SPARK

UNIT - III

What is Apache Spark?

Apache Spark is an open-source data processing engine to store and process data in real-time across various clusters of computers using simple programming constructs

Spark Definition

Spark is a Unified Computing Engine with a set of libraries

Unified : Spark is designed to support a wide range of data analytics tasks (ranging from data loading and SQL querying to Machine Learning and Real Time Streaming) on a single computing engine with consistent set of APIs.

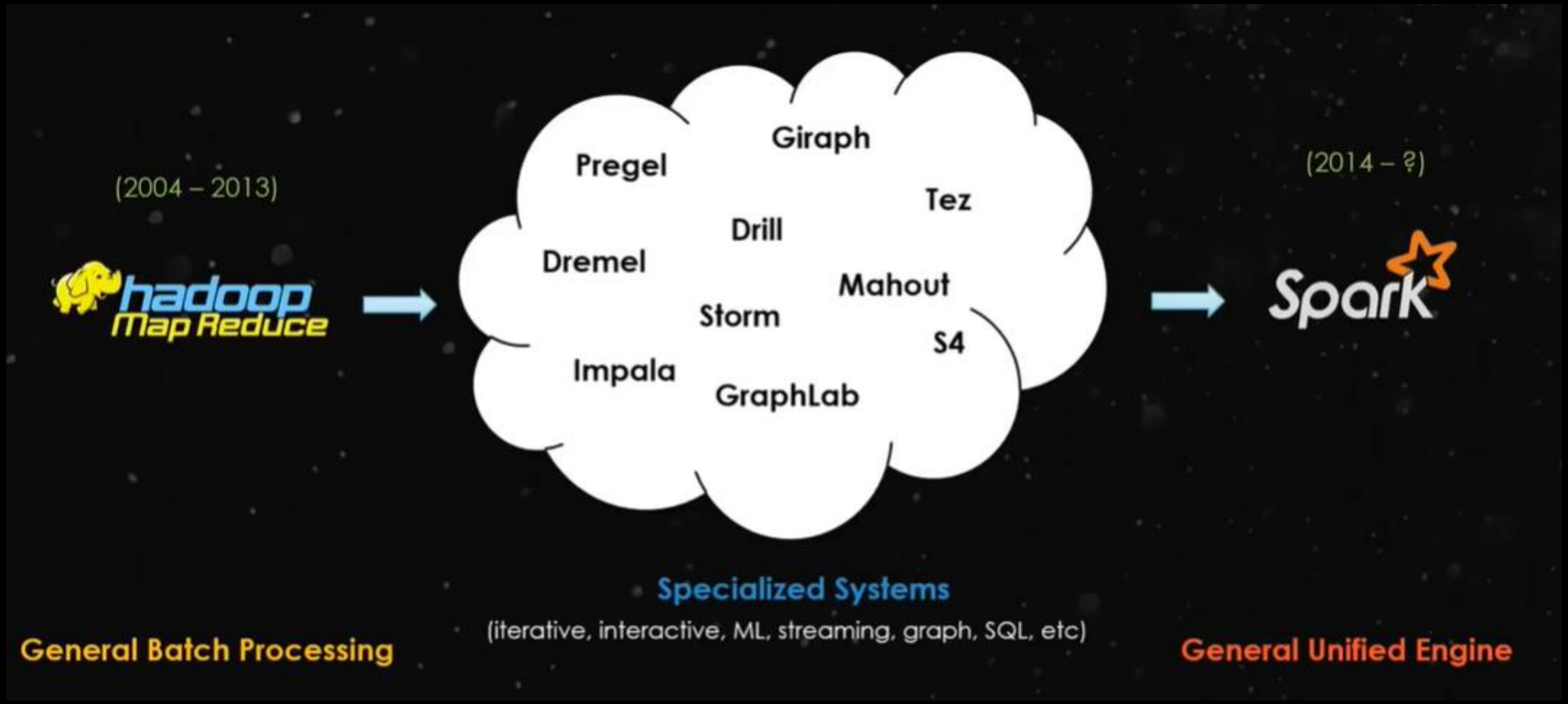
Computing Engine:

Spark provides parallel data computing/processing possible like MapReduce in Hadoop . The difference lies in that Mapreduce works only on top of the data stored in hadoop where as Spark can work on top of any distributed storage relieving the end user from worrying about where to store/retrieve his data.

History

- In 2009 when University of Berkley created a new resource manager in Hadoop called MESOS, Spark was created as a programming Framework to test the functionality of MESOS.
- By 2010 people found that Spark is more faster than MapReduce.
- By 2012 the developers of Spark have submitted Spark to Apachie to develop it as an opensource.
- Versions of Spark 0, 1.6, 2.0 ,..2.3
- Databricks is the company established by the developers of Spark to provide spark as a standalone project
- Spark works on top of Hadoop.

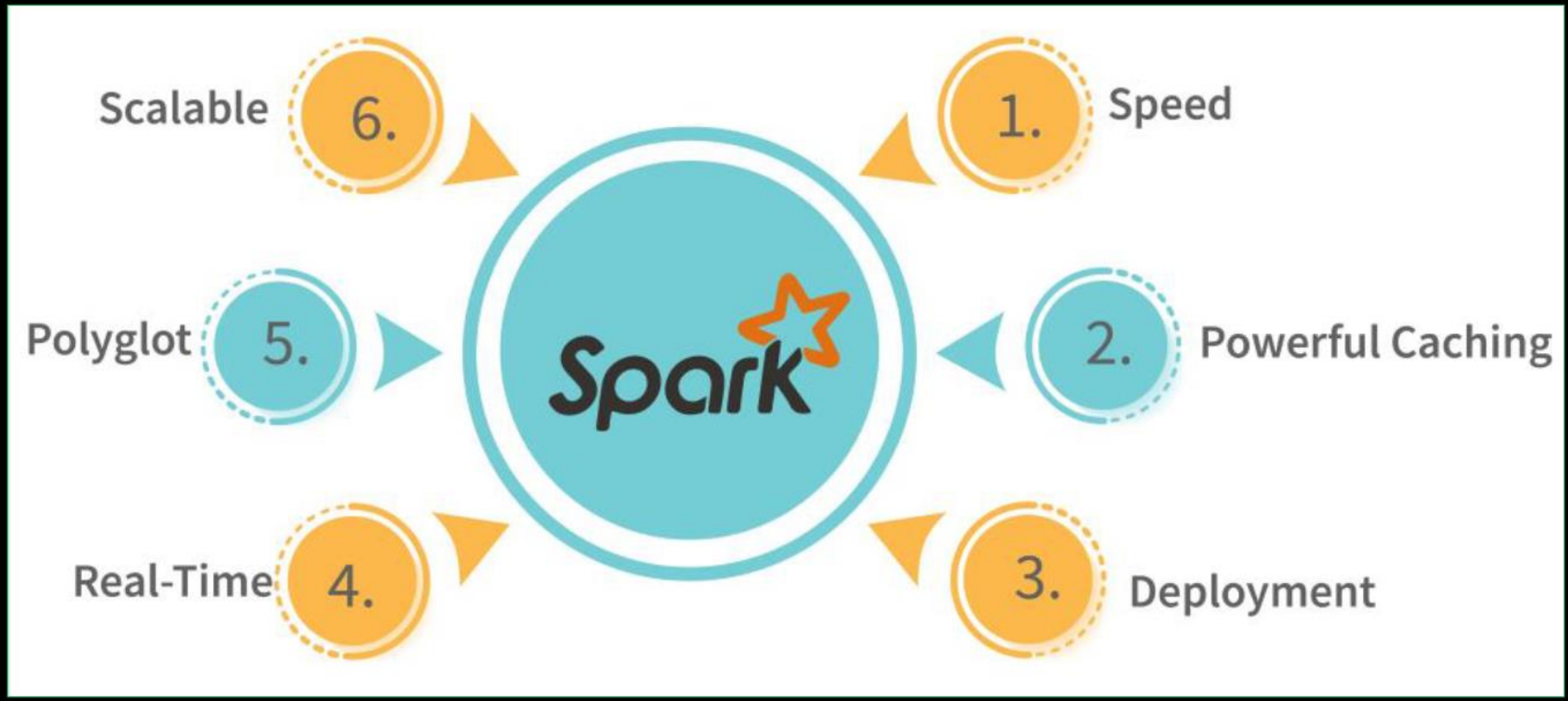
Importance of Spark Framework



Importance of Spark Framework

Apache Spark	MapReduce
Spark processes data in batches as well as in real-time.	MapReduce processes data in batches only.
Spark runs almost 100 times faster than Hadoop MapReduce.	Hadoop MapReduce is slower when it comes to large scale data processing.
Spark stores data in the RAM i.e. in-memory. So, it is easier to retrieve it.	Hadoop MapReduce data is stored in HDFS and hence takes a long time to retrieve the data.
Spark provides caching and in-memory data storage.	Hadoop is highly disk-dependent.

Apache Spark Features:



Spark Features

Better analytics

Spark has a rich set of SQL queries, machine learning algorithms, complex analytics, etc. With all these functionalities, analytics can be performed better

Fast processing

Spark contains Resilient Distributed Datasets (RDD) which saves time taken in reading, and writing operations and hence, it runs almost ten to hundred times faster than Hadoop

Flexible

Spark supports multiple languages and allows the developers to write applications in Java, Scala, R, or Python

In-memory computing

In Spark, data is stored in the RAM, so it can access the data quickly and accelerate the speed of analytics

Fault tolerance

Spark contains Resilient Distributed Datasets (RDD) that are designed to handle the failure of any worker node in the cluster. Thus, it ensures that the loss of data reduces to zero

Spark Features

- Speed:

Spark performs up to 100 times faster than MapReduce for processing large amounts of data.

