Scala-Spark

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

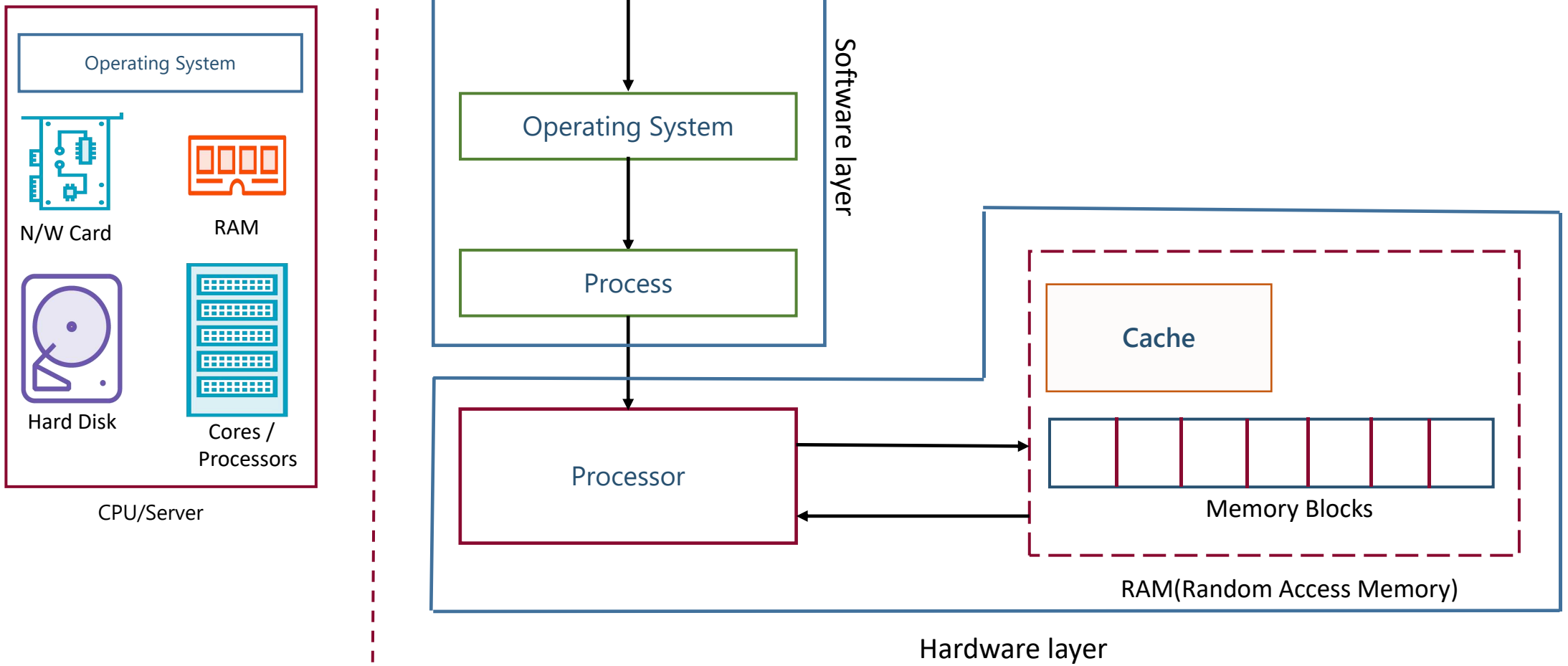
@PRUDHVI AKELLA
Senior Software Engineer-Big Data Analytics

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Where to Start is Always Important?

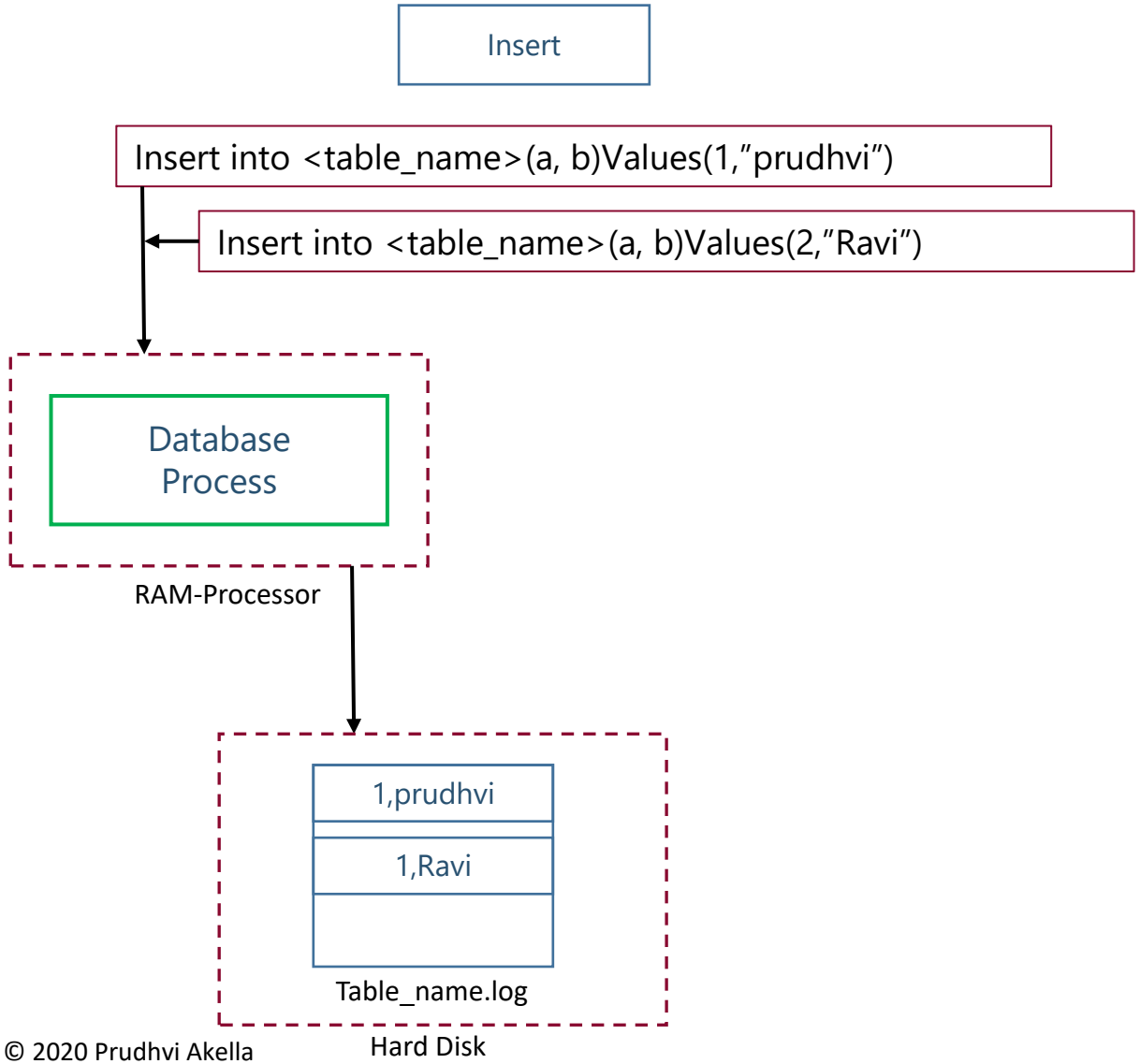
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

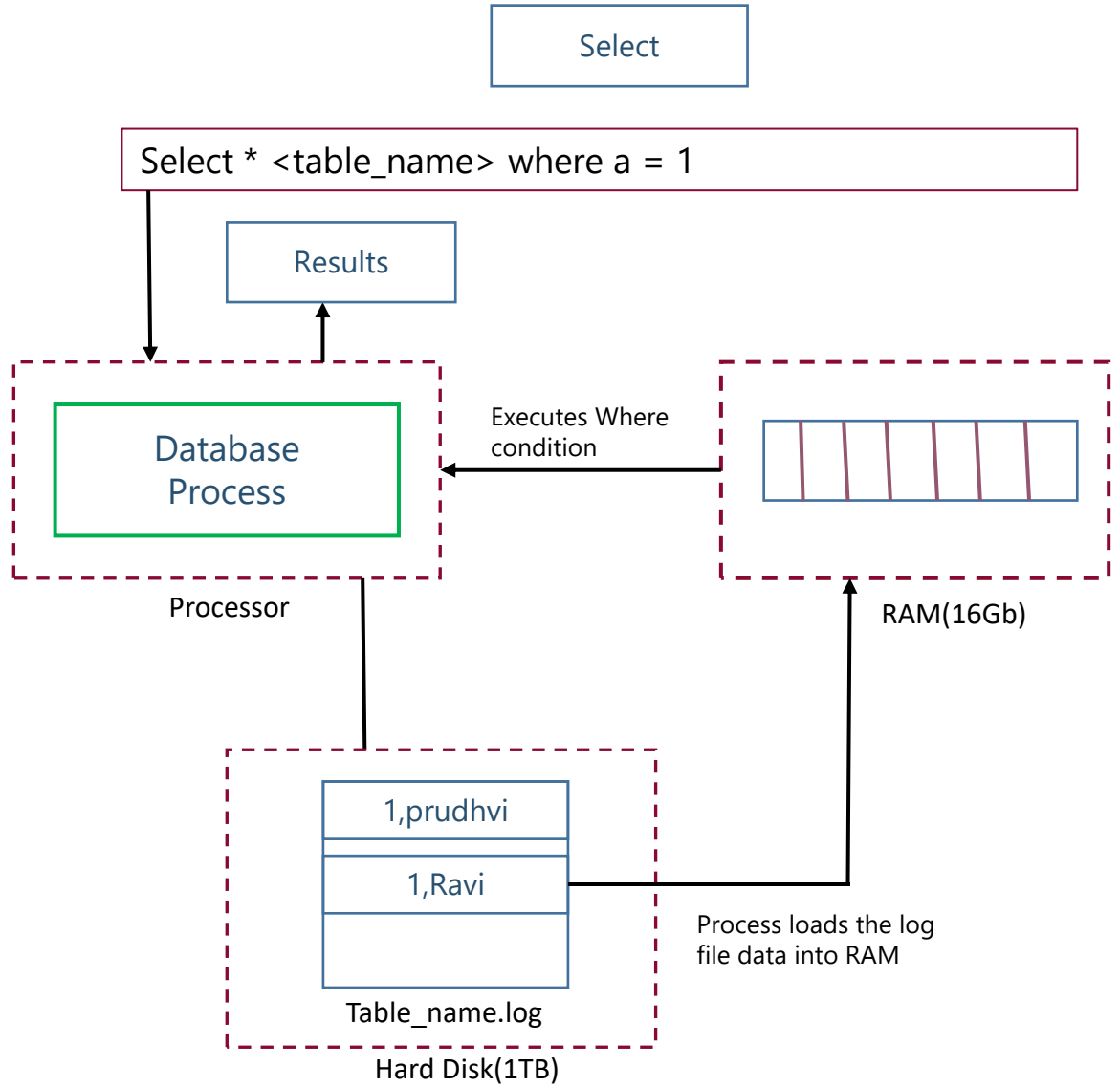
"Everything in the computer science space starts with understanding computer"



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Lets Talk about a bit about Relational Databases(Traditional System)

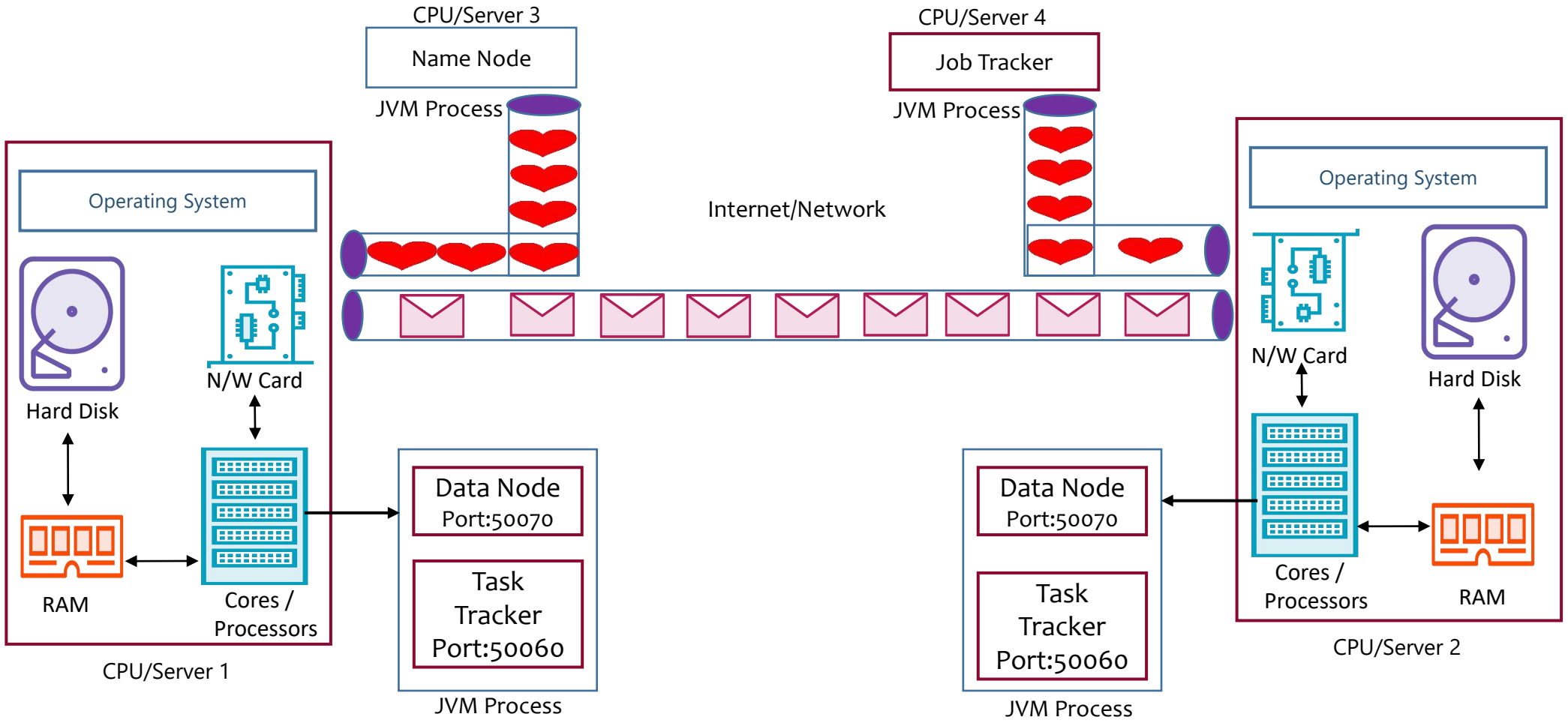




Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

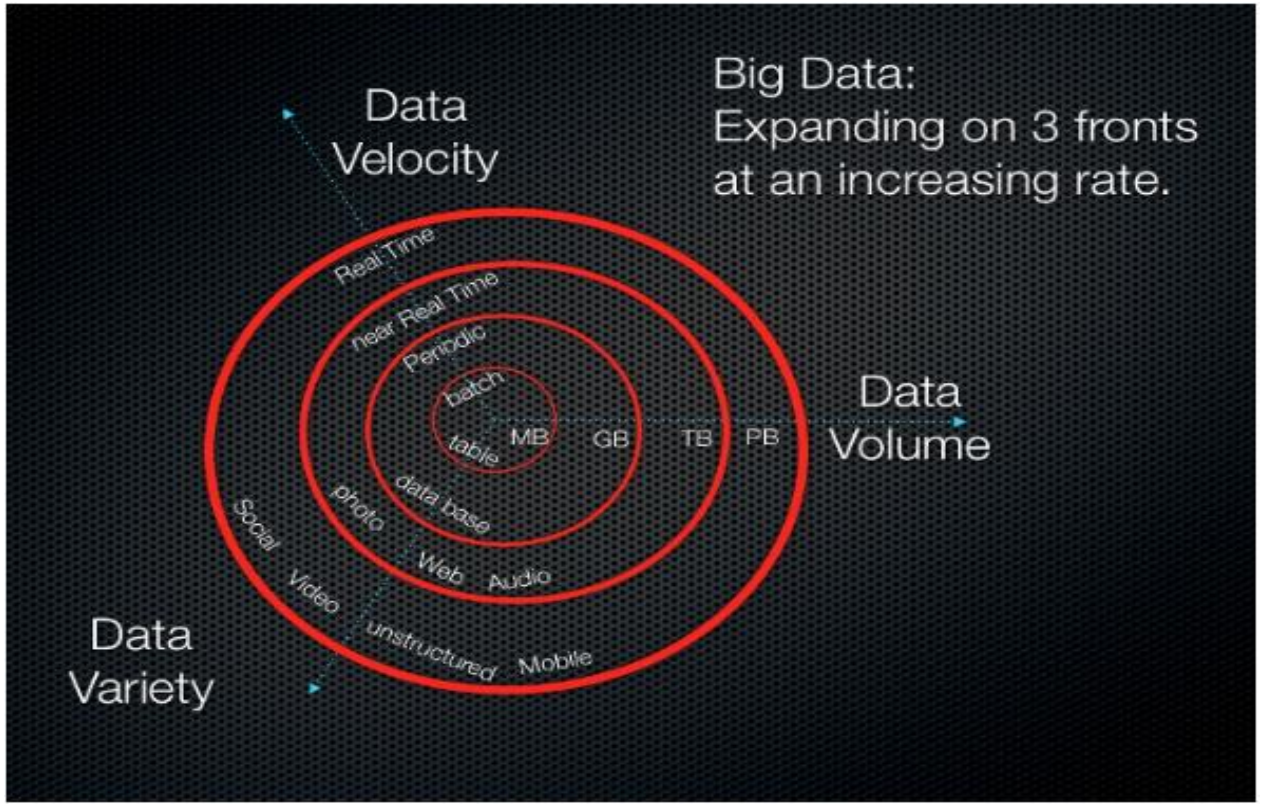
How Distributed System Work?

Simple funda is when storage is distributed then only you can distribute the processing

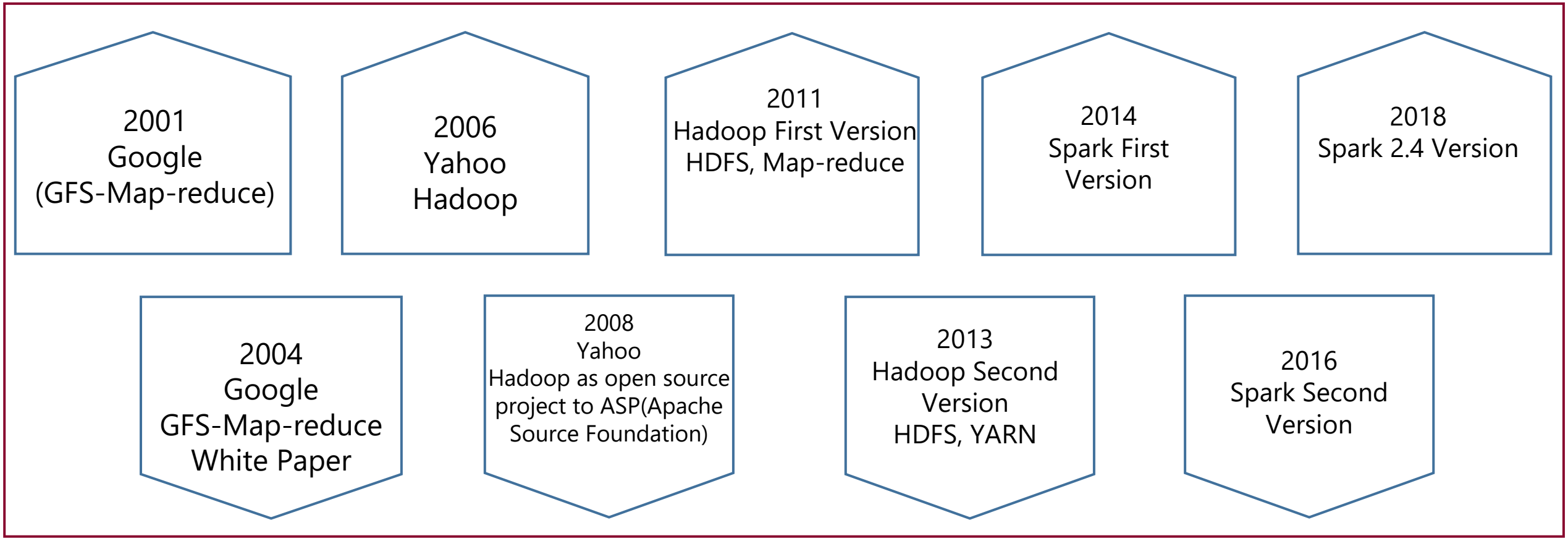


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

3V's or Three Dimensions:
V ---------- Volume
V ---------- Variety
V ---------- Velocity



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Why Not Map Reduce as Processing Engine?

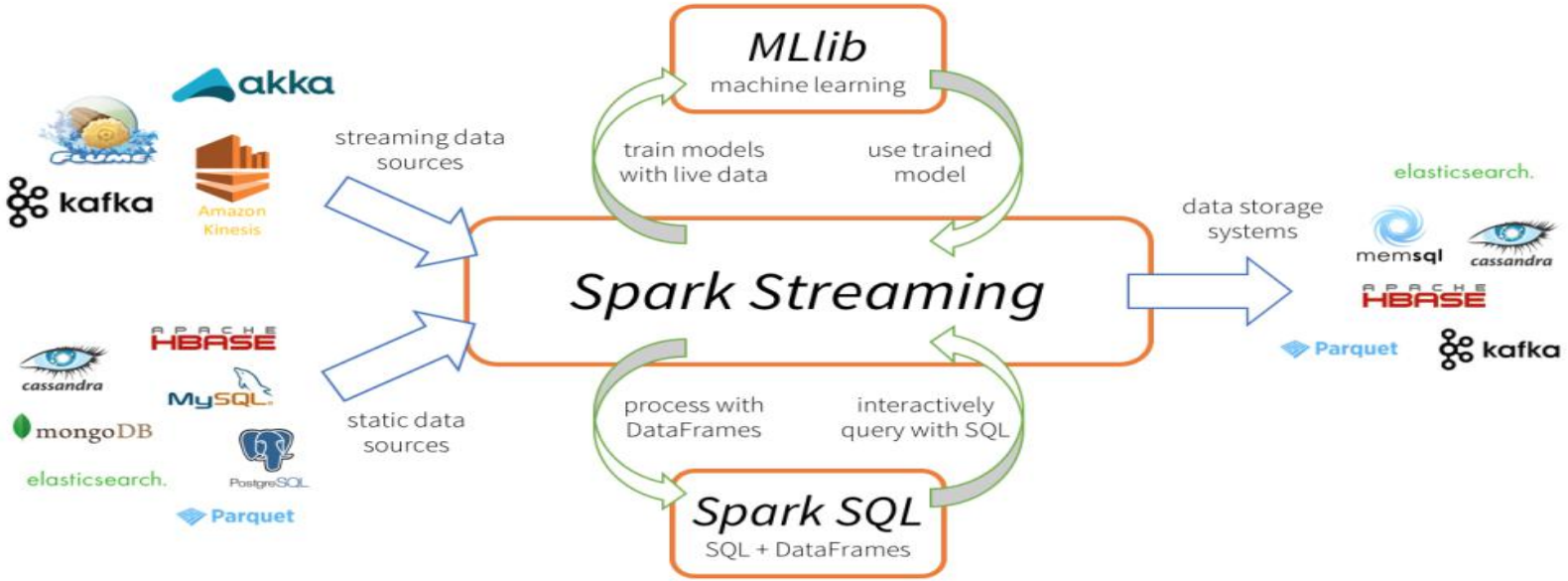
- Slow Processing Speed(unnecessary I/O operations with the disks)
- READ AND WRITE only to HDFS
- High Latency
- Support only Batch Processing
- Doesn’t support Real-Time Data Processing
- No Caching (Intermediate data will not be cached).
- Iterative model
- No Direct Machine Learning Support
- Needs to depends on external components to build ETL.
- No Ease of Use(Bigger codes)

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

NEDD A STRONG PROCESSING ENGINE THAN MAPREDUCE

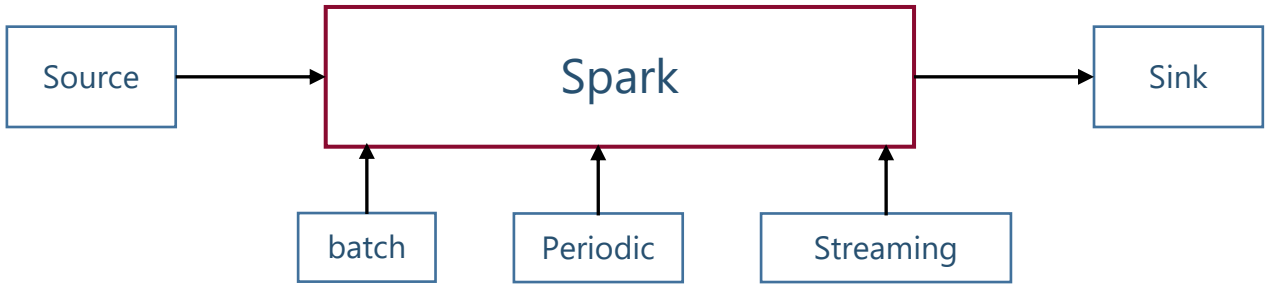
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

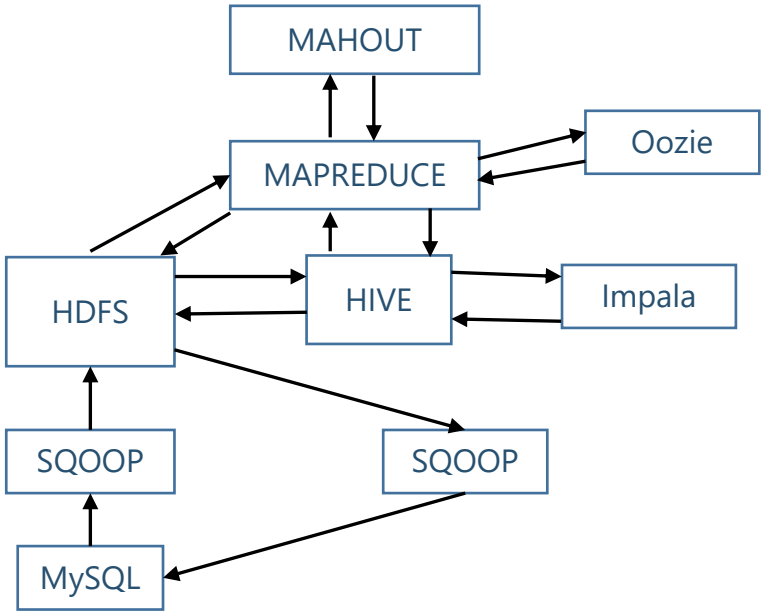
100x faster than Hadoop Map Reduce in memory, or 10x faster on disk

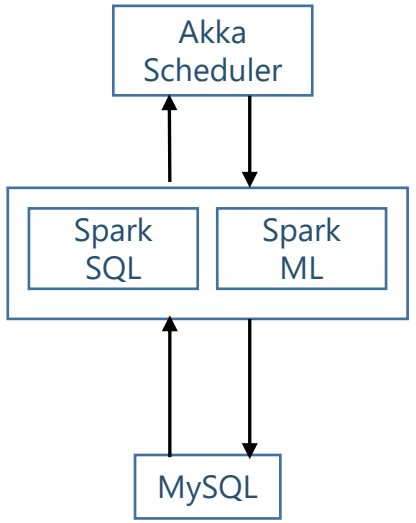


Spark Core	Spark SQL	Spark ML	Spark Streaming	Spark Graph

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in







Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

LETS MAKE OUR HAND DIRTY WITH SCALA

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Let's Starts with Spark

What is a Spark?

Apache Spark is an open source cluster computing framework for real-time data processing. The main feature of Apache Spark is its **in-memory cluster computing** that increases the processing speed of an application. Spark provides an interface for programming entire clusters with implicit **data parallelism** and **fault tolerance**. It is designed to cover a wide range of workloads such **as batch applications, iterative algorithms, interactive queries, and streaming**.

Spark Features

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Speed

Spark runs up to 100 times faster than Hadoop Map Reduce for large-scale data processing. It is also able to achieve this speed through controlled partitioning.

Powerful Caching

Simple programming layer provides powerful caching and disk persistence capabilities.

Deployment

It can be deployed through Mesos, Hadoop via YARN, or Spark's own cluster manager.

Real-Time

It offers Real-time computation & low latency because of in-memory computation.

Polyglot

Spark provides high-level APIs in Java, Scala, Python, and R. Spark code can be written in any of these four languages. It also provides a shell in Scala and Python.

Spark Eco System

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark Core

Spark Core is the base engine for large-scale parallel and distributed data processing. Further, additional libraries which are built on the top of the core allows diverse workloads for streaming, SQL, and machine learning. It is responsible for memory management and fault recovery, scheduling, distributing and monitoring jobs on a cluster & interacting with storage systems.

Spark SQL

Spark SQL is a new module in Spark which integrates relational processing with Spark's functional programming API. It supports querying data either via SQL or via the Hive Query Language. For those of you familiar with RDBMS, Spark SQL will be an easy transition from your earlier tools where you can extend the boundaries of traditional relational data processing.

Spark Streaming

Spark Streaming is the component of Spark which is used to process real-time streaming data. Thus, it is a useful addition to the core Spark API. It enables high-throughput and fault-tolerant stream processing of live data streams.

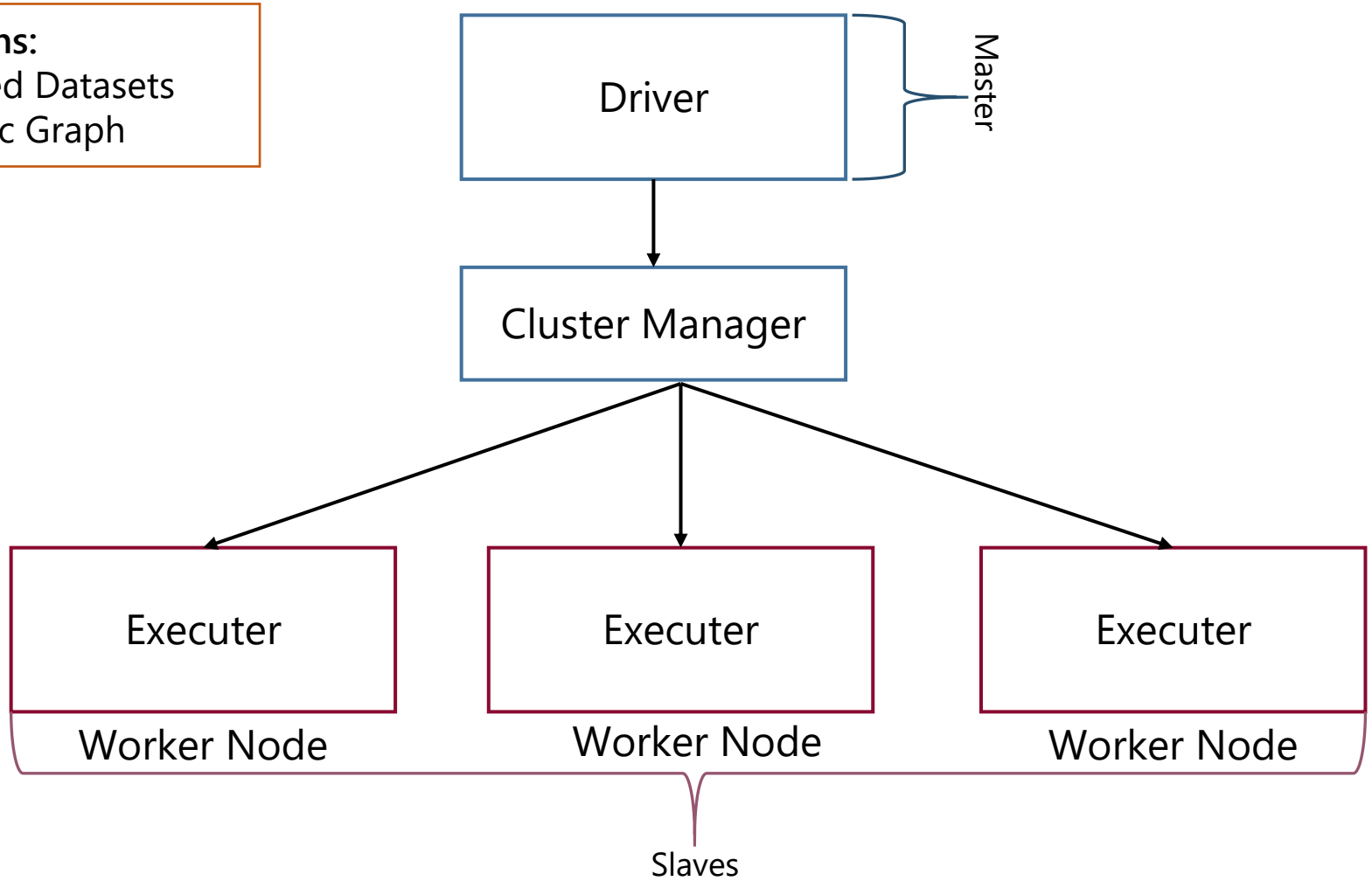
As you can see, Spark comes packed with high-level libraries, including support for R, SQL, Python, Scala, Java etc. These standard libraries increase the seamless integrations in a complex workflow. Over this, it also allows various sets of services to integrate with it like MLlib, GraphX, SQL + Data Frames, Streaming services etc. to increase its capabilities.

GraphX

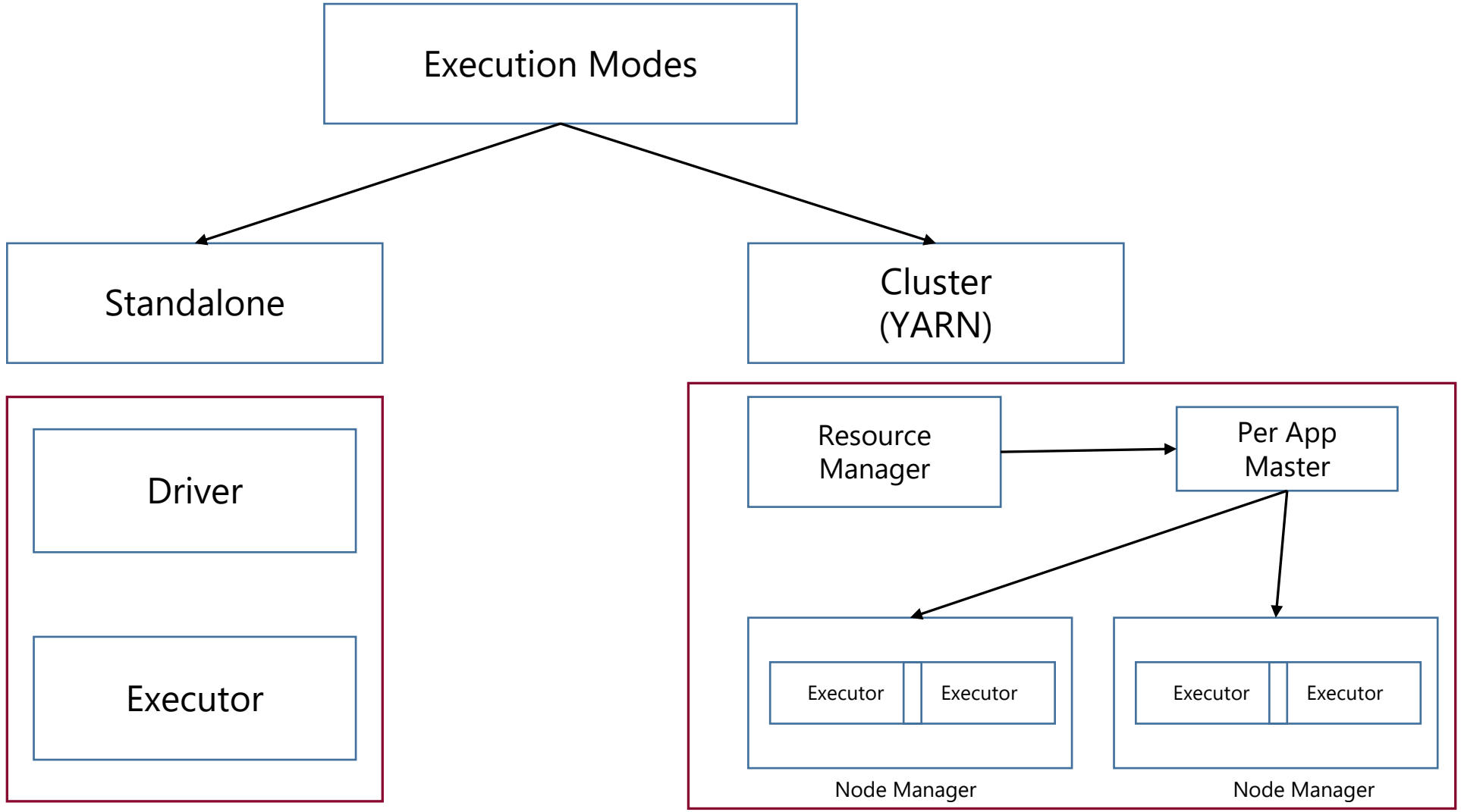
GraphX is the Spark API for graphs and graph-parallel computation. Thus, it extends the Spark RDD with a Resilient Distributed Property Graph. At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph (a directed multigraph with properties attached to each vertex and edge).

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Two Abstractions:
RDD: Resilient Distributed Datasets
DAG : Directed Acyclic Graph



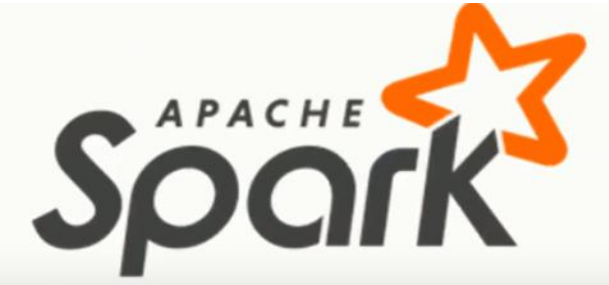
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Supported Cluster Manager in Spark

Cluster Managers are used to allocate resources for driver and executors

Standalone Mode



Cluster Mode

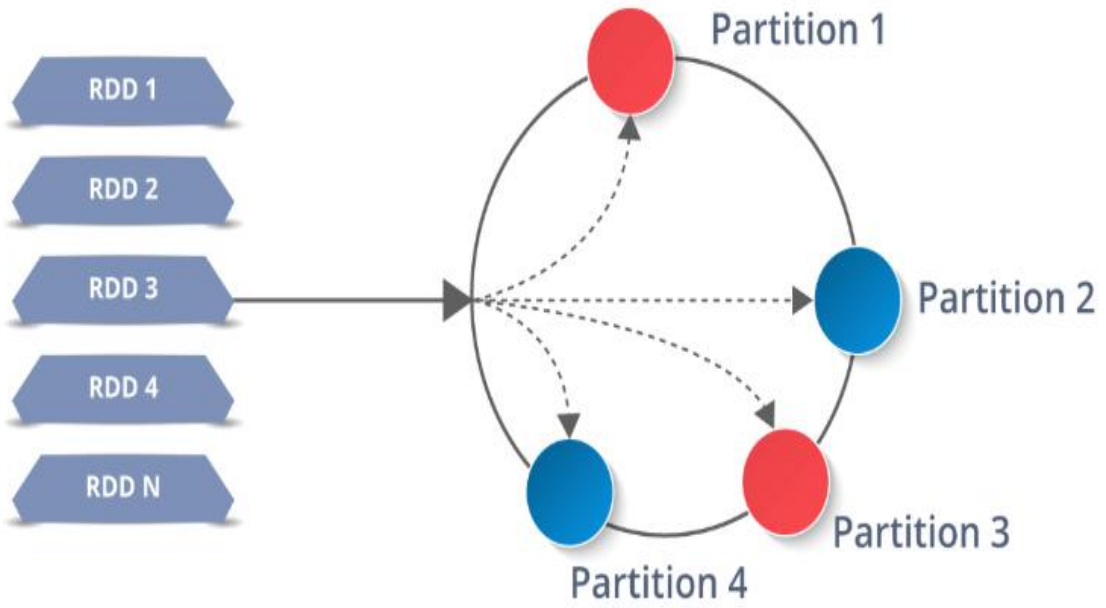
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

RDDs are the building blocks of any Spark application.

RDDs Stands for:
Resilient: Fault tolerant and is capable of rebuilding data on failure
Distributed: Distributed data among the multiple nodes in a cluster
Dataset: Collection of partitioned data with values

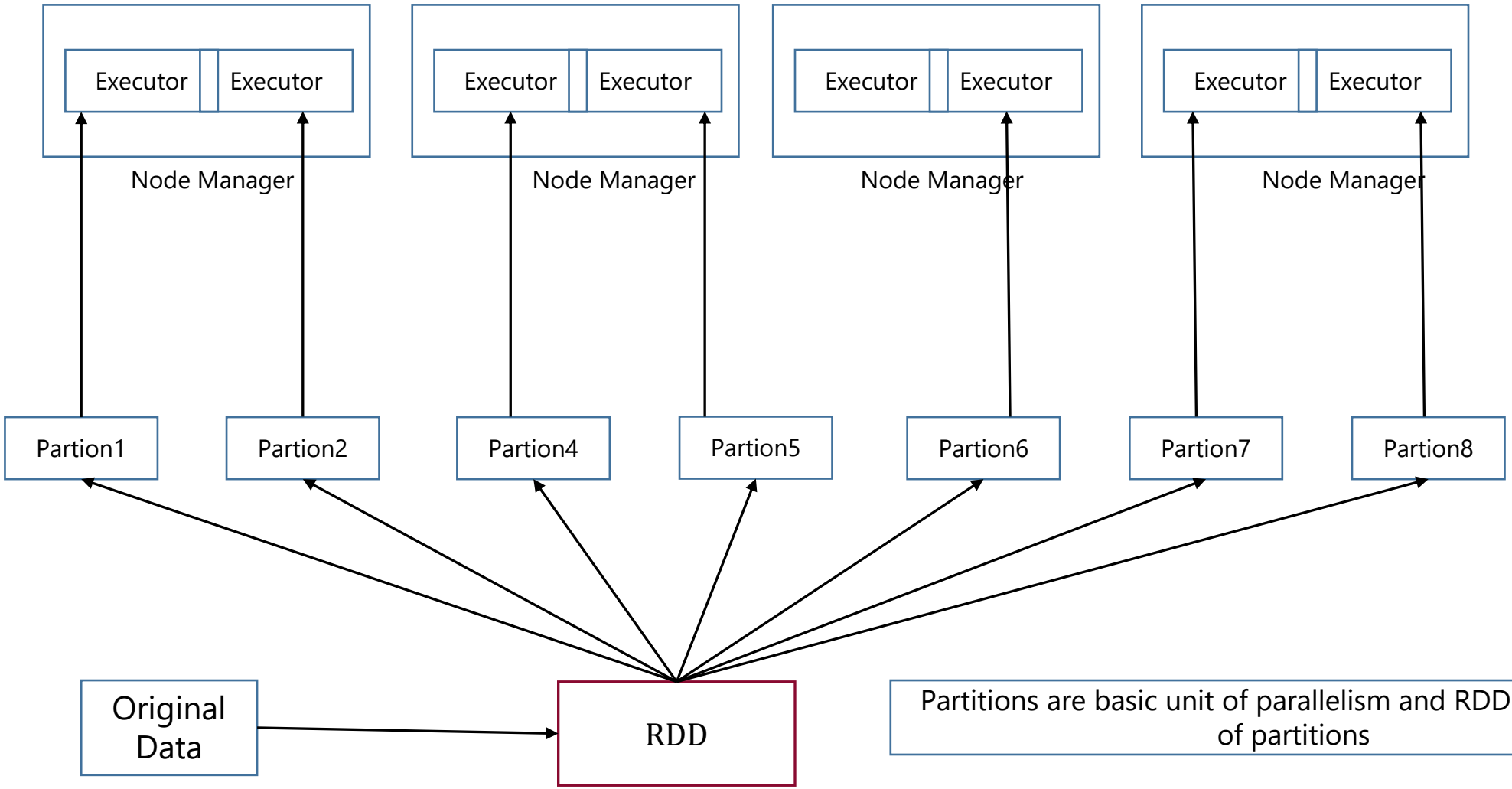
Points to Remember:

- → **Data in an RDD is split of chunks and each chuck is a partition. RDDs are highly resilient, Even in case of executor or worker failures RDD has capability to recover back by looking at RDD lineage.**
- → **Once you create an RDD it becomes immutable. By immutable I mean, an object whose state cannot be modified after it is created, but they can surely be transformed**



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark Temporary Distributed In-memory Storage



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

We had seen how RDD will be distributed as partitions into different work nodes.
Now Lets Look at Operation that you perform on RDD? To understand distributed processing or parallel processing on RDD

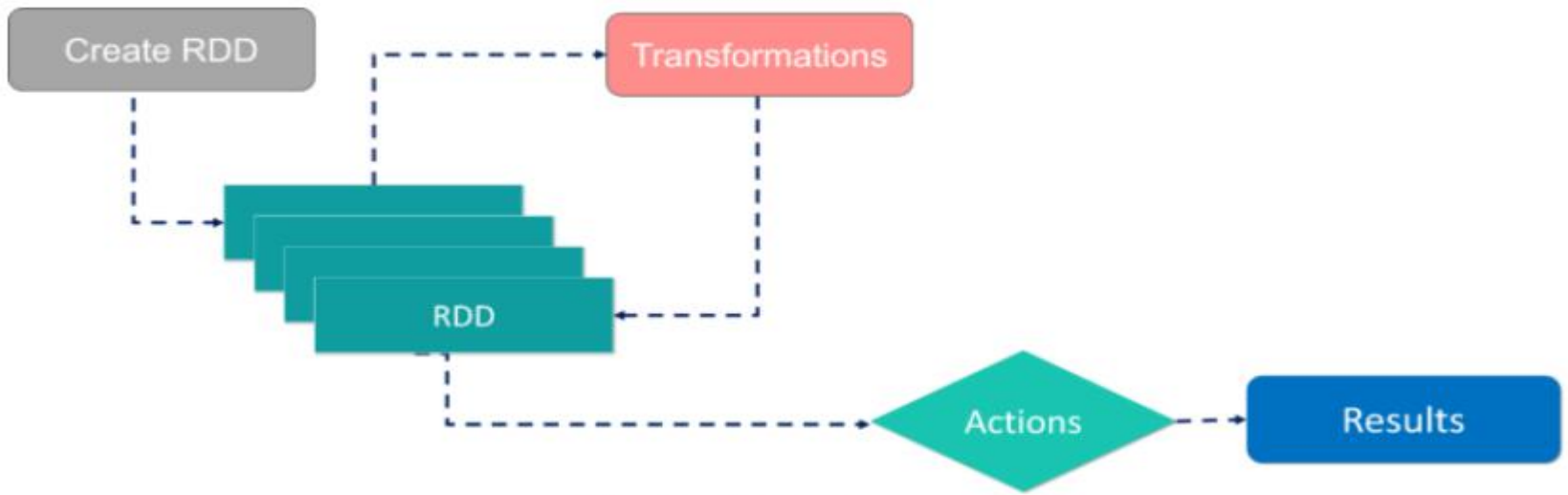
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

RDD Operations(Used to process the data that is stored in in-memory)

With RDDs, you can perform two types of operations:

Transformations: They are the operations that are applied to create a new RDD.

Actions: They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

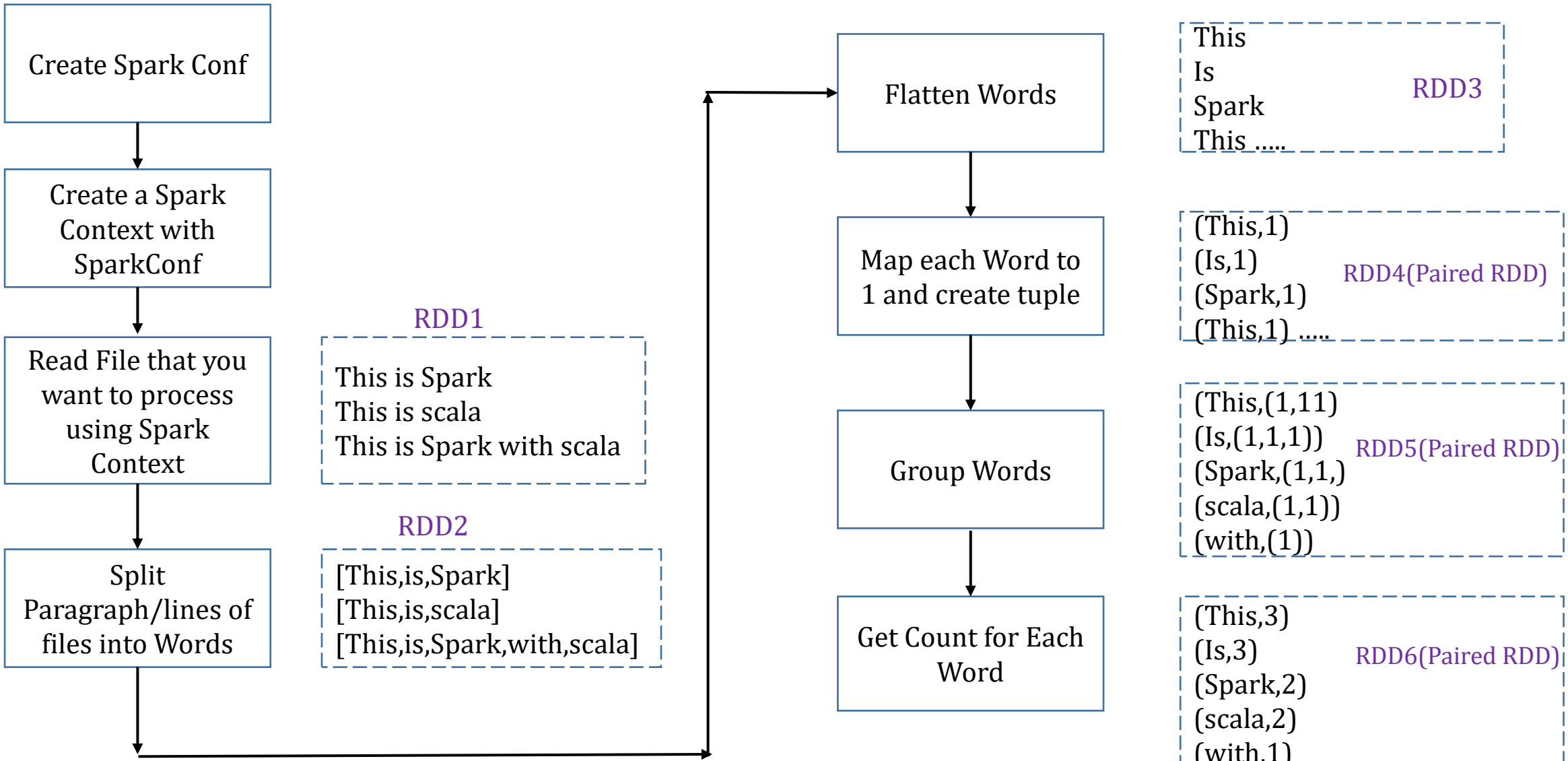


Workflow of RDD

Note: Transformation in Spark are lazy until and unless an action is performed on transformation job will not in spark

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Word Count Work Flow in Spark



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Lets Execute Word Count in Intellij in Standalone Mode

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

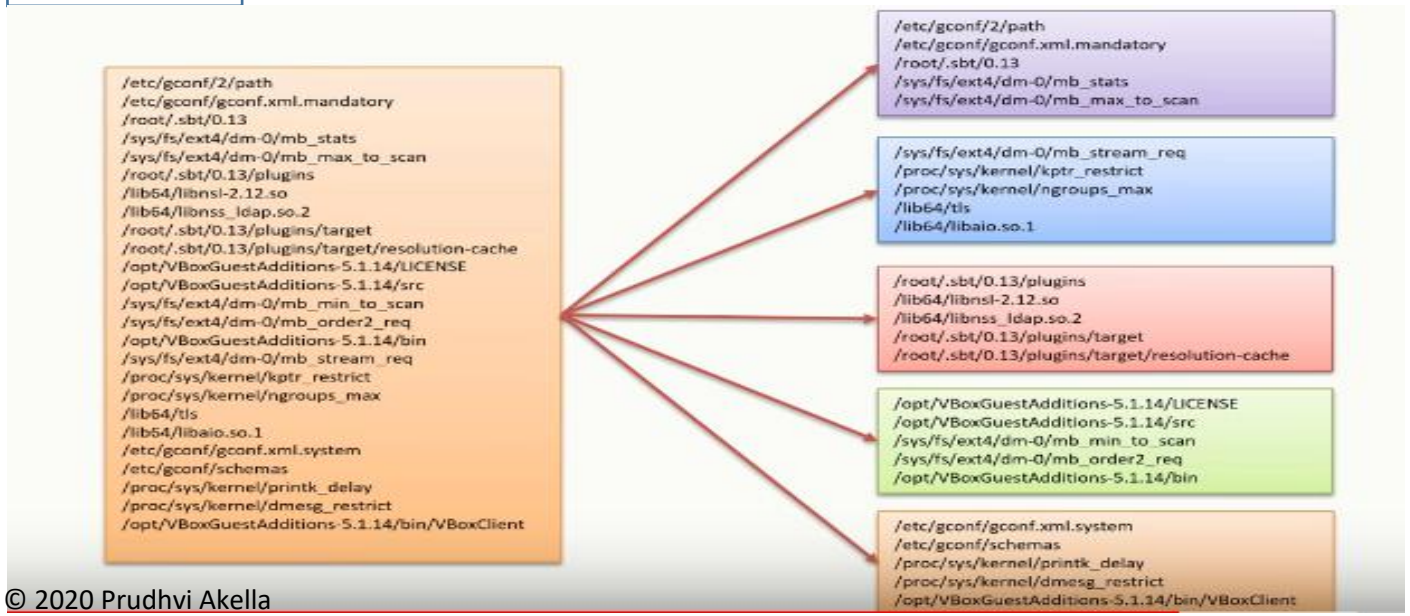
Types of Transformations and understand with Directories Word Count Example

```
/etc/cron.d
/etc/gnome-vfs-2.0
/etc/gnome-vfs-2.0/modules
/etc/audit
/etc/default
/etc/hbase-solr
/etc/hbase-solr/conf.dist
/etc/httpd
/etc/httpd/conf.d
/etc/httpd/conf
/etc/profile.d
/etc/hadoop-kms
/etc/hadoop-kms/conf.dist
/etc/hadoop-kms/tomcat-conf.https
```

```scala
val inputRDD = sc.textFile("<File Path>",5)
//Transformation1
val wordRDD = inputRDD.map(rec => rec.split("/"))
//Transformation2
val wordPairRDD = wordRDD.map(word => (word(1), 1))
//Transformation3
val wordCountRDD = wordPairRDD.reduceByKey(_ + _)
//Action
wordCountRDD.collect().foreach(println)
```

```
(home,12675)
(mnt,2)
(etc,338)
(lib,997)
(sys,1911)
(proc,3687)
(boot,6)
(usr,14611)
(tmp,81)
```

Step1



By Default if file is local file system and master is local then partition count is based on number of cores in cluster lets in our case setMaster conf is set to local[5] then partitions will be 5 if it set to local[4] then it 4.

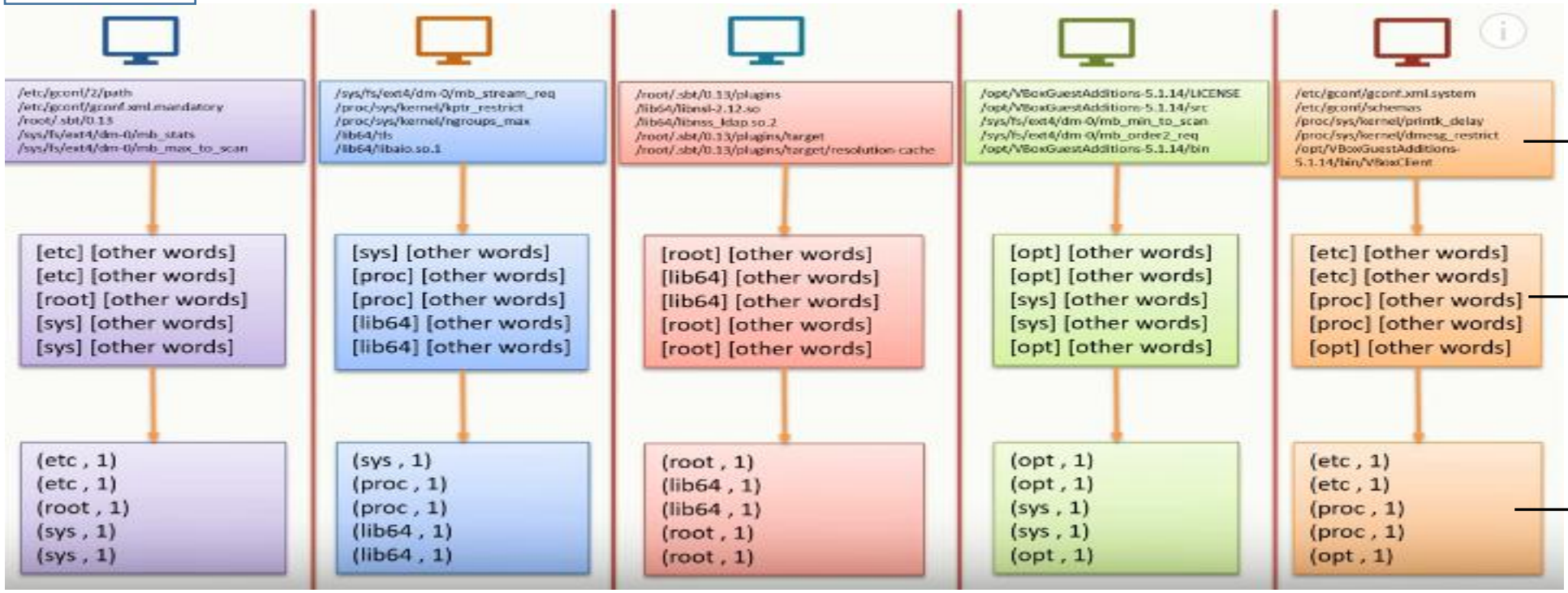
If File is in HDFS in that case partition count will be based on Input split size say 64MB(Hadoop version1) and 128MB for (Hadoop Version2) Lets say file size is 512 MB and block size or split size Is 64 then partition count = 512/64 = 8

User can define partition count while reading file or doing any transformations

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Narrow Transformation

Step2



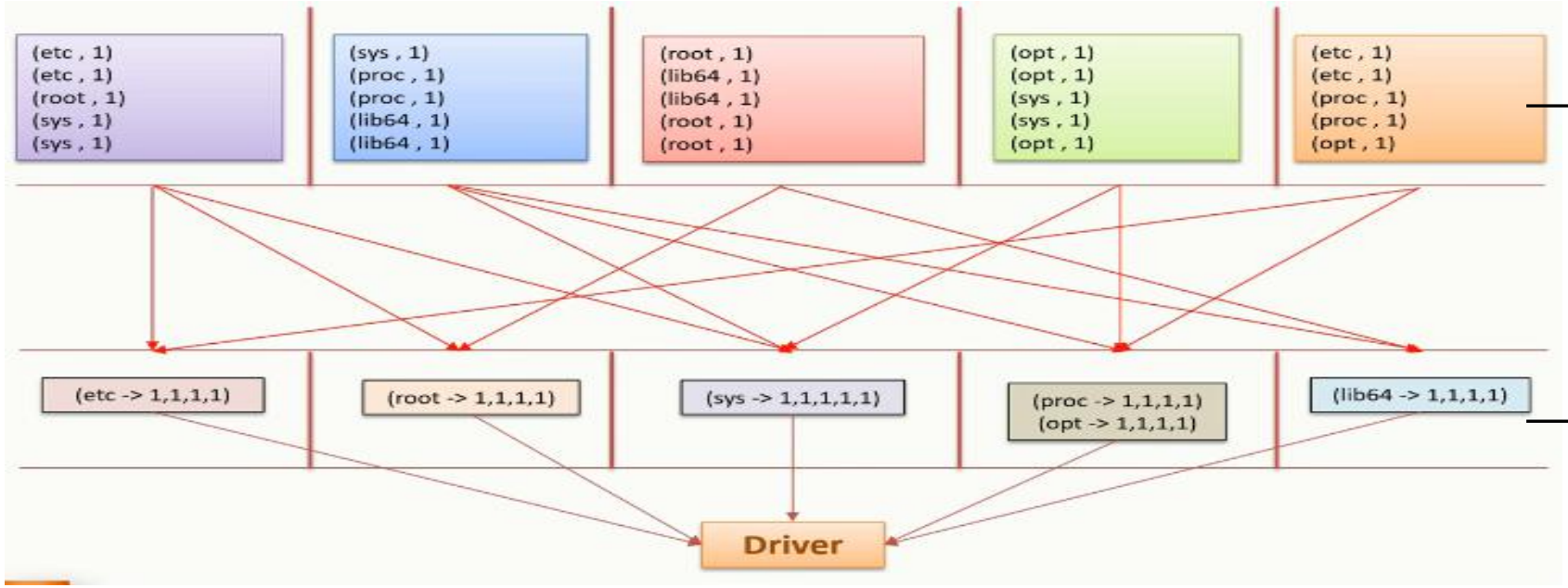
In *Narrow transformation*, all the elements that are required to compute the records in single partition live in the single partition of parent RDD.

