Big Data Analytics Hadoop and Spark

Shelly Garion, Ph.D.

IBM Research Haifa

What is Big Data?

What is Big Data?

- **Big data** usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time.
- **Big data** "size" is a constantly moving target, as of 2012 ranging from a few dozen terabytes to many petabytes of data.
- **Big data** is a set of techniques and technologies that require new forms of integration to uncover large hidden values from large datasets that are diverse, complex, and of a massive scale.

(From Wikipedia)

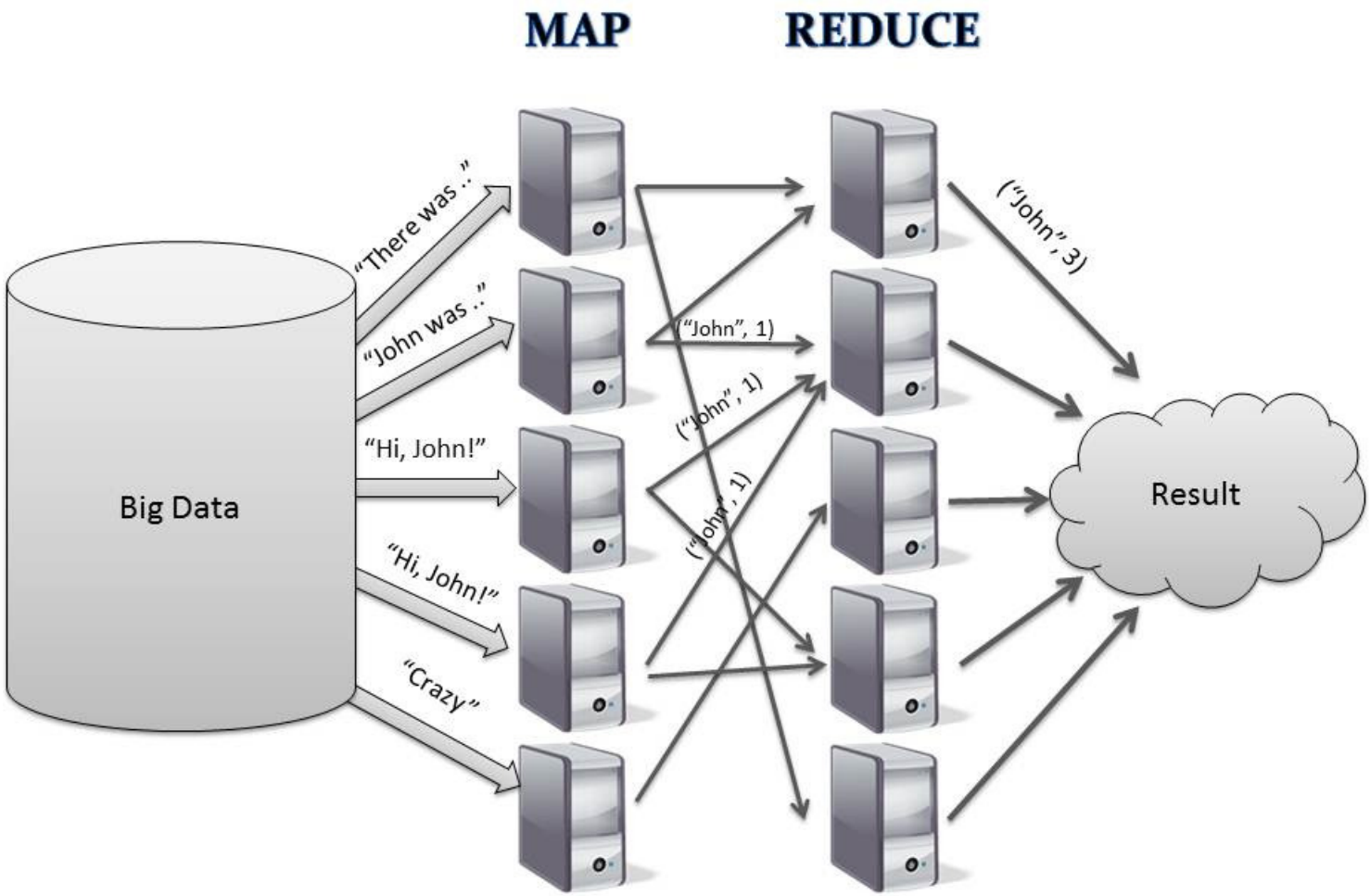
What is Big Data?

- Volume.
- Value.
- Variety.
- Velocity - the speed of generation of data.
- Variability - the inconsistency which can be shown by the data at times.
- Veracity - the quality of the data being captured can vary greatly.
- Complexity - data management can become a very complex process, especially when large volumes of data come from multiple sources.

(From Wikipedia)

How to analyze Big Data?

Map/Reduce


MAP
REDUCE
Big Data
"There was .."
"John was .."
"Hi, John!"
"Hi, John!"
"Crazy"
("John", 1)
("John", 1)
("John", 1)
("John", 3)
Result

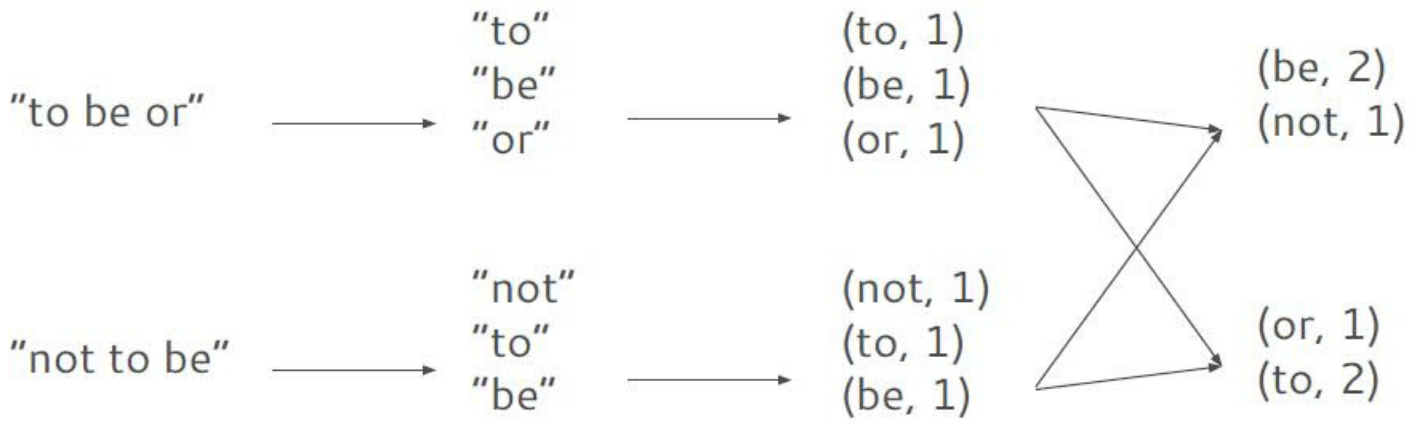
Map/Reduce

- MapReduce is a framework for processing **parallelizable** problems across huge datasets using a large number of computers (nodes), collectively referred to as a **cluster**.
- **Map step:** Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.
- **Shuffle step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- **Reduce step:** Worker nodes now process each group of output data, per key, in parallel.

(From Wikipedia)

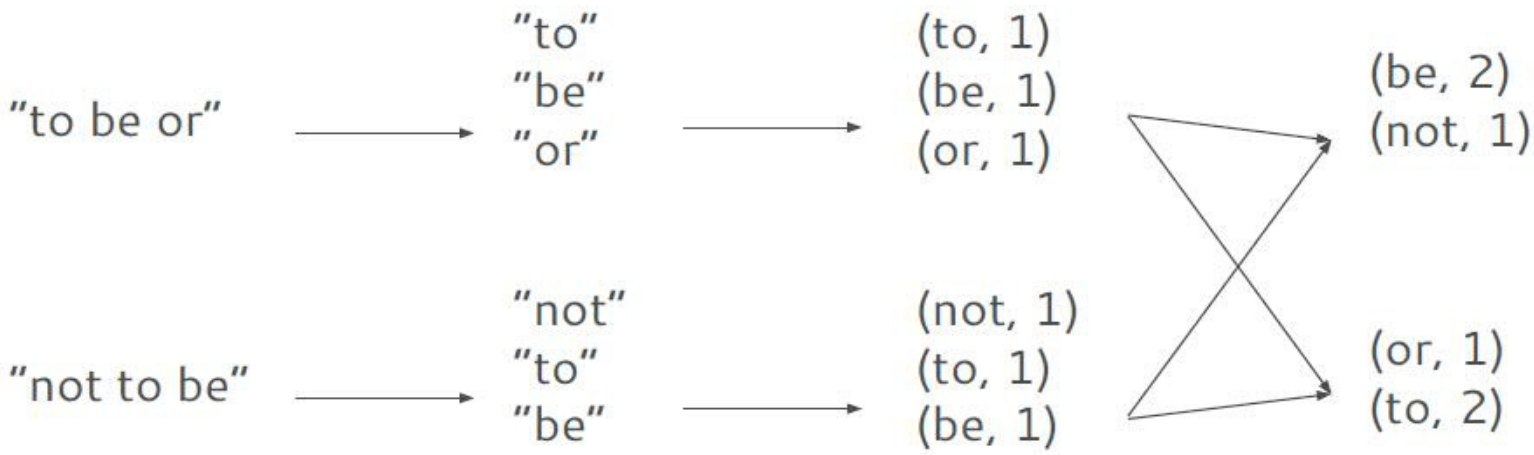
Basic Example: Word Count (Spark & Python)

```
>lines = sc.textFile("hamlet.txt")
>counts = lines.flatMap(lambda line: line.split(" "))
               .map(lambda word => (word, 1))
               .reduceByKey(lambda x, y: x + y)
```



Basic Example: Word Count (Spark & Scala)

```
>val lines = sc.textFile("hamlet.txt")
>val counts = lines.flatMap(_.split(" "))
                  .map((_, 1))
                  .reduceByKey(_ + _)
```



Map/Reduce – Parallel Computing

- No dependency among data
- Data can be split into equal-size chunks
- Each process can work on a chunk
- Master/worker approach

– Master:

- Initializes array and splits it according to # of workers
- Sends each worker the sub-array
- Receives the results from each worker

– Worker:

- Receives a sub-array from master
- Perfoms processing
- Sends results to master

MapReduce: what happens in between?

