CPSC 436C
Cloud Computing for Data Science

Stream Processing

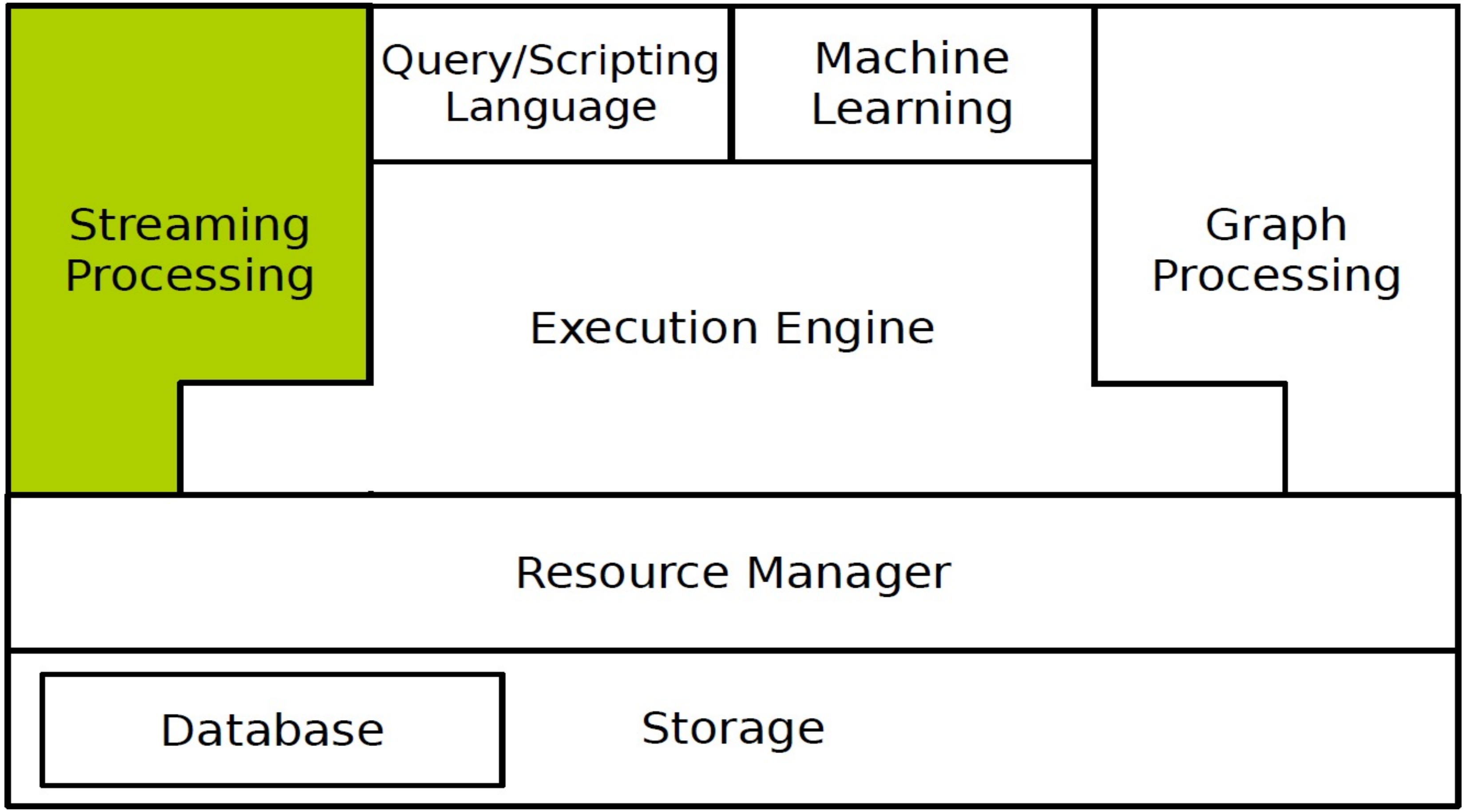
Maryam R.Aliabadi

mraiyata@cs.ubc.ca

Last Week‘s Review

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

Today’s Topics



Stream Processing

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

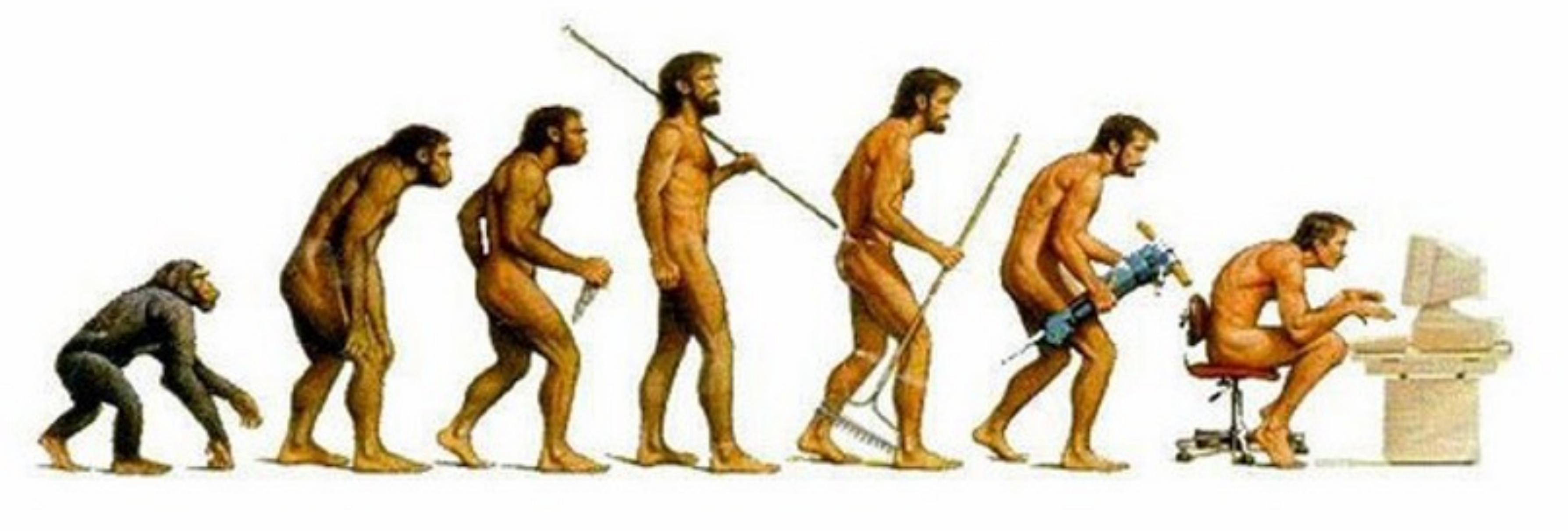
- The input data is unbounded.
  - A series of events, no predetermined beginning or end.
  - E.g., credit card transactions, clicks on a website, or sensor readings from IoT devices.

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.

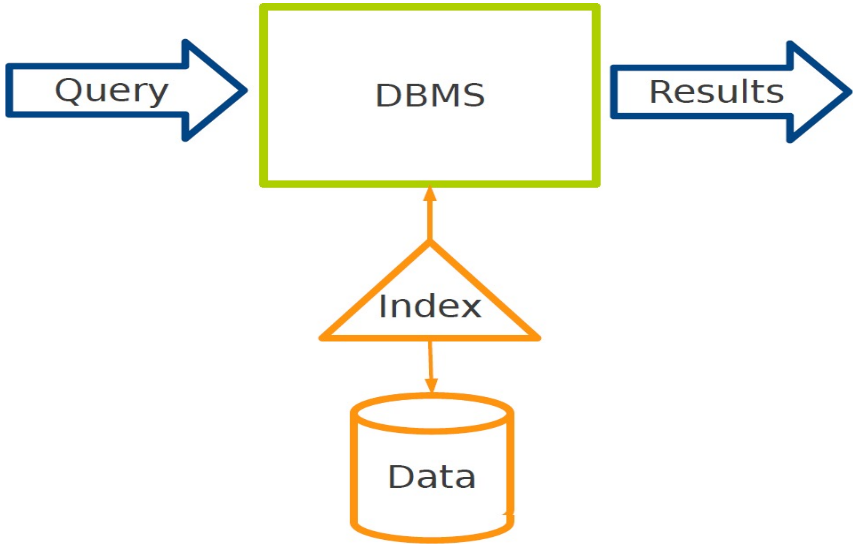
Data Stream Management Systems

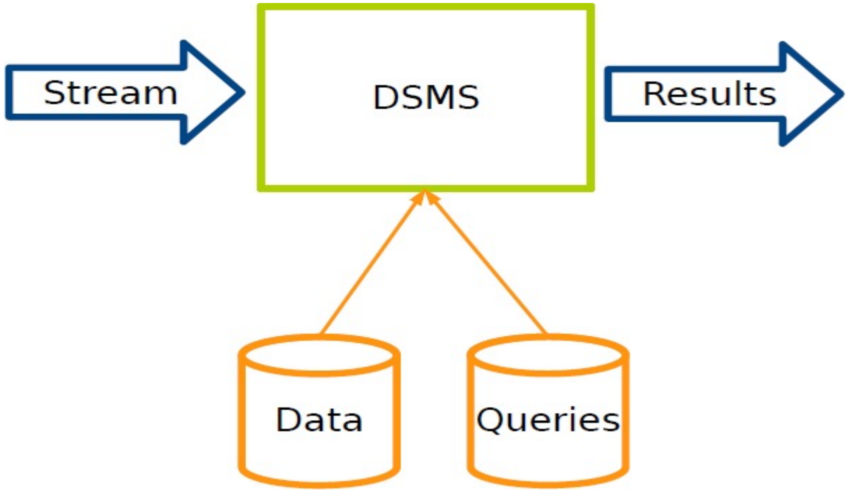
- An evolution of traditional data processing, as supported by DBMSs.



DBMS Vs. DSMS (1/3)

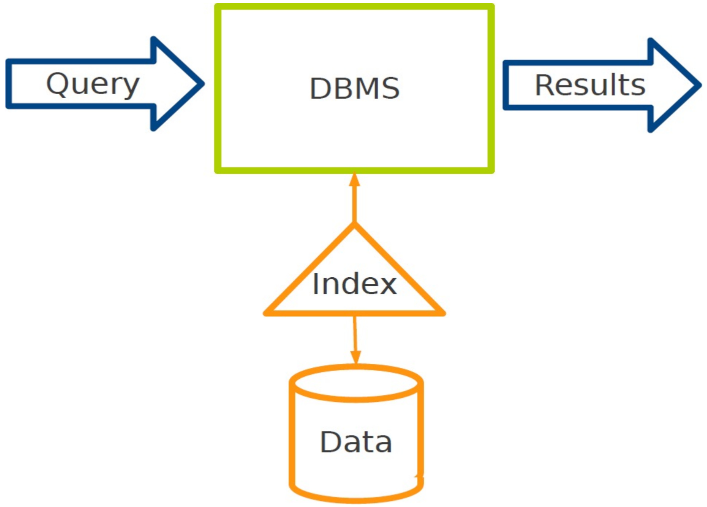
- DBMS: data-at-rest analytics
  - Store and index data before processing it.
  - Process data only when explicitly asked by the users.
- DSMS: data-in-motion analytics
  - Processing information as it flows, without storing them persistently.

