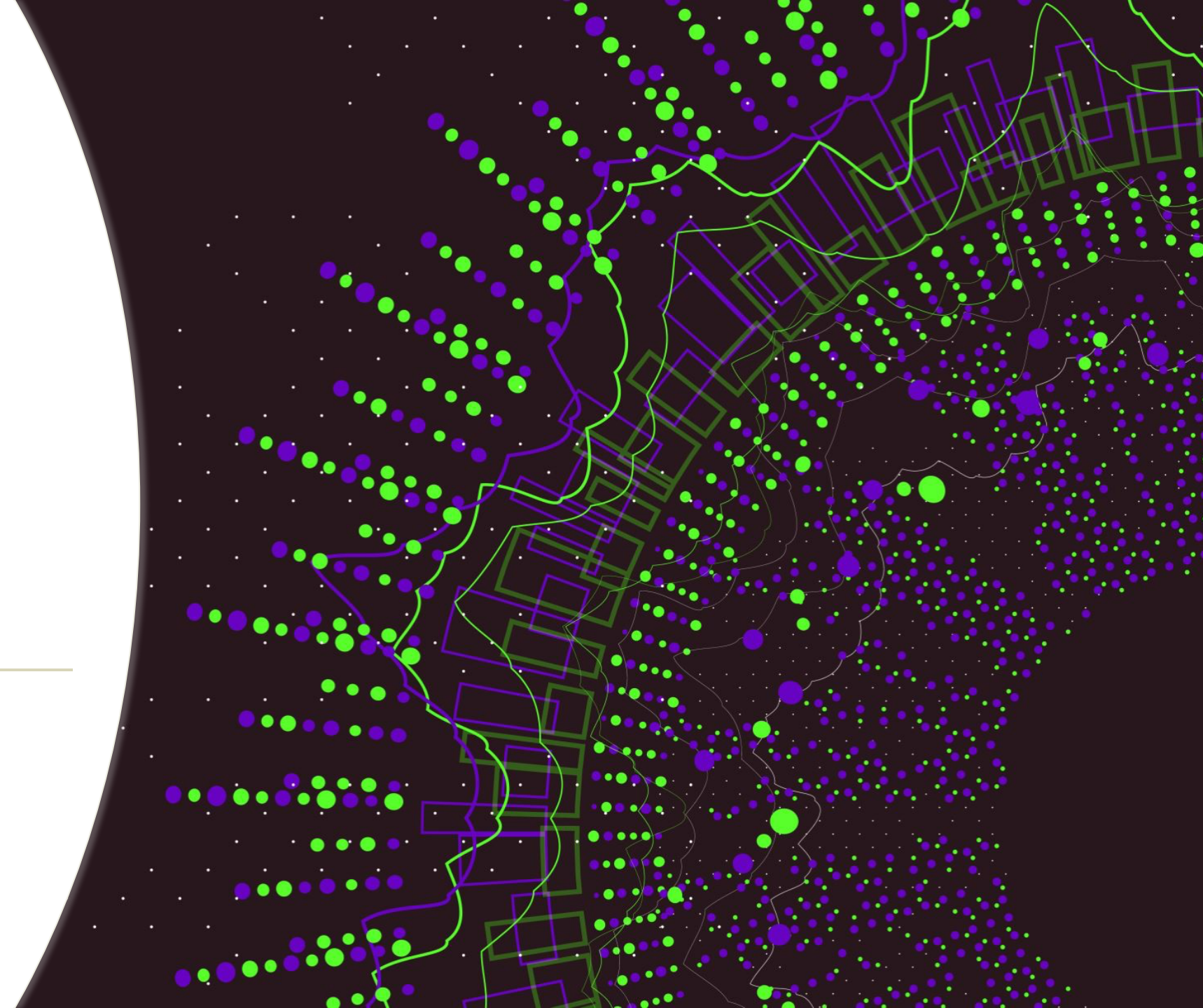
Mentor Session Cloud Computing

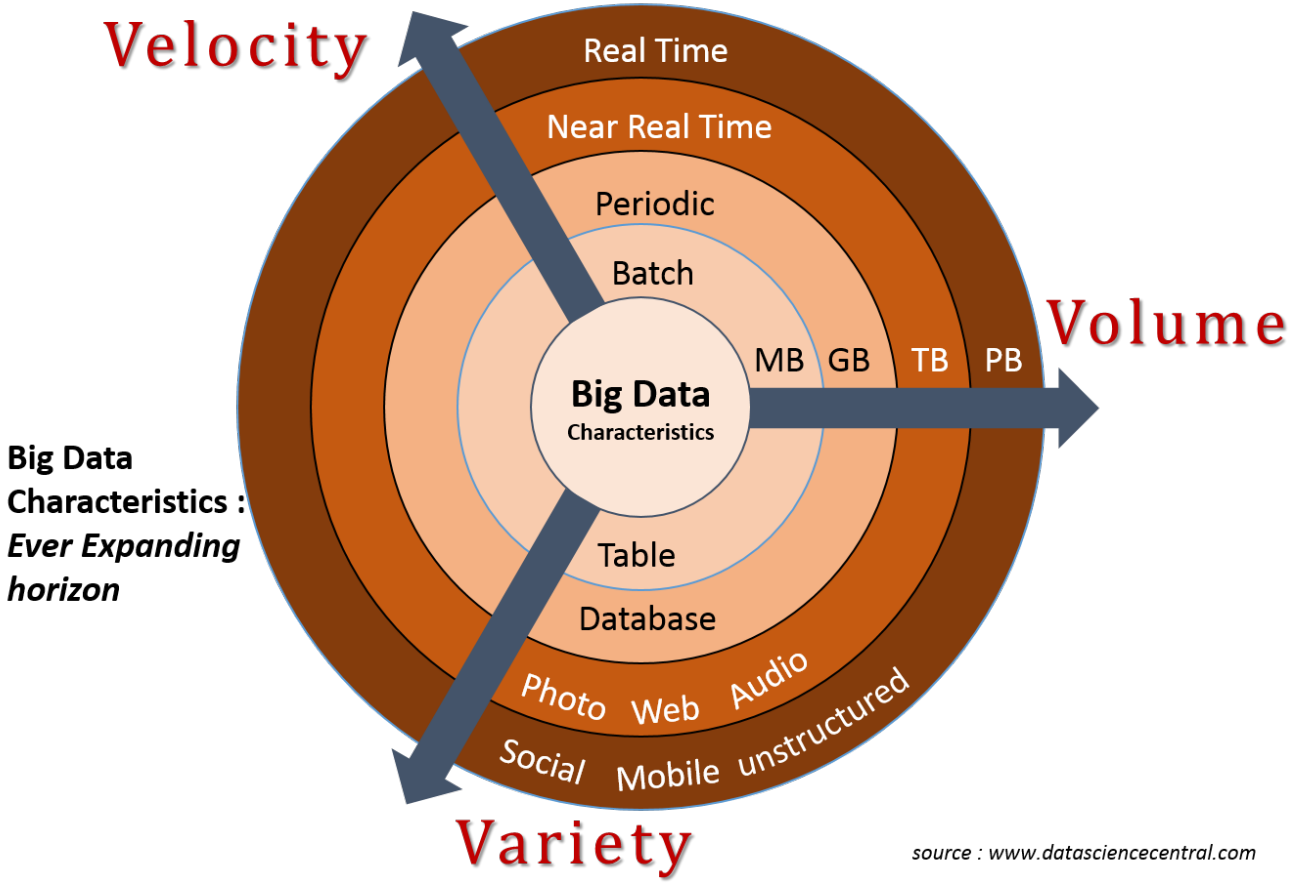
Machine Learning & Data Science II

Yingjie(Chelsea) Wang



What is “Big Data”?

