Apache Spark

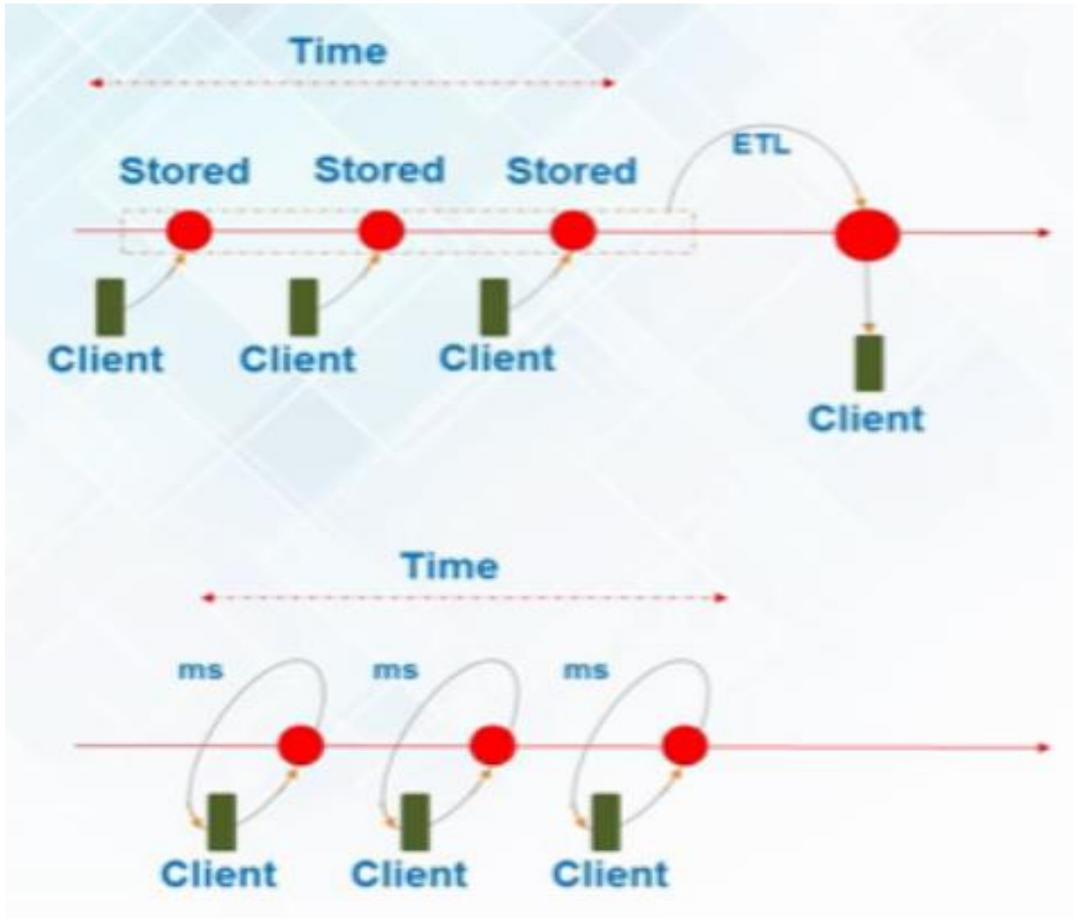
Types of Big Data Analytics

- Batch Analytics
- Real Time Analytics

Batch vs Real Time Analytics

- Analytics based on data collected over a period of time is Batch Analytics. e.g. Washing machine example

- Analytics based on real time data for instant result is Real Time Analytics/Stream Analytics. E.g. credit card example

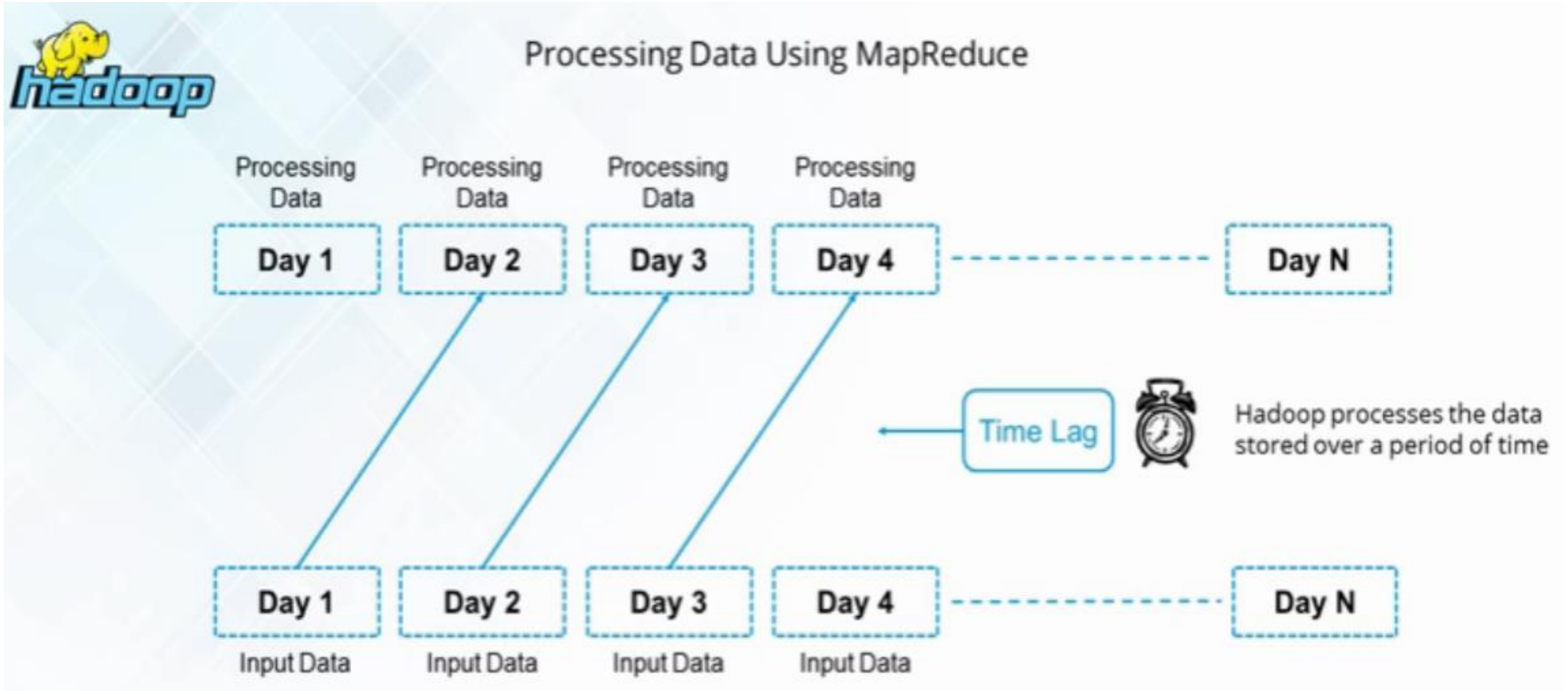


Use cases of Real Time Analytics

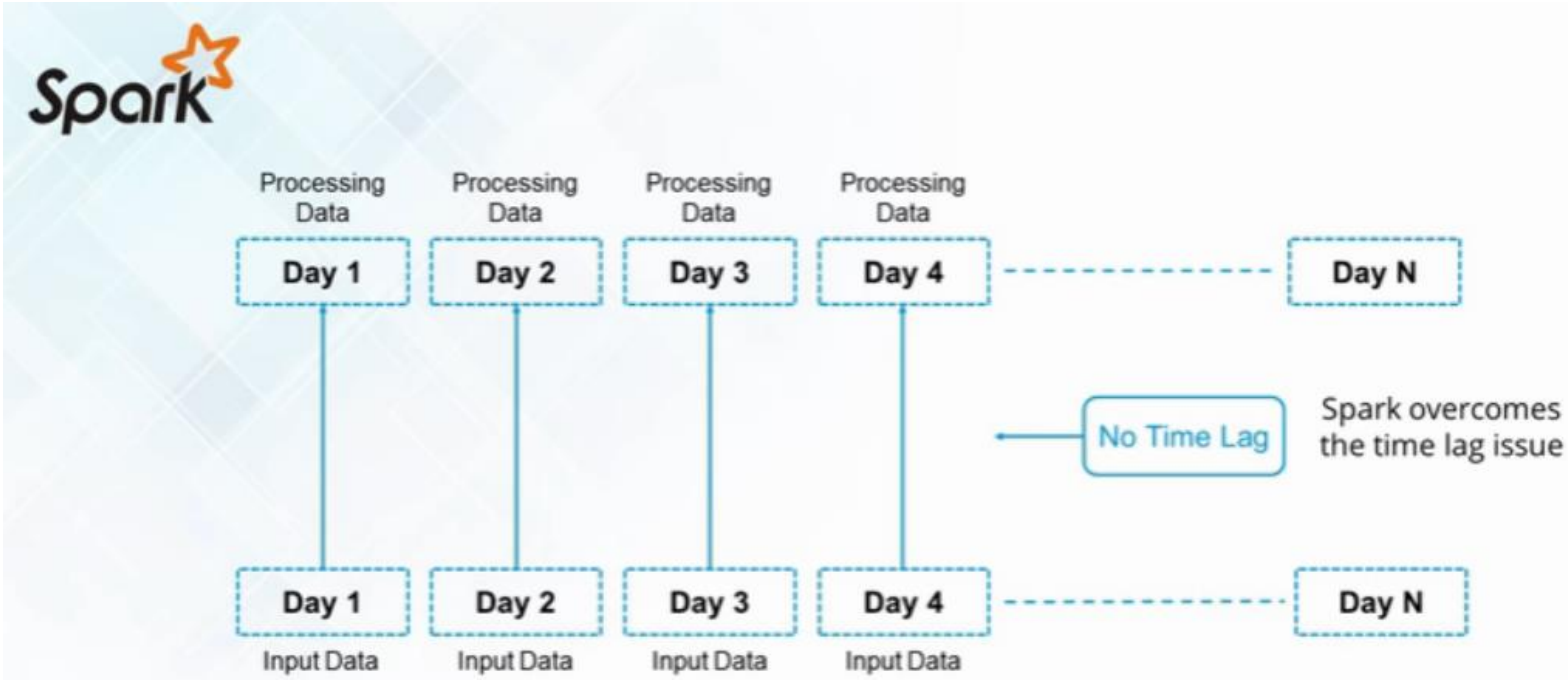
BZ-CBSSBI
Your AC XXXXX12345 Debited INR 12,345.67 on 28/06/13 - TFR. Avl Bal INR 9,876.54
Options Back
Banking
Government
Healthcare
Telecommunications
Stock Market

Why Spark when Hadoop is already there?

Batch Processing in Hadoop

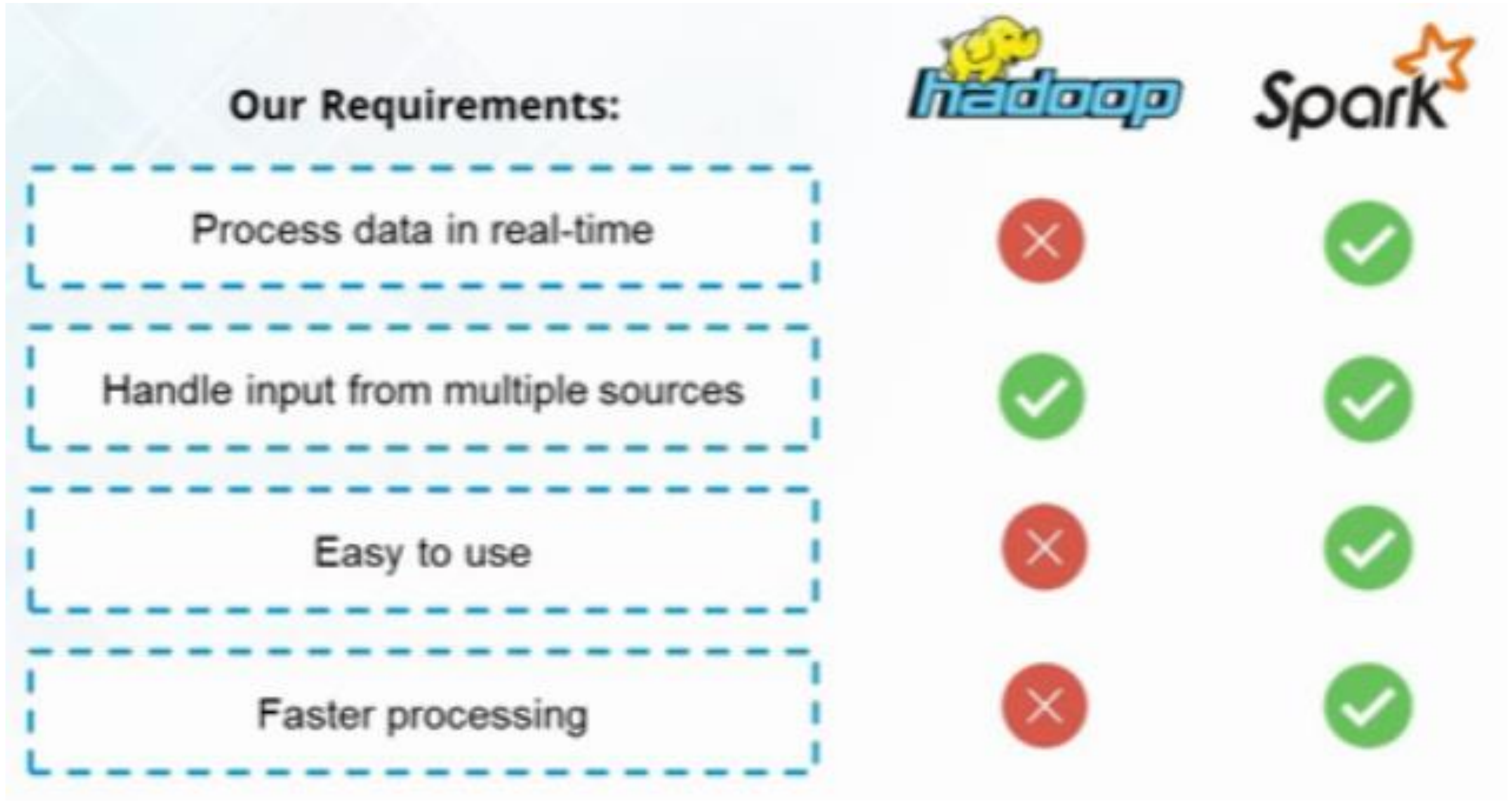


Real Time Processing in Spark



Spark vs Hadoop

- Hadoop implements Batch processing on big data and thus cannot deliver to Real Time use case needs.



