SWE404/DMT413
BIG DATA ANALYTICS

Lecture 5: Spark I

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

Outline

- Overview of Apache Spark
- Spark Basics

OVERVIEW OF APACHE SPARK

The Demand of Real Time Analytics

- Real time processing of big data has increasing demand in every aspect of our lives.
- There is a huge amount of data that the internet world necessitates to process *in seconds*.
  - Waiting for accumulating data with batch processing = losing money.

The Demand of Real Time Analytics

- **Government**
  - Government agencies perform real time analysis mostly in the field of national security.
    - E.g. Transportation, surveillance, anti-terrorist.
  - Countries need to continuously keep a track of all the military and police agencies for updates regarding threats to security.

Government

Image source: https://www.edureka.co/blog/spark-tutorial/

The Demand of Real Time Analytics

- **Banking**
  - Banking transacts with almost all of the world's money.
  - It becomes very important to ensure fault tolerant transactions across the whole system.
    - Nobody is willing to see money lost due to system failure.
  - Fraud detection is made possible through real-time analytics in banking.

Banking

Image source: https://www.edureka.co/blog/spark-tutorial/

The Demand of Real Time Analytics

- **Telecommunications**
  - Services like calls, video chats and streaming use real-time analysis to reduce customer churn and stay ahead of the competition.
  - They also extract measurements of jitter and delay in mobile networks to improve customer experiences.

Telecommunications

Image source: https://www.edureka.co/blog/spark-tutorial/

The Demand of Real Time Analytics

- **Stock Market**
  - Stockbrokers use real-time analytics to predict the movement of stock portfolios.
  - Companies re-think their business model after using real-time analytics to analyze the market demand for their brand.

Stock Market

Image source: https://www.edureka.co/blog/spark-tutorial/

Limitations of Hadoop

- We have introduced lots of benefit of Hadoop for big data analytics.
- Why do we need Spark when we have a so powerful Hadoop already?

Limitations of Hadoop

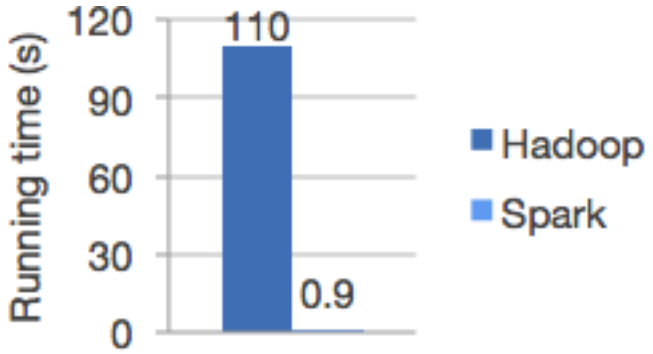
- Lack of long-lived MapReduce jobs.
  - MapReduce jobs are typically short-lived.
  - One would have to create fresh MapReduce jobs for every iteration in a lot of these classes of computations.
- Inability to store a working set of data in memory.
  - The results of every iteration would get stored in HDFS.
    - HDFS -> Map -> Local FS -> Network -> Reduce -> HDFS -> Map ->…
  - The next iteration would need data to be initialized, or read, from HDFS to memory.

Limitations of Hadoop

- The data processing model could be not flexible enough for many application.
  - A too strict programming model.
    - You must follow map/reduce programming style, which limits lots of applications.
  - Must be stateless.
    - Each node has no idea of the job on other nodes.
- Situations where MR is not efficient:
  - Long pipelines sharing data.
  - Interactive applications.
  - Streaming applications.
  - Iterative algorithms (optimization problems).

What is 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.



Logistic regression in Hadoop and Spark

Image source: https://www.techenablement.com/apache-spark-claims-10x-100x-faster-hadoop-mapreduce/

Development History of Spark

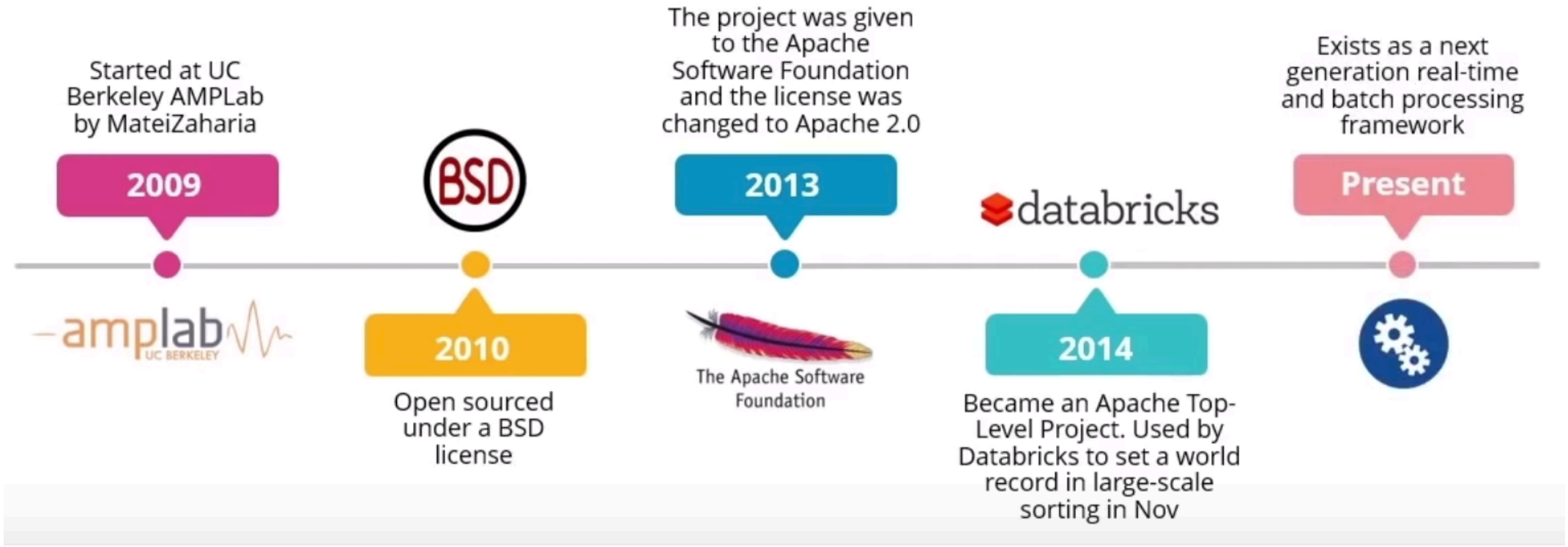


Image source: http://www.igfasouza.com/blog/what-is-apache-spark/

Record Set by Spark in Large-Scale Sorting

- How fast a system can sort 100 TB of data (1 trillion records).
- Compared with Hadoop, 3x faster with 10x less nodes.

