CPSC 436C
Cloud Computing for Data Science

Stream Processing – Part 1

Maryam R.Aliabadi

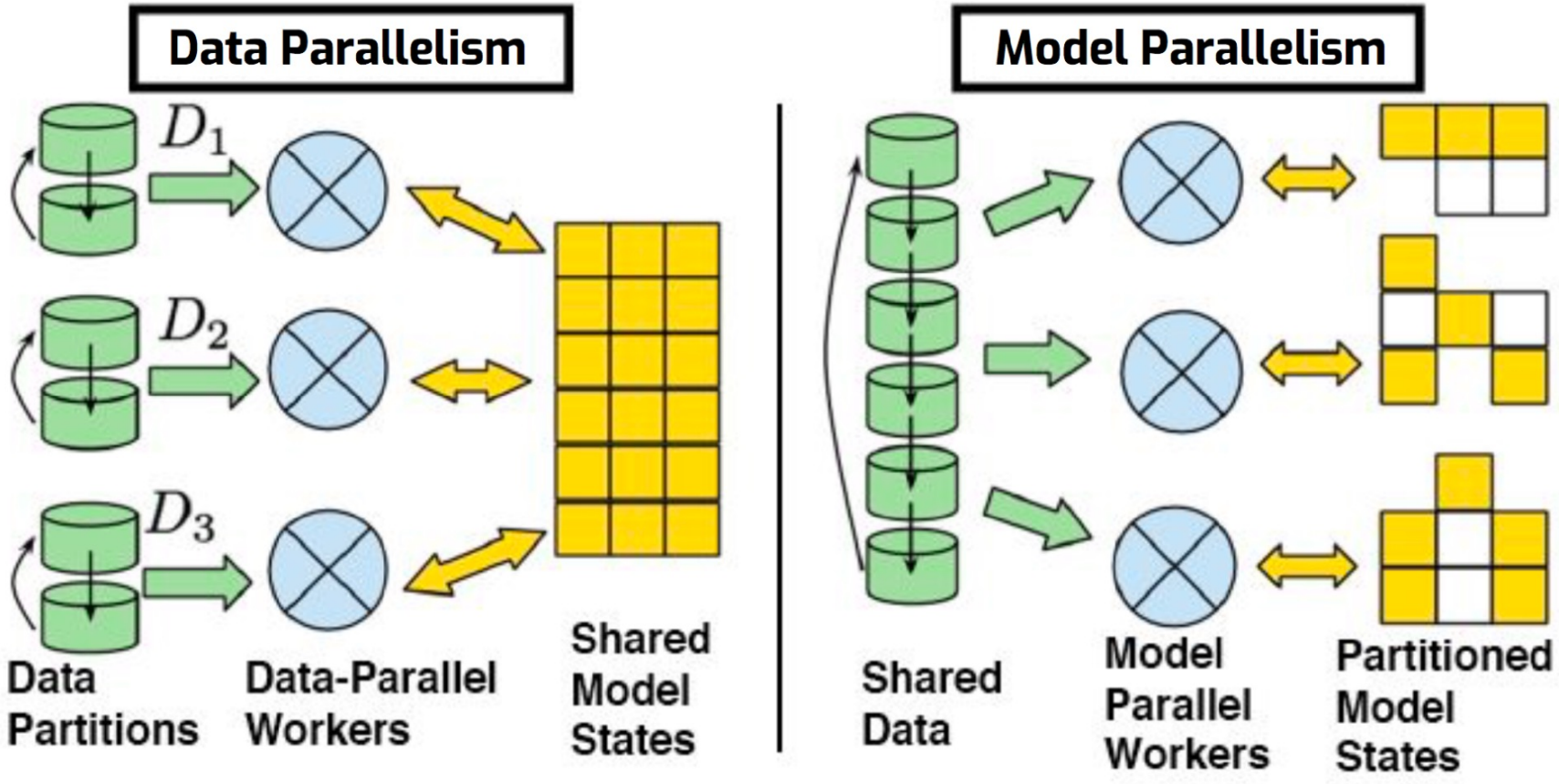
mraiyata@cs.ubc.ca

Spring 2024

Last Week's Review

- Scalability matters
- Parallelization
- Data Parallelization
  - Parameter server vs. AllReduce
  - Synchronized vs. asynchronized
- Model Parallelization

Data Vs. Model Parallelism



Data Parallelism Design Issues

- The aggregation algorithm
- Communication synchronization and frequency
- Communication compression

The Aggregation Algorithm

- How to aggregate gradients (compute the mean of the gradients)?
- Centralized - Parameter Server
- Decentralized - AllReduce
- Decentralized - Gossip

Communication Synchronization

- Synchronizing the model replicas in data-parallel training requires communication
  - between workers, in AllReduce
  - between workers and parameter servers, in the centralized architecture
- The communication synchronization decides how frequently all local models are synchronized with others.
- It will influence:
  - The communication traffic
  - The performance
  - The convergence of model training
- There is a trade-off between the communication traffic and the convergence.

Communication Compression

- Reduce the communication traffic with little impact on the model convergence.
- Compress the exchanged gradients or models before transmitting across the network.
  - Quantization
  - Sparsification

Quiz 1

Which of the following best describes data parallelism in distributed machine learning?

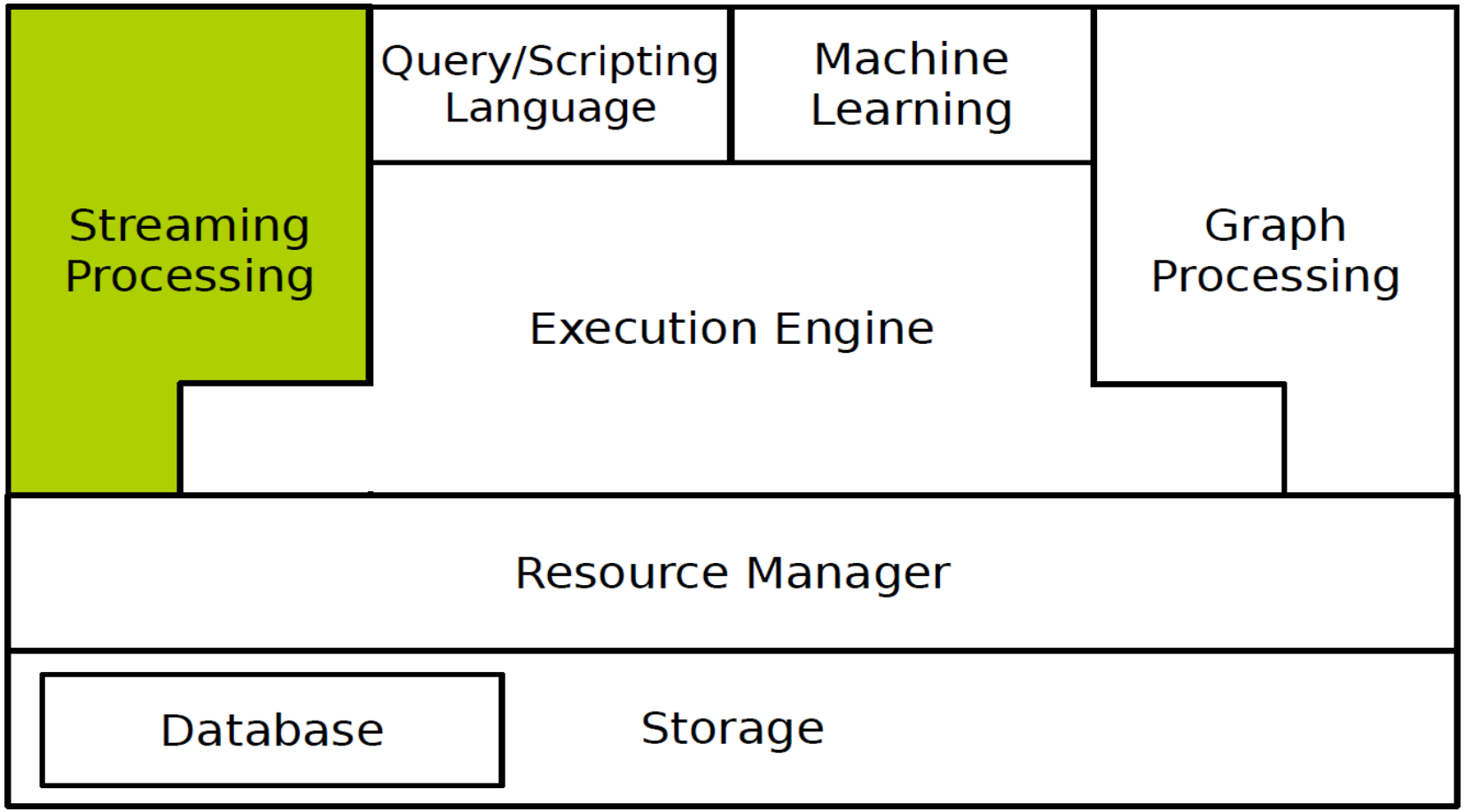
a) Training multiple models simultaneously on different datasets
b) Distributing data across multiple nodes for parallel processing
c) Synchronizing model updates across distributed nodes
d) Optimizing communication bandwidth for faster training

