MAKING BIG DATA COME ALIVE

Zero to pySpark in 50 minutes

Intermountain Big Data Conference

11/21/2015

Marissa Saunders

Thanks to our Sponsors!

THINKBIG
A TERADATA COMPANY

Yearly Partners

HealthEquity
Building Health Savings

Goldman Sachs

AMPIAN

stg SOFTWARE TECHNOLOGY GROUP

THE UNIVERSITY OF UTAH

MaritzCX

JOURNEYTEAM

Gold Sponsors

DELL Software

PASS

idera

Hortonworks

sorenson MEDIA

SSG
SQL Solutions Group

EMC² Isilon

Microsoft

Micron

actifio

Agenda

- Intro to Spark
- Practicalities of data science on pySpark
  - Data structures
  - Data profiling and stats
  - Machine learning
- Big data: practicalities and abstractions
  - Whys and whens of big data
  - To cache or not to cache
  - Which library goes where?
- Take away points

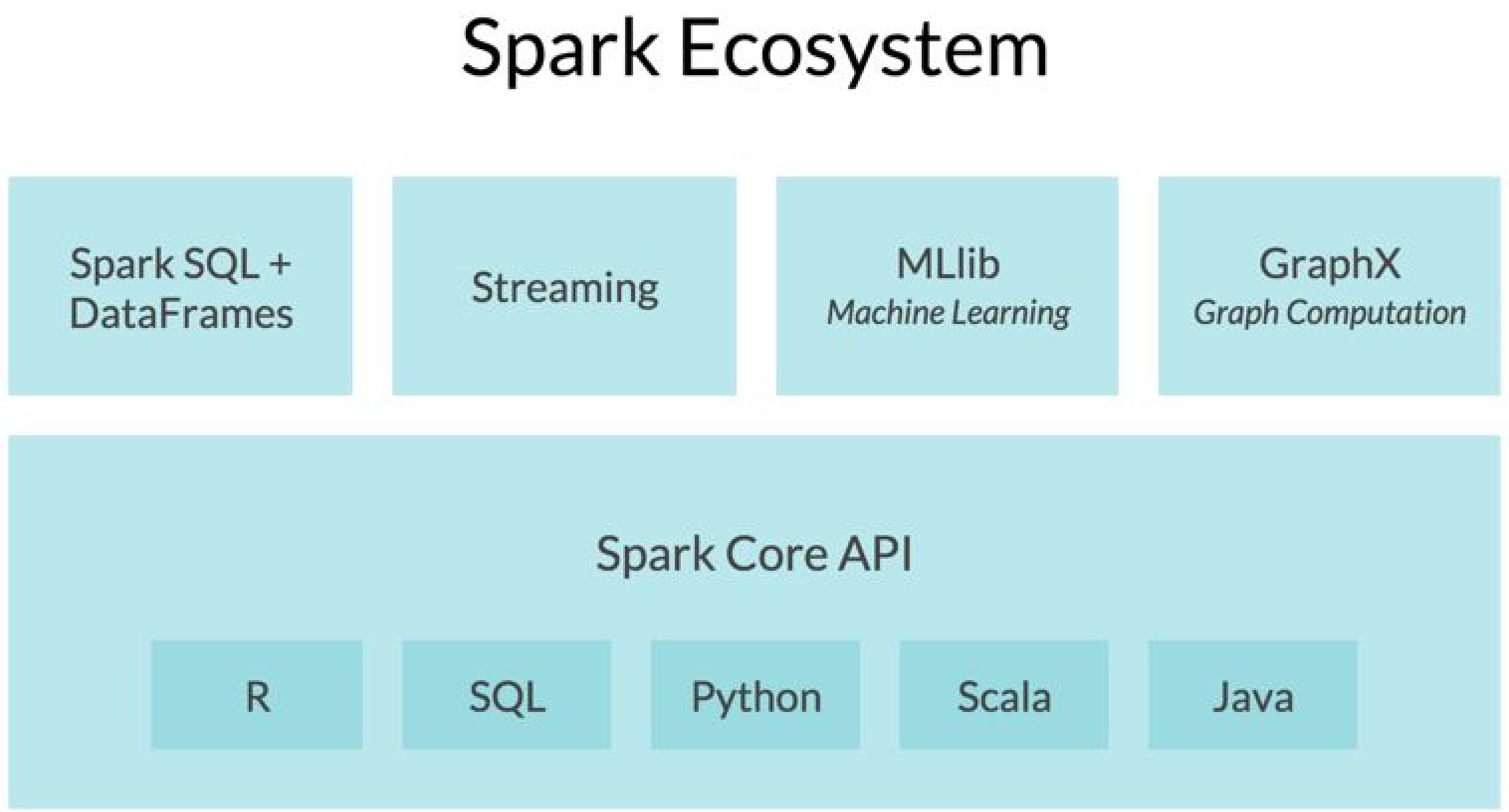
What is Spark?

Apache Spark™ is a fast and general engine for large-scale data processing.
- spark.apache.org

