Nitesh Kulkarni

niteshkulkarni8@gmail.com

Abstract

I am maintaining this book for my personal reference.

DATA SCIENCE Training

Zak & Akshay

Table of Contents

[Lecture 11: Intro to Tools and PreRequisite downloads 2](#_Toc503025027)

[Lecture 12: HDFS, Hadoop, MapReduce 3](#_Toc503025028)

[Apache Mirrors 4](#_Toc503025029)

[Lecture 13: HDFS, NameNode, DataNode, Job Tracker, Task Tracker 4](#_Toc503025030)

[Replication 5](#_Toc503025031)

[Rack 5](#_Toc503025032)

[Lecture 14, 15, 18, 19: Limitations of MapReduce, why we use Spark 7](#_Toc503025033)

[Map Reduce and its limitations 7](#_Toc503025034)

[Spark 8](#_Toc503025035)

[Unified Engine 9](#_Toc503025036)

[Spark Abstraction 10](#_Toc503025037)

[MapReduce vs. Spark 10](#_Toc503025038)

[Lecture 21,22: Scala Coding 10](#_Toc503025039)

[Lecture 23,24: Spark 11](#_Toc503025040)

[Spark RDD (Resilient Distributed Dataset) 11](#_Toc503025041)

[Spark RDD Operations 12](#_Toc503025042)

[Spark Architecture 13](#_Toc503025043)

[Spark Terminology 14](#_Toc503025044)

[Lecture 20: Scala 15](#_Toc503025045)

[Creating Values and Variables w/o type inference 16](#_Toc503025046)

[Creating value/Variable with type inference 16](#_Toc503025047)

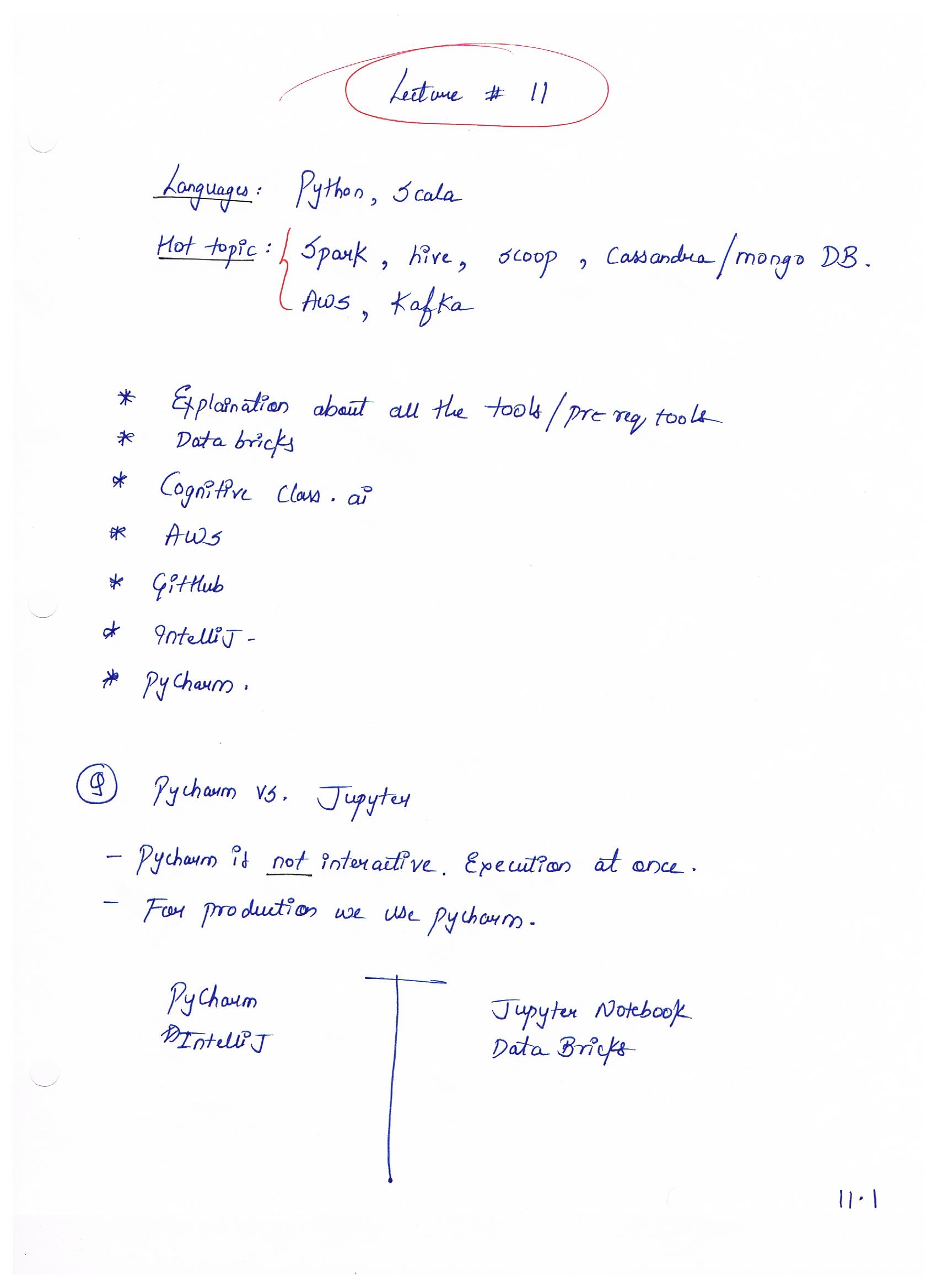
[Difference between Val and Var in Scala 16](#_Toc503025048)

[List in Scala 18](#_Toc503025049)

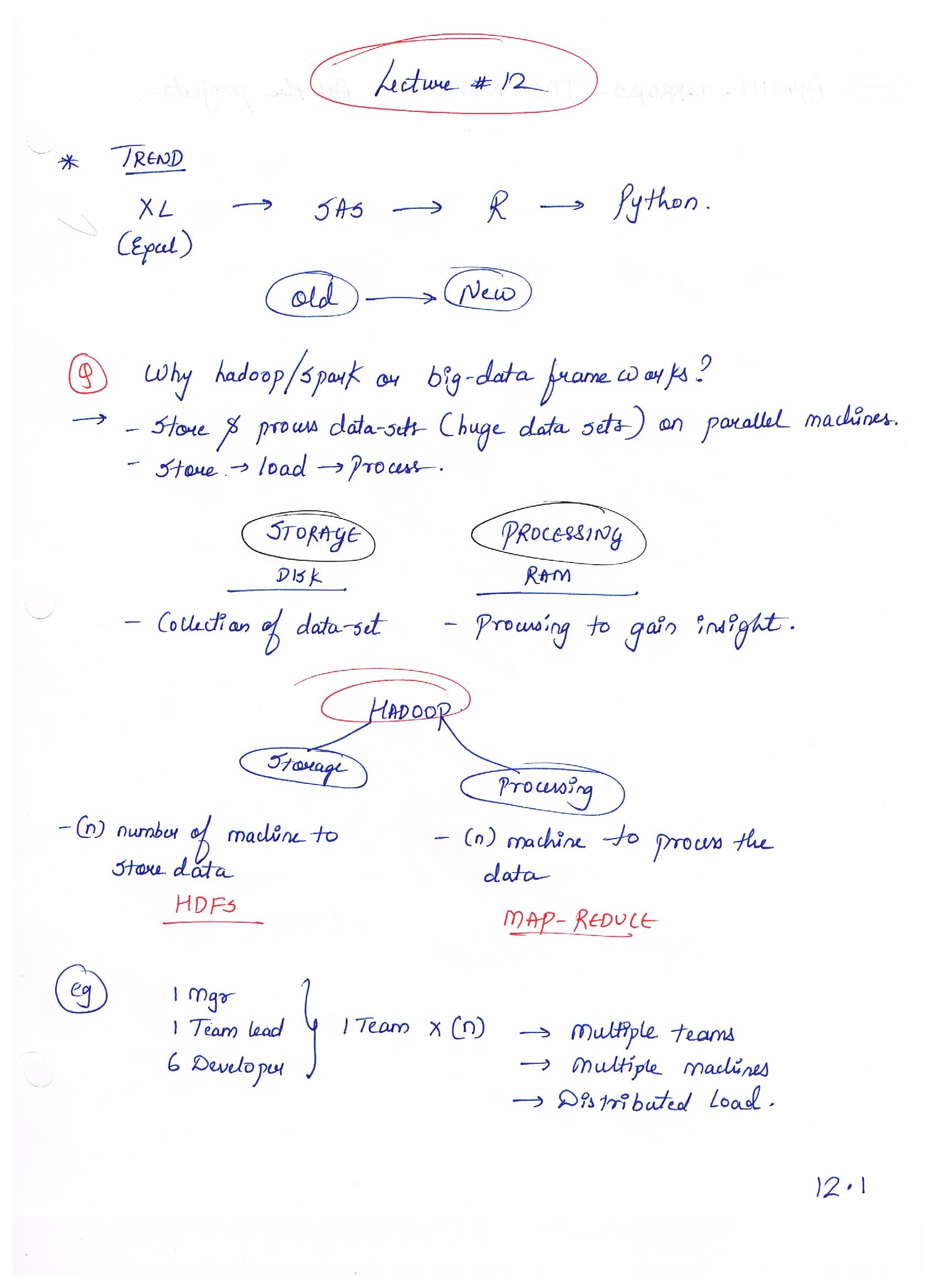
[Difference between Python and Scala on Data Types 18](#_Toc503025050)

[Assignment Zak: MapReduce, Combiner, Partitioner, Shuffle, HDFS, Git 21](#_Toc503025051)

