Sparking spark

Jonah Hooper

6 July 2018

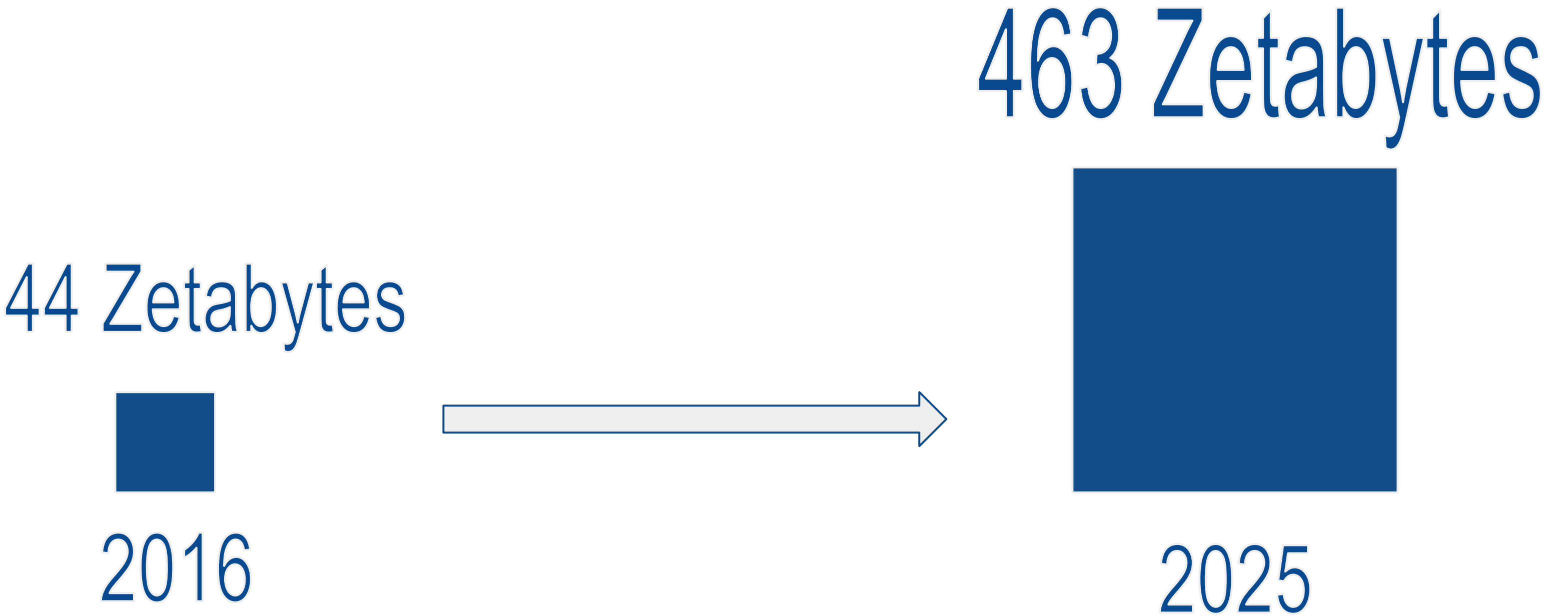
Purpose

- understand why spark exists
- have an idea of how it works
- have an idea of what to start reading if you want to understand more

Road Map for Talk

1. Theory
   a. Horizontal vs Vertical Scaling
   b. Lazy Evaluation
   c. Directed Acyclic Graphs
2. Spark Concepts
   a. RDDs
   b. Partitioning RDDs
   c. DataFrames
3. Code Example
   a. Pyspark code examples
   b. RDD example
   c. DataFrame Example
   d. Query plan explanations
4. Live Demo
   a. Subject to the law of Live Demos
5. Questions & Discussion

Motivation | Future Data Volumes



Motivation | Gigabyte -> Terrabyte

Gigabyte

All hacker news comments
(6.3 Gbs)