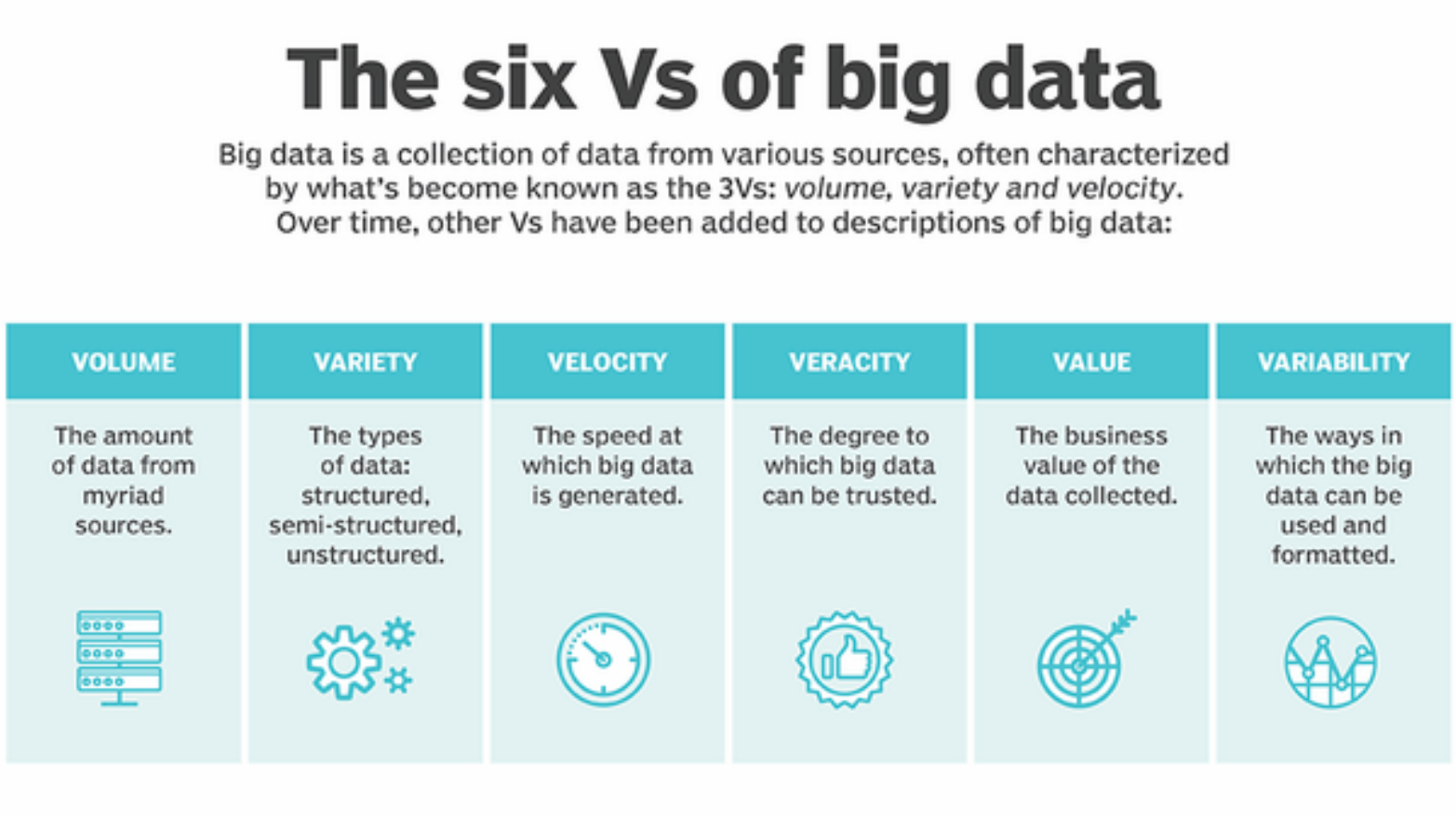
- O’Reilly:
  - “Big data is when the size of the data itself becomes part of the problem”
- EMC/IDC:
  - “Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from **very large volumes** of a **wide variety** of data, by enabling **high-velocity** capture, discovery, and/or analysis.”
- IBM: (The famous 3-V’s definition)
  - Volume (Gigabytes -> Exabytes)
  - Velocity (Batch -> Streaming Data)
  - Variety (Structured, Semi-structured, & Unstructured)



Big Data Characteristics : ***Ever Expanding horizon***

source : www.datasciencecentral.com

WHAT IS MORE


The six Vs of big data
Big data is a collection of data from various sources, often characterized by what's become known as the 3Vs: volume, variety and velocity. Over time, other Vs have been added to descriptions of big data:
VOLUME
The amount of data from myriad sources.
VARIETY
The types of data: structured, semi-structured, unstructured.
VELOCITY
The speed at which big data is generated.
VERACITY
The degree to which big data can be trusted.
VALUE
The business value of the data collected.
VARIABILITY
The ways in which the big data can be used and formatted.

Approaches to Managing Big Data

- Scaling “Up”: Improve the Components of One System
  - OS: multiple threads / VMs
  - CPU: increase clock speed, bus speed, cache size
  - RAM: increase capacity
  - Disk: Increase capacity, decrease seek, add platters
- Scaling “Out”: From Component Speedup to Distributed Computing
  - A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.
  - This massively parallel architecture can be treated as a single computer
    - Can exploit computational parallelism (near linear speedup)
    - Can have a vastly larger effective address space

