kafka

Data streaming with Kafka Streams

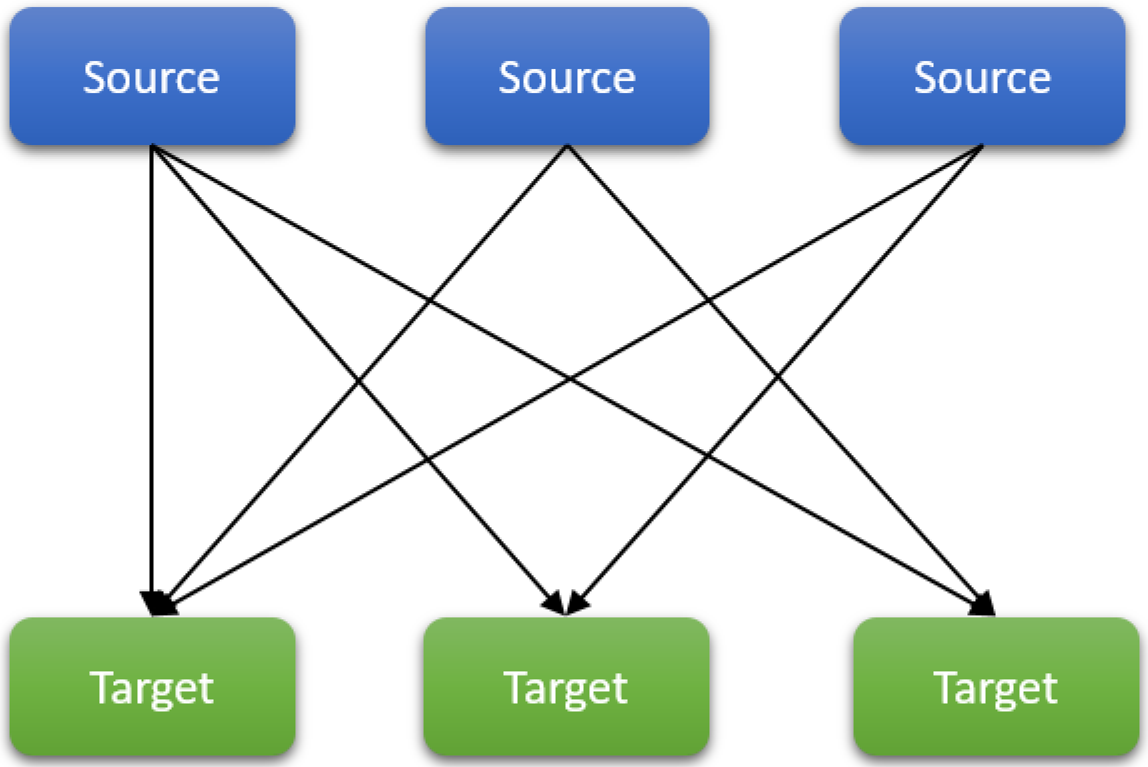
Kafka ?

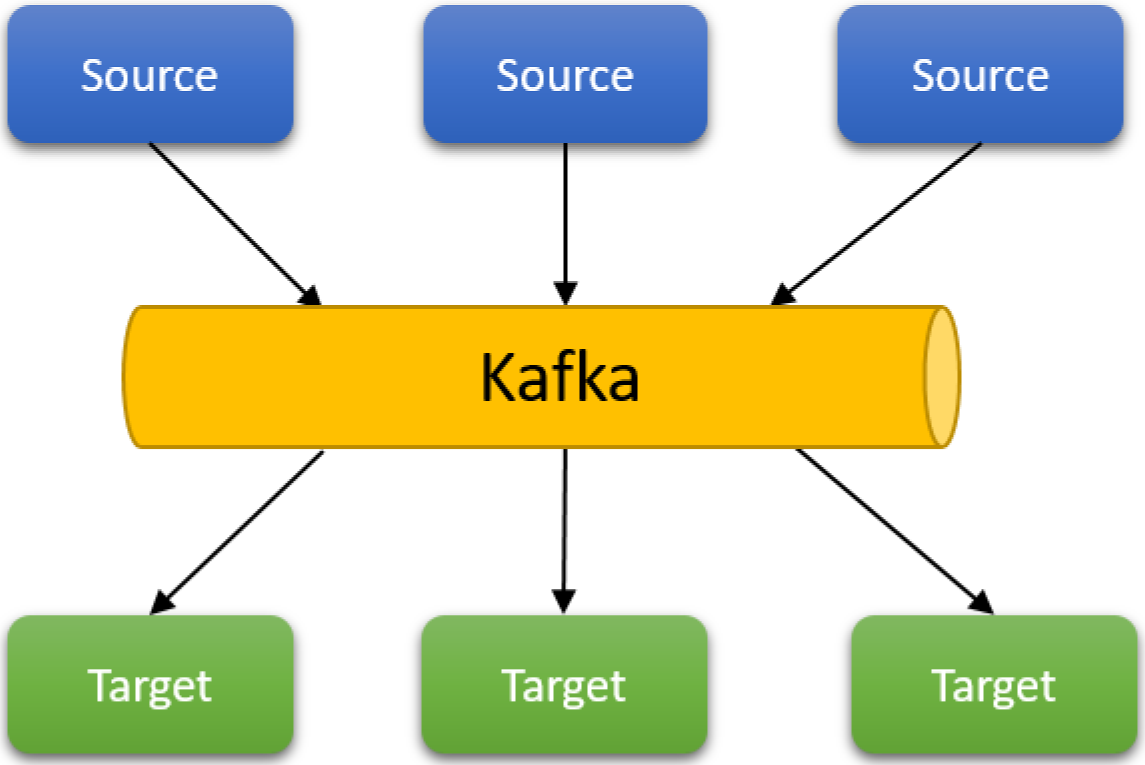
Use Cases

- Streaming processing
- Tracking user activity
- Log Aggregation
- De Coupling systems

Kafka ?

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.


Source
Source
Source
Target
Target
Target

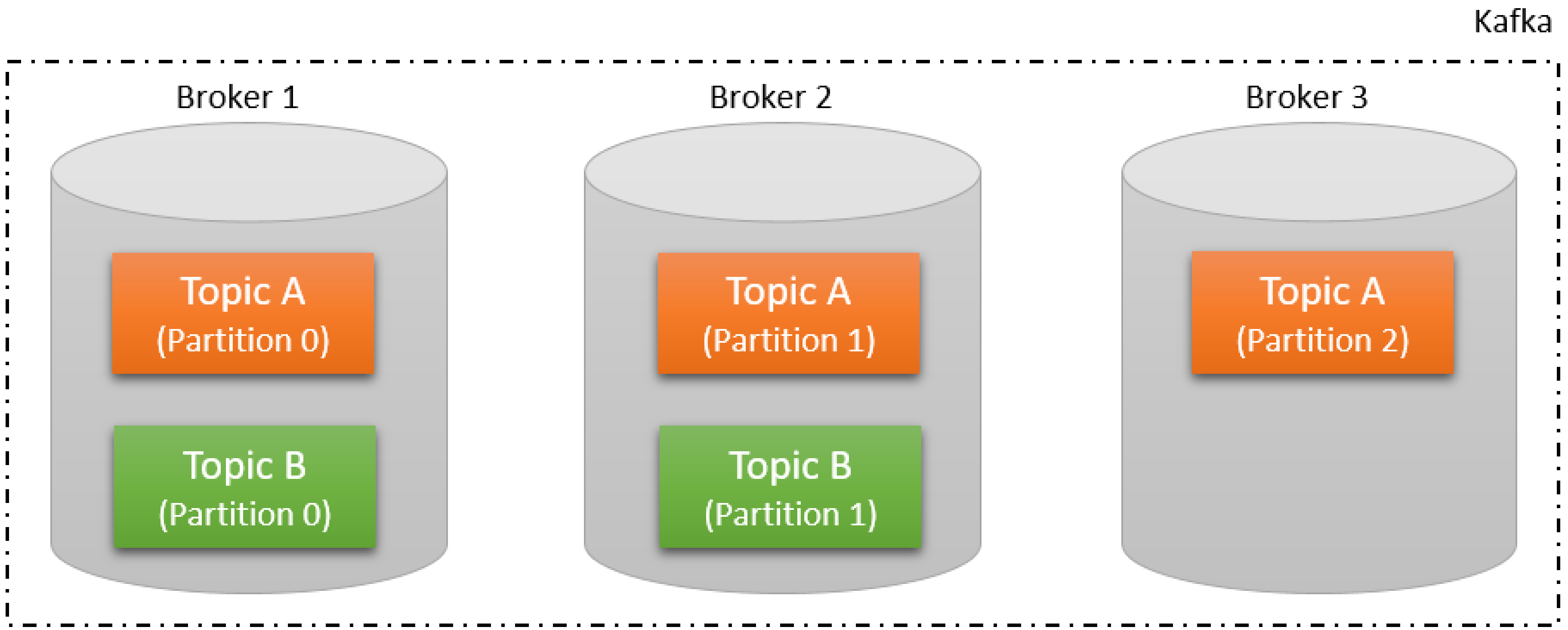

Source
Source
Source
Kafka
Target
Target
Target

Kafka

- Scale to 100s of nodes
- Handle millions of messages per second
- Real time data processing
- Event tracking
- Event Sourcing

