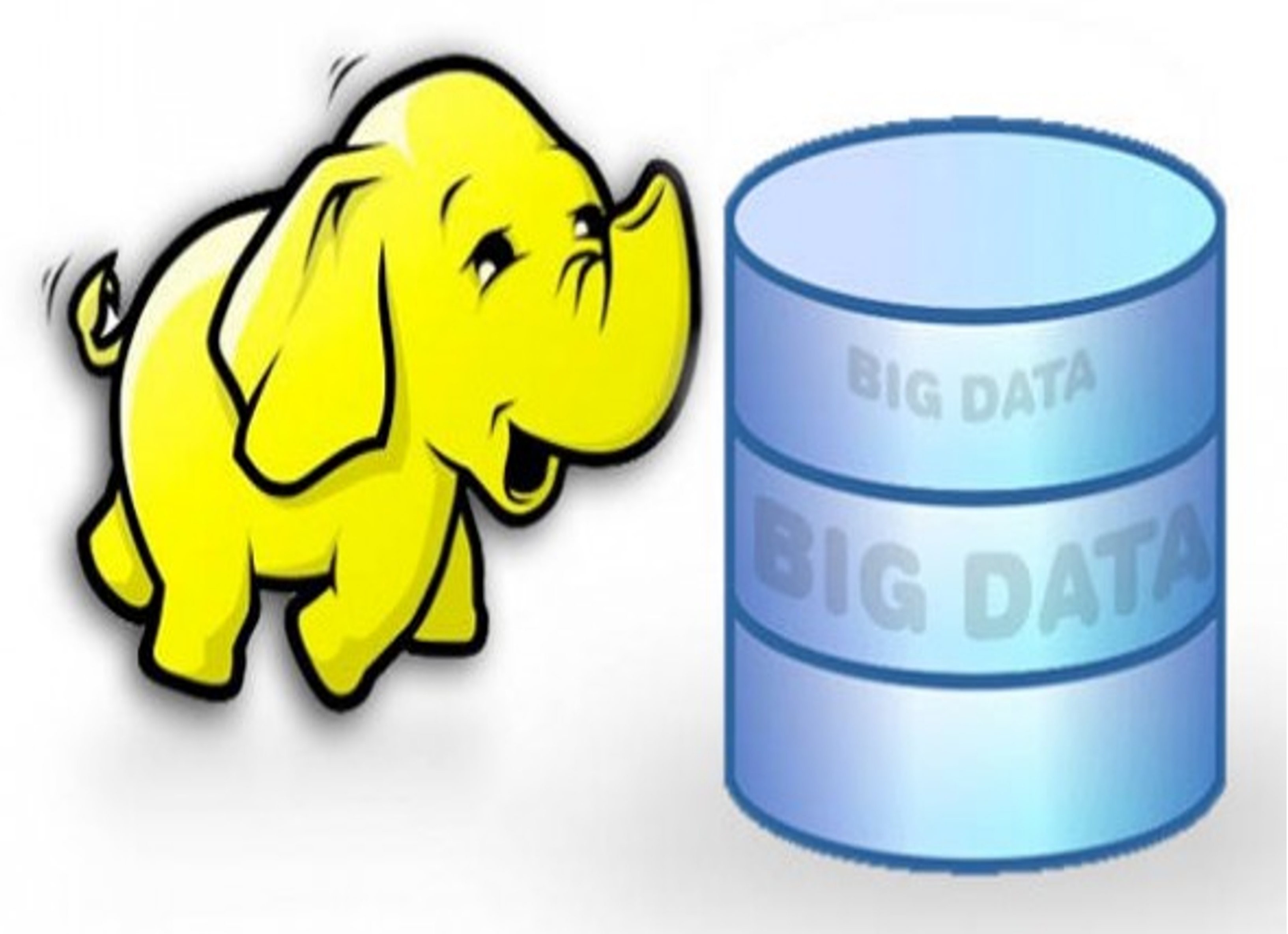
Spark Frameworks and Architecture

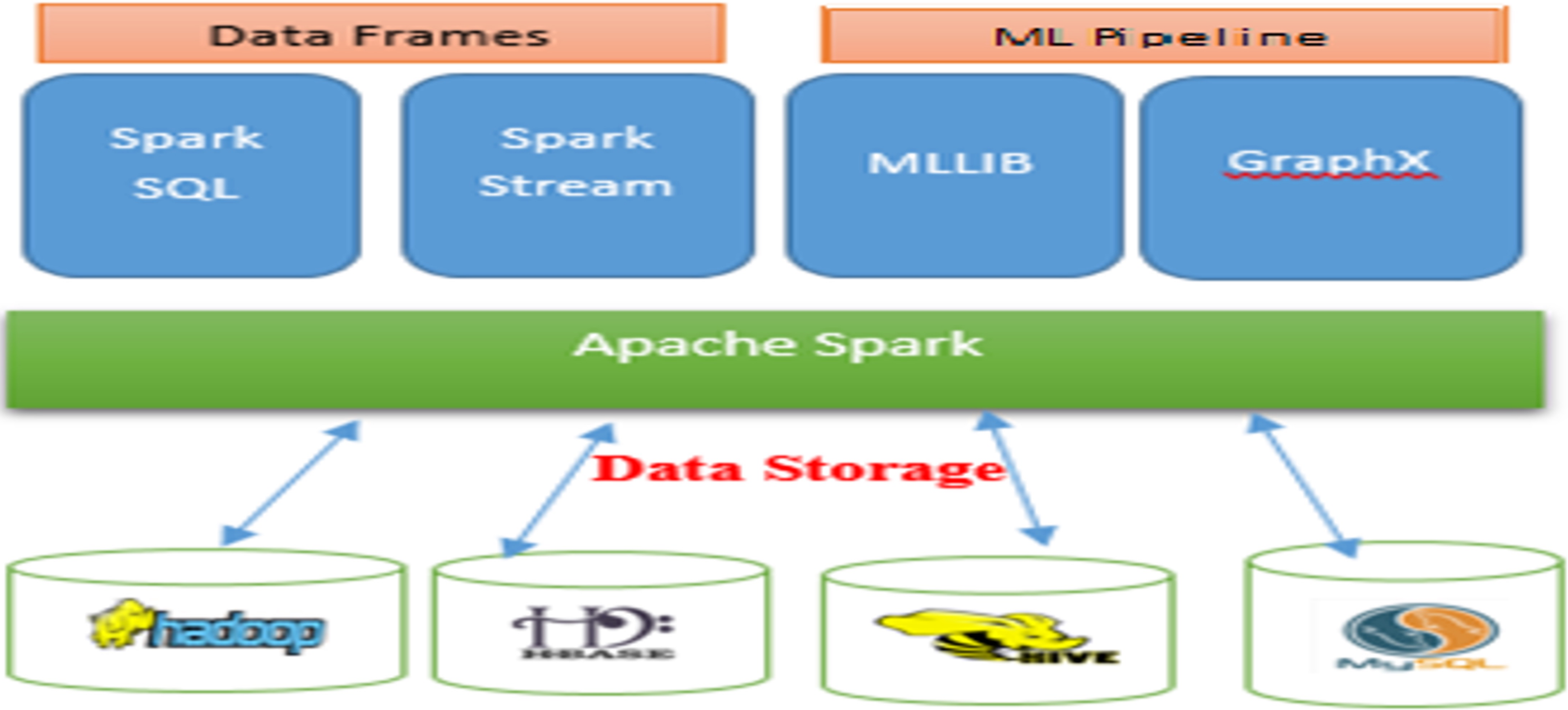


Jai Prakash Verma

Agenda

- Big Data
- Big Data Analytics: Open Source Solutions
- Introdution: Spark
- Spark Vs MapReduce
- Spark Essentials
- Spark Architecture
- Spark Components
- Advanced Spark Programming

Spark Architecture



Spark Components

- Apache Spark Core
- Spark SQL
- Spark Streaming
- MLlib (Machine Learning Library)
- GraphX

Apache Spark Core

- Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

Spark SQL

- Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.
- Blurs the lines between RDDs and relational tables.
- Intermix SQL commands to query external data, along with complex analytics, in a single app:
  - Allows SQL extentions based on Mllib
  - Shark is being migrated to SparkSQL
  - Demo

