Dataframes & Spark SQL

Loading XML

hadoop fs -cat /data/spark/books.xml

```
<?xml version="1.0"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>
         An in-depth look at creating applications
      ...
      ...
   </book>
   <book id="bk101">
         ...
         ...
   </book>
   ...
   ...
</catalog>
```

Loading XML

We will use: https://github.com/databricks/spark-xml

Start Spark-Shell:

/usr/spark2.0.1/bin/spark-shell ***--packages com.databricks:spark-xml_2.10:0.4.1***

Loading XML

We will use: https://github.com/databricks/spark-xml

Start Spark-Shell:

/usr/spark2.0.1/bin/spark-shell ***--packages com.databricks:spark-xml_2.10:0.4.1***

Load the Data:

val df = spark.read.format("xml").option("rowTag", "book").load("/data/spark/books.xml")

OR

val df = spark.read.format("com.databricks.spark.xml")
.option("rowTag", "book").load("books.xml")

Loading XML

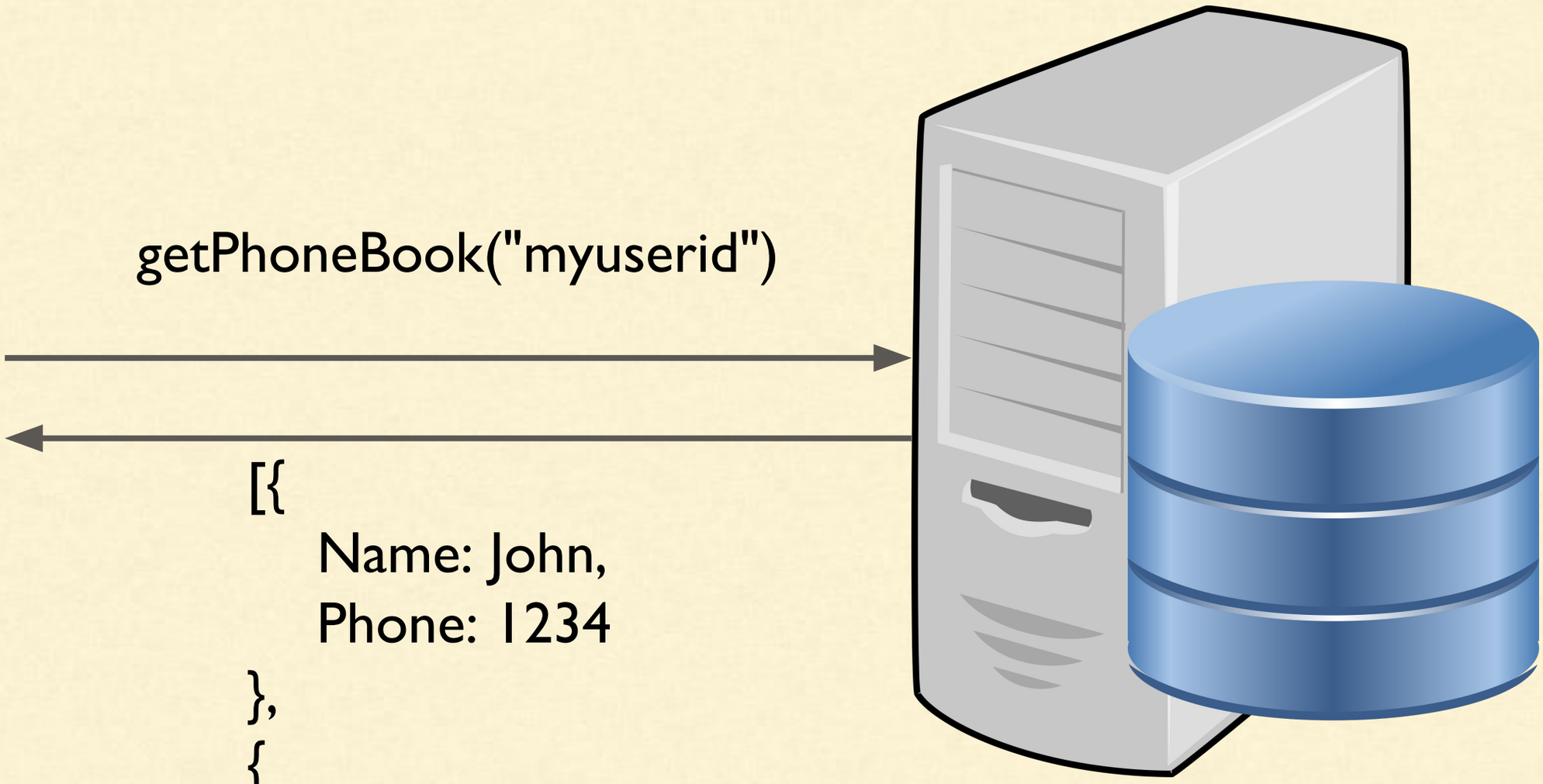
Display Data:

df.show()

```
scala> df.show()
+-----+--------------------+--------------------+--------------+-----+------------+--------------------+
|  _id|              author|         description|         genre|price|publish_date|               title|
+-----+--------------------+--------------------+--------------+-----+------------+--------------------+
|bk101|Gambardella, Matthew|         An in...|      Computer|44.95|  2000-10-01|XML Developer's G...|
|bk102|          Ralls, Kim|A former architec...|       Fantasy| 5.95|  2000-12-16|       Midnight Rain|
|bk103|        Corets, Eva|After the collaps...|       Fantasy| 5.95|  2000-11-17|     Maeve Ascendant|
|bk104|        Corets, Eva|In post-apocalyps...|       Fantasy| 5.95|  2001-03-10|     Oberon's Legacy|
|bk105|        Corets, Eva|The two daughters...|       Fantasy| 5.95|  2001-09-10|  The Sundered Grail|
|bk106|    Randall, Cynthia|When Carla meets ...|       Romance| 4.95|  2000-09-02|          Lover Birds|
|bk107|      Thurman, Paula|A deep sea diver ...|       Romance| 4.95|  2000-11-02|       Splish Splash|
|bk108|       Knorr, Stefan|An anthology of h...|        Horror| 4.95|  2000-12-06|     Creepy Crawlies|
|bk109|        Kress, Peter|After an inadvert...|Science Fiction| 6.95|  2000-11-02|         Paradox Lost|
|bk110|        O'Brien, Tim|Microsoft's .NET ...|      Computer|36.95|  2000-12-09|Microsoft .NET: T...|
|bk111|        O'Brien, Tim|The Microsoft MSX...|      Computer|36.95|  2000-12-01|MSXML3: A Compreh...|
|bk112|         Galos, Mike|Microsoft Visual ...|      Computer|49.95|  2001-04-16|Visual Studio 7: ...|
+-----+--------------------+--------------------+--------------+-----+------------+--------------------+
```

What is RPC - Remote Process Call

getPhoneBook("myuserid")



```
[{
    Name: John,
    Phone: 1234
},
{
    Name: John,
    Phone: 1234
},]
…
```

AVRO

Avro is:

1. A Remote Procedure call
2. Data Serialization Framework
3. Uses JSON for defining data types and protocols
4. Serializes data in a compact binary format
5. Similar to Thrift and Protocol Buffers
6. Doesn't require running a code-generation program

Its primary use is in Apache Hadoop, where it can provide both a serialization format for **persistent** data, and a **wire format** for communication between Hadoop nodes, and from client programs to the Hadoop services.

