Big Data Systems

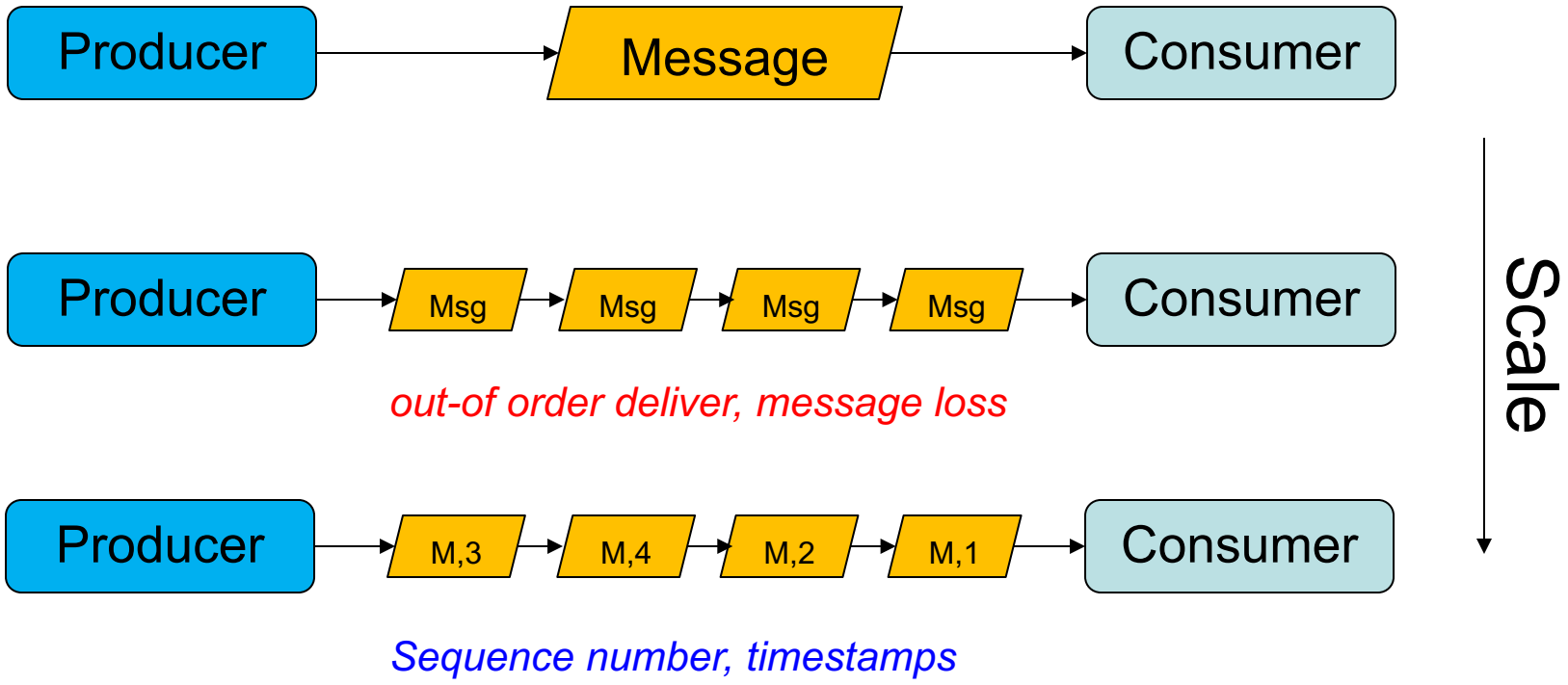
Djellel Difallah

Spring 2023

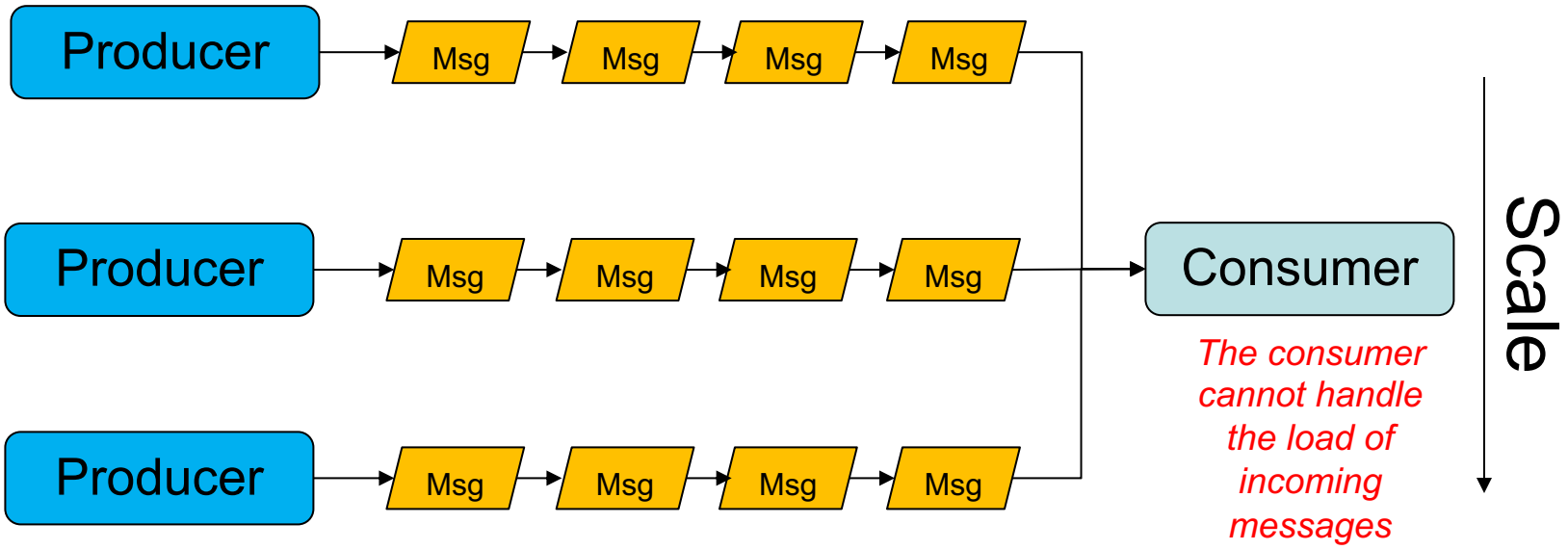
Lecture 12 (cont.) – Stream Processing

Apache Kafka

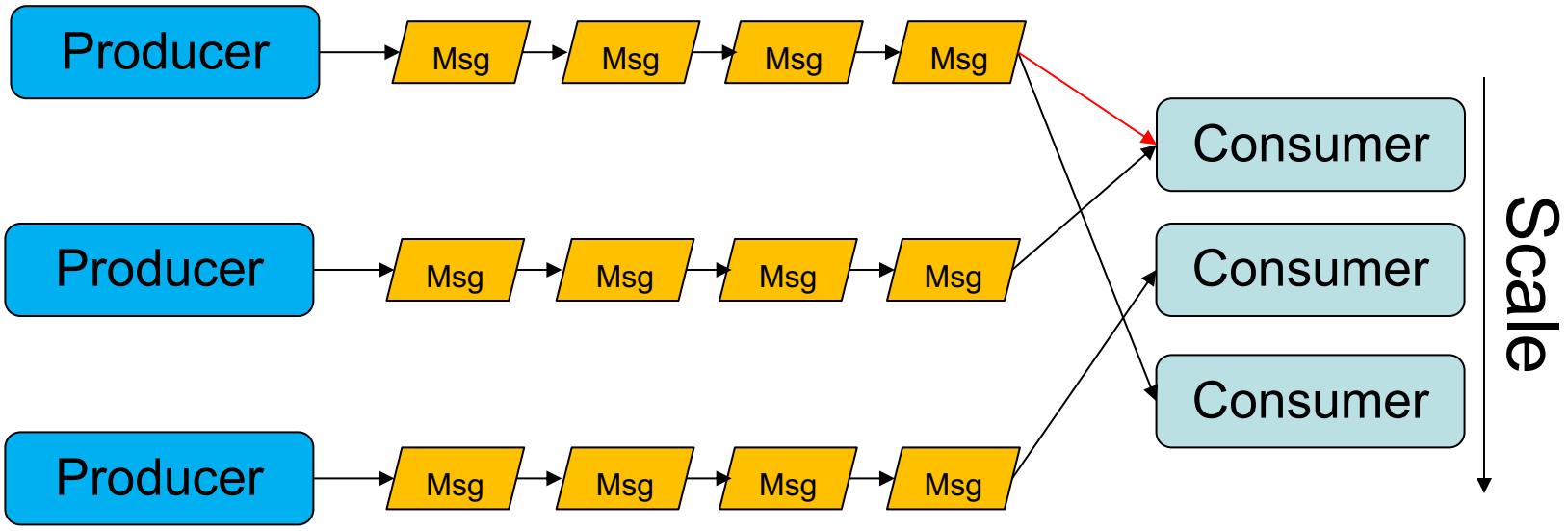
Producer Consumer Communication



