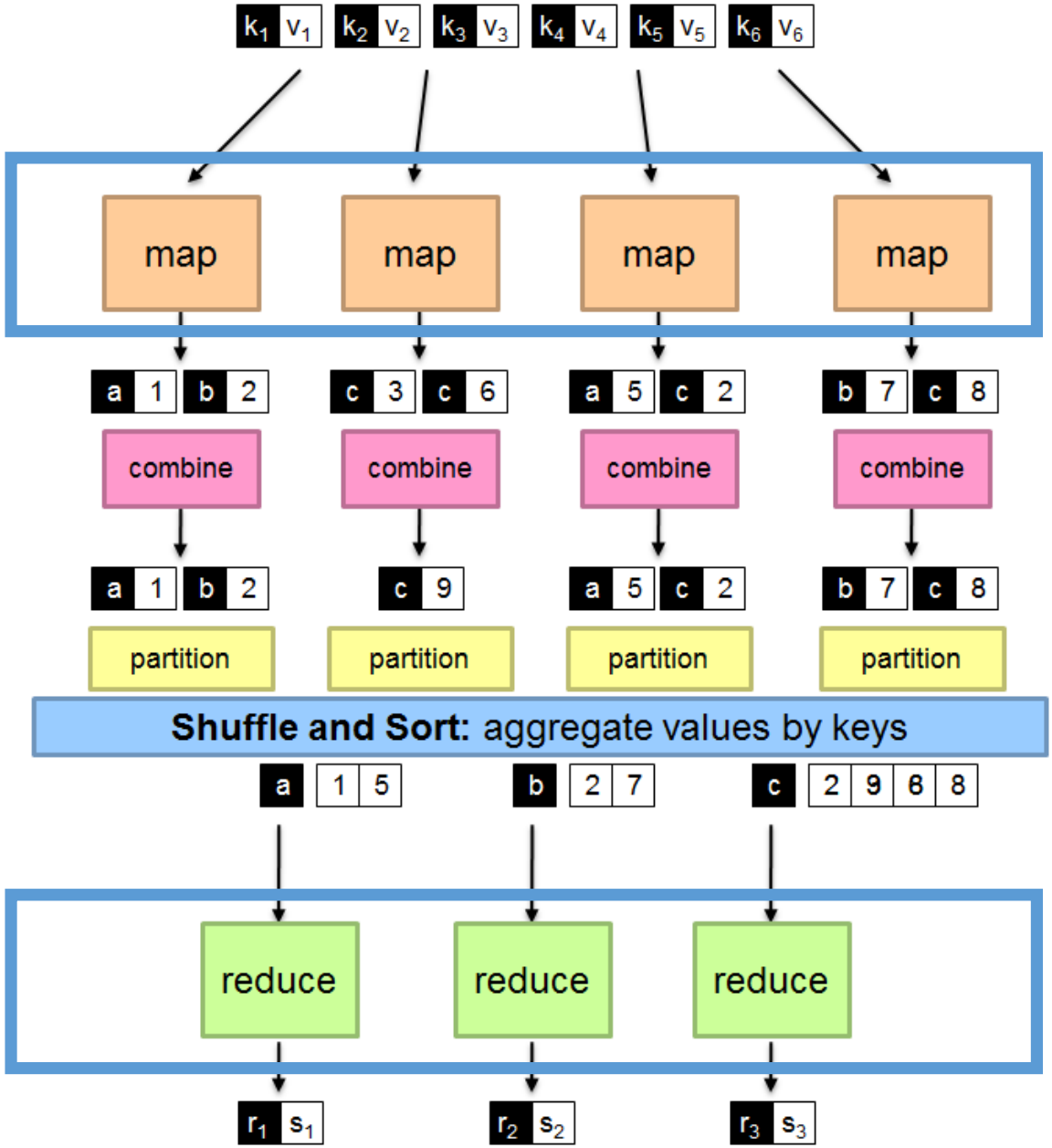
Costa Rica Big Data School: Large Scale Data Analysis with Spark

Weijia Xu
Research Scientist, Group Manager
Data Mining & Statistics
Texas Advanced Computing Center
University of Texas at Austin
Dec. 2018

TACC

Review of Day 1

- MapReduce programing model
  - Good fit for the big data problem
  - Unified data representation
  - Modularized data processing
  - Separate parallelism from computation by always moving computation to data
  - Elasticity
- Hadoop cluster
  - Popular backend for efficient large scale data storage and processing
  - Foundation to many big data applications with MapReduce or other models


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

From Hadoop to Spark

- Shortcomings of Hadoop and MapReduce
  - Disk based operations
    - All the intermediate steps are wrote onto the disk
    - Why?
  - Shuffle-sort between map and reduce process
    - Often becomes a bottleneck of the executions
  - Lacks of flexibilities for micro-grained parallelism
    - Map and reduce task assumes one core
    - High memory requirement may limit possibility of using all the cores available in the host
  - Best for batch simple disk based processing
    - Hard to be used for interactive analysis, streaming analysis …

From Hadoop to Spark

- Apache Spark is a cluster computing platform designed to be fast and general purpose.
  - Speed:
    - run computation in memory,
    - faster than MapReduce on disk
  - Generality:
    - supporting batch, interactive and streaming
  - Accessibility:
    - Simple APIs in Python, Java, Scala and SQL, and built-in libraries
  - Usability
    - Spark programming model is more than MapReduce model.

A Core Concept in Spark: RDD

- A Resilient Distributed Dataset (RDD),
  - the basic abstraction in Spark,
  - represents an immutable, partitioned collection
- An RDD contains a set of partitions
  - elements that can be operated on in parallel.
- An RDD is immutable
  - Immutable: value cannot be changed.
  - Why?

A Core Concept in Spark: RDD

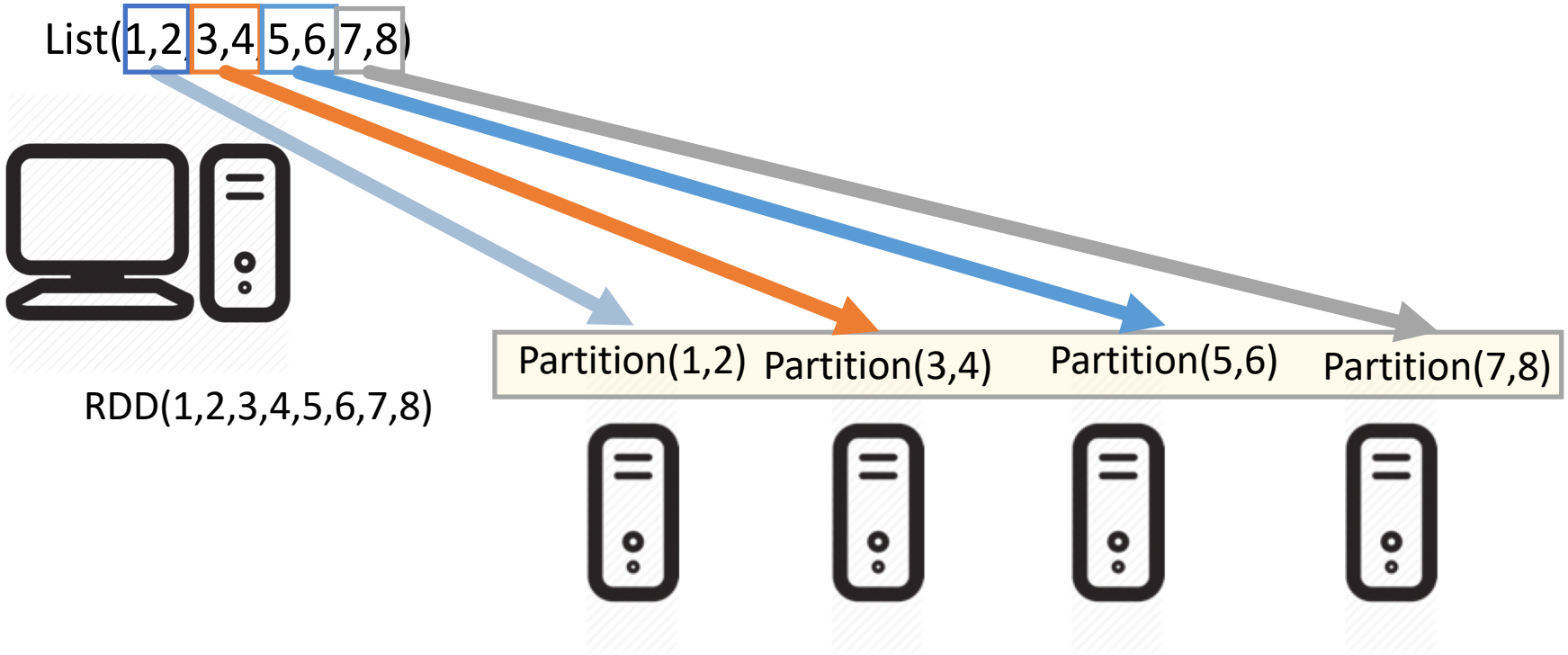
- Why RDD is immutable
  - Safe to share across multiple process
  - Easy to move around among resources, cache
  - Simplifies development

