Big Data Analytics

Dr. Rajiv Misra
Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

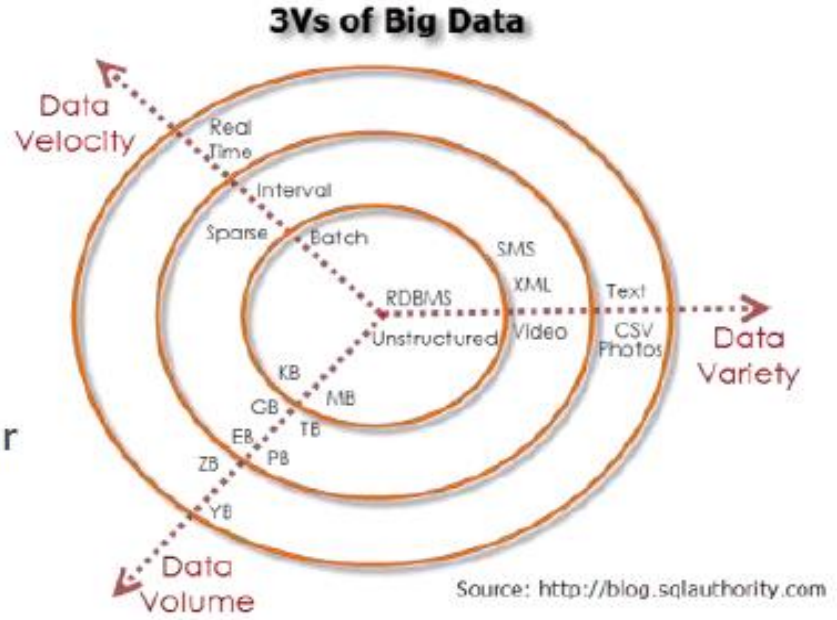
DATA VS BIG DATA

Big data is just data with:

- More **volume**
- Faster data generation (**velocity**)
- Multiple data format (**variety**)

World's data volume to grow 40% per year & 50 times by 2020 [1]

Data coming from various human & machine activity



CHALLENGES

- More data = more storage space
  - More storage = more money to spend ☹ (RDBMS server needs very costly storage)
- Data coming faster
  - Speed up data processing or we'll have backlog
- Needs to handle various data structure
  - How do we put JSON data format in standard RDBMS?
  - Hey, we also have XML format from other sources
  - Other system give us compressed data in gzip format
- Agile business requirement.
  - On initial discussion, they only need 10 information, now they ask for 25? Can we do that? We only put that 10 in our database
  - Our standard ETL process can't handle this

STORAGE COST

In Terms of storage cost, Hadoop has lower comparing to standard RDBMS.

Hadoop provides highly scalable storage and process with fraction of the EDW Cost

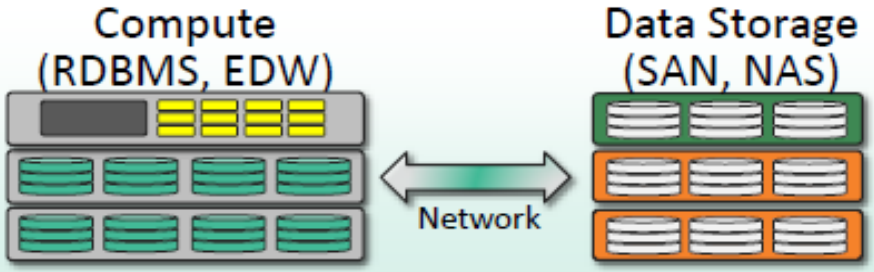
HADOOP PRICING ADVANTAGE ON A PER TERABYTE BASIS

	Hadoop	Netezza	Exadata	Extreme Data Appliance (1650)
Cost / Terabyte	**$333**	**$10,000**	**$14,000**	**$16,500***
Hadoop Benefit		30x saving	42x saving	50x saving

*Source: MapR (June 2013); * Teradata has since launched a new Extreme Data Appliance (1700) at $2,000/TB in October 2013*

STORAGE & COMPUTE TOGETHER

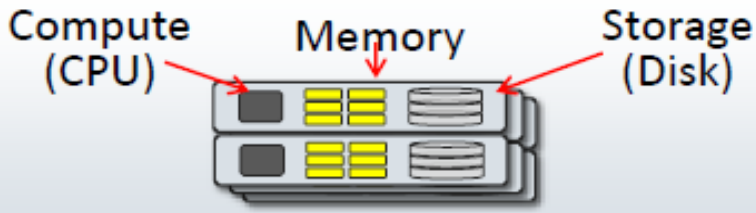
The Old Way



Expensive, Special purpose, "Reliable" Servers
Expensive Licensed Software

- Hard to scale
- Network is a bottleneck
- Only handles relational data
- Difficult to add new fields & data types

The Hadoop Way

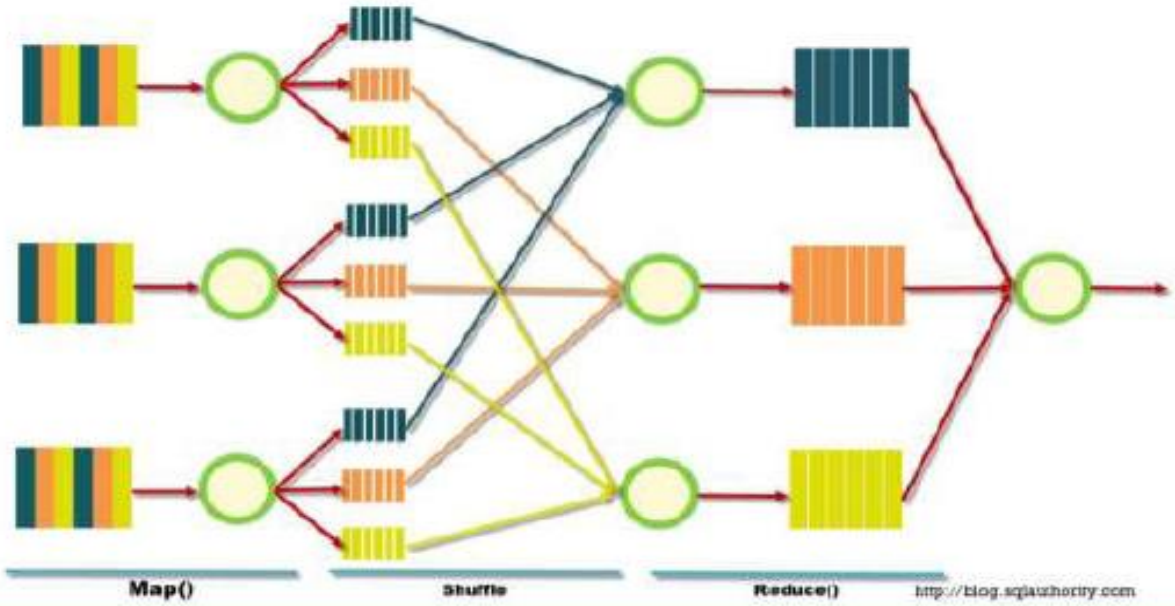


Commodity "Unreliable" Servers
Hybrid Open Source Software

- Scales out forever
- No bottlenecks
- Easy to ingest any data
- Agile data access

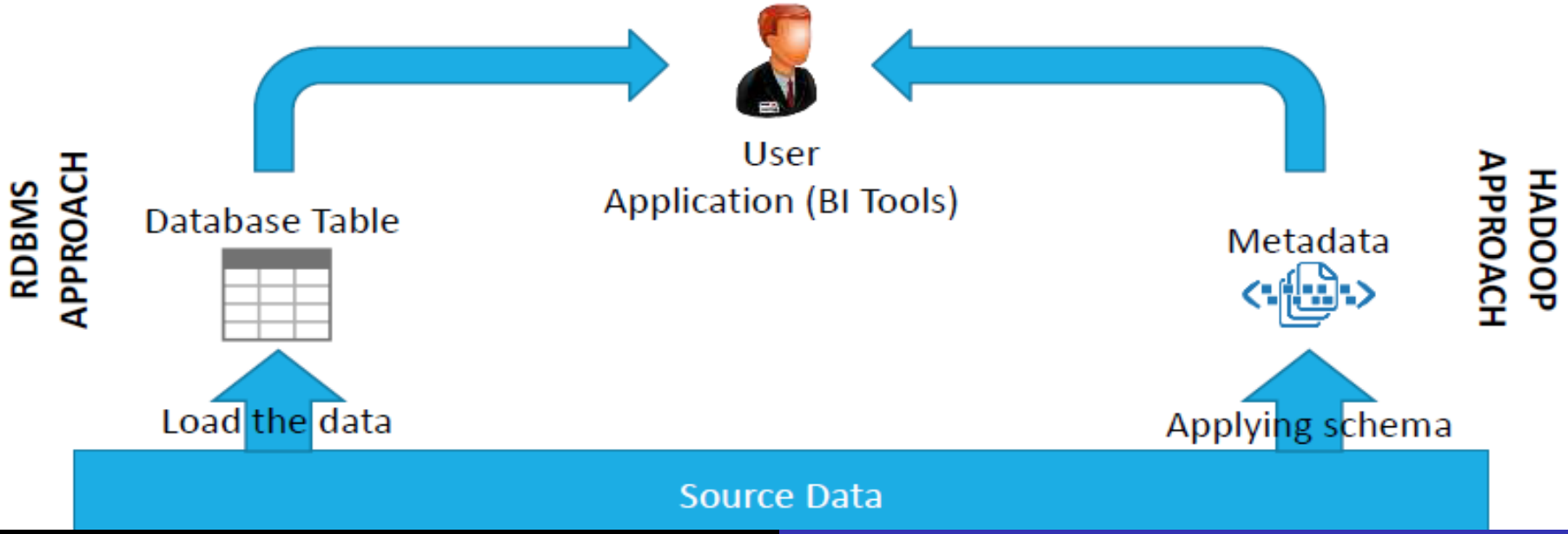
MAP REDUCE APPROACH

- Process data in parallel way using distributed algorithm on a cluster
- Map procedure performs filtering and sorting data locally
- Reduce procedure performs a summary operation (count, sum, average, etc.)



HADOOP vs UNSTRUCTURED DATA

- Hadoop has HDFS (Hadoop Distributed File System)
- It is just file system, so what you need is just drop the file there ☺
- Schema on read concept

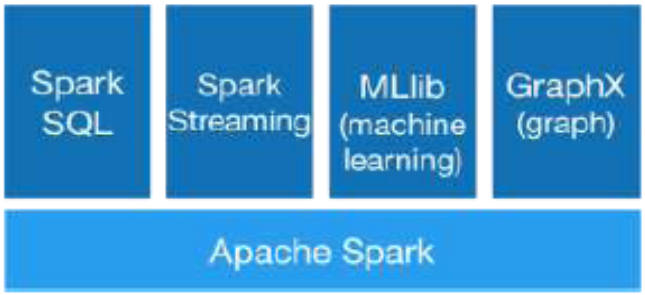


HIVE

- The Apache Hive ™ data warehouse software facilitates querying and managing large datasets residing in distributed storage.
- With Hive you can write the schema for the data in HDFS
- Hive provide many library that enable you to read various data type like XML, JSON, or even compressed format
- You can create your own data parser with Java language
- Hive support SQL language to read from your data
- Hive will convert your SQL into Java MapReduce code, and run it in cluster

- Apache spark is fast and general engine for large-scale data processing
- Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk
- You can write spark application in Java, Scala, Python, or R
- Spark support library to run SQL, streaming, and complex analysis like graph computation and machine learning
- https://spark.apache.org/

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```



DESCRIPTIVE & PREDICTIVE

- Descriptive statistics is the term given to the analysis of data that helps describe, show or summarize data in a meaningful way such that, for example, patterns might emerge from the data.
  - In Information System Design course, most of the student get C grade (11 people). There is 4 people get A, 7 get B, 7 get D, and 7 get E
  - Fulan only post his activity on Facebook at weekend
- Predictive analytics is the branch of data mining concerned with the prediction of future probabilities and trends.
- The central element of predictive analytics is the predictor, a variable that can be measured for an individual or other entity to predict future behavior.

PREDICTIVE ANALYTICS

There is 2 types of predictive analytics:

- Supervised

Supervised analytics is when we know the truth about something in the past

Example:

we have historical weather data. The temperature, humidity, cloud density and weather type (rain, cloudy, or sunny). Then we can predict today weather based on temp, humidity, and cloud density today

Machine learning to be used: Regression, decision tree, SVM, ANN, etc.

- Unsupervised

Unsupervised is when we don't know the truth about something in the past. The result is segment that we need to interpret

Example:

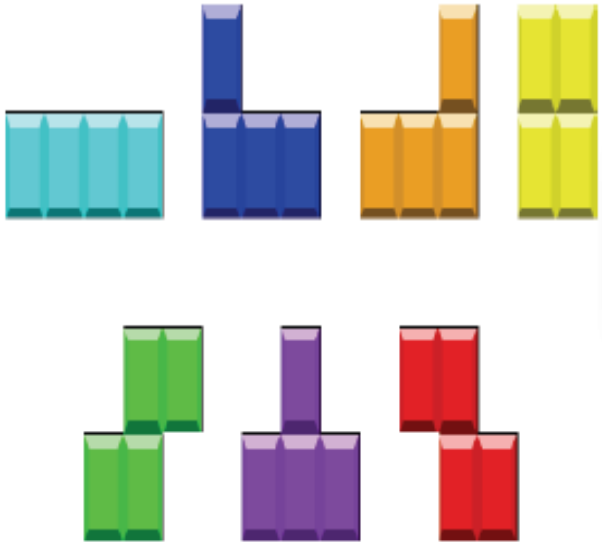
We want to do segmentation over the student based on the historical exam score, attendance, and late history

Mathematics

3 Common Ways of Creating Predictive Analytics:

Compare/Contrast

Fit the Pieces



Group the Knowledge

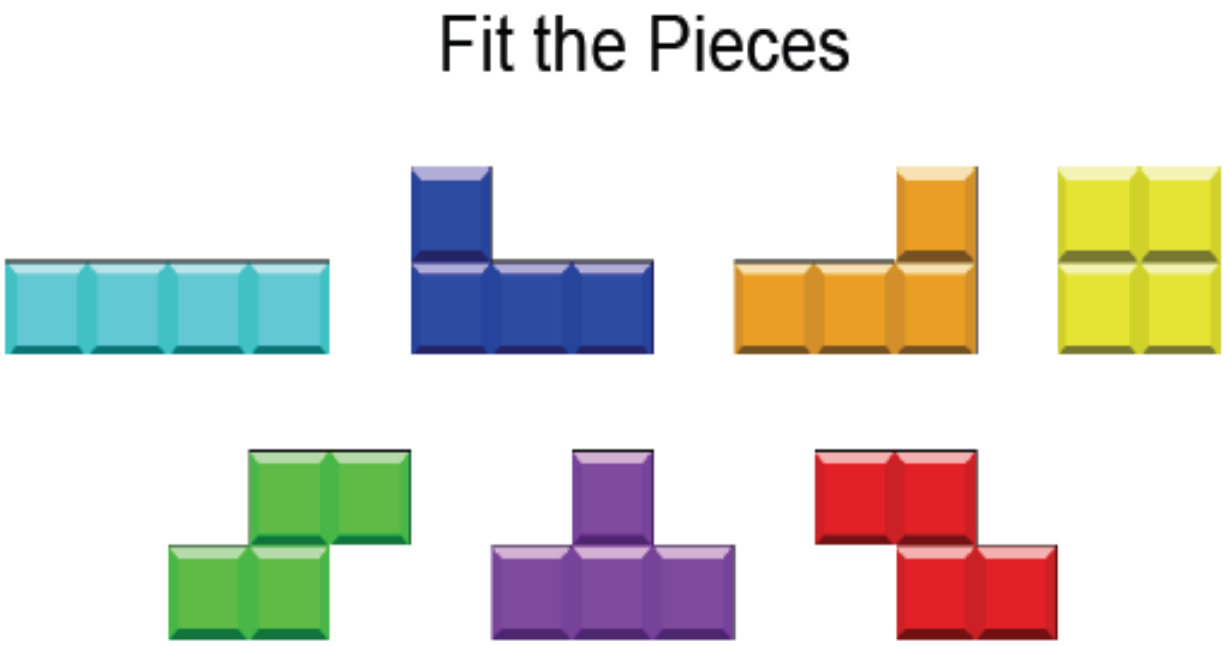
How Do We Make Sense Out of Data?

Multi-Colinearity Analysis

Compare/Contrast

How Do We Make Sense Out of Data?

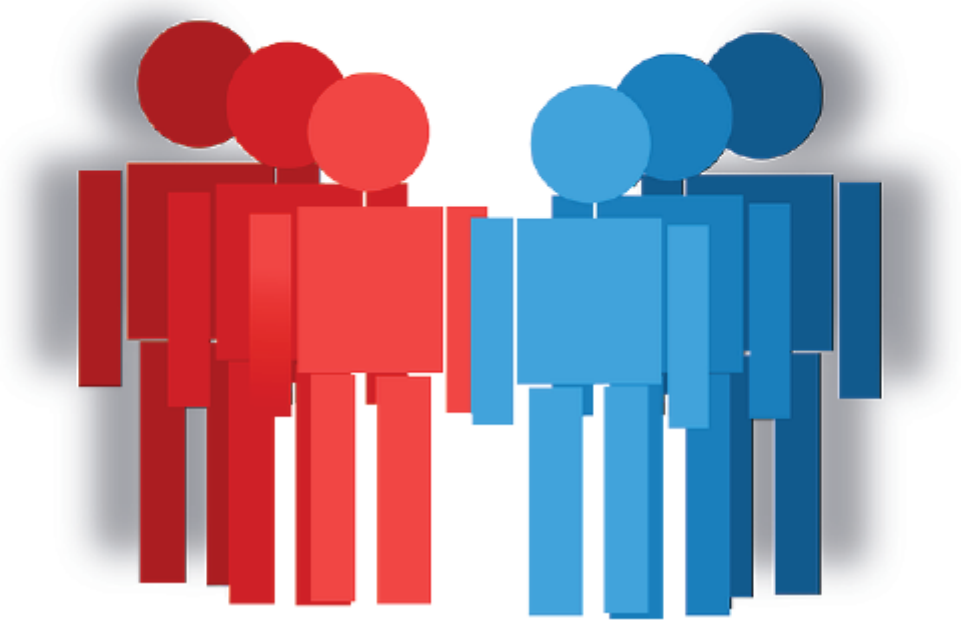
Linear Regression



How Do We Make Sense Out of Data?

Cluster Analysis

Group the Knowledge



BUILDING THE METHODOLOGY

Analysis Domain

- What is the analysis domain? Is it for male only? Is it for housewife or worker? Your "customer" segment has different behavior

Type of Analysis

- Do we need only descriptive analysis? Or we need to go with predictive analysis?

Supervised or Unsupervised?

- Do we need to build unsupervised clustering/segmentation for this analysis?

Define Analysis Time Window

- What time window of data we need for behavior observation?
- What is the prediction time window?
- Is there any seasonal event on that time window?

What is Big Data?

Big data is a collection of data sets so large and complex that it becomes difficult to process using traditional relational database management systems

What's making so much data?

- ubiquitous computing
- more people carrying data-generating devices (mobile phones with facebook, gps, cameras, etc.)

Source of Data Generation

12+ TBs
of tweet data every day

? TBs of data every day

25+ TBs **of log data every day**

30 billion RFID tags today (1.3B in 2005)

4.6 billion camera phones world wide

100s of millions of GPS enabled devices sold annually

2+ billion people on the Web by end 2011

76 million smart meters in 2009… 200M by 2014

Where is the problem?

- Traditional RDBMS queries isn't sufficient to get useful information out of the huge volume of data
- To search it with traditional tools to find out if a particular topic was trending would take so long that the result would be meaningless by the time it was computed.
- Big Data come up with a solution to store this data in novel ways in order to make it more accessible, and also to come up with methods of performing analysis on it.

Challenges:

- Capturing
- Storing
- Searching
- Sharing
- Analyzing
- Visualization

IBM considers Big Data(3V's):

- Big data spans four dimensions: **Volume**, Velocity, Variety.
- **Volume:** Enterprises are awash with ever-growing data of all types, easily amassing terabytes even Petabytes of information.
  - Turn 12 terabytes of Tweets created each day into improved product sentiment analysis
  - Convert 350 billion annual meter readings to better predict power consumption

IBM considers Big Data(3V's):

- Big data spans four dimensions: Volume, **Velocity**, Variety.
- **Velocity:** Sometimes 2 minutes is too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.
  - Scrutinize 5 million trade events created each day to identify potential fraud
  - Analyze 500 million daily call detail records in real-time to predict customer churn faster

IBM considers Big Data(3V's):

- Big data spans four dimensions: Volume, Velocity, **Variety**.
- **Variety:** Big data is any type of data –
  - Structured Data (example: tabular data)
  - Unstructured –text, sensor data, audio, video
  - Semi Structured : web data, log files

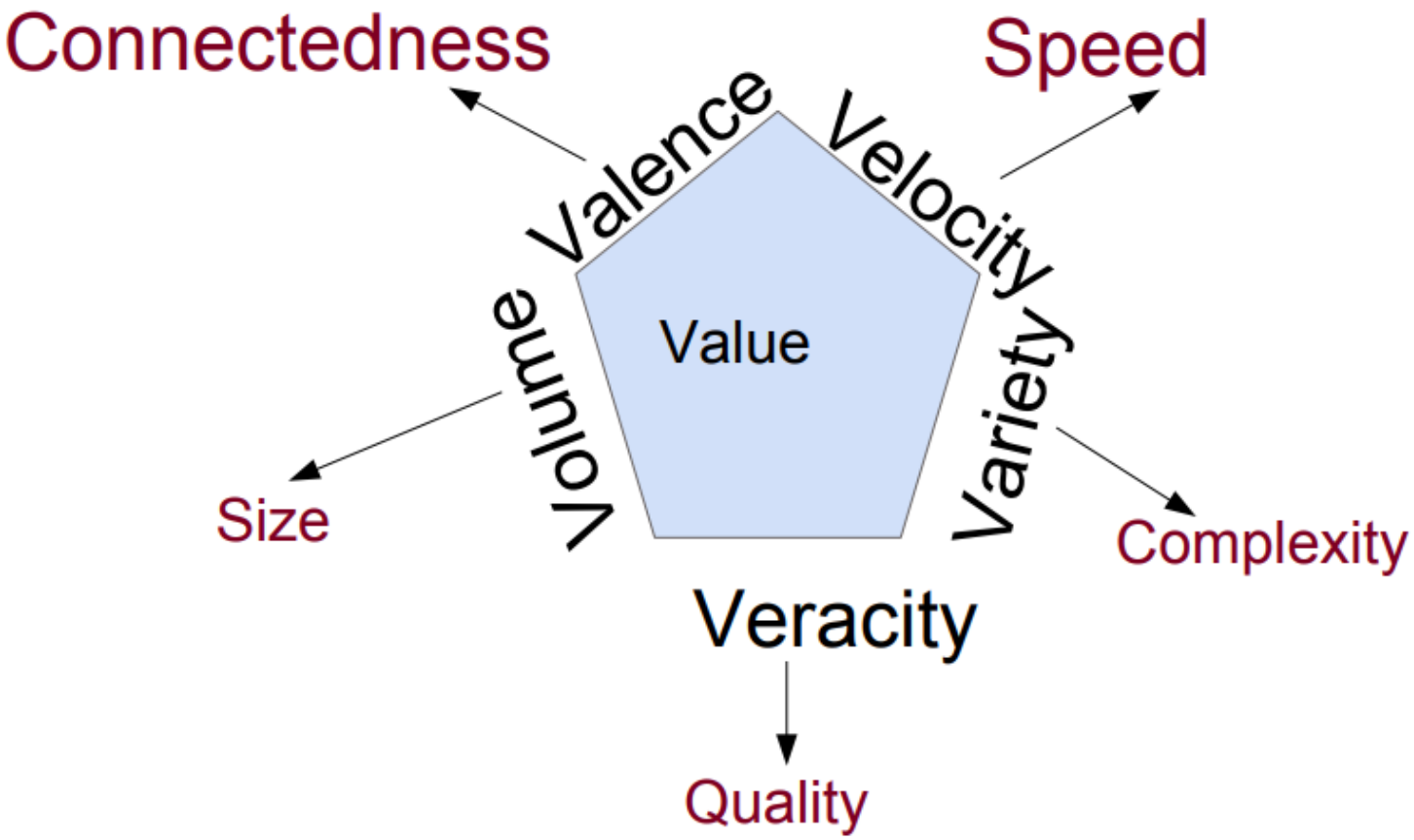
The 3 Big V's (+1) (+ N more)

- Big 3V's
  - Volume
  - Velocity
  - Variety
- Plus 1
  - Value

The 3 Big V's (+1) (+ N more)

- Plus many more
  - Veracity
  - Validity
  - Variability
  - Viscosity & Volatility
  - Viability,
    Venue,
    Vocabulary, Vagueness,
    …

Facts and Figures



Value

- Integrating Data
  - Reducing data complexity
  - Increase data availability
  - Unify your data systems
  - All 3 above will lead to increased data collaboration -> add value to your big data

Veracity

- Veracity refers to the biases ,noise and abnormality in data, trustworthiness of data.
- 1 in 3 business leaders don't trust the information they use to make decisions.
  - How can you act upon information if you don't trust it?
  - Establishing trust in big data presents a huge challenge as the variety and number of sources grows.

Valence

- Valence refers to the connectedness of big data.
- Such as in the form of graph networks

Validity

Accuracy and correctness of the data relative to a particular use

- Example: Gauging storm intensity

  satellite imagery vs social media posts

- prediction quality vs human impact

Variability

How the meaning of the data changes over time

- Language evolution
- Data availability
- Sampling processes
- Changes in characteristics of the data source

Viscosity & Volatility

- Both related to velocity
- Viscosity: *data velocity relative to timescale of event being studied*
- Volatility: *rate of data loss and stable lifetime of data*
  - Scientific data often has practically unlimited lifespan, but social / business data may evaporate in finite time

More V's

- Viability
  - Which data has meaningful relations to questions of interest?
- Venue
  - Where does the data live and how do you get it?
- Vocabulary
  - Metadata describing structure, content, & provenance
  - Schemas, semantics, ontologies, taxonomies, vocabularies
- Vagueness
  - Confusion about what "Big Data" means

Dealing with Volume

- Distill big data down to small information
- Parallel and automated analysis
- Automation requires standardization
- Standardize by reducing Variety:
  - Format
  - Standards
  - Structure

Big Data Enabling Technologies

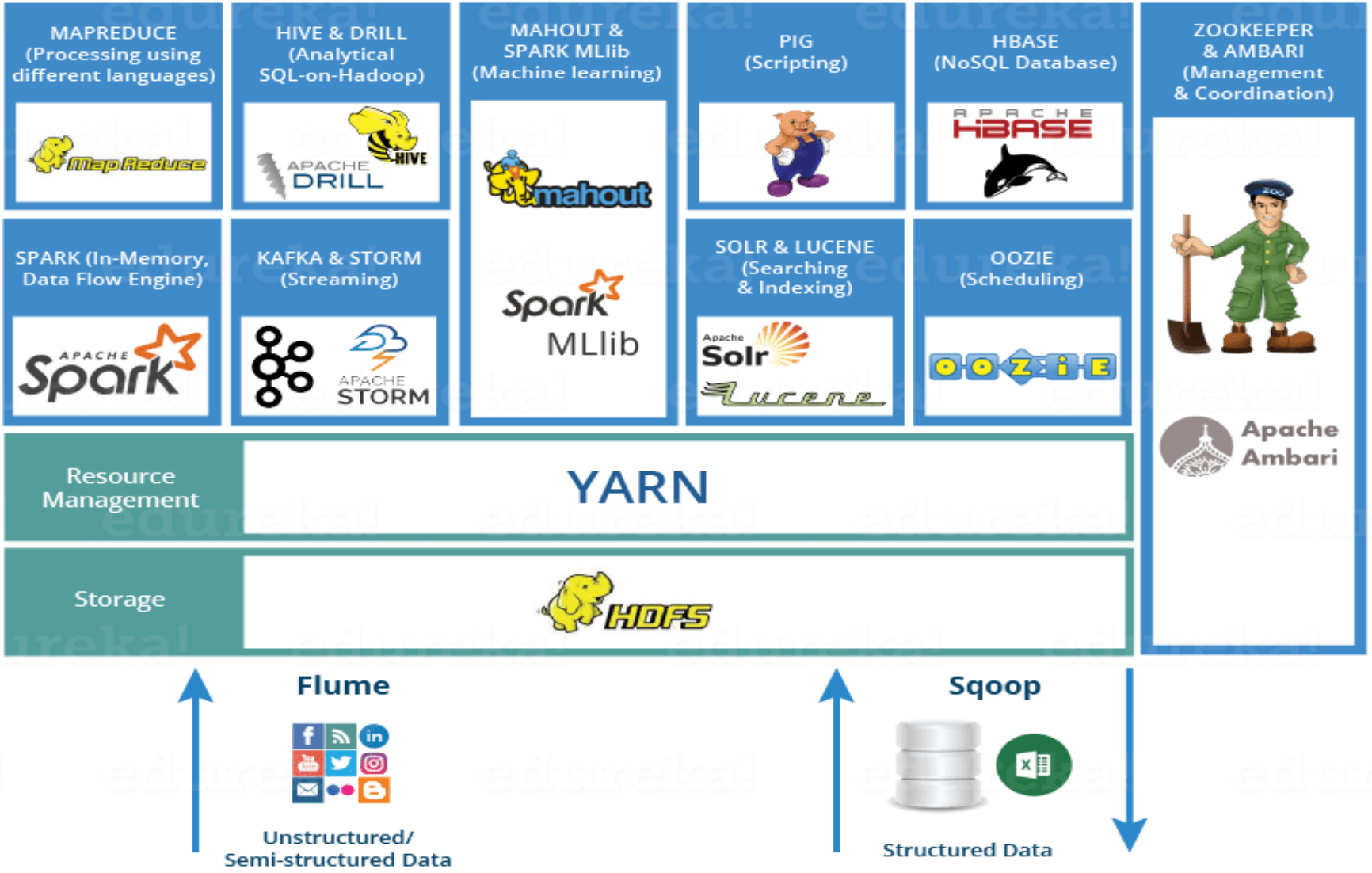
Introduction

- Big Data is used for a collection of data sets so large and complex that it is difficult to process using traditional tools.
- A recent survey says that 80% of the data created in the world are unstructured.
- One challenge is how we can store and process this big amount of data. Later on slide we discuss the top technologies used to store and analyse Big Data.

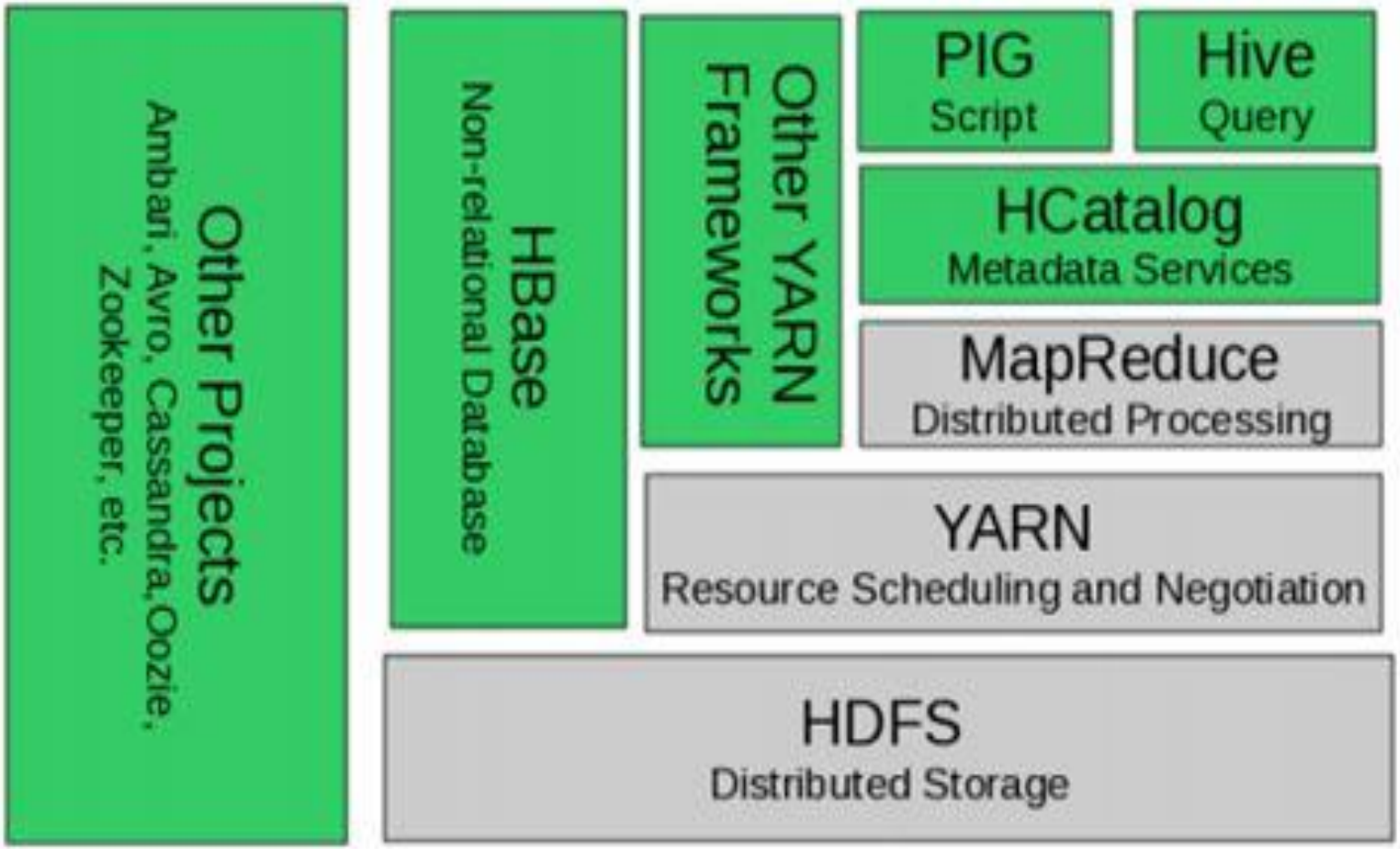
Apache Hadoop

- Apache Hadoop is a java based free software framework that can effectively store large amount of data in a cluster.
- This framework runs in parallel on a cluster and has an ability to allow us to process data across all nodes.
- Hadoop Distributed File System (HDFS) is the storage system of Hadoop which splits big data and distribute across many nodes in a cluster.

