Module 4 :

Big Data Analysis using Apache Spark

ALY 6110: Big Data and Data Management

Instructor: Ajit Appari

Module 3: Topics

- Introduction to Spark
- Key Modules
- How does Spark work?
- Installation of Spark
- Load data into Spark
- Building Spark Application using Maven

- Reference Books for Machine Learning using Spark
  - Adi Polak (2023). Scaling Machine Learning with Spark : distributed ML with MLlib, Tensorflow, and PyTorch
  - Luu Hien (2021). Beginning Apache Spark 3 : with DataFrame, Spark SQL, structured streaming, and Spark machine learning library
  - Jules S. Damji, Denny Lee, Brooke Wenig, Tathagata Das, Denny Lee (2020). Learning Spark: lightning-fast data analytics
  - Butch Quinto (2020) Next-Generation Machine Learning with Spark : Covers XGBoost, LightGBM, Spark NLP, Distributed Deep Learning with Keras, and More.
  - Javier Luraschi, Kevin Kuo, Edgar Ruiz (2019). Mastering Spark with R: https://therinspark.com/
  - Micheal Bowles (2019). Machine Learning with Spark and Python.

Introduction to Spark

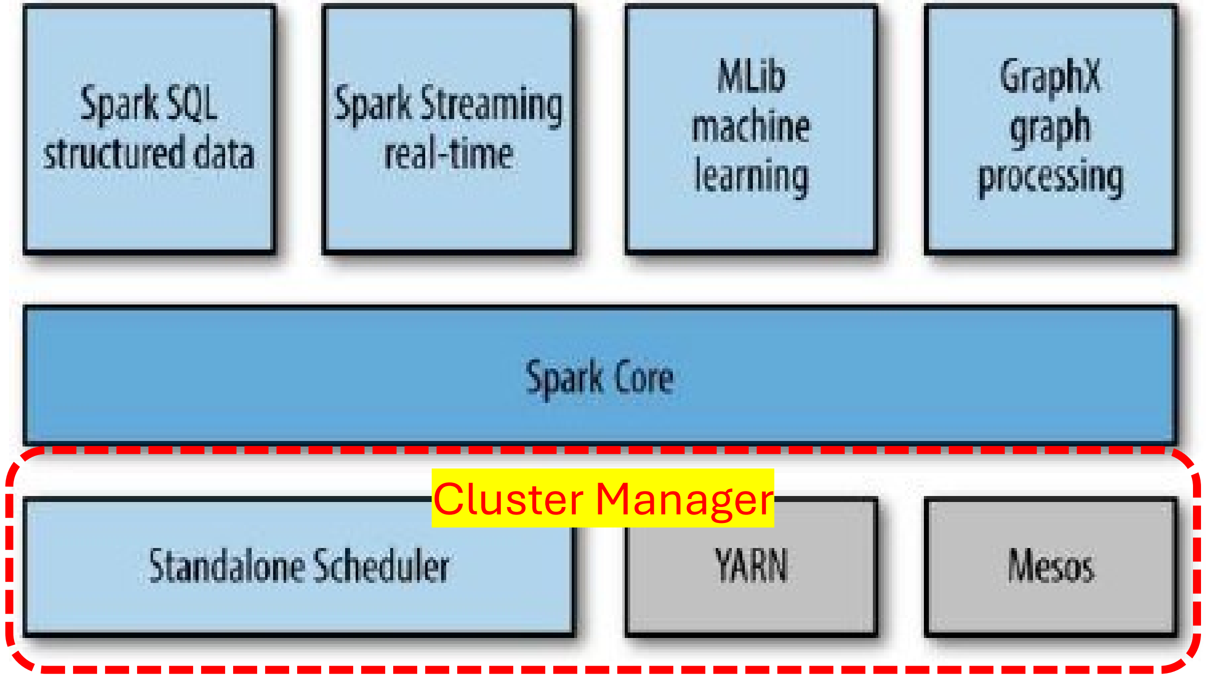
Introduction to Spark

- Spark is an open-source software solution that performs rapid calculations on in-memory distributed datasets referred as RDDs (Resilient Distributed Datasets);
  - RDD is a distributed collections of objects that can be cached in memory across cluster and can be manipulated in parallel;
  - RDD could be automatically recomputed on failure;
- Spark is designed as extension of MapReduce model to efficiently support a wide range of computational workloads including batch applications, iterative algorithms, interactive queries and stream processing;
- Spark is designed to be highly accessible:
  - offers simple APIs in Python, Java, Scala, SQL, and
  - Has rich built-in libraries;

Key Modules

Key Modules

- Spark has six core modules
- Spark Core: contains the basic functionality of Spark, including components:
  - Task scheduling;
  - Memory management;
  - Fault recovery;
  - Interacting with storage systems, etc.