- RDD can be recreated at any time.
  - List of dependencies
  - Function to compute a partition given its parent.
  - Think RDD as a deterministic function rather than a data object.
  - Why?

RDD: From Centralized to Distributed Data Structure

Courtesy Image from https://cdn0.iconfinder.com/data/icons/hardware-outline-icons/60/Hardware_Hardware-01-512.pr

RDD: From Centralized to Distributed Data Structure



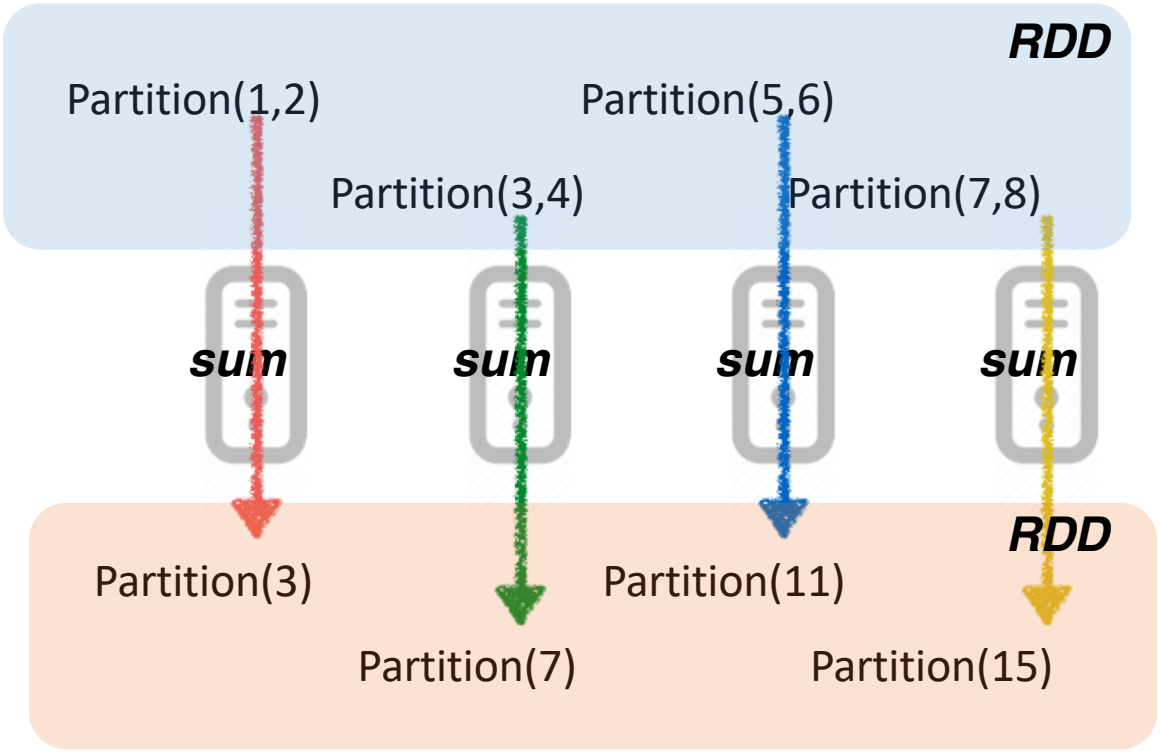
Courtesy Image from https://cdn0.iconfinder.com/data/icons/hardware-outline-icons/60/Hardware_Hardware-01-512.pr

RDD Processing

List(1,2,3,4,5,6,7,8)

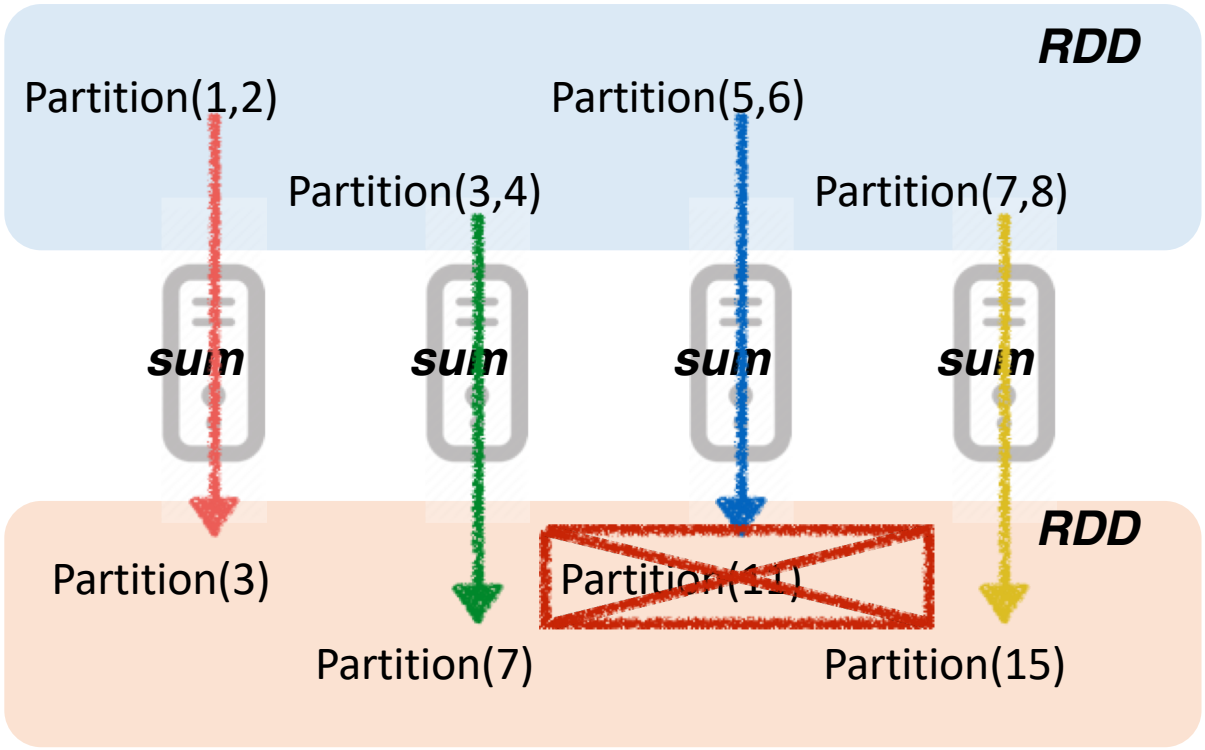
sum

List(3, 7, 11, 15)



