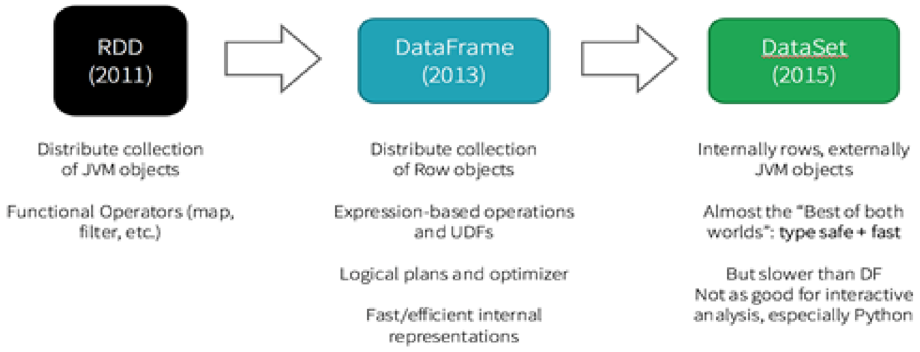
Big Data Real-time Processing with Apache Spark – (II)

Outline

- Spark SQL
- Spark Streaming
- Spark MLlib
- Spark GraphX
- Lambda architecture

Apache Spark

Spark data structures



RDD
(2011)

Distribute collection
of JVM objects

Functional Operators (map,
filter, etc.)

DataFrame
(2013)

Distribute collection
of Row objects

Expression-based operations
and UDFs

Logical plans and optimizer

Fast/efficient internal
representations

DataSet
(2015)

Internally rows, externally
JVM objects

Almost the "Best of both
worlds": type safe + fast

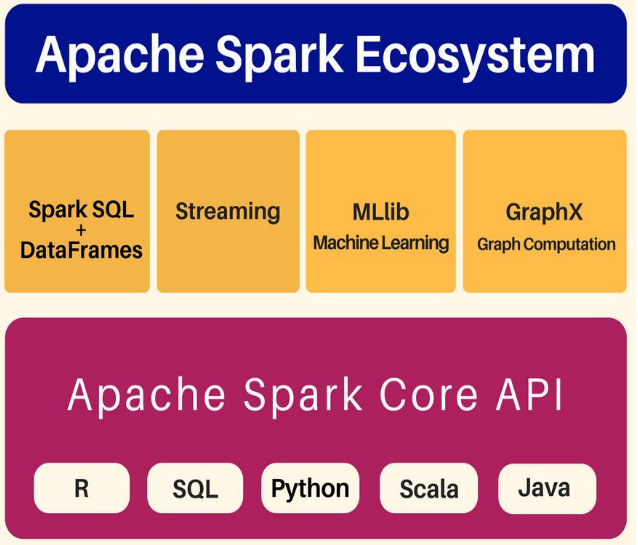
But slower than DF
Not as good for interactive
analysis, especially Python

- **RDD** – a fault-tolerant collection of elements that can be operated on in parallel.
- **DataFrame** – a dataset organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations
- **Dataset** – an extension to DataFrames that provides a type-safe, object-oriented programming interface. It is a strongly-typed, immutable collection of objects that are mapped to a relational schema.

Understanding data interfaces

Apache Spark

Spark ecosystem



Apache Spark Ecosystem

Spark SQL + DataFrames

Streaming

MLlib Machine Learning

GraphX Graph Computation

Apache Spark Core API

R

SQL

Python

Scala

Java

VIDEO Spark components

- **Spark core** – execution engine of Spark platform, providing memory management, task scheduling, fault recovery, etc.
- **Spark SQL** – Spark SQL performs the query on data through SQL and HQL (Hive Query Language) based on DataFrame model.
- **Spark streaming** – live stream data processing
- **Spark Mlib** – in-built library of Spark that contains the functionality of Machine Learning, providing various ML algorithms
- **Spark GraphX** – graph computation engine built on top of Apache Spark that enables to process graph data at scale.
- **Spark API** – Spark provides API in Scala, Java, Python and R.
  - **PySpark** is the Python package to use Apache Spark.

Spark SQL

- **Spark SQL** is Apache Spark's module for working with structured and semi-structured data.

- Spark SQL originated as Apache Hive to run on top of Spark and is now integrated with the Spark stack.

- Spark SQL blurs the line between RDD and relational table. It offers much tighter integration between relational and procedural processing, through declarative DataFrame APIs which integrates with Spark code.

- It also provides higher optimization. DataFrame API and Datasets API are the ways to interact with Spark SQL.

EXAMPLE

Features

- Integration with Spark

  Query structured data inside Spark programs, using either SQL or a familiar DataFrame API.

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Apply functions to results of SQL queries.

- Uniform data access

  DataFrames and SQL support a common way to access a variety of data sources. This joins the data across these sources. This is very helpful to accommodate all the existing users into Spark SQL

```
spark.read.json("s3n://...")
  .registerTempTable("json")
results = spark.sql(
  """SELECT *
     FROM people
     JOIN json ...""")
```

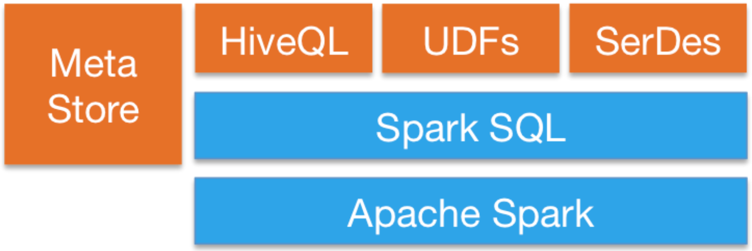
Query and join different data sources.

Spark SQL

Features

- **Hive integration**

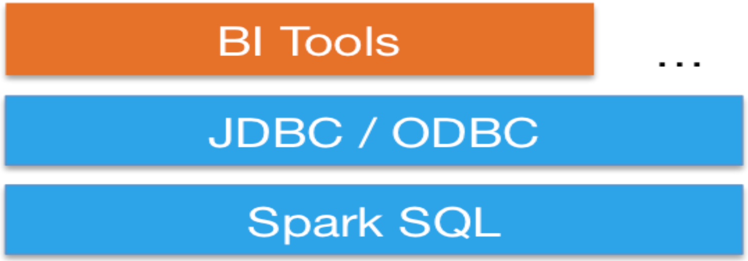
  Spark SQL supports the HiveQL syntax and has full compatibility with current Hive data, queries, and UDFs, allowing access existing Hive warehouses



Spark SQL can use existing Hive metastores, SerDes, and UDFs.

- **Standard connectivity**

  A server mode provides industry standard JDBC and ODBC connectivity for business intelligence tools.

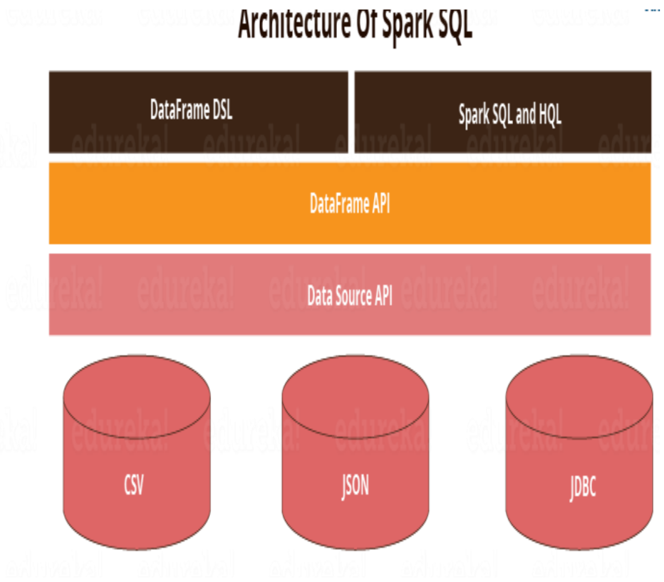


Use existing BI tools to query big data.

