(Scala) Spark for Data Scientists

Ryoko Kita

Feb. 13, 2016

Cornell Tech Hackathon
NY, NY

https://github.com/ryokokita/Spark4DS.git

Conversation Flow

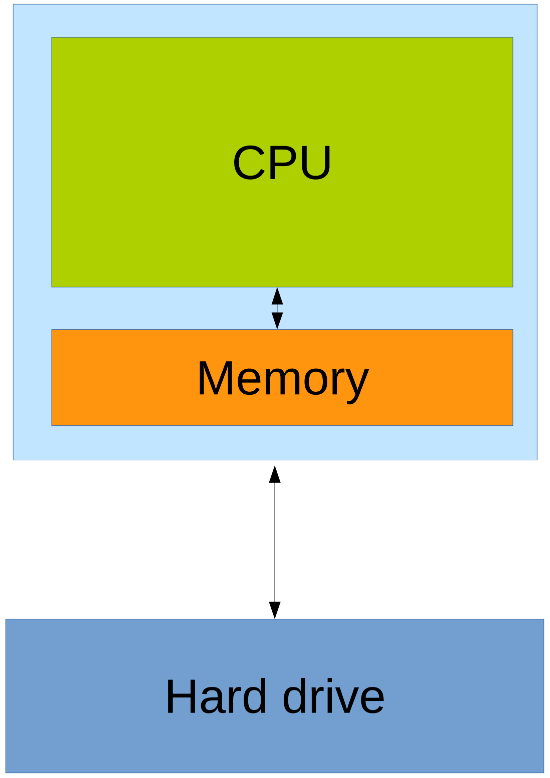
- whoami?
- End2End workflow for data scientists
- Think about your hardware to make your software GO!

Topics

- Hardware 101 for Data Scientists
  - CPU, Memory, Hard drive
  - Distributed Systems
- Software Stuff
  - Why Spark?
  - Why Scala?
- Data movement on hardware
- Airline data & data prep demo