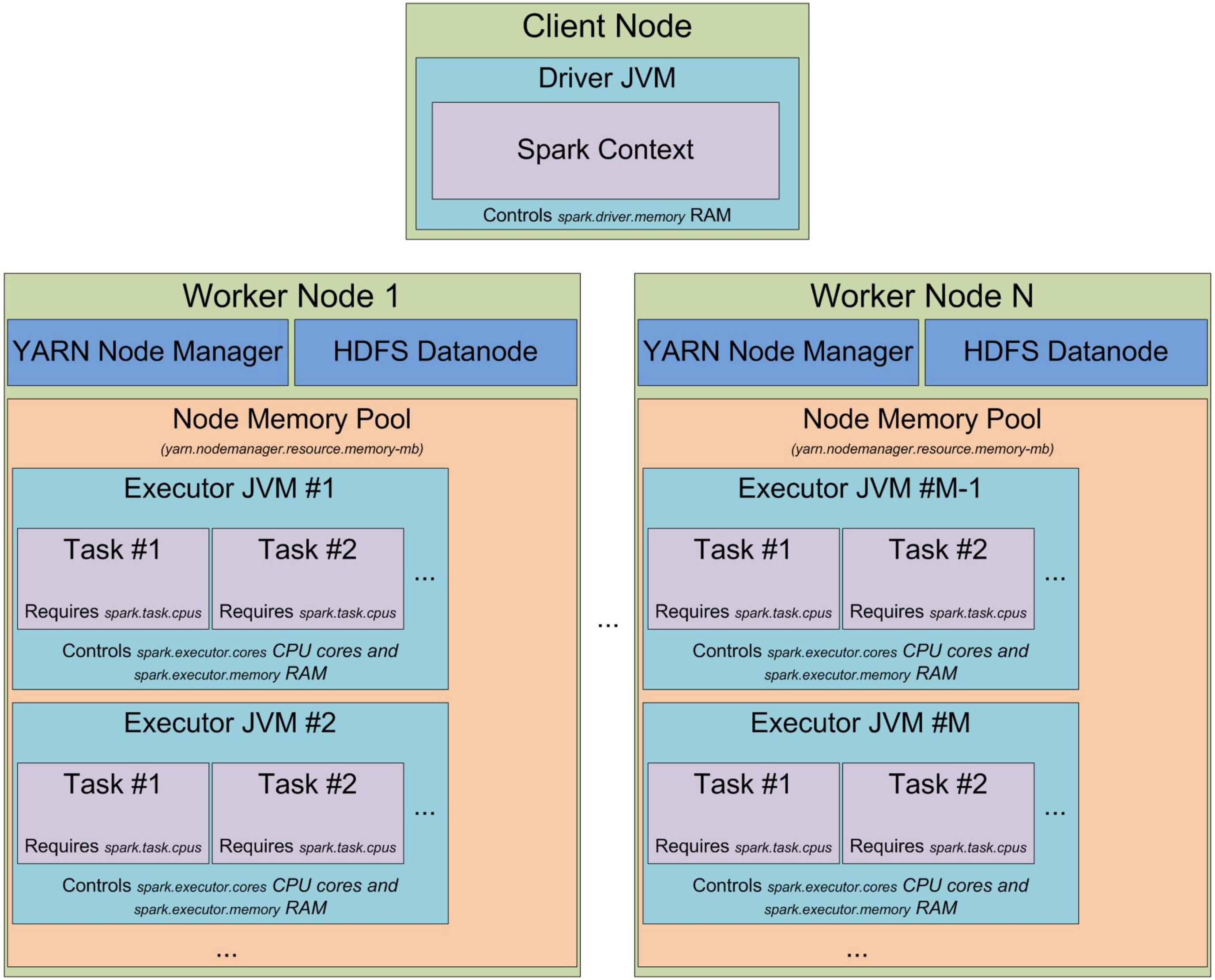
IS
- ❑ distributed
- ❑ python, R, scala, java
- ❑ Hadoop/HDFS
- ❑ in-memory

NOT
- ❑ magic

What is Spark?



How does Spark work?



http://0x0fff.com/spark-architecture/

Why is Spark a good choice for machine learning?

- Scalable
- Machine learning is iterative
- pySpark and SparkR = easy to adopt
- Spark is actively developing
  - new algorithms and functionality
  - extended APIs
  - Active user community

Enough background …
show me how this works

OSEMN data science with pySpark

THINKBIG
A TERADATA COMPANY

- OSEMN
- How pySpark does OSEM
- Practical example:
  - Bike share data set
- Highlight differences from python

Obtain

Scrub

Explore

Model

iNterpret

http://www.dataists.com/2010/09/a-taxonomy-of-data-science/
https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset

Obtain

Define HDFS path

```
In [1]: data_path = "hdfs:///user//hadoop//data//bike_sharing.csv"
        header_path = "hdfs:///user//hadoop//data//bike_sharing_headers.csv"
```

RDD = Resiliently Distributed Dataset

- the basic data structure in spark
- each entry is an unstructured record
- any type, any length
- distributed
- resilient
- lazy evaluation

```
In [2]: header_rdd = sc.textFile(header_path).filter(lambda x: x != "")

        plaintext_rdd = sc.textFile(data_path).filter(lambda x: x != "")

        plaintext_rdd = header_rdd.union(plaintext_rdd)

        print plaintext_rdd.take(2)
```

```
[u'instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersi
t,temp,atemp,hum,windspeed,casual,registered,cnt', u'1,1/1/11,sprin
g,0,1,0,0,6,0,clear,0.24,0.2879,0.81,0,3,13,16']
```

Obtain

Dataframe

- part of sparkSQL
- structured
- very similar to dataframes in pandas or R

```
In [1]: import pyspark_csv_mod as pycsv
        sc.addPyFile('pyspark_csv_mod.py')
```

```
In [10]: dataframe = pycsv.csvToDataFrame(sqlCtx, plaintext_rdd,
                                          sep =",", parseDate=False)
         dataframe.first()

Out[10]: Row(instant=1, dteday=u'1/1/11', season=u'spring', yr=0, mnth=1, h
         r=0, holiday=0, weekday=6, workingday=0, weathersit=u'clear', tem
         p=0.24, atemp=0.2879, hum=0.81, windspeed=0.0, casual=3, registere
         d=13, cnt=16)
```

Obtain

In [10]:

```
dataframe.dtypes
```

Out[10]:

```
[('instant', 'int'),
 ('dteday', 'string'),
 ('season', 'string'),
 ('yr', 'int'),
 ('mnth', 'int'),
 ('hr', 'int'),
 ('holiday', 'int'),
 ('weekday', 'int'),
 ('workingday', 'int'),
 ('weathersit', 'string'),
 ('temp', 'double'),
 ('atemp', 'double'),
 ('hum', 'double'),
 ('windspeed', 'double'),
 ('casual', 'int'),
 ('registered', 'int'),
 ('cnt', 'int')]
```

```
dataframe.describe(dataframe.columns[1:5]).show()
```

summary	dteday	season	yr	mnth
count	17379	17379	17379	17379
mean	null	null	0.5025605615973301	6.537775476149376
stddev	null	null	0.49999344348131836	3.43867677528789
min	1/1/11	fall	0	1
max	9/9/12	winter	1	12

```
dataframe.groupBy('season').count().show()
```