# Lecture 11: Intro to Tools and PreRequisite downloads



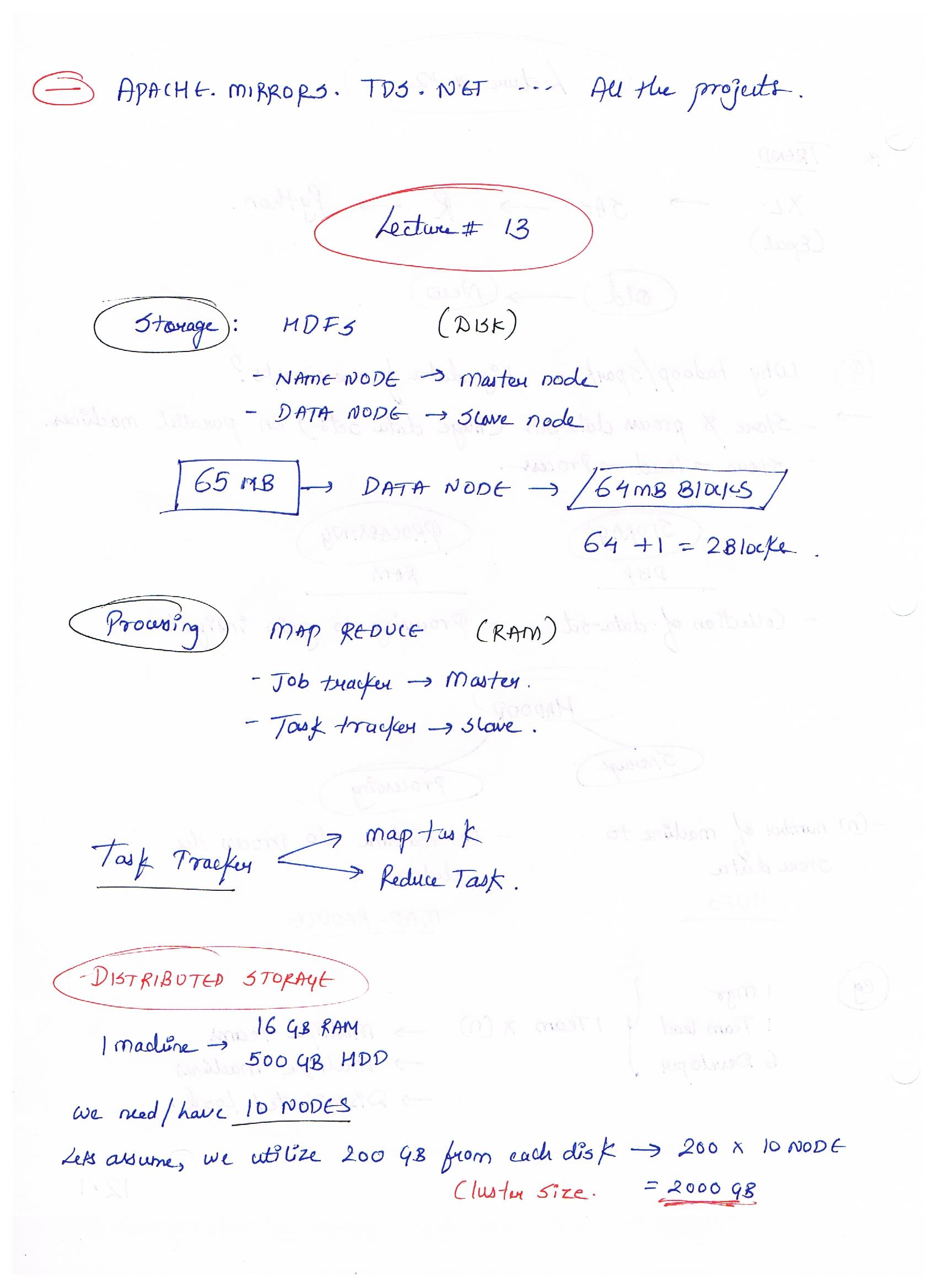
# Lecture 12: HDFS, Hadoop, MapReduce



Apache Mirrors  
apache.mirrors.tds.net

Will have multiple projects built for Data Science eg. TensorFlow, etc

# Lecture 13: HDFS, NameNode, DataNode, Job Tracker, Task Tracker



## Replication

**Replication Factor = 3 (By Default)**

Each block is replicated 3-times by default

Eg)

Block1 ->64 MB

Block2 -> 1MB

Rack1:

dn1 -> Block1 ->64 MB, Block2 -> 1MB

dn2 ->

dn3 -> Block1 ->64 MB

Rack2:

dn4 -> Block1 ->64 MB

**…**

dn10 -> Block2 -> 1MB

Rack **Rack** is like **drawer** where DN is stored.

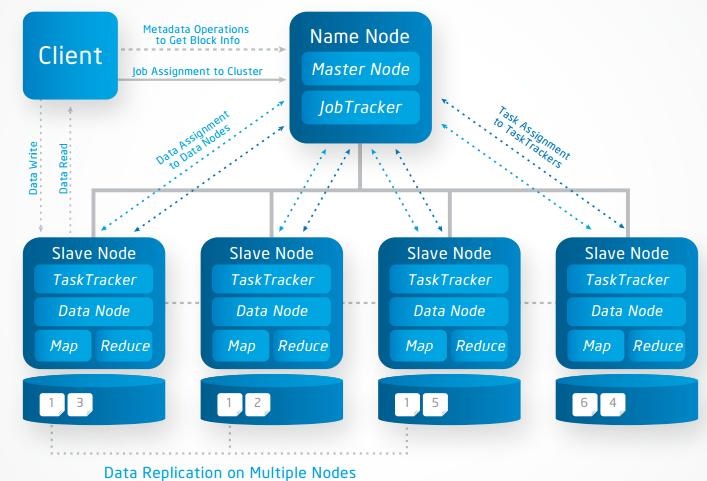
* Each rack will have multiple DN
* Multiple rack form a cluster

**Note: Replication is done for high-availability.**

* **If 1 data-node is down, then the block stored on other DN will be retrived.**
* **Eg) if dn1 is down, then the block is retrived from dn3.**

**Mapping is done will be by NN (Name-node)**

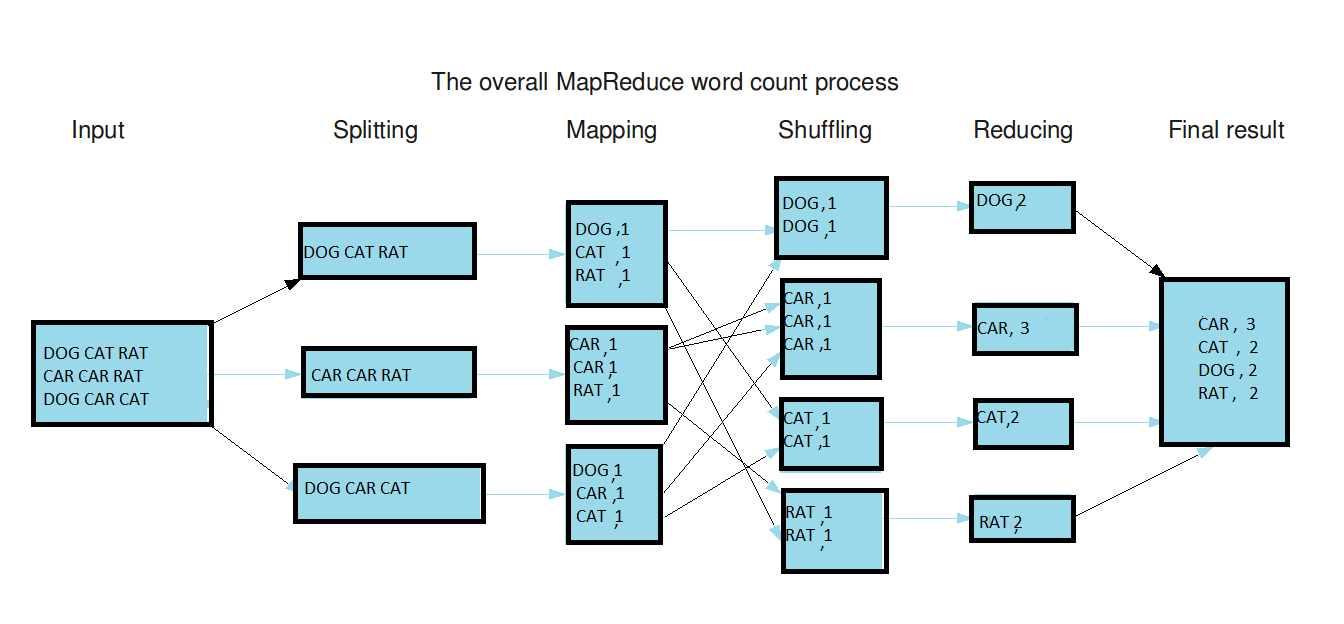
* **NameNode will remember all the replication mapping**

