Fig: **Big Data Word Cloud**

✓The Myth about Big Data

- ❑ Big Data Is *New*
- ❑ Big Data Is Only About *Massive Data Volume*
- ❑ Big Data Means *Hadoop*
- ❑ Big Data Need A *Data Warehouse*
- ❑ Big Data Means *Unstructured Data*
- ❑ Big Data Is for *Social Media & Sentiment Analysis*

Big Data is …….

- ✓Big data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making.

- ✓Big data is data which is too large, complex and dynamic for any conventional data tools to capture, store, manage and analyze.

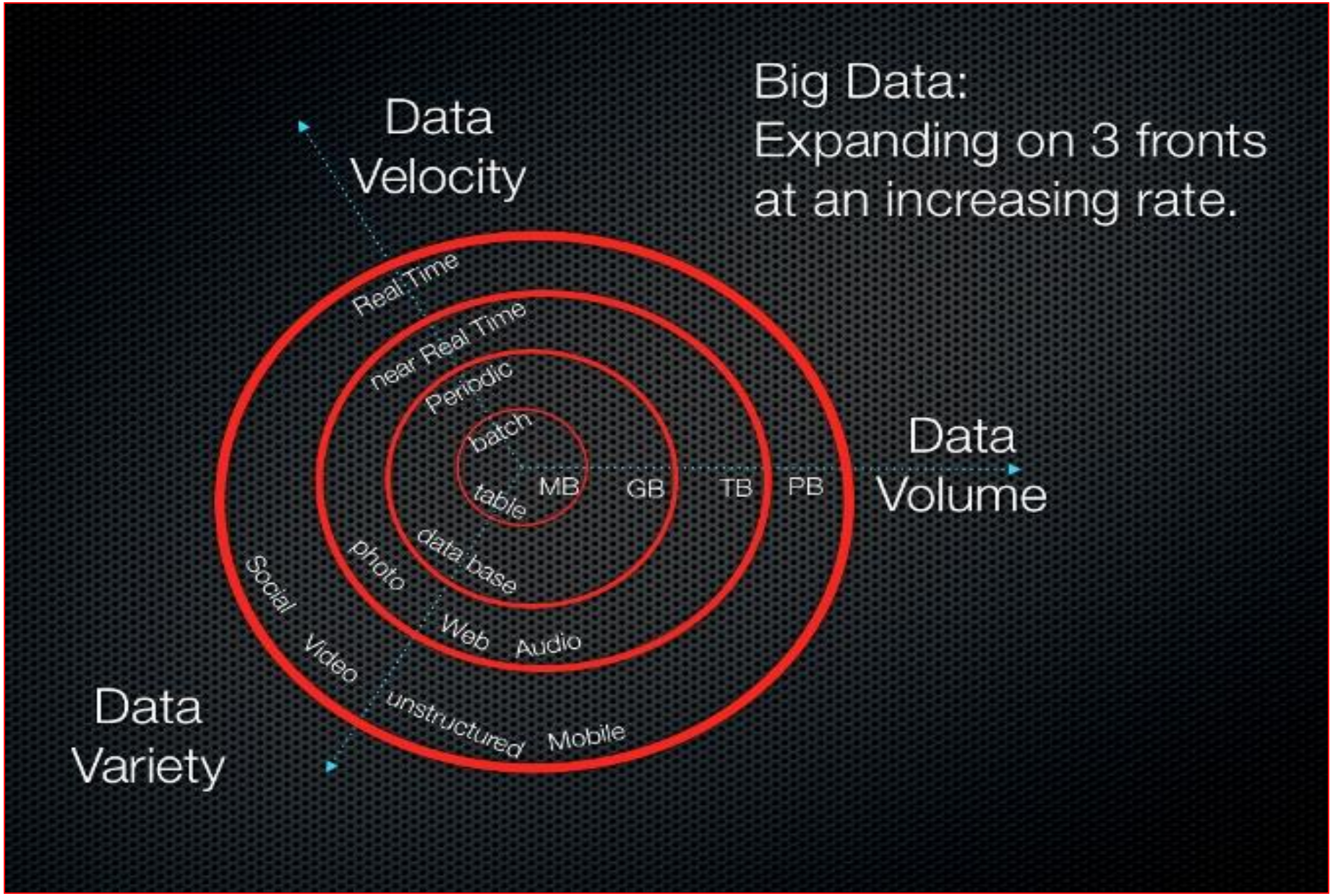


Fig: Big Data - Characteristics

- ✓ Big Data Analytics is the process of examining large amounts of data of a variety of types (big data) to uncover hidden patterns, unknown correlations and other useful information.

- ✓ Why do they care about Big Data?

- ✓ More knowledge leads to better customer engagement, fraud prevention and new products.

- ✓ Data Evolution is the 10% are structured and 90% are unstructured like emails, videos, facebook posts, website clicks etc.

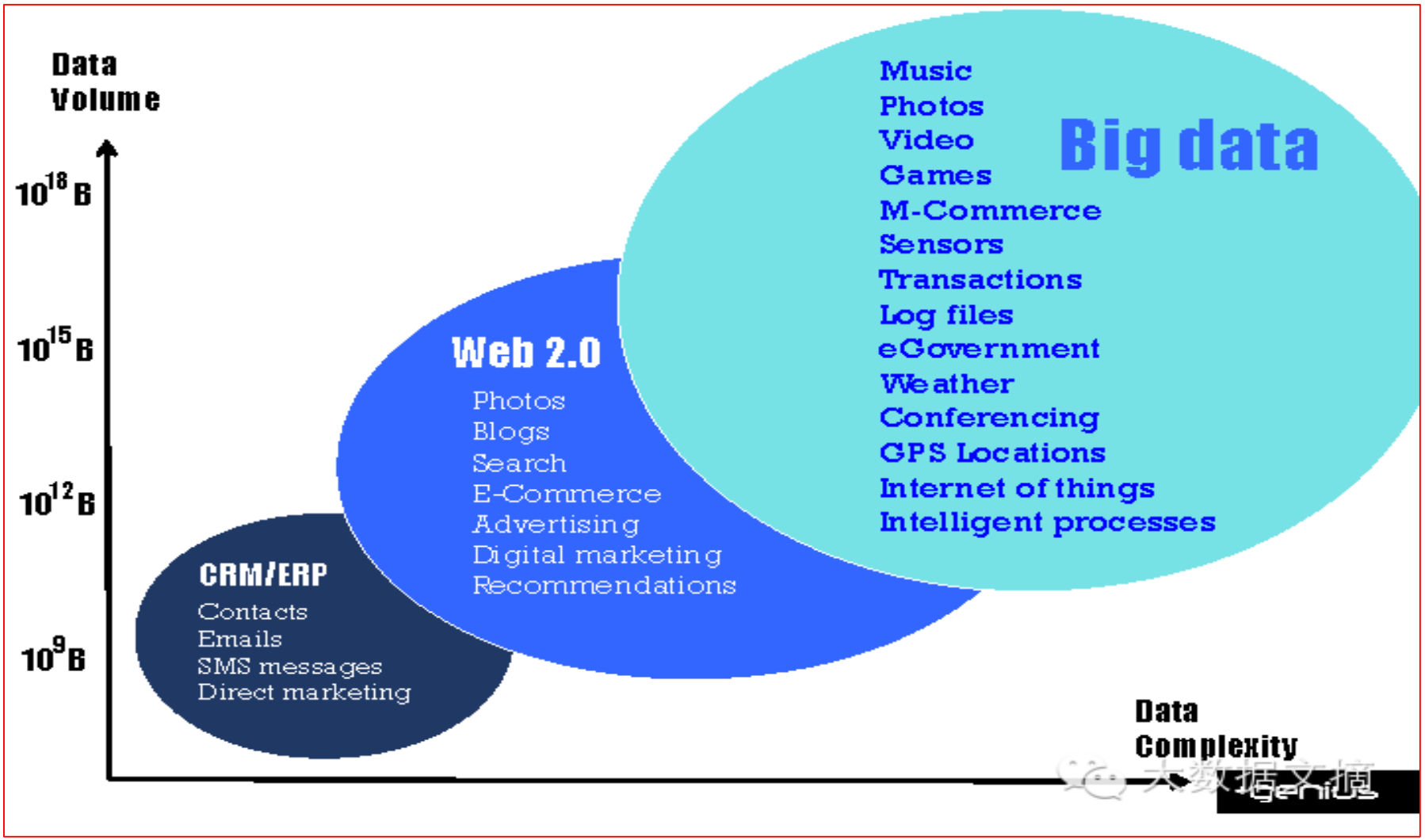


Fig: The Evolution of BIG DATA

- ✓ Big data is a collection of data sets which is so large and complex that it is difficult to handle using DBMS tools.

- ✓ Facebook alone generates more than 500 terabytes of data daily whereas many other organizations like Jet Air and Stock Exchange Market generates terabytes of data every hour.

- ✓ Types of Data:

1. Structured Data- These data is organized in a highly mechanized and manageable way. Ex: Tables, Transactions, Legacy Data etc…
2. Unstructured Data- These data is raw and unorganized, it varies in its content and can change from entry to entry. Ex: Videos, images, audio, Text Data, Graph Data, social media etc.
3. Semi-Structured Data- Ex: XML Database, 50% structured and 50% unstructured

✓ Big Data Matters…….

- ✓ Data Growth is huge and all that data is valuable.
- ✓ Data won't fit on a single system, that's why use Distributed data
- ✓ Distributed data = Faster Computation.

- ✓ More knowledge leads to better customer engagement, fraud prevention and new products.
- ✓ Big Data Matters for Aggregation, Statistics, Indexing, Searching, Querying and Discovering Knowledge.

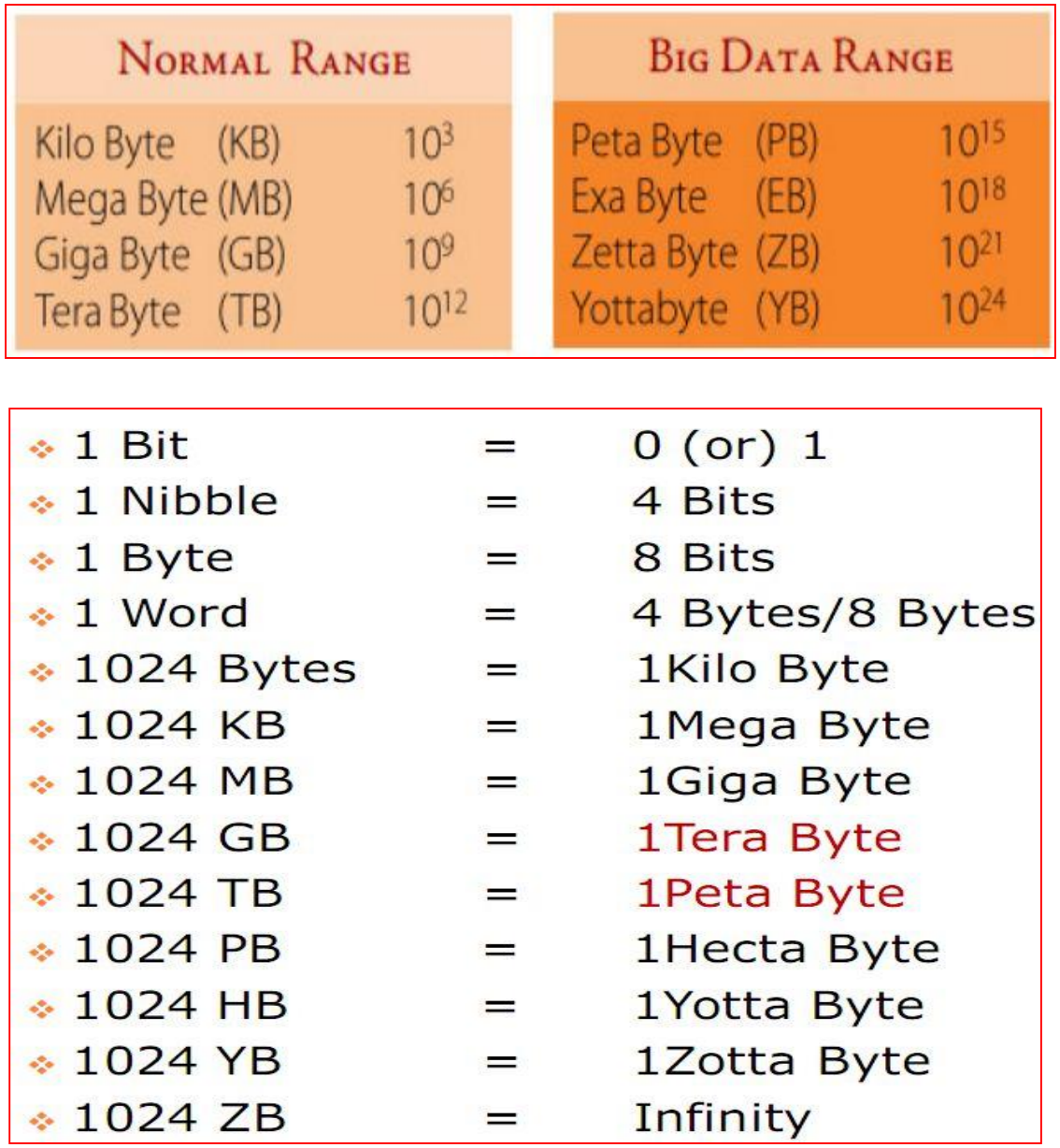


Fig: Measuring the Data in Big Data System

- ✓ **Big Data Sources:** Big data is everywhere and it can help organisations any industry in many different ways.

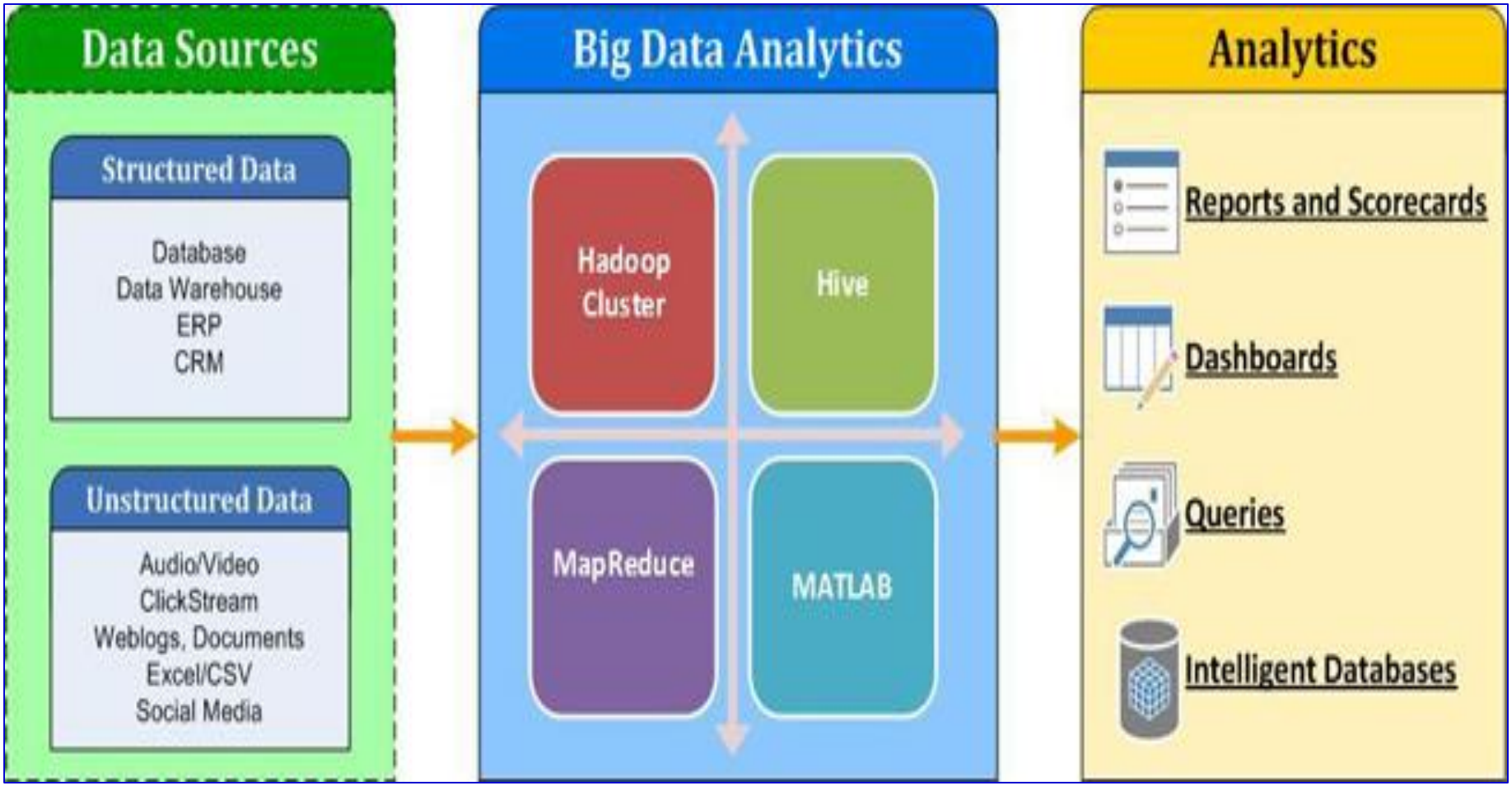
- ✓ Big data has become too complex and too dynamic to be able to process, store, analyze and manage with traditional data tools.

