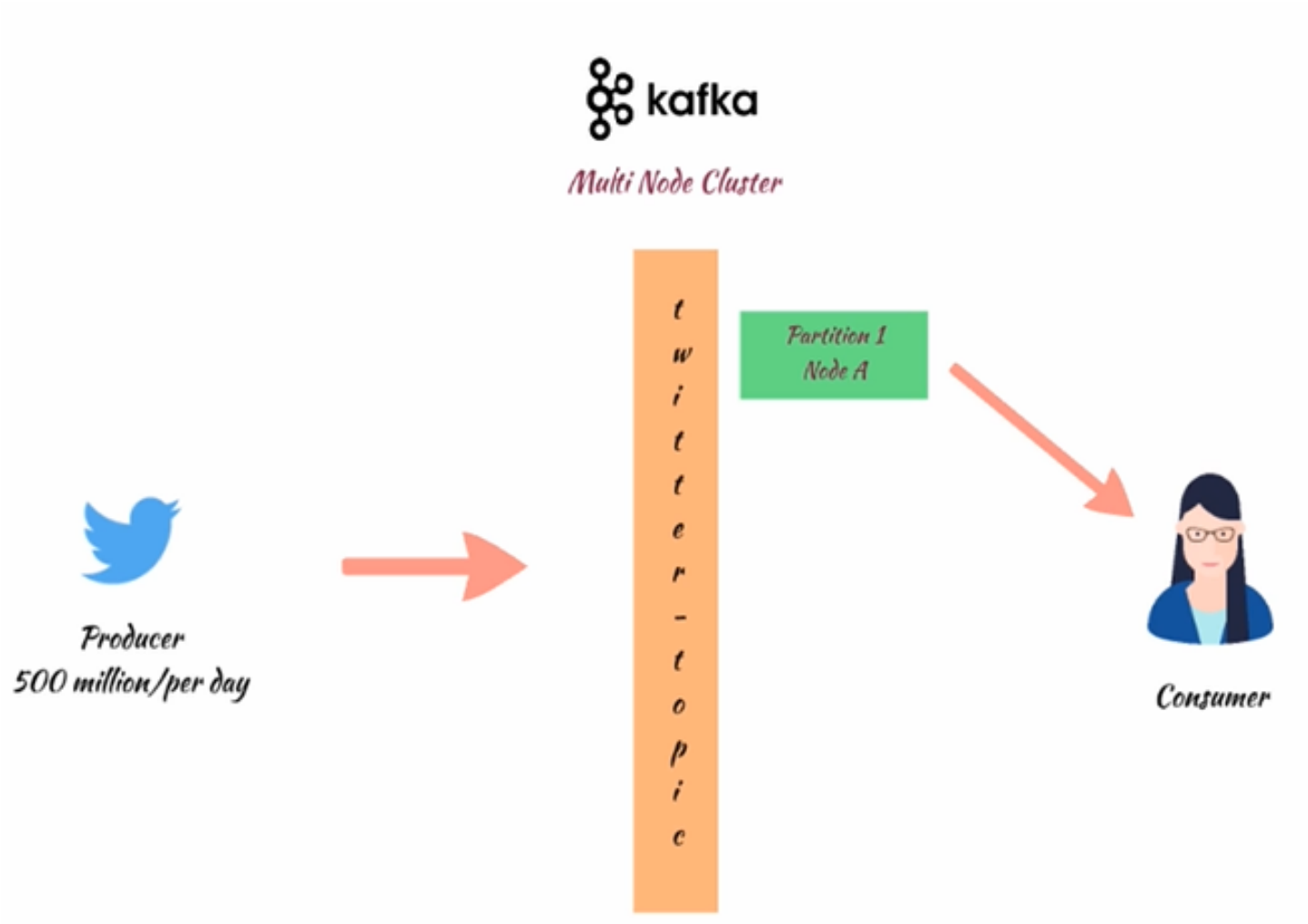
Kafka as a cluster

- Kafka is generally deployed as a cluster
  - A cluster of nodes running Kafka
  - Each node is called a broker or server
- Kafka broker is responsible for bringing the messages from the producer to the consumer ***reliably***
  - Not easy in Big Data environment
- Zookeeper helps manage the nodes
  - Which nodes are alive/dead
  - Elect leader
  - Metadata information