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

2014, 4.27 TB/min

Apache Spark
100 TB in 1,406 seconds
207 Amazon EC2 i2.8xlarge nodes x
(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)
Reynold Xin, Parviz Deyhim, Xiangrui Meng,
Ali Ghodsi, Matei Zaharia
Databricks

Features of Apache Spark

- **Polyglot**
  - Provide high-level APIs in Java, Scala, Python and R. Spark code can be written in any of these four languages.
  - In this course, we use Python with PySpark for instructional demonstration. However, Scala is usually adopted for industrial big data development.
- **Speed**
  - Run up to 100 times faster than Hadoop MapReduce for large-scale data processing.
  - Able to achieve this speed through controlled partitioning.
- **Multiple Formats**
  - Support multiple data sources such as Parquet, JSON, Hive and Cassandra apart from the usual formats such as text files, CSV and RDBMS tables.

Features of Apache Spark

- **Lazy Evaluation**
  - Delay its evaluation till it is absolutely necessary. This is one of the key factors contributing to its speed.
- **Real Time Computation**
  - Computation is real-time and has low latency because of its in-memory computation.
- **Machine Learning**
  - Spark's MLlib is the machine learning component which is handy when it comes to big data processing.
  - It eradicates the need to use multiple tools, one for processing and one for machine learning.

Features of Apache Spark

- **Hadoop Integration**
  - Provide smooth compatibility with Hadoop. This is a boon for all the Big Data engineers who started their careers with Hadoop.
    - **HDFS**: Spark can run on top of HDFS to leverage the distributed replicated storage.
    - **MapReduce**: Spark can be used along with MapReduce in the same Hadoop cluster or replace MapReduce as a more advanced processing framework.
    - **YARN**: Spark applications can be made to run on YARN.
    - **Batch & Real Time Processing**: MapReduce and Spark can be used together where MapReduce is used for batch processing and Spark for real-time processing.

Spark Components

- Spark components are what make Apache Spark fast and reliable.
- A lot of these Spark components were built to resolve the problems when using Hadoop MapReduce.
- Apache Spark has the following components:
  - Spark Core
  - Spark Streaming
  - Spark SQL
  - GraphX
  - MLlib