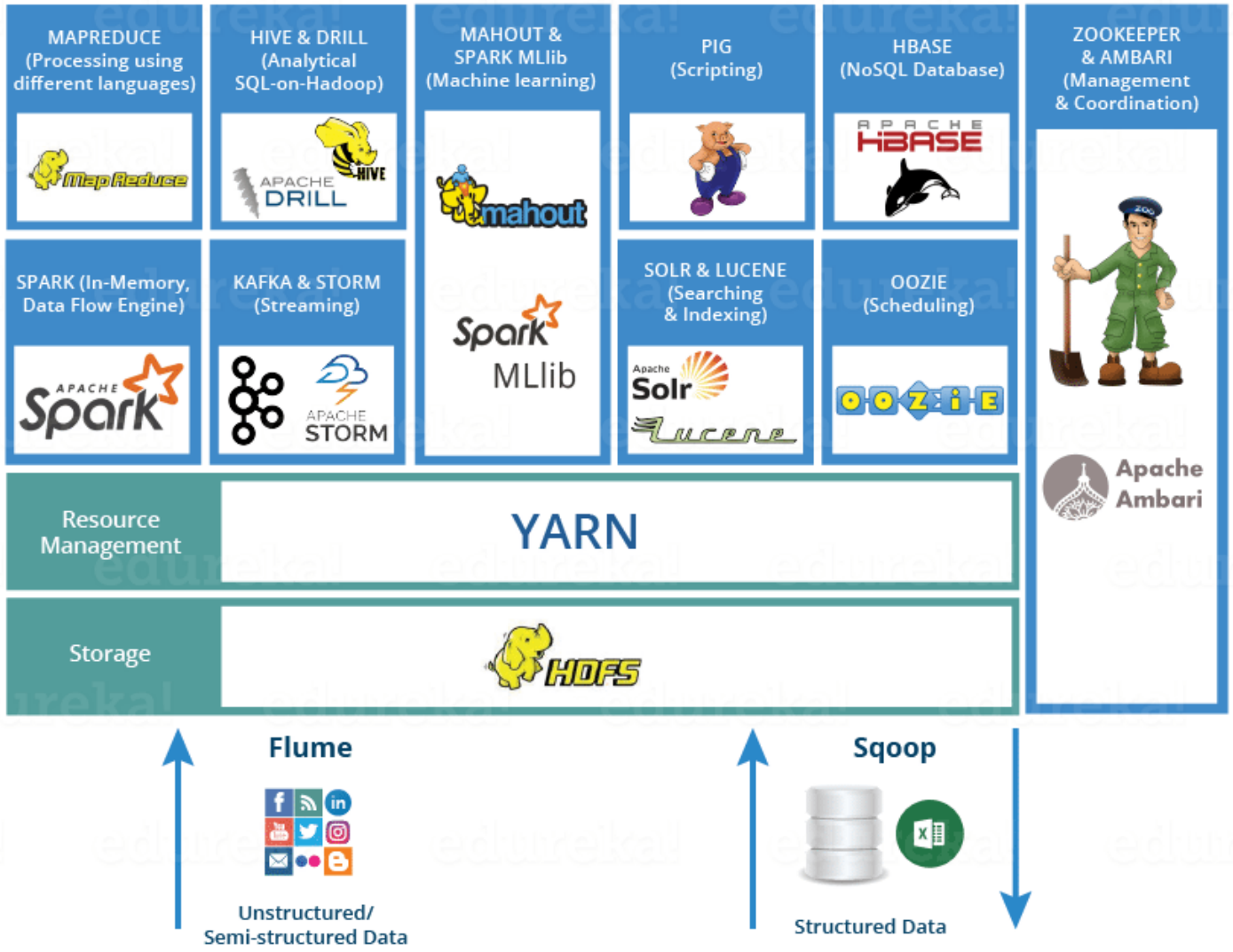
Apache Hadoop

- Open-source framework for developing & running parallel applications on hardware clusters
  - The Apache Hadoop software library is a framework that allows for the **distributed processing** of large data sets across clusters of computers using simple programming models.
  - It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.
  - Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Apache Hadoop

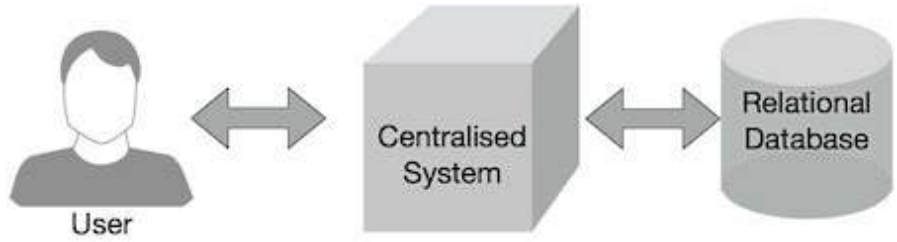
Key components:

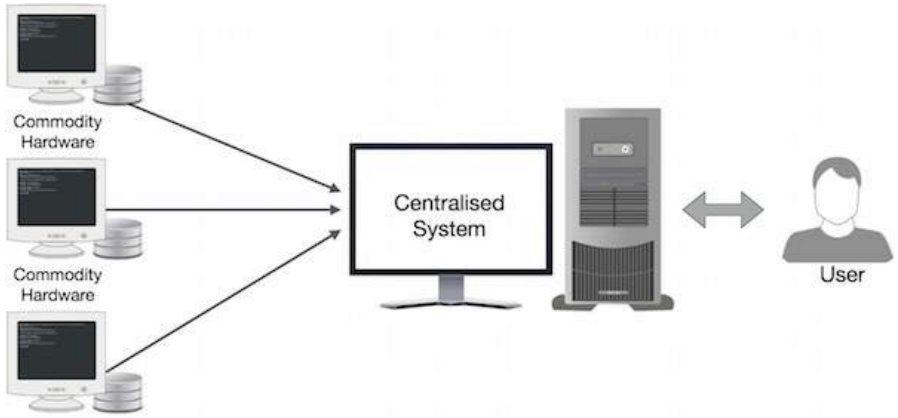
- HBase (key-value store)
- Hive, Pig (higher langs., compile to Map-Reduce)
- Map-Reduce (parallel programming pattern)
- HDFS (Hadoop Distributed File System)



MapReduce

- Traditional Enterprise Systems normally have a centralized server to store and process data.
  - Designed to handle only structured data that has well-designed rows and columns.
  - Relations Databases are vertically scalable which means you need to add more processing, memory, storage to the same system. This can turn out to be very expensive
  - It is not suitable to process huge volumes of scalable data. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.
- Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.





Map & Reduce Functions: Origins in Functional Programming

- The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
  - Map - Returns a list constructed by applying a function (the first argument) to all items in a list passed as the second argument
  - E.g.:

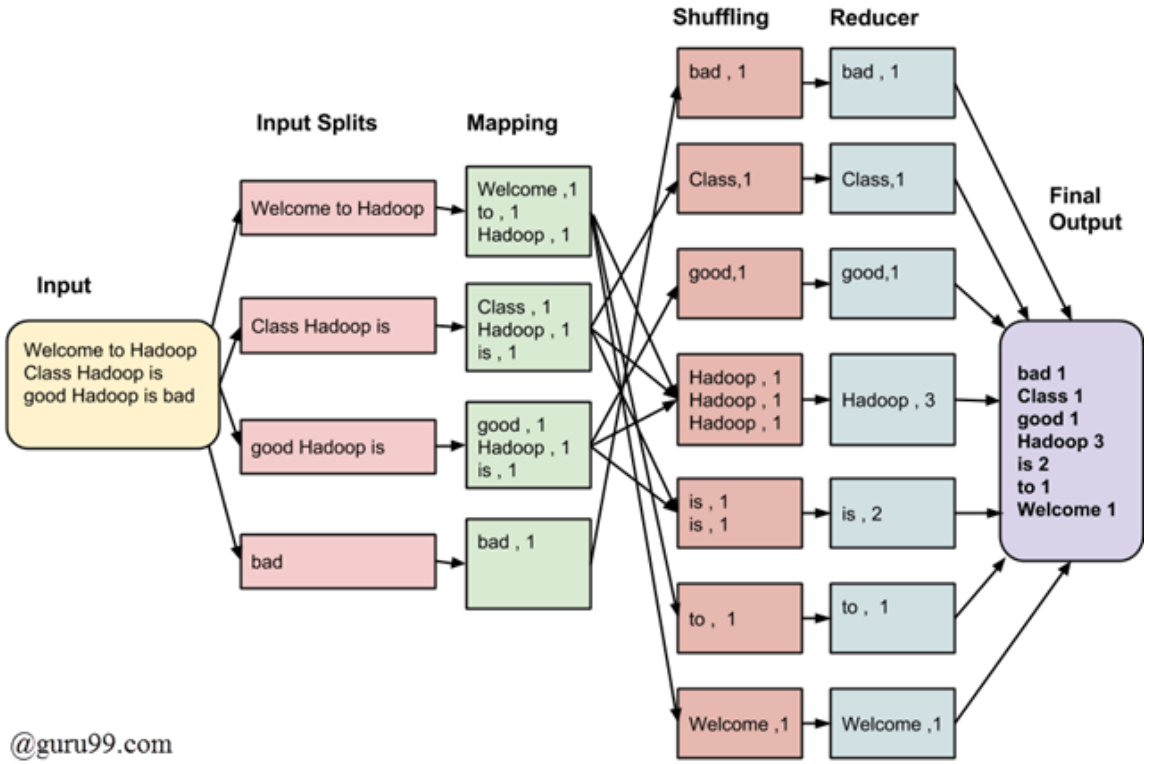
    map f [a, b, c] = [f(a), f(b), f(c)]
    map sq [1, 2, 3] = [sq(1), sq(2), sq(3)] = [1, 4, 9]
- The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.
  - Reduce - Returns a list constructed by applying a function (the first argument) on the list passed as the second argument. Can be identity (do nothing).
  - E.g.:
    reduce f [a, b, c] = f(a, b, c)
    reduce sum [1, 4, 9] = sum(1, sum(4,sum(9,sum(NULL)))) = 14