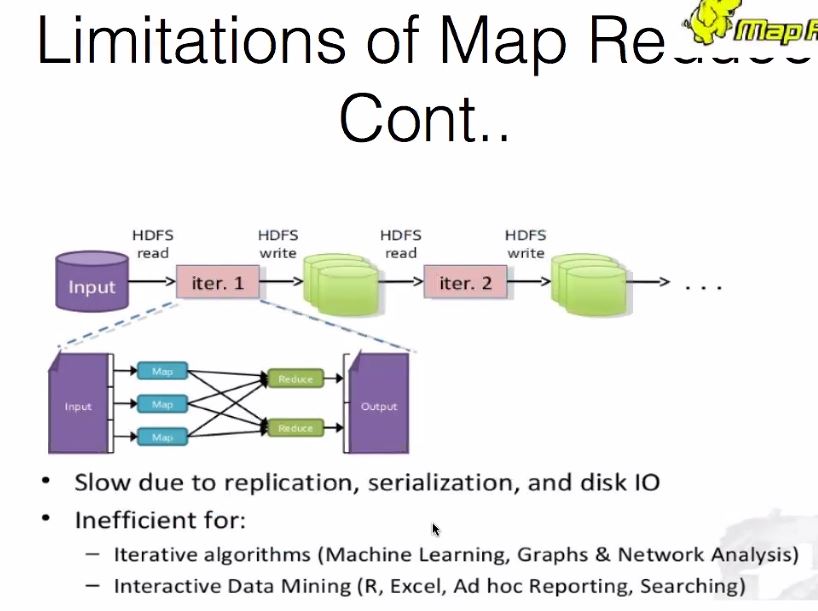
****

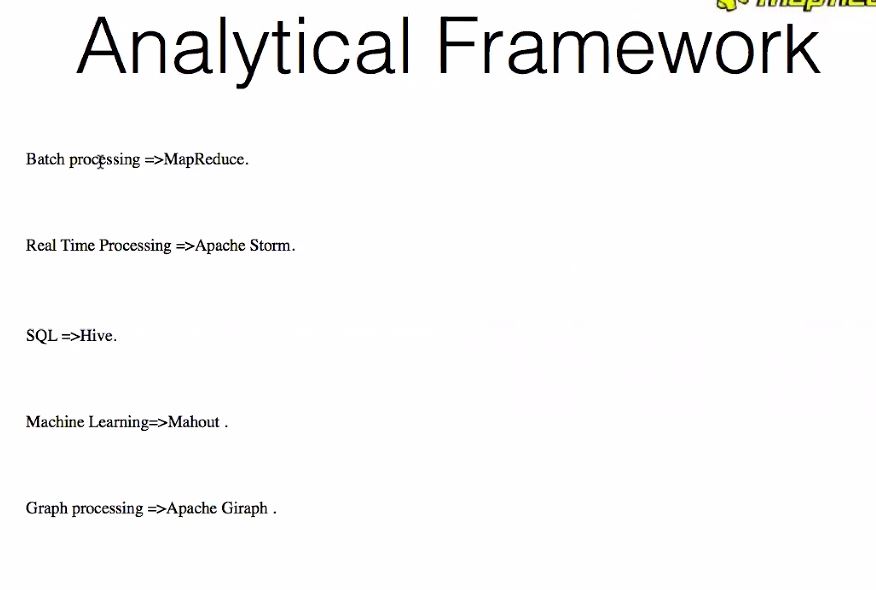
# 

# Lecture 14, 15, 18, 19: Limitations of MapReduce, why we use Spark

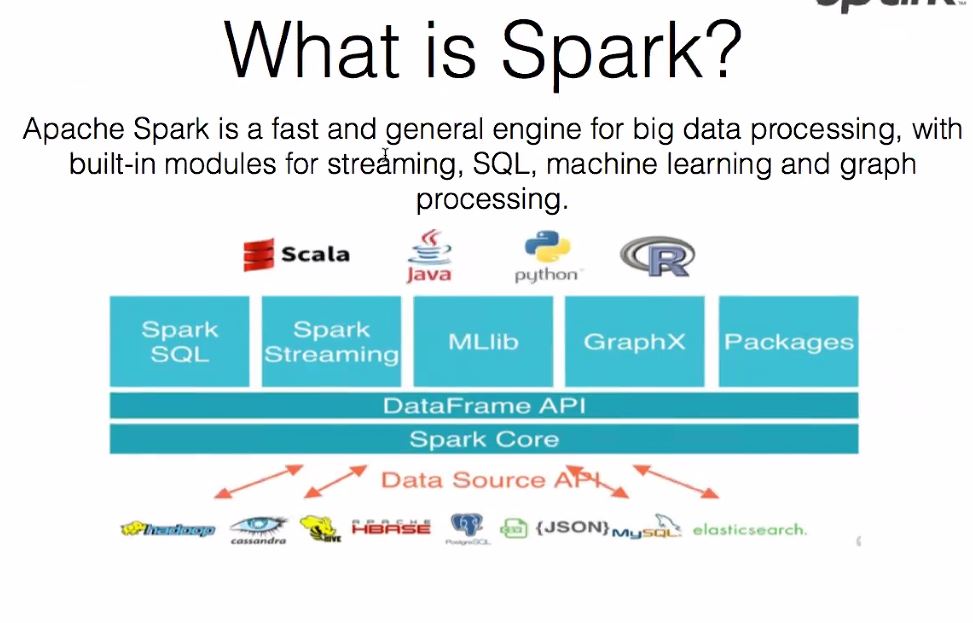
## Map Reduce and its limitations



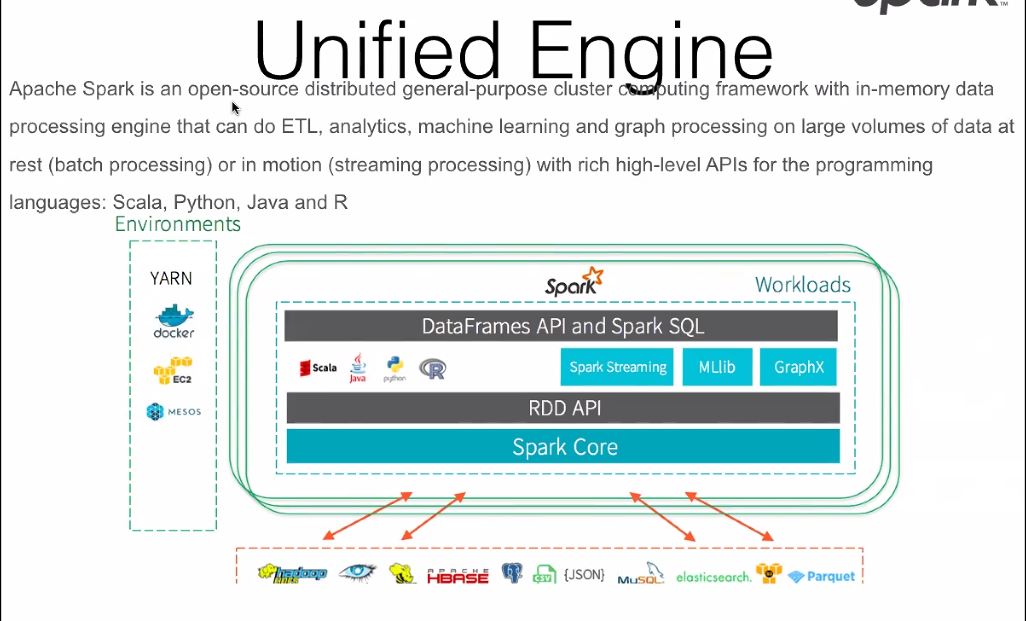


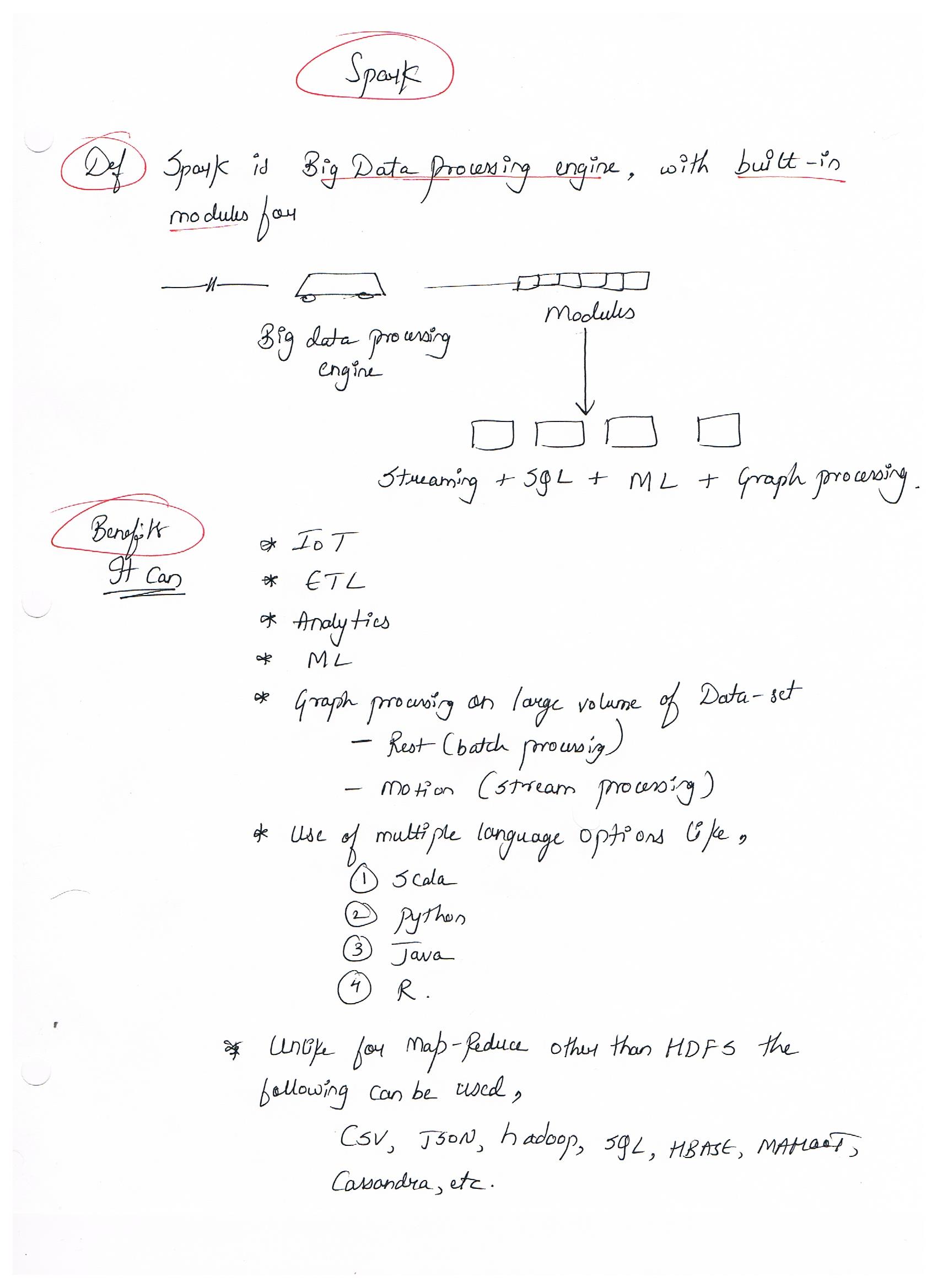


## Spark

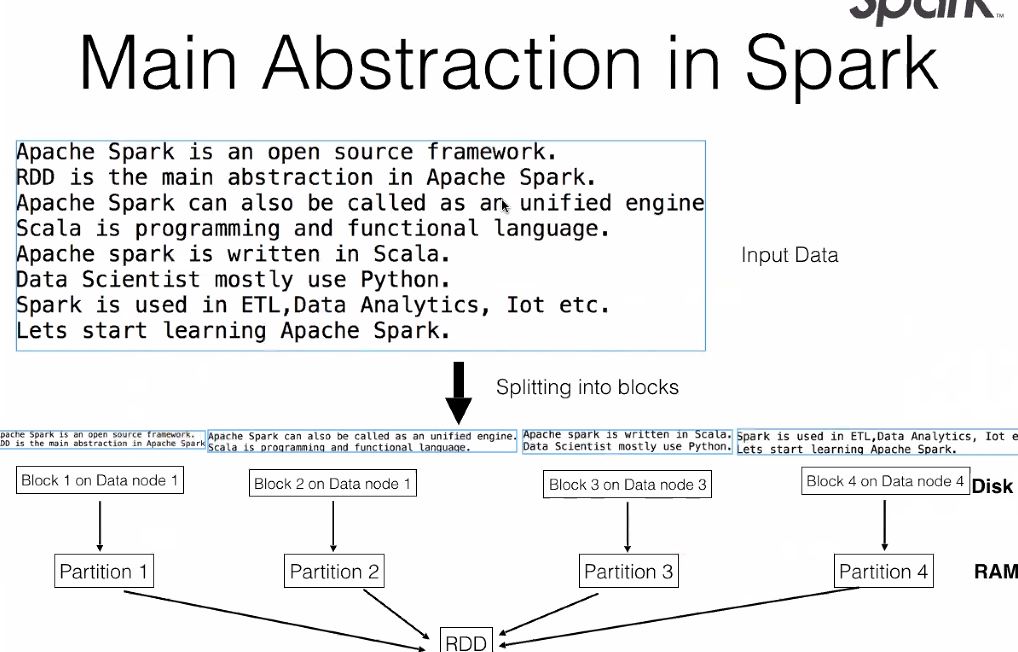


## Unified Engine





## Spark Abstraction



## MapReduce vs. Spark

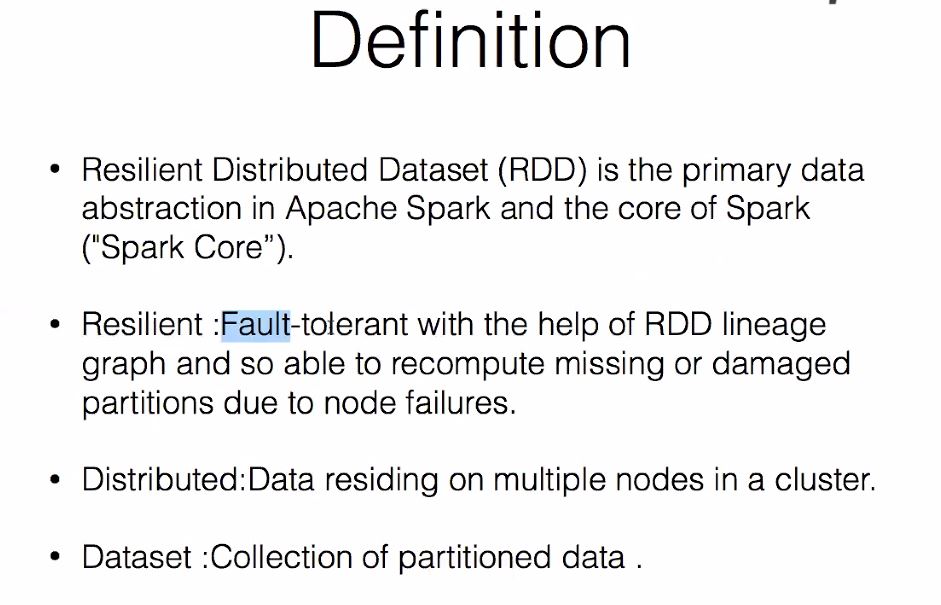
|  |  |
| --- | --- |
| **MapReduce** | **Spark** |
|  |  |
| * Input data is always HDFS | * Spark allows multiple data source |

