Kafka

By
ADITYA PRABHAKARA

Introducing Myself

Aditya S P (sp.aditya@gmail.com)
Freelance trainer and technologist

Boring Stuff about me:

- 15+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

Interesting Things about me:

- Actually Nothing !

My Current work

To create a real time score rendering system for a financial company

- ✓Platform - AWS
- ✓Architecture - Serverless
- ✓AWS Services - Glue, Lambda, API Gateway, RDS, Amazon ML, SageMaker
- ✓Programming Language - Python
- ✓Algorithms - Regression

Agenda

1) Kafka - Introduction
2) Kafka cli
3) Kafka topics, producers, consumer
4) Kafka consumer groups
5) Kafka Streaming

Lots of handson

Kafka Introduction

By
ADITYA PRABHAKARA

What is Kafka

- Distributed Streaming Platform
- Publish and Subscribe to streams of records
- Fault tolerant storage
- Replicates Topic Log Partitions to multiple servers
- Process records as they occur
- Fast, efficient IO, batching, compression, and more
- Used to decouple data streams Kafka

Who is using Kafka

- **Netflix** uses Kafka to apply recommendations in real-time while you're watching TV shows
- **Uber** uses Kafka to gather user, taxi and trip data in real-time to compute and forecast demand, and compute surge pricing in real-time
- **LinkedIn** uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real time.

What is Kafka

- Kafka cluster retains all published records
  - Time based – configurable retention period
  - Size based - configurable based on size
  - Compaction - keeps latest record
- Retention policy of three days or two weeks or a month
- It is available for consumption until discarded by time, size or compaction
- Consumption speed not impacted by size

Kafka's exponential growth

- Kafka growth exploding
- 1/3 of all Fortune 500 companies
- Top ten travel companies, 7 of top ten banks, 8 of top ten insurance companies, 9 of top ten telecom companies
- LinkedIn, Microsoft and Netflix process 4 comma message a day with Kafka (1,000,000,000,000)
- Real-time streams of data, used to collect big data or to do real time analysis (or both)

Kafka's need

- Real time streaming data processed for real time analytics
  - Service calls, track every call, IOT sensors
  - Apache Kafka is a fast, scalable, durable, and fault tolerant publish-subscribe messaging system
- Kafka is often used instead of JMS, RabbitMQ and AMQP
  - higher throughput, reliability and replication

Contd…

- Kafka can works in combination with Flume/Flafka, Spark Streaming, Storm, HBase and Spark for real-time analysis and processing of streaming data
- Feed your data lakes with data streams
- Kafka brokers support massive message streams for follow up analysis in Hadoop or Spark
- Kafka Streaming (subproject) can be used for real-time analytics

Kafka's speed

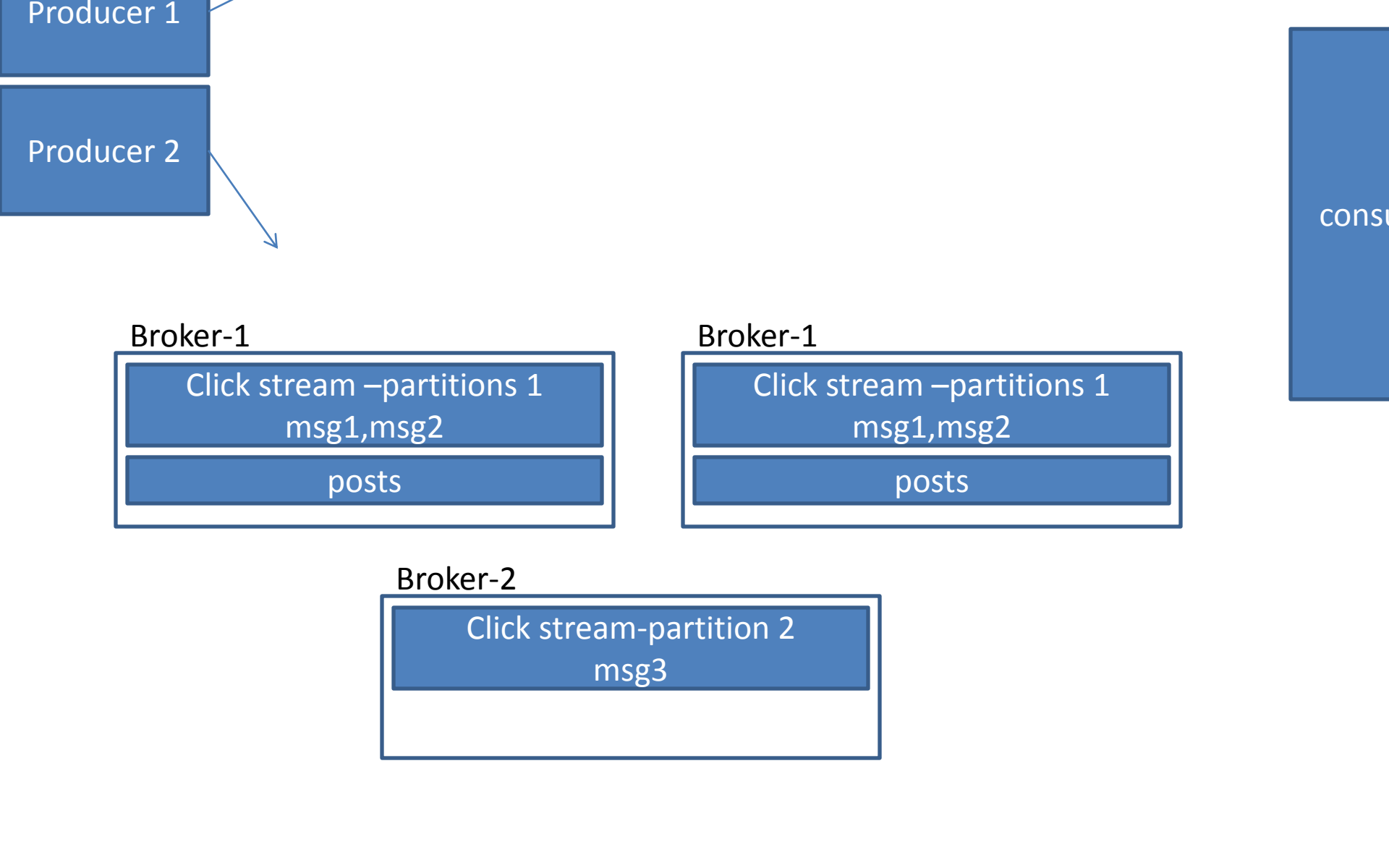
- **Zero Copy - calls the OS kernel direct rather to move data fast**
- **Batch Data in Chunks - Batches data into chunks**
  - end to end from Producer to file system to Consumer
  - Provides More efficient data compression. Reduces I/O latency
- **Sequential Disk Writes - Avoids Random Disk Access**
  - writes to immutable commit log. No slow disk seeking. No random I/O operations. Disk accessed in sequential manner
- **Horizontal Scale - uses 100s to thousands of partitions for a single topic**
  - spread out to thousands of servers
  - handle massive load

