Large-Scale Data Management and Distributed Systems

IV. Stream Processing

Vania Marangozova

Vania.Marangozova@imag.fr

2025-2026

References

- Lecture notes of V.Leroy
- Lecture notes of T.Ropars
- *Designing Data-Intensive Applications* by Martin Kleppmann
  - Chapter 11

Why Stream Processing ?

- Batch Processing
  - Data are stored in files
  - Process the **whole** data at once
  - Examples: Hadoop MapReduce, Spark, etc
- In many use-cases, **new data are generated continuously**
  - Data from sensors
  - Data from social networks
  - Web traffic
  - Etc.

=> Applications need real-time processing

A little detour...

- Businesses have not only their OLTP (Online Transaction Processing) systems, but also maintain Data Warehouses
- Data Warehouse = data system used for data analytics
  - How much have the PPP store sold this month ?
  - Which product is the most popular ?
  - ...
- OLAP = Online Analytic Processing Systems (OLAP)
- Access pattern
  - Scan over a large number of records
  - Compute statistics
- Updates hourly/daily/weekly...

Real-time Processing

In many cases, data should be processed **as early** as possible:

- Detecting fraudulent behavior
    - Log analysis
    - Access filtering
    - …
- Identifying malfunctioning systems
    - Monitoring information about crashes, non valid values, …
- Monitoring trends
    - social networks
    - system load
    - …

Adapting batch processing systems?

- Processing all the data of the day at the end of each day ?
  - High latency ☹
- How can we process data more frequently ?
  - Use stream processing/engines

Stream vs Batch processing

- Batch processing
  - Good for analyzing a static dataset
  - Focuses on **throughput**
  - Allows running **complex** analysis requiring multiple iterations on the data

- Stream processing
  - Good to analyze ***live*** data
  - Continuously updates results based on new data
  - Focuses on **latency** (between data production and update of the results)
  - Processes data once

Stream Processing Computations

Typical stream analytics

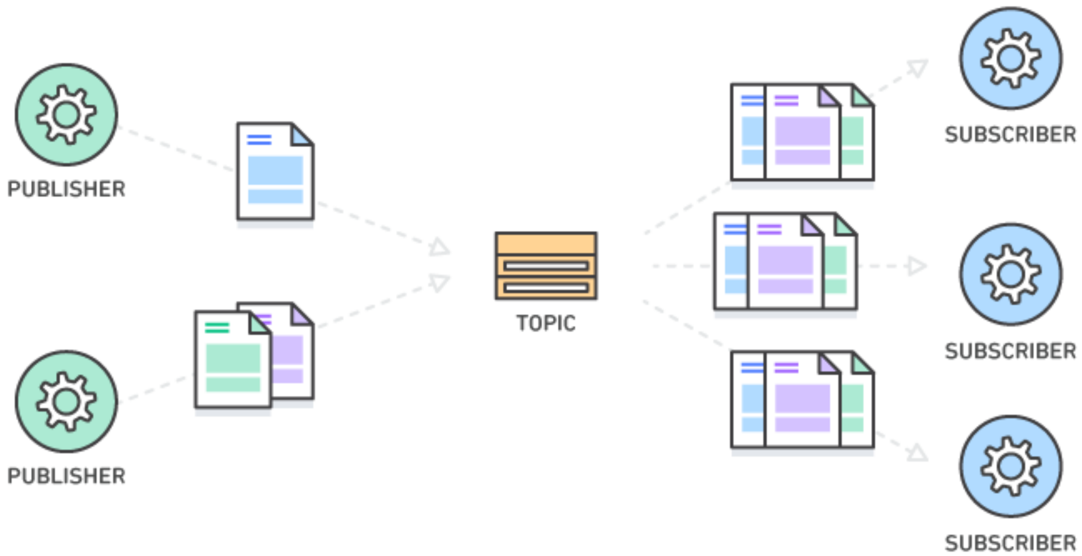
- Measuring event rates
- Computing rolling statistics (average, histograms, etc)
- Comparing statistics to previous values (detecting trends)
- Sampling data
- Filtering data
- Applying basic machine learning algorithms

Aspects of the "How" Question

- How to transmit data from the data sources (producers) to the data analyzers (consumers)?
- How to process events in a distributed way?
- How to deal with failures?
- How to reason about time?
- How to maintain a state over time?

Stream Processing Architecture
(Element of)

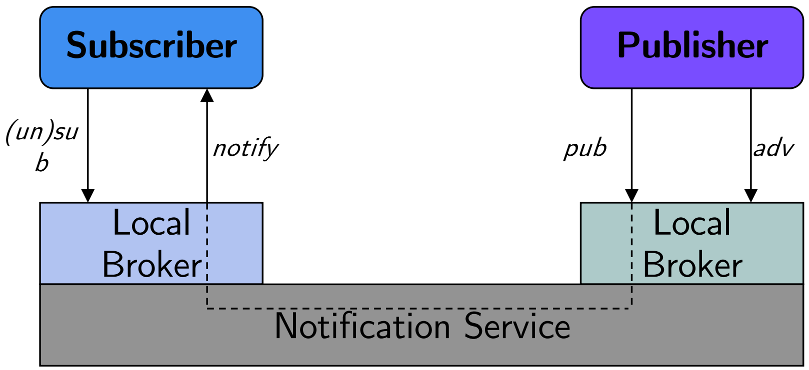
- Generalization of the **publish-subscribe** communication paradigm