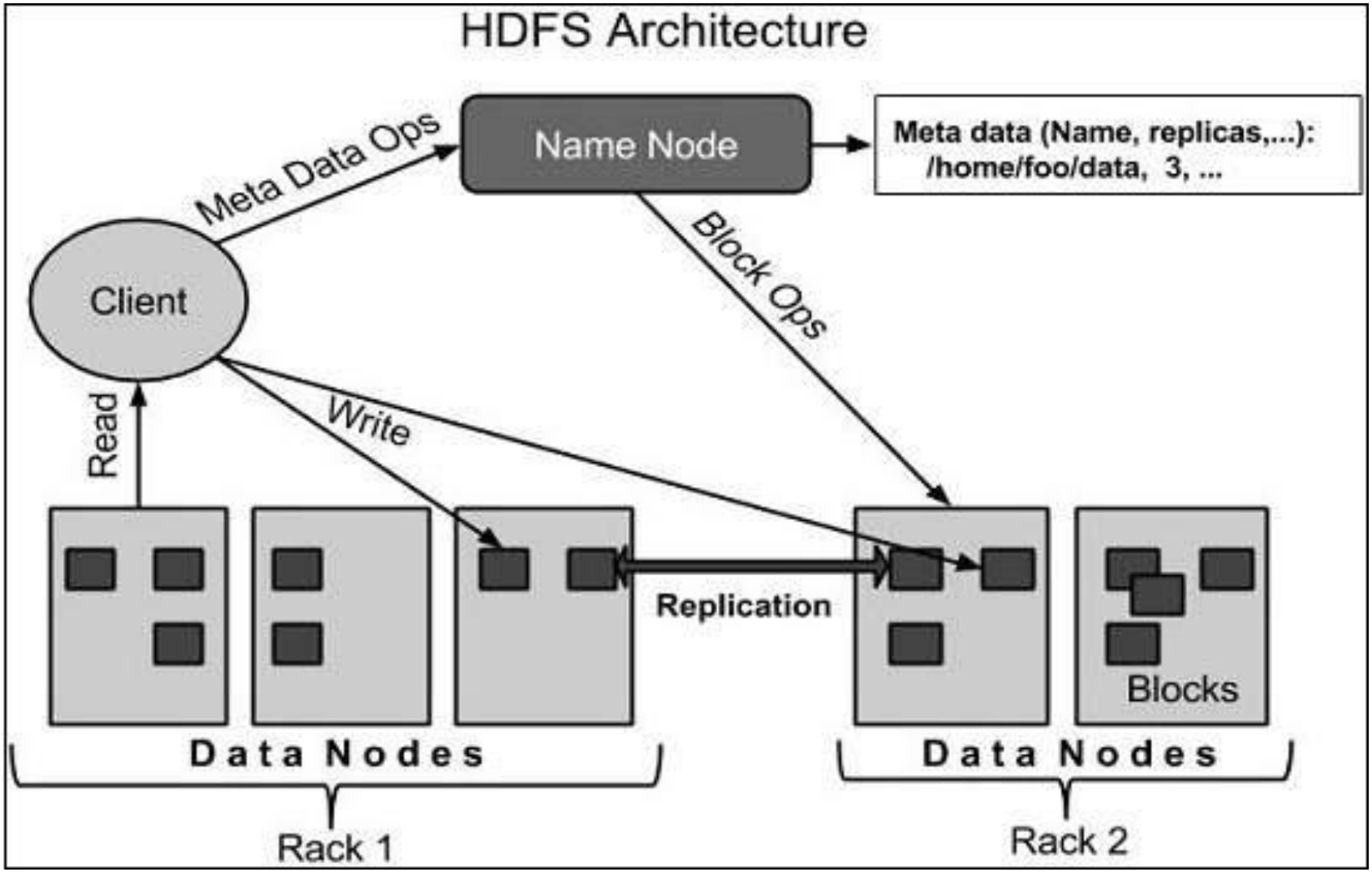
Hadoop Ecosystem



Hadoop Ecosystem



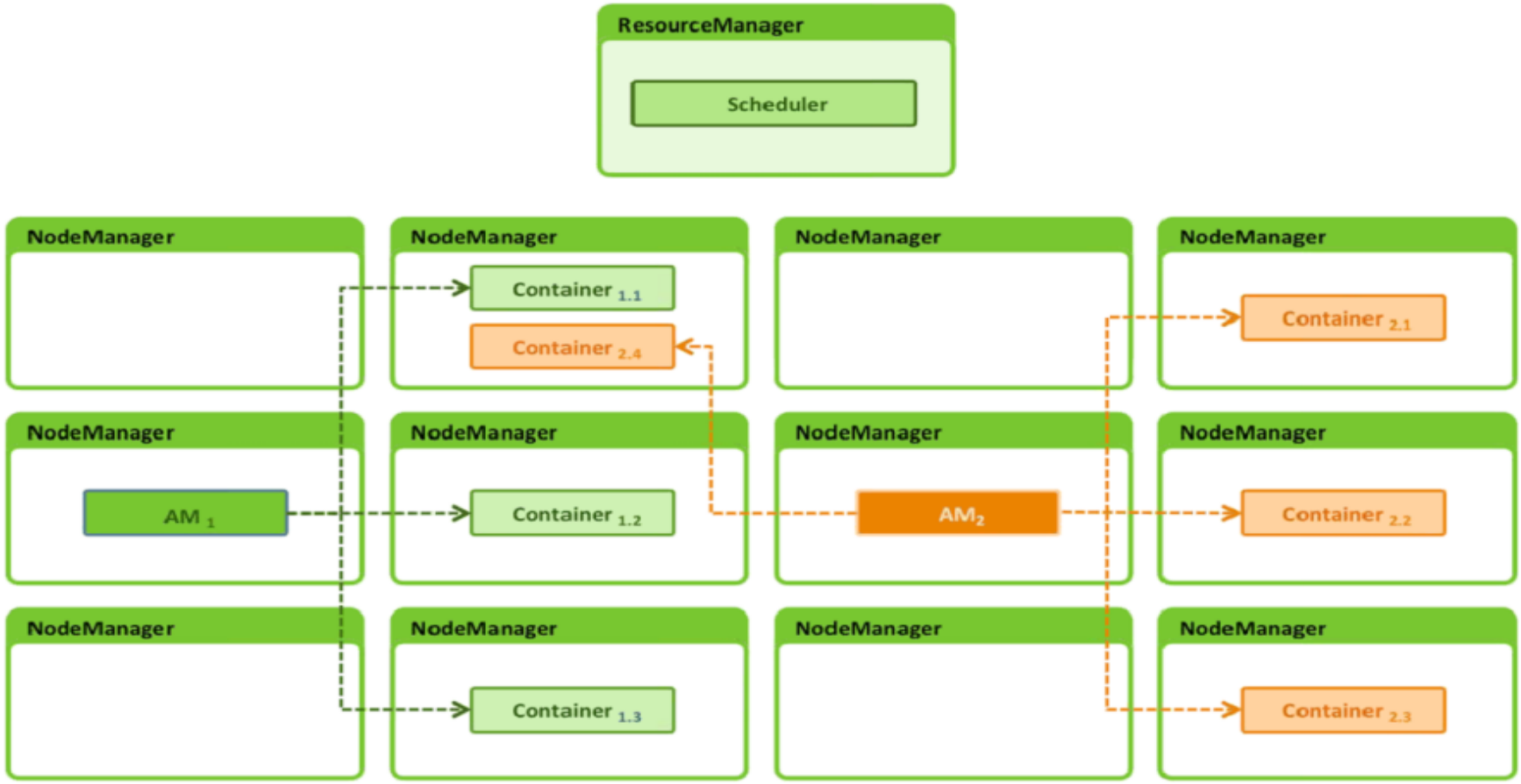
HDFS Architecture



YARN

- YARN – Yet Another Resource Manager.
- Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework.
- YARN is responsible for allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes.

YARN Architecture

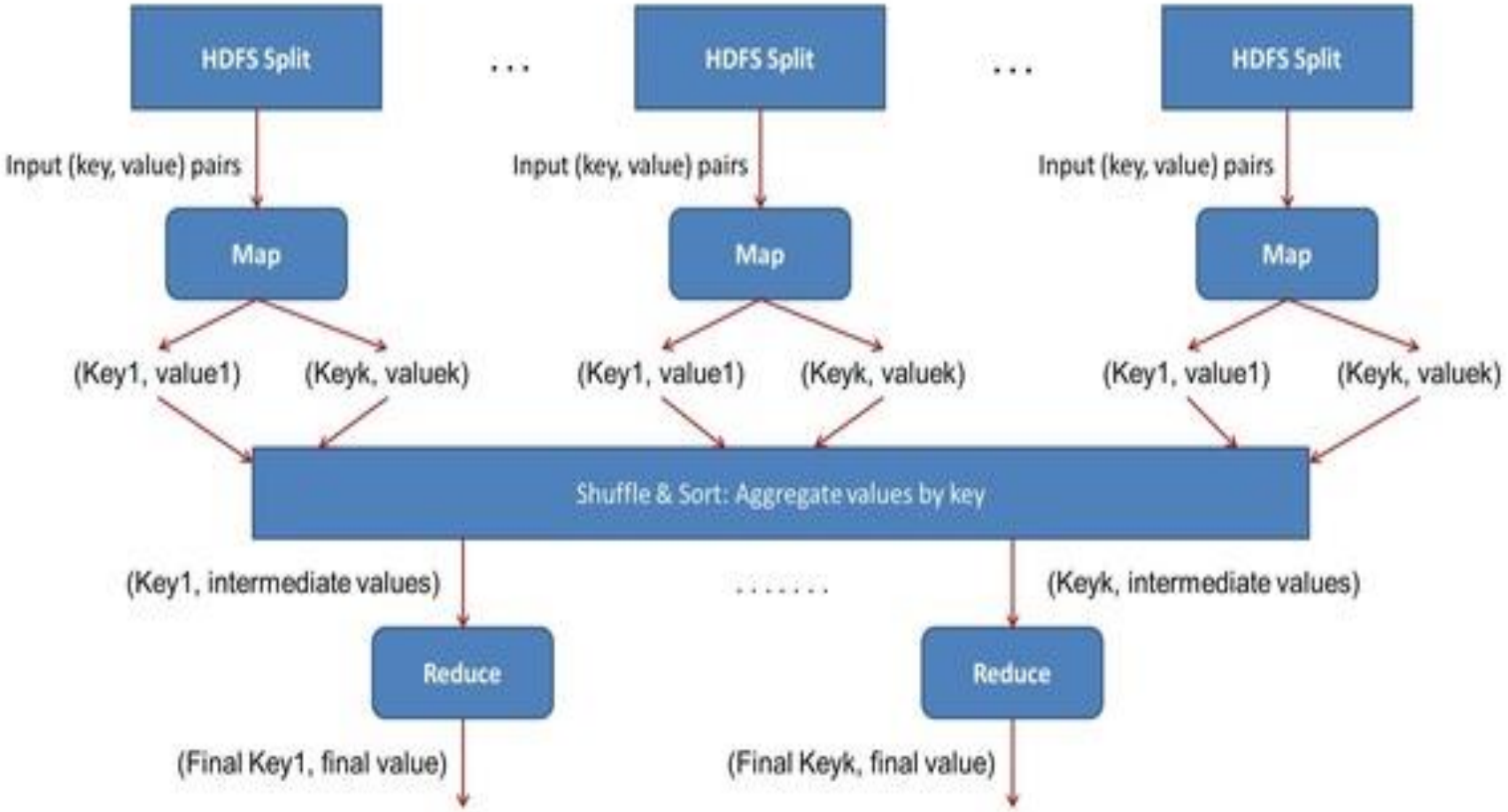


Map Reduce

- **MapReduce** is a programming model and an associated implementation for processing and generating large data sets.

- Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key

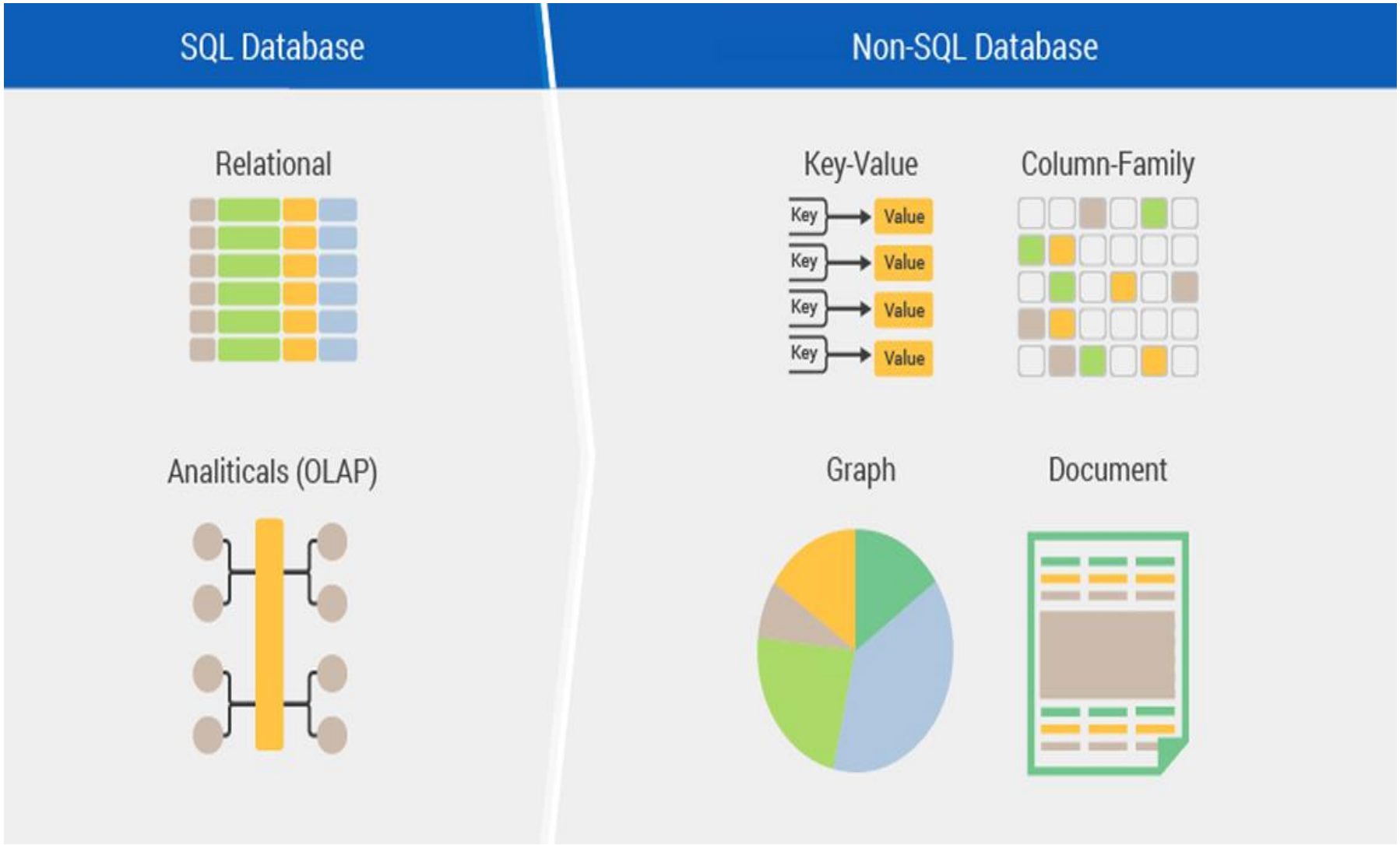
Map Reduce


HDFS Split
HDFS Split
HDFS Split
Input (key, value) pairs
Input (key, value) pairs
Input (key, value) pairs
Map
Map
Map
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
Shuffle & Sort: Aggregate values by key
(Key1, intermediate values)
(Keyk, intermediate values)
Reduce
Reduce
(Final Key1, final value)
(Final Keyk, final value)

NoSQL

- While the traditional SQL can be effectively used to handle large amount of structured data, we need NoSQL (Not Only SQL) to handle unstructured data.
- NoSQL databases store unstructured data with no particular schema
- Each row can have its own set of column values. NoSQL gives better performance in storing massive amount of data.

NoSQL


SQL Database
Non-SQL Database
Relational
Key-Value
Key
Value
Key
Value
Key
Value
Key
Value
Column-Family
Analiticals (OLAP)
Graph
Document

Hive

- This is a distributed data management for Hadoop.
- This supports SQL-like query option HiveSQL (HSQL) to access big data.
- This can be primarily used for Data mining purpose. This runs on top of Hadoop.

Apache Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.

- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.

- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation

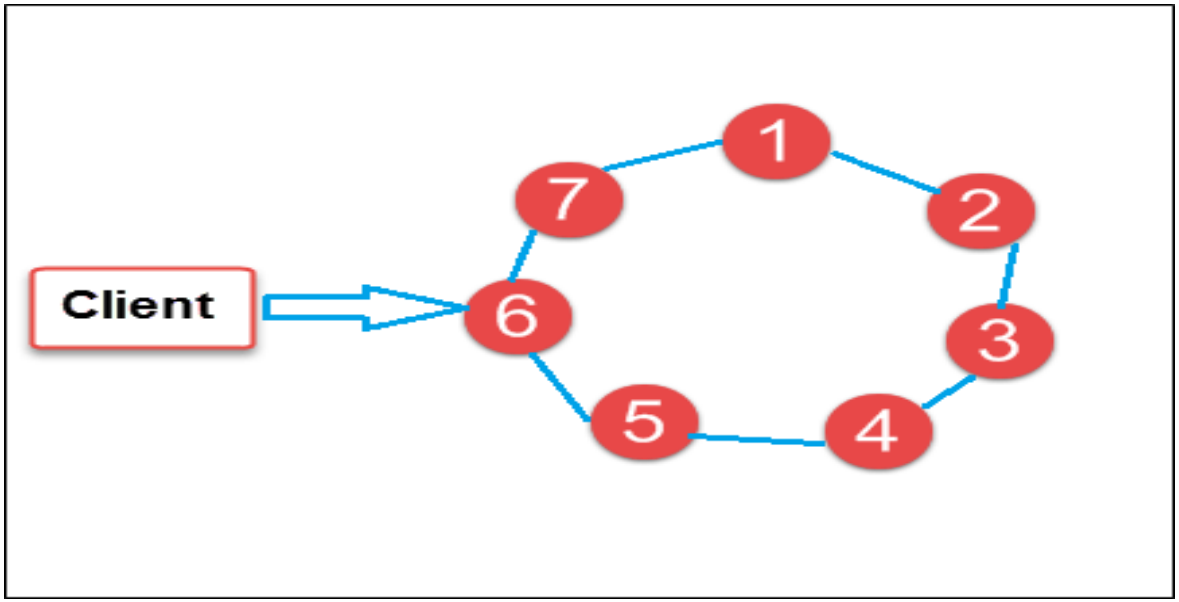
Cassandra

- Apache Cassandra is highly scalable, distributed and high-performance NoSQL database. Cassandra is designed to handle a huge amount of data.

- Cassandra handles the huge amount of data with its distributed architecture.

- Data is placed on different machines with more than one replication factor that provides high availability and no single point of failure.

Cassandra

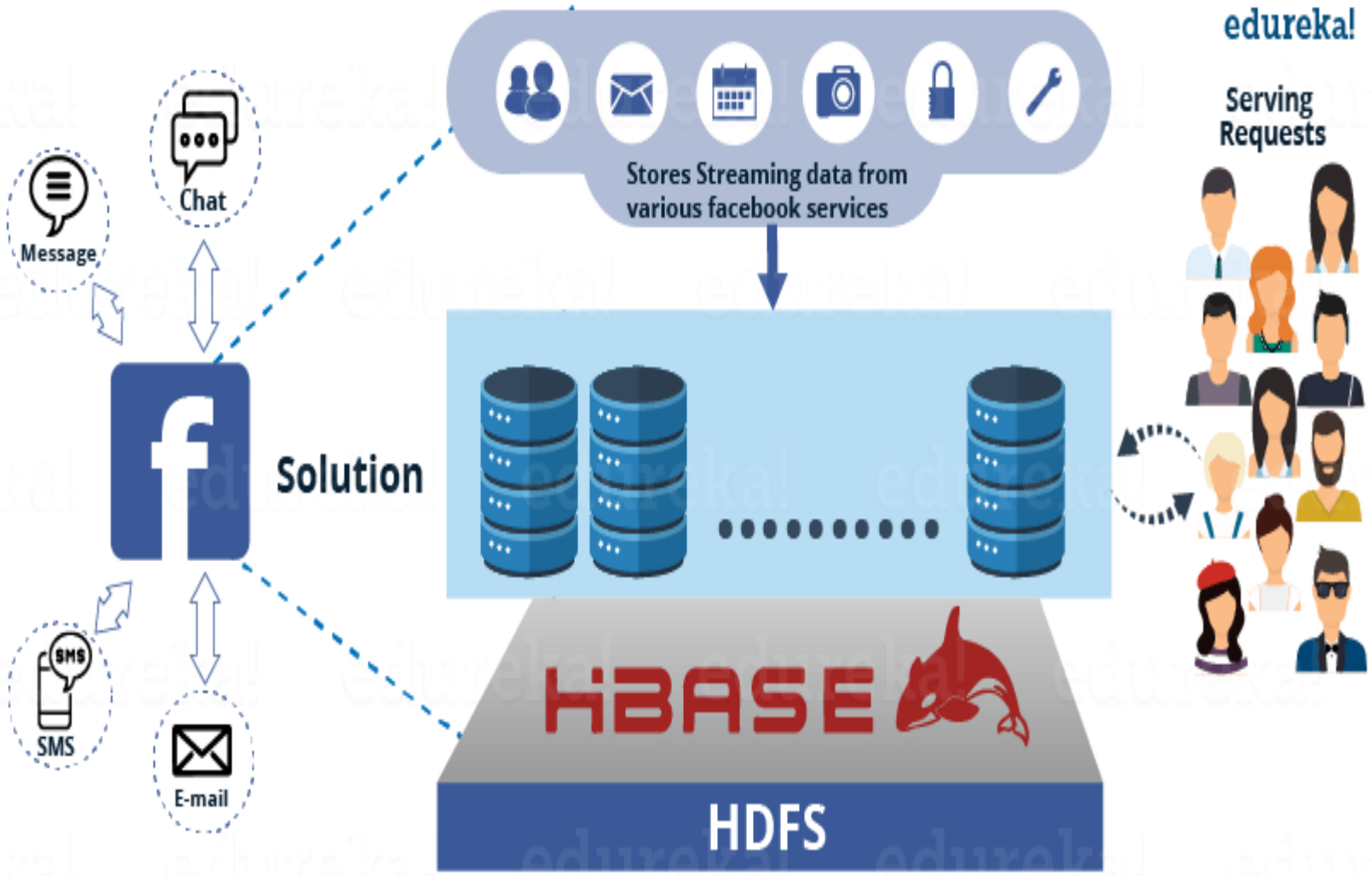


In the image above, circles are Cassandra nodes and lines between the circles shows distributed architecture, while the client is sending data to the node

HBase

- HBase is an open source, distributed database, developed by Apache Software foundation.
- Initially, it was Google Big Table, afterwards it was re-named as HBase and is primarily written in Java.
- HBase can store massive amounts of data from terabytes to petabytes.

HBase


edureka!
Stores Streaming data from various facebook services
Serving Requests
Chat
Message
Solution
SMS
E-mail
HBASE
HDFS

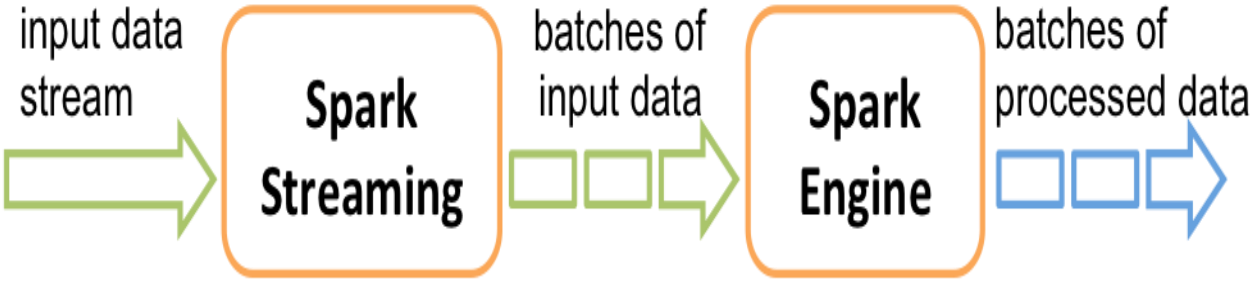
Spark Streaming

- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

- Streaming data input from HDFS, Kafka, Flume, TCP sockets, Kinesis, etc.

- Spark ML (Machine Learning) functions and GraphX graph processing algorithms are fully applicable to streaming data .

Spark Streaming



Spark MLlib

- Spark MLlib is a distributed machine-learning framework on top of Spark Core.

- MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction.

Spark Mllib Component

Algorithms
- Classification
- Regression
- Clustering
- Collaborative Filtering

Pipeline
- Constructing
- Evaluating
- Tuning
- Persistence

Featurization
- Extraction
- Transformation

Utilities
- Linear algebra
- Statistics

Spark GraphX

- GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new graph abstraction.

- GraphX reuses Spark RDD concept, simplifies graph analytics tasks, provides the ability to make operations on a directed multigraph with properties attached to each vertex and edge.

Spark GraphX

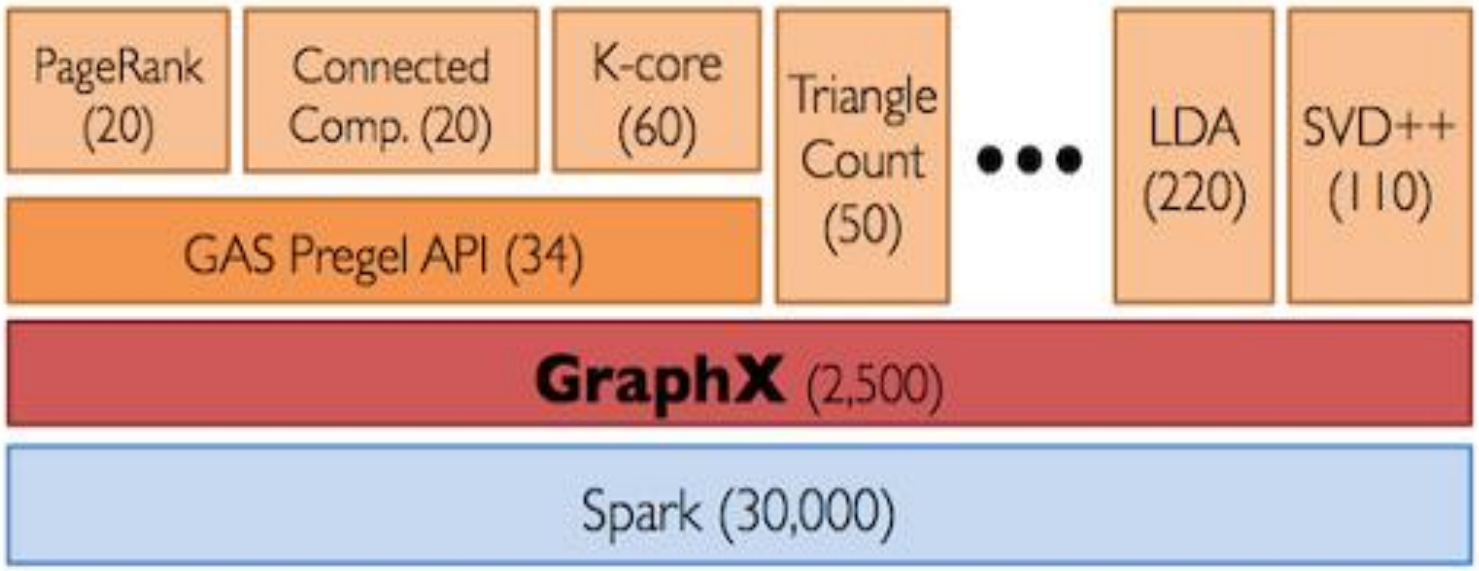


Figure 1: **GraphX** is a thin layer on top of the Spark general-purpose dataflow framework (lines of code).

Hadoop HDFS

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Hadoop HDFS

- Hadoop distributed File System (based on Google File System (GFS) paper, 2004)
  - Serves as the distributed file system for most tools in the Hadoop ecosystem
  - Scalability for large data sets
  - Reliability to cope with hardware failures

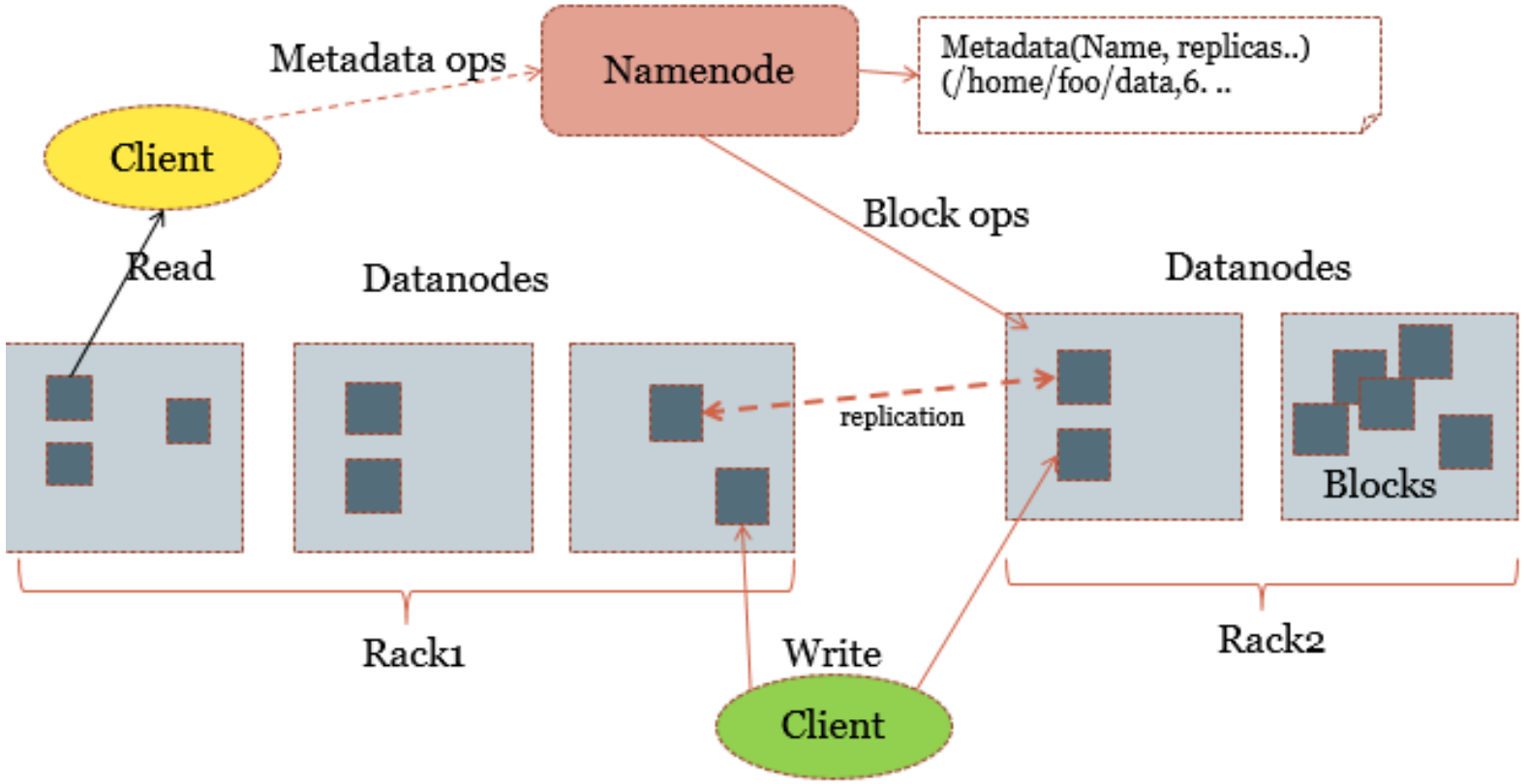
- HDFS good for:
  - Large files
  - Streaming data
- Not good for:
  - Lots of small files
  - Random access to files
  - Low latency access

Design of Hadoop Distributed title System(HDFS)

- Master-Slave design
- Master Node
  - Single NameNode for managing metadata
- Slave Nodes
  - Multiple DataNodes for storing data
- Other
  - Secondary NameNode as a backup

HDFS Architecture

NameNode keeps the metadata, the name, location and directory . **DataNode** provide storage for blocks of data



File system Namespace

- Hierarchical file system with directories and files
- Create, remove, move, rename etc.
- Namenode maintains the file system
- Any meta information changes to the file system recorded by the Namenode.
- An application can specify the number of replicas of the file needed: replication factor of the file. This information is stored in the Namenode.

Namenode

- Master
- Manages filesystem namespace
- Maintains filesystem tree and metadata-persistently on two files-namespace image and editlog
- Stores locations of blocks-but not persistently
- Metadata – inode data and the list of blocks of each file

Datanodes

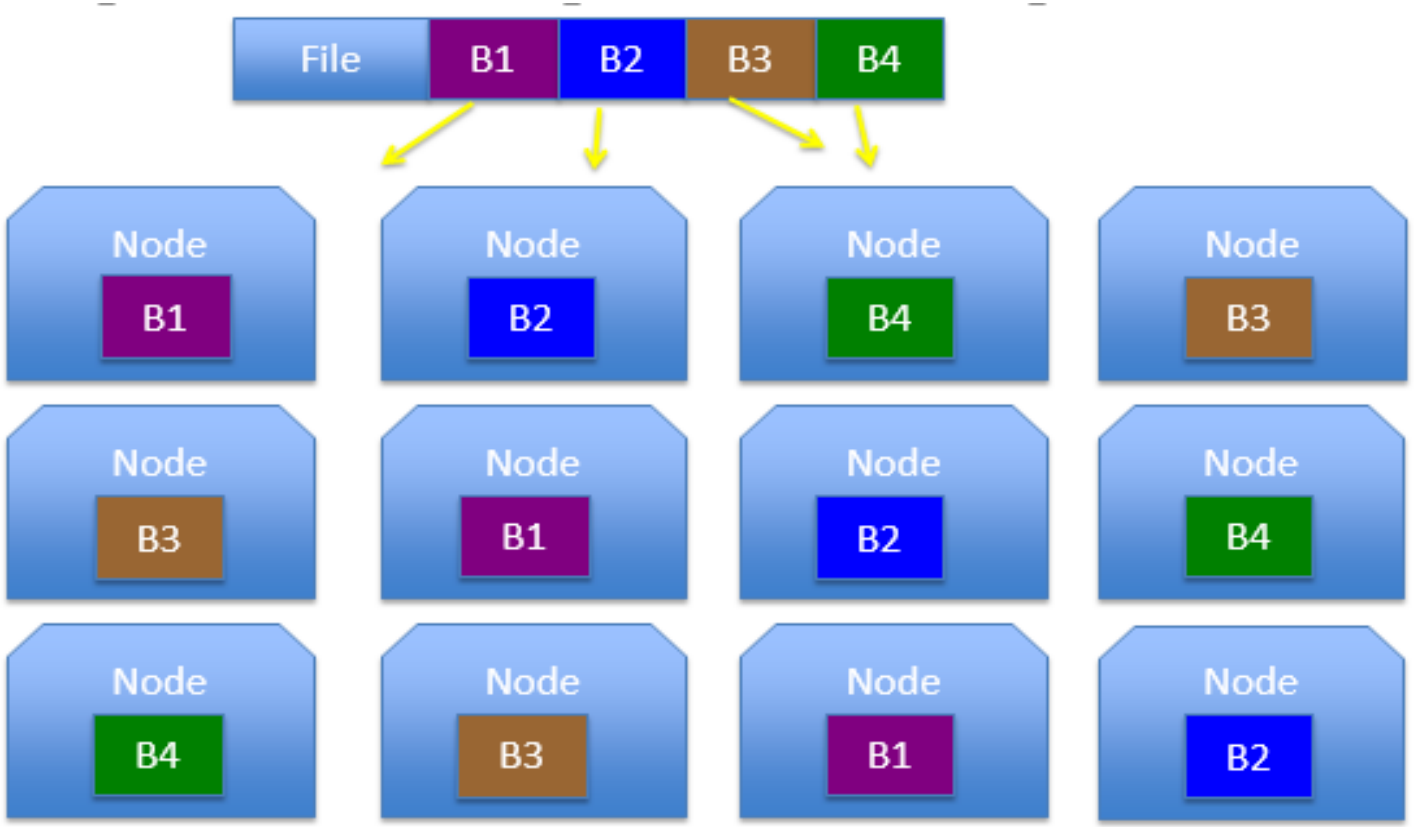
- Workhorses of the filesystem
- Store and retrieve blocks
- Send blockreports to Namenode
- Do not use data protection mechanisms like RAID…use replication
- Startup-handshake:
  - Namespace ID
  - Software version

After handshake:

- Registration
- Storage ID
- Block Report
- Heartbeats

if node(s) fail?

