Spark 2.0

RDDs, Dataframe and Datasets

Three Spark APIs

RDDs - Resilient Distributed Datasets

- original classes used for Spark
- still forms the core of Spark

DataFrames - grew up with Spark SQL, 3 years old

Datasets - recent and predominant (ease of use)

RDDs are...

... compile-time type-safe
... lazy
... based on the Scala collections API

similarity can be confusing to Scala programmers

```
def collect[U](f: PartialFunction[T, U])(implicit arg0: ClassTag[U]): RDD[U]
```

Return a new RDD that contains all matching values by applying `f`.

```
def collect(): Array[T]
```

Return an array that contains all of the elements in this RDD.

```
// load file as RDD
JavaRDD<String> nasdaqRDD = sc.textFile("/datasets/companies/companylistNASDAQ.csv");


// parse and clean up the data -> get ticker, market cap and sector fields
JavaRDD<Tuple3<String, Float, String>> parsedRDD = nasdaqRDD.map(ln -> {
    String[] f = REGEX.split(ln, -1);
    for (int i = 0; i < f.length; i++)
        f[i] = f[i].trim().replaceFirst("^\"", "").replaceFirst("\"$", "").replace("$", "").trim();
    return new Tuple3<String, Float, String>(f[TICKER], getCap(f[MARKET_CAP]), f[SECTOR]);
});


// filter sort and list
List<Tuple3<String, Float, String>> l = parsedRDD
        .filter(info -> (!info._1().equals("Symbol") && !info._3().equals("n/a")
                && info._2() > 10f && info._2() < 500f))
        .sortBy(info -> info._1(), true, 1)
        .filter(info -> (info._3().contains("Consumer")))
        .take(10);
```

RDDs

RDDs are type-safe

They are also low-level and focus on “how”, not “what”

- Cannot be optimized by Spark
- *Very* slow on non-JVM languages like Python
- *Too* easy to write an inefficient RDD transformation chain

```
// filter sort and list
List<Tuple3<String, Float, String>> l = parsedRDD
        .filter(info -> (!info._1().equals("Symbol") && !info._3().equals("n/a")
                && info._2() > 10f && info._2() < 500f))
        .sortBy(info -> info._1(), true, 1)
        .filter(info -> (info._3().contains("Consumer")))
        .take(10);
```

The DataFrame API

Provides a higher-level abstraction

- Can call it a domain-specific language (DSL)
- Gives you a query language
- Gives you a way to manipulating data
- And you can use SQL

DataFrame API - same net result as RDD code

Easier to read, same effects.

Starting out with the parsed RDD:

```
val df = parsedRDD.toDF("Ticker", "MarketCap", "Sector")
df.filter(
  ($"Sector".contains("Consumer"))
  &&($"MarketCap" <= 1000)&&($"MarketCap" >= 10)
  &&(!$"Ticker".equals("Symbol"))
  &&(!"$Sector".equals("n/a"))).
sort($"Ticker").
limit(10).
show(10)
```

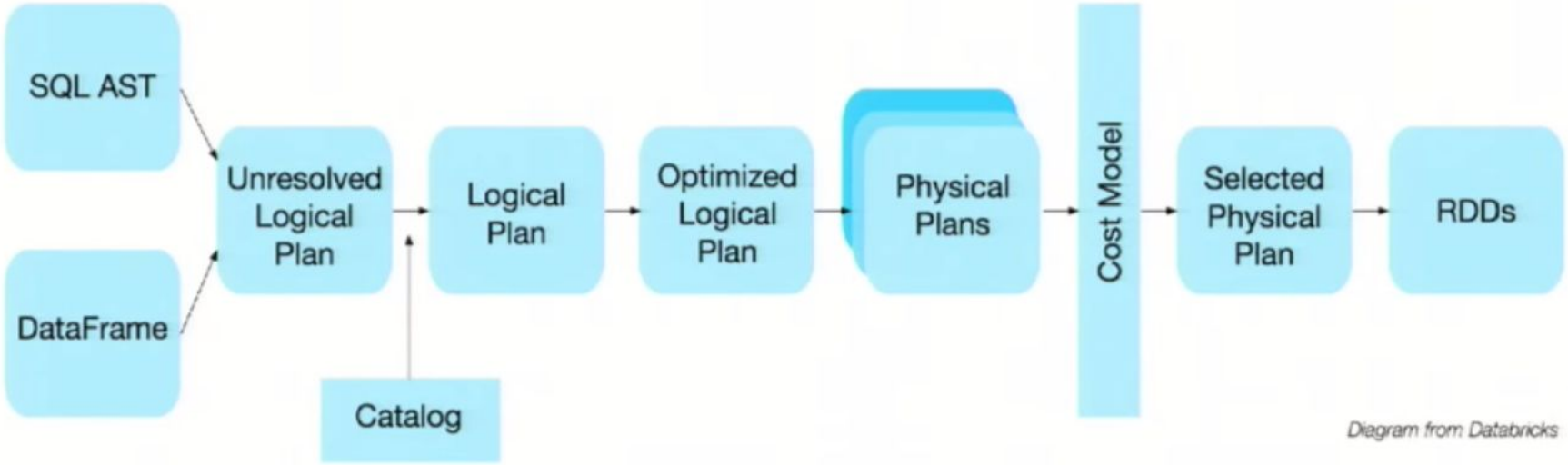
Notebook API: Loading DF as table, using SQL

Create a table based on the dataframe we just created:

df.createOrReplaceTempView("nasdaqTable")

```
%sql CREATE TABLE sectorInfo SELECT
  Sector,
  count(DISTINCT Ticker) as SectorCount,
  SUM(MarketCap) as TotalMarketCap
FROM nasdaqTable
GROUP BY Sector
```

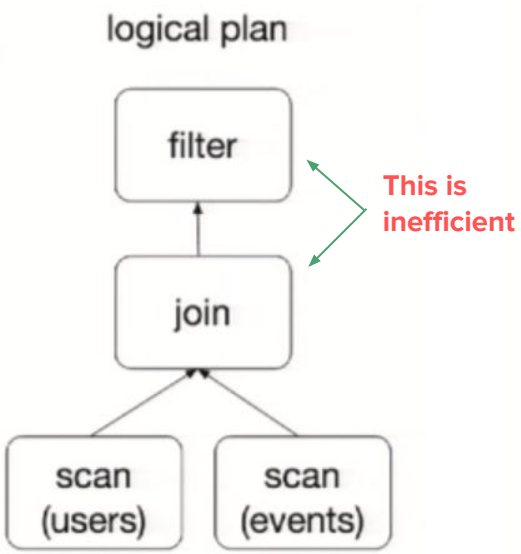
Dataframe execution

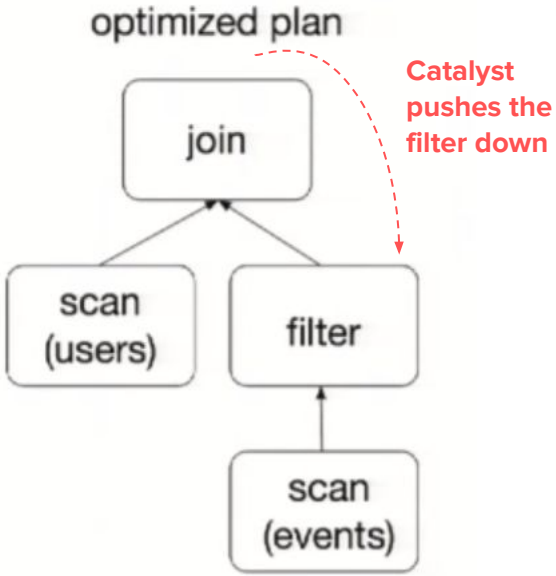


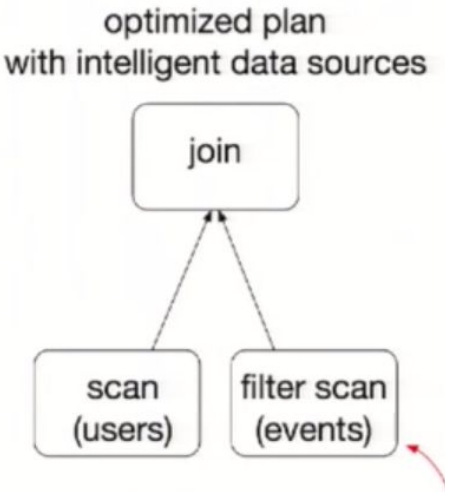
https://github.com/apache/spark/tree/master/sql/catalyst/src/main/scala/org/apache/spark/sql/catalyst