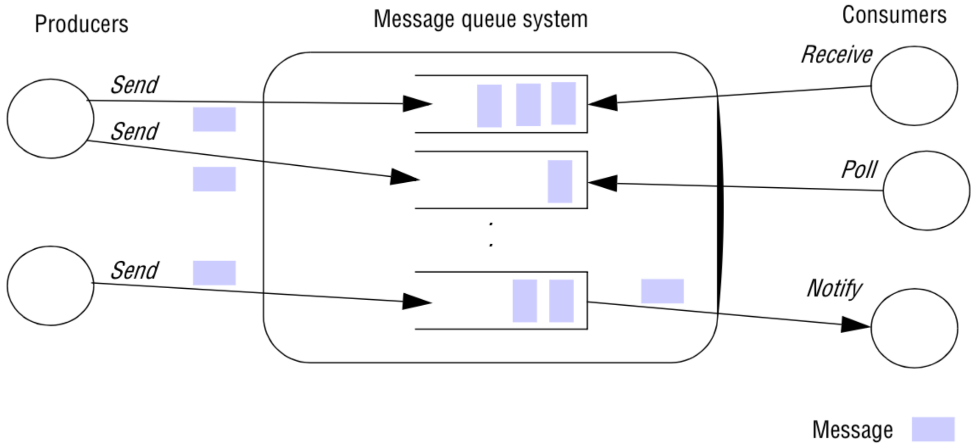


From https://aws.amazon.com/fr/what-is/pub-sub-messaging/

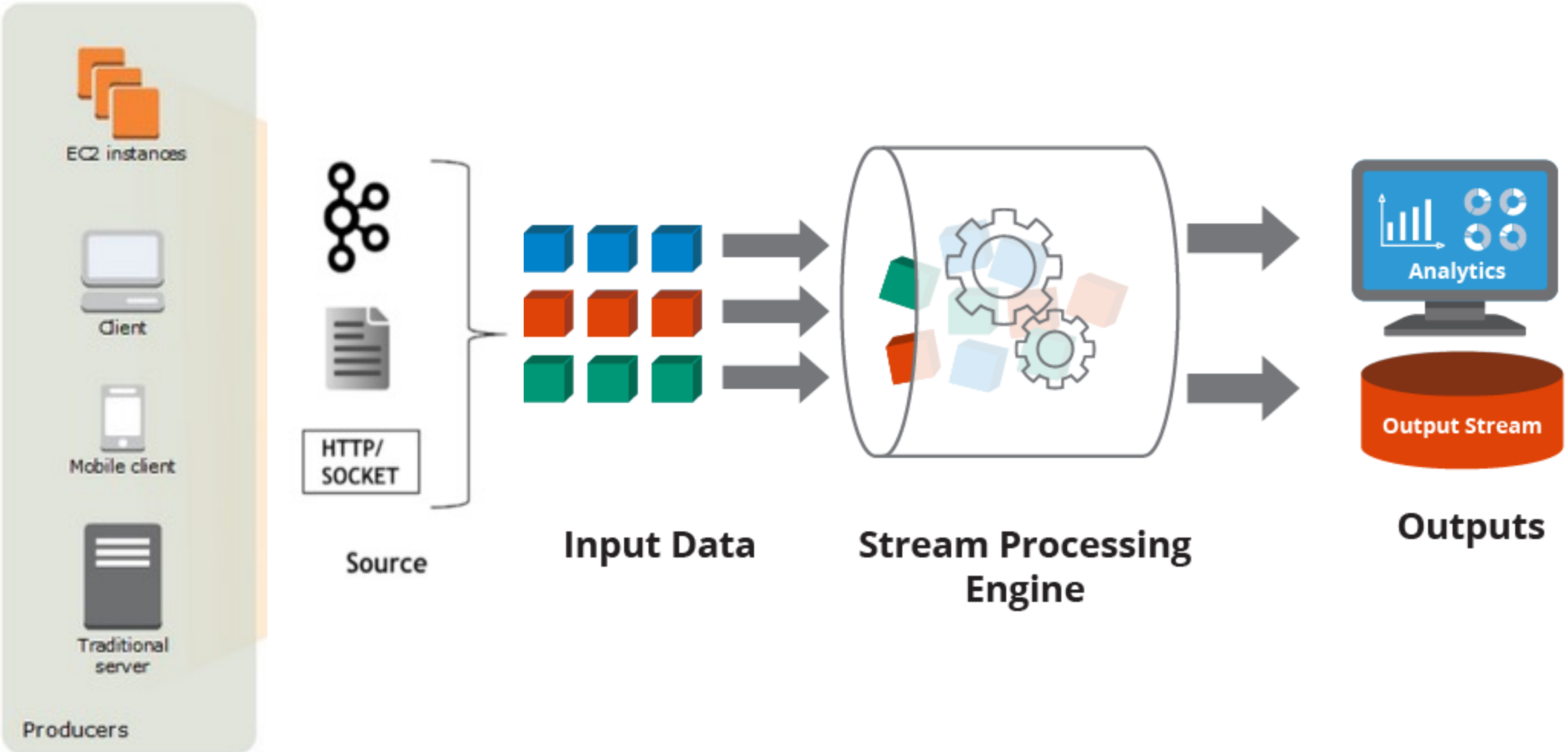
Publish-Subscribe

Broker **Messages**





Stream Processing Engines



Challenges

- Routing messages
  - Some consumers are only interested in some messages
  - Some messages are useful for multiple consumers
- Performance
  - Amount of produced data might be huge
  - Data might me produced faster than they are processed
- Fault tolerance
  - Clients might connect/disconnect at any time
  - The building blocks of the system (message-oriented middleware, MOM) may fail

What Data to Consider for Analysis?

- Messages, *in a standard context*, are transient
  - No permanent trace
  - Even if written to disk, quickly deleted because of the inherent logic and of the data volume (e.g. network packets)
  - **if a new consumer joins, it will analyze only recent data**
- Data in databases and files is persistent
  - everything written is permanently recorded, until explicitly deleted !
  - **if a new request is done, it will analyze the same data**

Can we not have a hybrid, combining the durable storage approach of databases with the low-latency notification facilities of messaging?

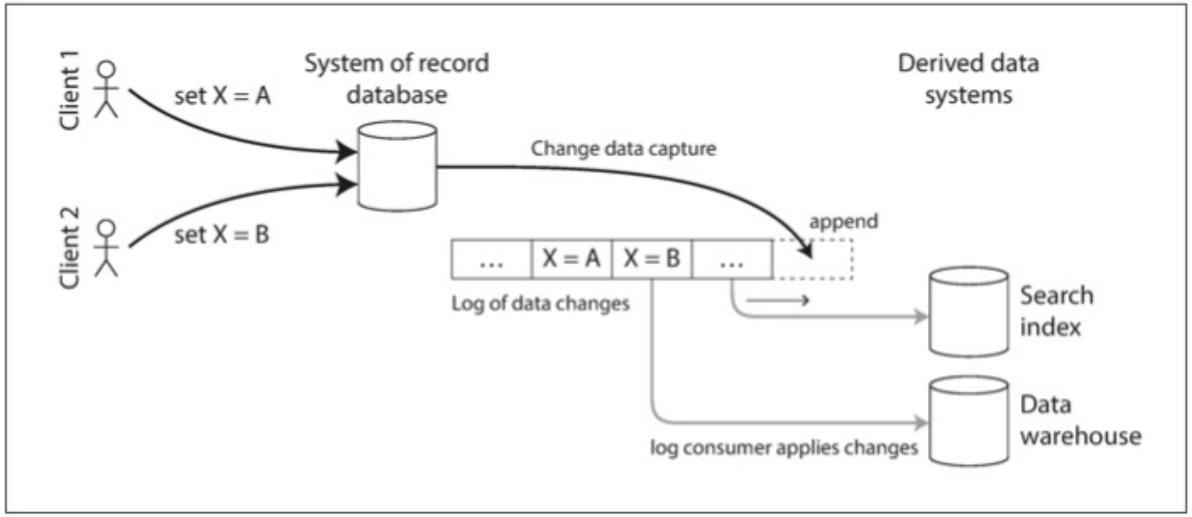
Log-based Message Broker

Main principles

- Maintain a log of all the messages received
  - Append-only sequence of records on disk
- Each record is identified with a sequence number
- The offset of each client in the log can be stored

