BITS Pilani
Pilani Campus

BITS Pilani presentation

Dr. Vivek V. Jog
Dept. of Computer Engineering

innovate achieve lead

BITS Pilani
Pilani Campus

Big Data Systems (S1-24_CCZG522)
Lecture No.15

Apache Spark ..Why?

What is Apache Spark ?

Apache Spark is a Unified Processing Engine and Set of Libraries for Parallel Database Processing on Computer Cluster

Unified

Spark is designed to support wide variety of tasks over the same computing engine

Ex: Data Scientist as well as Data Engineers both can use same platform for their analysis, transformation and modelling.

Engineers: Data Analysis

Scientists: Modelling and Prediction

Computing Engine

Spark is purely computing engine. It does not store any data

Spark can Connect with different data sources
Ex: HDFS, JDBC/ODBC, AZURE..etc

Spark works with almost all data storage systems.

Libraries

Spark had ready to use libraries

Spark SQL

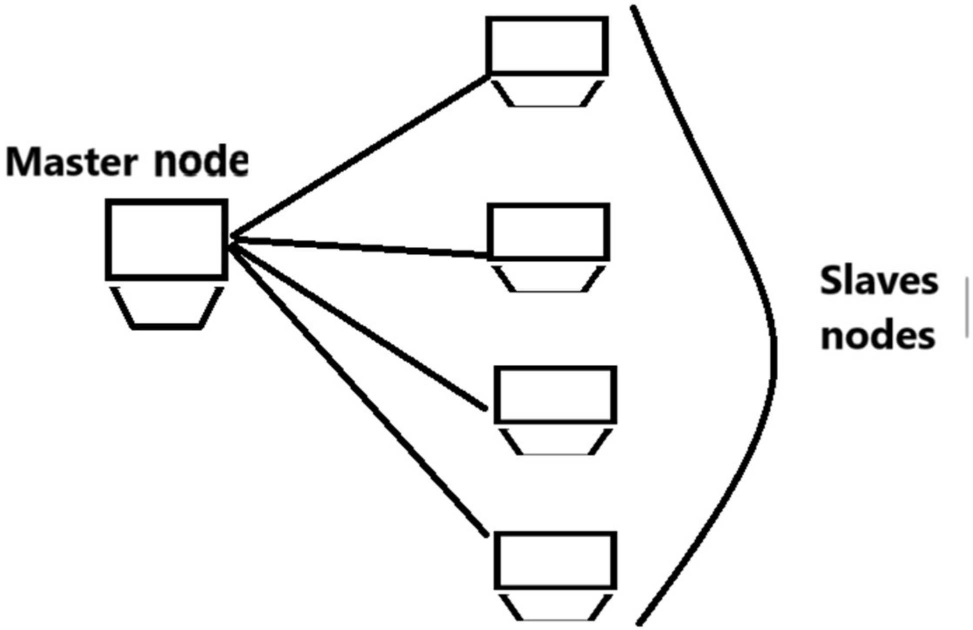
Spark Streams

ML Lib

..etc

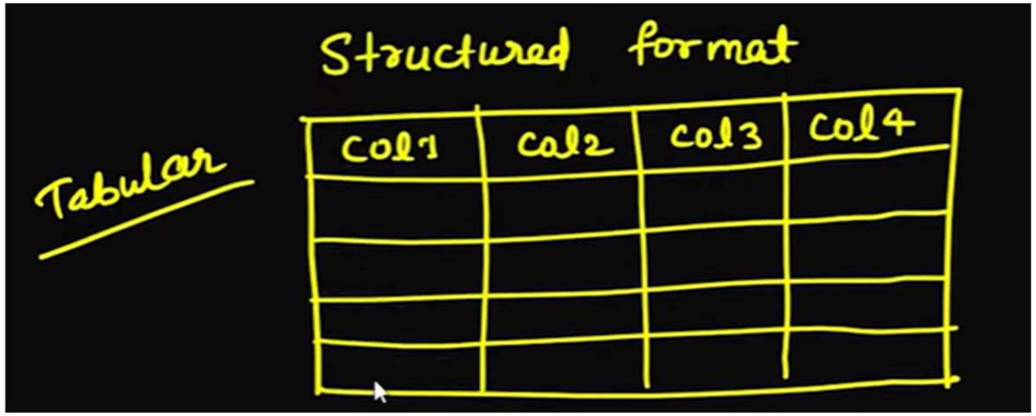
Computer Cluster for Parallel Processing

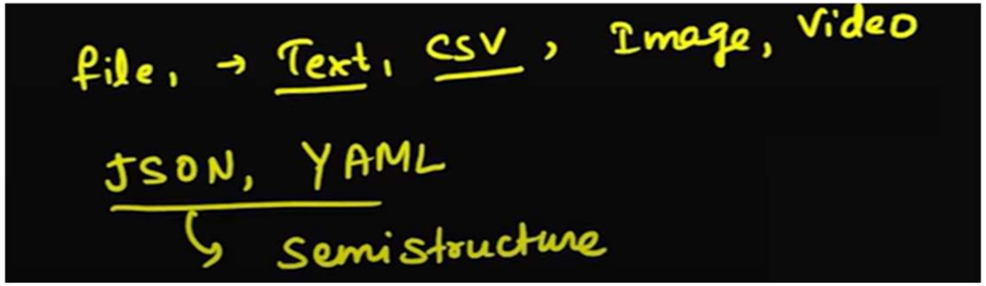
Spark works on computer cluster



Why Spark?


Database → Oracle, Teradata, exadata,
MySQL
Structured format


Structured format
Tabular
col1
col2
col3
col4


file, → Text, CSV, Image, Video
JSON, YAML
Semistructure

Issues

3 V's of Big Data

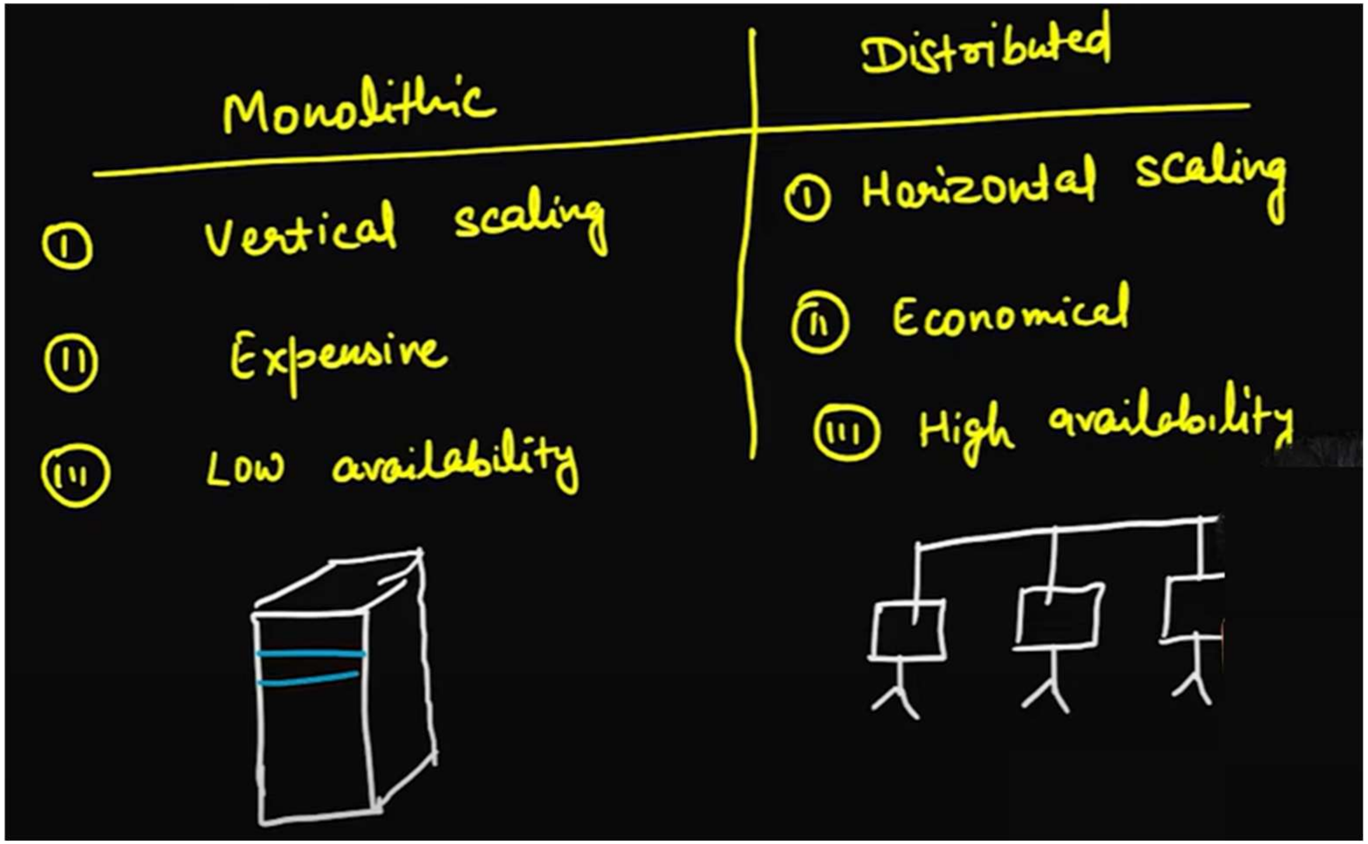
1. Velocity → 1sec , 1hour
2. Variety → Structured, Semistructured, Unstructured
3. Volume → 5GB , 10GB, 10TB

