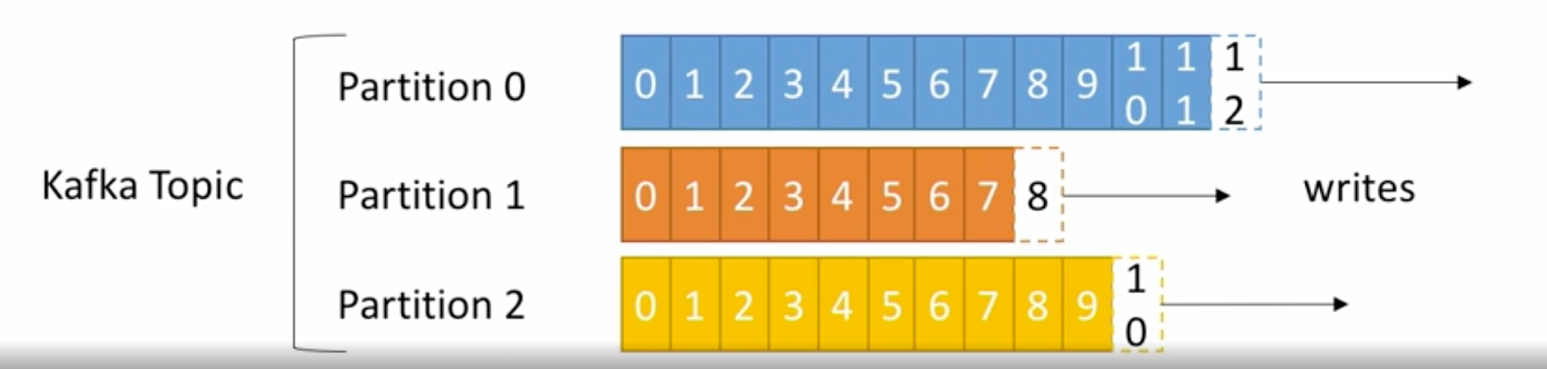
***KAFKA Notes***

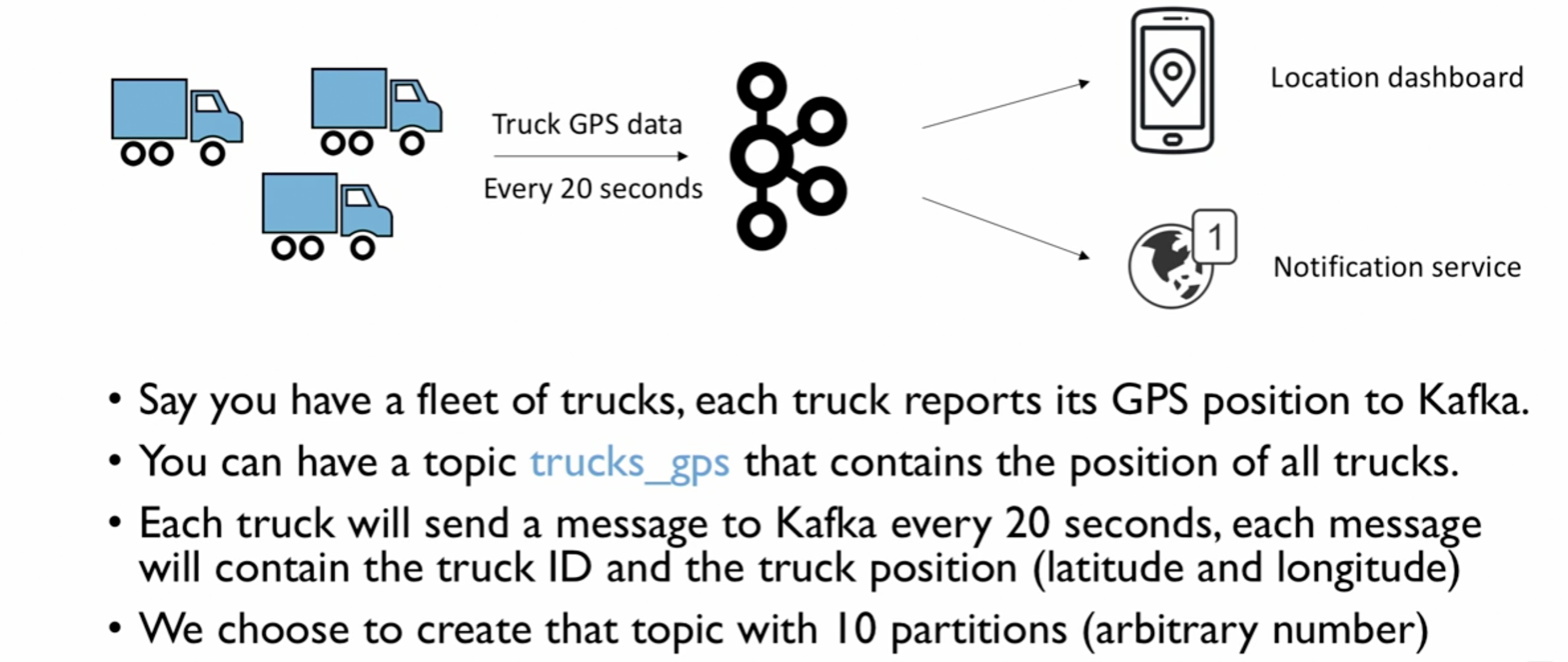
* **Topics**:
  + Topic is a particular stream of data in Kafka.
  + Similar to table in a relational database.
  + In Kafka, we can have as many topics as we want; just like there is no limitation on number of tables in a relational database.
  + A topic is uniquely identified by a name.
* **Partition**:
  + The Kafka topics are split into partitions.
  + Each partition is ordered.
  + Each message within each partition gets an incremental id called **offset**.
  + The number of partitions that you want in a topic is specified while creating a topic. The number of partitions can also change after the topic is created.
  + Each partition can have different number of messages.