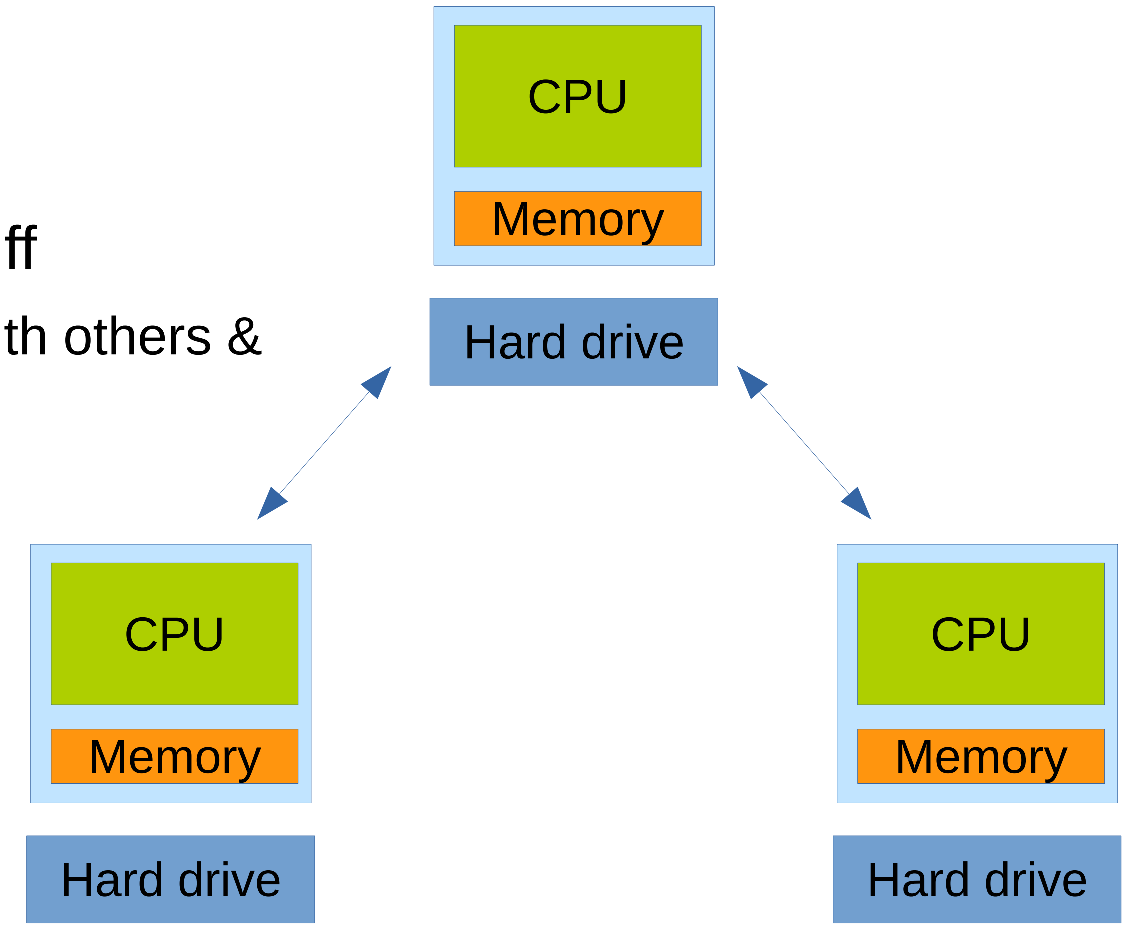
Computer Architecture 101

- CPU
  - Data transformations
  - Math-y stuff
  - Same for Spark & Hadoop MR
- Memory
  - Temporary/fast storage
  - Spark
- Hard disk/drive
  - (More) permanent storage
  - Hadoop hdfs



Distributed Systems 101

- Bunch of computers connected together
  - Via cables
  - Coordinated with software
- Generally running other stuff
  - i.e. you app has to share (with others & with other apps)
- Distributed data



Why Spark?

- One script, run anywhere:
  - Laptop, Hadoop cluster, etc.
- One script for end2end flow:
  - Data ingestion
  - Transformation
  - Analysis
  - Persist data
- Manages tasks for you
  - DAG analysis – Directed Acyclic Graph
  - Launch/Distribute/(Re)Launch...

Why Scala?

- A bit mind-bendy, but can code data transformations efficiently
- Schlepp functions around, not data
- Lazy eval
- Spark developed on Scala
  - GraphX only available in Scala
- Scala worksheets
  - Facilitates development in Scala for data scientists!

