APACHE SPARK ADVANCED

DS8003 – MGT OF BIG DATA AND TOOLS

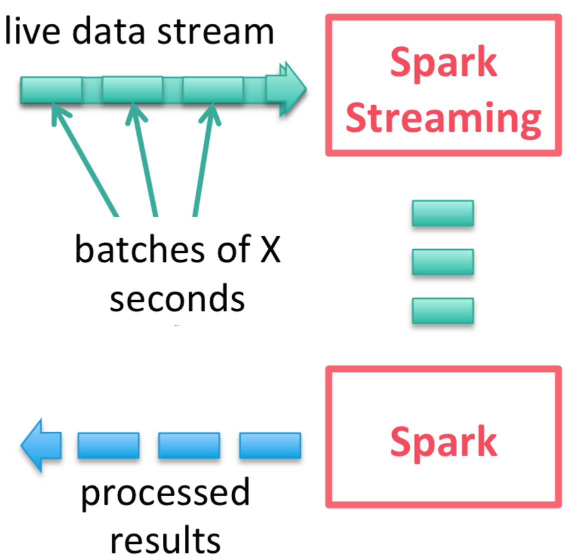
Ryerson University

Instructor: Kanchana Padmanabhan

Streaming

Run a streaming computation as a series of very small deterministic batch jobs

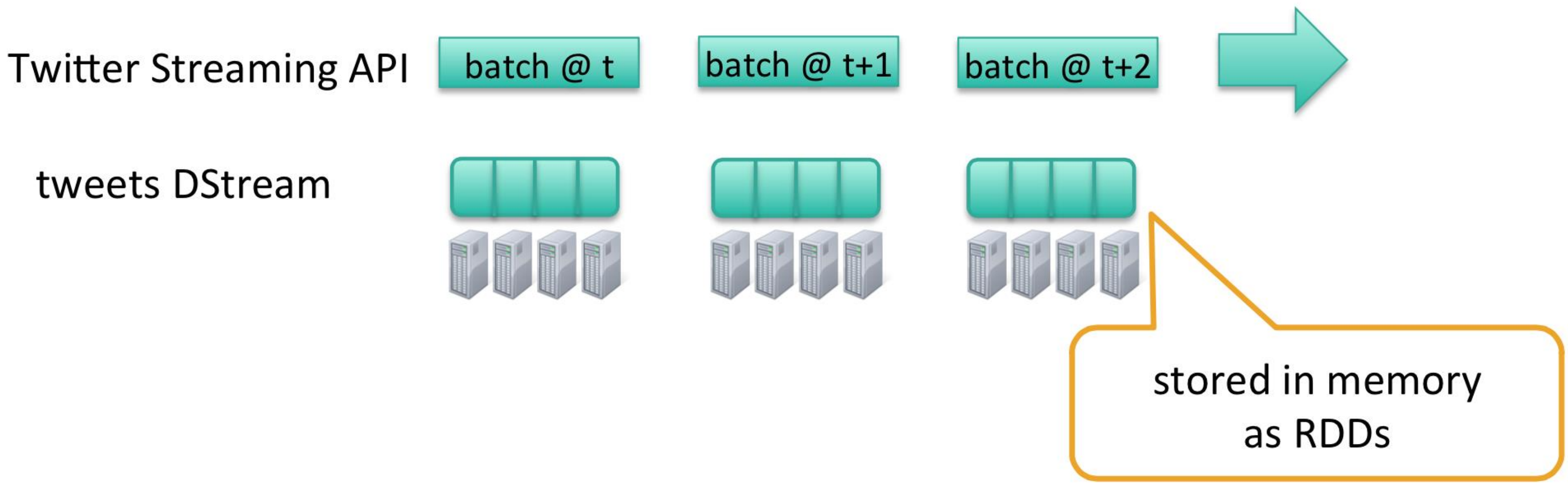
- Chop up the live stream into batches of X seconds (1/2 second or 1 second)
- Spark treats each batch of data as RDD
- Run same code in batch mode and streaming



https://databricks-training.s3.amazonaws.com/slides/Spark%20Summit%202014%20-%20Spark%20Streaming.pdf

Streaming - DStream

- Discretized Stream (Dstream)
  - Represents a stream of data
  - Implemented as sequence of RDDs
  - Created from streaming input sources
  - Created by applying transformations



Streaming - Code

1. from pyspark import SparkContext
2. from pyspark.streaming import StreamingContext
3. conf = SparkConf().setAppName(**"Testing Spark Streaming”**)
4. sc = SparkContext(conf = conf) [Required only if code is in python file. Not required in pyspark console]
5. stream = StreamingContext(sc, 1)
6. lines = stream.socketTextStream(**"localhost"**, 9999)
7. counts = lines.flatMap(**lambda** line: line.split(**" "**)).map(**lambda** word: (word, 1)).reduceByKey(**lambda** a, b: a+b)
8. counts.pprint()
9. stream.start()
10. stream.awaitTermination()

11. [Code is same as **streamingExample.py**]

Testing your streaming code

- Example details here:http://spark.apache.org/docs/latest/streaming-programming-guide.html
- Open two connections to VirtualBox (you will open two consoles)
- One first console type "spark-submit streamingExample.py"
- On second console type command "nc -lk 9999". Start typing some text on the console screen
- The text will get processed by the spark program running on the first console

DataFrames - SQLContext

- Spark SQL is a Spark module for structured data processing
- The DataFrameprovides easier access to data, because it looks conceptually like a table
- A DataFrame is a *Dataset* organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python
- An extension SQLContext is HiveContext
- SqlContext can also load JSON's into DataFrames
  - Uses "null" for missing values

http://spark.apache.org/docs/latest/sql-programming-guide.html#sql

Creating a DataFrame

- from pyspark.sql import SQLContext
- sqlContext = SQLContext(sc)
- v = sqlContext.createDataFrame([
    ("a", "Alice", 34),
    ("b", "Bob", 36),
    ("c", "Charlie", 30),
  ], ["id", "name", "age"])
- v.select(“id”).show()
- v.groupBy(“age”).count().show()

Machine Learning - Mllib

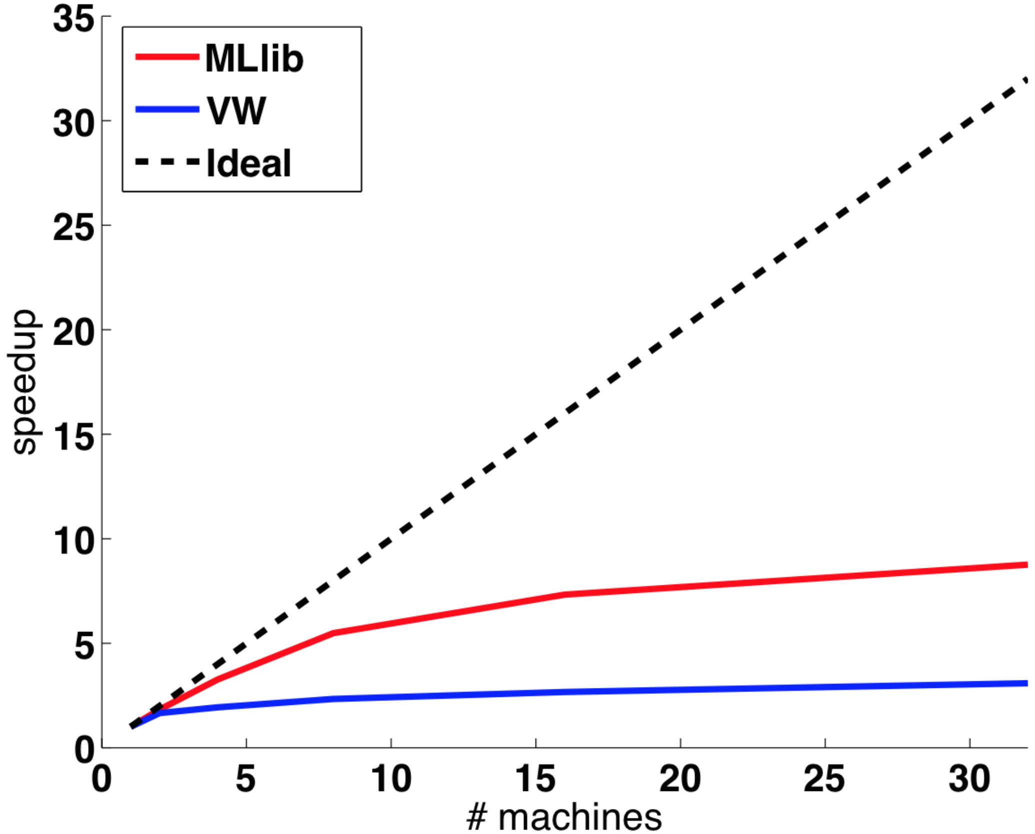
- MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy.
- It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction
- Spark excels at iterative computation, enabling MLlib to run fast. At the same time, we care about algorithmic performance: MLlib contains high-quality algorithms that leverage iteration, and can yield better results than the one-pass approximations sometimes used on MapReduce.
- Supposedly, running times or up to 100x faster than Hadoop MapReduce

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

https://databricks-training.s3.amazonaws.com/slides/Spark_Summit_MLlib_070214_v2.pdf

Mllib - scaling

Logistic Regression



http://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf

Classification

- Classification takes a set of data already divided into predefined groups and searches for patterns in the data that differentiate those groups.
- The discovered patterns then can be used to classify other data where the right group designation is unknown (though other attributes may be known).
- Logistic regression – as a classifier. It is widely used to predict a binary response.

http://www.britannica.com/technology/data-mining

Logistic Regression – As a Classifier

- Models relationship between set of variables
  - – dichotomous such as rain (yes/no)
  - – categorical (department, country)
  - – continuous (age, salary, weight, height…)
- Binary outcome (Y) variable (sentiment, 1= positive, and 0= negative)

https://www.nemoursresearch.org/open/StatClass/January2011/Class8.pdf

http://www.ncss.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Logistic_Regression.pdf