Replication of Blocks for fault tolerance



Secondary Namenode

- If namenode fails, the filesystem cannot be used
- Two ways to make it resilient to failure:
    - Backup of files
    - Secondary Namenode

- Periodically merge namespace image with editlog
- Runs on separate physical machine

Secondary Namenode

- Has a copy of metadata, which can be used to reconstruct state of the namenode
- Disadvantage: state lags that of the primary namenode
- Renamed as CheckpointNode (CN) in 0.21 release[1]
- Periodic and is not continuous
- If the NameNode dies, it does not take over the responsibilities of the NN

MapReduce

Dr. Rajiv Misra

Associate Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

Introduction

- **MapReduce** is a programming model and an associated implementation for **processing and generating large data sets.**

- Users specify a ***map*** function that processes a key/value pair to generate a set of intermediate key/value pairs, and a ***reduce*** function that merges all intermediate values associated with the same intermediate key.

- Many real world tasks are expressible in this model.

Contd…

- Programs written in this functional style **are automatically parallelized and executed** on a large cluster of commodity machines.
- The **run-time system** takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication.
- This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.
- A **typical MapReduce computation processes** many terabytes of data on thousands of machines. Hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

Distributed File System

Chunk Servers

- File is split into contiguous chunks
- Typically each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

Master node

- Also known as Name Nodes in HDFS
- Stores metadata
- Might be replicated

Client library for file access

- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

Motivation for Map Reduce (Why)

- **Large-Scale Data Processing**
  - Want to use 1000s of CPUs
  - But don’t want hassle of managing things

- **MapReduce Architecture provides**
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

What is MapReduce?

- Terms are borrowed from Functional Language (e.g., Lisp)

Sum of squares:

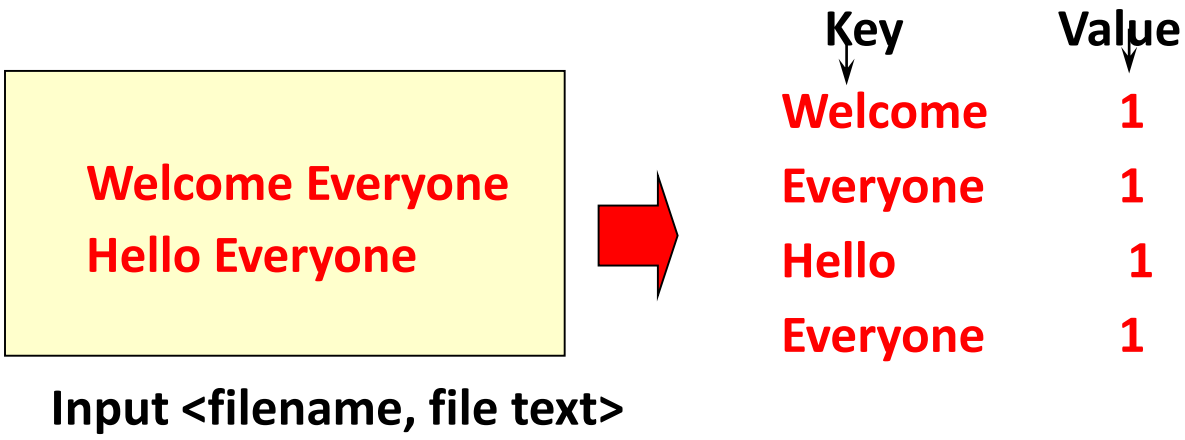
- **(map square '(1 2 3 4))**
  - **Output: (1 4 9 16)**

  [processes each record sequentially and independently]
- **(reduce + '(1 4 9 16))**
  - **(+ 16 (+ 9 (+ 4 1)))**
  - **Output: 30**

  [processes set of all records in batches]
- Let's consider a sample application: **Wordcount**
  - You are given a **huge** dataset (e.g., Wikipedia dump or all of Shakespeare's works) and asked to list the count for each of the words in each of the documents therein

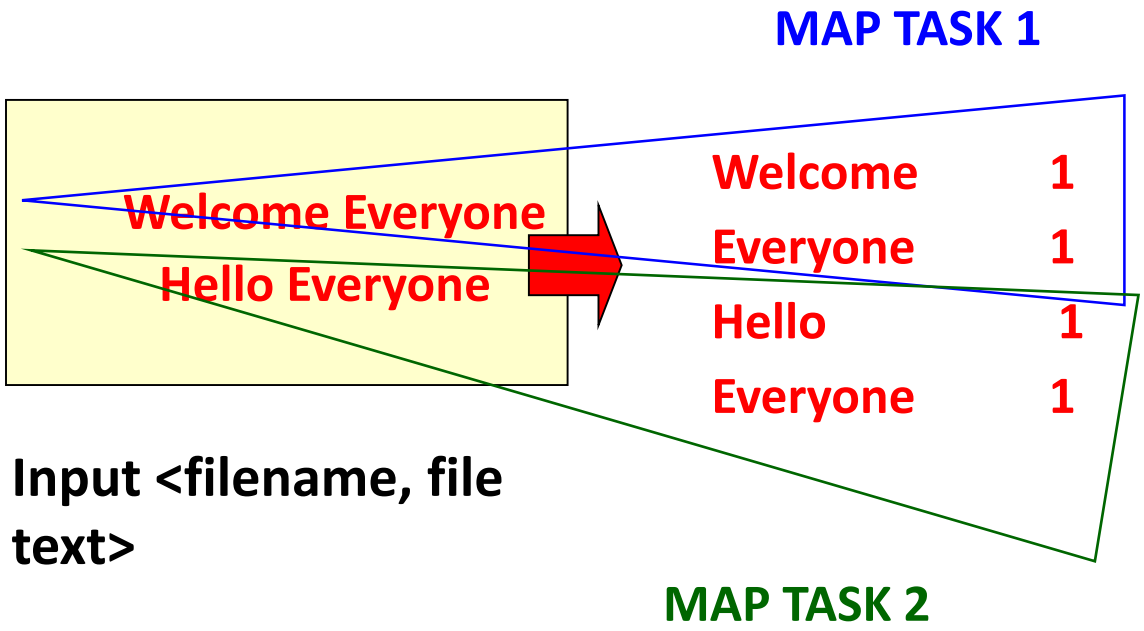
Map

- Process individual records to generate intermediate key/value pairs.



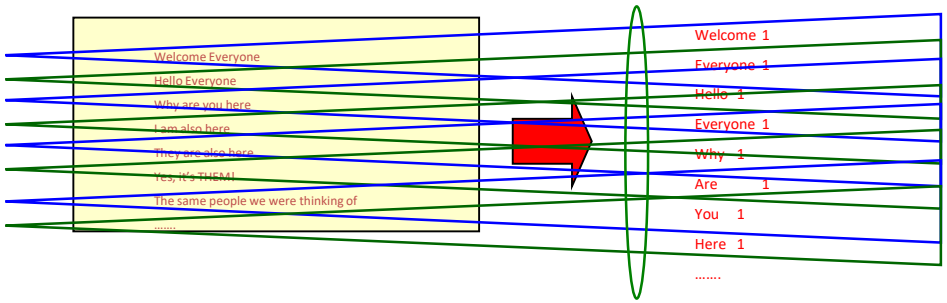
Map

- **Parallelly** Process individual records to generate intermediate key/value pairs.



Map

- **Parallelly** Process **a large number** of individual records to generate intermediate key/value pairs.

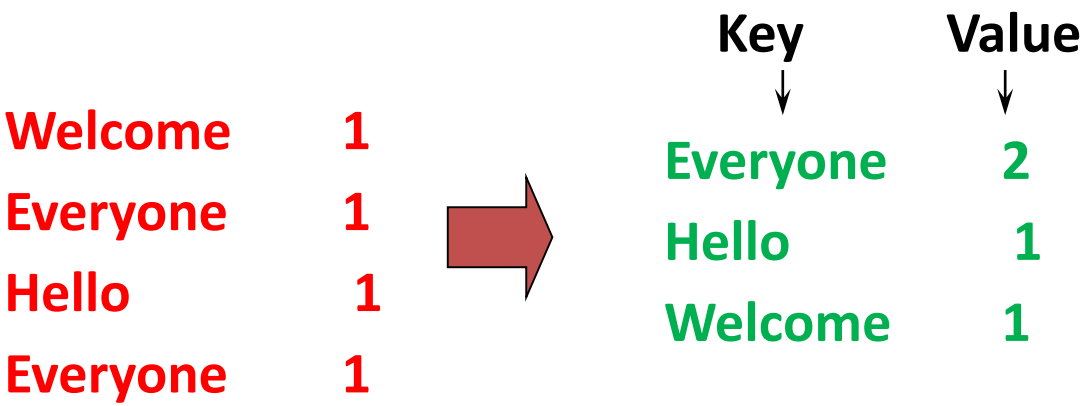


Input <filename, file text>

MAP TASKS

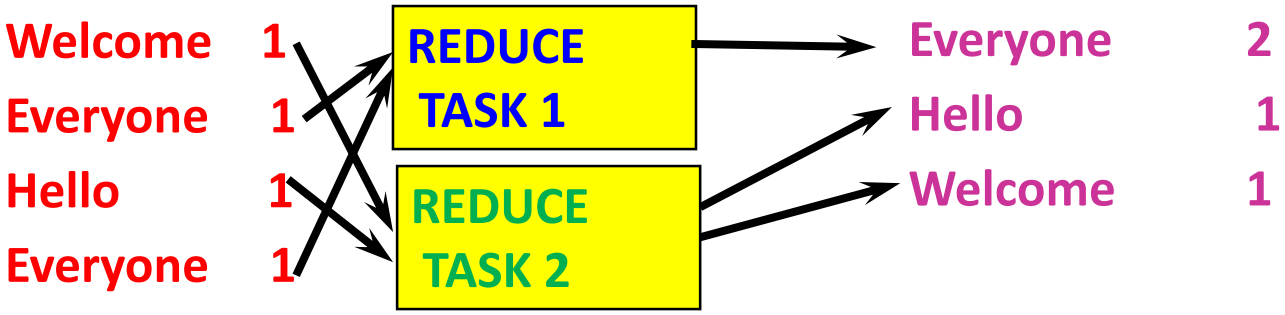
Reduce

- Reduce processes and merges all intermediate values associated per **key**



Reduce

- Each key assigned to one Reduce
- Parallelly Processes and merges all intermediate **values by partitioning keys**



- Popular: ***Hash partitioning***, *i.e.,* key is assigned to
  - reduce # = hash(key)%number of reduce tasks

Programming Model

- The computation takes a set of **input key/value pairs**, and produces a set of **output key/value pairs.**

- The user of the Map Reduce library expresses the computation as two functions:

(i) The Map

(ii) The Reduce

(i) Map Abstraction

- Map, written by the user, takes an input pair and produces a set of **intermediate key/value pairs.**

- The MapReduce library groups together all intermediate values associated with the same **intermediate key *'I'*** and passes them to the **Reduce function.**

(ii) Reduce Abstraction

- **The Reduce function,** also written by the user, accepts an **intermediate key *'I'*** and a set of values for that key.

- It merges together these values to form a **possibly smaller set of values.**

- Typically just **zero or one output value** is produced per Reduce invocation. The **intermediate values** are supplied to the user's reduce function via **an iterator.**

- This allows us to **handle lists of values** that are too large to fit in memory.

Map-Reduce Functions for Word Count

map(key, value):

```
// key: document name; value: text of document
  for each word w in value:
        emit(w, 1)
```

reduce(key, values):

```
// key: a word; values: an iterator over counts
        result = 0
        for each count v in values:
                result += v
        emit(key, result)
```

Map-Reduce Functions

- **Input:** a set of key/value pairs
- User supplies two functions:

  map(k,v) → list(k1,v1)

  reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- **Output** is the set of (k1,v2) pairs

MapReduce Applications

Applications

- Here are a few simple applications of interesting programs that can be easily expressed as **MapReduce computations.**
- **Distributed Grep:** The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- **Count of URL Access Frequency:** The map function processes logs of web page requests and outputs (URL; 1). The reduce function adds together all values for the same URL and emits a (URL; total count) pair.
- **ReverseWeb-Link Graph:** The map function outputs (target; source) pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: (target; list(source))

Contd…

- **Term-Vector per Host:** A term vector summarizes the most important words that occur in a document or a set of documents as a list of (word; frequency) pairs.

- The map function emits a (hostname; term vector) pair for each input document (where the hostname is extracted from the URL of the document).

- The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final (hostname; term vector) pair

Contd…

- **Inverted Index:** The map function parses each document, and emits a sequence of (word; document ID) pairs. The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a (word; list(document ID)) pair. The set of all output pairs forms a simple inverted index. It is easy to augment this computation to keep track of word positions.

- **Distributed Sort:** The map function extracts the key from each record, and emits a (key; record) pair. The reduce function emits all pairs unchanged.

Conclusion

- **The MapReduce programming model has been successfully used at Google for many different purposes.**

- The model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing.

- A large variety of problems are easily expressible as MapReduce computations.

- **For example, MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems.**

Conclusion

- Mapreduce uses **parallelization + aggregation** to schedule applications across clusters
- **Need to deal with failure**
- Plenty of ongoing research work in **scheduling and fault-tolerance for Mapreduce and Hadoop.**

Big Data Storage Technologies

BIG DATA STORAGE TECHNOLOGIES

- Distributed File Systems: Example Hadoop Distributed File System. It is well suited for quickly ingesting data and bulk processing.
- NoSQL Databases: Use data models that are outside the relational world.
- NewSQL Databases: A modern form of relational databases that aim for comparable scalability as NoSQL databases
- Big Data Querying Platforms: Technologies that provide query facades in front of big data stores such as distributed file systems or NoSQL databases.

RDBMS TRANSACTION – ACID RULE

- Atomic – All of the work in a transaction completes (commit) or none of it completes.
- Consistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
- Isolated – Modifications of data performed by a transaction must be independent of another transaction.
- Durable – When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

WHAT IS NOSQL?

- NoSQL is a non-relational database management systems
- Significantly different from traditional RDBMS
- It is designed for distributed data stores for very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users)
- These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

NOSQL

- Stands for Not Only SQL
- No declarative query language
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- CAP Theorem
- Prioritizes high performance, high availability and scalability

WHY NOSQL?

- Data is becoming easier to access and capture
- Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially
- It is required to process huge amount of data for which SQL databases were never designed
- NoSQL aims to handle these huge data properly

WHEN TO USE NOSQL

- Big amount of data
- Lots of reads/writes
- Economic
- Flexible schema
- No transactions needed
- ACID is not important
- No joins

NOSQL: PROS/CONS

- Advantages:
  - High scalability
  - Distributed Computing
  - Lower cost
  - Schema flexibility, semi-structure data
  - No complicated Relationships
- Disadvantages
  - No standardization
  - Limited query capabilities (so far)
  - Eventual consistent is not intuitive to program for

THE BASE

- Almost the opposite of ACID.
- The BASE acronym was defined by Eric Brewer, who is also known for formulating the CAP theorem.
- A BASE system gives up on consistency.
  - Basically Available indicates that the system *does* guarantee availability, in terms of the CAP theorem.
  - Soft state indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.
  - Eventual consistency indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

NOSQL DATABASE TYPES

- Document databases pair each key with a complex data structure known as a document. Eg. FirstName = “Rashmi”, LastName = “Taneja”, Address = “IIT Patna”, Spouse = [{Name: “Manoj”, Age: 30}]
- Graph stores are used to store information about networks, such as social connections
- In Key-value-store category of NoSQL database, a user can store data in schema-less way. A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
- Wide column stores are optimized for queries over large datasets and store columns of data together instead of rows.

NOSQL CATEGORIES

- Four general types (most common categories) of NoSQL databases:
  - Key-value stores
  - Column-oriented
  - Graph
  - Document oriented

KEY – VALUE STORES

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data.
- Based on Amazon's Dynamo paper.
- Key value stores allow developer to store schema-less data.
- In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON (JavaScript Object Notation), BLOB (basic large object) etc.
- A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
- For example a key-value pair might consist of a key like "Name" that is associated with a value like "Robin".
- Key-Value stores can be used as collections, dictionaries, associative arrays etc.
- Key-Value stores follows the 'Availability' and 'Partition' aspects of CAP theorem.
- Key-Values stores would work well for shopping cart contents, or individual values like colour schemes, a landing page URI, or a default account number.

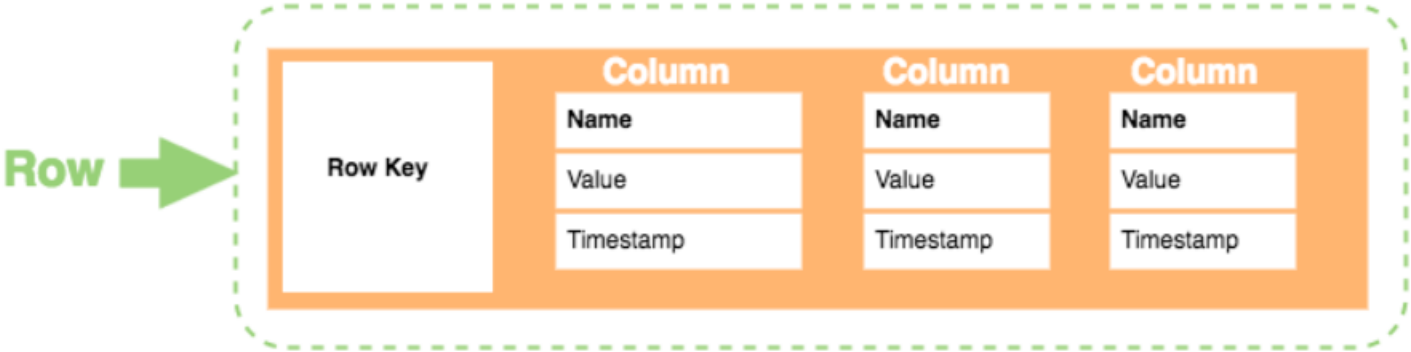
KEY – VALUE STORES

- Example: Redis, Dynamo, Riak.

COLUMN ORIENTED DATABASES

- Work on columns and every column is treated individually.
- Values of asingle column are stored contiguously.
- Column stores data in column specificfiles.
- In Column stores, query processors work on columns too.
- All data within each column have the same type which makes it ideal for compression.
- Column stores can improve the performance of queries as it can access specific columndata.
- High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).
- Works on data warehouses and business intelligence, customer relationship management (CRM), Library card catalogs etc.

COLUMN ORIENTED DATABASES



Row Key. Each row has a unique key, which is a unique identifier for that row.

Column. Each column contains a name, a value, and timestamp.

Name. This is the name of the name/value pair.

Value. This is the value of the name/value pair.

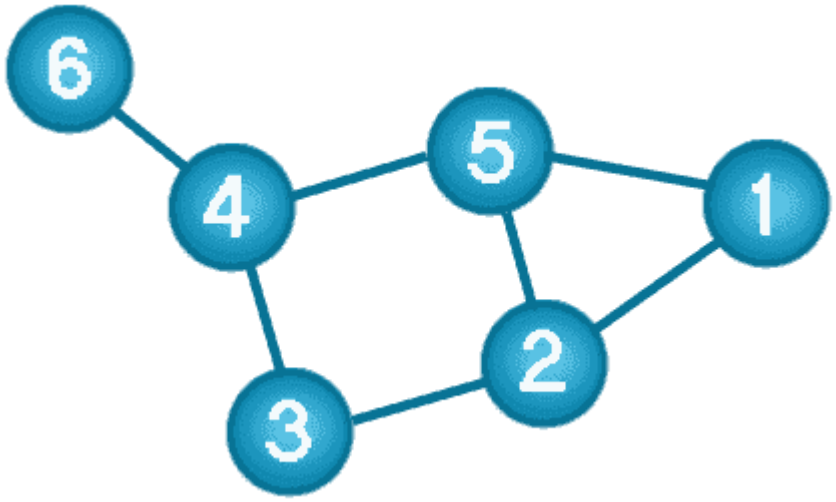
Timestamp. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

COLUMN ORIENTED DATABASES EXAMPLE

Examples: BigTable, Cassandra, SimpleDB

GRAPH DATABASES

- A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called edges or arcs, of certain entities called nodes or vertices.



GRAPH DATABASES

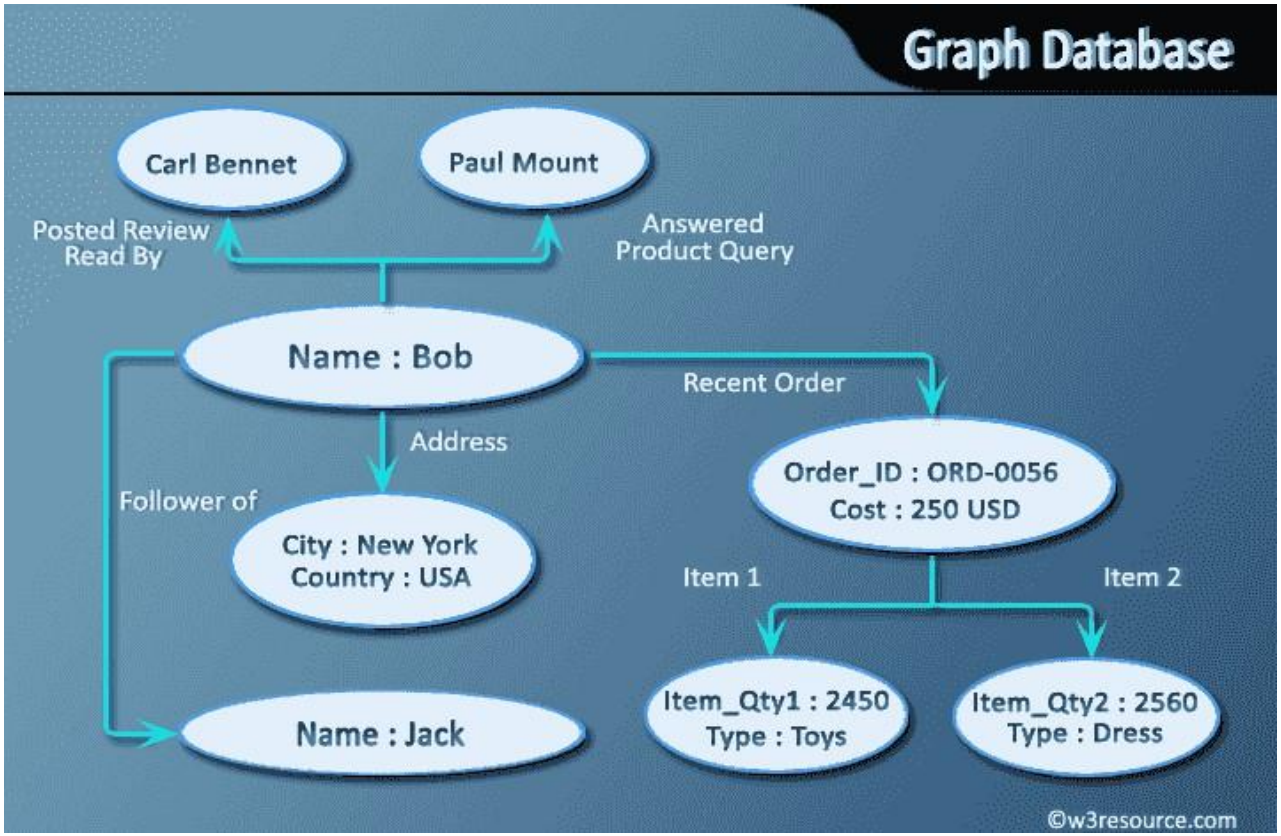
- A graph database stores data in a graph.
- It is capable of elegantly representing any kind of data in a highly accessible way.
- A graph database is a collection of nodes and edges
- Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.
- Every node and edge is defined by a unique identifier.
- Each node knows its adjacent nodes.
- As the number of nodes increases, the cost of a local step (or hop) remains the same.
- Index for lookups.

COMPARISON BETWEEN RELATIONALMODEL AND GRAPH MODEL

Relational Model	Graph Model
Tables	Vertices and Edges set
Rows	Vertices
Columns	Key/value pairs
Joins	Edges

GRAPH DATABASES EXAMPLE

Example: OrientDB, Neo4J, Titan

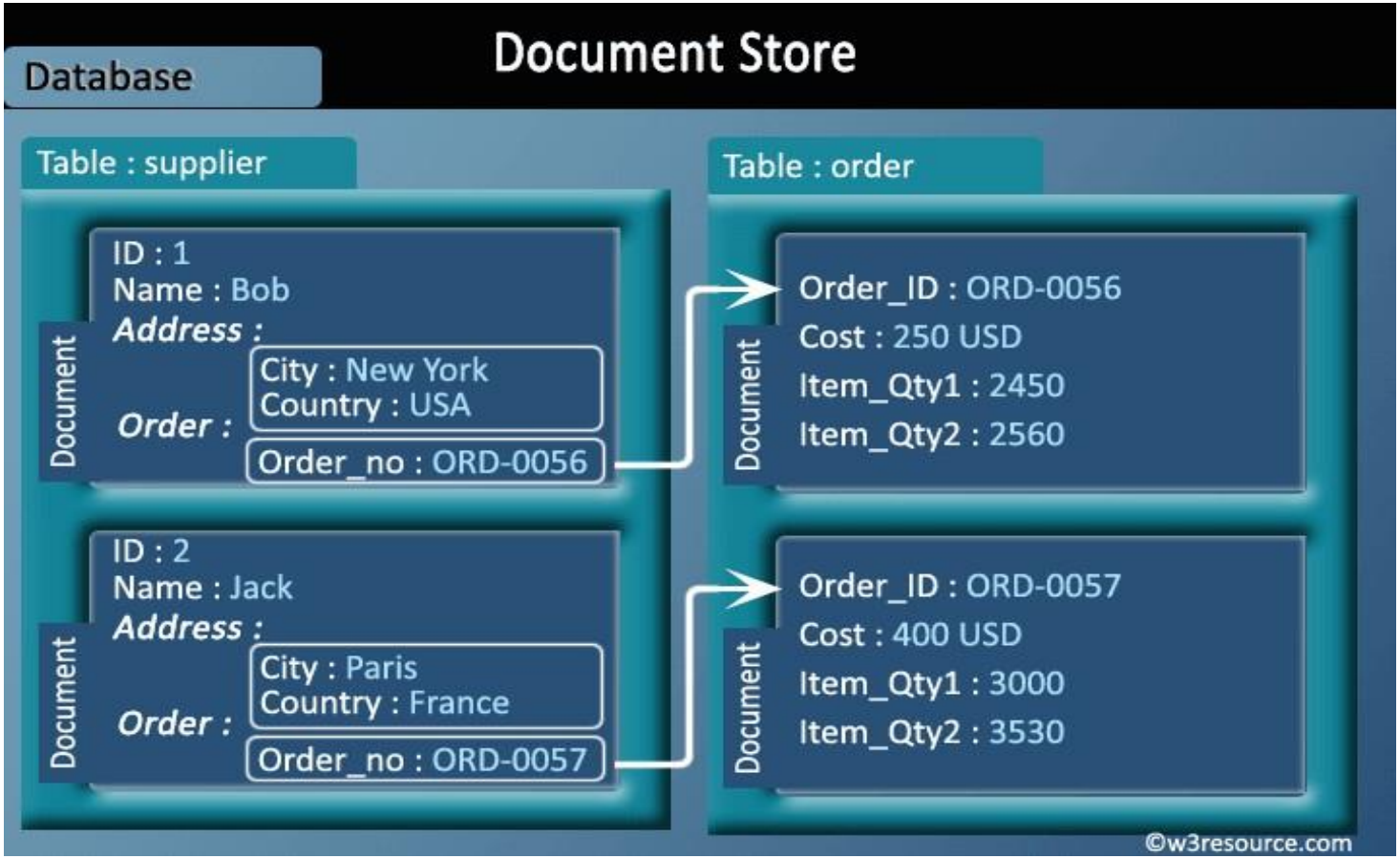


DOCUMENT ORIENTED DATABASES

- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy tochange.
- Documents are stored into collections in order togroup different kinds of data.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

DOCUMENT ORIENTED DATABASES

- Example: MongoDB, CouchDB



PRODUCTION DEPLOYMENT

- There is a large number of companies using NoSQL.
  - Google
  - Facebook
  - Mozilla
  - Adobe
  - Foursquare
  - LinkedIn
  - Digg
  - McGraw-Hill Education
  - Vermont Public Radio

NEWSQL

- Is a term coined by 451 Group analyst Matt Aslett
- Offers the best of both worlds:
  - Relational data model
  - ACID transactional consistency
  - Familiarity and interactivity of SQL
  - Scalability and speed of NoSQL
- Example: VoltDB, NuoDB, MemSQL

NEWSQL: WHAT IS NEW?

- Main memory storage: reading and writing blocks to memory cache is much faster.
  - Historically memory was much more expensive and had a limited capacity compared to disks.
  - Now the scenario is different.
  - Many NewSQL DBMSs are based on this:
    - Academic (e.g., H-Store, HyPer)
    - Commercial (e.g., MemSQL, SAP HANA, VoltDB) systems
  - NewSQL systems evict a subset of the database out to persistent storage to reduce its memory footprint.

CONCLUSION

- Big data can be operational or analytical.
- Two classes of technologies are complementary and frequently deployed together.
- Big data storage technologies have grown in the following areas:
  - DistributedFile Systems
  - NoSQL databases: complies BASE
  - NewSQL databases: complies ACID

Introduction to Spark

Dr. Rajiv Misra

Associate Professor

Dept. of Computer Science & Engg.

Indian Institute of Technology Patna

rajivm@iitp.ac.in

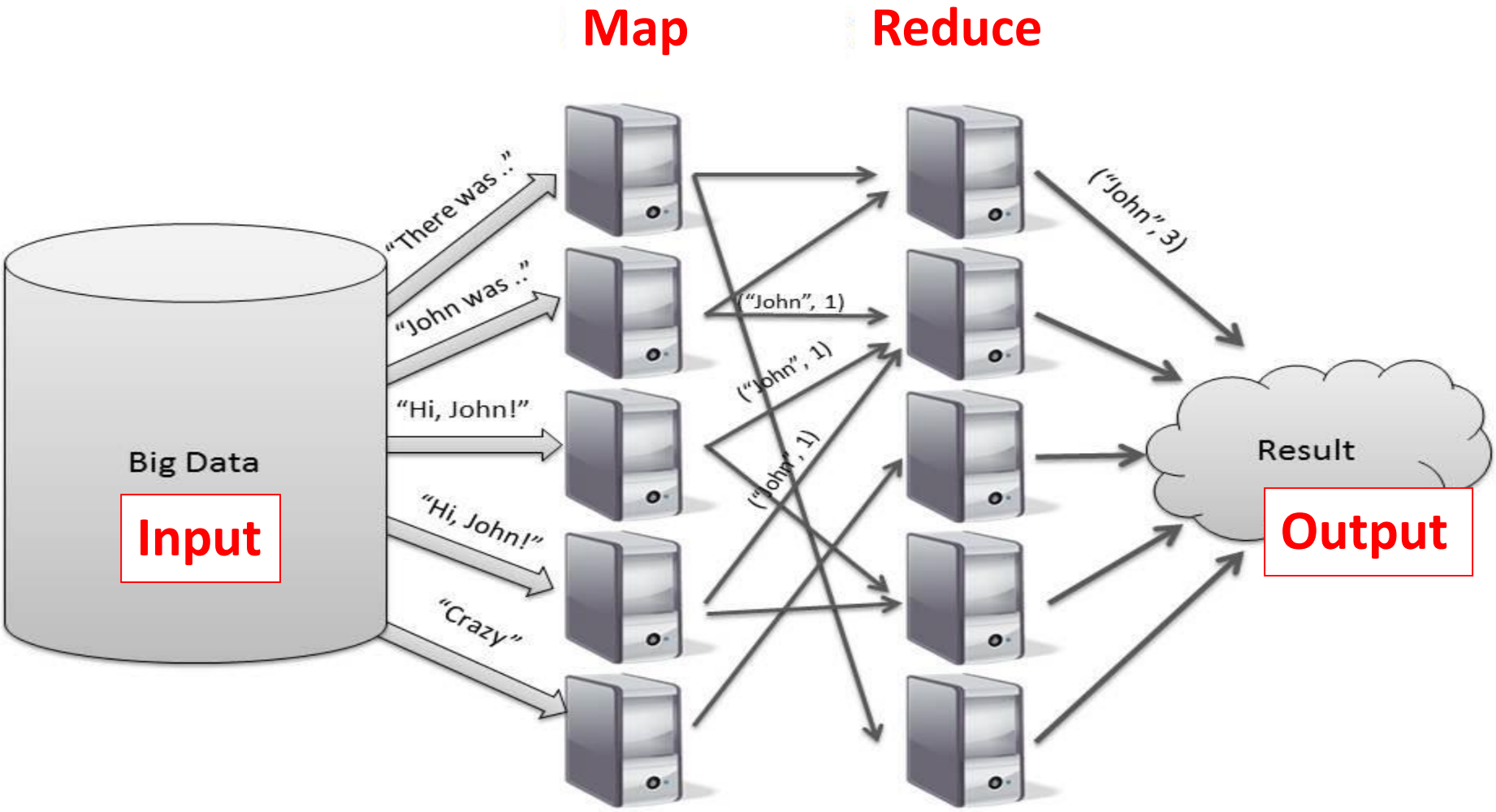
Need of Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.