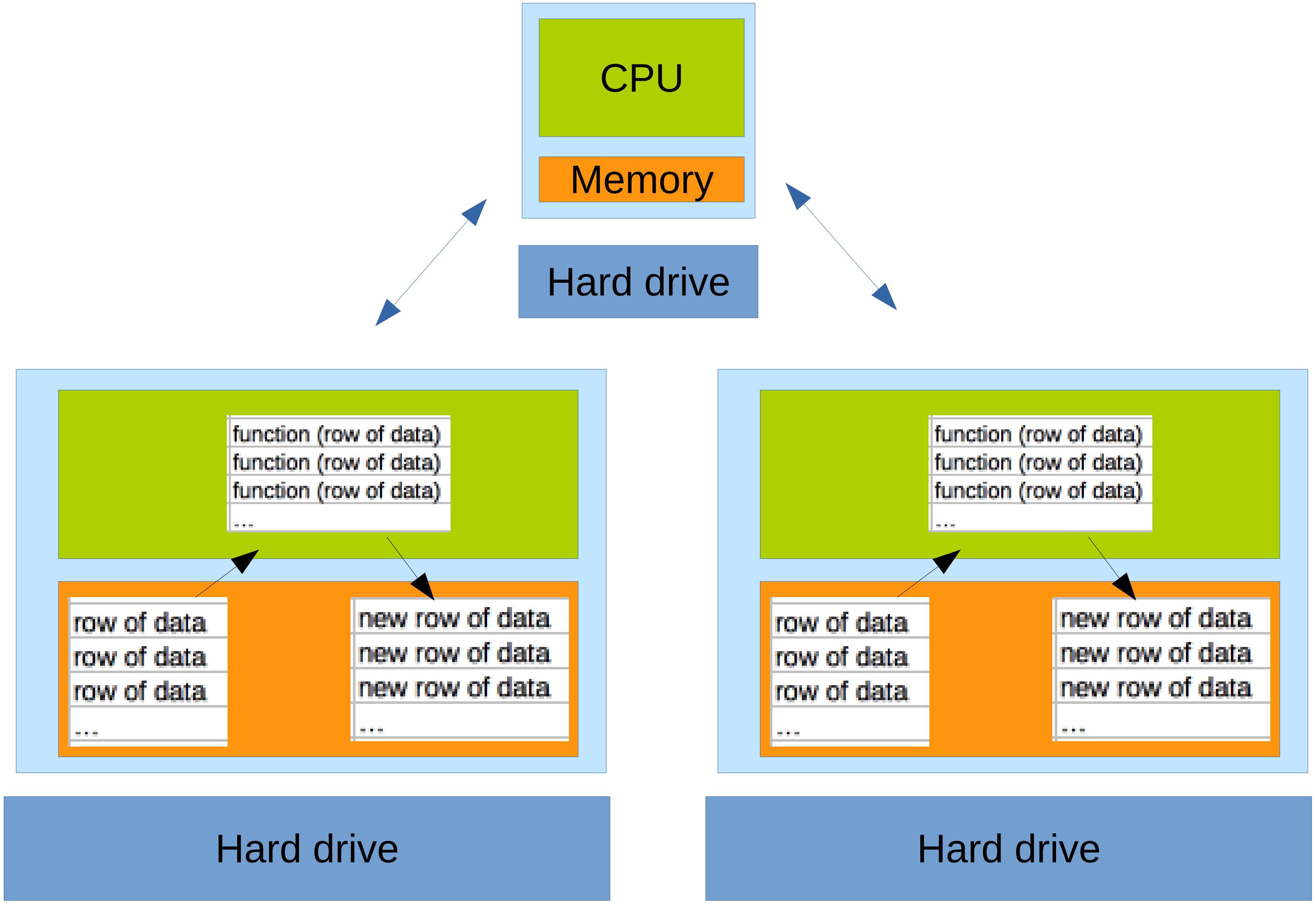
Airline Data for Demos

- Government data from airlines
  - http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236
- Airline, plane, flight info, etc
  - Clean up/reformatting required
  - Sample data available in data folder in git repo

Data Transformations

- What happens where?
  - Functions applied per row of data
  - Functions that aggregate rows
- Why is knowing this important?