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

Example

Spark Success Story

Twitter Sentiment Analysis With Spark

Trending Topics can be used to create campaigns and attract larger audience

Sentiment helps in crisis management, service adjusting and target marketing

NYSE: Real Time Analysis of Stock Market Data

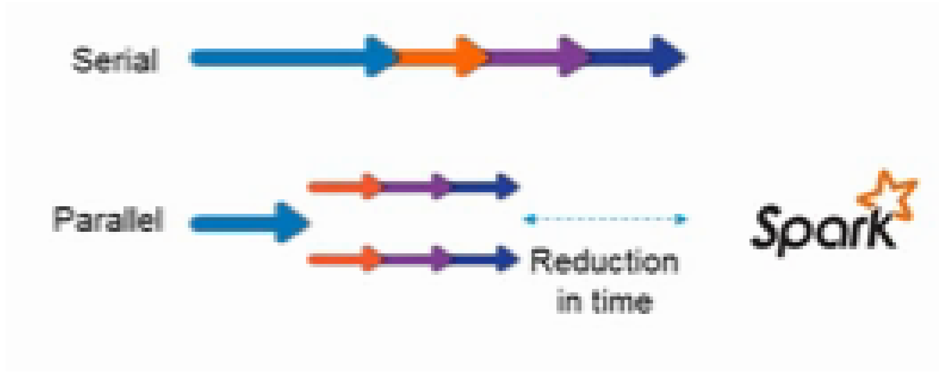
Banking: Credit Card Fraud Detection

Genomic Sequencing

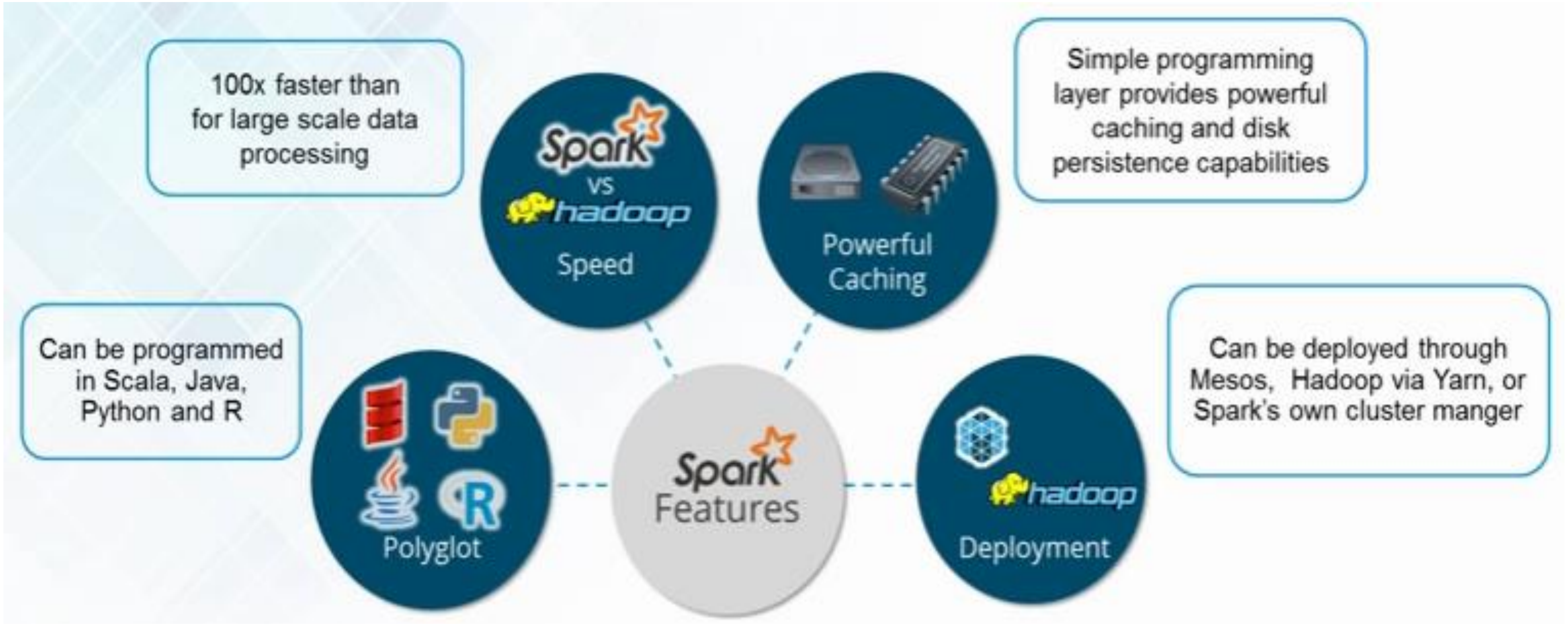
Spark Overview

What is Apache Spark?

- Apache Spark is an open-source cluster-computing framework for real time processing by Apache Software Foundation
- Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance
- It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations

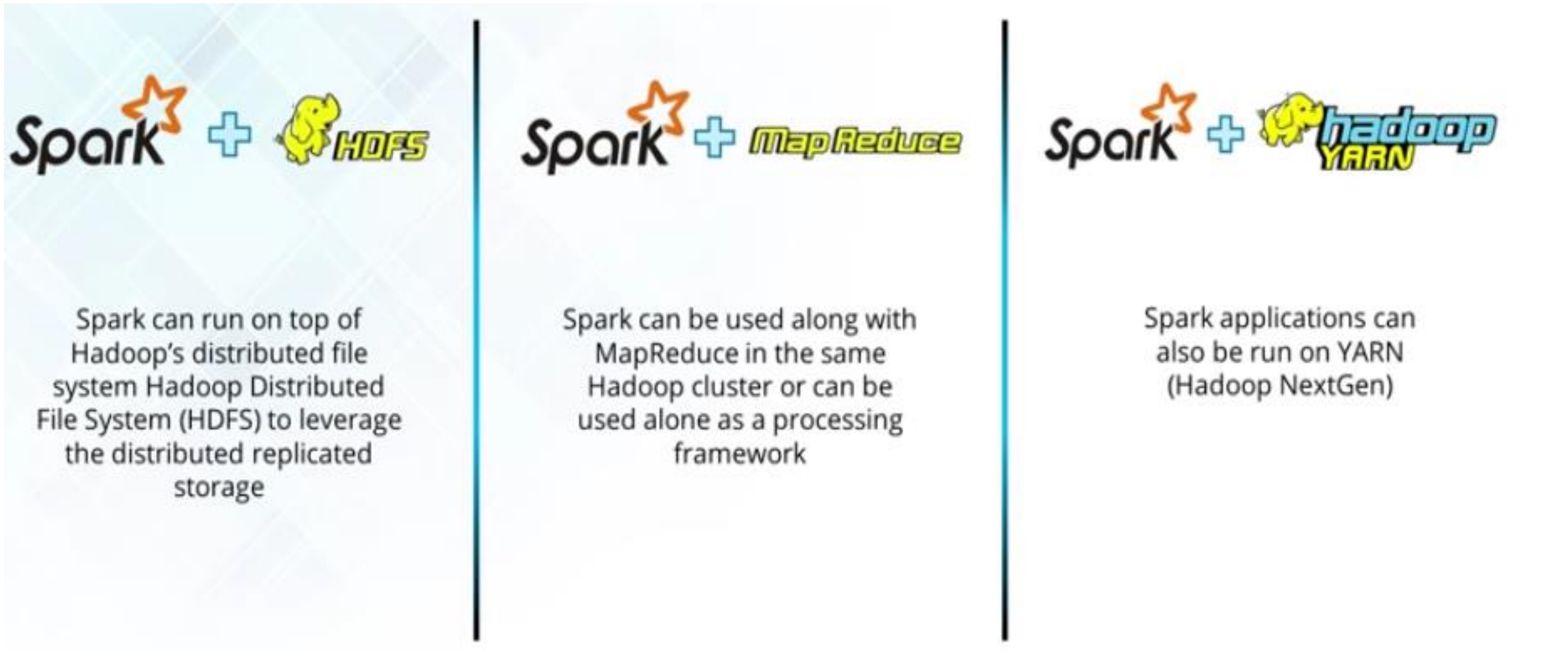


Why Spark?


100x faster than for large scale data processing
Spark vs hadoop
Speed
Powerful Caching
Simple programming layer provides powerful caching and disk persistence capabilities
Can be programmed in Scala, Java, Python and R
Polyglot
Spark Features
hadoop
Deployment
Can be deployed through Mesos, Hadoop via Yarn, or Spark's own cluster manger

Using Hadoop Through Spark

Spark and Hadoop



Spark + HDFS

Spark can run on top of Hadoop's distributed file system Hadoop Distributed File System (HDFS) to leverage the distributed replicated storage

Spark + MapReduce

Spark can be used along with MapReduce in the same Hadoop cluster or can be used alone as a processing framework

Spark + hadoop YARN

Spark applications can also be run on YARN (Hadoop NextGen)

Spark and Hadoop

Spark is not intended to replace Hadoop but it can regarded as an extension to it

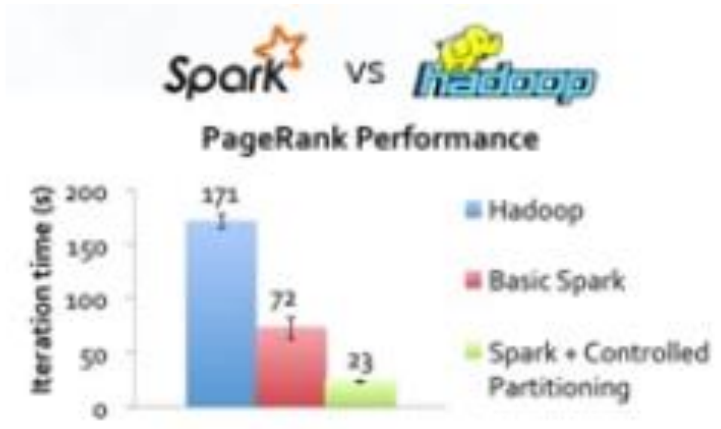
MapReduce and Spark are used together where MapReduce is used for batch processing and Spark for real-time processing

Spark Features

- Speed
- Polyglot
- Advanced Analytics
- In-Memory Computation
- Hadoop Integration
- Machine Learning

Spark Features

- Spark runs up to 100x times faster than MapReduce.



- Polyglot: Programming in Scala, Python, Java and R

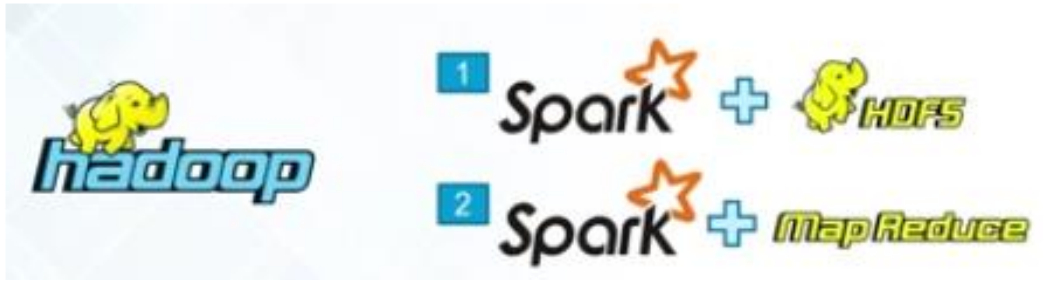
Spark Features

- Lazy Evaluation: Delays evaluation till needed

- Real time computation & low latency because of in-memory computation

Spark Features

- Hadoop Integration

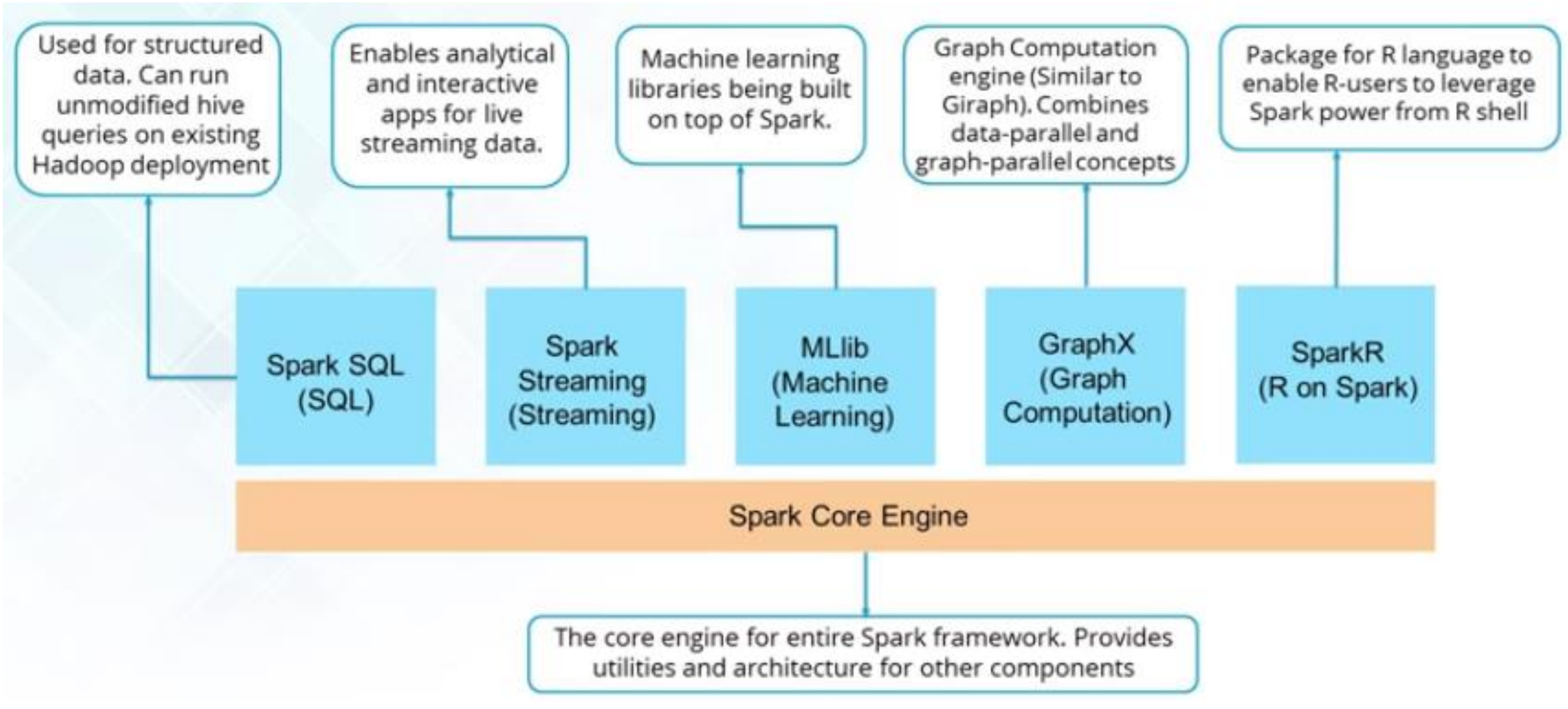


- Machine Learning for iterative tasks

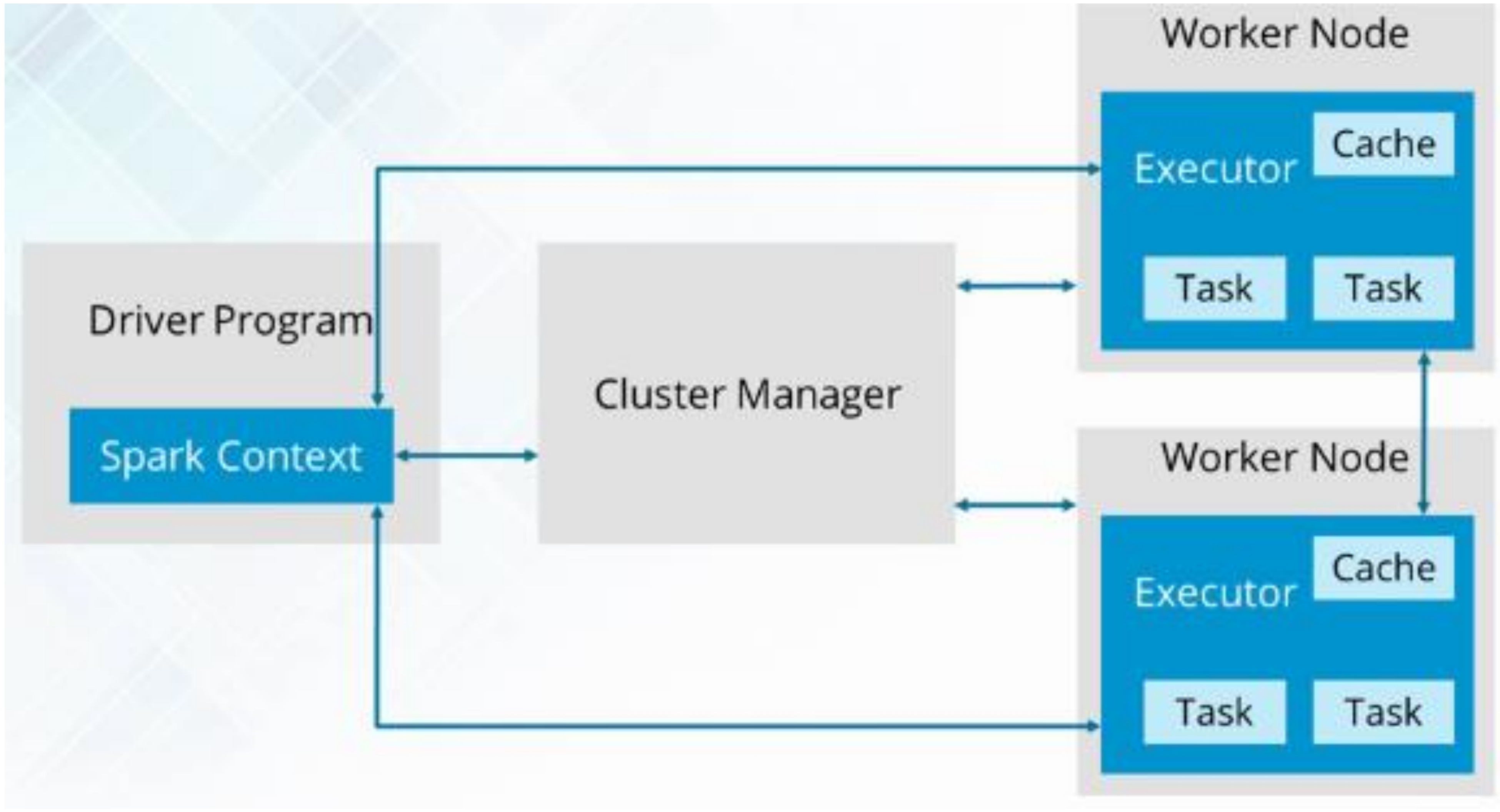
Spark Ecosystem

- Spark Core
- Spark SQL
- Spark Streaming
- MLlib
- GraphX
- Spark R

Spark Ecosystem


Used for structured data. Can run unmodified hive queries on existing Hadoop deployment
Enables analytical and interactive apps for live streaming data.
Machine learning libraries being built on top of Spark.
Graph Computation engine (Similar to Giraph). Combines data-parallel and graph-parallel concepts
Package for R language to enable R-users to leverage Spark power from R shell
Spark SQL (SQL)
Spark Streaming (Streaming)
MLlib (Machine Learning)
GraphX (Graph Computation)
SparkR (R on Spark)
Spark Core Engine
The core engine for entire Spark framework. Provides utilities and architecture for other components

