Big Data: Hadoop + Python

A practical guide to solving distributed (parallel) processing with modern analytics frameworks

Acknowledgement

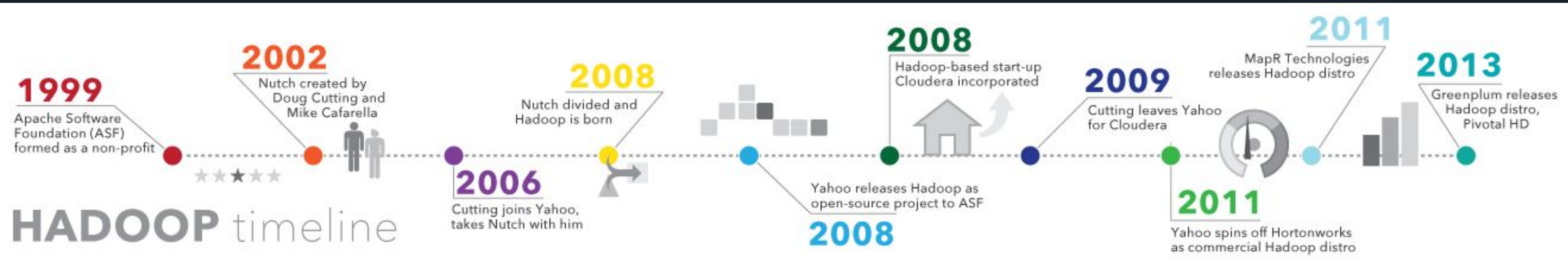
- Most of this material comes from tutorial written by Donald Miner
  - Github: https://github.com/donaldpminer/hadoop-python-tutorial
  - @donaldpminer
  - Email: dminer@minerkasch.com
- I have adapted much of the examples and content, but theme remains mostly the same
- Thank you **Donald!**

Hadoop

What is Hadoop?

- “Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.”
  - https://www.sas.com/en_us/insights/big-data/hadoop.html
- Hadoop cluster includes a master and multiple worker nodes
  - The master node consists of a Job Tracker, Task Tracker, NameNode, and DataNode
  - A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have data-only and compute-only worker nodes.