Per Row

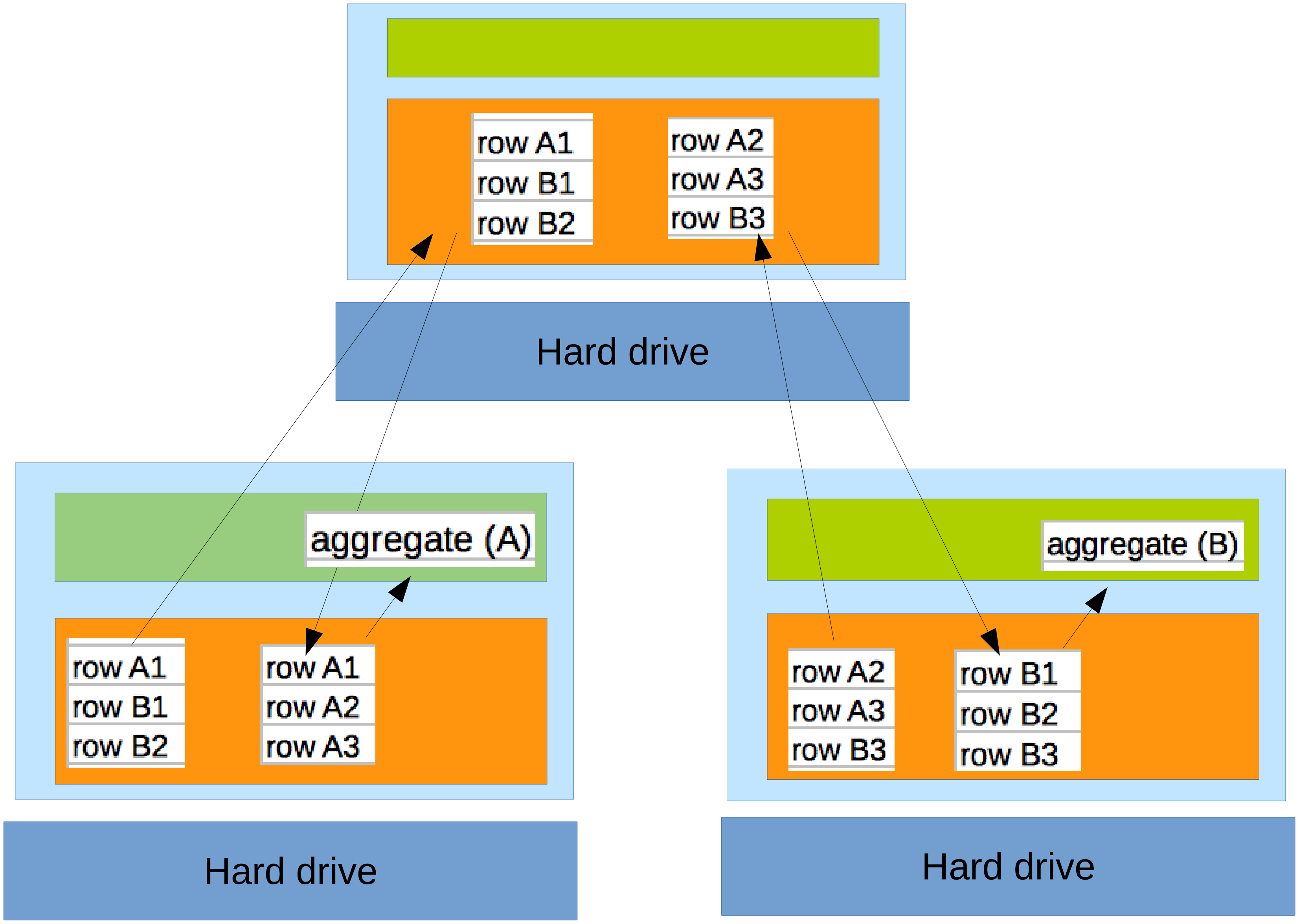

CPU
Memory
Hard drive
function (row of data)
function (row of data)
function (row of data)
...
row of data
row of data
row of data
...
new row of data
new row of data
new row of data
...
Hard drive
function (row of data)
function (row of data)
function (row of data)
...
row of data
row of data
row of data
...
new row of data
new row of data
new row of data
...
Hard drive

```
/* row transformations */
// This UDF (user defined function) takes an integer and converts it into a float that can be used for time calculations
val convertTime = udf((time: Integer) =>
  if (time > 100) time.toString().dropRight(2).toString().toFloat + time.toString().takeRight(2).toFloat/60
  else time.toString().toFloat/60
  )                                         //> convertTime  : org.apache.spark.sql.UserDefinedFunction = UserDefinedFuncti
                                            //| on(<function1>,FloatType,List(IntegerType))

val aprilDF1 = aprilDF.withColumn("CRS_Dep_Time_f",convertTime(aprilDF("CRS_DEP_TIME")))
                                            //> aprilDF1  : org.apache.spark.sql.DataFrame = [QUARTER: int, MONTH: int, DAY
                                            //| _OF_WEEK: int, FL_DATE: string, AIRLINE_ID: int, CARRIER: string, TAIL_NUM:
                                            //|  string, FL_NUM: int, ORIGIN_AIRPORT_ID: int, ORIGIN: string, DEST_AIRPORT_
                                            //| ID: int, DEST: string, CRS_DEP_TIME: int, DEP_TIME: int, TAXI_OUT: double,
                                            //| WHEELS_OFF: int, WHEELS_ON: int, TAXI_IN: double, CRS_ARR_TIME: int, ARR_TI
                                            //| ME: int, DISTANCE: double, : string, CRS_Dep_Time_f: float]

aprilDF1.select("CRS_DEP_TIME","CRS_Dep_Time_f").take(5)
                                            //> res0: Array[org.apache.spark.sql.Row] = Array([900,9.0], [900,9.0], [900,9.
                                            //| 0], [900,9.0], [900,9.0])
```