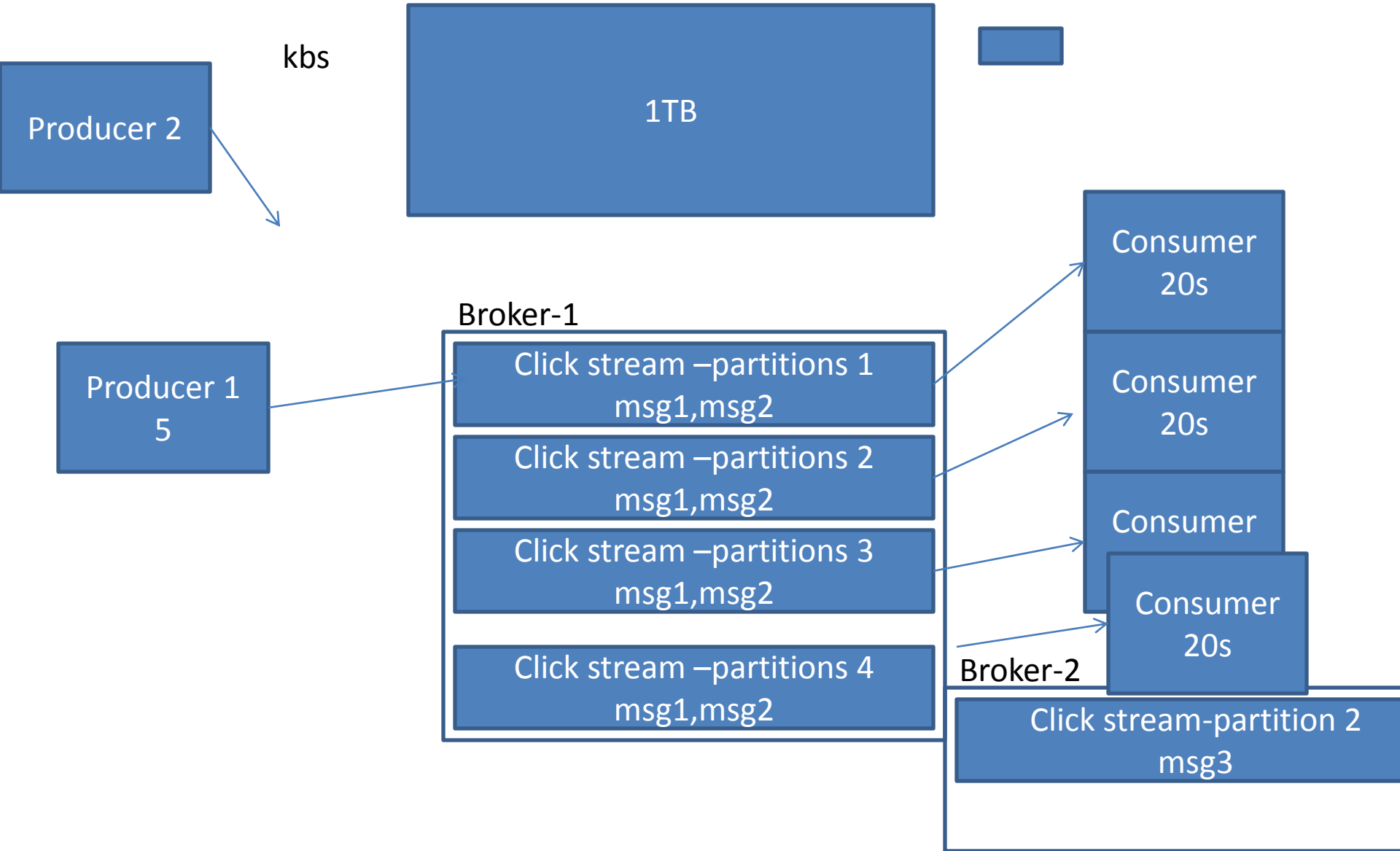
Topics

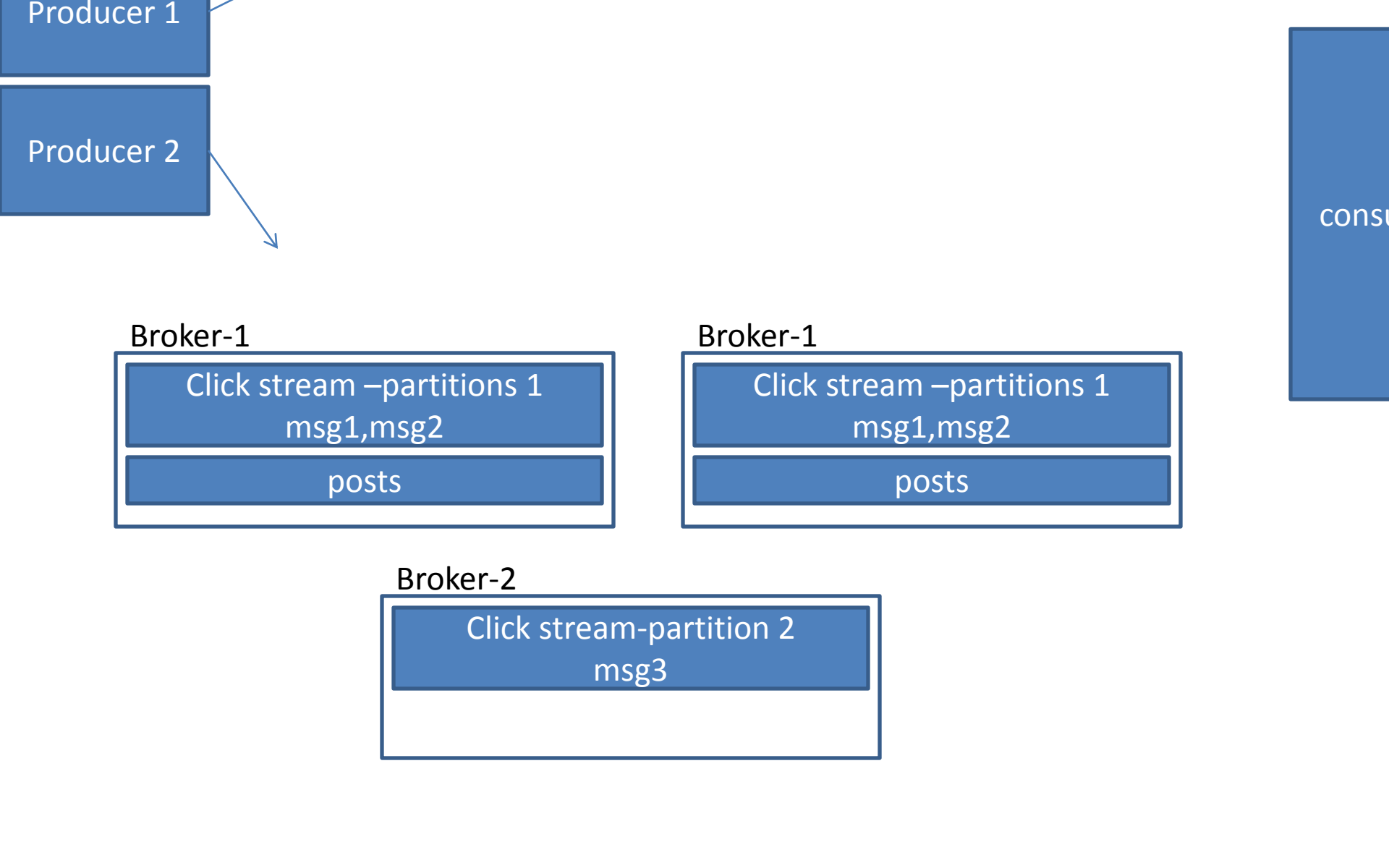
By
ADITYA PRABHAKARA

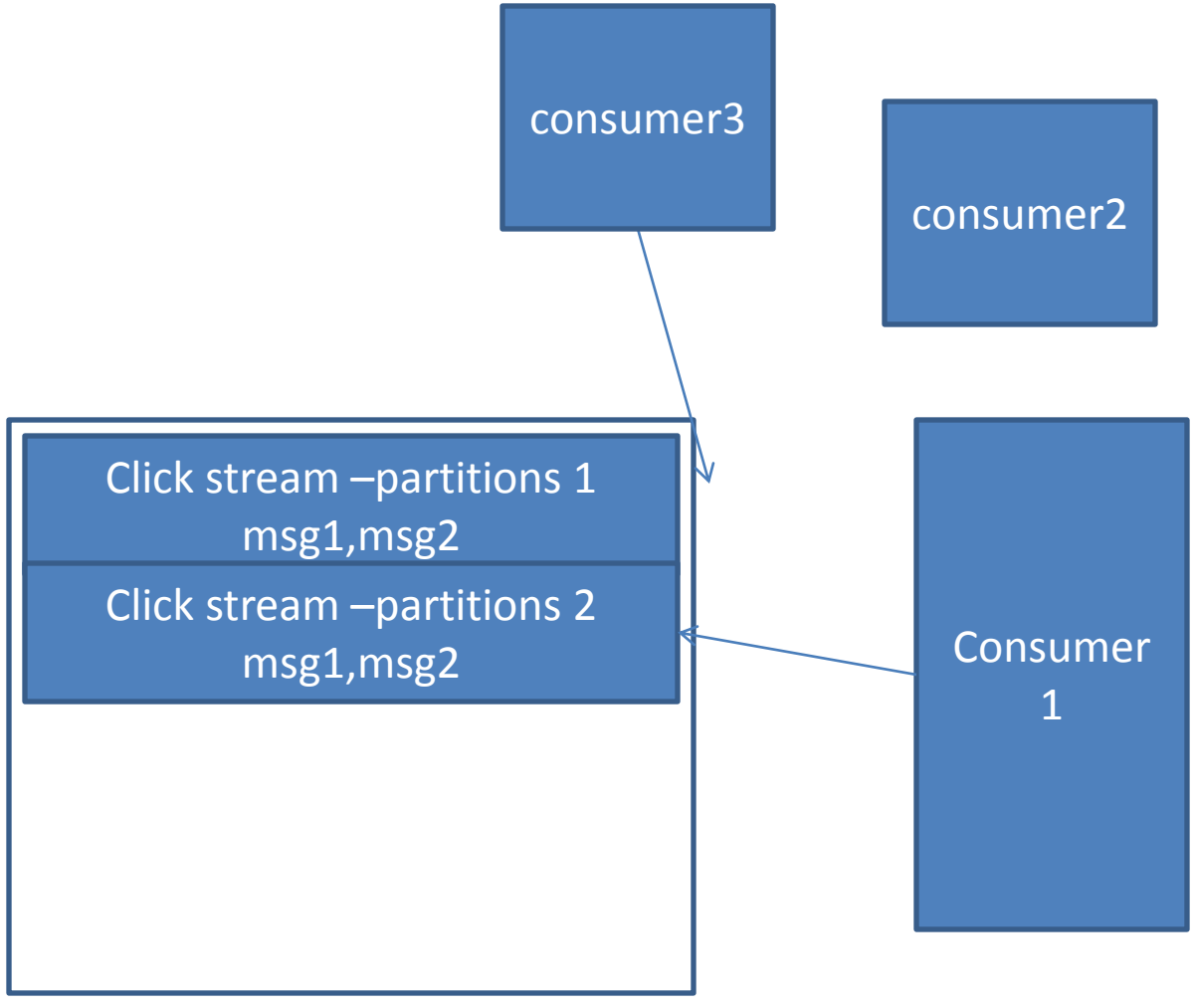
Broker

- A Kafka cluster is composed of multiple brokers (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is 3 brokers, but some big clusters have over 100 brokers
- In these examples we choose to number brokers starting at 100 (arbitrary)


Producer 1
Producer 2
cons
Broker-1
Click stream –partitions 1
msg1,msg2
posts
Broker-1
Click stream –partitions 1
msg1,msg2
posts
Broker-2
Click stream-partition 2
msg3

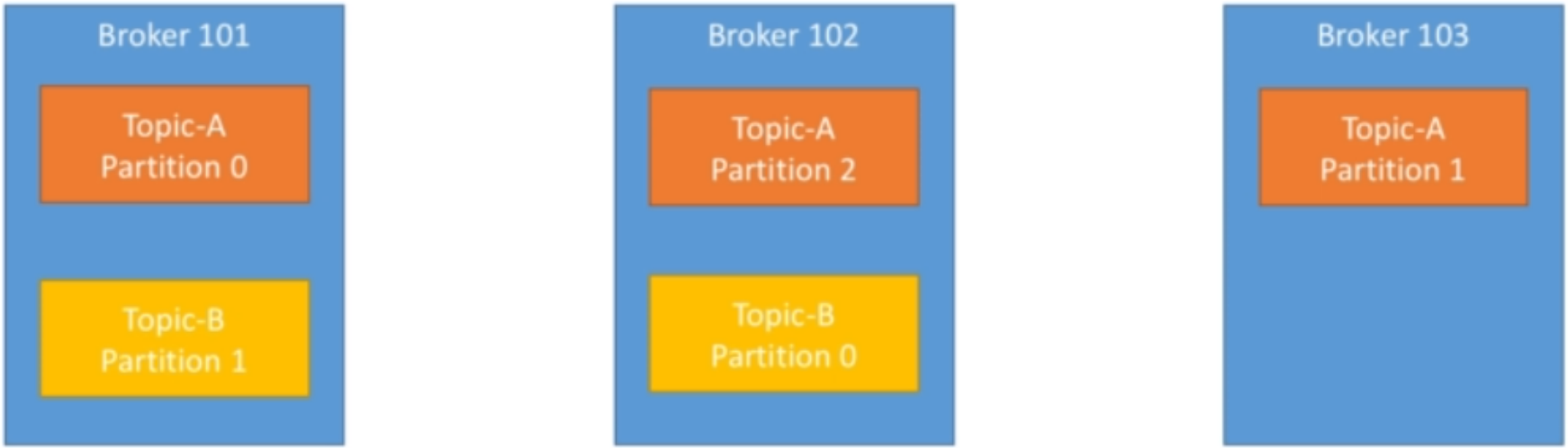

kbs
1TB
Producer 2
Producer 1
5
Broker-1
Click stream –partitions 1