Timeline



HADOOP timeline

1999
Apache Software Foundation (ASF) formed as a non-profit

2002
Nutch created by Doug Cutting and Mike Cafarella

2006
Cutting joins Yahoo, takes Nutch with him

2008
Nutch divided and Hadoop is born

2008
Yahoo releases Hadoop as open-source project to ASF

2008
Hadoop-based start-up Cloudera incorporated

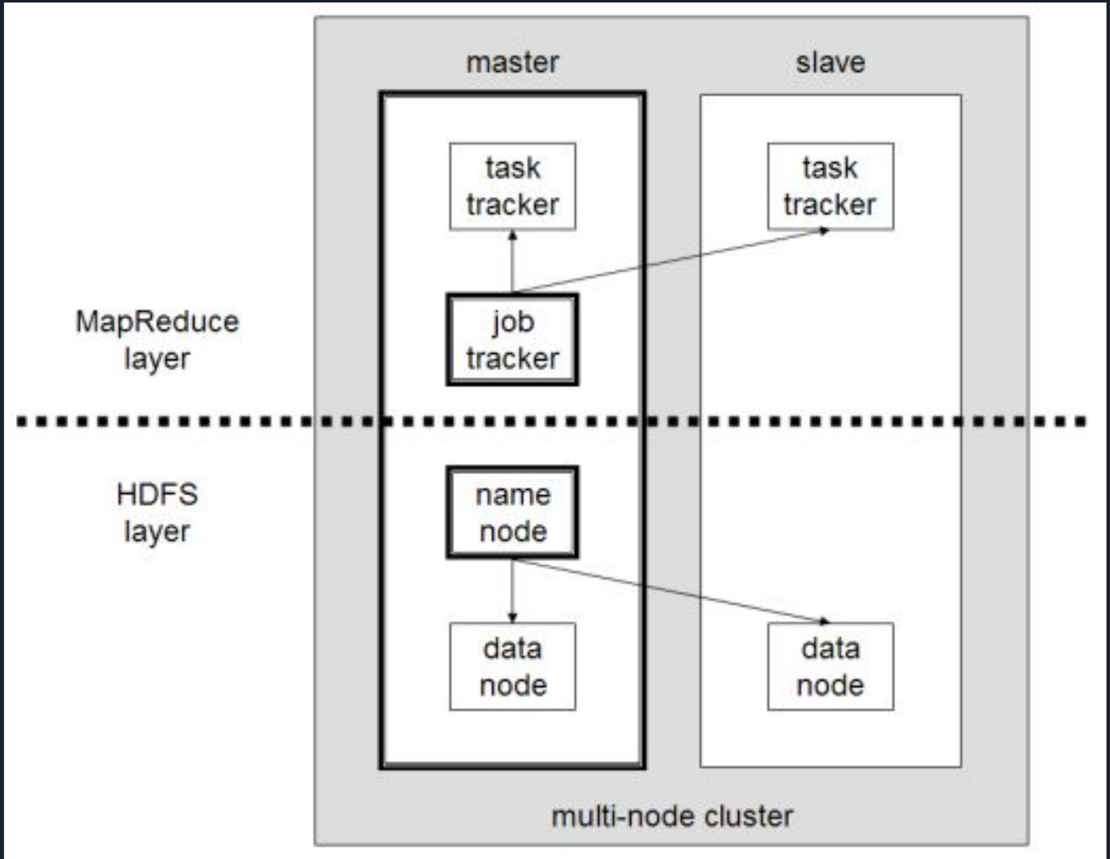
2009
Cutting leaves Yahoo for Cloudera

2011
Yahoo spins off Hortonworks as commercial Hadoop distro

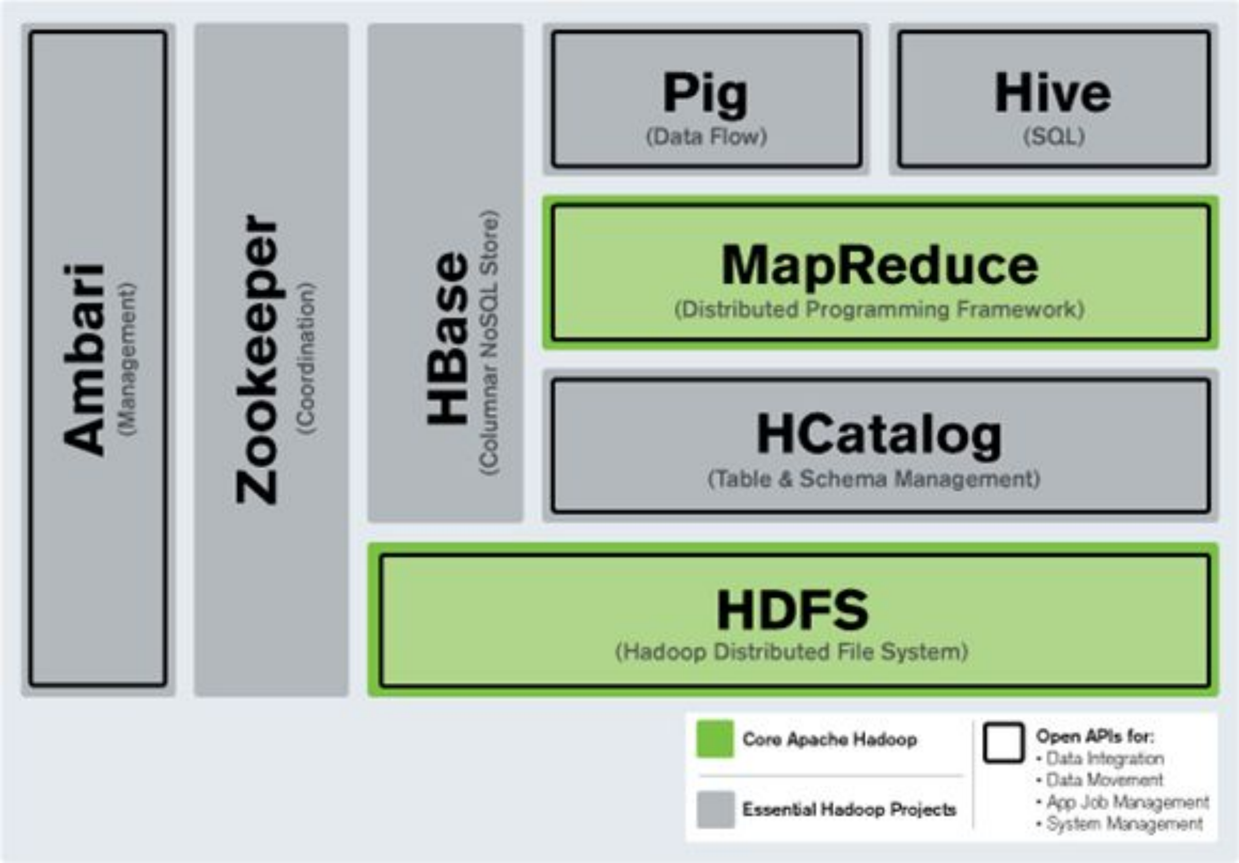
2011
MapR Technologies releases Hadoop distro

