CM2606 Data Engineering

Big Data Processing 02

Week 08 | Piumi Nanayakkara

Learning Outcomes

- Covers LO3 and LO4 for Module

- On completion of this lecture, students are expected to be able to:
  - Apply knowledge on big data processing frameworks to design and implement solid data pipelines.
  - Understand and describe Spark execution model
  - Investigate and evaluate Spark execution plan

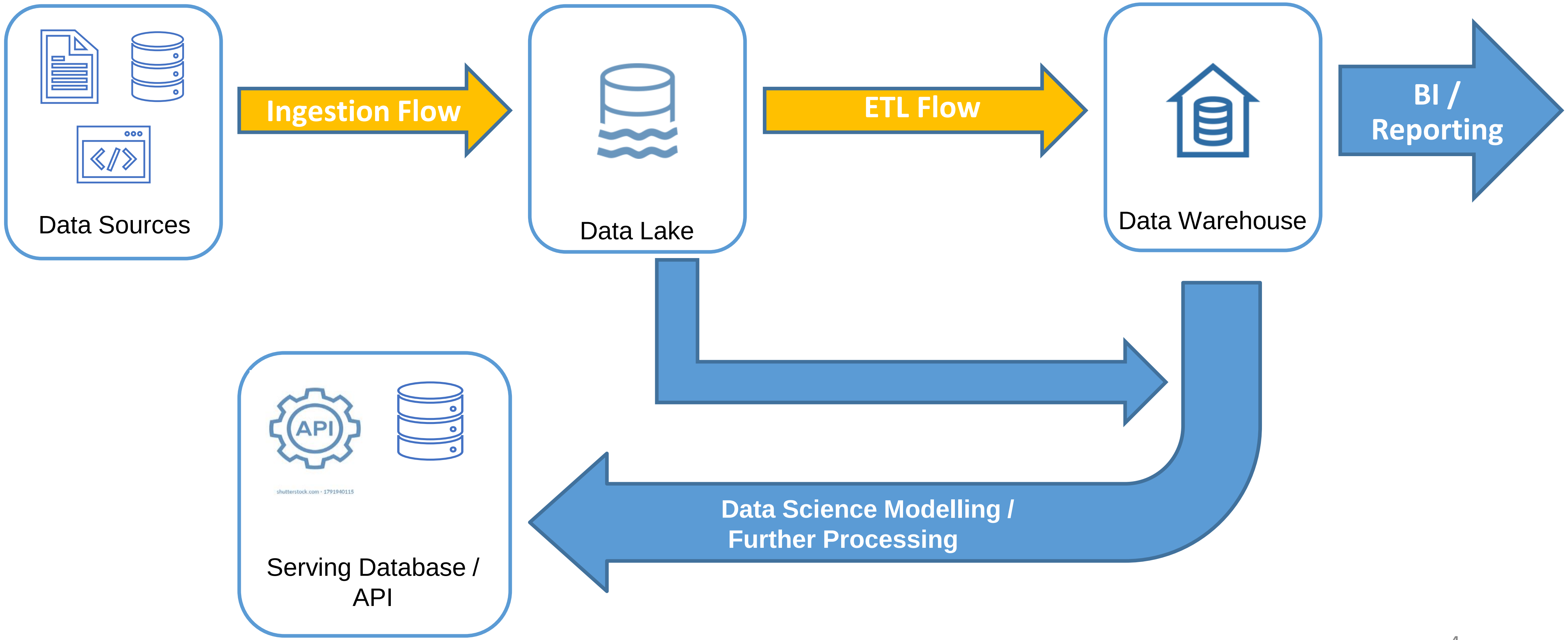
Content

- Coding ETL Jobs

- Big Data Processing Frameworks
  - Map Reduce
  - Spark

- Spark
  - Components
  - Execution Model
  - RDD Operations
  - Optimizations

Data Pipeline: Common Usage



No Code ETL

- Tools/Platforms provide GUI controls where one can drag and drop few icons and create a complete data pipeline
  - E.g., Talend, AWS Glue, Azure Data Factory

- Once the pipeline is created it could be triggered manually or scheduled to be run at a specific time/frequency.