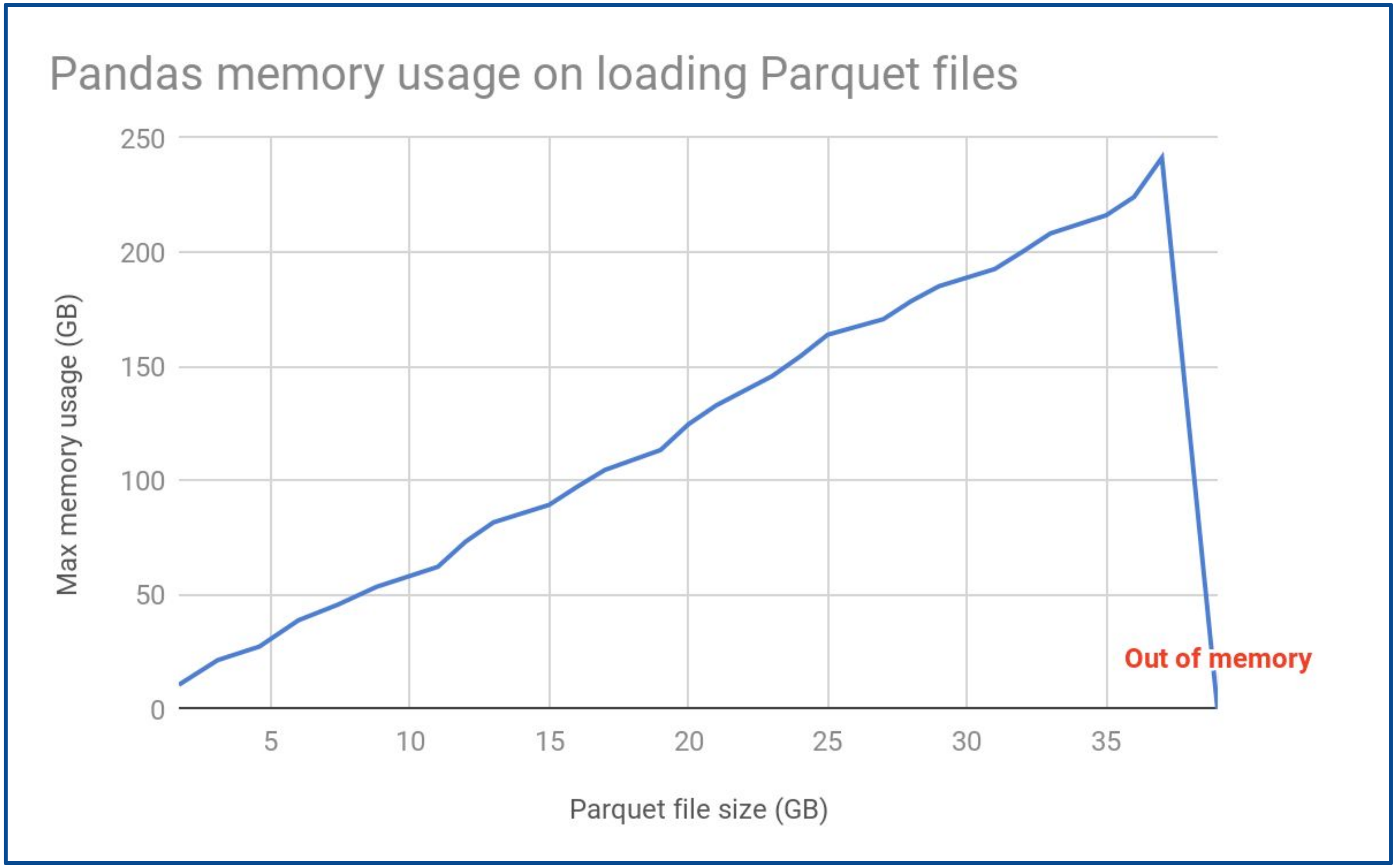
Terrabyte

Spark | In a nutshell

- Open source distributed cluster computing system
- Designed to scale horizontally across an arbitrarily large compute cluster
- Allows you to process huge amounts of data scalably
- Provides powerful API primitives for programmers to create high performance analytics queries
- Written in the Scala Programming language
- Has bindings for Python, Java, Scala and R
- We will be touching on Spark Core and Spark SQL

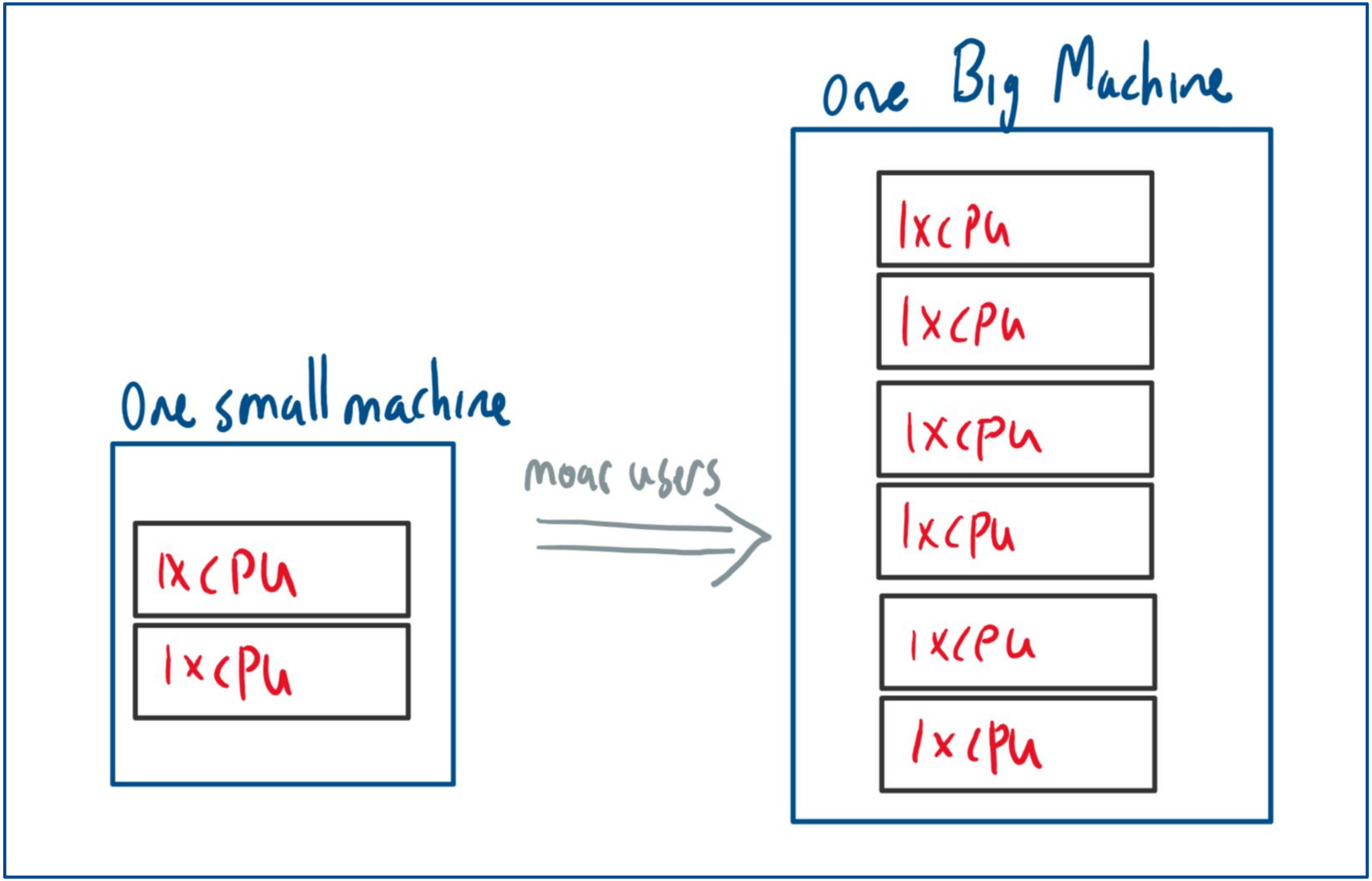
Pandas anyone?

