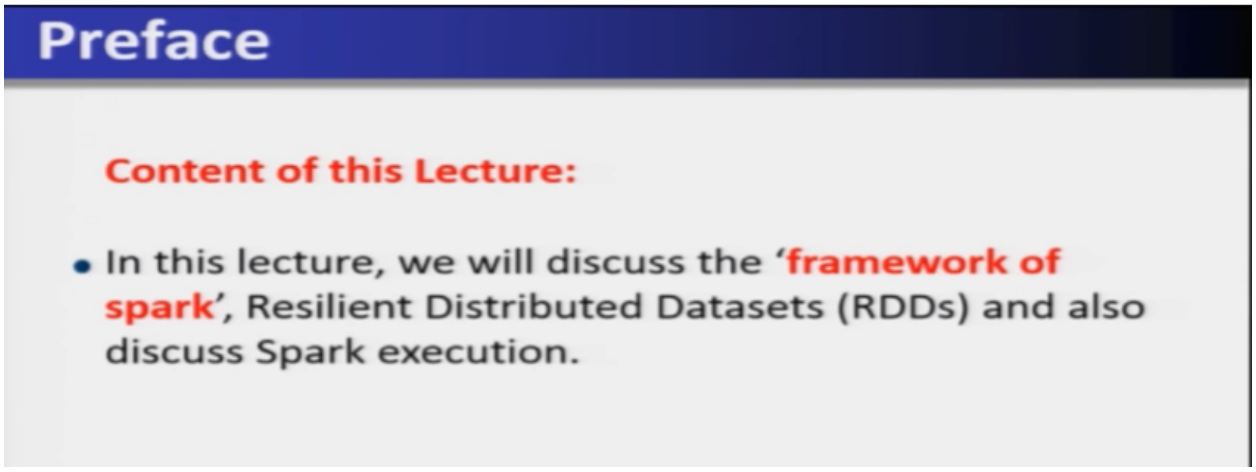
Lecture -10

Introduction to Spark

Refer slide time: (0:13)

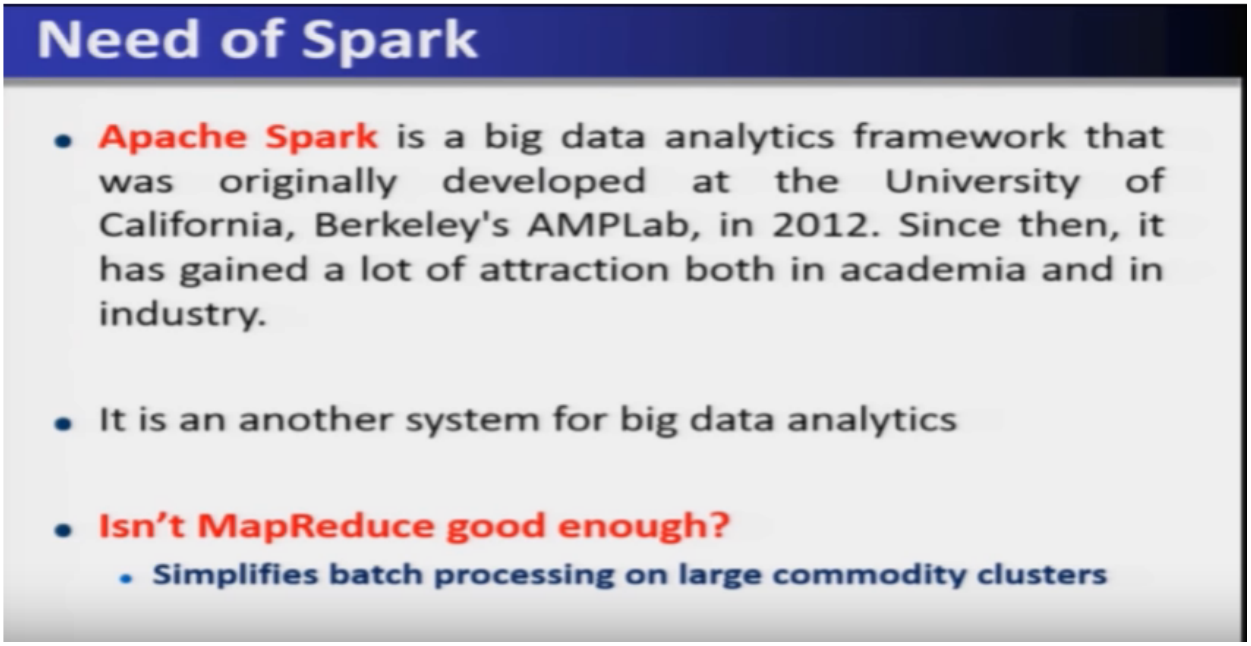
Introduction to Spark.

Refer slide time: (0:15)



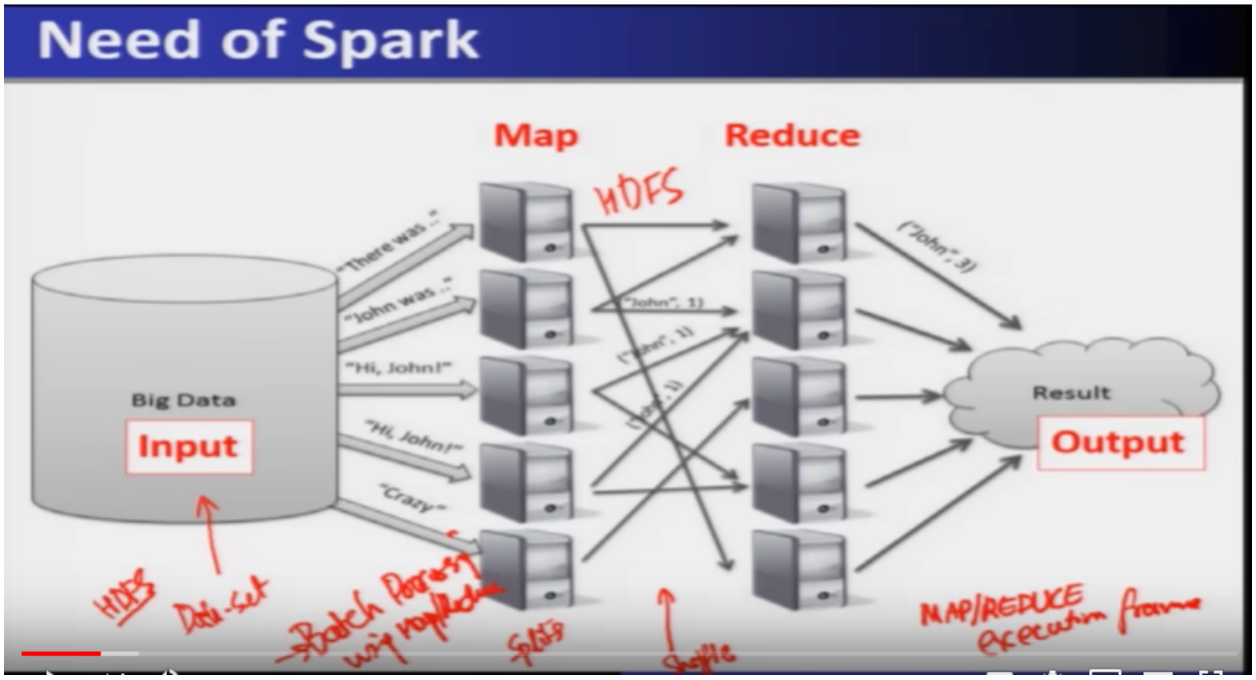
Preface: Content of this lecture: This lecture, we will discuss the 'framework of Spark'. Resilient distributed data sets and also, we will discuss the, Spark execution.

Refer slide time: (0:30)



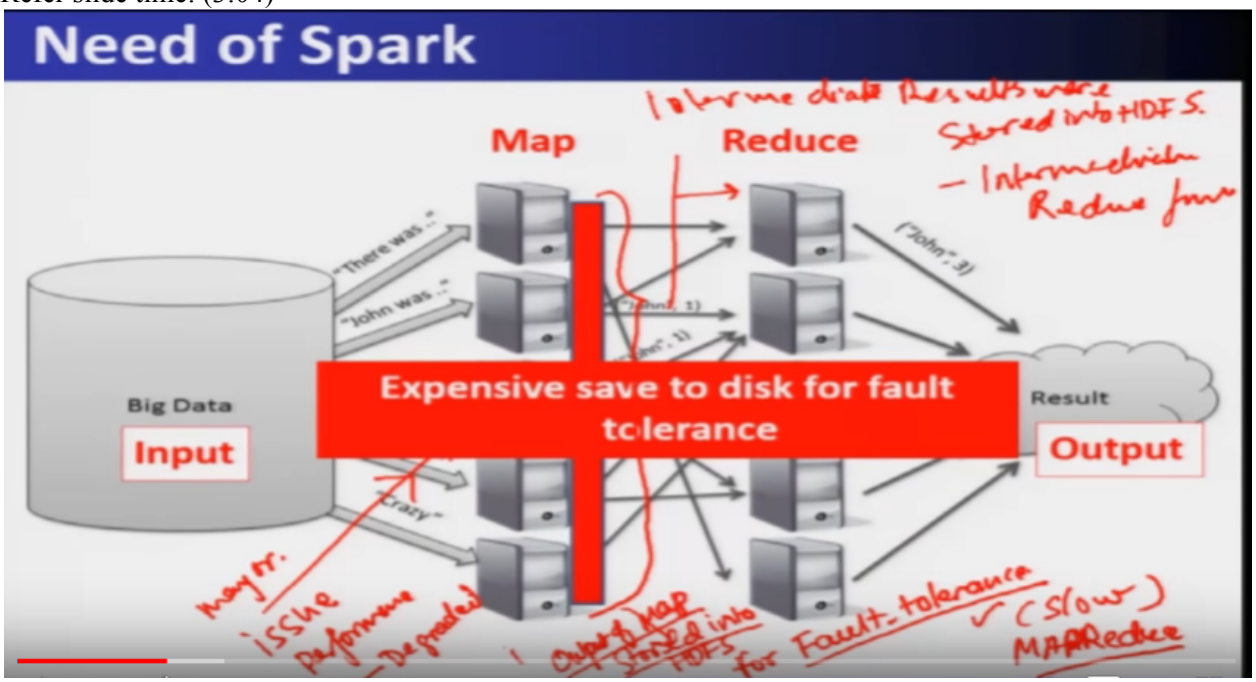
The need of his Spark. Apache Spark is a big data, analytics framework that was originally developed, at University of California, Berkeley, at AMP Lab, in 2012. Since then, it has gained a lot of attraction, both in the academia and in the industry. It is, an another system for big data analytics. Now, before this is parked the Map Reduce was, already in use. Now, why isn't the Map Reduce good enough? That we you are going to explore, to understand, the need of the Spark system, before that we have to understand that, Map Reduce simplifies, the batch processing on a large commodity clusters.

Refer slide time: (1:23)



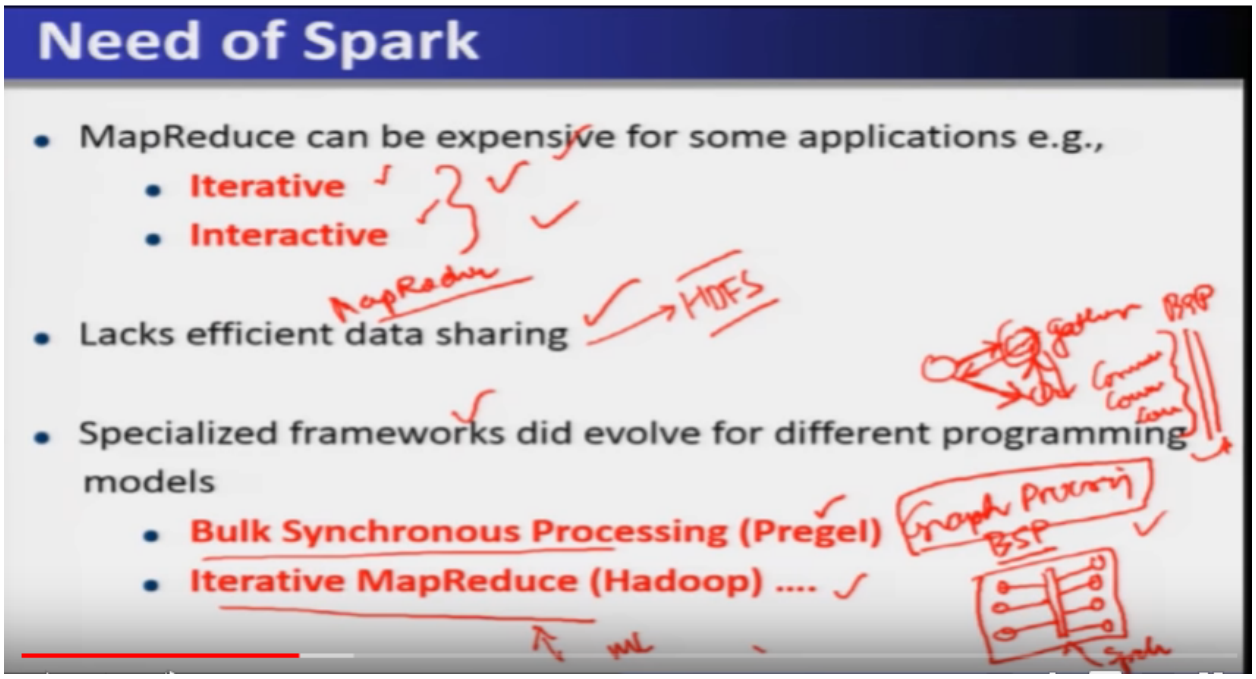
So, in that process, the data file or a dataset, which was the input? which was input, through the HDFS system, uses, the map function, to be spitted into across different, splits and then the map function was applied on to this particular splits, which in turn will generate, the intermediate values, which is in the form of shuffle and is stored in HDFS and then passed on, to the reduced function giving the output, so this is, the scenario of a Map Reduce, execution framework, as you have seen the data set, is now, stored in the HDFS file system. Hence, this particular computation is done for the batch processing system. So we have seen that, this particular batch processing, was done using the Map Reduce system and that was in use earlier, before the development of a Spark.

Refer slide time: (3:04)



Now let us see, why this particular model, of Map Reduce is not good enough to be there? Now, as you have seen, in the previous slide that, the output of the Map function, will be stored in HDFS and this will ensure the fault tolerance. So the output of, map function, was stored into HDFS, file system and this ensures of the fault tolerance. So in between, if there is a failure, when it is stored in the HDFS file system, still the data is not lost and is completely saved, that is why? It is ensuring the fault tolerance and because of that fault tolerance, the intermediate values or intermediate results, out of the map functions were stored into HDFS file. Now, this particular intermediate results will be now, further fed on to the reduced function. So that was, the typical Map Reduce, execution pipeline. Now, what is the issue? The problem is, mentioned over here, that the major issue is the performance. so that means, the performance is degraded and it has become expensive, to save to the disk for this particular activity of fault tolerance, hence this particular Map Reduce, framework becomes quite slow, to some of the applications, which are interactive and required to be processed in the real time, hence this particular framework, is not very useful, for certain applications.

