Data Streaming and Real-Time Analytics

Week 1: Introduction

Ekarat Rattagan, Ph.D.

Ref: Real Time Analytics: Algorithms and Systems, Twitter Inc.

Course Outline

1. Introduction to data streaming and real-time analytics
2. Lab1.1: Intro. to Kafka, Kafka Stream, wordcountdemo
3. Lab1.2: Kafa Stream VS others (e.g., Spark streaming) + Visualization
4. Lab1.3: Workshop I (10 points)
5. MQTT + Basic IoT + Mobile Application
6. Lab2.1: Mobile IoT (Kibana)
7. Lab2.2: Mobile IoT
8. Lab2.3: Workshop II (10 points)

Midterm exam (6 ~ 19 March) (25 points)

9. Real-time Case study: Multiplayer game online
10. Lab3.1: Implement Multiplayer game online system (Elasticsearch)
11. Lab3.2: Workshop III (10 points)
12. Project proposal presentation (5 points)
13. Real-time Case study: Real-time Bidding (RTB) / Trading platform
14. Lab4.1: Implement
15. Lab4.2: Workshop IV (10 points)
16. Final project presentation (30 points)

Overview

Data Streaming and Real-Time Analytics

Streaming Data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes). Streaming data includes a wide variety of data such as log files generated by customers using your mobile or web applications, ecommerce purchases, in-game player activity, information from social networks, financial trading floors, or geospatial services, and telemetry from connected devices or instrumentation in data centers.

This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows, and used for a wide variety of analytics including correlations, aggregations, filtering, and sampling. Information derived from such analysis gives companies visibility into many aspects of their business and customer activity such as –service usage (for metering/billing), server activity, website clicks, and geo-location of devices, people, and physical goods –and enables them to respond promptly to emerging situations. For example, businesses can track changes in public sentiment on their brands and products by continuously analyzing social media streams, and respond in a timely fashion as the necessity arises.

https://aws.amazon.com/streaming-data/

What is streaming data use for?

- Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance, detects any potential defects in advance, and places a spare part order automatically preventing equipment down time.
- A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements.
- A real-estate website tracks a subset of data from consumers' mobile devices and makes real-time property recommendations of properties to visit based on their geo-location.
- A solar power company has to maintain power throughput for its customers, or pay penalties. It implemented a streaming data application that monitors of all of panels in the field, and schedules service in real time, thereby minimizing the periods of low throughput from each panel and the associated penalty payouts.
- A media publisher streams billions of clickstream records from its online properties, aggregates and enriches the data with demographic information about users, and optimizes content placement on its site, delivering relevancy and better experience to its audience.
- An online gaming company collects streaming data about player-game interactions, and feeds the data into its gaming platform. It then analyzes the data in real-time, offers incentives and dynamic experiences to engage its players.

https://aws.amazon.com/streaming-data/

What is streaming data use for?

- Examples of use of streaming analytics include, but not limited to,
- (a) sequence mining [139, 121, 117] for, say, credit card fraud detection, motion capture sequences and chlorine levels in drinking water
- (b) discovering human activity, which often exhibit discontinuity (interruption) or varying frequencies, from sensor streams [140]
- (c) determing top-K traversal sequences in streaming clicks
- (d) finding closed structures in music melody streams [120].

Data processing

Data processing

1. **Data integration**
   - Process of data extraction-transformation-load (ETL)
   - Traditionally run at a regular time interval (not applicable for real-time fashion)
   - In real-time scenario, large amounts of small sized data are processed into the data warehouse. E.g.,
     - Click stream events
     - Innumerate online transactions
     - Social network data

Data processing

2. Real-time analytics

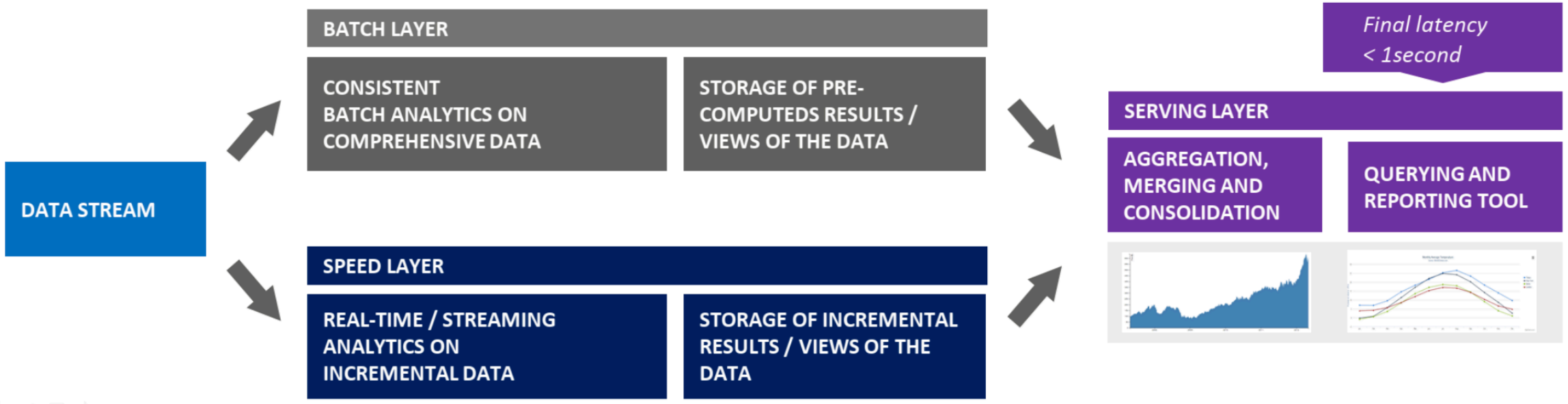
- Analyse the data residing in a data store, or in streaming
- Traditional analytic tool not able to return the analytics results within a time limit, thus loses the value.
- Real-time analytic tools address the need to have “as fast as disk” processing speeds, or “near real time”
- End-users and business analysts can access/query both streaming and historical data in a real-time manner

Offline VS Real-time

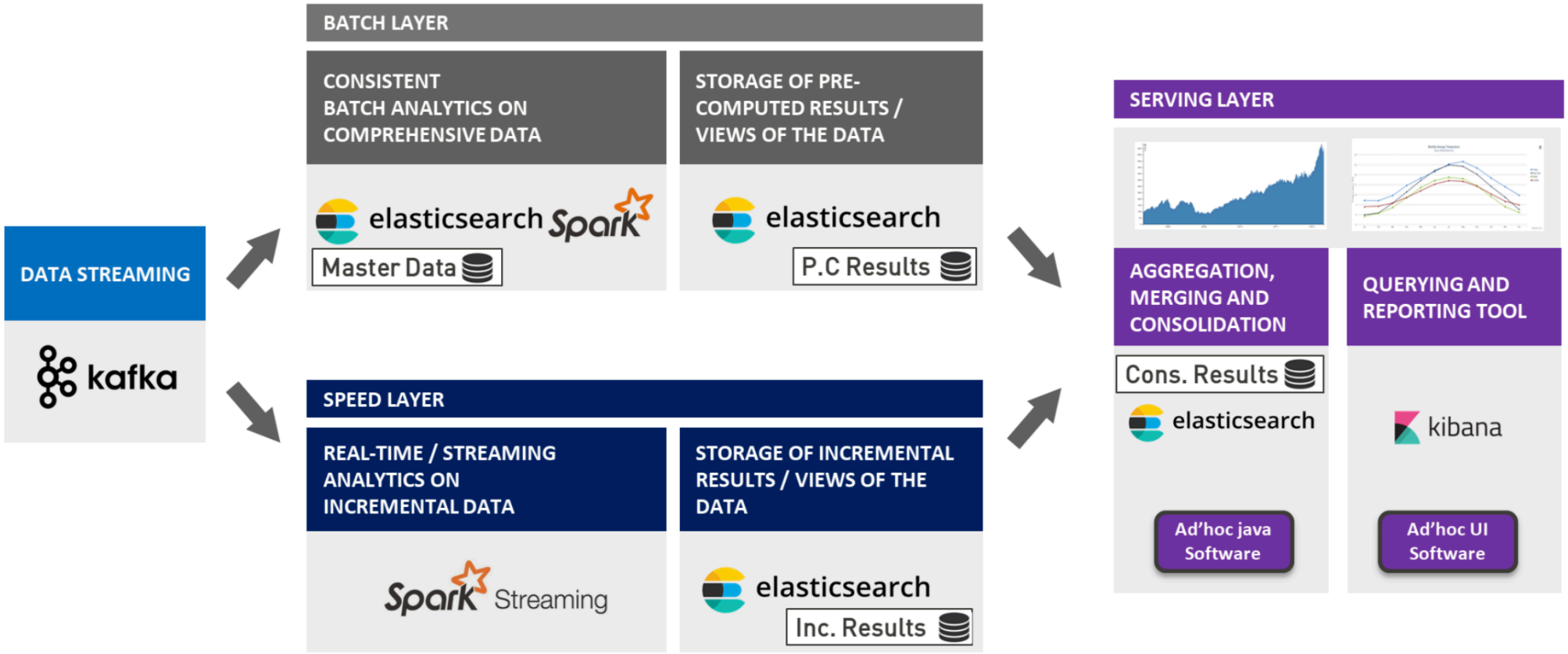
	Offline	Real-time
Timeliness	Day level delay or hour level delay	Minute level, second level or even millisecond level
Processing mode	Batch calculation	Microbatch calculation Streaming calculation
Main Resources	Disk	CPU,Memory
Analysis	Complex analysis	Simple statistical analysis of data
Representative technology	Hadoop HDFS、MapReduce Hive	Spark Streaming Storm,Flink

