Apache Spark

About

upGrad

Course: Data Engineering - I

Lecture On: Apache Spark

Instructor: Vishwa Mohan

Segment Learning Objectives

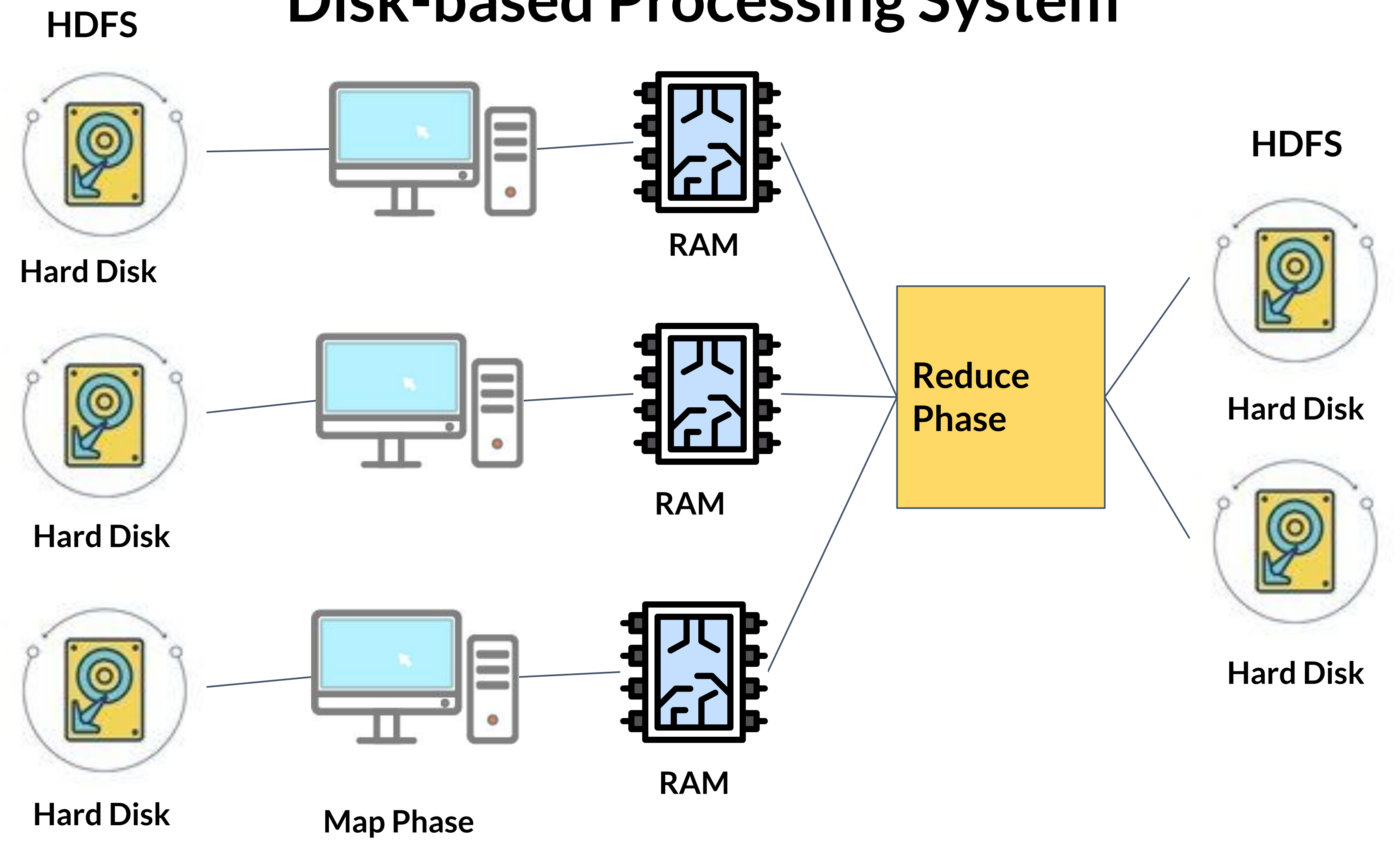
Understanding disk-based processing systems

Analysing why and why not to use disk-based processing systems

Explaining iterative and interactive queries in MapReduce and Spark

Understanding Spark vs MapReduce

Disk-based Processing System



Delays in Hard Disk

Seek Time:
This is the time taken by read/write head to move from the current position to a new position.

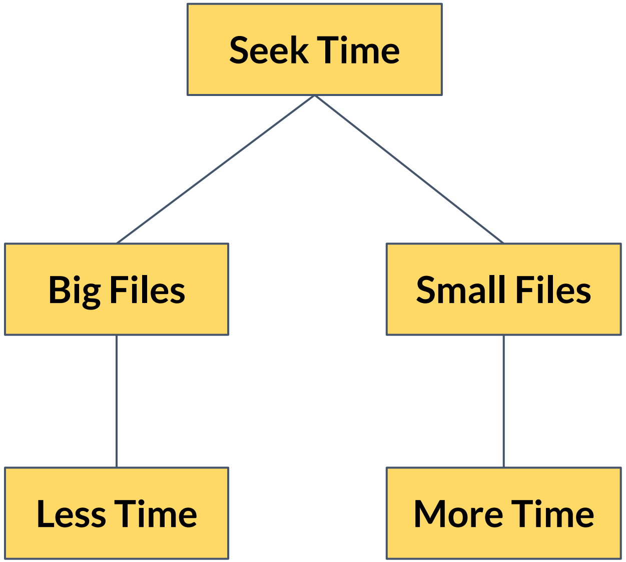
Rotational Delay:
This is the time taken by the disk to rotate so that read/write head points to the beginning of the data chunk.

