DISTRIBUTED COMPUTING AND MAPREDUCE

CKME 134 – BIG DATA ANALYTICS TOOLS

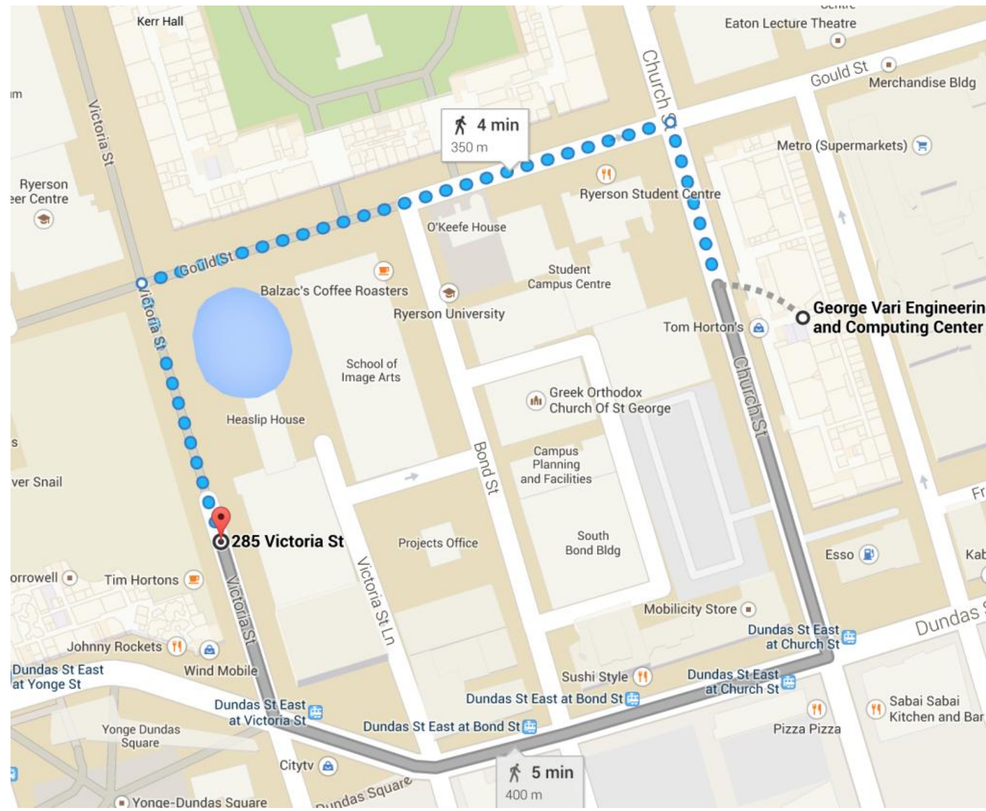
RYERSON UNIVERSITY

SPRING 2015

Instructor: Shaohua Zhang

General Course Information

- Instructor
  - Shaohua Zhang
  - Ryerson shaohua.zhang@ryerson.ca
  - Personal shaohua.zhang@live.com
- GA
  - Behjat Soltanifar
  - behjat.soltanifar@ryerson.ca
- Lectures
  - 6:30~8:30
  - ENGLG06
- Lab
  - 8:30~9:30
  - 285 Victoria St (403/404)
    - Take the elevator to 4FL



Course Outline (*subject to change*)

1. Intro to Big Data
2. **Distributed Computing and MapReduce**
3. Hadoop Ecosystem
4. Intro to Hive
5. Pig
6. Advanced Pig
7. Hadoop Performance Optimization
8. Big Data Use Cases: Location Intelligence and Marketing Analytics
9. Big Data Use Cases: Recommendation Engine and Computational Advertising
10. Hadoop In Action: Building Data Pipelines
11. Beyond Hadoop: Spark
12. Beyond Hadoop: Real-Time Analytics

Assignment Schedule

Date	*Out*	*In*
Assignment 1	***Jan 12***	***No Due Date***
Assignment 2	***Hive***	
Assignment 3	***Pig***	
Assignment 4	***Data Pipeline***	

Lecture1 Recap

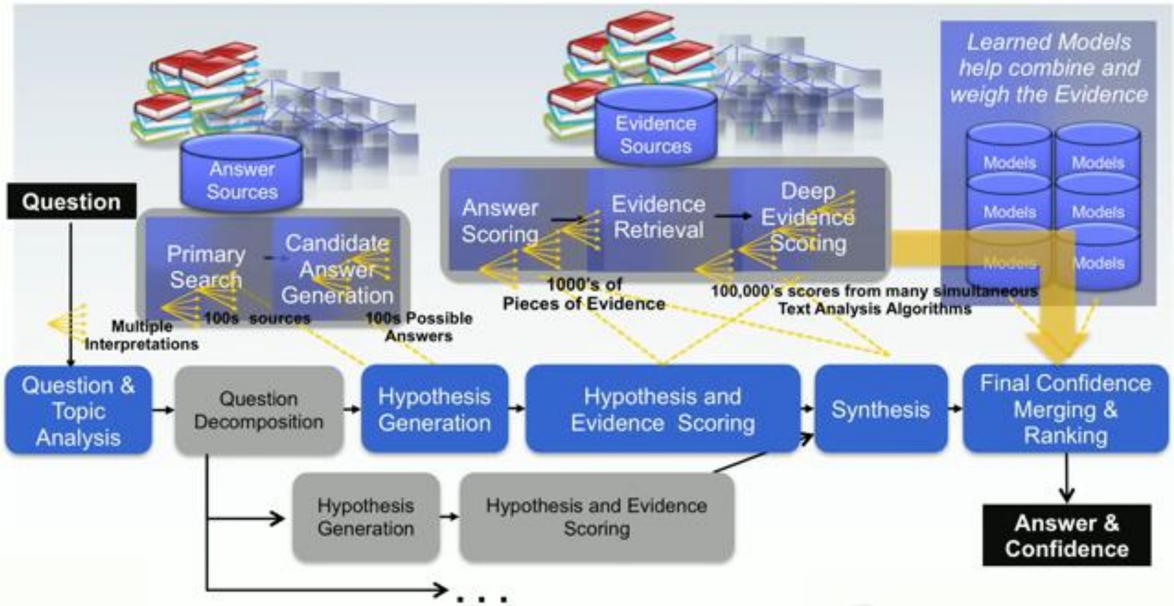
Course Blackboard
Lecture 1 Review
Watson and future of analytics
Questions from students

Lecture 1 Recap - Review

1. Big Data Introduction
2. Big Data Use Cases
3. Data Analytics Tooling
4. Big Data Job Market
5. Big Data Challenges
   - file://localhost/Users/DSinmotion/Dropbox/Startup/RyersonToolsCourse/Winter 2015/Session1-Introduction-to-big-data-analytics-tools.pptx - 43. 4. Big Data Challenges

Session 1 Recap – Watson and Future of Analytics

- DeepQA Project – The Jeopardy Game
- Future of analytics
  - Computers will replace human to some extent
    - Watson
    - Deep Learning
  - Machine learning automation
    - Enterprise Miner, KXEN etc.
    - Vincent Granville → consulting (analytics automation)
- We're still in the early days
  - Human judgment is still critical
  - We need many more Watson like projects to make it real



Building Watson: An Overview of the DeepQA Project http://www.aaai.org/ojs/index.php/aimagazine/article/download/2303/2165
Building Watson: A Brief Overview of the DeepQA Project (youtube) https://www.youtube.com/watch?v=3G2H3DZ8rNc
Your cognitive future: How next-gen computing changes the way we live and work: http://www-935.ibm.com/services/us/gbs/thoughtleadership/cognitivefuture/

Session 1 Recap - Questions From You

- Students with advanced tools knowledge
  - Start building your data science portfolio
    - Build your github repo
    - Join open source projects
    - Kaggle – a place to polish your machine learning skills and learn from other great data scientists
    - Build something real

Session 1 Recap - Questions From You

- Students business/marketing background who are NOT interested in pursuing data scientist career
  - Being good with data is becoming increasingly important
    - Most MBA programs are opening data analytics courses
    - Orgs are becoming more and more data driven
  - Communities are building user-friendly big data platforms for you
    - SQL, scripting languages are your friends
  - Understanding basic stats/data mining concepts are very helpful when communicating with the data team

Big Data Use Case: News & Media

- Newspaper (Globe and Mail, NYT, Bloomberg)
  - Content curation → editor vs. computer
    - Which piece of news should make the headline?
    - Content classification (topic modeling)
  - Lead generation
    - Where do you get new digital subscribers from?
      - Online advertising → Profile your existing user base and then bid on those user attributes
      - Traditional media → Where/when do you place your ads?
  - Revenue
    - Subscription model
    - Advertising – publisher revenue optimization
  - User tracking/understanding
    - How do you track your readers?
      - Browser session based
      - Email subscription
    - User demographic prediction
  - Content personalization
    - News recommendation
- Big data
    - Hadoop, text mining, topic modeling, recommendation etc.

