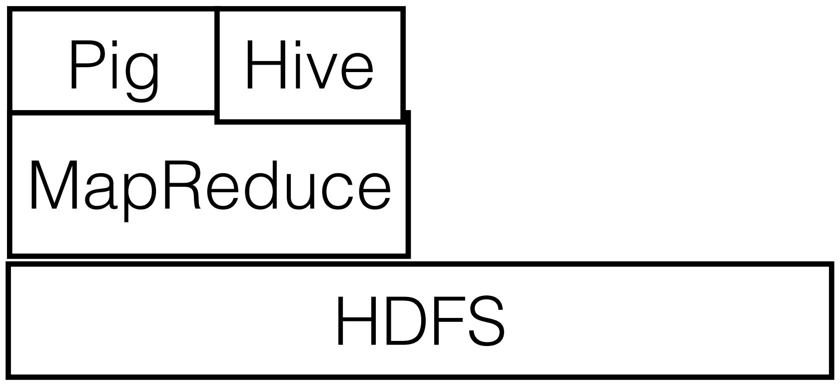
Pig

Data Access Components

- Data Access is an extremely important aspect in any project
- As we deal with Big Data for processing data
  - We perform ad hoc processing to get insights of data and design strategies.
- Hadoop's basic processing layer is MapReduce
  - A massively parallel processing framework that is scalable, faster, adaptable, and fault tolerant.
  - MapReduce is the key to perform processing on Big Data
  - MapReduce has a high learning curve,
    - In some cases can be complex to understand, design, code
    - Requires good programming skills to master.

Data Access Components

- Abstraction layers
  - Hive and Pig provide a user friendly language for faster development and management.
- Hive and Pig are quite useful and handy when it comes to ad hoc analysis



Pig

- Pig is a platform which has the abstraction wrapper of Pig Latin language on top of MapReduce.

- Pig was developed by Yahoo! around 2006
    - contributed to Apache as an open source project.

- Pig Latin is a data flow language
    - more comfortable for a procedural language developer or user.

- Pig can help manage the data in a flow:
    - Ideal for the data flow process, ETL or the ELT process ad hoc data analysis.

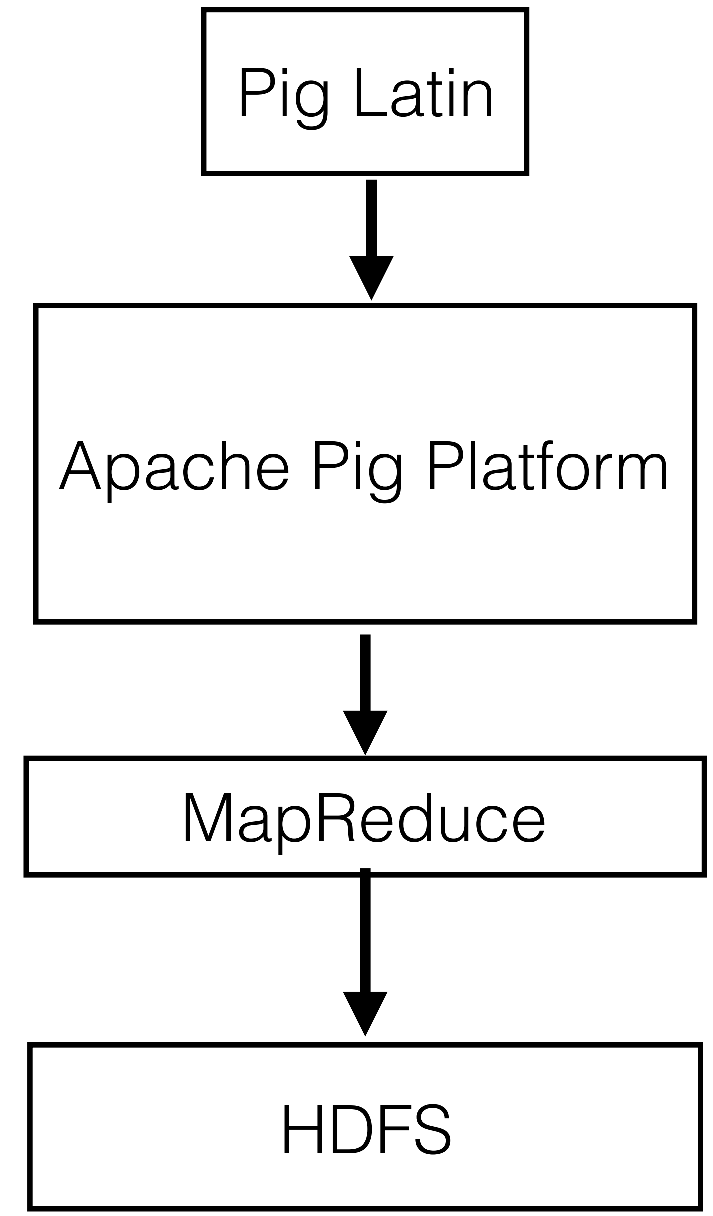
- Easier to use for structured and semi-structured data analysis.