Examples: map, Filter, MapPartition, Filter, Sample

If you look at the above example both map transformations 1 and 2 happen on the same Parent Partitions.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Wide Transformations



Now Spark has to perform reduce by key operation but keys are spread across the machine how spark is going to do that? so spark has to perform reparation in such a way that each key has to be in one partition this is called **shuffling and sort stage**. When every shuffling happens spark creates a new stage. when ever you perform group by or join or reduce by you will see **new stage**.
When a transformation creates a re-partition or new stage by shuffling and sorting the data across the partitions is called to be **Wide Transformations.** Examples: Intesection, Distinct, ReducebyKey, GroupbyKey, Join, Cartisian, Repartition, Coalsce.

hashcode(key)%num of partitions = which parition

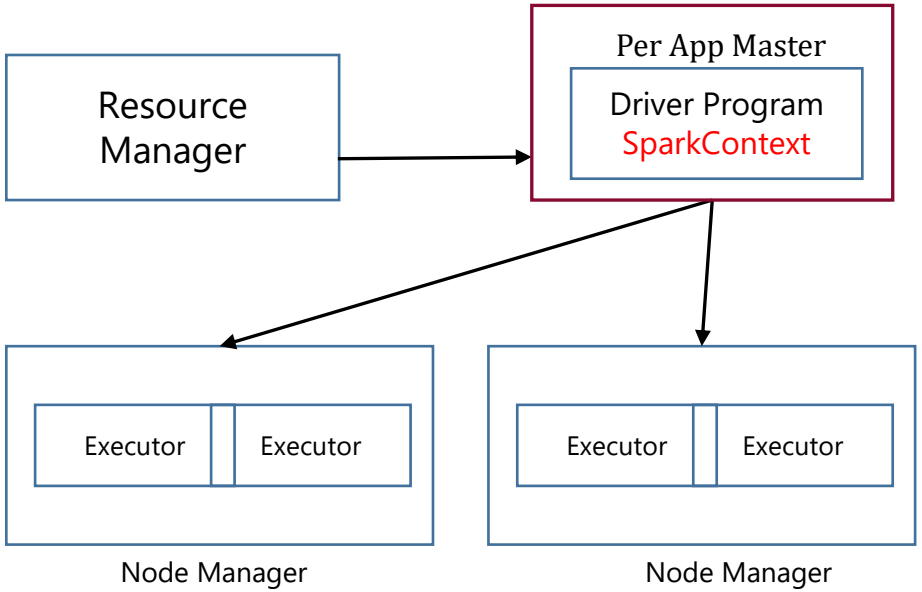
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

What is Spark Context and Spark Conf?

Spark Context is the main entry point into Spark functionality, and therefore the heart of any Spark application. It allows Spark Driver to access the cluster through its Cluster Resource Manager and can be used to create RDDs, accumulators and broadcast variables on the cluster. Spark Context also tracks executors in real-time by sending regular heartbeat messages.

Spark Context is created by Spark Driver for each Spark application when it is first submitted by the user. It exists throughout the lifetime of the Spark application.

Spark Context stops working after the Spark application is finished. For each JVM only one Spark Context can be active. You must stop()activate Spark Context before creating a new one.



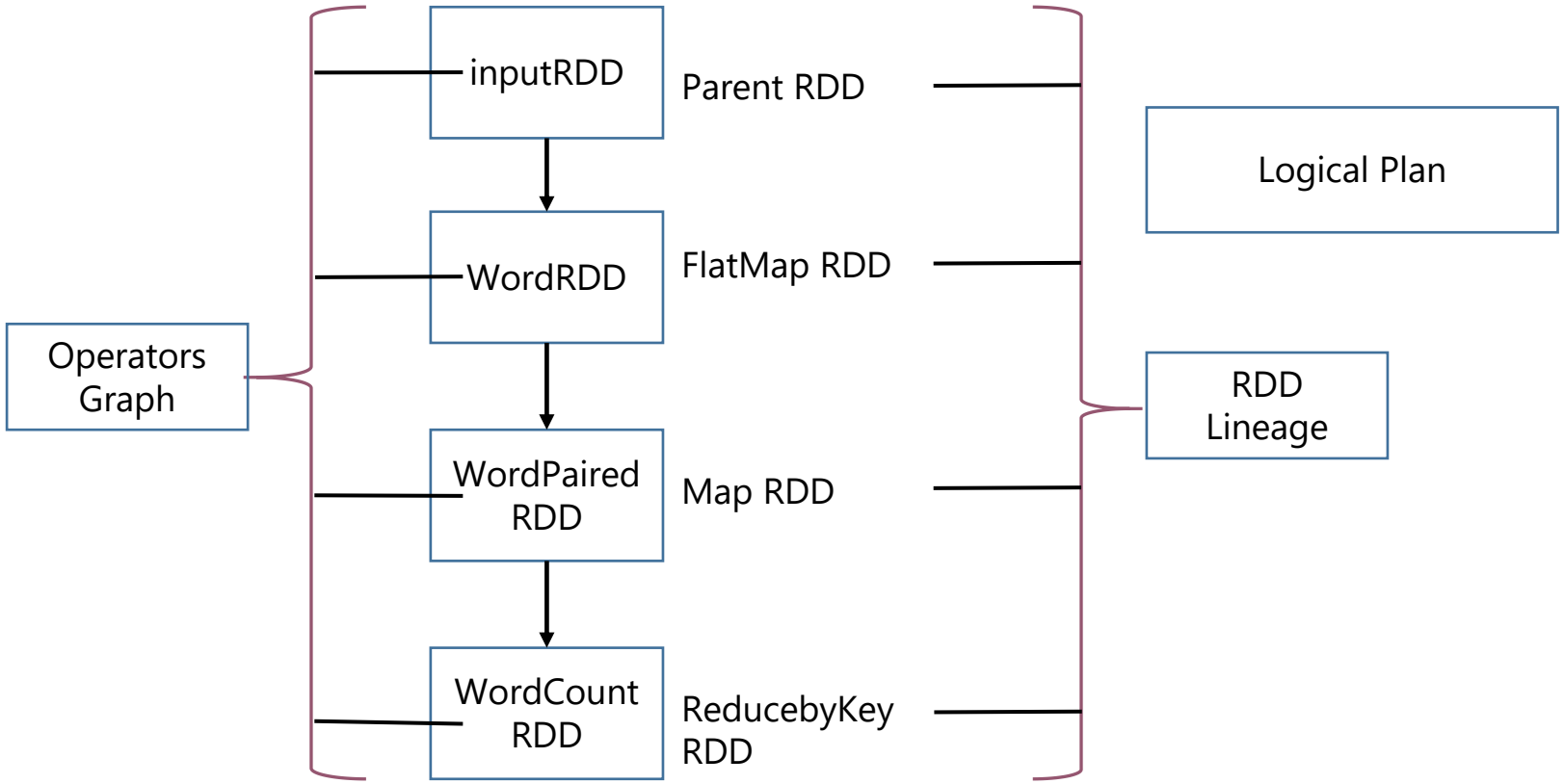
Programmatically-Scala

```
val conf = new SparkConf().setAppName("wordcount").setMaster(args(0))

val sc = new SparkContext(conf)
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

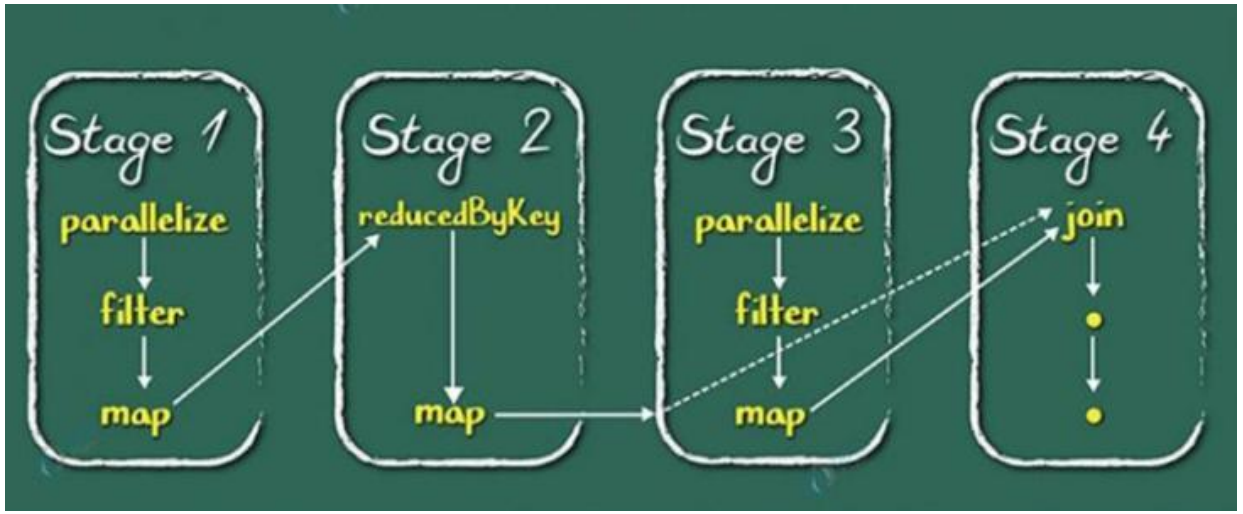
RDD Lineage or Operator Graph

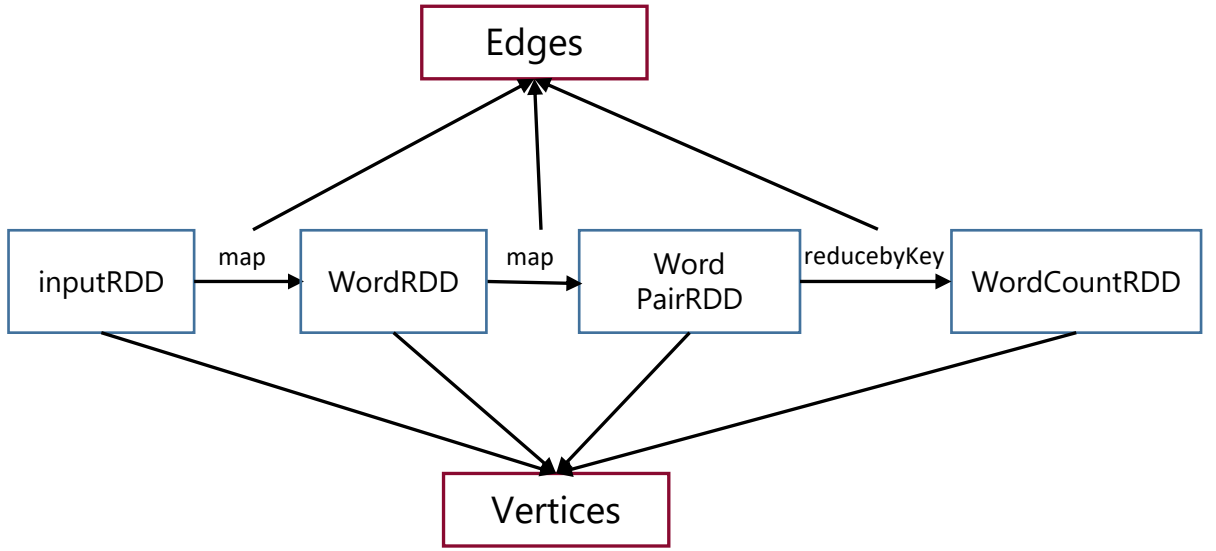
Spark is **lazy evaluated** means when a transformation(map or filter etc) is called, it is not executed by Spark immediately, instead each RDD maintains a pointer to one or more parent RDDs along with the metadata about what type of relationship it has with the parent RDD.It just keeps a reference (and never copies) to its parent RDD , that's a lineage. A lineage is created for each transformation. A lineage will keep track of what all transformations has to be applied on that RDD, including the location from where it has to read the data. It creates a logical execution plan. It will created by **Spark Interpreter** and its called to First Layer when you submit the job. This RDD lineage is used to re-compute the data if there are any faults as it contains the pattern of the computation.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

DAG(Direct Acyclic Graph)





Refer to Directory word count example for above graph

Vertices: RDD Data points
Edges: Edges performs transformations on Vertices and create a new Vertex.

DAG a finite direct graph with no directed cycles. There are finitely many ***vertices*** and ***edges***, where each edge directed from one vertex to another. It contains a sequence of vertices such that every edge is directed from earlier to later in the sequence. When an Action is observed the operator graph will be given to the **DAG scheduler** and it will be divided into **Stages and Tasks** based on the transformations and each task will be executed in **Executor** using **Task scheduler**.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Job View

Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at Directory_WordCount.scala:22	2020/01/29 04:39:43	0.1 s	0/2	0/2

Detail Stage View

Stages for All Jobs

Active Stages: 1
Pending Stages: 1

Active Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
0	map at Directory_WordCount.scala:18 (kill) +details	2020/01/29 04:42:10	0.5 s	0/1				

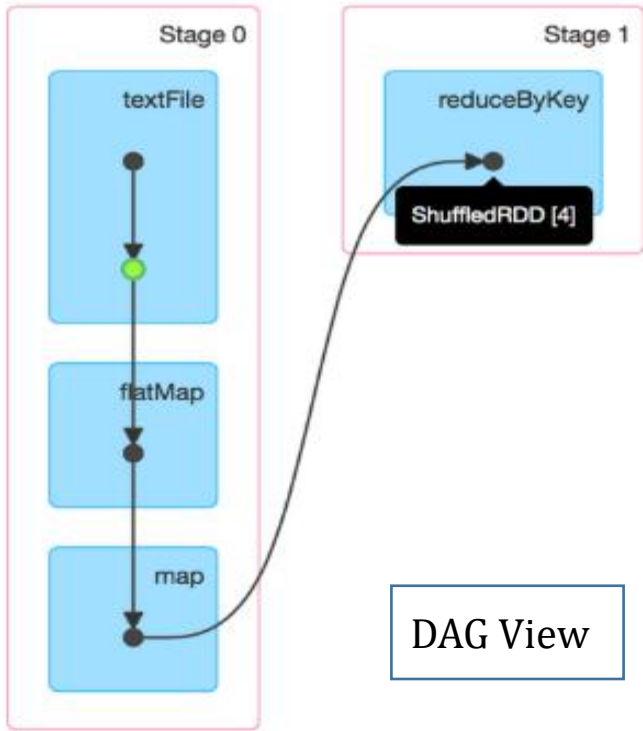
Pending Stages (1)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	collect at Directory_WordCount.scala:22 +details	Unknown	Unknown	0/1				

Details for Job 0

Status: SUCCEEDED
Completed Stages: 2

▸ Event Timeline
▾ DAG Visualization



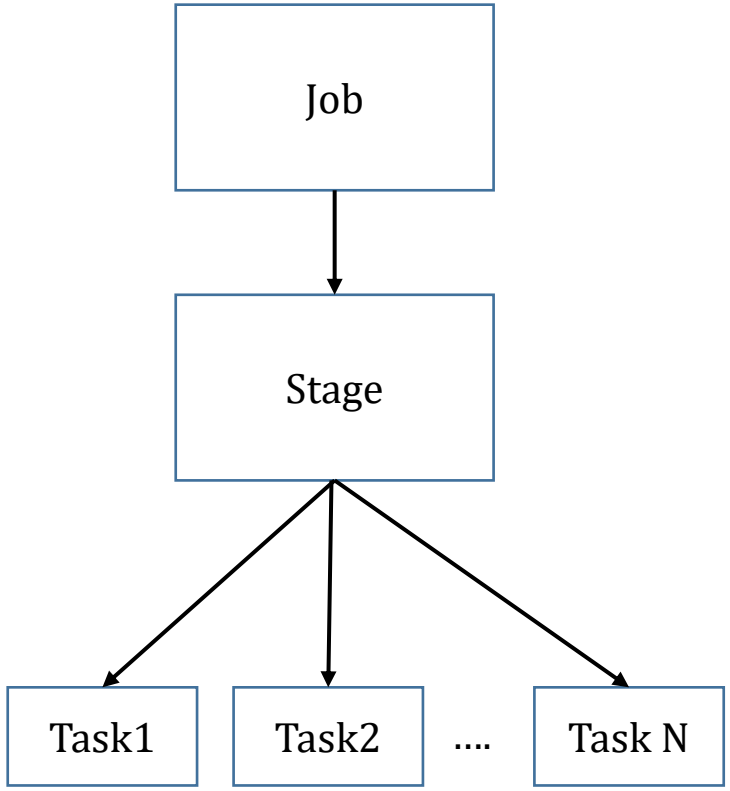
DAG View

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

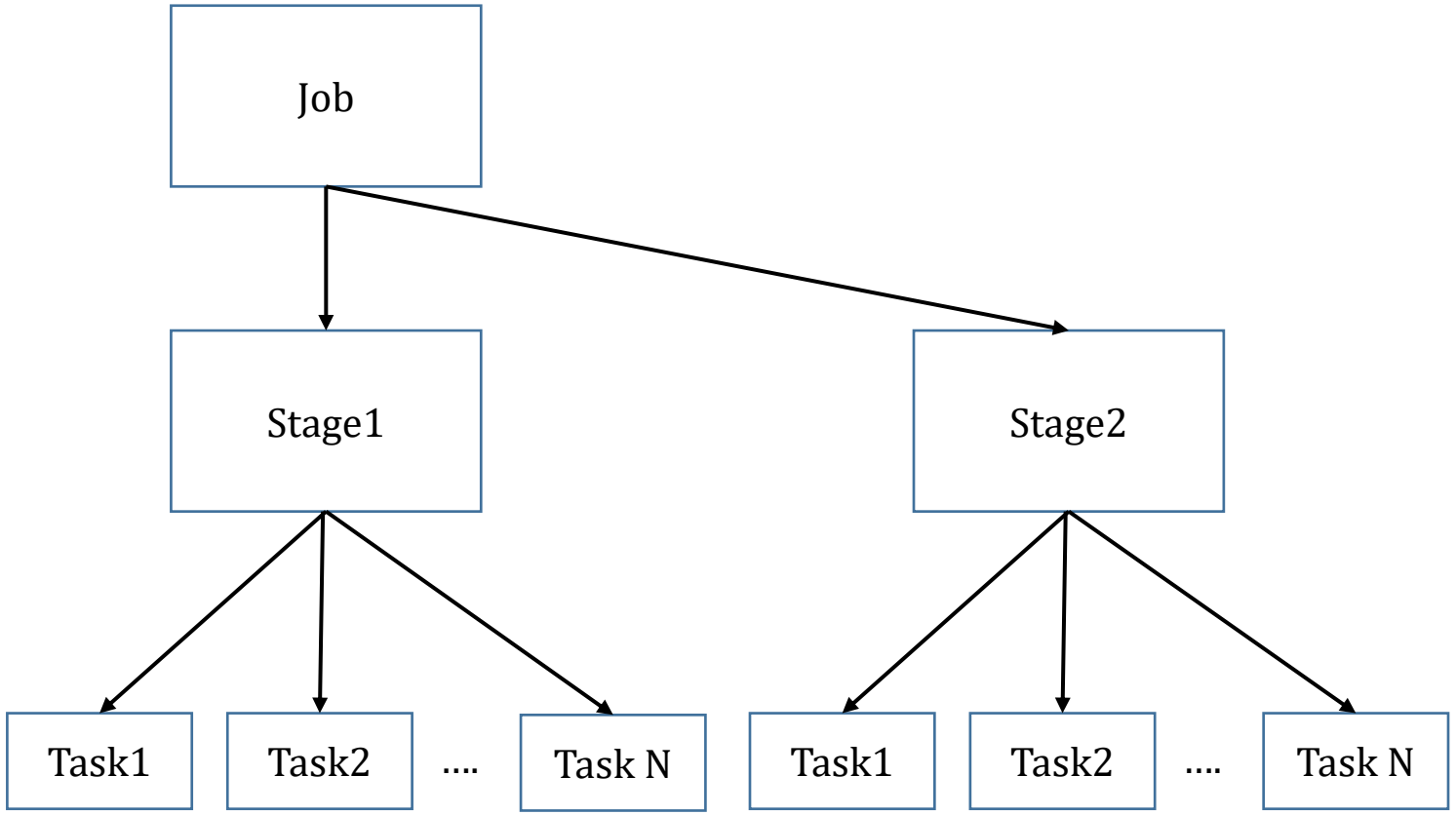
- Lineage graph deals with RDDs so it is applicable up-till transformations , Whereas, DAG shows the different stages of a spark job. it shows the complete task(transformation and also Action).
- Logical Execution Plan starts with the earliest RDDs (those with no dependencies on other RDDs or reference cached data) and ends with the RDD that produces the result of the action that has been called to execute.
- A logical plan, i.e. a DAG, is materialized and executed when SparkContext is requested to run a Spark job. The execution DAG or physical execution plan is the DAG of stages.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

High Level View of Spark Transformations Workflow



Narrow Transformations



Wide Transformations

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Important points to Remember

- In spark, a single concurrent task can run for every partition of an RDD. Even up to the total number of cores in the cluster.
- **val rdd= sc.textFile ("file.txt", 6)**
  we are creating an RDD named textFile with 6 partitions. Suppose that you have a cluster with 5 cores and assume that each partition needs to process for 5 minutes. In case of the above RDD with 6 partitions, 5 partition processes will run in parallel. As there are 5 cores and the 6th partition process will process after 5 minutes when one of the 5 cores, is free. The entire processing will finish in 10 minutes. During the 6th partition process, the resources will remain idle.
- Best way to decide a number of spark partitions in an RDD is to make the number of partitions equal to the number of cores over the cluster. This results in all the partitions will process in parallel. Also, use of resources will do in an optimal way.
- Task scheduling may take more time than the actual execution time if RDD has too many partitions. As some of the worker nodes could just be sitting idle resulting in less concurrency. Therefore, having too fewer partitions is also not beneficial. That may lead to improper resource utilization and also data skewing.
- The recommend number of partitions is around 3 or 4 times the number of CPUs cores in the cluster so that the work gets distributed more evenly among the CPUs cores.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Points to Remember:
Every Transformation has a specific return type and they are side effect free. So complier can easily infer the return type by looking at the right hand side expression.

```
val collection = [1,2,4,5] // explicit type Array
val c1 = collection.map( value => value + 1) // Array only
val c2 = c1.map( value => value +2) // Array only
val c3 = c2.count() // inferred as Int
val c4 = c3.map( value => value +3) // gets error. As you cannot map over an integers
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Essential Core & Intermediate Spark Operations

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Essential Core & Intermediate PairRDD Operations

General

- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey

Math / Statistical

- sampleByKey

Set Theory / Relational

- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure

- partitionBy

- keys
- values

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

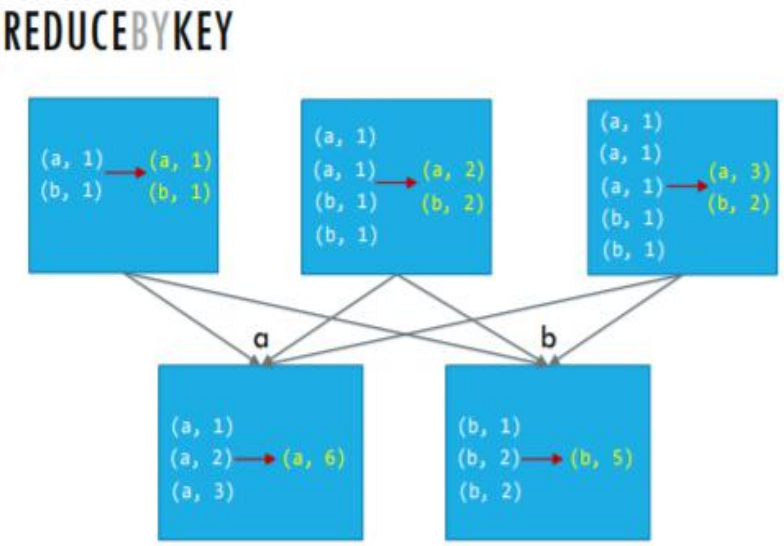
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

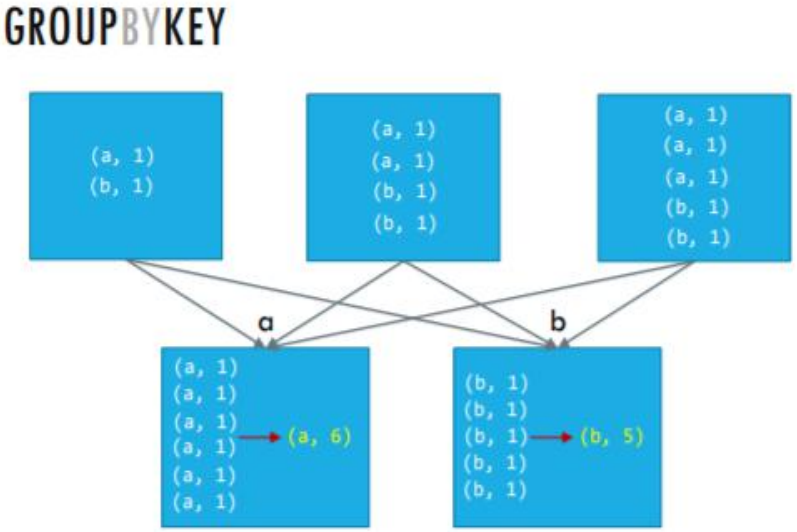
Important Transformations and Action as clearly explained in the Databricks Material given to you have look at it.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

GroupbyKey V/s ReducebyKey

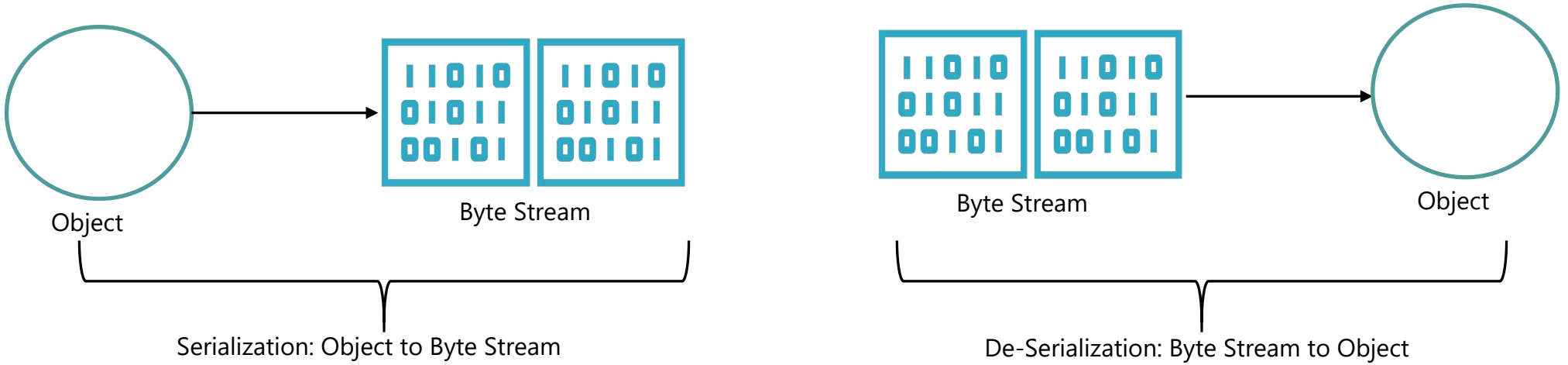
```
val words = Array("one", "two", "two", "three", "three", "three")
val wordPairsRDD = sc.parallelize(words).map(word => (word, 1))

val wordCountsWithReduce = wordPairsRDD
  .reduceByKey(_ + _)
  .collect()

val wordCountsWithGroup = wordPairsRDD
  .groupByKey()
  .map(t => (t._1, t._2.sum))
  .collect()
```





- → Both wide Transformations has to be performed on paired RDD only
- → Before Shuffling the data in the same machine ReducebyKey will perform first level transformations then the data will be sent over to the reducers for second level of transformation. However coming to GroupbyKey no first level transformation will be applied because of that all the data will be aggregated on key basis in reducer because of it lots of data will be exchanges b/w nodes through Network it causes the network congestion or increases network traffic.
- → groupbyKey + foldleftofValues = reducebyKey so when ever you want to use groupbyKey with map operation use ReducebyKey for better performance.

Serialization and De-serialization



Serialization is required when you want to write object to disk or when you want to send object from one computer to another over network. Once the data is serialized then if you want to convert back into object state then you need to de-serialize it

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Serialization and Deserialization plays an important role in distributed systems like spark. They play important role in 2 places
1) Storing data in serialized form(Cache)
2) Transferring data through nodes(shuffling)
There are two types of serialization supported in Spark:
1) Java(default)
2) Kyro

By default spark supports Java serialization and it serialize objects into Objectoutputstream it can work with any class that implement java.io.serializable. Java serialization is fixable but its quite slow, and leads to large serialized formats for many classes because

There are three considerations in tuning **memory usage: the amount of memory used by your objects** (you may want your entire dataset to fit in memory), **the cost of accessing those objects**, and the **overhead of garbage collection** (if you have high turnover in terms of objects).

By default, Java objects are fast to access, but can easily consume a factor of 2-5x more space than the "raw" data inside their fields. This is due to several reasons:

- Each distinct Java object has an "object header", which is about 16 bytes and contains information such as a pointer to its class. For an object with very little data in it (say one Int field), this can be bigger than the data.
- Java Strings have about 40 bytes of overhead over the raw string data (since they store it in an array of Chars and keep extra data such as the length), and store each character as two bytes due to String's internal usage of UTF-16 encoding. Thus a 10-character string can easily consume 60 bytes.
- Common collection classes, such as HashMap and LinkedList, use linked data structures, where there is a "wrapper" object for each entry (e.g. Map.Entry). This object not only has a header, but also pointers (typically 8 bytes each) to the next object in the list.
- Collections of primitive types often store them as "boxed" objects such as java.lang.Integer.

When your objects are still too large to efficiently store despite this tuning, a much simpler way to reduce memory usage is to store them in serialized form, using the serialized StorageLevels in the RDD persistence API, such as MEMORY_ONLY_SER. Spark will then store each RDD partition as one large byte array. The only downside of storing data in serialized form is slower access times, due to having to reserialize each object on the fly.

Spark can also use the Kryo library (version 4) to serialize objects more quickly. Kryo is significantly faster and more compact than Java serialization (often as much as 10x), but does not support all Serializable types and requires you to register the classes you'll use in the program in advance for best performance.
You can switch to using Kryo by initializing your job with a SparkConf and calling conf.set("**spark.serializer", "org.apache.spark.serializer.KryoSerializer**"). This setting configures the serializer used for not only shuffling data between worker nodes but also when serializing RDDs to disk. The only reason Kryo is not the default is because of the custom registration requirement, but we recommend trying it in any network-intensive application. **Since Spark 2.0.0, we internally use Kryo serializer when shuffling RDDs with simple types, arrays of simple types, or string type.**

Spark automatically includes Kryo serializers for the many commonly-used core Scala classes covered in the AllScalaRegistrar from the Twitter chill library.

To register your own custom classes with Kryo, use the registerKryoClasses method.

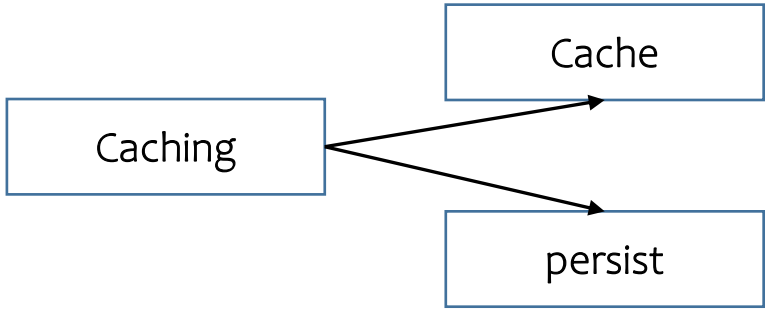
```
val conf = new SparkConf().setMaster(...).setAppName(...)
conf.registerKryoClasses(Array(classOf[MyClass1], classOf[MyClass2]))
val sc = new SparkContext(conf)
```

If your objects are large, you may also need to increase the **spark.kryoserializer.buffer config**. This value needs to be large enough to hold the largest object you will serialize.

highly recommend using Kryo if you want to cache data in serialized form, as it leads to much smaller sizes than Java serialization (and certainly than raw Java objects).

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Understanding Spark Cache Mechanism



Caching is optimization technique for iterative and interactive computations in Spark. There are two ways to cache the data in spark either by using cache() or persist() on a RDD. When you apply cache on RDD spark will store intermediate data inside cache instead in in-memory(RAM). Cache operations are also lazy like Transformation that means until Action is triggered cache will not happen. If cluster contains enough cache memory then entire intermediate data will fit into it but if it doesn't have it then data has to spill over into disk. There 4 different Storage level to control this mechanism.

MEMORY_ONLY: This storage level, RDD is stored as deserialized Java object in the cache. If the size of RDD is greater than memory, It will not cache some partition and recompute them next time whenever needed. In this level the space used for storage is very high, the CPU computation time is low, the data is stored in-memory. It does not make use of the disk.

MEMORY_AND_DISK: In this level, RDD is stored as deserialized Java object in the JVM. When the size of RDD is greater than the size of memory, it stores the excess partition on the disk, and retrieve from disk whenever required. In this level the space used for storage is high, the CPU computation time is medium, it makes use of both in-memory and on disk storage.

MEMORY_ONLY_SER: This level of Spark store the RDD as serialized Java object (one-byte array per partition). It is more space efficient as compared to deserialized objects, especially when it uses fast serializer. But it increases the overhead on CPU. In this level the storage space is low, the CPU computation time is high and the data is stored in-memory. It does not make use of the disk.

MEMORY_AND_DISK_SER: It is similar to MEMORY_ONLY_SER, but it drops the partition that does not fits into memory to disk, rather than recomputing each time it is needed. In this storage level, The space used for storage is low, the CPU computation time is high, it makes use of both in-memory and on disk storage.

DISK_ONLY: In this storage level, RDD is stored only on disk. The space used for storage is low, the CPU computation time is high and it makes use of on disk storage.

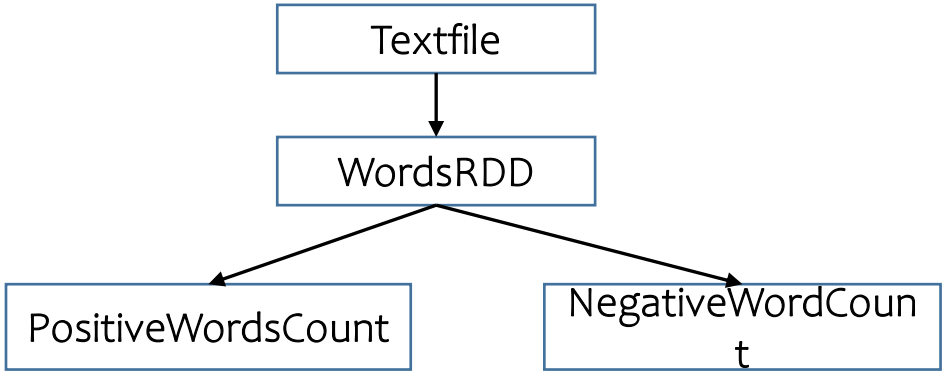
The difference b/w cache and persist is that cache() will cache the RDD into memory, whereas persist(level) can cache in memory, on disk, or off-heap memory according to the caching strategy specified by level. persist() without an argument is equivalent with cache(). Freeing up space from the Storage memory is performed by **unpersist().**

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Lets Understanding Spark Cache Mechanism with Example

Without Cache

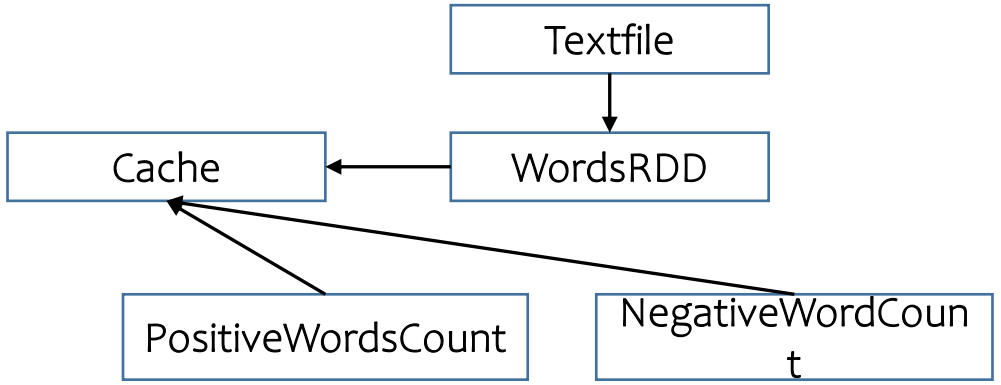
```
val text File = sc.textFile("/user/emp.txt")
val wordsRDD = textFile.flatMap(line => line.split("\\W"))
val positiveWordsCount = wordsRDD.filter(word => isPositive(word)).count()
val negativeWordsCount = wordsRDD.filter(word => isNegative(word)).count()
```



Lineage

With Cache

```
val text File = sc.textFile("/user/emp.txt")
val wordsRDD = textFile.flatMap(line => line.split("\\W"))
wordsRDD.cache()
val positiveWordsCount = wordsRDD.filter(word => isPositive(word)).count()
val negativeWordsCount = wordsRDD.filter(word => isNegative(word)).count()
```



Lineage

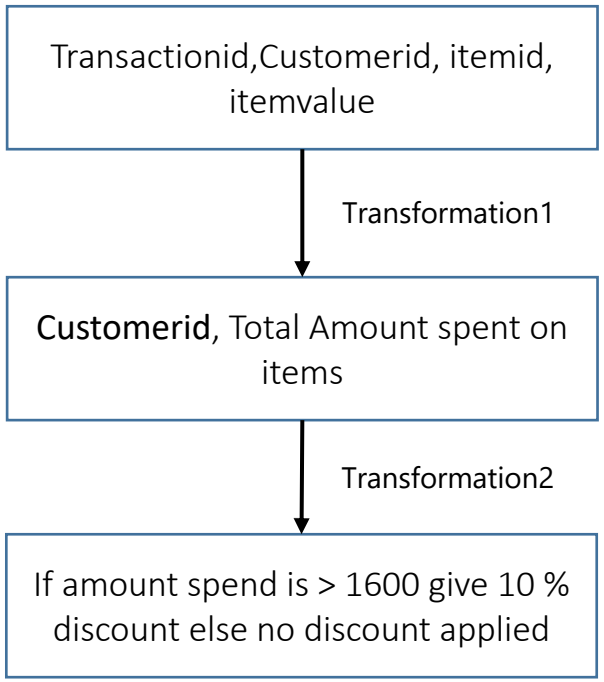
If you look at the lineage of spark application without cache creates multiple child branches for WordsRDD(ParentRDD) they are PositiveWordsCount(Child) and NegativeWordCount(Child) so when ever a new branch (Transformation) is created and executed then parentRDD will reload into the memory for every branch.so when you think RDD is reused in multiple operation they look the same then cache the data so that the cached data will be re-used instead of reloaded which is shown the figure with Cache.

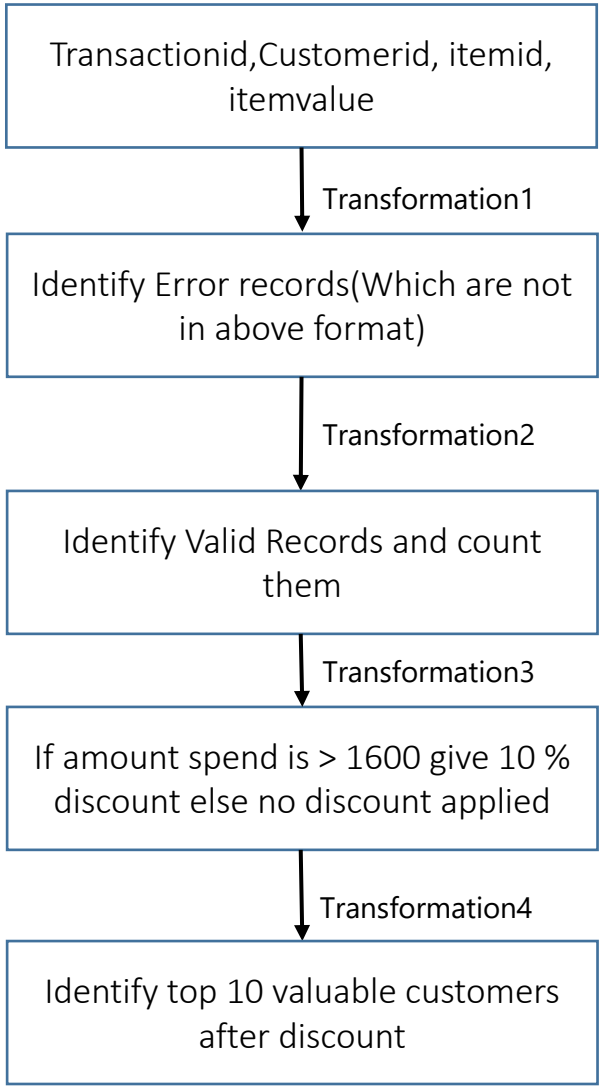
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

1) Item Filter
2) Item wise count

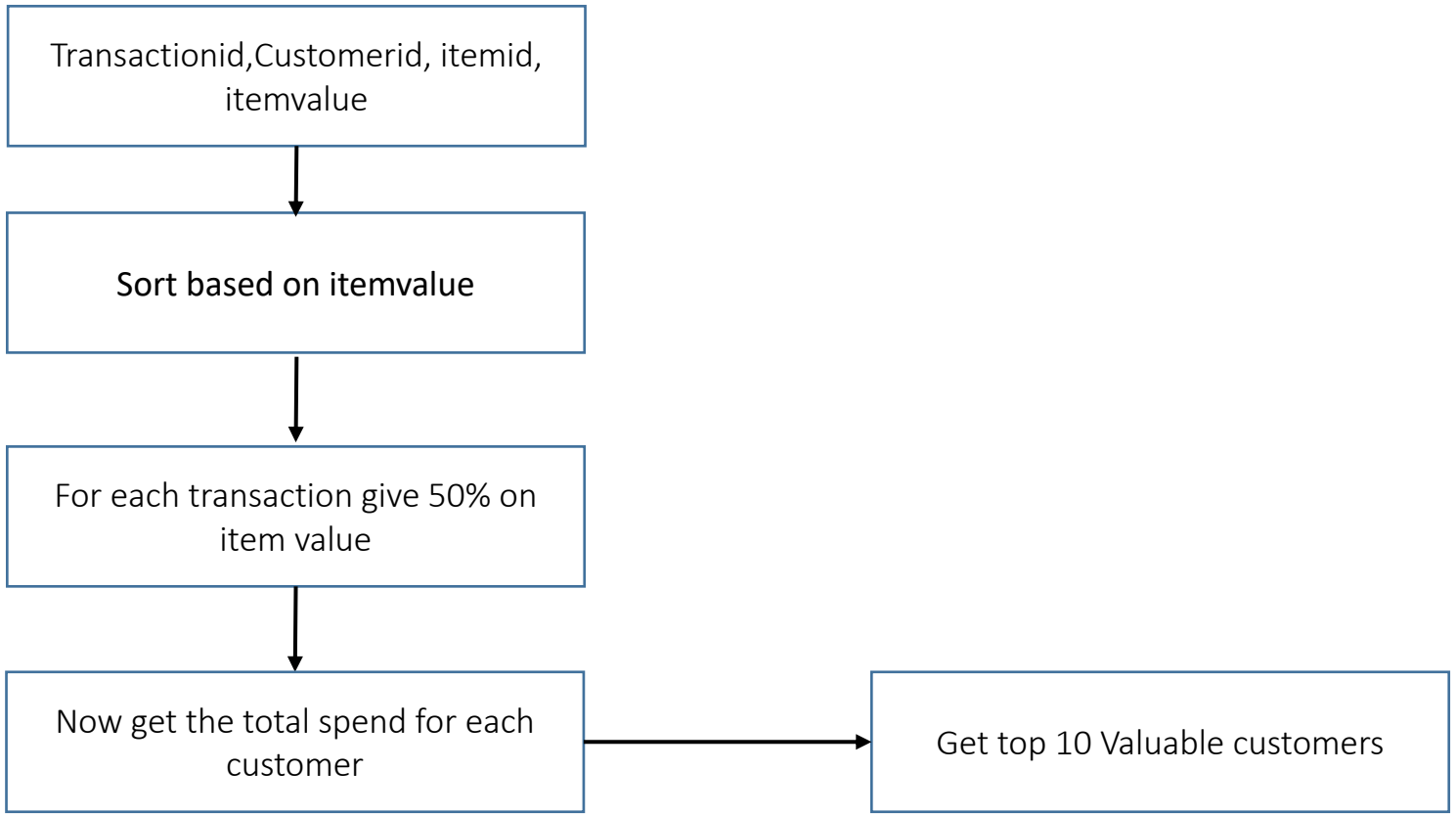
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Understanding Amount wise Algorithm with Java Serialization



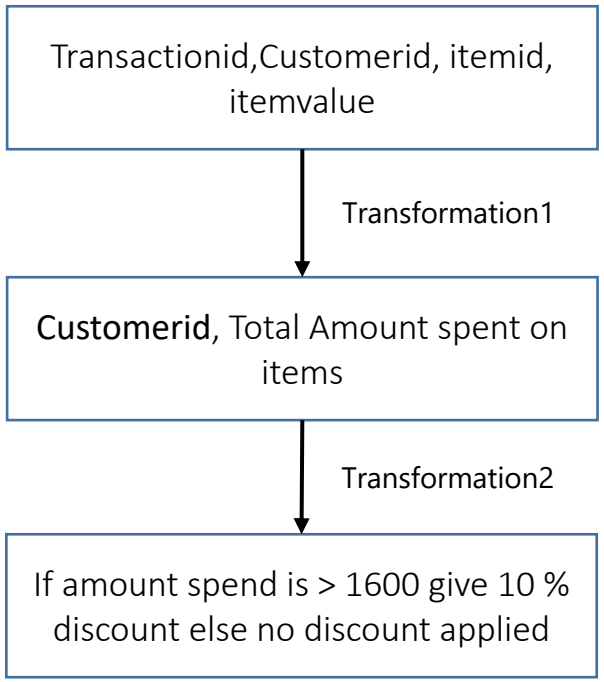


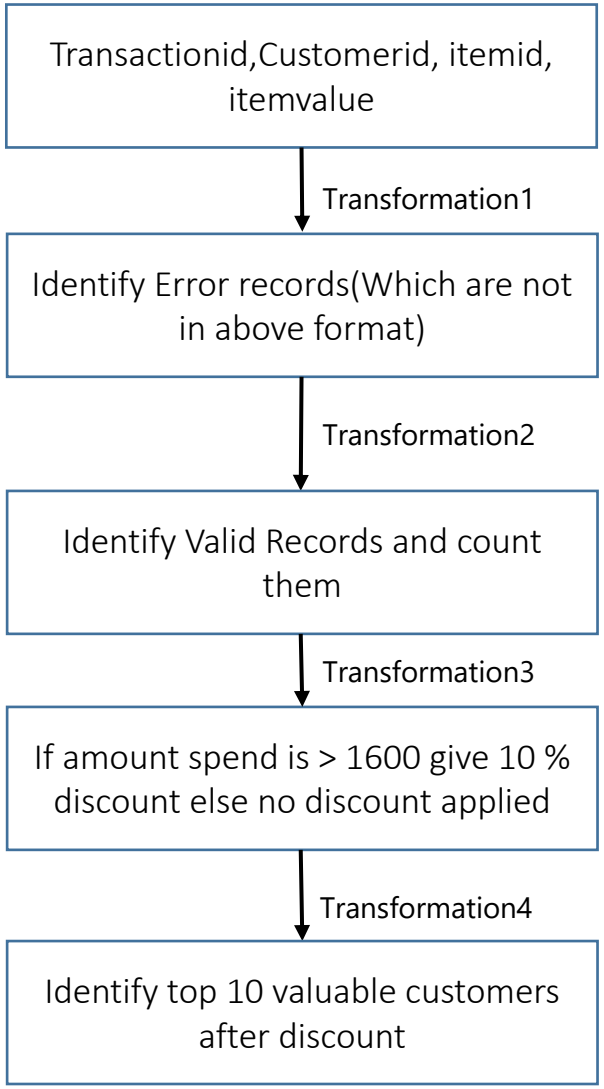
Understanding Item wise Algorithm with Java Serialization



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Understanding Amount wise Algorithm with Kyro Serialization



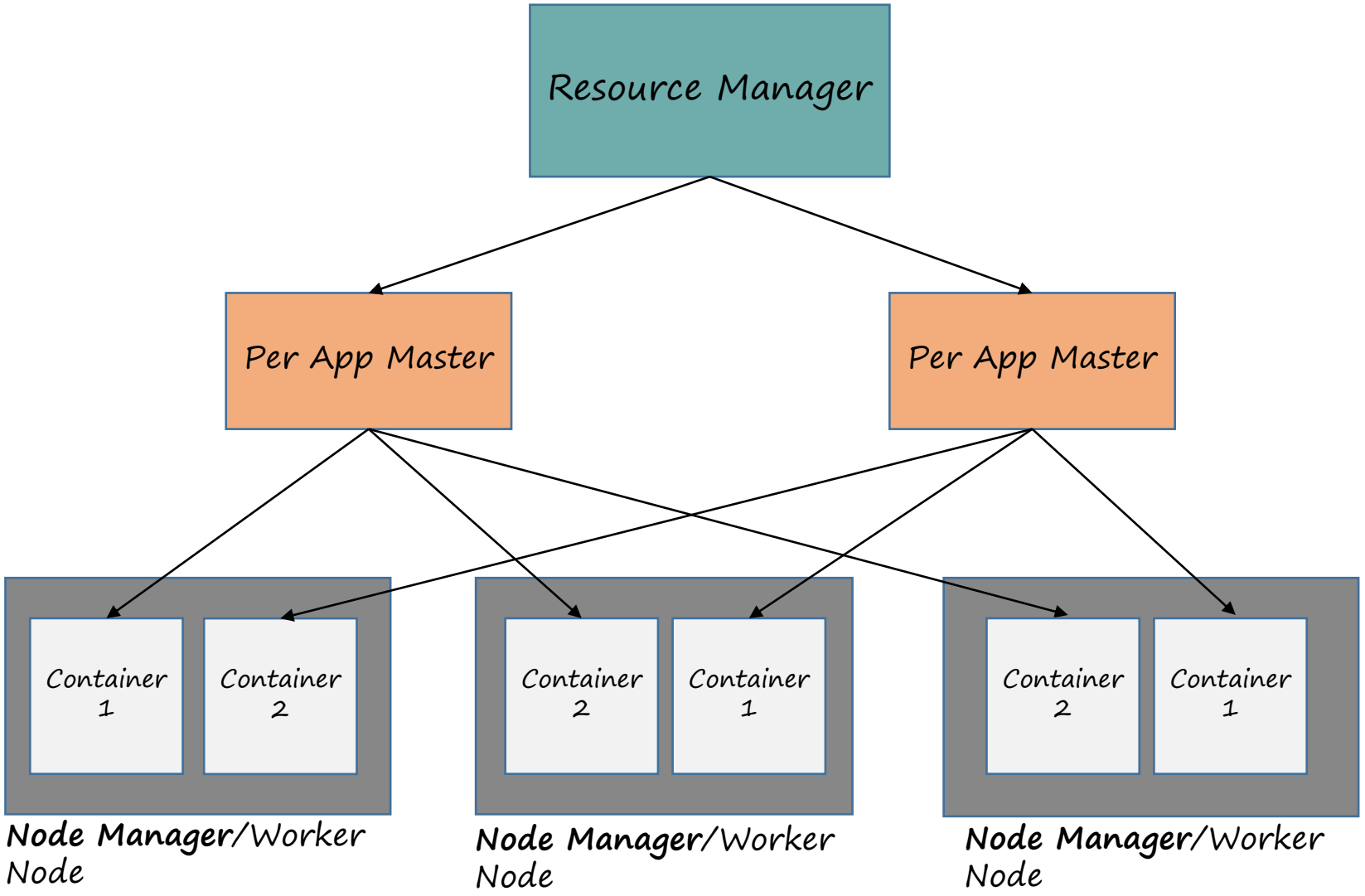


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark -YARN

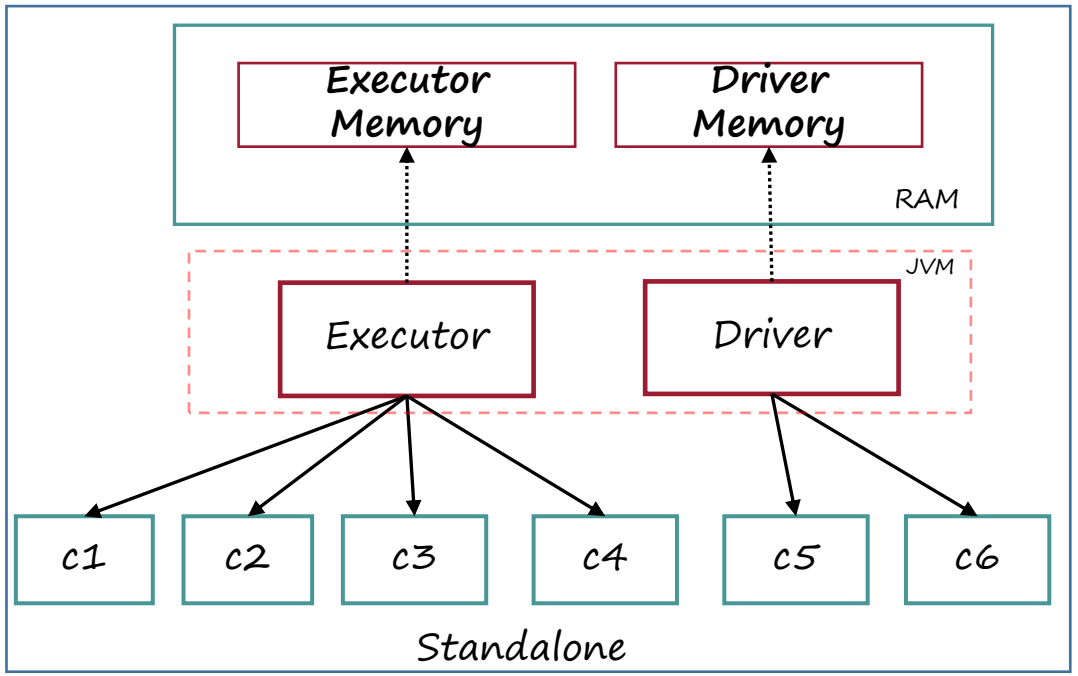
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

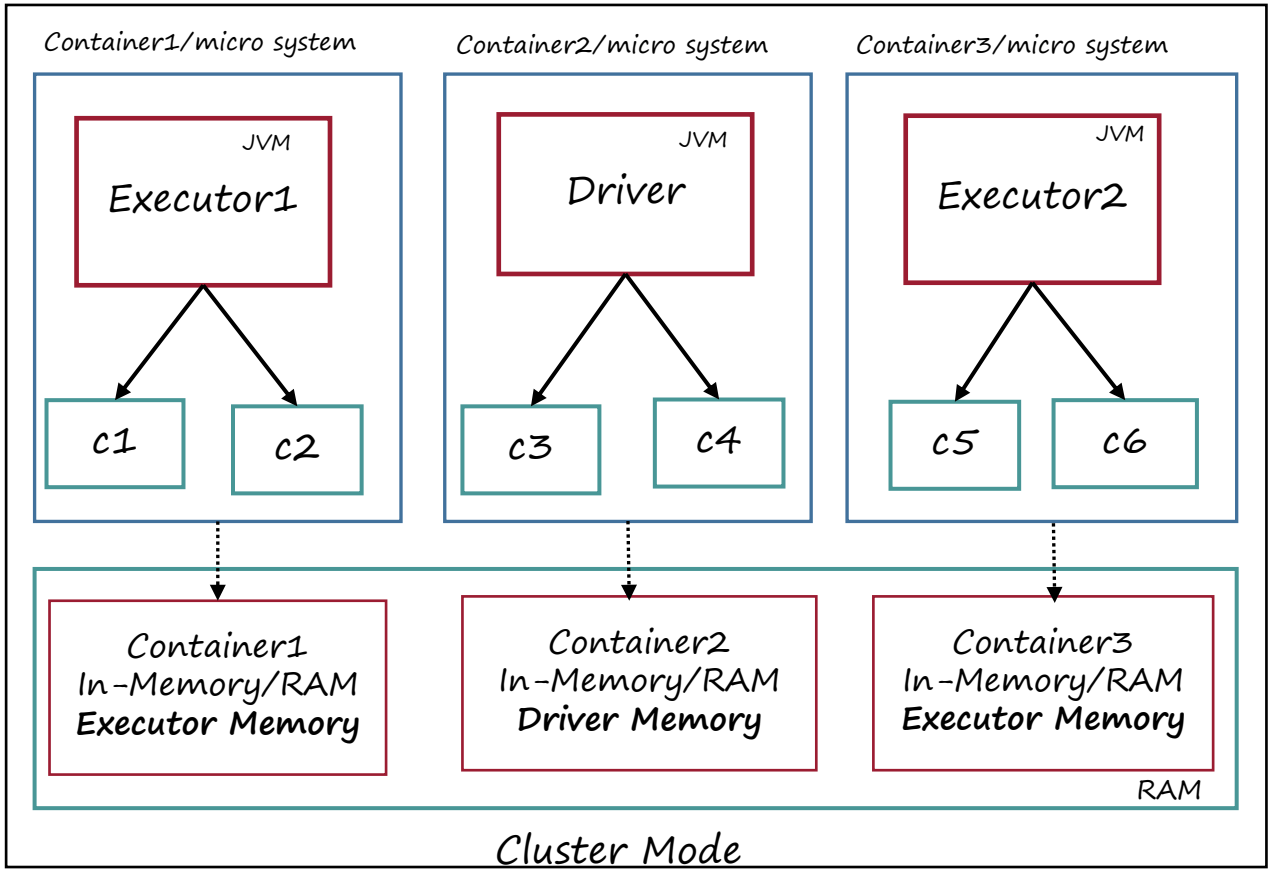
High level view of YARN Architecture



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Computer/Node/Server View in Standalone and Cluster Modes



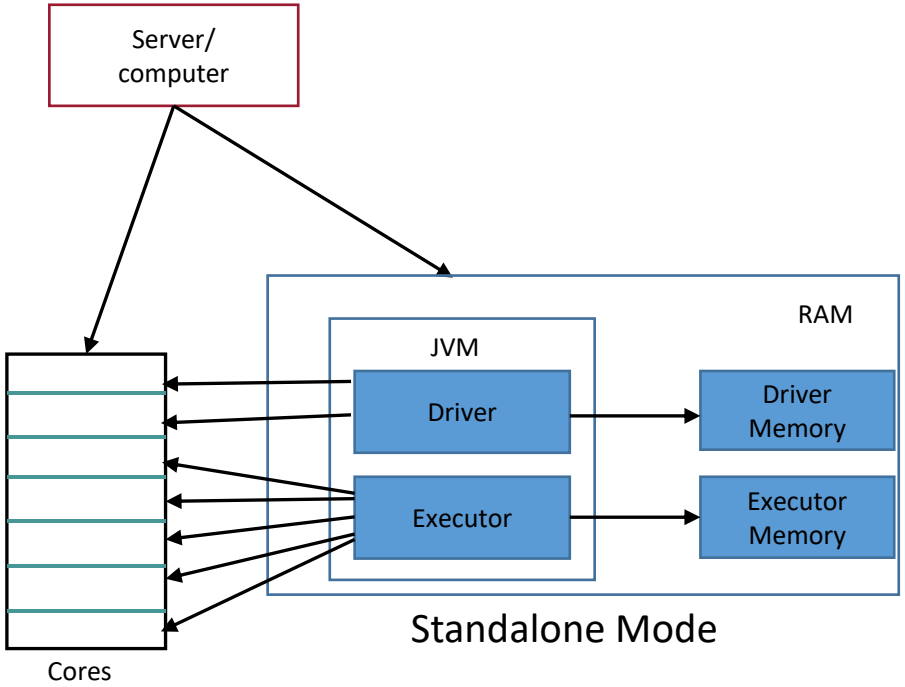


In Standalone mode both Driver and Executor runs with in a same JVM/server. Parallelism depends on the number of partitions and number of cores say you have only 4 cores allocated then executor can run only 4 parallel tasks at a time the partition count will be 4 * 4 = 16 and Task count will be also 16.