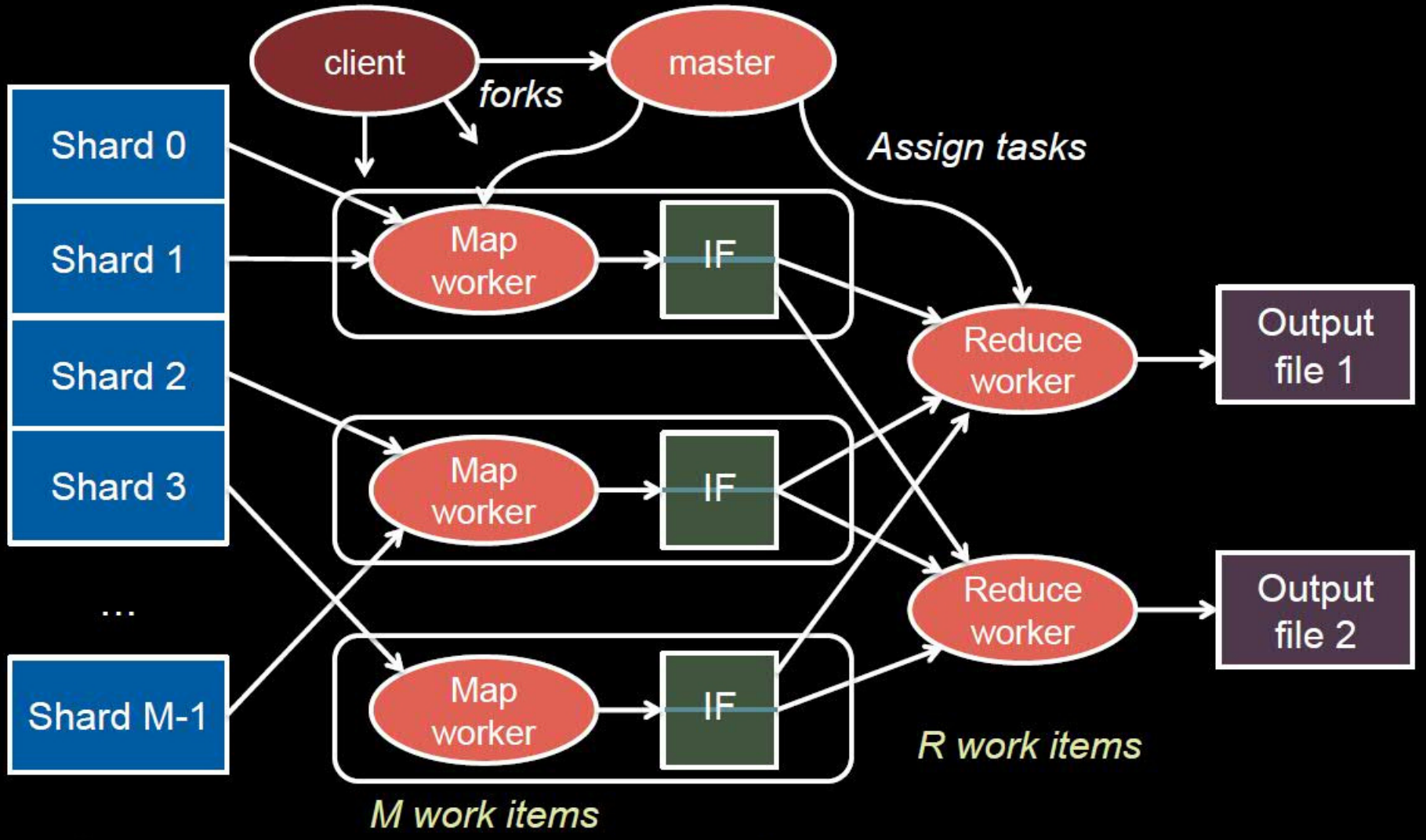
- **Map**
  - Grab the relevant data from the source (parse into key, value)
  - Write it to an intermediate file
- **Partition**
  - Partitioning: identify which of *R* reducers will handle which keys
  - Map partitions data to target it to one of *R* Reduce workers based on a partitioning function (both *R* and partitioning funciton user defined)

Map Worker

- **Sort**
  - Fetch the relevant partition of the output from all mappers
  - Sort by keys (different mappers may have output the same key)
- **Reduce**
  - Input is the sorted output of mappers
  - Call the user *Reduce* function per key with the list of values for that key to aggregate the results

Reduce Worker

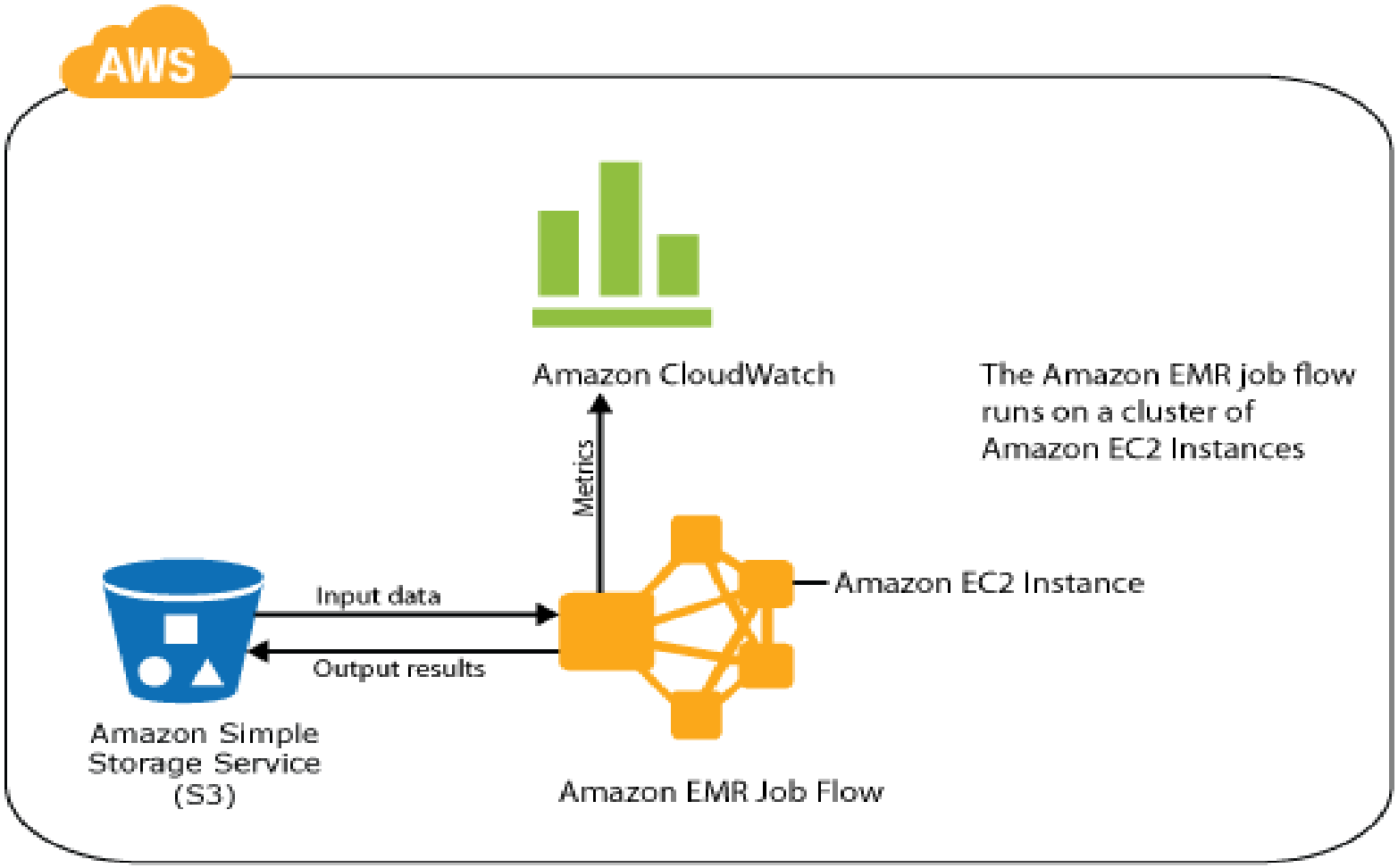
MapReduce: the complete picture



Map/Reduce History

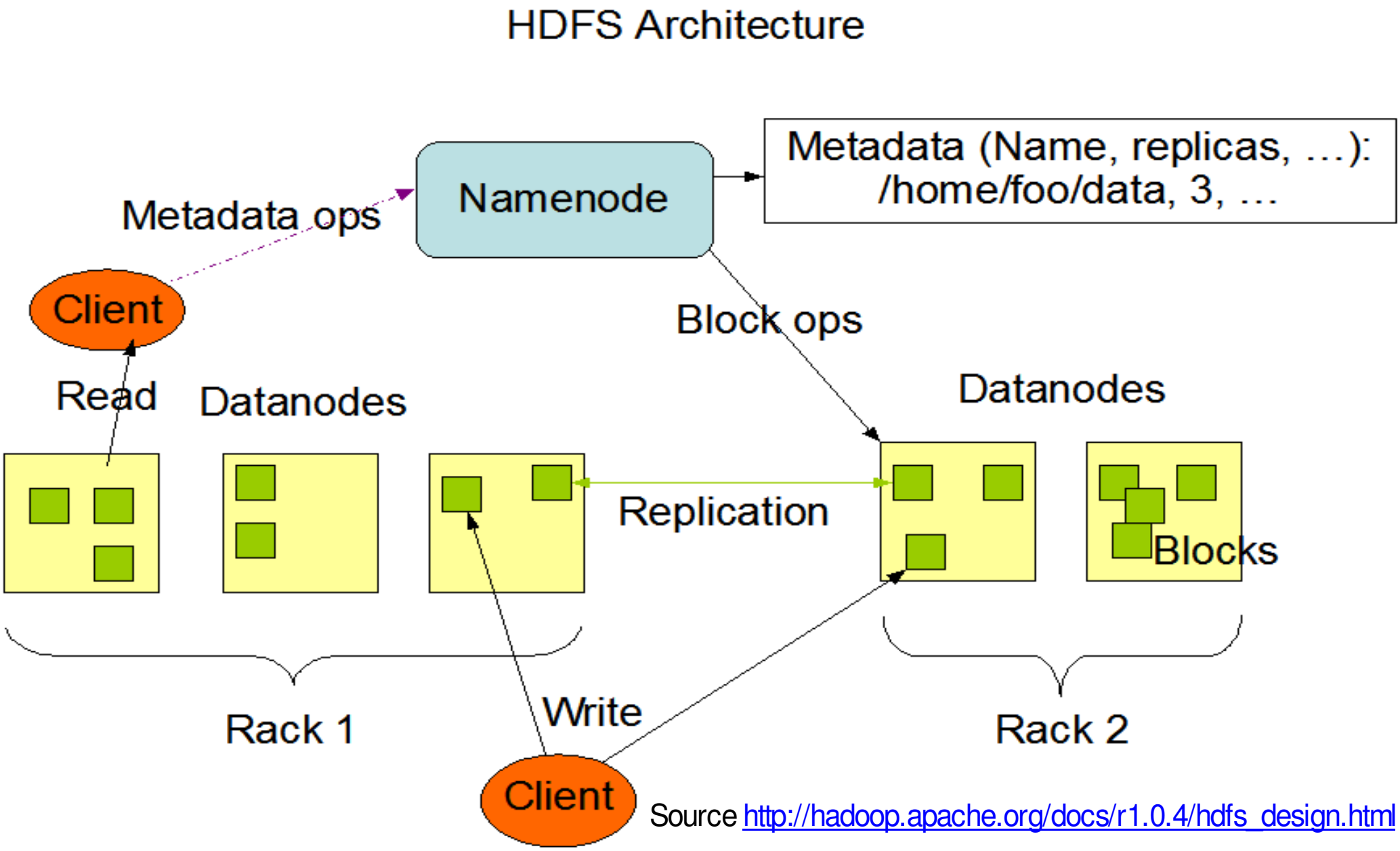
- **Invented by Google:**
  - Inspired by functional programming languages map and reduce functions
  - Seminal paper: Dean, Jeffrey & Ghemawat, Sanjay (OSDI 2004), "MapReduce: Simplified Data Processing on Large Clusters"
  - Used at Google to completely regenerate Google's index of the World Wide Web.
  - It replaced the old ad-hoc programs that updated the index and ran the various analysis
- **Uses:**
  - distributed pattern-based searching, distributed sorting, web link-graph reversal, term-vector per host, web access log stats, inverted index construction, document clustering, machine learning, statistical machine translation
- **Apache Hadoop:** Open source implementation matches Google's specifications
- **Amazon Elastic MapReduce** running on Amazon EC2

