Big Data Analysis
- Part III : Data Analysis with Spark on Hadoop

May 04, 2017

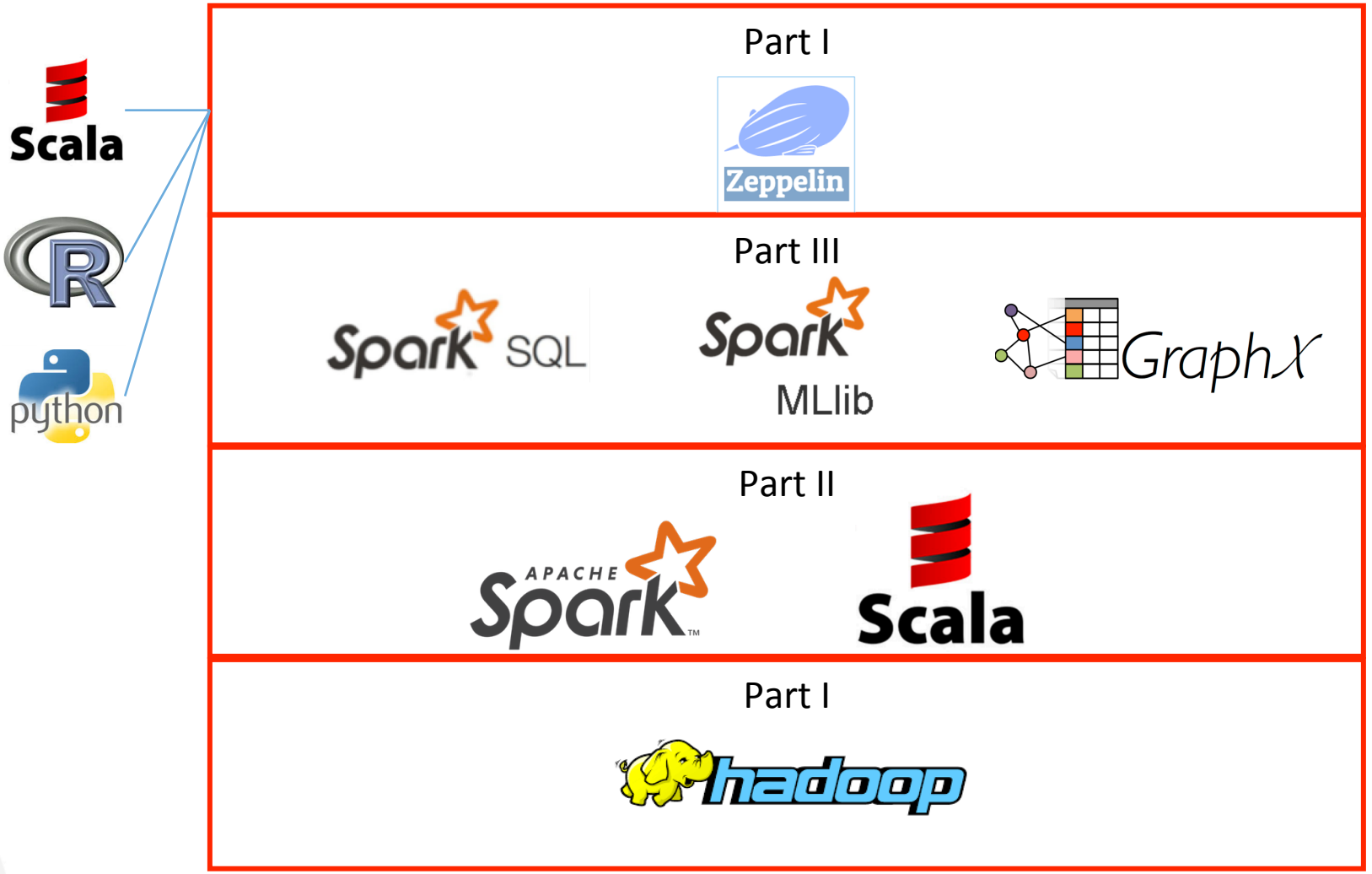
Dr. Weijia Xu (Group Manager)
Dr. Zhao Zhang
Data Mining & Statistics Group
Texas Advanced Computing Center (TACC)
The University of Texas at Austin

TACC

Big Data Training Series at TACC

- Introduction to Hadoop and Spark On Wrangler
- Introduction to programming with Scala/Spark
- Data Analysis Using Hadoop/Spark - May 4, 2017
  - 1:00-2:00 Dataframe and SparkSQL
  - 2:00-3:00 Data analysis with MLlib and Graphx
  - 3:00-4:00 Spark internal
  - 4:00-4:30 Hands on

About this series



Today's Agenda

1:00 – 1:50 Working with Spark Dataframe and SparkSQL

- Zeppelin Revisited
- Spark Dataframe
- Getting data in and out with Spark
- Working with Dataframe using SparkSQL

2:00 – 2:50 Analysis with MLlib and graphX

- Machine learning with MLlib
- Clustering with MLlib
- Recommendation system with Mlib
- Graph analysis example with graphX

3:00 – 3:50 Spark Internal and configuration

- Parallelism in Spark
- Memory management in Spark

4:00 – 4:30 Hands-on exercise

Zeppelin Revisited

- Start using sbatch command
  - Command syntax:

```
sbatch --reservation RESERVATION_NAME
       -A PROJECT_NAME
       JOB_SCRIPT
```

- For today's training session

```
sbatch --reservation hadoop+TRAINING-HPC+2188
       -A TRAINING-HPC
/data/apps/.zeppelin/job.zeppelin.work
```

- Let's wait and come back to zeppelin later.

Data APIs in Spark

- RDD
  - Resilient Distributed Dataset
  - Implemented from the beginning of Spark framework
  - Support a number of transformation functions, e.g.
    - Map, rdd.map(x => x*2)
    - reduce, *rdd.reduce(_+_)*
    - filter, *rdd.filter(_%3 ==0)*
    - …
  - Think it as a set of objects stored in memory across nodes, e.g.

    val rdd = sc.parallelize(0 until 10000)

Data APIs in Spark

- DataFrame
  - Since Spark 1.3
  - An abstract API built on top of RDD.
  - Have schema to describe the data.
  - Can use off-heap storage for large data.
  - Think it as a table stored across data nodes
- Dataset
  - Since Spark 1.6
  - An API to combine advantages of both RDD and DataFrame

- As of Spark 2.X, Dataset and DataFrame APIs are merged.

  DataFrame is a Dataset[Row]

Working with File in Spark

- One of the advantage of the DataFrame is easily read structured data file. e.g. reading a csv file
  - val df = spark.read.format(“csv”)
    .options(“header”, true)
    .load(“/tmp/data/mtcars.csv”)
- The results can easily be shown as a table with show

```
val df=spark.read.format("csv").option("header", true).load("/tmp/data/mtcars.csv")
df.show()
```

```
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|             model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|         Mazda RX4|  21|  6|  160|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
|     Mazda RX4 Wag|  21|  6|  160|110| 3.9|2.875|17.02|  0|  1|   4|   4|
|        Datsun 710|22.8|  4|  108| 93|3.85| 2.32|18.61|  1|  1|   4|   1|
|    Hornet 4 Drive|21.4|  6|  258|110|3.08|3.215|19.44|  1|  0|   3|   1|
| Hornet Sportabout|18.7|  8|  360|175|3.15| 3.44|17.02|  0|  0|   3|   2|
|           Valiant|18.1|  6|  225|105|2.76| 3.46|20.22|  1|  0|   3|   1|
|        Duster 360|14.3|  8|  360|245|3.21| 3.57|15.84|  0|  0|   3|   4|
```

Working with File in Spark

- Several common format can be easily read and write to/from DataFrame
  - text
  - JSON
  - parquet
  - ORC
  - JDBC
  - …
- Parquet and ORC
  - Columnar store file.
  - Data are compressed and stored as binary, could be 60~80% smaller than text file format.
  - Store the data schema as well.

Read and Write Files with DataFrame

- Write as a JSON file

```
df.write.json("cars.json")
```

- Write as a Parquet file

```
df.write.parquet("cars.parquet")
```

- Write as a delimited file

```
df.write.option("delimiter","\t").csv("cars.tab")
```

- Load from JSON file

```
val df_json = spark.read.json("cars.json")
```

- Load from Parquet file

```
val df_parquet = spark.read.parquet("cars.parquet")
```

Notes about I/O with Spark

- Default file system
  - When use with hadoop cluster mode, the default file system is inside hdfs. e.g.

    "/tmp/data" refers path within the hdfs
  - When use in local mode, the default is the local file system.

  - To explicitly specify file system uses prefix file:/// or hdfs:/// with the path
- e.g.
  - read from Linux file system

```
val df=spark.read.format("csv")
            .option("header", true)
            .load("file:///work/00791/xwj/DMS/R-training/RDataMining/mtcars.csv")
```

  - read from hdfs file system

```
val df=spark.read.format("csv")
            .option("header", true)
            .load("/tmp/data/mtcars.csv")
```

Notes about I/O with Spark

- Path to File and Path to Folder
  - Both file name and directory path can be used as variable for reading.
  - Will automatically read all the files within the directory when just path to directory is given
  - The output path is always treated as directory

```
hadoop fs -ls
drwxr-xr-x   - xwj hadoop          0 2017-05-03 23:46 cars.json
drwxr-xr-x   - xwj hadoop          0 2017-05-03 23:46 cars.parquet
drwxr-xr-x   - xwj hadoop          0 2017-05-04 00:02 cars.tab
```

- Save Modes
  - Default mode will report error when save to existing data files
  - Can change to different mode with *mode()* function with parameters

error, append, overwrite, ignore

RDD vs. DataFrame

- From DataFrame to RDD with rdd function, e.g.

```
val car_rdd = df.rdd
```

```
car_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitio
res87: Array[org.apache.spark.sql.Row] = Array([Mazda RX4,21,6,160,110,3.
258,110,3.08,3.215,19.44,1,0,3,1], [Hornet Sportabout,18.7,8,360,175,3.15
3.69,3.19,20,1,0,4,2], [Merc 230,22.8,4,140.8,95,3.92,3.15,22.9,1,0,4,2],
0,3,3], [Merc 450SL,17.3,8,275.8,180,3.07,3.73,17.6,0,0,3,3], [Merc 450SL
```

- From RDD to DataFrame with toDF

```
val rdd = sc.parallelize(0 until 10000)
val rdd_df = rdd.toDF
rdd_df.show
```

```
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[142] at p
rdd_df: org.apache.spark.sql.DataFrame = [value: int]
+-----+
|value|
+-----+
|    0|
|    1|
|    2|
```

Convert RDD[Object] to DataFrame

```
case class Person(id: Int, name: String)
val person = sc.parallelize(Seq( Person(1, "Mike"),
                                 Person(2, "Smith"),
                                 Person(3, "Brooke")))
val person_df = person.toDF
person_df.show
```

```
defined class Person
person: org.apache.spark.rdd.RDD[Person] = ParallelCollectionRDD[160] c
person_df: org.apache.spark.sql.DataFrame = [id: int, name: string]
+---+------+
| id|  name|
+---+------+
|  1|  Mike|
|  2| Smith|
|  3|Brooke|
+---+------+
```

Common Functions with DataFrame

- df.show(int n)
  - Display a number of rows from the DataFrame
  - The default value will only show 20 rows.
- df.printSchema
  - Display schema of the DataFrame in the stdout
- df.describe(*cols*)
  - Return a DataFrame with summary statistics of the selected columns.
  - Default will run on all columns.

```
df.describe("mpg").show
```

```
+-------+------------------+
|summary|               mpg|
+-------+------------------+
|  count|                32|
|   mean|20.090624999999996|
| stddev| 6.026948052089103|
|    min|              10.4|
|    max|              33.9|
+-------+------------------+
```

```
df.printSchema
```

```
root
 |-- model: string (nullable = true)
 |-- mpg: string (nullable = true)
 |-- cyl: string (nullable = true)
 |-- disp: string (nullable = true)
 |-- hp: string (nullable = true)
 |-- drat: string (nullable = true)
 |-- wt: string (nullable = true)
 |-- qsec: string (nullable = true)
 |-- vs: string (nullable = true)
 |-- am: string (nullable = true)
 |-- gear: string (nullable = true)
 |-- carb: string (nullable = true)
```

RDD vs Dataframe

Data Frame supports use of “column name”.

filter example

```
val car_rdd = df.rdd
car_rdd.filter(_(1).asInstanceOf[String].toDouble > 20).collect
```

```
car_rdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartit
ionsRDD[58] at rdd at <console>:27
res100: Array[org.apache.spark.sql.Row] = Array([Mazda RX4,21,6,160,110
,3.9,2.62,16.46,0,1,4,4], [Mazda RX4 Wag,21,6,160,110,3.9,2.875,17.02,0
,1,4,4], [Datsun 710,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1], [Hornet 4
Drive,21.4,6,258,110,3.08,3.215,19.44,1,0,3,1], [Merc 240D,24.4,4,146.7
,62,3.69,3.19,20,1,0,4,2], [Merc 230,22.8,4,140.8,95,3.92,3.15,22.9,1,0
,4,2], [Fiat 128,32.4,4,78.7,66,4.08,2.2,19.47,1,1,4,1], [Honda Civic,3
0.4,4,75.7,52,4.93,1.615,18.52,1,1,4,2], [Toyota Corolla,33.9,4,71.1,65
,4.22,1.835,19.9,1,1,4,1], [Toyota Corona,21.5,4,120.1,97,3.7,2.465,20.
01,1,0,3,1], [Fiat X1-9,27.3,4,79,66,4.08,1.935,18.9,1,1,4,1], [Porsche
914-2,26,4,120.3,91,4.43,2.14,16.7,0,1,5,2], [Lotus Europa,30.4,4,95.1,
113,3.77,1.513,16.9,1,1,5,2], [Volvo 142E,21.4,4,121,109,4.11,2.78,18.6
,1,1,4,2])
```

```
df.filter("mpg>20").show
```

```
+--------------+----+---+-----+---+---
|         model| mpg|cyl| disp| hp|dra
+--------------+----+---+-----+---+---
|     Mazda RX4|  21|  6|  160|110| 3.
| Mazda RX4 Wag|  21|  6|  160|110| 3.
|    Datsun 710|22.8|  4|  108| 93|3.8
|Hornet 4 Drive|21.4|  6|  258|110|3.0
|     Merc 240D|24.4|  4|146.7| 62|3.6
|      Merc 230|22.8|  4|140.8| 95|3.9
|      Fiat 128|32.4|  4| 78.7| 66|4.0
|   Honda Civic|30.4|  4| 75.7| 52|4.9
|Toyota Corolla|33.9|  4| 71.1| 65|4.2
| Toyota Corona|21.5|  4|120.1| 97| 3.
|     Fiat X1-9|27.3|  4|   79| 66|4.0
| Porsche 914-2|  26|  4|120.3| 91|4.4
|  Lotus Europa|30.4|  4| 95.1|113|3.7
|    Volvo 142E|21.4|  4|  121|109|4.1
+--------------+----+---+-----+---+---
```

Dataset and DataFrame

- DataFrame and Dataset shares the same API
- DataFrame is a specific types of Dataset[row]
- Most functions working with RDD will also work with Dataset, some may not work directly with DataFrame due to type conversion

```
val rdd = sc.parallelize(0 until 100)
val rdd_df =rdd.toDF
val rdd_ds = rdd.toDS

rdd.filter (_ < 10).collect
rdd_df.filter("value < 10").show
rdd_ds.filter("value < 10").show
rdd_ds.filter(_ < 10).show

rdd.map(_ * 2).collect
rdd_ds.map(_ * 2).show
rdd_df.map(_ * 2).show //this line will fail
rdd_df.select('value * 2).show
```

Spark SQL

- A Structured Query Language Syntax.
- It follows HiveQL syntax. Similar to standard database SQL but not identical.
- Can be used from command line, spark-shell, pyspark, sparkR shell or over JDBC drive
- Can be used in the form of function call or as SQL statement

Basic Spark SQL Support Functions

Select

```
df.select("model","mpg", "wt").show
```

```
+------------------+----+-----+
|             model| mpg|   wt|
+------------------+----+-----+
|         Mazda RX4|  21| 2.62|
|     Mazda RX4 Wag|  21|2.875|
|        Datsun 710|22.8| 2.32|
|    Hornet 4 Drive|21.4|3.215|
| Hornet Sportabout|18.7| 3.44|
|           Valiant|18.1| 3.46|
|        Duster 360|14.3| 3.57|
|         Merc 240D|24.4| 3.19|
|          Merc 230|22.8| 3.15|
|          Merc 280|19.2| 3.44|
|         Merc 280C|17.8| 3.44|
|        Merc 450SE|16.4| 4.07|
|        Merc 450SL|17.3| 3.73|
|       Merc 450SLC|15.2| 3.78|
| Cadillac Fleetwood|10.4| 5.25|
```

```
df.select($"model", $"mpg" * 1.6).show
```

```
+------------------+------------------+
|             model|       (mpg * 1.6)|
+------------------+------------------+
|         Mazda RX4|              33.6|
|     Mazda RX4 Wag|              33.6|
|        Datsun 710|36.480000000000004|
|    Hornet 4 Drive|             34.24|
| Hornet Sportabout|             29.92|
|           Valiant|28.960000000000004|
|        Duster 360|22.880000000000003|
|         Merc 240D|             39.04|
|          Merc 230|36.480000000000004|
|          Merc 280|             30.72|
|         Merc 280C|28.480000000000004|
|        Merc 450SE|             26.24|
|        Merc 450SL|27.680000000000003|
|       Merc 450SLC|             24.32|
```

Basic Spark SQL Support Function

```
df.select("model","mpg", "wt", "cyl")
  .filter("cyl > 4").show
```

```
+------------------+----+-----+---+
|             model| mpg|   wt|cyl|
+------------------+----+-----+---+
|         Mazda RX4|  21| 2.62|  6|
|     Mazda RX4 Wag|  21|2.875|  6|
|    Hornet 4 Drive|21.4|3.215|  6|
| Hornet Sportabout|18.7| 3.44|  8|
|           Valiant|18.1| 3.46|  6|
|        Duster 360|14.3| 3.57|  8|
|         Merc 280|19.2| 3.44|  6|
|        Merc 280C|17.8| 3.44|  6|
|       Merc 450SE|16.4| 4.07|  8|
|       Merc 450SL|17.3| 3.73|  8|
|      Merc 450SLC|15.2| 3.78|  8|
|Cadillac Fleetwood|10.4| 5.25|  8|
|Lincoln Continental|10.4|5.424|  8|
| Chrysler Imperial|14.7|5.345|  8|
```

```
df.groupBy("cyl").count().show
```

```
+---+-----+
|cyl|count|
+---+-----+
|  8|   14|
|  6|    7|
|  4|   11|
+---+-----+
```

Running SQL Statement

- Register the DataFrame as a table
- Using spark.sql(SQL_STATEMENT)

```
df.createOrReplaceTempView("cars")
spark.sql("SELECT * FROM cars WHERE cyl = 6")
     .show
```

```
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
|         model|mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
|     Mazda RX4|  21|  6|  160|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
| Mazda RX4 Wag|  21|  6|  160|110| 3.9|2.875|17.02|  0|  1|   4|   4|
|Hornet 4 Drive|21.4|  6|  258|110|3.08|3.215|19.44|  1|  0|   3|   1|
|       Valiant|18.1|  6|  225|105|2.76| 3.46|20.22|  1|  0|   3|   1|
|      Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4|
|     Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4|
|  Ferrari Dino|19.7|  6|  145|175|3.62| 2.77| 15.5|  0|  1|   5|   6|
+--------------+---+---+-----+---+----+-----+-----+---+---+----+----+
```

Basic SQL Syntax

SELECT col1, col2, ….