Spark SQL

Spark SQL Libraries

Spark SQL has the four libraries which are used to interact with relational and procedural processing



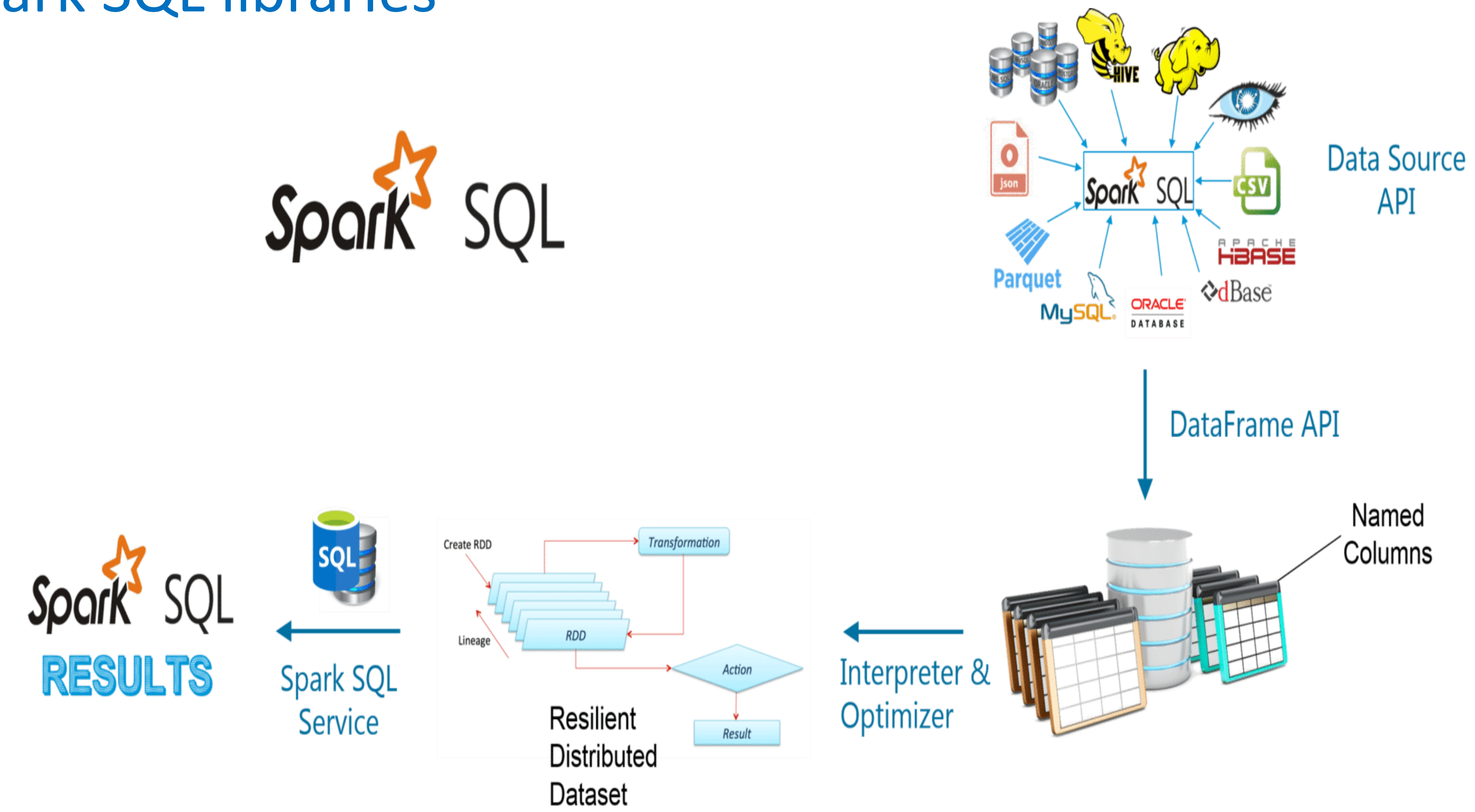
1. **Data Source API**
   A universal API for loading and storing structured data. Supports different data formats (CSV, JSON, etc.) and storage systems (e.g. HDFS, MySQL, Hive table, etc.)

2. **DataFrame API:**
   A distributed collection of data organized into named column, equivalent to a relational table in SQL used for storing data into tables.

3. **SQL Interpreter And Optimizer:**
   Performs analysis/evaluation, optimization, planning, and runtime code spawning

4. **SQL Service**
   The entry point for working along structured data in Spark. It allows the creation of DataFrame objects as well as the execution of SQL queries.

Spark SQL libraries


Data Source API

DataFrame API

Named Columns

Interpreter & Optimizer

Resilient Distributed Dataset

Spark SQL Service

RESULTS

Flow diagram represents a Spark SQL process using all four libraries in sequence

DataFrame

- Different from RDD which offers low-level functionality and control, DataFrame offers high-level custom view and structure, domain-specific operations, saves space, and executes at high speed.

- DataFrames are similar to the table in a relational database or data frame in R /Python. DataFrame contains rows with Schema. The **schema** is the illustration of the structure of data.

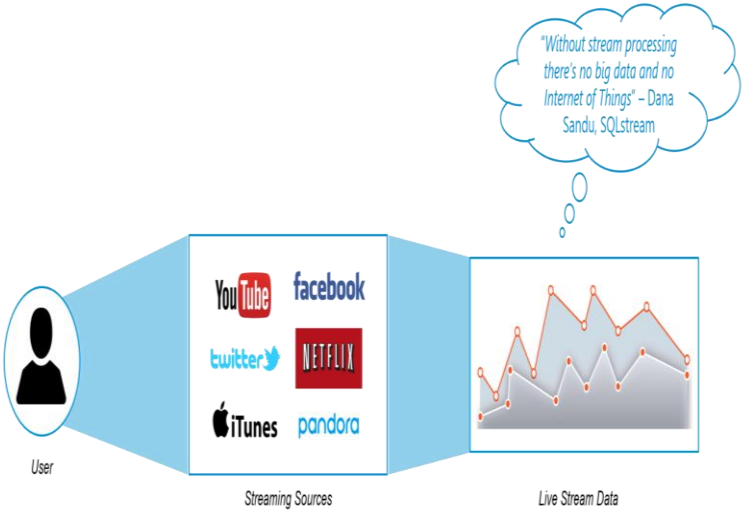
Spark SQL

INTRODUCTION TO SPARK SQL IN PYTHON

- Similar to RDD, DataFrame is **immutability**, **in-memory**, resilient, distributed computing capability. It allows the user to impose the structure onto a distributed collection of data. Thus provides higher level abstraction.

- A Dataframe can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs

APACHE Spark Streaming


"Without stream processing there's no big data and no Internet of Things" – Dana Sandu, SQLstream
User
YouTube
facebook
twitter
NETFLIX
iTunes
pandora
Streaming Sources
Live Stream Data

- Data Streaming is a technique for transferring data so that it can be processed as a steady and continuous stream

twitter
Spark Streaming

Spark Streaming is used to stream real-time data from various sources like Twitter, Stock Market and Geographical Systems and perform powerful analytics to help businesses.

- *Spark Streaming* is used for processing real-time streaming data. Spark Streaming enables high-throughput and fault-tolerant stream processing of live data

APACHE Spark Streaming

EXAMPLE

Features

- Ease of use

  Spark Streaming brings high-level APIs for stream processing, letting you write streaming jobs the same way you write batch jobs.

- Fault tolerance

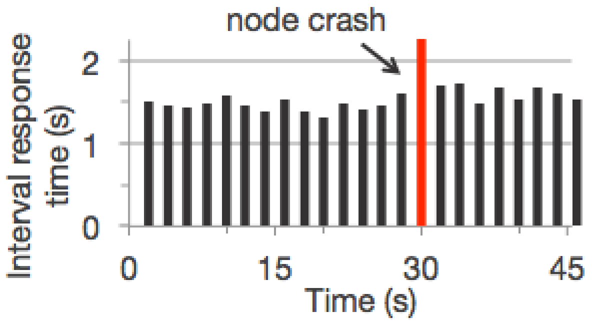
  Spark Streaming recovers both lost work and operator state (e.g. sliding windows) out of the box, without any extra code on your part.

- Spark integration

  Combine streaming with batch and interactive queries.

```
TwitterUtils.createStream(...)
  .filter(_.getText.contains("Spark"))
  .countByWindow(Seconds(5))
```

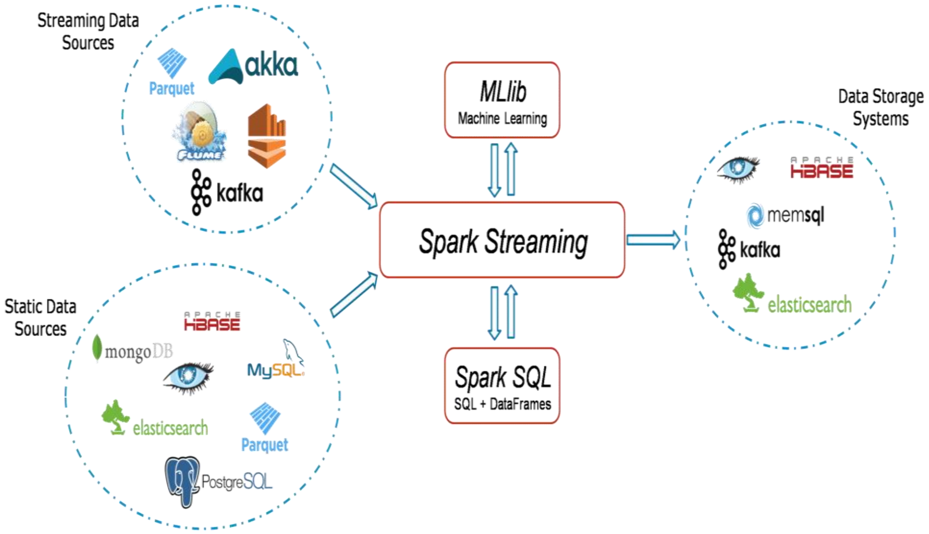
Counting tweets on a sliding window

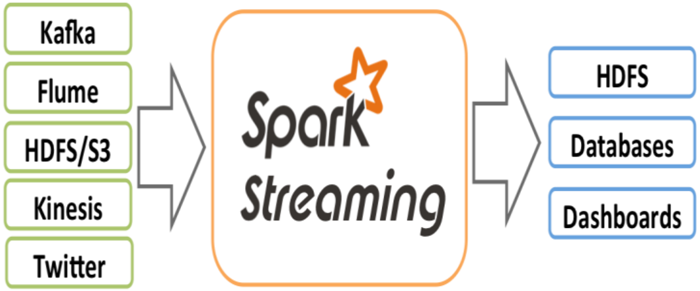


```
stream.join(historicCounts).filter {
 case (word, (curCount, oldCount)) =>
  curCount > oldCount
}
```

Find words with higher frequency than historic data

Spark Streaming Workflow





- Stream data from various sources (both streaming data sources like Kafka, Flume, AWS or Parquet for real-time streaming, or batch data sources like HBase, MySQL, PostgreSQL, Elastic Search, Mongo DB and Cassandra for static/batch streaming.