Amazon Elastic MapReduce



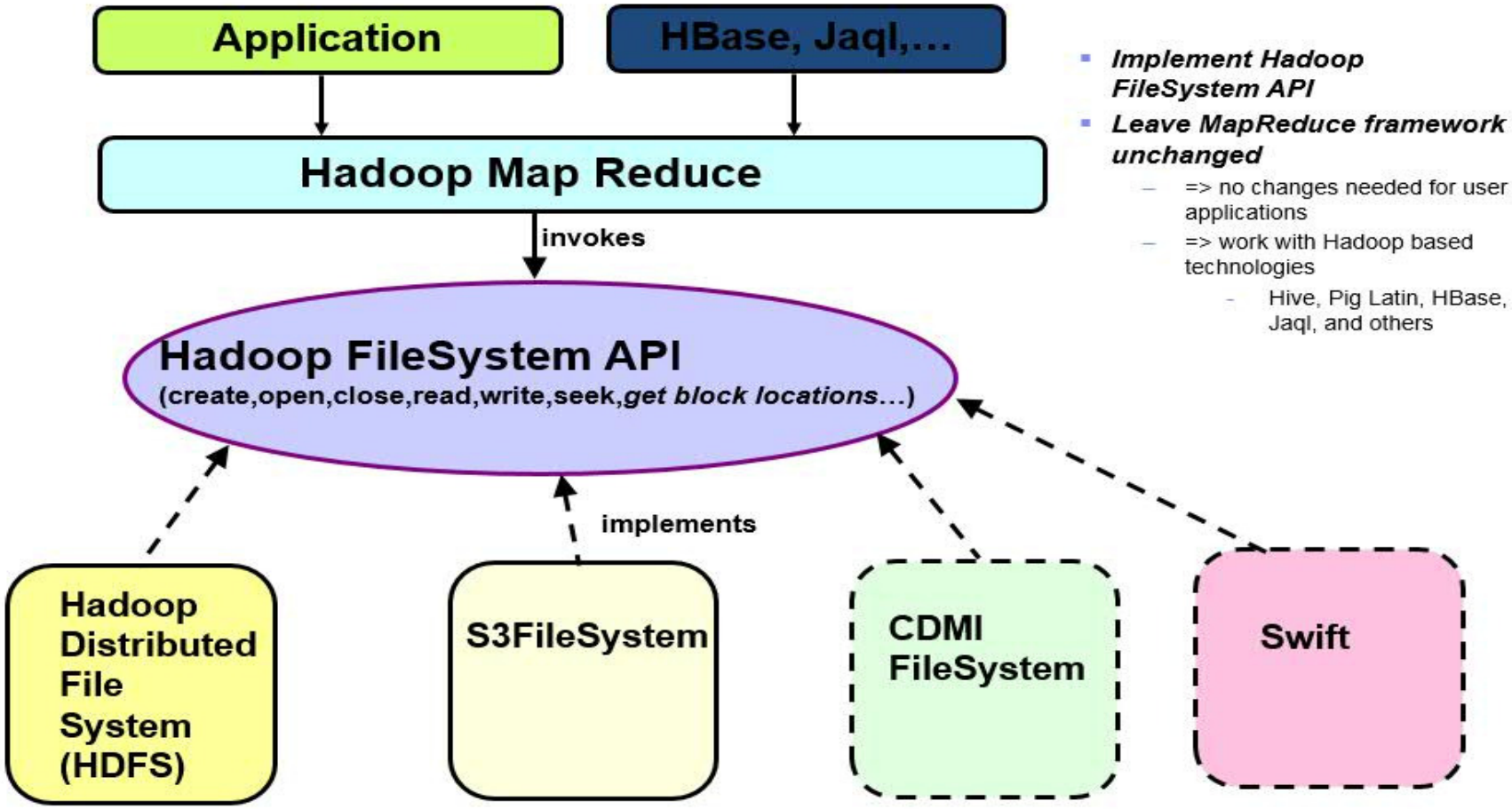
Source: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html

HDFS Architecture



Source http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html

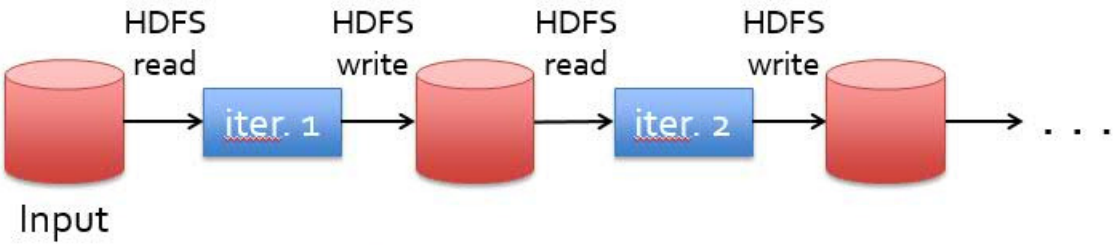
Hadoop & Object Storage

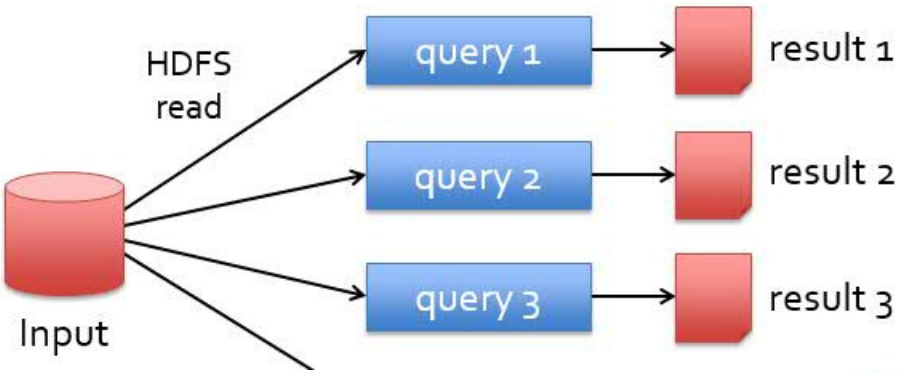


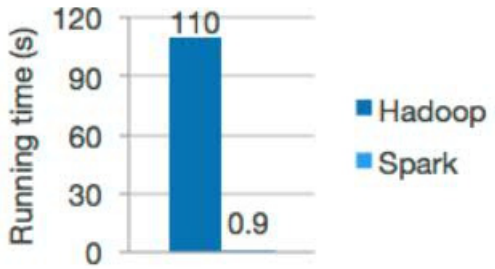
Apache Spark

- **Apache Spark**™ is a fast and general open-source engine for large-scale data processing.
- Includes the following libraries: SPARK SQL, SPARK Streaming, MLlib (Machine Learning) and GraphX (graph processing).
- Spark capable to run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- Spark can run on Apache Mesos or Hadoop 2's YARN cluster manager, and can read any existing Hadoop data.
- Written in **Scala** language (a 'Java' like, executed in Java VM)
- Apache Spark is built by a wide set of developers from over 50 companies. Since the project started in 2009, more than 400 developers have contributed to Spark.

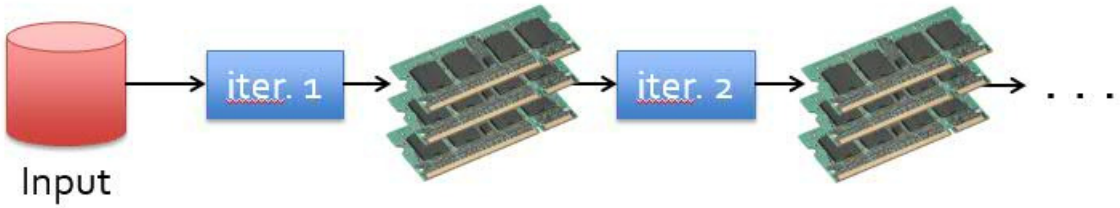
Spark vs. Hadoop

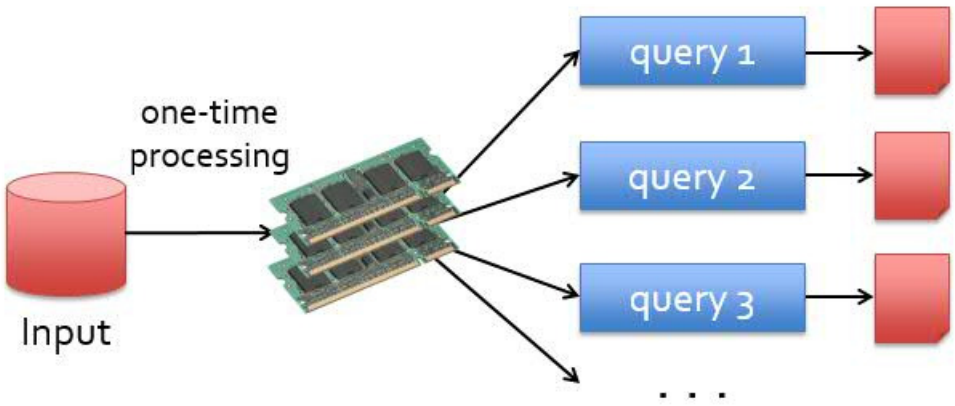






Logistic regression in Hadoop and Spark

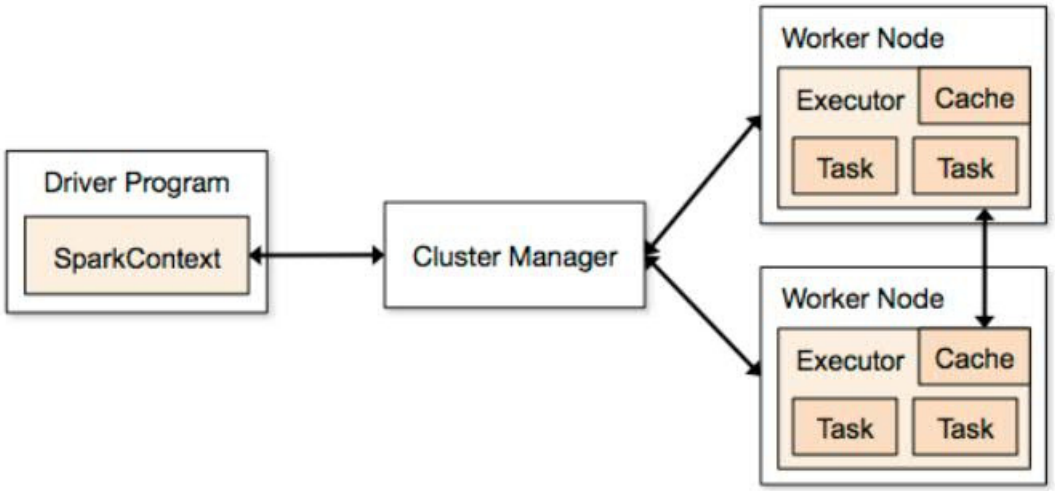




Resilient Distributed Datasets
A Fault-Tolerant Abstraction for In-Memory Cluster Computing
Matei Zaharia et al.
Usenix NSDI '12