Motivation | My pandas/numpy is borken

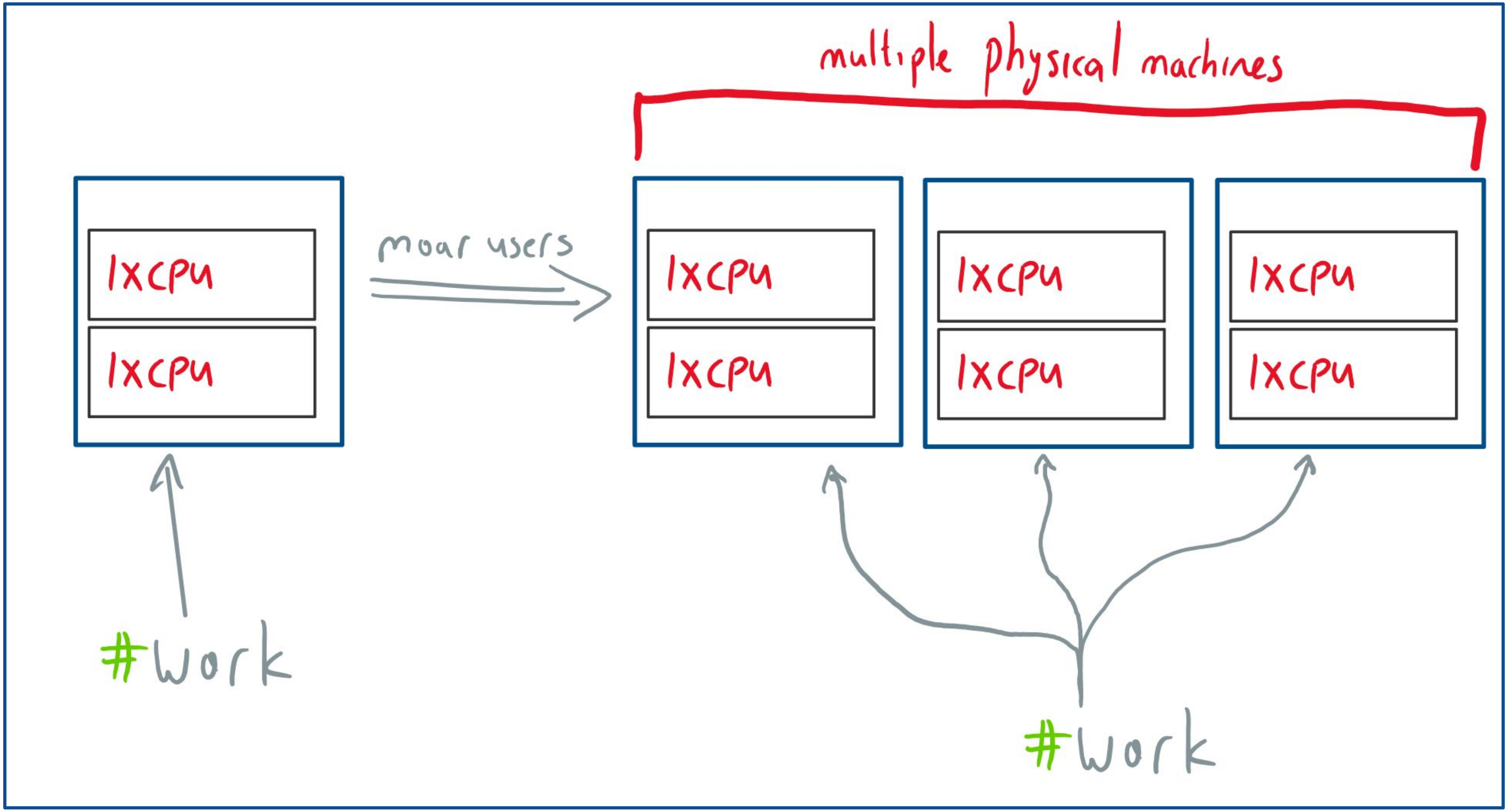


Horizontal Scaling?

Theory | Vertical Scaling



Theory | Horizontal Scaling



Lazy evaluation?

Theory | Lazy Evaluation

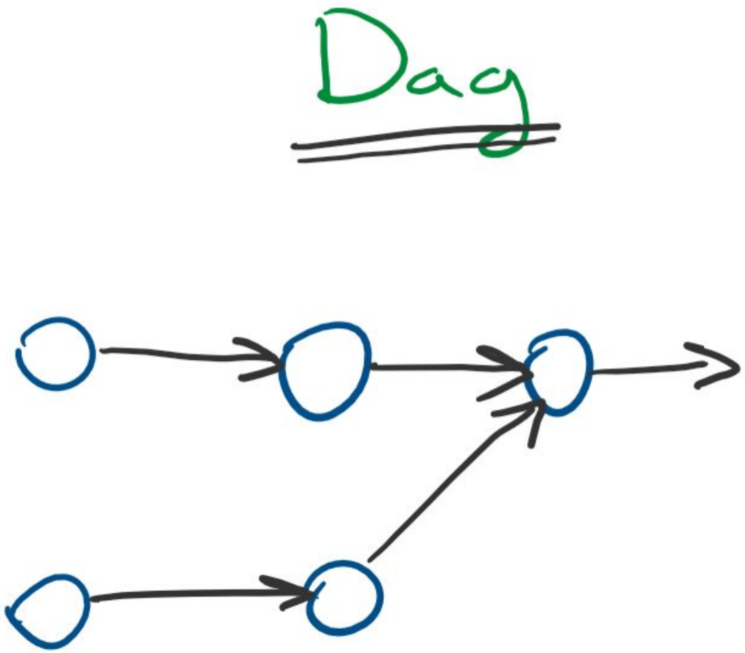
Also known as the student model of computation

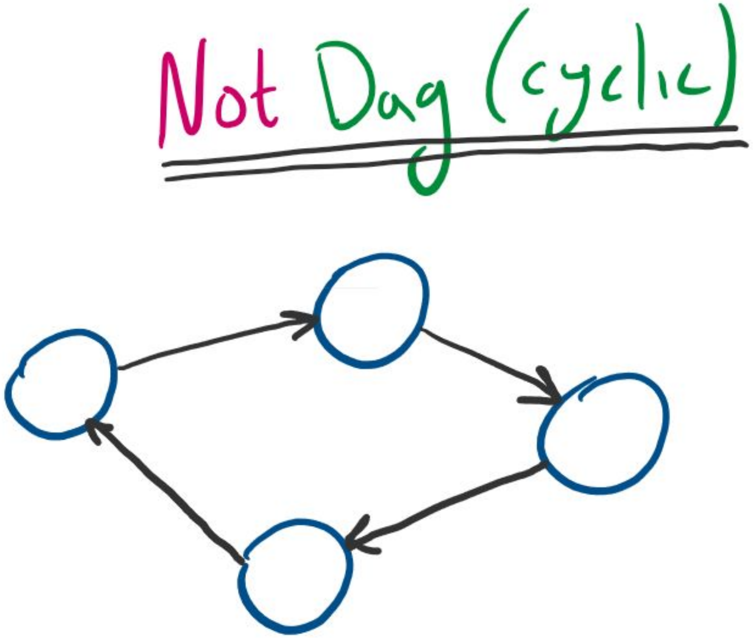
```
def integers_lazy():
    x = 1
    while True:
        yield x
        x += 1


for x in integers_lazy():
    print(x)
    if 10 <= x:
        break

>> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