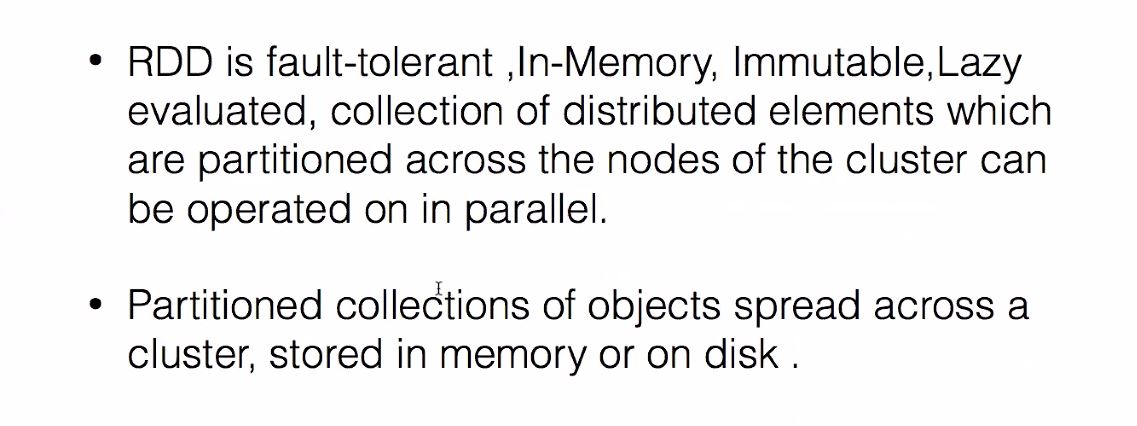
**Competition**

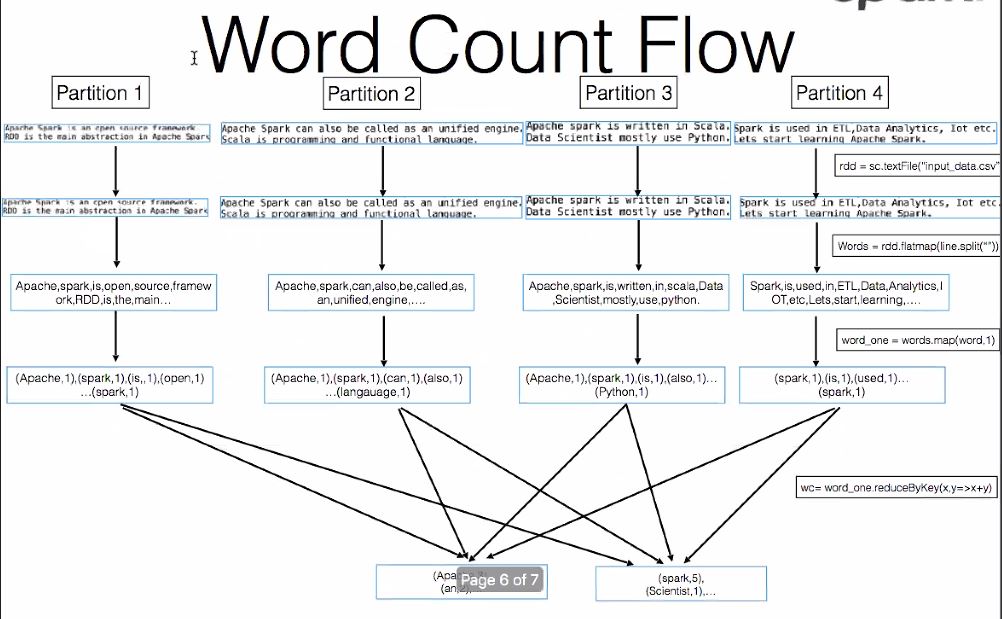
MapReduce 🡪 Spark 🡪 Flink

# Lecture 23,24: Spark

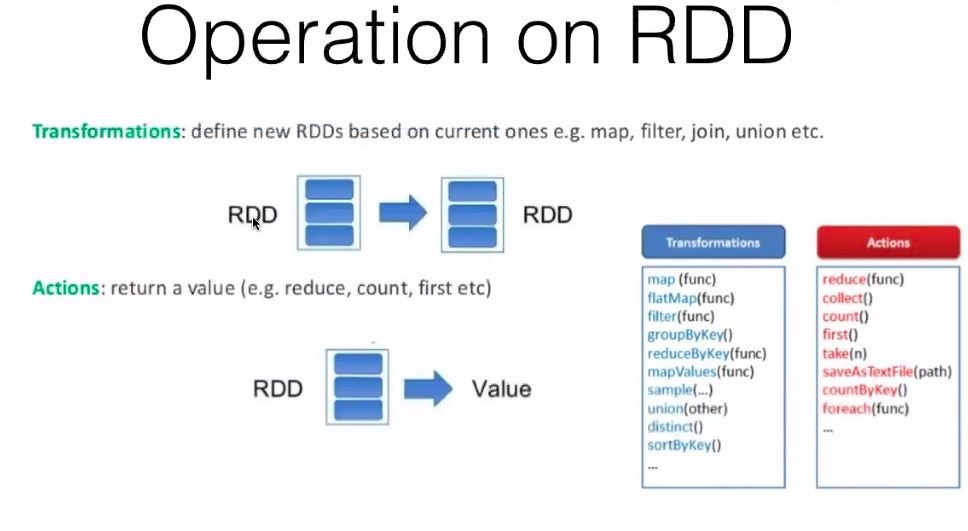
## Spark RDD (Resilient Distributed Dataset)



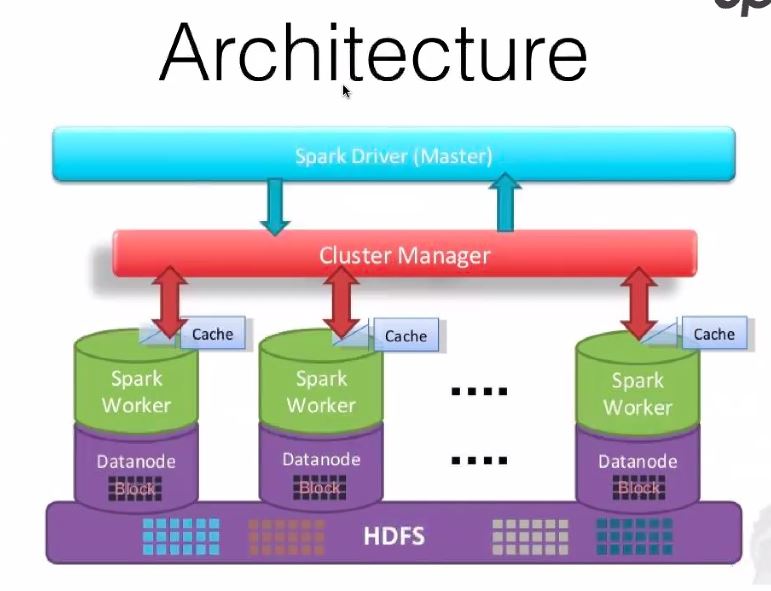


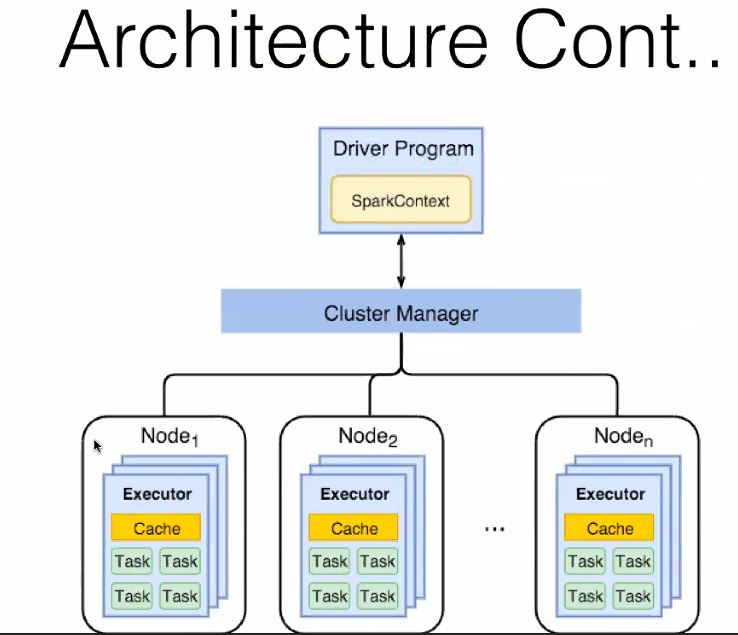


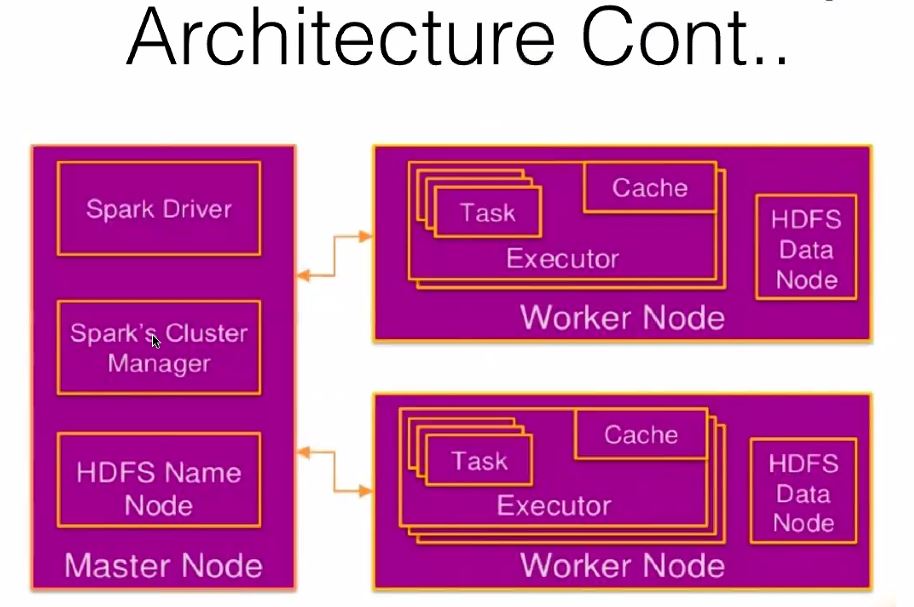
## Spark RDD Operations



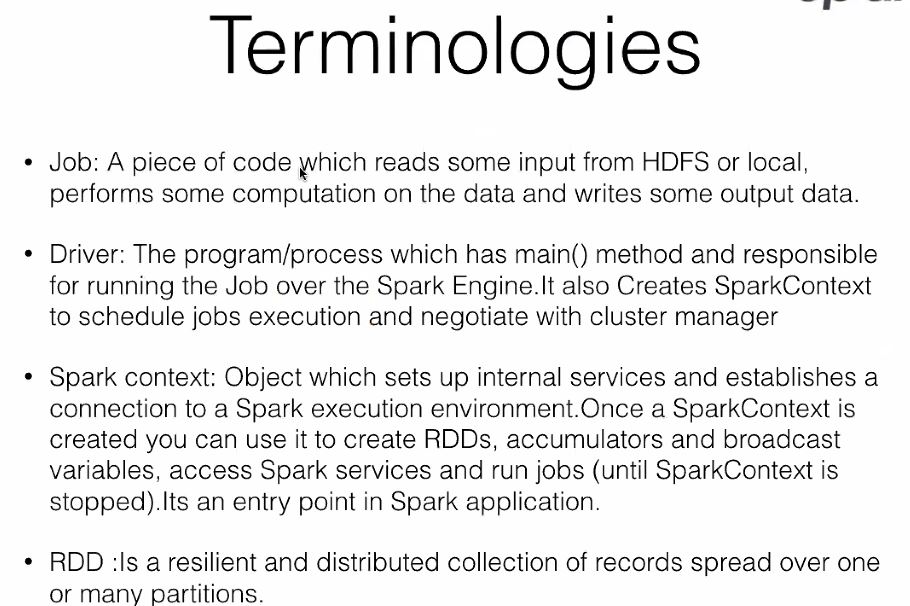
## Spark Architecture

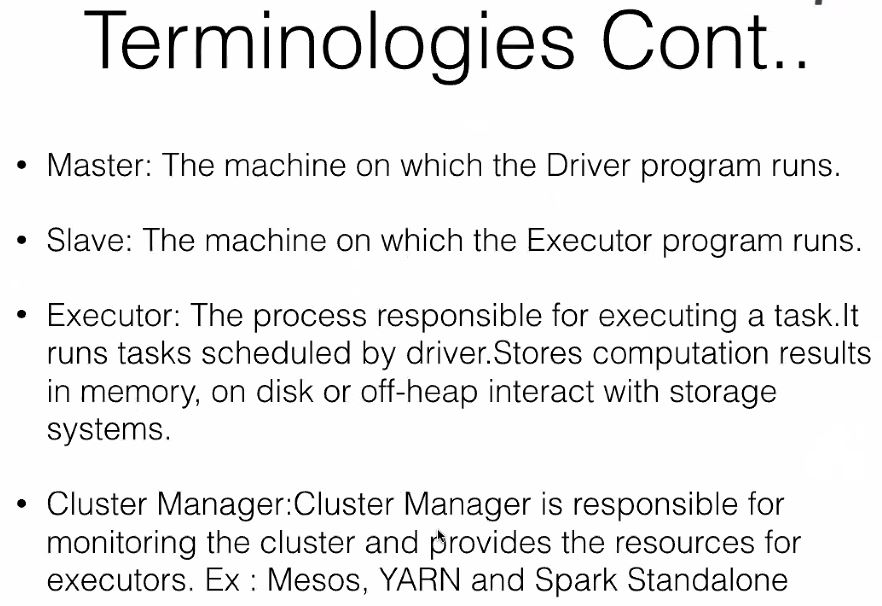






## Spark Terminology





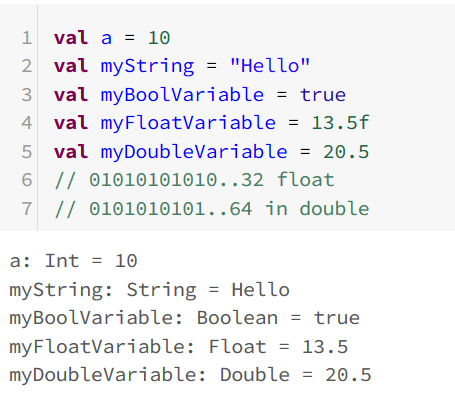
# Lecture 20: Scala

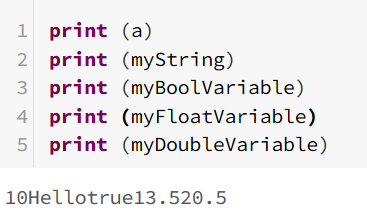
**Scala**it is JVM based, modern programming language, succinct syntax and supports both object-oriented and functional programming style with many more advanced features.

