APACHE SPARK STRUCTURED STREAMING

What is Apache Spark Streaming?

- **Apache Spark** Streaming is a component of Apache Spark for real-time data processing.
- It processes large data streams from sources like **Apache Kafka**, Flume, or TCP sockets, and outputs data to file systems, databases, or dashboards.
- Spark Streaming breaks down data streams into small batches for processing. This ***micro-batch architecture*** ensures high throughput and fault tolerance, making it suitable for processing continuous data flows.
- This streaming framework provides features for scalable and flexible processing of **unbounded datasets**.
- It supports tasks like window operations, transformations, and joins in a real-time context. Additionally, Spark's libraries for machine learning, graph computations, and SQL operations are fully accessible within Spark Streaming applications.

Key concepts in Spark Streaming

- There are several key concepts that users must understand to build real-time processing applications with Spark Streaming:
- **Discretized streams (DStreams):** DStreams are the primary abstraction in Spark Streaming, representing continuous streams of data. Internally, a DStream is a sequence of **resilient distributed datasets (RDDs)**, which are immutable collections of data partitioned across a cluster.
- DStreams can be created from various input sources, such as Apache Kafka, Flume, or Kinesis, or by applying operations on existing DStreams.
- **Transformations and actions:** Transformations are operations that create a **new DStream from an existing one**, such as map, filter, and reduceByKey. These operations are lazily evaluated, meaning they are not executed until an action is called. Actions, like print or saveAsTextFiles, trigger the execution of the transformations and produce output.
- **Windowed operations:** Windowed operations allow the **processing of data over sliding time windows**, enabling computations like rolling averages or sums. For example, a window operation can compute the word count over the last 10 seconds of data, updating every 5 seconds.

- **State management:** Stateful operations maintain information across batches, allowing for computations that depend on previous data. This is useful for tasks like **tracking session information** or maintaining running counts. Spark Streaming provides mechanisms to manage state efficiently.
- **Fault tolerance:** Spark Streaming ensures **fault tolerance through checkpointing and lineage information**.
- **Checkpointing** involves saving the state of the streaming application to reliable storage, enabling recovery in case of failures.
- **Lineage** information allows the system to **recompute lost data** by tracking the sequence of operations that produced the data.
- **Structured streaming:** Structured Streaming is an evolution of Spark Streaming that provides a higher-level API for stream processing.
- It treats streaming data as a **continuous table** and allows the use of SQL queries and DataFrame operations.
- Structured Streaming offers end-to-end exactly-once guarantees and integrates with batch processing.

How Spark Streaming works

Data sources and receivers

- Spark Streaming interacts with various data sources using receivers.
- Receivers are special objects that capture data from input sources and store it as RDDs.
- These can include data from Apache Kafka, Flume, or custom socket streams.
- **Receivers *run as tasks*** within **the Spark cluster**, utilizing Spark's distributed architecture to handle high-volume data rates.
- This design allows Spark Streaming to adapt to large-scale data ingestion needs promptly.