ETL → Extract Transform load

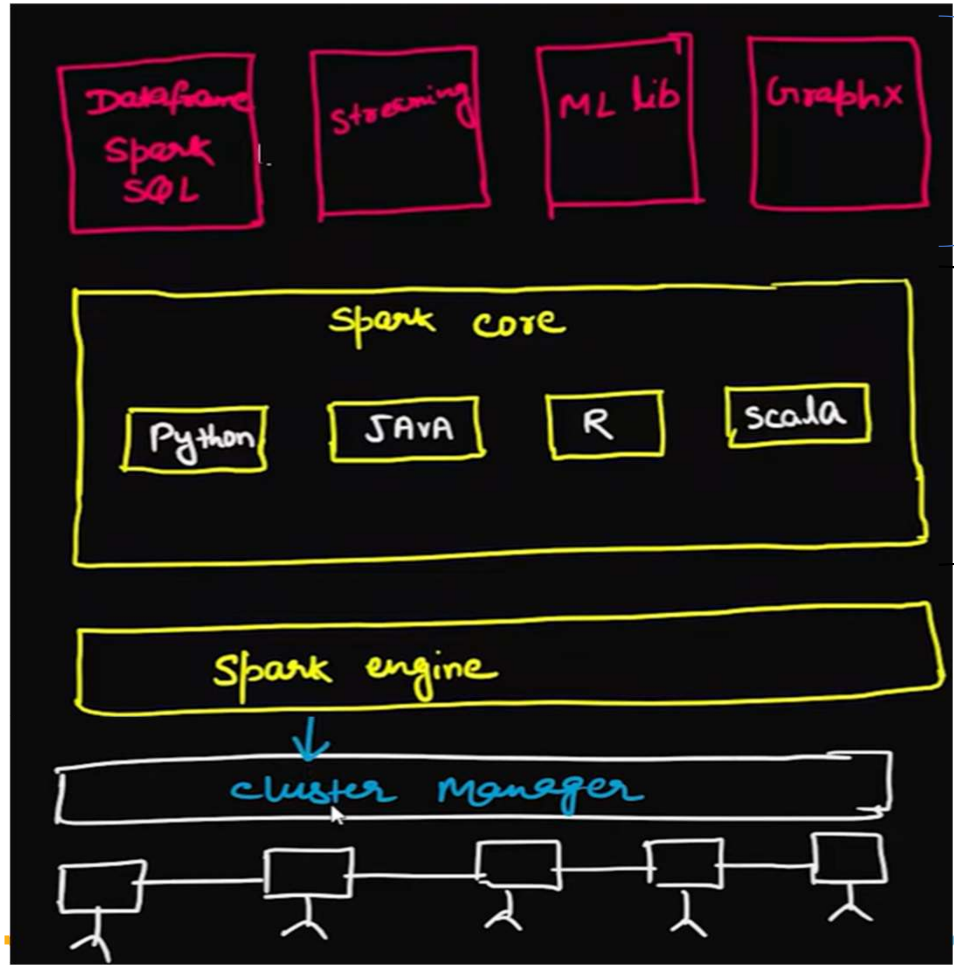
ELT → Extract load Transform

1. Storage → RAM
2. Processing → CPU

Approaches


Monolithic
Distributed
(i) Vertical scaling
(ii) Expensive
(iii) Low availability
(i) Horizontal scaling
(ii) Economical
(iii) High availability

Architecture

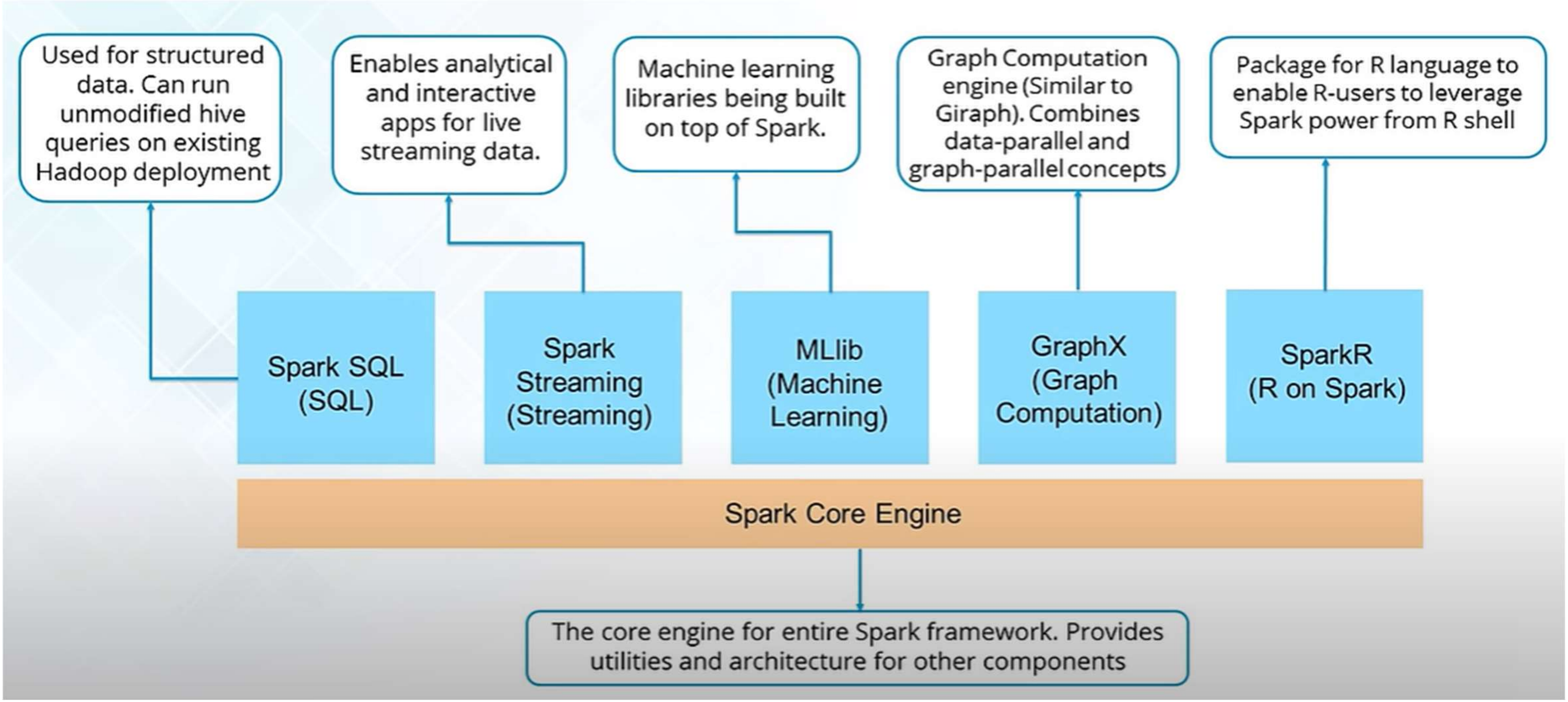


Librarie
High Level API
Dataframe/Dataset

Low Level API
RDD
SPARK CORE

Ex: YARN, MESOS, Kubernetes

Another view



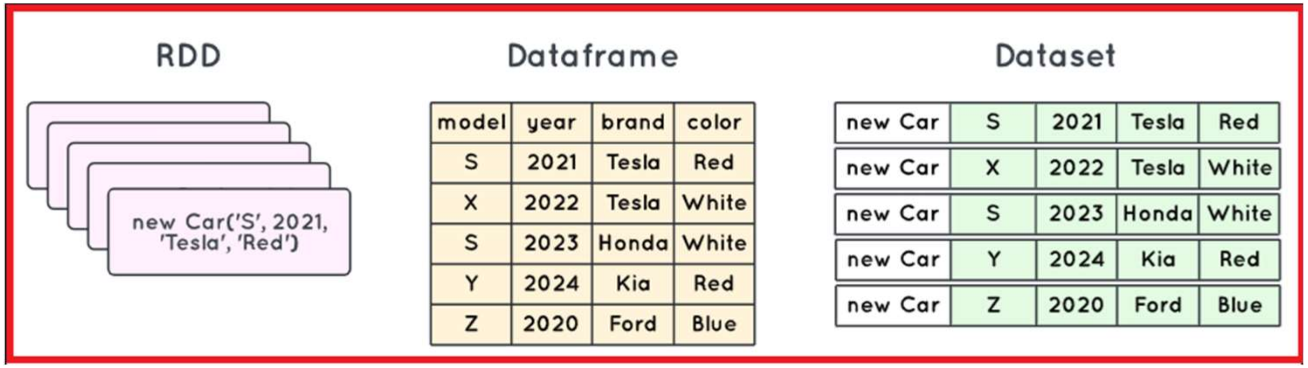
RDD V/S DataFrame V/S DataSet

Spark RDD stands for **Resilient Distributed Dataset** which is the core data abstraction API and is available since very first release of Spark (**Spark 1.0**).

It is a lower-level API for manipulating distributed collection of data. The RDD APIs exposes some extremely useful methods which can be used to get very tight control over underlying physical data structure.

It is an immutable (read only) collection of partitioned data distributed on different machines. RDD enables in-memory computation on large clusters to speed up big data processing in a fault tolerant manner.

Feature	RDD	DataFrame	DataSet
Immutable	Yes	Yes	Yes
Fault tolerant	Yes	Yes	Yes
Type-safe	Yes	No	Yes
Schema	No	Yes	Yes
Execution optimization	No	Yes	Yes
Level	Low	High	High



RDD

1. **Immutable collection:** RDD is an immutable partitioned collection distributed on different nodes. A partition is a basic unit of parallelism in Spark. The immutability helps to achieve fault tolerance and consistency.
2. **Distributed data:** RDD is a collection of distributed data which helps in big data processing by distributing the workload to different nodes in the cluster.
3. **Lazy evaluation:** The defined transformations do not gets evaluated until an action is called. It helps Spark in optimizing the overall transformations in one go.
4. **Fault tolerant:** RDD can be recomputed in case of any failure using DAG(Directed acyclic graph) of transformations defined for that RDD.
5. **Multi-language support:** RDD APIs supports **Python, R, Scala, and Java** programming languages.

Limitation : No optimization engine: RDD does not have an in-built optimization engine.

DataFrames

Spark 1.3 introduced two new data abstraction APIs – **DataFrame and DataSet**. The DataFrame APIs organizes the data into named columns like a table in relational database. It enables programmers to define schema on a distributed collection of data. Each row in a DataFrame is of object type row. Like an SQL table, each column must have same number of rows in a DataFrame. In short, DataFrame is lazily evaluated plan which specifies the operations needs to be performed on the distributed collection of the data. DataFrame is also an immutable collection.