season	count
fall	4496
spring	4242
summer	4409
winter	4232

Dataframe functions

```
dataframe.describe(["season", "casual", "registered", "cnt"]).show()
```

```
+-------+------+------------------+------------------+------------------+
|summary|season|            casual|        registered|               cnt|
+-------+------+------------------+------------------+------------------+
|  count| 17379|             17379|             17379|             17379|
|   mean|  null| 35.67621842453536|153.78686920996606|189.46308763450142|
| stddev|  null|49.303611843417976| 151.3529312470881| 181.3823804311691|
|    min|  fall|                 0|                 0|                 1|
|    max|winter|               367|               886|               977|
+-------+------+------------------+------------------+------------------+
```

describe()

```
(dataframe.filter("season = 'spring'")
          .filter("yr = 1")
          .describe(["season", "casual", "registered", "cnt"])
          .show())
```

```
+-------+------+-----------------+------------------+------------------+
|summary|season|           casual|        registered|               cnt|
+-------+------+-----------------+------------------+------------------+
|  count|  2174|             2174|              2174|              2174|
|   mean|  null|18.02989880404784|129.78426862925483|147.81416743330266|
| stddev|  null|33.17212128339912|126.00430006459575|143.66992177564128|
|    min|spring|                0|                 1|                 1|
|    max|spring|              367|               681|               801|
+-------+------+-----------------+------------------+------------------+
```

filter()

Make season an integer to facilitate modeling

Scrub - transforms

In [74]:
```
(dataframe.withColumn('temp1', dataframe.yr * 2 + 2)
          .select(["yr","temp1"])
          .show(5)
```

```
+--+-----+
|yr|temp1|
+--+-----+
| 0|    2|
| 0|    2|
| 0|    2|
| 0|    2|
| 0|    2|
+--+-----+
```

```
from pyspark.sql import functions as F
(dataframe.withColumn('temp2', (F.when(dataframe.season == 'spring',0)
                              .when(dataframe.season == 'summer',1)
                              .when(dataframe.season == 'fall',2)
                              .when(dataframe.season == 'winter',3)
                              .otherwise(5)))
                              .groupBy('temp2')
                              .count()
                              .show(5))
```

```
+-----+-----+
|temp2|count|
+-----+-----+
|    0| 4242|
|    1| 4409|
|    2| 4496|
|    3| 4232|
+-----+-----+
```

```
dataframe.groupBy('season').count().show()
```

```
+------+-----+
|season|count|
+------+-----+
|  fall| 4496|
|spring| 4242|
|summer| 4409|
|winter| 4232|
+------+-----+
```

Scrub - mapping

```
Row(instant=1, dteday=u'1/1/11', season=u'spring', yr=0, mnth=1, hr=0, ho
liday=0, weekday=6, workingday=0, weathersit=u'clear', temp=0.24, atem
p=0.2879, hum=0.81, windspeed=0.0, casual=3, registered=13, cnt=16)
```

```
def transform_season(record):
    new_record = []
    for index, entry in enumerate(record):
        if index != 2:
            new_record.append(entry)
        else:
            if entry == 'spring':
                new_record.append(0)
            elif entry == 'summer':
                new_record.append(1)
            elif entry == 'fall':
                new_record.append(2)
            elif entry == 'winter':
                new_record.append(3)
    return new_record
```

Index 2 = season

```
transformed_rdd = dataframe.map(transform_season)

print transformed_rdd.first()
```

```
[1, u'1/1/11', 0, 0, 1, 0, 0, 6, 0, u'clear', 0.24, 0.2879, 0.81, 0.0, 3,
13, 16]
```

Scrub - adding a schema

In [24]:

```
from pyspark.sql.types import *
transformed_schema = []
old_schema = dataframe.schema
for index, field in enumerate(old_schema.fields):
    if index != 2:
        transformed_schema.append(field)
    else:
        transformed_schema.append(StructField('season',IntegerType(),True))
transformed_schema = StructType(transformed_schema)
transformed_df = sqlContext.createDataFrame(transformed_rdd,
                                schema = transformed_schema)
```

Index 2 = season

In [26]:

```
dataframe.groupBy('season').count().show()
```

```
+------+-----+
|season|count|
+------+-----+
|  fall| 4496|
|spring| 4242|
|summer| 4409|
|winter| 4232|
+------+-----+
```

In [27]:

```
transformed_df.groupBy('season').count().show()
```

```
+------+-----+
|season|count|
+------+-----+
|     0| 4242|
|     1| 4409|
|     2| 4496|
|     3| 4232|
+------+-----+
```

RDD vs. dataframe

Be comfortable switching between them

RDD	Dataframe
unstructured	has a schema
variation between rows	all rows have same structure
map operator	withColumn
scrub	scrub/explore
records can differ	consistency check
more flexibility	higher level API
	faster processing

Explore

Univariate statistics:

```
In [41]: transformed_df['season','temp', 'windspeed'].describe().show()
```

```
+-------+------------------+-------------------+-------------------+
|summary|            season|               temp|          windspeed|
+-------+------------------+-------------------+-------------------+
|  count|             17379|              17379|              17379|
|   mean|1.5016399102364923|0.49698716842164814|  0.190097606306452|
| stddev|    1.106886292562009|0.19255058126209854|0.12233670875036765|
|    min|                 0|               0.02|                0.0|
|    max|                 3|                1.0|             0.8507|
+-------+------------------+-------------------+-------------------+
```

Bivariate statistics:

```
In [42]: transformed_df.corr('temp', 'cnt')
Out[42]: 0.404772275786604
```

```
In [49]: for hr in range(13,21):
             hr_corr = transformed_df.filter("hr = " + str(hr)).corr('temp', 'cnt')
             print hr, hr_corr
```

```
13 0.396276166065
14 0.378465467714
15 0.388162260163
16 0.551861291267
17 0.587931522618
18 0.601333819698
19 0.677768466383
20 0.710147294589
```

```
transformed_df.crosstab("season", "weathersit").show()
```