Topics

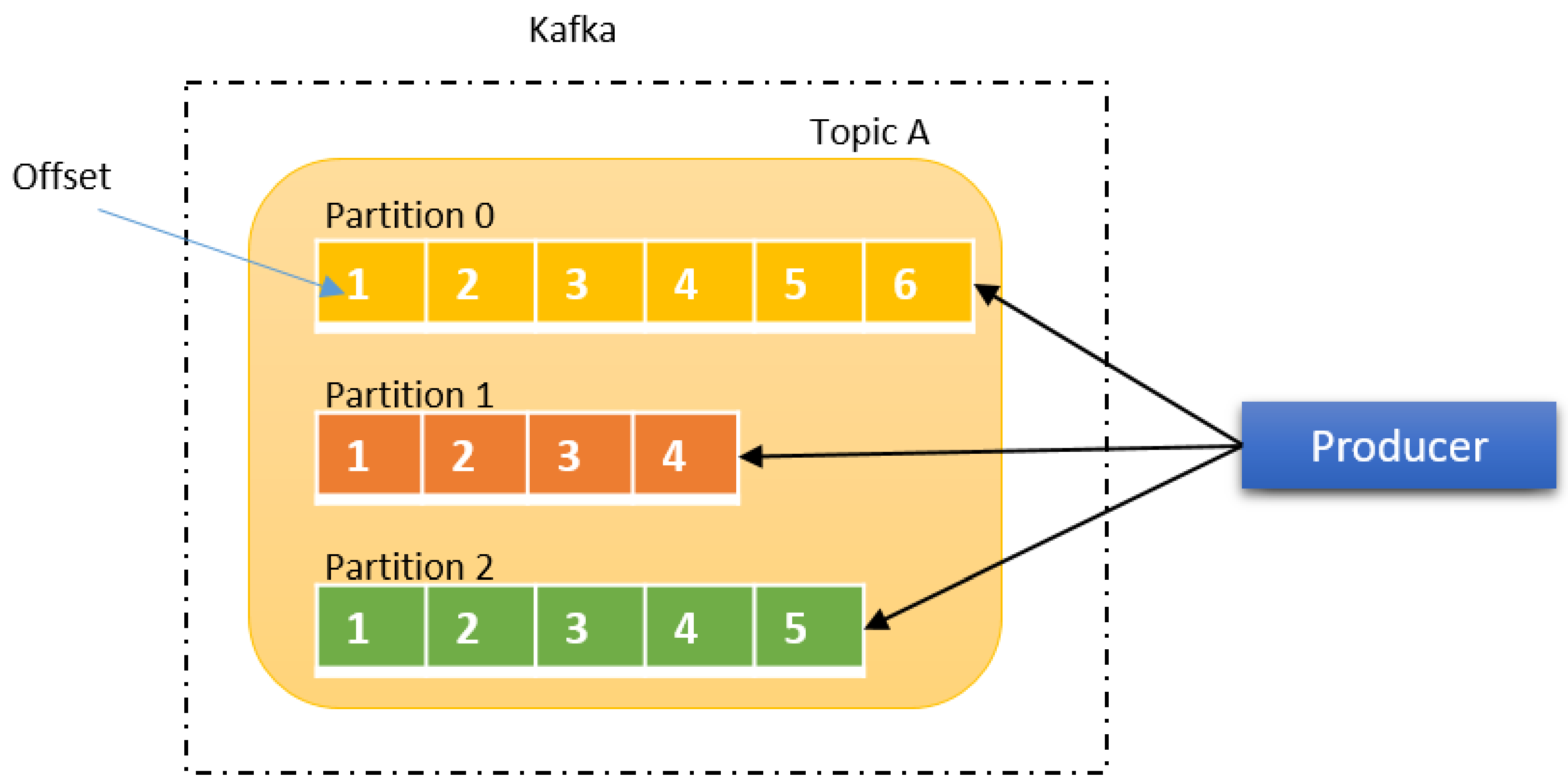
- Logical name to send messages
- Topics will be split into multiple partitions
- Partition enables topics to be distributed across the cluster
- Partition enable horizontal scalability
- Unit of parallelism
- Topics can span across nodes with partition



- Topic A – 3 Partitions
- Topic B – 2 Partitions

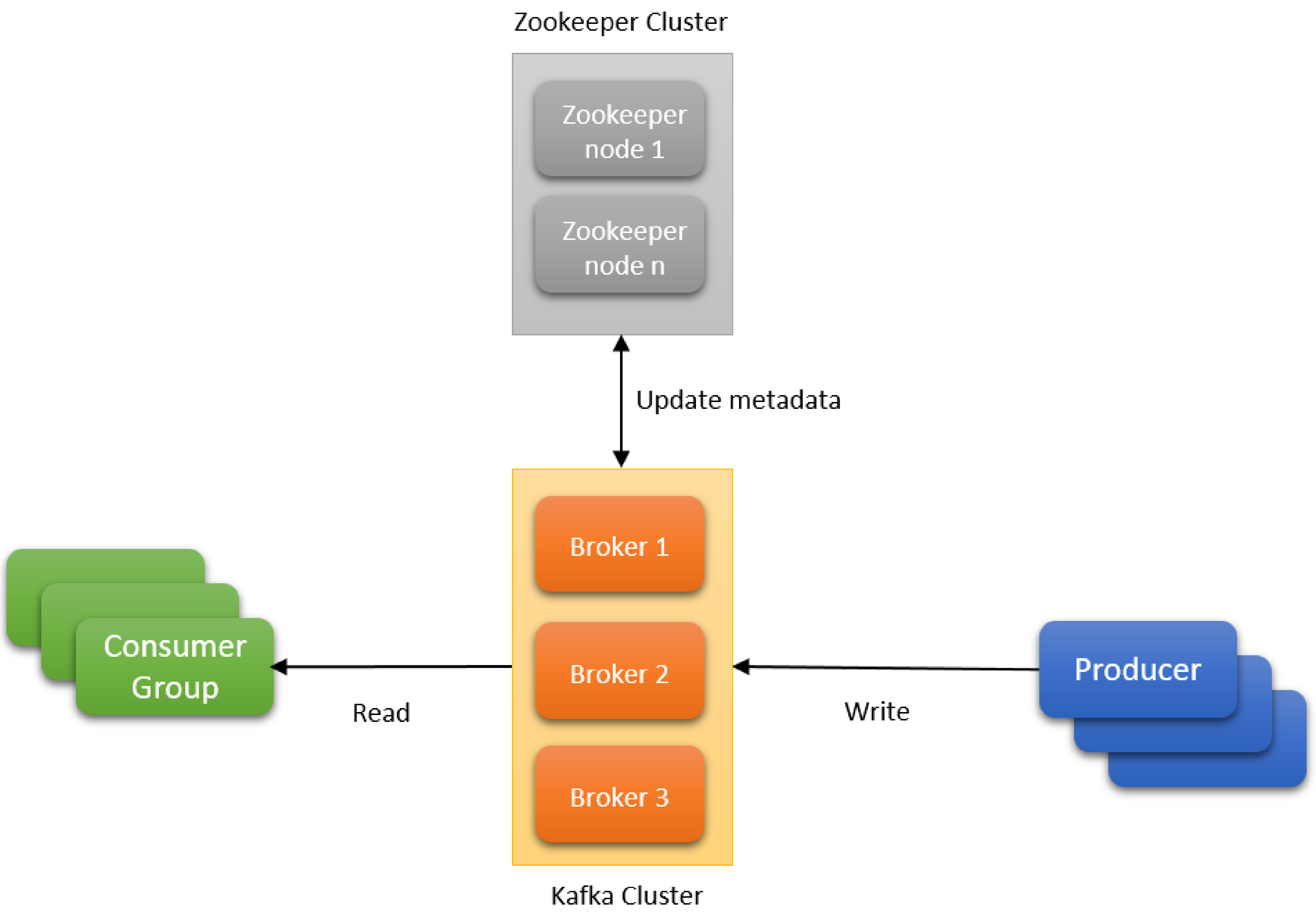
Partitions

- Message in partition are assigned offset
- Offsets are unique to partition
- Messages are ordered only in a partition
- Partition key dictates the partition without which the messages will be assigned to a random partition


Kafka
Topic A
Offset
Partition 0
1 2 3 4 5 6
Partition 1
1 2 3 4
Partition 2
1 2 3 4 5
Producer

