

@h2oai & @mmalohlava
presents

Sparkling Water

Meetup

Spark  + H₂O

SPARKLING
WATER



H₂O

User-friendly API for data transformation

Large and active community

Platform components - SQL

Multitenancy

Memory efficient

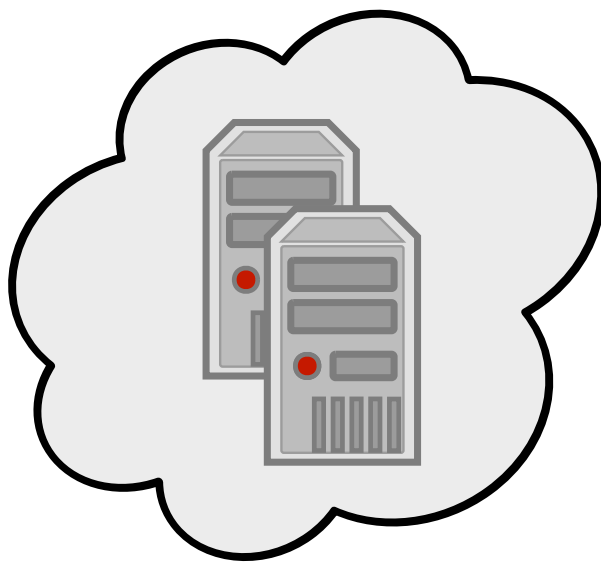
Performance of computation

Machine learning algorithms

Parser, GUI, R-interface

Sparkling Water

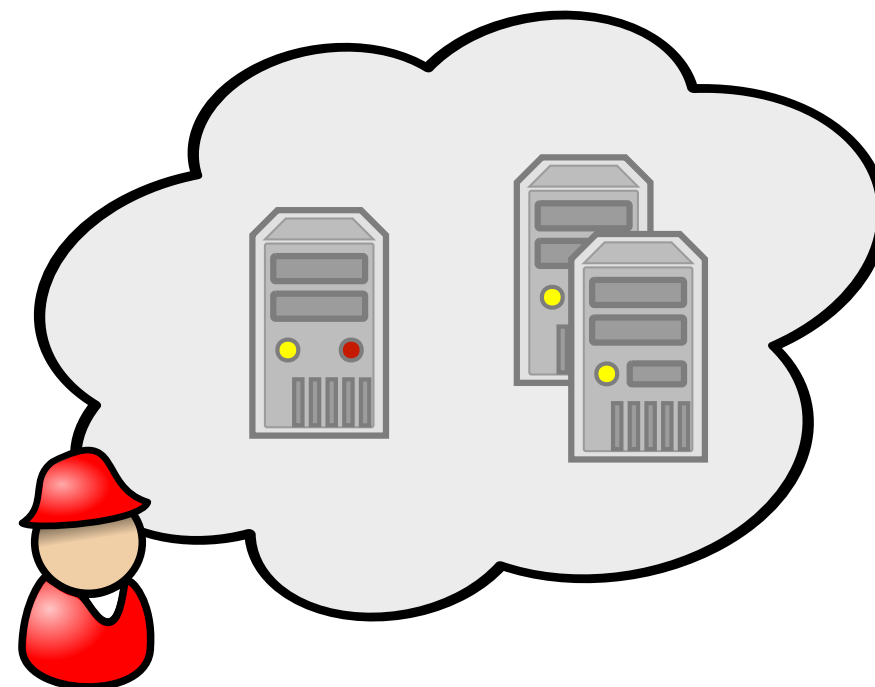
Spark



RDD
immutable
world

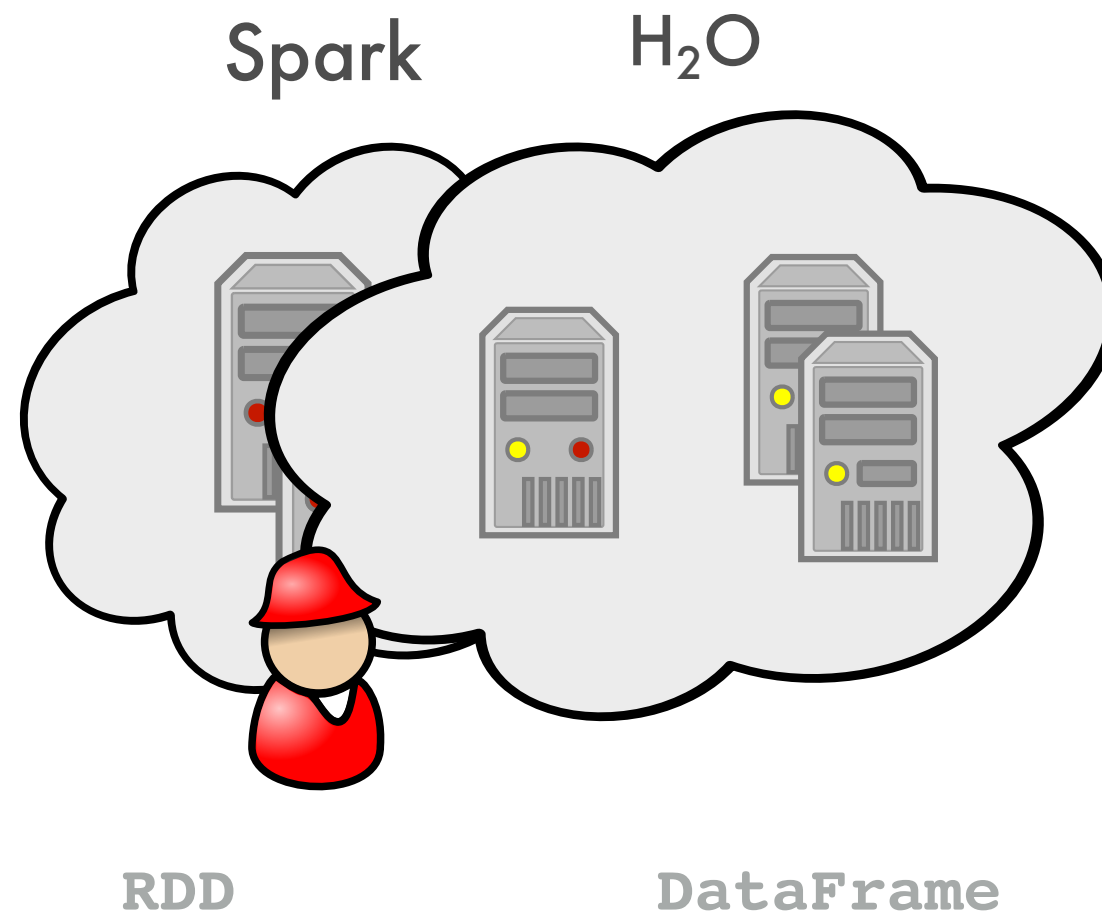


H₂O



DataFrame
mutable
world

Sparkling Water



Sparkling Water

Provides

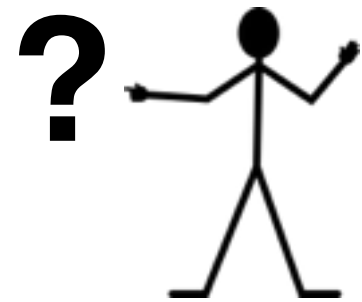
Transparent integration into Spark ecosystem

