Kafka Programming

Compiled by Amit S Khedkar

Training kickoff

Course Structure

Pre-requisites

- Ability to use the command line
- Some knowledge of Java, or programming in general (We are going to use Java 8)
- Linux and Mac are strongly preferred
- Willingness to learn an awesome technology 

Who is this course for?

- You could be a **developer** and you want to learn how to write an application using Kafka
- You may be an **architect** who understand the role of Kafka in the Enterprise pipeline, some different architectures
- Or finally, you may be a **devops** who wants to understand how Kafka works with regards to topics, partitions, and multi broker setup

Teaching Philosophy

- Clear coverage of concepts & fundamentals
- API: Enough to clearly understand how it works
  - But not so much that it obscures everything else — it's not a reference manual
- Highly interactive (questions, discussions, etc. are welcome)
- Hands-on (learn by doing)
- Current to recent releases of Kafka
  - Which is evolving rapidly

About You And Me

- About you
  - Your Name
  - Your background (developer, admin, manager …etc)
  - Technologies you are familiar with
  - Familiarity with Kafka (scale of 1 – 4 ; 1 – new, 4 – expert)
  - **Something non-technical about you!**
    (favorite ice cream flavor / hobby…etc)

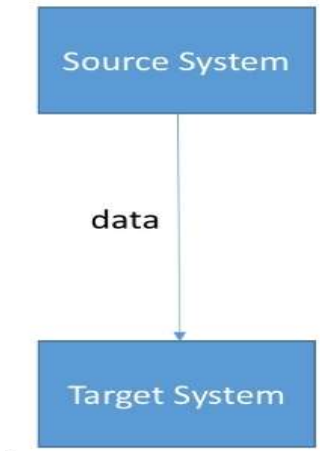
Topics Covered

- Kafka Introduction
- Kafka Fundamentals
- Linux Download and Setup Kafka (Self implemented Hands-on[1])
- Kafka CLI (Discussion – Hands-on Demos[2])
- Kafka Project Creation (IntelliJ) (Discussion – Hands-on)
- Kafka Java Producer and Consumer (Discussion – Hands-on Demos[3])
- Real world – Twitter Producer
- Real World – ElasticSearch Consumer

Kafka Introduction

How Companies Starts

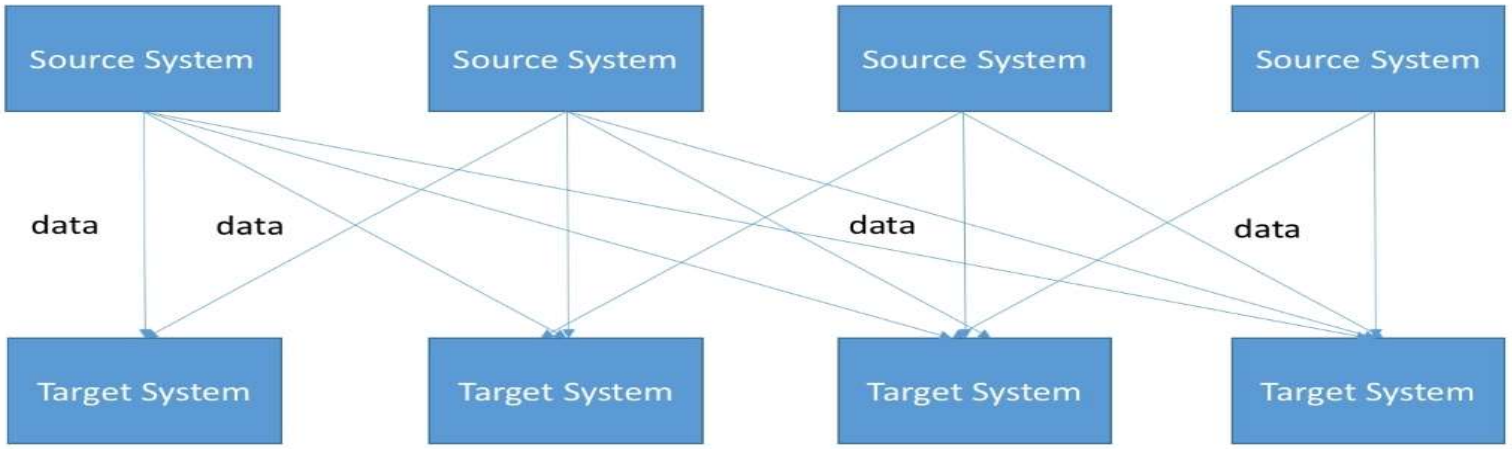
- So before we learn about Kafka, let's learn how companies starts
  - At first it's super simple. You get a source system, and you have a target system
  - And then you need to exchange data



Simple at first!

After a while...