FROM table1, table2, …

[WHERE condition1, AND|OR, condtion2 ….]

[GROUPBY col1,…]

[ORDERBY col1,…]

```
spark.sql("SELECT cyl, avg(mpg) as avg_mpg, count(1) as count "+
          "from cars "+
          "group by cyl")
     .show
+---+------------------+-----+
|cyl|           avg_mpg|count|
+---+------------------+-----+
|  8|15.100000000000003|   14|
|  6| 19.74285714285714|    7|
|  4|26.663636363636364|   11|
+---+------------------+-----+
```

Complex Column Types Support

- Array: *ARRAY<data type>*
  - A list of values of the data type. e.g.
    names array<string>
- Map: *MAP<data type, data type>*
  - A list of key value pairs
  - Key must be one of primitive types e.g
    pay_grade map<double, string>
- Struct: STRUCT <name1: data type, name2, data type>
  - A list of values of same or different types
  - Each field can have a name specified. e.g.
    address struct(street : string, City : string, State : string, Zip : int)
- UnionType UNIONTYPE<data type, data type, data type>
  - A list of potential data types
  - Only one type can be used at a given time. e.g.
    on_leave uniontype< float, string, Boolean>

Use Dataframe and SparkSQL in the analysis

```
val cars = spark.read.format("csv").option("header", true).load("/tmp/data/mtcars.csv")
              .selectExpr("model", "mpg + 0.0 as mpg", "disp + 0.0 as disp",
                          "hp + 0.0 as hp", "drat + 0.0 as drat", "wt + 0.0 as wt",
                          "cyl + 0.0 as label")

val training= cars.sample(false, 0.8)
val test= cars.except(training)

val assembler = new VectorAssembler()
  .setInputCols(Array("mpg", "disp", "hp", "drat", "wt"))
  .setOutputCol("features")

val lr = new LogisticRegression()
  .setMaxIter(10)
  .setRegParam(0.2)
  .setElasticNetParam(0.0)

val pipeline = new Pipeline().setStages(Array(assembler, lr))
val lrModel = pipeline.fit(training)

val result = lrModel.transform(test).select('model, 'label, 'prediction)
result.show
```

Prepare datasets

Assemble feature vectors

Define analysis

Run analysis

See results

Back to Zeppelin

To get the web URL of your Zeppelin Session:

tail zeppelin.out

Then you should see something like

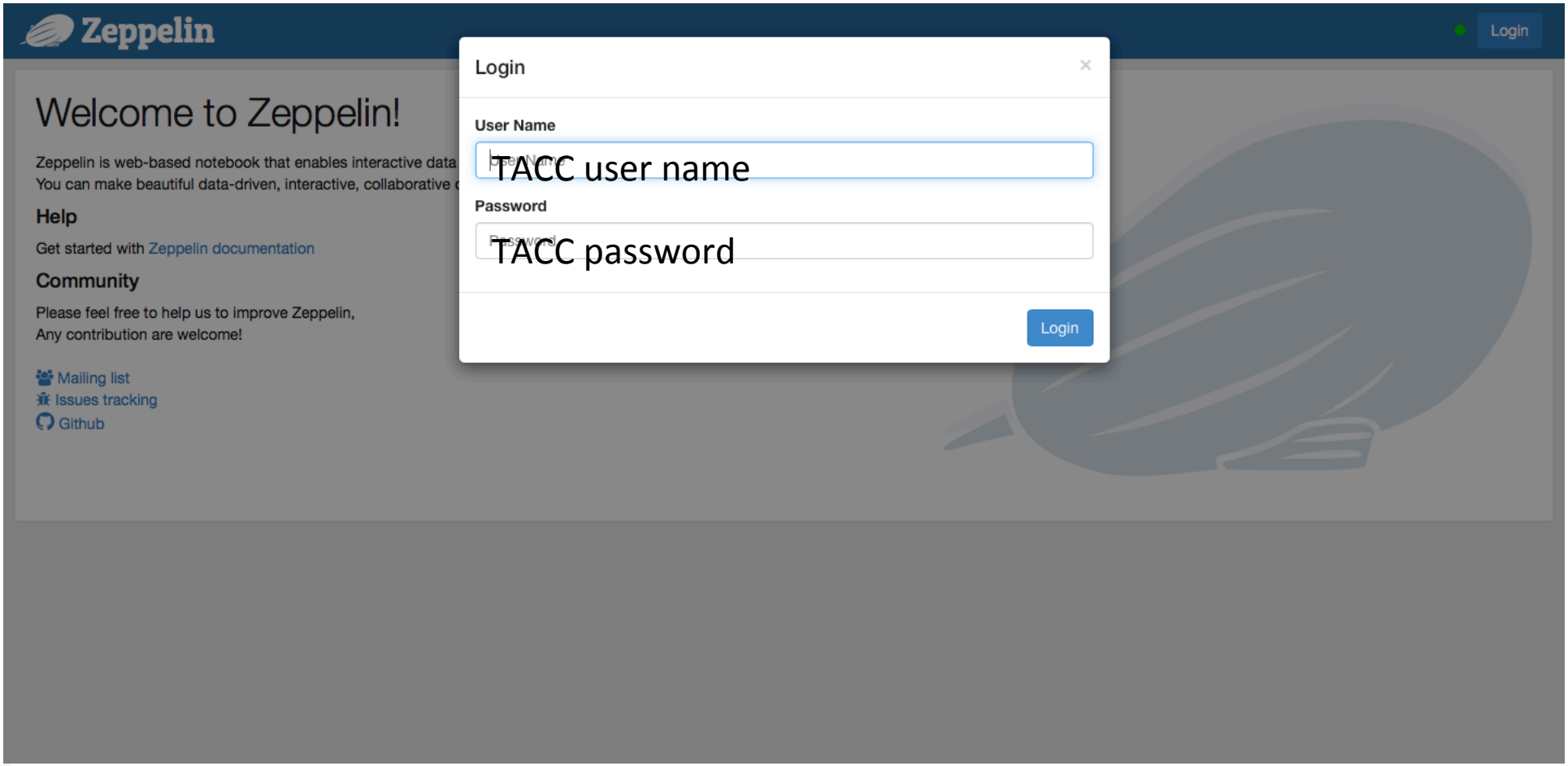
```
Your applicatin  is now running!
Application UI is at http://wrangler.tacc.utexas.edu:52644
```

Then open a browser to the URL:

Note: it seems Chrome some time have problem with Zeppelin. Safari seems working all the time.

Once the application is running

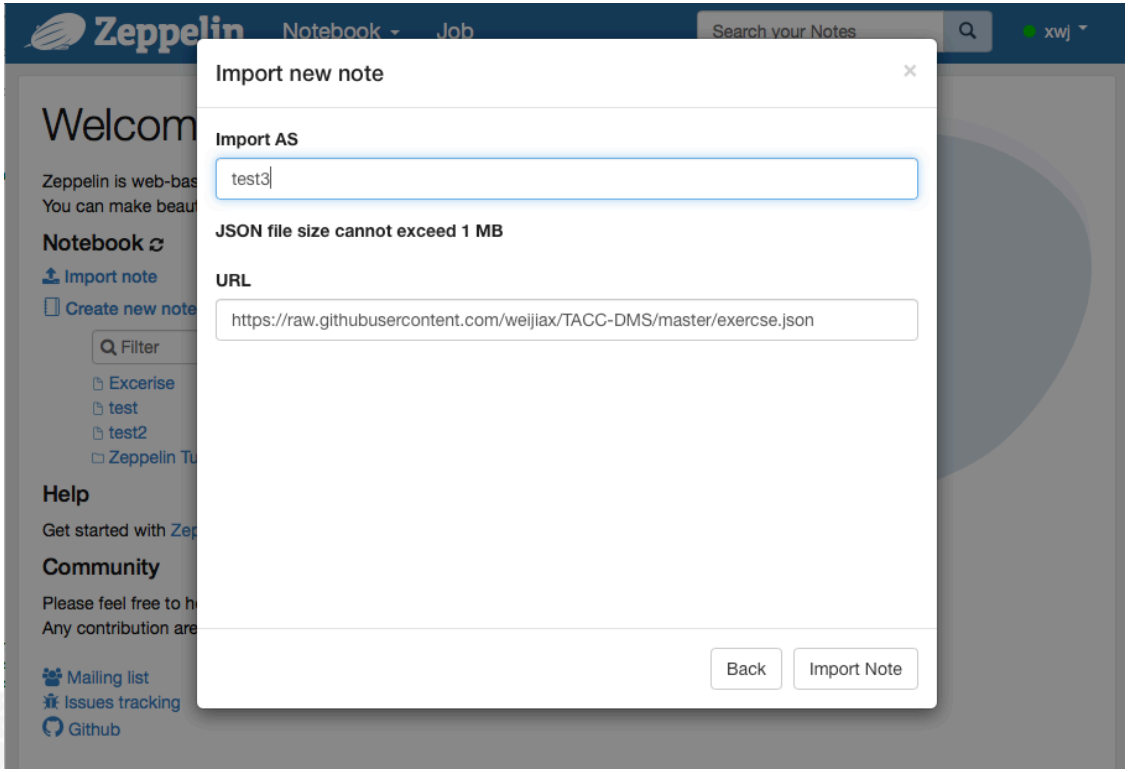
Now you can login with your TACC credential



Create or Add Note

- After you logon you can either create or import a new note.
- You can import an exercise note from URL directly

https://raw.githubusercontent.com/weijiax/TACC-DMS/master/dataframebasic.json



Using Zeppelin

Zeppelin has some nice feature built-in to work with data frame. Such as display a DataFrame better with

z.show

```
z.show(df.describe())
```

summary	model	mpg	cyl	disp	hp
count	32	32	32	32	32
mean	null	20.09062	6.1875	230.72187500000004	146.6875
stddev	null	6.02695	1.7859216469465444	123.93869383138195	68.5628684893205
min	AMC Javelin	10.4	4	108	105
max	Volvo 142E	33.9	8	95.1	97

Using Zeppelin

In addition to show a DataFrame in the table format, Zeppelin also provide a few charting options.

We will continue at 2:05

To use zeppelin on the reservation

sbatch --reservation hadoop+TRAINING-HPC+2188
-A TRAINING-HPC
/data/apps/.zeppelin/job.zeppelin.work

However, due to limited number of nodes in reservation, if you cannot get on try zeppelin without resevation

sbtach /data/apps/.zeppelin/job.zeppelin.local

URL to import note is

https://raw.githubusercontent.com/weijiax/TACC-DMS/master/dataframebasic.json

Or just access the text version of all script

https://github.com/weijiax/TACC-DMS/blob/master/dataframebasic.txt

Big Data Analysis - Part III: Practical Machine Learning with MLlib and GraphX

Zhao Zhang
Data Mining and Statistics Group
Texas Advanced Computing Center
zzhang@tacc.utexas.edu
May 4, 2017

Overview

Machine learning concept

MLlib

GraphX

Machine Learning Concept

Supervised Learning

Unsupervised Learning

Others

Supervised Learning

Supervised learning is the machine learning task of inferring a function from labeled training data. — Wiki

- Linear Regression
- Classification
  - Logistic Regression
  - Support Vector Machine (SVM)

Unsupervised Learning

Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data. — Wiki

- A lower dimension representation (e.g., Principle Component Analysis)
- A sparse representation (e.g., K-Means, Mixture Models)
- An independent representation (e.g., PCA)

Goals

Understand the basics of machine learning algorithms

Use MLlib and GraphX to train models with real datasets

Dataset

A small dataset of San Francisco and New York City real estate data

491 records in total

7 columns (in_sf, beds, bath, price, year_built, sqft, elevation)

Scaled features

Keep in Mind

The algorithm and code examples are for training purposes

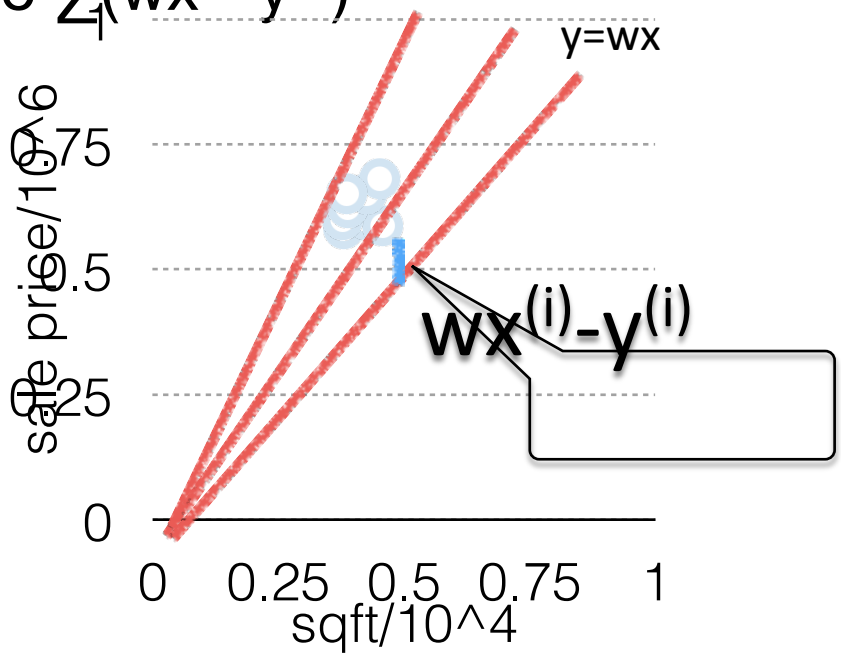
They do not necessarily reflect the performance of the algorithm

Linear Regression

Predicting the house price using sqft

Train a function y=wx to minimize $\sum_i (wx^{(i)}-y^{(i)})^2$

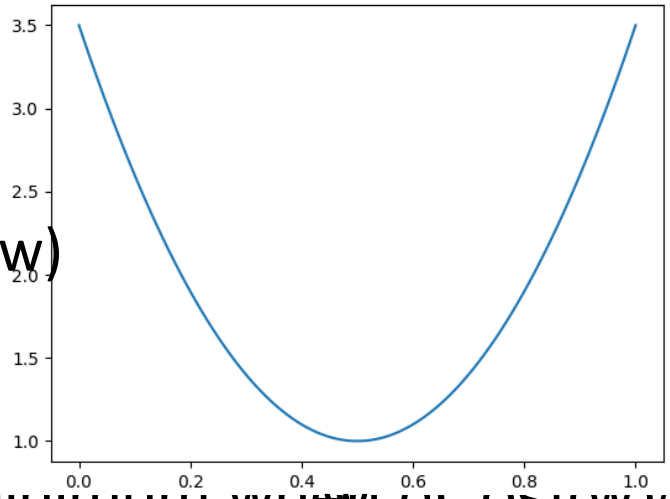
	sqft/10^4	price/10^6
	0.3801	0.58
	0.4271	0.5975
	0.4580	0.588
	0.3780	0.6
	0.3890	0.623
	0.4250	0.65
	0.4500	0.68
	0.3867	0.65
9	0.3815	?



Gradient Descent

$Cost(w) = \sum(wx^{(i)}-y^{(i)})^2$

$= \mathbf{w^2}\sum x^{(i)2} - 2\mathbf{w}y^{(i)}\sum x^{(i)}+\sum y^{(i)2}$



Cost(w)

w

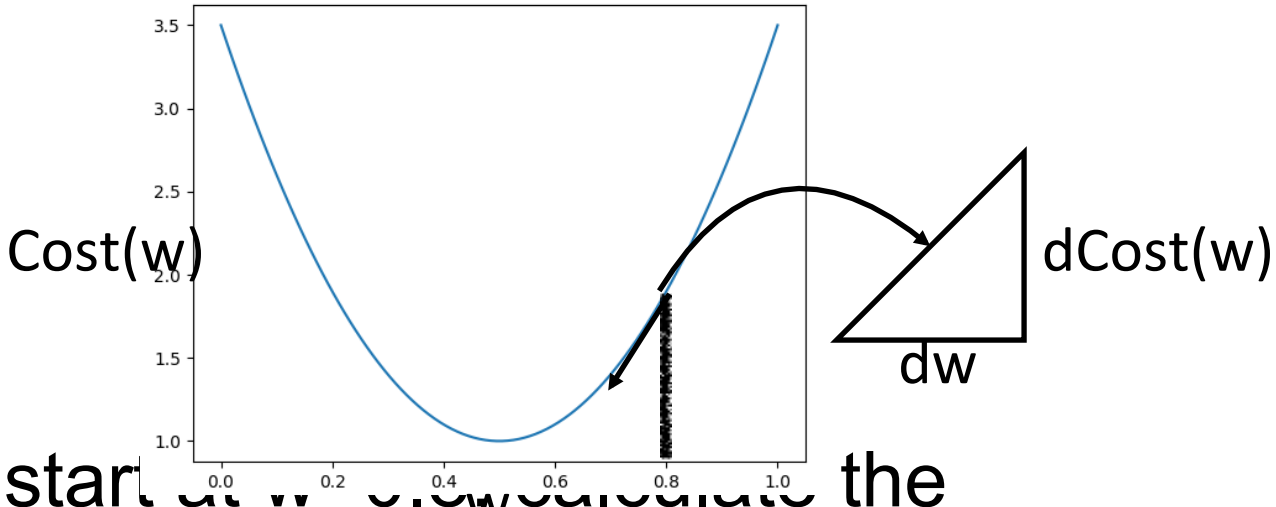
$dCost(w)/dw = 2w$

Cost(w) gets its minimum when $dCost(w)/dw = 0$

Gradient Descent

$$\text{Cost}(w) = \sum(wx^{(i)}-y^{(i)})^2$$

$$= \mathbf{w^2}\sum x^{(i)2} - 2\mathbf{w}y^{(i)}\sum x^{(i)}+\sum y^{(i)2}$$



Randomly start at w = 0.8 and calculate the derivative dCost(w)/dw at w=0.8

Update w = w-αdCost(w)/dw until dCost(w)/dw converges to 0

Using MLlib

```
import org.apache.spark.mllib.linalg.
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(3).toDouble, Vectors.dense(w(5).toDouble))
})
val model = LinearRegressionWithSGD.train(data,100)

model.weights

val trainError = lines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble))-w(3).toDouble
})
val mseTrain = trainError.map(x=>x*x).reduce(_+_)/400
> mseTrain: Double = 0.06472201882476669

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testError = tlines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble))-w(3).toDouble
})
val mseTest = testError.map(x=>x*x).reduce(_+_)/92
> mseTest: Double = 0.058970759386070 83
```