Pure **H2ORDD** encapsulating H₂O **DataFrame**

Transparent use of **H₂O data structures and algorithms** with Spark API

Excels in Spark workflows requiring advanced Machine Learning algorithms

Sparkling Water Design



implements

Sparkling
App

spark-submit

Spark
Master
JVM

Spark
Worker
JVM

Spark
Worker
JVM

Spark
Worker
JVM

Sparkling Water Cluster

Spark
Executor
JVM

H2O

Spark
Executor
JVM

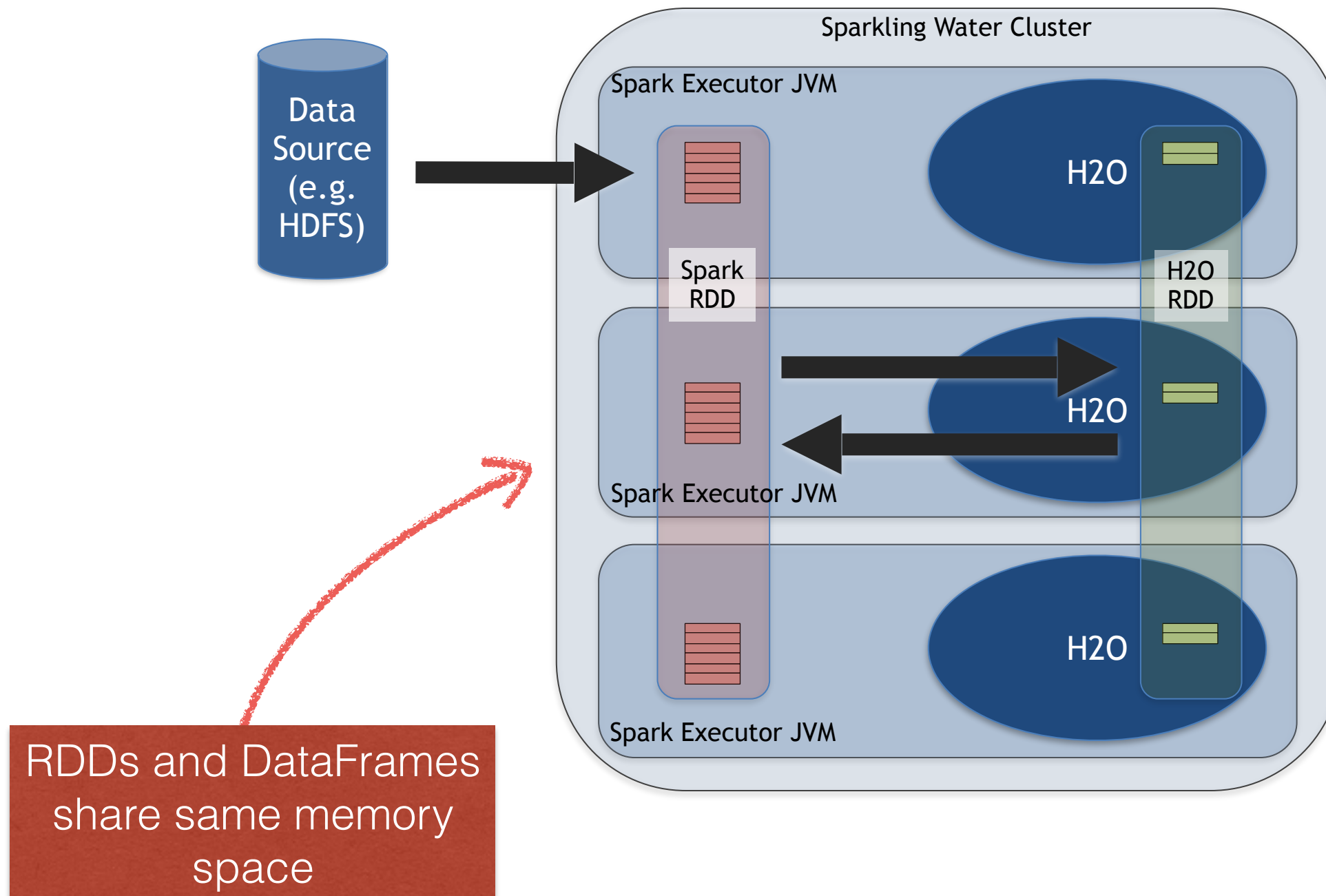
H2O

Spark
Executor
JVM

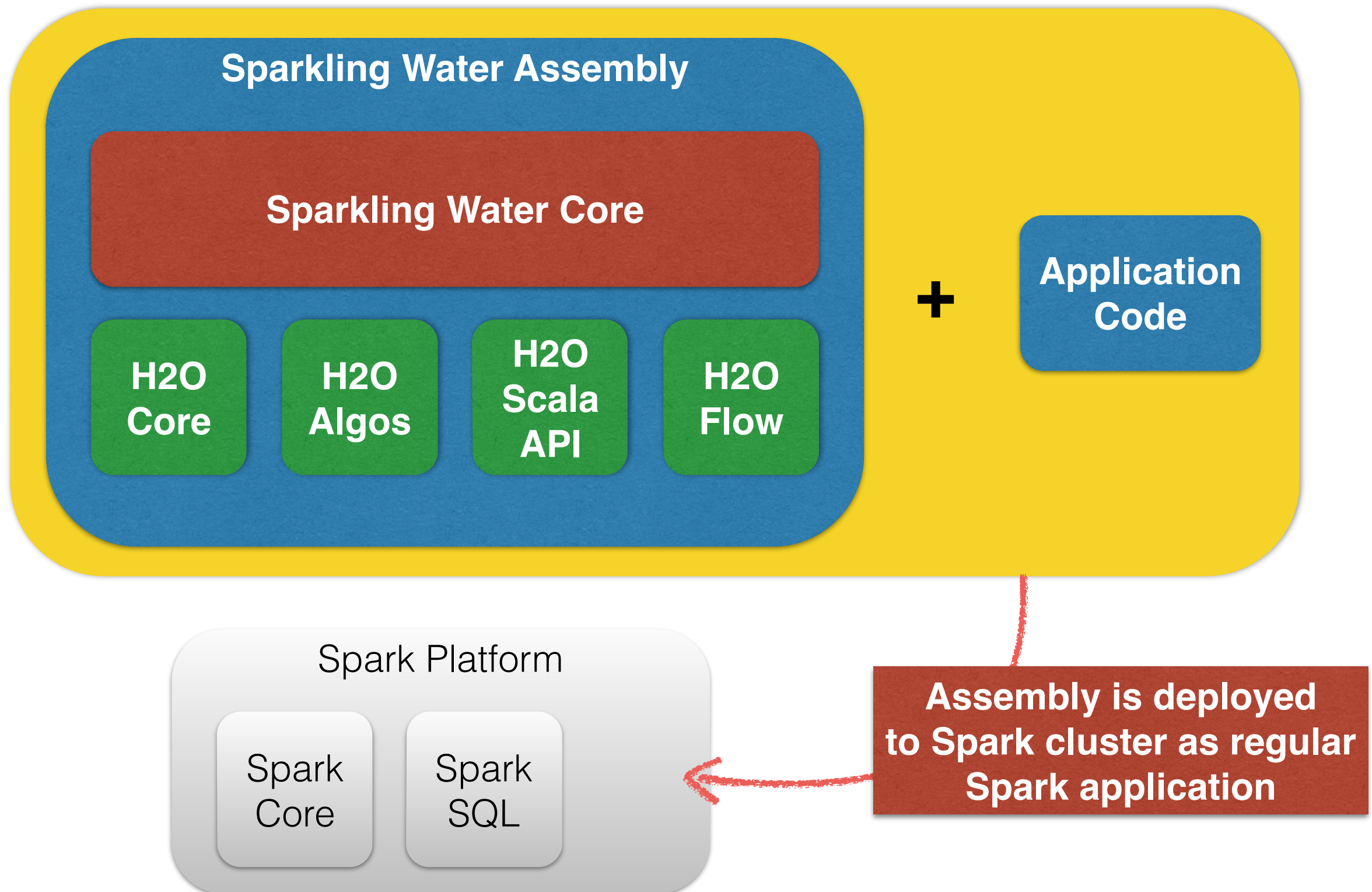
H2O

Contains application
and Sparkling Water
