Big Data
(Spark)

Instructor: Trong-Hop Do

November 24th 2020

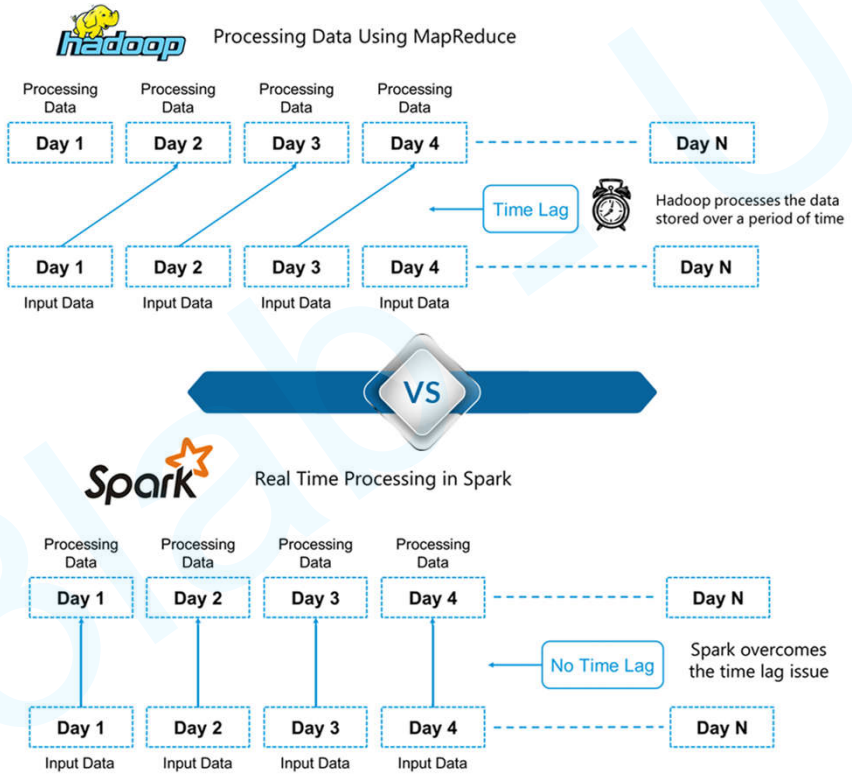
S3Lab
Smart Software System Laboratory


“Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming.”
– Chris Lynch, Vertica Systems

Introduction

- Apache Spark is an open-source cluster computing framework for real-time processing
- Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.
- Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming

Introduction

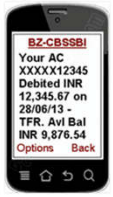


Introduction

Hadoop implements Batch processing on Big Data.
It thus cannot deliver to our Real-Time use case needs.

Our Requirements:	hadoop	Spark
Process data in real-time	✗	✓
Handle input from multiple sources	✓	✓
Easy to use	✗	✓
Faster processing	✗	✓

Applications



Banking

Government

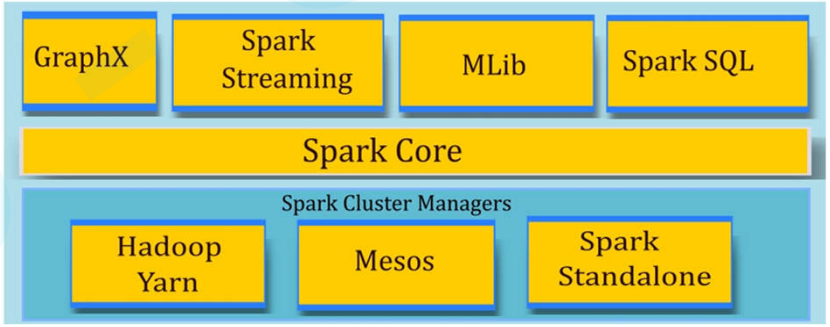
Healthcare

Telecommunications

Stock Market

Components

- Spark Core and Resilient Distributed Datasets or RDDs
- Spark SQL
- Spark Streaming
- Machine Learning Library or MLlib
- GraphX



Components


Spark
SQL
Spark
streaming
Spark
MLlib
Spark
GraphX
Spark
R
Spark
Scala
R

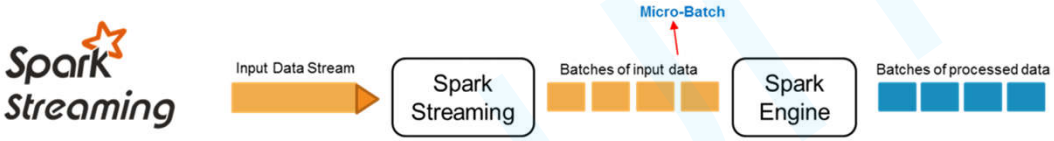
Components

Spark Core

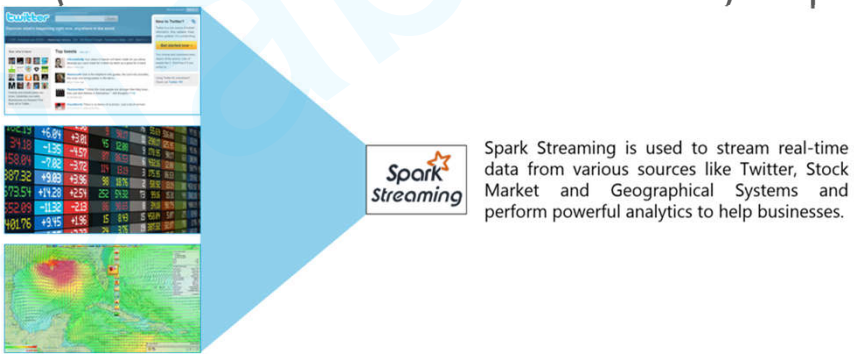
- The base engine for large-scale parallel and distributed data processing. The core is the distributed execution engine and the Java, Scala, and Python APIs offer a platform for distributed ETL application development. Further, additional libraries which are built atop the core allow diverse workloads for streaming, SQL, and machine learning. It is responsible for:
  - Memory management and fault recovery
  - Scheduling, distributing and monitoring jobs on a cluster
  - Interacting with storage systems