Spark Ecosystem

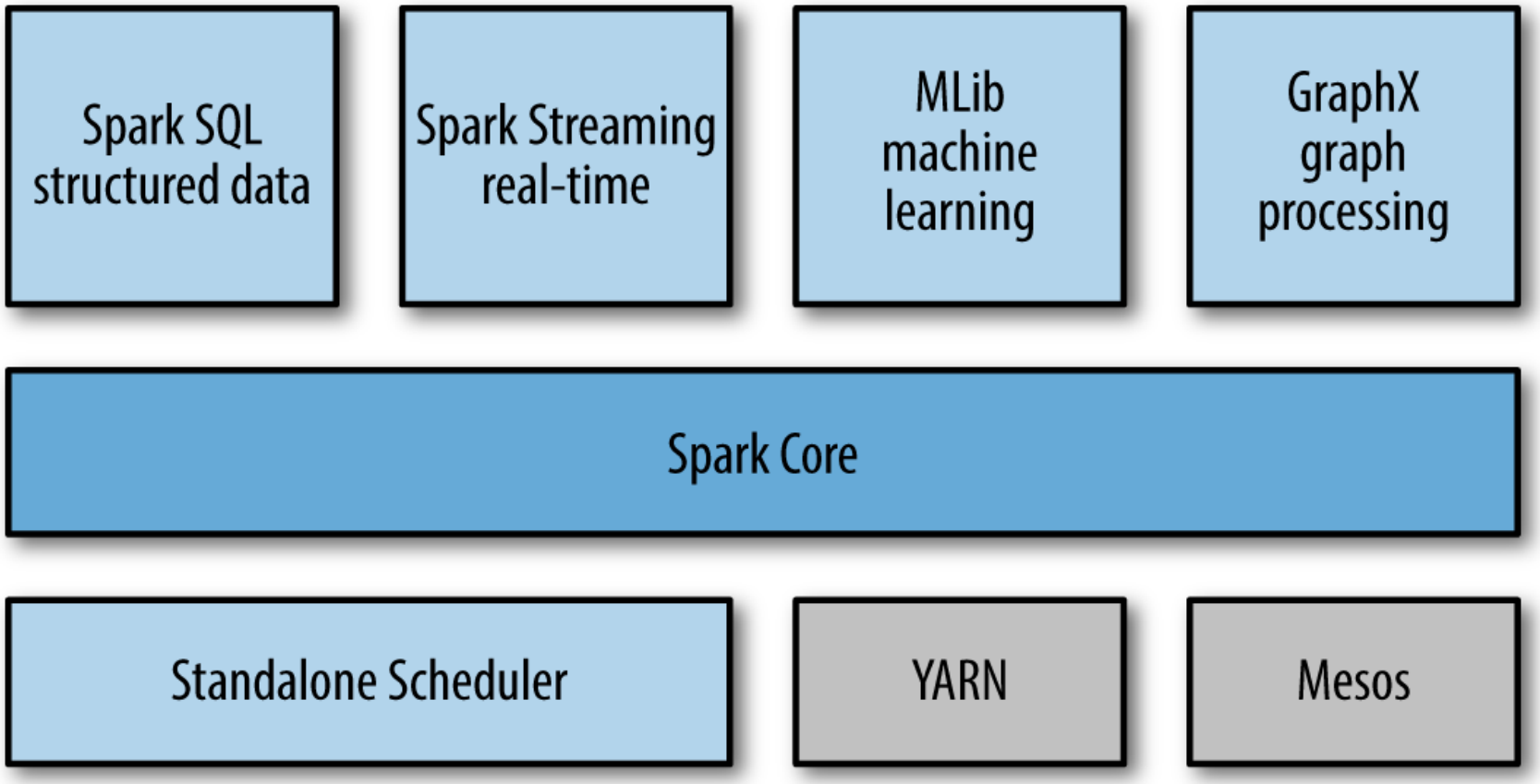


Image source: https://events.prace-ri.eu/event/850/sessions/2616/attachments/955/1531/Spark_Introduction.pdf

Spark Core

- Spark Core is the base engine for large-scale parallel and distributed data processing.
- It is responsible for:
  - Memory management and fault recovery.
  - Scheduling, distributing and monitoring jobs on a cluster.
  - Interacting with storage systems.

Spark Streaming

- *Spark Streaming* is a useful addition to the core Spark API and used to process real-time streaming data.
  - E.g. Twitter, stock market, geographical systems.
- It enables high-throughput and fault-tolerant stream processing of live data streams.

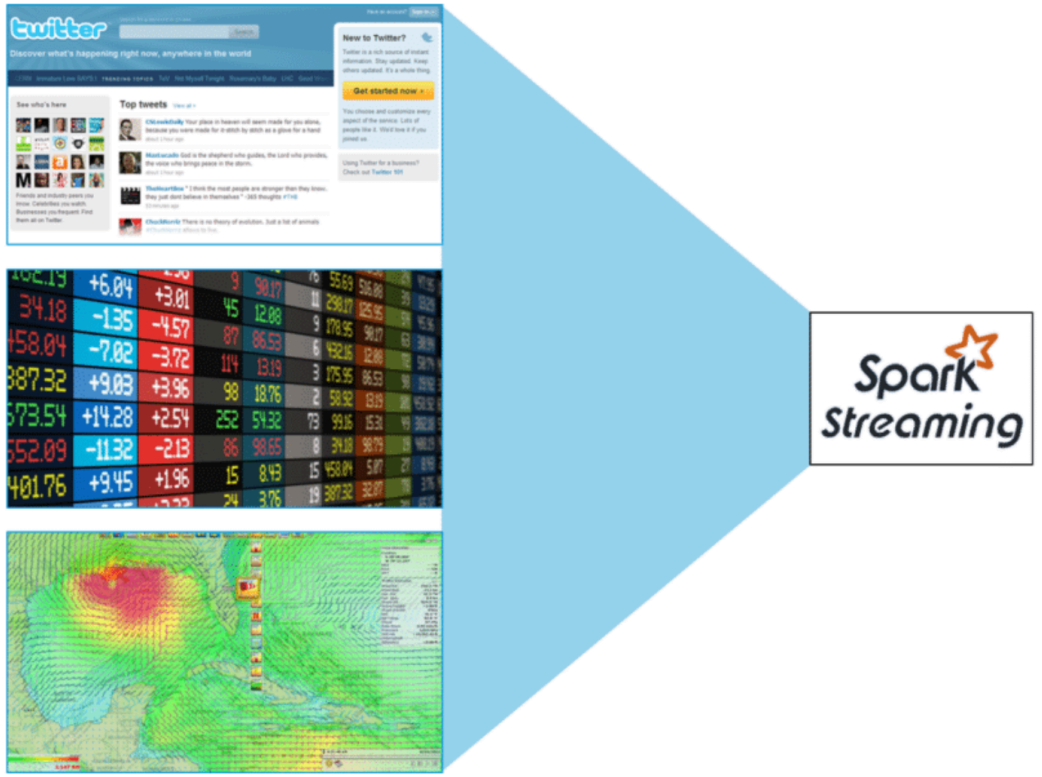


Image source: https://www.edureka.co/blog/spark-tutorial/

Spark SQL

- Spark SQL integrates relational processing with Spark's functional programming API.
  - For structured data.
- It provides support for various data sources including Hive tables, Parquet, and JSON.
- It allows developers to intermix SQL queries with the programmatic data manipulations