```
def integers_eager():
    x = 1
    integers = []
    while True:
        integers.append(x)
        x += 1
    return integers

for x in integers_eager():
    print(x)
    if 10 <= x:
        break

>> …… halting problem?
```

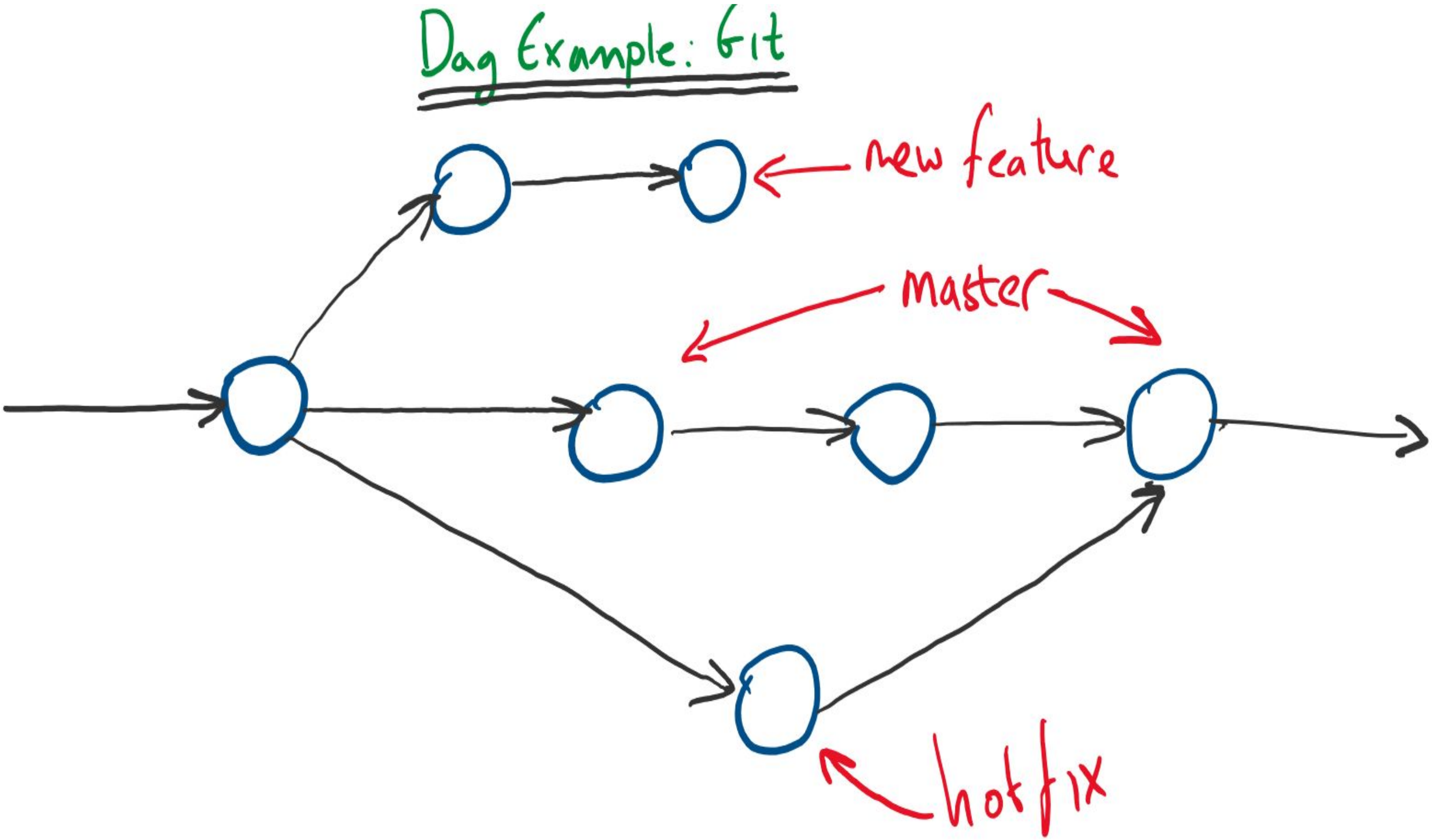