Pig Architecture

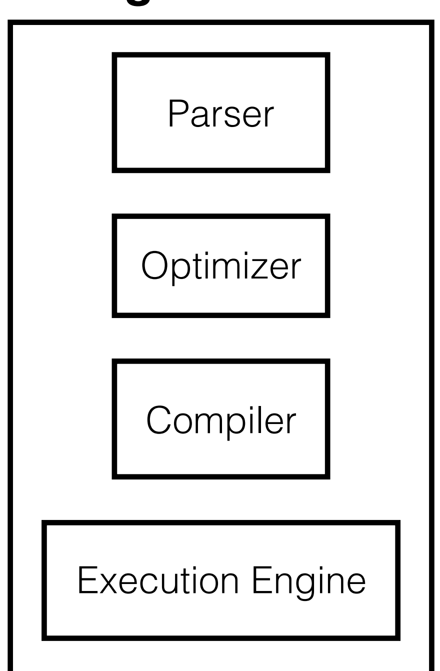
- Pig Latin is used for writing queries and simple scripts

- The Pig data flow architecture is layered for:

    - Transforming Pig Latin statements to MapReduce steps.



Pig Platform



Pig Architecture

- Three main phases in compiling and executing a Pig script:
  - Logical plan
  - Physical plan
  - MapReduce plan

Pig Architecture

- The logical plan
  - Pig statements are parsed for syntax errors and validation of the input files and input data structures.
  - A logical plan is created for each statement (line) in the pig latin script.
  - Each line is parsed for any syntactical errors
    - If no errors are found.. create a logical execution plan
    - With each line in the script, logical plan gets bigger.
  - The logical plan has a one-to-one correspondence with the operators.
  - Optimizations based on in-built rules happen at this stage.

Pig Architecture

- The physical plan
  - A translation of each operator into the physical form of execution happens during this stage.
  - For the MapReduce platform, except for a few, most operators have a one-to-one correspondence with the physical plan.
  - There are a few physical operators:
    - Local Rearrange (LR)
    - Global Rearrange (GR)
    - Package (P)
  - Logical Operators (GROUP, COGROUP, JOIN) translated into sequence of:
    - LR, GR, P operators.

Pig Architecture

- The MapReduce Plan

  - Final stage of pig compilation is to compile the physical plan to actual MapReduce Jobs

  - A Reduce task is required wherever a LR, GRP, and P sequence is present

Pig Data Types

- Pig has a collection of primitive data types, as well as complex data types.
- Inputs and outputs to Pig's relational operators are specified using these data types:
    - Primitive: int, long, float, double, chararray, and bytearray
    - Map: Map is an associative array data type that stores a char array key and its associated value.
        - The data type of a value in a map can be a complex type.
        - If the type of the value cannot be determined, Pig defaults to the bytearray data type.
        - The key and value association is specified as the # symbol. The key values within a map have to be unique.
        - Syntax: **[key#value, key1#value1…]**

Pig Data Types

- Tuple: A tuple data type is a collection of data values.
  - They are of fixed length and ordered.
  - Tuple is similar to a record in a SQL table, without restrictions on the column types.
  - Each data value is called a field.
  - Ordering of values offers the capability to randomly access a value within a tuple.
  - Syntax: **(value1, value2, value3…)**