Producer Consumer Communication



Producer Consumer Communication

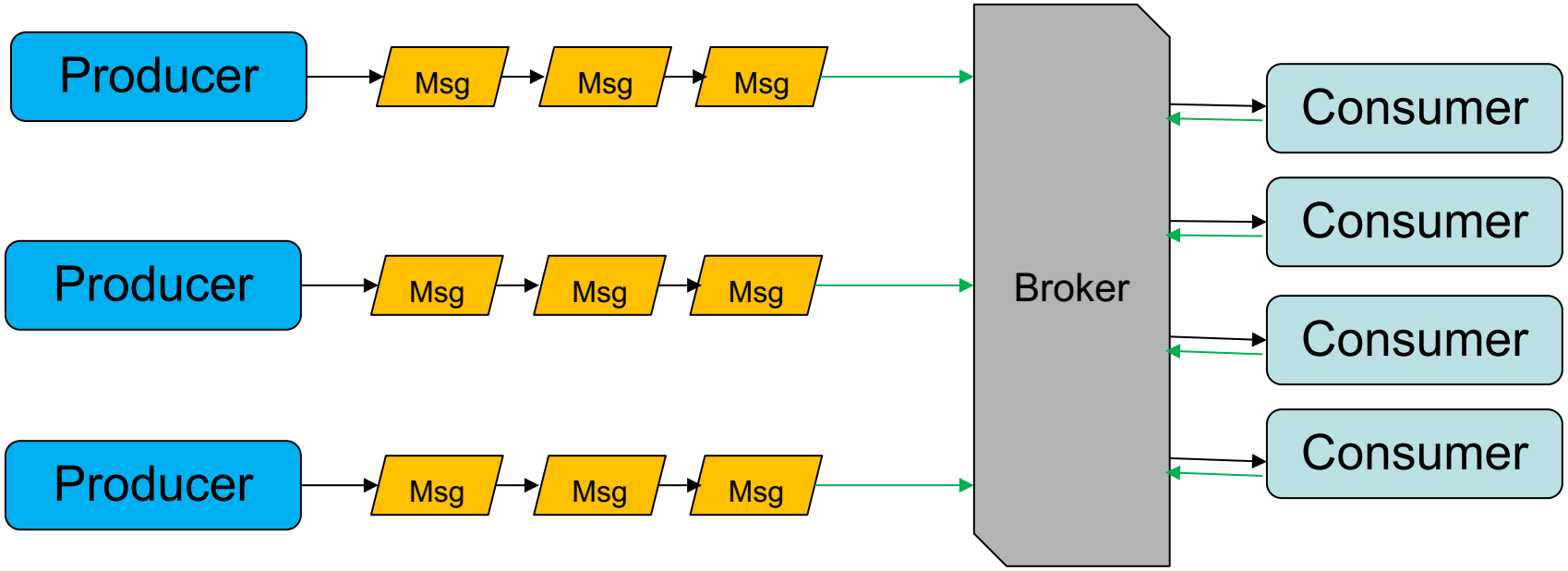


consumers don't need all the data → create topical streams of messages

Producer Consumer Communication

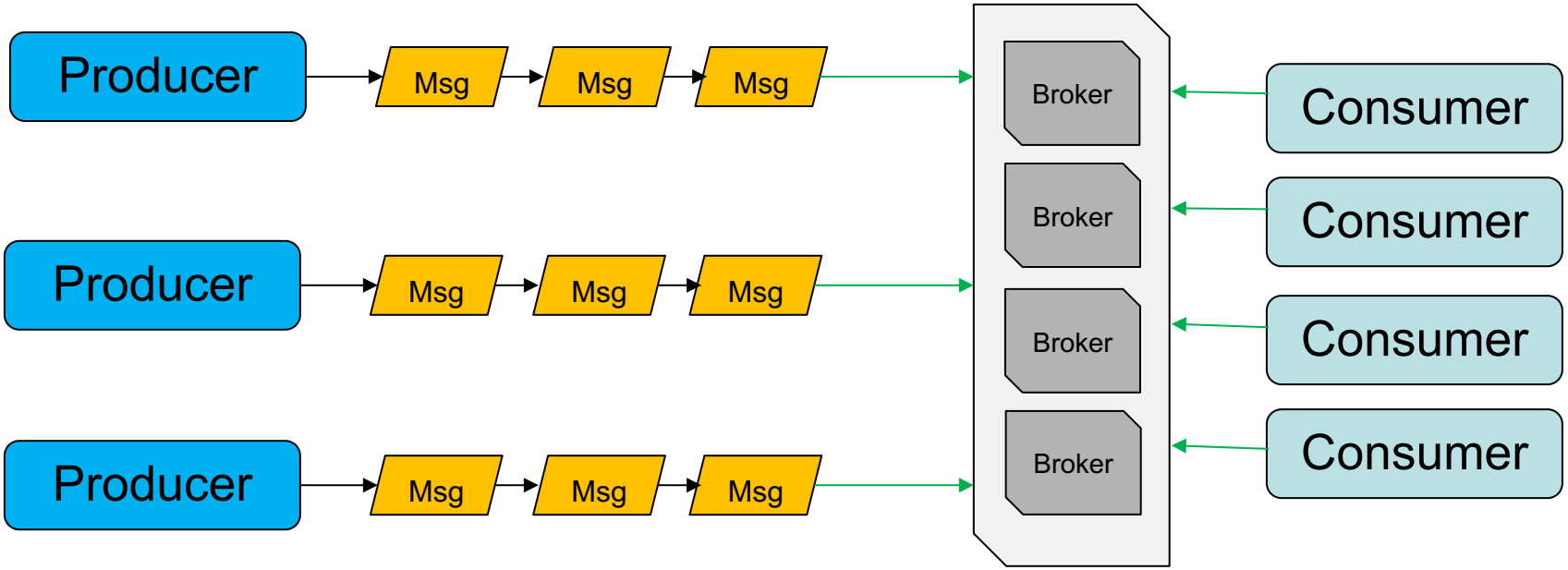
- Dedicate one consumer to one producer
  - Workload changes overtime → underutilized or over-utilized machines
- One producer to many consumers
  - Producer needs to keep track of consumers
  - Consumer failure puts load on producer
- How do we achieve proper load balancing?
  - Use a middleware (a broker) that handles the distribution of messages to consumers

Producer Consumer Communication



A message broker (distributor) between the consumers and producers is functionally independent from the consumers and producers but gives them an API to interact