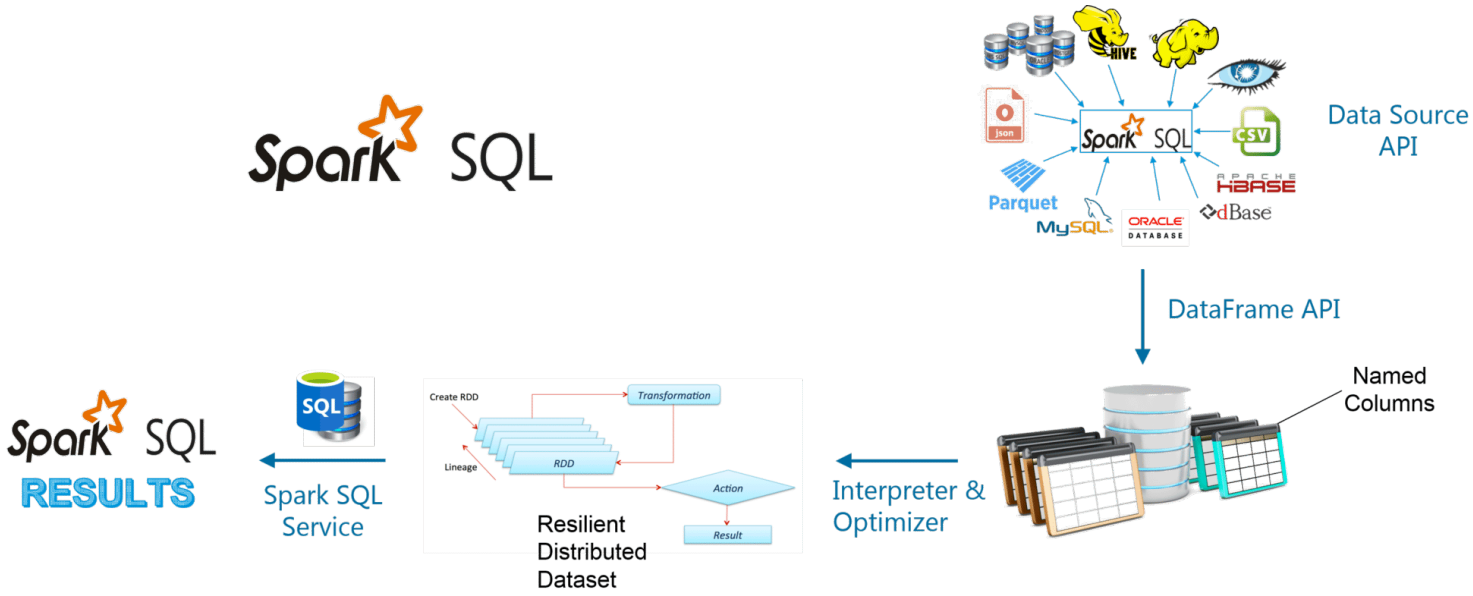
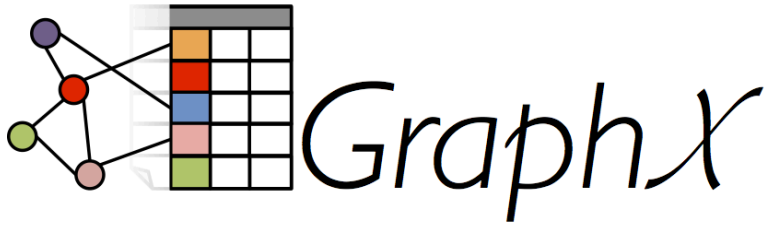


Figure: *The flow diagram represents a Spark SQL process using all the four libraries in sequence*

GraphX

- A library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations.
- Provides various operators for manipulating graphs and a library of common graph algorithms.



MLlib

- A distributed machine learning framework on top of Spark core
  - 9 times faster than Apache Mahout (2nd generation ML library for Hadoop)
- Provides multiple types of machine learning algorithms
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering

Image source: https://towardsdatascience.com/deduplication-using-sparks-mllib-4a08f65e5ab9?gi=30ecced57314

SPARK BASICS

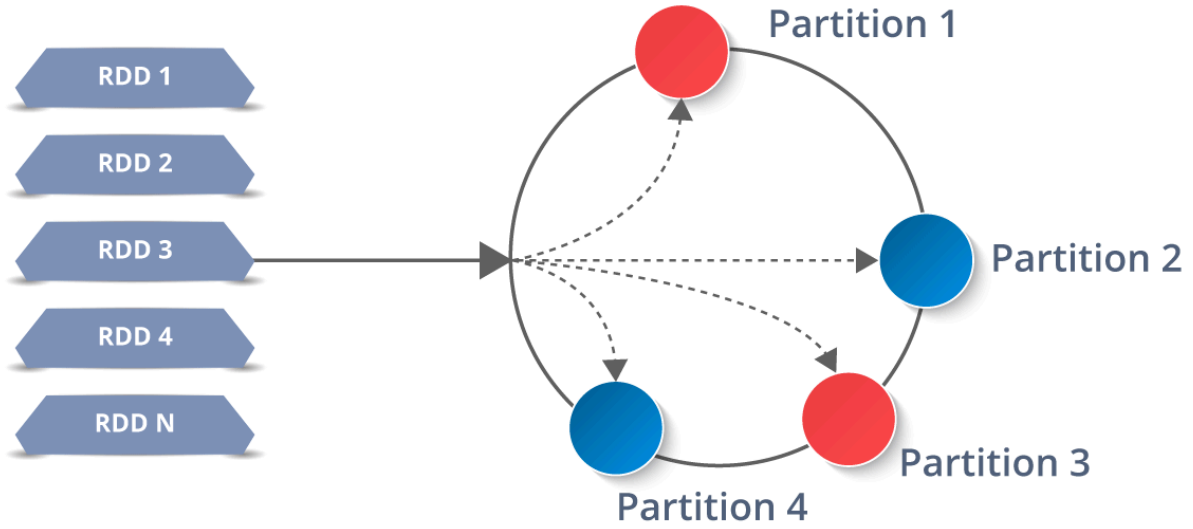
Spark Architecture

A *loosely coupled* system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components.

- Apache Spark has a well-defined layered architecture where all the spark components and layers are loosely coupled.
- This architecture is further integrated with various extensions and libraries.
- Apache Spark Architecture is based on two main abstractions:
  - Resilient Distributed Dataset (RDD)
  - Directed Acyclic Graph (DAG)

Resilient Distributed Dataset (RDD)

- 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.



one task per partition

Imgae source: https://www.edureka.co/blog/spark-architecture/#Spark

RDD Features

- **Fault tolerance**: RDDs are able to recover quickly from any issues as the same data chunks are replicated across multiple executor nodes.
  - Even if one executor node fails, another will still process the data.
- **Immutability**: Data stored in the RDD is *immutable (read-only)*.
  - You cannot edit the data which is present in the RDD. But, you can create new RDDs by performing transformations on the existing RDDs.
- **In-memory Computation**: RDD stores any immediate data that is generated in the memory (RAM) than on the disk so that it provides faster access.
  - Only in rare situations, where the data size is growing beyond the capacity, is it written to disk.

RDD Features

- **Caching**: RDD in Spark can be cached and used again for future transformations, which is a huge benefit for users.
- **Partitioning**: Partitions can be done on any existing RDDs to create logical parts that are mutable.

RDD and HDFS

- Both of them distribute data across the nodes for fault tolerant.
- However, they address different issues:
  - RDDs are about distributing *computation* and handling *computation* failures.
  - HDFS is about distributing *storage* and handling *storage* failures.