Quiz 2

How does the frequency of aggregation impact distributed machine learning?

a) Higher aggregation frequency leads to faster convergence
b) Lower aggregation frequency improves model accuracy
c) Aggregation frequency has no impact on training speed
d) Aggregation frequency determines the scalability of the algorithm

Today's Topics

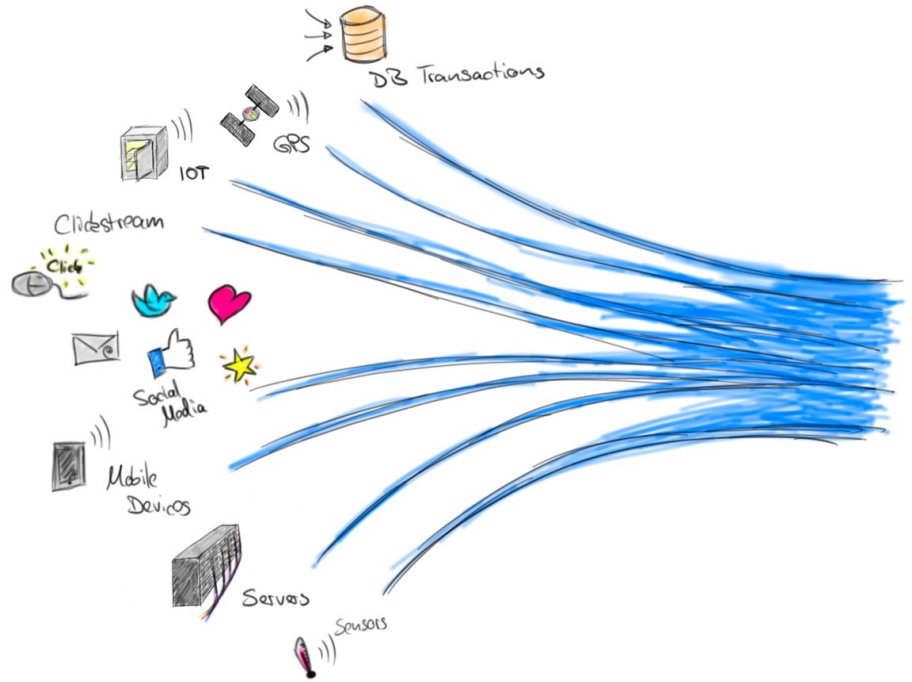


Stream Processing

- Stream processing is the act of continuously incorporating new data to compute a result.

Stream Processing

- The input data is unbounded.
  - A series of events, no predetermined beginning or end.
  - E.g., credit card transactions, clicks on a website, or sensor readings from IoT devices.

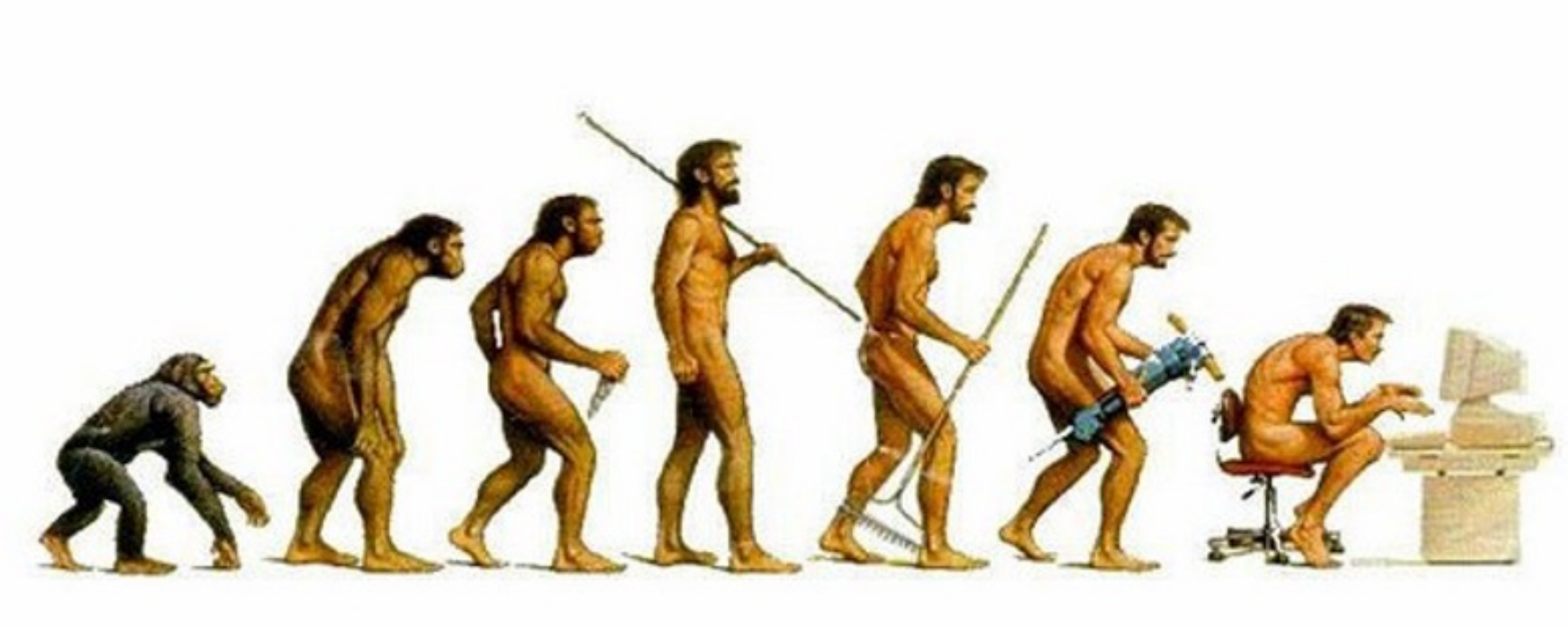


Motivation

- Many applications must process large streams of live data and provide results in real-time.
- Processing information as it flows, without storing them persistently.
- Traditional DBMSs:
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
  - Both aspects contrast with the above requirements.

Data Stream Management Systems

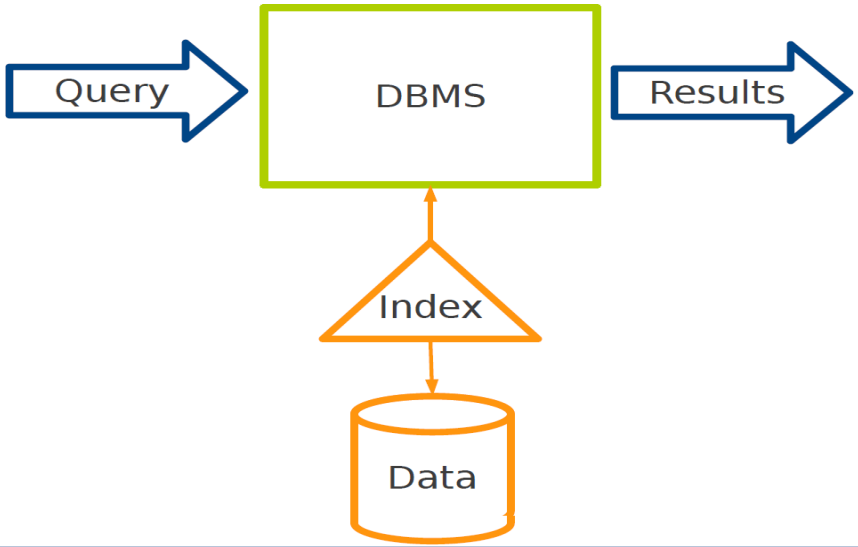
- An evolution of traditional data processing, as supported by DBMSs.

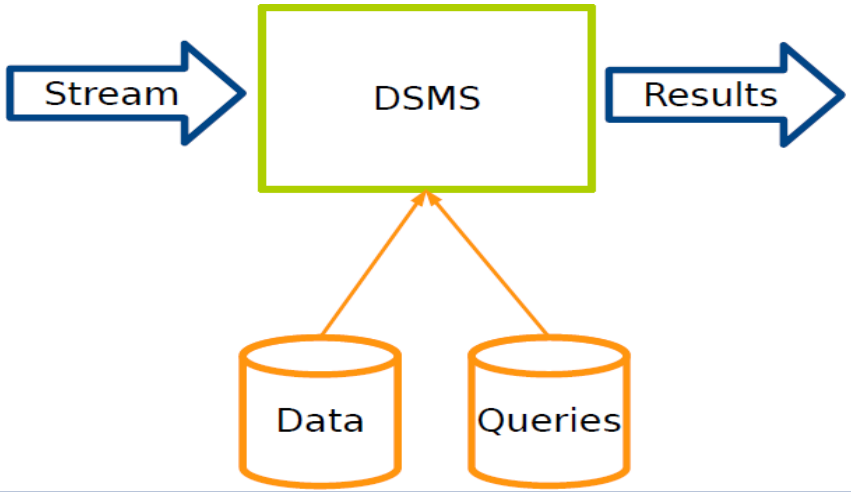


DBMS Vs. DSMS (1/3)

- DBMS: data-at-rest analytics
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
- DSMS (Data Streaming Management System): data-in-motion analytics

  Processing information as it flows, without storing them persistently.