In Cluster mode say YARN multiple containers/mini systems will be launched with in the system/server/node and the computation resources will be shared among them. Spark uses power of YARN/Mesos and it launches single executor with dedicated cores/memory with in each container. Each executor will handle multiple task. Here parallelism depends on number of executors core say you have 10 node cluster with 16 cores each then your partition count can 150*4=600 paritions and number of parallel task will be 150 and 150 tasks will be shared across multiple executors. We will discuss this in detail in further slides.

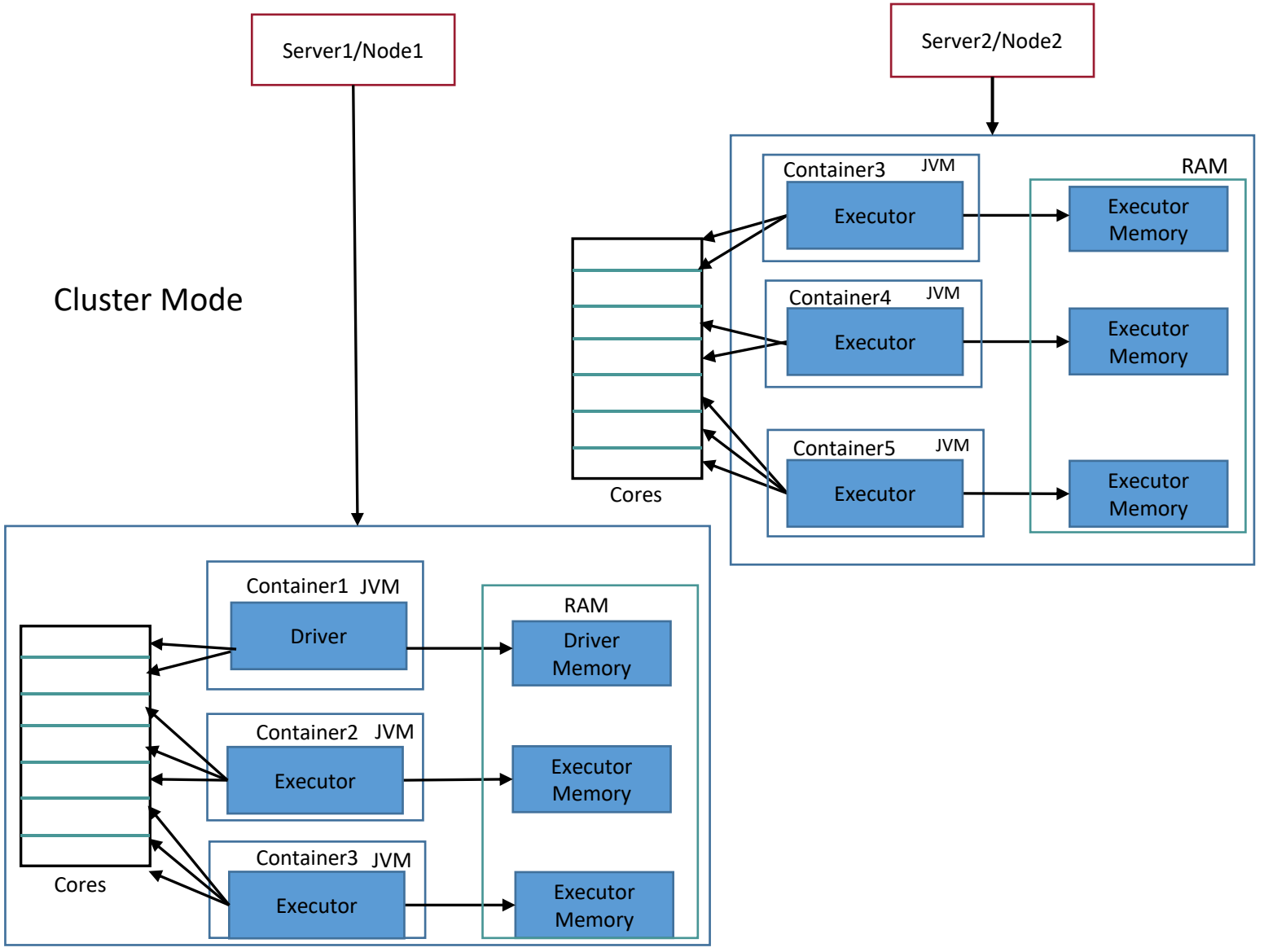
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

High level view of Standalone and Cluster Mode



Standalone Mode

Cluster Mode

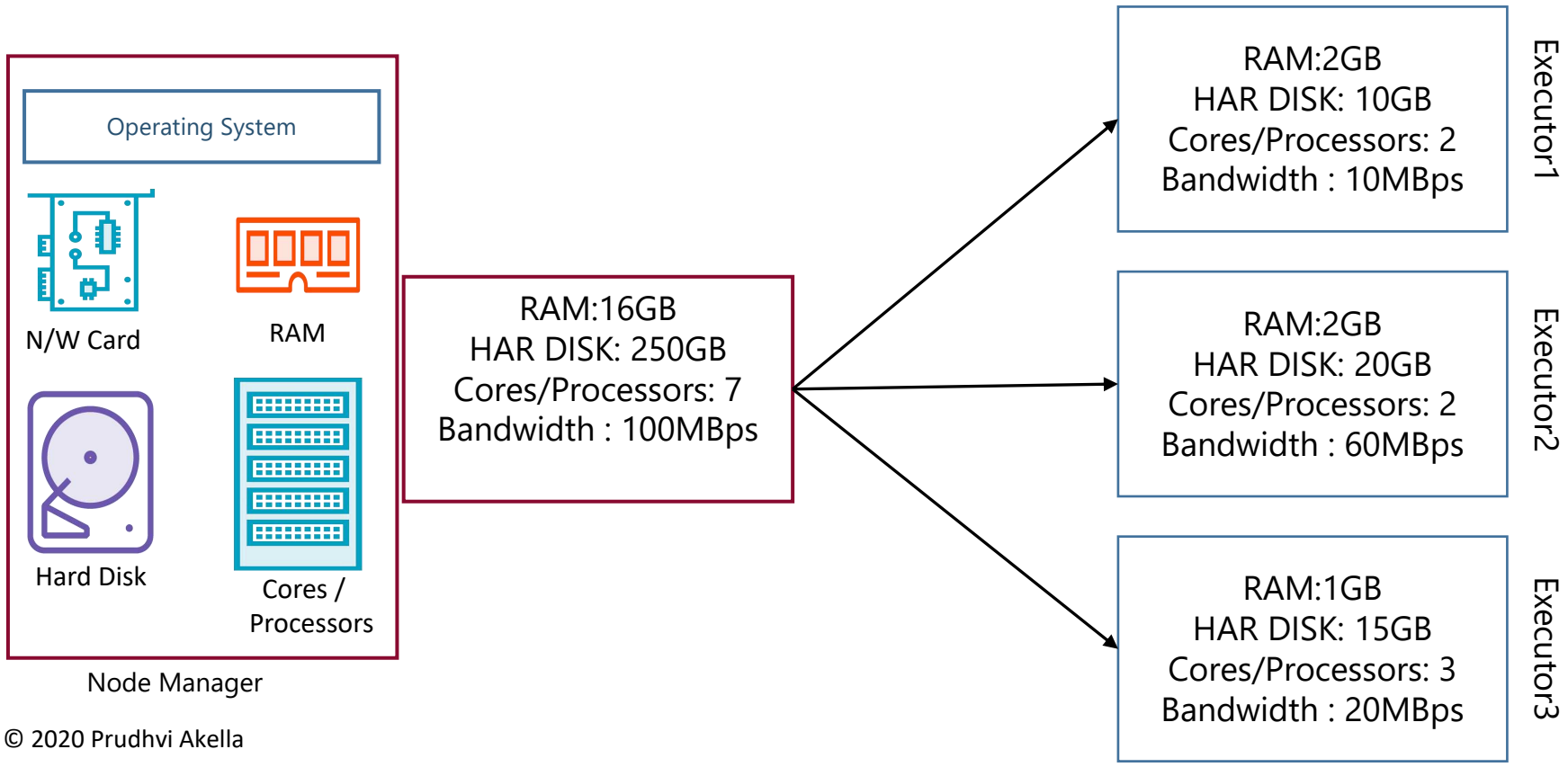


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

YARN Container?

Yarn container executes single unit of work and it will take care of execution of single entity like either map or reduce.
A container is supervised by node manager and scheduled by Resource Manager.

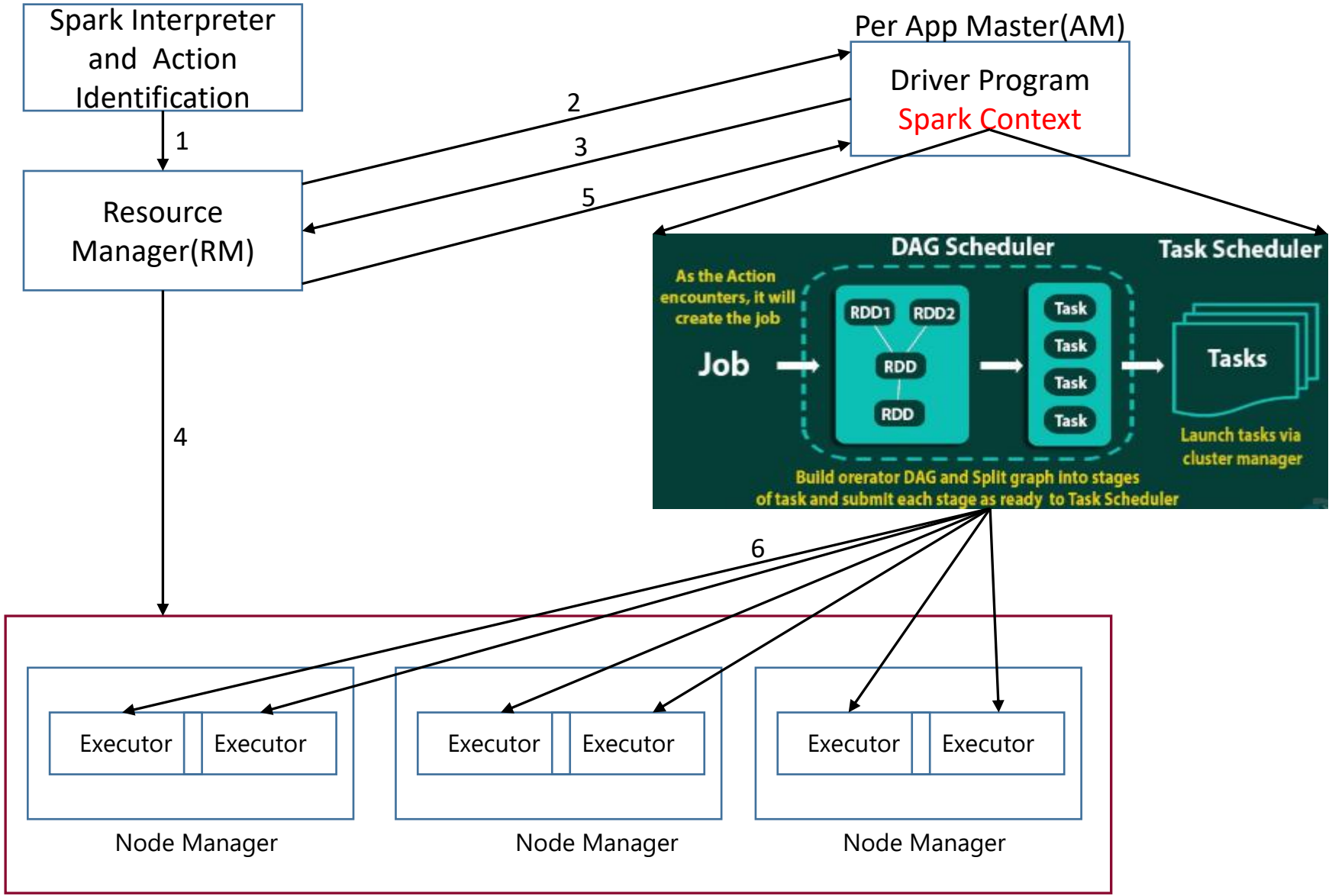
Spark Executors are used by spark to execute spark task. In YARN executors are launched as yarn containers in worker nodes/NodeManager.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Internals of Job Execution in Spark In YARN Aspect

Step1:Spark Interpreter is the first layer. It will interprets the code and creates a operator graph once the Action is identified it will request RM to run job along with operator graph or RDD Linage.
Step2: Then RM will create a AM container and launches the Driver Program. Where it create a Spark Context using Spark Conf . Once it is created a DAG Scheduler will be created and operator graph will send as input and it is responsible to creating RDD lineage graph which is used by spark for executing transformation that's the reason even though in case of failures spark uses DAG to re-execute transformation and DAG will converted to Stages and Tasks for physical execution and task will be scheduled by Task Scheduler
Step3: Once the driver Program is created AM will request RM to allocate the resources for execution
Step4: RM will instruct the Node Manager to create containers and launch the executors(JVM Process)
Step5:Once the Executors are ready RM will response saying to AM they are ready
Step6: Then AM's Task manager will start running the Task in Executors.

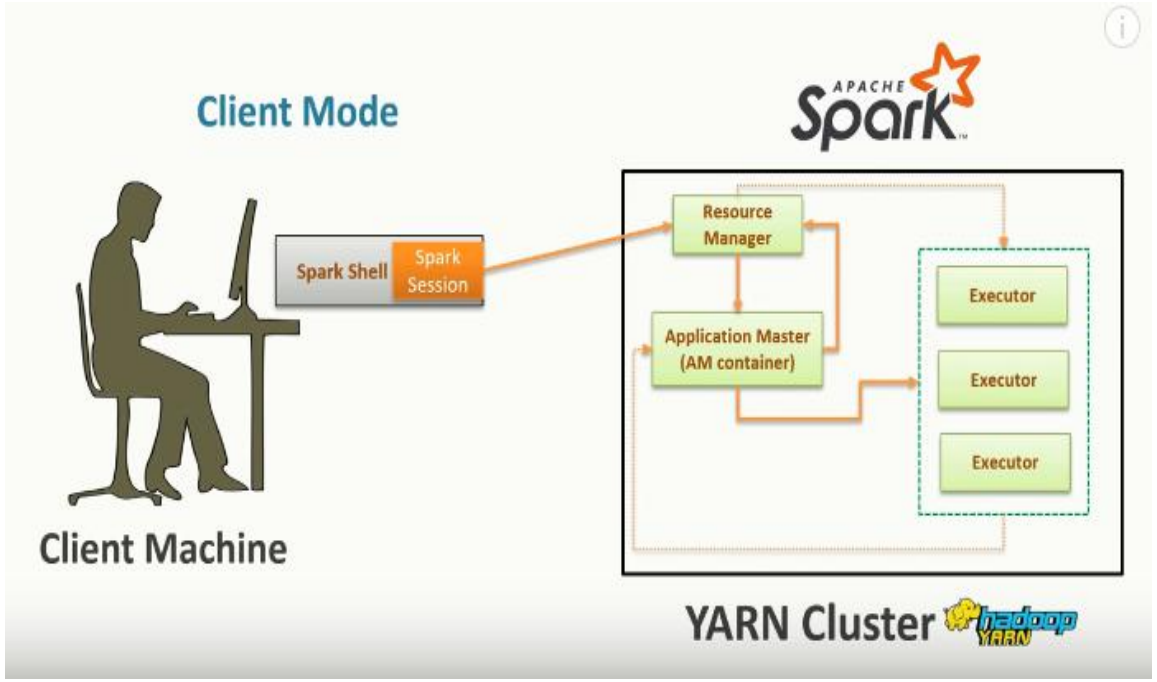


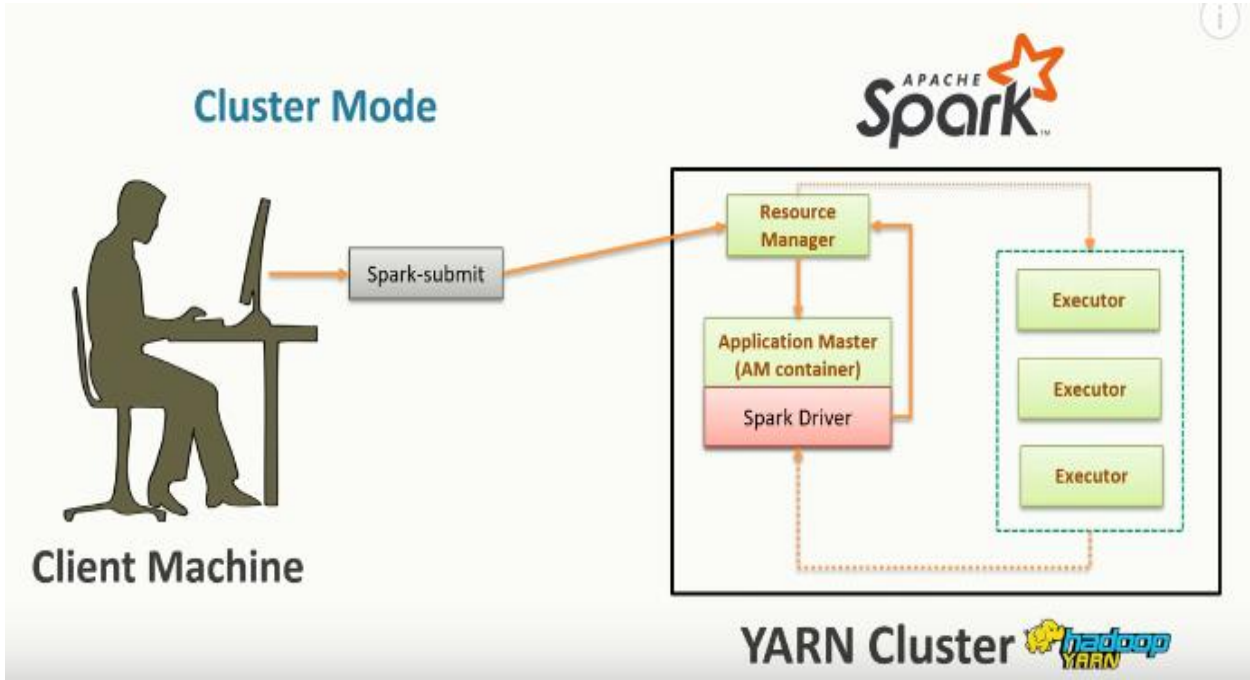
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Cluster (YARN)

Client Mode

Cluster Mode





Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Lets Execute Word count in Cluster Mode using YARN

1) Start HDFS Service from Command Line
   → sudo service hadoop-hdfs-namenode start
   → sudo service hadoop-hdfs-secondarynamenode start
   → sudo service hadoop-hdfs-datanode start

2) Start YARN Service from Command Line
   → sudo service hadoop-yarn-resourcemanager start
   → sudo service hadoop-yarn-nodemanager start

3) Install sbt(Scala Build Tool) which is already installed now lets build the sbt project to generate the jar file which is used to launch the spark job in cluster mode in YARN.

→ Open command line
→ Cd to location of build.sbt file in Word count case it is
cd /home/cloudera/projects/spark-core
→**sbt compile**(Compliation starts and wait for sometime)
→**sbt package** (This step will create a executable jar file)
The Jar file will be generated to Target folder

```
[info] Packaging /home/cloudera/projects/spark-core/target/scala-2.10/sp
ark-core_2.10-0.1.jar ...
[info] Done packaging.
[success] Total time: 3 s, completed Jan 28, 2020 1:51:21 AM
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

How to submit SPARK JOB in Cluster Mode?

→**spark-submit**
It is used to submit spark jobs in clusters.
Command line arguments:
Class Name: Name of the class along with packages you have mention
Example: org.training.spark.apiexamples.discount.AmountWiseDiscount
master : name of the master
Example: **Yarn**
deploy-mode: Client(Driver will be launched in Local) or Cluster(Driver will be launched as Per App Master)
driver-memory: container RAM Space for Driver program
Example: 4gb
num-executors : used to control the number of YARN containers
Example: 2
executor-memory: How much RAM space should each yarn container(JVM Process) can use
Example: 2g
executor-core: How many cores should each yarn container can use
Example:2
Jar file
Arguments to Program

When ever a job is launched in YARN it creates a unique Application id for it using that you can check the status using below command
Command: yarn logs –applicationId <ID>
Status:
Accepted : It is accepted by Resource Manager but still in Queue no resources are allocated
Running : Resources are allocated to Job and successfully running
Failure : There is some issue either while allocating the resource or running the job usually you see an detailed expection on screen then use the above command to Debug.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Client Mode: As you run spark submit in client mode the driver program will run the local server. So you can able to see the aggregated results on screen once the job is completed and you cannot kill the job until program gets complete if you do so you driver program get kill as it contains the spark context it will be closed and it doesn't have contact with executors.

```
spark-submit --class org.training.spark.apiexamples.discount.AmountWiseDiscount --master yarn --deploy-mode client spark-core_2.10-0.1.jar file:////home/cloudera/projects/spark-core/src/main/resources/sales.csv
```

Cluster Mode: Driver will run in Per Application master you cannot view on the screen for viewing the results you need to go to the Node manager UI(http://localhost:8042/node) and inside the container directory with you application ID you can view the stderr and stdout logs so there you can see the results. Once the job is launched using spark-submit you can interrupt it by CTRL+Z because driver will run in Per AppMaster

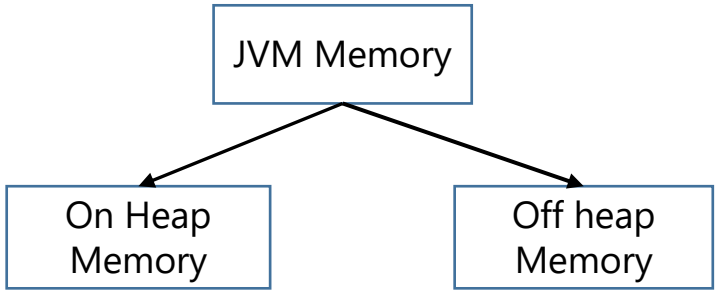
```
spark-submit --class org.training.spark.apiexamples.discount.AmountWiseDiscount --master yarn --deploy-mode cluster spark-core_2.10-0.1.jar file:////home/cloudera/projects/spark-core/src/main/resources/sales.csv
```

Note: As the file is local we are using file:// as extension usually it is not recommended.
If the file is in HDFS you have mention the hdfs://<namenode:host>:<port>/<file path>
If file is in S3 then it should be s3://<hostname>/<bucket>/<file path>

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Lets understand memory management in spark

Ultimately job will be converted to stages and each stage has multiple tasks and tasks has to be executed by executor which are called to be **jvm process**. As they are jvm process they have to adopt jvm memory management.Lets understand a bit about jvm memory management.



On-Heap memory management: Objects are allocated on the JVM heap and bound by GC.

Off-Heap memory management: Objects are allocated in memory outside the JVM by serialization, managed by the application, and are not bound by GC. This memory management method can avoid frequent GC, but the disadvantage is that you have to write the logic of memory allocation and memory release.

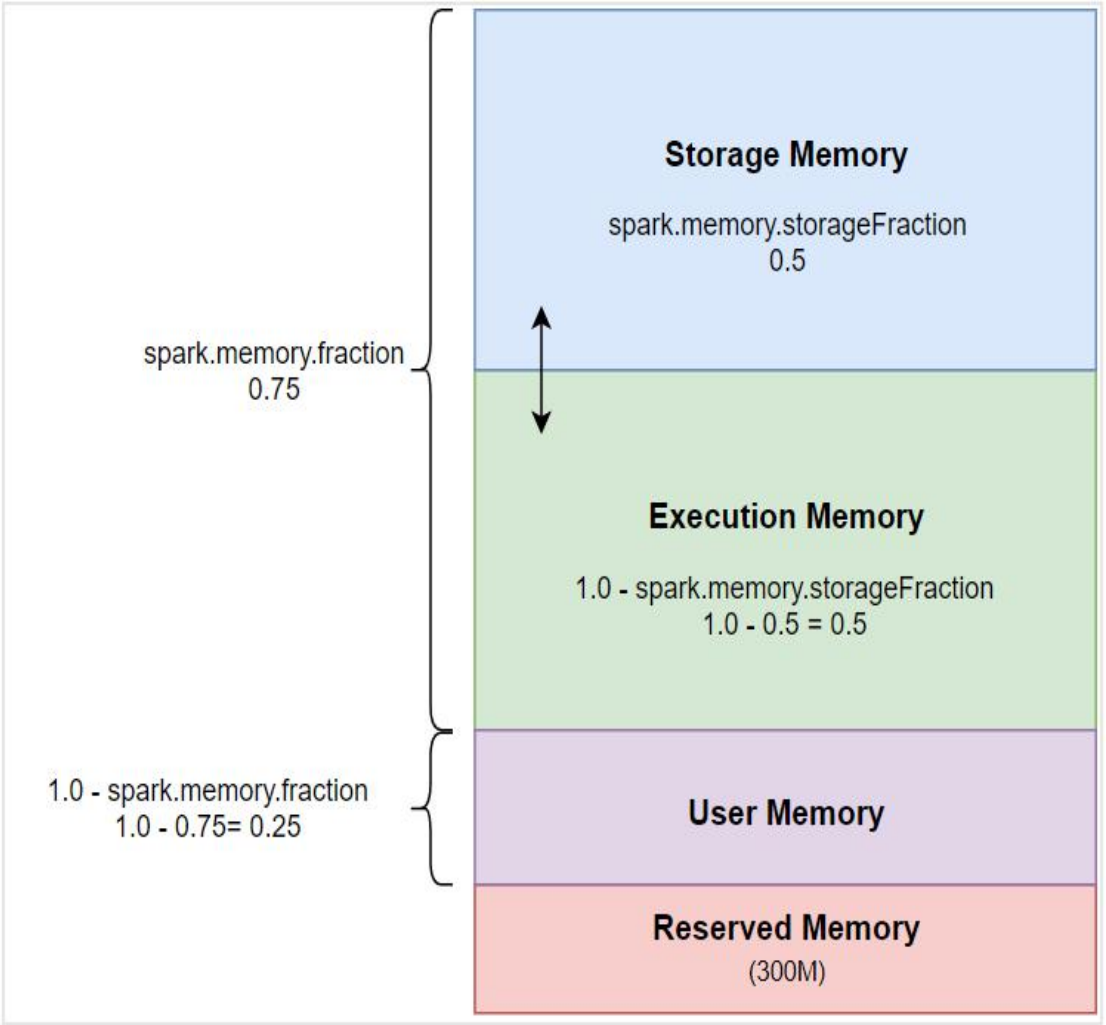
Usually this how read and write order happens
On-heap > off-heap > Disk
If on-heap is full it goes to off-heap then if goes to disk
Note: One heap is faster than off-heap , Off –heap is faster than disk

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

On Heap Memory

By default, Spark uses On-heap memory only. The size of the On-heap memory is configured by the –executor-memory or spark.executor.memory parameter when the Spark Application starts. The concurrent tasks running inside Executor share JVM's On-heap memory.
The On-heap memory area in the Executor can be roughly divided into the following four blocks:

Storage Memory: It's mainly used to store Spark cache data, such as RDD cache, Broadcast variable, Unroll data, and so on.
Execution Memory: It's mainly used to store temporary data in the calculation process of Shuffle, Join, Sort, Aggregation, etc.
User Memory: It's mainly used to store the data needed for RDD conversion operations, such as the information for RDD dependency.
Reserved Memory: The memory is reserved for system and is used to store Spark's internal objects



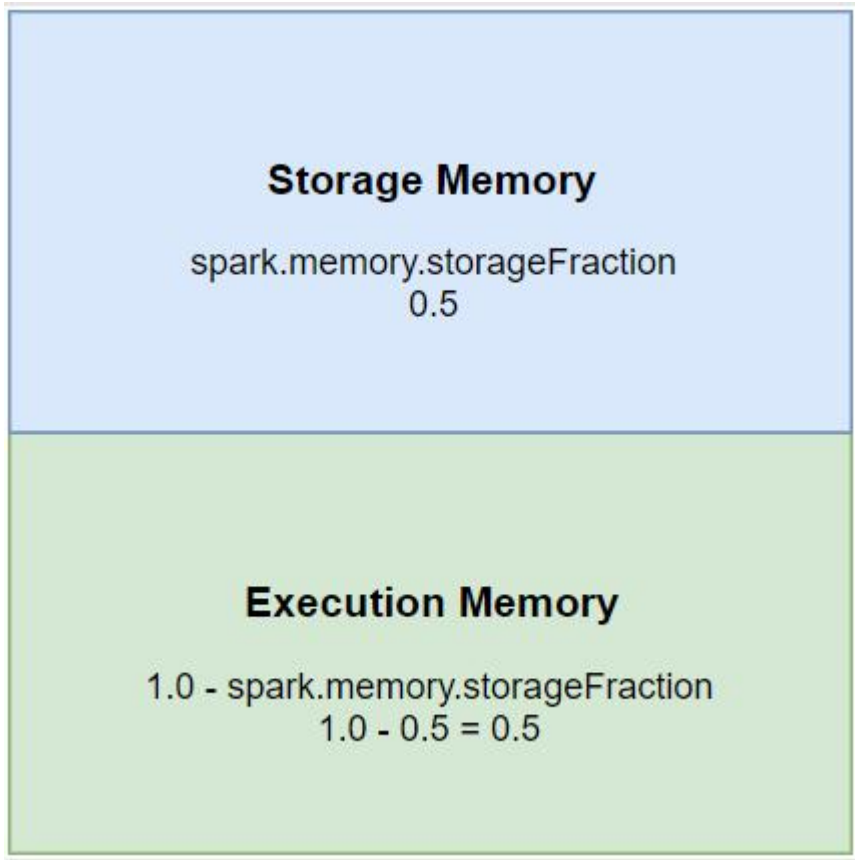
Executors memory = 1GB
spark.memory.fraction (Storage Memory + Executor Memory) = 75% of Executors Memory = 750

Storage Memory(spark.memory.storageFraction) = 50% of spark Memory Fraction
750 *50% = 350
Execution Memory = 100% – Storage Memory(50%) = 50%
750*50% = 350

User Memory = 100% - spark.memory.fraction(75%) = 25% of Executors Memory = 250

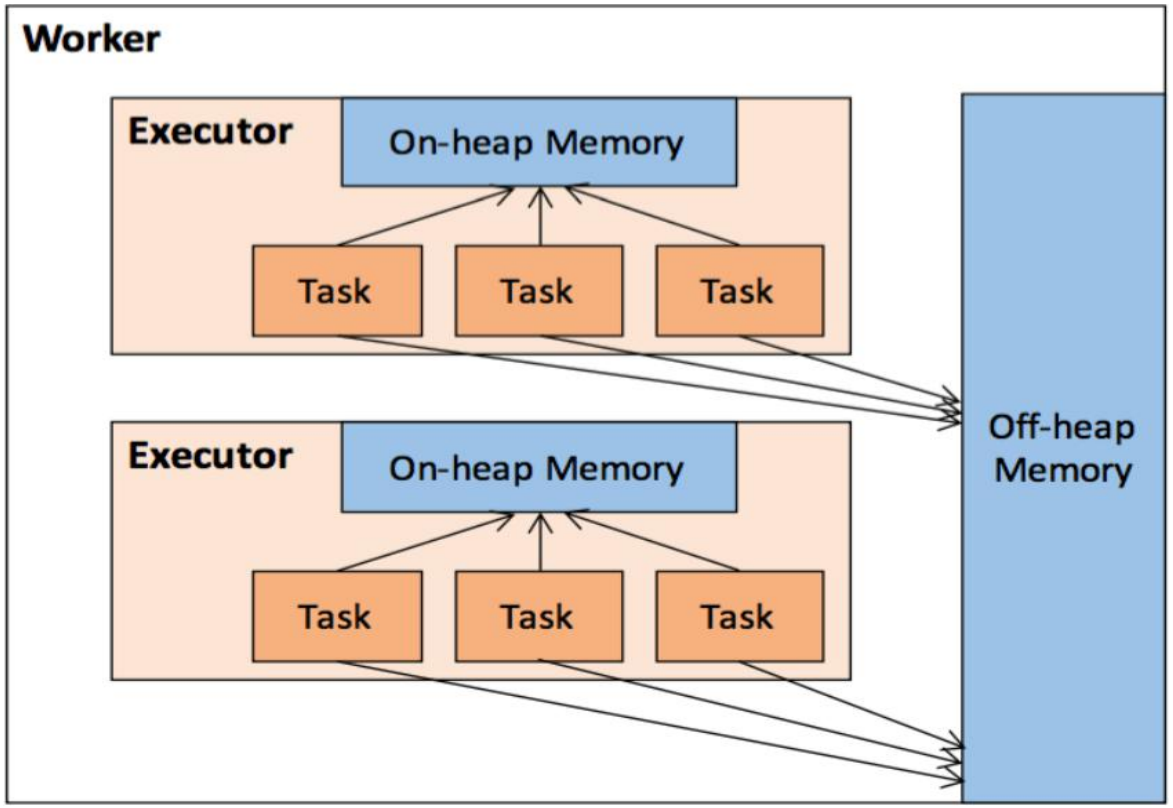
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark 1.6 began to introduce Off-heap memory .By default, Off-heap memory is disabled, but we can enable it by the **spark.memory.offHeap.enabled** parameter, and set the memory size by **spark.memory.offHeap.size** parameter. Compared to the On-heap memory, the model of the Off-heap memory is relatively simple, including only Storage memory and Execution memory.

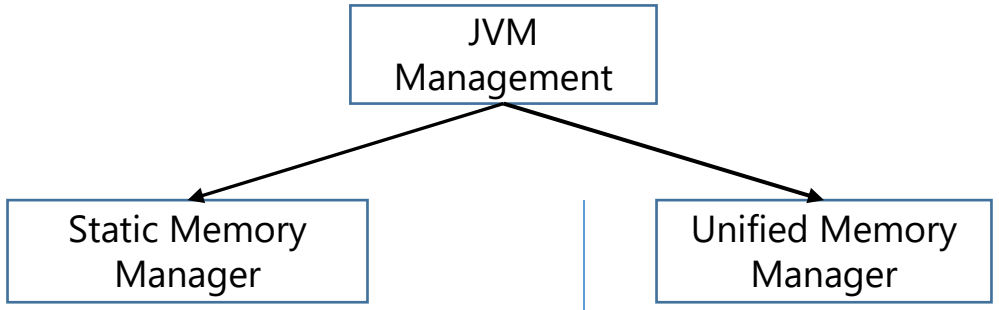


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

If the Off-heap memory is enabled, there will be both On-heap and Off-heap memory in the Executor. At this time, the Execution memory in the Executor is the sum of the Execution memory inside the heap and the Execution memory outside the heap. The same is true for Storage memory. The following picture shows the on-heap and off-heap memory inside and outside of the Spark heap.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Under the Static Memory Manager mechanism, the size of Storage memory, Execution memory, and other memory is fixed during the Spark application's operation, but users can configure it before the application starts. Though this allocation method has been eliminated gradually, Spark remains for compatibility reasons.

Here mainly talks about the drawbacks of Static Memory Manager: the Static Memory Manager mechanism is relatively simple to implement, but if the user is not familiar with the storage mechanism of Spark, or doesn't make the corresponding configuration according to the specific data size and computing tasks, it is easy to cause one of the Storage memory and Execution memory has a lot of space left, while the other one is filled up first—thus it has to be eliminated or removed the old content for the new content.

The Unified Memory Manager mechanism was introduced after Spark 1.6. The difference between Unified Memory Manager and Static Memory Manager is that under the Unified Memory Manager mechanism, the Storage memory and Execution memory share a memory area, and both can occupy each other's free area.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

spark-submit --class <CLASS_NAME> --num-executors ? --executor-cores ? --executor-memory ?

Following list captures some recommendations to keep in mind while configuring them:

Hadoop/Yarn/OS Deamons:

When we run spark application using a cluster manager like Yarn, there'll be several daemons that'll run in the background like **NameNode, Secondary NameNode, DataNode, Resource Manager and NodeManager**. So, while specifying num-executors, we need to make sure that we leave aside enough cores **(~1 core per node)** for these daemons to run smoothly.

Yarn ApplicationMaster (AM):

ApplicationMaster is responsible for negotiating resources from the ResourceManager and working with the NodeManagers to execute and monitor the containers and their resource consumption. If we are running spark on yarn, then we need to budget in the resources that AM would need **(~1024MB and 1 Core)**.

HDFS Throughput:

HDFS client has trouble with tons of concurrent threads. It was observed that HDFS achieves full write throughput with **~5 tasks per executor** . So it's good to keep the number of cores per executor below that number.

MemoryOverhead:

The value of the spark.yarn.executor.memoryOverhead property is added to the executor memory to determine the full memory request to YARN for each executor. It defaults to **max(7% of executors memory, with minimum of 384).**

Full memory requested to yarn per executor = spark-executor-memory + spark.yarn.executor.memoryOverhead.

spark.yarn.executor.memoryOverhead = Max(384MB, 7% of spark.executor-memory)

So, if we request 20GB per executor, AM will actually get 20GB + memoryOverhead = 20 + 7% of 20GB = ~21.4GB memory for us.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Cluster Config:

Number of Nodes : 10

Cores per each Node : 16

RAM per Node : 64GB

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

First Approach: Tiny executors [One Executor per core]:

Tiny executors essentially means one executor per core. Following table depicts the values of our spark config params with this approach.

--**num-executors** = n this approach, we'll assign one executor per core
= total-cores-in-cluster
= num-cores-per-node * total-nodes-in-cluster
= 16 x 10 = 160

--**executor-cores** = 1 (one executor per core)

--**executor-memory** = amount of memory per executor
= mem-per-node/num-executors-per-node
= 640GB/160 = 4GB + 7% of overhead = ? Storage Fraction =
On heap per executor = 3.6GB * 0.75 = 2.71G = Memory Fraction =>Storage Memory=50 % of Storage Fraction = 1.35GB and Execution Memory = 1.35GB UserMemory = 1GB
Off heap per Executor = 384MB => Memory Fraction => Storage Memory = 192MB Executor Memory = 192M

Analysis: With only one executor per core, as we discussed above, we'll not be able to take advantage of running multiple tasks in the same JVM. Also, shared/cached variables like broadcast variables and accumulators will be replicated in each core of the nodes which is 16 times. Also, we are not leaving enough memory overhead for Hadoop/Yarn daemon processes and we are not counting in **ApplicationManager**. NOT GOOD!

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Second Approach: Fat executors (One Executor per node):

Fat executors essentially means one executor per node. Following table depicts the values of our spark-config params with this approach:

--num-executors = In this approach, we'll assign one executor per node
= total-nodes-in-cluster
= 10

--executor-cores = one executor per node means all the cores of the node are assigned to one executor
= total-cores-in-a-node
= 16

--executor-memory = amount of memory per executor
= mem-per-node/num-executors-per-node
= 64GB/1 = 64GB

Analysis: With all 16 cores per executor, apart from ApplicationManager and daemon processes are not counted for, HDFS throughput will hurt and it'll result in excessive garbage results. Also, **NOT GOOD!**

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Third Approach: Balance between Fat (vs) Tiny:

According to the recommendations which we discussed above:

So we might think, more concurrent tasks for each executor will give better performance. But research shows that any application with more than 5 concurrent tasks, would lead to a bad show. So the optimal value is 5.

Leave 1 core per node for Hadoop/Yarn daemons => Num cores available per node = 16-1 = 15

So, Total available of cores in cluster = 15 x 10 = 150

Number of available executors = (total cores/num-cores-per-executor) = 150/5 = 30

Leaving 1 executor for ApplicationManager => **--num-executors = 29**

Number of executors per node = 30/10 => **3 executors per Node**

Memory per executor = 64GB/3 = 21GB

Counting off heap overhead = 7% of 21GB = 1.47GB. So, actual **--executor-memory = 21 - 1.47 = 19.5GB = 19.5 -300MB(Reserved Memory) = 19.2GB**

So, recommended config is: 29 executors, 19.2GB memory each and 5 cores each!!

Analysis: It is obvious as to how this third approach has found right balance between Fat vs Tiny approaches. Needless to say, it achieved parallelism of a fat executor and best throughputs of a tiny executor!!

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

```
spark-submit  \
--class org.training.spark.apiexamples.discount.AmountWiseDiscount  \
--master yarn  \
--deploy-mode cluster \
--driver-cores 2 \
--driver-memory 1G \
--num-executors 29 \
--executor-cores 5 \
--executor-memory 18G \
spark-core_2.10-0.1.jar file:///home/cloudera/projects/spark-core/src/main/resources/sales.csv
```

Note: If you don't pass
driver memory by default it's 1024MB/1GB and driver core is 1 in yarn in cluster mode it will be controlled by --driver-memory and driver-cores.
Default Executor memory is 2048MB/2GB and Number of cores is 2

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

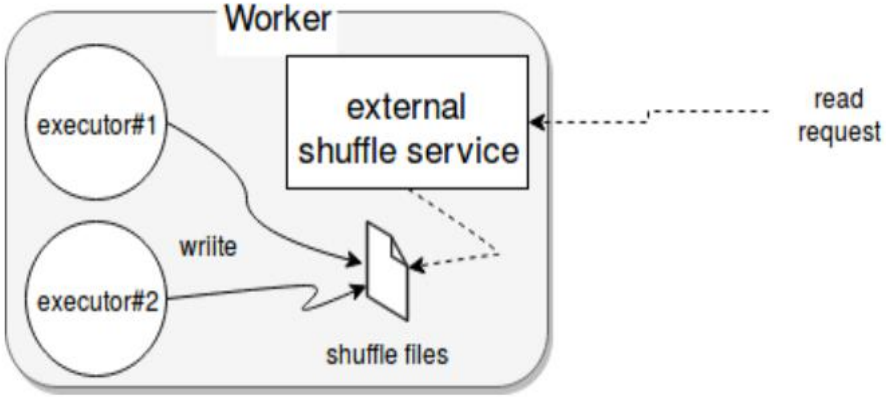
Say if you want to allocate the executors on fly after submitting the job. Spark provides a mechanism to dynamically adjust the resources your application occupies based on the workload. This means that your application may give resources back to the cluster if they are no longer used and request them again later when there is demand. This feature is particularly useful if multiple applications share resources in your Spark cluster.

There are two requirements for using this feature:

1) spark.dynamicAllocation.enabled to true

2) set up an **external shuffle service on each worker node** in the same cluster and set **spark.shuffle.service.enabled this for Graceful Decommission of Executors.**

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Spark executor exits either on failure or when the associated application has also exited. In both scenarios, all state associated with the executor is no longer needed and can be safely discarded. With dynamic allocation, however, the application is still running when an executor is explicitly removed.

This requirement is especially important for shuffles. During a shuffle, the Spark executor first writes its own map outputs locally to disk, and then acts as the server for those files when other executors attempt to fetch them. In the **event of stragglers**, which are tasks that run for much longer than their peers, dynamic allocation may remove an executor before the shuffle completes, in which case the shuffle files written by that executor must be recomputed unnecessarily.

Solution is Enabling External Shuffle Service. When enabled, the service is created on a worker node and every time when it exists there, newly created executor registers to it. During the registration process, the executor informs the service about the place on disk where are stored the files it creates. Thanks to this information the external shuffle service daemon is able to return these files to other executors during retrieval process.

External shuffle service presence also impacts files removal. In normal circumstances (no external shuffle service), when an executor is stopped, it automatically removes generated files. But when the service is enabled, the files aren't cleaned after the executor's shut down. So if you are application is not leading to shuffle stage don’t enable this even in case of dynamic memory allocation.

One big advantage of this service is reliability improvement. Even if one of executors goes down, its shuffled files aren't lost. Another advantage is the scalability because external shuffle service is required to run dynamic resource allocation in Spark. This service is really important because if executor is idea then that will be removed so then all resources(disk, RAM) will be taken back so if that executor is executing some shuffling tasks then all the data will be lost.

this service is located on every worker, back to executor(s) belonging to different applications. In fact, external shuffle service can be summarized to a proxy that fetches and provides block files. It doesn't duplicate them. Instead it only knows where they're stored by each of node's executors.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Name	Value	Default Value	Description
spark.dynamicAllocation.enabled	true/false	false	Whether to use dynamic resource allocation
spark.dynamicAllocation.maxExecutors	Based on cluster Config	infinity	Upper bound for the number of executors if dynamic allocation is enabled.
spark.dynamicAllocation.minExecutors	Based on cluster Config	0	Lower bound for the number of executors if dynamic allocation is enabled.
spark.dynamicAllocation.initialExecutors	Based on cluster Config	spark.dynamicAllocation.initialExecutors	Initial number of executors to run if dynamic allocation is enabled. If `--num-executors` (or `spark.executor.instances`) is set and larger than this value, it will be used as the initial number of executors.
spark.dynamicAllocation.schedulerBacklogTimeout	Based on cluster Config	1s	
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout	Based on cluster Config	BacklogTimeout	spark.dynamicAllocation.schedulerBacklogTimeout, but used only for subsequent executor requests

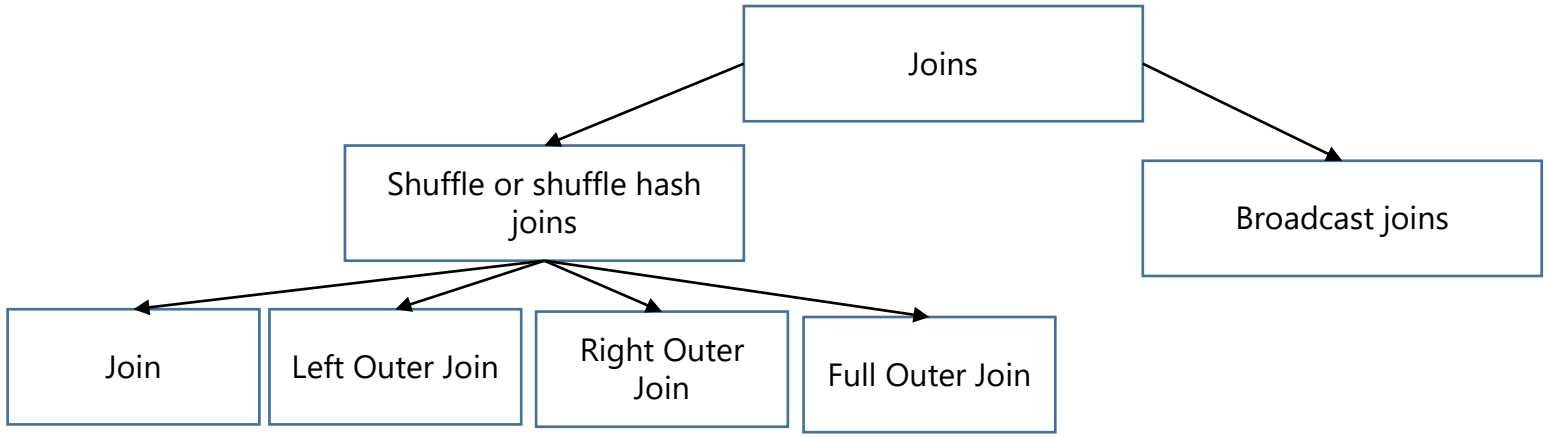
Request Policy	Remove Policy
Spark requests executors in rounds. The actual request is triggered when there have been pending tasks for spark.dynamicAllocation.schedulerBacklogTimeout seconds, and then triggered again every spark.dynamicAllocation.sustainedSchedulerBacklogTimeout seconds thereafter if the queue of pending tasks persists. Additionally, the number of executors requested in each round increases exponentially from the previous round. For instance, an application will add 1 executor in the first round, and then 2, 4, 8 and so on executors in the subsequent rounds.	The policy for removing executors is much simpler. A Spark application removes an executor when it has been idle for more than **spark.dynamicAllocation.executorIdle**Timeout seconds. Note that, under most circumstances, this condition is mutually exclusive with the request condition, in that an executor should not be idle if there are still pending tasks to be scheduled.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

```
spark-submit  \
--class org.training.spark.apiexamples.discount.AmountWiseDiscount  \
--master yarn  \
--deploy-mode cluster \
--driver-cores 2 \
--driver-memory 2G \
--num-executors 10 \
--executor-cores 5 \
--executor-memory 2G \
--conf spark.dynamicAllocation.enabled=True \
--conf spark.dynamicAllocation.minExecutors=5 \
--conf spark.dynamicAllocation.maxExecutors=30 \
--conf spark.dynamicAllocation.initialExecutors=10 \
 spark-core_2.10-0.1.jar file:////home/cloudera/projects/spark-core/src/main/resources/sales.csv
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→Joins in general are expensive since they require that corresponding keys from each RDD are located at the same partition so that they can be combined locally. If the RDDs do not have known partitioners, they will need to be shuffled so that both RDDs share a partitioner, and data with the same keys lives in the same partitions.



In order to join data, Spark needs the data that is to be joined (i.e., the data based on each key) to live on the same partition. The default implementation of a join in Spark is a **shuffled hash join**. The shuffled hash join ensures that data on each partition will contain the same keys by partitioning the second dataset with the same default partitioned as the first, so that the keys with the same hash value from both datasets are in the same partition.

Shuffle Joins
Program reference: org.tranining.spark.apiexamples.ShuffleBased.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→ Cluster manager create a each job for every action in the spark application. Say you have 2 Actions in the application then 2 jobs will be created with there respective stages and task.

→ Like Narrow Transformations joins also leads to shuffle stage which leads to network congestion(Data transfer across different partitions). The distribution of the data will happen based on the partitioned . By default if user didn't provide the partitoner along with join then spark uses Hash partitioner to distribute the RDD data across the partitions.

→ Joins are also Lazy until an action gets triggered on them no job will be created or no memory will be allocated.

→ Here are the Some Optimization Rules you can follow while performing joins

Rule1: When both RDDs have duplicate keys, the join can cause the size of the data to expand dramatically. It may be better to perform a distinct or combineByKey operation to reduce the key space or to use cogroup to handle duplicate keys instead of producing the full cross product. By using smart partitioning during the combine step, it is possible to prevent a second shuffle in the join (we will discuss this in detail later).

Rule2: If keys are not present in both RDDs you risk losing your data unexpectedly. It can be safer to use an outer join, so that you are guaranteed to keep all the data in either the left or the right RDD, then filter the data after the join.

Rule3: If one RDD has some easy-to-define subset of the keys, in the other you may be better off filtering or reducing before the join to avoid a big shuffle of data, which you will ultimately throw away anyway.

Rule4: In order to join data, Spark needs the data that is to be joined (i.e., the data based on each key) to live on the same partition. The default implementation of a join in Spark is a shuffled hash join. The shuffled hash join ensures that data on each partition will contain the same keys by partitioning the second dataset with the same default partitioner as the first, so that the keys with the same hash value from both datasets are in the same partition. While this approach always works, it can be more expensive than necessary because it requires a shuffle. The shuffle can be avoided if:

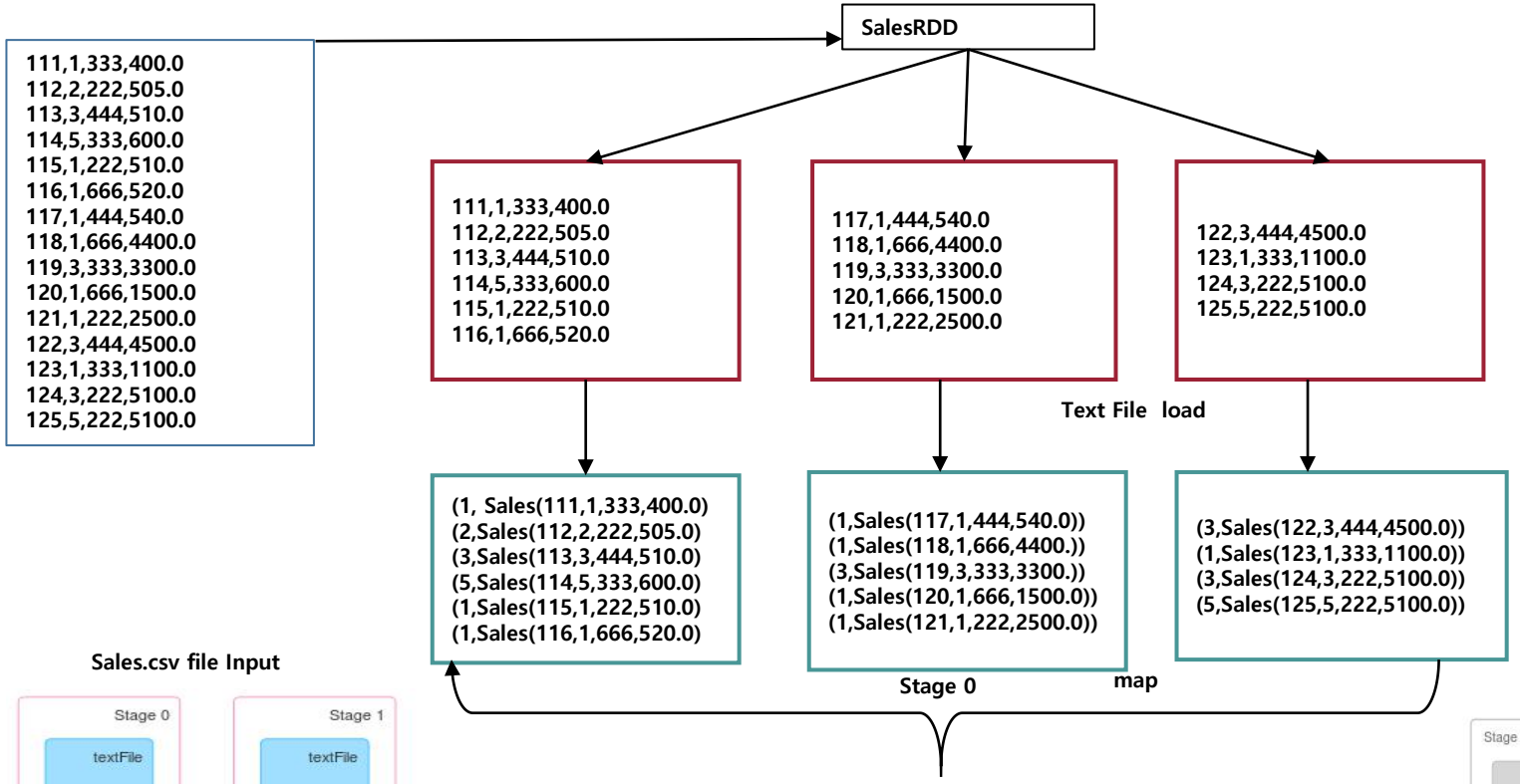
Both RDDs have a known partitioner.

One of the datasets is small enough to fit in memory, in which case we can do a **broadcast** hash join (we will explain what this is later).

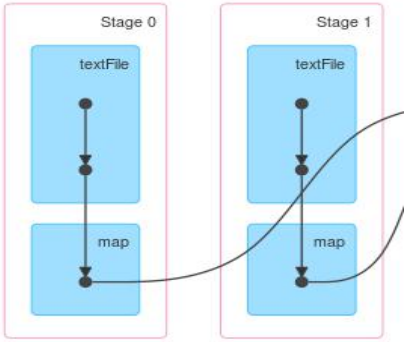
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Shuffle Joins: In detail Understanding of the code
Program reference: org.tranining.spark.apiexamples.ShuffleBased.

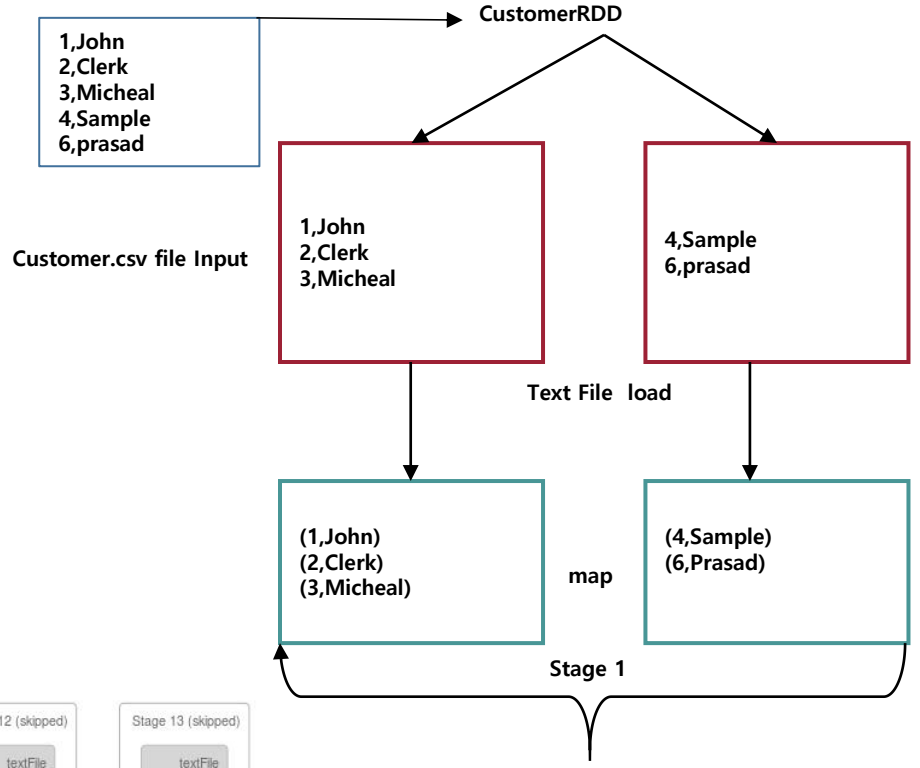
These two stages are common across jobs expect for job4 because we are avoiding shuffling stage by deriving inner join from Left outer join so these will stages will be skipped.