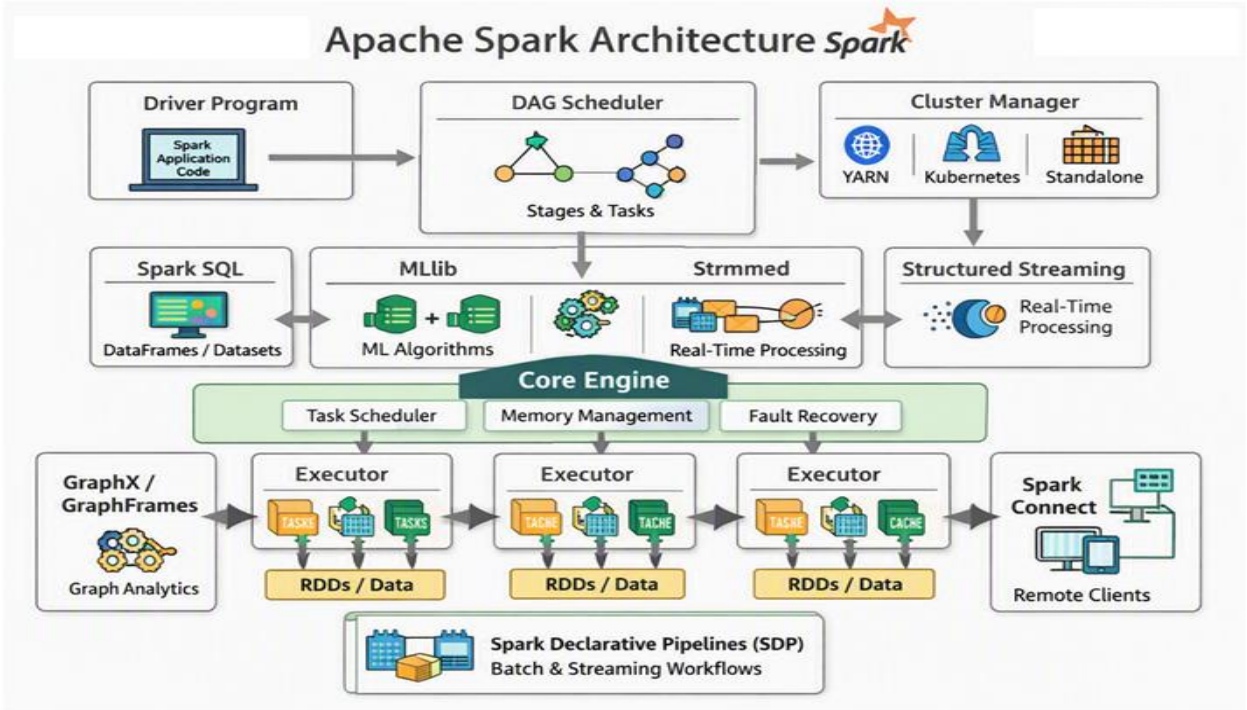
Transformations on DStreams

- Transformations in Spark Streaming allow processing and manipulation of DStreams, adapting batch-level Spark operations to real-time data streams.
- These operations include common transformations like map, filter, reduce, and windowing, enabling compounding of processing logic across the stream data.
- By applying these transformations, Spark Streaming applications can execute complex analytics, providing immediate insights from incoming data.

Output operations

- Output operations in Spark Streaming dictate how processed data is delivered at the end of transformations.
- These operations control where and how the results from streamed data computations are stored or sent for subsequent use.
- Some common output operations include saving results to databases, file systems, or pushing them to live dashboards.

<u>Architecture of Apache Spark</u>

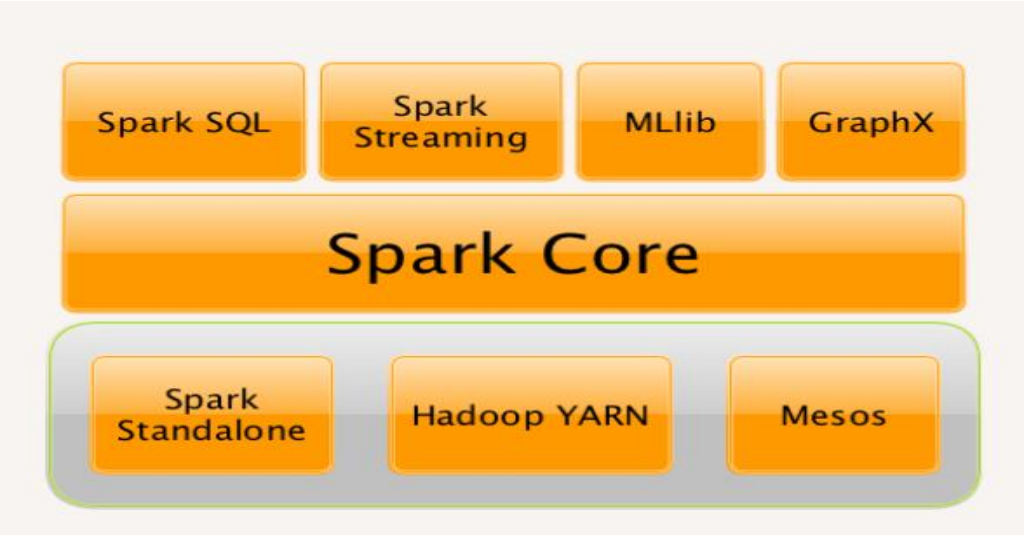


Apache Spark Architecture Overview

- Unified platform for **large-scale distributed data processing**
- Components:
  - Core Engine,
  - Driver,
  - Cluster Manager,
  - Executors,
  - Libraries