Cost Function

Regularization

$$Cost(w) = \sum(wx^{(i)}-y^{(i)})^2 + \lambda/2*w^2$$

Other options

- Maximum Likelihood
- KL divergence
- cross-entropy

Linear Regression

Extend the single-variant solution to the multi-variant solution

- x is a vector of features, $x \in R^N$
- w is a vector of weights, $w \in R^N$
- Pre-requisite: linear algebra, multi-variant calculus

Using MLlib

Predict the house price using sqft, year_built, beds

We describe each house with a 3-element vector, e.g.,
$house^{(1)}$=(0.0769, 0.642, 0.1)

$price^{(1)}$ = (0.0798)

Using MLlib

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(3).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = LinearRegressionWithSGD.train(data,100)

model.weights

val trainError = lines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))-w(3).toDouble
})
val mseTrain = trainError.map(x=>x*x).reduce(_+_)/400
> mseTrain: Double = 0.06222798683227797

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testError = tlines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))-w(3).toDouble
})
val mseTest = testError.map(x=>x*x).reduce(_+_)/92
> mseTest: Double = 0.05444971758384607
```

Classification

We train a classifier to tell if a house is in San Francisco or the New York city

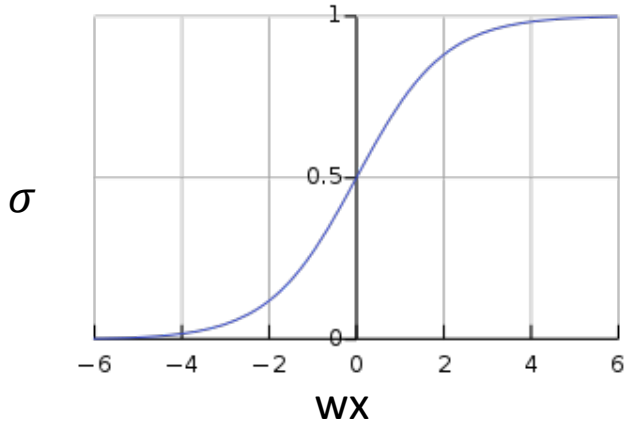
Intuition — linear regression

One class with f(wx) > threshold

and the other class with f(wx) < threshold

Using an outer sigmoid function on wx

$\sigma(wx) = 1/1+e^{-wx}$



Using Logistic Regression

```
import org.apache.spark.mllib.classification.{LogisticRegressionModel,
LogisticRegressionWithLBFGS}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(0).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = new LogisticRegressionWithLBFGS().setNumClasses(2).run(data)

model.weights

val trainPrediction = lines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(trainPrediction)
metrics.precision
> res12: Double = 0.6575

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testPrediction = tlines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(testPrediction)
metrics.precision
> res12: Double = 0.6956521739130435
```

Using SVM

```
import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(0).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = SVMWithSGD.train(data, 1000)

model.weights

val trainPrediction = lines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(trainPrediction)
metrics.precision
> res12: Double = 0.525

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testPrediction = tlines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(testPrediction)
metrics.precision
> res12: Double = 0.6304347826086957
```

Classification

Result interpretation

- Random precision = 50%
- Both linear classifier and Support Vector Machine are limited
- Support Vector Machine is good for non-linear classification

Multi-class problem

- Use multiple classifier with maximum likelihood

Supervised Learning

Linear regression

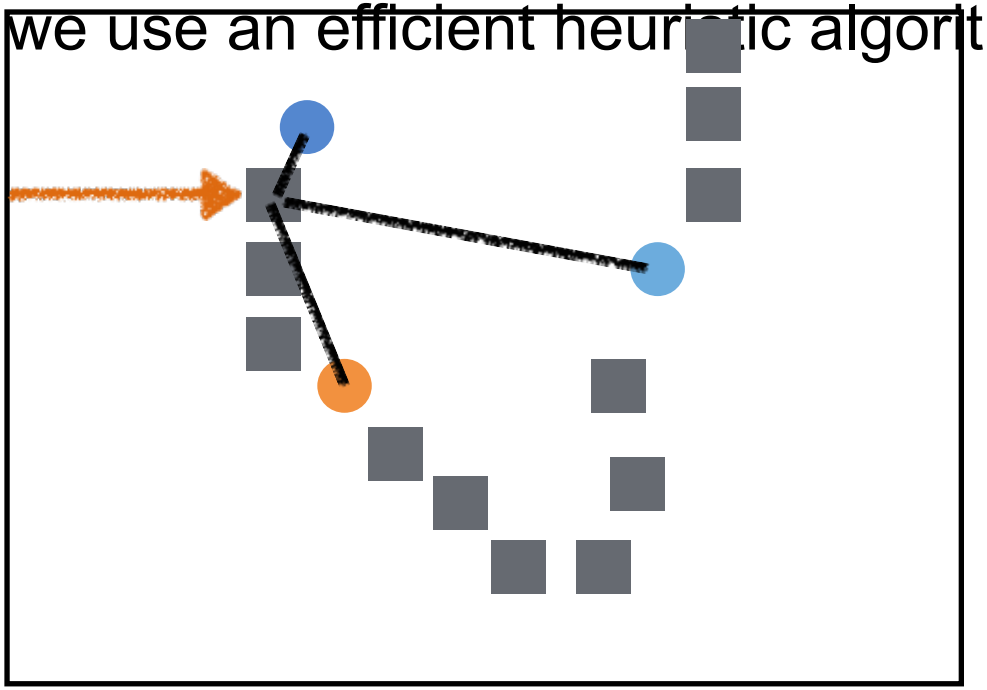
Logistic regression

How to use MLlib to train the model

Unsupervised Learning

k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

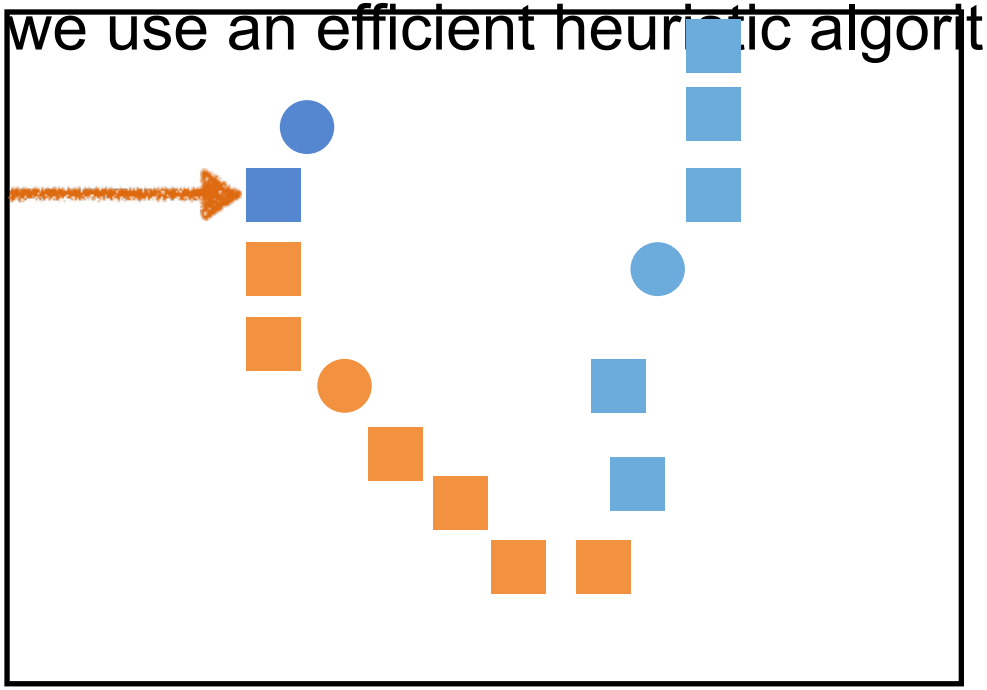
In practice, we use an efficient heuristic algorithm



k-means Clustering

k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

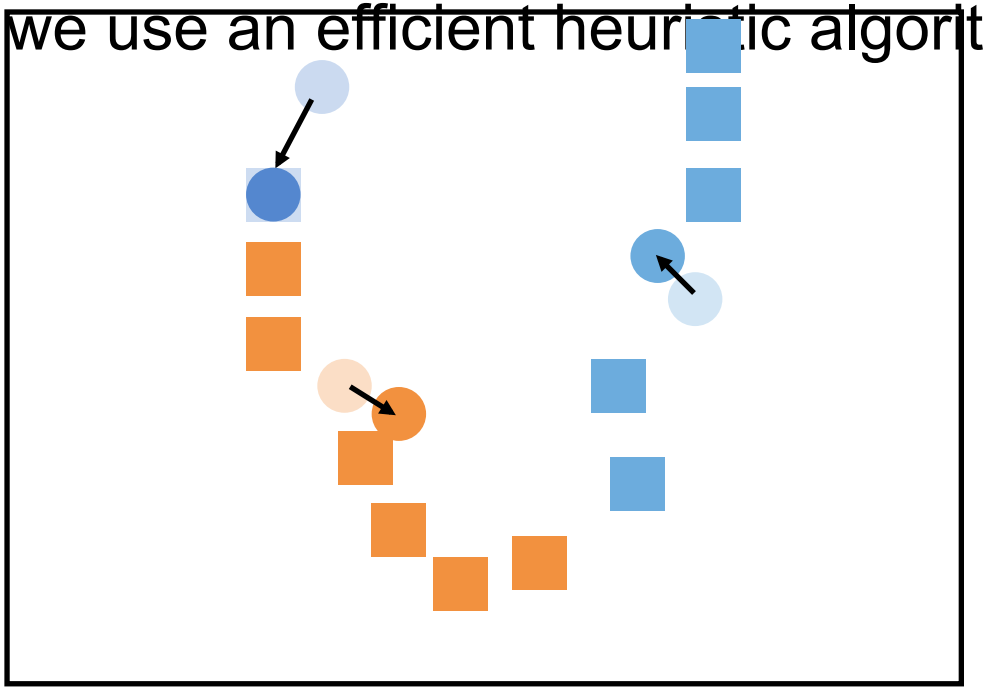
In practice, we use an efficient heuristic algorithm



k-means Clustering

k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

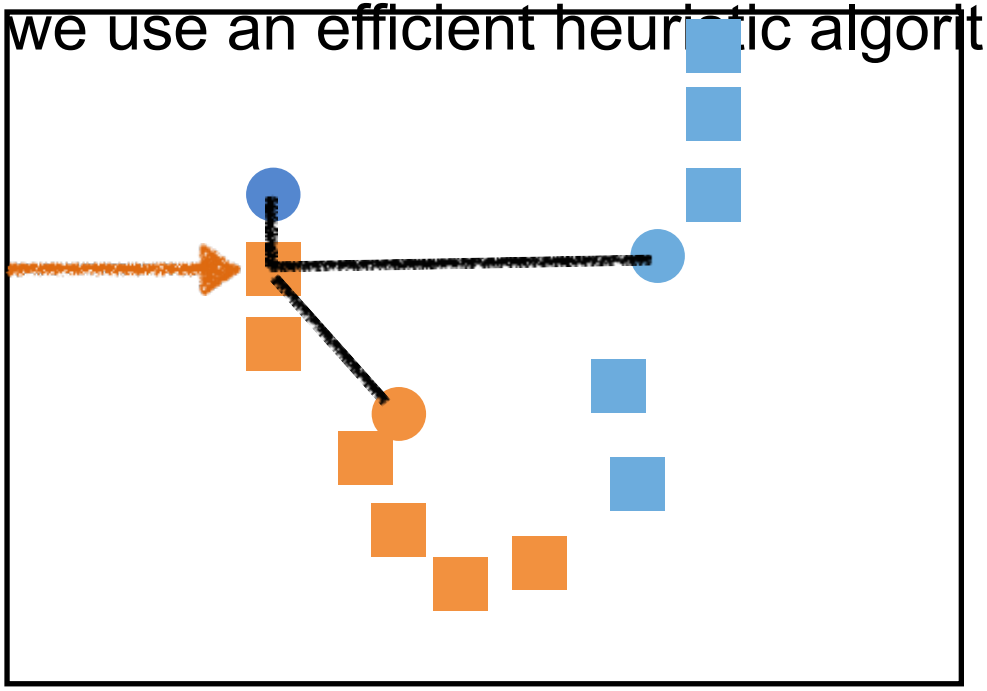
In practice, we use an efficient heuristic algorithm



k-means Clustering

k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

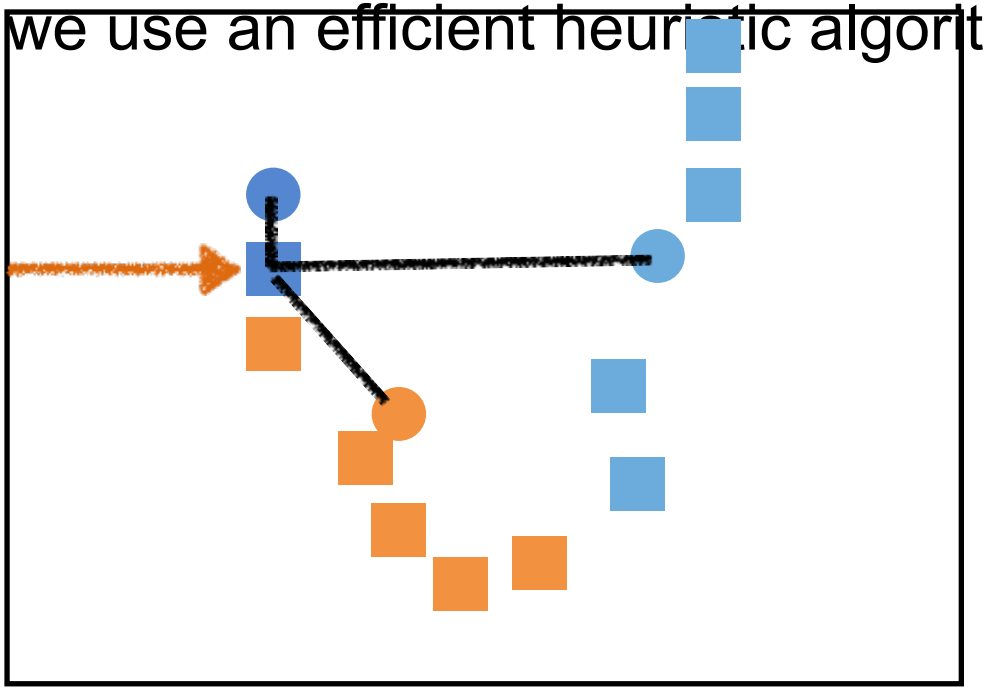
In practice, we use an efficient heuristic algorithm



k-means Clustering

k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

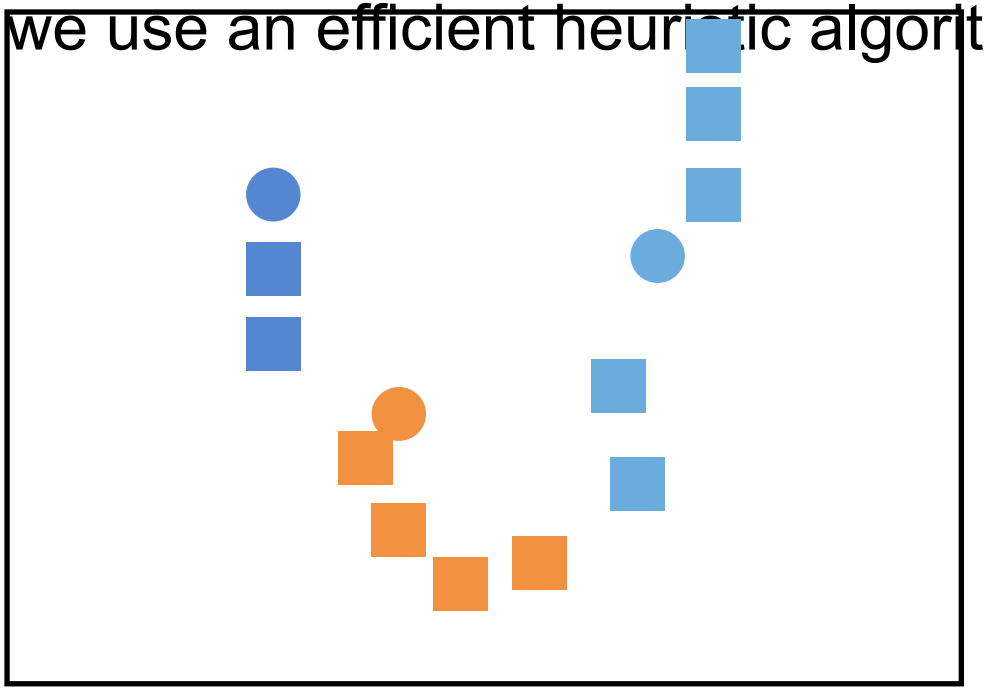
In practice, we use an efficient heuristic algorithm



k-means Clustering

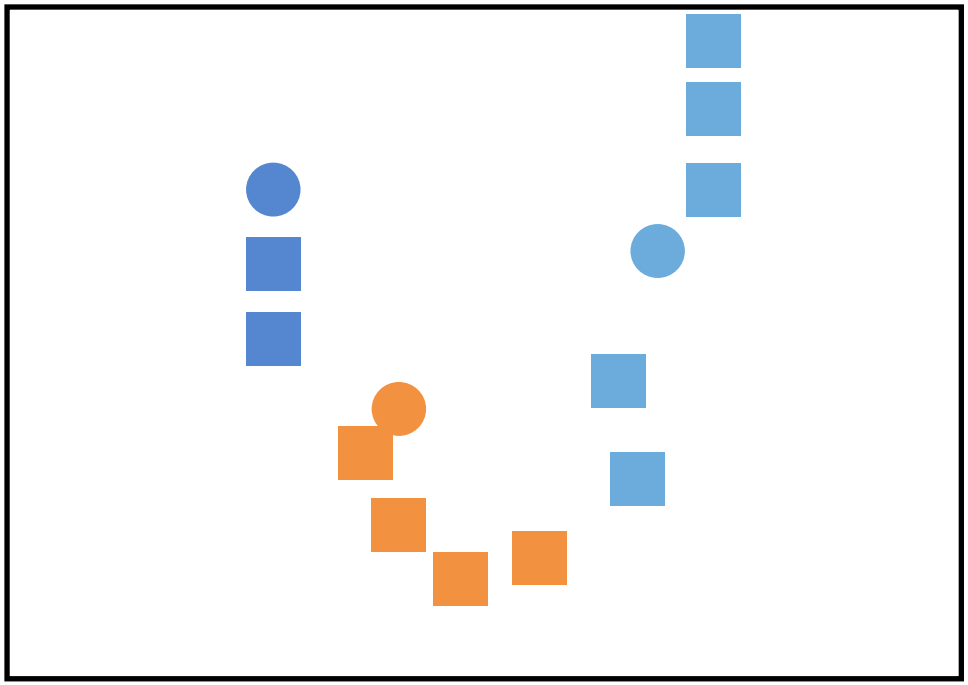
k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki

In practice, we use an efficient heuristic algorithm



k-means Clustering

Each iteration partitions the n observations into clusters with nearer mean than the previous iteration



KMeans with MLlib

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

val lines = sc.textFile("/Users/zzhang/Works/training2016/data/scaled-sf-ny-housing-train.csv")

val data = lines.map(l => {
  val w = l.split(",")
  Vectors.dense(w(1).toDouble, w(2).toDouble, w(3).toDouble, w(4).toDouble, w(5).toDouble,
w(6).toDouble)
})

val clusters = KMeans.train(data, 2, 100)

val pred = lines.map(l => {
  val w = l.split(",")
  val v = Vectors.dense(w(1).toDouble, w(2).toDouble, w(3).toDouble, w(4).toDouble, w(5).toDouble,
w(6).toDouble)
  math.pow(cluster.predict(v) - w(0).toInt, 2)
})

res = pred.reduce(_+_)
>res: Int = 177
```