The Shuffle

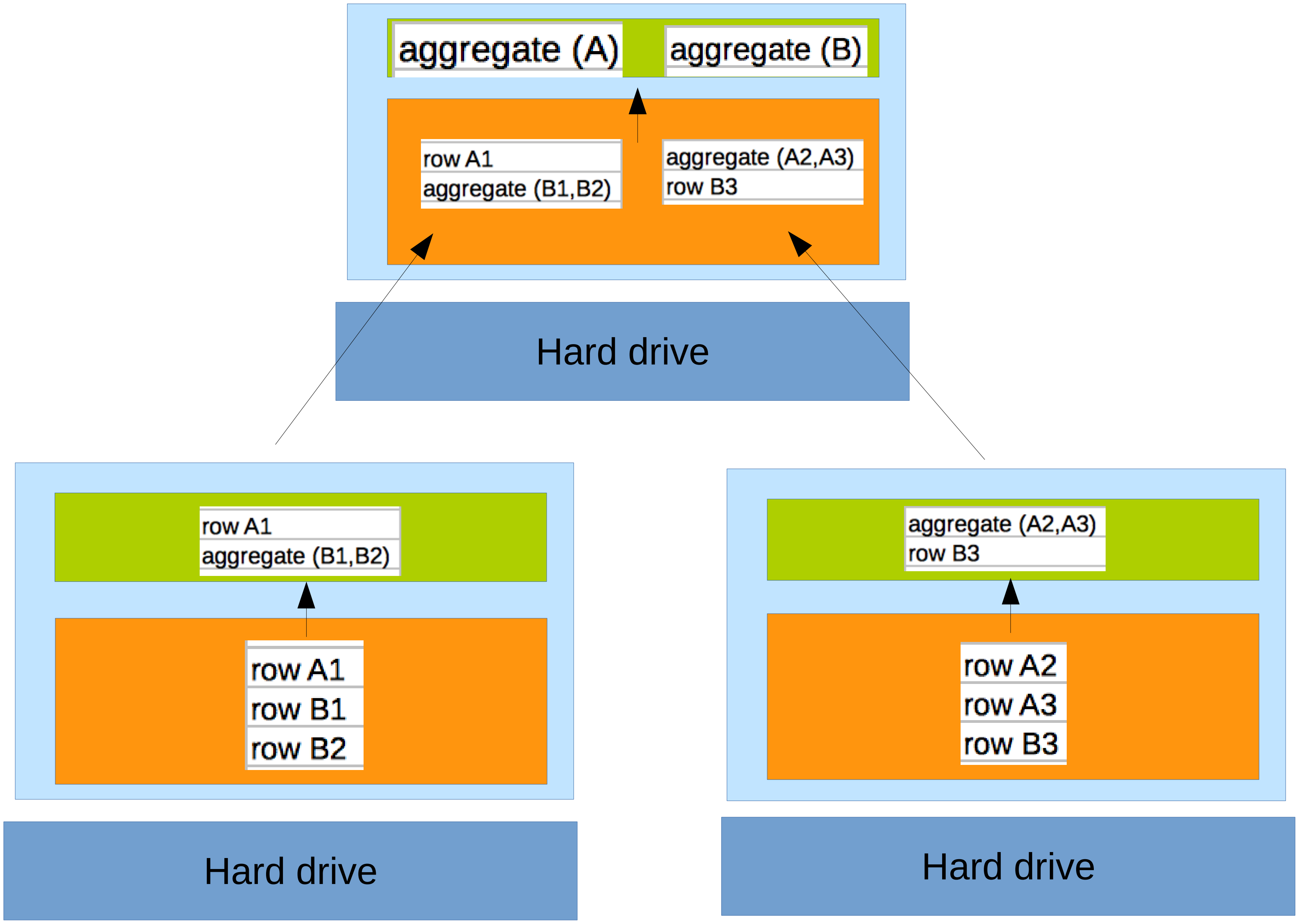

row A1
row B1
row B2
row A2
row A3
row B3
Hard drive
aggregate (A)
row A1
row B1
row B2
row A1
row A2
row A3
Hard drive
aggregate (B)
row A2
row A3
row B3
row B1
row B2
row B3
Hard drive

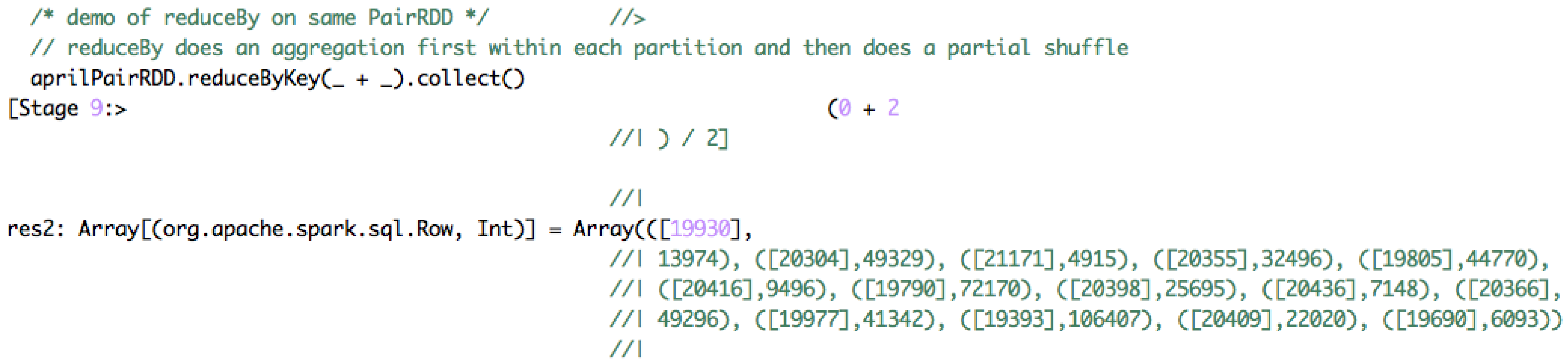
```
  /* demo of groupBy on PairRDDs */
  // create PairRDD of AIRLINE_ID and count (1)
  val aprilPairRDD = aprilDF.select("AIRLINE_ID").map(id => (id,1))
                                                //> aprilPairRDD  : org.apache.spark.rdd.RDD[(org.apache.spark.sql.Row, Int)] =
                                                //|  MapPartitionsRDD[30] at map at Spark4DS.developDemo.scala:50

  // do a groupBy on PairRDD (causes full shuffle)
  aprilPairRDD.groupByKey().map(t => (t._1, t._2.sum)).collect()
                                                //>
[Stage 7:>                                                        (0 + 2
                                                //| ) / 2]
[Stage 7:=============================>
                                                //| (1 + 1) / 2]
[Stage 8:>
                                                //|      (0 + 2) / 2]
[Stage 8:=============================>
                                                //|           (1 + 1) / 2]

                                                //|
res1: Array[(org.apache.spark.sql.Row, Int)]
                                                //|  = Array(([19930],13974), ([20304],49329), ([21171],4915), ([20355],32496),
                                                //|  ([19805],44770), ([20416],9496), ([19790],72170), ([20398],25695), ([20436
                                                //| ],7148), ([20366],49296), ([19977],41342), ([19393],106407), ([20409],22020
                                                //| ), ([19690],6093))
```

