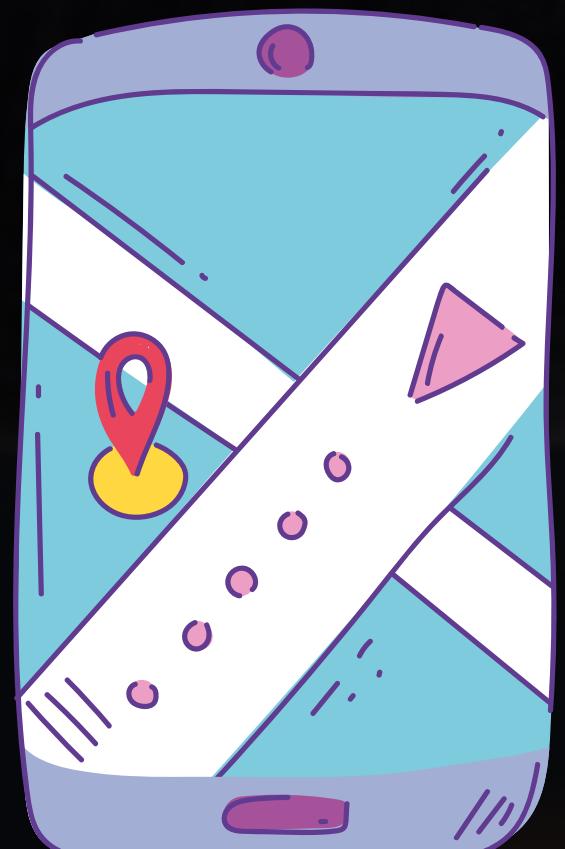


MPRASHANT

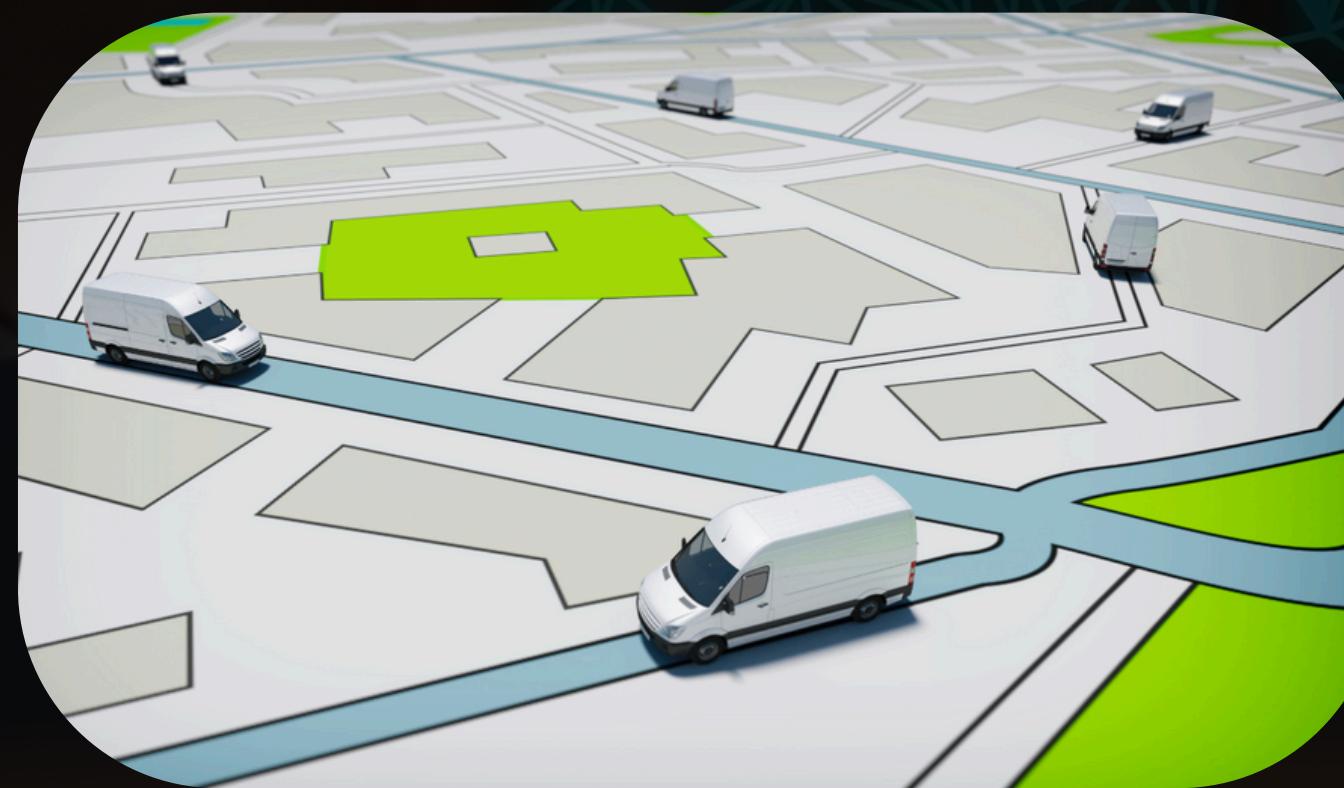


# Importance of real-time data streaming

MPRASHANT



Delivery

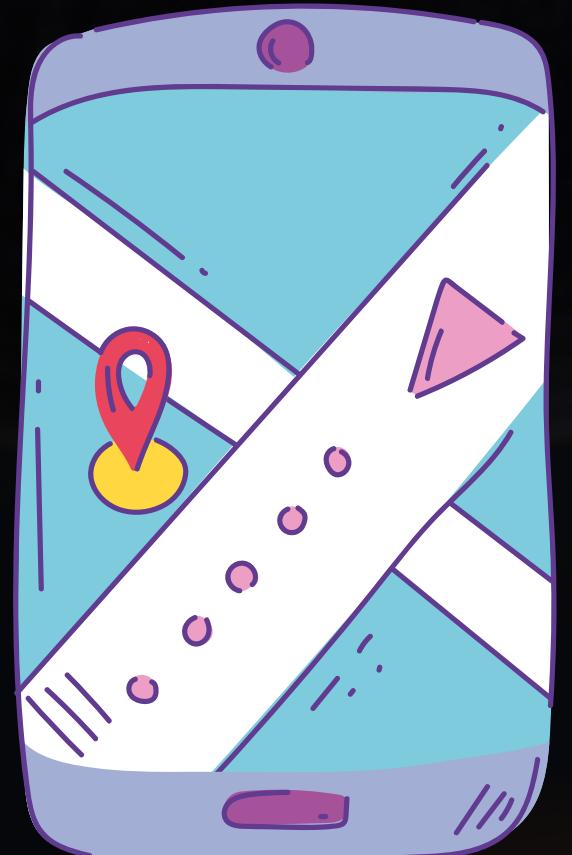


parcel

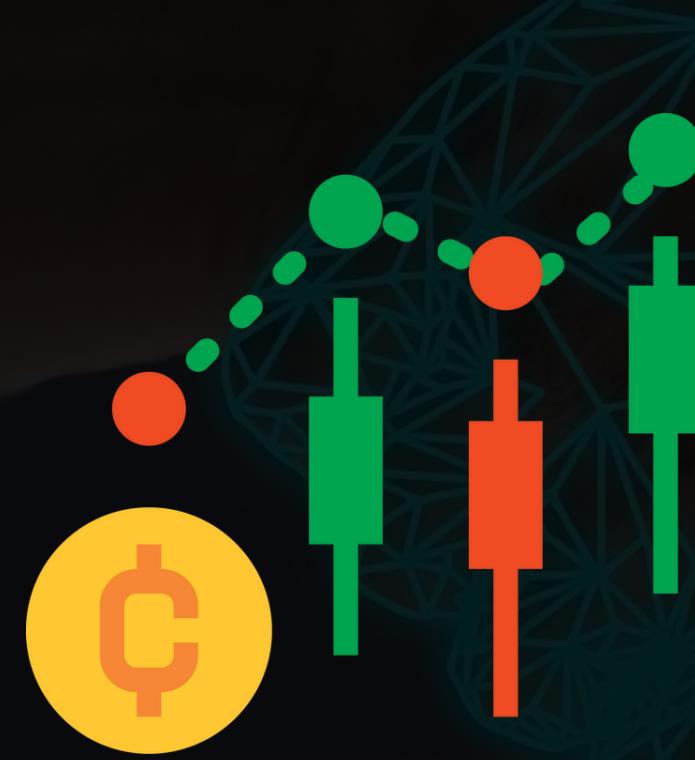
food

car

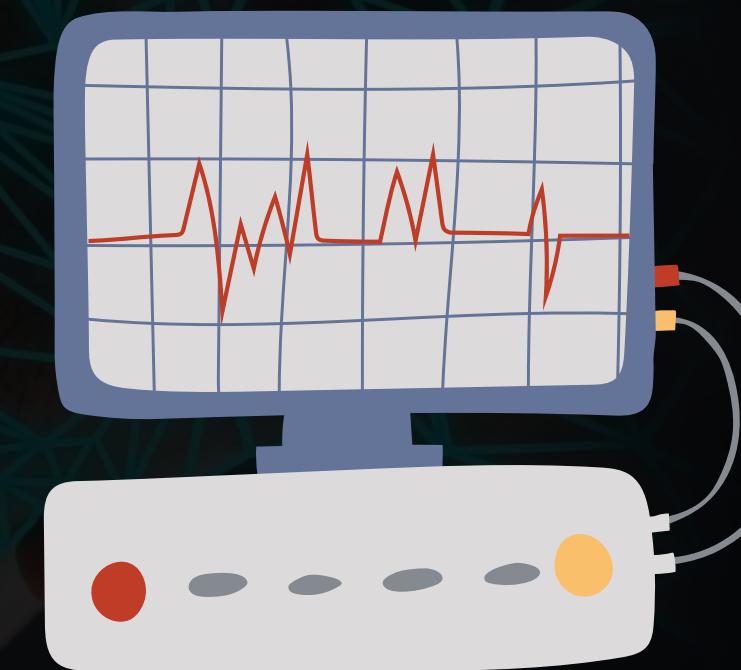
MPRASHANT



Delivery



Stock Price



Health Monitor

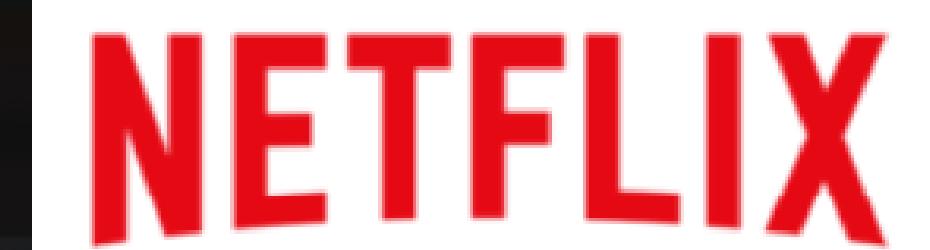
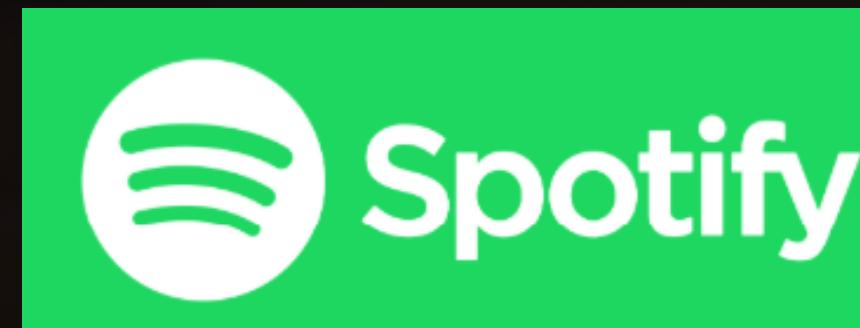
MPRASHANT



# WHAT IS APACHE KAFKA?

# Kafka is a **distributed streaming platform**.

Allows applications to send, store, and process data in real time.



# APACHE KAFKA

*More than 80% of all Fortune 100 companies trust, and use 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.