Spark Architecture

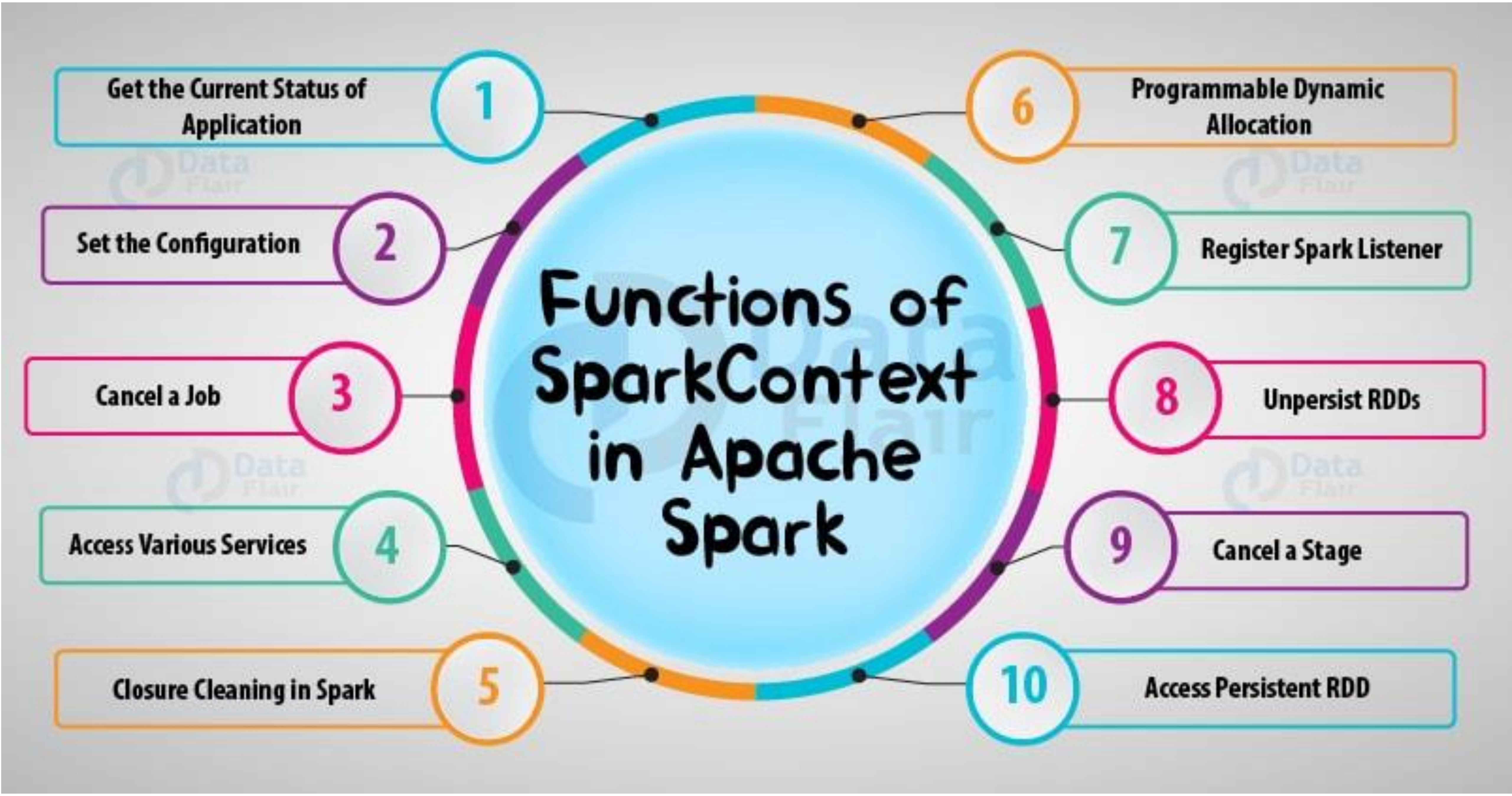


SparkContext

- **SparkContext** is the entry gate of Apache Spark functionality.
- The most important step of any Spark driver application is to generate SparkContext. It allows Spark Application to access Spark Cluster with the help of Resource Manager (**YARN/Mesos**).
- To create SparkContext, first **SparkConf** should be made.
- The SparkConf has a configuration parameter that our Spark driver application will pass to SparkContext.

SparkContext

- After the creation of a SparkContext object, we can invoke functions such as **textFile, sequenceFile, parallelize** etc.
- It can also be used to create RDDs, broadcast variable, and accumulator, ingress Spark service and run jobs.
- All these things can be carried out until SparkContext is stopped.
- Only one SparkContext may be active per JVM.
- It must be stopped before creating a new one as below:
  stop(): Unit
  It will display the following message:
  INFO SparkContext: Successfully stopped SparkContext


Functions of SparkContext in Apache Spark
1 Get the Current Status of Application
2 Set the Configuration
3 Cancel a Job
4 Access Various Services
5 Closure Cleaning in Spark
6 Programmable Dynamic Allocation
7 Register Spark Listener
8 Unpersist RDDs
9 Cancel a Stage
10 Access Persistent RDD

Spark Streaming

- Spark Streaming is used for processing real-time streaming data
- It is useful addition to the core Spark API
- Spark Streaming 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 to process the real-time data