Below are the features

1. **In-built Optimization:** When an action is called on a DataFrame, the Catalyst engine analyzes the code and resolves the references. Then, it creates a logical plan. After that, the created logical plan gets translated into an optimized physical plan. Finally, this physical plan gets executed on the cluster.
2. **Hive compatible:** The DataFrame is fully compatible with Hive query language. We can access all hive data, queries, UDFs, etc using Spark SQL from hive MetaStore and can execute queries against these hive databases.
3. **Structured, semi-structured, and highly structured data support:** DataFrame APIs supports manipulation of all kind of data from structured data files to semi-structured data files and highly structured parquet files.
4. **Multi-language support:** DataFrame APIs are available in **Python, R, Scala, and Java**.
5. **Schema support:** We can define a schema manually or we can read a schema from a data source which defines the column names and their data types.

Limitation: **Type safety:** Each row in a DataFrame is of object type row and hence is not strictly typed. That is why DataFrame does not support compile time safety.

DataSet

As an extension to the DataFrame APIs, **Spark 1.3** also introduced DataSet APIs which provides strictly typed and object-oriented programming interface in Spark. It is immutable, type-safe collection of distributed data. Like DataFrame, DataSet APIs also uses Catalyst engine in order to enable execution optimization. DataSet is an extension to the DataFrame APIs. Features and limitations of the DataSet are as below:

Features:

1.Combination of RDD and DataFrame: DataSet enables functional programming like RDD APIs and relational queries and execution optimization like DataFrame APIs. Thus, it provides the benefit of best of both worlds – RDDs and DataFrames.

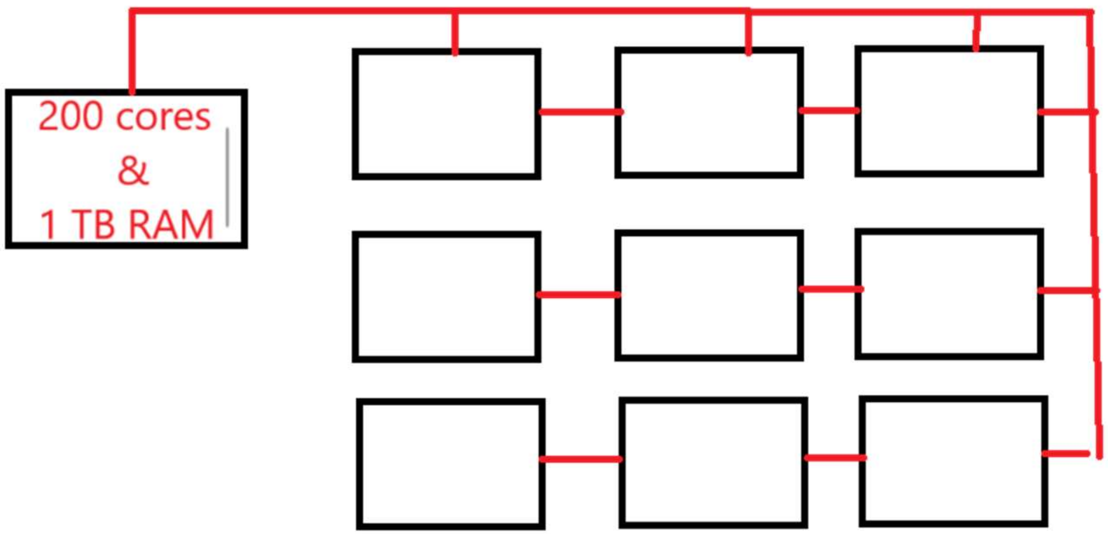
2.Type-safe: Unlike DataFrames, DataSet APIs provides compile time type safety. It conforms the specification at compile time using defined case classes (for Scala) or Java beans (for Java).

Limitations:

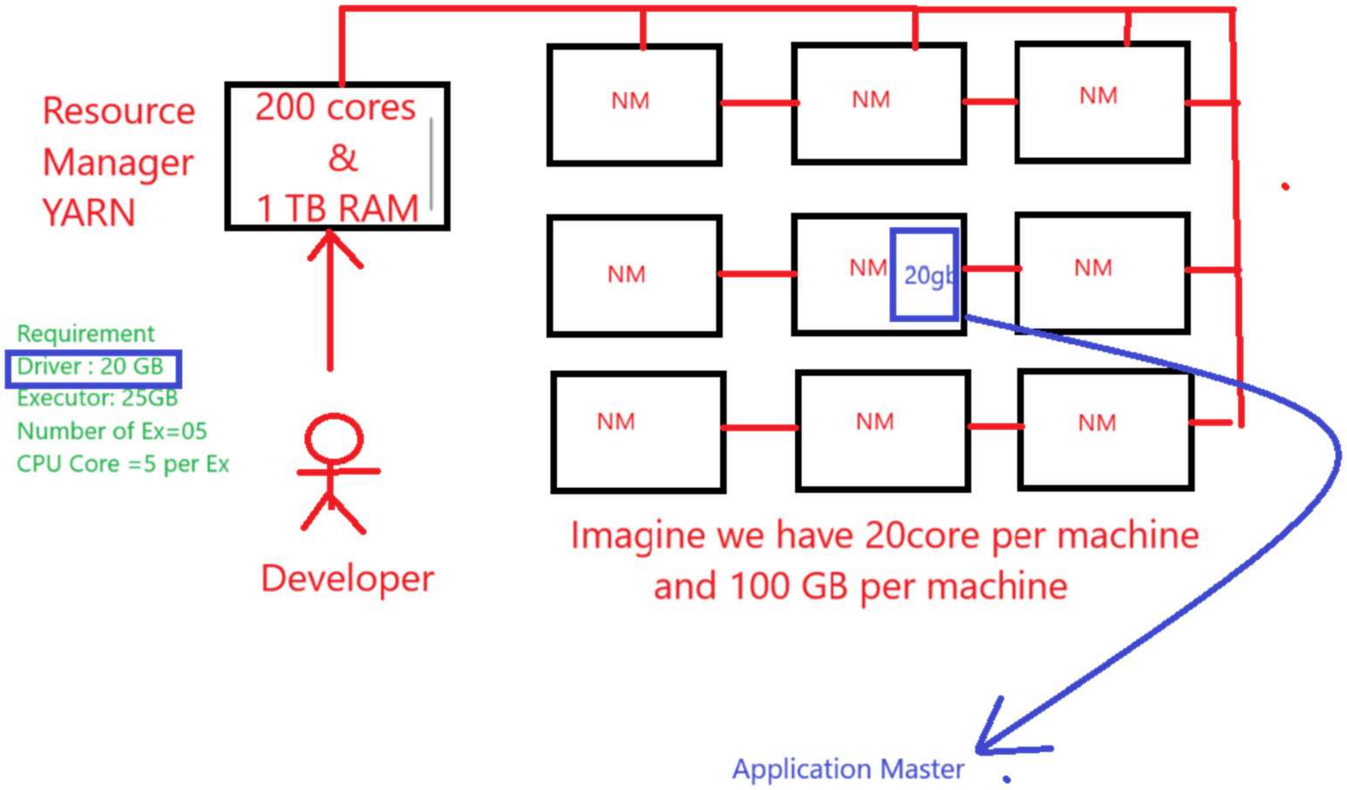
1.Limited language support: DataSet is only available to JVM based langauges like Java and Scala. Python and R do not support DataSet because these are dynamically typed languages.

2.High garbage collection: JVM types can cause high garbage collection and object instantiation cost.

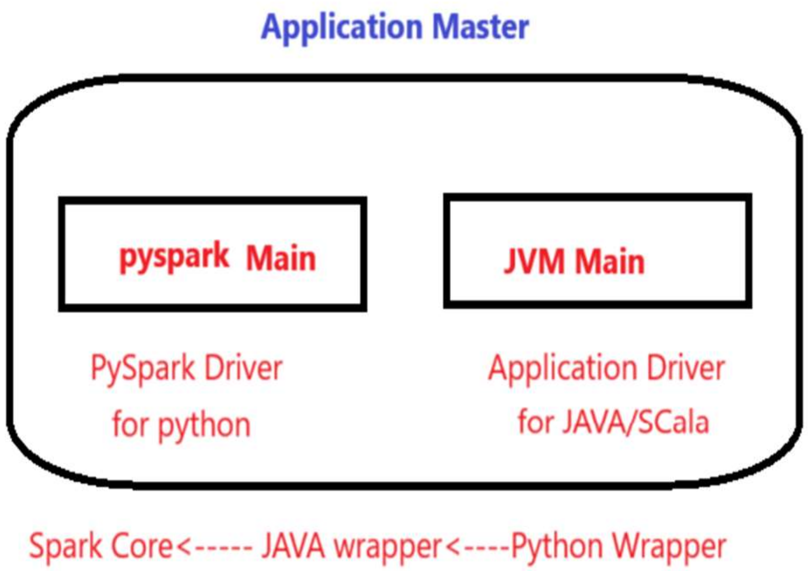
How it Works



Imagine we have 20core per machine
and 100 GB per machine


Resource
Manager
YARN
200 cores
&
1 TB RAM
Requirement
Driver : 20 GB
Executor: 25GB
Number of Ex=05
CPU Core =5 per Ex
Developer
NM
NM
NM
NM
NM
20gb
NM
NM
NM
NM
NM
Imagine we have 20core per machine
and 100 GB per machine
Application Master