msg1,msg2
Click stream –partitions 2
msg1,msg2
Click stream –partitions 3
msg1,msg2
Click stream –partitions 4
msg1,msg2
Consumer
20s
Consumer
20s
Consumer
Consumer
20s
Broker-2
Click stream-partition 2
msg3


Producer 1
Producer 2
cons
Broker-1
Click stream –partitions 1
msg1,msg2
posts
Broker-1
Click stream –partitions 1
msg1,msg2
posts
Broker-2
Click stream-partition 2
msg3


consumer3
consumer2
Click stream –partitions 1
msg1,msg2
Click stream –partitions 2
msg1,msg2
Consumer
1

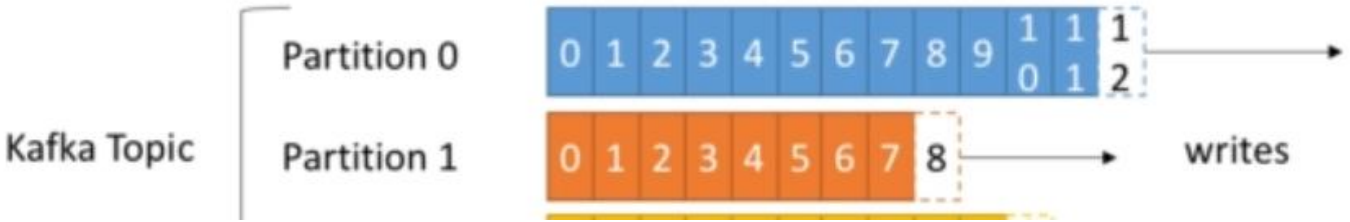
Broker and Topics

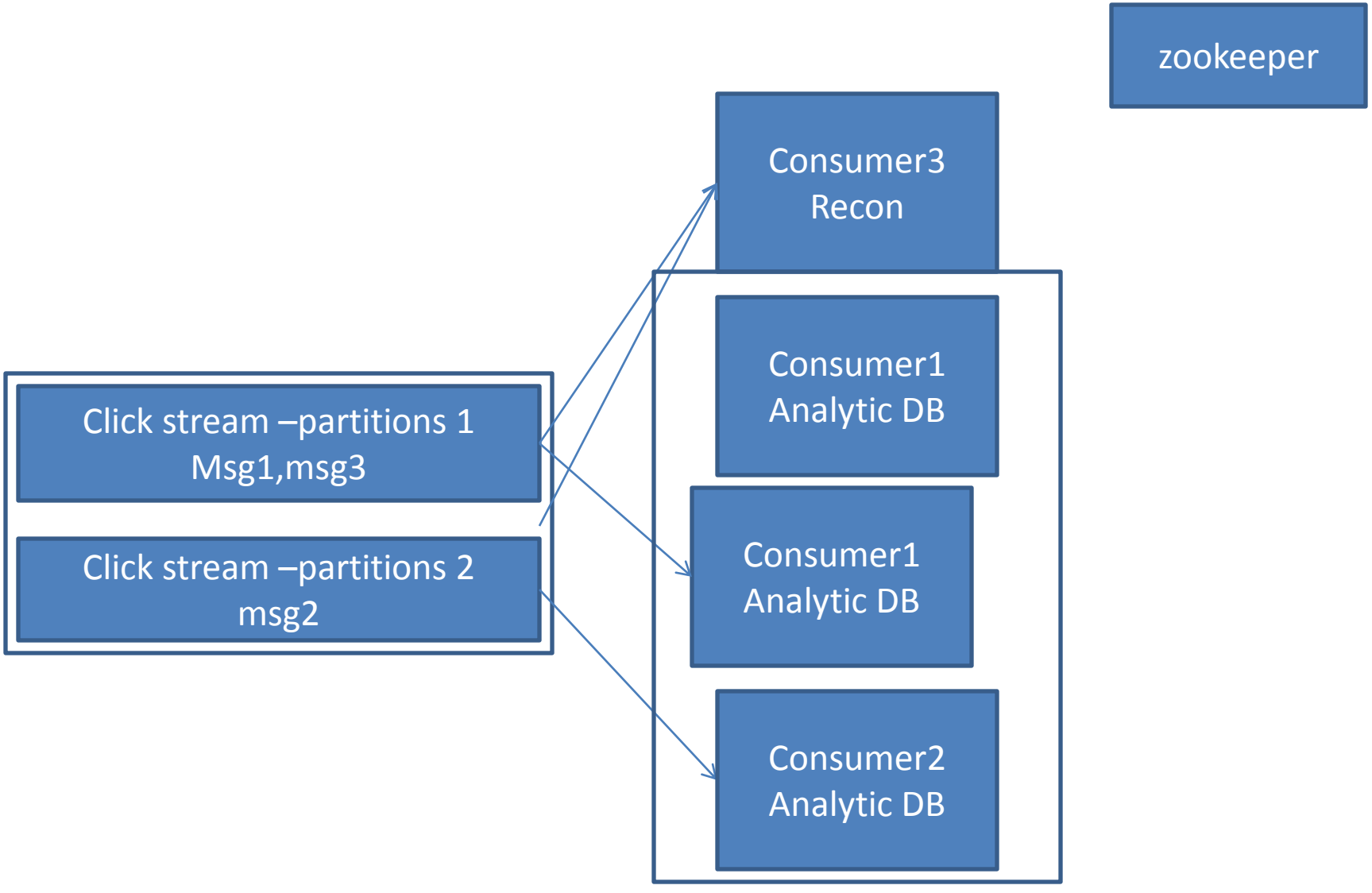
- Example of **Topic-A** with **3 partitions**
- Example of **Topic-B** with **2 partitions**