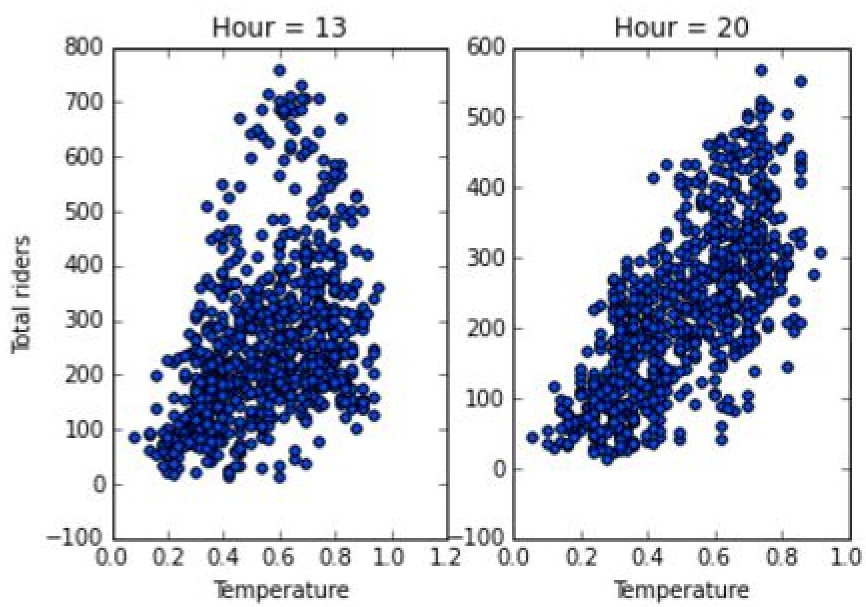
```
+-----------------+----+----+---+-+
|season_weathersit|   0|   1|  2|3|
+-----------------+----+----+---+-+
|                2|3280| 947|269|0|
|                1|2859|1144|406|0|
|                3|2609|1248|375|0|
|                0|2665|1205|369|3|
+-----------------+----+----+---+-+
```

0. Clear
1. Cloudy
2. Light precip.
3. Heavy precip.

Explore - plotting

Plots render on the driver node - must collect data

```
In [62]: count = 1
         for hr in [13, 20]:
             temp_data = (transformed_df
                          .filter("hr = " + str(hr))
                          .select("temp", "cnt")
                          .toPandas())
             plt.subplot(1,2,count)
             plt.scatter(x = temp_data['temp'], y = temp_data['cnt'])
             plt.title("Hour = " + str(hr))
             if count == 1:
                 plt.ylabel("Total riders")
             plt.xlabel("Temperature")
             count = count + 1
```



Model - MLlib - spark.ml vs. spark.mllib

MLlib divides into two packages:

- spark.mllib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.

spark.mllib	spark.ml
RDD	DataFrame
Labeled point	label and features columns
More algorithms (for now)	Recommended by Apache
	Pipeline abstraction

http://spark.apache.org/docs/latest/mllib-guide.html

Model - Prepare data for modeling

```
0          instant
1           dteday
2           season
3               yr
4             mnth
5               hr
6          holiday
7          weekday
8       workingday
9       weathersit
10            temp
11           atemp
12             hum
13       windspeed
14          casual
15      registered
16             cnt
```

Scale numerical variables

```
fields_to_model = ["temp", "atemp", "hum", "windspeed"]
stats = transformed_df.describe(fields_to_model).collect()

for index, i in enumerate(fields_to_model):
    col_avg = float(stats[1][index + 1])
    col_stdev = float(stats[2][index + 1])
    transformed_df = transformed_df.withColumn(i + '_norm',
                                                (transformed_df[i] - col_avg)/col_stdev)
```

```
(transformed_df.describe(["temp", "atemp",
                          "temp_norm", "atemp_norm"])
               .show())
```

```
+-------+------------------+------------------+-------------------+-------------------+
|summary|              temp|             atemp|          temp_norm|         atemp_norm|
+-------+------------------+------------------+-------------------+-------------------+
|  count|             17379|             17379|              17379|              17379|
|   mean|0.4969871684216481| 0.475775102134759|-1.77459141933407...|-1.39578575088677...|
| stddev|0.1925505812620985|0.1718452713725202| 1.0000000000002938| 0.9999999999971958|
|    min|              0.02|               0.0| -2.477204510605825| -2.76862492830718|
|    max|               1.0|               1.0|  2.612367245433343| 3.050563414847416|
+-------+------------------+------------------+-------------------+-------------------+
```

Model - prepare data for modeling

0	instant	9	weathersit
1	dteday	10	temp
2	season	11	atemp
3	yr	12	hum
4	mnth	13	windspeed
5	hr	14	casual
6	holiday	15	registered
7	weekday	16	cnt
8	workingday		

Labeled Point = (Label, Features)

```
from pyspark.mllib.regression import LabeledPoint
parsedData_rdd = transformed_df.map(lambda x: LabeledPoint(x[16], x[17:21]))
```

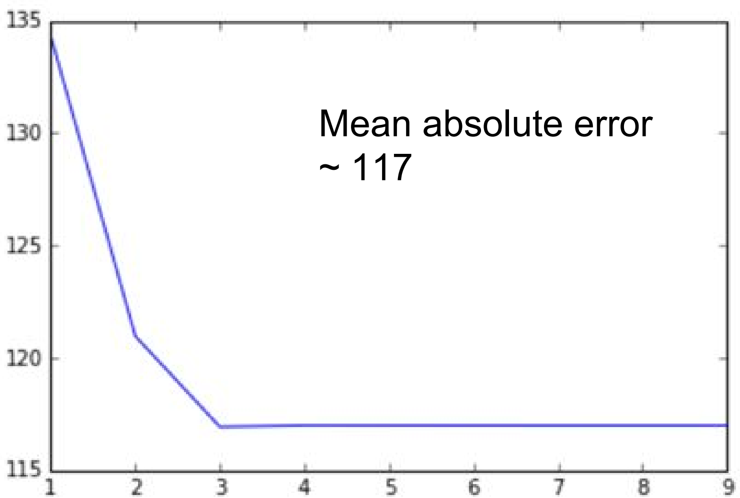
```
print transformed_df.first()
print parsedData_rdd.first()
```

```
Row(instant=1, dteday=u'1/1/11', season=0, yr=0, mnth=1, hr=0, holiday=0, weekday=6, workingd
ay=0, weathersit=0, temp=0.24, atemp=0.2879, hum=0.81, windspeed=0.0, casual=3, registered=1
3, cnt=16, temp_norm=-1.3346475857807065, atemp_norm=-1.0932806043129715, hum_norm=0.94737249
99667292, windspeed_norm=-1.5538885118650048)
(16.0,[-1.33464758578,-1.09328060431,0.947372499967,-1.55388851187])
```