- ✓ Big Data Sources are
  - ✓ ERP Data
  - ✓ Transactions Data
  - ✓ Public Data
  - ✓ Social Media Data
  - ✓ Sensor Media Data
  - ✓ Big Data in Marketing
  - ✓ Big Data in Health & Life Sciences
  - ✓ Cameras Data
  - ✓ Mobile Devices
  - ✓ Machine sensors
  - ✓ Microphones

Fig: Big Data Sources

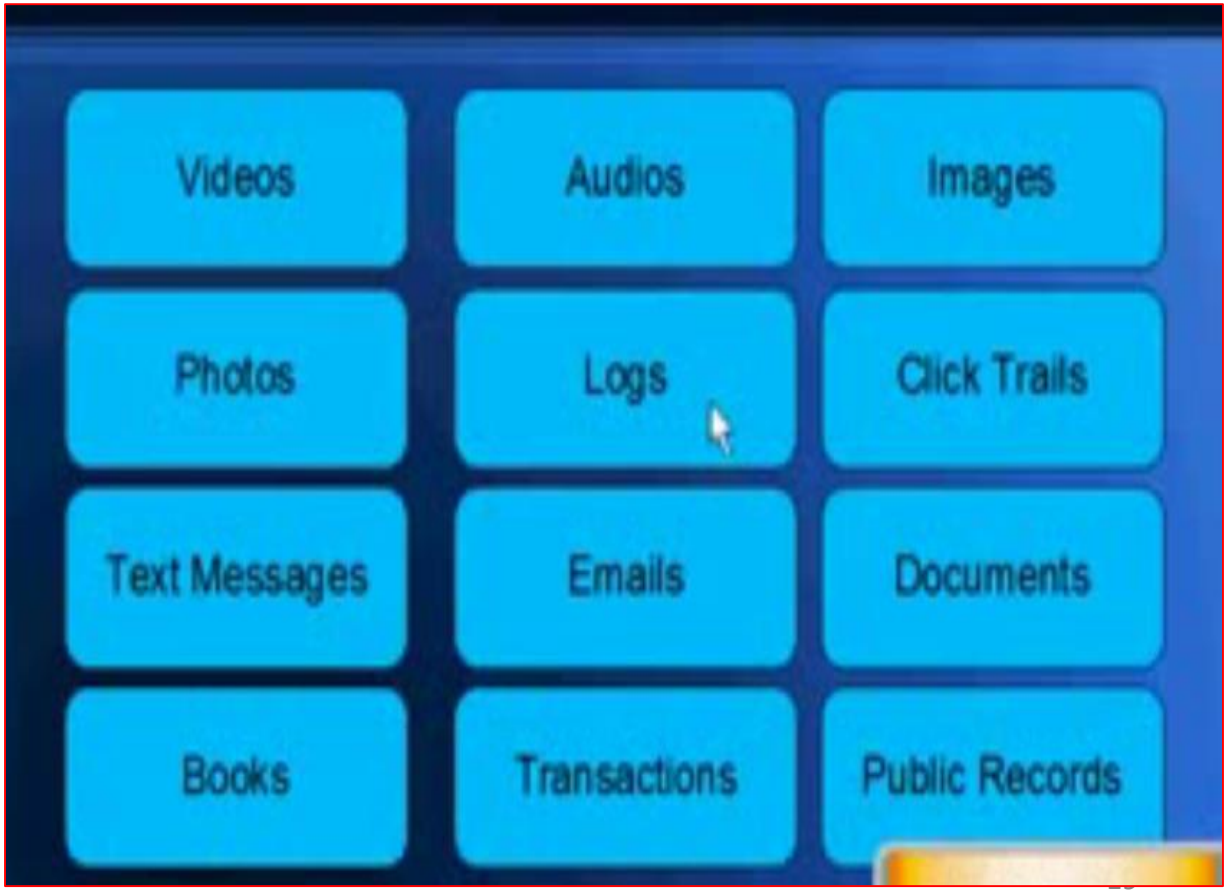
Structure of Big Data

- ✓ Big Data Processing: So-called big data technologies are about discovering patterns (in semi/unstructured data) development of big data standards & (open source) software commonly driven by companies such as Google, Facebook, Twitter, Yahoo! …



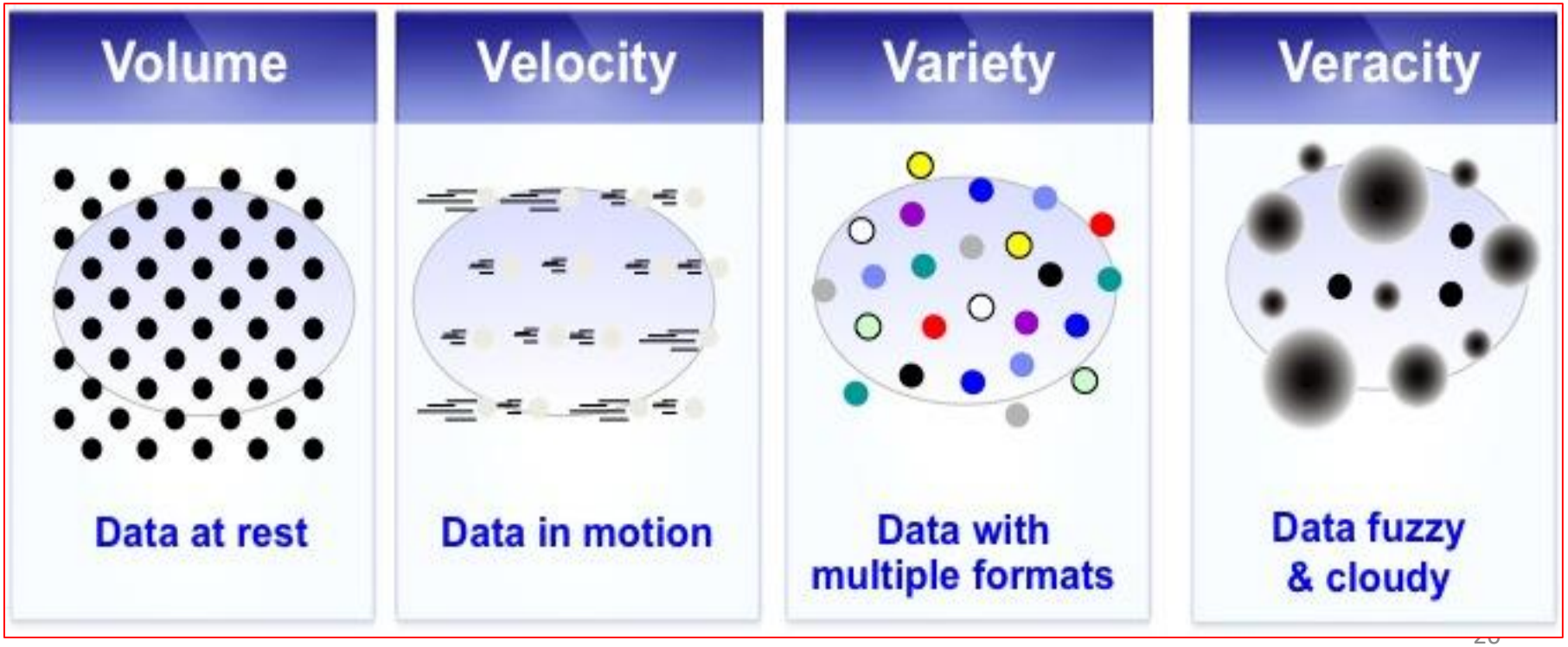
Big Data File Formats

- ✓ Videos
- ✓ Audios
- ✓ Images
- ✓ Photos
- ✓ Logs
- ✓ Click Trails
- ✓ Text Messages
- ✓ E-Mails
- ✓ Documents
- ✓ Books
- ✓ Transactions
- ✓ Public Records
- ✓ Flat Files
- ✓ SQL Files
- ✓ DB2 Files
- ✓ MYSQL Files
- ✓ Tera Data Files
- ✓ MS-Access Files



Characteristics of Big Data

- ✓ The are seven characteristics of Big Data are volume, velocity, variety, veracity, value, validity and visibility. Earlier it was assessed in megabytes and gigabytes but now the assessment is made in terabytes.



1. Volume: Data size or the amount of Data or Data quantity or Data at rest..

2. Velocity: Data speed or Speed of change or The content is changing quickly or Data in motion.

3. Variety: Data types or The range of data types & sources or Data with multiple formats.

4. Veracity: Data fuzzy & cloudy or Messiness or Can we trust the data.

5. Value: Data alone is not enough, how can value be derived from it.

6. Validity: Ensure that the interpreted data is sound.

7. Visibility: Data from diverse sources need to be stitched together.

Advantages

- ✓ Flexible schema
- ✓ Massive scalability
- ✓ Cheaper to setup
- ✓ Improving Healthcare and Public Health
- ✓ Financial Trading
- ✓ Improving Security and Law Enforcement
- ✓ No declarative query language
- ✓ Higher performance
- ✓ Detect risks and check frauds
- ✓ Reduce Costs

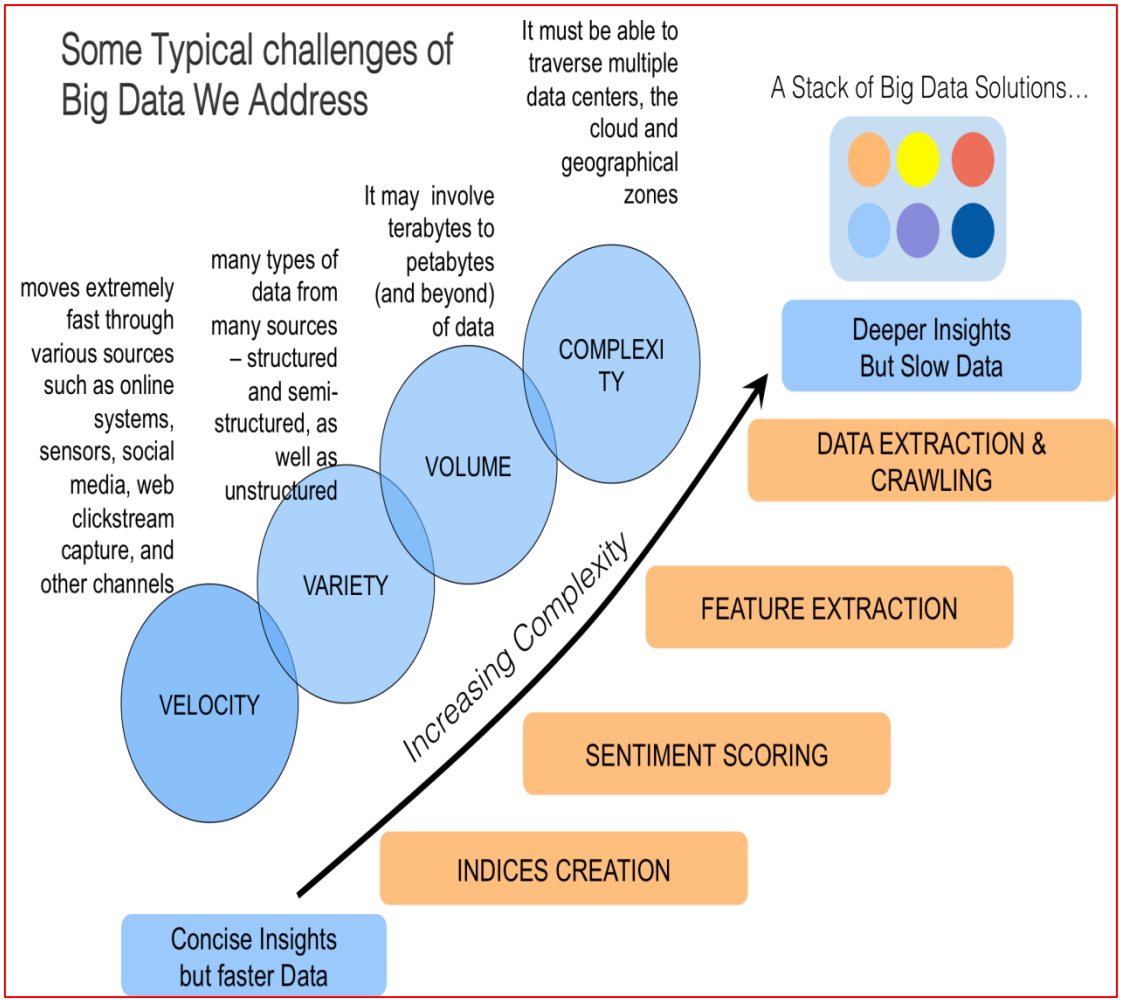
Fig) Advantages of Big Data

Disadvantages

- ✓ Big data violates the privacy principle.
- ✓ Data can be used for manipulating customers.
- ✓ Big data may increase social stratification.
- ✓ Big data is not useful in short run.
- ✓ Faces difficulties in parsing and interpreting.
- ✓ Big data is difficult to handle -more programming
- ✓ Eventual consistency - fewer guarantees

Big Data Challenges

- ✓ Data Complexity
- ✓ Data Volume
- ✓ Data Velocity
- ✓ Data Variety
- ✓ Data Veracity
- ✓ Capture data
- ✓ Curation data
- ✓ Performance
- ✓ Storage data
- ✓ Search data
- ✓ Transfer data
- ✓ Visualization data
- ✓ Data Analysis
- ✓ Privacy and Security



✓ Big Data Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, and information privacy.