* 10 times faster than Python, Better performance
* JVM based
* Support OOP and Functional programming
* Scalable

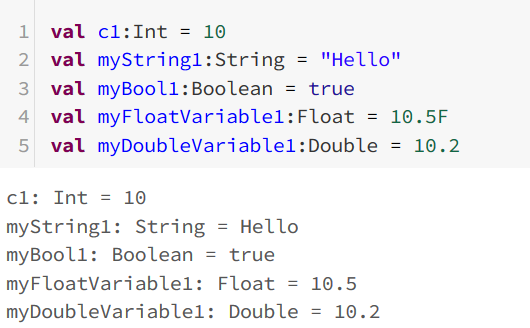
Look for scala\_learning\_notebook on **Data Bricks**

## Creating Values and Variables w/o type inference



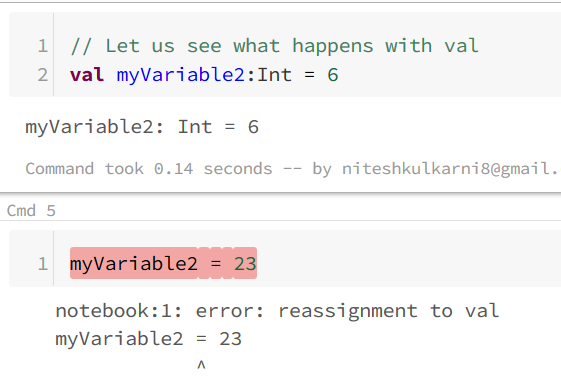


## Creating value/Variable with type inference

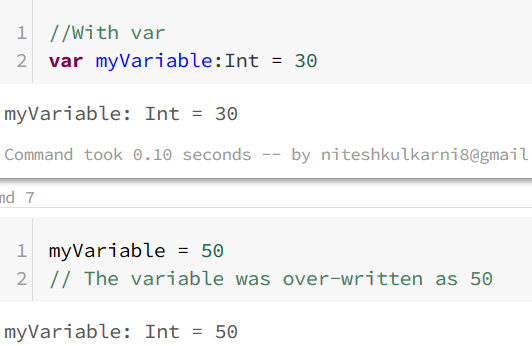


## Difference between Val and Var in Scala

**For Val**

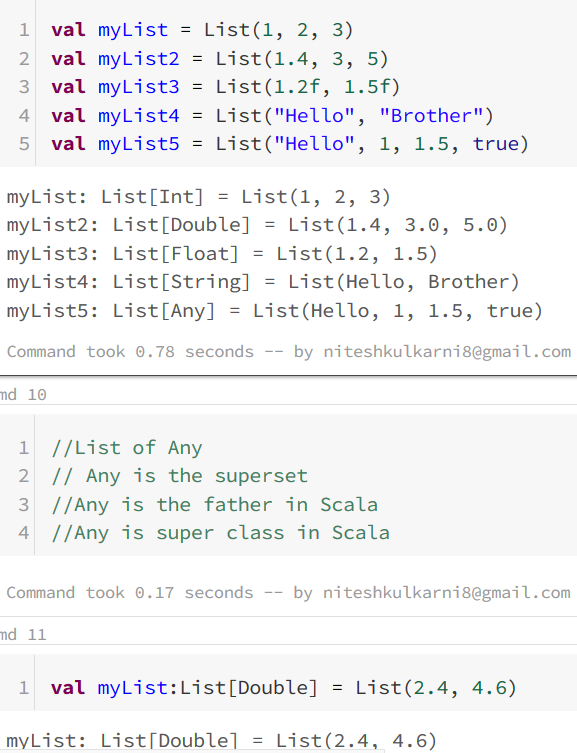


**For Var**



* Var = **mutabale**
* Val = **immuatable**
* **Scala allows one to decide whether a variable is immutable or mutable. Immutable is Read only whereas mutable is read-write. Immutable variables are declared with the keyword “val “.**

## List in Scala



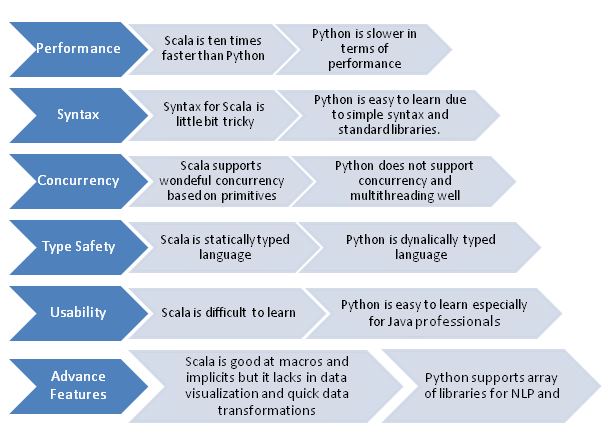
## Difference between Python and Scala on Data Types

**Read this for details**: <https://bugra.github.io/work/notes/2014-10-18/scala-basics-for-python-developers/>

In **Scala**: you could have a mutable and immutable string or integer, whereas

In **Python** you have predefined datasets which are immutable (string and tuple) and mutable (list and dictionary).

<https://www.janbasktraining.com/blog/scala-vs-python/>



**Differentiating Scala and Python based on Performance**

Scala is ten times faster than Python because of the presence of Java Virtual Machine while Python is slower in terms of performance for data analysis and effective data processing. Python first calls to Spark libraries that involves voluminous code processing and performance goes slower automatically.

At the same time, Scala is good when the number of cores is limited. If they increase in the count, then Scala also start behaving strangely and not liked by the professionals. Here, the question comes performance should be decided based on cores or data processing. Obviously, data processing should be taken as a major deciding factor for performance and there is no doubt that Scala delivers better performance than python for big data Apache Spark projects.

**Differentiating Scala and Python based on the Learning Curve**

The syntax for Scala is a little bit tricky while Python is easy to learn due to simple syntax and standard libraries. Data professionals have to be extremely cautious while working with Scala. The syntax errors are quite common that can make you crazy sometimes. The libraries are hard to define and they are difficult to be understood by beginners or new programmers.

For a professional developer, not only syntax, but code readability is also taken utmost requirement. There are only few Scala developers that are able to understand this tough programming for big data projects.

At the same time, Python is easy to learn due to simpler syntax and availability of standard libraries, but it cannot be taken as an ideal choice for highly scalable systems like Twitter or SoundCloud. The above discussion concludes that learning a tough language like Scala not only increases developer efficiency, but optimized overall programming functionality too.

**Differentiating Scala and Python based on Concurrency**

Based on the complexity of big data systems, there is quick need of programming language that can integrate various database programs or services together. Scala enjoys high preference here offering multiple standard libraries and core that helps in quick integration of databases in the big data ecosystem.

With Scala, developers can write more efficient, maintainable, and readable code with multiple concurrency primitives. At the same time, Python does not support concurrency and multithreading well. If you are using Python for big data projects, there is only one CPU active in the python process during that particular time interval.

In case, you are interested in deploying new code to the system, then there is an emergency need that multiple processes should be initiated for effective memory management and data processing. Python fails here when it comes to multi-threading and concurrency while Scala has been proved more efficient and easy language to handle these workloads.

**Differentiating Scala and Python based on Type Safety**

When developing code for Apache Spark projects, it needs to be continuously re-factored by the developers. Scala is a statically-typed language provides an interface to catch compile-time errors. Refactoring code in Scala is hassle-free and easier experience than a dynamically-typed language likes Python.

Python language is highly prone to bugs every time you make changes to the existing code. This is always better to use Scala for big data projects wherever scalable code is the primary requirement. Python can be used for small-scale projects, but it does not provide the scalable, feature that may affect productivity at the end.

**Differentiating Scala and Python based on Usability**

When it comes to usability, both Scala and Python are equally expressive and you may achieve desired functionality as required for big data projects. Python is taken more user-friendly language than Scala and it is less verbose too, that makes it easy for the developers to write code in Python for Apache Spark projects. Usability is considered as a subjective factor because it depends on the personal choice of programmer which programming language he likes the most.

**Differentiating Scala and Python based on Advanced Features**

Scala has various existential types, implicit, and macros. The syntax with advanced features may be little hard as compared to usual functions. If we talk about the professionals then Scala is always more powerful in terms of framework, libraries, implicit, macros etc.

At the same time, Python is taken primary choice for **NLP (Natural Language Processing)** while Scala does not have that many tools to work machine learning and NLP. The discussion clearly concludes that it completely depends on the nature of the project and it’s processing requirement which programming language you prefer the most.

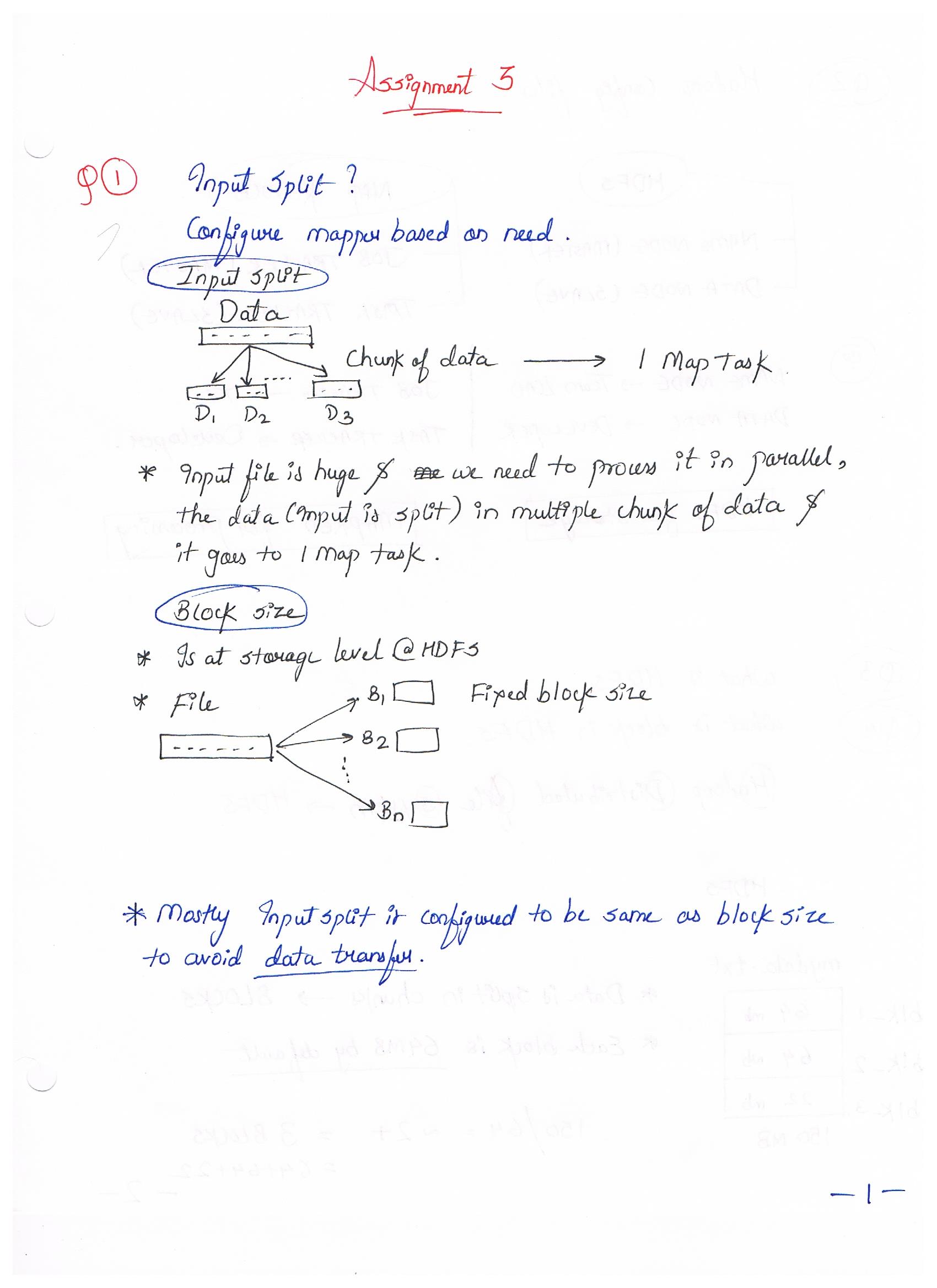
* **Python**: NLP and machine learning, Python is the best choice
* **Scala**: stream, streaming, implicit, macros go well with Scala programming language.