It is also able to divide the data into chunks in a controlled way.

- Powerful Caching:

Powerful caching and disk persistence capabilities are offered by a simple programming layer.

- Deployment: Mesos, Hadoop via YARN, or Spark's own cluster manager can all be used to deploy it.
- Real-Time: Because of its in-memory processing, it offers real-time computation and low latency.

- Polyglot: In addition to Java, Scala, Python, and R, Spark also supports all four of these languages.

We can write Spark code in any one of these languages. Spark also provides a command-line interface in Scala and Python.


hadoop
HDFS
mongoDB
Learning for Life
RDBMS
FILE://
Neo4j
APACHE
HBASE
SQL
Streaming
GraphX
MLlib
Hadoop Input Format
FLUME

Layered View of Spark

- Layer 1 : When downloaded we can get Spark core which can run on both RAM and Hard disk
- Layer2 : Supporting Languages, Java, R, Python, Scala
  Spark was written in Scala
  With Spark 1.6 Java, R, Python used to run slow due to lack of optimization
- Layer 3: Components of Spark
- Layer 4: Modes of Operation
- Layer 5: Echo System of Spark

Components of Apache Spark

Components of Spark Unified Stack

Spark Core:

- The Spark Core is the heart of Spark and performs the core functionality.
- It holds the components for task scheduling, fault recovery, interacting with storage systems and memory management

Spark SQL:

- The Spark SQL is built on the top of Spark Core. It provides support for structured data.
- It allows to query the data via SQL (Structured Query Language) as well as the Apache Hive variant of SQL called the HQL (Hive Query Language).
- It supports JDBC and ODBC connections that establish a relation between Java objects and existing databases, data warehouses and business intelligence tools.
- It also supports various sources of data like Hive tables, Parquet, and JSON.

Components of Spark Unified Stack

Spark Streaming:

- Spark Streaming is a Spark component that supports scalable and fault-tolerant processing of streaming data.
- It uses Spark Core's fast scheduling capability to perform streaming analytics.
- It accepts data in mini-batches and performs RDD transformations on that data.
- Its design ensures that the applications written for streaming data can be reused to analyze batches of historical data with little modification.
- The log files generated by web servers can be considered as a real-time example of a data stream

Components of Spark Unified Stack

MLlib:

- The MLlib is a Machine Learning library that contains various machine learning algorithms.
- These include correlations and hypothesis testing, classification and regression, clustering, and principal component analysis.
- It is nine times faster than the disk-based implementation used by Apache Mahout.

Components of Spark Unified Stack

GraphX:

- The GraphX is a library that is used to manipulate graphs and perform graph parallel computations.
- It facilitates to create a directed graph with arbitrary properties attached to each vertex and edge.
- To manipulate graph, it supports various fundamental operators like subgraph, join Vertices, and aggregate Messages.

Spark Core

Spark Core is the base engine for large-scale parallel and distributed data processing

It is responsible for:

memory management

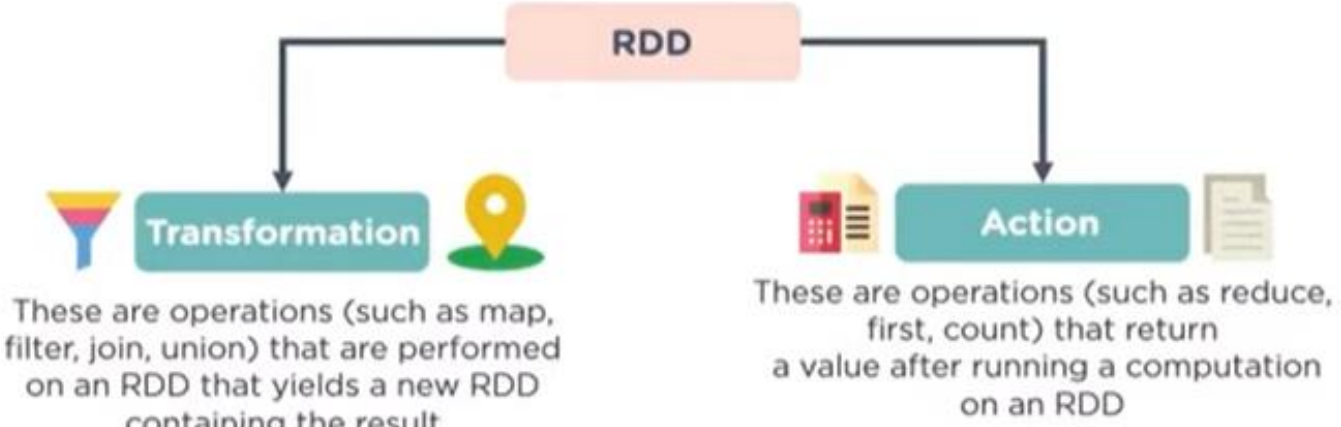
fault recovery

scheduling, distributing and monitoring jobs on a cluster

interacting with storage systems

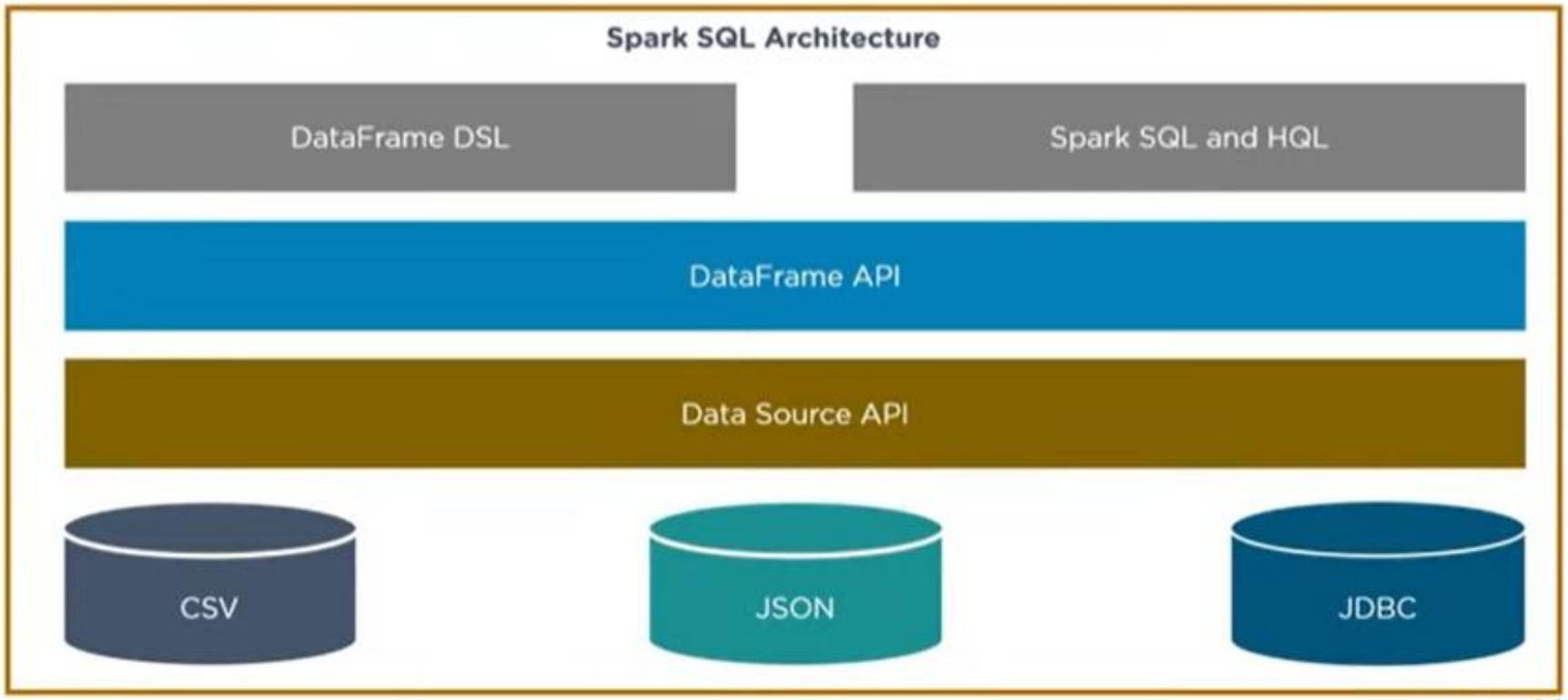
Resilient Distributed Dataset

Spark Core is embedded with RDDs (Resilient Distributed Datasets), an immutable fault-tolerant, distributed collection of objects that can be operated on in parallel



Spark SQL

Spark SQL framework component is used for structured and semi-structured data processing



Spark Streaming

Spark Streaming is a lightweight API that allows developers to perform batch processing and real-time streaming of data with ease

Provides secure, reliable, and fast processing of live data streams

Spark Streaming

Spark MLlib

MLlib is a low-level machine learning library that is simple to use, is scalable, and compatible with various programming languages

MLlib eases the deployment and development of scalable machine learning algorithms

It contains machine learning libraries that have an implementation of various machine learning algorithms