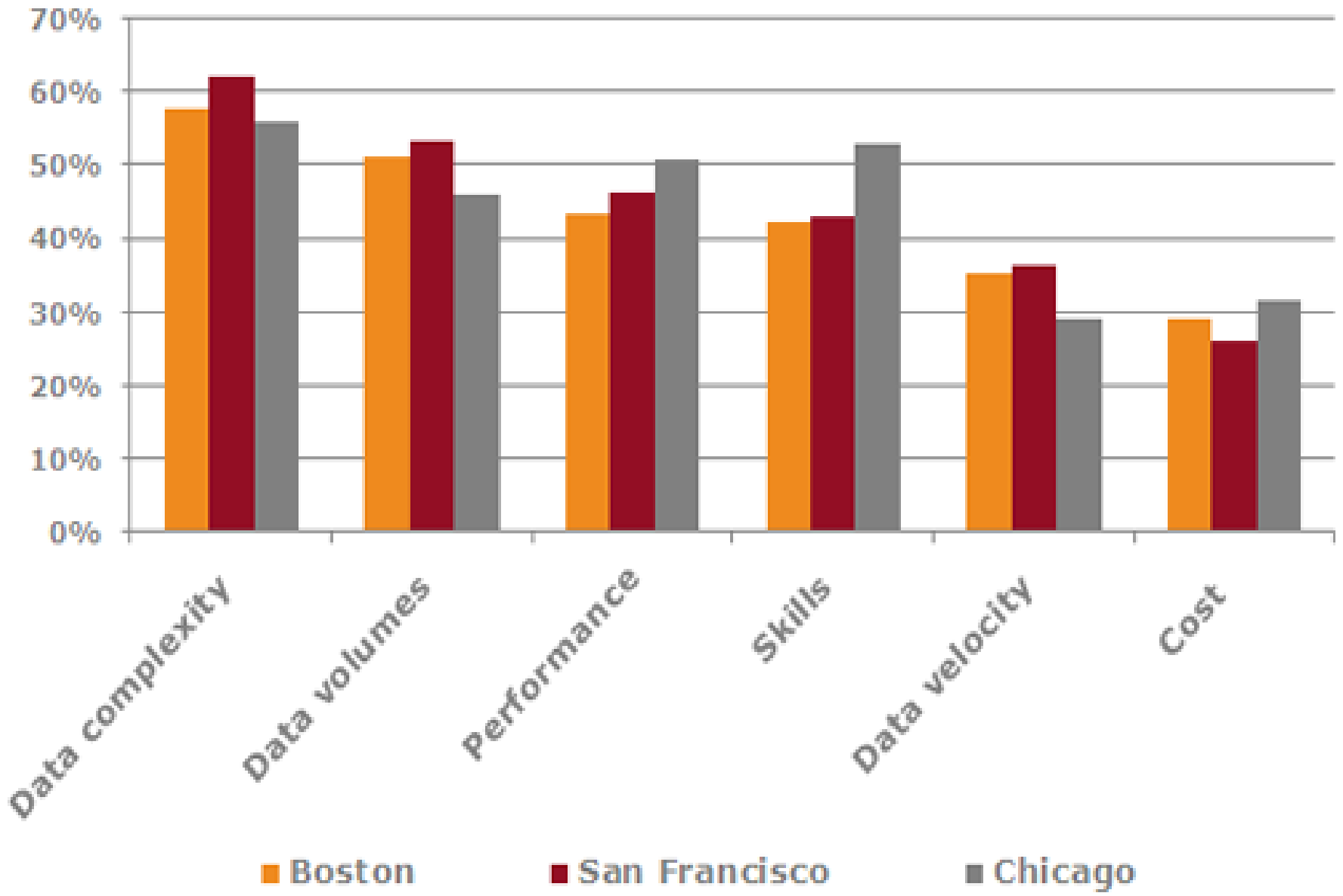


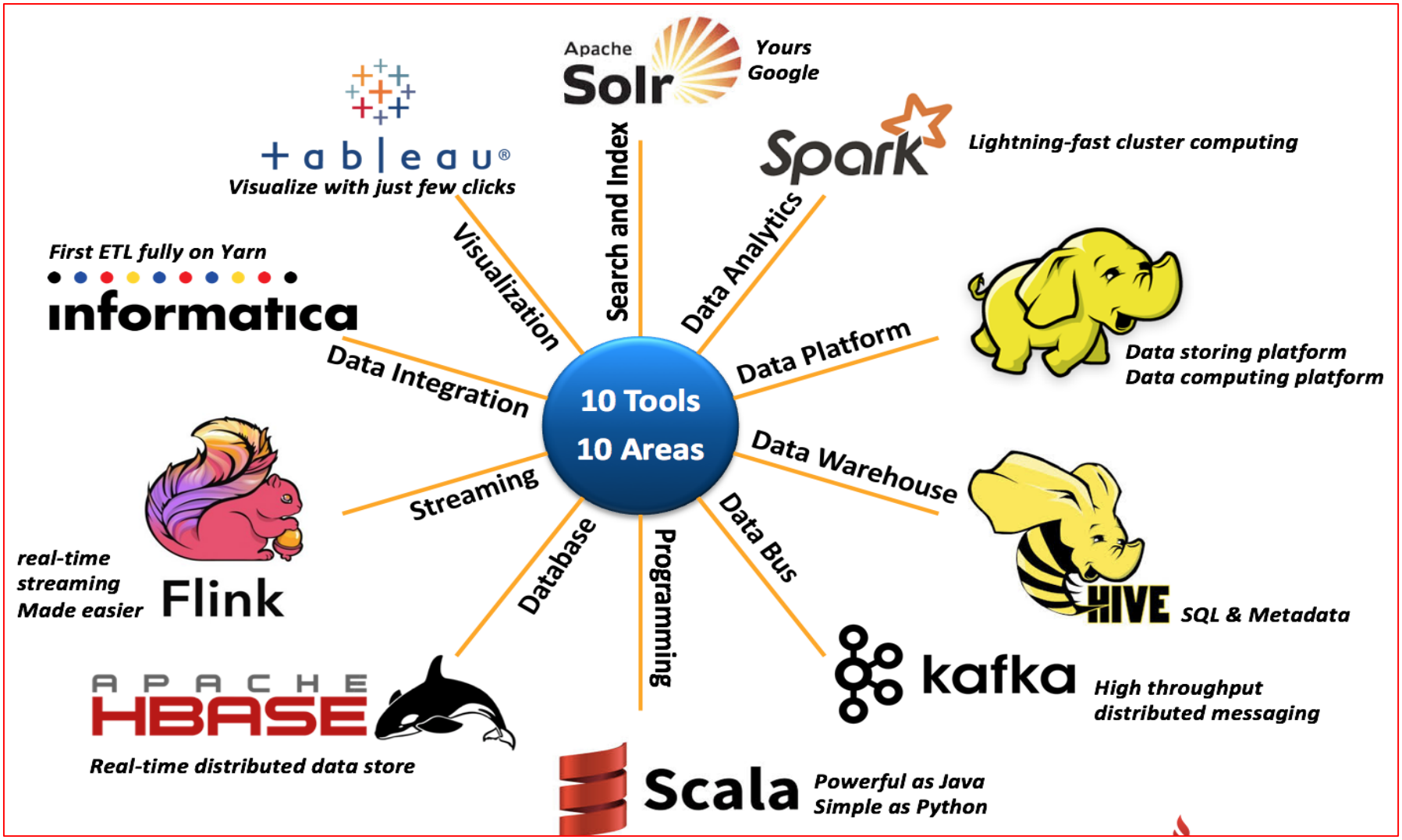
Fig: Challenges of Big Data

✓ Research issues in Big Data Analytics

1. Sentiment Analysis in Big Data Hadoop using Mahout Machine Learning Algorithms
2. Opinion mining Analysis in Big Data Hadoop using Mahout Machine Learning Algorithms
3. Predictive mining Analysis in Big Data Hadoop using Mahout Machine Learning Algorithms
4. Post-Clustering Analysis in Big Data Hadoop using Mahout Machine Learning Algorithms
5. Pre-Clustering Analysis in Big Data Hadoop using Mahout Machine Learning Algorithms
6. How we can capture and deliver data to right people in real-time
7. How we can handle variety of forms and data
8. How we can store and analyze data given its size and computational capacity.

Big Data Tools

- ✓ Big Data Tools are Hadoop, Cloudera, Datameer, Splunk, Mahout, Hive, HBase, LucidWorks, R, MapR, Ubuntu and Linux flavors.



Applications

- ✓ Social Networks and Relationships
- ✓ Cyber-Physical Models
- ✓ Internet of Things (IoT)
- ✓ Retail Market
- ✓ Retail Banking
- ✓ Real Estate
- ✓ Fraud detection and prevention
- ✓ Telecommunications
- ✓ Healthcare and Research
- ✓ Automotive and production
- ✓ Science and Research
- ✓ Trading Analytics

Fig: Applications of Big Data Analytics

Contents

- ✓ Spark Basics
- ✓ RDD(Resilient Distributed Dataset)
- ✓ Spark Transformations
- ✓ Spark Actions
- ✓ Spark with Machine Learning
- ✓ Hands on Spark
- ✓ Research Challenges in Spark

- ✓ **Spark** is a free and open source **software** web application framework and domain-specific language written in Java.
- ✓ Spark is an alternative to other Java web application frameworks such as JAX-RS, Play framework and Spring MVC.

- ✓ Apache Spark is a general-purpose cluster in-memory computing system.
- ✓ Provides high-level APIs in Java, Scala and Python and an optimized engine that supports general execution graphs.
- ✓ Provides various high level tools like Spark SQL for structured data processing, MLlib for Machine Learning and more….

What is Spark?

Fast and **Expressive** Cluster Computing System Compatible with Apache Hadoop

Up to **10x** faster on disk, **100x** in memory

2-5x less code

Efficient

- General execution graphs
- In-memory storage

Usable

- Rich APIs in Java, Scala, Python
- Interactive shell

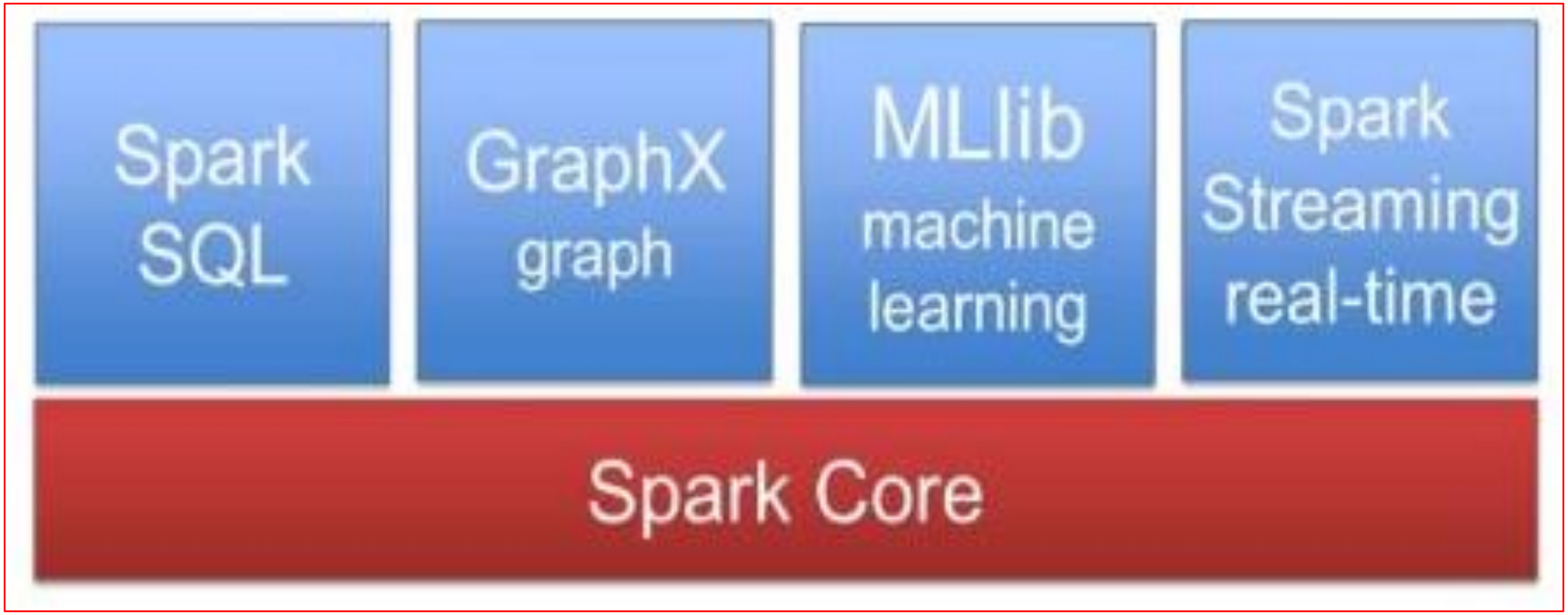
- ✓ Spark is a successor of MapReduce.
- ✓ Map Reduce is the ‘heart‘ of Hadoop that consists of two parts – ‘map’ and ‘reduce’.
- ✓ Maps and reduces are programs for processing data.
- ✓ ‘Map’ processes the data first to give some intermediate output which is further processed by ‘Reduce’ to generate the final output.
- ✓ Thus, MapReduce allows for distributed processing of the map and reduction operations.

- ✓ Apache Spark is the Next-Gen Big Data Tool i.e. Considered as future of Big Data and successor of MapReduce.

- ✓ Features of Spark are
  - Speed.
  - Usability.
  - In - Memory Computing.
  - Pillar to Sophisticated Analytics.
  - Real Time Stream Processing.
  - Compatibility with Hadoop & existing Hadoop Data.
  - Lazy Evaluation.
  - Active, progressive and expanding community.

- ✓ Spark is a Distributed data analytics engine, generalizing MapReduce.

- ✓ Spark is a core engine, with streaming, SQL, Machine Learning and Graph processing modules.



Precisely

- Alternative to MapReduce
- Compatibility with HDFS, HBASE and YARN
- Spark SQL component to access structured data
- Stable API
- Support for multiple languages
- Library support

Why Spark

- ✓ Spark is faster than MR when it comes to processing the same amount of data multiple times rather than unloading and loading new data.
- ✓ Spark is simpler and usually much faster than MapReduce for the usual Machine learning and Data Analytics applications.

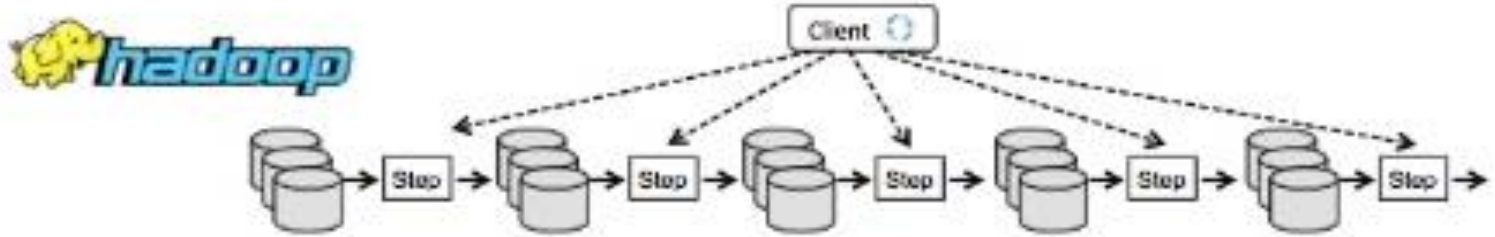
5 Reasons Why Spark Matters to Business

1. Spark enables use cases “traditional” Hadoop can’t handle.
2. Spark is fast
3. Spark can use your existing big data investment
4. Spark speaks SQL
5. Spark is developer-friendly

Why Spark is fast(er)

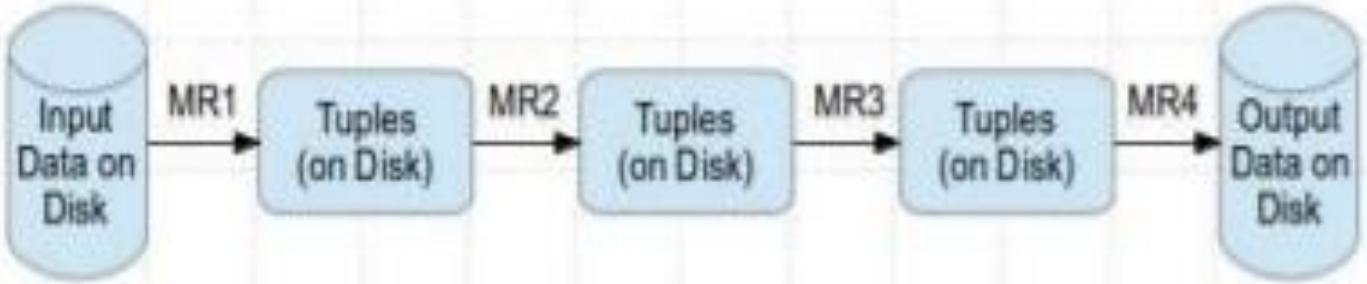
- Whom do we compare to?