- Spark can be used to perform Machine Learning on the data through its MLlib API.

- Spark SQL is used to perform further operations on this data.

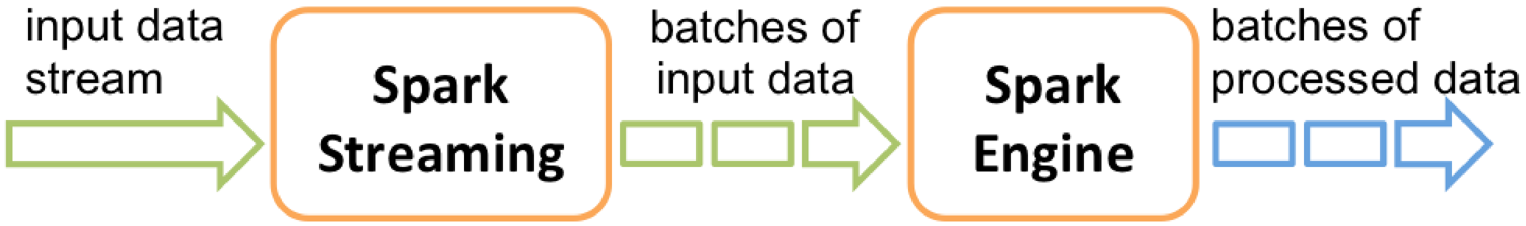
- Finally, the streaming output can be stored into various data storage systems like HBase, Cassandra, MemSQL, Kafka, Elastic Search, HDFS and local file system.

Two flavours of Spark Streaming

- **Spark Streaming (Dstreams)**
  - Introduced with Spark 0.7.0
  - Based on the idea of discretized streams or DStreams
  - Typically used for low-level transformations and processing

- **Structured streaming**
  - Introduced in Spark 2.0 as an extension built on top of Spark SQL.
  - Takes advantage of Spark SQL code and memory optimizations.
  - Structured Streaming also gives very powerful abstractions like Dataset/DataFrame APIs as well as SQL.
  - No more dealing with RDD directly

Spark Streaming - workflow


input data stream

Spark Streaming

batches of input data

Spark Engine

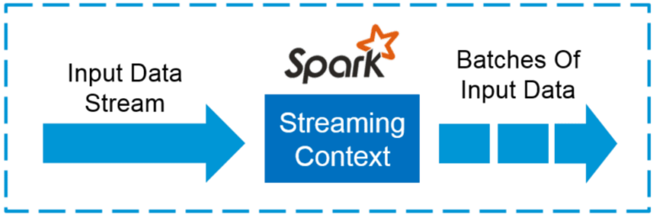
batches of processed data

- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

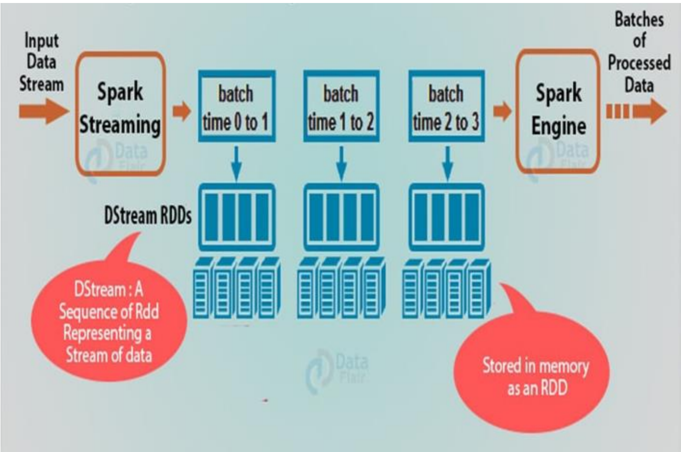
- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

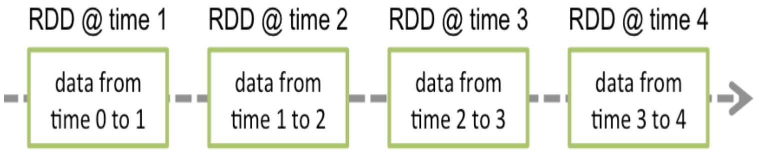
- Spark Streaming provides a high-level abstraction called ***discretized stream*** or ***DStream***, which represents a continuous stream of data. DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams.

Spark Streaming - components



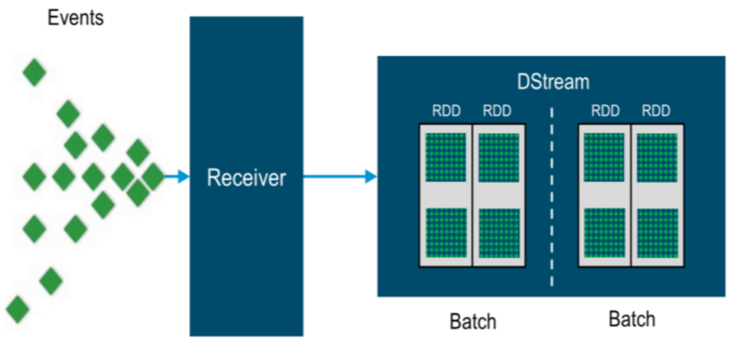
- **Streaming Context** consumes a stream of data in Spark. It registers an *Input DStream* to produce a *Receiver* object. It is the main entry point for Spark functionality.


Input Data Stream
Spark Streaming
batch time 0 to 1
batch time 1 to 2
batch time 2 to 3
Spark Engine
Batches of Processed Data
DStream RDDs
DStream : A Sequence of Rdd Representing a Stream of data
Stored in memory as an RDD

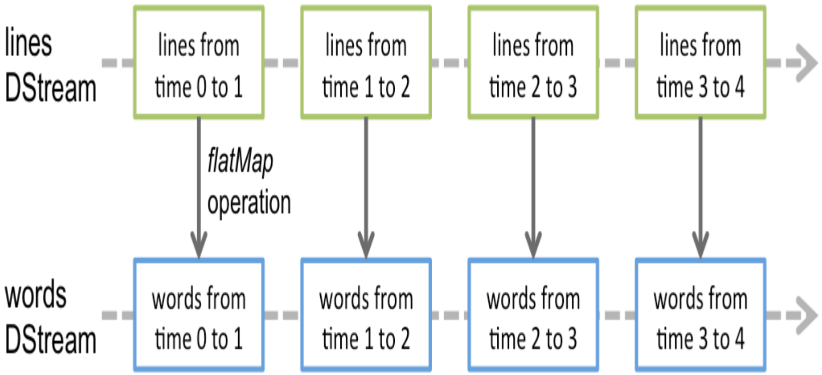


- The fundamental stream unit **DStream** is basically a series of RDDs to process the real-time data.
- It is a continuous stream of data. It is received from a data source or a processed data stream generated by transforming the input stream.
- Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval.

Spark Streaming - components

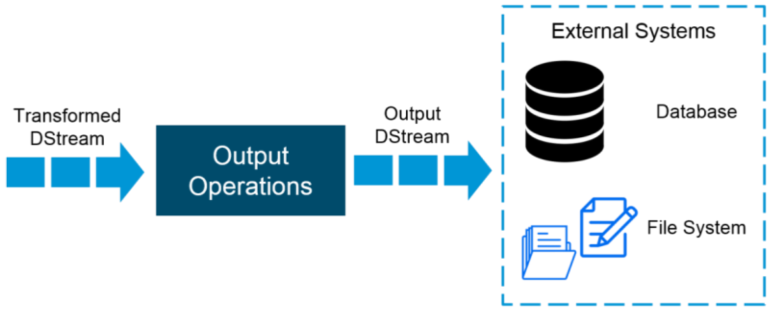


- **Input DStreams** are DStreams representing the stream of input data received from streaming sources.
- Every input DStream is associated with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.

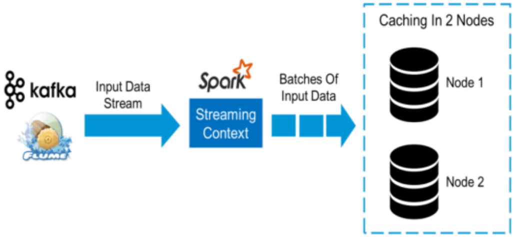


- Any operation applied on a DStream translates to operations on the underlying RDDs. Transformations allow the data from the input DStream to be modified similar to RDDs. DStreams support many of the transformations available on normal Spark RDDs.