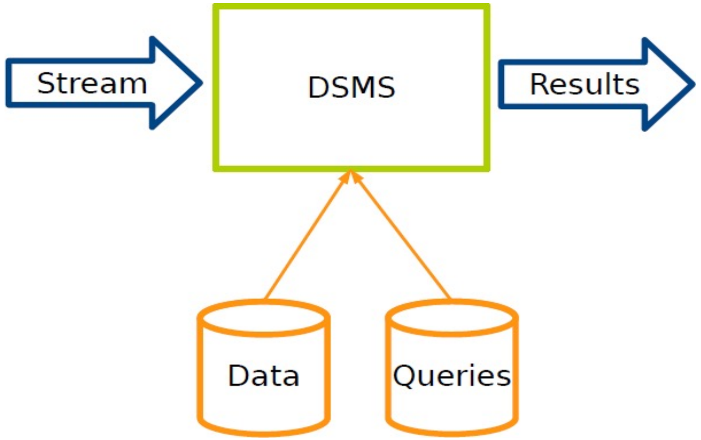




DBMS Vs. DSMS (2/3)

- 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.

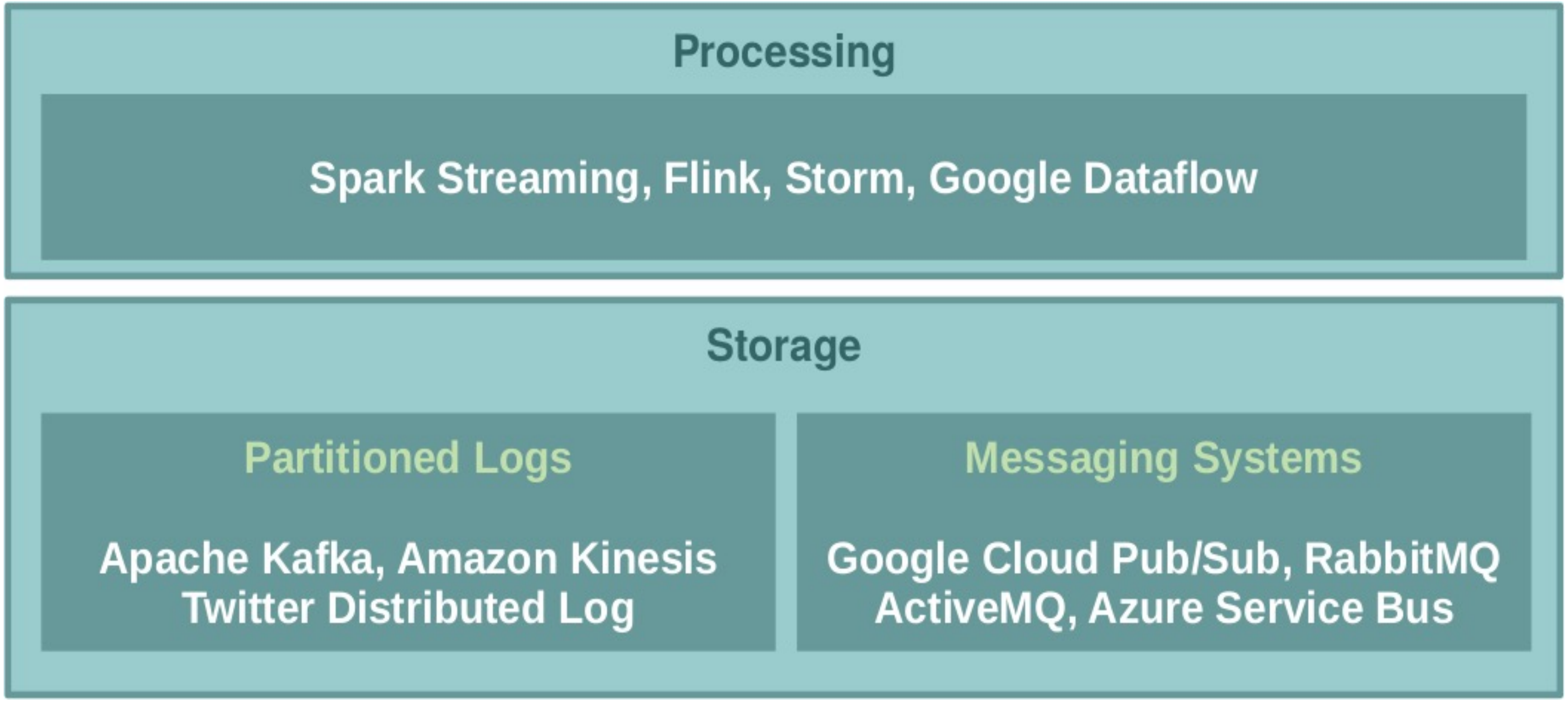




DBMS Vs. DSMS (3/3)