https://www.youtube.com/watch?v=nPaEUYZsFak

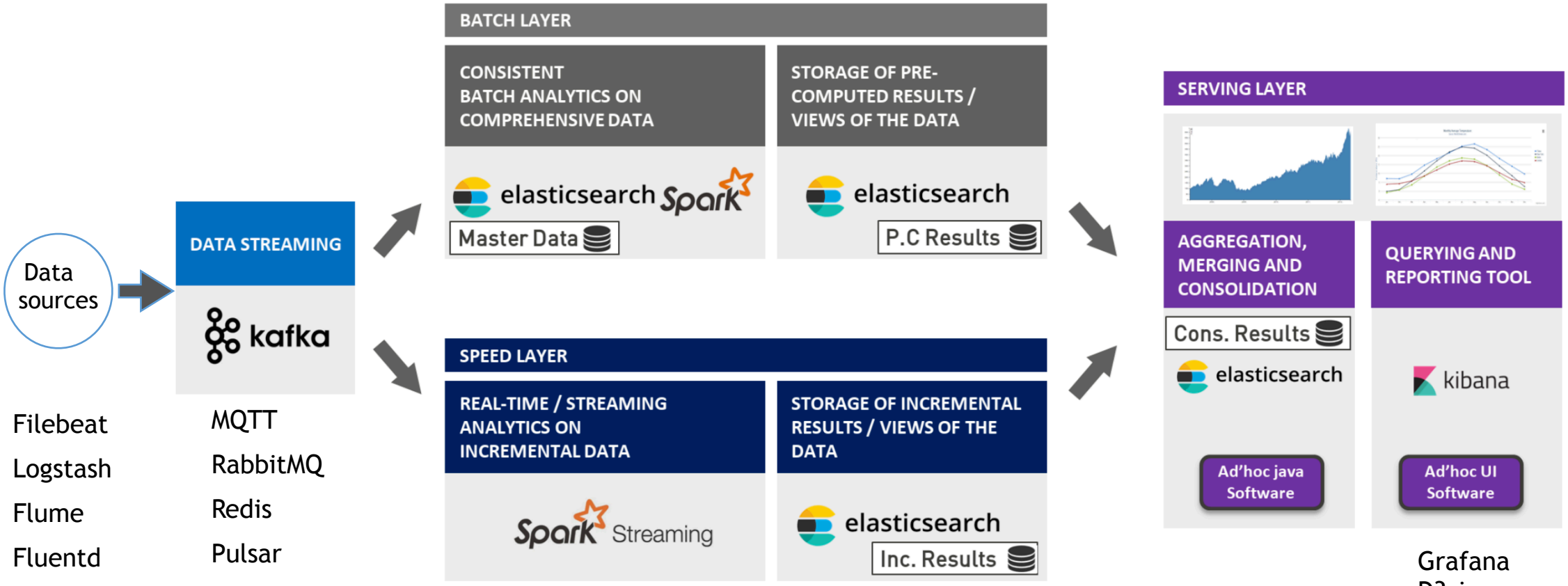
Lambda Architecture



Lambda Architecture (Tools)



Lambda Architecture (Tools)



Filebeat
Logstash
Flume
Fluentd

MQTT
RabbitMQ
Redis
Pulsar

Flink
Kafka Stream
Storm
Samza

Druid
AthenaX

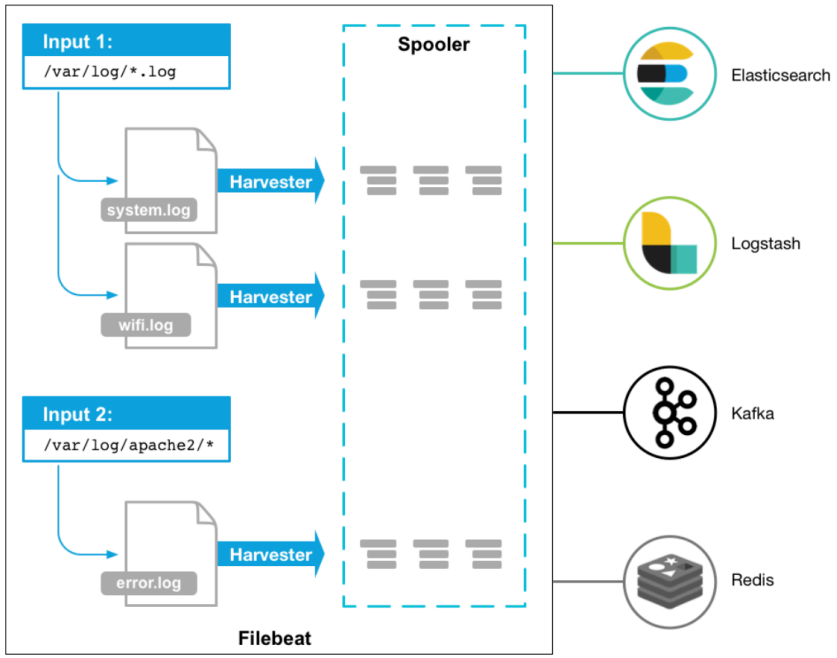
Grafana
D3.js

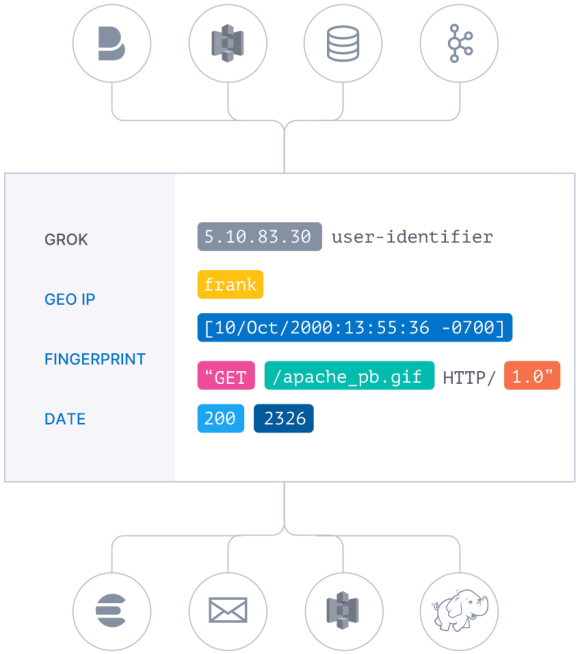
Data Sources

- FileBeat
  - A lightweight shipper for forwarding and centralising log data
- Logstash
  - A free and open server-side data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to your favorite "stash."
- Apache Flume
  - A distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data
- Fluentd
  - An open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data.

Data Sources

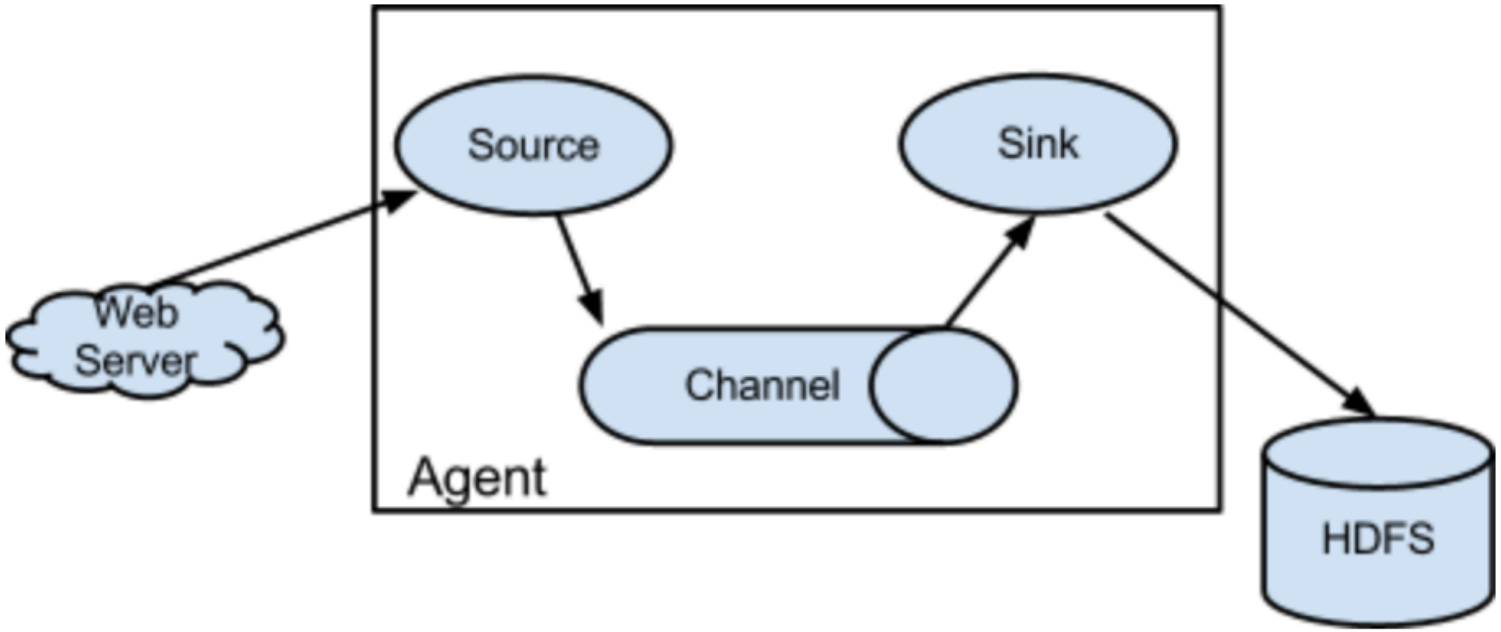
- FileBeat (https://www.elastic.co/beats/filebeat)
- Logstash (https://www.elastic.co/logstash)