Example: Catalyst optimization

```
users.join(events, users("id") === events("uid"))
     .filter(events("date") > "2015-01-01")
```



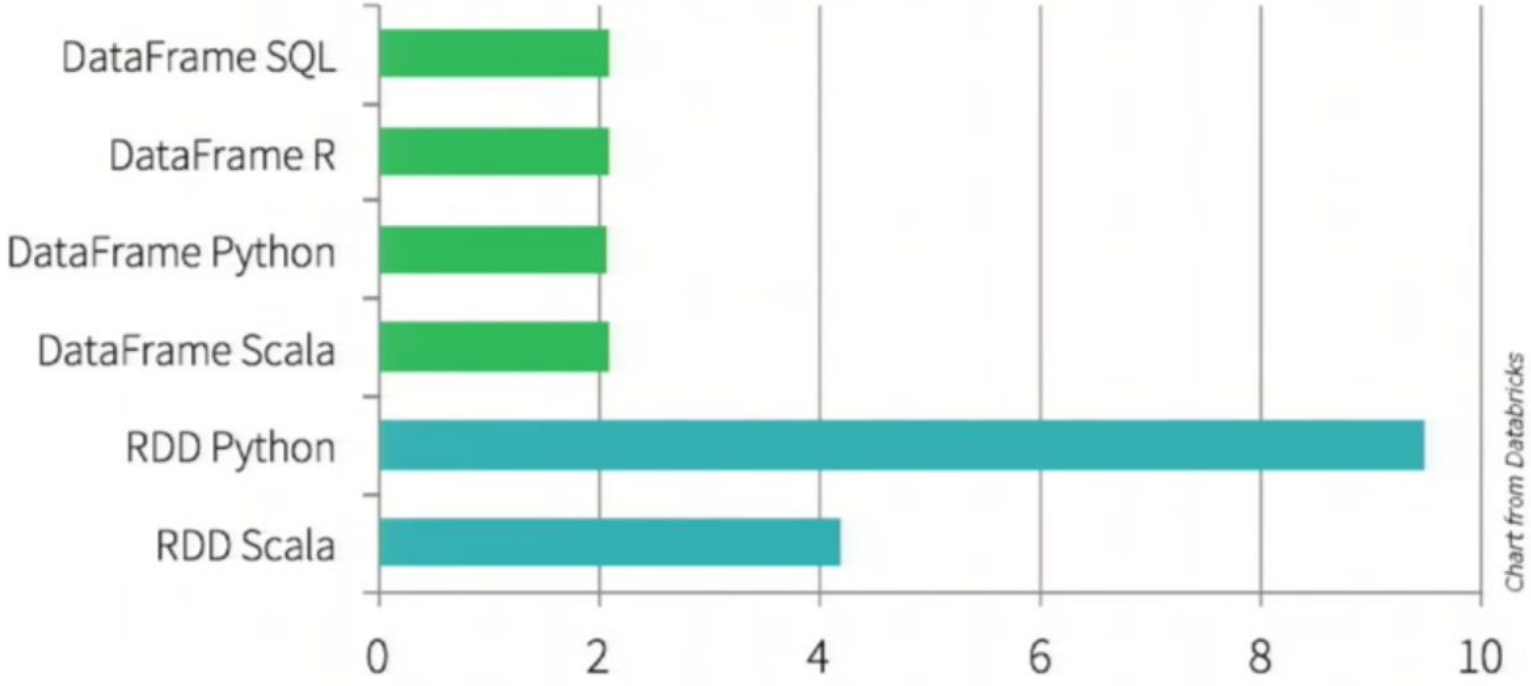




Catalyst pushes the filter into the data source
e.g.: SELECT * FROM events WHERE userid=...

DataFrames are faster

Because of the optimization, they tend to outperform RDDs.



We've lost type safety

Suppose we call collect()

```
scala> :type df.collect()
Array[org.apache.spark.sql.Row]
```

Row is not typesafe - it is defined as

```
trait Row extends Serializable
```

Mapping a Row to something useful is ugly and error-prone

```
df.collect().map { row =>
    val ticker = row(0).asInstanceOf[String]        // Yuck.
    val marketCap = row(1).asInstanceOf[Float]      // Yuck.
```

Type-safety plus optimization -> Datasets!!

Datasets:

- Extend the DataFrame API
- Provide datatypes and functions (lambdas) like RDDs
- Uses Tungsten for fast in-memory encoding
  - Off-heap and unsafe - not JVM objects, not serialized objects on the heap
  - Columnar-based storage
- Expose expressions and fields to the DataFrame query planner, where the optimizer uses them (not available with RDDs)
- Available in Spark 1.6, 2.0 and 2.1 - experimental preview. Will be crucial to Spark MLLib and Streaming in the future.

Dataset with a dataframe API

You can access the DataFrame API

```
// Read a DataFrame from a JSON file
val df = sqlContext.read.json("people.json")
// Convert the data to a domain object.
case class Person(name: String, age: Long)
val ds: Dataset[Person] = df.as[Person]
```

And then convert easily to a Dataset

DataFrame is just a Dataset[Row]

CASE CLASSES? HUH? SEE: HTTP://DOCS.SCALA-LANG.ORG/TUTORIALS/TOUR/CASE-CLASSES.HTML

Datasets can be coded like RDDs

RDDs:

lines is an RDD

```
val lines = sc.textFile("hdfs://path/to/some/ebook.txt")
val words = lines.flatMap(_.split("""\s+""")).filter(_.nonEmpty)
val counts = words.groupBy(_.toLowerCase).map { case (w, all) => (w, all.size) }
```

Datasets:

lines is a Dataset

```
val lines = sqlContext.read.text("hdfs://path/to/some/ebook.txt").as[String]
val words = lines.flatMap(_.split("""\s+""")).filter(_.nonEmpty)
val counts = words.groupBy(_.toLowerCase).count()
```

Datasets can be coded like RDDs

The Dataset version includes count() function

```
// RDD
val counts = words.groupBy(_.toLowerCase).map { case (w, all) => (w, all.size) }
// Dataset
val counts = words.groupBy(_.toLowerCase).count()
```