Big Data Use Case: News & Media

- Bloomberg
  - News Recommender
    - Big data engineer *(15k ~ 20k salary)*
    - http://www.slideshare.net/Hadoop_Summit/shah-june27-425pmroom210av2
    - http://www.youtube.com/watch?v=nNAbBXc1EYo
- New York Times
  - Content digitization
    - The New York Times used Amazon's EC2 compute cloud to crunch through four terabytes of scanned archives from the paper, converting them to PDFs for the Web

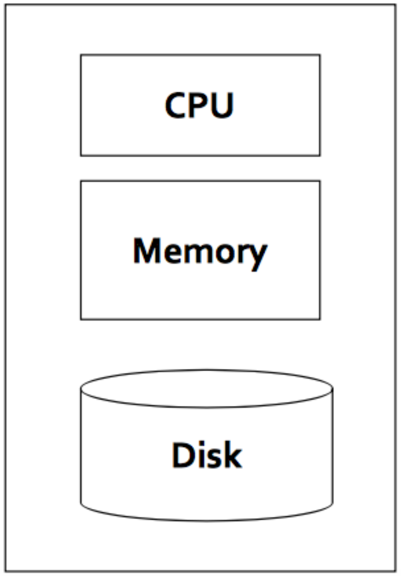
Lecture 2 - Outline

- Distributed Computing
- MapReduce
- Algorithms
  - Word Count
  - K-Means
- Beyond MapReduce

Distributed Computing

Single Node Architecture

- Traditionally, computation has been CPU bound
  - Complex computation on small data
- For decades, the primary push is to increase the computing power of a single machine

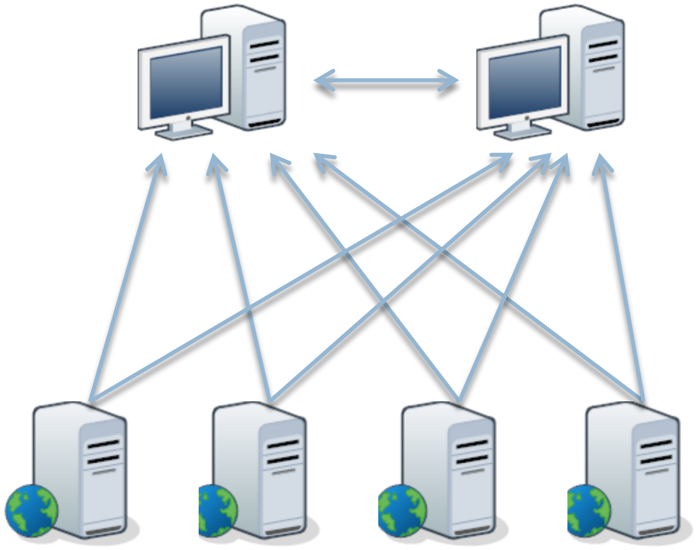


Scale Up vs. Scale Out

- Single Node Architecture
  - Scaling up advantage
    - Programming is easier than distributed computing
    - Faster processing on smaller data
  - Scale up disadvantage
    - Hardware cost
    - Scalability
- Advantage of scale-out systems
  - Scalability
  - Cost

Traditional Distributed Systems: Problems

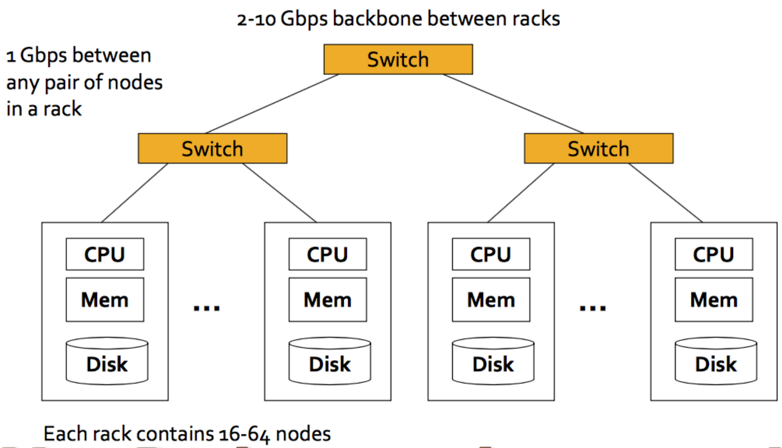
- Problems with traditional distributed systems:
  - Complex programming model
  - Network bandwidth is the bottleneck
  - It is difficult to deal with partial failures of the system
  - Typically at compute time, data is copied to the compute nodes
    - This doesn’t scale to today’s big data problems!



Data Becomes the Bottleneck

- Traditional distributed systems don't scale to today's Internet-scale data
- Getting data to the computer processor becomes the bottleneck
  - Disk I/O is slow
  - Network bandwidth is bottleneck
- Solution → moving computation to the data!

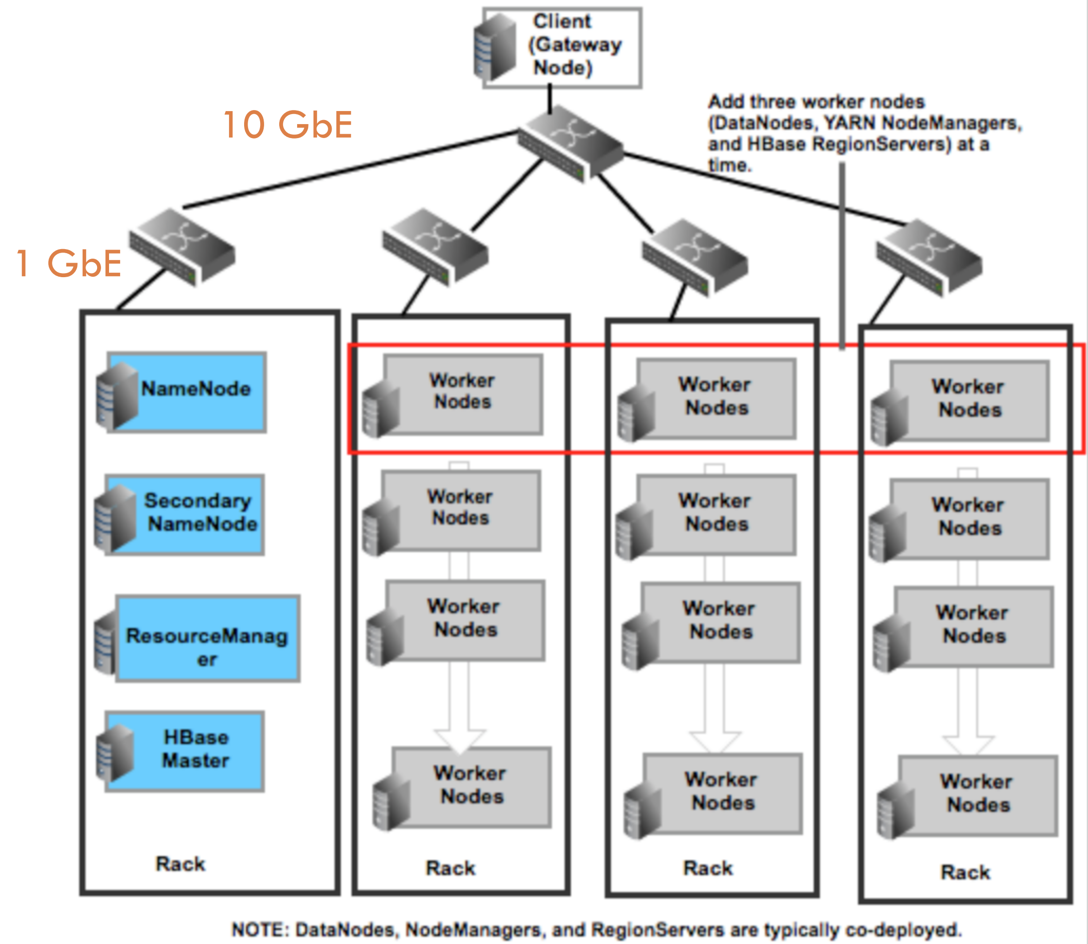
• *Internet*	◦ *2.5 exabytes ($2.5x10^{18}$) per day – 2012* ◦ *2.3 zettabytes ($2.3x10^{21}$) per day - 2014*
• *Facebook*	◦ *500+ terabytes per day* ◦ *100+ petabytes in a single Hadoop cluster*



MapReduce to the rescue!

Modern Distributed Computing Cluster

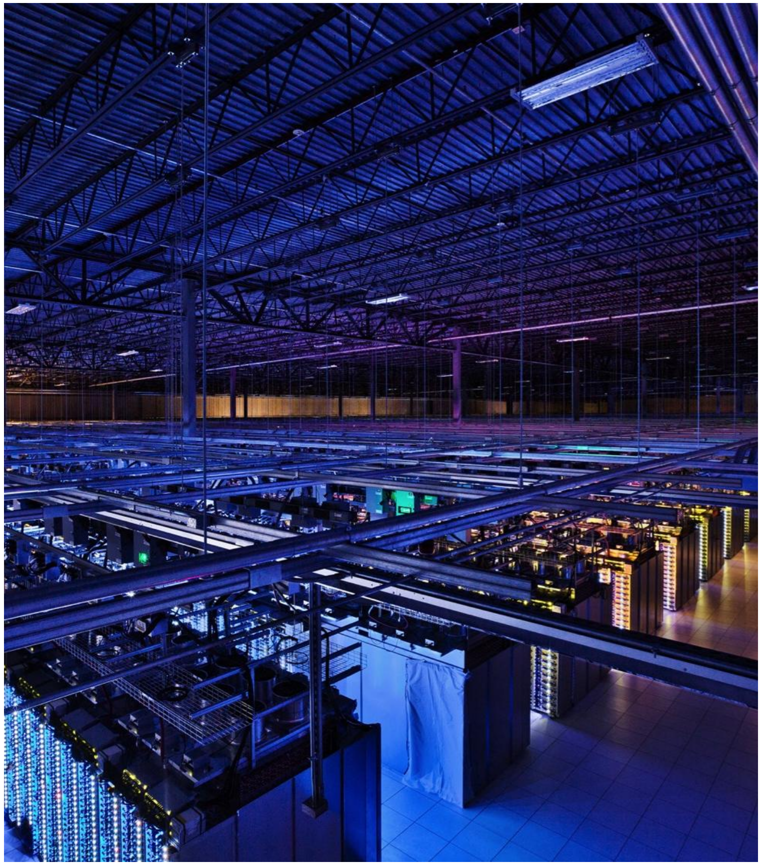
- Cluster architecture
  - A medium-to -large Hadoop cluster consists of a two-level or three-level architecture built with rack-mounted servers. Each rack of servers is interconnected using a 1 Gigabyte Ethernet switch. Each rack-level switch is connected to a cluster-level switch (which is typically a larger port-density 10GbE switch).