---



---

**10 OUT OF 10**

---

*MANUFACTURING*



---

**7 OUT OF 10**

---

*BANKS*



---

**10 OUT OF 10**

---

*INSURANCE*



---

**8 OUT OF 10**

---

*TELECOM*

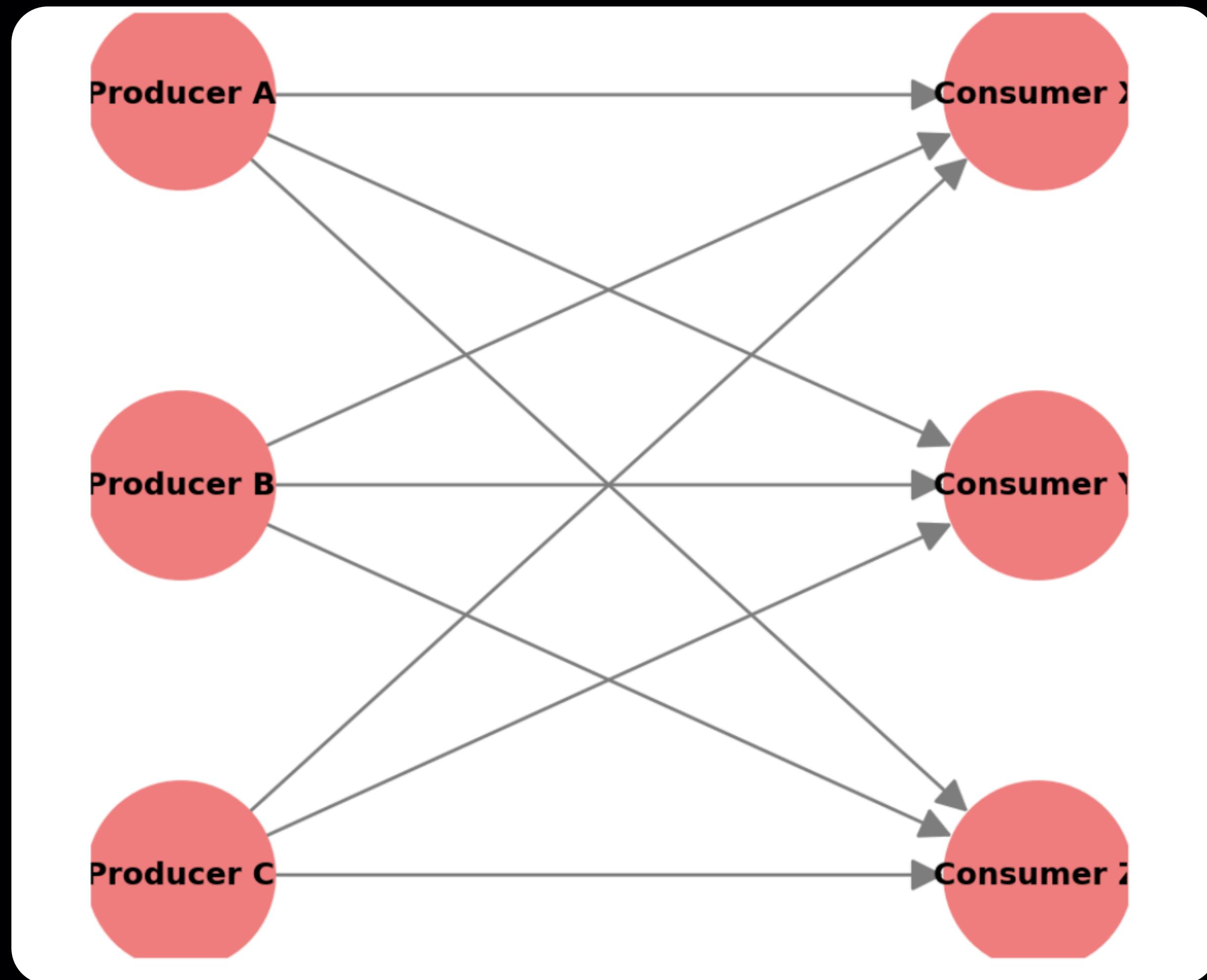
MPRASHANT

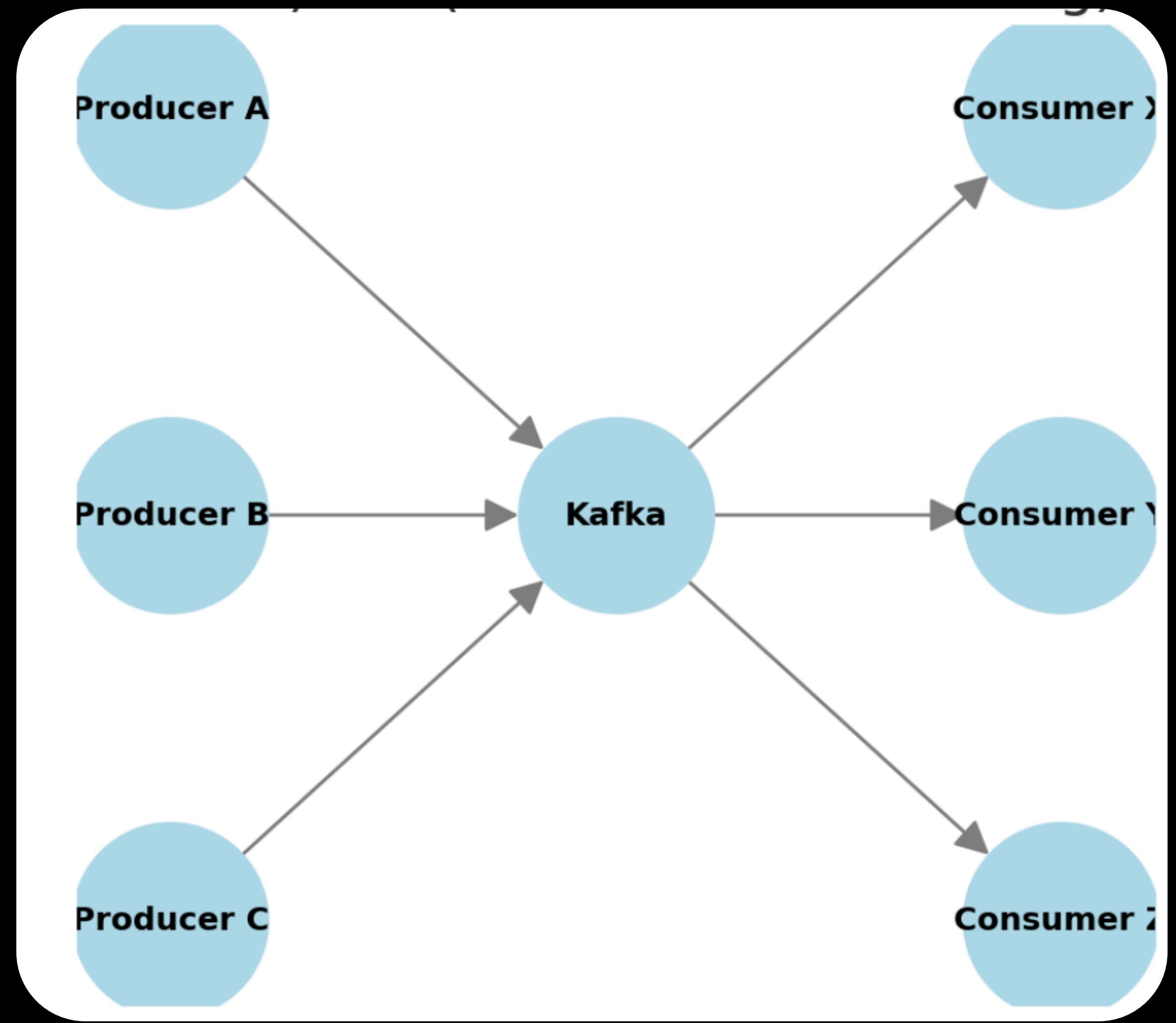


**DEVELOPED BY LINKEDIN AND OPEN-SOURCED UNDER  
THE APACHE SOFTWARE FOUNDATION.**



# Why was Kafka created?



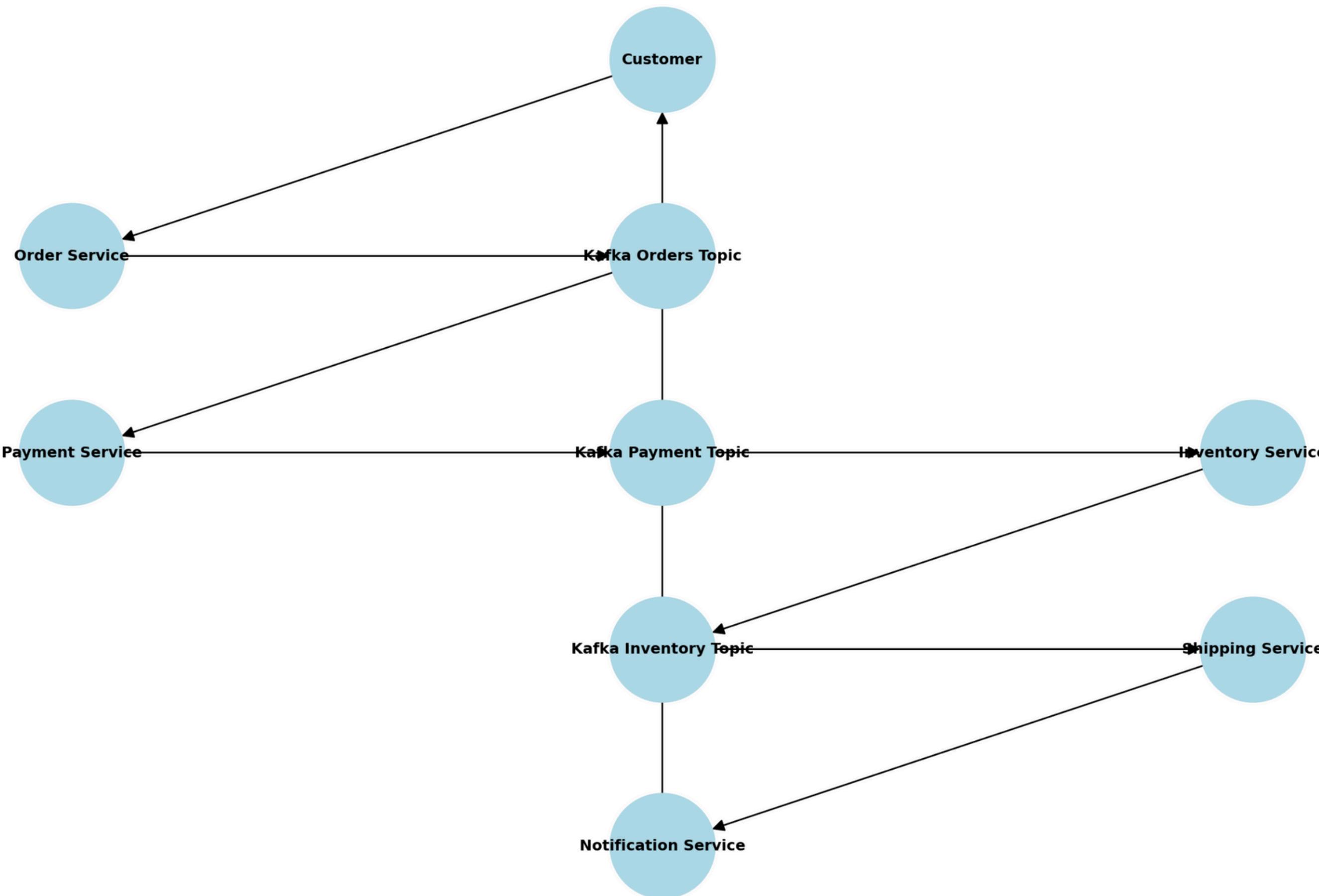


# **EDA (Event Driven Architecture)**

**EDA is a system design where services communicate through events instead of direct calls.**

**In EDA, one service publishes an event (e.g., "Order Created"), and other services react to it asynchronously (e.g., process payment, update inventory, ship the product).**