Pig Data Types

- Bag: A bag data type is a container for tuples and other bags.
  - They are unordered, that is, a tuple or a bag within a bag cannot be accessed randomly.
  - There are no constraints on the structure of the tuples contained in a bag.
  - Duplicate tuples or bags are allowed within a bag.
  - Syntax: **{(tuple1), (tuple2)…}**

Pig Data Types

- Pig allows nesting of complex data structures
  - You can nest a tuple inside a tuple, a bag, and a Map.
  - Pig Latin statements work with relations, which can be thought of as:
    - A relation (similar to, database table) is a bag
    - A bag is a collection of tuples
    - A tuple (similar to, database row) is an ordered set of fields
    - A field is a piece of data

Pig Modes

- The user can run Pig in two modes:

- Local Mode

  - With access to a single machine, all files are installed and run using a localhost and filesystem.

  - **pig -x local**

- MapReduce Mode

  - This is the default mode, which requires access to a Hadoop cluster.

  - **pig -x mapreduce**

Pig Modes

- In Pig, there are three modes of execution:
- Interactive mode or grunt mode
  - Interactive mode of pig, great for ad-hoc data exploration
- Batch mode or script mode
  - Similar to SQL script file
- Embedded mode
  - Embed Pig commands in a host language such as Python or JavaScript and run the program
- These modes of execution can be either executed in the Local mode or in the MapReduce mode.

Grunt Shell

- Grunt is Pig's interactive shell.
- It is used to enter Pig Latin interactively, provides a shell for users to interact with HDFS.
- For Local mode:
  - Specify local mode using the -x flag:
  - $ pig –x local
  - For MapReduce mode:
    - Point Pig to a remote cluster by placing HADOOP_CONF_DIR on PIG_CLASSPATH.
      - HADOOP_CONF_DIR is the directory containing the hadoop-site.xml, hdfs-site.xml, and mapred-site.xml files.

Loading Data

- For loading your data in Pig, we use the LOAD command and map it to an alias of relation
    - Which can read data from the filesystem or HDFS and load it for processing within Pig.
    - Different storage handlers are available in Pig for handling different types of records by mentioning USING and the storage handler function; few of the frequently used storage handler functions are:
        - PigStorage which is used for structured text files with a delimiter that can be specified and is the default storage handler
        - HBaseStorage which is used for handling data from HBase tables
        - BinStorage which is used for binary and machine readable formats
        - JSONStorage which is used for handling JSON data and a schema that should be specified
        - TextLoader which is used for unstructured data in UTF-8
    - By default, PigStorage will be used by default, and PigStorage and TextStorage will support the compression files gzip and bzip.

Loading Data

- Example:
  - **grunt> movies = LOAD '/user/srafiqi/movies_data.csv' USING PigStorage(',') as (id,name,year,rating,duration);**
- We can use schemas to assign types to fields:
  - **A = LOAD 'data' AS (name, age, gpa); // name, age, gpa default to bytearrays**
  - **A = LOAD 'data' AS (name:chararray, age:int, gpa:float); // name is now a String (chararray), age is integer and gpa is float**

Commands

- Dump
  - The dump command is very useful to interactively view the values stored in the relation
  - grunt> DUMP movies;
  - Dump does not save the data
- Store
  - Used to write or continue with the data.
  - Pig starts a job only when a DUMP or STORE is encountered.
    - We can use the handlers mentioned in LOAD with STORE too.
    - **grunt> STORE movies INTO '/temp' USING PigStorage(','); //This will write contents of movies to HDFS in /temp location**

Commands

- Filter is used to get rows matching the expression criteria.
  - **grunt> movies_greater_than_four = FILTER movies BY (float)rating>4.0;**
  - **grunt> DUMP movies_greater_than_four;**
- We can use multiple conditions with filters and Boolean operators (AND, OR, NOT):
  - **grunt> movies_greater_than_four_and_2012 = FILTER movies BY (float)rating>4.0 AND year > 2012;**
  - **grunt> DUMP movies_greater_than_four_and_2012;**

Commands

- Cogroup
    - Instead of collecting records of one input based on a key, it collects records of n inputs based on a key.
    - The result is a record with a key and a bag for each input. Each bag contains all records from that input that have the given value for the key:

```
$ cat > pets.csv
    nemo,fish
    fido,dog
    rex,dog
    paws,cat
    wiskers,cat>>
```

```
$ cat > owners.csv
        adam,cat
    adam,dog
    alex,fish
    alice,cat
    steve,dog
```