- 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



Processing

Spark Streaming, Flink, Storm, Google Dataflow

Storage

Partitioned Logs

Apache Kafka, Amazon Kinesis
Twitter Distributed Log

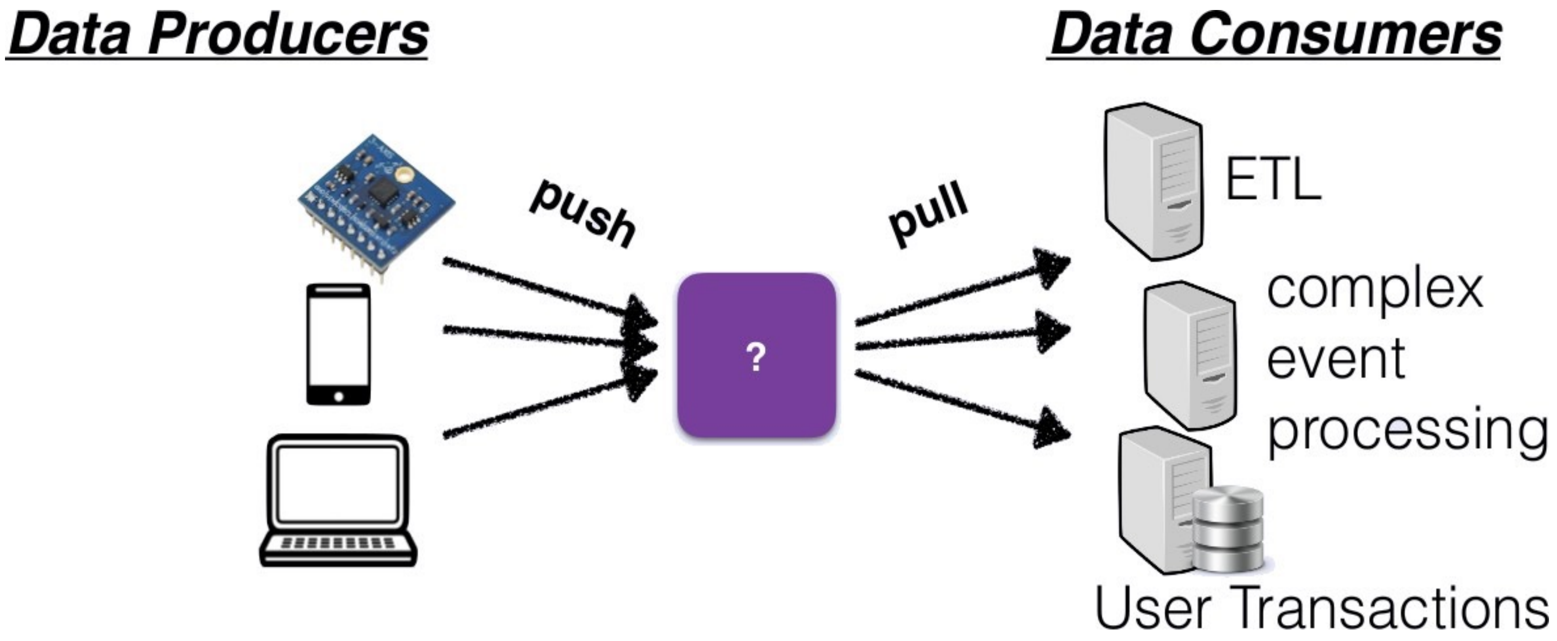
Messaging Systems

Google Cloud Pub/Sub, RabbitMQ
ActiveMQ, Azure Service Bus

Data Stream Storage

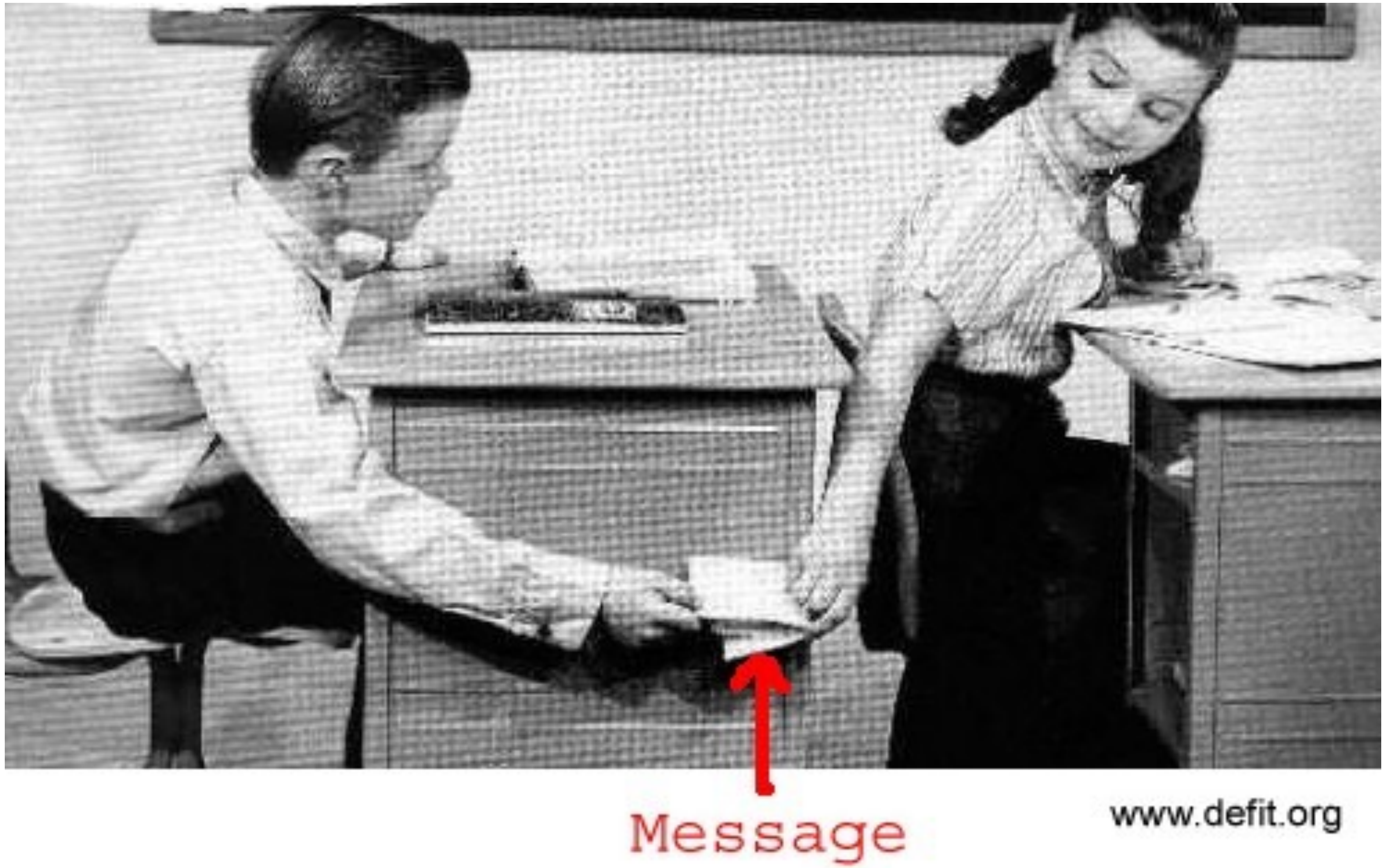
The Problem

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



Possible Solution

- Message systems



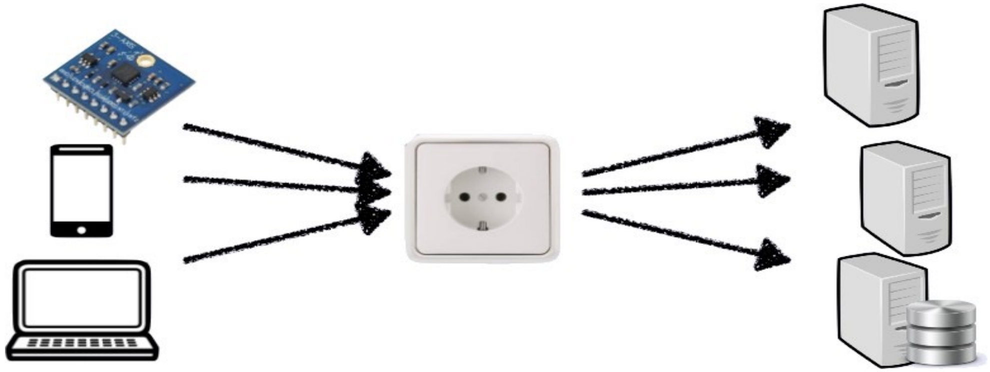
www.defit.org

What is a messaging system?

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

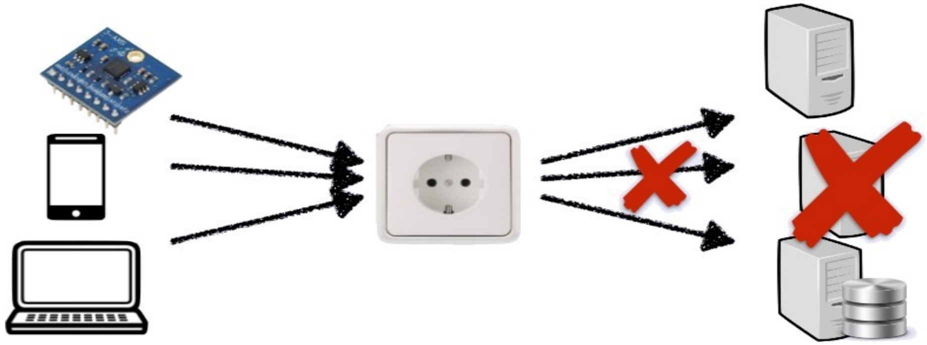
Direct messaging

- 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.

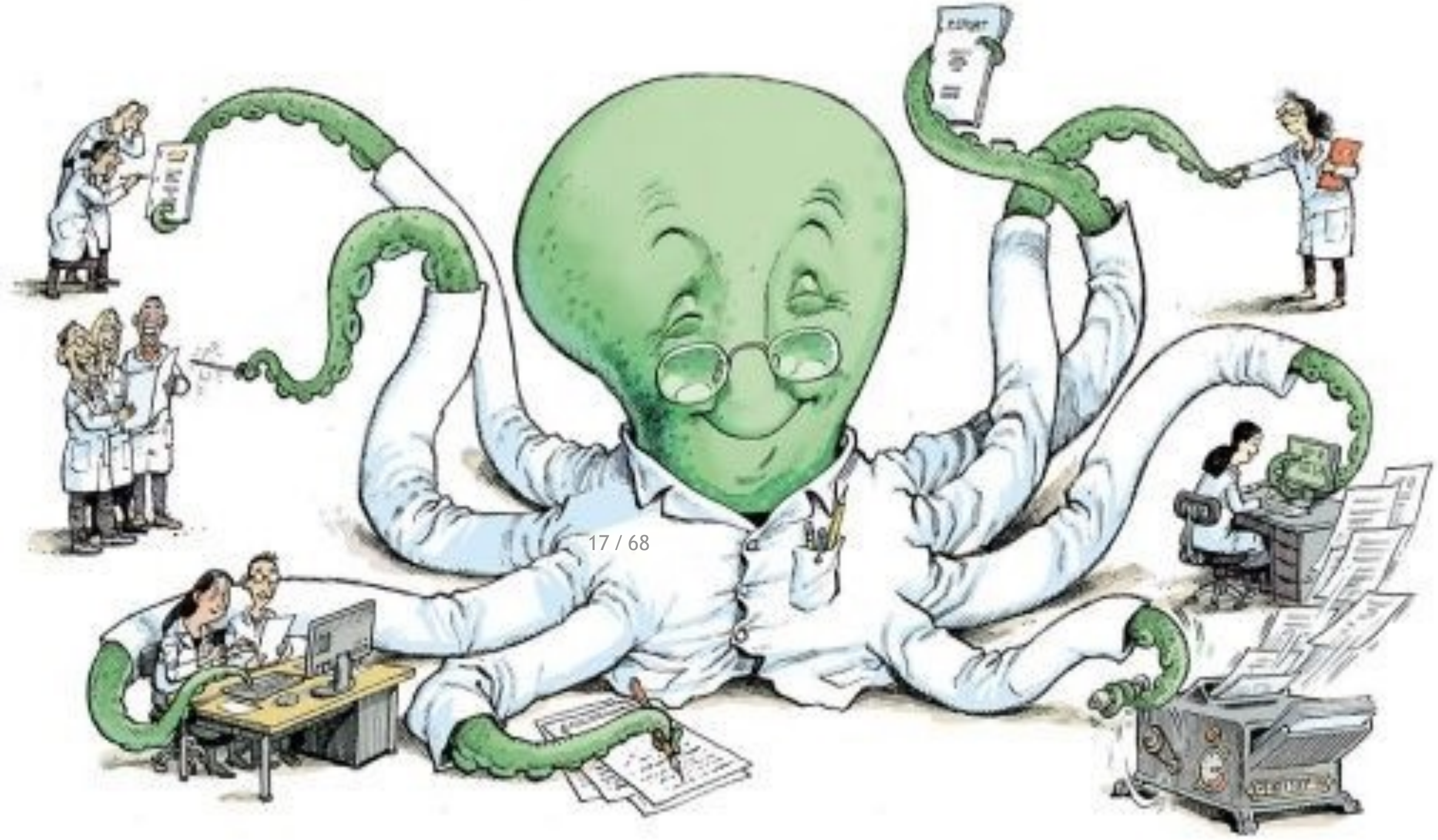


Direct messaging

- 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.



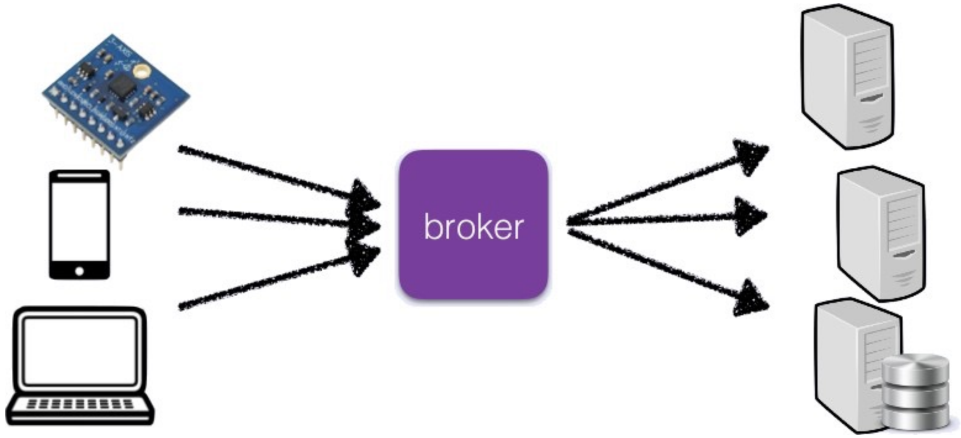
Message Broker



[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.



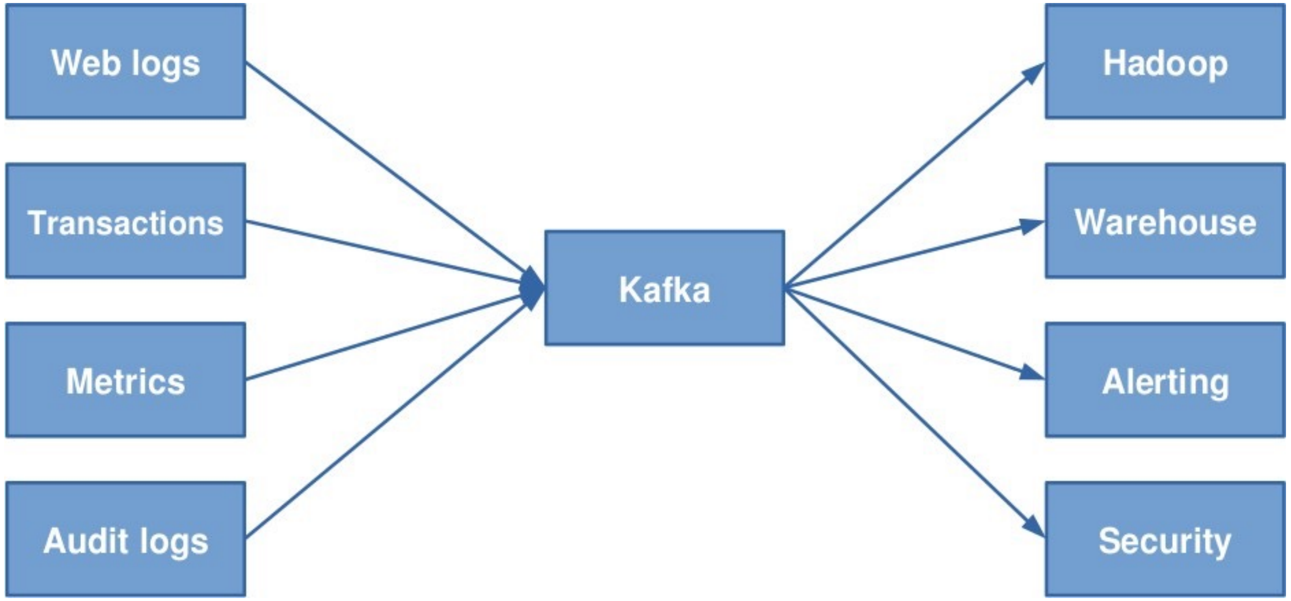
Partitioned Log

- 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.