Transfer Time:
This is the time taken to read/write the data from/in the data chunk in the hard disk.

Access Time = Seek Time + Rotational Delay + Transfer Time

Delays in Disk-based Processing System

Seek Time Delay Analysis:



Rotational Time Delay Analysis:

Since hard disk always retrieves data sequentially from the very start of the data chunk, the disk has to rotate so that read/write head points to the start of the chunk.

If you want to retrieve data randomly, it will take a lot of time to access that data.

Why and Why not Disk-based Processing System?

Why?

It is suitable for storing and managing large amounts of data. Since the data is in hard drive, it is much safer and fault tolerant.

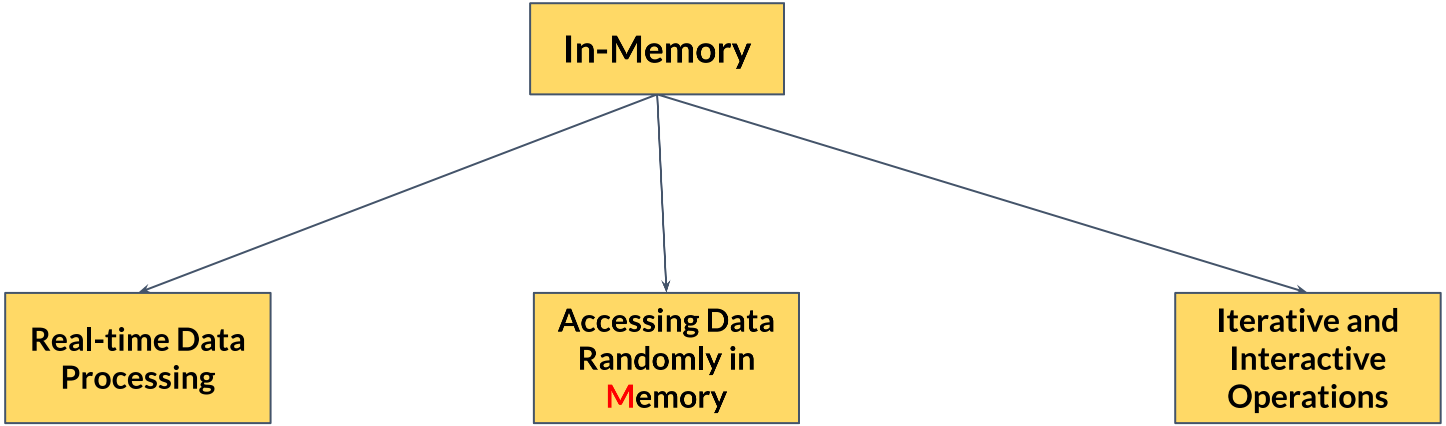
Since the historical data is stored in large amounts, it is suitable for batch processing of the data.

Why not?

Even for batch processing of data, the disk I/O consumes a lot of job's run-time.

It cannot be used for real-time data for immediate results.

Why In-Memory Processing?

