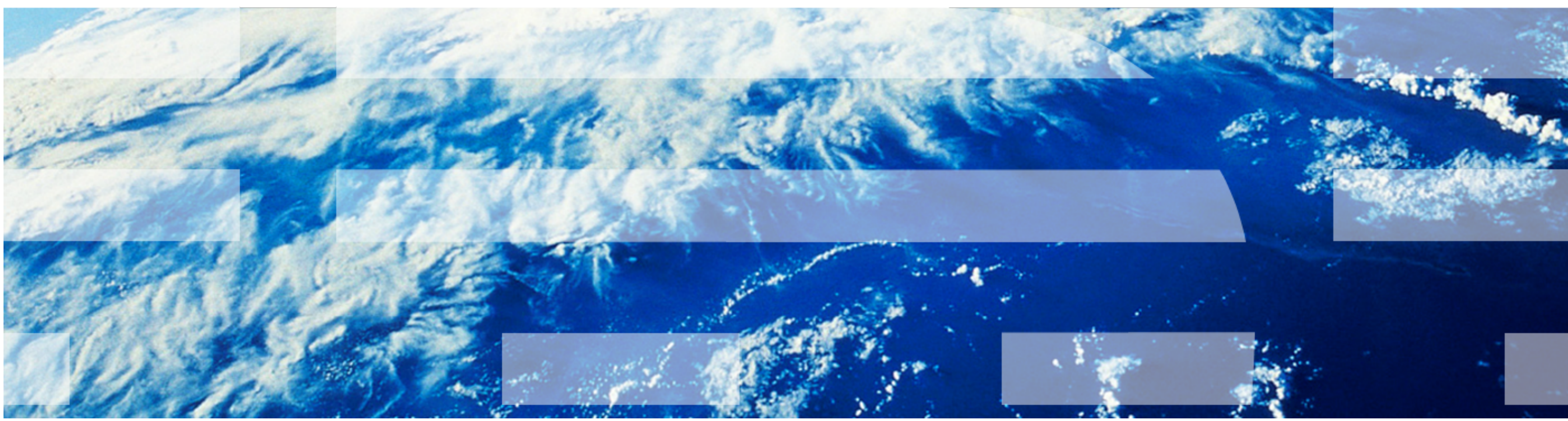
E6893 Big Data Analytics Lecture 2:

Big Data Analytics Platforms

Ching-Yung Lin, Ph.D.

Adjunct Professor, Depts. of Electrical Engineering and Computer Science



September 13th, 2024

Remind -- Apache Hadoop

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

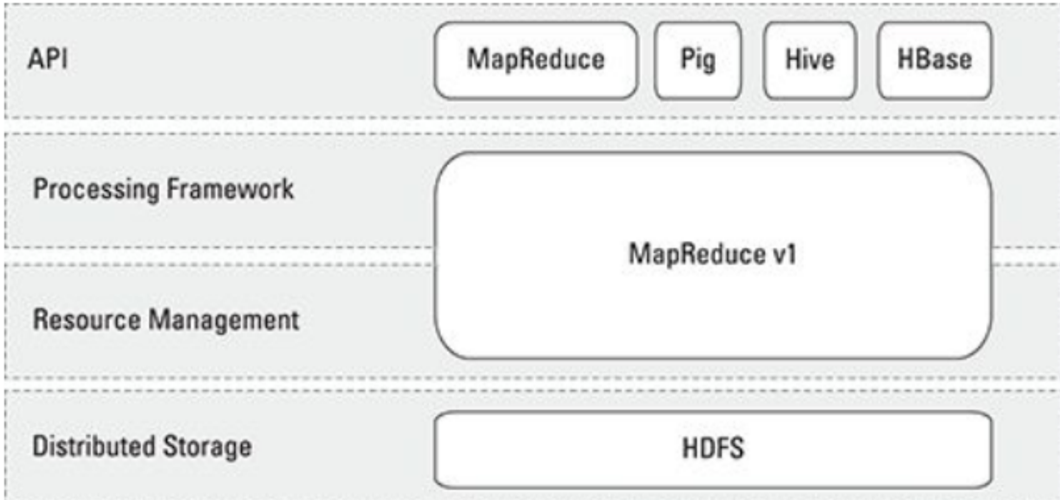
- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

http://hadoop.apache.org

Remind -- Hadoop-related Apache Projects

- Ambari™: A web-based tool for provisioning, managing, and monitoring Hadoop clusters.It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig and Hive applications visually.
- Avro™: A data serialization system.
- Cassandra™: A scalable multi-master database with no single points of failure.
- Chukwa™: A data collection system for managing large distributed systems.
- HBase™: A scalable, distributed database that supports structured data storage for large tables.
- Hive™: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- Mahout™: A Scalable machine learning and data mining library.
- Pig™: A high-level data-flow language and execution framework for parallel computation.
- Spark™: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- Tez™: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- ZooKeeper™: A high-performance coordination service for distributed applications.

Four distinctive layers of Hadoop



Distributed storage: The Hadoop Distributed File System (HDFS) is the storage layer where the data, interim results, and final result sets are stored.

Resource management: In addition to disk space, all slave nodes in the Hadoop cluster have CPU cycles, RAM, and network bandwidth. A system such as Hadoop needs to be able to parcel out these resources so that multiple applications and users can share the cluster in predictable and tunable ways. This job is done by the JobTracker daemon.

Processing framework: The MapReduce process flow defines the execution of all applications in Hadoop 1. As we saw in Chapter 6, this begins with the map phase; continues with aggregation with shuffle, sort, or merge; and ends with the reduce phase. In Hadoop 1, this is also managed by the JobTracker daemon, with local execution being managed by TaskTracker daemons running on the slave nodes.

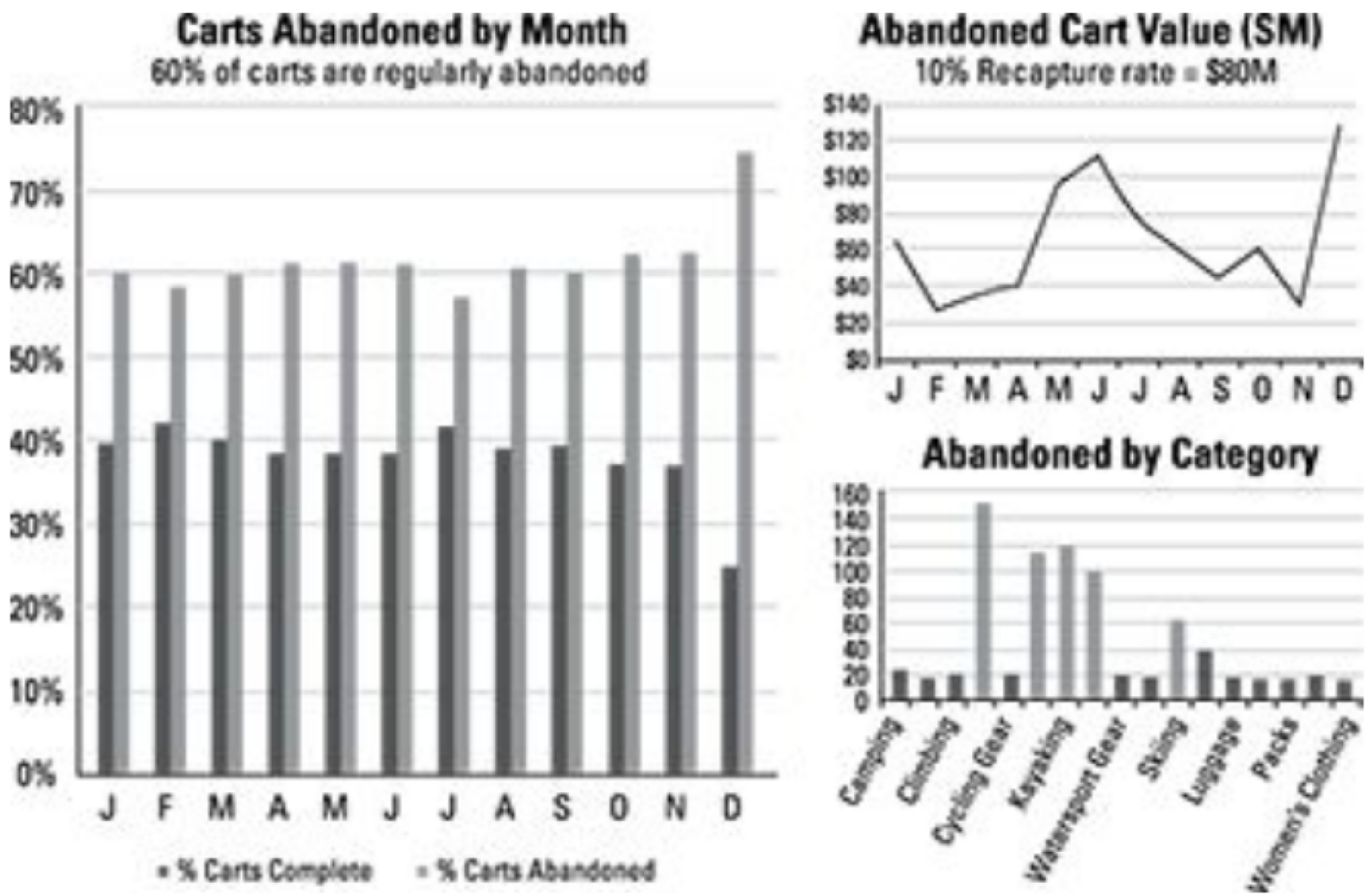
Application Programming Interface (API): Applications developed for Hadoop 1 needed to be coded using the MapReduce API. In Hadoop 1, the Hive and Pig projects provide programmers with easier interfaces for writing Hadoop applications, and underneath the hood, their code compiles down to MapReduce.

Common Use Cases for Big Data in Hadoop

- Log Data Analysis
  - most common, fits perfectly for HDFS scenario: **Write once & Read often.**
- Data Warehouse Modernization
- Fraud Detection
- Risk Modeling
- Social Sentiment Analysis
- Image Classification
- Graph Analysis
- Beyond

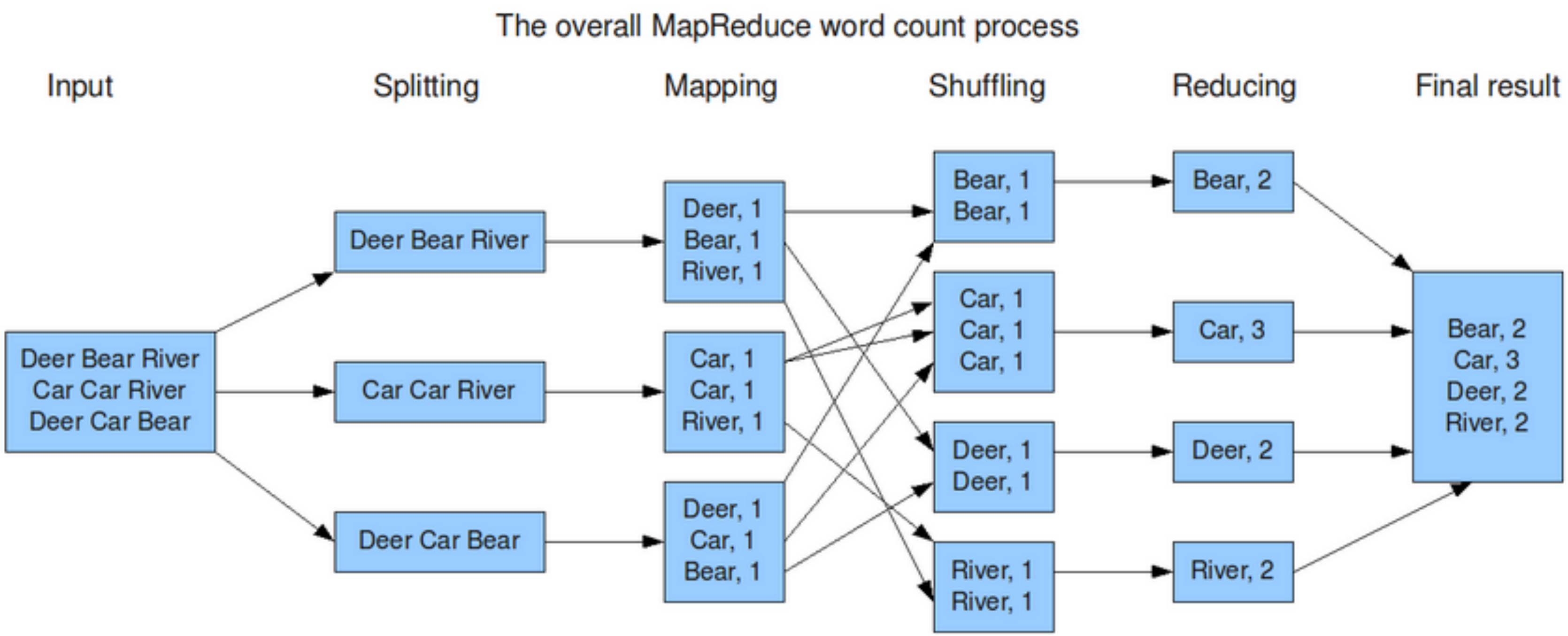
D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Example: Business Value of Log Analysis – "Struggle Detection"



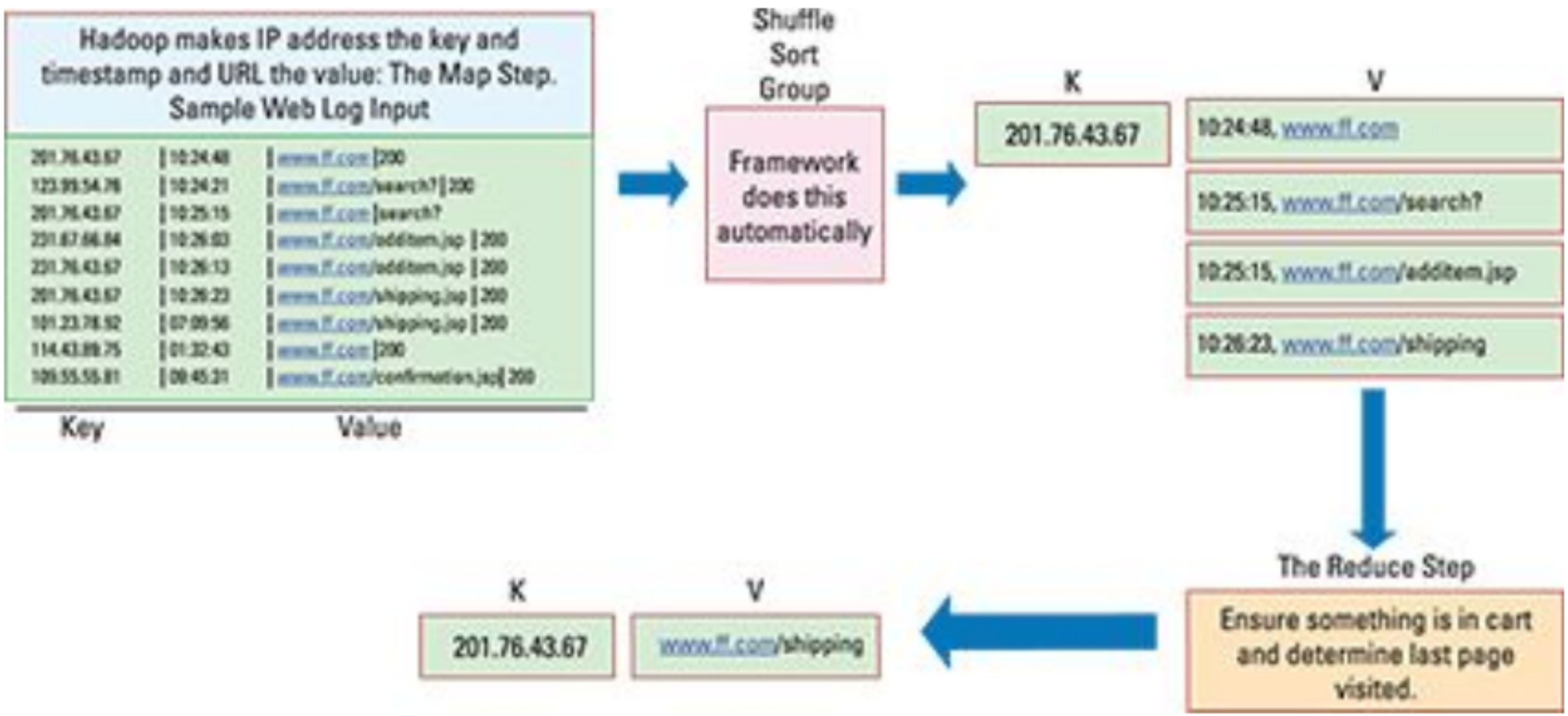
D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Remind -- MapReduce example



http://www.alex-hanna.com

MapReduce Process on User Behavior via Log Analysis



D. deRoos *et al*, Hadoop for Dummies, John Wiley & Sons, 2014

Setting Up the Hadoop Environment

- Local (standalone) mode
- Pseudo-distributed mode
- Fully-distributed mode

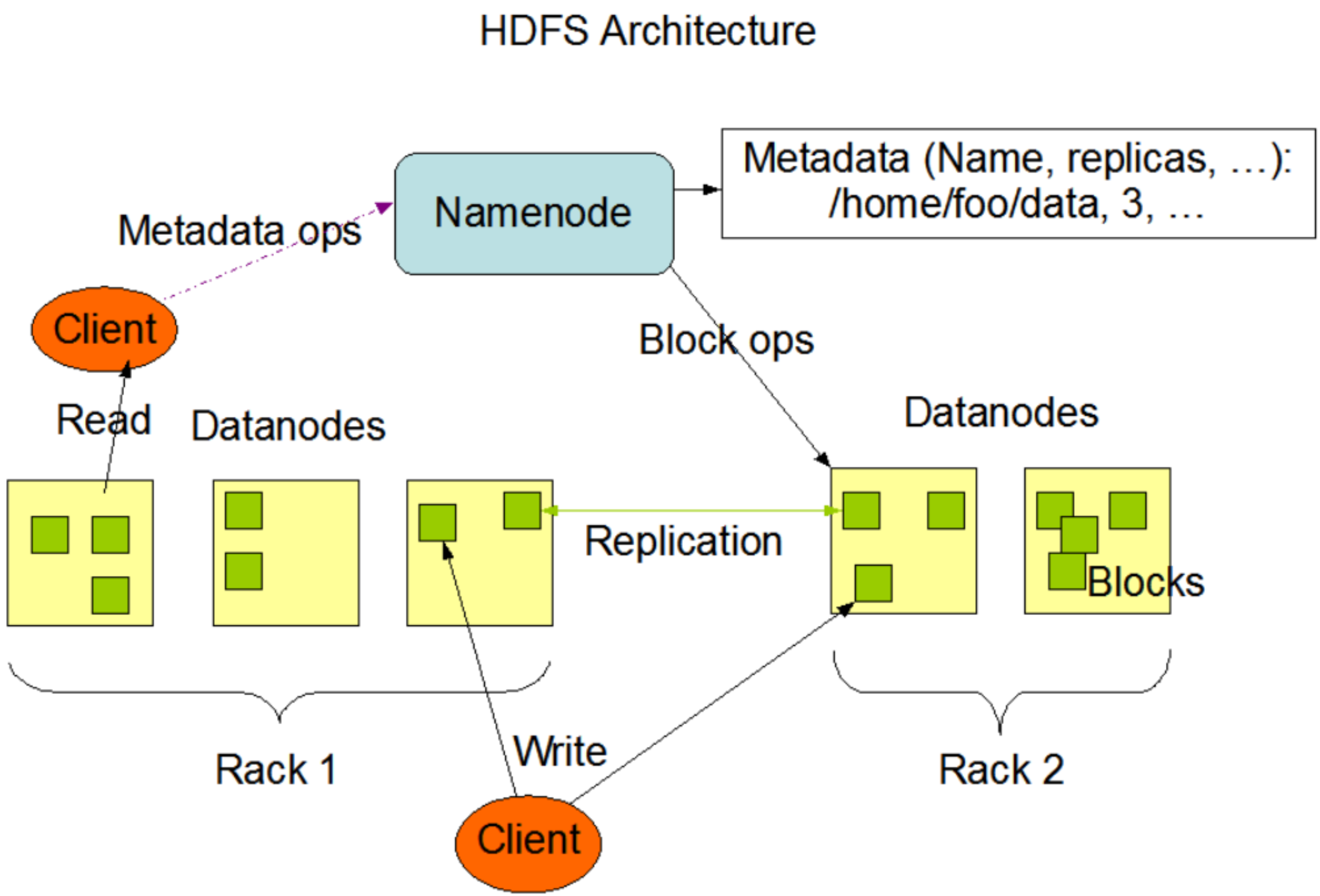
Data Storage Operations on HDFS

- Hadoop is designed to work best with a modest number of extremely large files.
- Average file sizes ➔ larger than 500MB.