## Kafka Event-Driven Architecture (EDA) for Order Processing

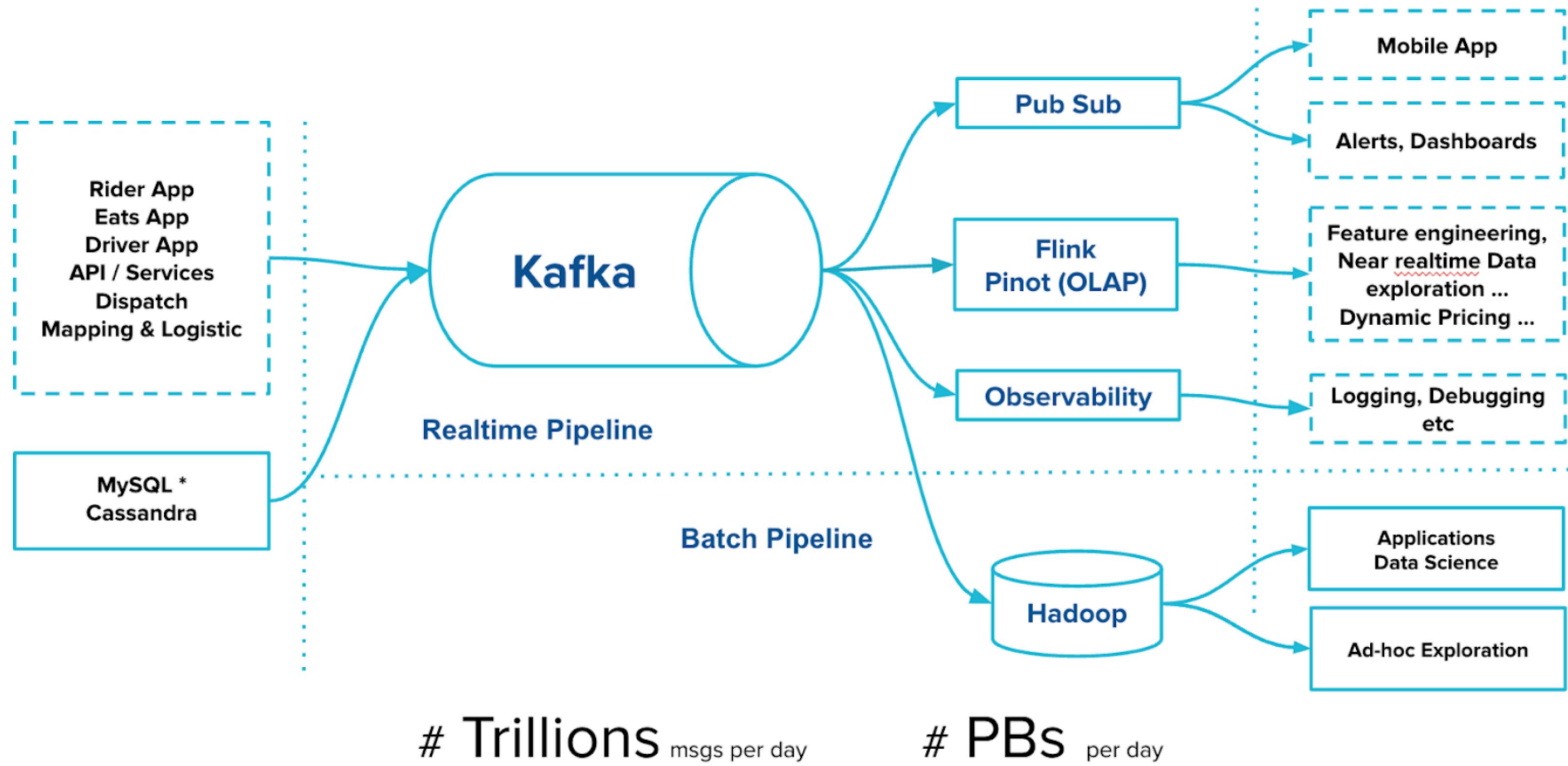




**Powered By**

Apache Kafka

 apache.org



# BENEFITS OF USING KAFKA

- **SCALABILITY:** HANDLE LARGE AMOUNTS OF DATA.
- **FAULT TOLERANCE:** REPLICATION OF DATA.
- **HIGH THROUGHPUT:** CAPABLE OF PROCESSING MILLIONS OF MESSAGES PER SECOND.
- **DURABILITY:** RELIABLE STORAGE OF MESSAGES.

# KEY COMPONENTS OF KAFKA

- TOPICS
- PRODUCERS
- CONSUMERS
- BROKERS
- PARTITIONS
- ZOOKEEPER / KRAFT

# Kafka Broker

When you run Kafka, you start one or more “broker” processes.  
It is basically the server (or node) in a Kafka cluster.

- Stores messages
- Handle Read/write requests

MPRASHANT



# Kafka Topic

A Kafka topic is like a “channel” or “category” where messages (data) are published. Think of it as a folder that holds all the messages belonging to a specific stream of data.

- When you send a message to Kafka, you choose which topic to send it to.
- Consumers then subscribe to that topic to receive the messages.

MPRASHANT

**Broker**

**Topic A**

**Topic B**

## Producer:

- An application or process that sends events to Kafka topics. Producers write data to Kafka.

## Consumer:

- An application or process that reads events from Kafka topics.
- Consumers subscribe to topics and process the data.