2013
Greenplum releases Hadoop distro, Pivotal HD

Master/Slave Relationship



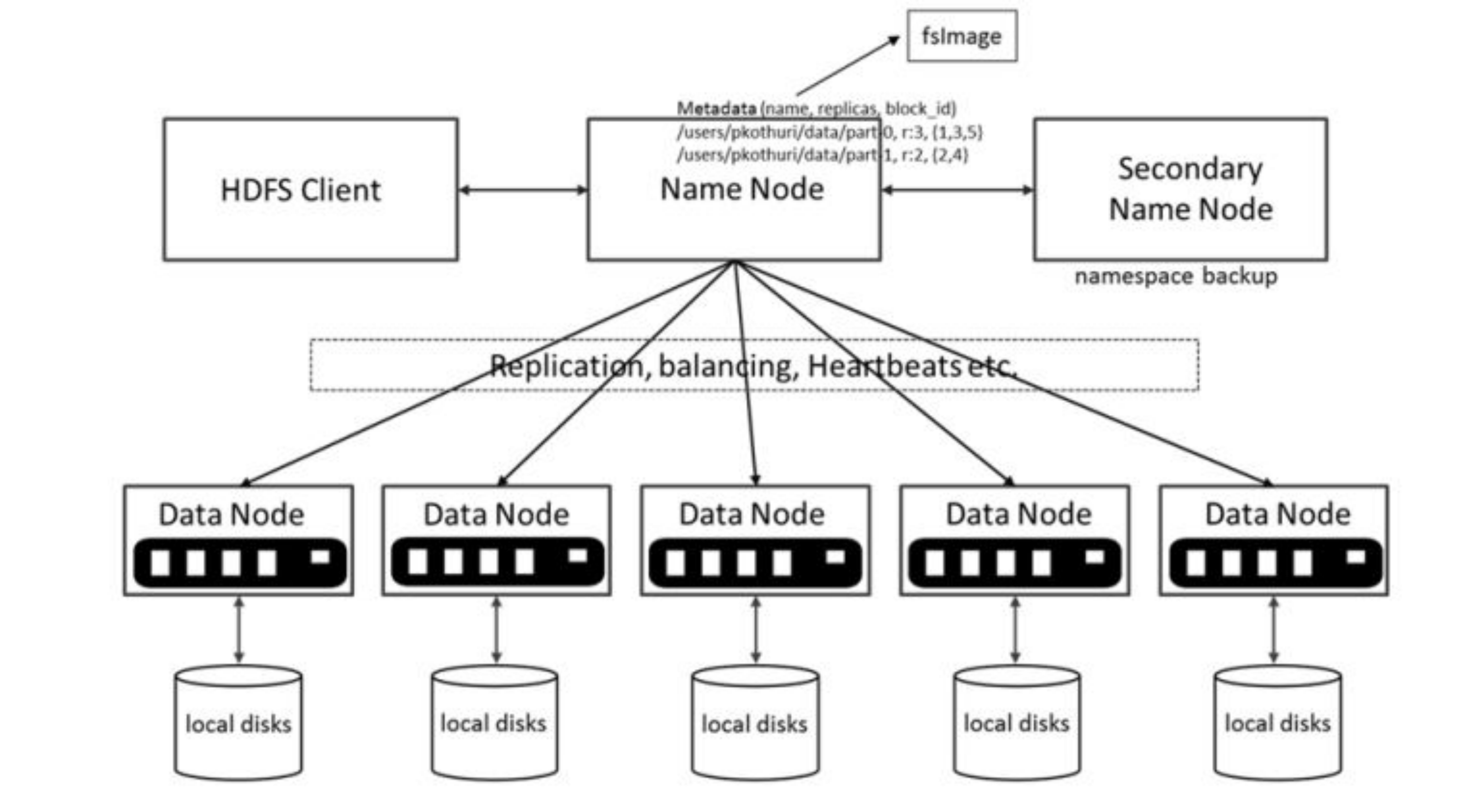
Hadoop v1


Ambari
(Management)
Zookeeper
(Coordination)
HBase
(Columnar NoSQL Store)
Pig
(Data Flow)
Hive
(SQL)
MapReduce
(Distributed Programming Framework)
HCatalog
(Table & Schema Management)
HDFS
(Hadoop Distributed File System)
Core Apache Hadoop
Essential Hadoop Projects
Open APIs for:
• Data Integration
• Data Movement
• App Job Management
• System Management

Hadoop Distributed File System (HDFS)