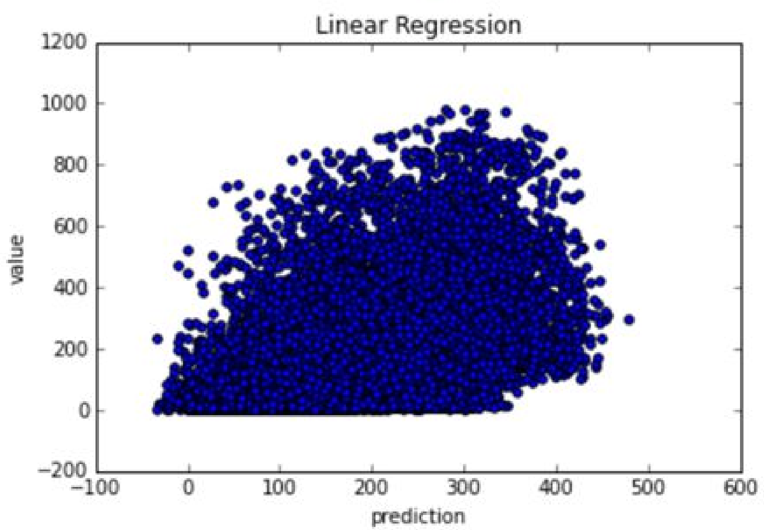
Model - linear regression with spark.mllib

```
from pyspark.mllib.regression import LinearRegressionWithSGD, LinearRegressionModel
nrecords = float(parsedData_rdd.count())
errors = []
for num_iter in range(1,10):
    #Train Model
    model = LinearRegressionWithSGD.train(parsedData_rdd, intercept = True,
                                          iterations = num_iter, step = 1)
    #Evaluate Model
    valuesAndPreds = parsedData_rdd.map(lambda x:(x.label, model.predict(x.features)))
    testMAE = valuesAndPreds.map(lambda (v, p): abs(v - p)).sum() / nrecords
    errors.append(testMAE)
    print valuesAndPreds.take(1)
```

```
temp = plt.plot(range(1, 10),errors)
```



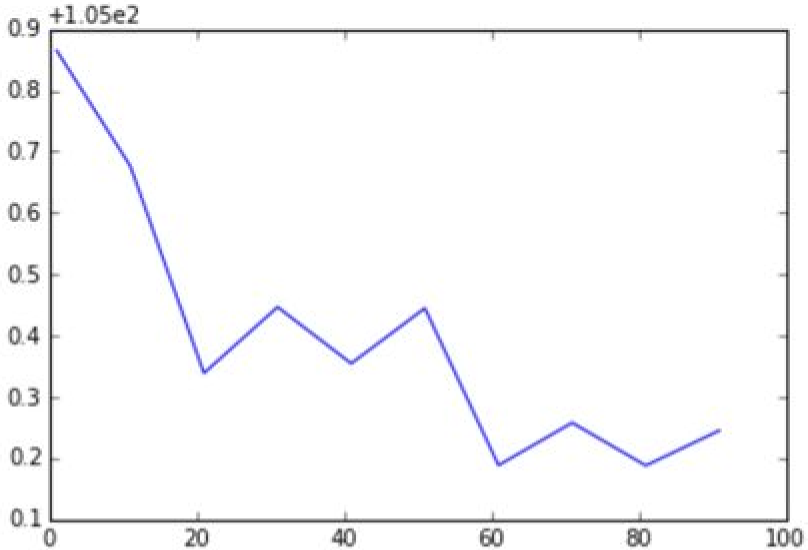

```
valuesAndPreds_local = valuesAndPreds.collect()

temp = plt.scatter([x[1] for x in valuesAndPreds_local],
                   [x[0] for x in valuesAndPreds_local])
plt.xlabel("prediction")
plt.ylabel("value")
temp = plt.title("Linear Regression")
```



Model - random forest with mllib

```
from pyspark.mllib.tree import RandomForest
errors = []
for num_trees in range(1, 101, 10):
    model = RandomForest.trainRegressor(parsedData_rdd, {},
                                        numTrees = num_trees, maxDepth = 10
                                        seed=42)
    #Evaluate Model
    predictions = model.predict(parsedData_rdd.map(lambda x: x.features))
    valuesAndPreds = parsedData_rdd.map(lambda x: x.label).zip(predictions)
    testMAE = (valuesAndPreds.map(lambda (v, p): abs(v - p))
                           .sum() / nrecords)
    errors.append(testMAE)
```

```
# Plot Errors
temp = plt.plot(range(1, 101, 10),errors)
```



Mean Absolute Error ~ 105

Model - random forest with mllib plus categorical

```
0          instant
1          dteday
2          season
3          yr
4          mnth
5          hr
6          holiday
7          weekday
8          workingday
```

```
 9        weathersit
10        temp
11        atemp
12        hum
13        windspeed
14        casual
15        registered
16        cnt
```