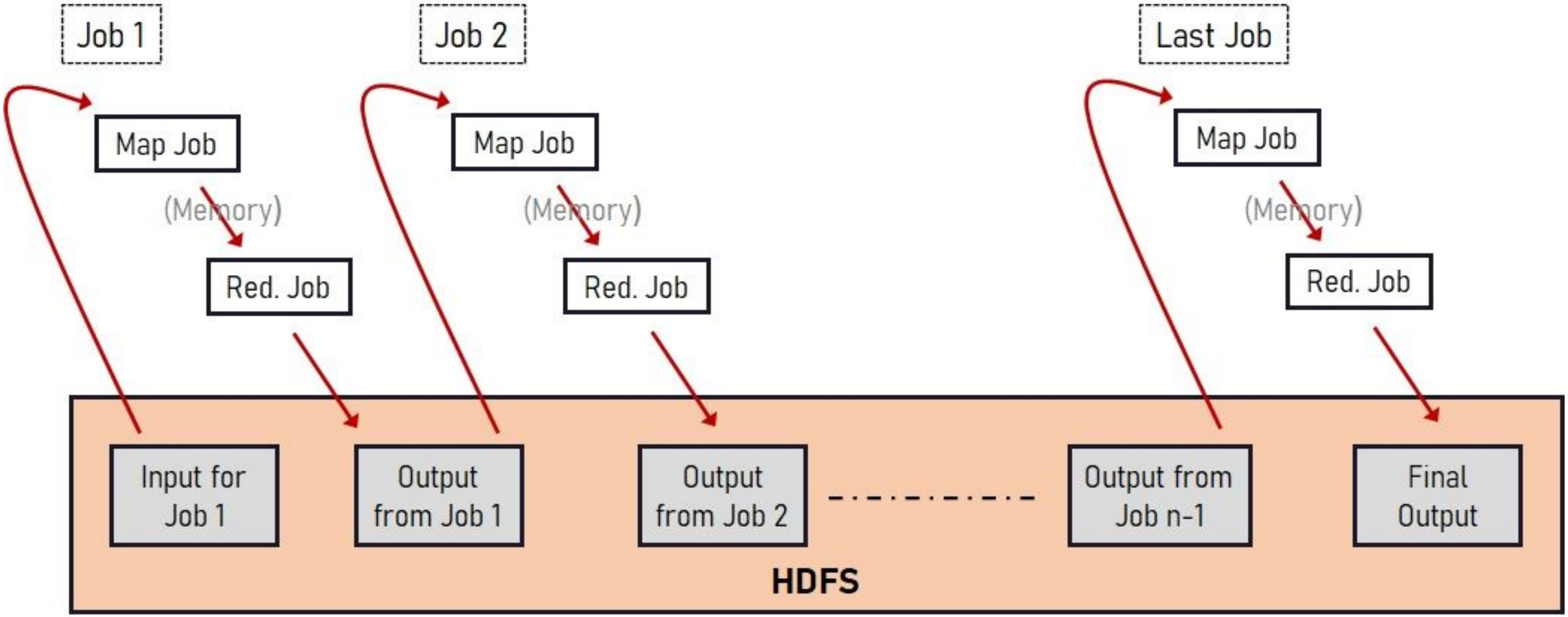


Since data can be accessed very fast, In-Memory processing can be used in cases where immediate results are required.

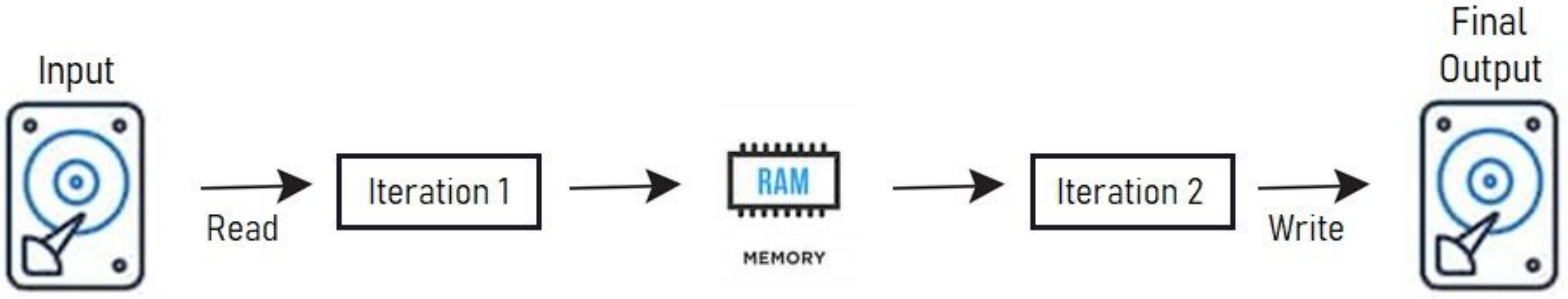
Since data is stored in the RAM, memory can be randomly accessed without scanning the entire storage.

Intermediate results are stored in memory and not in disk storage, so we can use this output in other computations.