Components

Spark Streaming

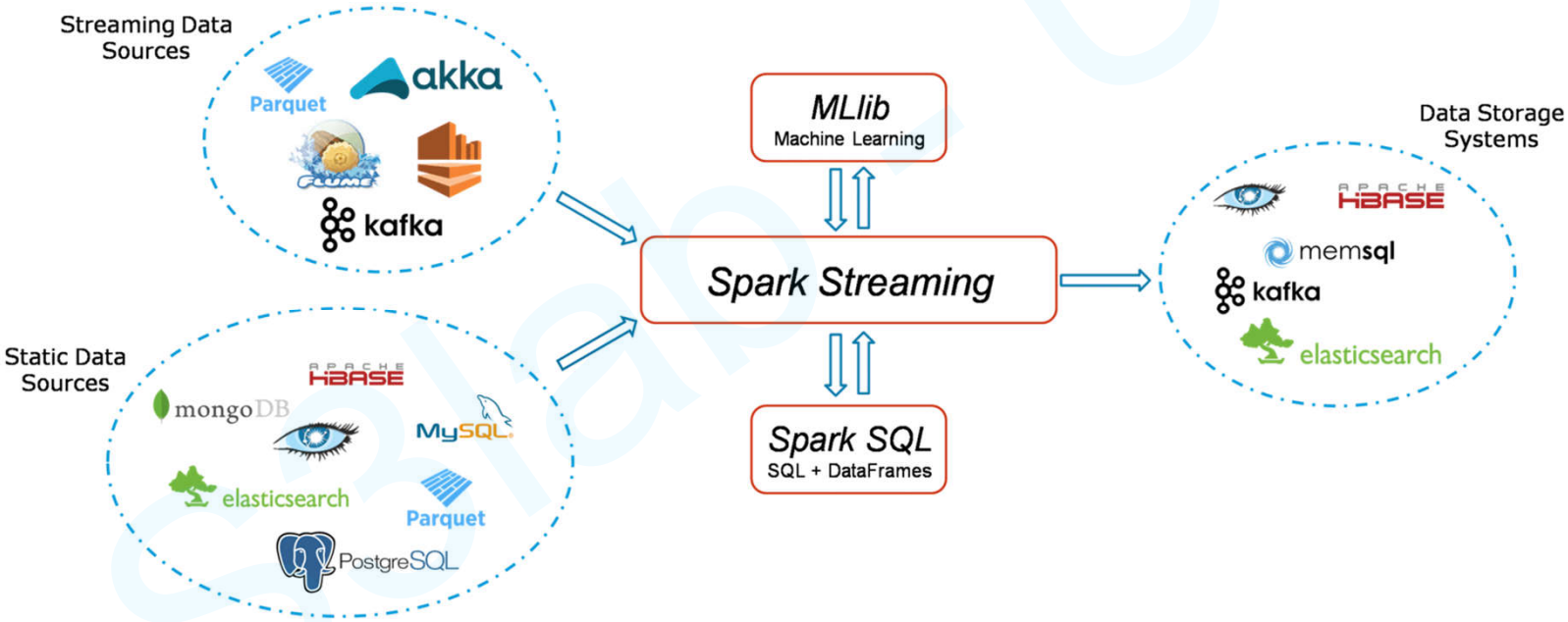


- Used to process real-time streaming data.
- It enables high-throughput and fault-tolerant stream processing of live data streams. The fundamental stream unit is **DStream** which is basically a series of **RDDs** (Resilient Distributed Datasets) to process the real-time data.



Components

Spark Streaming - Workflow



Components

Spark Streaming - Fundamentals

- Streaming Context
- DStream (Discretized Stream)
- Caching
- Accumulators, Broadcast Variables and Checkpoints