Other MLlib Functionalities

Classification

org.apache.spark.mllib.classification.SVMWithSGD

org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS

Regression

org.apache.spark.mllib.regression.LinearRegressionWithSGD

org.apache.spark.mllib.regression.RidgeRegressionWithSGD

org.apache.spark.mllib.regression.LassoWithSGD

Collaborative filtering

org.apache.spark.mllib.recommendation.ALS

- Clustering
  - org.apache.spark.mllib.clustering.KMeans
  - org.apache.spark.mllib.clustering.GaussianMixture
- Dimensionality Reduction
  - org.apache.spark.mllib.linalg.Matrix.computeSVD
  - org.apache.spark.mllib.linalg.Matrix.computePrincipalComponents
- Many Others

Graph Processing

Google first released its PageRank algorithms in 1997, which formulates the Web as a directed graph and ranks the web pages

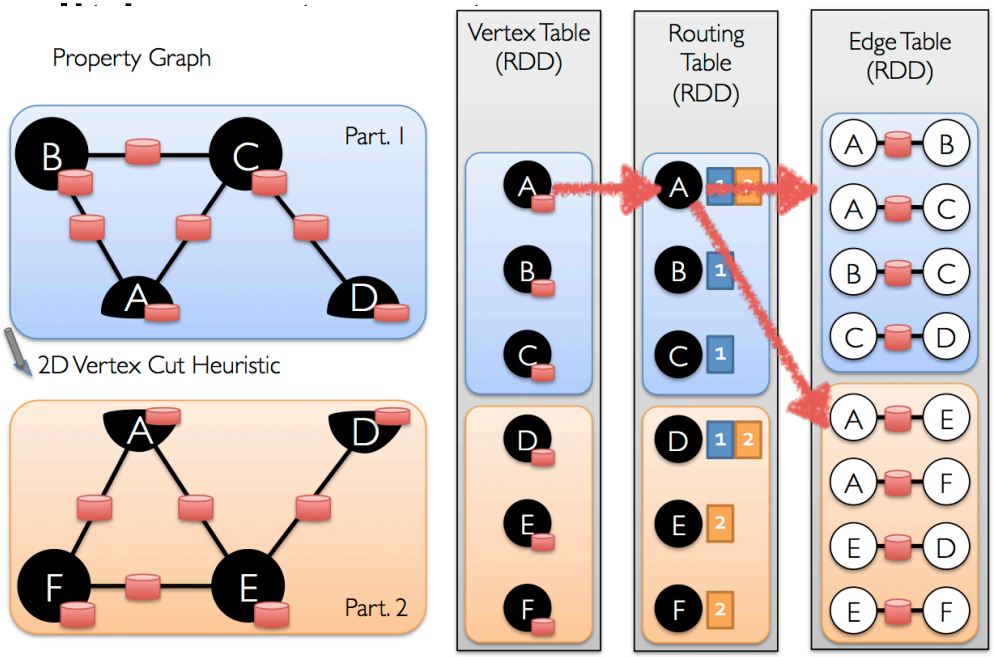
Social networks (e.g., Facebook, Linkedin) formulates the social networks as directed graphs the do community detection and friends recommendation

Inspired by Hadoop, there are numerous graph processing frameworks implemented in the past decade: Pregel, Giraph, GraphLab (acquired by Apple), GraphX, …

GraphX

GraphX abstracts a graph with an RDD of vertices and an RDD of edges

A graph is s



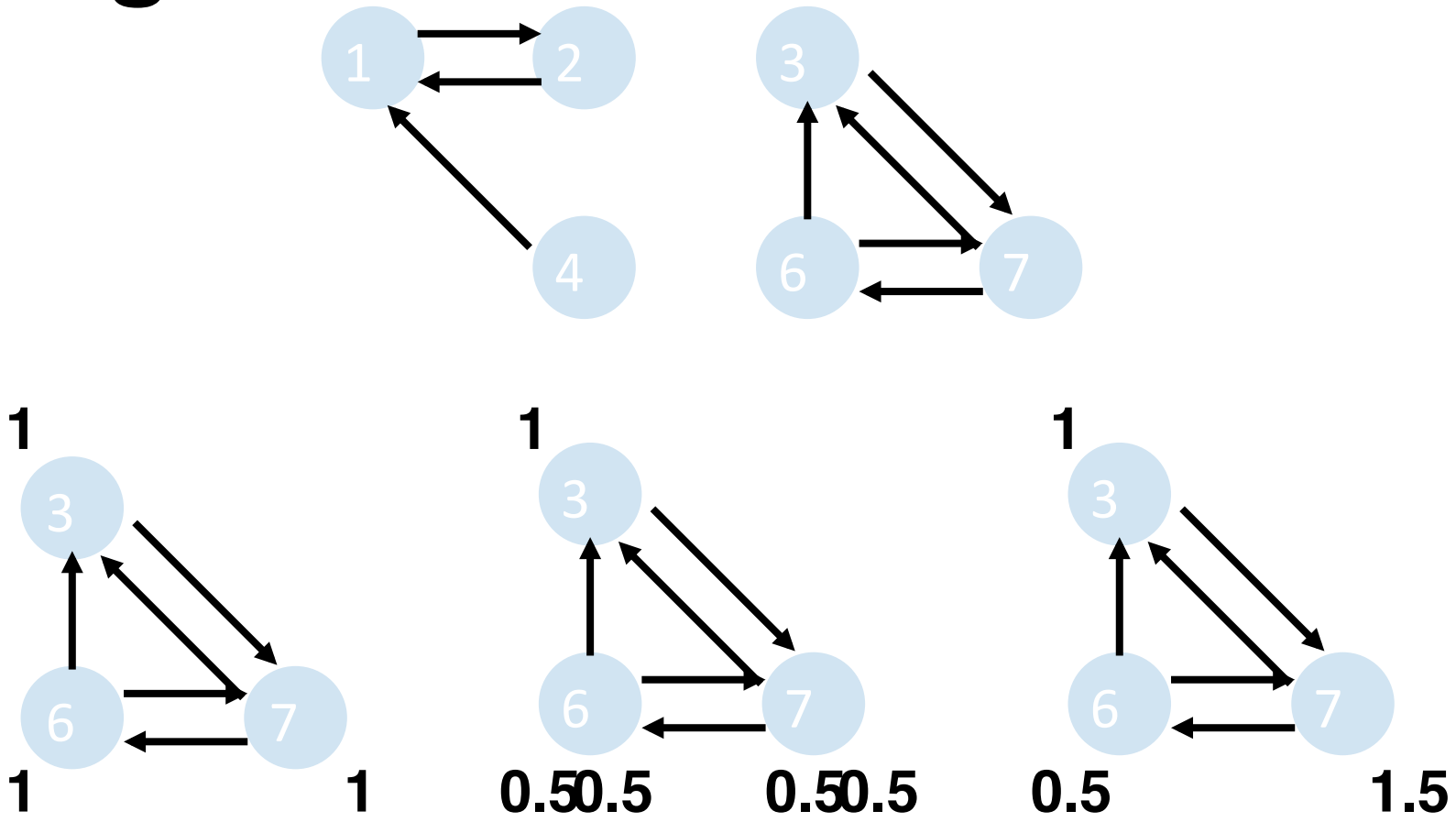
Courtesy image from http://spark.apache.org/docs/latest/graphx-programming-guide.

GraphX

High level Algorithm

- PageRank
- Connected components
- Triangle counting
- Shortest path

PageRank



score = 0.15 +0.85*pagerank

Using GraphX

```
import org.apache.spark.graphx._
import org.apache.spark.graphx.util.GraphGenerators
val graph = GraphLoader.edgeListFile(sc, "/tmp/spark-training/data/followers.txt")
val ranks = graph.pageRank(0.0001).vertices
ranks.sortBy(_._2, false).collect
```

```
res10: Array[(org.apache.spark.graphx.VertexId, Double)] =
Array((1,1.4588814096664682), (2,1.390049198216498),
(7,1.2973176314422592), (3,0.999344203850723),
(6,0.7013599933629602), (4,0.15))
```

Other GraphX Functionalities

Connected Components

org.apache.spark.graphx.lib.connectedComponents

Triangle Counting

org.apache.spark.graphx.lib.triangleCount

Shortest Paths

org.apache.spark.graphx.lib.Shortestpaths

Many others

Big Data Analysis - Part III: Spark Internals and Configurations

Zhao Zhang
Data Mining and Statistics Group
zzhang@tacc.utexas.edu
Texas Advanced Computing Center
May 4, 2017

These slides include talks given by Jey Kottalam of AMPLab at LBNL 2015 and
Reynold Xin and Aaron Davidson of Databricks at Spark Summit 2014

Goals

Understand Spark's architecture and components

Understand the logic when an application is submitted

Understand how the application is logically partitioned

Understand how Spark manages memory

Outline

Task Management

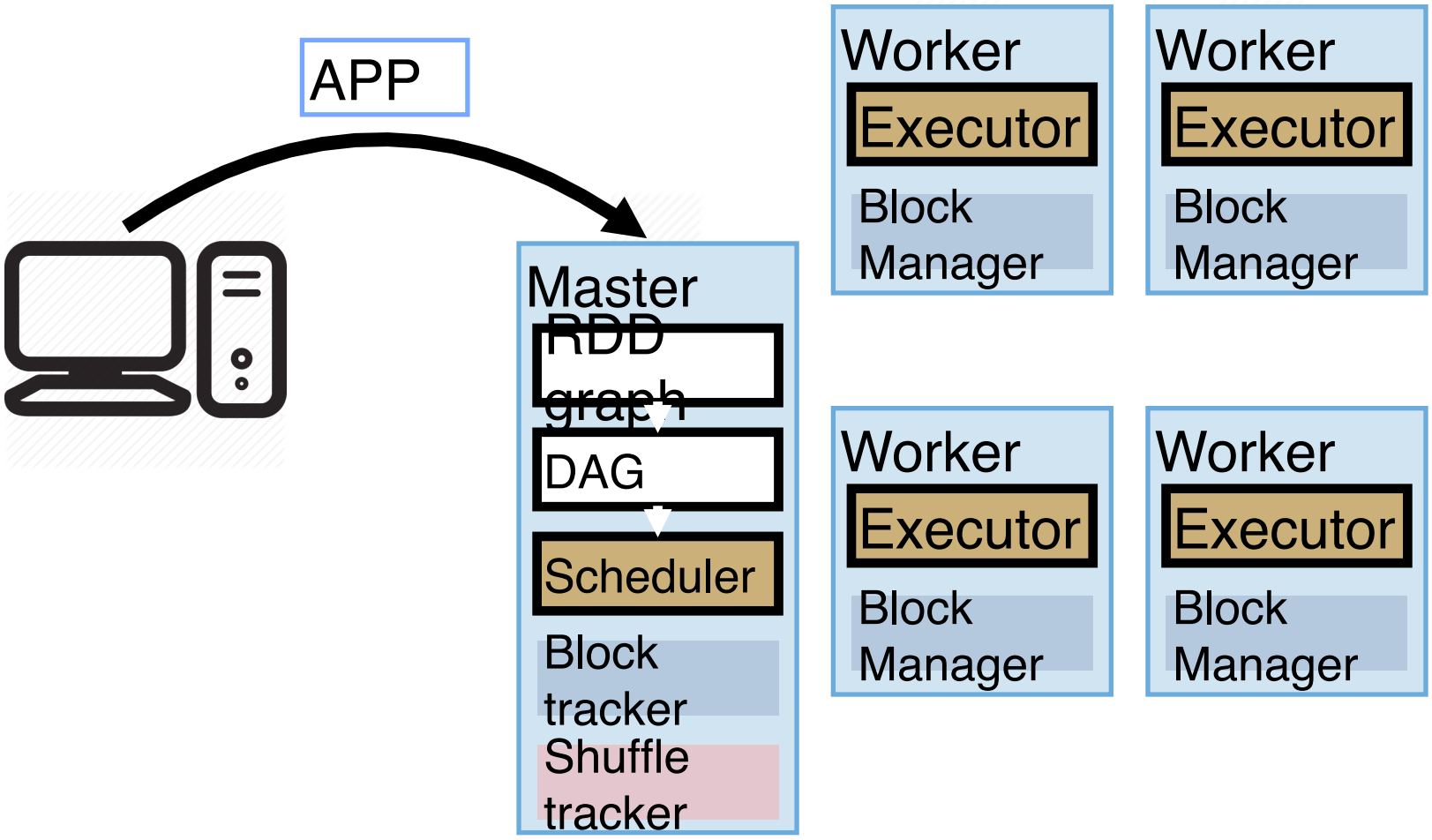
Memory Management

Outline

Task Management

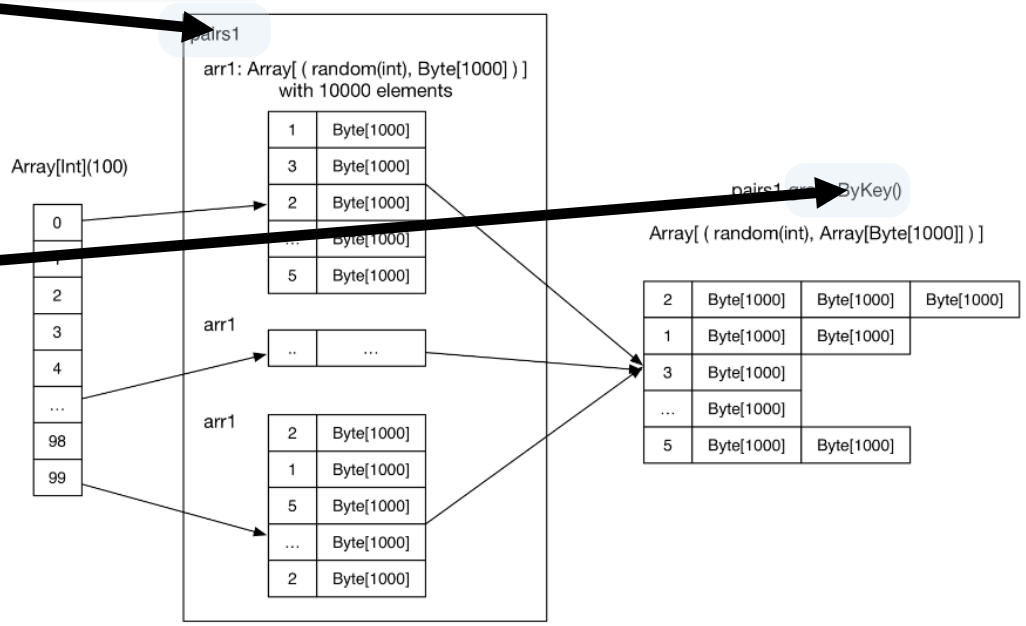
Memory Management

Spark Architecture

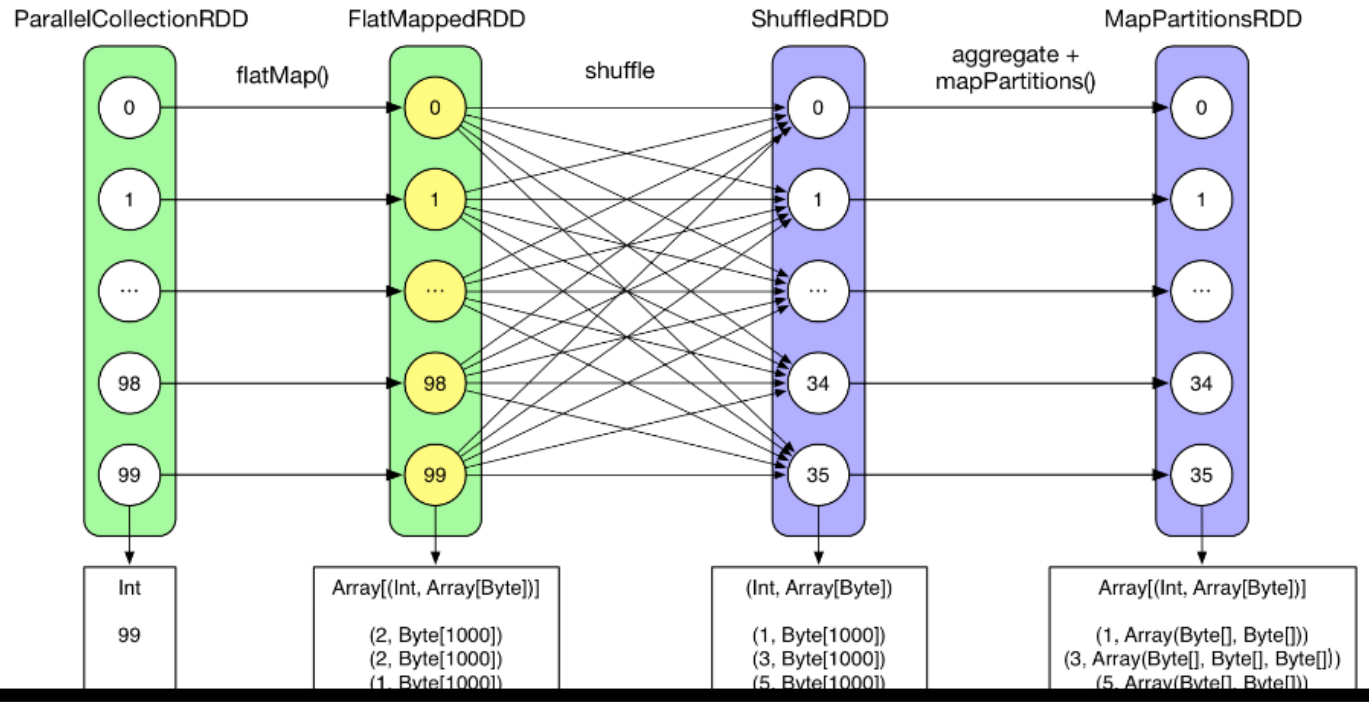


GroupByTest Example

```
object GroupByTest {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("GroupBy Test")
    var numMappers = 100
    var numKVPairs = 10000
    var valSize = 1000
    var numReducers = 36

    val sc = new SparkContext(sparkConf)

    val pairs1 = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>
      val ranGen = new Random
      var arr1 = new Array[(Int, Array[Byte])](numKVPairs)
      for (i <- 0 until numKVPairs) {
        val byteArr = new Array[Byte](valSize)
        ranGen.nextBytes(byteArr)
        arr1(i) = (ranGen.nextInt(Int.MaxValue), byteArr)
      }
      arr1
    }.cache
    // Enforce that everything has been calculated and in cache
    pairs1.count

    println(pairs1.groupByKey(numReducers).count)

    sc.stop()
  }
}
```