Stunning Photos Of Google's Massive Data Centers: http://www.forbes.com/pictures/edej45emjgl/up-above-the-massive-floor/

Big Data Made Possible

- Hardware
  - Big cluster of commodity machines at lower cost
    - Faster processor
    - Cheaper memory
    - Bigger hard drive space
    - Faster network bandwidth
- Software
  - Algorithms to allow parallel computing (map-reduce)



MapReduce

Setting the Expectation

- If your goal is to become a data analyst, you probably don't have to learn MapReduce programming
  - Alternatively, you need to be good with Pig/Hive
  - Still, it is important to under the M/R basics
- But if you want to become a Hadoop data architect, data engineer or research engineer or maybe data scientist who work with very large data…
  - You'll need to understand M/R programming patterns well
  - You'll still use Pig/Hive 80% of your time, but being able M/R allows you to do more complex data processing
    - Writing customized UDF (user-defined functions) in Pig/Hive also requires a good understanding M/R

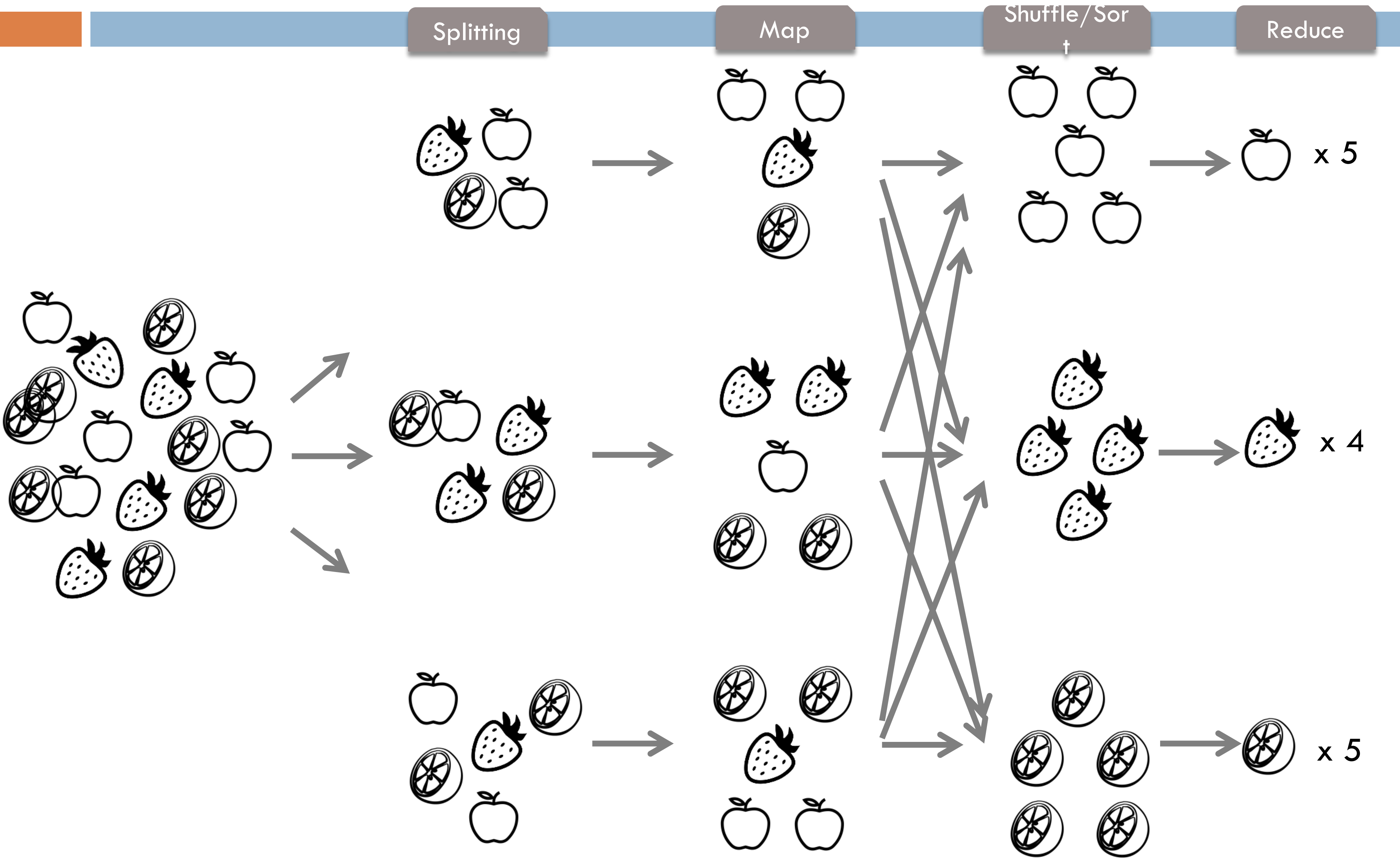
MapReduce

- *MapReduce is a computing model that decomposes large data manipulation jobs into individual tasks that can be executed in parallel across a cluster of servers*
- Each node processes data stored on that node
- Consists of two phases
  - Map
  - Reduce

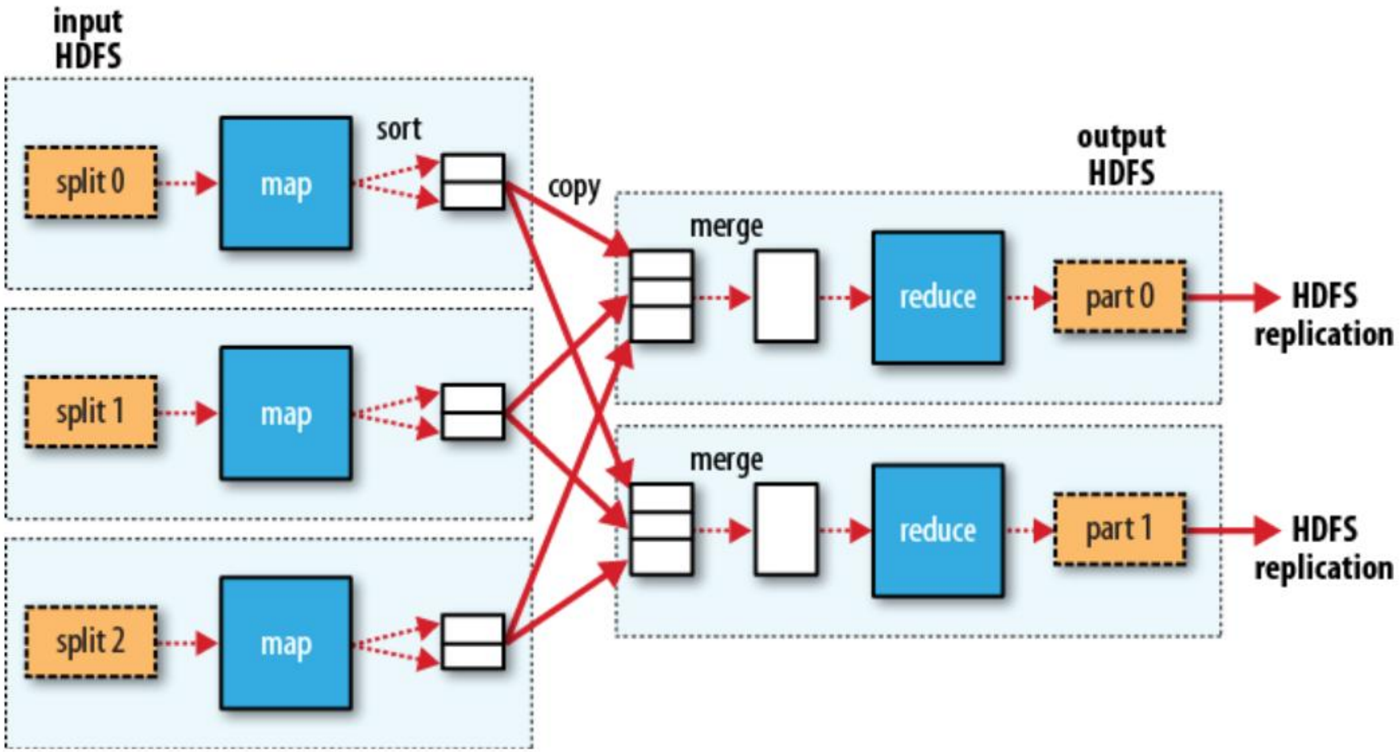
MapReduce

- Automatic parallelization and distribution
  - It makes M/R programming much easier
- Developer simply need to focus on writing the *map* and *reduce* functions
- M/R is written in Java
- It also supports Python Streaming
  - writing *map* and *reduce* function in python

MapReduce – Toy Example



MapReduce – Map & Reduce



Word Count Example

- Word Count is a classic programming example, used in many tutorials
- Counting word occurrences is widely used in many natural language processing related tasks
  - TF-IDF (Term Frequency-Inverse Document Frequency) is key input into many complex algorithms such as PageRank and document classification

Note: we will walk through TF-IDF in more detail in later lectures

Word Count Explained

- The WordCount program reads/scans through the document line by line
- It tokenizes/splits the line by delimiters (space, tab etc.)
- Each occurrence of a word/term will increment the corresponding word count by 1

I wish to wish the
wish you wish to
wish, but if you
wish the wish the
witch wishes, I
won't wish the wish
you wish to wish