Components

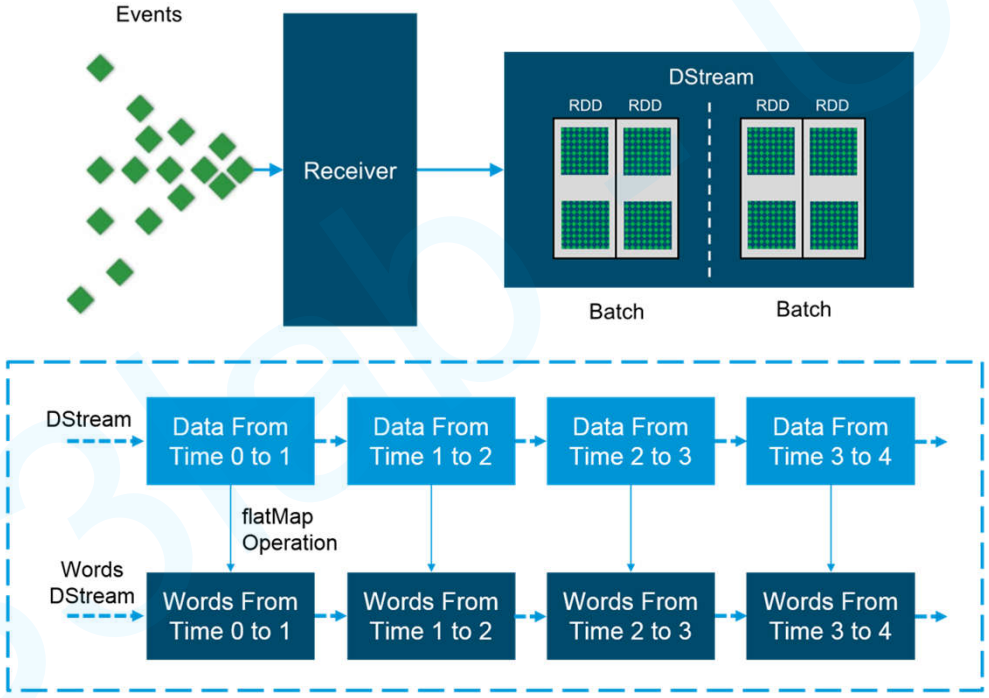
Spark Streaming - Fundamentals

Figure: *Spark Streaming Context*

Figure: *Default Implementation Sources*

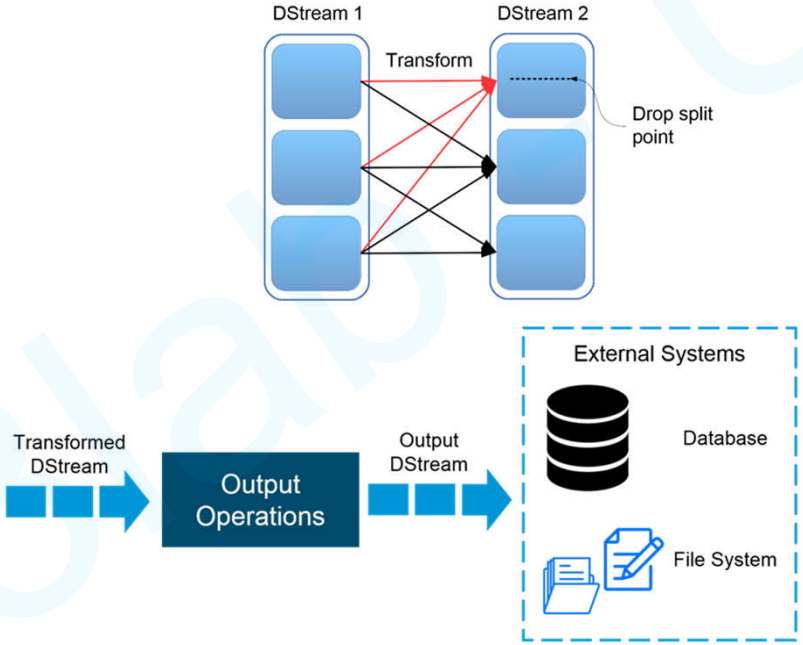
Components

Spark Streaming - Fundamentals



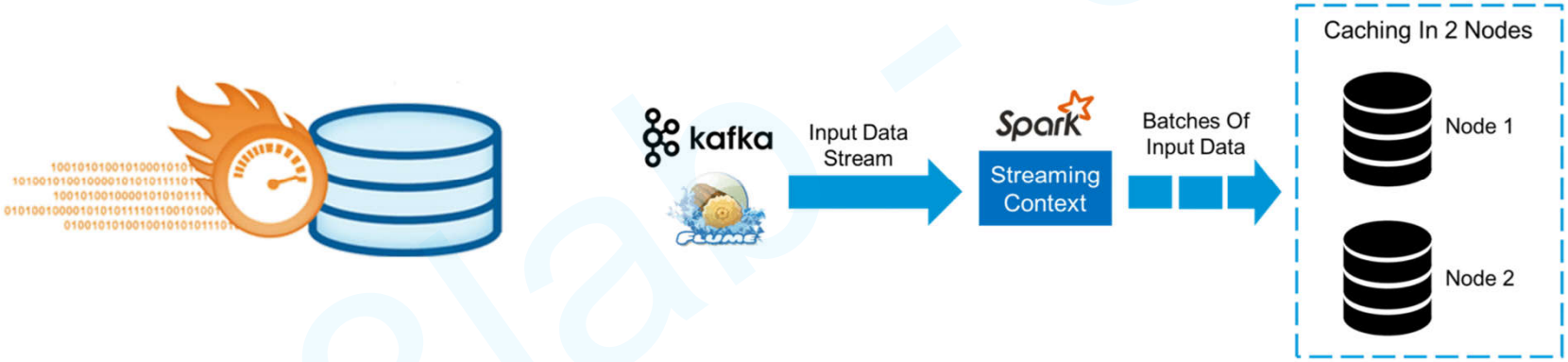
Components

Spark Streaming - Fundamentals



Components

Spark Streaming - Fundamentals



Components

Spark SQL

- Spark SQL integrates relational processing with Spark functional programming. Provides support for various data sources and makes it possible to weave SQL queries with code transformations thus resulting in a very powerful tool. The following are the four libraries of Spark SQL.
  - Data Source API
  - DataFrame API
  - Interpreter & Optimizer
  - SQL Service

Components

Spark SQL

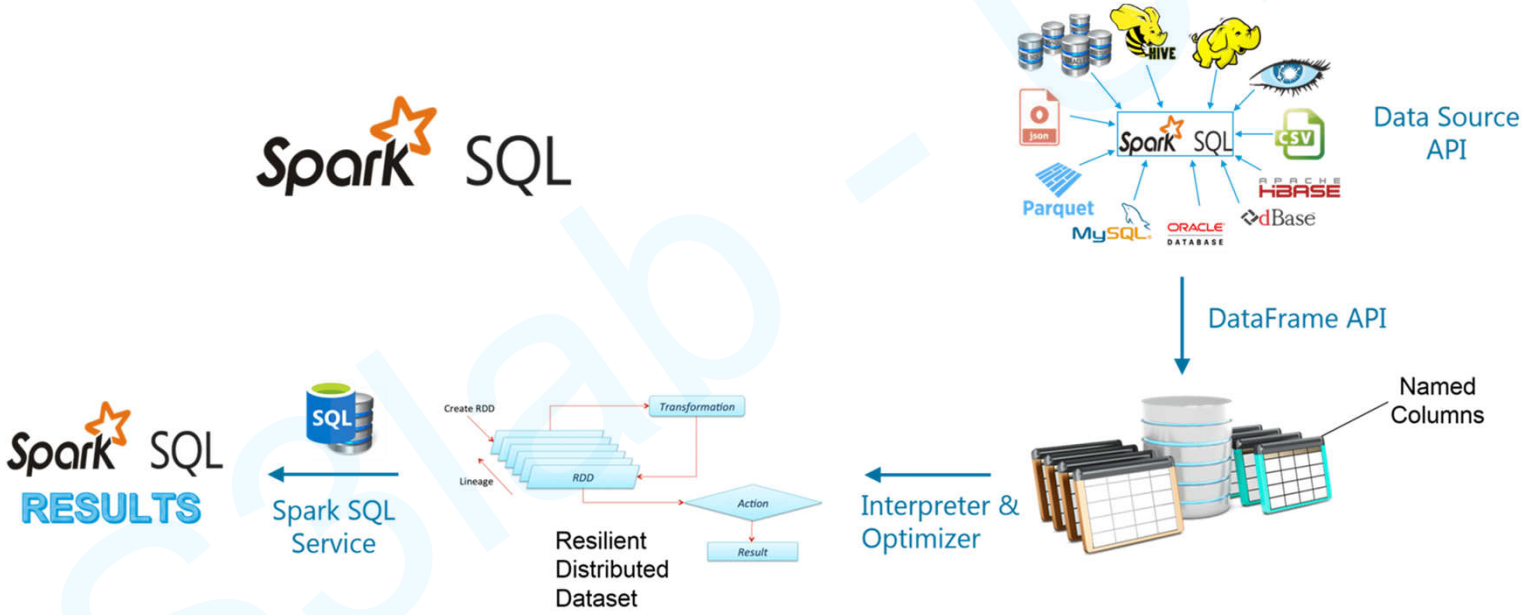


Figure: *The flow diagram represents a Spark SQL process using all the four libraries in sequence*

Components

Spark SQL - Data Source API

- Universal API for loading and storing structured data.
  - Built in support for Hive, JSON, Avro, JDBC, Parquet, ect.
  - Support third party integration through spark packages
  - Support for smart sources
  - Data Abstraction and Domain Specific Language (DSL) applicable on structure and semi-structured data
  - Supports different data formats (Avro, CSV, Elastic Search and Cassandra) and storage systems (HDFS, HIVE Tables, MySQL, etc.)
  - Can be easily integrated with all Big Data tools and frameworks via Spark-Core.
  - It processes the data in the size of Kilobytes to Petabytes on a single-node cluster to multi-node clusters.

Components

Spark SQL - DataFrame API

- A Data Frame is a distributed collection of data organized into named column. It is equivalent to a relational table in SQL used for storing data into tables.

Components

Spark SQL - SQL Interpreter And Optimizer

- based on functional programming constructed in Scala.
  - provides a general framework for transforming trees, which is used to perform analysis/evaluation, optimization, planning, and run time code spawning.
  - This supports cost based optimization (run time and resource utilization is termed as cost) and rule based optimization, making queries run much faster than their RDD (Resilient Distributed Dataset) counterparts.

Components

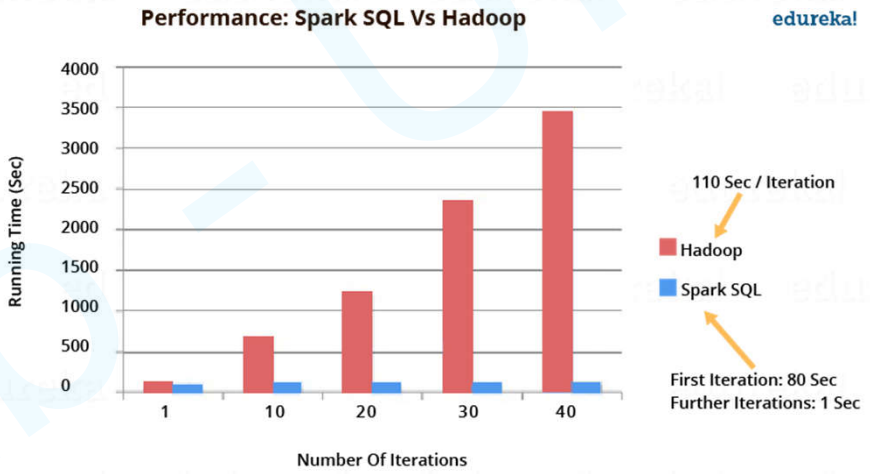
Spark SQL - SQL Service

- SQL Service is 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.