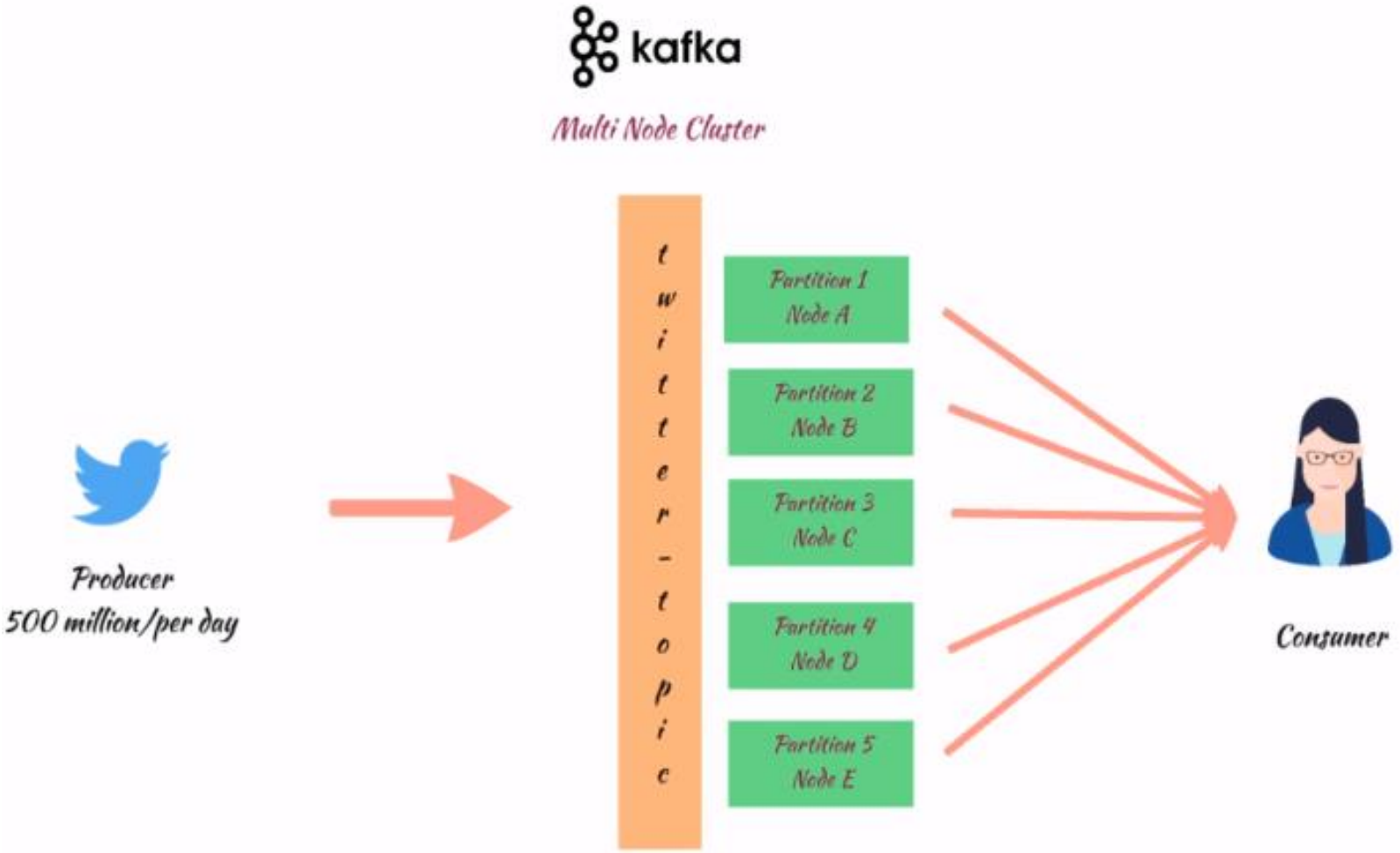
Kafka Use Case



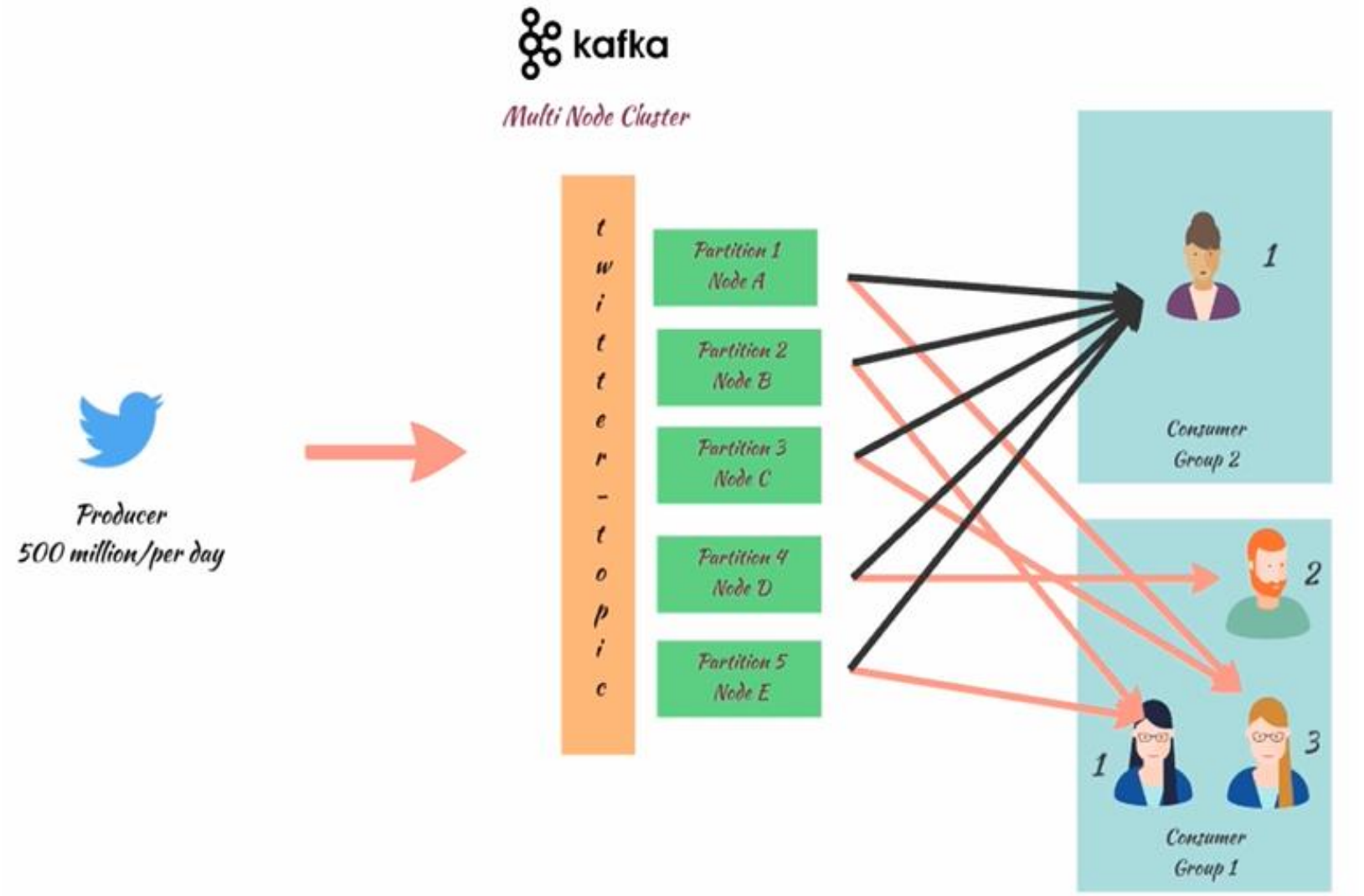
Kafka partitions



Multiple nodes



Multiple consumers

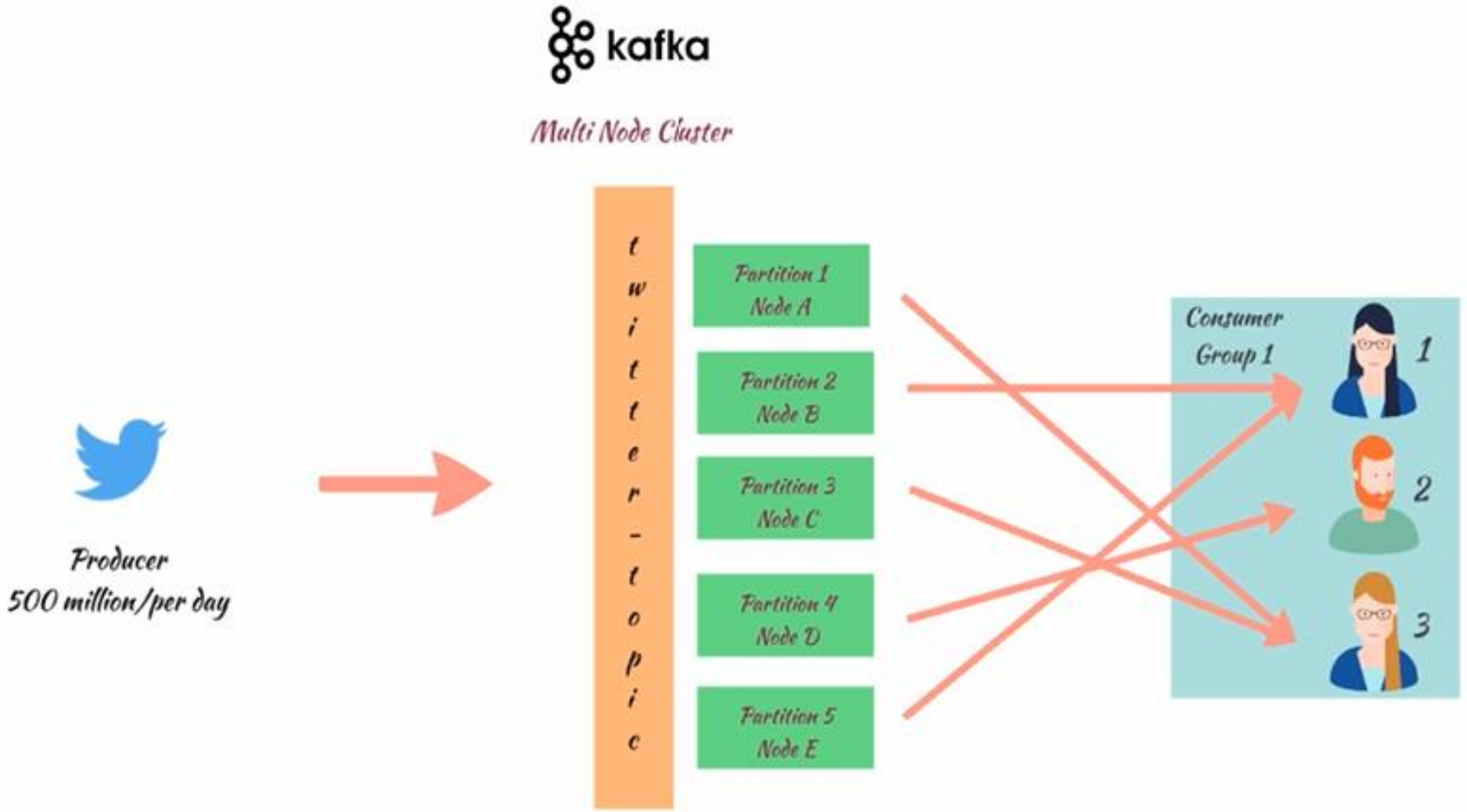


Consumer groups

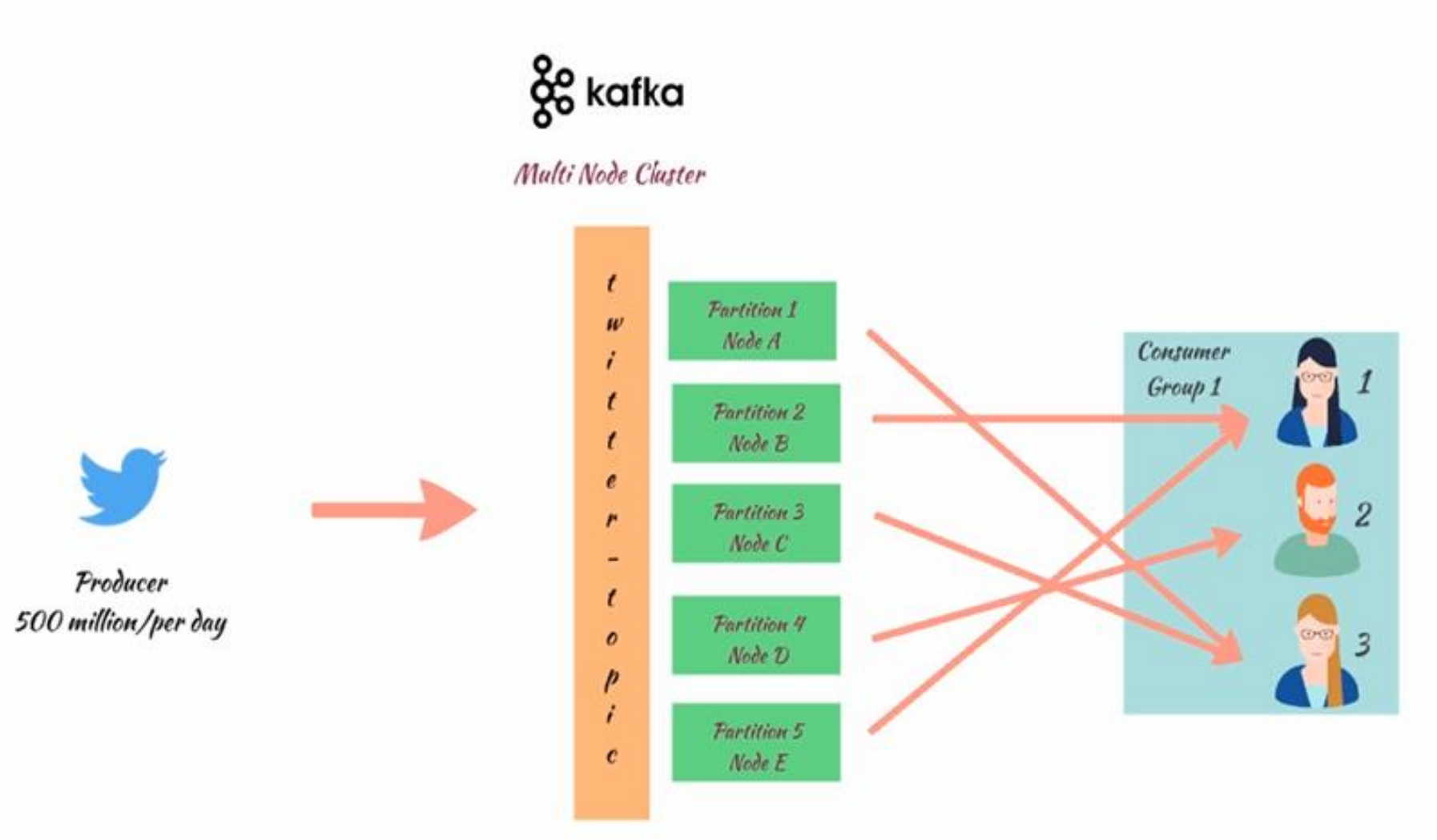
- When Kafka sees consumers from the same group, it will make sure that a single message is not sent to more than one consumer
- Every consumer is assigned to a partition
- Only one consumer can consumer from a partition
- Consumer groups are like different applications.