- Spark SQL: is Spark’s package for working with structured data
  - allows querying data via SQL as well as the Apache Hive variant of SQL—Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON
  - allows intermixing of SQL queries with the programmatic data manipulations in Python, Java, and Scala, all within a single application, thus combining SQL with complex analytics.

Key Modules

- **Spark Streaming:** enables processing of live streams of data such as log files of web servers, or queues of messages.
  - **Spark Streaming API** for manipulating data streams closely matches the Spark Core's RDD API, making it easy to move between apps to manipulate data in memory, on disk, or arriving in real time;

- **MLib:** is a library containing common machine learning functionalities
  - Provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import;
  - All of ML methods are designed to scale out across a cluster.

- **GraphX:** is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations;
  - Extends the Spark RDD API, allowing to create a directed graph with arbitrary properties attached to each vertex and edge;
  - Provides various operators for manipulating graphs (e.g., subgraph and mapVertices) and a library of common graph algorithms (e.g., PageRank and triangle counting).

Key Modules

- **Cluster Manager:** allow Spark to efficiently scale up from one to many thousands of compute nodes;
  - Spark can run over a variety of cluster managers, including
    - Hadoop YARN,
    - Apache Mesos,
    - Standalone Scheduler and a simple cluster manager included in Spark itself);

How does Spark work?

How does Spark work?

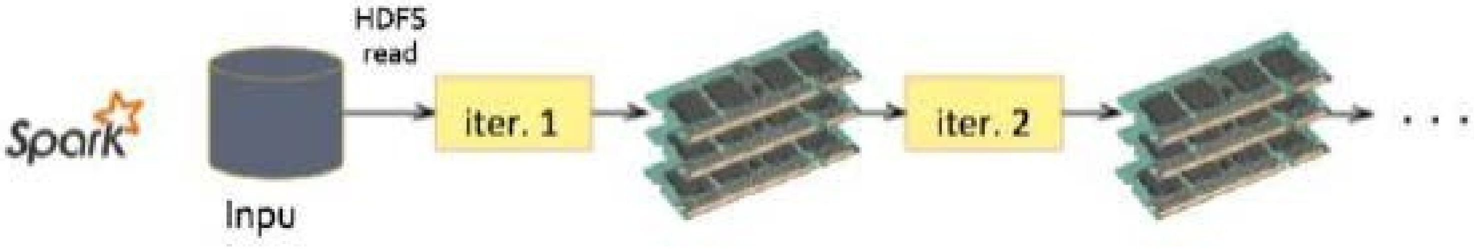
- RDD: Your data is loaded in parallel into structured collections
- Actions: Manipulate the state of the working model by forming new RDDs and performing calculations upon them
- Persistence: Long-term storage of an RDD's state