Clustering

Classification

Collaborative Filtering

GraphX

GraphX is Spark's own Graph Computation Engine and data store

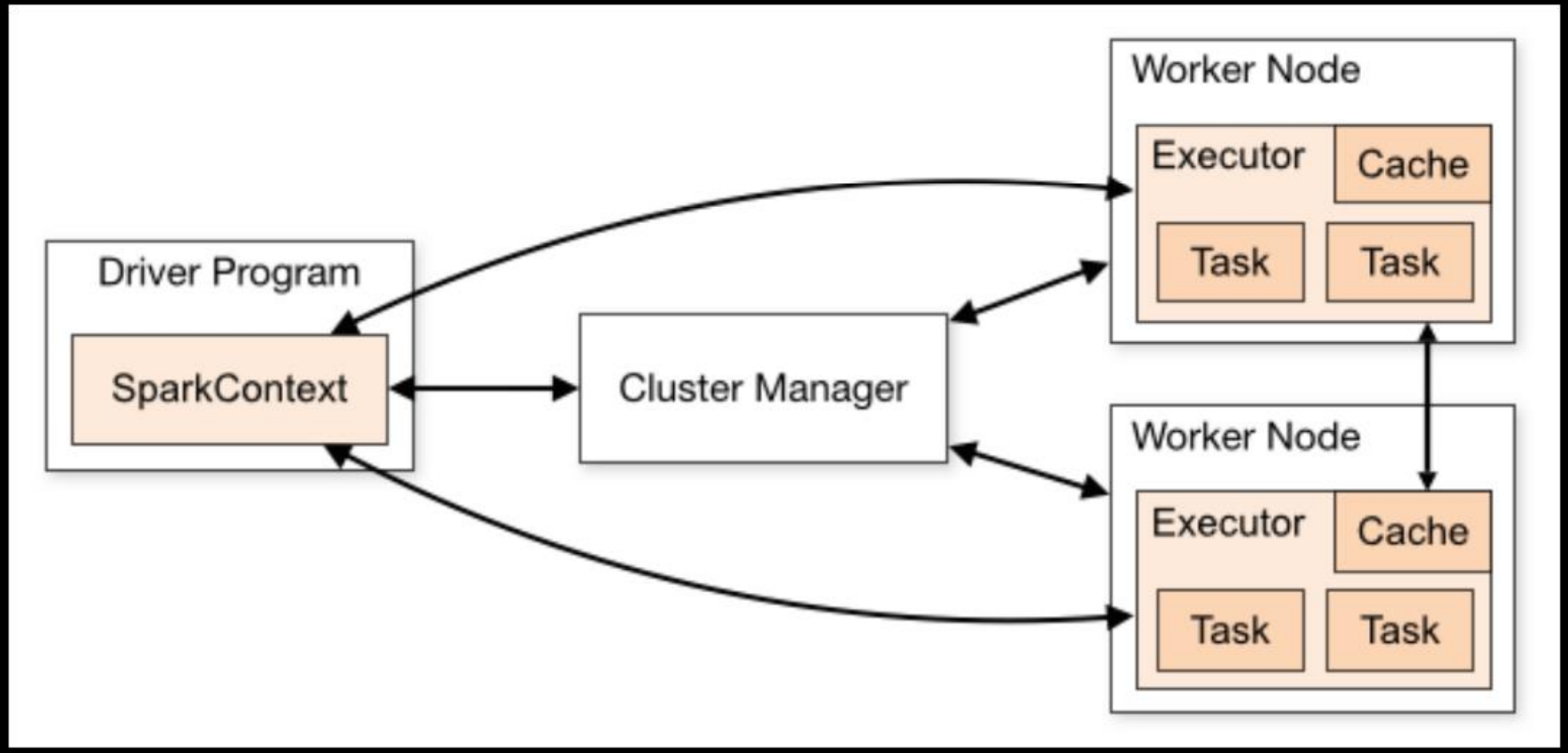
GraphX

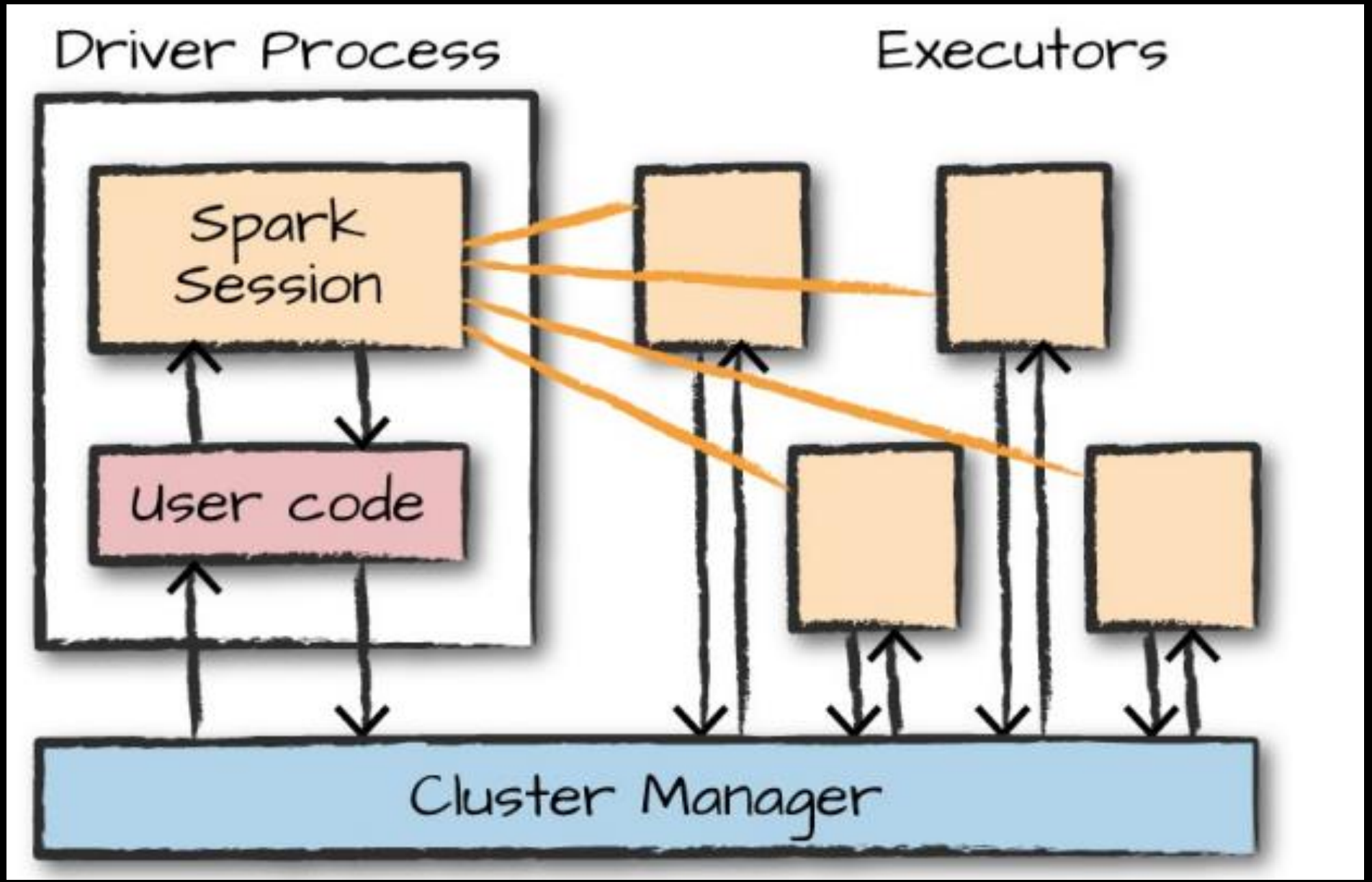
Provides a uniform tool for ETL

Exploratory data analysis

Interactive graph computations

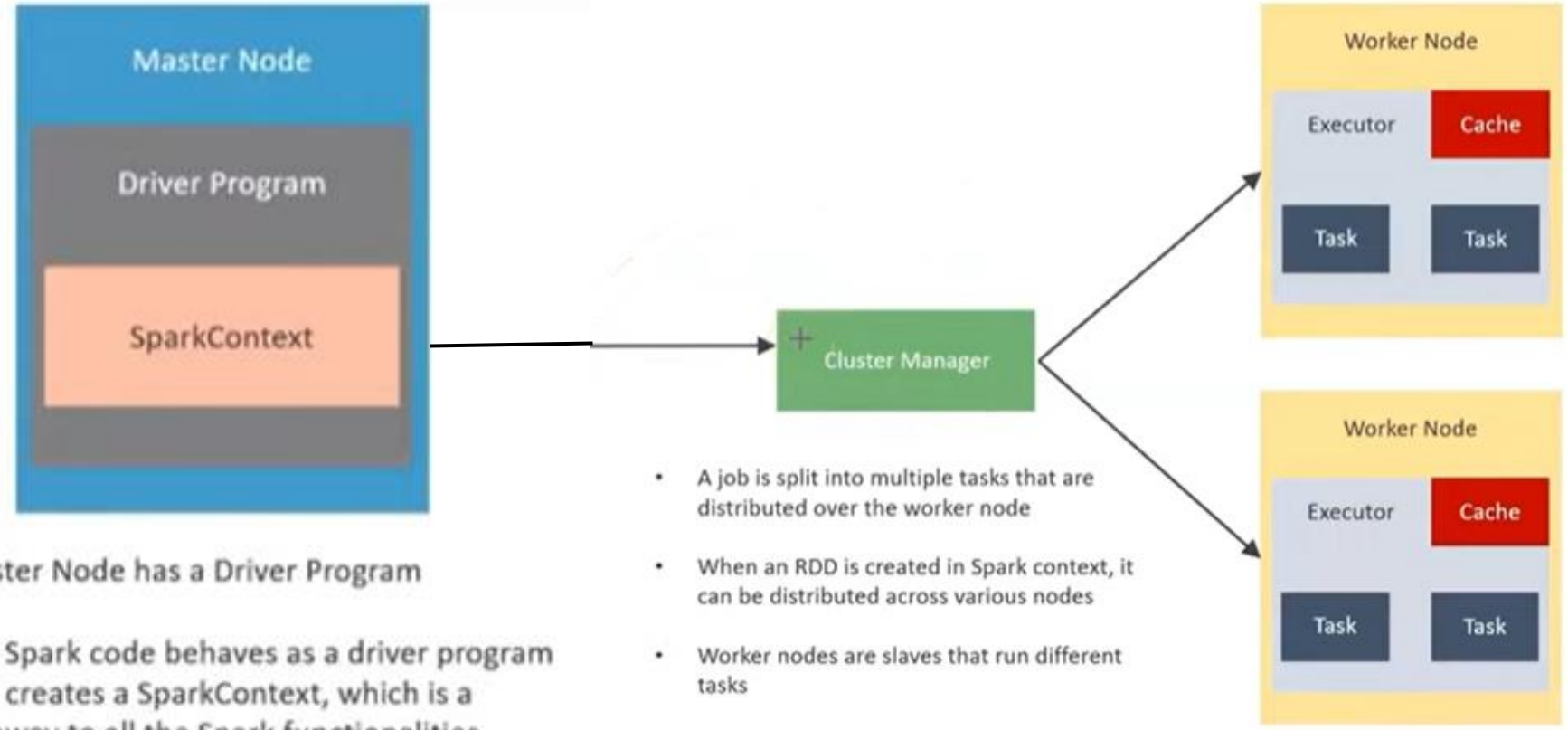
Spark Application Architecture




Driver Process
Executors
Spark Session
User code
Cluster Manager

Spark Architecture

Apache Spark uses a master-slave architecture that consists of a driver, that runs on a master node, and multiple executors which run across the worker nodes in the cluster



- Master Node has a Driver Program
- The Spark code behaves as a driver program and creates a SparkContext, which is a gateway to all the Spark functionalities

- A job is split into multiple tasks that are distributed over the worker node
- When an RDD is created in Spark context, it can be distributed across various nodes
- Worker nodes are slaves that run different tasks

Spark Cluster Managers

Standalone mode

1

By default, applications submitted to the standalone mode cluster will run in FIFO order, and each application will try to use all available nodes

2

Apache Mesos is an open-source project to manage computer clusters, and can also run Hadoop applications

3

Apache YARN is the cluster resource manager of Hadoop 2. Spark can be run on YARN

kubernetes

4

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications

Spark Application Architecture

Spark applications run as independent sets of processes on a cluster, coordinated by the Spark Context object in our main program (called the driver program).

Specifically, to run on a cluster, the Spark Context can connect to several types of cluster managers (either Spark's own standalone cluster manager, Mesos, YARN or Kubernetes), which allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for our application. Next, it sends our application code (defined by JAR or Python files passed to Spark Context) to the executors. Finally, Spark Context sends tasks to the executors to run.

Spark Application Architecture

Useful things to note

1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs).

   However, it also means that data cannot be shared across different Spark applications (instances of Spark Context) without writing it to an external storage system.

2. Spark is computing to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN/Kubernetes).

Spark Application Architecture

3. The driver program must listen for and accept incoming connections from its executors throughout its lifetime. As such, the driver program must be network addressable from the worker nodes.

4. Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network. If you'd like to send requests to the cluster remotely, it's better to open an RPC to the driver and have it submit operations from nearby than to run a driver far away from the worker nodes.

Spark Application Architecture

A high-level view of the architecture of the Apache Spark application is as follows:

The Spark driver:

- The master node (process) in a driver process coordinates workers and oversees the tasks.
- Spark is split into jobs and scheduled to be executed on executors in clusters.

Spark contexts (gateways) are created by the driver to monitor the job working in a specific cluster and to connect to a Spark cluster.