Reliability and Fault Tolerance in Kafka Producers and Consumers

Producer Failure



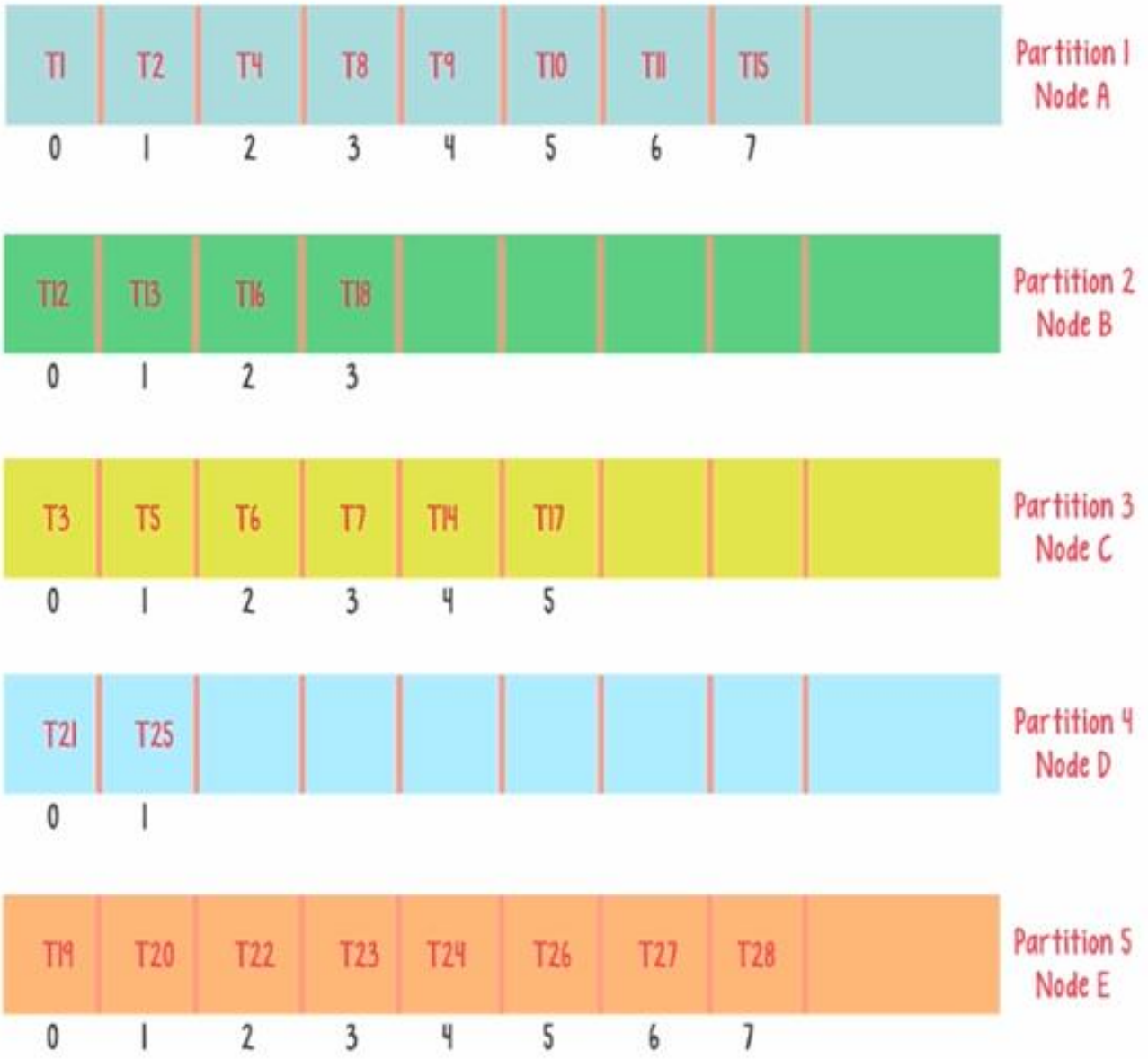
Consumer Failure



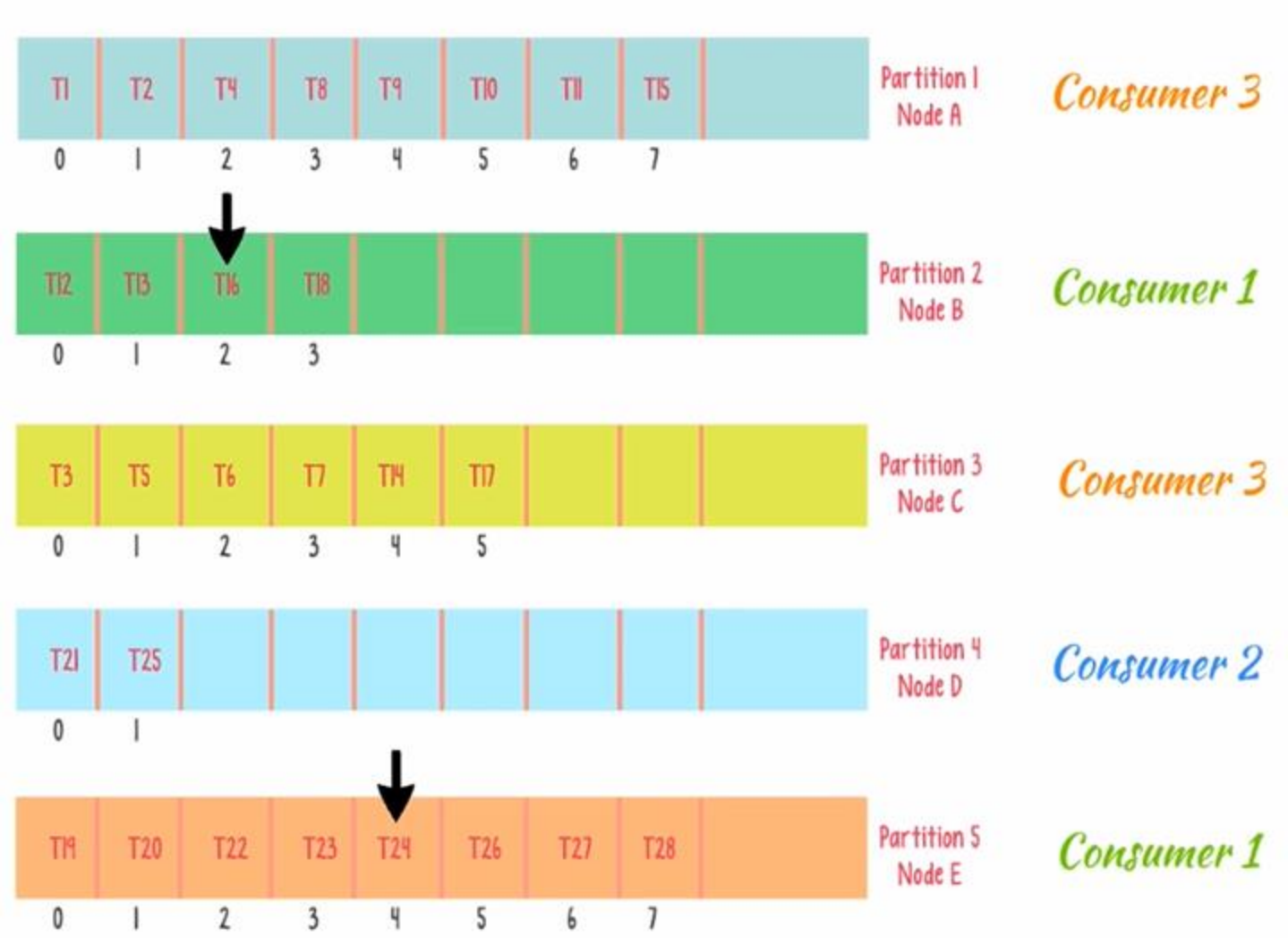
Interview Question

- Assume Consumer 1 goes down
- Does Consumer 2 start where Consumer 1 left off or start from the beginning?
  - Ideally should be from where you left off
- Follow up question:
  - How does Consumer 2 know where to start?