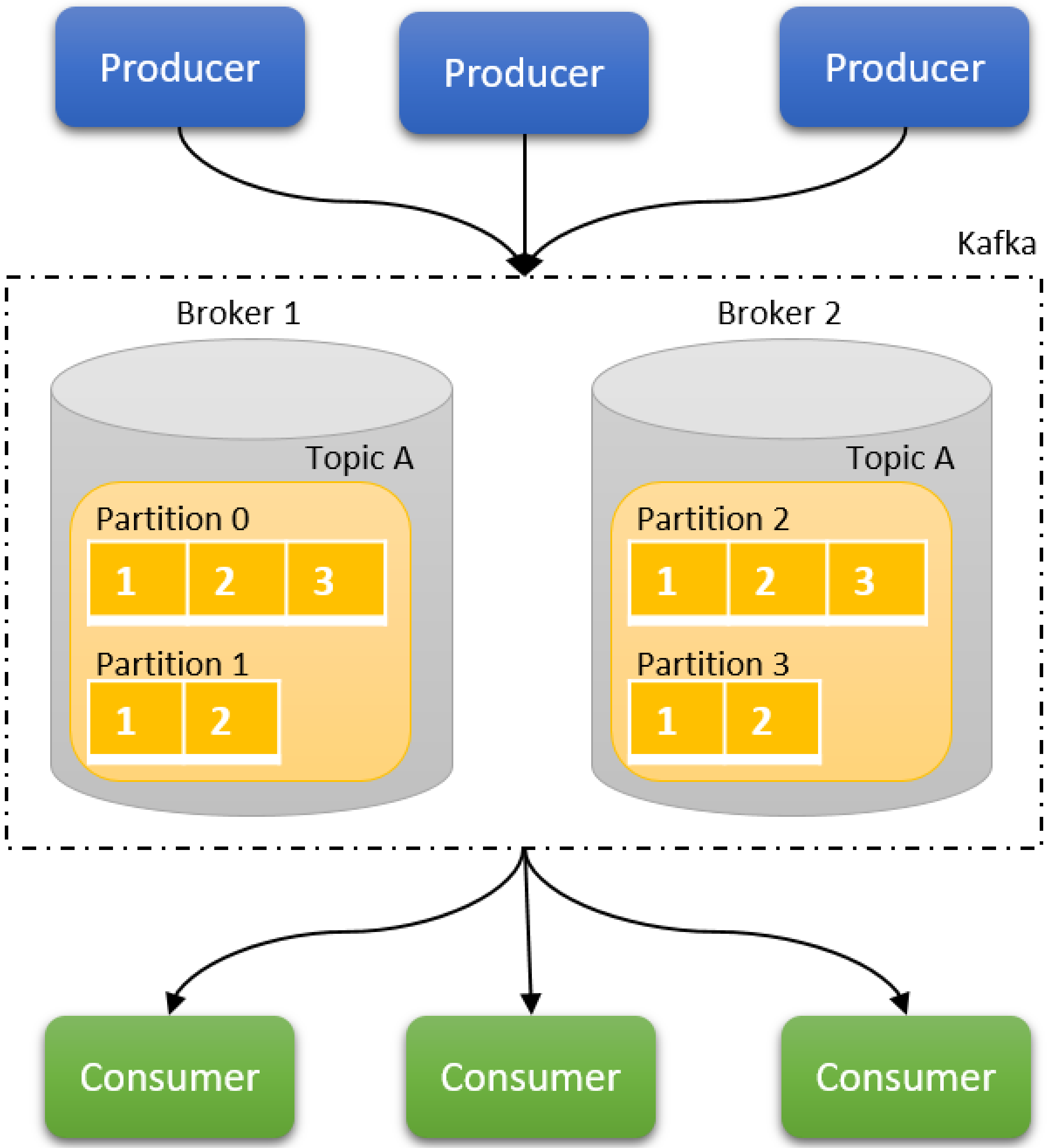
Zookeeper

- Manages the list of brokers
- Elects a leader broker in case of broker failure
- Leader zookeeper handles all the writes
- Follower zookeeper handles only the reads.


Zookeeper Cluster
Zookeeper node 1
Zookeeper node n
Update metadata
Broker 1
Broker 2
Broker 3
Kafka Cluster
Consumer Group
Read
Write
Producer

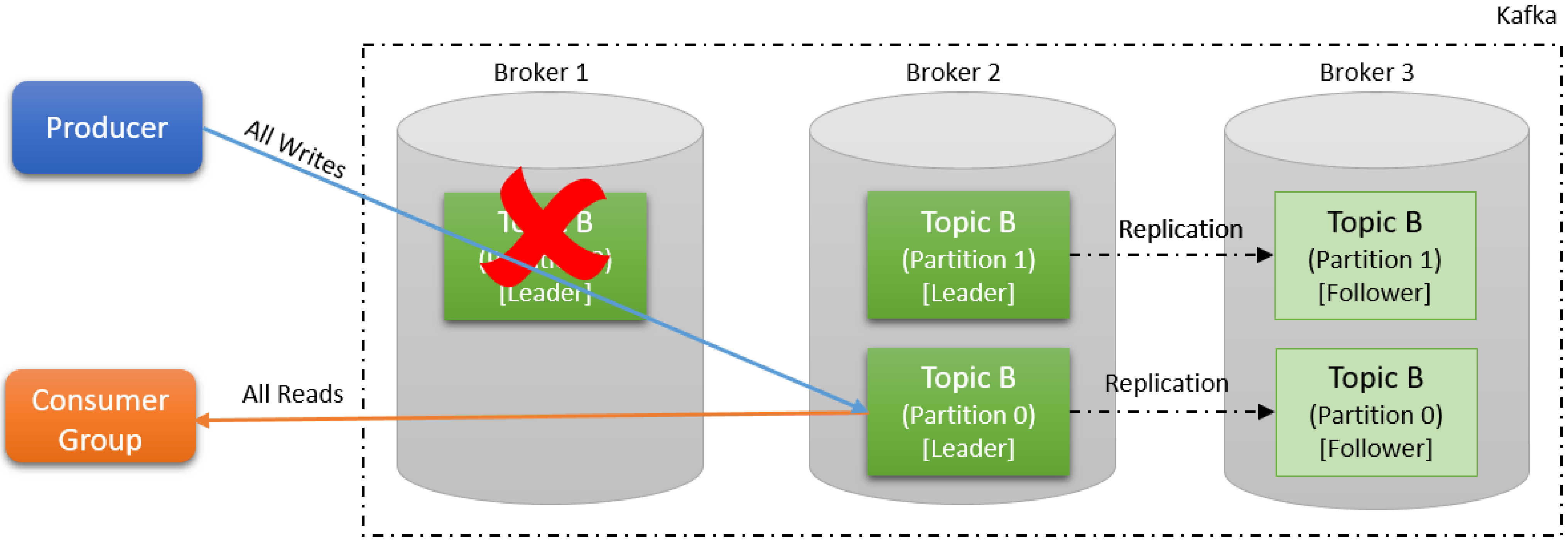
Broker

- Single node managed by the Zookeeper
- Set of brokers form a cluster
- Topics created are distributed across brokers based on partition and replications
- In case of broker node failure, the zookeeper rebalances the cluster
- If a leader partition is lost, the follower partition is elected as a new leader


Producer
Producer
Producer
Kafka
Broker 1
Broker 2
Topic A
Topic A
Partition 0
1 2 3
Partition 1
1 2
Partition 2
1 2 3
Partition 3
1 2
Consumer
Consumer
Consumer

Replication

- Making a copy of the partition available in another broker
- Makes Kafka fault tolerant to broker and partition failure
- Leader partition is elected in case of partition available on multiple brokers
- The remaining partitions are called followers
- Both producers and consumers are served by the same broker
- In case of broker failure, the partition from another broker is elected as leader and starts serving the producers and consumers