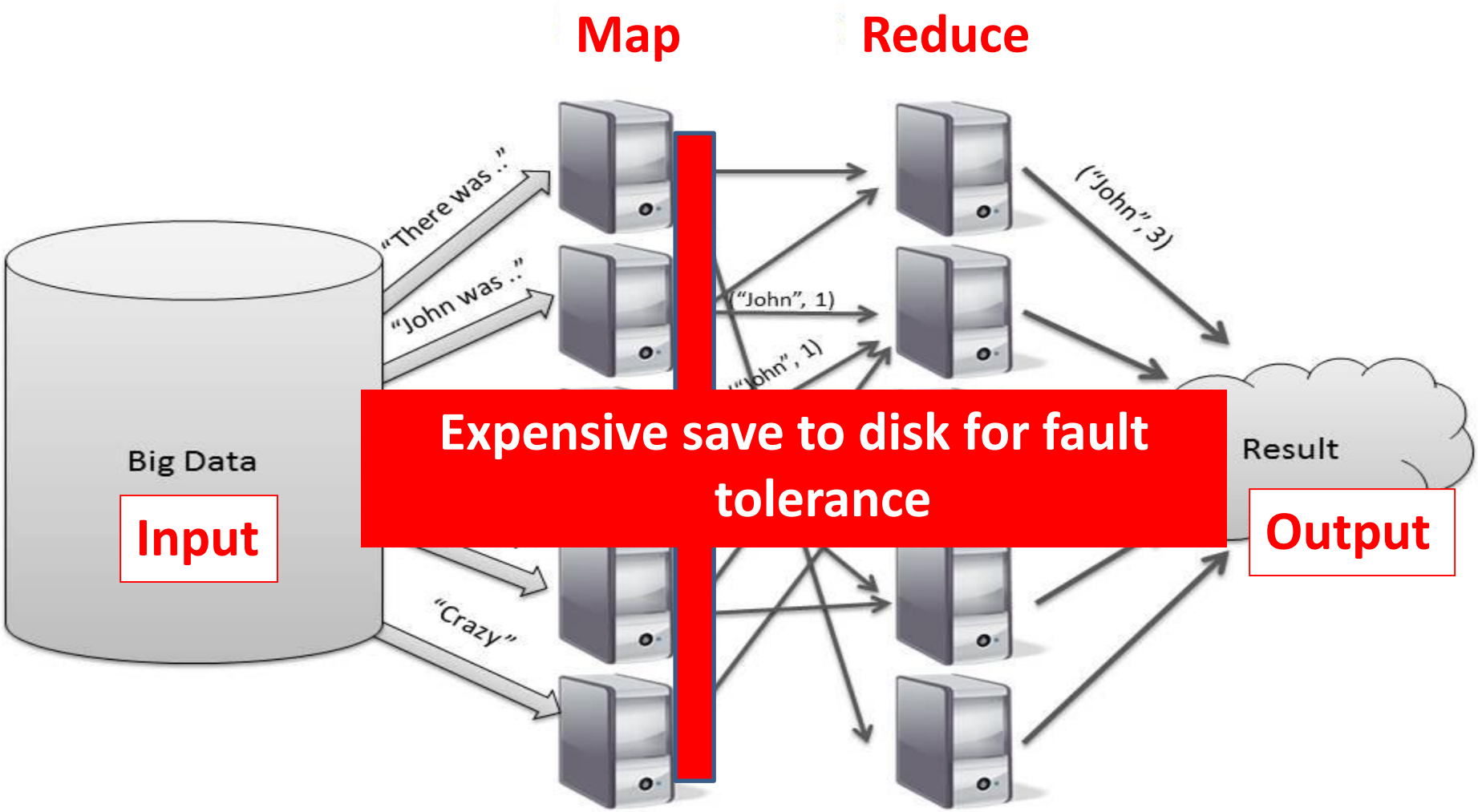
- It is an another system for big data analytics

- **Isn’t MapReduce good enough?**
  - **Simplifies batch processing on large commodity clusters**

Need of Spark



Need of Spark



Need of Spark

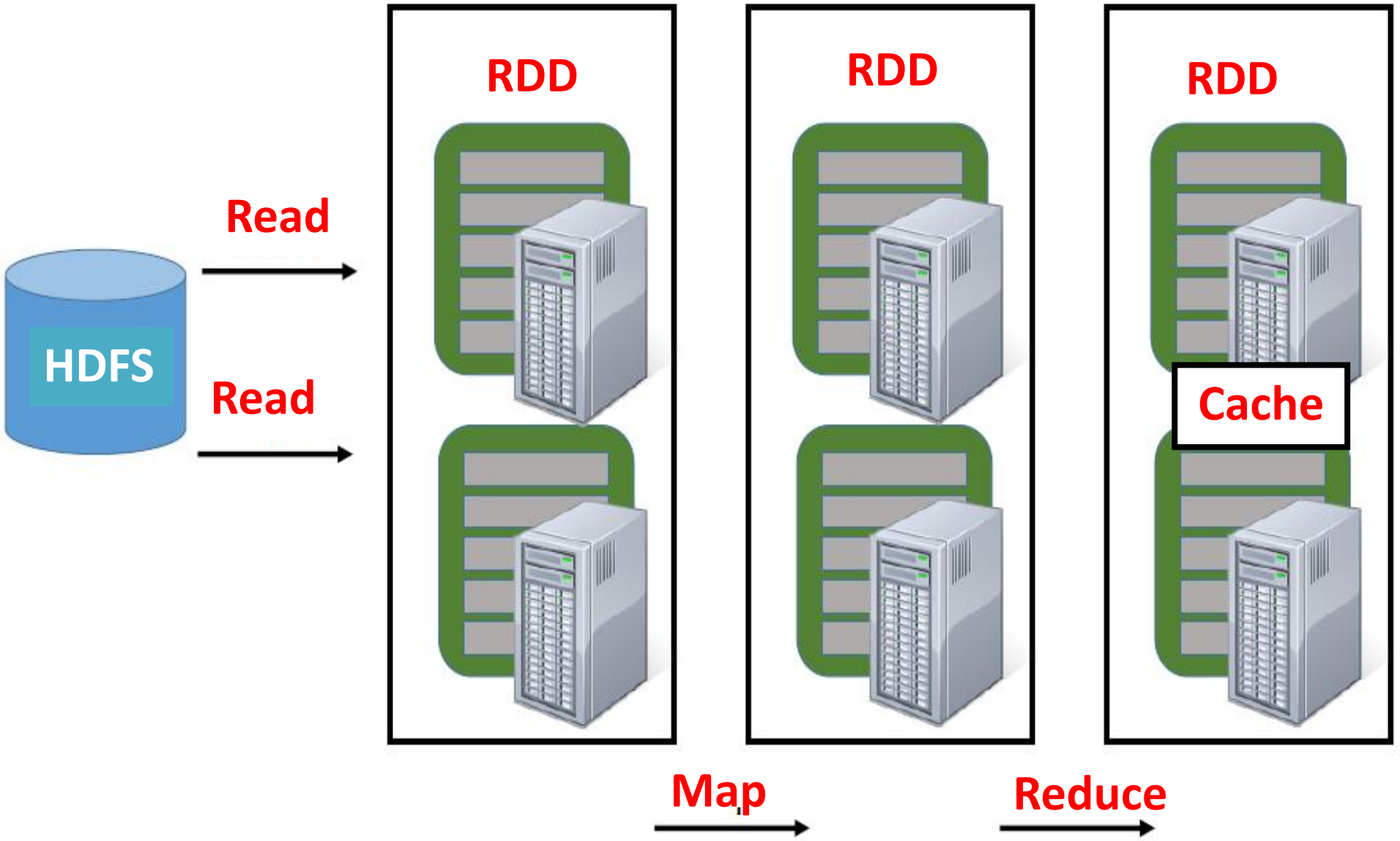
- MapReduce can be expensive for some applications e.g.,
  - **Iterative**
  - **Interactive**
- Lacks efficient data sharing
- Specialized frameworks did evolve for different programming models
  - **Bulk Synchronous Processing (Pregel)**
  - **Iterative MapReduce (Hadoop) ….**

Solution: Resilient Distributed Datasets (RDDs)

Resilient Distributed Datasets (RDDs)

- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)
- Can be cached for efficient reuse

Need of Spark



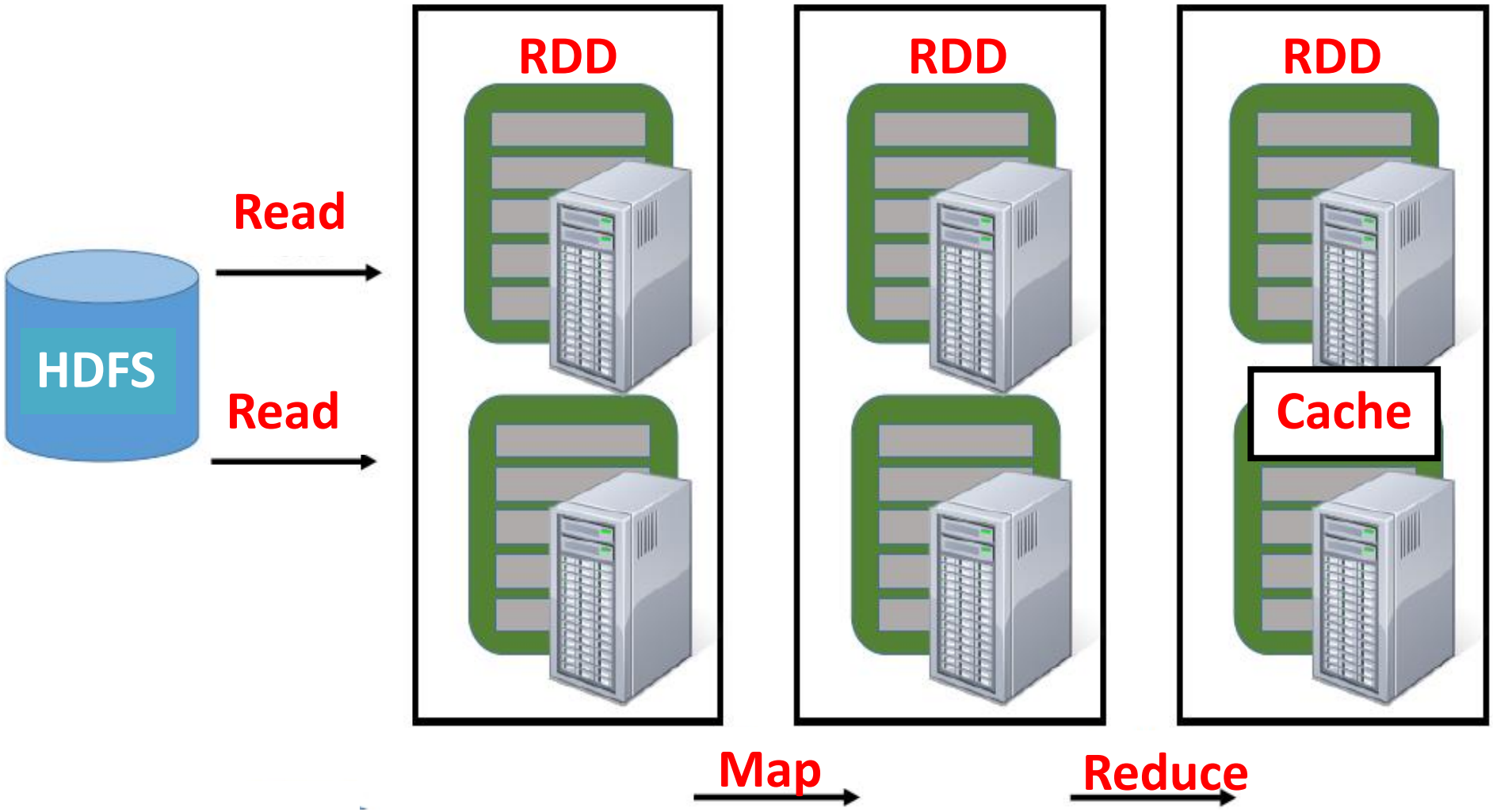
Solution: Resilient Distributed Datasets (RDDs)

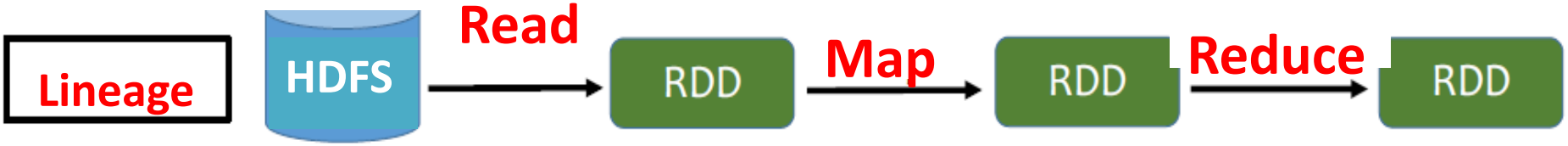
Resilient Distributed Datasets (RDDs)

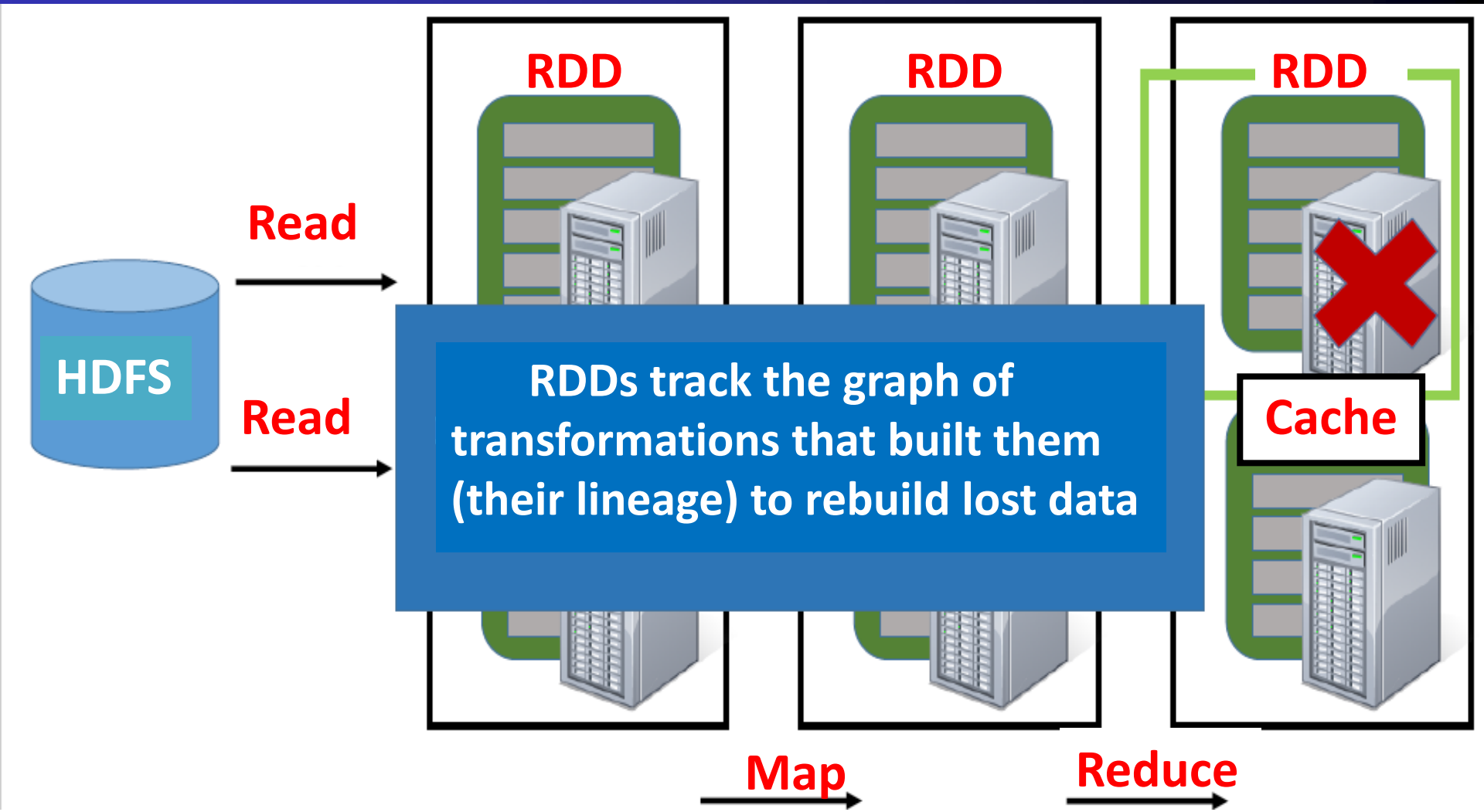
- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join …)

Fault Recovery?

- Lineage!
  - Log the coarse grained operation applied to a partitioned dataset
  - Simply recompute the lost partition if failure occurs!
  - No cost if no failure


RDD
RDD
RDD
Read
Read
HDFS
Cache
Map
Reduce

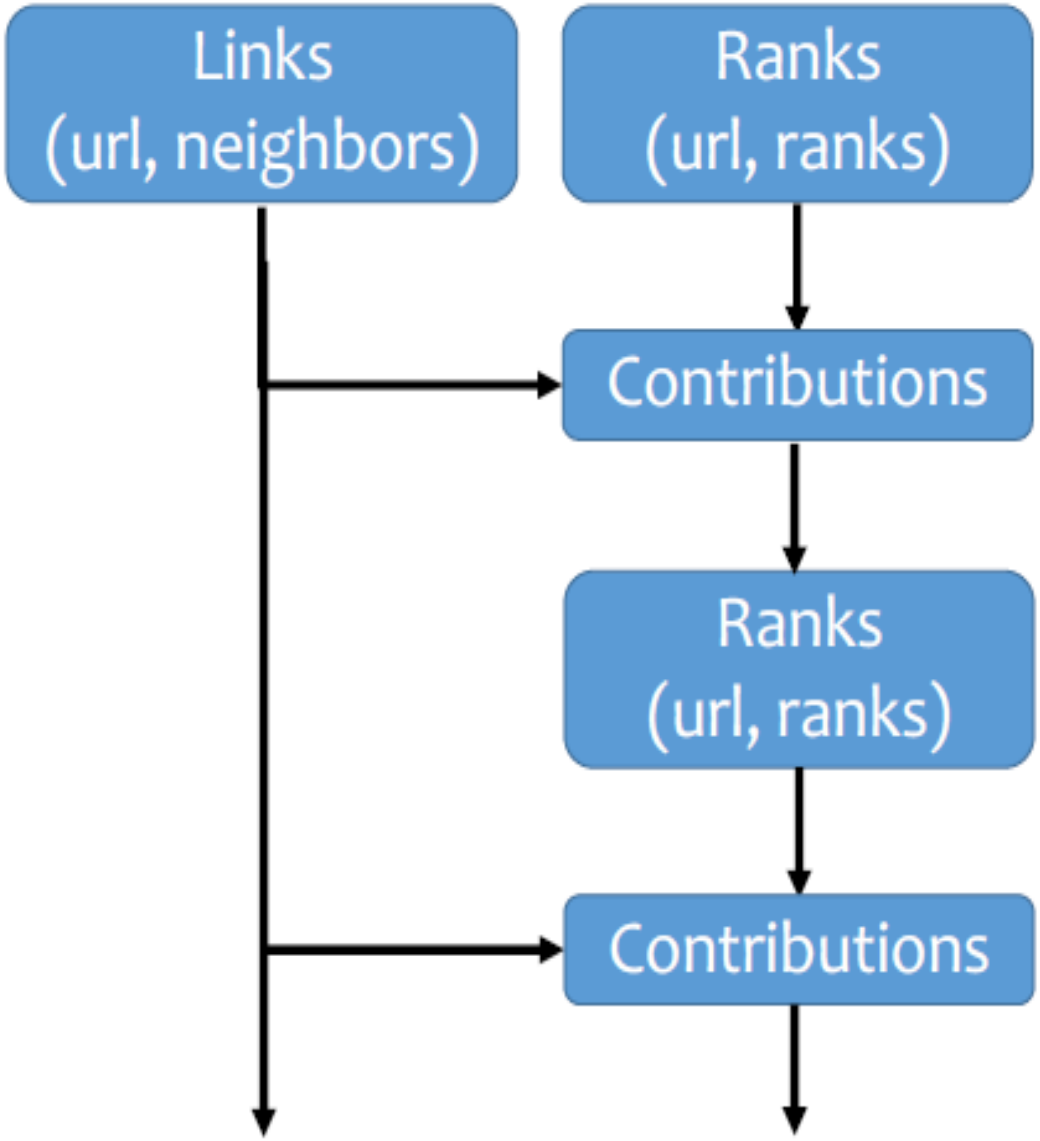

Lineage
HDFS
Read
RDD
Map
RDD
Reduce
RDD


RDD
RDD
RDD
Read
HDFS
Read
RDDs track the graph of transformations that built them (their lineage) to rebuild lost data
Cache
Map
Reduce

What can you do with Spark?

- **RDD operations**
  - Transformations e.g., filter, join, map, group-by …
  - Actions e.g., count, print …
- **Control**
  - **Partitioning:** Spark also gives you control over how you can partition your RDDs.
  - **Persistence:** Allows you to choose whether you want to persist RDD onto disk or not.

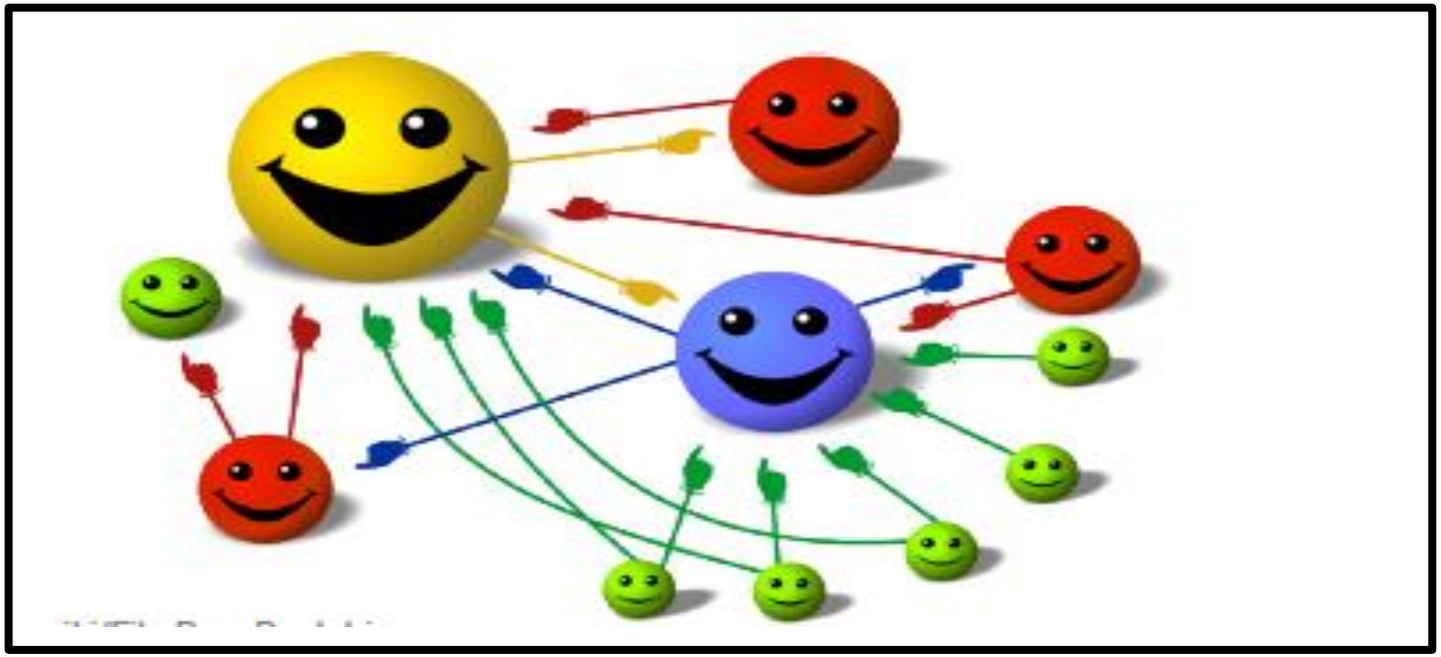
Partitioning: PageRank



- Joins take place repeatedly
- Good partitioning reduces shuffles

Example: PageRank

- Give pages ranks (scores) based on links to them
- Links from many pages → high rank
- Links from a high-rank page → high rank

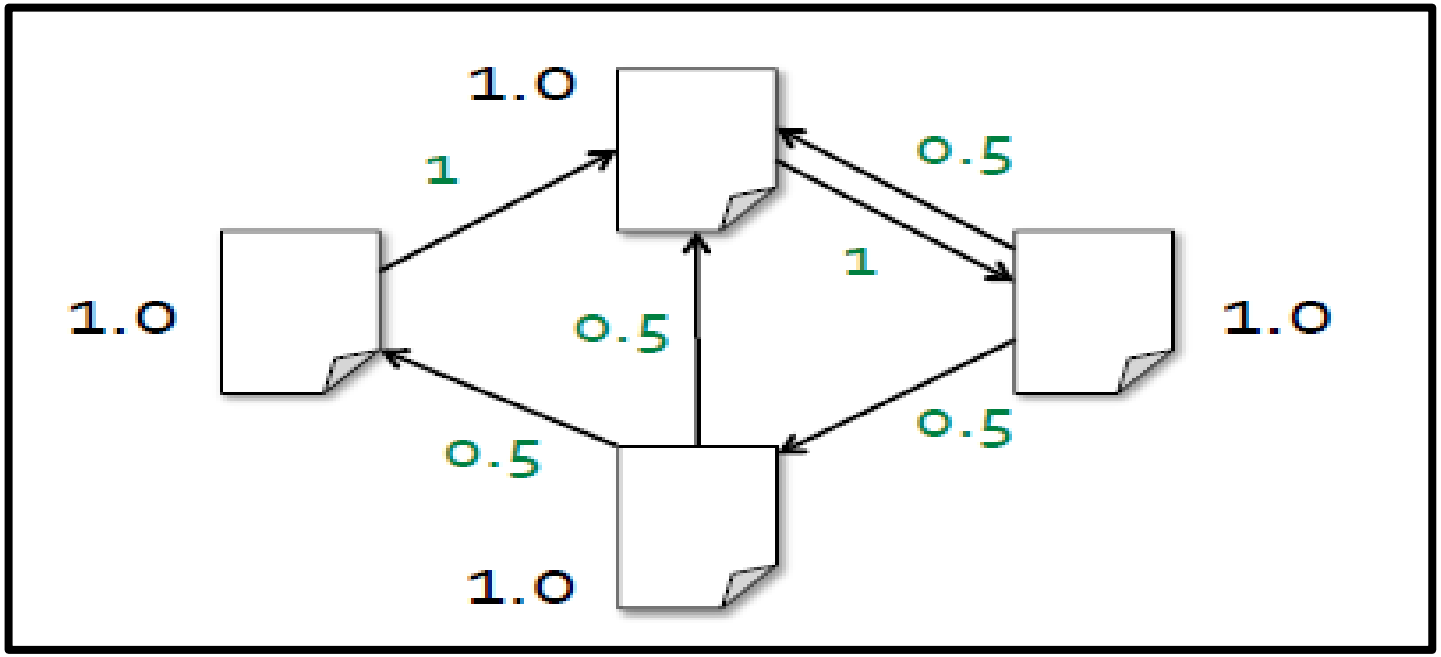


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p / |neighbors_p|$** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

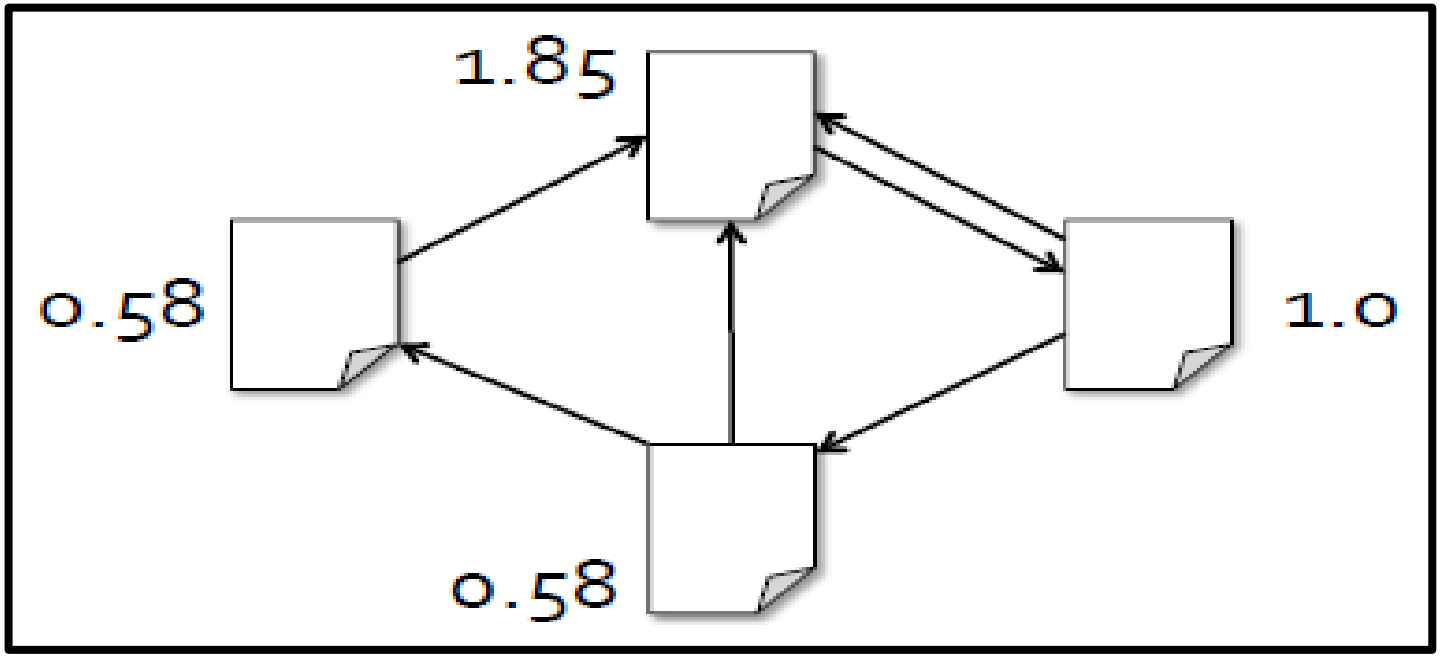


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p / |neighbors_p|$** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

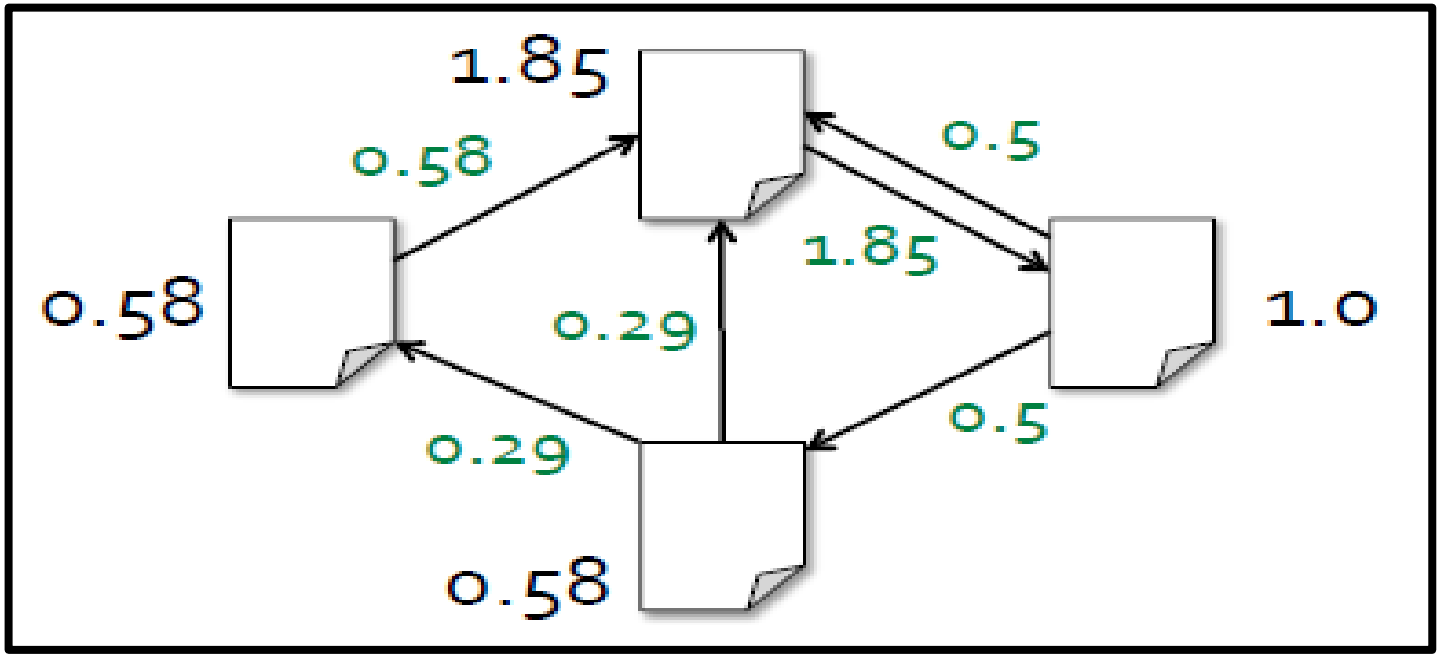


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p / |neighbors_p|$** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

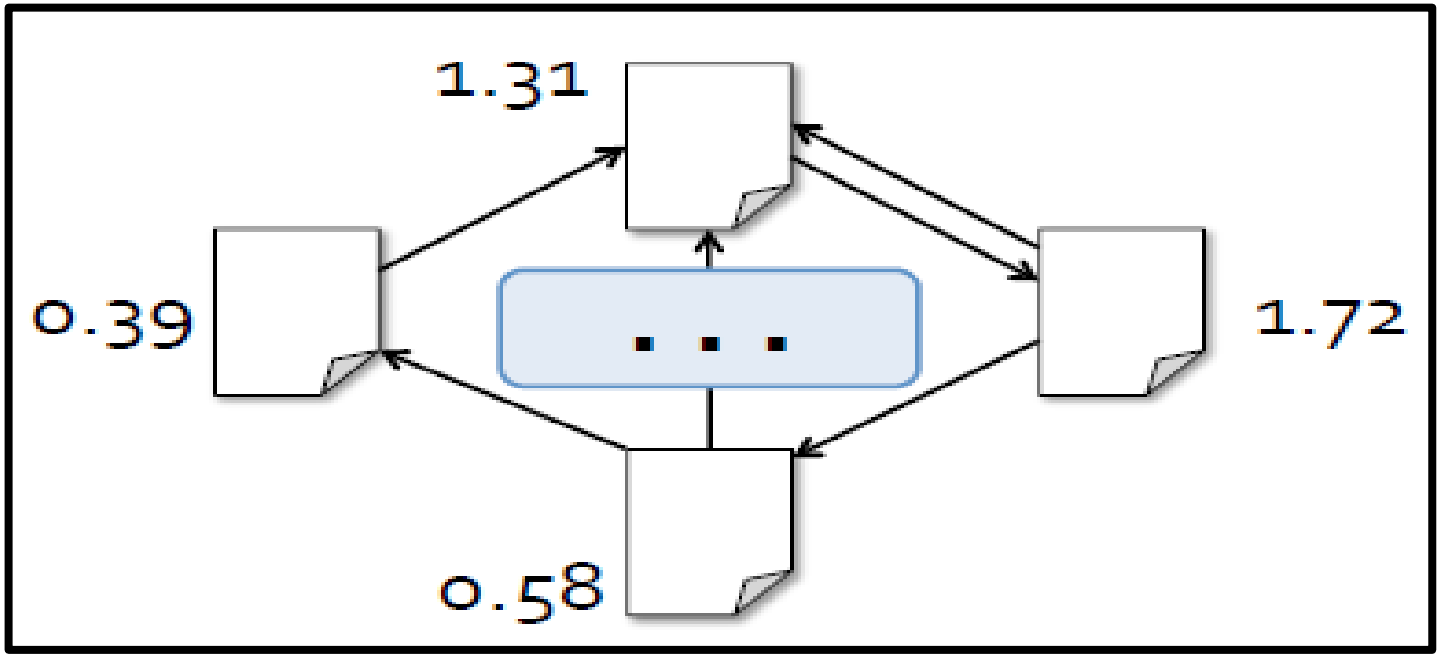


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p / |neighbors_p|$** to its neighbors