**Points to remember:**

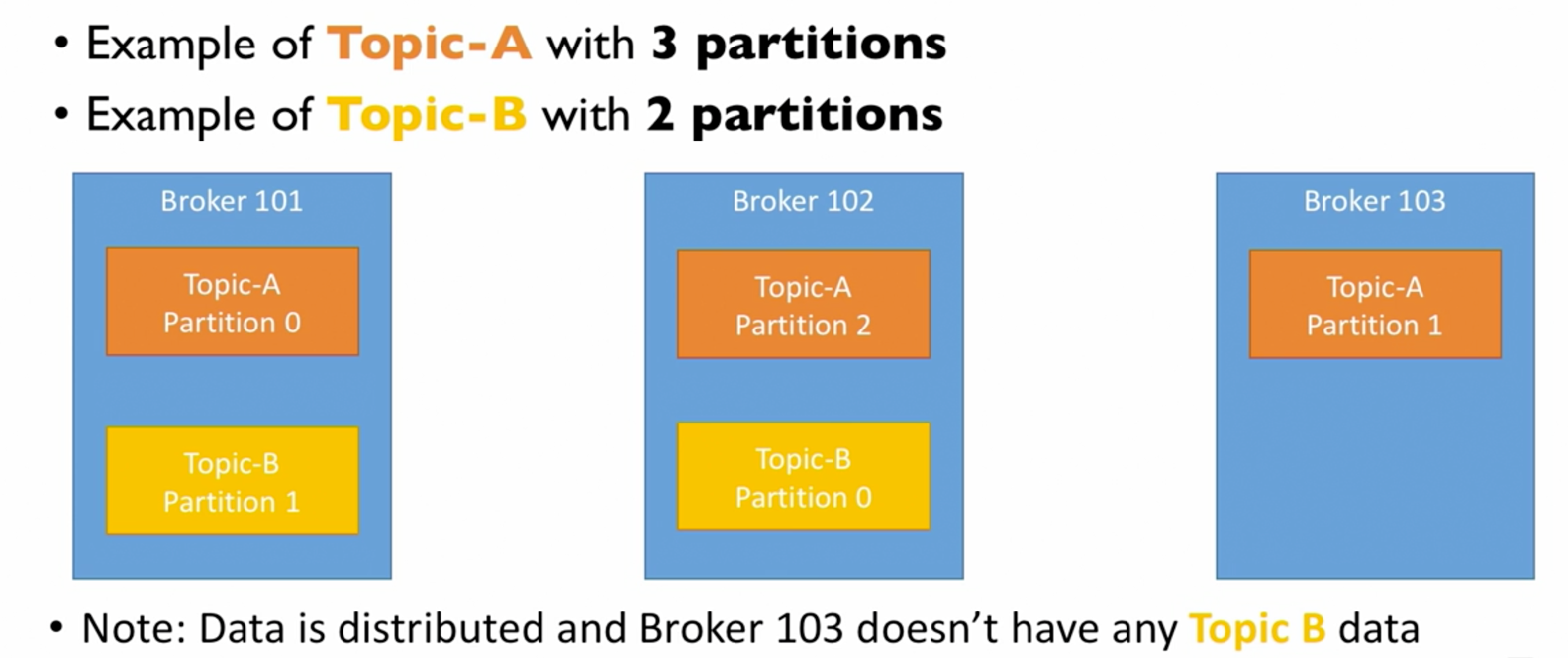
* **Offset** has a meaning only for a specific partition.
* Orderis guaranteed only within a partition; not across partitions.
* Data in kafka is kept only for a limited time; defaults to 1 week.
* Offsets keep incrementing, never decrease; even when the older messages are dropped.
* Once the data is written into a partition, it cannot be changed; immutability.
* Data is assigned randomly to a partition unless a key is provided.