Topic

- Topics: a particular stream of data
  - Similar to a table in a database (without all the constraints)
  - You can have as many topics as you want
  - A topic is identified by its name
- Topics are split in partitions
  - Each partition is ordered
  - Each message within a partition gets an incremental id, called offset



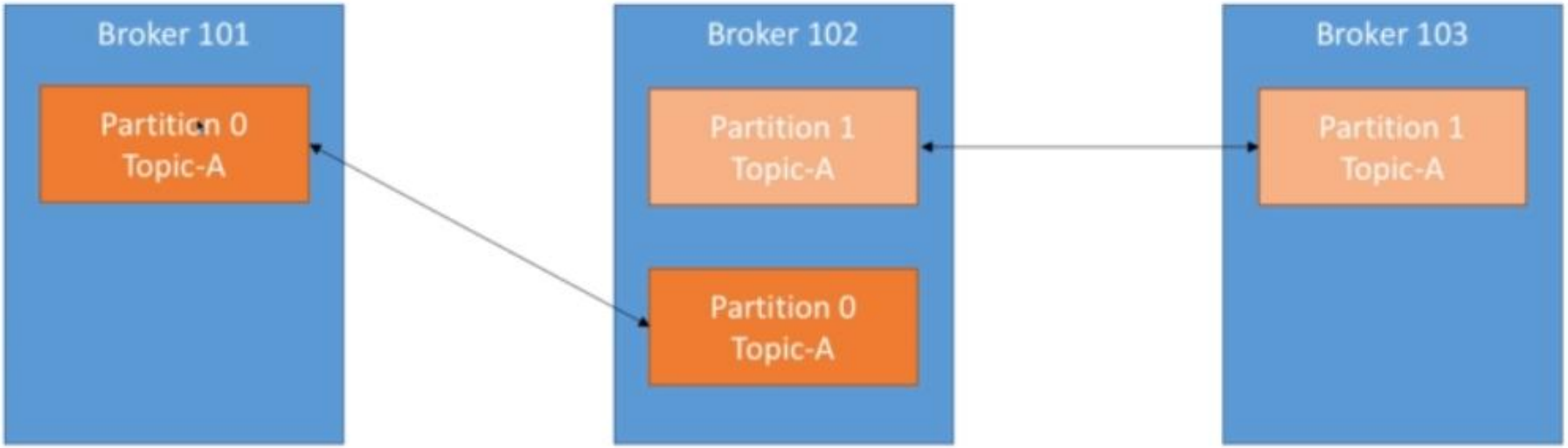

zookeeper
Consumer3
Recon
Consumer1
Analytic DB
Consumer1
Analytic DB
Consumer2
Analytic DB
Click stream –partitions 1
Msg1,msg3
Click stream –partitions 2
msg2