Step-3 Set each page's rank to **0.15 + 0.85 x contributions**

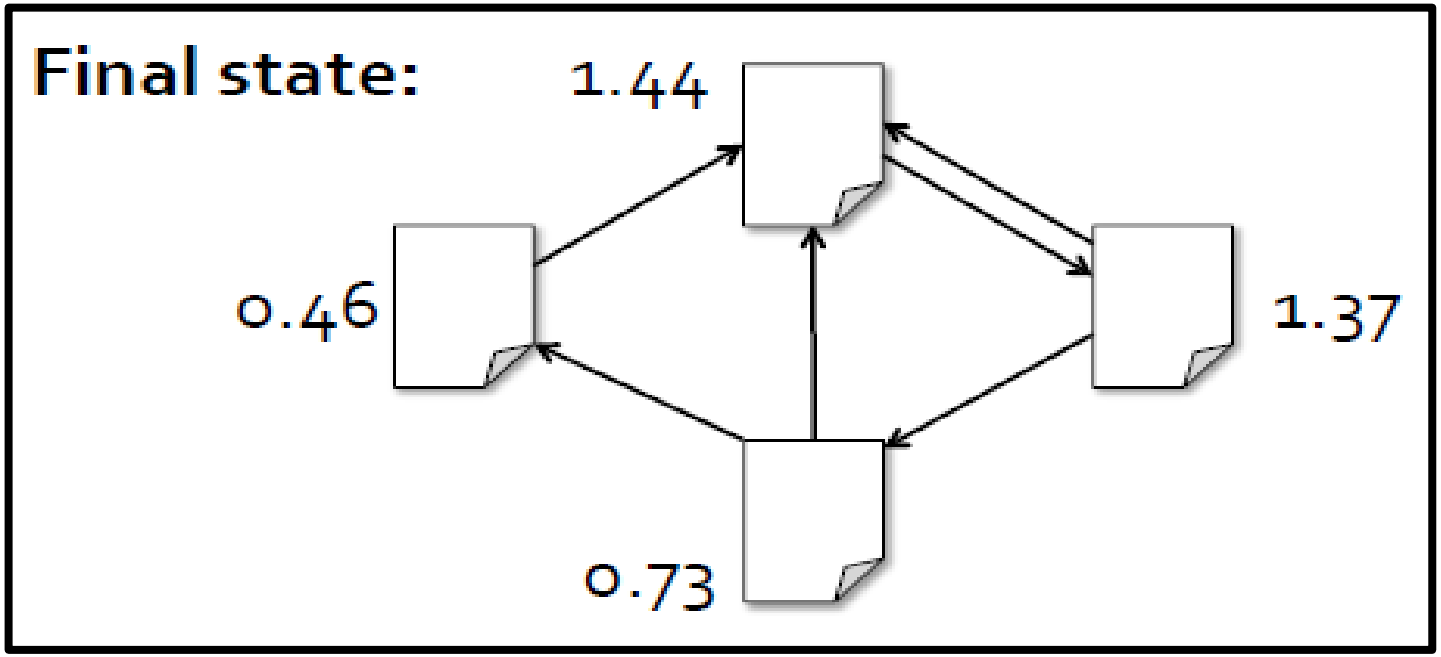


Algorithm

Step-1 Start each page at a rank of 1

Step-2 On each iteration, have page p contribute **$rank_p$/ $|neighbors_p|$** to its neighbors

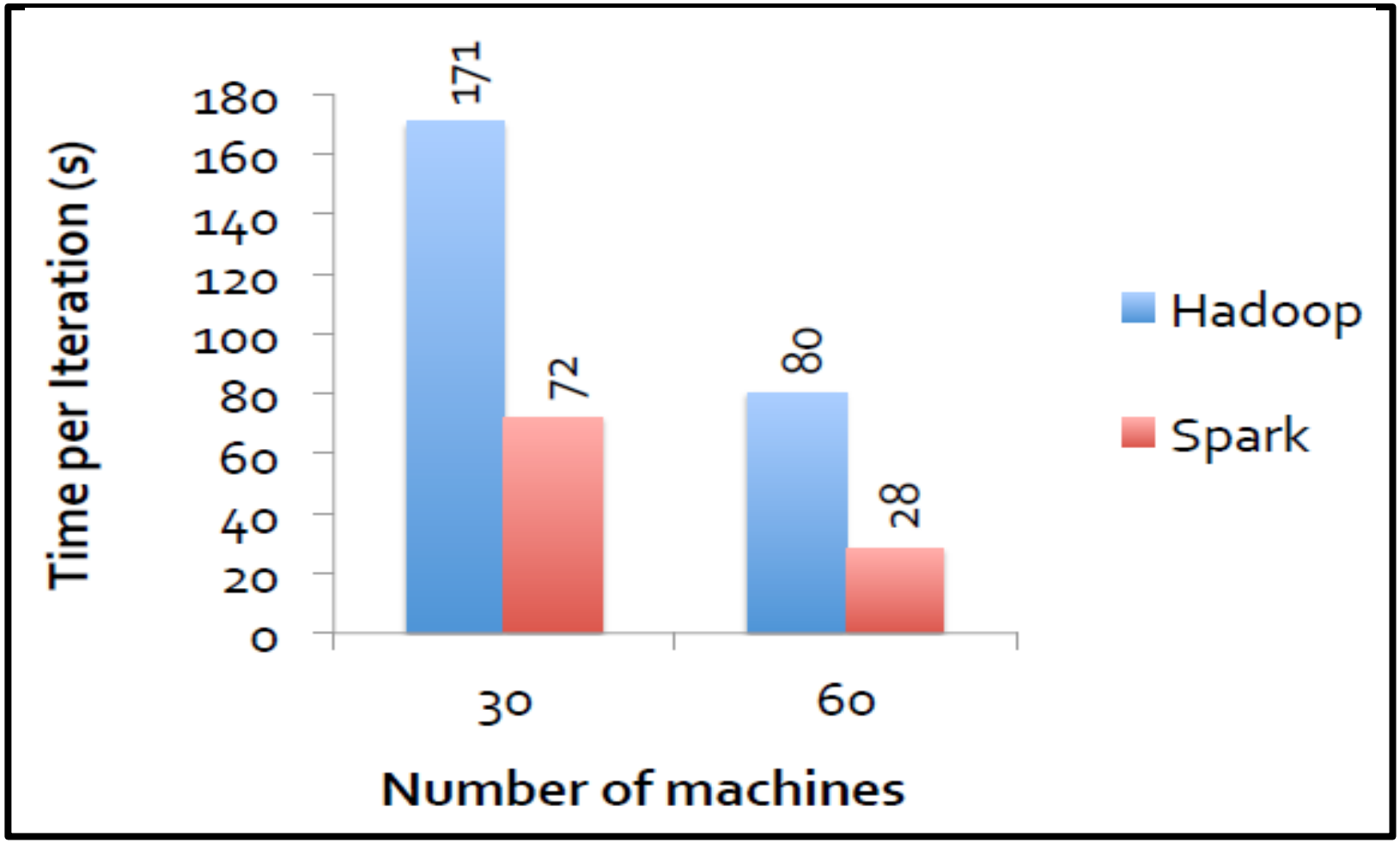
Step-3 Set each page's rank to **0.15 + 0.85 x contributions**



Spark Program

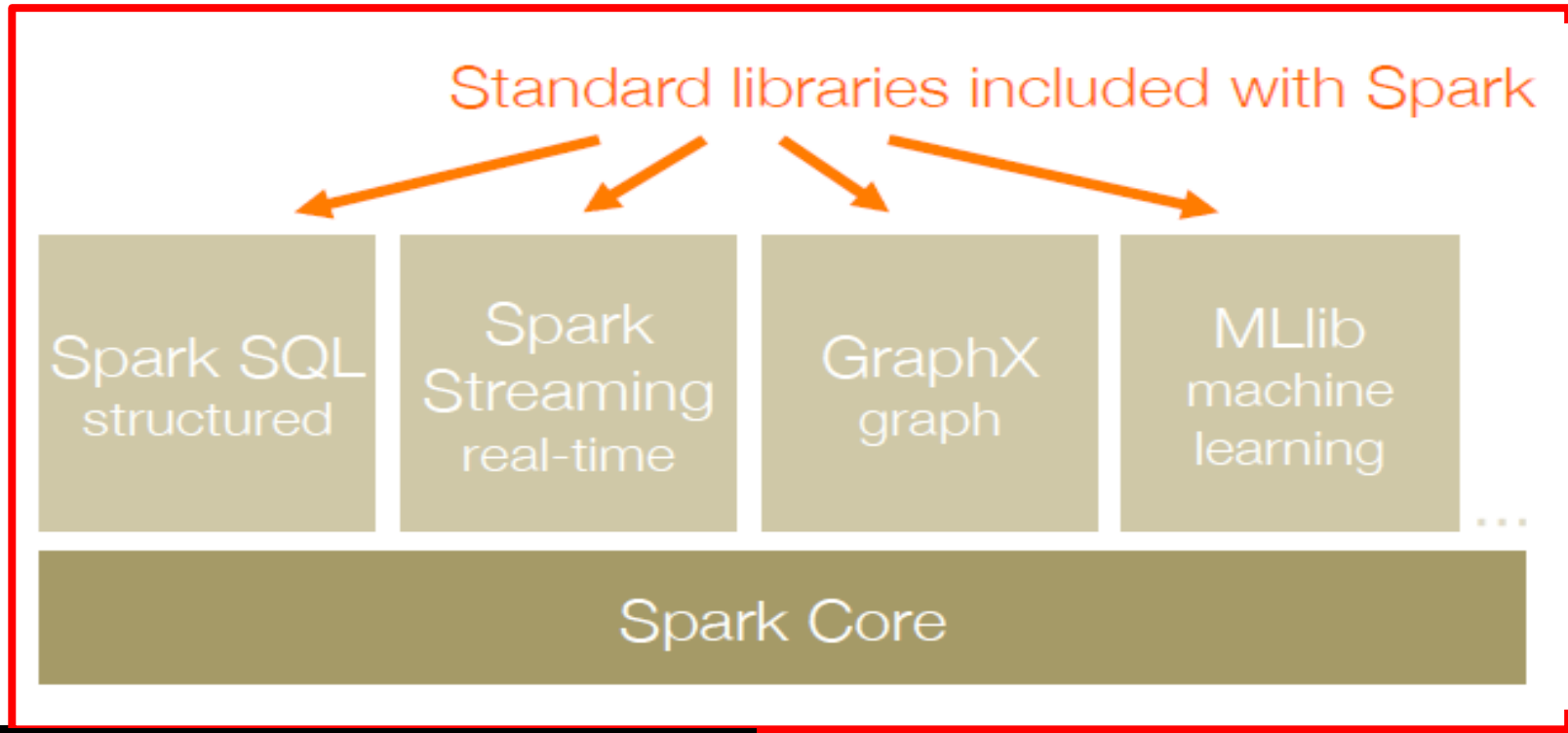
```
val links = // RDD of (url, neighbors) pairs
var ranks = // RDD of (url, rank) pairs
for (i <- 1 to ITERATIONS) {
    val contribs = links.join(ranks).flatMap {
        case (url, (links, rank)) =>
            links.map(dest => (dest, rank/links.size))
    }
    ranks = contribs.reduceByKey (_ + _)
                    .mapValues (0.15 + 0.85 * _)
}
ranks.saveAsTextFile(...)
```

PageRank Performance



Generality

- RDDs allow unification of different programming models
  - Stream Processing
  - Graph Processing
  - Machine Learning

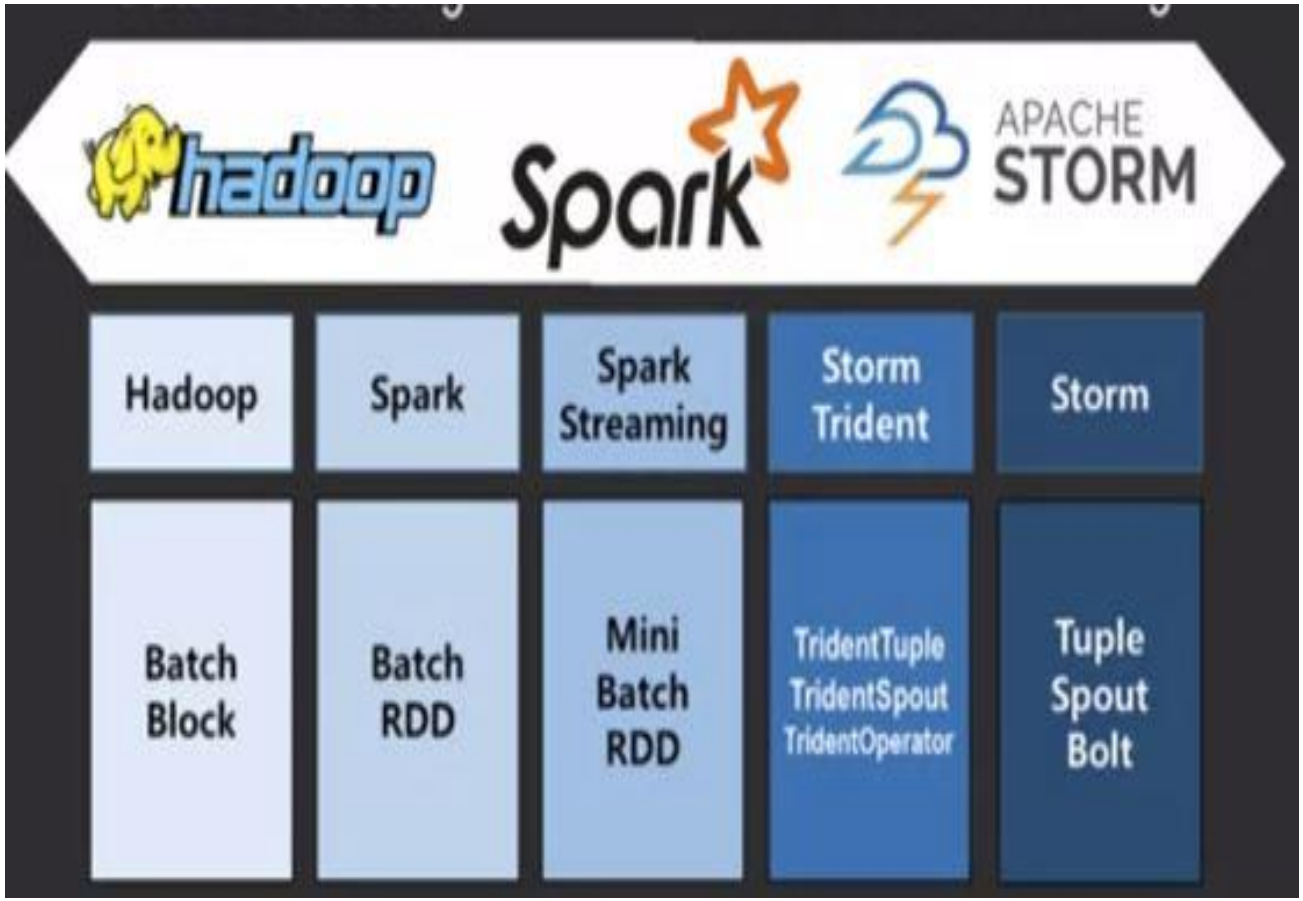


Big Data Analytics
Spark Streaming

What is Spark Streaming?

- Receive data streams from input sources, process them in a cluster, push out to databases/ dashboards
- Scalable, fault-tolerant, second-scale latencies

Spark Streaming



Spark Streaming

- Spark Streaming Characteristics
  - Spark Streaming Characteristics
  - Extension to the Spark Core API
  - Live data streams can be processed
  - Fault-tolerant and scalable
  - High throughput (near) real-time data processing 3 0.5 s or longer
  - Streaming data input from HDFS, Kafka, Flume, TCP sockets, Kinesis, etc.

Spark Streaming

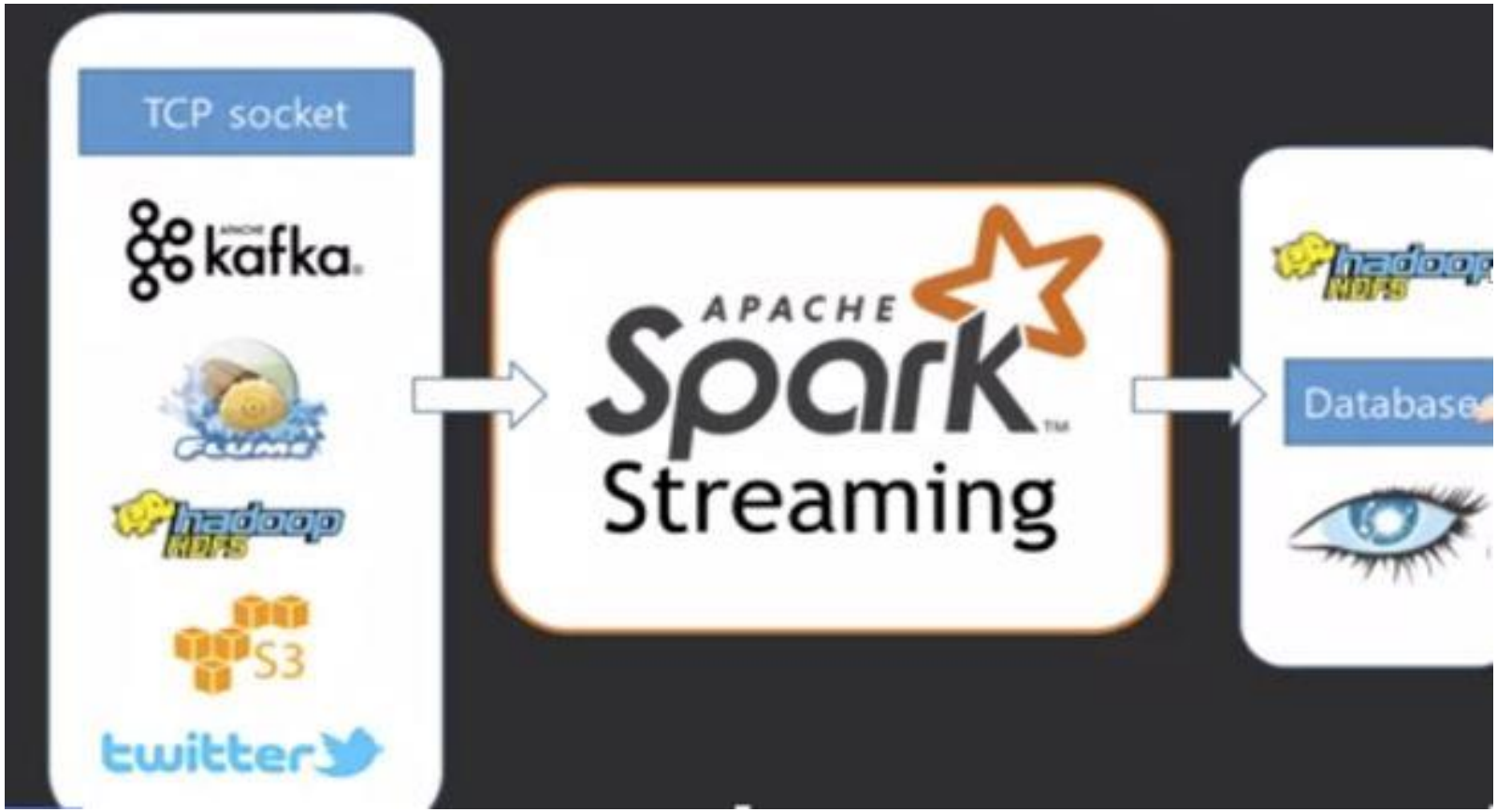
- Spark Streaming Characteristics
  - Stream processing high-level functions
- Map, Reduce, Join, Window, etc.
- Processed Output saved on Filesystems, Databases, and Live Dashboards
- Spark ML (Machine Learning) functions and GraphX graph processing algorithms are fully applicable to streaming data

Spark Streaming

- Spark Streaming Characteristics
  - Spark uses (size controllable) micro-batch processing of data for real-time analysis
  - Hadoop uses batch processing of data, which is time consuming to obtain results
  - Spark uses RDD to arrange data and recover from failures

Spark Streaming

- Spark Streaming Input and Output



Spark Streaming

- Streaming Receiver Types
  - Basic
    - File systems
    - Socket connectors
    - Akka Actors
    - Sources directly available in Streaming Context API
  - Custom
    - Requires implementing a user-defined receiver
    - Anywhere
  - Advanced
    - Requires linking with systems that have extra dependencies.
    - Kafka,Flume,Twitter

Spark Streaming

- Spark Streaming process
  1. Live input data stream received
  2. Input data stream is divided into Mini-Batches called a DStream (Discretized Stream), which is saved as a small RDD every mini-batch perm.
  3. Spark Stream engine cores process the mini batches and generate a final output stream of mini batches

Spark Streaming

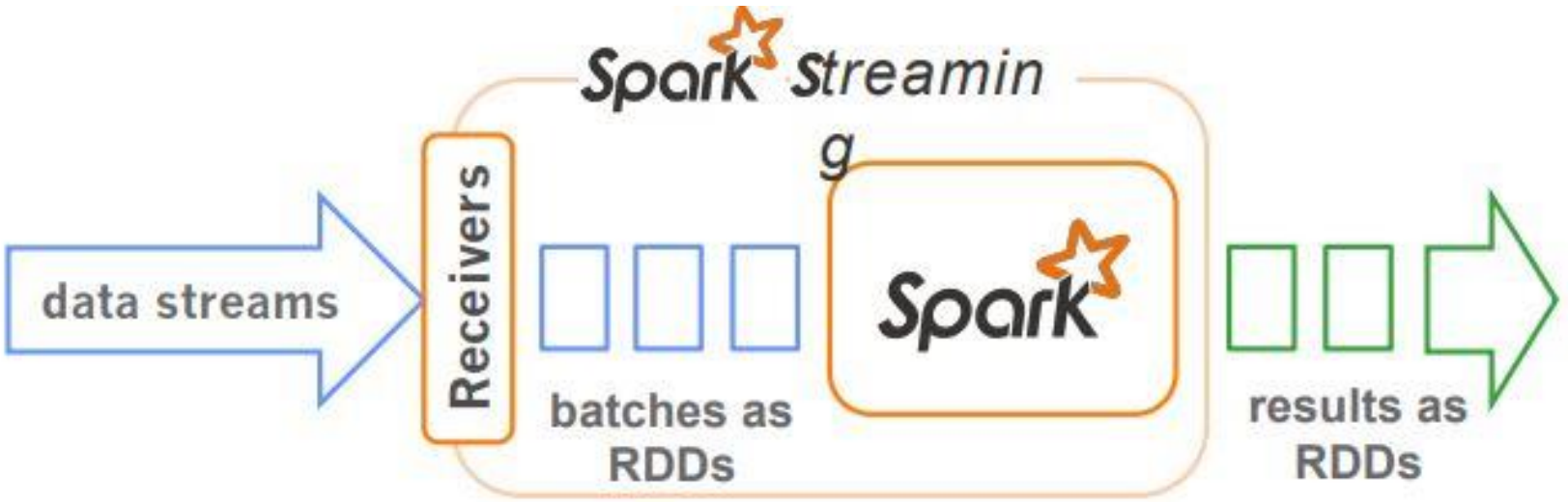
- Dstream
  - Dstream (Discretized Stream) is a continuous stream of data with high-level abstraction
  - DStreams are created from input data stream sources (Kafka, Flume, Kinesis, etc.) or high-level processing operationon other DStreams

Spark Streaming

- Dstream
  - Dstreamsare represented as a sequence of small RDDs
  - Mini-Batch size is 0.5 s or longer
  - RDD is processed through the DAG
  - Processing latency (through the DAG) has to be smaller than the mini-batch period

How does Spark Streaming work?

- Chop up data streams into batches of few secs
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Processed results are pushed out in batches



Spark Streaming Programming Model

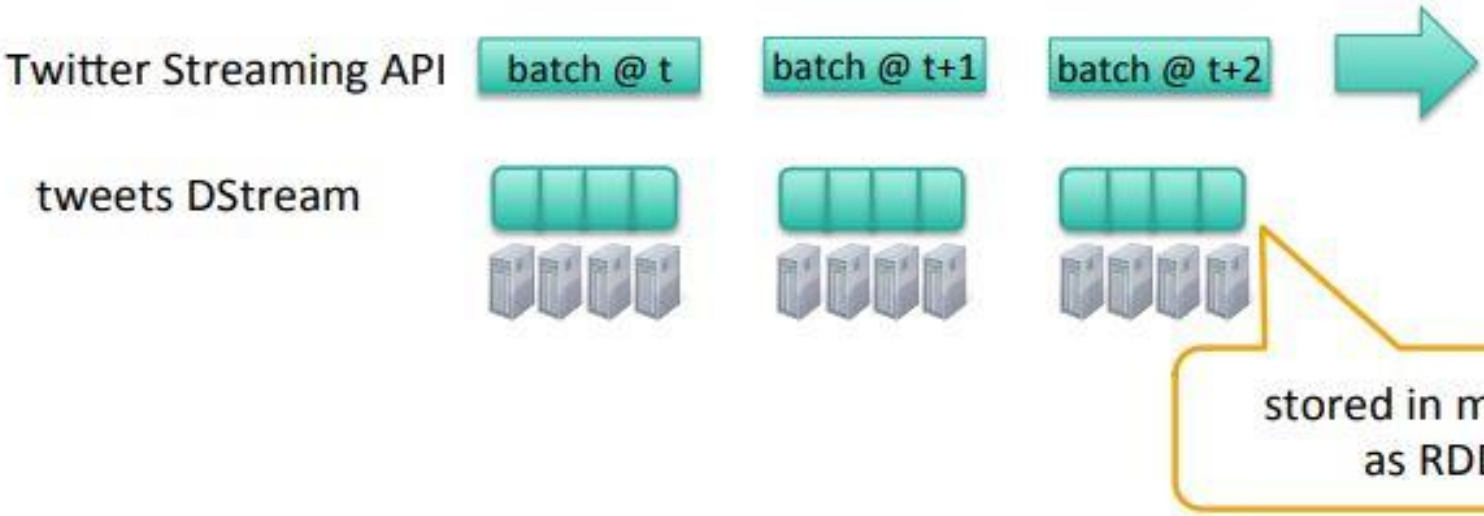
- Discretized Stream (DStream)
  - Represents a stream of data
  - Implemented as a sequence of RDDs
- DStreams API very similar to RDD API
  - Functional APIs in Scala, Java
  - Create input DStreams from different sources
  - Apply parallel operations

Example – Get hashtags from Twitter

Example – Get hashtags from Twitter

```
val ssc = new StreamingContext(sparkContext, Seconds(1))
val tweets = TwitterUtils.createStream(ssc, auth)
```

Input DStream

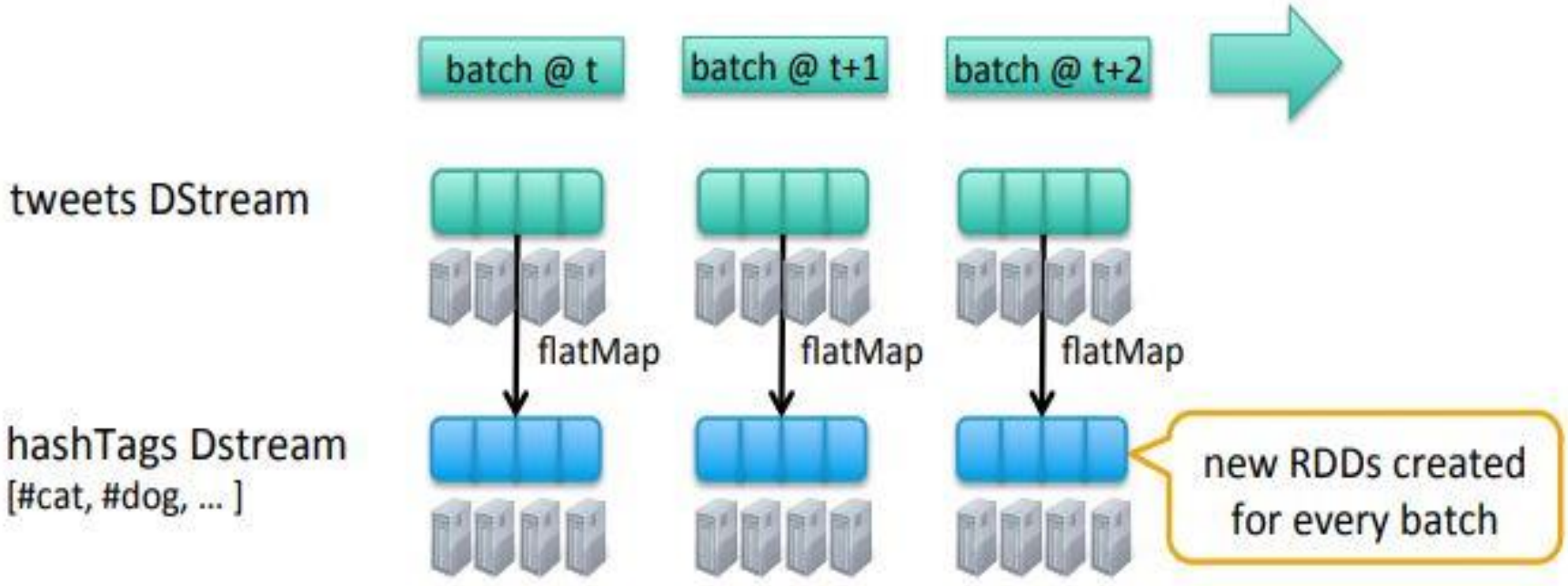


Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
```

transformed DStream

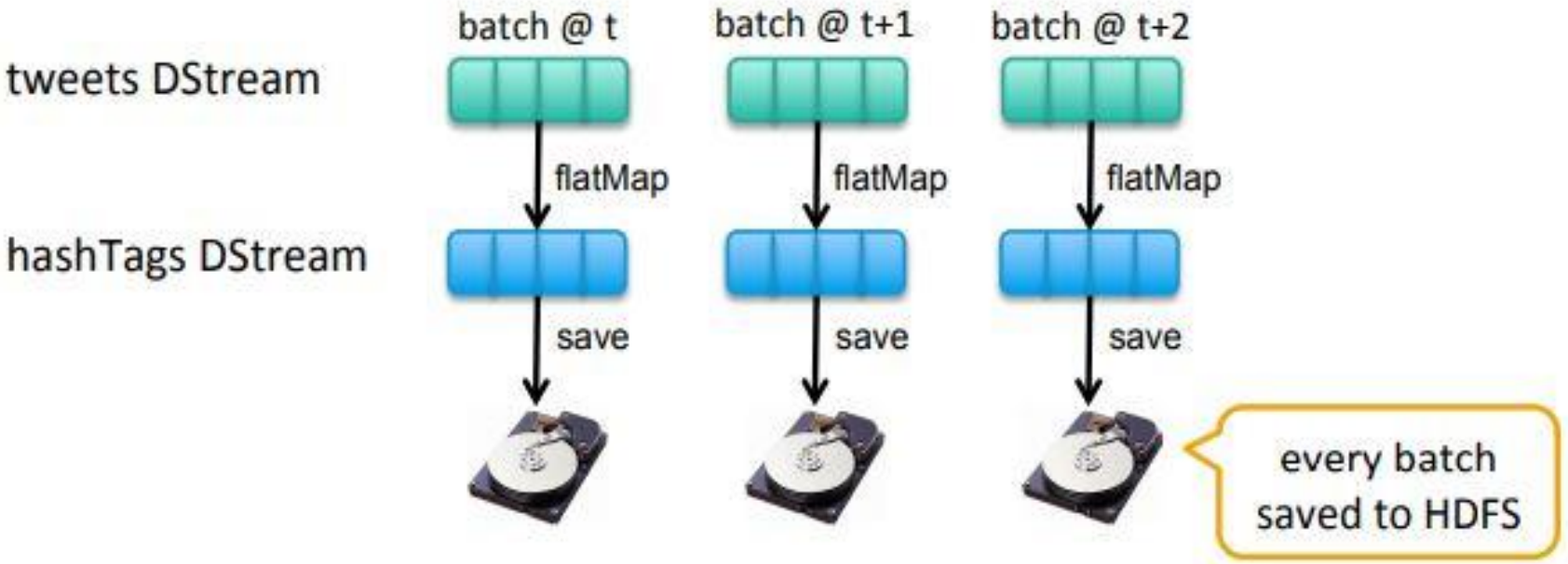
transformation: modify data in one DStream to create another DStream



Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

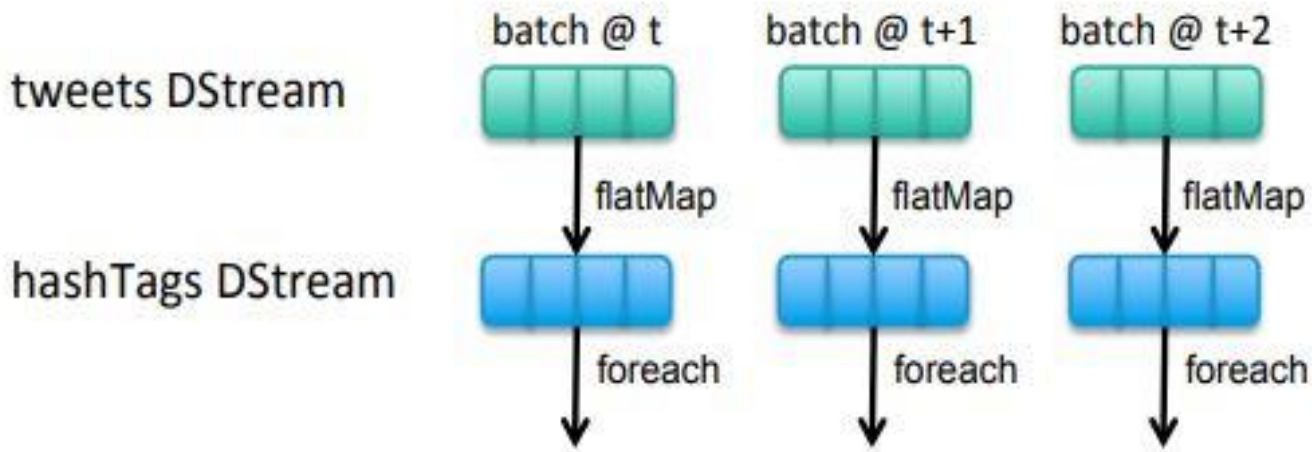
output operation: to push data to external storage



Example – Get hashtags from Twitter

```
val tweets = TwitterUtils.createStream(ssc, None)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.foreachRDD(hashTagRDD => { ... })
```

foreach: do whatever you want with the processed data



Write to a database, update analytics UI, do whatever you want

Languages

Scala API

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

Java API

Function object

```
JavaDStream<Status> tweets = ssc.twitterStream()
JavaDstream<String> hashTags = tweets.flatMap(new Function<...> { })
hashTags.saveAsHadoopFiles("hdfs://...")
```

Python API

...soon

Window-based Transformations

- Window Operations
  - Window Length
    - Number of blocks (partitions) to conduct a RDD DAG process together

  - Sliding Interval
    - Number of blocks (partitions) to slide the Window after a RDD process is conducted

Window-based Transformations

- Transformations for Windows
  - Key parameters: windowLength, slideInterval
  - window( )
  - countByWindow( )
  - reduceByWindow ( ) IP
  - reduceByKeyAndWindow( )
  - countByValueAndWindow( )
  - etc.

Window-based Transformations

- Window Operations
  - If Window Length > Sliding Interval, then Overlap Blocks = (Window Length - Sliding Interval) will exist for each RDD process

  - Overlap Blocks help to analyze correlation (dependency) of sequential blocks of the streamed data.

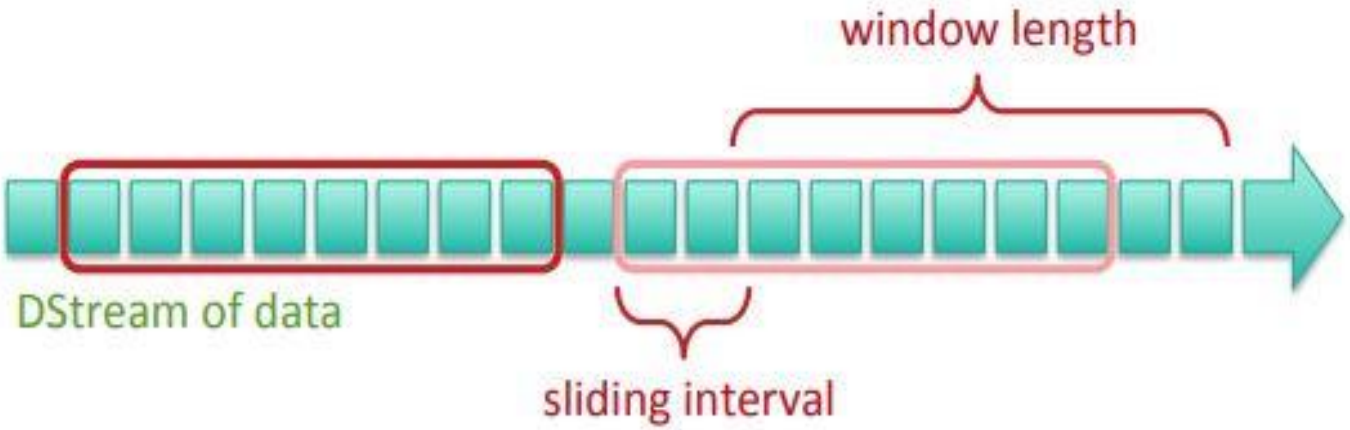
Window-based Transformations