Spark Streaming - components



- **Output operations** allow DStream's data to be pushed out to external systems like databases or file systems. Output operations trigger the actual execution of all the DStream transformations.



- *DStreams* allow developers to **cache**/ persist the stream's data in memory. This is useful if the data in the DStream will be computed multiple times. This can be done using the *persist()* method on a DStream.

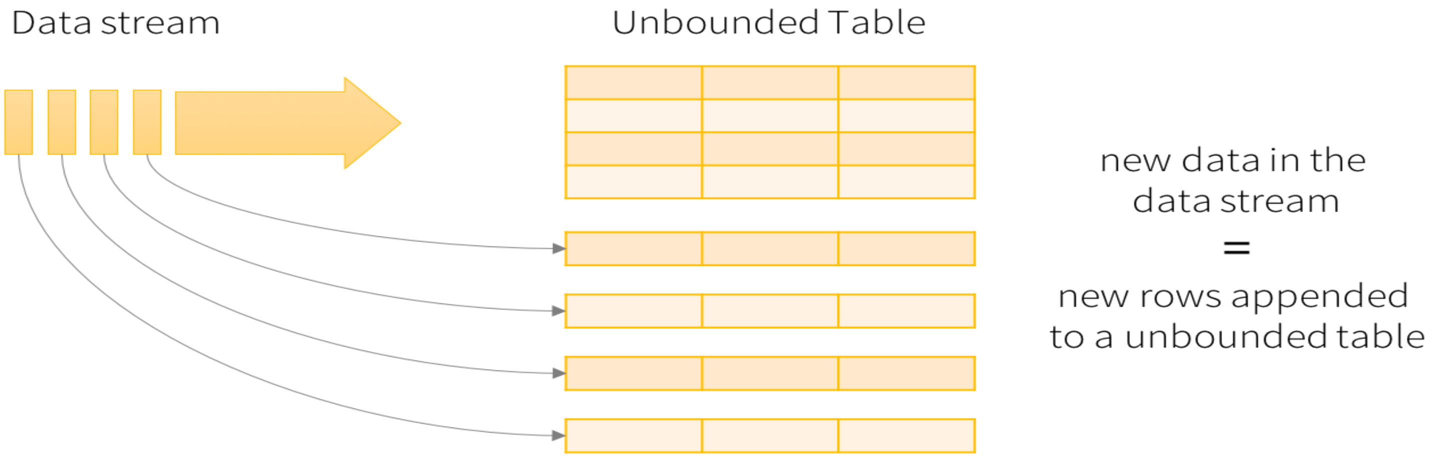
Structured streaming

- Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.

- Treat a live data stream as a table that is being continuously appended. This leads to a new stream processing model that is very similar to a batch processing model.

- Express streaming computation as standard batch-like query as on a static table, and Spark runs it as an incremental query on the unbounded input table.

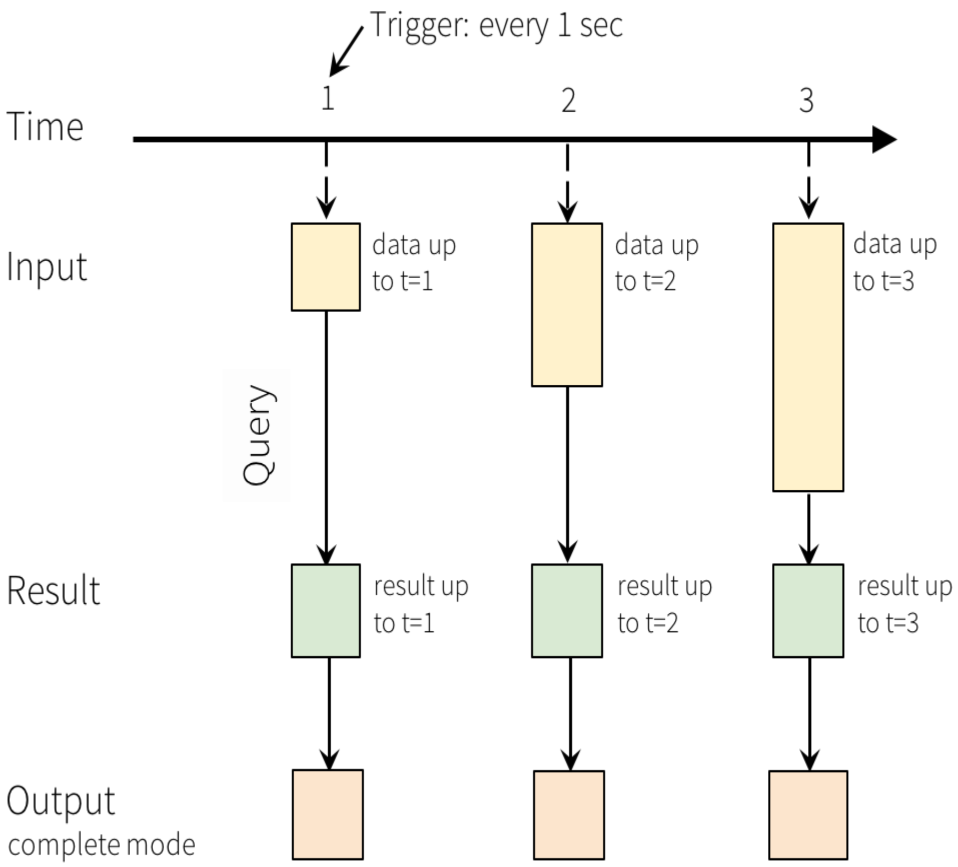
Structured streaming


Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table

Data stream as an unbounded table

- Treat input data stream as the “Input Table”.
- Every data item that is arriving on the stream is like a new row being appended to the Input Table.

Structured streaming

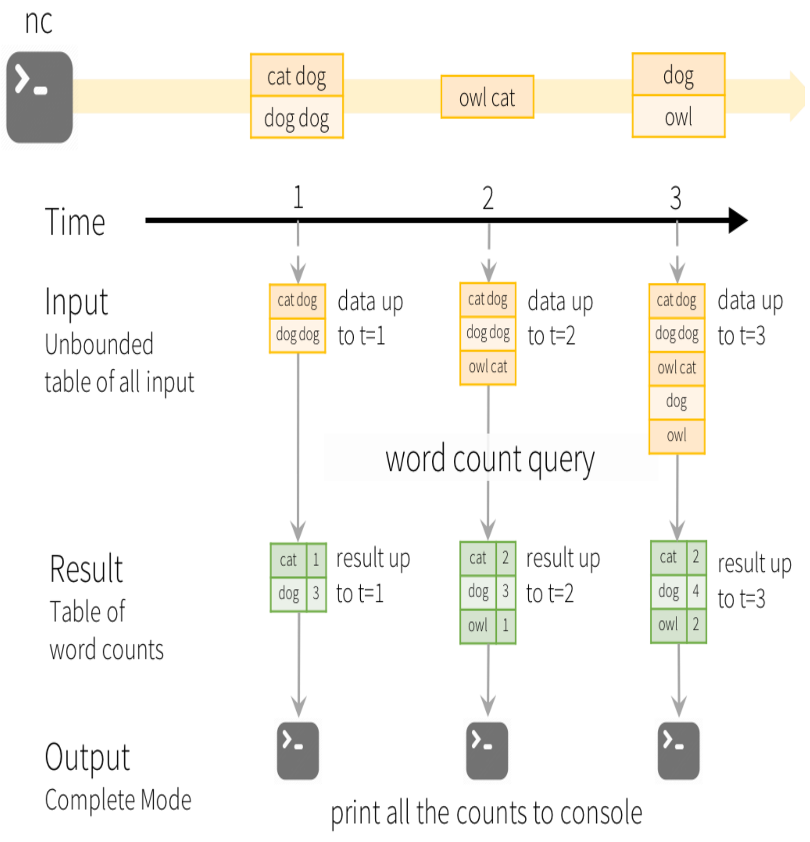


Programming Model for Structured Streaming

- A query on the input will generate the “Result Table”.
- Every trigger interval (say, every 1 second), new rows get appended to the Input Table, which eventually updates the Result Table.
- Whenever the result table gets updated, we would want to write the changed result rows to an external sink.