Refer slide time: (5:50)

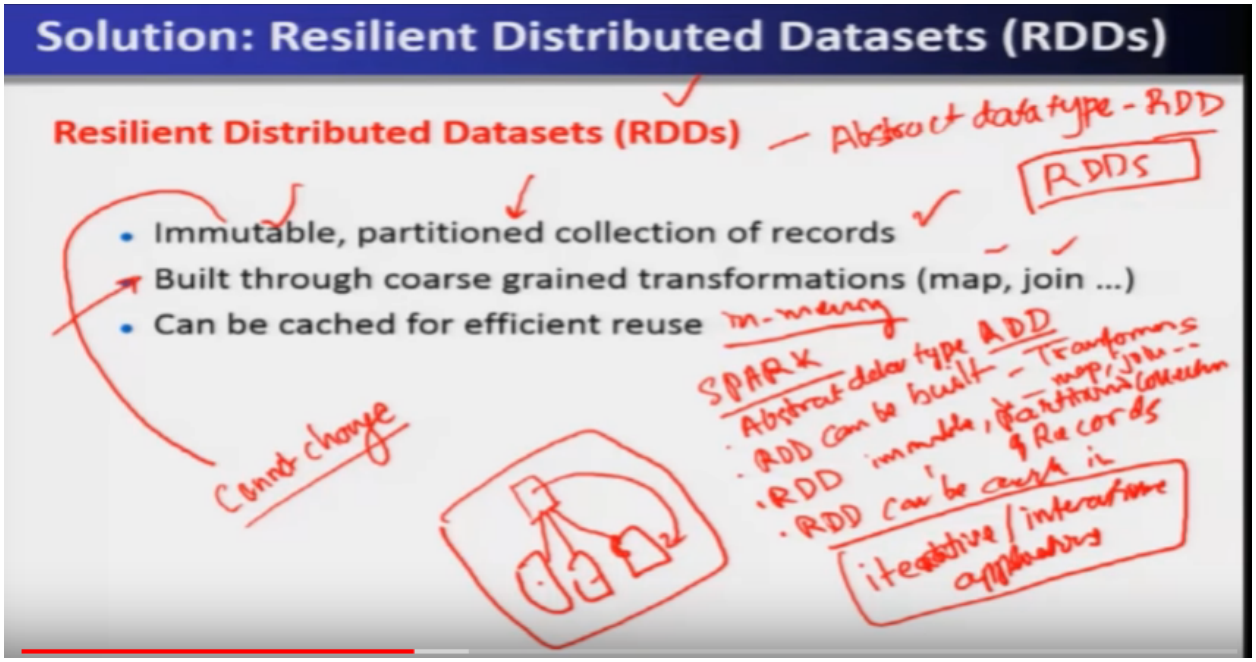


Therefore, let us summarize that, Map Reduce can be expensive, for some applications and for example, interactive and iterative application. They are very expensive and also, this particular framework, that is the Map Reduce framework, lacks efficient data sharing, across the map and reduce phase, of operation or iterations. So lacks efficient data sharing in the sense, the data which is, the intermediate form of Map Reduce is, map is stored in the file system, HDFS file system. Hence, it is not shared, in an efficient manner. Hence, this particular sharing the data across map and reduce phase, has to be through the disk and which is a slow operation, not an efficient operation. Hence, this statement which says that, Map Reduce, lacks the efficient data sharing, into the system .so due to these, drawbacks, there were several specialized framework did evolve, for different programming models, such as, so the pregel system was, there for graph processing, using bulk synchronous processing BSP. and then, there is another framework, which is called,' Iterative Map Reduce' was also made and they allowed, a different framework some a specialized framework, for different applications to be supported.

So the drawback of Map Reduce has, given the way to several frameworks and they were involved a different programming models, so bulk synchronous, parallel bulk, synchronous parallel processing framework, is about there is a synchronization barrier, so the processes at different speed the joints at the synchronization barrier and then they will proceed further on, so this is called, 'BSP model'. so this BSP model also, allows a fast processing for the typical graph application, so graph processing is done using BSP, to make it more faster. Because, the Map Reduce doesn't support the graph processing within it. so grab processing framework requires, that the data which is being, taken up by the neighboring nodes, then basically these nodes ,the neighboring node will collect the data, will gather the data and then perform the computation and then again scatter, these data to the other points. so this particular operation, the communication, computation and then communication, is to be completed as one step, that is the lockstep and hence the bulk synchronous processing, comes into the and effect where this all three operations, all three actions, are to be performed and then only the next step will takes place using bulk synchronous processing. Hence, this kind of paradigm, that is called,' Bulk Synchronous' processing framework ,is

useful for the graph processing, that we will see that, how this particular different programming paradigm, such as bulk synchronous processing and iterative Map Reduce ,for the iterative applications, for example, the iterations of a Map Reduce, is basically you can see, in the machine learning algorithms. so these different frameworks were evolved, out of this Map Reduce drawbacks and they, they existed, over the several period, over the period of time, to fill up this particular gap. Now, seeing all these scenarios and to provide the data sharing and to support the applications such as interactive and iterative applications, there was a need for the SPARC system.

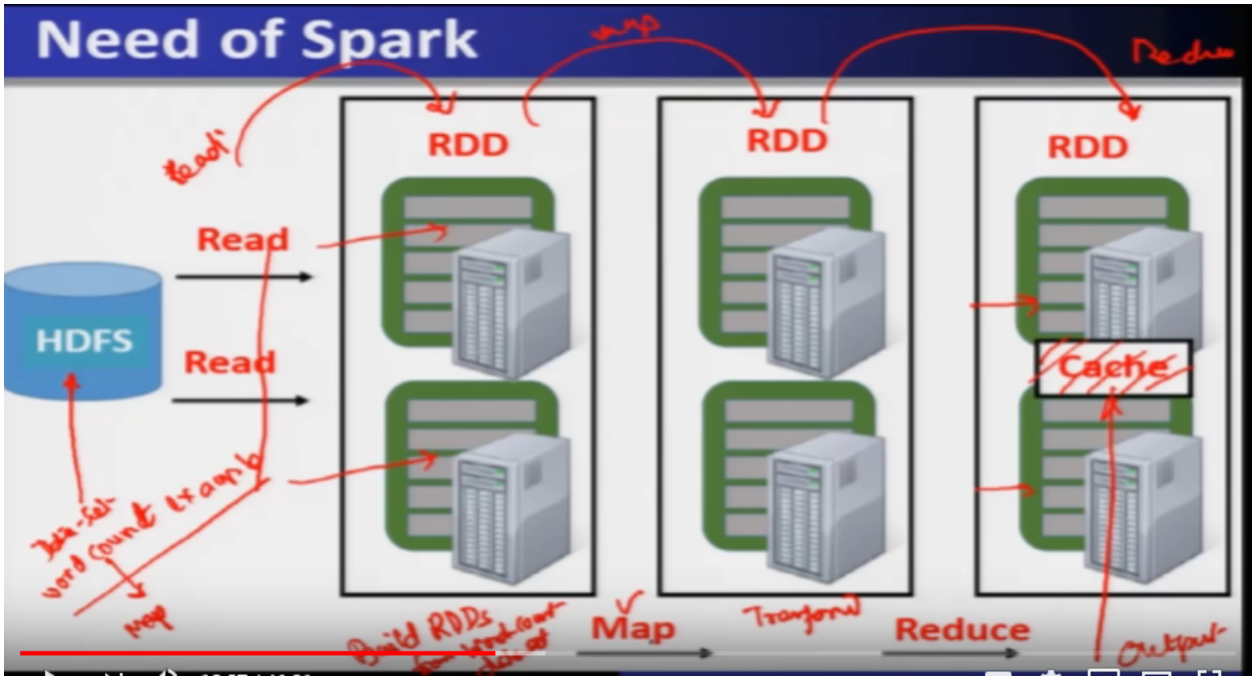
Refer slide time: (10:38)



So the solution: which is Spark has given, is in the form of an abstract data type, which is called, 'Resilient Distributed Data Set'. So the SPARK provides, a new way, of supporting the abstract data type, which is called resilient distributed data set or in short it is our DTS. Now, we will see in this part of the discussion, how this RDDs are going to solve ,the drawbacks of the batch processing Map Reduce, into an more efficient data sharing and also going to be supporting, the iterative and interactive applications. Now, this particular RDDs, are resilient distributed data sets, they are immutable, immutable means, we cannot change. It cannot be changed, so that means once an RDD is formed, so it will be an immutable. Now, in this particular way, this immutable resilient distributed data set can be partitioned in a various ways, across different cluster nodes. So partition collection of Records, so partitioning can happen, in s for example, if this is a cluster, of machines which are connected, with each other. So the RDDs can be partitioned and stored, at different places, at different segments. Hence, the immutable partition collection of records is possible and in this particular scenario, that is called, 'RDDs'. Now, another thing is, once an RDD is formed, then it will be formed using, it will be built, RDDs will be built, through a course gain transformations, such as, map, join and so on. Now, these RDDs can be cached for efficient reuse, so that we are going to see that, lot of new operations can be performed on it, so again let us summarize, that the Spark has, given a solution, in the form of an abstract data type ,which is called as a, 'RDD.' and our

RDD can be built, using the transformations and also can be changed, can be changed into another form, that is RDD can become another, RDD by making various transformations, such as map, join and so on. That we will see in due course of action, these RDDs are, are immutable, partition collection of record. Means that, once an RDD is formed, so as an immutable, immutable means, we cannot change, this entire collection of records, can be stored and in a convenient manner onto the, onto the cluster system. Hence, this is an immutable partition collection of the record, these RDDs can be cached, can be cached in memory, for efficient reuse. So as we have seen that, Map Reduce lacks, this data shearing and now using the RDDs, a Spark will provide, the efficient, sharing of data in the form of, all RDDs.

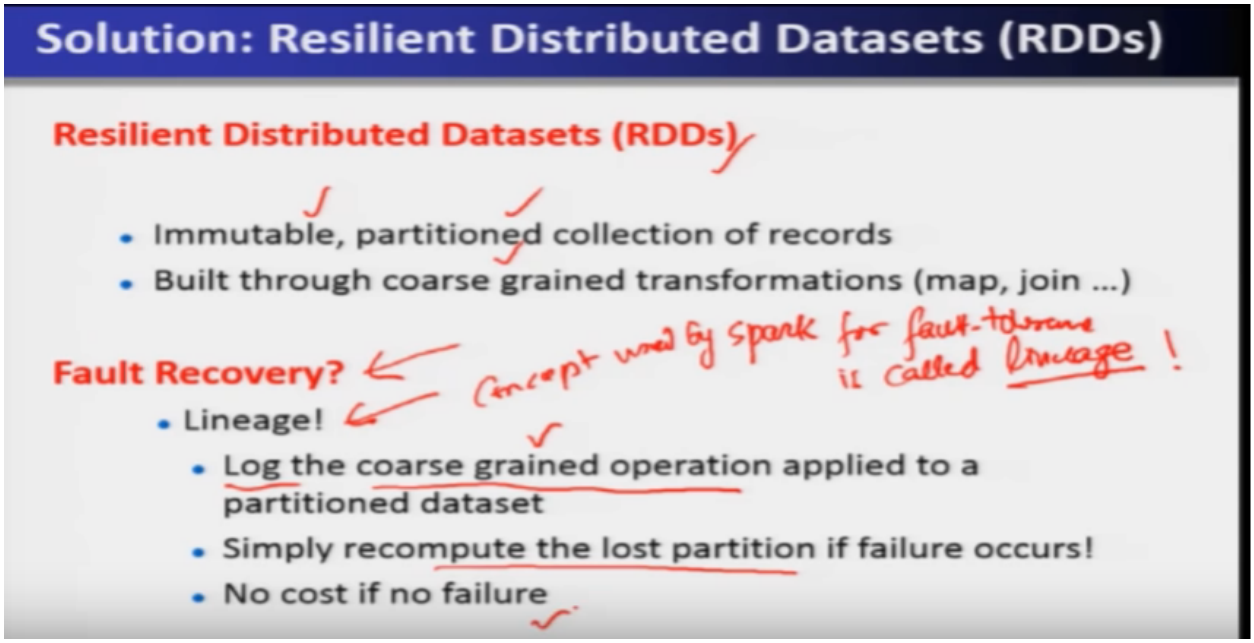
Refer slide time: (15:02)



Now let us see, through an example of a word count. Word count example, to understand the need of Spark. So in the word count application, the data set, is installed through the HDFS file system and is being read, so after reading this particular data, set from the file system, this particular reading operation, will build the RDDs. and which is shown here, in this block number 1. So that means once, the data is read, it will build, the RDDs from the word-count data set. Once these RDDs are built, then they can be stored at different places, so it's an immutable partitioned collection and now various operations we can perform. So we know that, these RDDs, we can perform various transformations and first transformation which we are going to perform on these RDDs, which are, which is called a, 'Map Function'. Map function for, the word count program, is being applied on different RDDs, which are restored. So after applying the map function, this particular output, will be stored in memory and then again, the reduced function will be applied, on these RDDs, which is the output? Or which is the transformations? Which is the transform or RDDs, out of the map function? Again the reduce function will apply. And the data and the result of this reduce, will remain in cache. so that, it can be, used up by different application .so you can see that, this particular transformation, which is changing the RDDs from one form, to another form that means after reading, from the file, it will become an RDD and after applying the map function, it will

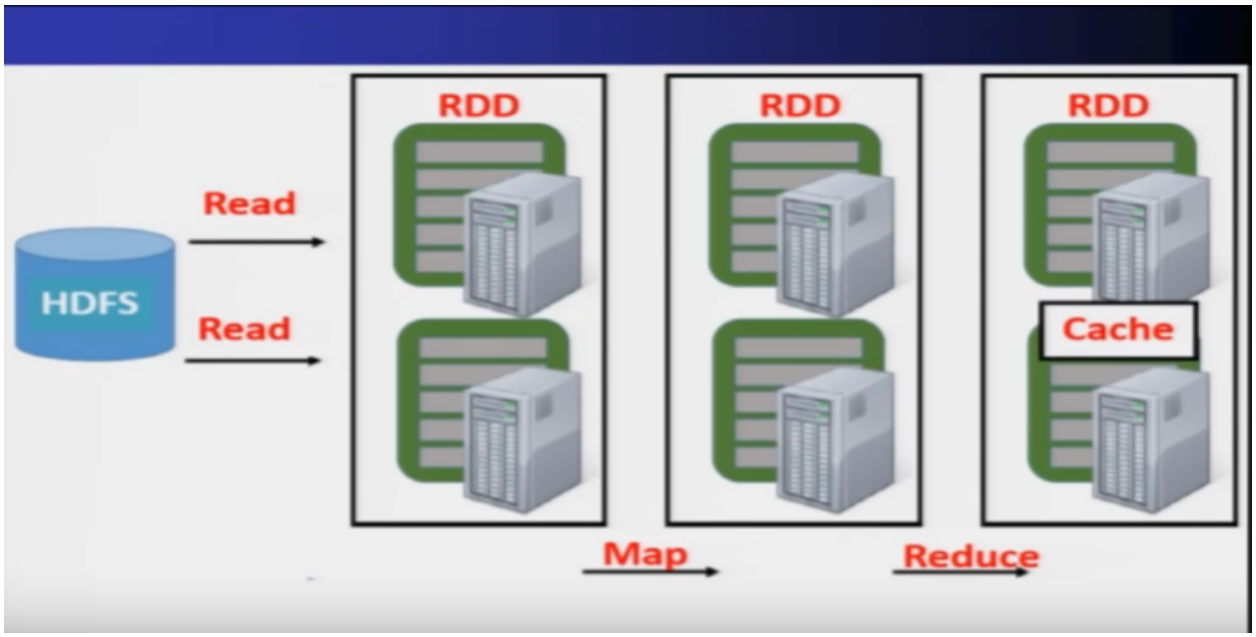
change to another RDD and map function and after applying the reduce function. it will change to, another form and the final output, will be remained in the cache memory. Output will remain in the cache, so that, whatever application requires, this output can be used up, so this particular pipeline, which we have shown, is quite, easily, understandable and is convenient to manage and part and to store, in the partition, collection manner, in this cluster computing system.