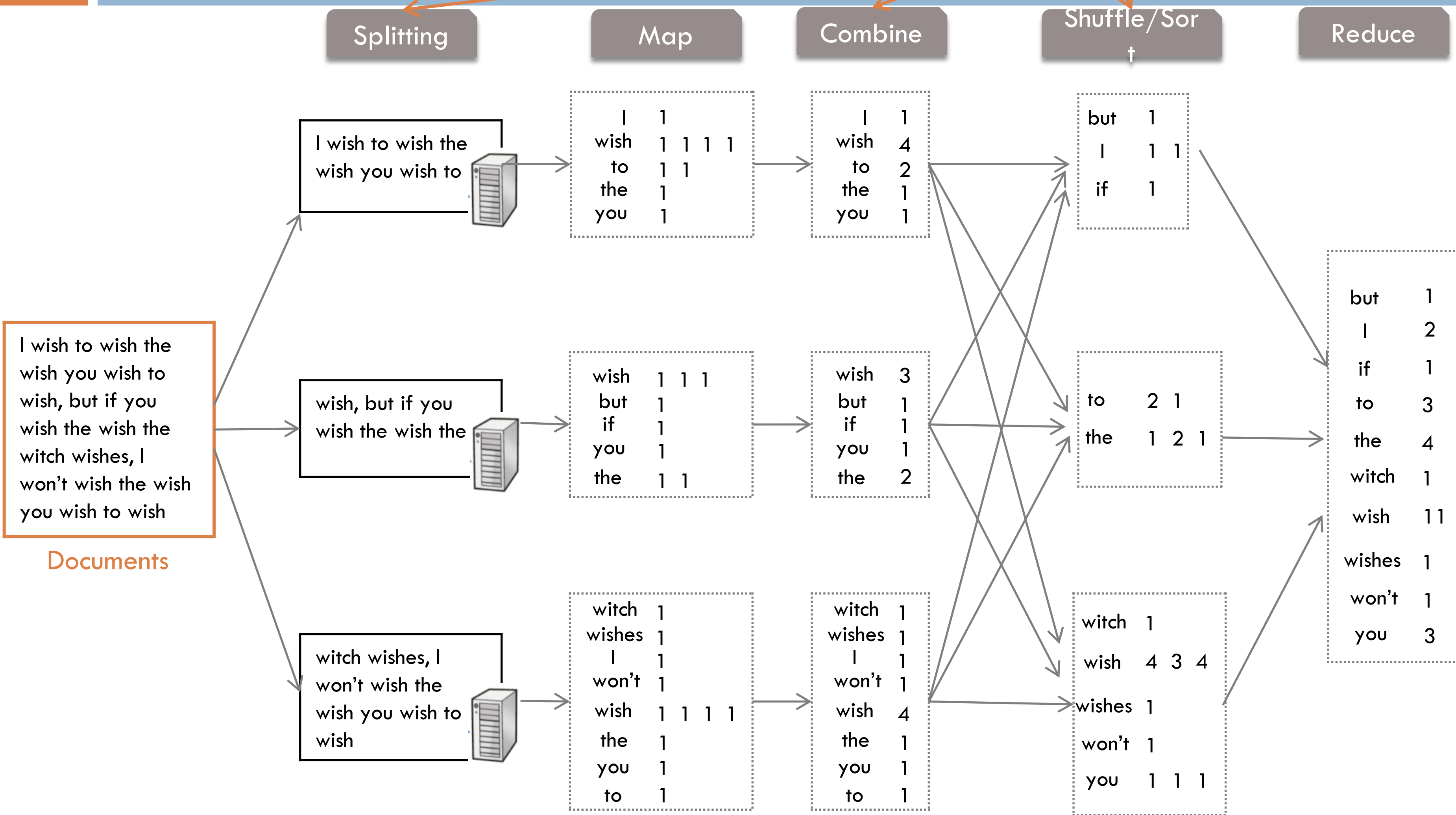
Document

I	--->	1 1
wish	--->	1 1 1 1 1 1 1 1 1 1 1
to	--->	1 1 1
the	--->	1 1 1 1
you	--->	1 1 1
but	--->	1
if	--->	1
witch	--->	1
wishes	--->	1
won't	--->	1

I	--->	2
wish	--->	11
to	--->	3
the	--->	4
you	--->	3
but	--->	1
if	--->	1
witch	--->	1
wishes	--->	1
won't	--->	1

Word Count – MapReduce

MapReduce handles these automatically for you!!



MapReduce

```java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

Hive

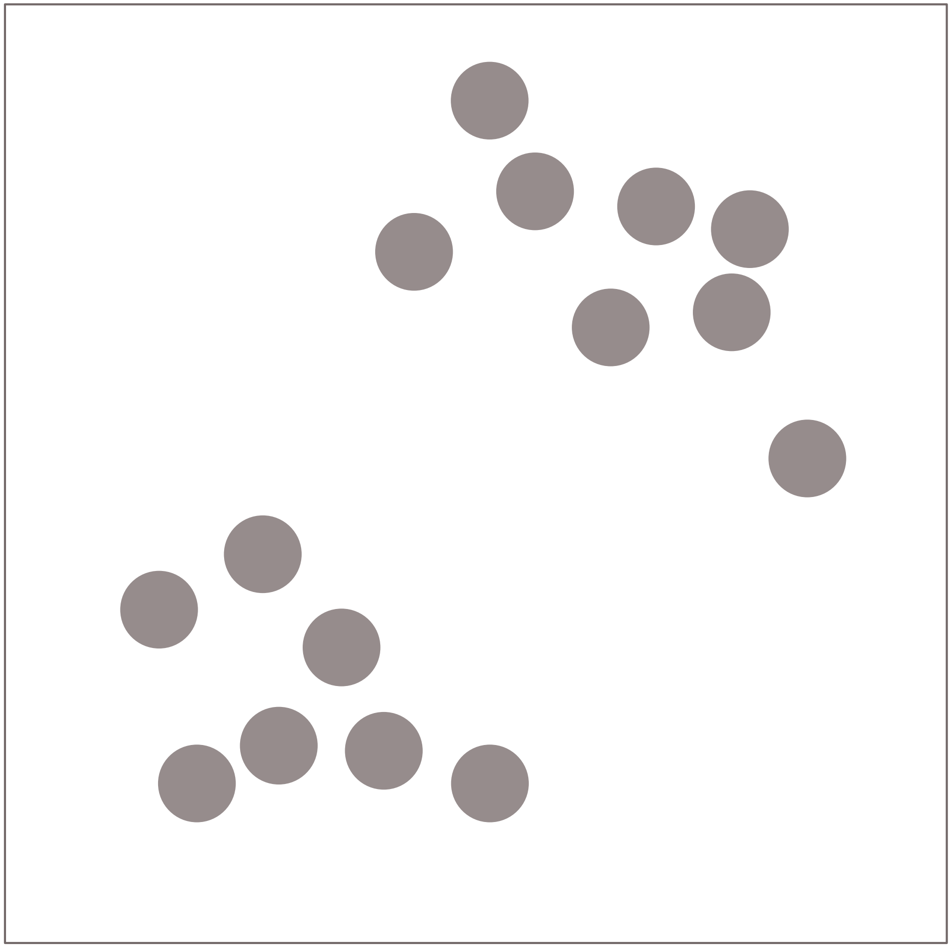
```sql
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

K-Means Clustering

- Clustering Use Cases
  - Customer value segmentation
    - High value customer
  - Behavior segmentation
    - Marketing
    - Sales
    - Product
  - Location clustering
  - Multi-stage algorithms
    - Stage 1 → clustering
    - Stage 2 → classification