**Example of topic:**



* **Kafka Cluster:**
  + Kafka is usually clustered (distributed system).
  + Kafka cluster is composed of multiple servers called **brokers.**
  + Each broker is identified by an ID (integer type).
  + Each broker contains certain topic partitions; i.e. each broker will have some data, but not all of the data for a topic.
  + After connecting to any broker (called a **bootstrap broker**), you will be connected to an entire cluster.
  + Good number to start with brokers is 3, but in production systems, it may be up to hundreds, depending on the need and complexity of the system.

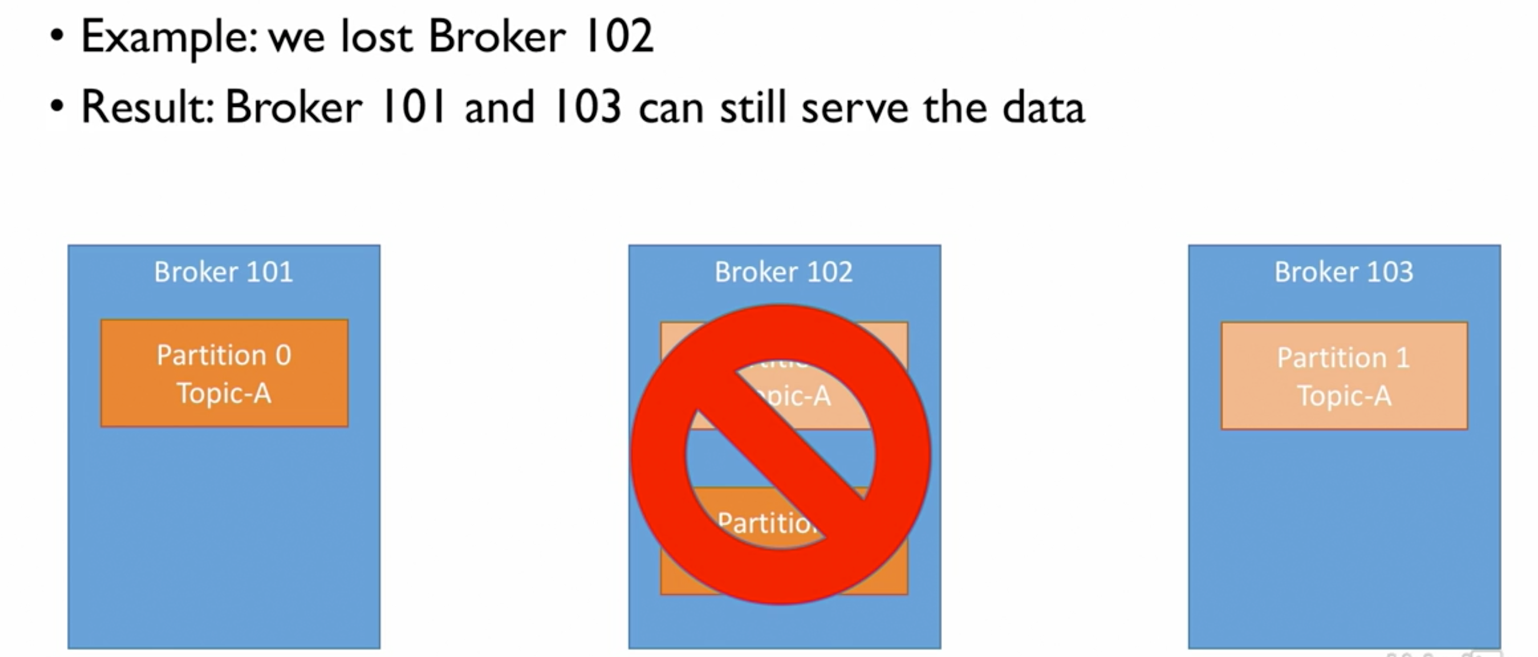
**Broker and Partitions:**

****

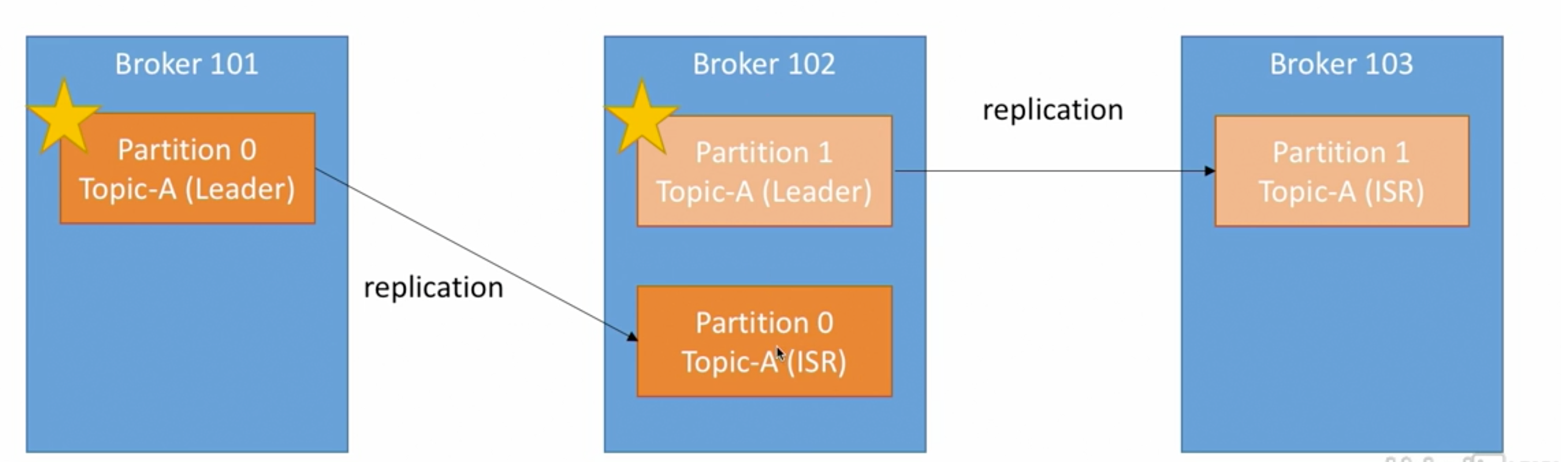
* **Replication Factor:**