Existing systems

- Apache Kafka
- Amazon Kinesis Data Streams



Kafka

https://kafka.apache.org/

- Originally developed at LinkedIn
- Open-source
- Used by many companies

Kafka Main Principles

A partitioned log

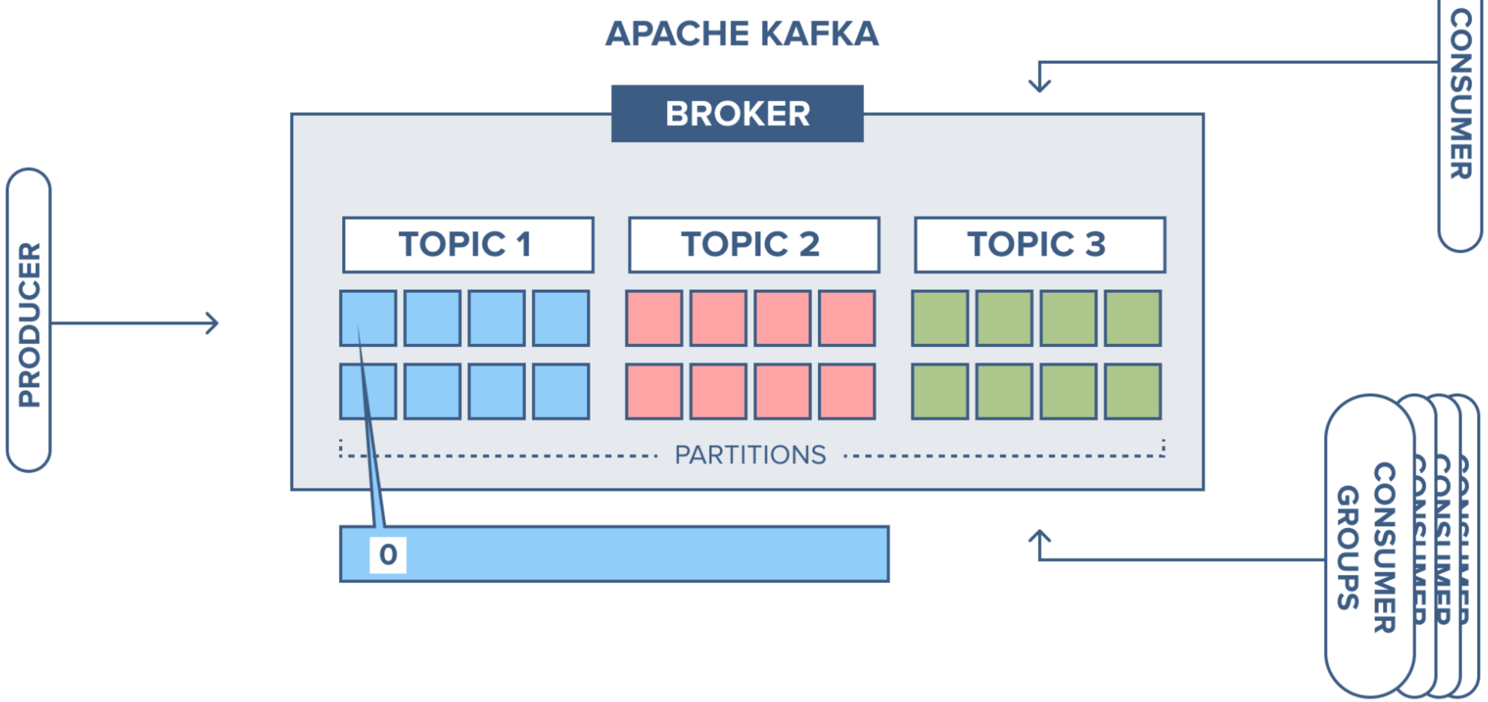
The log is divided into multiple partitions

- Each partition has its own monotonically increasing sequence number
- Partitions can be hosted on different machines
- There is no global order among partitions

Advantages of logs

- Old records can be replayed
- Data are buffered in the log
  - Deal with the case where the consumers are slower than the producers

Kafka Communication Abstractions



Kafka Cluster

Kafka runs multiple brokers, in a cluster

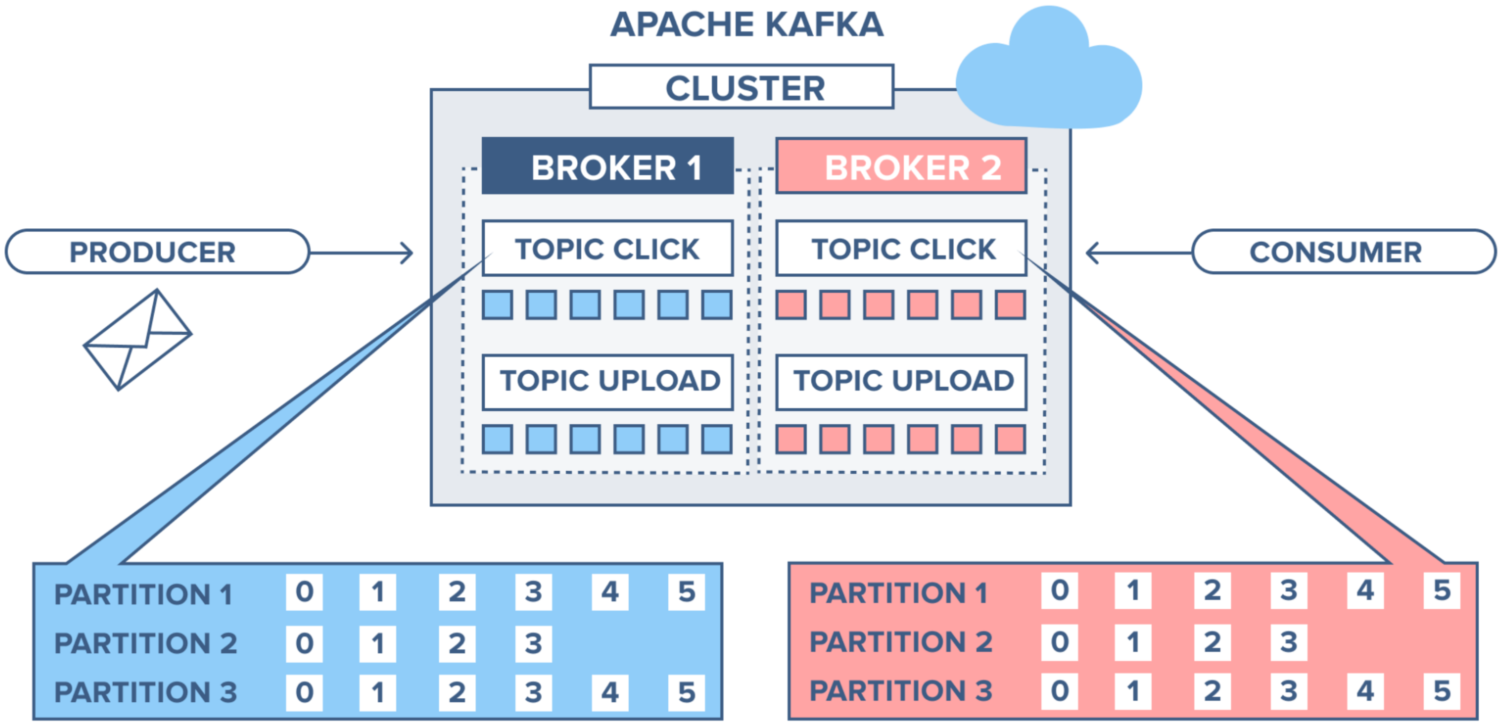
Among brokers, one broker is the *controller*