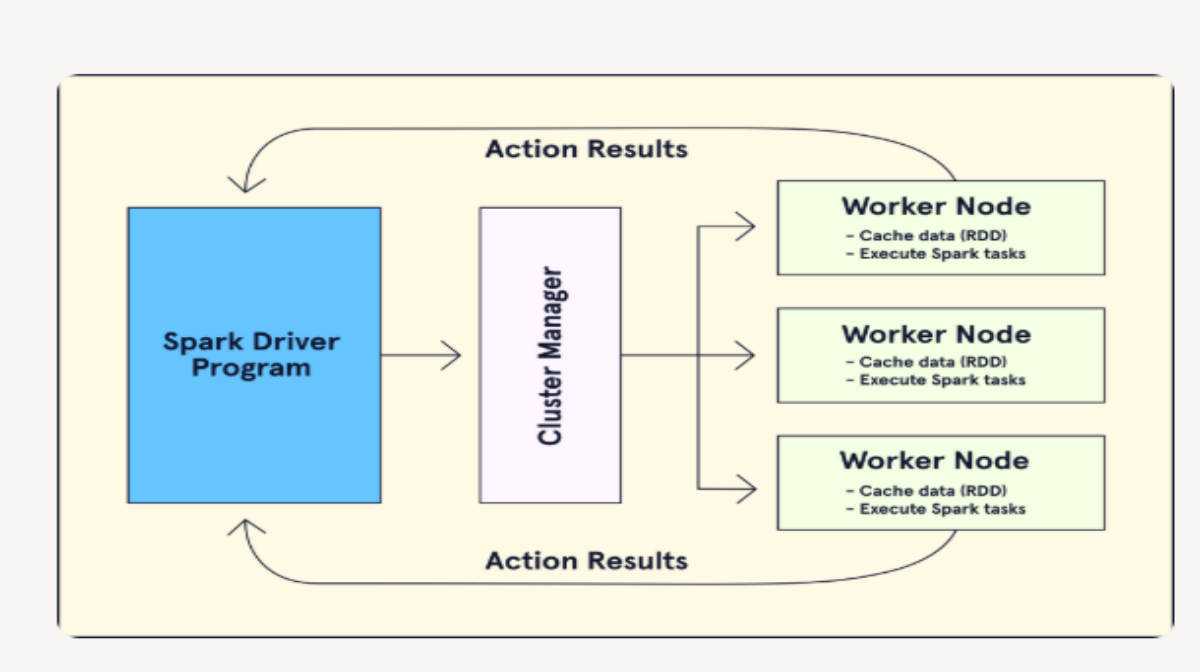
Spark Core Engine

- **Core Engine** is the foundation of Spark.
- Handles:
- Task scheduling and dispatch
- Memory management and caching
- Fault tolerance via lineage tracking
- Enables parallel execution across cluster nodes.



Driver Program

- **Driver Program** orchestrates the workflow.
- Responsibilities:
- Initializes SparkSession/SparkContext
- Builds logical execution plan
- Requests resources from Cluster Manager
- Monitors task execution and collects results



Cluster Manager

- **Cluster Manager** allocates resources.
- Types:
- Spark Standalone
- YARN (Hadoop)

- Kubernetes (Cloud)
- Launches executors on worker nodes for each application.