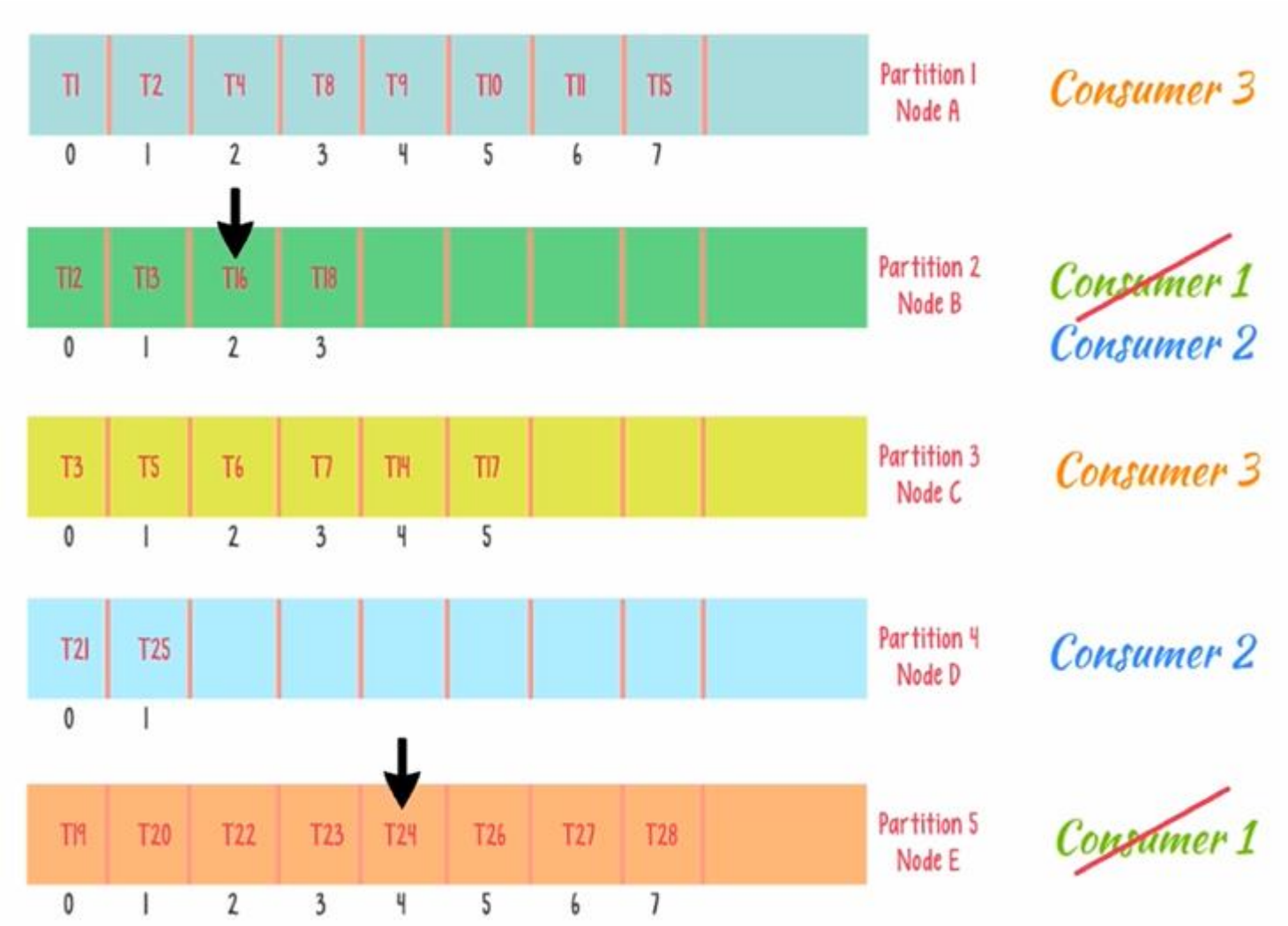
How messages are stored in partitions



How messages are stored in partitions



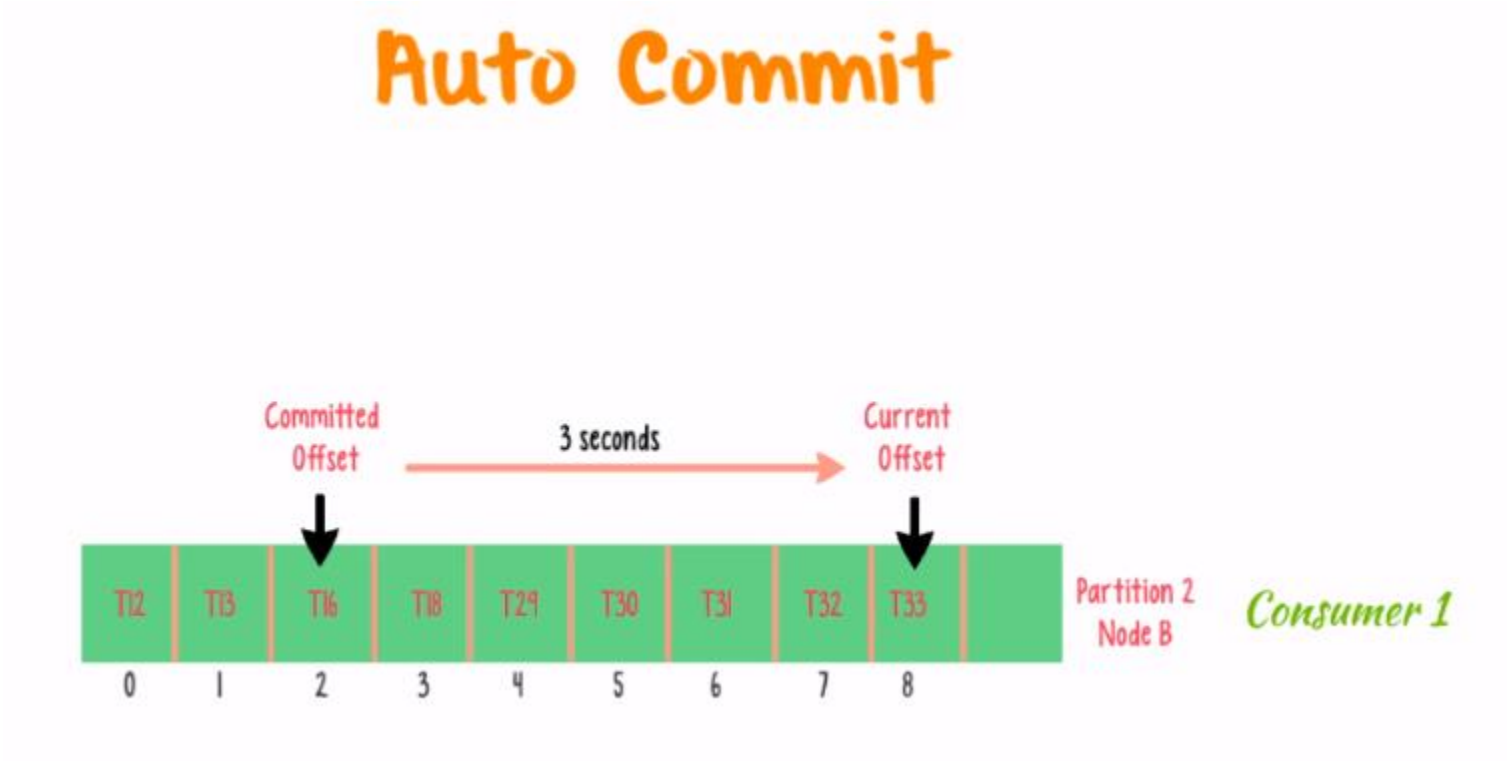
How messages are stored in partitions



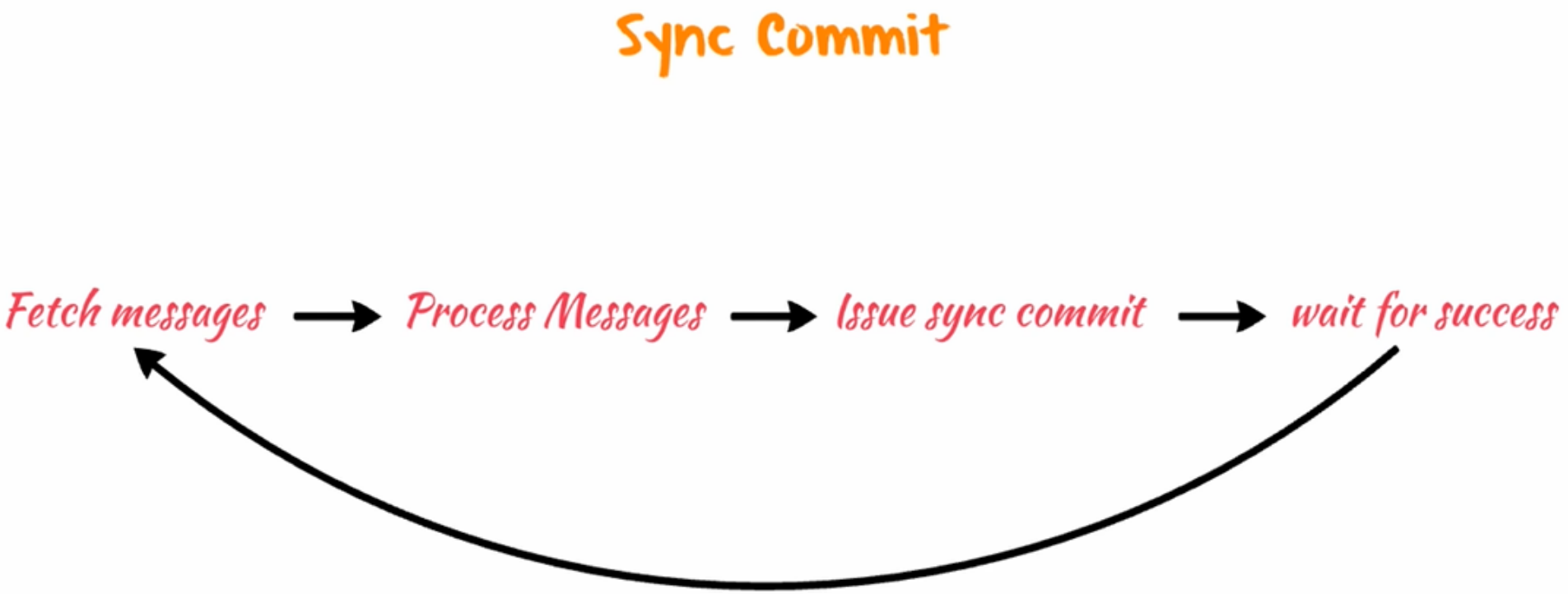
Overhead with internal topic

- Recording index every time is a huge overhead
- How to overcome this?
- Few options

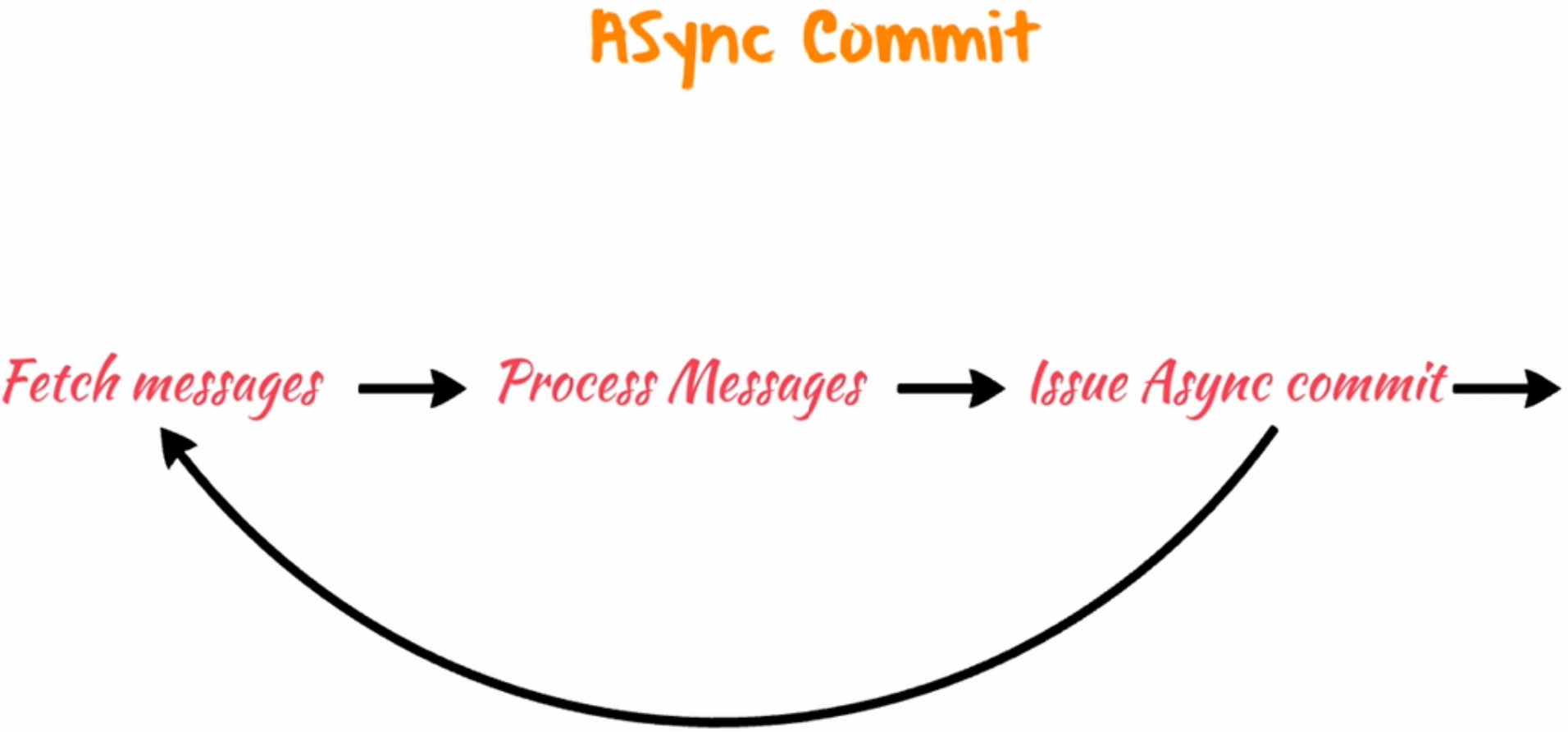
Overhead with internal topic