classes

Data Distribution



Devel Internals



Hands-On #1

Sparkling Shell

Sparkling Water Requirements

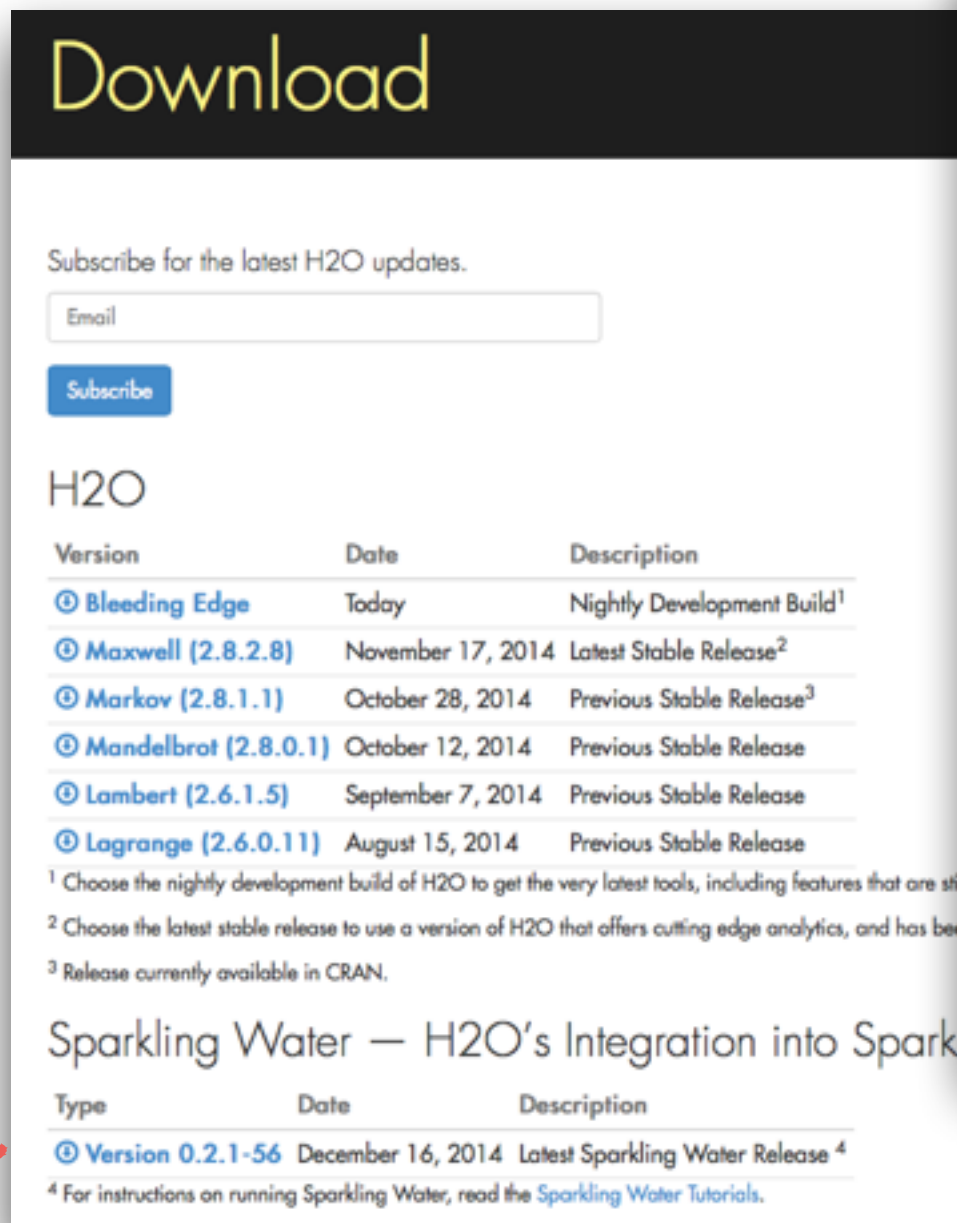
Linux or Mac OS X

Oracle Java 1.7+

Spark 1.1.0

Download

<http://h2o.ai/download/>



The screenshot shows the 'Download' section of the H2O website. It features a subscription form for H2O updates, a table of H2O versions, and a section for Sparkling Water. A red arrow points from the 'Download' text in the main image to the 'Download' header of this screenshot.

Download

Subscribe for the latest H2O updates.

Email

[Subscribe](#)

H2O

Version	Date	Description
Bleeding Edge	Today	Nightly Development Build ¹
Maxwell (2.8.2.8)	November 17, 2014	Latest Stable Release ²
Markov (2.8.1.1)	October 28, 2014	Previous Stable Release ³
Mandelbrot (2.8.0.1)	October 12, 2014	Previous Stable Release
Lambert (2.6.1.5)	September 7, 2014	Previous Stable Release
Lagrange (2.6.0.11)	August 15, 2014	Previous Stable Release

¹ Choose the nightly development build of H2O to get the very latest tools, including features that are still in development.