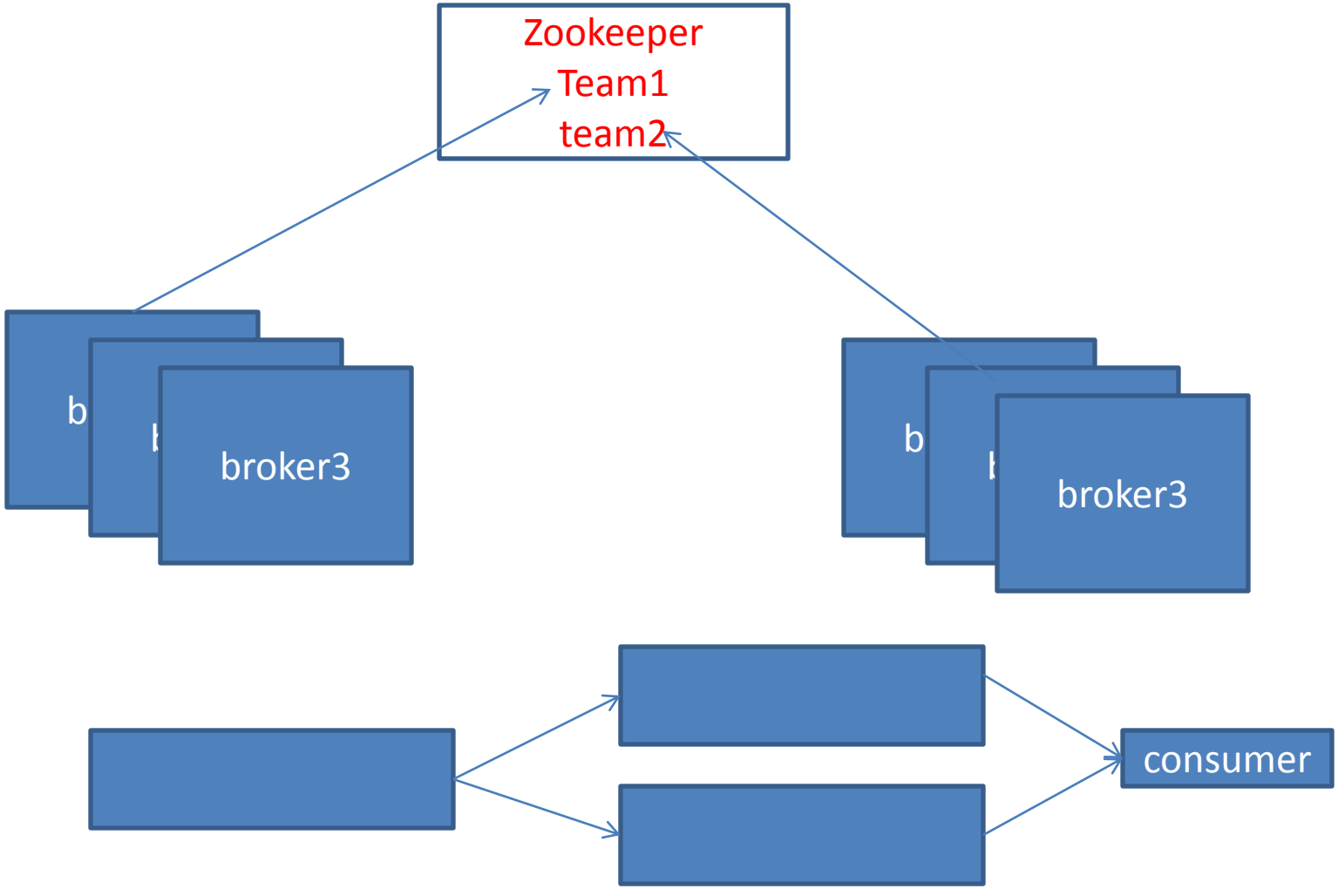
Topic

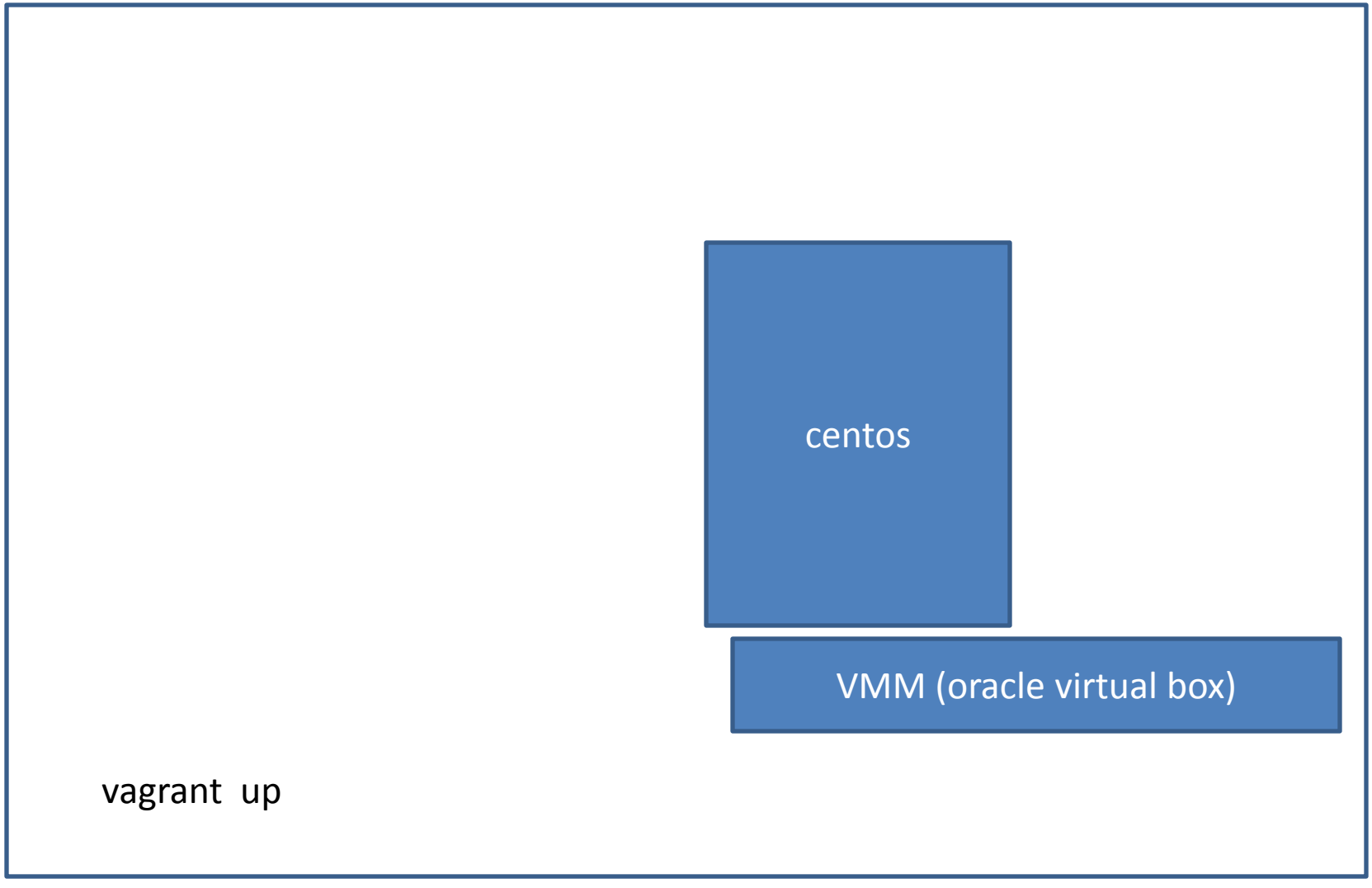
- Offset only have a meaning for a specific partition.
  - E.g. offset 3 in partition 0 doesn't represent the same data as offset 3 in partition 1
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is one week)
- Once the data is written to a partition, it can't be changed (immutability)
- Data is assigned randomly to a partition unless a key is provided (more on this later)

Replication

- Topics should have a replication factor > 1 (usually between 2 and 3)
- This way if a broker is down, another broker can serve the data
- Example: Topic-A with 2 partitions and replication factor of 2

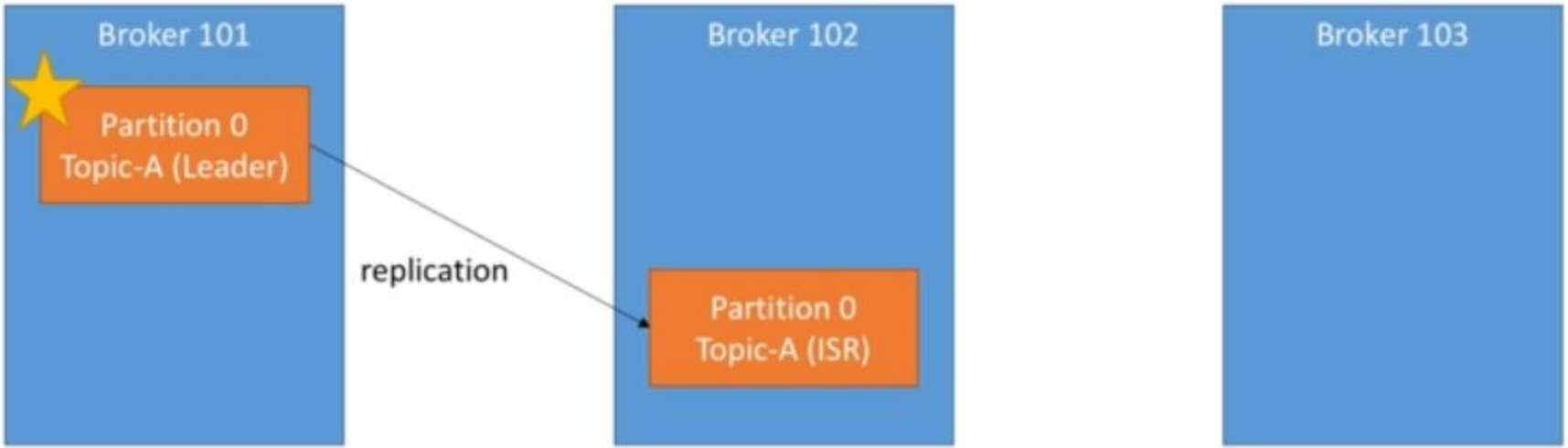



Zookeeper
Team1
team2
broker3
broker3
consumer


centos
VMM (oracle virtual box)
vagrant up

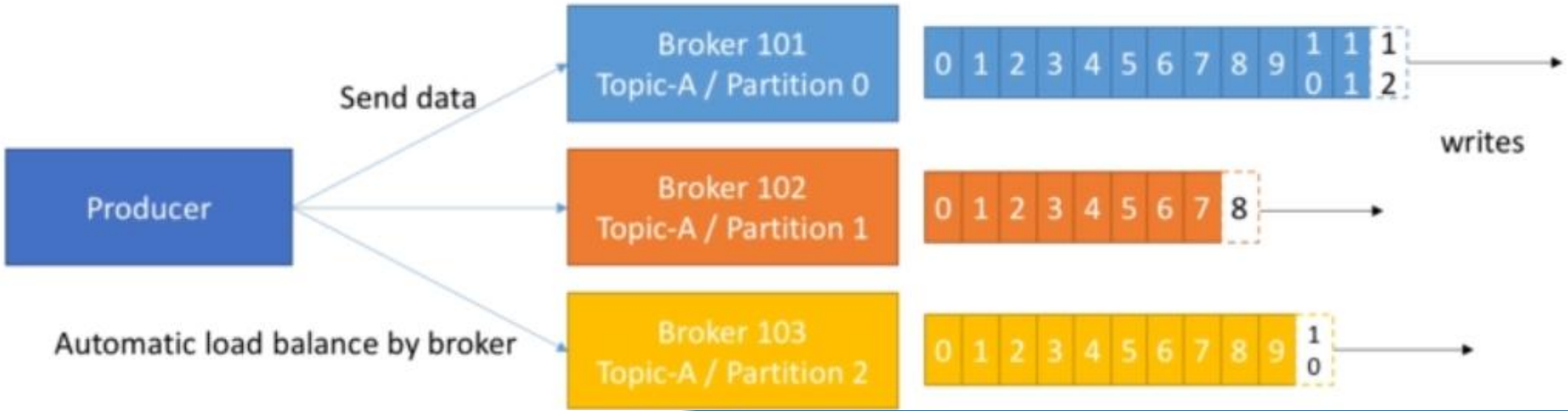
Leader for a partition

- **At any time only ONE broker can be a leader for a given partition**
- **Only that leader can receive and serve data for a partition**
- The other brokers will synchronize the data
- Therefore each partition has one leader and multiple ISR (in-sync replica)