Refer slide time: (18:11)



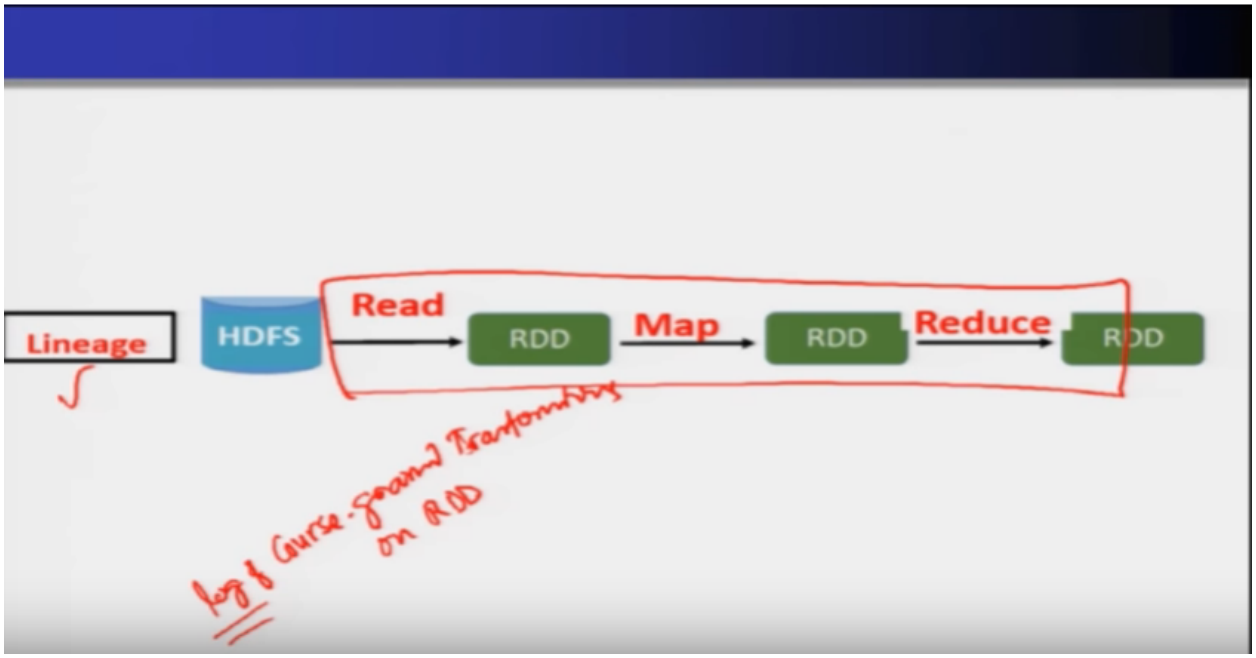
So, we have seen that, this RDDs has simplified this particular task and has also made this operation efficient. Hence, RDDs is an immutable, partition collection of the records. And they are built through the coarse grained transformation that we have seen in the previous example. Now, another question is, since the Map Reduce was storing ,the intermediate results of a map, before it is being used in the reduced function, into an SDFS file system, for ensuring the fault tolerance .now since, by produced since the SPARC is not using, this intermediate results storage, through the SD FS rather, it will be in memory storage, so there will be how this Spark ensures the fault tolerance that we have to understand now, the concept which Spark uses, for fault tolerance is called,' Lineage'. So this park uses the lineage to achieve the fault tolerance and that we have to understand now, so what Spark does is? It locks, the coarse-grained operations, which are applied, to the partition data set. meaning to say that, all the operation like, reading of a file and that becomes an RDD and then making a transformation on an RDD using map function and then again another transformation, of RDDs using reduced function, join function and so on. All these operations they form, the course gained operations and they are to be logged into a file, of before applying it. so if the data is, so basically ,if the data is, lost or if the, if the system, crashes the node crisis, they simply recomputed, these lost partition and whenever there is a failure, if there is no failure, obviously no extra cost ,is required in this process.

Refer slide time: (20:37)



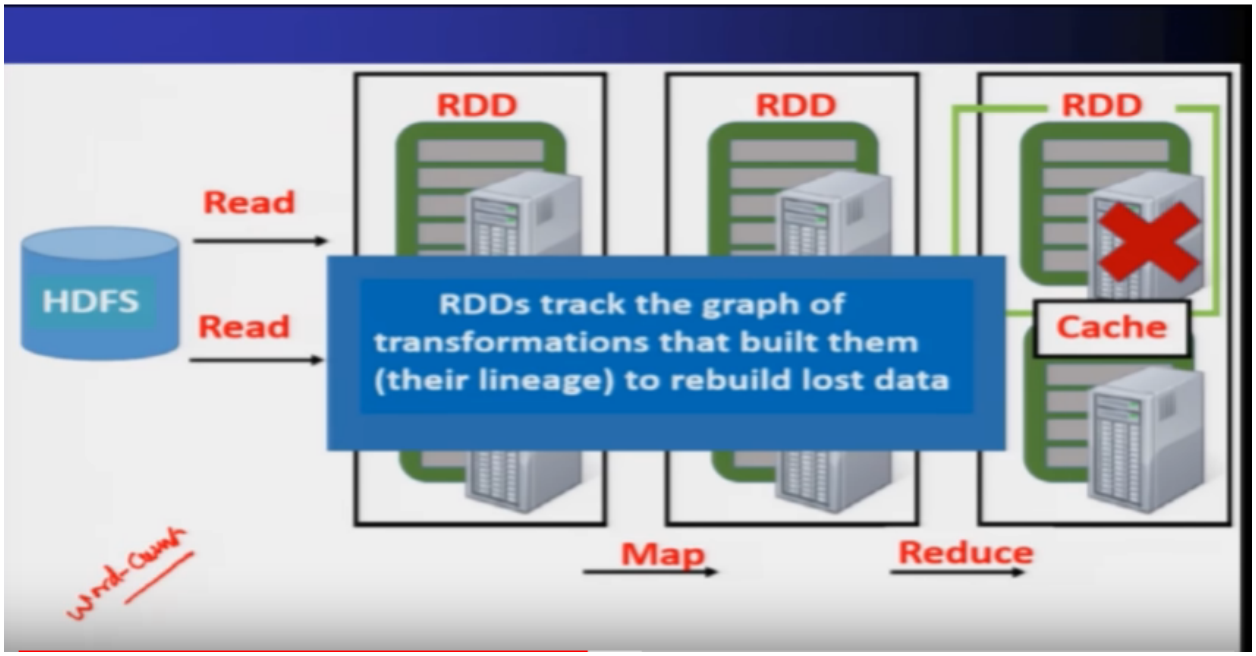
Let us see this, through an example.

Refer slide time: (20:40)



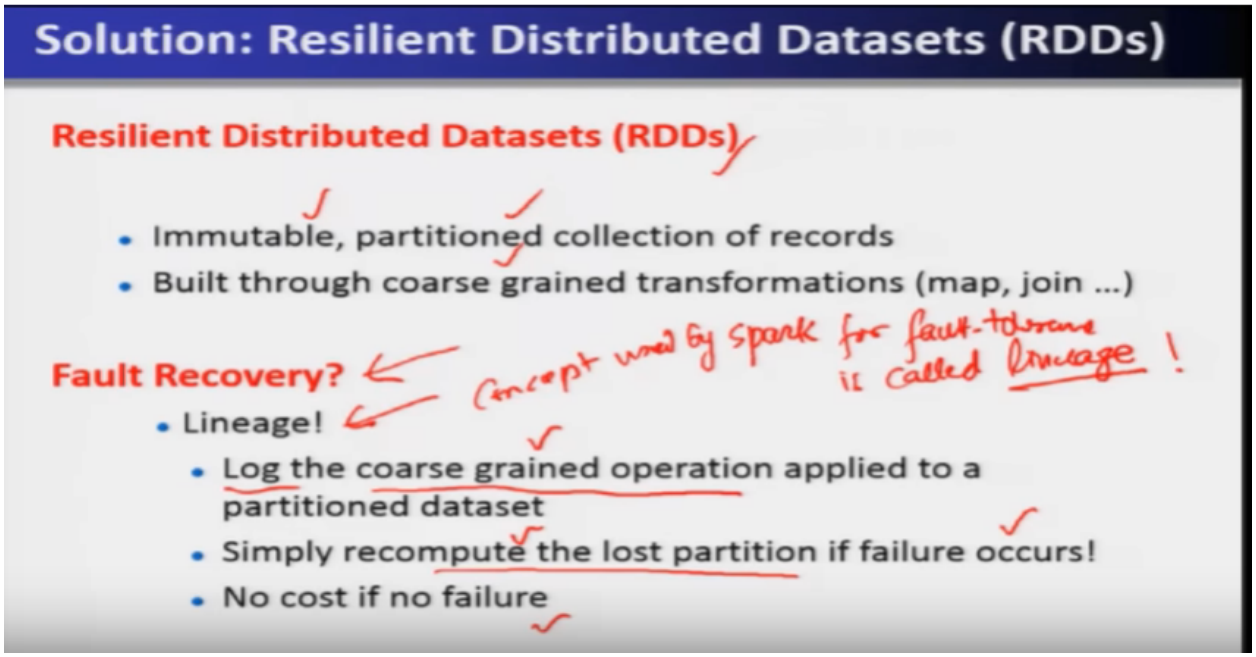
So again, let us explain that, lineage is in the form of the course grained, you said, it's a log of a coarse grained operation. Now this particular, lineage will, keep a record of all these operations, coarse-grained operations, which are applied and that will be kept in a log file. Now we will see, how using this lineage or a log files, the fault tolerance can be achieved.

Refer slide time: (21:24)



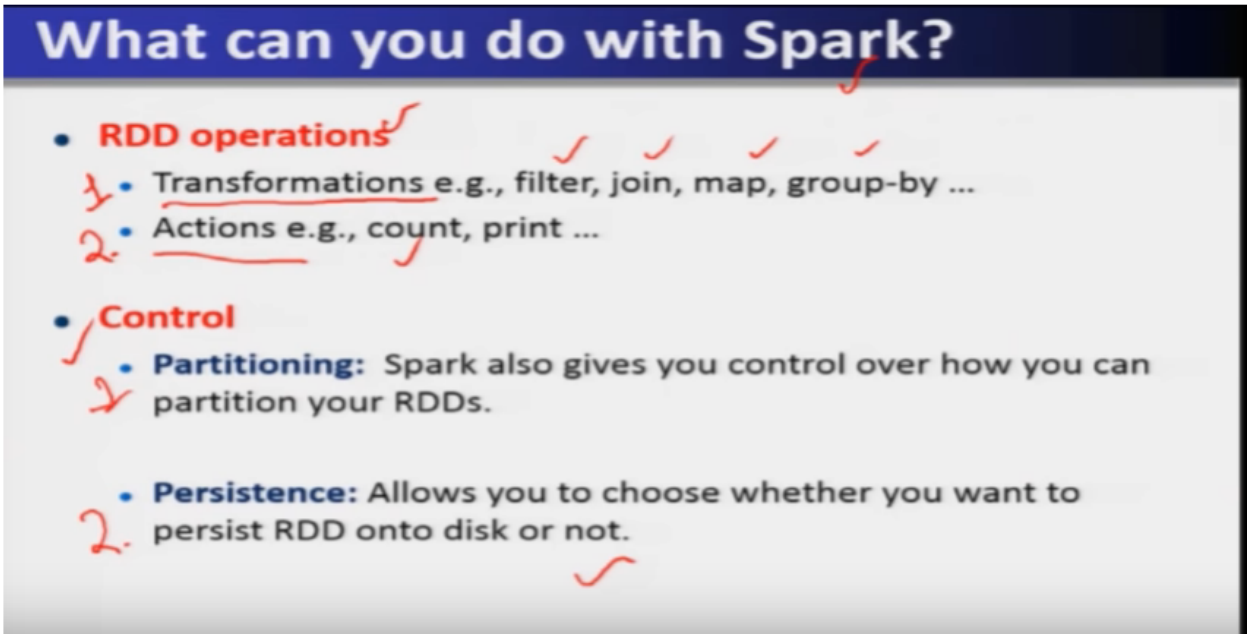
Let us see, through this particular example. That let us see, that the word count example, which we have seen in the last slide. Now the, the same word count example, we have to, we have, we have to see that, these RDDs, will keep track of ,oddities will keep track, the graph of transformation, that build them, their lineage to rebuild lost data. So the, so there will be a log of all the coarse grained operation, which are performed and which has, built these RDDs transformations. and this is called,' Lineage'. let us see, what happens is for example, after reading this particular RDD will be formed after the read operations, on the data set and then, on this particular data, this RDD we have performed, the map operation, RDD transformed RDDs and this transformed RDD again, is now applied with the reduced function, to make this particular RDD and is stored in the cache. Now, consider that if this particular node, which has stored this transformed RDD if it is filled, obviously it has to trace back, to the previous RDD and consult, this lineage, which will tell that, this is an output of the map function, this is an RDD transformed and RDD, this RDD when we apply, the reduced function, it will recreate, the same RDD which is lost. So let us see, what is written? What we have just seen?