Courtesy Image from https://cdn0.iconfinder.com/data/icons/hardware-outline-icons/60/Hardware_Hardware-01-512.pr

From Centralized to Distributed



Fault-tolerance: if the data on one computer gets lost, the system should be able to recover it

Courtesy Image from https://cdn0.iconfinder.com/data/icons/hardware-outline-icons/60/Hardware_Hardware-01-512.pr

Introduction of Zeppelin

- A web-based notebook that enables interactive data analytics and visualization.
- Multiple Language Backend

Starting Zeppelin on Wrangler

1. Connect to Wrangler
   - Open terminal window or application and run following:
   - ssh USERNAME@wrangler.tacc.utexas.edu
   - If succeed, you should see

```
login1.wrangler(1)$
```

   - You are now on the login node on wrangler
2. Then start Zeppelin session on compute node
   - sbatch --reservation=hadoop+TRAINING-OPEN+2526 /data/apps/zeppelin_user/job.zeppelin

```
login1.wrangler(2)$ sbatch --reservation=hadoop+TRAINING-OPEN+2552 /data/apps/zeppelin_user/job.zeppelin
```

   - If succeed, you should see

```
--> Checking available allocation (TRAINING-OPEN)...OK
Submitted batch job 98613
```

- Congratulation, You have started your first computing job on a cluster

Job scheduler Hadoop reservation job scripts

```
login1.wrangler(2)$ sbatch --reservation=hadoop+TRAINING-OPEN+2552 /data/apps/zeppelin_user/job.zeppelin

------------------------------------------------------------
          Welcome to the Wrangler Supercomputer
------------------------------------------------------------

--> Verifying valid submit host (login1)...OK
--> Verifying valid jobname...OK
--> Enforcing max jobs per user...OK
--> Verifying availability of your home dir (/home/0002/train292)...OK
--> Verifying availability of your work dir (/work/0002/train292/wrangler)...OK
--> Verifying valid ssh keys...OK
--> Verifying access to desired queue (hadoop)...OK
--> Verifying job request is within current queue limits...OK
--> Checking available allocation (TRAINING-OPEN)...OK
Submitted batch job 98613
```

Job ID

Zeppelin job

- The job request will start a Zeppelin session using one of nodes in Hadoop cluster
- Zeppelin is a web application, similar to Jupyter
- Zeppelin supports multiple kernels (programming language) within the same notebook
- Zeppelin serves as a GUI interface to the Hadoop cluster and support interactive analysis using Spark.

- You have sent the job request, as specified in the job script, to be run on compute nodes in the cluster.

- You can check your job (request) status with *'squeue'* command, e.g.
  - squeue -u train292 or squeue -j 98613

```
[login1.wrangler(3)$ squeue -u train292
             JOBID   PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             98613      hadoop zeppelin train292  R       0:28      1 c252-101
```

Job status
R Running
PD Pending

The request has to wait for available resource to run.
Since nodes in Hadoop cluster is limited, you may need team up.
If you have multiple jobs, please cancel one using *scancel JOB_ID*

Access your Zeppelin session

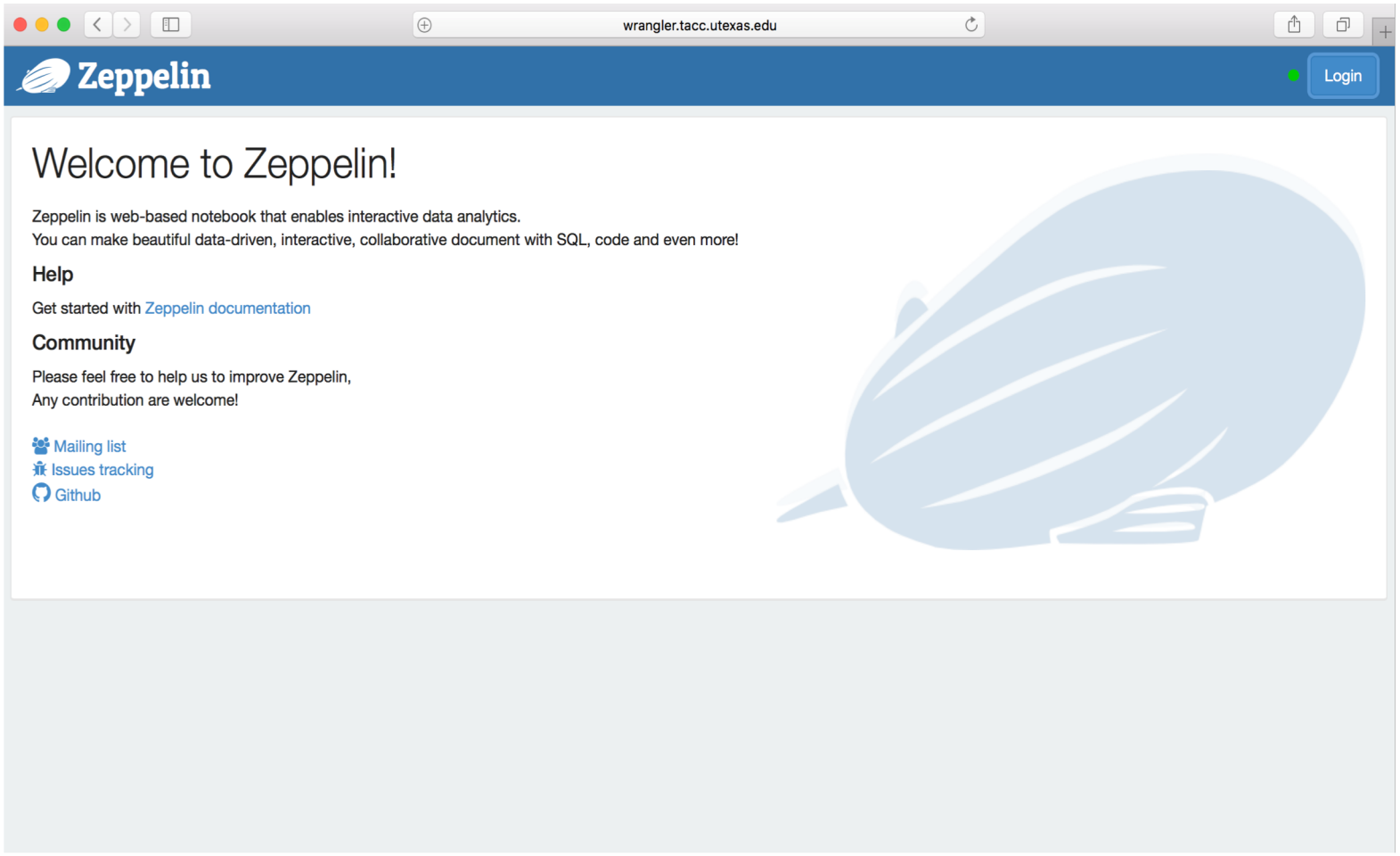
3. Find the port zeppelin is accessible with
   - cat zeppelin.out

```
login1.wrangler(6)$ cat zeppelin.out
```