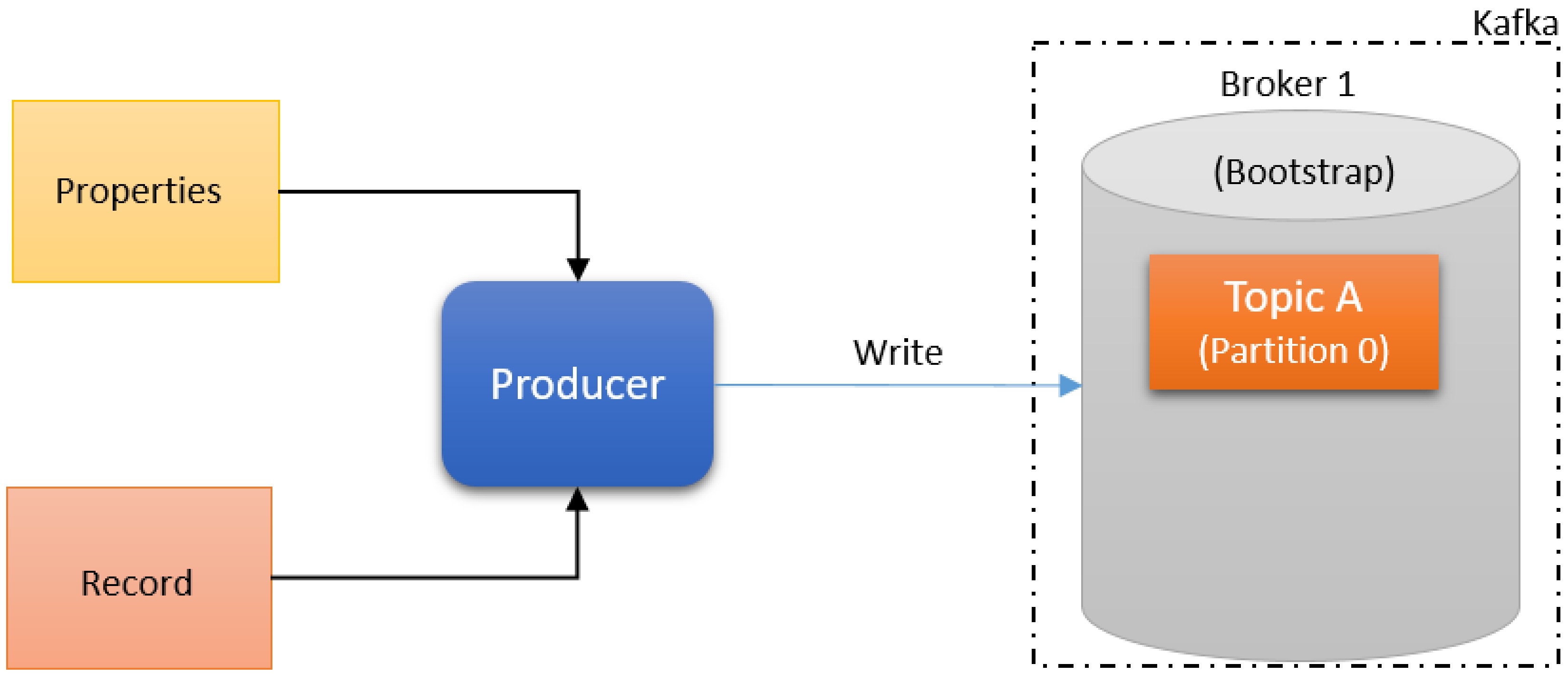

Kafka
Broker 1
Broker 2
Broker 3
Producer
All Writes
Consumer Group
All Reads
Topic B (Partition 0) [Leader]
Topic B (Partition 1) [Leader]
Topic B (Partition 0) [Leader]
Replication
Replication
Topic B (Partition 1) [Follower]
Topic B (Partition 0) [Follower]

- Topic B – 2 Partitions
- Replication factor of 2

Producers

Producer

- Writes the messages to appropriate broker
- Perform message serialization, partitions, compression.
- Load balances messages across brokers based on the partitions.
- In case of broker failure, the partition from another broker is elected as leader and starts serving the producers and consumers

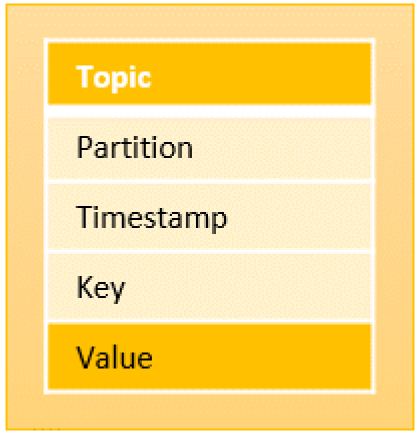

Kafka
Broker 1
Properties
(Bootstrap)
Topic A
(Partition 0)
Producer
Write
Record

Producer Properties

- Bootstrap servers (brokers)
- Acknowledgements
- Batch.size
- Key.serializer
- Value.serializer

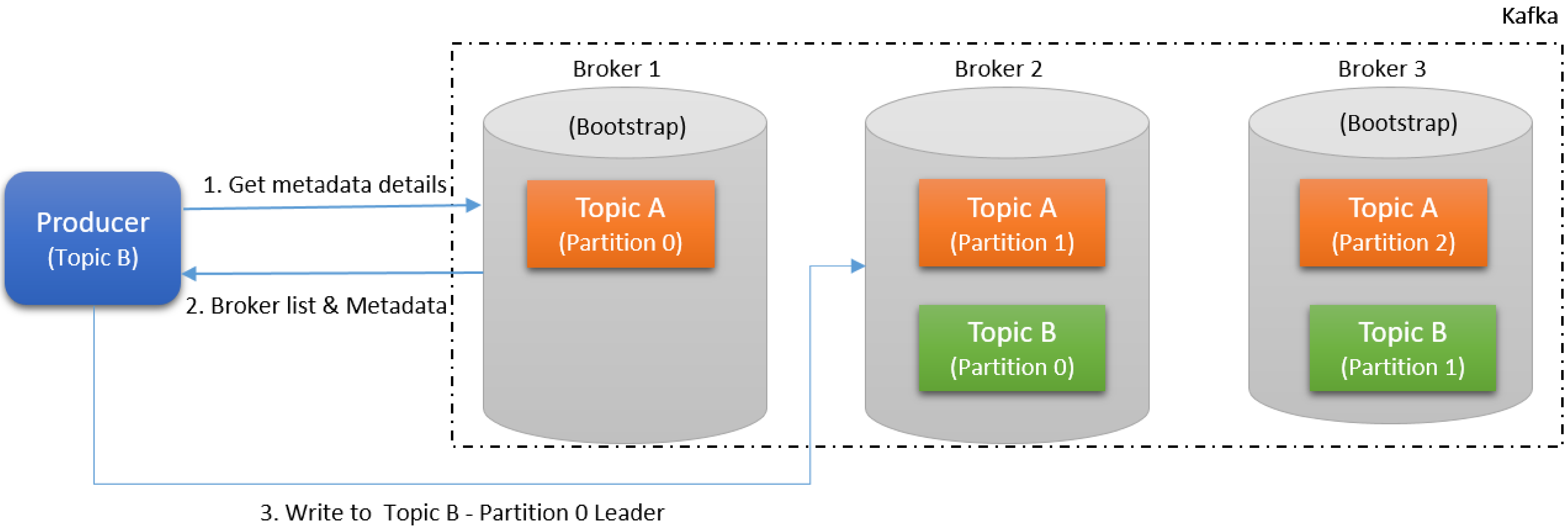
Producer Record

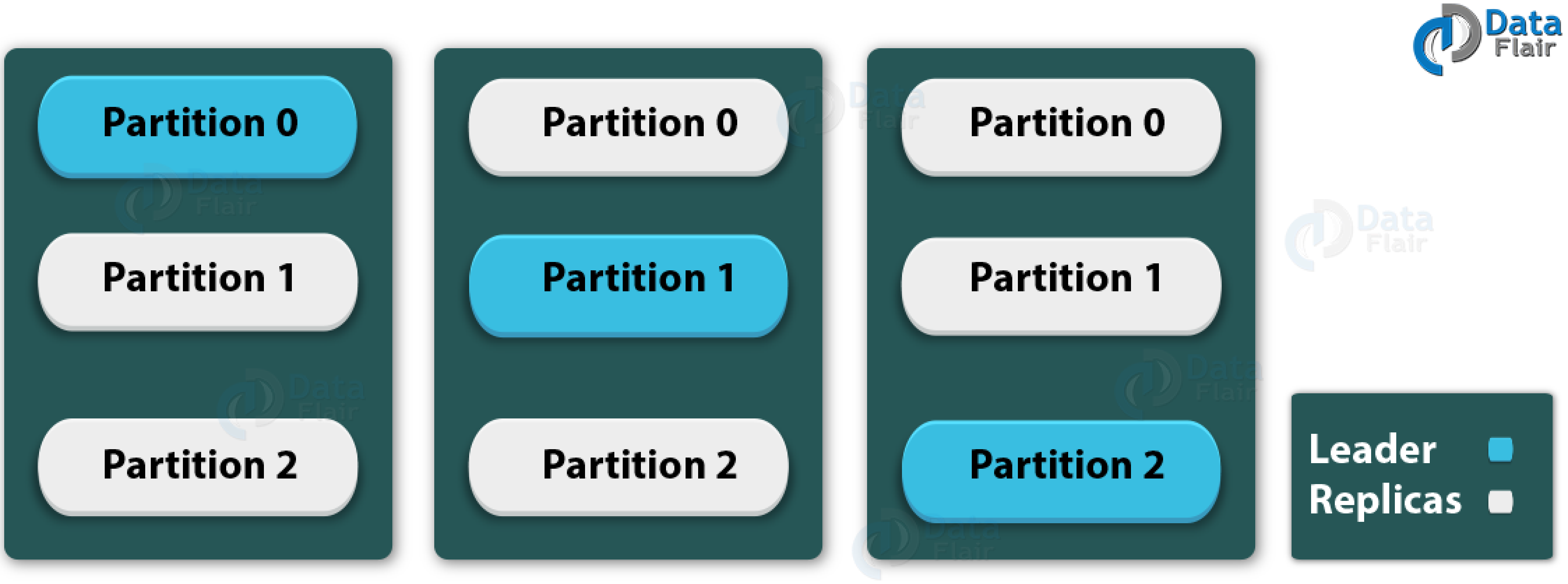
- The message written to the Kafka is called as a producer record
- Name of the topic and value is mandatory
- Other fields like partition, timestamp and key are optional


Topic
Partition
Timestamp
Key
Value

Key Points

- Bootstrap servers – list of brokers are called bootstrap servers. At least 2 bootstrap servers are recommended.
- Producer first establishes the connection with the one of the bootstrap server
- The bootstrap server returns all the brokers and the metadata like topics, partitions, replication factor etc.
- The Producer then identifies the leader broker for the leader partition and post the message to the leader broker.