Commands

- grunt> owners = LOAD ‘owners.csv' USING PigStorage(‘,') AS (owner:chararray,animal:chararray);
- grunt> pets = LOAD ‘pets.csv' USING PigStorage(‘,') AS (name:chararray,animal:chararray);
- grunt> grouped = COGROUP owners BY animal, pets by animal;
- grunt> DUMP grouped;

group	owners	pets
cat	{(adam,cat),(alice,cat)}	{(paws,cat),(wiskers,cat)}
dog	{(adam,dog),(steve,dog)}	{(fido,dog),(rex,dog)}
fish	{(alex,fish)}	{(nemo,fish)}

Summary

- Pig is a wrapper of Pig Latin language on top of MapReduce
  - Enable easier and faster development in MapReduce
- Pig is used in the data flow model,
- Transforms the Pig Latin language to the MapReduce job.
- Pig does the transformation in three plans:
  - Logical to Physical to MapReduce,
  - Each plan translates the statements and produces an optimized plan of execution.
  - Pig also has the grunt mode for analyzing data interactively.

Hive

Hive

- Hive provides a SQL-like interface for data stored in Hadoop
- Translates SQL-like commands to MapReduce jobs
- SQL commands in Hive are called HiveQL
- Hive works in data warehouse environment
  - Does not handle transactions
  - Does not provide row-level updates and real-time queries.

Hive Architecture

- Driver
  - Driver manages the lifecycle of a HiveQL statement as it moves through Hive and also maintains a session handle for session statistics.
- Metastore
  - Stores the system catalog and metadata about tables, columns, partitions, and so on.
- Query Compiler
  - It compiles HiveQL into optimized map/reduce tasks.

Hive Architecture

- Execution Engine
  - Executes the tasks produced by the compiler in a proper dependency order.
  - The execution engine interacts with the underlying Hadoop instance.
- HiveServer2
  - Provides a thrift interface and a JDBC/ODBC server
  - Provides a way of integrating Hive with other applications and supports multi-client concurrency and authentication.
- Command Line Interface (CLI), the web UI, and drivers.

Process Flow

- A HiveQL statement can be submitted from:
  - CLI, the web UI, or
  - An external client using interfaces such as thrift, ODBC, or JDBC.
- Compiler parses the query, type check using metadata stored in Metastore.
- The compiler generates a logical plan which is then optimized through a simple rule-based optimizer.
- Finally MapReduce tasks and HDFS tasks is generated.
- The execution engine then executes these tasks in the order of their dependencies by using Hadoop.

Metastore

- The Metastore stores all the details about the tables, partitions, schemas, columns etc.
- Whenever a Hive table is created, table definition is stored in Hive Metastore
- It acts as a system catalog for Hive.
- It can be called from clients from different programming languages, as the details can be queried using Thrift.
- Without Metastore structure design details cannot be retrieved and data cannot be accessed.
- Hive ensures that Metastore is not directly accessed by Mappers and Reducers of a job

Data types

- Hive supports all the primitive numeric data types such as:
    - TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, and DECIMAL.
- Hive also supports string types:
    - CHAR, VARCHAR, and STRING data types.
- Like SQL
    - Timestamp, Date, Boolean, and Binary are also supported
    - . The BOOLEAN and BINARY miscellaneous types are available too.
- Number of complex data types are also available
    - Struct, Map, Array, Union

HiveQL

- Hive supports all the primitive numeric data types such as:
  - TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, and DECIMAL.
- Hive also supports string types:
  - CHAR, VARCHAR, and STRING data types.
- Like SQL
  - Timestamp, Date, Boolean, and Binary are also supported
  - . The BOOLEAN and BINARY miscellaneous types are available too.