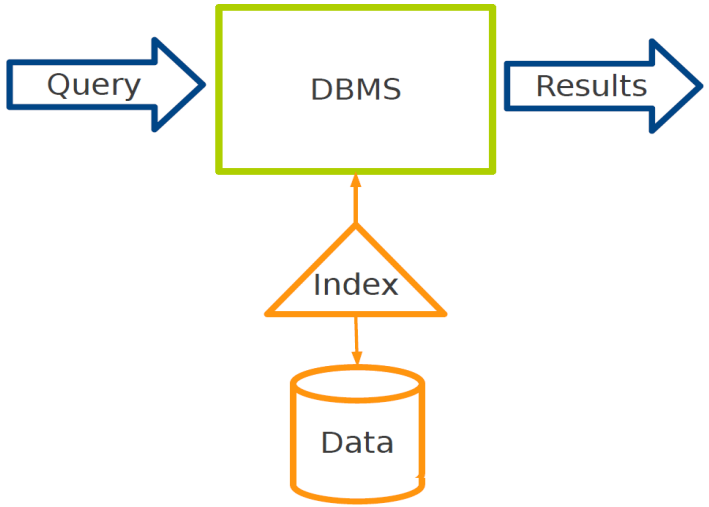


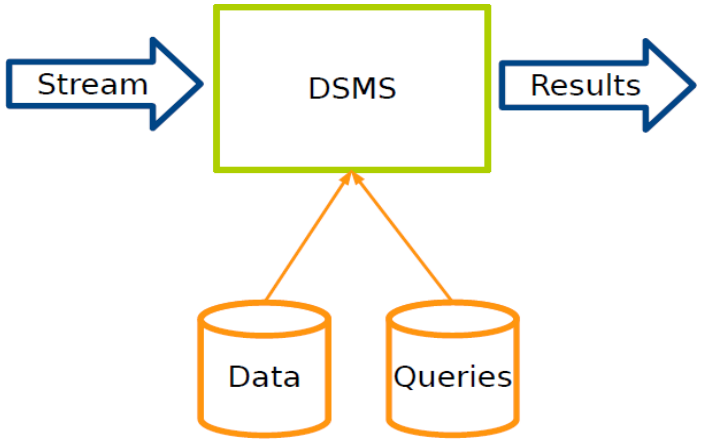


DBMS Vs. DSMS (2/3)

- DBMS: runs queries just once to return a complete answer.

- DSMS: executes standing queries, which run continuously and provide updated answers as new data arrives.

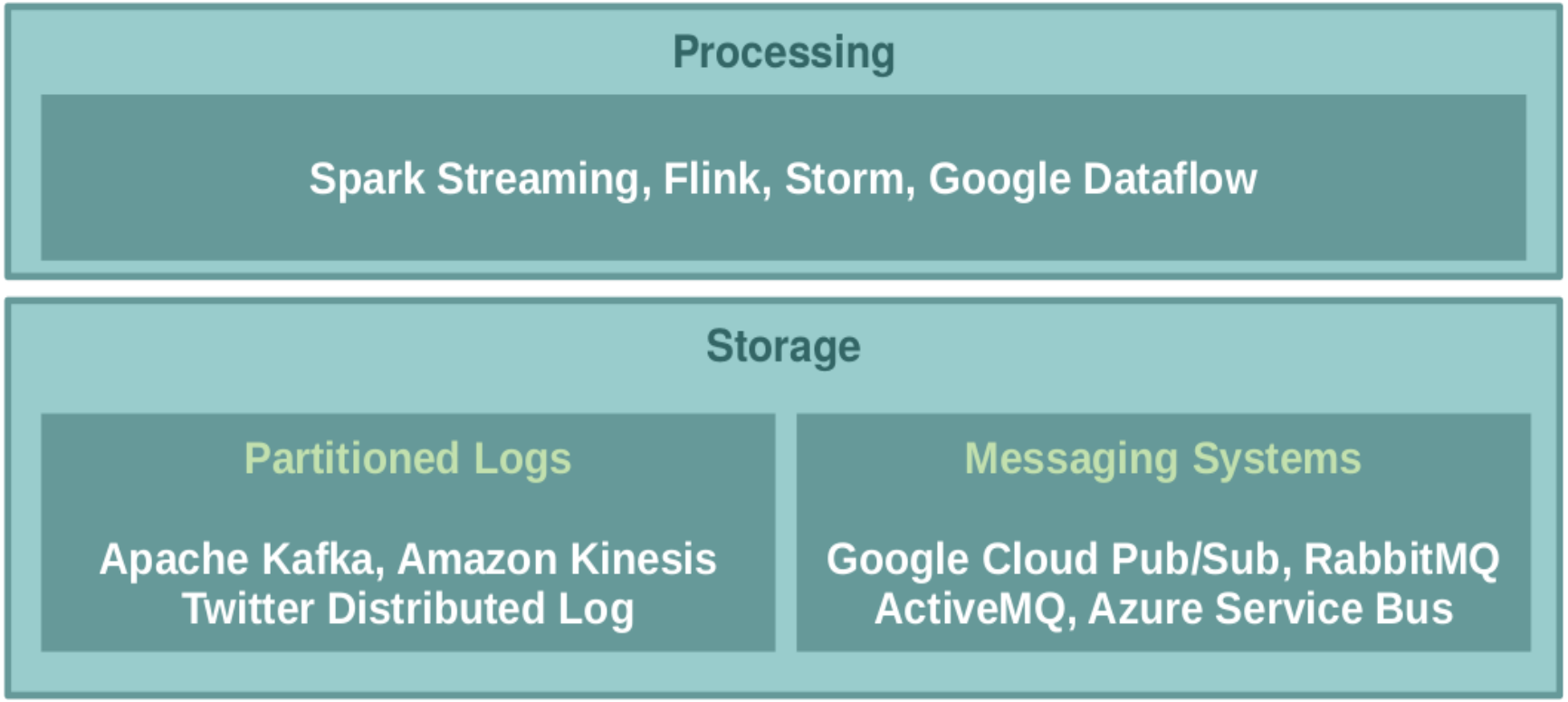




DBMS Vs. DSMS (3/3)

- Despite these differences, DSMSs resemble DBMSs: both process incoming data through a sequence of transformations based on SQL operators, e.g., selections, aggregates, joins.