(Semi) Aggregate, then Shuffle

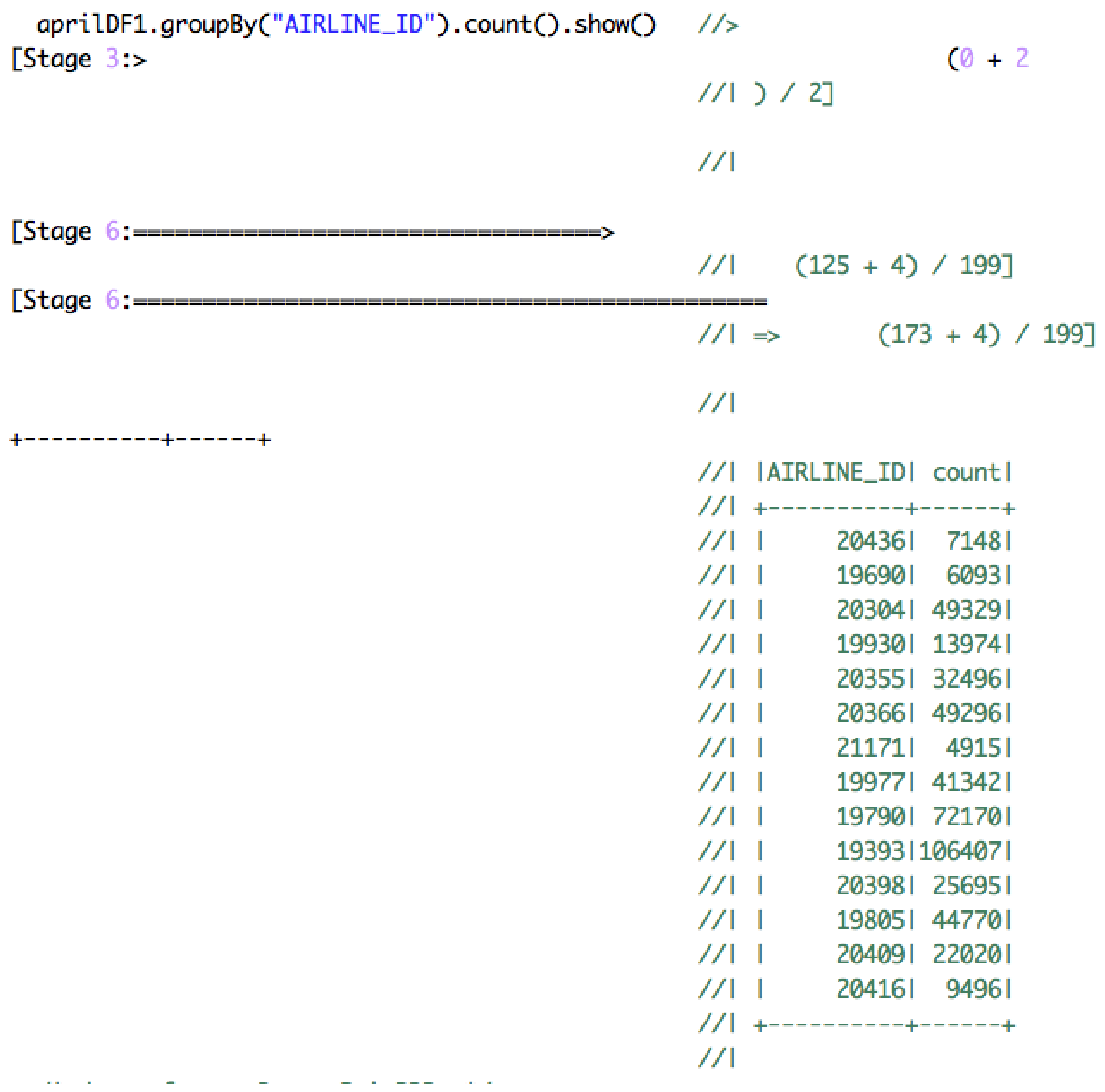




```
  /* demo of reduceBy on same PairRDD */          //>
  // reduceBy does an aggregation first within each partition and then does a partial shuffle
  aprilPairRDD.reduceByKey(_ + _).collect()
[Stage 9:>                                                         (0 + 2
                                                //| ) / 2]

                                                //|
res2: Array[(org.apache.spark.sql.Row, Int)] = Array(([19930],
                                                //| 13974), ([20304],49329), ([21171],4915), ([20355],32496), ([19805],44770),
                                                //| ([20416],9496), ([19790],72170), ([20398],25695), ([20436],7148), ([20366],
                                                //| 49296), ([19977],41342), ([19393],106407), ([20409],22020), ([19690],6093))
                                                //|
```