- What do we mean by fast?
  - fast to write
  - fast to run



- Most of Machine Learning Algorithms are iterative because each iteration can improve the results
- With Disk based approach each iteration's output is written to disk making it slow

Hadoop execution flow



Spark execution flow

Speed

- Enables applications in Hadop clusters to run upto:
  - 100x faster in memory
  - 10X on disks

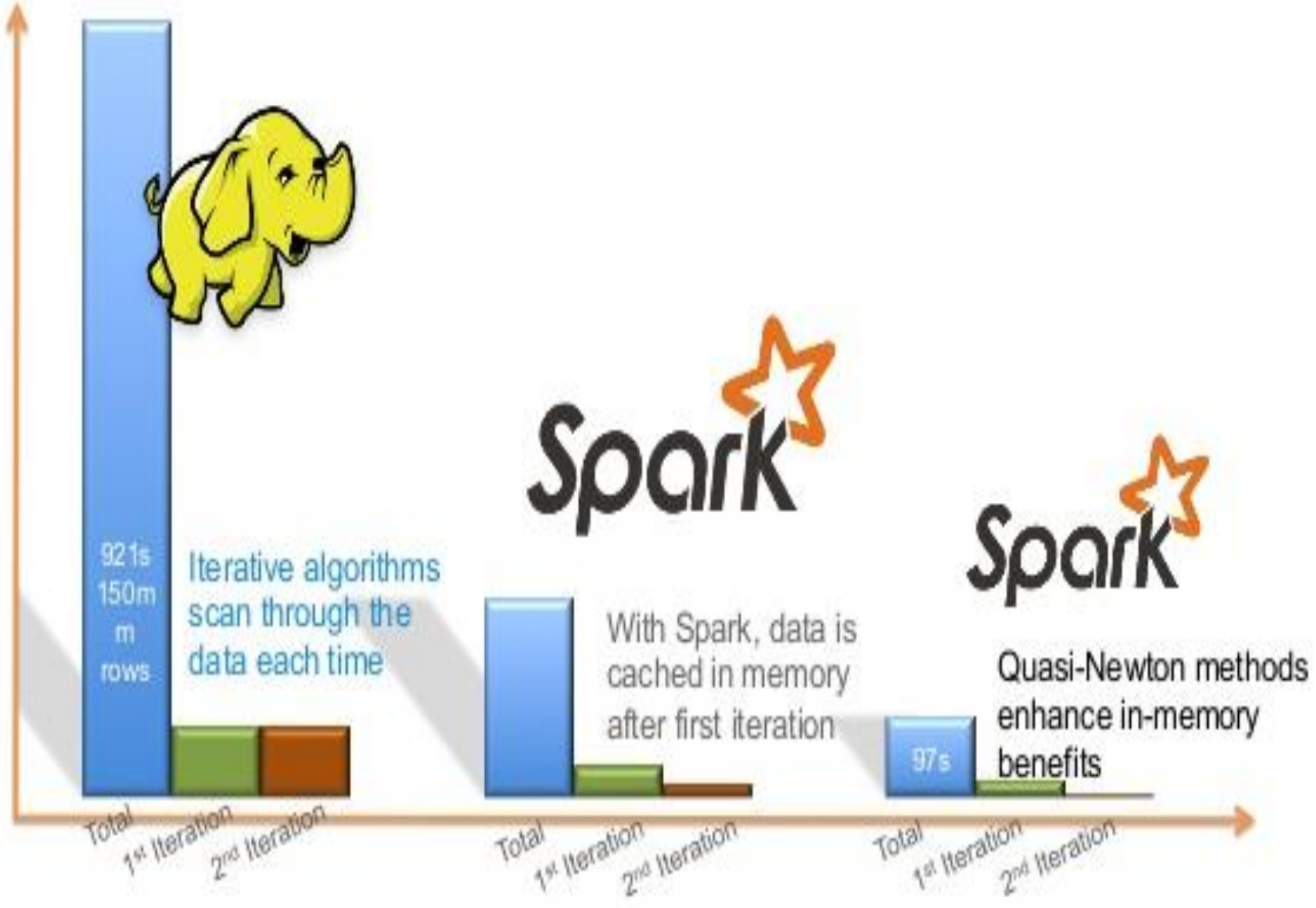
Ease of Use

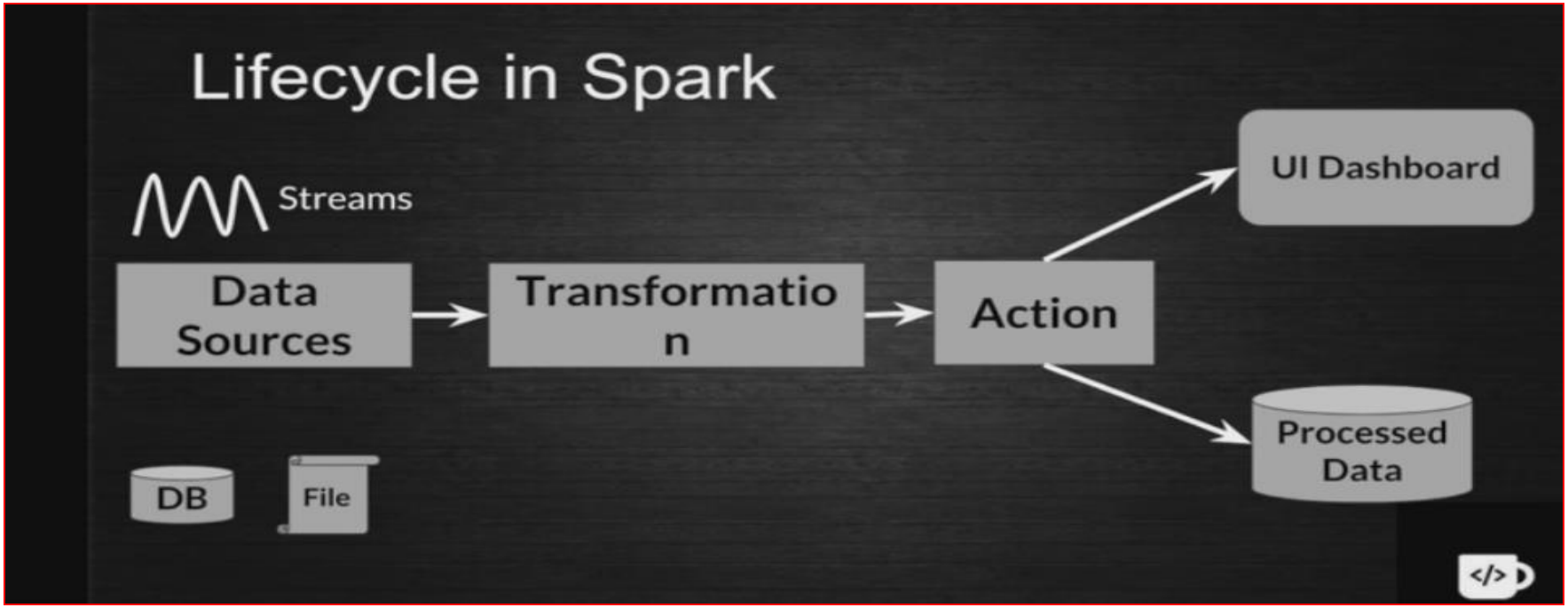
- Allows quickly write applications in Java, SCALA or Python

Sophisticated Analytics

- Supports SQL queries, streaming data and complex analytics

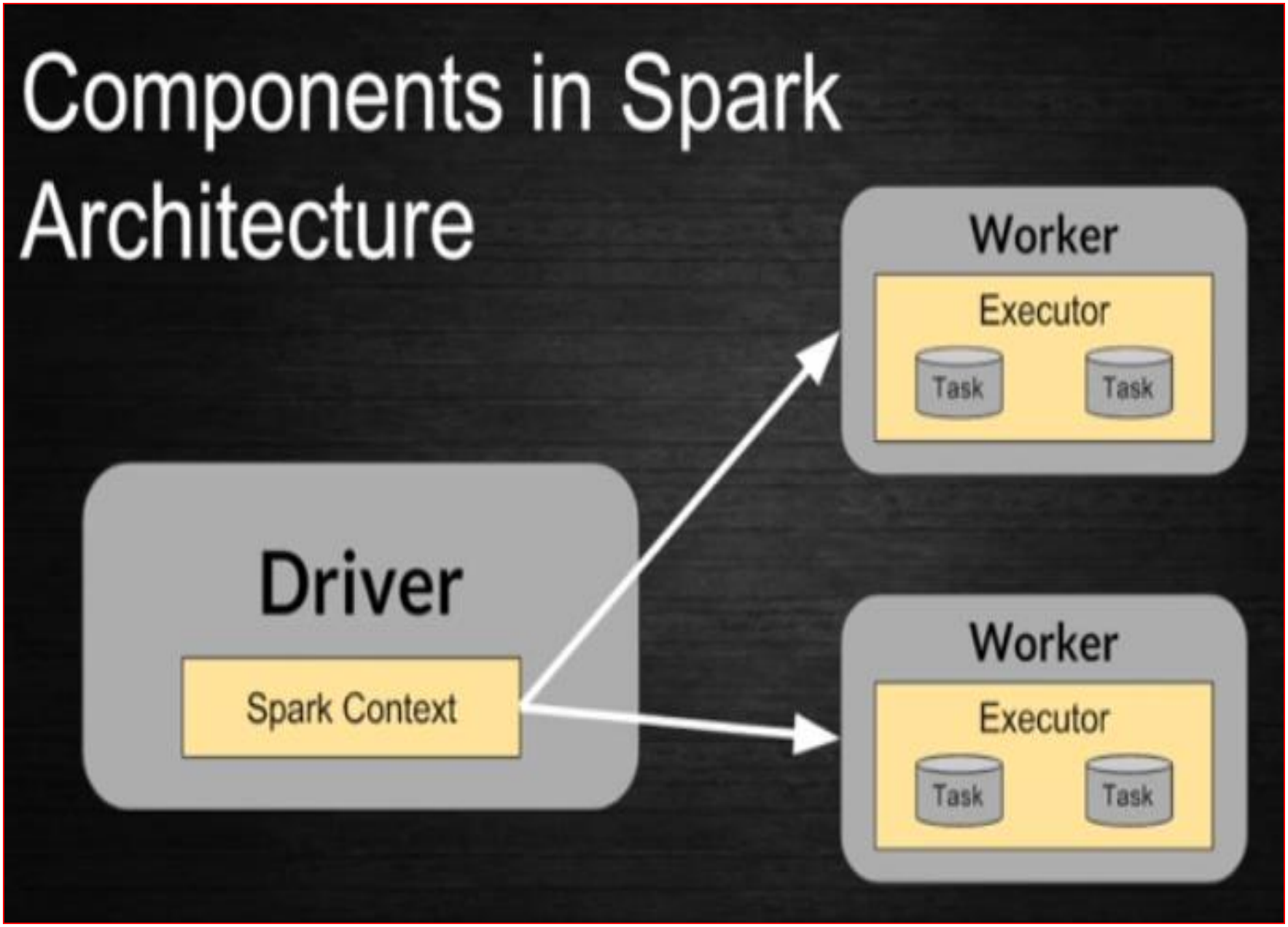
Apache Spark Utilizing in-memory Cache for M/R job

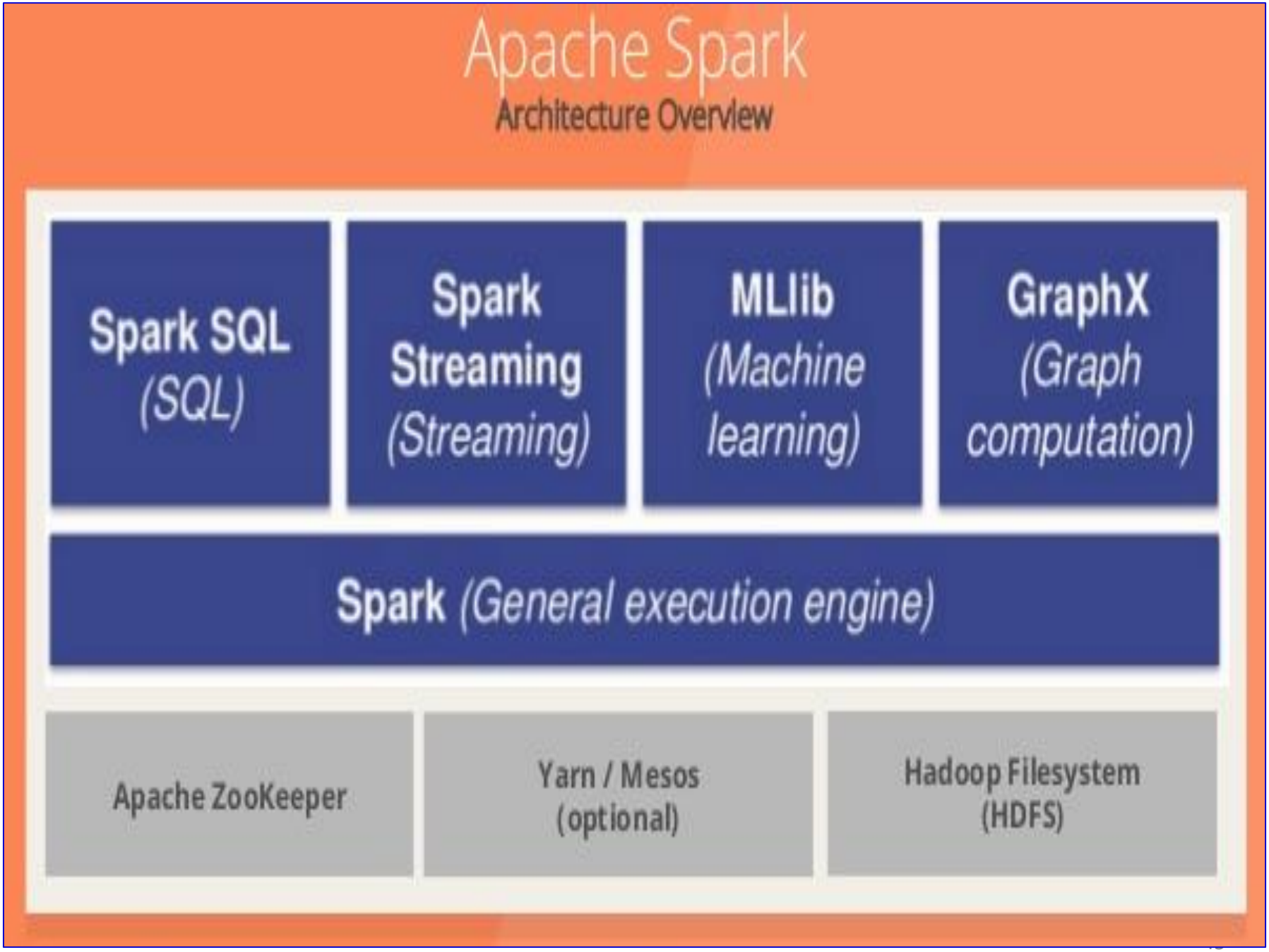




Lifecycle in Spark

- Load data on Cluster
- Create RDD
- Do Transformation
- Perform Action
- Create Data Frame
- Perform Queries on Data Frame
- Run SQL on Data Frames


Components in Spark Architecture
Worker
Executor
Task
Task
Driver
Spark Context
Worker
Executor
Task
Task


Apache Spark
Architecture Overview
Spark SQL
(SQL)
Spark
Streaming
(Streaming)
MLlib
(Machine
learning)
GraphX
(Graph
computation)
Spark (General execution engine)
Apache ZooKeeper
Yarn / Mesos
(optional)
Hadoop Filesystem
(HDFS)

✓ **Spark Ecosystem:** Spark Ecosystem is still in the stage of work-in-progress with Spark components, which are not even in their beta releases.

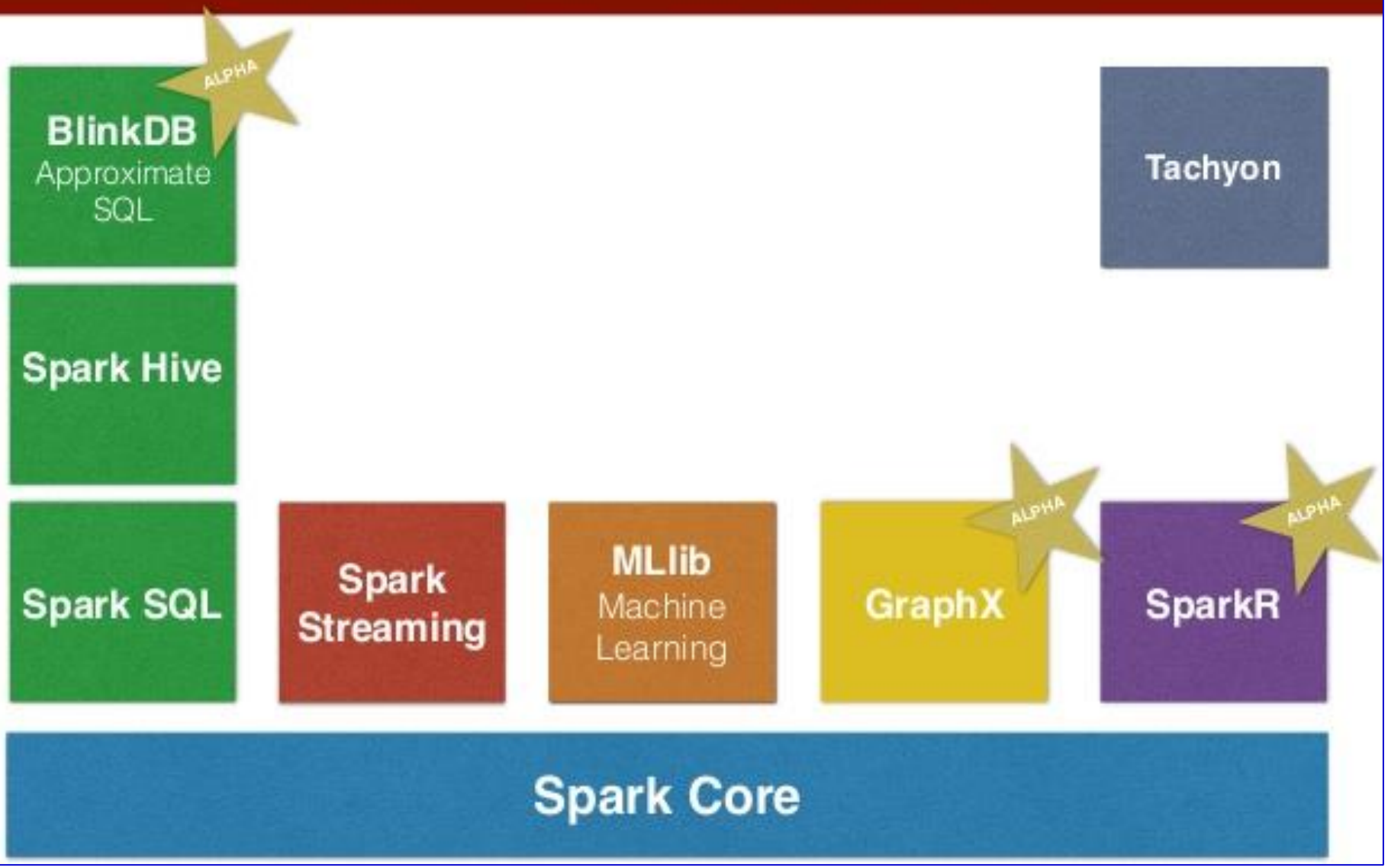
Components of Spark Ecosystem

✓ The components of Spark ecosystem are getting developed and several contributions are being made every now and then.