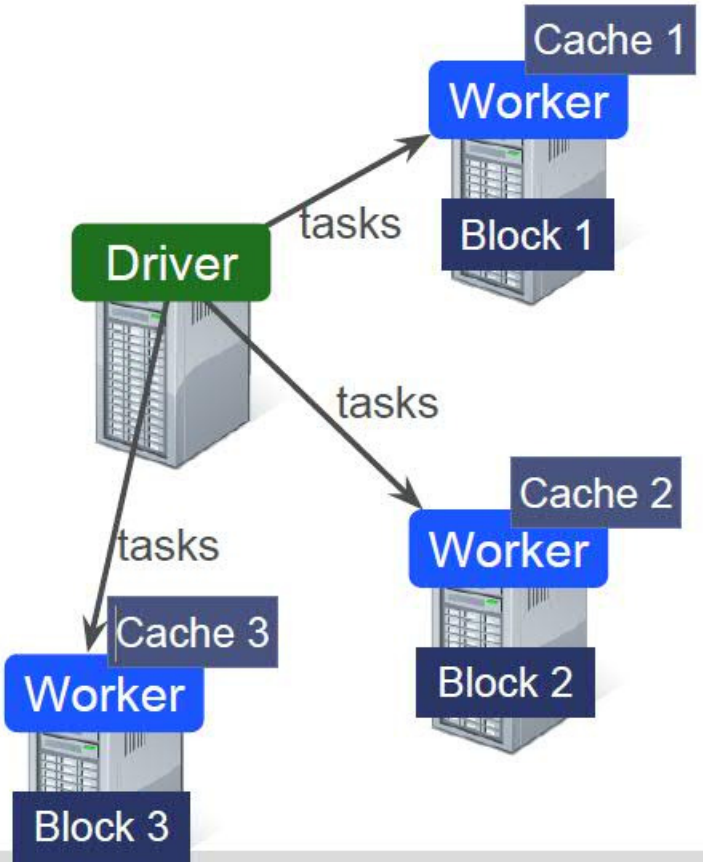
Spark Cluster

- Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext (SC) object in your main .
- SC can connect to several types of resource *cluster managers* (either Spark's own standalone cluster manager or Mesos/YARN), which allocate resources across applications.
- Once connected, Spark acquires *executors* on nodes in the cluster, which are worker processes that run computations and store data for your application. Next, it sends your application code (JAR or Python files) to the executors. Finally, SC sends *tasks* for the executors to run.



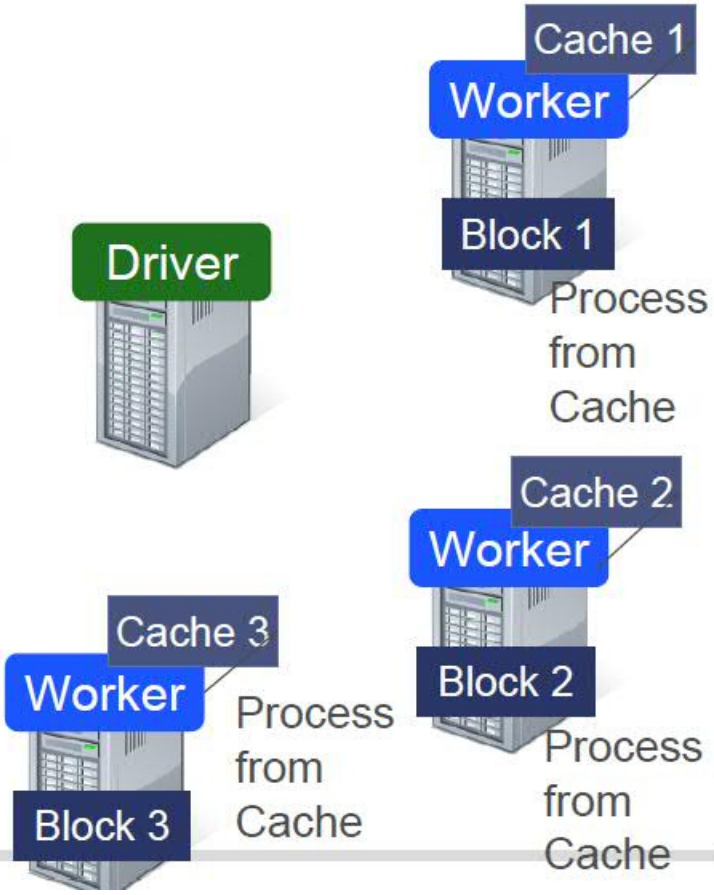
Spark Cluster Example: Log Mining

```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
messages.cache()


messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



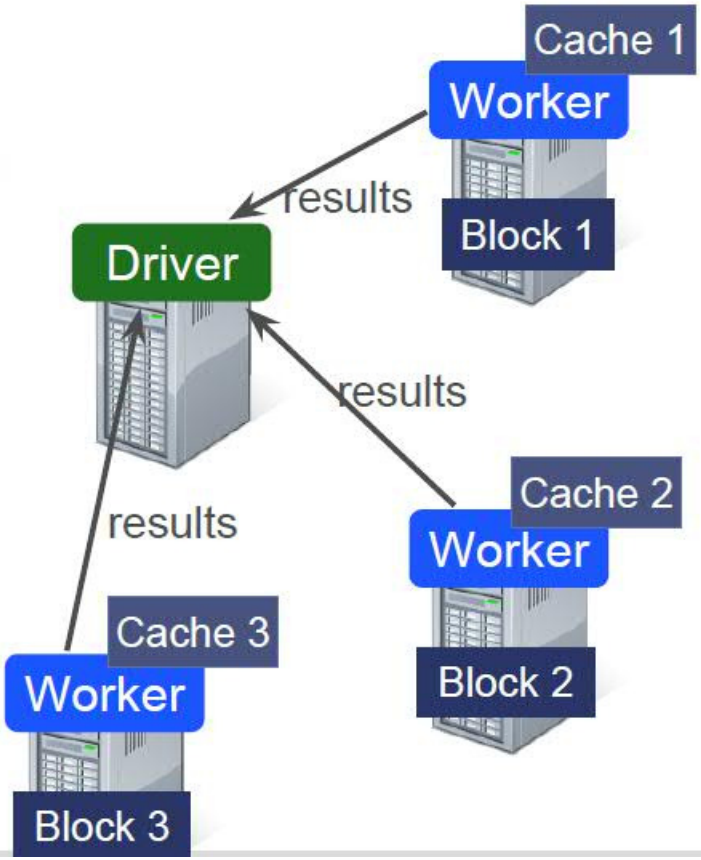
Spark Cluster Example: Log Mining

```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
messages.cache()


messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



Spark Cluster Example: Log Mining

```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
messages.cache()


messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```



Spark RDD

Write programs in terms of **transformations** on **distributed datasets**

Resilient Distributed Datasets

- Immutable, partitioned collections of objects spread across a cluster, stored in RAM or on Disk
- Built through lazy parallel transformations
- Automatically rebuilt on failure

Operations

- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)

Spark & Scala: Creating RDD

```
# Turn a Python collection into an RDD
>sc.parallelize([1, 2, 3])

# Turn a Scala collection into an RDD
>sc.parallelize(List(1, 2, 3))

# Load text file from local FS, HDFS, or S3
>sc.textFile("file.txt")
>sc.textFile("directory/*.txt")
>sc.textFile("hdfs://namenode:9000/path/file")
```

Spark & Scala: Basic Transformations

```
>val nums = sc.parallelize(List(1, 2, 3))

// Pass each element through a function
>val squares = nums.map(x: x*x)   // {1, 4, 9}

// Keep elements passing a predicate
>val even = squares.filter(x => x % 2 == 0) // {4}

// Map each element to zero or more others
>nums.flatMap(x => 0.to(x))
//=> {0, 1, 0, 1, 2, 0, 1, 2, 3}

 // Pass each partition through a function
 >val squares = nums.mapPartition(x.map(x * x))   // {1
 4, 9}
```

Spark & Scala: Basic Actions

```
>val nums = sc.parallelize(List(1, 2, 3))

// Retrieve RDD contents as a local collection
>nums.collect() //=> List(1, 2, 3)

// Return first K elements
>nums.take(2)   //=> List(1, 2)

// Count number of elements
>nums.count()   //=> 3

// Merge elements with an associative function
>nums.reduce{case (x, y) => x + y}  //=> 6

// Write elements to a text file
>nums.saveAsTextFile("hdfs://file.txt")
```

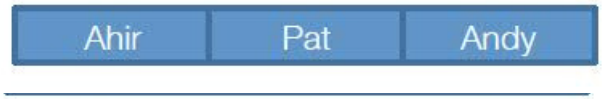
Spark & Scala: Key-Value Operations