- Even the non-technical employees can be trained to setup pipelines

- Challenges of this approach are:
  - Issues arise with high volumes of data leading to performance issues
  - Writing high complex transformations
  - Could be costly compared to manual ETL

SQL based Data Processing

- Could be implemented as
  - SQL Scripts
  - PL/SQL packages or views
  - ETL tools with the capability to execute SQL commands
    - E.g., SSIS, Oracle Data Integrator

SQL based ETL Jobs: Tuning

- Coding
  - Select set Operations over row operations
    - “Insert into XX As Select from YY” over inserting each row in a loop
  - Avoid Functions in WHERE Conditions
    - Confuses the optimizer to estimate the cost
    - Custom functions are black boxes
  - Reduce data early as possible

SQL based ETL Jobs: Tuning

- Run Statements in Parallel
  - Warehouse tool kits support MPP

- Indexes might help in querying (selecting small % of rows from a large table), but they will increase the load times

- Use direct path inserts
  - Might impact foreign key constraints
  - avoids the effort for reusing of free space in existing table blocks

- Gather statistics after each Load
  - Missing or wrong statistics can cause poor estimations for the next ETL steps

Python based Data Processing

- Python offers strong support to work with indexed data structures and dictionaries, which are important in ETL operations.
- Offers support for handling semi structured and unstructured data
- Simple versatile enough that users can code almost any ETL process with native data structures.

Big Data Processing Frameworks

- Amount of data is simply too big to handle on a single machine.

  - *Assume: a Read operation of 256MB block take 2 seconds*

  - *Single Node with 500GB Disk*
    - *500GB / 256 MB = 1954 Blocks*
    - *2 seconds * 1954 =* ***1 hour (approx.)***

  - *10 Node cluster with 5 Disks on each node*
    - *1954 Blocks / (5 * 10) Disks = 40 Blocks per disk*
    - *2 seconds * 40 =* ***1.3 minutes***

Big Data Processing Frameworks

From Scripting to a Framework

- Features a programming interface and APIs
- Every processing job is divided into sub tasks where Framework decides
  - How to distribute tasks among operators
  - Where each task would run
- Concurrency and Fault Tolerance is handled by framework