Spark DF GroupBy



```
  aprilDF1.groupBy("AIRLINE_ID").count().show()   //>
[Stage 3:>                                                          (0 + 2
                                                  //| ) / 2]

                                                  //|


[Stage 6:==================================>
                                                  //|    (125 + 4) / 199]
[Stage 6:==============================================
                                                  //| =>        (173 + 4) / 199]

                                                  //|


+----------+------+
                                                  //| |AIRLINE_ID| count|
                                                  //| +----------+------+
                                                  //| |     20436|  7148|
                                                  //| |     19690|  6093|
                                                  //| |     20304| 49329|
                                                  //| |     19930| 13974|
                                                  //| |     20355| 32496|
                                                  //| |     20366| 49296|
                                                  //| |     21171|  4915|
                                                  //| |     19977| 41342|
                                                  //| |     19790| 72170|
                                                  //| |     19393|106407|
                                                  //| |     20398| 25695|
                                                  //| |     19805| 44770|
                                                  //| |     20409| 22020|
                                                  //| |     20416|  9496|
                                                  //| +----------+------+
                                                  //|
```

Summary: Spark for Data Scientists

- Think
  - Scala Spark
  - Hardware AND software!
- Next steps for airline data analysis
  - Are there factors that indicate a propensity for being late?
    - What is a good metric for lateness?
    - What other derived variables can you create (in a performance optimized way!) to help your analysis?
  - Can you identify airline hubs using the data?
    - GraphX and connected components?

Appendix

- Build Spark with Scala 2.11
- Eclipse with Scala
- Scala Worksheets
  - New worksheet
  - Worksheet settings
- Build using SBT

Build Spark with Scala 2.11

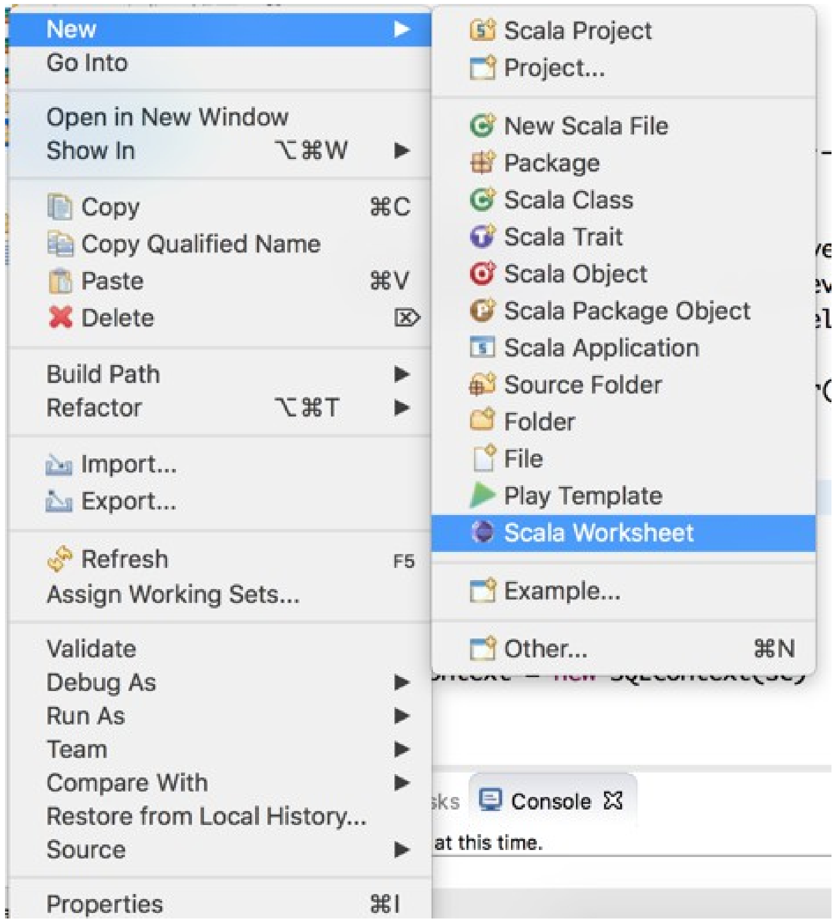
- Instructions developed for MacBook Pro running OS X El Capitan 10.11.2
- Download source file from from http://spark.apache.org/downloads.html
  - Should get a file called spark-version.tgz
- Move file to /usr/local/
  - mv /Users/username/Documents/spark-1.6.0.tgz /usr/local/
- Unpack
  - tar -xf spark-1.6.0.tgz
- Delete zipped file
  - rm spark-1.6.0.tgz
- Change to spark directory
  - cd spark-1.6.0
- Explicitly set Scala version (May or may not need to do this step)
  - ./dev/change-scala-version.sh 2.11
- Build using maven (also adding in Yarn, Hadoop (2.6.0), and Hive)
  - mvn -Pyarn -Phadoop-2.6 -Dhadoop.version=2.6.0 -Phive -Phive-thriftserver -Dscala-2.11 -DskipTests clean package
- Run Spark from this folder (or set alias to this folder)
  - bin/run-example SparkPi 10
  - bin/spark-shell
  - bin/spark-submit