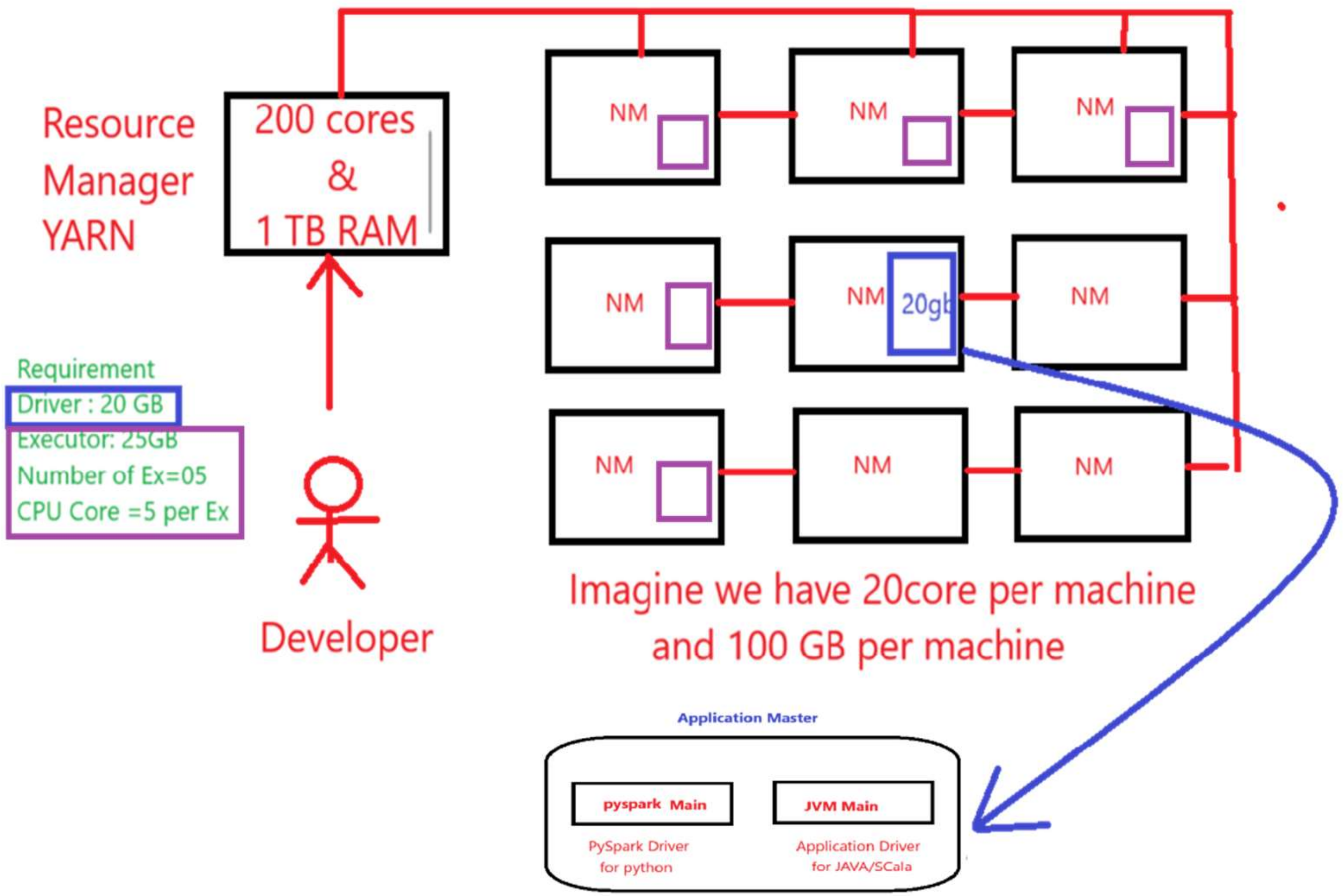
Application Master



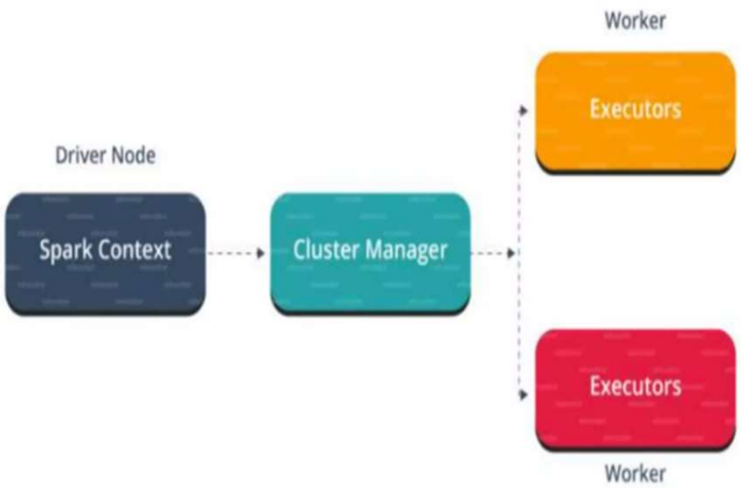
Executors and Driver

"driver" is the central coordinating process that manages the overall execution of a Spark application,

"executors" are distributed worker processes running on different nodes in the cluster that actually perform the computation tasks assigned by the driver; essentially, the driver is the brain that directs the work, and the executors are the workers who do the actual processing on the data.



Another View



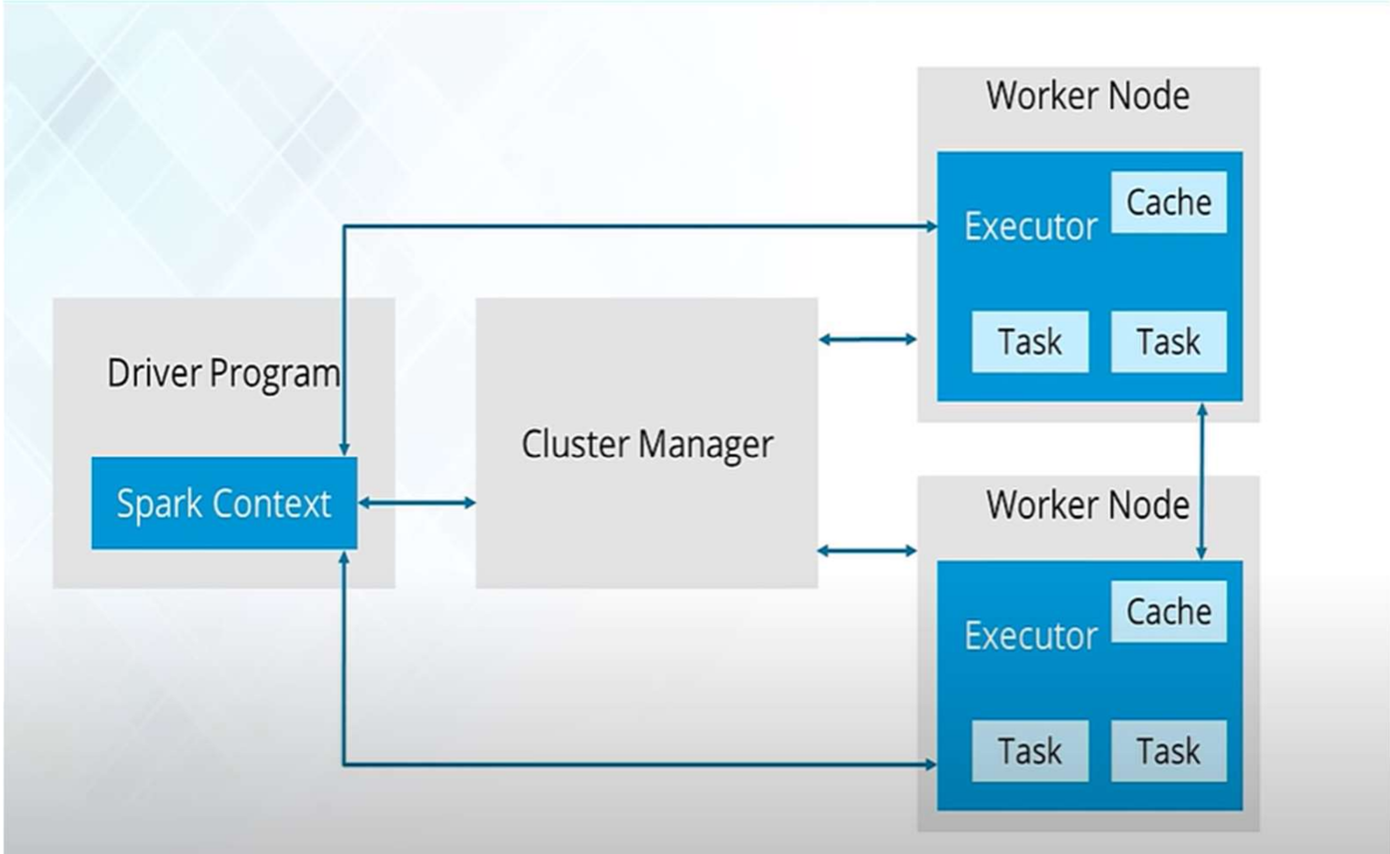
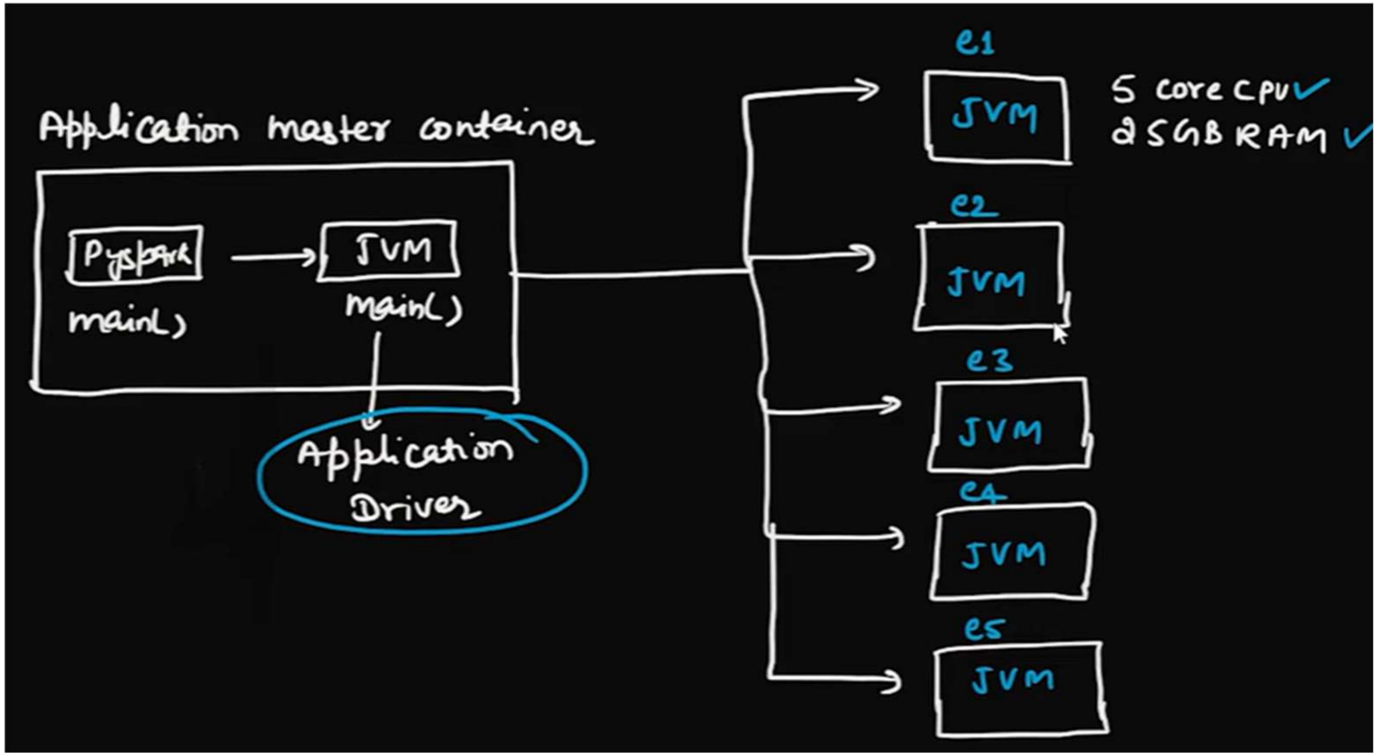