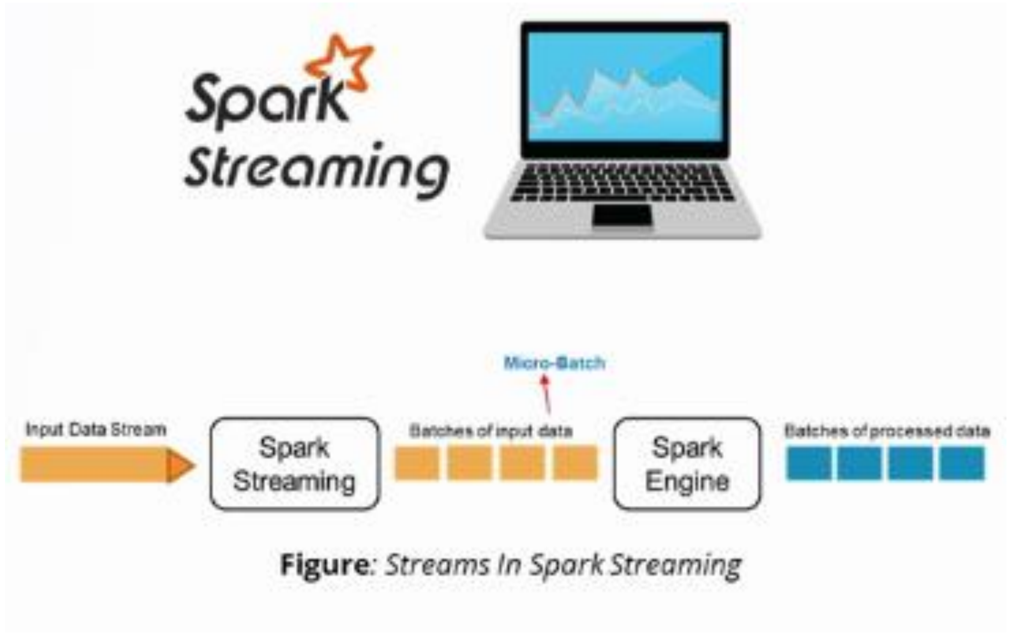


Figure: *Streams In Spark Streaming*

Spark Streaming

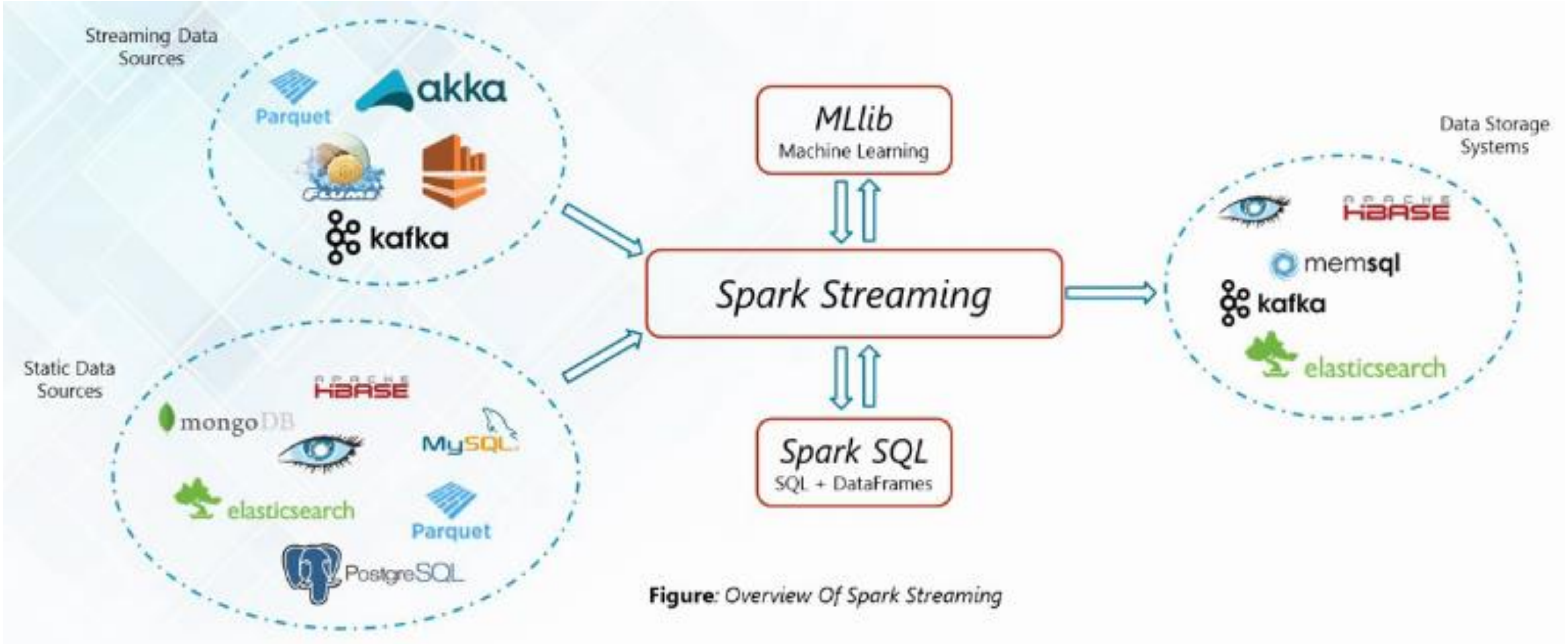


Figure: *Overview Of Spark Streaming*

Spark Streaming

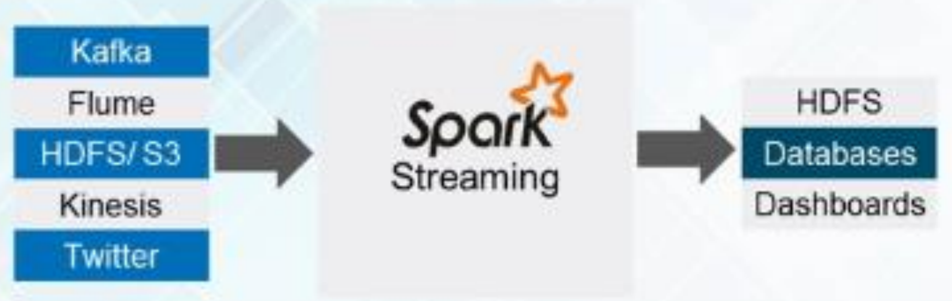


Figure: *Data from a variety of sources to various storage systems*

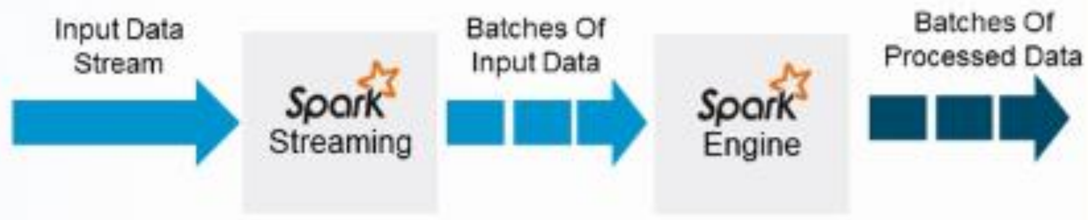


Figure: *Incoming streams of data divided into batches*

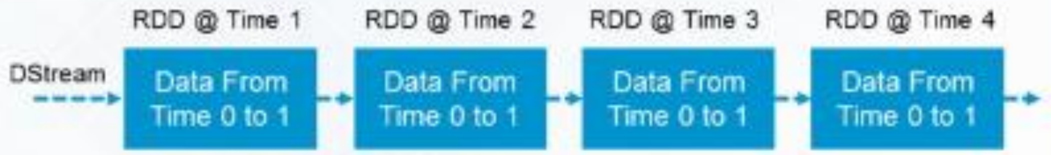


Figure: *Input data stream divided into discrete chunks of data*

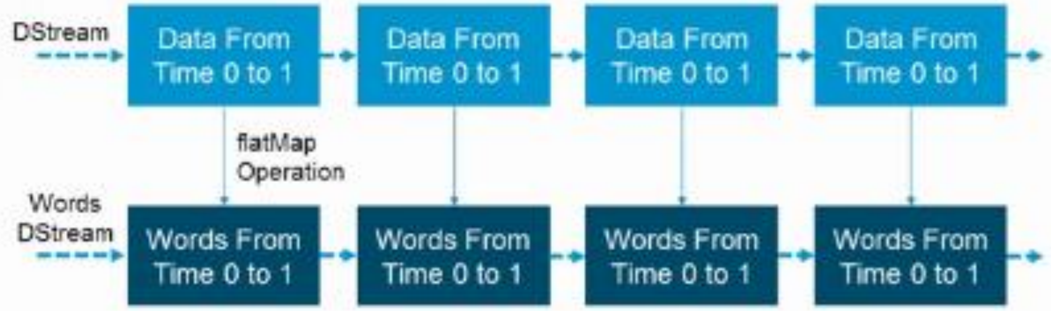


Figure: *Extracting words from an InputStream*

Spark SQL

Spark SQL Features

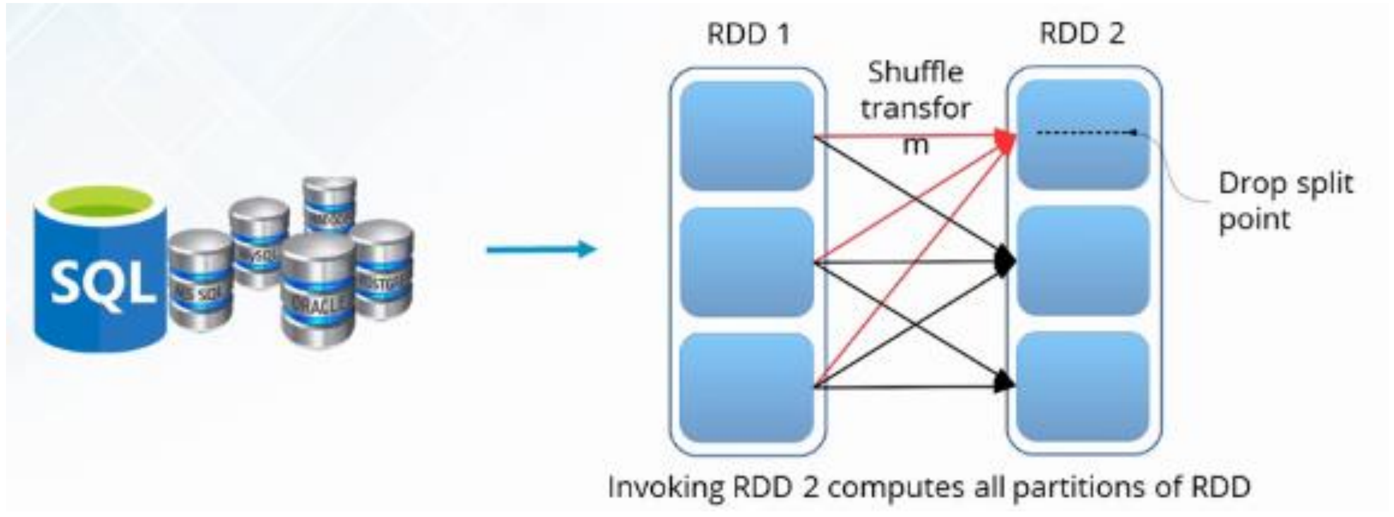
- Spark SQL integrates relational processing with Spark's fundamental programming

- Spark SQL is used for the structured/semi structured data analysis in Spark

Spark SQL Features

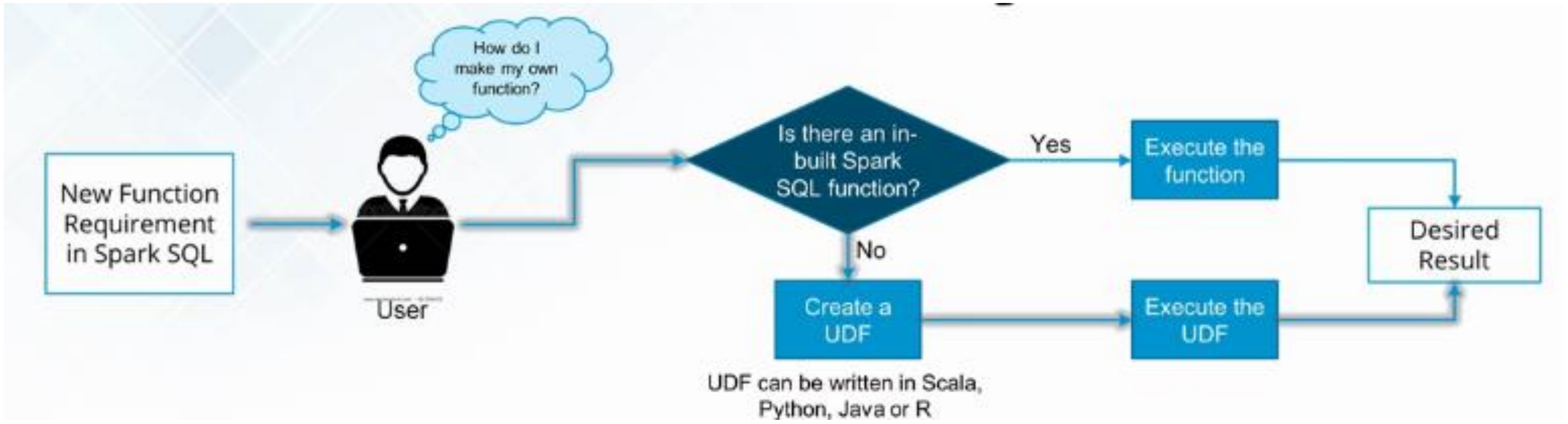
- Support for various data formats

- SQL queries can be converted into RDDs for transformations



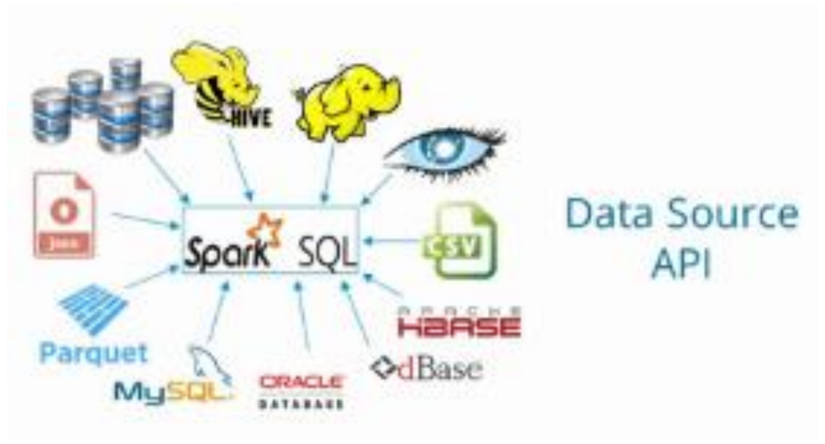
Spark SQL Features

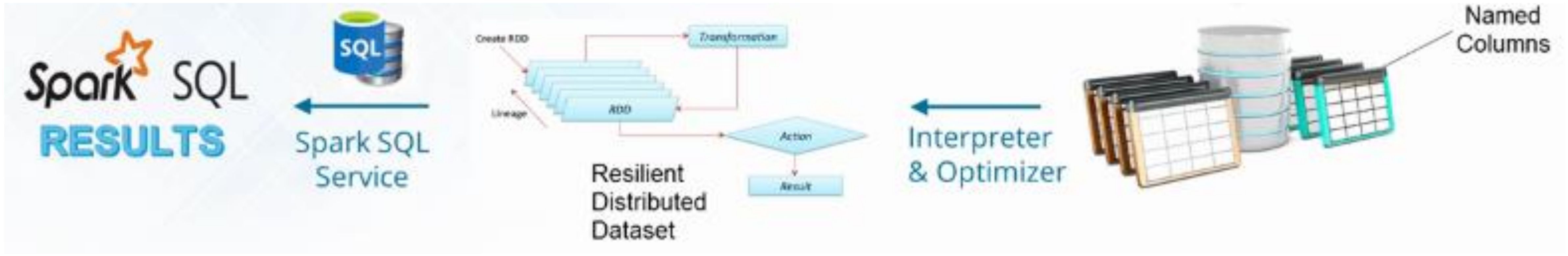
- Standard JDBC/ODBC Connectivity
- User Defined Functions lets user define new Column-based functions to extend the Spark vocabulary


How do I make my own function?
New Function Requirement in Spark SQL
User
Is there an in-built Spark SQL function?
Yes
No
Execute the function
Create a UDF
Execute the UDF
Desired Result
UDF can be written in Scala, Python, Java or R

Spark SQL Flow Diagram

- Spark SQL has the following libraries:
  - Data Source API
  - DataFrame API
  - Interpreter & Optimizer
  - SQL Service
- The flow diagram represents a Spark SQL process using all the four libraries in sequence

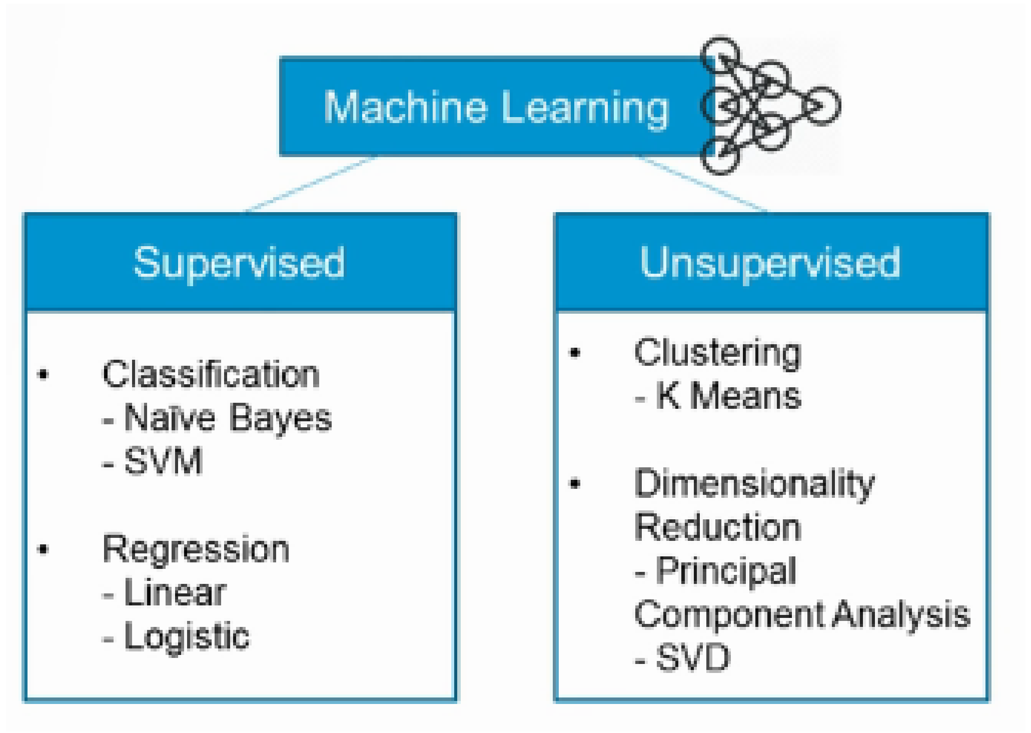




MLlib

- Machine learning can be broken down into two classes of algorithms:

1. Supervised algorithms use labeled data
2. Unsupervised algorithms make sense of the data without labels



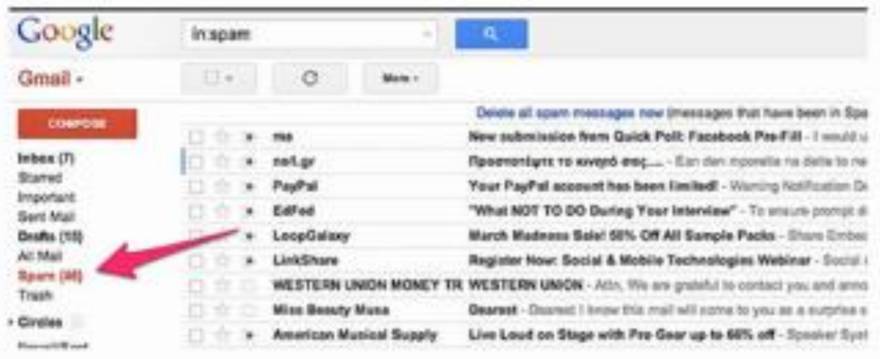
MLlib Techniques

- Three common categories of techniques:

1. Classification
2. Clustering
3. Collaborative Filtering

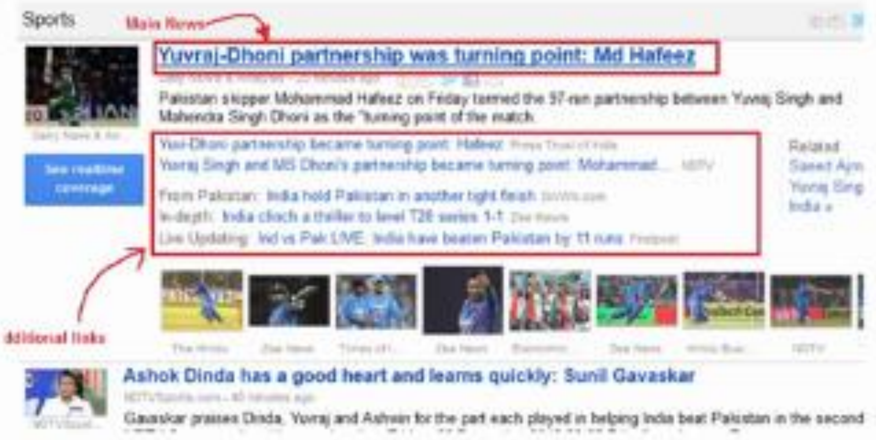
Mllib - Techniques

- Classification: It is family of supervised machine learning algorithms that designate input as belonging to one of several pre-defined classes
  - Some common use cases for classification include: Credit Card fraud detection, Email spam detection
- Clustering: In clustering, an algorithm groups objects into categories by analyzing similarities between input examples



Delete all spam messages now (messages that have been in Spa

me	New submission from Quick Poll: Facebook Pre-Fill - I would u
nafl.gr	Προσφοπήστε το κινητό σας.... - Εαν δεν μπορείτε να δείτε το με
PayPal	Your PayPal account has been limited! - Warning Notification D
EdFed	"What NOT TO DO During Your Interview" - To ensure prompt d
LoopGalaxy	March Madness Sale! 50% Off All Sample Packs - Share Embed
LinkShare	Register Now: Social & Mobile Technologies Webinar - Social
WESTERN UNION MONEY TR	WESTERN UNION - Attn, We are grateful to contact you and anno
Miss Beauty Musa	Dearest - Dearest I know this mail will come to you as a surprise a
American Musical Supply	Live Loud on Stage with Pro Gear up to 60% off - Speaker Syst



Yuvraj-Dhoni partnership was turning point: Md Hafeez

Pakistan skipper Mohammad Hafeez on Friday termed the 57-run partnership between Yuvraj Singh and Mahendra Singh Dhoni as the "turning point of the match.