- For broker management (add brokers if needed)
- Assigns partitions to brokers i.e. which broker manages which partition

For a partition

- One broker is the leader
- Other brokers serve for fault tolerance and are updated by order of the leader

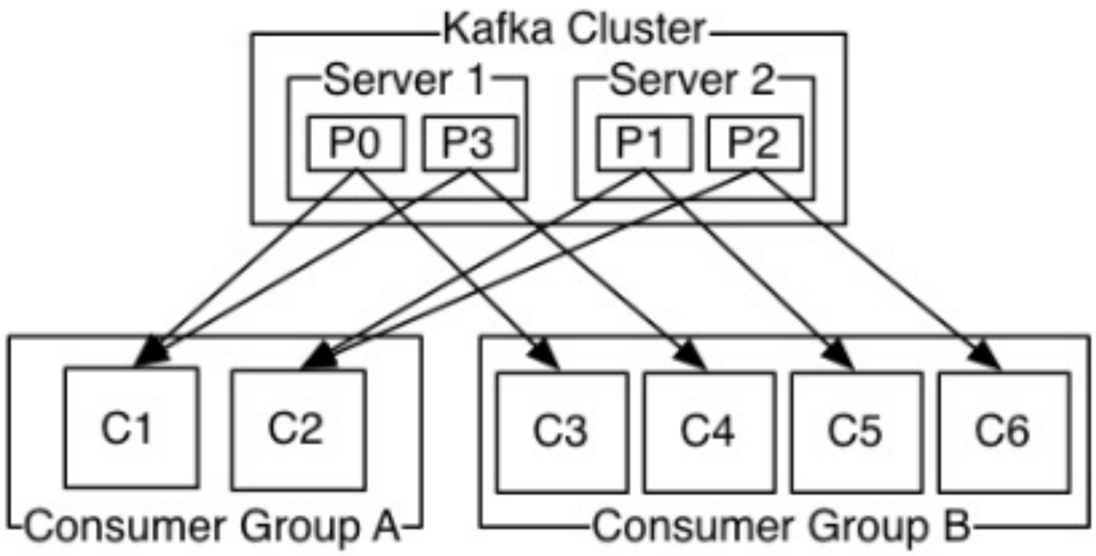
Kafka Cluster



Kafka Consumer Groups

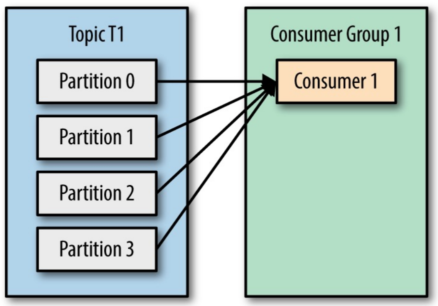
source https://kafka.apache.org/documentation/#gettingStarted
https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html