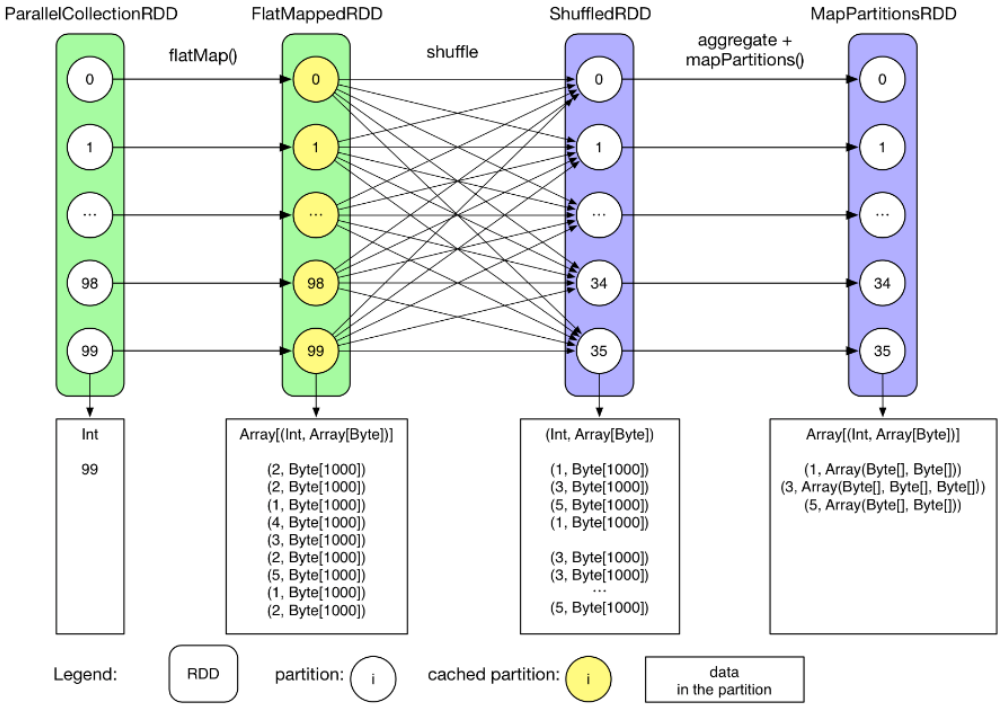
RDD Graph



MapPartitionsRDD[3] at groupByKey at GroupByTest.scala:51 (36 partit
ShuffledRDD[2] at groupByKey at GroupByTest.scala:51 (36 partitions)
FlatMappedRDD[1] at flatMap at GroupByTest.scala:38 (100 partition
ParallelCollectionRDD[0] at parallelize at GroupByTest.scala:38 (10

DAG Generation

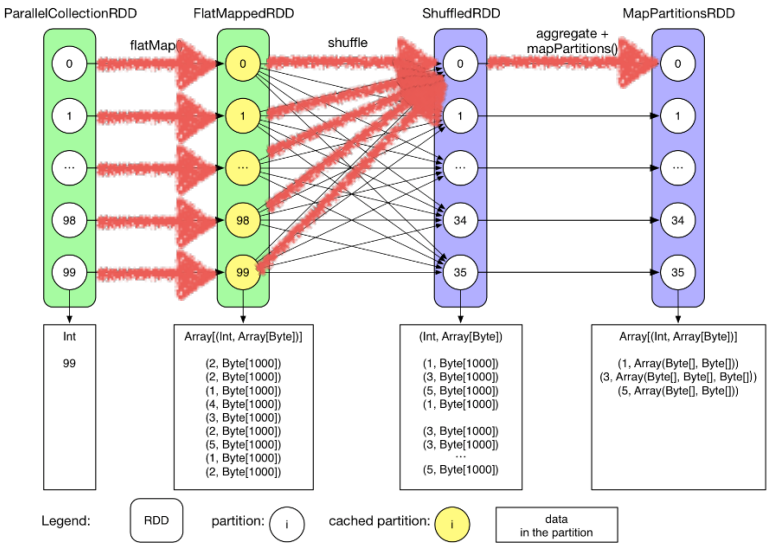
- One task per arrow?
- 100 map + 100x36 shuffle + 36 reduce tasks
- Intermediate result storage



DAG Generation

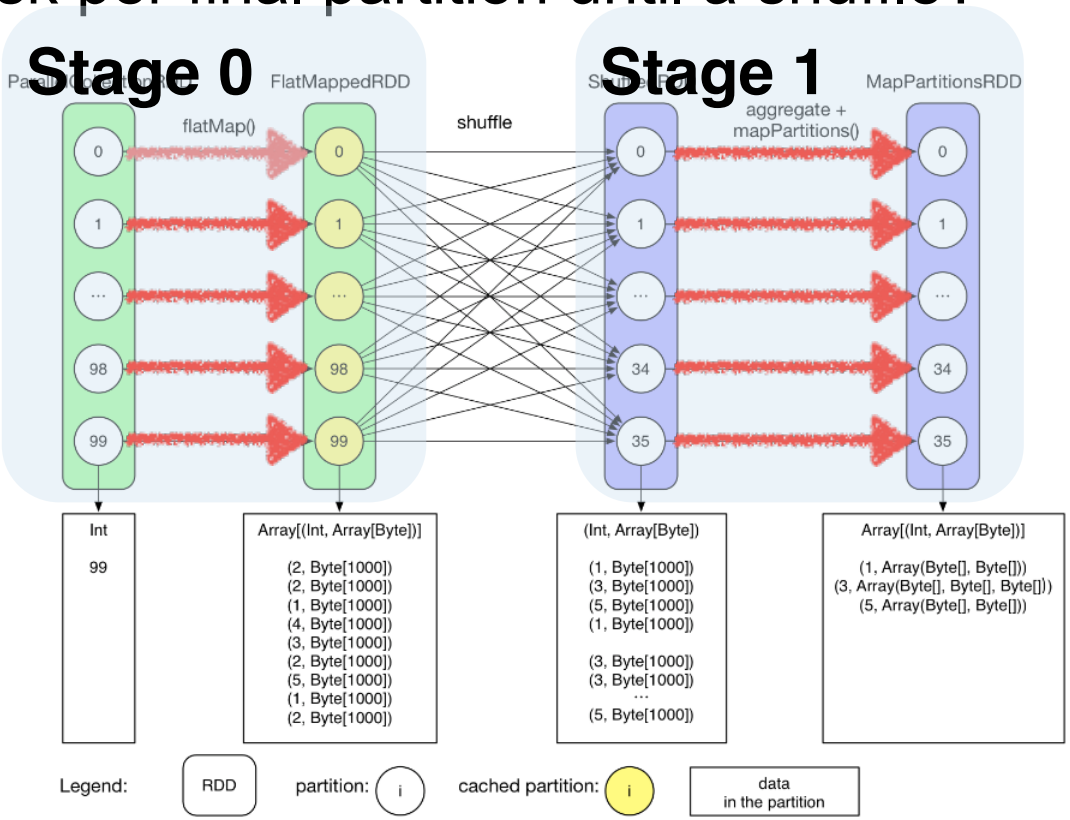
- The first task is too big
- One task per final partition?
- Need a smart algorithm to cache intermediate data

☑ Data is not computed until needed

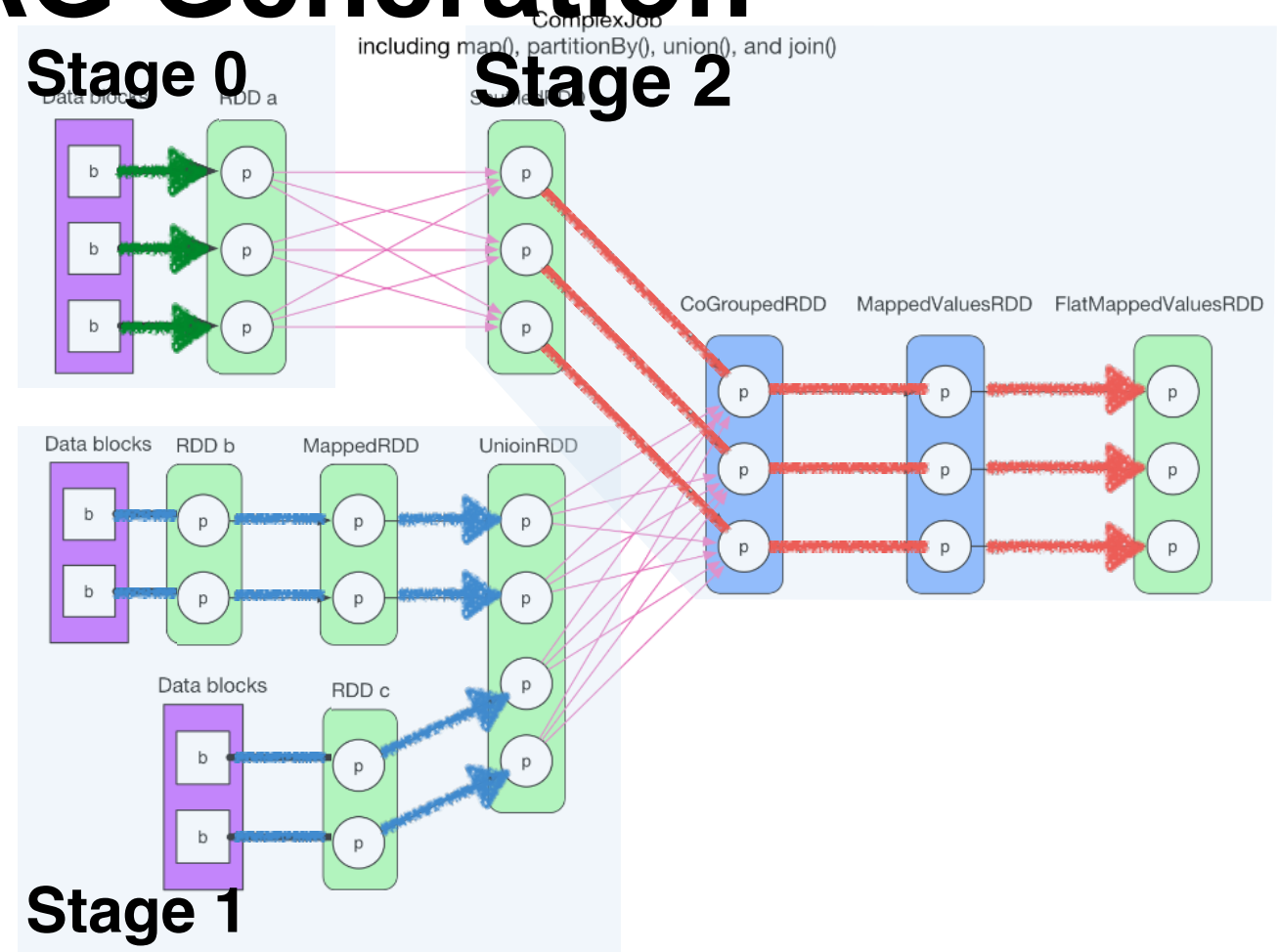


DAG Generation

- What if there are consecutive map stages?
- One task per final partition until a shuffle?



DAG Generation



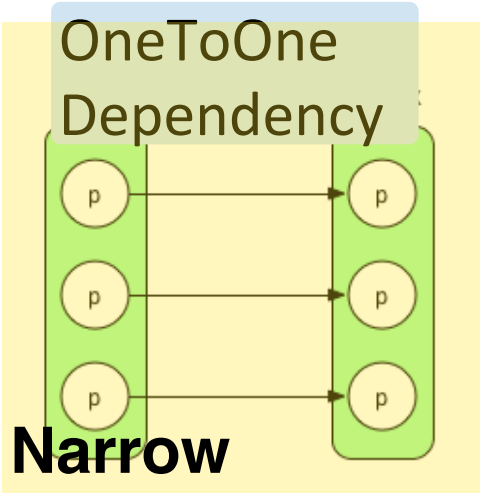
RDD Dependency

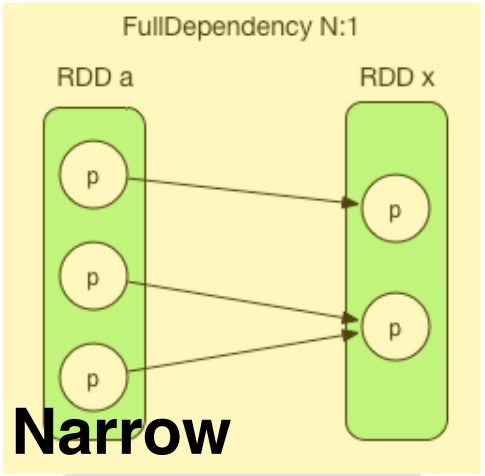
How many parent RDDs are there for the current RDD?

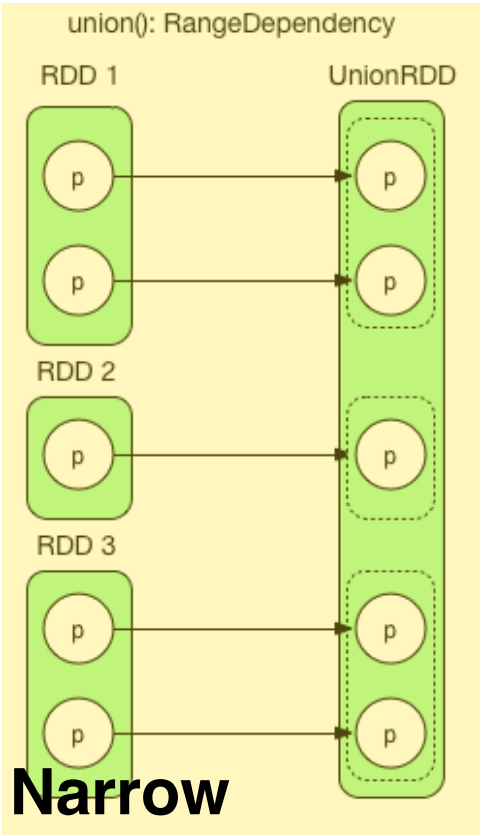
How many partitions are there in the current RDD?

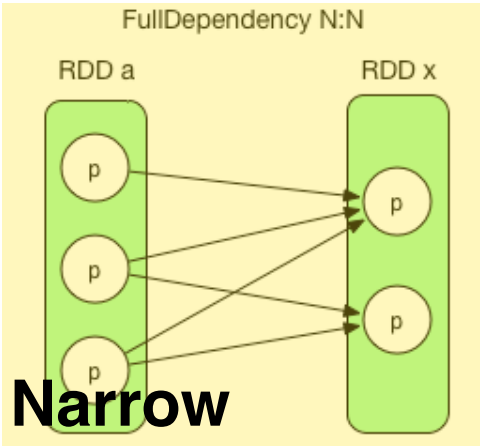
How does each partition in the current RDD depend on the partitions in parent RDDs?