Big Data Processing Frameworks: History

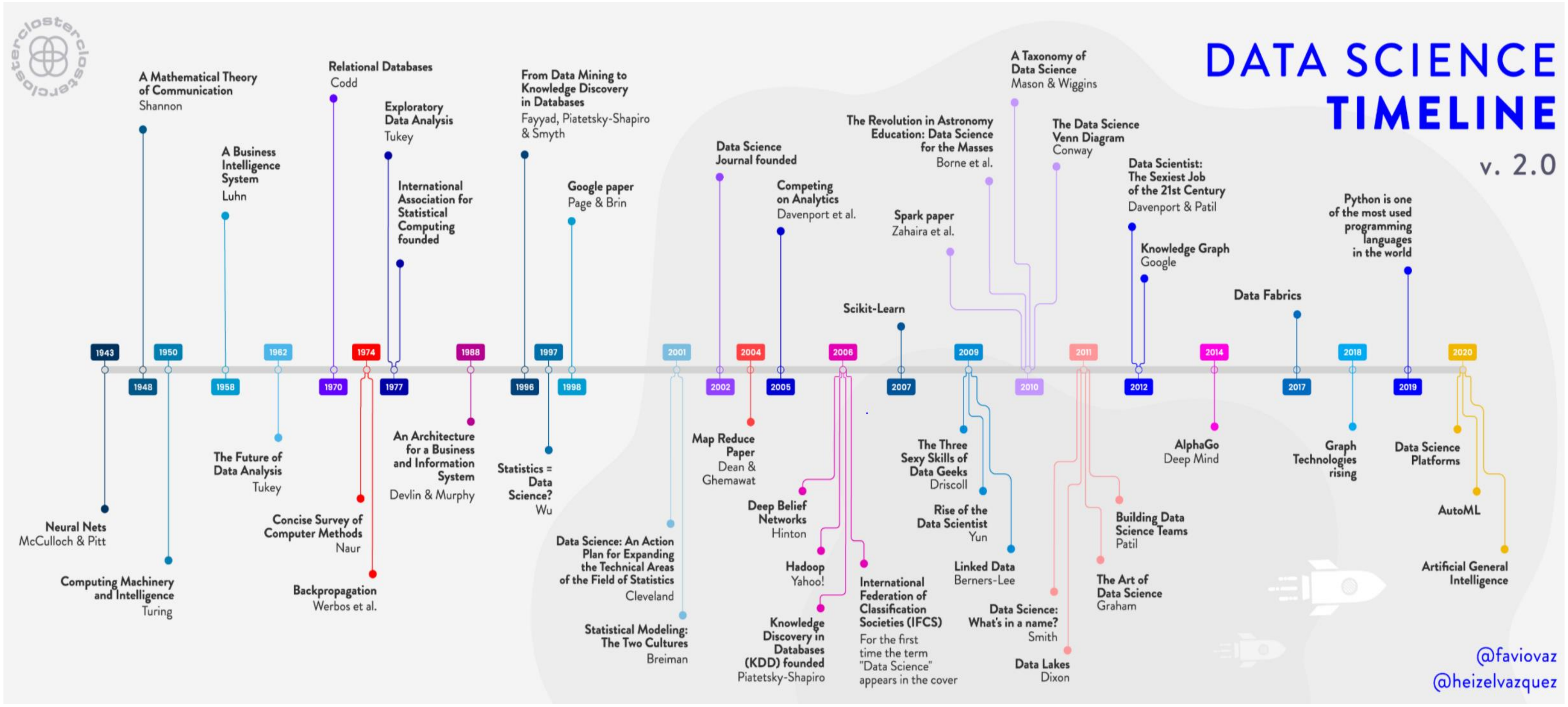
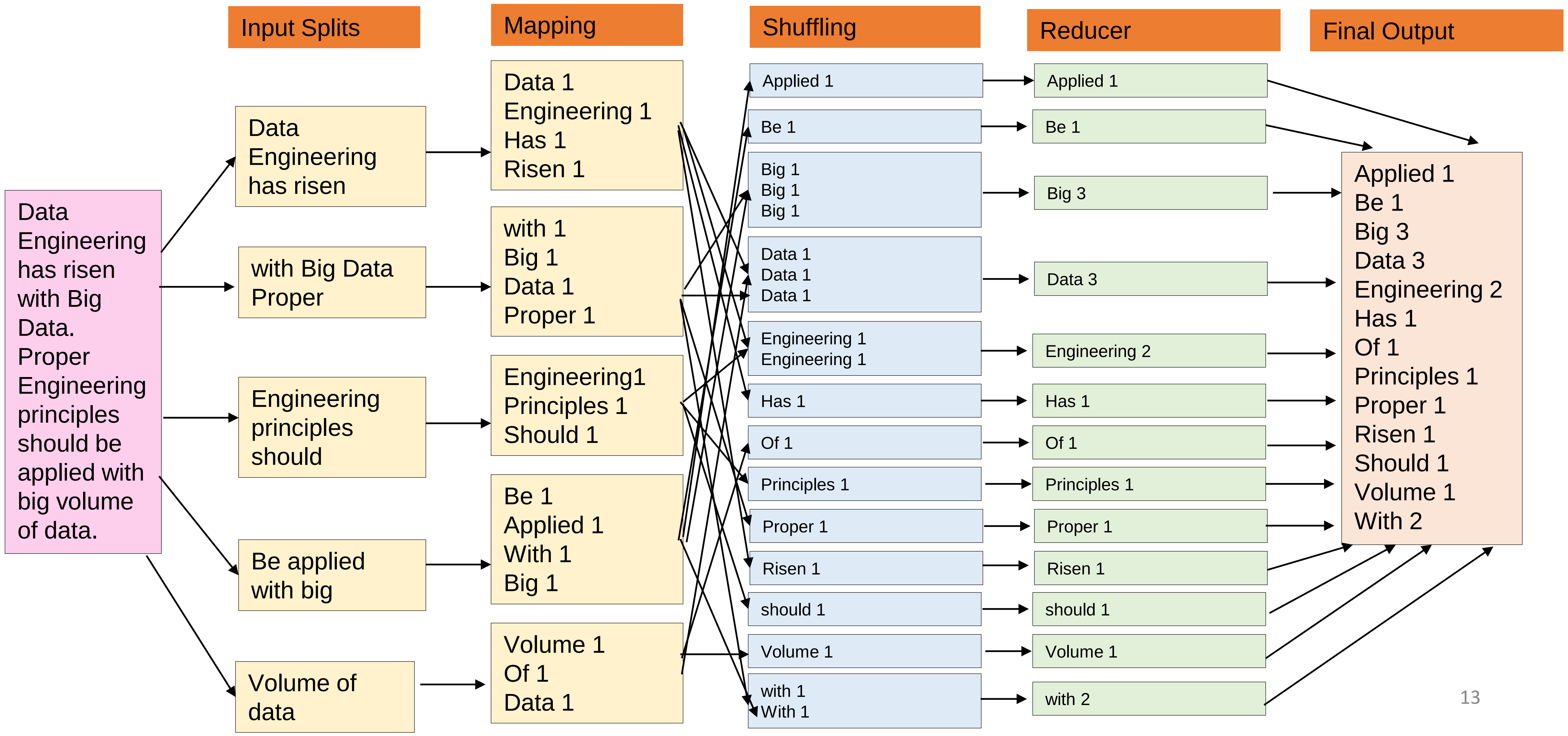