Refer slide time: (23:17)



So we have to simply, recomputed, the last partition, whenever there will be a failure, how we have to trace back and apply ,the same transformation again, on RDD and we can recomputed, that the, the RDD which is lost in the partition ,due to the failures. So now, using lineage, concept we have seen that the fault tolerance, is achieved in a Spark system.

Refer slide time: (23:48)

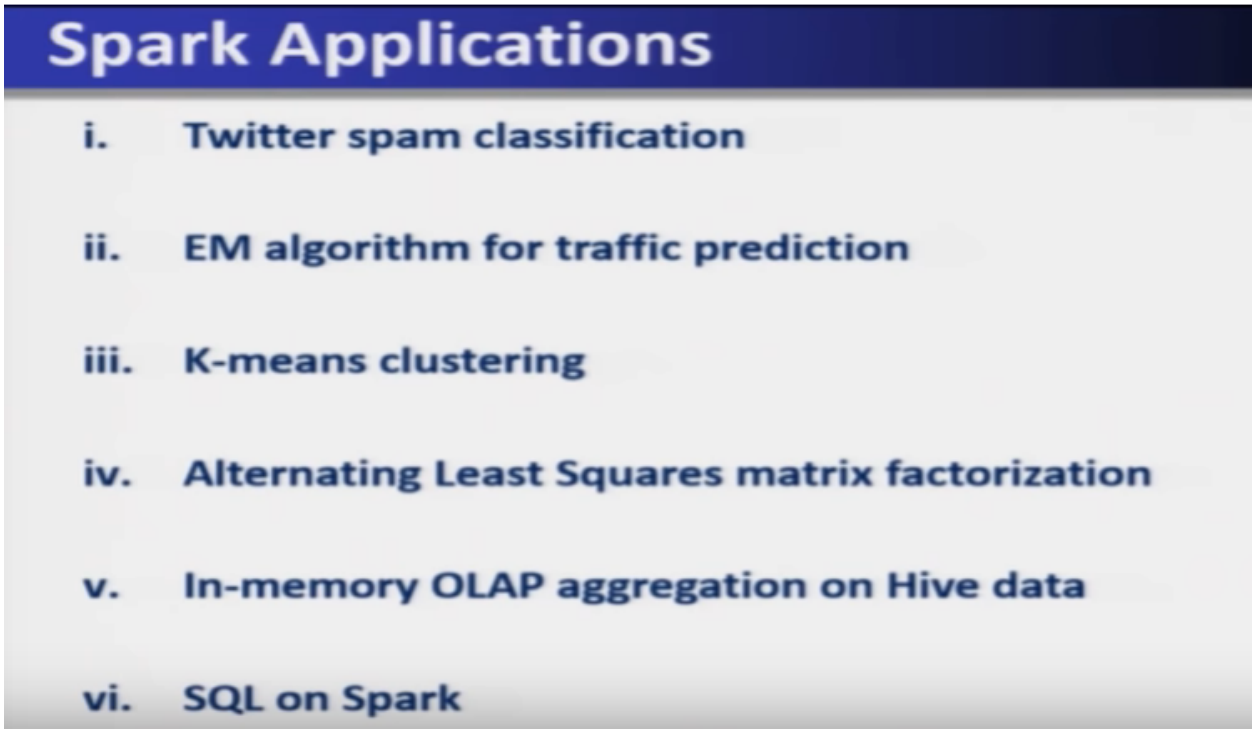


Now we will see that, what more we can do here in the Spark? So RDDs transfer, which RDDs provide various operations and all these operations are divided into two different categories, the first category is called, ‘Transformations’. Which we can apply, as an RDD operation. second is called, ‘Actions’, which

we can perform using RDDs, operations so as far as the transformations, which RDD supports is in the form of filter, join, map, group by all these are different transformations, which RDD supports, in the Spark system .Now, another set of, operation which RDD supports is called, 'Actions'. so actions, in the sense the output of some, some operations, is whenever there then it is called,' Action'. For example, count, print and so on. Now, then another thing which, Spark can provide is called,' Control Operations', to the programmer level.

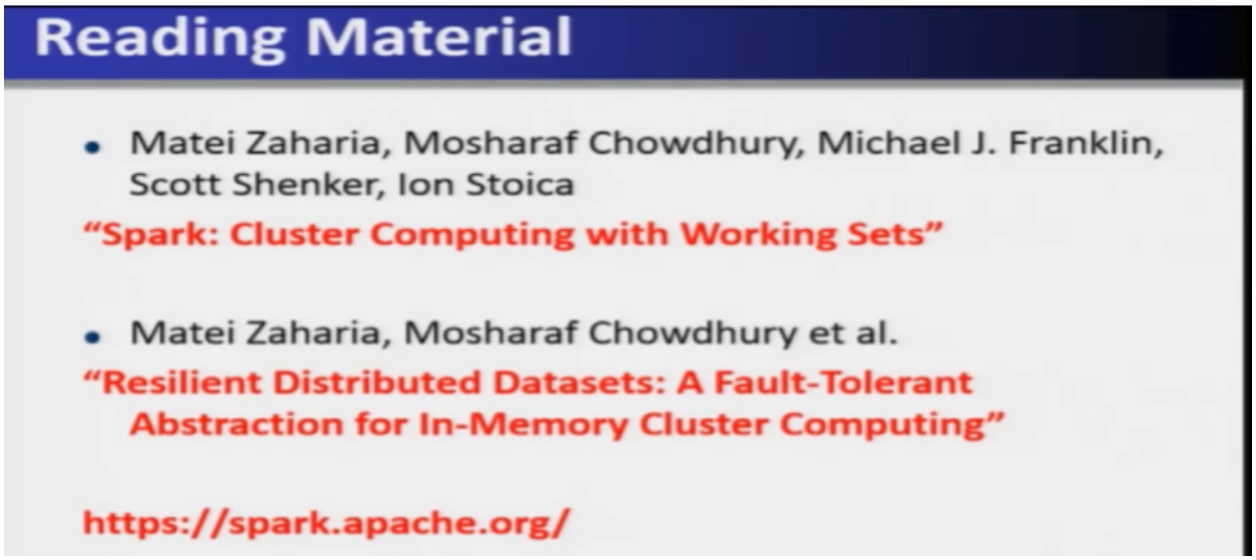
So there, are two interesting control, which is being provided by the Spark, to the programmers. The first is called,' partitioning'. So, the Spark gives the control or how you can partition your RDDs, across different cluster systems. and second one is called the,' Persistence'. Persistence allows you to choose, whether you want to persist RDDs on to the disk or not. So by default, it is not persisted, but if you allow, if you choose this persistent, RDDs then the, RDDs I have to be stored in HDFS. Hence, the persistent and partitioning both controls are, given to the, to the programmer the user in buys by the Spark system.

Refer slide time: (25:36)



There are various other, Spark applications, where Spark can be used first, these applications are such as, Twitter respond classification, algorithm for traffic prediction, k-means clustering algorithms, alternating least square matrix factorization, in memory OLAP aggregation on his, pond hive data and SQL on Spark.

Refer slide time: (26:02)

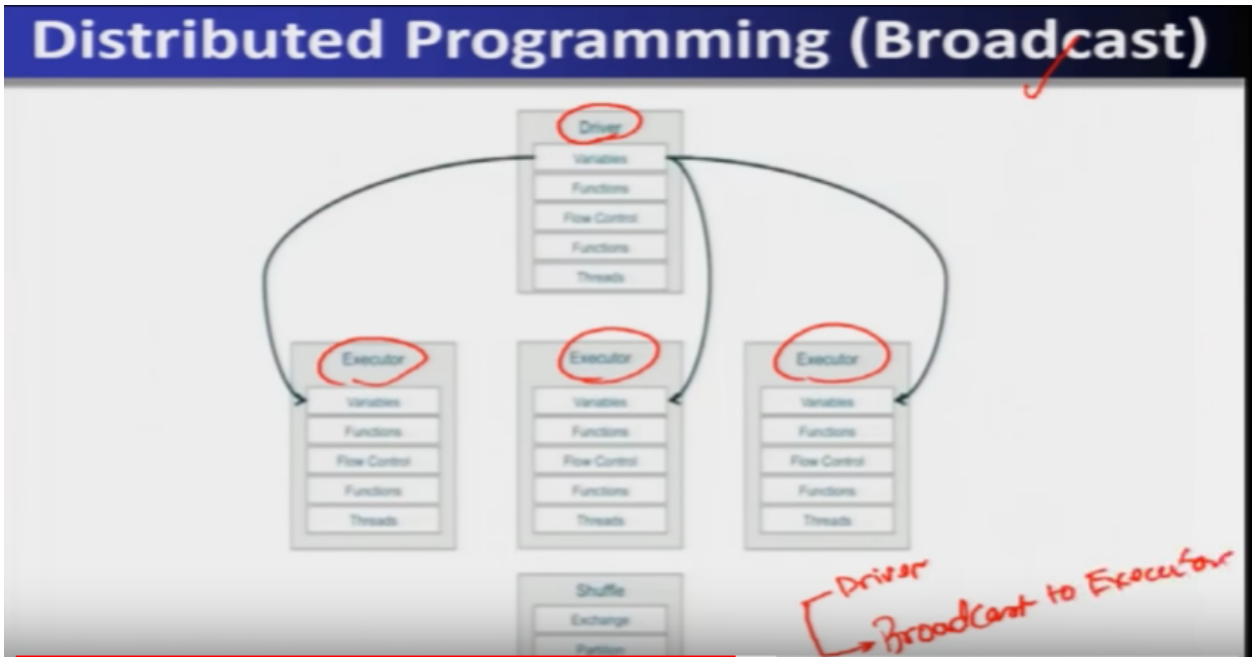


These are some of the applications and these are the reference, material for further studies, on the Spark system. That we have, that is available on HTTPS, Spark dot Apache dot org.

Refer slide time: (26:18)

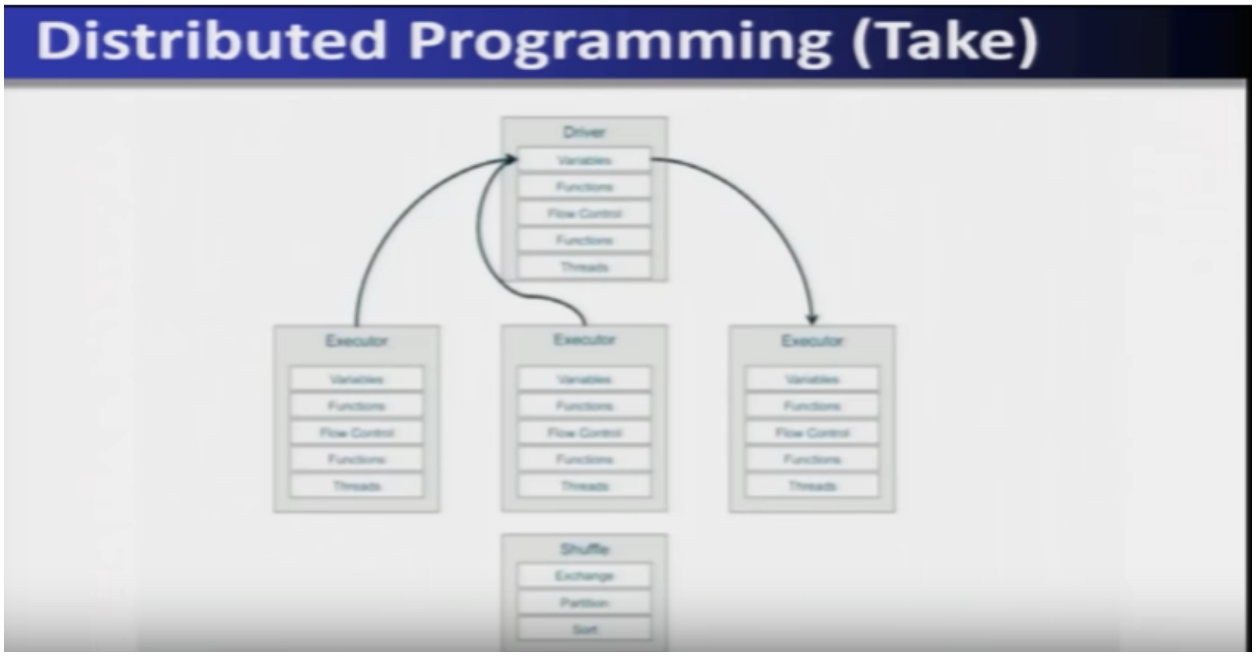
Now we will see, about the Spark execution.

Refer slide time: (26:21)



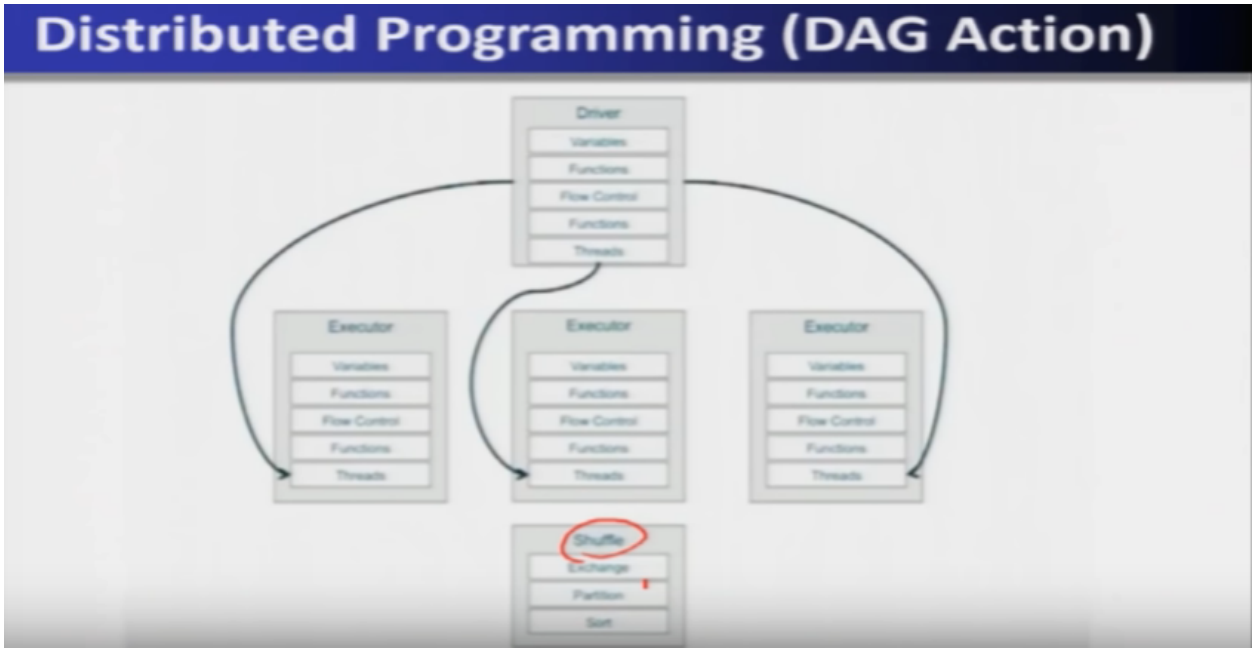
So a Spark execution, is done, in the form of distributed programming, that is, the first operation is called,' Broadcast'. So there is a, there are three different entities, one is called the, 'Driver Entity', of the Spark, the other entity is called, 'Executors'. Which are there on different nodes, data nodes and then there is a shuffle operation. So let us see the, first operation is called, 'Broadcast'. So the driver will broadcast, these different commands, to the different executors, that is called a, 'Broadcast Operation'. We'll broadcast, to the executors.