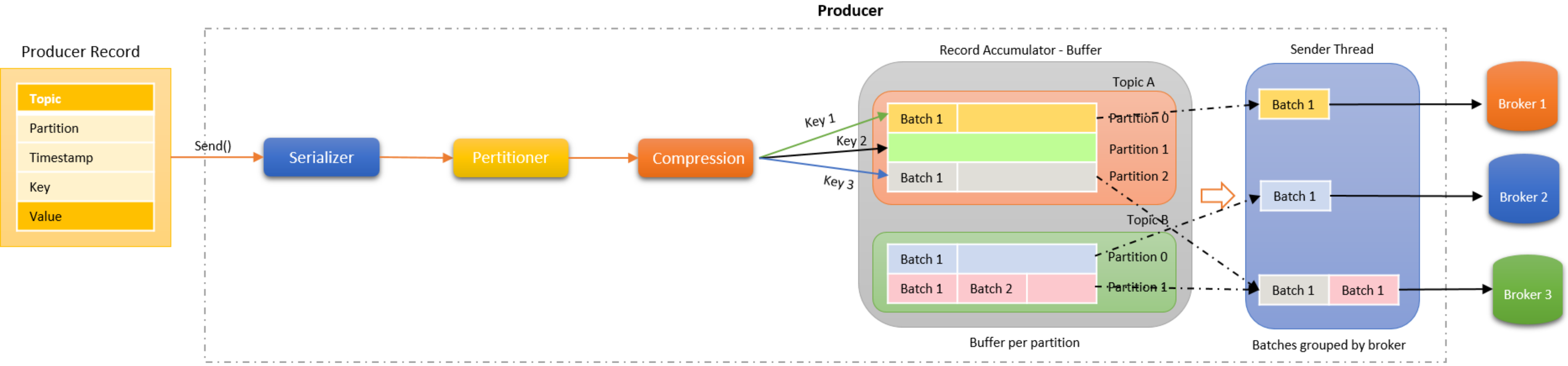

Kafka
Broker 1
Broker 2
Broker 3
(Bootstrap)
(Bootstrap)
Producer
(Topic B)
1. Get metadata details
2. Broker list & Metadata
3. Write to Topic B - Partition 0 Leader
Topic A
(Partition 0)
Topic A
(Partition 1)
Topic A
(Partition 2)
Topic B
(Partition 0)
Topic B
(Partition 1)


Partition 0
Partition 1
Partition 2
Partition 0
Partition 1
Partition 2
Partition 0
Partition 1
Partition 2
Leader
Replicas

Workflow

Workflow

- Serializer
  - Producer first serialize the record based on the serializers passed.
  - Both key and value are serialized
- Partition
  - Producer decides which partition of the topic the message to be written to.
  - Partition is decided after hashing the partition key
  - In case of no partition key, round-robin algorithm is used


Producer
Producer Record
Topic
Partition
Timestamp
Key
Value
Send()
Serializer
Partitioner
Compression
Key 1
Key 2
Key 3
Record Accumulator - Buffer
Topic A
Batch 1
Partition 0
Partition 1
Batch 1
Partition 2
Topic B
Batch 1
Partition 0
Batch 1
Batch 2
Partition 1
Buffer per partition
Sender Thread
Batch 1
Batch 1
Batch 1
Batch 1
Batches grouped by broker
Broker 1
Broker 2
Broker 3

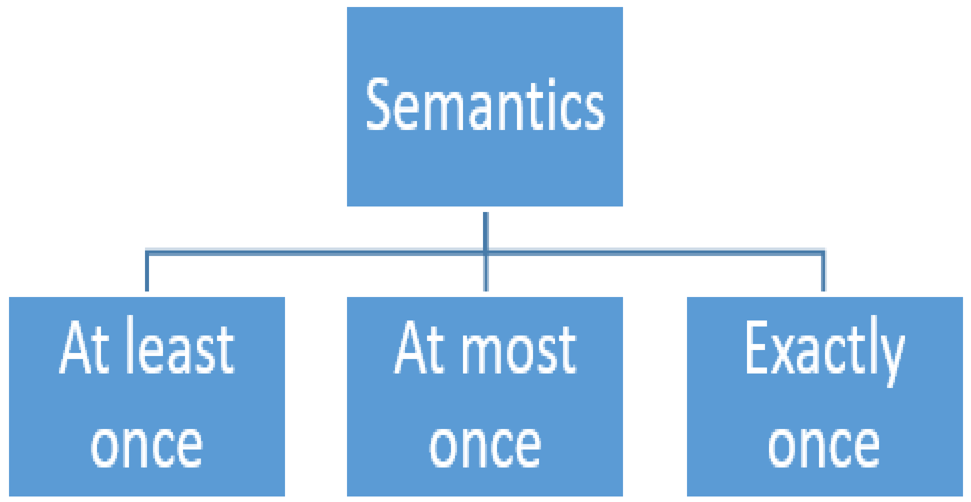
Workflow

- Partition
  - Order is maintained per partition in the order received
  - Partition key plays the role to decided which partition the message is written to
  - Custom partition can be used to control the partition to be written to.
- Compression
  - Records are compressed before they are written to the record accumulator.
  - Enables faster replication and faster transfer
  - Helps better throughput , low latency and better disk utilization.

Workflow

- Record Accumulator
  - Records are accumulated in buffer per partition of the topic.
  - Records are grouped into batches based on the producer ***batch.size*** property.
  - Each partition in a broker gets a separate buffer/accumulator.
- Sender Thread
  - Batches of the partition in the record accumulator are grouped by broker to which they are sent.
  - The records are dispatched to the broker.

Producer delivery Semantics


Semantics
At least once
At most once
Exactly once

At least once

- Records are acceptable to be read more than once.
- Unacceptable to loose a message
- Can result in message duplication.
- Retries are allowed by the producer till the producer gets the acknowledgement from the broker.
- Ensures that all messages are delivered.
- Most preferred semantics of all.
- Results in moderate throughput and moderate latency.

At most once

- Records are delivered at the most once or maximum one time.
- Acceptable to loose a message
- No retries and follow send and forget protocol.
- Unacceptable to have duplicate messages
- Application adopting at most once can easily achieve high throughput and low latency

  Ex: metrics, transaction, log events

Exactly once

- Records should be read only once.
- Unacceptable to loose a message
- Most difficult delivery semantics of all.
- Results in lower throughput and high latency.

	At most once	At least once	Exactly once
Duplicates	No	Yes	No
Data loss	Yes	No	No
Processing	Zero or one time	One or more times	Exactly one time

Producer delivery semantics

- Ack = 0
  - At most once delivery semantics
  - Producer do not wait for acknowledgement from the broker after sending message
  - Messages will not be retried.
  - Send and forget approach.
  - Chances of data loss is high
  - Can result in data loss in case the message has not reached the broker and the broker dies.

Producer delivery semantics

- Ack = 1
  - At least once delivery semantics
  - Producer wait for response from the broker after sending message
  - Messages will be retried in case the producer does not receive the acknowledgement from the broker.
  - Producer will retry based on the configuration value.
  - Default value of retry is 0.
  - Moderate chances of data loss
  - Data loss can happen after sending the ack but before replication to other broker fails and in this case producer will not retry sending the record.

Producer delivery semantics

- Ack = All
  - Exactly once delivery semantics
  - Producer wait for response from the broker after sending message
  - Messages will be retried in case the producer does not receive the acknowledgement from the broker.
  - Producer will retry based on the configuration value.
  - Broker will send the ack only after ISR is met.
  - Less chances of data loss
  - Data loss can happen after sending the ack but all the brokers with ISR failed.