Directed Acyclic Graphs (DAGS)?

Theory | Directed Acyclic Graphs



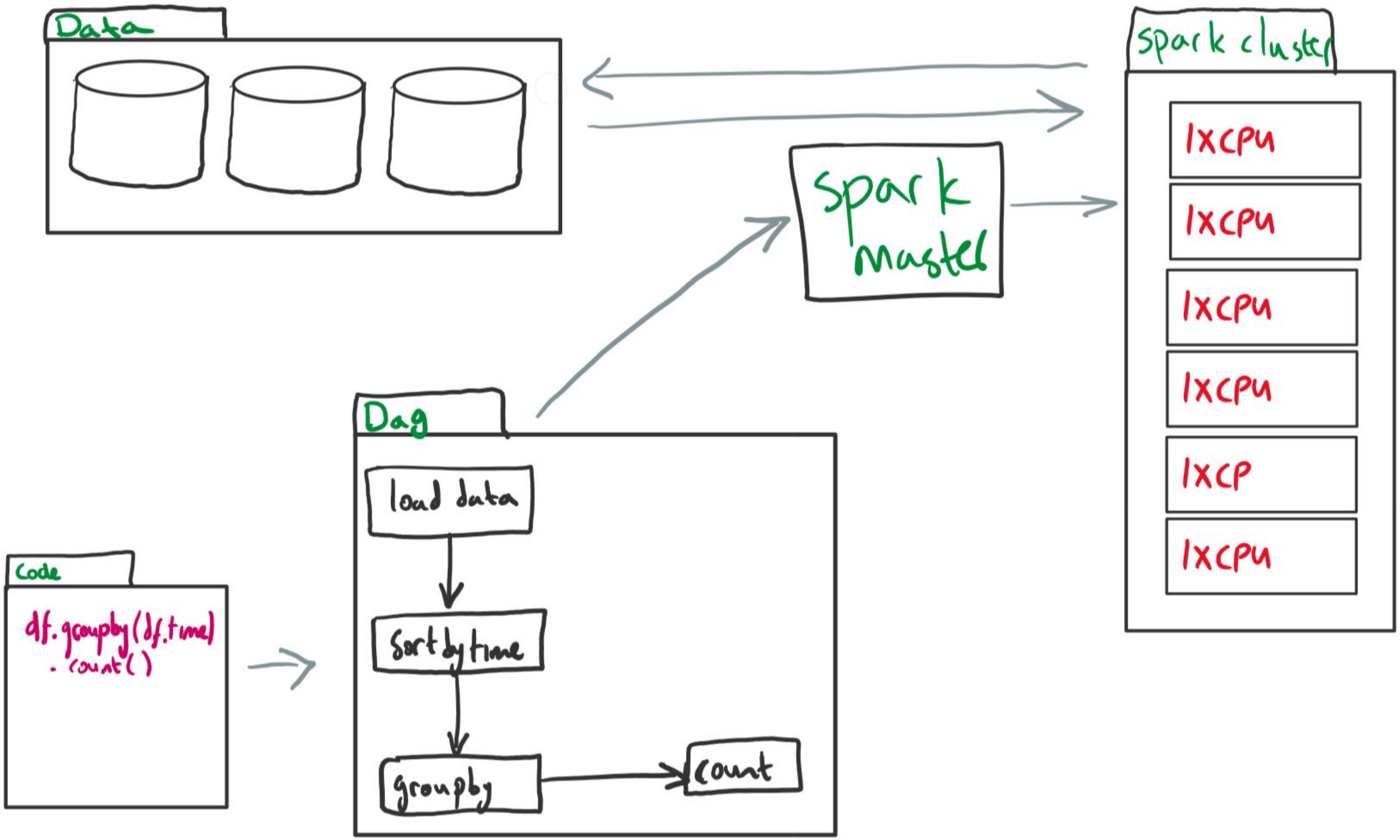


Theory | Dag Examples

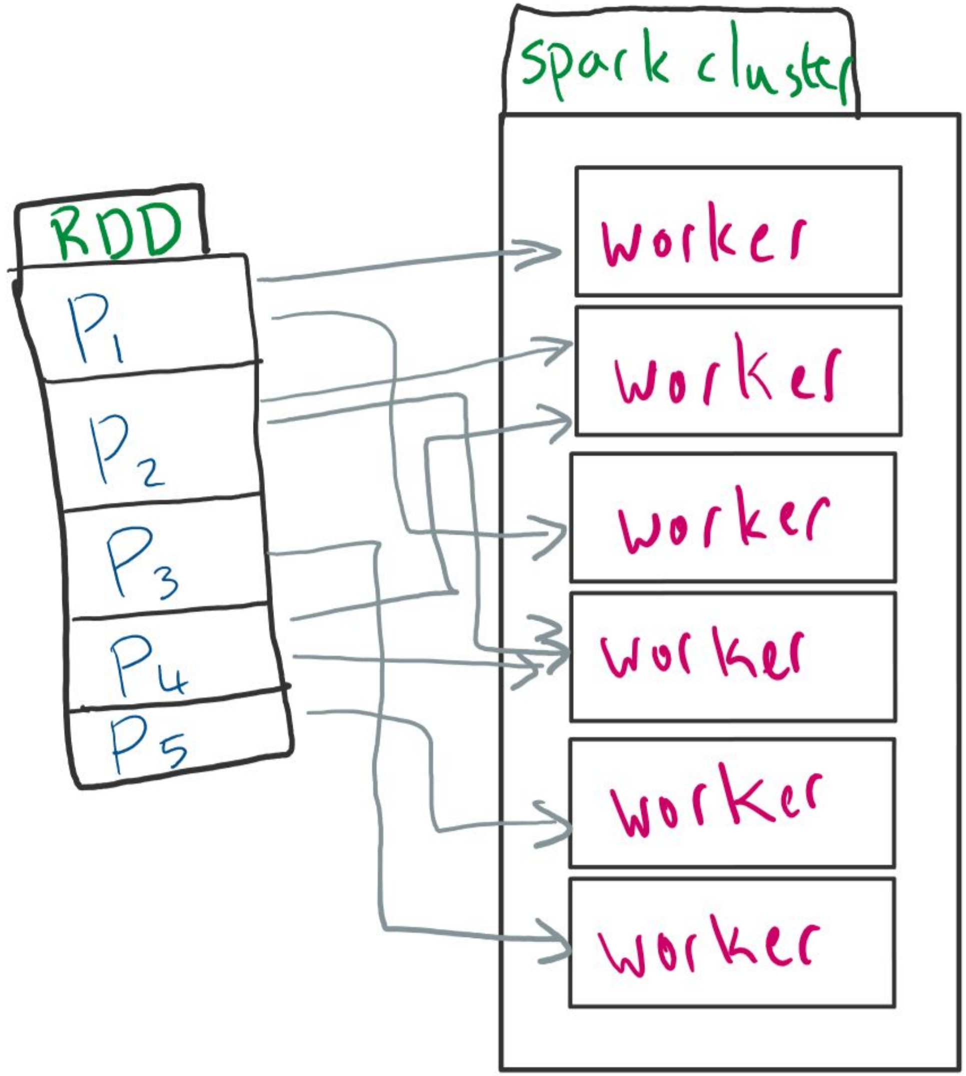


APACHE Spark™

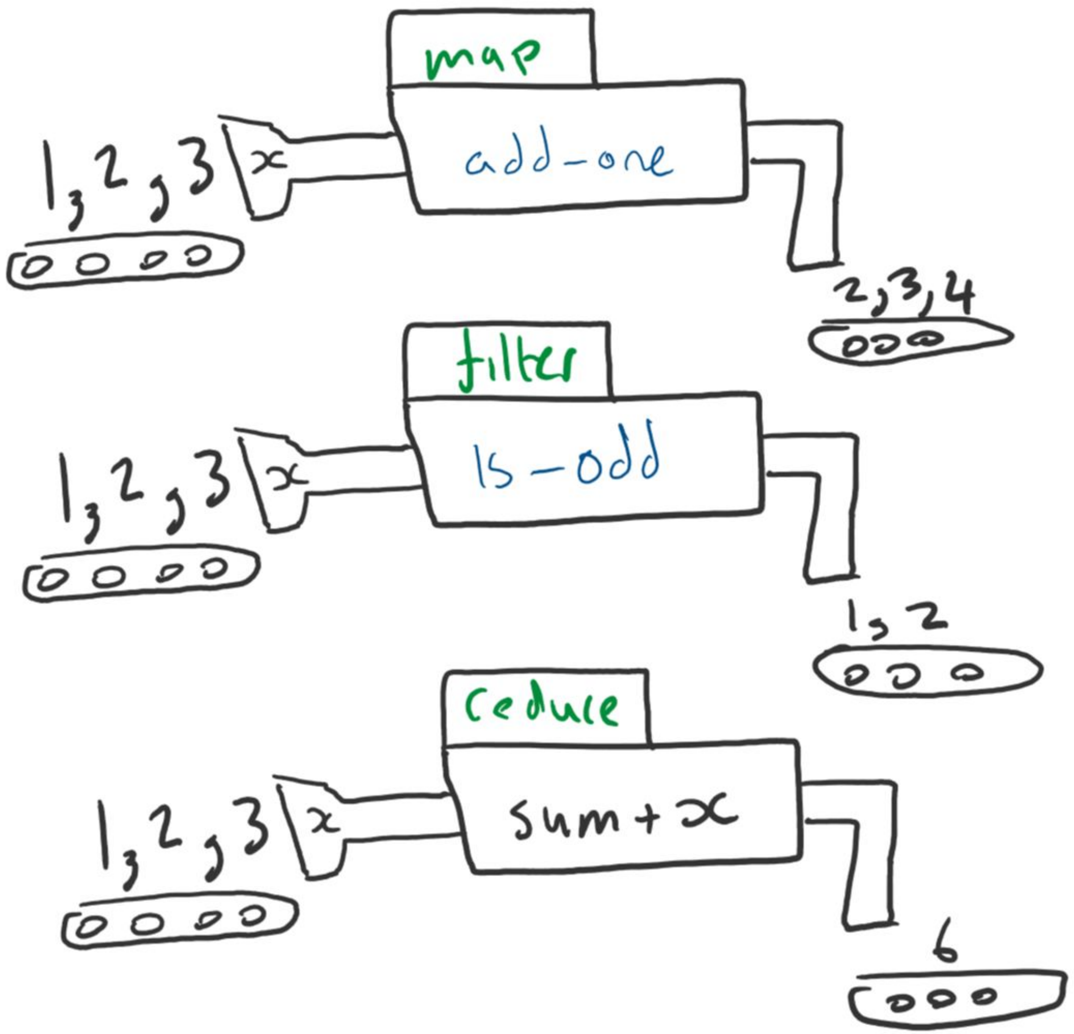
Spark | High Level Overview