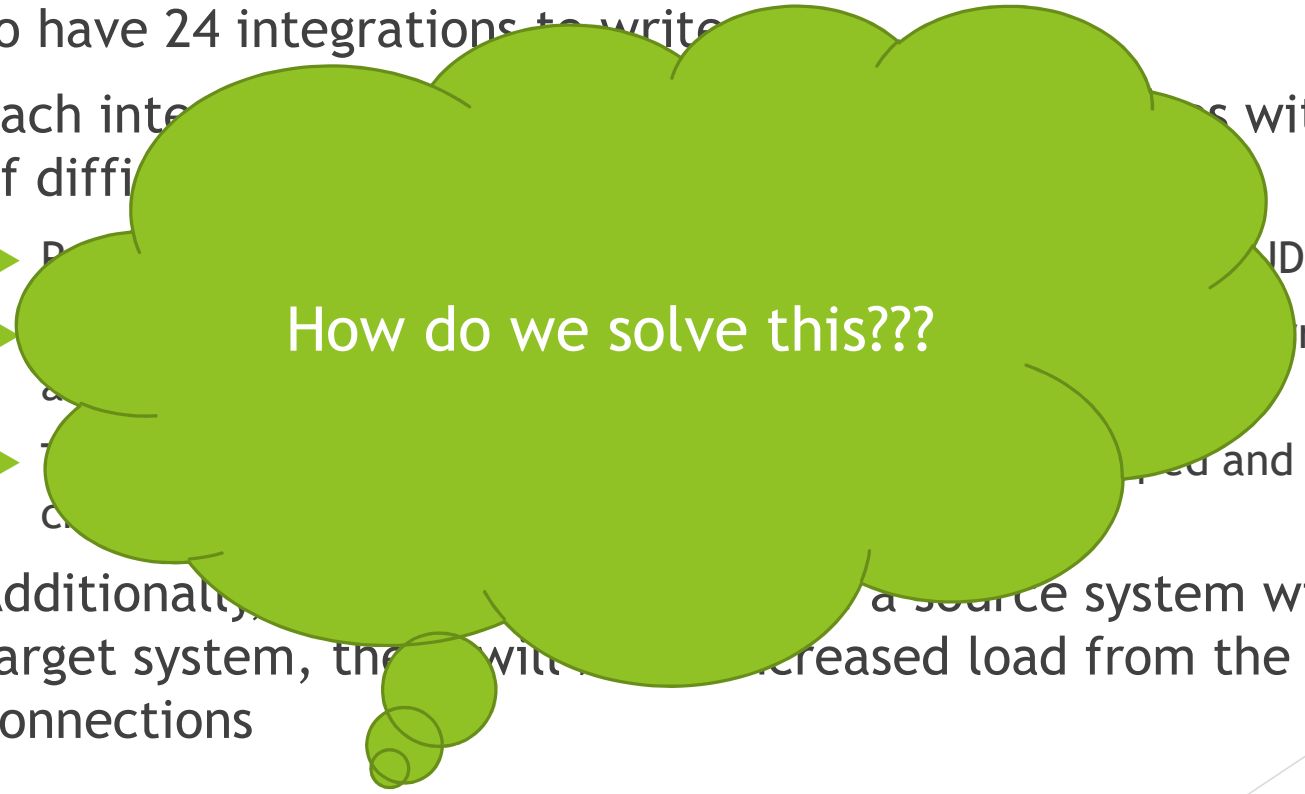
- What happens is...
  - You have many source systems, and many target systems
  - They all have to exchange data with one another, and things become really complicated



Very complicated!

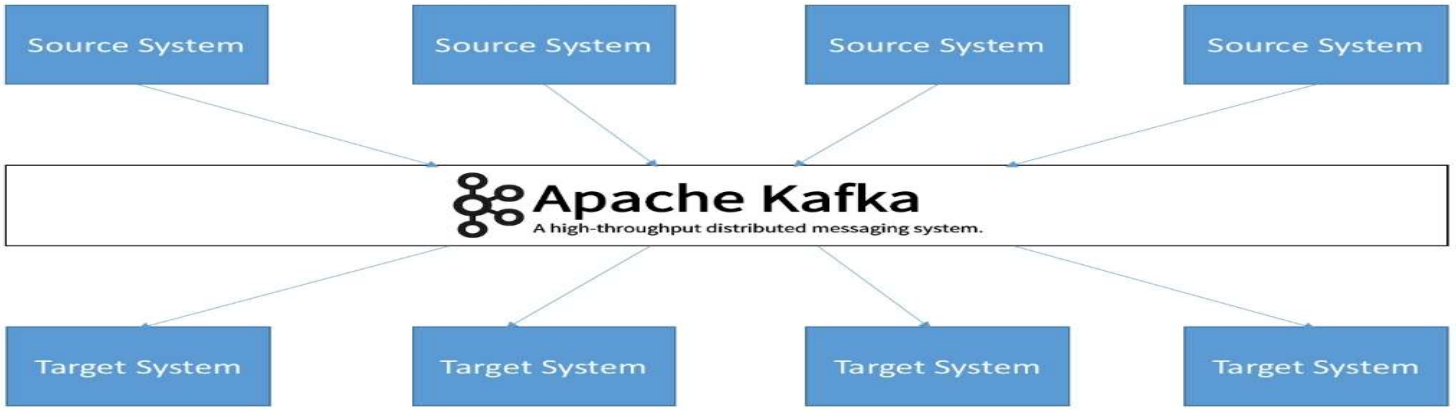
Problems Organizations are facing with previous architecture

- If we have four source systems and six target systems; You need to have 24 integrations to write
- Each inte s with a lot of diffi
  - P IDBC....)
  - ro, Thrift a
  - d and how it may c
- Additionall a source system with the target system, the will ncreased load from the connections



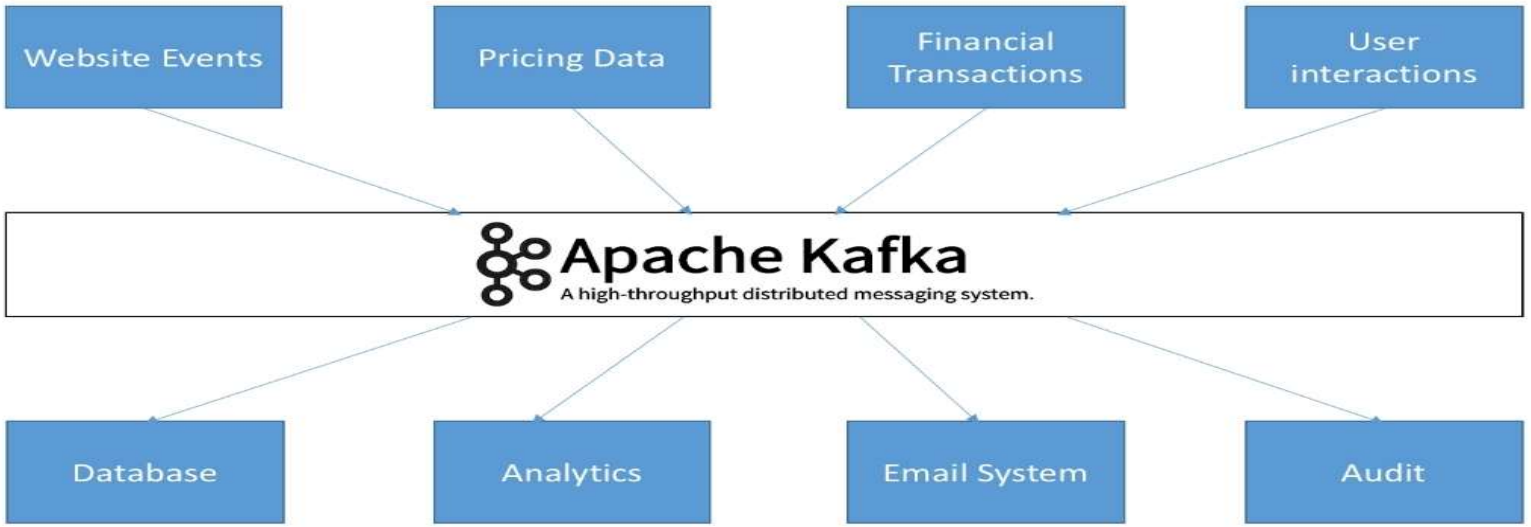
Why Apache Kafka: Decoupling of data streams and systems