Apache Spark SQL can access Avro as a data source.[1]

Loading AVRO

We will use: https://github.com/databricks/spark-avro

Start Spark-Shell:

```
/usr/spark2.0.1/bin/spark-shell --packages com.databricks:spark-avro_2.11:3.2.0
```

Loading AVRO

We will use: https://github.com/databricks/spark-avro

Start Spark-Shell:

```
/usr/spark2.0.1/bin/spark-shell --packages com.databricks:spark-avro_2.11:3.2.0
```

Load the Data:

```
val df = spark.read.format("com.databricks.spark.avro")
   .load("/data/spark/episodes.avro")
```

Display Data:

```
df.show()
```

```
+--------------------+-----------------+------+
|               title|        air_date|doctor|
+--------------------+-----------------+------+
|   The Eleventh Hour|    3 April 2010|    11|
|   The Doctor's Wife|     14 May 2011|    11|
```

Data Sources

- Columnar storage format
- Any project in the Hadoop ecosystem
- Regardless of
  - Data processing framework
  - Data model
  - Programming language.

https://parquet.apache.org/

Data Sources

Method1 - Automatically (parquet unless otherwise configured)

```
var df = spark.read.load("/data/spark/users.parquet")
```

Data Sources

Method1 - Automatically (parquet unless otherwise configured)

```
var df = spark.read.load("/data/spark/users.parquet")
df = df.select("name", "favorite_color")
df.write.save("namesAndFavColors.parquet")
```

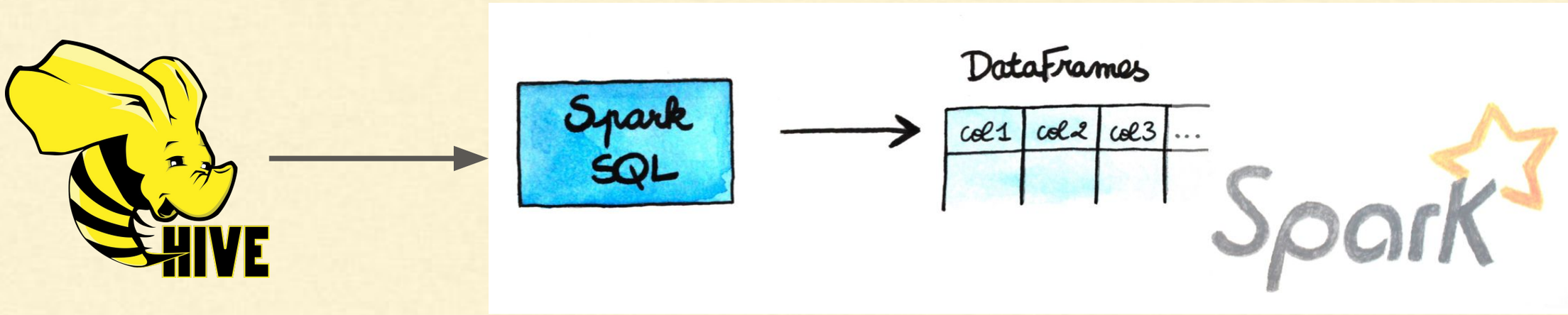
Method2 - Manually Specifying Options

```
df = spark.read.format("json").load("/data/spark/people.json")
df = df.select("name", "age")
df.write.format("parquet").save("namesAndAges.parquet")
```