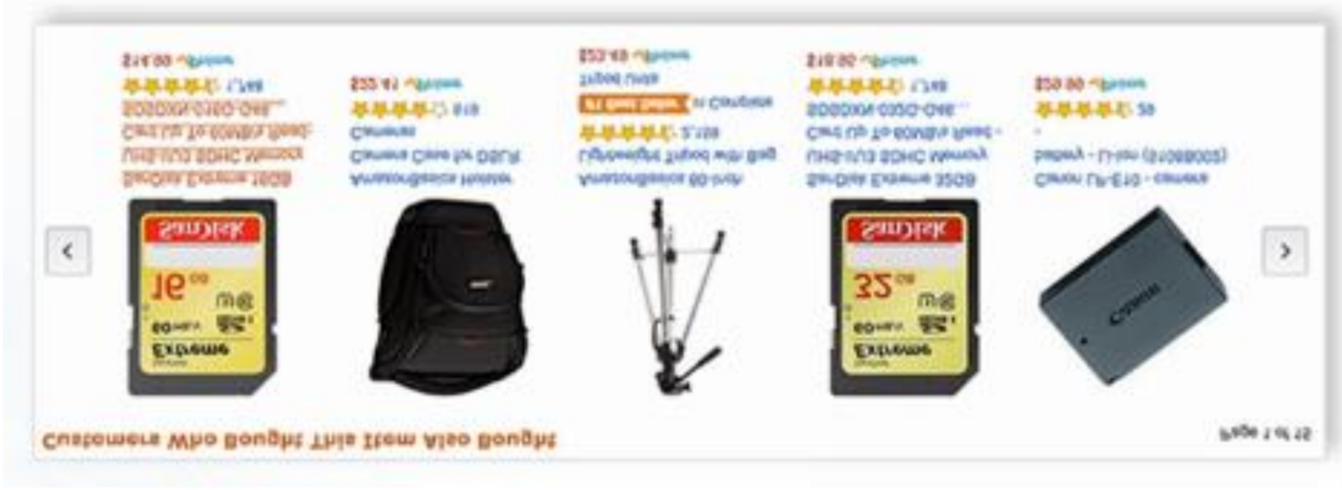
Yuvi-Dhoni partnership became turning point: Hafeez Press Trust of India
Yuvraj Singh and MS Dhoni's partnership became turning point: Mohammad... NDTV
From Pakistan: India hold Pakistan in another tight finish DAWN.com
In-depth: India clinch a thriller to level T20 series 1-1 Zee News
Live Updating: Ind vs Pak LIVE: India have beaten Pakistan by 11 runs Firstpost

Ashok Dinda has a good heart and learns quickly: Sunil Gavaskar

Gavaskar praises Dinda, Yuvraj and Ashwin for the part each played in helping India beat Pakistan in the second

Mllib - Techniques

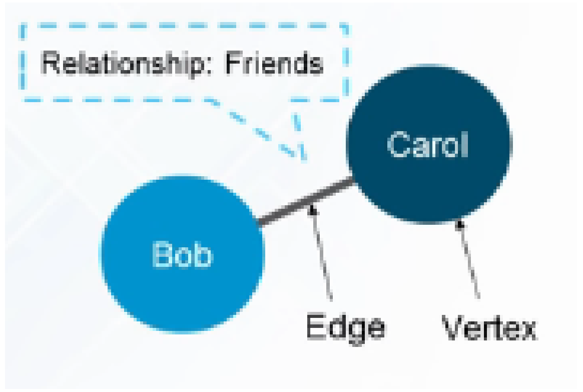
- Collaborative Filtering: These algorithms recommend items(filtering) based on preference information from many users(collaborative)



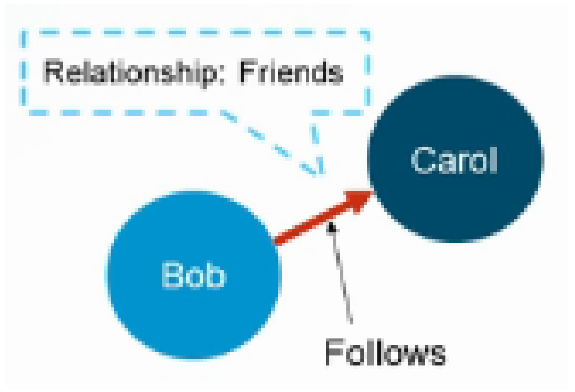
GraphX

GraphX

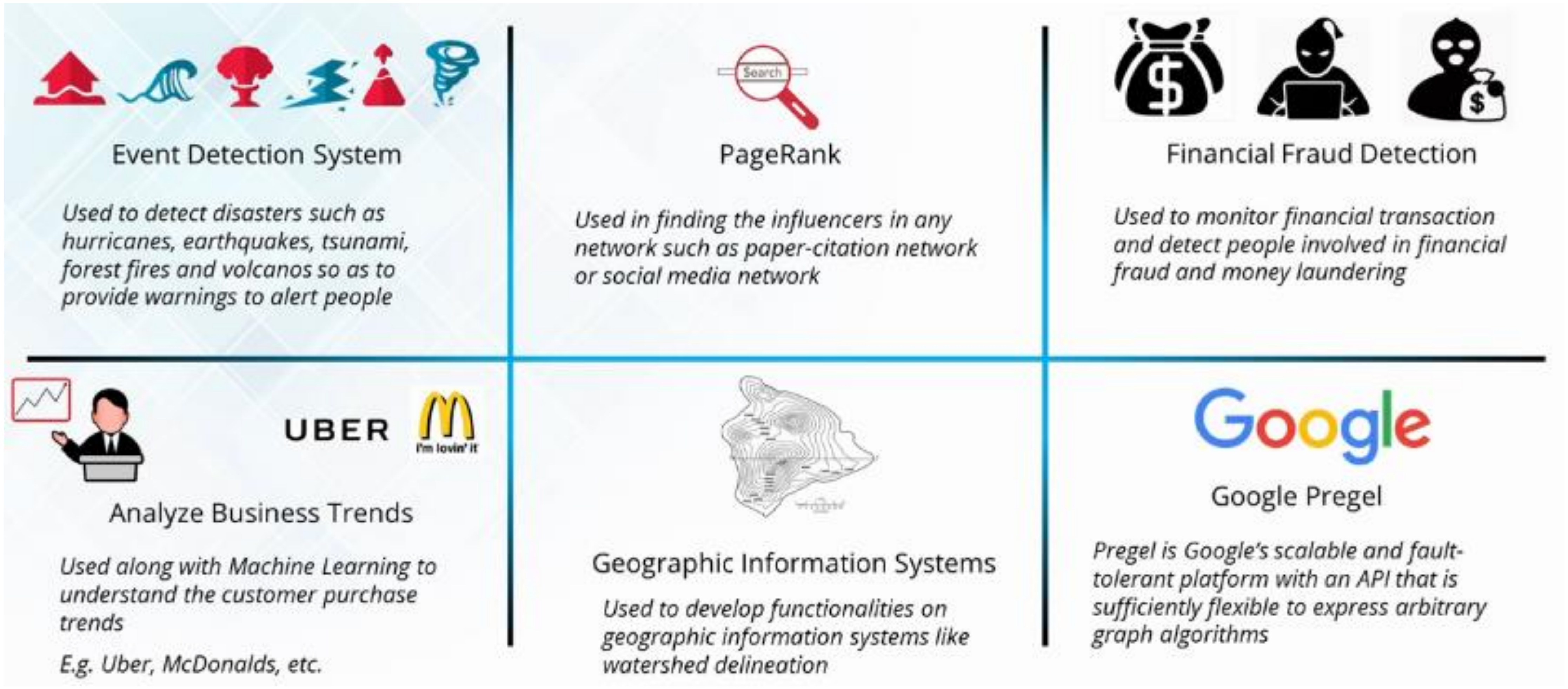
- A graph is a mathematical structure used to model relations between objects. A graph is made up of vertices and edges that connect them. The vertices are the objects and the edges are the relationships between them.



- A directed graph is a graph where the edges have a direction associated with them. E.g. User Bob follows Carol on Facebook.



GraphX Use Cases



Event Detection System

Used to detect disasters such as hurricanes, earthquakes, tsunami, forest fires and volcanos so as to provide warnings to alert people

PageRank

Used in finding the influencers in any network such as paper-citation network or social media network

Financial Fraud Detection

Used to monitor financial transaction and detect people involved in financial fraud and money laundering

Analyze Business Trends

Used along with Machine Learning to understand the customer purchase trends

E.g. Uber, McDonalds, etc.

Geographic Information Systems

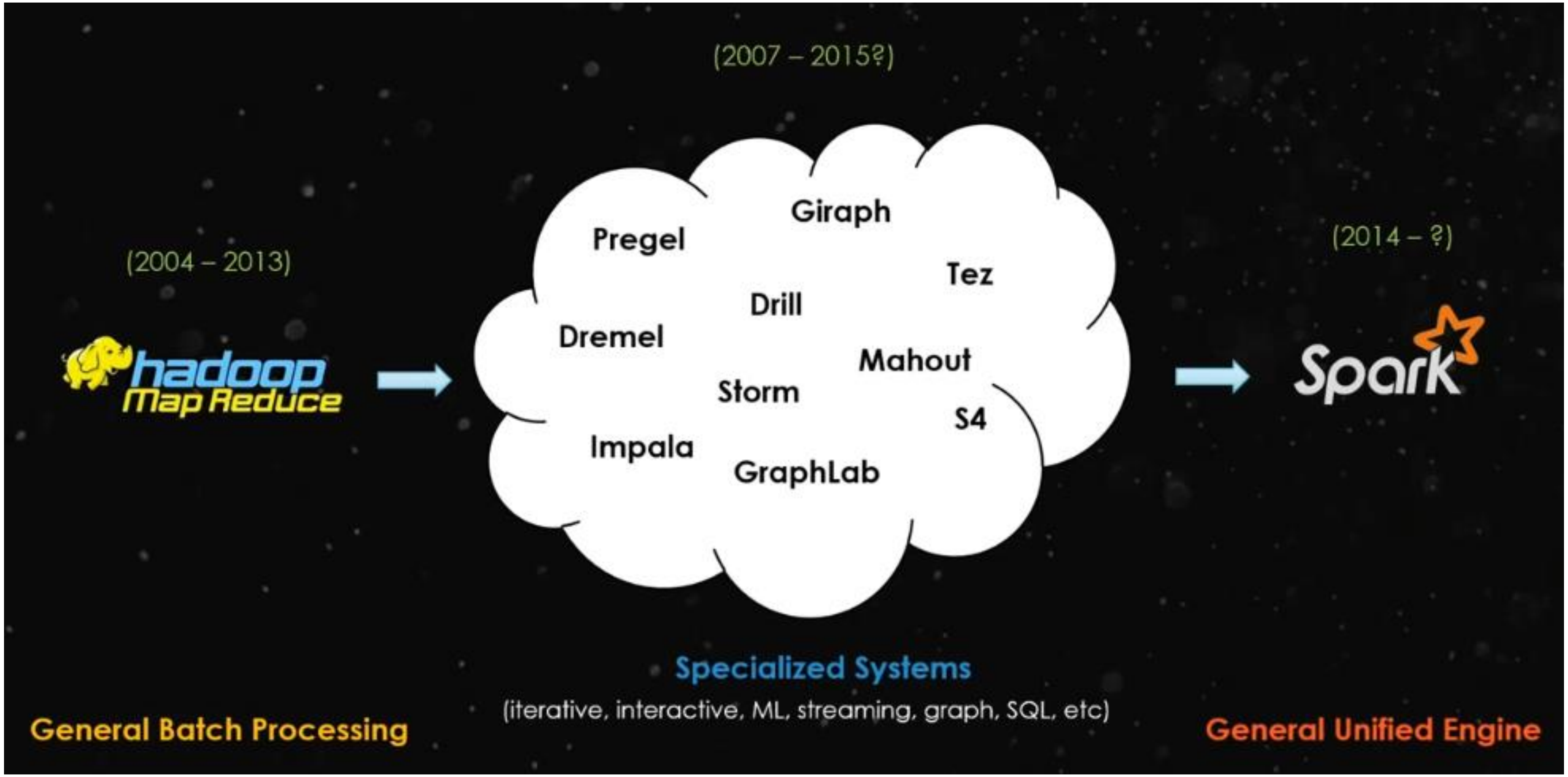
Used to develop functionalities on geographic information systems like watershed delineation