Components

Spark SQL - Features

- Integration With Spark
- Uniform Data Access
- Hive Compatibility
- Standard Connectivity
- Performance And Scalability
- User Defined Functions



Components

GraphX

- GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. The property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex have user defined properties associated with it. Here, the parallel edges allow multiple relationships between the same vertices.

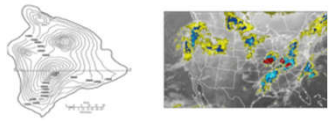
Components

GraphX

- GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API.
- In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.
- GraphX unifies ETL (Extract, Transform & Load) process, exploratory analysis and iterative graph computation within a single system.

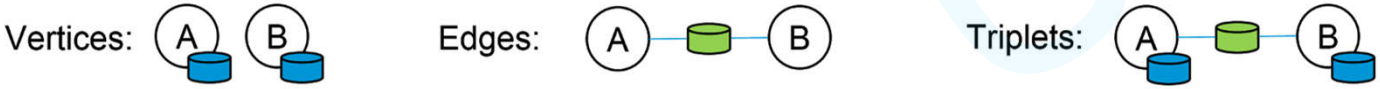
Components

GraphX - use cases

- Disaster Detection System 
- Page Rank 
- Financial Fraud Detection 
- Business Analysis 
  - Machine Learning, understanding the customer purchase trends
- Geographic Information Systems 
  - Watershed delineation and weather prediction
- Google Pregel

Components

GraphX - Graph and Examples



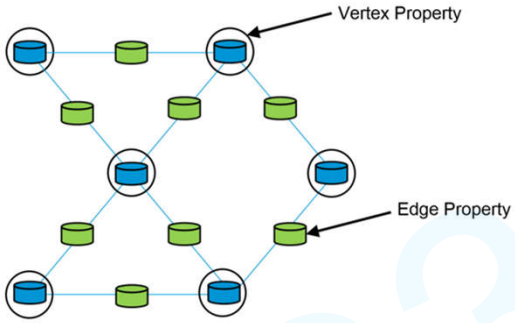


Figure: *Property Graph*

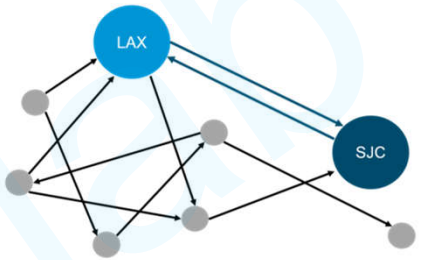
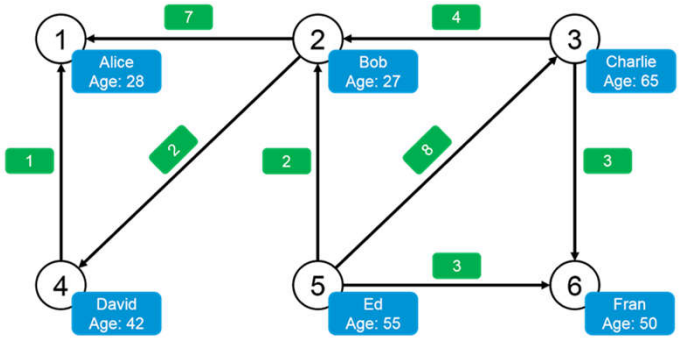


Figure: *An example of property graph*



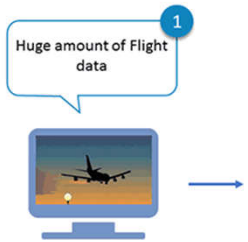
Components

GraphX - Flight Data Analysis using Spark GraphX

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	dOfM	dOfW	carrier	tailNum	flNum	origin_id	origin	dest_id	dest	crsdeptime	deptime	depdelaymins	crsarrtime	arrtime	arrdelaymins	crselapsedtime	dist
2	1	3	AA	N338AA	1	12478	JFK	12892	LAX	900	914	14	1225	1238	13	385	2475
3	2	4	AA	N338AA	1	12478	JFK	12892	LAX	900	857	0	1225	1226	1	385	2475
4	4	6	AA	N327AA	1	12478	JFK	12892	LAX	900	1005	65	1225	1324	59	385	2475
5	5	7	AA	N323AA	1	12478	JFK	12892	LAX	900	1050	110	1225	1415	110	385	2475
6	6	1	AA	N319AA	1	12478	JFK	12892	LAX	900	917	17	1225	1217	0	385	2475
7	7	2	AA	N328AA	1	12478	JFK	12892	LAX	900	910	10	1225	1212	0	385	2475
8	8	3	AA	N323AA	1	12478	JFK	12892	LAX	900	923	23	1225	1215	0	385	2475
9	9	4	AA	N339AA	1	12478	JFK	12892	LAX	900	859	0	1225	1204	0	385	2475
10	10	5	AA	N319AA	1	12478	JFK	12892	LAX	900	929	29	1225	1245	20	385	2475

Components

GraphX - Flight Data Analysis using Spark GraphX



Components

GraphX - Flight Data Analysis using Spark GraphX

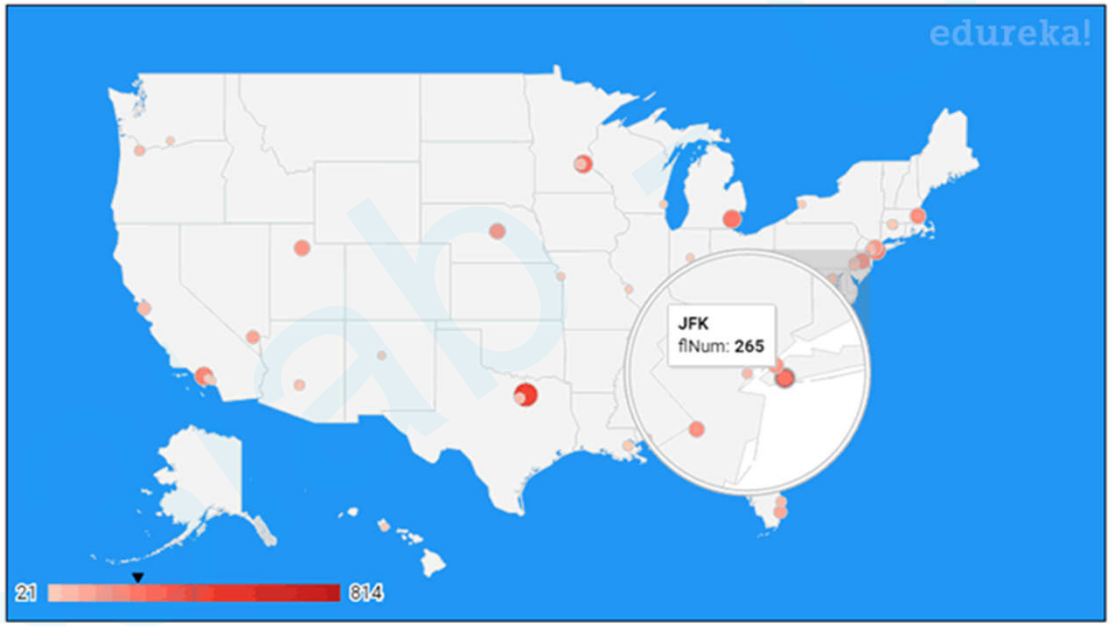


Figure: *Total Number of Flights from New York*

Components

MLLib

- stands for Machine Learning Library. Spark MLlib is used to perform machine learning in Apache Spark.