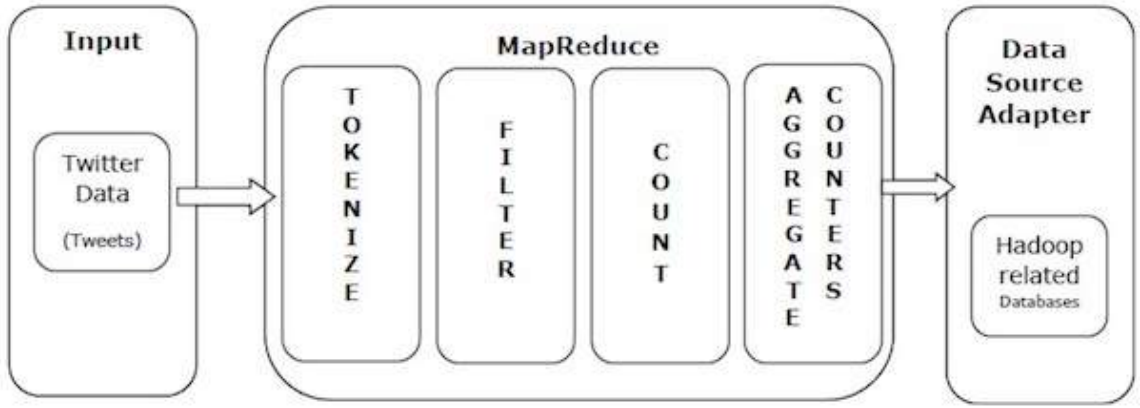
MapReduce-Example

Let us take a real-world example to comprehend the power of MapReduce. Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.

MapReduce algorithm performs the following actions:

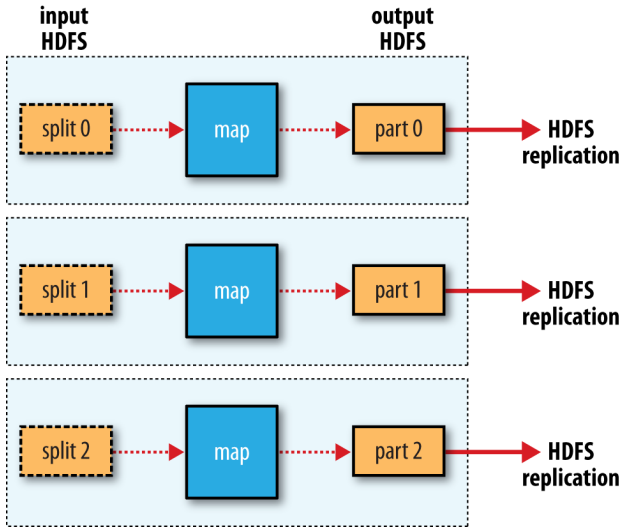
- **Tokenize** − Tokenizes the tweets into maps of tokens and writes them as key-value pairs.
- **Filter** − Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.
- **Count** − Generates a token counter per word.
- **Aggregate Counters** − Prepares an aggregate of similar counter values into small manageable units.

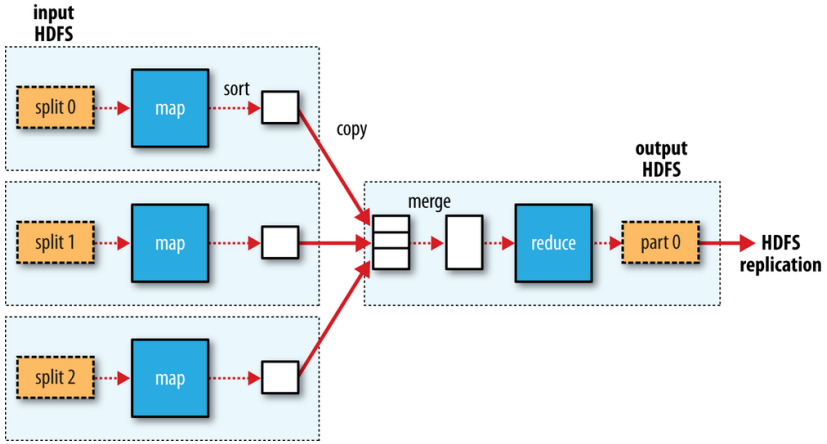


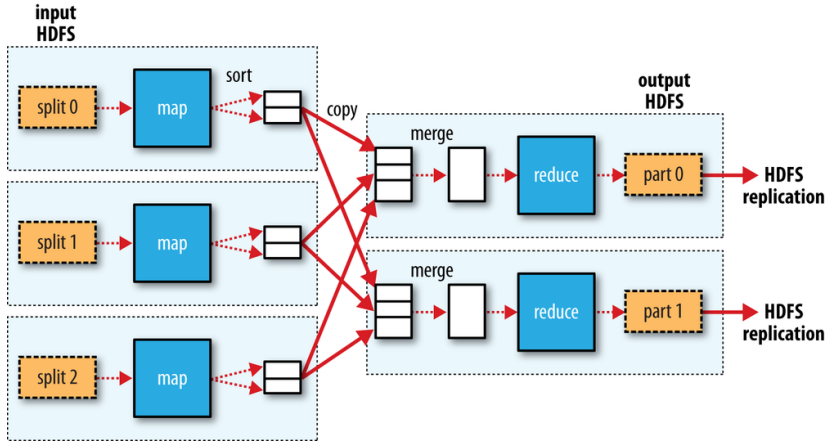


MapReduce Data Flow

- MapReduce Data flow with No Reduce Tasks
- MapReduce Data Flow with a Single Reduce Task
- MapReduce Data Flow with a Multiple Reduce Tasks