Refer slide time: (27:21)



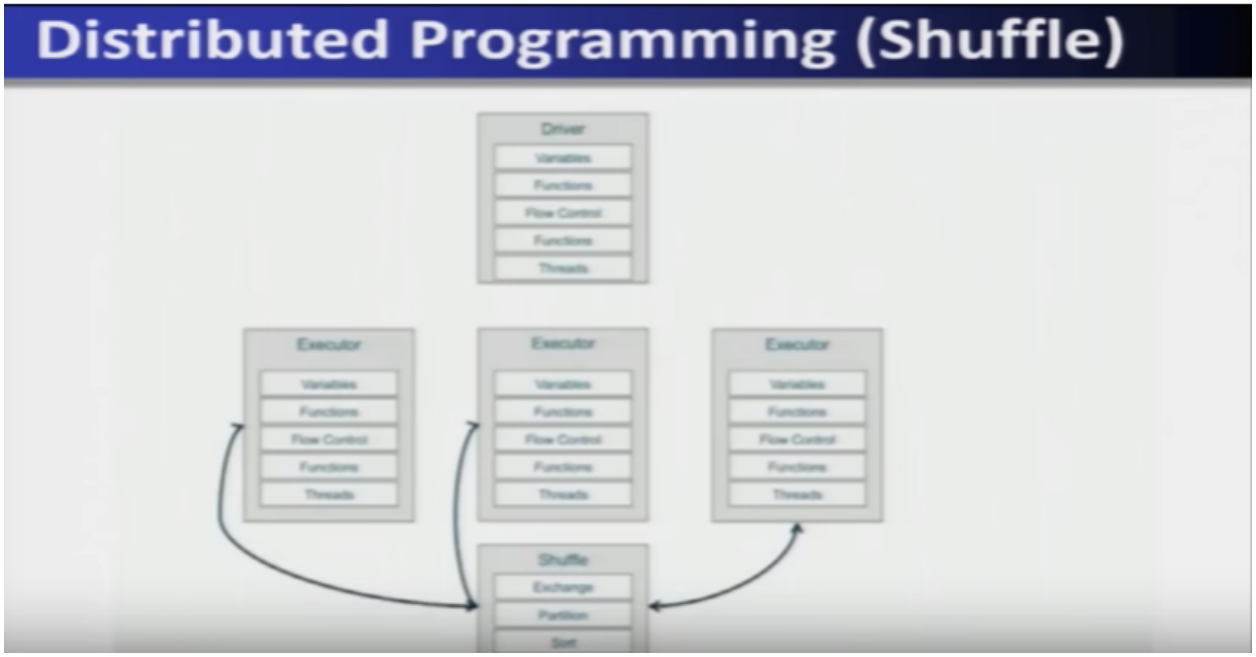
Now, these executors will execute and give the result back to the drivers.

Refer slide time: (27:22)



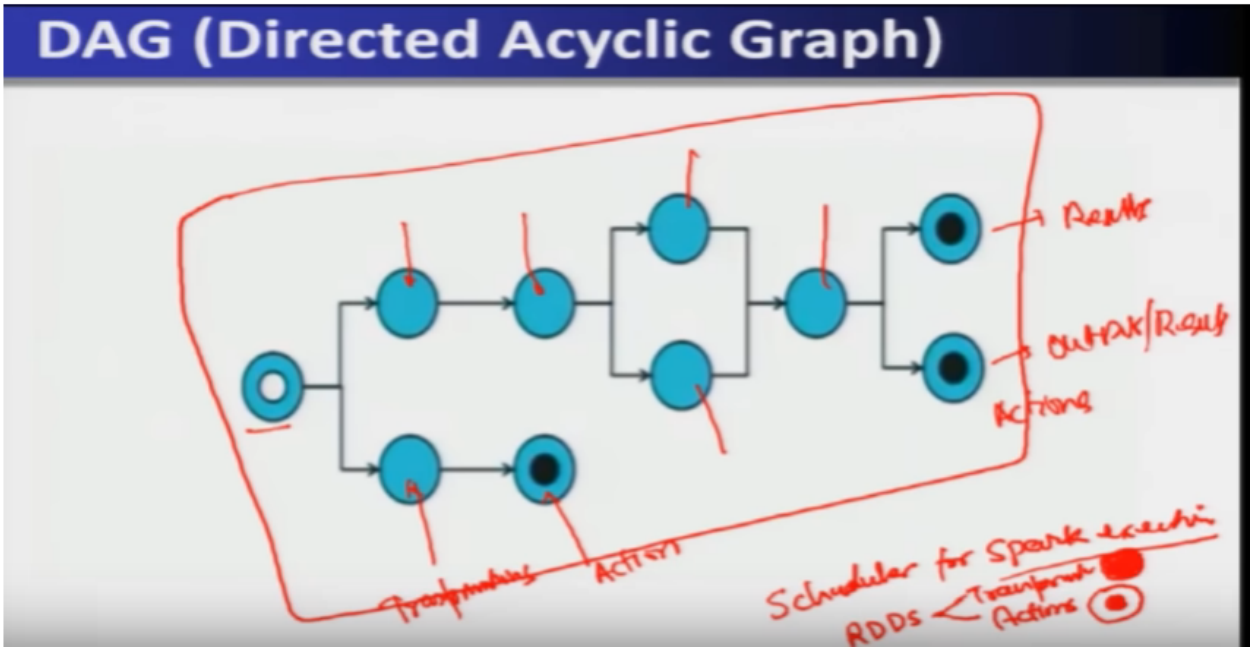
And then again, the driver will give further operations or the functions.

Refer slide time: (27:34)



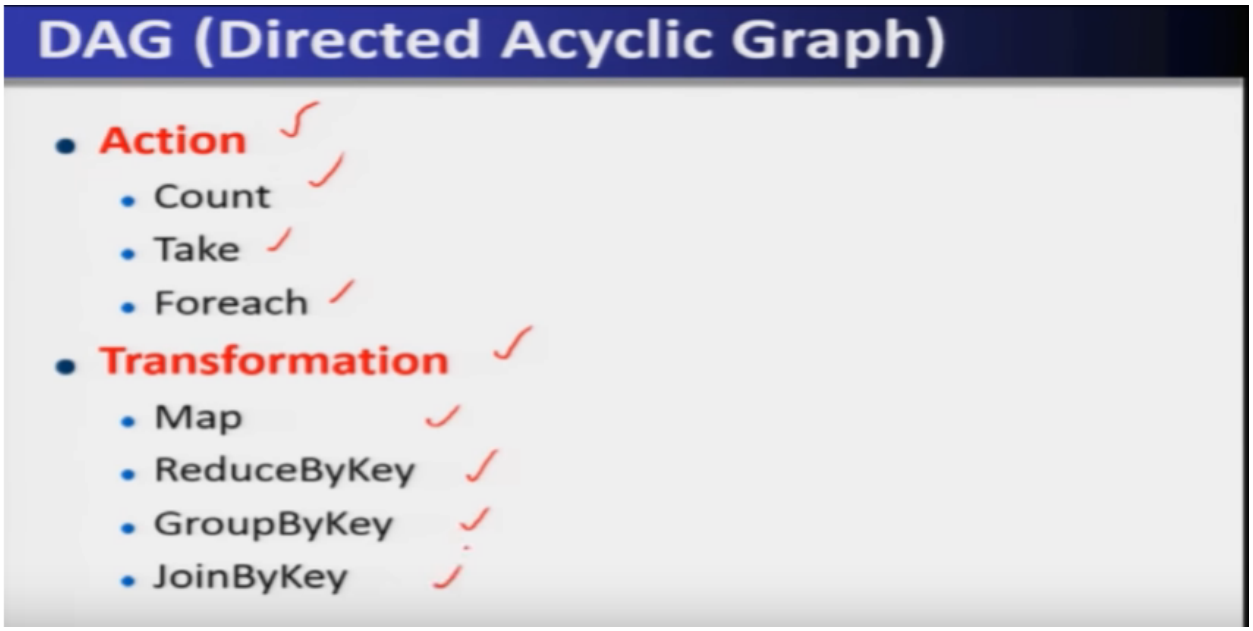
And these, particular functions are, used by the shuffle and again given back to the executors.

Refer slide time: (27:42)



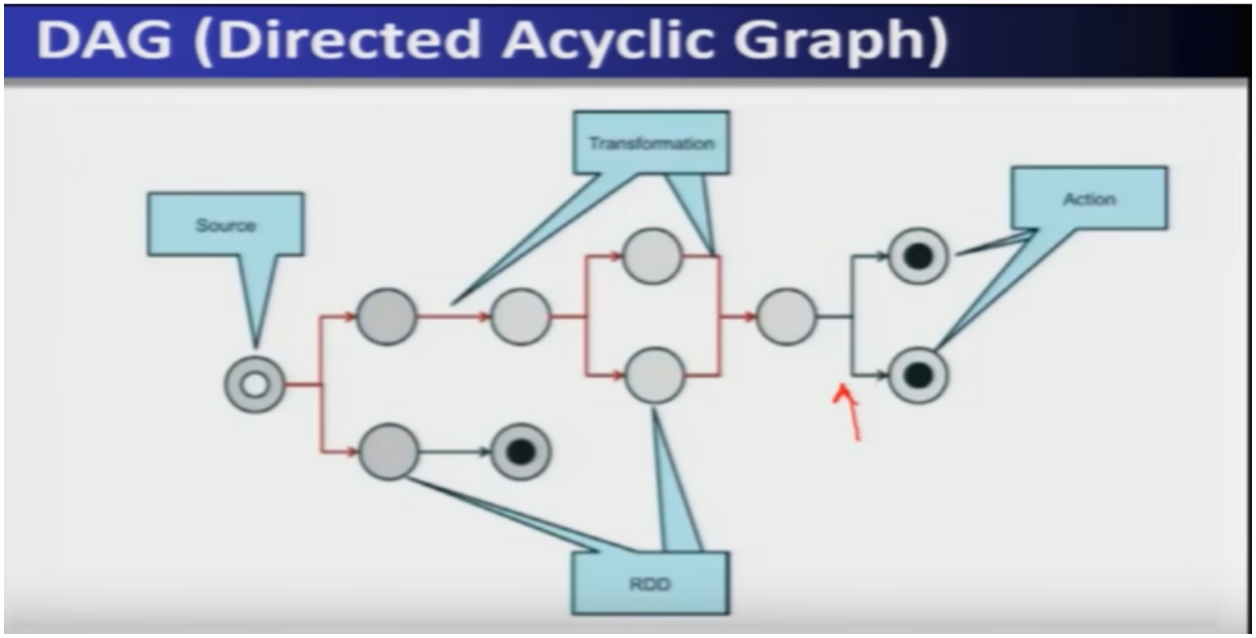
This all will be performed, the entire task operations, will be performed using, the directed acyclic graph. So directed acyclic graph is the Schuler, for the SPARC execution. So SPARC execution, as we have seen that RDDs, can be executed using, two operations. One is called, 'Transformations'. The other one is called, 'Actions'. So, in this particular RDD, we have shown you the actions, in the form of, the dotted circle and transformation in the form of a gray circle. so you can see here, that, this shows that, this is an RDD and from this, this particular RDD, is obtained using the transformations and further, this particular RDD is now giving performing the action part and this is also a transformation, transformation, transformation and these are the actions. So these actions will give the, output or the results, of the execution. This complete schedule is, available at the driver and using this particular, scheduler the driver in turn will supply, the different actions, different operations on RDDs, in the form of transformations and actions as, it is defined, in the directed acyclic graph scheduler.

Refer slide time: (29:52)

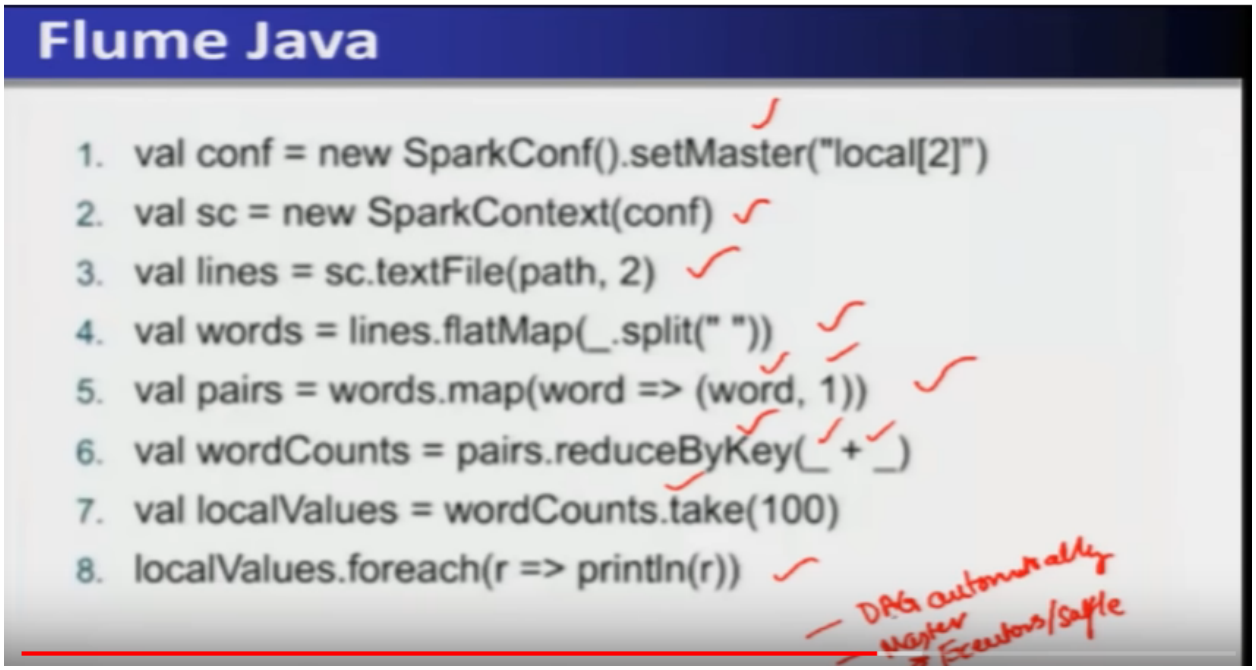


Therefore, this directed acyclic graph or a dag, will take either the actions or transformations. So actions include the, count, take, foreach and the transformation involves, the map, reduced by key, joined by key and group by key.