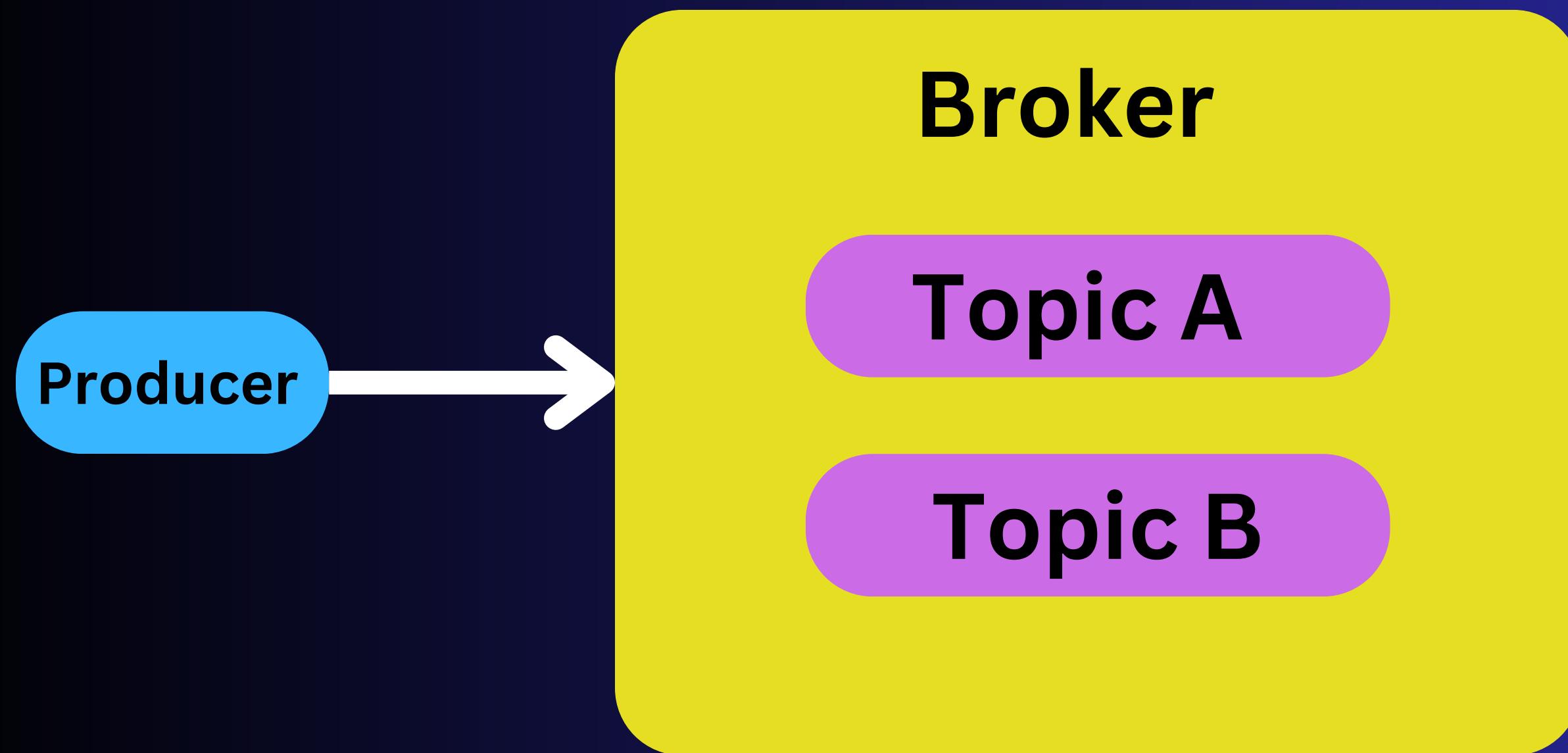
MPRASHANT

**Broker**

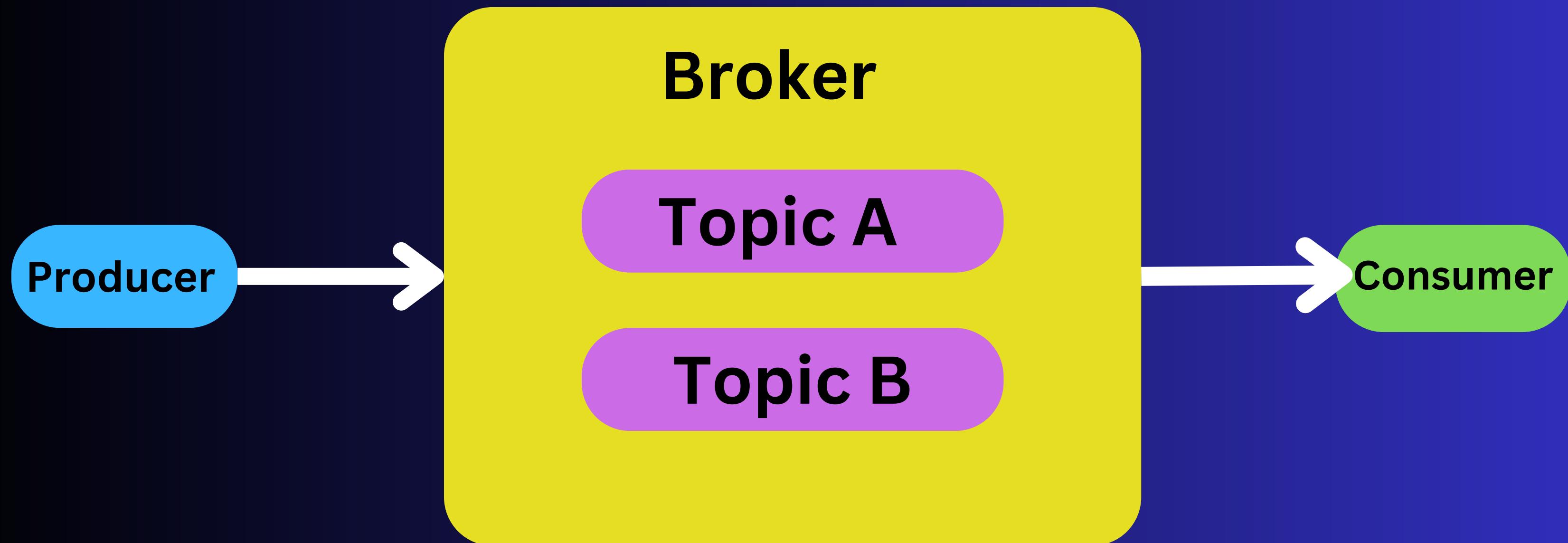
**Topic A**

**Topic B**

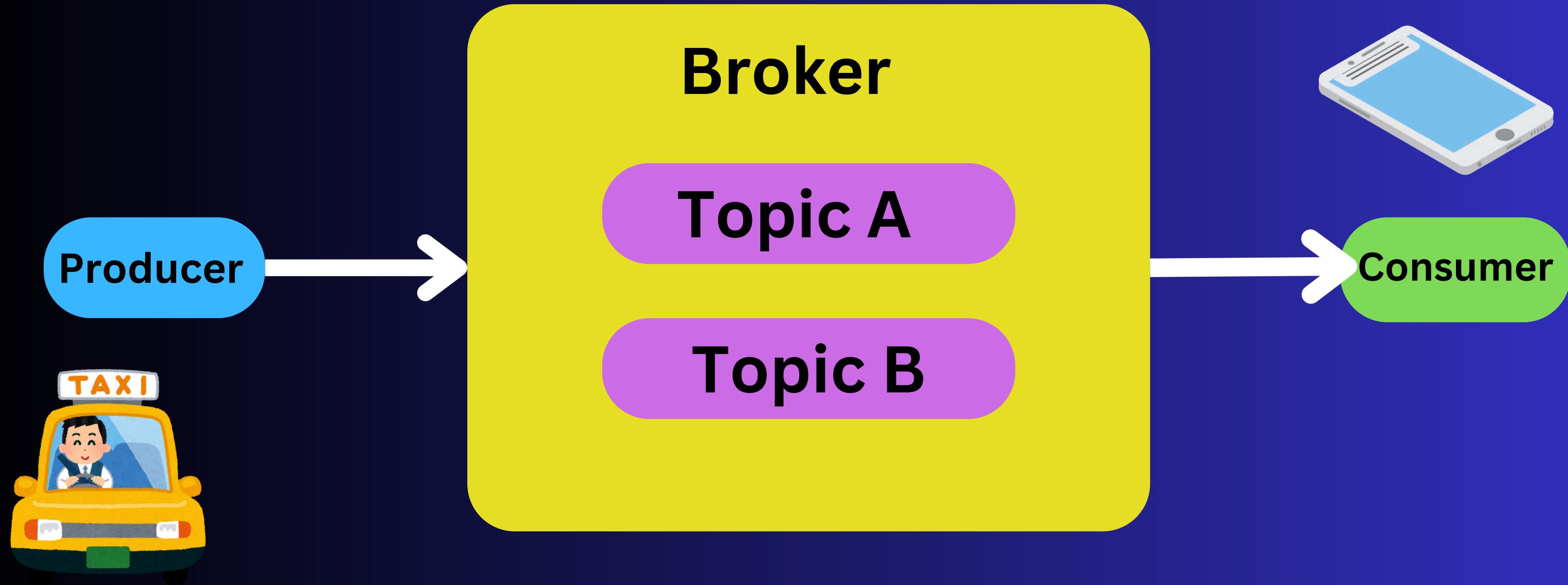
MPRASHANT



MPRASHANT



MPRASHANT



# **Some systems that can act as producers for Kafka:**

**SQL or NoSQL Databases**

**REST APIs**

**Log Aggregators**

**Message Queues**

**Stream Processing Frameworks**

**IoT Devices and Sensors**

**Data Integration Tools**

**Custom Applications**

**Data Analytics and BI Tools**



## Kafka Producers Can Be Built Using:

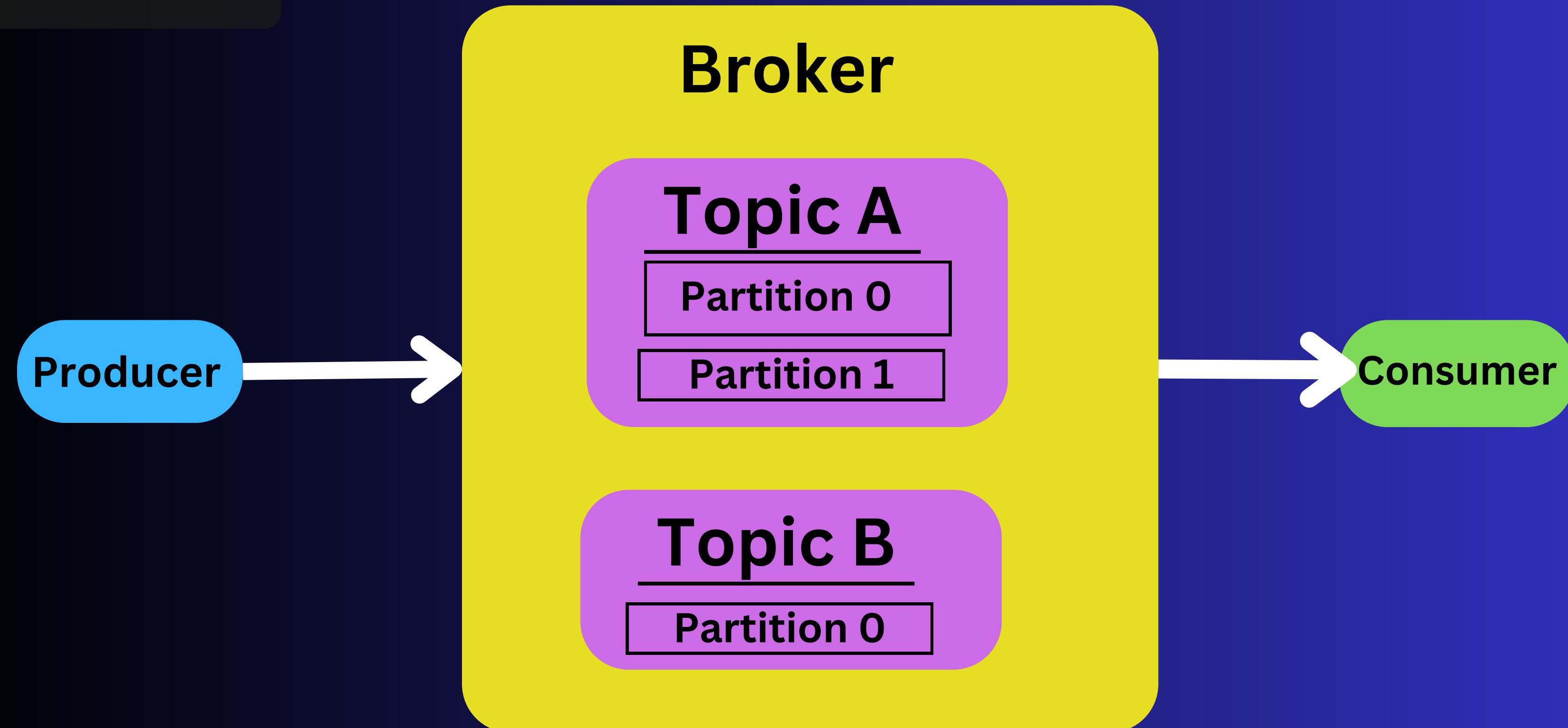
- ✓ Kafka Connect (JDBC, Debezium, HTTP Source)
- ✓ Custom Producers (Java, Python, Go, Node.js)
- ✓ Open-source Agents (Fluentd, Logstash, Filebeat)

# Kafka Partition

A partition is a subsection of a Kafka topic. Topics can have multiple partitions so data can be spread out across brokers.

Each partition maintains a sequential order of messages, allowing parallel read and write for higher performance.

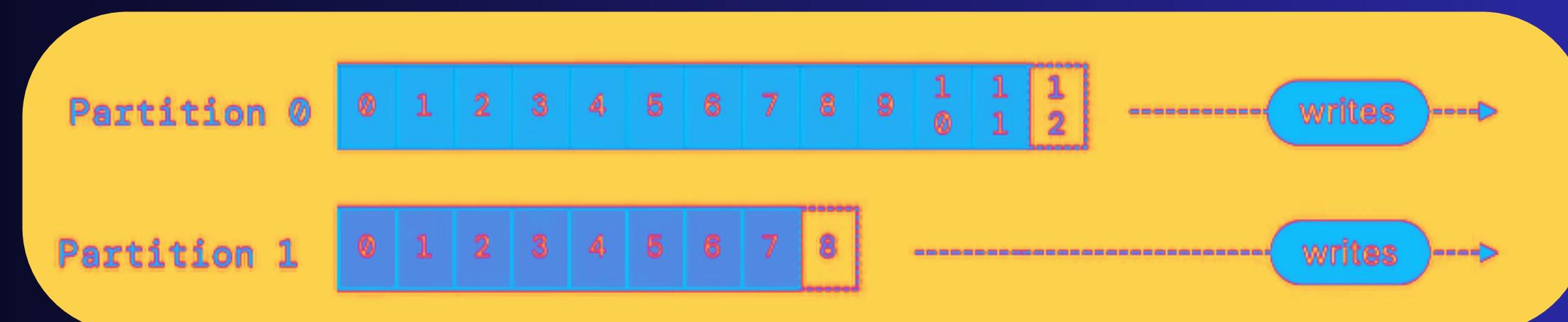
MPRASHANT



# Kafka Offset

A Kafka offset is a sequential number assigned to each message within a partition. Think of it like page numbers in a book.

Offsets are unique per partition, helps in tracking.



# Topic A

## Partition 0

0	1	2	3	4	5	6	...
---	---	---	---	---	---	---	-----

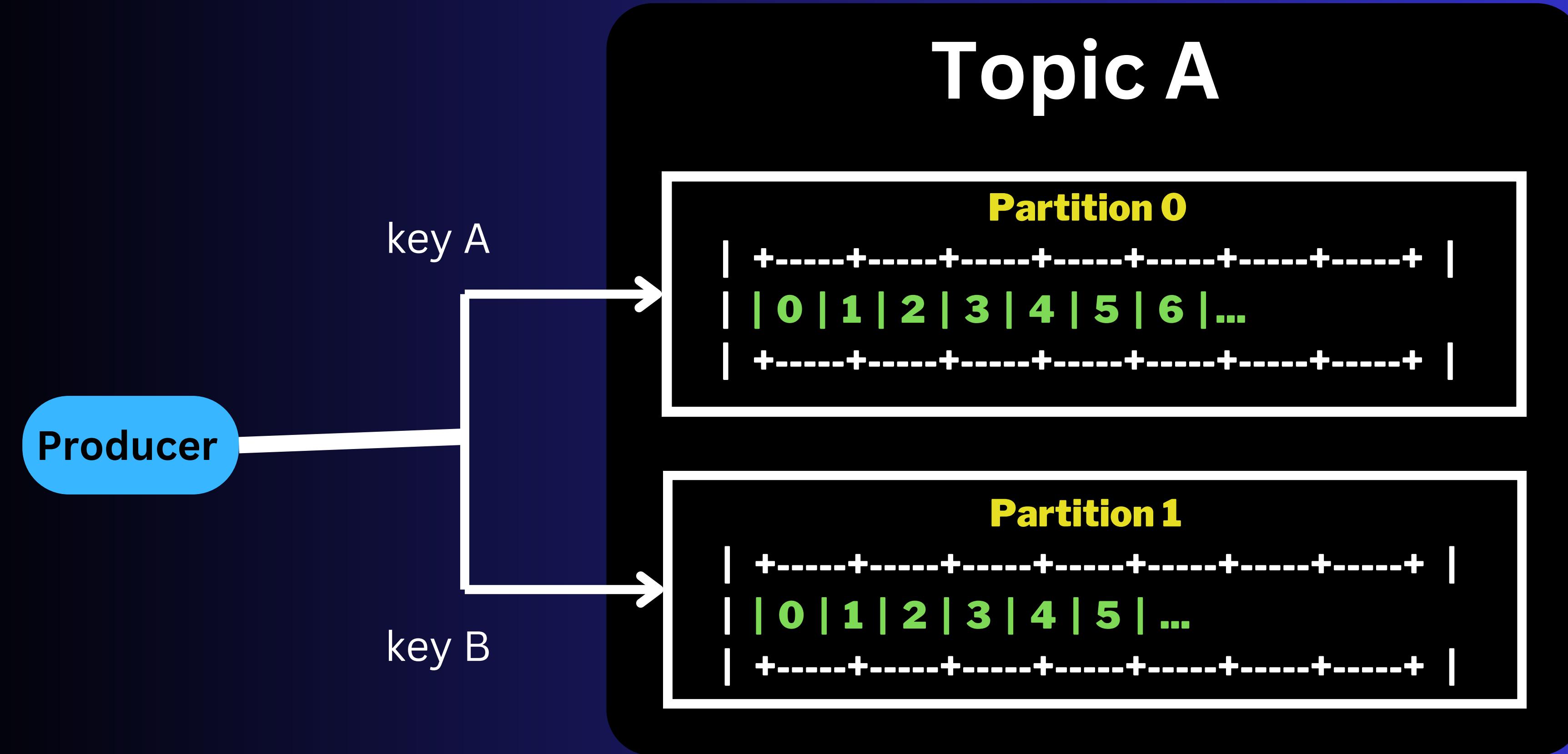
## Partition 1

0	1	2	3	4	5	...
---	---	---	---	---	---	-----

- **Assigned by Kafka** – The Kafka broker assigns offsets, not the producer.
- **Unique per Partition** – Each partition has its own independent offsets (e.g., Partition 0: 0,1,2..., Partition 1: 0,1,2...).
- **Maintains Order (Per Partition)** – Messages are always read in order within a partition.
- **Tracks Consumer Progress** – Consumers store their last read offset to resume processing.
- **Supports Message Replay** – Consumers can rewind or seek to reprocess older messages.
- **Stored in Kafka (\_consumer\_offsets)** – Kafka tracks offsets internally, unless configured otherwise.
- **Retention Affects Offsets** – If a message exceeds Kafka's retention period, it may no longer exist.

# Message Key

- Used for Partitioning – Kafka uses the message key to decide which partition the message should go to.
- Ensures Ordering – Messages with the same key always go to the same partition, preserving order.
- Optional – Producers can send messages with or without a key (if no key, Kafka distributes messages round-robin).
- Stored with Message – The key is stored along with the message in Kafka but does not affect the message content.
- Can Be Any Data Type – Typically a string, but can be any serializable data.