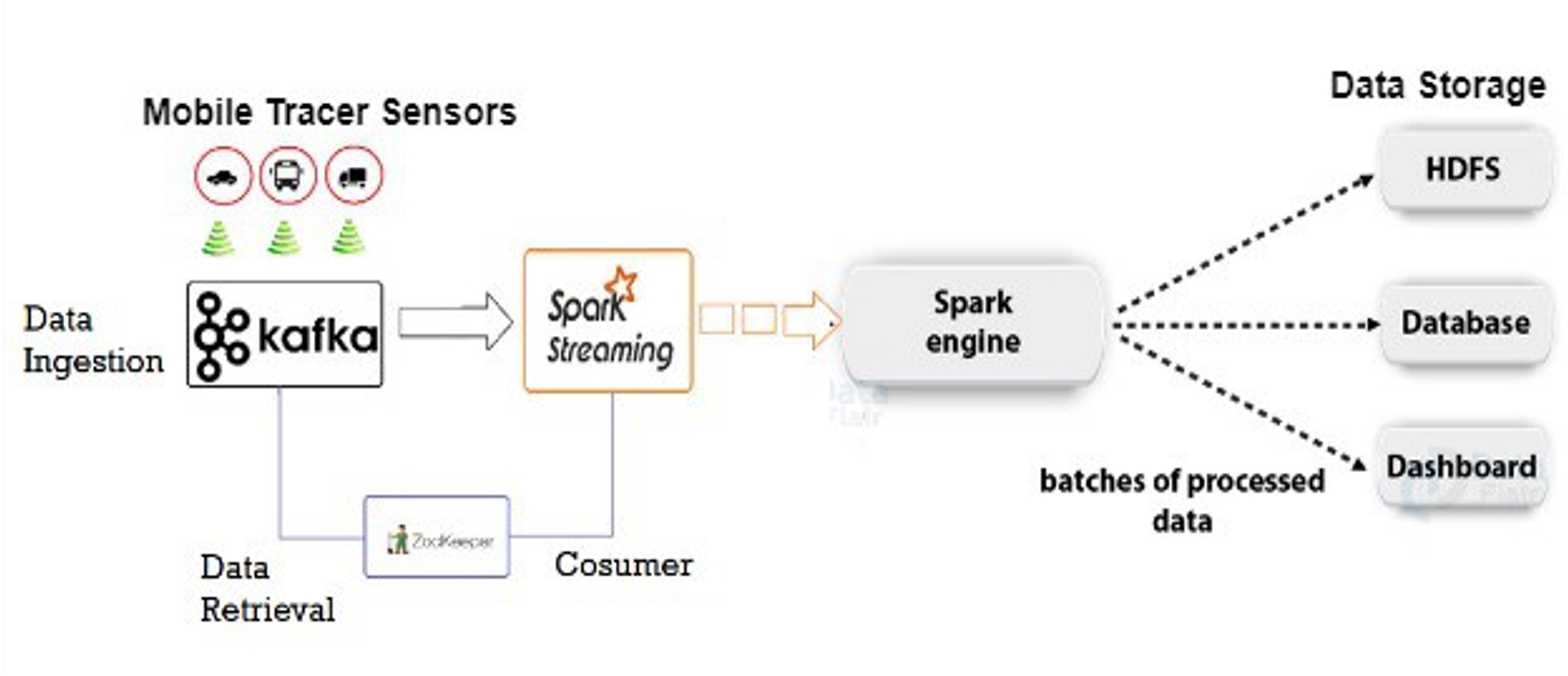
SparkSQL Continue…

- from pyspark.sql import SQLContext
- from pyspark import SparkContext
- sc = SparkContext()
- sqlCtx = SQLContext(sc)
- # Load a text file and convert each line to a dictionary!
- lines = sc.textFile("hdfs://localhost:9000/user/people.txt")
- parts = lines.map(lambda l: l.split(","))
- people = parts.map(lambda p: {"name": p[0], "age": int(p[1])})
- # Infer the schema, and register the SchemaRDD as a table.!
- # In future versions of PySpark we would like to add support !
- # for registering RDDs with other datatypes as tables!
- peopleTable = sqlCtx.inferSchema(people)
- peopleTable.registerAsTable("people")
- # SQL can be run over SchemaRDDs that have been registered as a table!
- teenagers = sqlCtx.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
- teenNames = teenagers.map(lambda p: "Name: " + p.name)
- teenNames.collect()
- teenNames.saveAsTextFile("hdfs://localhost:8020/user/output7")

Spark Streaming

- Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.
- Spark Streaming extends the core API to allow high-throughput, fault-tolerant stream processing of live data streams

Spark Streaming- Use Case



Example- Spark Streaming

- Start Zookeeper :- Since zookeepr is a long-running service, you should run it in its own terminal
  - sudo zookeeper-server-start /etc/kafka/zookeeper.properties

- Start Kafka :- also in its own termin
  - sudo kafka-server-start /etc/kafka/server.properties

- Start producer :- use a new terminal
  - kafka-console-producer --broker-list localhost:9092 --topic test

- Start Kafka Consumer :- in a new terminal
  - kafka-console-consumer --zookeeper localhost:2181 --topic test --from-beginning

- For Spar Streaming:
  - spark-submit --jars spark-streaming-kafka-assembly_2.10-1.6.0.jar kafkawordcount.py localhost:2181 test

  - Source: https://datasciencenovice.wordpress.com/2016/07/04/installing-kafka-spark-on-ubuntu-14-04-16-04-lts/

MLlib (Machine Learning Library)

- MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture.
- Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout (before Mahout gained a Spark interface).

Machine Learning Algorithm - MLlib

- MLlib is Spark’s machine learning (ML) library.

- Its goal is to make practical machine learning scalable and easy.

- It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.