Workflow of RDD

- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
  - Same, you don't have to worry about the distribution, because Spark takes care of that.
- There are two ways to create RDDs:
  - Parallelize an existing collection in your driver program.
  - Reference a dataset in an external storage system, such as HDFS, HBase, etc.
  - Creating RDD from already existing RDDs.

Workflow of RDD

With RDDs, you can perform two types of operations:

- **Transformations**
  - Functions which accept existing RDDs as the input and outputs one or more RDDs.
    - Provide RDD values.
  - A code block is shipped from the driver program to multiple remote worker server, which hold a partition of the RDD.
  - The result of a transformation is a brand new RDD (the original RDD is not modified).
- **Actions:**
  - Functions which return the end result of RDD computations.
    - Provide non-RDD values.
  - Dump the RDD into a storage, or return its value data to the driver program.

Workflow of RDD

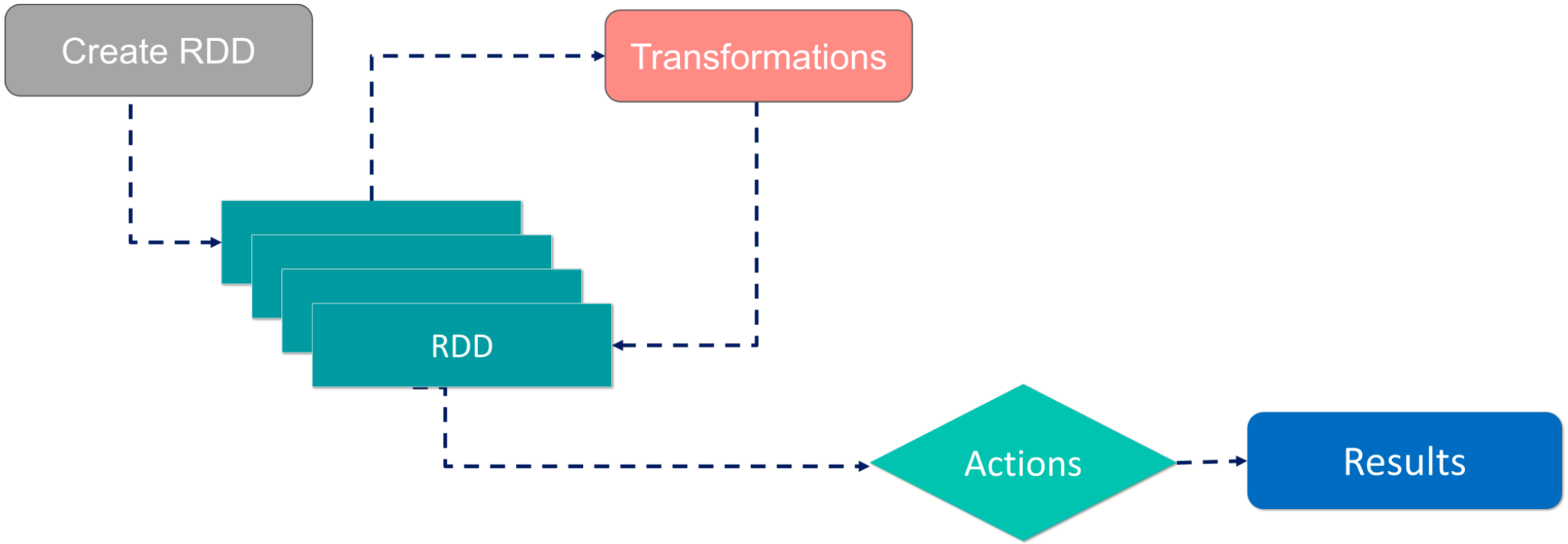
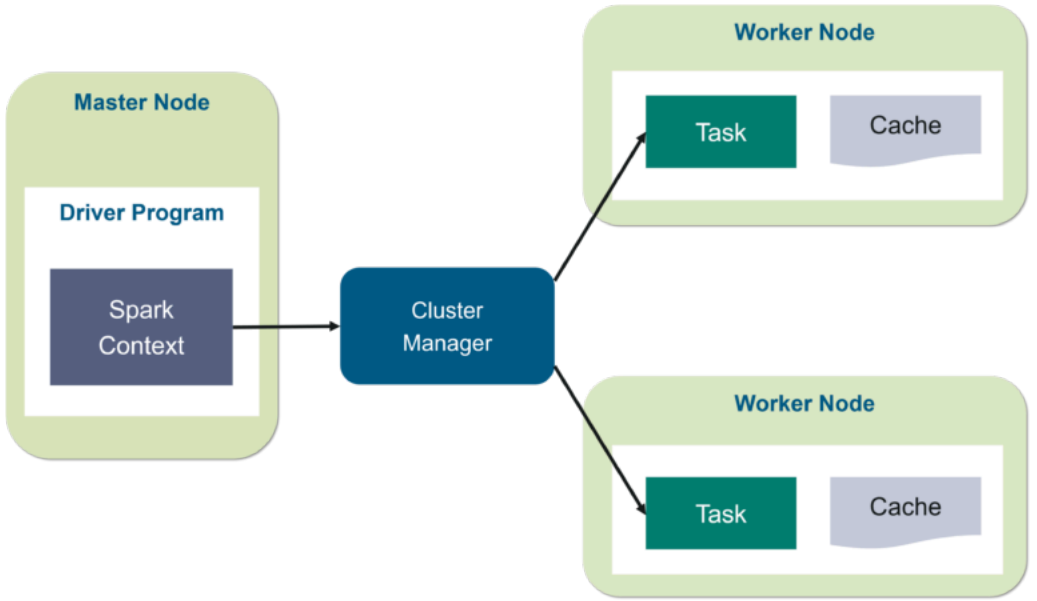