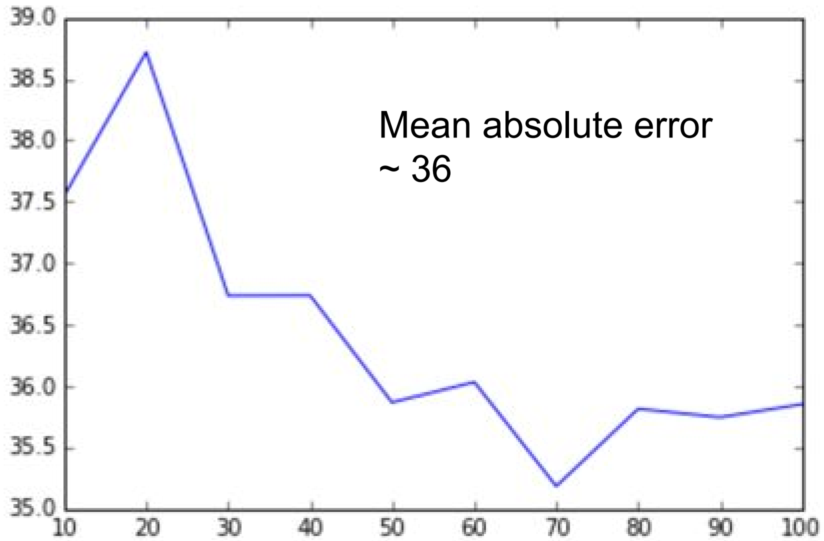
```
transformed_df.describe("season","yr", "hr", "mnth", ).show()
```

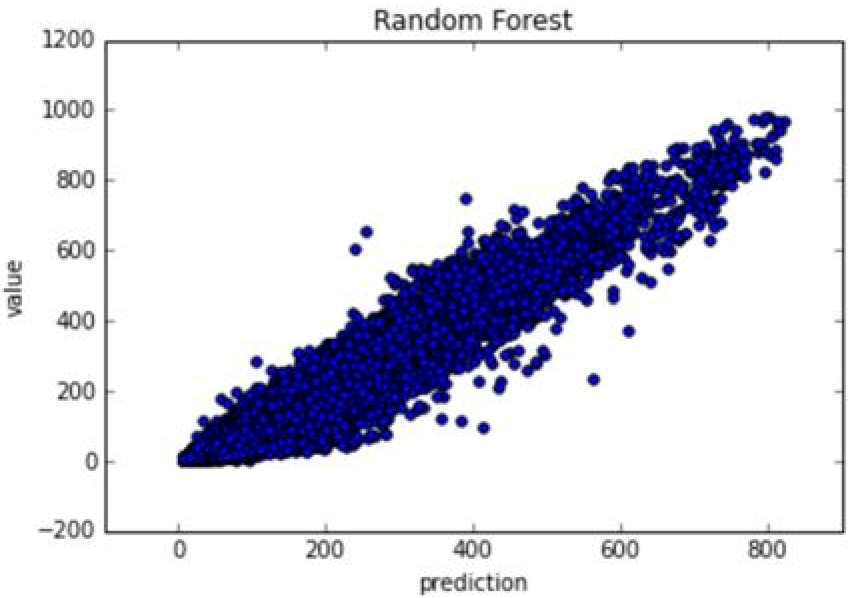
summary	season	yr	hr	mnth
count	17379	17379	17379	17379
mean	1.5016399102364923	0.5025605615973301	11.546751826917545	6.537775476149376
stddev	1.10688629256209	0.49999344348131836	6.914206162513383	3.438676777528789
min	0	0	0	1
max	3	1	23	12

```
parsedData_rdd2 = (transformed_df.withColumn("mnth2", transformed_df.mnth - 1)
                                 .select("season", "yr", "mnth2", "hr",
                                         "holiday", "weekday", "workingday", "weathersit",
                                         "temp", "atemp", "hum", "windspeed", "cnt"))
print parsedData_rdd2.first()
parsedData_rdd2 = (parsedData_rdd2.map(lambda x: LabeledPoint(x[12], x[0:12])))
```

Model - random forest with mllib plus categorical

```
errors = []
for num_trees in range(10, 101, 10):
    print num_trees,
    model = RandomForest.trainRegressor(parsedData_rdd2, {0:4, 1:2, 2:12, 3:24, 4:2,
                                                           5:7, 6:2, 7:4},
                                        numTrees = num_trees, maxDepth = 10, seed=23)
    #Evaluate Model
    predictions = model.predict(parsedData_rdd2.map(lambda x: x.features))
    valuesAndPreds = (parsedData_rdd2.map(lambda x: x.label)
                      .zip(predictions))
    testMAE = valuesAndPreds.map(lambda (v, p):
                              abs(v - p)).sum()/ nrecords
    errors.append(testMAE)
print ""
```