Figure: *Components of a Spark cluster*

Cont.…



Application master container
Pyspark
main()
JVM
main()
Application Driver
e1
JVM
5 core CPU
25GB RAM
e2
JVM
e3
JVM
e4
JVM
e5
JVM

compare

Hadoop is Build for Batch Data Processing

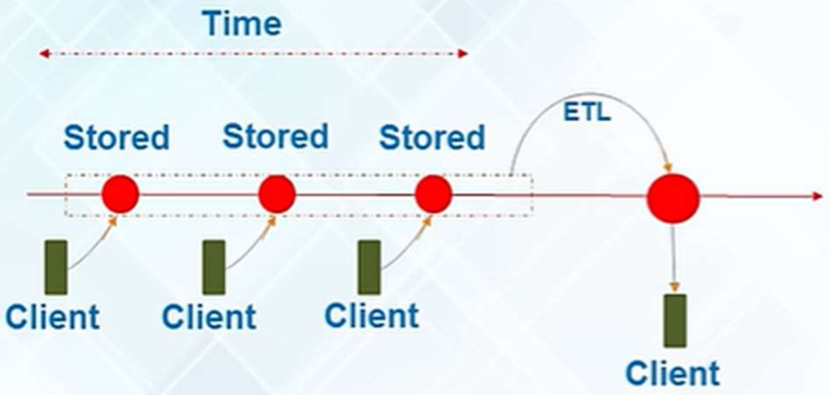
Difficult to write code in Hadoop…Hive was built as easier alternative

Spark is build for Stream data processing

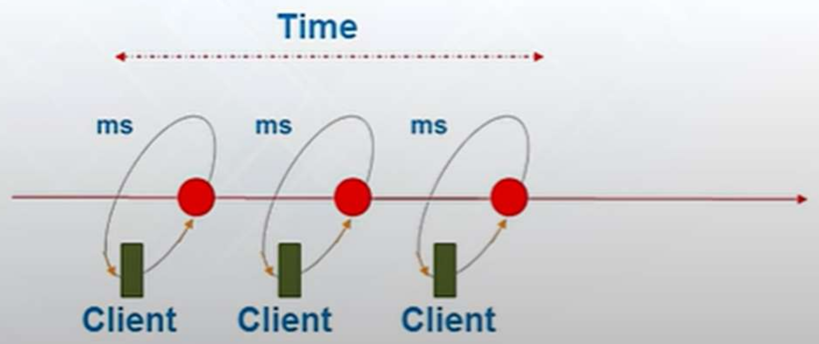
Very Easy to write and debug the code

Spark provide low and high level API

Batch and Real Time Processing

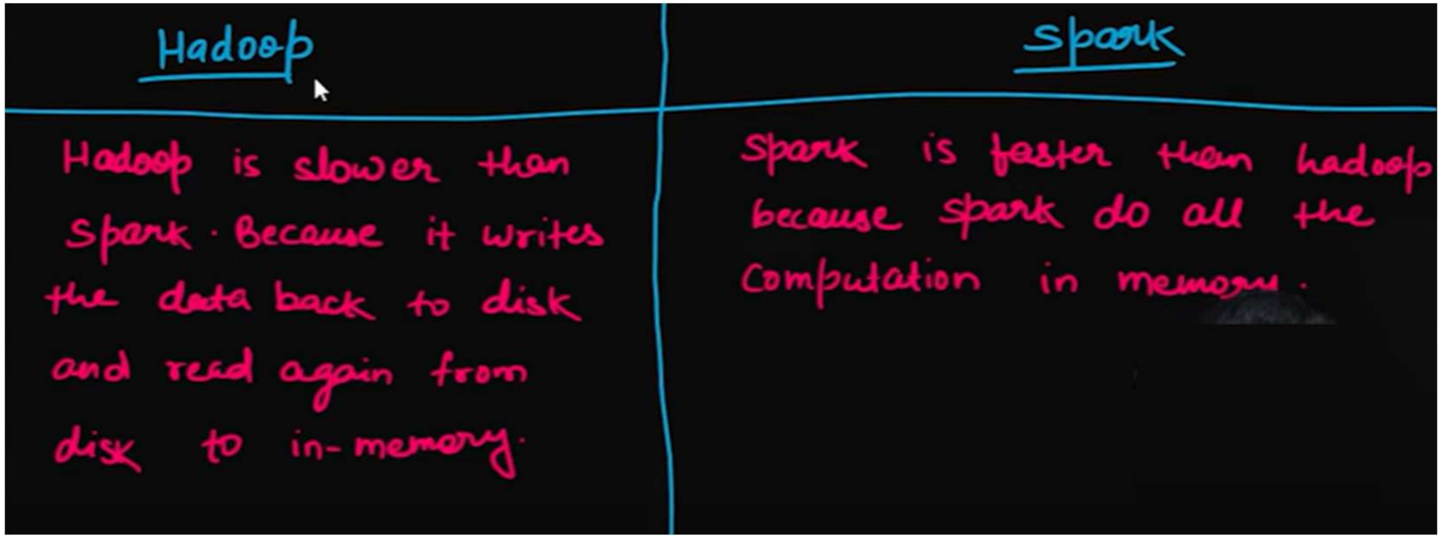


Analytics based on the data collected over a period of time is Batch Analytics

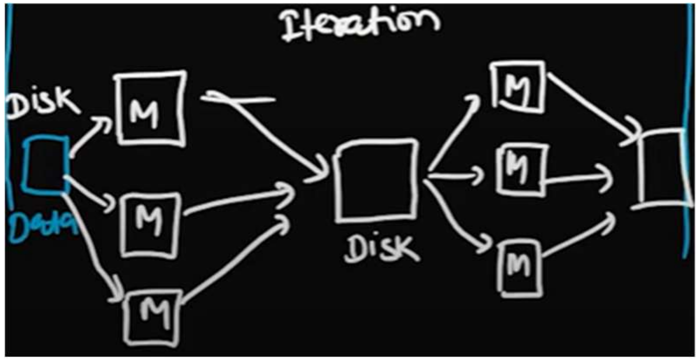


Analytics based on immediate data for instant result is Real-Time (Stream) Analytics

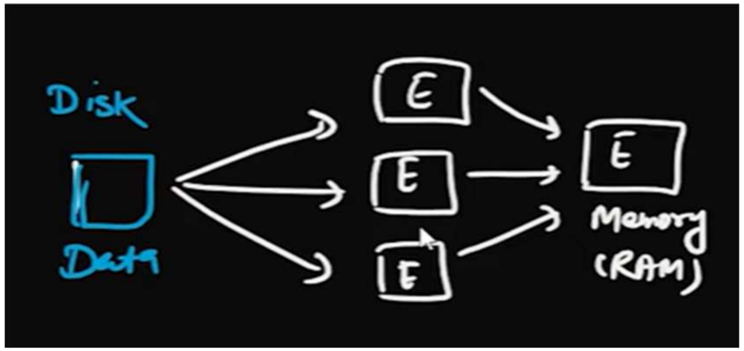
Compare



Hadoop
Spark
Hadoop is slower than spark. Because it writes the data back to disk and read again from disk to in-memory.
Spark is faster than hadoop because spark do all the computation in memory.