- Well this is where Apache Kafka comes in
- Apache Kafka, allows you to decouple your data streams and your systems
  - So now your source systems will have their data end up in Apache Kafka
  - While your target systems will source their data straight from Apache Kafka



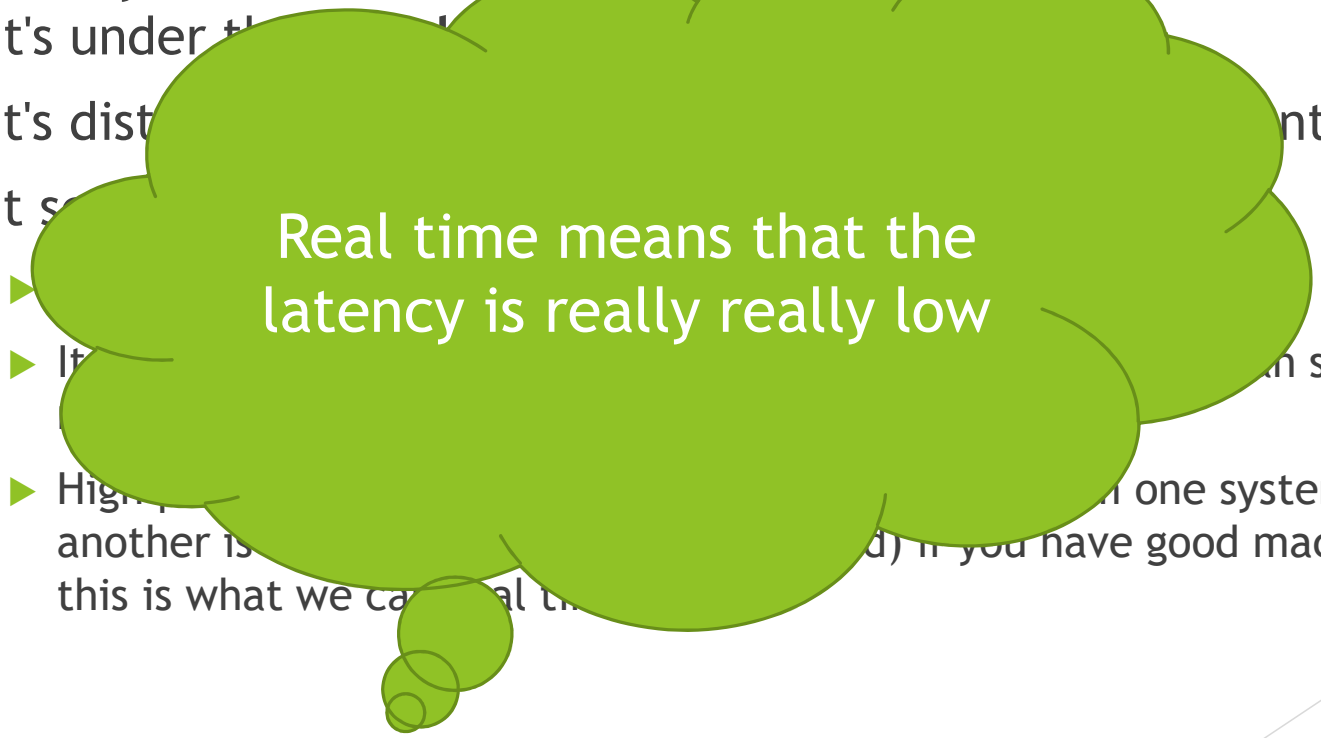
Why Apache Kafka: Decoupling of data streams and systems

- So for example, what do we have in Kafka?
  - Well you can have any data stream you can think about
  - Additionally, once the data is in Kafka, you may want to put it into any system you like



Why Apache Kafka

- It was created by **LinkedIn**, and it's now an open source project, mainly maintained by a private [illegible] **Confluent.** But it's under [illegible]
- It's dist[illegible]nt
- It s[illegible]
  - [illegible]
  - It[illegible]n scale to
  - High[illegible] one system to another is [illegible]d) if you have good machines. And this is what we ca[illegible]al ti[illegible]



Why Apache Kafka

Apache Kafka: Use cases

- Messaging system
- Activity tracking by gather metrics from many different locations or your IoT devices
- Gather logs from your applications
- Stream processing (Using Kafka streams API, or with Spark as an example)
- De-coupling of system dependencies
- Integration with Spark, Flink, Storm, Hadoop, and other big data technologies

Kafka examples...

- So considering a wide array of use cases, many companies are using Apache Kafka as their backbone in their systems
  - Netflix is using Kafka to apply recommendations in real time while you're watching TV shows (And this is why, basically, when you leave a TV show, you'll get a new recommendation right away)
  - Uber uses Kafka to gather user, taxi, and trip data in real time to compute and forecast demand, and computes the surge pricing in real time
  - LinkedIn uses Kafka to prevent spam, and their platform, collect user interactions and make better connection recommendations all of that in real time

Kafka examples...