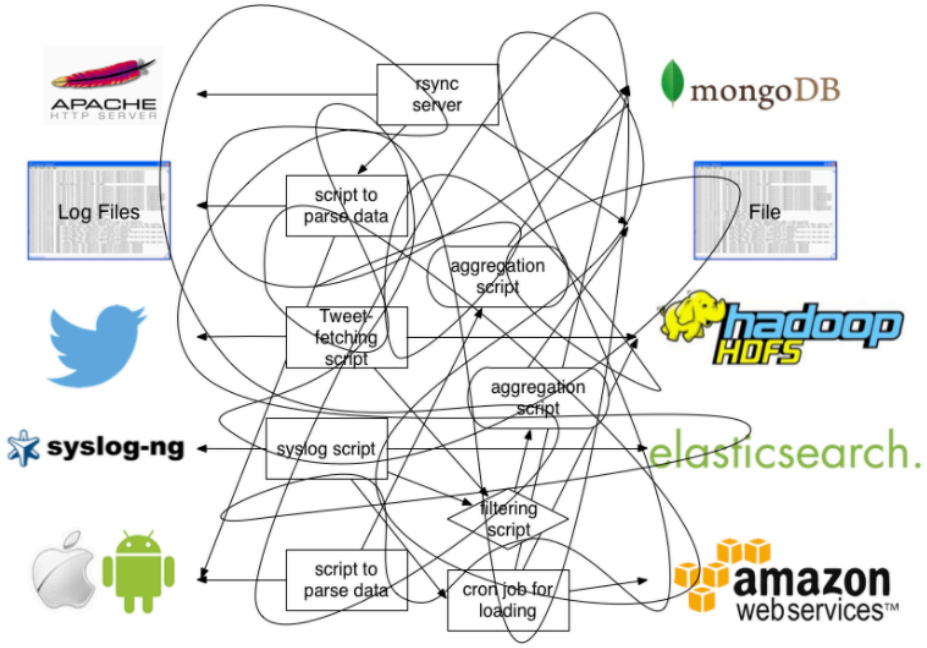
Data Sources

- Apache Flume (https://flume.apache.org/)
-

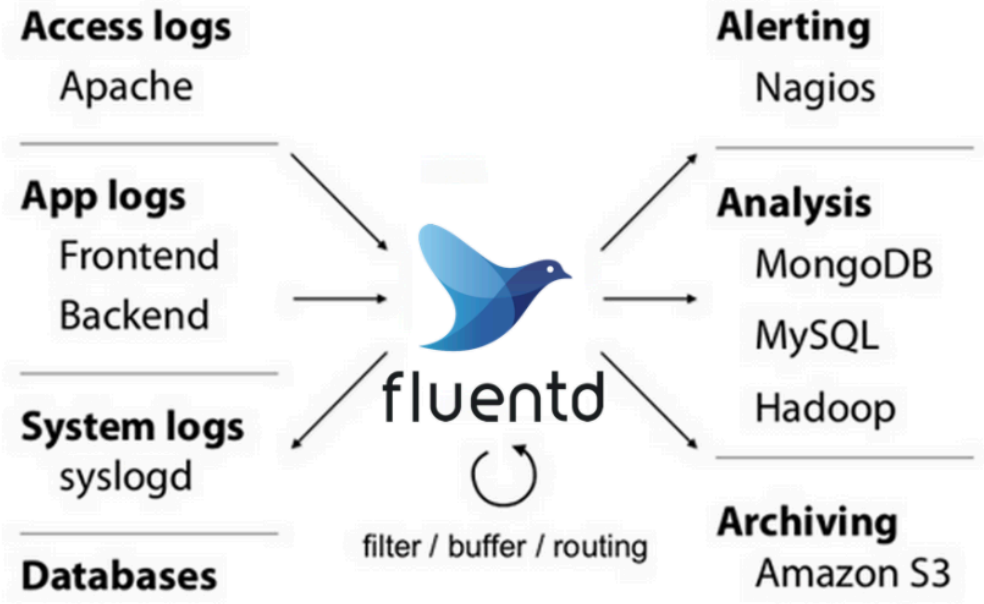


Data Sources

- Fluentd (https://www.fluentd.org/)



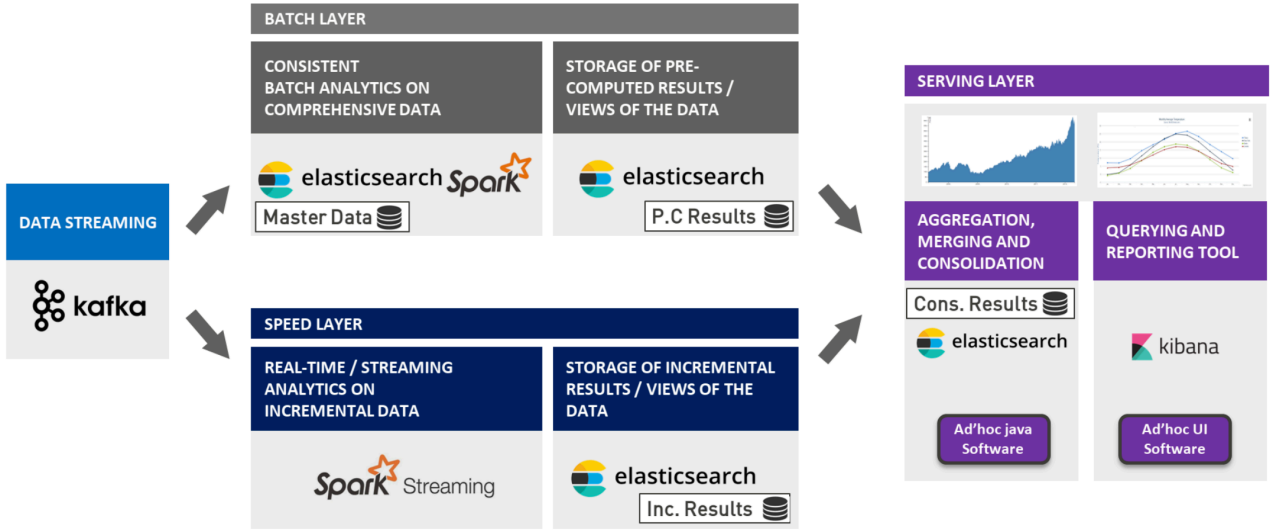
Before Fluentd



After Fluentd

Data Streaming

- Kafka
- MQTT
- RabbitMQ
- Redis
- Apache Pulsar



Data Streaming

- Kafka
  - An event streaming platform
  - Kafka combines three key capabilities
    1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
    2. To **store** streams of events durably and reliably for as long as you want.
    3. To **process** streams of events as they occur or retrospectively.

Data Streaming

- MQTT
  - An OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.
  - MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

https://mqtt.org/

Data Streaming

- RabbitMQ
  - An open-source message-broker software (sometimes called message-oriented middleware) that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), and other protocols.
  - The RabbitMQ server program is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages.

https://en.wikipedia.org/wiki/RabbitMQ

Data Streaming

- Redis
  - An open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker.
  - Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.
  - Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

https://redis.io/

Data Streaming

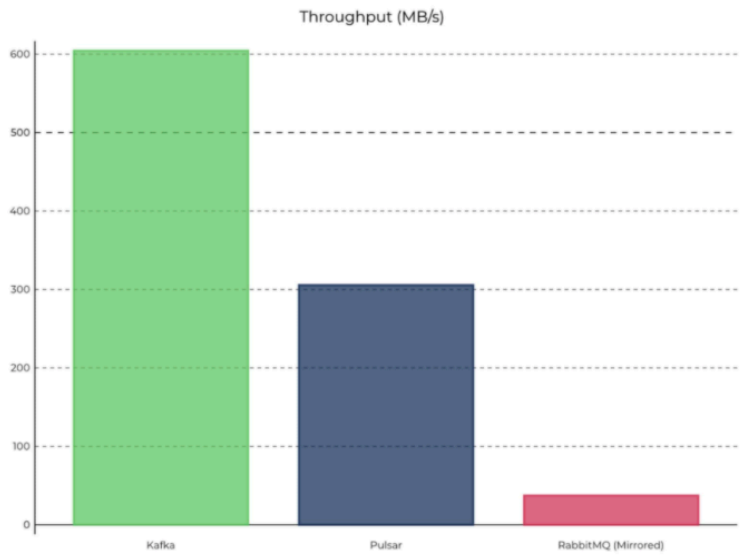
- Apache pulsar
  - Apache Pulsar is an open-source distributed messaging system. Originally developed as a queuing system, it has been broadened in recent releases to add event streaming features.
  - Pulsar makes use of Apache BookKeeper™ for its storage layer—a project created at Yahoo as a high-availability solution to Hadoop's HDFS NameNode (although not ultimately used for that use case).
  - It shares properties with both Kafka and RabbitMQ. Pulsar is a largely community-led project with no enterprise-grade commercial backing today.

https://redis.io/

Throughput

Kafka provides the highest throughput of all systems, writing 15x faster than RabbitMQ and 2x faster than Pulsar, based on the popular OpenMessaging Benchmark*

*Full results described in the associated: benchmarking comparison



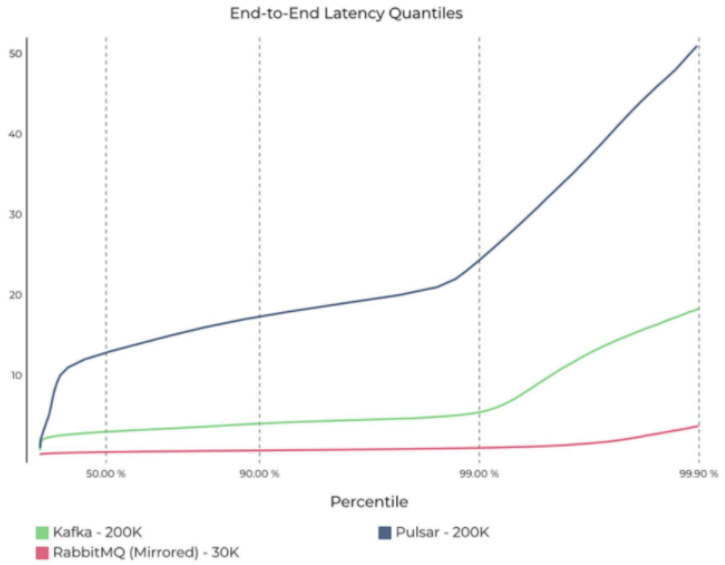
Latency

Kafka provides the lowest latency (5ms at p99) at higher throughputs, while also providing strong durability and high availability*.

Kafka in its default configuration is faster than Pulsar in all latency benchmarks, and it is faster up to p99.9 when set to fsync on every message.

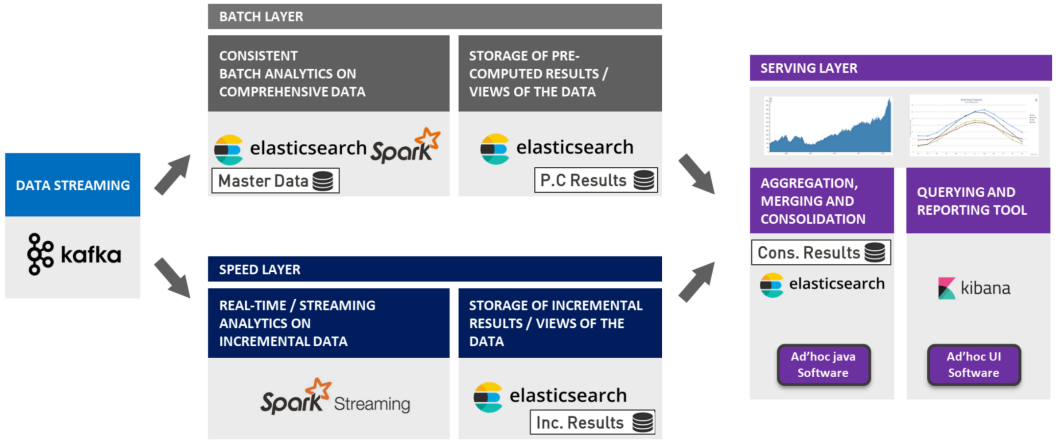
RabbitMQ can achieve lower end-to-end latency than Kafka, but only at significantly lower throughputs (30K messages/sec versus 200K messages/sec for Kafka), after which its latency degrades significantly.