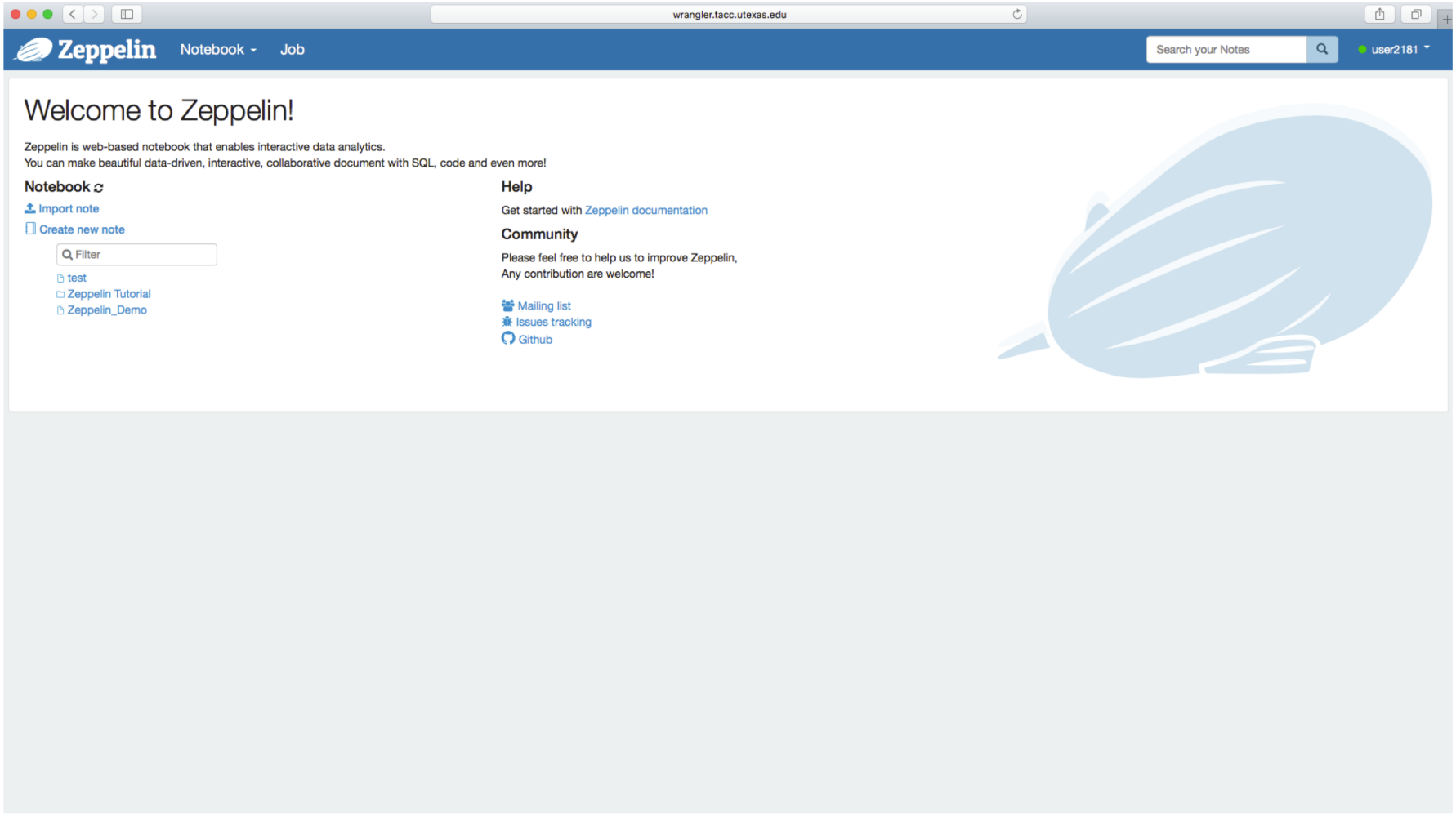
   - At the end of output you may see something like

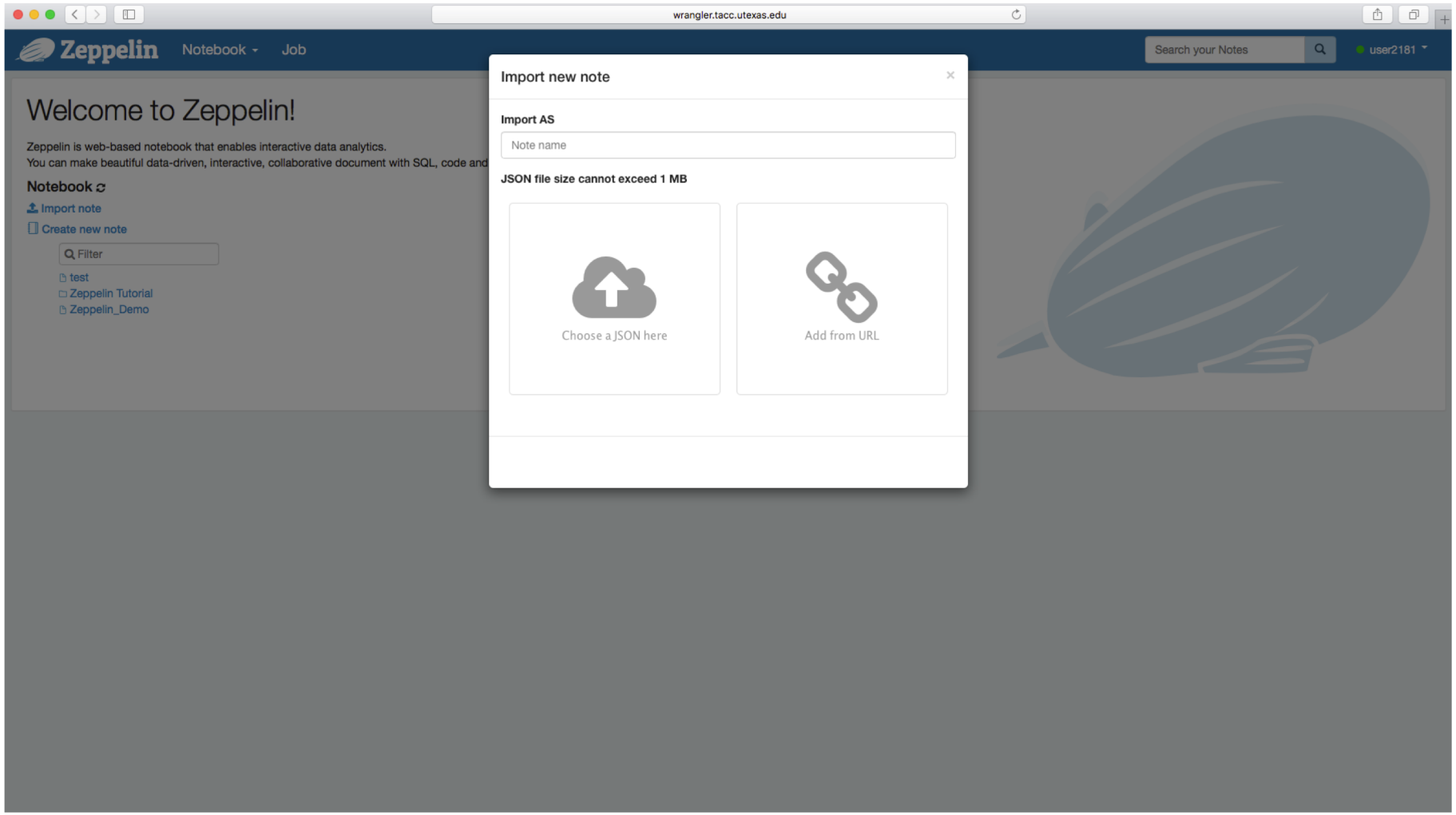
```
Your applicatin  is now running!
Application UI is at http://wrangler.tacc.utexas.edu:58091
Zeppelin username and password: use your TACC credential
```

   - Go to the URL using a **incongnito** web browser window
   - If nothing happened yet, just wait couple of more minutes.