Iterative Operations in MapReduce

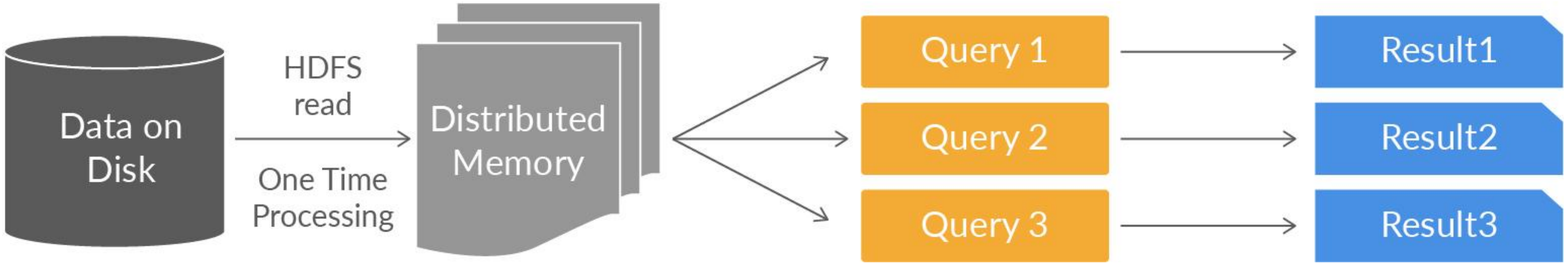


Iterative Operations in Spark



Interactive Operations in MapReduce

Interactive Operations in Spark

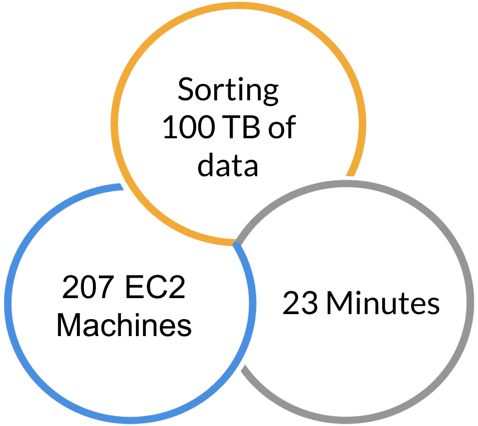


Spark vs MapReduce

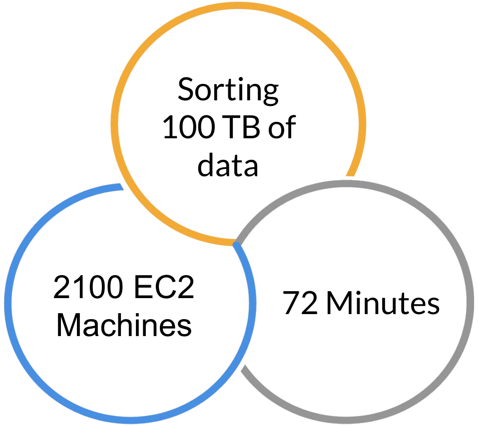
MapReduce	Spark
This involves disk-based processing; hence, the processing is slow.	This involves in-memory processing; hence, the processing is fast.
This involves only batch processing.	This involves batch and real-time processing.
It supports HDFS as data storage.	It can connect to multiple data sources: local files and/or various distributed file systems, i.e. HDFS, S3 etc.
It was originally developed in Java but supports C++, Ruby, Groovy, and Perl using Hadoop streaming library.	It was originally developed in Scala but also supports Java. Further, it supports Python, R and SQL to the extent possible.
Raw API, which does not have much support.	Rich APIs.

Spark vs MapReduce Processing Speed

Spark



Map Reduce



Experiment done by Data Bricks

Why is Spark so fast?

1. Because of its in-memory computation
2. Because of its optimised execution through DAG
3. Because of its lazy evaluation

Segment Summary

Disk-based processing systems such as MapReduce are slow for batch processing and not suitable for real-time data analysis

Spark is an in-memory processing system and is 100x faster than MapReduce

Iterative and Interactive queries are slow in MapReduce as intermediate results are written on disk and data has to be read from disk for every new query.

Segment Learning Objectives

Understanding RDD

Illustration of how RDD is stored in different partitions in memory

Creating RDD in pyspark on jupyter notebook

R D D

Similar to a data structure such as Arrays or Lists

Resilient

- This means that data in RDDs are easy to recover and fault tolerant.

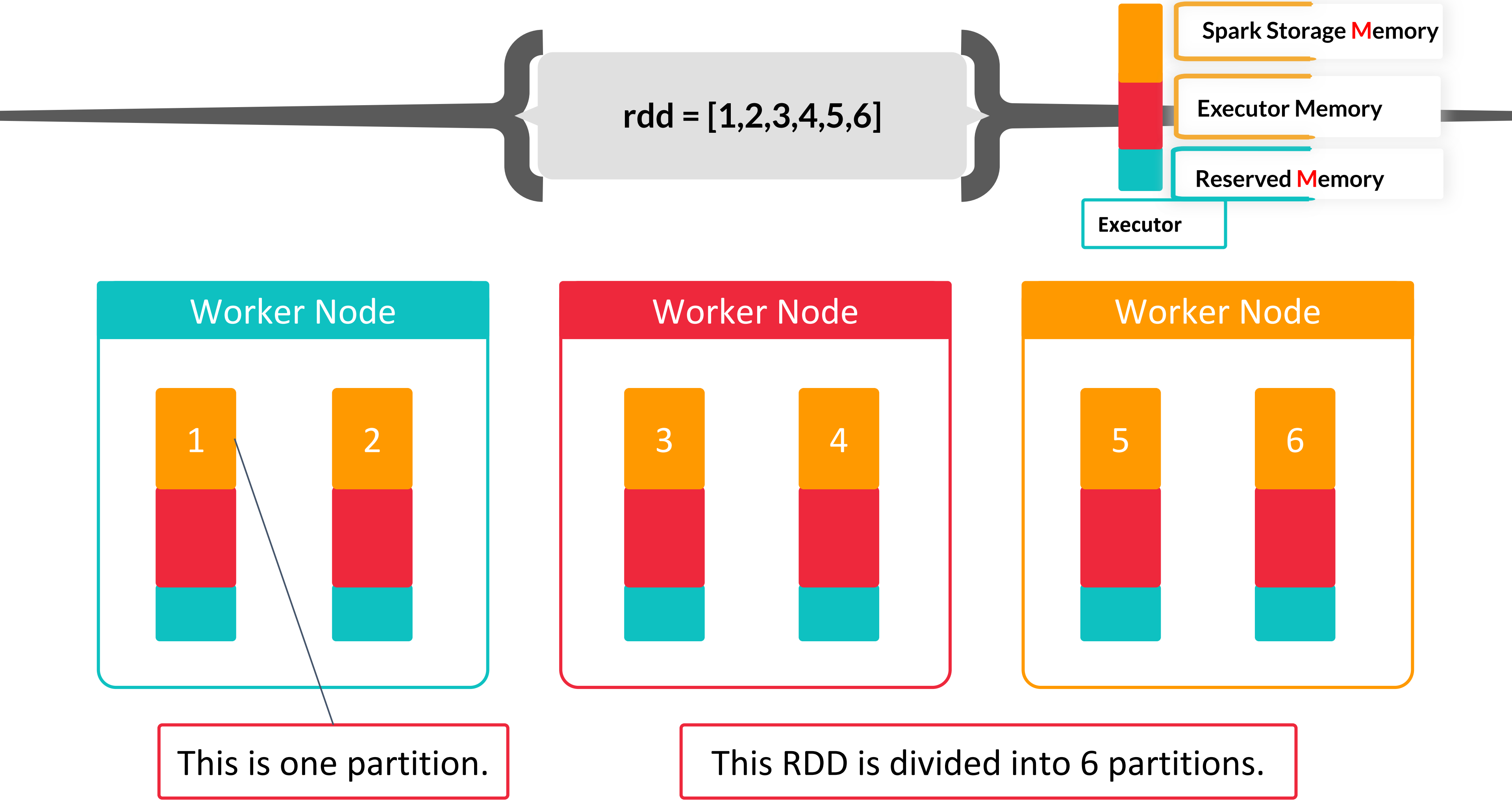
- This property comes from how spark constructs a data lineage and stores information about how each RDD was built.