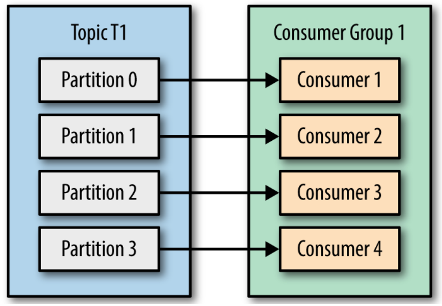
- Load balancing
- Broadcasting

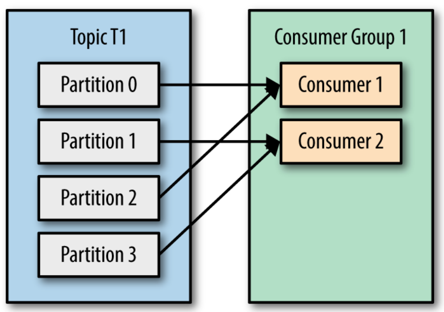


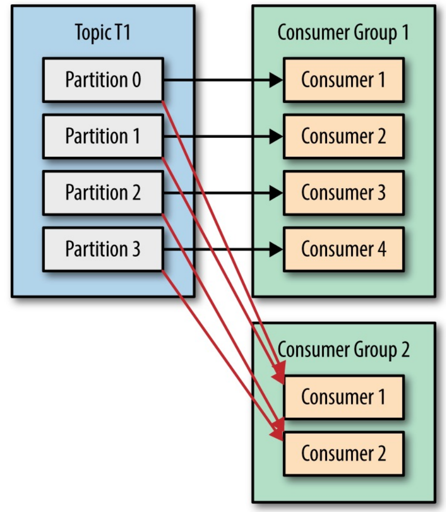
Kafka Consumer Groups

source https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html



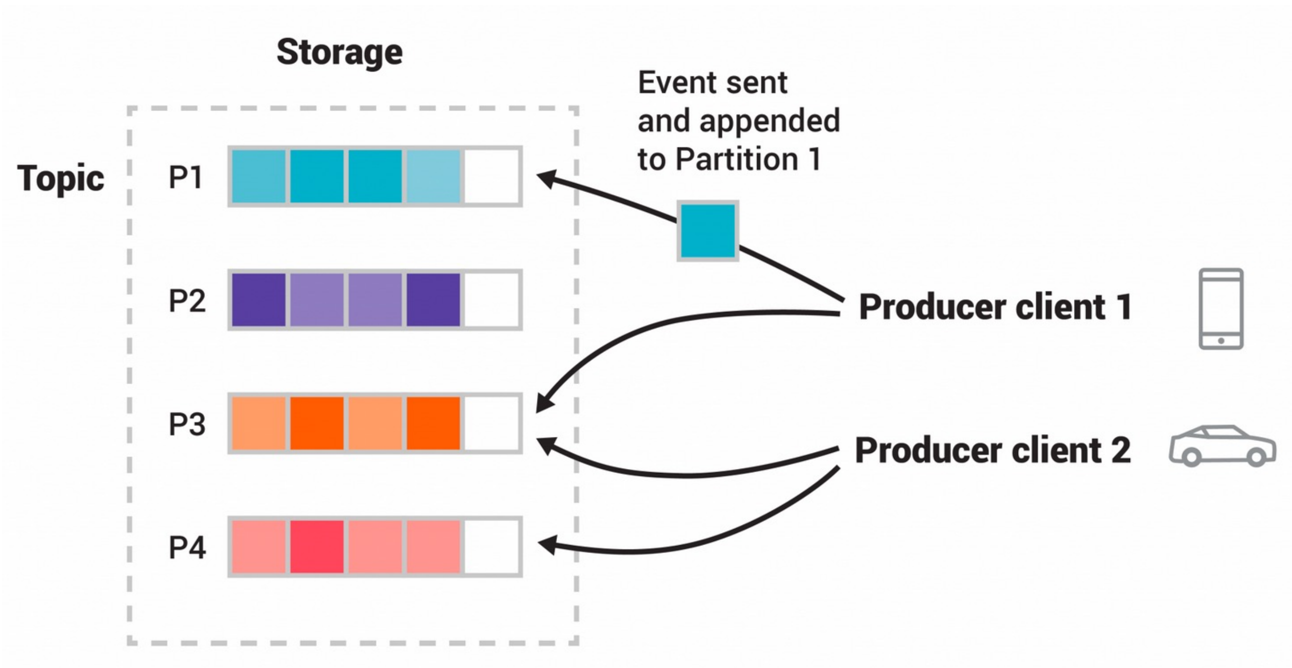






Kafka Topics & Partitions

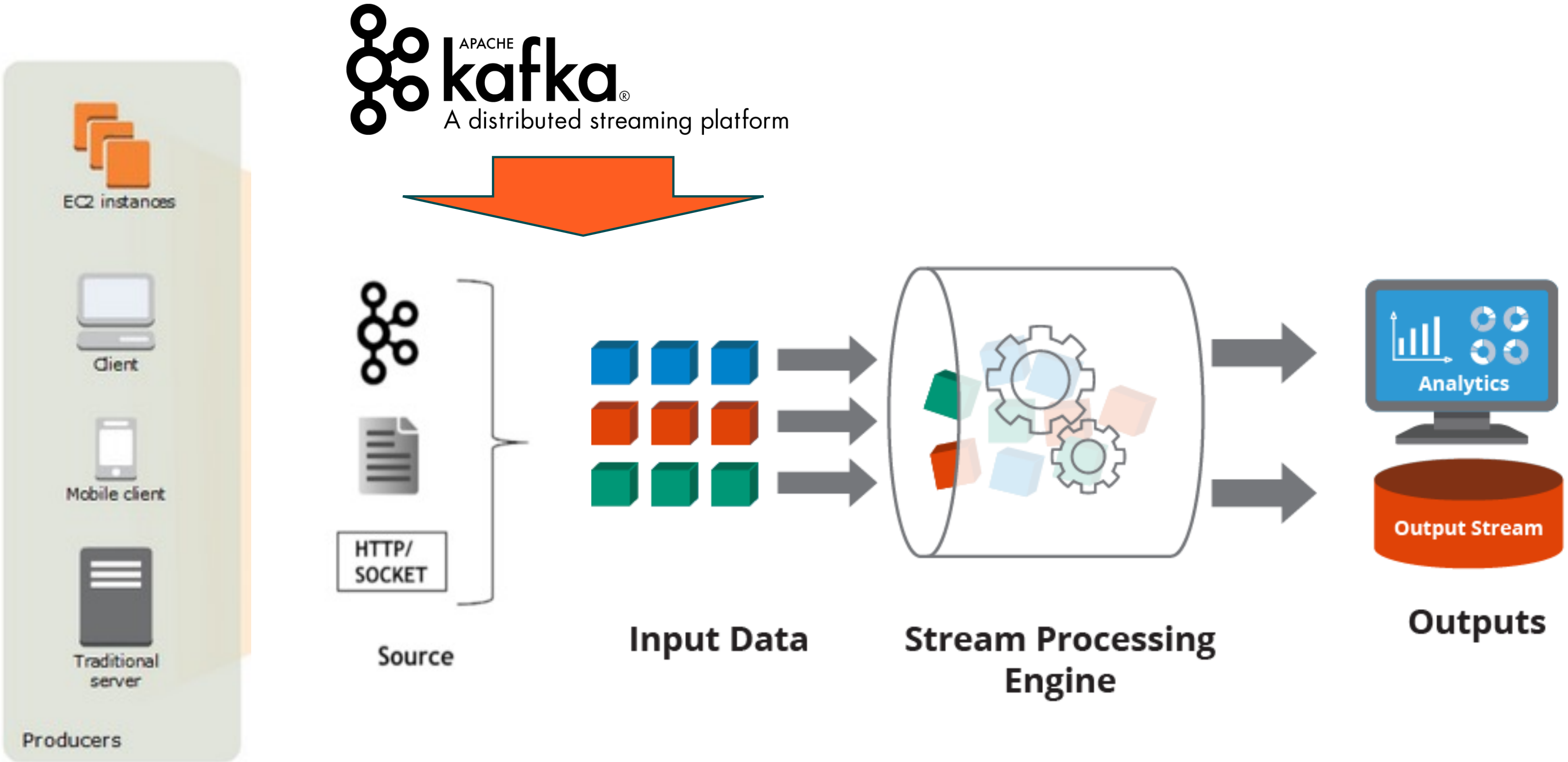
- By default, the producer will balance messages over all partitions of atopic evenly.
- In some cases, the producer will direct messages to specific partitions. This is typically done using the message key.



Kafka Fault Tolerance

- Data availability
  - A Kafka cluster spans multiple nodes
  - Partitions are replicated on multiple nodes
- Dealing with consumer disconnections/failures
  - Offset of the consumer in the log partition is recorded permanently
  - The same/another consumer can start processing records from this point
  - Provided delivery semantics:
    - **At-least-once**
    - **At-most-once**
    - In some cases, **exactly-once** semantic can be ensured (relies on transaction mechanisms)