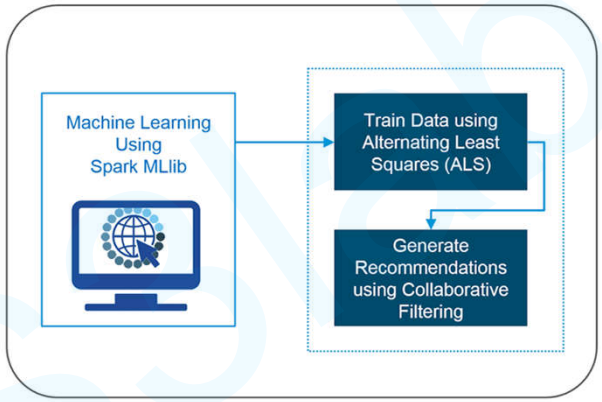


Figure: *Machine Learning Flow Diagram*

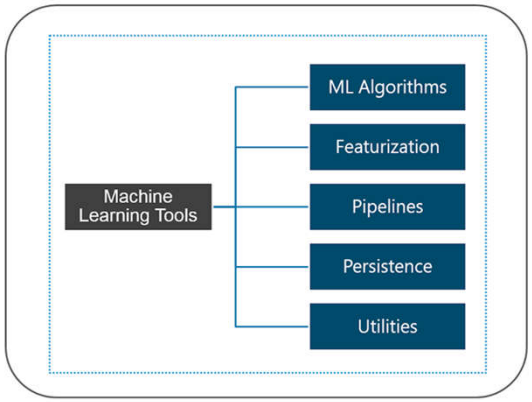
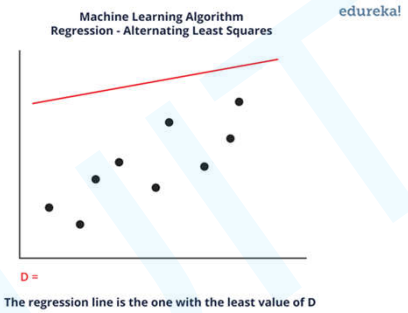


Figure: *Machine Learning Tools*

Components

MLLib - Algorithms

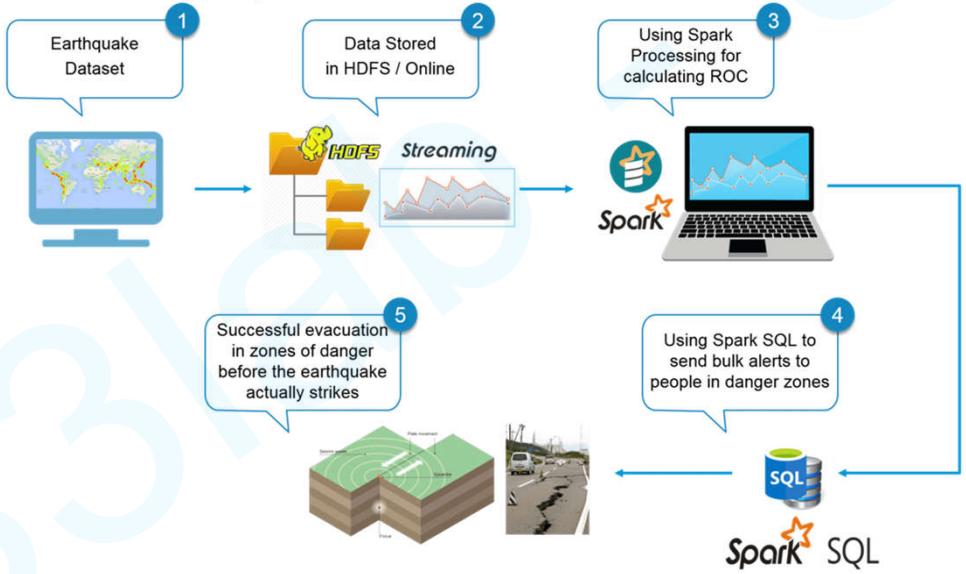


- **Basic Statistics**: Summary, Correlation, Stratified Sampling, Hypothesis Testing, Random Data Generation.
- Regression
- Classification
- Recommendation System: Collaborative, Content-Based
- Clustering
- Dimensionality Reduction: Feature Selection, Feature Extraction
- Feature Extraction
- Optimization

Components

MLLib - Use case

- Earthquake Detection System



Components

MLLib - Use case

- Movie Recommendation System

Components

MLLib - Use case

- Movie Recommendation System

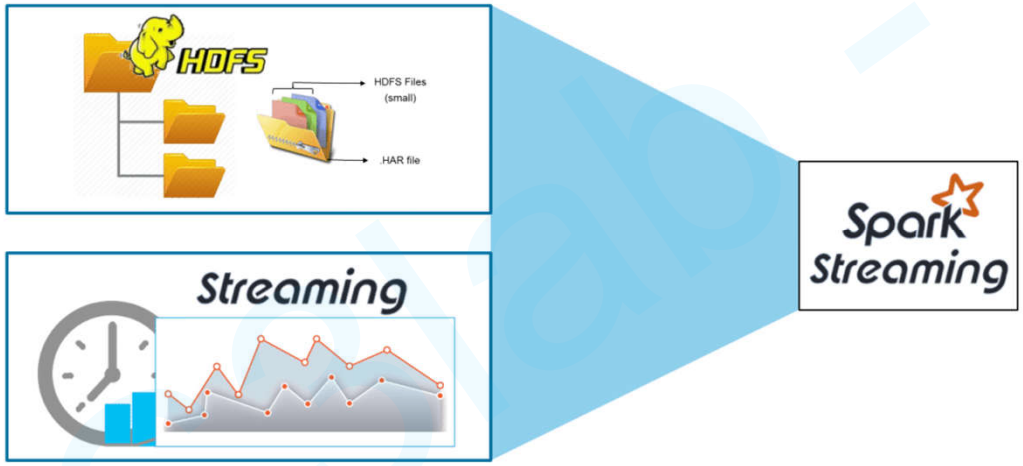


Figure: *Various File Formats*

Challenges of Distributed computing

- How to divide the input data
- How to assign the divided data to machines in the cluster
- How to check and monitor a machine in the cluster is live and has resources to perform its duty
- How to retry or reassign failed chunks to another machine or worker

Challenges of Distributed computing

- If the computation involves any aggregation operation like a sum, how to collate results from many workers and compute the aggregation
- Efficient use of memory , cpu and network
- Monitoring the tasks
- Overall job coordination
- Keeping a global time

Usecases

- ETL
- Analytics
- Machine Learning
- Graph processing
- SQL queries on large data sets
- Batch processing
- Stream processing

Features

- **Multiple languages support** namely Java, R, Scala, Python for building applications. Spark provides high-level APIs in **Java**, **Scala**, **Python** and **R**. Spark code can be written in any of these four languages. It provides a **shell** in **Scala** and **Python**. The Scala shell can be accessed through *./bin/spark-shell* and Python shell through *./bin/pyspark* from the installed directory.