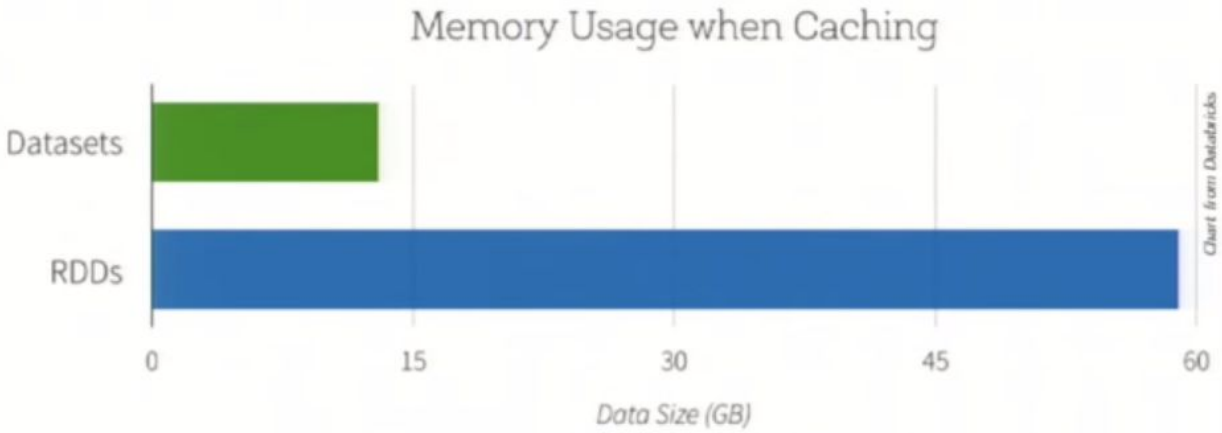
Datasets use less memory

Spark:

- understands the data in Datasets because they're typed.
- uses encoders to translate between types ("domain object") and Tungsten's compact data format.
- generates encoders using runtime code-generation.
- uses generated code to operate directly on the Tungsten managed memory

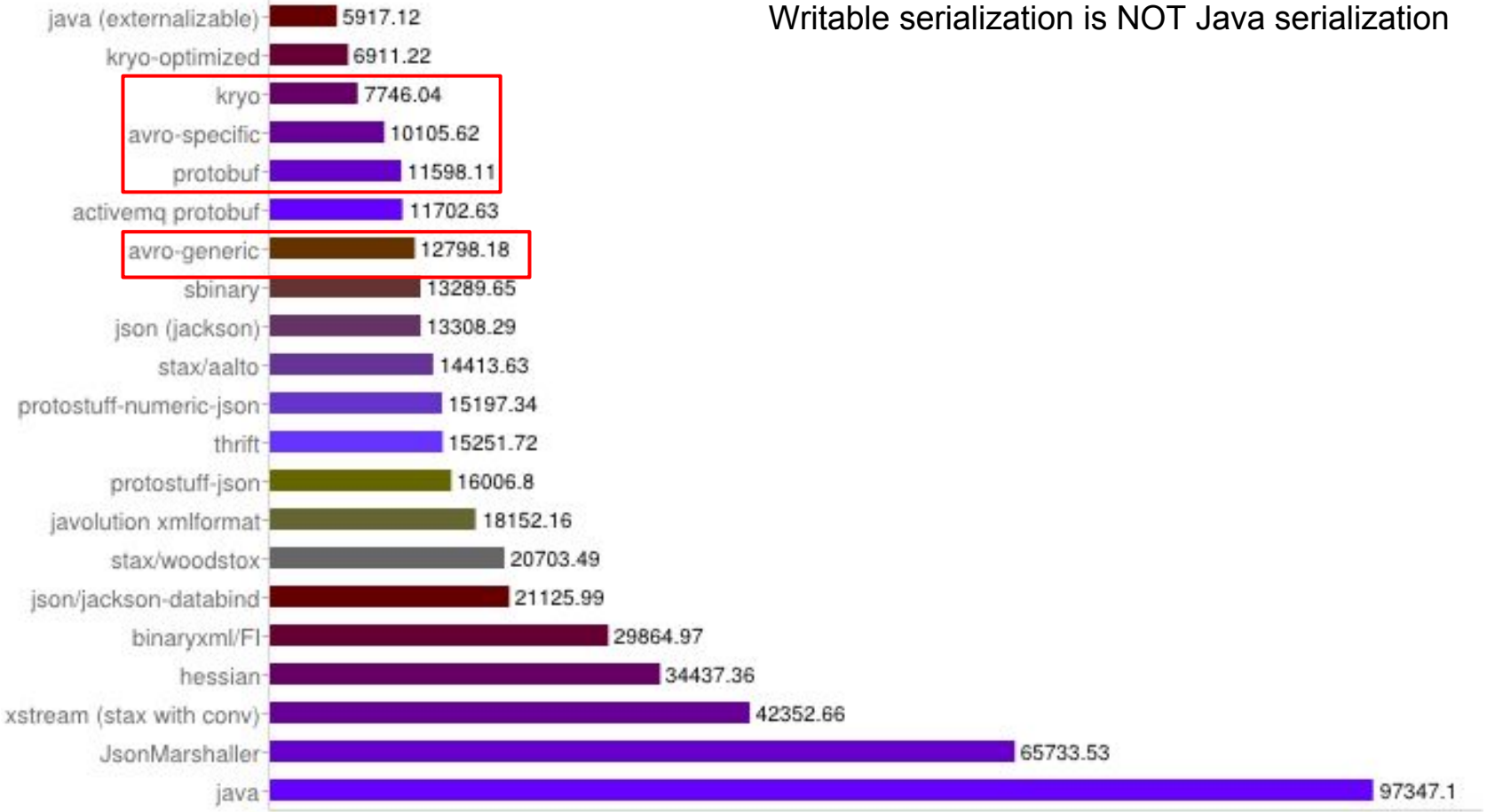
Memory is conserved, because of the compact format. Speed is improved by the code-generation.

Datasets use less memory



Serialization benchmarks

Writable serialization is NOT Java serialization

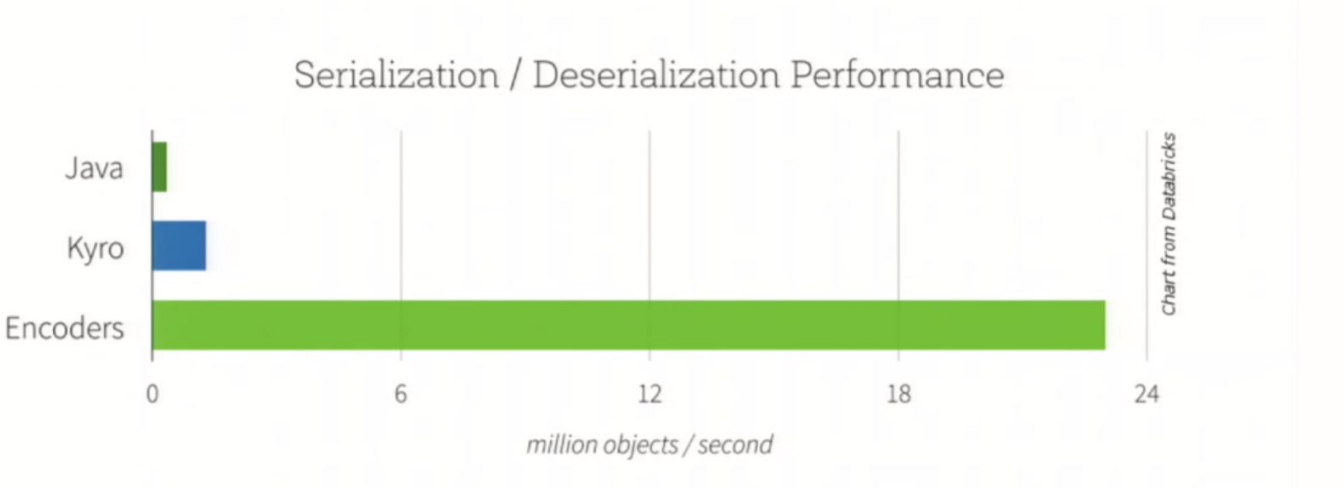


https://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking

Datasets and serialization

Spark serializes data all the time...