Logistic Regression - Example

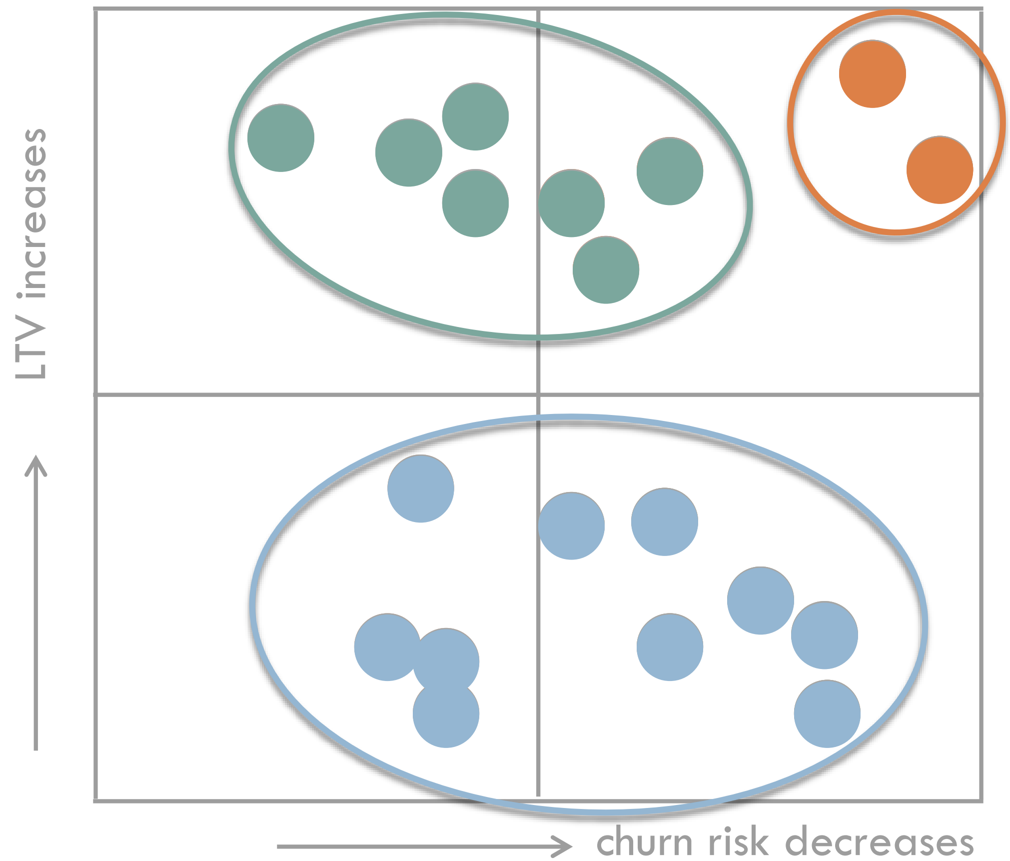

```
from pyspark.mllib.classification import LogisticRegressionWithSGD, LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName("Testing Spark Commands”)
sc = SparkContext(conf = conf)
data = [
    LabeledPoint(0.0, [0.0, 1.0]),
    LabeledPoint(1.0, [1.0, 0.0])
]
lrm = LogisticRegressionWithSGD.train(sc.parallelize(data), iterations=10)
lrm.predict([1.0, 0.0])
#Output: 1.0
lrm.predict([0.0, 1.0])
#Output: 0.0
# Save and load model
lrm.save(sc, “lrsgd”)
sameModel = LogisticRegressionModel.load(sc, “lrsgd”)
sameModel.predict([1.0, 0.0])
sameModel.predict([0.0, 1.0])
```

[Code is same as logistic_regression_simple.py]

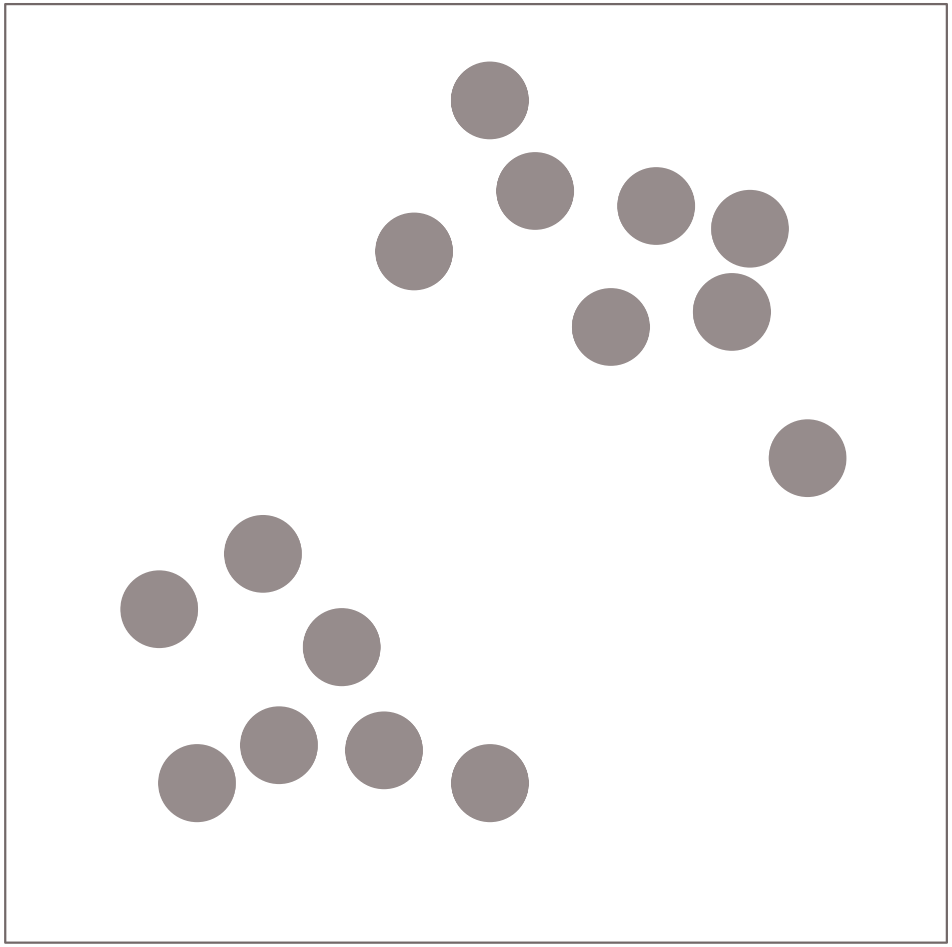
Clustering

- Customer Value Segmentation
  - Value-based segmentation is useful for designing CRM strategies
  - Segmentation is easy to explain and useful for designing your campaigns/offers/messages
  - Clustering is more effective algorithmically but suffers explainability



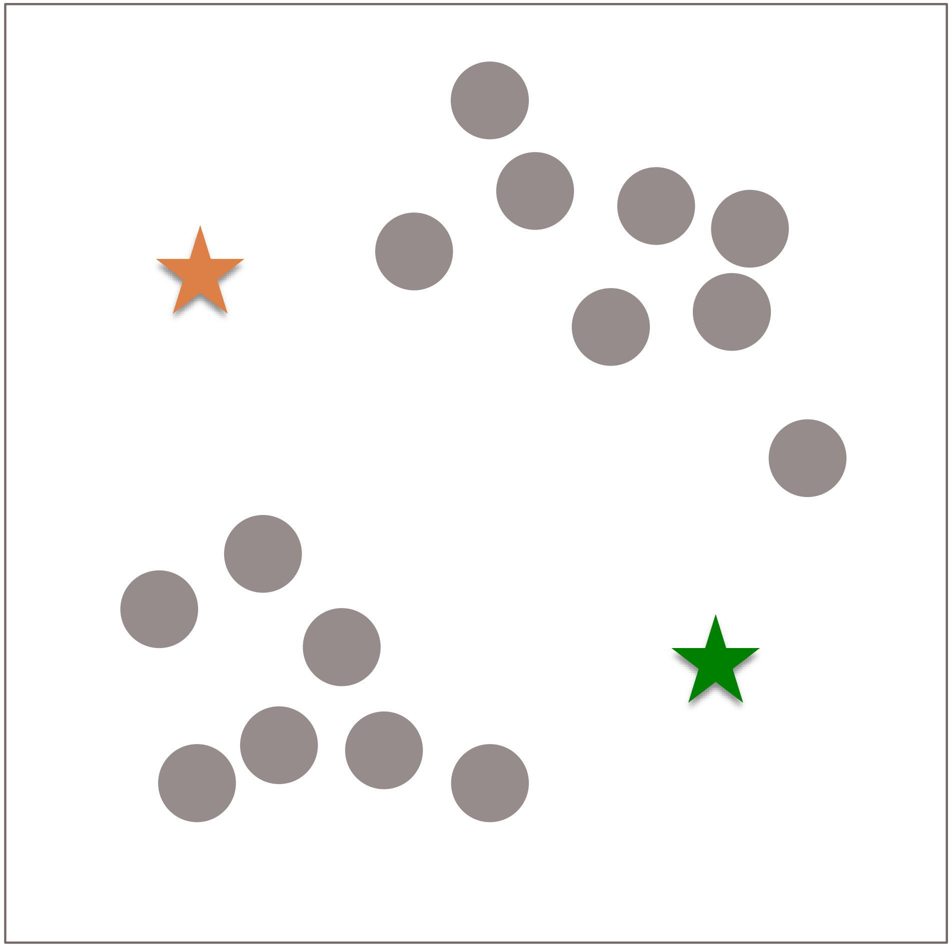
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are different from the old centers (previous iteration) → Go to Step 2
     - Otherwise, stop



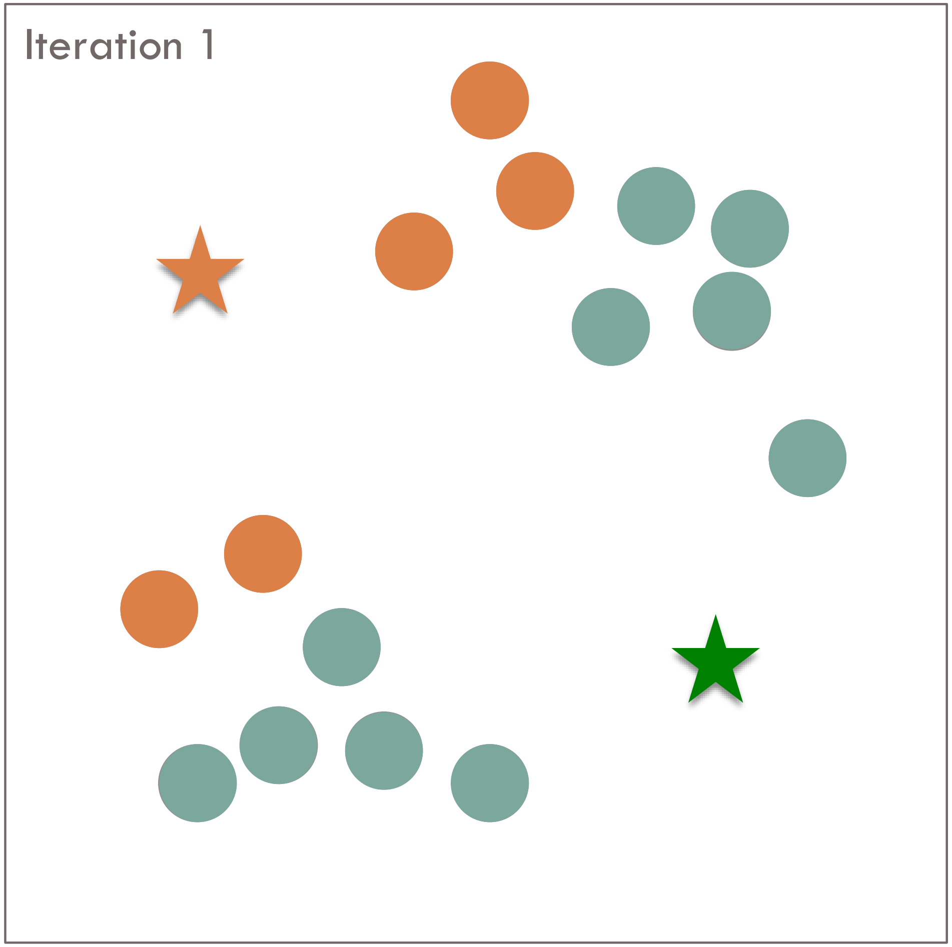
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are different from the old centers (previous iteration) → Go to Step 2
     - Otherwise, stop