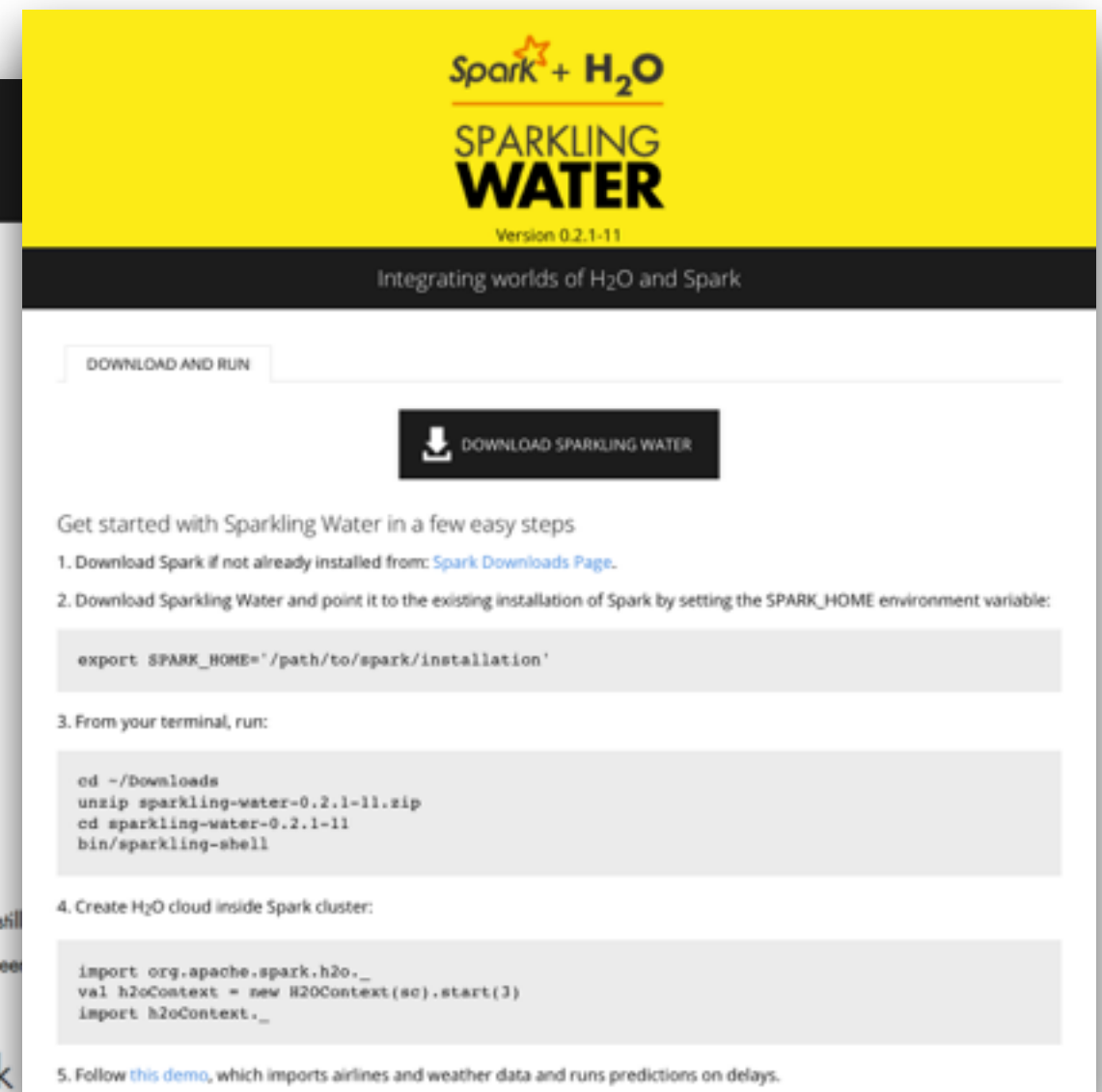
² Choose the latest stable release to use a version of H2O that offers cutting edge analytics, and has been tested for stability.

³ Release currently available in CRAN.

Sparkling Water — H2O's Integration into Spark

Type	Date	Description
Version 0.2.1-56	December 16, 2014	Latest Sparkling Water Release ⁴

⁴ For instructions on running Sparkling Water, read the [Sparkling Water Tutorials](#).



The screenshot shows the 'Spark + H2O SPARKLING WATER' section of the website. It includes a 'DOWNLOAD AND RUN' button, a 'DOWNLOAD SPARKLING WATER' button, and a list of steps to get started with Sparkling Water. A red arrow points from the 'Download' text in the main image to the 'Download Sparkling Water' button.