```
>pets = sc.parallelize(
  List(("cat", 1), ("dog", 1), ("cat", 2)))
>pets.reduceByKey(_ + _)
                    //=> ((cat, 3), (dog, 1))
>pets.groupByKey() //=> {(cat, [1, 2]), (dog, [1])}
>pets.sortByKey()  //=> {(cat, 1), (cat, 2), (dog, 1)}
```

Example:
Spark Map/Reduce

Goal:

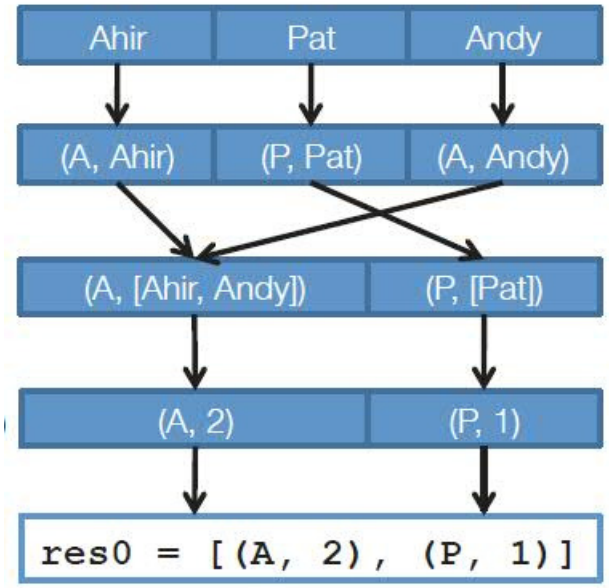
Find number of distinct names per "first letter".



Example:
Spark Map/Reduce

Goal:

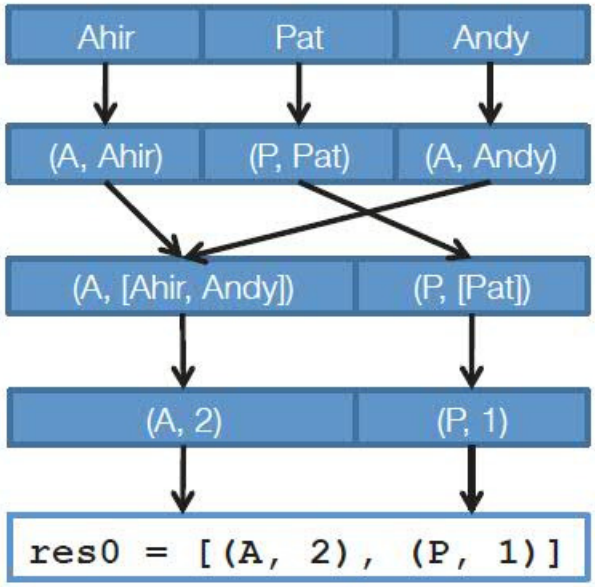
Find number of distinct names per "first letter".



Example: Spark Map/Reduce

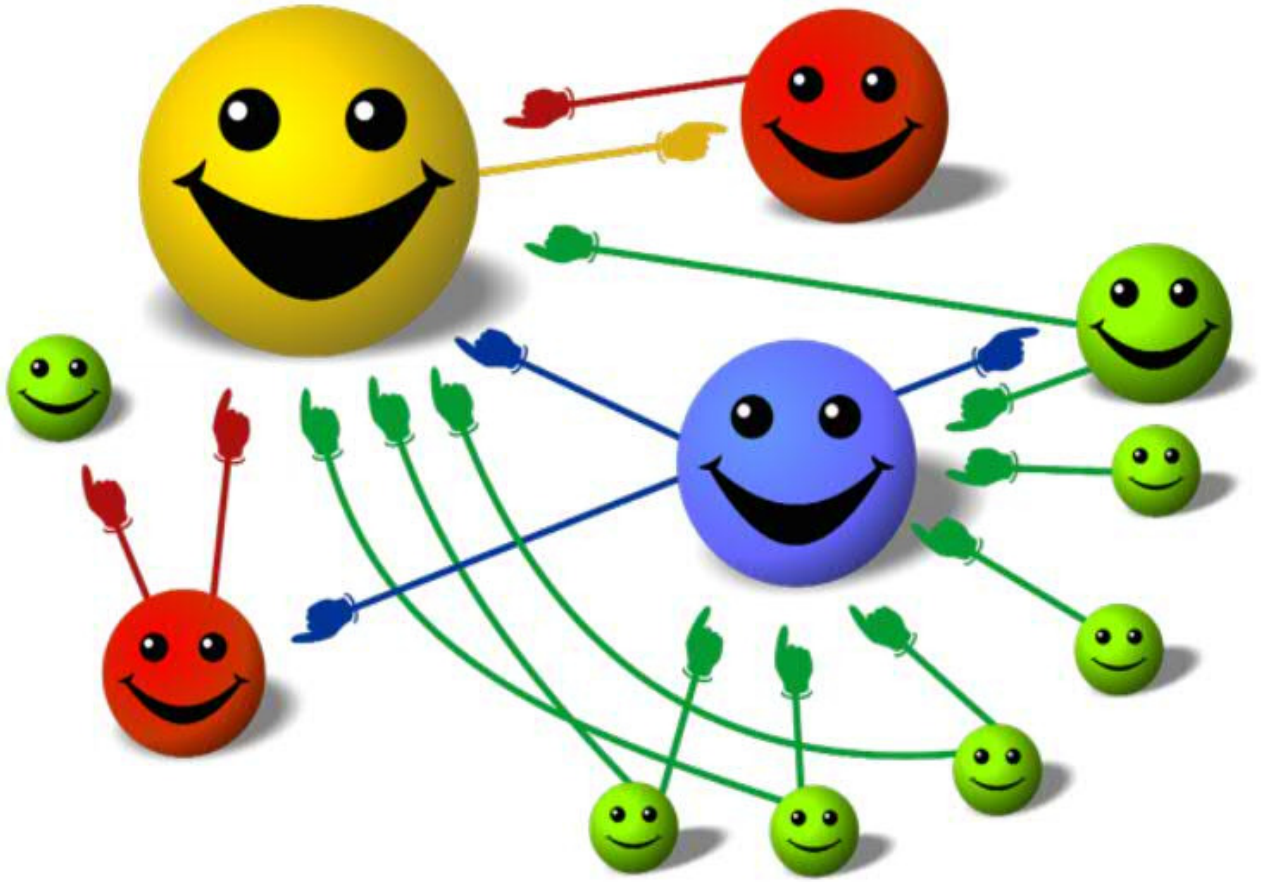
Goal: Find number of distinct names per "first letter"

```
sc.textFile("hdfs:/names")

   .map(name => (name.charAt(0), name))

   .groupByKey()

   .mapValues(names => names.toSet.size)

   .collect()
```



Example: Page Rank

Popular algorithm originally introduced by Google



Example: Page Rank

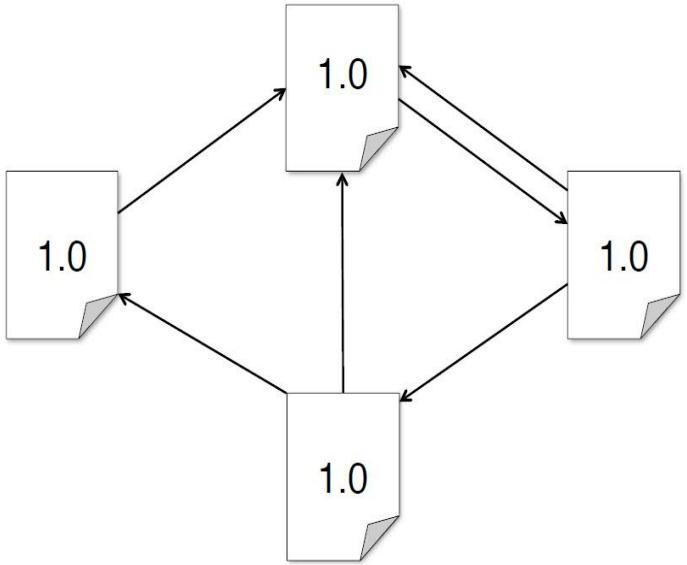
PageRank Algorithm

- Start each page with a rank of 1
- On each iteration:

A. $$contrib = \frac{curRank}{|neighbors|}$$

B. $$curRank = 0.15 + 0.85 \sum_{neighbors} contrib_i$$

PageRank Example



$$A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

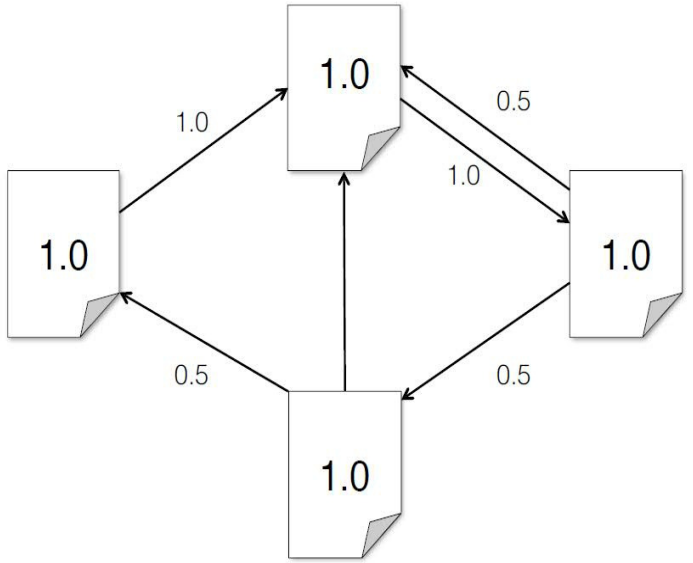
$$V = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$B = 0.85 * A$$

$$U = 0.15 * V$$

$$B*V+U = ?$$

PageRank Example



DATABRICKS

16

$$A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

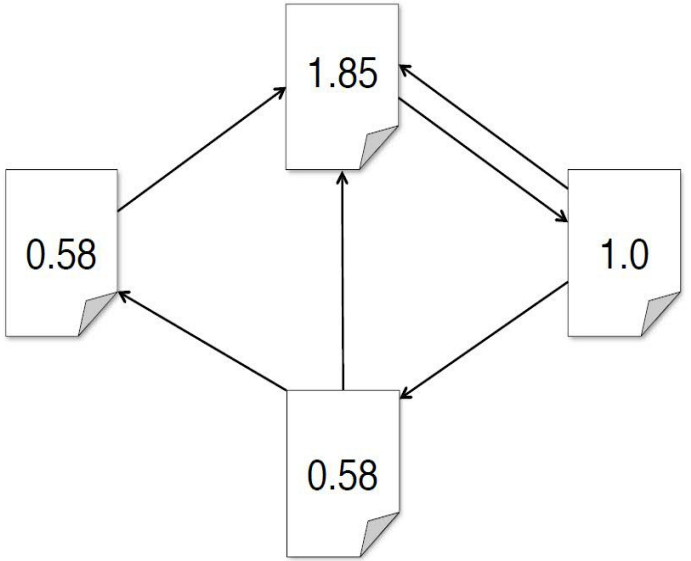
$$V = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$B = 0.85 * A$$

$$U = 0.15 * V$$

$$B*V+U = \begin{bmatrix} 1.85 \\ 1.0 \\ 0.575 \\ 0.575 \end{bmatrix}$$

PageRank Example



DATABRICKS

17

PageRank Example

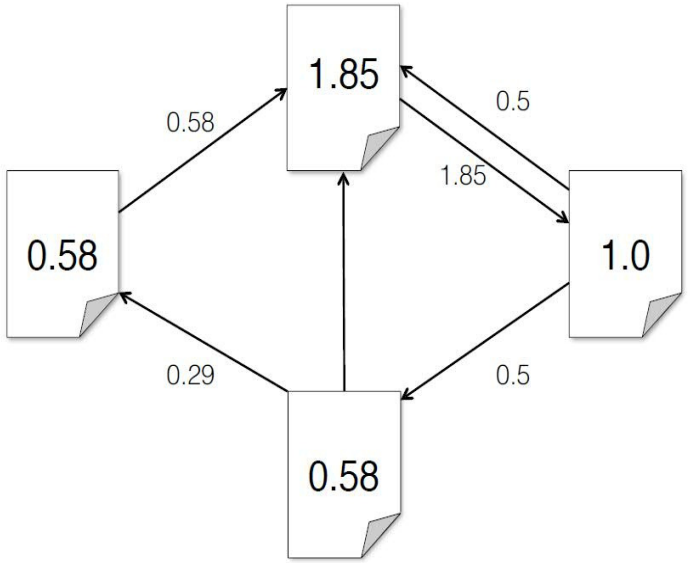
$$A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

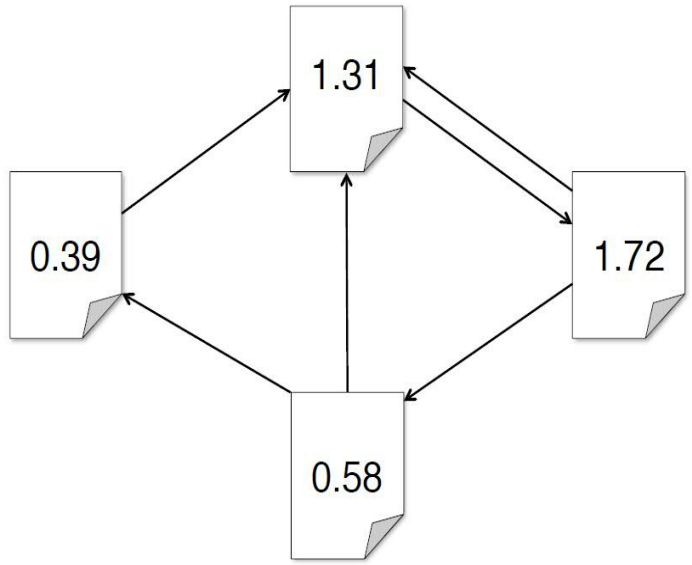
$$B = 0.85 * A$$

$$U = 0.15 * V$$

$$B*(B*V+U)+U = ?$$



PageRank Example



DATABRICKS 19

$$A = \begin{bmatrix} 0 & 0.5 & 1 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

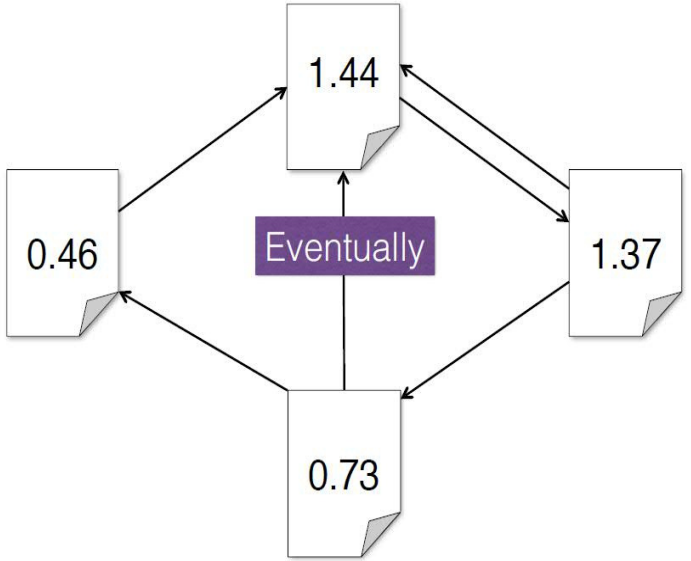
$$B = 0.85 * A$$

$$U = 0.15 * V$$

$$B*(B*V+U)+U = \begin{bmatrix} 1.31 \\ 1.72 \\ 0.39 \\ 0.58 \end{bmatrix}$$

$$B*(B*(B*V+U)+U)+U = \ldots$$

PageRank Example



DATABRICKS 20