- Demo : MLlib Algorithm

- Source:http://spark.apache.org/docs/latest/mllib-guide.html

Example - MLLib

- Clustering:
  - spark-submit kmean_test_center.py kdata.txt 3
- Classification
  - spark-submit multilayer_perceptron_classification.py
- Frequent Itemset
  - spark-submit FPGrowth.py

GraphX

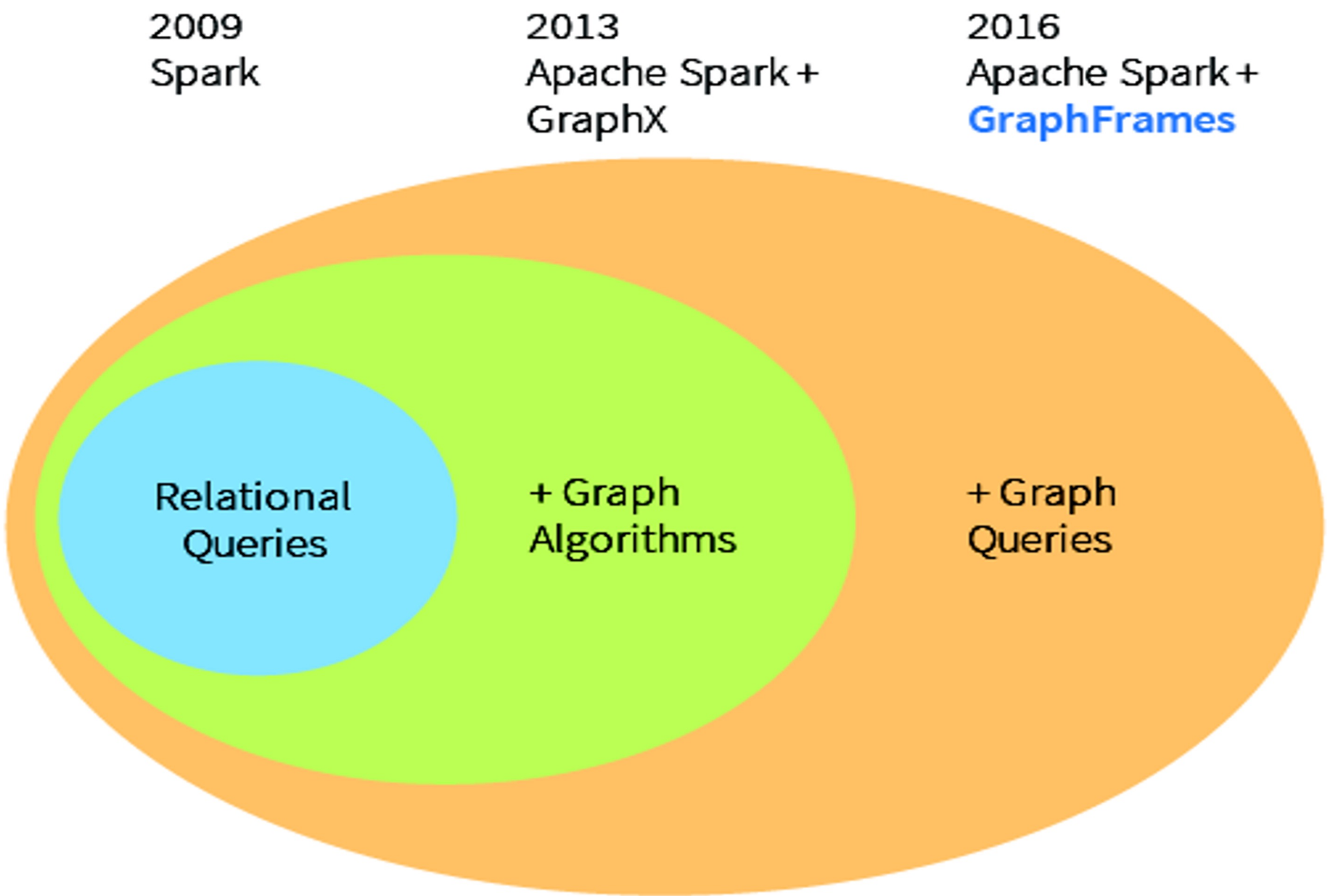
- GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

GraphFrame

- GraphFrames is an API for doing Graph Analytics on Spark DataFrames.