```
val tweets = TwitterUtils.createStream(ssc, auth)
val hashTags = tweets.flatMap(status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```

sliding window operation

window length

sliding interval



Spark Streaming

- Spark Streaming Examples
  - IPTV or Web Page Live statistics
    - Channel or Page view of clicks
    - Use Kafka for buffering
    - Spark Streaming for processing
    - Draw a Heap Map of the current Channel or Page view clicks

Spark Streaming

- Spark Streaming Examples
  - Sales Product Type Monitoring
    1. Online Sales
       - Read through Kafka
    2. Department Store Sales
       - Read through Flume
  - Join 2 live data streams into Spark Streaming

Spark Streaming

Spark Streaming

- One Stream Input (e.g., from kafka)
  1. One Task slot in the Executor will serve as a Receiver (thread) to receive the live streaming data into a Block (Partition) of the RDD on the node
  2. Receiver will also make a copy of this Partition to another node (e.g., replication factor 2)
  3. DAG Transformations are executed on the new RDD

Spark Streaming

- Another Stream Added (e.g., from flume)
  1. On a different node, assign one Task slot in the Executor to serve as a Receiver (thread) to receive the live streaming data into a Block (Partition) of the RDD on the node
  2. Receiver will also make a copy of this Partition to another node (e.g. Replication Factor 2)

Spark Streaming

- Another Stream Added (e.g., from flume)
  3. DAG Transformations are executed on the new RDD
  4. Union can be used to unify the two RDDs into one RDD

Conclusion

Spark 1.0.0 Overview Programming Guides API Docs Deploying More

Spark Streaming Programming Guide

- Overview
- A Quick Example
- Basics
  - Linking
  - Initializing
  - DStreams
  - Input Sources
  - Operations
    - Transformations
    - Output Operations
  - Persistence
  - RDD Checkpointing
  - Deployment
  - Monitoring
- Performance Tuning
  - Reducing the Processing Time of each Batch
    - Level of Parallelism in Data Receiving
    - Level of Parallelism in Data Processing
    - Data Serialization
    - Task Launching Overheads
  - Setting the Right Batch Size
  - Memory Tuning
- Fault-tolerance Properties
  - Failure of a Worker Node
  - Failure of the Driver Node
- Migration Guide from 0.8.1 or below to 1.x
- Where to Go from Here

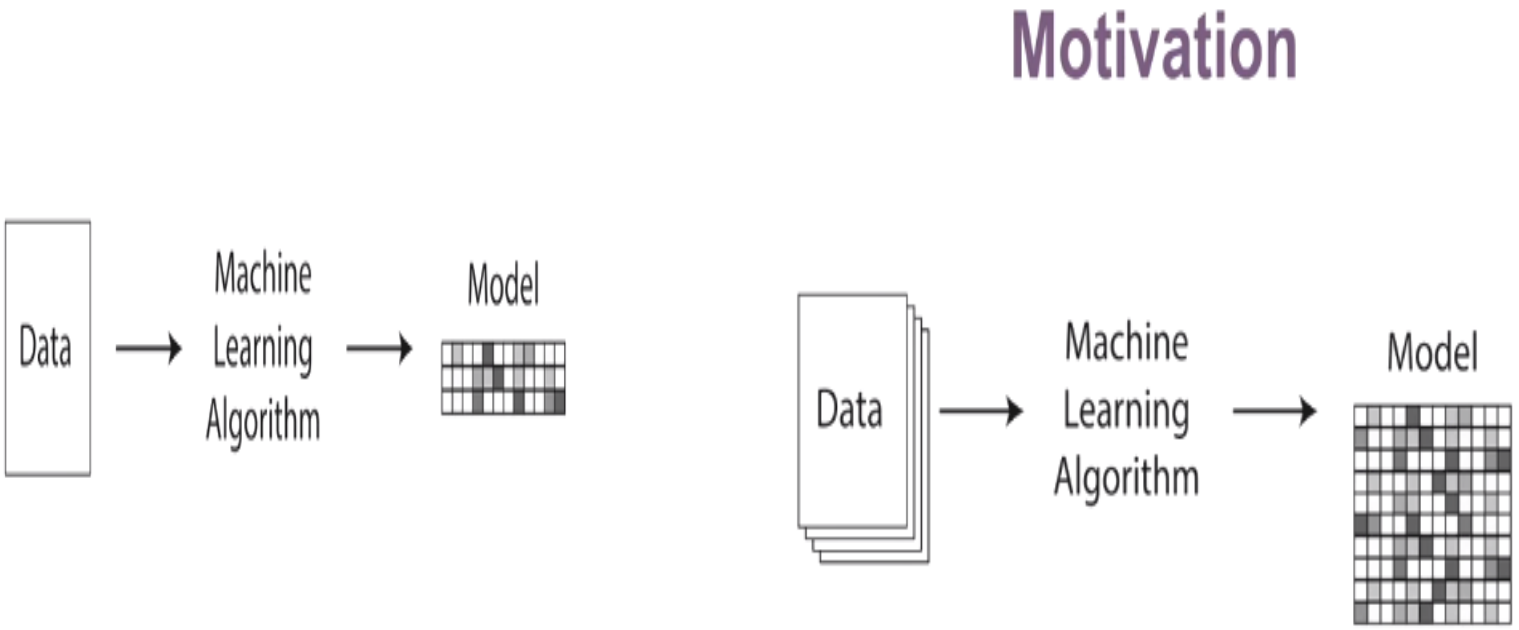
Overview

Spark Streaming is an extension of the core Spark API that allows enables high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ or plain old TCP sockets and be processed using complex algorithms expressed with high-level functions like `map`, `reduce`, `join` and `window`. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark's in-built machine learning algorithms, and graph processing algorithms on data streams.

Machine Learning Using Spark

What is Machine Learning ?

- Science of how machines learn without being explicitly programmed.



Machine Learning Use Cases

Healthcare

- Predict diagnosis
- Prioritize screenings
- Reduce re-admittance rates

Financial services

- Fraud Detection/prevention
- Predict underwriting risk
- New account risk screens

Public Sector

- Analyze public sentiment
- Optimize resource allocation
- Law enforcement & security

Retail

- Product recommendation
- Inventory management
- Price optimization

Telco/mobile

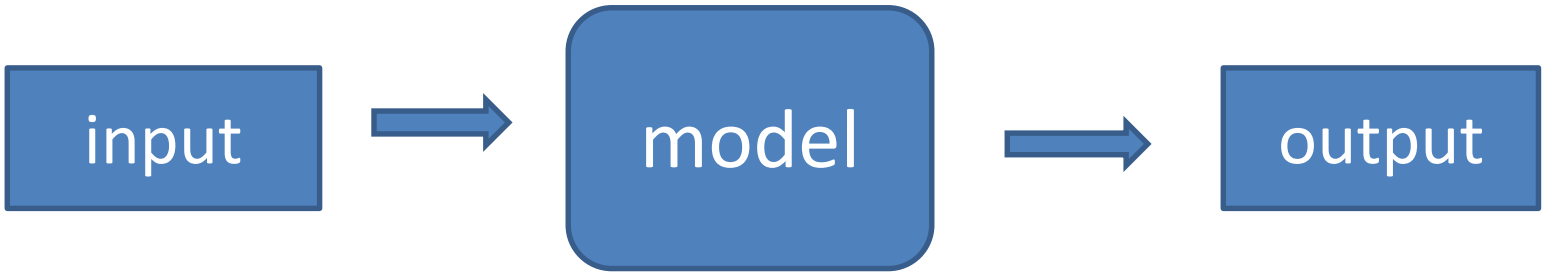
- Predict customer churn
- Predict equipment failure
- Customer behavior analysis

Oil & Gas

- Predictive maintenance
- Seismic data management
- Predict well production levels

What is ML Model ?

- **Mathematical formula** with a number of **parameters** that need to be **learned** from the **data**. And **fitting a model to the data** is a process known as **model training**.
- E.g. Linear Regression
  - Goal: fit a line y = mx + c to data points
  - After model training: y = 2x + 5



Components Of Machine Learning

- **CLASSIFICATION**

  Logistic Regression

  Naïve Bayes

  Support Vector Machine(SVM)

  Random Forest

- **COLLABORATIVE FILTERING**

  Alternating Least Squares(ALS)

- **REGRESSION**

  Linear regression

- **CLUSTERING**

  Kmeans , LDA

- **DIMENSIONALITY REDUCTION**

  Principal Component Analysis(PCA)

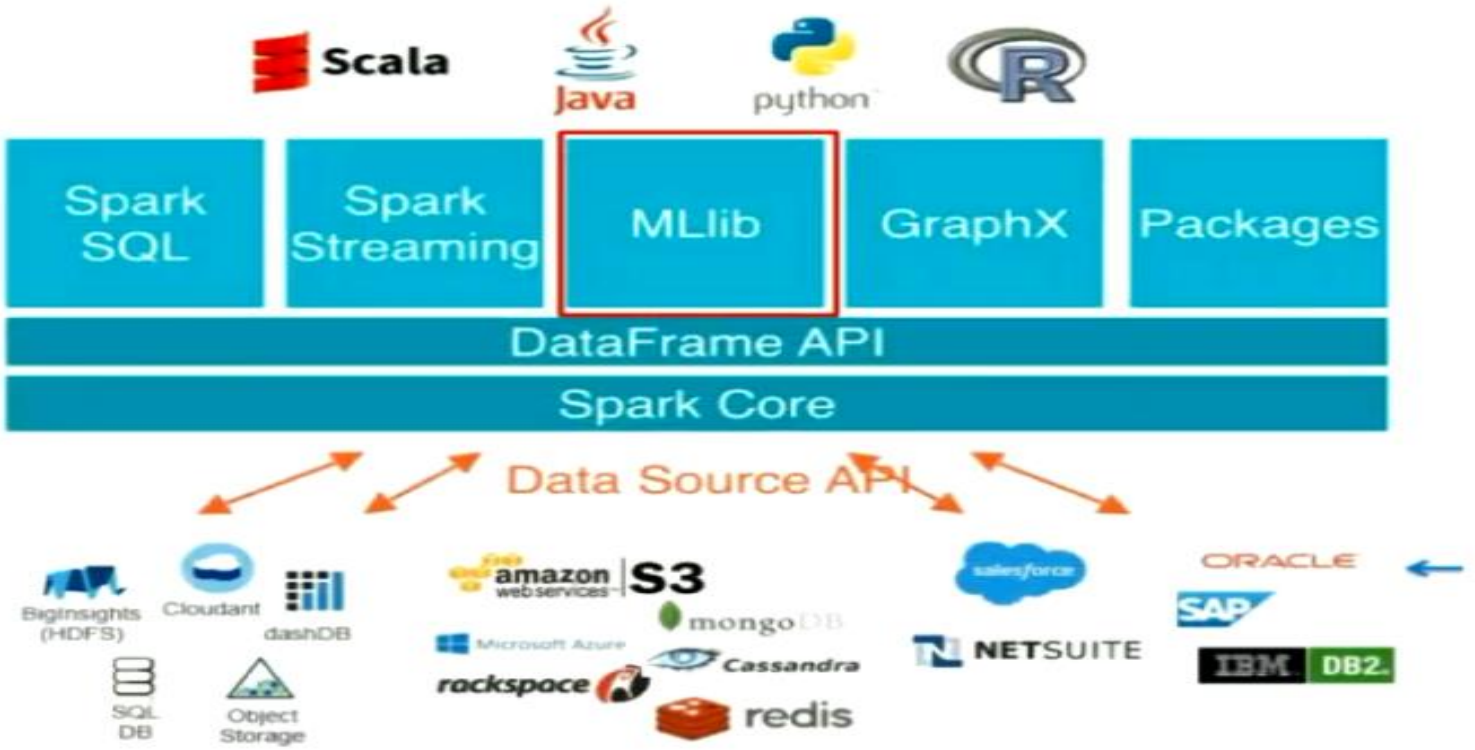
Linear Regression Model Training

- Linear regression is used to predict a quantitative response Y from the predictor variable X.
- Linear Regression is made with an assumption that there's a linear relationship between X and Y.
- **ONE FEATURE**

```
import org.apache.spark.ml.regression.LinearRegre

// Initialize model
val lr2D = new LinearRegression()

// Fit the model
val lrModel2D = lr1.fit(scatterData)
```

ML in SPARK

- We have an utility in SPARK to implement machine learning called **MLlib** .



Spark MLlib (Goals)

- Practical machine learning is scalable and easy to use.
- Simplify the development and deployment of scalable machine learning pipelines.
- ~75 organizations, ~200 individuals, ~20 companies.

Spark MLlib Components

ALGORITHMS

- Classification
- Regression
- Collaborative Filtering
- Clustering

PIPELINE

- Constructing
- Evaluating
- Tuning
- Persistence

Spark MLlib Components

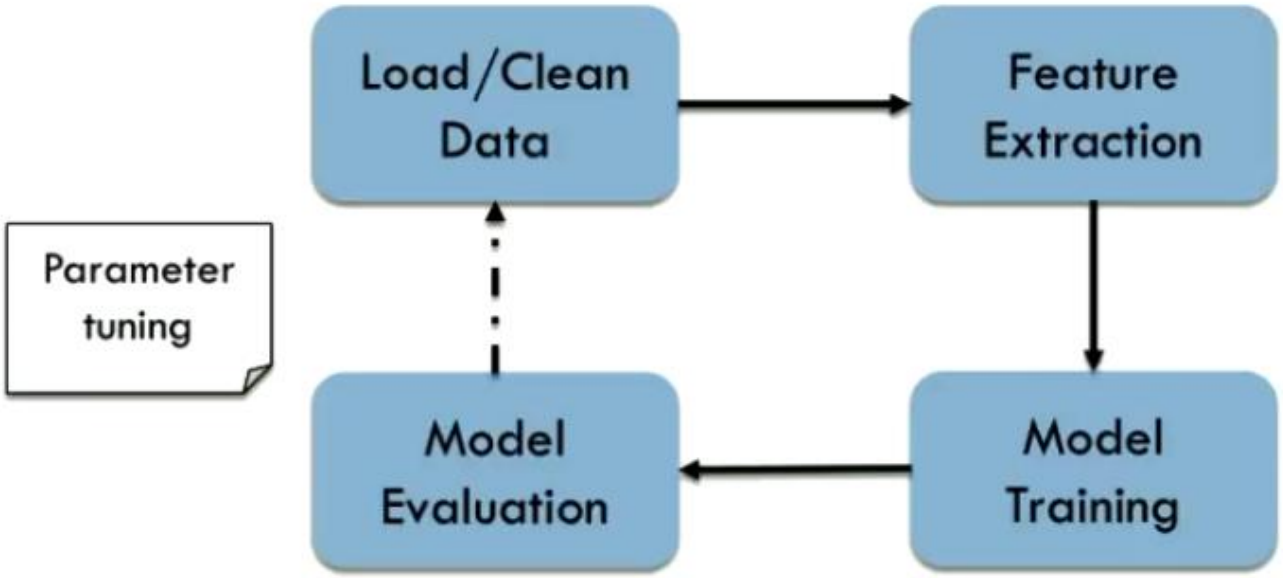
UTILITIES

- Linear Algebra
- Statistics

FEATURIZATION

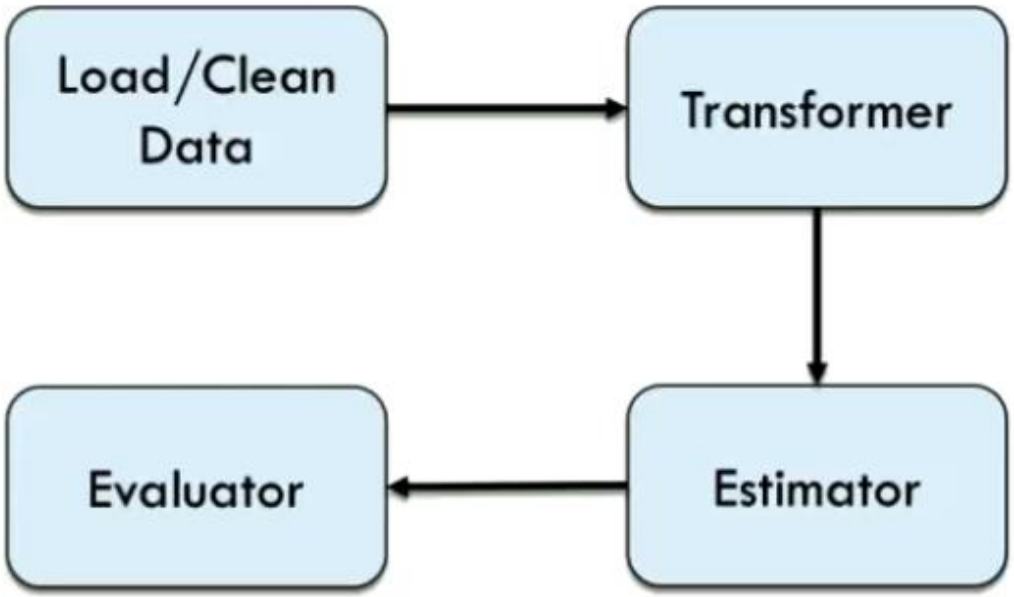
- Extraction
- Transformation

Spark MLlib Pipeline



Repeatable Workflow Process

Spark MLlib Pipeline Concepts



Spark MLlib: Transformer

- Preprocessing step of feature extraction.
- Transforming data into consumable format.
- Take input column, transform it to an output form.
- Examples:-
    - Normalizes the data.
    - Tokenization – sentences into words.
    - Converting categorical values into numbers.

Spark MLlib: Estimator

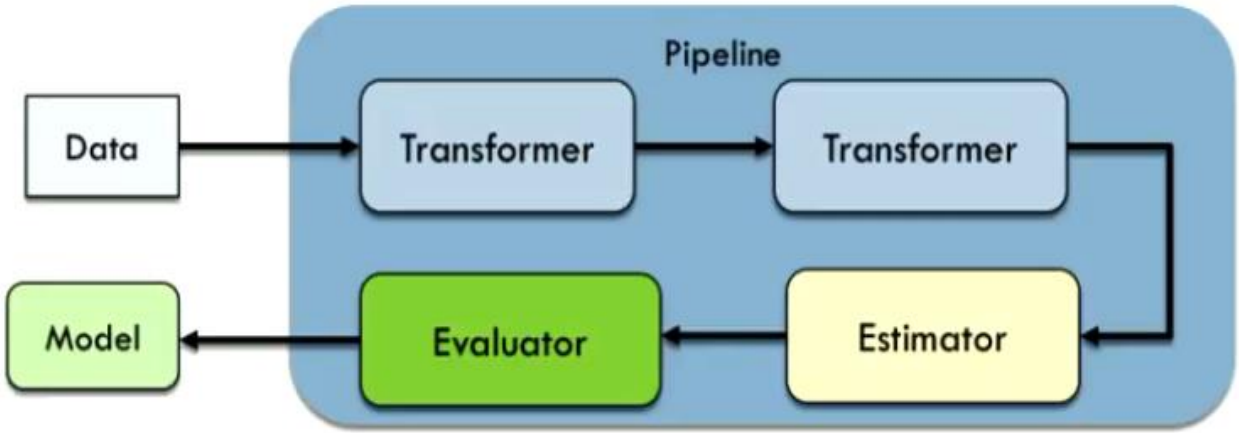
- Another kind of Transformer.
  - Transform data by requiring two passes over data.
- Learning algorithm that trains (fits) on data.
- Return a model, which is a type of Transformer.
- Examples:
  - LogisticRegression.fit() => LogisticRegressionModel

Spark MLlib: Evaluator

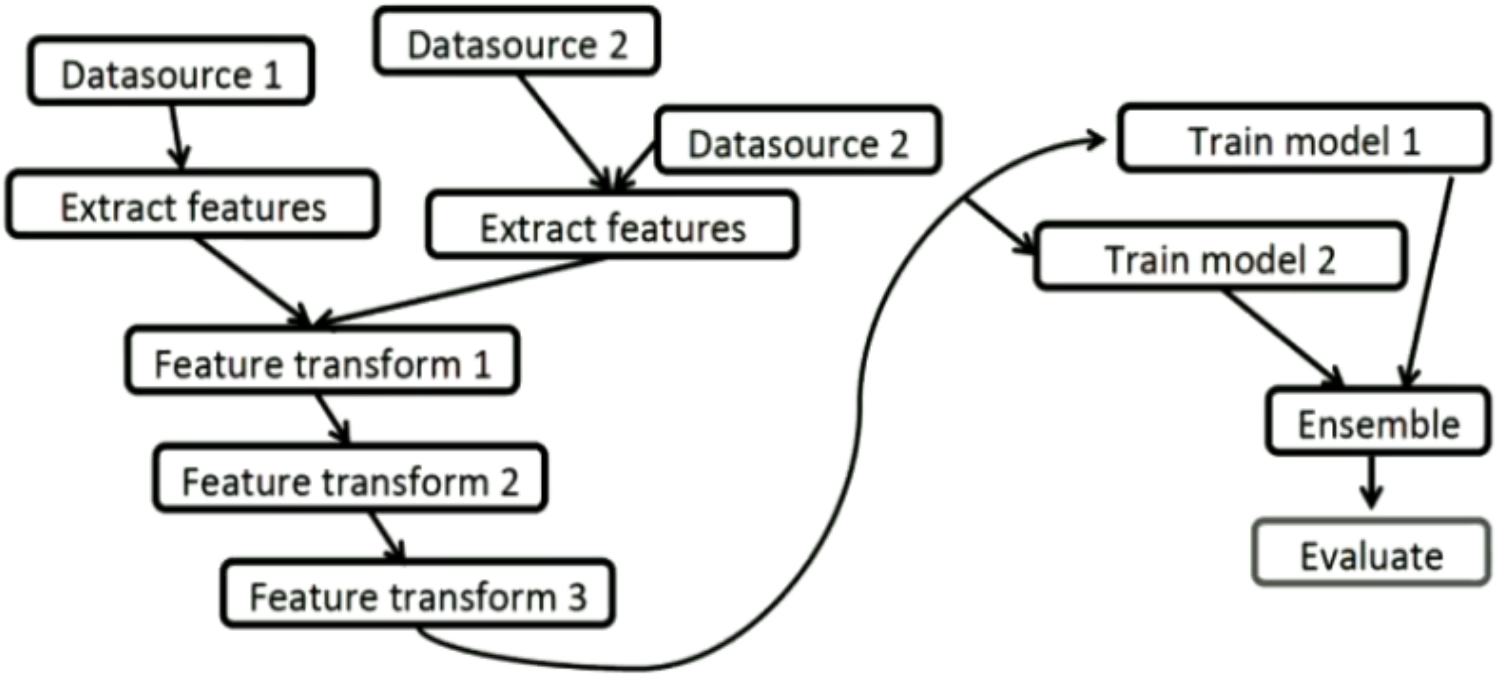
- Evaluate the model performance based certain metric.
    - ROC, RMSE
- Help with automating the model tuning process.
    - Comparing model performance.
    - Select the best model for generating predictions.
- Examples:
    - Binary Classification Evaluator, Cross Validator.

Spark MLlib : Pipeline

- To represent a ML workflow.
- Consist of a set of stages.
- Leverage the uniform API of Transformer & Estimator.
- A type of Estimator - fit().
- Can be persisted.



Spark MLlib: Machine Learning Pipeline

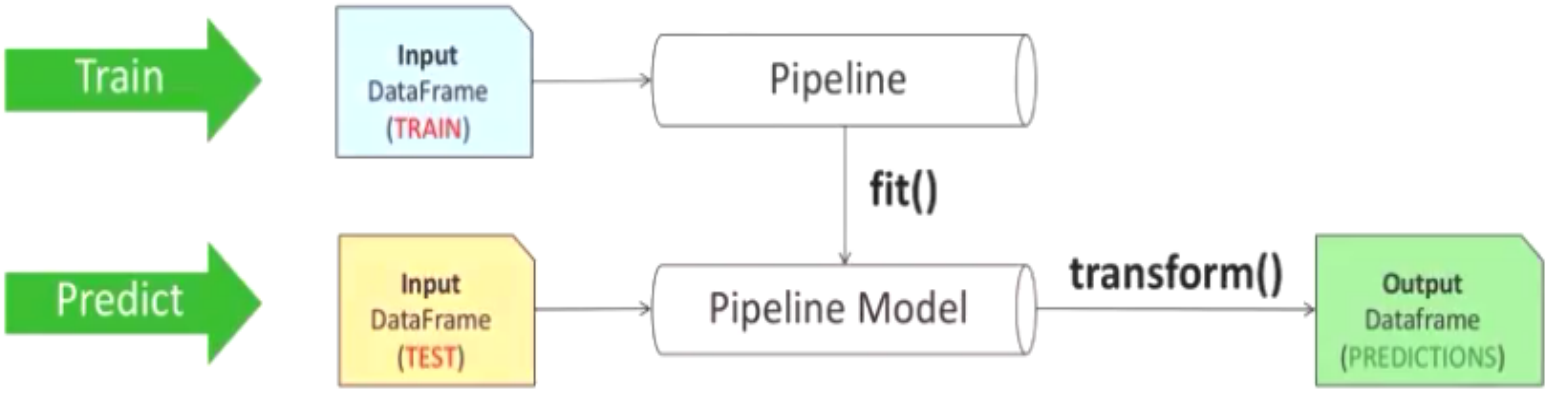


Reality is messy

Spark ML Pipeline

Spark ML Pipeline

- **fit()** is for **training**
- **transform()** is for **prediction**

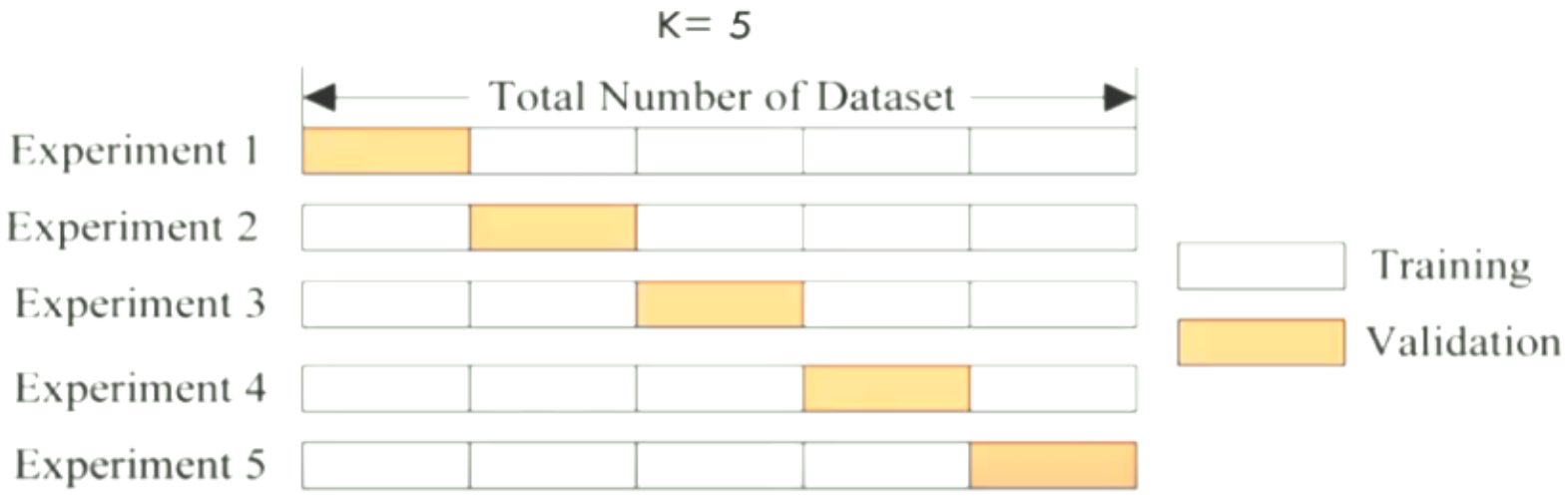


Spark MLlib : Pipeline Example in Scala

```
val tokenizer = new Tokenizer().setInputCol("text")
                               .setOutputCol("words")

val hashingTF = new HashingTF().setNumFeatures(1000)
                    .setInputCol(tokenizer.getOutputCol)
                    .setOutputCol("features")

val lr = new LogisticRegression().setMaxIter(10)
                                 .setRegParam(0.001)

val pipeline = new Pipeline().setStages(
                       Array(tokenizer, hashingTF, lr)
                    )

// Fit the pipeline to training data
val model = pipeline.fit(training)
```

Spark MLlib: Automatic model tuning

- ParamGridBuilder
- Cross Validator (k-fold)



Exporting ML Models - PMML

- Predictive Model Markup Language (PMML)
  - XML-based predictive model interchange format
- Supported models
  - K-means
  - Linear Regression
  - Ridge Regression
  - Lasso
  - SVM
  - Binary

Spark ML Pipeline – Sample Code

```
indexer = …
parser = …
hashingTF = …
vecAssembler = …

rf = RandomForestClassifier(numTrees=100)
pipe = Pipeline(stages=[indexer, parser, hashingTF, vecAssembler, rf])

model = pipe.fit(trainData)                # Train model
results = model.transform(testData)        # Test model
```

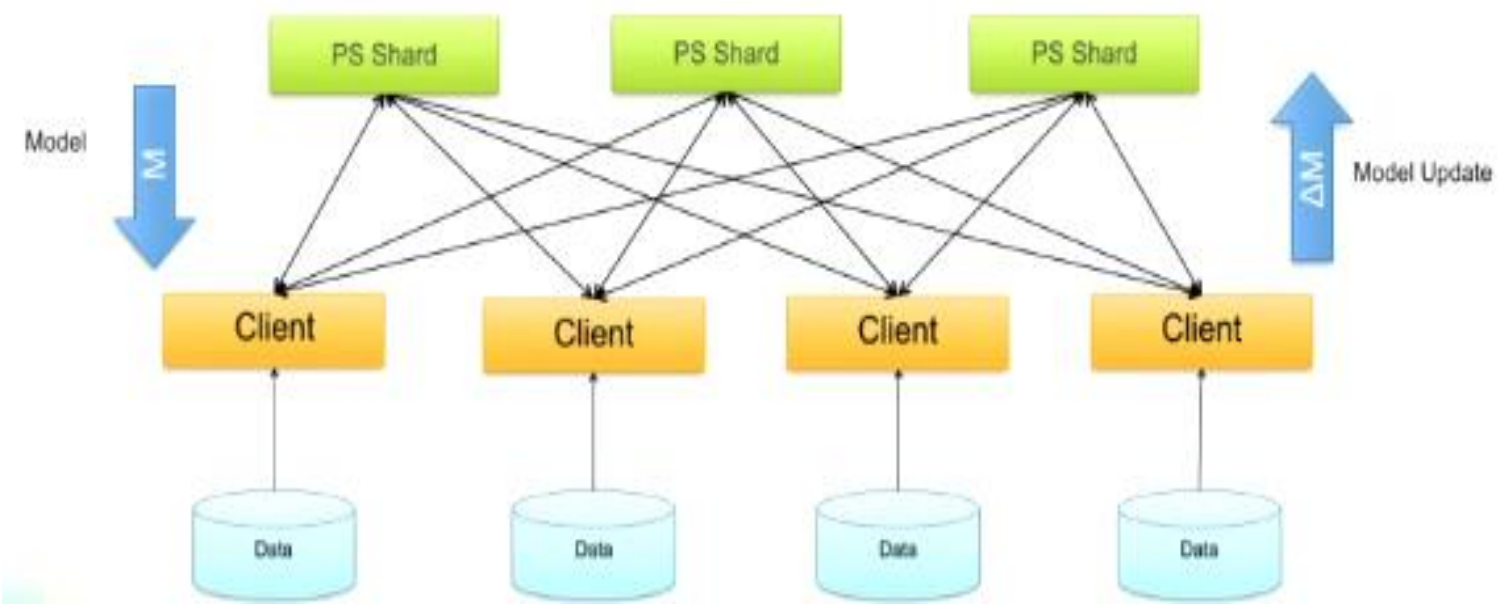
Parameter Server

- A machine learning framework
- Distributes a model over multiple machines
- Offers two operations:
  - Pull: query parts of the model
  - Push: update parts of the model
- Machine learning update equation

- (Stochastic) gradient descent
- Collapsed Gibbs sampling for topic modeling
- Aggregate push updates via addition (+)

Parameter Server (PS)

- Training state stored in PS shards, asynchronous updates.



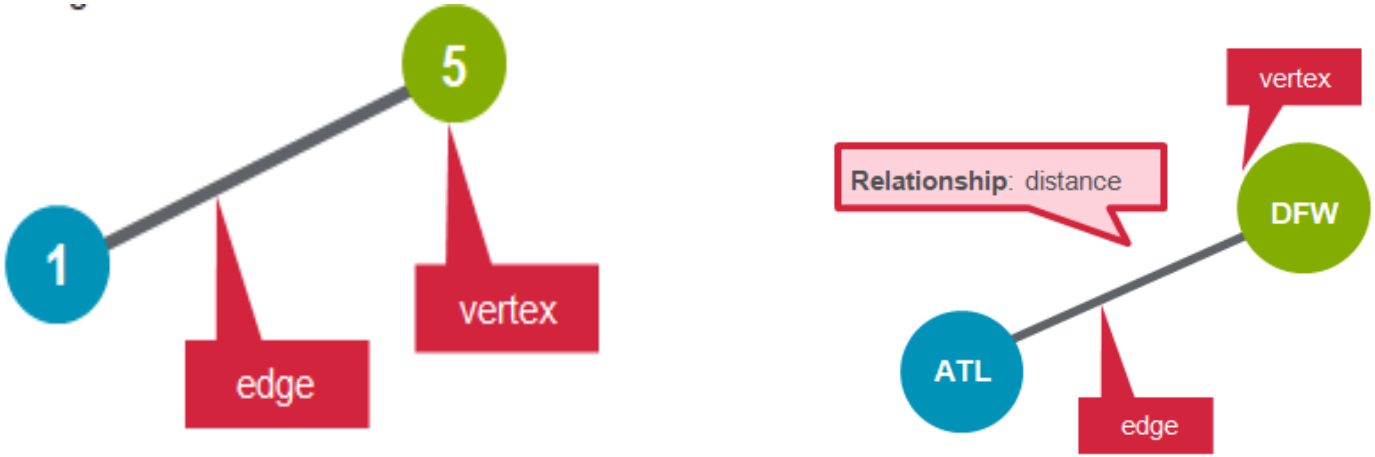
The GraphX API

What is a Graph?