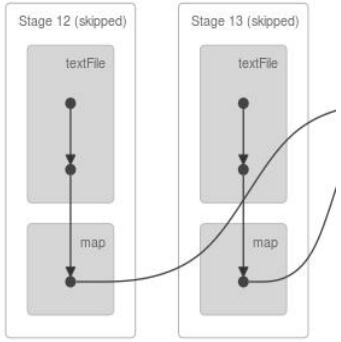


Sales.csv file Input



DAG View for Job 0,1,2,3





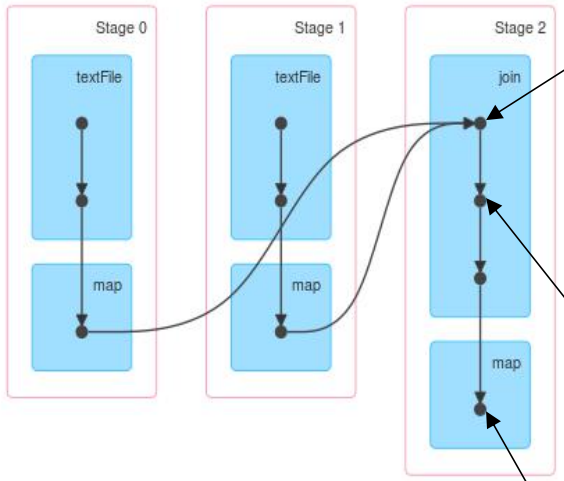
DAG View for Job 4

Shuffle Joins: In detail Understanding of the code
Program reference: org.tranining.spark.apiexamples.ShuffleBased.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Job0: In this Stage Inner Join happens .first shuffling will happen and its done by hash paritioner and it performs "shuffle has join" by hashing key the keys with the same hash value from both datasets are in the same partition. Once the data is partitioned then Inner join(Returns records that have matching values in both tables) join will happen on top it.

DAG View



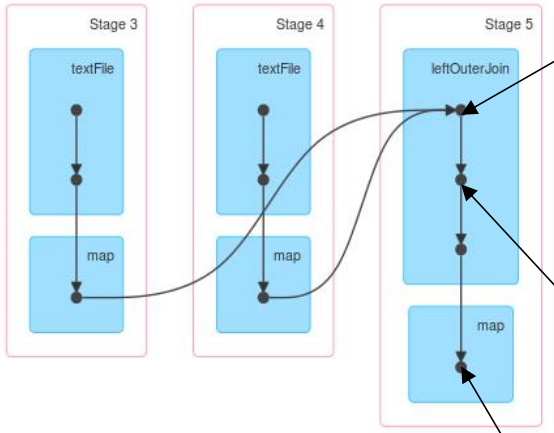
	(1,Sales(111,1,333,400.0)) (1,Sales(115,1,222,510.0)) (1,Sales(116,1,666,520.0)) (1,Sales(117,1,444,540.0)) (1,Sales(118,1,666,4400.0)) (1,Sales(120,1,666,1500.0)) (1,Sales(121,1,222,2500.0)) (1,Sales(123,1,333,1100.0)) (1,John) (6,Prasad)	Hash Shuffle	(2,Sales(112,2,222,505.0)) (2,Clerk) (5,Sales(114,5,333,600.0)) (5,Sales(125,5,222,5100.0))	(3,Sales(113,3,444,510.0)) (3,Sales(119,3,333,3300.0)) (3,Sales(124,3,222,5100.0)) (3,Micheal) (4,Sample)
	(6,"Prasad")→Eliminated (1,("John", Sales(115,1,222,510.0))) (1,("John" ,Sales(116,1,666,520.0))) (1,("John" ,Sales(117,1,444,540.0))) (1,("John" ,Sales(118,1,666,4400.0))) (1,("John" ,Sales(120,1,666,1500.0))) (1,("John" ,Sales(121,1,222,2500.0))) (1,("John" ,Sales(123,1,333,1100.0)))	Join	(2,("Clerk", Sales(112,2,222,505.0))) (5,"114,5,333,600.0") → Eliminated (5,"125,5,222,5100.0") →Eliminated	(3,("Micheal", Sales(113,3,444,510.0))) (3,("Micheal", Sales(119,3,333,3300.0))) (3,("Micheal", Sales(124,3,222,5100.0))) (4,"Sample") → Elminated
	("John", Sales(116,1,666,520.0)) ("John", Sales(116,1,666,520.0)) ("John", Sales(117,1,444,540.0)) ("John", Sales(118,1,666,4400.0)) ("John", Sales(120,1,666,1500.0)) ("John", Sales(123,1,333,1100.0))	map	("Clerk", Sales(112,2,222,505.0))	("Micheal", Sales(113,3,444,510.0)) ("Micheal", Sales(119,3,333,3300.0)) ("Micheal", Sales(119,3,333,3300.0))

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Shuffle Joins: In detail Understanding of the code
Program reference: org.tranining.spark.apiexamples.ShuffleBased.

Job1: In this Stages Left Outer Join happened as it also a join first shuffling and its done by hash paritioner and it performs "shuffle has join" by hashing key the keys with the same hash value from both datasets are in the same partition. Once the data is partitioned then left outer(returns all records from the left RDD , and the matched records from the right RDD. The result is NULL from the right side, if there is no match) join will be applied on top shuffled data.

DAG View



(1,Sales(111,1,333,400.0))
(1,Sales(115,1,222,510.0))
(1,Sales(116,1,666,520.0))
(1,Sales(117,1,444,540.0))
(1,Sales(118,1,666,4400.0))
(1,Sales(120,1,666,1500.0))
(1,Sales(121,1,222,2500.0))
(1,Sales(123,1,333,1100.0))
(1,John)
(6,Prasad)

Hash Shuffle

(2,Sales(112,2,222,505.0))
(2,Clerk)
(5,Sales(114,5,333,600.0))
(5,Sales(125,5,222,5100.0))

(3,Sales(113,3,444,510.0))
(3,Sales(119,3,333,3300.0))
(3,Sales(124,3,222,5100.0))
(3,Micheal)
(4,Sample)

(6,("Prasad", null))
(1,("John", Sales(115,1,222,510.0)))
(1,("John" ,Sales(116,1,666,520.0)))
(1,("John" ,Sales(117,1,444,540.0)))
(1,("John" ,Sales(118,1,666,4400.0)))
(1,("John" ,Sales(120,1,666,1500.0)))
(1,("John" ,Sales(121,1,222,2500.0)))
(1,("John" ,Sales(123,1,333,1100.0)))

Left Join

(2,("Clerk", Sales(112,2,222,505.0)))
(5,"114,5,333,600.0") → Eliminated
(5,"125,5,222,5100.0") →Eliminated

(3,("Micheal", Sales(113,3,444,510.0)))
(3,("Micheal", Sales(119,3,333,3300.0)))
(3,("Micheal", Sales(124,3,222,5100.0)))
((4,("Sample",null))

("Prasad", "NA")
("John", Sales(116,1,666,520.0))
("John", Sales(116,1,666,520.0))
("John", Sales(117,1,444,540.0))
("John", Sales(118,1,666,4400.0))
("John", Sales(120,1,666,1500.0))
("John", Sales(123,1,333,1100.0))

map

("Clerk", Sales(112,2,222,505.0))

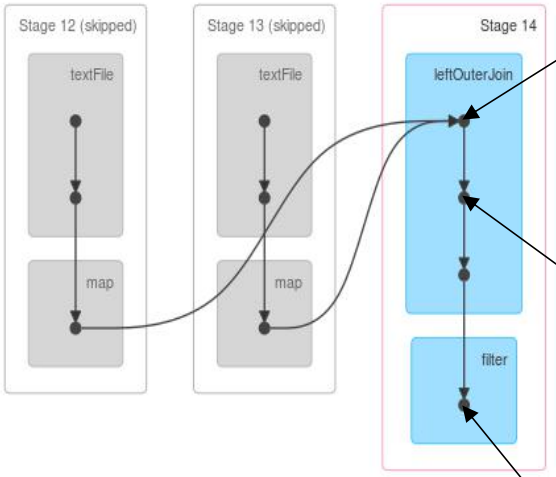
("Micheal", Sales(113,3,444,510.0))
("Micheal", Sales(119,3,333,3300.0))
("Micheal", Sales(119,3,333,3300.0))
("Sample", "NA")

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Shuffle Joins: In detail Understanding of the code
Program reference: org.tranining.spark.apiexamples.ShuffleBased.

Job4:Optimized Inner Join which is getting derived from Left Outer Join. So no shuffle Extra Shuffle required for Inner Join

DAG View



(1,Sales(111,1,333,400.0))
(1,Sales(115,1,222,510.0))
(1,Sales(116,1,666,520.0))
(1,Sales(117,1,444,540.0))
(1,Sales(118,1,666,4400.0))
(1,Sales(120,1,666,1500.0))
(1,Sales(121,1,222,2500.0))
(1,Sales(123,1,333,1100.0))
(1,John)
(6,Prasad)

Hash Shuffle

(2,Sales(112,2,222,505.0))
(2,Clerk)
(5,Sales(114,5,333,600.0))
(5,Sales(125,5,222,5100.0))

(3,Sales(113,3,444,510.0))
(3,Sales(119,3,333,3300.0))
(3,Sales(124,3,222,5100.0))
(3,Micheal)
(4,Sample)

(6,("Prasad", null))
(1,("John", Sales(115,1,222,510.0)))
(1,("John" ,Sales(116,1,666,520.0)))
(1,("John" ,Sales(117,1,444,540.0)))
(1,("John" ,Sales(118,1,666,4400.0)))
(1,("John" ,Sales(120,1,666,1500.0)))
(1,("John" ,Sales(121,1,222,2500.0)))
(1,("John" ,Sales(123,1,333,1100.0)))

Left Join

(2,("Clerk", Sales(112,2,222,505.0)))
(5,"114,5,333,600.0") → Eliminated
(5,"125,5,222,5100.0") →Eliminated

(3,("Micheal", Sales(113,3,444,510.0)))
(3,("Micheal", Sales(119,3,333,3300.0)))
(3,("Micheal", Sales(124,3,222,5100.0)))
((4,("Sample",null))

(6,("Prasad", null))
(1,("John", Sales(115,1,222,510.0)))
(1,("John" ,Sales(116,1,666,520.0)))
(1,("John" ,Sales(117,1,444,540.0)))
(1,("John" ,Sales(118,1,666,4400.0)))
(1,("John" ,Sales(120,1,666,1500.0)))
(1,("John" ,Sales(121,1,222,2500.0)))
(1,("John" ,Sales(123,1,333,1100.0)))

filter (derived Inner Join)

(2,("Clerk", Sales(112,2,222,505.0)))

(3,("Micheal", Sales(113,3,444,510.0)))
(3,("Micheal", Sales(119,3,333,3300.0)))
(3,("Micheal", Sales(124,3,222,5100.0)))

If you want to understand JOINS lets understand how partitioning works will work?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

RDD

When you use **parallelize** method or reading data from a **file** then no practitioner will be used so you look at RDD.partitioner then it gives None as output. In parallelize case data will be evenly distributed among the partitions. In case of reading file lets say reading file that's in HDFS size of the partition depends block size(128MB)
mapreduce.input.fileinputformat.split.minsize or **mapreduce.input.fileinputformat.split.maxsize** .
So splitting input into multiple partitions where data is simply divided into chunks containing consecutive records to enable distributed computation. Exact logic depends on a specific source but it is either number of records or size of a chunk.

PairedRDD

When you are performing **reducebyKey** or **groupbyKey** operation shuffling has to happen that means all the key has to come in one partition so that data will be aggregated. In these cases **Hashparitioner** will be used by Default by spark.

If you have to do an operation before the join that requires a shuffle, such as aggregateByKey or reduceByKey, you can prevent the shuffle by adding a hash partitioner with the same number of partitions as an explicit argument to the first operation before the join.

```
def joinScoresWithAddress3(scoreRDD: RDD[(Long, Double)],
  addressRDD: RDD[(Long, String)]) : RDD[(Long, (Double, String))]= {
   // If addressRDD has a known partitioner we should use that,
   // otherwise it has a default hash parttioner, which we can reconstruct by
   // getting the number of partitions.
   val addressDataPartitioner = addressRDD.partitioner match {
    case (Some(p)) => p
    case (None) => new HashPartitioner(addressRDD.partitions.length)
   }
   val bestScoreData = scoreRDD.reduceByKey(addressDataPartitioner,
    (x, y) => if(x > y) x else y)
   bestScoreData.join(addressRDD)
 }
```

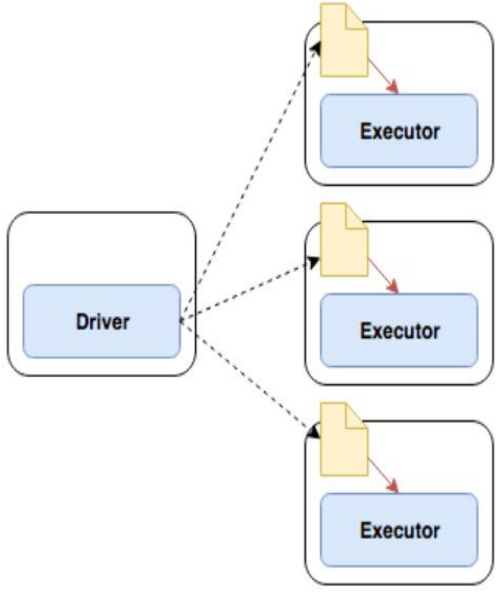
Broadcast Join

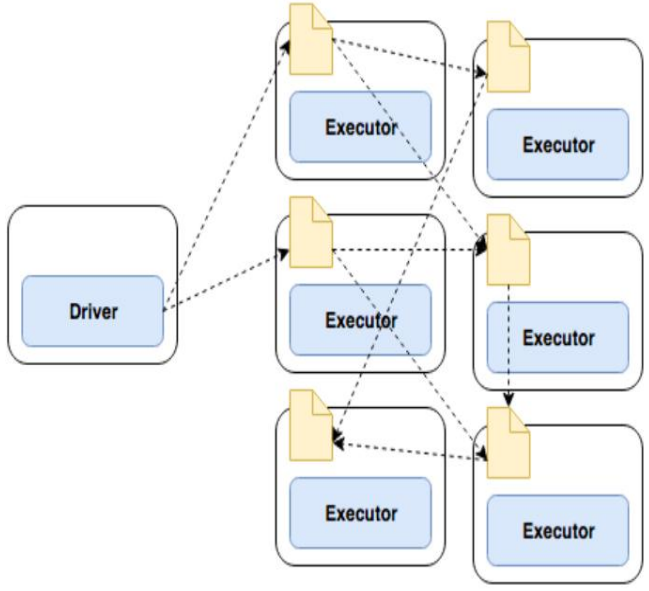
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

To improve performance of join operations in Spark developers can decide to materialize one side of the join equation for a map-only join avoiding an expensive sort an shuffle phase. The table is being send to all mappers as a file and joined during the read operation of the parts of the other table. As the data set is getting materialized and send over the network it does only bring significant performance improvement, if it considerable small. Another constraint is that it also needs to fit completely into memory of each executor. ***Not to forget it also needs to fit into the memory of the Driver!***

In Spark broadcast variables are shared among executors using the Torrent protocol. The **Torrent protocol** is a Peer-to-Peer protocol which is know to perform very well for distributing data sets across multiple peers. The advantage of the Torrent protocol is that peers share blocks of a file among each other not relying on a central entity holding all the blocks.

Broadcast variables are read-only variables which are shared among the executors by caching it on each machine.





Torrent protocol

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

In spark application developers can also use there own custom partitioner by extending the spark application with Partitiner class by overriding the getPartition() method which gives key as an Input and we need to return back Interger which is partition number as an output.

Difference between map, mappartitions, mapPatitionswithIndex

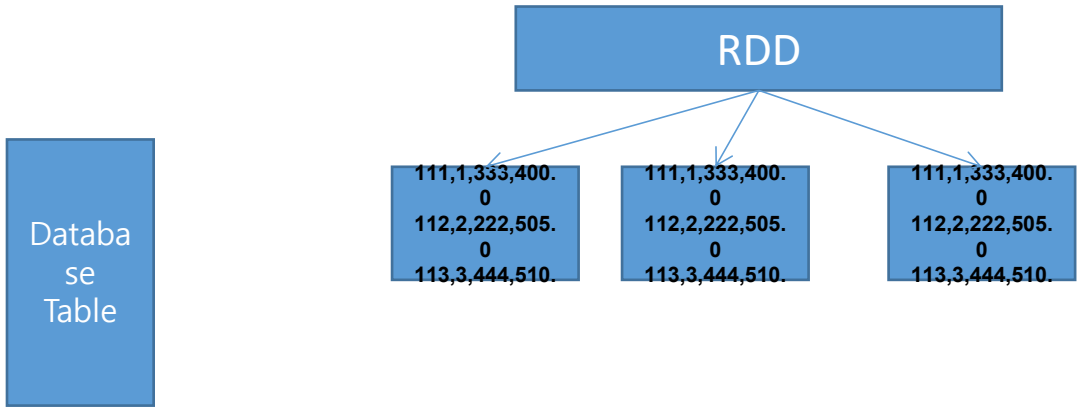
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

mapPartitions() can be used as an alternative to map() & foreach(). mapPartitions() is called once for each Partition unlike map() & foreach() which is called for each element in the RDD. The main advantage being that, we can do initialization on Per-Partition basis instead of per-element basis(as done by map() & foreach())

Consider the case of Initializing a database. If we are using map() or foreach(), the number of times we would need to initialize will be equal to the no of elements in RDD. Whereas if we use mapPartitions(), the no of times we would need to initialize would be equal to number of Partitions

We get Iterator as an argument for mapPartition, through which we can iterate through all the elements in a Partition.

In this example, we will use mapPartitionsWithIndex(), which apart from similar to mapPartitions() also provides an index to track the Partition No



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

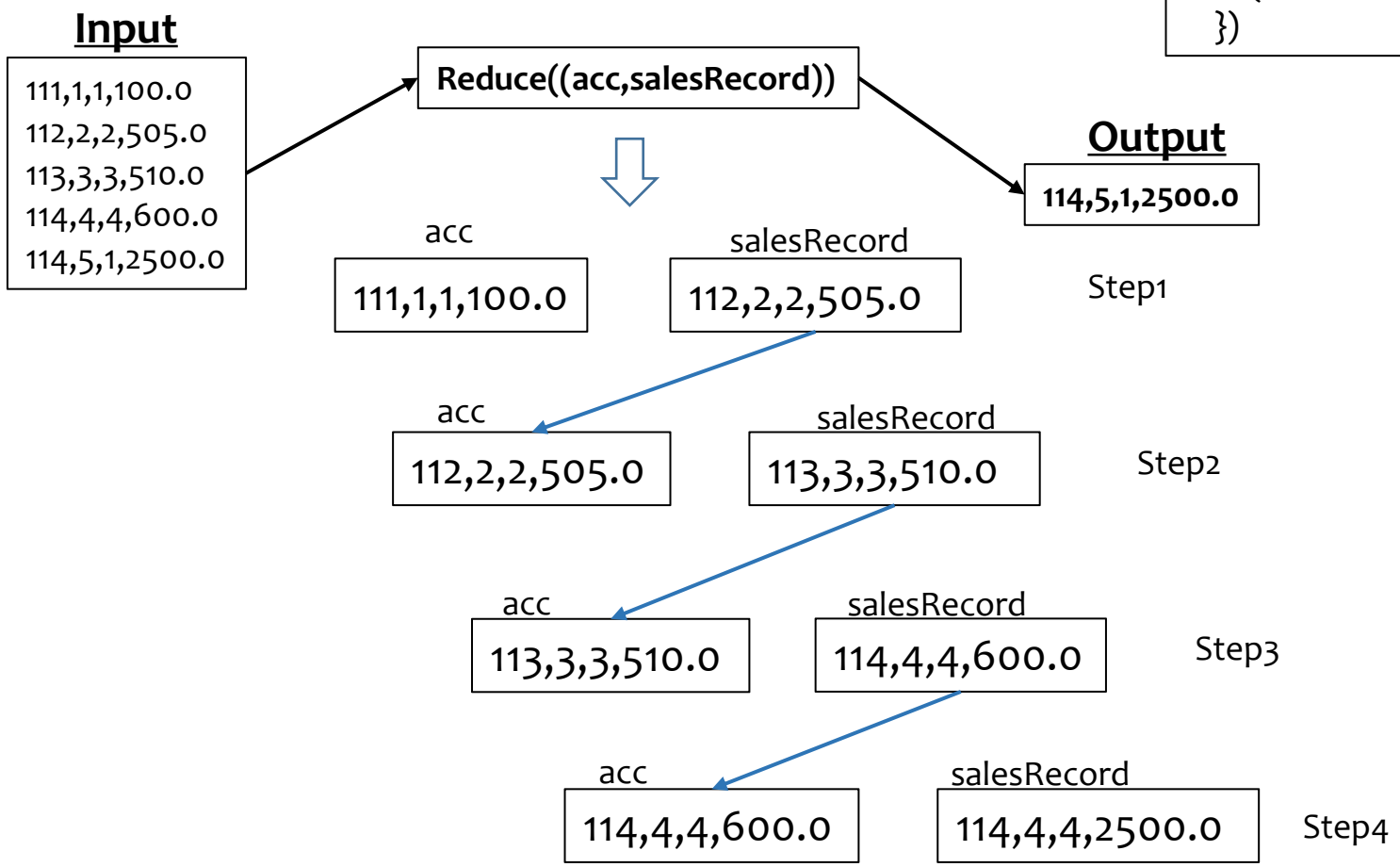
Accumulators are is one of the shared variable and write-only variables shared among executors and created with **SparkContext.accumulator** with default value ,modified with **+=** and and accessed with **value** method. Using accumulators is complicated by Spark's run-at-least-once guarantee for transformations. If a transformation needs to be recomputed for any reason, the accumulator updates during that transformation will be repeated. This means that accumulator values may be very different than they would be if tasks had run only once.

In other words, Accumulators are the write only variables which are initialized once and sent to the workers. These workers will update based on the logic written and sent back to the driver which will aggregate or process based on the logic. Only driver can access the accumulator's value.

Program : errorhandling.counters

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

```
val maxSalesRecord = salesRecordRDD.reduce((acc,salesRecord)=>{
  if(acc.itemValue < salesRecord.itemValue) salesRecord else acc
})
```

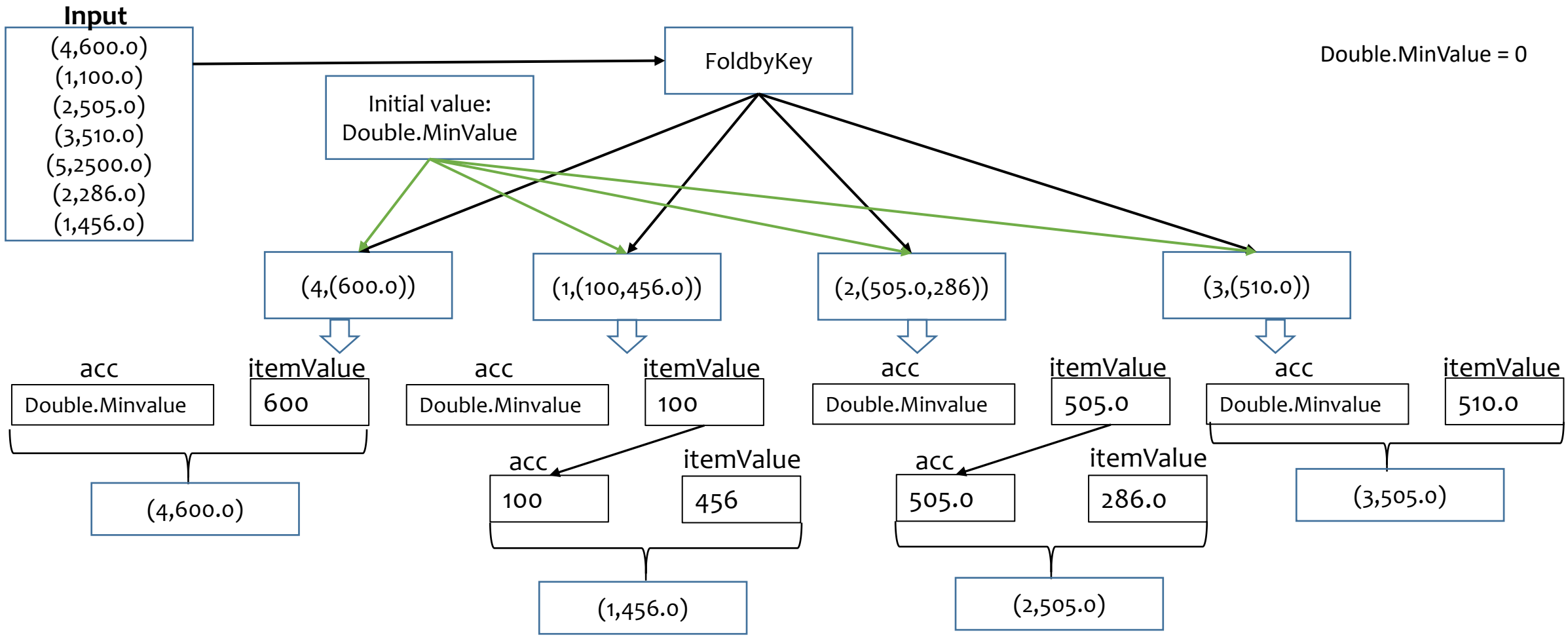


Accumulator will be keep updated with new result and compared with new record.
The difference between reduce and fold and is fold will take initial value where reduce will not

FoldbyKey
Program Reference : apiexamples.advanced.FoldbyKey.scala

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

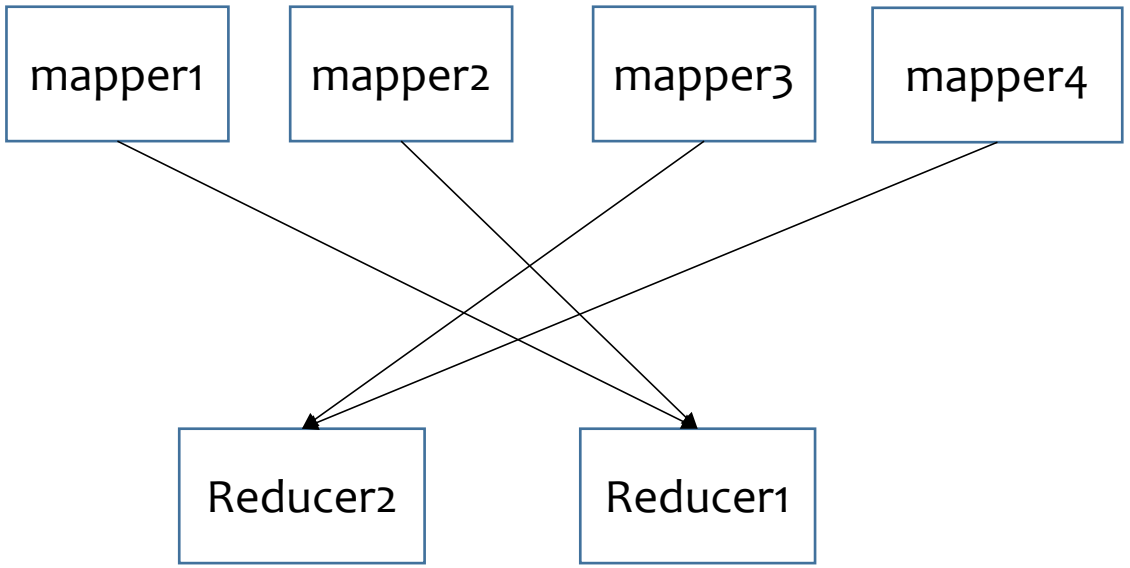
In FoldbyKey the folding will happens at the key level that means values of same key will be folded to a result. Like fold FoldbyKey will also default value



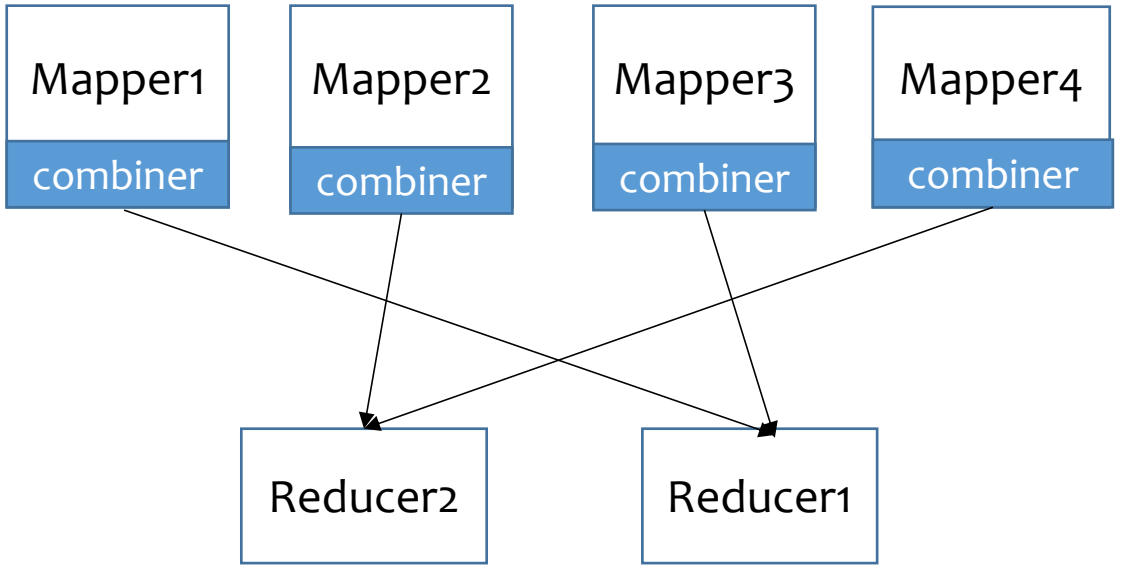
MapPartition as Combiner
Program Reference : apiexamples.advanced.MapPartition.scala

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

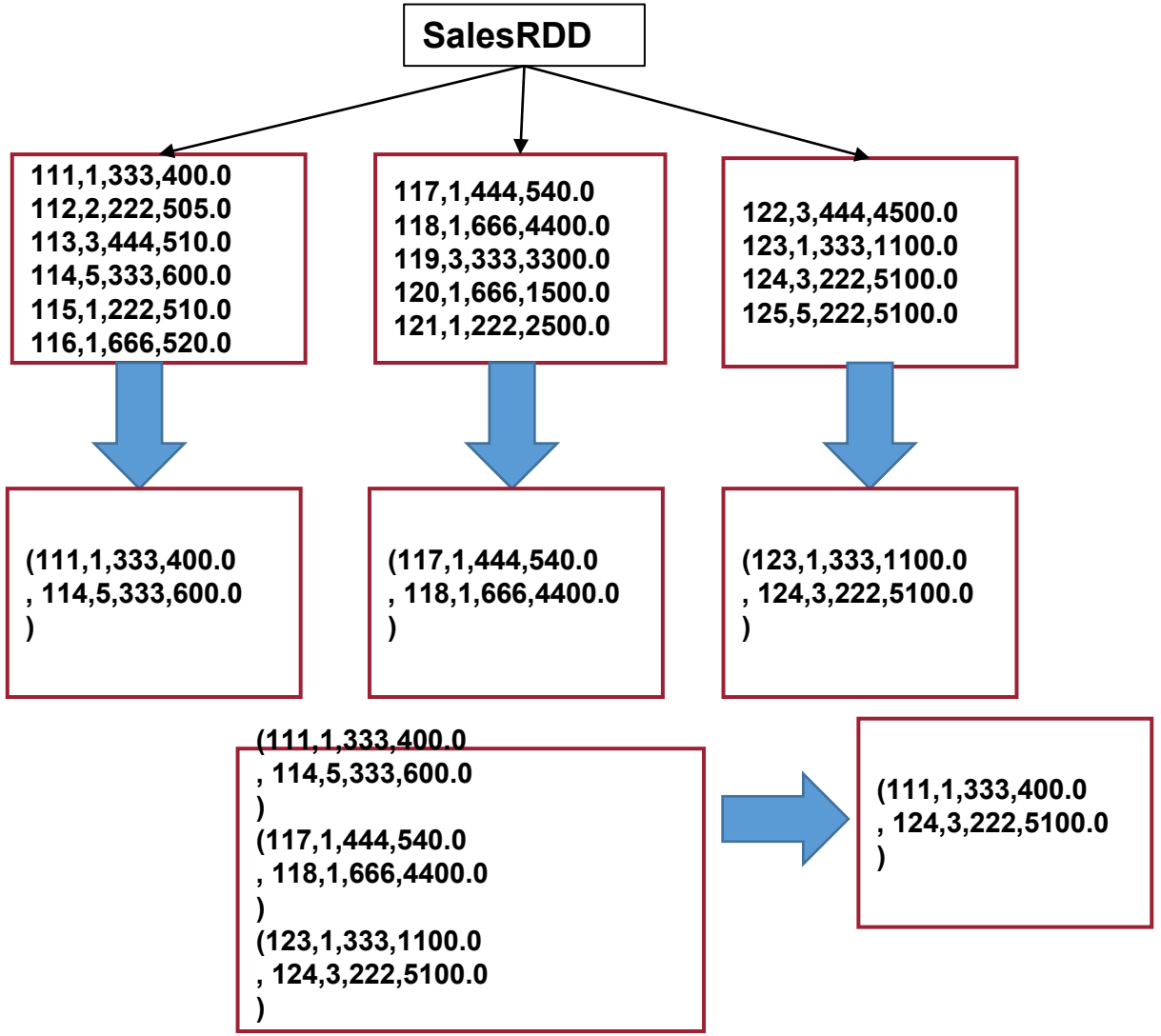
Map partition can also acts like combiner or like mini reducer that means when ever you want to perform some sort of aggregation then spark application will enter reduce phase because without having all the values that belongs to same key in same partition we cannot perform aggregation in this process lots of data will be shuffle across the network which causes network congestion to decrease that what ever the logic your reducer is doing the same thing we put in mapper phase as combiner to achieve this we use mapPartitions



Data Transfer b/w mapper and reducer are high



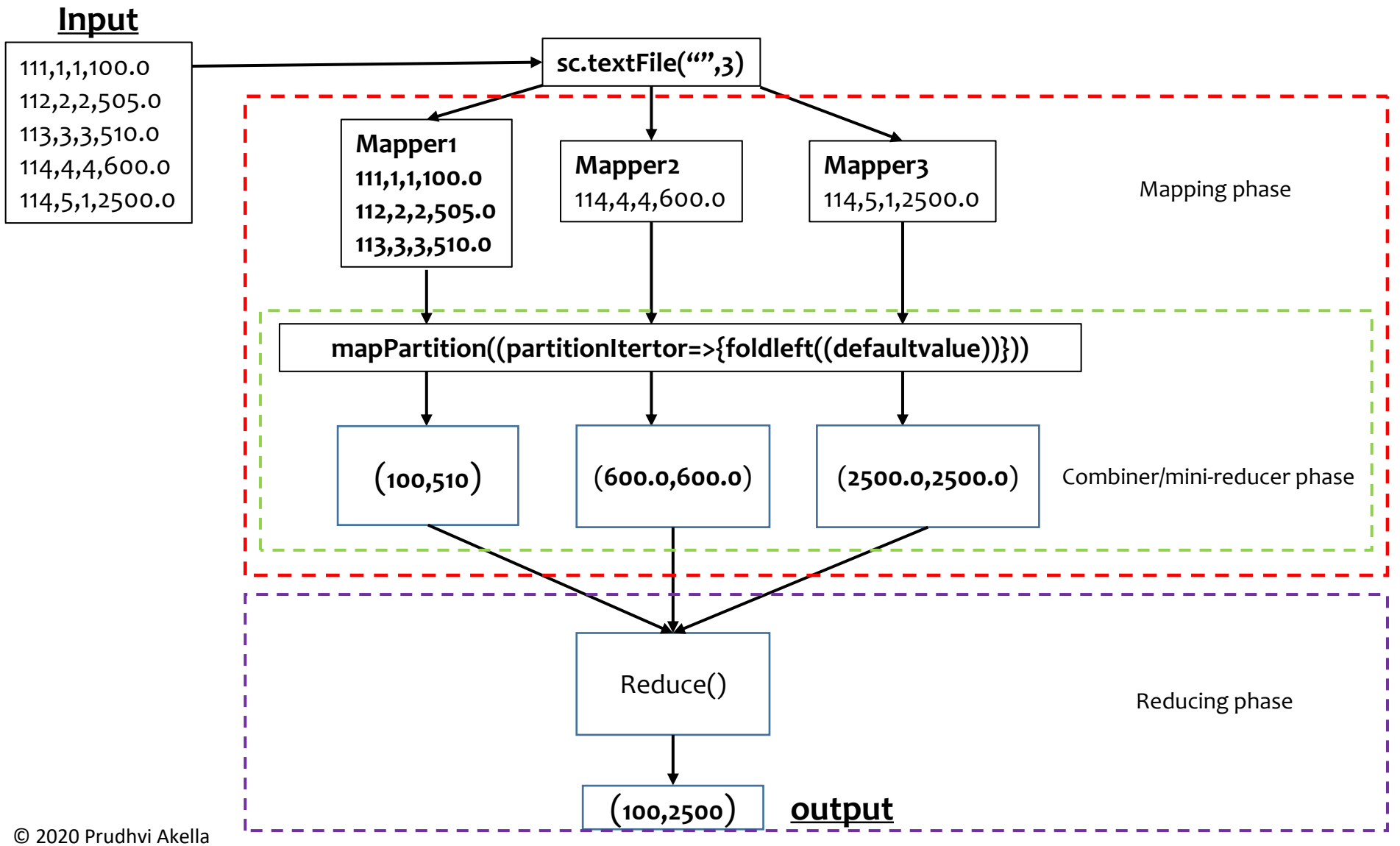
Data Transfer b/w mapper and reducer are low because combiner is acting like mini-reducer and reducing data at mapper side it self


SalesRDD
111,1,333,400.0
112,2,222,505.0
113,3,444,510.0
114,5,333,600.0
115,1,222,510.0
116,1,666,520.0
117,1,444,540.0
118,1,666,4400.0
119,3,333,3300.0
120,1,666,1500.0
121,1,222,2500.0
122,3,444,4500.0
123,1,333,1100.0
124,3,222,5100.0
125,5,222,5100.0
(111,1,333,400.0
, 114,5,333,600.0
)
(117,1,444,540.0
, 118,1,666,4400.0
)
(123,1,333,1100.0
, 124,3,222,5100.0
)
(111,1,333,400.0
, 114,5,333,600.0
)
(117,1,444,540.0
, 118,1,666,4400.0
)
(123,1,333,1100.0
, 124,3,222,5100.0
)
(111,1,333,400.0
, 124,3,222,5100.0
)

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

MapPartition as Combiner

Program Reference : apiexamples.advanced.MapPartition.scala



Points to Remember:
Min function will compare two numeric values or precision value and it returns the min value of it
10.min(100) = 10
100.min(10) = 100

Reverse of min is max
100.max(10) = 100
10.max(100) = 100

Combiner or mini reducer is part of Mapper.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Aggregate also works like “mapPartition as combiner” it has three operations

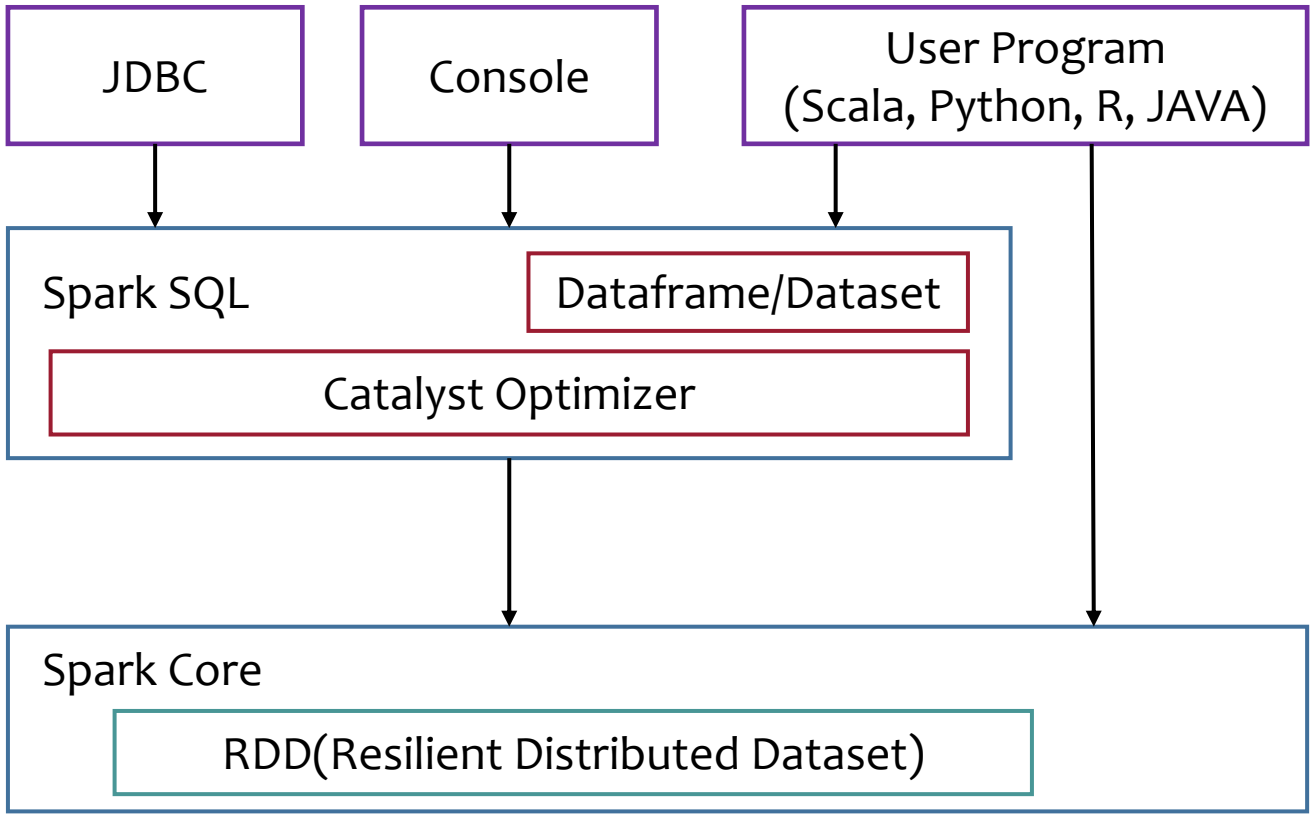
1) Initial Value → Default Value
2) Sequence operation → This acts like mini reducer. Before data goes to combiner at each partition/mapper the data will be aggregated so that the send to the reducer will decrease.
3) Combiner operation → Last stage where final aggregation happens on all the partition outputs.

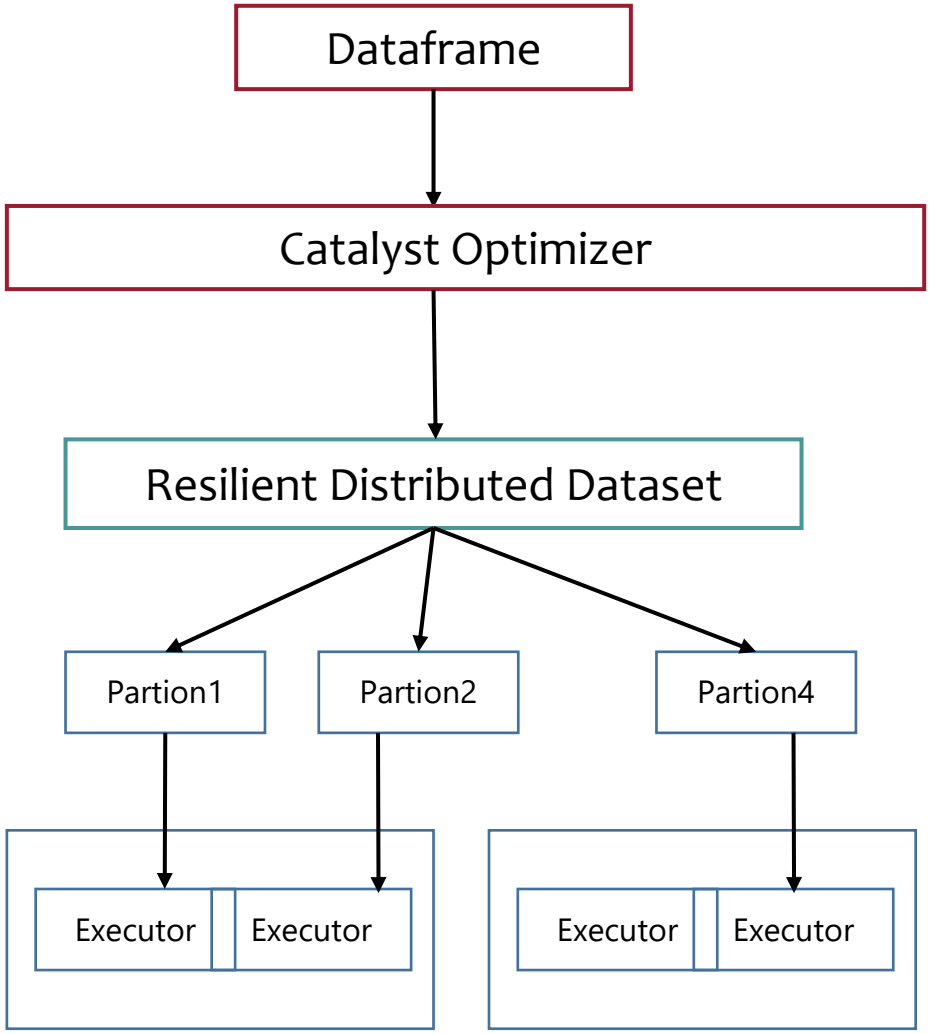
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark SQL

Spark SQL High level Architecture

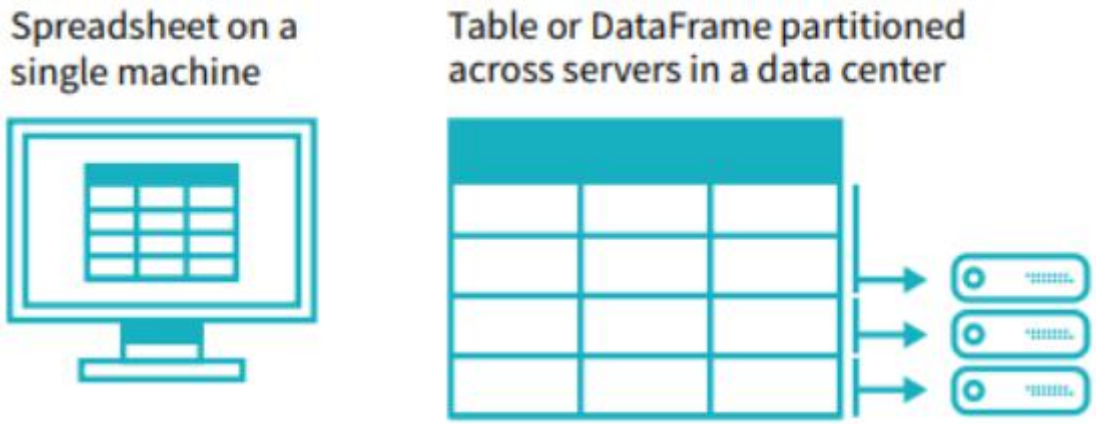
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in





Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

A **DataFrame** is the most common **Structured API** and simply represents a table of data with **rows and columns.** The list of columns and the types in those columns the **schema.** A simple analogy would be a spreadsheet with named columns. The fundamental difference is that while a spreadsheet sits on one computer in one specific location, a Spark DataFrame can span thousands of computers. The reason for putting the data on more than one computer should be intuitive: either the data is too large to fit on one machine or it would simply take too long to perform that computation on one machine.



DataFrames are Faster than RDDS.HOW?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Dataframe

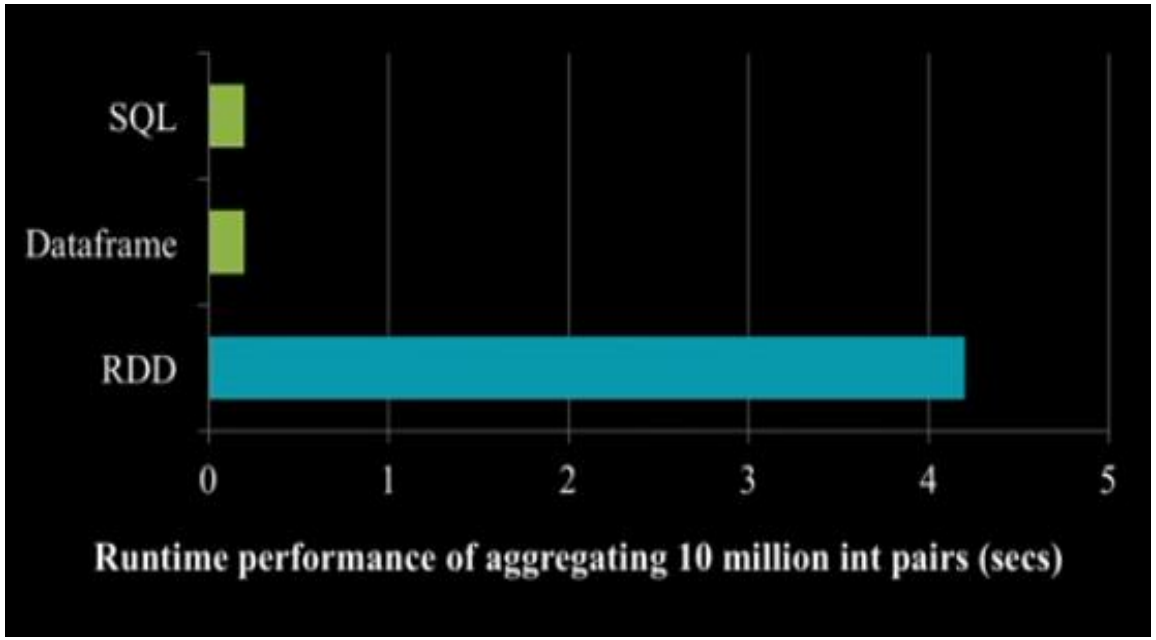
```
data.groupBy("dept").avg("age")
```

SQL

```
select dept, avg(age) from data group by 1
```

RDD

```
data.map { case (dept, age) => dept -> (age, 1) }
    .reduceByKey { case ((a1, c1), (a2, c2)) => (a1 + a2, c1 + c2)}
    .map { case (dept, (age, c)) => dept -> age / c }
```



Window Functions.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Window function calculates a return value for every input row of a table based on a group of rows, called the Frame. Every input row can have a unique frame associated with it. Spark SQL supports three kinds of window functions: ranking functions, analytic functions, and aggregate functions. The available ranking functions and analytic functions are summarized in the table below. For aggregate functions, users can use any existing aggregate function as a window function.

	SQL	DataFrame API
Ranking functions	rank	rank
	dense_rank	denseRank
	percent_rank	percentRank
	ntile	ntile
	row_number	rowNumber
Analytic functions	cume_dist	cumeDist
	first_value	firstValue
	last_value	lastValue
	lag	lag
	lead	lead

There steps involved in defining window function

1) **Partitioning Specification:** controls which rows will be in the same partition with the given row.

2) **Ordering Specification:** controls the way that rows in a partition are ordered, determining the position of the given row in its partition.

3) **Frame Specification:** states which rows will be included in the frame for the current input row, based on their relative position to the current row. For example, "the three rows preceding the current row to the current row" describes a frame including the current input row and three rows appearing before the current row.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

There are five types of boundaries:
UNBOUNDED PRECEDING -> First row of the partition
UNBOUNDED FOLLOWING -> last row of the partition
Below three types of boundaries, they specify the offset from the position of the current input row and their specific meanings are defined based on the type of the frame. There are two types of frames, ROW frame and RANGE frame
CURRENT ROW
<value> PRECEDING
<value> FOLLOWING.

ROW frames:

ROW frames are based on physical offsets from the position of the current input row, which means that CURRENT ROW, <value> PRECEDING, or <value> FOLLOWING specifies a physical offset. If CURRENT ROW is used as a boundary, it represents the current input row. <value> PRECEDING and <value> FOLLOWING describes the number of rows appear before and after the current input row, respectively. The following figure illustrates a ROW frame with a 1 PRECEDING as the start boundary and 1 FOLLOWING as the end boundary (ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING in the SQL syntax).

1 Preceding and current row

current row and 1 following

1 preceding and 1 following

Visual representation of frame
ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING

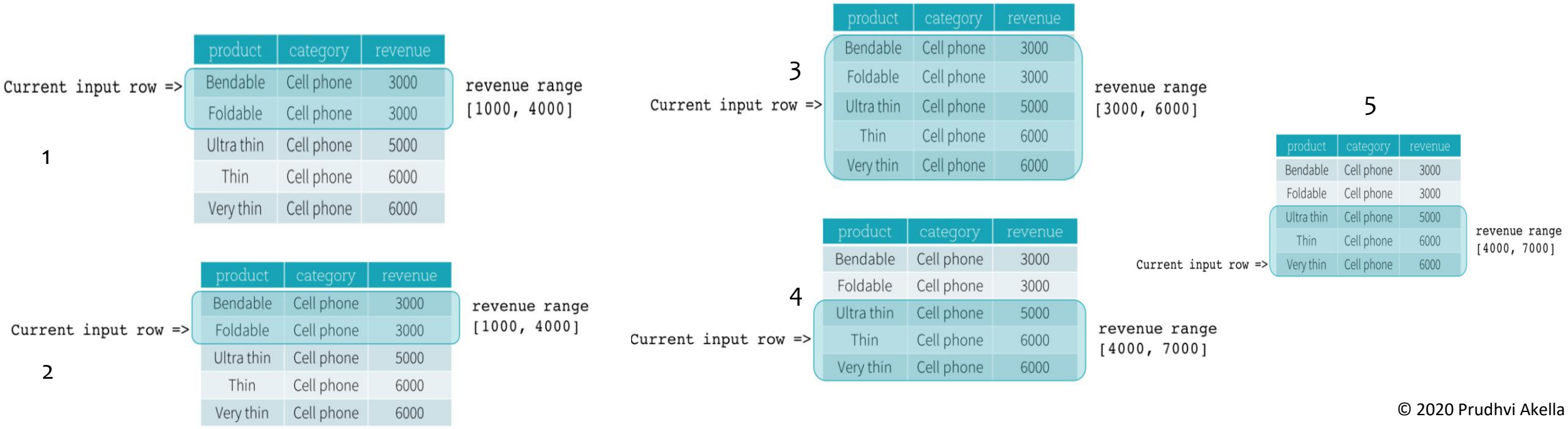
	product	category	revenue	
	Bendable	Cell phone	3000	
	Foldable	Cell phone	3000	<= 1 PRECEDING
Current input row =>	Ultra thin	Cell phone	5000	
	Thin	Cell phone	6000	<= 1 FOLLOWING
	Very thin	Cell phone	6000	

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

RANGE frames are based on logical offsets from the position of the current input row, and have similar syntax to the ROW frame. A logical offset is the difference between the value of the ordering expression of the current input row and the value of that same expression of the boundary row of the frame.

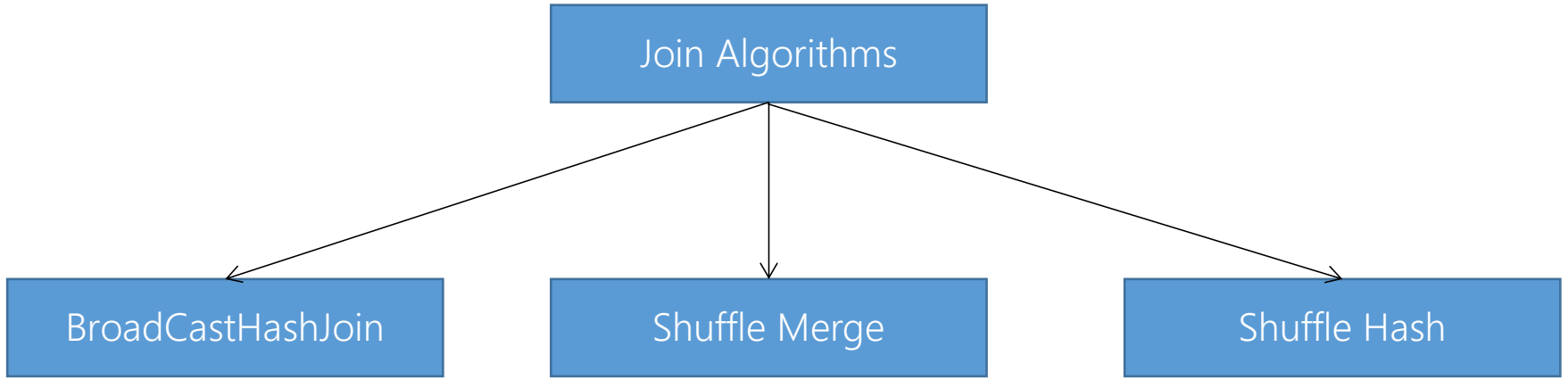
Now, let's take a look at an example. In this example, the ordering expressions is revenue; the start boundary is 2000 PRECEDING; and the end boundary is 1000 FOLLOWING (this frame is defined as RANGE BETWEEN 2000 PRECEDING AND 1000 FOLLOWING in the SQL syntax). The following five figures illustrate how the frame is updated with the update of the current input row. Basically, for every current input row, based on the value of revenue, we calculate the revenue range [current revenue value - 2000, current revenue value + 1000]. All rows whose revenue values fall in this range are in the frame of the current input row.

Visual representation of frame
RANGE BETWEEN 2000 PRECEDING AND 1000 FOLLOWING
(ordering expression: revenue)



product	category	revenue
Bendable	Cell phone	3000
Foldable	Cell phone	3000
Ultra thin	Cell phone	5000
Thin	Cell phone	6000
Very thin	Cell phone	6000

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Spark provides a couple of algorithms for join execution and will choose one of them according to some internal logic. This choice may not be the best in all cases and having a proper understanding of the internal behavior may allow us to lead Spark towards better performance.

Spark 2.X/3.0 provides a flexible way to choose a specific algorithm using strategy hints:

dfA.join(dfB.hint(algorithm), join_condition)

and the value of the algorithm argument can be one of the following:
broadcast,
shuffle_hash,
shuffle_merge

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

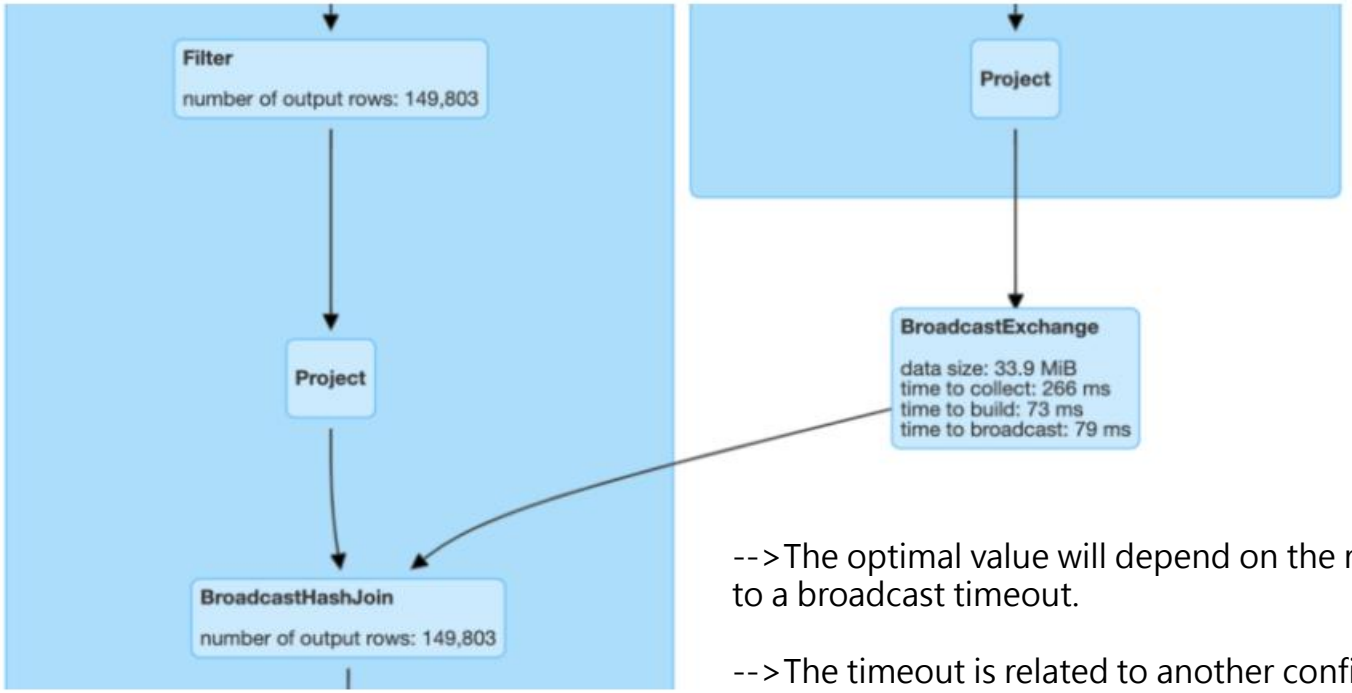
Spark decides what algorithm will be used for joining the data in the phase of physical planning, where each node in the logical plan has to be converted to one or more operators in the physical plan using so-called strategies. The strategy responsible for planning the join is called JoinSelection. Among the most important variables that are used to make the choice belong:

--> the hint
--> the joining condition (whether or not it is equi-join)
--> the join type (inner, left, full outer, ...)
--> the estimated size of the data at the moment of the join

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

BroadcastHashJoin is the preferred algorithm if one side of the join is small enough (in terms of bytes). In that case, the dataset can be broadcasted send over to each executor. This has the advantage that the other side of the join doesn't require any shuffle and it will be beneficial especially if this other side is very large, so not doing the shuffle will bring notable speed-up as compared to other algorithms that would have to do the shuffle.

broadcast-> the smaller dataset is broadcasted across the executors in the cluster where the larger table is located.
hash join-> A standard hash join is performed on each executor.