Distributed

RDDs are not stored in a single executor but distributed over many executors

Dataset

Collection of related information which can be of any data type


rdd = [1,2,3,4,5,6]
Spark Storage Memory
Executor Memory
Reserved Memory
Executor
Worker Node
1
2
Worker Node
3
4
Worker Node
5
6
This is one partition.
This RDD is divided into 6 partitions.

Segment Learning Objectives

Understanding transformations and actions

Illustration of various operation on Basic RDD

Transformation and Actions

Transformation operations on an RDD result in a new RDD

Actions result in an output and not stored as RDD

```
rdd2 = rdd1.map()
rdd2 = rdd1.filter()
rdd3 = rdd1.union(rdd2)
```

```
rdd1.collect()
rdd1.count()
rdd1.top(2)
```

Segment Learning Objectives

Understanding Lazy evaluation in Spark

Understanding Directed Acyclic Graph

Understanding why Lazy evaluation in Spark makes it fault tolerant

Lazy Evaluation in Spark

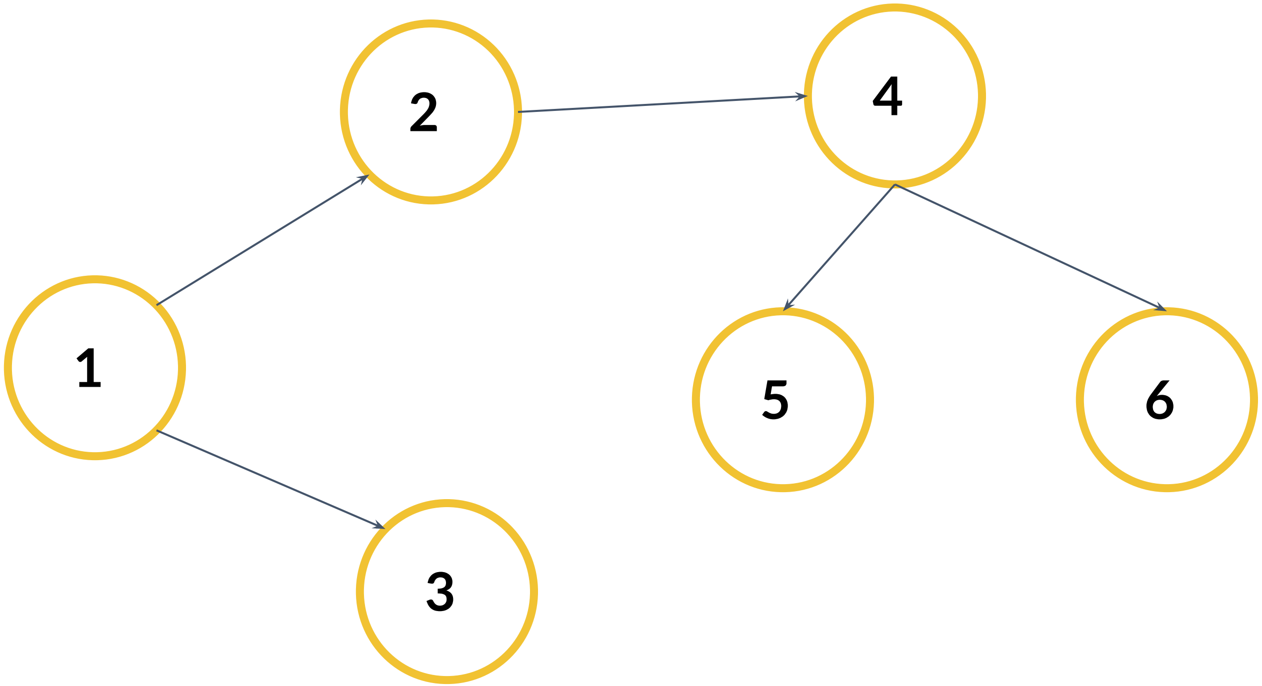
Spark does not perform any transformation until an action is called on an RDD.