Graph: vertices connected by edges

$$G = (V, E)$$

Graph Vertices Edges



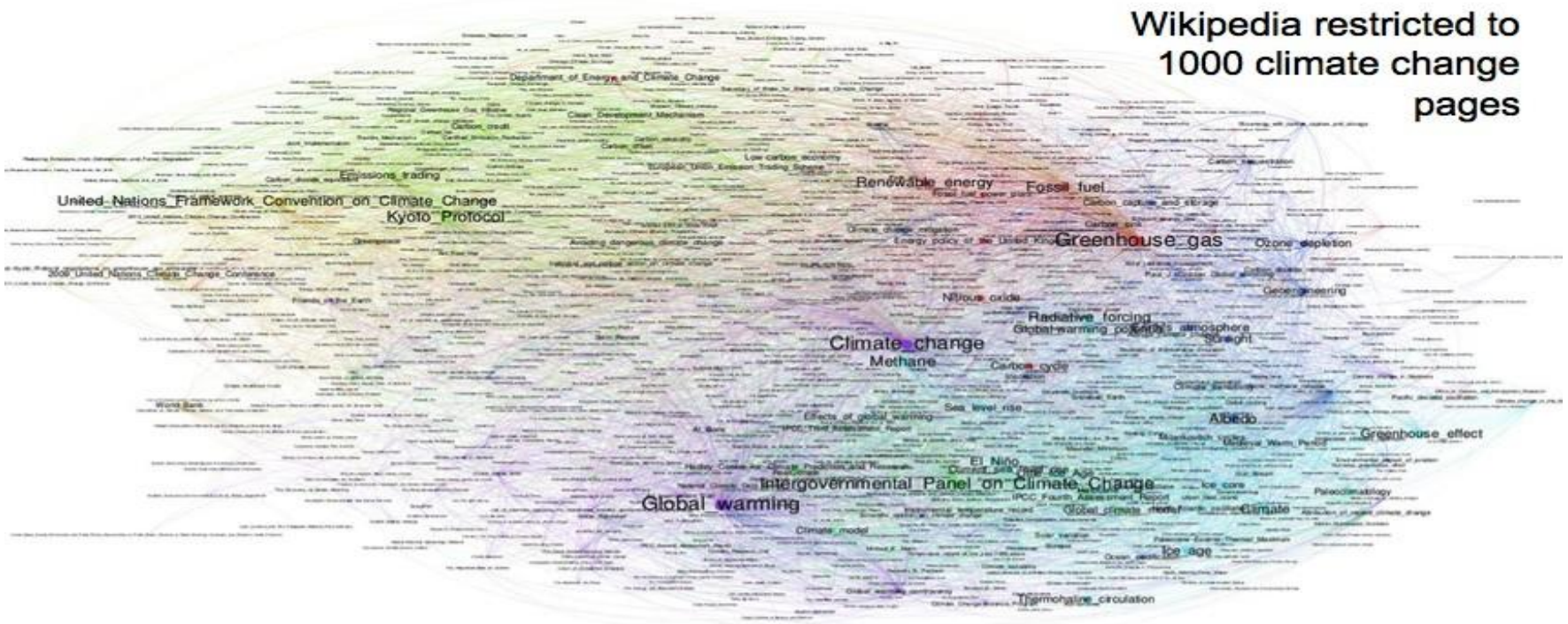
Graphs are Essential to Data Mining and Machine Learning

- Identify influential entities (people, information…)
- Find communities
- Understand people's shared interests
- Model complex data dependencies

Real World Graphs

- Web pages

Web Graphs



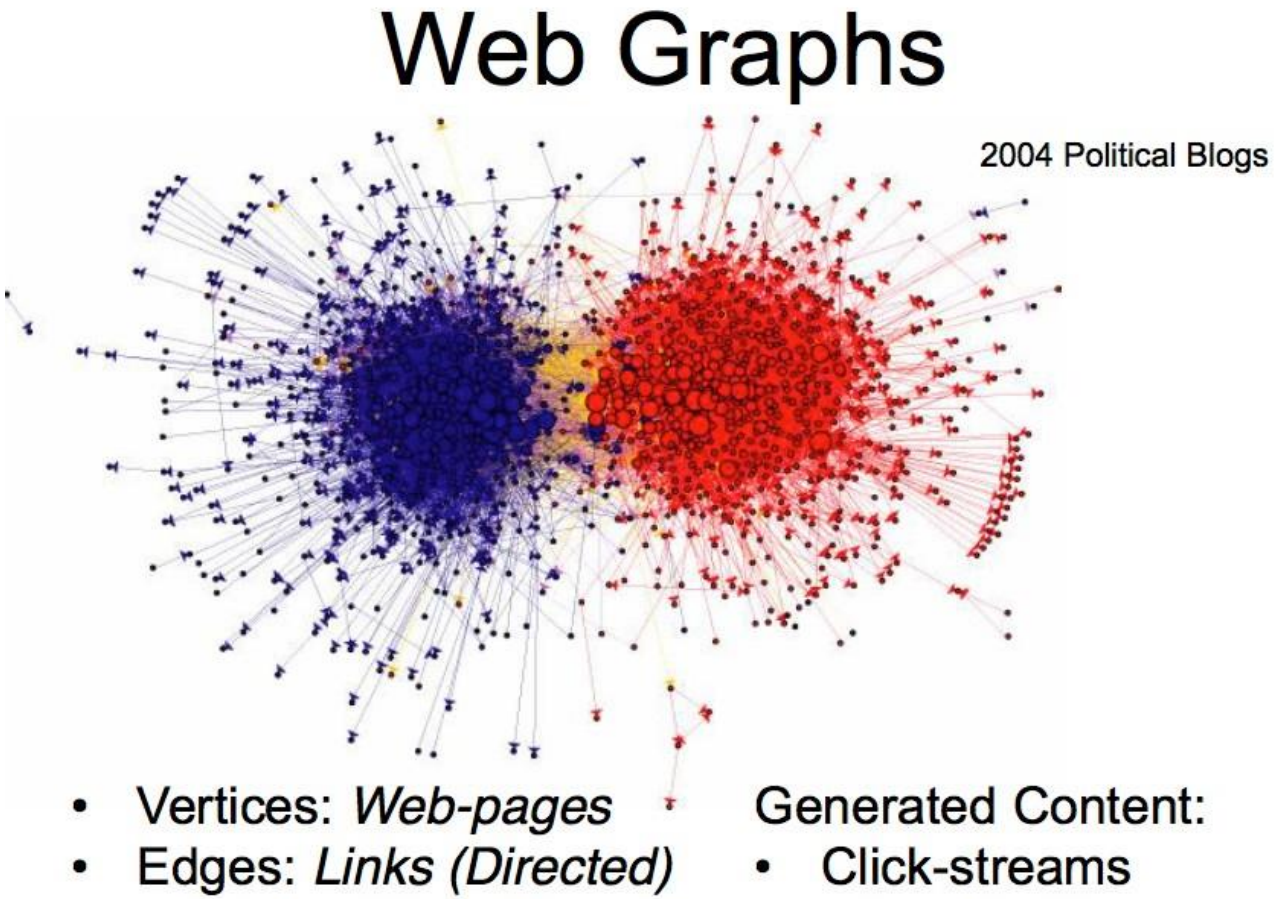
- Vertices: *Web-pages*
- Edges: *Links (Directed)*

Generated Content:

- Click-streams

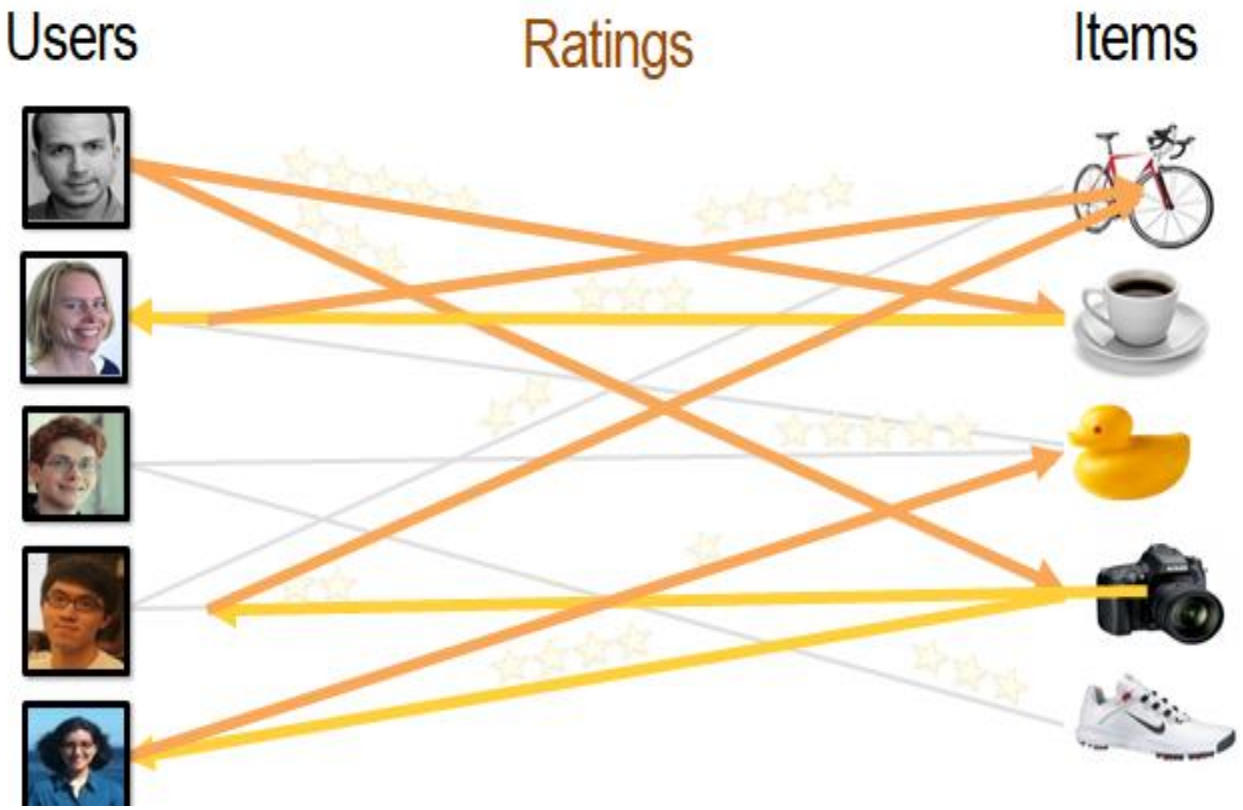
Real World Graphs

- Web pages



Real World Graphs

- Recommendation



Real World Graphs

- Credit card fraud detection

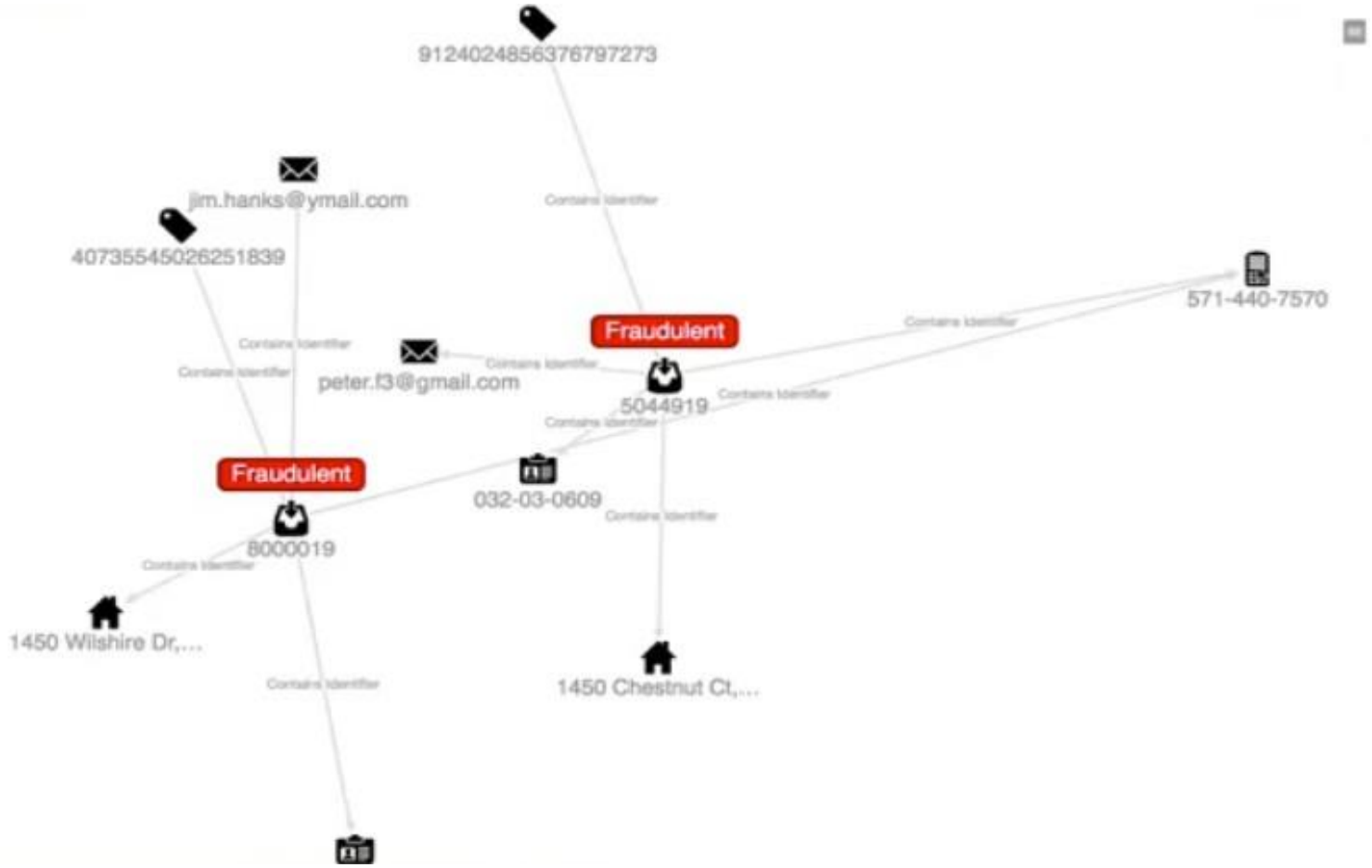
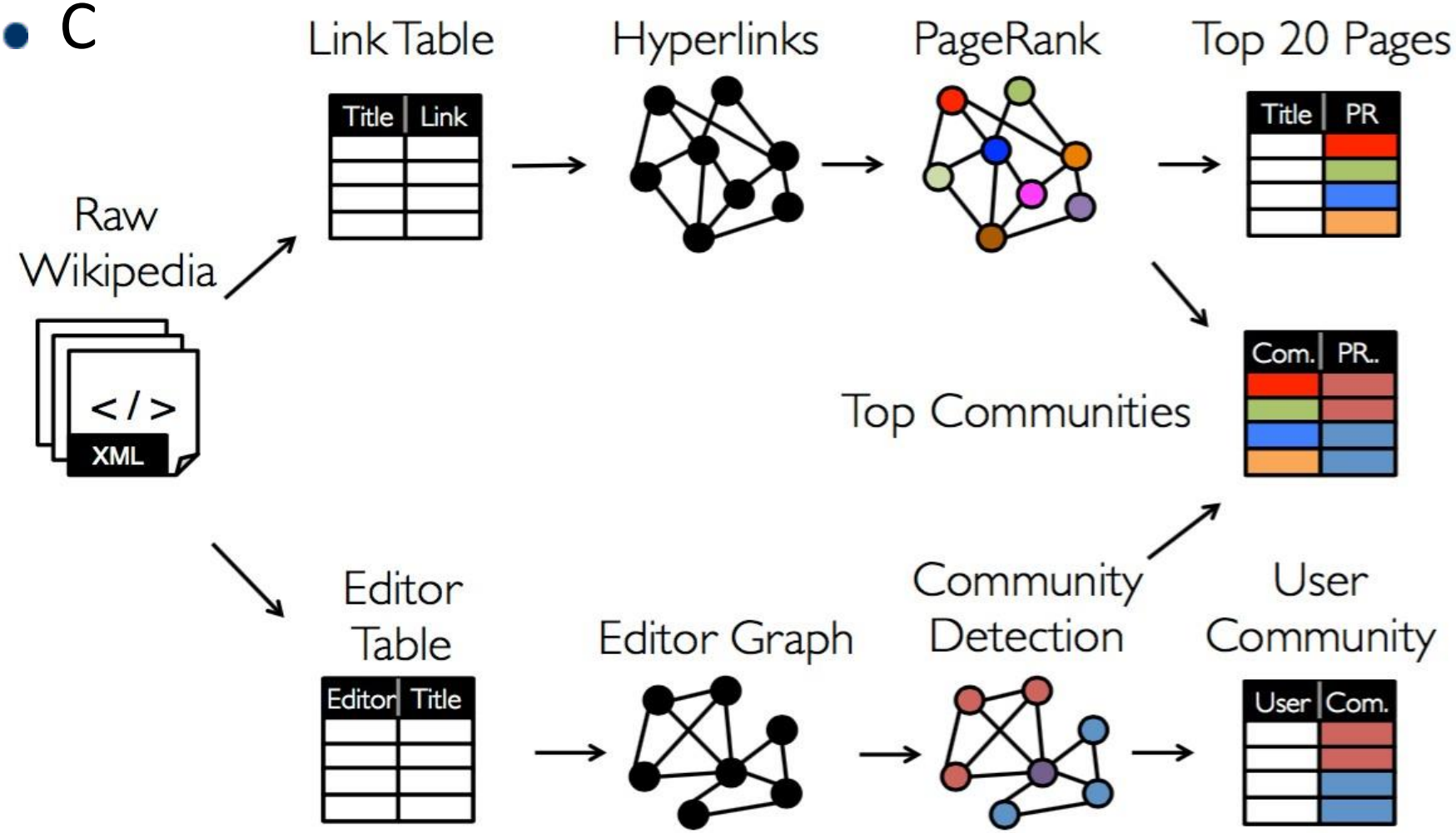


Table and Graph Analytics

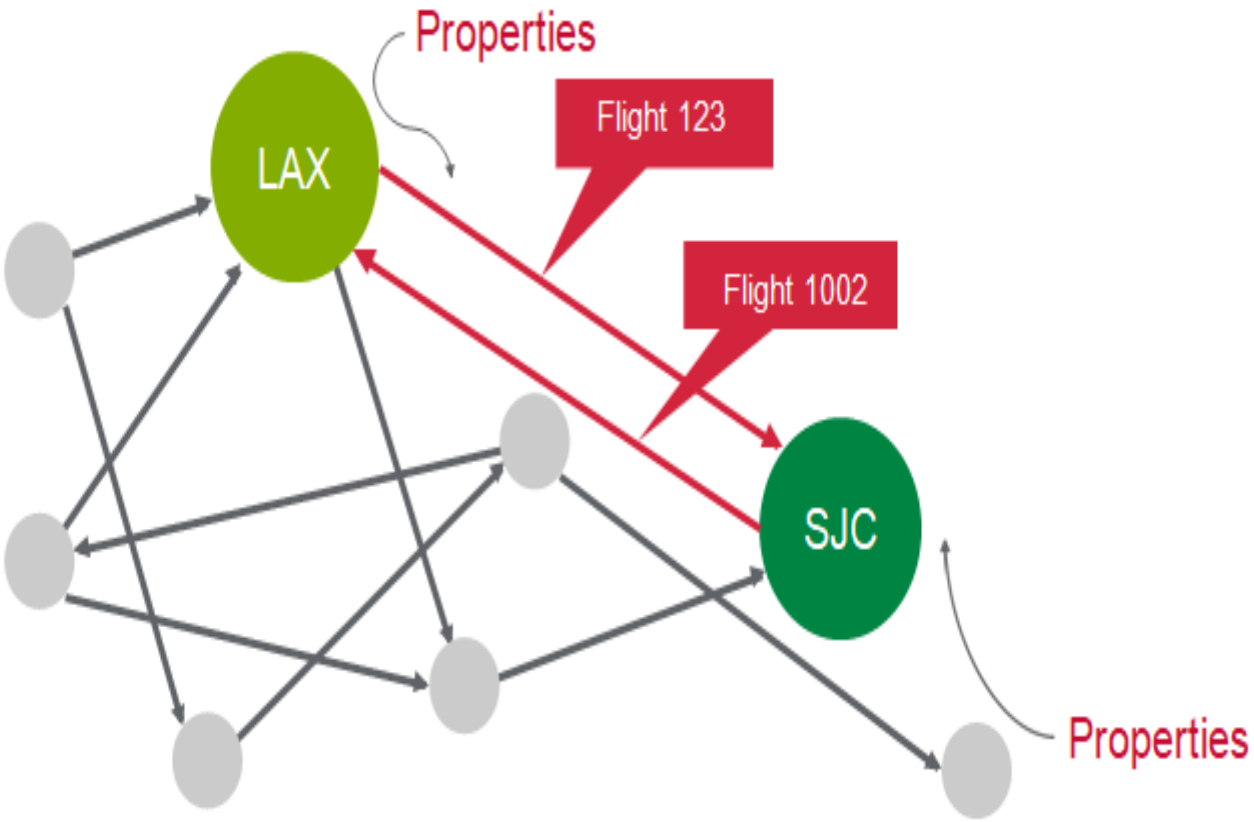
- C



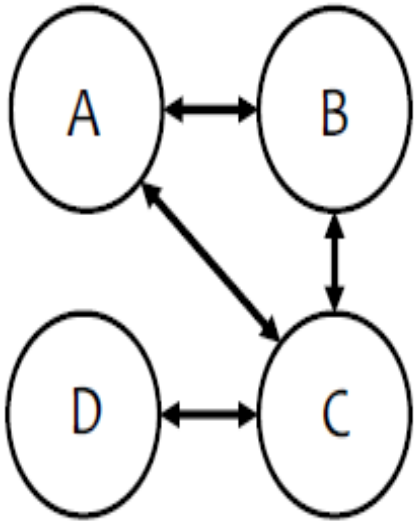
Apache Spark GraphX

- Spark component for graphs and graph- parallel computations
- Combines data parallel and graph parallel processing in single API
- View data as graphs and as collections (RDD)
  - no duplication or movement of data
- Operations for graph computation
  - includes optimized version of Pregel
- Provides graph algorithms and builders

Property Graph



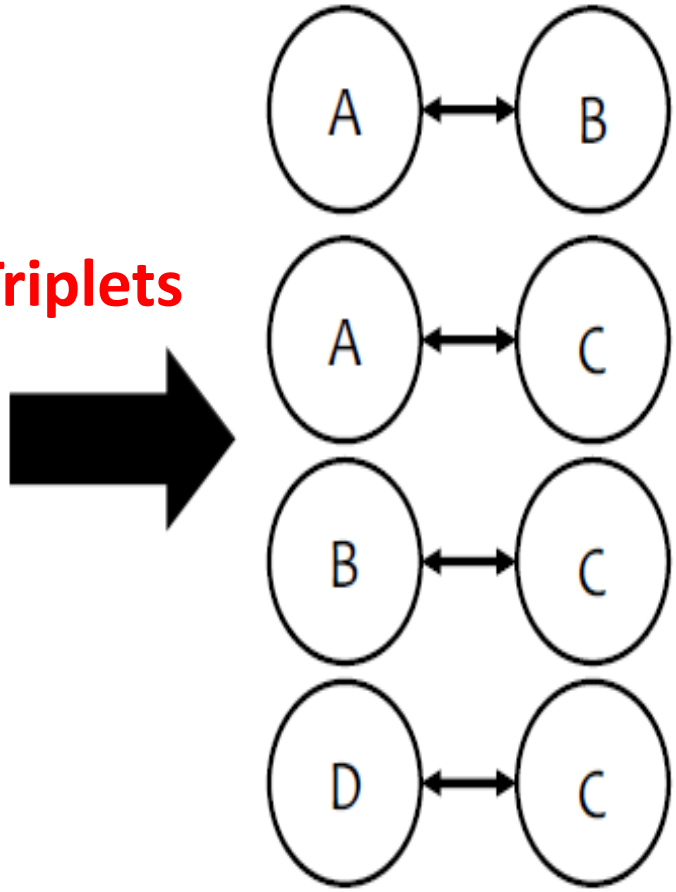
Gather-Apply-Scatter on GraphX



Vertices	Neighbors
A	B
A	C
B	C
D	C

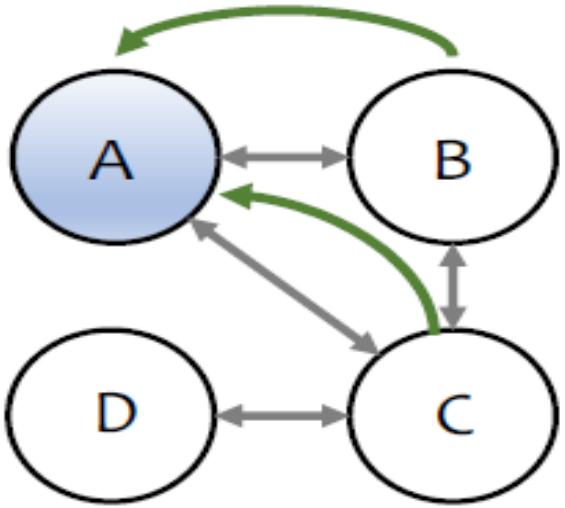
Graph Represented In a Table

Triplets

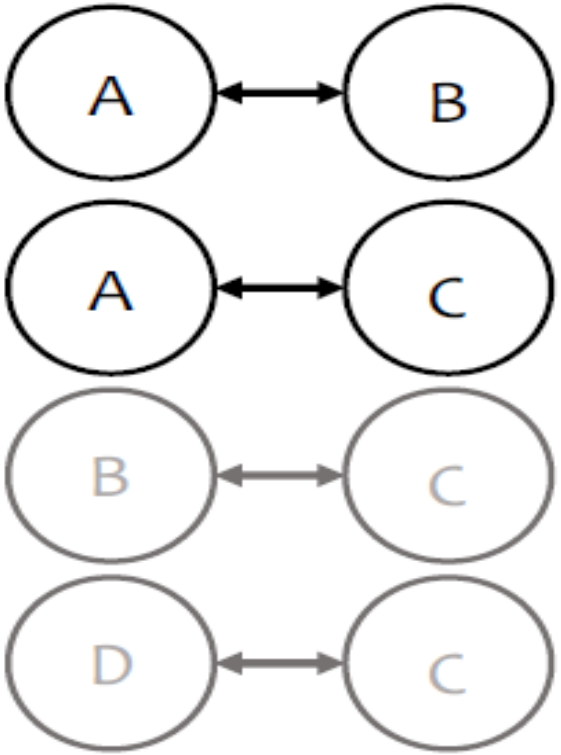


Triplets

Gather-Apply-Scatter on GraphX

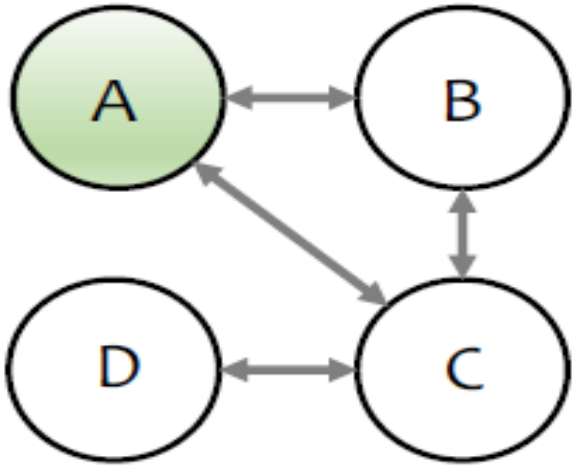


Gather at A

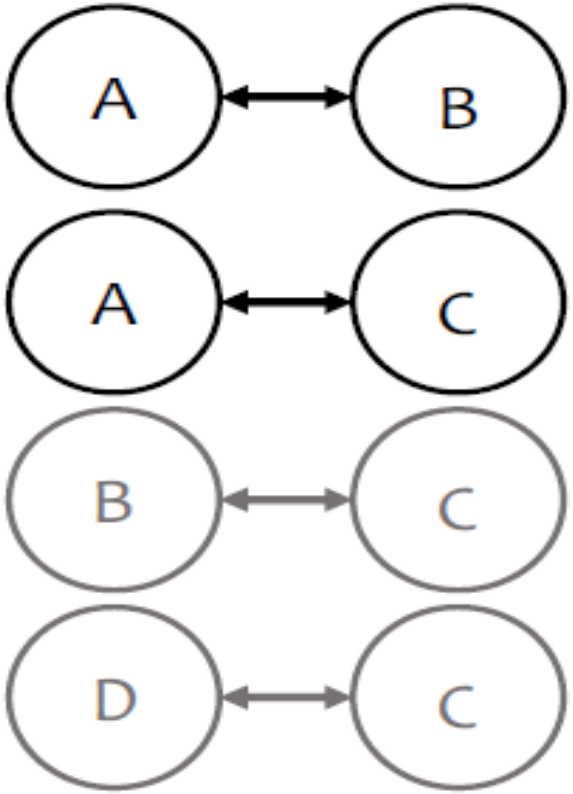


Group-By A

Gather-Apply-Scatter on GraphX

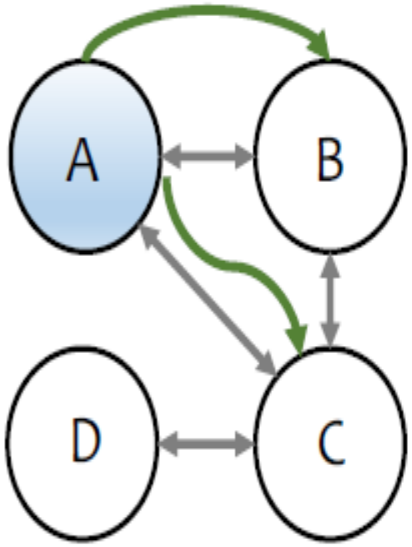


Apply

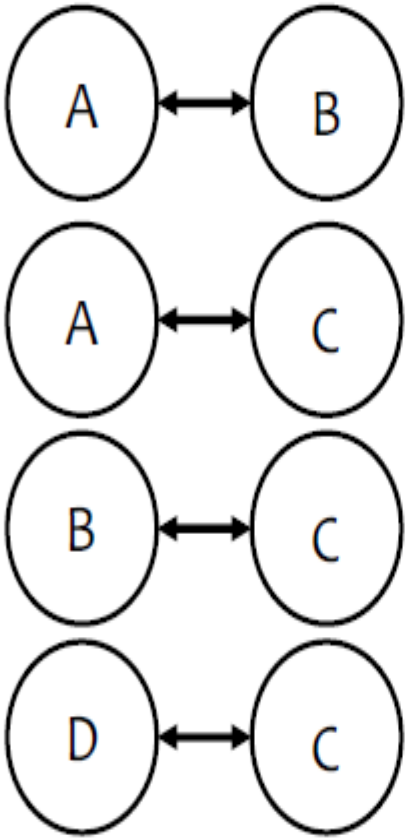


Map

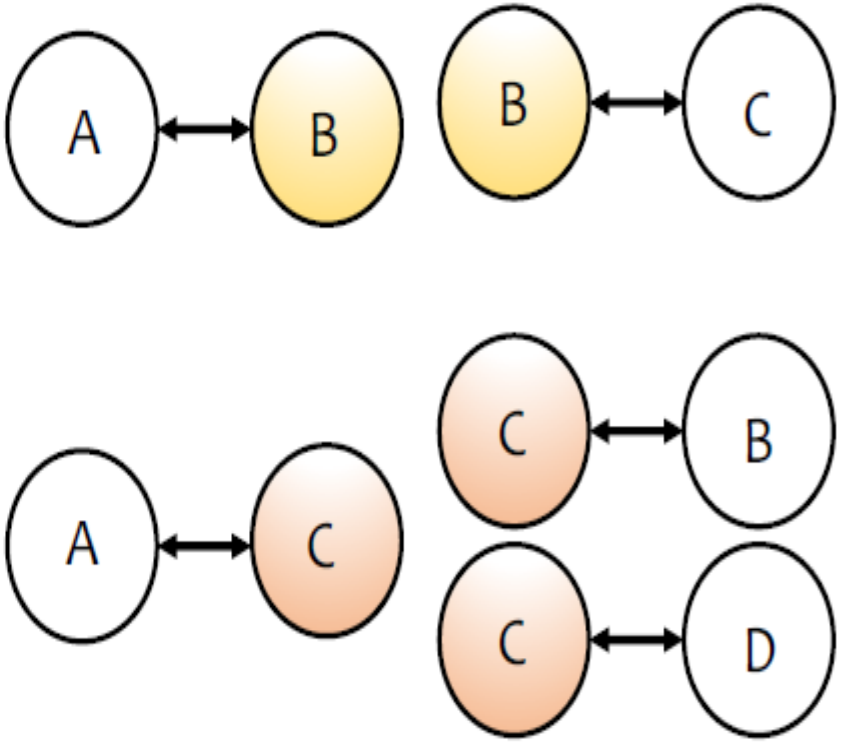
Gather-Apply-Scatter on GraphX



Scatter

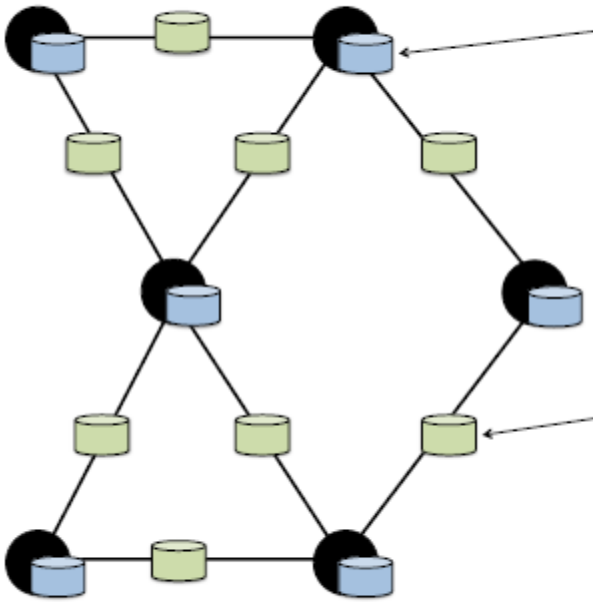


Triplets



Join

Graphs Property



Vertex Property:

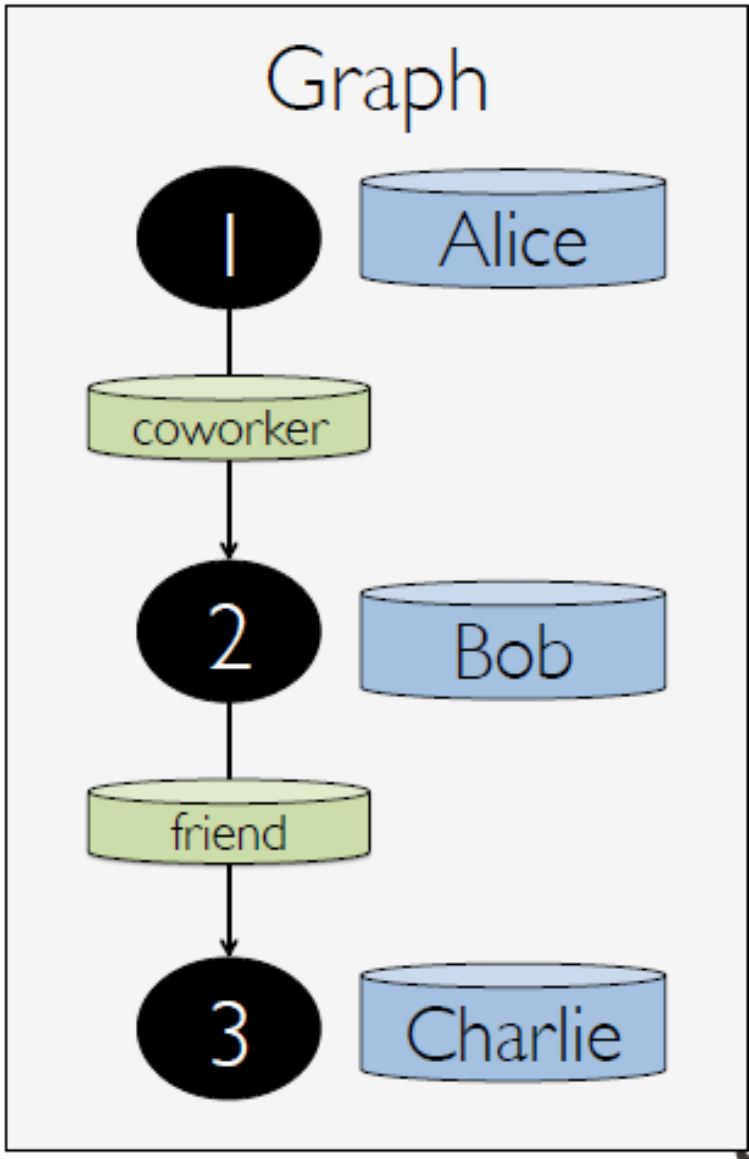
- User Profile
- Current PageRank Value

Edge Property:

- Weights
- Relationships
- Timestamps

Creating a Graph (Scala)

```
type VertexId = Long

val vertices: RDD[(VertexId, String)] =
  sc.parallelize(List(
    (1L, “Alice”),
    (2L, “Bob”),
    (3L, “Charlie”)))

class Edge[ED](
  val srcId: VertexId,
  val dstId: VertexId,
  val attr: ED)

val edges: RDD[Edge[String]] =
  sc.parallelize(List(
    Edge(1L, 2L, “coworker”),
    Edge(2L, 3L, “friend”)))

val graph = Graph(vertices, edges)
```



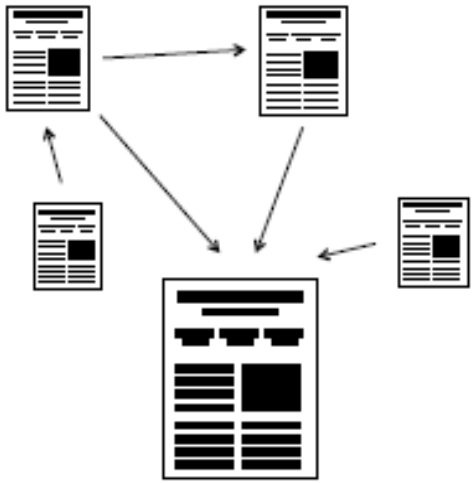
Graph Operations (Scala)

```
class Graph[VD, ED] {
    // Table Views -----------------------
    def vertices: RDD[(VertexId, VD)]
    def edges: RDD[Edge[ED]]
    def triplets: RDD[EdgeTriplet[VD, ED]]
    // Transformations ------------------------------------
    def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
    def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD2, ED]
    def reverse: Graph[VD, ED]
    def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
    // Joins ----------------------------------------
    def outerJoinVertices[U, VD2]
        (tbl: RDD[(VertexId, U)])
        (f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
    // Computation ----------------------------------
    def aggregateMessages[A](
        sendMsg: EdgeContext[VD, ED, A] => Unit,
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]
```

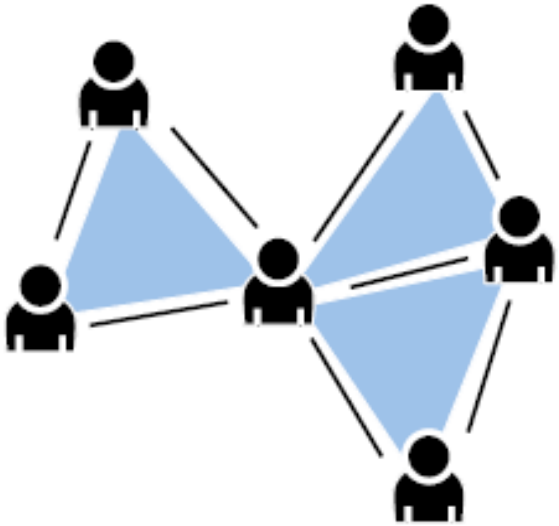
Built-in Algorithms (Scala)

```
    // Continued from previous slide
    def pageRank(tol: Double): Graph[Double, Double]
    def triangleCount(): Graph[Int, ED]
    def connectedComponents(): Graph[VertexId, ED]
    // ...and more: org.apache.spark.graphx.lib
}
```

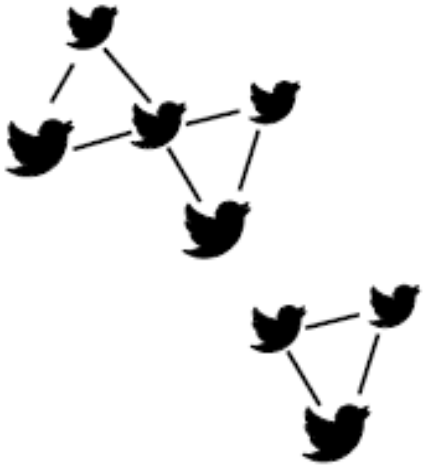
PageRank



Triangle Count

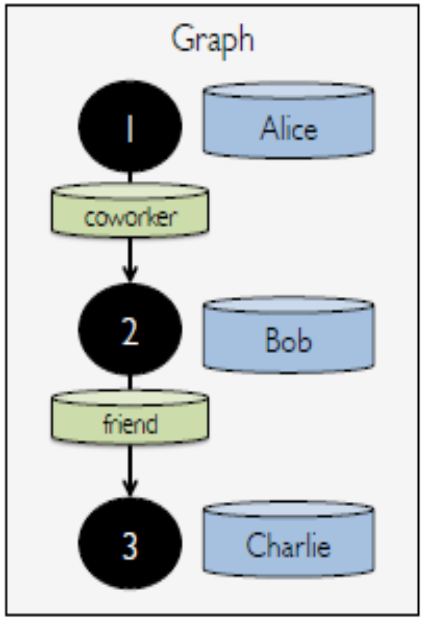


Connected Components



The triplets view

```
class Graph[VD, ED] {
  def triplets: RDD[EdgeTriplet[VD, ED]]
}

class EdgeTriplet[VD, ED](
  val srcId: VertexId, val dstId: VertexId, val attr: ED,
  val srcAttr: VD, val dstAttr: VD)
```



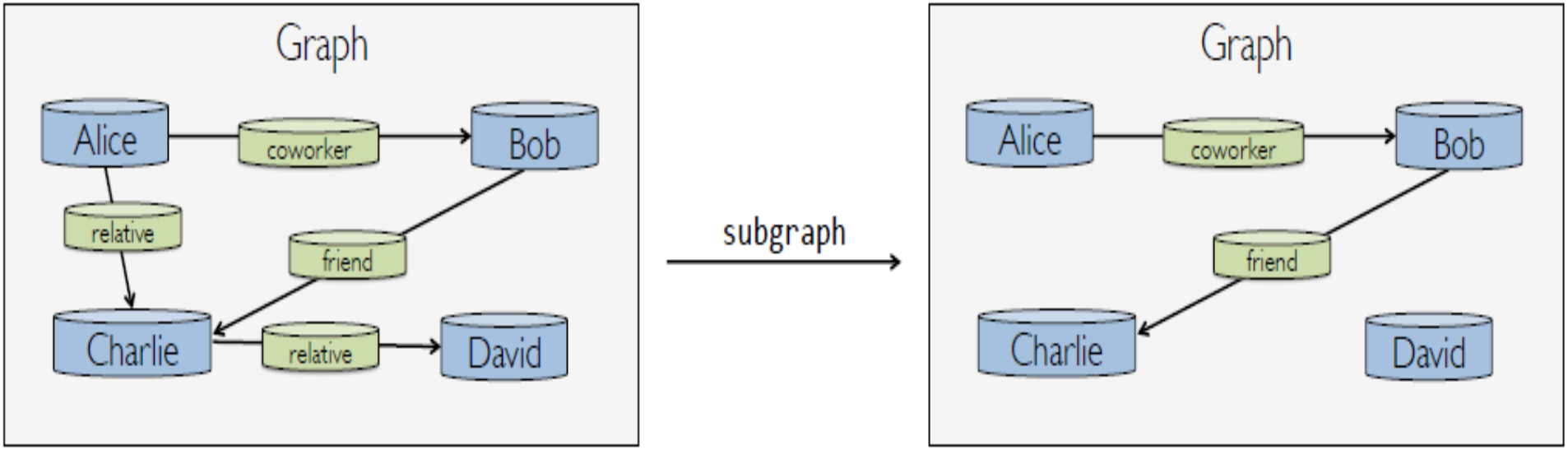
triplets

RDD

srcAttr	dstAttr	attr
Alice	coworker	Bob
Bob	friend	Charlie

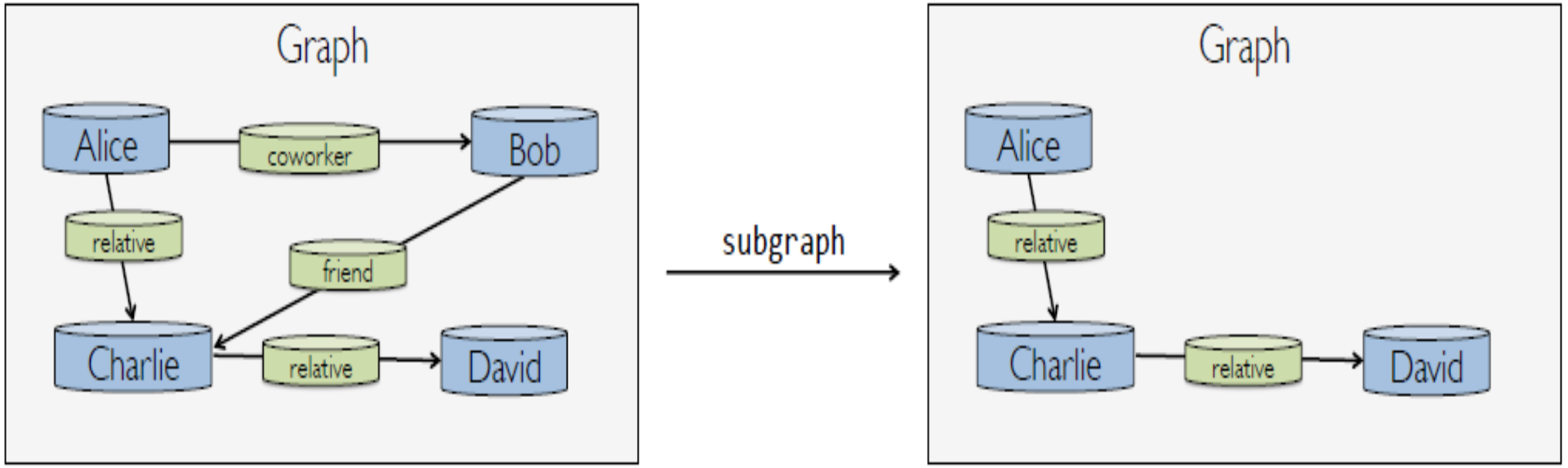
The subgraph transformation

```
class Graph[VD, ED] {
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(epred = (edge) => edge.attr != "relative")
```



The subgraph transformation

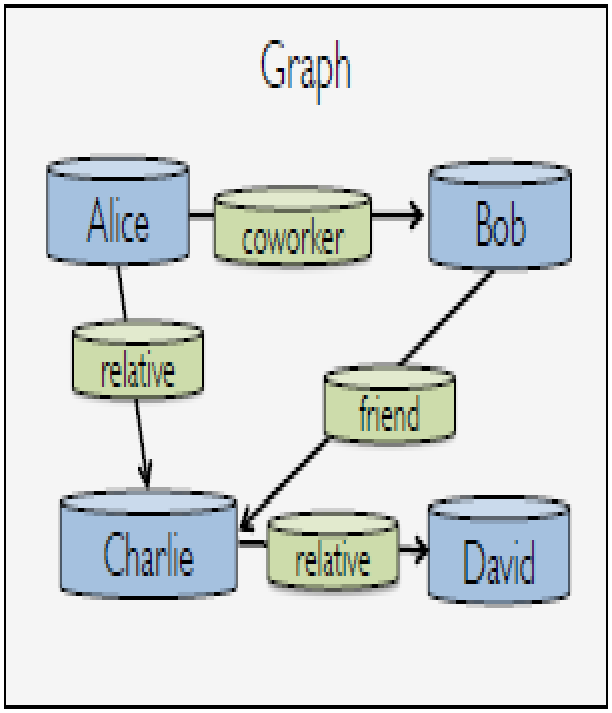
```
class Graph[VD, ED] {
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
}

graph.subgraph(vpred = (id, name) => name != "Bob")
```



Computation with aggregateMessages

```
class Graph[VD, ED] {
  def aggregateMessages[A](
    sendMsg: EdgeContext[VD, ED, A] => Unit,
    mergeMsg: (A, A) => A): RDD[(VertexId, A)]
}

class EdgeContext[VD, ED, A](
    val srcId: VertexId, val dstId: VertexId, val attr: ED,
    val srcAttr: VD, val dstAttr: VD) {
  def sendToSrc(msg: A)
  def sendToDst(msg: A)
}

graph.aggregateMessages(
  ctx => {
    ctx.sendToSrc(1)
    ctx.sendToDst(1)
  },
  _ + _)
```

Computation with aggregateMessages

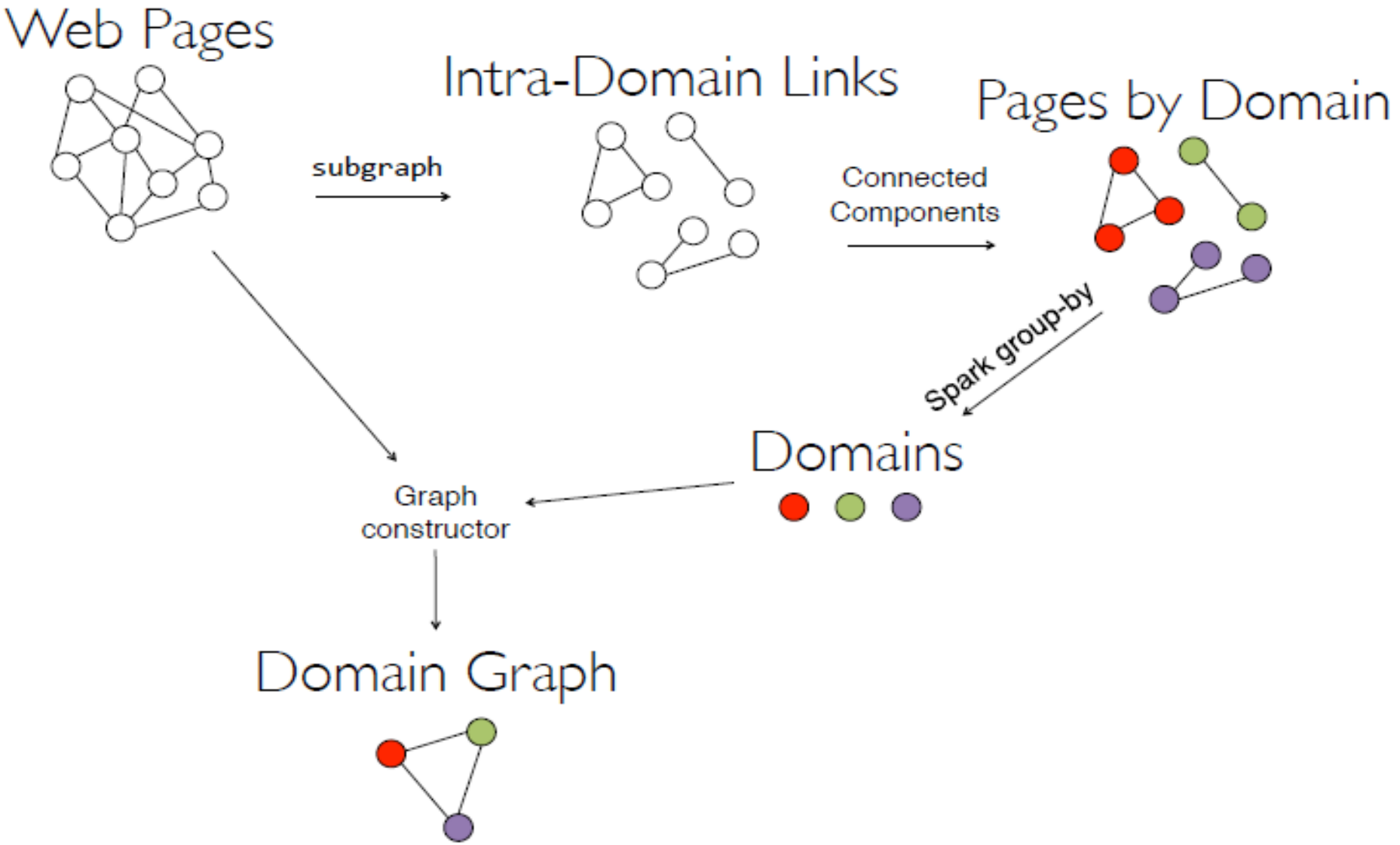


aggregateMessages

RDD

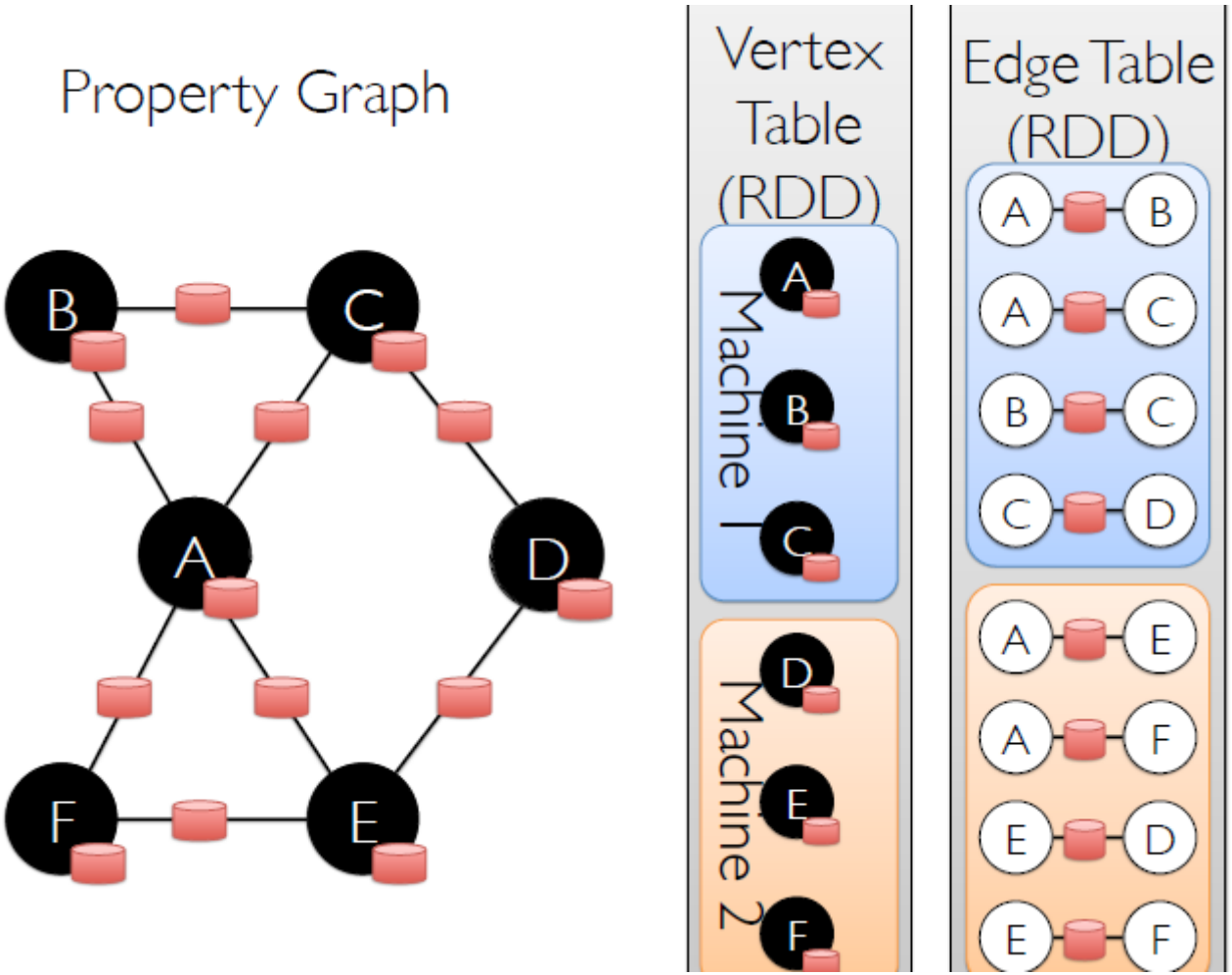
vertex id	degree
Alice	2
Bob	2
Charlie	3
David	1

Example: Graph Coarsening



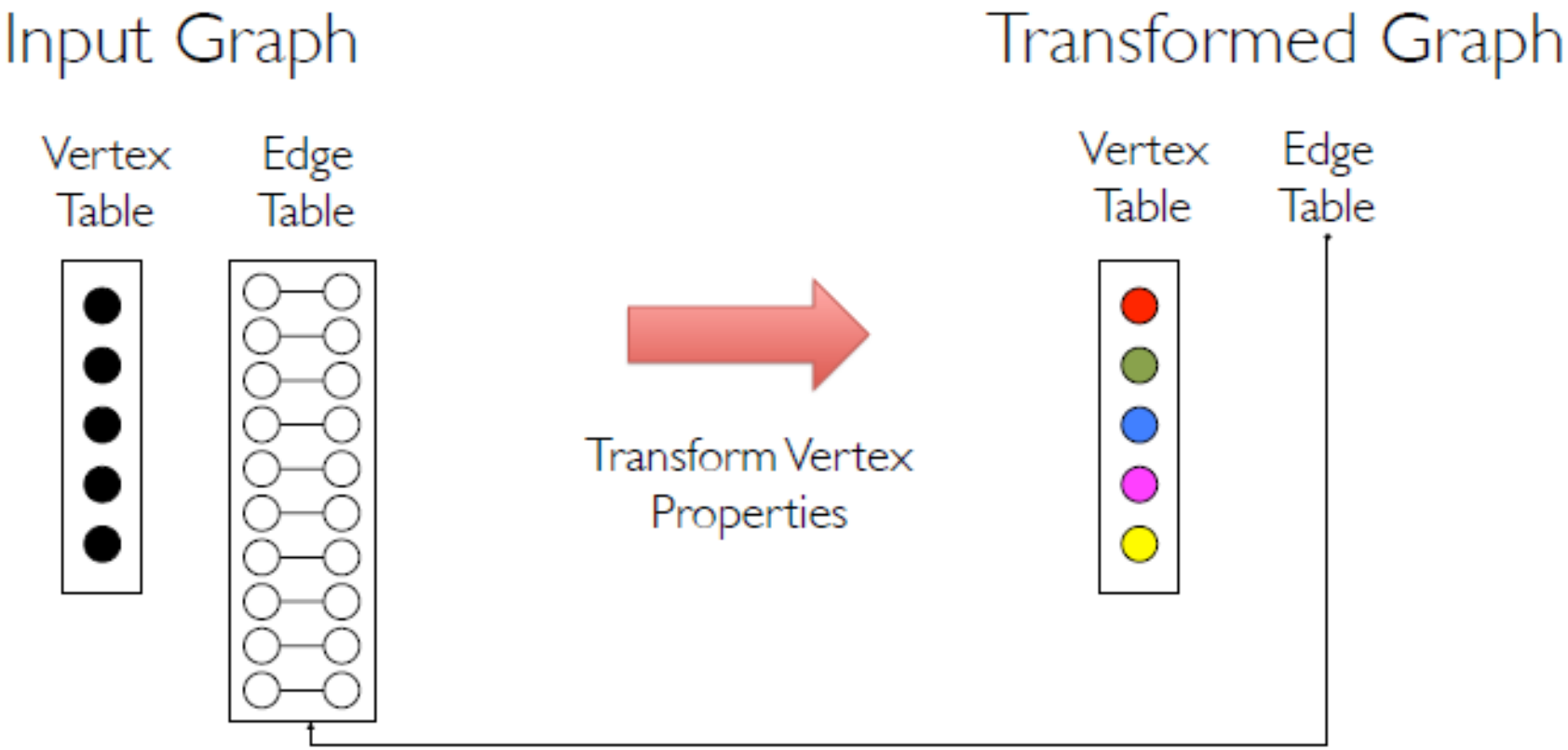
How GraphX Works

Storing Graphs as Tables

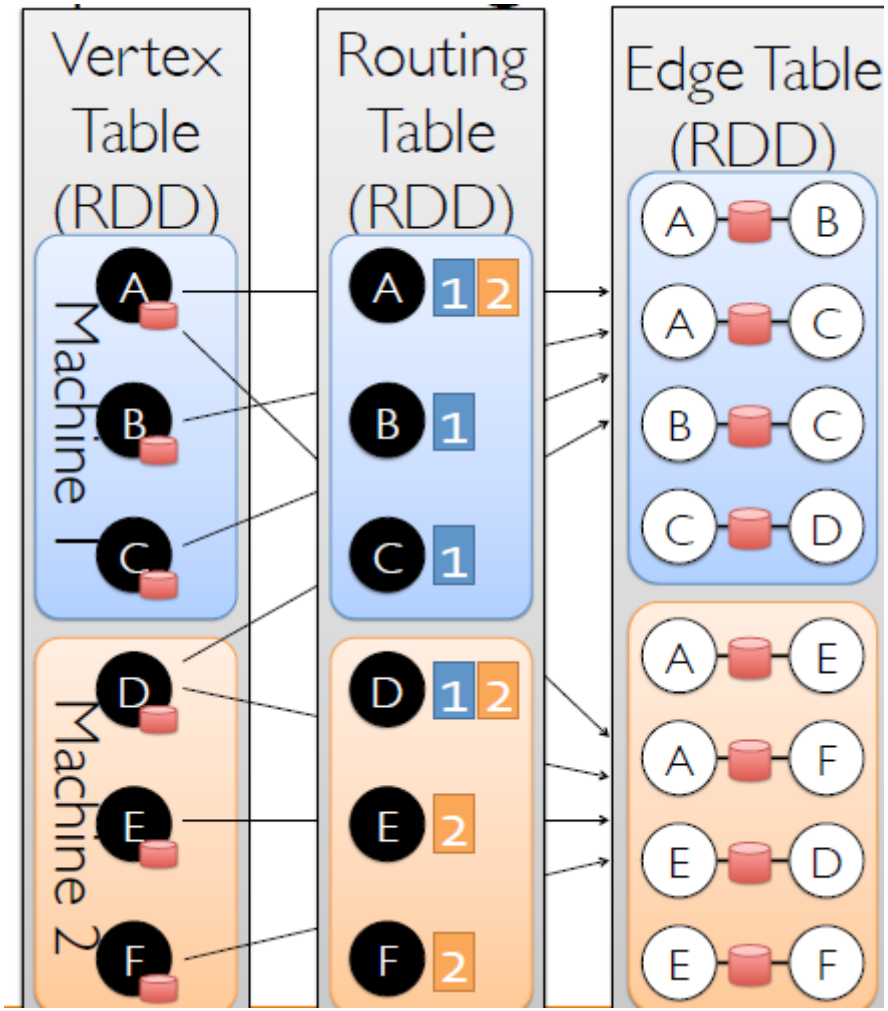


Simple Operations

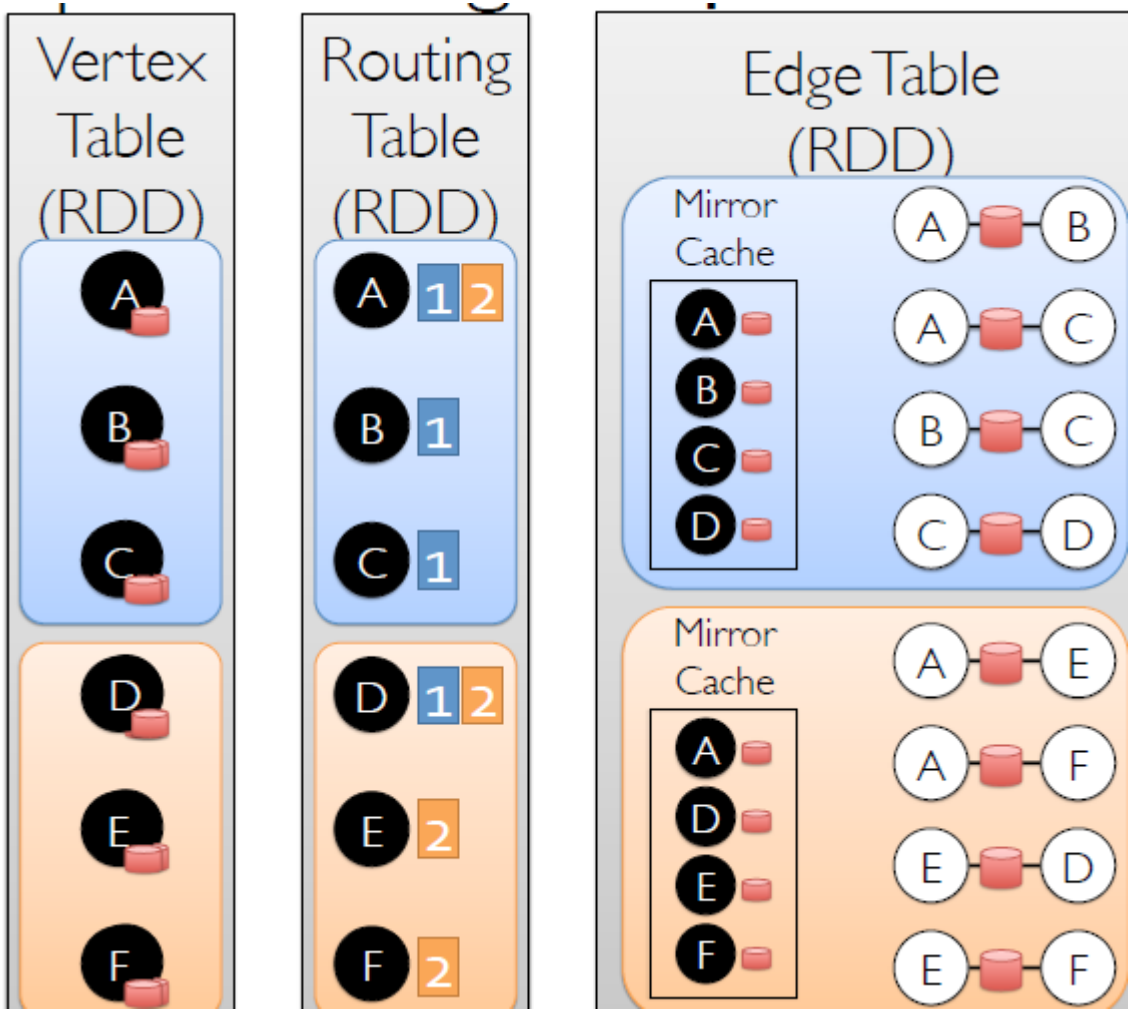
Reuse vertices or edges across multiple graphs



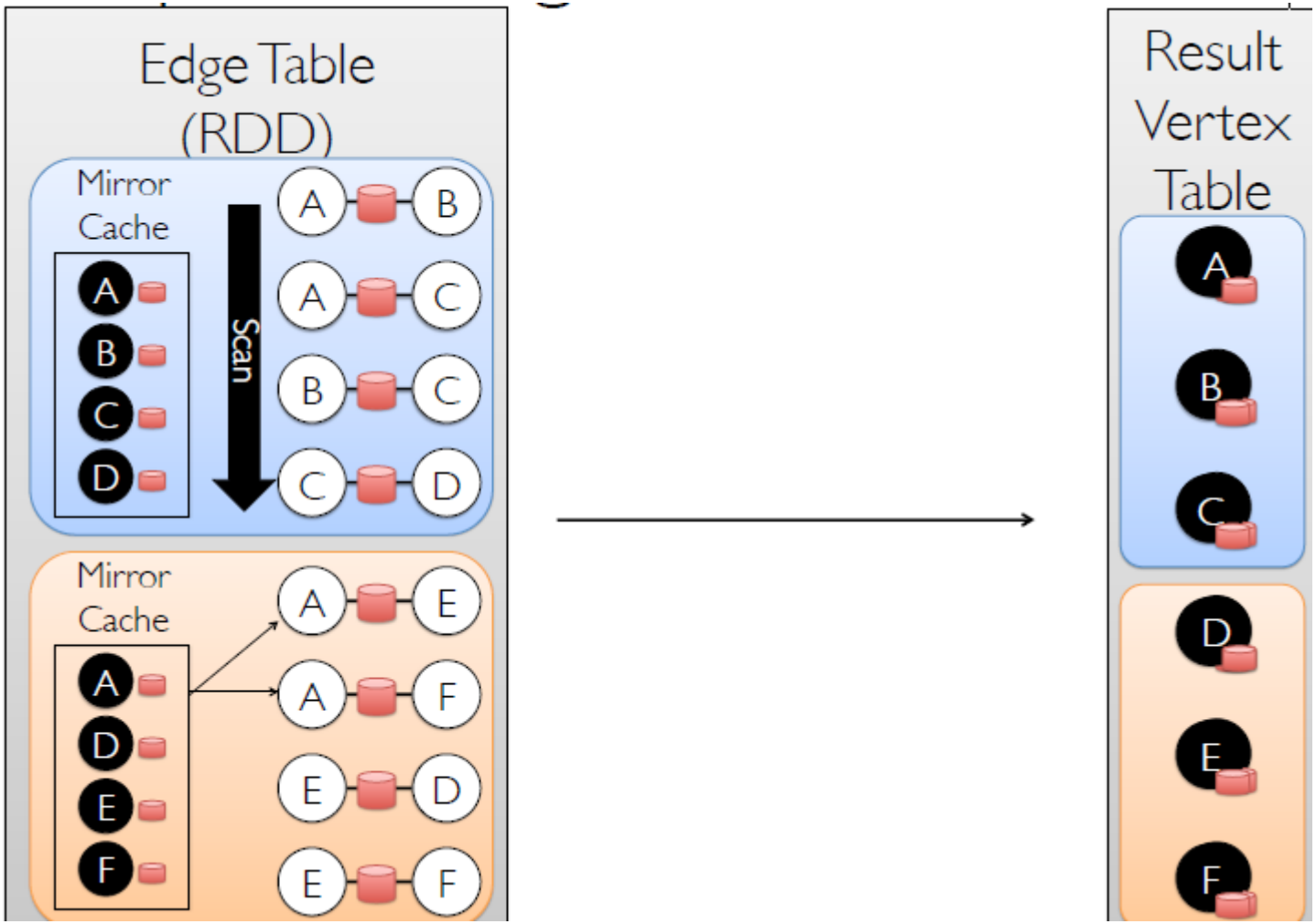
Implementing triplets



Implementing triplets



Implementing aggregateMessages



Future of GraphX

1. Language support

a) Java API

b) Python API: collaborating with Intel, SPARK-3789

2. More algorithms

a) LDA (topic modeling)

b) Correlation clustering

3. Research

a) Local graphs

b) Streaming/time-varying graphs

c) Graph database–like queries

Other Spark Applications

i. Twitter spam classification

ii. EM algorithm for traffic prediction

iii. K-means clustering

iv. Alternating Least Squares matrix factorization

v. In-memory OLAP aggregation on Hive data

vi. SQL on Spark

Thank You!

References

https://www.crayondata.com/blog/top-big-data-technologies-used-store-analyse-data

https://www.guru99.com/cassandra-tutorial.html

- Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica

"Spark: Cluster Computing with Working Sets"

References

- Matei Zaharia, Mosharaf Chowdhury et al.

"Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing"

https://spark.apache.org/

- Jeffrey Dean and Sanjay Ghemawat,

"MapReduce: Simplified Data Processing on Large Clusters"

http://labs.google.com/papers/mapreduce.html