Kafka: A Log-based Message Broker

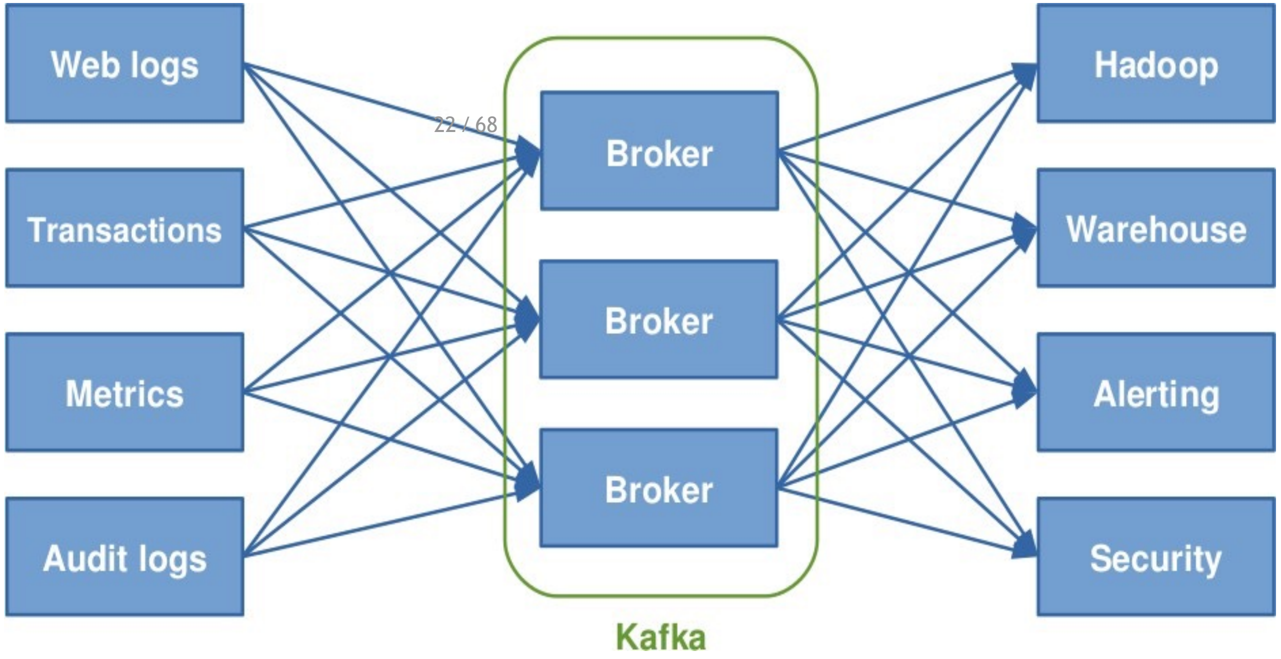
Kafka

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



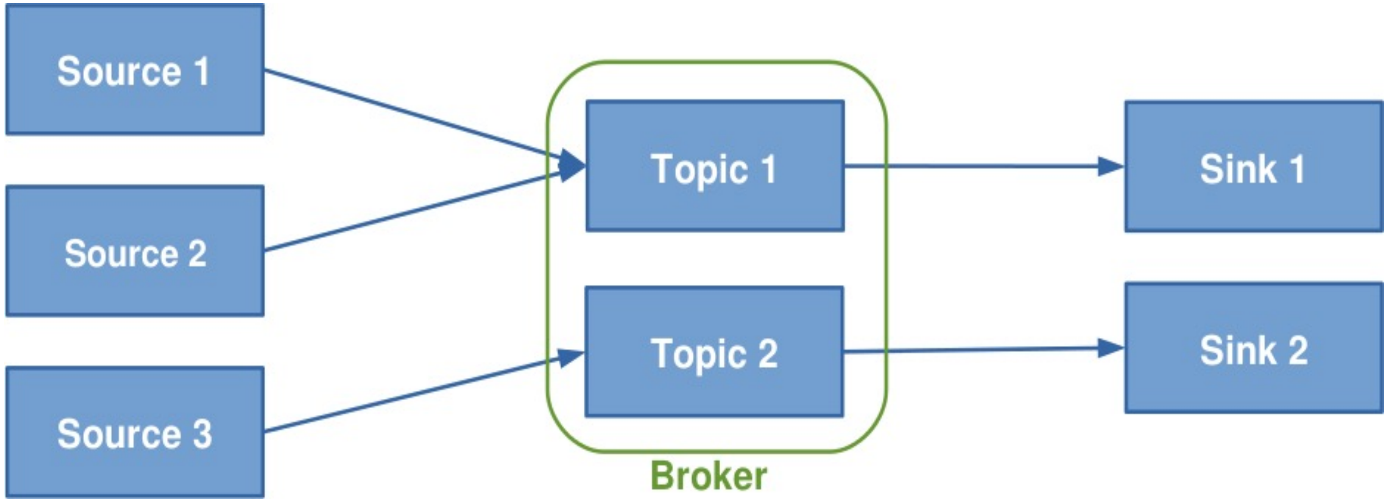
Kafka

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



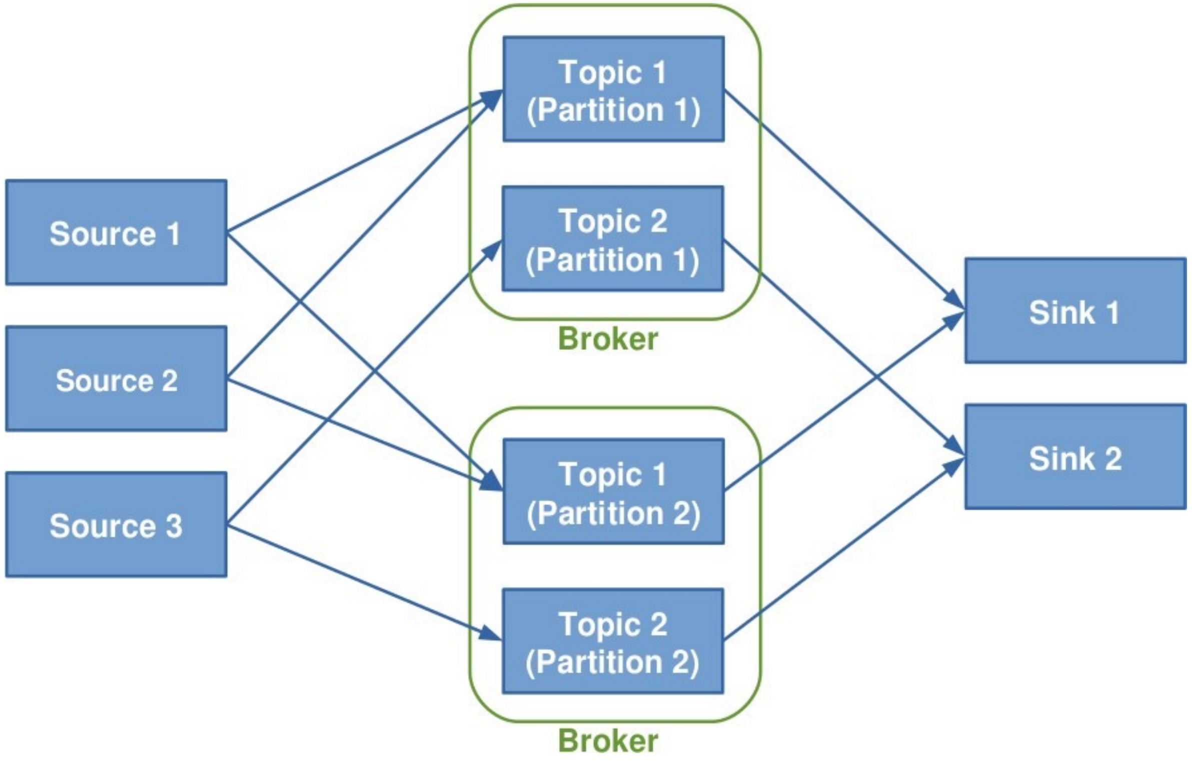
Kafka

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



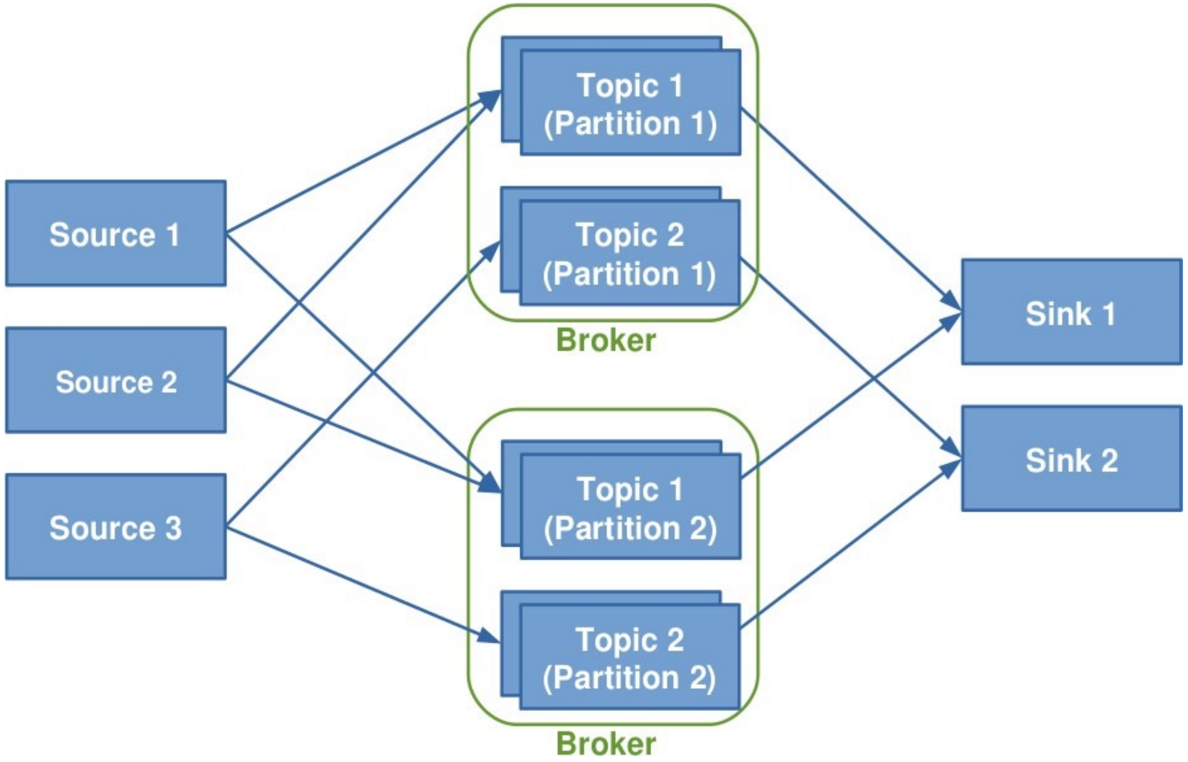
Kafka

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



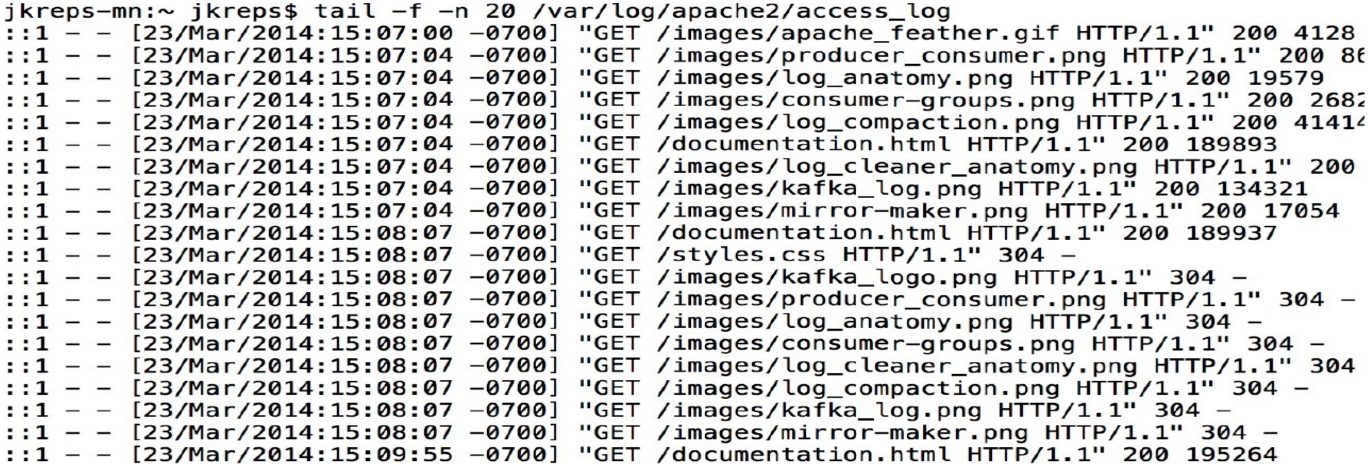
Kafka

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

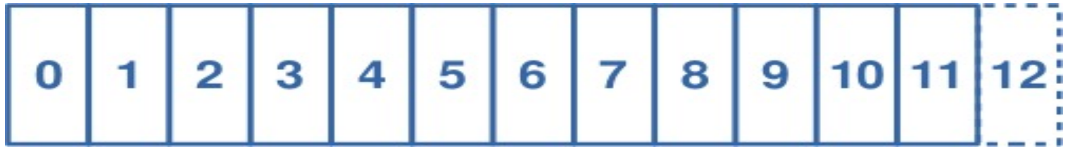


Logs, Topics and Partitions

- Kafka is about logs.
- Topics are queues: a stream of messages of a particular type

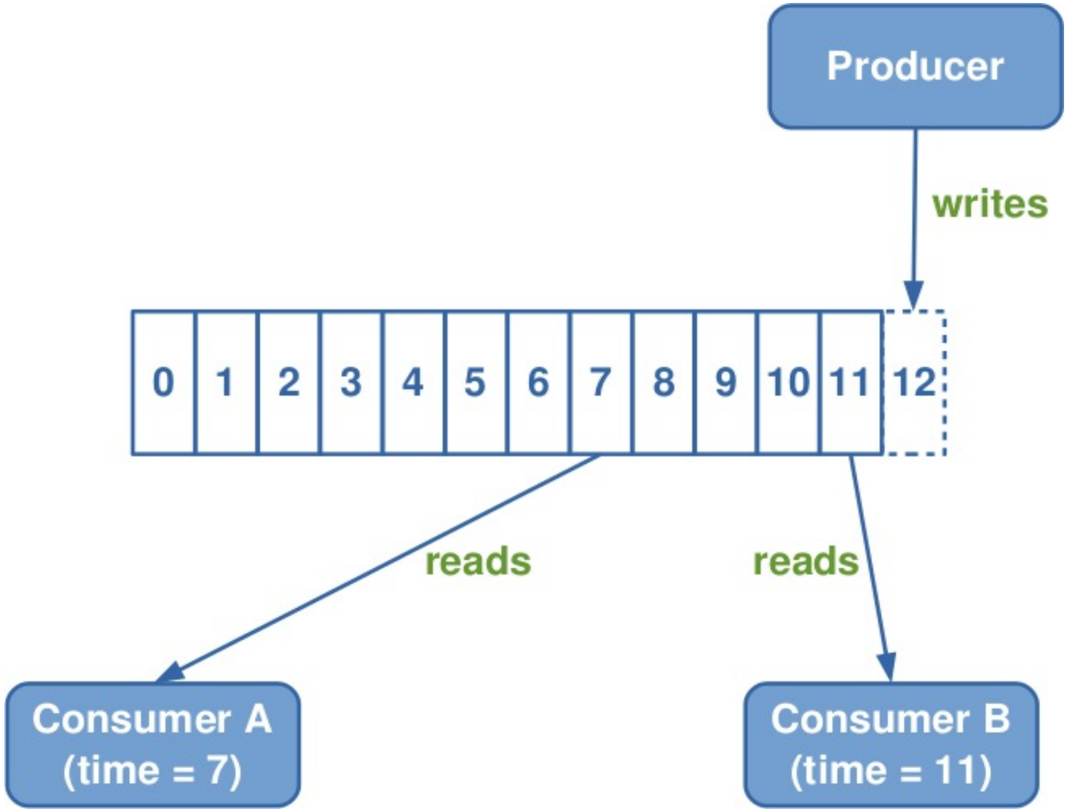


```
jkreps-mn:~ jkreps$ tail -f -n 20 /var/log/apache2/access_log
::1 - - [23/Mar/2014:15:07:00 -0700] "GET /images/apache_feather.gif HTTP/1.1" 200 4128
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/producer_consumer.png HTTP/1.1" 200 86
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_anatomy.png HTTP/1.1" 200 19579
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/consumer-groups.png HTTP/1.1" 200 2682
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_compaction.png HTTP/1.1" 200 41414