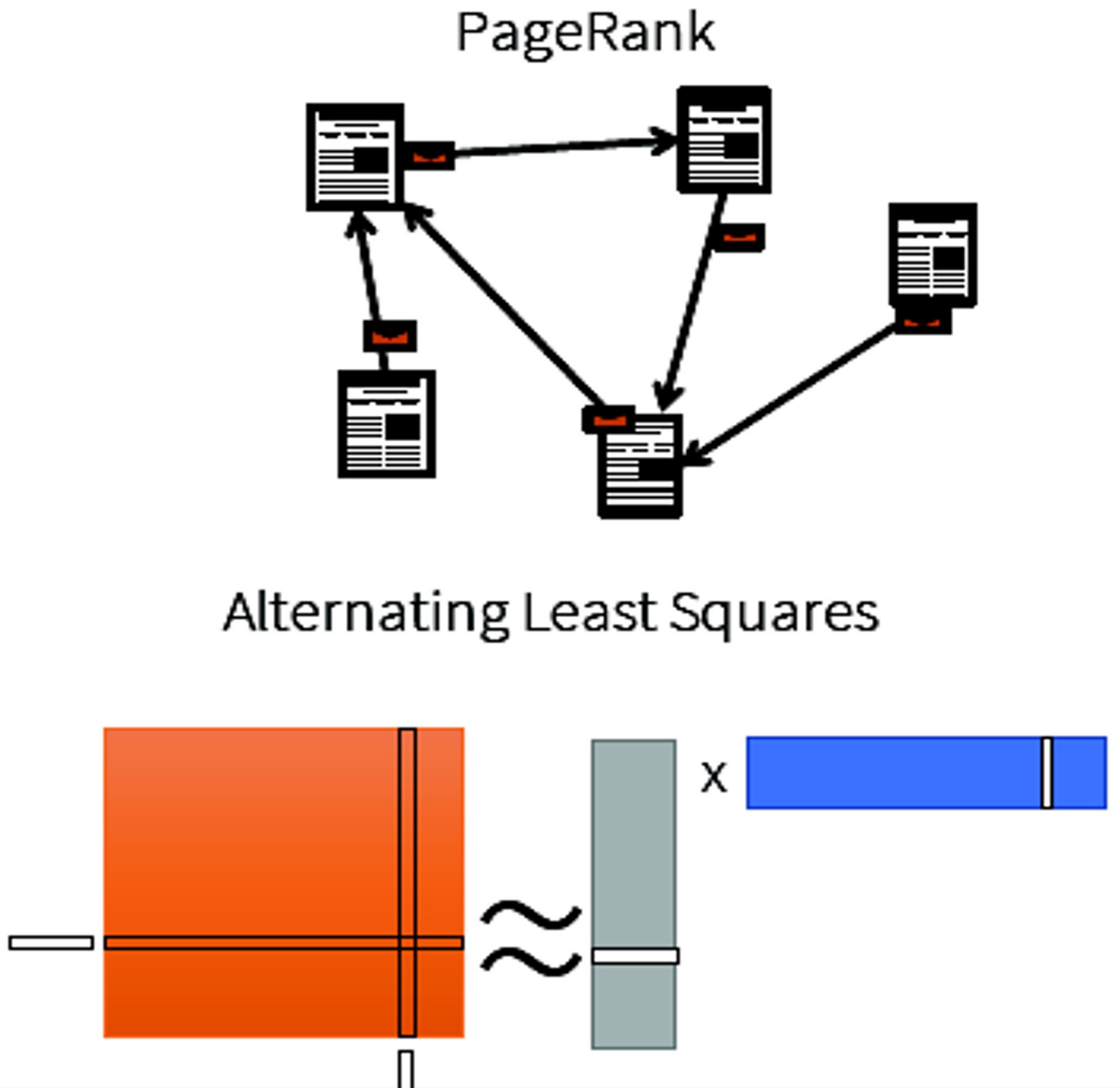
- This way, we can try to recreate SQL queries in Graphs and have a better grasp of the graph concepts.