closterclosterclosterclost
DATA SCIENCE TIMELINE
v. 2.0
1943
Neural Nets
McCulloch & Pitt
1948
A Mathematical Theory of Communication
Shannon
1950
Computing Machinery and Intelligence
Turing
1958
A Business Intelligence System
Luhn
1962
The Future of Data Analysis
Tukey
1970
Relational Databases
Codd
1974
Concise Survey of Computer Methods
Naur
Backpropagation
Werbos et al.
1977
Exploratory Data Analysis
Tukey
International Association for Statistical Computing founded
1988
An Architecture for a Business and Information System
Devlin & Murphy
1996
From Data Mining to Knowledge Discovery in Databases
Fayyad, Piatetsky-Shapiro & Smyth
1997
Statistics = Data Science?
Wu
1998
Google paper
Page & Brin
2001
Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics
Cleveland
Statistical Modeling: The Two Cultures
Breiman
2002
Data Science Journal founded
2004
Map Reduce Paper
Dean & Ghemawat
2005
Competing on Analytics
Davenport et al.
2006
Deep Belief Networks
Hinton
Hadoop
Yahoo!
Knowledge Discovery in Databases (KDD) founded
Piatetsky-Shapiro
International Federation of Classification Societies (IFCS)
For the first time the term "Data Science" appears in the cover
2007
Scikit-Learn
2009
The Three Sexy Skills of Data Geeks
Driscoll
Rise of the Data Scientist
Yun
Linked Data
Berners-Lee
The Revolution in Astronomy Education: Data Science for the Masses
Borne et al.
Spark paper
Zahaira et al.
2010
A Taxonomy of Data Science
Mason & Wiggins
The Data Science Venn Diagram
Conway
2011
Data Science: What's in a name?
Smith
Data Lakes
Dixon
Building Data Science Teams
Patil
The Art of Data Science
Graham
2012
Data Scientist: The Sexiest Job of the 21st Century
Davenport & Patil
Knowledge Graph
Google
2014
AlphaGo
Deep Mind
2017
Data Fabrics
2018
Graph Technologies rising
2019
Python is one of the most used programming languages in the world
2020
Data Science Platforms
AutoML
Artificial General Intelligence
@faviovaz
@heizelvazquez

Image Source

Map Reduce: Divide and Conquer



Input Splits
Mapping
Shuffling
Reducer
Final Output
Data Engineering has risen with Big Data. Proper Engineering principles should be applied with big volume of data.
Data Engineering has risen
with Big Data Proper
Engineering principles should
Be applied with big
Volume of data
Data 1 Engineering 1 Has 1 Risen 1
with 1 Big 1 Data 1 Proper 1
Engineering1 Principles 1 Should 1
Be 1 Applied 1 With 1 Big 1
Volume 1 Of 1 Data 1
Applied 1
Be 1
Big 1 Big 1 Big 1
Data 1 Data 1 Data 1
Engineering 1 Engineering 1
Has 1
Of 1
Principles 1
Proper 1
Risen 1
should 1
Volume 1
with 1 With 1
Applied 1
Be 1
Big 3
Data 3
Engineering 2
Has 1
Of 1
Principles 1
Proper 1
Risen 1
should 1
Volume 1
with 2
Applied 1 Be 1 Big 3 Data 3 Engineering 2 Has 1 Of 1 Principles 1 Proper 1 Risen 1 Should 1 Volume 1 With 2

Apache Spark

- Open-source framework for implementing distributed processing of big data
- Spark is part of the Hadoop ecosystem related projects.
- Spark works in the in-memory computing paradigm
- Multiple Language Support
  - Scala, Python, R, Java

Spark Vs. Map Reduce

Feature	Hadoop / Map Reduce	Spark
Data Loads	Batch Processing only	Both Batch and Real time processing
Speed	Slower, Data is written to drive time to time	10X - 100X Faster with in-memory processing
optimizations	Dev responsible for optimizations	Spark auto Optimize
Language	Java	Scala - Functional
Coding	Need several classes for a small task	One or two lines of code
Processing	Need to write intermediate results if there are multiple map phases.	In memory processing
Iterative Algorithms	Each Mapper and Reducer killed after each iteration and data is written to Hard Disk.	Data is kept in memory and Executors are reused.