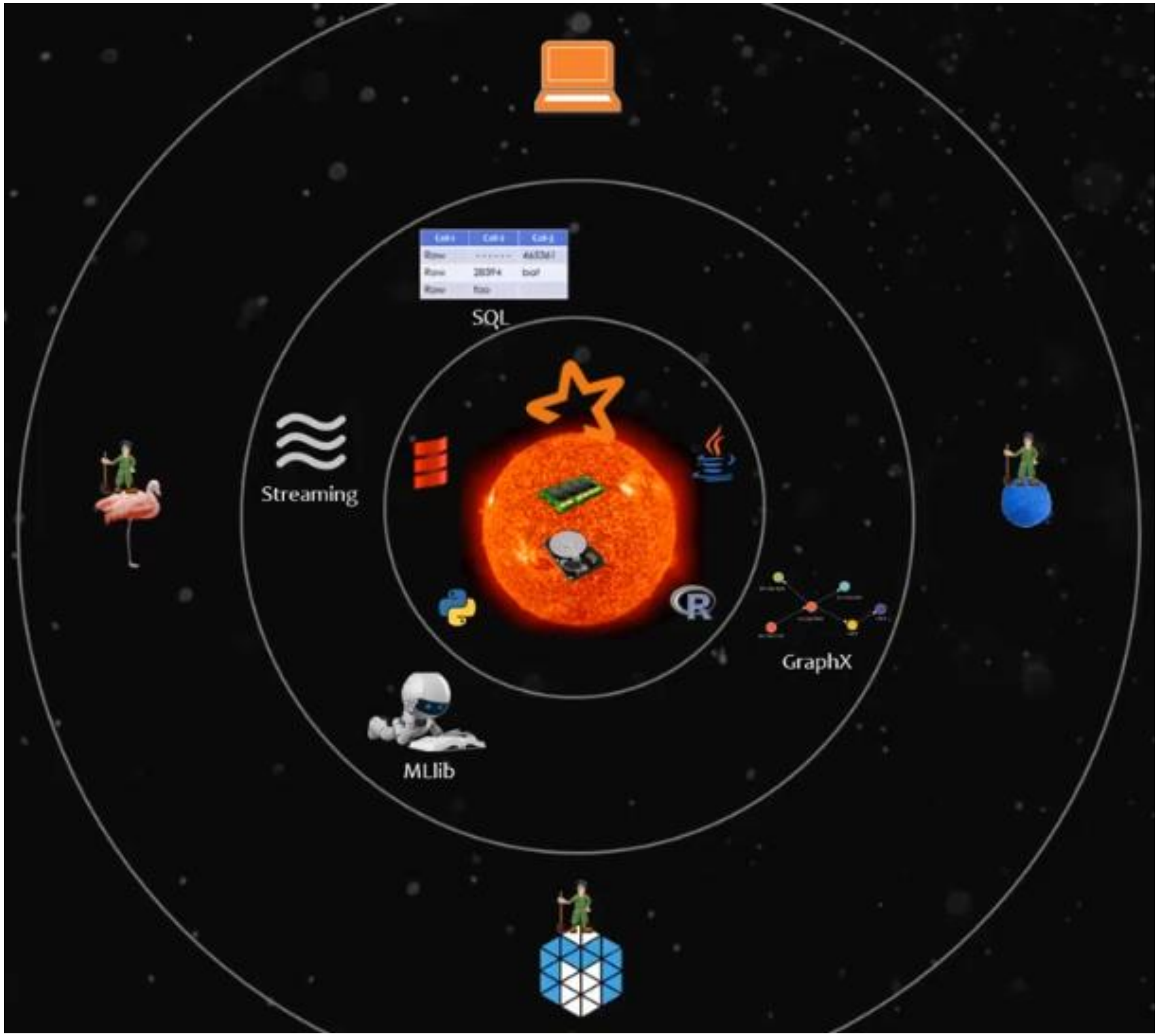
Google Pregel

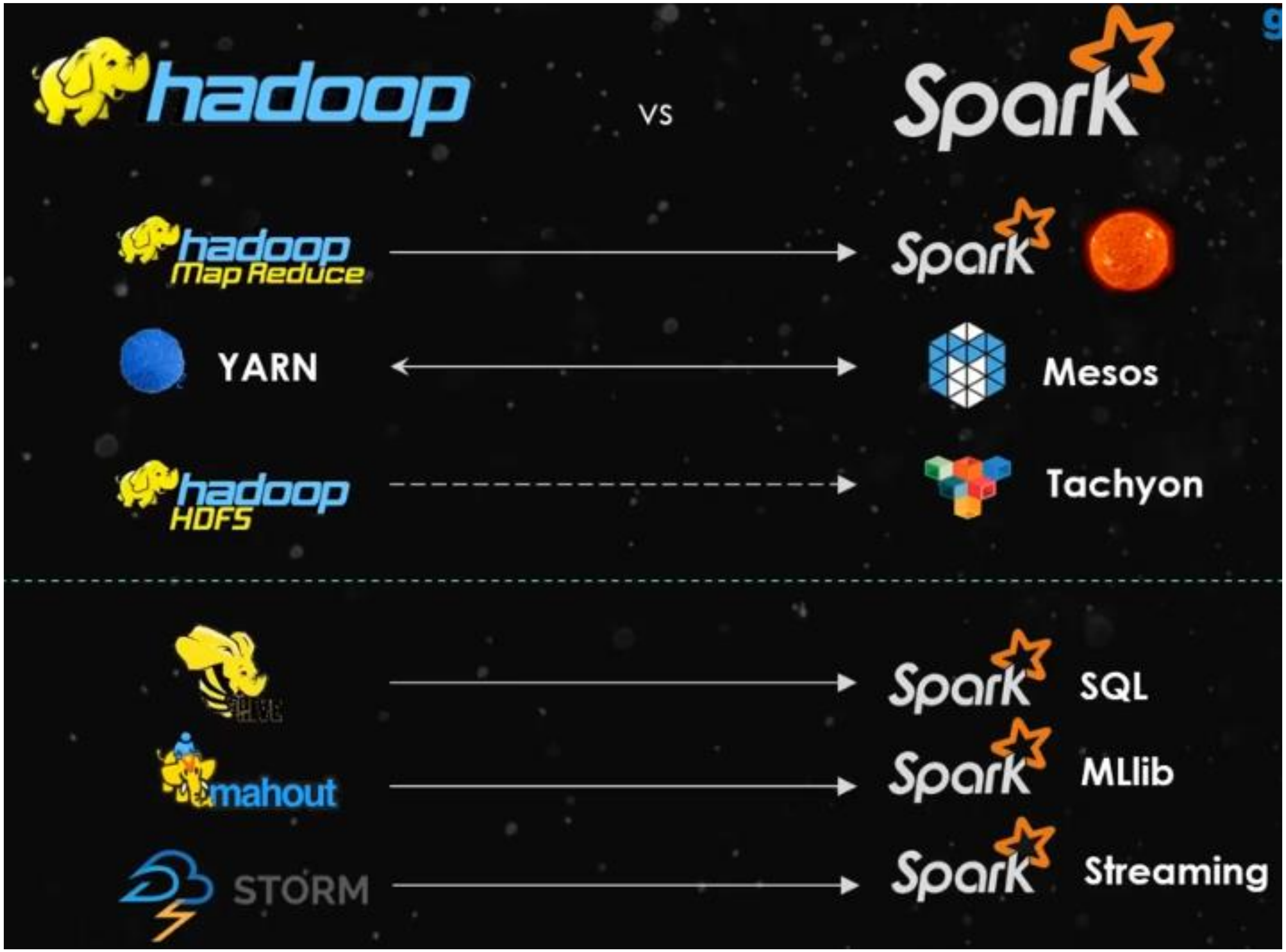
Pregel is Google's scalable and fault-tolerant platform with an API that is sufficiently flexible to express arbitrary graph algorithms

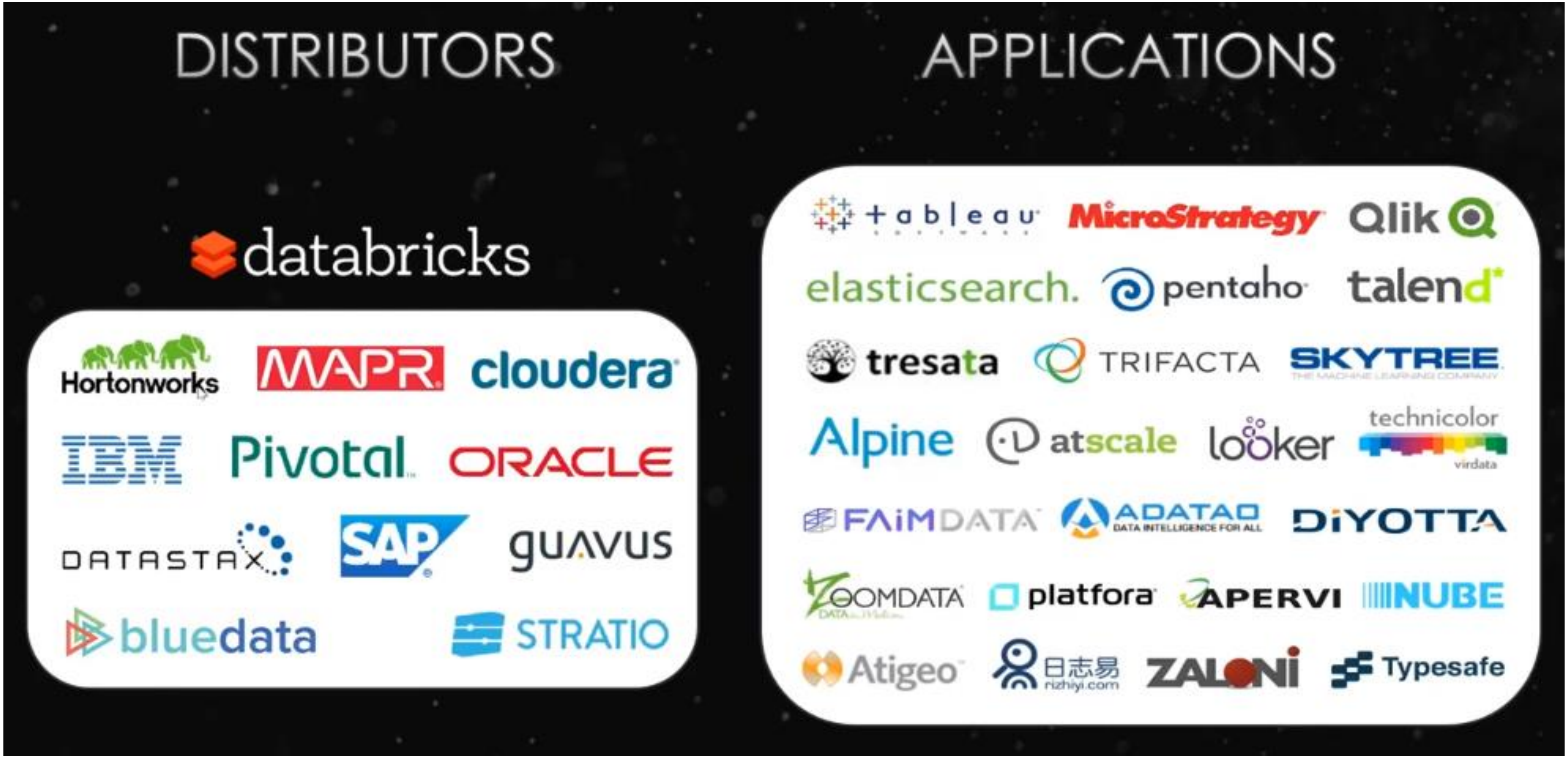
Use Case: Earthquake Detection using Spark

Why Spark became popular?




SQL
Streaming
MLlib
GraphX


hadoop
vs
Spark
hadoop Map Reduce
Spark
YARN
Mesos
hadoop HDFS
Tachyon
HIVE
Spark SQL
mahout
Spark MLlib
STORM
Spark Streaming



DISTRIBUTORS

databricks

Hortonworks
MAPR
cloudera
IBM
Pivotal
ORACLE
DATASTAX
SAP
guavus
bluedata
STRATIO

APPLICATIONS

tableau
MicroStrategy
Qlik
elasticsearch.
pentaho
talend
tresata
TRIFACTA
SKYTREE THE MACHINE LEARNING COMPANY
Alpine
atscale
looker
technicolor virdata
FAiMDATA
ADATAO DATA INTELLIGENCE FOR ALL
DiYOTTA
ZOOMDATA DATA in Motion
platfora
APERVI
NUBE
Atigeo
日志易 rizhiyi.com
ZALONI
Typesafe

Spark Processing in cluster

- Assume,
    - Cluster of five nodes 1,2,3,4,5.
    - 10 GB file stored on Hadoop. Divided in three parts and available at node 2,3,4 and 5.
    - Spark programs asks for 4 executors each of 5 GB memory and 2 cores
    - Driver starts at Node 1
    - Node 2,3,4,5 selected to run these executors
- Size of these executors has to be specified in spark program

Size of Executor

- We cannot ask for one executor of 100 GB RAM and 100 cores!
- Also, asking for one executor of 20 GB RAM and 8 cores is not good. It is similar to executing in single local machine.
- Appropriate size of executors should be demanded in code. Generally, this decision is taken after discussing with admin. E.g. 4 executors of 5 GB RAM and 2 cores each.
- This would increase parallelism.

MapReduce

- Mappers are always launched on nodes where data is available
- No. of mappers = no. of blocks

Spark

- When executors are launched, there is no guarantee that they will be launched on same machines where data is available as YARN does not know anything about data locality.
- Executors might be launched on same machines or different machines, so initially it might take some time to fetch data in memory, but even then it would be faster than MapReduce.
- Number of executors and its size has to be decided while writing spark program.

MapReduce Block

- One or more blocks will be processed by mappers.
- Size of each block is 128 MB.
- Blocks are stored on disk.

Spark Partition

- One or more partitions will be processed by executors.
- Each block becomes one partition in spark.
- While using other data storage, data has to be divided into partitions first. Normally, spark tries to set the number of partitions automatically based on cluster.
- More partitions results in more parallelism.
- Partitions are stored in RAM

Executor Memory Utilization

- If 10 GB executor is launched, we cannot use full capacity for data storage.
- 10 % of memory is allocated to system calls. E.g. from 10 GB capacity, 1 GB is used for system calls.
- From remaining 90% of memory, we can utilize only 60% of memory.
- Remaining is used by garbage collector, JVM management and all.
- So, we can utilize only 54% of full capacity for data storage. e.g., from 10 GB storage, we can utilize only 5.8 GB.
- Each executor needs at least one processor core. So, on dual core, only two executors can be launched.

SparkDynamicMemoryUtilization

- If set to true, it will go for dynamic memory utilization. It means, if 8 executors are launched, after the work is finished, it will automatically kill idle executors.
- If set to false, it will not go for dynamic memory utilization. It means, if 8 executors are launched, after the work is finished, it will still keep them as it is and will not kill idle executors.

- Executors can be killed automatically after completing their jobs, but driver has to be stopped. Otherwise even after the job is completed, driver will still be there and eat resources.(e.g. default 1 core and all)

RDD Fundamentals

- Spark revolves around the concept of a *resilient distributed dataset* (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel.

- There are two ways to create RDDs:
  - *parallelizing* an existing collection in your driver program, or
  - referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.
- RDDs are immutable. If any operation is performed on RDDs, it will create a new RDD.

Parallelized collections

- Parallelized collections are created by calling SparkContext’s parallelize method on an existing collection in your driver program.
- The elements of the collection are copied to form a distributed dataset that can be operated on in parallel.
- E.g.

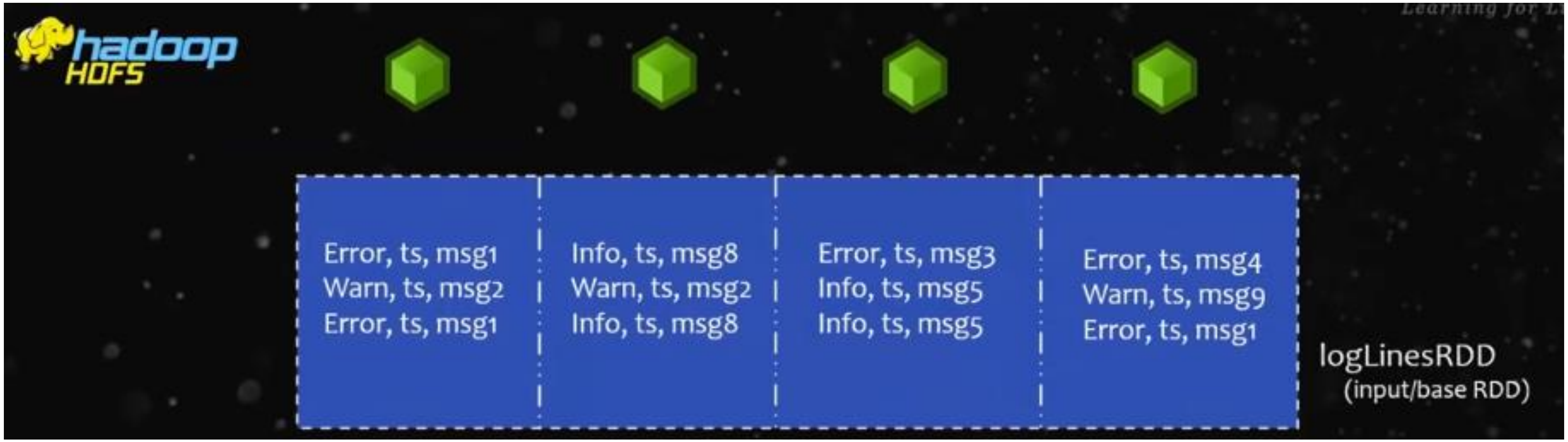
```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

External Datasets

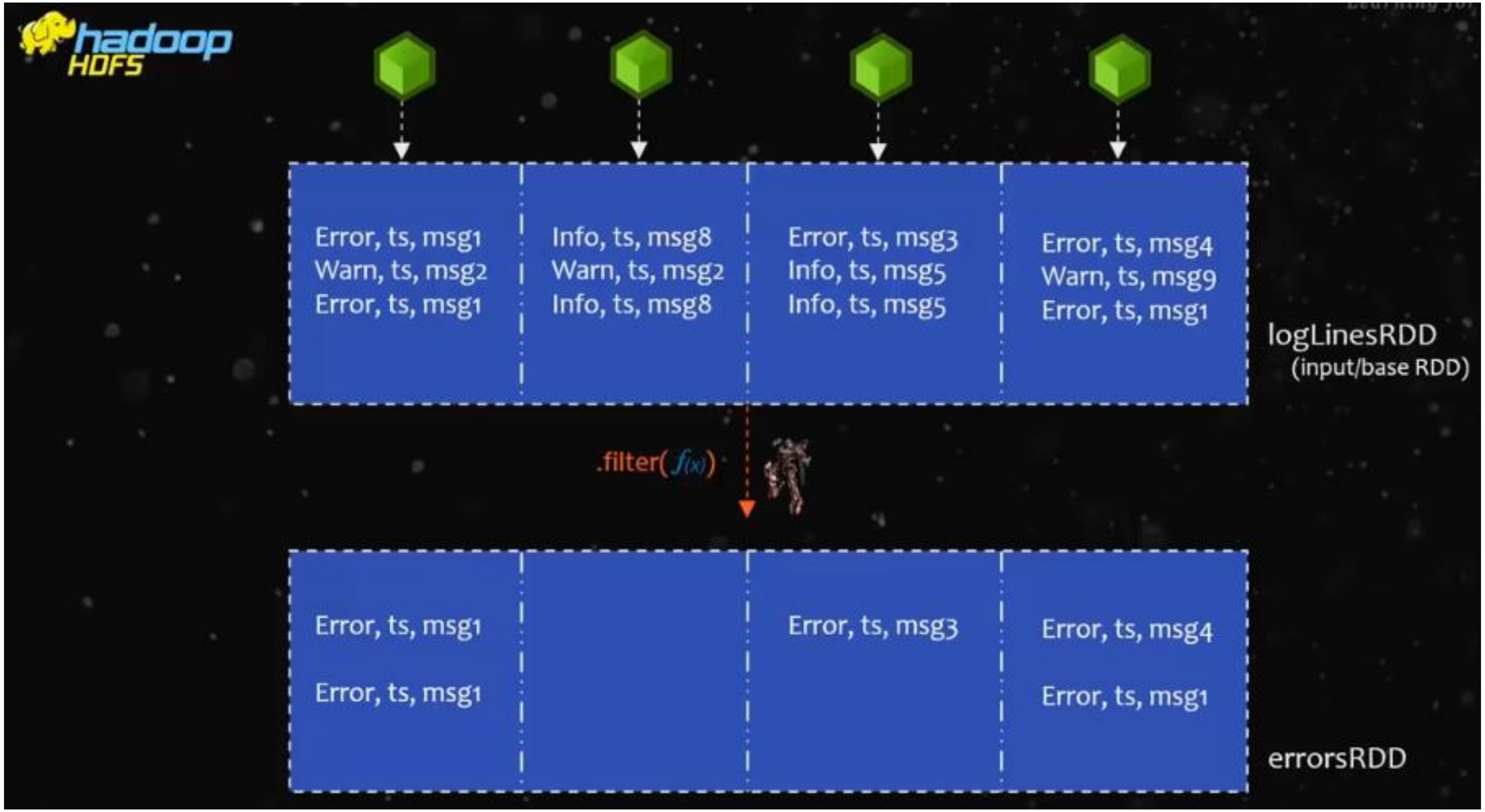
- PySpark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc.
- Spark supports text files, SequenceFiles, and any other Hadoop InputFormat.
- Text file RDDs can be created using SparkContext’s textFile method.
- This method takes a URI for the file (either a local path on the machine, or a hdfs://, s3a://, etc URI) and reads it as a collection of lines. Here is an example invocation:

```
distFile = sc.textFile("data.txt")
```

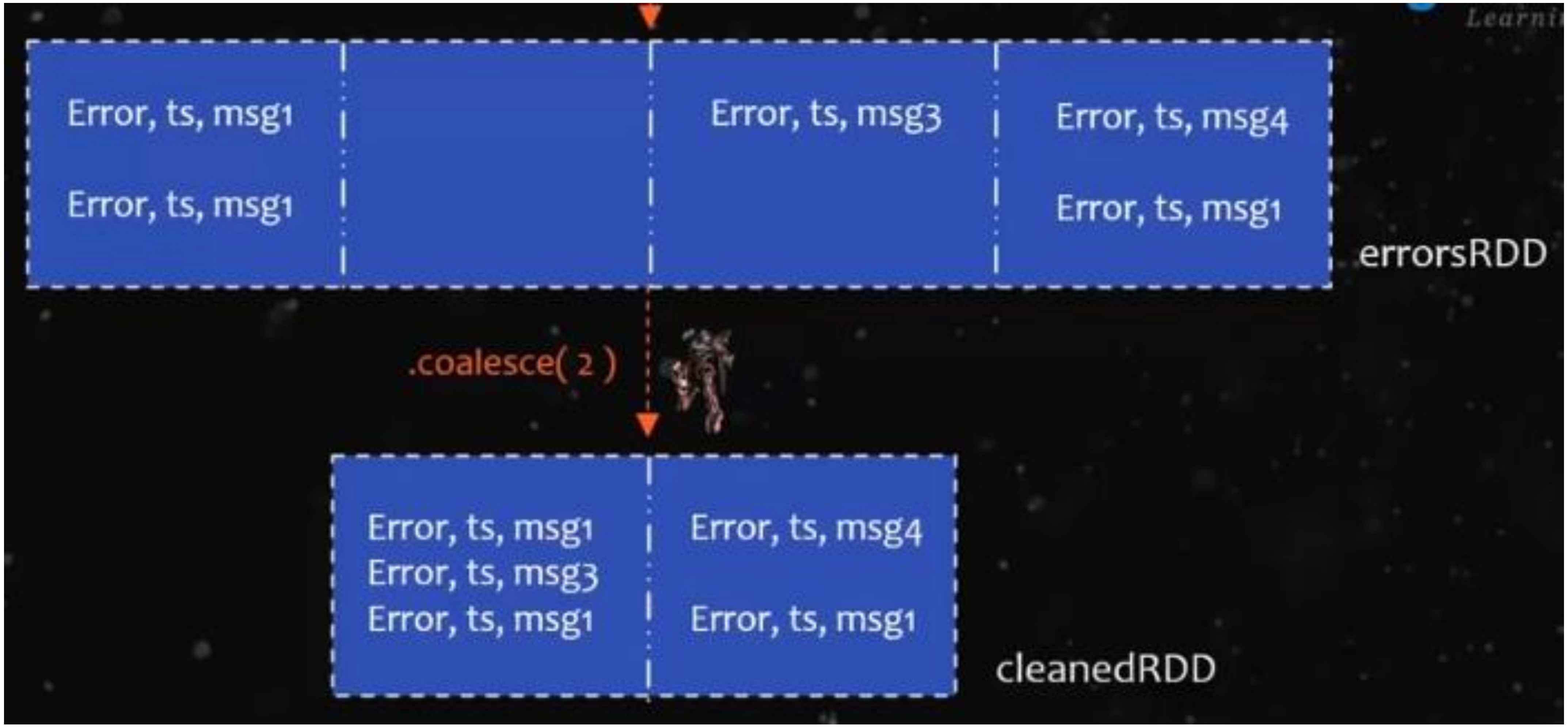
Example - Four executors



Example - Four executors



Example - Four executors

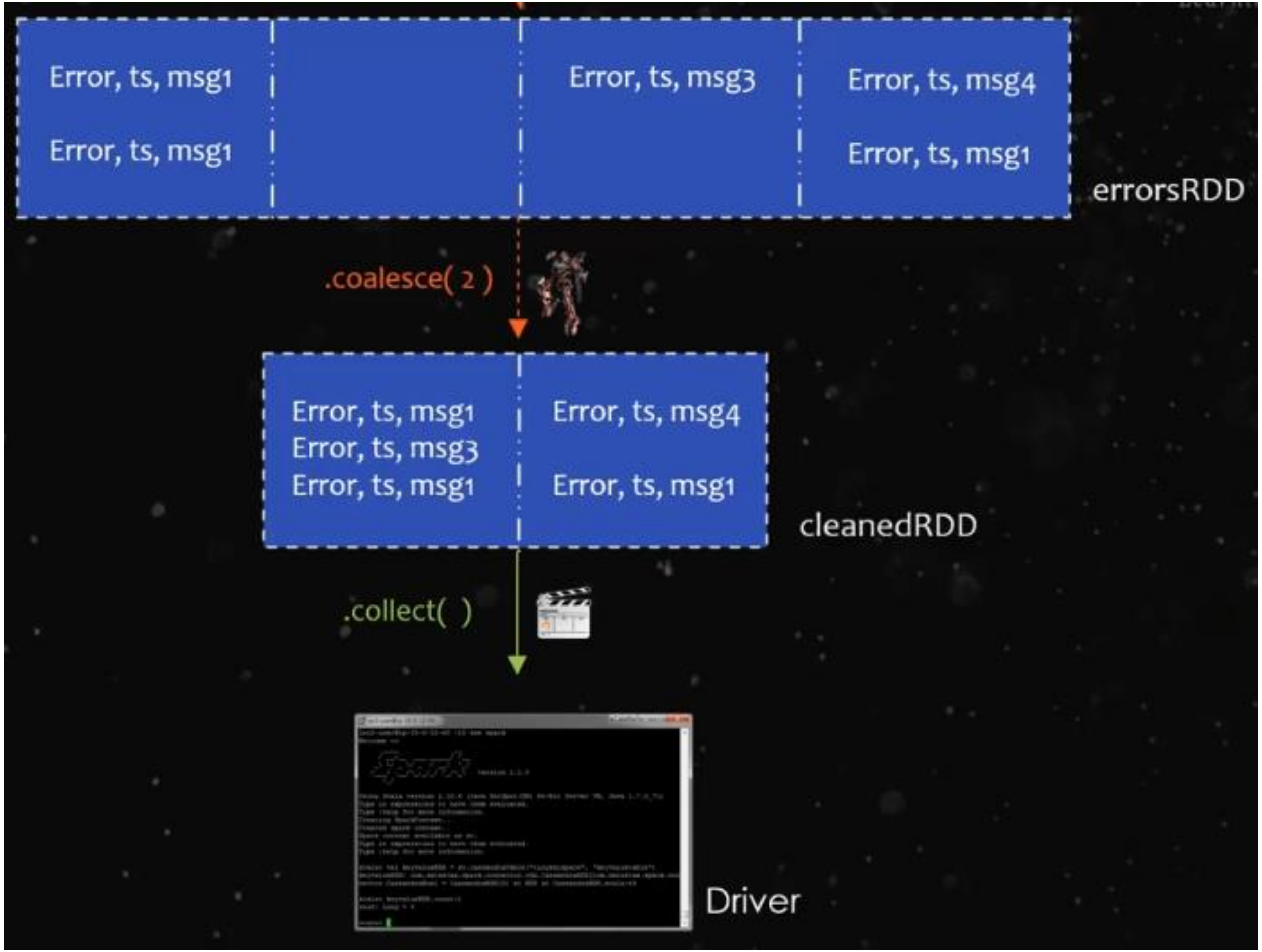


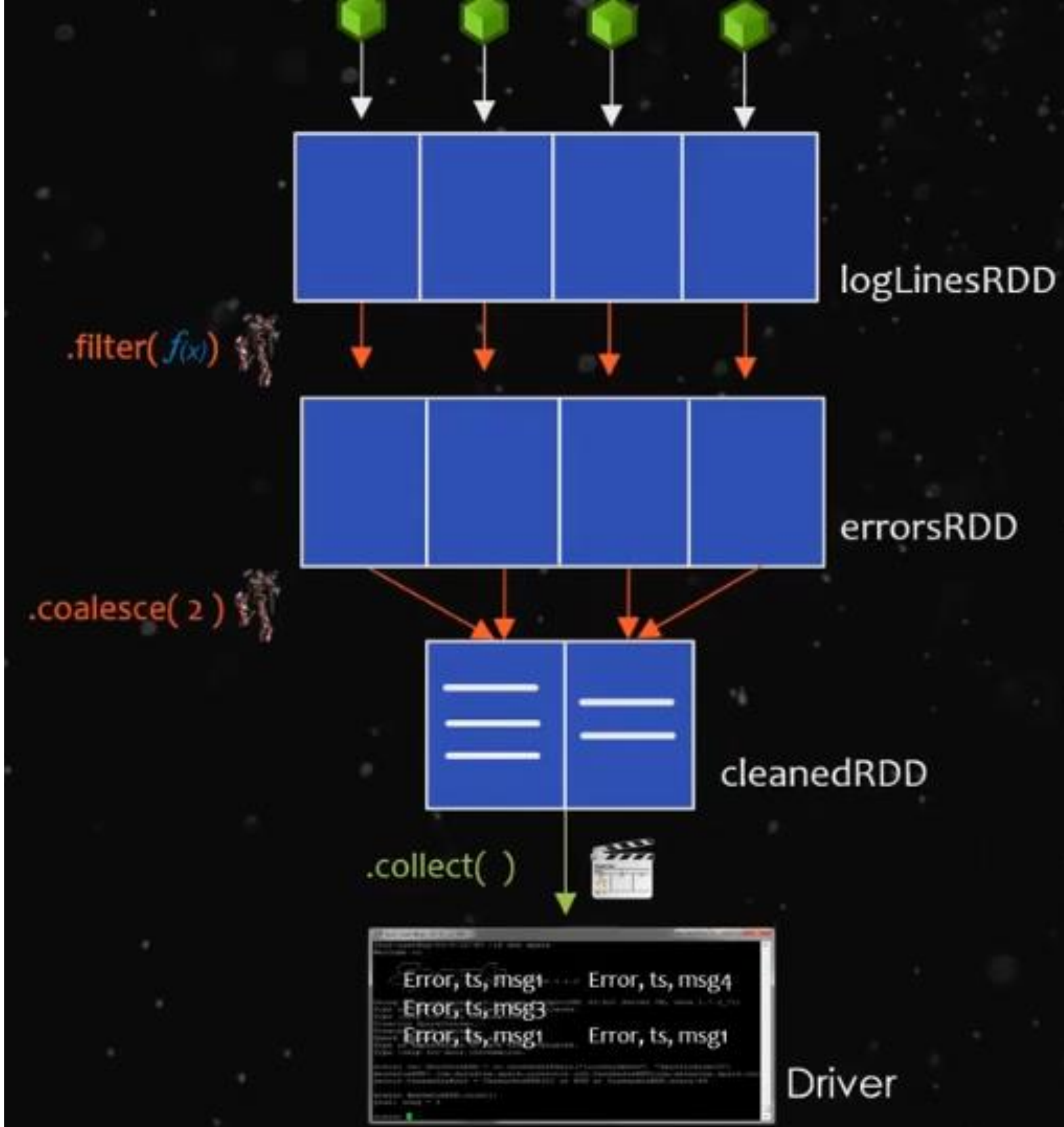
Difference between coalesce and repartition

- Repartition shuffles full data to reduce partitions, whereas coalesce is more intelligent and does less data movement.
- Repartition can also be used to increase number of partitions.
- Coalesce cannot be used to increase number of partitions.

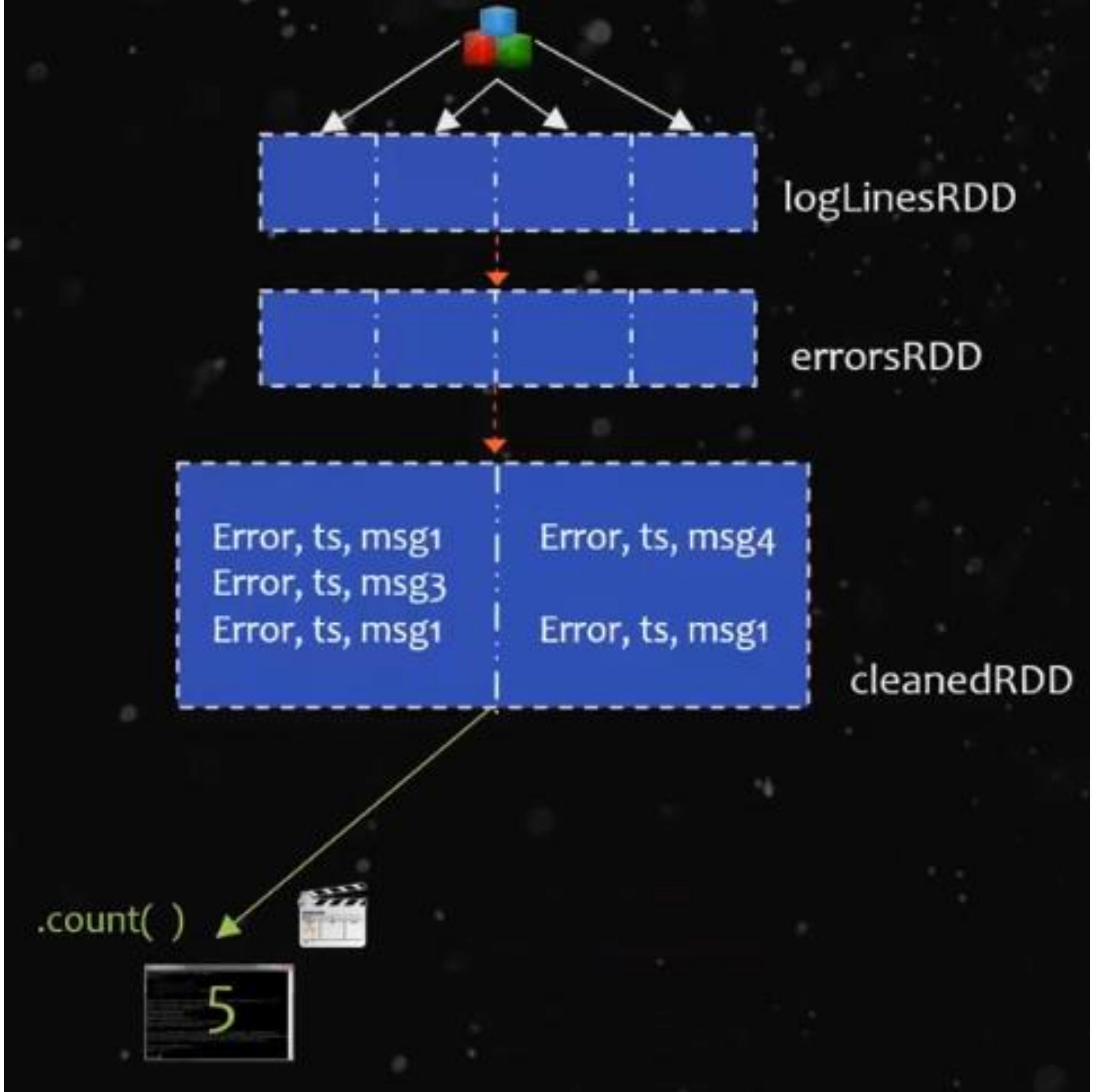
RDD Fundamentals

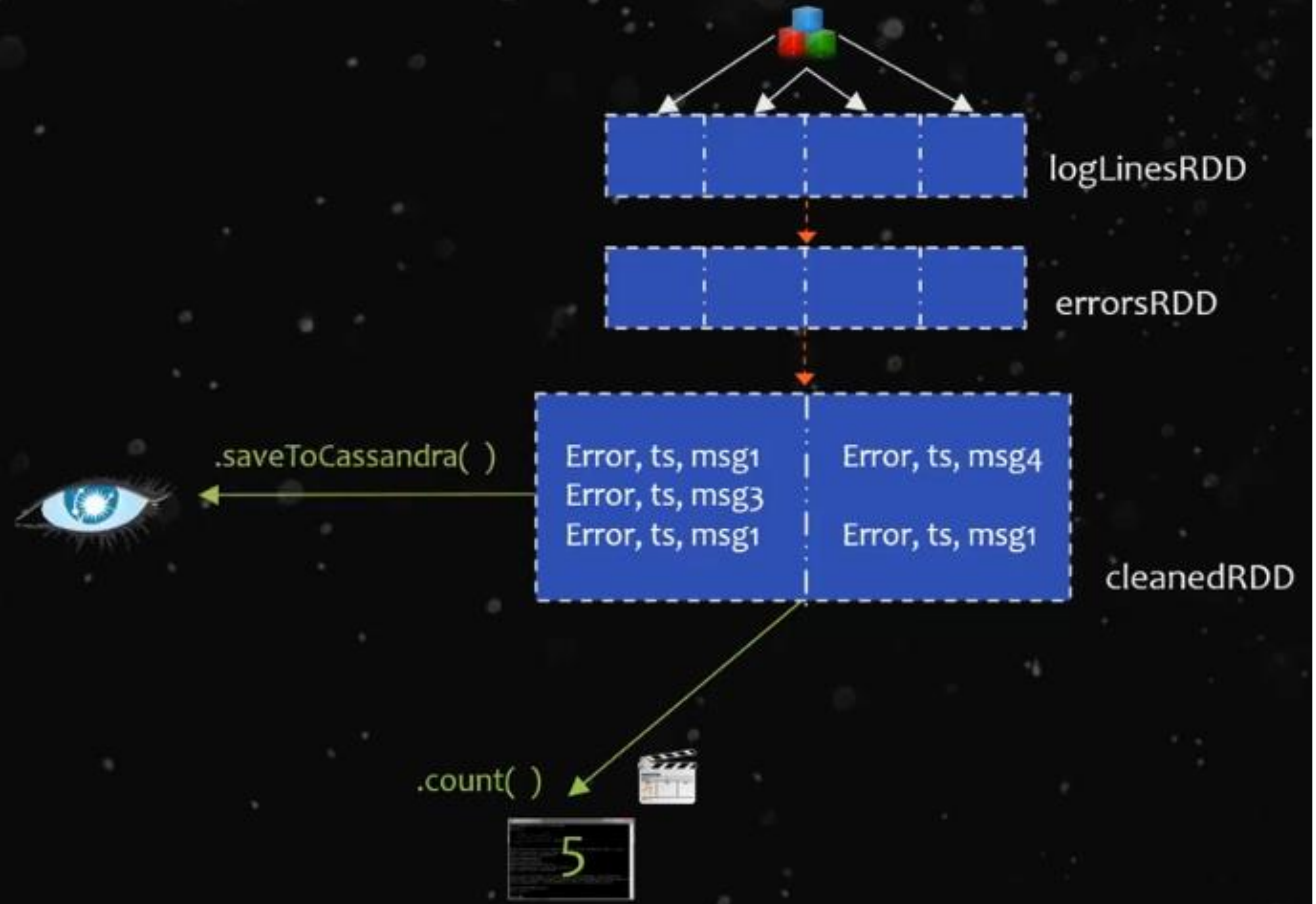
- Nothing will happen till now due to lazy evaluation.
- The reason is till now all are transformations and no action.
- To see the desired result, actions should be used.
- To decide which RDD operations should be applied, first sampled data can be used and checked.
- In previous example, it can be seen that after applying filter, data reduced by almost 50%. So we can call coalesce() to reduce number of partitions.

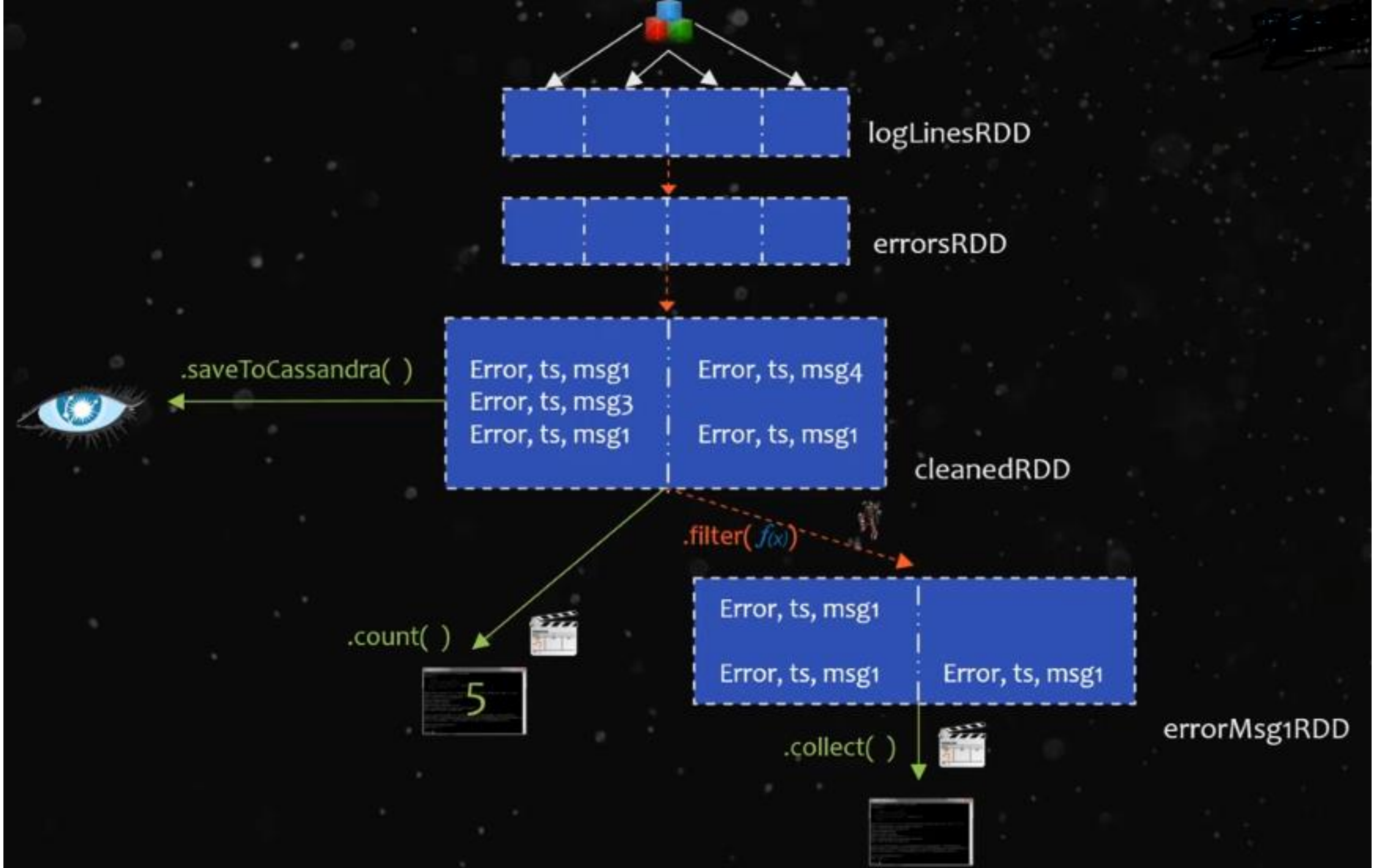

Error, ts, msg1
Error, ts, msg1
Error, ts, msg3
Error, ts, msg4
Error, ts, msg1
errorsRDD
.coalesce(2)
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.collect( )
Driver

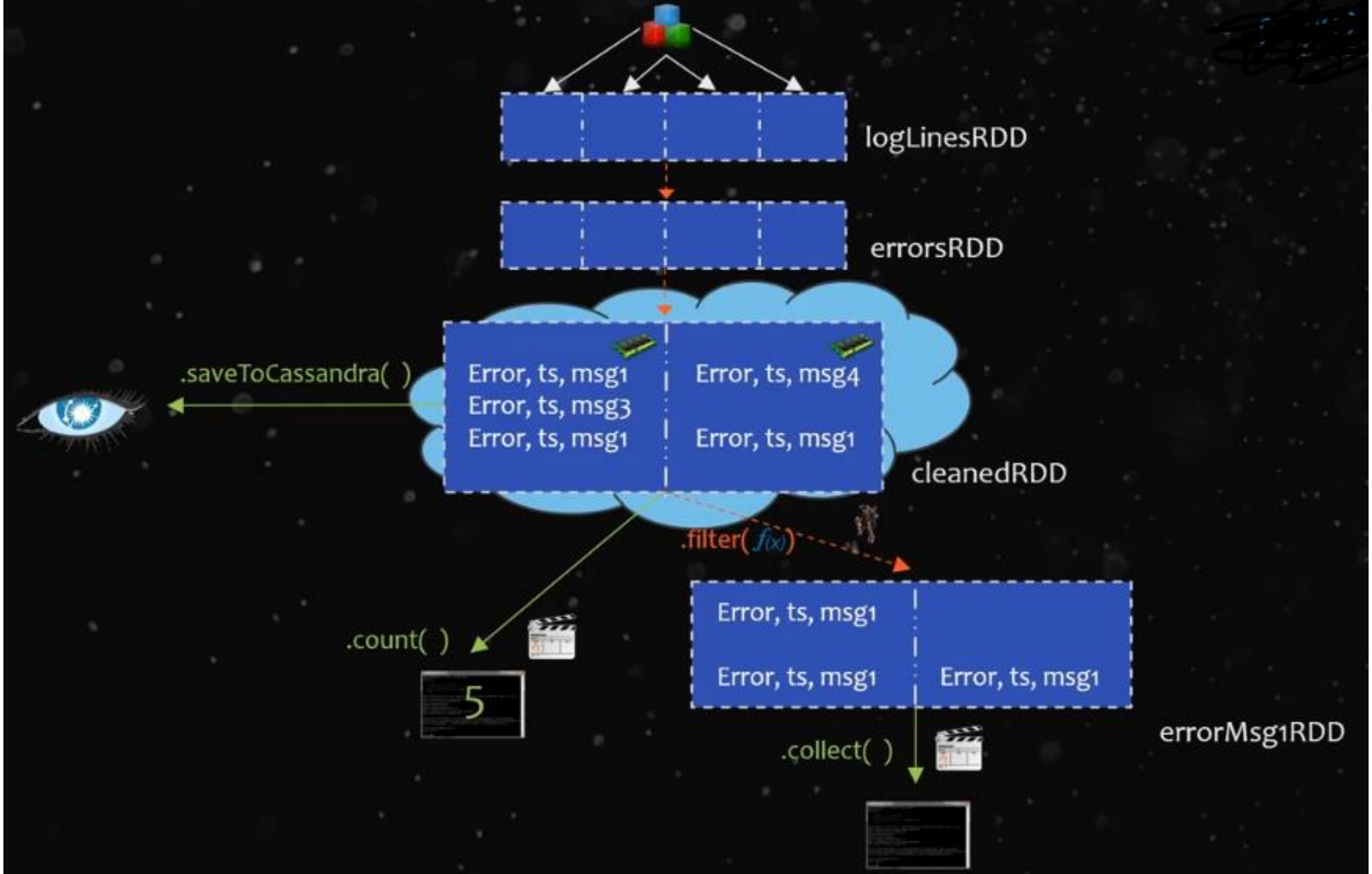