- Non-POSIX filesystem that stores files in folders
  - Not like your OS filesystem
- Chunks large files (big data) into blocks
  - Typically ~64MB-2GB per block
  - Triplicates data blocks and moves them throughout nodes of Hadoop cluster
- 3x data blocks means 3x storage overhead for every file
  - Great for redundancy, DR, …
  - Bad for storage costs/performance

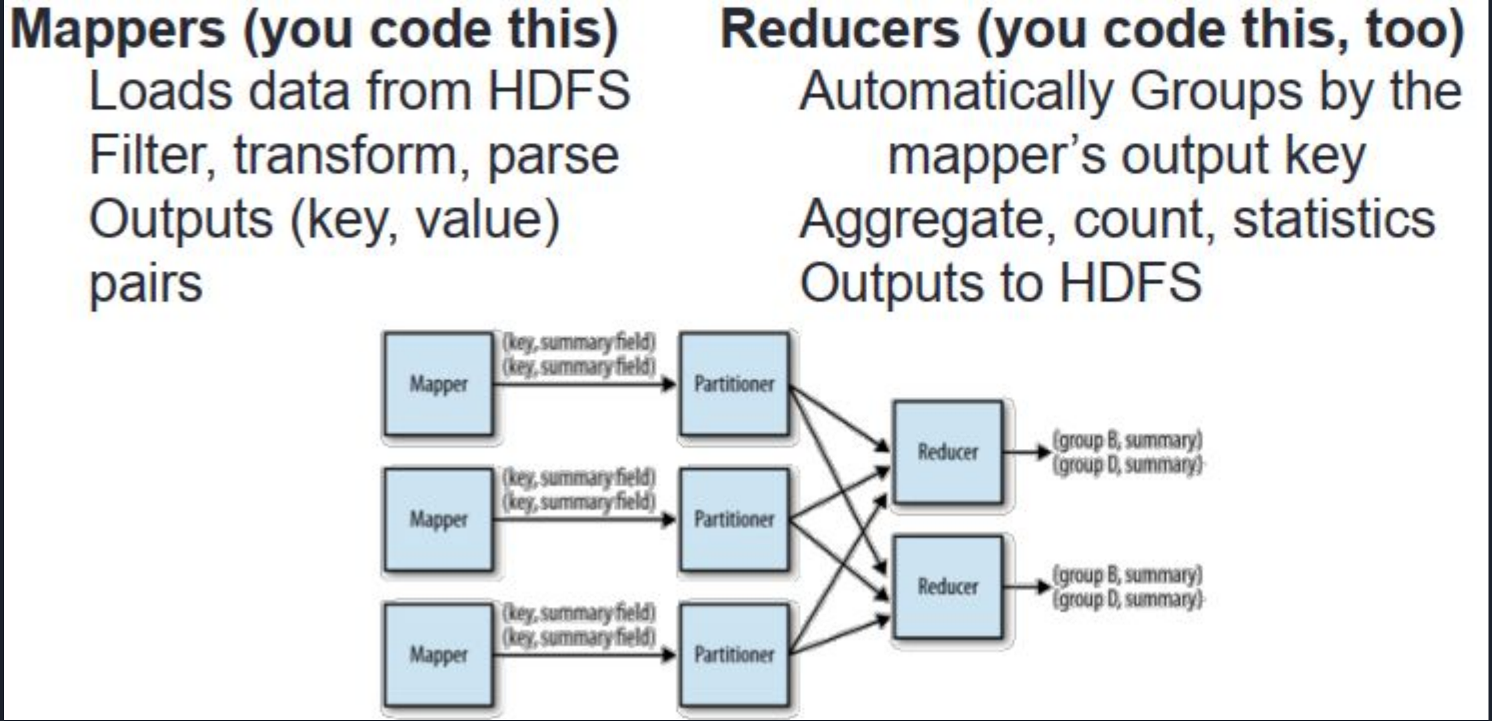
HDFS Diagram



MapReduce

- “It is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions.”
  - https://www.ibm.com/analytics/hadoop/mapreduce
- Analyzes raw data in HDFS where the data is
- Jobs are split into Mappers and Reducers

MapReduce v1



Mappers (you code this)