Eclipse with Scala

- Download from here
  - http://scala-ide.org/download/sdk.html
  - Need Scala 2.11.7 version for worksheets to work with Spark!
  - If you already have the Java version of Eclipse & want to use the Scala plugin…
    - Good luck. Scala Worksheets & some other functionality was really buggy the last time I tried it (2014), but maybe it's grown up since then… I just use two separate Eclipse apps (one for Scala & one for Java)
- IDE installation should include Scala & SBT
  - See notes on Building using SBT page for working with Eclipse from SBT

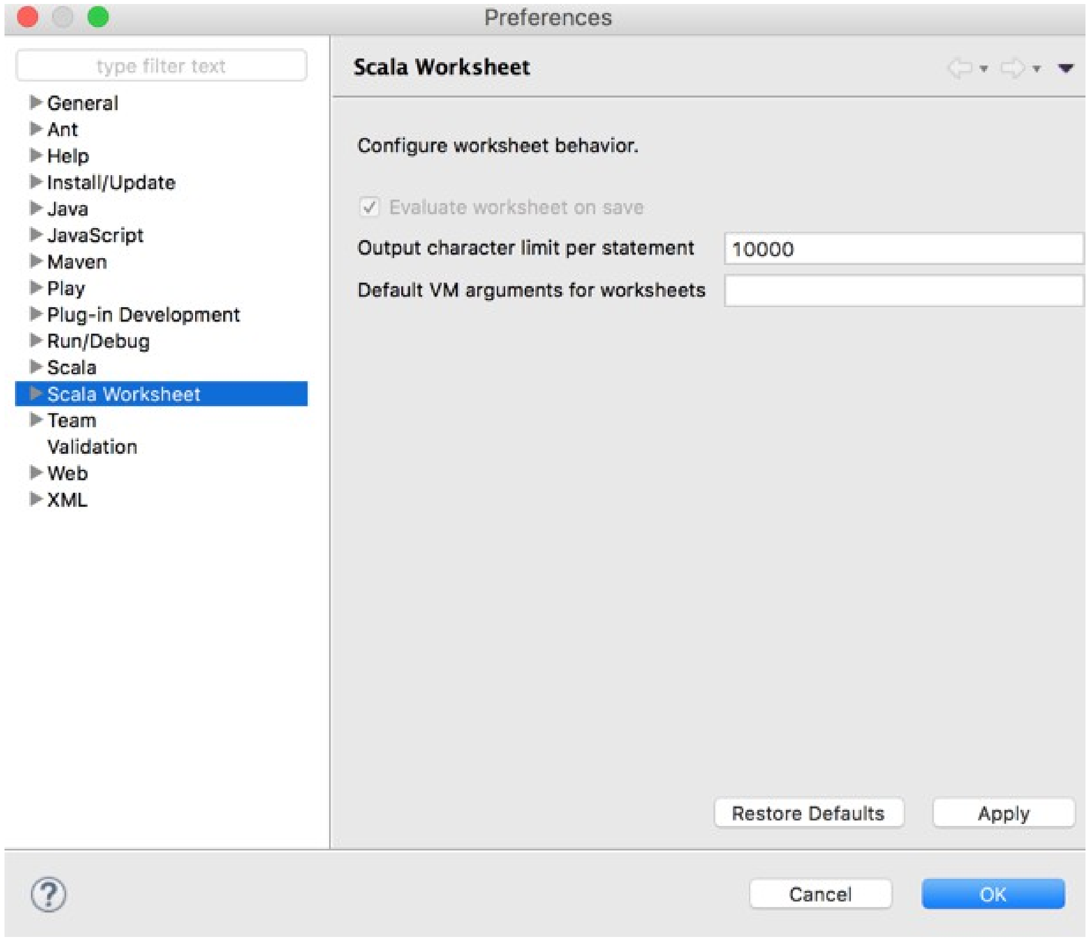
Create New Scala Worksheet

- Create new Scala project
- Create new Scala worksheet in project



Scala Worksheet Settings

- Scala IDE/Preferences
  - Evaluate worksheet on save
  - Output character limit



Scala Worksheet

Configure worksheet behavior.

✓ Evaluate worksheet on save

Output character limit per statement: 10000

Default VM arguments for worksheets:

Build using SBT

- Need build.sbt in your project folder
  - See example build.sbt in git repo
- Need plugins in sbt plugin folder (for Eclipse & assembly)
  - .sbt/0.13/plugins/plugins.sbt
  - See example plugins.sbt in git repo
- Creating new projects
  - Directly in Eclipse
  - Manual creation and/or cloning git repos
    - cd to project folder
    - Create files needed by eclipse by typing following command
      - sbt eclipse
- Building executable jar
  - cd to project folder
  - sbt
    - clean
    - (compile)
    - assembly
  - Executable jar will be in project/target/Scala folder