- Write Once, Read Often model.
- Content of individual files cannot be modified, other than appending new data at the end of the file.

- What we can do:
  – Create a new file
  – Append content to the end of a file
  – Delete a file
  – Rename a file
  – Modify file attributes like owner

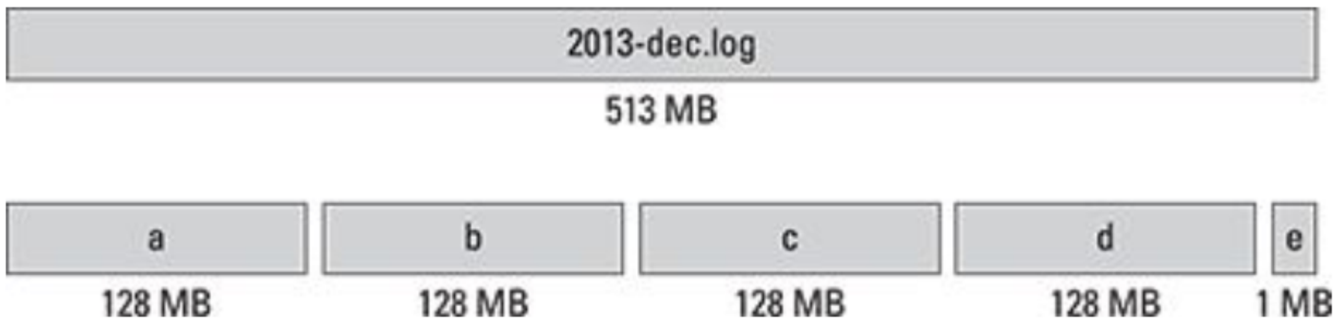
Remind -- Hadoop Distributed File System (HDFS)



http://hortonworks.com/hadoop/hdfs/

HDFS blocks

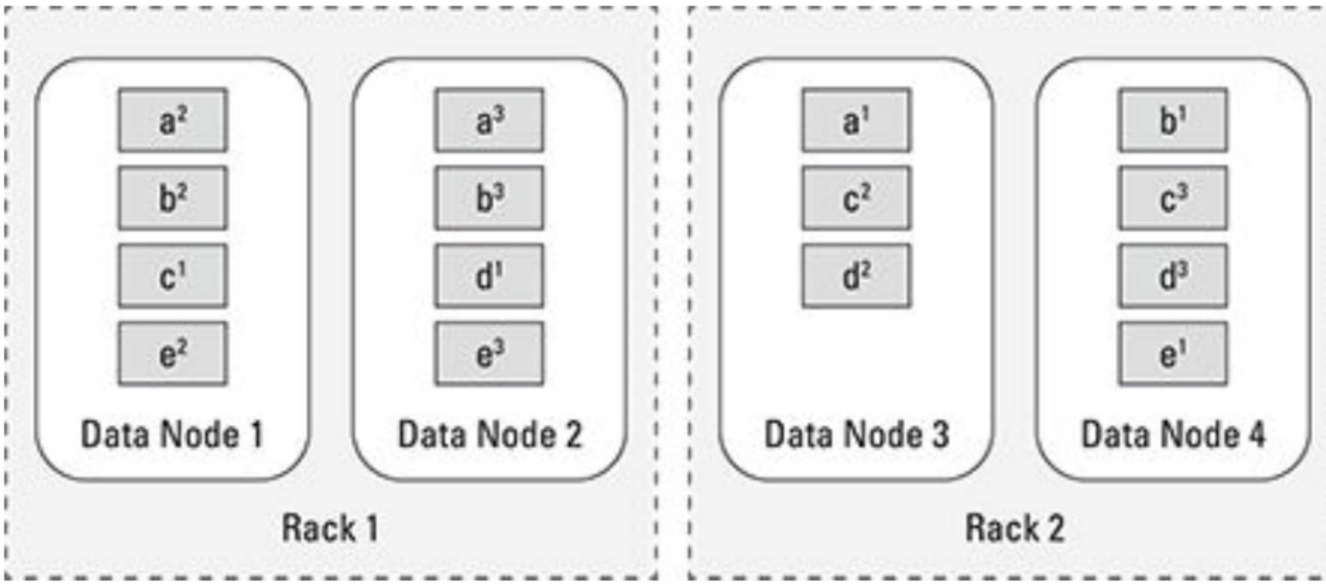
- File is divided into blocks (default: 64MB) and duplicated in multiple places (default: 3)



- Dividing into blocks is normal for a file system. E.g., the default block size in Linux is 4KB. The difference of HDFS is the scale.
- Hadoop was designed to operate at the petabyte scale.
- Every data block stored in HDFS has its own metadata and needs to be tracked by a central server.

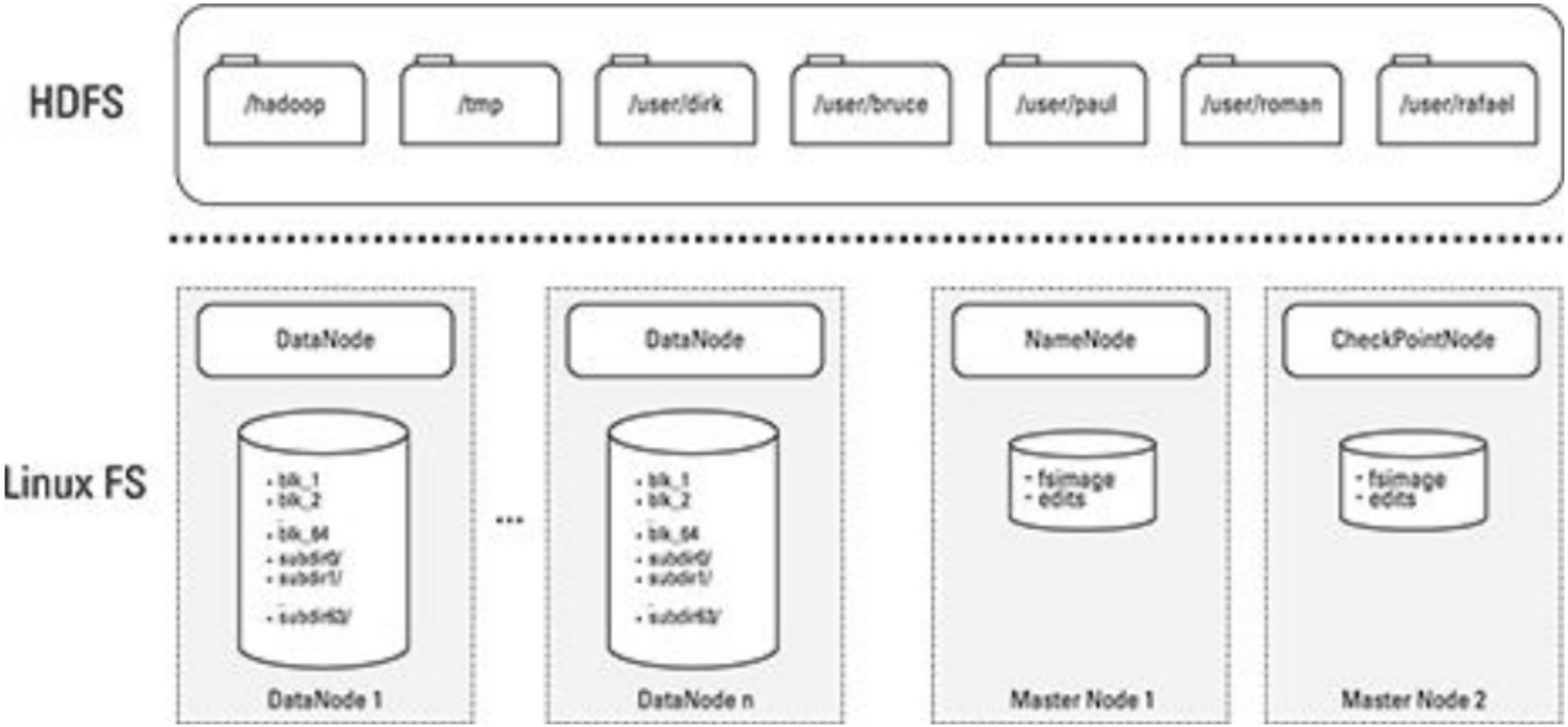
HDFS blocks

- Replication patterns of data blocks in HDFS.

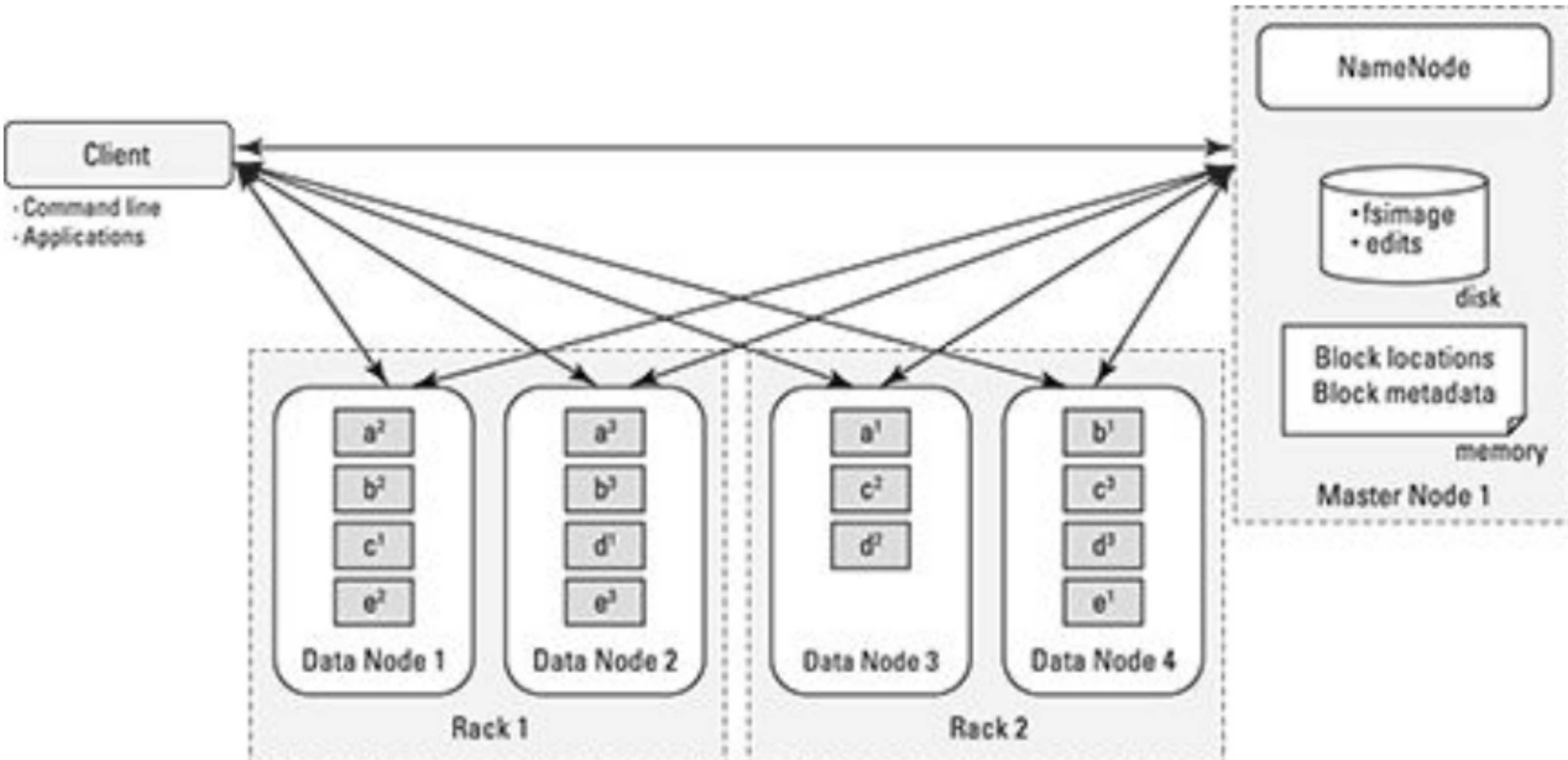


- When HDFS stores the replicas of the original blocks across the Hadoop cluster, it tries to ensure that the block replicas are stored in different failure points.