Rather Spark created a lineage that stores how each RDD can be derived from transformations.

Since Spark knows how to derive each RDD, in case of system failure, RDD can be recreated.

Lazy evaluation in Spark makes RDD resilient and fault tolerant.

Directed Acyclic Graph



Spark Lineage

Consider the code in the following slides:

1

```
rdd1=sc.parallelize([11,12,13,14,15])
```

rdd1 = [11,12,13,14,15]

2

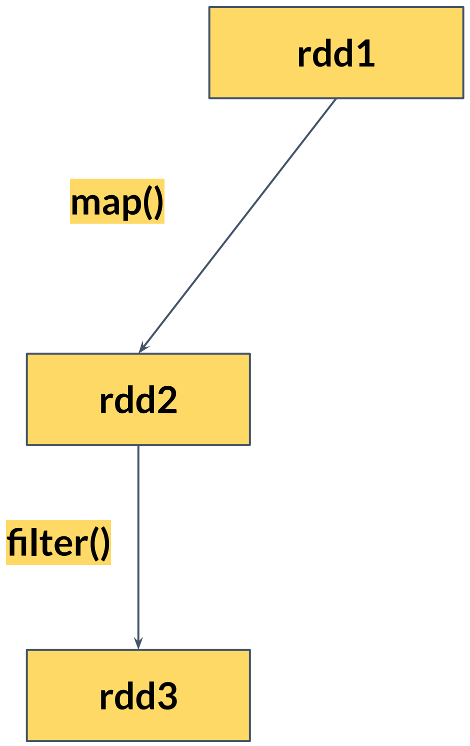
```
rdd2 = rdd1.map(lambda x:x+1)
```

rdd2 = [12,13,14,15,16]

3

```
rdd3 = rdd2.filter(lambda x:x > 12)
```

rdd3 = [13,14,15,16]



Spark Lineage

4

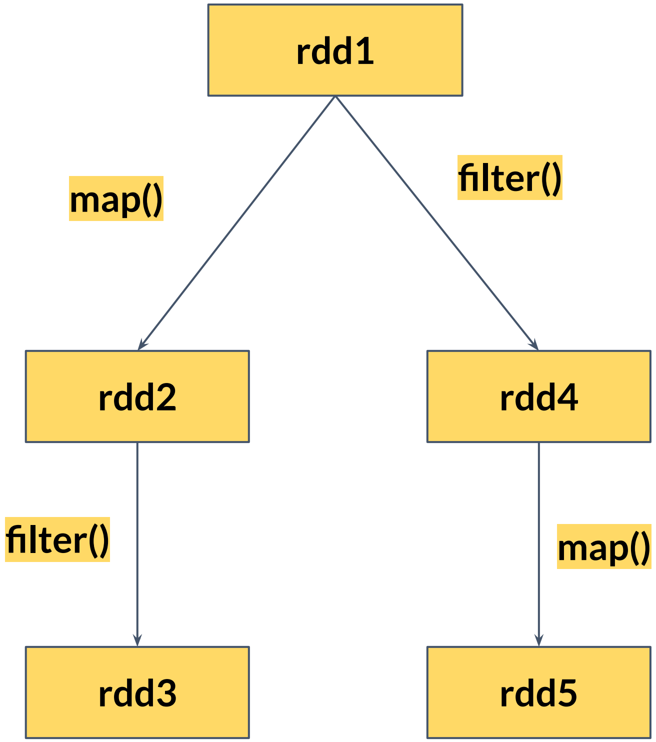
```
rdd4 = rdd1.filter(lambda x:x>11)
```

rdd4 = [12,13,14,15]

5

```
rdd5 = rdd4.map(lambda x:x*2)
```

rdd5 = [24,26,28,30]



Spark Lineage

6

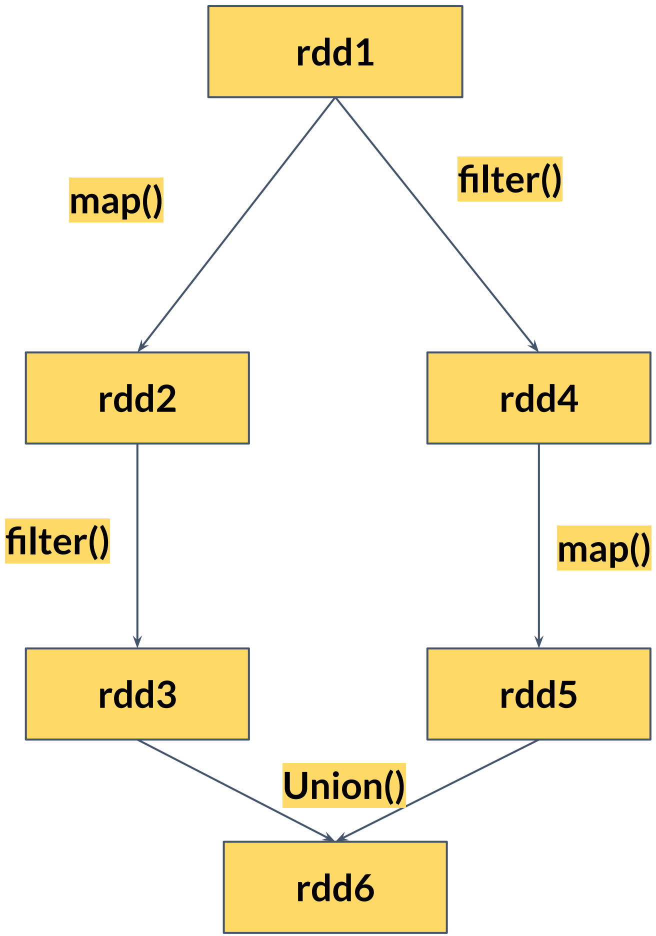
rdd6 = rdd3.union(rdd5)