Executors

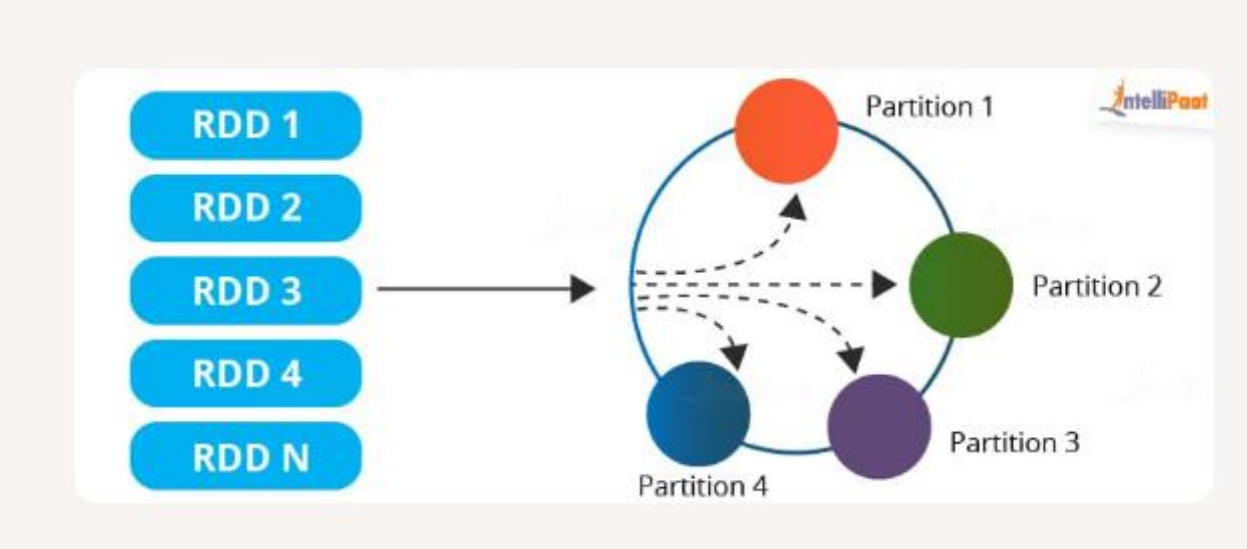
- **Executors** perform actual data processing.
- Each application gets its own executors.
- Tasks:
- Execute transformations on data partitions
- Cache intermediate results
- Report status to driver
- Handle data shuffles for joins/aggregations

DAG Scheduler

- **DAG Scheduler** organizes tasks into a Directed Acyclic Graph.
- Converts logical transformations into stages and tasks.
- Optimizes execution by minimizing data movement and pipelining stages.

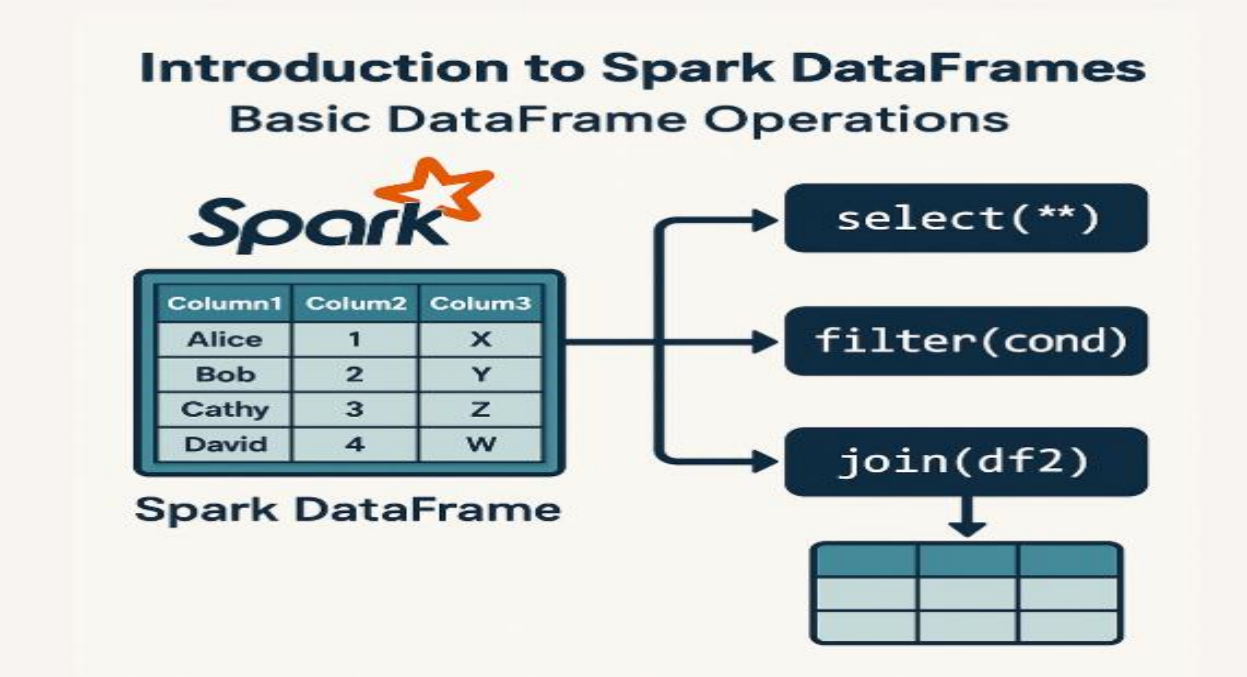
RDDs (Resilient Distributed Datasets)