```
Error, ts, msg1      Error, ts, msg4
Error, ts, msg3
Error, ts, msg1      Error, ts, msg1
```


logLinesRDD
errorsRDD
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.count( )
5


logLinesRDD
errorsRDD
cleanedRDD
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
.saveToCassandra( )
.count( )
5


logLinesRDD
errorsRDD
.saveToCassandra( )
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.filter(f(x))
.count( )
5
Error, ts, msg1
Error, ts, msg1
Error, ts, msg1
errorMsg1RDD
.collect( )


logLinesRDD
errorsRDD
Error, ts, msg1
Error, ts, msg3
Error, ts, msg1
Error, ts, msg4
Error, ts, msg1
cleanedRDD
.saveToCassandra( )
.count( )
5
.filter(f(x))
Error, ts, msg1
Error, ts, msg1
Error, ts, msg1
errorMsg1RDD
.collect( )

RDD Operations

- RDDs support two types of operations:
  - **transformations**, which create a new dataset from an existing one, and
  - actions, which return a value to the driver program after running a computation on the dataset.
- For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results.
- On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset).

RDD Operations

- All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file).
- The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently.
- By default, each transformed RDD may be recomputed each time you run an action on it.
- However, you may also persist an RDD in memory using the persist (or cache) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it.
- There is also support for persisting RDDs on disk, or replicated across multiple nodes.
- Caching is useful when calling multiple actions, otherwise it is not very much useful.

Types of transformations

- Narrow Transformation
    - E.g. map,filter

- Wide Transformation
    - E.g. join, groupbykey

Narrow Transformations

- These type of transformations can be executed independently on different partitions.
- E.g. map(lambda x : (x,1)). This transformation can be executed independently as it has no dependency.
- Better performance

Wide Transformations

- These type of transformations cannot be executed independently on different partitions.
- E.g. groupbykey. This transformation can be executed only after processing the data of all partitions as it has to group data by same key.
- Costly operation compared to narrow transformation
- Advise: In spark code, more narrow transformations should be used and less wide transformations should be used.

Job in Spark

- A parallel computation consisting of multiple tasks that gets spawned in response to a **Spark** action is called a job.
- E.g. a.map then filter then groupbykey. This all together is single job if next operation is some action like count, collect etc.
- The jobs are divided into **stages** depending on how they can be separately carried out (mainly on shuffle boundaries).
- Then, these stages are divided into tasks. Tasks are the smallest unit of work that has to be done the executor.

Stages in Spark

- *A step in a physical execution plan*
- It is a physical unit of the execution plan.
- It is a set of parallel tasks i.e. one task per partition. In other words, each job which gets divided into smaller sets of tasks is a stage.
- E.g. map.filter.groupbykey, if this is the sequence of operations in spark, map and filter can execute independently on different partitions, but to execute groupbykey, all executors must finish execution up to filter. Only when all executors finish execution up to filter, groupbykey can be executed. So map and filter forms stage 0 and groupbykey forms stage 1.

References

- https://spark.apache.org
- https://www.youtube.com/watch?v=9mELEARcxJo
- https://www.edureka.co/community/41392/what-are-the-spark-job-and-spark-task-and-spark-staging
- https://stackoverflow.com/questions/28973112/what-is-spark-job
- https://data-flair.training/blogs/learn-apache-spark-sparkcontext/