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /documentation.html HTTP/1.1" 200 189893
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 200
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/kafka_log.png HTTP/1.1" 200 134321
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/mirror-maker.png HTTP/1.1" 200 17054
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /documentation.html HTTP/1.1" 200 189937
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /styles.css HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_logo.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/producer_consumer.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_anatomy.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/consumer-groups.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 304
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_compaction.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_log.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/mirror-maker.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:09:55 -0700] "GET /documentation.html HTTP/1.1" 200 195264
```



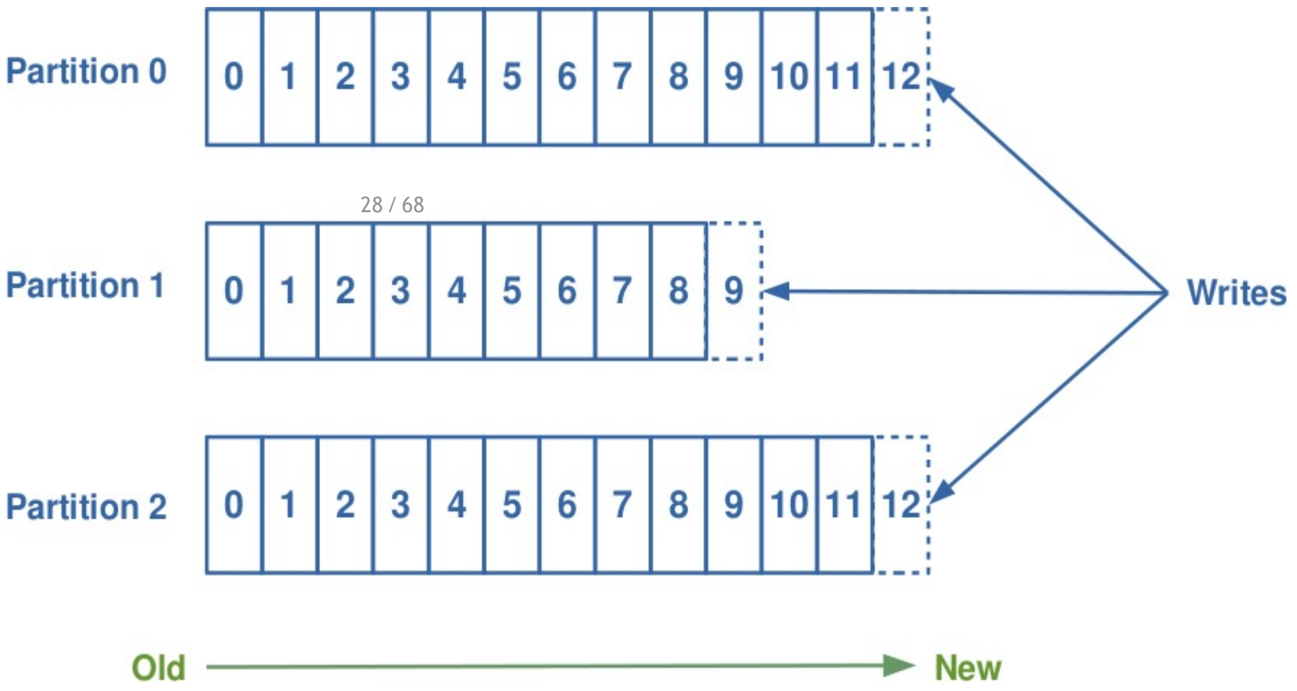
Logs, Topics and Partitions

- Each message is assigned a sequential id called an offset.