- Loads data from HDFS
- Filter, transform, parse
- Outputs (key, value) pairs

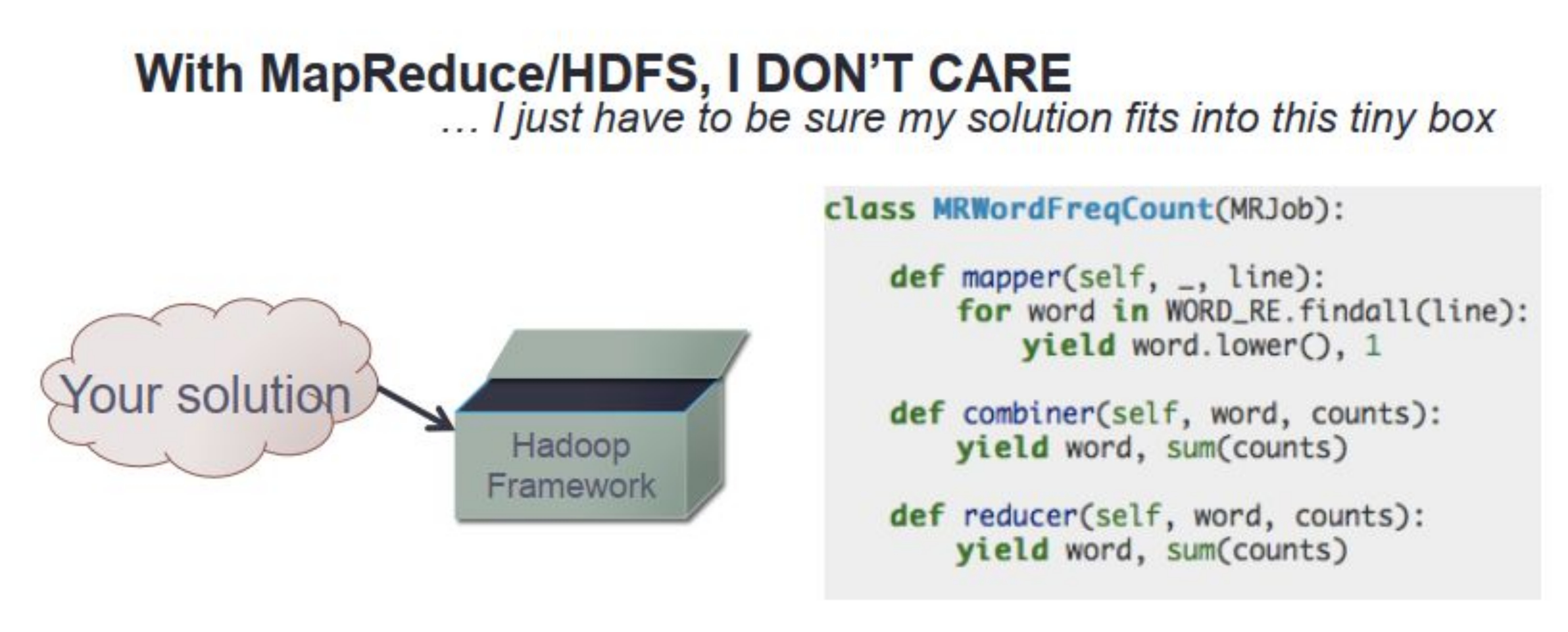
Reducers (you code this, too)

- Automatically Groups by the mapper's output key
- Aggregate, count, statistics
- Outputs to HDFS

Why Choose Hadoop?

- Linear scalability
  - HDFS and MapReduce scale with number of nodes added to cluster
  - If you increase computers by 2x, jobs run 2x faster*
  - If you increase data on cluster by 2x, jobs run 2x slower*
  - If you increase storage by 2x, you can store 2x more data*
- MapReduce frees you to develop schema (data model) on read/execution
  - Before: ETL, schema design, data analysis/study…
  - After: Data is parsed/interpreted as it loads from HDFS (Schema on Read)
- Unstructured Data
  - Keep your data in original (unstructured) format!
  - Have multiple views of the same data!
  - Common framework to build apps!

Not Having to Care (as much...)