Stream Processing Engines



publishers

consumer

Stream Processing Engines

Description

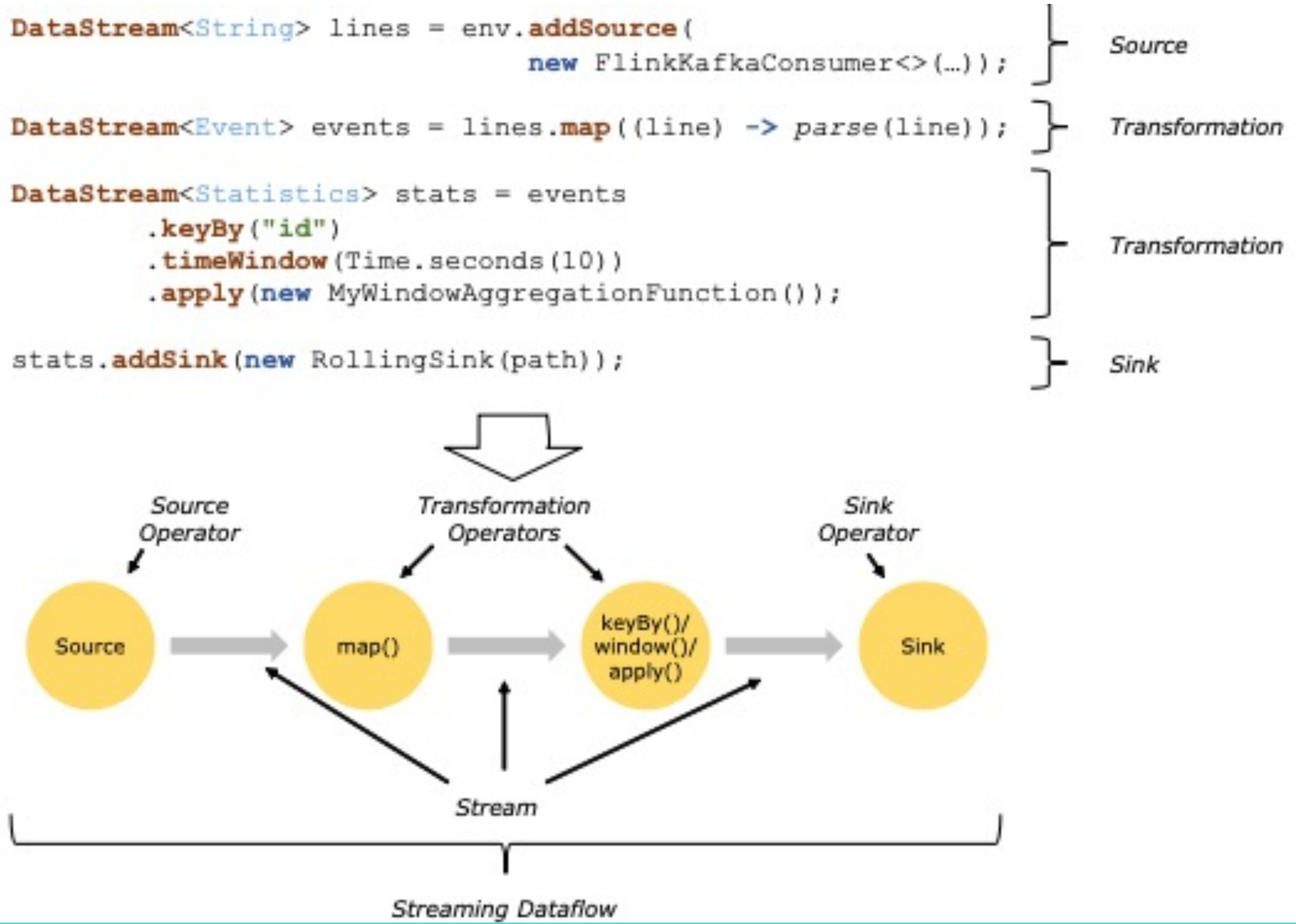
- A set of transformations is applied to a stream of records
- A program is a graph of transformations (Directed acyclic graph)
- Transformations are the same operations as in batch processing systems

Examples

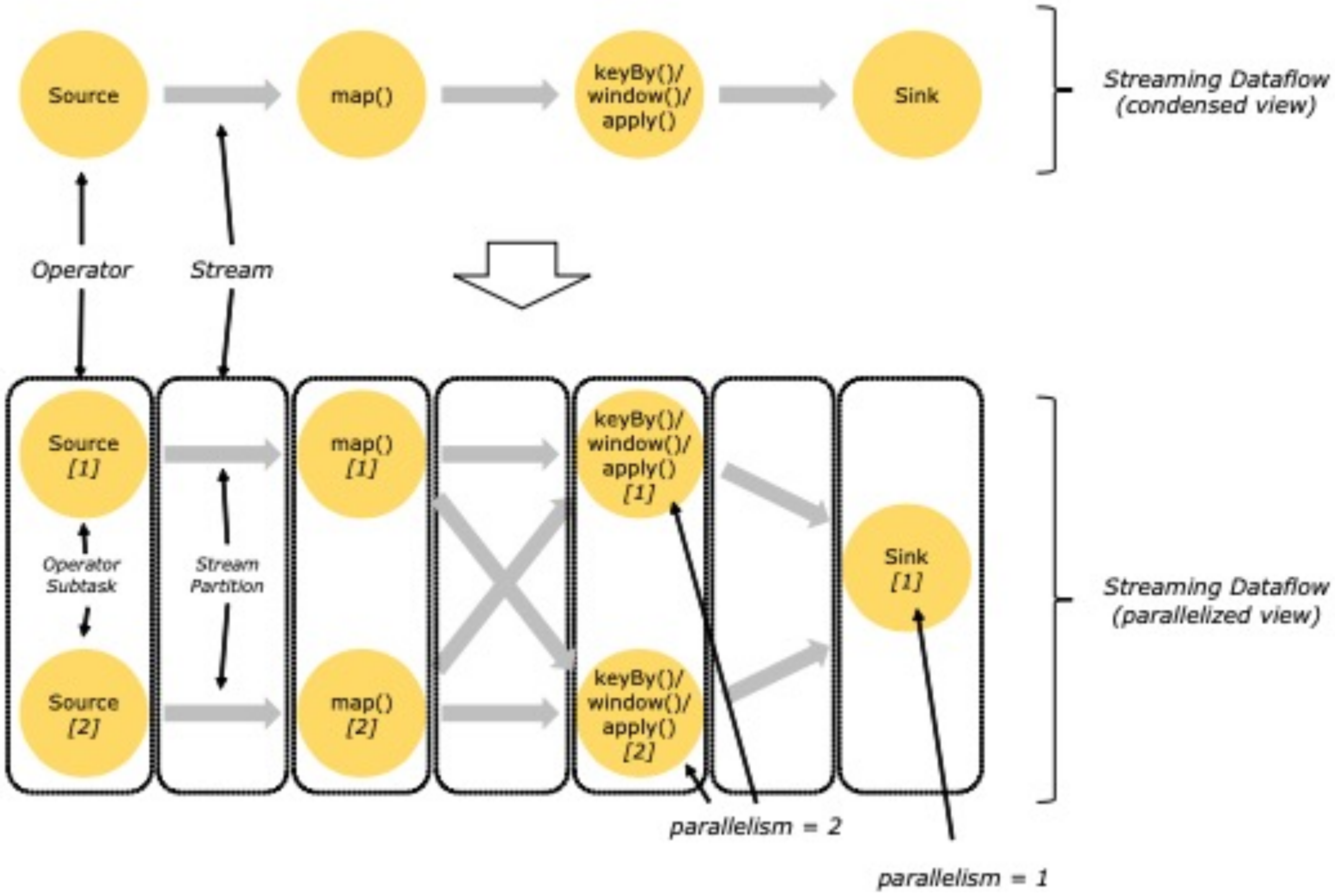
- Storm
- Flink
- Samza
- Spark streaming

Graph of transformations (Flink)

source https://ci.apache.org/projects/flink/flink-docs-release-1.6/ concepts/programming-model.html



Parallel Dataflow (Flink)



Making Time Discrete

To run computations on a continuous stream, it has to be split into **windows**.