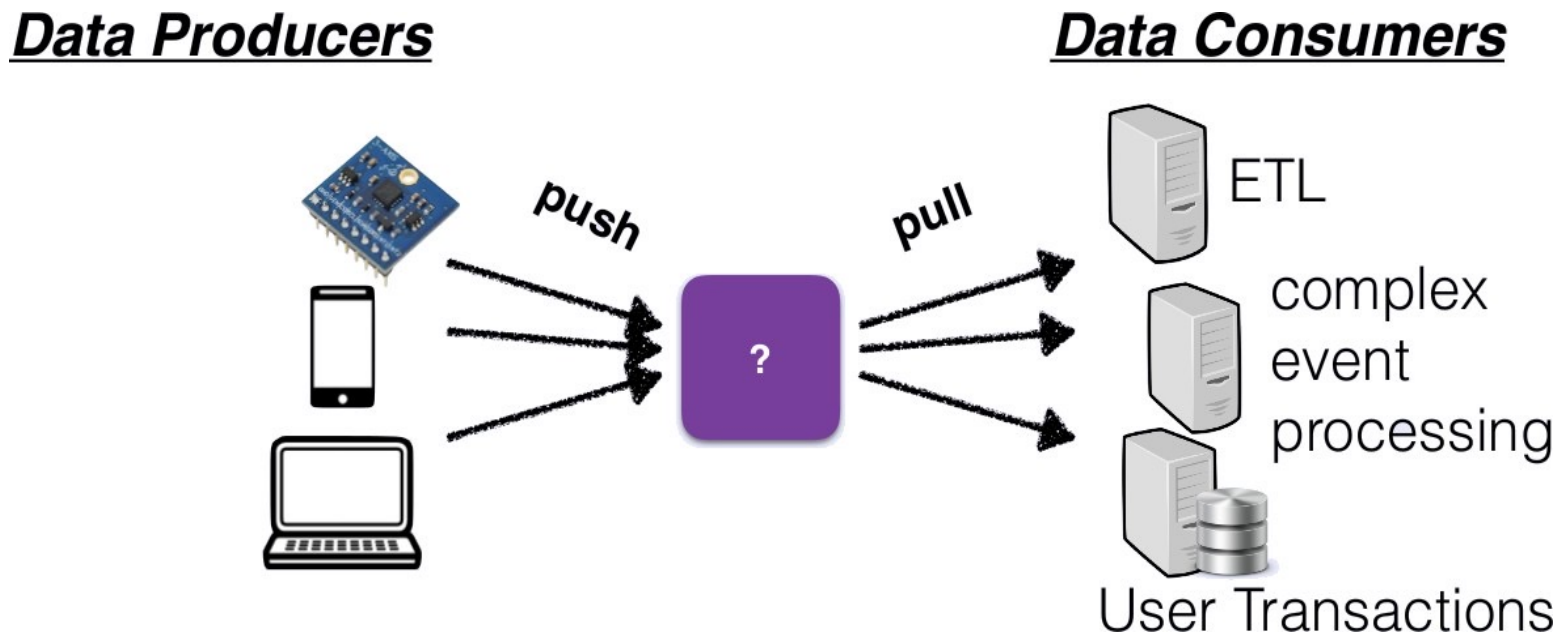
Stream Processing System Stack



Data Stream Storage

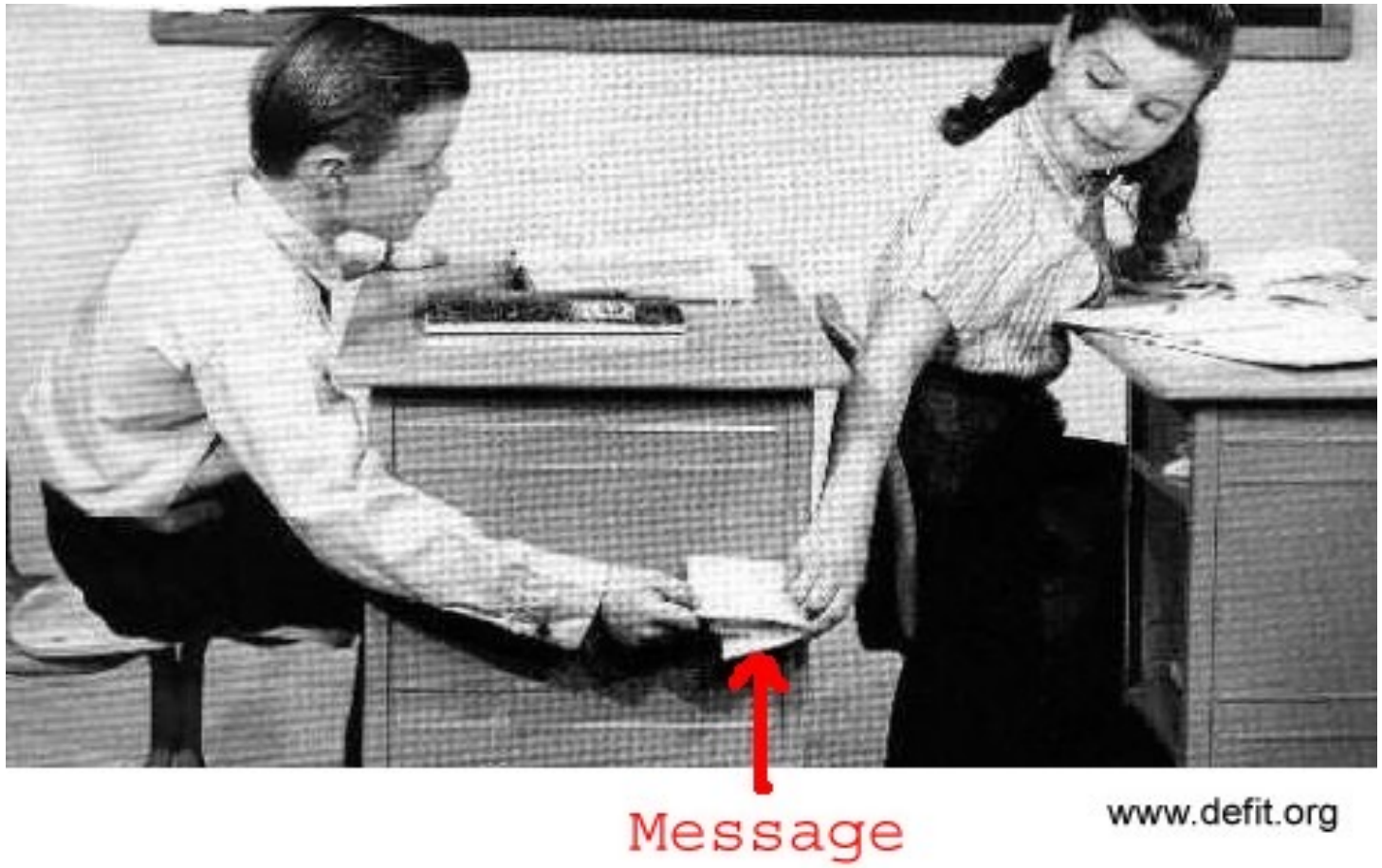
The Problem

- We need disseminate streams of events from various producers to various consumers.



Possible Solution

- Messaging systems

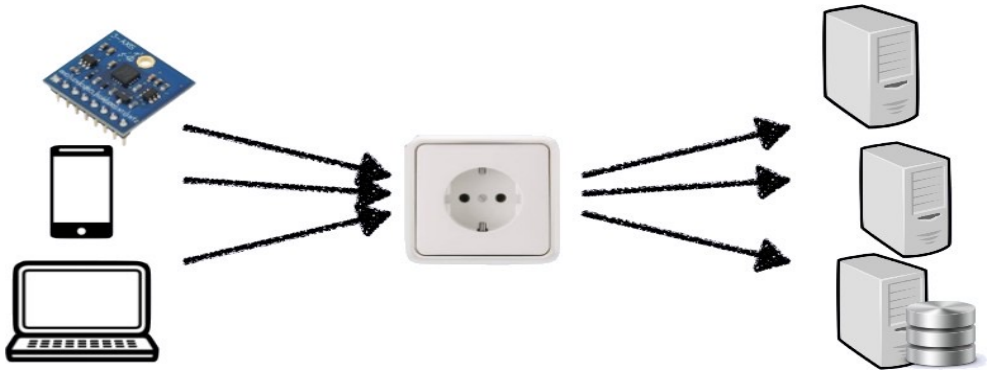


What is a messaging system?

- Messaging system is an approach to notify consumers about new events.
- Messaging systems
  - Direct messaging
  - Message brokers

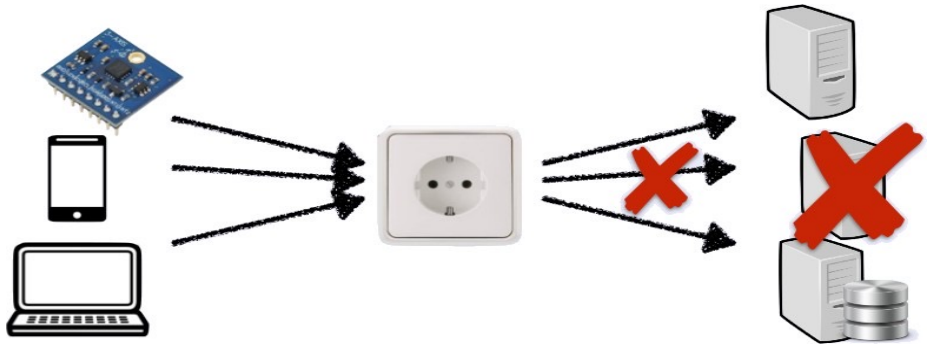
Direct messaging

- Necessary in latency critical applications (e.g., remote surgery).
- A producer sends a message containing the event, which is pushed to consumers.
- Both consumers and producers have to be online at the same time.