Overhead with internal topic



Overhead with internal topic

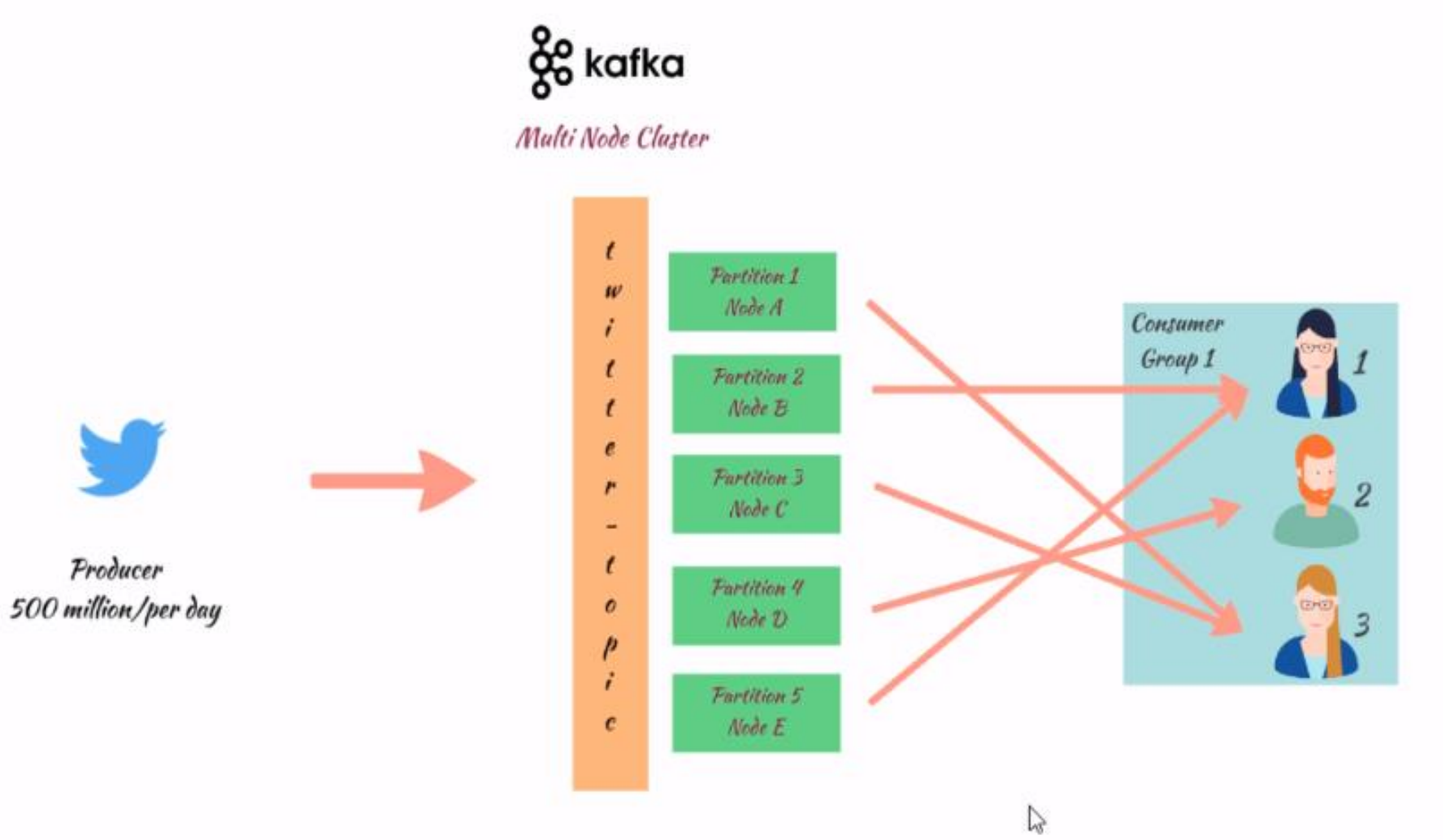


Overhead with internal topic

- Commit option you choose depends on your use case
- If you care less about duplication of data and velocity of data is light, you can go auto-commit
- If you care about data being unique and also have light velocity, use synchronous commit
- Somewhere in between choose asynchronous commit