- Because of code-generated encoders, Tungsten serialization can be significantly faster than Kryo.
- The resulting serialized data is 2x smaller, which translates into speed (less disk IO, less network traffic)



Dataset Limitations

- They are experimental - the APIs might change in upcoming Spark releases
- The APIs are incomplete. For example, Datasets lack common aggregators like sum(), avg() and sortBy().
- Spark Apps on Eclipse/Maven is (apparently) a despised step-child
  - Use Databricks Notebooks
  - Use IntelliJ and sbt
  - Use Cloudera's distributions
    - though your VM uses Spark 1.6 -- even Cloudera is afraid of Spark 2.0
- It's really a pain to Google "datasets"...
  - almost as bad as "R"

Reading

RDDs, DataFrames and Datasets Guide
http://spark.apache.org/docs/latest/sql-programming-guide.html

Introducing Spark Datasets (Databricks blog post):
https://databricks.com/blog/2016/01/04/introducing-spark-datasets.html

More of Brian Clapper's talks - amazing
http://scala-phase.org/

Spark Notebooks

Pick Up a Notebook or Build your own

- Get the community edition from Databricks (***recommended***)
- Use a generated notebook from `spark-notebook.io`
  - Comes in many flavors
  - Can ask for various releases and formats
- But, if you ask for the latest stable releases:

  Spark 2.0, Scala 2.11, Hadoop 2.7.2

  Requested build available in 2 days!
- There's also Jupiter and Zeppelin

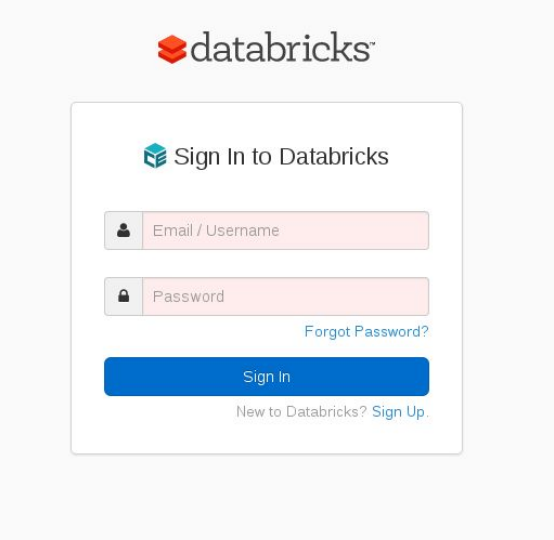
Spark 2016 - Notebook demo 15 minutes

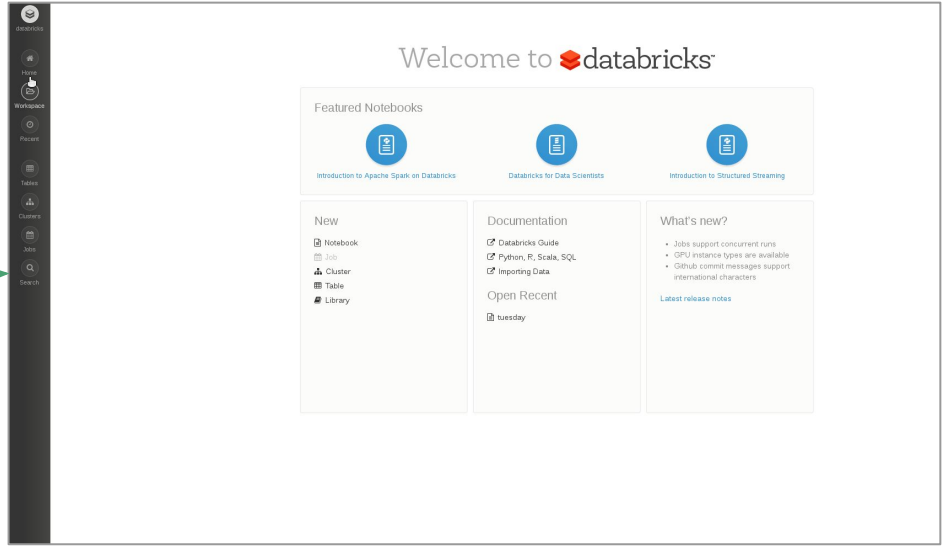
Link to summit demo

OK, let's try this

Login to your notebook

https://community.cloud.databricks.com/login.html



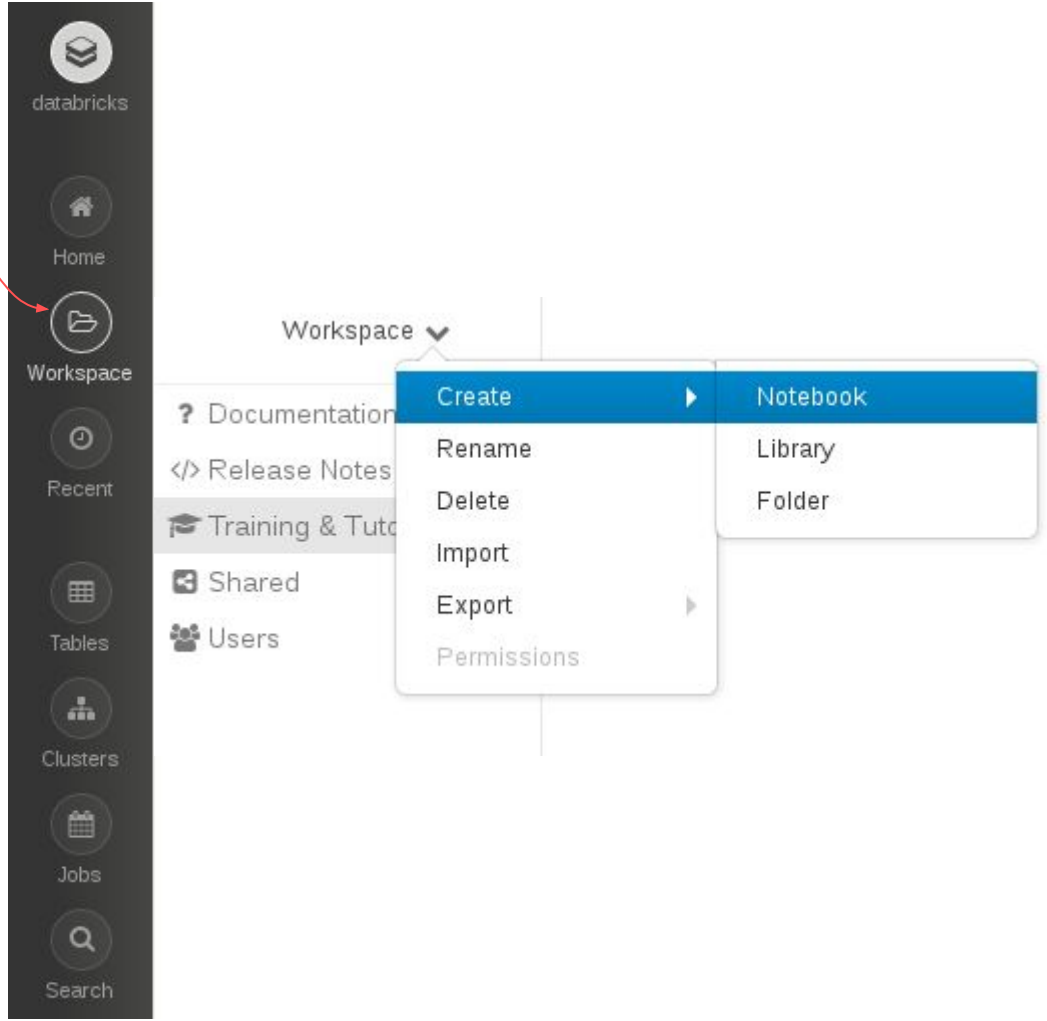


Get a place to work

first, click on workspace

next, create a Notebook

finally, define it



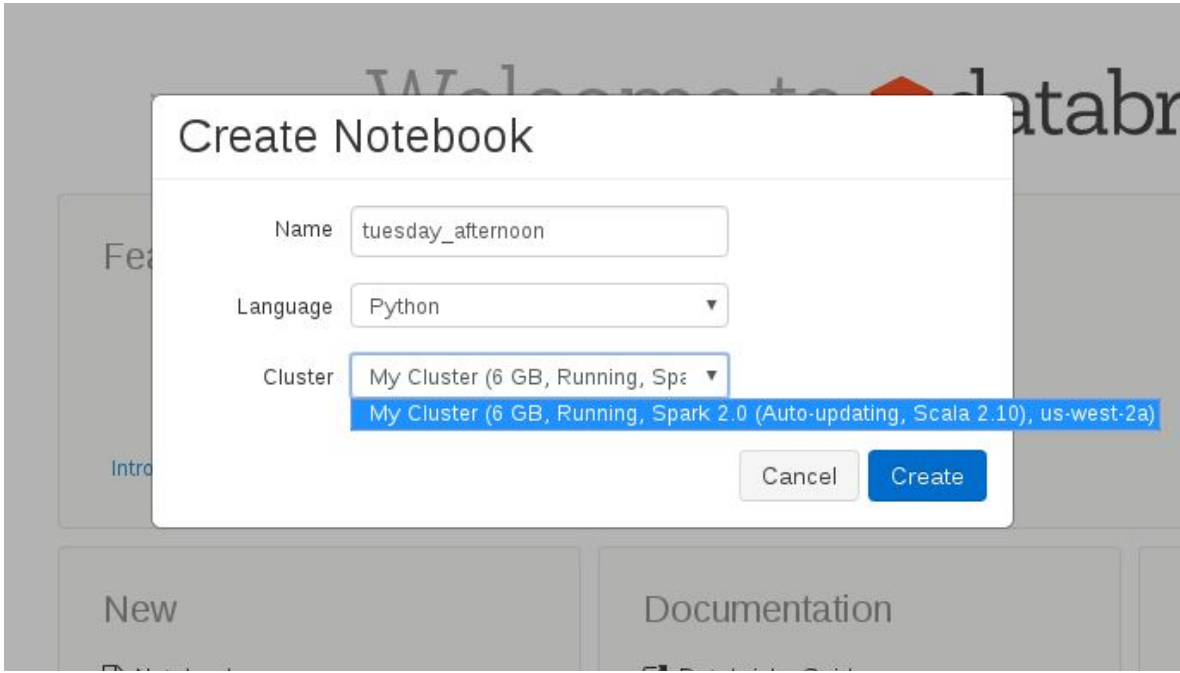
Get a place to work

first, click on workspace

next, create a Notebook

finally, define it

- name: <whatever>
- language: Python, Scala or SQL
- cluster -> only one choice
    - something familiar?

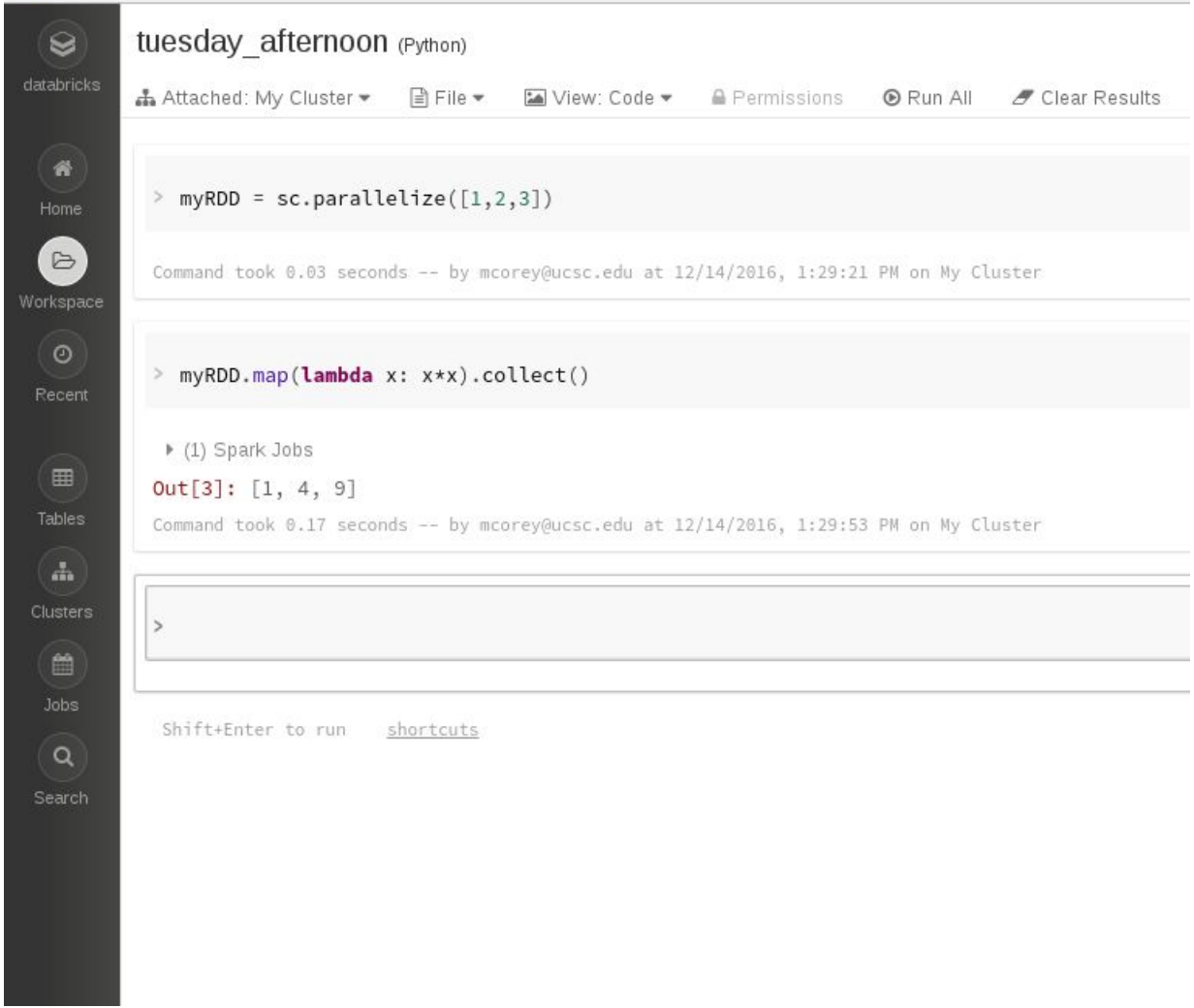


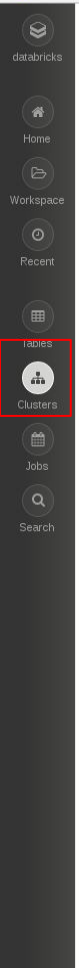
Run some commands

- Create test data
- Run a map function

Hmm...

it would be nice to have some data.