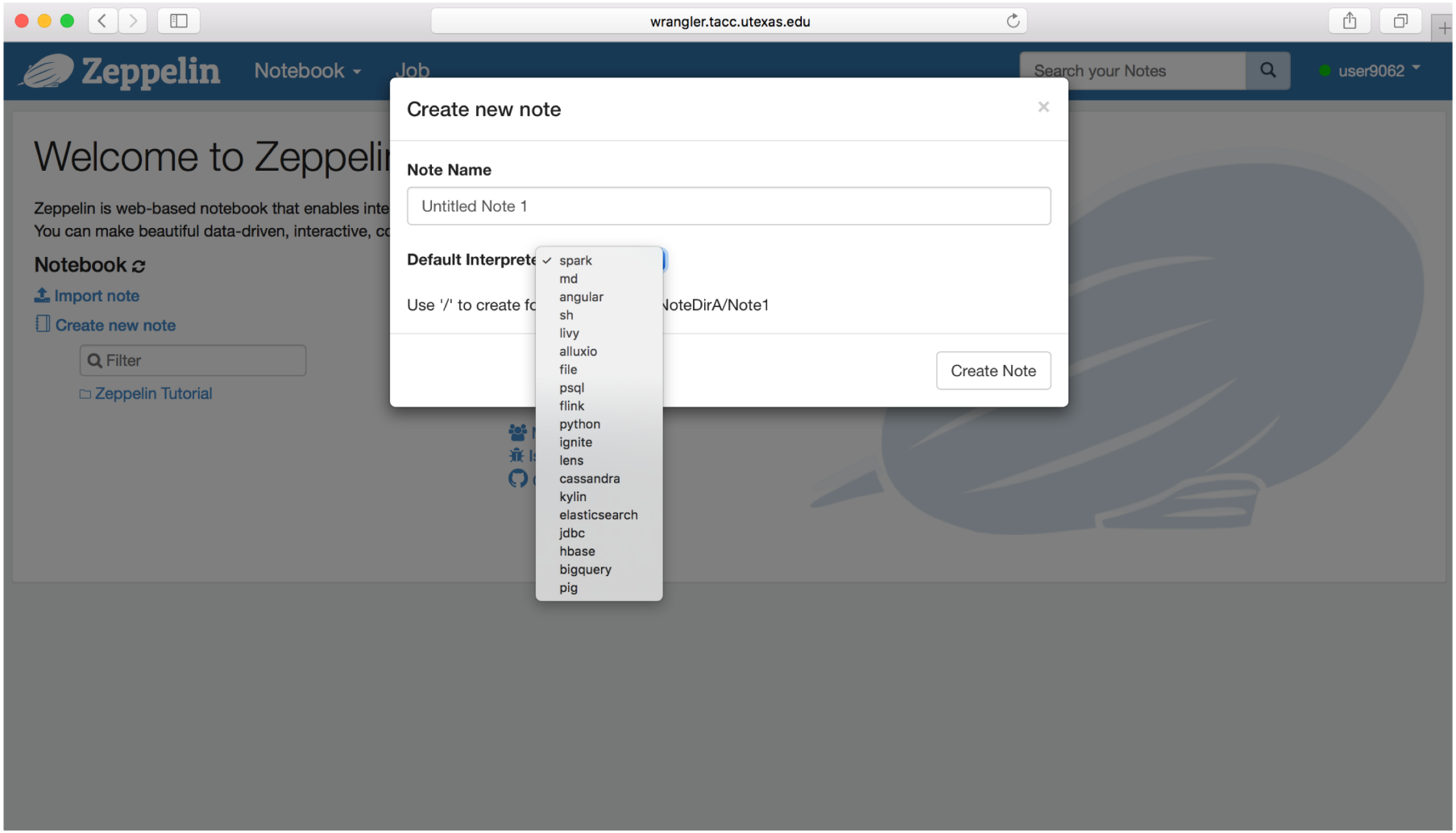

wrangler.tacc.utexas.edu
Zeppelin
Login
Welcome to Zeppelin!
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!
Help
Get started with Zeppelin documentation
Community
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!
Mailing list
Issues tracking
Github

TACC

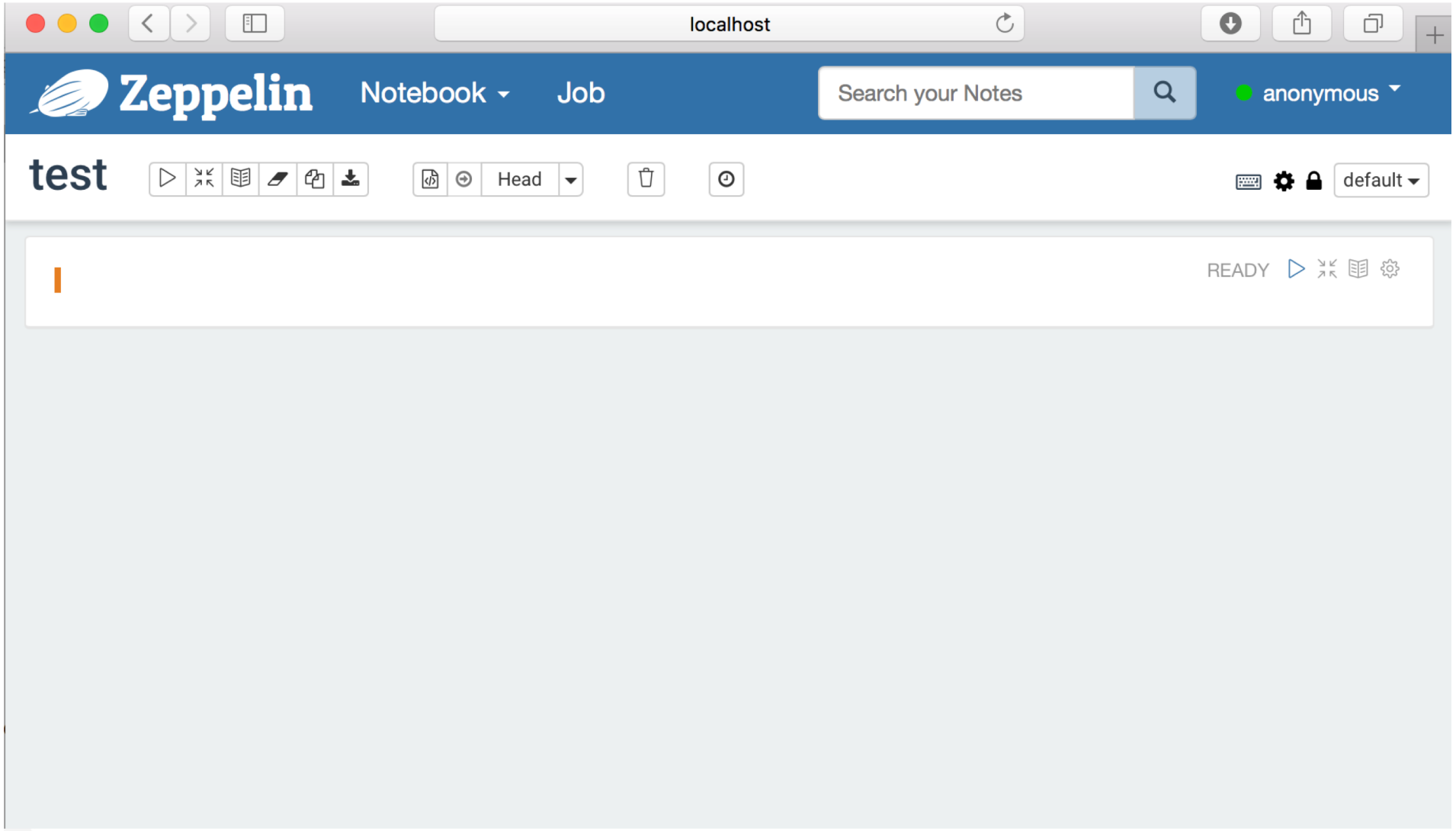

wrangler.tacc.utexas.edu
Zeppelin
Notebook
Job
Search your Notes
user2181
Welcome to Zeppelin!
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!
Notebook
Import note
Create new note
Filter
test
Zeppelin Tutorial
Zeppelin_Demo
Help
Get started with Zeppelin documentation
Community
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!
Mailing list
Issues tracking
Github