  + The integral number by which the partition data is replicated on brokers is called a replication factor.
  + Since kafka is a distributed system, we need to make sure that the data integrity is maintained even if some of the brokers are down. Replication helps to achieve this.
  + The general strategy of replication is that the partition data on a broker is replicated to another broker. So when one of the brokers is down, the other broker can serve the same data.
  + The replication factor should always be > 1 (ideally between 2 & 3)
  + The replication factor has to be decided while creating a topic.
  + Replication factor of N means that there will be a total of N copies of each partition in the topic.

**For example:**

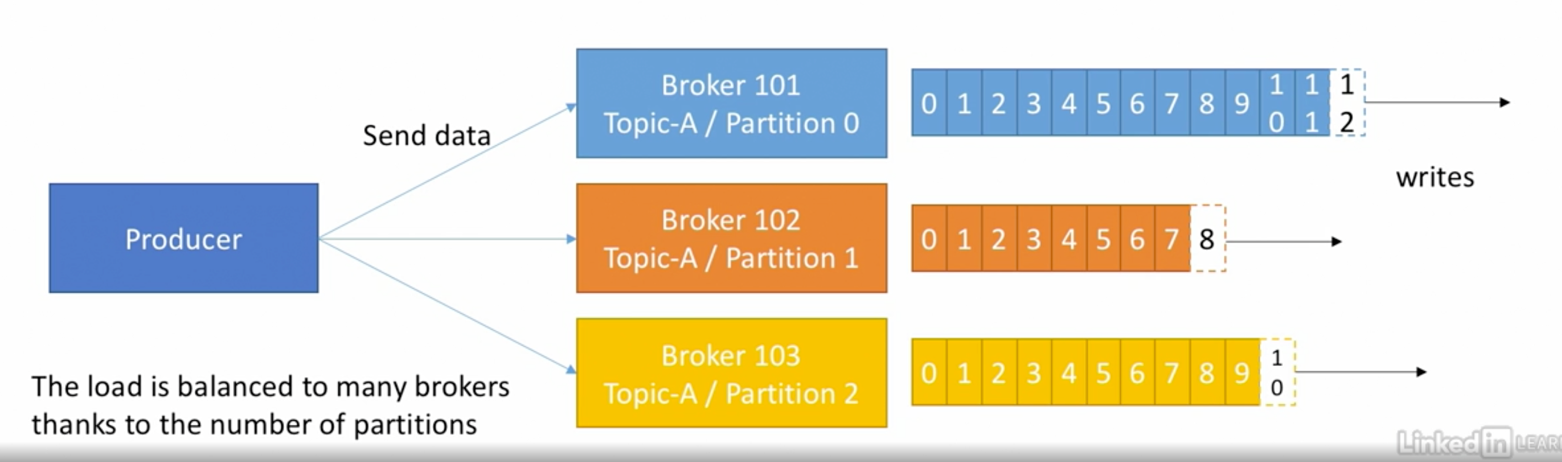
**Topic A has 2 partition and the replication factor is also 2**