Acks	Latency	Throughput	Durability
0	Low	High	No guarantee
1	Medium	Medium	Leader only
All	High	Low	All Replicas

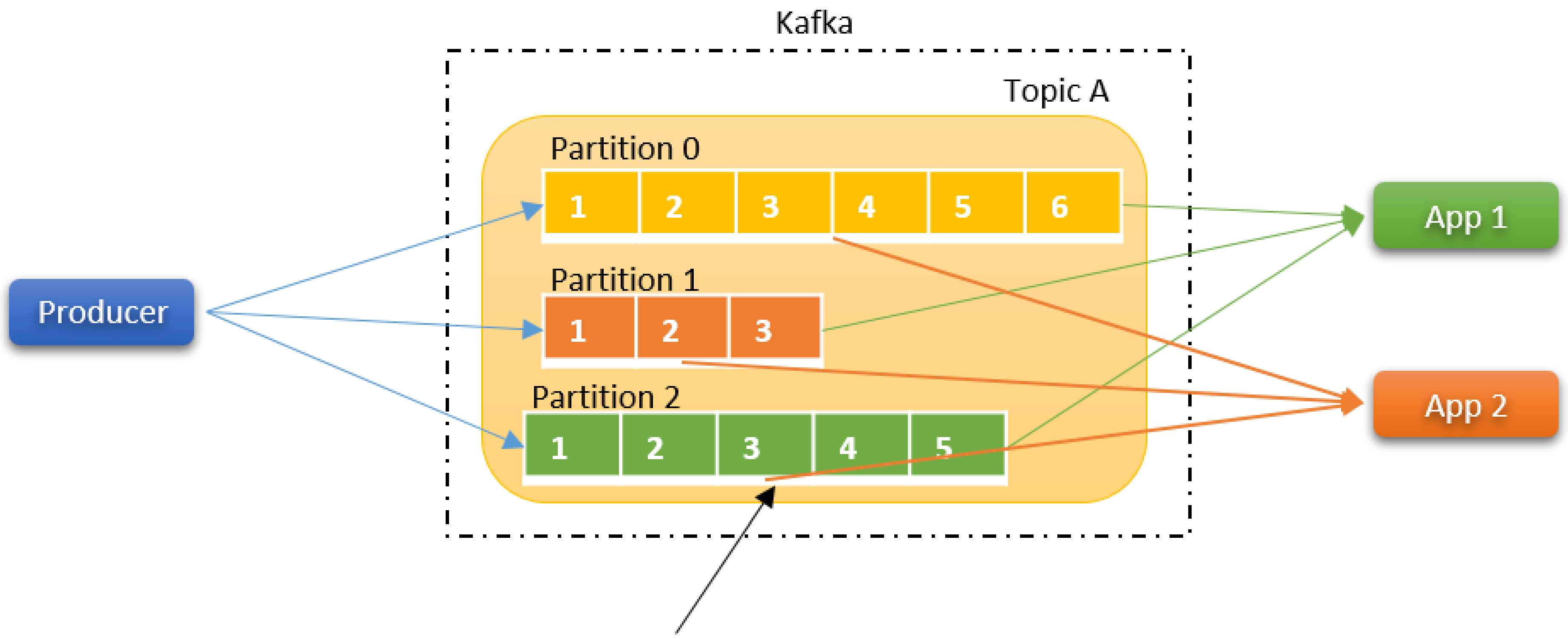
Consumer delivery Semantics

Consumer delivery semantics

- Consumer
  - Kafka Consumer can subscribe to one or more topics
  - Can also subscribe to a list of topics matching a regular expression
  - Optimal consumption of data
  - Takes a Kafka connection and consumer properties to read records from the appropriate broker.

Consumer delivery semantics

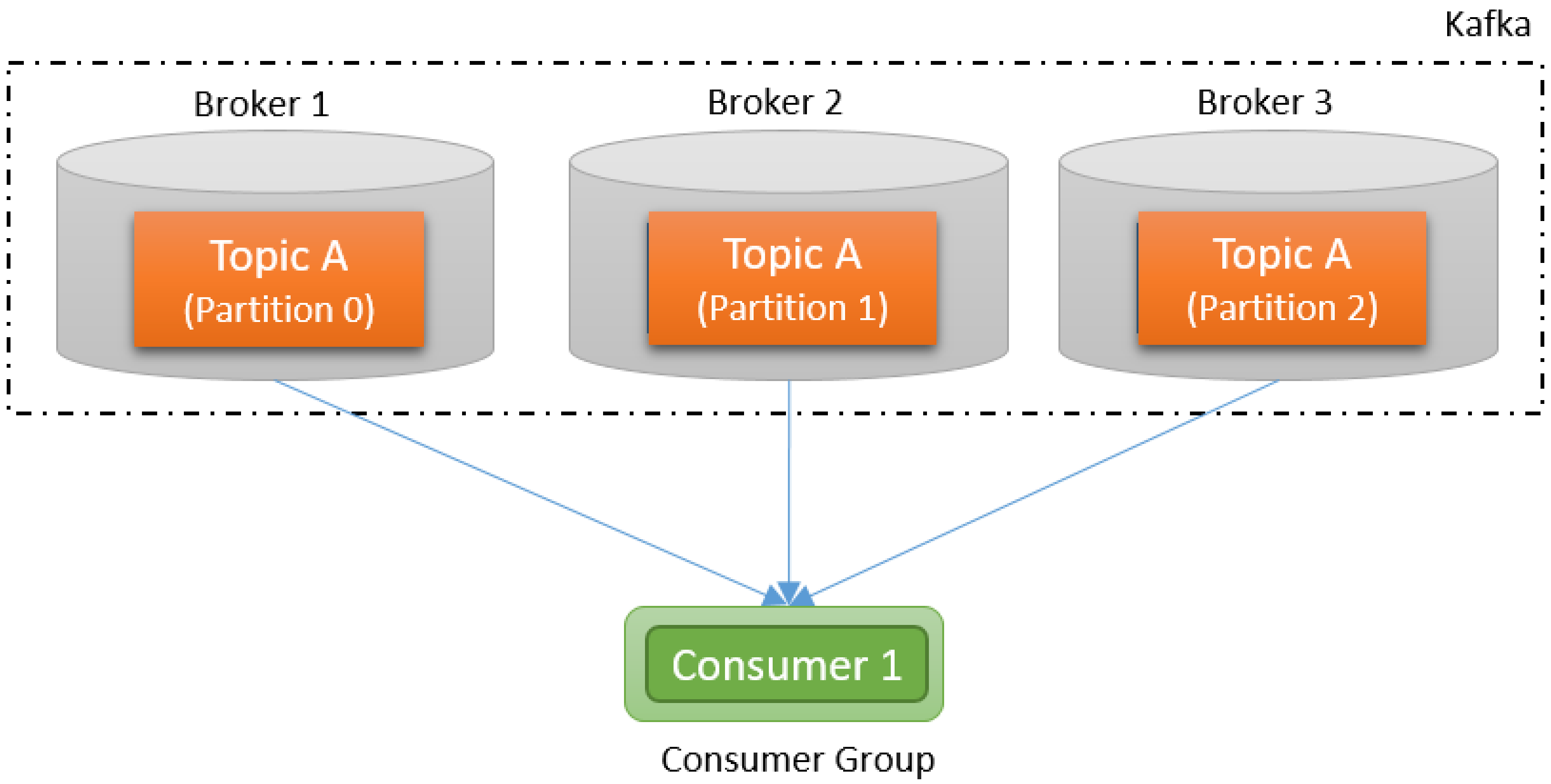
- Multi app Consumption
  - Multiple applications can consume records from the same topic
  - Each application that consumes the data from the broker, gets its own copy and read at its own speed.
  - Offset consumed from one application can be different from the another application.
  - Kafka keeps track of offsets consumed by each application in an internal topic called `_consumer_offset` topic.