Size of the window

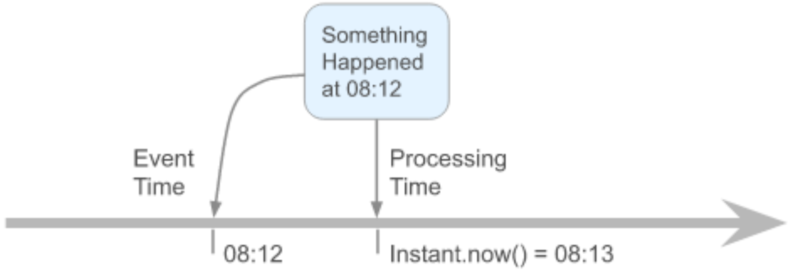
- 1 event: Each event is processed separately (Storm)
- Window's limits are based on:
  - Amount of data received
  - Time
  - Activity (concept of sessions)

2 reference times co-exists in the system

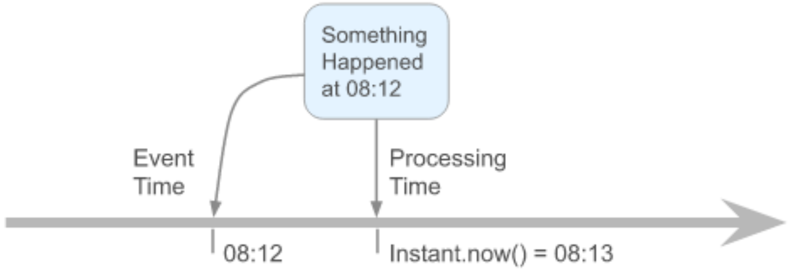
- Event time: time at which the events happened
  - There is also the time at which the event has been published
- Processing time: time at which the events are processed
  - Most systems build windows based on the processing time

Time Disorder

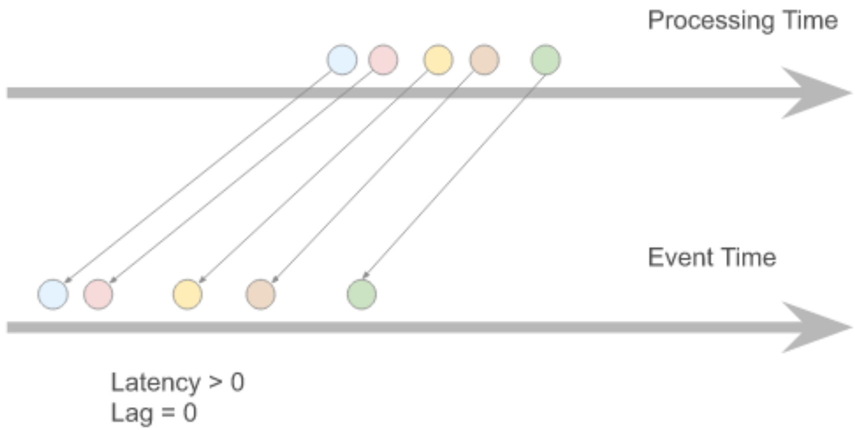
https://docs.hazelcast.com/hazelcast/5.3/pipelines/event-time



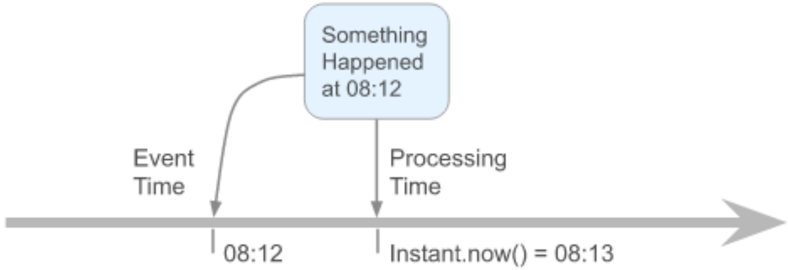
Time Disorder

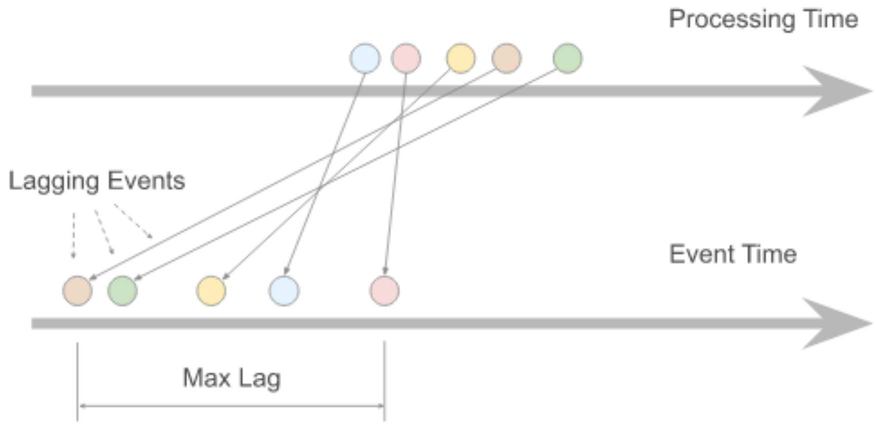


Ordered



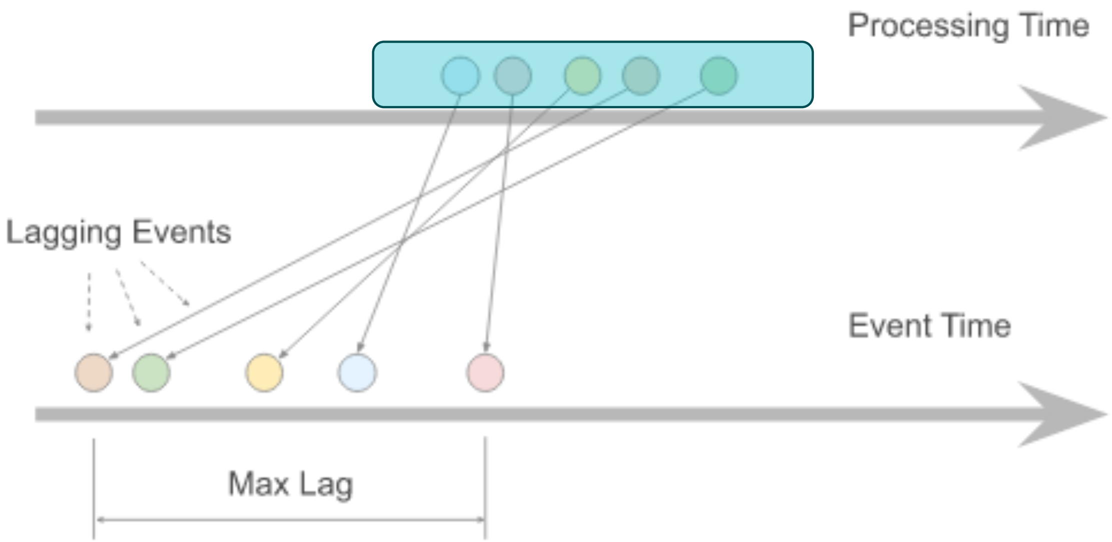
Time Disorder





Not ordered => Maximum event lag ?