✓ Primarily, Spark Ecosystem comprises the following components:

1) Spark (SQL)
2) Spark Streaming (Streaming)
3) MLLib (Machine Learning)
4) GraphX (Graph Computation)
5) SparkR (R on Spark)
6) BlindDB (Approximate SQL)

Spark Ecosystem



- ✓ **Spark's official ecosystem consists of the following major components.**

- ✓ **Spark DataFrames** - Similar to a relational table
- ✓ **Spark SQL** - Execute SQL queries or HiveQL
- ✓ **Spark Streaming** - An extension of the core Spark API
- ✓ **MLlib** - Spark's machine learning library
- ✓ **GraphX** - Spark for graphs and graph-parallel computation
- ✓ **Spark Core API** - provides R, SQL, Python, Scala, Java

- ✓MLlib library has implementations for various common machine learning algorithms

1. Clustering: K-means
2. Classification: Naïve Bayes, logistic regression, SVM
3. Decomposition: Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
4. Regression : Linear Regression
5. Collaborative Filtering: Alternating Least Squares for Recommendations

Apache Spark Ecosystem

1 Language Support in Apache Spark

✓ Language Support in Apache Spark

- ✓ Apache Spark ecosystem is built on top of the core execution engine that has extensible API's in different languages.

- ✓ A recent 2016 Spark Survey on 62% of Spark users evaluated the Spark languages

- ✓ 58% were using Python in 2017
- ✓ 71% were using Scala
- ✓ 31% of the respondents were using Java and
- ✓ 18% were using R programming language.

- ✓ **What is Scala?:** Scala is a general-purpose programming language, which expresses the programming patterns in a concise, elegant, and type-safe way.

- ✓ It is basically an acronym for "Scalable Language".
- ✓ Scala is an easy-to-learn language and supports both Object Oriented Programming as well as Functional Programming.

- ✓ It is getting popular among programmers, and is being increasingly preferred over Java and other programming languages.

- ✓ It seems much in sync with the present and future Big Data frameworks, like Scalding, Spark, Akka, etc.

✓Why is Spark Programmed in Scala?

- ✓ Scala is a pure object-oriented language, in which conceptually every value is an object and every operation is a method-call. The language supports advanced component architectures through classes and traits.

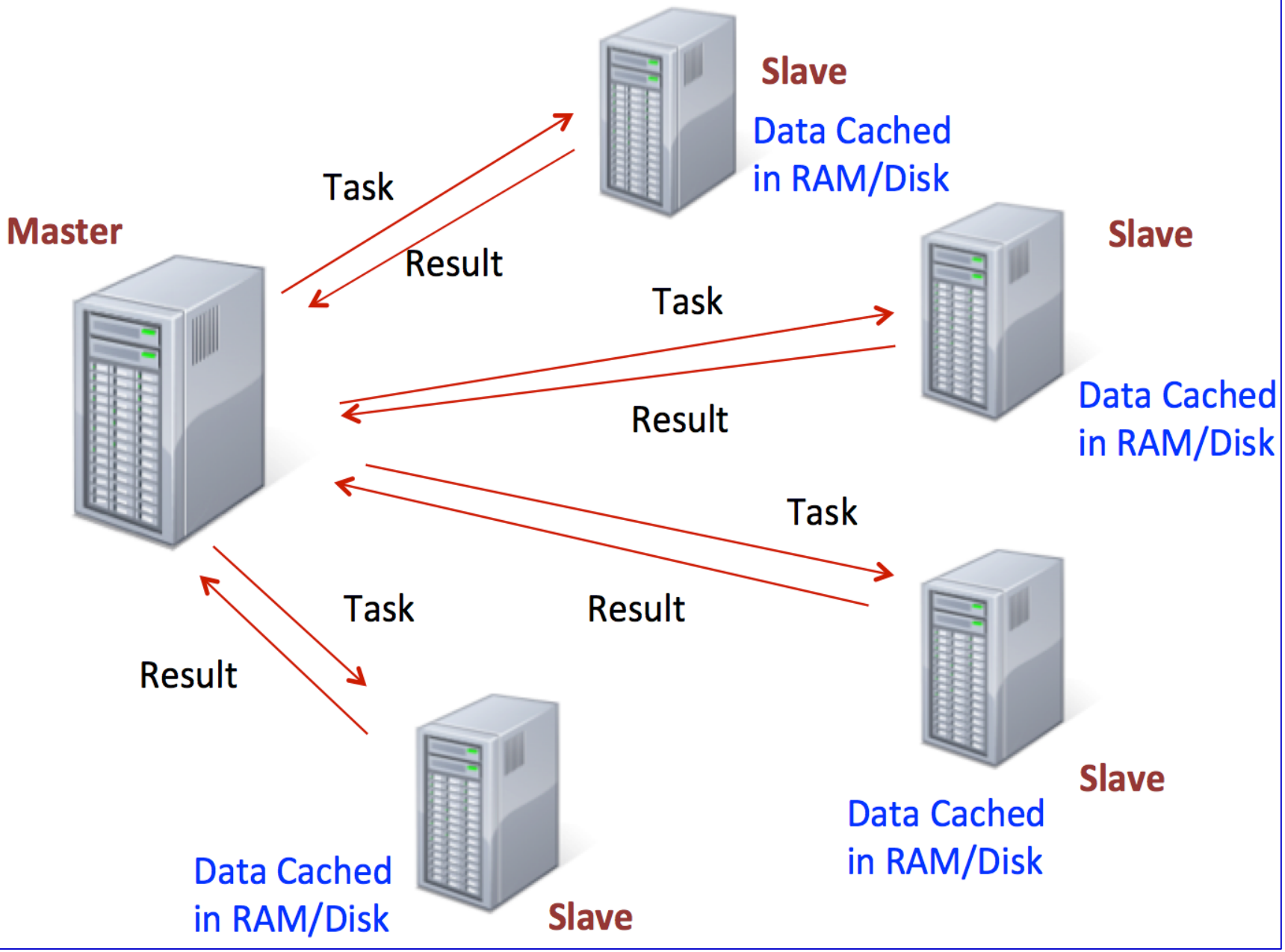
- ✓ Scala is also a functional language. It supports functions, immutable data structures and gives preference to immutability over mutation.

- ✓ Scala can be seamlessly integrated with Java

- ✓ It is already being widely used for Big Data platforms and development of frameworks like Akka, Scalding, Play, etc.

- ✓ Being written in Scala, Spark can be embedded in any JVM-based operational system.

How does Spark execute a job



✓ Procedure: **Spark Installation in Ubuntu**

✓ **Apache Spark** is a fast and general engine for large-scale data processing.

✓ Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.

✓ It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

Step 1: Installing Java.

java –version

Step 2: Installing Scala and SBT.

sudo apt-get update

sudo apt-get install scala

Step 3: Installing Maven plug-in.

- ✓ Maven plug-in is used to compile java program for spark. Type below command to install maven.
  sudo apt-get install maven

Step 4: Installing Spark.

- ✓ Download “tgz” file of spark by selecting specific version from below link *http://spark.apache.org/downloads.html*
- ✓ Extract it and remember its path where ever it stored.
- ✓ Edit .bashrc file by placing below lines (terminal command: *gedit .bashrc*)*export SPARK_HOME=/path_to_spark_directory*
  export PATH=$SPARK_HOME/bin:$PATH
- ✓ Replace path_to_spark_directory in above line with address of your spark directory.
- ✓ Restart .bashrc by saving and close it and type “..bashrc” in terminal.
- ✓ If it doesn’t work restart system. Thus we installed spark successfully.
- ✓ Type *spark-shell* in terminal to start spark shell.

Spark Installation on Windows

Step 1: Install Java (JDK)

Download and install java from https://java.com/en/download/

Step 2: Set java environment variable

- ✓ Open "control panel" and choose "system & security" and select "system".
- ✓ Select "Advanced System Settings" located at top right.
- ✓ Select "Environmental Variables" from pop-up.
- ✓ Next select *new* under system variables (below), you will get a pop-up.
- ✓ In variable name field type JAVA_HOME
- ✓ In variable value field provide installation directory of java, say *C:\Program Files\Java\jdk1.8.0_25*
- ✓ Or you can simply choose the directory by selecting browse directory.
- ✓ Now close everything by choosing ok every time.
- ✓ Check whether java variable is set or not by pinging *javac* in command promt. If we get java version details then we are done.

Step 3: Installing SCALA

- ✓ Download scala.msi file from https://www.scala-lang.org/download/
- ✓ Set scala environment variable just like java done above.
  Variable name = SCALA_HOME
  Variable value = path to scala installed directory, say C:\Program Files (x86)\scala

Step 4: Installing SPARK

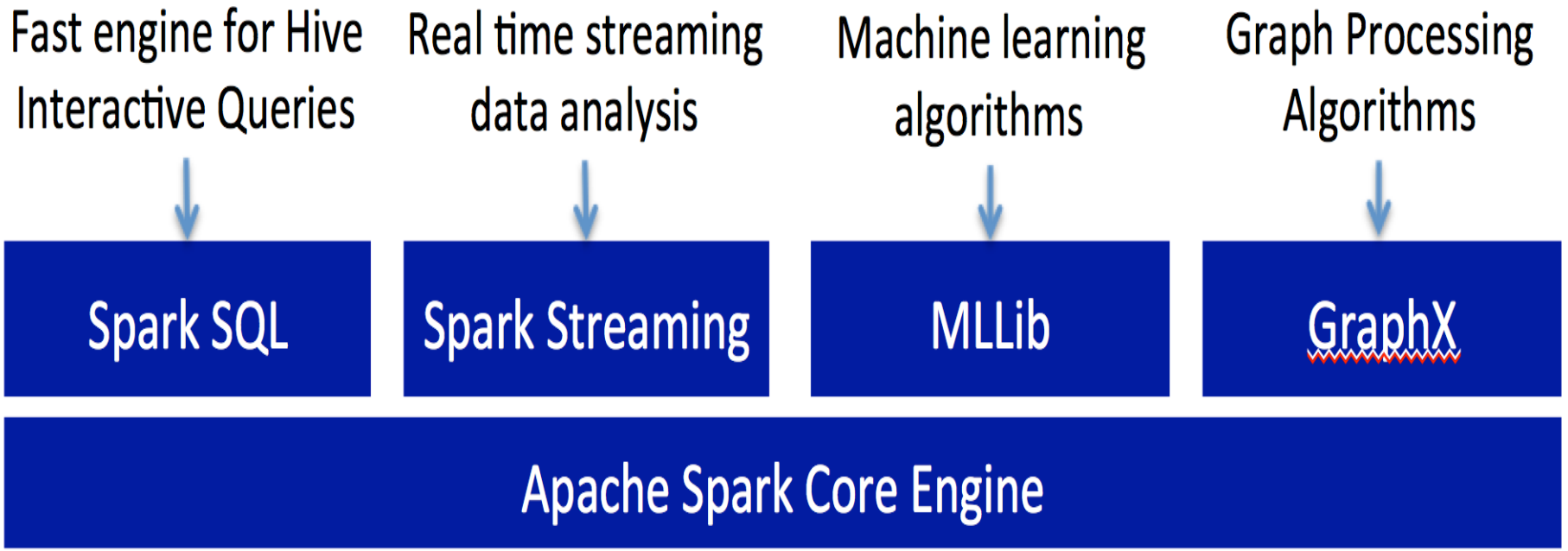
- ✓ Download and extract spark from *http://spark.apache.org/downloads.html*
- ✓ You can set SPARK_HOME just like java.
- ✓ Note:-We can only run spark-shell at bin folder in spark folder on windows.

Step 5: Installing SBT Download and install sbt.msi from http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Windows.html

Step 6: Installing Maven

- ✓ Download maven from http://maven.apache.org/download.cgi and unzip it to the folder you want to install Maven.
- ✓ Add both M2_HOME and MAVEN_HOME variables in the Windows environment, and point it to your Maven folder.
- ✓ Update PATH variable, append Maven bin folder – %M2_HOME%\bin, so that you can run the Maven's command everywhere.
- ✓ Test maven by pinging *mvn –version* in command prompt

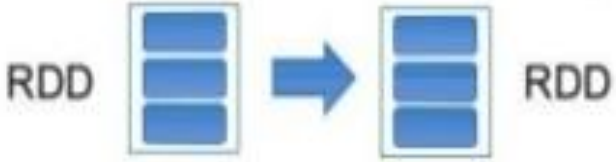
- ✓ **Practice on Spark Framework with Transformations and Actions:** You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos. Access data in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source.
- ✓ Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.



- ✓ RDD (Resilient Distributed Dataset) is main logical data unit in **Spark**. An RDD is distributed collection of objects. ... Quoting from Learning **Spark** book, "In **Spark** all work is expressed as creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.“

- ✓ Spark performs Transformations and Actions:

Transformations: define new RDDs based on current ones e.g. map, filter, join, union etc.



Actions: return a value (e.g. reduce, count, first etc)

Transformations	Actions
map (func)	reduce(func)
flatMap(func)	collect()
filter(func)	count()
groupByKey()	first()
reduceByKey(func)	take(n)
mapValues(func)	saveAsTextFile(path)
sample(...)	countByKey()
union(other)	foreach(func)
distinct()	...
sortByKey()	
...	

- ✓ Resilient Distributed Datasets overcome this drawback of Hadoop MapReduce by allowing - fault tolerant ‘in-memory’ computations.

- ✓ RDD in Apache Spark.
- ✓ Why RDD is used to process the data ?
- ✓ What are the major features/characteristics of RDD (Resilient Distributed Datasets) ?

- ✓ Resilient Distributed Datasets are immutable, partitioned collection of records that can be operated on - in parallel.

- ✓ RDDs can contain any kind of objects Python, Scala, Java or even user defined class objects.

- ✓ RDDs are usually created by either transformation of existing RDDs or by loading an external dataset from a stable storage like HDFS or HBase.