- In the diagram, the driver programs call the main application and create a spark context (acts as a gateway) that jointly monitors the job working in the cluster and connects to a Spark cluster.
- Everything is executed using the spark context.

The Spark driver:

- Each Spark session has an entry in the Spark context.
- Spark drivers include more components to execute jobs in clusters, as well as cluster managers.
- Context acquires worker nodes to execute and store data as Spark clusters are connected to different types of cluster managers.
- When a process is executed in the cluster, the job is divided into stages with gain stages into scheduled tasks.

Spark Application Architecture

The Spark Context:

- Each Spark session has an entry in the Spark context.
- Spark drivers include more components to execute jobs in clusters, as well as cluster managers.
- Context acquires worker nodes to execute and store data as Spark clusters are connected to different types of cluster managers.
- When a process is executed in the cluster, the job is divided into stages with gain stages into scheduled tasks. Spark Architecture Applications Prof R Madana Mohana || Big Data

Spark Application Architecture

The Spark executors:

- An executor is responsible for executing a job and storing data in a cache at the outset.
- Executors first register with the driver programme at the beginning. • These executors have a number of time slots to run the application concurrently.
- The executor runs the task when it has loaded data and they are removed in idle mode.
- The executor runs in the Java process when data is loaded and removed during the execution of the tasks.
- The executors are allocated dynamically and constantly added and removed during the execution of the tasks.
- A driver program monitors the executors during their performance. Users' tasks are executed in the Java process.

Spark Application Architecture

Cluster Manager:

A driver program controls the execution of jobs and stores data in a cache.

- At the outset, executors register with the drivers.
- This executor has a number of time slots to run the application concurrently.
- Executors read and write external data in addition to servicing client requests.
- A job is executed when the executor has loaded data and they have been removed in the idle state.
- The executor is dynamically allocated, and it is constantly added and deleted depending on the duration of its use.
- A driver program monitors executors as they perform users' tasks. • Code is executed in the Java process when an executor executes a user's task.

Spark Application Architecture

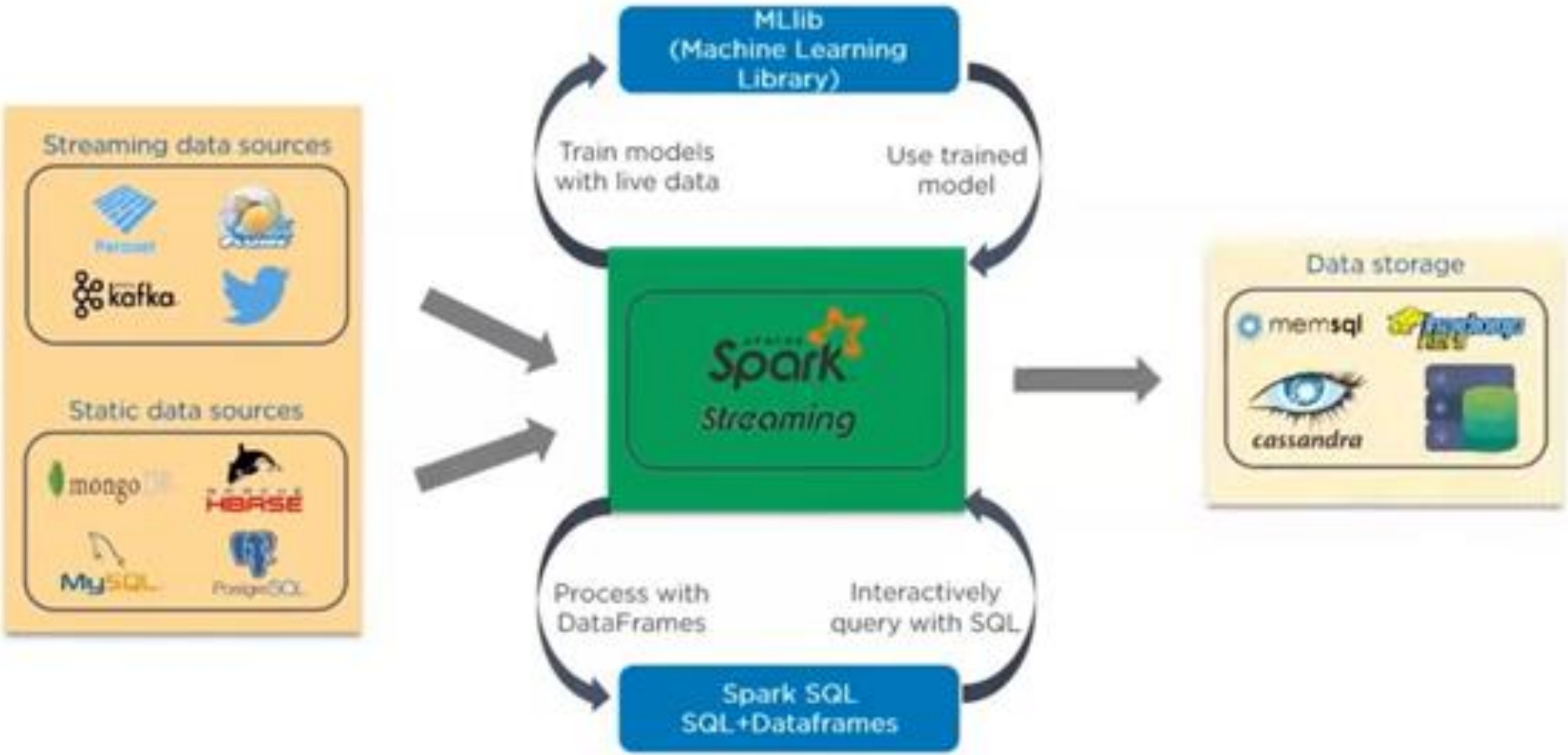
Worker Nodes:

- The slave nodes function as executors, processing tasks, and returning the results back to the spark context.
- The master node issues tasks to the Spark context and the worker nodes execute them.
- They make the process simpler by boosting the worker nodes (1 to n) to handle as many jobs as possible in parallel by dividing the job up into sub-jobs on multiple machines.
- A Spark worker monitors worker nodes to ensure that the computation is performed simply.
- Each worker node handles one Spark task.
- In Spark, a partition is a unit of work and is assigned to one executor for each one.