HDFS is a User-Space-Level file system

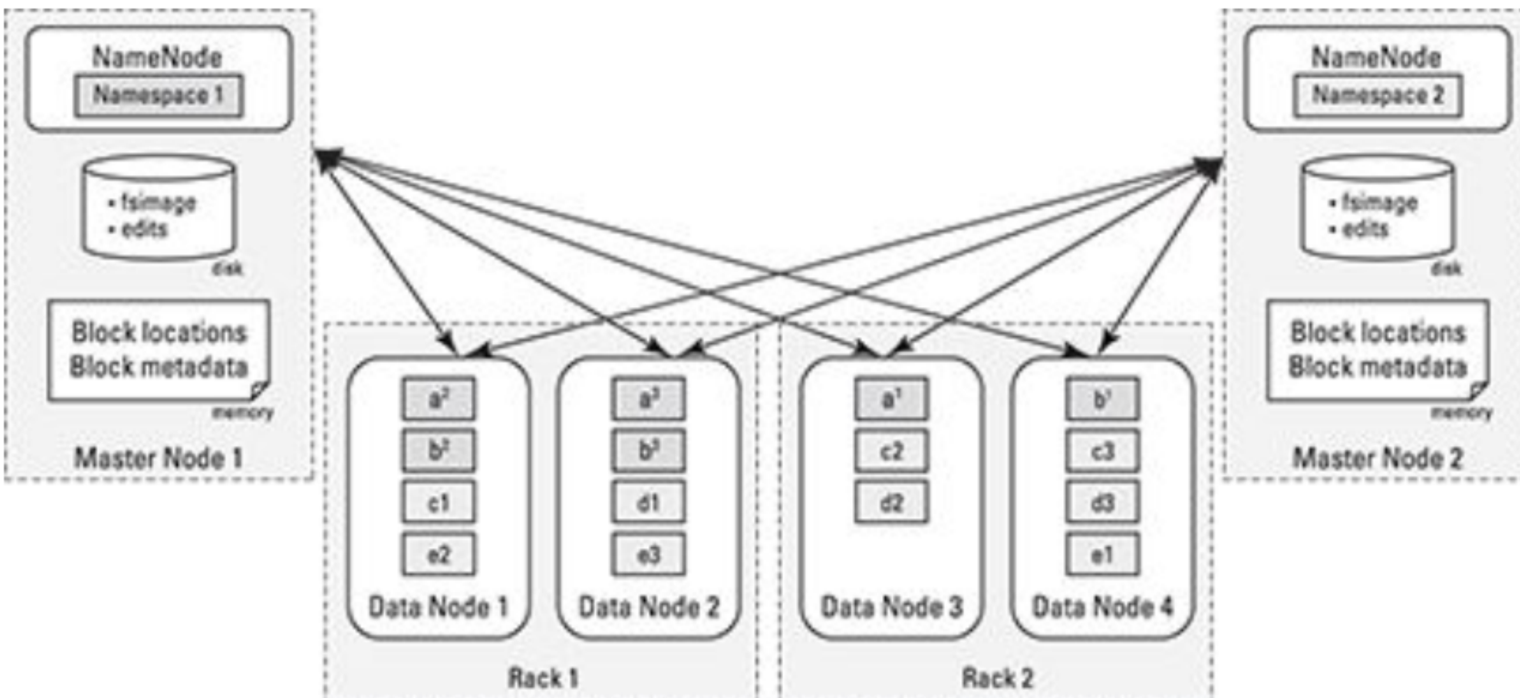


Interaction between HDFS components



HDFS Federation

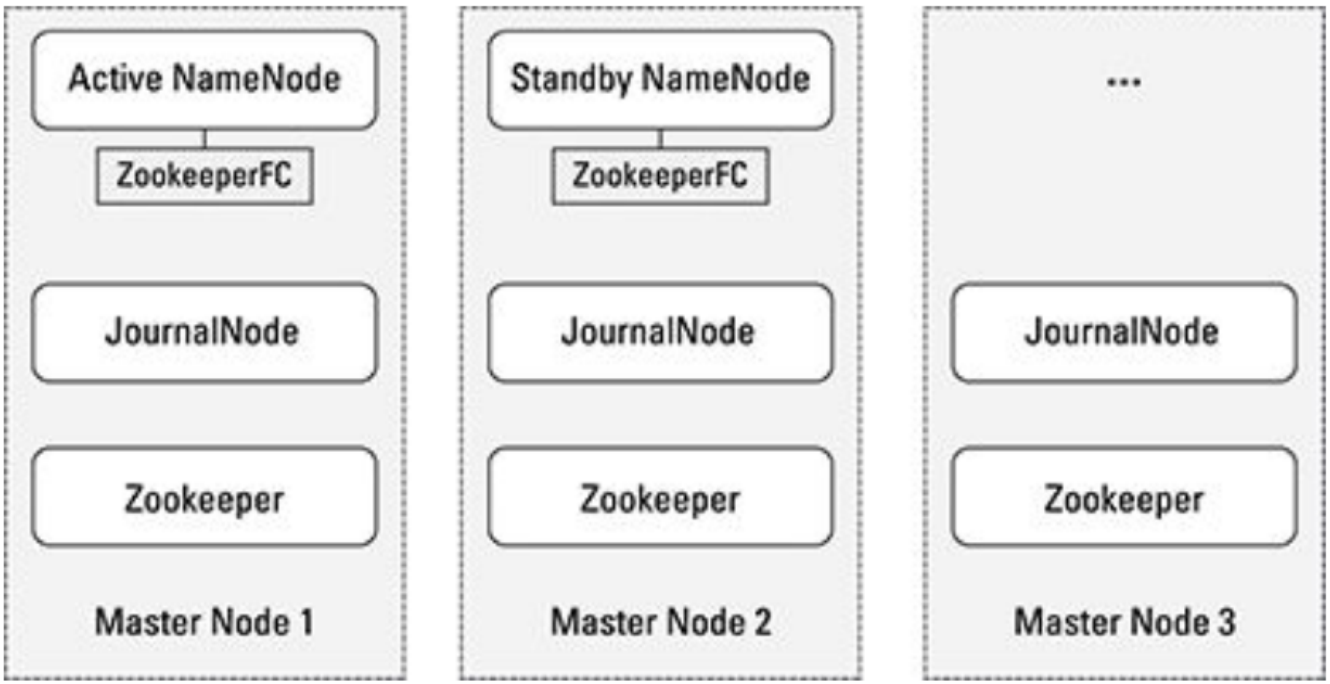
- Before Hadoop 2.0, NameNode was a single point of failure and operation limitation.
- Before Hadoop 2, Hadoop clusters usually have fewer clusters that were able to scale beyond 3,000 or 4,000 nodes.
- Multiple NameNodes can be used in Hadoop 2.x. (HDFS High Availability feature – one is in an Active state, the other one is in a Standby state).



http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html

High Availability of the NameNodes

- Active NameNode
- Standby NameNode – keeping the state of the block locations and block metadata in memory -> HDFS checkpointing responsibilities.



- JournalNode – if a failure occurs, the Standby Node reads all completed journal entries to ensure the new Active NameNode is fully consistent with the state of cluster.
- Zookeeper – provides coordination and configuration services for distributed systems.

Several useful commands for HDFS

- All hadoop commands are invoked by the bin/hadoop script.

  hadoop [--config confdir] [COMMAND] [GENERIC_OPTIONS] [COMMAND_OPTIONS]

- % hadoop fsck / -files –blocks:

  ➔ list the blocks that make up each file in HDFS.

- For HDFS, the schema name is hdfs, and for the local file system, the schema name is file.
- A file or director in HDFS can be specified in a fully qualified way, such as:
  hdfs://namenodehost/parent/child or hdfs://namenodehost

- The HDFS file system shell command is similar to Linux file commands, with the following general syntax: ***hadoop hdfs –file_cmd***

- For instance mkdir runs as:
  $hadoop hdfs dfs –mkdir /user/directory_name

Several useful commands for HDFS -- II

For example, to create a directory named "joanna", run this mkdir command:

$ hadoop hdfs dfs -mkdir /user/joanna

Use the Hadoop put command to copy a file from your local file system to HDFS:

$ hadoop hdfs dfs -put *file_name* /user/*login_user_name*

For example, to copy a file named data.txt to this new directory, run the following put command:

$ hadoop hdfs dfs -put data.txt /user/joanna

Run the ls command to get an HDFS file listing:

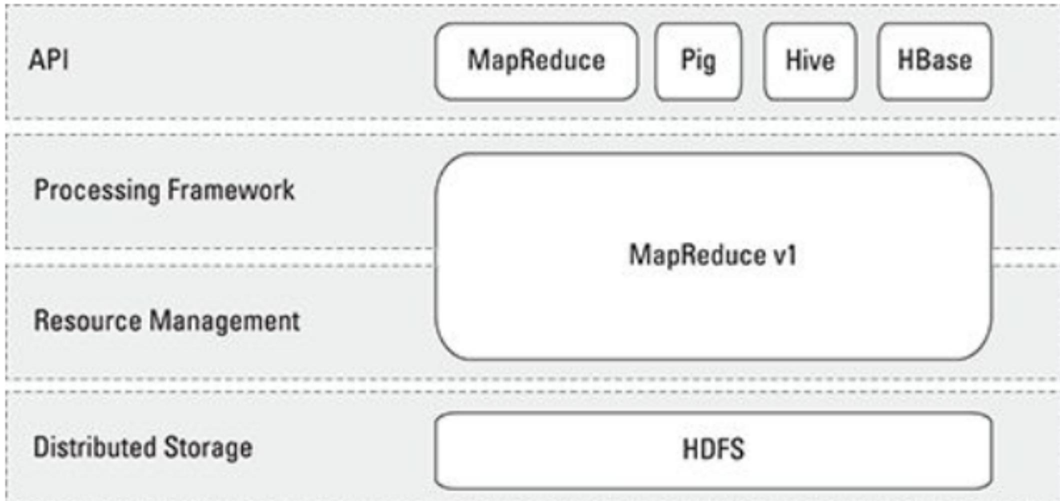
$ hadoop hdfs dfs -ls .

YARN

- YARN – Yet Another Resource Negotiator:
  - A Tool that enables the other processing frameworks to run on Hadoop.
  - A general-purpose resource management facility that can schedule and assign CPU cycles and memory (and in the future, other resources, such as network bandwidth) from the Hadoop cluster to waiting applications.

➔YARN has converted Hadoop from simply a batch processing engine into a platform for many different modes of data processing, from traditional batch to interactive queries to streaming analysis.

Four distinctive layers of Hadoop



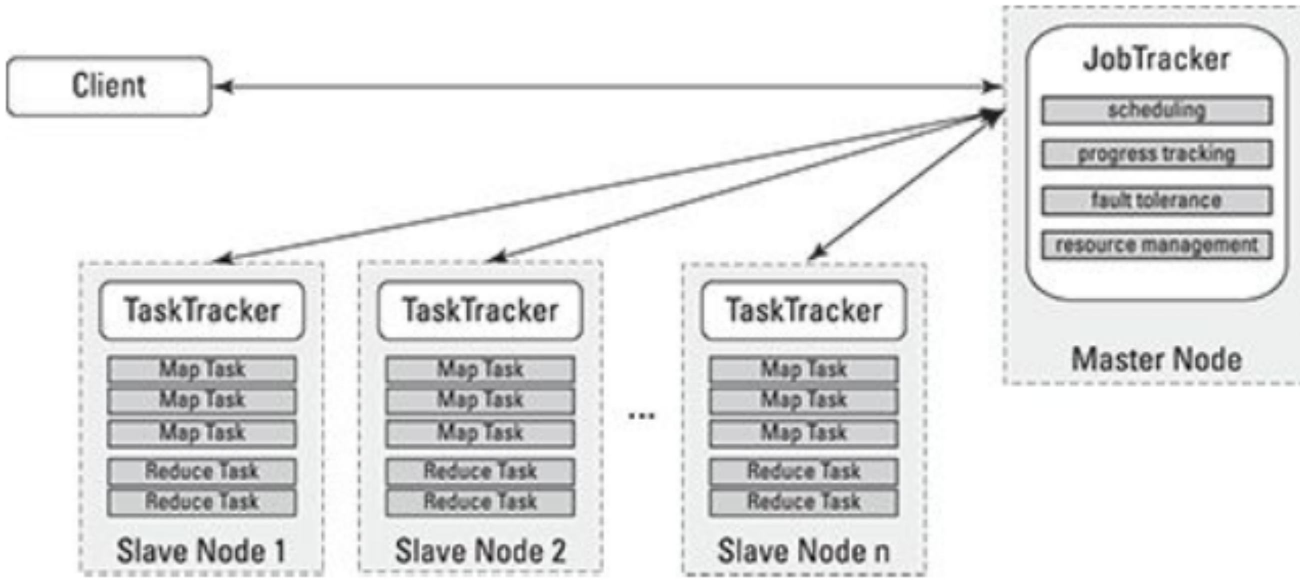
Distributed storage: The Hadoop Distributed File System (HDFS) is the storage layer where the data, interim results, and final result sets are stored.

Resource management: In addition to disk space, all slave nodes in the Hadoop cluster have CPU cycles, RAM, and network bandwidth. A system such as Hadoop needs to be able to parcel out these resources so that multiple applications and users can share the cluster in predictable and tunable ways. This job is done by the JobTracker daemon.

Processing framework: The MapReduce process flow defines the execution of all applications in Hadoop 1. As we saw in Chapter 6, this begins with the map phase; continues with aggregation with shuffle, sort, or merge; and ends with the reduce phase. In Hadoop 1, this is also managed by the JobTracker daemon, with local execution being managed by TaskTracker daemons running on the slave nodes.

Application Programming Interface (API): Applications developed for Hadoop 1 needed to be coded using the MapReduce API. In Hadoop 1, the Hive and Pig projects provide programmers with easier interfaces for writing Hadoop applications, and underneath the hood, their code compiles down to MapReduce.

Hadoop execution



1. The client application submits an application request to the JobTracker.
2. The JobTracker determines how many processing resources are needed to execute the entire application.
3. The JobTracker looks at the state of the slave nodes and queues all the map tasks and reduce tasks for execution.
4. As processing slots become available on the slave nodes, map tasks are deployed to the slave nodes. Map tasks are assigned to nodes where the same data is stored.
5. The JobTracker monitors task progress. If failure, the task is restarted on the next available slot.
6. After the map tasks are finished, reduce tasks process the interim results sets from the map tasks.
7. The result set is returned to the client application.

Limitation of original Hadoop 1

- MapReduce is a successful batch-oriented programming model.
- A glass ceiling in terms of wider use:
  - Exclusive tie to MapReduce, which means it could be used only for batch-style workloads and for general-purpose analysis.
- Triggered demands for additional processing modes:
  - Graph Analysis
  - Stream data processing
  - Message passing

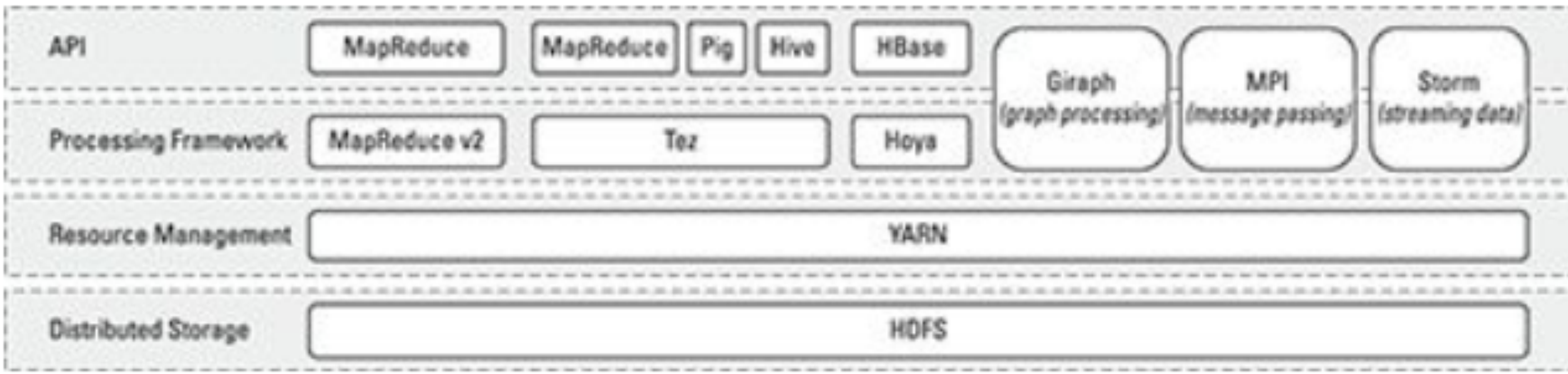
➔ Demand is growing for real-time and ad-hoc analysis

➔ Analysts ask many smaller questions against subsets of data and need a near-instant response.

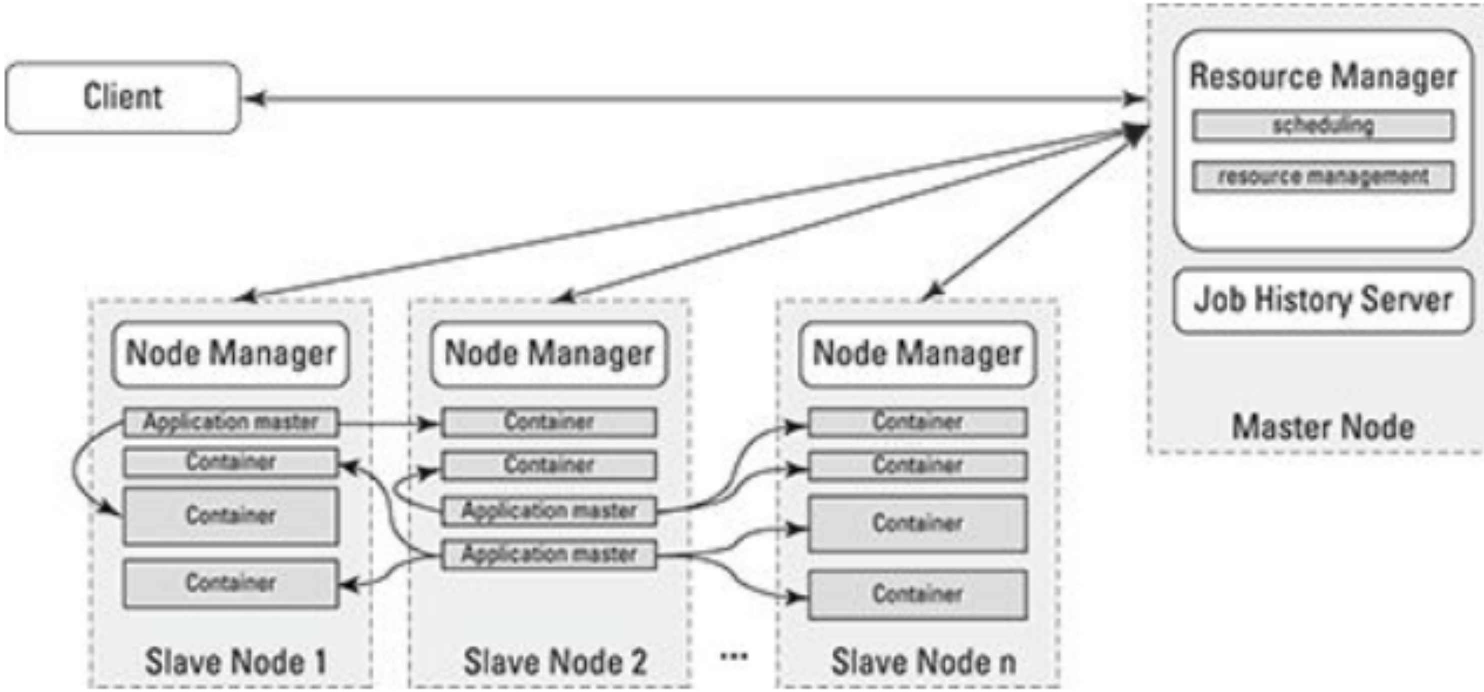
➔ Some analysts are more used to SQL & Relational databases

YARN was created to move beyond the limitation of a Hadoop 1 / MapReduce world.

Hadoop Data Processing Architecture

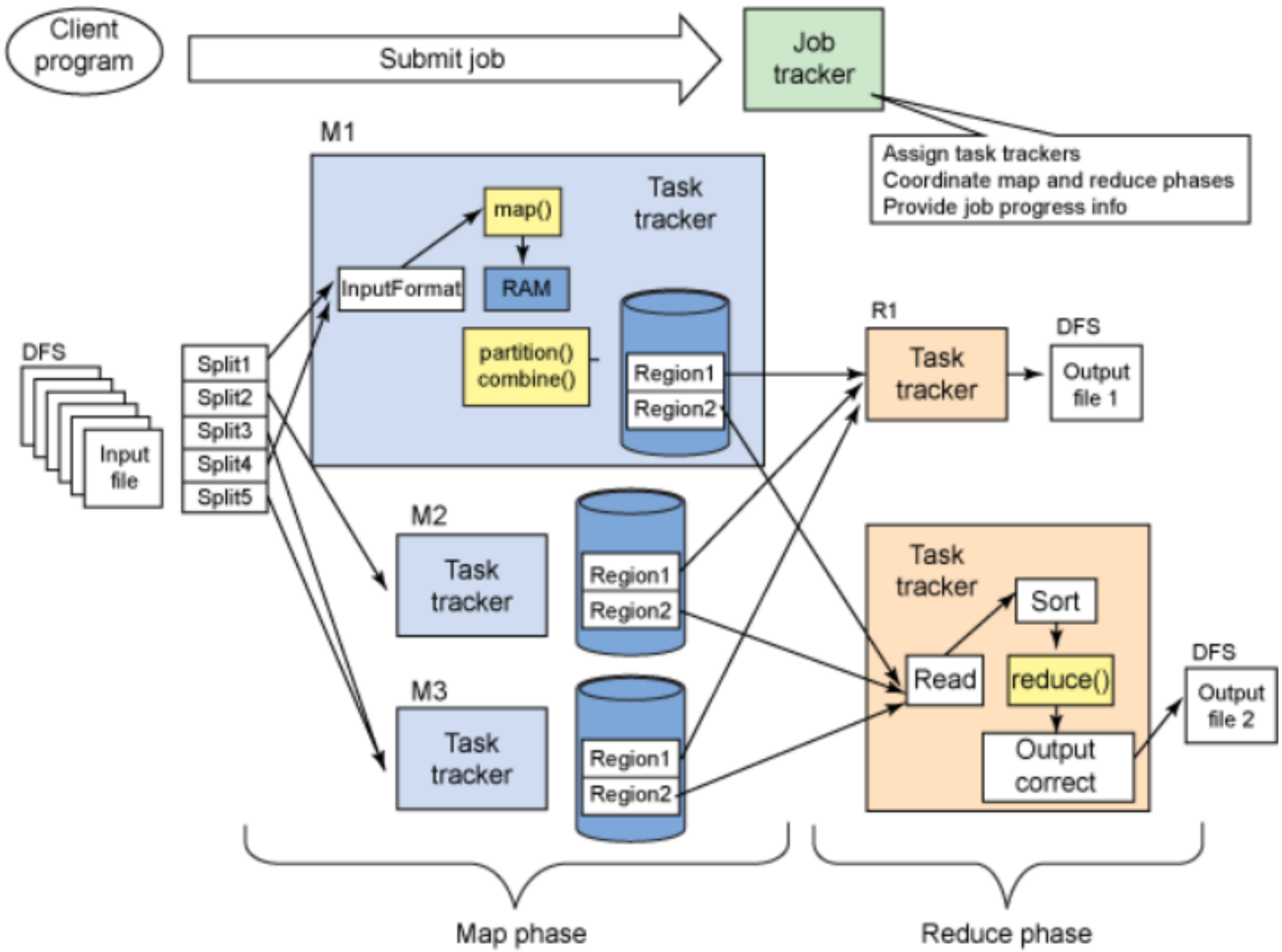


YARN's application execution



- Client submits an application to Resource Manager.
- Resource Manager asks a Node Manager to create an Application Master instance and starts up.
- Application Manager initializes itself and register with the Resource Manager
- Application manager figures out how many resources are needed to execute the application.
- The Application Master then requests the necessary resources from the Resource Manager. It sens heartbeat message to the Resource Manager throughout its lifetime.
- The Resource Manager accepts the request and queue up.
- As the requested resources become available on the slave nodes, the Resource Manager grants the Application Master leases for containers on specific slave nodes.
- …. ➔ only need to decide on how much memory tasks can have.