DDL

- hive> Create database University

- hive> show databases;

- hive> use shiva;

- hive> Create table person (name STRING , add STRING);

DML

- A file in Hive can be loaded from local, as well as from HDFS; by default Hive will look in HDFS.
    - **hive>LOAD DATA INPATH 'hdfs://localhost:9000/user/hive/srafiqi/SalesData.csv' OVERWRITE INTO TABLE sales;**
- Loading data to table srafiqi.sales
- The preceding command will load data from an HDFS file/directory to the table
    - The process of loading data from HDFS will result in moving the file/directory.
- For Local Data load, use the following code:
    - **hive>LOAD DATA LOCAL INPATH './examples/srafiqi/sales.txt' OVERWRITE INTO TABLE sales;**

Select

- SELECT [ALL | DISTINCT] select_expr, select_expr, FROM table_reference [WHERE where_condition] [GROUP BY col_list] [HAVING having_condition] [CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]] [LIMIT number];

- **hive>select * from person where name = 'Alvin Joyner';**

Summary

- Hive is used by users comfortable with SQL-like development as it has HiveQL.

- The Hive architecture contains:

  - Driver, Metastore, Query compiler, Execution engine, and HiveServer.

- HiveQL has an exhaustive list of

  - Built-in functions and commands to analyze the data.

  - Supports user-defined functions.