wrangler.tacc.utexas.edu
Zeppelin
Notebook
Job
Search your Notes
user2181
Welcome to Zeppelin!
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and
Notebook
Import note
Create new note
Filter
test
Zeppelin Tutorial
Zeppelin_Demo
Import new note
Import AS
Note name
JSON file size cannot exceed 1 MB
Choose a JSON here
Add from URL

TACC


wrangler.tacc.utexas.edu
Zeppelin
Notebook
Job
Search your Notes
user9062
Welcome to Zeppeli
Zeppelin is web-based notebook that enables inte
You can make beautiful data-driven, interactive, c
Notebook
Import note
Create new note
Filter
Zeppelin Tutorial
Create new note
Note Name
Untitled Note 1
Default Interprete
Use '/' to create fo
NoteDirA/Note1
Create Note
spark
md
angular
sh
livy
alluxio
file
psql
flink
python
ignite
lens
cassandra
kylin
elasticsearch
jdbc
hbase
bigquery
pig

TACC


localhost
Zeppelin
Notebook
Job
Search your Notes
anonymous
test
Head
default
READY

TACC

SCALA Programming Language and SPARK

- JVM based
  - More abstraction
  - More concise
- A functional programming language
- Enable scripting and interactive usage
- Used in Apache Spark and Apache Kafka
- https://www.scala-lang.org/

A Quick Tour of Scala

- Variable
  - Mutable and immutable variable
  - Primitive and composite variable
- Function
- Control flow

Variables

Mutable and immutable variables: ***val*** and ***var***

```
> val i = 5
> i: Int = 5
```

val variable is immutable

```
> i = 6
> error: reassignment to val
```

var variable is mutable

```
> var j = 5
> j = 6
```

Primitive Variables

Primitive Types:

Double, Float, Long, Int, Short, Byte

Char, Boolean, Unit

```
> val a = 5
> val a: Double = 5
```

Composite Variables

- Composite Types — Data Structures

  List, Map, Seq, Set, Tuple, String

- List is immutable

```
> val l = List(1,2,3)
> l(1)
> res0: Int = 2
> l(1) = 5
> error: value update is not a member of List[Int]
```

- What is immutable when we say a list is immutable?
  - The length of *l* ?
  - The elements of *l* ?

Composite Variables

Is List a type?

List[Int], List[Float], List[Double] are types

Scala uses type inference for missing type declarations

```
> val l = List(1,2,3)
> val l: List[Int] = List(1,2,3)
> val l: List[Double] = List(1,2,3)
> val l = List(1, 2.0, 3)    // what is the type of l ?
> l: List[Double] = List(1.0, 2.0, 3.0)
```

Composite Variables

List

- ()
- head
- tail
- last
- length
- map
- reverse
- sorted
- …

```
val l = List(3,1,2,4)
> l(1)
> l.head
> l.tail
> l.last
> l.length
> l.map(x=>x*2)
> l.reverse
> l.sorted
```

Google Scala API
http://www.scala-lang.org/api/2.11.8/#package

Composite Variables

Iterate a List(1,2,3) and multiply each element by 2

Using while:

```
> var r = ListBuffer[Int]()
> val l = List(1,2,3)
> var i = 0
> while (i<l.length){
>   r += l(i)*2
>   i += 1
> }
```

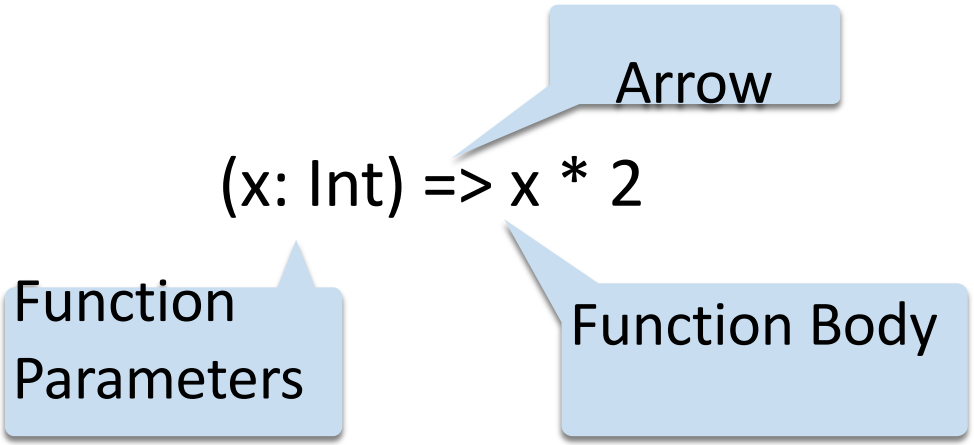
Using for:

```
> val l = List(1,2,3)
> val r = for (x <- l)
    yield(x*2)
```

Using map

```
> val l = List(1,2,3)
> val r = l.map(x => x*2)
```

Scala Functions



What if we have multiple statements in the body?

```
(x: Int) => {
  println(x)
  x * 2
}
```

Return Value

Scala Functions

(x: Int) => x * 2

```
val l = List(1,2,3)

l.map((x: Int) => x*2)
```

```
val l = List(1,2,3)

l.map(x => x*2)
```

```
val l = List(1,2,3)

l.map(_*2)
```

Place holder

Scala Functions

Multiple parameters?

(x: Int, y: Int) => x + y

```
val l = List((1,2),(3,4))
l.map(
  x: (Int, Int) => x._1+x._2
)
```

```
val l = List((1,2),(3,4))
l.map({
   case (x: Int, y:Int) => x+y
})
```

Pattern Matching Anonymous Function

Scala Functions

Multiple statements?

```
(x: Int, y: Int) => {
   println(x)
   println(y)
   x + y
}
```

Scala Functions

Multiple return values?

```
(x: Int, y: Int) => (x+3, y+5)
```

Return a tuple

Scala Functions

Give it a name

```
(x: Int) => x * x
```

Function Parameter

```
def func(x: Int) = x * x
```

Function Name

Function Body

```
def func(x: Int): Int = {
    println(x)
    x * x
}
```

Return Type

Return Value

Scala Control Flow

Iterate over List(1,2,3) and print out the elements in order

While:

```
> val l = List(1,2,3)
> var i = 0
> while(i < l.length){
>   println(l(i))
>   i += 1
> }
```

For:

```
> val l = List(1,2,3)
> for(i <- l)
>   print(i)
```

Foreach:

```
> val l = List(1,2,3)
> l.foreach(x =>
>   println(x))
```

Map:

```
> val l = List(1,2,3)
> val r = l.map(x =>
>   println(x))
```

Scala Control Flow

- if … else …
- Iterate over List(1,2,3) and print out the odd elements

```
> val l = List(1,2,3)
> l.foreach(x => {
>     if (x%2 == 1)
>         println(x)
> })
```