rdd6 = [13,14,15,16,24,26,28,30]

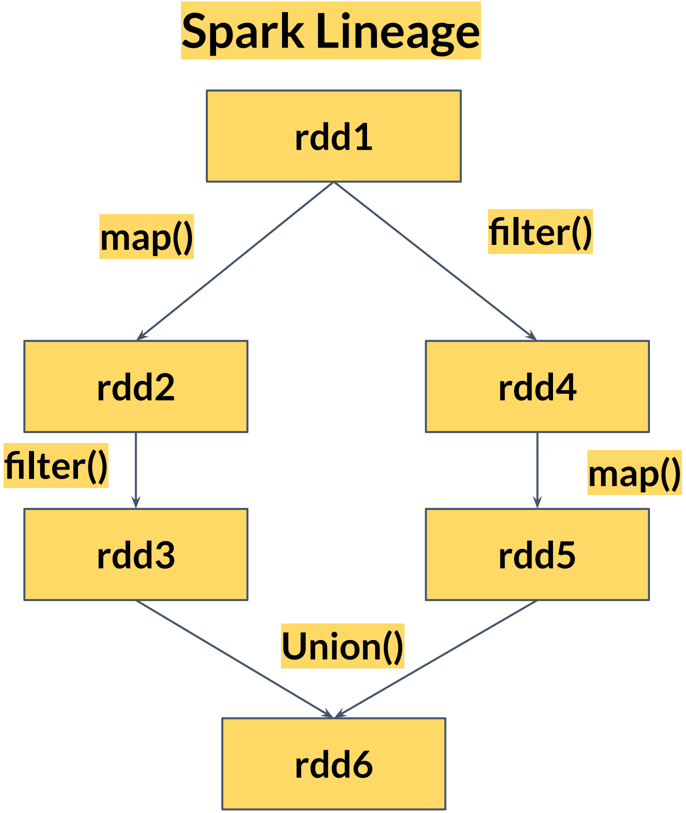
7

rdd6.collect()

rdd6 = [24,26,28,30]



Directed Acyclic Graph (DAG)



Segment Summary

Lazy Evaluation in Spark means any computation will happen only when an action is called.

Spark creates a DAG to store the information on how each RDD can be derived.

This makes RDD fault tolerant.