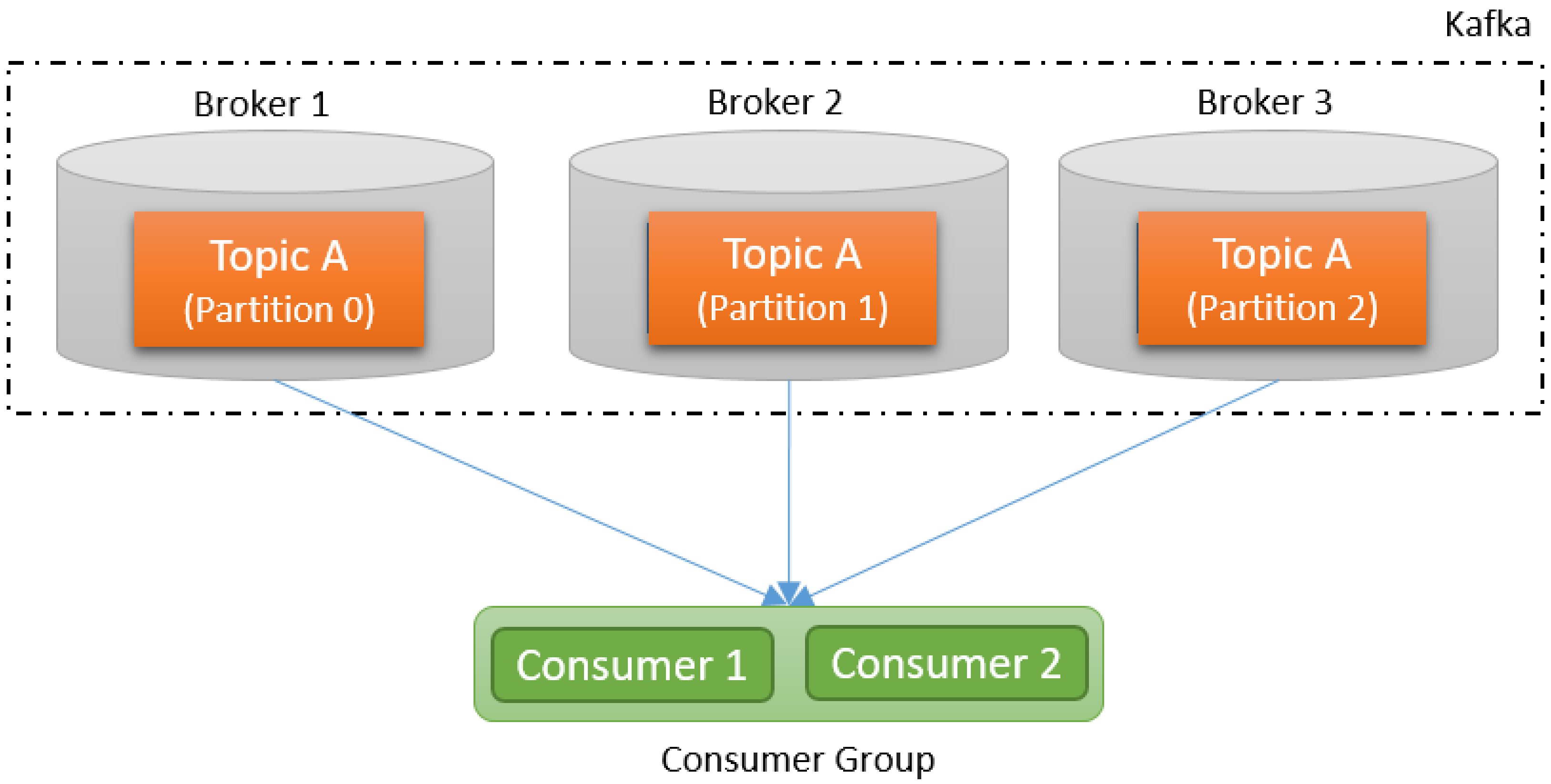


Consumed offset stored in internal "__consumer_offset" topic by application (Consumer group)

Consumer Group

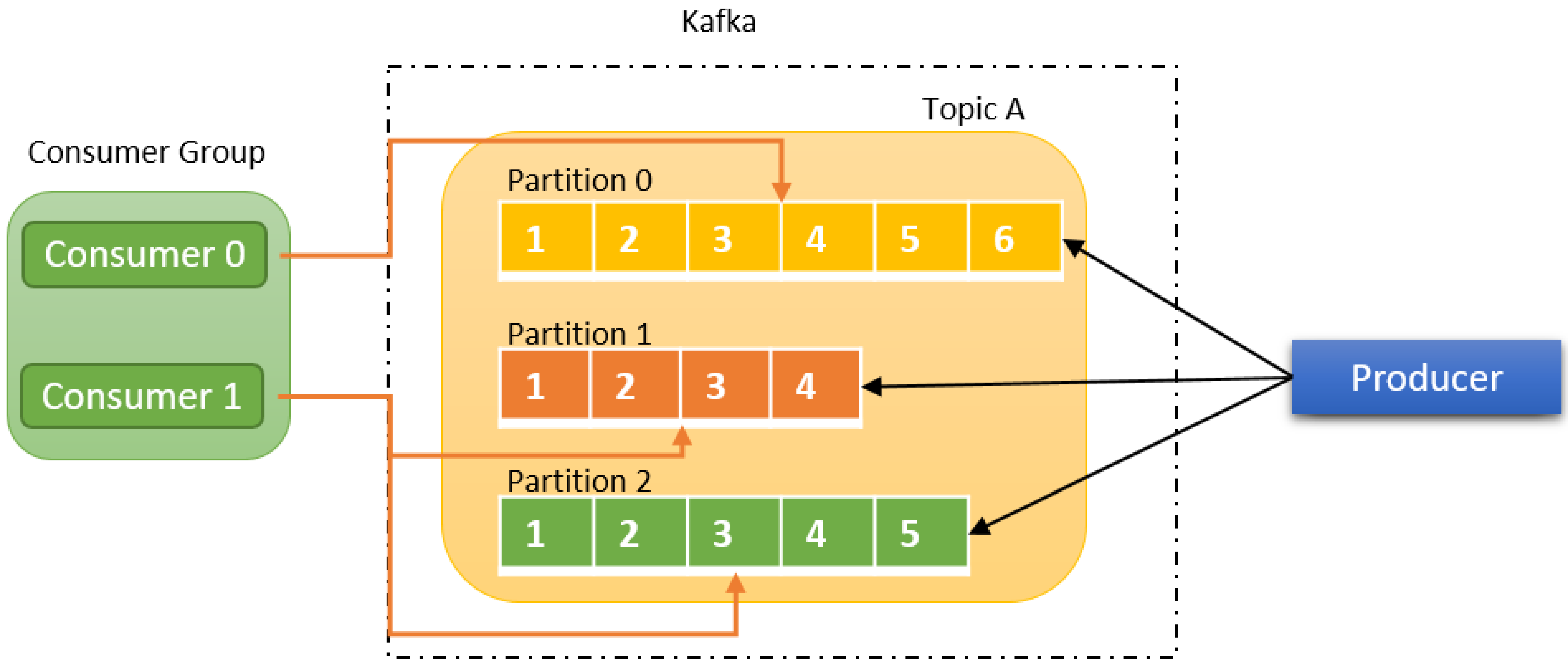
- Each application consuming the records from Kafka is referred to as Consumer Group.
- Application = Consumer Group
- Each consumer group can have one or more consumers
- Consumer from a consumer group will consume all the partitions of a topic.
- To increase the rate of processing and in parallel, additional consumers can be added to the consumer group.
- Kafka takes care of keeping track of the offsets consumed per consumer in a consumer group and rebalancing when adding/removing consumers.

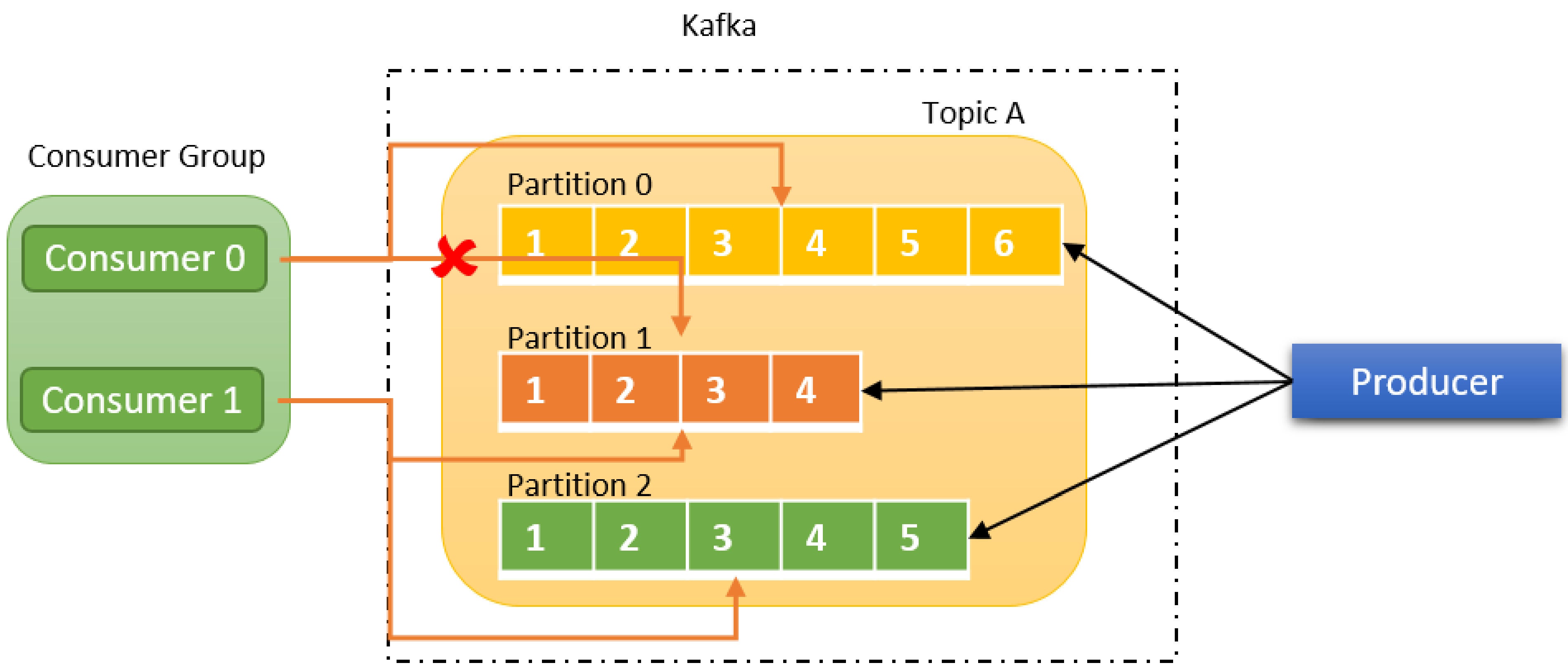

Kafka
Broker 1
Broker 2
Broker 3
Topic A (Partition 0)
Topic A (Partition 1)
Topic A (Partition 2)
Consumer 1
Consumer Group