Spark + H₂O

SPARKLING WATER

Version 0.2.1-11

Integrating worlds of H2O and Spark

[DOWNLOAD AND RUN](#)

[DOWNLOAD SPARKLING WATER](#)

Get started with Sparkling Water in a few easy steps

- Download Spark if not already installed from: [Spark Downloads Page](#).
- Download Sparkling Water and point it to the existing installation of Spark by setting the SPARK_HOME environment variable:

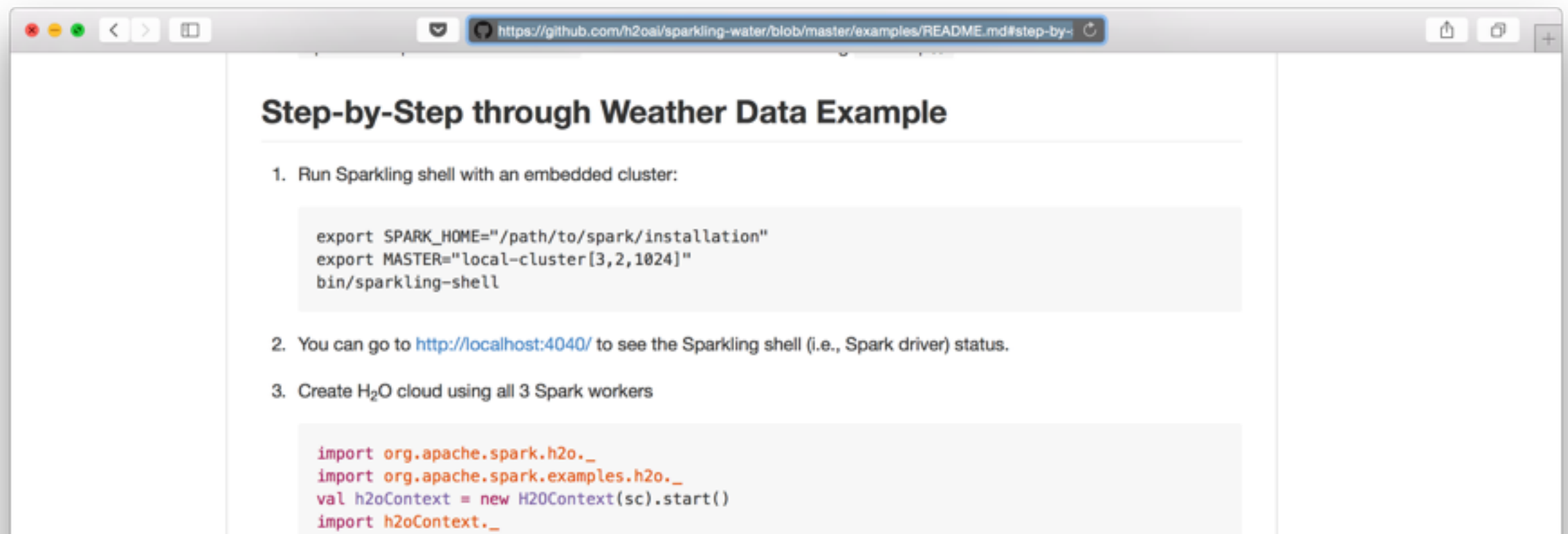
```
export SPARK_HOME="/path/to/spark/installation"
```
- From your terminal, run:

```
cd ~/Downloads
unzip sparkling-water-0.2.1-11.zip
cd sparkling-water-0.2.1-11
bin/sparkling-shell
```
- Create H2O cloud inside Spark cluster:

```
import org.apache.spark.h2o._
val h2oContext = new H2OContext(sc).start()
import h2oContext._
```
- Follow [this demo](#), which imports airlines and weather data and runs predictions on delays.

Where is the code?

<https://github.com/h2oai/sparkling-water/blob/master/examples/scripts/>



Flight delays prediction

“Build a model using weather and flight data to predict delays of flights arriving to Chicago O’Hare International Airport”

Example Outline

Load & Parse CSV data from 2 data sources

Use Spark API to filter data, do SQL query for join

Create regression models

Use models to predict delays

Graph residual plot from R

Install and Launch

Unpack zip file

and

Point SPARK_HOME to your Spark 1.1.0 installation

and

Launch `bin/sparkling-shell`

What is Sparkling Shell?

Standard **spark-shell**

With additional Sparkling Water classes

```
export MASTER="local-cluster[3,2,1024]"
```

```
spark-shell \  
  --jars sparkling-water.jar
```