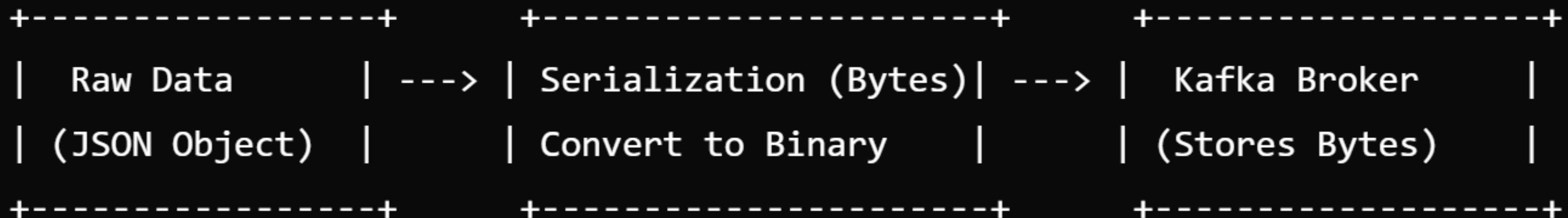


```
+-----+  
|           Kafka Message (Record)           |  
+-----+  
| Topic: "orders"                          |  
| Partition: (Decided by Key or Round-robin) |  
| Key: "order-12345" (Ensures Ordering per Key) |  
| Value: {  
|   "order_id": "12345",  
|   "customer": "John Doe",  
|   "amount": 250.75,  
|   "status": "pending"  
| }                                         |  
| Headers: { "source": "web-app", "timestamp": "1700000000000" } |  
| Timestamp: "1700000000000"                  |  
+-----+
```

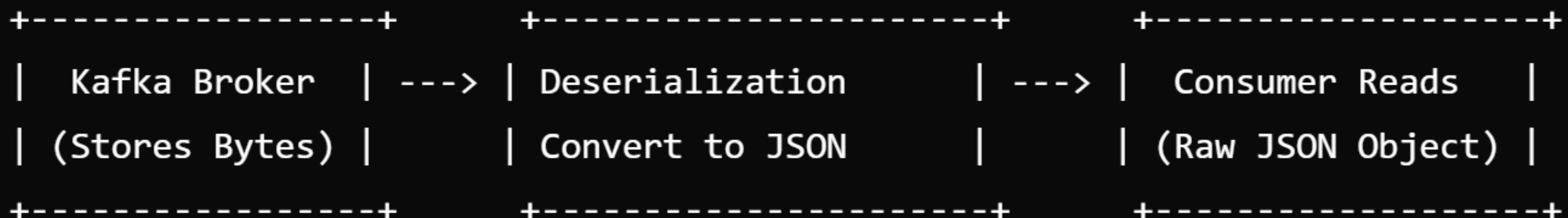
# **Kafka Serialization - Key Points & Flow**

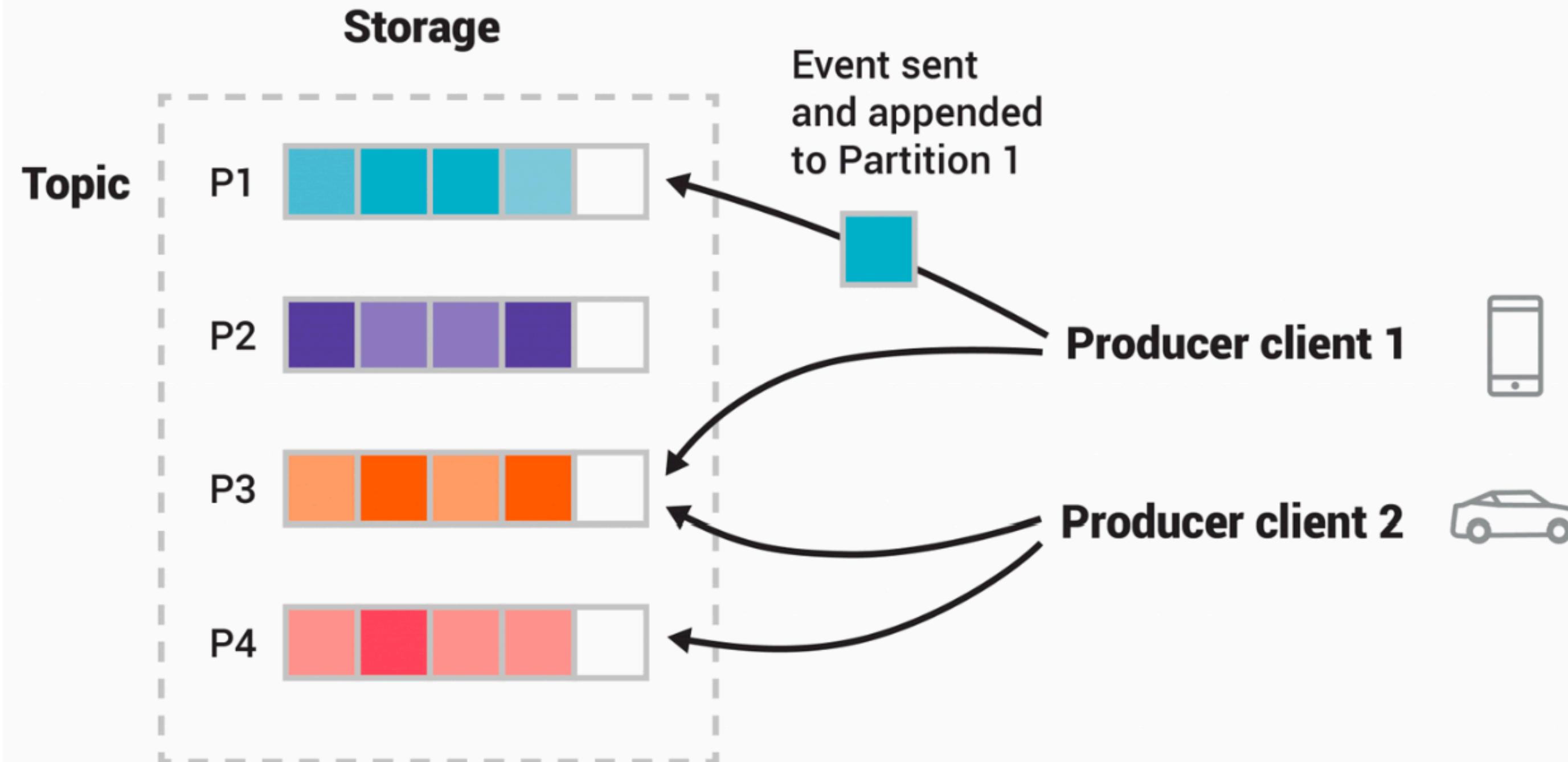
**Serialization in Kafka is the process of converting data into bytes before sending it to Kafka (Producer Side) and deserializing it back when reading from Kafka (Consumer Side).**

## 1 Producer Side: Serialization



## 2 Consumer Side: Deserialization



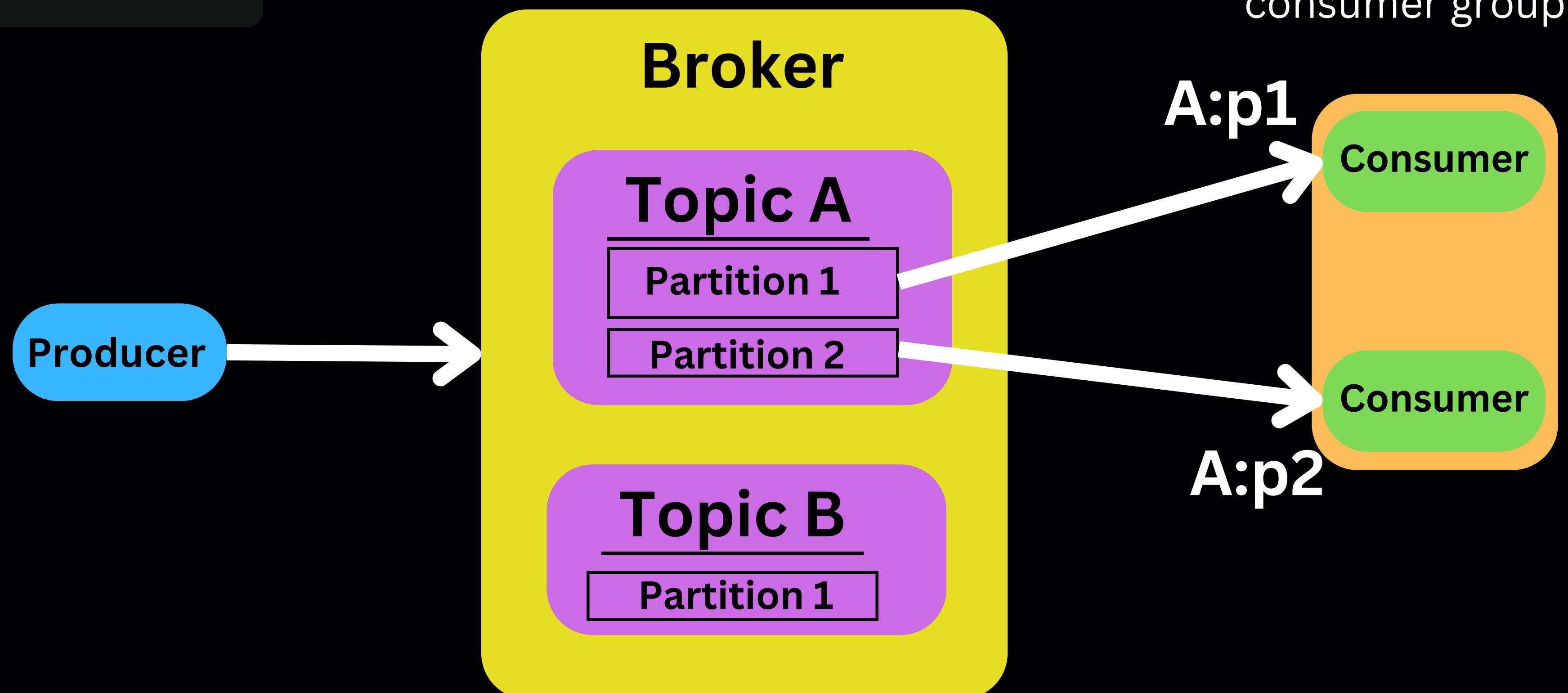


# What is a Consumer Group in Kafka?

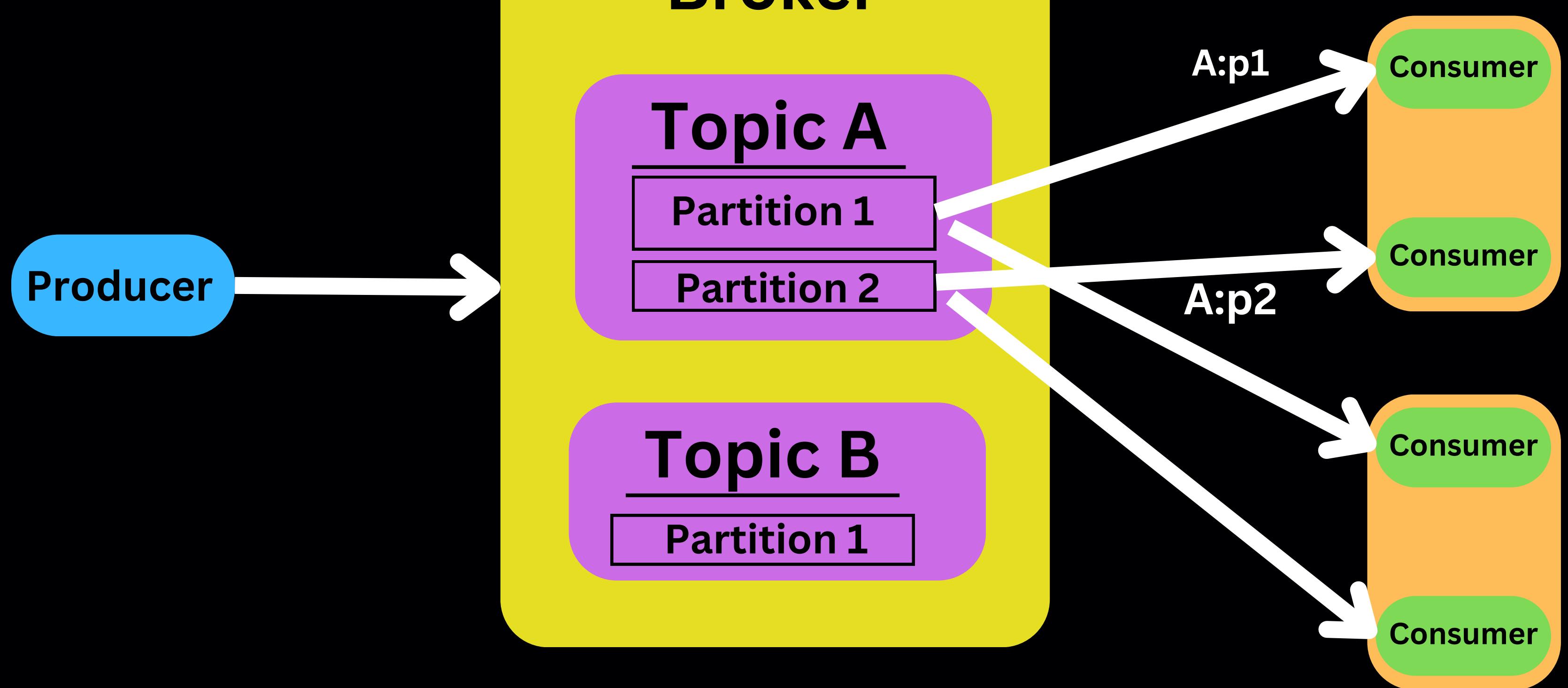
A Consumer Group in Kafka is a logical group of consumers that work together to consume messages from a Kafka topic in parallel.