Refer slide time: (30:16)

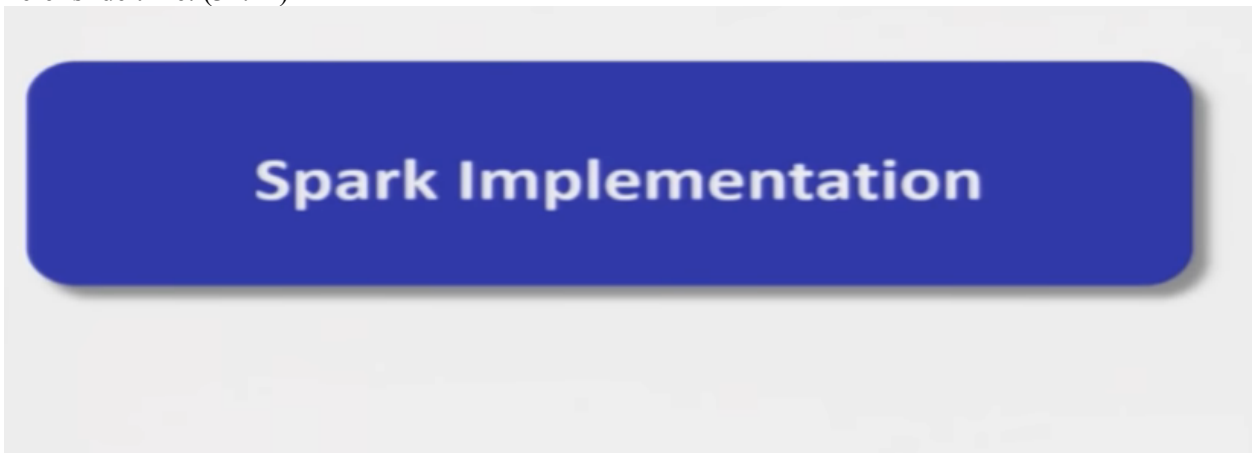


So these are all, a diagram I have already explained, that these are all RDDs and these are all transformations, the arrows are transformations, from one RDDs to another RDDs and if this is an, action. On these RDDs, it will be performed on the action.
Refer slide time: (30:44)



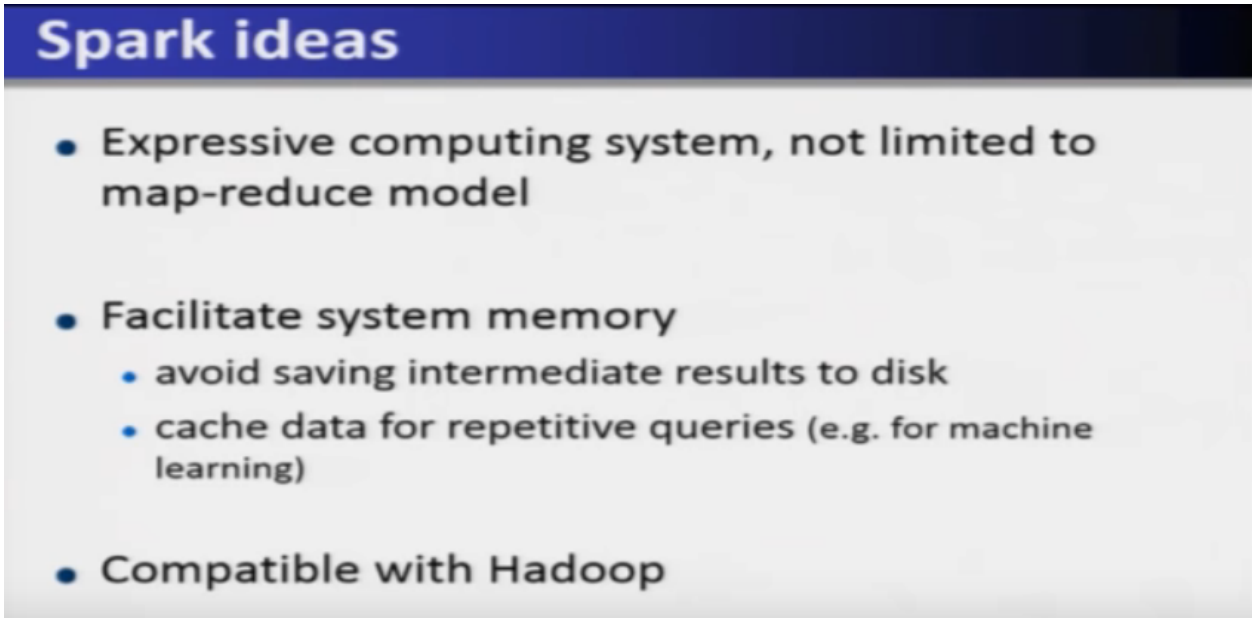
So let us see, a simple word count application, which is written in, a Spark, using flume Java. So here, we can see that, we have to set a master, which will, which will take care of running the entire dag scheduler. And now then, we have to create a Spark context and we have to, read the text file and then we have to do a flat map, which will split different words, which are separated by, the blank. after that, flat map then we have to, do the map operation, of a word count, which will omit, the word with and the value 1 .then we will perform the reduce by key, that means for, for, for a particular world, all such list of numbers or instances which is emitted by the map function, will be now, doing the summation of that. And then, the word count it will now take and then it will, print it for each value of key. The, the word count will be printed. So this way, this dag automatically once the program is made, the dag will be constructed, automatically and the dag will be given, to the master node. And the master in, in turn will communicate, with the executors and shuffle, for this execution in this way, that is performed in this dak way.

Refer slide time: (32:44)



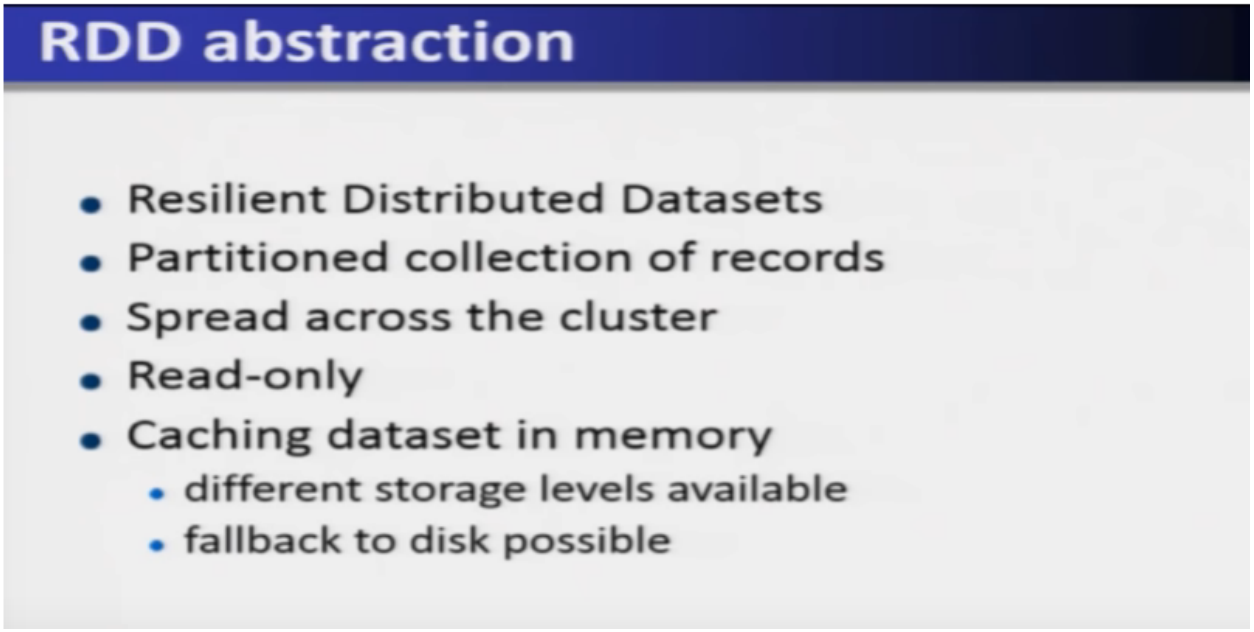
Let us, see the SPARC implementation, in more details.

Refer slide time: (32:46)



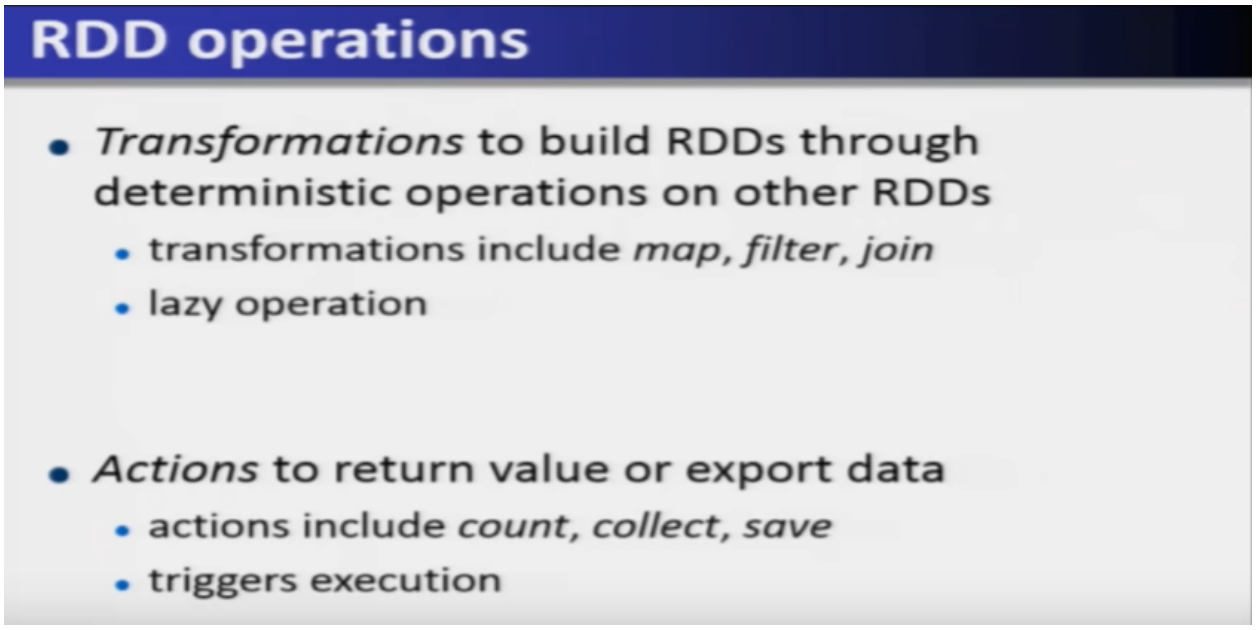
So, Spark ideas are an expressive computing system, which is not limited by, the Map Reduce model. That means, beyond Map Reduce also, the programming can be now done, in the SPARC. Now, this Spark will facilitate the system memory and it will avoid saving that immediate, results to the disk, it will cache for repeated, repetitive queries .that means, the output of the transform actions or the transformation ,will remain in the cache, so that, iterative applications can, can make use of ,this fast our efficient data sharing .

Refer slide time: (33:31)



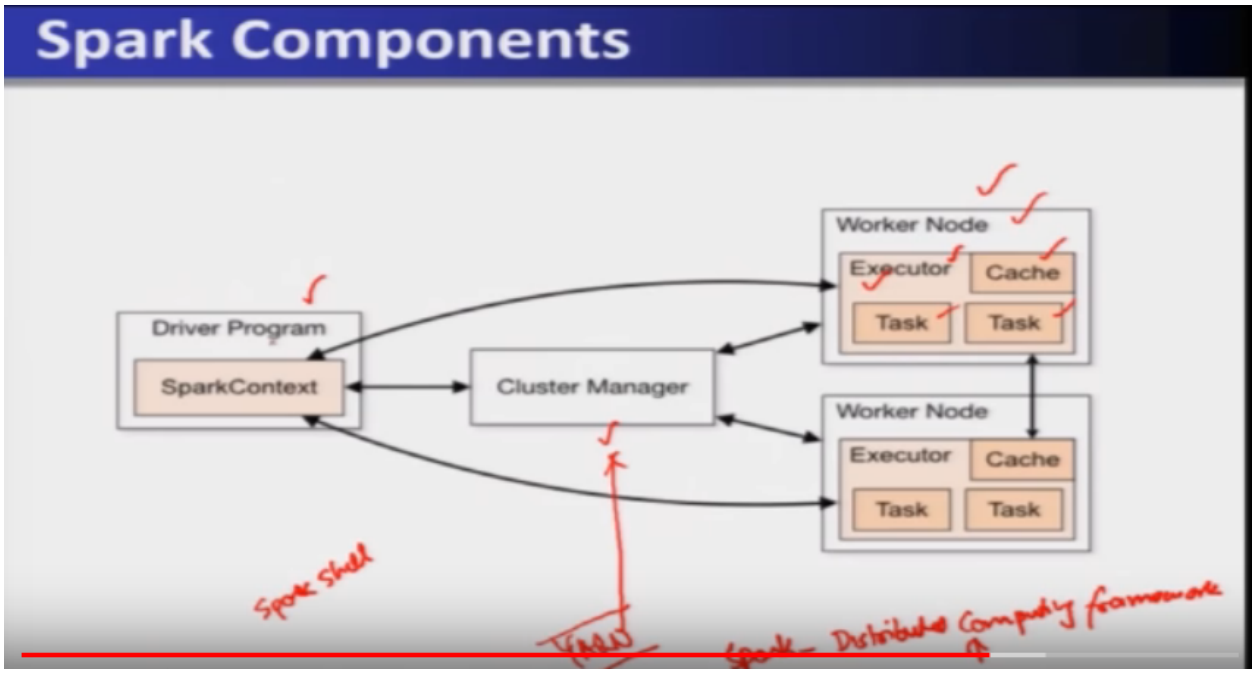
This Park is also compatible with, with the Hadoop system. RDDs is an abstraction as I told you, it's a resilient distributed datasets, a partition collection of record ,they are spread across the cluster, they are here only and caching data sets, in the possible, in memory and different storage levels are possible.