Spark Overview

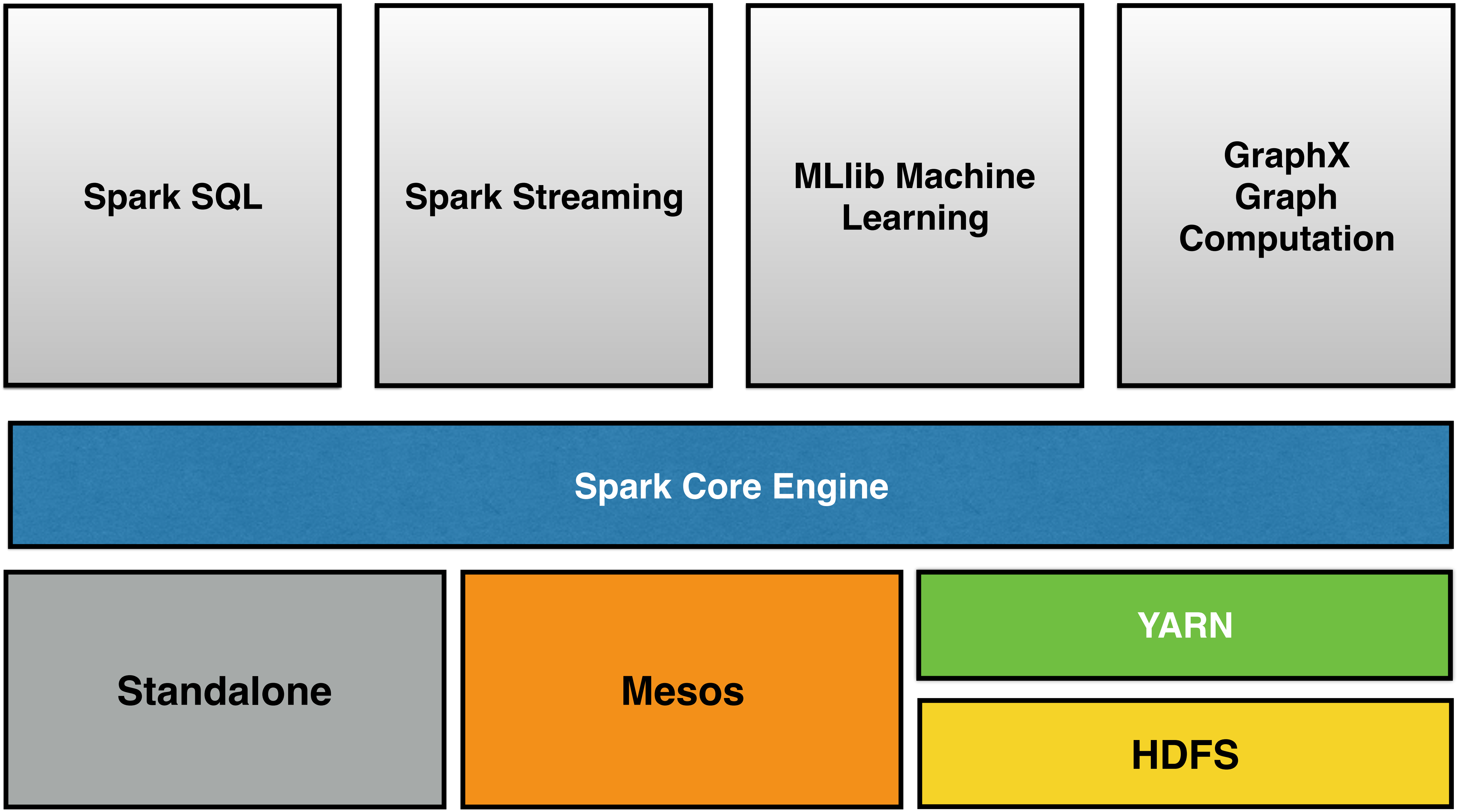
Spark

- Spark is an in-memory based data processing framework
  - Much faster in processing than MapReduce.
  - Spark stores intermediate results in-memory unlike MapReduce
    - Disk-based data access and transfer makes MapReduce slower
  - Spark is an alternative to MapReduce due to the limitations and overheads of the latter, but not as a replacement.
- Spark is widely used for:
  - Streaming data analytics, graph analytics, fast interactive queries, and machine learning.

Features of Spark

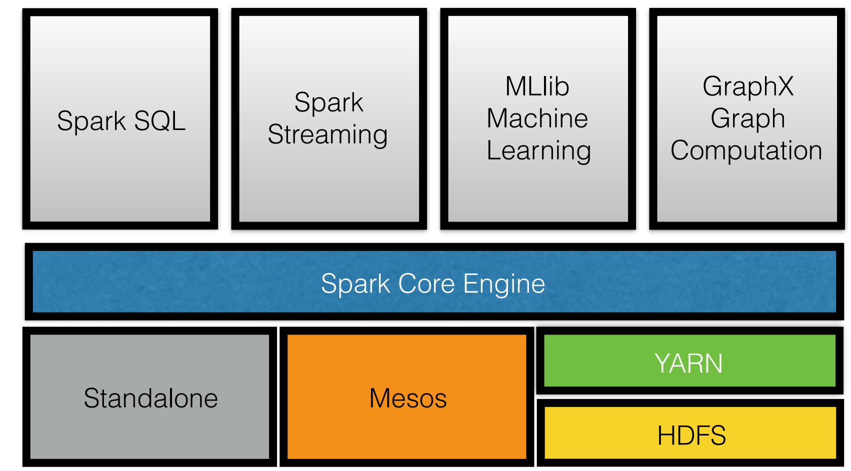
- Runs faster than MapReduce
  - 100 times when running in-memory and 10 times faster when running on disk
- Can process iterative and interactive analytics
- Many functions and operators available for data analysis
- Written in Scala and runs in JVM environment
  - Applications using Spark can be written in Scala, Java, Python, R, Clojure
- Runs in environments such as Hadoop and Mesos, or standalone

Spark Architecture



Spark Framework

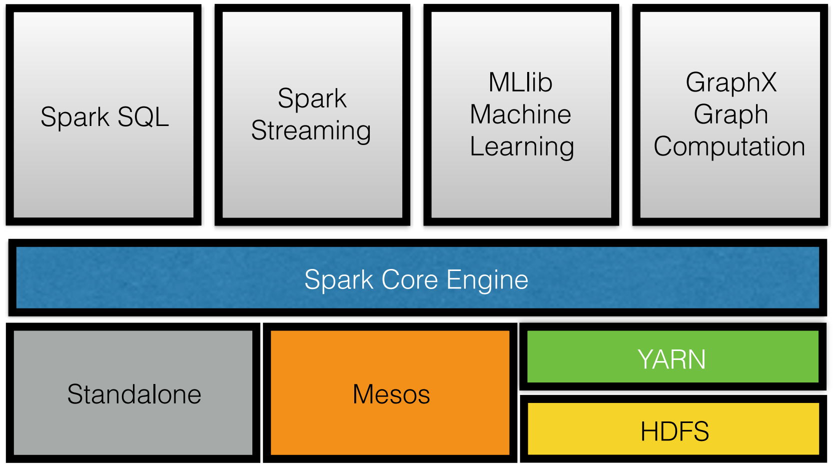
- Spark Core
  - The main component from which all the others are based.
  - Spark Core includes task distribution, scheduling, and the input/output operations.