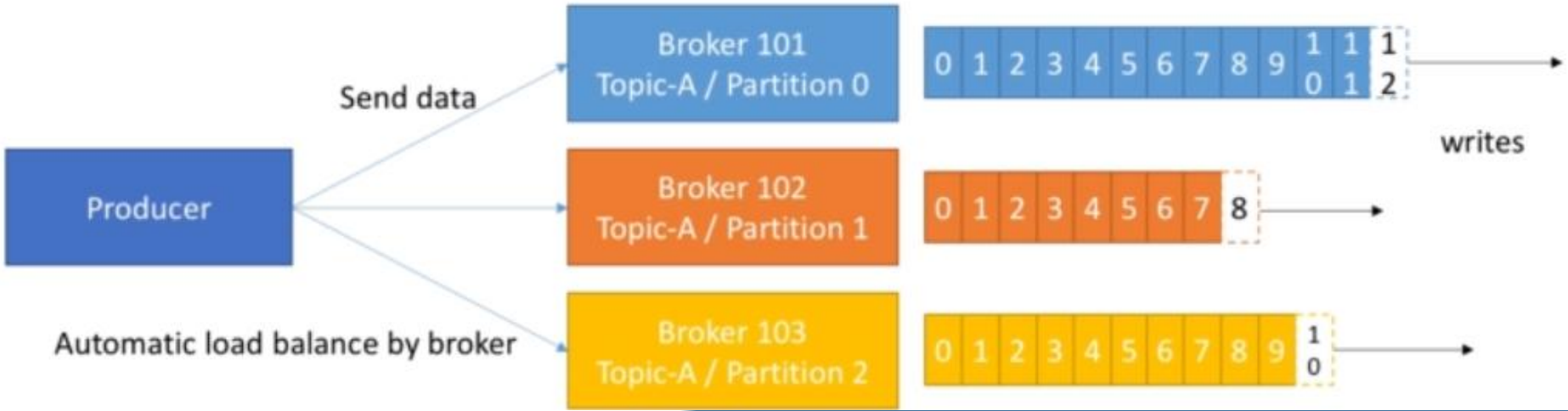
Producers

- **Producers write to and Consumers read from Topic(s)**
- **Topic associated with a log which is data structure on disk**
- **Producer(s) append Records at end of Topic log**
  - Topic **Log consist of Partitions -**
  - Spread to multiple files on multiple nodes

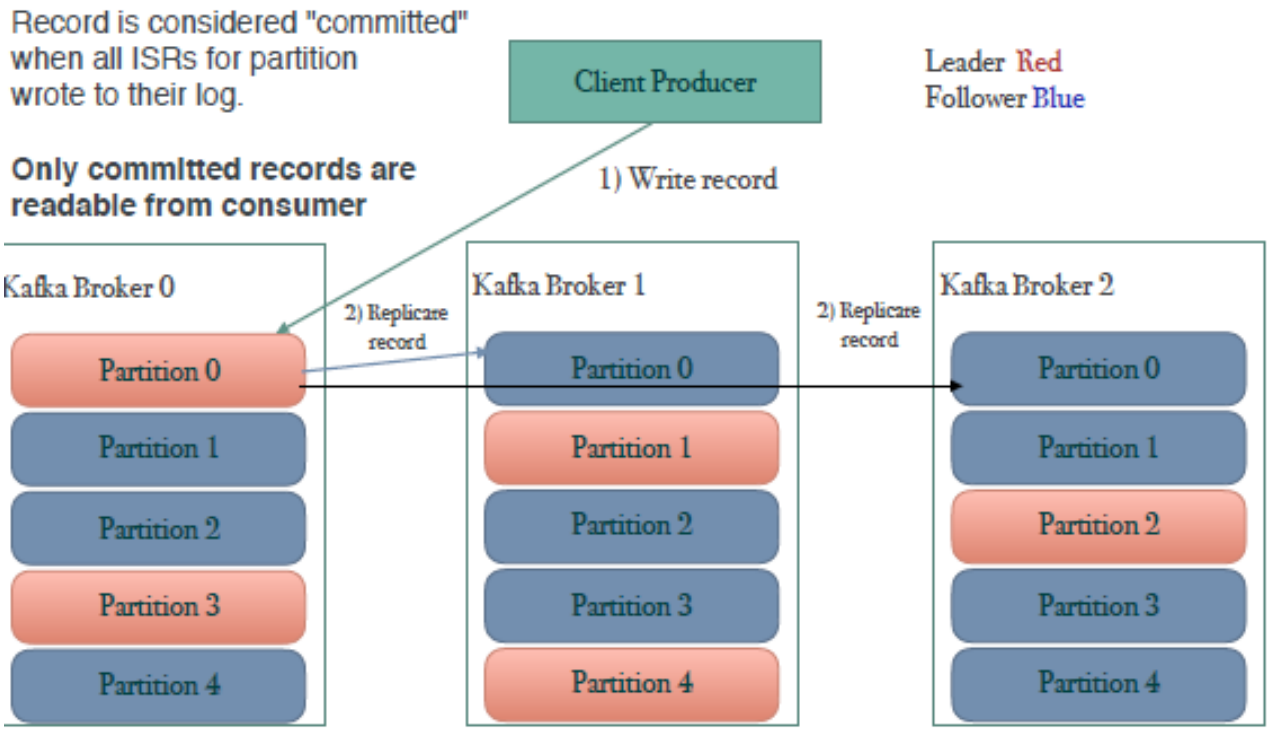


Producers

- Producers can choose to receive acknowledgment of data writes:
  - acks=0: Producer won't wait for acknowledgment (possible data loss)
  - acks=1: Producer will wait for leader acknowledgment (limited data loss)
  - acks=all: Leader + replicas acknowledgment (no data loss)

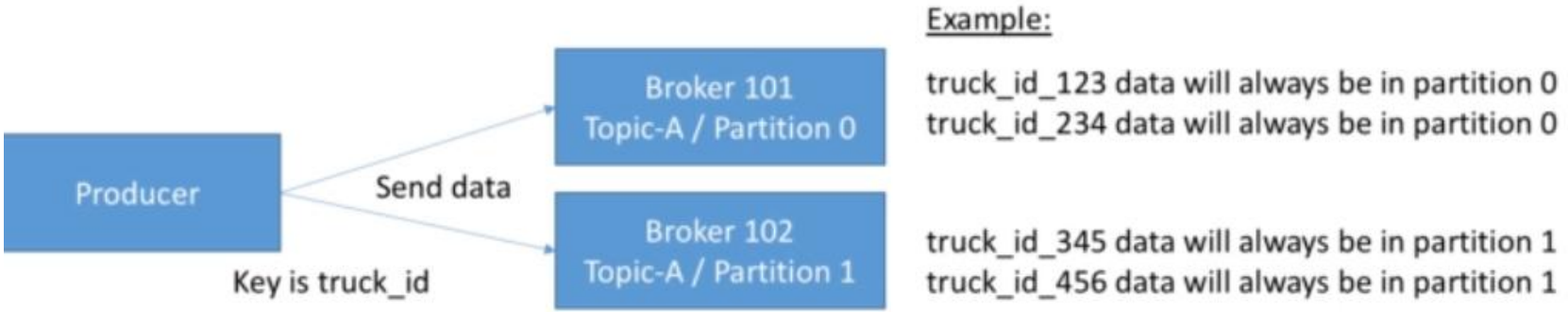


Producers


Record is considered "committed" when all ISRs for partition wrote to their log.
Only committed records are readable from consumer
Client Producer
Leader Red
Follower Blue
1) Write record
Kafka Broker 0
Kafka Broker 1
Kafka Broker 2
2) Replicare record
2) Replicare record
Partition 0
Partition 1
Partition 2
Partition 3
Partition 4
Partition 0
Partition 1
Partition 2
Partition 3
Partition 4
Partition 0
Partition 1
Partition 2
Partition 3
Partition 4