Spark Master
address



JAR containing
Sparkling
Water



**Lets play with Sparkling
shell...**

Create H₂O Client

Contains implicit utility functions

Demo specific classes

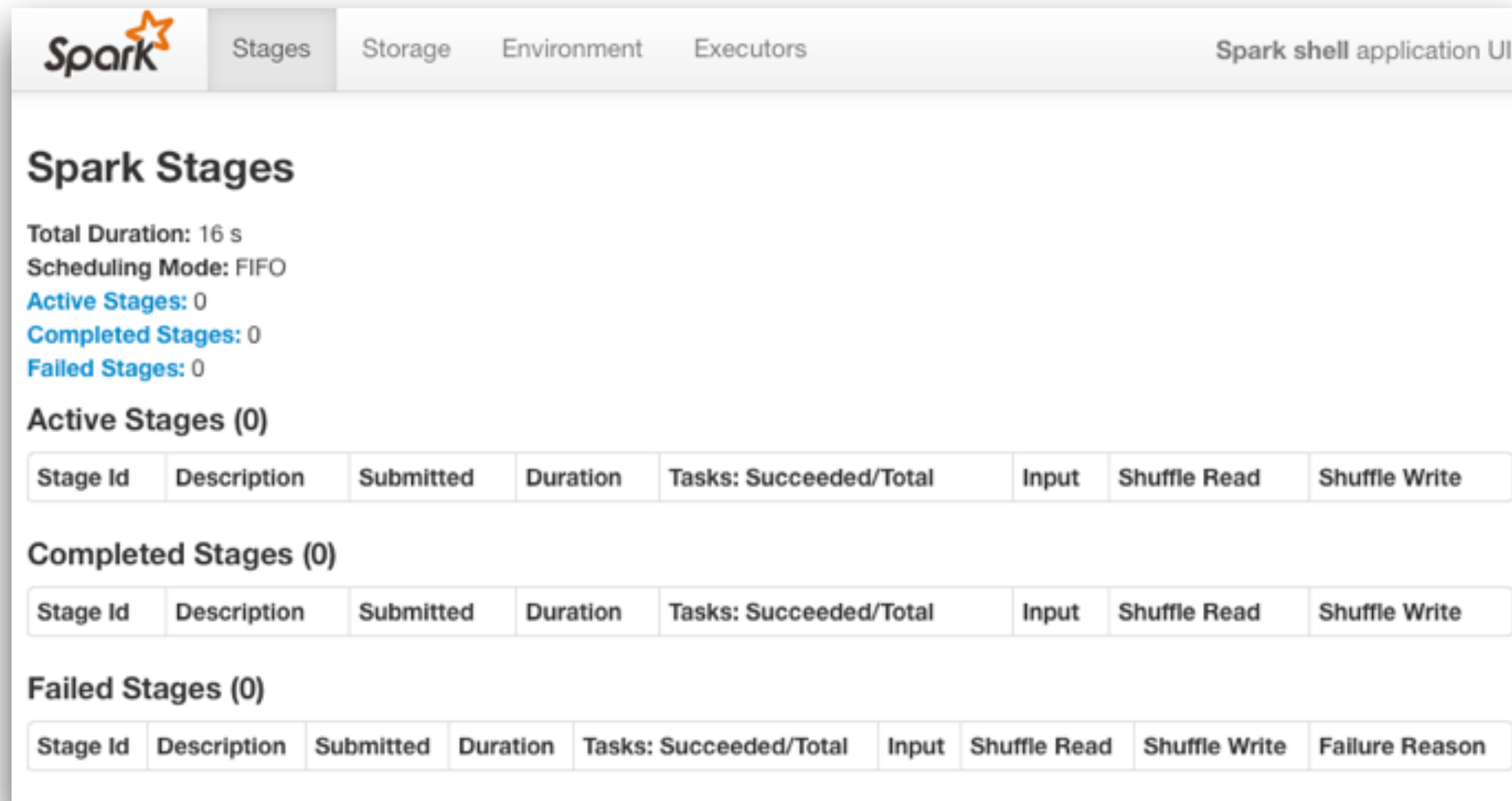
Size of demanded H₂O cloud

```
import org.apache.spark.h2o._  
import org.apache.spark.examples.h2o._  
  
val h2oContext = new H2OContext(sc).start(3)  
import h2oContext._
```

Regular Spark context
provided by Spark shell

Is Spark Running?

Go to <http://localhost:4040>



The screenshot shows the Spark Stages UI. At the top, there is a navigation bar with the Spark logo and tabs for Stages, Storage, Environment, and Executors. The Stages tab is selected. The main content area is titled 'Spark Stages' and displays summary statistics: Total Duration: 16 s, Scheduling Mode: FIFO, Active Stages: 0, Completed Stages: 0, and Failed Stages: 0. Below these statistics, there are three sections: 'Active Stages (0)', 'Completed Stages (0)', and 'Failed Stages (0)'. Each section contains a table with columns for Stage Id, Description, Submitted, Duration, Tasks: Succeeded/Total, Input, Shuffle Read, and Shuffle Write. The 'Failed Stages' table also includes a 'Failure Reason' column. All tables are currently empty.

Spark Stages

Total Duration: 16 s
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 0
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
----------	-------------	-----------	----------	------------------------	-------	--------------	---------------

Completed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write
----------	-------------	-----------	----------	------------------------	-------	--------------	---------------

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Shuffle Read	Shuffle Write	Failure Reason
----------	-------------	-----------	----------	------------------------	-------	--------------	---------------	----------------

Is H₂O running?

<http://localhost:54321/flow/index.html>

The screenshot displays the H2O Flow web interface in a browser window. The address bar shows `localhost:54321/flow/index.html`. The interface includes a menu bar with `Flow`, `Edit`, `View`, `Format`, `Run`, and `Help`. Below the menu, there's a toolbar with icons for creating, saving, and running flows. The main workspace is titled "Untitled Flow" and contains an "Expression..." input field. A help dialog is open, titled "Help" and "Using Flow for the first time?", providing instructions on how to use the interface. The background shows system metrics for two nodes, including CPU usage, memory, and disk space.

H₂O FLOW Flow Edit View Format Run Help

Untitled Flow

Expression...

Help

Using Flow for the first time?

Assist Me!

H₂O Flow is a web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document, much like [IPython Notebooks](#).