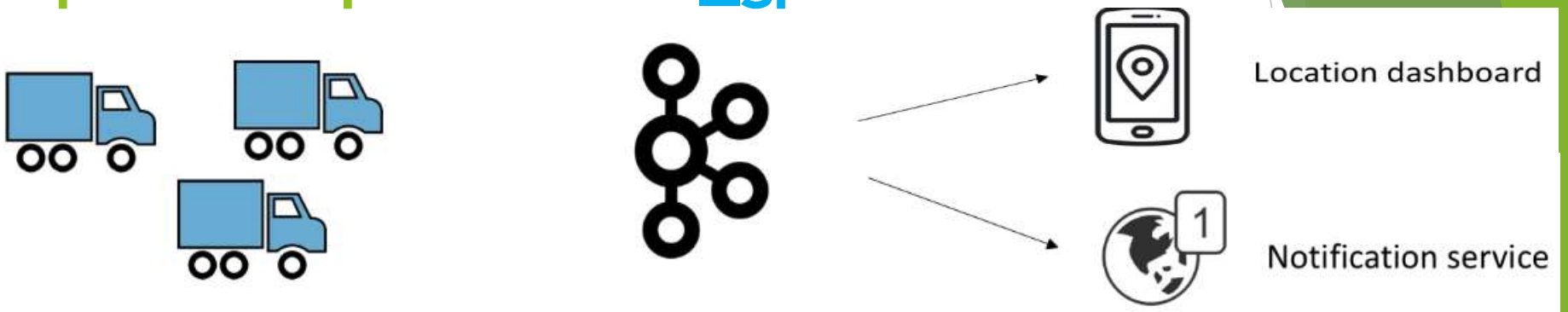
- Basically as you can see, all these companies are using Kafka so that they can make
  - real time recommendation
  - real time decisions
  - give real time insights to their users
- Remember that, Kafka is only used as a transportation mechanism
  - People need, will still write their applications or web applications to make things work, but Kafka is really good at making your data move really fast at scale

Kafka Fundamentals

Topics, partitions and offsets

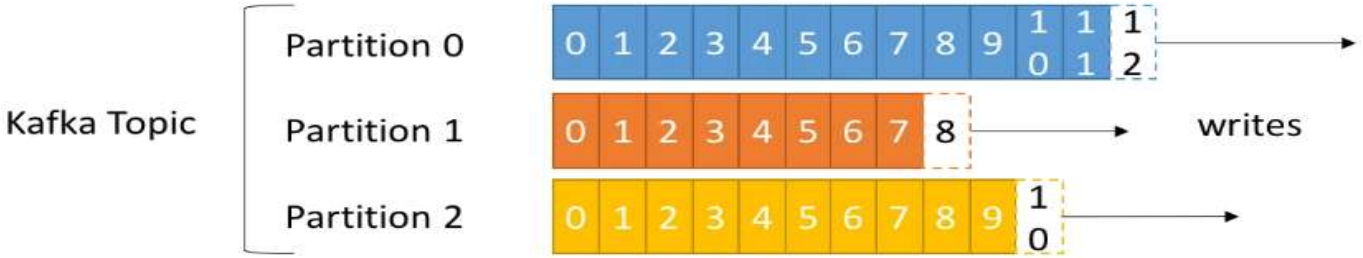
- Topics – A particular stream of data
  - It's basically similar to a table in a database[1]
  - you can have as many topics as you want
  - A topic is going to be identified by its name
- Topics are split into partitions[2]
  - Each partition is ordered
  - Each message within a partition gets an incremental id called as **offset**

Topic example - trucks_gps



- Say you have a fleet of trucks, each truck reports its gps position to kafka
- You can have a topic trucks_gps that contains the position of all trucks
- Each truck will send a message to kafka every 20 seconds, each message will contain the truck id and the truck position (latitude and longitude
- We choose to create that topic with 10 partitions(arbitrary number)

Topics, partitions and offsets



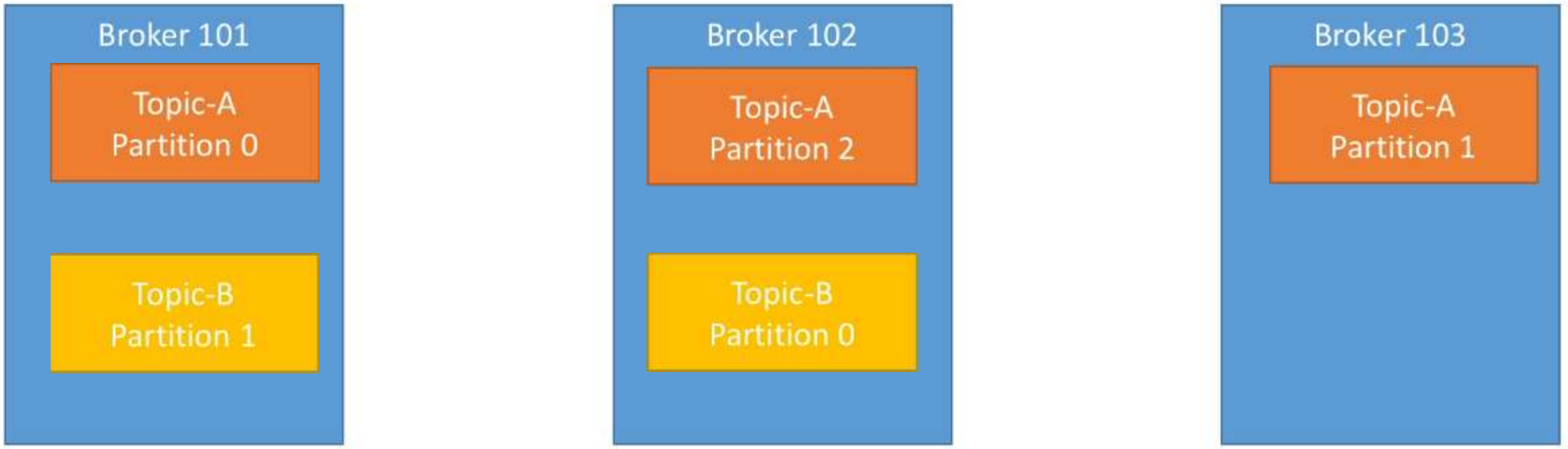
- Offsets only have a meaning for a specific partition
  - E.g. offset 3 in partition 0 doesn't represent the same data as offset 3 in partition 1
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited period (default is one week)
- Once the data is written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

Brokers

- Okay, so we've talked about topics, but what holds the topics? What holds the partitions?
  - The answer is a 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 100 brokers

Brokers and Topics

- We have 3 brokers
- Example of Topic-A with three partitions
- Example of Topic-B with two partitions