--> Spark will choose this algorithm if one side of the join is smaller than the **autoBroadcastJoinThreshold**, which is 10MB as default

--> The default size of the threshold is rather conservative and can be increased by changing the internal configuration. For example, to increase it to 100MB, you can just call

spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 100 * 1024 * 1024)
--disable broadcast
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", -1)
--spark.broadcast.compress default to True
--spark.io.compression.codec default lz4
supported lz4,lzf, snappy, ZStandard

-->The optimal value will depend on the resources on your cluster. Broadcasting a big size can lead to OoM error or to a broadcast timeout.

-->The timeout is related to another configuration that defines a time limit by which the data must be broadcasted and if it takes longer, it will fail with an error. The default value of this setting is 5 minutes and it can be changed as follows

spark.conf.set("spark.sql.broadcastTimeout", time_in_sec)

BroadcastHashJoin Might Take time

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Besides the reason that the data might be large, there is also another reason why the broadcast may take too long. Imagine a situation like this

```
dfA = spark.table(...)
dfB = (
  data
  .withColumn("x", udf_call())
  .groupBy("id").sum("x")
)
dfA.join(dfB.hint("broadcast"), "id")
```

-->In this query we join two DataFrames, where the second dfB is a result of some expensive transformations, there is called a user-defined function (UDF) and then the data is aggregated

-->Suppose that we know that the output of the aggregation is very small because the cardinality of the id column is low. That means that after aggregation, it will be reduced a lot so we want to broadcast it in the join to avoid shuffling the data.

--> The problem however is that the UDF (or any other transformation before the actual aggregation) takes to long to compute so the query will fail due to the broadcast timeout.

```
dfA = spark.table(...)
dfB = (
  data
  .withColumn("x", udf_call())
  .groupBy("id").sum("x")
).cache()

dfB.count()

dfA.join(dfB.hint("broadcast"), "id")
```

-->Besides increasing the timeout, another possible solution for going around this problem and still leveraging the efficient join algorithm is to use caching

-->the query will be executed in three jobs.

-->The first job will be triggered by the count action and it will compute the aggregation and store the result in memory (in the caching layer).

--> The second job will be responsible for broadcasting this result to each executor and this time it will not fail on the timeout because the data will be already computed and taken from the memory so it will run fast

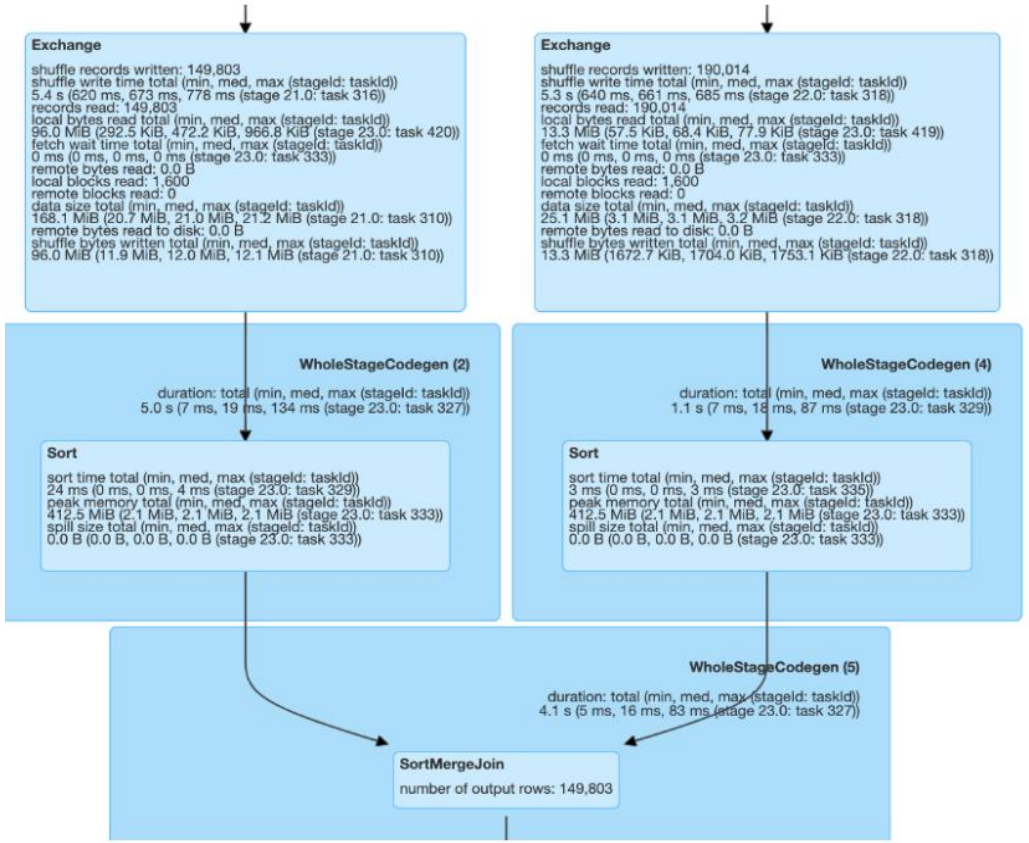
--> Finally, the last job will do the actual join.

SortMergeJoin (SMJ)

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Sort merge join is the default join strategy if the matching join keys are sortable and not eligible for broadcast join or shuffle hash join. It is a very scalable approach and performs better than other joins most of the times. It has its traits from the legendary map-reduce programs. What makes it scalable is that it can spill the data to the disk and doesn't require the entire data to fit inside the memory.

SMJ requires both sides of the join to have correct partitioning and order and in the general case this will be ensured by shuffle and sort in both branches of the join, so the typical physical plan looks like this.



Default spark uses SMJ if not broadcast join

spark.sql.join.preferSortMergeJoin = True

As you can see there is an Exchange and Sort operator in each branch of the plan and they make sure that the

It has 3 phases:
1)Shuffle Phase(Exchange): The 2 large tables are repartitioned as per the join keys across the partitions in the cluster.

2)Sort Phase(Sort): Sort the data within each partition parallelly.

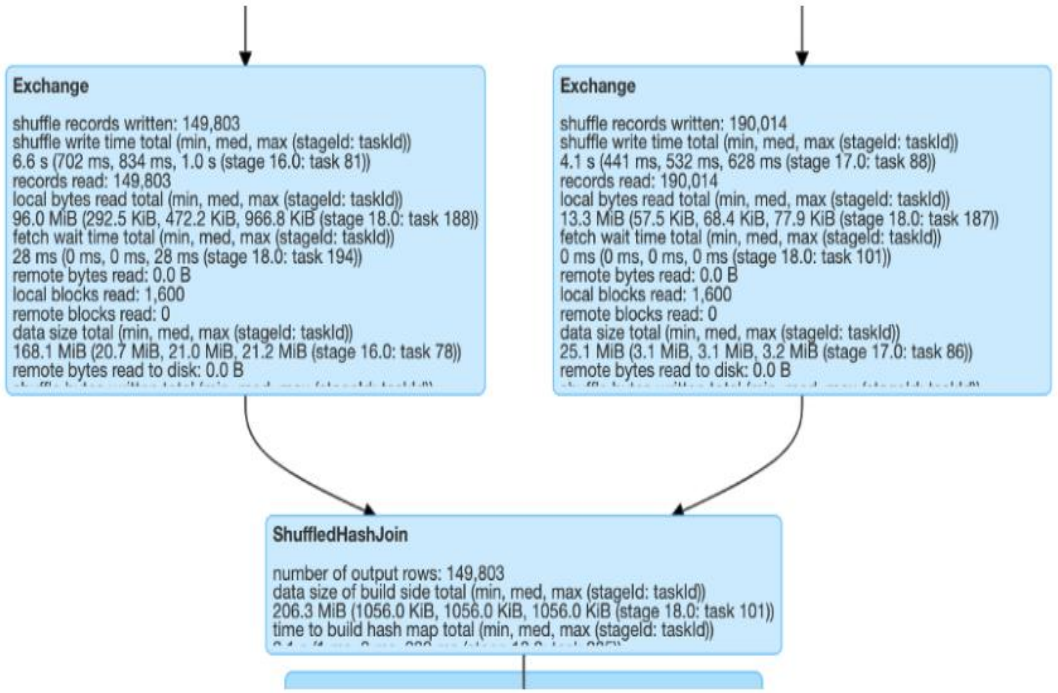
3)Merge Phase(Merge): Join the 2 sorted + partitioned data. This is basically merging of the dataset by iterating over the elements and joining the rows having the same value for the join key.

Advantage:
-->if one partition doesn't fit in memory, Spark will just spill data on disk, which will slow down the execution but it will keep running.

Disadvantage:
-->Costly Sorting phase.

ShuffledHashJoin (SHJ)

If you don't call it by a hint, you will not see it very often in the query plan. The reason behind that is an internal configuration setting **spark.sql.join.preferSortMergeJoin which is set to True** as default. In other words, whenever Spark can choose between SMJ and SHJ it will prefer SMJ. The reason why is SMJ preferred by default is that it is more robust with respect to OoM errors. In the case of SHJ, if one partition doesn't fit in memory, the job will fail, however, in the case of SMJ, Spark will just spill data on disk, which will slow down the execution but it will keep running.



-->Similarly to SMJ, SHJ also requires the data to be partitioned correctly so in general it will introduce a shuffle(exchange) in both branches of the join and creates hashtable and performs the join. However, as opposed to SMJ, it doesn't require the data to be sorted, which is actually also a quite expensive operation and because of that, it has the potential to be faster than SMJ.

-->If you switch the preferSortMergeJoin setting to False,

-->it will choose the SHJ only if one side of the join is at least three times smaller then the other side

--> if the average size of each partition is smaller than the autoBroadcastJoinThreshold (used also for BHJ).

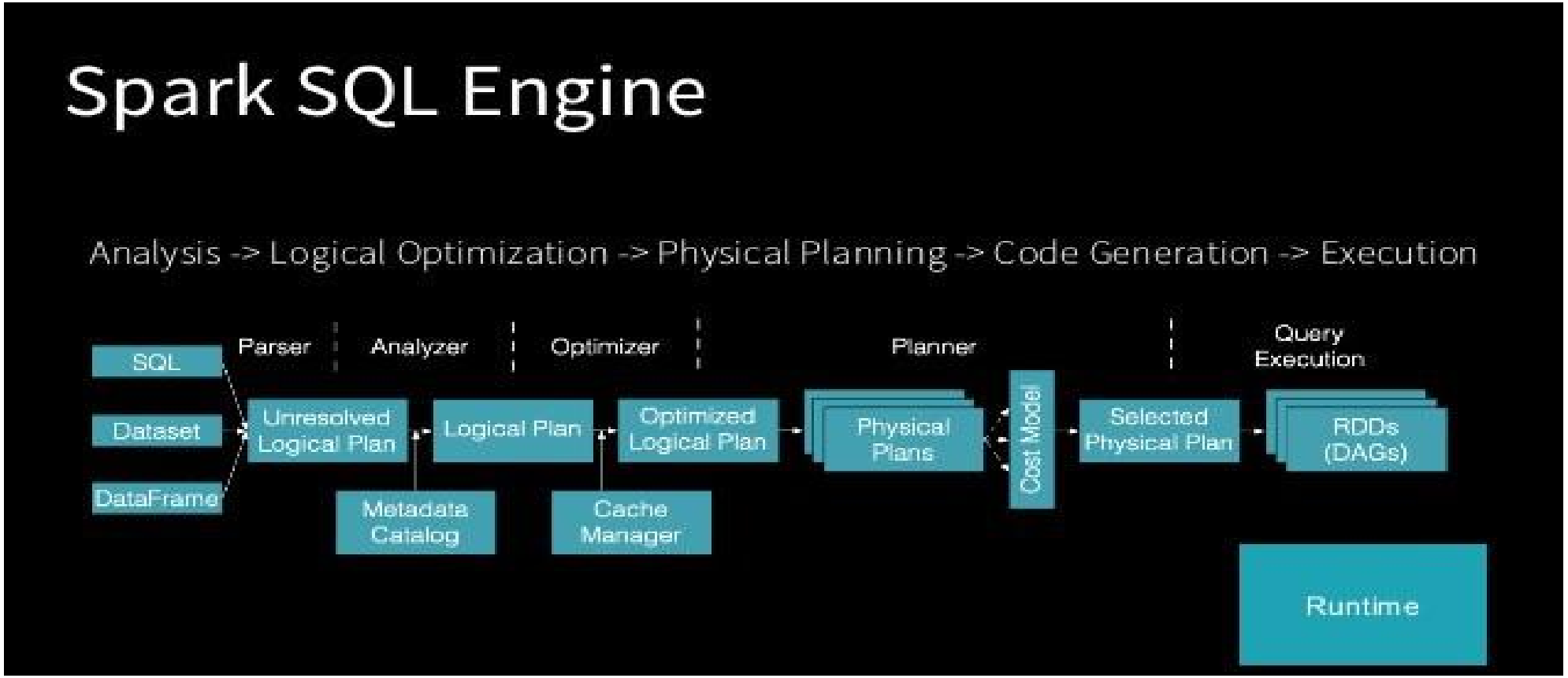
This is to avoid the OoM error, which can however still occur because it checks only the average size, so if the data is highly skewed and one partition is very large, so it doesn't fit in memory, it can still fail.

--> performance of this is based on the distribution of keys in the dataset. The greater number of unique join keys the better data distribution we get. The maximum amount of parallelism that we can achieve is proportional to the number of unique keys.

Example :Say we are joining 2 datasets based on something which would be unique like **empId** would be a good candidate over something like **DepartmentName** which wouldn't have a lot of unique keys and would limit the maximum parallelism that we could achieve.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

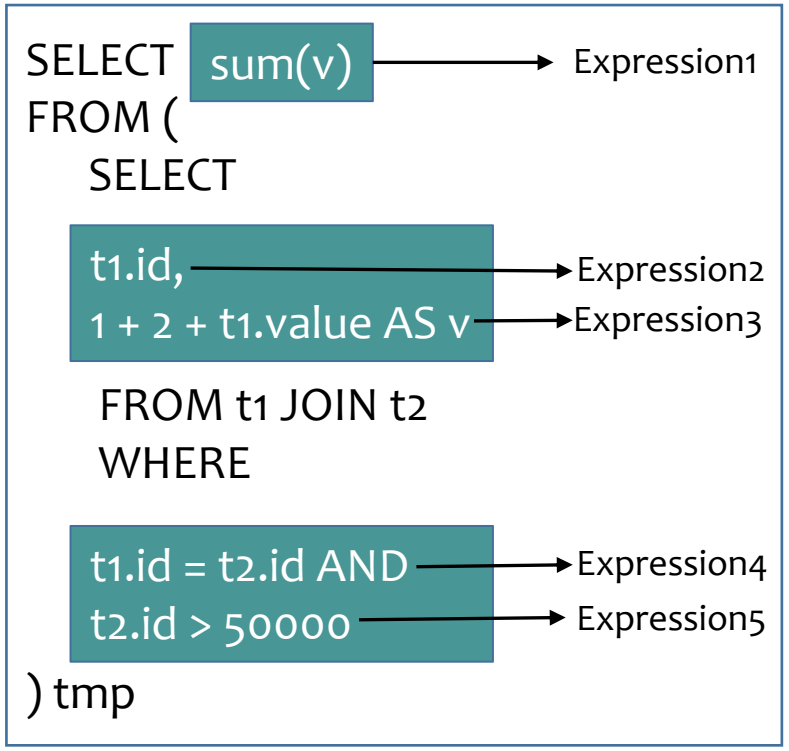
Catalyst Optimizer can automatically finds out the **most efficient execution plan** to execute data operations specified in users Program.



How Parsed Logical plan is converting into Optimized plan?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

This conversion is completely abstracted or not visible to end user or spark developers. however behind the screen parsed logical plan will be converted into **tree data structure**. Lets understand a bit about that with below example which generates a new column by taking input.



IF you look at the simple query we need a way to generate a new column using input column and in spark expression are used for this purpose.

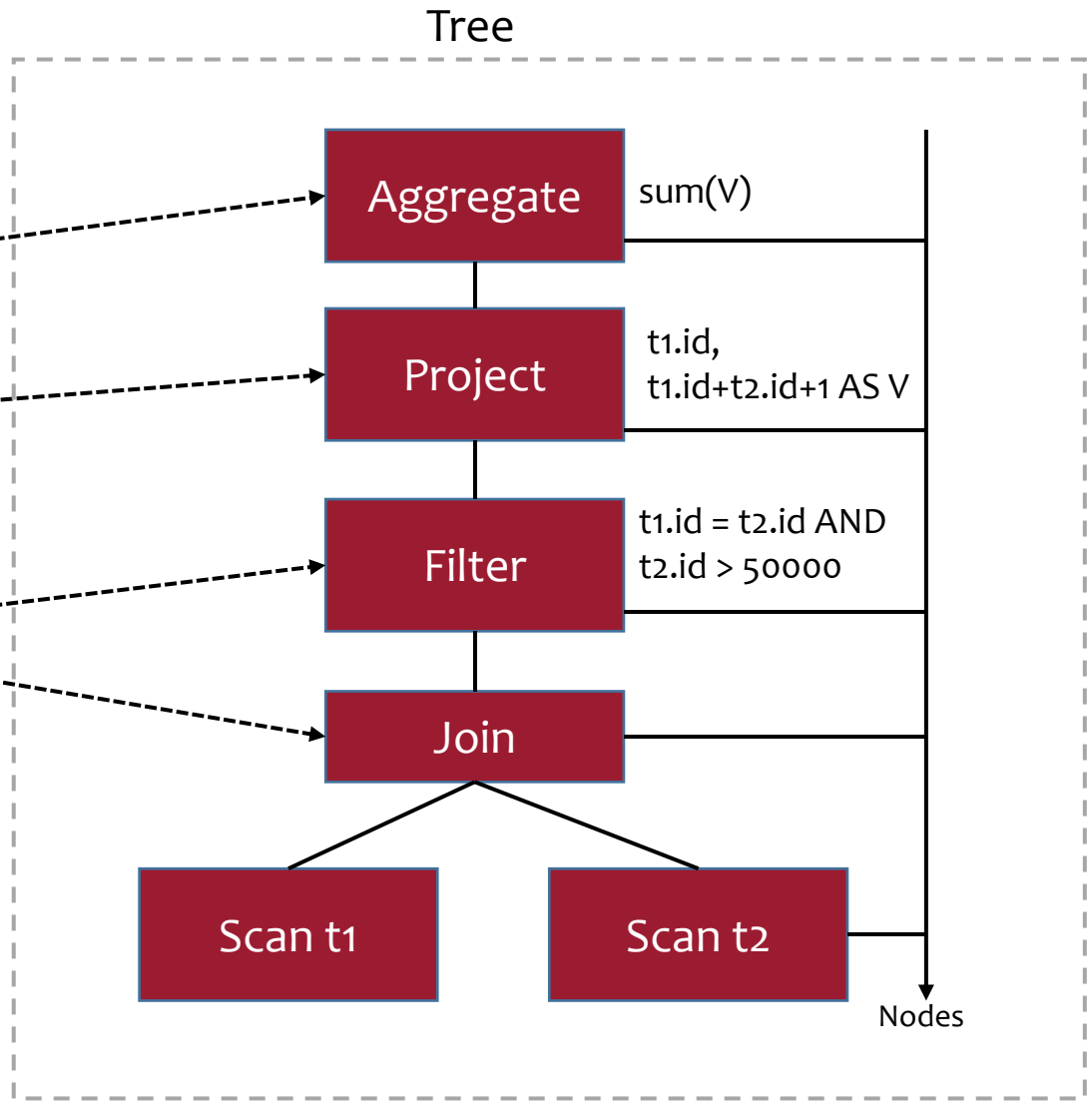
→There are 5 expression in that query and every expression will be converted to a value.
→In Spark columns are also represented by expression we called them as attributes.

An attribute is represented as column of an dataset (Example **t1.id) or column generated by specific data operation(Eg: V)**

So we can use the expressions to represent the operations of generating a new value by taking a new value. In the same way we need a way to generate the new data from input datasets query plan will do that in spark. The next slide talks about that.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

```
SELECT sum(V)
FROM (
    SELECT
     t1.id,
     t1.id+t2.id+1 AS V
    FROM t1 JOIN t2
    WHERE
     t1.id = t2.id AND
     t2.id > 50000
) tmp
```



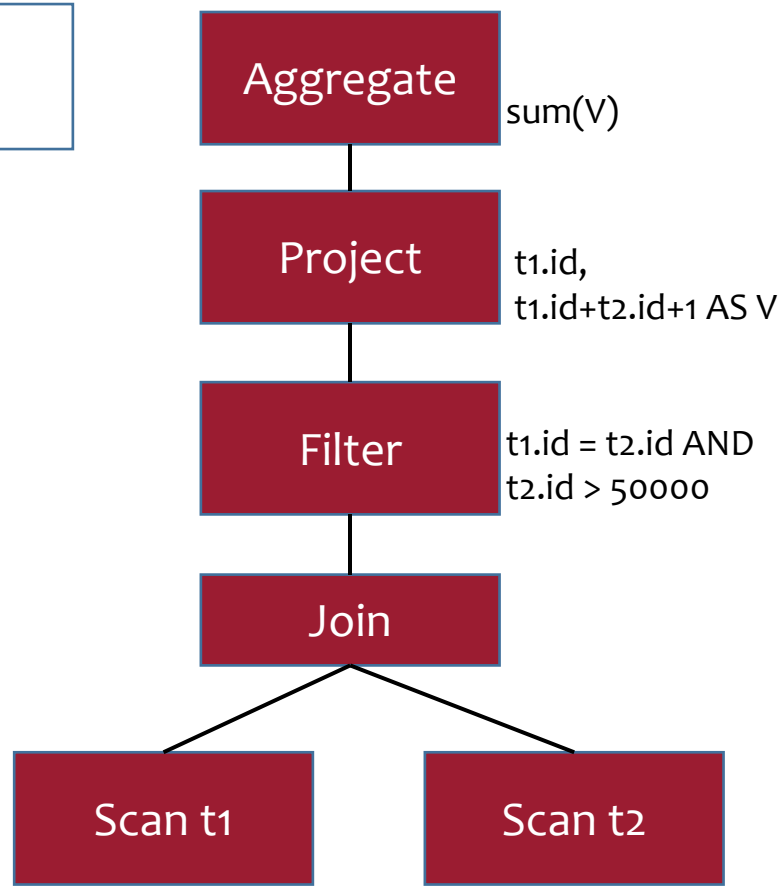
Every query that you write in spark will be converted to Tree and each operation is considered to be a node or a leaf. Always Evaluate the Logical plan from bottom to top that mean first scan will happen then join, filter, Aggregate

Query Plan: Logical Plan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Logical plan describes a computation on datasets without defining how to perform computation

Output	List of Attributes generated by Logical plan Example : id, V
Constraints	A set of invariants(something that should stick to condition no matter whatever changes t2.id>5000 it will remains always same) about the rows generated by the Logical plan Example: t2.id > 50000
Statistics	**Plan statistics**: Size of plan in bytes **Column Statistics**: max, min, nvds(number of distinct value), nnds(number of null values)



Logical plan

Catalyst Optimizer: Transformations

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

There are two types of transformation performed by catalyst optimizer

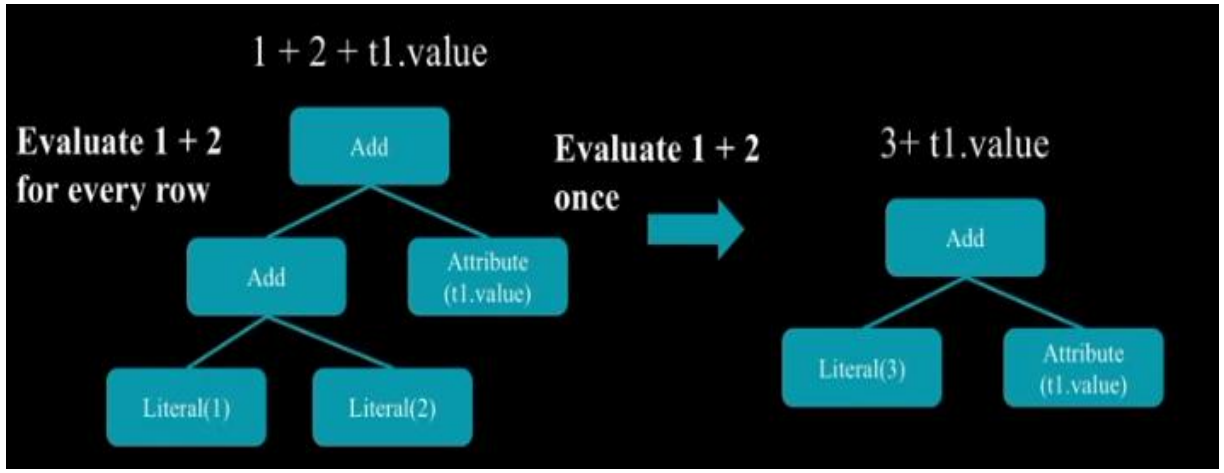
1) Transormation1: Transformations without changing a Tree(Transform and Rule Executer)
   - → Expression => Expression
   - → Logical plan => Logical Plan
   - → Physical plan => Physical Plan

2) Transormation2: Transforming tree to another kind of Tree
   - → Logical plan => Physical Plan

Catalyst Optimizer: Transform Function(Transformation 1)

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

In Catalyst a single transformation is done by a single rule and rule is implemented by function call Transform. This function is associate with every tree you can use this function to convert expressions or you can also use with tree conversion also. Transform is a **Partial Function.**



Expression Evaluation

1 + 2 + t1.value

```
val expression: Expression = ...
expression.transform {
  case Add(Literal(x, IntegerType), Literal(y, IntegerType)) =>
    Literal(x + y)
}
```

Case statement determines if the partial function is defined for a given input

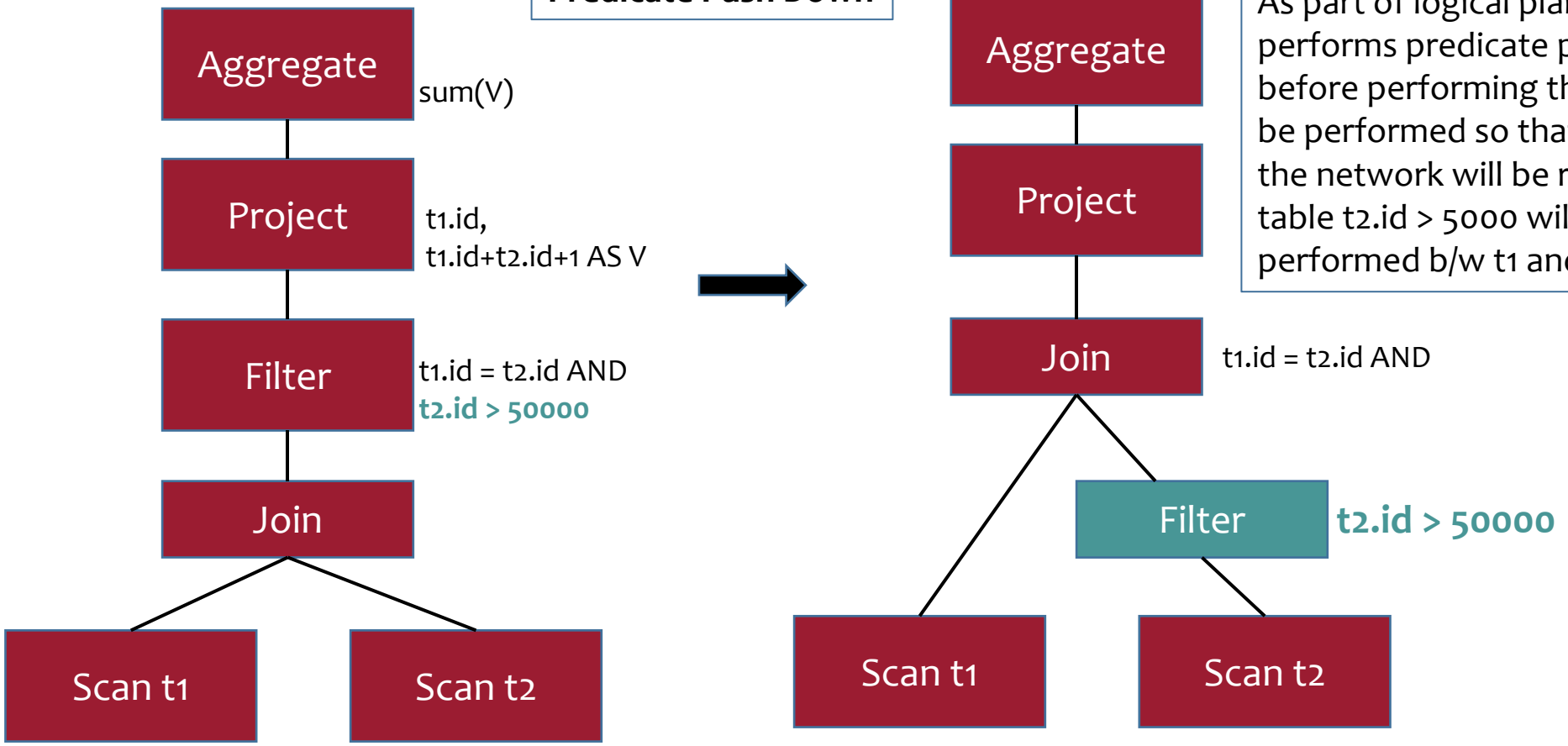
Transform is a Partial Function and it looks like above and it will get only triggered only when you are trying to add two integer values

Catalyst Optimizer: Transform Function(Transformation 1) Combining Multiple Rules

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

As we keep performing transformation on expressions and tree to another tree at one point in time we need to combine different types of transformation rules which cannot be done with single transformation. In catalyst we can combine multiple rules together lets see an example.

Predicate Push Down

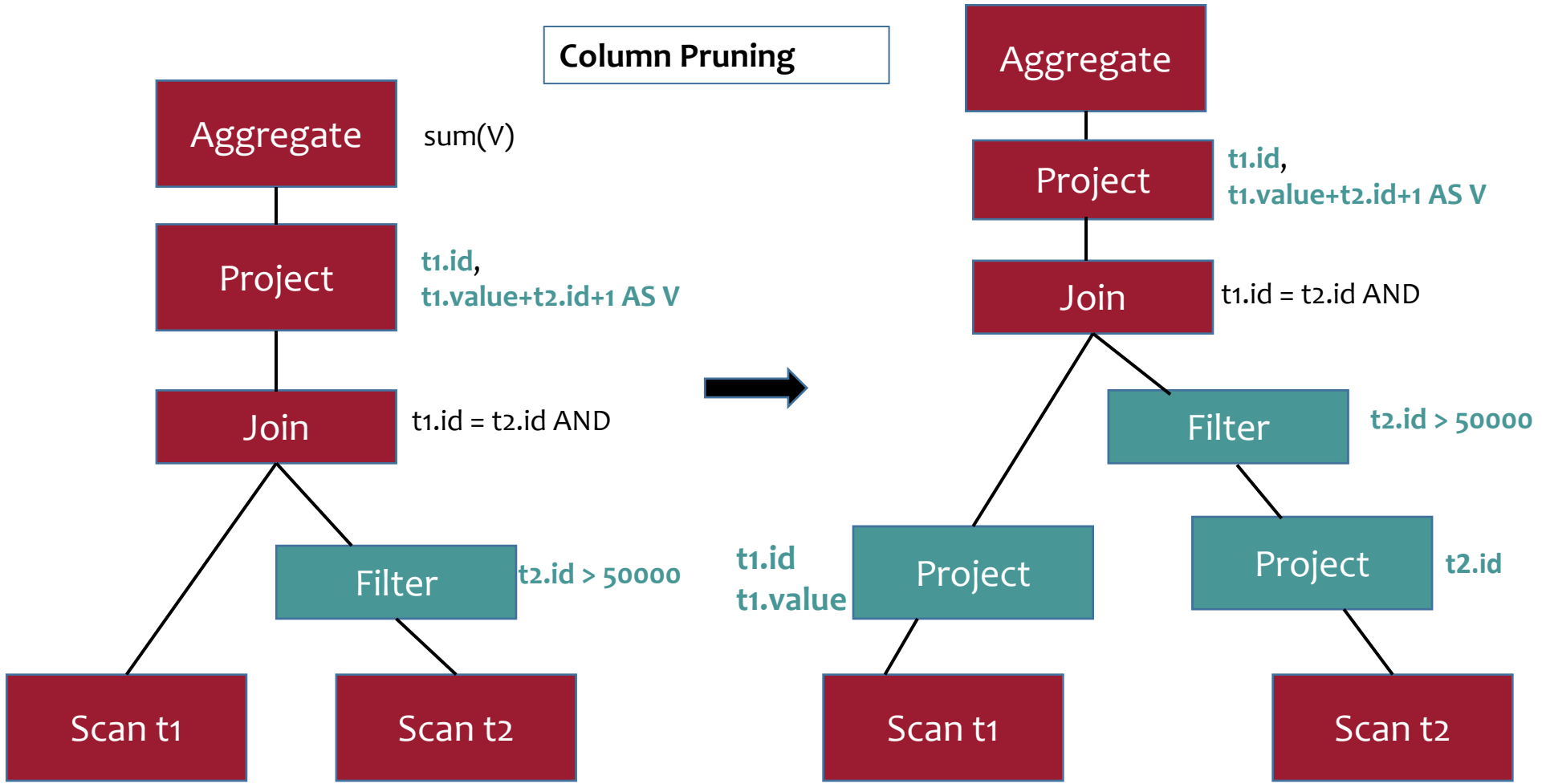


As part of logical plan catalyst optimizer will performs predicate push down which means before performing the join operation filters will be performed so that the data transfer across the network will be reduced. In this case on t2 table t2.id > 5000 will be applied the join will be performed b/w t1 and t2

Catalyst Optimizer: Transform Function(Transformation 1) Combining Multiple Rules

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

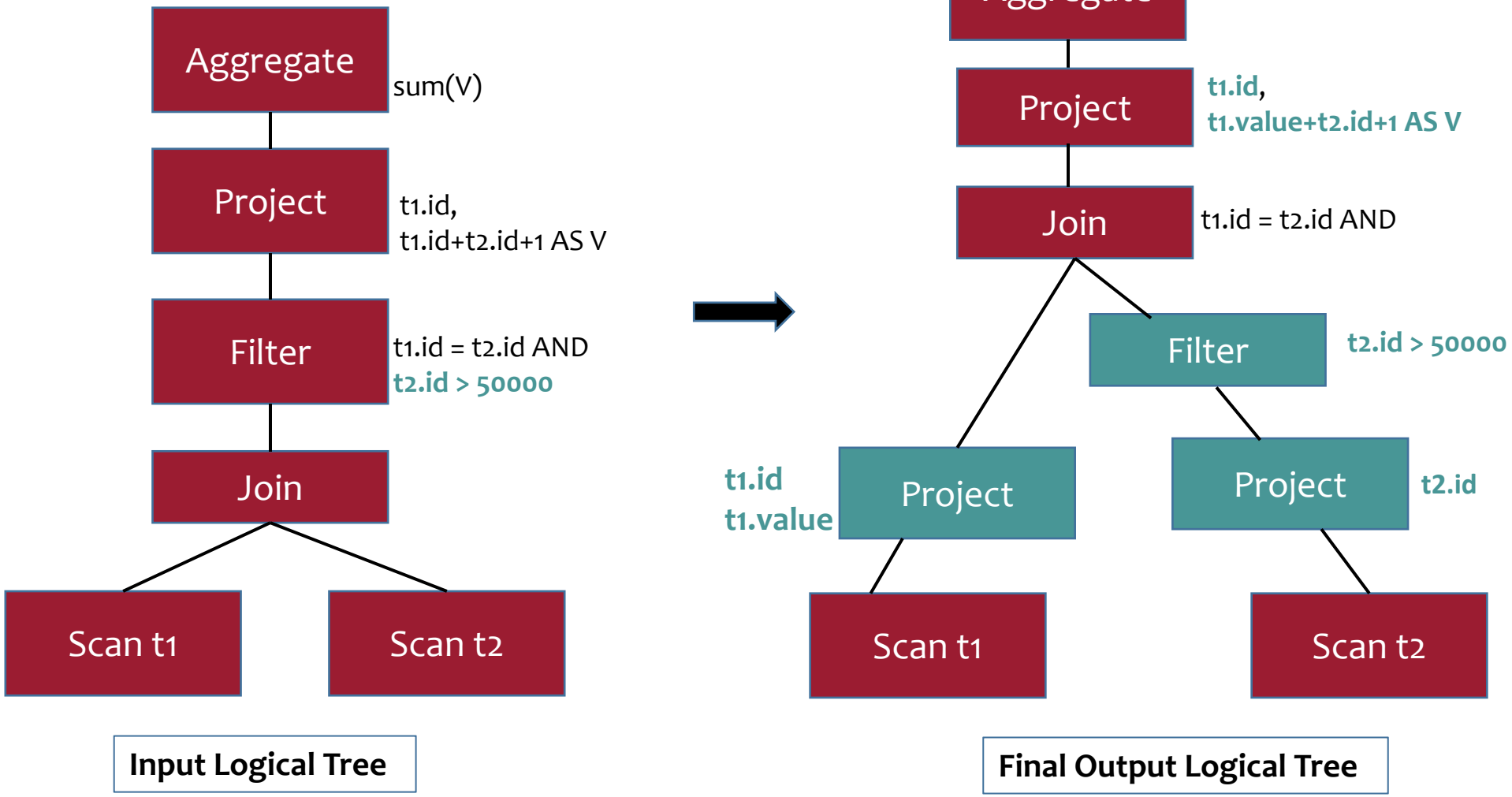
Column Pruning



If you look at the query clearly you need only three columns t1.id ,t1.value and t2.id so for that instead of sending the all the columns from both tables that columns that are required only for aggregation those will be sent its called **Column Pruning**

Catalyst Optimizer: Transform Function(Transformation 1) Combining Multiple Rules

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



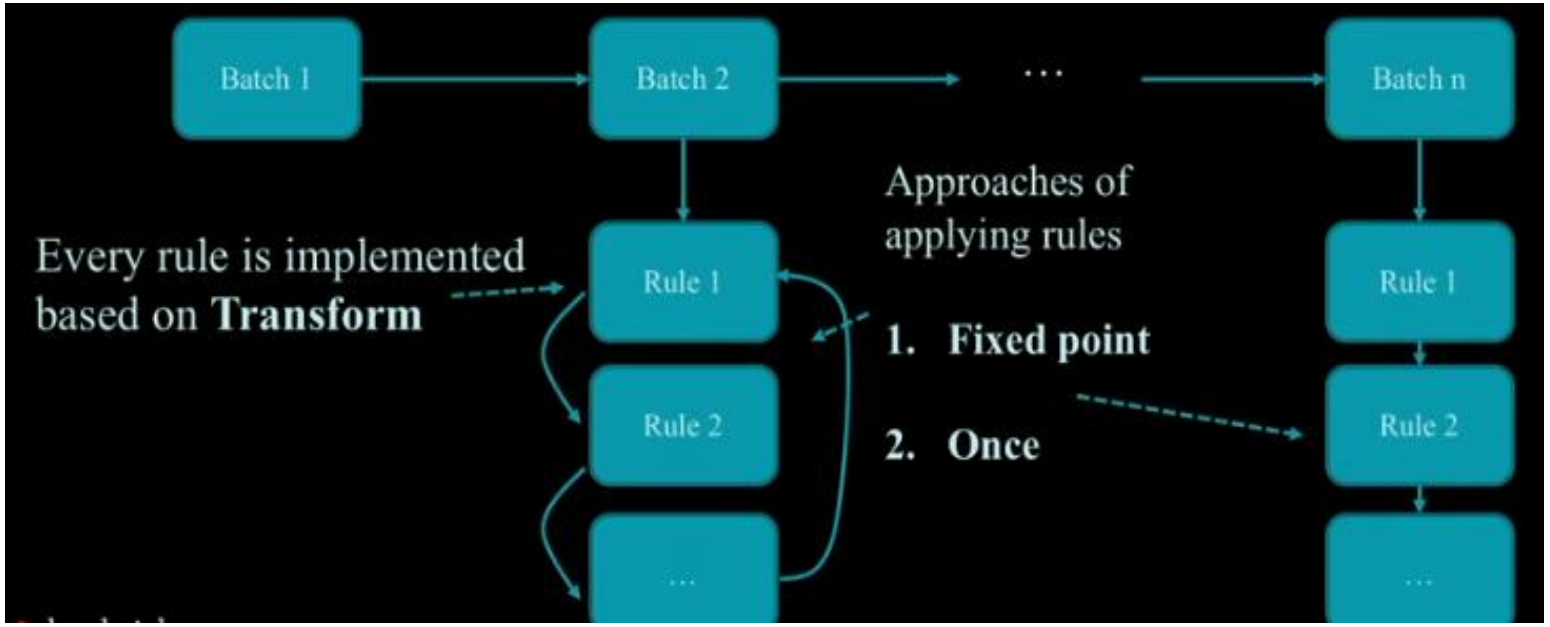
Input Logical Tree

Final Output Logical Tree

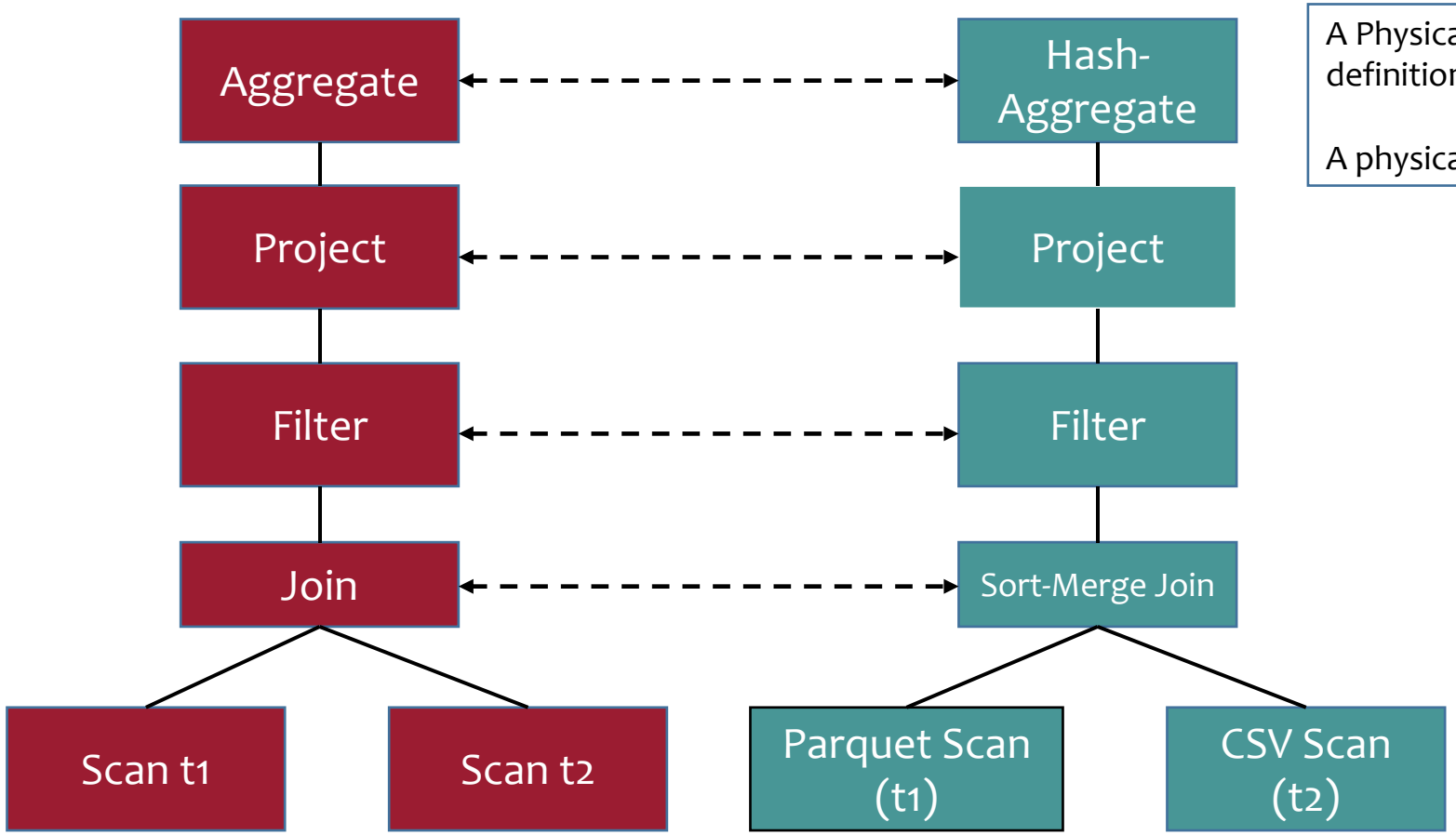
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

A Rule Executor transforms a tree to another same Type of tree by applying many rules defined in batches.
Its divided into two types.

1) Fixed point: In Fixed approach we will apply rules over and over again until that tree doesn't change anymore.
2) Once : It will apply all rules in the same batch and get them all triggered.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



A Physical plan describes computation on dataset with specific definitions on how to conduct the computation

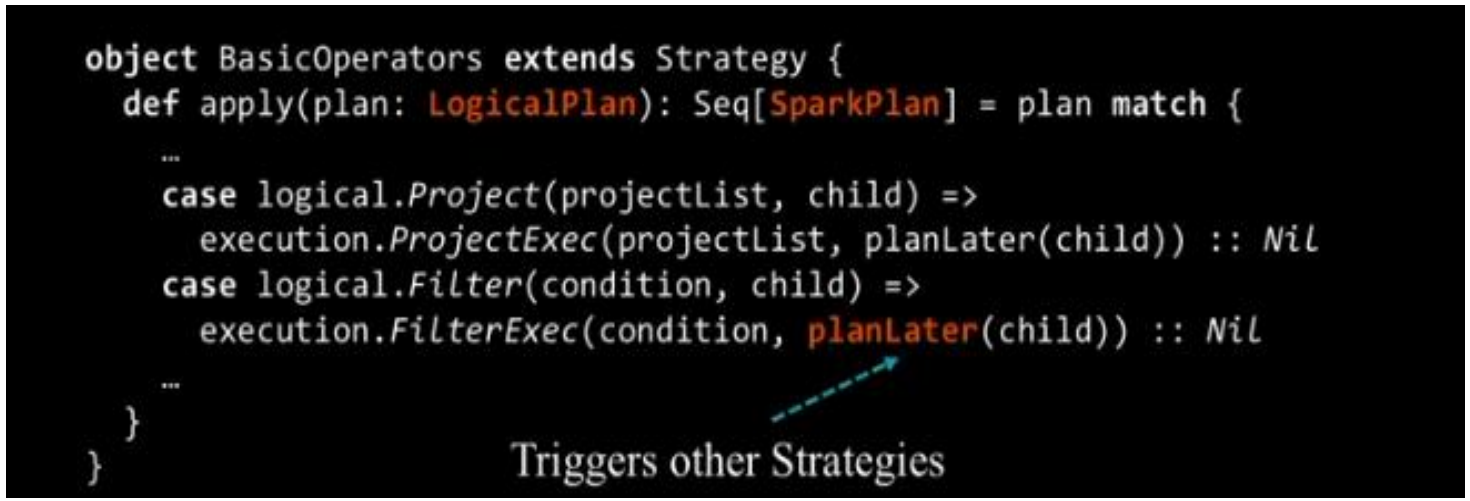
A physical plan is Executable.

Note:
T1: dataframe 1 is created using parquet file
T2: dataframe 2 is created using CSV file

Catalyst Optimizer: Transforming Logical plan to physical plan(Transformation2)

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→A Logical plan is transformed to physical plan by applying a set of **Strategies**.
→Every Strategy uses pattern match or basic operators to convert logical plan to physical plan. For example if you in the below Image Logical Project is converting to Physical project that is ProjectExec. Some times single strategy may not be able to convert all kind of a logical plan in such cases we will call **planLater** it will trigger different kind of Strategies it combine all of the strategies to convert logical plan tree to physical plan tree.



Query Plan: Physical Plan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The purpose of this phase is to take the logical plan and turn it into a physical plan which can be then executed. Unlike the logical plan which is very abstract, the physical plan is much more specific regarding details about the execution, because it contains a concrete choice of algorithms that will be used during the execution.

The physical planning is also composed of two steps because there are two versions of the physical plan;

1) spark plan
2) executed plan

The spark plan is created using so-called strategies where each node in a logical plan is converted into one or more operators in the spark plan. One example of a strategy is JoinSelection, where Spark decides what algorithm will be used to join the data

After the spark plan is generated, there is a set of additional rules that are applied to it to create the final version of the physical plan which is the executed plan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

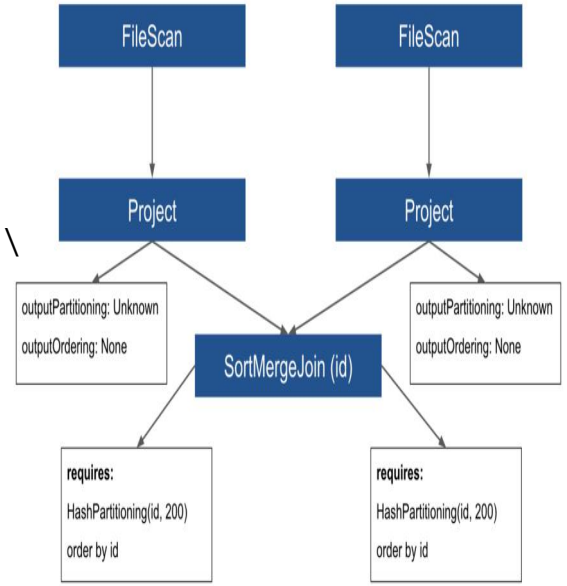
One of these additional rules that are used to transform the spark plan into the executed plan is called **EnsureRequirements** and this rule is going to make sure that the data is distributed correctly as is required by some transformations (for example joins and aggregations).

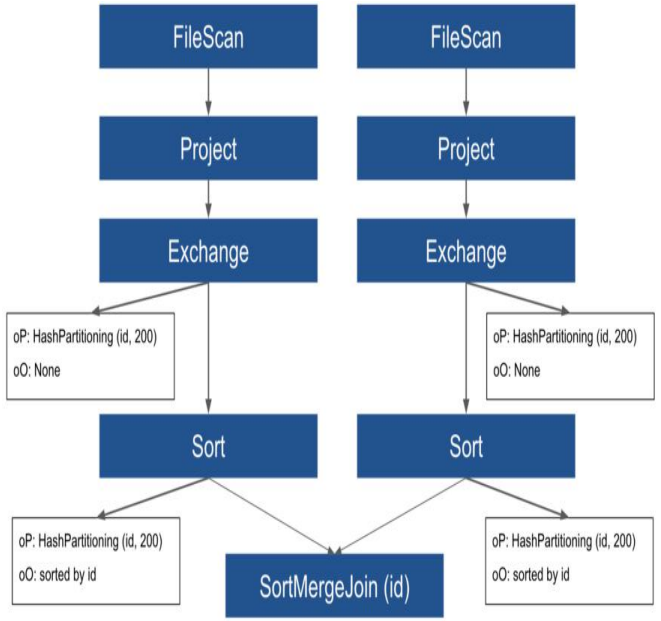
Each operator in the physical plan is having two important properties **outputPartitioning** and **outputOrdering** which carry the information about the data distribution, how the data is **partitioned and sorted** at the given moment.

Besides that, each operator also has two other properties **requiredChildDistribution** and **requiredChildOrdering** by which it puts requirements on the values of **outputPartitioning** and **outputOrdering** of its child nodes.

Let's see this on a simple example with SortMergeJoin, which is an operator that has strong requirements on its child nodes, it requires that the data must be partitioned and sorted by the joining key so it can be merged correctly.

```
spark.table("tableA") \
.join(spark.table("tableB"), "id") \
.write
```





From the spark plan we can see that the child nodes of the SortMergeJoin (two Project operators) have no oP or oO (they are Unknown and None) and this is a general situation where the data has not been repartitioned in advance and the tables are not bucketed. When the ER rule is applied on the plan it can see that the requirements of the SortMergeJoin are not satisfied so it will fill Exchange and Sort operators to the plan to meet the requirements. The Exchange operator will be responsible for repartitioning the data to meet the requiredChildDistribution requirement and the Sort will order the data to meet the requiredChildOrdering

Query Plan: Physical Plan:Bucketing

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Bucketing is a technique for storing the data in a pre-shuffled and possibly pre-sorted state where the information about bucketing is stored in the metastore.

In such a case the FileScan operator will have the **outputPartitioning** set according to the information from the metastore.

if there is exactly one file per bucket, the **outputOrdering** will be also set and it will all be passed downstream to the Project.

If both tables were bucketed by the joining key to the same number of buckets, the requirements for the **outputPartitioning** will be satisfied and the ER rule will add no Exchange to the plan.