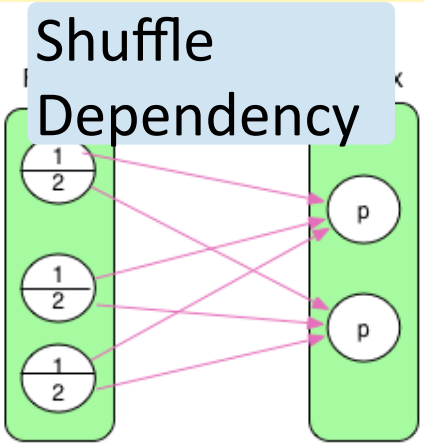
RDD Dependency



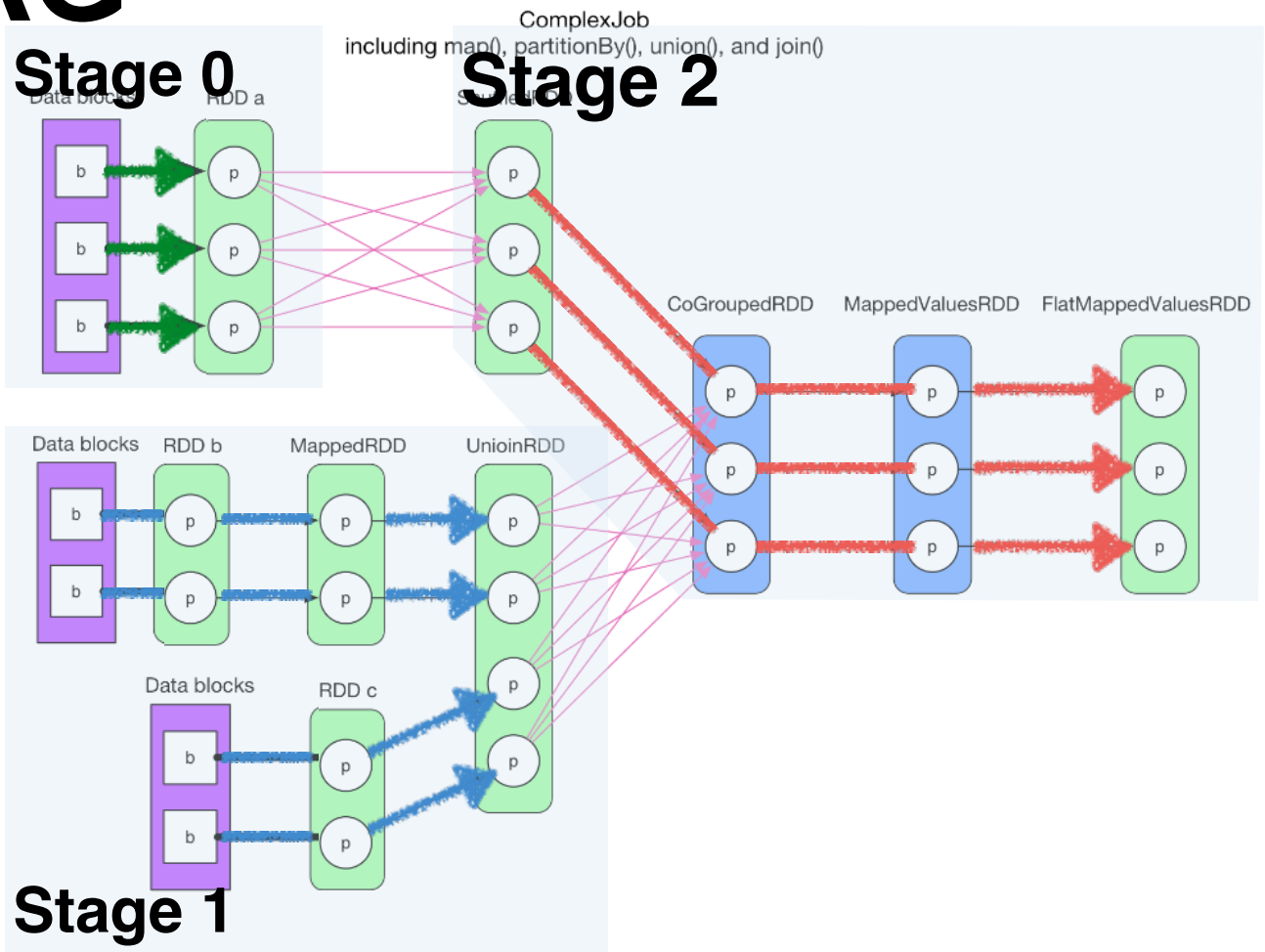








DAG Generation Revisited



Sample Execution of a Spark Job

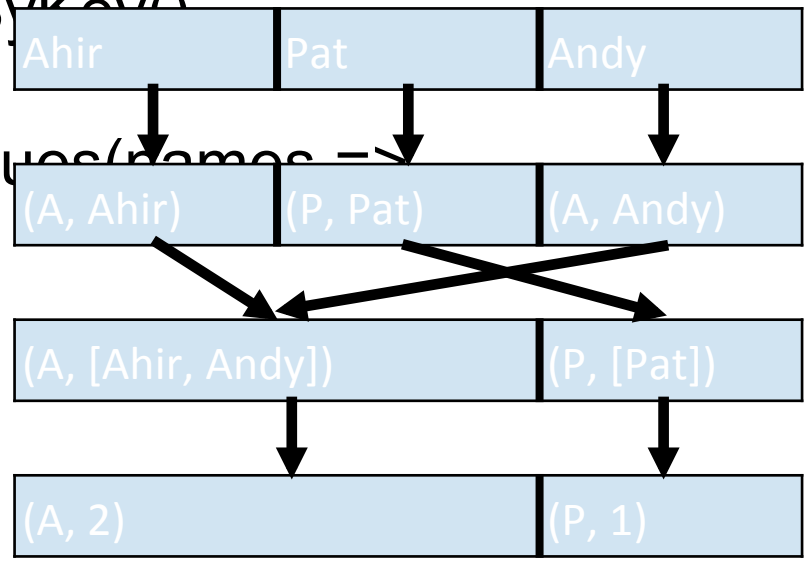
1. val lines = sc.textFile("hdfs://names")

2. val kvp = lines.map(name => (name(0), name))

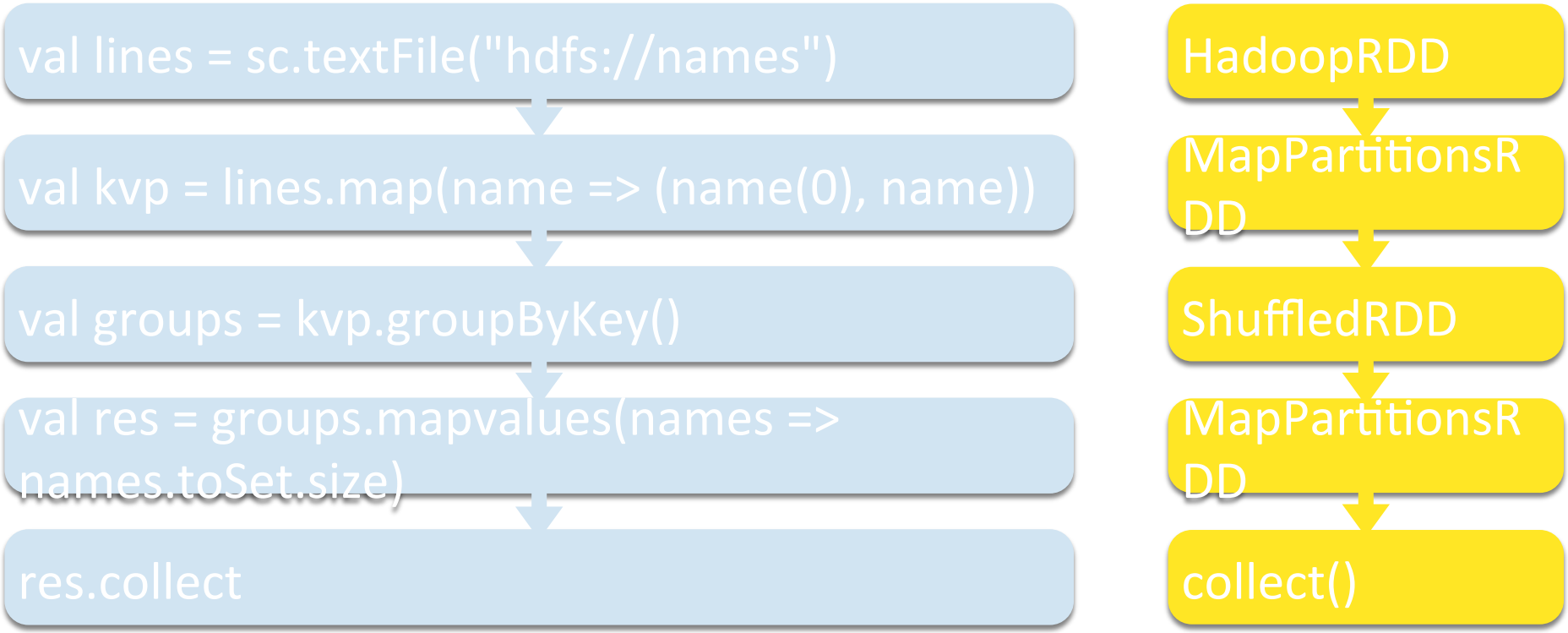
3. val groups = kvp.groupBy

4. val res = groups.mapValues(
names.toSet.size)

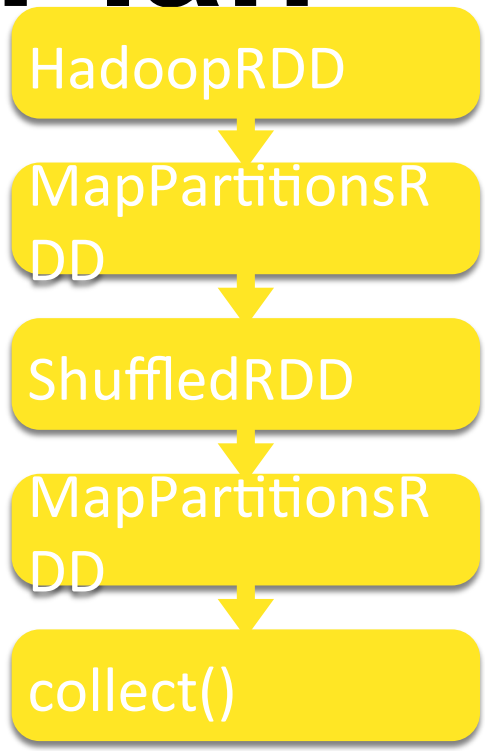
5. res.collect

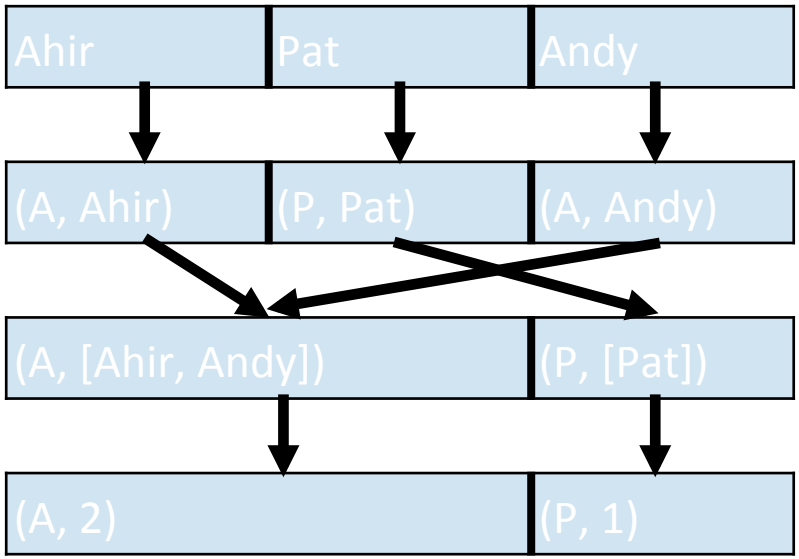


Create RDDs


val lines = sc.textFile("hdfs://names")
val kvp = lines.map(name => (name(0), name))
val groups = kvp.groupByKey()
val res = groups.mapvalues(names => names.toSet.size)
res.collect
HadoopRDD
MapPartitionsRDD
ShuffledRDD
MapPartitionsRDD
collect()

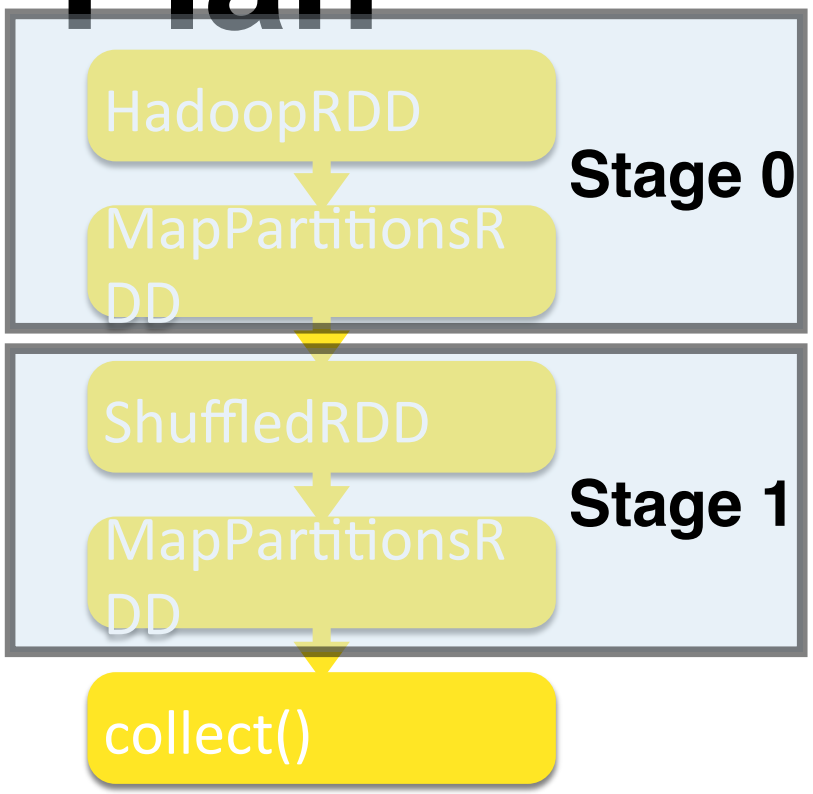
Create Execution Plan

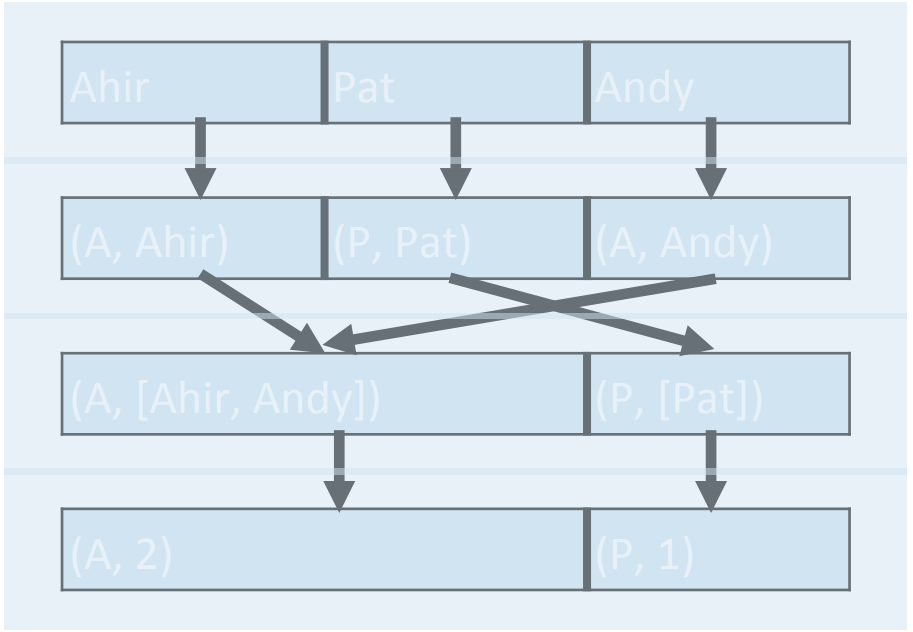




res0 = [(A, 2), (P, 1)]

Create Execution Plan





res0 = [(A, 2), (P, 1)]

Log Interpretation

17/05/03 17:19:33 INFO SparkContext: Starting job: collect at <console>:45
17/05/03 17:19:33 INFO DAGScheduler: Got job 321 (collect at <console>:45) with 2 output partitions
17/05/03 17:19:33 INFO DAGScheduler: Final stage: **ResultStage 340** (collect at <console>:45)
17/05/03 17:19:33 INFO DAGScheduler: Parents of final stage: List(**ShuffleMapStage 339**)
17/05/03 17:19:33 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 339)
17/05/03 17:19:33 INFO DAGScheduler: Submitting ShuffleMapStage 339 (MapPartitionsRDD[658] at map at <console>:38), which has no missing parents
17/05/03 17:19:33 INFO DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 339 (MapPartitionsRDD[658] at map at <console>:38)
17/05/03 17:19:33 INFO TaskSchedulerImpl: Adding task set 339.0 with 2 tasks
...
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 0.0 in stage 339.0** (TID 688) in 4 ms on localhost (1/2)
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 1.0 in stage 339.0** (TID 689) in 4 ms on localhost (2/2)
17/05/03 17:19:33 INFO DAGScheduler: **ShuffleMapStage 339** (map at <console>:38) **finished** in 0.004 s
...
17/05/03 17:19:33 INFO DAGScheduler: waiting: Set(**ResultStage 340**)
17/05/03 17:19:33 INFO DAGScheduler: Submitting ResultStage 340 (MapPartitionsRDD[660] at mapValues at <console>:42), which has no missing parents
17/05/03 17:19:33 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 340 (MapPartitionsRDD[660] at mapValues at <console>:42)
17/05/03 17:19:33 INFO TaskSchedulerImpl: Adding task set 340.0 with 2 tasks
...
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 1.0 in stage 340.0** (TID 691) in 2 ms on localhost (1/2)

Schedule Tasks

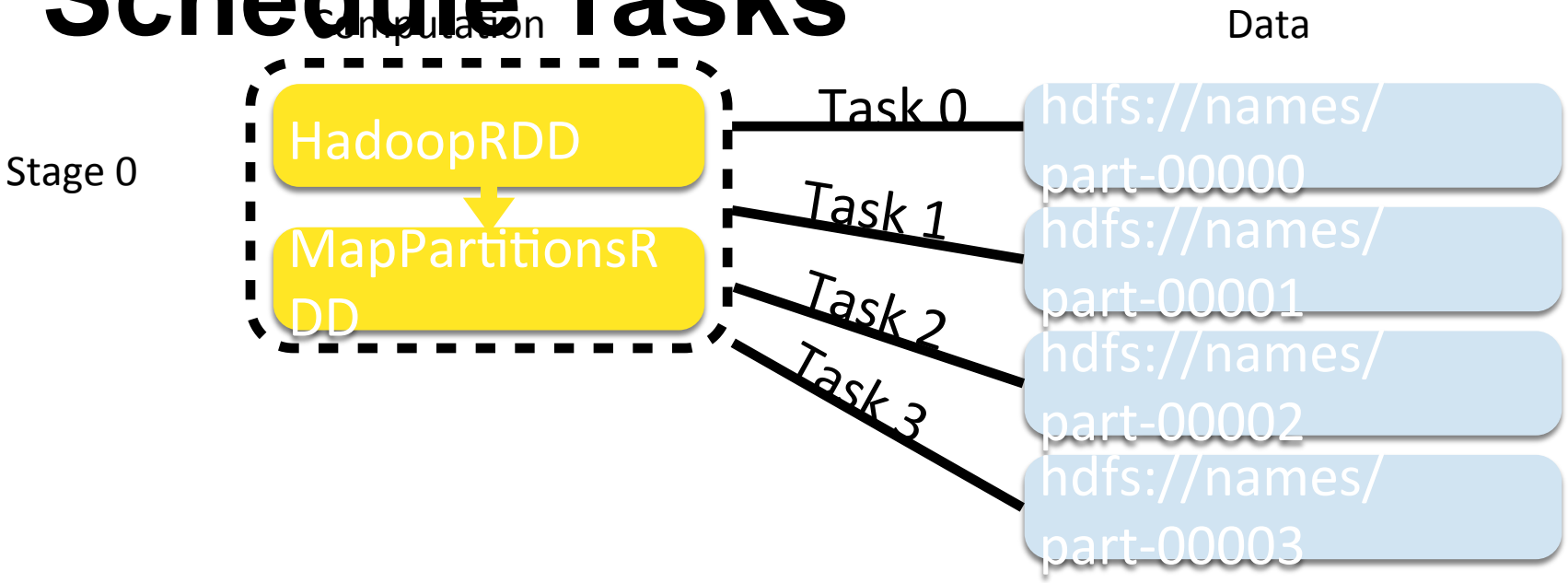
Split each stage into tasks based on partitions