*Full results described in the associated: benchmarking comparison



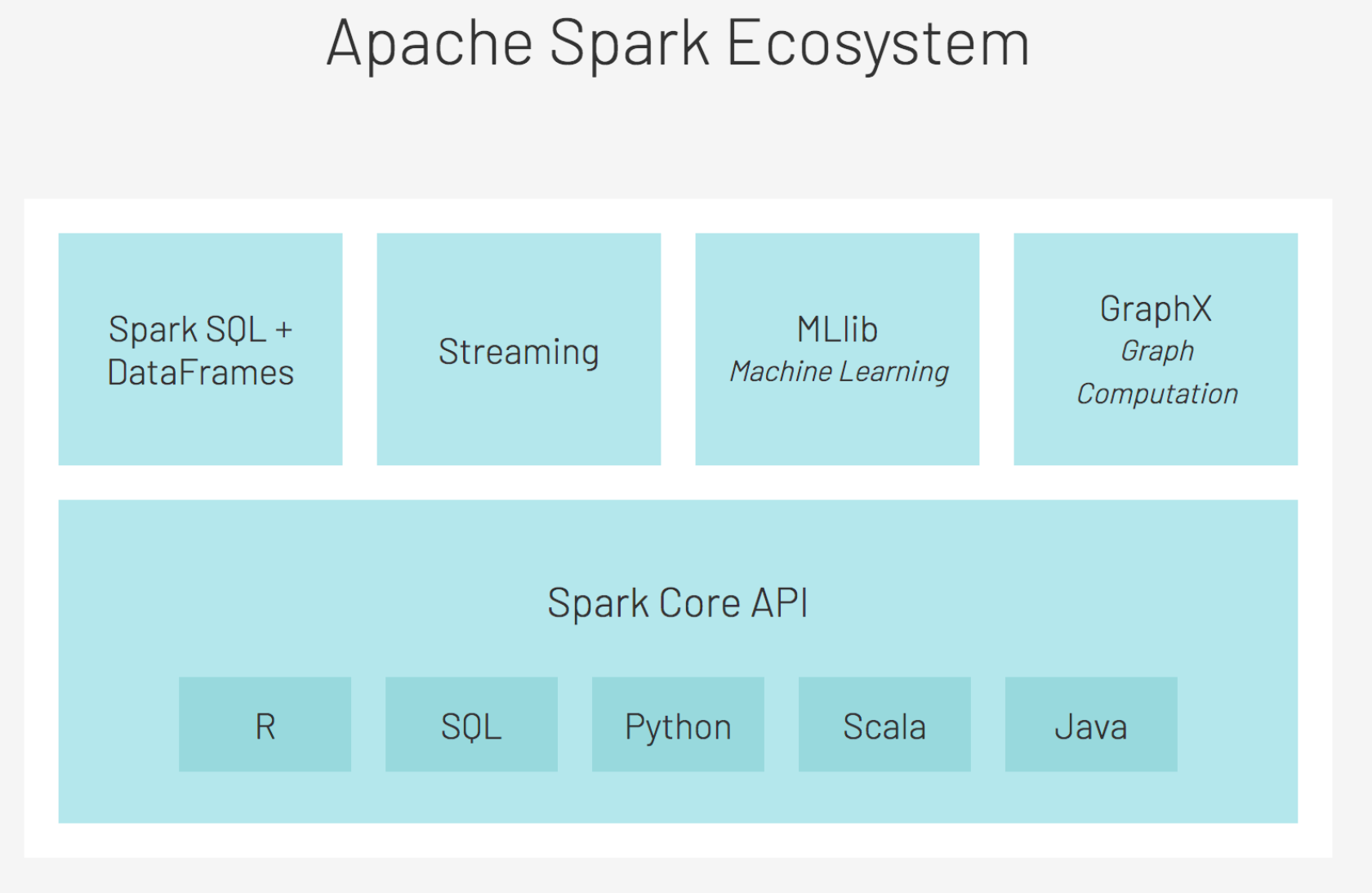
https://www.confluent.io/kafka-vs-pulsar/

Processing

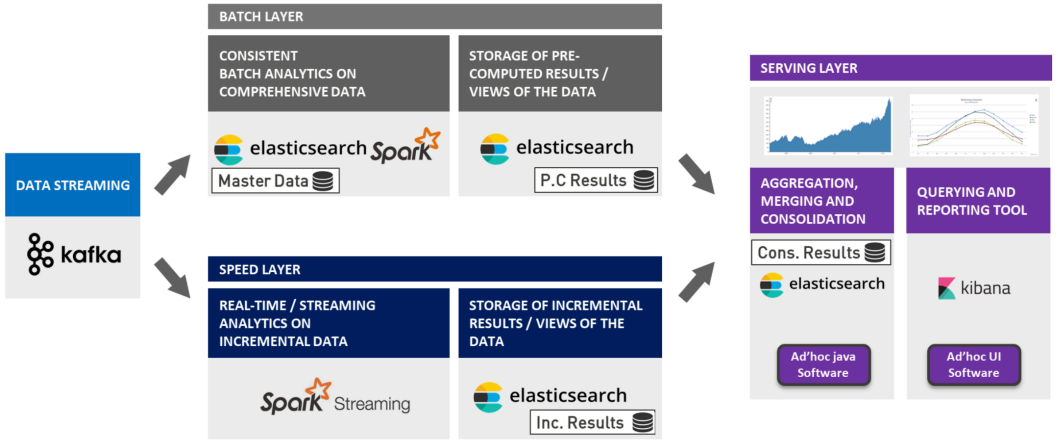


- Batch layer
  - Apache Spark
    - An open-source distributed general-purpose cluster-computing framework. A lightning-fast **unified analytics engine** for big data and machine learning.
    - Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.
    - Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since.

Processing


Apache Spark Ecosystem
Spark SQL + DataFrames
Streaming
MLlib
Machine Learning
GraphX
Graph Computation
Spark Core API
R
SQL
Python
Scala
Java

Processing



- Speed layer (Stream processing)
  - Two types of stream processing
    1. **Native streaming**
       - Apache Storm
       - Apache Flink
       - Kafka Stream
       - Apache Samza
    2. **Micro-batching streaming**
       - Spark streaming
       - Storm-Trident

Native streaming Processing

- Apache storm
  - A free and open source distributed realtime computation system.
  - Apache Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing.
  - Apache Storm is simple, can be used with any programming language, and is a lot of fun to use!
  - Apache Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more.
  - Apache Storm is fast: a benchmark clocked it at over **a million tuples processed per second per node**. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.
  - Apache Storm integrates with the queueing and database technologies you already use. An Apache Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.

https://storm.apache.org/

Native streaming Processing

- Apache Flink
  - A framework and distributed processing engine for stateful computations over *unbounded and bounded* data streams.
  - Flink has been designed to run in *all common cluster environments*, perform computations at *in-memory speed* and at *any scale*

https://flink.apache.org/flink-architecture.html

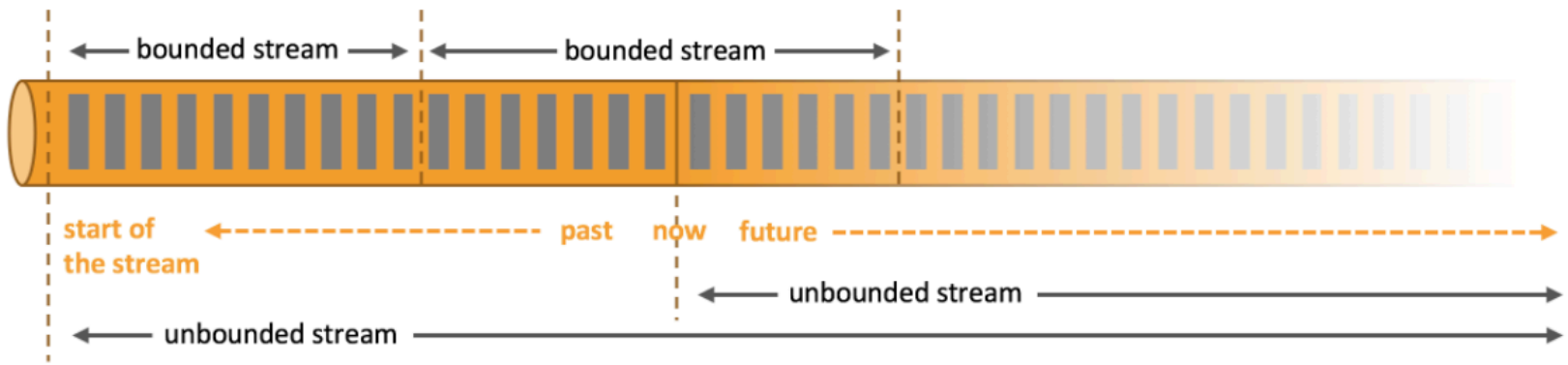
Process Unbounded and Bounded Data

Any kind of data is produced as a stream of events. Credit card transactions, sensor measurements, machine logs, or user interactions on a website or mobile application, all of these data are generated as a stream.

Data can be processed as *unbounded* or *bounded* streams.

1. **Unbounded streams** have a start but no defined end. They do not terminate and provide data as it is generated. Unbounded streams must be continuously processed, i.e., events must be promptly handled after they have been ingested. It is not possible to wait for all input data to arrive because the input is unbounded and will not be complete at any point in time. Processing unbounded data often requires that events are ingested in a specific order, such as the order in which events occurred, to be able to reason about result completeness.

2. **Bounded streams** have a defined start and end. Bounded streams can be processed by ingesting all data before performing any computations. Ordered ingestion is not required to process bounded streams because a bounded data set can always be sorted. Processing of bounded streams is also known as batch processing.



Native streaming Processing

- Kafka Stream
  - A client library for building applications and microservices, where the input and output data are stored in Kafka clusters.
  - It combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the benefits of Kafka's server-side cluster technology.

https://kafka.apache.org/documentation/streams/

Kafka Streams use cases

The New York Times uses Apache Kafka and the Kafka Streams to store and distribute, in real-time, published content to the various applications and systems that make it available to the readers.

Pinterest uses Apache Kafka and the Kafka Streams at large scale to power the real-time, predictive budgeting system of their advertising infrastructure. With Kafka Streams, spend predictions are more accurate than ever.

As the leading online fashion retailer in Europe, Zalando uses Kafka as an ESB (Enterprise Service Bus), which helps us in transitioning from a monolithic to a micro services architecture. Using Kafka for processing event streams enables our technical team to do near-real time business intelligence.

Rabobank is one of the 3 largest banks in the Netherlands. Its digital nervous system, the Business Event Bus, is powered by Apache Kafka. It is used by an increasing amount of financial processes and services, one of which is Rabo Alerts. This service alerts customers in real-time upon financial events and is built using Kafka Streams.

Native streaming Processing

LINE

LINE uses Apache Kafka as a central datahub for our services to communicate to one another. Hundreds of billions of messages are produced daily and are used to execute various business logic, threat detection, search indexing and data analysis. LINE leverages Kafka Streams to reliably transform and filter topics enabling sub topics consumers can efficiently consume, meanwhile retaining easy maintainability thanks to its sophisticated yet minimal code base.

trivago®

Trivago is a global hotel search platform. We are focused on reshaping the way travelers search for and compare hotels, while enabling hotel advertisers to grow their businesses by providing access to a broad audience of travelers via our websites and apps. As of 2017, we offer access to approximately 1.8 million hotels and other accommodations in over 190 countries. We use Kafka, Kafka Connect, and Kafka Streams to enable our developers to access data freely in the company. Kafka Streams powers parts of our analytics pipeline and delivers endless options to explore and operate on the data sources we have at hand.