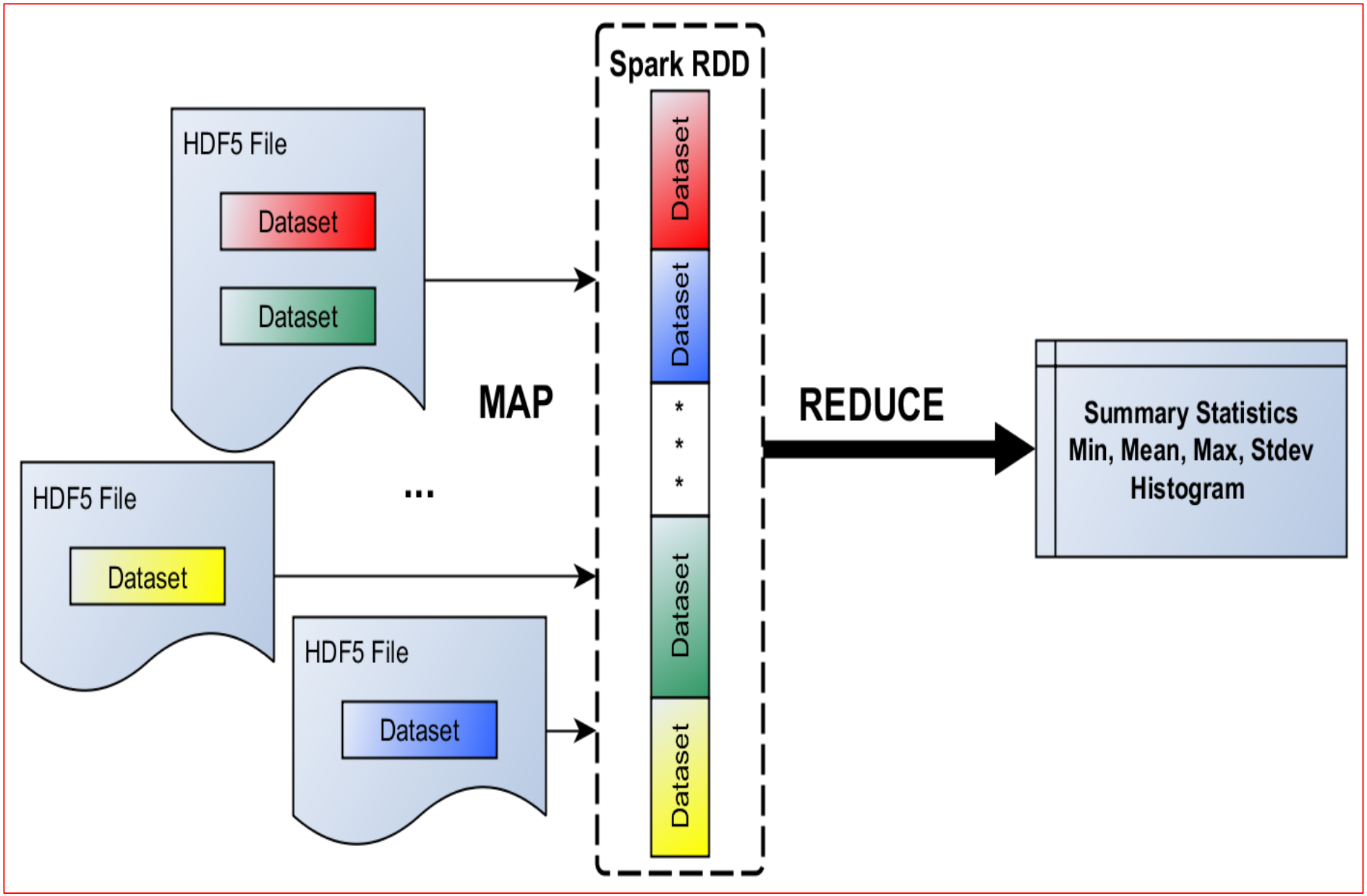


Fig: Process of RDD Creation

✓ Operations on RDDs

- ✓ **i) Transformations:** Coarse grained operations like join, union, filter or map on existing RDDs which produce a new RDD, with the result of the operation, are referred to as transformations. All transformations in Spark are lazy.

- ✓ **ii) Actions:** Operations like count, first and reduce which return values after computations on existing RDDs are referred to as Actions.

- **Properties / Traits of RDD:**

✓ **Immutable (Read only cant change or modify):** Data is safe to share across processes.

✓ **Partitioned:** It is basic unit of parallelism in RDD.
✓ **Coarse gained operations:** it's applied to any or all components in datasets through maps or filter or group by operation.

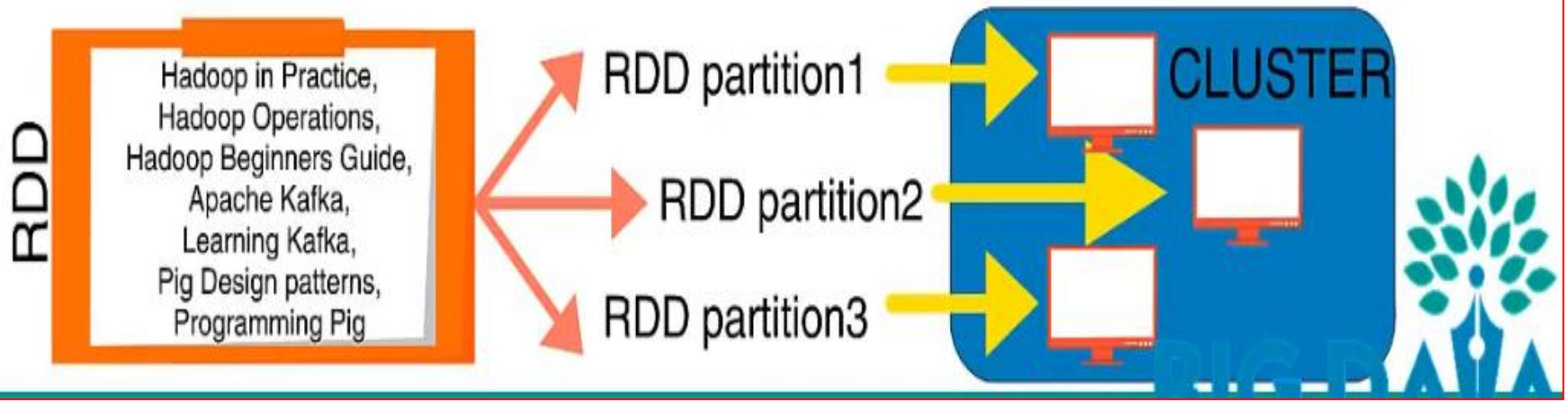
✓ **Action/Transformations:** All computations in RDDs are actions or transformations.
✓ **Fault Tolerant:** As the name says or include Resilient which means its capability to reconcile, recover or get back all the data using lineage graph.

✓ **Cacheable:** It holds data in persistent storage.
✓ **Persistence:** Option of choosing which storage will be used either in-memory or on-disk.

What is RDD ?

WHAT IS RDD ?

- ✓ RDD is the spark's core abstraction which is resilient distributed dataset
- ✓ It is the immutable distributed collection of objects
- ✓ Internally spark distributes the data in RDD, to different nodes across the cluster to achieve parallelization



Transformations

- Create a new dataset from an existing one.
- Lazy in nature, executed only when some action is performed.
- Example
  - Map(func)
  - Filter(func)
  - Distinct()

Actions

- Returns a value or exports data after performing a computation.
- Example:
  - Count()
  - Reduce(func)
  - Collect
  - Take()

Persistence

- Caching dataset in-memory for future operations
- store on disk or RAM or mixed
- Example:
  - Persist()
  - Cache()

How Spark Works - RDD Operations

- ✓ **Task 1:** Practice on Spark Transformations i.e. map(), filter(), flatmap(), groupBy(), groupByKey(), sample(), union(), join(), distinct(), keyBy(), partitionBy and zip().

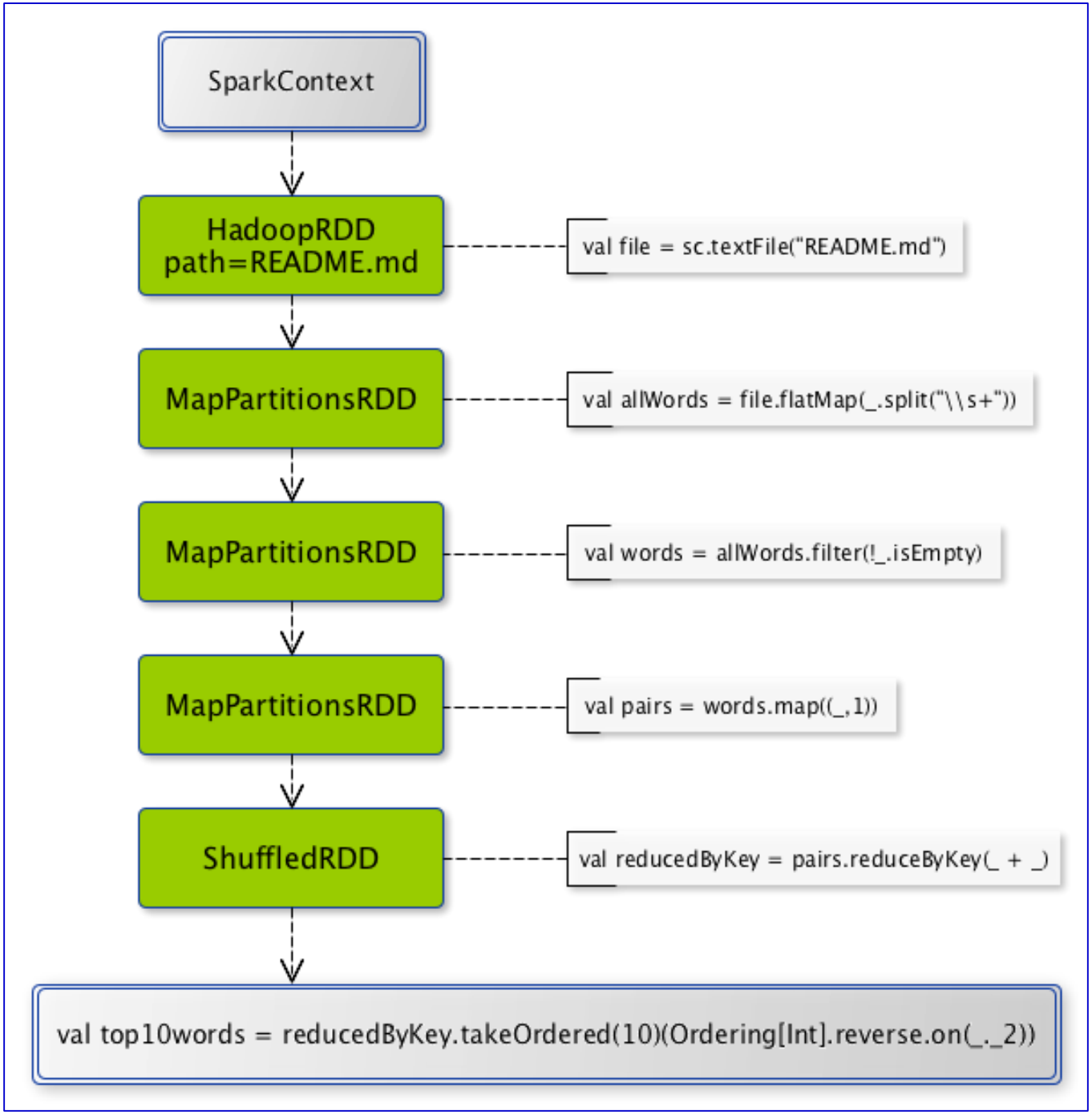
- ✓ RDD (Resilient Distributed Dataset) is main logical data unit in **Spark**. An RDD is distributed collection of objects. ... Quoting from Learning **Spark** book, "In **Spark** all work is expressed as creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.“

- ✓ Transformations are lazy evaluated operations on RDD that create one or many new RDDs, e.g. map, filter, reduceByKey, join, cogroup, randomSplit.

- ✓ Transformations are lazy, i.e. are not executed immediately.

- ✓ Transformations can be executed only when actions are called.

- ✓ Transformations are lazy operations on a RDD that create one or many new RDDs.
- ✓ Ex: map , filter , reduceByKey , join , cogroup , randomSplit .
- ✓ In other words, transformations are functions that take a RDD as the input and produce one or many RDDs as the output.
- ✓ RDD allows you to create dependencies between RDDs.
- ✓ Dependencies are the steps for producing results i.e. a program.
- ✓ Each RDD in lineage chain, string of dependencies has a function for operating its data and has a pointer dependency to its ancestor RDD.
- ✓ Spark will divide RDD dependencies into stages and tasks and then send those to workers for execution.


SparkContext
HadoopRDD
path=README.md
val file = sc.textFile("README.md")
MapPartitionsRDD
val allWords = file.flatMap(_.split("\\s+"))
MapPartitionsRDD
val words = allWords.filter(!_.isEmpty)
MapPartitionsRDD
val pairs = words.map((_,1))
ShuffledRDD
val reducedByKey = pairs.reduceByKey(_ + _)
val top10words = reducedByKey.takeOrdered(10)(Ordering[Int].reverse.on(_._2))

1.map(): Pass each element of the RDD through the supplied function.

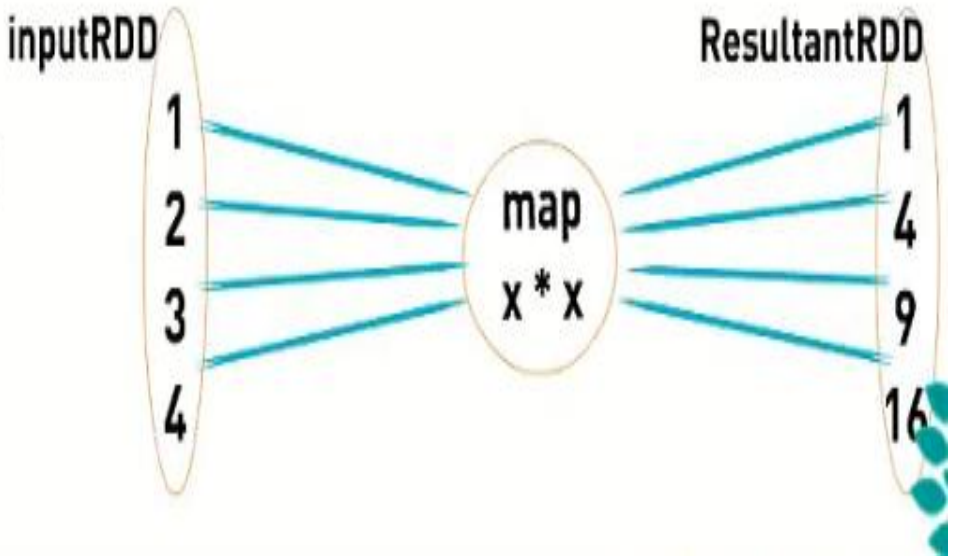
```
val  x = Array("b", "a", "c")
val  y = x.map(z => (z,1))
```

Output: y: [('b', 1), ('a', 1), ('c', 1)]

Map

map() is the transformation that takes a function and applies the function to each element of the input RDD.

The result of the function , will become the value of each element in the resultant RDD.

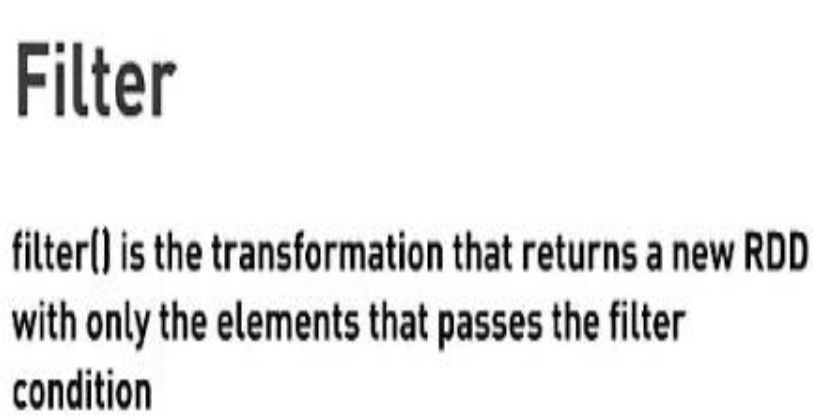


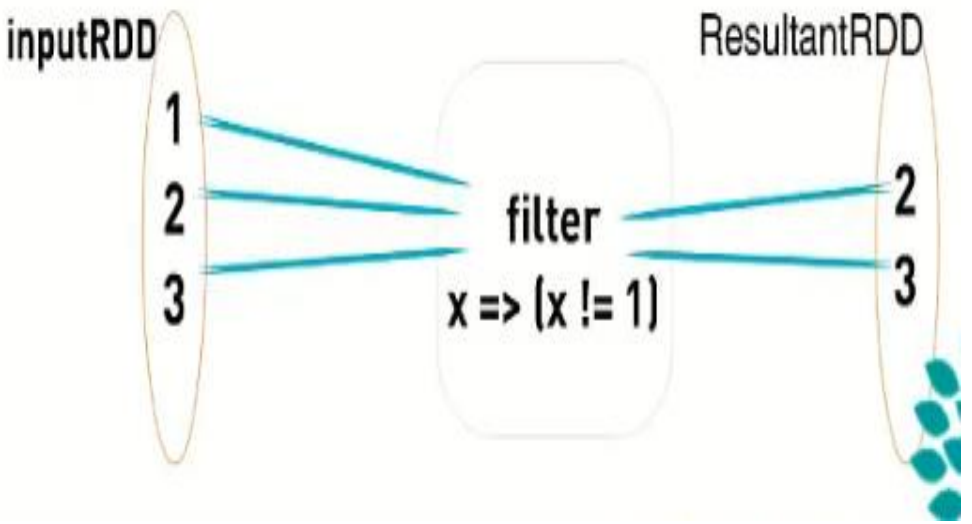
2.filter(): Filter creates a new RDD by passing in the supplied function used to filter the results.

```
val x = sc.parallelize(Array(1,2,3))
val y = x.filter(n => n%2 == 1)

println(y.collect().mkString(", "))
```