# Lecture 21,22: Scala Coding

Refer code from DataBricks

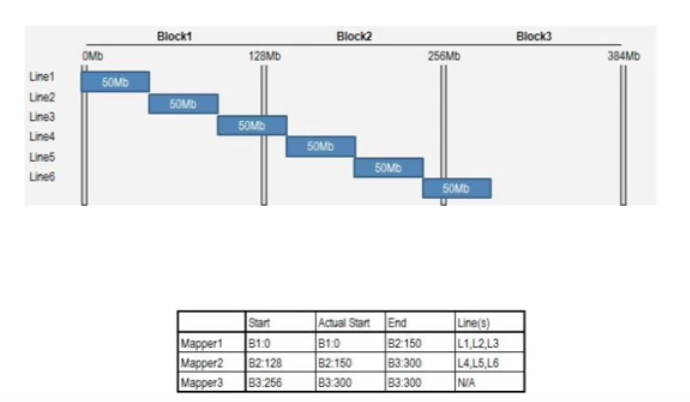
Scala - > scala\_ zak

# Assignment Zak: MapReduce, Combiner, Partitioner, Shuffle, HDFS, Git

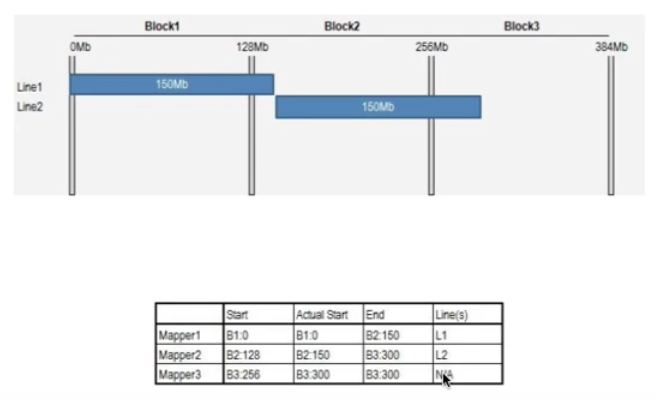
1. What is **input split** and how to configure mappers based on need?  
     
   <https://www.youtube.com/watch?v=weTqF4whlYY>  
     
   

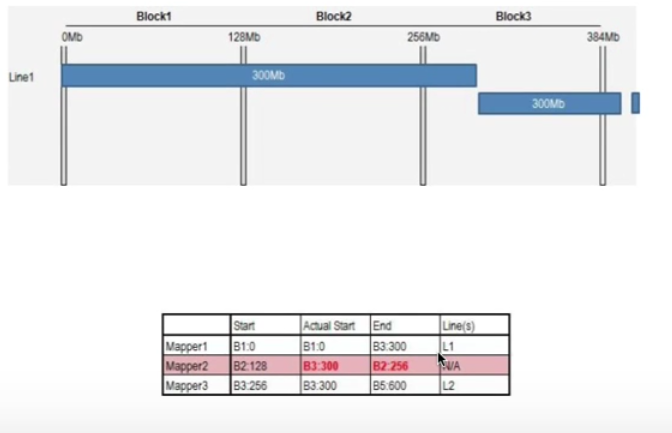
**Mapper example 1**

File is composed on 6 Lines of 50 MB each



* Record reader reads 1 line at a time
* Offset as key and content as the task
* Mapper 1 reads the 1st Block within the splits
* Mapper 1: Reads all the records within the split-boundaries. Therefore, it reads Record 1, record 2 and record 3 from Block 1.
* Mapper 3: Will not read a single record. Because, there are no records left to read.

**Mapper example 2**

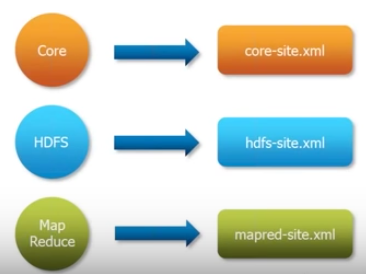
**Mapper example 3**

* Mapper 1: Reads record 1.
* Mapper 2: Tries to find a new record. No records exist therefore, does not process any records

1. Hadoop configuration files in details.  
   <https://www.edureka.co/blog/explaining-hadoop-configuration/>

<https://youtu.be/By_bibVn_x0>

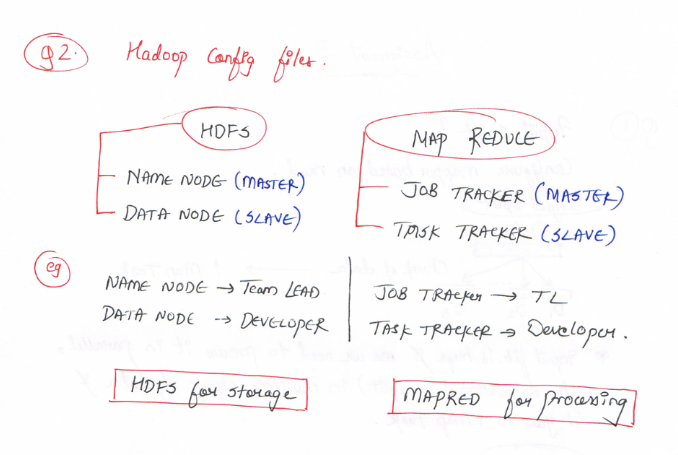
**Core configuration file**

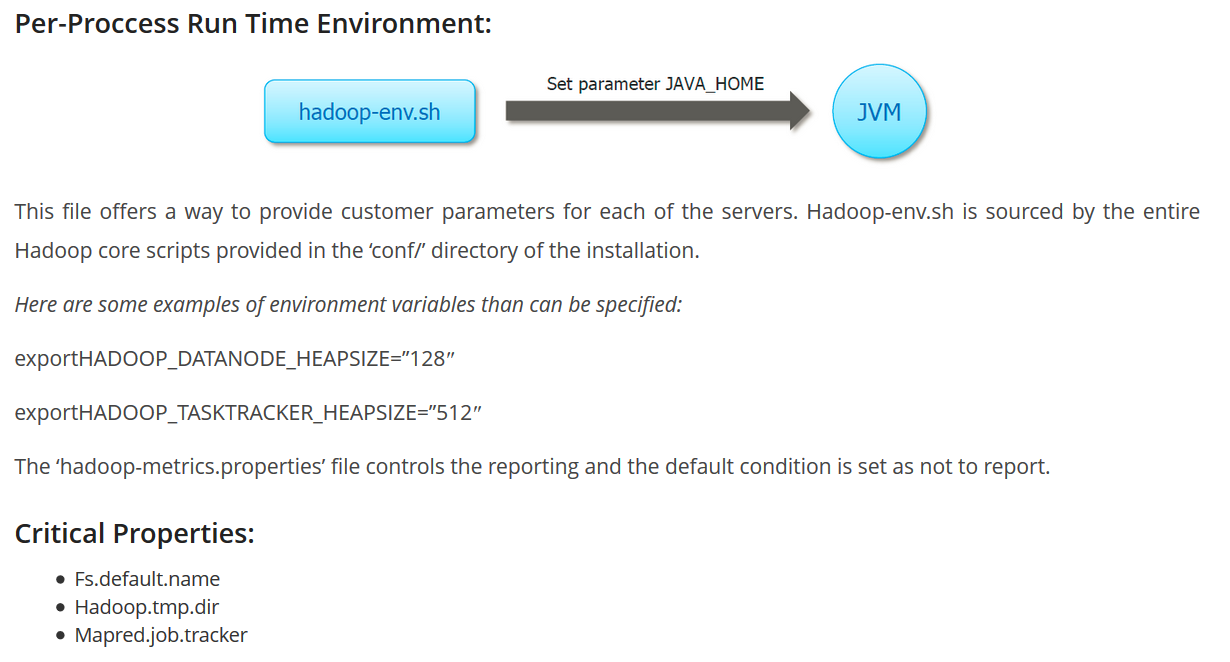


**Core-site.xml file**The core-site.xml file informs Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.

**hdfs-site.xml file**The hdfs-site.xml file contains the configuration settings for HDFS daemons; the NameNode, the Secondary NameNode, and the DataNodes. Here, we can configure hdfs-site.xml to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created. The default is used if replication is not specified in create time.

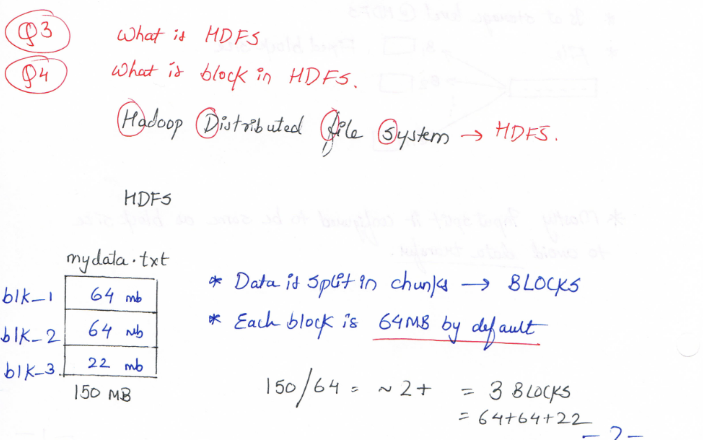
**mapred-site.xml file**The mapred-site.xml file contains the configuration settings for MapReduce daemons; the job tracker and the task-trackers