Features of Spark Streaming

1. Fast recovery from failures
2. Better load balancing and resource usage
3. Combining the streaming data with static datasets and interactive queries
4. Native integration with advanced processing libraries

Spark Streaming data sources



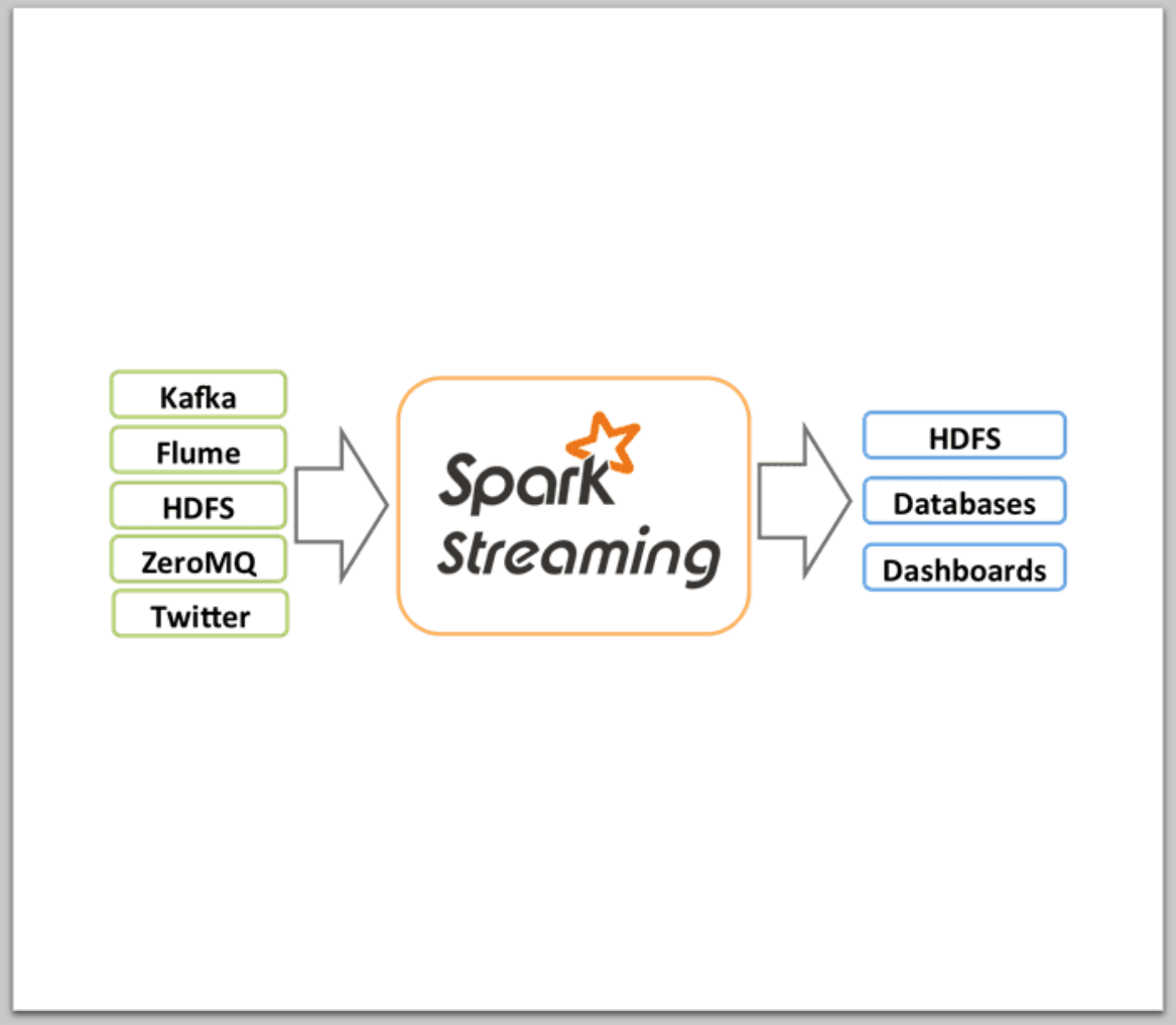
RealTime Analytics with Spark

- Apache Spark architecture allows a **continuous stream** of data by dividing the stream into **micro**-batches called Discretized stream or Dstream which is an API
- Dstream is a sequence of RDDs that are created from input data or from sources such as Kafka, Flume, or by applying operations on other Dstream
- RDDs thus generated can be converted into data frames and queried using Spark SQL.
- Dstream can be subjected to any application that can query RDD through Spark's JDBC driver and stored in Spark's working memory to query it later on-demand of Spark's API.

RealTime Analytics with Spark

Most Real time Analytics Systems can be broken down into

- A Receiver System,
- a Stream Processing System and
- a Storage System.



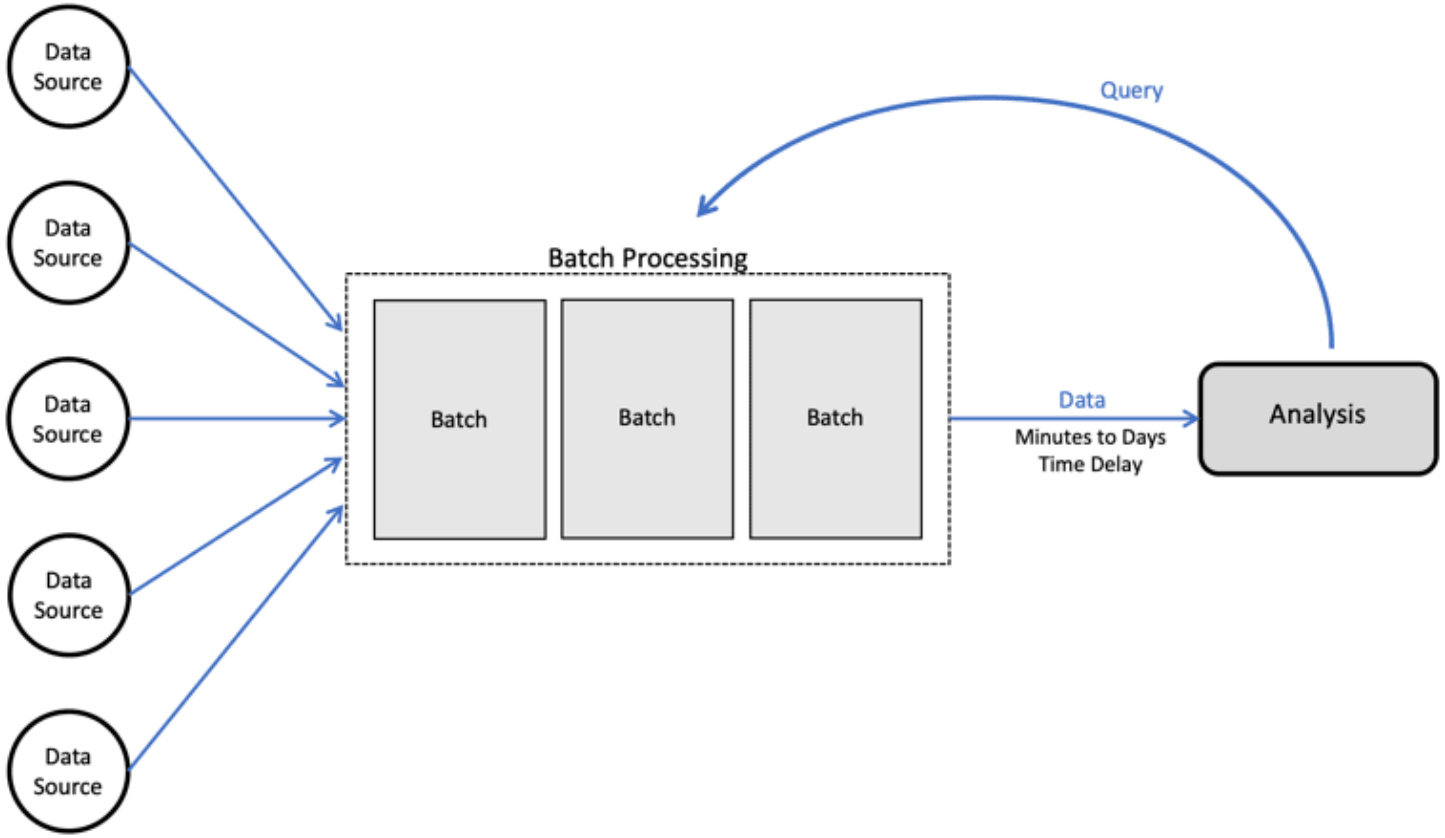
Batch Analytics with Spark

- Batch processing is used to deal with enormous amounts of data for implementing high-volume and repeating data jobs, each of which performs a specific operation without the need for user intervention.