- **RDDs** are immutable distributed collections.
- Support functional transformations: map, filter, reduce.
- Track lineage for fault recovery.
- Foundation for higher-level APIs.



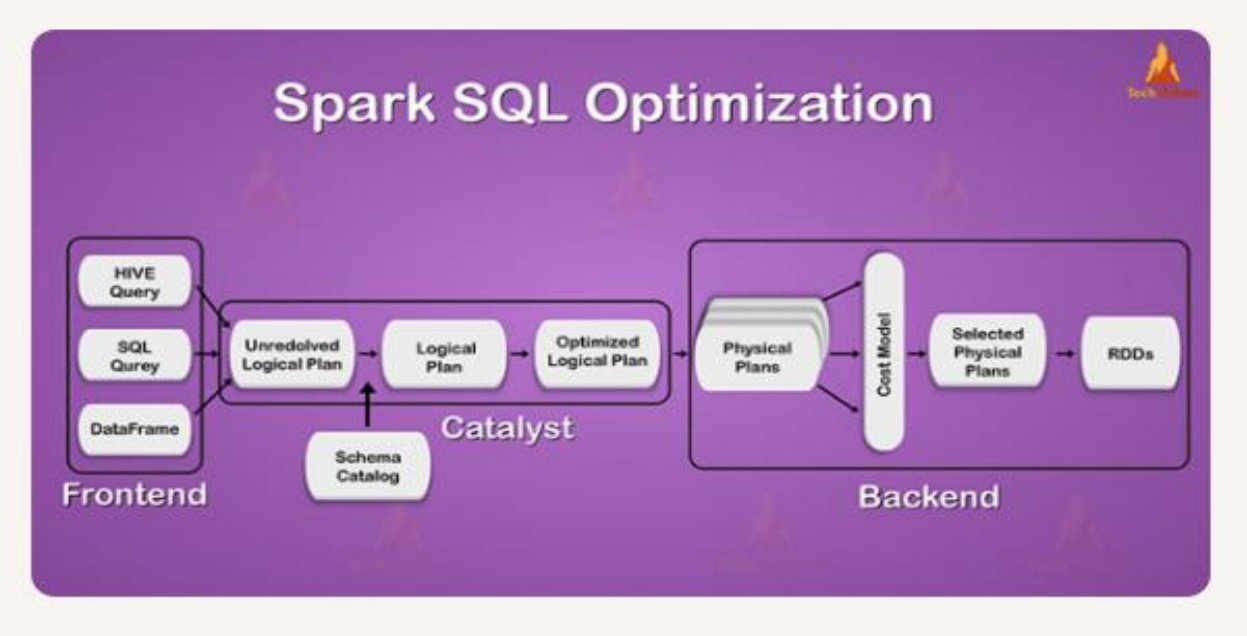
DataFrames and Datasets

- **DataFrames and Datasets** provide schema-aware abstractions.
- Enable relational-style queries.
- Optimized by Catalyst query optimizer.
- Easier to write and maintain than raw RDDs.