Spark Components

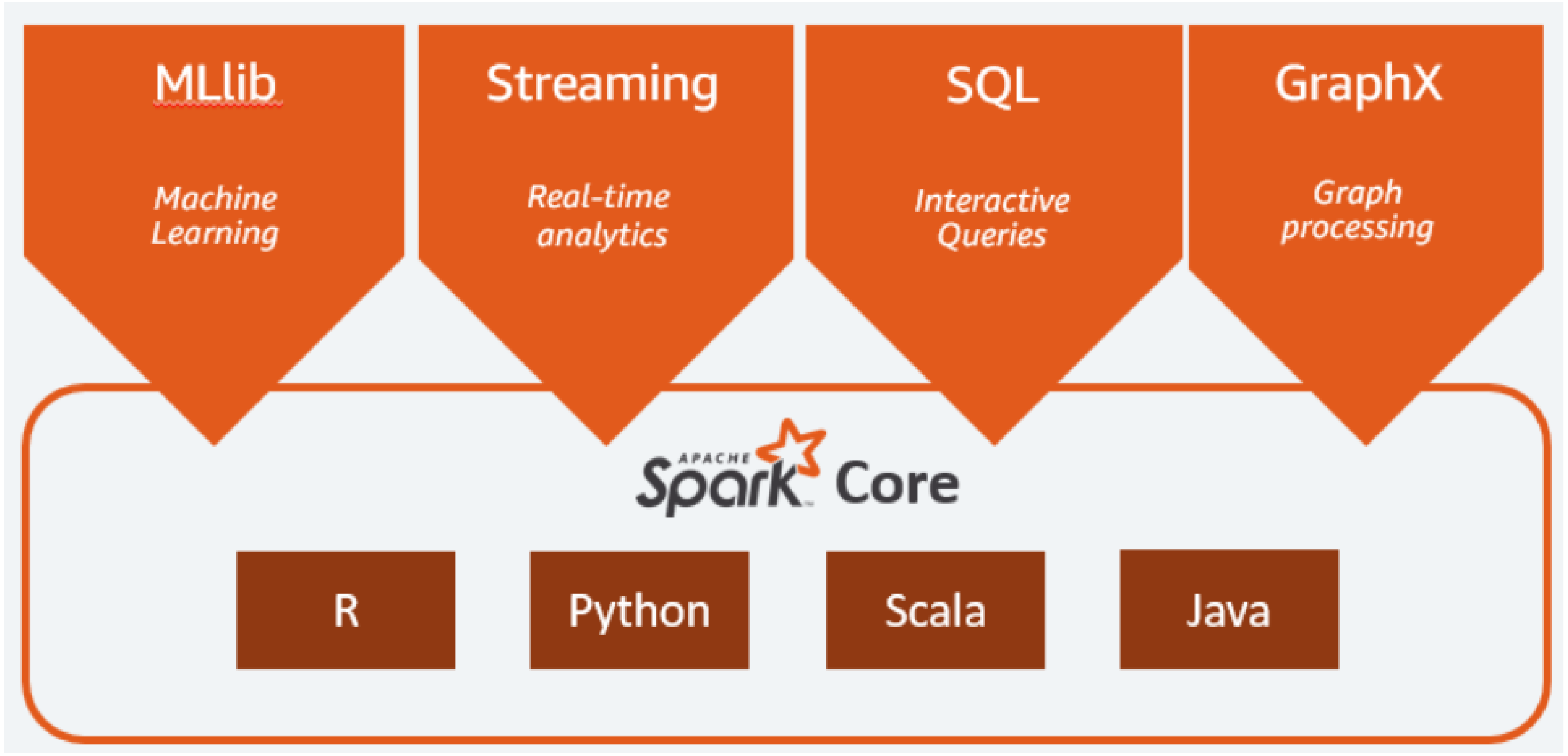


Image Source

Spark Components

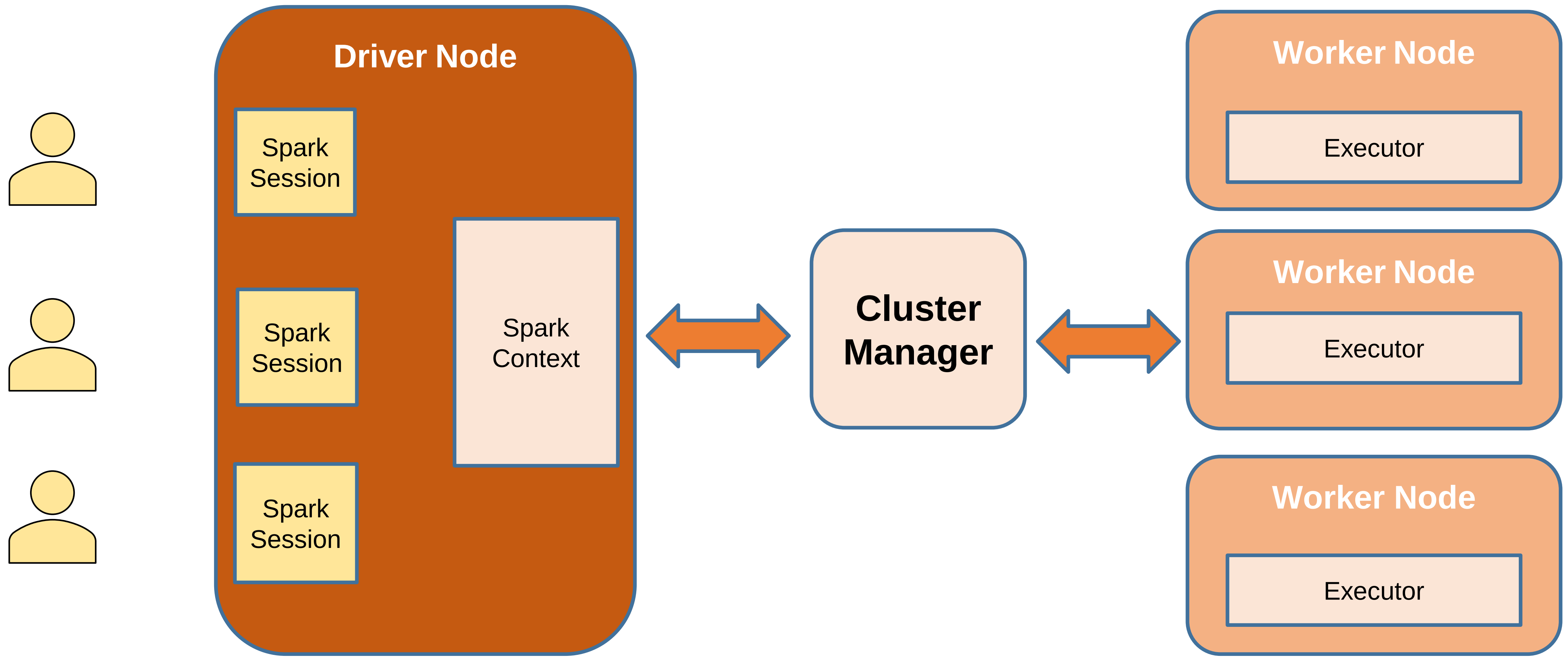
- Spark Core:
  - Foundation of the platform
  - Responsible for memory management, fault recovery, scheduling, distributing & monitoring jobs, and interacting with storage systems.
  - Exposed through language APIs hiding the complexity of distributed processing behind simple, high-level operators.

- Spark MlLib
  - Library to support machine learning on data at scale.

Spark Components

- Spark Streaming
  - Real-time solution which Ingests data in mini-batches (DStreams) and enables analytics on that data with the same application code written for batch analytics.
- Spark SQL
  - Distributed query engine that provides low-latency, interactive queries
- Spark GraphX
  - Distributed graph processing framework which provides ETL, exploratory analysis, and iterative graph computation