Method3 - Directly running sql on file

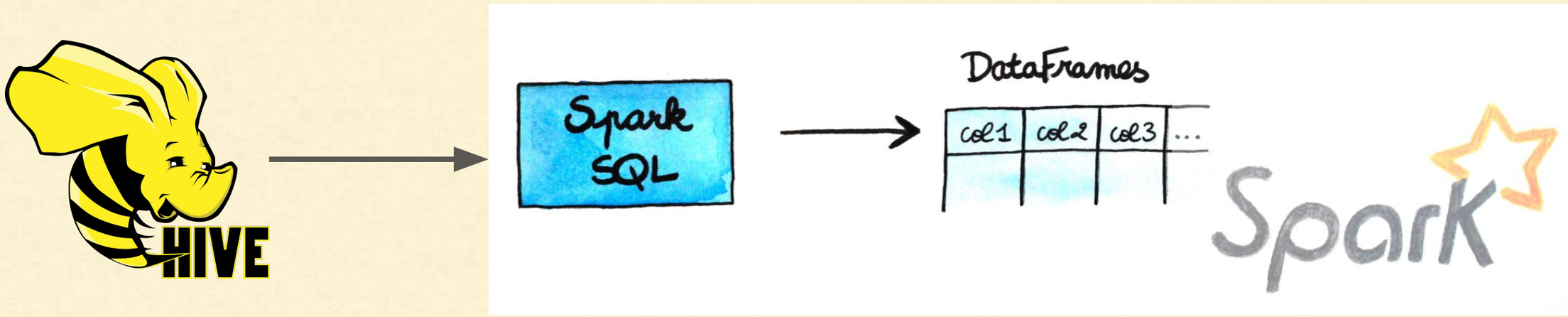
```
val sqlDF = spark.sql("SELECT * FROM parquet.`/data/spark/users.parquet`")
val sqlDF = spark.sql("SELECT * FROM json.`/data/spark/people.json`")
```

Hive Tables



- Spark SQL also supports reading and writing data stored in Apache Hive.
- Since Hive has a large number of dependencies, it is not included in the default Spark assembly.

Hive Tables



- Place your hive-site.xml, core-site.xml and hdfs-site.xmlfile in conf/
- Not required in case of CloudxLab, it already done.

Hive Tables - Example

```
/usr/spark2.0.1/bin/spark-shell

scala> import spark.implicits._
import spark.implicits._

scala> var df = spark.sql("select * from a_student")
scala> df.show()
+---------+-----+-----+------+
|     name|grade|marks|stream|
+---------+-----+-----+------+
| Student1|    A|    1|   CSE|
| Student2|    B|    2|    IT|
| Student3|    A|    3|   ECE|
| Student4|    B|    4|   EEE|
| Student5|    A|    5|  MECH|
| Student6|    B|    6|  CHEM|
```

Hive Tables - Example

```
import java.io.File

val spark = SparkSession
  .builder()
  .appName("Spark Hive Example")
  .enableHiveSupport()
  .getOrCreate()
```

From DBs using JDBC

- Spark SQL also includes a data source that can read data from DBs using JDBC.
- Results are returned as a DataFrame
- Easily be processed in Spark SQL or joined with other data sources

From DBs using JDBC

```
hadoop fs -copyToLocal /data/spark/mysql-connector-java-5.1.36-bin.jar
```