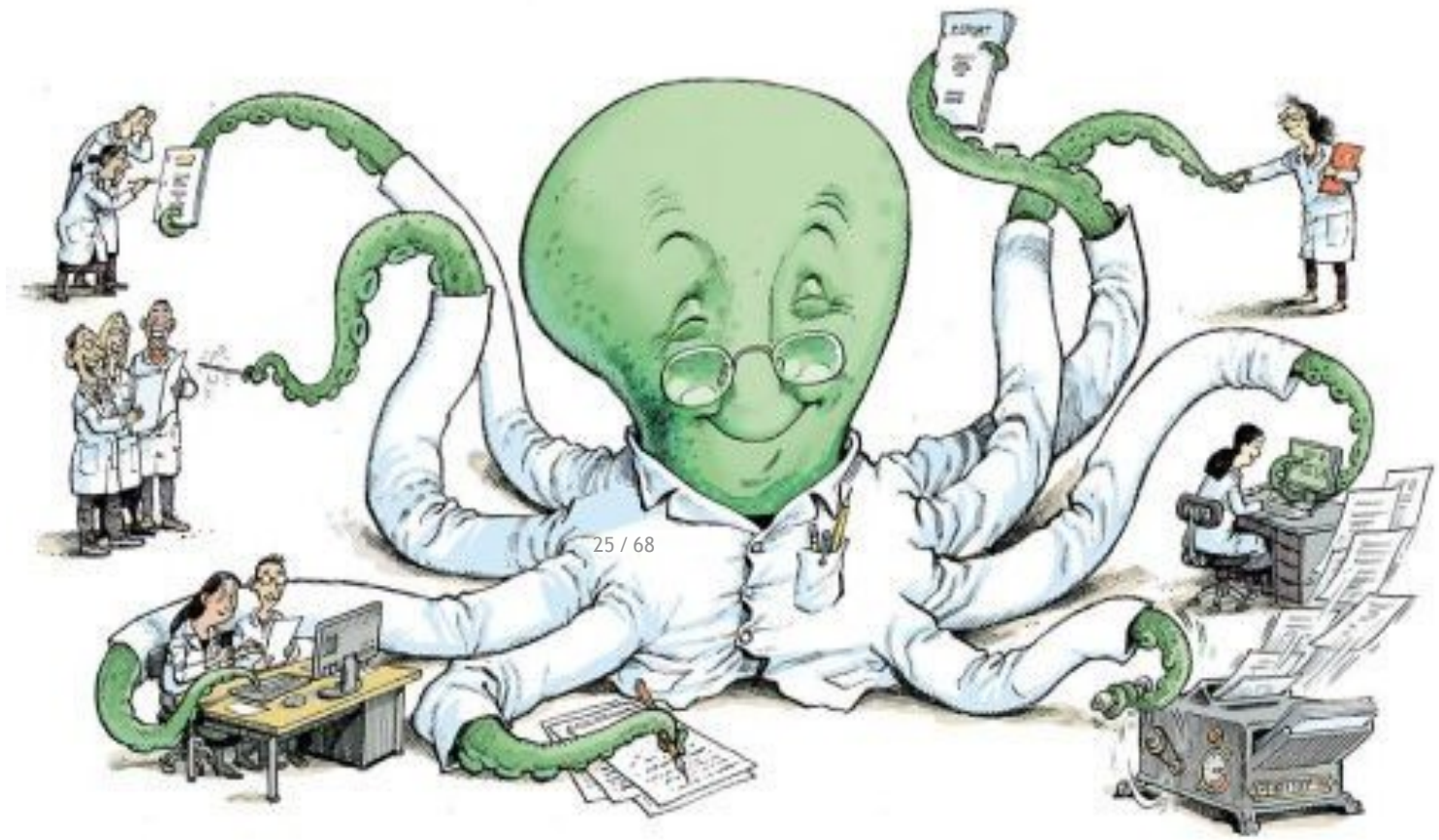


Direct messaging

- What happens if a consumer crashes or temporarily goes offline? (not durable)
- What happens if producers send messages faster than the consumers can process?
  - Dropping messages
  - Backpressure
- We need **message brokers** that can log events to process at a later time.



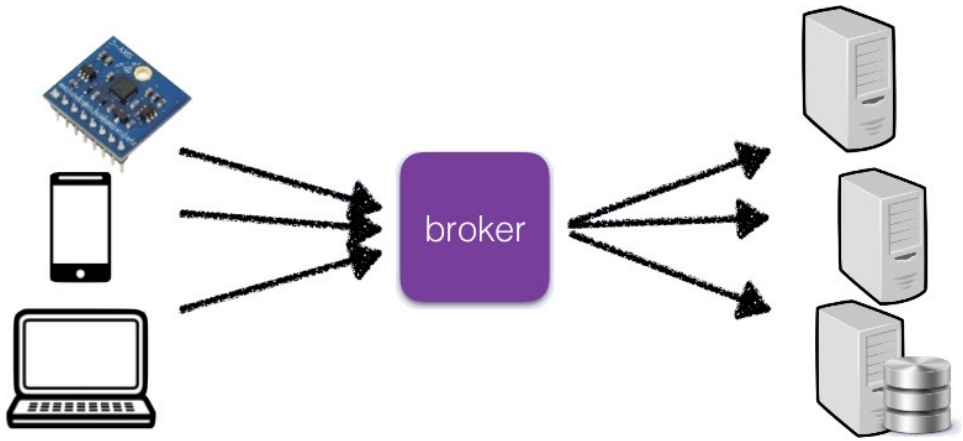
Message Broker



[https://bluesyemre.com/2018/10/16/thousands-of-scientists-publish-a-paper-every-five-days]

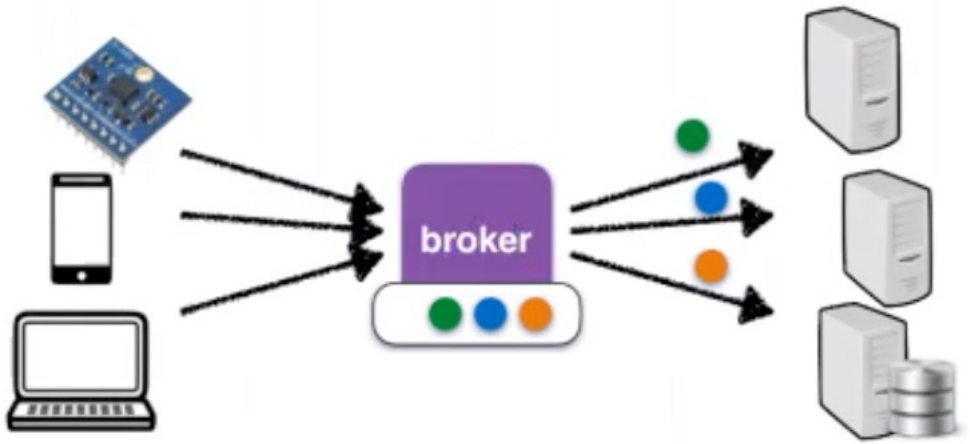
Message Broker

- A message broker decouples the producer-consumer interaction.
- It runs as a server, with producers and consumers connecting to it as clients.
- Producers write messages to the broker, and consumers receive them by reading them from the broker.
- Consumers are generally asynchronous.



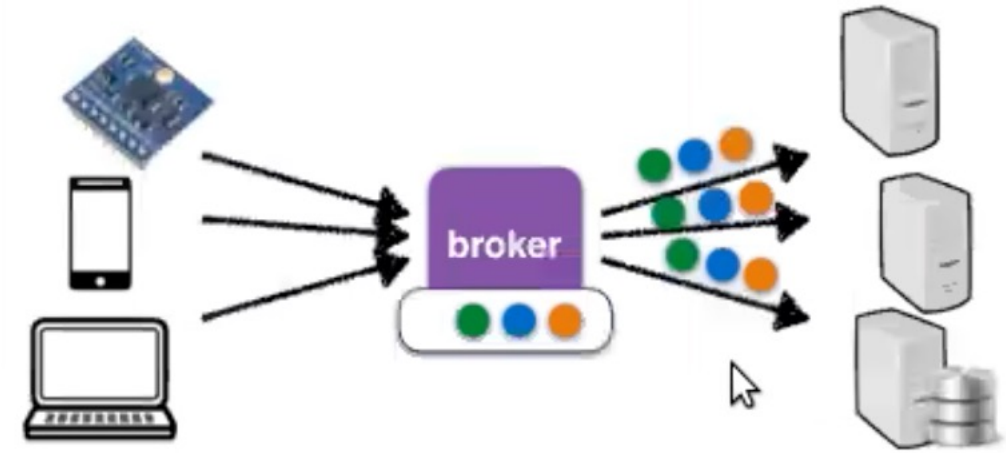
Message Broker

- When multiple consumers read messages in the same topic.
- Load Balancing: each message is delivered to one of consumers.
  - Distributing incoming messages efficiently among multiple consumers



Message Broker

- Fan-out: each message is delivered to all of consumers

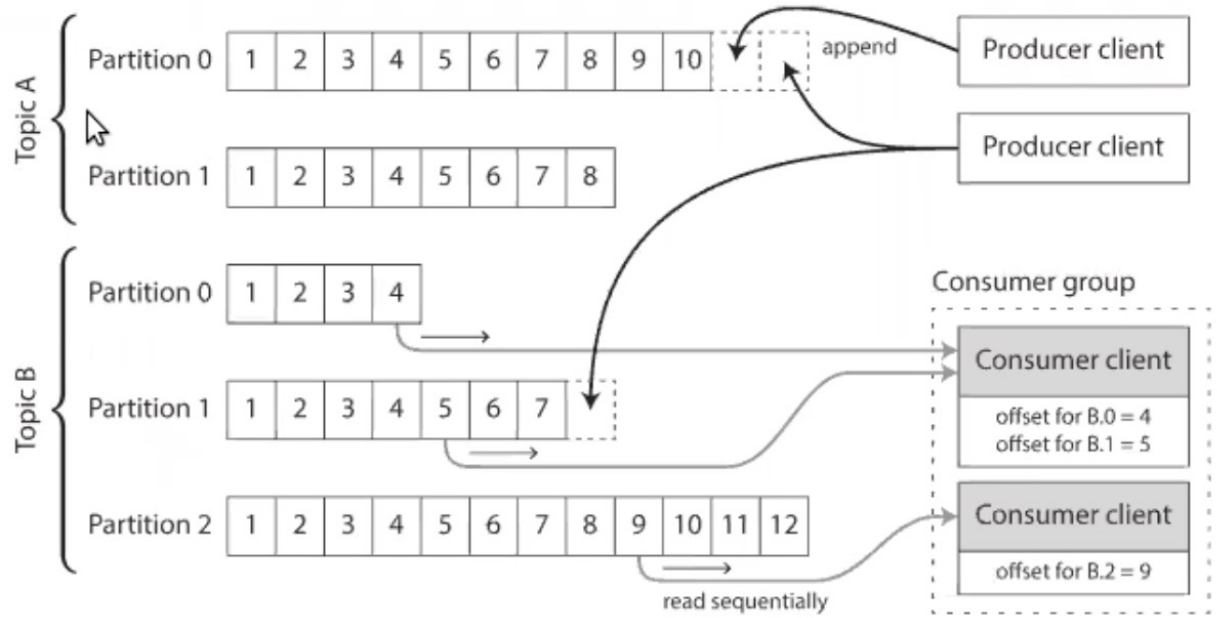


Partitioned Log

- In typical message brokers, once a message is consumed, it is deleted.
- Log-based message brokers durably store all events in a sequential log.
- A log is an append-only sequence of records on disk.
- A producer sends a message by appending it to the end of the log.
- A consumer receives messages by reading the log sequentially.