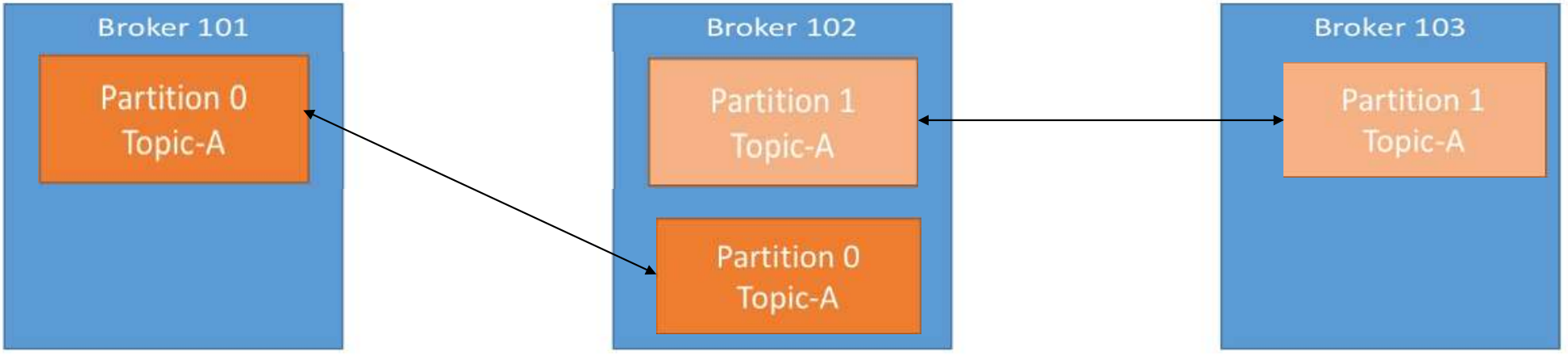
- Note – Data is distributed and Broker 103 does not have any Topic-B data

Topic replication factor

- Kafka is a distributed system[1]
- So, when there's a distributed system in the big data world we need to have replication to achieve fault tolerance[2]

Topic replication factor

- Here is our cluster with three brokers
- Topics should have a replication factor > 1 (ususally between 2 and 3)[1]
- This way if a broker is down, another broker can serve the data
- Example – Topic-A with 2 partitions and replication factor of 2

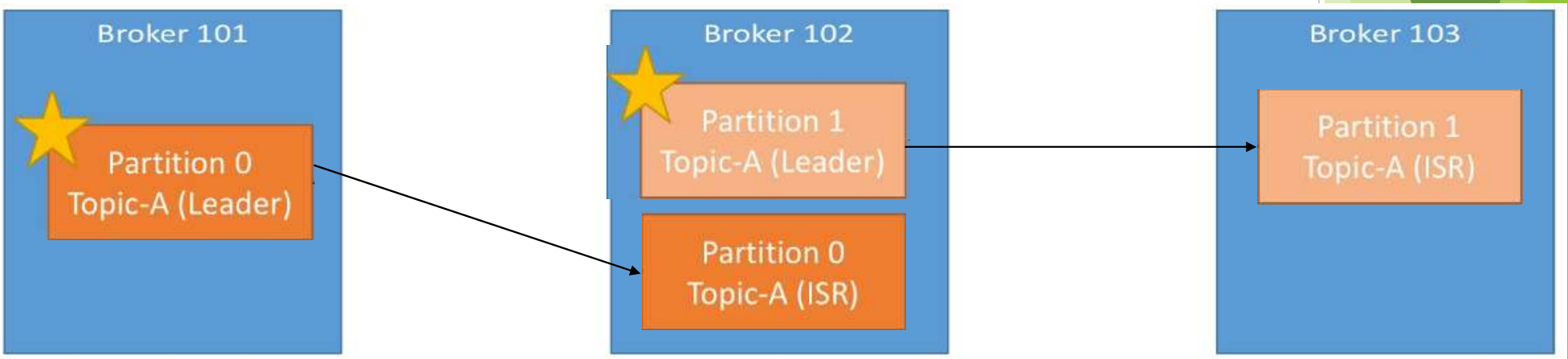


Topic replication factor

- Example: We lost broker 102
- Result: Broker 101 and 103 can still serve the data