Logs, Topics and Partitions

- Topics are logical collections of partitions (the physical files).
  - Ordered
  - Append only
  - Immutable

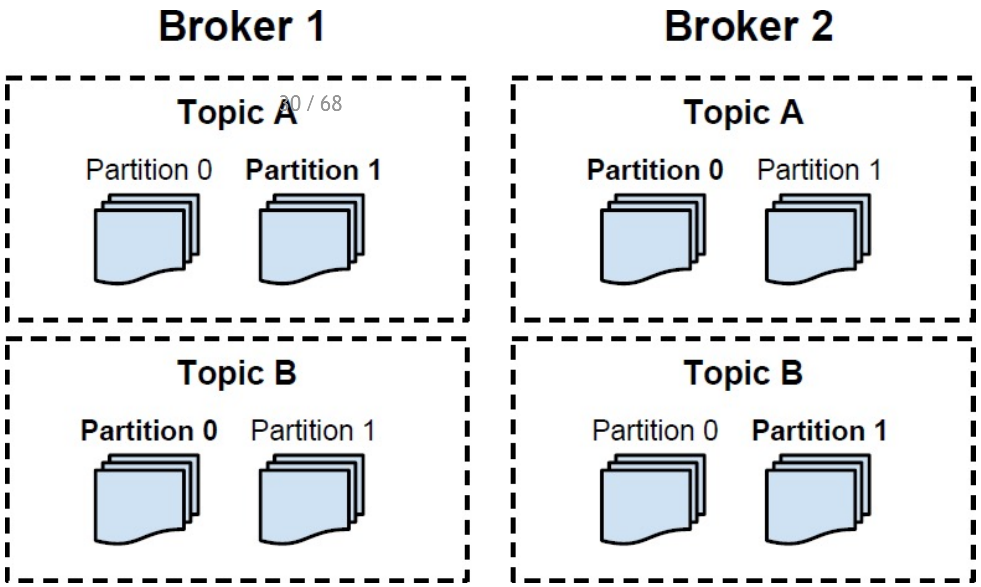


Logs, Topics and Partitions

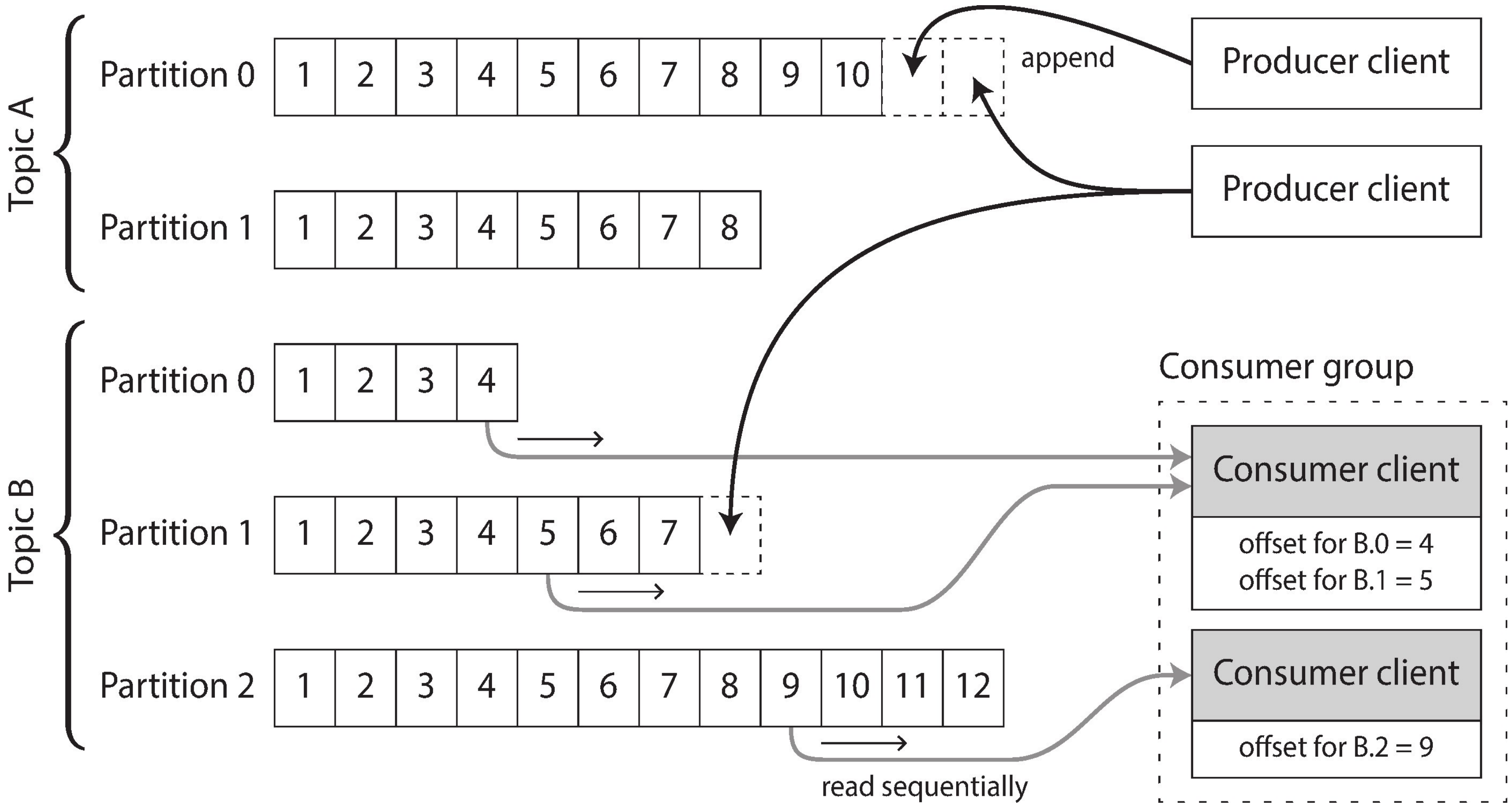
- Ordering is only guaranteed within a partition for a topic.
- Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
- A consumer instance sees messages in the order they are stored in the log.

Logs, Topics and Partitions

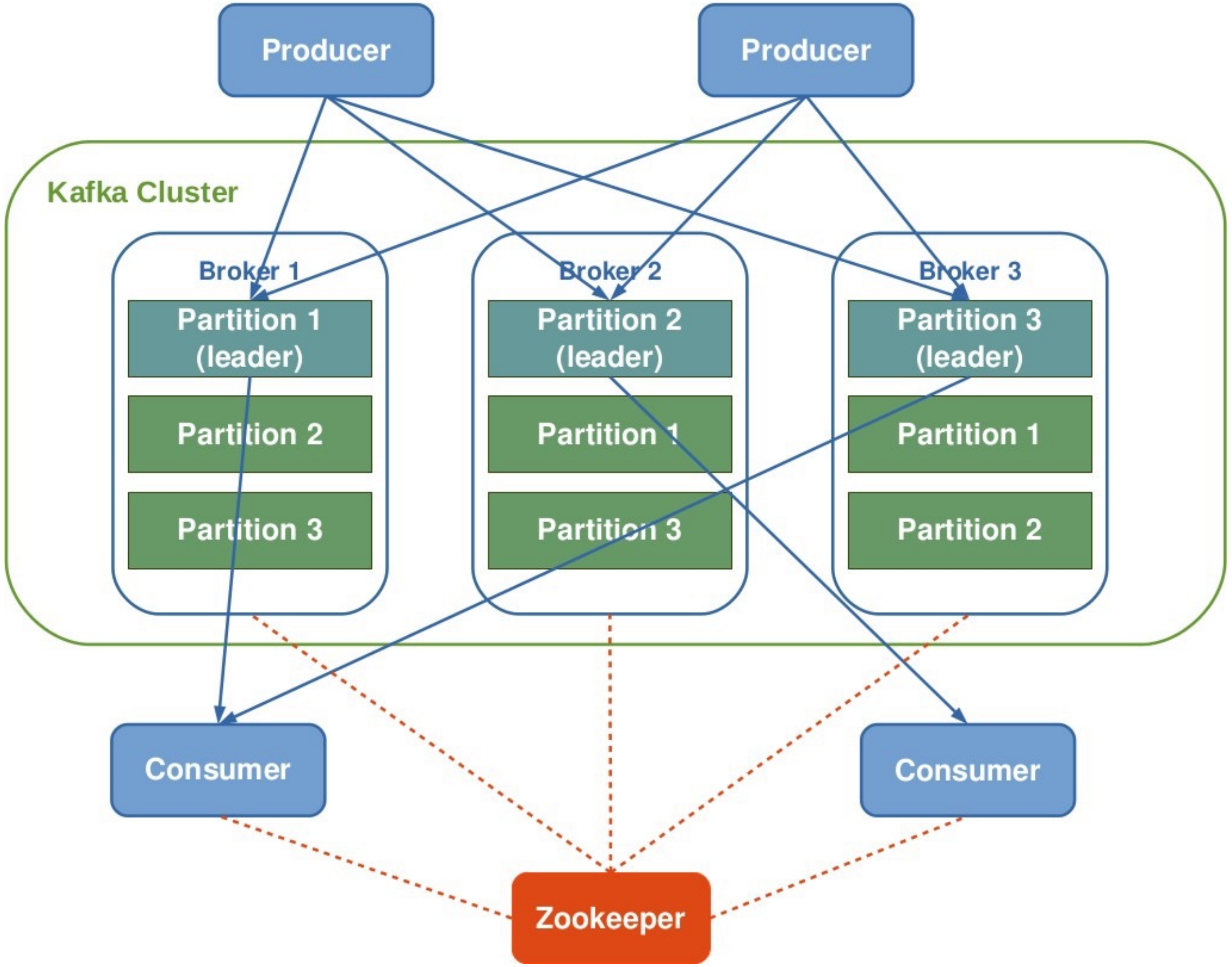
- Partitions of a topic are replicated: fault-tolerance
- A broker contains some of the partitions for a topic.
- One broker is the leader of a partition: all writes and reads must go to the leader.



Partition Logs



Kafka Architecture



Coordination

- Kafka uses Zookeeper for the following tasks:
- Detecting the addition and the removal of brokers and consumers.
- Keeping track of the consumed offset of each partition.

State in Kafka

- Brokers are sateless: 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).

Delivery Guarantees

- Kafka guarantees that messages from a single partition are delivered to a consumer in order.

- There is no guarantee on the ordering of messages coming from different partitions.

- Kafka only guarantees at-least-once delivery.

Start and Work with Kafka

```
# Start the ZooKeeper
zookeeper-server-start.sh config/zookeeper.properties
```

```
# Start the Kafka server
kafka-server-start.sh config/server.properties
```

```
# Create a topic, called "avg"
kafka-topics.sh --create --topic avg --bootstrap-server localhost:9092 --replication-factor 1
                             --partitions 1
```

```
# Produce messages and send them to the topic "avg"
kafka-console-producer.sh --topic avg --bootstrap-server localhost:9092
```

```
# Consume the messages sent to the topic "avg"
kafka-console-consumer.sh --topic avg --from-beginning --bootstrap-server localhost:9092
```

Data Stream Processing

Streaming Data

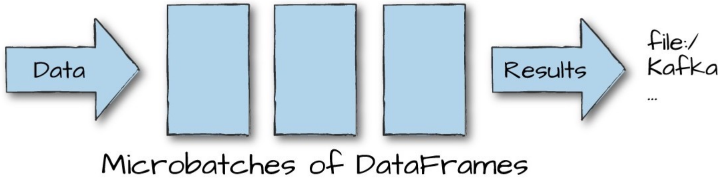
- Data stream is unbound data, which is broken into a sequence of individual tuples.
- A data tuple is the atomic data item in a data stream.
- Can be structured, semi-structured, and unstructured.