Refer slide time: (33:48)



As I told you that, the transformations and actions, there are two operations, RDD supports and transformations include map filter joint, they are lazy operations and actions, include the count collect sale and they are trigger executions. Spark components, let us go and discuss

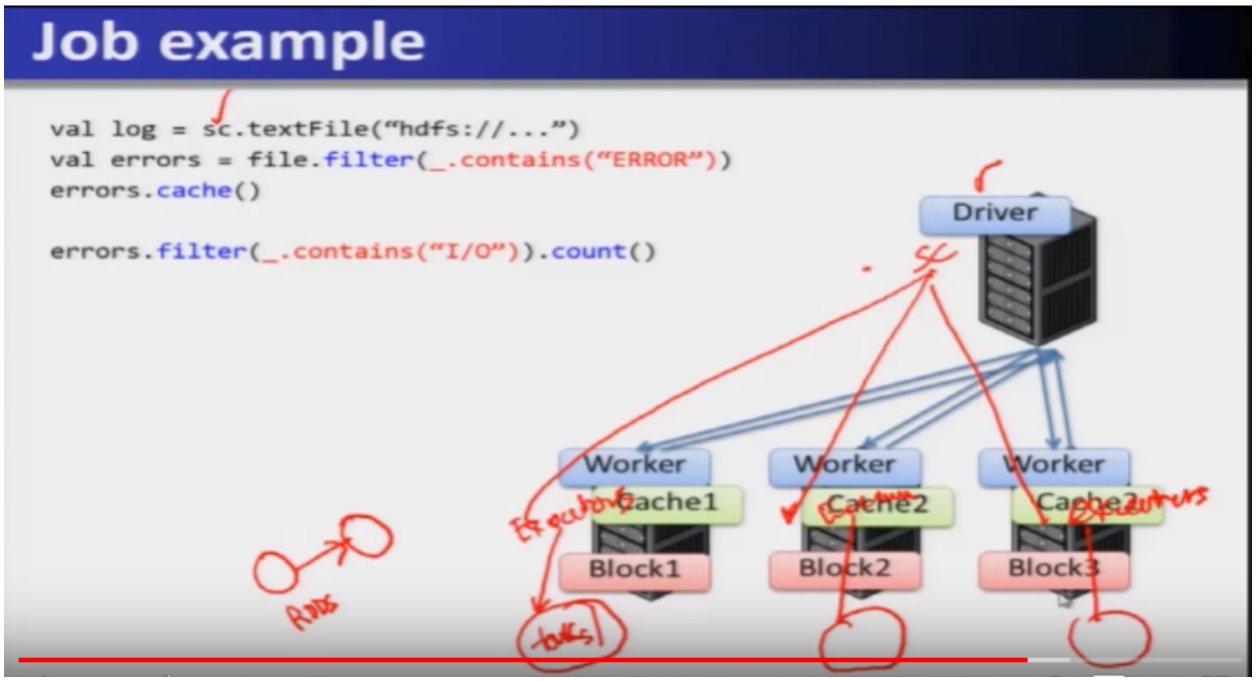
Refer slide time: (34:13)



The Spark components, in more details. So, as you know that, is Spark, is a distributed computing framework. So now, we will see here, what are the different components? Which together will form? The computational environment of the Spark execution. Now, we have seen, there is a cluster manager, there is a driver program, there is a worker nodes and within the worker nodes, you have the executors, and within the executors what are the different tasks? And what is the cash? All these different components together will form the distributed computing framework, which will give an efficient, execution, environment, for the Spark program or a Spark applications. So here, so within, a driver program, when whenever a Spark, shell is being prompt, that will be inside the driver program, will create the Spark context.

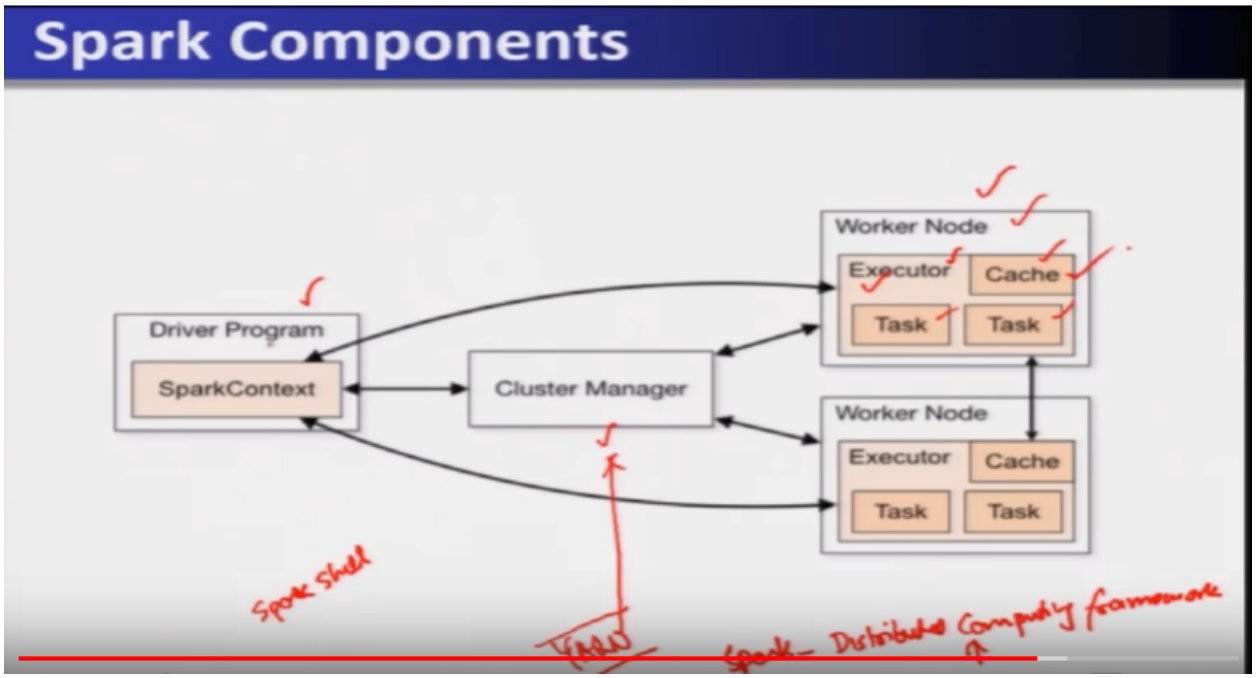
Now, executing the Spark context means that, it will communicate with the worker nodes, within the worker nodes the executors, so a Spark context will, create will, interact or communicate with the executors. And within the executors, the tasks will be executed. so executors within the executable, yes tasks will be executed and executors will be computed or will be executing on the worker nodes, so the driver program, then interacts with the cluster manager and cluster manager intern will interact with these worker nodes .so this all will happen, inside the Spark and Spark will dust the cluster manager. Now, there is an option in the Spark that instead of going through the cluster manager, you can also use the yarn and other, such resource manager and Scheduler. Now, let us see, what do you mean, by the, driver program, Spark context, server and cluster manager, worker node, executor, then tasks and through ,this to understand this.

Refer slide time: (36:56)



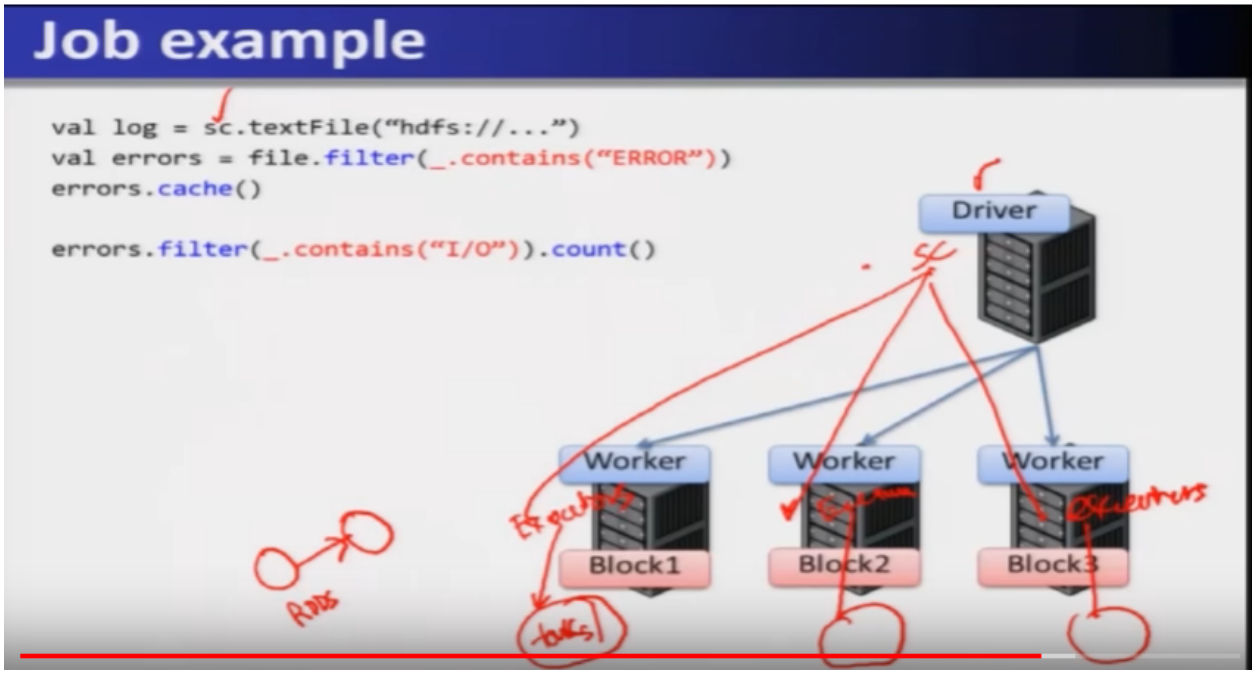
Let us see, a simple application. Now, we have seen here, the driver program and this driver program will create a Spark context, it will create a Spark context SC. and this in turn will now, communicate with the executors, which are running inside, which are running inside. So, so this particular driver program, intern knows the, the different worker program communication and the Spark context, will now communicate to the executors. And these executors in turn, will communicate or will, will execute the tasks. So these tasks are, nothing but, the various transformations, the RDDs, through the dag, they are being transformed and they are done through the tasks. So different executors various tasks are being created and executing. So this will create the job execution and let us go back and, and see that, these

Refer slide time: (38:18)



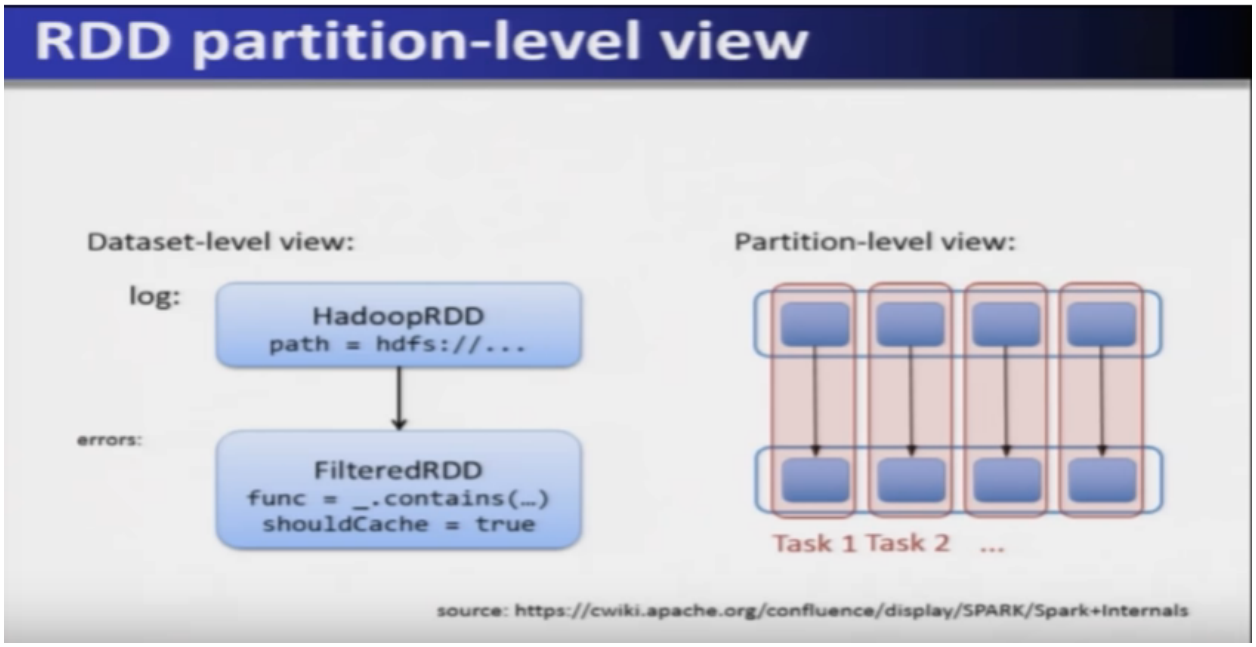
So this particular way, the driver program, will execute, the dag and server Spark context SC. will be created which in turn will communicate, inside the worker node with the executors and these executors in turn will execute, various, transformations and actions and the result will be remained in the cache.

Refer slide time: (38:46)



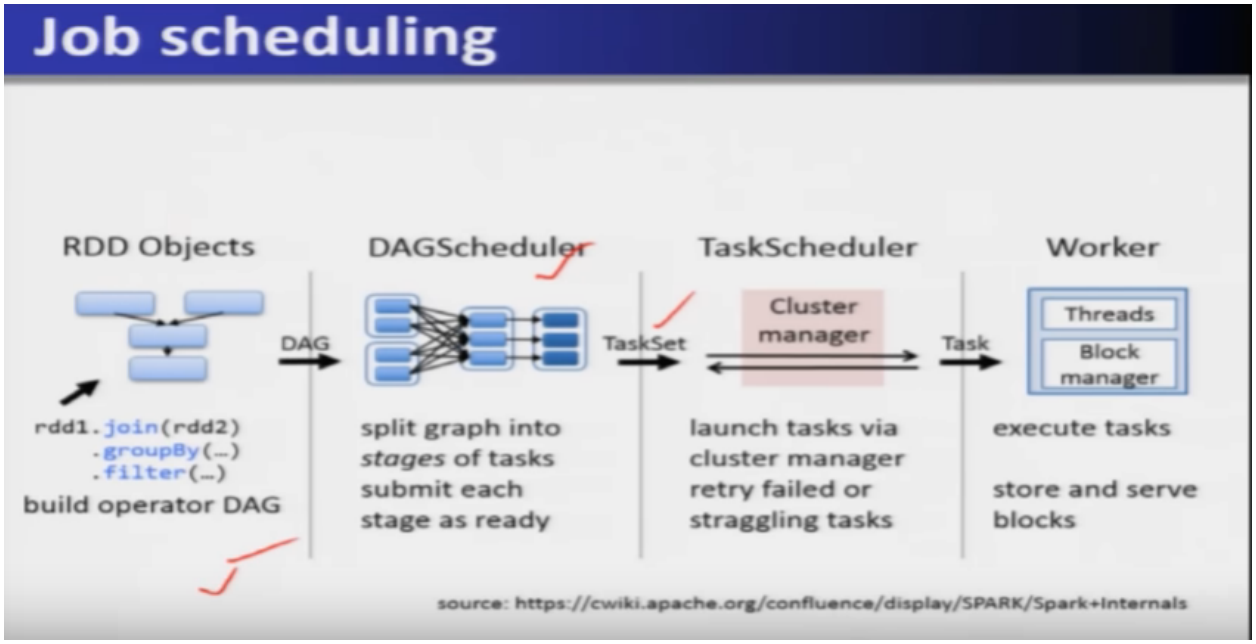
So that was, about the Spark components.