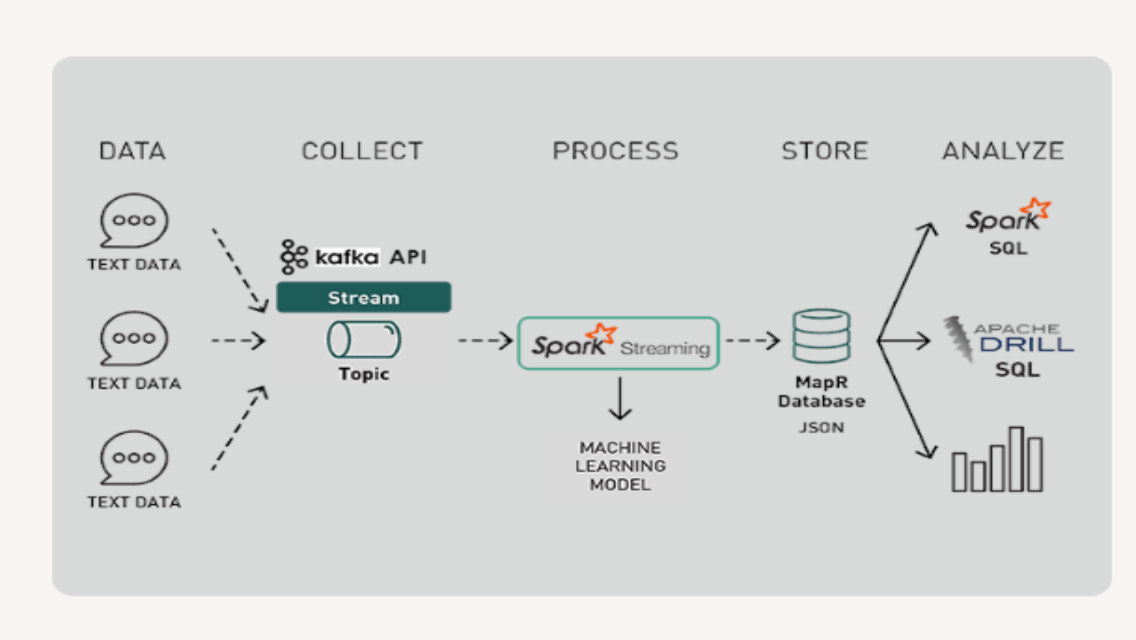
Spark SQL

- **Spark SQL** enables SQL queries on structured data.
- Integrates with BI tools via JDBC/ODBC.
- Uses Catalyst optimizer for efficient query planning.