Remind -- MapReduce Data Flow



http://www.ibm.com/developerworks/cloud/library/cl-openstack-deployhadoop/

MapReduce Use Case Example – flight data

- Data Source: Airline On-time Performance data set (flight data set).
  - All the logs of domestic flights from the period of October 1987 to April 2008.
  - Each record represents an individual flight where various details are captured:
    - Time and date of arrival and departure
    - Originating and destination airports
    - Amount of time taken to taxi from the runway to the gate.
  - Download it from Statistical Computing: http://stat-computing.org/dataexpo/2009/

Bi-Annual Data Exposition

Every other year, at the Joint Statistical Meetings, the Graphics Section and the Computing Section join in sponsoring a special Poster Session called **The Data Exposition** , but more commonly known as **The Data Expo**. All of the papers presented in this Poster Session are reports of analyses of a common data set provided for the occasion. In addition, all papers presented in the session are encouraged to report the use of graphical methods employed during the development of their analysis and to use graphics to convey their findings.

Data sets

- 2013: Soul of the Community
- 2011: Deepwater horizon oil spill
- 2009: Airline on time data
- 2006: NASA meteorological data. Electronic copy of entries
- 1997: Hospital Report Cards
- 1995: U.S. Colleges and Universities
- 1993: Oscillator time series & Breakfast Cereals
- 1991: Disease Data for Public Health Surveillance
- 1990: King Crab Data
- 1988: Baseball
- 1986: Geometric Features of Pollen Grains
- 1983: Automobiles

http://stat-computing.org/dataexpo/

Flight Data Schema

	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	unique carrier code
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes

17	Origin	origin IATA airport code
18	Dest	destination IATA airport code
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

MapReduce Use Case Example – flight data

- Count the number of flights for each carrier

- Serial way (not MapReduce):

Listing 6-1: Pseudocode for Calculating The Number of Flights By Carrier Serially

```
create a two-dimensional array
create a row for every airline carrier
  populate the first column with the carrier code
  populate the second column with the integer zero

for each line of flight data
  read the airline carrier code
  find the row in the array that matches the carrier code
    increment the counter in the second column by one

print the totals for each row in the two-dimensional array
```

MapReduce Use Case Example – flight data

- Count the number of flights for each carrier

- Parallel way:

Listing 6-2: Pesudocode for Calculating The Number of Flights By Carrier in Parallel

```
 Map Phase:
 for each line of flight data
  read the current record and extract the airline carrier code
  output the airline carrier code and the number one as a key/value pair

Shuffle and Sort Phase:
 read the list of key/value pairs from the map phase
 group all the values for each key together
  each key has a corresponding array of values
 sort the data by key
 output each key and its array of values

Reduce Phase:
 read the list of carriers and arrays of values from the shuffle and sort phase
 for each carrier code
  add the total number of ones in the carrier code's array of values together

print the totals for each row in the two-dimensional array
```

MapReduce application flow

Determine the exact data sets to process from the data blocks. This involves calculating where the records to be processed are located within the data blocks.

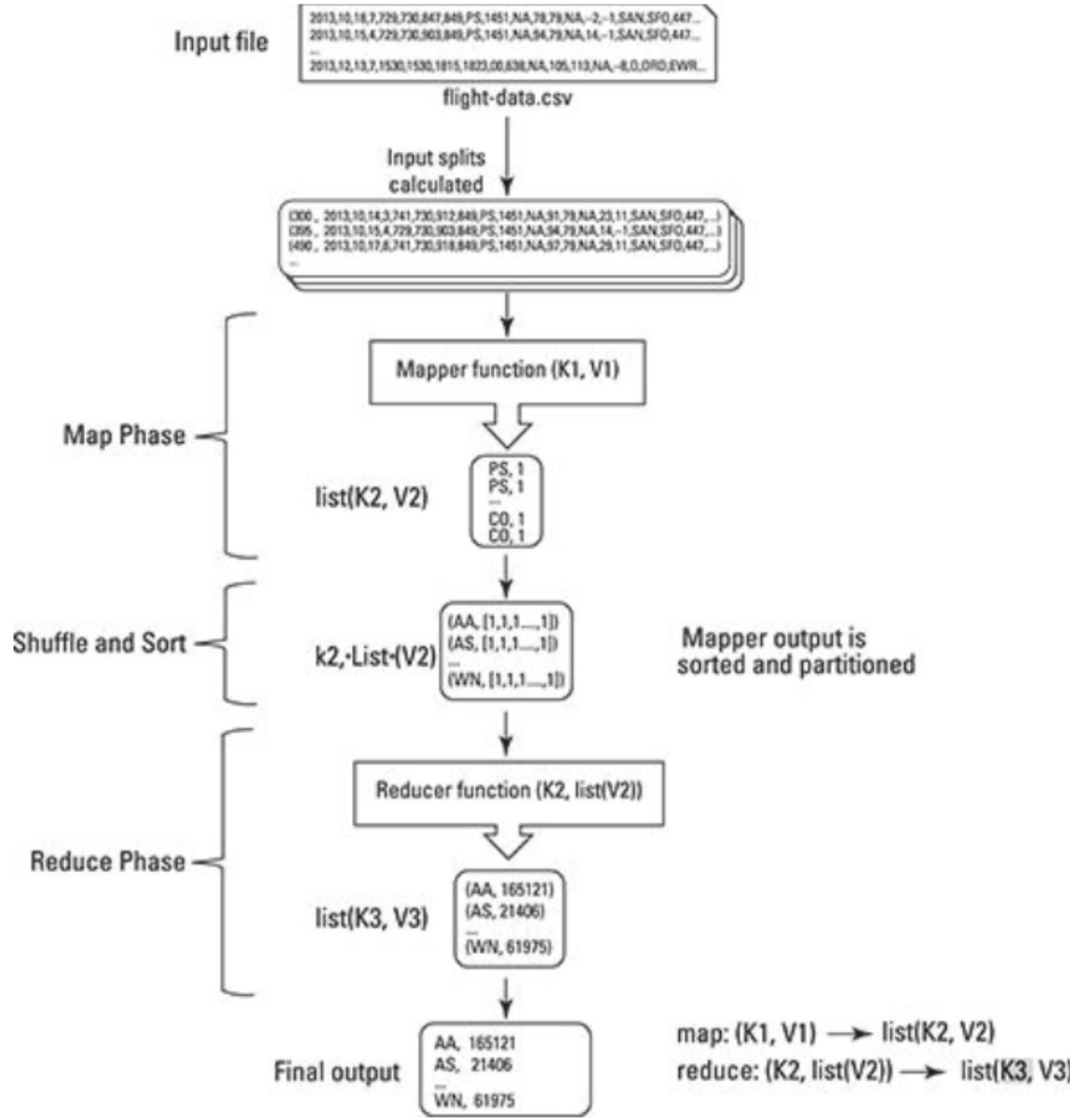
Run the specified algorithm against each record in the data set until all the records are processed. The individual instance of the application running against a block of data in a data set is known as a *mapper task*. (This is the mapping part of MapReduce.)

Locally perform an interim reduction of the output of each mapper. (The outputs are provisionally combined, in other words.) This phase is optional because, in some common cases, it isn't desirable.

Based on partitioning requirements, group the applicable partitions of data from each mapper's result sets.

Boil down the result sets from the mappers into a single result set — the Reduce part of MapReduce. An individual instance of the application running against mapper output data is known as a *reducer task*.

MapReduce steps for flight data computation



HBase is modeled after Google's BigTable and written in Java. It is developed on top of HDFS.

It provides a fault-tolerant way of storing large quantities of **sparse data** (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

HBase features compression, in-memory operation, and Bloom filters on a per-column basis

An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. An HBase column represents an attribute of an object

APACHE HBASE

Characteristics of data in HBase

Sparse data

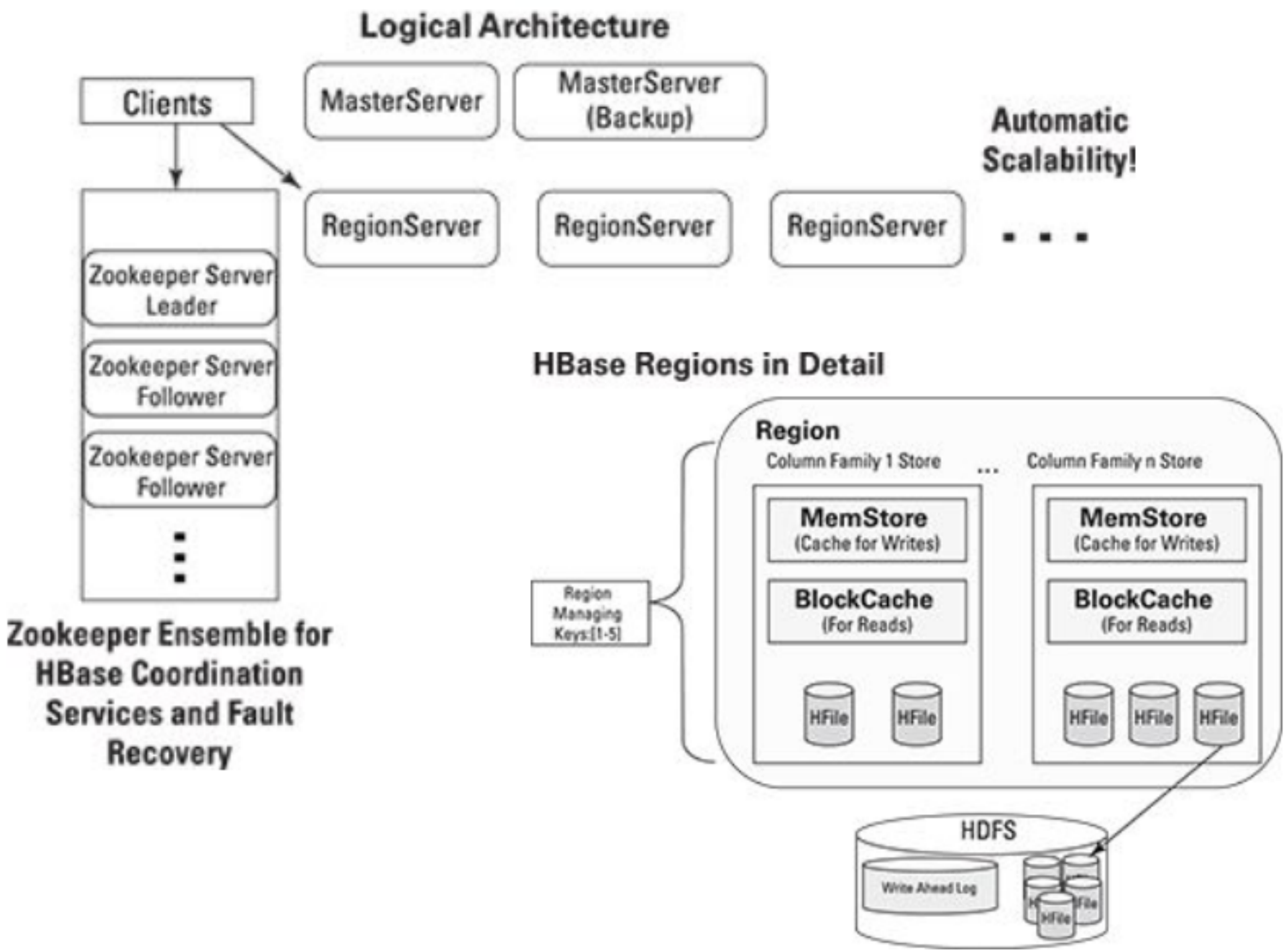
Table 12-1 Traditional Customer Contact Information Table

Customer ID	*Last Name*	*First Name*	*Middle Name*	*E-mail Address*	*Street Address*
00001	Smith	John	Timothy	John. Smith@ xyz.com	1 Hadoop Lane, NY 11111
00002	Doe	Jane	NULL	NULL	7 HBase Ave, CA 22222

Row Key	***Column Family: {Column Qualifier:Version:Value}***
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'}
	ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
00002	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe',
	ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

HDFS lacks **random read and write access**. This is where HBase comes into picture. It's a **distributed, scalable, big data store**, modeled after Google's BigTable. It stores data as key/value pairs.

HBase Architecture



Creating a table

```
hbase(main):002:0> create 'CustomerContactInfo', 'CustomerName', 'ContactInfo'
0 row(s) in 1.2080 seconds
```

Entering Records

```
hbase(main):008:0> put 'CustomerContactInfo', '00001', 'CustomerName:FN', 'John'
0 row(s) in 0.2870 seconds

hbase(main):009:0> put 'CustomerContactInfo', '00001', 'CustomerName:LN', 'Smith'
0 row(s) in 0.0170 seconds

hbase(main):010:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'T'
0 row(s) in 0.0070 seconds

hbase(main):011:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'Timothy'
0 row(s) in 0.0050 seconds

hbase(main):012:0> put 'CustomerContactInfo', '00001', 'ContactInfo:EA', 'John.Smith@xyz.com'
0 row(s) in 0.0170 seconds

hbase(main):013:0> put 'CustomerContactInfo', '00001', 'ContactInfo:SA', '1 Hadoop Lane, NY 11111'
0 row(s) in 0.0030 seconds

hbase(main):014:0> put 'CustomerContactInfo', '00002', 'CustomerName:FN', 'Jane'
0 row(s) in 0.0290 seconds

hbase(main):015:0> put 'CustomerContactInfo', '00002', 'CustomerName:LN', 'Doe'
0 row(s) in 0.0090 seconds

hbase(main):016:0> put 'CustomerContactInfo', '00002', 'ContactInfo:SA', '7 HBase Ave, CA 22222'
0 row(s) in 0.0240 seconds
```

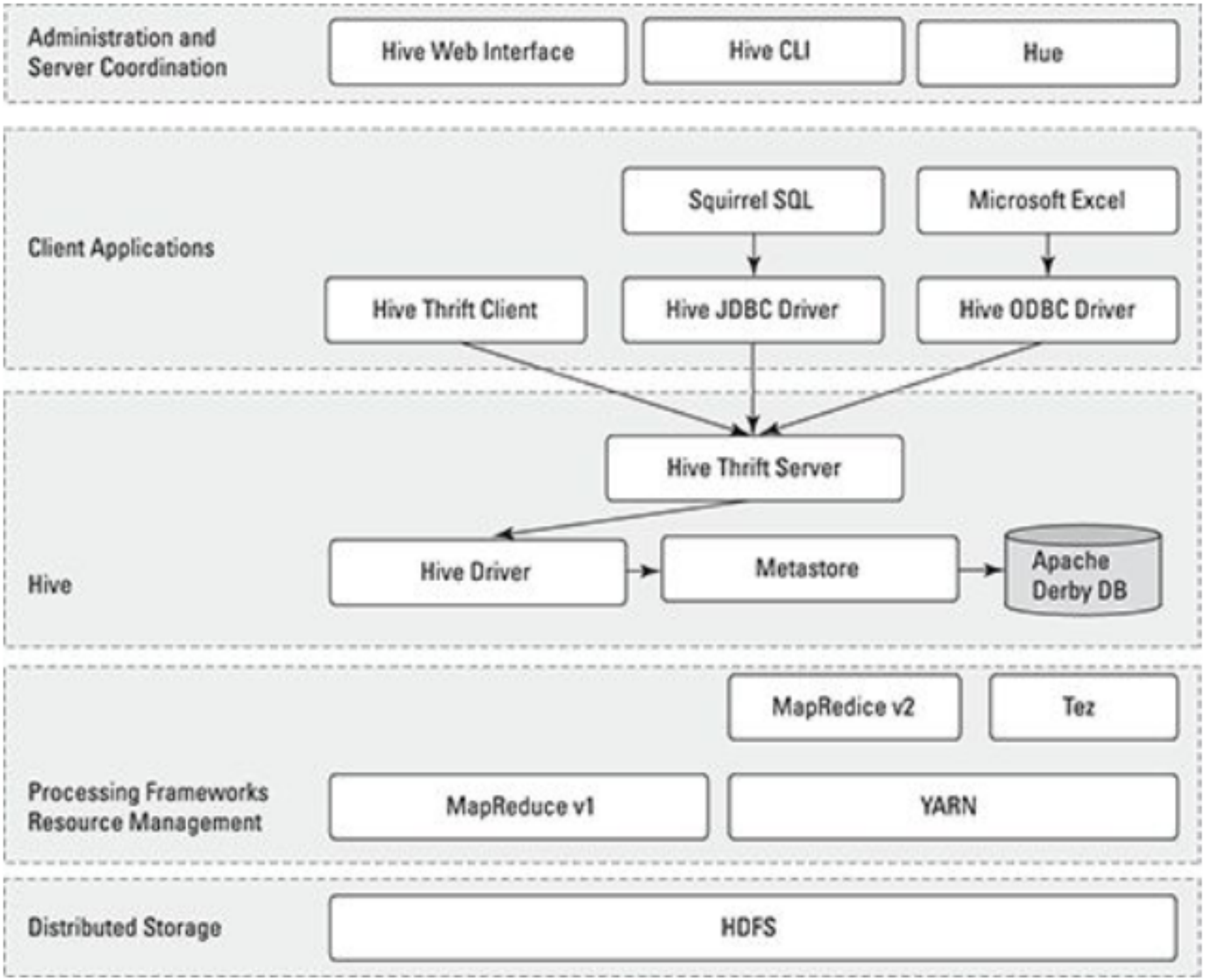
Scan Results

```
hbase(main):020:0> scan 'CustomerContactInfo', {VERSIONS => 2}
ROW        COLUMN+CELL
 00001     column=ContactInfo:EA, timestamp=1383859183030, value=John.Smith@xyz.com
 00001     column=ContactInfo:SA, timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
 00001     column=CustomerName:FN, timestamp=1383859182496, value=John
 00001     column=CustomerName:LN, timestamp=1383859182858, value=Smith
 00001     column=CustomerName:MN, timestamp=1383859183001, value=Timothy
 00001     column=CustomerName:MN, timestamp=1383859182915, value=T
 00002     column=ContactInfo:SA, timestamp=1383859185577, value=7 HBase Ave, CA 22222
 00002     column=CustomerName:FN, timestamp=1383859183103, value=Jane
 00002     column=CustomerName:LN, timestamp=1383859183163, value=Doe
2 row(s) in 0.0520 seconds
```