Output: y: [1, 3]



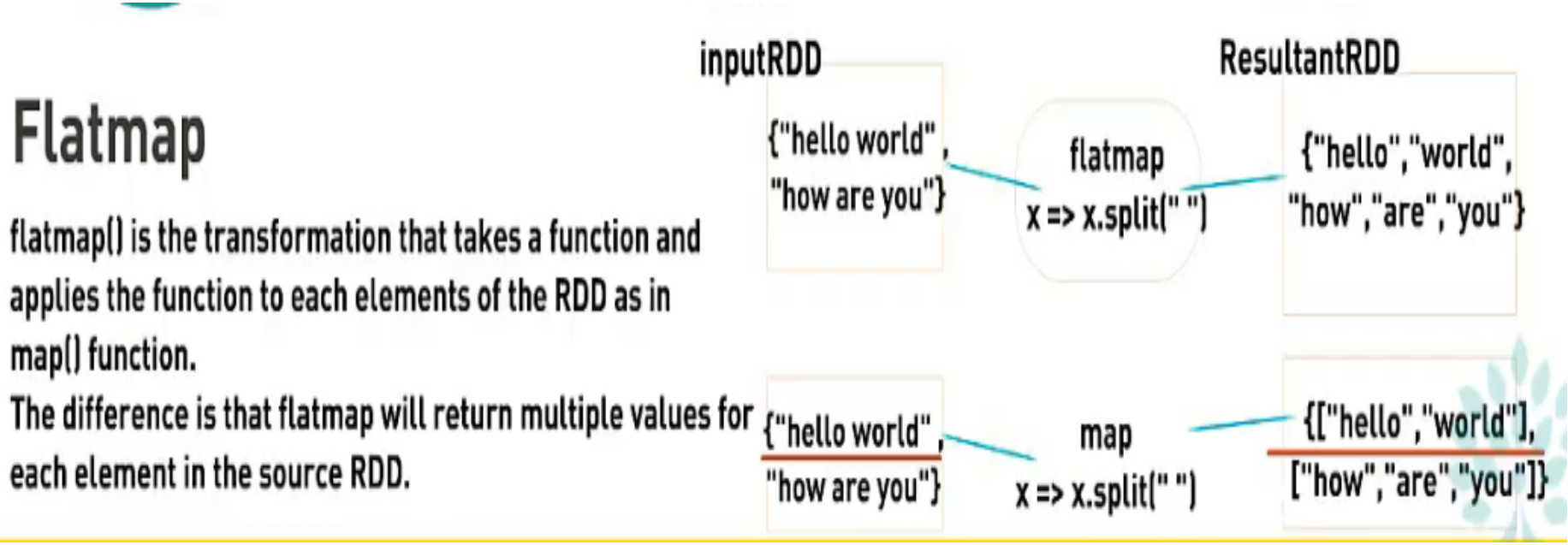


3.flatmap() : Similar to map, but each input item can be mapped to 0 or more output items.

```
val x = sc.parallelize(Array(1,2,3))
val y = x.flatMap(n => Array(n, n*100, 42))

println(y.collect().mkString(", "))
```

Output: y: [1, 100, 42, 2, 200, 42, 3, 300, 42]



4.groupBy() : When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

```
val x = sc.parallelize( Array("John", "Fred", "Anna", "James"))
val y = x.groupBy(w => w.charAt(0))

println(y.collect().mkString(", "))
```

Output: y: [('A',['Anna']),('J',['John','James']),('F',['Fred'])]

5.groupByKey() : When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.

```
val x = sc.parallelize( Array(('B',5),('B',4),('A',3),('A',2),('A',1)))
val y = x.groupByKey()

println(y.collect().mkString(", "))
```

Output: y: [('A', [3, 2, 1]),('B',[5, 4])]

6.sample() : Return a random sample subset RDD of the input RDD.

```
val x= sc.parallelize(Array(1, 2, 3, 4, 5))
val y= x.sample(false, 0.4)

// omitting seed will yield different output

println(y.collect().mkString(", "))
```

Output: y: [1, 3]

7.union() : Simple. Return the union of two RDDs.

```
val x= sc.parallelize(Array(1,2,3), 2)
val y= sc.parallelize(Array(3,4), 1)
val z= x.union(y)
val zOut= z.glom().collect()
```

Output z: [[1], [2, 3], [3, 4]]

8.join() : If you have relational database experience, this will be easy. It's joining of two datasets.

```
val x= sc.parallelize(Array(("a", 1), ("b", 2)))
val y= sc.parallelize(Array(("a", 3), ("a", 4), ("b", 5)))
val z= x.join(y)

println(z.collect().mkString(", "))
```

Output z: [('a', (1, 3)), ('a', (1, 4)), ('b', (2, 5))]

9.distinct() : Return a new RDD with distinct elements within a source RDD.

```
val x= sc.parallelize(Array(1,2,3,3,4))
val y= x.distinct()

println(y.collect().mkString(", "))
```

Output: y: [1, 2, 3, 4]

10) keyBy() : Constructs two-component tuples (key-value pairs) by applying a function on each data item.

```
val x= sc.parallelize(Array("John", "Fred", "Anna", "James"))
val y= x.keyBy(w => w.charAt(0))

println(y.collect().mkString(", "))
```

Output: y: [('J','John'),('F','Fred'),('A','Anna'),('J','James')]

11) partitionBy() : Repartitions as key-value RDD using its keys. The partitioner implementation can be supplied as the first argument.

```
import org.apache.spark.partiotioner

val x=sc.parallelize(Array(‘J’,”James”),(‘F’,”Fred”),(‘A’,”Anna”),(‘J’,”John”),3)
val y= x.partitionBy(new Partitioner() { val numPartitions= 2
defgetPartition(k:Any) = {
if (k.asInstanceOf[Char] < 'H') 0 else 1
}
})

val yOut= y.glom().collect()
```

Output:y: Array(Array((F,Fred), (A,Anna)), Array((J,John), (J,James)))

12) zip() : Joins two RDDs by combining the i-th of either partition with each other.

```
val x= sc.parallelize(Array(1,2,3))
val y= x.map(n=>n*n)
val z= x.zip(y)

println(z.collect().mkString(", "))
```

Output: z: [(1, 1), (2, 4), (3, 9)]

- ✓ Task 2: Practice on Spark Actions i.e. getNumPartitions(), collect(), reduce(), aggregate(), max(), sum(), mean(), stdev(), countByKey().

- ✓ RDD (Resilient Distributed Dataset) is main logical data unit in **Spark**. An RDD is distributed collection of objects. ... Quoting from Learning **Spark** book, "In **Spark** all work is expressed as creating new RDDs, transforming existing RDDs, or calling operations on RDDs to compute a result.“
- ✓ Actions returns final result of RDD computations / operation.

- ✓ Action produces a value back to the Spark driver program. It may trigger a previously constructed, lazy RDD to be evaluated.

- ✓ Action function materialize a value in a Spark program. So basically an action is RDD operation that returns a value of any type but RDD[T] is an action.

- ✓ **Actions:** Unlike Transformations which produce RDDs, action functions produce a value back to the Spark driver program.
- ✓ Actions may trigger a previously constructed, lazy RDD to be evaluated.

1. collect()
2. reduce()
3. aggregate ()
4. mean()
5. sum()
6. max()
7. stdev()
8. countByKey()
9. getNumPartitions()

1) collect() : collect returns the elements of the dataset as an array back to the driver program.

```
val x= sc.parallelize(Array(1,2,3), 2)

val y= x.collect()
```

Output: y: [1, 2, 3]

2) reduce() : Aggregate the elements of a dataset through *function*.

```
val x= sc.parallelize(Array(1,2,3,4))

val y= x.reduce((a,b) => a+b)
```

Output: y: 10

3) aggregate(): The aggregate function allows the user to apply two different reduce functions to the RDD.

```
val inputrdd=sc.parallelize(List(("maths",21),("english",22), ("science",31)),3)

valresult=inputrdd.aggregate(3)((acc,value)=>(acc+value._2),(acc1,acc2)=>(acc1+acc2))
```

Partition 1 : Sum(all Elements) + 3 (Zero value)
Partition 2 : Sum(all Elements) + 3 (Zero value)
Partition 3 : Sum(all Elements) + 3 (Zero value)

✓ Result = Partition1 + Partition2 + Partition3 + 3(Zero value)

So we get 21 + 22 + 31 + (4 * 3) = 86

✓ Output: y: Int = 86

4) max() : Returns the largest element in the RDD.

```
val x= sc.parallelize(Array(2,4,1))

val y= x.max()
```

Output: y: 4

5) count() : Number of elements in the RDD.

```
val x= sc.parallelize(Array("apple", "beatty", "beatrice"))

val y=x.count()
```

Output: y: 3

6) sum() : Sum of the RDD.

```
val x= sc.parallelize(Array(2,4,1))

val y= x.sum()
```

Output: y: 7

7) mean() : Mean of given RDD.

```
val x= sc.parallelize(Array(2,4,1))

val y= x.mean()
```

Output: y: 2.3333333

8) stdev() : An aggregate function that standard deviation of a set of numbers.

```
val x= sc.parallelize(Array(2,4,1))

val y= x.stdev()
```

Output: y: 1.2472191

9) countByKey() : This is only available on RDDs of (K
returns a hashmap of (K, count of K).

```
val x= sc.parallelize(Array(('J',"James"),('F',"Fred"), ('A',"Anna"),('J',"Jo
```

```
val y= x.countByKey()
```

Output: y: {'A': 1, 'J': 2, 'F': 1}

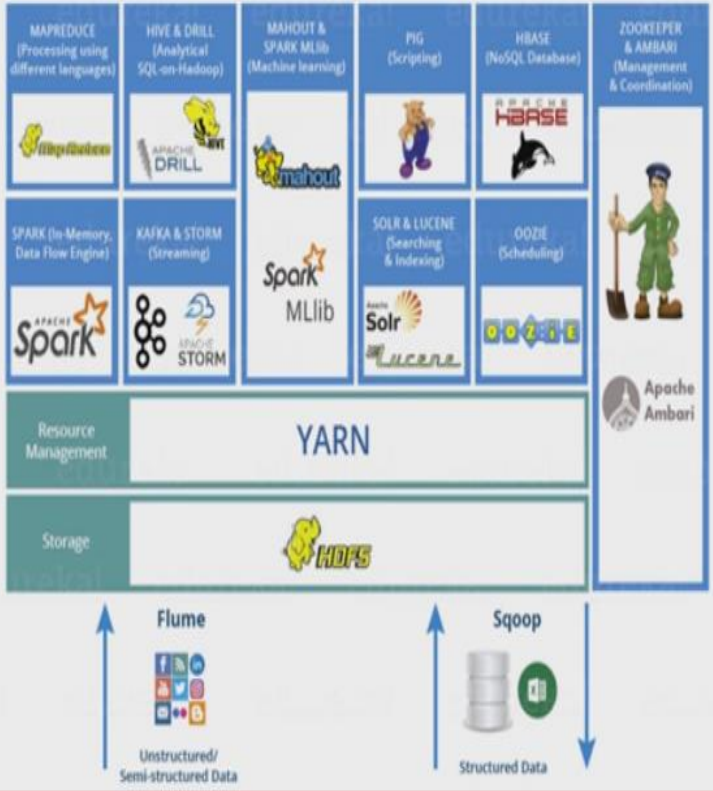
10) getNumPartitions()

val **x= sc.parallelize(Array(1,2,3), 2)**

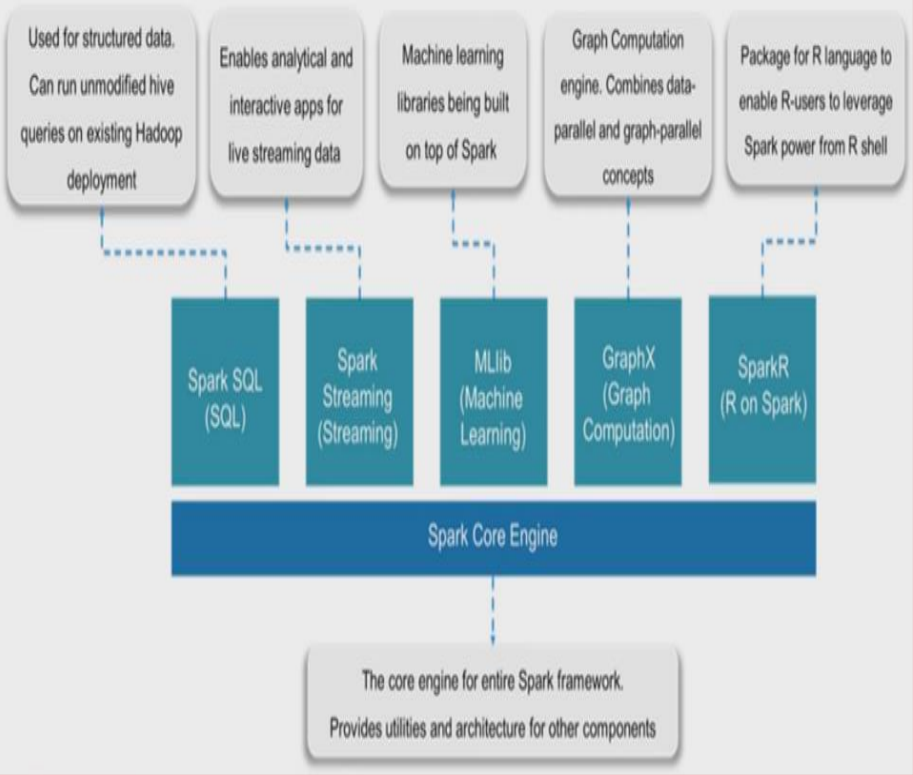
val y= x.partitions.size

Output: y: 2

Hadoop



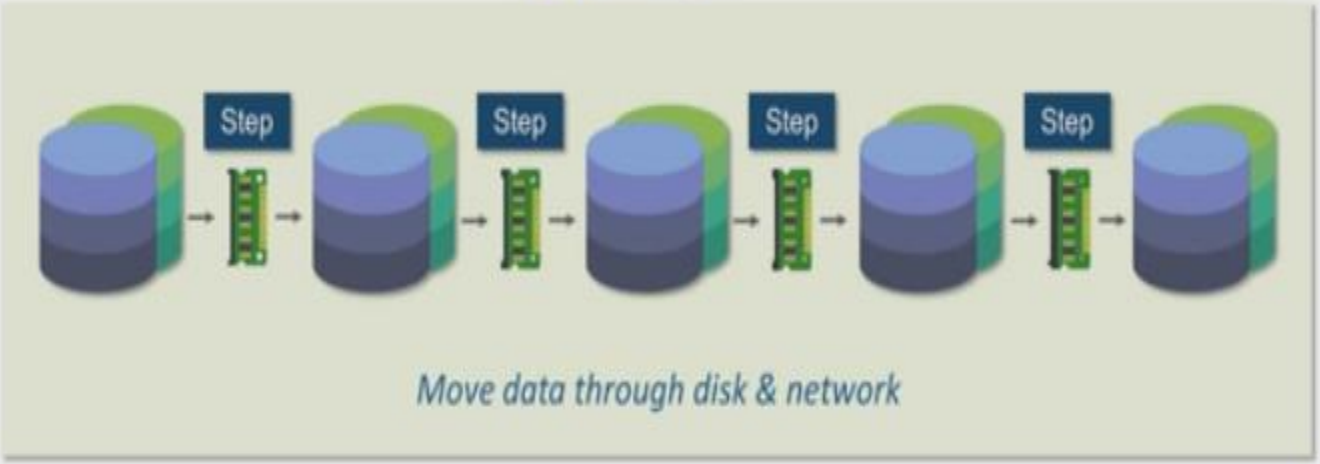
Spark

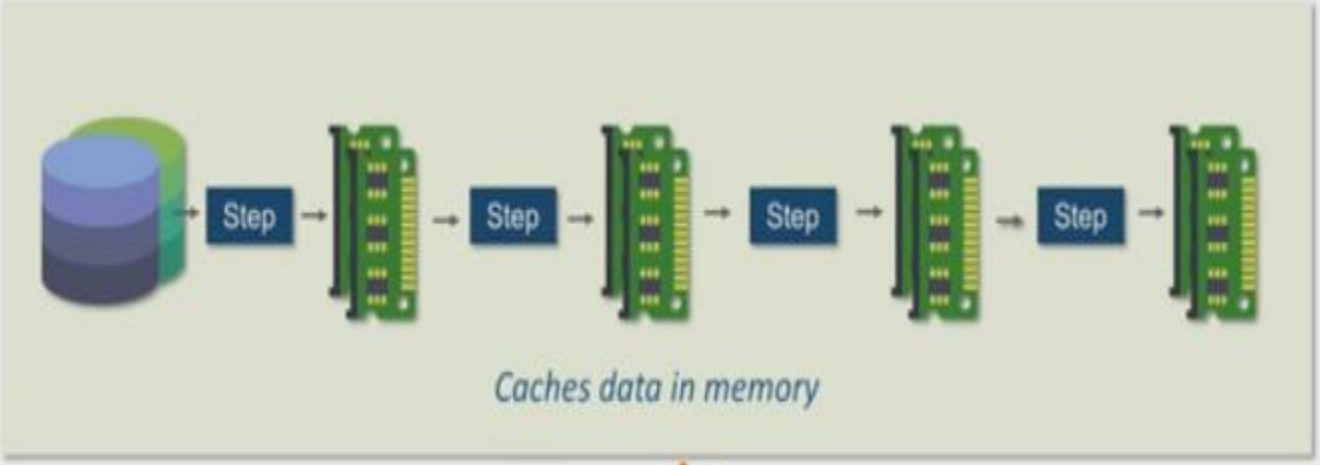


Performance

- Performance
- Ease of Use
- Costs
- Data Processing
- Fault Tolerance
- Security

hadoop





APACHE Spark

Ease of Use

- Performance
- Ease of Use
- Costs
- Data Processing
- Fault Tolerance
- Security