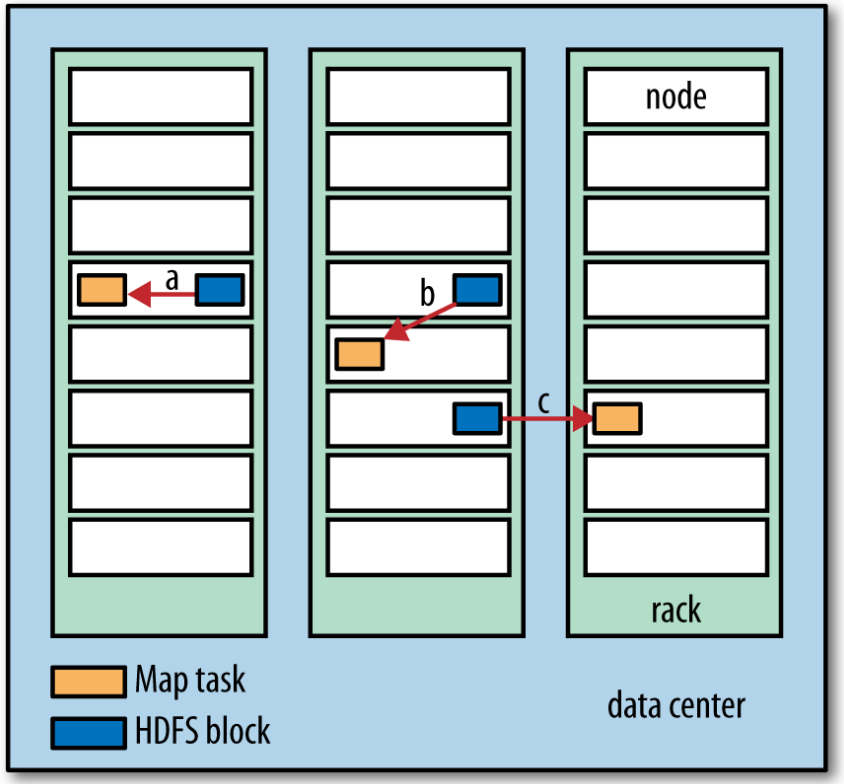






MapReduce Hadoop Implementation: Data Storage

- To support distributed processing, we need a distributed data storage. Hadoop Distributed File System is the core component, the backbone of Hadoop Ecosystem.
- HDFS is a distributed filesystem
  - Large files are split into (HDFS block size) chunks
- Each chunk can be stored on a separate node
- Provides data locality
- Can be linearly expanded to increase storage space
- Stores data on commodity (inexpensive) machines
- Provides redundant storage of extremely large datasets and replicated across the cluster (default of 3x)
  - At 3X, a 1TB file needs 3TB of storage



Hadoop 1.0 had two main systems: MapReduce and HDFS

MapReduce – Performs computation

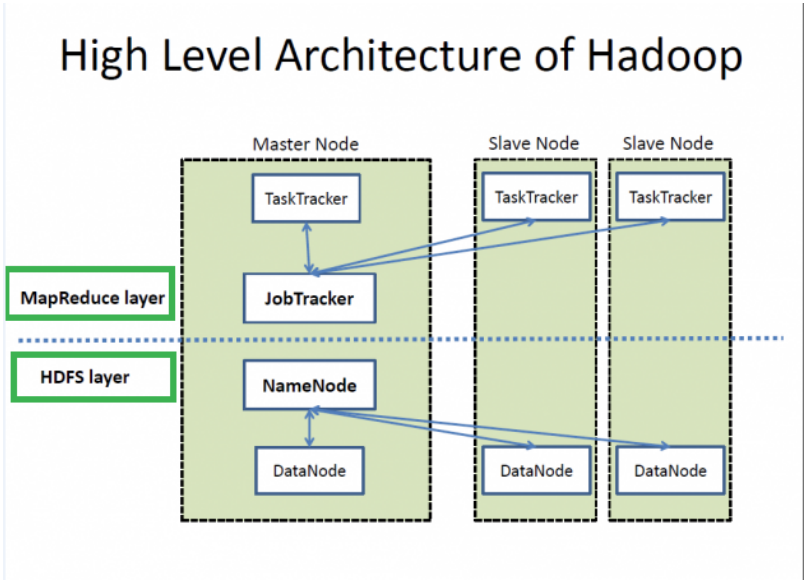
- Job Tracker – Master planner
- Task Tracker – Runs each task

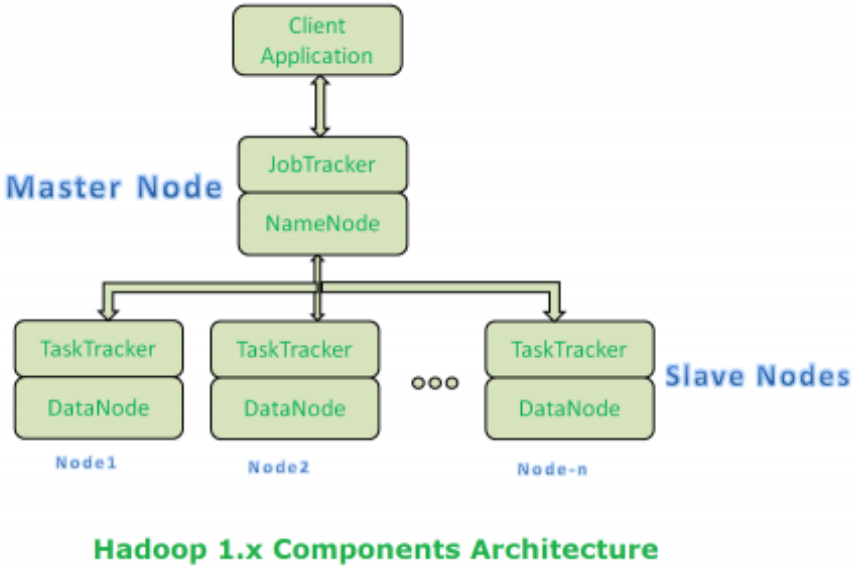
HDFS – Hadoop Distributed File System (Stores the data)

- Data Node – Stores the blocks for each file.
- Name Node – Keeps track of every file and where it is stored.
  - *Controls the Data Nodes.*

Data must be stored where MapReduce can reach it

- Cluster HDFS
- Amazon S3
- Microsoft Azure Blob





Hadoop 1.x Components Architecture

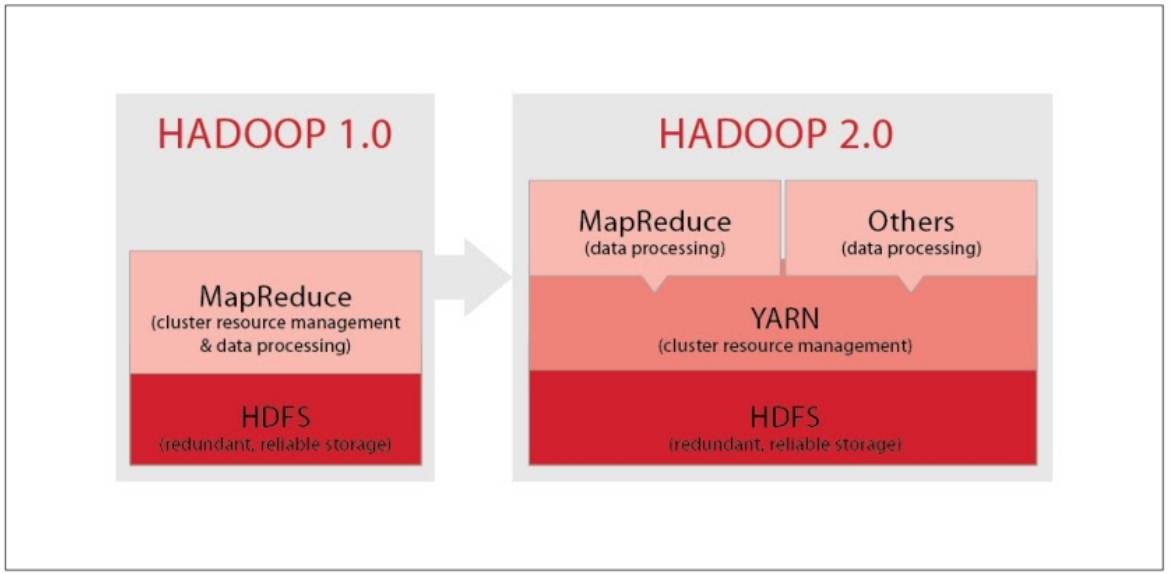
Hadoop 1.0 and 2.0

Hadoop 1.0 had some issues

- Only supported MapReduce
- Worked on a concept of "slots" - slots could either run a map task or a reduce task
- Had a single name node to manage the entire cluster (single point of failure)
- Limited to 4,000 nodes per cluster

What's new in Hadoop 2.0

- YARN is the resource management and computation framework that is new as of Hadoop 2, which was released late in 2013. YARN supports multiple processing models in addition to MapReduce.
- YARN brings in the concept of a central resource management. This allows multiple applications to run on Hadoop, sharing a common resource management.