Spark Concept | RDD



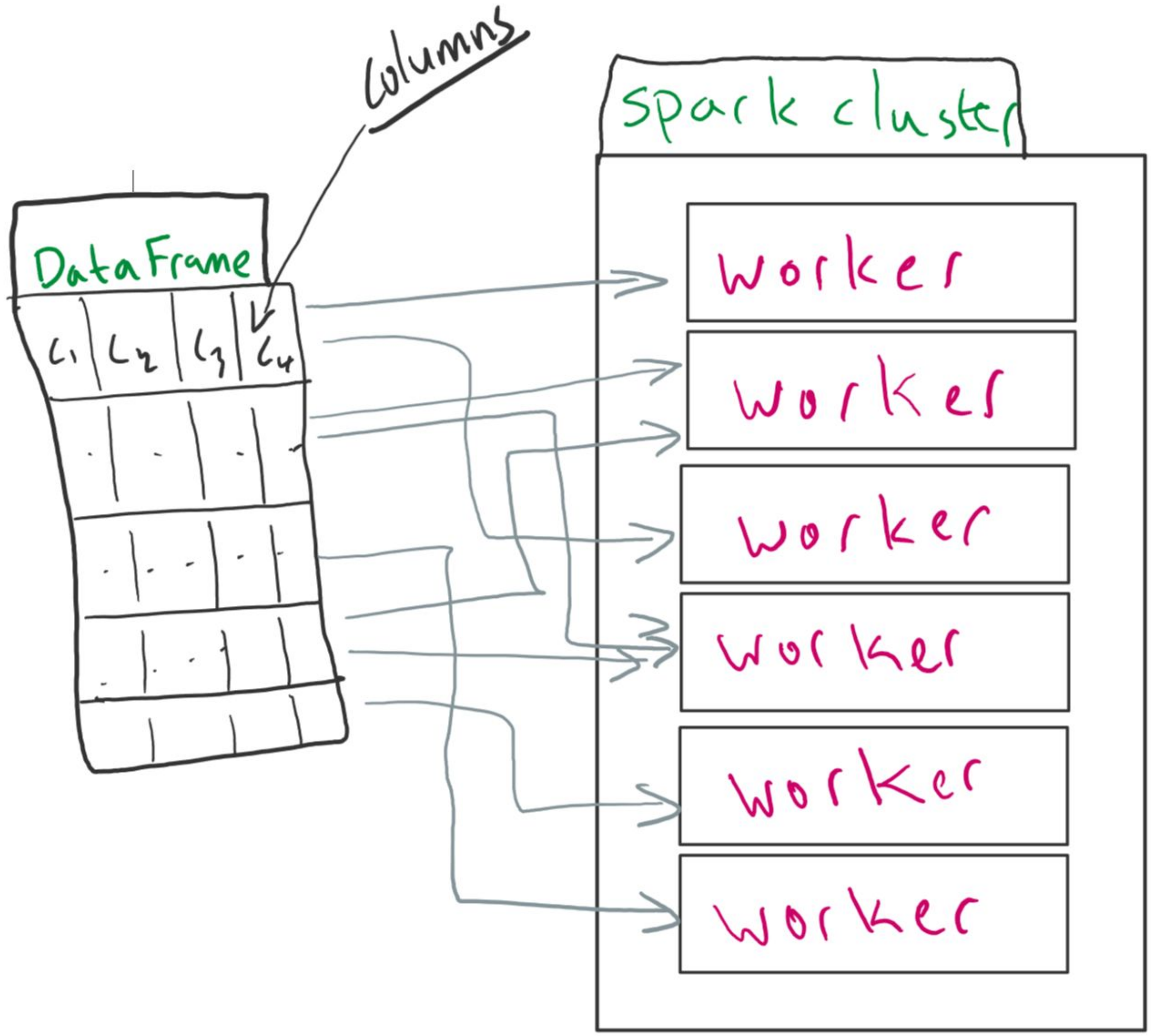
Spark Concept | Map Filter Reduce



Spark Concept | RDD Operations

- **Transformations** (map, flatMap)
- **Reduce** (reduce)

Spark Concept | DataFrame



Spark | RDD Programming Example