Scala Control Flow

- Pattern Matching

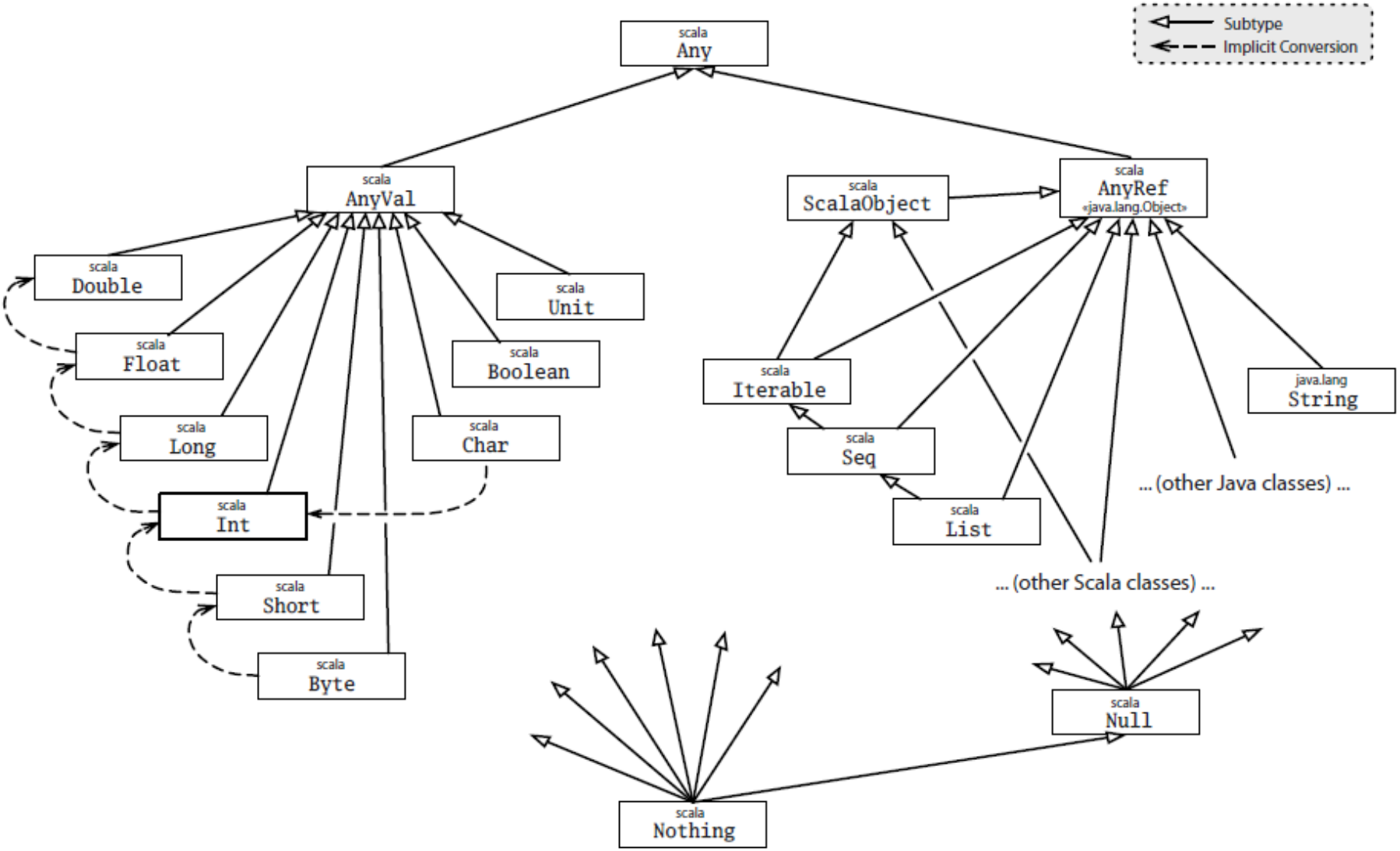
  match … case …

- e.g. find odd number in the list

```
> val l = List(1,2,3)
> l.foreach(x => {
>     if (x%2 == 1)
>         println(x)
> })
```

```
> val l = List(1,2,3)
> l.foreach(x => x%2 match{
>     case 1 => println(x)
>     case _ =>
> })
```

Scala Class Hierarchy



Scala Control Flow

- Pattern Matching — Types
- A function that can handles various parameter types

```
> def func(a: Any) = a match {
>     case i:Int => println(“a is an int”)
>     case f:Float => println(”a is a float”)
>     case d:Double => println(“a is a double”)
>     case s:String => println(“a is a string”)
>     case l: List[_] => println(“a is a list”)
>     case _ => println(“unknown type”)
> })
```

```
> func(1)
> func(1.0)
> func(1.toFloat)
> func(“abc”)
> func(List(1,2,3))
> func(Array(1,2,3))
```