Spark Execution Model



Spark Execution Model

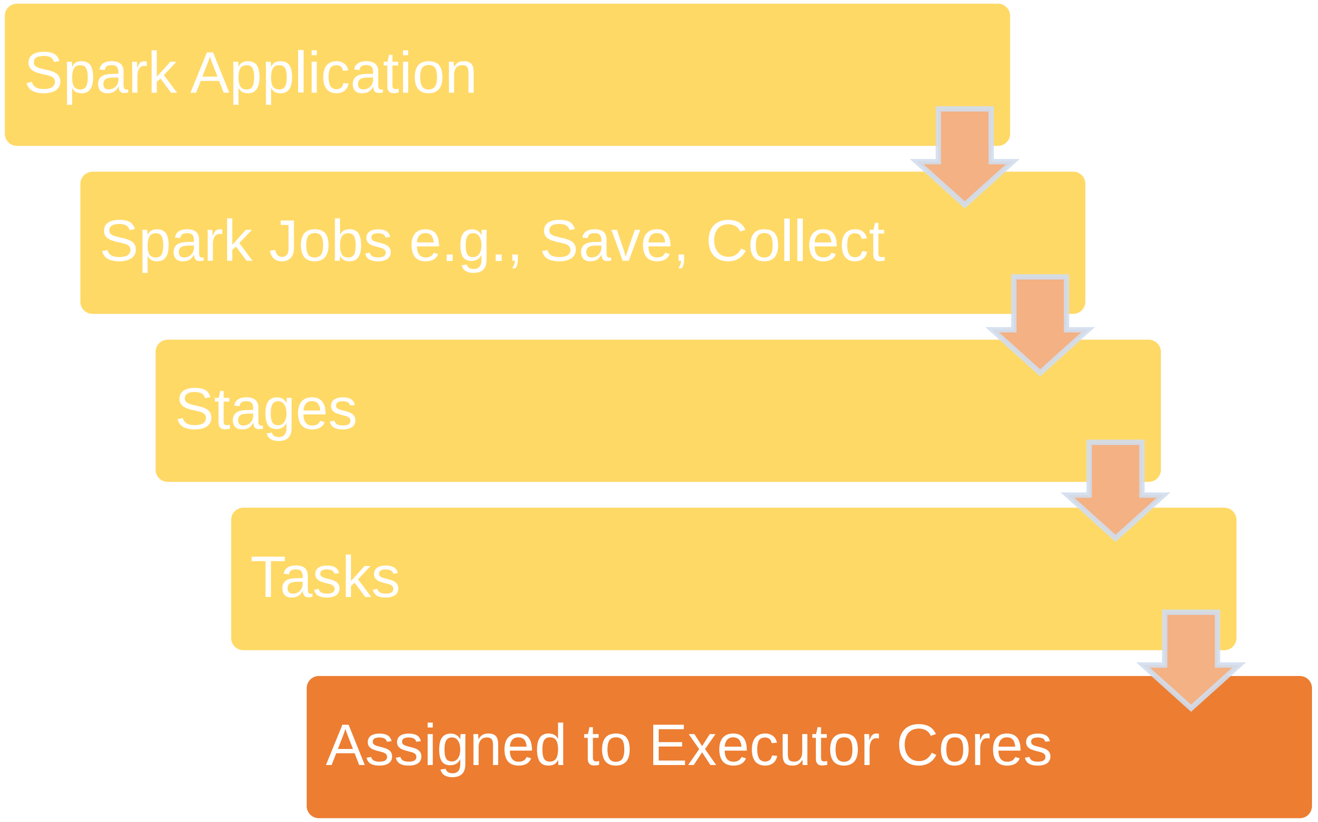
- All application code needs to be bundled into a .jar file, which will be copied to each executor in the cluster once the application is submitted to be run.

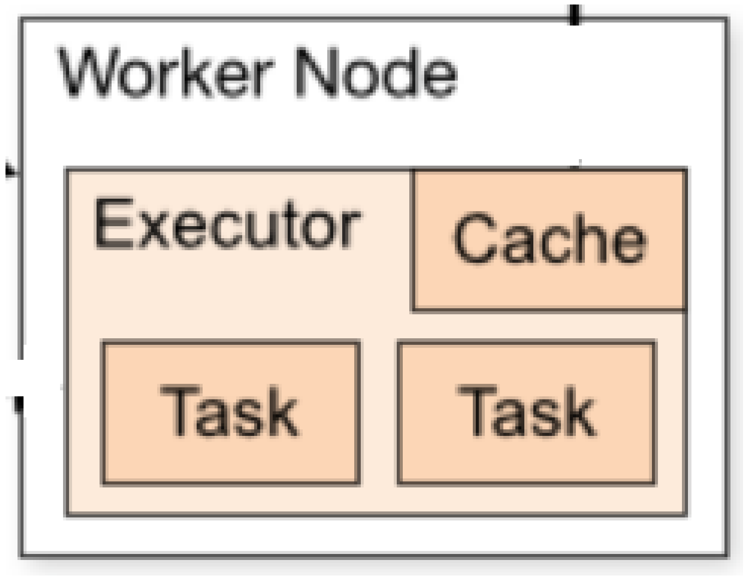
- An execution plan will be created for each application such that all processing needs to be performed are modelled as jobs that gets spawned in response to a Spark action (e.g. save)

- A job is then divided into stages that depend on each other where a stage can have multiple tasks and these tasks are executed by the cores. If an executor has 4 cores, it can run 4 tasks in parallel.

- An executor is a process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.