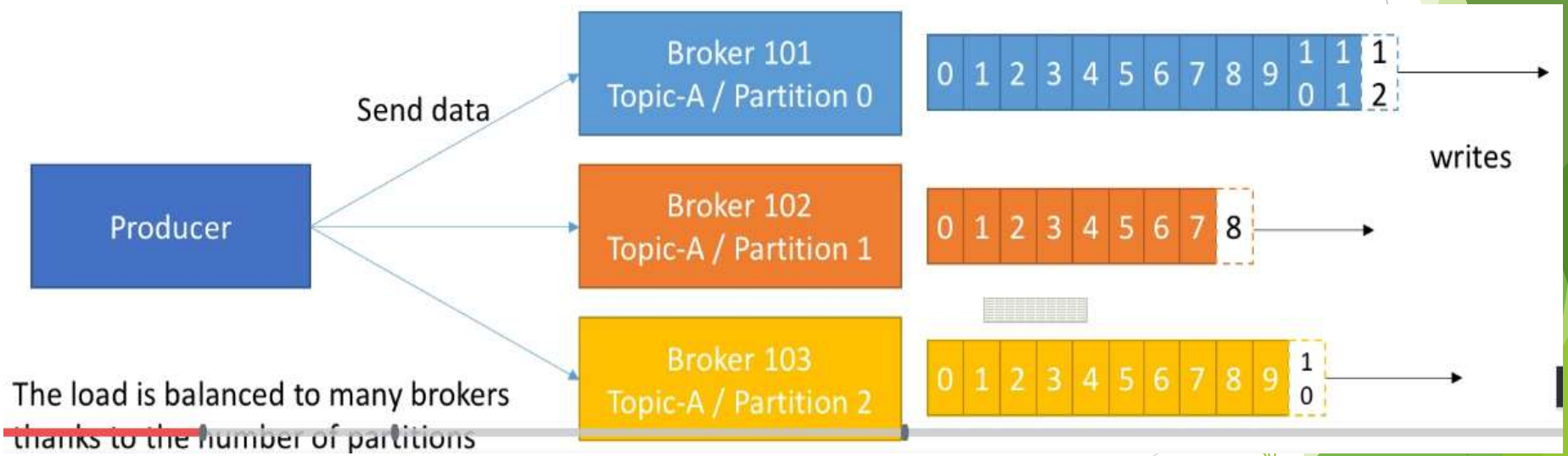
Concept of Leader for a Partition

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



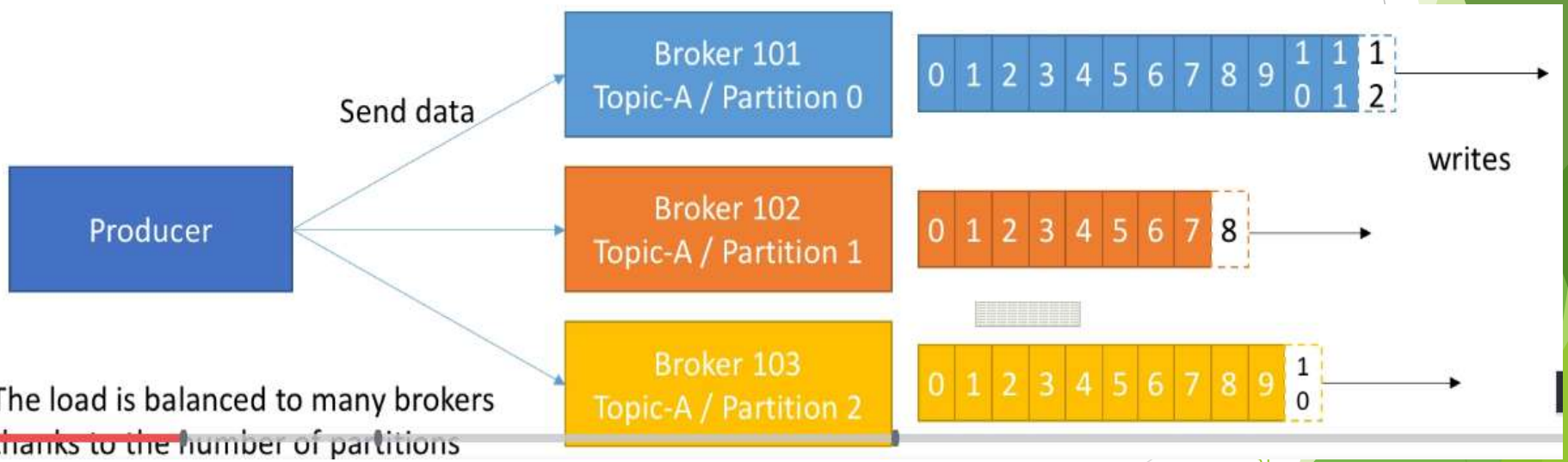
Producers

- Producers write data to topics(which is made up of partitions)
- Producers automatically know to which broker and partition to write to
- In case of broker failure, Producers will automatically recover



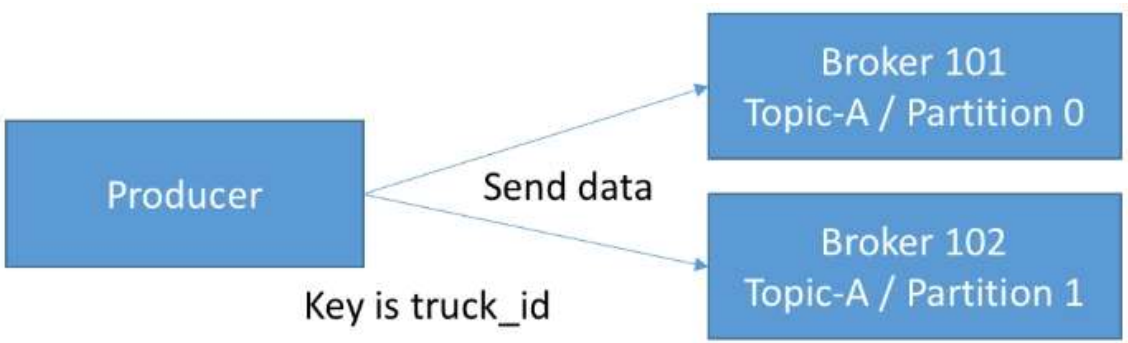
Producers

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



Producers: Message keys

- Producers can choose to send a **key** with message(string, number, etc…)
- If key=null data sent round robin
- 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 (eg – truck_id)



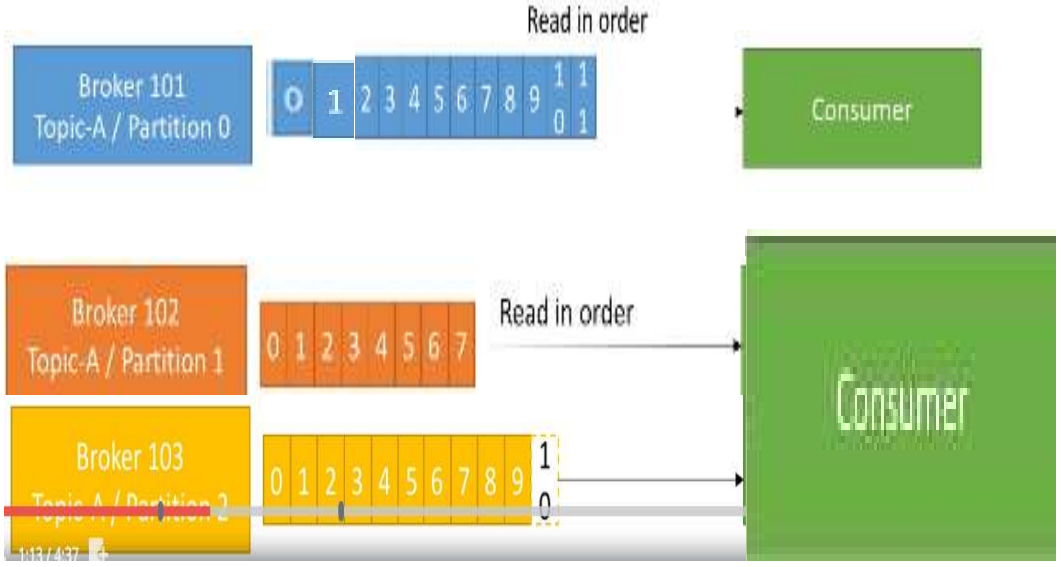
Example:

truck_id_123 data will always be in partition 0
truck_id_234 data will always be in partition 0

truck_id_345 data will always be in partition 1
truck_id_456 data will always be in partition 1