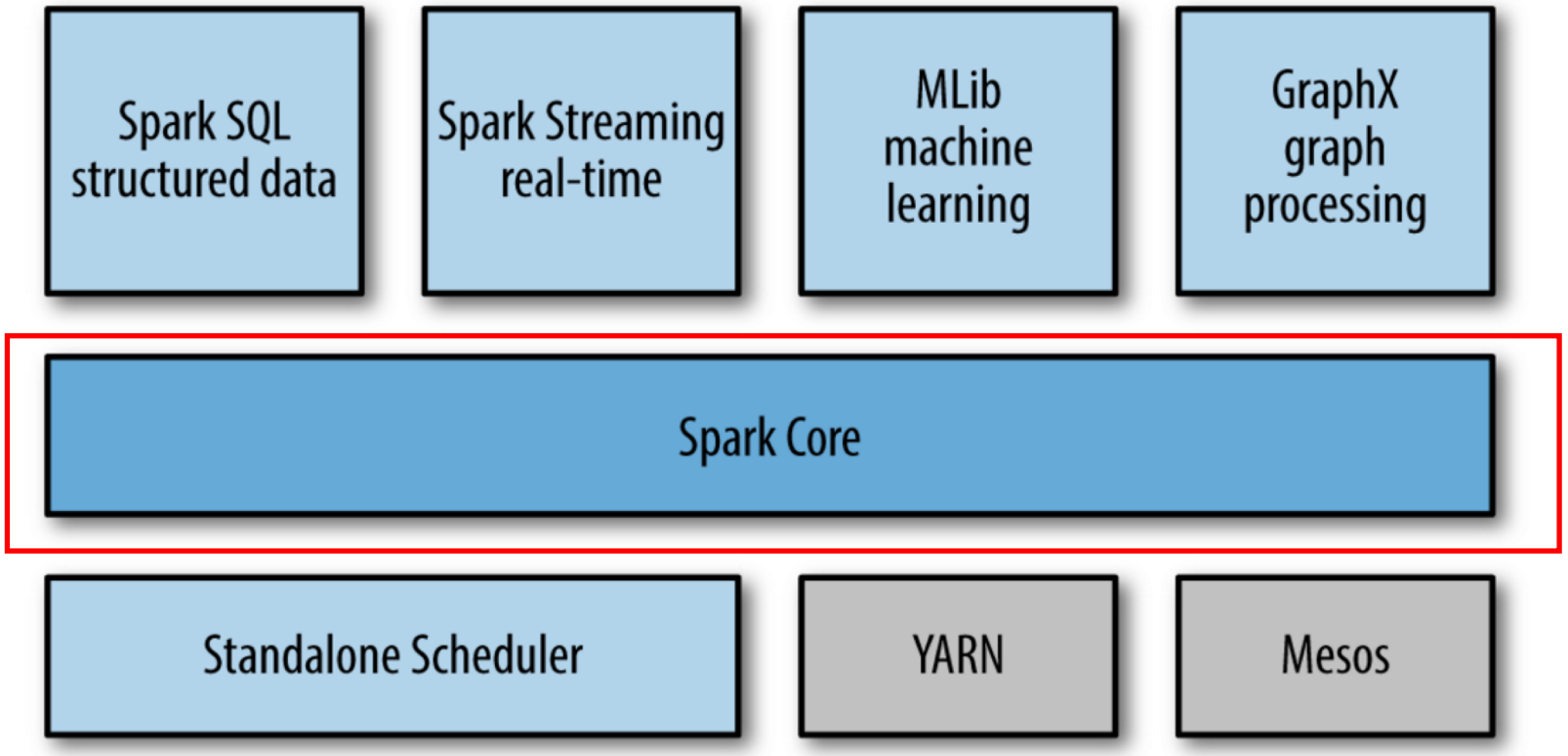
Recap: Scala

- Variable
  - var vs. val
  - Primitive vs. Composite
  - list, tuple
- Function
  - How to define one
  - How to name one
  - What are the input types
  - What are the return types
- Control flow
  - Loop, conditional, match

Exercise

- Try to implement how to calculate Pi with Scala

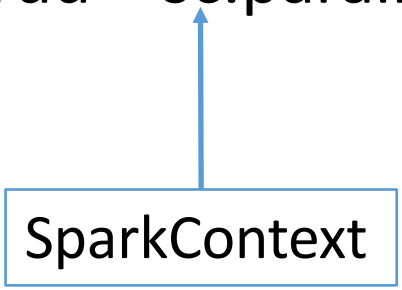
The Spark Stack



Spark RDD

- RDD is a basic data structure in Spark
- Create a new RDD

```
val rdd = sc.parallelize(List(0,1,…,99))
```



Lazy Evaluation in Spark

- Not all statements are immediate executed
- There are two types functions with RDD:

- Transformations
  - val res = rdd.map(x => x*2)
  - val res = rdd.filter(x => x%2 == 0)
  - val res = rdd.groupBy(x => x%2)
  - val lines = sc.textFile("path-to-file")
  - val rdd = sc.binaryFiles("path-to-file")
- Actions
  - res.count()
  - res.collect()
  - res.take()
  - res.reduce()
  - res.saveAsTextFiles("path-to-file")

RDD Transformations

- def map[U](f: (T) => U): RDD[U]
  - Return a new RDD by applying a function to all elements of this RDD
  - val rdd = sc.parallelize(0 until 100)
  - val res = rdd.map(x => x*2)
  - res.collect()

```
scala> res.collect
res0: Array[Int] = Array(0, 2, 4, 6, 8, 10,
12, 14, 16, … 198)
```

RDD Transformations

- def reduce(f: (T, T) ⇒ T): T
  - Reduces the elements of this RDD using the specified commutative and associative binary operator.
  - val rdd = sc.parallelize(0 until 10)
  - val res = rdd.reduce(_ + _)

```
scala> rdd.reduce(_+_)
res9: Int = 45
```

RDD Transformations

- def flatMap[U](f: (T) ⇒ TraversableOnce[U]): RDD[U]
  - Return a new RDD by first applying a function to all elements of this RDD, and then flattening the results
  - val rdd = sc.parallelize(List(1,2,3,4))
  - val res = rdd.flatMap(x => List(x, x, x))
  - res.collect()

```
scala> res.collect
res1: Array[Int] = Array(1, 1, 1, 2, 2,
2,3, 3, 3, 4, 4, 4)
```

RDD Transformations

- def mapPartitions[U](f: (Iterator[T]) ⇒ Iterator[U]): RDD[U]
  - Return a new RDD by applying a function to each partition of this RDD
  - What is an iterator?

  - val rdd = sc.parallelize(0 until 8, 2)
  - val res = rdd.mapPartitions(x => List(x.sum).iterator)
  - res.collect()

```
scala> res.collect
res5: Array[Int] = Array(6, 22)
```

RDD Transformations

- def filter(p: (A) ⇒ Boolean): List[A]
  - Selects all elements of this traversable collection which satisfy a predicate
  - val rdd = sc.parallelize(0 until 100)
  - val res = rdd.filter(x => x%2 == 0)
  - res.collect()

```
scala> res.collect
res4: Array[Int] = Array(0, 2, 4,…, 96, 98)
```

RDD Transformations

- groupBy[K](f: (T) ⇒ K): RDD[(K, Iterable[T])]
  - Return an RDD of grouped items. Each group consists of a key and a sequence of elements mapping to that key.
  - val rdd = sc.parallelize(0 until 100)
  - val res = rdd.groupBy(x => x%2 == 0)
  - res.collect()

```
scala> res.collect()
res7: Array[(Boolean, Iterable[Int])] =
Array((false,CompactBuffer(1, 3, …,
99)),(true,CompactBuffer(0, 2, 4, …, 98)))
```

RDD Transformations

- def zipWithIndex(): RDD[(T, Long)]
  - Zips this RDD with its element indices.
  - Use this method when you need index of the element with the value
  - val rdd = sc.parallelize(List("a", "b", "c", "d"))
  - val res = rdd.zipWithIndex()
  - res.collect()

```
scala> res.collect()
res6: Array[(String, Long)] = Array((a,0),
(b,1), (c,2), (d,3))
```

PairRDD Transformations

- Pair RDD assumes the elements are tuples [(K, V)]
- def groupByKey(): RDD[(K, Iterable[V])]
  - Group the values for each key in the RDD into a single sequence.
  - val l = List("a", "b", "a", "b").zipWithIndex
  - val rdd = sc.parallelize(l)
  - val res = rdd.groupByKey()
  - res.collect()

```
l: List[(String, Int)] = List((a,0), (b,1), (a,2), (b,3))

scala> res.collect()
res18: Array[(String, Iterable[Int])] =
Array((a,CompactBuffer(0, 2)), (b,CompactBuffer(1, 3)))
```

PairRDD Transformations

- def reduceByKey(func: (V, V) ⇒ V): RDD[(K, V)]
  - Merge the values for each key using an associative reduce function.
  - val l = List("a", "b", "a", "b").zipWithIndex
  - val rdd = sc.parallelize(l)
  - val res = rdd.reduceByKey(_ + _)
  - res.collect()

```
scala> res.collect()
res22: Array[(String, Int)] = Array((a,2), (b,4))
```

PairRDD Transformations

- def join[W](other: RDD[(K, W)]): RDD[(K, (V, W))]
  - Return an RDD containing all pairs of elements with matching keys in this and other
  - val person = List("adam", "ben", "chris", "david")
  - val age = List(27, 42, 53, 23)
  - val dept = List("HPC", "Data", "Vis", "Edu")
  - val rdd1 = sc.parallelize(person.zip(age))
  - val rdd2 = sc.parallelize(person.zip(dept))
  - val res = rdd1.join(rdd2)
  - res.collect()

```
res.collect()
res30: Array[(String, (Int, String))] = Array(
(ben,(42,Data)), (david,(23,Edu)),
(chris,(53,Vis)), (adam,(27,HPC)))
```

Recap: Spark RDD

- Action
  - Count, collect, take, reduce, saveAsText
- Transformation
  - filter, map, reduce,
  - flatmap, mapPartition,
  - zipWithIndex, groupBy
- Pair RDD transformation
  - groupByKey, reduceByKey, join
- Lazy Evaluation

Exercise

- Implement calculating Pi with Spark.

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

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	68.562868489320
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.