Reliability and Fault Tolerance in Kafka Brokers

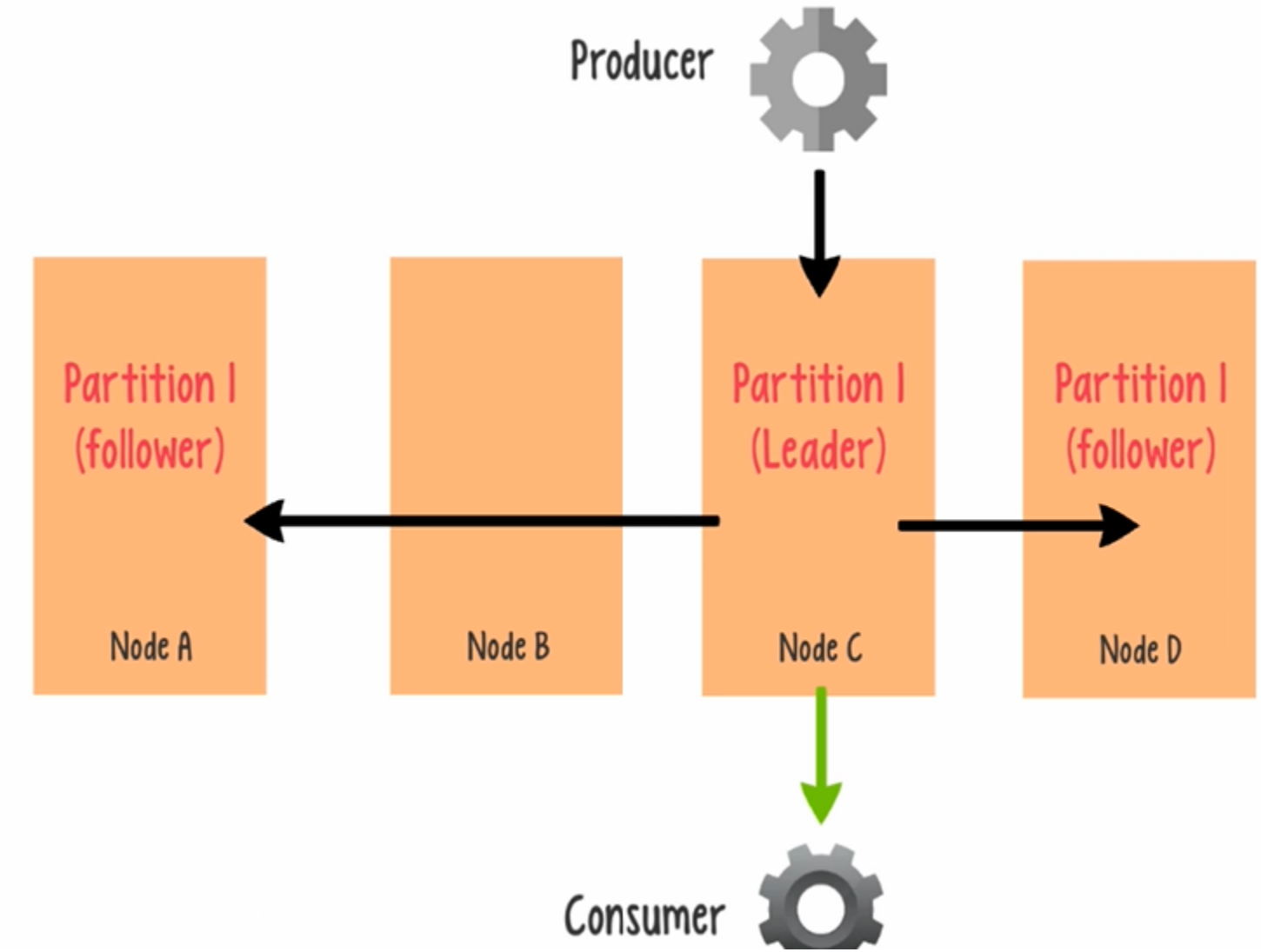
Fault tolerance in Kafka



Fault tolerance Kafka Nodes

- To tolerate failure we need repetition
- Very similar to Hadoop HDFS
  - Replication factor usually 3
  - But there is a big problem in this case
- How do we keep the replicas in sync
  - Data is constantly changing as new messages come in

Node Failure Recovery



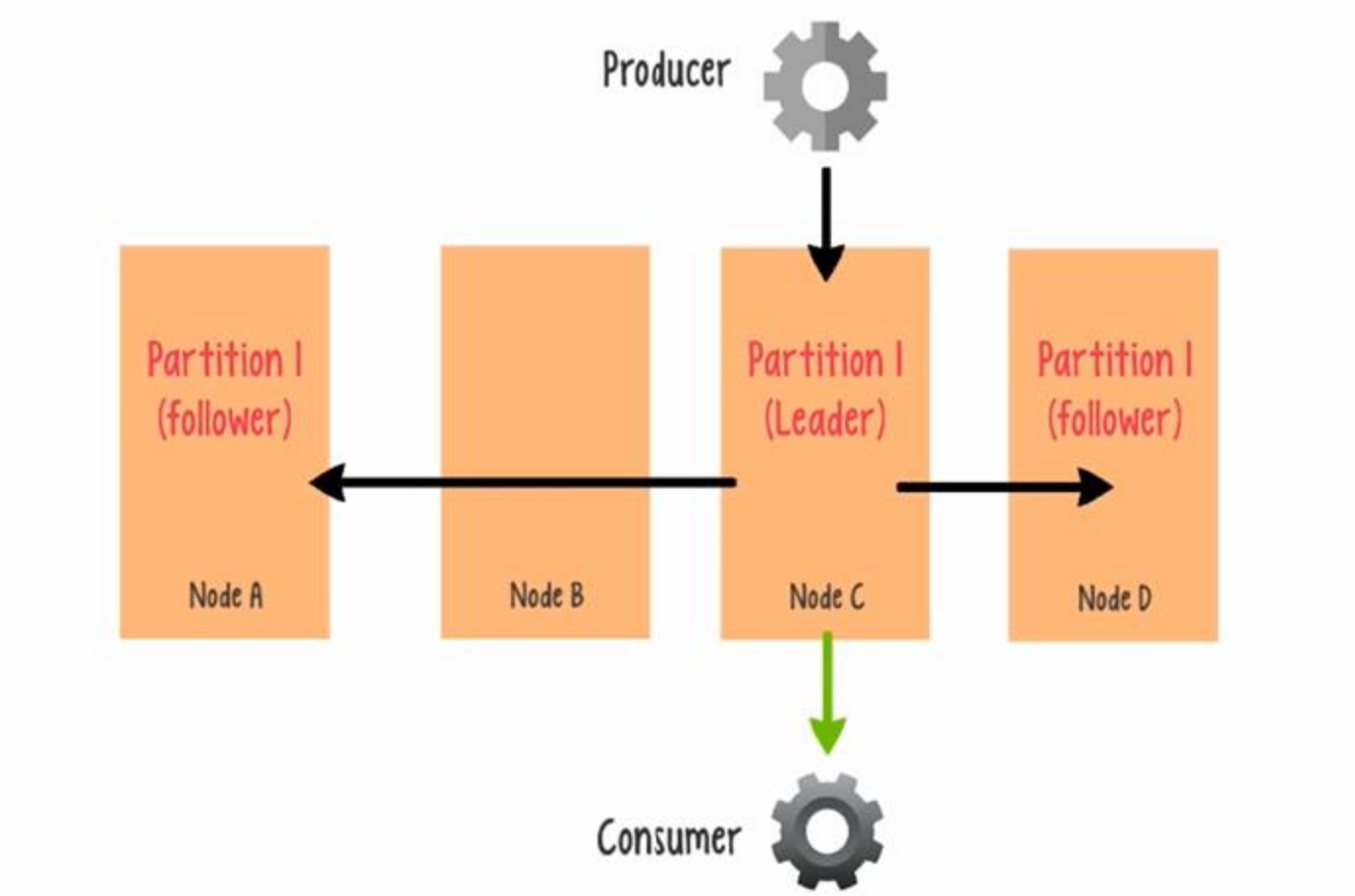
Fault tolerance Kafka Nodes

- Leader
  - Gets data from producer
  - Persists it in disk
- Follower
  - Gets data from leader
- Replicas that are in sync with the leader are called in-sync replicas (ISR's)

Fault tolerance Kafka Nodes

- It is hard for the replicas to be in-sync with the leader
  - Especially when you have high volume
- The leader gets the message first
  - So replicas can never be in-sync in real time
- So how do we achieve fault tolerance?