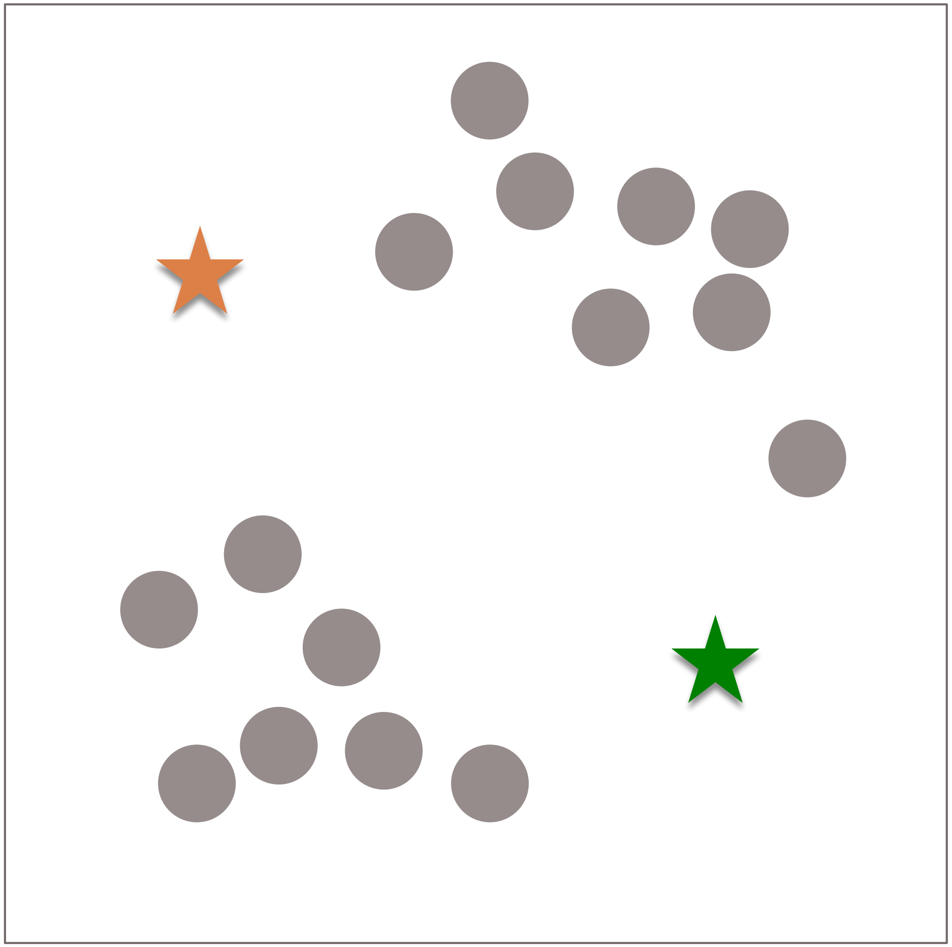
K-Means – Single Node

- 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



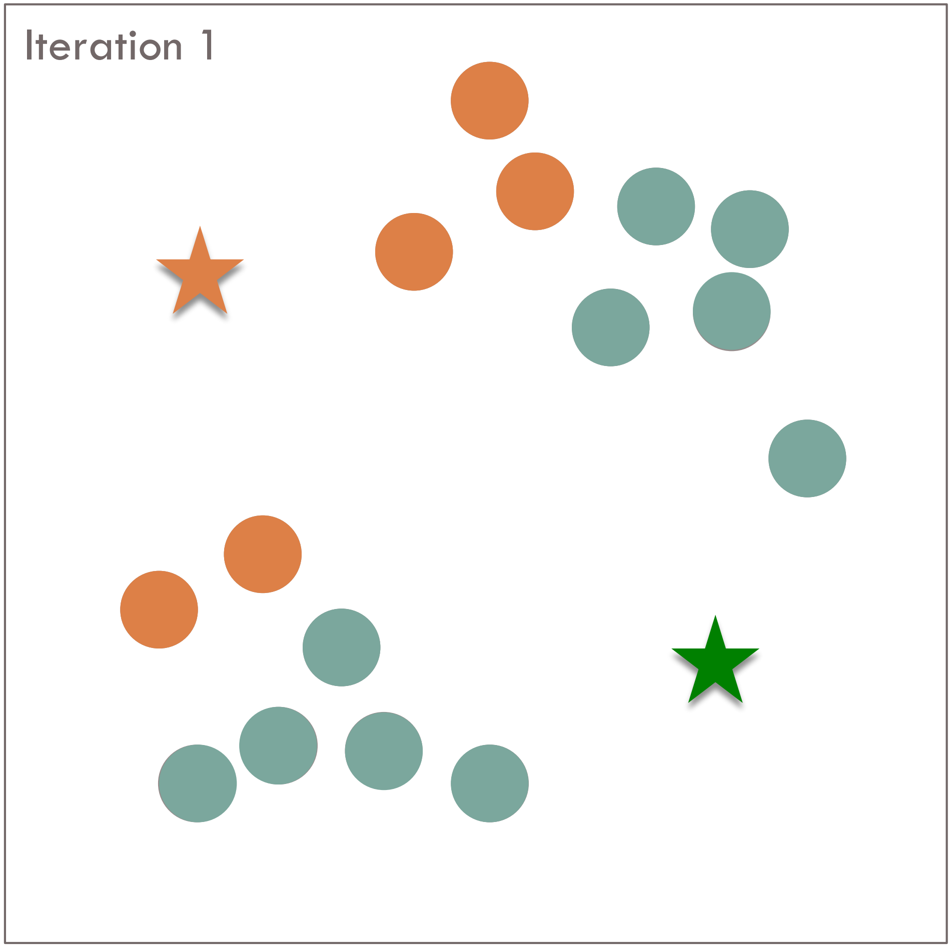
K-Means – Single Node

- 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



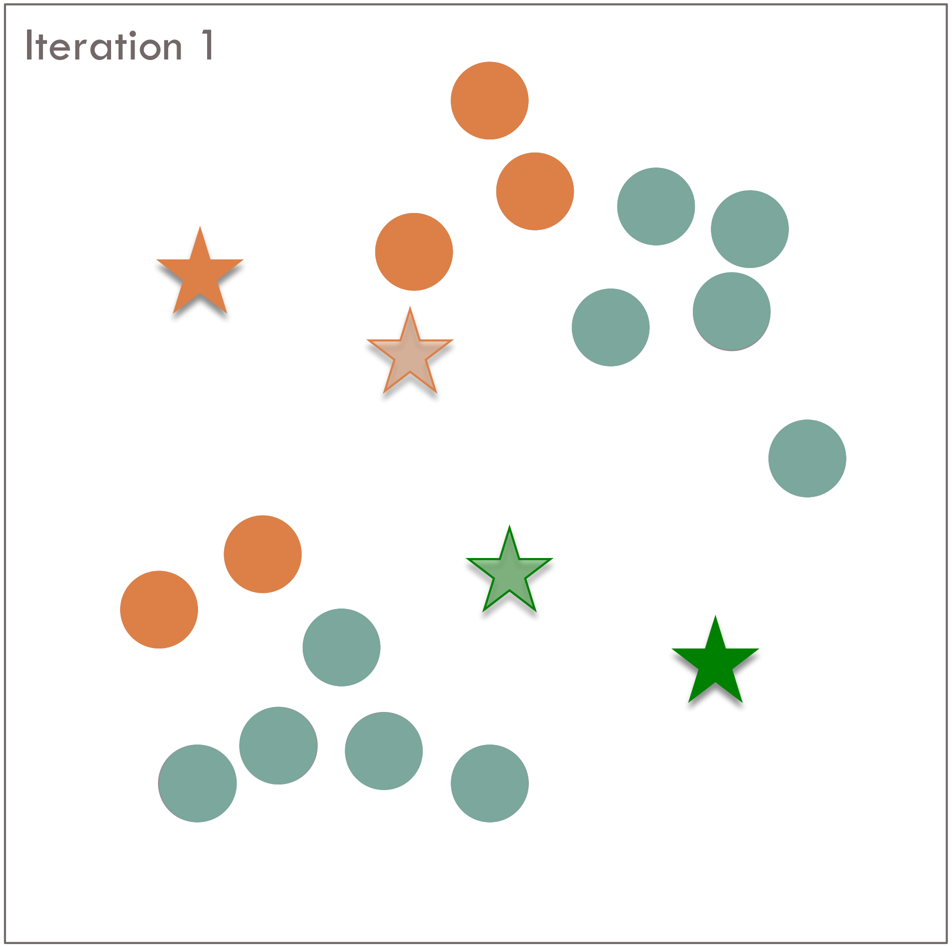
K-Means – Single Node

- 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



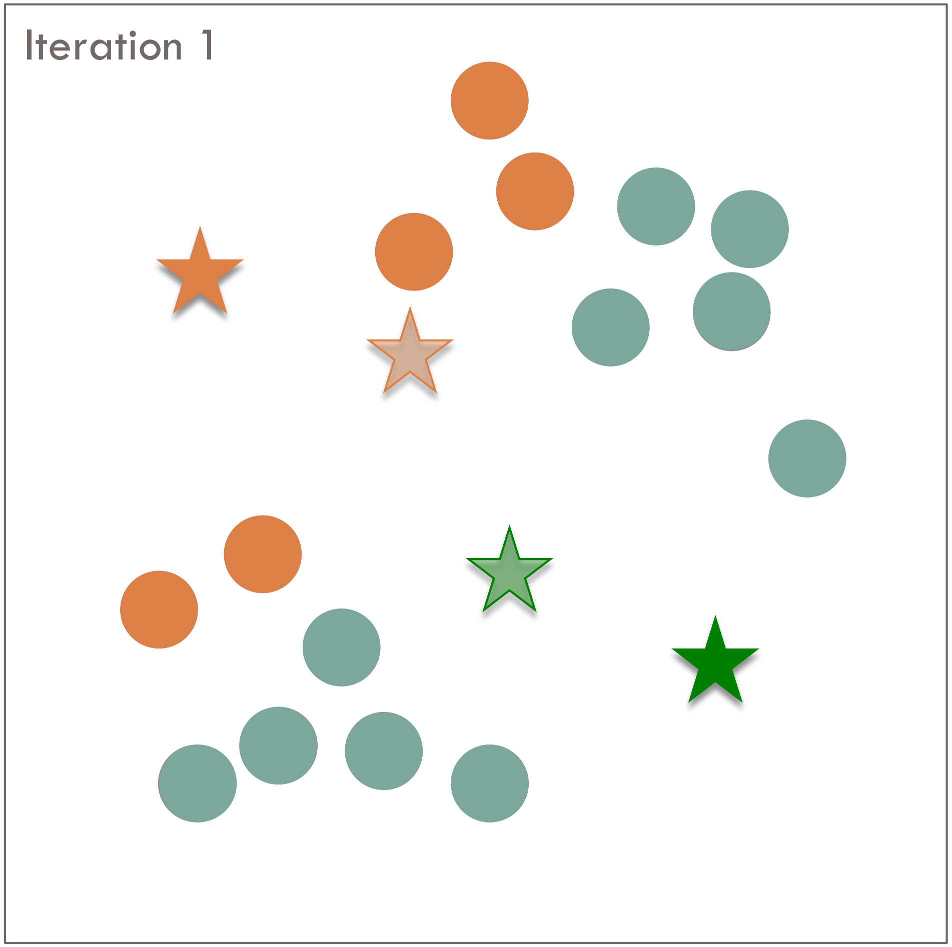
K-Means – Single Node

- 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

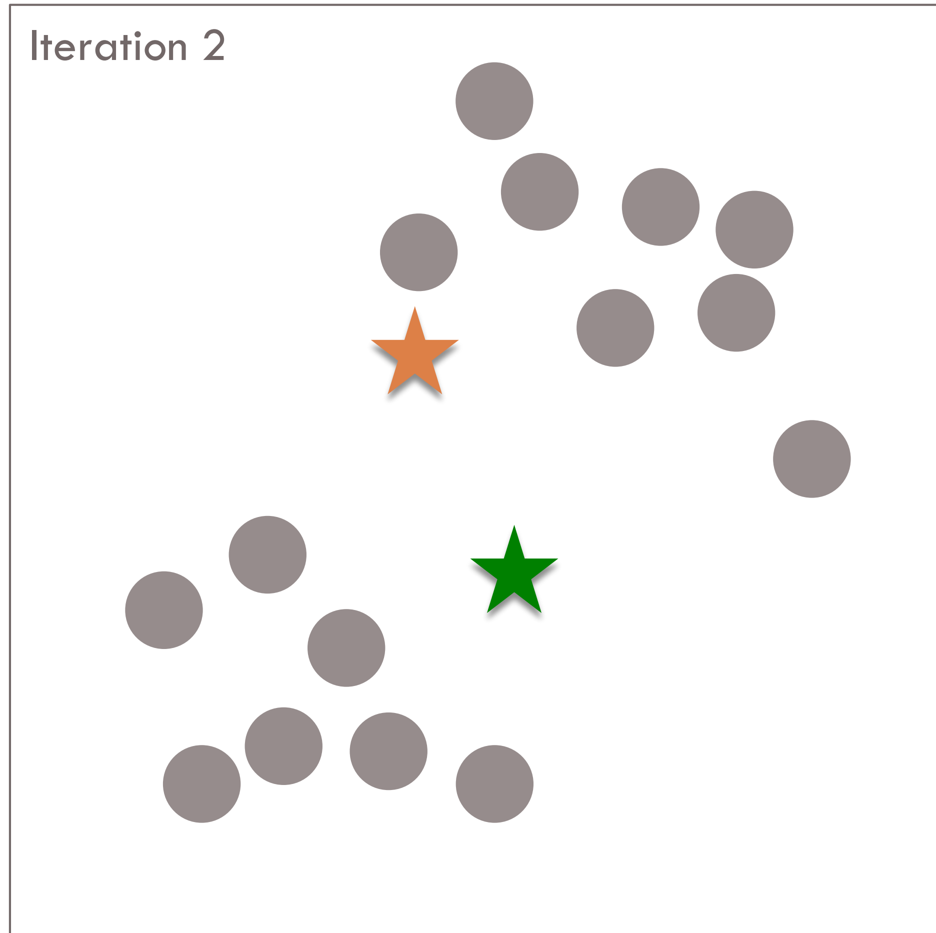
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average 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