Using the *get* Command to Retrieve Entire Rows and Individual Values

```
(1) hbase(main):037:0> get 'CustomerContactInfo', '00001'
COLUMN              CELL
ContactInfo:EA      timestamp=1383859183030, value=John.Smith@xyz.com
ContactInfo:SA      timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
CustomerName:FN     timestamp=1383859182496, value=John
CustomerName:LN     timestamp=1383859182858, value=Smith
CustomerName:MN     timestamp=1383859183001, value=Timothy
5 row(s) in 0.0150 seconds

(2) hbase(main):038:0> get 'CustomerContactInfo', '00001',
       {COLUMN => 'CustomerName:MN'}
COLUMN              CELL
CustomerName:MN     timestamp=1383859183001, value=Timothy
1 row(s) in 0.0090 seconds

(3) hbase(main):039:0> get 'CustomerContactInfo', '00001',
       {COLUMN => 'CustomerName:MN',
       TIMESTAMP => 1383859182915}
COLUMN              CELL
CustomerName:MN     timestamp=1383859182915, value=T
1 row(s) in 0.0290 seconds
```

Apache Hive


Administration and Server Coordination
Hive Web Interface
Hive CLI
Hue
Client Applications
Squirrel SQL
Microsoft Excel
Hive Thrift Client
Hive JDBC Driver
Hive ODBC Driver
Hive
Hive Thrift Server
Hive Driver
Metastore
Apache Derby DB
Processing Frameworks Resource Management
MapRedice v2
Tez
MapReduce v1
YARN
Distributed Storage
HDFS

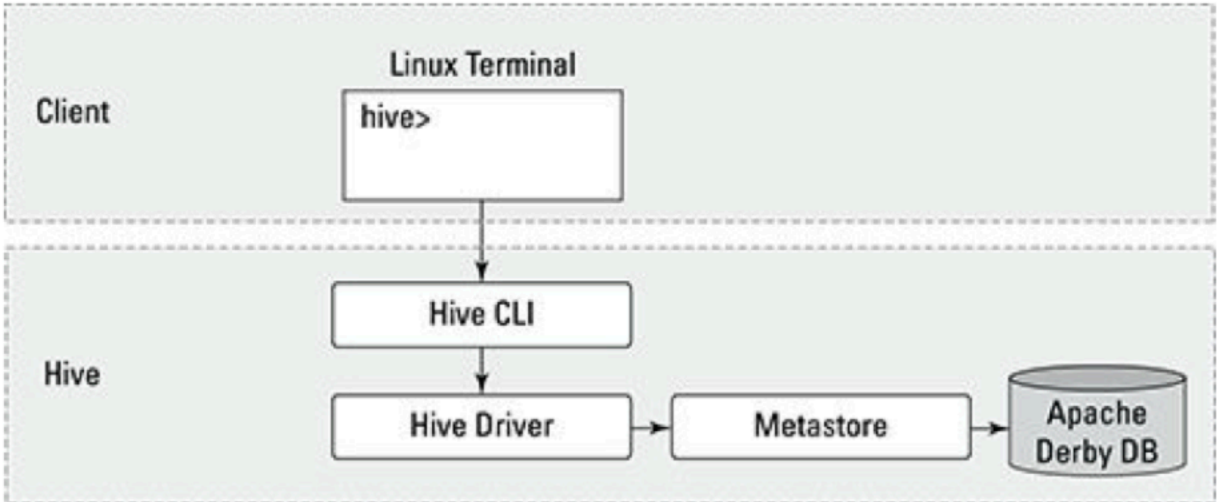
Creating, Dropping, and Alternating DB in Hive

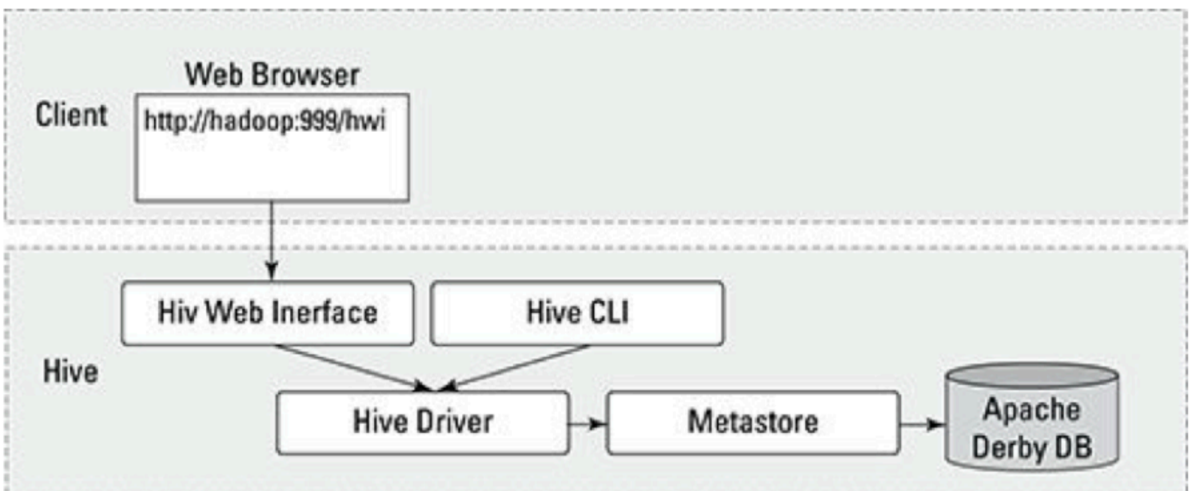
```
(1) $ $HIVE_HOME/bin hive --service cli
(2) hive> set hive.cli.print.current.db=true;
(3) hive (default)> USE ourfirstdatabase;
(4) hive (ourfirstdatabase)> ALTER DATABASE
ourfirstdatabase SET DBPROPERTIES
('creator'='Bruce Brown', 'created_for'='Learning Hive
DDL');
OK
Time taken: 0.138 seconds
(5) hive (ourfirstdatabase)> DESCRIBE DATABASE
EXTENDED ourfirstdatabase;
OK
ourfirstdatabase                     file:/home/biad
min/Hive/warehouse/ourfirstdatabase.db  {created_f
or=Learning Hive DDL, creator=Bruce Brown}
Time taken: 0.084 seconds, Fetched: 1 row(s)CREATE
(DATABASE|SCHEMA) [IF NOT EXISTS]
database_name
(6) hive (ourfirstdatabase)> DROP DATABASE
ourfirstdatabase CASCADE;
OK
Time taken: 0.132 seconds
```

Another Hive Example

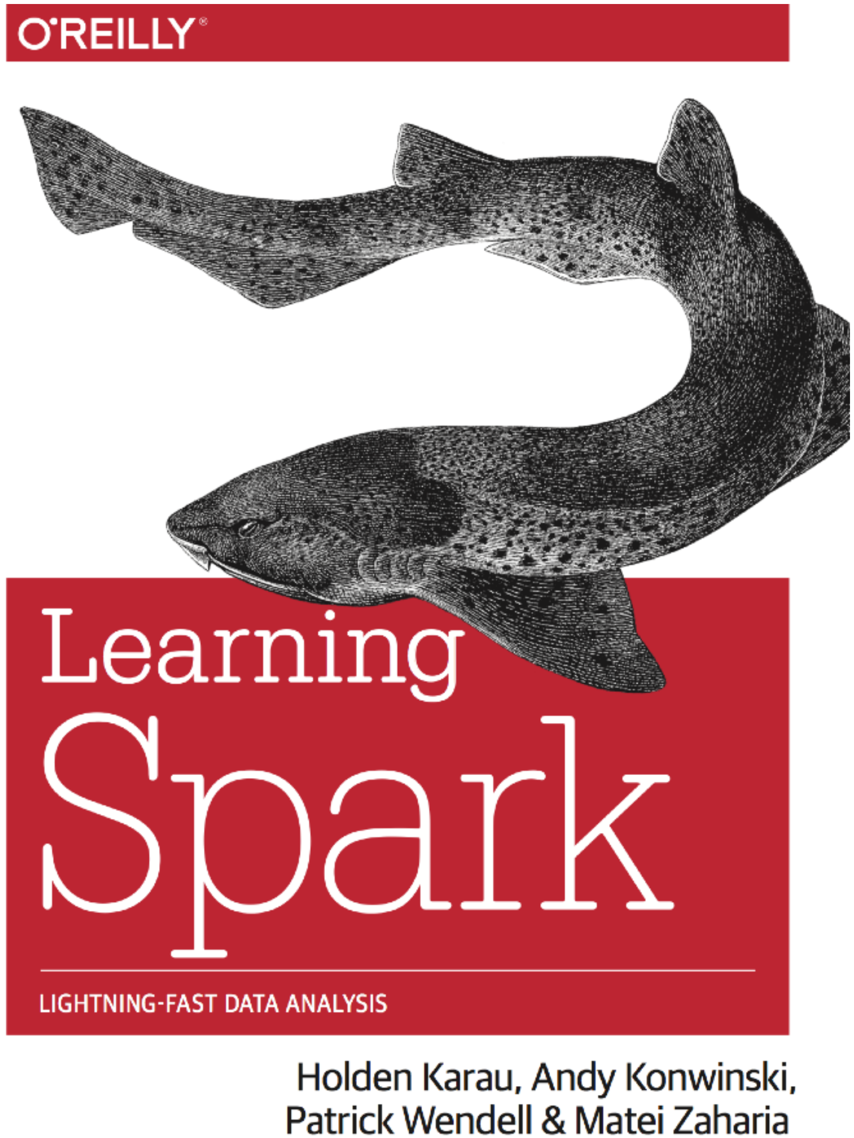
```
(A) CREATE TABLE IF NOT EXISTS FlightInfo2007 (
 Year SMALLINT, Month TINYINT, DayofMonth TINYINT, DayOfWeek TINYINT,
 DepTime SMALLINT, CRSDepTime SMALLINT, ArrTime SMALLINT, CRSArrTime SMALLINT,
 UniqueCarrier STRING, FlightNum STRING, TailNum STRING,
 ActualElapsedTime SMALLINT, CRSElapsedTime SMALLINT,
 AirTime SMALLINT, ArrDelay SMALLINT, DepDelay SMALLINT,
 Origin STRING, Dest STRING,Distance INT,
 TaxiIn SMALLINT, TaxiOut SMALLINT, Cancelled SMALLINT,
 CancellationCode STRING, Diverted SMALLINT,
 CarrierDelay SMALLINT, WeatherDelay SMALLINT,
 NASDelay SMALLINT, SecurityDelay SMALLINT, LateAircraftDelay SMALLINT)
COMMENT 'Flight InfoTable'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
TBLPROPERTIES ('creator'='Bruce Brown', 'created_at'='Thu Sep 19 10:58:00 EDT 2013');
```

Hive's operation modes

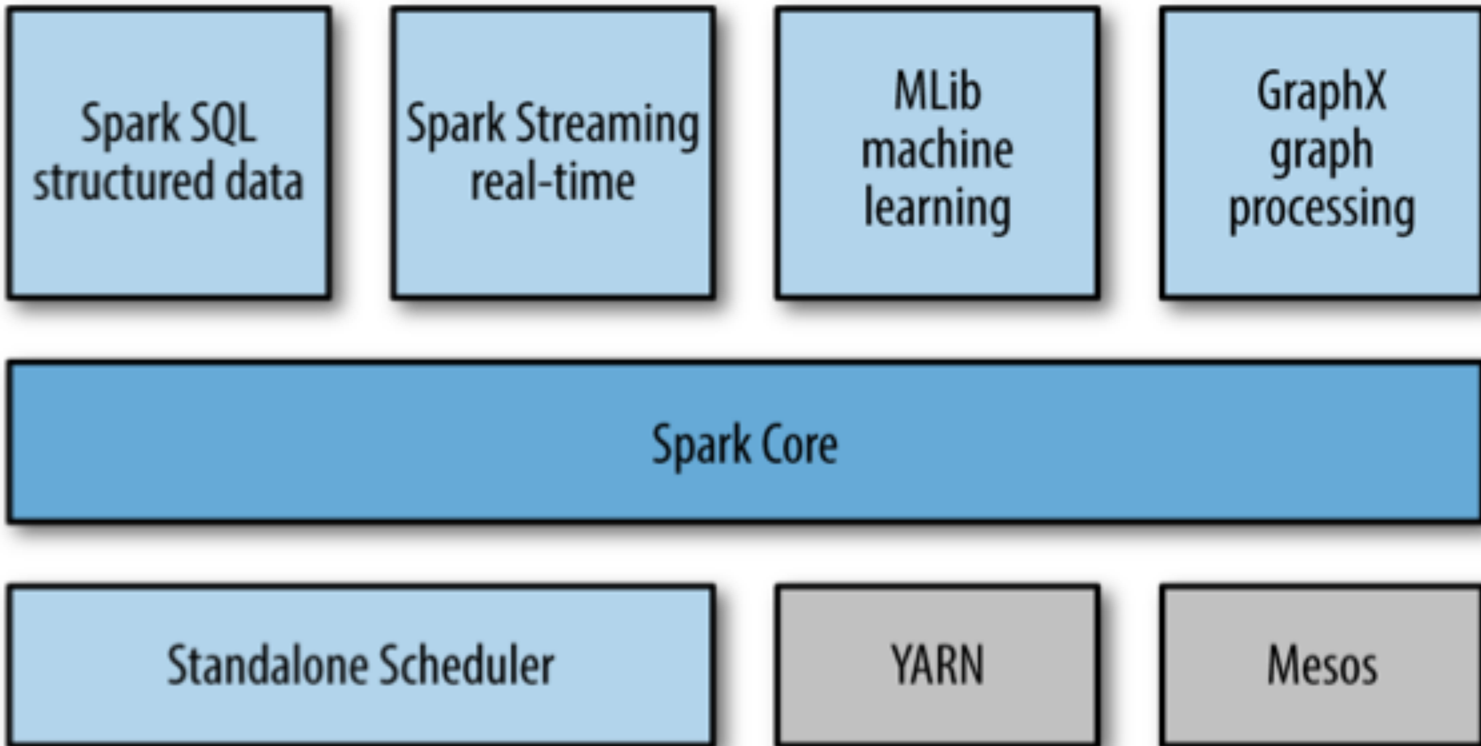




Reference


O'REILLY
Learning
Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia

Spark Stack



Spark SQL
structured data
Spark Streaming
real-time
MLib
machine
learning
GraphX
graph
processing
Spark Core
Standalone Scheduler
YARN
Mesos

Spark Core

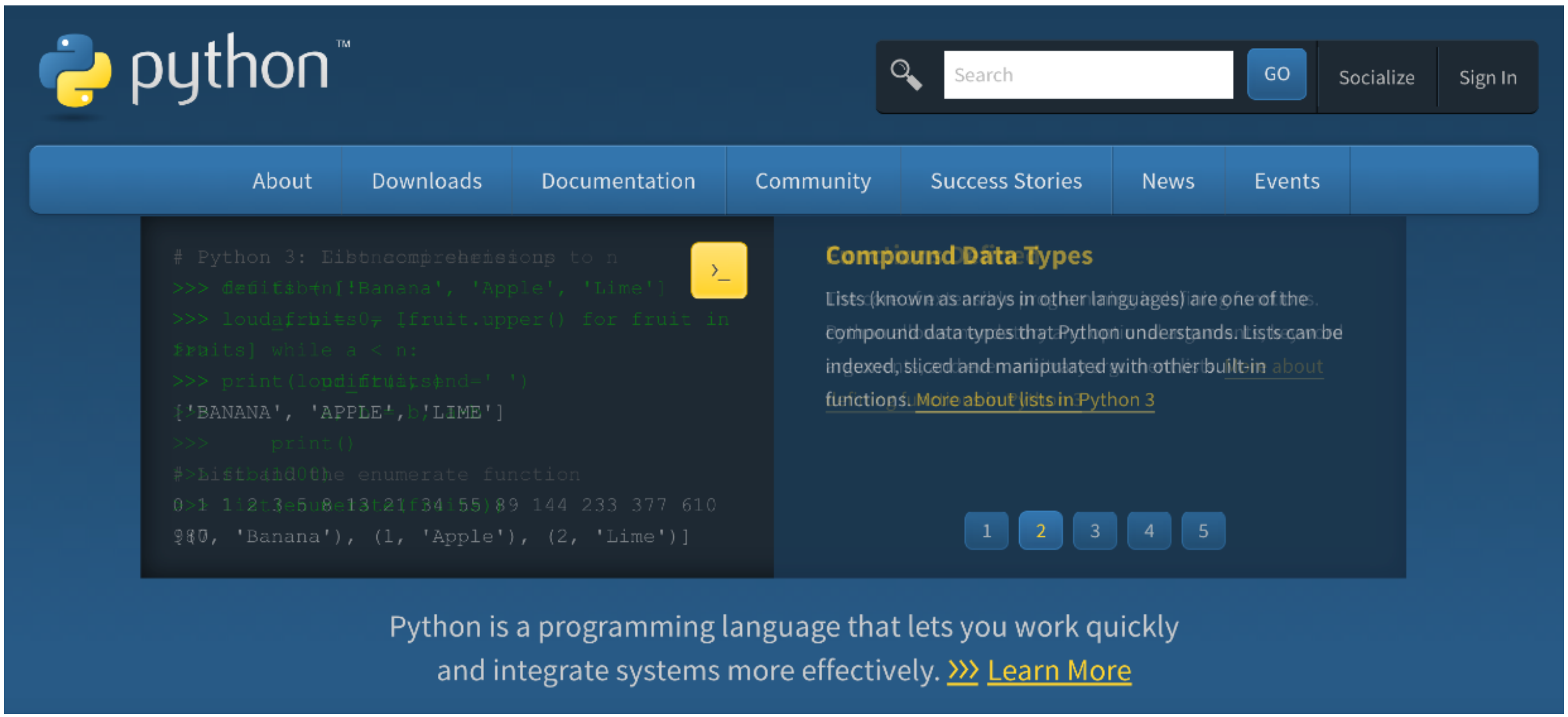
Basic functionality of Spark, including components for:

- Task Scheduling
- Memory Management
- Fault Recovery
- Interacting with Storage Systems
- and more

Home to the API that defines resilient distributed datasets (RDDs) - Spark's main programming abstraction.