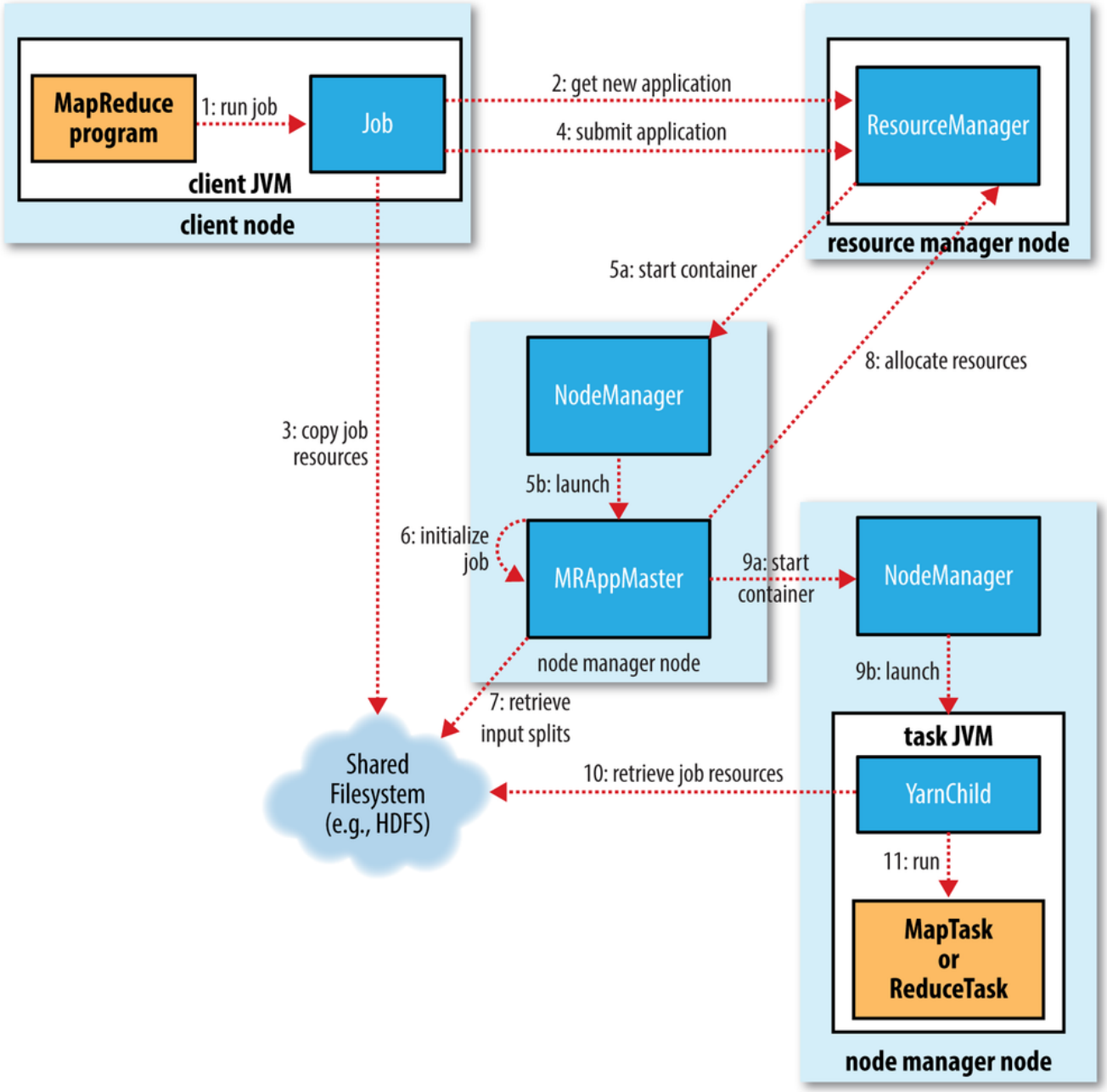


MapReduce 1	YARN
Jobtracker	Resource manager, application master, timeline server
Tasktracker	Node manager
Slot	Container