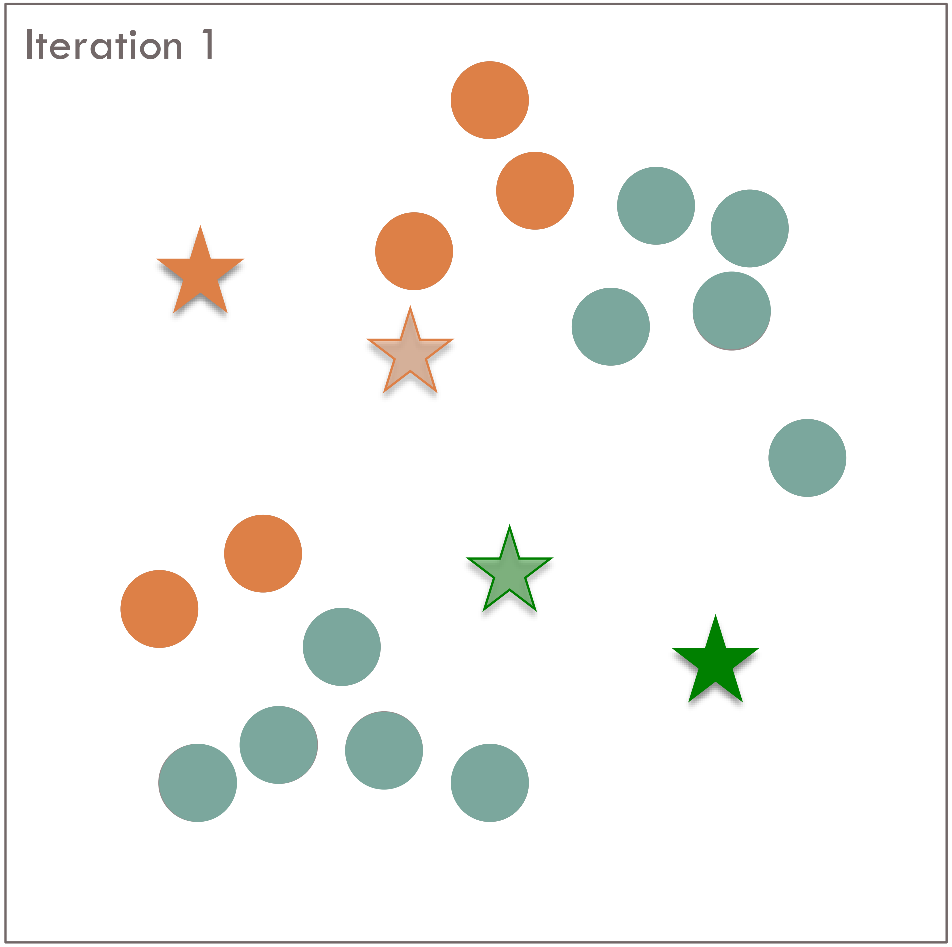
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are different from the old centers (previous iteration) → Go to Step 2
     - Otherwise, stop



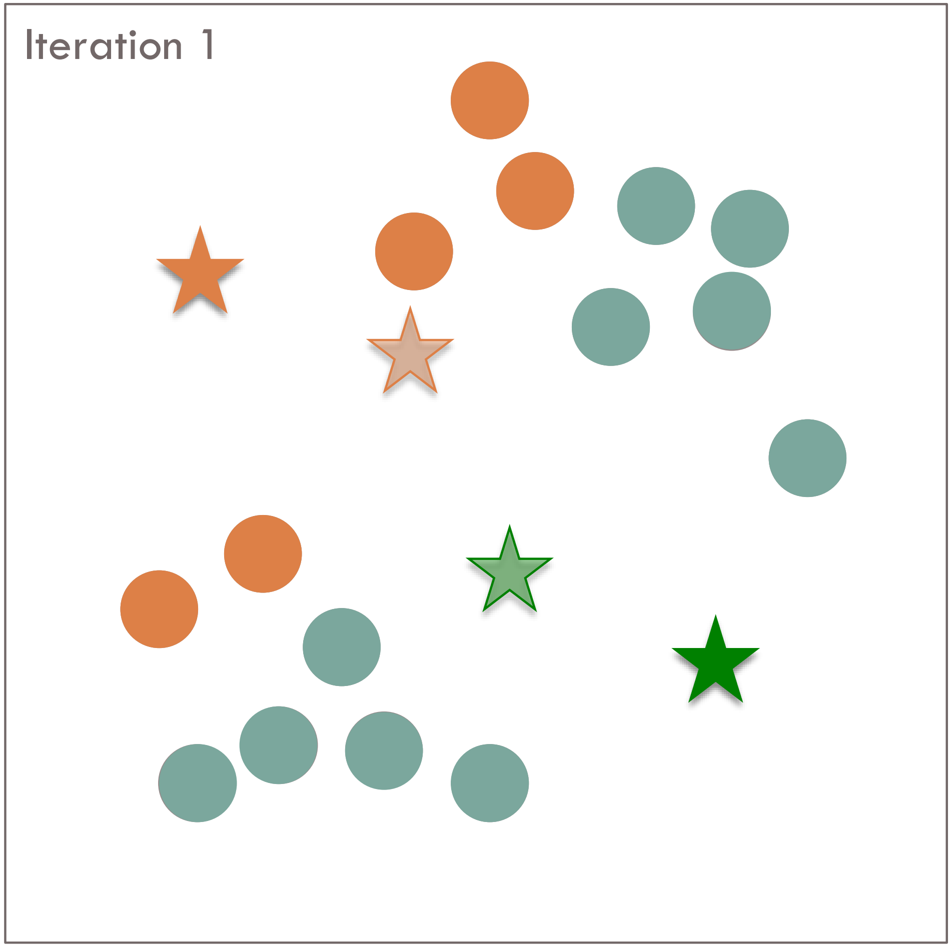
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are different from the old centers (previous iteration) → Go to Step 2
     - Otherwise, stop



K-Means – Single Node

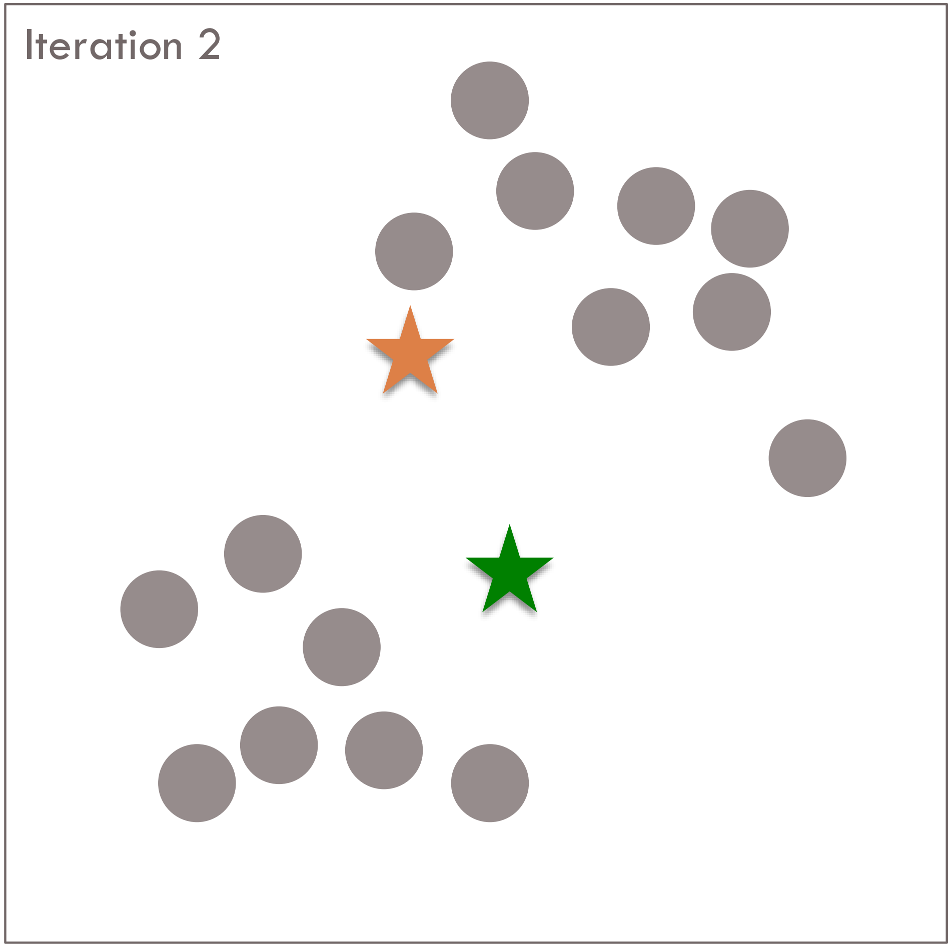
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



K-Means – Single Node

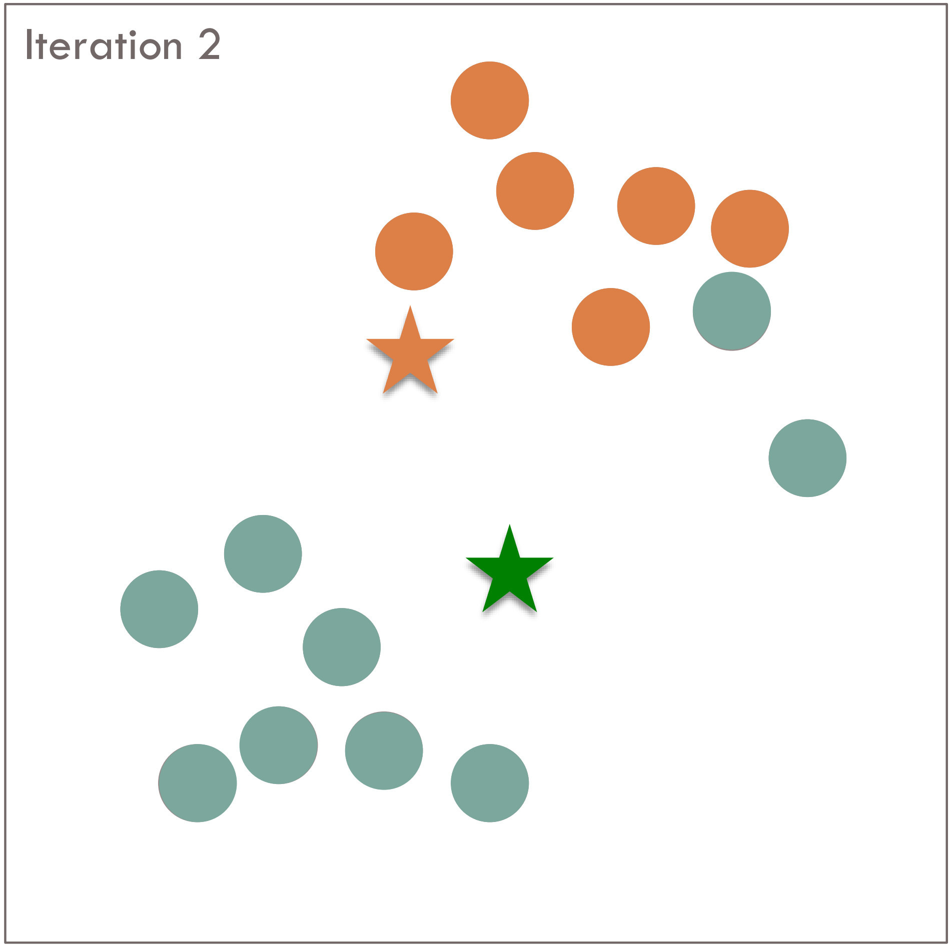
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop

Iteration 2 Starts



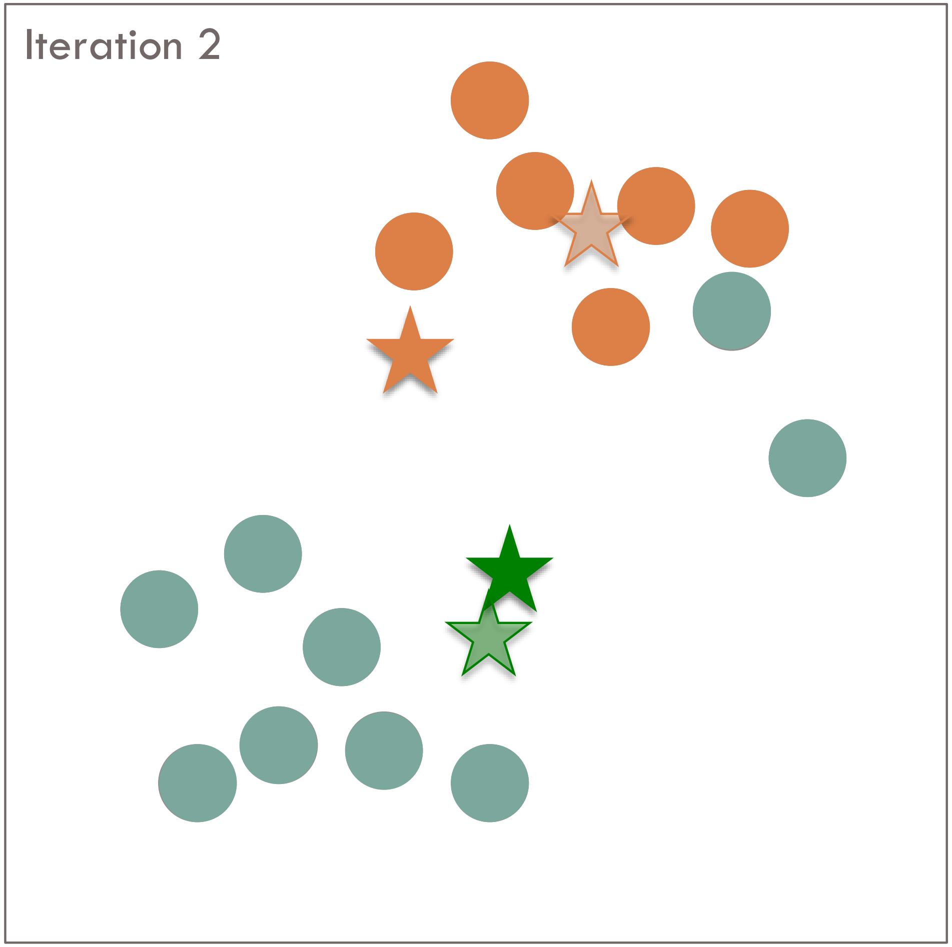
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



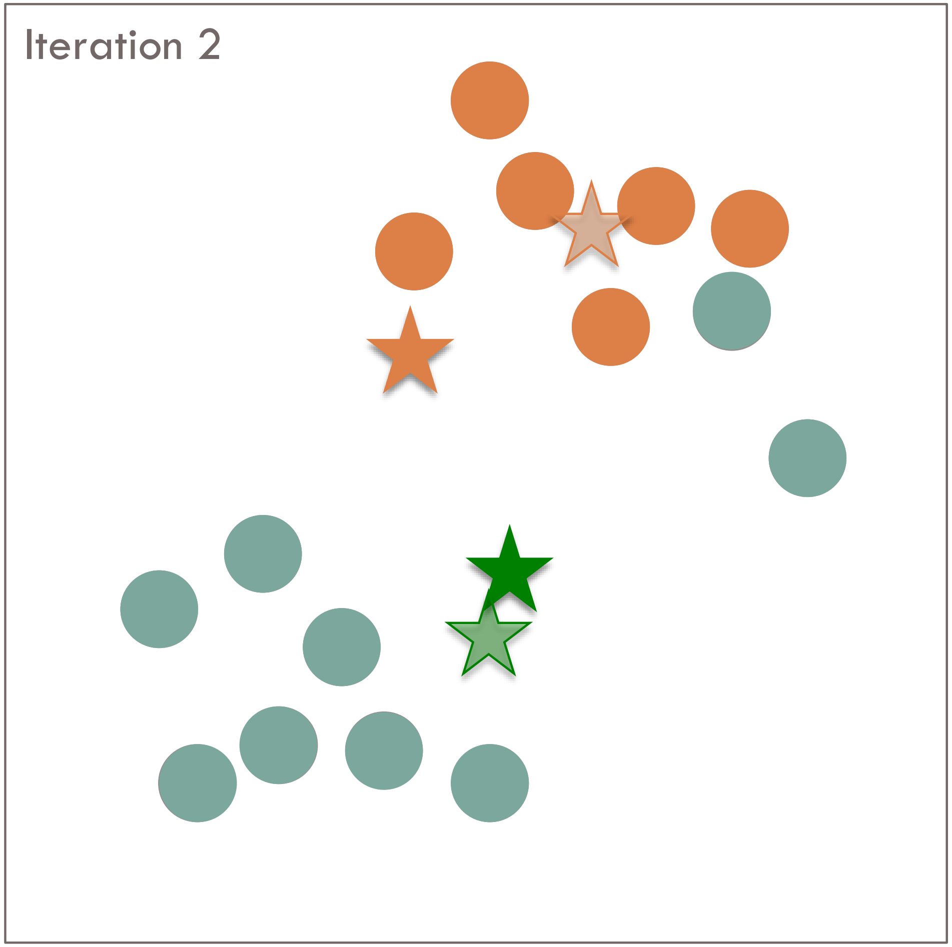
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



K-Means – Single Node

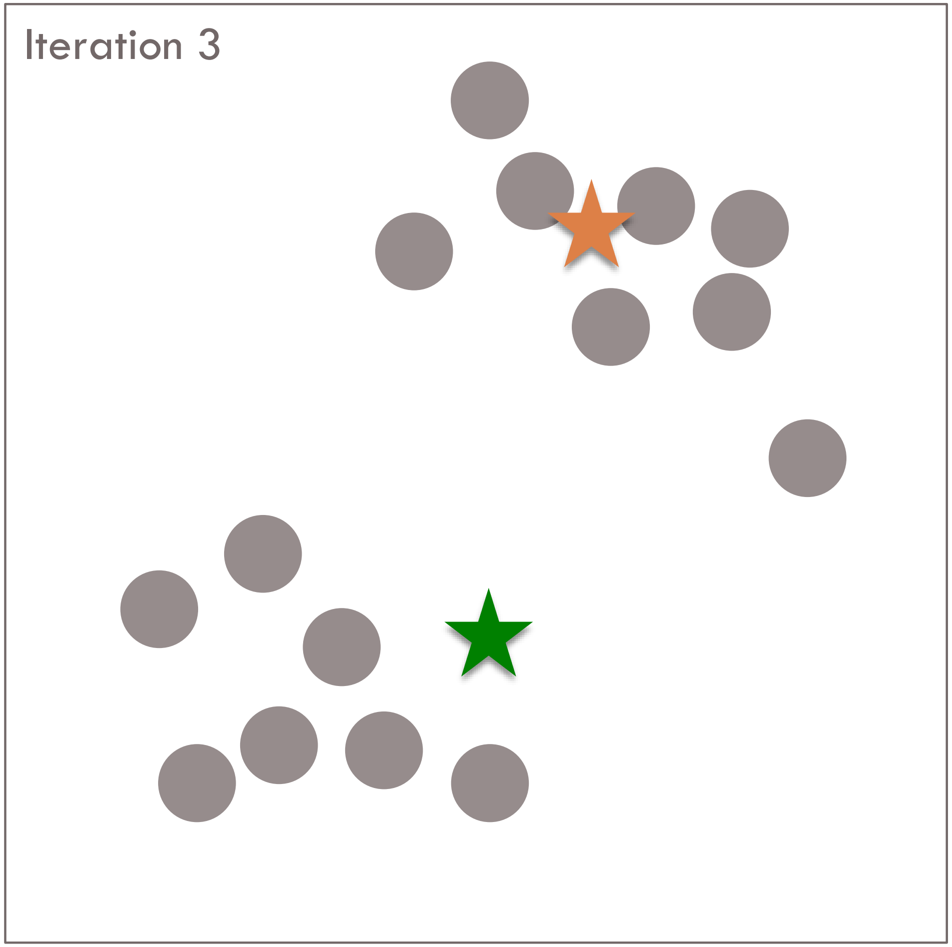
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



K-Means – Single Node

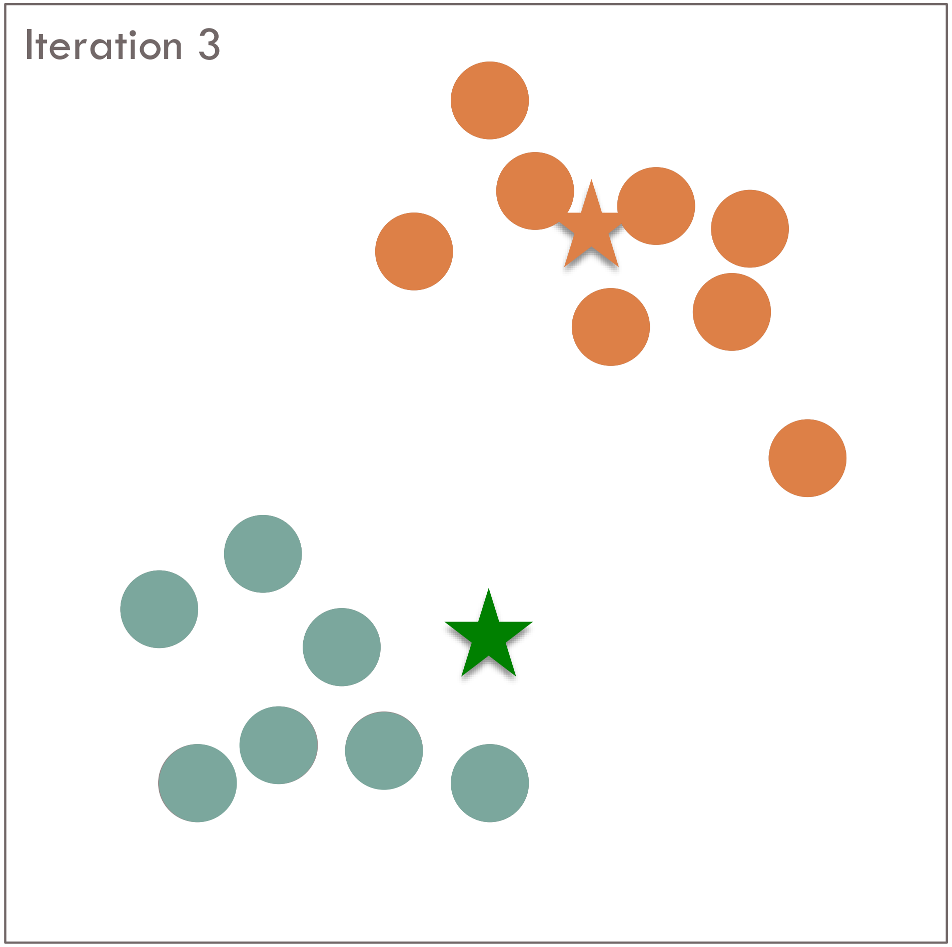
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop

Iteration 3 Starts



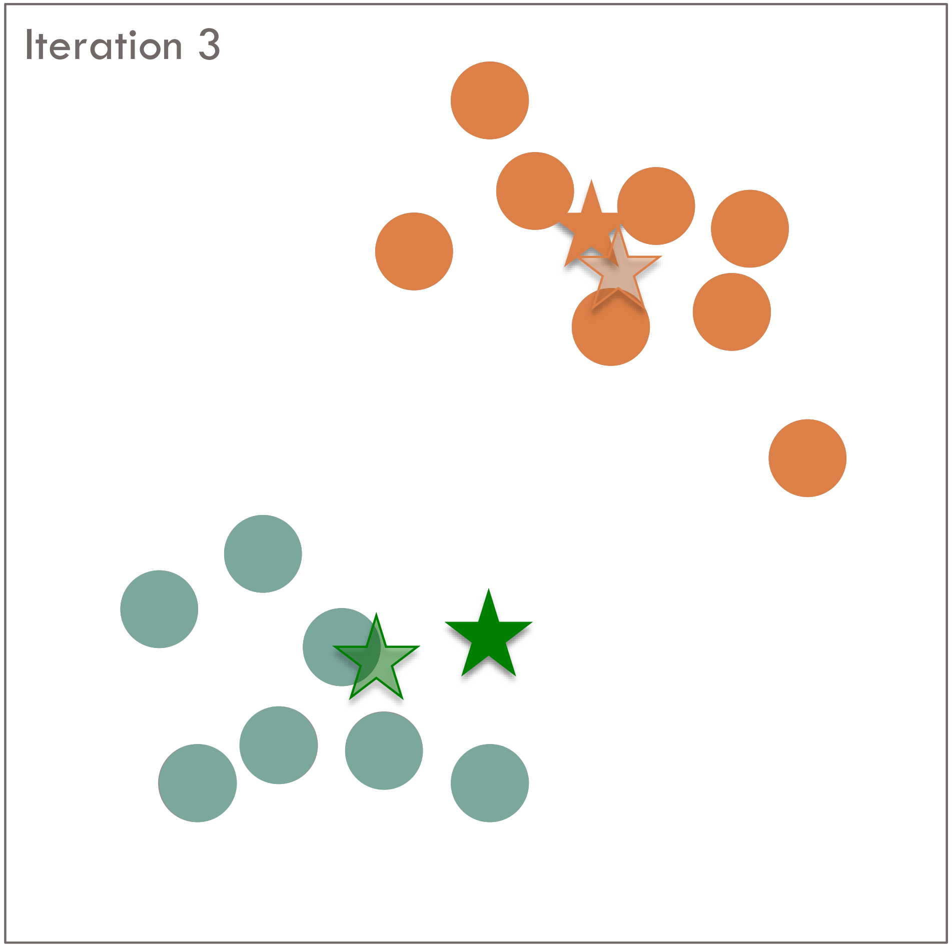
K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



K-Means – Single Node

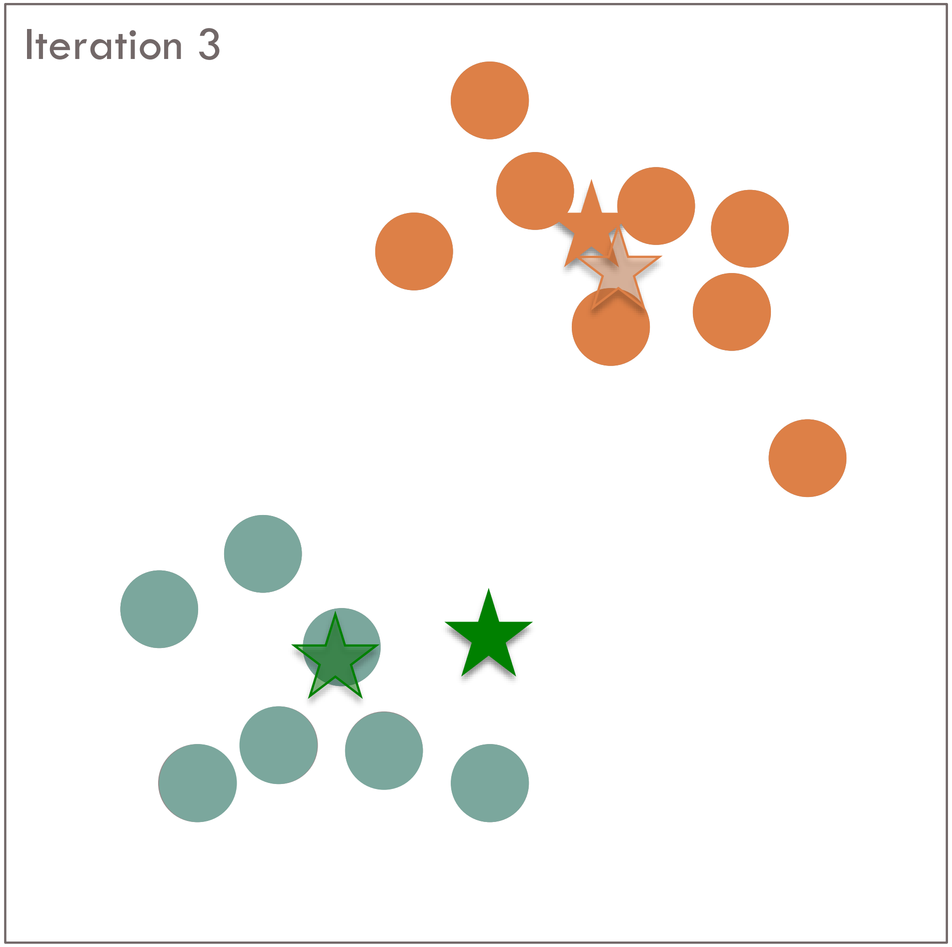
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop



K-Means – Single Node

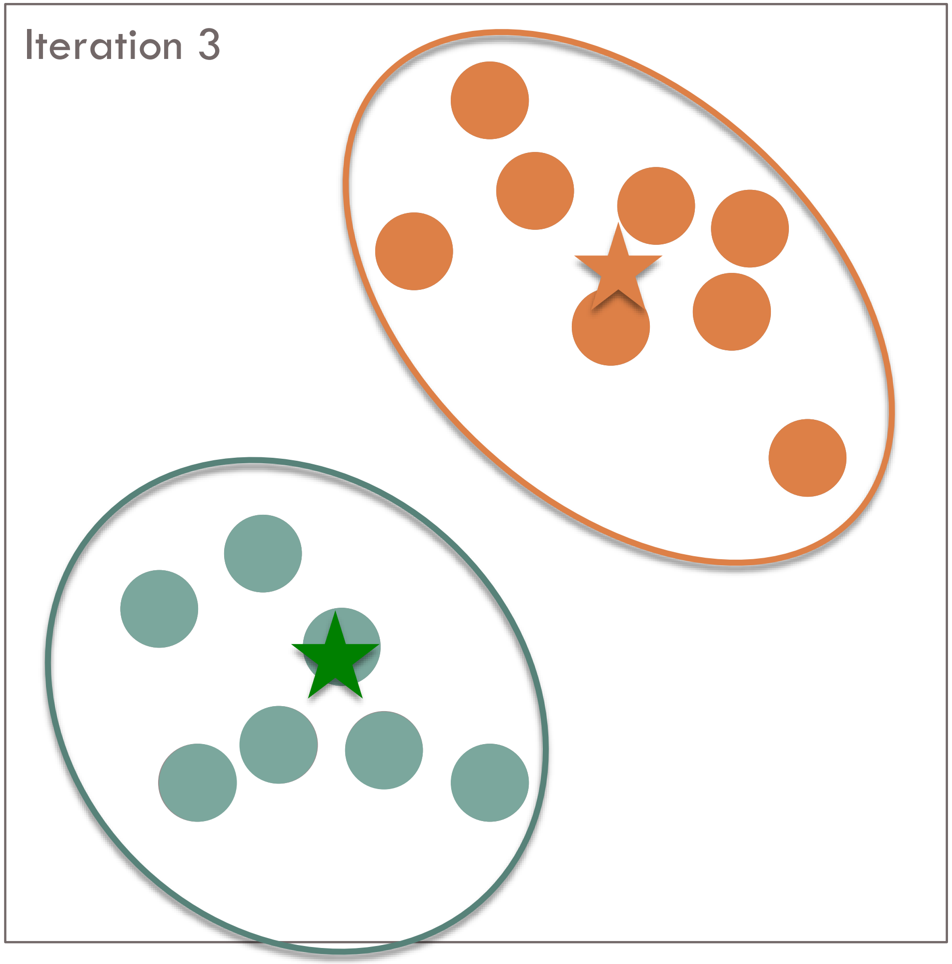
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop

Meets convergence criteria



K-Means – Single Node

- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average over all x-axis points in the cluster
       - Y: average over all y-axis points in the cluster
  4. If the new centers are significantly different from the old centers (previous iteration)
     - Set the new centers then Go to Step 2
     - Otherwise, stop