The same number of partitions on both sides of the join is crucial here and if these numbers are different, Exchange will still have to be used for each branch where the number of partitions differs from spark.sql.shuffle.partitions configuration setting (default value is 200). So with a correct bucketing in place, the join can be shuffle-free.

```
spark.sql.sources.bucketing.enabled=true
```

```
df.write\
   .bucketBy(16, 'key') \
   .sortBy('value') \
   .saveAsTable('bucketed', format='parquet')
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

There is a function **repartition** that can be used to change the distribution of the data on the Spark cluster. The function takes as argument columns by which the data should be distributed (optionally the first argument can be the number of partitions that should be created).
What happens under the hood is that it adds a **RepartitionByExpression** node to the logical plan which is then converted to Exchange in the spark plan using a strategy and it sets the oP to HashPartitioning with the key being the column name used as the argument.

Another usage of the repartition function is that it can be called with only one argument being the number of partitions that should be created (repartition(n)), which will distribute the data randomly.

Lets see the power of repartition with two examples.
Example I: One-side shuffle-free join

```
# match number of buckets in the right branch of the join with the number of shuffle partitions:
spark.conf.set("spark.sql.shuffle.partitions", 50)
spark.table("tableA") \
.repartition(50, "id") \
.join(spark.table("tableB"), "id") \
.write \
...
```

Let's see what happens if one of the tables in the above join is bucketed and the other is not. In such a case the requirements are not satisfied because the oP is different on both sides (on one side it is defined by the bucketing and on the other side it is Unknown). In this case, the ER rule will add Exchange to both branches of the join so each side of the join will have to be shuffled! Spark will simply neglect that one side is already pre-shuffled and will waste this opportunity to avoid the shuffle. Here we can simply use repartition on the other side of the join to make sure that oP is set before the ER rule checks it and adds Exchanges.

Calling repartition will add one Exchange to the left branch of the plan but the right branch will stay shuffle-free because requirements will now be satisfied and ER rule will add no more Exchanges. So we will have only one shuffle instead of two in the final plan. Alternatively, we could change the number of shuffle partitions to match the number of buckets in tableB, in such case the repartition is not needed (it would bring no additional benefit), because the ER rule will leave the right branch shuffle-free and it will adjust only the left branch

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Example II: Aggregation followed by a join

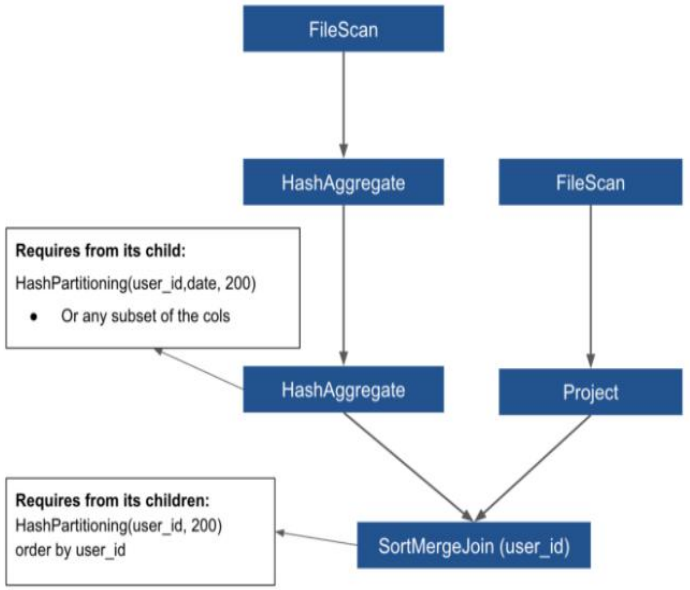
Another example where repartition becomes useful is related to queries where we aggregate a table by two keys and then join an additional table by one of these two keys (neither of these tables is bucketed in this case). Let's see a simple example which is based on transactional data of this kind:

```
{"id": 1, "user_id": 100, "price": 50, "date": "2020-06-01"}
{"id": 2, "user_id": 100, "price": 200, "date": "2020-06-02"}
{"id": 3, "user_id": 101, "price": 120, "date": "2020-06-01"}
```

Each user can have many rows in the dataset because he/she could have made many transactions. These transactions are stored in tableA. On the other hand, tableB will contain information about each user (name, address, and so on). The tableB has no duplicities, each record belongs to a different user. In our query we want to count the number of transactions for each user and date and then join the user information:

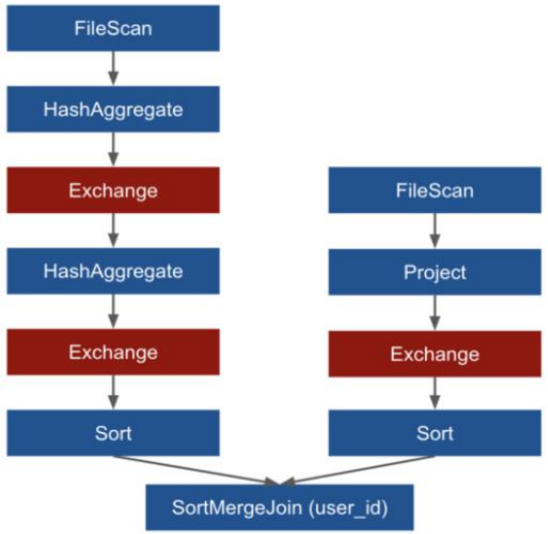
```
dfA = spark.table("tableA") # transactions (not bucketed)
dfB = spark.table("tableB") # user information (not bucketed)
dfA \
.groupBy("user_id", "date") \
.agg(count("*")) \
.join(dfB, "user_id")
```

In the spark plan, you can see a pair of HashAggregate operators, the first one (on the top) is responsible for a partial aggregation and the second one does the final merge. The requirements of the SortMergeJoin are the same as previously. The interesting part of this example are the HashAggregates. The first one has no requirements from its child, however, the second one requires for the oP to be HashPartitioning by user_id and date or any subset of these columns and this is what we will take advantage of shortly. In the general case, these requirements are not fulfilled so the ER rule will add Exchanges (and Sorts). This will lead to this executed plan:



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

As you can see we end up with a plan that has three Exchange operators, so three shuffles will happen during the execution



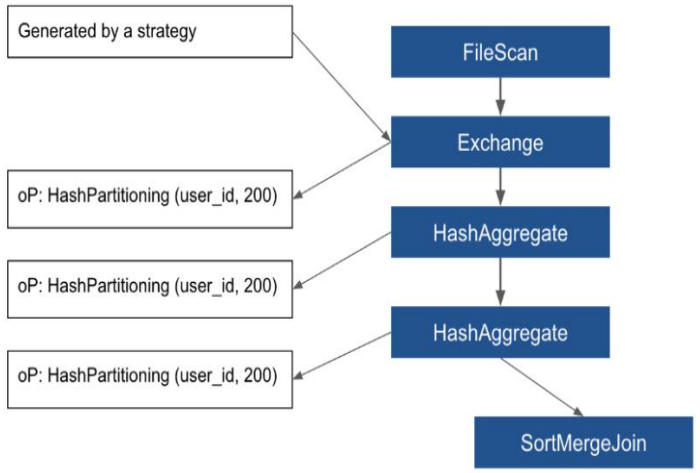
Let's now see how using repartition can change the situation:

```
dfA =
spark.table("tableA").repartition("user_id")
dfB = spark.table("tableB")
dfA \
.groupBy("user_id", "date") \
.agg(count("*")) \
.join(dfB, "user_id")
```

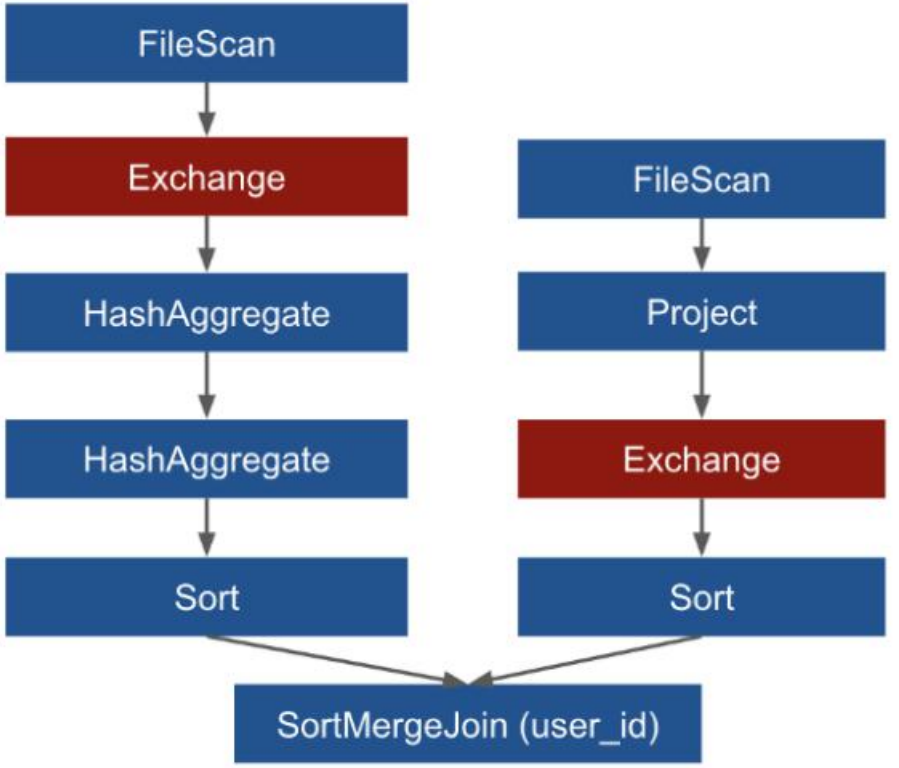
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

As you can see we end up with a plan that has three Exchange operators, so three shuffles will happen during the execution

The spark plan will now look different, it will contain Exchange that is generated by a strategy that converts RepartitionByExpression node from the logical plan. This Exchange will be a child of the first HashAggregate operator and it will set the oP to HashPartitioning (user_id) which will be passed downstream:



The requirements for oP of all operators in the left branch are now satisfied so ER rule will add no additional Exchanges (it will still add Sort to satisfy oO). The essential concept in this example is that we are grouping by two columns and the requirements of the HashAggregate operator are more flexible so if the data will be distributed by any of these two fields, the requirements will be met. The final executed plan will have only one Exchange in the left branch (and one in the right branch) so using repartition we reduced the number of shuffles by one:

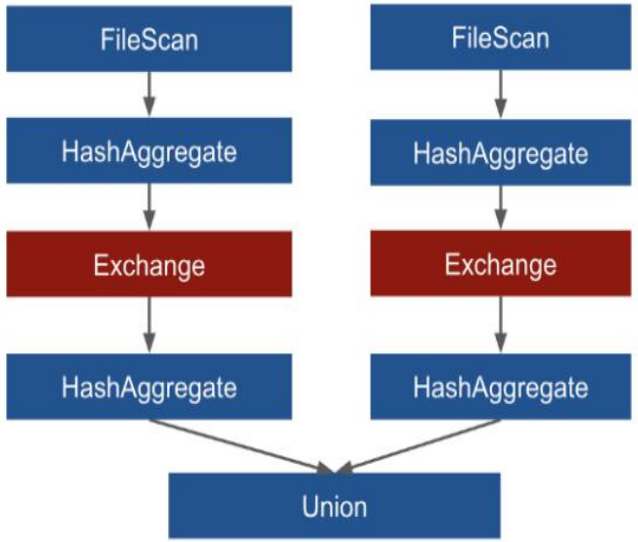


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Example III: Union of two aggregations

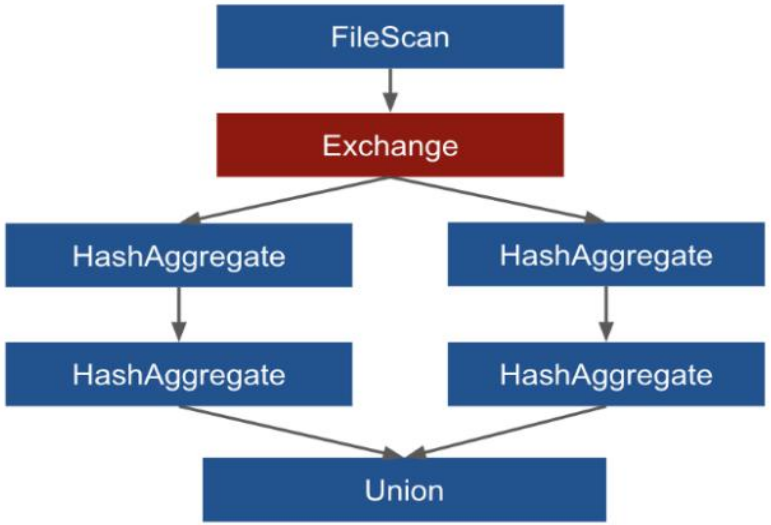
Let's consider one more example where repartition will bring optimization to our query. The problem is based on the same data as the previous example. Now in our query we want to make a union of two different aggregations, in the first one we will count the rows for each user and in the second we will sum the price column:

```
countDF = df.groupBy("user_id") \
.agg(count("*").alias("metricValue")) \
.withColumn("metricName", lit("count"))

sumDF = df.groupBy("user_id") \
.agg(sum("price").alias("metricValue")) \
.withColumn("metricName", lit("sum"))

countDF.union(sumDF)
```



It is a typical plan for a union-like query, one branch for each DataFrame in the union. We can see that there are two shuffles, one for each aggregation. Besides that, it also follows from the plan that the dataset will be scanned twice. Here the repartition function together with a small trick can help us to change the shape of the plan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



The repartition function will move the Exchange operator before the HashAggregate and it will make the Exchange sub-branches identical so it will be reused by another rule called ReuseExchange. In the count function, changing the star to the price column becomes important here because it will make sure that the projection will be the same in both DataFrames (we need to project the price column also in the left branch to make it the same as the second branch). It will however produce the same result as the original query only if there are no null values in the price column.

Similarly as before, we reduced here the number of shuffles by one, but again we have now a total shuffle as opposed to reduced shuffles in the original query. The additional benefit here is that after this optimization the dataset will be scanned only once because of the reused computation.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark splits data into partitions and executes computations on the partitions in parallel. You should understand how data is partitioned and when you need to manually adjust the partitioning to keep your Spark computations running efficiently.

Coalesce:

The coalesce method reduces the number of partitions in a DataFrame. you cannot increase the number of partitions using coalesce.

val newDF = DF.coalesce(2)

Repartition:

The repartition method can be used to either increase or decrease the number of partitions in a DataFrame. Replace will perform a full shuffle and make sure data is equally distributed across the partitions.

val newDF = DF. repartition(2)
val newDF = DF. repartition(6)

You can also perform repartition based on a columns as well and When partitioning by a column, Spark will create a minimum of 200 partitions by default. Open UI and check task execution time if one task is taking more time and other are taking less time that means data is not partitioned properly across the partitions in this case use coalesce or repartition the dataframe with partition count = number of cpus * 4

Differences between coalesce and repartition

The repartition algorithm does a full shuffle of the data and creates equal sized partitions of data. coalesce combines existing partitions to avoid a full shuffle.

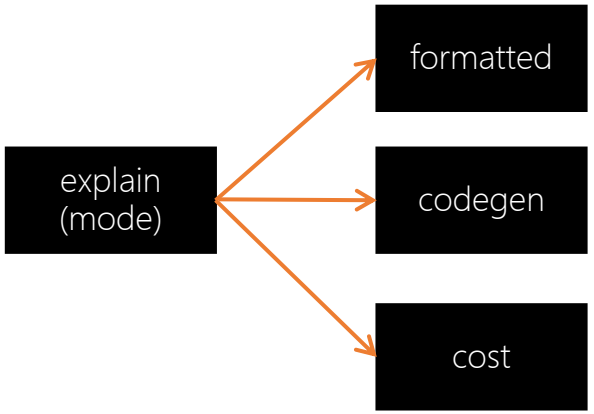
Query Plan: Physical Plan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

There are two basic ways how to see the physical plan. The first one is by calling explain function on a DataFrame which shows a textual representation of the plan:

```
query.explain()
```

```
== Physical Plan ==
*(3) Project [user_id#216L, cnt#390L, display_name#231, about#232, location#233, downvotes#234L, upvotes#235L, reputa
tion#236L, views#237L]
+- *(3) BroadcastHashJoin [user_id#216L], [user_id#230L], Inner, BuildRight
   :- *(3) HashAggregate(keys=[user_id#216L], functions=[count(1)])
   :  +- Exchange hashpartitioning(user_id#216L, 200), true, [id=#332]
   :     +- *(1) HashAggregate(keys=[user_id#216L], functions=[partial_count(1)])
   :        +- *(1) Project [user_id#216L]
   :           +- *(1) Filter isnotnull(user_id#216L)
   :              +- *(1) ColumnarToRow
   :                 +- FileScan parquet [user_id#216L,year#218] Batched: true, DataFilters: [isnotnull(user_id#216
L)], Format: Parquet, Location: InMemoryFileIndex[file:/Users/david.vrba/data/questions], PartitionFilters: [isnotnul
l(year#218), (year#218 = 2019)], PushedFilters: [IsNotNull(user_id)], ReadSchema: struct<user_id:bigint>
   +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, bigint, true])), [id=#340]
      +- *(2) Project [user_id#230L, display_name#231, about#232, location#233, downvotes#234L, upvotes#235L, reputat
ion#236L, views#237L]
         +- *(2) Filter isnotnull(user_id#230L)
            +- *(2) ColumnarToRow
               +- FileScan parquet [user_id#230L,display_name#231,about#232,location#233,downvotes#234L,upvotes#235L,
reputation#236L,views#237L] Batched: true, DataFilters: [isnotnull(user_id#230L)], Format: Parquet, Location: InMemor
yFileIndex[file:/Users/david.vrba/data/users], PartitionFilters: [], PushedFilters: [IsNotNull(user_id)], ReadSchema:
struct<user_id:bigint,display_name:string,about:string,location:string,downvotes:bigint,upvotes:b...
```




```
query.explain(mode='formatted')
```

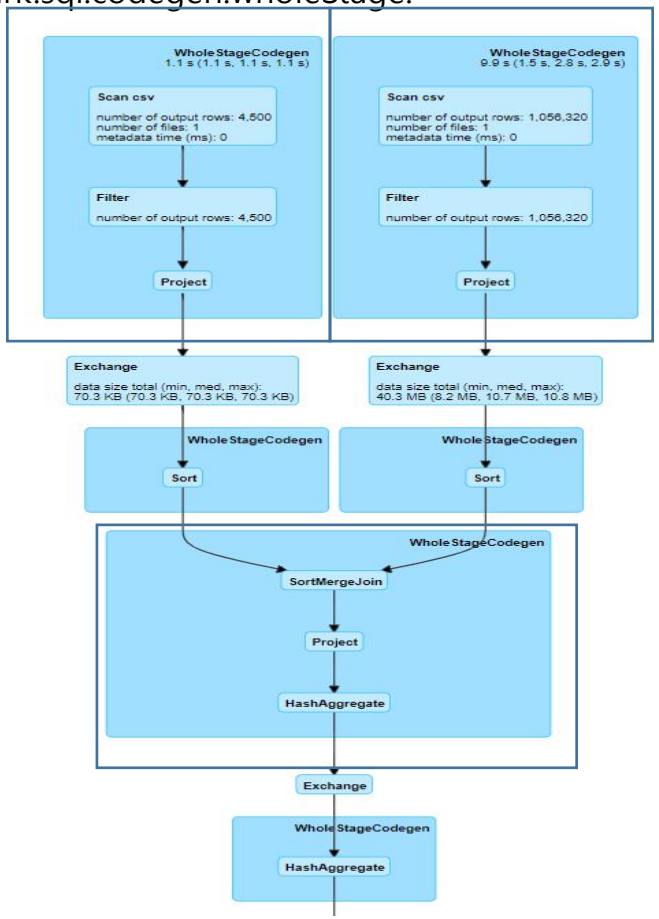
```
== Physical Plan ==
* Project (14)
+- * BroadcastHashJoin Inner BuildRight (13)
   :- * HashAggregate (7)
   :  +- Exchange (6)
   :     +- * HashAggregate (5)
   :        +- * Project (4)
   :           +- * Filter (3)
   :              +- * ColumnarToRow (2)
   :                 +- Scan parquet  (1)
   +- BroadcastExchange (12)
      +- * Project (11)
         +- * Filter (10)
            +- * ColumnarToRow (9)
               +- Scan parquet  (8)


(1) Scan parquet
Output [2]: [user_id#216L, year#218]
Batched: true
Location: InMemoryFileIndex [file:/Users/david.vrba/data/questions]
PartitionFilters: [isnotnull(year#218), (year#218 = 2019)]
PushedFilters: [IsNotNull(user_id)]
ReadSchema: struct<user_id:bigint>

(2) ColumnarToRow [codegen id : 1]
Input [2]: [user_id#216L, year#218]

(3) Filter [codegen id : 1]
Input [2]: [user_id#216L, year#218]
Condition : isnotnull(user_id#216L)

(4) Project [codegen id : 1]
Output [1]: [user_id#216L]
Input [2]: [user_id#216L, year#218]
```

Query Plan: Physical Plan:CollapseCodegenStages

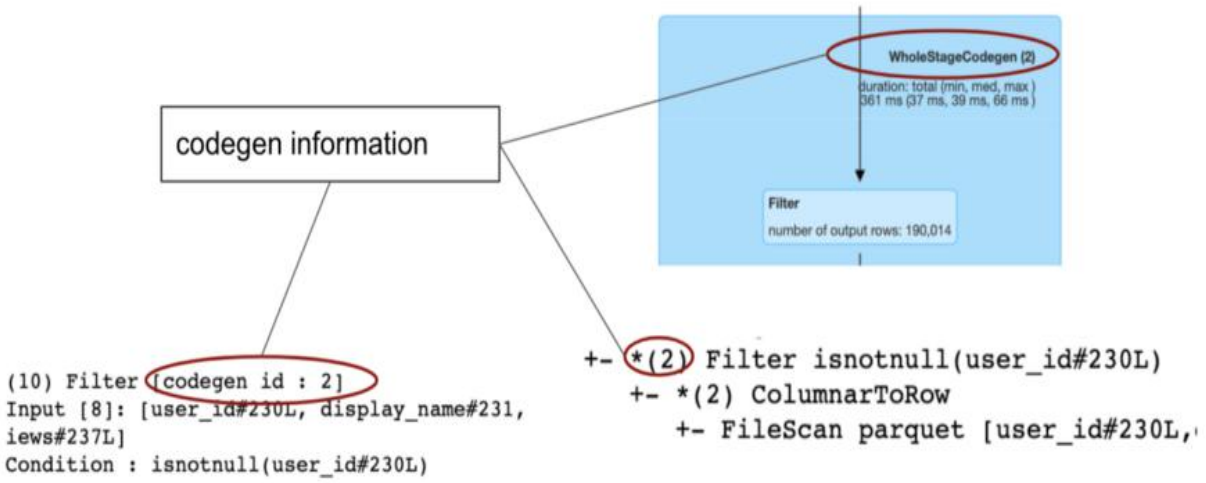
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Whole-Stage CodeGen is also known as Whole-Stage Java Code Generation, which is a physical query optimization phase in Spakr SQL that clubs multiple physical operations together to form a single Java function.Whole-Stage Java code generation improves the execution performance by converting a query tree into an optimized function that eliminates unnecessary calls and leverages CPU registers for intermediate data.

spark.sql.codegen.wholeStage.



These big rectangles correspond to codegen stages. It is an optimization feature, which takes place in the phase of physical planning. There is a rule called **CollapseCodegenStages** which is responsible for that and the idea is to take operators that support code generation and collapse it together to speed-up the execution by eliminating virtual function calls.Not all operators support code generation, so some operators (for instance Exchange) are not part of the big rectangles.Also from the tree, you can tell if an operator supports the codegen or not because there is an asterisk with corresponding stage codegen id in the parenthesis if the codegen is supported.



Query Plan: Physical Plan:Scan

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The Scan parquet operator represents reading the data from a csvfile format. From the detailed information, you can directly see what columns will be selected from the source. Even though we do not select specific fields in our query, there is a **ColumnPruning** rule in the optimizer that will be applied and it makes sure that only those columns that are actually needed will be selected from the source.

We can also see here two types of filters: **PartitionFilters and PushedFilters**.
The PartitionFilters are filters that are applied on columns by which the datasource is partitioned in the file system. These are very important because they allow for skipping the data that we don't need. It is always good to check whether the filters are propagated here correctly. The idea behind this is to read as little data as possible since the I/O is expensive.

The PushedFilters are on the other hand filters on fields that can be pushed directly to parquet files and they can be useful if the parquet file is sorted by these filtered columns because in that case, we can leverage the internal parquet structure for data skipping as well. The parquet file is composed of row groups and the footer of the file contains metadata about each of these row groups. This metadata contains also statistical information such as min and max value for each row group and based on this information Spark can decide whether it will read the row group or not.

Query Plan: Physical Plan:Filter and project

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The Filter operator is quite intuitive to understand, it simply represents the filtering condition.

PushDownPredicates — this rule will push filters closer to the source through several other operators, but not all of them. For example, it will not push them through expressions that are not deterministic. If we use functions such as first, last, collect_set, collect_list, rand (and some other) the Filter will not be pushed through them because these functions are not deterministic in Spark.

CombineFilters — combines two neighboring operators into one (it collects the conditions from two following filters into one complex condition).

InferFiltersFromConstraints — this rule actually creates a new Filter operator for example from a join condition (from a simple inner join it will create a filter condition joining key is not null).

PruneFilters — removes redundant filters (for example if a filter always evaluates to True).

Project operator simply represents what columns will be projected (selected). Each time we call select, withColumn, or drop transformations on a DataFrame, Spark will add the Project operator to the logical plan which is then converted to its counterpart in the physical plan. Again there are some optimization rules applied to it before it is converted:

ColumnPruning — this is a rule we already mentioned above, it prunes the columns that are not needed to reduce the data volume that will be scanned.

CollapseProject — it combines neighboring Project operators into one.

PushProjectionThroughUnion — this rule will push the Project through both sides of the Union operator.

Query Plan: Physical Plan:Exchange

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The Exchange operator represents shuffle, which is a physical data movement on the cluster. This operation is considered to be quite expensive because it moves the data over the network. The information in the query plan contains also details about how the data will be repartitioned. In our example, it is hashpartitioning(user_id, 200) as you can see below:
Image for post

This means that the data will be repartitioned according to the user_id column into 200 partitions and all rows with the same value of user_id will belong to the same partition and will be located on the same executor. To make sure that exactly 200 partitions are created, Spark will always compute the hash of the <join column >and then will compute positive modulo 200. The consequence of this is that more different user_ids will be located in the same partition. And what can also happen is that some partitions can become empty. There are other types of partitioning worth to mention:

RoundRobinPartitioning — with this partitioning the data will be distributed randomly into n approximately equally sized partitions, where n is specified by the user in the repartition(n) function

SinglePartition — with this partitioning all the data are moved to a single partition to a single executor. This happens for example when calling a window function where the window becomes the whole DataFrame (when you don't provide an argument to the partitionBy() function in the Window definition).

RangePartitioning — this partitioning is used when sorting the data, after calling orderBy or sort transformations.

```
(6) Exchange
Input [2]: [user_id#216L, count#311L]
Arguments: hashpartitioning(user_id#216L, 200), true, [id=#198]
```

Query Plan: Physical Plan:HashAggregate

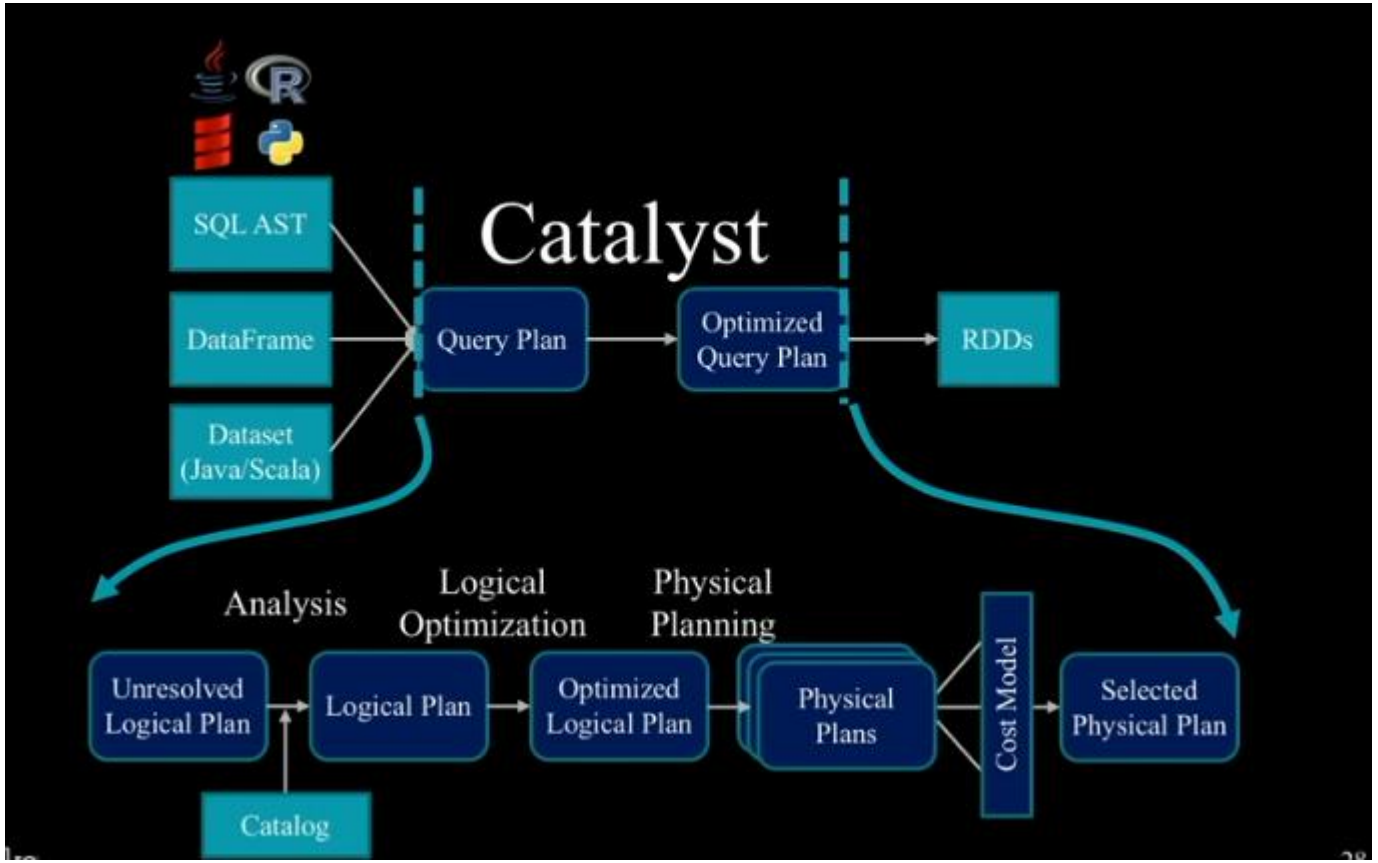
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

This operator represents data aggregation. It usually comes in pair of two operators which may or may not be divided by an Exchange as you can see here:

```
(5) HashAggregate [codegen id : 1]
Input [1]: [user_id#216L]
Keys [1]: [user_id#216L]
Functions [1]: [partial_count(1)]
Aggregate Attributes [1]: [count#310L]
Results [2]: [user_id#216L, count#311L]

(6) Exchange
Input [2]: [user_id#216L, count#311L]
Arguments: hashpartitioning(user_id#216L, 200), true, [id=#198]

(7) HashAggregate [codegen id : 3]
Input [2]: [user_id#216L, count#311L]
Keys [1]: [user_id#216L]
Functions [1]: [count(1)]
Aggregate Attributes [1]: [count(1)#297L]
Results [2]: [user_id#216L, count(1)#297L AS cnt#298L]
```

The reason for having two HashAggregate operators is that the first one does a partial aggregation, which aggregates separately each partition on each executor. The final merge of the partial results follows in the second HashAggregate. The operator also has the Keys field which shows the columns by which the data is grouped. The Results field shows the columns that are available after the aggregation.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Analysis(Rule Executor): Transform unresolved logical plan to Resolved Logic plan.

→ Unresolved => Resolved: Uses catalog to find where datasets and columns are coming from what are their datatypes.

Logical Optimizations(Rule Executor): Transforms a resolved logical plan to optimized logical plan by applying some optimizations like predicate push downs and columns pruning etc.

Physical Planning(Strategies + Rule Executor):

Phase1: Transforms a optimized logical plan to Physical plan

Phase2: Rule Executor is used to adjust the physical plan to make it ready for execution in this phase it will figure out how to shuffle the data and partition the data and what kind of columns are used to partition the data

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Physical storage layout models

Logical

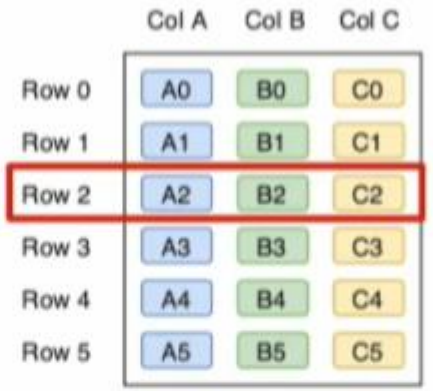


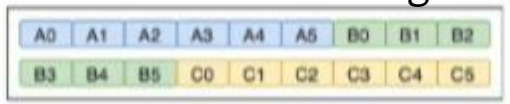
Table :
Combination of Rows and Columns

Physical

Horizontal Partitioning

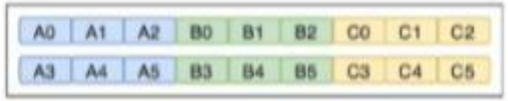
Row-wise

Vertical Partitioning



Columnar

Vertical then Horizontal Partitioning



Hybrid

OLTP(Online Transaction Processing) AND OLAP(Online Analytics Processing) Work Loads

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

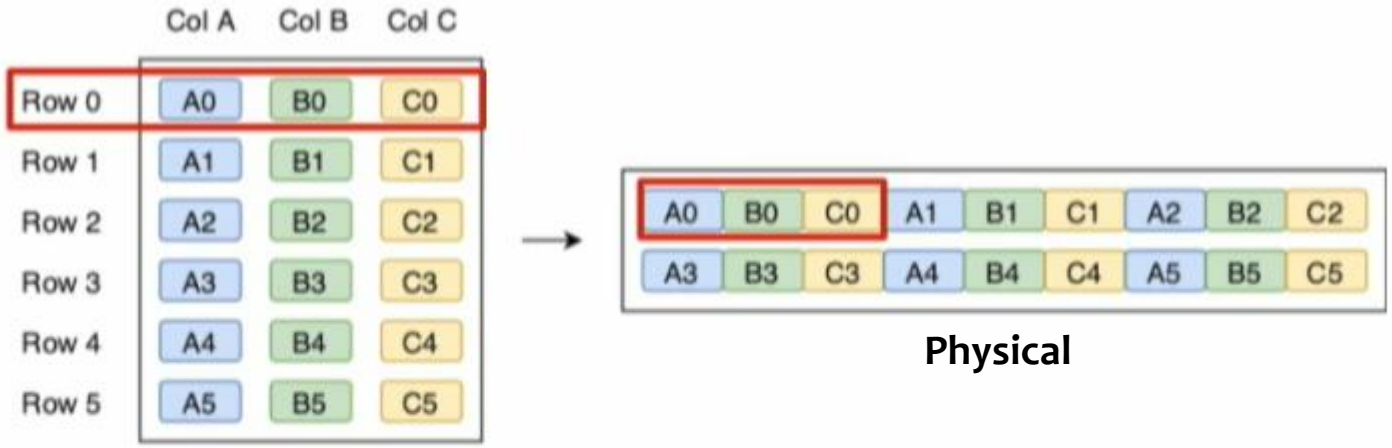
OLTP: Lots of small operations involve whole row say Insert, Delete, Update are the small operations by doing that entire row will get effected.

OLAP: Few Large operations involving in subset of columns sum, avg, count, groupby for these operations there will a large scan and end result is very small.

For OLTP and OLAP I/O are expensive(Memory, disk, Network)

OLTP AND OLAP in Row Format Context

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

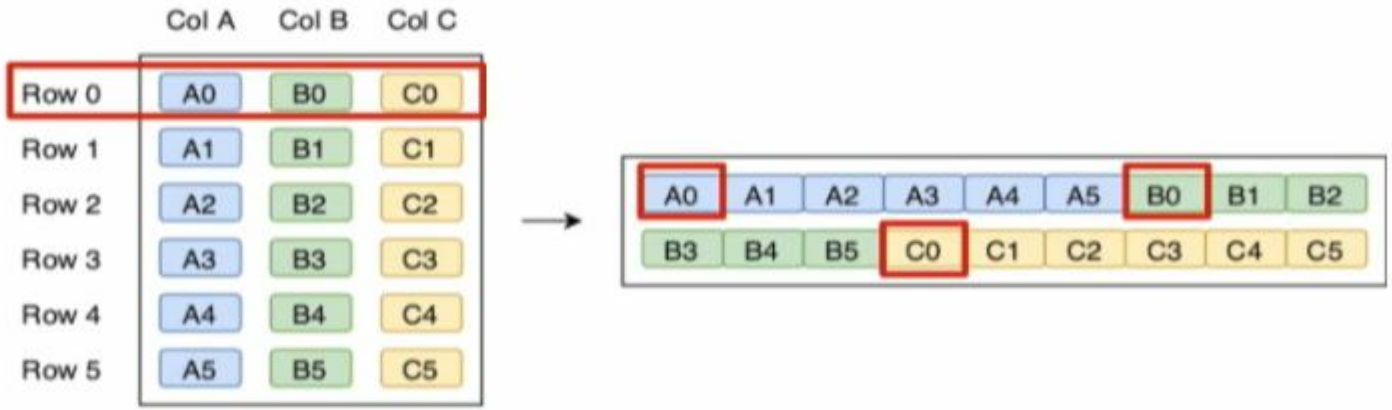


- Horizontal partitioning
- OLTP ✓, OLAP ✖

This is more suited for OLTP because lets say you have insert operation what you can do is append all these columns values at the end of your file. if its an update operation you can find the location and update the column values in place and same for the delete as well. Its not that good for OLAP because you are only interested in subsets of the columns and this model works on the base on the entire rows you can be wasting I/O reading columns values that you are not goanna read.

OLTP AND OLAP in Columnar Format Context

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



- Vertical partitioning
- OLTP ✖, OLAP ✔
  - Free projection pushdown
  - Compression opportunities

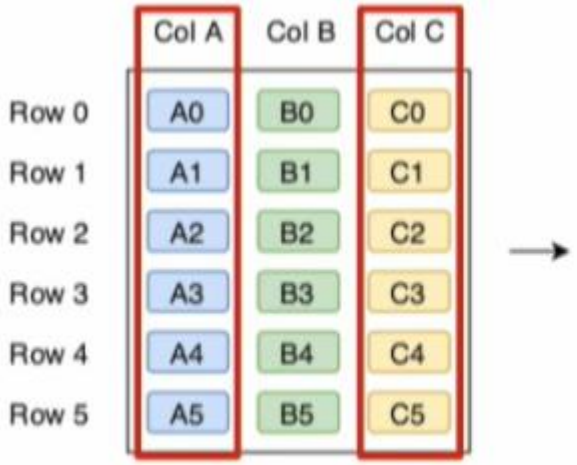
(This is called column pruning means read the columns that you are only interested in)

(As you are storing the same values in sequence you can person compression or encoding)

Instead of storing column values of row back to back what you do is store all the columns for all rows back to back and its not very suited for OTLP because if you have to insert a record then you have to insert the column values at various column locations as shown in the above figure if there is a big file you wanted to insert then it will very inefficient and goanna have **fragmented memory** access patterns and c**omputers really don't like fragmented memory pattern.** for OLAP its very good because as I said we are interested In subset of columns

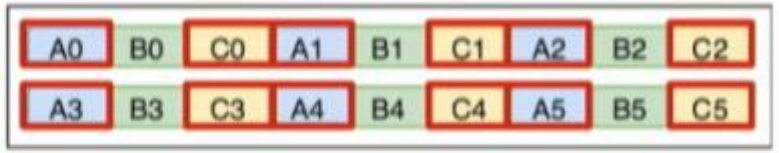
Row Vs Columnar Storage

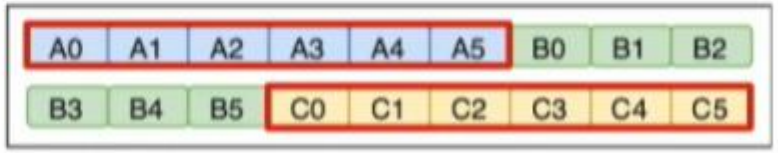
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Read two columns from table

Row format: Its very fragmented where computers doesn't like it



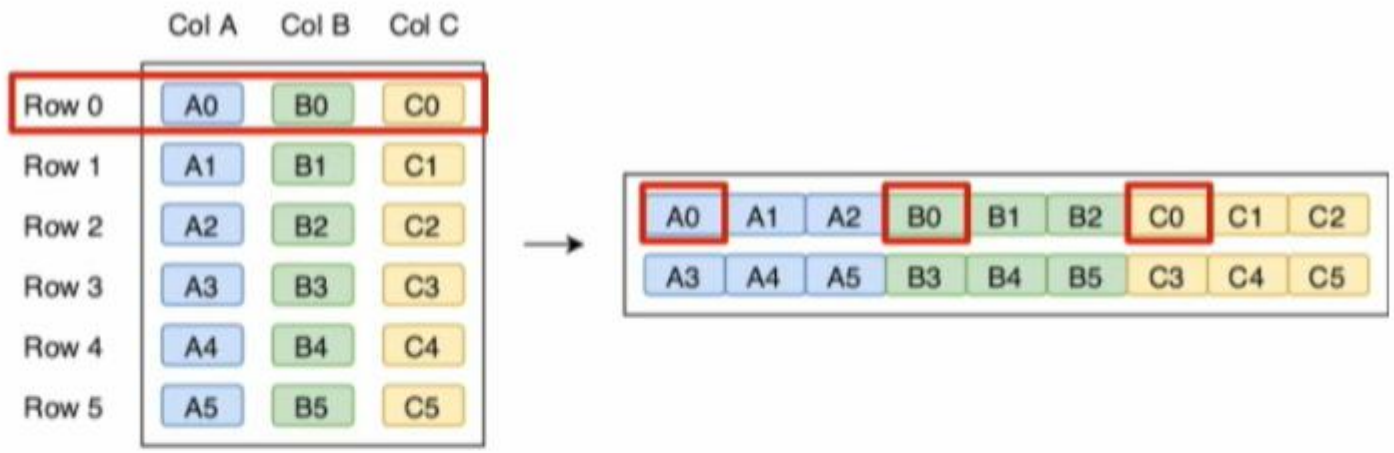


Columnar format: Its Sequential where computers like it but if you want to do row reconstruction then its very difficult with Columnar

Lets say I you have 100 gigabyte file where you wanted to reconstruct in the row then its very difficult say read as parquet to store into MySQL so columnar also doesn't work in this case

Hybrid is best for both Row and Columnar operations

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



- Horizontal & vertical partitioning
- Used by Parquet & ORC
- Best of both worlds

Parquet

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→Initial effort by Twitter and Cloudera
→Open storage format
→Hybrid Storage Model(PAX: Partitions Attributes Across)
→Widely used in Spark and Hadoop Ecosystems
→One of the widely used formats by Databricks customers

Parquet:Files

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→One disk its Multiple files

→ Logical file is defined by root directory

→ Root contains one or multiple files

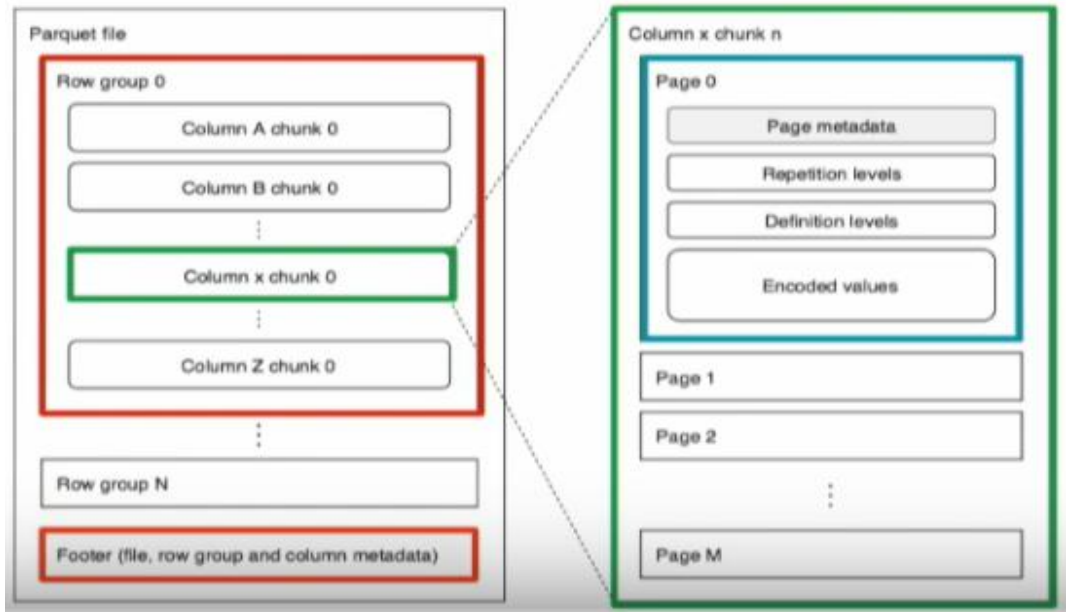
```
./example_parquet_file/
./example_parquet_file/part-00000-87439b68-7536-44a2-9eaa-1b40a236163d-c000.snappy.parquet
./example_parquet_file/part-00001-ae3c183b-d89d-4005-a3c0-c7df9a8e1f94-c000.snappy.parquet
```

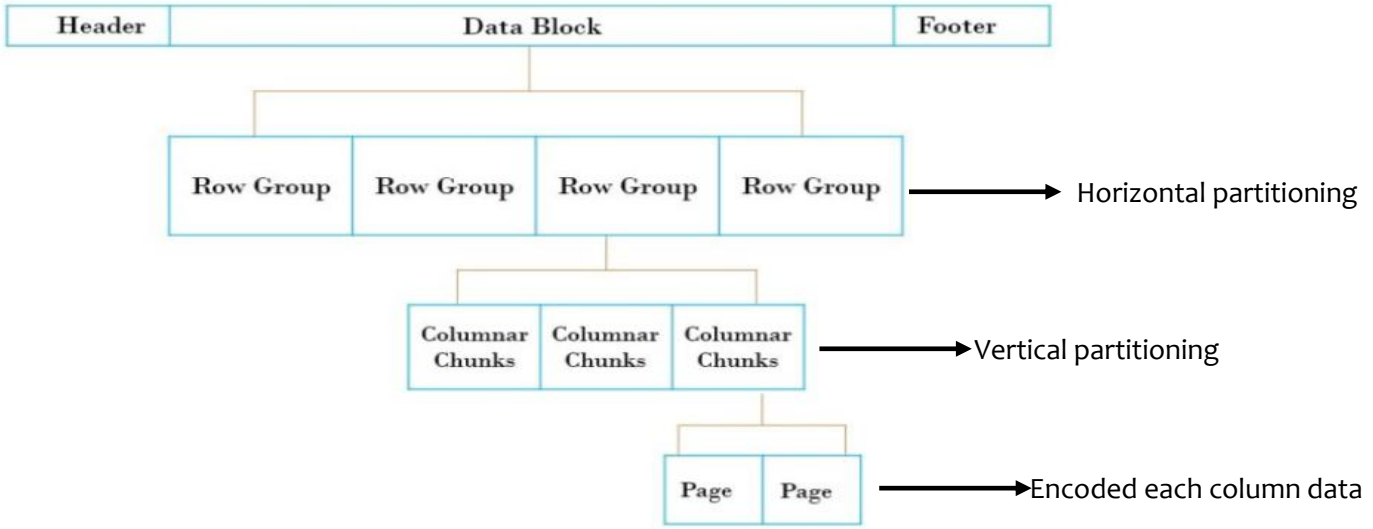
→ or contains files in sub-directories with files in left directory(partitioned directories)

```
./example_parquet_file/
./example_parquet_file/country=Netherlands/
./example_parquet_file/country=Netherlands/part-00000-...-475b15e2874d.c000.snappy.parquet
./example_parquet_file/country=Netherlands/part-00001-...-c7df9a8e1f94.c000.snappy.parquet
```

Parquet: Data Organization

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in





Header: At a high level, the parquet file consists of header, one or more blocks and footer. The parquet file format contains a 4-byte magic number in the header (PAR1) and at the end of the footer. This is a magic number indicates that the file is in parquet format. All the file metadata stored in the footer section.

Blocks, Row-Group, Chunks, Page :Each block in the parquet file is stored in the form of row groups. So, data in a parquet file is partitioned into multiple row groups. These row groups in turn consists of one or more column chunks which corresponds to a column in the data set. The data for each column chunk written in the form of pages. Each page contains values for a particular column only, hence pages are very good candidates for compression as they contain similar values. At every row group and column chunks also holds metadata information like Min value, Max value, Count. Default size of Row group: 128MB, Page : 1Mb

Footer: The footer's metadata includes the version of the format, the schema, any extra key-value pairs, and metadata for columns in the file. The column metadata would be type, path, encoding, number of values, compressed size etc. Apart from the file metadata, it also has a 4-byte field encoding the length of the footer metadata, and a 4-byte magic number (PAR1)

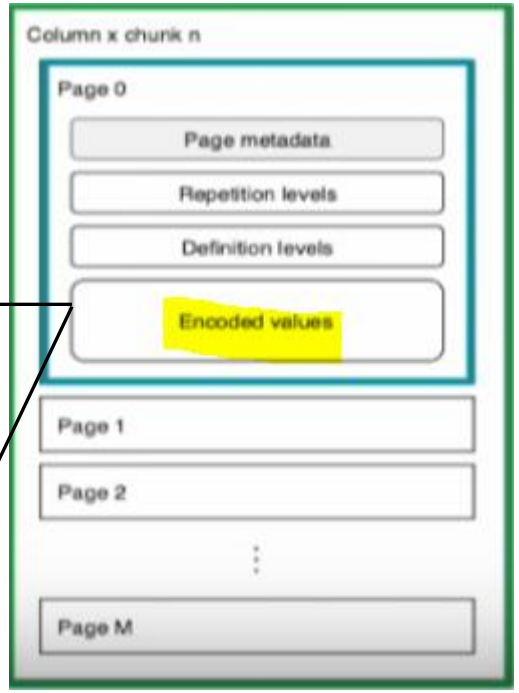
Parquet: Encoding

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→**Plain:**

Fixed Width: If there values like fixed width like Int then values will store back to back

Non Fixed Width : The length will be prefixed lets say Strings for example **INDIA:** Length is 5 it will be prefixed so that it know where to start and stop the reading.



→**RLE_Dictionary :** (RLE mean Run Length Encoding)

It will gather the unique value from column and builds a dictionary and based on that data will **encoded** which is called **dictionary compression** on top of it **bit-packing** will happen and its widely used when you have duplicated and repeated values **This is optimization technique in spark.**

→Note: If are two many unique values then dictionary will be will too big then automatically fall back to Plain

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

If file size is less I/O will decrease
Every column chunk will have one dictionary.
Note: If are two many unique values then dictionary will be will too big then automatically fall back to Plain. How can we avoid that?
→Increase Maximum Dictionary size
Property : **parquet.dictionary.page.size**
→Decrease Row group size it means sense because if you decease it will decrease the number of rows and it will decrease the number of values
Property: parquet.block.size

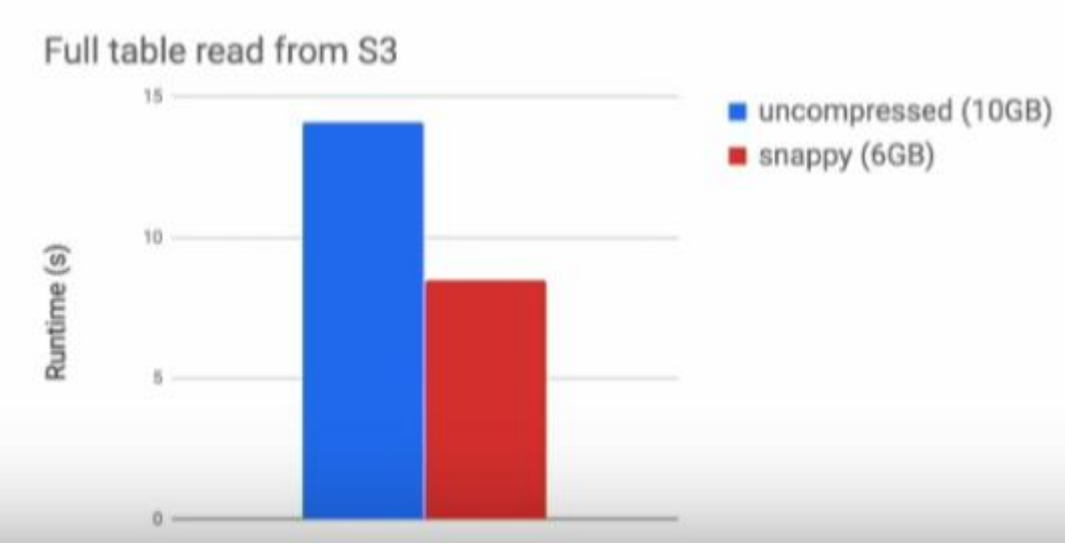
Parquet: Optimization: Page compression

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Compression of Entire Pages:

→compression schemes(snappy,gzip,lzo)
property: spark.sql.parquet.compression.codec

→ Decompression speed v/s I/O Trade off



Parquet: Optimization: Predicate Pushdown

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

While writing to parquet file at an each row-group level statics will maintained so while reading that meta data information will be read into memory so the below query is getting the records that are greater than 5 so first 2 row groups are picked and 3 one is skipped as the condition is not satisfied . As row group is 128MB in size and you are skipping that which is good thing

```
SELECT * FROM table WHERE x > 5

Row-group 0: x: [min: 0, max: 9]
Row-group 1: x: [min: 3, max: 7]
Row-group 2: x: [min: 1, max: 4]
...
```

You can enable this property by enabling the below property and by default its enabled

spark.sql.parquet.filterPushdown

Note: Predicates will not work on well unsorted data

Large value range within in the row-group then low min and high max for that pre-sort on the predicate column before writing the data as parquet

Parquet: Optimization: Equality Predicate Pushdown

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

```
SELECT * FROM table WHERE x = 5

Row-group 0: x: [min: 0, max: 9]
Row-group 1: x: [min: 3, max: 7]
Row-group 2: x: [min: 1, max: 4]
```

There is possibility of 5 in row-group 1 and 2 correct then in this case how will row-groups will be skipped. For these cases parquet has something called dictionary filtering if you remember dictionary is a collection of unique values in a column chunk so it uses that to identify whether 5 is in that chunk are not. Property to enable is **parquet.filtering.dictionary.eanbled**

Parquet: Optimization: Partitioning

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Embed predicates in directory structure.

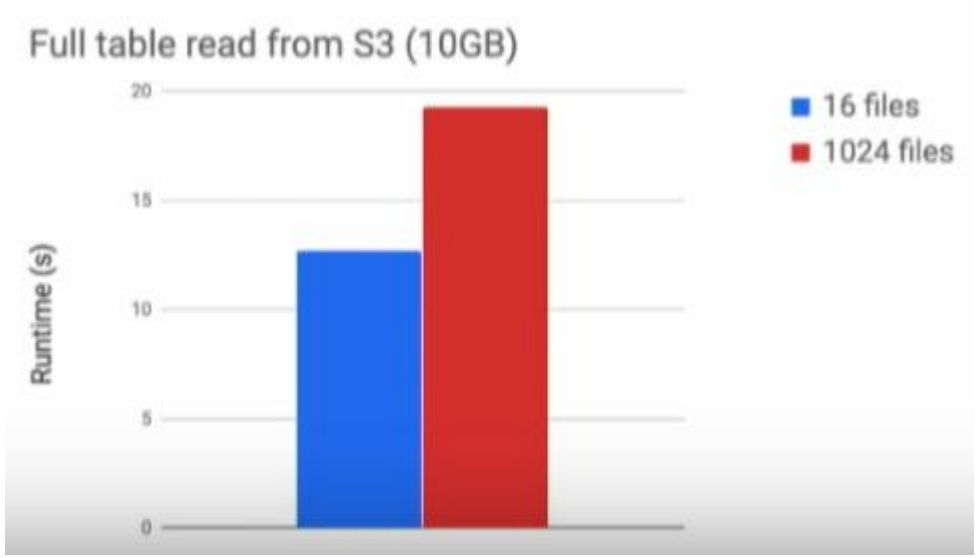
```
df.write.partitionBy("date").parquet(...)

./example_parquet_file/date=2019-10-15/...
./example_parquet_file/date=2019-10-16/...
./example_parquet_file/date=2019-10-17/part-00000-...-475b15e2874d.c000.snappy.parquet
```

Parquet: Optimization: Avoid very small files

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

For Every file

- Setup internal data structure
- Instantiate reader objects
- Fetch file
- Parse Parquet partitions



Parquet: Optimization: to Avoid very small files