Features

- **Fast Speed** in Data Processing, 10 times faster on Disk and 100 times swifter in Memory (compare to Hadoop). Spark is able to achieve this speed through controlled partitioning. It manages data using partitions that help parallelize distributed data processing with minimal network traffic.
- Spark with abstraction-RDD provides **Fault Tolerance** with ensured Zero Data loss

Features

- Increase in system efficiency due to **Lazy Evaluation** of transformation in RDD: Apache Spark delays its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed. For **transformations**, Spark adds them to a DAG (Directed Acyclic Graph) of computation and only when the driver requests some data, does this DAG actually gets executed.

Features

Lazy evaluation

As we have seen in the log analytics example, transformations on RDDs are lazily evaluated to optimize disk and memory usage in Spark.

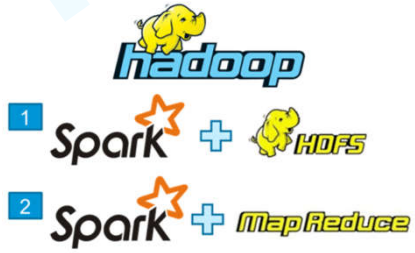
- RDDs are empty when they are created, only the type and ID are determined
- Spark will not begin to execute the job until it sees an action.
- Lazy evaluation is used to reduce the number of passes it has to take over the data by grouping operations together
- In MapReduce, developers have to spend a lot of time thinking about how to group together operations to minimize the number of MR passes
- A task and all its dependencies are largely omitted if the partition it generates is already cached.

Features

- **In-Memory Processing** resulting in high computation speed and acyclic data-flow
- With **80- high level operators** it is easy to develop **Dynamic & Parallel applications**
- **Real-time data stream processing** with Spark Streaming

Features

- Flexible to run **Independently** and can be **integrated with Hadoop** Yarn Cluster Manager. Apache Spark provides smooth compatibility with Hadoop. This is a boon for all the Big Data engineers who started their careers with Hadoop. Spark is a potential replacement for the MapReduce functions of Hadoop, while Spark has the ability to run on top of an existing Hadoop cluster using YARN for resource scheduling.



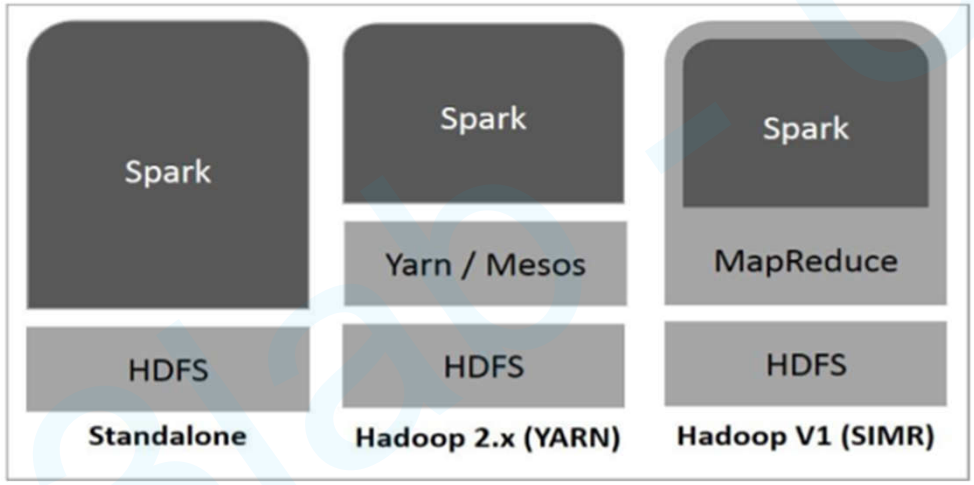
Features

- **Cost Efficient** for Big data as minimal need of storage and data center
- Futuristic analysis with **built-in tools** for machine learning (MLLib), interactive queries & data streaming
- **Persistence** and **Immutable** in nature with data paralleling processing over the cluster
- Graphx simplifies Graph Analytics by collecting algorithm and builders
- **Re-using Code** for batch processing and to run ad-hoc queries
- Progressive and expanding Apache community active for **Quick Assistance**

Features

- Spark supports **multiple data sources** such as Parquet, JSON, Hive and Cassandra apart from the usual formats such as text files, CSV and RDBMS tables. The Data Source API provides a pluggable mechanism for accessing structured data though Spark SQL. Data sources can be more than just simple pipes that convert data and pull it into Spark.

Spark build on Hadoop



RDD - Resilient Distributed Dataset

Definition

- **R**esilient **D**istributed **D**ataset is the fundamental data structure abstraction of Spark
- RDD is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. For instance you can create an RDD of integers and these gets partitioned and divided and assigned to various nodes in the cluster for parallel processing.
- RDDs can contain any type of **Python**, **Java**, or **Scala** objects, including **user-defined** classes.

RDD - Resilient Distributed Dataset

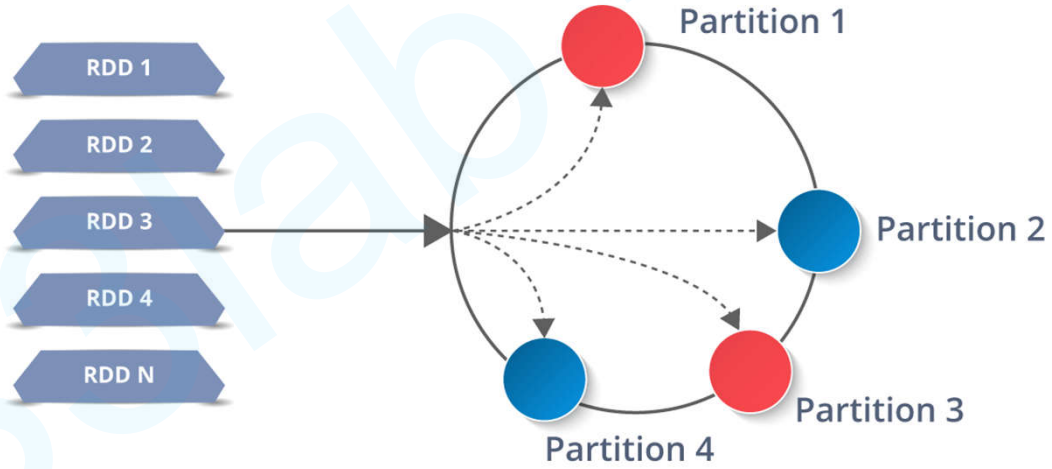
Definition

- **Resilient** - They are fault tolerant and able to detect and recompute missing or damaged partitions of an RDD due to node or network failures.
- **Distributed** - Data is partitioned and resides on multiple nodes depending on the cluster size, type and configuration
- **In Memory Data Structure** - Mostly they are in memory so that iterative operations runs faster and performs way better than traditional hadoop programs in executing iterative algorithms

RDD - Resilient Distributed Dataset

Definition

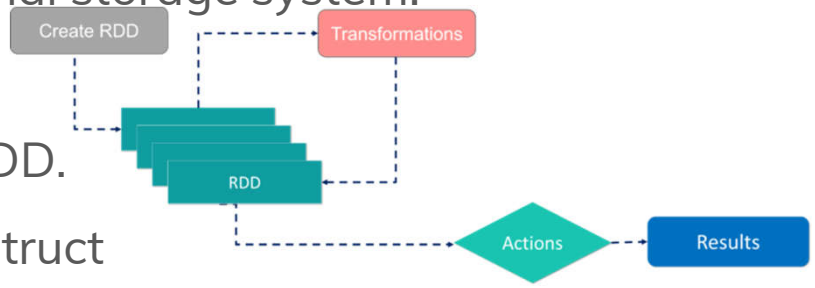
- **Dataset** - it can represent any form of data be it reading from a csv file, loading data from a table using rdbms, text file, json , xml.