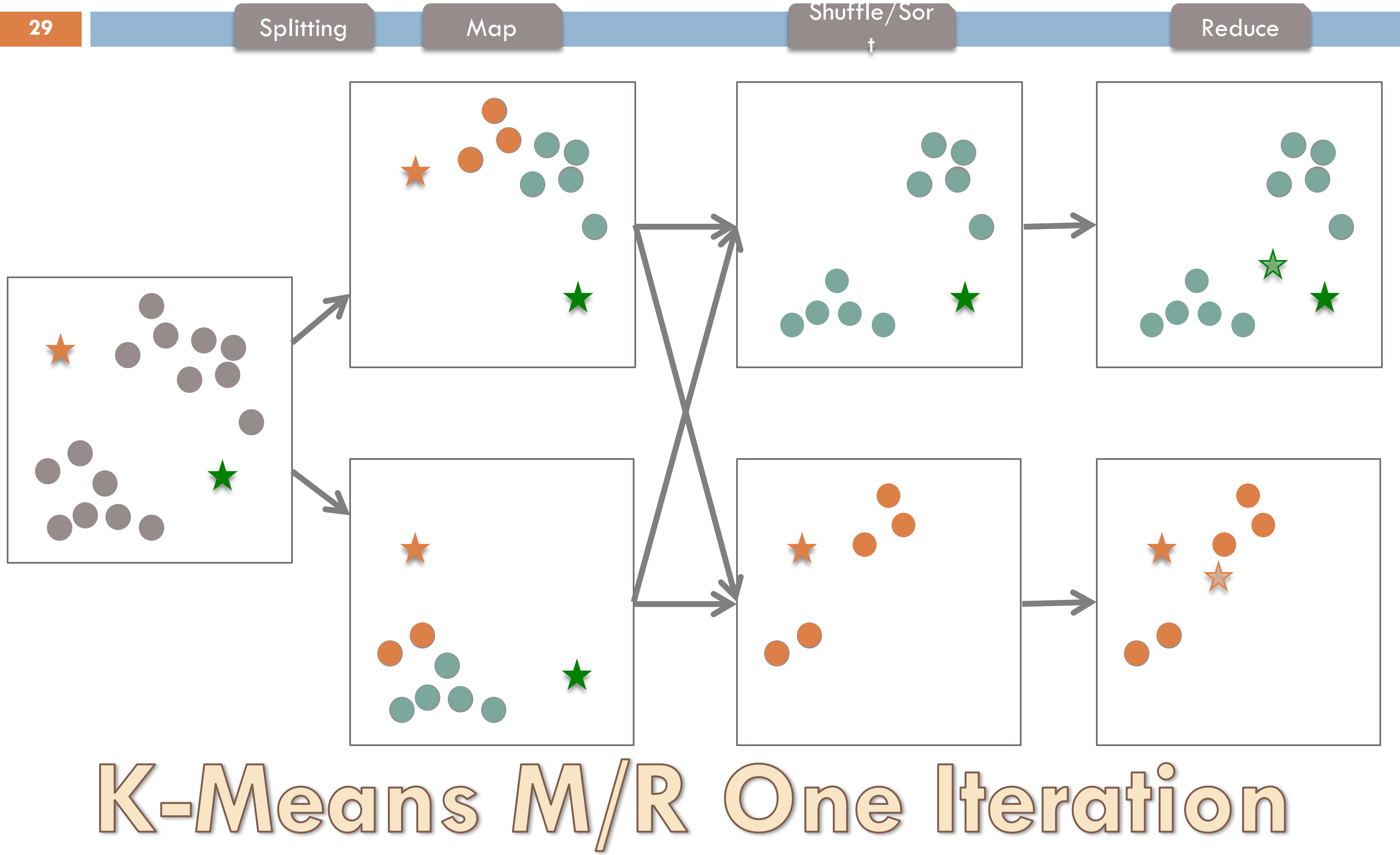


Meets convergence criteria

Distributed K-Means - MapReduce

- Map-side
  - Each map reads the K-centroids + one block from dataset
  - Assign each point to the closest centroid
  - Output <centroid, point>
- Reduce-side
  - Gets all points for a given centroid
  - Re-compute a new centroid for this cluster
  - Output: <new centroid>
- Iteration Control
  - Compare the old and new set of K-centroids
    - If similar → Stop
    - Else
      - If max iterations has reached → Stop
      - Else → Start another Map-Reduce Iteration
- Use of combiners
  - Similar to the reducer
  - Computes for each centroid the local sums (and counts) of the assigned points
  - Sends to the reducer <centroid, <partial sums>>
- Use of single reducer
  - Amount of data to reducers is very small
  - Single reducer can tell whether any of the centers has changed or not
  - Creates a single output file

Distributed K-Means - MapReduce



KMeans - Example


```
from pyspark import SparkConf, SparkContext
from pyspark.mllib.clustering import KMeans, KMeansModel
conf = SparkConf().setAppName("Testing Spark Commands”)
sc = SparkContext(conf = conf)
data = [[1.0,1.0],[1.0,0.8],[-1.0,1.0],[-1.0,-1.0]]
parsedData=sc.parallelize(data)
kmeansModel = KMeans.train(parsedData, 2, maxIterations=10, runs=10, initializationMode=“random”)
kmeansModel.predict([1.0, 1.0])
#Output: 1
kmeansModel.predict([1.0, -2.0])
#Output: 0
# Save and load model
kmeansModel.save(sc, ”KMeansModel")
model = KMeansModel.load(sc, “KmeansModel")
model.predict([1.0, 1.0])
model.predict([1.0, -2.0])
```

[Code same as kmeans_simple.py]

Collaborative Filtering: Recommendation

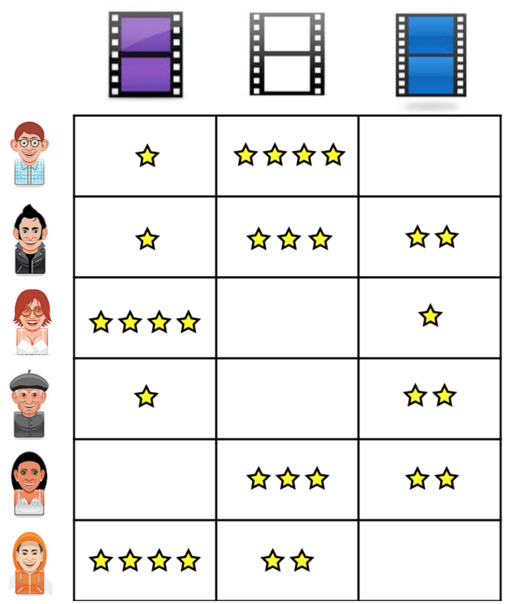
- Crowdsourced data
- Netflix Movie recommendation
- Amazon Product Recommendation

Image via Erik Bernhardsson

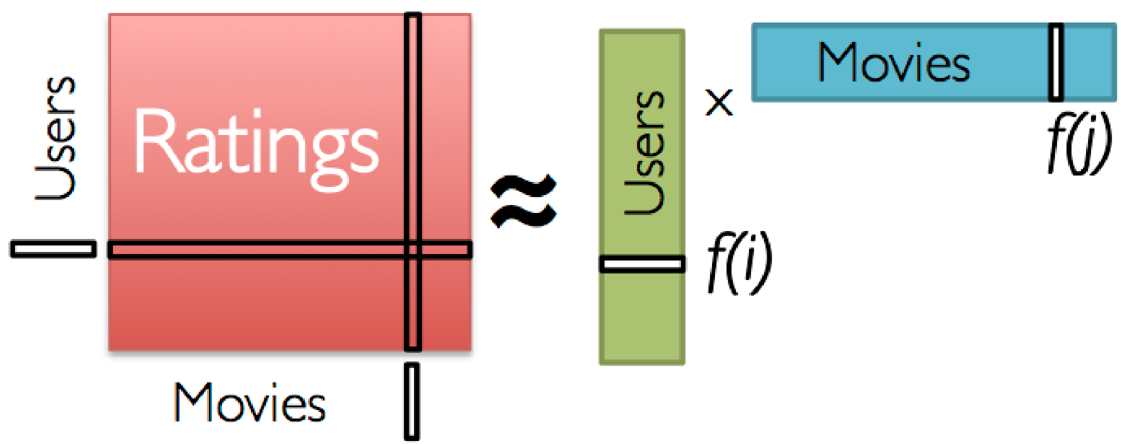
http://www.slideshare.net/MrChrisJohnson/collaborative-filtering-with-spark

Collaborative Filtering

- Users rate some subset of items
- Goal: predict how users will rate new items



Low-Rank Matrix Factorization:



https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

- **ALS – Alternating Least Squares**

ALS - Example


```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from pyspark import SparkConf, SparkContext
conf = SparkConf().setAppName("Testing Spark Commands”)
sc = SparkContext(conf = conf)
r1 = (1, 1, 1.0)
r2 = (1, 2, 2.0)
r3 = (2, 1, 2.0)
ratings = sc.parallelize([r1, r2, r3])
model = ALS.train(ratings, 10, 10)
# Evaluate the model on training data
testdata = sc.parallelize([(2,2),(1,1)])
predictions = model.predictAll(testdata)
predictions.collect()
 # Save and load model
model.save(sc, ”myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, ”myCollaborativeFilter")
#Try using the loaded model
sameModel.predictAll(testdata).collect()
```

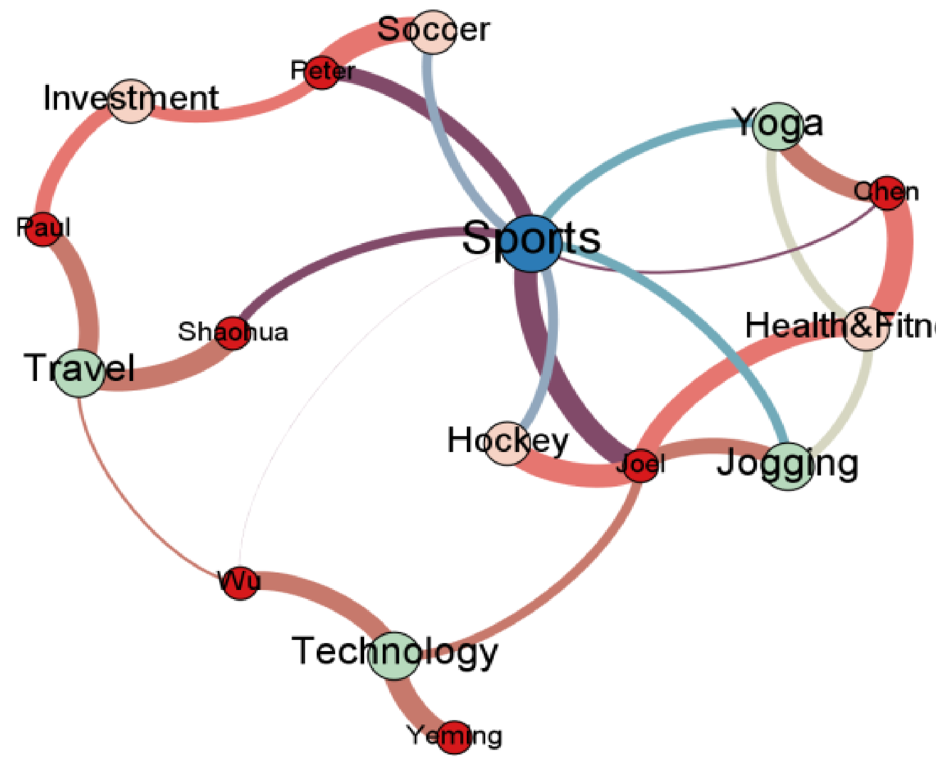
- #Output: [Rating(user=1, product=1, rating=1.0000230136195141), Rating(user=2, product=2, rating=0.9635523006463966 3)]
- [these numbers can change]

Graph Processing

The Graph phenomenon!