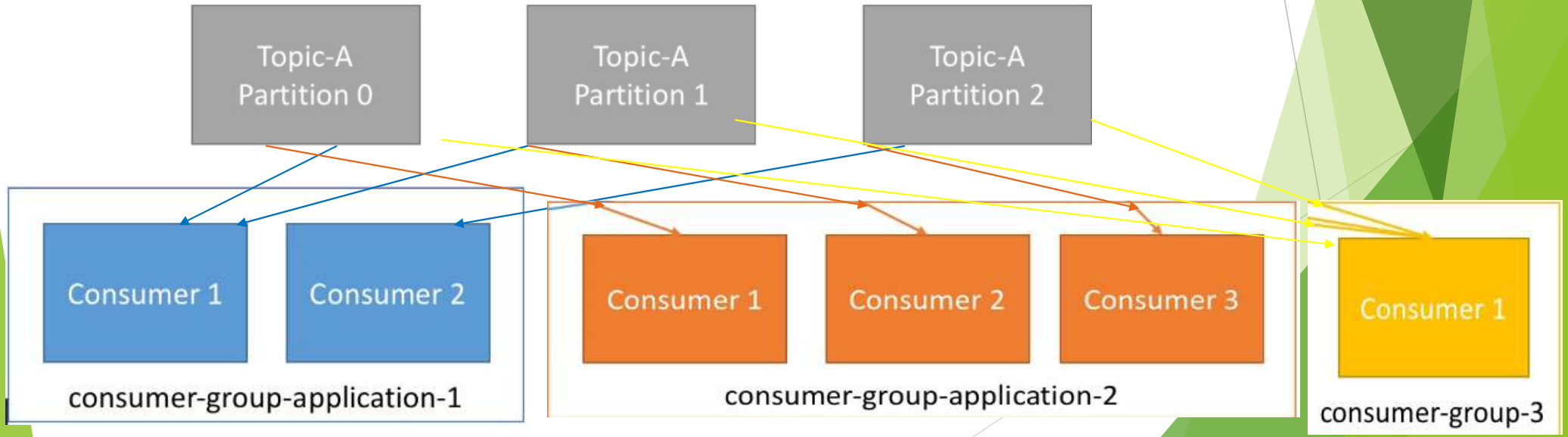
Consumers

- 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 partition



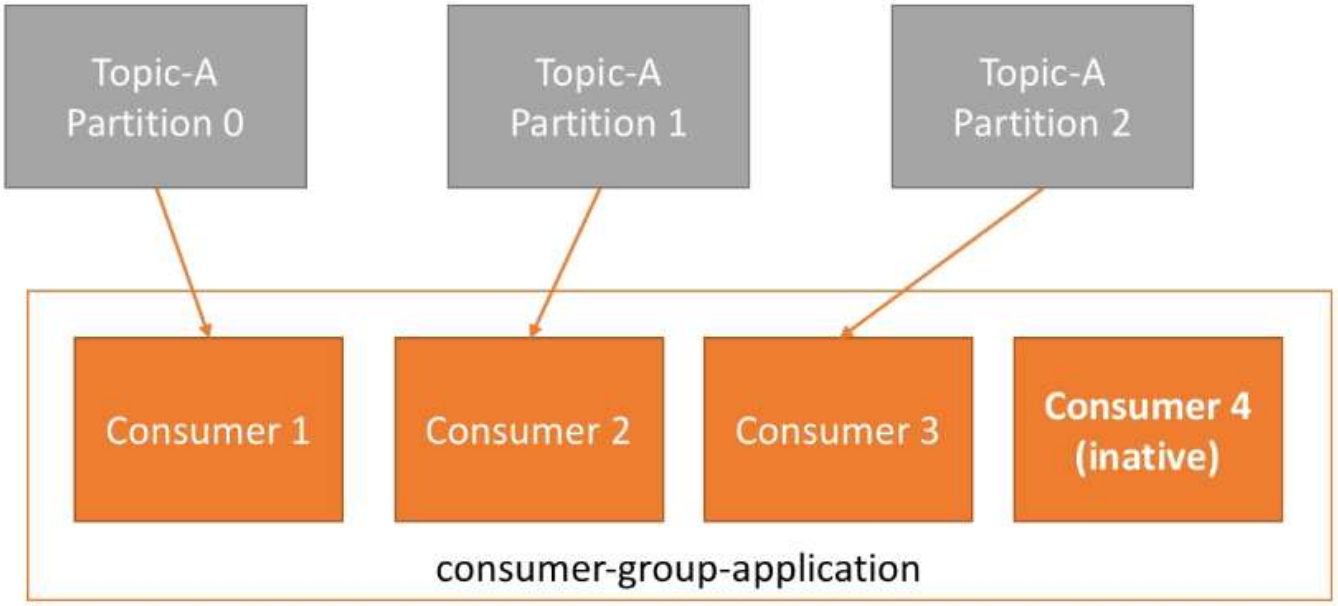
Consumer groups

- Now, how do these consumers read data from all the partitions?
- 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



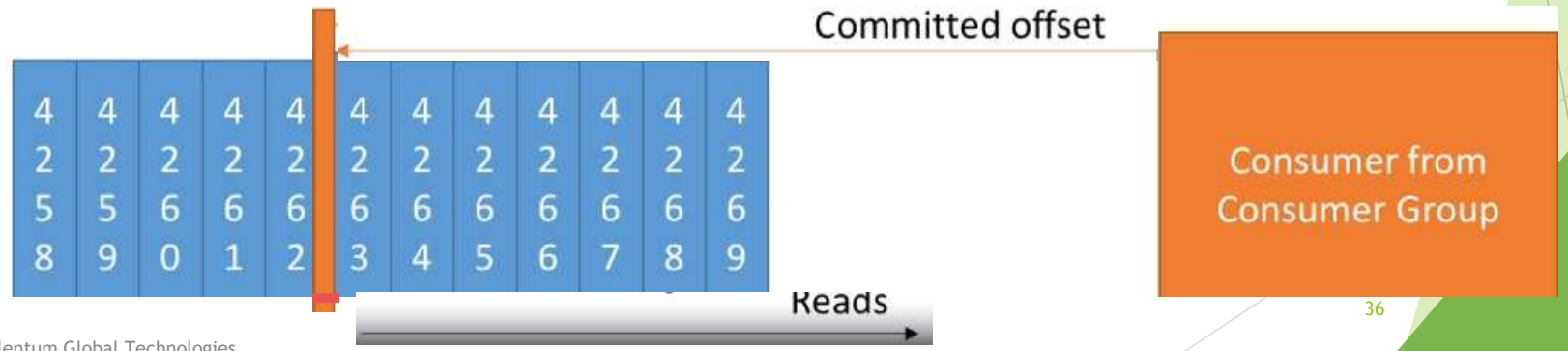
Consumer groups, what if too many consumers?