Image source: https://www.edureka.co/blog/spark-architecture/#Spark

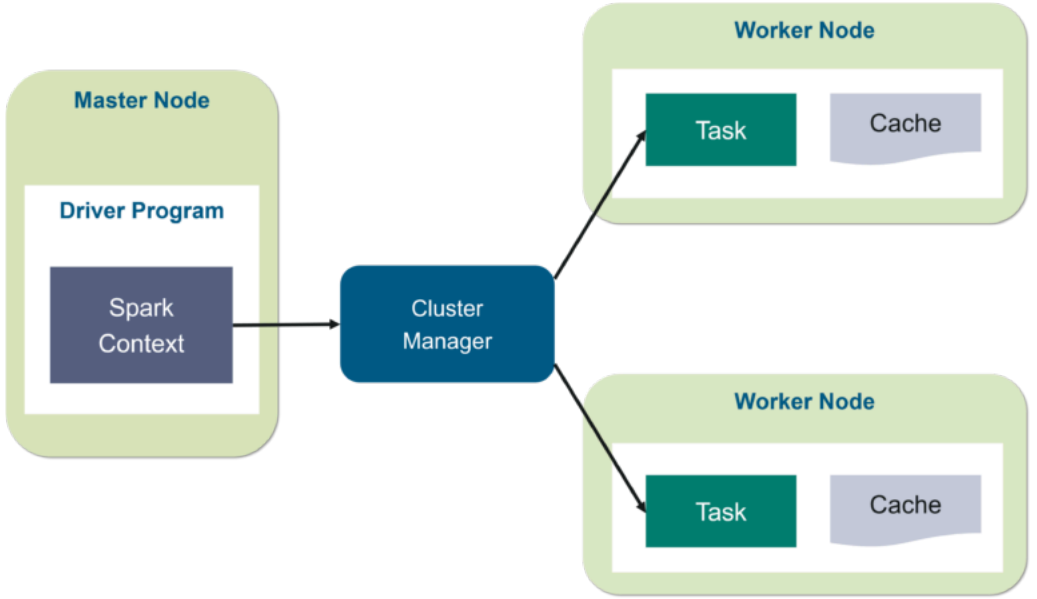
Spark Architecture

- In the master node, *Driver Program* calls the main program of an application and also creates the *Spark Context*.
  - Spark Context is like a gateway to all the Spark functionalities.
  - It is similar to database connection.
  - Any executed command in database goes through the database connection. Likewise, anything on Spark goes through Spark Context.



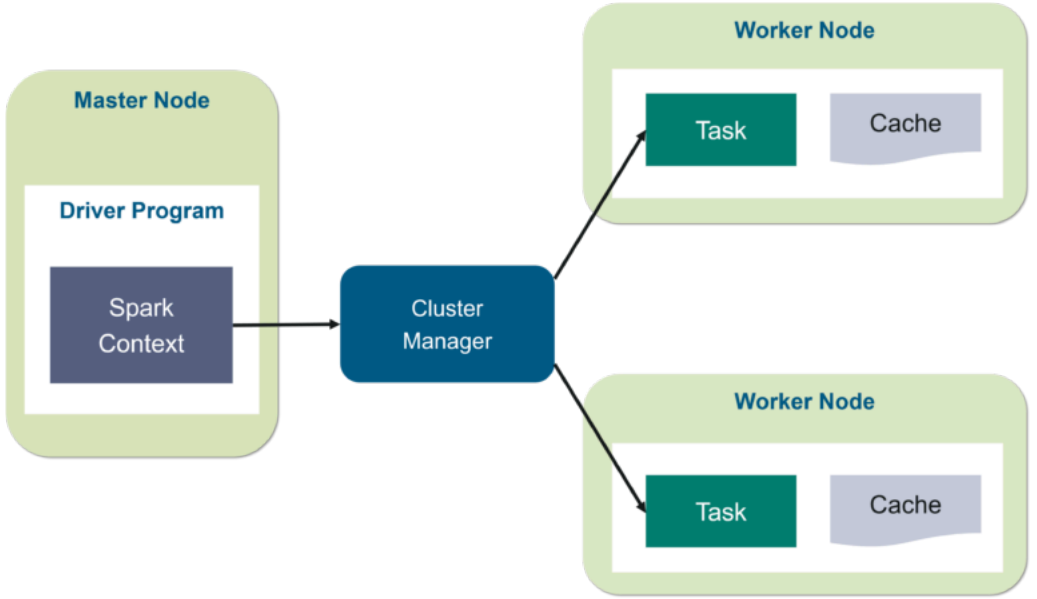
Spark Architecture

- Spark Context works with the *cluster manager* to manage various jobs.
- A job is split into multiple tasks which are distributed over the worker node.
- Anytime an RDD is created in Spark Context, it can be distributed across various nodes and can be cached there.



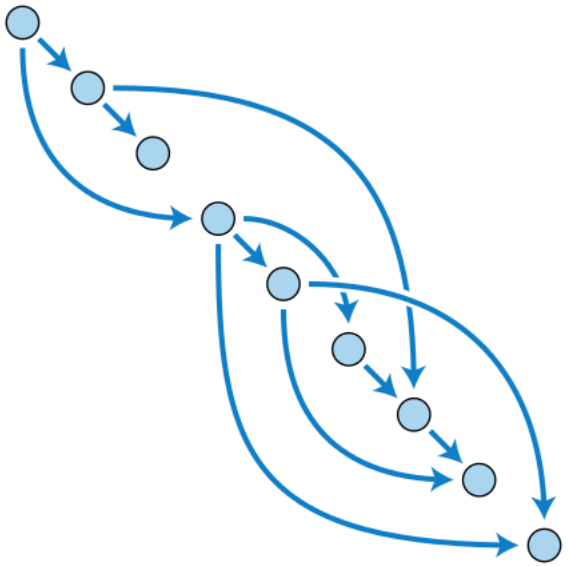
Spark Architecture

- *Worker nodes* are the slave nodes for executing the tasks.
- Worker nodes execute the tasks on the partitioned RDDs assigned by the Cluster Manager and returns it back to the Spark Context.
- Executor is responsible for the execution of these tasks.
  - Lifetime of executors is same as that of the Spark Application.
  - If you want to increase the performance of the system, you can increase the number of workers so that the jobs can be divided into more logical portions.