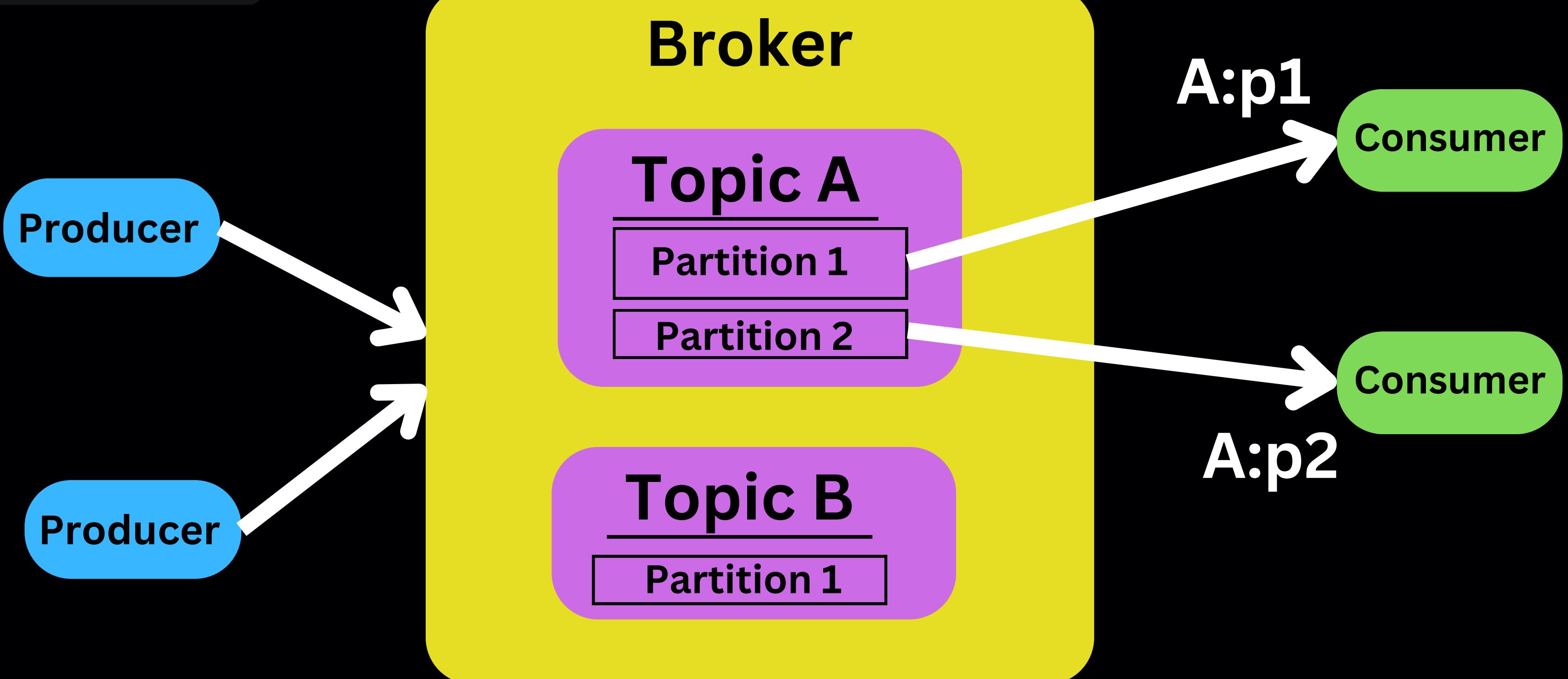
MPRASHANT



MPRASHANT

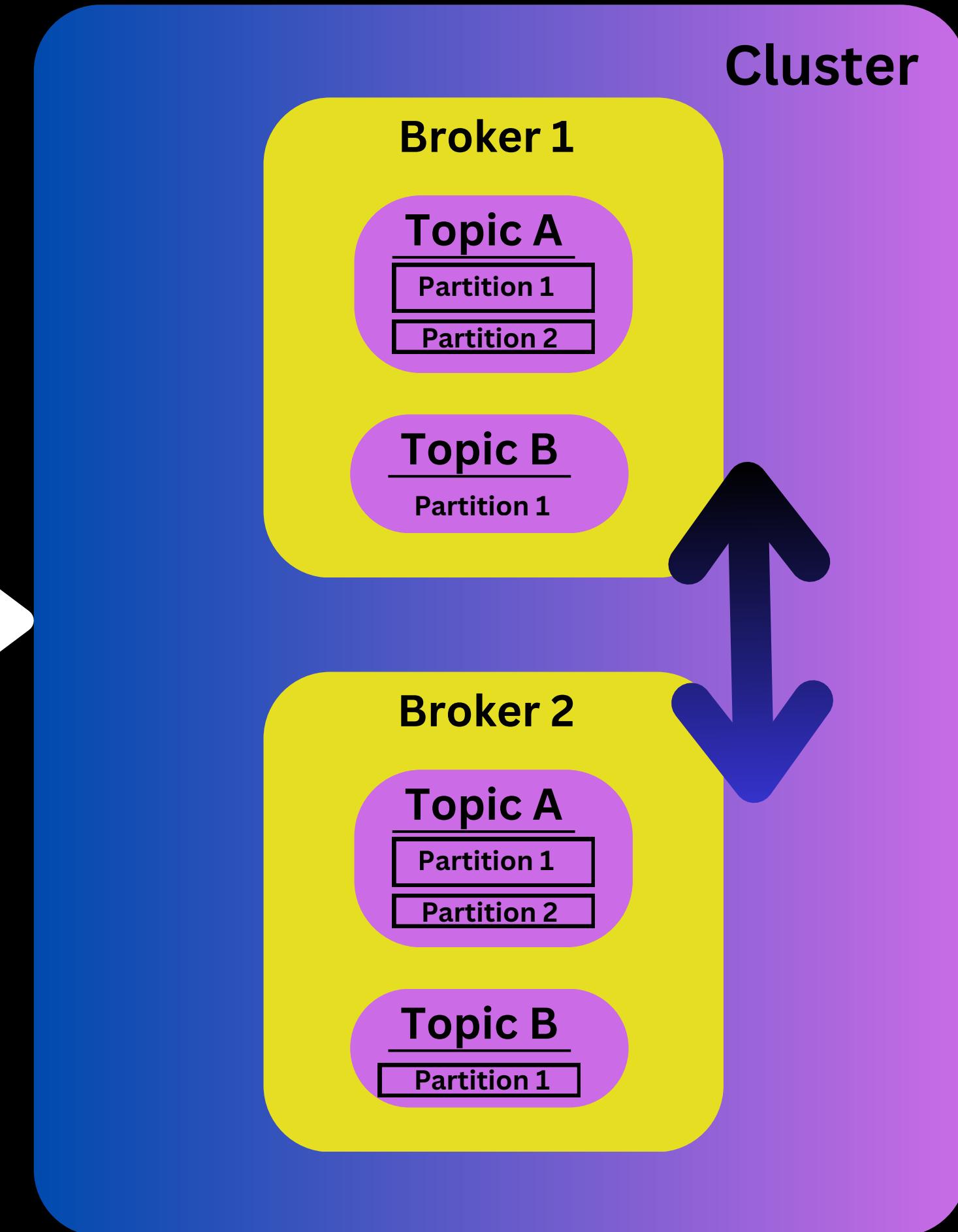


MPRASHANT



MPRASHANT

Producer



Consumer

# 1 Broker, 3 Partitions, Replication Factor 1 (No Replication)

```
pgsql
```

Kafka Cluster: 1 Broker (No Replication)

Broker 1	
P0	(Leader)
P1	(Leader)
P2	(Leader)

Here is the Kafka cluster representation for 2 brokers, 2 partitions, replication factor 2:

pgsql

Copy

Kafka Cluster: 2 Brokers, 2 Partitions, Replication Factor 2

Broker 1		Broker 2	
P0 (Leader)		P1 (Leader)	
P1 (Replica)		P0 (Replica)	

# 3 Brokers, 3 Partitions, Replication Factor 3 (Highly Available)

pgsql

Kafka Cluster: 3 Brokers (Highly Available)

Broker 1		Broker 2		Broker 3	
P0 (Leader)		P1 (Leader)		P2 (Leader)	
P1 (Replica)		P2 (Replica)		P0 (Replica)	
P2 (Replica)		P0 (Replica)		P1 (Replica)	

👉 Each partition is replicated across all brokers for maximum availability.

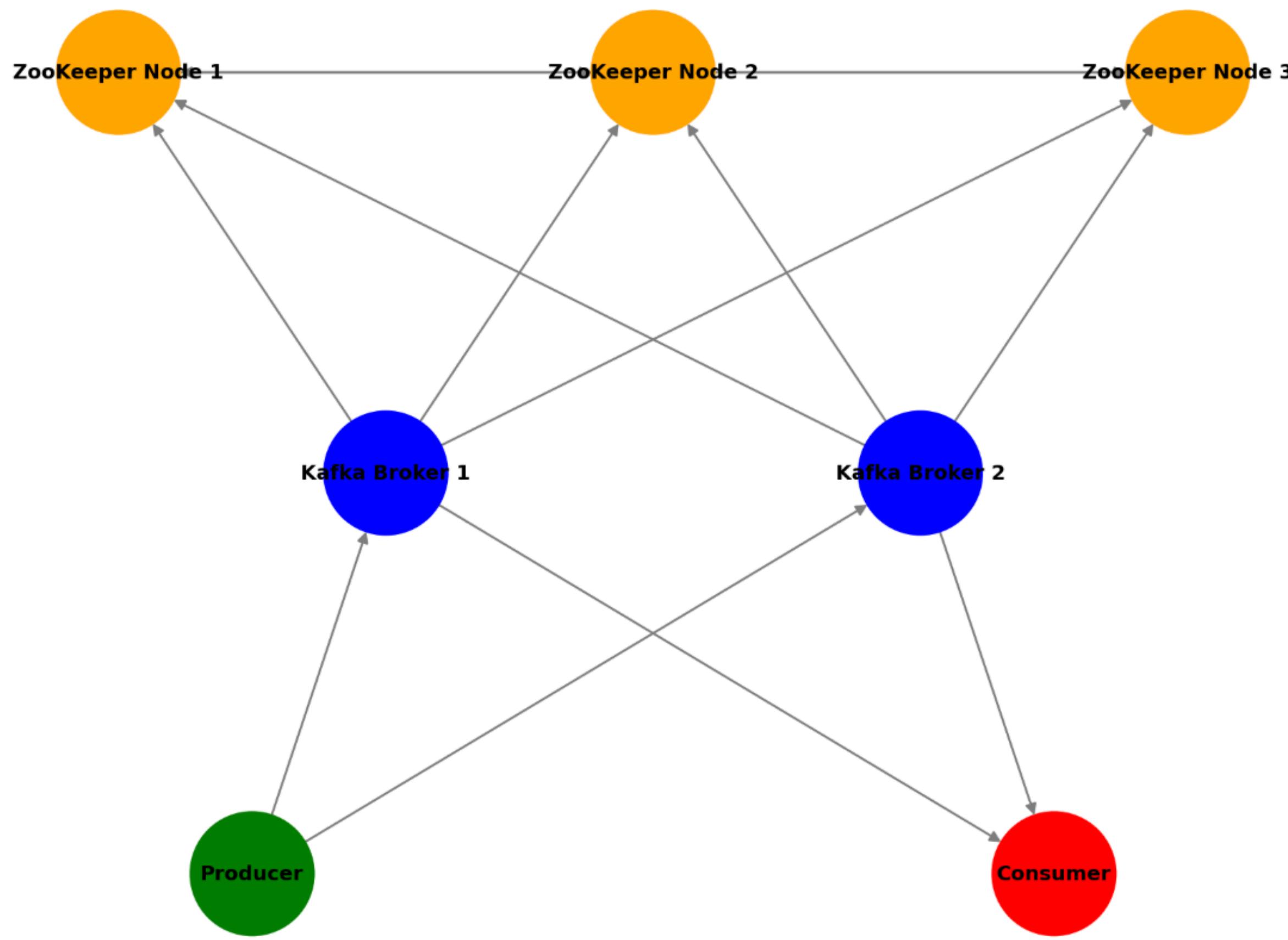
# What is Zookeeper?