Native streaming Processing

- Apache Samsa

  - **Apache Samza** is an open-source, near-realtime, asynchronous computational framework for stream processing developed by the Apache Software Foundation in Scala and Java. It has been developed in conjunction with Apache Kafka. Both were originally developed by LinkedIn.
  - Unlike batch systems such as Apache Hadoop or Apache Spark, it provides continuous computation and output, which result in sub-second[2] response times.

https://en.wikipedia.org/wiki/Apache_Samza

Micro-batching processing

- Apache Storm-Trident
  - Trident is a high-level abstraction for doing realtime computing on top of Storm.
  - It allows you to seamlessly intermix high throughput (millions of messages per second), stateful stream processing with low latency distributed querying.
  - If you're familiar with high level batch processing tools like Pig or Cascading, the concepts of Trident will be very familiar – Trident has joins, aggregations, grouping, functions, and filters.
  - In addition to these, Trident adds primitives for doing stateful, incremental processing on top of any database or persistence store. Trident has consistent, exactly-once semantics, so it is easy to reason about Trident topologies.

http://storm.apache.org/releases/current/Trident-tutorial.html

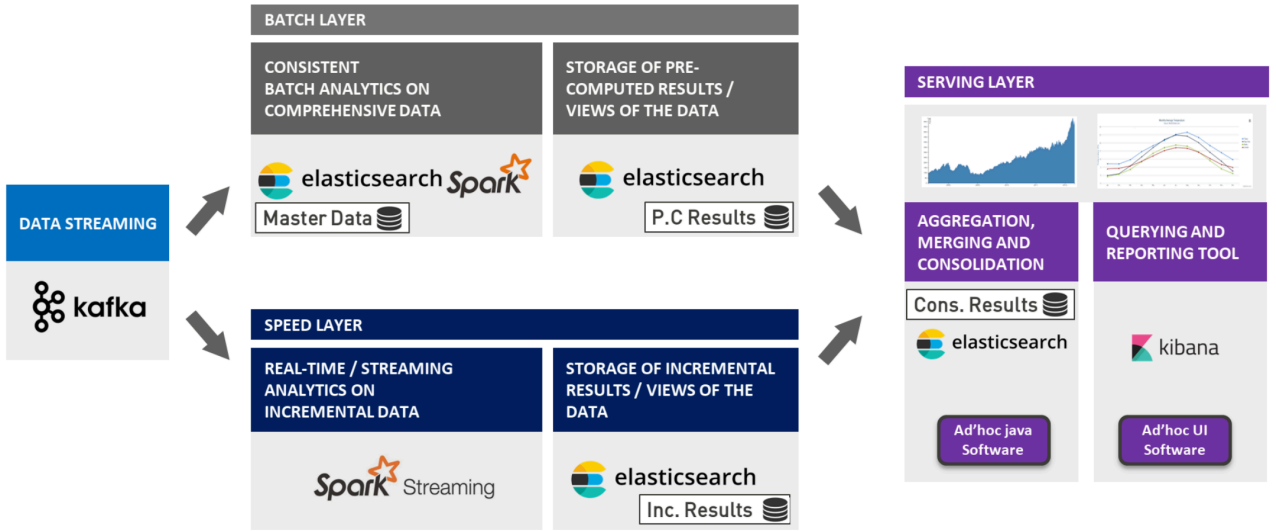
Micro-batching processing

- Apache Spark Streaming
  - Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
  - Data can be ingested from many sources like Kafka, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like `map`, `reduce`, `join` and `window`.
  - Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data streams.

http://storm.apache.org/releases/current/Trident-tutorial.html

Serving Layer

- Elasticsearch
- Kibana
- D3.js



Serving Layer

- Elasticsearch

What is Elasticsearch?

edit

You know, for search (and analysis)

Elasticsearch is the distributed search and analytics engine at the heart of the Elastic Stack. Logstash and Beats facilitate collecting, aggregating, and enriching your data and storing it in Elasticsearch. Kibana enables you to interactively explore, visualize, and share insights into your data and manage and monitor the stack. Elasticsearch is where the indexing, search, and analysis magic happens.

Elasticsearch provides near real-time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data, or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches. You can go far beyond simple data retrieval and aggregate information to discover trends and patterns in your data. And as your data and query volume grows, the distributed nature of Elasticsearch enables your deployment to grow seamlessly right along with it.

While not *every* problem is a search problem, Elasticsearch offers speed and flexibility to handle data in a wide variety of use cases:

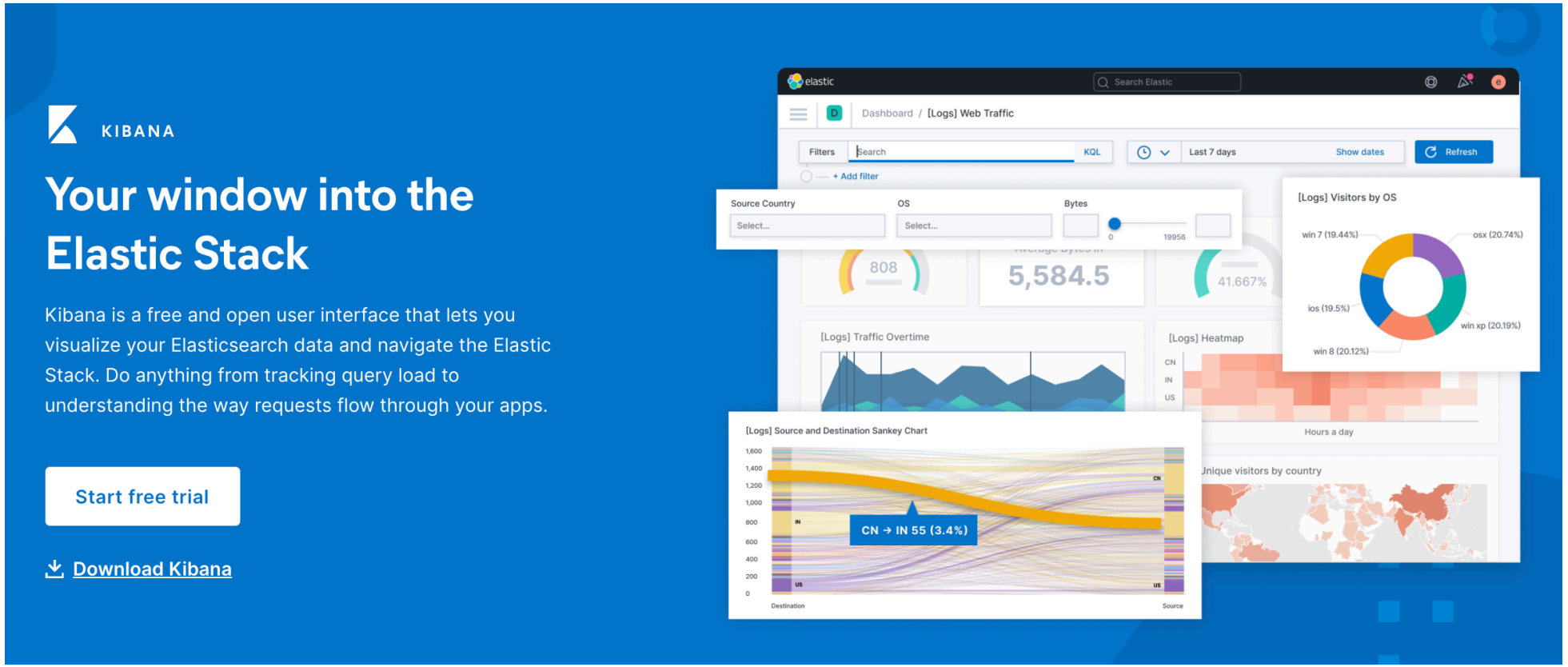
- Add a search box to an app or website
- Store and analyze logs, metrics, and security event data
- Use machine learning to automatically model the behavior of your data in real time
- Automate business workflows using Elasticsearch as a storage engine
- Manage, integrate, and analyze spatial information using Elasticsearch as a geographic information system (GIS)
- Store and process genetic data using Elasticsearch as a bioinformatics research tool

We're continually amazed by the novel ways people use search. But whether your use case is similar to one of these, or you're using Elasticsearch to tackle a new problem, the way you work with your data, documents, and indices in Elasticsearch is the same.

https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

Serving Layer

- Kibana

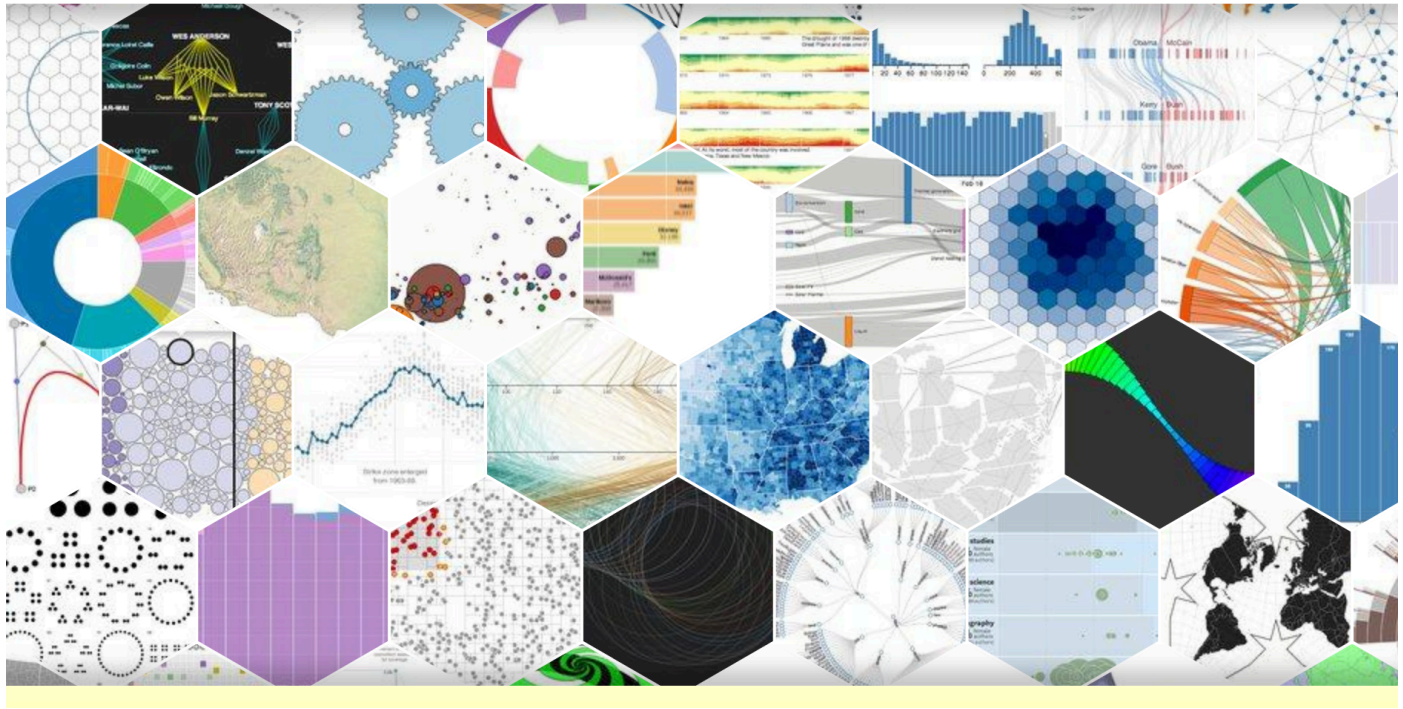


https://www.elastic.co/kibana

Serving Layer

- D3.js

Data-Driven Documents



https://d3js.org/

Case study

Case study I (IoT)