Windows



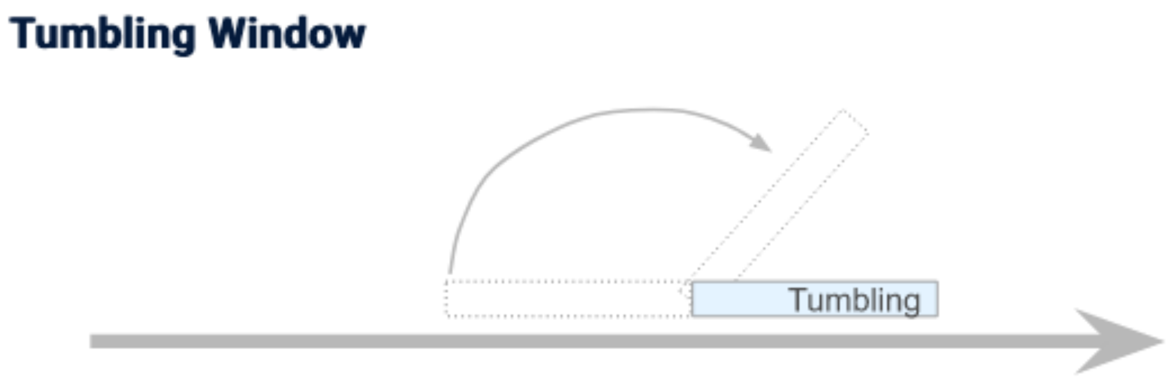
Types of Windows

Sliding window: Fixed-size window

- A new window is considered at each time step

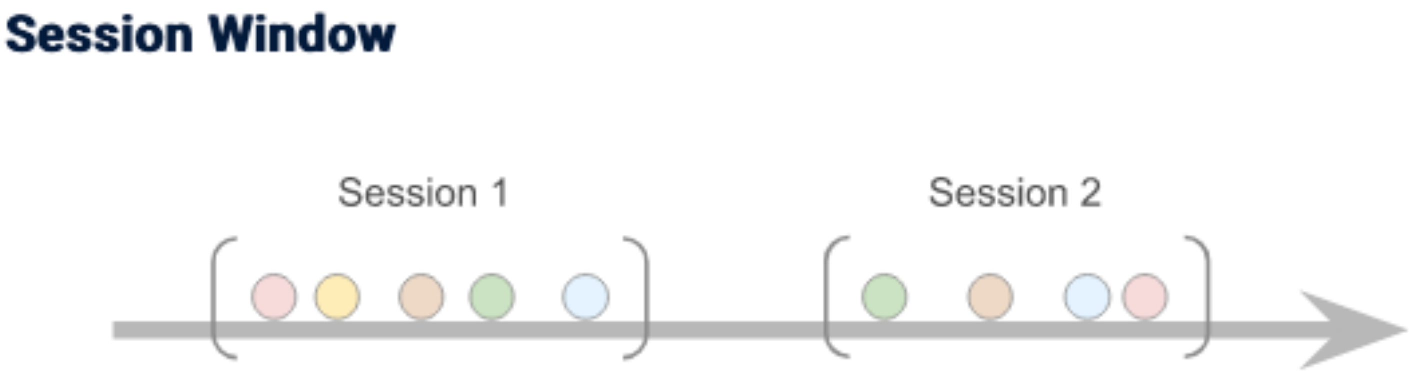
Types of Windows

- Tumbling window: Fixed-size window
  - Each event belongs to one window



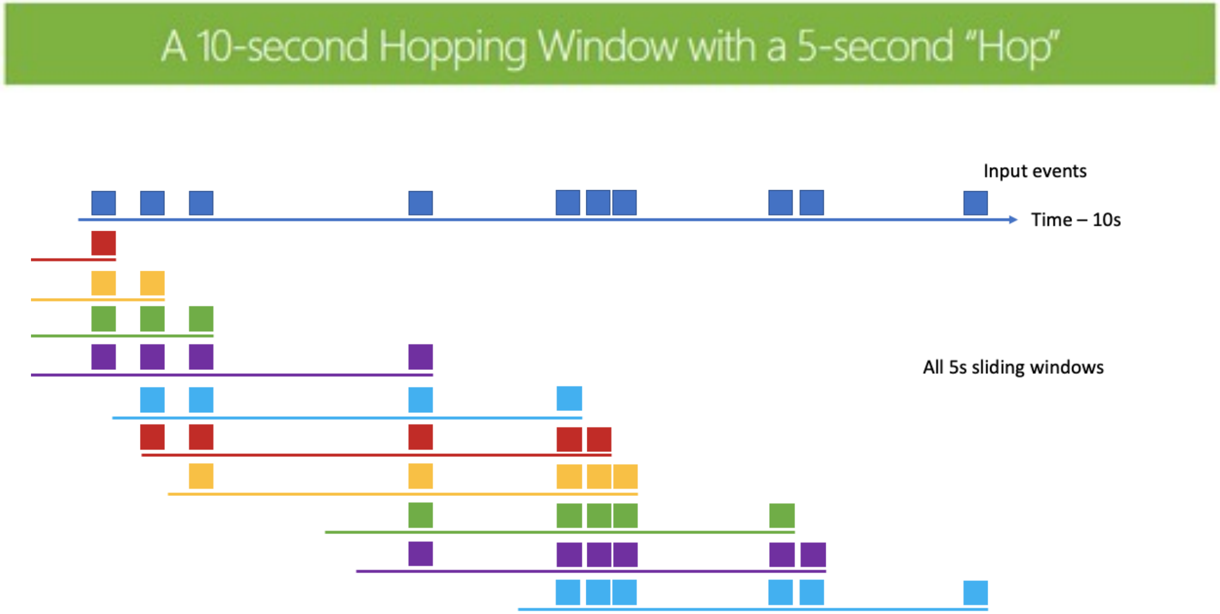
Types of Windows

- Session window: size not fixed
  - Group together events that happened closely together in time



Types of Windows

- Hopping window: Fixed-size window, windows overlap
  - hop size = time between the generation of two windows
  - hop size < window size



Every 5 seconds give me the count of Tweets over the last 10 seconds

Spark Streaming

Based on micro-batches

- The data stream is divided into micro-batches
  - Tumbling windows
  - Typically 1 to 4 seconds
- Each micro-batch is a RDD
- Multiple receivers can be created to manipulate multiple data streams in parallel
- The receiver tasks are distributed over the workers

Additional References

- https://www.oreilly.com/ideas/ the-world-beyond-batch-streaming-101, T. Akidau, 2015.
- Apache Flink: Stream and Batch Processing in a Single Engine., P. Carbone et al., IEEE, 2015.
- https://www.oreilly.com/ideas/ questioning-the-lambda-architecture, J. Kreps, 2014.
- and many more.