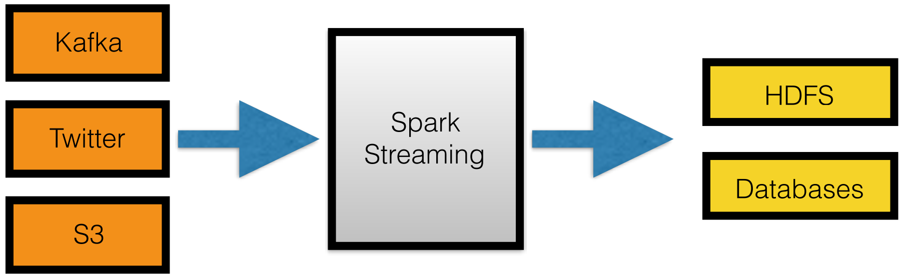


- Spark SQL
  - Exposes Spark dataset over JDBC API and allow running SQL-like queries
  - Wrapper of SQL on top of Spark.
  - It transforms SQL queries into Spark jobs
  - Spark SQL can work with a variety of data sources, such as Hive tables, Parquet files, and JSON files.

Spark Framework

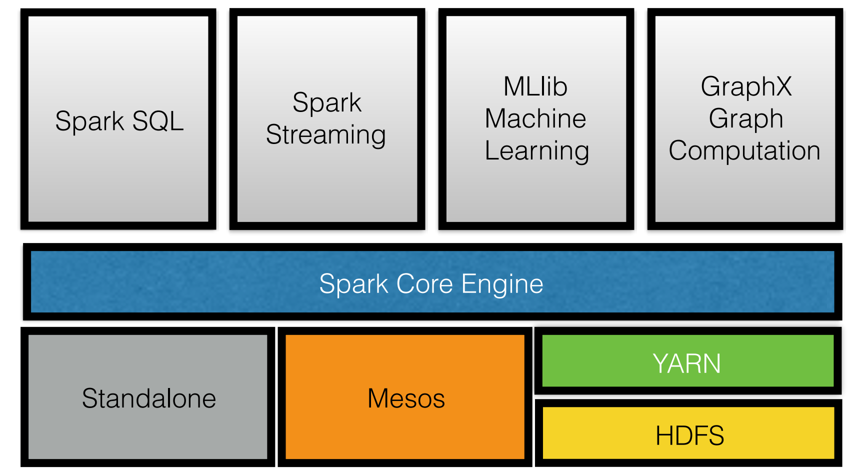
- Spark streaming
  - Library that provides APIs to process streaming data in real time.
  - Spark Streaming is well integrated with many sources, such as:
    - Kinesis, HDFS, S3, Flume, Kafka, Twitter, etc.