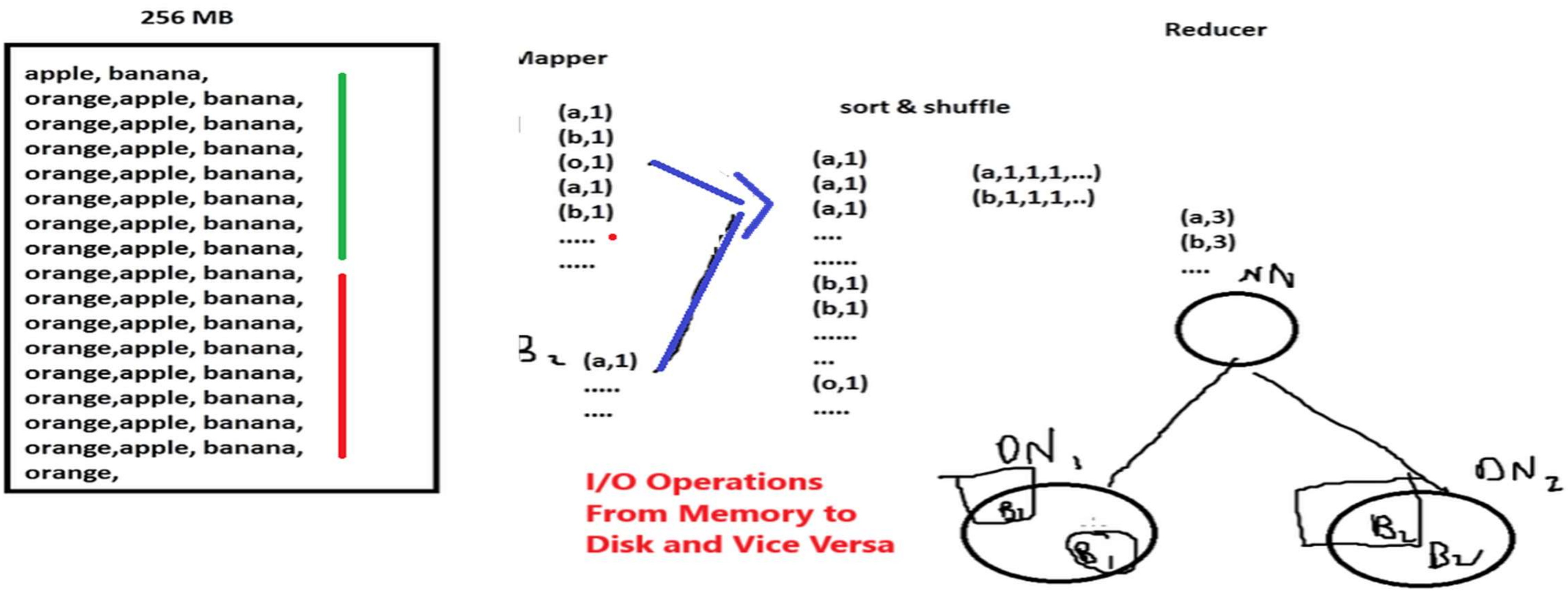


Iteration
Disk
Data
M
M
M
Disk
M
M
M

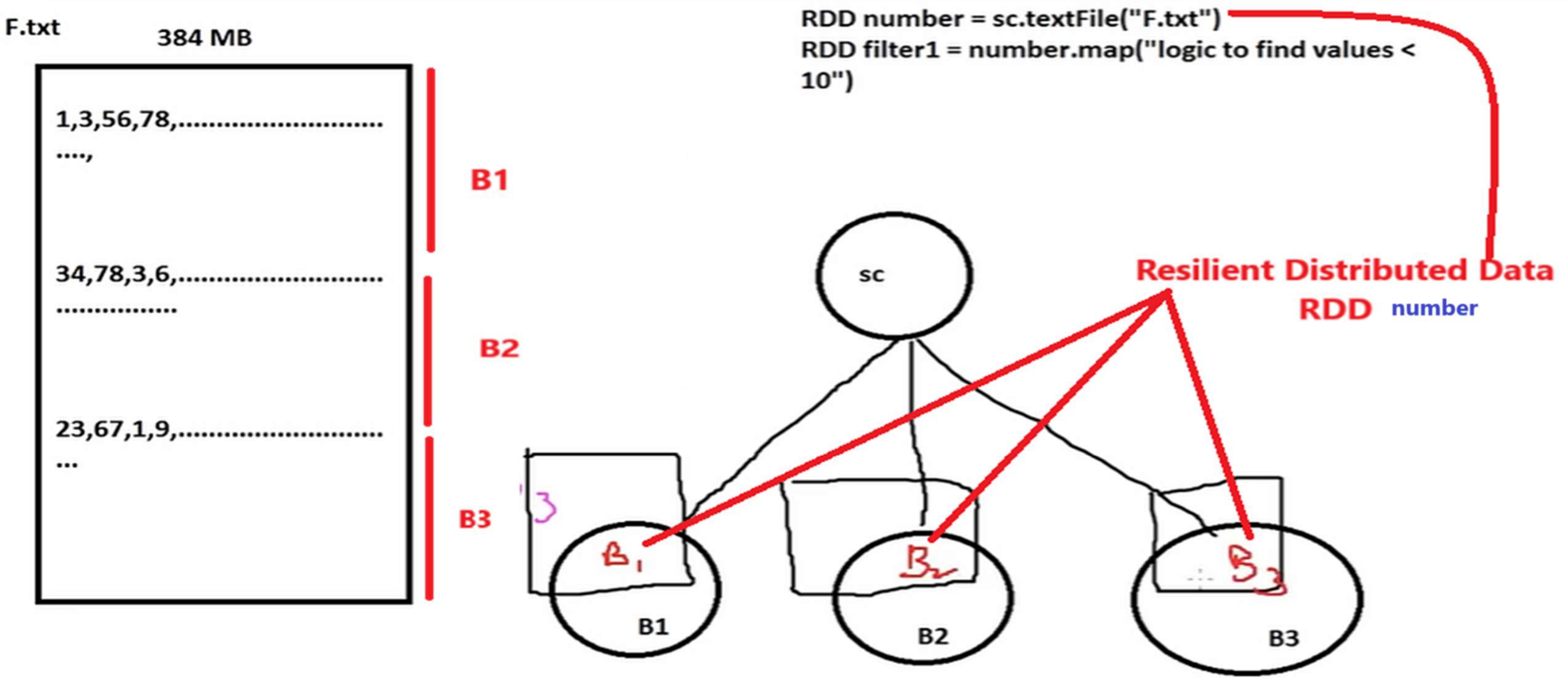


Disk
Data
E
E
E
E
Memory
(RAM)

Slow operations –in Map Reduce



Faster processing of Spark

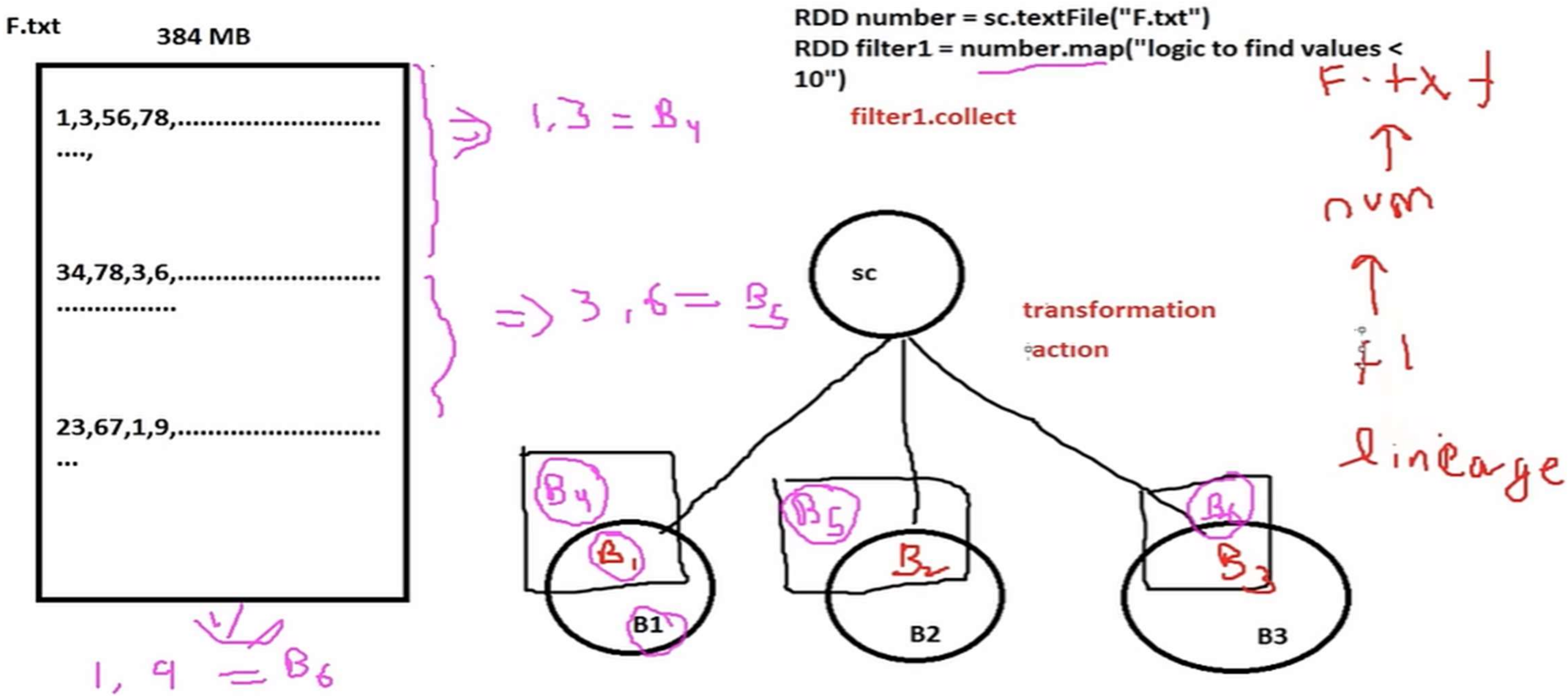


Hadoop has Data stored in blocks

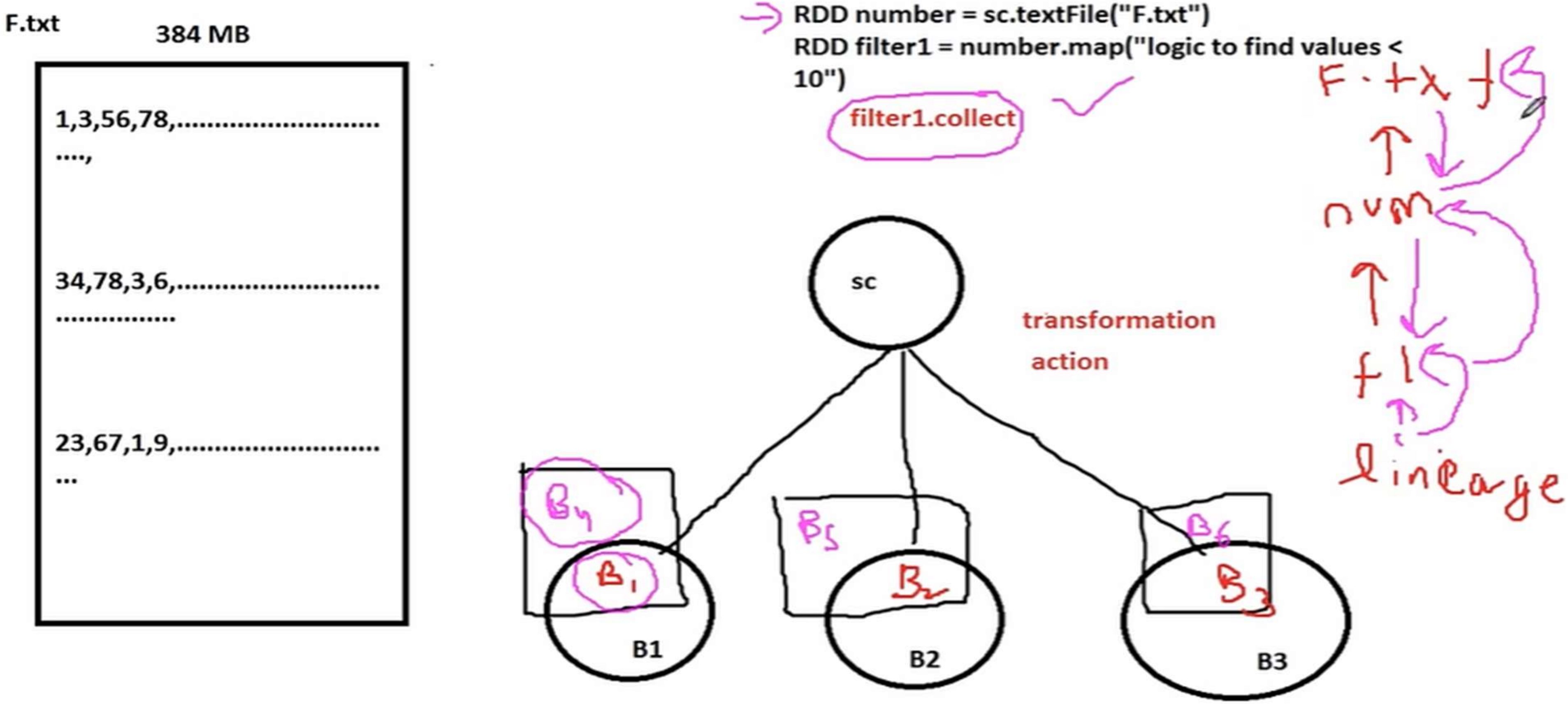
It replicated these blocks to handle failure