****

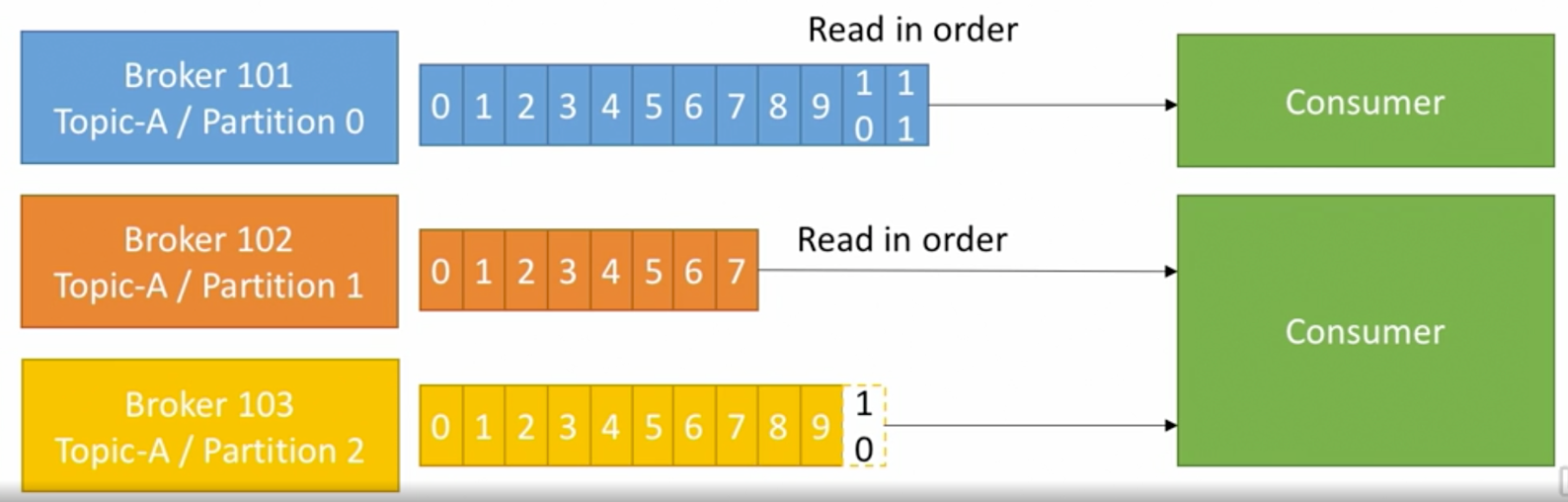
* **Concept of leader for a partition:**
  + At any point, only one broker can be a leader for a given partition.
  + And only that leader can send and receive the data for a partition.
  + The other brokers only synchronize the data.
  + So, each partition has One leader and multiple **in-sync Replica (ISR)**



* **Producers:**
  + Producers are the entities in kafka that write data in a topic.
  + Producers know by default to which partition and broker to write data to.
  + In case any broker goes down, the producers will automatically recover.