Keys

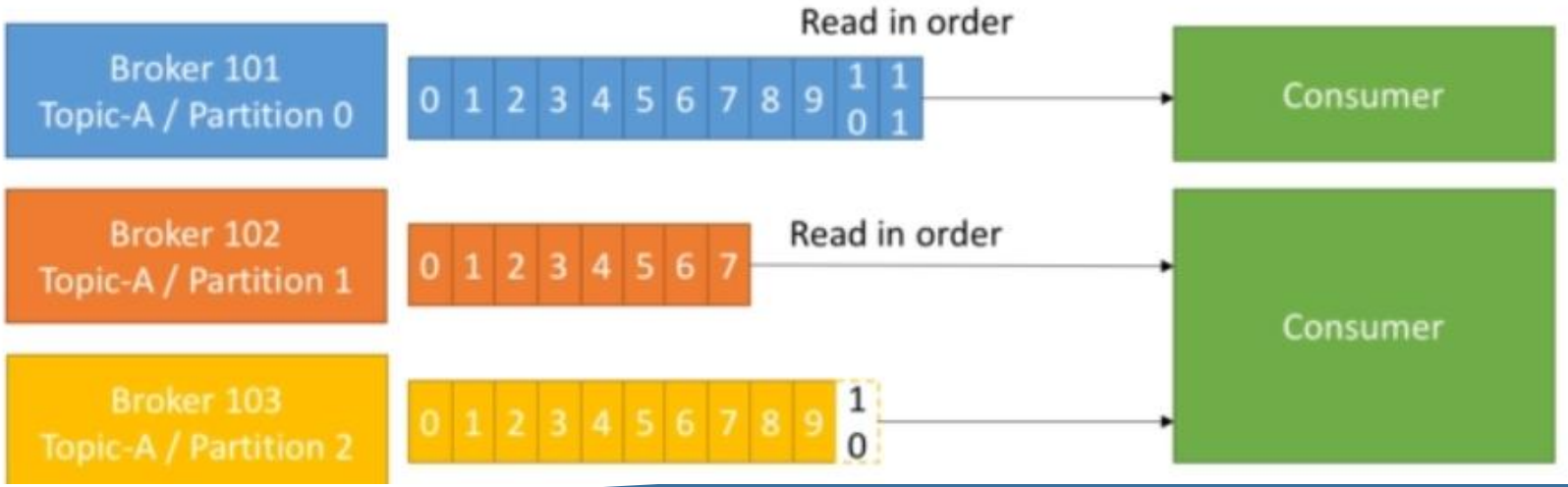
- Producers can choose to send a **key** with the message (string, number, etc..)
- If key=null, data is sent round robin (broker 101 then 102 then 103…)
- If a key is sent, then all messages for that key will always go to the same partition
- A key is basically sent if you need message ordering for a specific field (ex: truck_id)



(Advanced: we get this guarantee thanks to key hashing, which depends on the number of partitions)

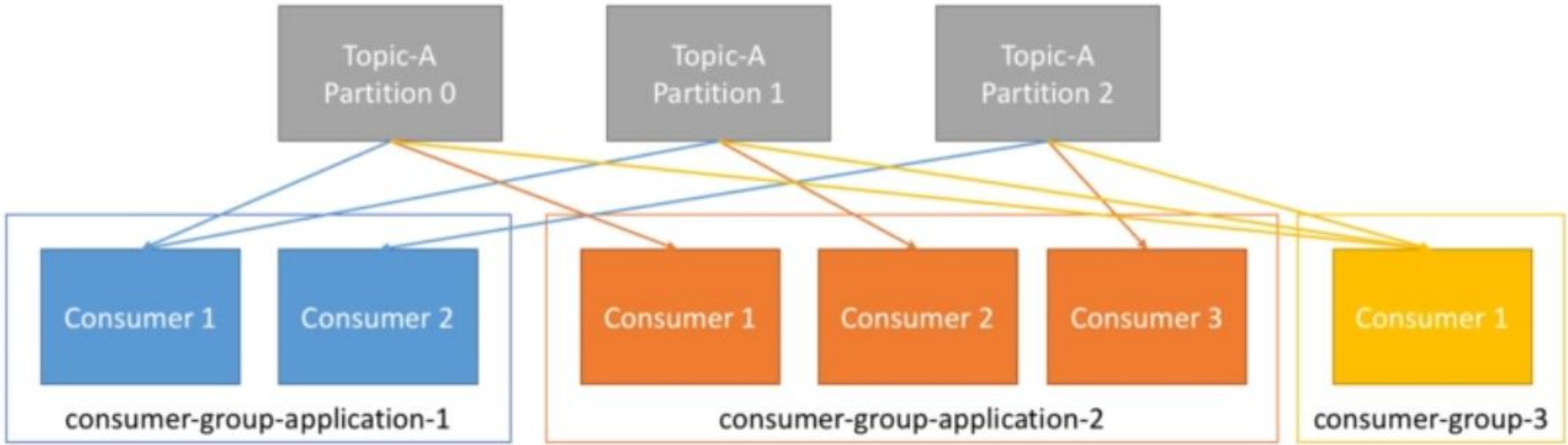
Consumers

- Consumers read data from a topic (identified by name)
- Consumers know which broker to read from
- In case of broker failures, consumers know how to recover
- Data is read in order **within each partitions**



Consumer groups

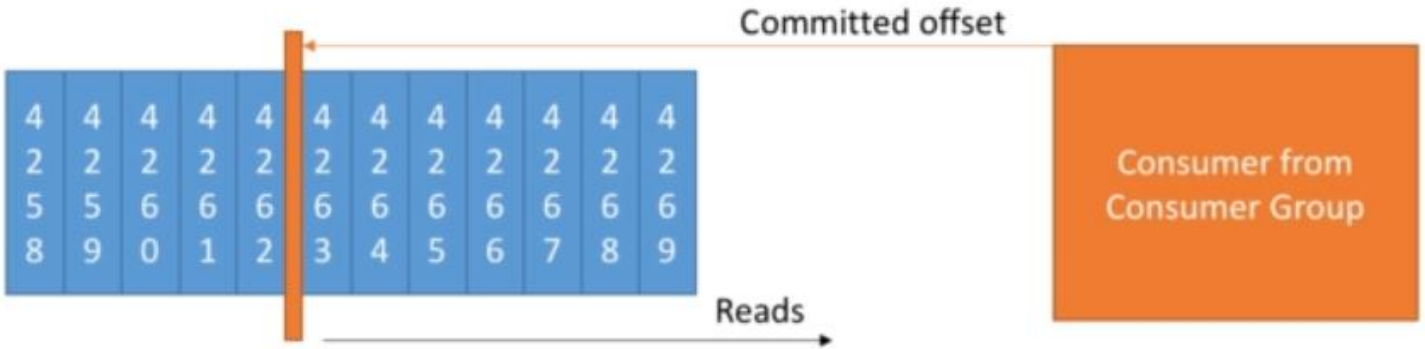
- Consumers read data in consumer groups
- Each consumer within a group reads from exclusive partitions
- If you have more consumers than partitions, some consumers will be inactive



- *Note: Consumers will automatically use a GroupCoordinator and a ConsumerCoordinator to assign a consumers to a partition.*

Consumer offsets

- Kafka stores the offsets at which a consumer group has been reading
- The offsets committed live in a Kafka topic named __consumer_offsets
- When a consumer in a group has processed data received from Kafka, it should be committing the offsets
- If a consumer dies, it will be able to read back from where it left off thanks to the committed consumer offsets!



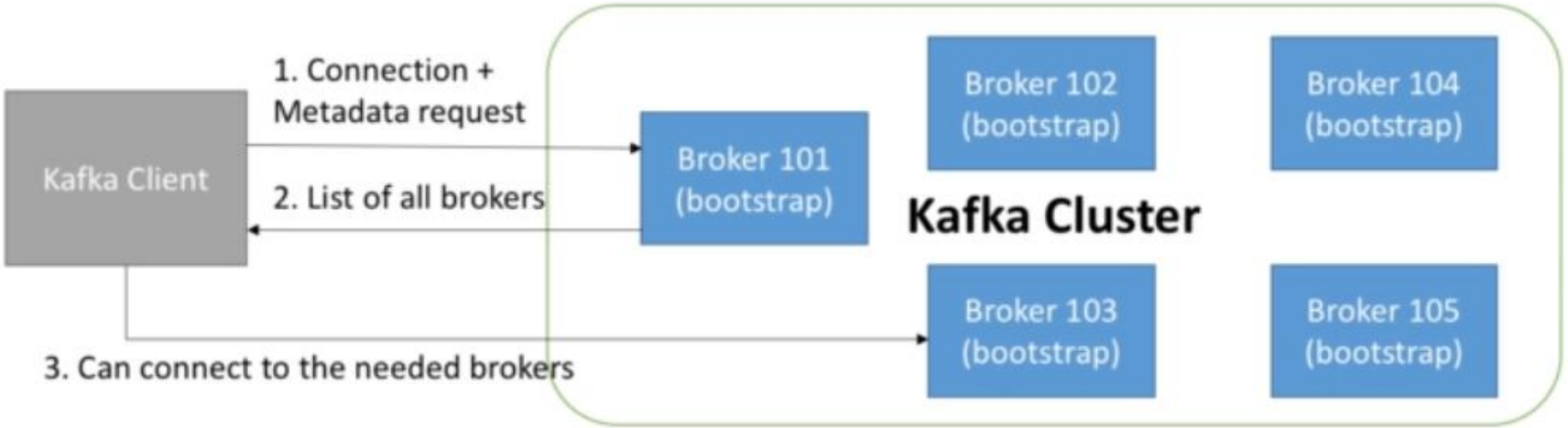
Consumer offsets

- Consumers choose when to commit offsets.
- There are 3 delivery semantics:

- At most once:
  - offsets are committed as soon as the message is received.
  - If the processing goes wrong, the message will be lost (it won't be read again).
- At least once (usually preferred):
  - offsets are committed after the message is processed.
  - If the processing goes wrong, the message will be read again.
  - This can result in duplicate processing of messages. Make sure your processing is idempotent (i.e. processing again the messages won't impact your systems)
- Exactly once:
  - Can be achieved for Kafka => Kafka workflows using Kafka Streams API

Broker Discovery

- Every Kafka broker is also called a "bootstrap server"
- That means that **you only need to connect to one broker**, and you will be connected to the entire cluster.
- Each broker knows about all brokers, topics and partitions (metadata)

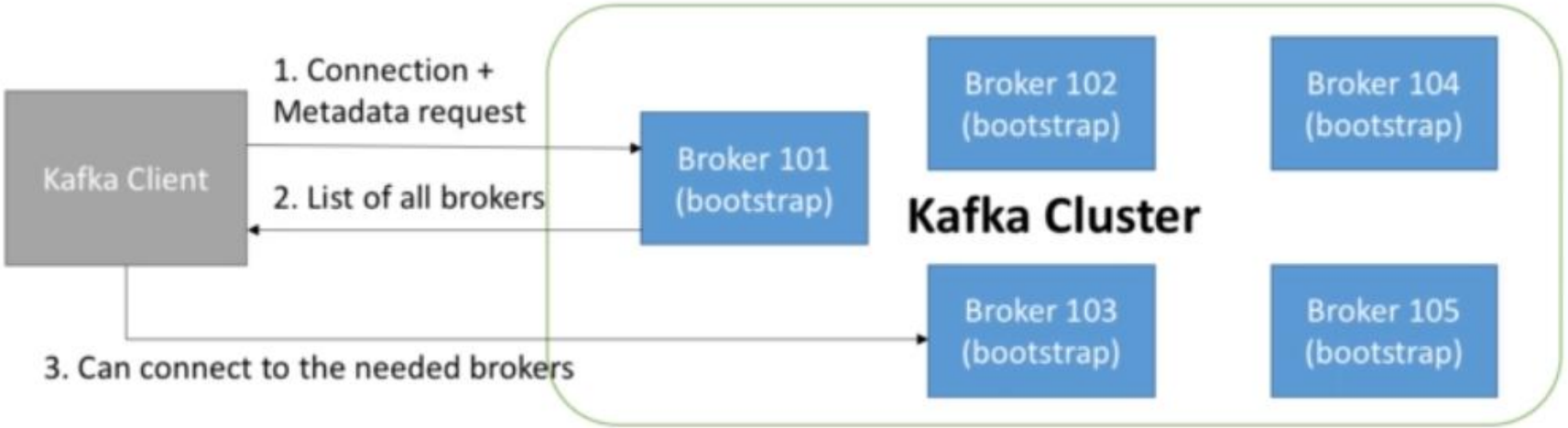


Zoo keeper

- Zookeeper manages brokers (keeps a list of them)
- Zookeeper helps in performing leader election for partitions
- Zookeeper sends notifications to Kafka in case of changes (e.g. new topic, broker dies, broker comes up, delete topics, etc.…)
- **Kafka can't work without Zookeeper**
- Zookeeper by design operates with an odd number of servers (3, 5, 7)
- Zookeeper has a leader (handle writes) the rest of the servers are followers (handle reads)
- (Zookeeper does NOT store consumer offsets with Kafka > v0.10)

Broker Discovery

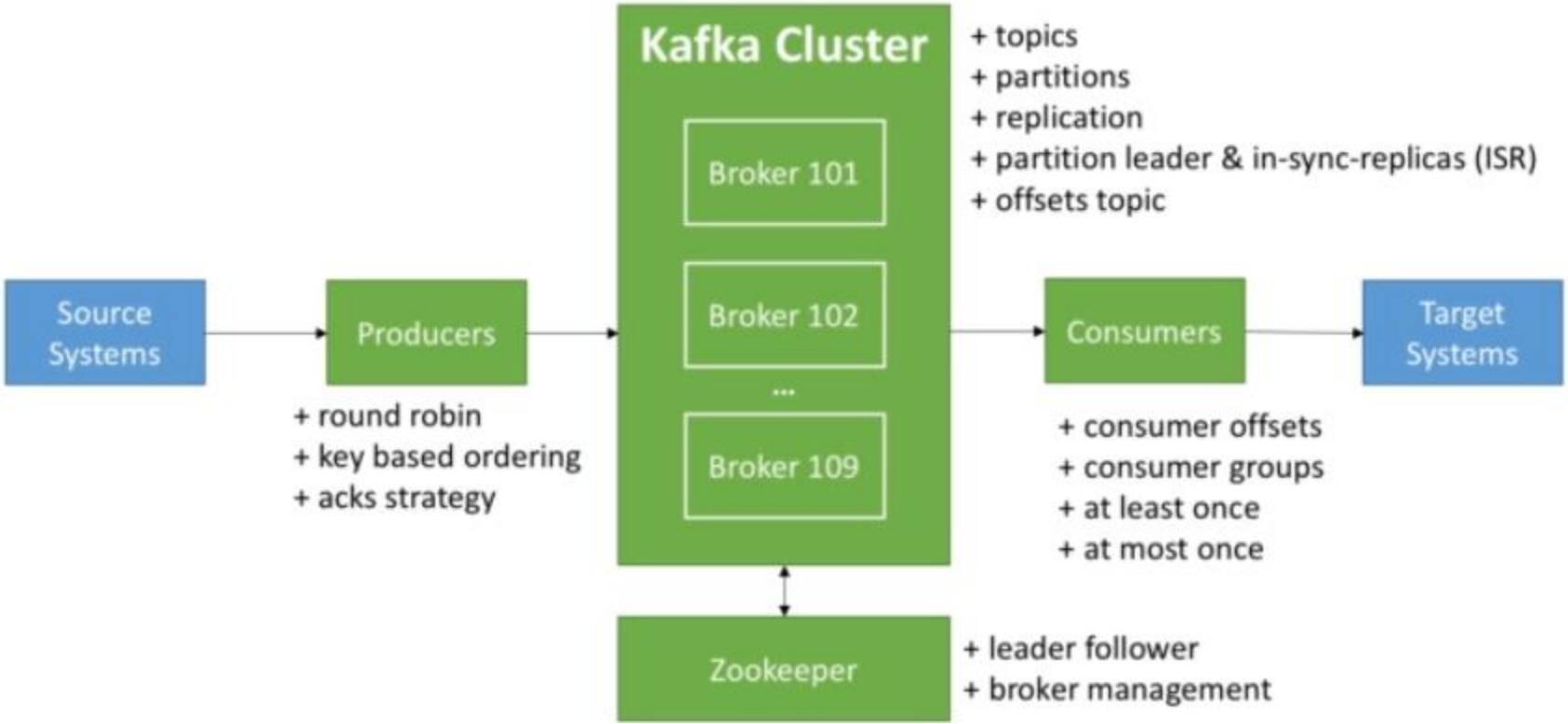
- Every Kafka broker is also called a "bootstrap server"
- That means that **you only need to connect to one broker**, and you will be connected to the entire cluster.
- Each broker knows about all brokers, topics and partitions (metadata)



Kafka guarantees

- Messages are appended to a topic-partition in the order they are sent
- Consumers read messages in the order stored in a topic-partition
- With a replication factor of N, producers and consumers can tolerate up to N-1 brokers being down
- This is why a replication factor of 3 is a good idea:
  - Allows for one broker to be taken down for maintenance
  - Allows for another broker to be taken down unexpectedly
- As long as the number of partitions remains constant for a topic (no new partitions), the same key will always go to the same partition

Overview


Kafka Cluster
+ topics
+ partitions
+ replication
+ partition leader & in-sync-replicas (ISR)
+ offsets topic
Broker 101
Broker 102
...
Broker 109
Source Systems
Producers
+ round robin
+ key based ordering
+ acks strategy
Consumers
+ consumer offsets
+ consumer groups
+ at least once
+ at most once
Target Systems
Zookeeper
+ leader follower
+ broker management