With MapReduce/HDFS, I DON'T CARE

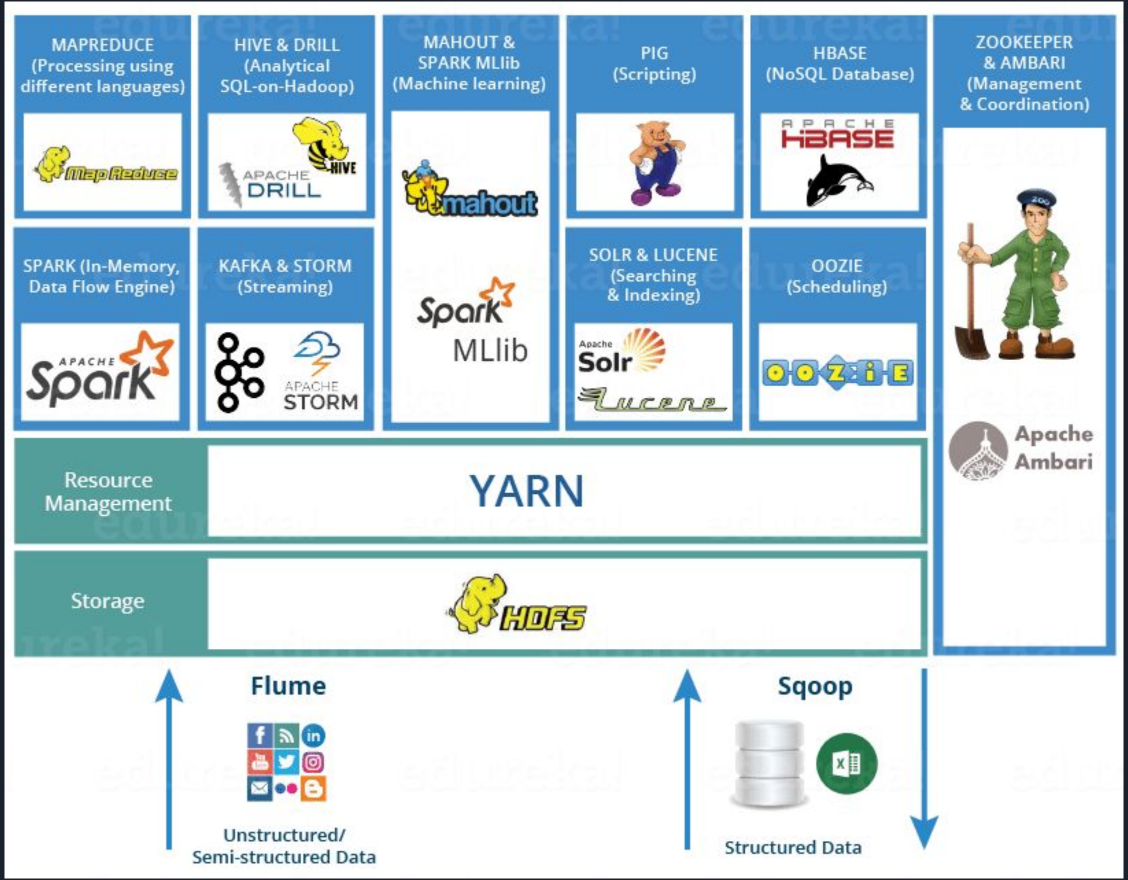
… I just have to be sure my solution fits into this tiny box

```
class MRWordFreqCount(MRJob):

    def mapper(self, _, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def combiner(self, word, counts):
        yield word, sum(counts)

    def reducer(self, word, counts):
        yield word, sum(counts)
```

Hadoop v1 Has Limitations

- “Main drawback of Hadoop 1.x is that MapReduce Component in it’s Architecture. That means it supports only MapReduce-based Batch/Data Processing Applications.”
  - https://www.journaldev.com/8806/differences-between-hadoop1-and-hadoop2
- It is only suitable for Batch Processing of Huge amount of Data, which is already in Hadoop System
- Assigns resources to Map/Reduce jobs, it cannot re-use them even though some slots are idle
- Example: 10 Map and 10 Reduce Jobs are running with 10 + 10 Slots to perform a computation
  - All Map Jobs are doing their tasks but all Reduce jobs are idle
  - We cannot use these Idle jobs for other purpose.
- It is not suitable for Real-time Data Processing
- It is not suitable for Data Streaming