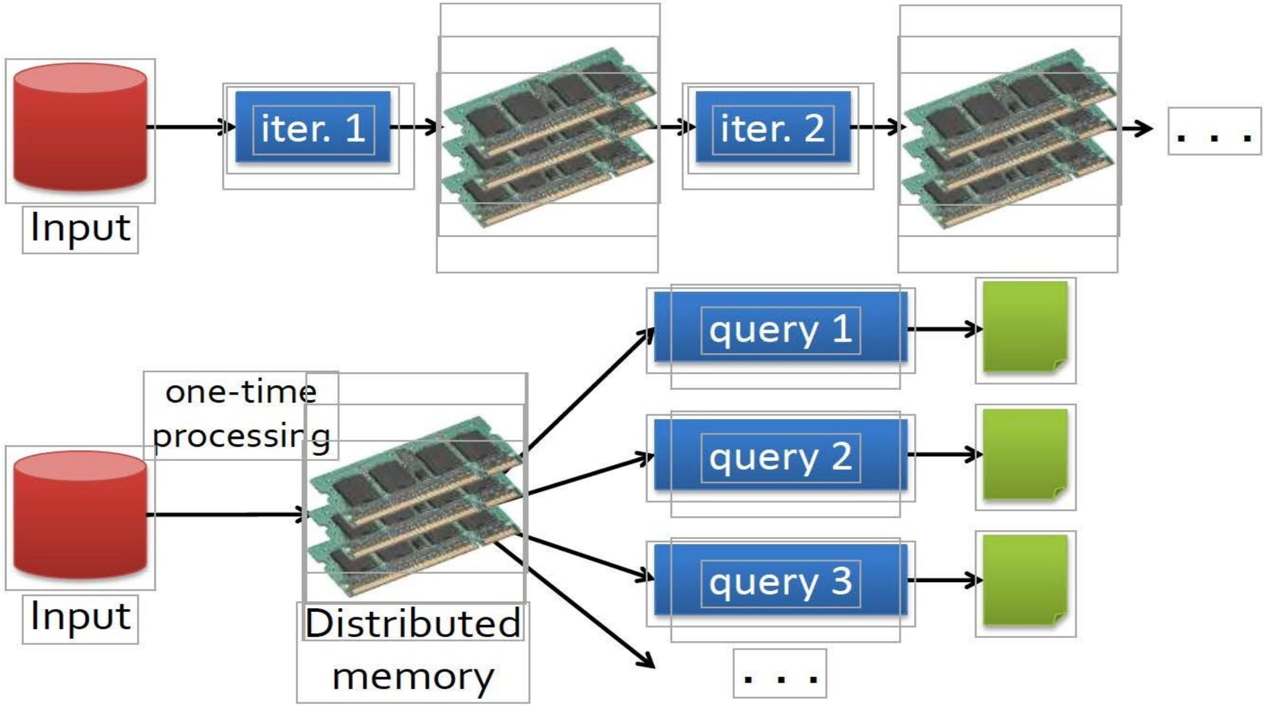
- Spark Application is a definition in code of
  - RDD creation
  - Actions
  - Persistence

- Spark Application results in the creation of a DAG (Directed Acyclic Graph)
  - Each DAG is compiled into stages
  - Each Stage is executed as a series of Tasks
  - Each Task operates in parallel on assigned partitions
  - It all starts with the SparkContext 'sc'

How does Spark work?