Refer slide time: (38:49)



So we have seen that, in this manner, the operation of Spark is being performed. Now, another view of partition level view, we can see here that, the partitioning so that means, RDDs are partitioned

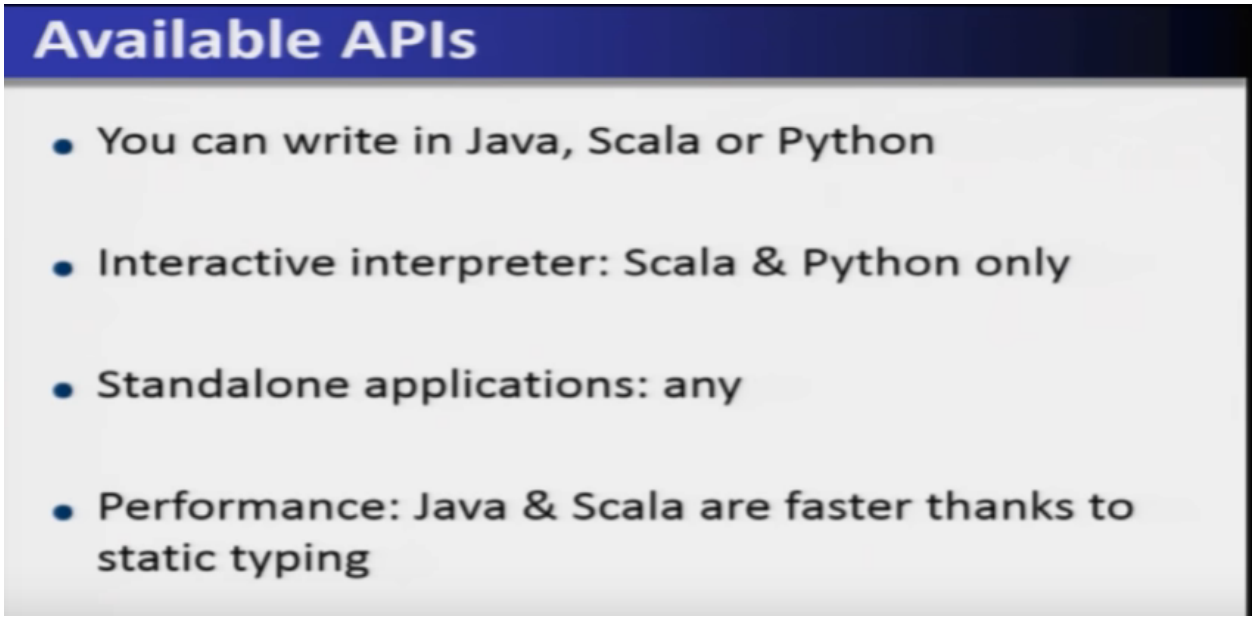
Refer slide time: (39:09)



And different tasks are being executed. similarly job scheduling, that means, once an RDD, is that means operations are, given automatically it will build, the dag and dag will be given to the dag scheduler and

that, dag scheduler will split the graph into the stages of, task and submit each stage as it is ready. So task set is created and given to the task a scheduler and as far as the cluster manager, is concerned it launches, the tasks, via the cluster manager and retry the field or straggling task. And this task is, given to the worker that we have seen in the previous slide and this particular workers will create the thread and execute them, there.

Refer slide time: (39:59)



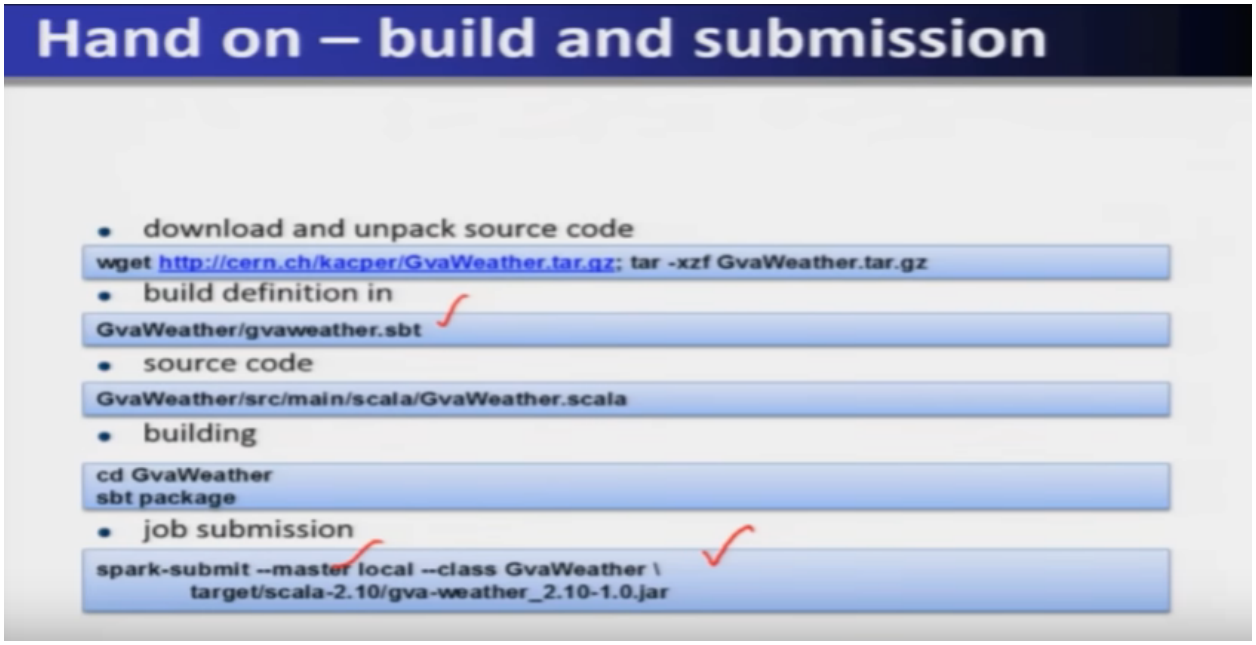
There are different APIs, which are available and you can write, these APIs is using Java program, scala or a Python. There is also an interactive interpreter: available, access through, the scalar and Python. Standby applications are, there are many applications and performance: if we see that Java and C are faster and thanks to the static typing

.

Refer slide time: (40:26)

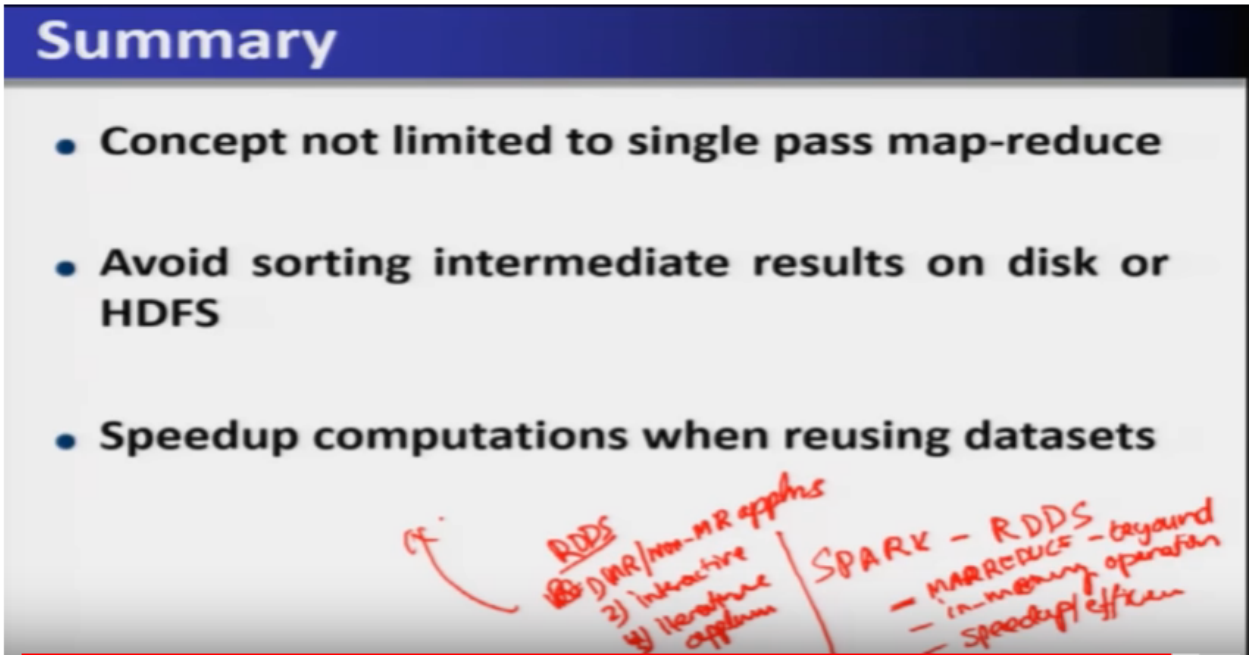
Now let us see, the hands-on session, how we can perform, using Spark. So a S Spark, we can run, as a scalar, so Spark shall, will be created.

Refer slide time: (40:44)



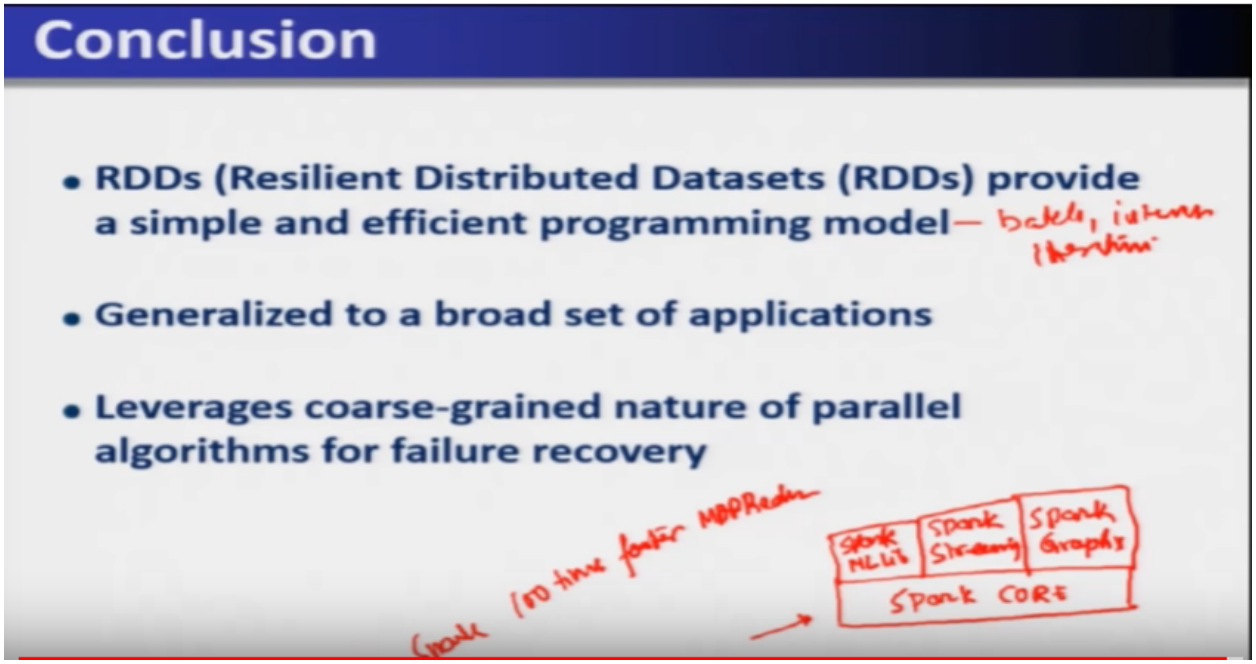
And we can download, the ,the, the file, that data set file and then, it can be, built using ,the package and then this particular task or a data file can be submitted ,to the, to the master node of that Spark system. So, so directly Spark, can store it into the system and now it can perform, various operations, on this particular data set, using a scalar program.

Refer slide time: (41:20)



Now summary, the concept of, of Map Reduce are ,limited to the single path Map Reduce ,is basically limiting various other applications .and this particular, concept is avoiding, the sorting intermediate results, storing intermediate results on the disk or on HDFS. And also, speed-up computations are required, when reusing the datasets. and all these features are available, as part of ,this part that we have seen using RDDs. So using RDDs, now Spark provides, the not only Map Reduce, operations, beyond Map Reduce, it can also use .second thing is, it can be in memory, operations, not necessarily to be stored, in SDFS in to store the intermediate results. So this way of in-memory computations, will make the speed-up and brings about the efficiency, data sharing across different iterations. So iterative and interactive applications both are, easily supported and Map Reduce and non Map Reduce applications are also supported, by the Spark system. So all this is possible, with the help of RDDs and their operations. so we have seen that ,now ,the RDDs are saw a Spark is very much required and all the drawbacks of Map Reduce and Hadoop, is now not there with the SPARK and therefore the Spark now has, now ,various new applications. For example, the Spark system will, Spark.

Refer slide time: (43:49)



So the Spark is a core, as a core, can be used for building different applications, such as, Spark MLlib, that is the machine learning or the Spark, then Spark streaming ,that is the real-time applications, over the Spark and Spark graphics, the graph computation or the Spark. So, now Spark can use SD, SDFS or may not use SDFS, Spark is independent. Therefore, let us, conclude this discussion, that RDD, is resilient distributed datasets, will provide a simple and efficient programming model, for different supporting various applications, which are the batch and interactive and iterative applications, all are supported using this concept, which is called,' RDDs'. Now, this is generalized, to a broad set of applications. And it will, leverage the coarse-grained nature of parallel algorithm, for fault recovery. So that is why, this is a hundred times, faster, compared to the, traditional Map Reduce. So, Spark is 100 times faster, that is what is compared, with the performance, by the Spark, production clusters. Thank you.