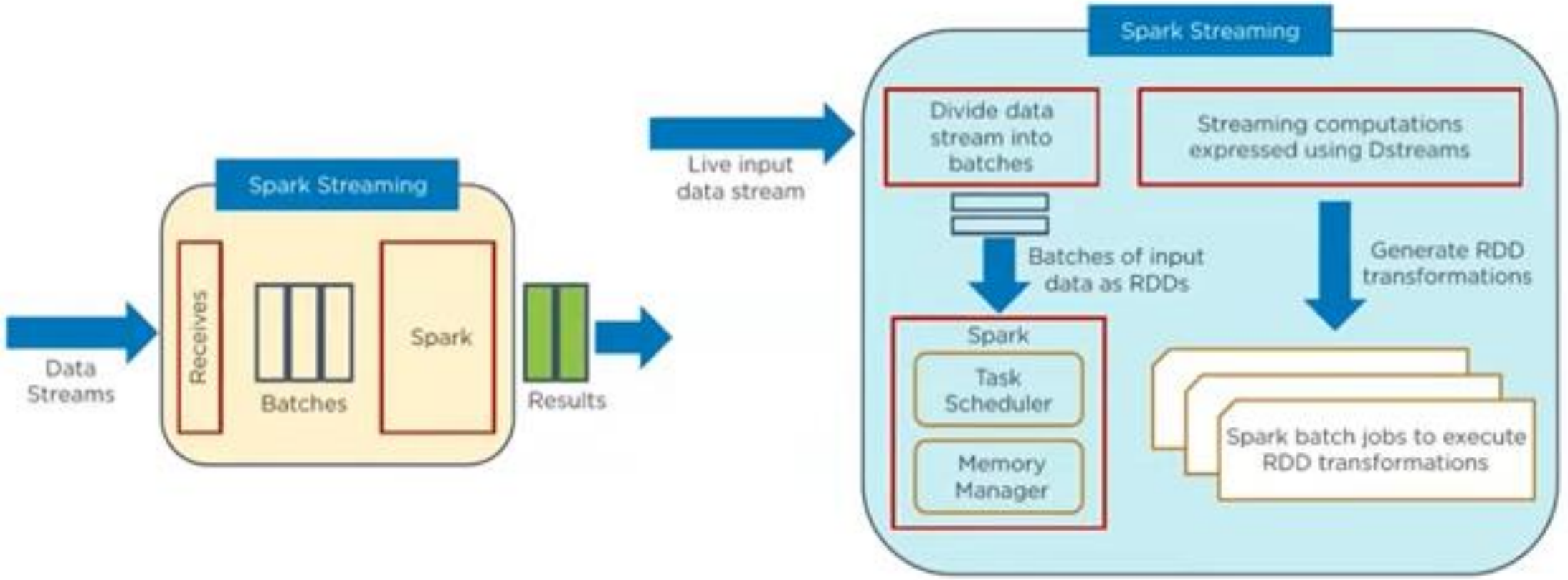
- This technique is especially useful for repetitive and monotonous operations for supporting several data workflows.

- Due to significant gains in precision and accuracy through automation, organizations can achieve superior data quality while decreasing bottlenecks in data processing activities.

Batch Analytics with Spark



Working of Spark Streaming



Discretized Streams (DStreams)

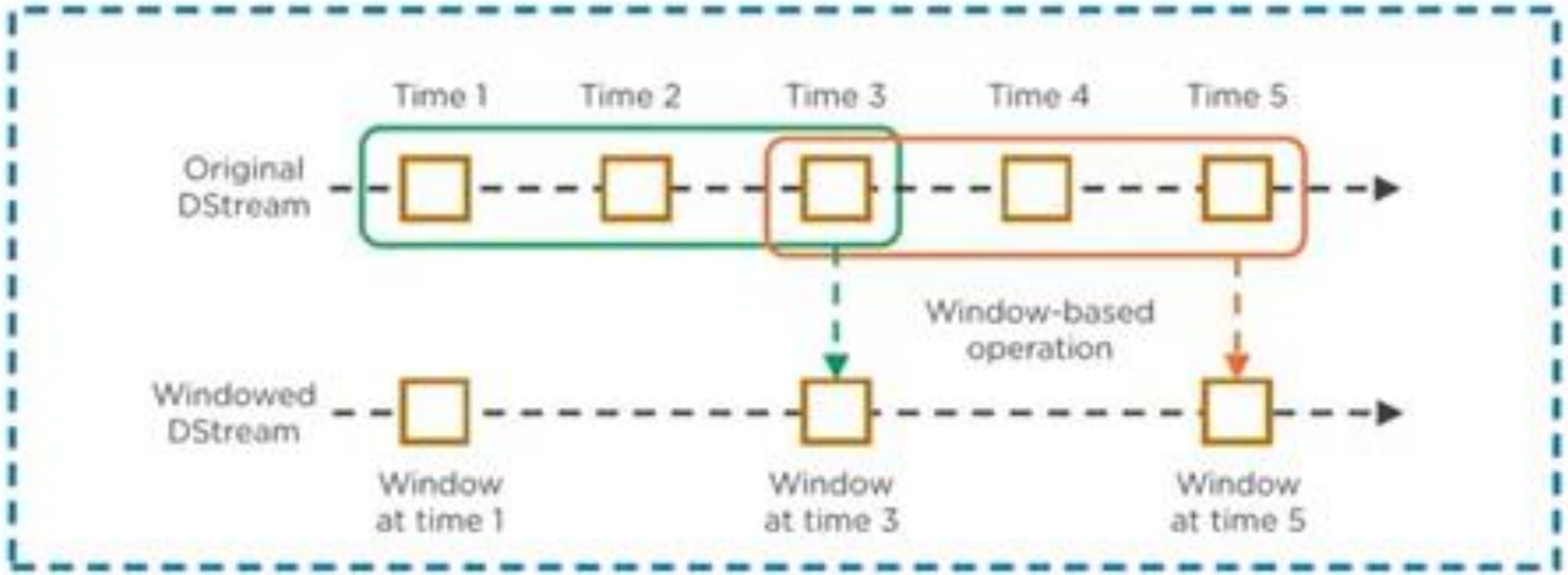
Discretized Stream is the basic abstraction provided by Spark Streaming. It represents a continuous stream of data, either the input data stream received from the source or the processed data stream generated by transforming the input stream

Transformations on DStreams

Transformation	Meaning
map(func)	Return a new DStream by passing each element of the source DStream through a function *func*
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items
filter(func)	Return a new DStream by selecting only the records of the source DStream on which *func* returns true
union(otherStream)	Return a new DStream that contains the union of the elements in the source DStream and *otherDStream*
transform(func)	Return a new DStream that contains the union of the elements in the source DStream and *otherDStream*
count()	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream
join(otherStream, [numTasks])	When called on two DStreams of (K, V) and (K, W) pairs, return a new DStream of (K, (V, W)) pairs with all pairs of elements for each key

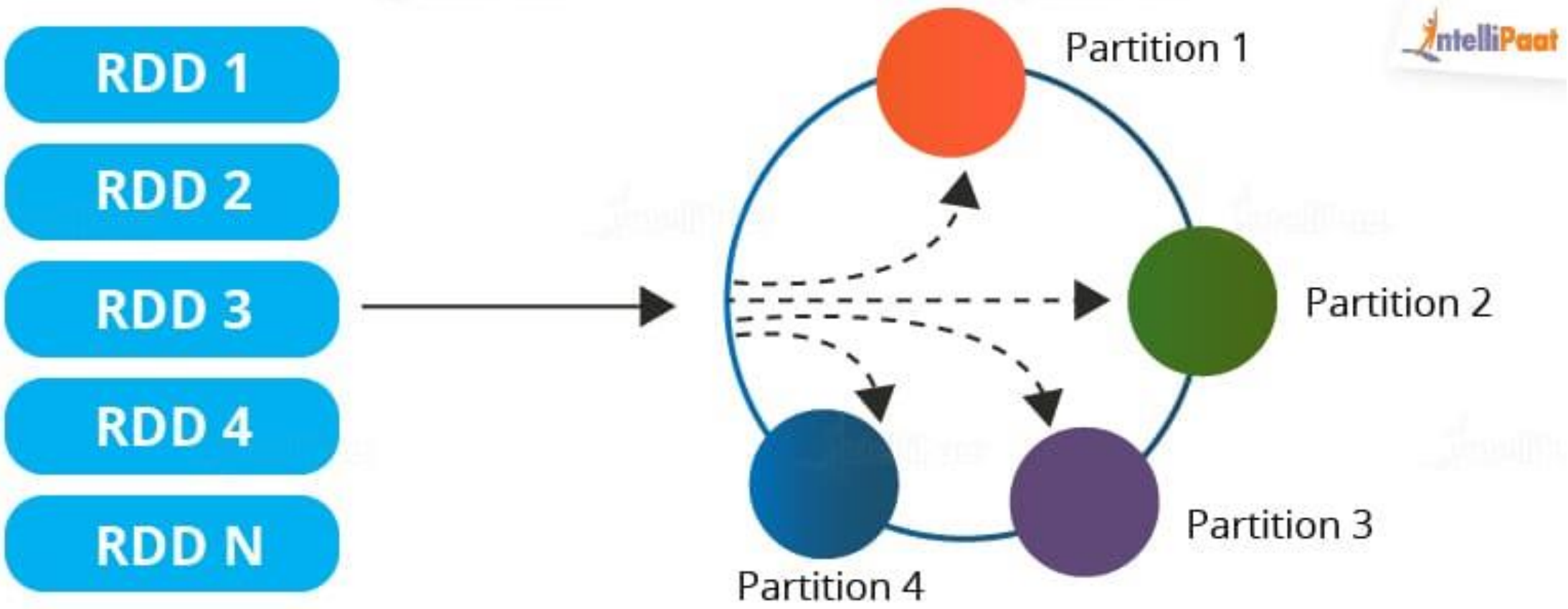
Windowed Stream Processing

Spark Streaming allows you to apply transformations over a sliding window of data.
This operation is called as *windowed computation*