```
     [0  0.5  1  0.5]
A =  [1  0    0  0  ]
     [0  0    0  0.5]
     [0  0.5  0  0  ]

     [1]
V =  [1]
     [1]
     [1]

B = 0.85*A
U = 0.15*V
```

At the k-th step: $B^k * V + (B^{k-1} + B^{k-2} + ... + B^2 + B + I) * U = B^k * V + (I - B^k) * (I - B)^{-1} * U$

For k=10:

```
[1.43]
[1.37]
[0.46]
[0.73]
```

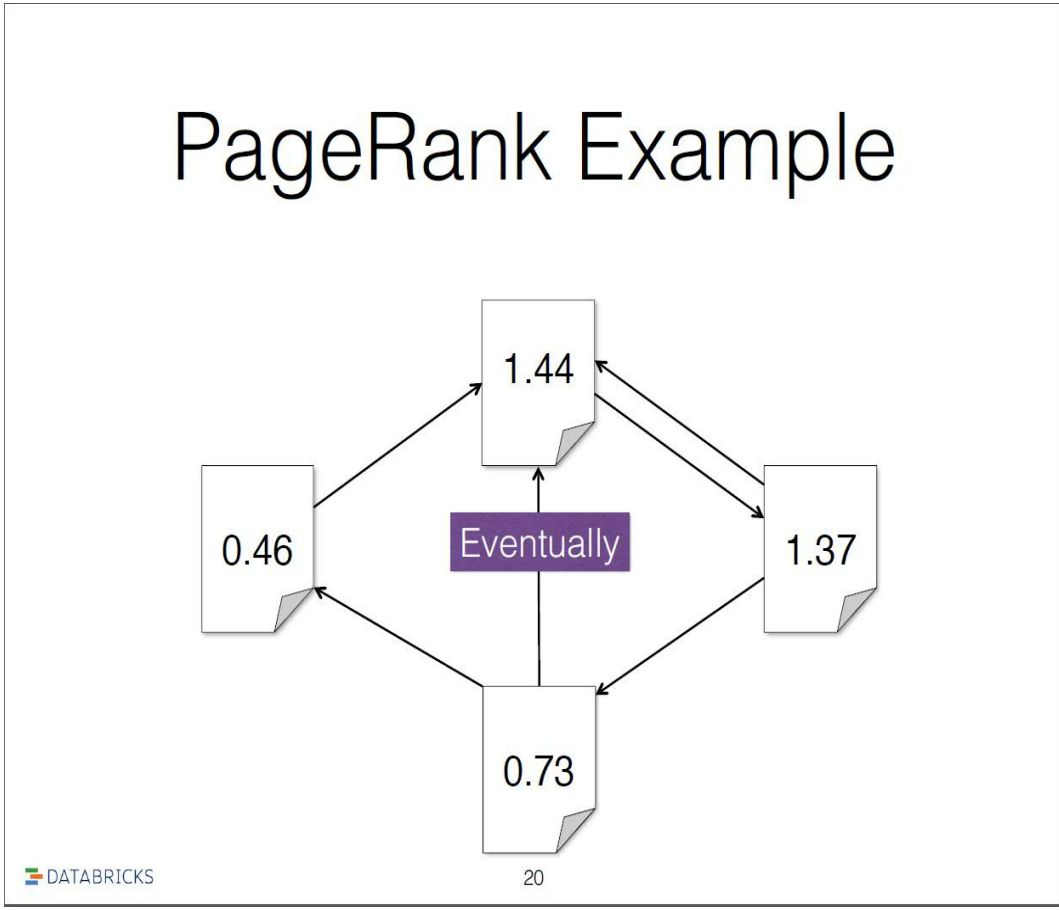
```
     [0  0.5  1  0.5]
A =  [1  0    0  0  ]
     [0  0    0  0.5]
     [0  0.5  0  0  ]

     [1]
V =  [1]
     [1]
     [1]

B = 0.85*A
U = 0.15*V
```

Where k goes to infinity:

$$B^k * V \rightarrow 0$$

$$B^k * V + (I - B^k) * (I - B)^{-1} * U \rightarrow (I - B)^{-1} * U$$

```
                    [1.44]
(I-B)^(-1)*U  =     [1.37]
                    [0.46]
                    [0.73]
```



```
     [0  0.5  1  0.5]
A =  [1  0    0   0 ]    B=0.85*A
     [0  0    0  0.5]
     [0  0.5  0   0 ]
```

- Characteristic polynomial of A:

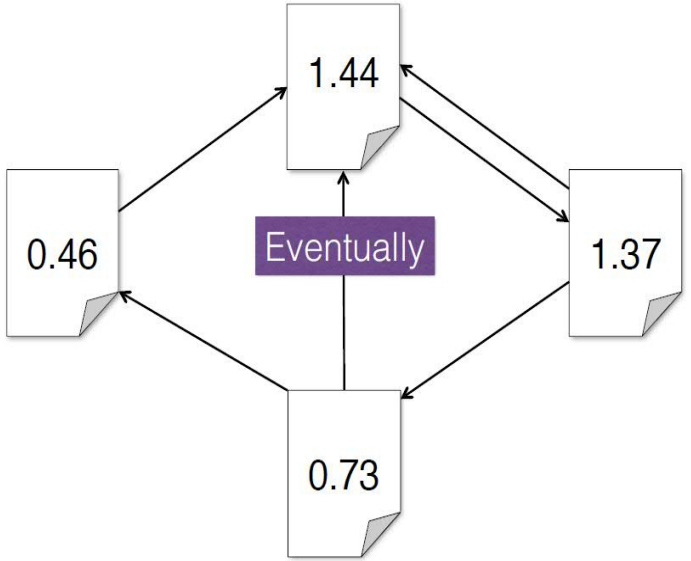
$$x^4 - 0.5x^2 - 0.25x - 0.25$$

- A is diagonalizable,
- 1 is the largest eigen value of A (in its absolute value),
- 1 corresponds to the eigen vector:

```
     [1.0 ]
E =  [1.0 ]
     [0.25]
     [0.5 ]
```

Where k goes to infinity: $A^k * V \to cE$, $B^k * V \to 0$

PageRank Example



DATABRICKS 20

Example: Page Rank

- Rank of each page is the probability of landing on that page for a random surfer on the web
- Probability of visiting all pages after k steps is

$$V_k = A^k \times V^t$$

V: the initial rank vector

A: the link structure (sparse matrix)

- Each page is identified by its unique URL rather than an index
- Ranks vectors (**V**): RDD[(URL, Double)]
- Links matrix (**A**): RDD[(URL, List(URL))]

Example: Page Rank

```
val links = // load RDD of (url, neighbors) pairs
var ranks = // load 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(...)
```