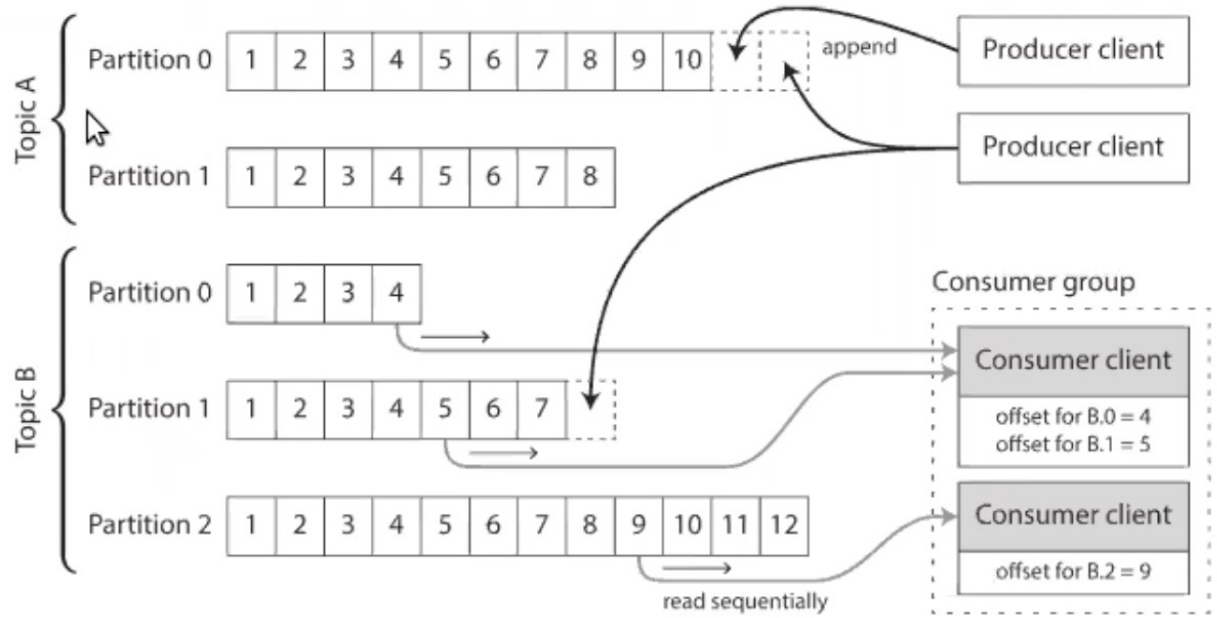
Partitioned Log

- To scale up the system, logs can be partitioned hosted on different machines.
- Each partition can be read and written independent of others.
- A topic is a group of partitions that all carry messages of the same type.



Partitioned Log

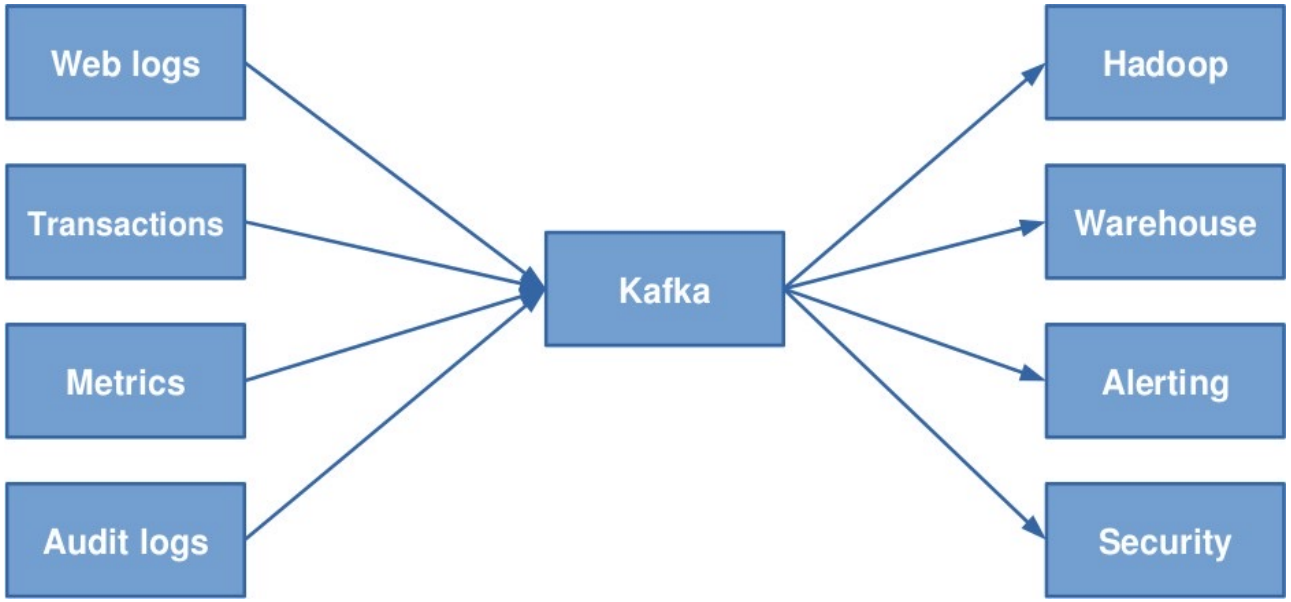
- Within each partition, the broker assigns monotonically increasing sequence number (offset) to every message.
- No ordering guarantee across partitions.



Kafka: A Log-based Message Broker

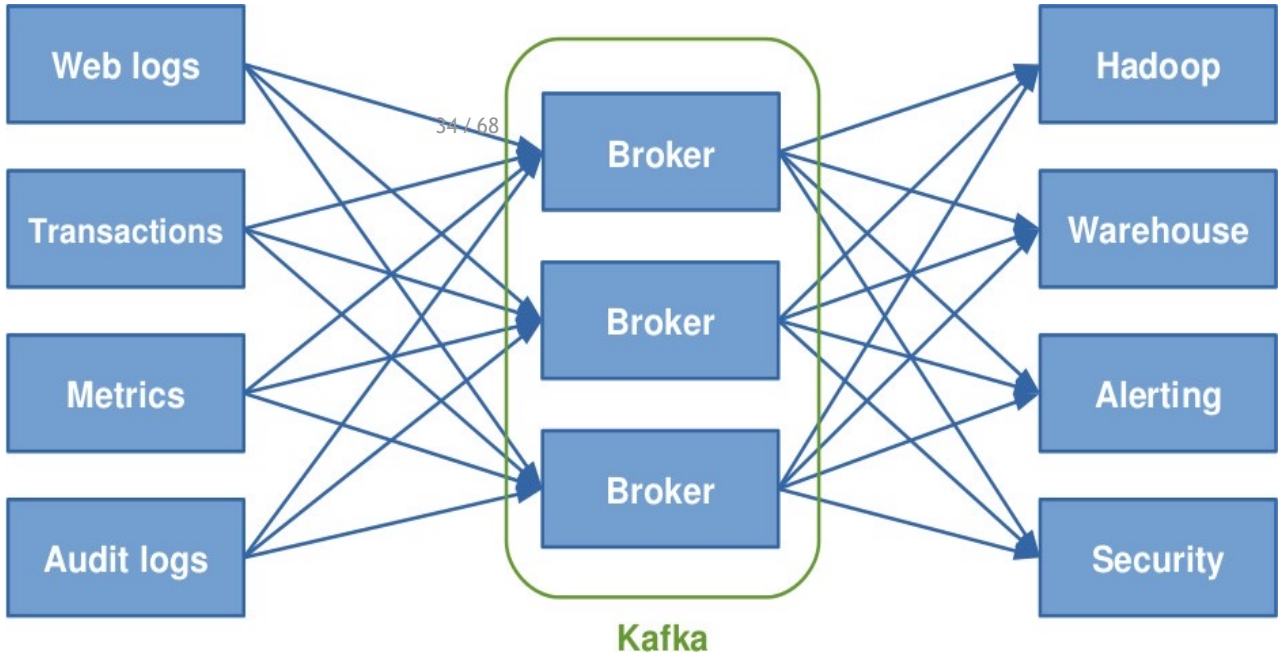
Kafka

- Kafka is a distributed, topic oriented, partitioned, replicated commit log service.