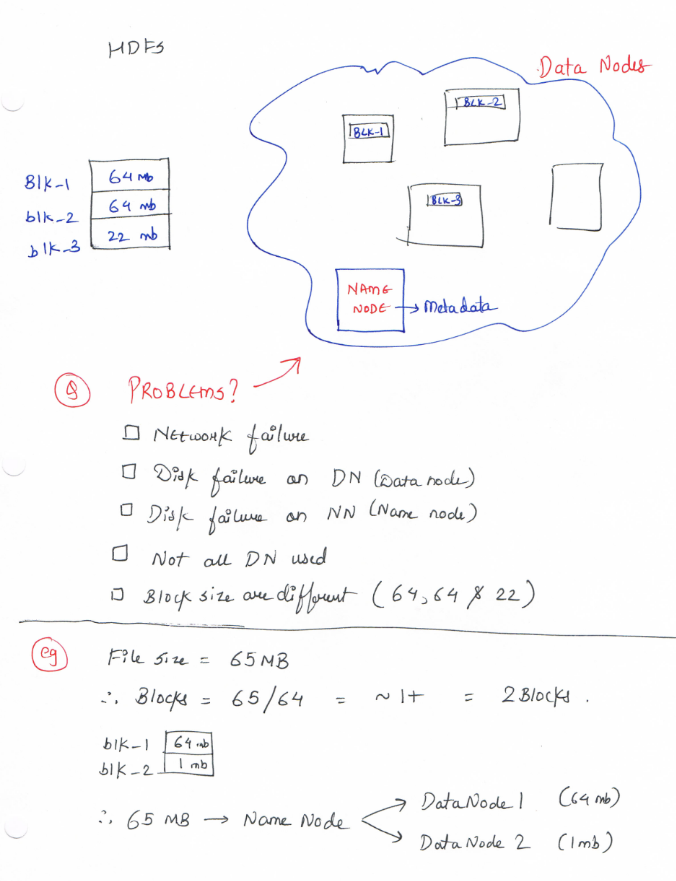




1. What is HDFS?
2. What is block in HDFS

<https://youtu.be/vdkx2xasGlM>





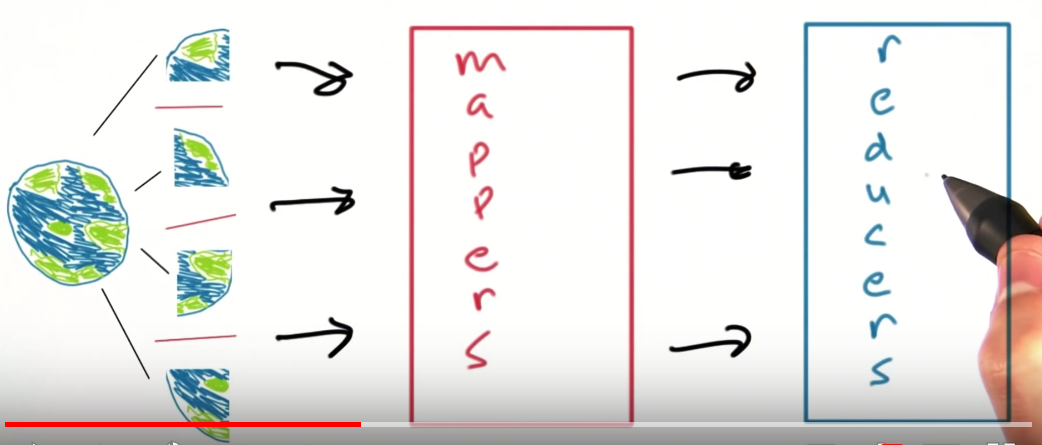
1. What is MapReduce?
2. Different stages of MapReduce?

<https://youtu.be/gI4HN0JhPmo>

<https://youtu.be/PhdRyrmbRYQ>

**MapReduce:**

MapReduce is a parallel programming module to process large data-sets across cluster of computers





* We can use hash tables and have;

City Name as --> Key

Amount as --> Value

* But due to huge data set, it will take a significant time and our machine may run out of memory/crash.
* Therefore, we use **MapReduce**



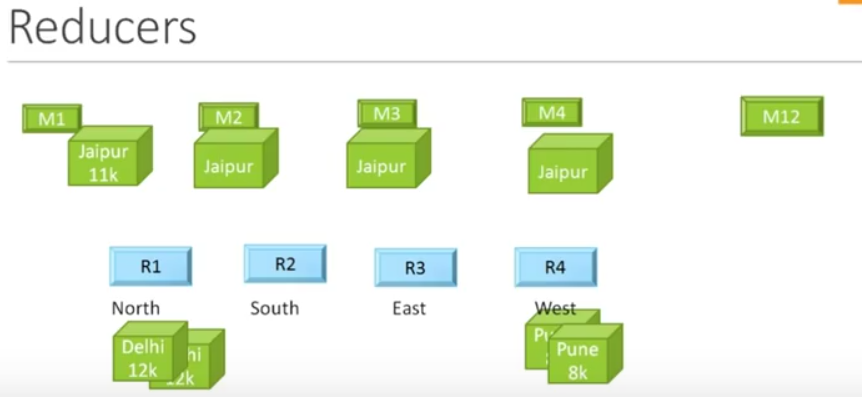
**Mapper**: Splits the entire data in chunks of months.

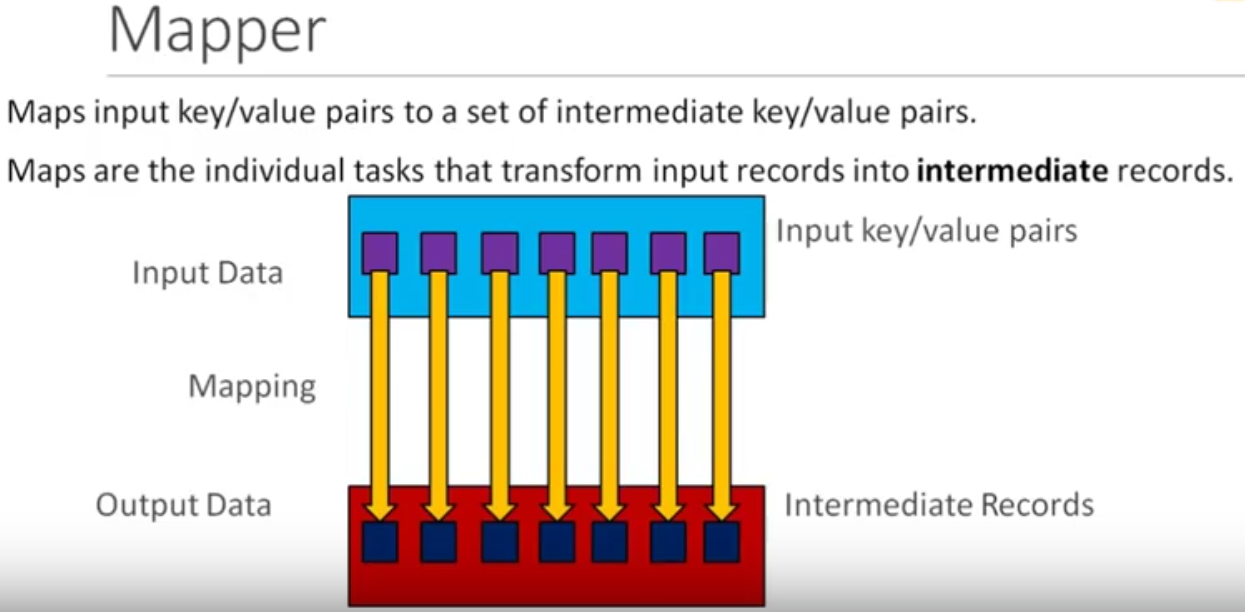
* Therefore, processes parallelly

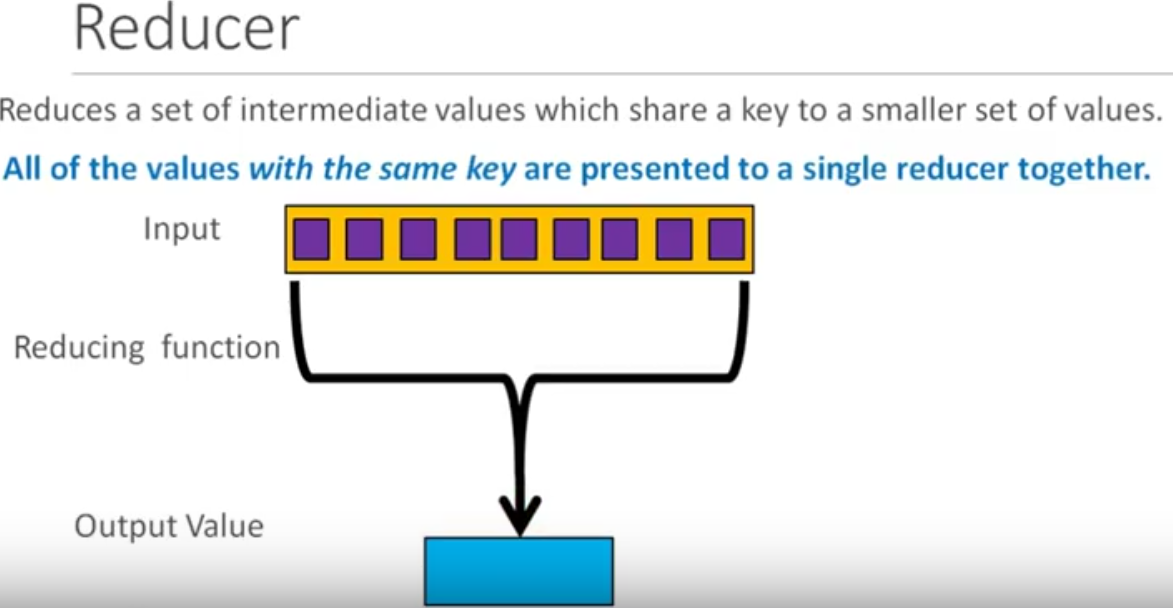




**Reducer**: We tell the reducer to be responsible for each region







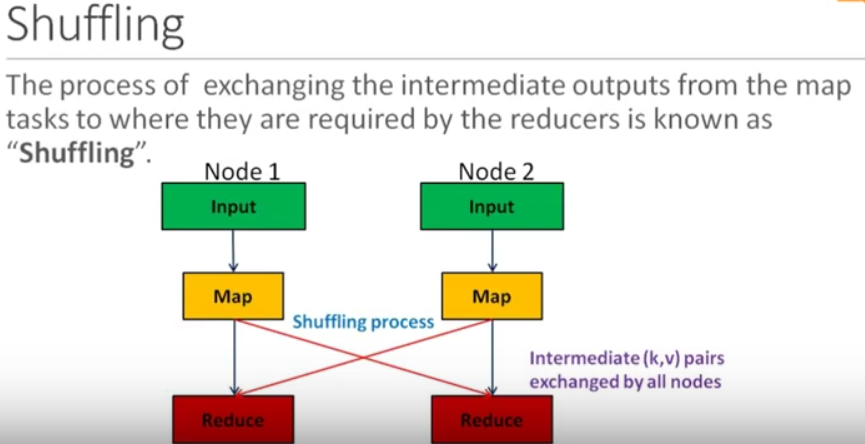
1. How shuffle works in MapReduce?

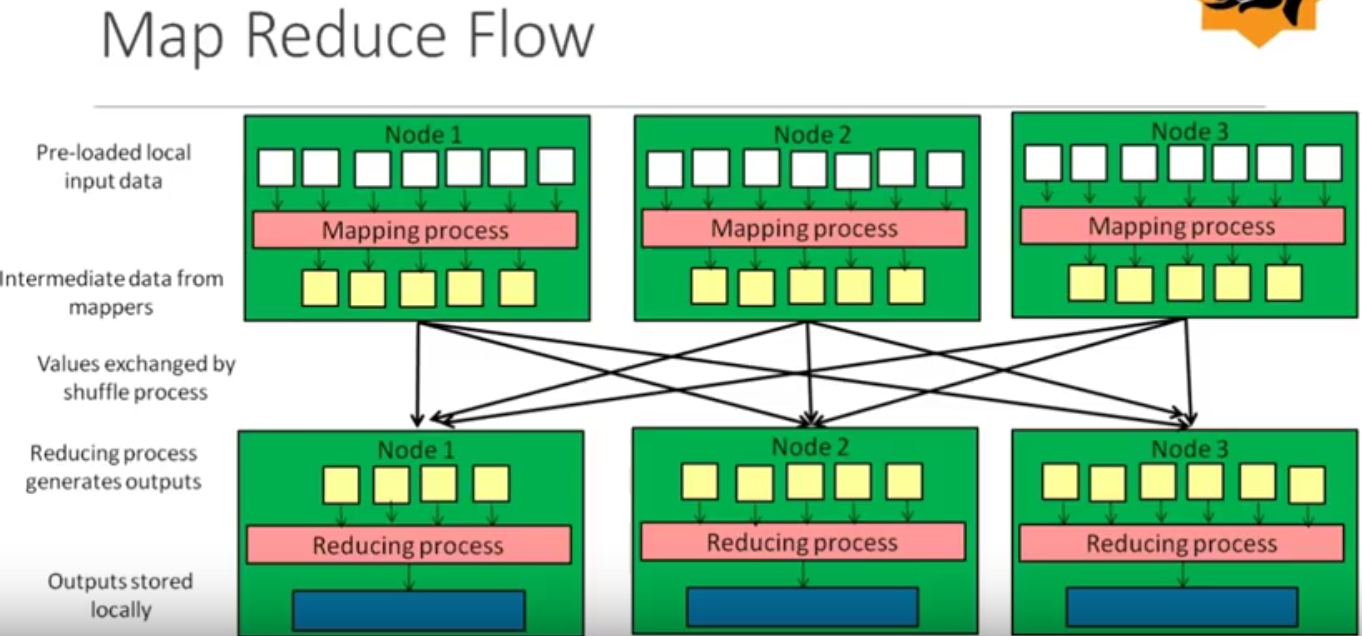
Sorting in MapReduce?