Spark use DAG to provide Fault tolerance

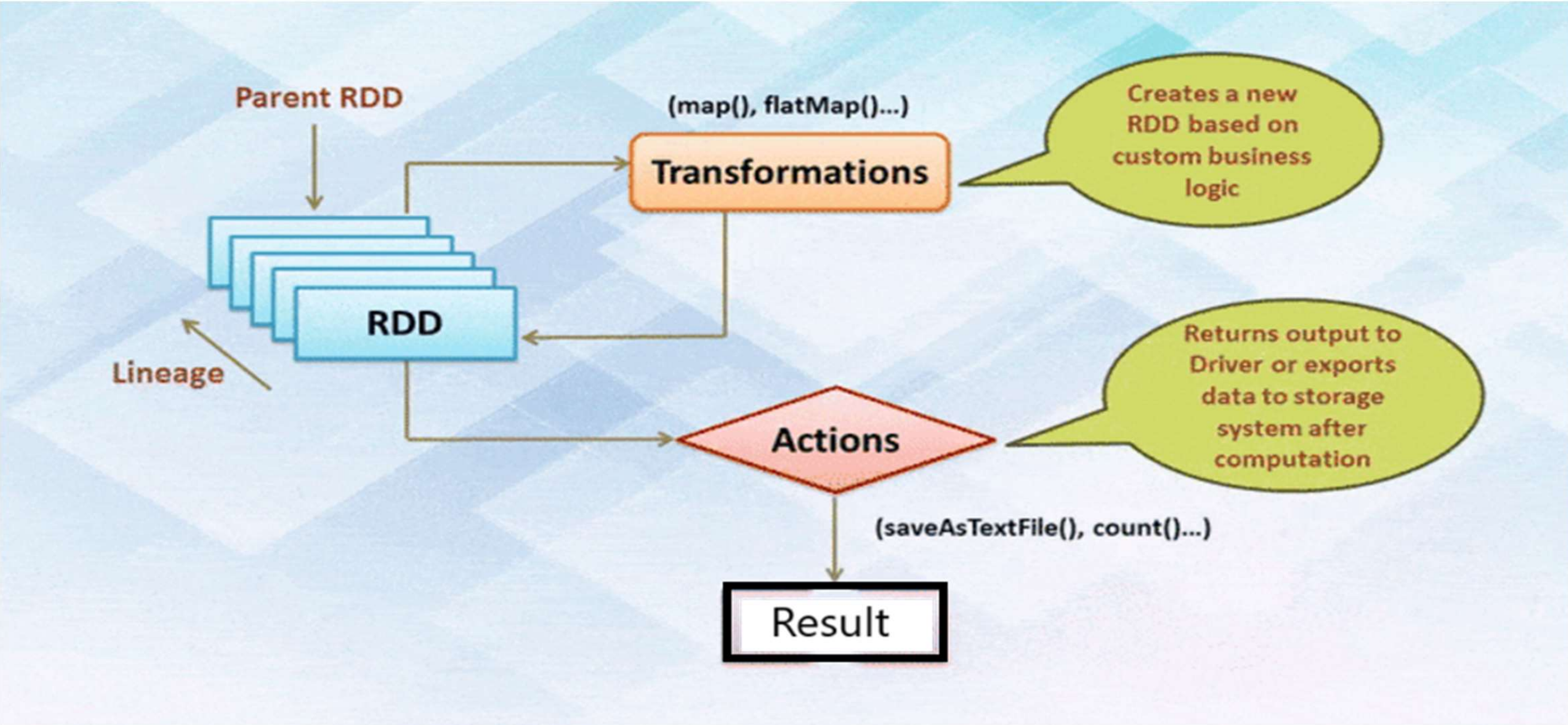
Filter..Collect..transform…action



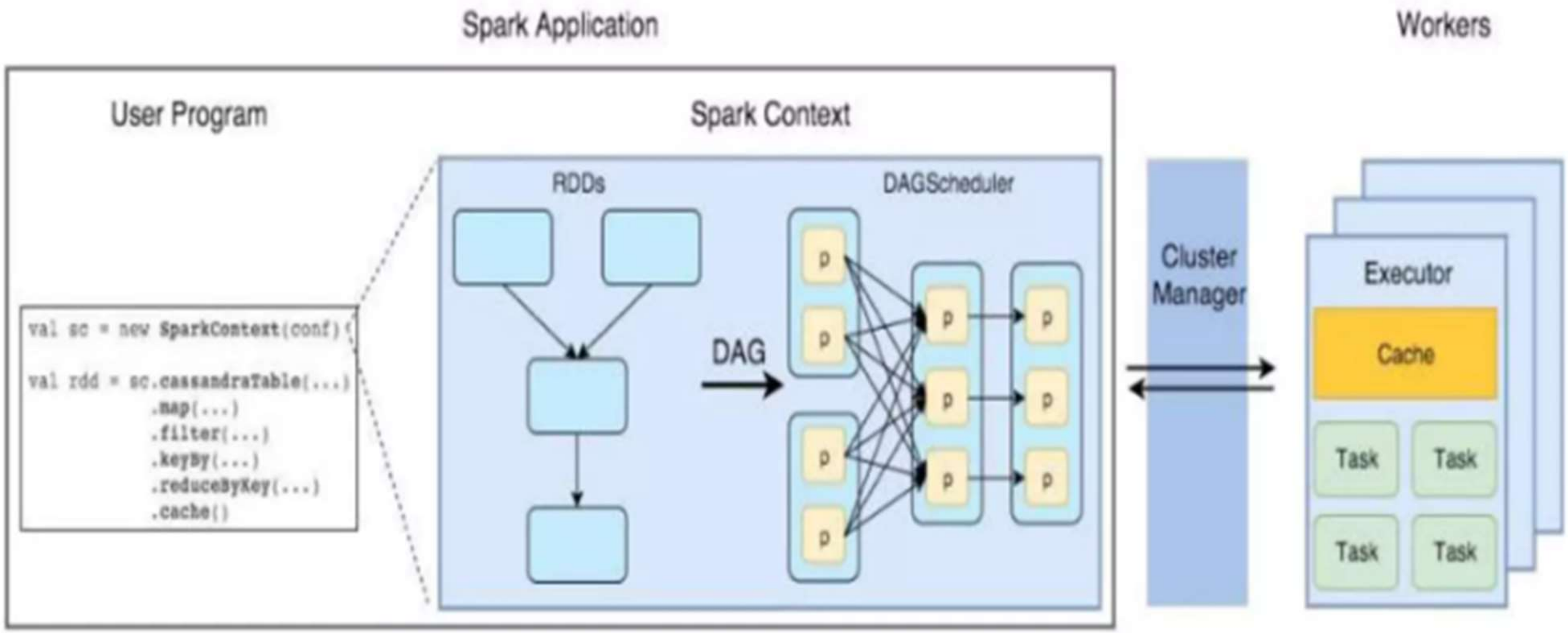
Lazy Evaluation - Lineage



Spark Lazy Evaluation



Another View



Some Actions - Count

In PySpark, actions are operations that trigger the execution of a computation on a DataFrame and return a result to the driver program. Here's a breakdown of the actions you mentioned:

count():

- **Purpose:** Returns the number of rows in a DataFrame.
- **Example:**

Python

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
df = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

row_count = df.count()
print(row_count)  # Output: 2
```

Some Actions - Show

show():

- **Purpose:** Displays the first few rows of a DataFrame in a tabular format.
- **Example:**

Python

```
df.show()
# Output:
# +---+-----+
# | id| name|
# +---+-----+
# |  1|Alice|
# |  2|  Bob|
# +---+-----+
```

Some Actions - collect

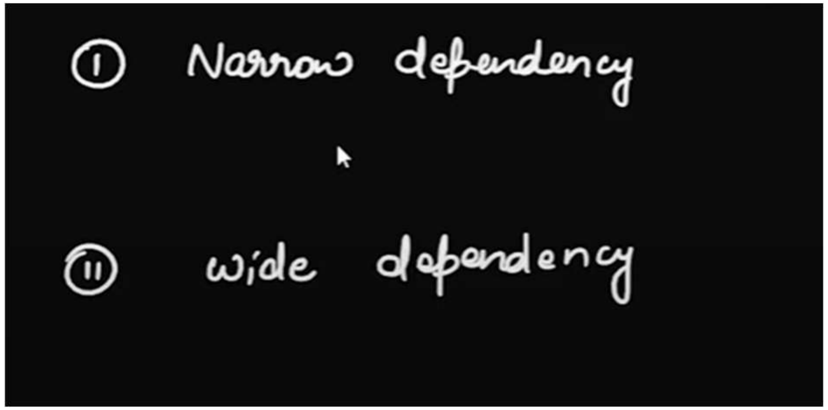
collect():

- **Purpose:** Retrieves all the rows of a DataFrame as a list of Row objects on the driver program.
- **Caution:** Use with caution on large datasets, as it can cause memory issues on the driver.
- **Example:**

Python