Producer Consumer Communication



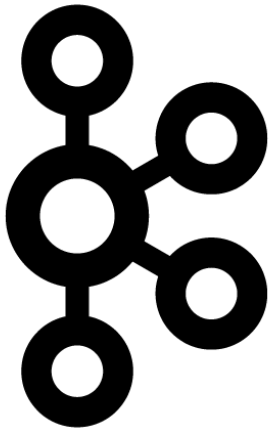
More Brokers to handle the load when necessary

Scaling out the Brokers

- What are the issues of having multiple brokers?
  - Consumers need to know which brokers are responsible for a specific topic
  - With high traffic, If a broker goes down, the entire topic's message transfer is lost
  - Brokers needs to synchronize
- Failure: Topic replications
  - Assign a lead broker per topic
  - Copy to other brokers
- Scalability: Partition the topics data
  - Apply some topic wide order (Sequence/Timestamp).
    - Works as long as the consumers read from ALL partitions.
  - Dynamic partitioning
  - Add replication by having a "lead" broker not by topic but by partition

Distributed Messaging System

APACHE KAFKA

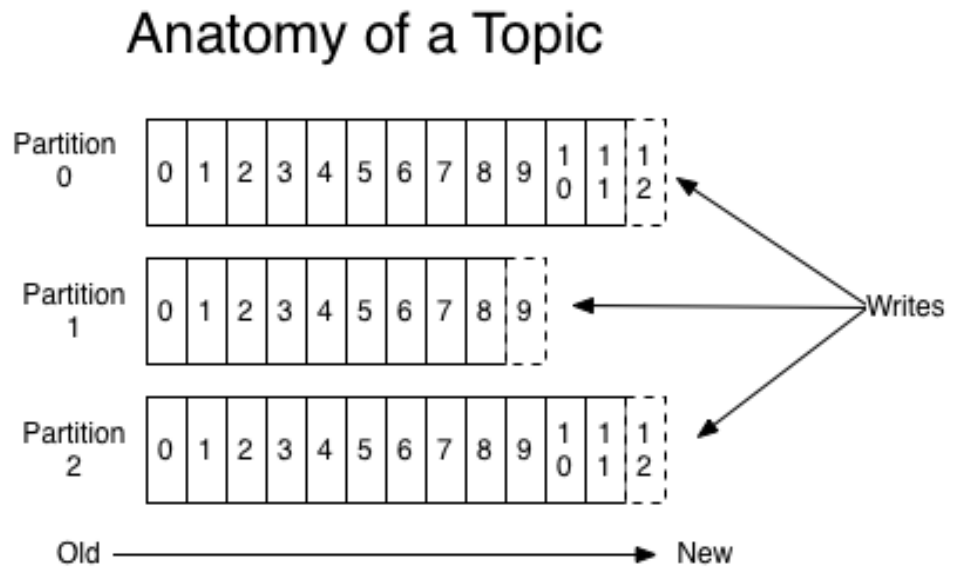


Apache Kafka

- Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system
  - It offers high throughput for both publishing and subscribing.
  - It supports multi-subscribers and automatically balances the consumers during failure.
- Kafka is strongly consistent
  - 1 Leader replica
  - Wait for "all" the replicas to complete (commit)
  - Tolerate N-1 failures with N replicas
  - Tradeoff based on the assumption that the Kafka cluster is within the same datacenter →network delay is small