From DBs using JDBC

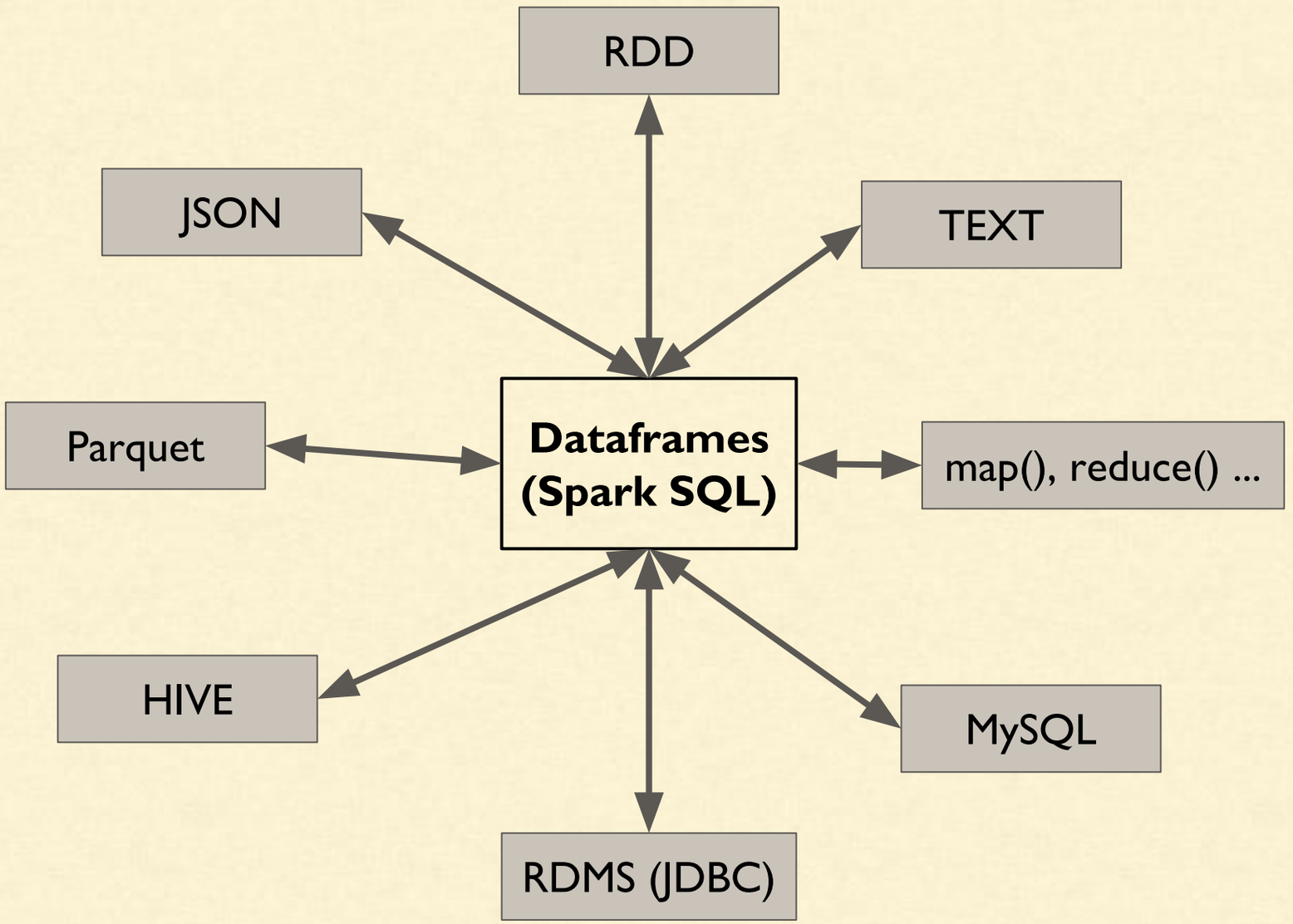
```
hadoop fs -copyToLocal /data/spark/mysql-connector-java-5.1.36-bin.jar

/usr/spark2.0.1/bin/spark-shell --driver-class-path
mysql-connector-java-5.1.36-bin.jar --jars
mysql-connector-java-5.1.36-bin.jar

val jdbcDF = spark.read
.format("jdbc")
.option("url", "jdbc:mysql://ip-172-31-13-154/sqoopex")
.option("dbtable", "widgets")
.option("user", "sqoopuser")
.option("password", "NHkkP876rp")
.load()

jdbcDF.show()
```

Data Frames


RDD
JSON
TEXT
Parquet
Dataframes
(Spark SQL)
map(), reduce() ...
HIVE
MySQL
RDMS (JDBC)

Distributed SQL Engine

- Spark SQL as a distributed query engine
- using its JDBC/ODBC
- or command-line interface.

- Users can run SQL queries on Spark
- without the need to write any code.

Distributed SQL Engine - Setting up

Step 1: Running the Thrift JDBC/ODBC server

The thrift JDBC/ODBC here corresponds to HiveServer. You can start it from the local installation:

```
./sbin/start-thriftserver.sh
```

It starts in the background and writes data to log file. To see the logs use, tail -f command

Distributed SQL Engine - Setting up

Step 2: Connecting

Connect to thrift service using beeline:

./bin/beeline

On the beeline shell:

```
!connect jdbc:hive2://localhost:10000
```

You can further query using the same commands as hive.

Distributed SQL Engine

Demo

Dataframes & Spark SQL

Thank you!