```
df.repartition(numPartitions).write.parquet(...)
```

or

```
df.coalesce(numPartitions).write.parquet(...)
```

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Kafka

Definition

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Kafka is a High Throughput Distributed Messaging System used to build Low Latency System.

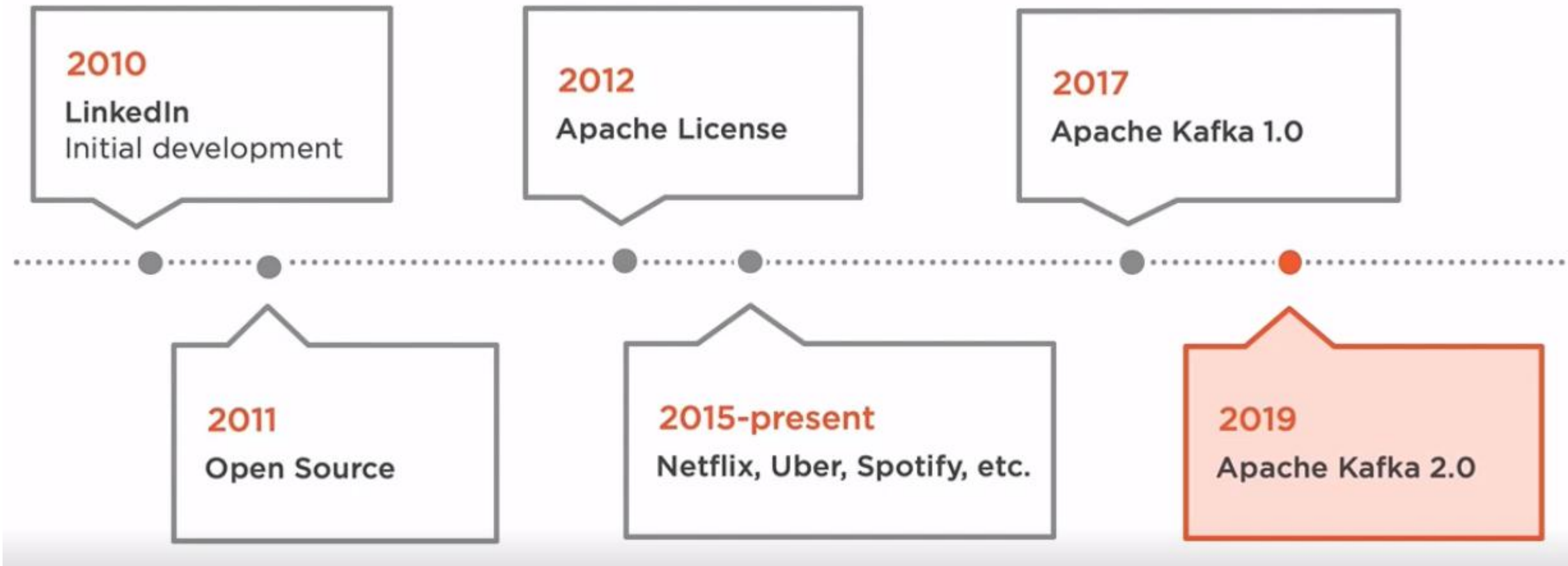
100Mb/Sec

100Mb = Through Put(How much data is getting transferred from source system to Target System)

Sec = Latency (How many time it takes to Transfer)

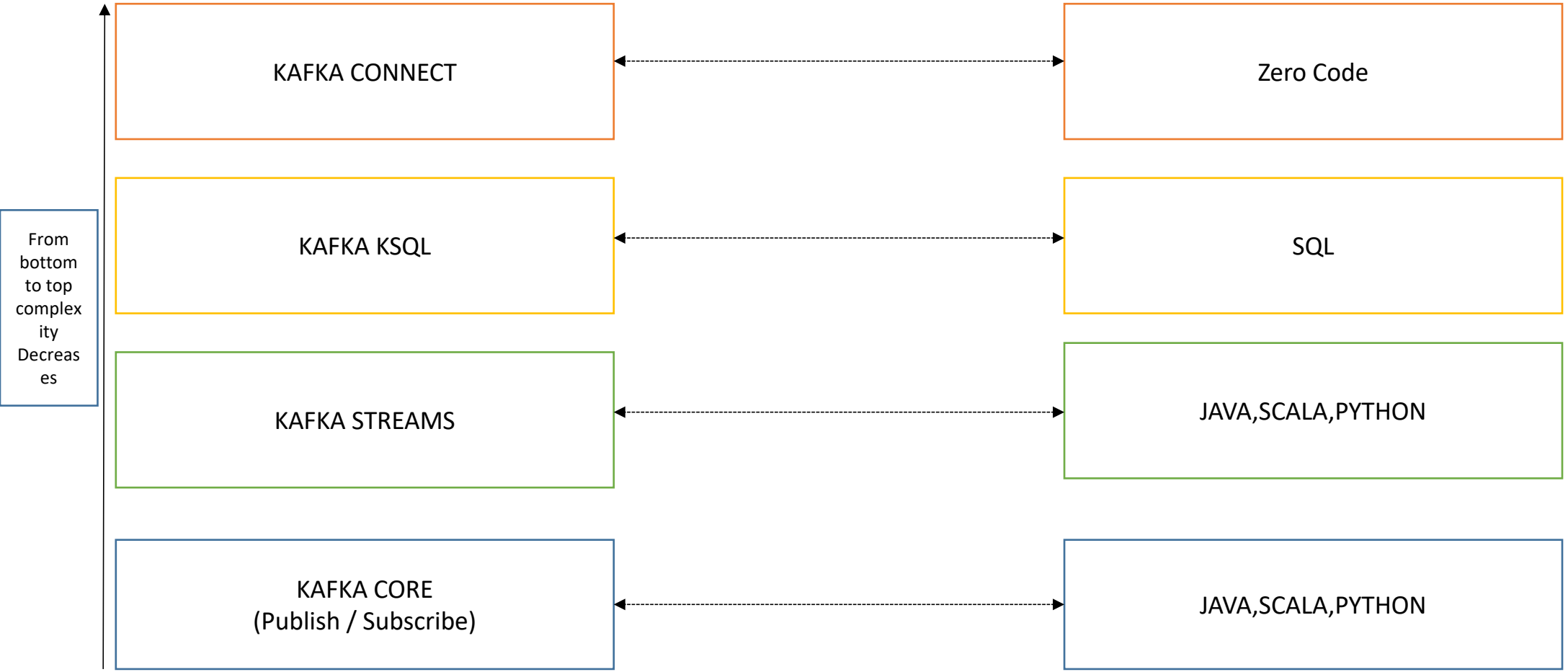
History of Kafka

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

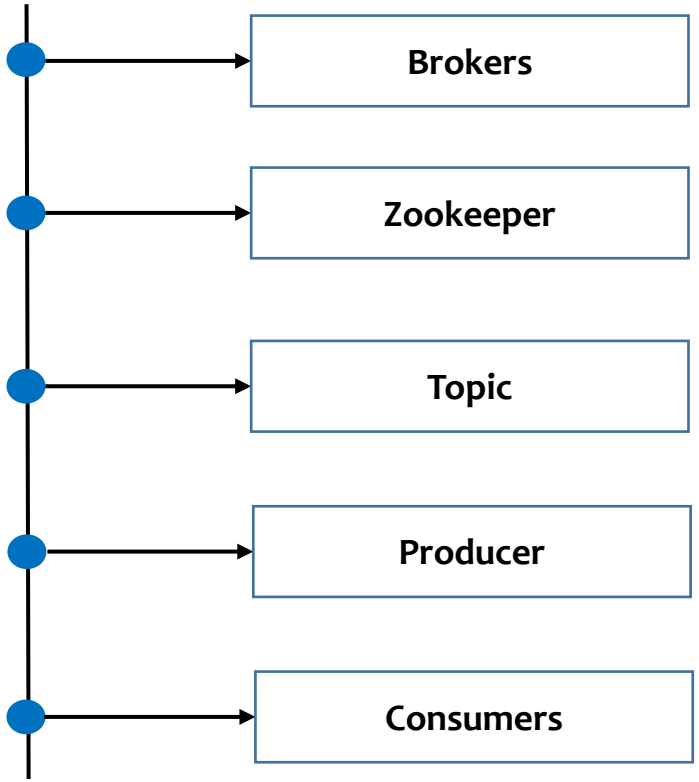


Who all can work with Kafka?

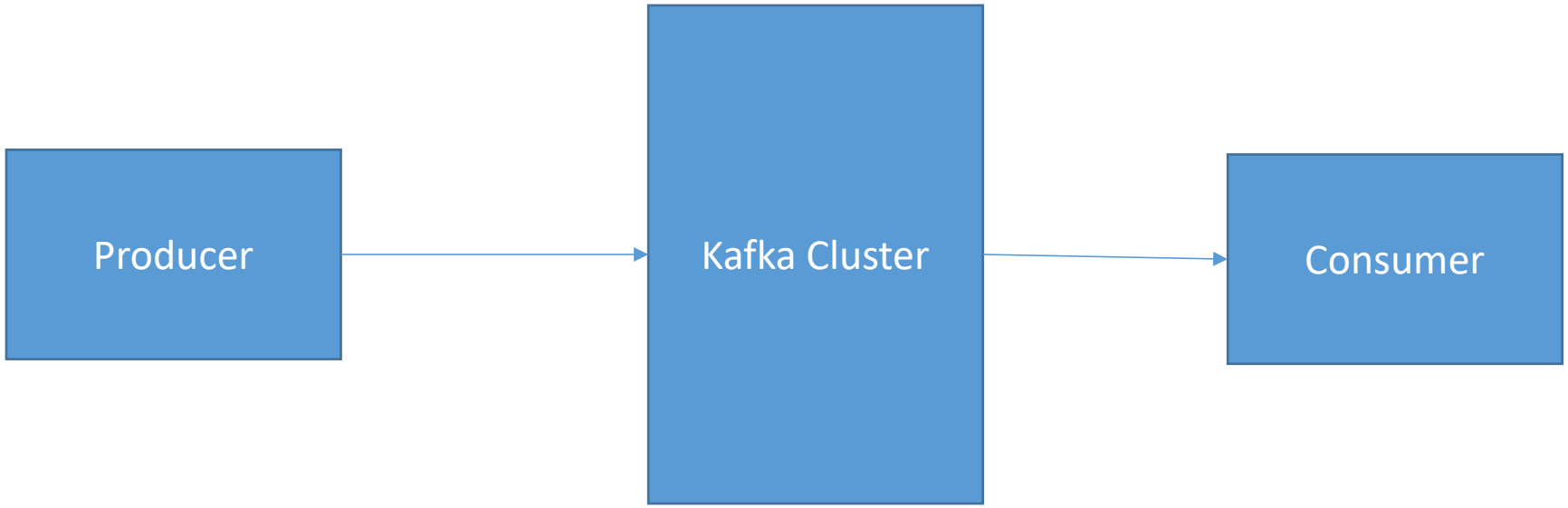
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

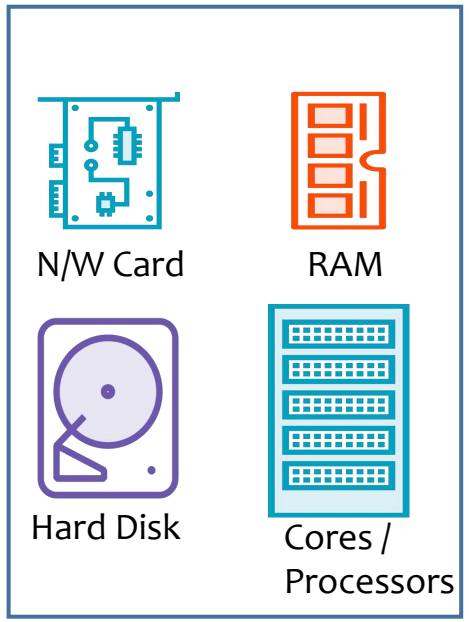


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

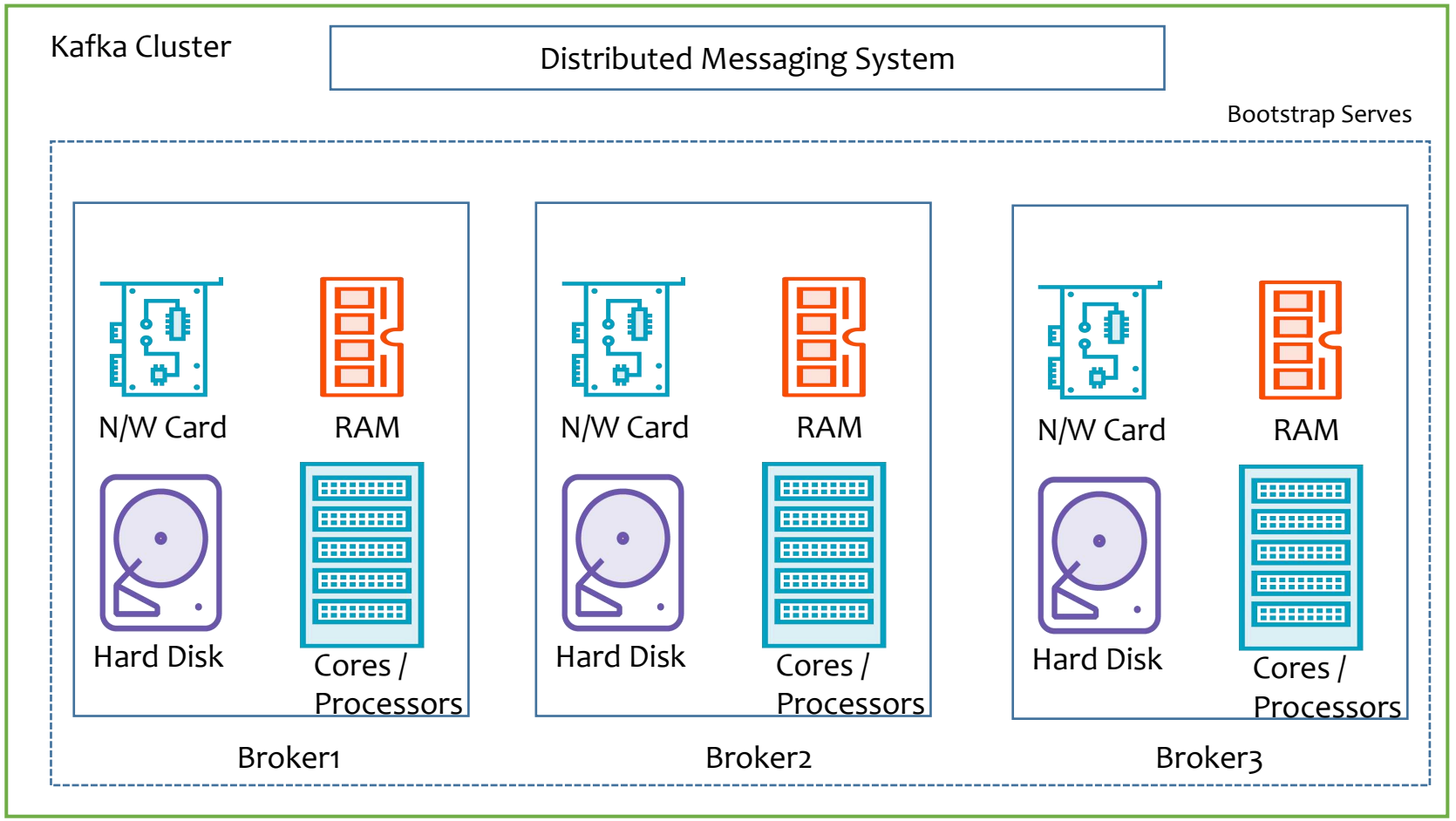


What is a Brokers/Bootstrap servers

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

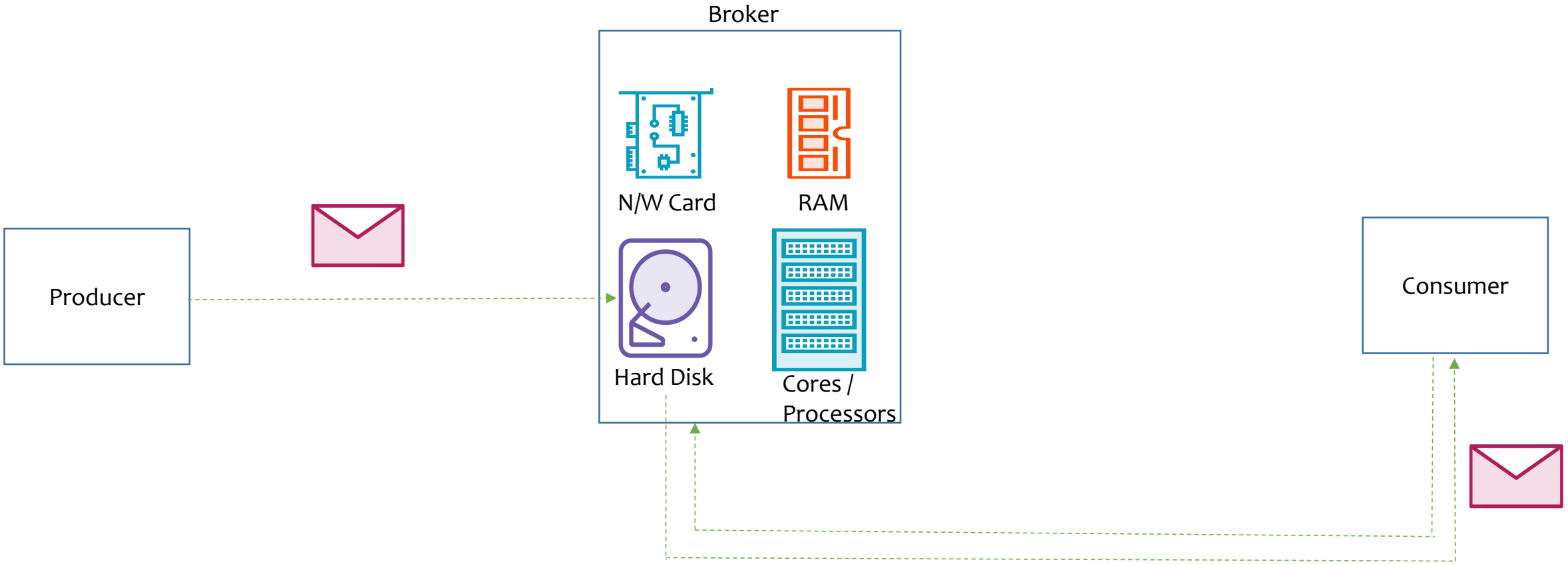


Single Broker / Server



What does a broker do?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

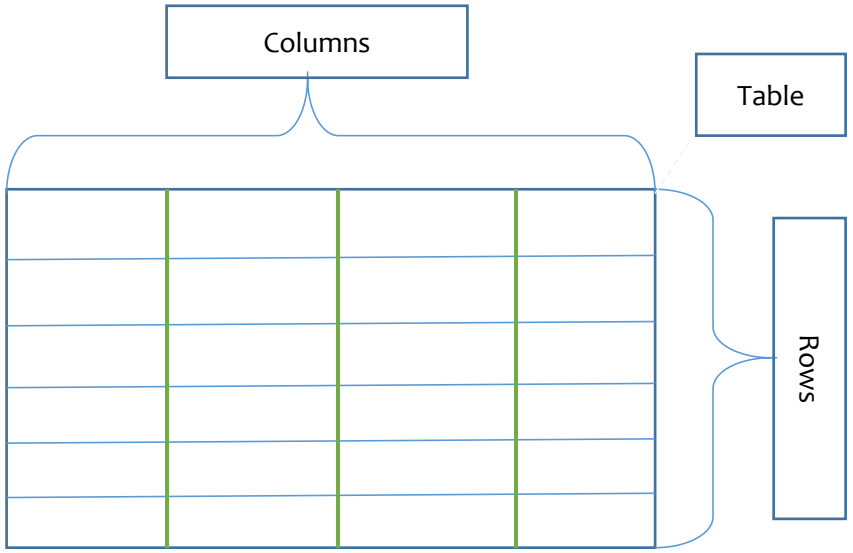


- When producer send the data it will persist that in hard disk.
- When consumer request for data it fetch data from hard disk and send to consumer

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

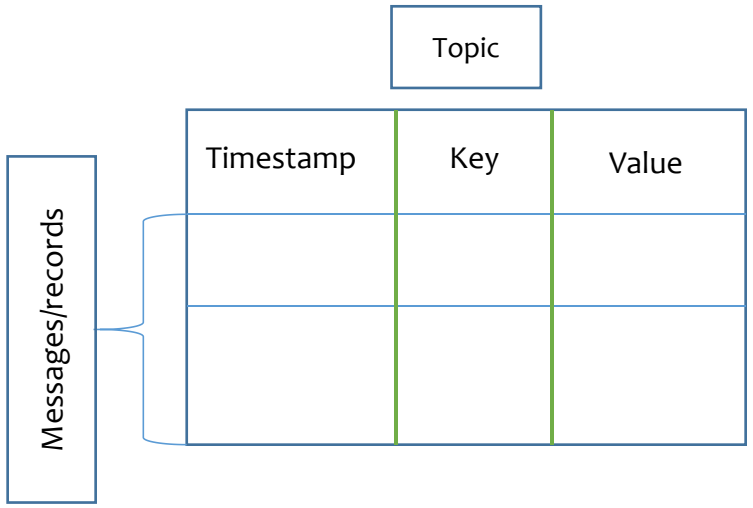
Topic

Before starting with Topic lets resume here and understand how table work in RDBMS?



- To store the data into RDBMS have to create a table
  With columns and there datatypes
- Once the table is create you can insert, update , select, delete data using table name as reference
- One row can be inserted into table at single point of time and that is considered to be a record
- When ever you do a bulk insert each row will be inserted in sequence(one after the other)

Lets understand how topic works?



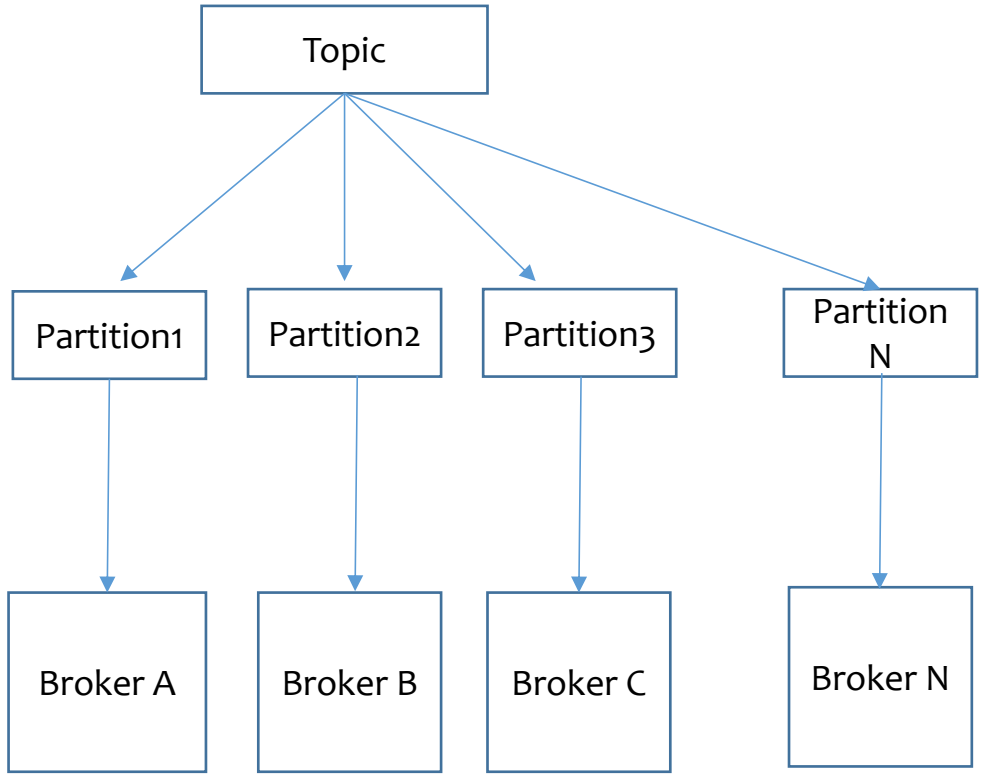
- To store the data into Kafka have to create a Topic
- Every topic in Kafka will have only 3 fields Timestamp, Key, Value
- When every a producer is sending records to cluster it has to include
- Four Things
  - 1)Key
  - 2)Value
  - 3)Topic Name
  - 4)Timestamp: Its Optional if producer add it will be used or else producer will add to record
  - If Consumer wants to get the records it has subscribe to topic by connecting to cluster.
- What is the Datatype of key, value?

  Its Byte what ever the data stores in Kafka topic it will be in the form of bytes.

  We will discuss about this in detail when we talk about producers and consumers

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Kafka topic is divided into partitions and they are distributed across brokers so that cluster will be balanced. We can achieve parallel processing/distributed processing only when we have a distributed storage.

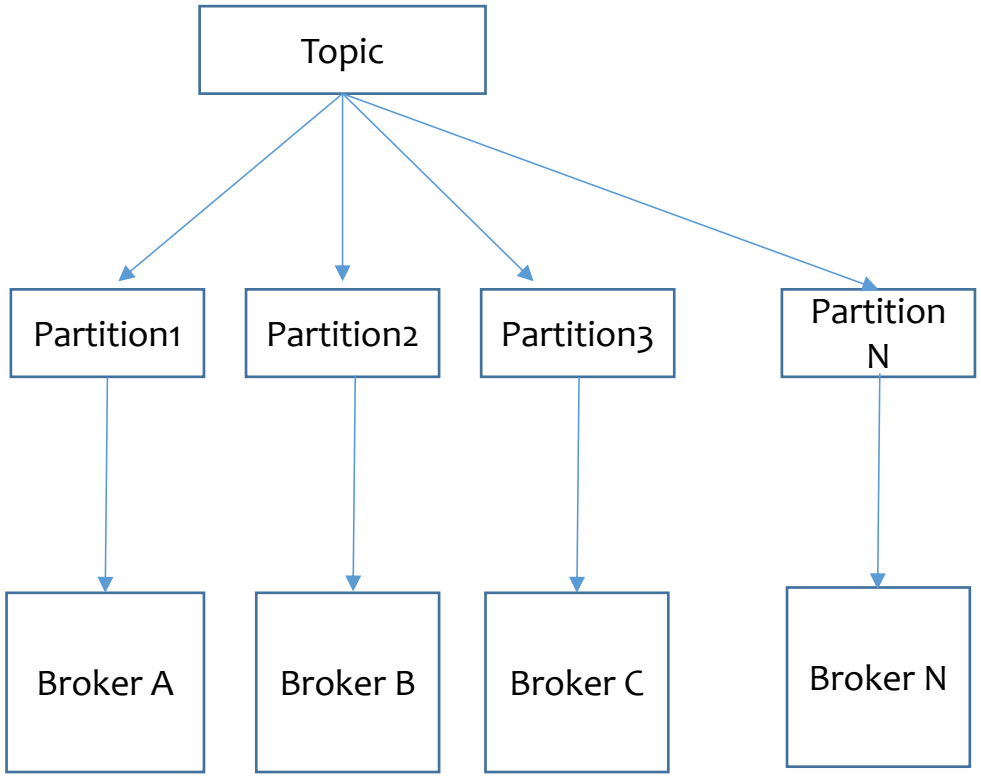


Small question? Lets say cluster contains 3 brokers and user created a topic with 4 partitions. Now how will be partitions distributed across brokers?

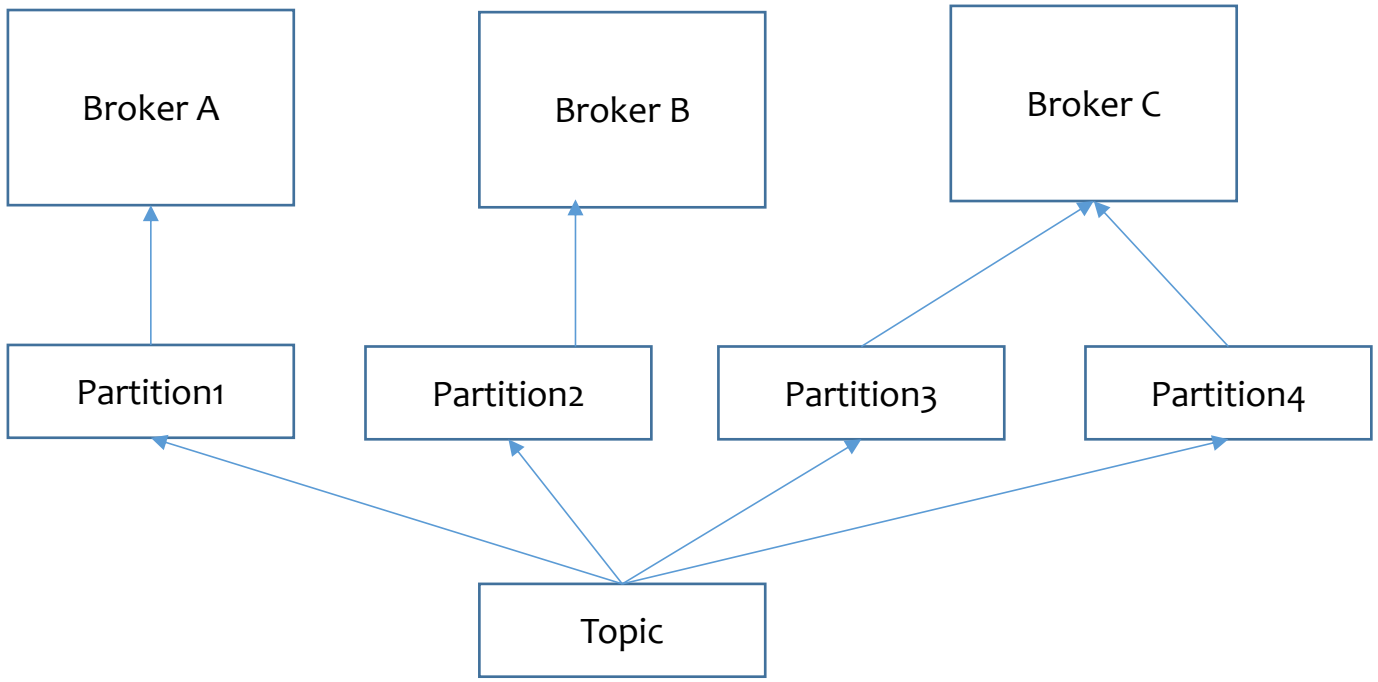
Topic Partitions

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

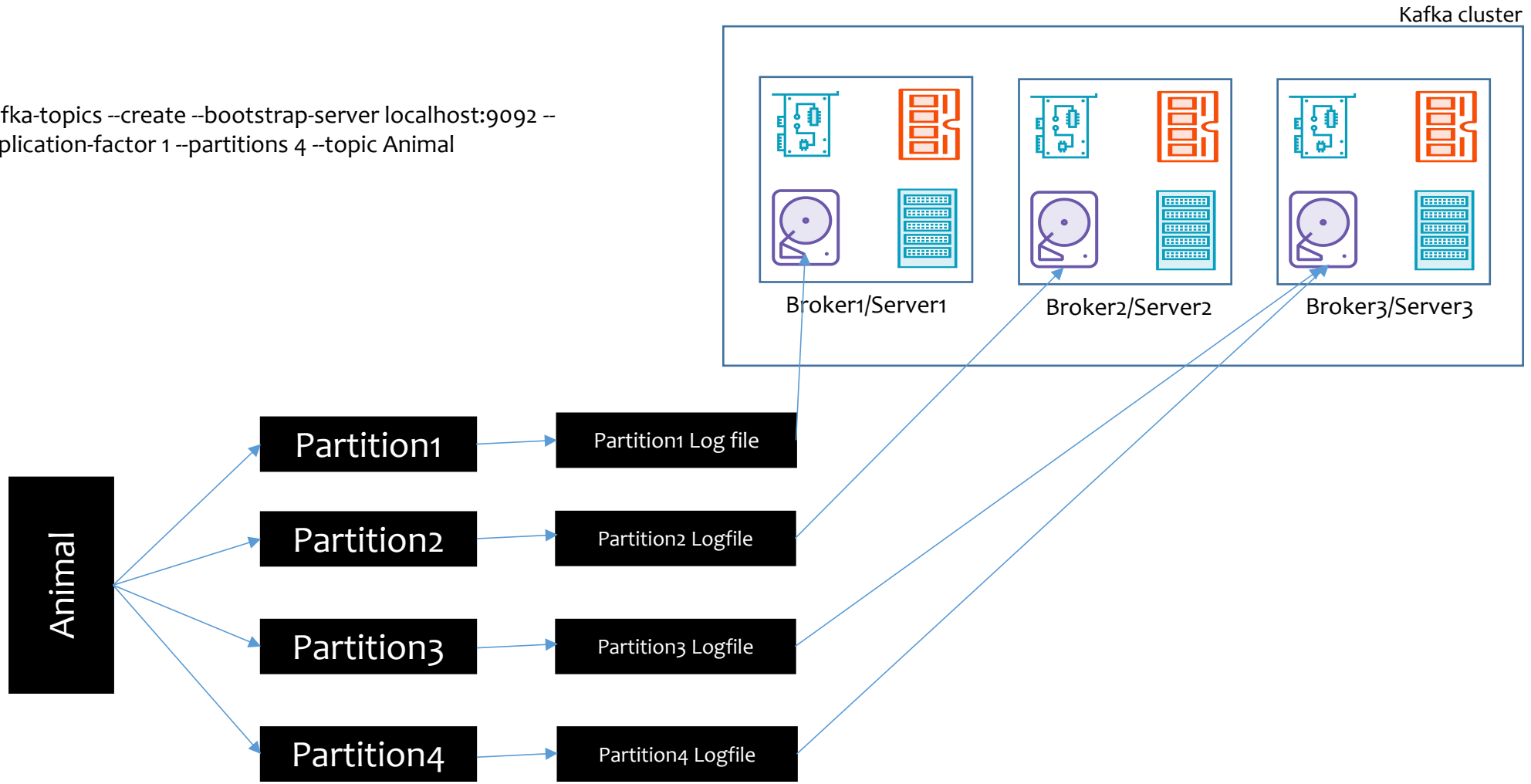
Topic Partitions maintain rebalancing in cluster among broker Kafka topic is divided into partitions and partitions are distributed across brokers. This is a very common thing happens distributed system. We can achieve parallel processing/distributed processing only when we have a distributed storage



Small question? Lets say cluster contains 3 brokers and user created a topic with 4 partitions. Now how will be partitions distributed across brokers?



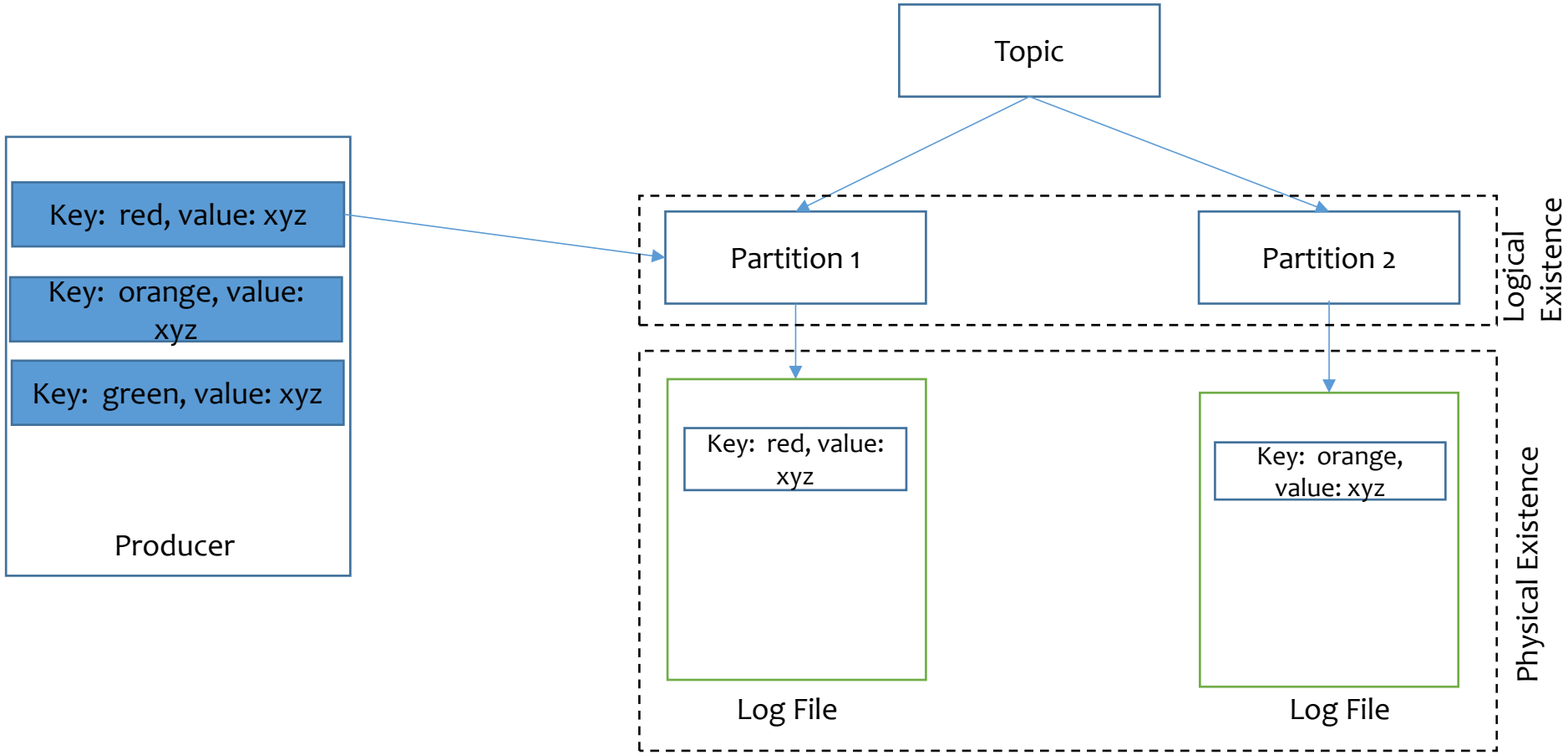
kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 4 --topic Animal



What is Log File in Kafka?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

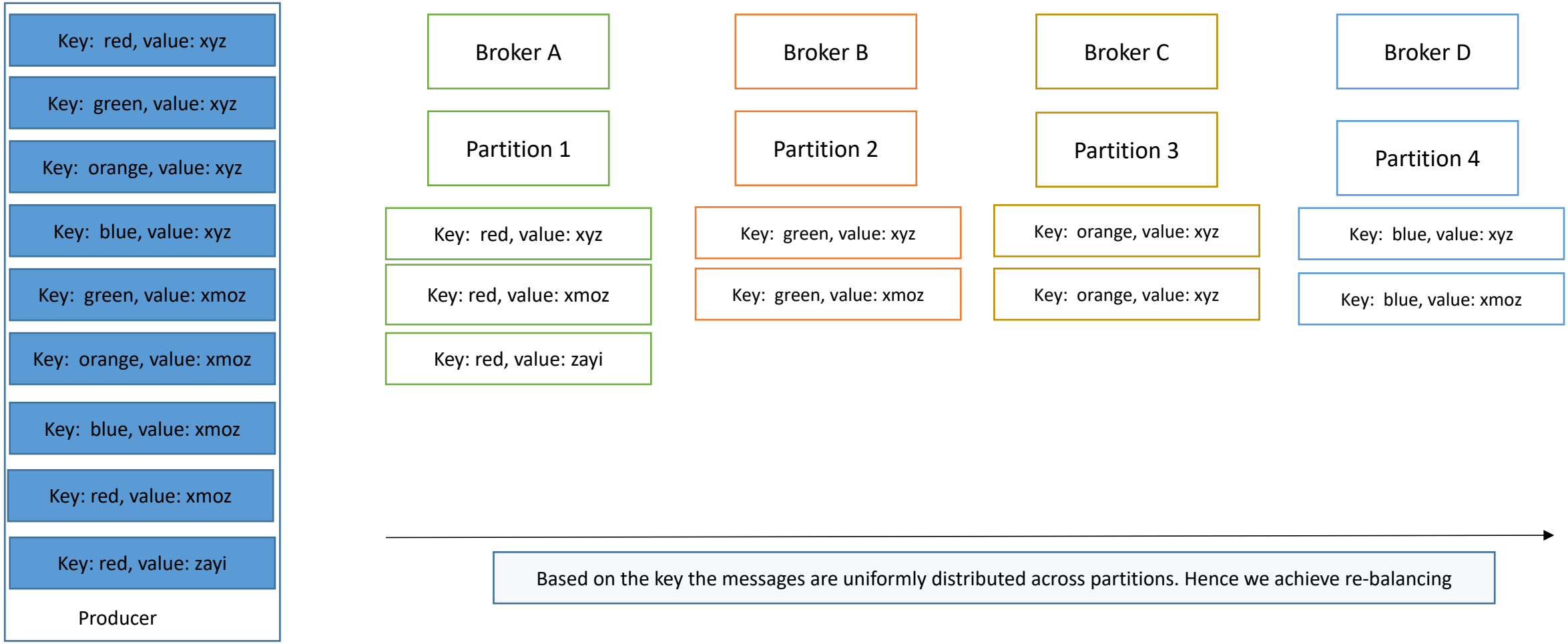
Log File is a place where messages will be stored physically and partition are called logical existence.



How messages distributed across partitions in topic?

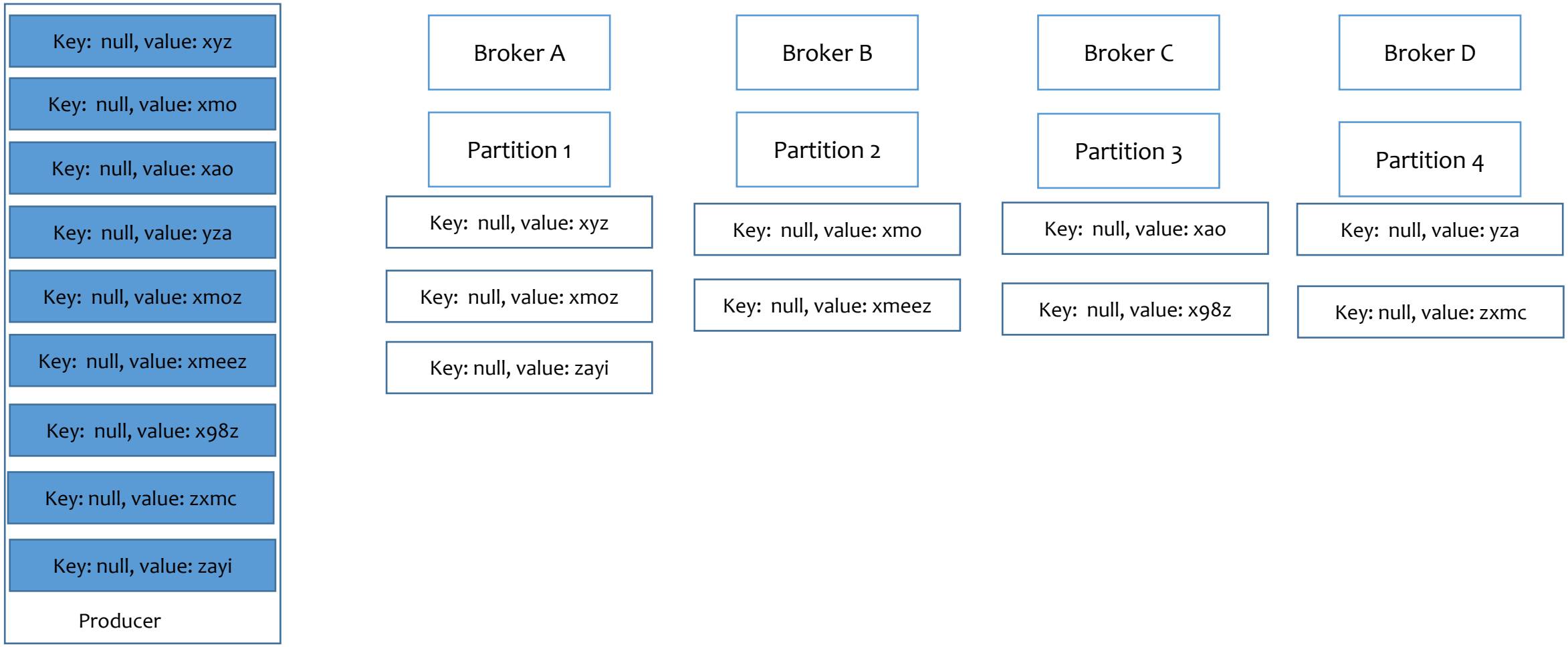
Assume that you have 4 brokers in cluster and user created a topic called colors with 4 partitions key of message will either of the one in
red, green, blue, orange ?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



How partitioning works if key=null?
Then Messages will be distributed in a round robin fashion

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



How will messages get distributed across partitions?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Default Pratitioner is Murmur2 Hash Algorithm.

Murmur2 Algorithm hashes the key and puts the records into a particular partition and using a below formula

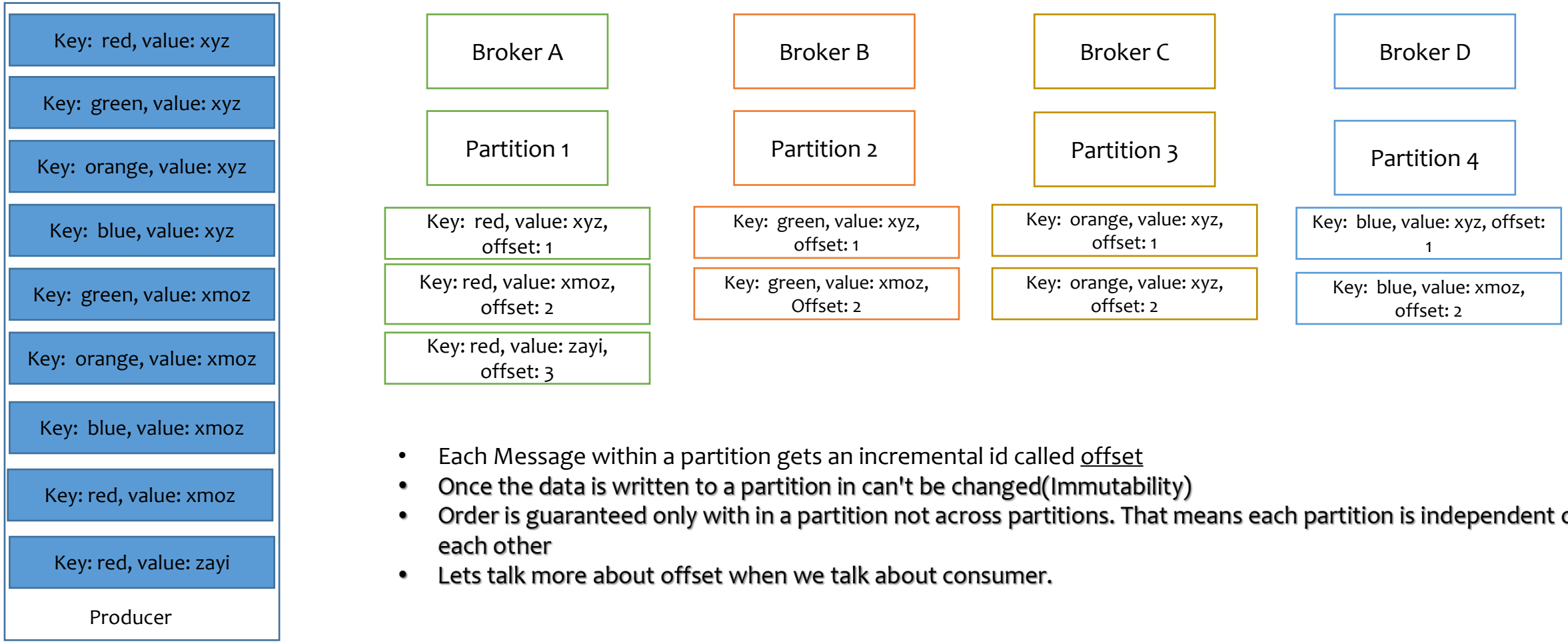
Targetpartition = Utils.abs(utils.murmur2(record.key) % numparitions)

We can change the default behavior by overriding Partitioner class usually we wont do it.

When ever you create a topic you will mention the number of partitions

Offset

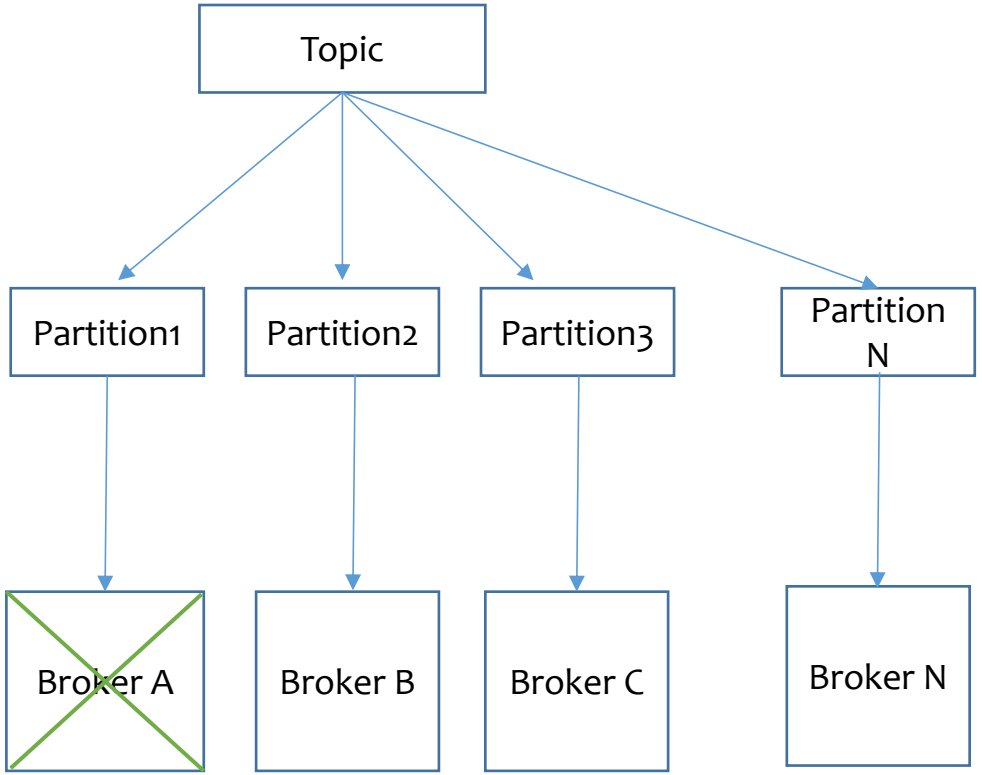
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



- Each Message within a partition gets an incremental id called offset
- Once the data is written to a partition in can't be changed(Immutability)
- Order is guaranteed only with in a partition not across partitions. That means each partition is independent of each other
- Lets talk more about offset when we talk about consumer.

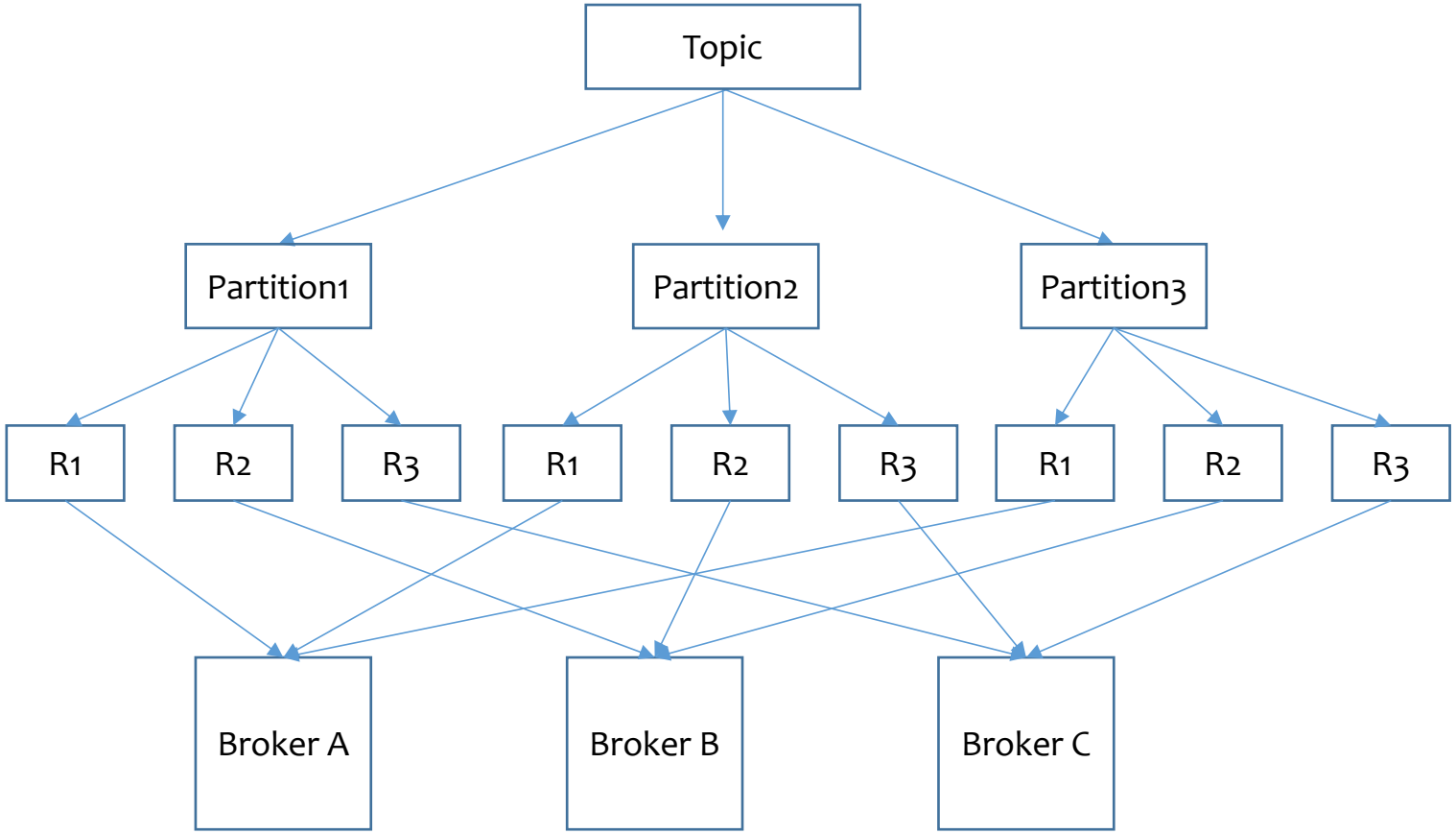
What if Broker goes down?
What happens to partition data?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



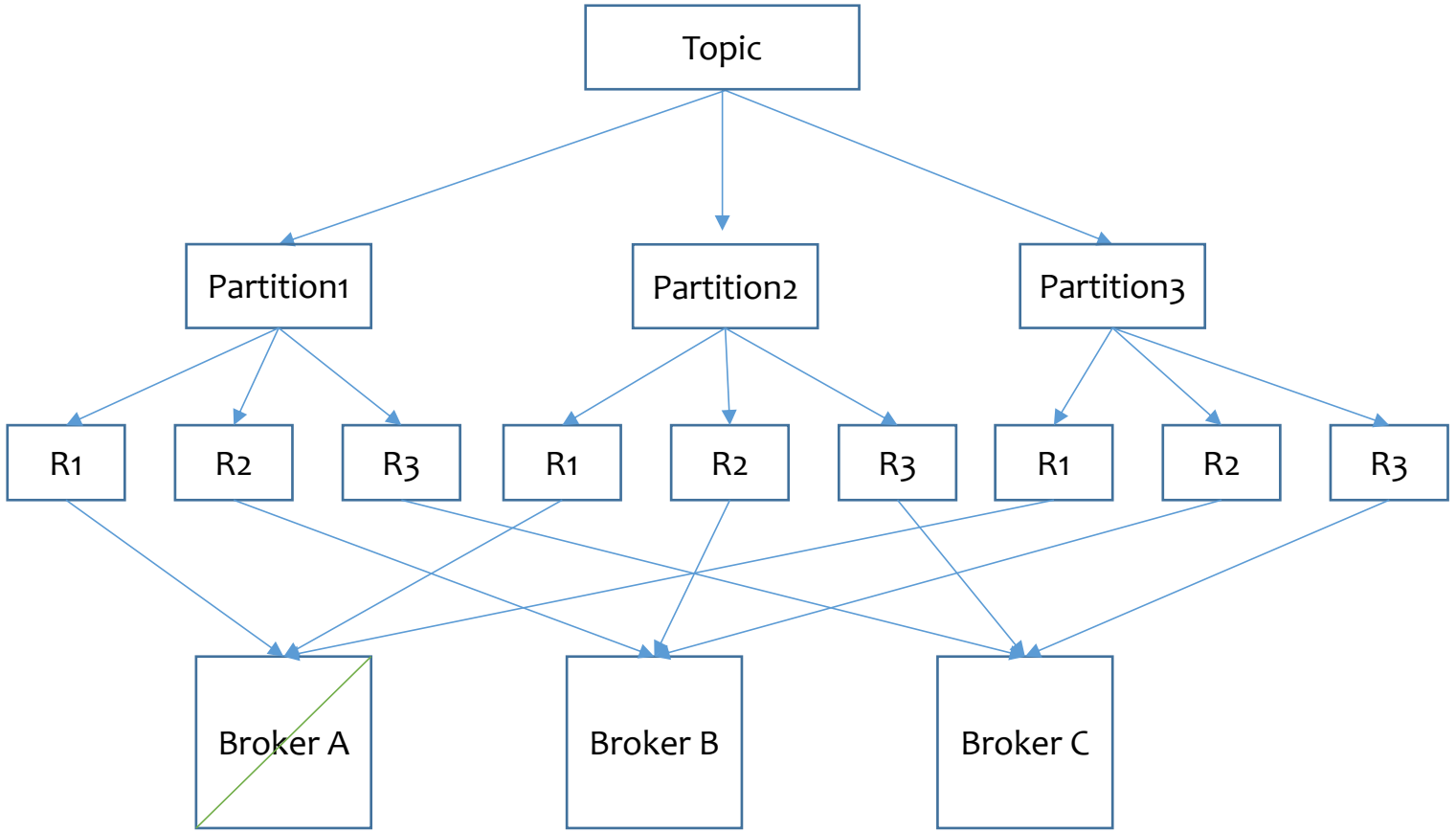
Kafka has a plan to achieve fault tolerance. That is partition replication

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



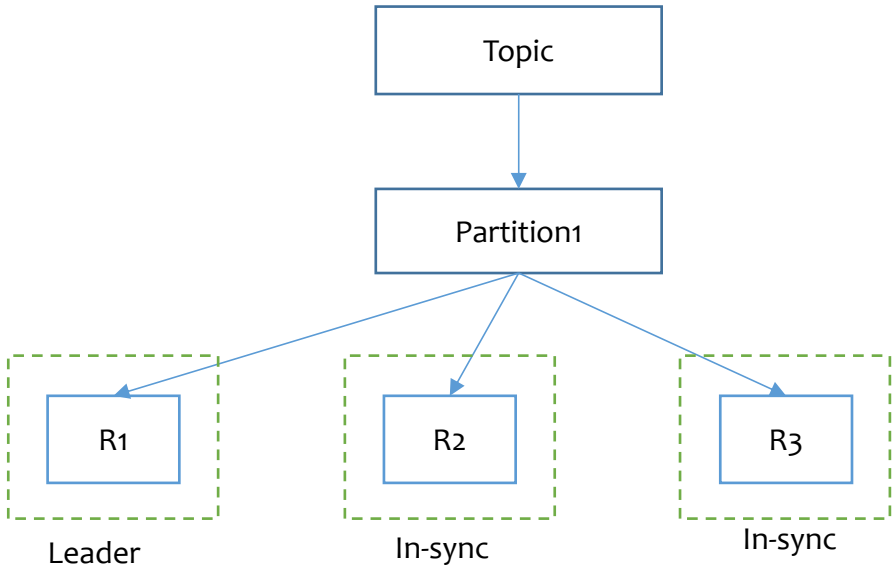
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Now if one broker goes down. The partition replica is available on other broker



Leader and In-sync replicas

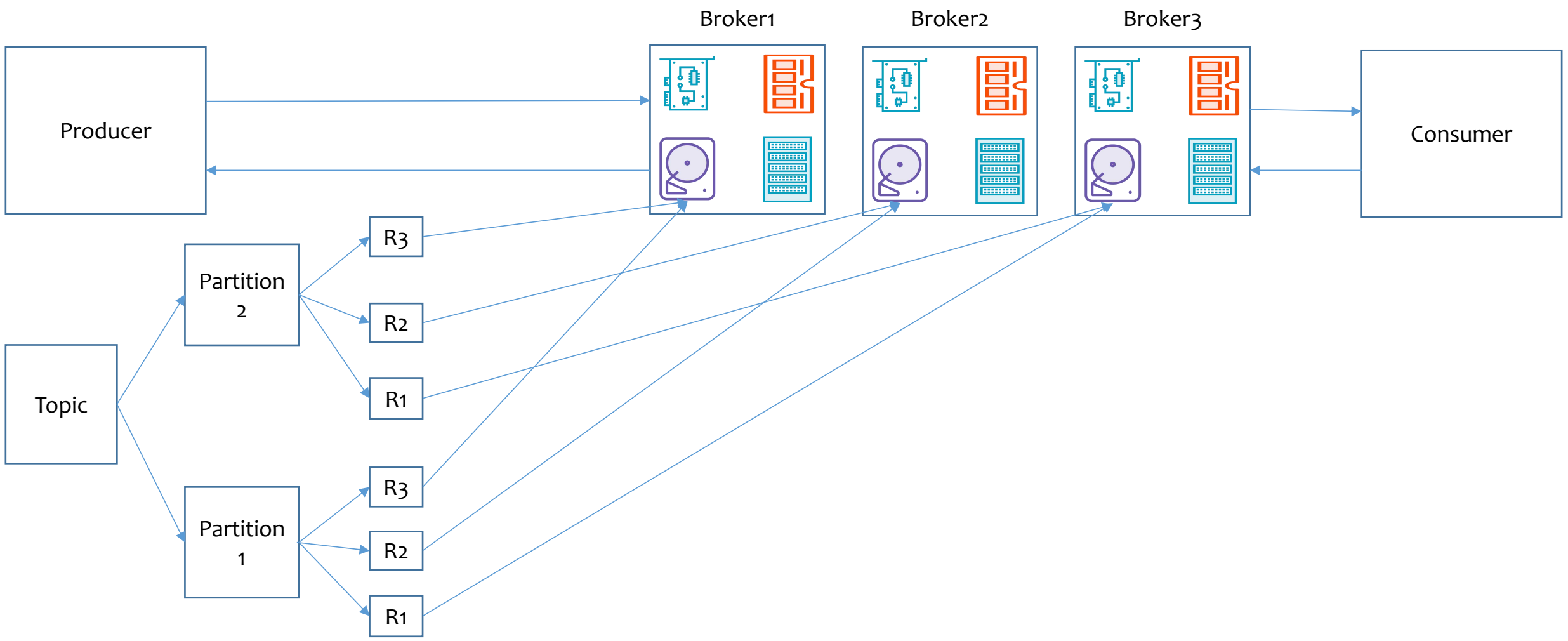
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



- Every topic partition will have replicas
- Out of replicas there will be one Leader and remaining are called In-sync replicas
- Zookeeper will conduct election b/w replicas and choose the leader out of it.
- We will discuss more about this when we talk about Producer Configuration

Bird View

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Partition Count and Replica Count

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

- Its best to get particulars right at the first time at the time of topic creation.
- If partition count increases during a topic lifecycle of topic will break and keys ordering guarantees.
- If replication factor increases during a topic life cycle you put more pressure on your cluster which can lead to a unexpected performance decrease

Partition Count:

- Each Partition can handle a throughput of few MB/s
- More Partitions better performance and better throughput
- Ability to run more consumer groups at scale (This we will see when we talk about consumer and consumer groups)
- But more elections to perform for Zookeeper
- But more Logs file will open(Log files is a place where messages will store in partitions)

Partitions = max(NP, NC)

→NP is the number of required producers determined by calculating: TT/TP
→NC is the number of required consumers determined by calculating: TT/TC
→TT is the total expected throughput for our system
→TP is the max throughput of a single producer to a single partition
→TC is the max throughput of a single consumer from a single partition

For example, if you want to be able to read 1 GB/sec, but your consumer is only able process 50 MB/sec, then you need at least 20 partitions and 20 consumers in the consumer group. Similarly, if you want to achieve the same for producers, and 1 producer can only write at 100 MB/sec, you need 10 partitions. In this case, if you have 20 partitions, you can maintain 1 GB/sec for producing and consuming messages. You should adjust the exact number of partitions to number of consumers or producers, so that each consumer and producer achieve their target throughput.

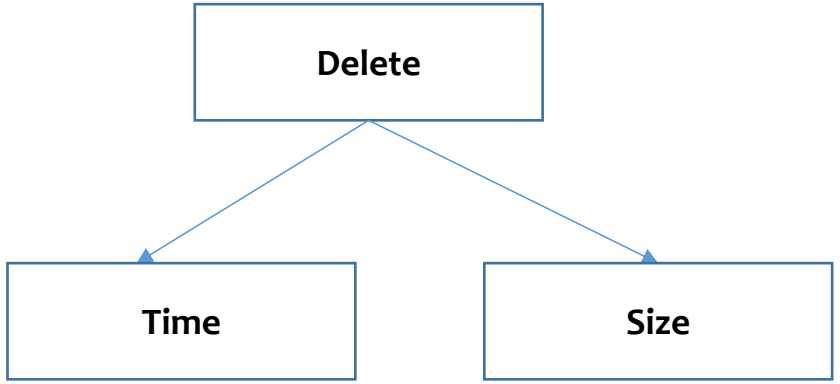
Note: Partitions should be not more than 2000 to 4000 for broker and 20,000 per cluster because if broker goes down zookeeper has to perform lots of leader elections.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

- Replication Factor should be Atleast:2, Usually 3, Maximum: 4
- Better the resilience of your system N-1 brokers can fail
- In case of more replications(higher the latency if acks = all(we will discuss in details when we talk about producer).

How long data will reside in Topic?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Time:

- By Default broker is configured to delete the messages in 7 days.
- The property to set this property is log.retention.hours
- Lets say if you set your retention period to 1 day the message produced on day 1 will be deleted on day2

Size:

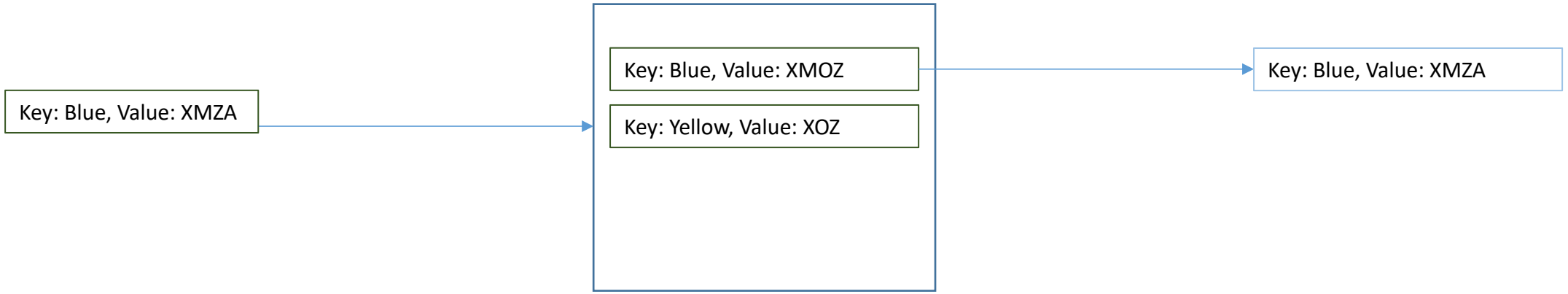
- The broker starts cleaning up the messages based on the space.
- Lets say maximum size for topic is set to 20KB and lets say each message has 5 kb so in your topic we can store max 4 messages. now lets say when 5 message arrives the system the old ones are deleted.
- By default no value will be set in configuration .
- The Property to set size is log.retention.bytes

How long data will reside in Topic?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Compaction

Compaction in Kafka works as Upsert(**Update + Insert**).That means when a new message is produced by the producer to broker then broker check whether record with key exists or not if exist it updates the value and if it is not will insert the value



What is Zookeeper?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Zookeeper provides multiple features for distributed applications:

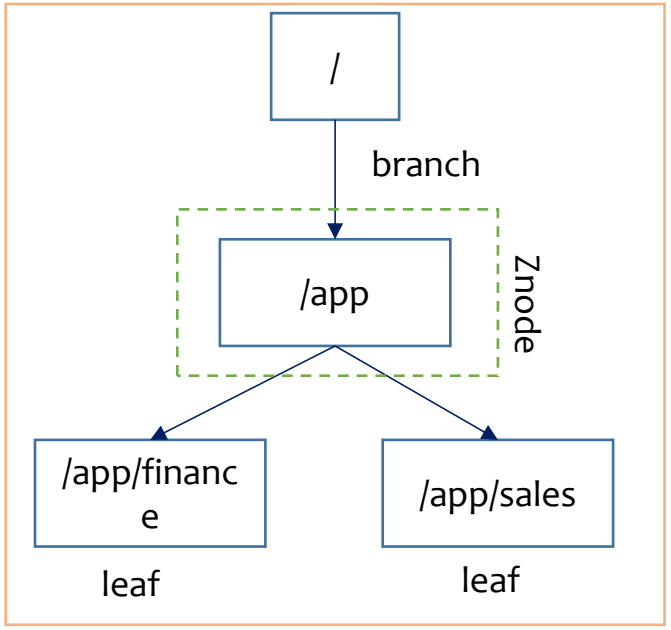
- Distributed configuration management
- Self election
- Coordination and locks(low level)
- Key value store

- Zookeeper used in many distributed systems such as Hadoop, Kafka, Hbase etc.
- Its an apache project that's proven to be very stable and hasn't had a major release in many years
- 3.4.x stable version
- 3.5.x is in development for many years, and its still beta(not for production use)

What is Zookeeper?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

We all know how Linux file system looks like its starts from / folder then extended by different directories
Example /home/ec2-user/
zookeeper also looks in the same way

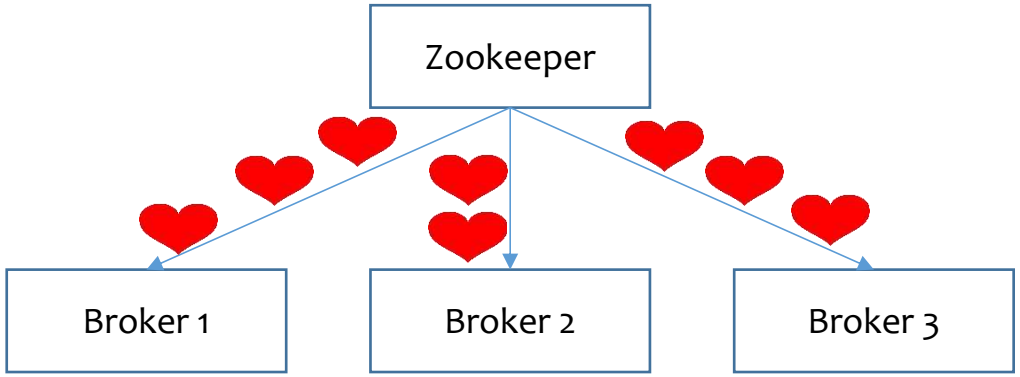


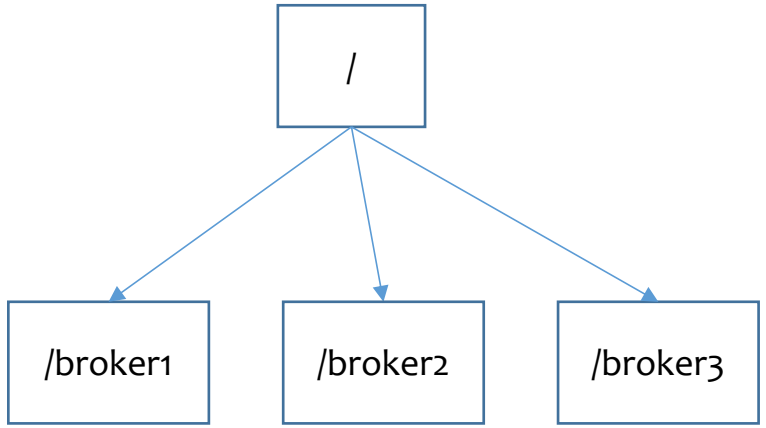
Tree

- Zookeeper Internal data structure structure is like Tree. It has leafs and branch's.
- Each node is called a zNode.
- Each node has a path
- Each node can be persistence or ephemeral. what the difference?
- persistence zNode will alive all the time.
- ephemeral zNode go away if your app disconnect
- Each Znode can store multiple zNode or it can store data.
- We cannot rename zNode.
- One of the best feature of zookeeper is it Watched for Changes. say if any change is occurred in /app/finance it will let me know hey hey there is some change in /app/finance check it out.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Broker Registration, with heart beat mechanism to keep the list of current.
When broker registration happens a zNode will be created.





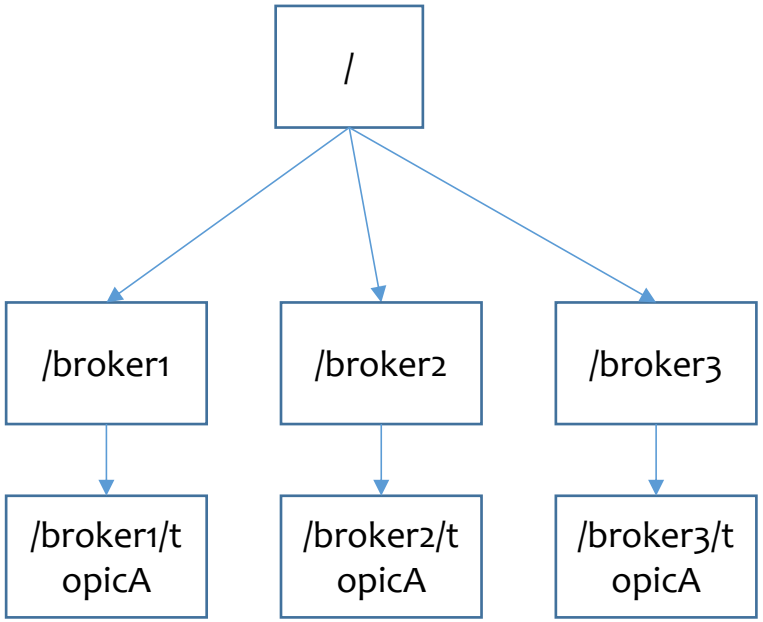
Role of Zookeeper?

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Maintaining the list of topic alongside(when ever a topic is created it create a zNode in zookeeper all the info for topic will be stored there)

- Their configurations(Partitions, replication factor, additional configurations)
- The list of ISR(Insync replicas) for partitions

Performing leader elections in case of broker goes down



Role of Zookeeper?

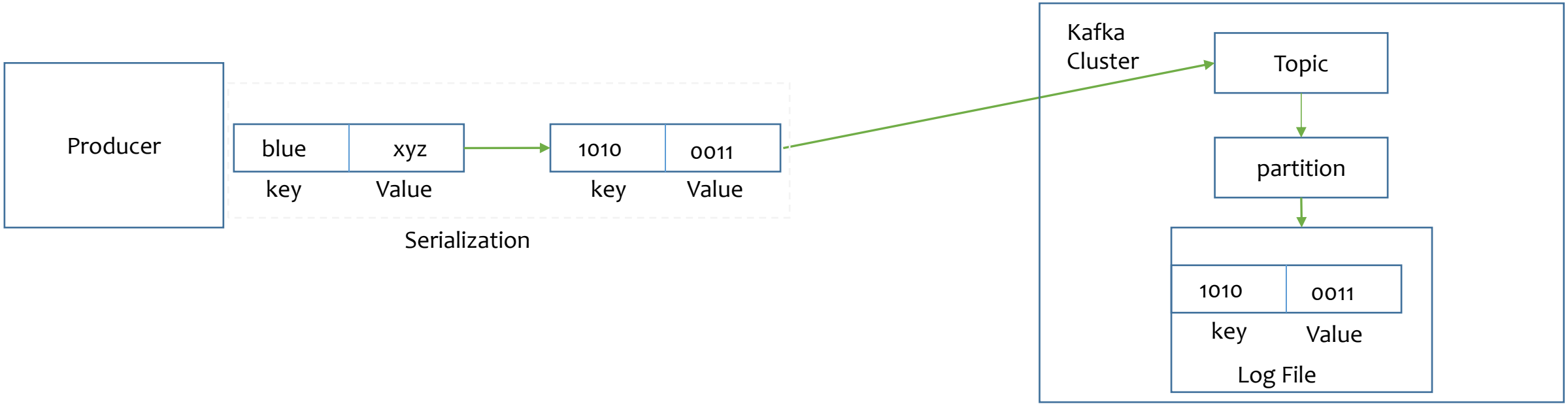
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

- Storing the Kafka Cluster ID(randomly generated at 1st startup of cluster)
- Storing ACL's(Access control list) if security is enabled
- Topics
- Consumer groups
- user
- (Depreciated: Not in use) Used by old consumer API to store offsets.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

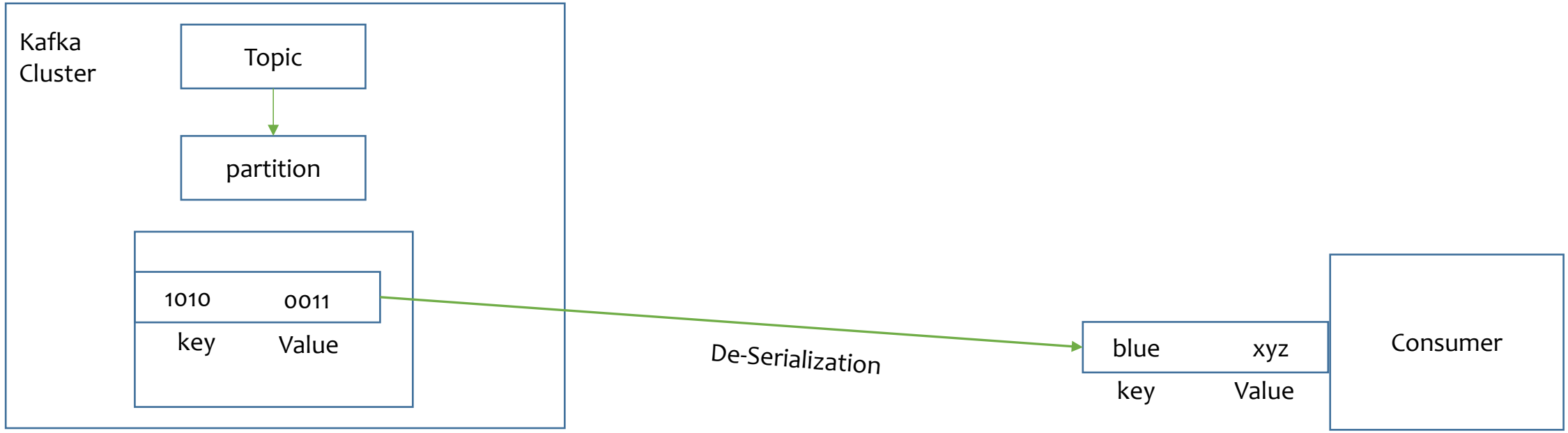
The process of transforming object to byte is called serialization. Kafka cluster/topic can store only bytes so when producer is sending the messages to topic messages(Key, Value) has to be serialized or converted to bytes. Conversion process will happen at producer end using serializes

Default serializers provided by Kafka are String, Long, Int. for custom object serialization we have to depend on AVRO Serialization we will talk in detail about this.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The process of transforming byte to object is called de-serialization. When Consumer connects to the Kafka and subscribe for a topic then Kafka send messages in bytes which has to be de-serialized back to message at consumer for further processing.



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

awks:

0: Possibility of data loss is very high no acknowledgement from Leader or In-sync replicas.

1: Possibility of data loss is moderate Leader will send the acknowledgement to producer once the messages is received.

All: Possibility of data loss is very less because both leader and In-sync replicas has to acknowledgement to producer.

min.insync.replicas:

This can be set either in cluster level(applicable to all topics) or topic level

If this property is set to 2 and awks = All then at any point of time min brokers has to be available = 2 or else it will throw an exception

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

retries:

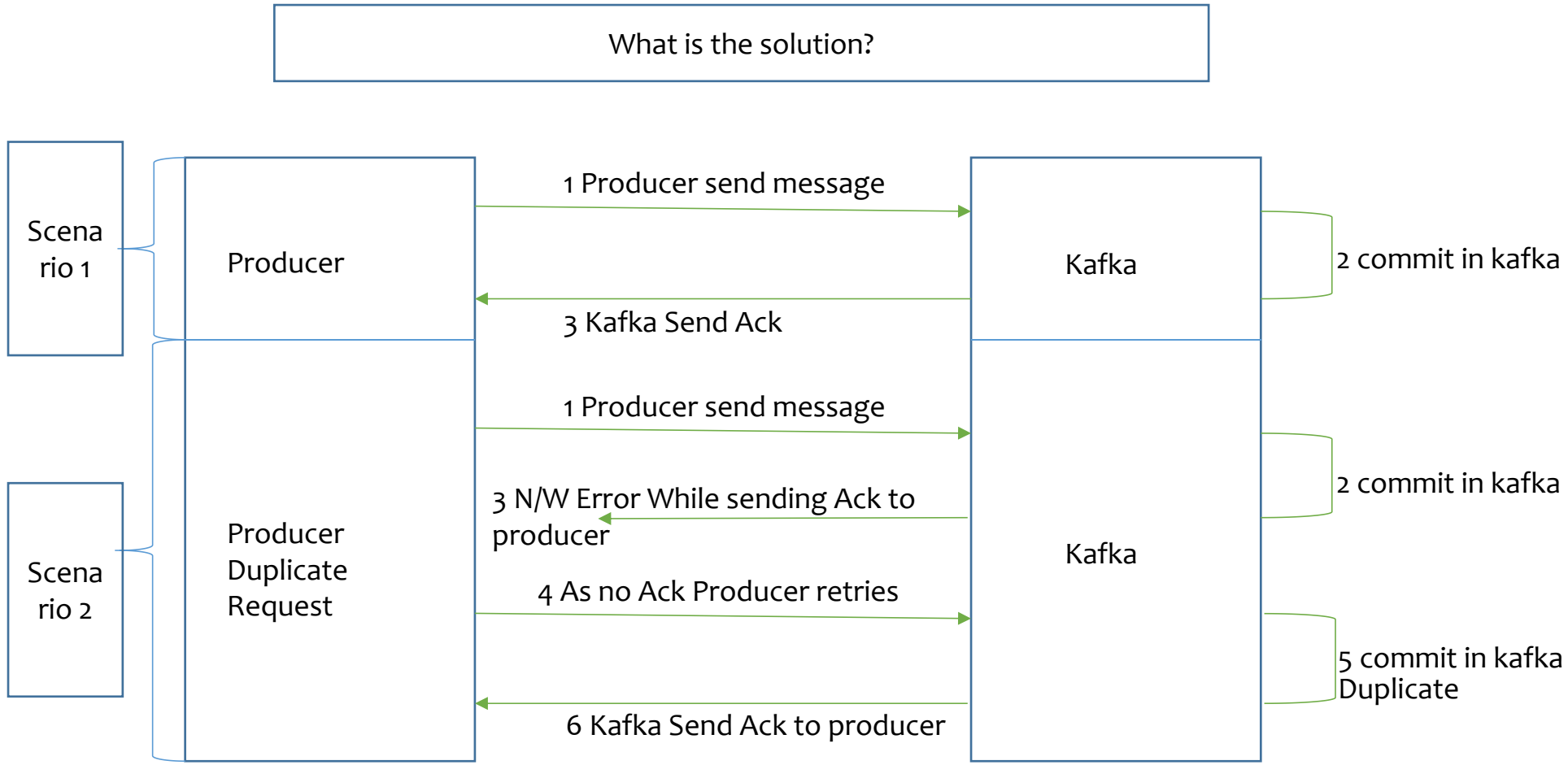
In case of N/W or Hardware failures the developer has to handle the exceptions otherwise there will loss of data. if we set retries property producer will be keep retrying until cluster comes up.

By default this property is set to 0 for zero data loss set it to Interger.Max_Value

max.in.flight.request.per.connection:

In case of more retries there is a possibility of messages out of order that means messages will not send in a proper order one after another. for that reason if messages has to go in a proper order have to set this property. Set this property to 5 for proper ordering and high performance.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



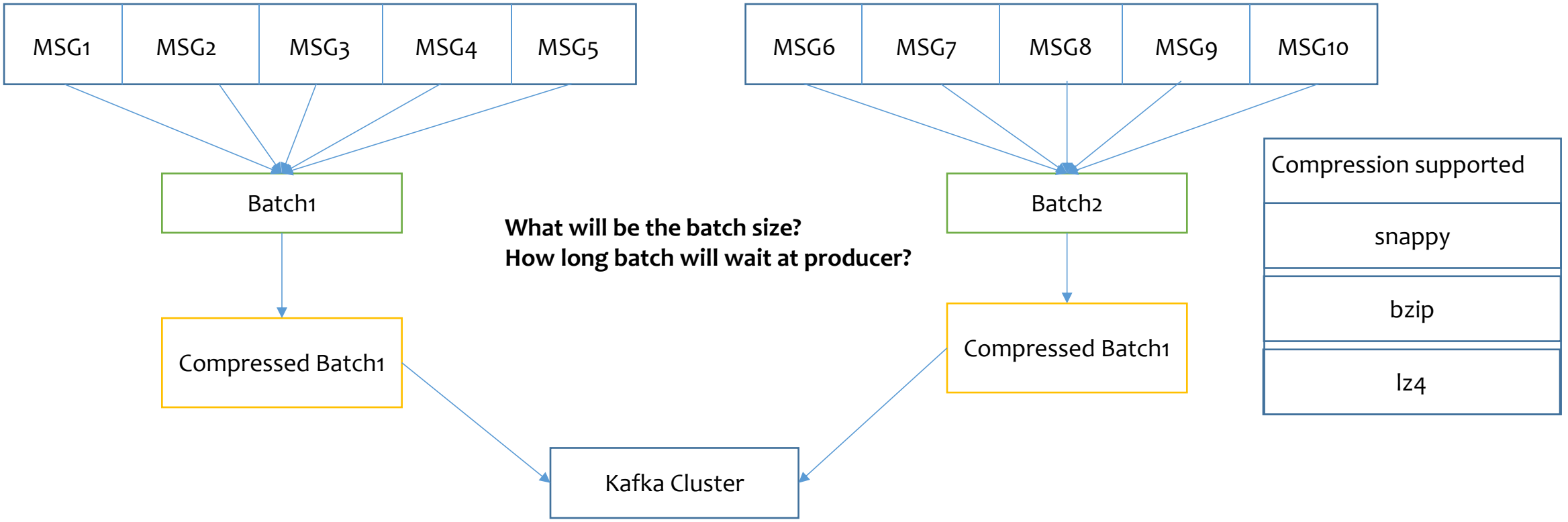
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Solution: Idempotent Producer
Property : enable.idompatance = true

Now if producer is Idempotent there will no chance of duplicate commits because when there is a retry producer request it will append request ID to message it checks whether it is already committed or not with that id if it is already committed it will not commit again.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Compressing the batch of messages is one of the optimization used in kafka to Increase the Throughput.
Property : **compression= snappy**



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Batch.size:

Max number of bytes that will included in batch. The default is 16KB.

Increasing batch size to 32Kb or 64 Kb can help increasing the compression, throughput and efficiency of requests.

Linger.ms:

By default kafka tries to minimize the latency that means as soon as the message is received the kafka sends the message to cluster.