Streaming Data Processing Design Points

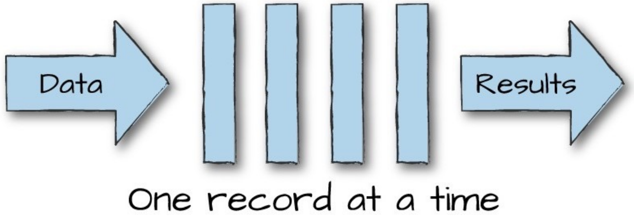
- Continuous vs. micro-batch processing
- Record-at-a-Time vs. declarative APIs
- Event time vs. processing time
- Windowing

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 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.

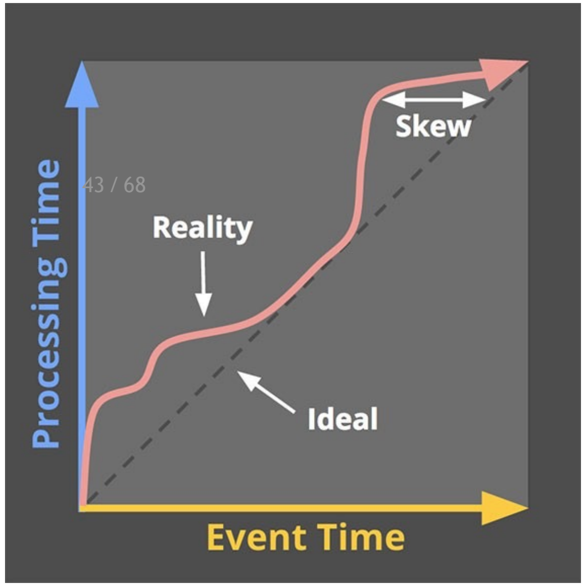
Event time Vs. Processing time

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

- Processing time: the time when the record is received at the streaming application.

Event time Vs. Processing time

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



[https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101]

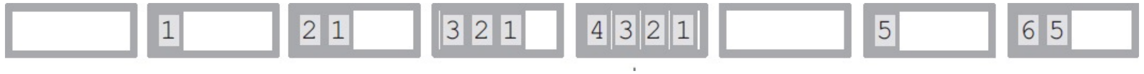
Windowing

- Window: a buffer associated with an input port to retain previously received tuples.
- Four different windowing management policies.
  - Count-based policy: the maximum number of tuples a window buffer can hold
  - Delta-based policy: a delta threshold in a tuple attribute
  - Punctuation-based policy: a punctuation is received

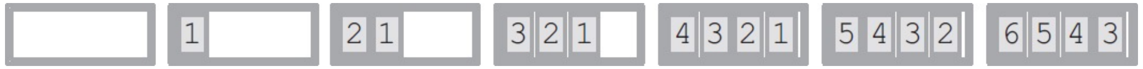
Windowing

- Two types of windows: tumbling and sliding

- Tumbling window: supports batch operations.
  - When the buffer fills up, all the tuples are evicted.



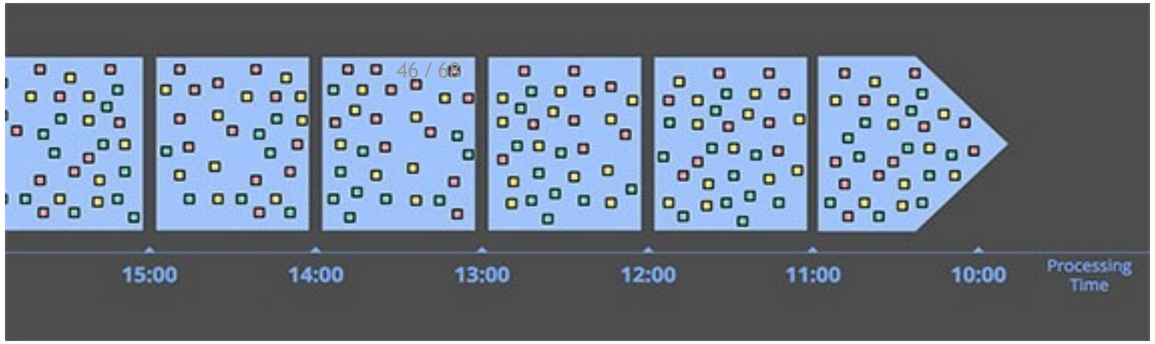
- Sliding window: supports incremental operations.
  - When the buffer fills up, older tuples are evicted.



Windowing by Processing Time

- The system buffers up incoming data into windows until some amount of processing time has passed.

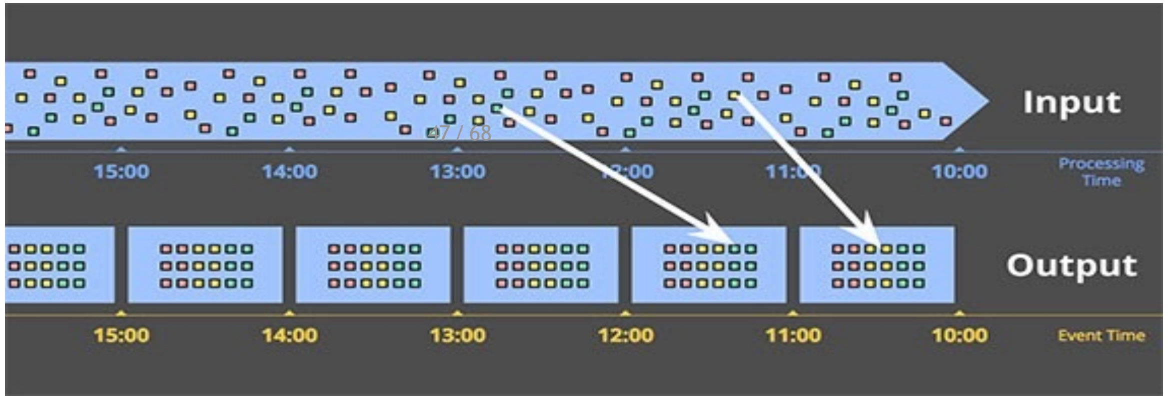
- E.g., five-minute fixed windows



[https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101]

Windowing by Event Time

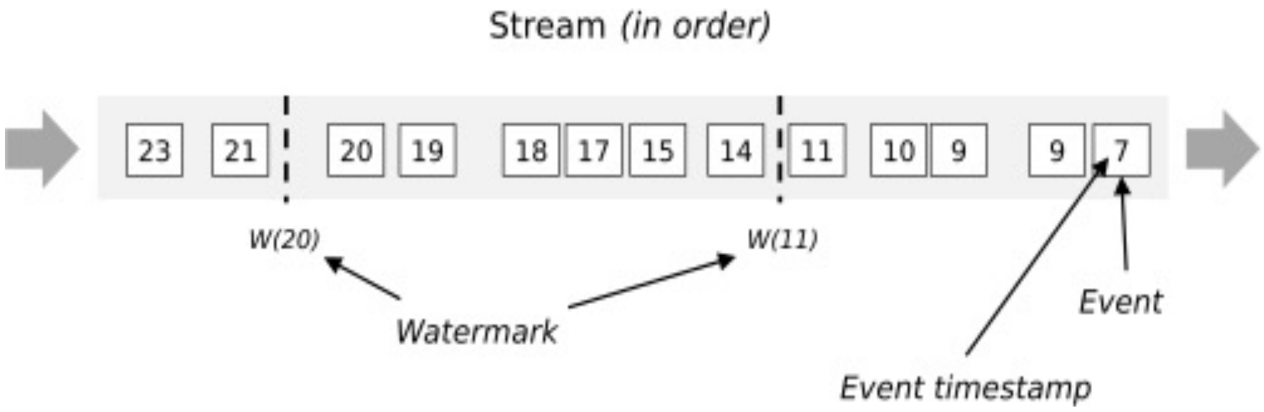
- Reflect the times at which events actually happened.
- Handling out-of-order evnets.



[https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101]

Windowing by Event Time - Watermark

- Watermarking helps a stream processing system to deal with lateness.
- Watermarks flow as part of the data stream and carry a timestamp t.
- A watermark is a threshold to specify how long the system waits for late events.
- Streaming systems uses watermarks to measure progress in event time.



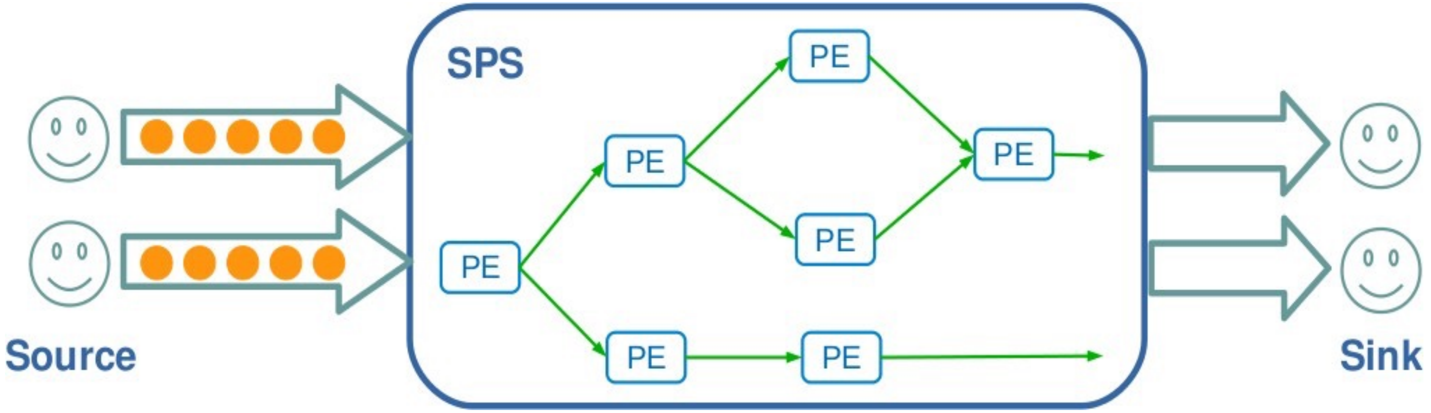
Windowing by Event Time - Watermark

- A W(t) declares that event time has reached time t in that stream
  - There should be no more elements from the stream with a timestamp $t' \leq t$.
- It is possible that certain elements will violate the watermark condition.
  - After the W(t) has occurred, more elements with timestamp $t' \leq t$ will occur.
- If an arriving event lies within the watermark, it gets used to update a query.
- Streaming programs may explicitly expect some late elements.