RDD represents a collection of items distributed across many compute nodes that can be manipulated in parallel.

First language to use — Python



COLUMBIA UNIVERSITY

Spark's Python Shell (PySpark Shell)

bin/pyspark

holden@hmbp2: ~/Downloads/spark-1.1.0-bin-hadoop1 | holden@hmbp2: ~/Downloads/spark-1.1.0-bin-hadoop1 | holden@hmbp2: ~/Downloads/spark-1.1.0-bin-hadoop1

```
holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1$ ./bin/pyspark
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
14/11/19 14:33:49 WARN Utils: Your hostname, hmbp2 resolves to a loopback address: 127.0.1.1; using 172.17.42.1 instead (on interface docker0)
14/11/19 14:33:49 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
14/11/19 14:33:49 INFO SecurityManager: Changing view acls to: holden,
14/11/19 14:33:49 INFO SecurityManager: Changing modify acls to: holden,
14/11/19 14:33:49 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(holden, )
; users with modify permissions: Set(holden, )
14/11/19 14:33:49 INFO Slf4jLogger: Slf4jLogger started
14/11/19 14:33:49 INFO Remoting: Starting remoting
14/11/19 14:33:49 INFO Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriver@172.17.42.1:35021]
14/11/19 14:33:49 INFO Remoting: Remoting now listens on addresses: [akka.tcp://sparkDriver@172.17.42.1:35021]
14/11/19 14:33:49 INFO Utils: Successfully started service 'sparkDriver' on port 35021.
14/11/19 14:33:49 INFO SparkEnv: Registering MapOutputTracker
14/11/19 14:33:49 INFO SparkEnv: Registering BlockManagerMaster
14/11/19 14:33:49 INFO DiskBlockManager: Created local directory at /tmp/spark-local-20141119143349-5776
14/11/19 14:33:49 INFO Utils: Successfully started service 'Connection manager for block manager' on port 57218.
14/11/19 14:33:49 INFO ConnectionManager: Bound socket to port 57218 with id = ConnectionManagerId(172.17.42.1,57218)
14/11/19 14:33:49 INFO MemoryStore: MemoryStore started with capacity 265.4 MB
14/11/19 14:33:49 INFO BlockManagerMaster: Trying to register BlockManager
14/11/19 14:33:49 INFO BlockManagerMasterActor: Registering block manager 172.17.42.1:57218 with 265.4 MB RAM
14/11/19 14:33:49 INFO BlockManagerMaster: Registered BlockManager
14/11/19 14:33:49 INFO HttpFileServer: HTTP File server directory is /tmp/spark-399c53ec-0be8-4043-9a7d-9345e970576d
14/11/19 14:33:49 INFO HttpServer: Starting HTTP Server
14/11/19 14:33:49 INFO Utils: Successfully started service 'HTTP file server' on port 49008.
14/11/19 14:33:49 INFO Utils: Successfully started service 'SparkUI' on port 4040.
14/11/19 14:33:49 INFO SparkUI: Started SparkUI at http://172.17.42.1:4040
14/11/19 14:33:49 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@172.17.42.1:35021/user/HeartbeatReceiver
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.1.0
      /_/

Using Python version 2.7.6 (default, Mar 22 2014 22:59:56)
SparkContext available as sc.
>>>
```

Test installation

Example 2-1. Python line count

```
>>> lines = sc.textFile("README.md") # Create an RDD called lines

>>> lines.count() # Count the number of items in this RDD
127
>>> lines.first() # First item in this RDD, i.e. first line of README.md
u'# Apache Spark'
```

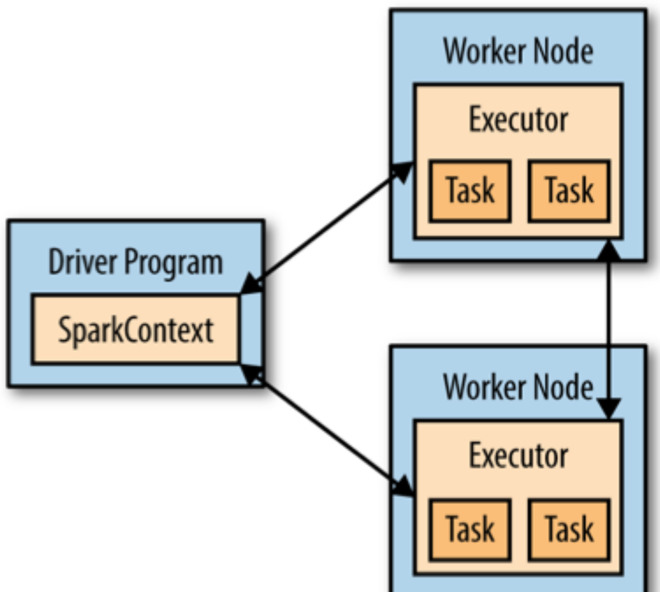
Core Spark Concepts

- At a high level, every Spark application consists of a **driver program** that launches various parallel operations on a cluster.
- The driver program contains your application’s main function and defines distributed databases on the cluster, then applies operations to them.
- In the preceding example, the driver program was the Spark shell itself.
- Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster.
- In the shell, a SparkContext is automatically created as the variable called sc.

Driver Programs

Driver programs typically manage a number of nodes called **executors**.

If we run the count() operation on a cluster, different machines might count lines in different ranges of the file.



Example filtering

```
>>> lines = sc.textFile("README.md")

>>> pythonLines = lines.filter(lambda line: "Python" in line)

>>> pythonLines.first()
u'## Interactive Python Shell'
```

lambda —> define functions inline in Python.

```
def hasPython(line):
    return "Python" in line

pythonLines = lines.filter(hasPython)
```

Example — word count

```
import sys
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    if len(sys.argv) != 2:
        print >> sys.stderr, "Usage: wordcount <file>"
        exit(-1)
    sc = SparkContext(appName="PythonWordCount")
    lines = sc.textFile(sys.argv[1], 1)
    counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print "%s: %i" % (word, count)

    sc.stop()
```

Resilient Distributed Dataset (RDD) Basics

- An RDD in Spark is an immutable distributed collection of objects.
- Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster.
- Users create RDDs in two ways: by loading an external dataset, or by distributing a collection of objects in their driver program.
- Once created, RDDs offer two types of operations: **transformations** and **actions**.

```
>>> lines = sc.textFile("README.md")
```
<== create RDD

```
>>> pythonLines = lines.filter(lambda line: "Python" in line)
```
<== transformation

```
>>> pythonLines.first()
u'## Interactive Python Shell'
```
<== action

Transformations and actions are different because of the way Spark computes RDDs.
==> Only computes when something is, the first time, in an action.

Persistance in Spark

- By default, RDDs are computed each time you run an action on them.
- If you like to reuse an RDD in multiple actions, you can ask Spark to persist it using RDD.persist().
- RDD.persist() will then store the RDD contents in memory and reuse them in future actions.
- Persisting RDDs on disk instead of memory is also possible.
- The behavior of not persisting by default seems to be unusual, but it makes sense for big data.

Example 3-4. Persisting an RDD in memory

```
>>> pythonLines.persist

>>> pythonLines.count()
2

>>> pythonLines.first()
u'## Interactive Python Shell'
```

To summarize, every Spark program and shell session will work as follows:

1. Create some input RDDs from external data.
2. Transform them to define new RDDs using transformations like `filter()`.
3. Ask Spark to `persist()` any intermediate RDDs that will need to be reused.
4. Launch actions such as `count()` and `first()` to kick off a parallel computation, which is then optimized and executed by Spark.

Spark SQL

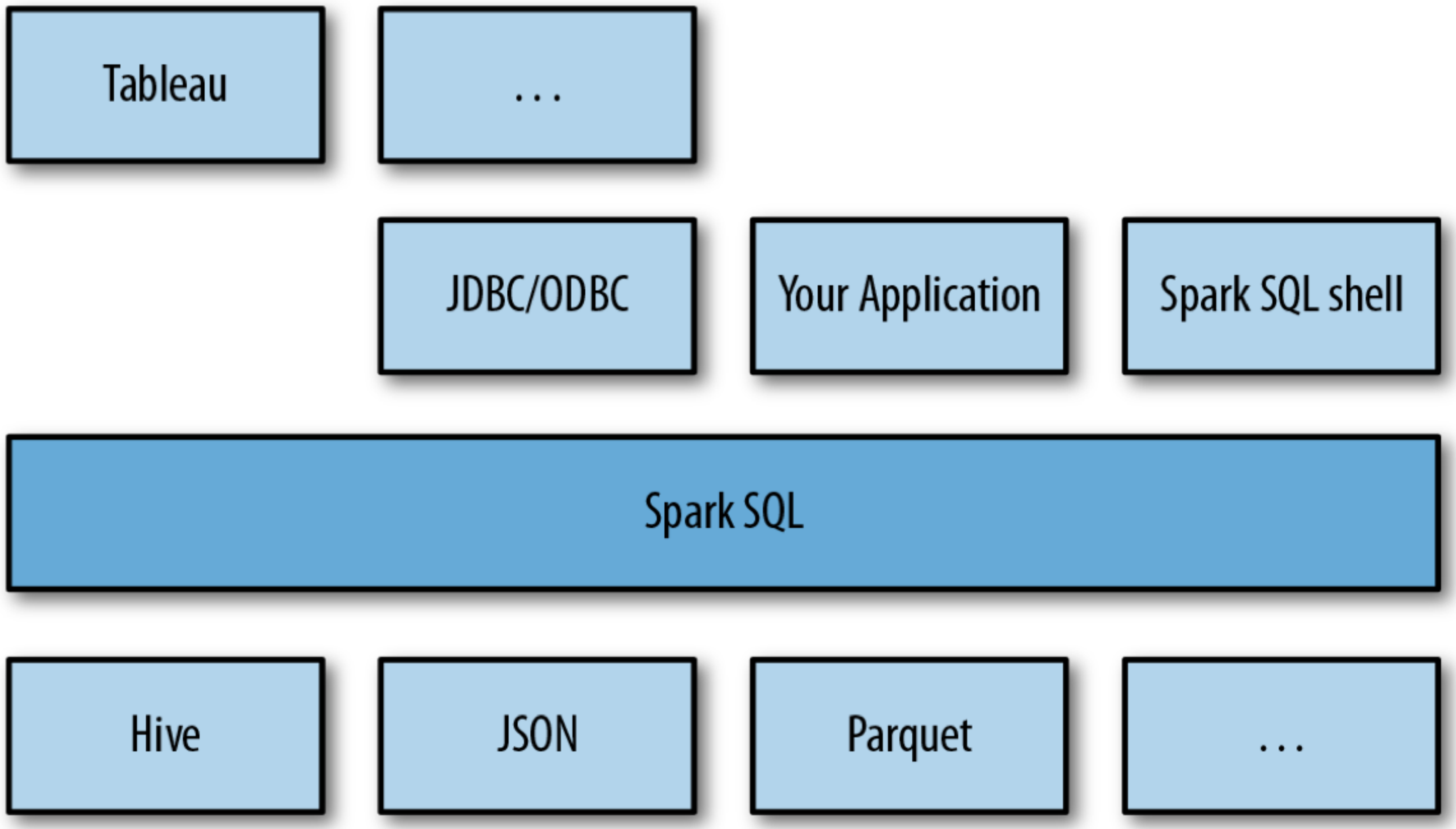


Tableau
...
JDBC/ODBC
Your Application
Spark SQL shell
Spark SQL
Hive
JSON
Parquet
...

Spark SQL

Spark SQL can be built with or without Apache Hive, the Hadoop SQL engine. Spark SQL with Hive support allows us to access Hive tables, UDFs (user-defined functions), SerDes (serialization and deserialization formats), and the Hive query language (HiveQL). Hive query language (HQL) It is important to note that including the Hive libraries does not require an existing Hive installation. In general, it is best to build Spark SQL with Hive support to access these features. If you download Spark in binary form, it should already be built with Hive support. If you are building Spark from source, you should run `sbt/sbt -Phive assembly`.

Using Spark SQL — Steps and Example

Example 9-5. Python SQL imports

```
# Import Spark SQL
from pyspark.sql import HiveContext, Row
```

Example 9-8. Constructing a SQL context in Python

```
hiveCtx = HiveContext(sc)
```

Example 9-11. Loading and quering tweets in Python

```
input = hiveCtx.jsonFile(inputFile)
# Register the input schema RDD
input.registerTempTable("tweets")
# Select tweets based on the retweetCount
topTweets = hiveCtx.sql("""SELECT text, retweetCount  FROM
  tweets ORDER BY retweetCount LIMIT 10""")
```

Columbia University

Query testtweet.json

Get it from Learning Spark Github ==> https://github.com/databricks/learning-spark/tree/master/files

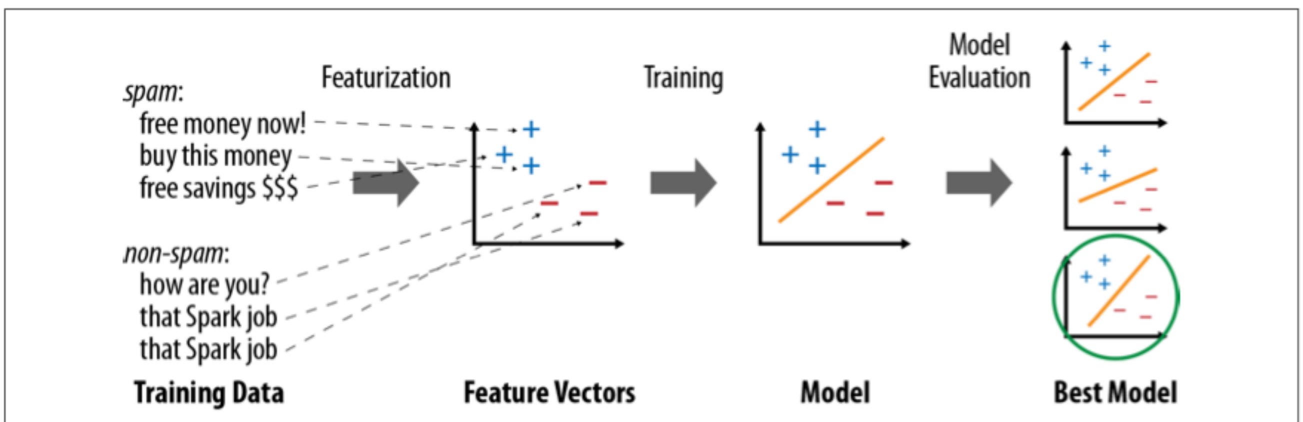

```
{"createdAt":"Nov 4, 2014 4:56:59 PM","id":529799371026485248,"text":"Adventures With
Coffee, Code, and Writing.","source":"\u003ca href\u003d\"http://twitter.com\"
rel\u003d\"nofollow\"\u003eTwitter Web
Client\u003c/a\u003e","isTruncated":false,"inReplyToStatusId":-1,"inReplyToUserId":-1,"
isFavorited":false,"retweetCount":0,"isPossiblySensitive":false,"contributorsIDs":[],"
userMentionEntities":[],"urlEntities":[],"hashtagEntities":[],"mediaEntities":[],"
currentUserRetweetId":-1,"user":{"id":15594928,"name":"Holden
Karau","screenName":"holdenkarau","location":"","description":"","descriptionURLEntities":
[],"isContributorsEnabled":false,"profileImageUrl":"http://pbs.twimg.com/profile_images/
3005696115/2036374bbadbed85249cdd50aac6e170_normal.jpeg","profileImageUrlHttps":"https://
pbs.twimg.com/profile_images/3005696115/2036374bbadbed85249cdd50aac6e170_normal.jpeg","
isProtected":false,"followersCount":1231,"profileBackgroundColor":"C0DEED","
profileTextColor":"333333","profileLinkColor":"0084B4","profileSidebarFillColor":"DDEEF6",
"profileSidebarBorderColor":"FFFFFF","profileUseBackgroundImage":true,"showAllInlineMedia"
:false,"friendsCount":600,"createdAt":"Aug 5, 2011 9:42:44
AM","favouritesCount":1095,"utcOffset":-3,"profileBackgroundImageUrl":"","
profileBackgroundImageUrlHttps":"","profileBannerImageUrl":"","profileBackgroundTiled":
true,"lang":"en","statusesCount":6234,"isGeoEnabled":true,"isVerified":false,"translator":
false,"listedCount":0,"isFollowRequestSent":false}}
```


```
>>> print topTweets.collect()
[Row(text=u'Adventures With Coffee, Code, and Writing.', retweetCount=0)]
```

Machine Learning Library in Spark — MLlib

An example of using MLlib for text classification task, e.g., identifying spammy emails.

1. Start with an RDD of strings representing your messages.
2. Run one of MLlib's *feature extraction* algorithms to convert text into numerical features (suitable for learning algorithms); this will give back an RDD of vectors.
3. Call a classification algorithm (e.g., logistic regression) on the RDD of vectors; this will give back a model object that can be used to classify new points.
4. Evaluate the model on a test dataset using one of MLlib's evaluation functions.



MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- K-means
  - Input Columns
  - Output Columns
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)
  - Input Columns
  - Output Columns

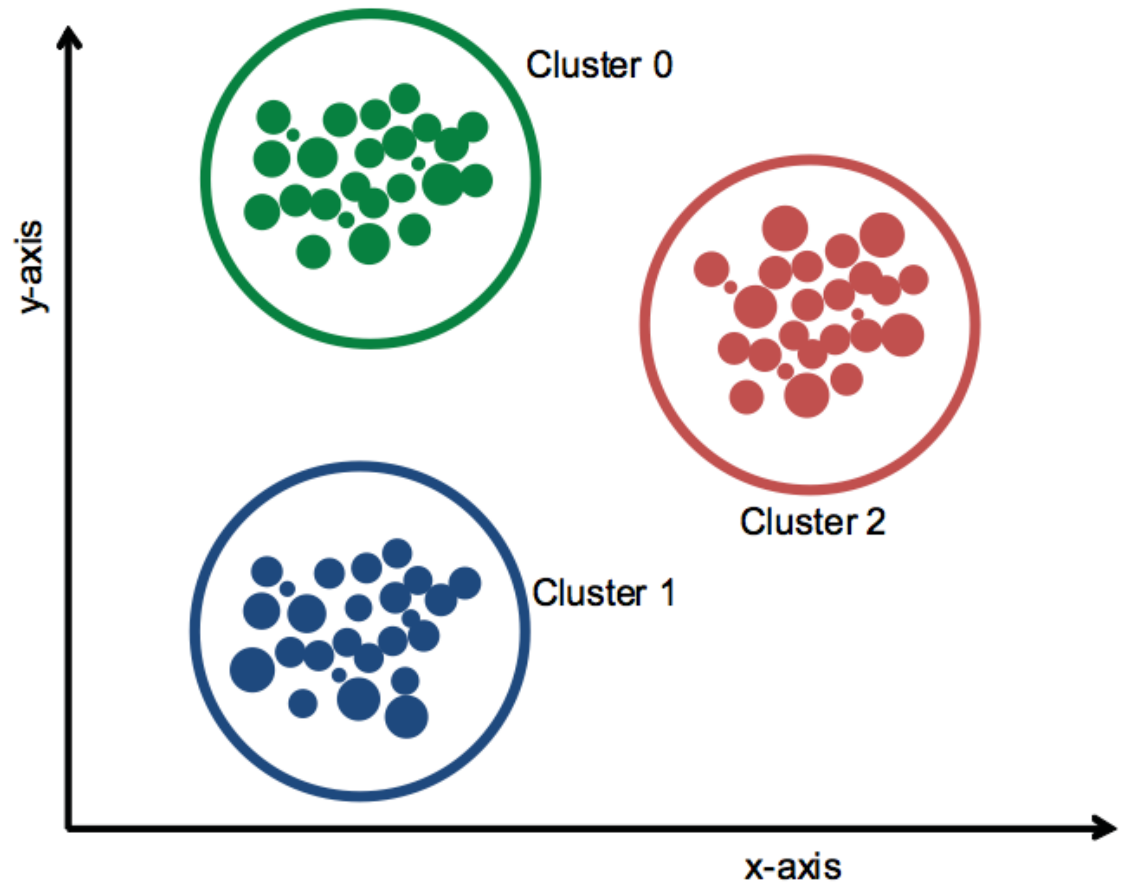
Example: clustering

Feature
Space

Clustering a collection involves three things:

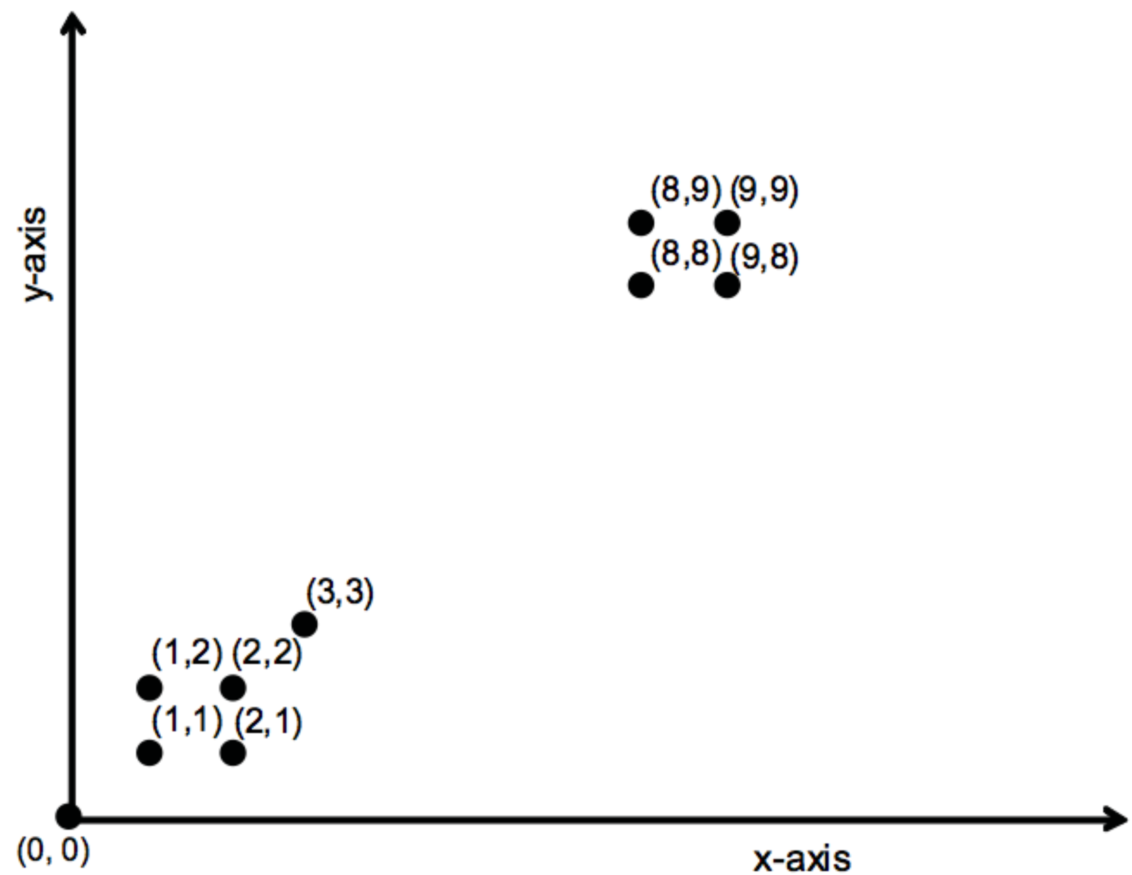
- *An algorithm*—This is the method used to group the books together.
- *A notion of both similarity and dissimilarity*—In the previous discussion, we relied on your assessment of which books belonged in an existing stack and which should start a new one.
- *A stopping condition*—In the library example, this might be the point beyond which books can't be stacked anymore, or when the stacks are already quite dissimilar.

Clustering — on feature plane

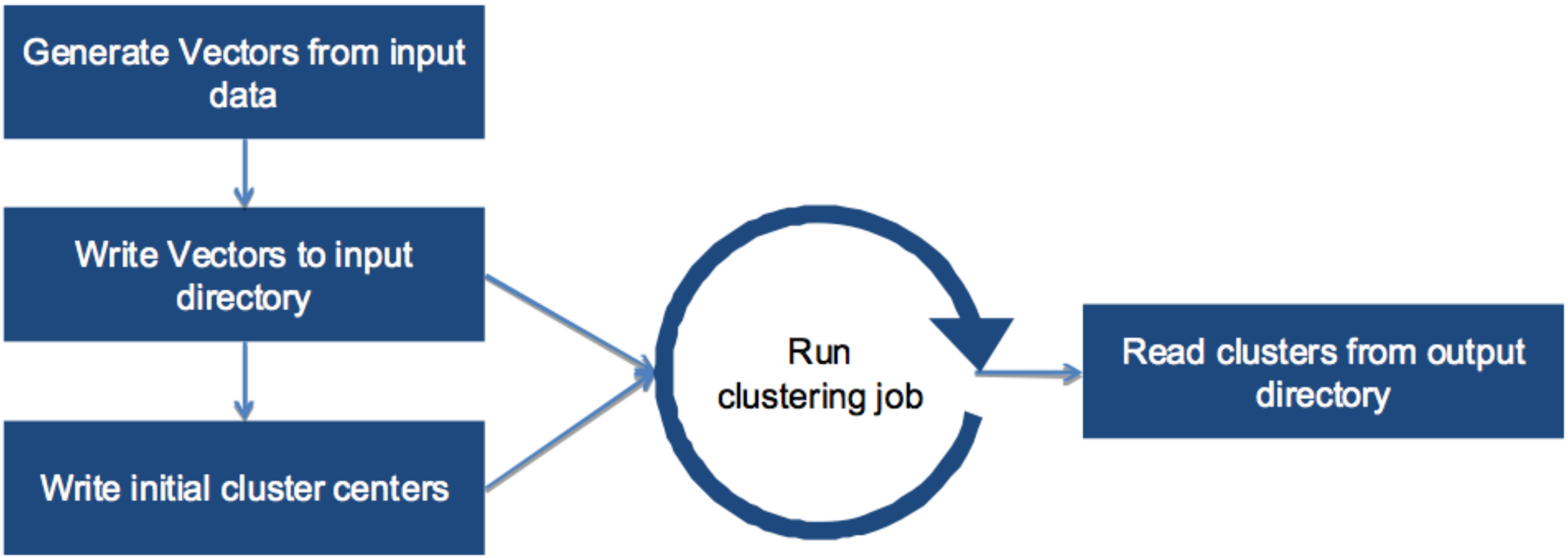


Clustering example

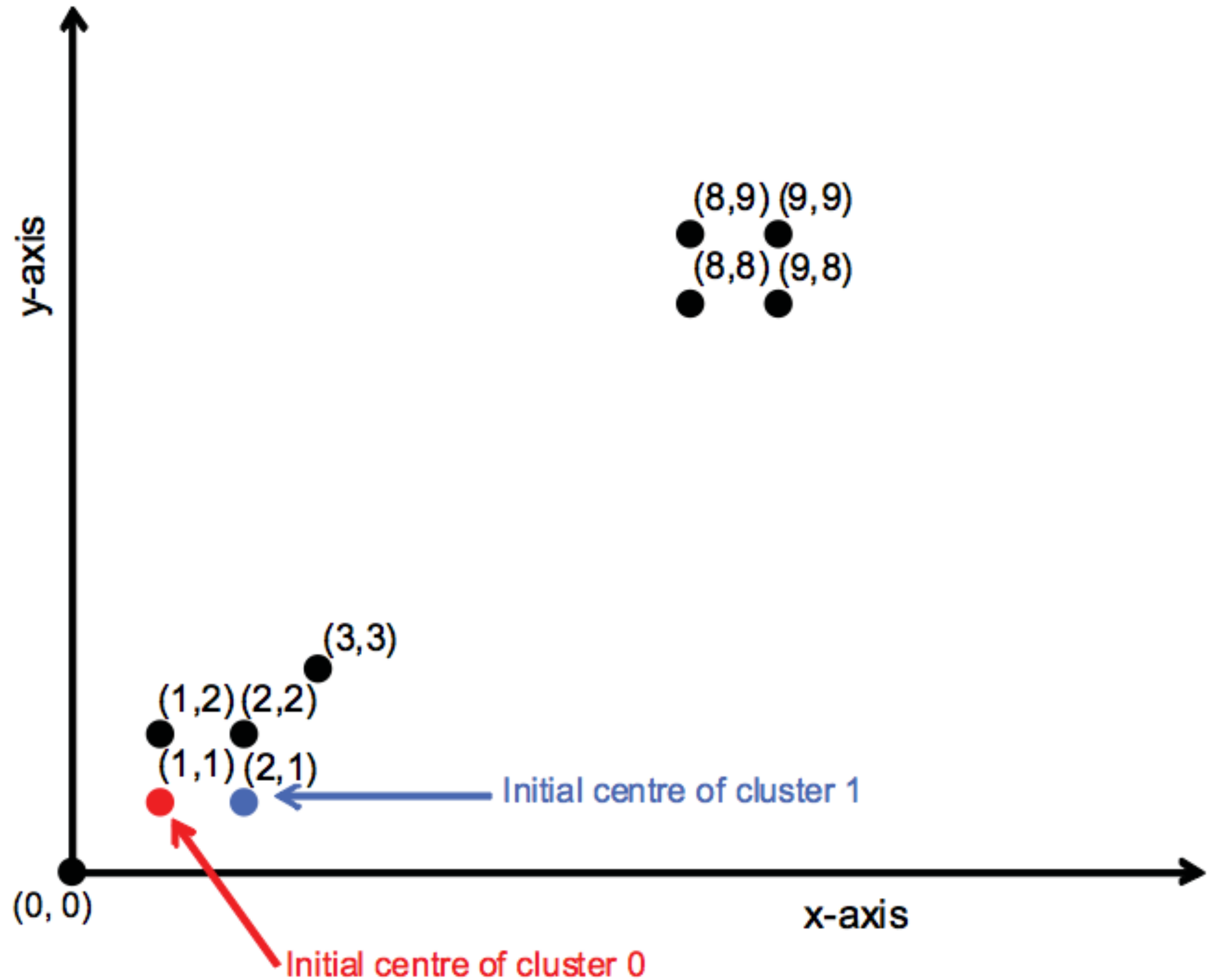
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)



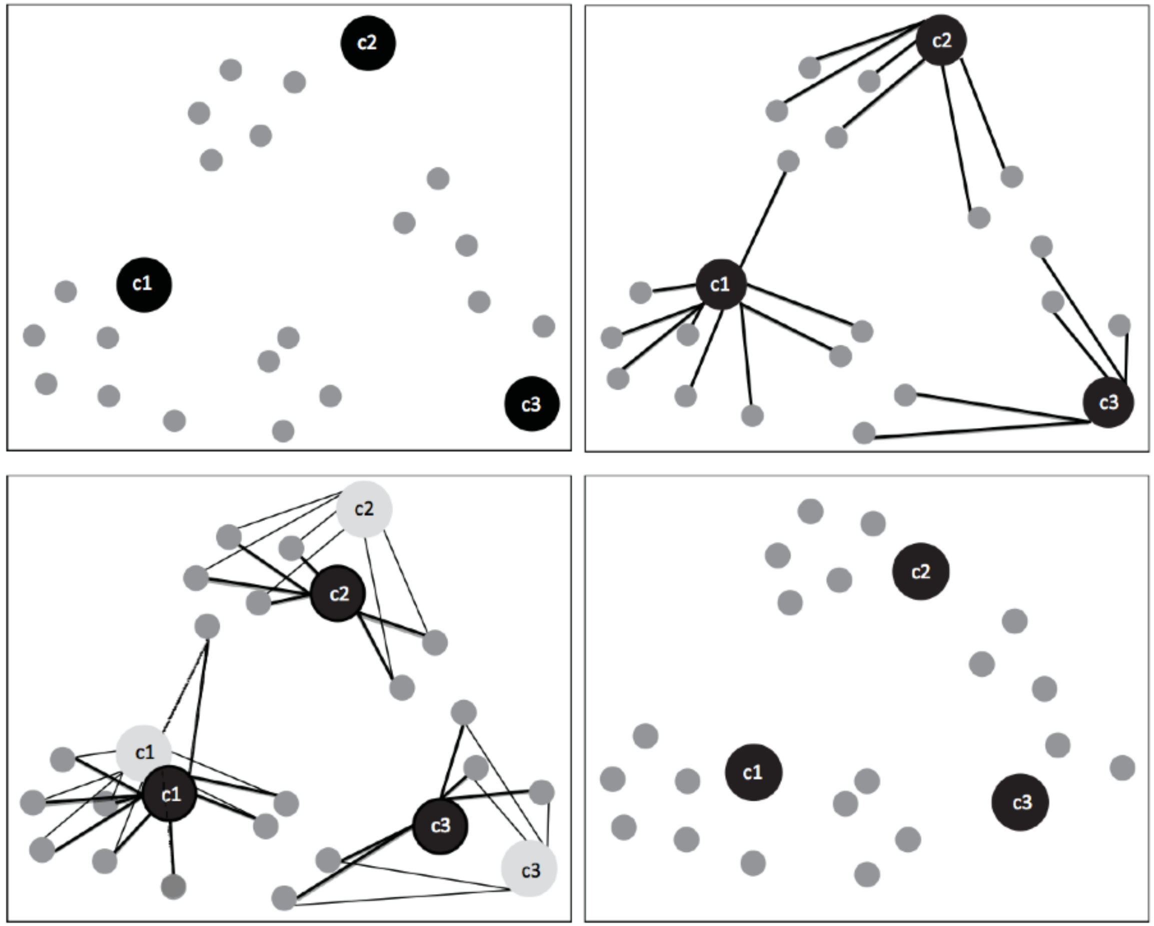
Steps on clustering



Making initial cluster centers



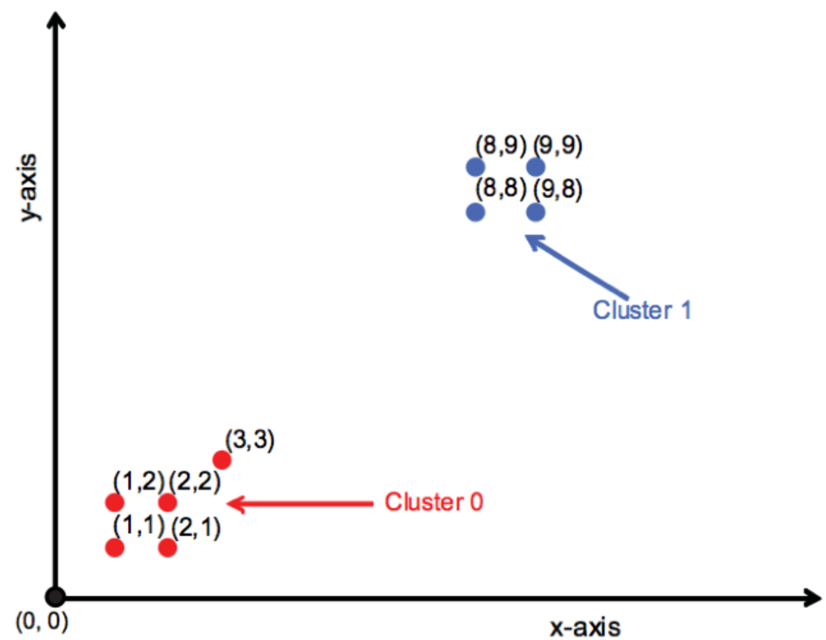
K-mean clustering



K-means clustering in action. Starting with three random points as centroids (top left), the map stage (top right) assigns each point to the cluster nearest to it. In the reduce stage (bottom left), the associated points are averaged out to produce the new location of the centroid, leaving you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.