To Change this behavior and make producer wait for a while to form a batch linger.ms is used to increase the throughput while maintaining the low latency.

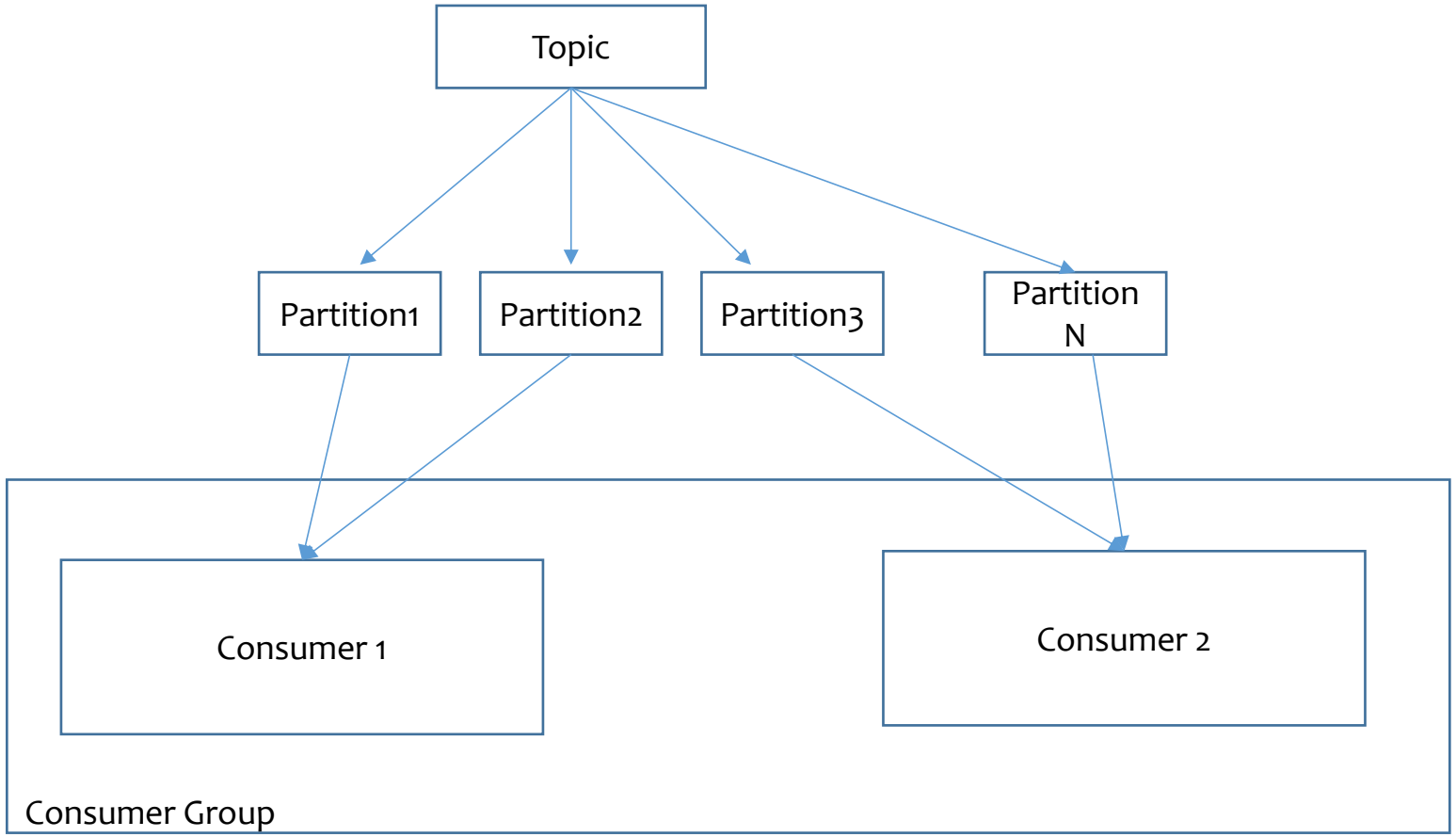
Linger.ms = Number of milliseconds a producer is going to wait to send the batch by default its set to 0.

Introducing the little delay will increases the throughput, compression and efficiency of a producer.

If the batch is full before the end of linger.ms period it will send kafka right away.

Consumer Configuration: Consumer Group

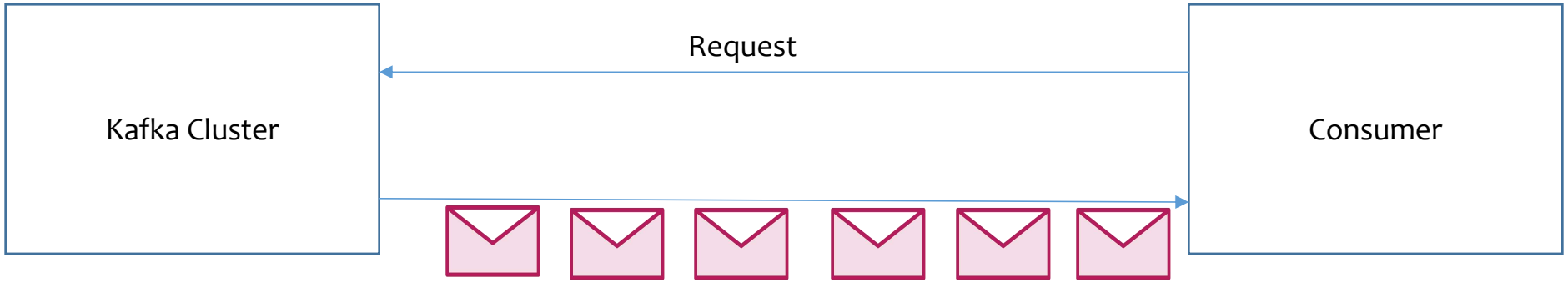
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Consumer Configuration: Poll

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Poll is used to get messages from the kafka. lets say if it is set to 100ms at every 100ms of time consumer will request the kafka for messages that is called fetch. If no messages are available then it returns null.



You can control the poll data:

1) fetch.min.bytes: how much data you want to pull at least on each request. Default 1 MB
2) fetch.max.bytes: Max data returned for each fetch request. Default 500 Mb
3) Max.partition.fetch.bytes: Max data returned by broker per partition. Default 1MB
4) Max.poll.records: how man records to receive per poll request. Default 500

Consumer Delivery Schematics

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Kafka stores offsets at which a consumer group has been reading. these offsets are committed live in kafka topic name _consumer_offsets. In case of consumer dies it will be able to read back from where it left off thanks to committed consumer offsets. Offset commit depends on the schematics that you are choosing.

Atmost Once: Offsets are committed as soon as messages is received, If processing goes wrong the message will be lost it wont read again. It is not preferred.

Atleast Once: Offsets are committed only if message is processed at consumer side. If processing goes wrong the messages will read again there is a chance of duplication.so we have to make consumer idempotent. It is usually preferred.

Exactly Once : This can be achieved by kafka work flow using stream API's. Even in case of any failures record will be processed only once. No chance of duplication here.

Consumer Configuration: Offset commit

enable.auto.commit:

If this property is set to true the moment the message is processed offset will be committed. By this we can achieve Atleast Once behavior.

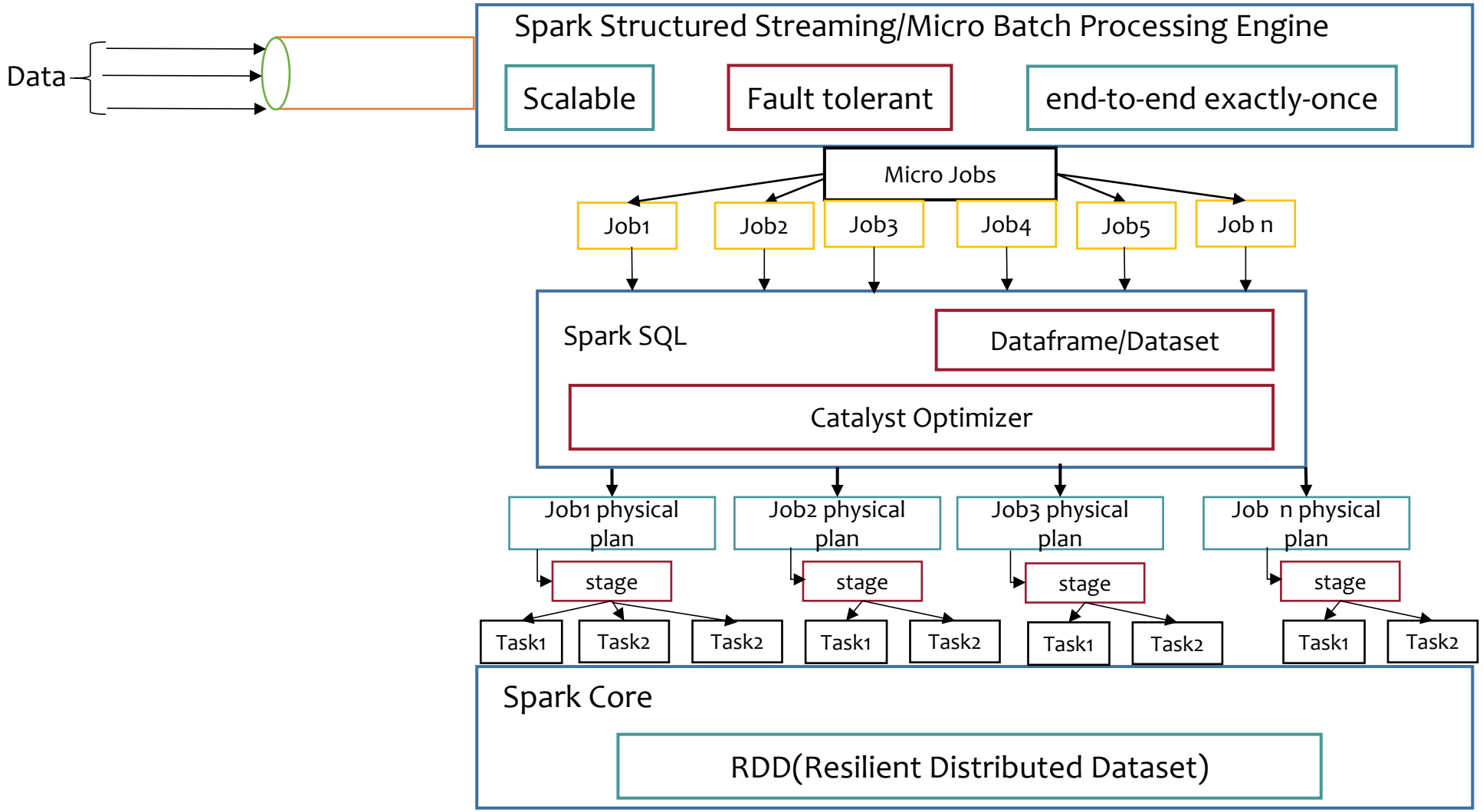
If it is set to false It means manually user has to commit the offset using sync() method which is not recommended in production

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Spark Structured Streaming

Spark Structured Streaming Engine High level Architecture

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

→You can express your streaming computation the same way you would express a batch computation on static data. The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive.

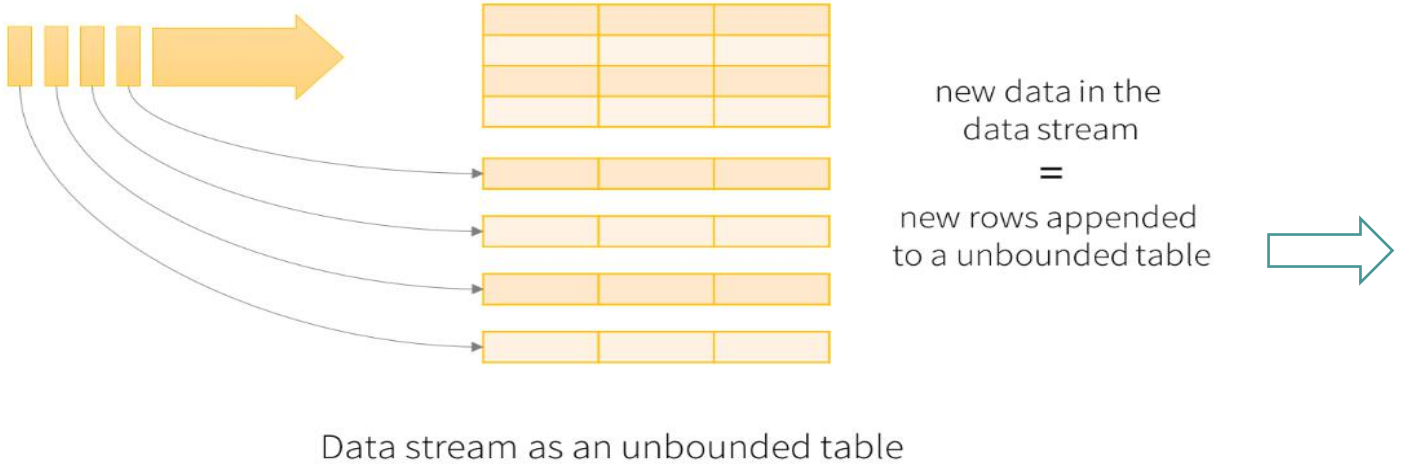
→The computation is executed on the same optimized Spark SQL engine.

→Internally, by default, Structured Streaming queries are processed using a **micro-batch processing engine**, which processes data streams as a series of **small batch** jobs thereby achieving end-to-end latencies as low as **100 milliseconds** and **exactly-once fault-tolerance** guarantees.

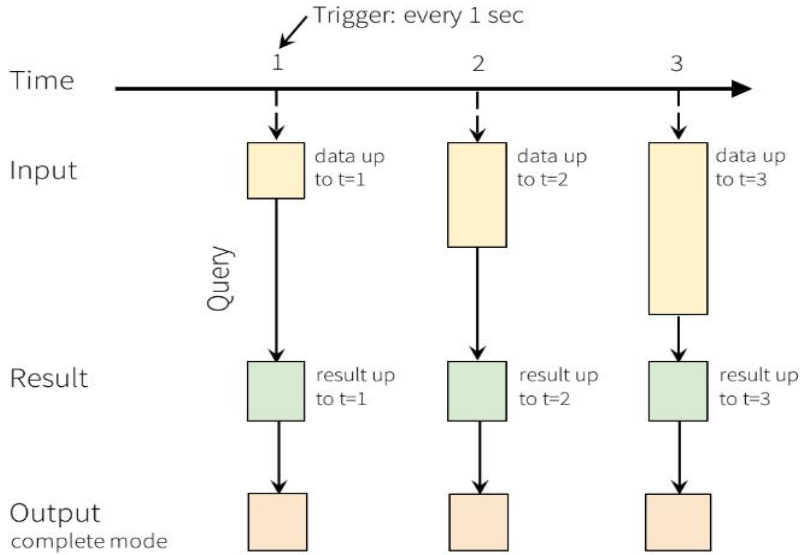
→Since Spark 2.3, we have introduced a new low-latency processing mode called **Continuous Processing**, which can achieve end-to-end latencies as low as 1 millisecond with at-least-once guarantees.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The key idea of structured streaming is to treat a live stream as table that is being continuously appended. Consider the input data stream as the "Input Table". Every data item that is arriving on the stream is like a new row being appended to the Input Table.



Data stream as an unbounded table



Programming Model for Structured Streaming

A query on the input will generate the "Result Table". Every trigger interval (say, every 1 second), new rows get appended to the Input Table, which eventually updates the Result Table. Whenever the result table gets updated, we would want to write the changed result rows to an external sink.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Source Name	Description
Complete Mode	The entire updated Result Table will be written to the external storage. It is up to the storage connector to decide how to handle writing of the entire table. Usually this mode will be used when aggregations are performed
Append Mode	Only the new rows appended in the Result Table since the last trigger will be written to the external storage. This is applicable only on the queries where existing rows in the Result Table are not expected to change.
Update Mode	Only the rows that were updated in the Result Table since the last trigger will be written to the external storage. Note that this is different from the Complete Mode in that this mode only outputs the rows that have changed since the last trigger. If the query doesn't contain aggregations, it will be equivalent to Append mode

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Trigger Name	Description
unspecified (default)	If no trigger setting is explicitly specified, then by default, the query will be executed in micro-batch mode, where micro-batches will be generated as soon as the previous micro-batch has completed processing.
Fixed interval micro-batches	The query will be executed with micro-batches mode, where micro-batches will be kicked off at the user-specified intervals. →If the previous micro-batch completes within the interval, then the engine will wait until the interval is over before kicking off the next micro-batch. →If the previous micro-batch takes longer than the interval to complete (i.e. if an interval boundary is missed), then the next micro-batch will start as soon as the previous one completes (i.e., it will not wait for the next interval boundary). →If no new data is available, then no micro-batch will be kicked off.
One-time micro-batch	The query will execute *only one* micro-batch to process all the available data and then stop on its own. This is useful in scenarios you want to periodically spin up a cluster, process everything that is available since the last period, and then shutdown the cluster. In some case, this may lead to significant cost savings.
Continuous with fixed checkpoint interval (experimental)	The query will be executed in the new low-latency, continuous processing mode

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Source Name	Description
File Source	Reads files written in a directory as a stream of data. Supported file formats are text, csv, json, orc, parquet. By implementing DataStreamReader interface you can your different file formats
Kafka Source	Reads data from Kafka. It's compatible with Kafka broker versions 0.10.0 or higher
Socket source (for testing)	Reads UTF8 text data from a socket connection. The listening server socket is at the driver. Note that this should be used only for testing as this does not provide end-to-end fault-tolerance guarantees.
Rate source (for testing)	Generates data at the specified number of rows per second, each output row contains a timestamp and value. Where timestamp is a Timestamp type containing the time of message dispatch, and value is of Long type containing the message count, starting from 0 as the first row. This source is intended for testing and benchmarking.

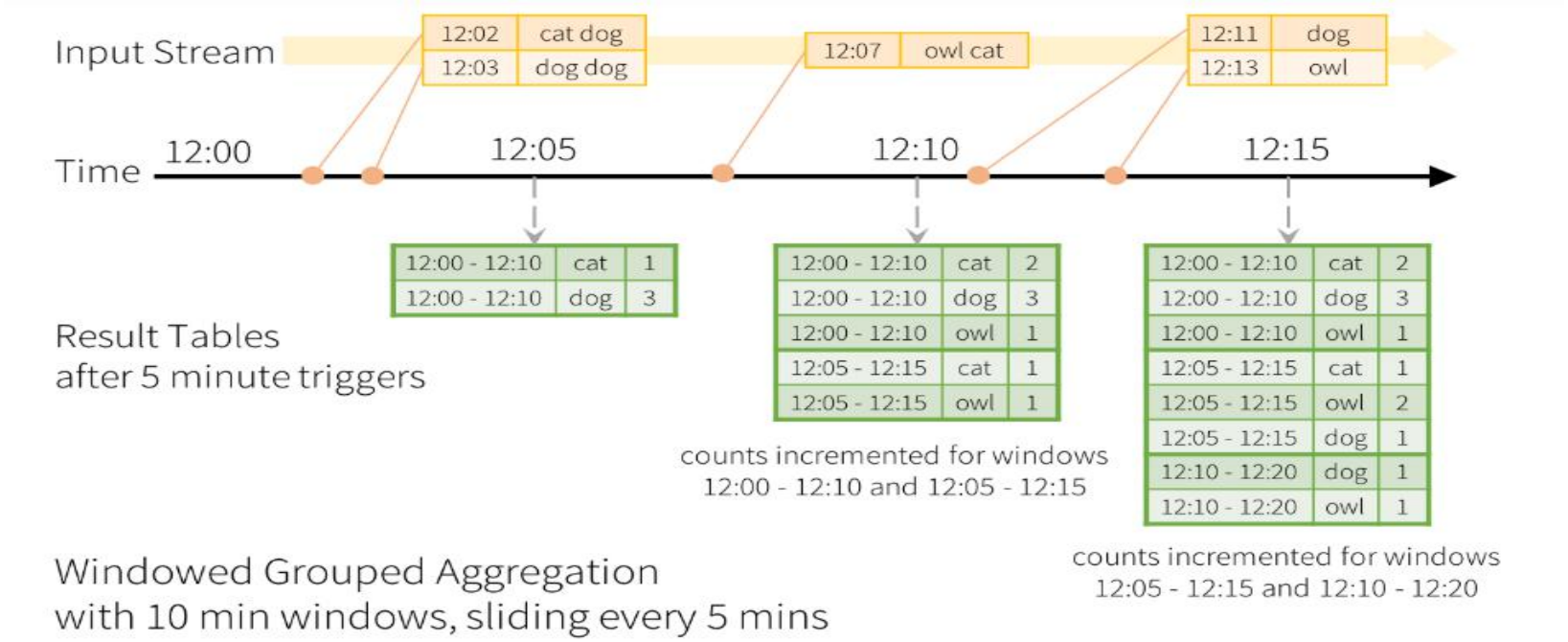
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Source Name	Options	
File Source	→path: path to the input directory, and common to all file formats. →maxFilesPerTrigger: maximum number of new files to be considered in every trigger (default: no max) →latestFirst: whether to process the latest new files first, useful when there is a large backlog of files (default: false)	
Socket Source	host: host to connect to, must be specified port: port to connect to, must be specified	

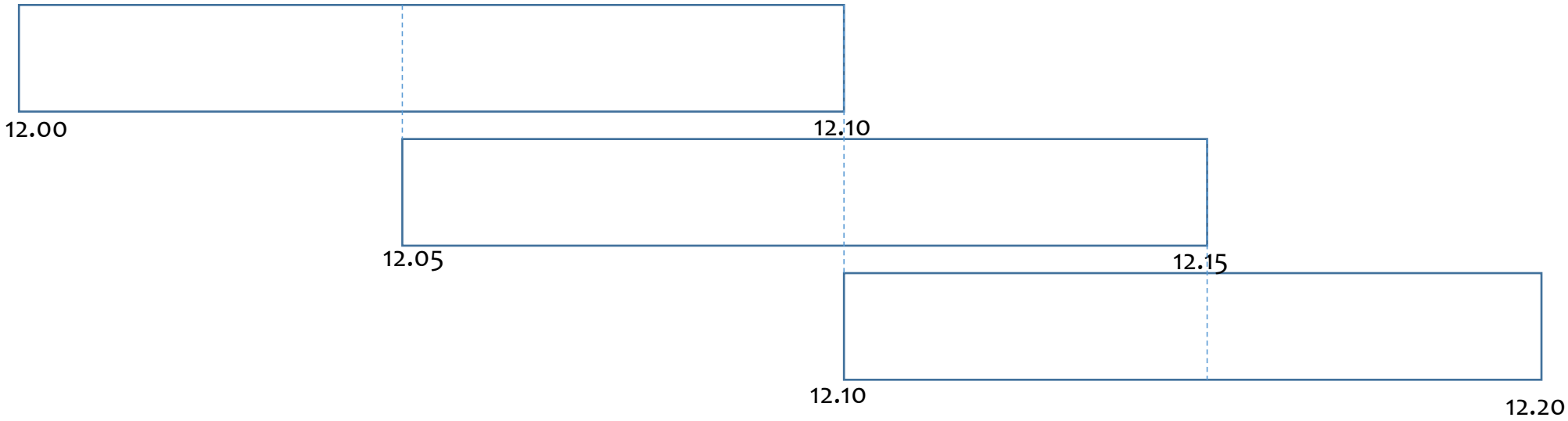
Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

Aggregations over a sliding event-time window are straightforward with Structured Streaming and are very similar to grouped aggregations. In a grouped aggregation, aggregate values (e.g. counts) are maintained for each unique value in the user-specified grouping column. In case of window-based aggregations, aggregate values are maintained for each window the event-time of a row falls into. Let's understand this with an illustration.

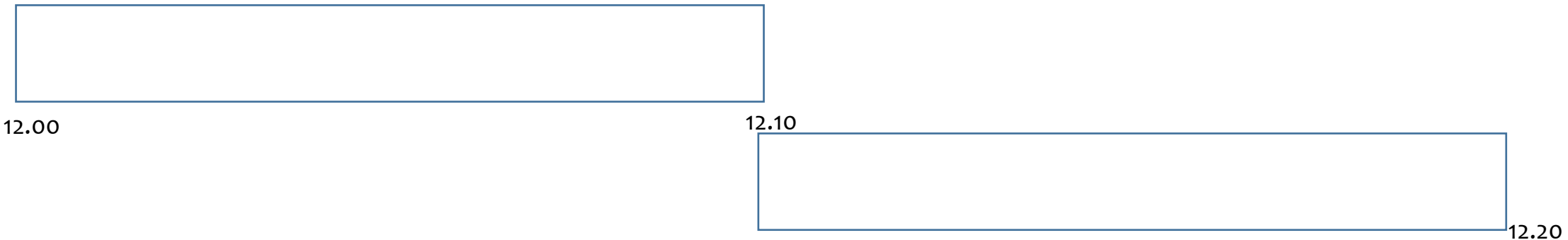
Imagine our quick example is modified and the stream now contains lines along with the time when the line was generated. Instead of running word counts, we want to count words within 10 minute windows, updating every 5 minutes. That is, word counts in words received between 10 minute windows 12:00 - 12:10, 12:05 - 12:15, 12:10 - 12:20, etc. Note that 12:00 - 12:10 means data that arrived after 12:00 but before 12:10. Now, consider a word that was received at 12:07. This word should increment the counts corresponding to two windows 12:00 - 12:10 and 12:05 - 12:15. So the counts will be indexed by both, the grouping key (i.e. the word) and the window (can be calculated from the event-time).



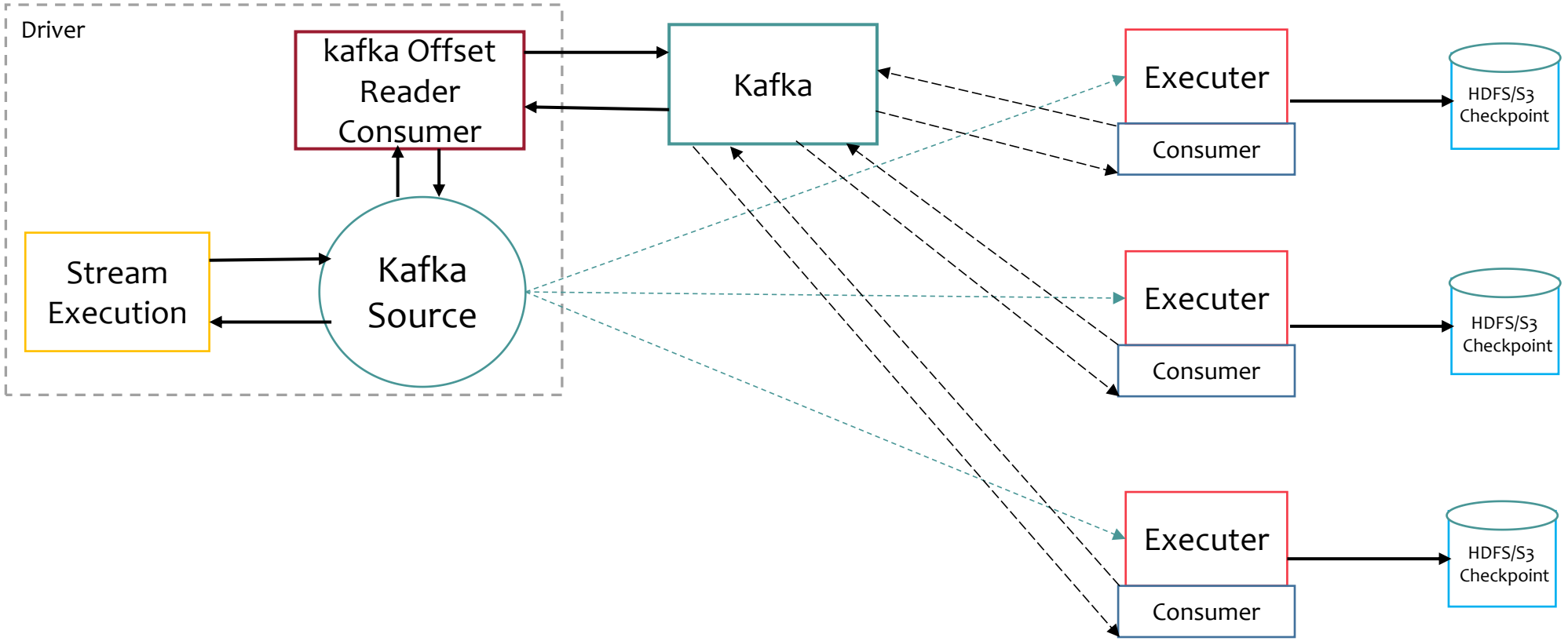
Sliding Interval: Windows will overlap



Tumbling Interval: No Overlapping b/w windows

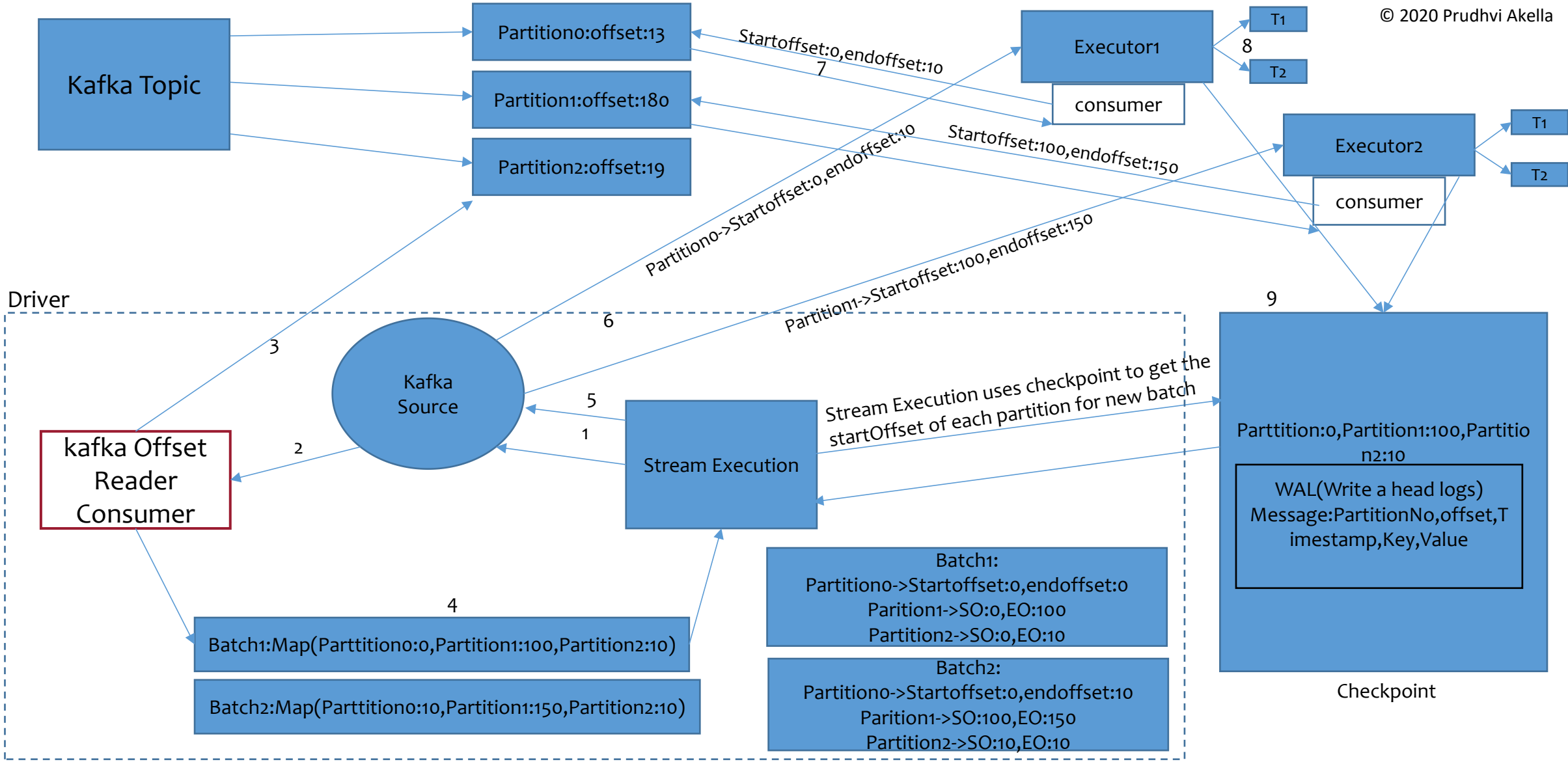


Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Kafka Spark Streaming Integration Internals

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

When spark application runs:
→In driver program three objects will be created 1) **kafka source,**

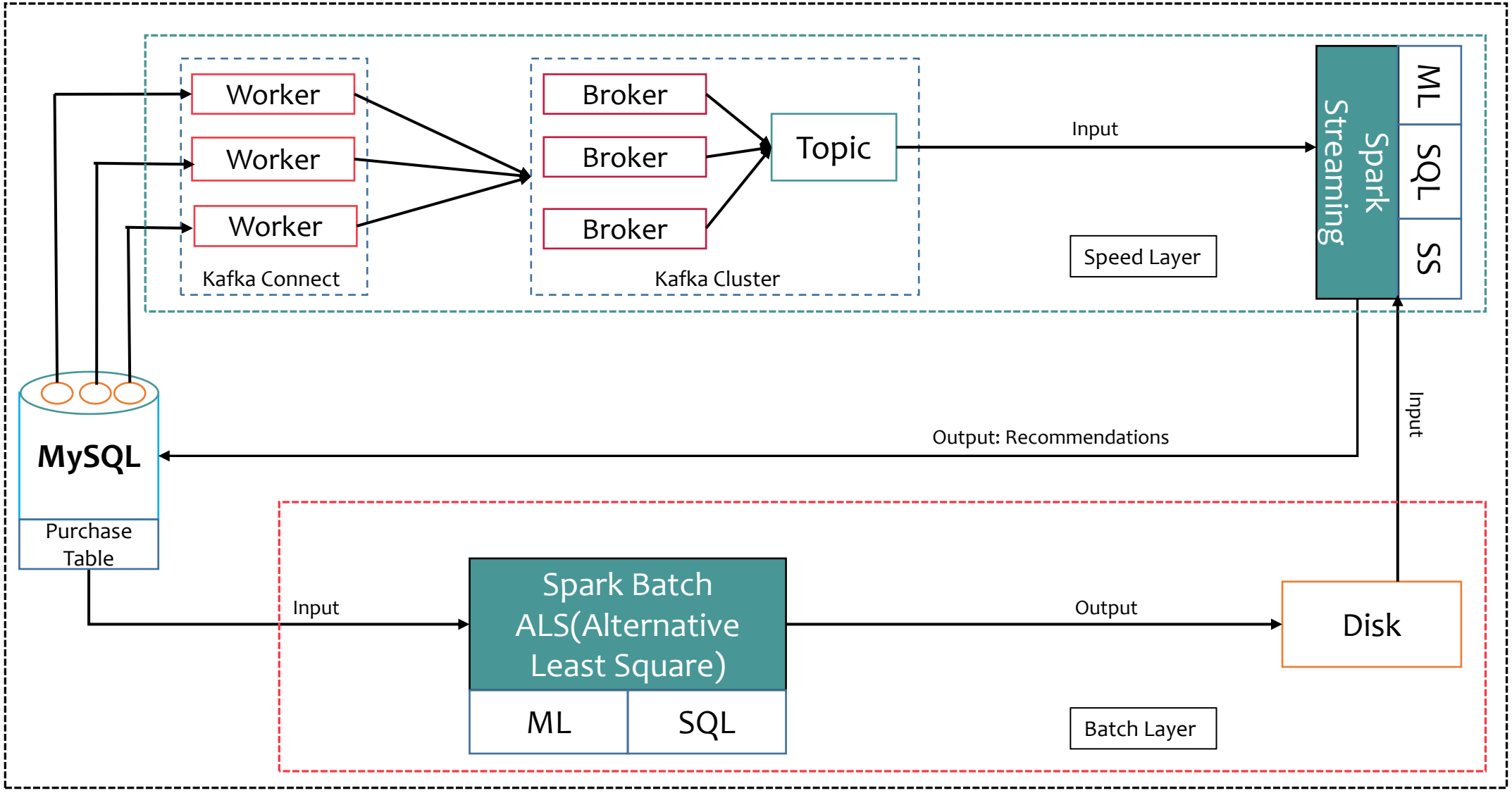
(Used to read latest offsets from kafka but it doesn't commit any offsets) and **stream Execution**.

→The first thing **StreamExecution** does with Kafka is the retrieval of latest offsets for each topic partition using kafka offset reader consumer **it returns Map** (topic partition number, offset). If you are running the spark streaming application for the first time then no checking pointing metadata will be available so it uses latest offsets from later-on from second query execution or on a application restart onwards by simple comparison between new offset and current offset for each partition.

→If new data is available then StreamExecution will call kafka source to distribute the offsets across the executors for real processing. Then executors will launch the consumers and launched consumers will get the partition offset data and stores it into the executor memory. If any executor and driver fails then executors will loss all the data. When new executor starts or driver doesn't know from which offset it has to process so if you want zero data loss or exactly-once schematics then enable the checkpointing and WAL(Write a head log) . If you enable WAL then executor will write messages to logs first before writing it to buffer. Once the offset record is processed successfully by executor then status of record will change to processed in log. It will effect the throughput. Lets say if checkpointing directory gets deleted then all the offset information is gone.

→ There are two types of check pointing
1) Metadata check pointing : In case of kafka it checkpoints the information about partition no, offset number, batch-id, group-id ect.
2) Data Check pointing: state of your aggregate operations.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in



Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

The Agenda of the project is to build Real Time Recommendation Engine which recommends different products to customer based on purchase History.
I was build using Lambda Architecture which has two layers:

1) Speed Layer to get Real Time Recommendations Engine
   Components User:
   →Kafka Connect(JDBC Source)
   →Spark(Structured Streaming(kafka Source, ForeachSink(JDBC)), ML)
2) Batch Layer to train ALS(Alternative Least Square Collaborative filtering Model)
   →Spark ML(AlS and Regression Evaluator(Root Mean Square Error to Evaluate ALS) Algorithm) ,SQL

Note: ALS can provide Recommendations for two types of rating(Implicit(clicks, views, purchase, shares, like) and Explicit(Rating)) .

As part of this project recommendation will be recommended to the Customers based on the explicit rating(Purchase history) to do that first step is in batch Layer ALS will be get trained ,tested, Evaluated using Root Mean Square Error Algorithm and trained output will be saved into output directory which will be used by spark streaming in speed layer to give recommendations to customer .
Below are steps involved in batch layer
→Connect SparkSQL to Mysql using JDBC connector and create a Data frame for purchasereco table(This step is skipped as part of our project and we are directly reading the rows from OnlineRetail.csv file)
→Preprocessing : Once the DF is created to improve the quality of the data filter the corrupted row before calling the ALS
→ Select CustomerID, ItemID column and add rating column(in our case purchase column lit(1)) which ALS requires for recommendations.
→Train Data, Test Data: Split the entire data into two DF using ArrayRandomSplit one is Train data which is used to Train the ALS and Test Data to validate whether Algorithm is trained properly or not.

- → Create ALS Algo with different by passing required like rank, iterations, customer id, Itemid, Rating columns parameters to it.
- → Train the ALS with train data using fit() method. Once its trained successfully then test the Model test data using transform() method. Notice one thing in the Data frame returned by transform will have predicted column append to test data which is predication given by ALS.
- → Check the performance of the Model using Regression Evaluator(RMSE) by passing DF that is returned by transform so that it provides a Double value it has to be as low as possible. Get 5 Recommendations for all users and save the model output into output directory

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

When ever the customer purchases any item then customer has to get the recommendation to achieve this in speed layer Kafka Connect is used to create incremental streaming layer on top of the MySQL PurchaseReco Table. So when every a new record gets inserted or any record gets updated then kafka connect worker will picks the record and pushes in to Kafka topic as soon as the record is committed to the topic the records will be pushed down to the spark to get the recommendations using ALS model trained output and recommendations will be stored into the MySQL recommendation Table.

Link might help further.

Address: DVS Technologies, Opp Home Town, Beside Biryani Zone, Marathahalli, Bangalore-37;Phone: 9632558585, 8892499499 |E-mail: info@dvstechnologies.in | www.dvstechnologies.in

http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
https://spark.apache.org/docs/latest/job-scheduling.html#scheduling-within-an-application
https://www.confluent.io/blog/configure-kafka-to-minimize-latency/
https://docs.databricks.com/delta/index.html
https://docs.databricks.com/spark/latest/dataframes-datasets/index.html
https://docs.databricks.com/spark/latest/structured-streaming/index.html

Use Cases:
https://databricks.com/blog/2017/10/05/build-complex-data-pipelines-with-unified-analytics-platform.html
https://databricks.com/blog/2018/07/09/analyze-games-from-european-soccer-leagues-with-apache-spark-and-databricks.html
https://databricks.com/blog/2018/08/09/building-a-real-time-attribution-pipeline-with-databricks-delta.html
https://docs.databricks.com/delta/index.html