Spark Streaming

Structured streaming



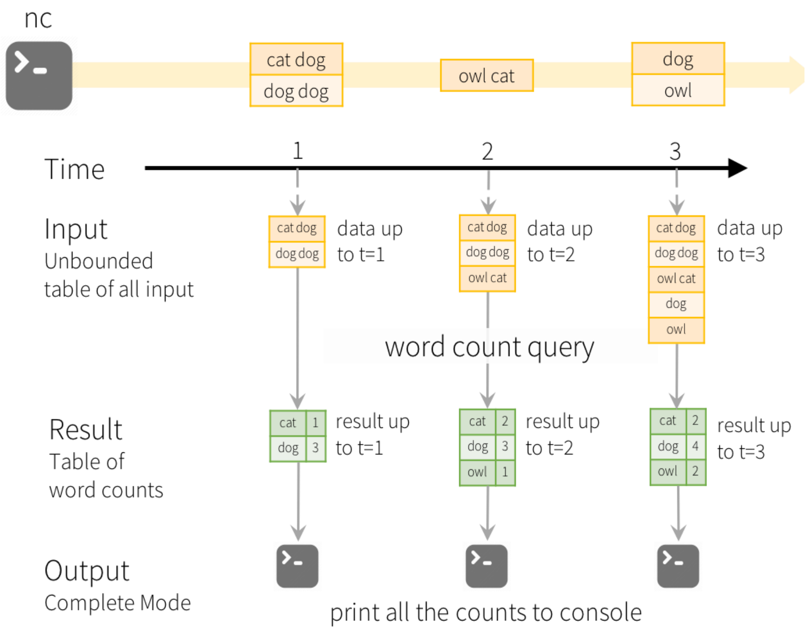
Model of the Quick Example

WordCount example with PySpark

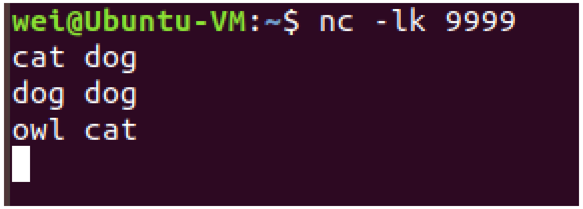
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

spark = SparkSession.builder.appName("WordCount").getOrCreate()

# Create DataFrame representing the stream of input lines from connection to localhost:9999
lines = spark.readStream.format("socket").option("host", "localhost") \
        .option("port", 9999).load()

# Split the lines into words
words = lines.select(explode(split(lines.value, " ")).alias("word"))

# Generate running word count
wordCounts = words.groupBy("word").count()

# Start running the query that prints the running counts to the console
query = wordCounts.writeStream.outputMode("complete") \
            .format("console").start()

query.awaitTermination()
```

Streaming

Structured streaming

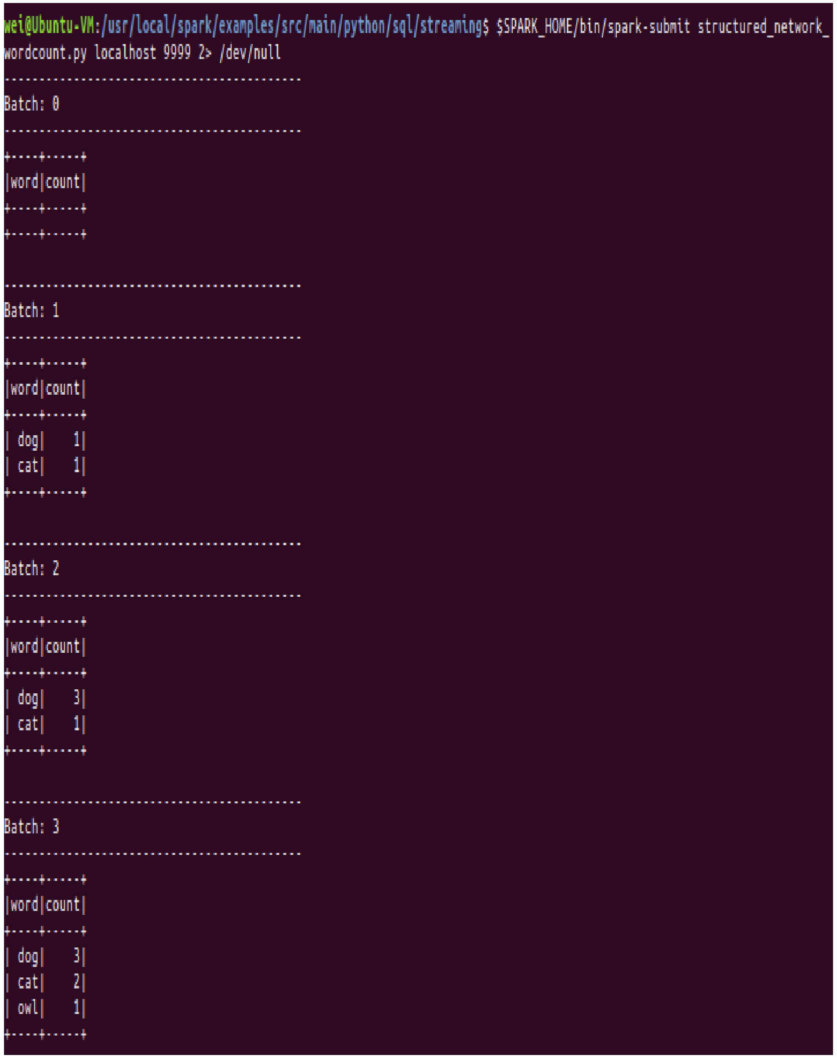


Model of the Quick Example



```
wei@Ubuntu-VM:~$ nc -lk 9999
cat dog
dog dog
owl cat
```

Run Netcat (a Linux utility) as a data server on port 9999



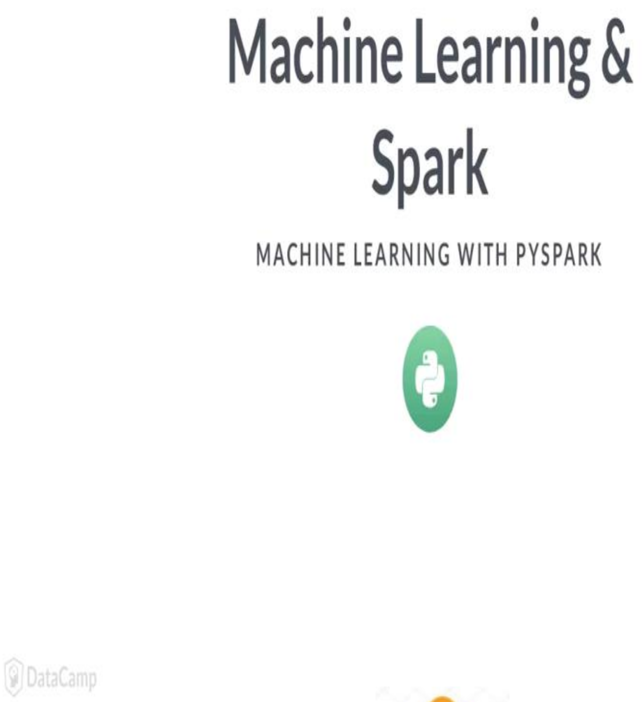
```
wei@Ubuntu-VM:/usr/local/spark/examples/src/main/python/sql/streaming$ $SPARK_HOME/bin/spark-submit structured_network_
wordcount.py localhost 9999 2> /dev/null
-------------------------------------------
Batch: 0
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
+----+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
| dog|    1|
| cat|    1|
+----+-----+

-------------------------------------------
Batch: 2
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
| dog|    3|
| cat|    1|
+----+-----+