```
// Load the edges as a graph
val graph = GraphLoader.edgeListFile(sc, "graphx/data/followers.txt")
// Run PageRank
val ranks = graph.pageRank(0.0001).vertices
```

Spark Platform

Patrick Wendell
Databricks

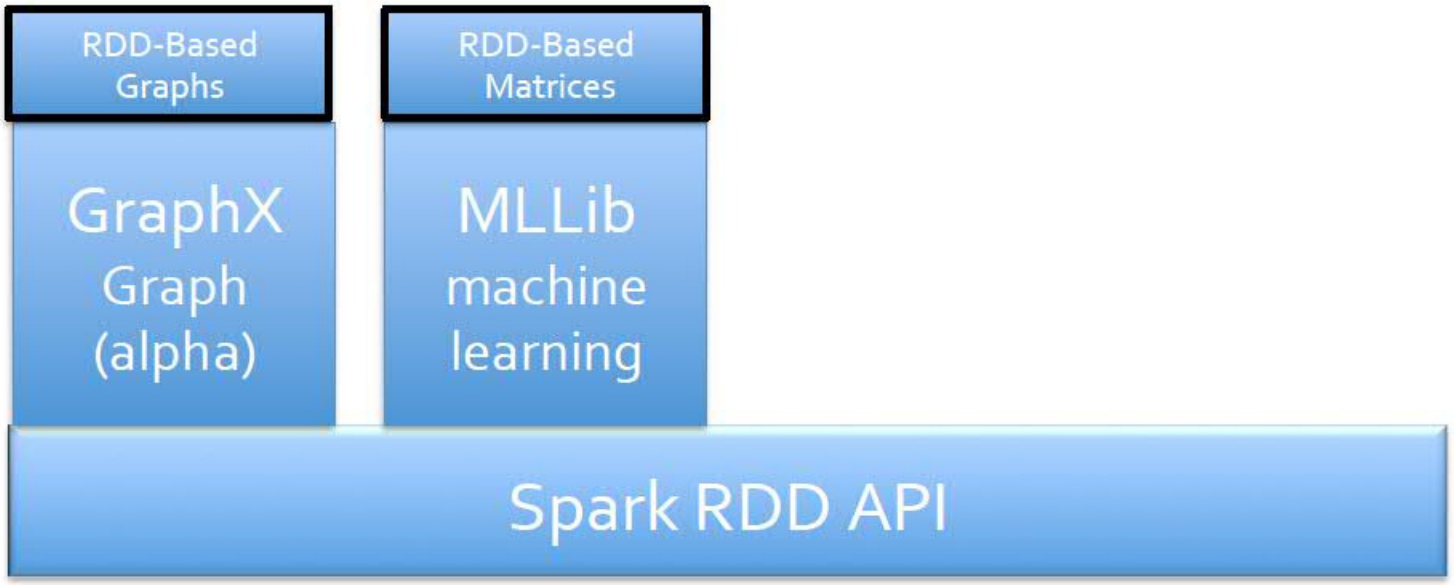
Spark Platform: GraphX

```
graph = Graph(vertexRDD, edgeRDD)
graph.connectedComponents() # returns a new RDD
```

Patrick Wendell
Databricks

Spark Platform: MLLib

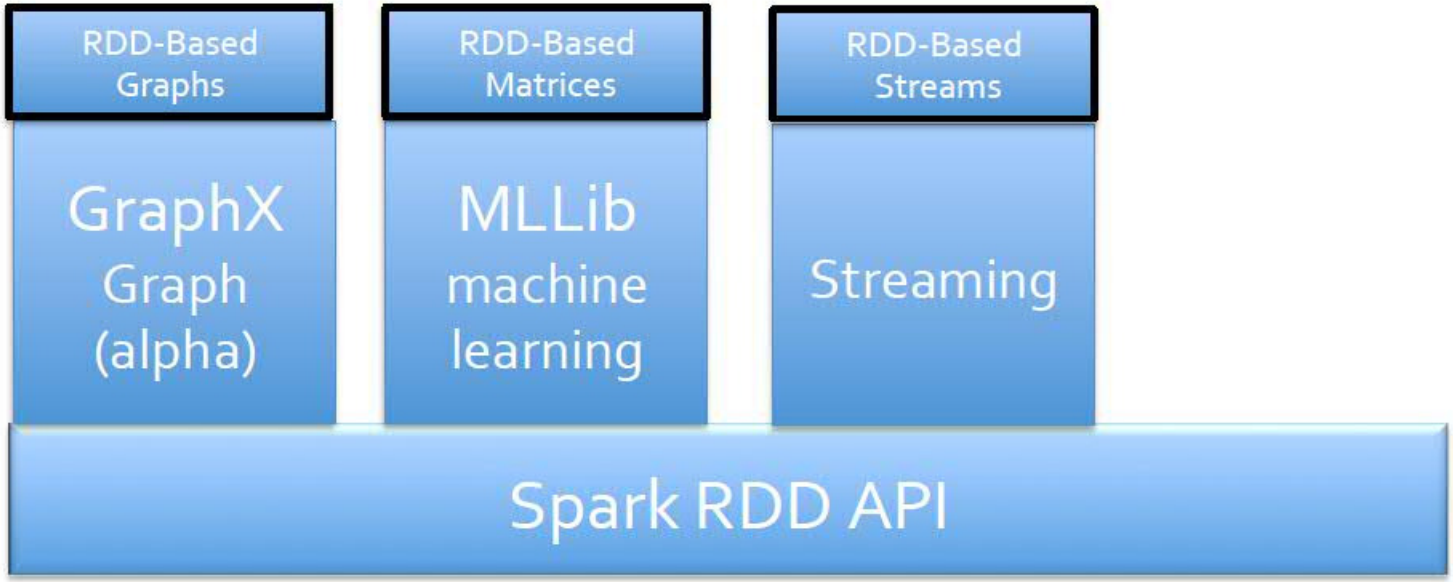
```
model = LogisticRegressionWithSGD.train(trainRDD)
dataRDD.map(point => model.predict(point))
```



Patrick Wendell
Databricks

Spark Platform: Streaming

```
dstream = spark.networkInputStream()
dstream.countByWindow(Seconds(30))
```



Patrick Wendell
Databricks

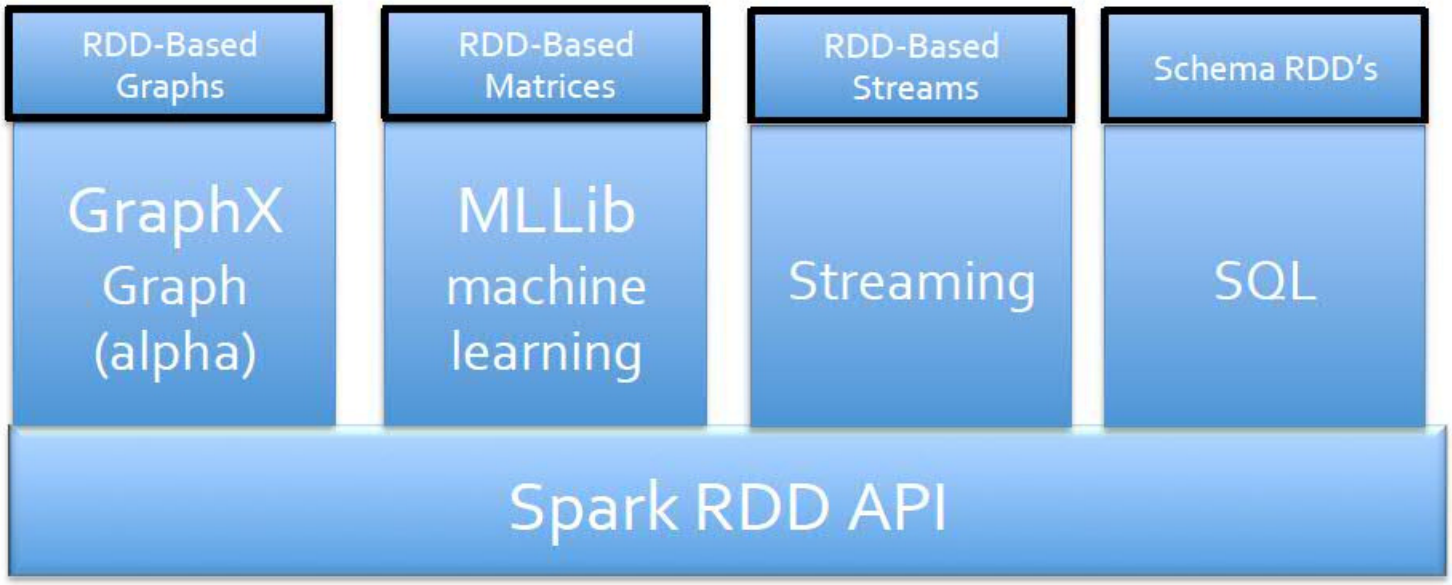
Spark Streaming

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

- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

Spark Platform: SQL

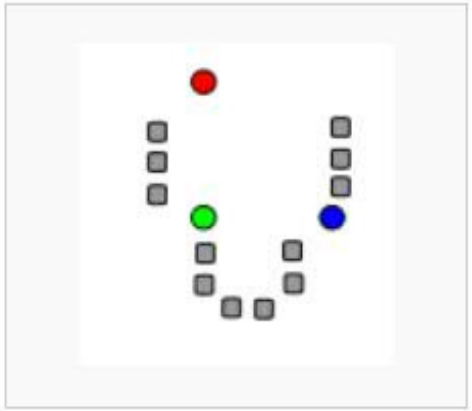
```
rdd = sql“select * from rdd1 where age > 10”
```



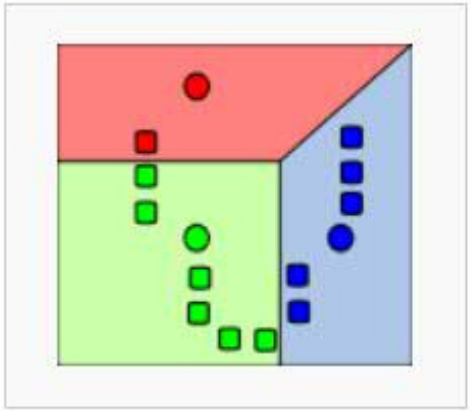
Patrick Wendell
Databricks

Machine Learning: K-Means clustering

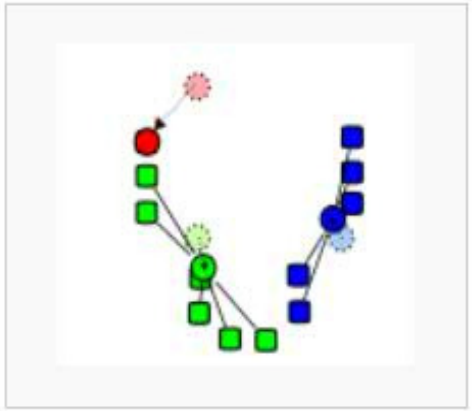
Demonstration of the standard algorithm



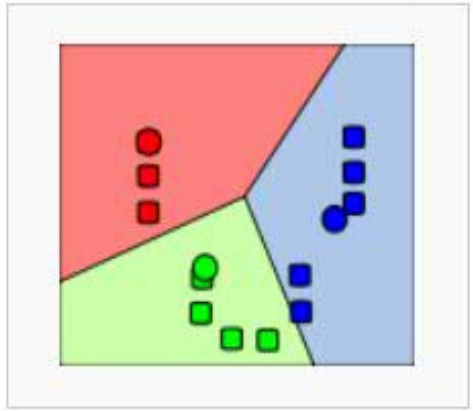
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the *k* clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

(from Wikipedia)

Example:
Spark MLLib & Geolocation

Goal:

Segment tweets into clusters by geolocation

using Spark MLLib K-means clustering