GraphFrames

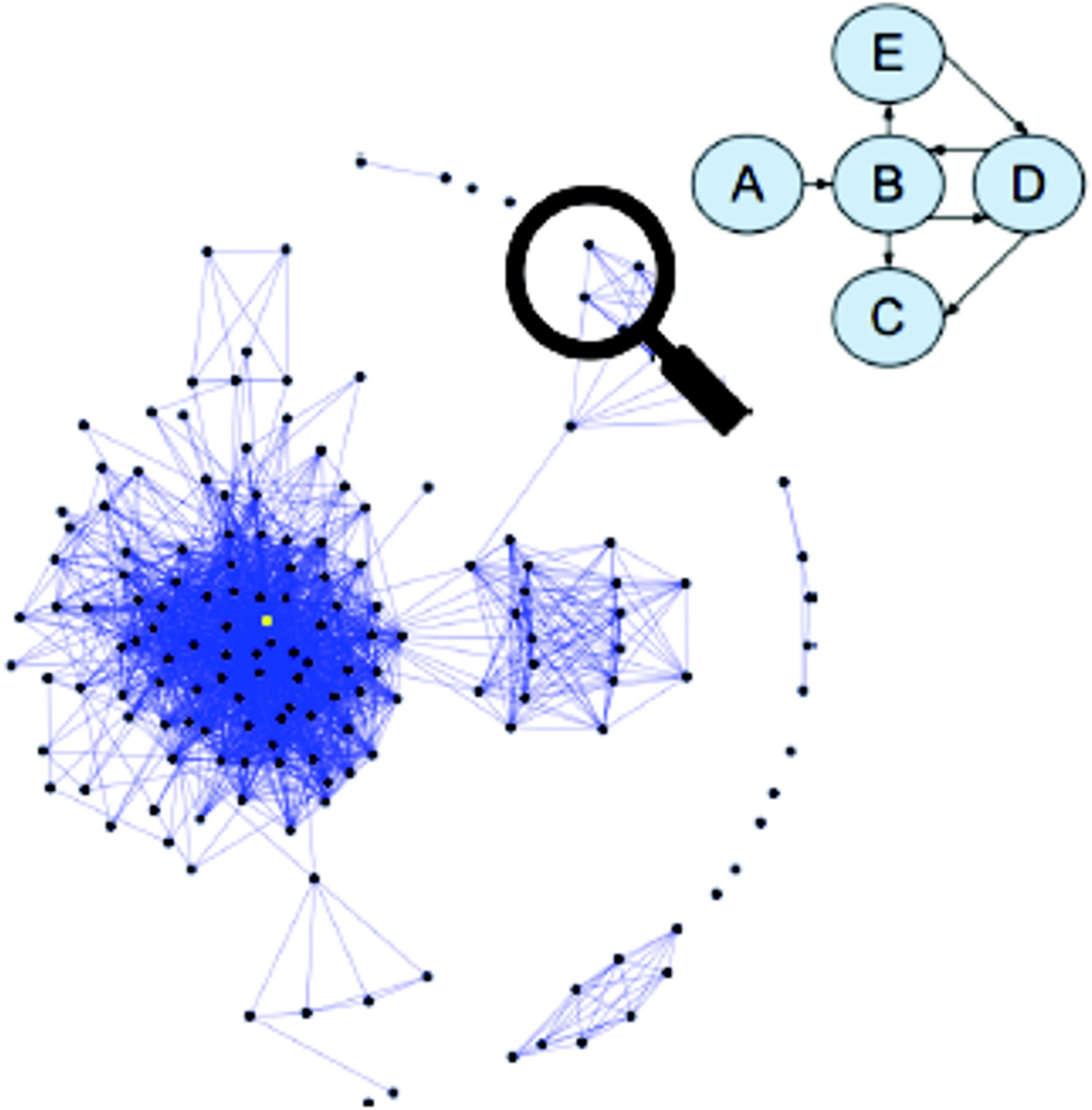


Graph Algorithm vs Graph Queries

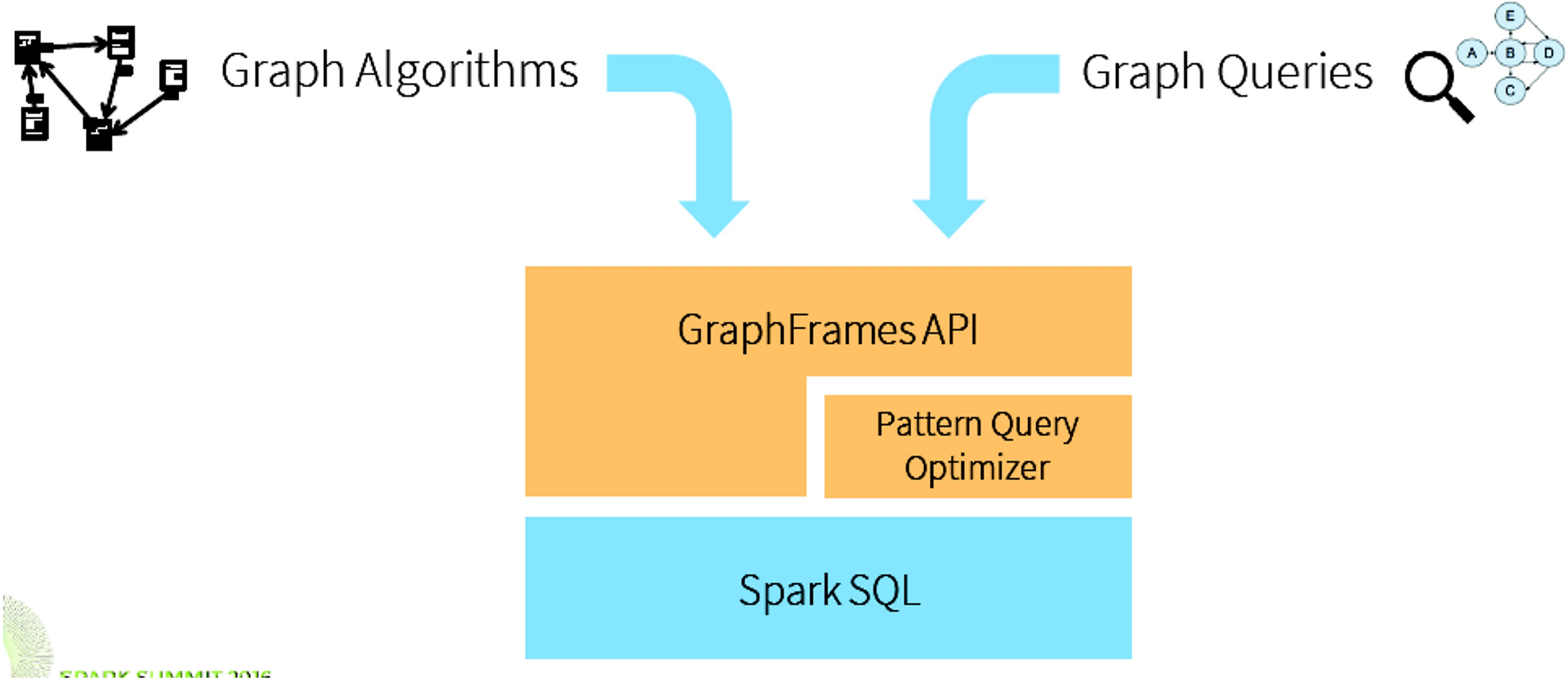
Graph Algorithms



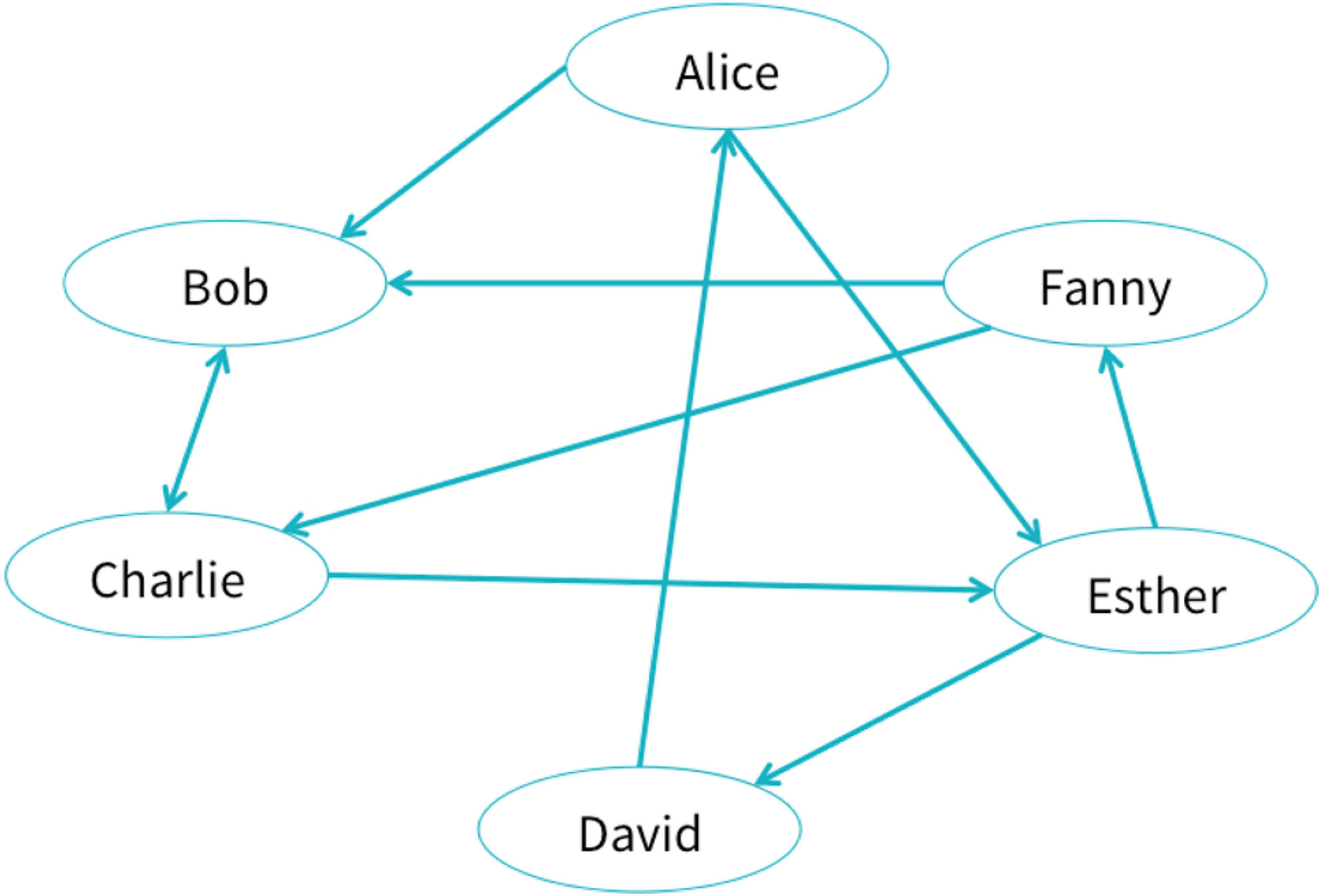
Graph Queries



GraphFrames



An example social network



Demo Graph Analytics

- Using GraphFrames with pyspark
- pyspark --packages graphframes:graphframes:0.1.0-spark1.6
  - import graphframes
  - from graphframes import *

BI Questions

- Which users are most influential?
- Users A and B do not know each other, but should they be introduced?