Clusters

Active Clusters

+ Create Cluster

	Name	Memory	Type	State	Nodes	Spark	Libraries	Notebooks	Default Cluster	Actions
●	My Cluster	6 GB	Community Optimized, Spark 2.0 (Auto-updating, Scala 2.10)	Running	1 On-demand	Spark UI Logs	--	1 notebook	Default	

Terminated Clusters

Name	Memory	Type	State	Nodes	Spark	Libraries	Notebooks	Default Cluster	Actions

Add file to DBFS

- Goto Tables
- Select “Create Table”
- Drag-and-drop or click to browse to local datafiles.

databricks

Home

Workspace

Recent

Tables

Clusters

Jobs

Search

Create Table

Data Source: File

File(s)

companylistl

0.4 MB

Remove file

Uploaded to DBFS

/FileStore/tables/xc6hafws1481751854579/companylistNASDAQ.csv

Preview Table

Table Details

Table name: nasdaqInfo

File type: CSV

Column Delimiter: ,

First row is header

Create Table

Previewing table

Symbol	Name	LastSale	MarketCap	IPOyear	Sector	industry	Summary Quote
STRING	STRING	STRING	STRING	STRING	STRING	STRING	STRING
TFSC	1347 Capital Corp.	9.75	$57.99M	2014	Finance	Business Services	http://www.nasdaq.com/s
TFSCR	1347 Capital Corp.	0.38	n/a	2014	Finance	Business Services	http://www.nasdaq.com/s
TFSCU	1347 Capital Corp.	10.27	$42.93M	2014	Finance	Business Services	http://www.nasdaq.com/s
TFSCW	1347 Capital Corp.	0.235	n/a	2014	Finance	Business Services	http://www.nasdaq.com/s
PIH	1347 Property Insurance Holdings, Inc.	8.11	$51.56M	2014	Finance	Property-Casualty Insurers	http://www.nasdaq.com/s
FLWS	1-800 FLOWERS.COM, Inc.	10.44	$681.34M	1999	Consumer Services	Other Specialty Stores	http://www.nasdaq.com/s
FCTY	1st Century Bancshares, Inc	7.05	$71.67M	n/a	Finance	Major Banks	http://www.nasdaq.com/s
FCCY	1st Constitution Bancorp (NJ)	11.5	$86.3M	n/a	Finance	Savings Institutions	http://www.nasdaq.com/s
SRCE	1st Source Corporation	33.99	$810.77M	n/a	Finance	Major Banks	http://www.nasdaq.com/s
VNET	21Vianet Group, Inc.	20.33	$1.34B	2011	Technology	Computer Software: Programming, Data Processing	http://www.nasdaq.com/s
TWOU	2U, Inc.	30.29	$1.25B	2014	Technology	Computer Software: Prepackaged Software	http://www.nasdaq.com/s
DGLD	3X Inverse Gold ETN Velocityshares	75.931	$10.33M	n/a	Finance	Investment Bankers/Brokers/Service	http://www.nasdaq.com/s
DSLV	3X Inverse Silver ETN Velocityshares	60.09	$34.07M	n/a	Finance	Investment Bankers/Brokers/Service	http://www.nasdaq.com/s
UGLD	3X Long Gold ETN Velocityshares	10.25	$10.1M	n/a	Finance	Investment	.com/s

databricks

Home

Workspace

Recent

Tables

Clusters

Jobs

Search

Table: nasdaqInfo

nasdaqInfo

Refresh

Schema:

col_name	data_type	comment
Symbol	string	null
Name	string	null
LastSale	string	null
MarketCap	string	null
IPOyear	string	null
Sector	string	null
industry	string	null
Summary Quote	string	null
_c8	string	null

Sample Data:

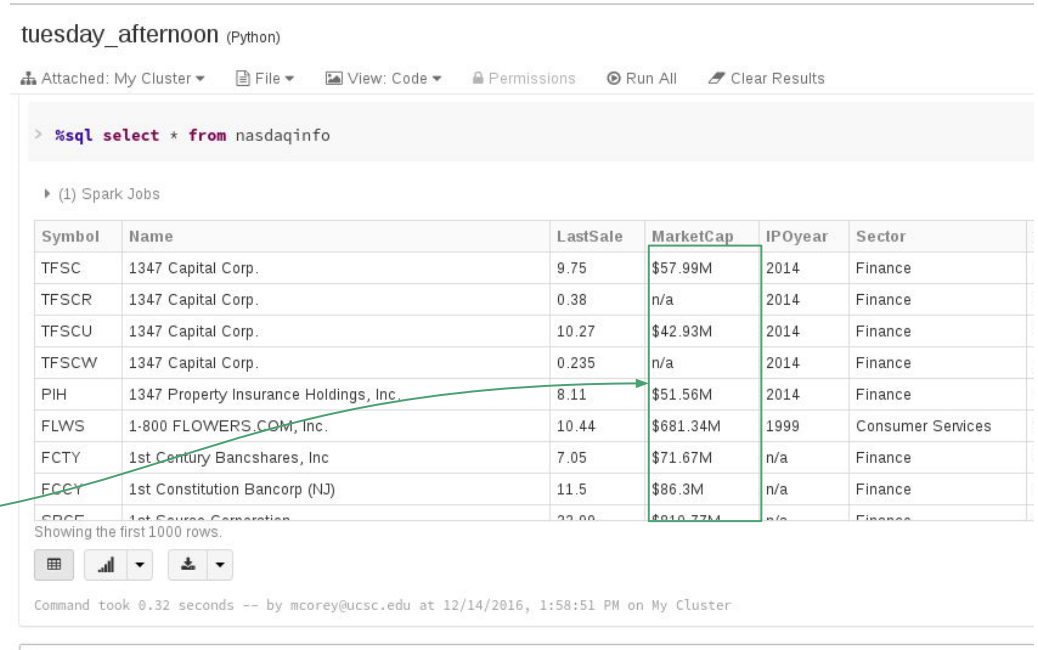
Symbol	Name	LastSale	MarketCap	IPOyear	Sector	industry	Summary Quote	_c8
TFSC	1347 Capital Corp.	9.75	$57.99M	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfsc	null
TFSCR	1347 Capital Corp.	0.38	n/a	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscr	null
TFSCU	1347 Capital Corp.	10.27	$42.93M	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscu	null
TFSCW	1347 Capital Corp.	0.235	n/a	2014	Finance	Business Services	http://www.nasdaq.com/symbol/tfscw	null
PIH	1347 Property Insurance Holdings, Inc.	8.11	$51.56M	2014	Finance	Property-Casualty Insurers	http://www.nasdaq.com/symbol/pih	null
FLWS	1-800 FLOWERS.COM, Inc.	10.44	$681.34M	1999	Consumer Services	Other Specialty Stores	http://www.nasdaq.com/symbol/flws	null
FCTY	1st Century Bancshares, Inc	7.05	$71.67M	n/a	Finance	Major Banks	http://www.nasdaq.com/symbol/fcty	null
FCCY	1st Constitution Bancorp (NJ)	11.5	$86.3M	n/a	Finance	Savings Institutions	http://www.nasdaq.com/symbol/fccy	null
SRCE	1st Source Corporation	33.99	$810.77M	n/a	Finance	Major Banks	http://www.nasdaq.com/symbol/srce	null
VNET	21Vianet Group, Inc.	20.33	$1.34B	2011	Technology	Computer Software: Programming, Data Processing	http://www.nasdaq.com/symbol/vnet	null
TWOU	2U, Inc.	30.29	$1.25B	2014	Technology	Computer Software: Prepackaged Software	http://www.nasdaq.com/symbol/twou	null
DGLD	3X Inverse Gold ETN Velocityshares	75.931	$10.33M	n/a	Finance	Investment Bankers/Brokers/Service	http://www.nasdaq.com/symbol/dgld	null
DSLV	3X Inverse Silver ETN Velocityshares	60.09	$34.07M	n/a	Finance	Investment Bankers/Brokers/Service	http://www.nasdaq.com/symbol/dslv	null
UGLD	3X Long Gold ETN Velocityshares	10.25	$10.1M	n/a	Finance	Investment Bankers/Brokers/Service	http://www.nasdaq.com/symbol/ugld	null