Directed Acyclic Graph (DAG)

- DAG is a finite directed graph with no directed cycles.
  - It consists of finitely many *vertices* and *edges*.
  - *Directed*: each edge point from one vertex to another.
  - *Acyclic*: there is no way to start at any vertex v and follow a consistently-directed sequence of edges that eventually loops back to v again.
- Equivalently, a DAG is a sequence of the vertices such that every edge is directed from earlier to later in the sequence.



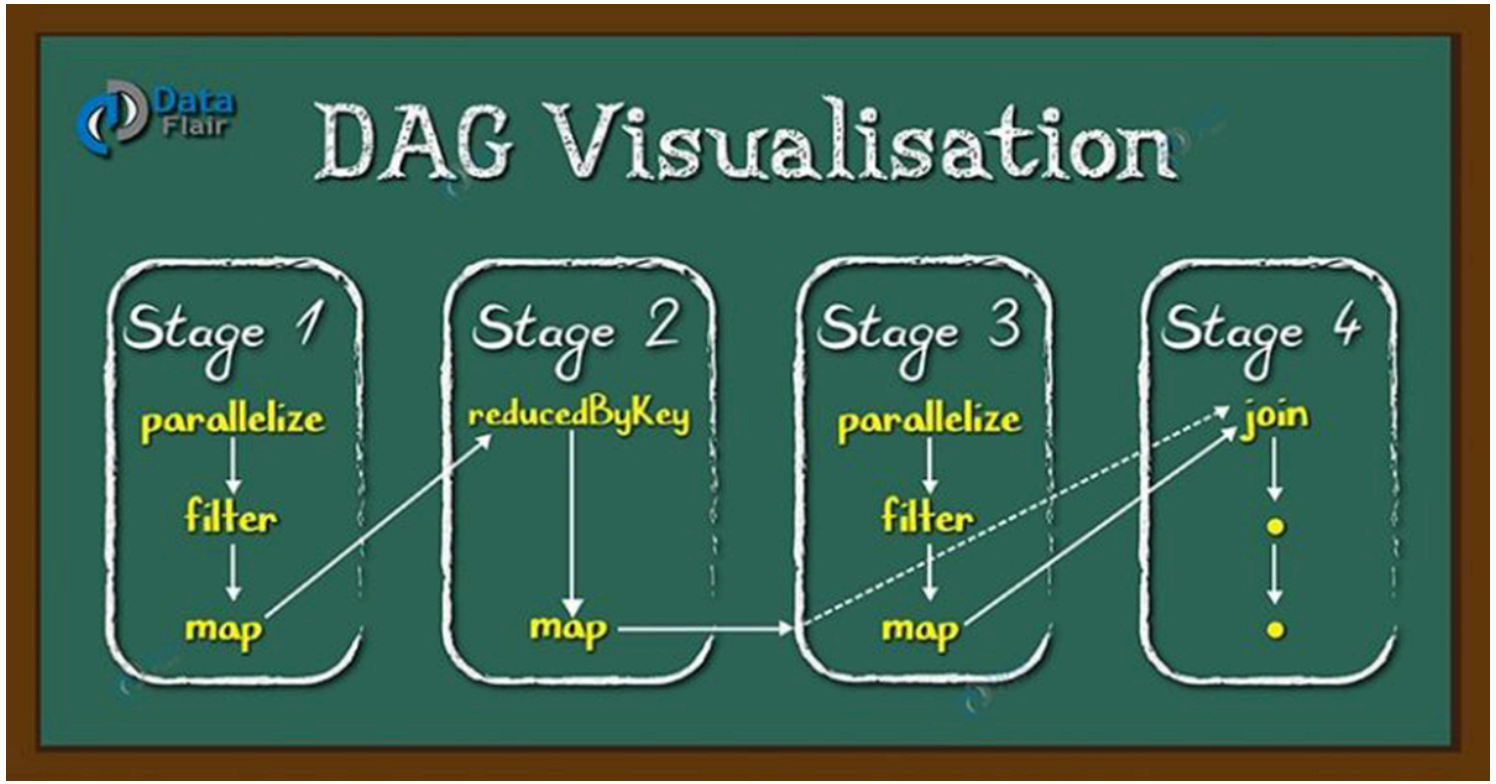
A DAG

Image source: https://en.wikipedia.org/wiki/Directed_acyclic_graph

Directed Acyclic Graph (DAG)

- DAG in Apache Spark is a set of Vertices and Edges.
  - Vertices represent the RDDs and the edges represent the Operations to be applied on RDD.
- In Spark DAG, every edge directs from earlier RDD to later RDD in the sequence.
- On the calling of Action, the created DAG submits to DAG Scheduler which further splits the graph into the stages of the task.
- It is a strict generalization of MapReduce model.

Directed Acyclic Graph (DAG)



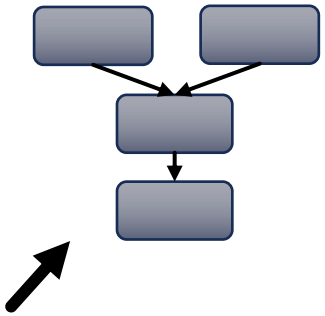
DAG in Apache Spark

Need of DAG in Spark

- Each MapReduce operation is independent of each other and Hadoop has no idea of which operation would come next.
- For multiple-step MapReduce, till the completion of the previous job all the jobs block from the beginning.
  - As a result, complex computation can require a long time.
- While in Spark, a DAG of consecutive computation stages is formed.
  - In this way, we optimize the execution plan, e.g. to minimize shuffling data around.