RDD - Resilient Distributed Dataset

Workflow

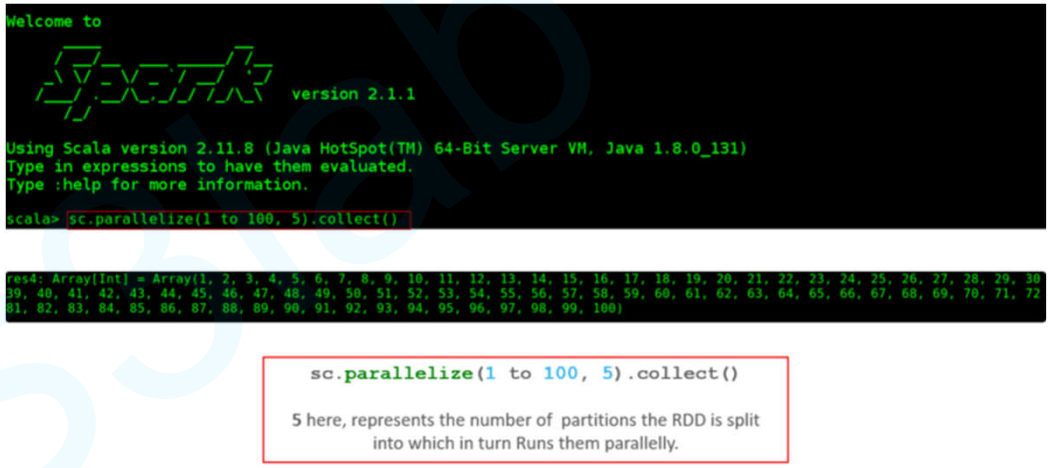
- Create RDDs
  - An existing collection in your **driver program**.
  - Referencing a dataset in an external storage system.
- 2 types of operations
  - **Transformation**: to create new RDD.
  - **Actions**: applied on an RDD to instruct Spark to apply computation and pass the result back to the driver.



RDD - Resilient Distributed Dataset

Partition and Parallelism

- **Partition** - is a logical chunk of a large distributed data set. By default. Spark tries to read data into an RDD from the nodes that are close to it.

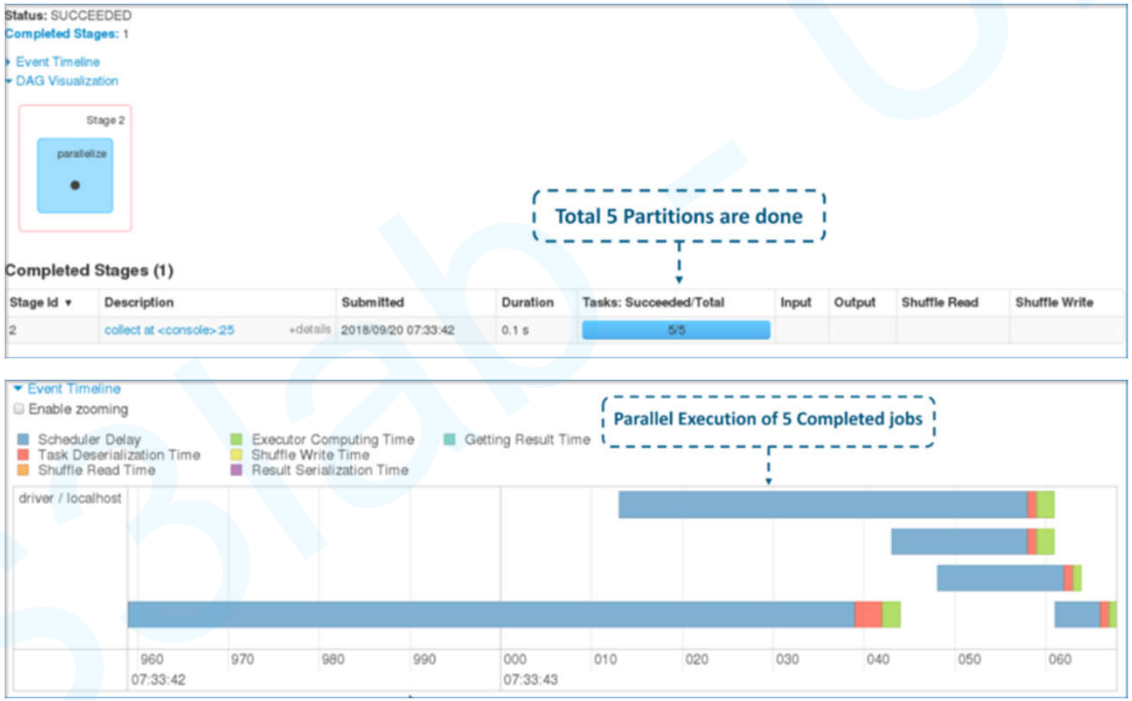



```
sc.parallelize(1 to 100, 5).collect()
```

5 here, represents the number of partitions the RDD is split into which in turn Runs them parallelly.