-------------------------------------------
Batch: 3
-------------------------------------------
+----+-----+
|word|count|
+----+-----+
| dog|    3|
| cat|    2|
| owl|    1|
+----+-----+
```

Run Spark streaming receiving data flow from port 9999

APACHE Spark MLlib

Spark MLlib is used to perform machine learning in Apache Spark. It provides the following tools:

- **ML Algorithms:** ML Algorithms form the core of MLlib. These include common learning algorithms such as classification, regression, clustering and collaborative filtering.
- **Featurization:** Featurization includes feature extraction, transformation, dimensionality reduction and selection.
- **Pipelines:** Pipelines provide tools for constructing, evaluating and tuning ML Pipelines.
- **Persistence:** Persistence helps in saving and loading algorithms, models and Pipelines.
- **Utilities:** Utilities for linear algebra, statistics and data handling.



APACHE Spark MLlib

EXAMPLE

Features

- Ease of use

  Spark Mlib brings high-level APIs in Java, Python, Scala and R. You can use any Hadoop data source (e.g. HDFS, HBase, or local files), making it easy to plug into Hadoop workflows.

- Performance

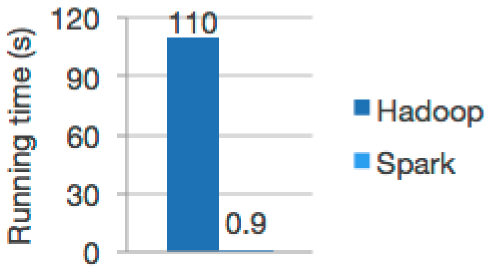
  MLlib contains high-quality algorithms that leverage iteration, and can run 100x faster than MapReduce

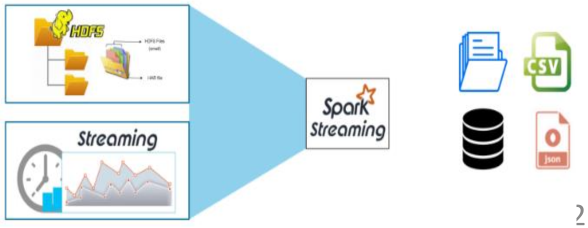
- Run everywhere

  Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud, against diverse data sources like HDFD, Apache Cassandra, Hive, and others.

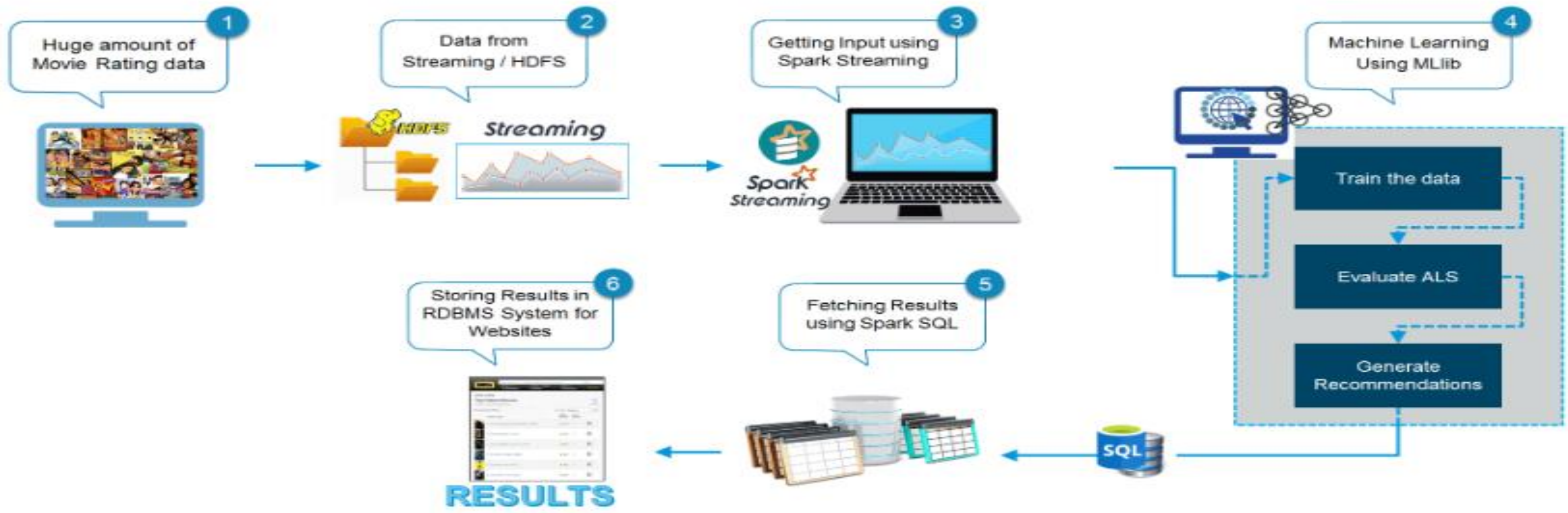
```
data = spark.read.format("libsvm")\
  .load("hdfs://...")

model = KMeans(k=10).fit(data)
```

Calling MLib





Use Case – Movie Recommendation System


1
Huge amount of Movie Rating data
2
Data from Streaming / HDFS
HDFS
Streaming
3
Getting Input using Spark Streaming
Spark Streaming
4
Machine Learning Using MLlib
Train the data
Evaluate ALS
Generate Recommendations
SQL
5
Fetching Results using Spark SQL
6
Storing Results in RDBMS System for Websites
RESULTS

❶ ❷ ❸ Stream the data live from movies websites or download and store them in local file system or HDFS

❹ The whole recommendation system is based on ML algorithm *Alternating Least Squares* (ALS), and use Collaborative Filtering (CF) to predict the ratings for users for particular movies based on their ratings for other movies.

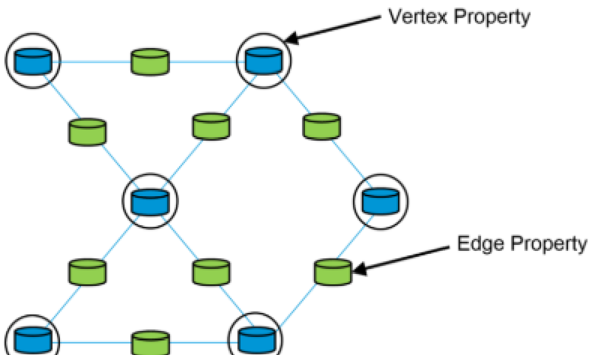
❺ Use Spark SQL's DataFrame, Dataset and SQL Service to fetch ML results.

❻ Store the results on RDBMS to display on websites

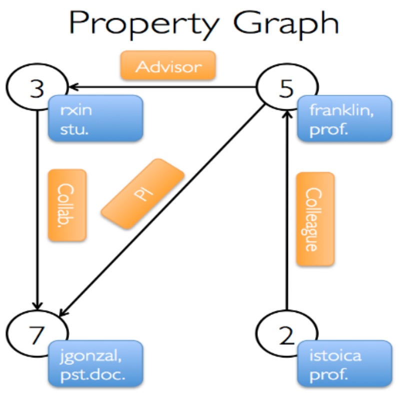
Spark GraphX

- **GraphX** is the Spark API for graphs and graph-parallel computation. It includes a growing collection of graph algorithms and builders to simplify graph analytics task

- GraphX extends the Spark RDD with a **Resilient Distributed Property Graph**.
  - The property graph is a directed multi-graph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. The parallel edges allow multiple relationships between the same vertices.



Property graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Example Property graph

APACHE Spark GraphX

EXAMPLE

Features

- Flexibility

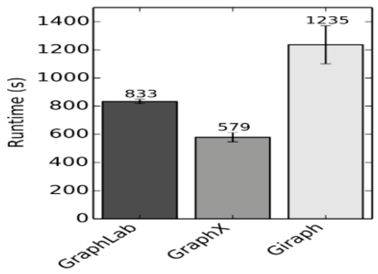
  GraphX works with both graphs and computations. Same data can be viewed as both graphs and collections (e.g. RDD). Transform and join graphs with RDDs efficiently and write custom iterative graph algorithms using the Pregel API