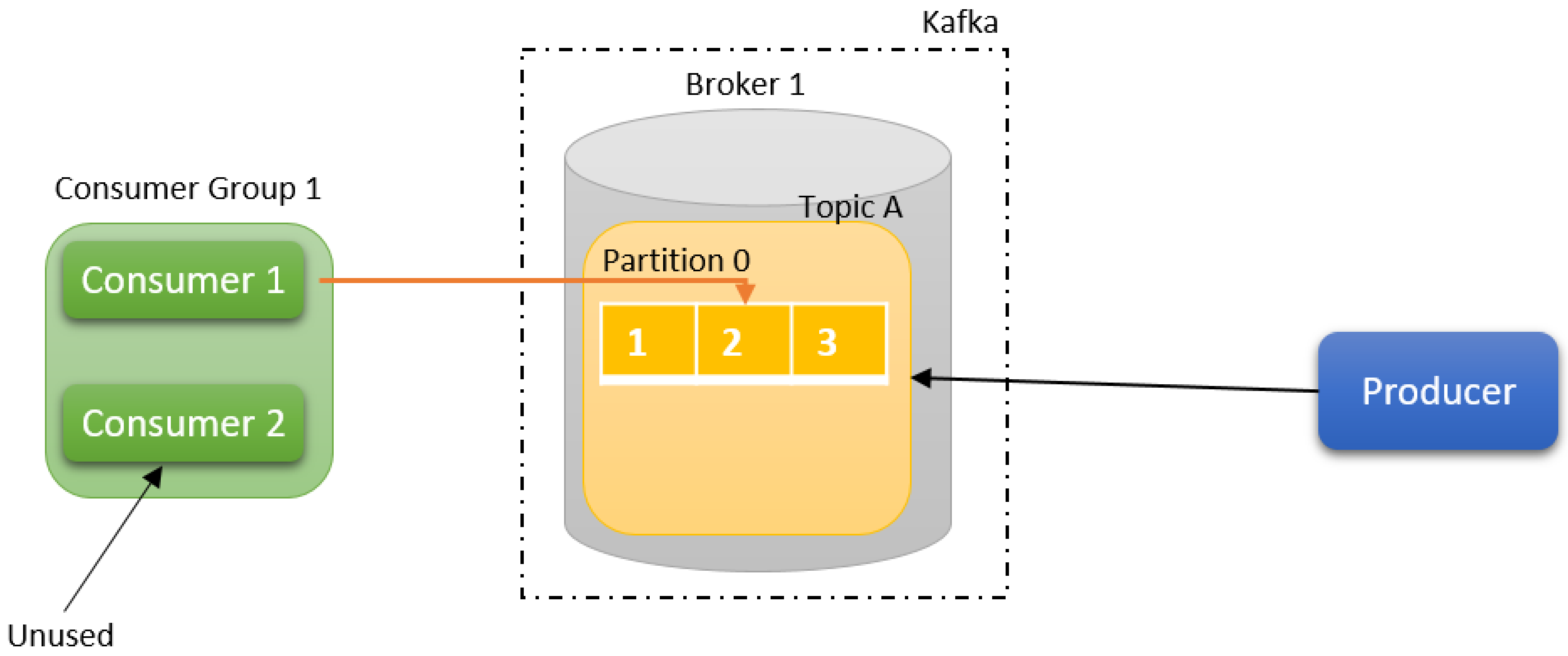

Kafka
Broker 1
Broker 2
Broker 3
Topic A (Partition 0)
Topic A (Partition 1)
Topic A (Partition 2)
Consumer 1
Consumer 2
Consumer Group

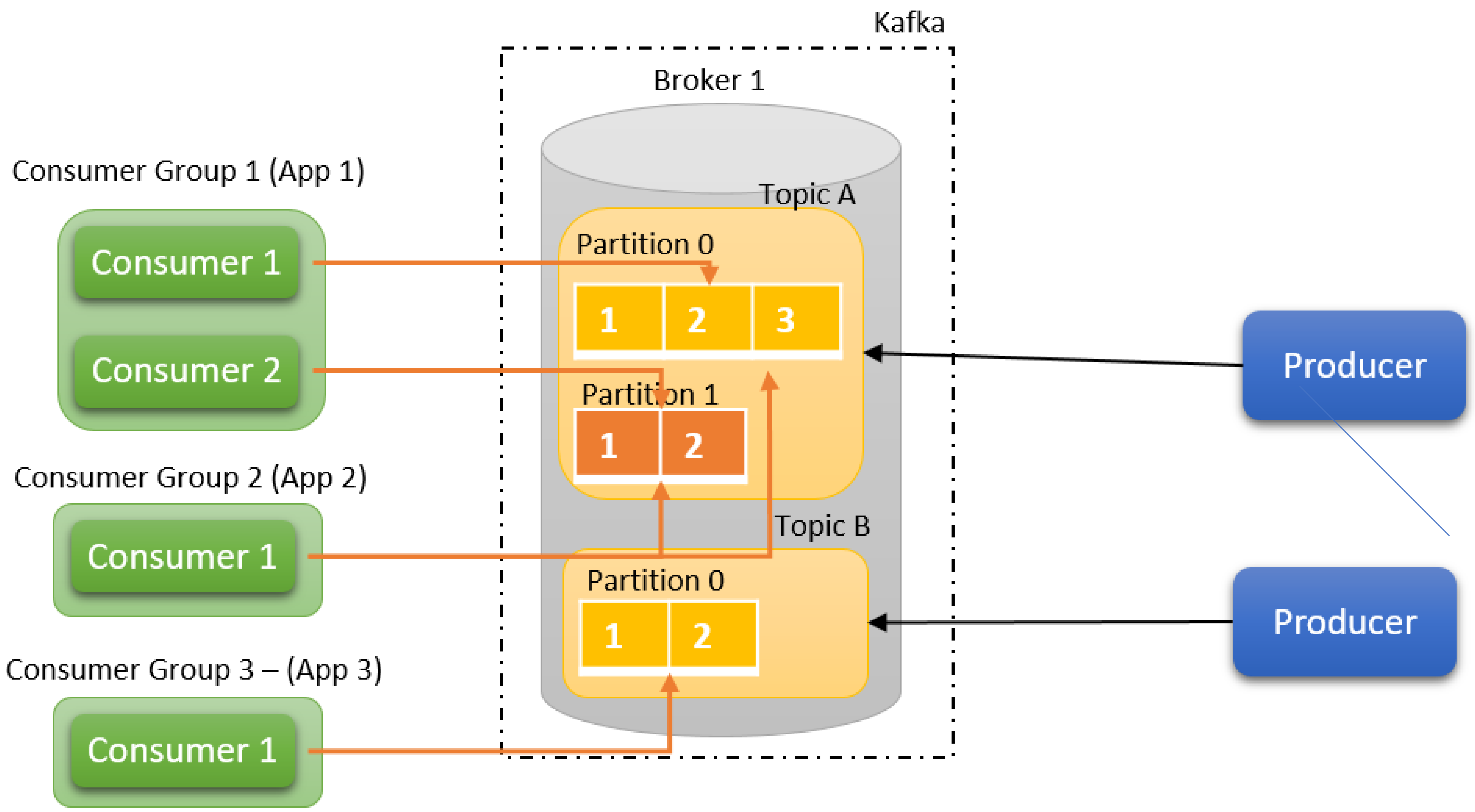
Consumer Group

- In case of multiple consumers in a consumer group, each consumer is assigned with one or more partitions.
- Each consumer in the group will process the records in parallel from each of the leader partition of the brokers.
- A consumer can read from multiple partitions.
- No single partition can be assigned to two consumers in the same consumer group.
- When there are more numbers of consumers in a consumer group is more than the number of partitions in a topic, the consumers will be idle.


Kafka
Topic A
Consumer Group
Consumer 0
Consumer 1
Partition 0
1 2 3 4 5 6
Partition 1
1 2 3 4
Partition 2
1 2 3 4 5
Producer


Kafka
Topic A
Consumer Group
Consumer 0
Consumer 1
Partition 0
1 2 3 4 5 6
Partition 1
1 2 3 4
Partition 2
1 2 3 4 5
Producer


Kafka
Broker 1
Topic A
Partition 0
1
2
3
Consumer Group 1
Consumer 1
Consumer 2
Unused
Producer


Kafka
Broker 1
Topic A
Partition 0
1
2
3
Partition 1
1
2
Topic B
Partition 0
1
2
Consumer Group 1 (App 1)
Consumer 1
Consumer 2
Consumer Group 2 (App 2)
Consumer 1
Consumer Group 3 – (App 3)
Consumer 1
Producer
Producer