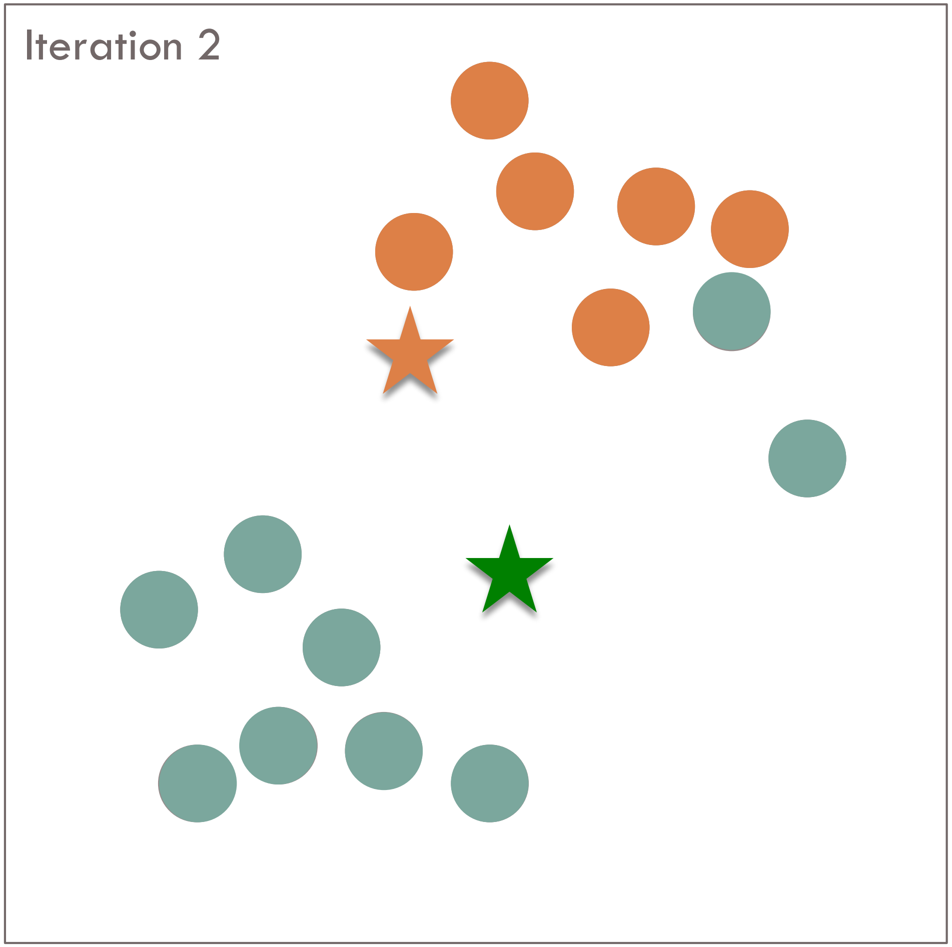
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average 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



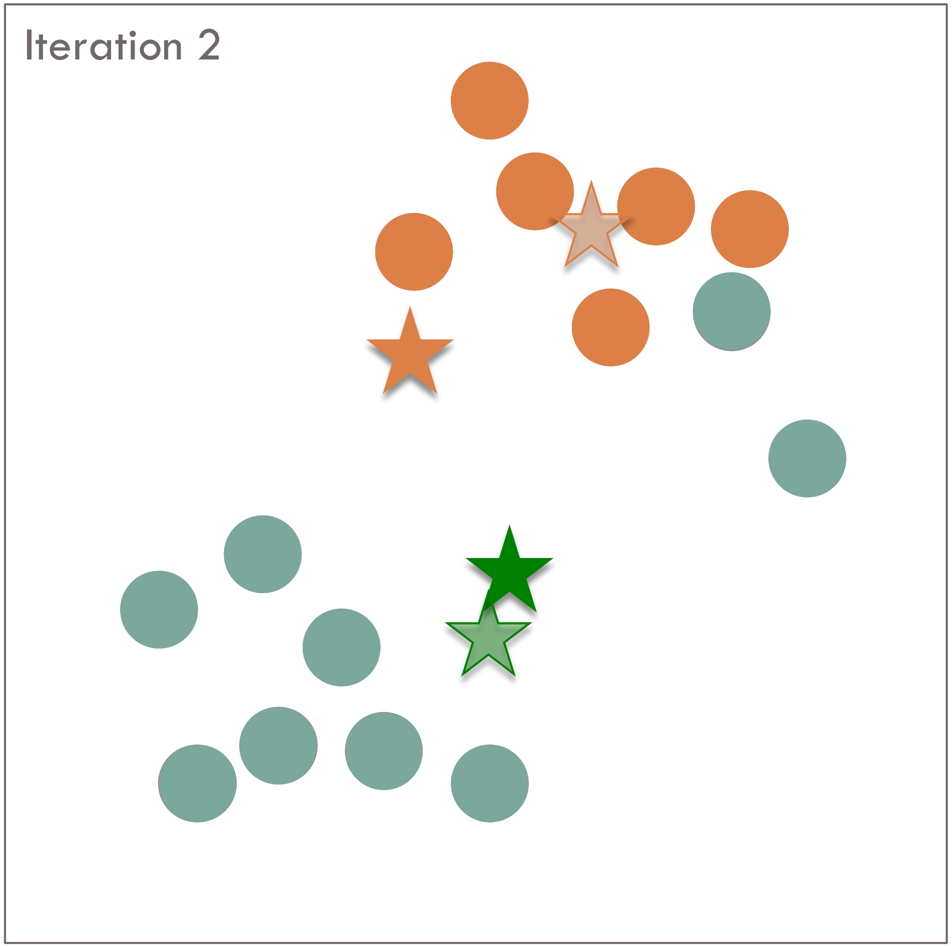
K-Means – Single Node

- 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



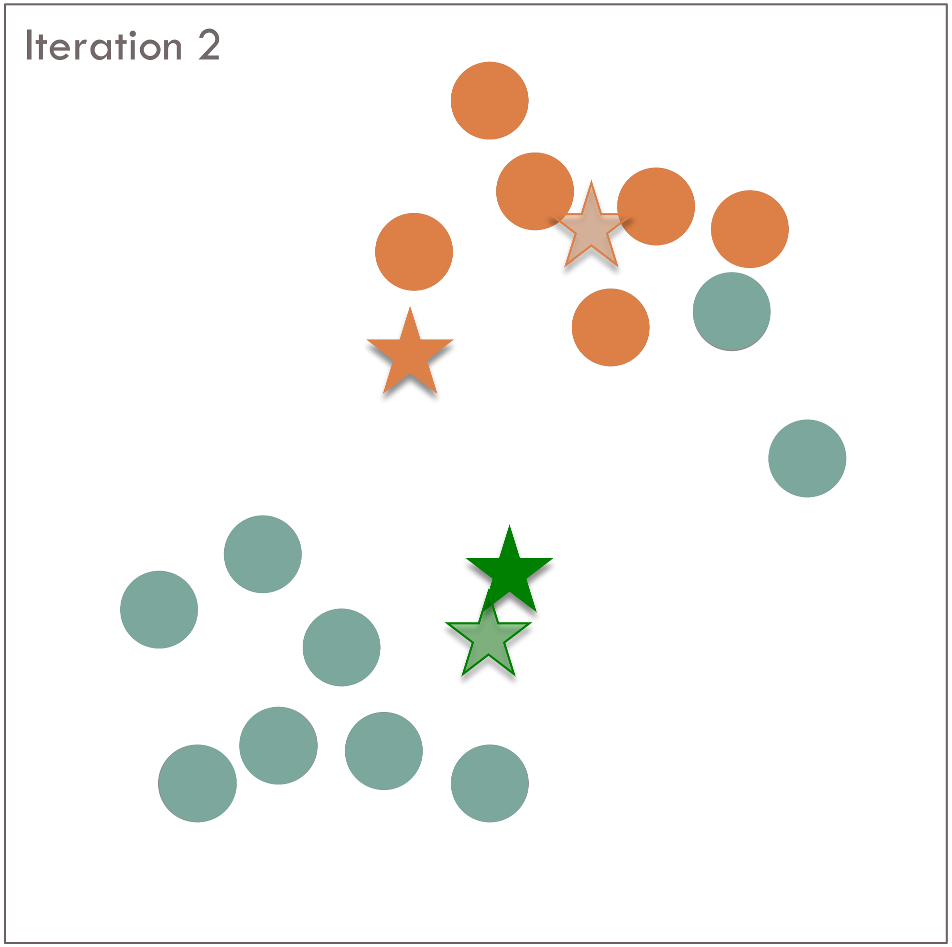
K-Means – Single Node

- 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

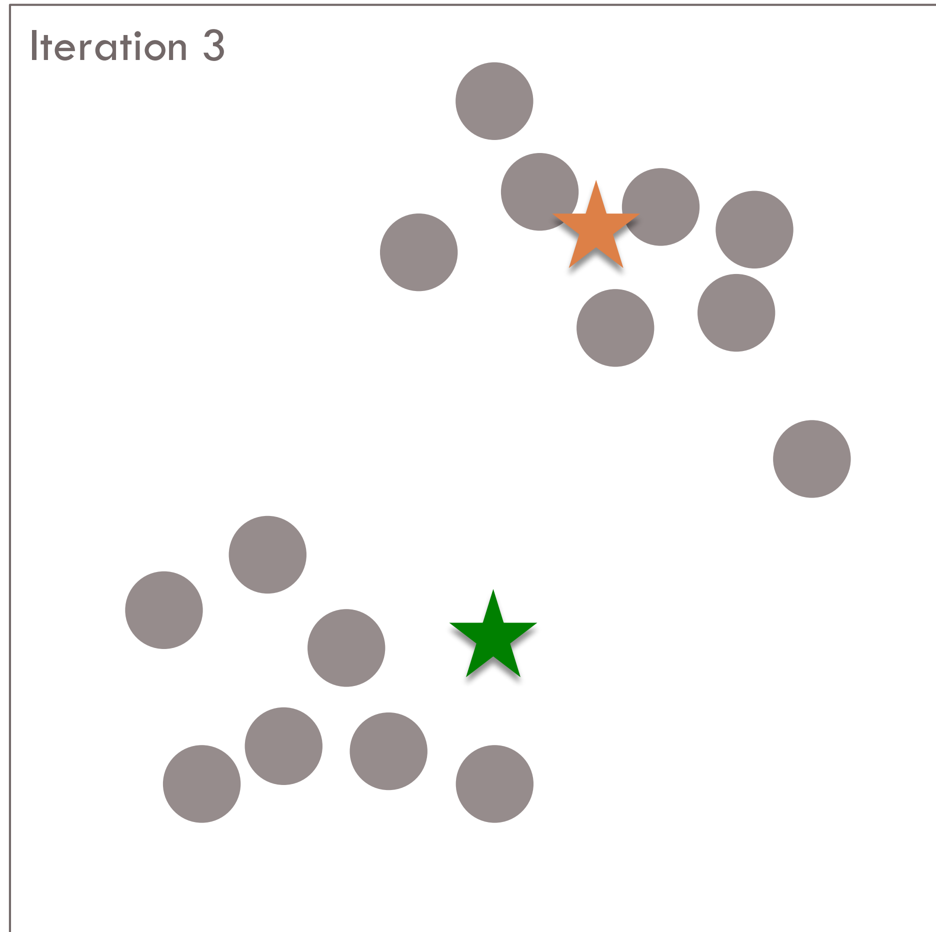
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average 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