Streaming Data Processing Model

Streaming Data Processing

- The tuples are processed by the application’s operators or processing element (PE).
- A PE is the basic functional unit in an application.
  - A PE processes input tuples, applies a function, and outputs tuples.
  - A set of PEs and stream connections, organized into a data flow graph.



PEs State

- A PE can either maintain internal state across tuples while processing them, or process tuples independently of each other.
- Stateful vs. stateless tasks
  - Stateless tasks: do not maintain state and process each tuple independently of prior history, or even from the order of arrival of tuples.
  - Easily parallelized.
  - No synchronization.
  - Restart upon failures without the need of any recovery procedure

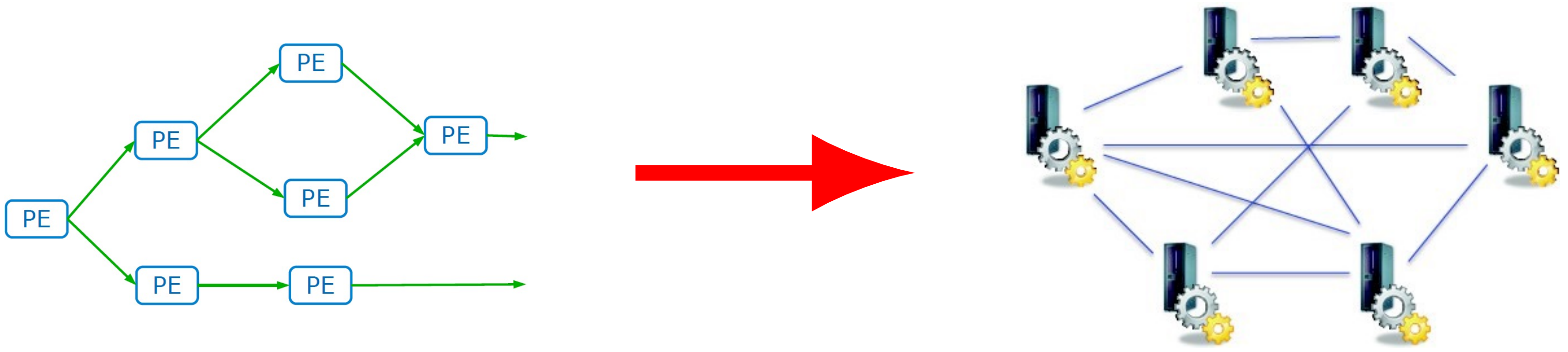
Job and Job Management

- At runtime, an application is represented by one or more jobs.
- Jobs are deployed as a collection of PEs.
- Job management component must identify and track individual PEs, the jobs they belong to, and associate them with the user that instantiated them.

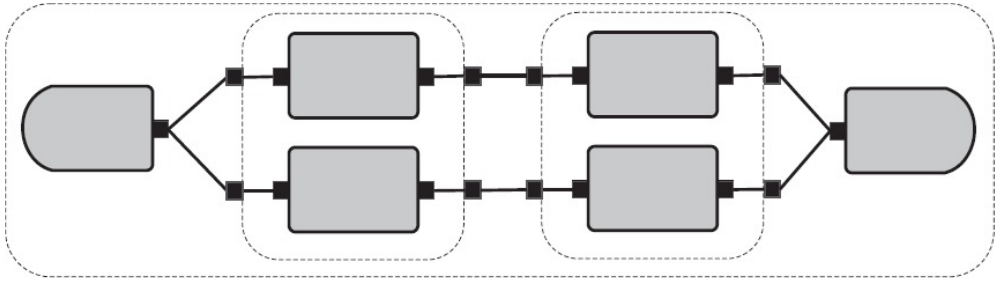
Logical Vs. Physical Plans

- Logical plan: a data flow graph, where the vertices correspond to PEs, and the edges to stream connections.

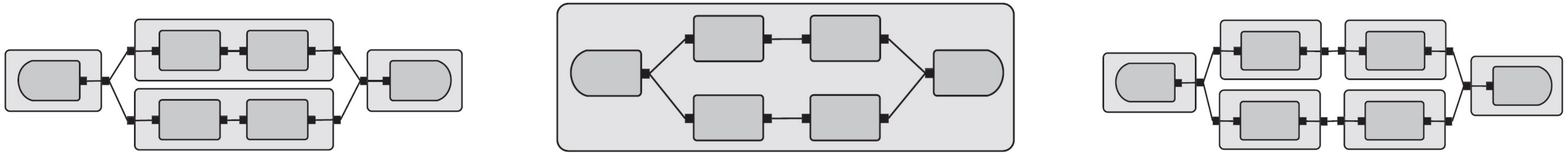
- Physical plan: a data flow graph, where the vertices correspond to OS processes, and the edges to transport connections.



Logical Vs. Physical Plans



Logical plan



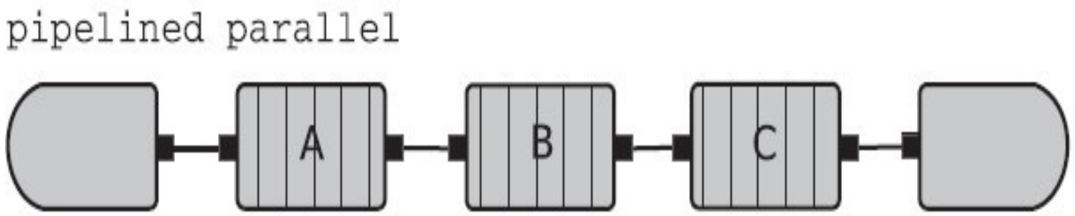
Different physical plans

Parallelization

- How to scale with increasing the number queries and the rate of incoming events?
- Three forms of parallelisms.
  - Pipelined parallelism
  - Task parallelism
  - Data parallelism

Pipeline Parallelism

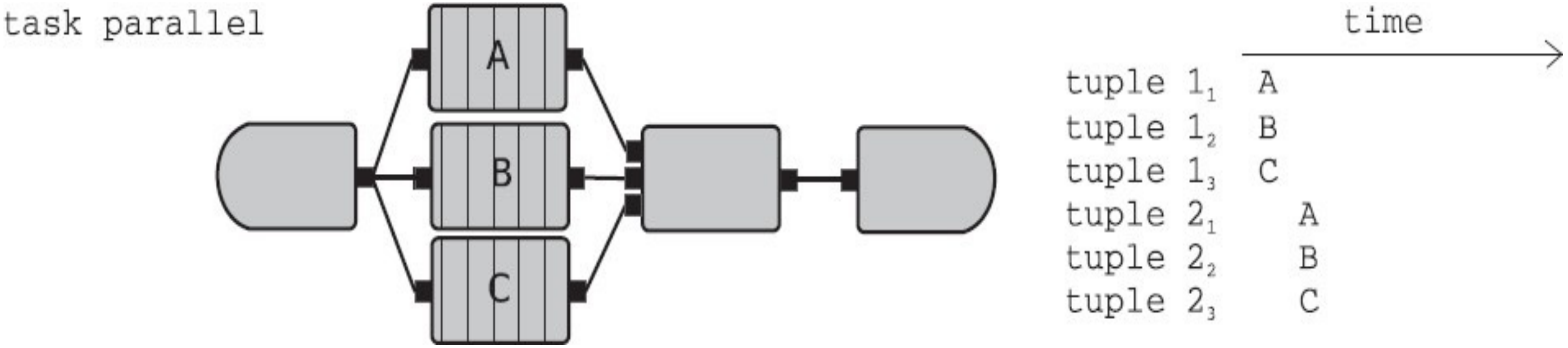
- Sequential stages of a computation execute concurrently for different data items.



```
                  time
                ────────────────→
tuple 1   A B C
tuple 2     A B C
tuple 3       A B C
tuple 4         A B C
tuple 5           A B C
tuple 6             A B C
```

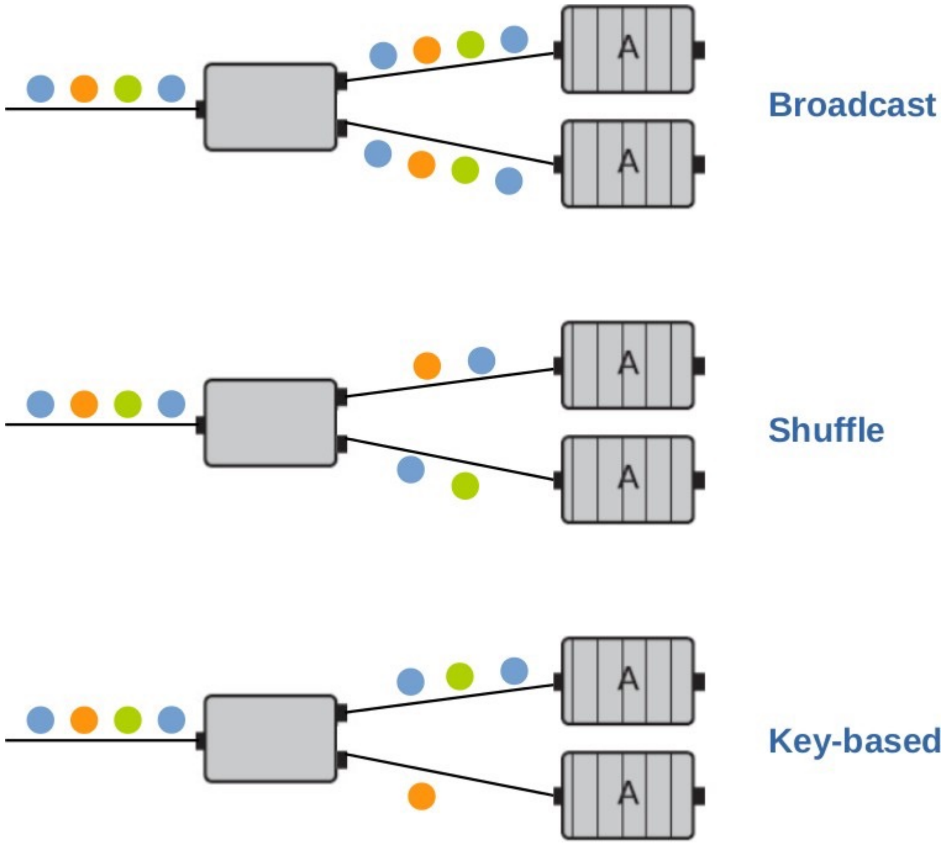
Task Parallelism

- Independent processing stages of a larger computation are executed concurrently on the same or distinct data items.



Data Parallelism

- How to allocate data items to each computation instance?



Broadcast

Shuffle

Key-based

Recap

- Messaging system and partitioned logs
- Decoupling producers and consumers
- Kafka: Distributed, topic oriented, partitioned, replicated log service
- Data stream, unbounded data, tuples
- Event-time vs. processing time
- Micro-batch vs. continues processing (windowing)
- PEs and dataflow
- Stateless vs. Stateful PEs

Next Topic:
Spark Streaming