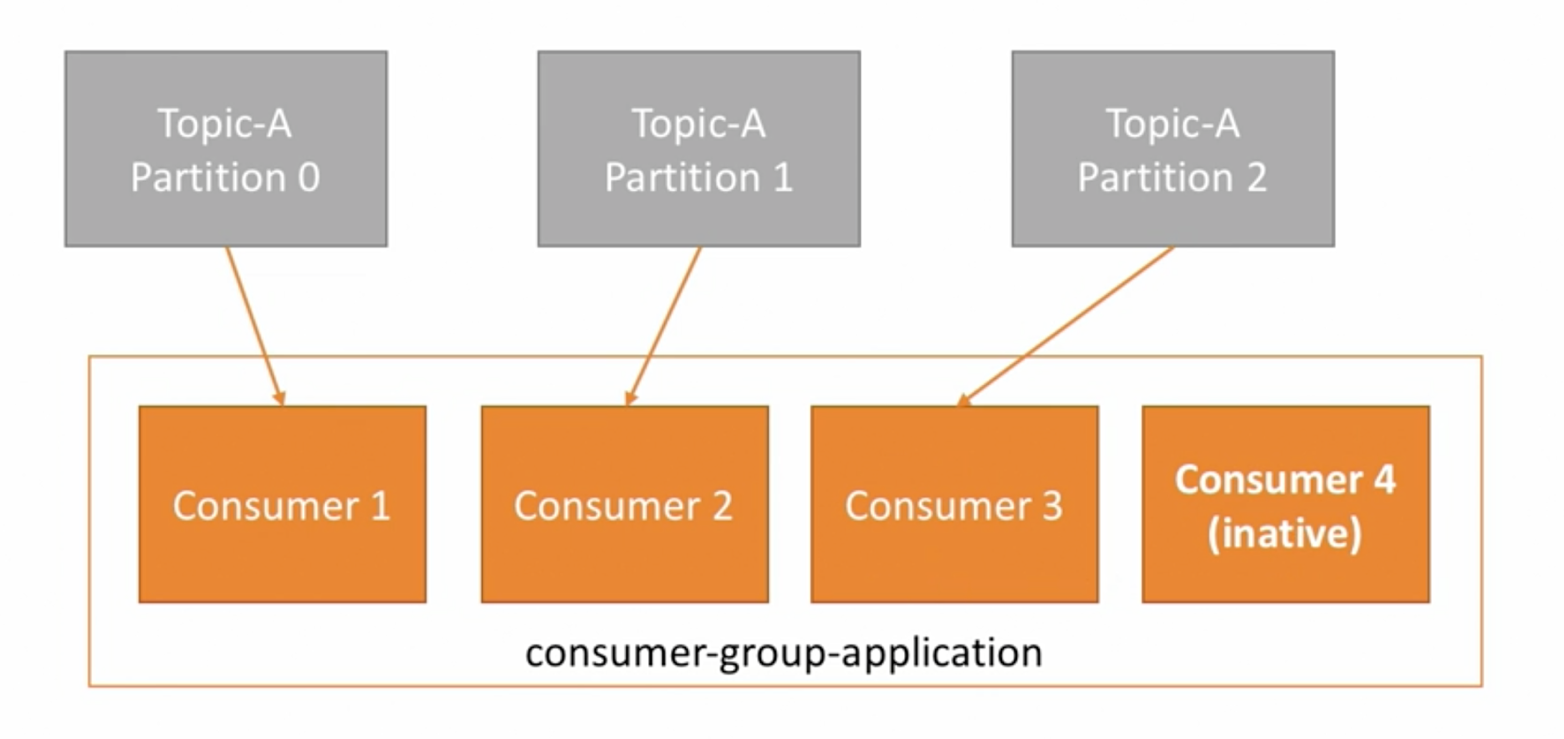


* **Writing data in topic:**
  + Producers are responsible to write the data in a topic.
  + **In the absence of key (key = null):**
    - The producer writes the data following the round robin technique on each partition.
  + **In the presence of key (key = any string, number etc):**
    - The producer writes in such a way that it is guaranteed that data with same key will always be written in a same partition.
    - In this case, a mapping is created by kafka itself for key to partition. This mapping is based on hashing, which depeds on number of partitions in a topic.
  + Producers can choose to receive the **acknowledgement** of data writes in a topic. There are three kinds of data write acknowledgement. They are:
    - acks=0 : Producer does not wait for receiving any acknowledgement for data write. There could be a possible data loss in this strategy.
    - acks=1 : (Default) the producer will wait for the leader acknowledgement (limited loss of data; since data on ISR may not be replicated)
    - acks=all : leader + replicas acknowledgement; No data loss.
* **Consumers:**
  + Consumers are the entities in kafka that reads the data form topic; the topic is identified by its name.
  + Consumers know by default which partition to read from.
  + In case of broker failure, consumer knows how to recover.
  + Data is read in order in each partition.
  + A consumer can read from single or multiple partitions (hence brokers); but the read will always be in order within partition.

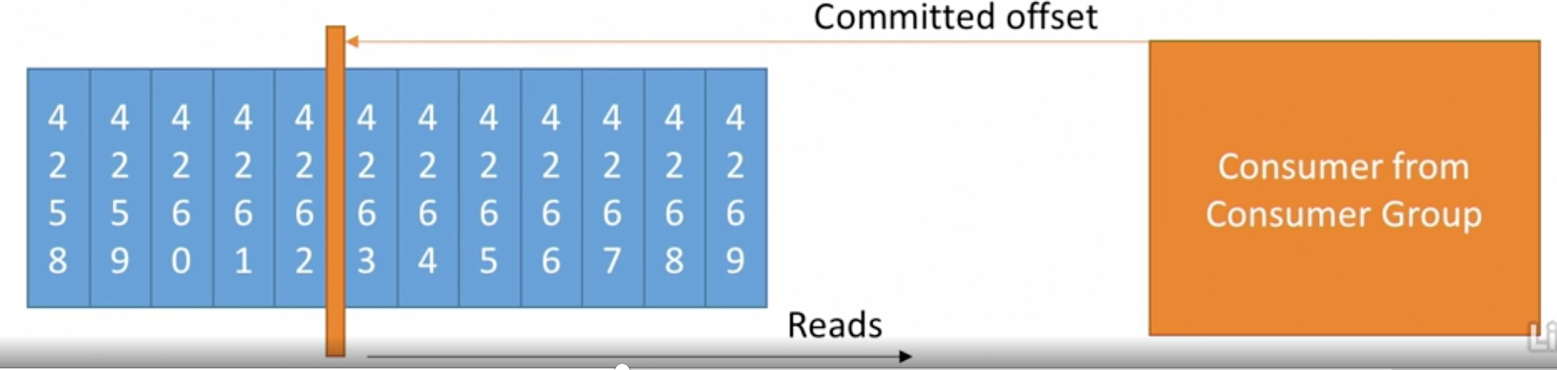


* **Consumer Groups:**
  + Consumers read data in consumer groups.
  + Each consumer within a group reads from exclusive partition.
  + If there are more consumers than partition, some consumers will be inactive.