Spark Execution Model





Spark Execution Model

- Driver:
  - Runs the main() function of the application, takes the set of commands to be executed, creates the DAG/execution plan
  - Then the *executors* on worker nodes are acquired and application code (defined by JAR or Python files) is sent to all executors
  - Finally, tasks are sent to the executors to run
  - Based on where the driver process runs there are two modes in running a spark application
    - Cluster Mode: the framework launches the driver inside of the cluster.
    - Client Mode: the submitter launches the driver outside of the cluster, generally in the same node where user submit the command.

Spark Execution Model

- Spark Context:
  - Act as a bridge between Driver and Cluster Manager
  - Have all the configs defined such as executor memory, cores per executor
- Spark Session:
  - When there are multiple users from Spark 2, they can have their own spark sessions with own properties and set of tables
    - not visible to other users/sessions unless made global
  - But there will only be one spark context common to all sessions
- Cluster Manager:
  - Helps with Resource Allocation. There are 4 types:
    - Standalone, Mesos, Kubernetes and YARN - widely used

Spark Data Structures: RDD

- Resilient Distributed Dataset (RDD) is a fault-tolerant distributed collection of elements that can be operated on in parallel.

- There are two ways to create RDDs:
  - Parallelizing an existing collection in your driver program, or
  - Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, or any data source offering a Hadoop InputFormat.

- RDDs are immutable.
  - You can modify an RDD with a transformation, but the transformation returns you a new RDD whereas the original RDD remains the same.

- Partitions: Refers to the number of partitions to cut the dataset into. Spark will run one task for each partition of the cluster.

RDD Operations

- GroupBy vs ReduceBy

  - ReduceBy Do partial calculations
    - Combines all the records of the same key, before sending over the data for shuffling to the other executor.
    - Far more superior in terms of performance as it uses bandwidth efficiently.

  - No such combining is done in GroupBy operation

  - In GroupBy operations there's a chance that memory errors are being thrown.

RDD Operations

- Partitions in a data structure could be skewed where some partitions carry bulk of the data while some others carry very less amount of data.
  - Because of this some executors might finish early but wait in idle for other executors to finish.

- Repartition vs Coalesce
  - To mitigate this and balance out the data across partitions, repartition is used. This would involve a full shuffling of data and spark will attempt to make all partitions even.
    - Repartition can either increase or decrease the number of partitions.

  - Coalesce is used to reduce the number of partitions when there are too many. It will first combine the partitions on the same machine, and only if required the partitions across executors will be shuffled and combined.
    - Spark will try to minimize the shuffling here.

  - From a performance perspective coalesce should always be preferred over repartitioning.

RDD Operations

- Transformations
  - Create a new dataset from an existing one
  - “map” is a transformation that passes each dataset element through a function and returns a new RDD representing the results
  - *E.g., map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, and coalesce.*

- Actions
  - Return a value to the driver program after running a computation on the dataset.
  - “reduce” is an action that aggregates all the elements of the RDD using some function and returns the result to the driver program
  - *E.g., reduce, collect, count, first, take, countByKey, and foreach*

RDD: Lazy Evaluation

- All transformations in Spark are lazy, in that they do not compute their results right away. Instead, Spark keeps lineage information of RDDs (all the transformations happened to it).

- The transformations are only computed when an action requires a result to be returned to the driver program.

- This design enables Spark to run more efficiently.

Abstraction of RDDs

- Data Frames
  - Gives the user the feeling that they are working on a table.
  - Spark take the responsibility of converting that code to RDDs
  - Can not use map/ReduceBy functions on data frames
  - Compile time type safety is only partially guaranteed.

- Data Sets
  - Combining best of RDDs and Data Frames.
  - Can use map, ReduceBy functions
  - Guaranteed compile time type safety.
  - Memory efficient since encoders are instead of serialization for shuffling and caching data

Spark Auto-optimization

Unresolved Logical Plan	• Based on the lineage information catalyst optimizer will create an unresolved logical plan • Represented as a tree.
Logical Plan	• First analysis is done to validate the syntax, column names etc. then the logical plan is created. • This guarantees that you have something that can be executed.
Optimized Logical Plan	• Then this logical plan is optimized using predicate/filter pushdown and other techniques. • This optimized logical plan just states what we expect as output.
Physical Plan	• Based on this logical plan DAG scheduler creates a physical plan, a DAG of stages • Explains how we can achieve output E.g. How the joins to be performed, hashed broadcast etc.
Cost Model	• For each possible physical plans time and resources would be analysed in creating cost model
RDD Code	• The RDD code would be generated for the most optimal plan

READING

- Spark: The Definitive Guide by Bill Chambers, Matei Zaharia Released February 2018