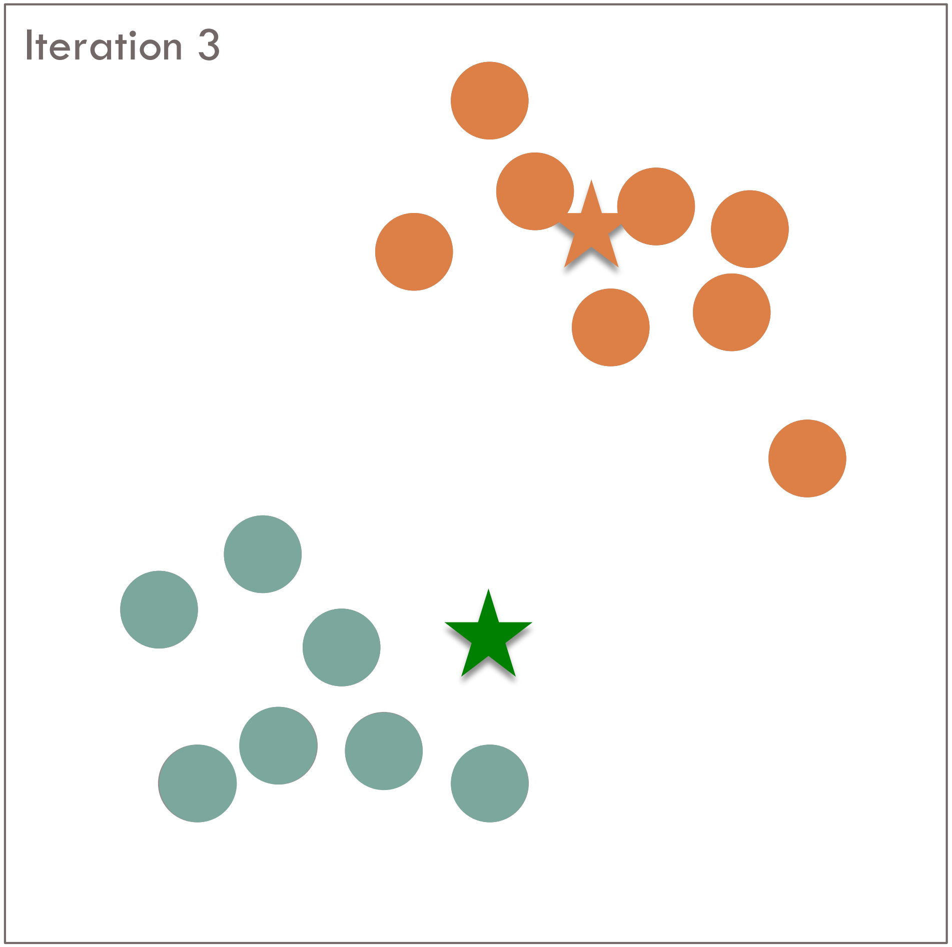
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average 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



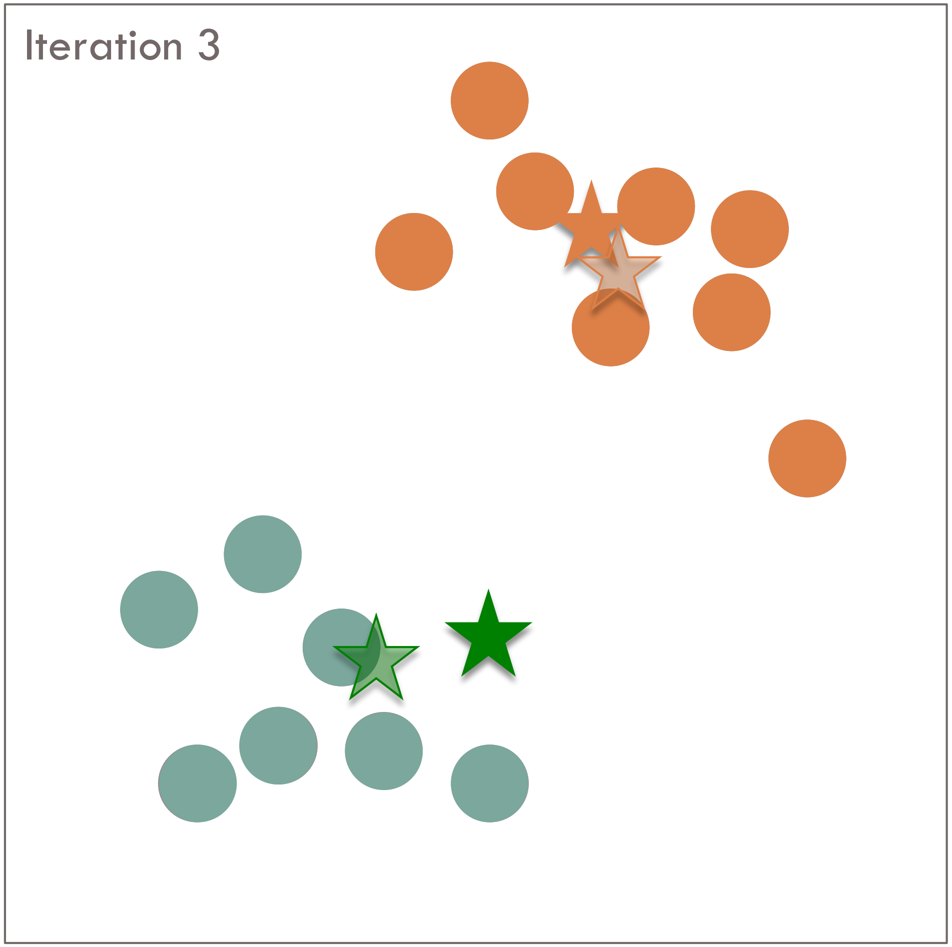
K-Means – Single Node

- 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

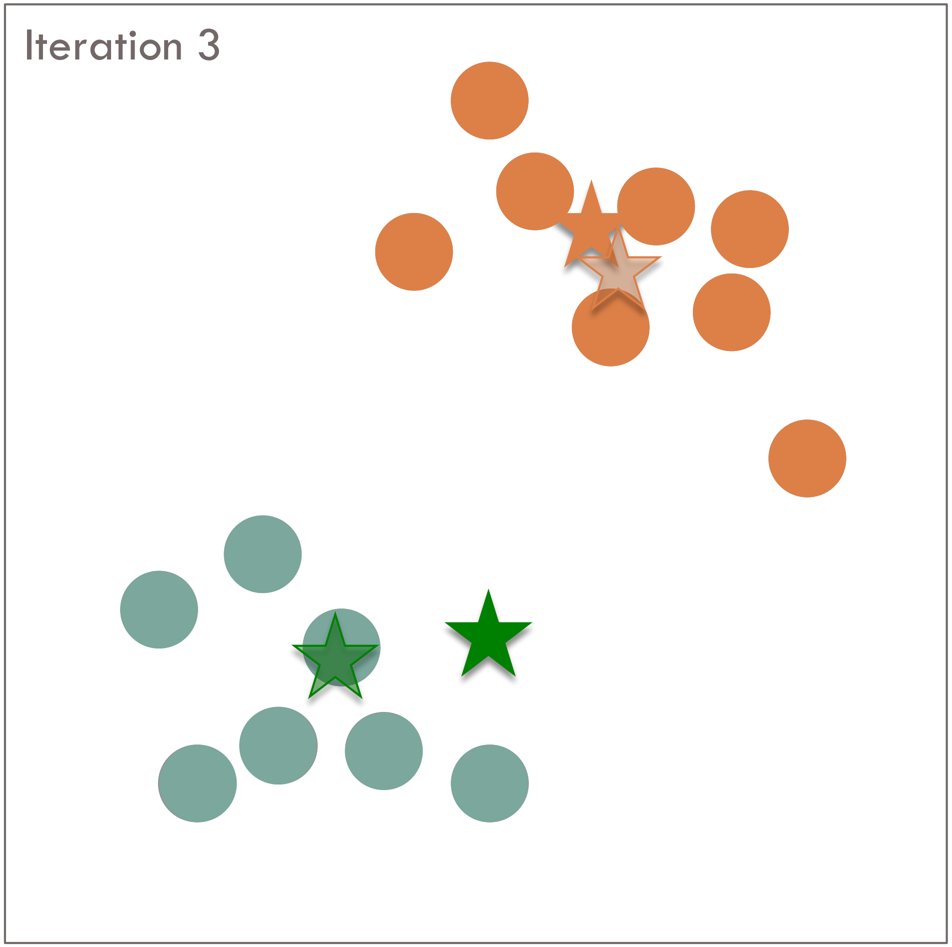
- Kmeans → Iterative algorithm until convergence
  1. Select K points at random as cluster centroids (centers)
  2. For each data point, assign it to the closest center
     - Now we formed K clusters
  3. For each cluster, re-compute the centers
     - E.g., in the case of 2D points →
       - X: average 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