```
graph = Graph(vertices, edges)
messages = spark.textFile("hdfs://...")
graph2 = graph.joinVertices(messages) {
  (id, vertex, msg) => ...
}
```

Using GraphX in Scala

- Speed

  Comparable with the fastest graph systems while retaining Spark's flexibility, fault tolerance and ease of use.

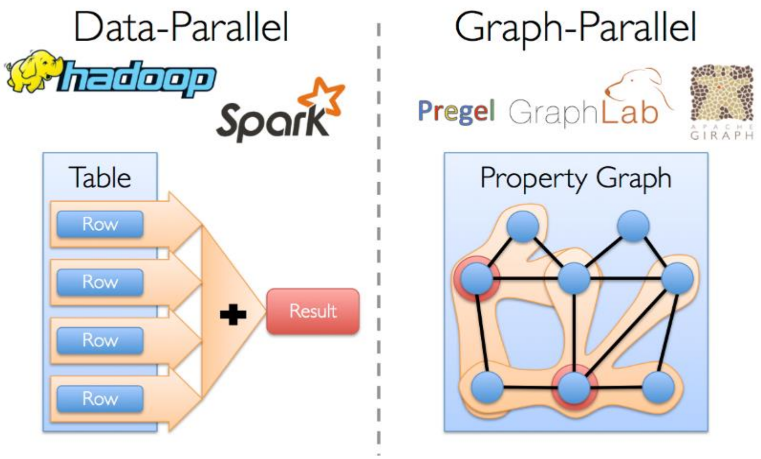


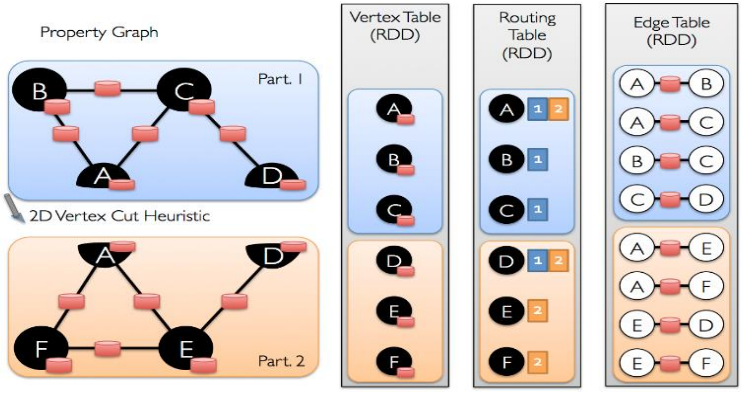
- Algorithms

  GraphX comes with a variety of graph algorithms

- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count

Graph parallel computation





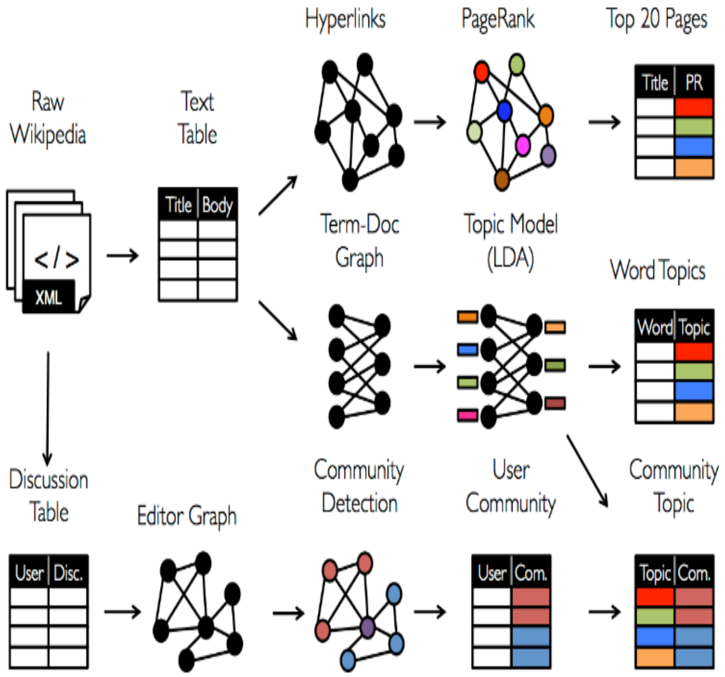
Graphs partitions

- GraphX partitions and distributes graphs, and it can efficiently execute sophisticated graph algorithms orders of magnitude faster than more general *data-parallel* system.

- GraphX restricts the types of computation that can be expressed, and it is optimized for iterative diffusion algorithms like PageRank, but not well suited to more basic tasks like constructing the graph, modifying its structure, or expressing computation that spans multiple graphs.

Apache Spark GraphX

Graph parallel computation



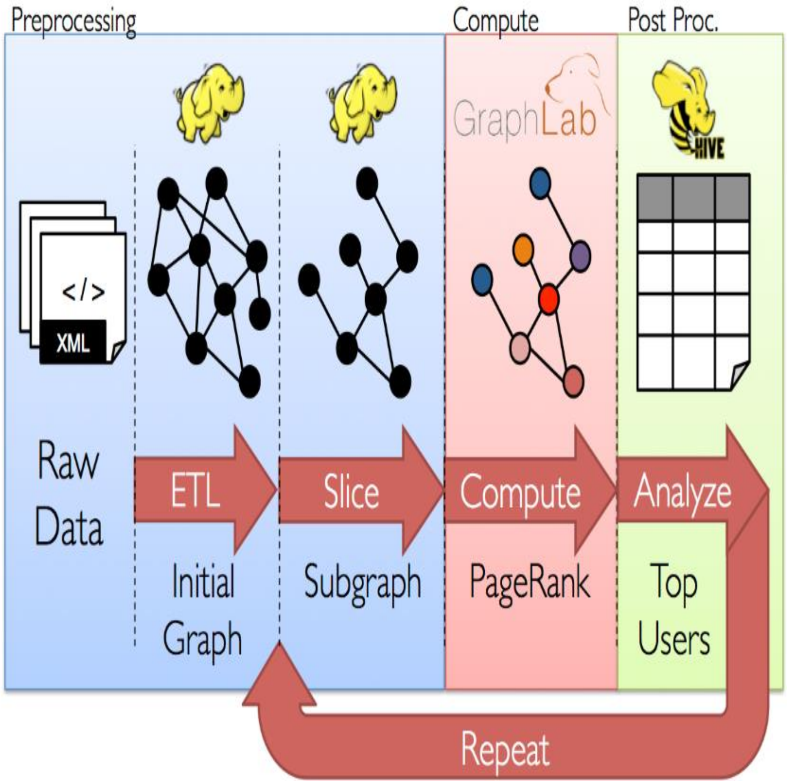
Tables and graphs

- GraphX tasks typically require data-movement outside of the graph topology and are often more naturally expressed as operations on tables in more traditional data-parallel systems.

- How we look at data depends on our objectives and the same raw data may require many different table and graph views throughout the analysis process

Apache Spark GraphX

Graph parallel computation

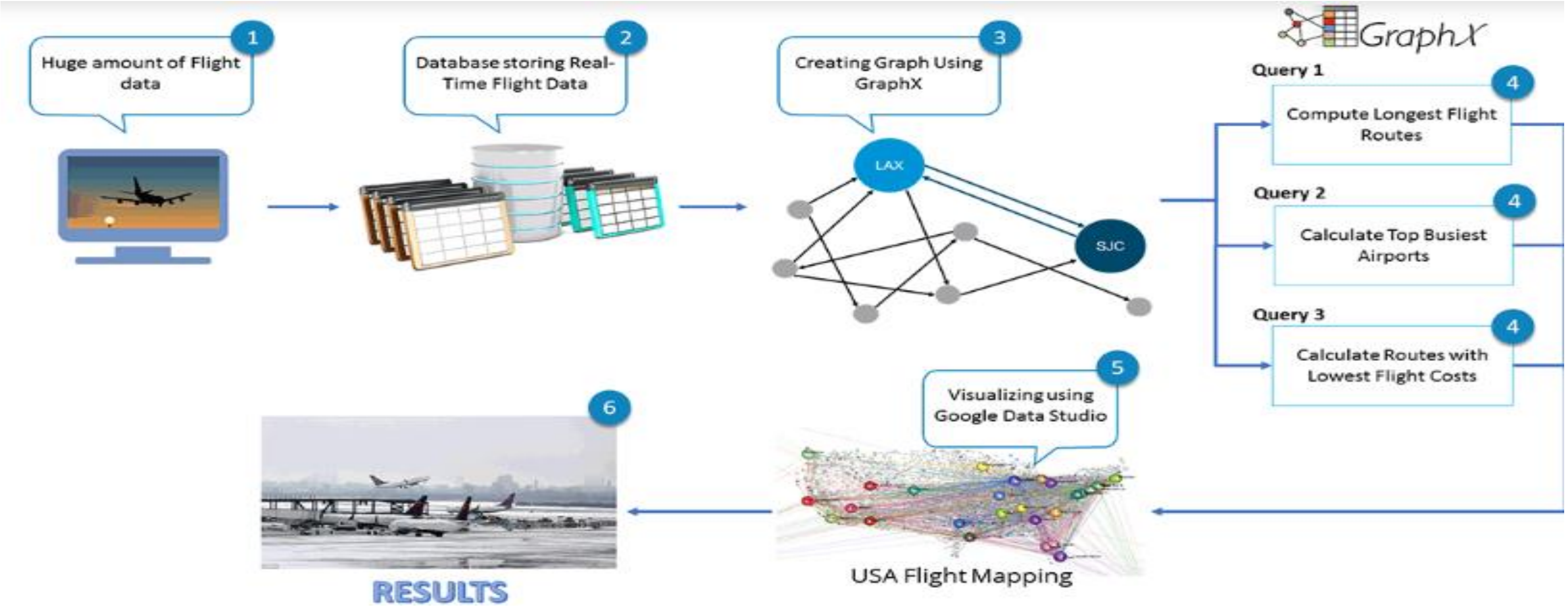


Existing graph analytics pipeline

- Existing graph analytics pipelines compose graph-parallel and data-parallel systems, leading to extensive data movement and duplication and a complicated programming model.

- GraphX unify graph-parallel and data-parallel computation in one system with a single composable API. exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API

Use Case – Flight Data Analysis using Spark GraphX


1
Huge amount of Flight data
2
Database storing Real-Time Flight Data
3
Creating Graph Using GraphX
LAX
SJC
GraphX
Query 1
4
Compute Longest Flight Routes
Query 2
4
Calculate Top Busiest Airports
Query 3
4
Calculate Routes with Lowest Flight Costs
5
Visualizing using Google Data Studio
USA Flight Mapping
6
RESULTS

❶ ❷ Stream the data to databses

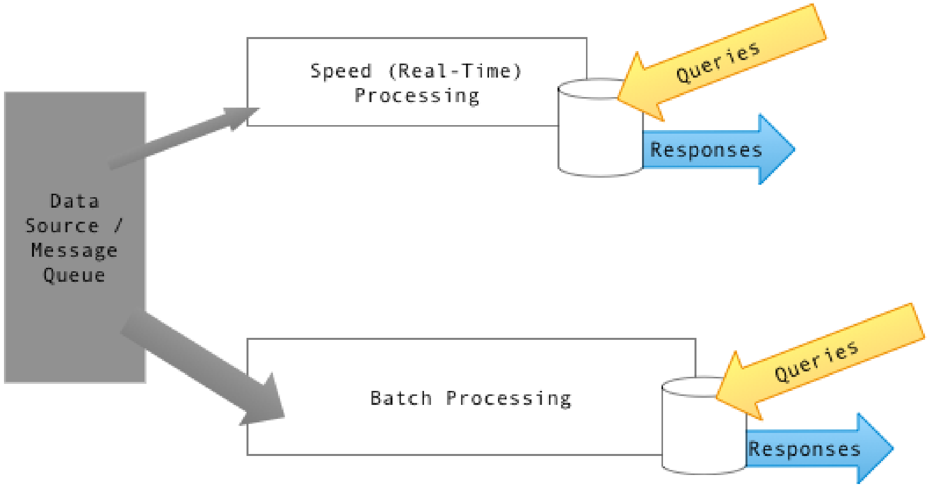
❸ Create graph using GraphX

❹ Use GraphX operations to service queries

❺ ❻ Use Google Data Studio to visualize analysis results

Lambda architecture

- Lambda architecture is new big data processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods.



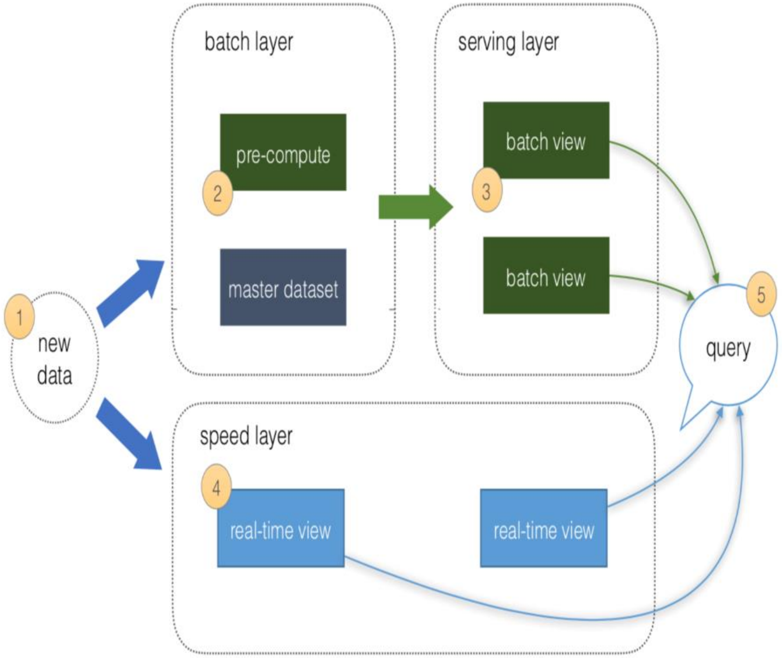
- This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data.