- Web Graph (Google)
- Social Graph (Facebook)
- Follower Graph (Twitter)
- Interest Graph (Pinterest!)
- Music/Location/Food Graphs

- Available in Scala and Java
- (Not yet in PySpark)



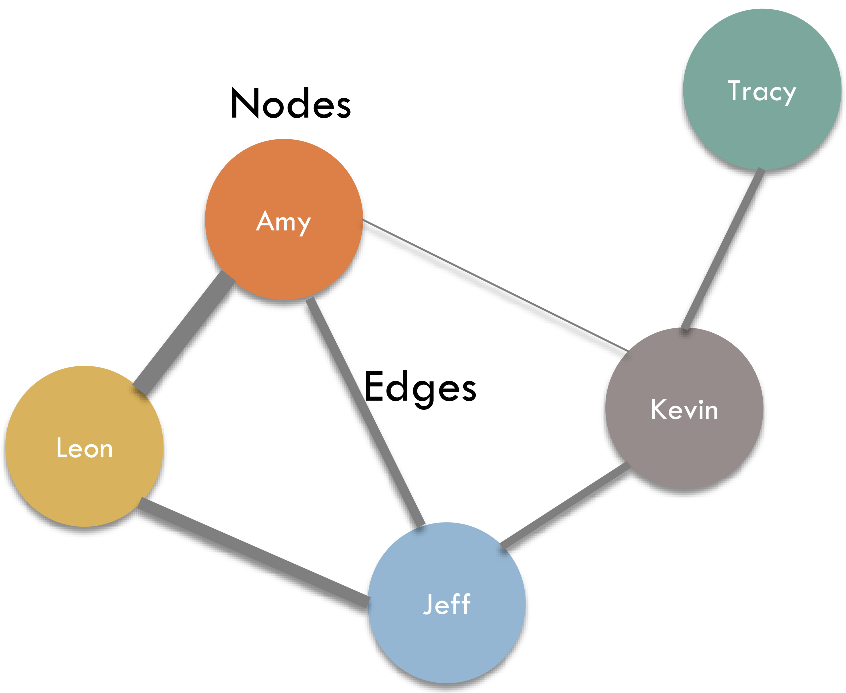
Graph Representation

	Friend				
User	Amy	Kevin	Jeff	Tracy	Leon
Amy		0.8	1.7		4
Kevin	0.8		1.3	1	
Jeff	1.7	1.3			2.2
Tracy		1			
Leon	4		2.2		

Matrix

Key	*Value*
Amy	*(Kevin,0.8), (Jeff, 1.7) (Leon,4)*
Kevin	*(Amy,0.8), (Jeff,1.3), (Tracy,1)*
Jeff	*(Amy, 1.7), (Kevin,1.3), (Leon,2.2)*
Tracy	*(Kevin,1)*
Leon	*(Amy,4), (Jeff,2.2)*

Adjacency List



Graph (undirected)

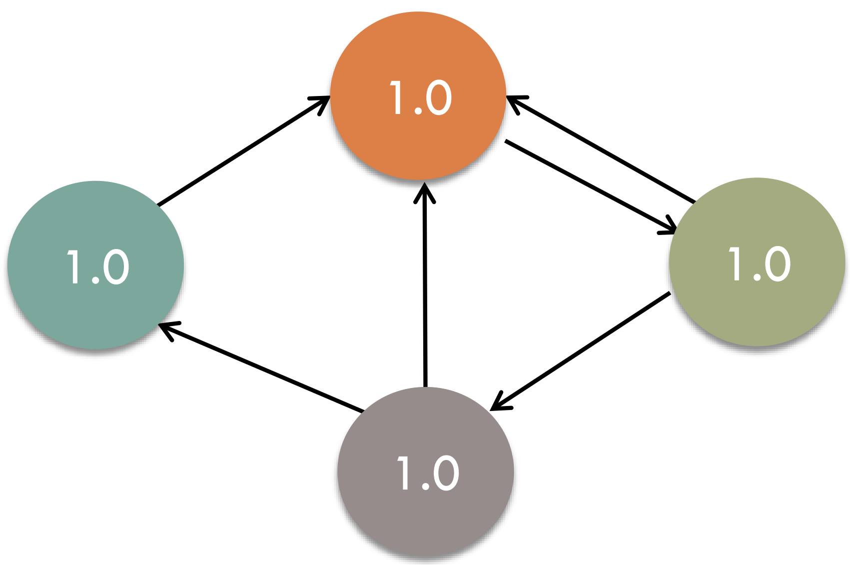
PageRank Algorithm

- PageRank gives web pages a ranking score on links from other pages
  - Links from many pages ➔ high rank
  - Link from a high-rank page ➔ high rank

Graph (directed)

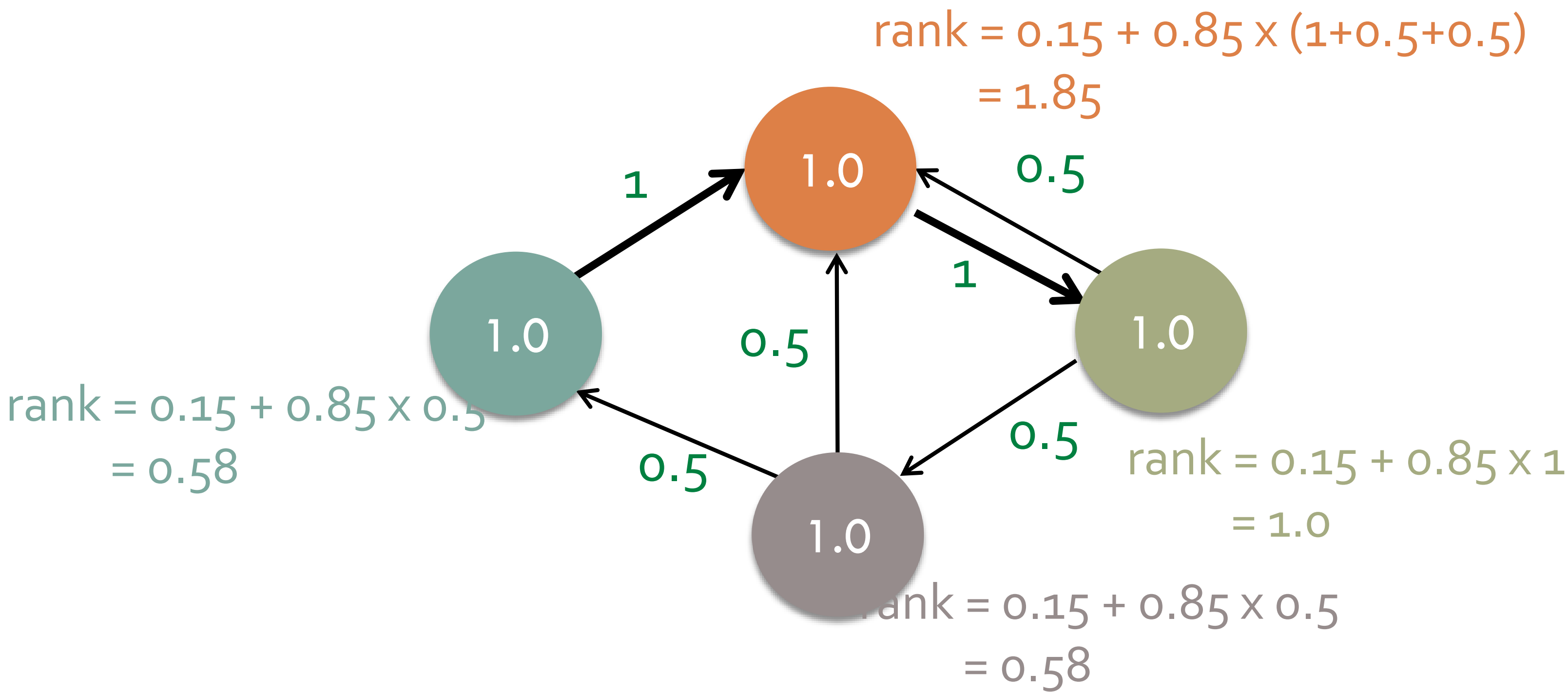
PageRank Explained – Initial State

1. Start each page at a rank of 1



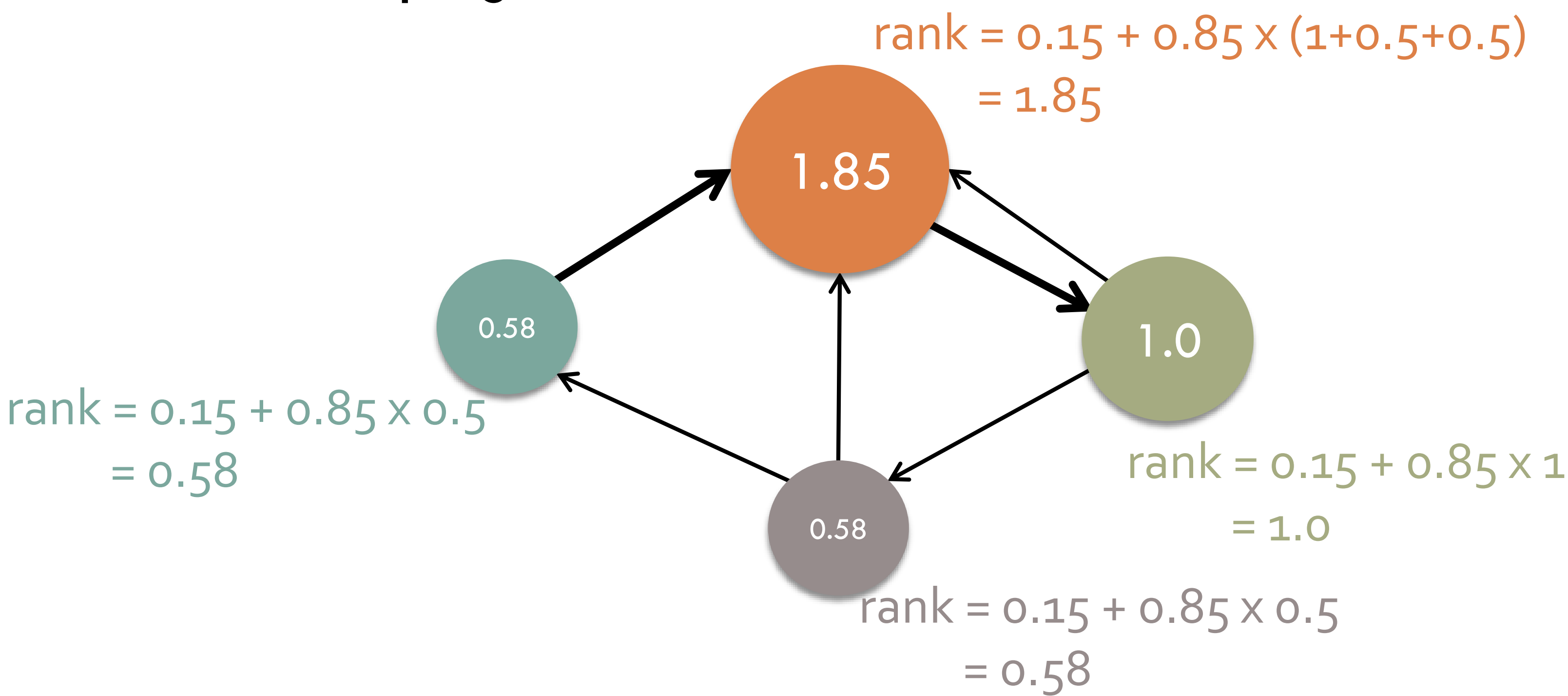
PageRank Explained – Iteration 1

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p \,/\, neighbors_p$ to its neighbors