Scheduling Workflow

RDD Objects

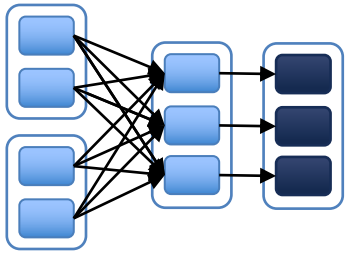


rdd1.join(rdd2)
.groupBy(…)
.filter(…)

build operator DAG

DAGScheduler

DAG

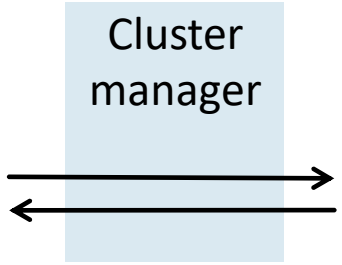


split graph into *stages* of tasks

submit each stage as ready

TaskScheduler

TaskSet

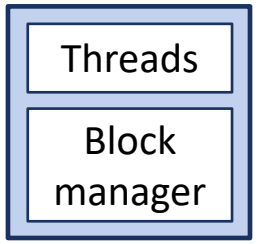


launch tasks via cluster manager

retry failed or straggling tasks

Worker

Task



execute tasks

store and serve blocks

RDD Lineage

- While we create a new RDD from an existing Spark RDD, that new RDD also carries a pointer to the parent RDD.
  - RDD lineage is nothing but the graph of all the parent RDDs of an RDD.
- Rather than the actual data, the lineage graph stores all dependencies between the RDDs.
  - Also called RDD operator graph or RDD dependency graph.

Fault Tolerance

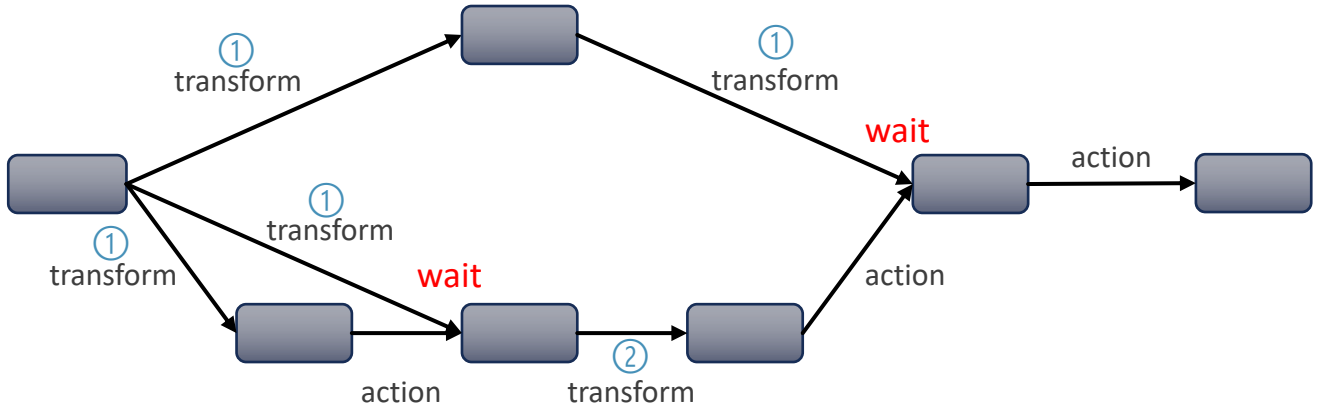
- Since RDD is an immutable dataset, each Spark RDD remembers the lineage of the deterministic operation that was used on fault-tolerant input dataset (e.g. HDFS) to create it.
- If due to a worker node failure any partition of an RDD is lost, then that partition can be re-computed from the original fault-tolerant dataset using the lineage of operations.
  - You know where is someone born (disk) and how does someone grow up (lineage), it doesn't matter if s/he die.

Lazy Evaluation

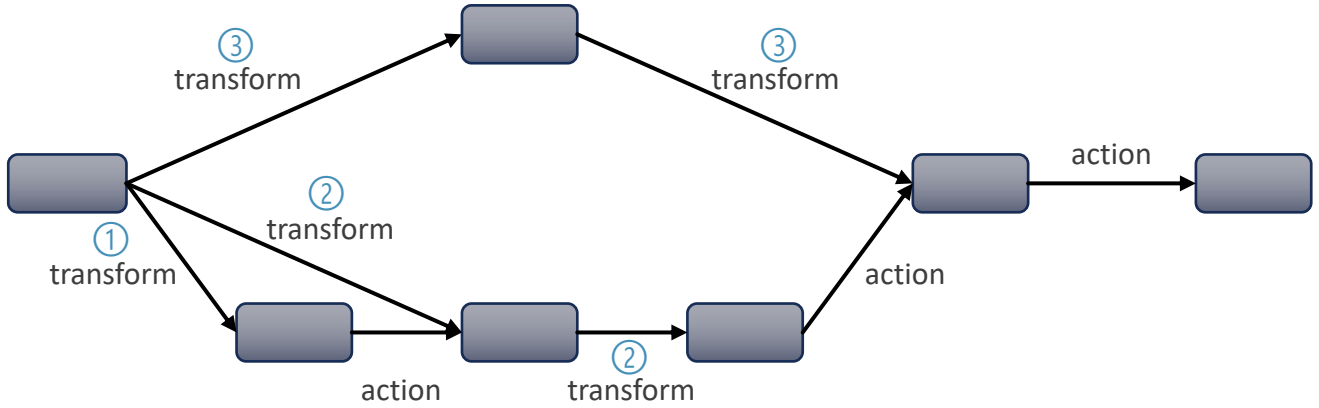
- "LAZY" the word itself indicates its meaning '*not at the same time*'.
  - That means, it evaluates something only when we require it.
- In Spark, it does not execute each transformation right away, until we trigger any action.
  - Once an action is called all the transformations will execute in one go.
- So, in this way by waiting until last minute to execute our code will also enhance the performance.
  - It boosts the efficiency of the process over the cluster.
  - Less chance to wait.

Lazy Evaluation

Non-lazy evaluation



Lazy evaluation



Conclusion

After this lecture, you should know:

- Why do we need Spark.
- What is RDD.
- What are transformations and actions of RDD.
- How does Spark work.
- What is DAG.

Thank you!

- Any question?
- Don’t hesitate to send email to me for asking questions and discussion. ☺

Acknowledgement: Thankfully acknowledge slide contents shared by Dr. Ye Luo