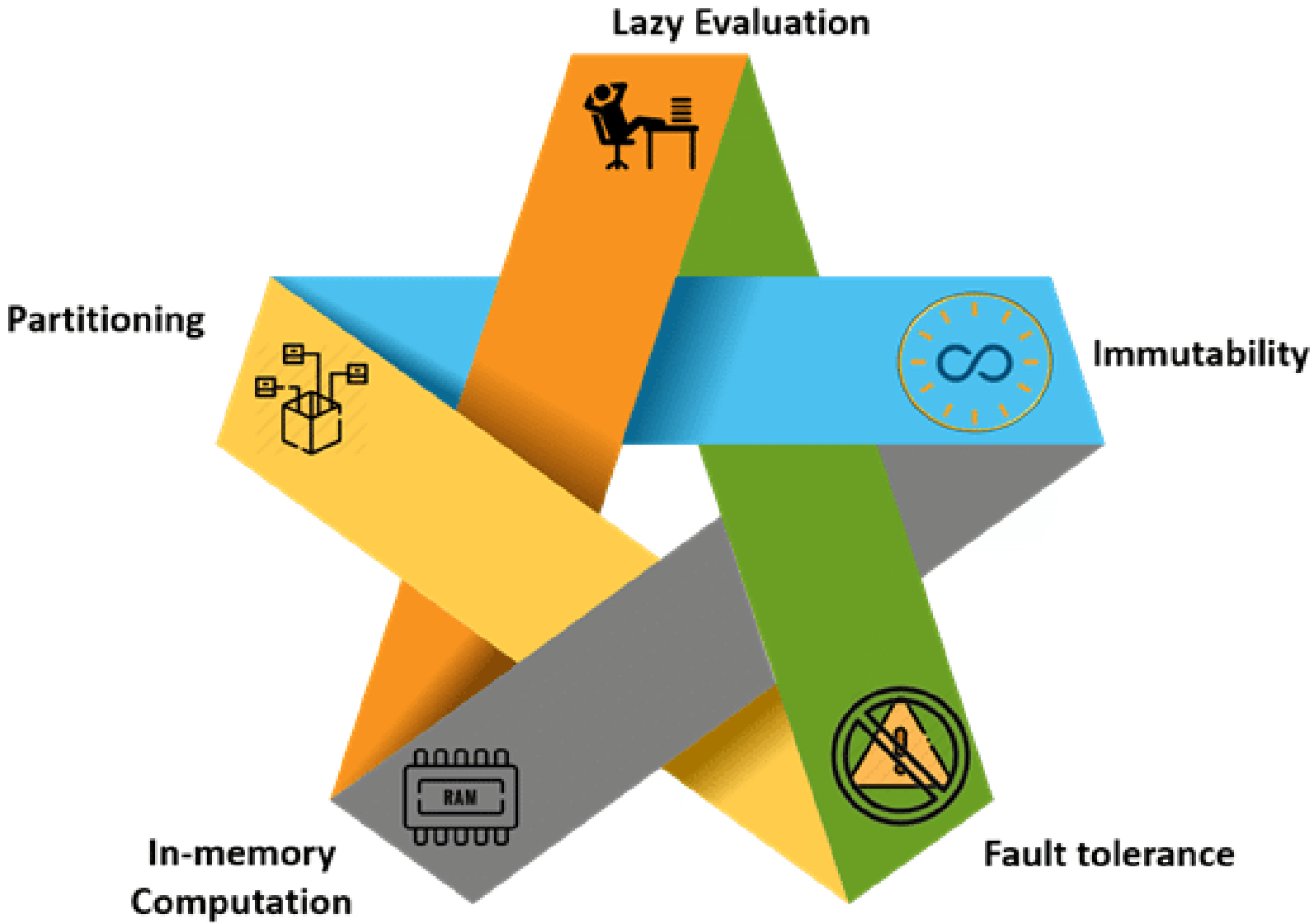
Resilient Distributed Dataset (RDD)

- RDDs are the main logical data units in Spark.
- They are a distributed collection of objects, which are stored in memory or on disks of different machines of a cluster.
- A single RDD can be divided into multiple logical partitions so that these partitions can be stored and processed on different machines of a cluster.
- RDDs are immutable (read-only) in nature.
- You cannot change an original RDD, but you can create new RDDs by performing coarse-grain operations, like transformations, on an existing RDD.
- An RDD in Spark can be cached and used again for future transformations, which is a huge benefit for users.
- RDDs are said to be lazily evaluated, i.e., they delay the evaluation until it is really needed. Thus saves time.


RDD 1
RDD 2
RDD 3
RDD 4
RDD N
Partition 1
Partition 2
Partition 3
Partition 4
IntelliPaat

Features of an RDD in Spark

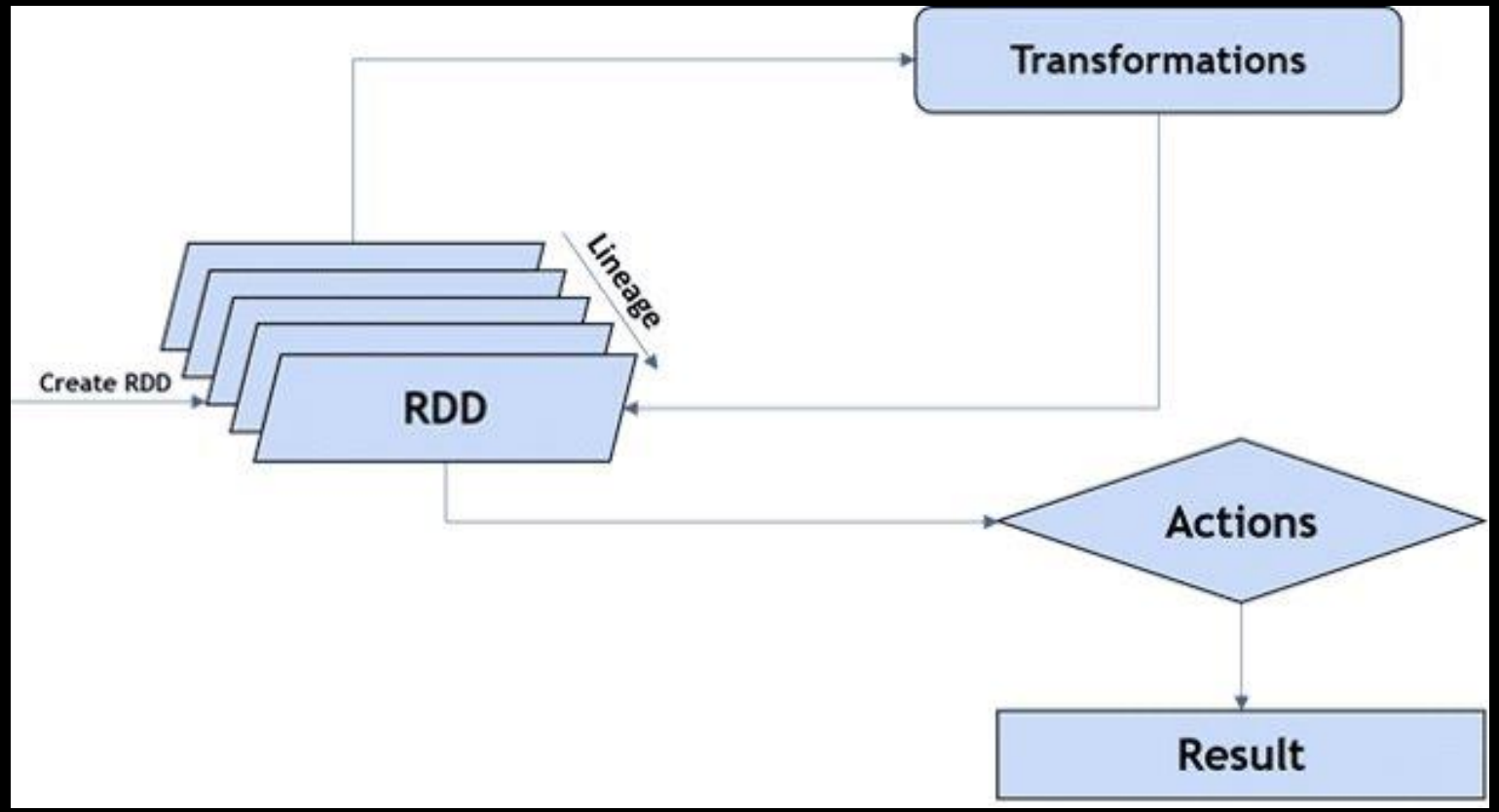
- **Resilience:** RDDs track data lineage information to recover lost data, automatically on failure. It is also called fault tolerance.
- **Distributed:** Data present in an RDD resides on multiple nodes. It is distributed across different nodes of a cluster.
- **Lazy evaluation:** Data does not get loaded in an RDD even if you define it. Transformations are actually computed when you call action, such as count or collect, or save the output to a file system.