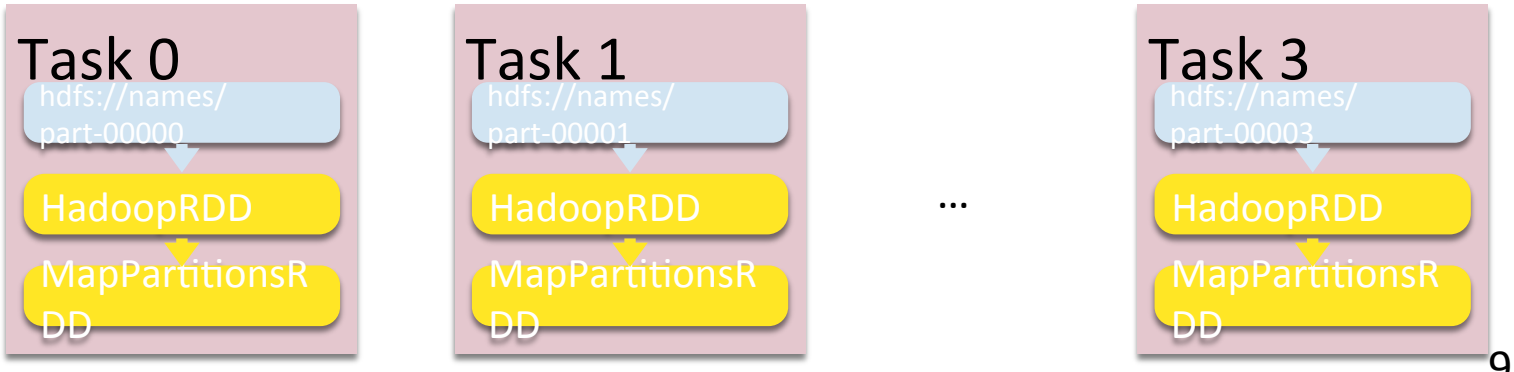
A task is data + computation

From the last stage, recursively find parent stages, then schedule the stage that all parent stages have been executed or there is no parent stage

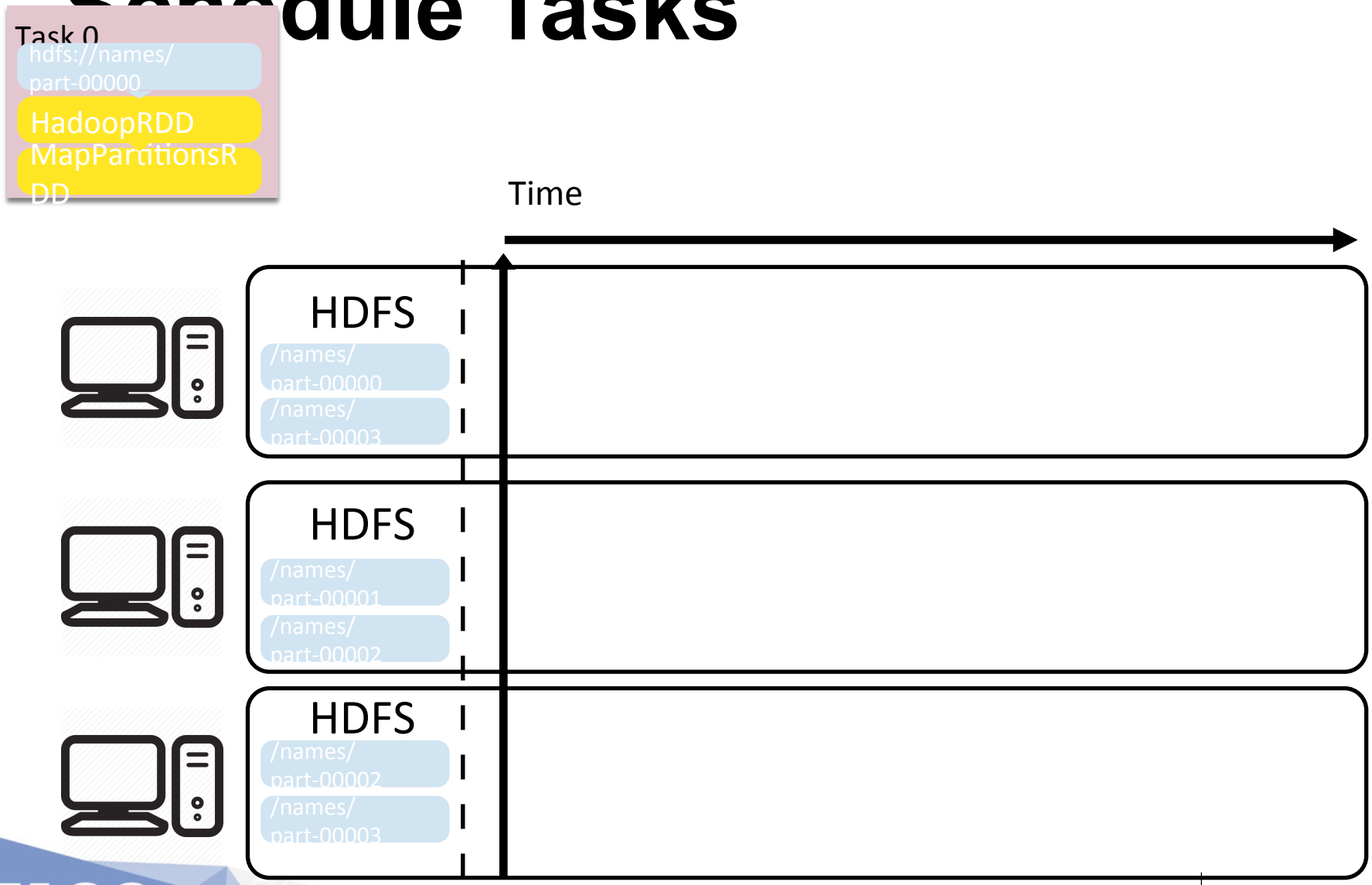
Execute all tasks within a stage before moving on to the next

Schedule Tasks

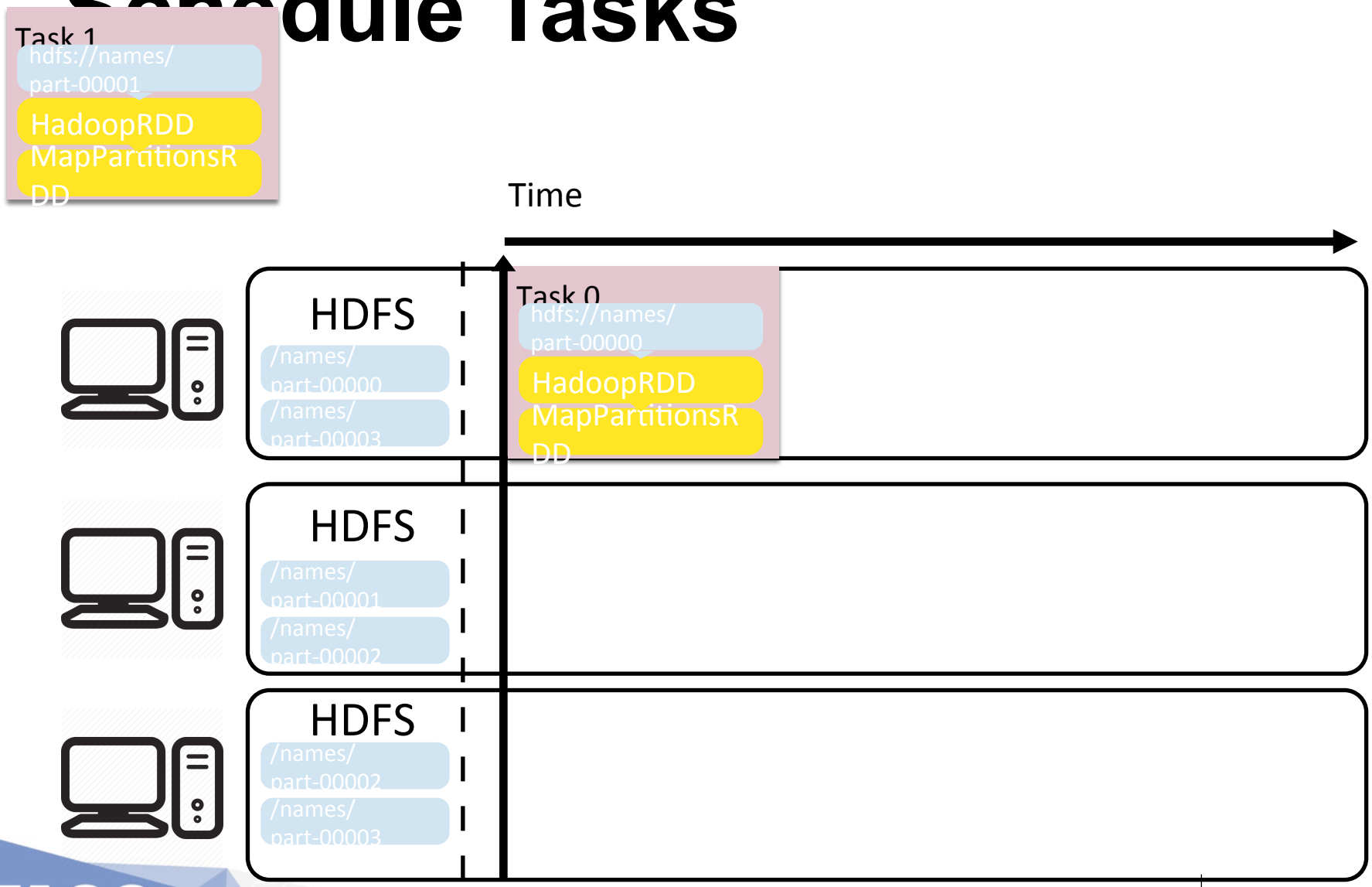




Schedule Tasks



Schedule Tasks

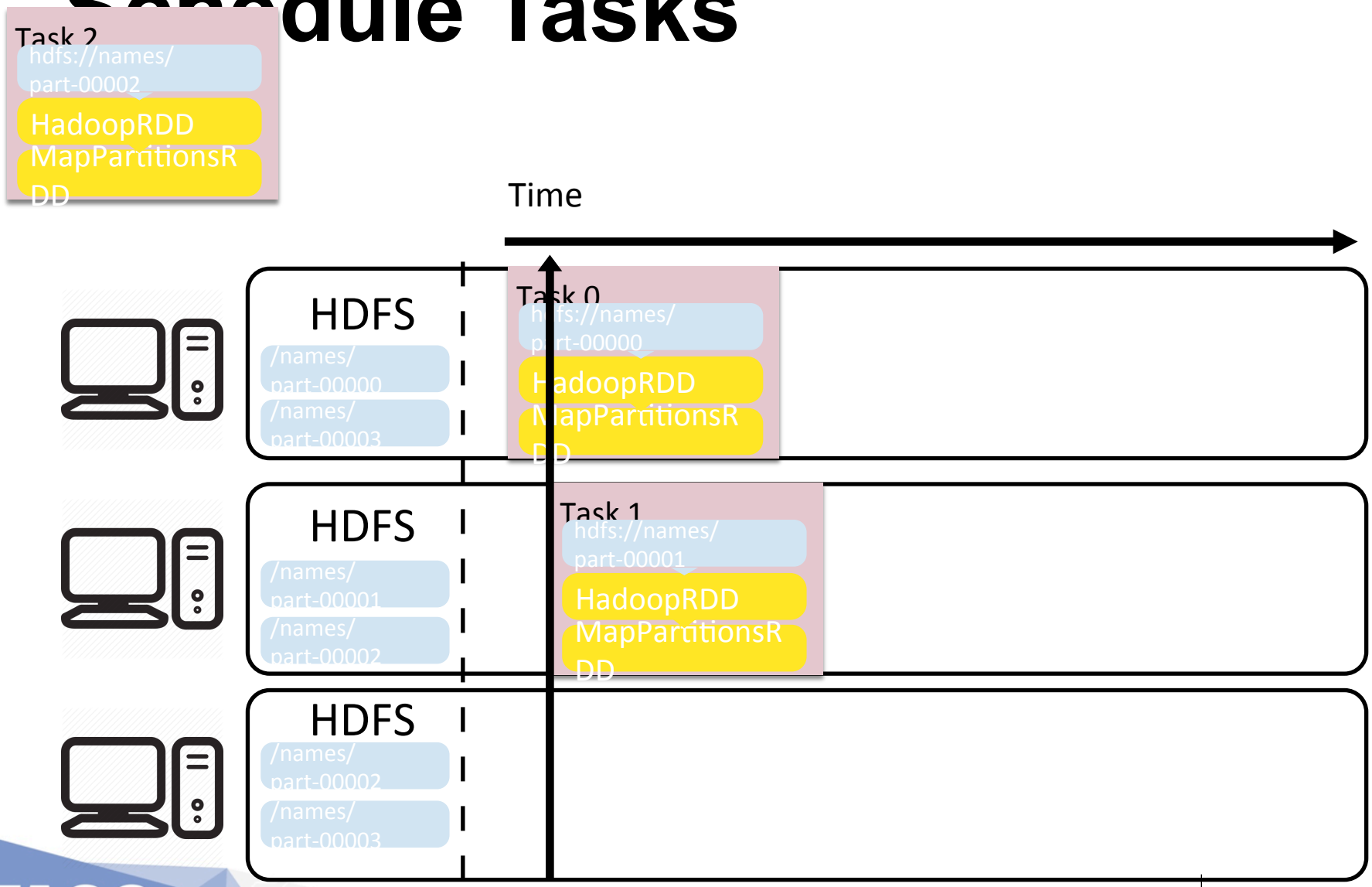


Delay Scheduling

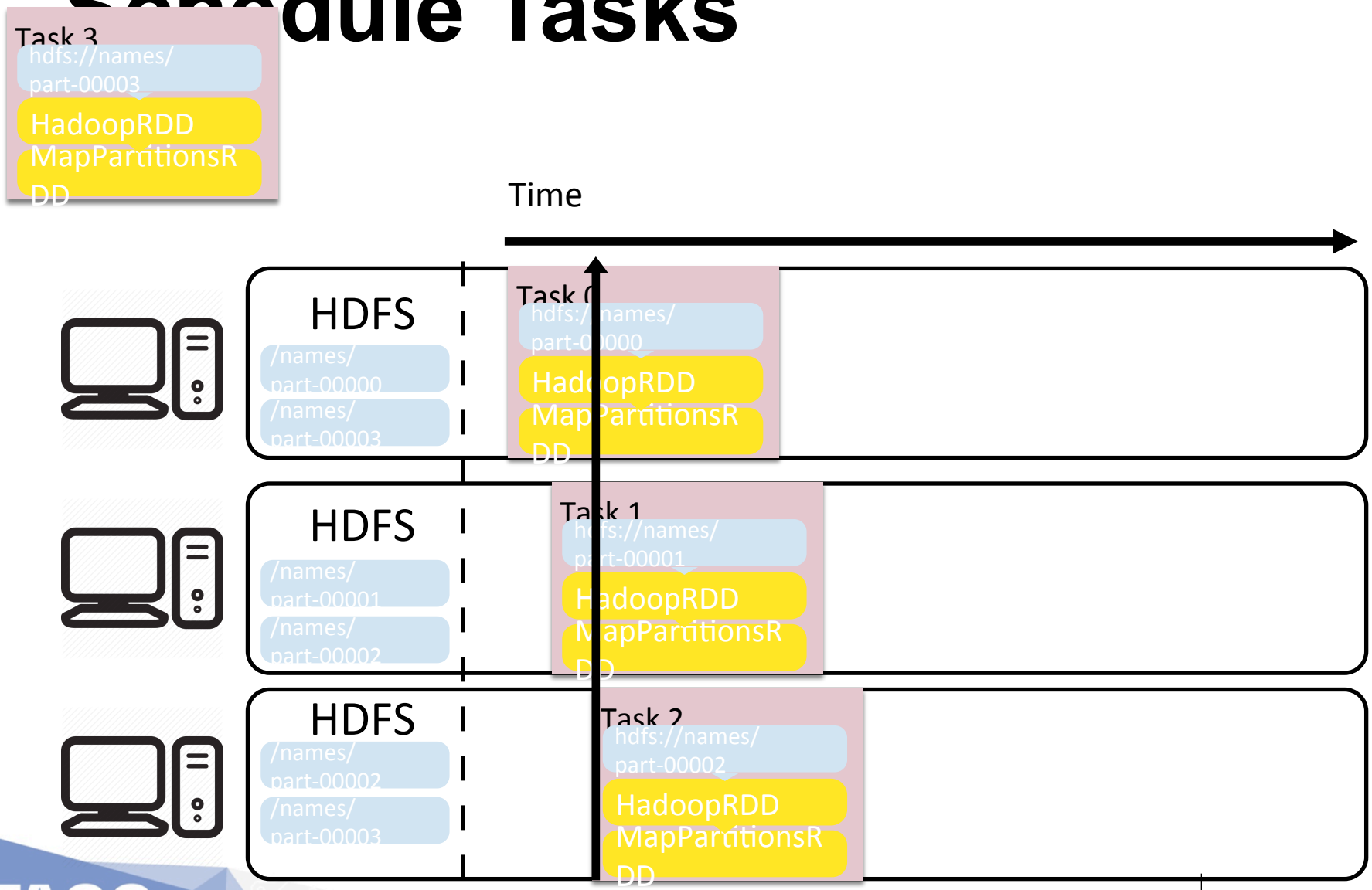
How long a task should waits for the node that has the data?