- The two view outputs may be joined before presentation. The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latencies of MapReduce.

Lambda architecture

Lambda architecture describes a system consisting of three layers: batch processing, speed (or real-time) processing, and a serving layer for responding to queries

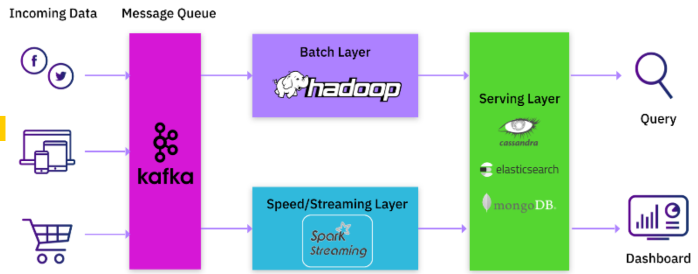
- Batch layer
- Speed layer
- Serving layer



1. All **data** entering the system is dispatched to both the batch layer and the speed layer for processing.
2. The **batch layer** has two functions: (i) managing the master dataset (an immutable, append-only set of raw data), and (ii) to pre-compute the batch views.
3. The **serving layer** indexes the batch views so that they can be queried in low-latency, ad-hoc way.
4. The **speed layer** compensates for the high latency of updates to the serving layer and deals with recent data only.
5. Any incoming **query** can be answered by merging results from batch views and real-time views.

Lambda architecture

Data flow

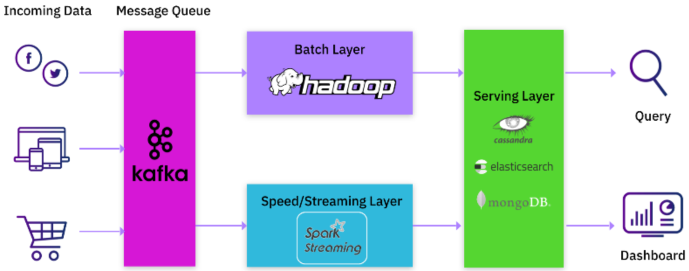


- Kafka is used for building a data pipeline that sits which accepts data from various inputs, and deliver the data to streaming processors or applications.

- The batch layer aims at perfect accuracy by being able to process *all* available data when generating views
  – Batch processing systems used include Hadoop,

- Stream-processing technologies typically used in this layer include Apache Storm, SQLstream, Apache Spark, and Azure Cosmos DB. Output is typically stored on fast NoSQL databases

Lambda architecture

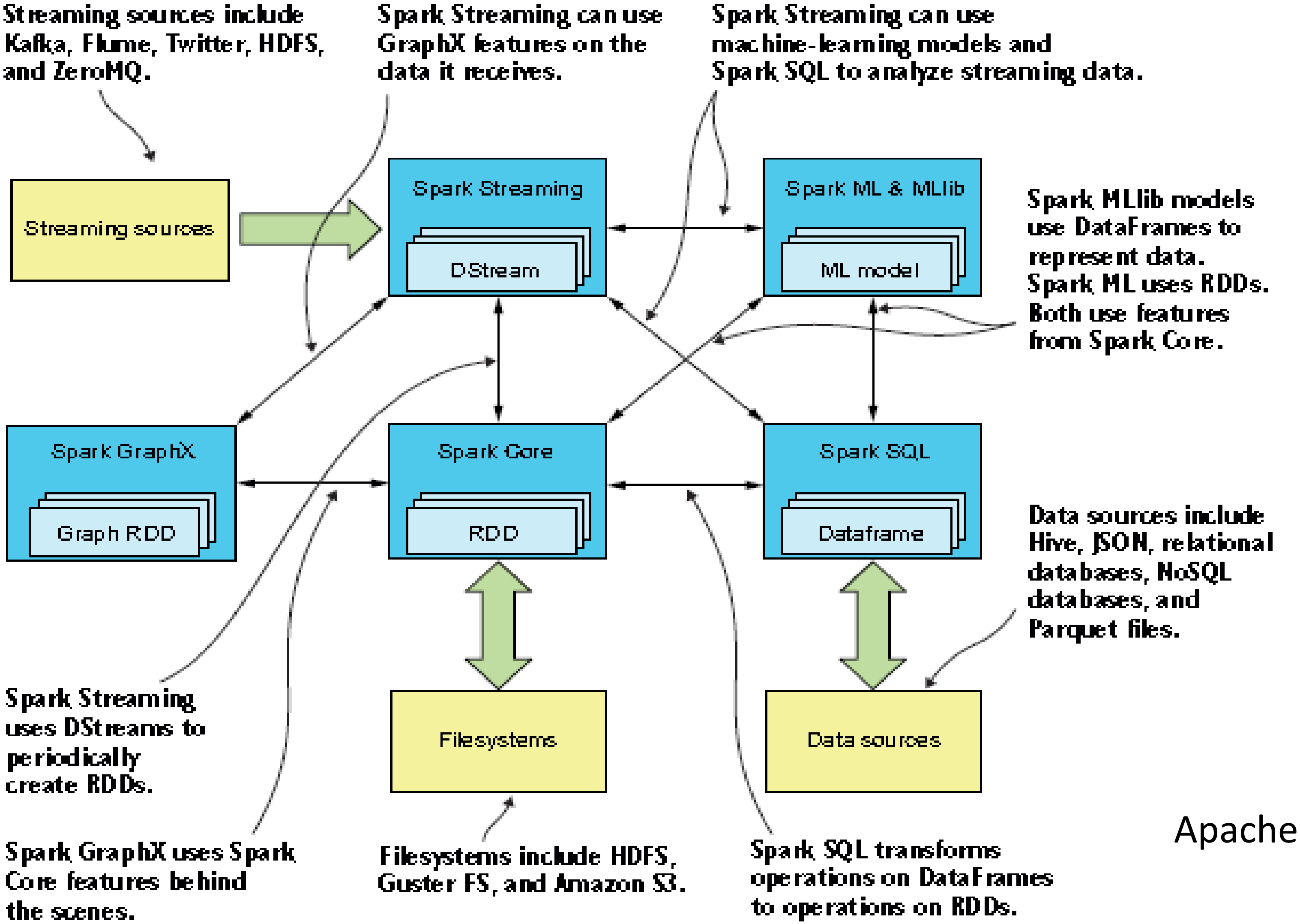


Data flow

- Output from the batch and speed layers are stored in the serving layer, which responds to ad-hoc queries by returning pre-computed views or building views from the processed data.

- Examples of technologies used in the serving layer include Druid, which provides a single cluster to handle output from both layers. Dedicated stores used in the serving layer include Apache Cassandra, Apache HBase, Azure Cosmos DB, MongoDB, or Elasticsearch for speed-layer output, and Elephant DB, Apache Impala, or Apache Hive for batch-layer output

Summary



Streaming sources include Kafka, Flume, Twitter, HDFS, and ZeroMQ.

Spark Streaming can use GraphX features on the data it receives.

Spark Streaming can use machine-learning models and Spark SQL to analyze streaming data.

Spark MLlib models use DataFrames to represent data. Spark ML uses RDDs. Both use features from Spark Core.

Data sources include Hive, JSON, relational databases, NoSQL databases, and Parquet files.

Spark Streaming uses DStreams to periodically create RDDs.

Spark GraphX uses Spark Core features behind the scenes.

Filesystems include HDFS, Guster FS, and Amazon S3.

Spark SQL transforms operations on DataFrames to operations on RDDs.

Apache Spark