Run SQL on Spark

- preface with **%sql**
- write SQL

```
select * from nasdaqinfo
```

Yuck.
Unusable data



Symbol	Name	LastSale	MarketCap	IPOyear	Sector
TFSC	1347 Capital Corp.	9.75	$57.99M	2014	Finance
TFSCR	1347 Capital Corp.	0.38	n/a	2014	Finance
TFSCU	1347 Capital Corp.	10.27	$42.93M	2014	Finance
TFSCW	1347 Capital Corp.	0.235	n/a	2014	Finance
PIH	1347 Property Insurance Holdings, Inc.	8.11	$51.56M	2014	Finance
FLWS	1-800 FLOWERS.COM, Inc.	10.44	$681.34M	1999	Consumer Services
FCTY	1st Century Bancshares, Inc	7.05	$71.67M	n/a	Finance
FCCY	1st Constitution Bancorp (NJ)	11.5	$86.3M	n/a	Finance

Load and clean up the data (1)

```
> %scala // use scala and read the file into an RDD
  val nasdaqRDD = sc.textFile("/FileStore/tables/ovsk64s41481754163954/companylistNASDAQ.csv")
```

```
nasdaqRDD: org.apache.spark.rdd.RDD[String] = /FileStore/tables/ovsk64s41481754163954/companylistNASDAQ.csv MapPartitionsRDD[13]
```

Command took 0.34 seconds -- by mcorey@ucsc.edu at 12/15/2016, 8:47:40 AM on My Cluster

```
> %scala  // split each line into an array of strings, remove extraneous quotes and dollar signs
  val nasdaqFields = nasdaqRDD.map( ln=>
      ln.split(",(?=([^\"]*\"[^\"]*\")*[^\"]*$)" )
          .map( s=>s.trim().replaceFirst("^\"", "").replaceFirst("\"$","").replace("$", "").trim() ) )
```

```
nasdaqFields: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[219] at map at <console>:34
```

Command took 0.14 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:16:23 AM on My Cluster

```
> %scala  // filter on market caps (array index 3) - only include caps in the millions
  val filteredFields = nasdaqFields.filter(s=>s(3).endsWith("M"))
```

```
filteredFields: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[221] at filter at <console>:36
```

Command took 0.09 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:16:25 AM on My Cluster

Clean up the data some more

```
> %scala  // create an array with just ticker, market cap and sector name
  val goodFields = filteredFields.map(s=>(s(0), s(3).replace("M",""), s(5)))
```

```
goodFields: org.apache.spark.rdd.RDD[(String, String, String)] = MapPartitionsRDD[494] at map at <console>:39
```

Command took 0.10 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:37:26 AM on My Cluster

```
> %scala  // filter out market caps that are not well-formed "double" numbers
  val betterFields = goodFields.filter(s=> (
          try{s._2.toDouble}catch{case e:Exception=>false})!=false)
```

```
betterFields: org.apache.spark.rdd.RDD[(String, String, String)] = MapPartitionsRDD[496] at filter at <console>:41
```

Command took 0.10 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:37:28 AM on My Cluster

Process the new data as a table using SQL

```
> %scala  //
  val betterDF = betterFields.toDF()
  betterDF.cache()
```

```
betterDF: org.apache.spark.sql.DataFrame = [_1: string, _2: string ... 1 more field]
res15: betterDF.type = [_1: string, _2: string ... 1 more field]
```

Command took 0.39 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:51:33 AM on My Cluster

```
> %scala // create a table we can use in the notebook.  Note - we're using `createOrReplaceTempView` instead of `registerTempTable`
  betterDF.createOrReplaceTempView("betterTable")
```

Command took 0.06 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:51:42 AM on My Cluster

```
> %sql CREATE TABLE sectorInfo SELECT _3 as Sector, count(DISTINCT _1) as SectorCount, SUM(_2) as TotalCap
  FROM betterTable
  GROUP BY _3
```

▸ (2) Spark Jobs

OK

Command took 17.78 seconds -- by mcorey@ucsc.edu at 12/15/2016, 9:40:00 AM on My Cluster

View job info

```
%scala // create a table we can use in the notebook.  Note - we're using `createOrReplaceTempView` instead of
nasdaqDF.createOrReplaceTempView("nasdaqTable")
```

Command took 0.07 seconds -- by mcorey@ucsc.edu at 12/15/2016, 11:43:56 AM on My Cluster

```
%sql DROP TABLE IF EXISTS sectorInfo
```

▸ (1) Spark Jobs

OK

Command took 7.82 seconds -- by mcorey@ucsc.edu at 12/15/2016, 11:45:23 AM on My Cluster

```
%sql CREATE TABLE sectorInfo SELECT Sector, count(DISTINCT Ticker) as SectorCount, SUM(MarketCap) as TotalMar
FROM nasdaqTable
GROUP BY Sector
```

▾ (2) Spark Jobs

▾ Job 37 View (Stages: 3/3)

Stage 61: 2/2

Stage 62: 200/200

Stage 63: 200/200

▾ Job 38 View (Stages: 1/1)

Stage 64: 1/1

OK

Command took 15.55 seconds -- by mcorey@ucsc.edu at 12/15/2016, 11:45:35 AM on My Cluster

```
%sql select * from sectorInfo
```

Jobs | Stages | Storage | Environment | Executors | SQL | JDBC/ODBC Server

Details for Stage 61 (Attempt 0)

Total Time Across All Tasks: 1 s
Locality Level Summary: Process local: 2
Shuffle Write: 93.5 KB / 2050

▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.4 s	0.4 s	1.0 s	1.0 s	1.0 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Write Size / Records	46.3 KB / 1021	46.3 KB / 1021	47.2 KB / 1029	47.2 KB / 1029	47.2 KB / 1029

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Shuffle Write Size / Records
driver	10.172.169.21:34506	1 s	2	0	2	93.5 KB / 2050

Tasks (2)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Write Time	Shuffle Write Size / Records	Errors
0	2863	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/12/15 19:45:35	1.0 s		22 ms	46.3 KB / 1021	
1	2864	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/12/15 19:45:35	0.4 s		22 ms	47.2 KB / 1029	

Table: sectorinfo

sectorinfo

Refresh

Schema:

col_name	data_type	comment
Sector	string	null
SectorCount	bigint	null
TotalCap	double	null

Sample Data:

Sector	SectorCount	TotalCap
Consumer Non-Durables	72	16682.63
Consumer Services	210	62329.83000000002
Consumer Durables	63	13257.360000000002
Basic Industries	60	14051.079999999998
Public Utilities	48	12864.500000000004
Transportation	35	13458.33
Miscellaneous	66	15244.519999999999
Capital Goods	141	35391.619999999995
Health Care	418	112288.84999999999
Technology	268	70953.47999999998
Finance	421	100120.77
Energy	56	11177.280000000002
n/a	192	41151.080000000016

To view this Spark Notebook

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/7343321560978772/2054956948486888/5054810217291396/latest.html

Reference slides

Comments on IO formats, footnotes

The changing face of IO in Spark

Data input - new “adapters” for loading data

In Spark SQL:

- **default (parquet):** sqlContext.read.load("examples/src/resources/users.parquet")
- **json:** sqlContext.read.format("json").load("examples/src/resources/people.json")
- **csv:** sqlContext.read.format("com.databricks.spark.csv").load("cars.csv")

-- or you can use the file directly in a SQL query --

```
spark.sql("SELECT * FROM parquet.`examples/src/main/resources/users.parquet`")
```

Old school: IO Formats on Spark

Data input - SparkContext methods wrap Hadoop IO

TextInputFormat - **`sc.textFile(pathStr)`**

SequenceFileInputFormat

- **`sc.SequenceFile(pathStr,keyClass, valueClass)`**
- keyClass and valueClass are Writable

WholeFileInputFormat

- **`sc.wholeFile(directory)`**
- lots of little files (e.g. json files)

All input formats available through Hadoop can be accessed using the "newAPIHadoopFile".

Spark wrappers for Hadoop IO - InputFormats

To use TextInputFormat:

```
scala> val lines = sc.textFile(pathStr)
```

To use SequenceFileInputFormat:

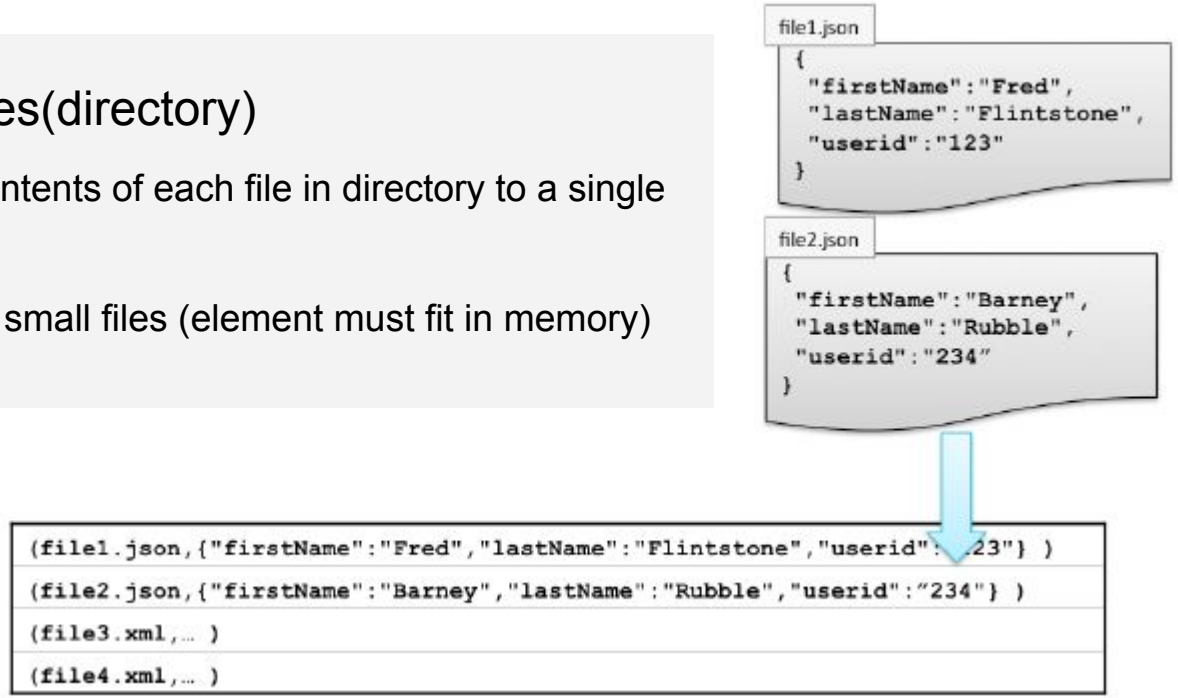
```
scala> val records = sc.sequenceFile(pathStr, key.class, value.class)
```

EXAMPLE: READING A SEQUENCE FILE FROM THE SPARK-SHELL

```
scala> import org.apache.hadoop.io.IntWritable
scala> import org.apache.hadoop.io.Text
scala> val rdd = sc.sequenceFile("mySequenceFile"), IntWritable.class, Text.class)
```

using wholeTextFiles(1): Handling small files

- sc.wholeTextFiles(directory)
  - Maps entire contents of each file in directory to a single RDD element
  - Works only for small files (element must fit in memory)



Using wholeTextFiles(2): Reading a JSON file

python

```
> import json
> myrdd1 = sc.wholeTextFiles(mydir)
> myrdd2 = myrdd1
  .map(lambda (fname,s): json.loads(s))
> for record in myrdd2.take(2):
>     print record["firstName"]
```

scala

```
> import scala.util.parsing.json.JSON
> val myrdd1 = sc.wholeTextFiles(mydir)
> val myrdd2 = myrdd1
  .map(pair => JSON.parseFull(pair._2).get.
            asInstanceOf[Map[String,String]])
> for (record <- myrdd2.take(2))
    println(record.getOrElse("firstName",null))
```

Output:

```
Fred
Barney
```

Input for other Hadoop Input formats (1)

KeyValueText files

`newAPIHadoopFile(path, InputFormat class, key.class, value.class, [conf])`

```
> var file = sc.newAPIHadoopFile(“/home/cloudera/datasets/key_values”,
    'org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat',
    'org.apache.hadoop.io.Text',
    'org.apache.hadoop.io.IntWritable',
    conf = {'mapreduce.input.keyvaluelinerecordreader.key.value.separator' : ','}
    )
```

Input for other Hadoop Input formats (2)

Avro files

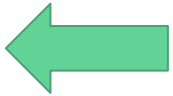
```
newAPIHadoopFile(path, InputFormat class, key.class, value.class, [conf])
```

```
> import org.apache.hadoop.conf.Configuration
> var file = sc.newAPIHadoopFile(‘myLocalAvroFile’,
    'org.apache.hadoop.mapreduce.lib.input.AvroKeyInputFormat',
    'org.apache.avro.mapred.AvroKey',
    'org.apache.hadoop.io.NullWritable',
    new Configuration())
```

Footnotes

Off-heap memory

- Off-heap memory is nothing special.
- On the JVM, the thread stacks, application code, NIO buffers are all off-heap.
- Off-heap memory is not managed -- there’s no garbage collection
  - C/C++ doesn’t have any managed-memory
  - If you don’t need garbage-collection, or you do your own, you don’t need memory from the heap



The DataFrame API in Spark 2.0 - nitty gritty

To use the DataFrame API, we use/create a SparkSession

- do not need to create SparkConf, SparkContext or SQLContext
- encapsulated within the SparkSession.

Example:

```
import org.apache.spark.sql.SparkSession;

SparkSession spark = SparkSession
                .builder()
                .master("local")
                .appName("Java Spark SQL wikipages example")
                .getOrCreate();
```