ZooKeeper is a distributed coordination service that manages synchronization, configuration, and leader election in distributed systems.

It basically ensures everything stays in sync and knows what's happening where.

- ✓ ZooKeeper ensures high availability by electing partition leaders.
- ✓ Monitors broker health and reassigns leadership on failures.
- ✓ Stores cluster metadata, including topics, partitions, and replicas.
- ✓ Coordinates consumer groups for message ordering.

## Kafka Cluster with ZooKeeper (2 Brokers)



# What is Kraft?

Kraft (Kafka Raft) is Kafka's new way of handling coordination and metadata without needing ZooKeeper.

It uses a built-in Raft consensus protocol so that Kafka brokers manage everything themselves.

## Zookeeper vs KRaft (Kafka Raft Mode) - Key Differences

Feature	Zookeeper 	KRaft (Kafka Raft Mode) 
Architecture	Separate service outside Kafka	Built-in to Kafka (No external dependency)
Leader Election	Managed by Zookeeper	Managed by Kafka using <b>Raft consensus</b>
Fault Tolerance	Requires multiple Zookeeper nodes for HA	Kafka controllers handle failures natively
Performance	Extra network calls to Zookeeper	Faster (direct in-cluster coordination)
Scalability	Limited scalability due to Zookeeper overhead	Better scalability, more partitions per cluster
Maintenance	Requires separate Zookeeper cluster management	Simplifies deployment (Zookeeper-free Kafka)

 KRaft is the future of Kafka, replacing Zookeeper for better performance, simplicity, and scalability!

# Practical

- Java
- Kafka
- Start Kafka with Zookeeper
- Start Kafka with KRaft
- More examples

# Installation

- Kafka requires Java 11 or higher.
  - Can install openjdk from
    - <https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/downloads-list.html>
  - Configure JAVA\_HOME (if not work, try removing space in the path)
  - Java is using in **kafka-run-class** file so adjust path according to it or in config file.

# Kafka Installation

- <https://kafka.apache.org/downloads>
  - Download and unzip on your system in any location.
  - Windows
    - Download Scala version
  - MAC / Linux
    - `wget <download_link>`

# Start Kafka with Zookeeper

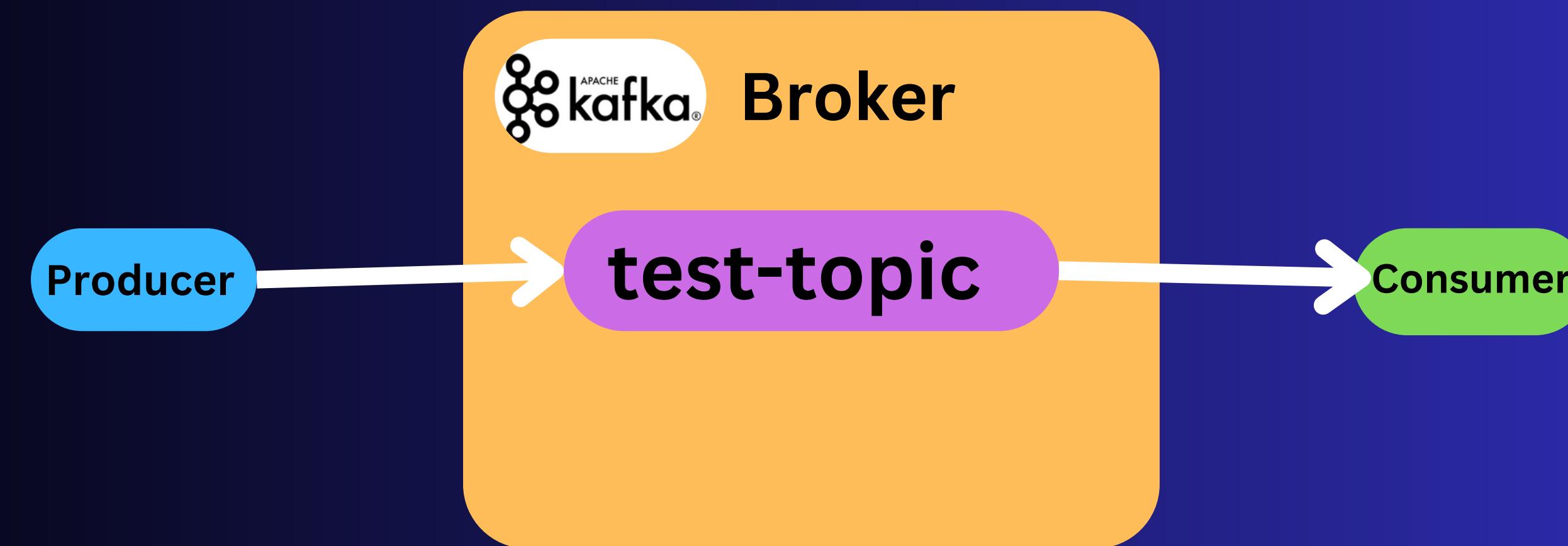
- In the installation under bin folder there is a windows folder too for batch script.
- Start zookeeper or kraft
  - *bin/zookeeper-server-start.sh config/zookeeper.properties*
- Start kafka broker
  - *bin/kafka-server-start.sh config/server.properties*

# Start Kafka with KRaft

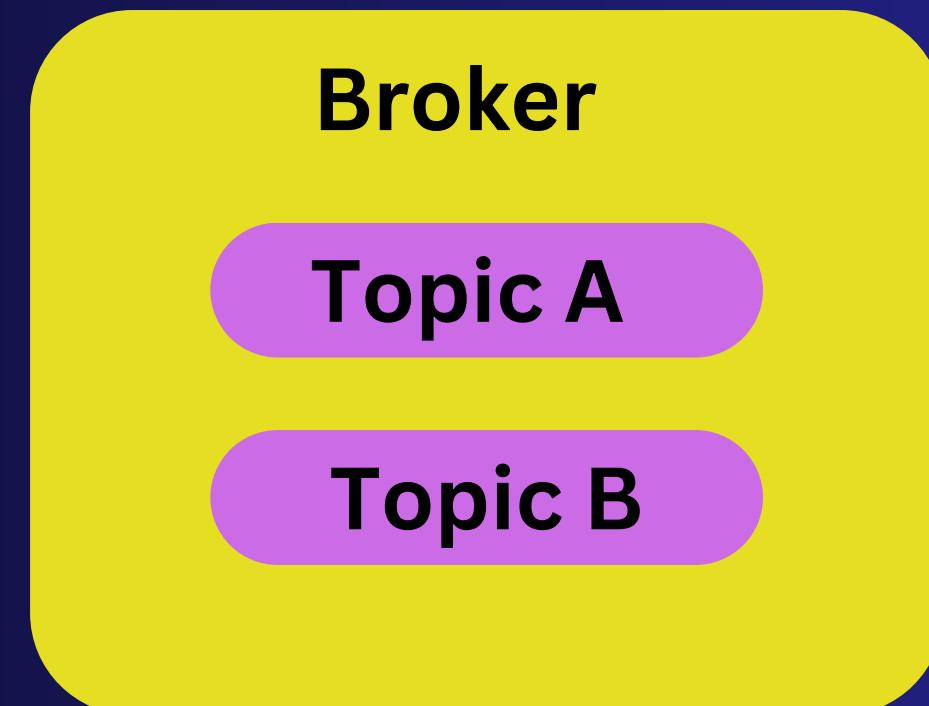
- **Generate a unique ID**
  - `bin/kafka-storage.sh random-uuid`
- **Format the storage directory with the generated cluster ID**
  - `bin/kafka-storage.sh format -t UUID -c config/kraft/server.properties`
- **Start Kafka server**
  - `bin/kafka-server-start.sh config/kraft/server.properties`
  -

- To delete all Kafka data manually.
  - `rm -rf /tmp/kafka-logs`
  - `rmdir /s /q C:\tmp\kafka-logs`
- When you want to reinitialize Kafka in KRaft mode.
  - `kafka-storage.sh` format

# Example 1



# Creating Topic



# Create a Topic

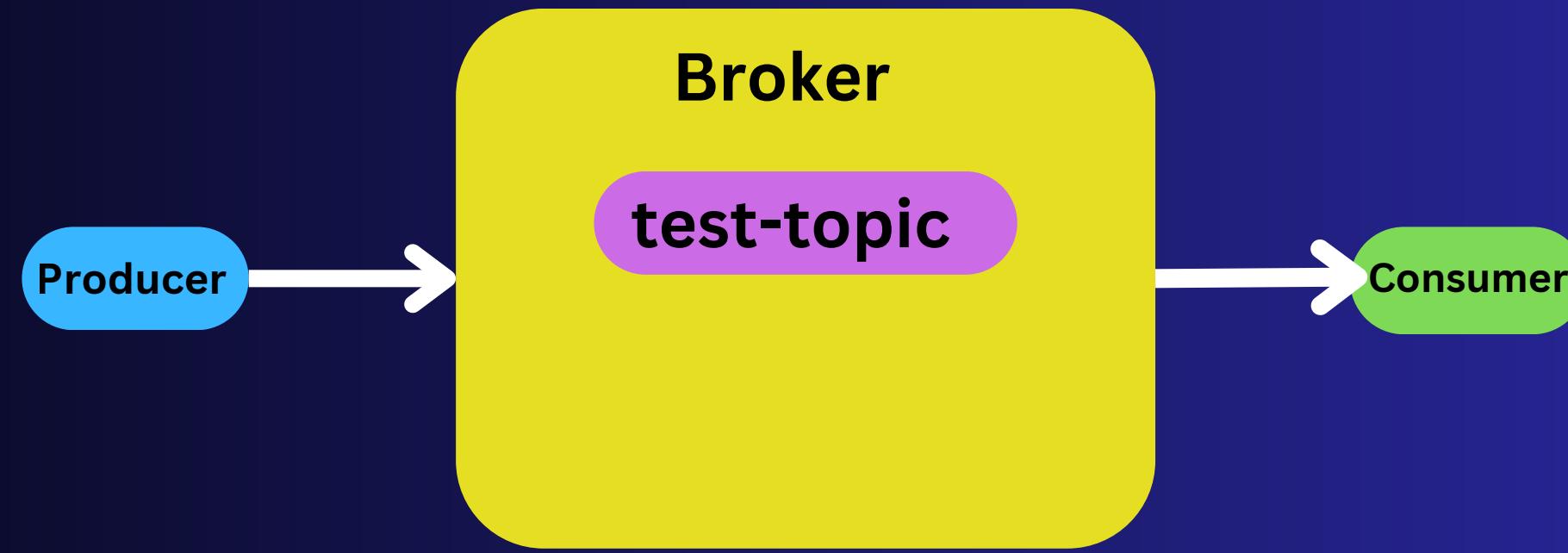
```
bin/kafka-topics.sh --create \  
--topic test-topic \  
--bootstrap-server localhost:9092 \  
--partitions 1 \  
--replication-factor 1
```

## Testing created topics

- `kafka-topics.sh --bootstrap-server localhost:9092 --list`
- `kafka-topics.sh --bootstrap-server localhost:9092 --topic chat --describe`