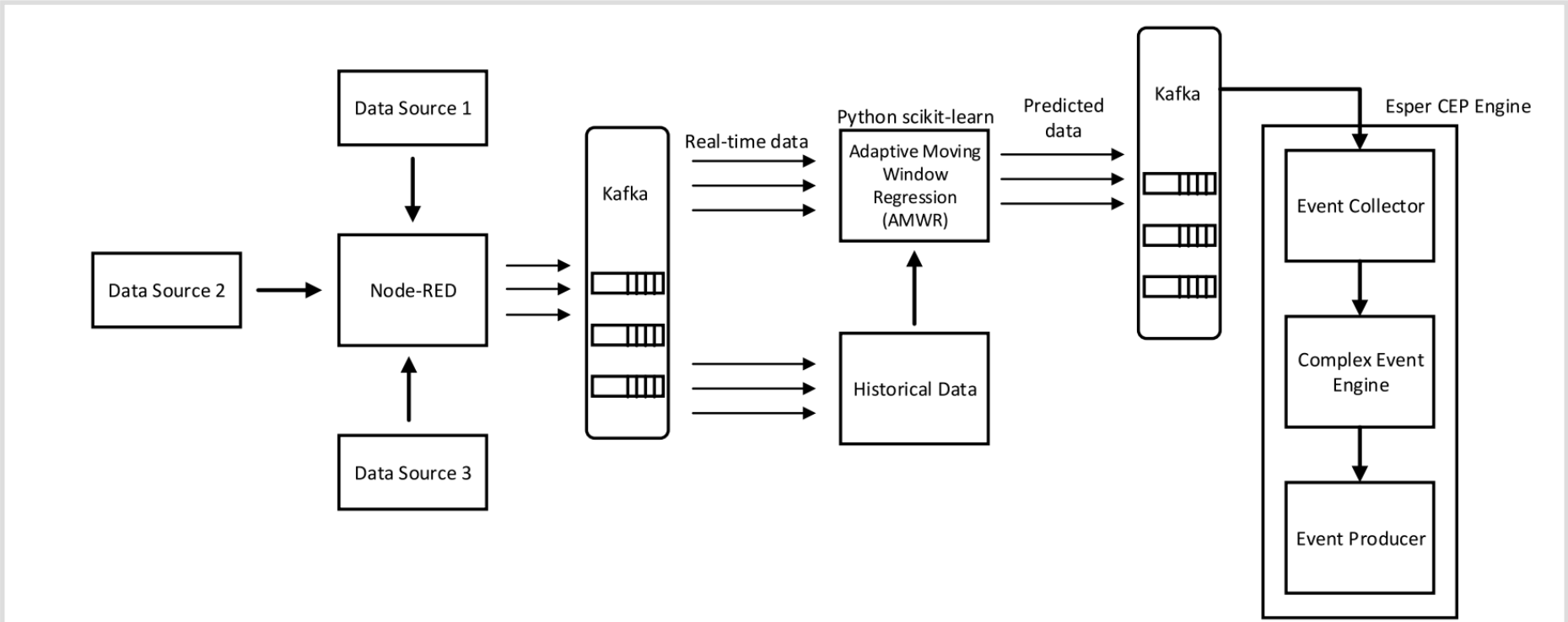
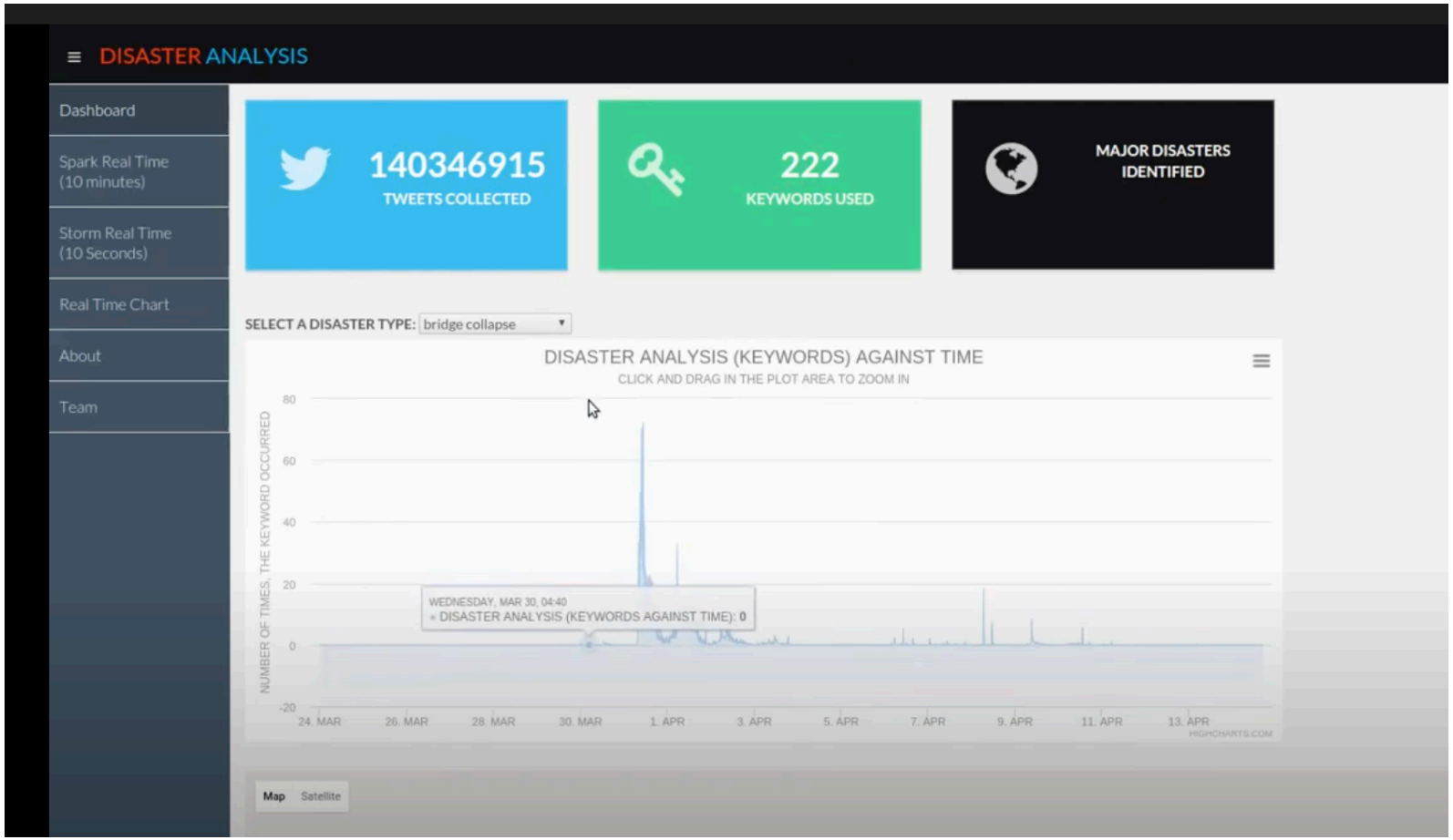


Fig. 1. Proposed architecture and block diagram.

AKBAR et al.: PAs FOR COMPLEX IoT DATA STREAMS

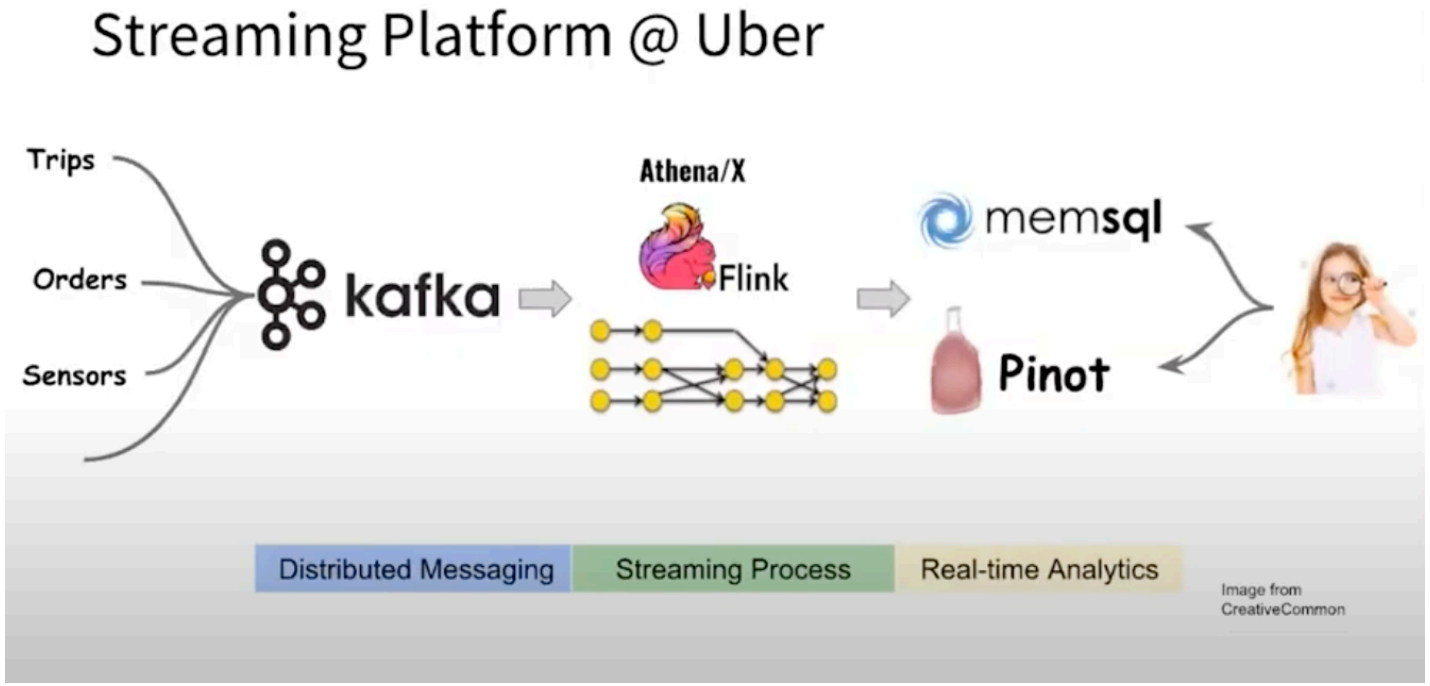
Case study II (Twitter)



https://www.youtube.com/watch?v=Z7wXQmQcQ4Y&t=228s

Case study III (Uber 1/3)