Fault tolerance Kafka Nodes



A message is available for a consumer to read from a partition only after the message is written or in other words committed to the in-sync replicas

Fault tolerance Kafka Nodes

replica.lag.time.max.ms = 10 sec

Stuck follower =

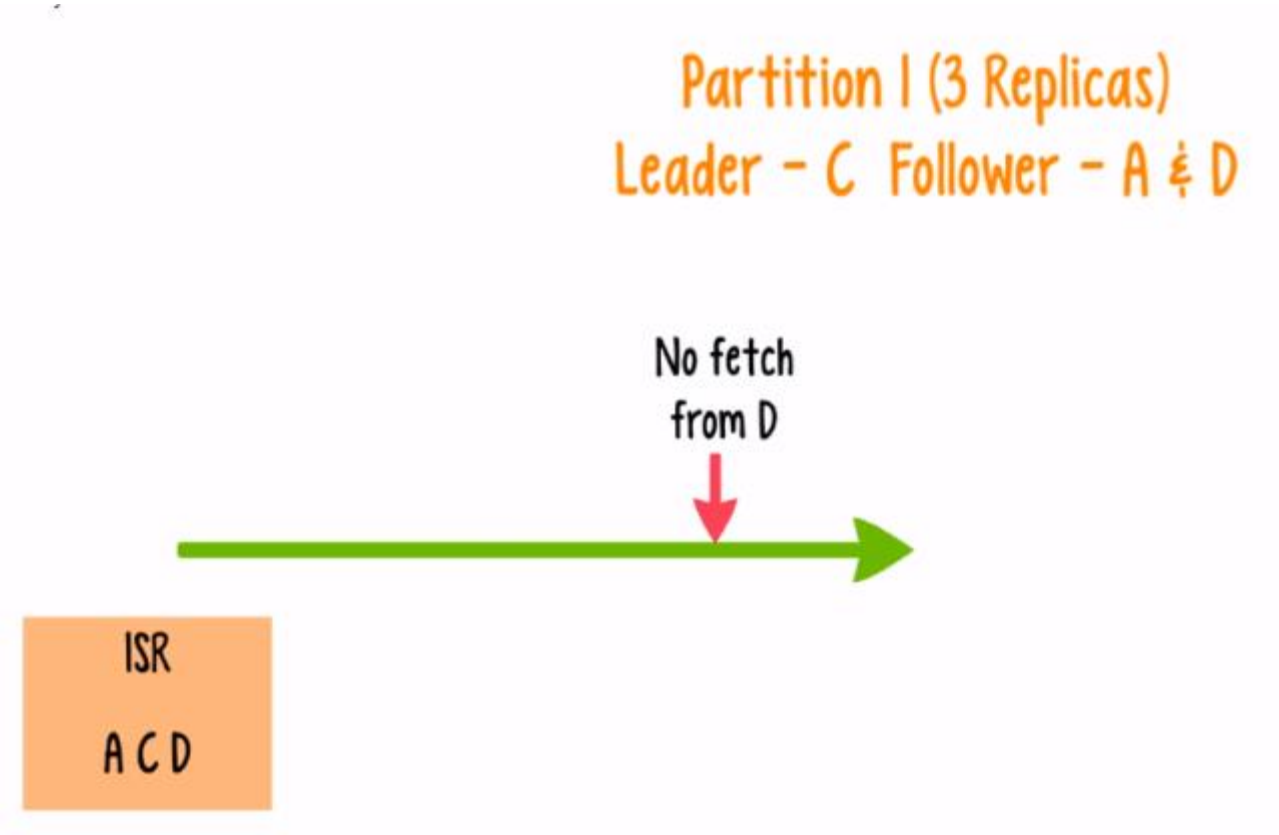
Follower did not issue a fetch request to the leader in the past 10 seconds

Slow follower =

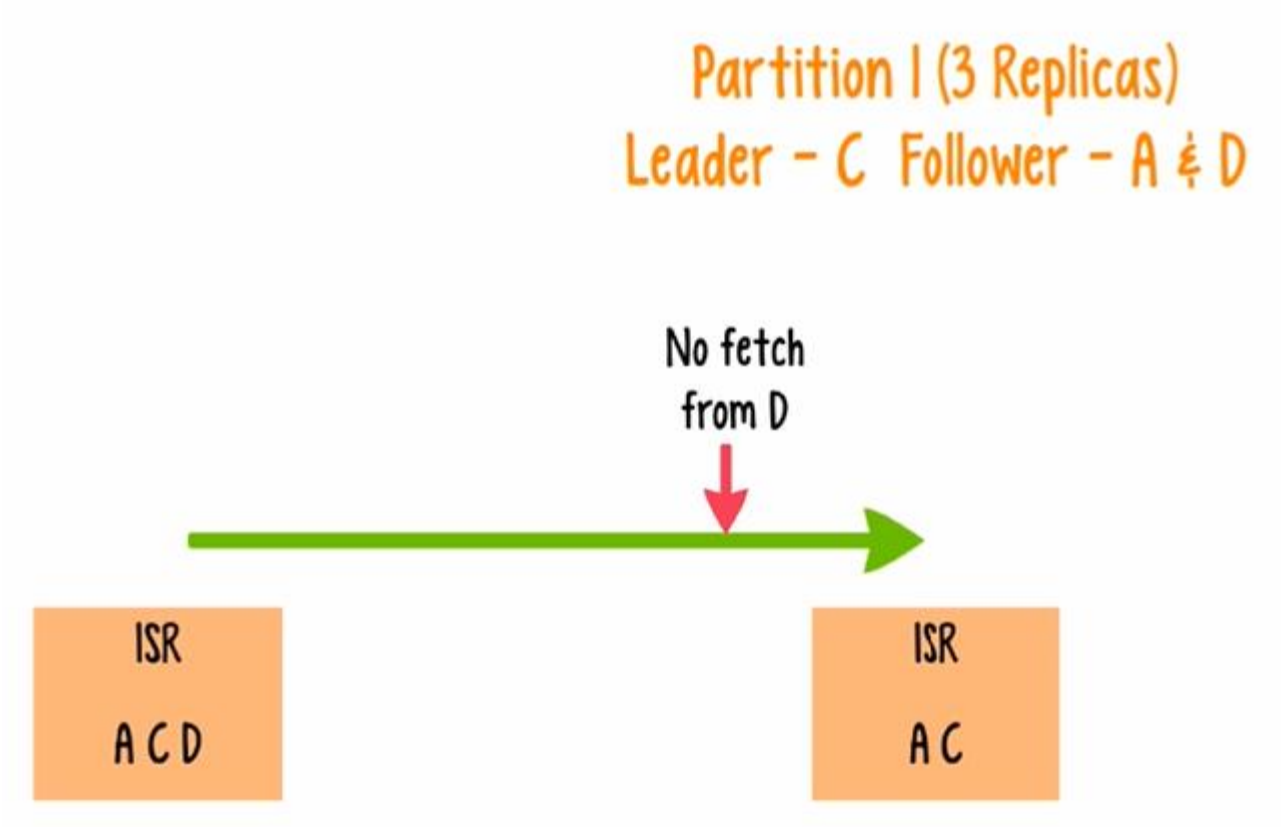
Latest message in the follower replica is older than 10 seconds compared to the leader