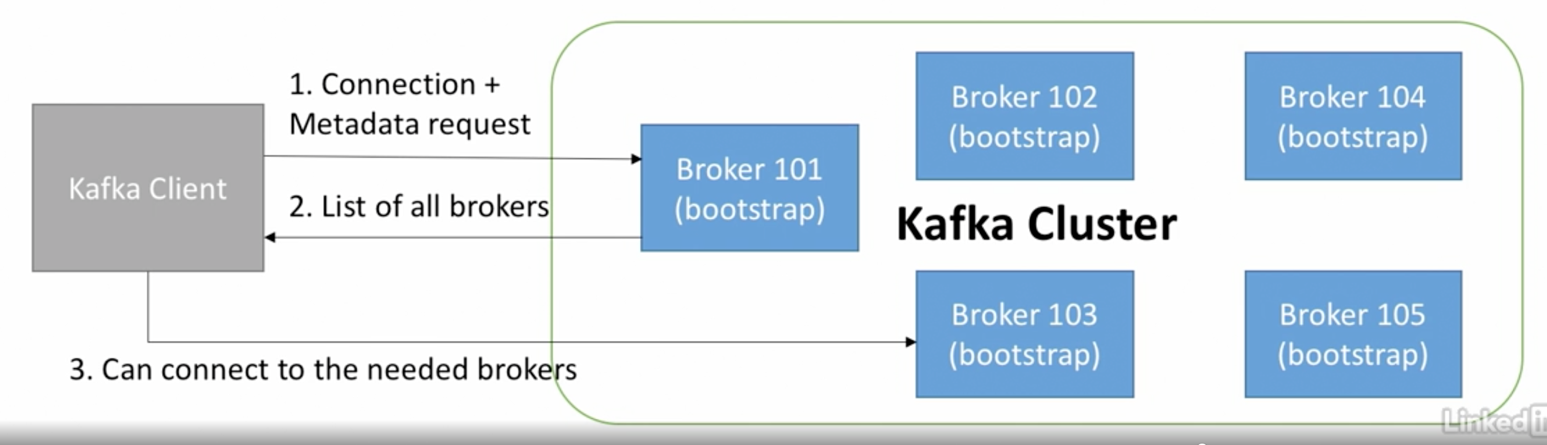




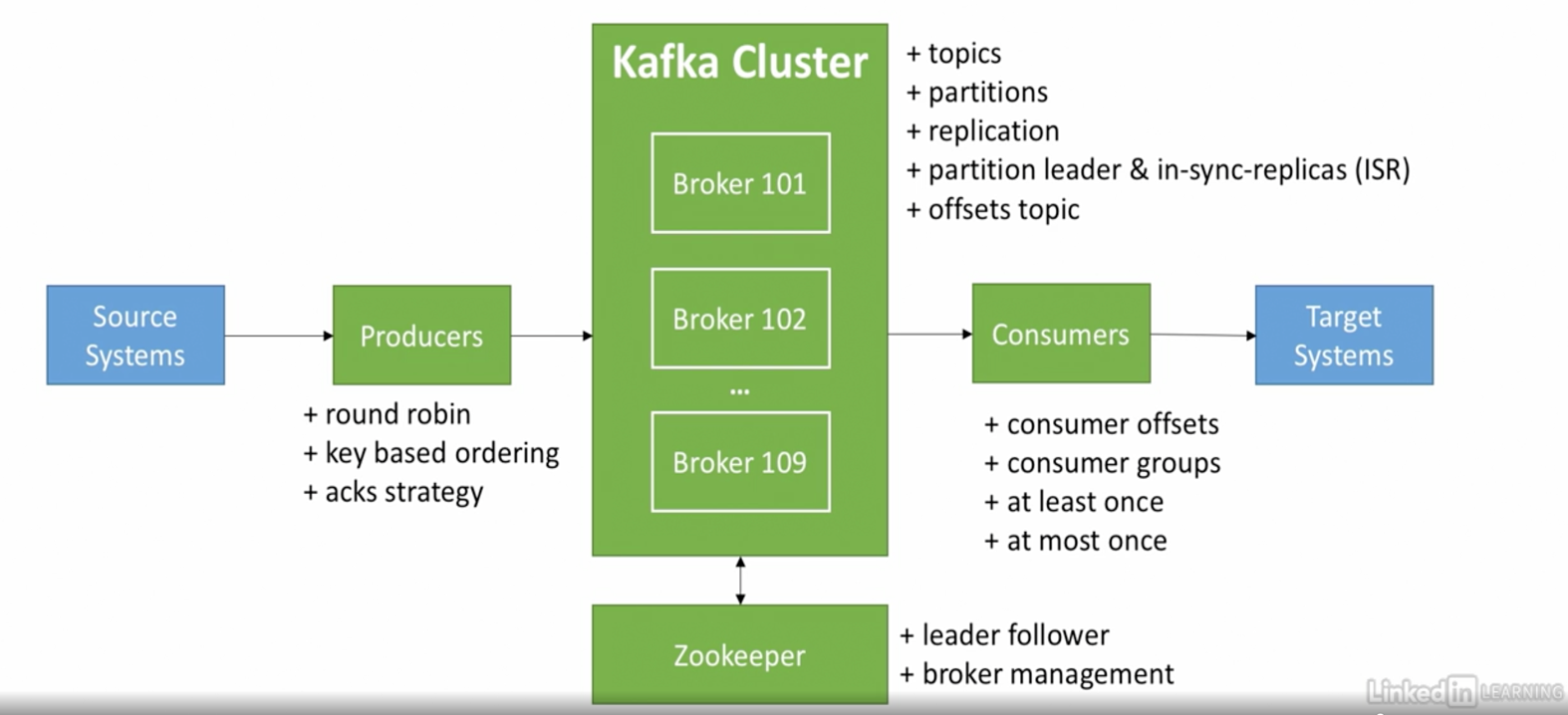
* **Consumer offsets:**
  + Kafka stores the offsets at which consumer group has been reading.
  + Offsets committed live in a kafka topic named- \_\_consumer\_offsets.
  + When a consumer in a group has processed data received from kafka, it commits the offsets. This is pre-programmed and we need not write logic for this.
  + **Importance:** if a consumer dies, it will be able to read back from where it left off. Thanks to the committed offsets.