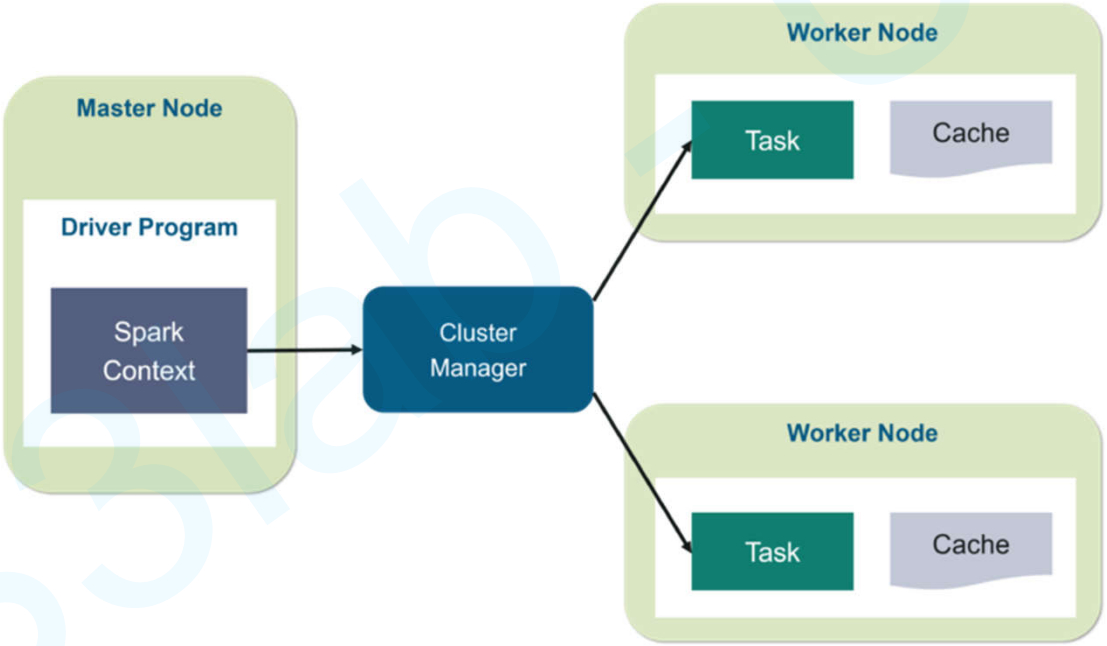
RDD - Resilient Distributed Dataset

Partition and Parallelism



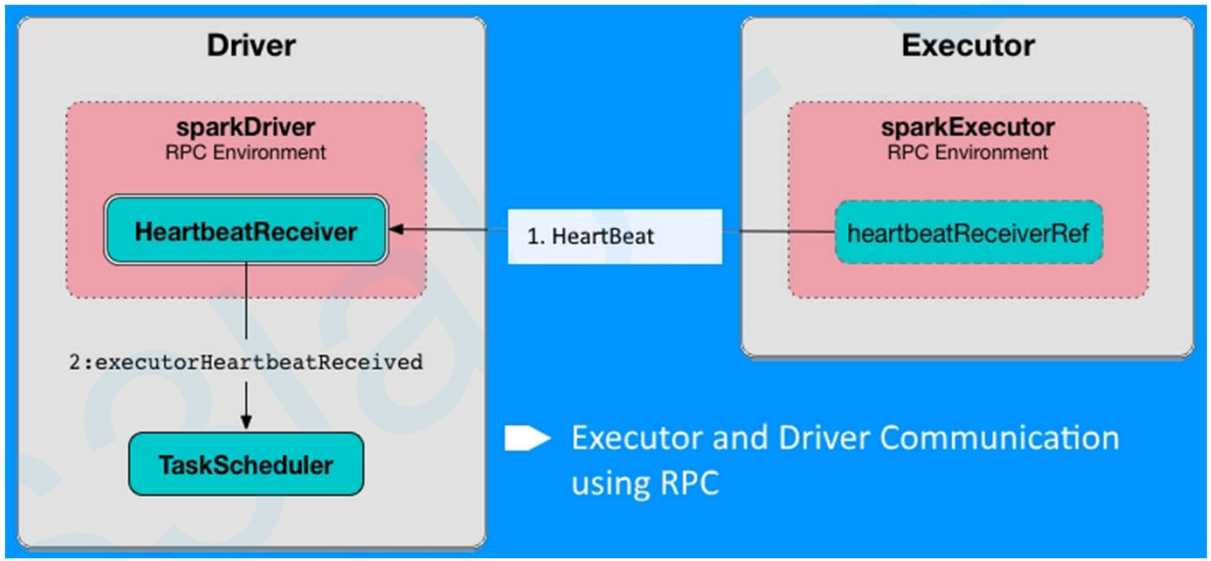
Spark Program

Architecture



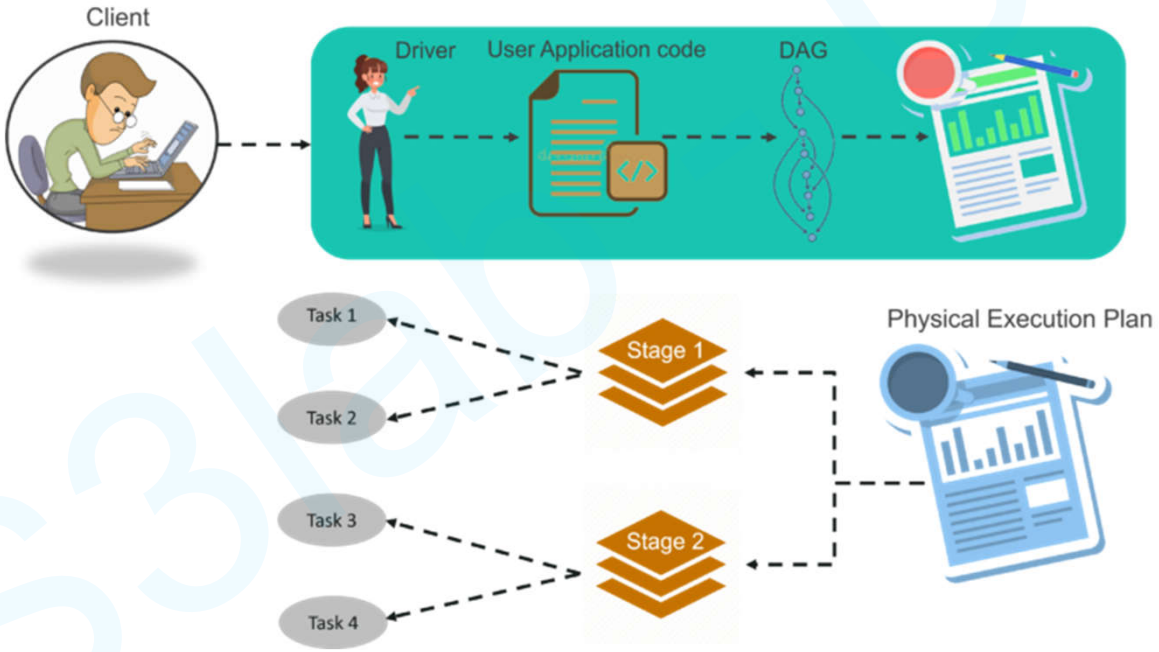
Spark Program

Architecture



Spark Program

Architecture

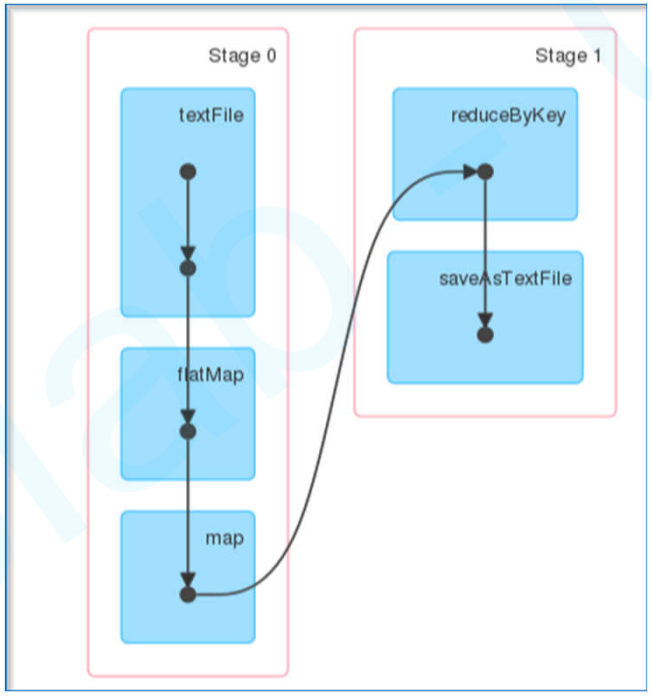


Spark Program

Architecture

Spark Program

Directed Acyclic Graph (DAG) Visualization



Q & A

Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.