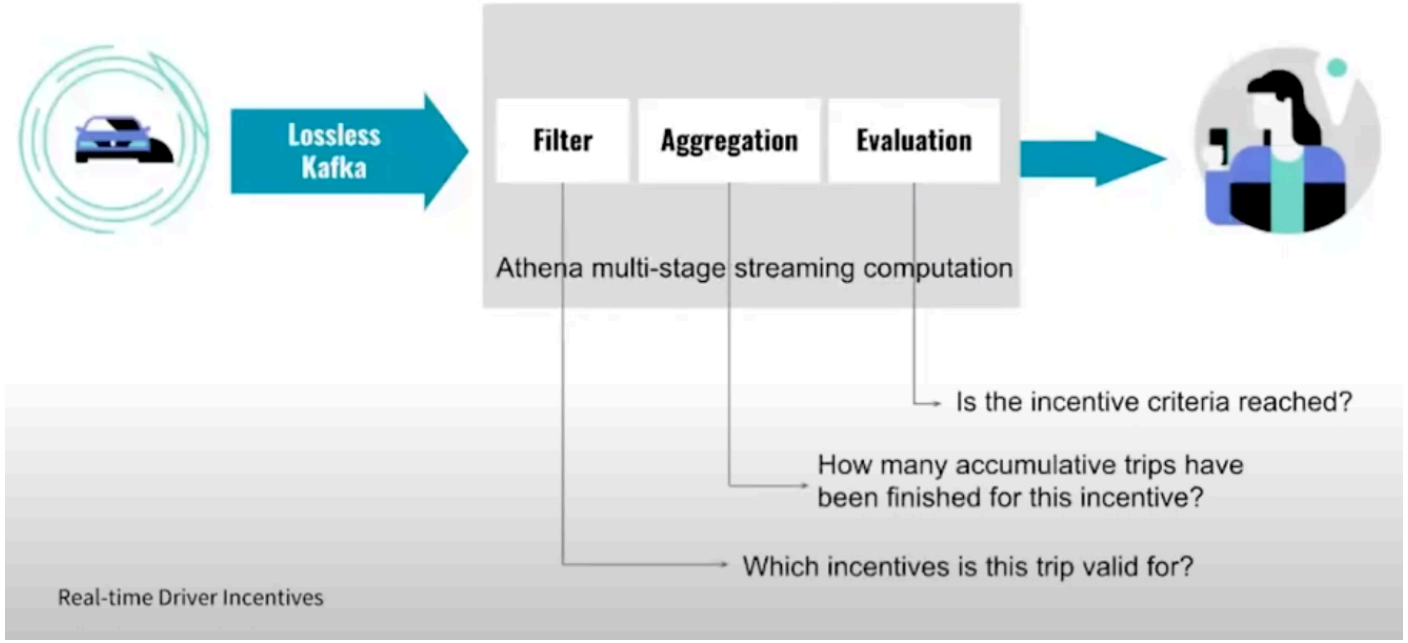
Real-time driver incentives



https://www.youtube.com/watch?v=PKc-RdrW8Ec

Case study III (Uber 2/3)

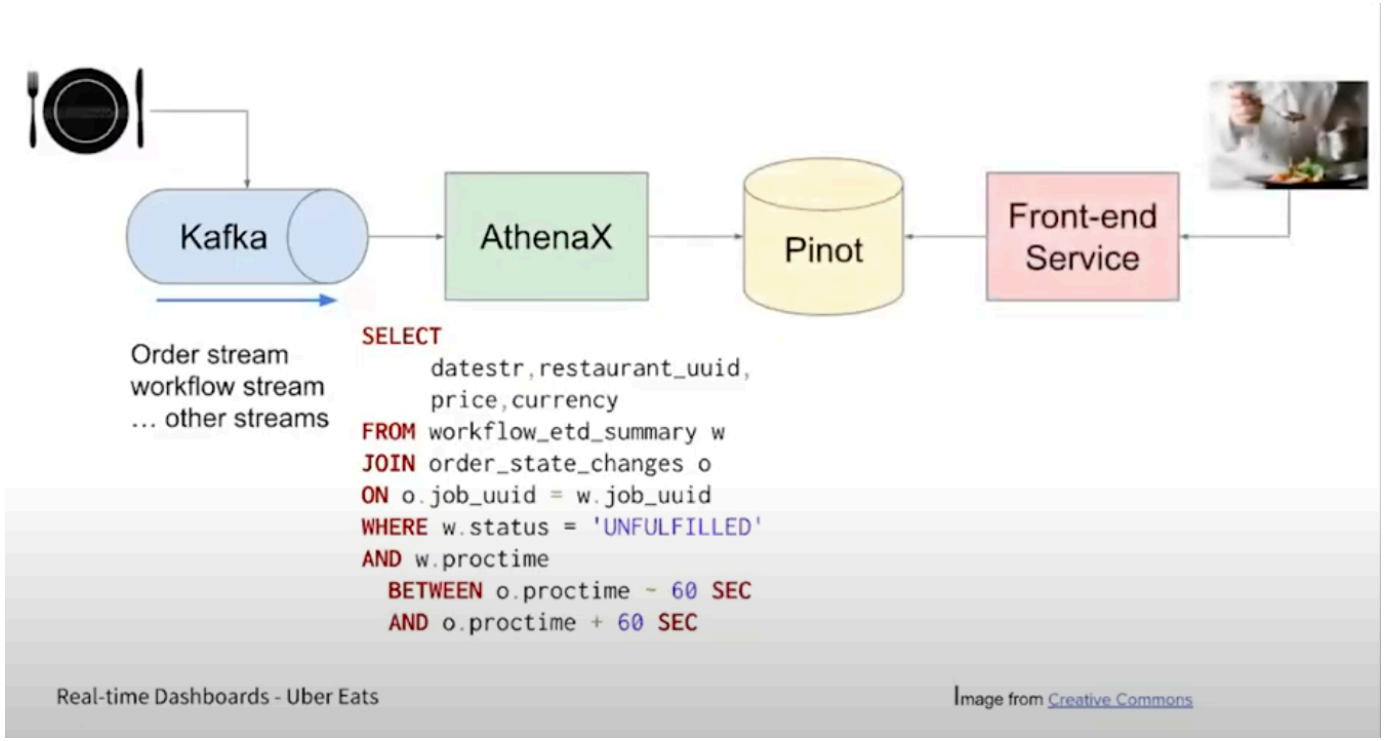
Real-time driver incentives



https://www.youtube.com/watch?v=PKc-RdrW8Ec

Case study III (Uber 3/3)

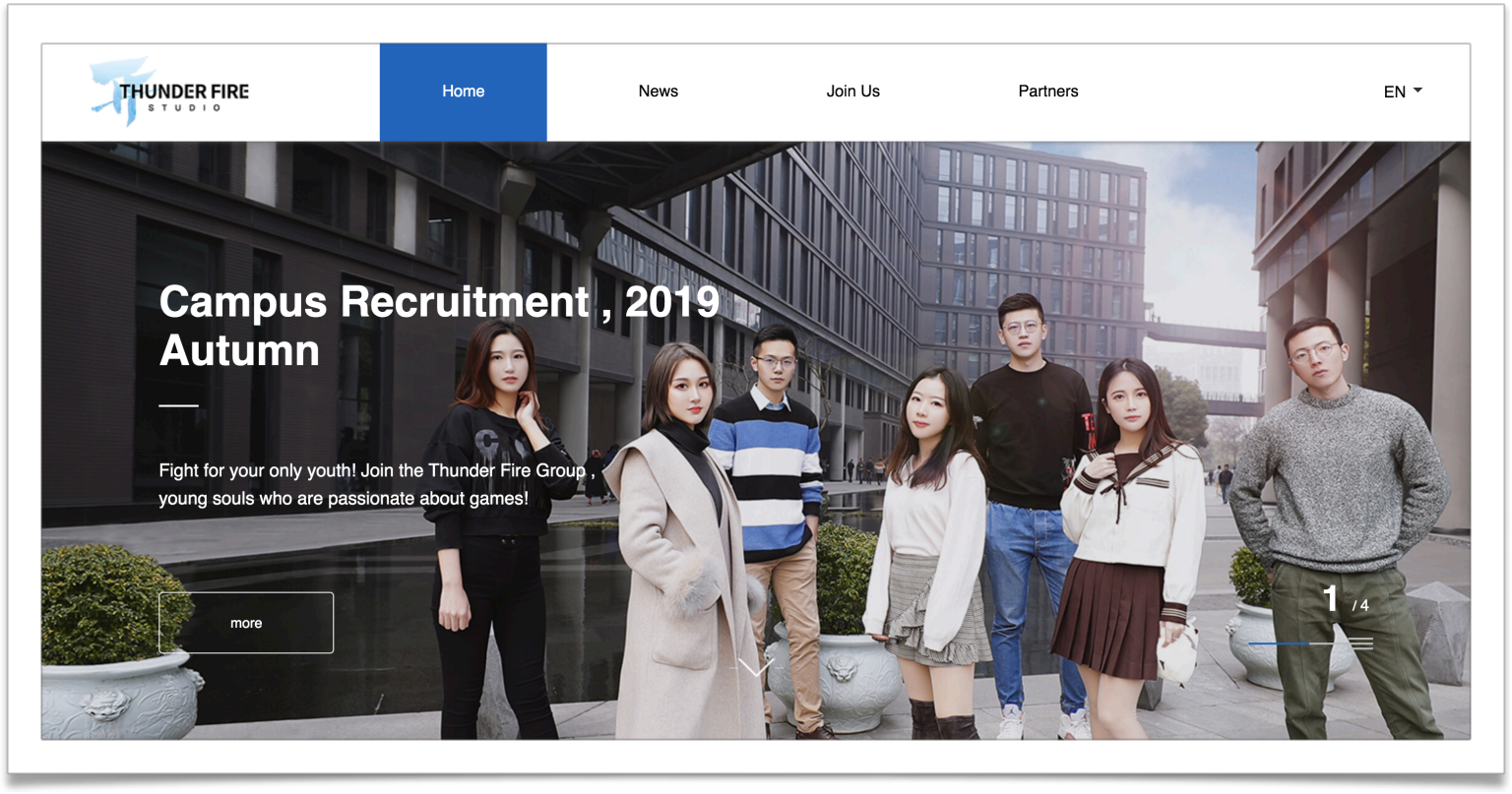
Real-time Dashboard Restaurant manager



https://www.youtube.com/watch?v=PKc-RdrW8Ec

Case study IV (Multiplayer Online Games)

Thunder Fire (Netease)



https://leihuo.163.com/en/index.html

Case study IV (Multiplayer Online Games)

https://www.youtube.com/watch?v=nPaEUYZsFak

Case study IV (Multiplayer Online Games)

Boss： How many new player in our game today?