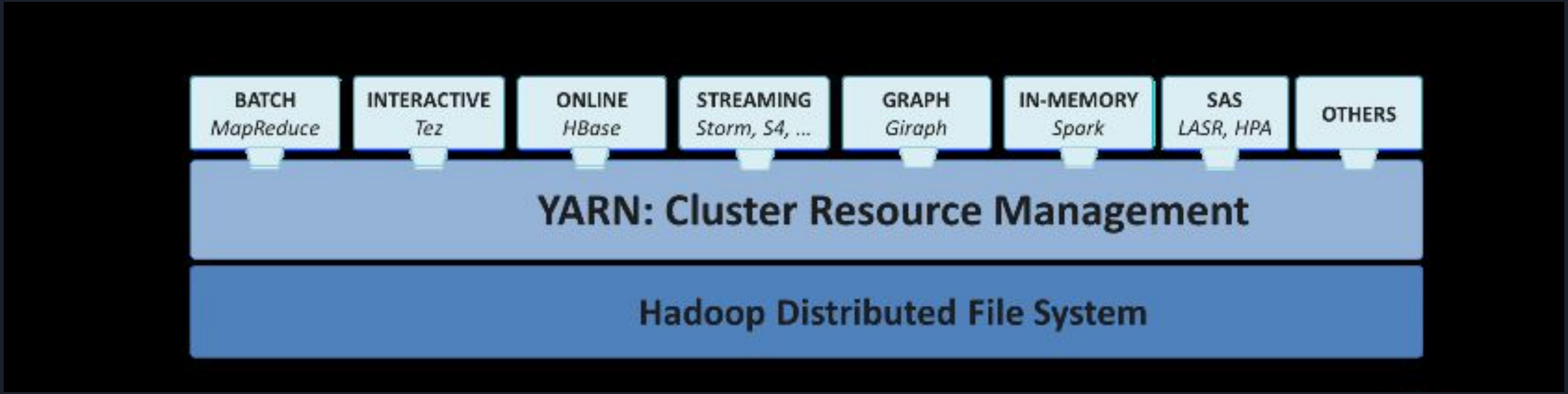
Hadoop v2


MAPREDUCE (Processing using different languages)
MapReduce
HIVE & DRILL (Analytical SQL-on-Hadoop)
HIVE
APACHE DRILL
MAHOUT & SPARK MLlib (Machine learning)
mahout
Spark MLlib
PIG (Scripting)
HBASE (NoSQL Database)
APACHE HBASE
ZOOKEEPER & AMBARI (Management & Coordination)
Apache Ambari
SPARK (In-Memory, Data Flow Engine)
APACHE Spark
KAFKA & STORM (Streaming)
APACHE STORM
SOLR & LUCENE (Searching & Indexing)
Apache Solr
Lucene
OOZIE (Scheduling)
OOZIE
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

Yet Another Resource Negotiator (YARN)

- “Yarn allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS (Hadoop Distributed File System). Apart from resource management, Yarn also does job Scheduling.”
  - https://data-flair.training/blogs/hadoop-yarn-tutorial/
- By decoupling MapReduce component and assigning responsibilities through YARN, Hadoop v2 supports multiple namespace, Multi-tenancy, Higher Availability and Higher Scalability
- Manages the global assignments of resources (CPU and memory) among all the applications
  - Scheduler
  - Application manager

Interface Through YARN



Python

Python is Faster, Java is Stronger

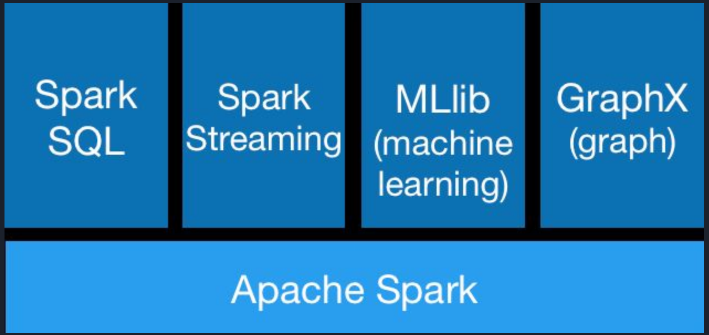
- Hadoop is built on Java (ew), so why not use it to code (cause we have lives)
- All jokes aside, Java is a powerful language that provides portability and universality of platforms through JVM
- **BUT**, we are not building long-running, cross-platform services/apps
- Objective is to take as much (un)structured data and do rapid prototyping/analysis
- Python is great because of the libraries and APIs available to us
    - No need to compile, packages can be installed easily with **pip/conda/easy_install** ...

Where/How to use Python in Hadoop

- No wrong answer here
  - Can write MR jobs in python (mrjob)
  - Can interface with HDFS for download/upload (snakebite)
  - Can do streaming analysis/search (pyspark)
- The main reason we want to go with python with hadoop is because it allows us to extend beyond the framework easily
- We will be focusing on using Spark + HDFS + Python

About Spark

- “Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Scala, Java, and Python that make parallel jobs easy to write, and an optimized engine that supports general computation graphs. It also supports a rich set of higher-level tools including Shark (Hive on Spark), MLlib for machine learning, GraphX for graph processing, and Spark Streaming.”
    - https://spark.apache.org/