Spark is much smarter, it keeps all results in memory





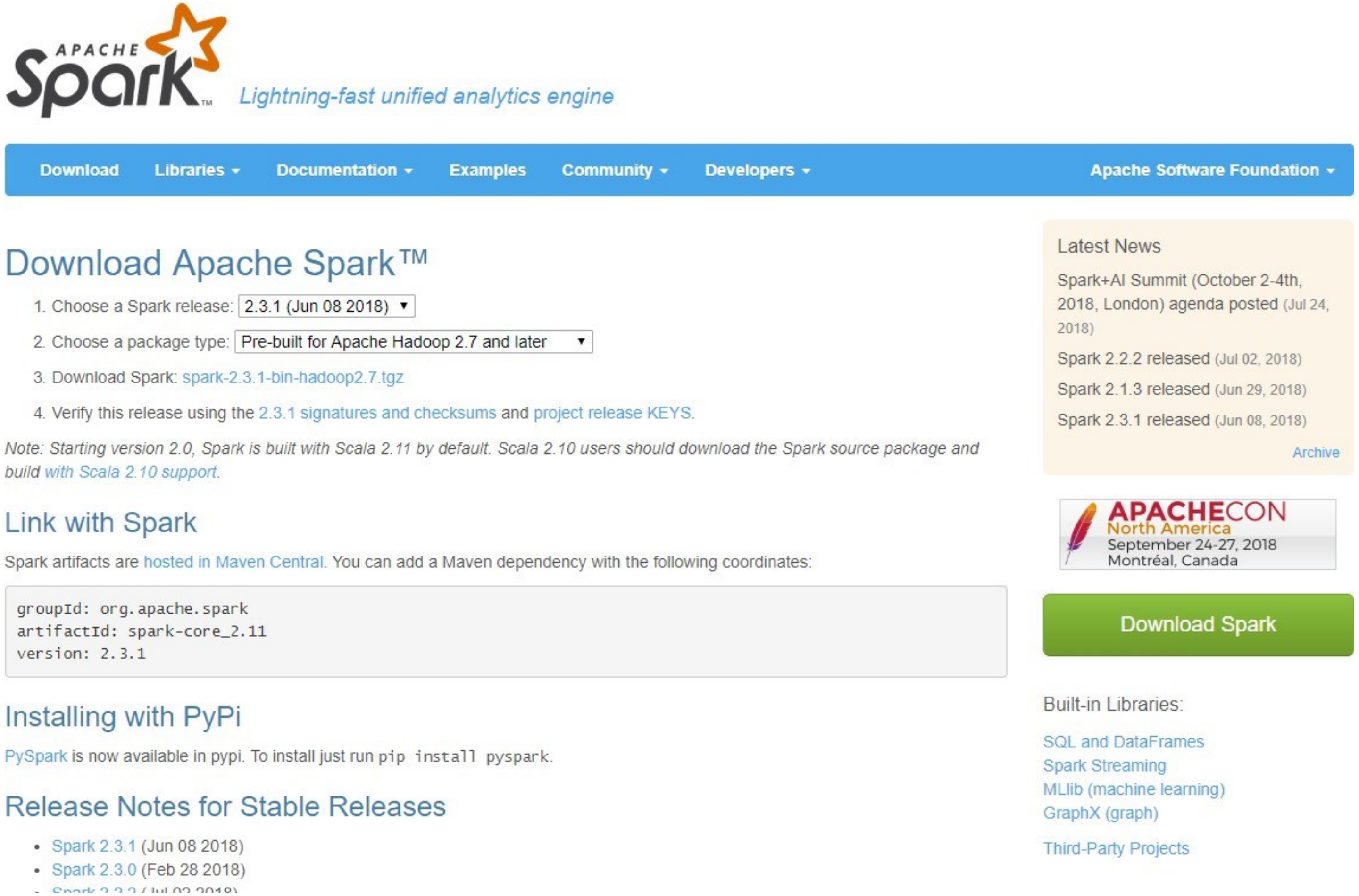
Distributed memory is 10-100X faster than network and disk Data Sharing in Spark

Spark's speed comes from its ability to run computations in memory;

Installation of Spark

Installation of Spark

- Visit http://spark.apache.org/downloads and fetch the latest release of Spark;
- There are a few options on what can be downloaded:
  - The source code;
  - Binaries;



APACHE Spark™
Lightning-fast unified analytics engine
Download
Libraries ▾
Documentation ▾
Examples
Community ▾
Developers ▾
Apache Software Foundation ▾
APACHECON North America September 24-27, 2018 Montréal, Canada
Download Spark

Download Apache Spark™

1. Choose a Spark release: 2.3.1 (Jun 08 2018) ▾
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later ▾
3. Download Spark: spark-2.3.1-bin-hadoop2.7.tgz
4. Verify this release using the 2.3.1 signatures and checksums and project release KEYS.

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.

Link with Spark

Spark artifacts are hosted in Maven Central. You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.11
version: 2.3.1
```

Installing with PyPi

PySpark is now available in pypi. To install just run `pip install pyspark`.

Release Notes for Stable Releases

- Spark 2.3.1 (Jun 08 2018)
- Spark 2.3.0 (Feb 28 2018)
- Spark 2.2.2 (Jul 02 2018)

Latest News

Spark+AI Summit (October 2-4th, 2018, London) agenda posted (Jul 24, 2018)

Spark 2.2.2 released (Jul 02, 2018)

Spark 2.1.3 released (Jun 29, 2018)

Spark 2.3.1 released (Jun 08, 2018)

Archive

Built-in Libraries:

SQL and DataFrames
Spark Streaming
MLlib (machine learning)
GraphX (graph)

Third-Party Projects

- Use Maven for Source Code installation

Installation of Spark

- Working with Cloudera’s VM-s we will do the following:
  - Make sure we have JDK 7+ installed;
  - Make sure we have hadoop-client package installed;

- For example, you could do:

  ```
  $ sudo yum install hadoop-client
  ```

  - yum will either install the package or tell you it is already there;

- Make sure all HDFS and YARN services are installed and running;
- Stop them and start them all;
- Use yum to run installation of Spark, by typing all on one line:

  ```
  $ sudo yum install spark-core spark-master spark-worker spark-history-server spark-python
  ```

  - Answer Yes to all the questions during installation process executed by yum

Installation of Spark

- Set your SPARK_HOME environment variable to /usr/lib/spark
- Add $SPARK_HOME/bin to your PATH environmental variable.
  - …. And you are all set now!

- yum made sure that you have Spark version which matches your Hadoop (Yarn) version;

- Without any further changes we will run Spark locally on this single machine;

Spark Interactive Shells for Python, Scala, R

- Spark comes with interactive shells that enable ad hoc data analysis.

- Spark's shells allow interaction with data distributed on disks or in memory across many machines.

- Spark can load data into memory on the worker nodes, and distributed computations, even ones that process large volumes of data across many machines, can run in a few seconds;

- This makes iterative, ad hoc, and exploratory analysis commonly done in shells a good fit for Spark;

Spark Interactive Shells for Python, Scala, R

- Spark provides both (and only) Python and Scala shells that have been augmented to support access to a cluster of machines.

- Python shell opens with PySpark command;

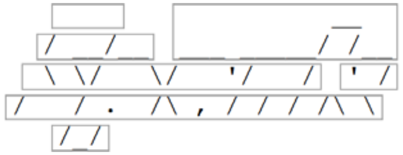
- Scala shell is very similar and opens with spark-shell command;

- You can program Spark from R using package SparkR;

Spark Interactive Shells: PySpark

PySpark

```
[cloudera@localhost conf]$ pyspark
Python 2.8.1 (r231:90122, Aug 23 2025, 23:20:16)
[GCC 4.4.7 20201212 (Red Hat 7.4.1-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information. SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory] Welcome to
```



```
version 1.5.0-cdh5.5.1
```

```
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Spark Interactive Shells: PySpark

- The output is long and annoying. It would have been even longer had we not created log4j.properties file in the directory $SPARK_HOME/conf;
- You create that file by copying provided file log4j.properties.template and by changing line:
  - log4j.rootCategory=INFO, console **to:** log4j.rootCategory=ERROR, console
  - That lowered the logging level so that we see only the ERROR messages, and above;

- Another option is WARN, which is more verbose that ERROR but less than INFO;

- Before we proceed, let’s see what kind of files and how many lines we have in HDFS
- hadoop fs –ls ulysis

```
-rw-r--r-- 1 cloudera 5258688 2015-04-01 14:32 input/4300.txt
```

- hadoop fs -cat ulysis/4300.txt | wc

```
33056 267980 1573079
```

Load data into Spark

Load data into Spark

- **Load Data (RDD) from HDFS {Python commands}**
- We create an RDD when we load some data (i.e. a file) into a shell variable:

```
>>> lines = sc.textFile("ulysis/4300.txt")
>>> lines.count()
    33056
```

- We populated that RDD called “lines” with data in HDFS file “ulysis/4300.txt”;
  - “sc” stands for an implicit SparkContext that allows us to communicate with the execution environment;
  - RDD-s are objects and has many methods,
  - The method count() gives the number of lines in file 4300.txt;

Load data into Spark

- **Load Data (RDD) from Local File (i.e. on your VM desktop) {Python commands}**
- You can find the 4300.txt file in **/home/cloudera** directory and could do the following:

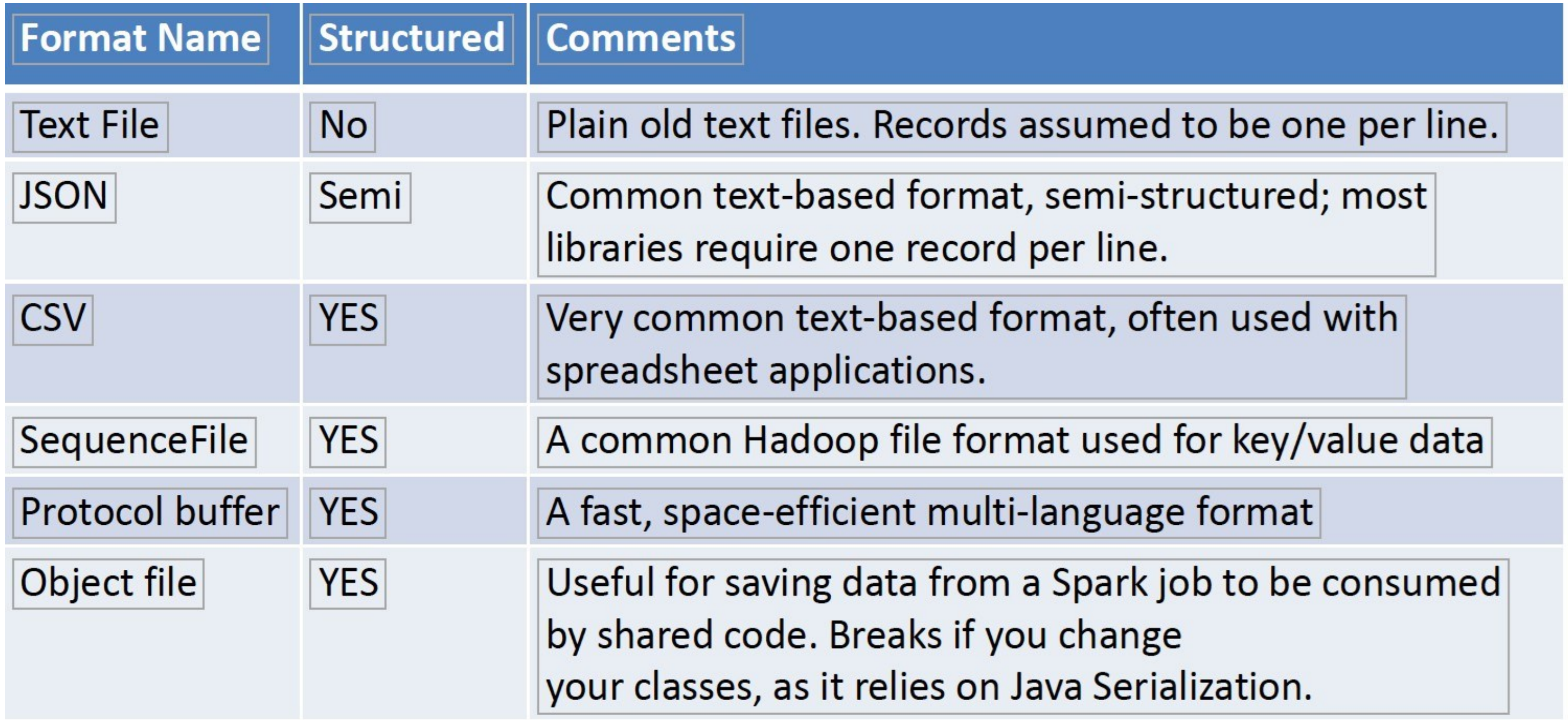
- We create an RDD when we load some data (i.e. a file) into a shell variable:
- `>>> blines = sc.textFile("file:///home/cloudera/4300.txt")`
- `>>> blines.count()`

  ```
  33056
  ```

- You can print the first row using method
- `>>> blines.first()`

Load data into Spark: Source Formats

- Spark makes it very simple to load and save data in a large number of file formats;
- Spark transparently handles compressed formats based on the file extension;
- We can use both Hadoop’s new and old file APIs for keyed (or paired) data;
- We can use those only with key/value data;



Format Name	Structured	Comments
Text File	No	Plain old text files. Records assumed to be one per line.
JSON	Semi	Common text-based format, semi-structured; most libraries require one record per line.
CSV	YES	Very common text-based format, often used with spreadsheet applications.
SequenceFile	YES	A common Hadoop file format used for key/value data
Protocol buffer	YES	A fast, space-efficient multi-language format
Object file	YES	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.