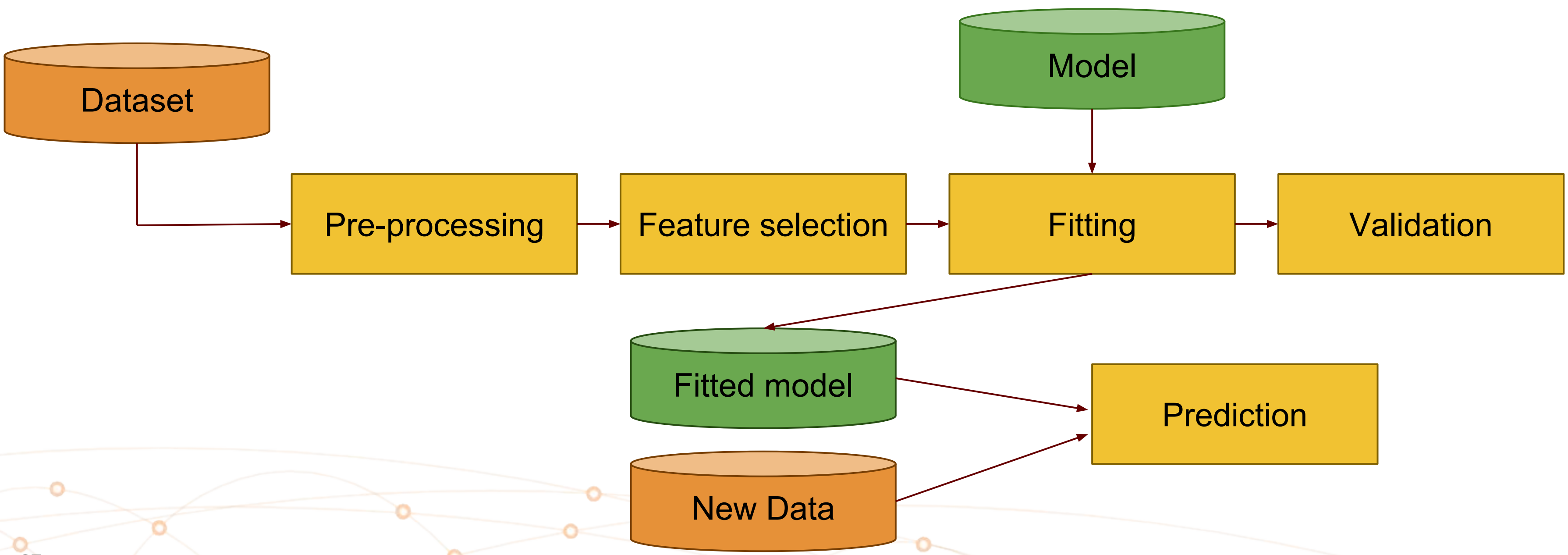




Model - linear regression with spark.ml

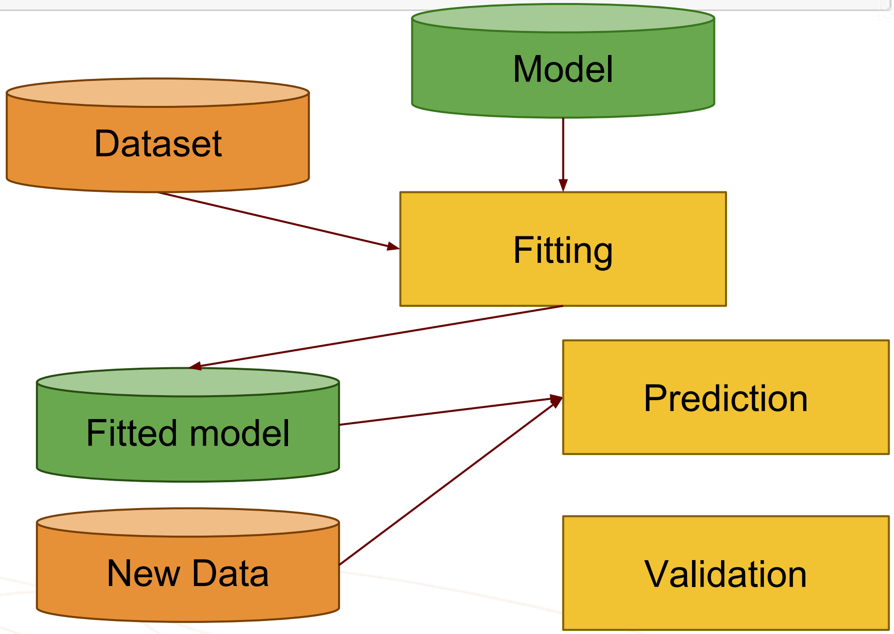
```
In [218]: # Put data into right format

          from pyspark.mllib.linalg import Vectors

          temp = transformed_df.map(lambda x: (float(x[16]), Vectors.dense(x[10:13])))
          parsedData_df = sqlContext.createDataFrame(temp, ["label", "features"])
          parsedData_df.first()

Out[218]: Row(label=16.0, features=DenseVector([0.24, 0.2879, 0.81]))
```

Pipeline concept



Model - linear regression with spark.ml

In [211]:
```
from pyspark.ml.regression import LinearRegression
errors = []
for num_iter in range(1,10):
    # Define model
    lr = LinearRegression(maxIter=num_iter, regParam=0.0)
    # Fit model
    lrModel = lr.fit(parsedData_df)
    # Predict model
    predictions = lrModel.transform(parsedData_df.select("features"))
    # Evaluate model
    valuesAndPreds = parsedData_df.select("label").rdd.zip(predictions.rdd)
    testMAPE = valuesAndPreds.map(lambda (v, p): abs(v.label - p.prediction)
                                  /v.label).sum() / nrecords
    errors.append(testMAPE)
# Plot Errors
temp = plt.plot(range(1,10),errors)
```



Deciding between ml and mllib

Pick one and stick with it

spark.mllib	spark.ml
RDD	DataFrame
Labeled point	label and features columns
SVMs	neural networks
	Pipeline abstraction -cross validation -oneHotEncoding
	probabilities for classifications

So that covers how, what about why?

When is big data a challenge?

Big data = when the amount of data that you have to look at exceeds your capability to look at it