```python
data = df.collect()
print(data)  # Output: [Row(id=1, name='Alice'), Row(id=2, name='Bob')]
```

Transformation Types



•Narrow transformations

•Each input partition is used to compute one output partition. These transformations are preferred because they are more efficient and require less data movement. Examples of narrow transformations include map(), filter(), and union().

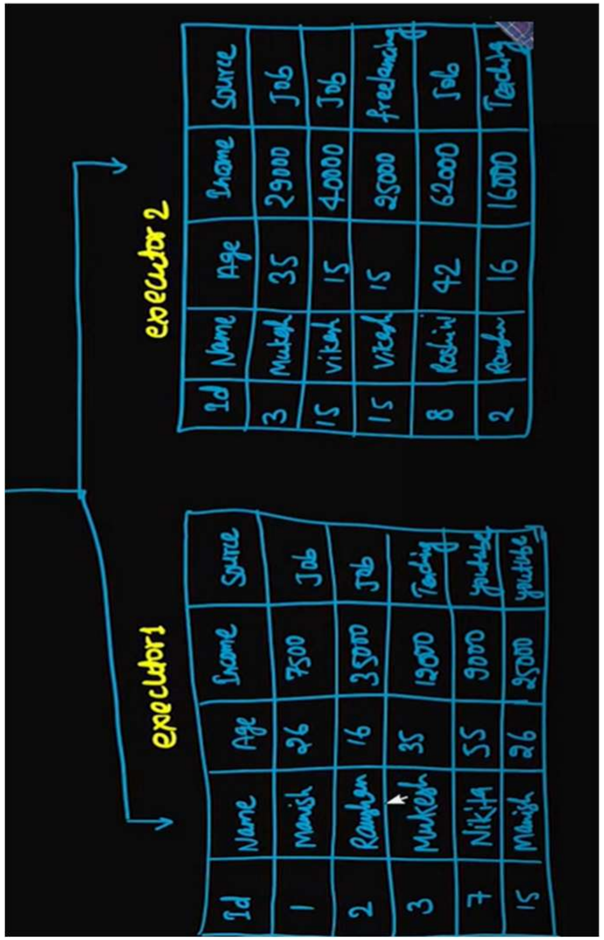
•Wide transformations

•Each input partition is used to compute multiple output partitions. These transformations are more resource-intensive and time-consuming than narrow transformations, especially when dealing with large datasets. Wide transformations may change the number of partitions in the output RDD or DataFrame.

The choice between narrow and wide transformations is important for optimizing performance and resource utilization. Developers can use this knowledge to design jobs that execute efficiently across the cluster.

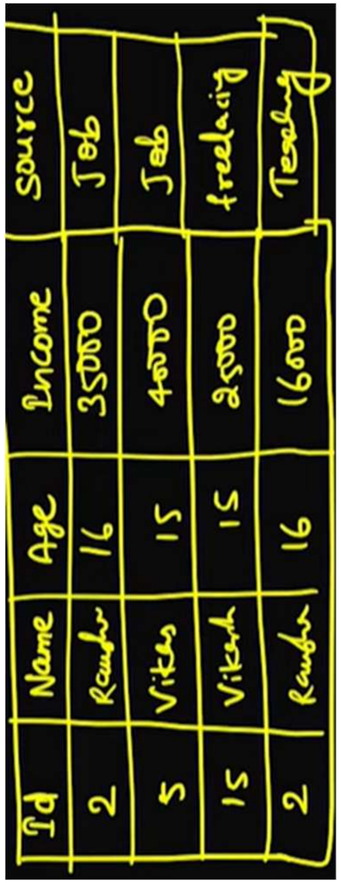
Show name of employee with age less than 18?
Find total income of each employee

Id	Name	Age	Income	source
1	Manish	26	7500	Job
2	Raushan	16	35000	Job
3	Mukesh	35	12000	Teaching
7	Nikita	55	9000	youtube
15	Vikash	15	25000	Freelancing
3	Mukesh	35	29000	Job
15	Vikash	15	40000	Job
1	Manish	26	25000	youtube
8	Roshini	42	62000	Job
2	Raushan	16	16000	Teaching

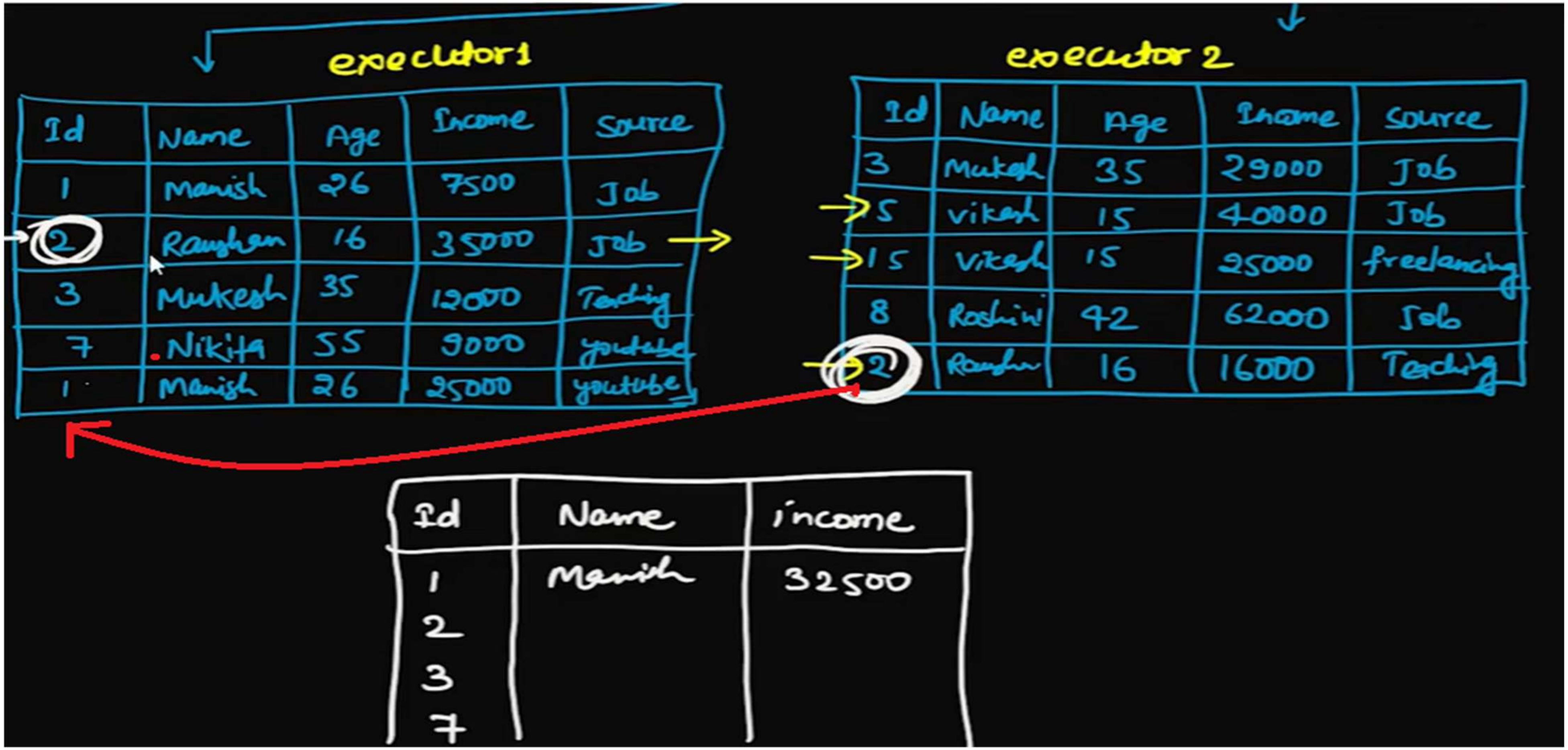


No
Data
Movement

=



Wide transformation—Costly affair



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	✗	✓

Compare

Hadoop Use : Kerberos for Authentication
ACL for Authorization

Spark: Do not have its own strong security features

On having access to HDFS it gains ACL controls

On having access to YARN Spark gains Kerberos Authentication credentials

Spark Gels well

innovate achieve lead

Spark + HDFS

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

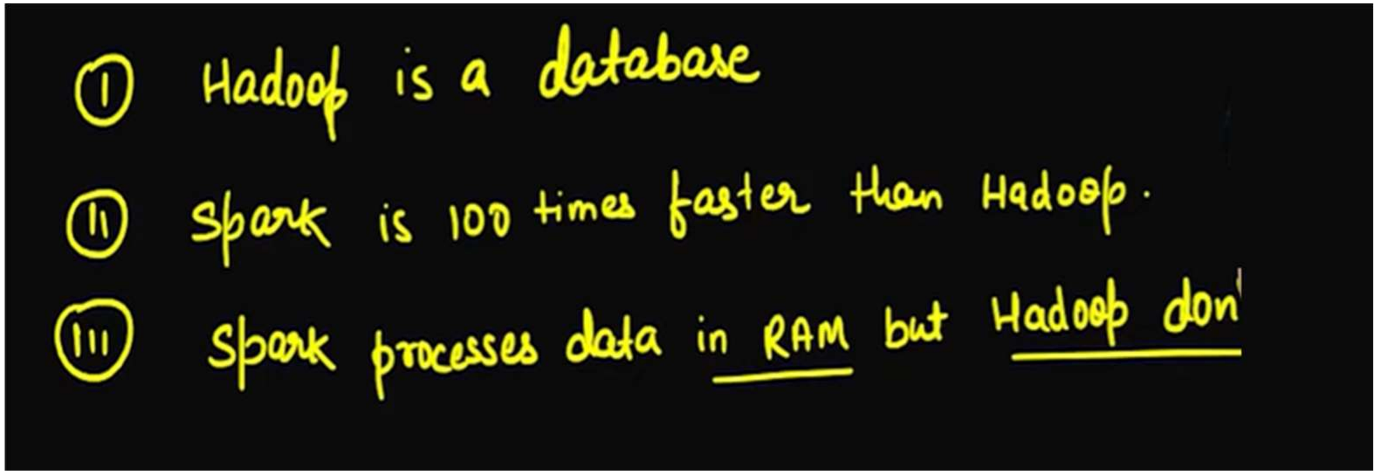
Spark + MapReduce

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)

Some Misconceptions



That is all…..

Heading to Revision lecture …..Next Week