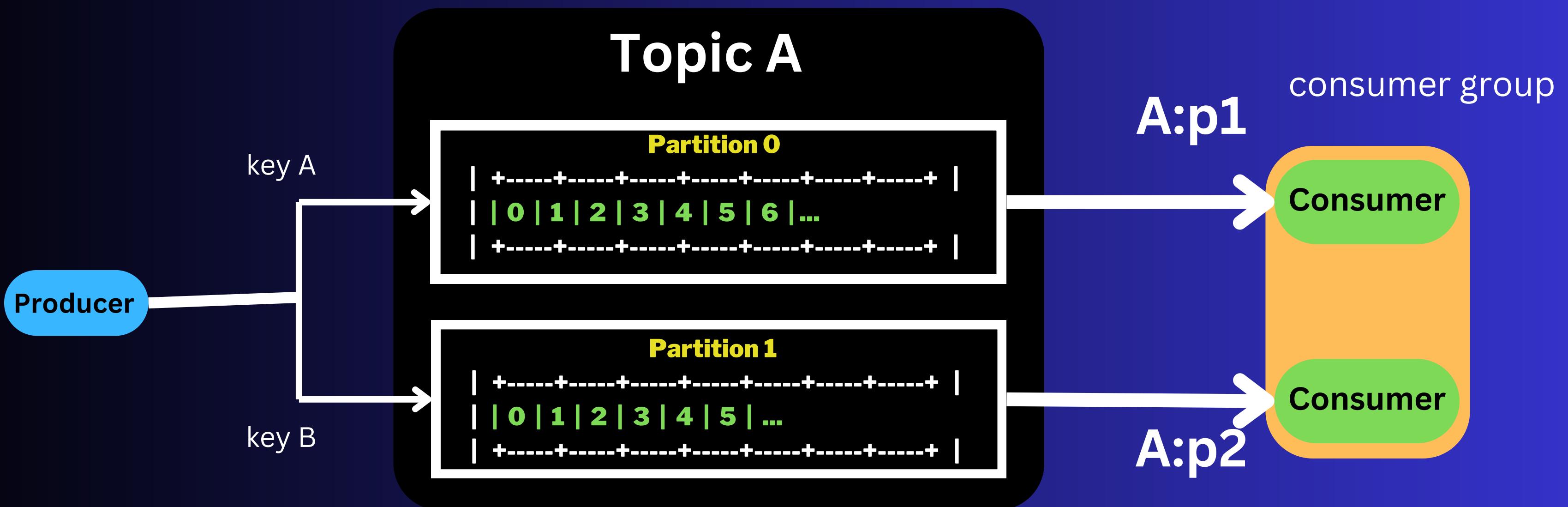
## Test Produce/Consume:

- `kafka-console-producer.sh --bootstrap-server localhost:9092  
--topic test-topic`
- `kafka-console-consumer.sh --bootstrap-server localhost:9092  
--topic test-topic --from-beginning`



# Example 2

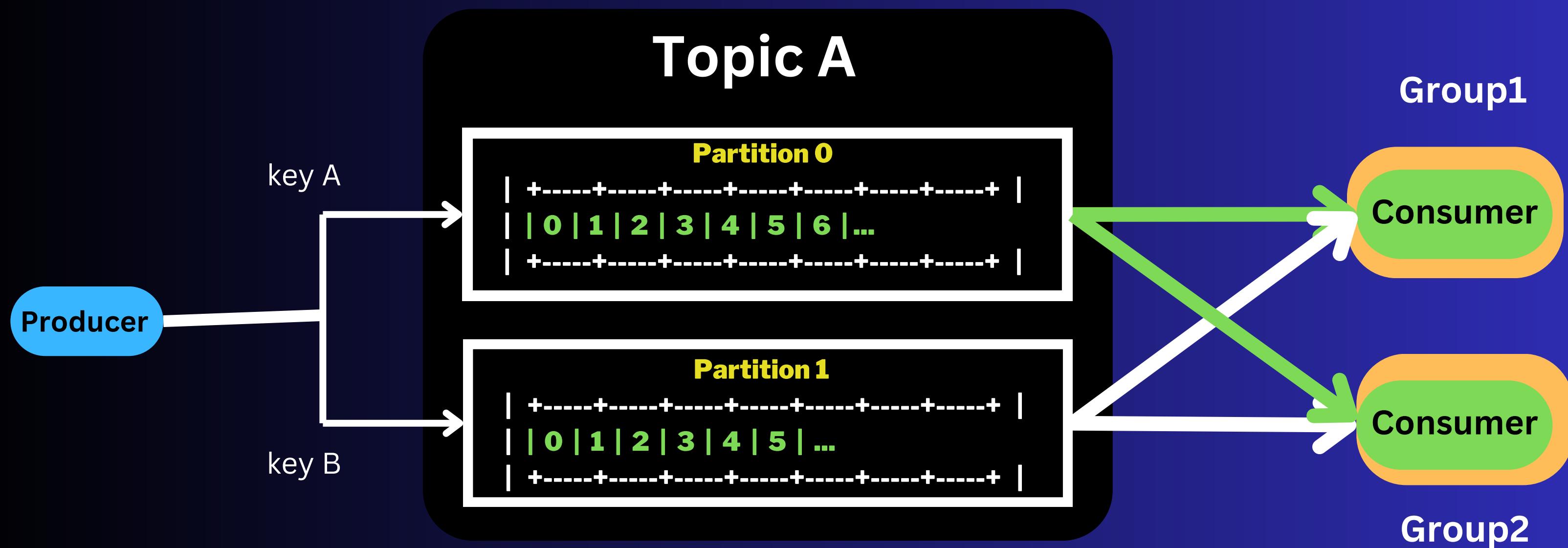
# Practical



- Create a Topic with 2 partitions
- Start producer and send messages using key so that it will distribute among both partitions
  - `kafka-console-producer.bat --bootstrap-server localhost:9092 --topic test-topic3 --property parse.key=true --property key.separator=:`
- Create Two consumers connected to topic part of same group
  - `kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test-topic3 --group tg`

- Info about consumer group
  - **kafka-consumer-groups.bat --bootstrap-server localhost:9092 --describe --group my-group**
- check all groups
  - **kafka-consumer-groups.bat --bootstrap-server localhost:9092 --list**

# Example 3



# Kafka UI

# Kafka UI

Web-based open-source tool that provides an easy-to-use interface for monitoring, managing, and interacting with Apache Kafka clusters, including topics, partitions, consumer groups, and messages.

- **Edit kraft/server.properties**
  - **advertised.listeners=PLAINTEXT://<IP>:9092,CONTROLLER://localhost:9093**
    - Windows: ipconfig
    - Linux/MAC: ifconfig.
  - Restart kafka
- **docker run --rm -d -p 8080:8080 -e KAFKA\_CLUSTERS\_0\_NAME=my-cluster -e KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS=<IP>:9092 --name kafkaui provectuslabs/kafka-ui**

# Python + Kafka

- Python program which takes user email and store in a text file.
- Producer.py
  - It will keep checking the text file if any new user sign up (email added) and send it to Kafka topic.
- Consumer.py
  - It will keep checking any new data in kafka topic and consume the messages.

# Kafka Streams

**Kafka Streams is a Java library for real-time stream processing on Kafka topics. It allows you to read, transform, aggregate, and write data back to Kafka—all within your application.**

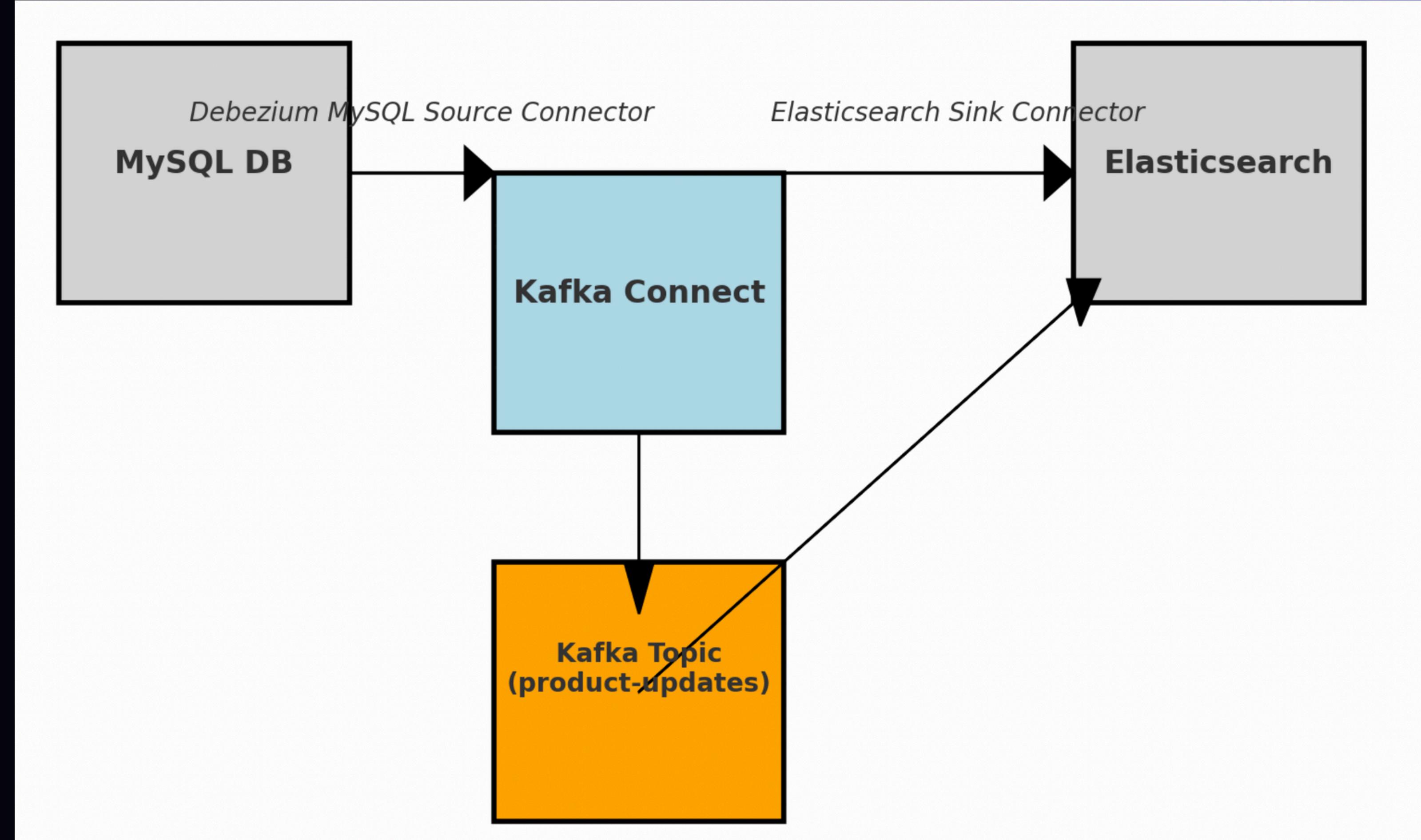
### ◆ How Kafka Streams Works?

- 1 Consumes data from a Kafka topic** (reads messages as a continuous stream).
- 2 Processes data in real time** (filters, maps, aggregates, joins, etc.).
- 3 Writes transformed data to another Kafka topic** (or external databases).

# Kafka Connect

**Kafka Connect** is a tool in Apache Kafka that helps move data between Kafka and other systems (like databases, cloud storage, or message queues) automatically using connectors, without needing to write custom code. 

- Uses **Source Connectors** to pull data from external systems into Kafka topics.
- Uses **Sink Connectors** to send data from Kafka topics to external systems.





# Interview Questions

## ◆ Basic Kafka Questions

1. What is Apache Kafka, and why is it used?
2. Explain the core components of Kafka.
3. What is a Kafka broker? How does it function?
4. What is a Kafka topic, and how is it different from a partition?
5. How does Kafka ensure durability and fault tolerance?
6. What is the role of Zookeeper in a Kafka cluster?
7. What is a producer in Kafka, and how does it send messages?
8. What is a Kafka consumer, and how does it consume messages?
9. Explain the difference between consumer groups and partitions.
10. How does Kafka handle message retention and log compaction?

## ◆ Intermediate Kafka Questions

1. **How does Kafka achieve high throughput and scalability?**
2. **What is ISR (In-Sync Replicas) in Kafka? How does it work?**
3. **What happens if a Kafka broker goes down? How does Kafka recover?**
4. **What are Kafka replication factors, and why are they important?**
5. **How does Kafka handle backpressure when consumers are slow?**
6. **What is Kafka's exactly-once semantics, and how does it work?**
7. **Explain the differences between Kafka's at-most-once, at-least-once, and exactly-once delivery guarantees.**
8. **How does Kafka handle leader election for partitions?**
9. **What is Kafka Streams, and how is it different from Kafka Connect?**
10. **What are Kafka's different types of acks (acknowledgment) settings, and how do they impact performance?**