What is the game revenue today?

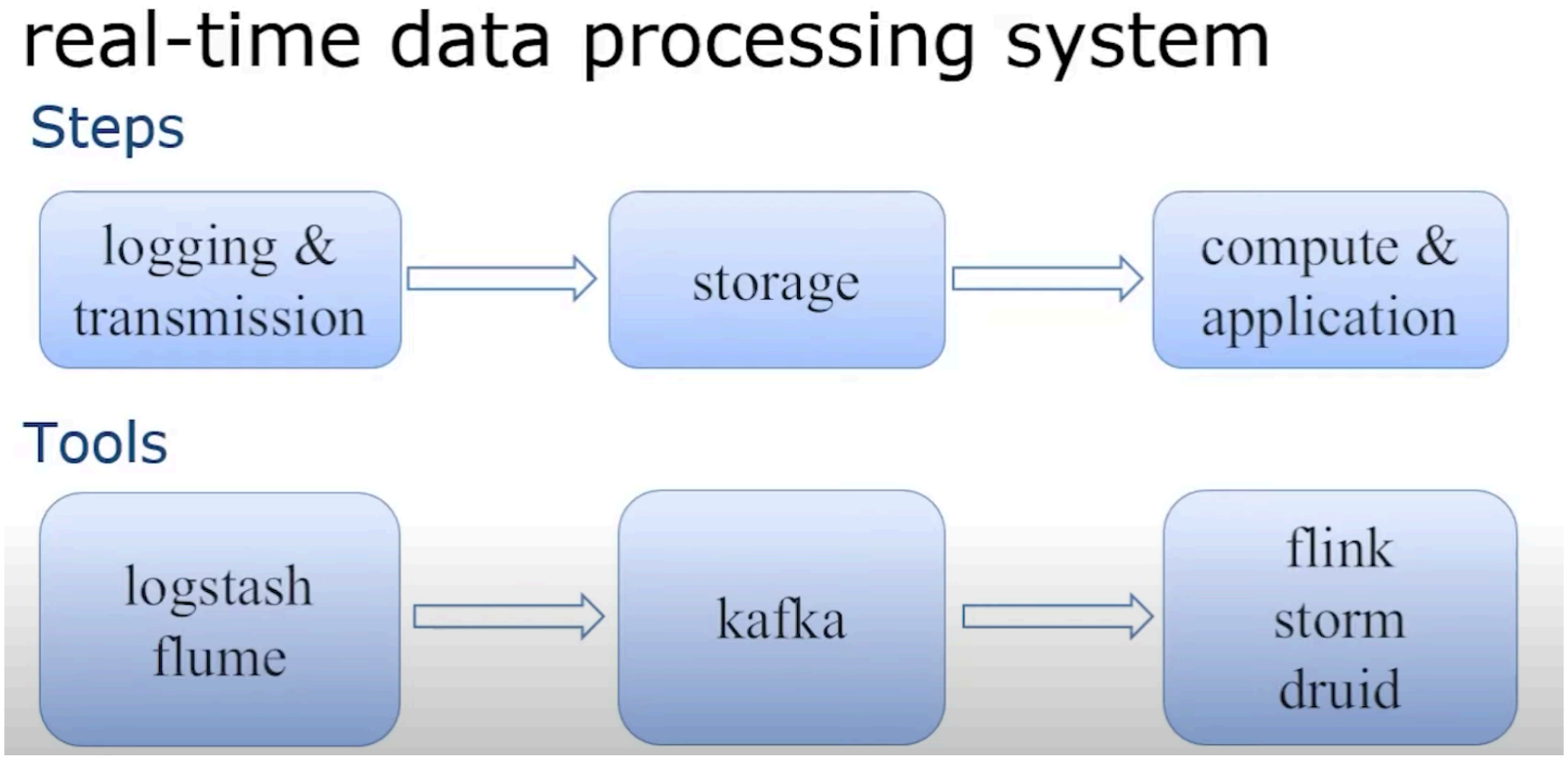
What are the 5 best-selling items in our game today?

Our items were copied by players in an illegal way.

Why didn't we find them in time?

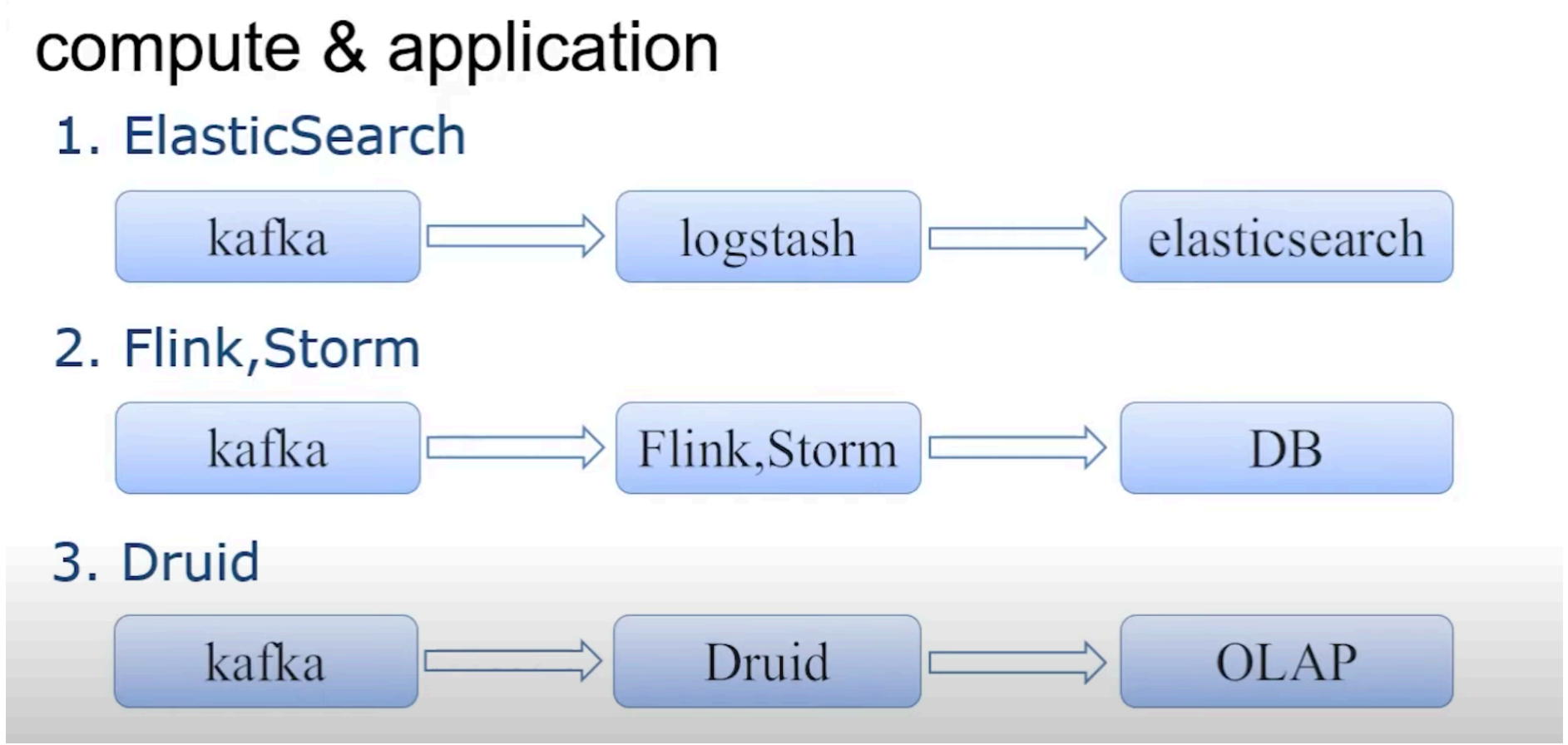
https://www.youtube.com/watch?v=nPaEUYZsFak

Case study IV (Multiplayer Online Games)



https://www.youtube.com/watch?v=nPaEUYZsFak

Case study IV (Multiplayer Online Games)



https://www.youtube.com/watch?v=nPaEUYZsFak

Problem	Description	Application
Sampling [45, 169, 68, 33, 156, 88, 90, 51, 69, 70]	Obtain a representative set of the stream	A/B Testing
Filtering [49, 62, 133, 76, 50, 102, 116, 143, 138, 129, 73, 171, 144, 82]	Extract elements which meet a certain criterion	Set membership
Correlation [163, 146, 134, 165, 99]	Find data subsets (subgraphs) in (graph) data stream which are highly correlated to a given data set	Fraud detection
Estimating Cardinality [86, 46, 78, 92, 85, 89, 54, 112, 103, 59, 157]	Estimate the number of distinct elements	Site audience analysis
Estimating Quantiles [93, 42, 170, 97, 107, 123, 148]	Estimate quantiles of a data stream with small amount of memory	Network analysis
Estimating Moments [39, 63, 109, 48, 96]	Estimating distribution of frequencies of different elements	Databases
Finding Frequent Elements [125, 75, 114, 66, 110, 57, 67, 128, 65, 154, 155, 124, 84, 106, 104, 166, 145, 52, 137]	Identify items in a multiset with frequency more than a threshold θ	Trending Hashtags
Counting Inversions [36]	Estimate number of inversions	Measure sortedness of data
Finding Subsequences [122, 152, 87, 159]	Find Longest Increasing Subsequences (LIS), Longest Common Subsequence (LCS), subsequences similar to a given query sequence	Traffic analysis
Path Analysis [79]	Determine whether there exists a path of length $\leq \ell$ between two nodes in a dynamic graph	Web graph analysis
Anomaly Detection [135, 151, 150, 115, 71, 77, 47, 153, 43]	Detect anomalies in a data stream	Sensor networks
Temporal Pattern Analysis [60, 168, 38]	Detect patterns in a data stream	Traffic analysis
Data Prediction [111, 162, 100, 164, 142, 160]	Predict missing values in a data stream	Sensor data analysis
Clustering [98, 132, 105]	Cluster a data stream	Medical imaging
Graph analysis [83, 101, 35, 113, 127, 61, 80]	Extract unweighted and weighted matching, vertex cover, independent sets, spanners, subgraphs (sparsification) and random walks, computing min-cut	Web graph analysis
Basic Counting [72]	Estimate $\hat{m}$ of the number m of 1-bits in the sliding window (of size n) such that $\lvert\hat{m} - m\rvert \leq \epsilon m$	Popularity Analysis
Significant One Counting [119]	Estimate $\hat{m}$ of the number m of 1-bits in the sliding window (of size n) such that if $m \geq \theta n$, then $\lvert\hat{m} - m\rvert \leq \epsilon m$	Traffic accounting [81]

Table 1: Streaming algorithms and their applications

End