Hadoop 1.0 and 2.0

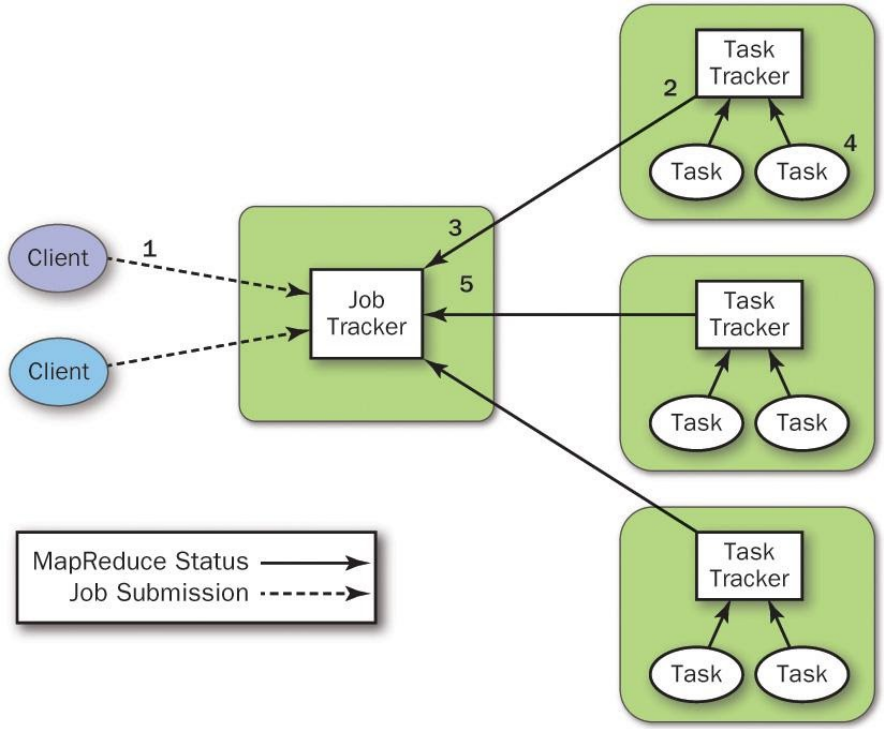
Hadoop 2 on YARN

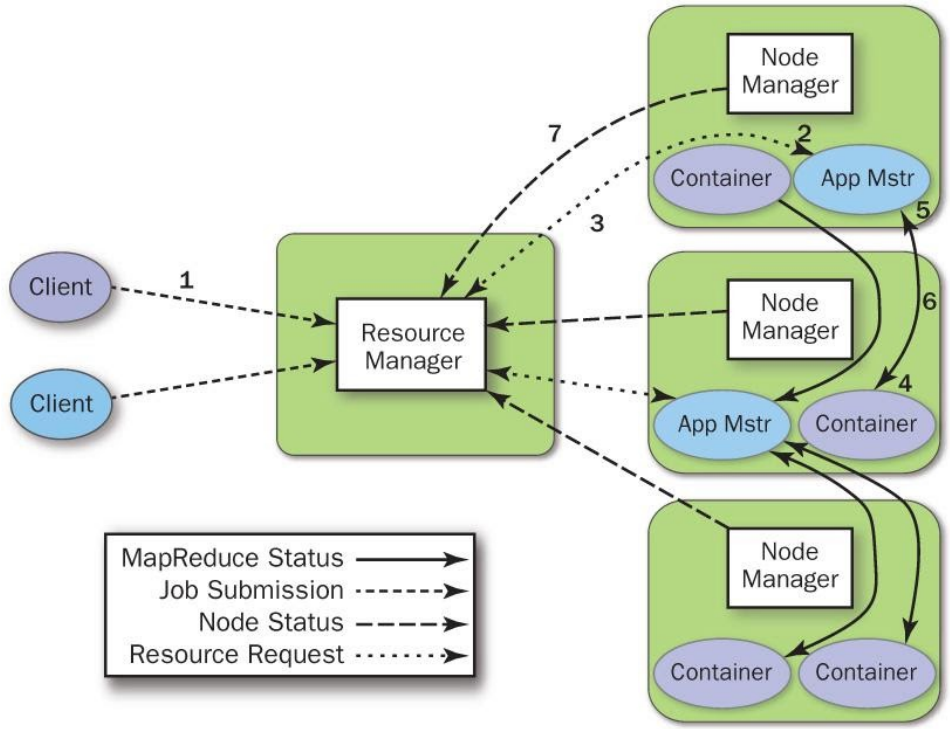
- Resource Manager (RM) – serves as the central agent for managing and allocating cluster resources
- Node Manager (NM) - per node agent that manages and enforces node resources
- Application Master (AM) – per application manager that manages lifecycle and task scheduling



Hadoop 1.0 and 2.0

- YARN framework
  - Better utilization of resources
  - Running non-MapReduce applications
  - Backward compatibility
  - JobTracker no longer exists





Hadoop 2.0 & Hadoop 3.0

Hadoop Version	2.0	3.0
Minimum Java Version	Java 7	Java 8
Fault Tolerance	Replication	Erasure Coding
Data Balancing	HDFS Balancer	Intra-data Node Balancer
Storage Scheme & Overhead	3x, 200% overhead	Erasure Coding, 50% overhead
Compatible File System	HDFS, S3, Azure Blob Storage	Same + Azure Datalake FS
Max Size	10,000	+10,000

Hadoop/MapReduce Benefits

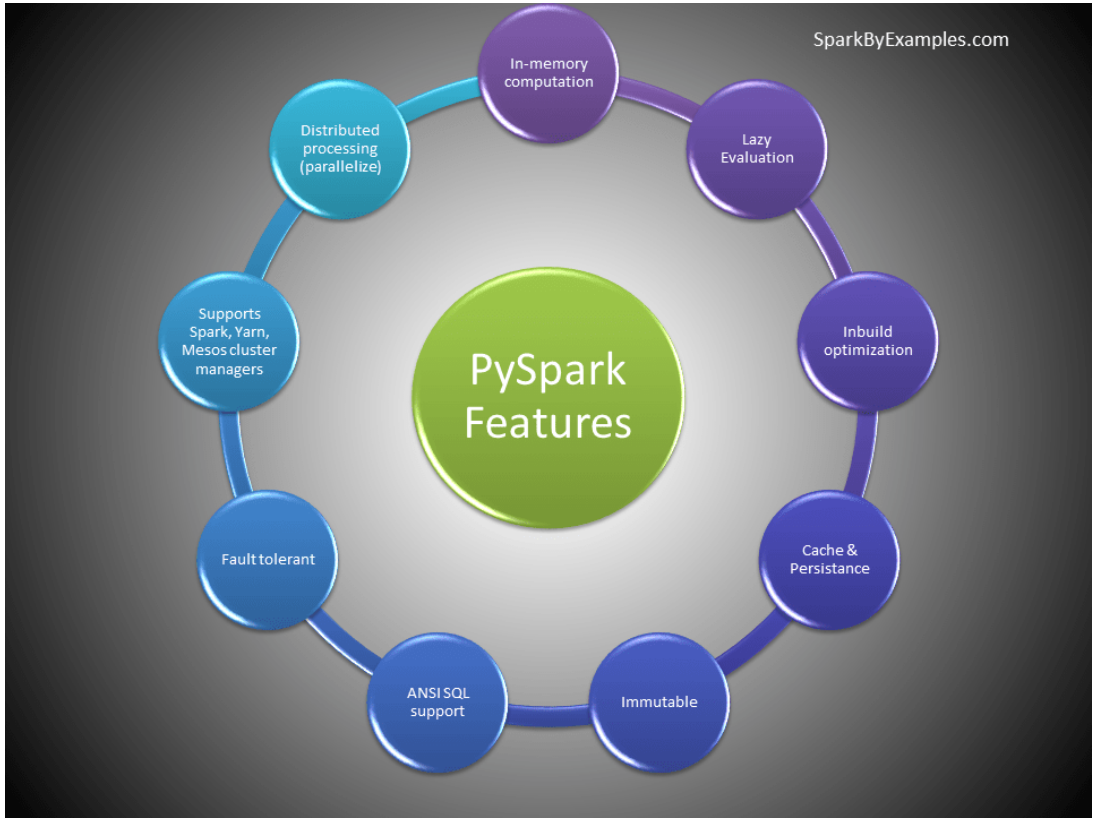
- **Cost**: Hadoop is open-source and uses cost-effective commodity hardware which provides a cost-efficient model
- **Scalability**: Businesses can process petabytes of data stored in the Hadoop Distributed File System (HDFS).
- **Flexibility**: Hadoop enables easier access to multiple sources of data and multiple types of data.
- **Speed**: With parallel processing and minimal data movement, Hadoop offers fast processing of massive amounts of data.
- **Fault Tolerance**
  - If a worker fails, that job can be run on another machine.
  - The master writes periodic checkpoints. If it dies, it is restarted.
- **Minimizes Network Bandwidth**
  - Attempts schedule workers on the same network node as the data resides.
  - Failing that, it tries to schedule the worker on the same network switch
- **Simple**: Developers can write code in a choice of languages, including Java, C++ and Python.