Costs

- Performance
- Ease of Use
- Costs
- Data Processing
- Fault Tolerance
- Security

hadoop

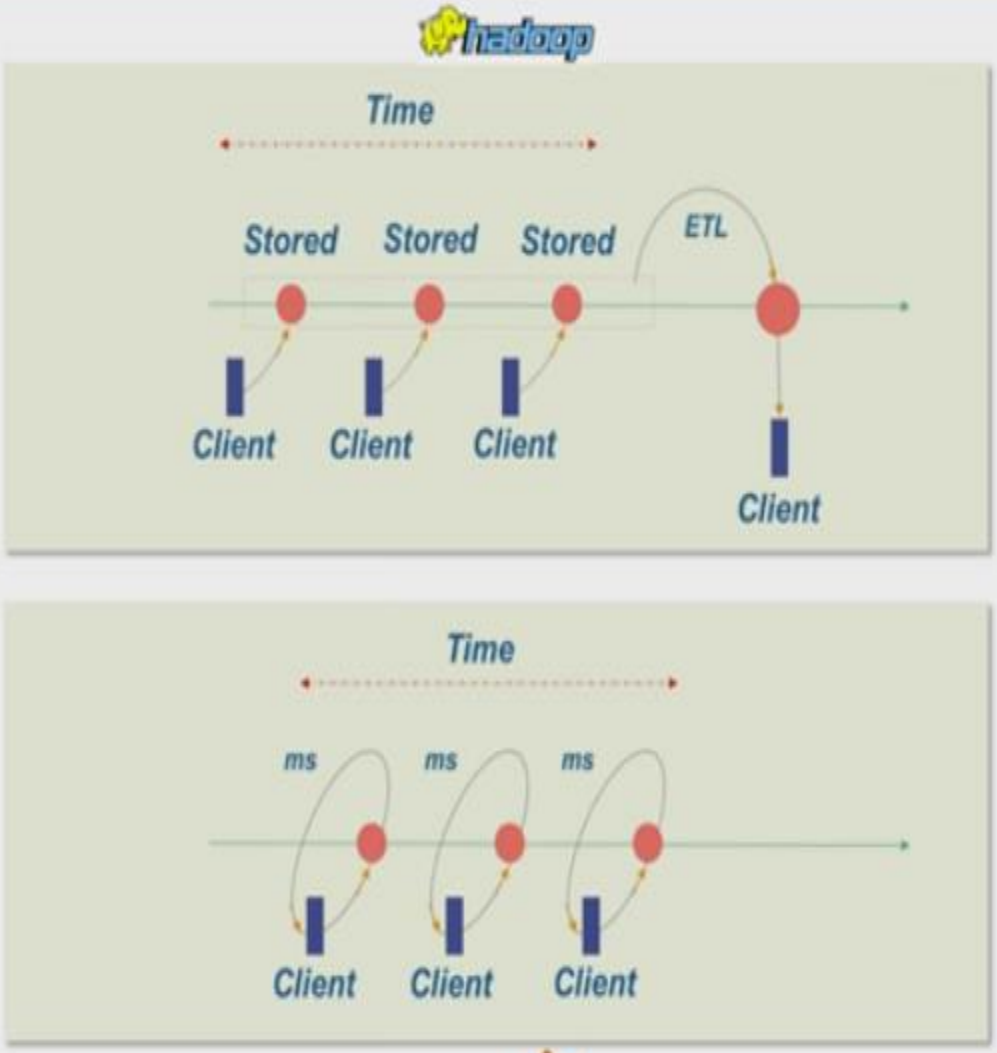
Hadoop requires lot of disk space as well as faster disks

Spark requires large amounts of RAM for executing everything in memory

APACHE Spark

Data Processing

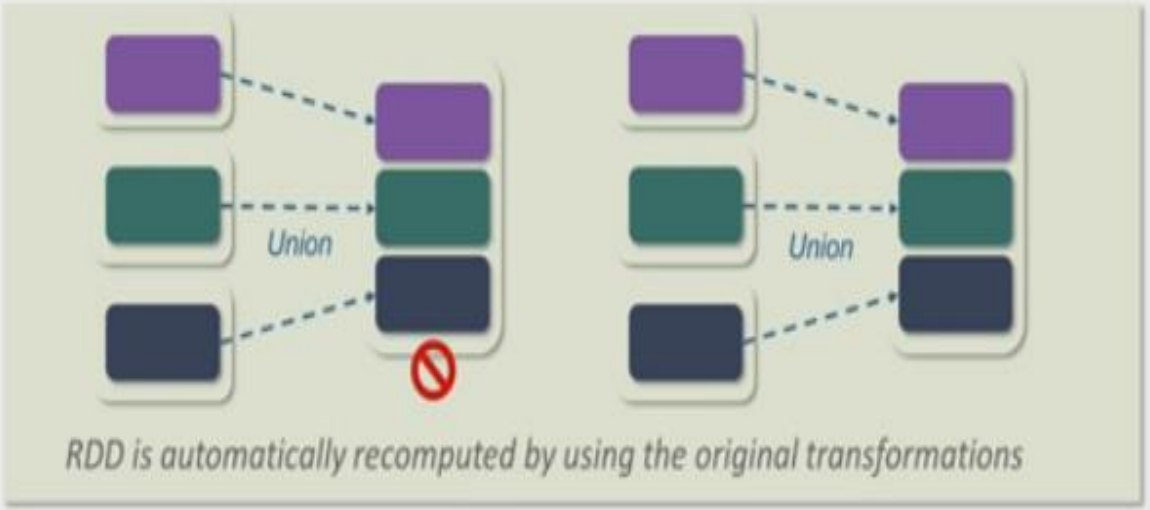
- Performance
- Ease of Use
- Costs
- **Data Processing**
- Fault Tolerance
- Security



Fault Tolerance

- Performance
- Ease of Use
- Costs
- Data Processing
- **Fault Tolerance**
- Security

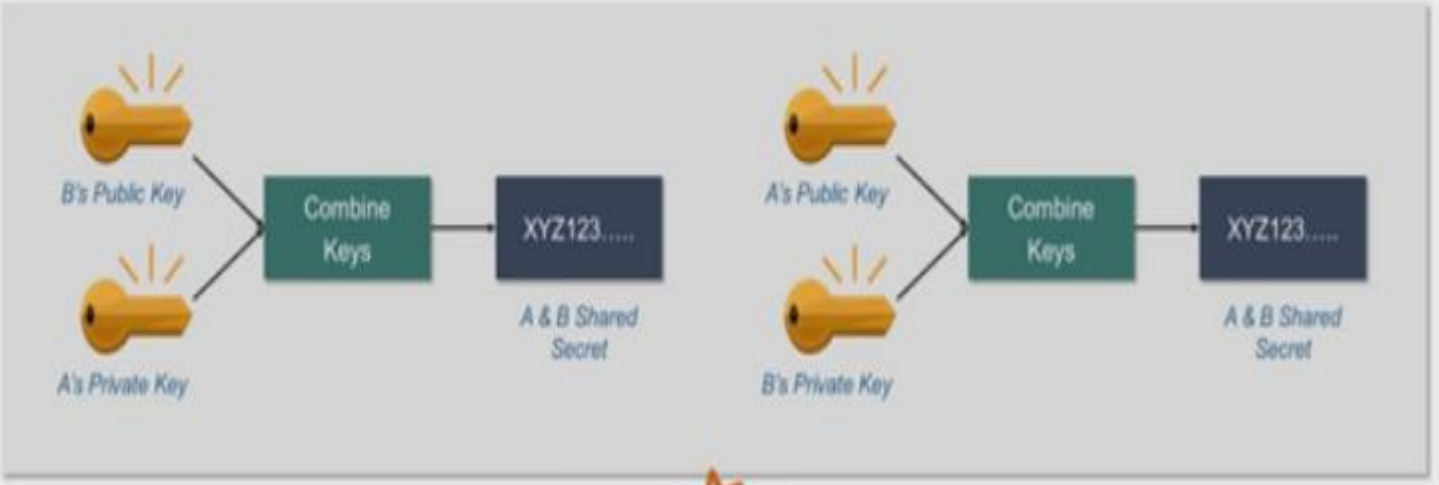
hadoop



Spark

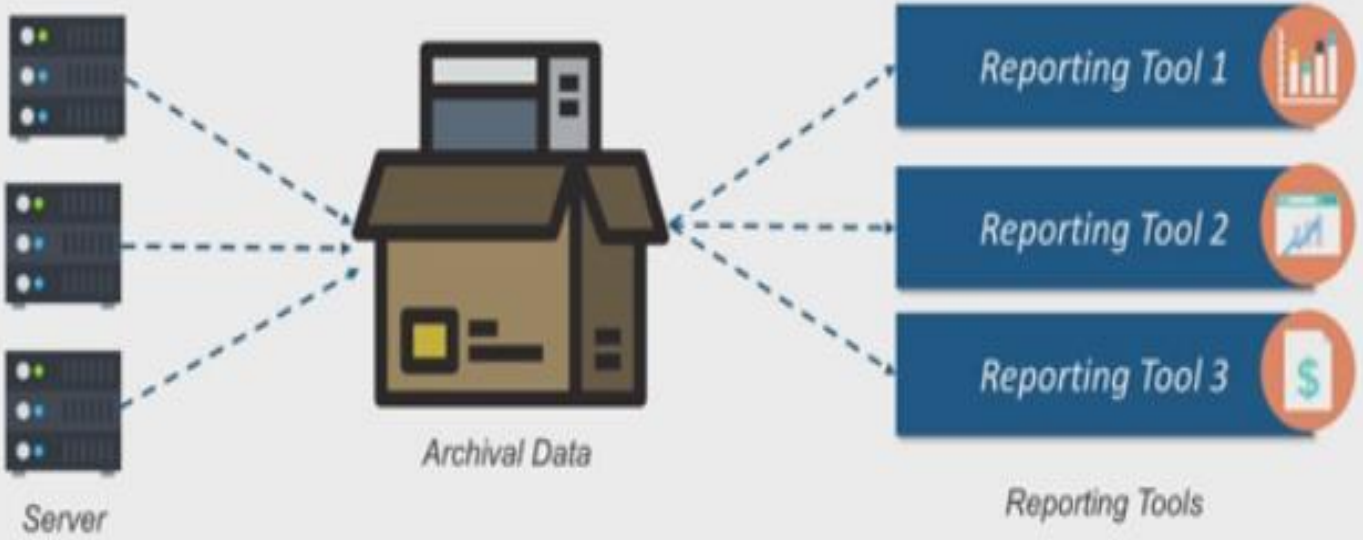
Security

- Performance
- Ease of Use
- Costs
- Data Processing
- Fault Tolerance
- Security

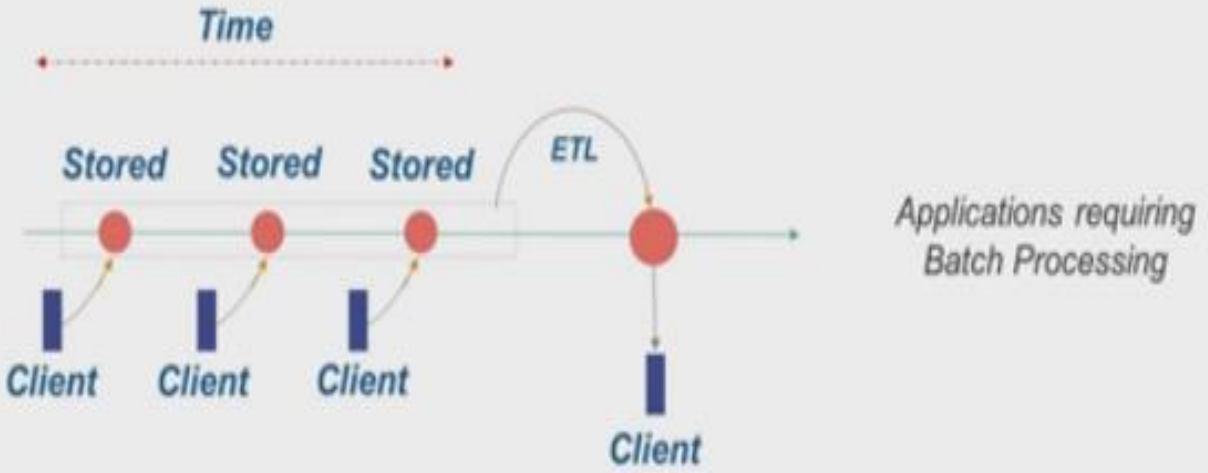


Hadoop Use-cases

1



2



Relational DB's vs. Big Data (Spark)

1. It deals with Giga Bytes to Terabytes
2. It is centralized
3. It deals with structured data
4. It is having stable Data Model
5. It deals with known complex inter relationships
6. Tools are Relational DB's: SQL,MYSQL,DB2.
7. Access is Interactive and batch.
8. Updates are Read and write many times.
9. Integrity is high
10. Scaling is Nonlinear

1. It deals with Petabytes to Zettabytes
2. It is distributed
3. It deals with semi-structured and unstructured
4. It is having unstable Data Model
5. It deals with flat schemas and few Interrelationships
6. Tools are Hadoop,R,Mahout
7. Access is Batch
8. Updates are Write once, read many times.
9. Integrity is low
10. Scaling is Linear

	HADOOP	SPARK
Performance	Process data on disk	Process data in-memory
Ease of use	Java need to be proficient in MapReduce	Java, Scala, R, Python more expressive and intelligent
Data processing	Need other platforms for streaming, graphs	MLLib, Streaming, Graphs
Failure tolerance	Continue from the point it left off	Start the processing from the beginning
Cost	Hard disk space cost	Memory space cost
Run	Everywhere	Runs on Hadoop
Memory	HDFS uses MapReduce to process and analyse data map reduces takes a backup of all the data in a physical server after each operation this is done because the data stored in a ram	This is called in memory operation

Fast	Hadoop works less faster than spark	Spark works more fast than Hadoop(100 times) batchfile,10xfaster on disk
Version	Hadoop 2.6.5 Release Notes	Spark 2.3.1
Software	It is a open source s/w or, Reliable, scalable distributed, computing It is a Big Data Tool	It is Fast and General Engine for large scale data processing
Execution Engine	DAG	It is a Big Data Tool
Big Data frame works	Hadoop	Advanced DAG, support for acyclic data flow and in memory computing
Hardware cost	More	Less
Library	External machine lib	Internal Machine lib

Recovery	Easier than spark Checkpoints are present	Failure recovery is difficult but still good
FileManagement system	Its own FMS(File Management System)	It does not come with own FMS it support to the cloud based data platform Spark was designed for Hadoop
Support	HDFS, Hadoop YARN Apache, Mesos	It support for RDD
Technologies	Cassandra, HBase, HIVE Tachyon, and any Hadoop source	Supports all systems in Hadoop Processed by Batch system
Use Places	Marketing Analysis, computing analysis, cyber security analytics	Online product
Run	Clusters Data bases, server	Cloud based systems and Data sets