Spark SQL

Spark Streaming

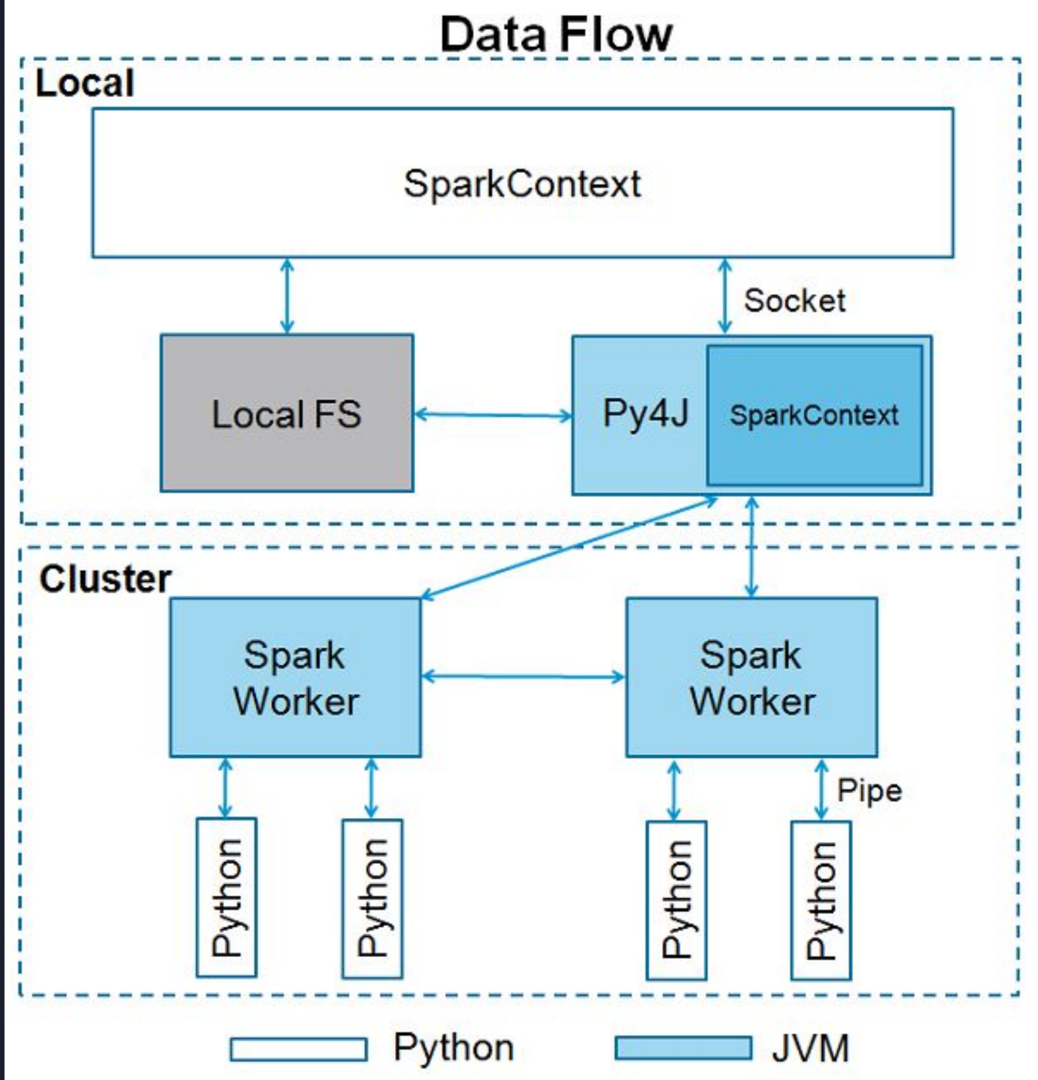
MLlib (machine learning)

GraphX (graph)

Apache Spark

Python + Spark = PySpark

- Programming in Spark (and PySpark) is in the form of chaining transformations and actions on RDDs
- RDDs are “Resilient Distributed Datasets”
  - Collection of elements partitioned across the nodes of the cluster that can be operated on in parallel
- RDDs are kept in memory for the most part
  - Allowing it to be reused efficiently across parallel operations
- PySpark offers PySpark Shell which links the Python API to the spark core and initializes the Spark context
  - Majority of data scientists and analytics experts today use Python because of its rich library set
- When we run any PySpark application, a driver program starts, which has the main function and your SparkContext gets initiated here.


Data Flow
Local
SparkContext
Socket
Local FS
Py4J
SparkContext
Cluster
Spark Worker
Spark Worker
Pipe
Python
Python
Python
Python
Python
JVM

DEMO

Conclusions

- Hadoop framework allows to leverage a proven, production-ready platform to store, extract, and analyze disparate data sources
- It is a great framework to help us get into data analytics and/or machine learning
- Not a perfect solution, but it works well enough to get us started
- Makes distributed (parallel) processing a much easier task
- Python is flexible, easy to use language to write MR jobs, streaming analysis, search algo, etc.
- May not be as powerful as Java, but is far less complex
- Python + Spark is a good way to leverage Hadoop
- **Don’t forget:** there is more than one solution out there (Kubernetes, MPI, …)

Thank you!