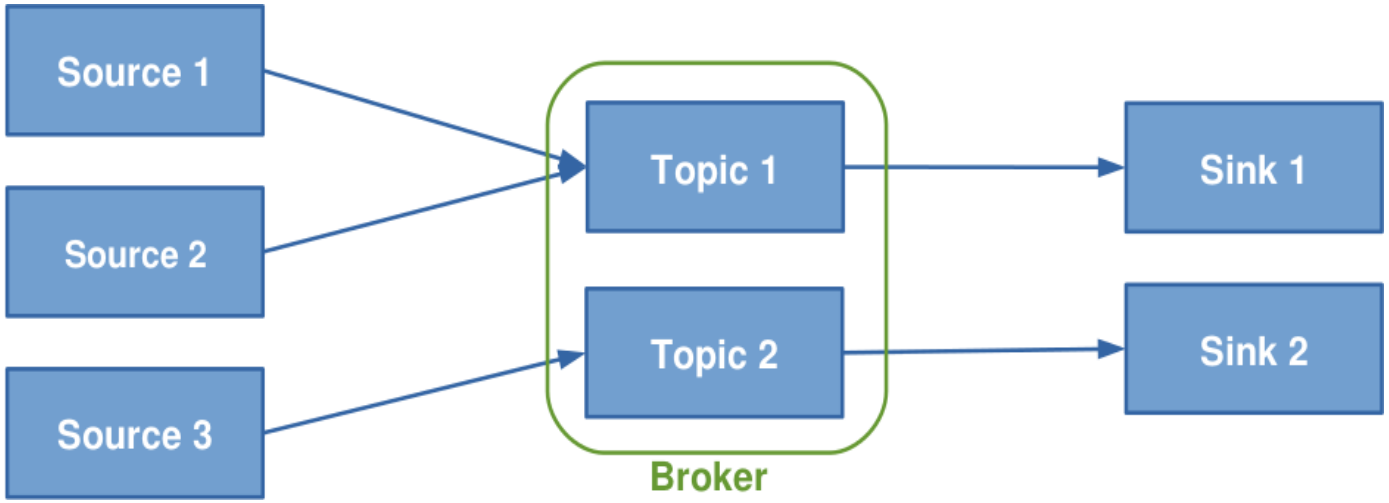
Kafka

- Kafka is a **distributed**, topic oriented, partitioned, replicated commit log service.



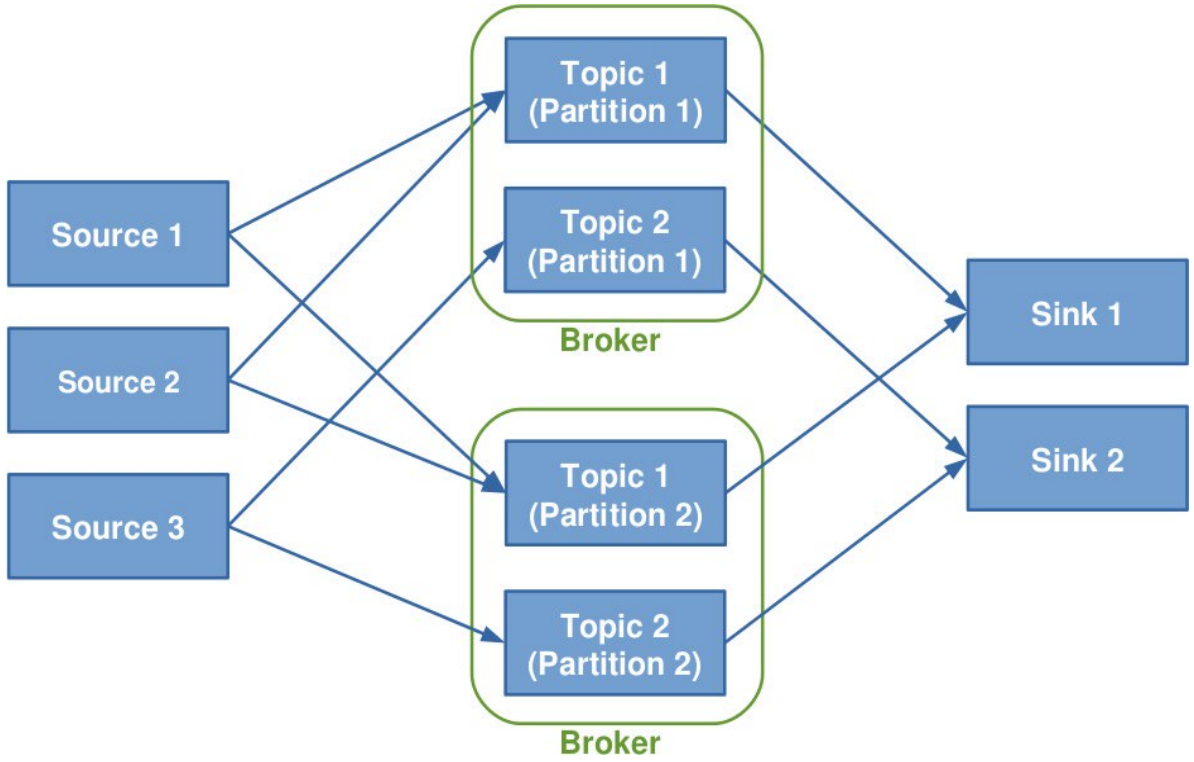
Kafka

- Kafka is a distributed, **topic oriented**, partitioned, replicated commit log service.



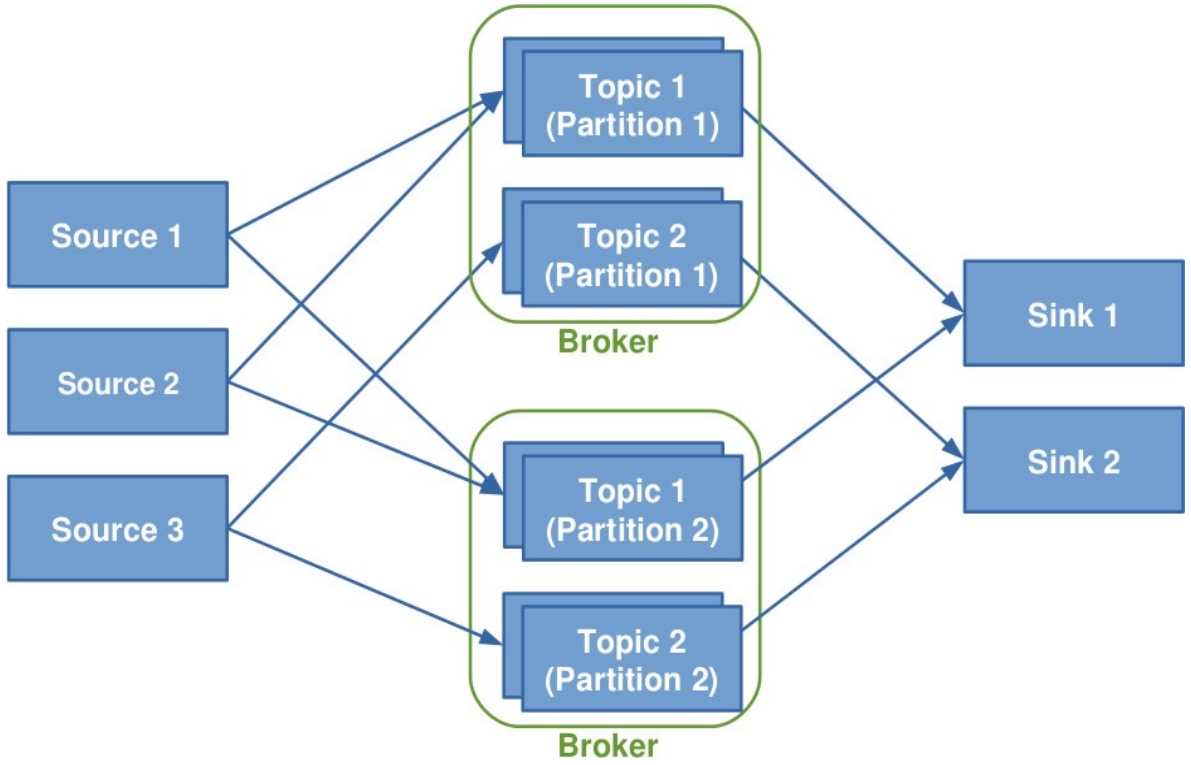
Kafka

- Kafka is a distributed, topic oriented, **partitioned**, replicated commit log service.