**Shuffling** is the process by which **intermediate data from mappers are transferred to 0,1 or more reducers**.

Movement of intermediate records between Mapper and Reducer as required.

**Sorting**: Once the shuffling is complete, the reducer sorts the records





1. Where is mapper output written?

**OutputCollector** is a generalization of the facility provided by the MapReduce framework to collect data output by the Mapper or the Reducer (either the intermediate outputs or the output of the job).

The output of the maps jobs is stored in the **local disk of the mappers**. Once the map job finishes these local outputs are then transferred to reducers.

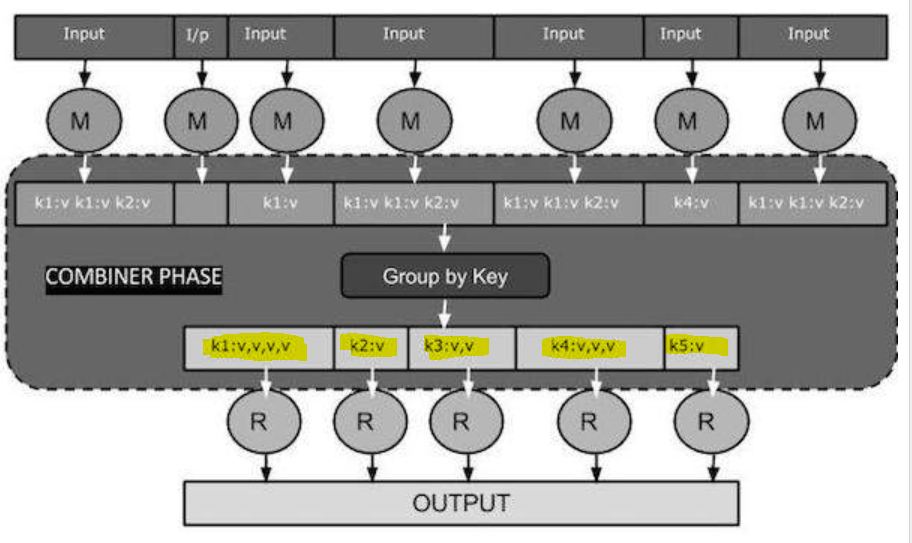
* You can check your $HADOOP\_HOME/conf/mapred-site.xml to check where mapper outputs are stored.

1. What is combiner?

<https://www.tutorialspoint.com/map_reduce/map_reduce_combiners.htm>

A Combiner, also known as a **semi-reducer**, is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.

The main function of a Combiner is to summarize the **map output records with the same key**. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.



1. What is Partitioner in MapReduce?

<https://data-flair.training/blogs/hadoop-partitioner-tutorial/>

**Partitioning** of the keys of the intermediate map output is controlled by the Partitioner. By hash function, key (or a subset of the key) is used to derive the partition. According to the **key-value** each mapper output is partitioned and records having the same key value go into the same partition (within each mapper), and then each partition is sent to a reducer. Partition class determines which partition a given (key, value) pair will go.

* Partition phase takes place after map phase and before reduce phase.

Q) How many Partitioners in Hadoop?

The total number of Partitioners that run in Hadoop is equal to the number of reducers

i.e. Partitioner will divide the data according to the number of reducers which is set by JobConf.setNumReduceTasks() method.

* Thus, the data from single partitioner is processed by a single reducer.
* And partitioner is created only when there are multiple reducers.

Q) Hadoop Default Partitioner

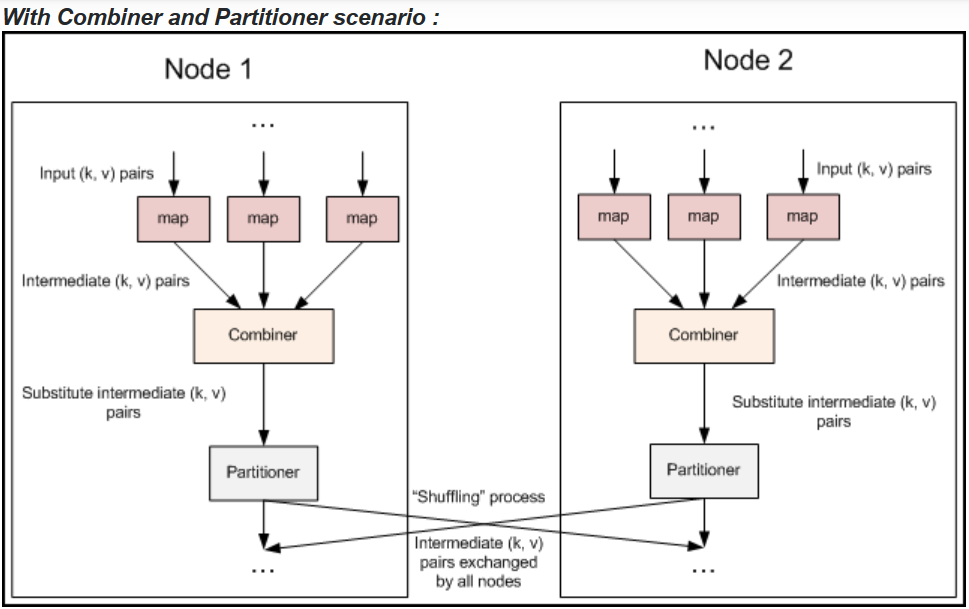
Default partitioner in MapReduce Hadoop is **Hash Partitioner** which computes a hash value for the key and assigns the partition based on this result.

Combiner vs. Partitioner

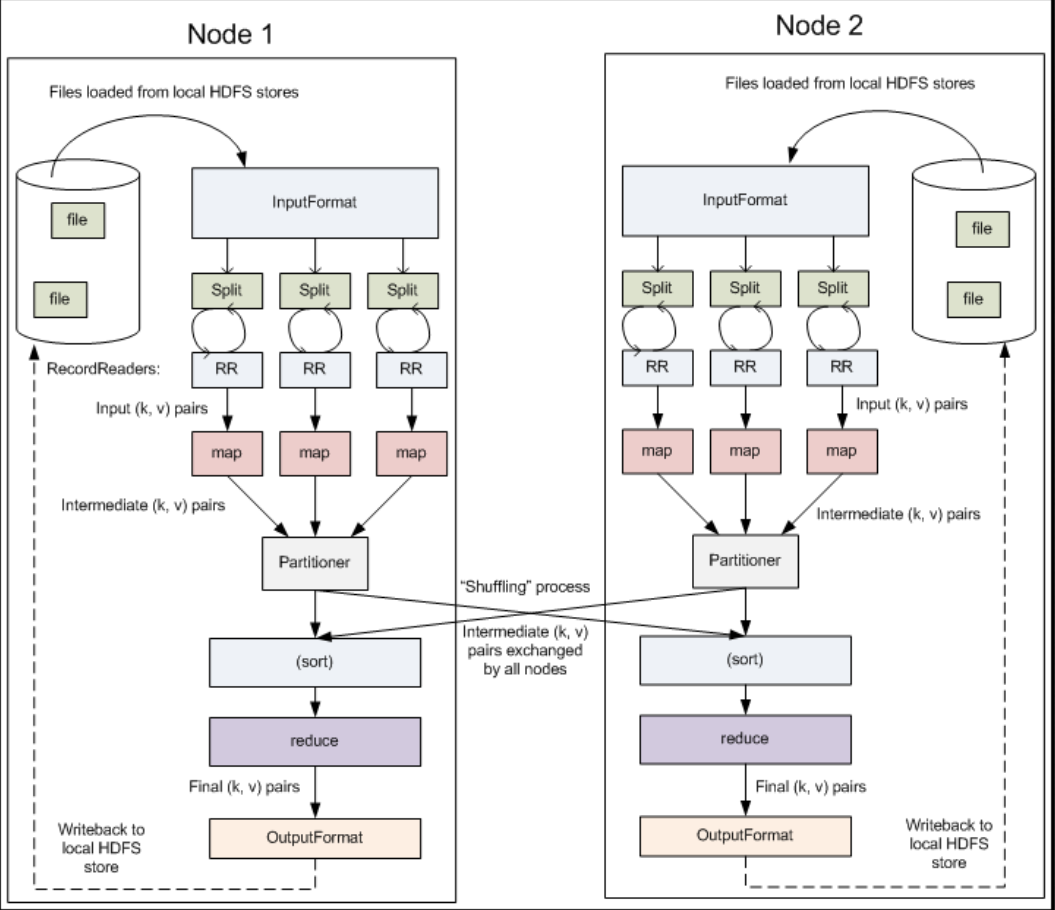
<https://stackoverflow.com/questions/38562889/difference-between-combiner-and-partitioner>

**Combiner** can be viewed as mini-reducers in the map phase. They perform a local-reduce on the mapper results before they are distributed further. Once the Combiner functionality is executed, it is then passed on to the Reducer for further work.

**Partitioner** come into the picture when we are working on more than one Reducer. So, the partitioner decide which reducer is responsible for a particular key. They basically take the Mapper Result (if Combiner is used then Combiner Result) and send it to the responsible Reducer based on the key



**With Partitioner only scenario:**



1. What is GitHub and GitHub commands.

­

<https://github.com/joshnh/Git-Commands>

GitHub is a Web-based Git version control repository hosting service. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features.

* GitHub is developer community

**Getting & Creating Projects**

| **Command** | **Description** |
| --- | --- |
| git init | Initialize a local Git repository |
| git clone ssh://git@github.com/[username]/[repository-name].git | Create a local copy of a remote repository |

**Basic Snapshotting**

| **Command** | **Description** |
| --- | --- |
| git status | Check status |
| git add [file-name.txt] | Add a file to the staging area |
| git add -A | Add all new and changed files to the staging area |
| git commit -m "[commit message]" | Commit changes |
| git rm -r [file-name.txt] | Remove a file (or folder) |

**Branching & Merging**

| **Command** | **Description** |
| --- | --- |
| git branch | List branches (the asterisk denotes the current branch) |
| git branch -a | List all branches (local and remote) |
| git branch [branch name] | Create a new branch |
| git branch -d [branch name] | Delete a branch |
| git push origin --delete [branchName] | Delete a remote branch |
| git checkout -b [branch name] | Create a new branch and switch to it |
| git checkout -b [branch name] origin/[branch name] | Clone a remote branch and switch to it |
| git checkout [branch name] | Switch to a branch |
| git checkout - | Switch to the branch last checked out |
| git checkout -- [file-name.txt] | Discard changes to a file |
| git merge [branch name] | Merge a branch into the active branch |
| git merge [source branch] [target branch] | Merge a branch into a target branch |
| git stash | Stash changes in a dirty working directory |
| git stash clear | Remove all stashed entries |

**Sharing & Updating Projects**

| **Command** | **Description** |
| --- | --- |
| git push origin [branch name] | Push a branch to your remote repository |
| git push -u origin [branch name] | Push changes to remote repository (and remember the branch) |
| git push | Push changes to remote repository (remembered branch) |
| git push origin --delete [branch name] | Delete a remote branch |
| git pull | Update local repository to the newest commit |
| git pull origin [branch name] | Pull changes from remote repository |
| git remote add origin ssh://git@github.com/[username]/[repository-name].git | Add a remote repository |
| git remote set-url origin ssh://git@github.com/[username]/[repository-name].git | Set a repository's origin branch to SSH |

**Inspection & Comparison**

| **Command** | **Description** |
| --- | --- |
| git log | View changes |
| git log --summary | View changes (detailed) |
| git diff [source branch] [target branch} | Preview changes before merging |