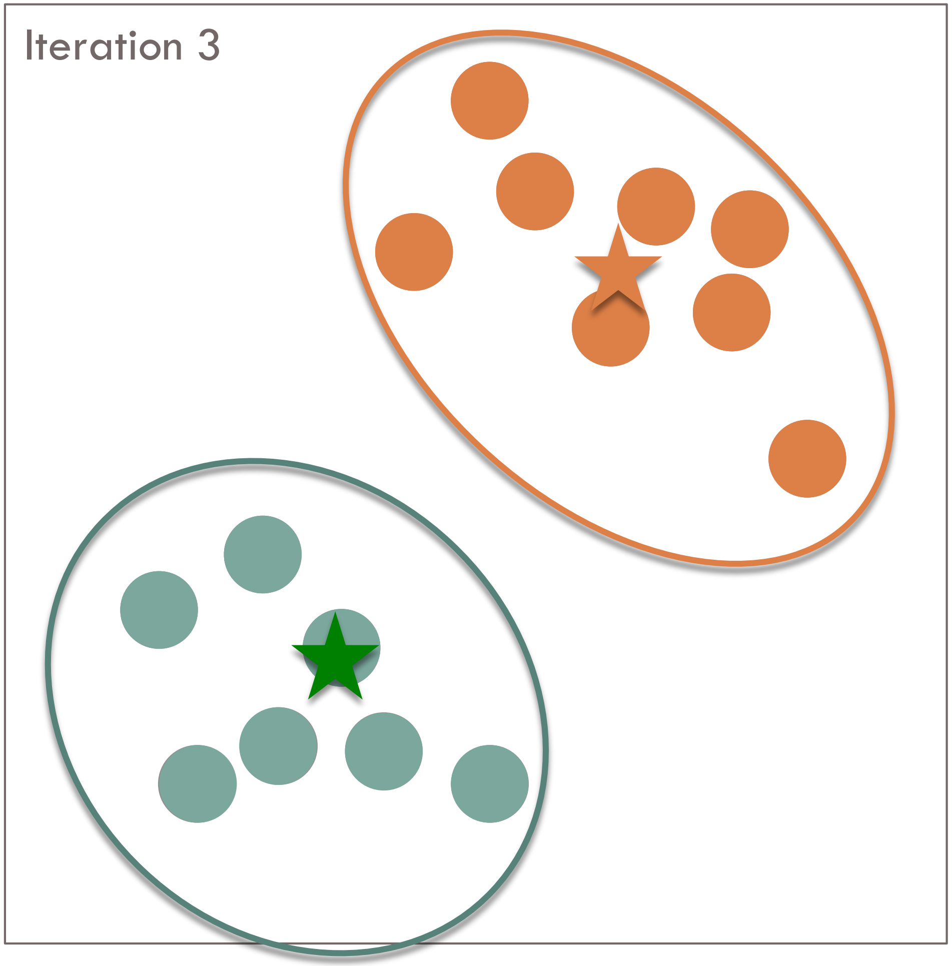
Meets convergence criteria



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

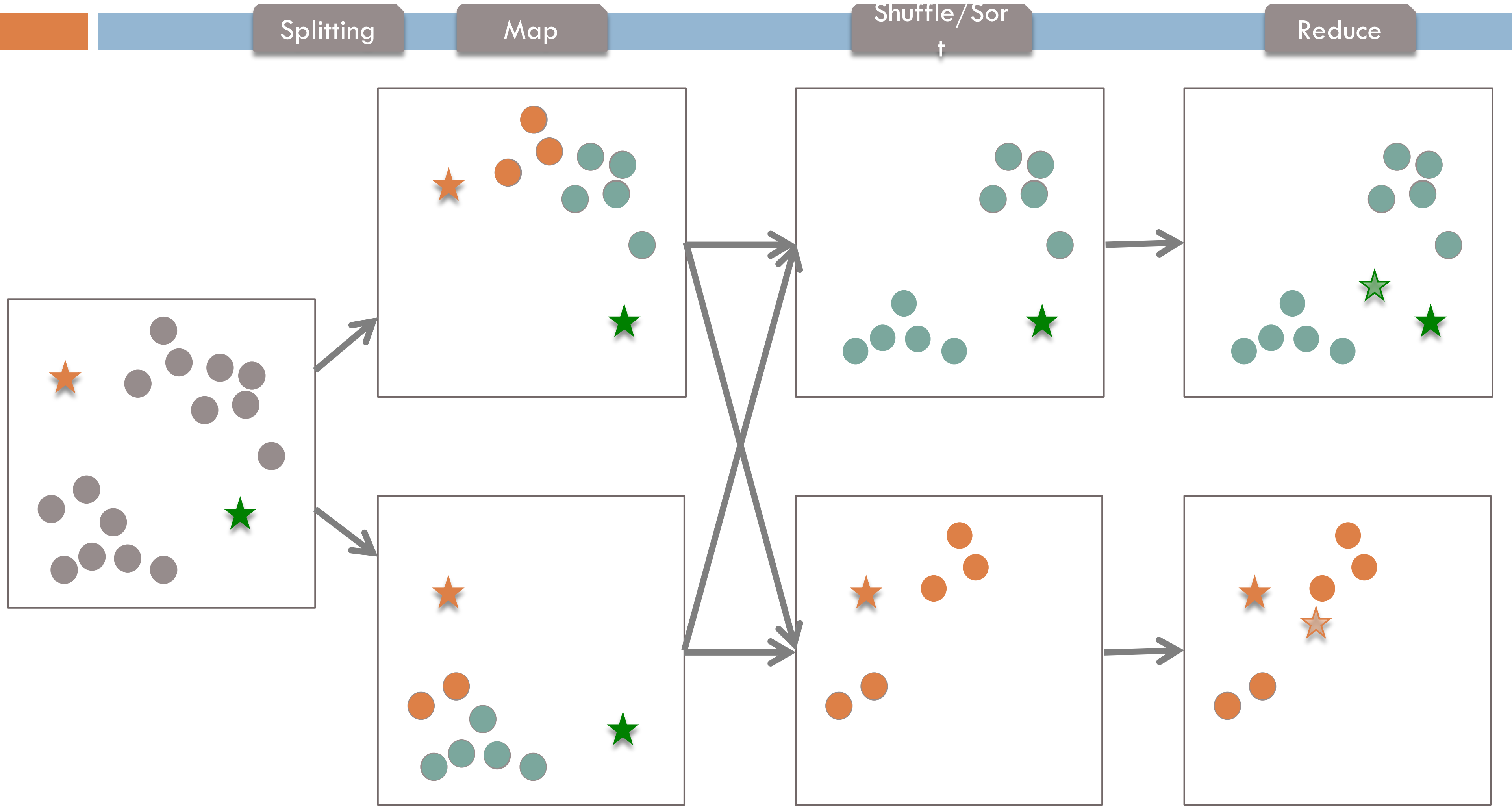
Meets convergence criteria



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



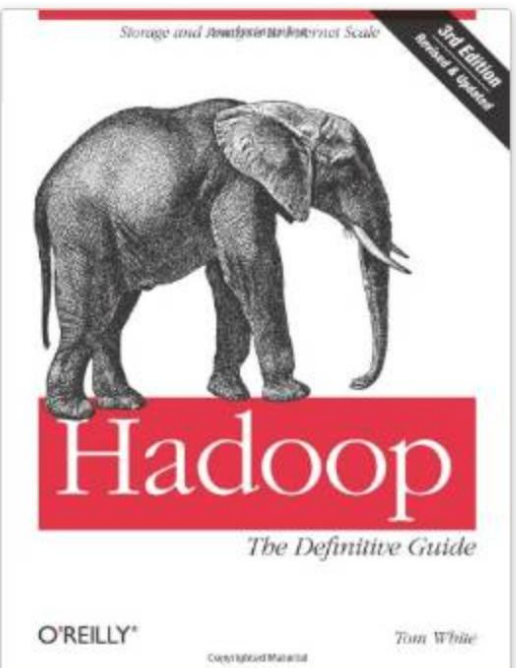
K-Means M/R One Iteration

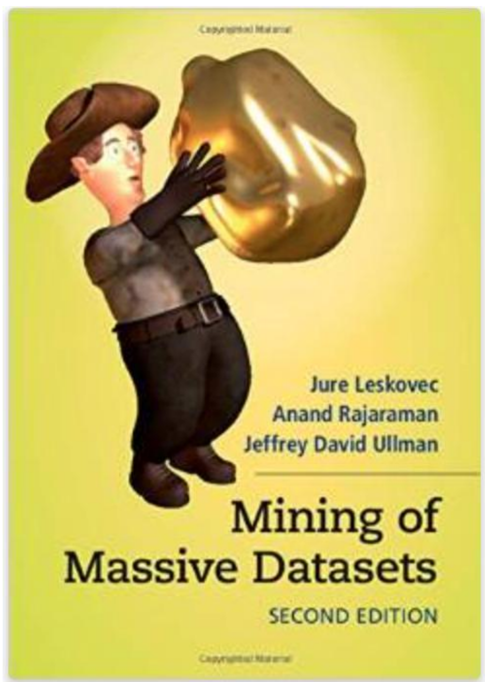
Recommended Readings - MapReduce

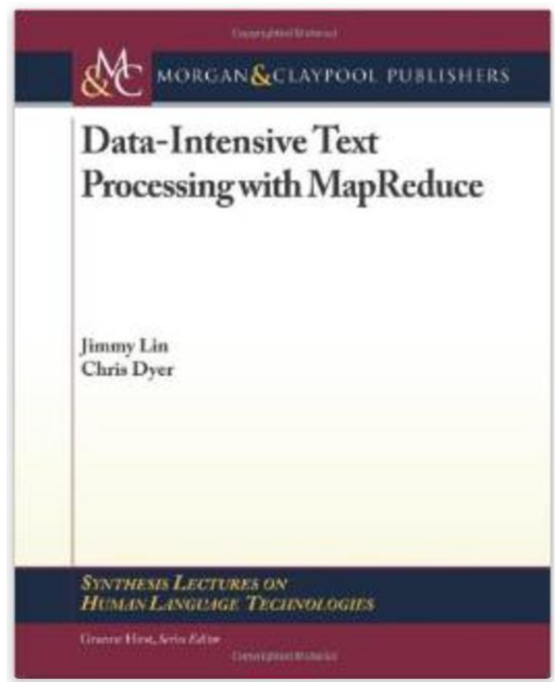
- (Google) Google MapReduce (link)
- (Google) The Google File System (link)
- (Microsoft) Scaling up vs scale out for hadoop: time to rethink? (link)
- (Google) Vision paper: towards an understanding of the limits of mapreduce computation (link)
- (Twitter) MapReduce is good enough? If all you have is a hammer, throw away everything that is not a nail! (link)

Recommended Readings – MapReduce/Hadoop

- Books
  - Hadoop: The Definitive Guide – Chap 2
  - Mining of Massive Datasets – Chap 2.1,2.2
  - Data-Intensive Text Processing with MapReduce – Chap 2







Recommended Readings – Data Analytics

- The Data Analytics Handbook
  - https://www.teamleada.com/handbook#download

Recommended Reading – Linux Commands

- Data Science at the Command Line
  - http://datascienceatthecommandline.com