Creating GraphFrames

- Vertex DataFrame
- Edge DataFrame

Create a Vertex DataFrame with unique ID column "id"

```
v = sqlContext.createDataFrame([
      ("a", "Alice", 34),
      ("b", "Bob", 36),
      ("c", "Charlie", 30),
      ("d", "David", 29),
      ("e", "Esther", 32),
      ("f", "Fanny", 36),
     ("g", "Gabby", 60) ], ["id", "name", "age"])
```

Create an Edge DataFrame with "src" and "dst" columns

```
e = sqlContext.createDataFrame([
("a", "b", "friend"),
("b", "c", "follow"),
("c", "b", "follow"),
("f", "c", "follow"),
("e", "f", "follow"),
("e", "d", "friend"),
("d", "a", "friend"),
("a", "e", "friend")
], ["src", "dst", "relationship"])
```

Graph Analytics (Continue...)

- Create a GraphFrame
  - g = GraphFrame(v, e)
- Query: Get in-degree of each vertex.
  - g.inDegrees.show()
- Query: Count the number of "follow" connections in the graph.
  - g.edges.filter("relationship = 'follow'").count()
- Run PageRank algorithm, and show results.
  - results = g.pageRank(resetProbability=0.01, maxIter=20)
  - results.vertices.select("id", "pagerank").show()

Graph Analytics (Continue...)

- How many users in our social network have “age” > 35?
  - `g.vertices.filter("age > 35").show()`
- How many users have at least 2 followers?
  - `g.inDegrees.filter("inDegree >= 2").show()`

Graph algorithms support complex workflows

- what are the most important users?
  - results = g.pageRank(resetProbability=0.15, maxIter=10)
  - results.vertices.show()

GraphX algorithms supported by GraphFrames

- PageRank: Identify important vertices in a graph
- Shortest paths: Find shortest paths from each vertex to landmark vertices
- Connected components: Group vertices into connected subgraphs
- Strongly connected components: Soft version of connected components
- Triangle count: Count the number of triangles each vertex is part of
- Label Propagation Algorithm (LPA): Detect communities in a graph

Spark Programming

- Spark contains two different types of shared variables − one is broadcast variables and second is accumulators.
  - Broadcast variables − used to efficiently, distribute large values.
  - Accumulators − used to aggregate the information of particular collection.

Broadcast Variables

- Broadcast variables let programmer keep a read only variable cached on each machine rather than shipping a copy of it with task.
- For example, to give every node a copy of a large input dataset efficiently
- Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

Broadcast Variables continue....

- >>> broadcastVar = sc.broadcast(list(range(1, 4)))
- 16/11/18 21:57:43 INFO storage.MemoryStore: Block broadcast_0 stored as values in memory (estimated size 296.0 B, free 296.0 B)
- 16/11/18 21:57:43 INFO storage.MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 101.0 B, free 397.0 B)
- 16/11/18 21:57:43 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on localhost:44951 (size: 101.0 B, free: 511.1 MB)
- 16/11/18 21:57:43 INFO spark.SparkContext: Created broadcast 0 from broadcast at PythonRDD.scala:430
- >>> broadcastVar.value
- [1, 2, 3]
- >>>

Accumulators

- Accumulators are variables that can only be “added” to through an associative operations.
- Used to implement counters and sums, efficiently in parallel
- Spark nativily supports accumulators of numeric value types and standard mutable collections, and programmers can extend of new types
- Only the driver program can read an accumulator's value, not the task.

Accumulators continue….

- accum = sc.accumulator(0)
- rdd = sc.parallelize([1, 2, 3, 4])
- def f(x):
-         global accum
-         accum += x
- rdd.foreach(f)
- accum.value

Spark - Installation

- For Spark Installation with Hadoop:
  - http://hadooptutorials.co.in/tutorials/spark/install-apache-spark-on-ubuntu.html
- For Spark Web Console:
  - http://localhost:4040
- For Spark Tutorial:
  - https://www.tutorialspoint.com/apache_spark/apache_spark_quick_guide.htm

- Questions
- Jaiprakash.verma@nirmauni.ac.in
- 9427621081