Lazy Evaluation
Partitioning
Immutability
RAM
In-memory
Computation
Fault tolerance

Features of an RDD in Spark

- **Immutability:** Data stored in an RDD is in the read-only mode—you cannot edit the data which is present in the RDD. But, you can create new RDDs by performing transformations on the existing RDDs.
- **In-memory computation:** An RDD stores any immediate data that is generated in the memory (RAM) than on the disk so that it provides faster access.
- **Partitioning:** Partitions can be done on any existing RDD to create logical parts that are mutable. You can achieve this by applying transformations to the existing partitions.

Operations on RDD



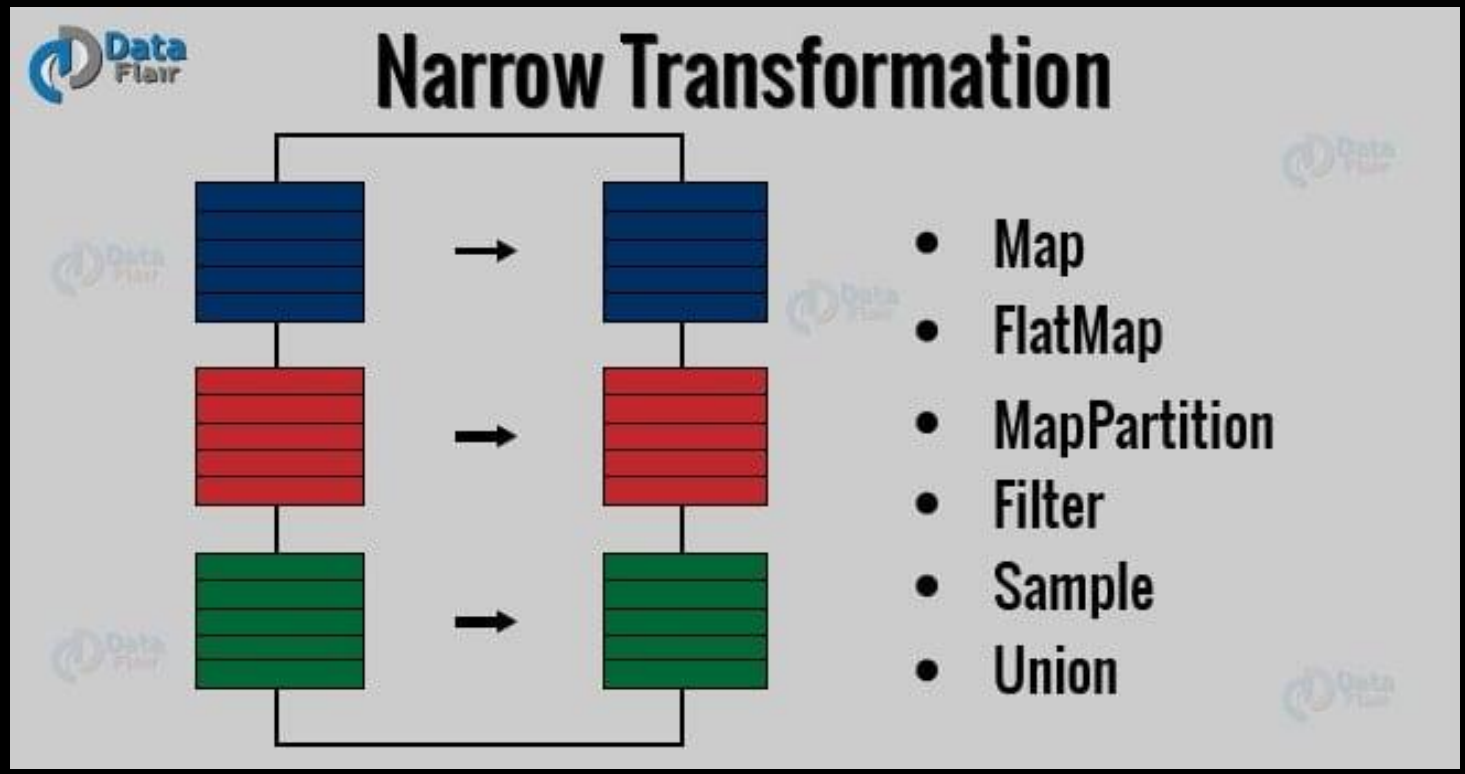
Transformations

- These are functions that accept the existing RDDs as input and output one or more RDDs.
- The data in the existing RDD in Spark does not change as it is immutable.

Function	Description
map()	Returns a new RDD by applying the function on each data element
filter()	Returns a new RDD formed by selecting those elements of the source on which the function returns true
reduceByKey()	Aggregates the values of a key using a function
groupByKey()	Converts a (key, value) pair into a (key, <iterable value>) pair
union()	Returns a new RDD that contains all elements and arguments from the source RDD
intersection()	Returns a new RDD that contains an intersection of the elements in the datasets

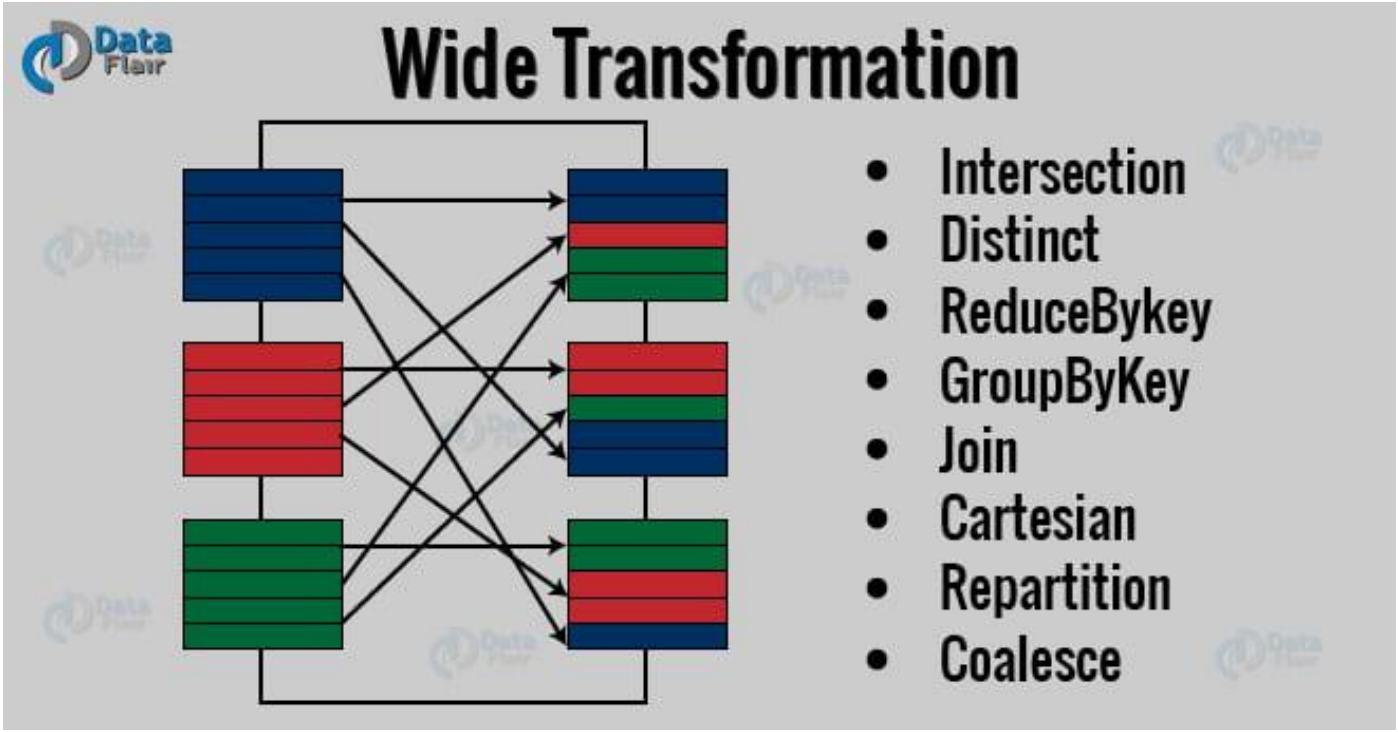
Types of Transformations

- **Narrow transformation –** In *Narrow transformation*, all the elements that are required to compute the records in single partition live in the single partition of parent RDD. A limited subset of partition is used to calculate the result. *Narrow transformations* are the result of *map(), filter().*



Types of Transformations

Wide transformation – In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of parent RDD. The partition may live in many partitions of parent RDD. *Wide transformations* are the result of *groupbyKey()* and *reducebyKey()*.



Actions

Actions are functions that return the end result of RDD computations.

It uses a lineage graph to load data onto the RDD in a particular order.

Function	Description
count()	Gets the number of data elements in an RDD
collect()	Gets all the data elements in an RDD as an array
reduce()	Aggregates data elements into an RDD by taking two arguments and returning one
take(n)	Fetches the first n elements of an RDD
foreach(operation)	Executes the operation for each data element in an RDD
first()	Retrieves the first data element of an RDD