```
import random

rdd = sc.range(100)\                                  <=  tell spark to lazily create a list of numbers
    .map(lambda i: random.randint(1, 10))\            <=  for each of those random numbers - create a random integer
    .filter(lambda x: x > 5)                          <=  get all the random numbers that are greater than 5

print("Items:", rdd.take(20))                         <=  get the first 20 numbers in the RDD

sum_of_rdd = rdd.reduce(lambda sum_so_far,  x: sum_so_far + x)    <=  compute the sum of the first N numbers
print("Count:", sum_of_rdd)
```

Spark | Another RDD Programming Example

```
import re

from pprint import pprint



data = sc.textFile("*.py")\
       .flatMap(lambda line: re.split("\s+", line))\
       .filter(lambda word: len(word) >= 1) \
       .map(lambda word: (word, 1)) \
       .reduceByKey(lambda x, y: x + y)
print("First 10 terms according to spark")
pprint(data.take(10))
print("What are the most commonly occuring symbols in the code
base?")
pprint(data.takeOrdered(10, lambda word_count: -word_count[1]))
print("How many functions did we define?")
pprint(data.filter(lambda word_count: word_count[0] == "def").take(1))
```

<= flatMap :: f: (A -> [B]) -> [A] -> [B]

<= get rid of words with length greater than 1

<= make a list of tuples where the word is the first item, second item is 1

<= for each word that is the same, add whatever it’s value is

Spark | Code Example

```
import re

from pprint import pprint



data = sc.textFile("*.py")\

      .flatMap(lambda line: re.split("\s+", line))\

      .filter(lambda word: len(word) >= 1) \

      .map(lambda word: (word, 1)) \

      .reduceByKey(lambda x, y: x + y)

print("First 10 terms according to spark")

pprint(data.take(10))

print("What are the most commonly occuring symbols in the code base?")

pprint(data.takeOrdered(10, lambda word_count: -word_count[1]))

print("How many functions did we define?")

pprint(data.filter(lambda word_count: word_count[0] == "def").take(1))
```

```
First 10 terms according to spark
[('import', 58),
 ('main():', 2),
 ('11.0,', 1),
 ('[5.0,', 1),
 ('[7.0,', 1),
 ('])', 5),
 ('==', 5),
 ('googleapiclient', 1),
 ('termcolor', 1),

("os.environ['GOOGLE_APPLICATION_CRED
ENTIALS']", 1)]
What are the most commonly occuring
symbols in the code base?
[('=', 87),
 ('import', 58),
 ('#', 30),
 ('from', 23),
 ('as', 21),
 ('def', 18),
 ('return', 13),
 ('"""', 12),
 ('\\', 11),
 ('->', 11)]
How many functions did we define?
[('def', 18)]
```

Spark | DataFrame Example

```
import pyspark.sql.functions as F

df = spark.read.format("csv").option("header", "true").load("trades.csv")

df.show()
```

```
+------+--------+------+-------------+-----+-------+-----+
|    id|exchange|symbol|         date|price| amount| sell|
+------+--------+------+-------------+-----+-------+-----+
|956749|      bb|btcjpy|1464652825000|59683|  0.645| true|
|956750|      bb|btcjpy|1464652827000|59717| 0.0225| true|
|956751|      bb|btcjpy|1464652847000|59813|    0.8|false|
|956752|      bb|btcjpy|1464652869000|59803| 0.3665|false|
|956753|      bb|btcjpy|1464652899000|59778|  0.445|false|
|956754|      bb|btcjpy|1464652913000|59778| 0.8225|false|
|956755|      bb|btcjpy|1464652928000|59774|   0.01| true|
|956756|      bb|btcjpy|1464652928000|59774|   0.01| true|
|956757|      bb|btcjpy|1464652930000|59807| 0.3015|false|
|956759|      bb|btcjpy|1464652947000|59774|   0.01| true|
```

Spark | DataFrame Example

```
import pyspark.sql.functions as F

df = spark.read.format("csv").option("header", "true").load("trades.csv")

df.show()
```

```
StructType(
  List(
     StructField(id,StringType,true),
     StructField(exchange,StringType,true),
     StructField(symbol,StringType,true),
     StructField(date,StringType,true),
     StructField(price,StringType,true),
     StructField(amount,StringType,true),
     StructField(sell,StringType,true)
  )
)
```

Spark | Another DataFrame Example

```
df = spark.read.format("csv").option("header", "true").load("trades.csv")

df = df.withColumn("date", df.date / 1000)

df = df.select(df.date, df.price, df.amount)

df.show()
```

```
+-------------+-----+------+
|         date|price|amount|
+-------------+-----+------+
|1.464652825E9|59683| 0.645|
|1.464652827E9|59717|0.0225|
|1.464652847E9|59813|   0.8|
|1.464652869E9|59803|0.3665|
|1.464652899E9|59778| 0.445|
|1.464652913E9|59778|0.8225|
|1.464652928E9|59774|  0.01|
|1.464652928E9|59774|  0.01|
| 1.46465293E9|59807|0.3015|
|1.464652947E9|59774|  0.01|
+-------------+-----+------+
```

Spark | Another DataFrame Example

```
df = spark.read.format("csv").option("header", "true").load("trades.csv")
```

```
df = df.withColumn("date", df.date / 1000)
```

```
df = df.select(df.date, df.price, df.amount)
```

```
df.show()
```

```
StructType(
  List(
    StructField(id,StringType,true),
    StructField(exchange,StringType,true),
    StructField(symbol,StringType,true),
    StructField(date,DoubleType,true),
    StructField(price,StringType,true),
    StructField(amount,StringType,true),
    StructField(sell,StringType,true)
  )
)
```

Spark | When is the market most liquid?