HelloWorld clustering scenario result

```
1.0: [1.000, 1.000] belongs to cluster 0
1.0: [2.000, 1.000] belongs to cluster 0
1.0: [1.000, 2.000] belongs to cluster 0
1.0: [2.000, 2.000] belongs to cluster 0
1.0: [3.000, 3.000] belongs to cluster 0
1.0: [8.000, 8.000] belongs to cluster 1
1.0: [9.000, 8.000] belongs to cluster 1
1.0: [8.000, 9.000] belongs to cluster 1
1.0: [9.000, 9.000] belongs to cluster 1
```



Testing difference distance measures

Euclidean distance measure

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

Squared Euclidean distance measure

$$d = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2$$

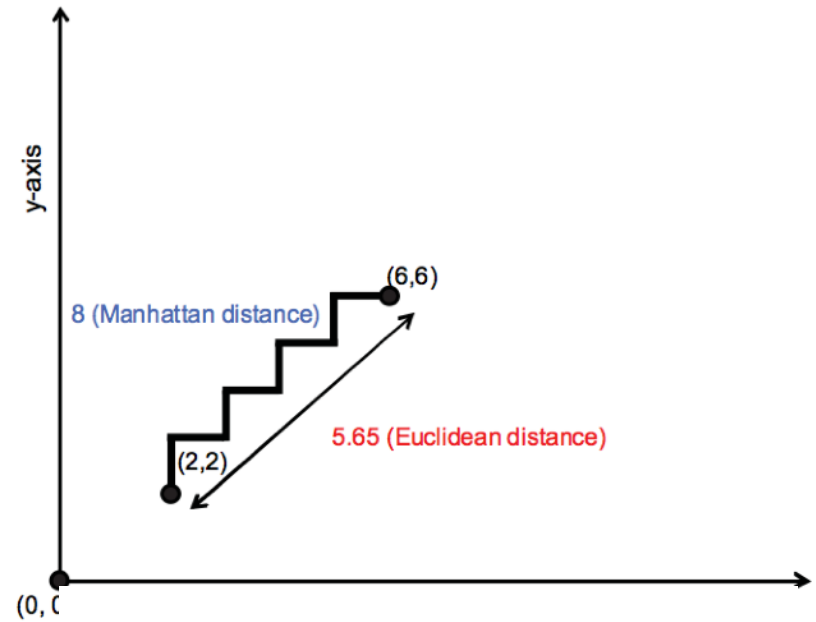
Manhattan distance measure

$$d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

Manhattan and Cosine distances

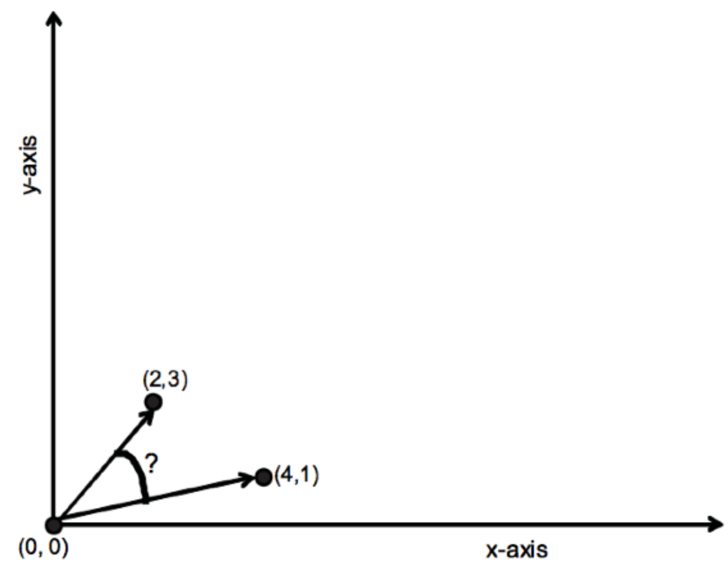
Manhattan distance measure

$$d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$



Cosine distance measure

$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\left(\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)}\sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)}\right)}$$



Tanimoto distance measure

$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\sqrt{(a_1^{\,2} + a_2^{\,2} + \ldots + a_n^{\,2})} + \sqrt{(b_1^{\,2} + b_2^{\,2} + \ldots + b_n^{\,2})} - (a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}$$

Weighted distance measure

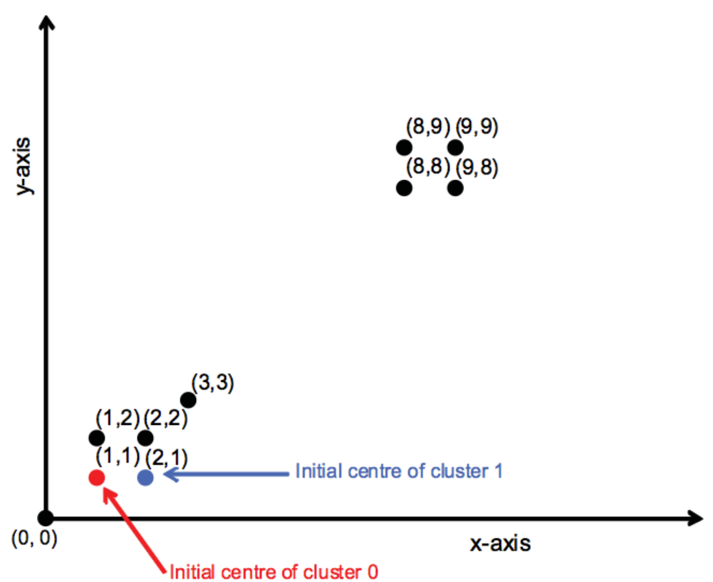
Mahout also provides a `WeightedDistanceMeasure` class, and implementations of Euclidean and Manhattan distance measures that use it. A weighted distance measure is an advanced feature in Mahout that allows you to give weights to different dimensions in order to either increase or decrease the effect of a dimension

Results comparison

Columbia University

Distance measure	Number of iterations	Vectors[a] in cluster 0	Vectors in cluster 1
EuclideanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
SquaredEuclideanDistanceMeasure	5	0, 1, 2, 3, 4	5, 6, 7, 8
ManhattanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
CosineDistanceMeasure	1	1	0, 2, 3, 4, 5, 6, 7, 8
TanimotoDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8

```
(1,1)
(2,1)
(1,2)
(2,2)
(3,3)
(8,8)
(8,9)
(9,8)
(9,9)
```

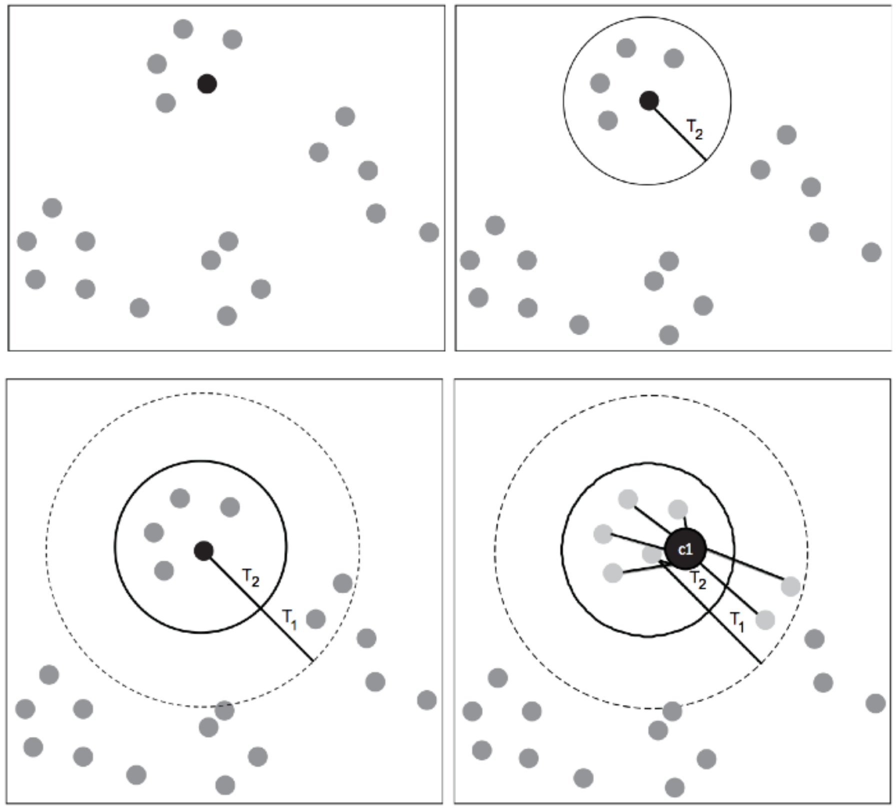


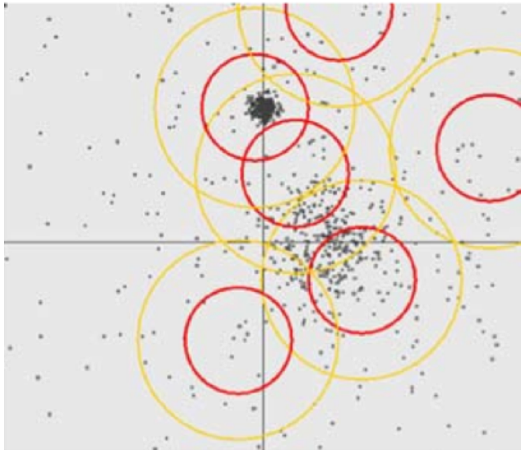
Sample code of K-mean clustering in Spark

```python
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Loads data.
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Make predictions
predictions = model.transform(dataset)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Canopy clustering to estimate the number of clusters

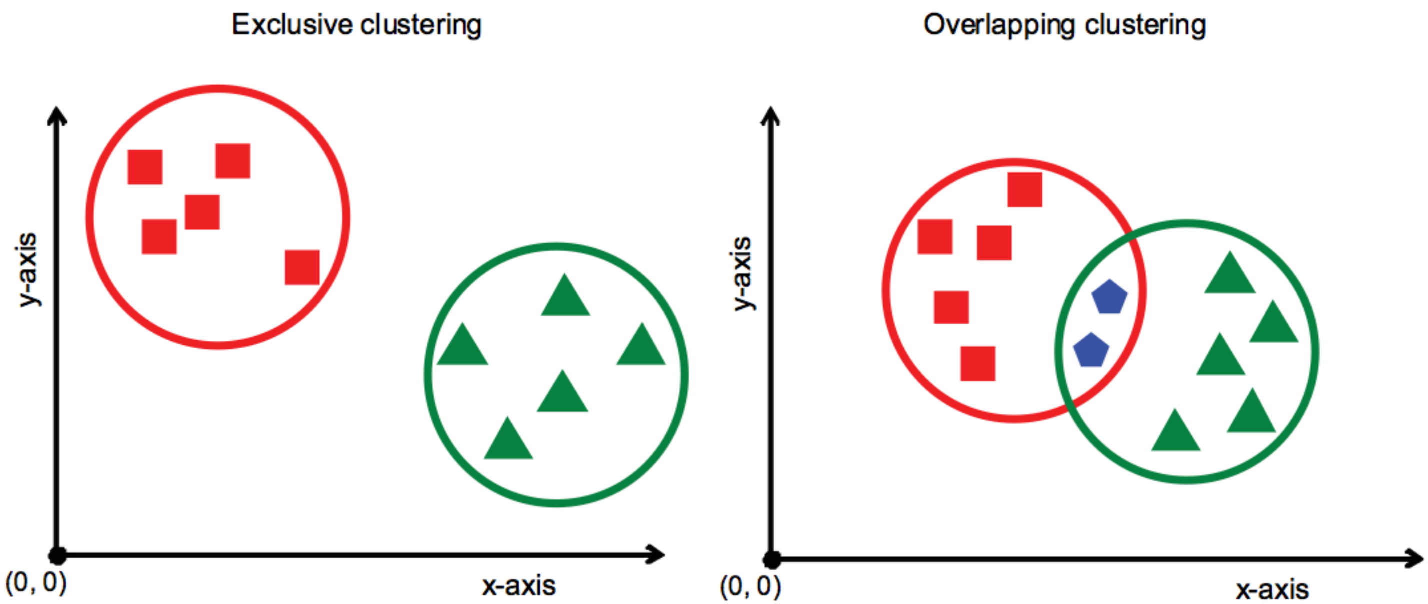
Tell what size clusters to look for. The algorithm will find the number of clusters that have approximately that size. The algorithm uses two distance thresholds. This method prevents all points close to an already existing canopy from being the center of a new canopy.



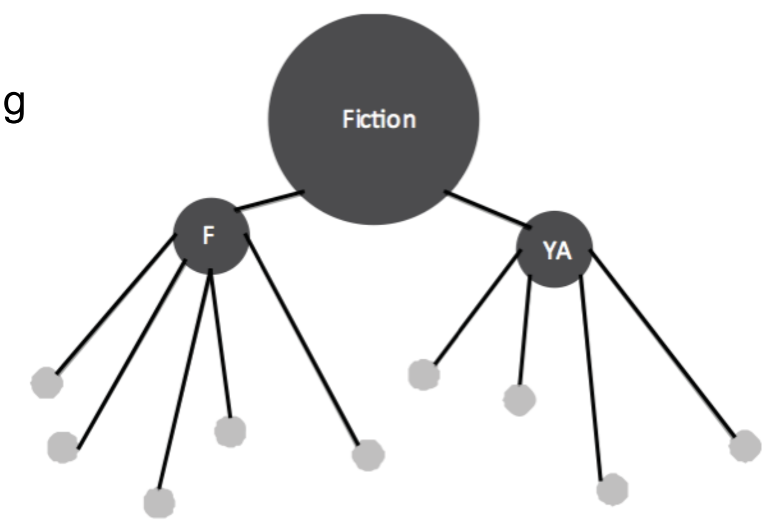


Canopy clustering: if you start with a point (top left) and mark it as part of a canopy, all the points within distance T_2 (top right) are removed from the data set and prevented from becoming new canopies. The points within the outer circle (bottom-right) are also put in the same canopy, but they're allowed to be part of other canopies. This assignment process is done in a single pass on a mapper. The reducer computes the average of the centroid (bottom right) and merges close canopies.

Other clustering algorithms



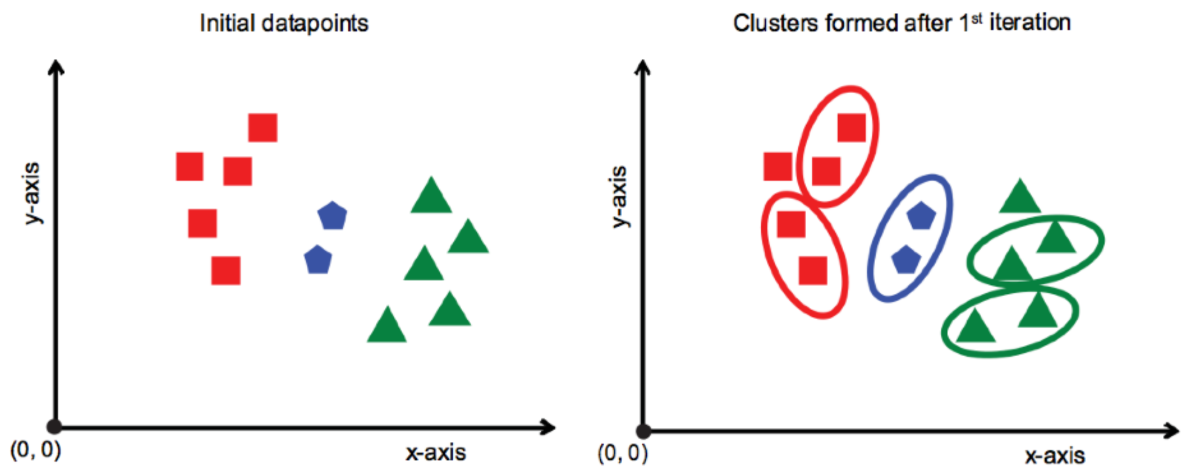
Hierarchical clustering



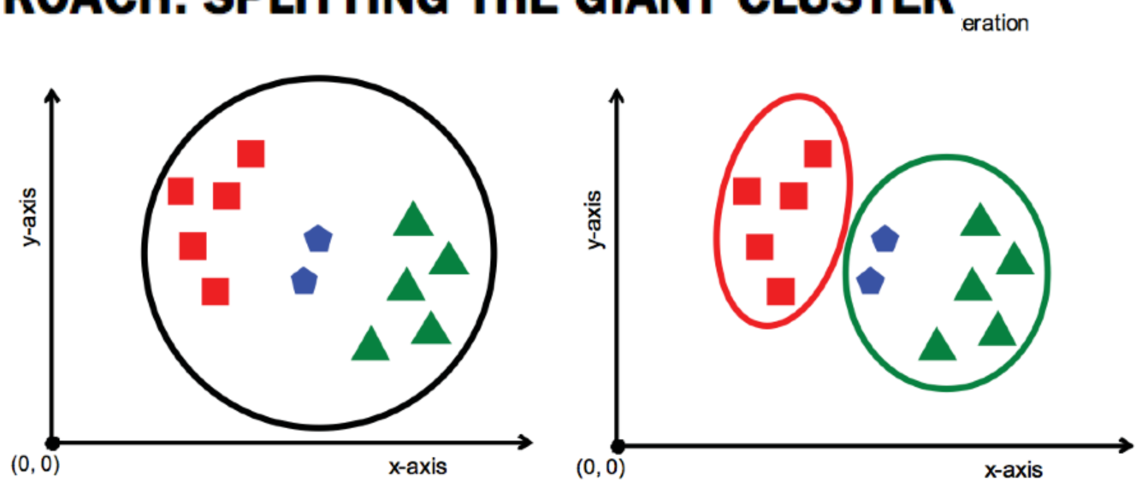
Different clustering approaches

FIXED NUMBER OF CENTERS

BOTTOM-UP APPROACH: FROM POINTS TO CLUSTERS VIA GROUPING



TOP-DOWN APPROACH: SPLITTING THE GIANT CLUSTER



Questions?