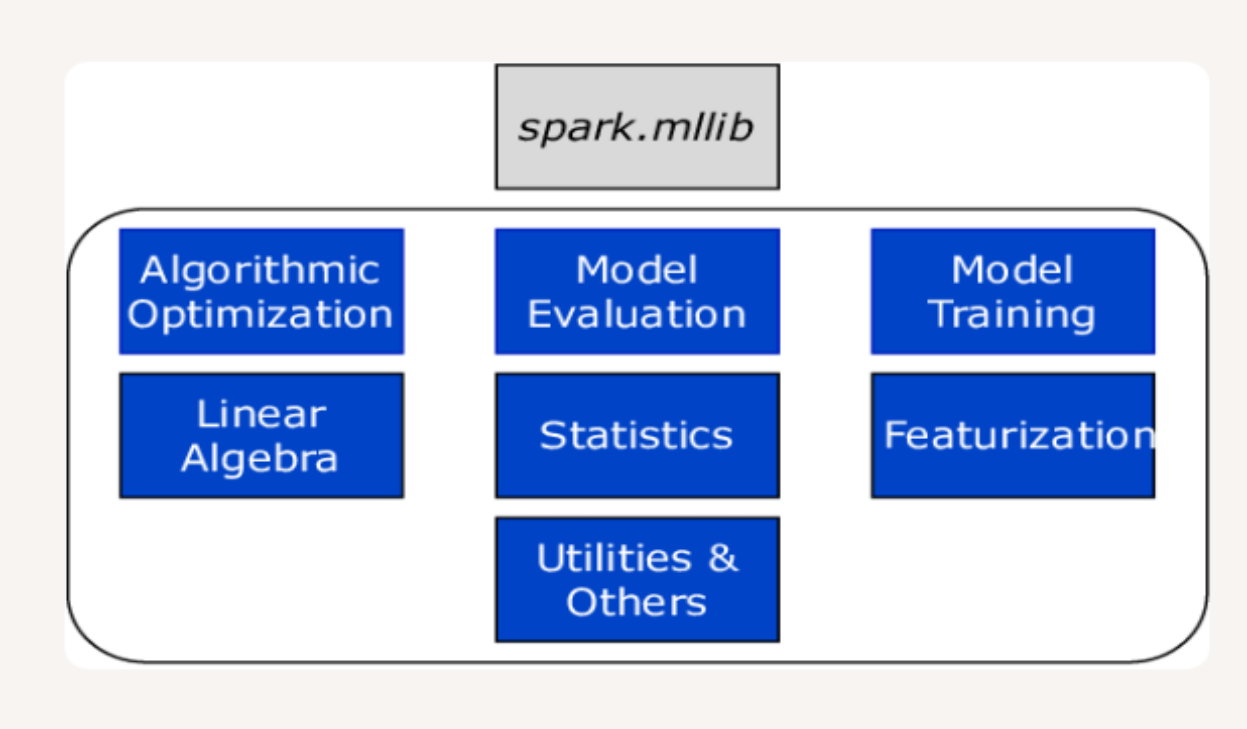
Structured Streaming

- **Structured Streaming** treats live data as a continuously appended table.
- Supports event-time processing, watermarks, and stateful operations.
- Real-Time Mode for sub-second latency.



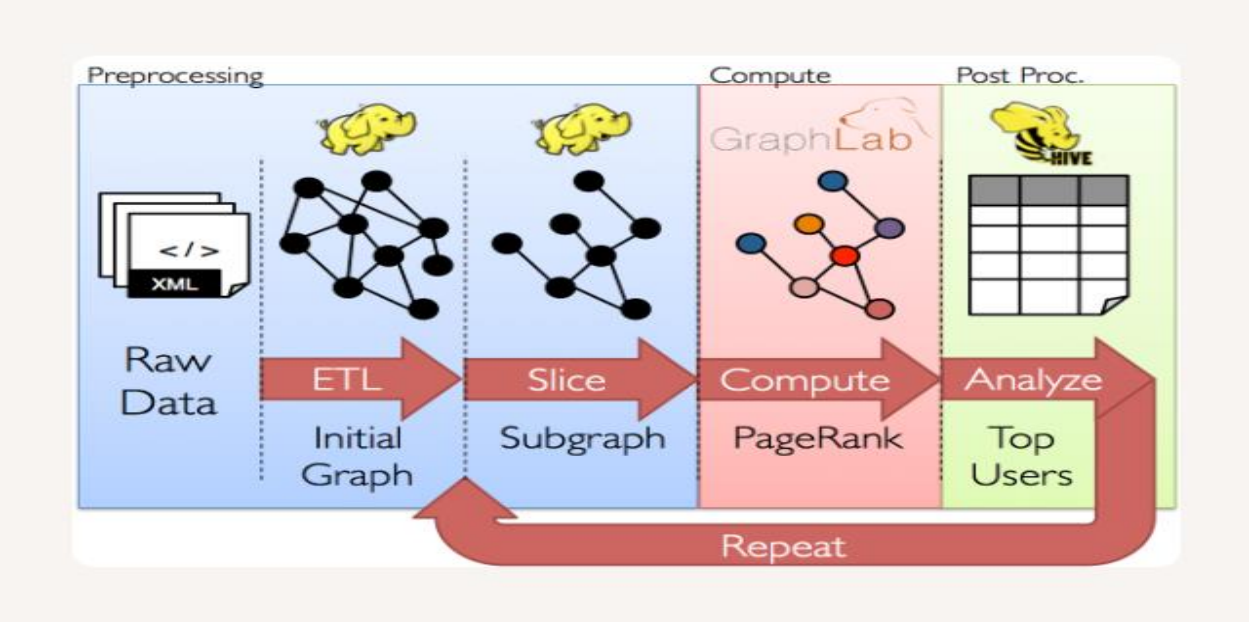
MLlib

- **MLlib** provides distributed machine learning algorithms.
- Supports classification, regression, clustering, and collaborative filtering.



GraphX / GraphFrames

- **GraphX and GraphFrames** enable graph analytics.
- GraphX: Scala/Java API for graph algorithms.
- GraphFrames: DataFrame-based graph queries.



Spark Connect

- **Spark Connect** decouples client applications from the execution engine.
- Enables remote execution and debugging.
- Improves stability and upgrade flexibility.

Spark Declarative Pipelines (SDP)

- **Spark Declarative Pipelines** simplify pipeline definition.
- Automatically handles dependencies, checkpoints, and retries.
- Works for both batch and streaming workflows.