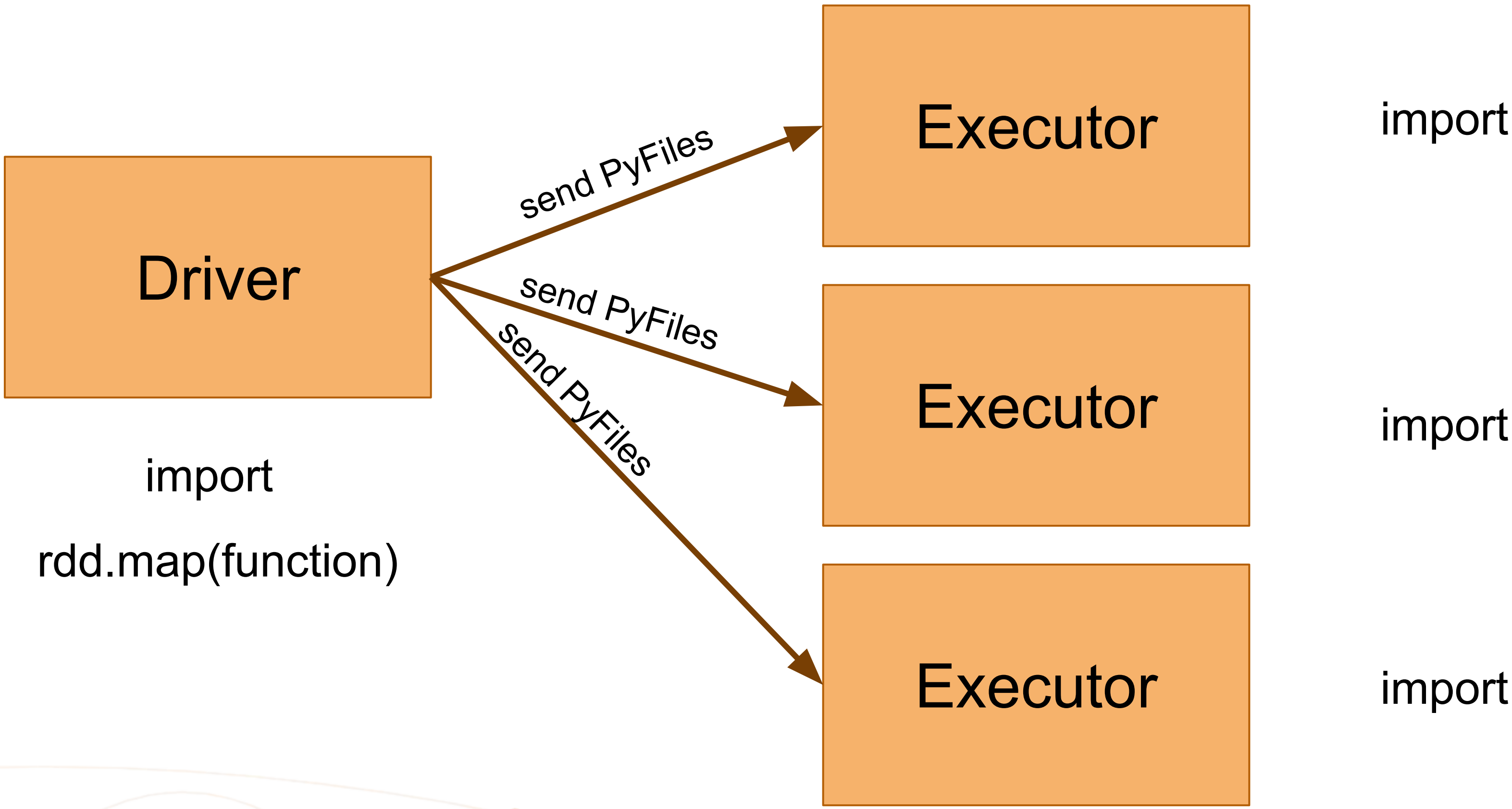
- Accessing data
  - data is too big to fit on one computer
  - reading data from one disk is too slow
- Computing with data
  - Single transformations - e.g. word count problem
  - Iterations - numerical solutions work around the analytical solution
    - e.g.gradient descent is a way to get an optimum set of parameters when solving the normal equation is too hard/ the matrix is too big.
- Visualizing data
  - Local machine - must render each data point
  - Viewer - must see each data point

Solutions:

- Distributed computing
- Pre-aggregate or sample

Distributed practicalities: which library goes where

```
In [1]: import pyspark_csv_mod as pycsv
        sc.addPyFile('pyspark_csv_mod.py')
```



Computing practicalities: caching RDDs

To cache or not to cache:

- Spark’s big advantage: in-memory
  - requires memory management

- Several options for storing RDDs
  - cache
    - Store data and lineage in-memory
  - persist
    - Store data and lineage, specify where
      - memory, disk, serialized, off-heap
  - checkpoint
    - store data on an external storage system
    - slower, but can survive worker failure

Take away points

1. Spark APIs make moving to distributed computing easier … but they are not magic.
2. RDDs and Dataframes:
    a. the basic distributed data structures in Spark.
    b. Dataframes have schemas; RDDs don't.
    c. Be comfortable switching between these
3. mllib and ml
    a. two machine learning libraries in Spark
    b. hard to interconvert
    c. suggest ml for new developers

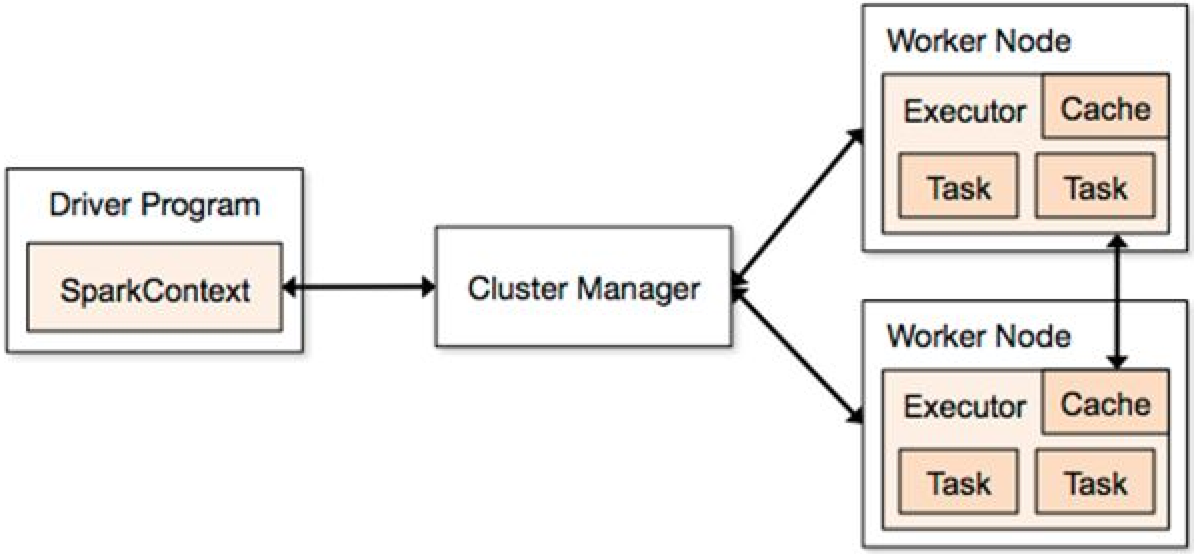
Questions?

Full Jupyter notebook available at:
https://github.com/QuantumPlatypus/Big-Mountain-Data-Talk

THINKBIG

A TERADATA COMPANY

How does Spark work?



http://spark.apache.org/docs/1.3.0/cluster-overview.html

Debugging/optimizing practicalities in pySpark

- lazy evaluation
- timing
- communication vs. computation cost
  - quantiles example