```
import pyspark.sql.functions as F


df = spark.read.format("csv").option("header", "true").load("trades.csv")
df = df.withColumn("date", df.date / 1000)
df = df.select(df.date, df.price, df.amount)
df = df.withColumn("day", F.dayofweek(F.from_unixtime(df.date)))\
     .withColumn("hour", F.hour(F.from_unixtime(df.date)))\
     .withColumn("volume", df.price * df.amount)
trades_by_day = df.groupBy(df.day, df.hour).sum("volume")
ordered_trades = trades_by_day.orderBy("sum(volume)", ascending=False)
ordered_trades.show()
```

Similarities to the RDD processing example?

Recall

```
data = sc.textFile("*.py")\
      .flatMap(lambda line: re.split("\s+", line))\
      .filter(lambda word: len(word) >= 1) \
      .map(lambda word: (word, 1)) \
      .reduceByKey(lambda x, y: x + y)
```

```
(10) PythonRDD[554] at RDD at PythonRDD.scala:49 []
 |  MapPartitionsRDD[548] at mapPartitions at PythonRDD.scala:129 []
 |  ShuffledRDD[547] at partitionBy at NativeMethodAccessorImpl.java:0 []
 +-(10) PairwiseRDD[546] at reduceByKey at <ipython-input-53-a0211c700b62>:4 []
    |  PythonRDD[545] at reduceByKey at <ipython-input-53-a0211c700b62>:4 []
    |  *.py MapPartitionsRDD[544] at textFile at NativeMethodAccessorImpl.java:0 []
    |  *.py HadoopRDD[543] at textFile at NativeMethodAccessorImpl.java:0 []
```

Spark | Dataframe query plans

Dataframe Query Plans

```
ordered_trades = trades_by_day.orderBy("sum(volume)", ascending=False)
```

```
== Optimized Logical Plan ==
Sort [sum(volume)#2653 DESC NULLS LAST], true
+- Aggregate [day#2628, hour#2633], [day#2628, hour#2633, sum(volume#2639) AS sum(volume)#2653]
   +- Project [dayofweek(...) AS day#2628, hour(...) AS hour#2633,
               (cast(price#2563 as double) * cast(amount#2564 as double)) AS volume#2639]
      +- Relation[id#2559,exchange#2560,symbol#2561,date#2562,price#2563,amount#2564,sell#2565] csv
```

Spark | Unoptimized plan is quite scary

Eventually - operations on **DataFrames** become operations on **RDDS**

```
ordered_trades = trades_by_day.orderBy("sum(volume)", ascending=False)
```

```
== Analyzed Logical Plan ==
day: int, hour: int, sum(volume): double
Sort [sum(volume)#2653 DESC NULLS LAST], true
+- Aggregate [day#2628, hour#2633], [day#2628, hour#2633, sum(volume#2639) AS sum(volume)#2653]
   +- Project [date#2603, price#2563, amount#2564, day#2628, hour#2633, (cast(price#2563 as double) * cast(amount#2564 as double)) AS volume#2639]
      +- Project [date#2603, price#2563, amount#2564, day#2628, hour(cast(from_unixtime(cast(date#2603 as bigint), yyyy-MM-dd HH:mm:ss, Some(Africa/Johannesburg)) as timestamp), Some(Africa/Johannesburg)) AS hour#2633]
         +- Project [date#2603, price#2563, amount#2564, dayofweek(cast(from_unixtime(cast(date#2603 as bigint), yyyy-MM-dd HH:mm:ss, Some(Africa/Johannesburg)) as date)) AS day#2628]
            +- Project [date#2603, price#2563, amount#2564]
               +- Project [id#2559, exchange#2560, symbol#2561, (cast(date#2562 as double) / cast(1000 as double)) AS date#2603, price#2563, amount#2564, sell#2565]
                  +- Relation[id#2559,exchange#2560,symbol#2561,date#2562,price#2563,amount#2564,sell#2565] csv
```

Spark | RDD from Dataframe

Eventually - operations on **DataFrames** become operations on **RDDS**

```
ordered_trades = trades_by_day.orderBy("sum(volume)", ascending=False).rdd
```

```
(21) MapPartitionsRDD[621] at javaToPython at NativeMethodAccessorImpl.java:0 []
 |   MapPartitionsRDD[620] at javaToPython at NativeMethodAccessorImpl.java:0 []
 |   MapPartitionsRDD[619] at javaToPython at NativeMethodAccessorImpl.java:0 []
 |   ShuffledRowRDD[618] at javaToPython at NativeMethodAccessorImpl.java:0 []
 +-(200) MapPartitionsRDD[617] at javaToPython at NativeMethodAccessorImpl.java:0 []
    |   MapPartitionsRDD[613] at javaToPython at NativeMethodAccessorImpl.java:0 []
    |   ShuffledRowRDD[612] at javaToPython at NativeMethodAccessorImpl.java:0 []
    +-(1) MapPartitionsRDD[611] at javaToPython at NativeMethodAccessorImpl.java:0 []
       |  MapPartitionsRDD[610] at javaToPython at NativeMethodAccessorImpl.java:0 []
       |  FileScanRDD[609] at javaToPython at NativeMethodAccessorImpl.java:0 []
```

Spark | Our experience at DataProphet

- Processing 10s of Gigabytes of trading Data (not big data but getting there)
- You have to have a lot of data to justify using spark
- Pandas will get you results quickly until it doesn’t - runs out of memory
- Rather use tools like **Google DataProc** or **Amazon Elastic Map Reduce**
- Managing your own spark cluster can be taxing
- Use spark for computing features for your ML models

Spark | Other features

>> **links**.show( )

- ➔ https://spark.apache.org/graphx/
- ➔ https://spark.apache.org/streaming/
- ➔ https://spark.apache.org/mllib/

Questions and comments?