****

* **Delivery semantics for consumers:**
  + Consumer can choose when to commit offsets.
  + There are three semantics for this:
    - **At most once:**
      * Offsets are committed as soon as the message is received.
      * If the processing goes wrong, the message would be lost and could not be read again.
    - **At least once (preferred):**
      * Offsets are committed after the message is processed.
      * If processing goes wrong, the message will be read again.
      * This can result in duplicate processing and hence, it should be made sure that processing is idempotent i.e. processing again the messages will not impact your systems.
    - **Exactly once:**
      * Can be achieved for kafka to kafka workflows using Kafka Streams API.
      * For kafka to external system workflows, use an idempotent consumer.
* **Kafka broker discovery:**
  + Each kafka broker is called the bootstrap broker.
  + This means that you only need to connect to one broker and you will be connected to entire cluster.
  + Each broker knows about the other broker, topic and partitions (metadata).



* **Zookeeper:**
  + Zookeeper manages all the brokers.
  + The zookeeper keeps the list of all the brokers in a cluster.
  + Zookeeper sends notification to the kafka in case of changes like new topic, broker dies, broker comes up, delete topic etc.
  + Kafka can not work without zookeeper.
  + Zookeeper by design operates with odd number of servers.
  + Zookeeper has a leader that handles the write operations, rest of the servers are the followers and handle the read operation.
  + Zookeeper does not store consumer offsets starting kafka version > 0.10
* **Theory roundup:**

****

**Kafka CLI setup:**

Install kafka cli using brew for mac

Install the binaries tar from <https://kafka.apache.org/downloads>

Extract the files from tar.

In cli, navigate to the directory extracted.

In this directory, all the kafka commands can be executed.

**CLI Important commands:**

The setup can be verified using command: (**Start the zookeeper**)

zookeeper-server-start config/zookeeper.properties

This command should start the zookeeper at localhost port 2181; which is default.

**Start the kafka server**, defaults to port 9092:

kafka-server-start config/server.properties

**creating topic:**

kafka-topics --create --topic first-topic --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

**List the topics present on a zookeeper:**

kafka-topics --list --bootstrap-server localhost:9092

**Describe a topic:**

kafka-topics --bootstrap-server localhost:9092 --topic first-topic –describe

**Delete a topic:**

kafka-topics --bootstrap-server localhost:9092 --topic testTopic –delete

**Kafka producer commands:**

**Starting the producer console:**

kafka-console-producer --bootstrap-server localhost:9092 --topic first-topic

(the prompter changes to > where messages can be provided)

**Starting the producer console ad setting acks all property:**

kafka-console-producer --bootstrap-server localhost:9092 --topic first-topic --producer-property acks=all

**Note:**

Starting a producer with a topic that does not exist yet; This will cause a warning when first message is sent to the producer. However, the producer is self-healing and **self-recovering** in nature. Hence the topic gets created after the error. Subsequent messages can be processed as usual.

The topic created in such a way has default properties- Replication factor 1 and Partitions 1. Which is not what is usually expected. So have a topic created prior to starting a producer on it.

These defaults ca be overridden in conf/server.properties

**Kafka Consumer commands:**

**Starting the consumer:**

kafka-console-consumer --bootstrap-server localhost:9092 --topic first-topic

message streaming is real time; a consumer starts reading from the point when the consumer is created/started.

**Consuming messages from the beginning:**

kafka-console-consumer --bootstrap-server localhost:9092 --topic first-topic --from-beginning

**Using consumer in consumer groups:**

**Staring a consumer in a consumer group:**

kafka-console-consumer --bootstrap-server localhost:9092 --topic first-topic --from-beginning –group <group name>

**list all consumers on a bootstrap server:**

kafka-consumer-groups –bootstrap-server localhost:9092 --list

**Describe a consumer group:**

kafka-consumer-groups –bootstrap-server localhost:9092 --describe --group <group name>

**Resetting consumer offsets to start:**

kafka-consumer-groups --bootstrap-server localhost:9092 –group <group name> --reset-offsets --to-earliest --execute --topic <topic name>

**Resetting consumer offsets:**

kafka-consumer-groups --bootstrap-server localhost:9092 –group <group name> --shift-by 2 --execute --topic <topic name>

**Note:** shift the offsets forward-backward by using – before number of shifts.