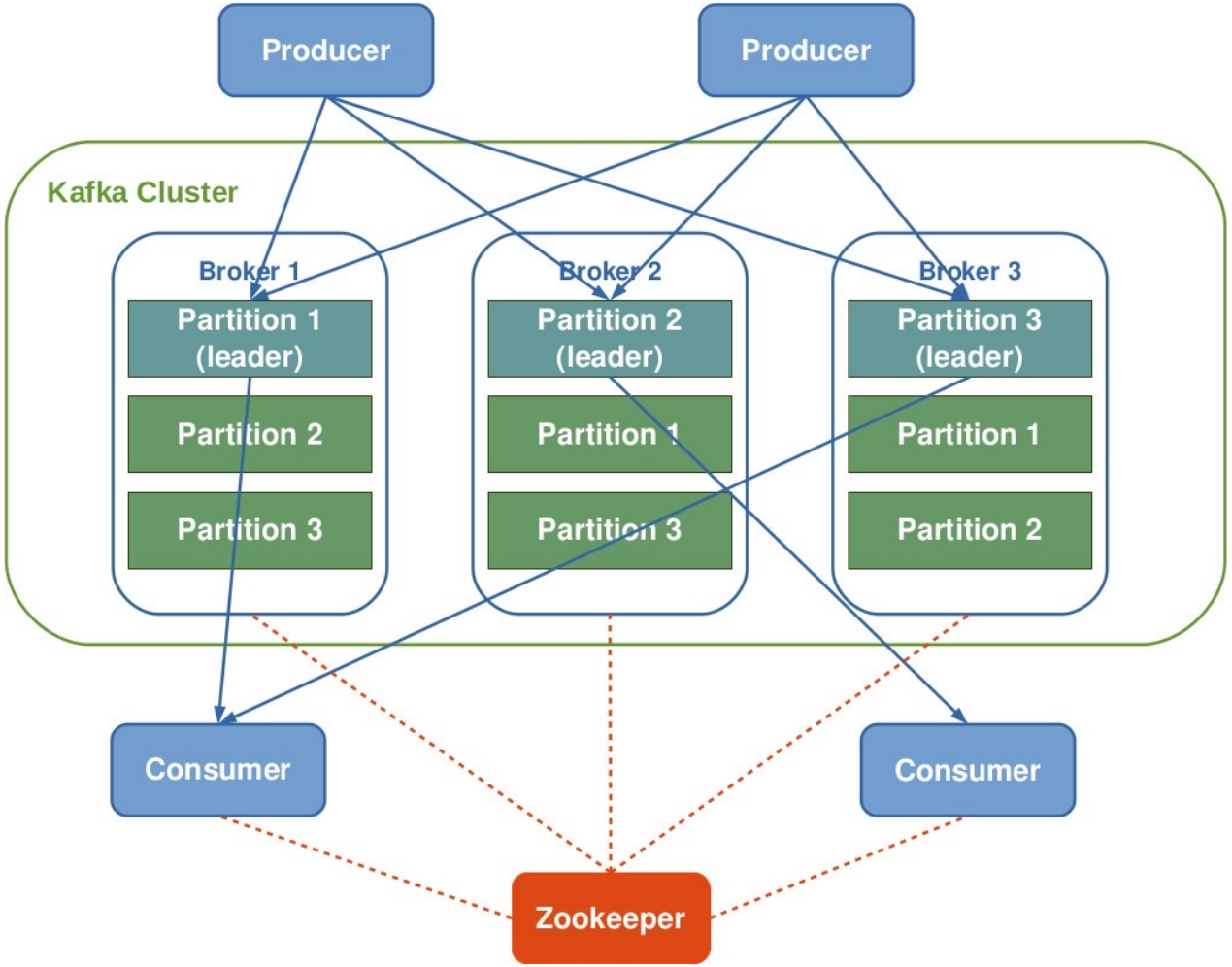


Kafka

- Kafka is a distributed, topic oriented, partitioned, **replicated** commit log service.



Kafka Architecture



Coordination

- Kafka uses Zookeeper for the following tasks:
  - Detecting the addition and the removal of brokers and consumers (keeping track of dynamic cluster membership)
  - Keeping track of the consumed offset of each partition.

State in Kafka

- Brokers are stateless: no metadata for consumers-producers in brokers.
- Consumers are responsible for keeping track of offsets.
- Messages in queues expire based on pre-configured time periods (e.g., once a day).

Data Stream Processing

Streaming Data

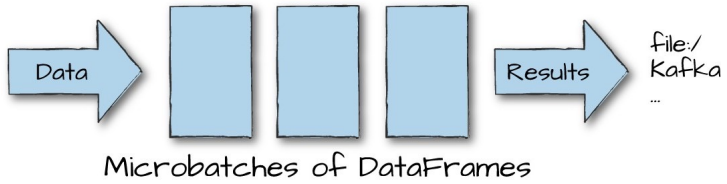
- Data stream is unbound data, which is broken into a sequence of individual tuples.
- A data tuple is the atomic data item in a data stream.
- Can be structured, semi-structured, and unstructured.

Streaming Data Processing Design Points

- Continuous vs. micro-batch processing
- Record-at-a-Time vs. declarative APIs
- Event time vs. processing time
- Windowing

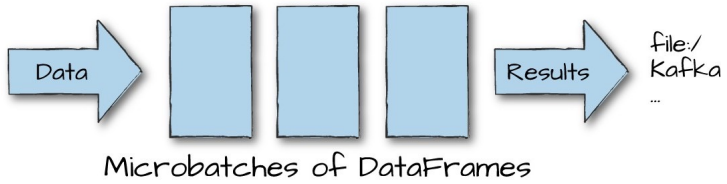
Continuous vs. micro-batch processing

- Micro-batch systems
  - Batch engines
  - Slicing up the unbounded data into a sets of bounded data, then process each batch.

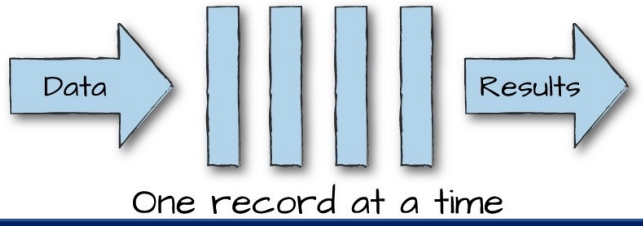


Continuous vs. micro-batch processing

- Micro-batch systems
  - Batch engines
  - Slicing up the unbounded data into a sets of bounded data, then process each batch.



- Continuous processing-based systems
  - Each node in the system continually listens to messages from other nodes and outputs new updates to its child nodes.



Record-at-a-Time vs. Declarative APIs

- Record-at-a-Time API (e.g., Storm)
  - Low-level API
  - Passes each event to the application and let it react.
  - Useful when applications need full control over the processing of data.
  - Complicated factors, such as maintaining state, are governed by the application.
- Declarative API (e.g., Spark streaming, Flink, Google Dataflow)
  - Applications specify what to compute not how to compute it in response to each new event.

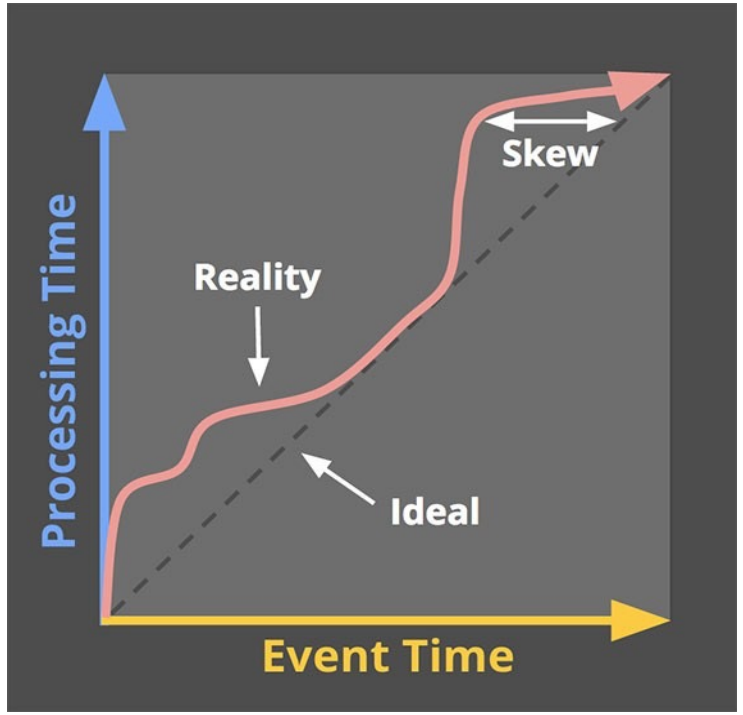
Event time Vs. Processing time

- Event time: the time at which events actually occurred.
  - Timestamps inserted into each record at the source.
  - Use case: Security monitoring

- Processing time: the time when the record is received at the streaming application.
  - Use case: Performance monitoring of stream processing application

Event time Vs. Processing time

- Ideally, event time and processing time should be equal.
- Skew between event time and processing time.



Next Topic:
Spark Streaming