```
<longitude>, <latitude>, <timestamp>, <userId>, <tweet message>

-56.544541,-29.089541,1403918487000,1706271294,Por que ni estamos jugando, son más pajeros e
-69.922686,18.462675,1403918487000,2266363318,Aprenda hablar amigo
-118.565107,34.280215,1403918487000,541836358,today a boy told me I'm pretty and he loved me
121.039399,14.72272,1403918487000,362868852,@Kringgelss labuyoo. Hahaha
-34.875339,-7.158832,1403918487000,285758331,@keithmeneses_ oi td bem? sdds 😔💚
103.766123,1.380696,1403918487000,121042839,Xian Lim on iShine 3 2
```

https://chimpler.wordpress.com/2014/07/11/segmenting-audience-with-kmeans-and-voronoi-diagram-using-spark-and-mllib/

Example: Spark MLLib & Geolocation

To run the k-means algorithm in Spark, we need to first read the csv file

```
val sc = new SparkContext("local[4]", "kmeans")
// Load and parse the data, we only extract the latitude and longitude of each line
val data = sc.textFile(arg)
val parsedData = data.map {
  line =>
    Vectors.dense(line.split(',').slice(0, 2).map(_.toDouble))
}
```

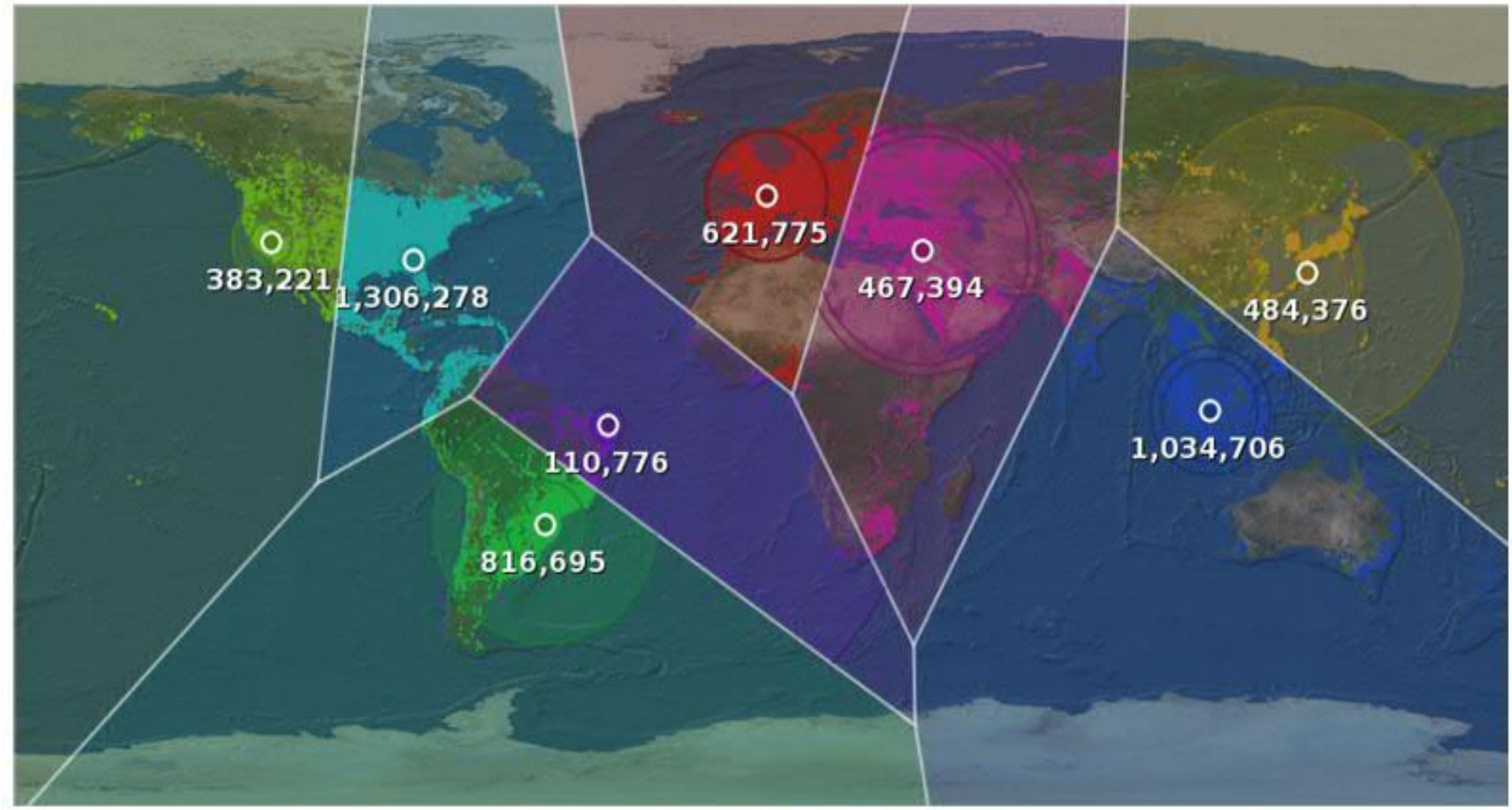
Then we can run the spark kmeans algorithm:

```
val iterationCount = 100
val clusterCount = 10
val model = KMeans.train(parsedData, clusterCount, iterationCount)
```

From the model we can get the cluster centers and group the tweets by cluster:

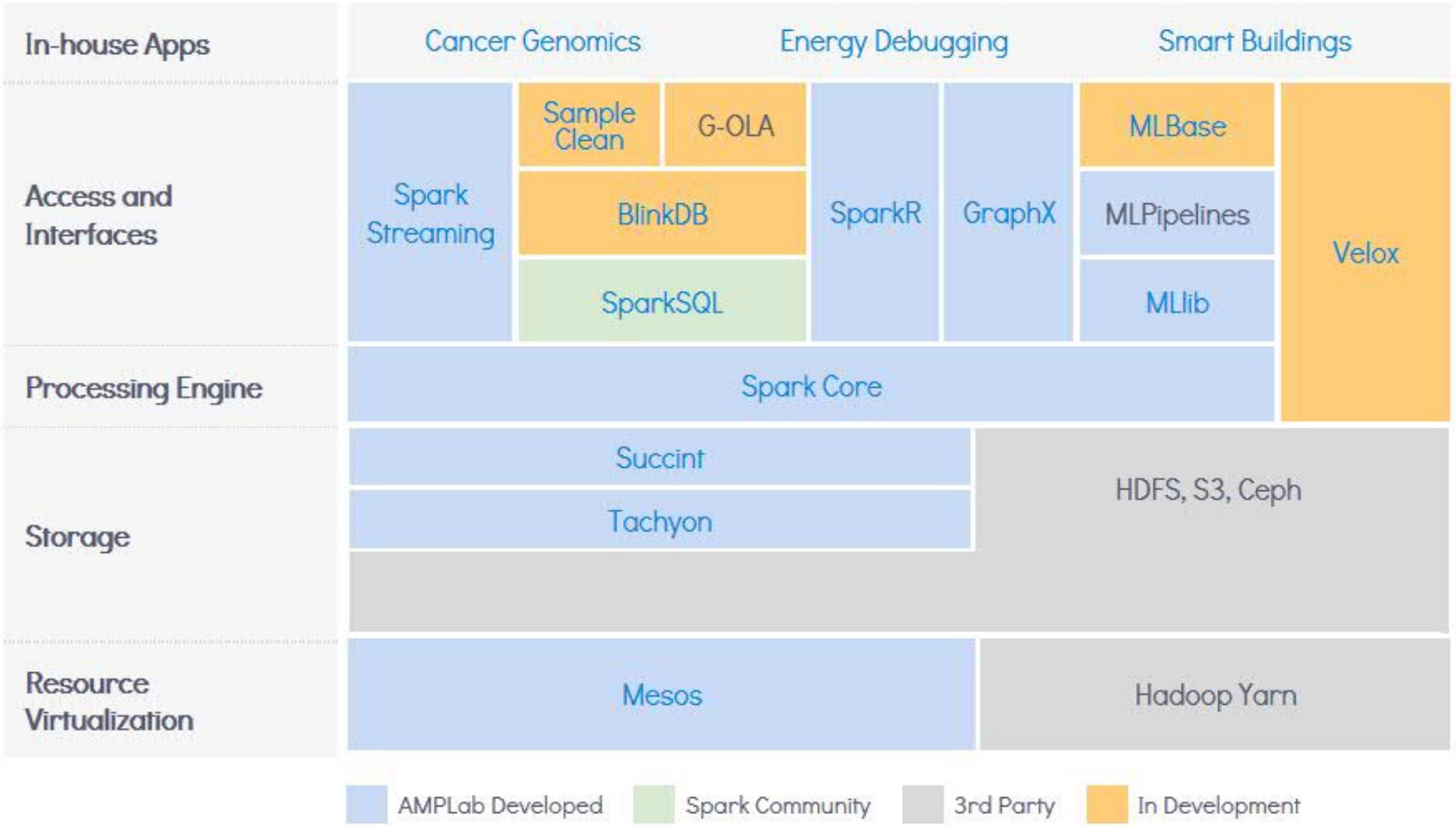
```
val clusterCenters = model.clusterCenters map (_.toArray)

val cost = model.computeCost(parsedData)
println("Cost: " + cost)

val tweetsByGoup = data
  .map {_.split(',').slice(0, 2).map(_.toDouble)}
  .groupBy{rdd => model.predict(Vectors.dense(rdd))}
  .collect()
sc.stop()
```

Example:
Spark MLLib & Geolocation



https://chimpler.wordpress.com/2014/07/11/segmenting-audience-with-kmeans-and-voronoi-diagram-using-spark-and-mllib/

The Next Generation...



https://amplab.cs.berkeley.edu/software/

References

Papers:

- Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica, *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,* NSDI 2012, April 2012.

Videos and Presentations:

- Spark workshop hosted by Stanford ICME, April 2014 - Reza Zadeh, Patrick Wendell, Holden Karau, Hossein Falaki, Xianguri Meng, http://stanford.edu/~rezab/sparkworkshop/
- Spark summit 2014 - Matei Zaharia, Aaaron Davidson, http://spark-summit.org/2014/
- Paul Krzyzanowski, "Distributed systems": //www.cs.rutgers.edu/~pxk/

Links:

- https://spark.apache.org/docs/latest/index.html
- http://hadoop.apache.org/


thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
спасибо
Баярлалаа
рахмат
dank je
bedankt
dziękuję
obrigado
sukriya
kop khun krap
go raibh maith agat
mochchakkeram
merci
grazie
arigatō
takk
dakujem
мерси
terima kasih
ありがとう
감사합니다
ευχαριστώ
tapadh leat
hvala
mauruuru
xiexie
dhanyavad
faafetai lava
blagodaram
paldies
хвала
asante
manana
trugarez
shukriya
dhanyavadagalu
diolch
rahmet
tanemirt
chnorakaloutioun
gracies
sulpay
sobodi
dekuji
mesi
najis tuke
kam sah hamnida
mersi
barka
welalin
tack
misaotra
matondo
vinaka
kiitos
dankie
enkosi
djiere dieuf