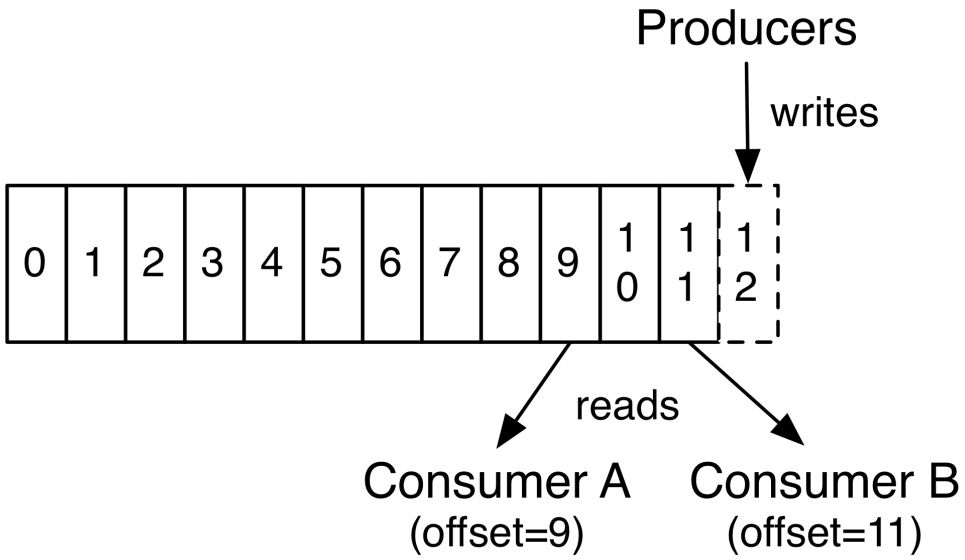
Commit Log Abstraction

Topic

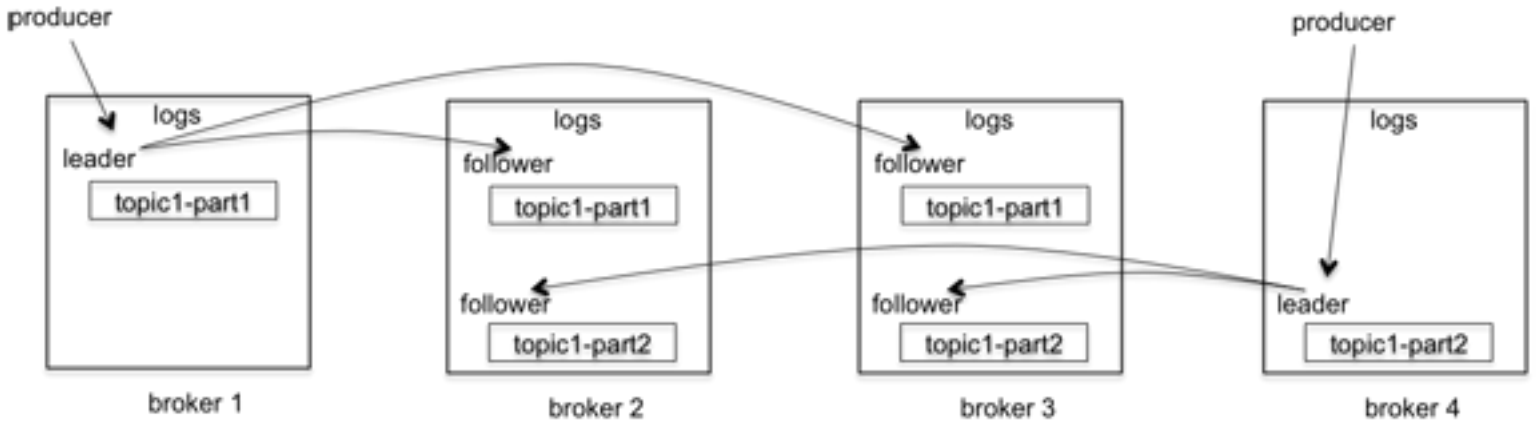


Commit Log Abstraction

Topic Partition



Kafka Partitioning and Replication



Kafka Operations

- Write: Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
  - That is, if a procuder sends 2 recors (M1 then M2): M1 will have a lower offset than M2 and appear earlier in the log.
- Read: A consumer instance sees records in the order they are stored in the log.
- Log Cleanup:
  - Deletion:
    - This can be set based on time (e.g., retain messages for 7 days) or based on the size of the log (e.g., retain messages until the log reaches 1GB in size).
    - Record deletion uses FIFO and is applied per topic and/or per partition
  - Compaction: Keep the latest value for a given key.

Apache Kafka Streaming Use Case

- Raw input data is consumed from Kafka topics and then aggregated, enriched, or otherwise transformed into new topics for further consumption or follow-up processing