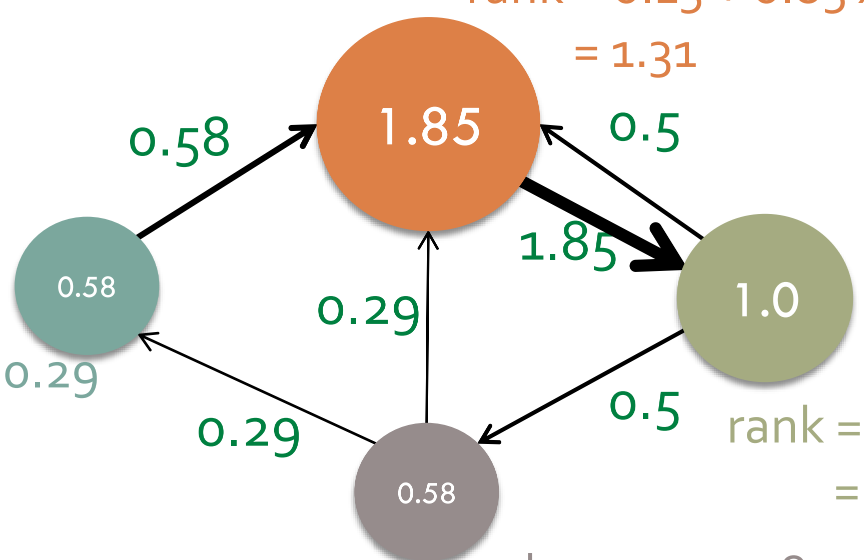
PageRank Explained – Iteration 1

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p \,/\, neighbors_p$ to its neighbors
3. **Set each page's rank to 0.15 + 0.85 × contribs**



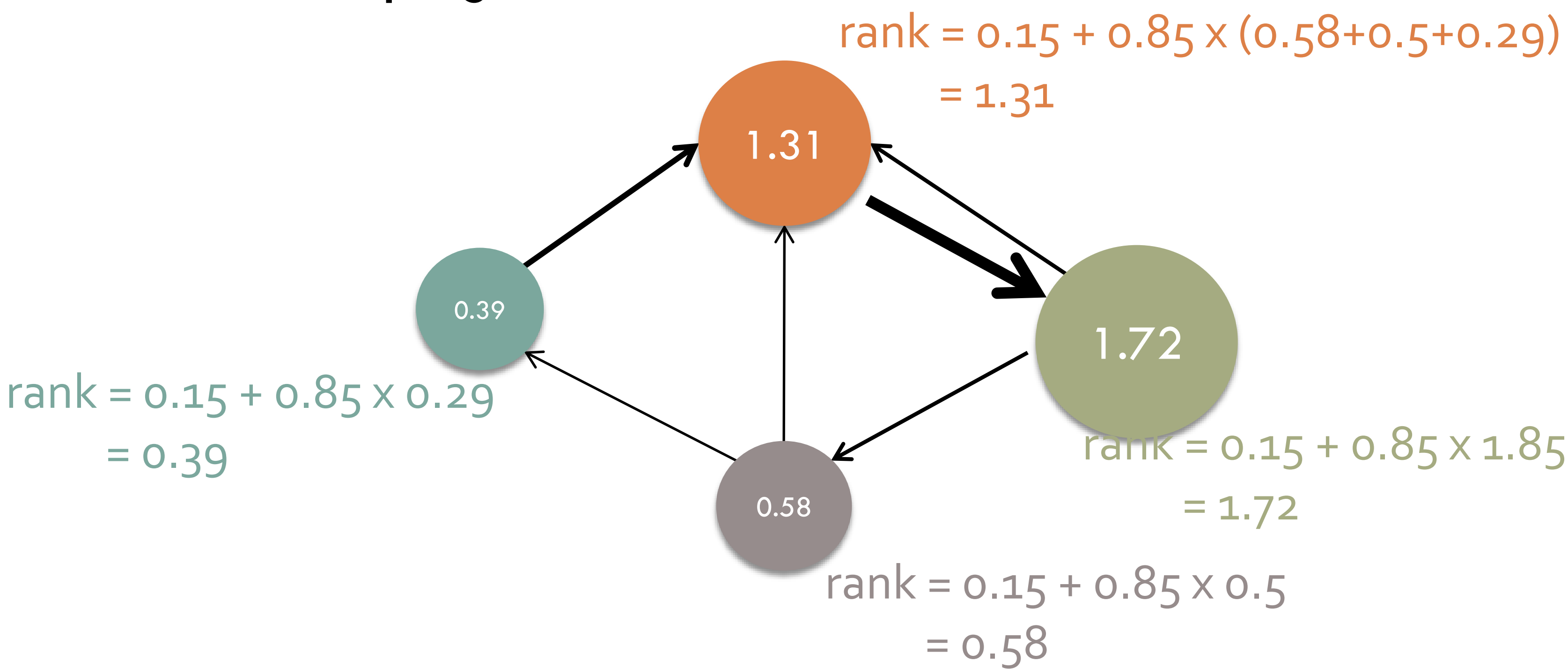
PageRank Explained – Iteration 2

1. Start each page at a rank of 1
2. On each iteration, have page p contribute
   $contrib_p = rank_p \,/\, neighbors_p$ to its neighbors
3. Set each page's rank to 0.15 + 0.85 × contribs



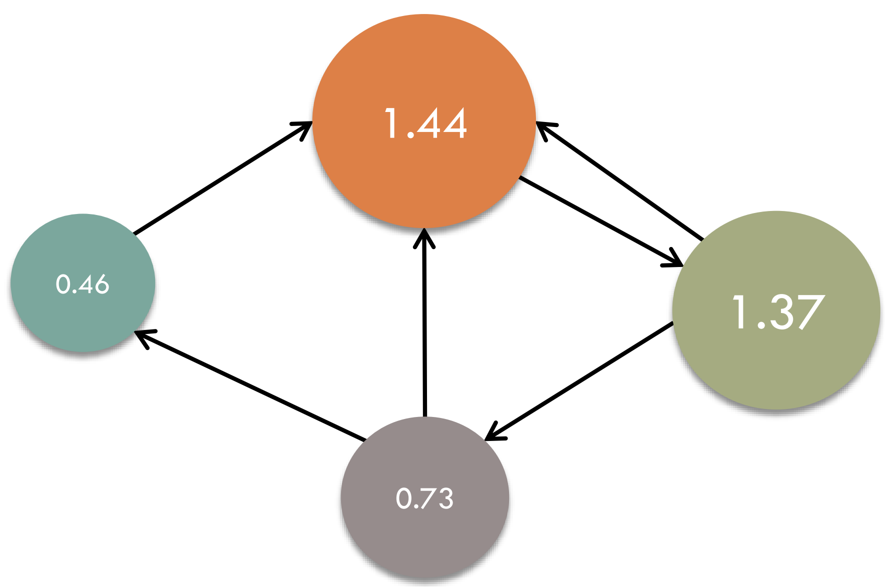
PageRank Explained – Iteration 2

1. Start each page at a rank of 1
2. On each iteration, have page p contribute $contrib_p = rank_p / neighbors_p$ to its neighbors
3. **Set each page's rank to 0.15 + 0.85 × contribs**



PageRank Explained – Final State

1. Start each page at a rank of 1
2. On each iteration, have page p contribute
   $contrib_p = rank_p / neighbors_p$ to its neighbors
3. **Set each page's rank to $0.15 + 0.85 \times$ contribs**



PagrRank using GraphFrames

- from graphframes import *
- v = sqlContext.createDataFrame([
  ("a", "Alice", 34),
  ("b", "Bob", 36),
  ("c", "Charlie", 30),
  ], ["id", "name", "age"])

- # Create an Edge DataFrame with "src" and "dst" columns
- e = sqlContext.createDataFrame([
  ("a", "b", "friend"),
  ("b", "c", "follow"),
  ("c", "b", "follow"),
  ], ["src", "dst", "relationship"])
- # Create a GraphFrame
- g = GraphFrame(v, e)
- results = g.pageRank(resetProbability=0.15, tol=0.01)
- results.vertices().show()

PageRank – Example in Scala using GraphX API

- spark-shell [Command to Open Scala Spark shell]

```
import org.apache.spark.graphx._
val graph = GraphLoader.edgeListFile(sc, "/user/root/followers.txt")
// Run PageRank
val ranks = graph.pageRank(0.0001).vertices
// Join the ranks with the usernames
val users = sc.textFile("/user/root/users.txt").map { line =>
    val fields = line.split(",")
    (fields(0).toLong, fields(1))
  }
val ranksByUsername = users.join(ranks).map {
    case (id, (username, rank)) => (username, rank)
  }
// Print the result
println(ranksByUsername.collect().mkString("\n"))
```

- Output:
  - (justinbieber,0.15)
  - (matei_zaharia,0.7013599933629602)
  - (ladygaga,1.390049198216498)
  - (BarackObama,1.4588814096664682)
  - (jeresig,0.9993442038507723)
  - (odersky,1.2973176314422592)
- [Copy and paste commands into "spark-shell" console; Copy followers.txt and users.txt from D2L into hdfs]

PageRank Implementation

- PageRank with Pig
  - Using PageRank to Detect Anomalies and Fraud in Healthcare (Hortonworks Blog Post)
    - (PART1) http://hortonworks.com/blog/using-pagerank-detect-anomalies-fraud-healthcare/
    - (PART2) http://hortonworks.com/blog/using-pagerank-to-detect-anomalies-and-fraud-in-healthcare-part2/
    - (PART3) http://hortonworks.com/blog/using-pagerank-to-detect-anomalies-and-fraud-in-healthcare-part3/
- PageRank with Spark

Getting Started with Spark

- Install Spark Standalone on Linux
- Cloudera/HDP Distributions
  - http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html

Spark Documentation

Spark 1.0.0 Documentation: http://spark.apache.org/docs/latest/
Spark Programming Guide (Scala, Python): http://spark.apache.org/docs/latest/programming-guide.html#overview
Spark Cassandra Connector - DataStax (github)
Spark HBase - lighting Spark with HBase (link)
Spark MLlib Documentation (link)
Spark GraphX Documentation (link)
(Spark) Spark Cluster Mode Overview (link)
(Spark) Running Spark on EC2 (link)
(Cloudera) Pig is Flying: Apache Pig on Apache Spark (link)