Concept in detail

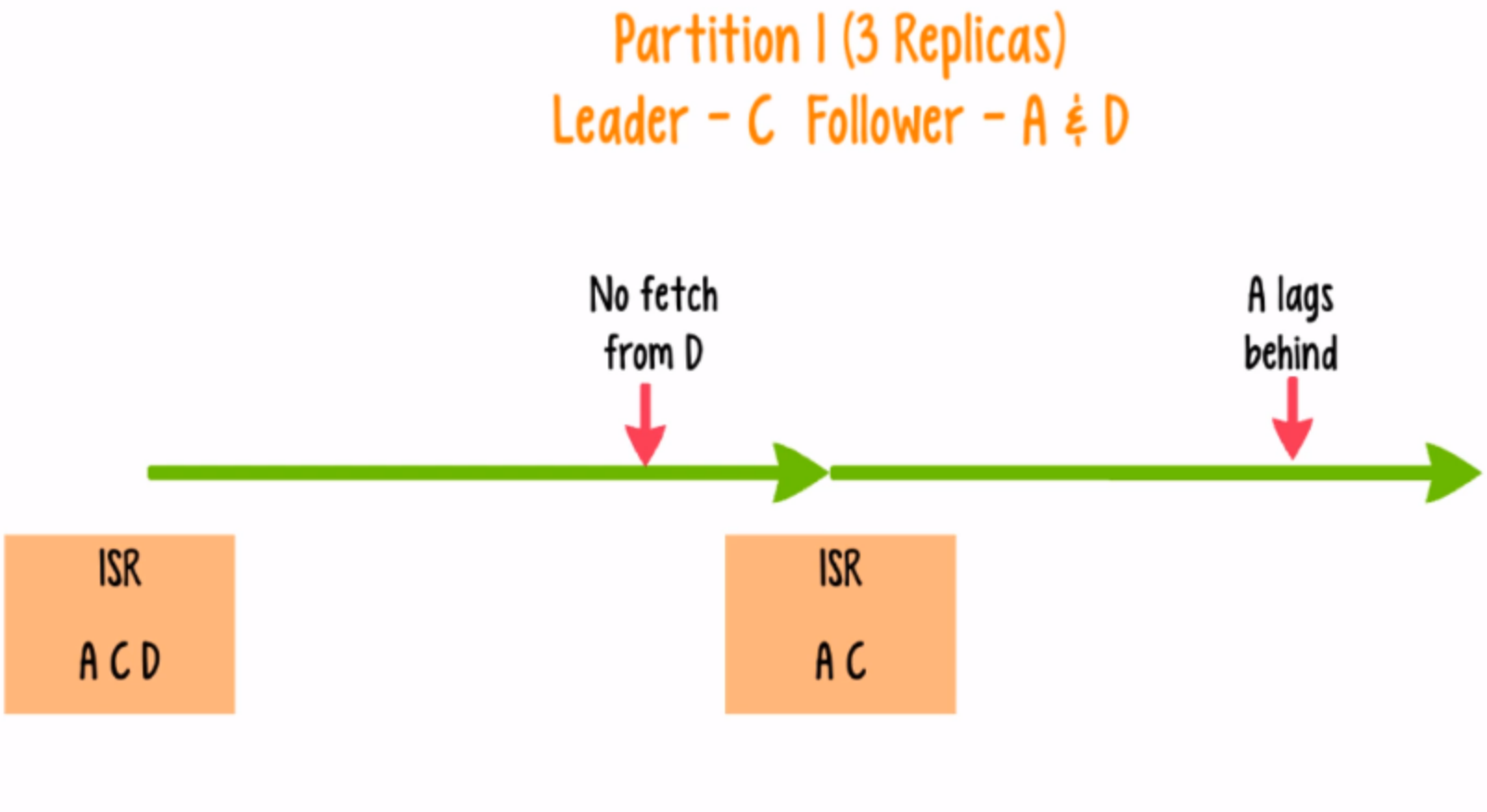
Concept in detail



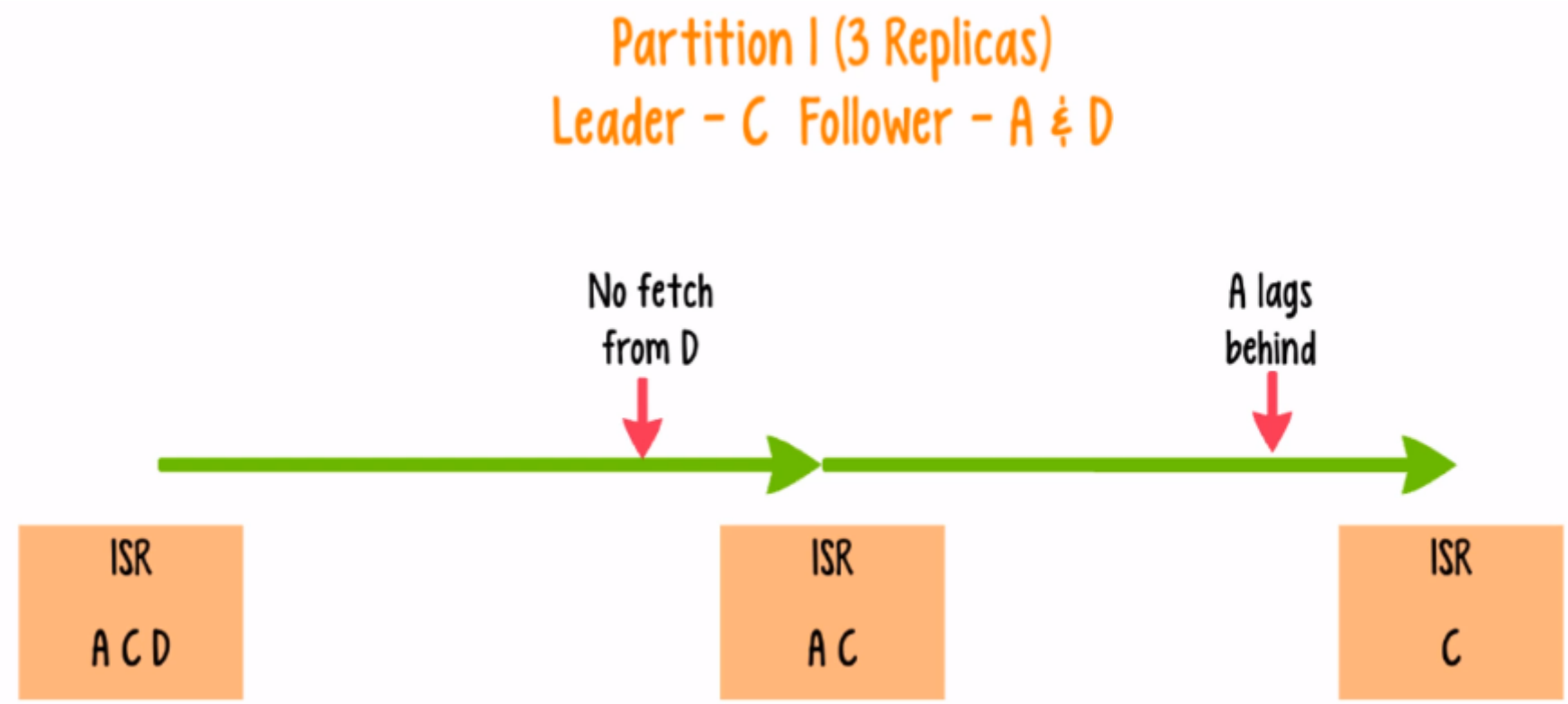
Concept in detail



Concept in detail



Concept in detail



If Data Loss is acceptable

unclean.leader.election.enable

If Data Loss is Unacceptable

min.insync.replicas

Fault tolerance Kafka Nodes

- Set value to greater than 1
- Example for 2 → if the ISR is 1, producer gets an exception and Kafka will stop taking messages
  - Give administrators chance to see what's wrong
- Usually we have a few nodes going in and out of ISR lists for short periods of time
- Setting value to 2 is a good option

Kafka Installations

Kafka Instruction Details

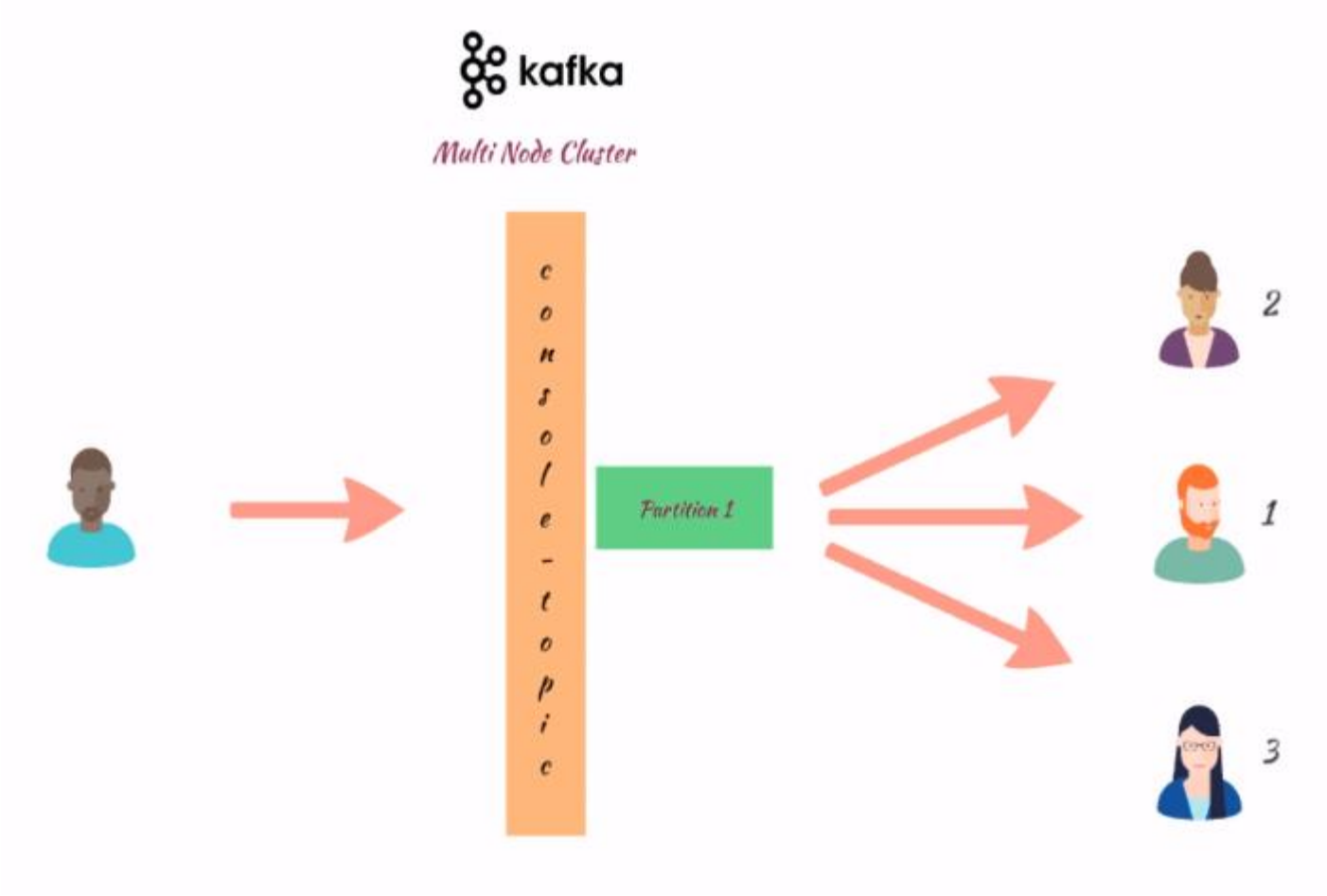
- We need to install and configure both Kafka and Zookeeper
- We will use a pre-configured version of Kafka from a company called Confluent
  - Similar to Cloudera
  - Started by engineers in LinkedIn

Kafka Experimentation

Kafka Instruction Details

- We will try to demonstrate all of the concepts in this lecture using Kafka and Zookeeper installation

Three consumers



Three consumers