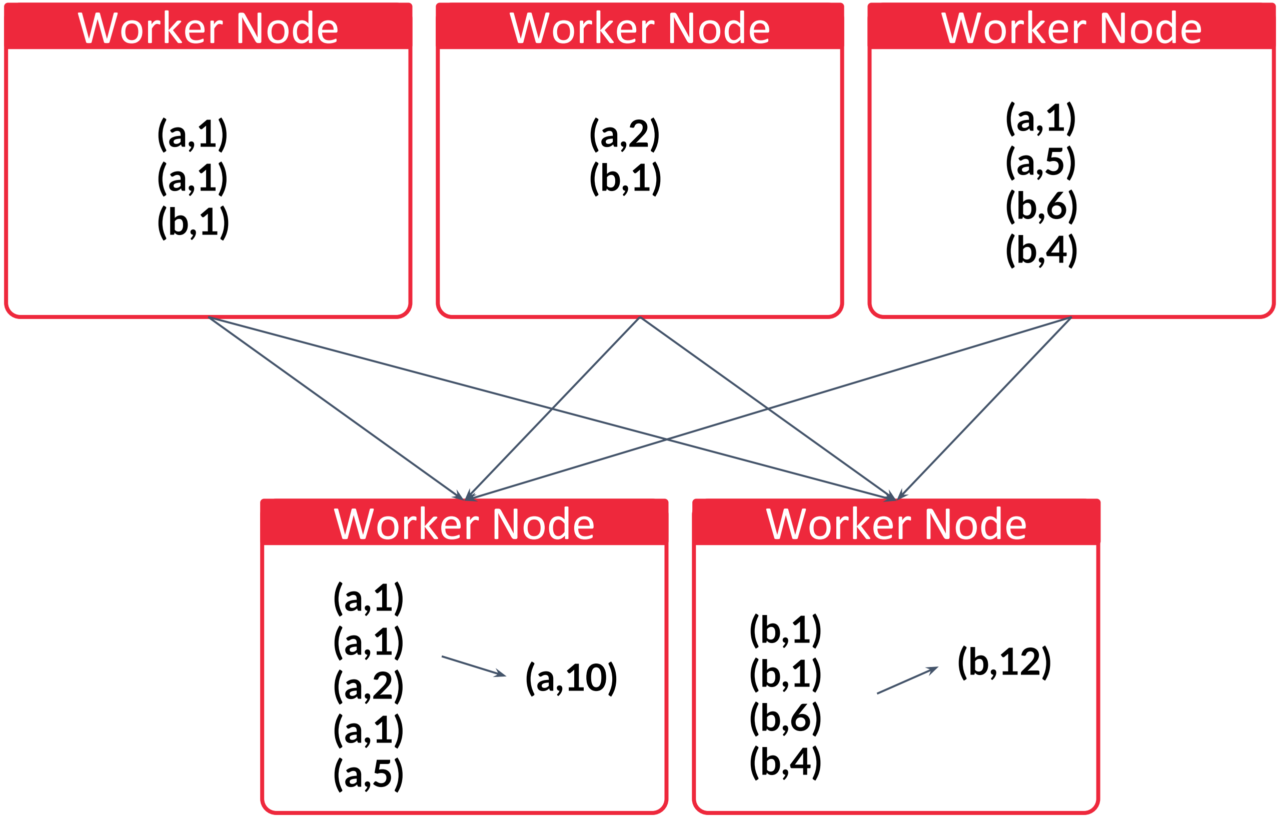
Features of Spark


Speed
Ease of Use
APACHE Spark
Generality
Runs Everywhere

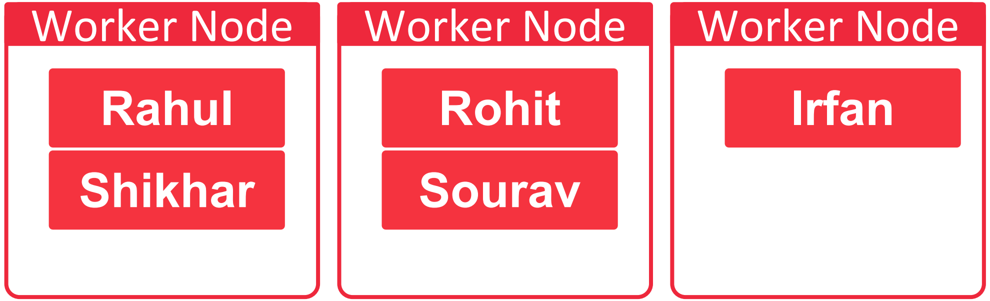
lambda x:x*2

Each element of RDD

Operation on each element of RDD


Worker Node
(a,1)
(a,1)
(b,1)
Worker Node
(a,2)
(b,1)
Worker Node
(a,1)
(a,5)
(b,6)
(b,4)
Worker Node
(a,1)
(a,1)
(a,2)
(a,1)
(a,5)
(a,10)
Worker Node
(b,1)
(b,1)
(b,6)
(b,4)
(b,12)

Rahul	50
Shikhar	100
Rohit	150
Sourav	200
Irfan	250



Executors

Executors are JVM machines on worker nodes on which spark runs

One Executor can consume one or more than one core on a single worker node

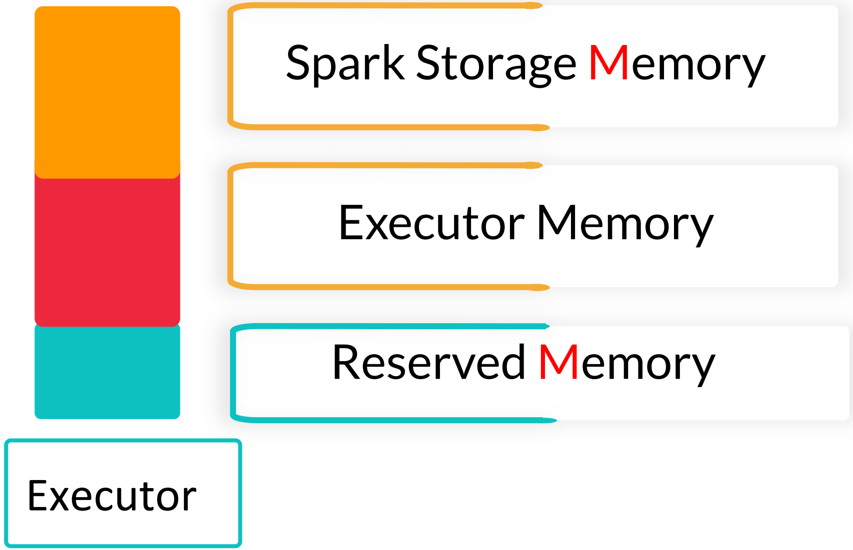
Assume:
1 worker node = 8 cores
1 Executor = 1 core
1 worker node = 8 Executors

Operations on the data in executor cannot be parallelised

Assume:
1 worker node = 8 cores
1 Executor = 2 cores
1 worker node = 4 Executors

Operations on the data in executor can be parallelised

Executors


Spark Storage Memory
Executor Memory
Reserved Memory
Executor

Thank You