- If you have more consumers than partitions then some consumers will be inactive



Consumer Offsets

- Kafka stores the offsets at which a consumer group has been reading
- These offsets committed live in a Kafka topic named **__consumer_offsets**
- When a consumer in a gropup 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 consumers offsets!

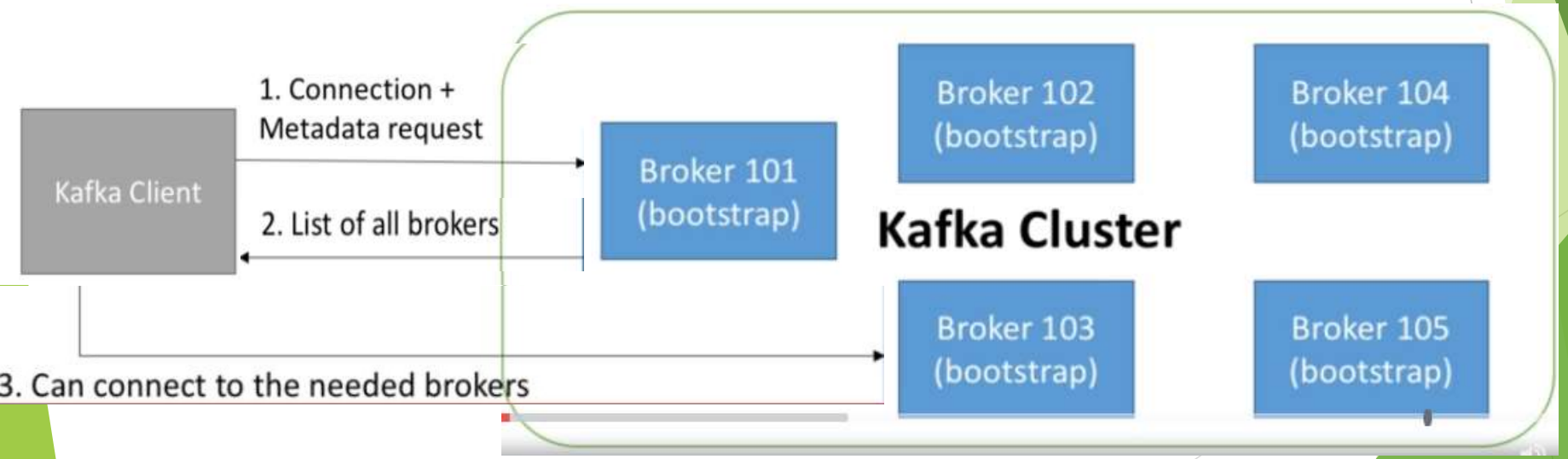


Delivery semantics for consumers

- 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)
- **Exctly once:**
  - Can be achieved for Kafks => Kafka workflows using Kafka Streams API
  - For Kafka => External system work flows, use an idempotent consumer

Kafka broker discovery

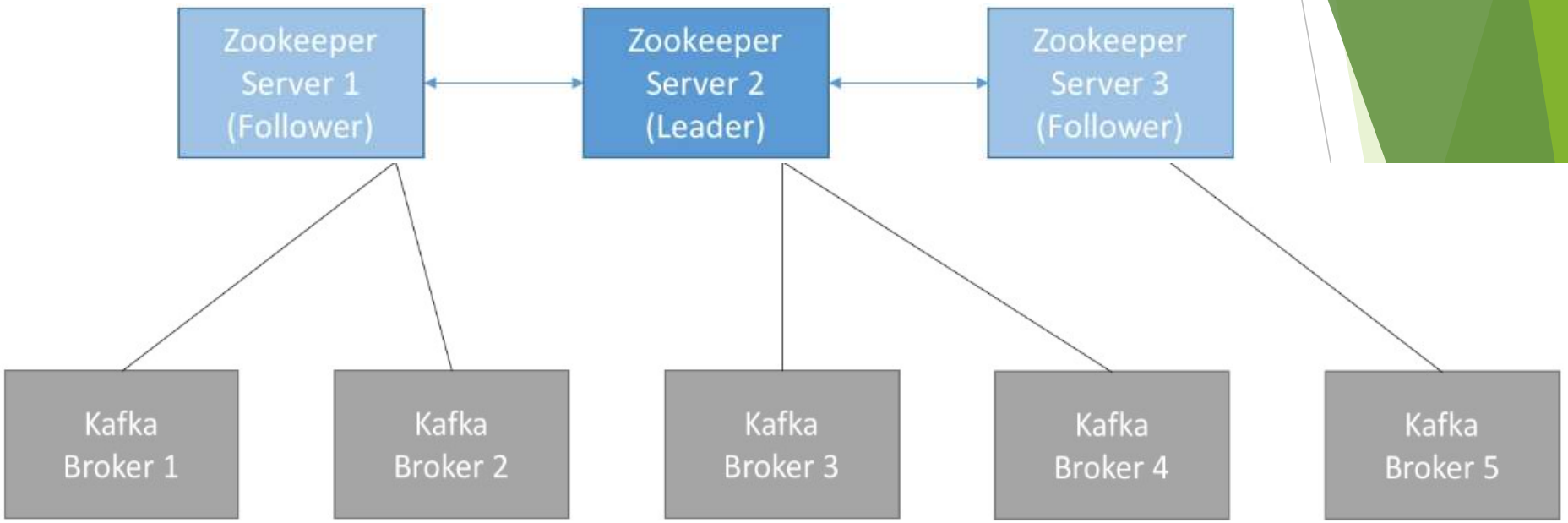
- Every Kafka broker is also called a “bootstrap server”
- That means that **you only need to connect to one broker,** and you will be connected to the entire cluster
- Each broker knows about all brokers, topics, and partitions (metadata)



Zookeeper

- Zookeeper manages brokers (keep a list of them)
- Zookeeper helps in performing leader election for partitions
- Zookeeper sends notification to Kafka in case of changes (e.g new topic, broker dies, broker comes up, delete topics…)
- **Kafka cannot 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

Zookeeper


Zookeeper Server 1 (Follower)
Zookeeper Server 2 (Leader)
Zookeeper Server 3 (Follower)
Kafka Broker 1
Kafka Broker 2
Kafka Broker 3
Kafka Broker 4
Kafka Broker 5

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

Theory roundup