- spark.locality.wait (default 3s)
- spark.locality.wait.process
- spark.locality.wait.node
- spark.locality.wait.rack

Schedule Tasks



Schedule Tasks



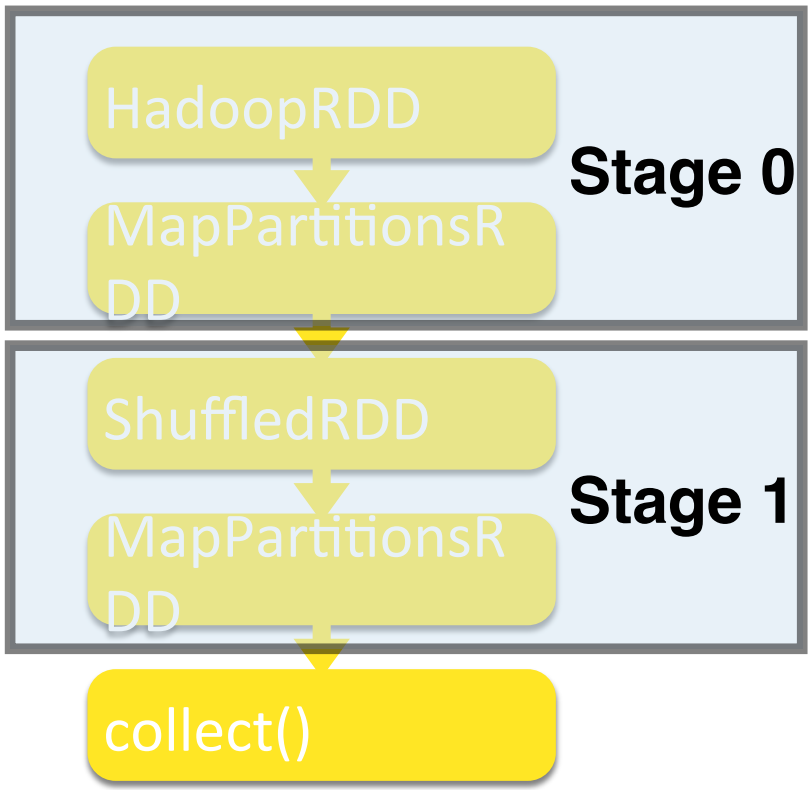
Now Stage 1 Finishes

Shuffle

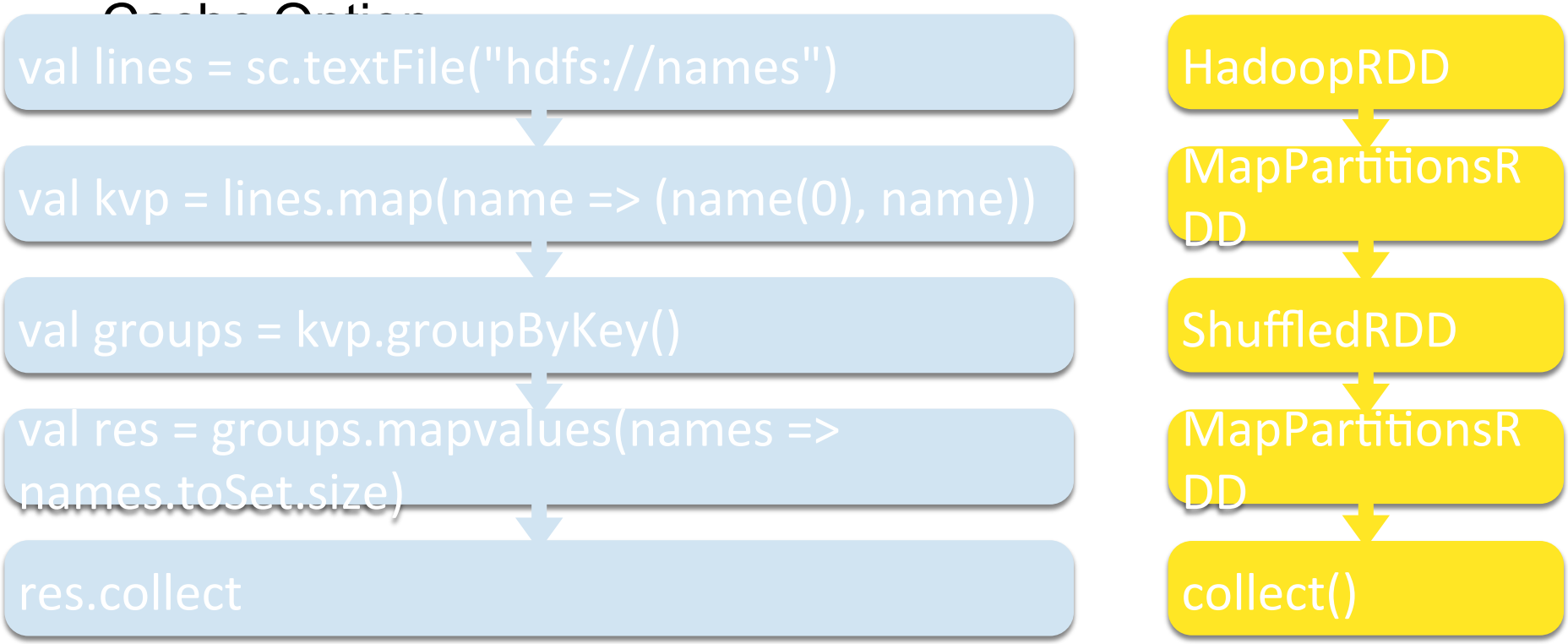
Stage 0 Execution

Results

Done

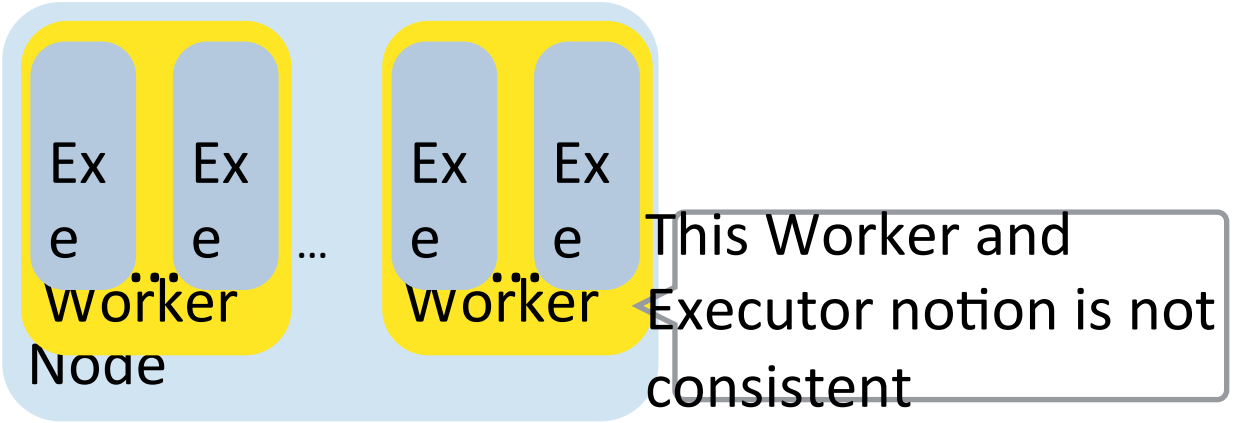

HadoopRDD
MapPartitionsRDD
Stage 0
ShuffledRDD
MapPartitionsRDD
Stage 1
collect()

Cache


val lines = sc.textFile("hdfs://names")
val kvp = lines.map(name => (name(0), name))
val groups = kvp.groupByKey()
val res = groups.mapvalues(names => names.toSet.size)
res.collect
HadoopRDD
MapPartitionsRDD
ShuffledRDD
MapPartitionsRDD
collect()

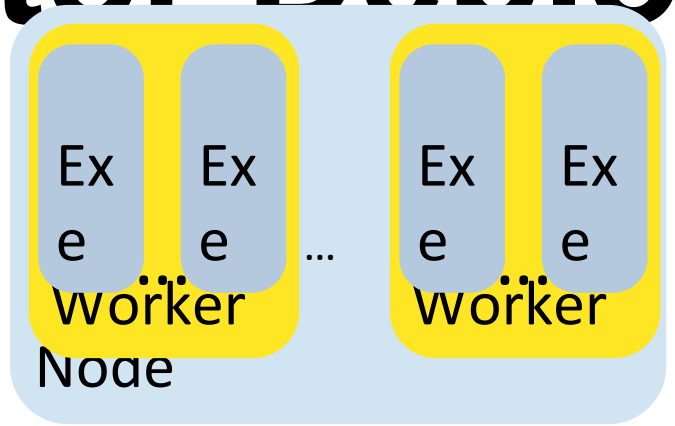
Executor Deployment

Node — Worker — Executor



- Workers share the same physical node
- Executors in the same worker shared the same process

Executor Deployment



Spark YARN mode, in conf/spark-env.sh

SPARK_EXECUTOR_INSTANCES (default 2)

SPARK_EXECUTOR_CORES (default 1)

Review

How does Spark generate DAG?

How does Spark partition a DAG into tasks?

How do Spark tasks get schedule?

How to set Spark executors

Outline

Task Management

Memory Management

Spark Memory Consumption

Spark consumes more memory than you think

- Each Java object has a roughly 16-byte "object header"
- Java Strings have ~40 bytes of overhead
- Common collection classes, such as HashMap and LinkedList, has a "wrapper object" for each entry
- Collections of primitive types are stored as "boxed" objects such as Java.lang.Integer

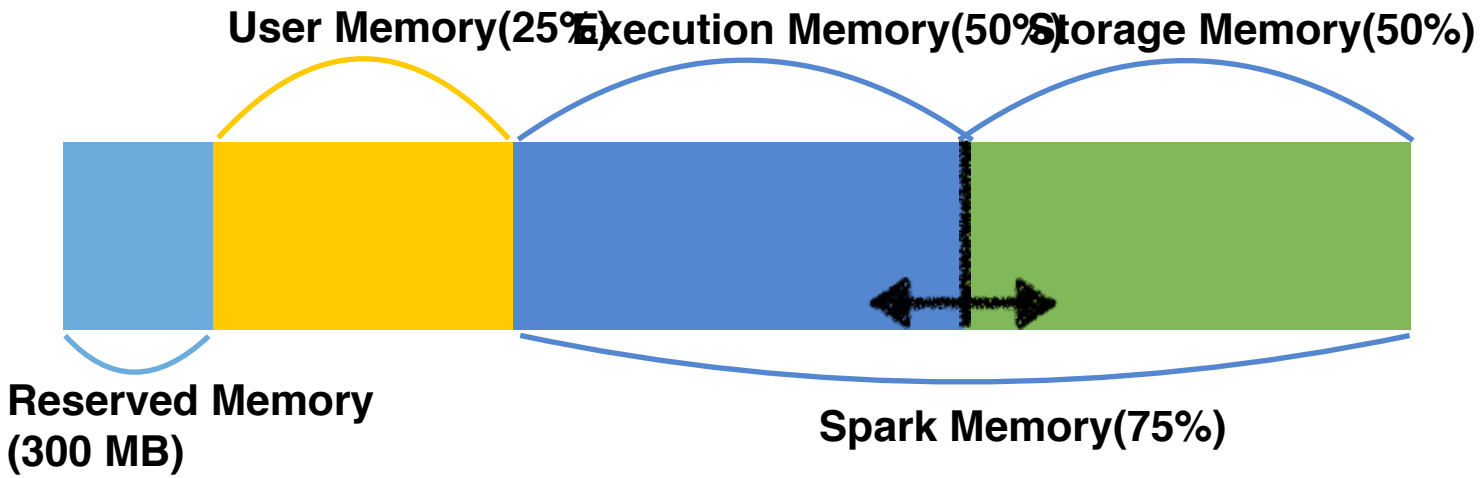
Spark Symptoms

If Spark is running with insufficient memory

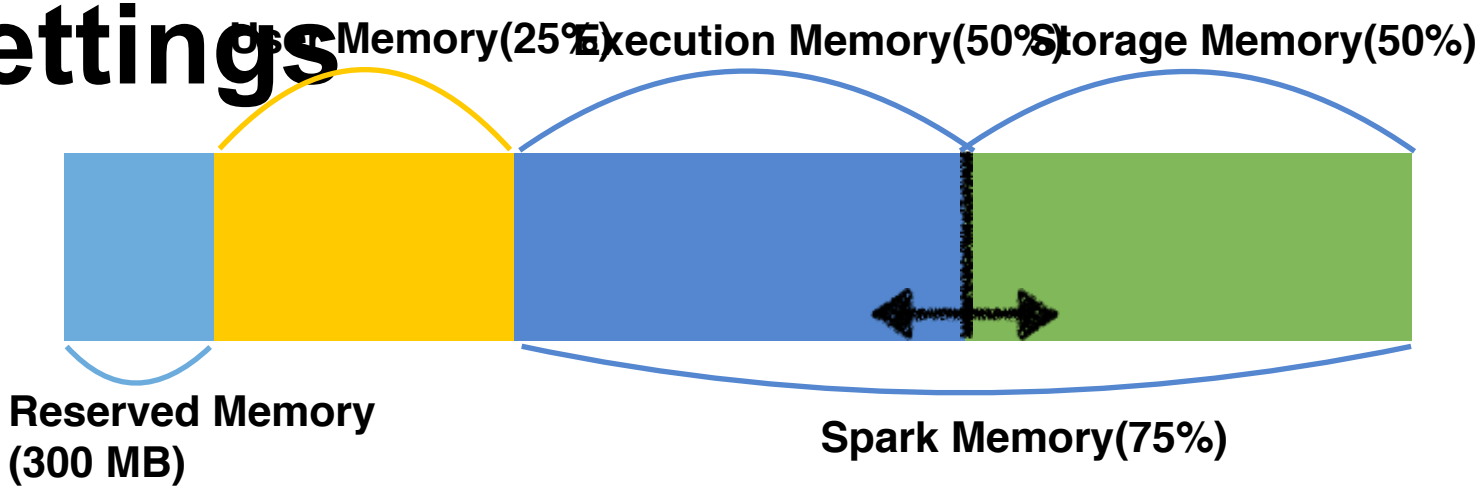
- Spark is running slow due to garbage collection
- Executor get lost due to Out-of-Memory (OOM) exception

Memory Management Model

Spark Memory Allocation



Memory Management Settings

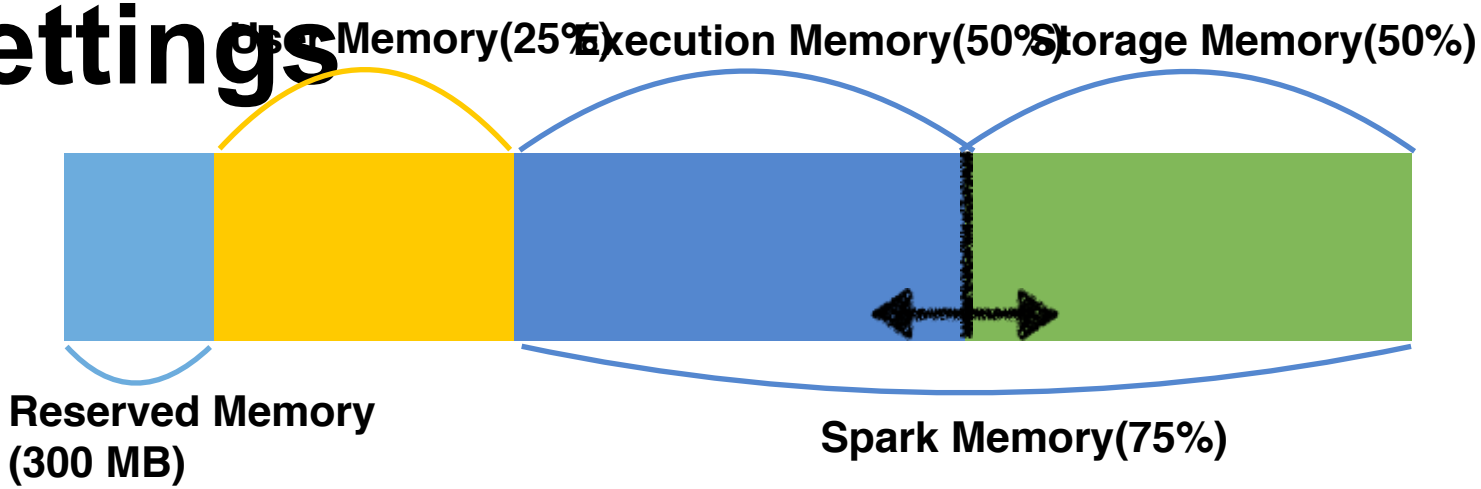


User Memory: users' data structures used in RDD transformations

Execution Memory: for Spark's internal storage of objects, e.g., shuffle buffer on the mapper

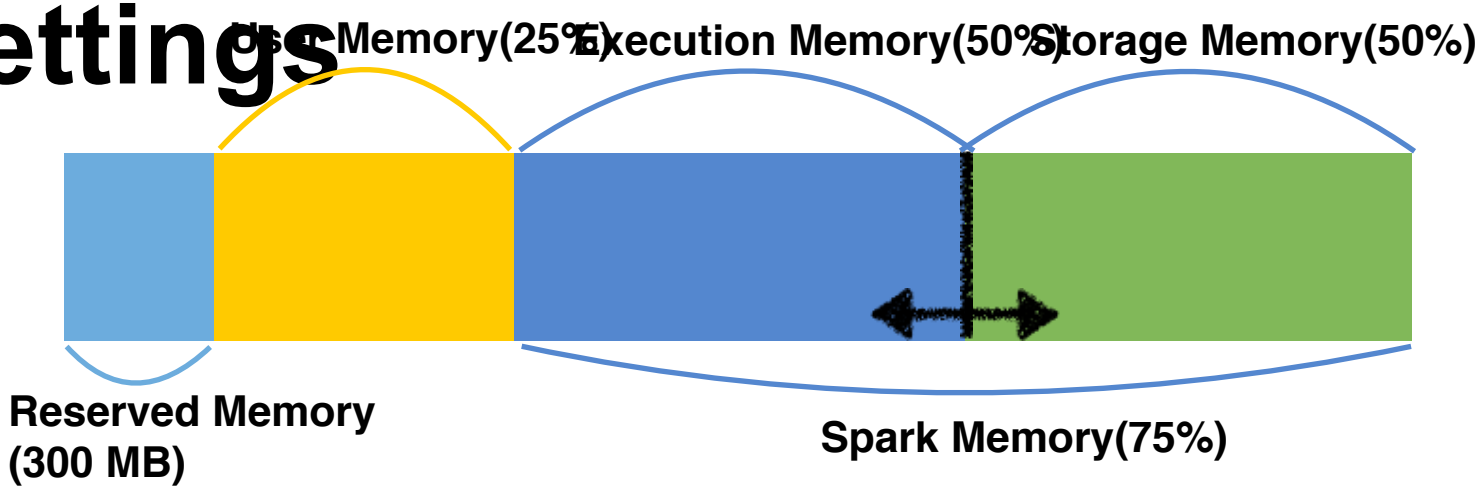
Storage Memory: for Spark's cached RDD, broadcast blocks, and serialized data unrolling

Memory Management Settings




```
val rdd =
    sc.parallelize(List(1,2,3,4,5,6),
    2)
val temp = rdd.mapPartitions(x
    => {
  val m = x.zipWithIndex.toMap
  m.toIterator
})
temp.cache
val res =
    temp.reduceByKey(_+_)
res.count
```

Memory Management Settings



In conf/spark-defaults.conf

spark.testing.reservedMemory — DO NOT USE

spark.memory.fraction (default 0.75)

spark.memory.storageFraction (dynamic, default 0.5)

Solutions to Symptoms

If Spark is running with insufficient memory
- Spark is running slow due to garbage collection
- Executor get lost due to Out-of-Memory (OOM) exception

The usual solution on Spark standalone cluster
- Allocate more memory for Spark executor by setting "**export SPARK_WORKER_MEMORY 96g**" in spark-env.sh
- Increase parallelism
  - Try more reducers by setting "**spark.default.parallelism**" in spark-defaults.conf
  - Set the number of partitions of the largest parent RDD (e.g., **sc.textFile(path, 128)**),
  - Set the second argument of the reduce function (e.g.,

Solutions to Symptoms

The usual solution on YARN cluster

Allocate more memory for Spark worker by passing "--executor-memory 2g" to the spark-submit command line

```
spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    lib/spark-examples*.jar \
    10
```

Self-checklist

How does Spark partition a DAG into stages of tasks?

How does Spark schedule tasks on a cluster?

How does Spark configure parallelism?

How does Spark executor manages its memory?

More Questions

Data Group @ TACC
data@tacc.utexas.edu

Weijia Xu
xwj@tacc.utexas.edu

Zhao Zhang
zzhang@tacc.utexas.edu

Thank You!

Spark SQL

- Similar to standard SQL
  - Supporting common syntax such as
    Select, group by, join, union etc.
- Has built in support of a list of aggregation function and keywords
  - E.g. count, distinct, avg, max, min etc.
- Support advanced functions
  - window … OVER
  - Create a window over rows and perform aggregations.
- Support common primitive types as column types as well as complex types.