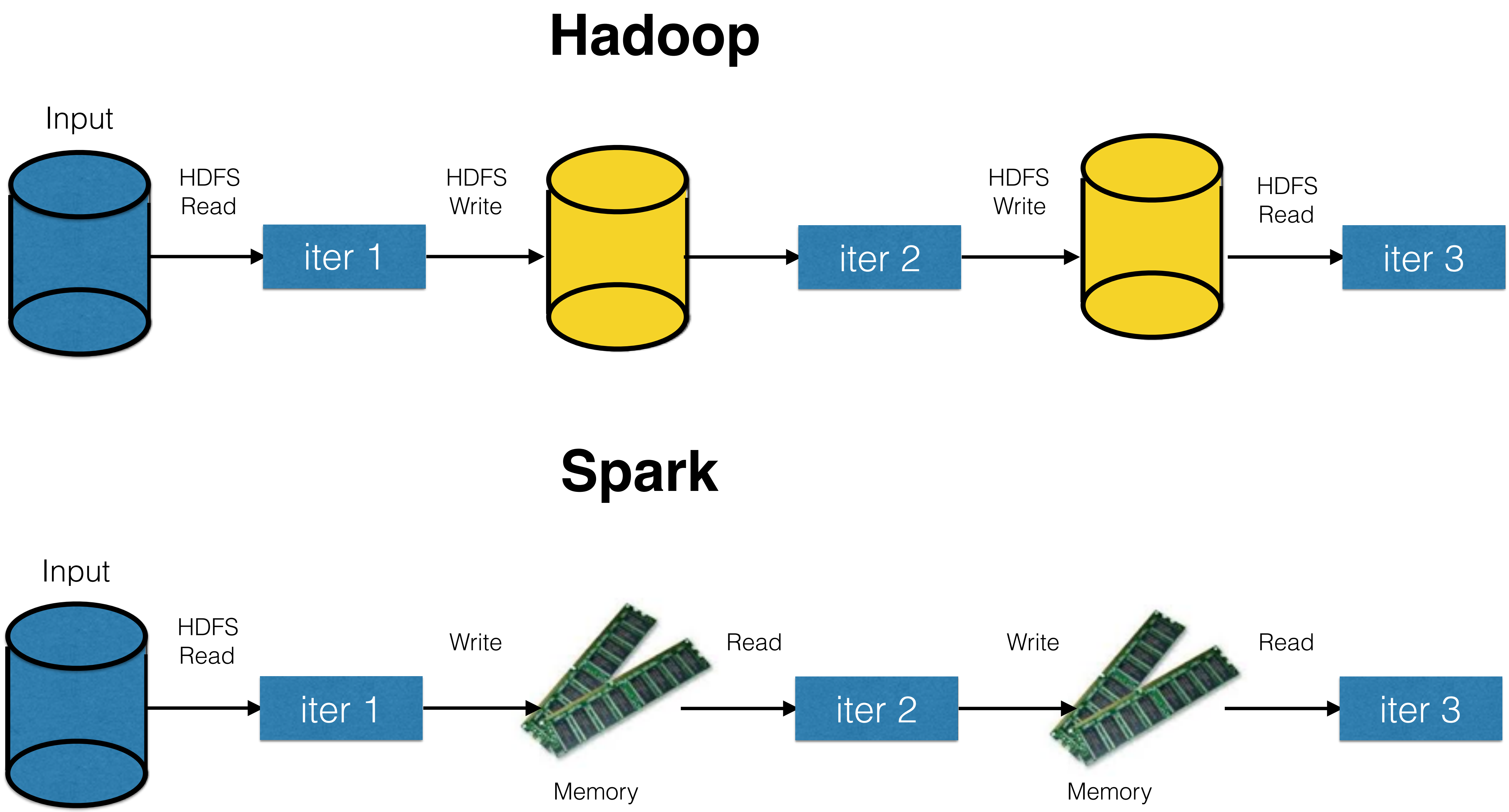


Spark Framework

- MLib
  - Scalable machine learning library including classification, regression and clustering
- GraphX
  - Provides APIs for graph-based algorithms
    - PageRank, Connected components,
    - Label propagation,, Triangle count, and so on.



Data Flow



Sohail Rafiqi

Resilient Distributed Dataset (RDD)

- "***RDDs are distributed memory abstraction that lets programmer perform in-memory operations in large cluster in a distributed environment***"
- An RDD stores the data in-memory as long as possible.
  - If the data grows larger than the threshold, it spills into the disk.
    - Makes computation faster. On the other hand, if some node holding the data in memory fails, then that part of computations has to be processed again
- Resilient
  - Has ability to recompute missing or damaged partitions in case of a failure
  - Achieved by tracking the lineage of transformations applied to the dataset.
- Distributed
  - Data residing in multiple nodes in a cluster

Resilient Data Set

- There are two ways to create RDDs
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in an external storage system
    - Such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.
- Once data is loaded in the RDD two types of operations are performed:
  - Transformations — create the new RDD (e.g., filtering or mapping).
  - Actions can return a value after some executions, such as reduce or count.
- Original RDD remains unchanged
  - Chain of transformation is saved and can be replayed in case of a failure
- Lazy Transformation
  - Transformation is not executed until someone needs the results of transformation

Data Pipeline

- Spark offers integrated data pipeline
  - Combine multitude of inputs with processing to deliver consistent results
  - Without Spark data transformation from various inputs and transformation requires multiple frameworks
  - Sparks provides unique ability to combine:
    - Streaming, Interactive, and and batch workflows
    - Easier to use

Hadoop or Spark

- Is Spark replacing Hadoop
- Is Spark replacing MapReduce
- Benefits of using Spark over Hadoop