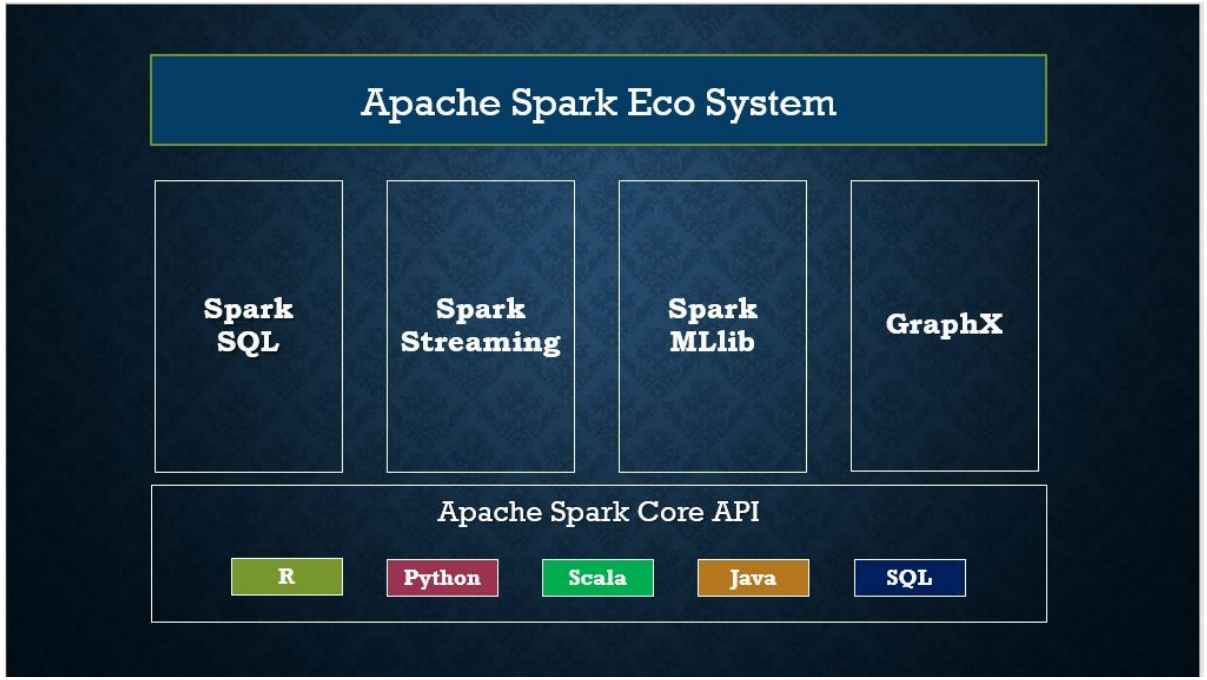
Hadoop/MapReduce Limitations

- Programmer:
  - Everything is a Map or Reduce, but we don't think of problems that way
  - No control over the order in which map() or reduce() runs.
- Performance
  - Problem with Small files: Hadoop fails when it needs to access the small size file in a large amount. Too many small files surcharge the name node and make it difficult to work.
  - Lack of Security: Hadoop uses Kerberos for security feature which is not easy to manage. Storage and network encryption are missing in Kerberos which makes us more concerned about it.
- Operational:
  - High overhead
  - Batch processing
  - Does not produce immediate answers

PySpark

- PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine learning applications.





Spark MLlib

MLlib is a package included in Spark that provides interfaces for:

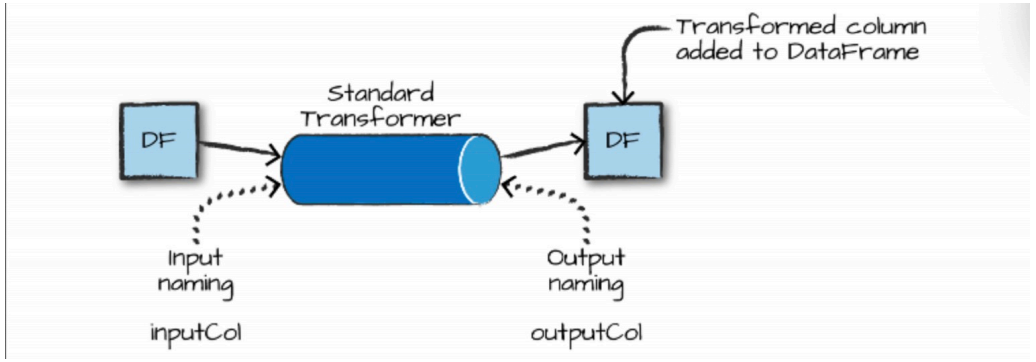
- Gathering & cleaning data
- Feature engineering
- Feature selection
- Training & tuning large-scale supervised and unsupervised models
- Using models in production

MLlib API is divided into two packages

- org.apache.spark.mllib (original API - predates DataFrames)
  - It contains the original APIs built on top of RDDs
- org.apache.spark.ml (newer API)
  - It provides higher-level API built on top of DataFrames for constructing ML pipelines
  - It is recommended because with DataFrames the API is more versatile and flexible
  - It provides the pipeline concept

High level MLlib Concepts

- DataFrame
  - Spark ML uses DataFrames from Spark SQL as ML datasets, which can hold a variety of data types
  - DataFrame could have different columns storing text, feature vectors, (true) labels, and predictions
- Transformer
  - A Transformer is an algorithm which can transform one DataFrame into another DataFrame
  - A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended.
  - A classification model is a Transformer which can be applied on a DataFrame with features and transforms it into a DataFrame with also predictions.



High level MLlib Concepts

- Estimator
  - An Estimator is an algorithm which can be applied on a DataFrame to produce a Transformer (a model)
  - An Estimator implements a method fit(), which accepts a DataFrame and produces a Model of type Transformer
  - An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on an input dataset and returns a model
  - A classification algorithm such as Logistic Regression is an Estimator, and calling fit() on it a Logistic Regression Model is built, which is a Model and hence a Transformer
- Pipeline
  - A Pipeline chains multiple Transformers and Estimators together to specify a Machine learning/Data Mining workflow
  - The output of a transformer/estimator is the input of the next one in the pipeline
  - A simple text document processing workflow aiming at building a classification model includes several steps
    - Split each document into a set of words
    - Convert each set of words into a numerical feature vector
    - Learn a prediction model using the feature vectors and the associated class labels

Classification in MLLib

- The following classification models are available in MLLib:
  - Logistic regression, Decision trees, Random forests, Gradient-boosted trees
- No multi-labeled predictions supported directly
  - Need to train one model per label and combine manually
- Model scalability

Model	Features count	Training examples	Output classes
Logistic regression	1 to 10 million	No limit	Features x Classes < 10 million
Decision trees	1,000s	No limit	Features x Classes < 10,000s
Random forest	10,000s	No limit	Features x Classes < 100,000s
Gradient-boosted trees	1,000s	No limit	Features x Classes < 10,000s

Regression in MLLib

- The following regression models are available in MLLib:
  - Linear regression
  - Generalized linear regression
  - Isotonic regression
  - Decision trees
  - Random forest
  - Gradient-boosted trees
  - Survival regression
- Model scalability

Model	Number features	Training examples
Linear regression	1 to 10 million	No limit
Generalized linear regression	4,096	No limit
Isotonic regression	N/A	Millions
Decision trees	1,000s	No limit
Random forest	10,000s	No limit
Gradient-boosted trees	1,000s	No limit
Survival regression	1 to 10 million	No limit

Unsupervised Learning in MLLib

- The following unsupervised models are available in MLLib:
  - k-means clustering, Bisecting k-means, Gaussian mixture models, Latent Dirichlet Allocation
- All the clustering algorithms available in Spark work only with numerical data
  - Categorical values must be mapped to integer values (i.e, numerical values)
- Model scalability

Model	Statistical recommendation	Computation limits	Training examples
k-means	50 to 100 maximum	Features x clusters < 10 million	No limit
Bisecting *k*-means	50 to 100 maximum	Features x clusters < 10 million	No limit
GMM	50 to 100 maximum	Features x clusters < 10 million	No limit
LDA	An interpretable number	1,000s of topics	No limit