Flow is a modal editor, which means that you are either in edit mode or command mode. A Flow is composed of a series of executable cells. Each cell has an input and one or more outputs.

To get started, just memorize these six simple keyboard shortcuts: `enter`, `esc`, `ctrl+enter`, `a`, `b` and `d`.

- To edit a cell, press `enter` to get into edit mode.
- When you're done editing, press `esc` to get back into command mode.
- To execute a cell, press `ctrl+enter`.
- `a` adds a new cell above the current cell.

Connections: 0

16.2.223:54323 a few seconds ago

LOAD 4.792

DATA

% Cached

GC

465.12 GB (17%) % Free

172.16.2.223:54325 a few seconds ago

LOAD 4.792

DATA

% Cached

GC

81.08 GB / 465.12 GB (17%) % Free

CPU

RPCs 0

Threads

Tasks

Load Data #1

Load weather data into RDD

```
val weatherDataFile =  
  "examples/smalldata/weather.csv"
```

```
val wrawdata = sc.textFile(weatherDataFile, 3)  
                .cache()
```

```
val weatherTable = wrawdata  
  .map(_.split(","))  
  .map(row => WeatherParse(row))  
  .filter(!_isWrongRow())
```

Regular Spark API



Ad-hoc Parser



Weather Data

```
case class Weather( val Year      : Option[Int],  
                    val Month     : Option[Int],  
                    val Day       : Option[Int],  
                    val TmaxF     : Option[Int], // Max temperatur in F  
                    val TminF     : Option[Int], // Min temperatur in F  
                    val TmeanF    : Option[Float], // Mean temperatur in F  
                    val PrcpIn    : Option[Float], // Precipitation (inches)  
                    val SnowIn    : Option[Float], // Snow (inches)  
                    val CDD       : Option[Float], // Cooling Degree Day  
                    val HDD       : Option[Float], // Heating Degree Day  
                    val GDD       : Option[Float]) // Growing Degree Day
```



Simple POJO to hold one row of weather data

Load Data #2

Load flights data into H2O frame

```
import java.io.File
```

```
val dataFile =  
    "examples/smalldata/year2005.csv.gz"
```

```
val airlinesData = new DataFrame(new File(dataFile))
```



Shortcut for data load
and parse

Where is the data?

Go to <http://localhost:54321/flow/index.html>

The screenshot shows the H2O Flow web interface in a browser. The address bar displays `localhost:54321/flow/index.html`. The interface includes a menu bar with options like Flow, Edit, View, Format, Run, and Help. Below the menu, there's a toolbar with icons for adding, deleting, and saving flows. The main content area displays a data table for a file named `year2005.hex`. The table has columns for LABEL, MISSING, ZEROS, PINFS, NINFS, MIN, MAX, MEAN, SIGMA, TYPE, CARDINALITY, PRECISION, and ACTIONS. The data rows include various flight-related attributes such as Year, Month, DayofMonth, DayOfWeek, DepTime, CRSDepTime, ArrTime, CRSArrTime, UniqueCarrier, FlightNum, TailNum, ActualElapsedTime, CRSElapsedTime, AirTime, ArrDelay, DepDelay, Origin, Dest, Distance, TaxiIn, TaxiOut, Cancelled, and CancellationCode. Each row has a 'Summary...' link in the ACTIONS column. On the right side, there's a sidebar with tabs for OUTLINE, FLOWS, CLIPS, and HELP. The CLIPS tab is active, showing 'My Clips' and 'System (8)' sections. The 'My Clips' section contains instructions on how to save and execute clips. The 'System (8)' section lists various system operations like assist, importFiles, getFrames, getModels, getPredictions, getJobs, buildModel, and predict. At the bottom right, there's a 'Trash' section with instructions on how to delete cells. The status bar at the bottom shows 'Connections: 0'.

LABEL	MISSING	ZEROS	PINFS	NINFS	MIN	MAX	MEAN	SIGMA	TYPE	CARDINALITY	PRECISION	ACTIONS
Year					2005	2005	2005		Int	-	-1	Summary...
Month					1	12	6.48414	3.4094317272311785	Int	-	-1	Summary...
DayofMonth					1	31	15.75905	8.794331498193229	Int	-	-1	Summary...
DayOfWeek					1	7	3.93476	1.9948241866483505	Int	-	-1	Summary...
DepTime	1830	1830			1	2647	1345.4830090659061	477.61665093706125	Int	-	-1	Summary...
CRSDepTime					8	2359	1339.2692	465.18911178192417	Int	-	-1	Summary...
ArrTime	2026	2026			1	2735	1492.8678935227713	500.2999107184945	Int	-	-1	Summary...
CRSArrTime		2				2359	1500.80398	481.21381231577834	Int	-	-1	Summary...
UniqueCarrier		9320				19	10.78603	6.230538904079296	enum	20	-1	Summary...
FlightNum					1	9584	2040.50077	1843.255624093514	Int	-	-1	Summary...
TailNum		973				4899	2537.59825	1421.0382209156921	enum	4900	-1	Summary...
ActualElapsedTime	2026	2026			16	1621	124.96217363790393	71.70160232529602	Int	-	-1	Summary...
CRSElapsedTime					19	660	126.13038	70.03463392889893	Int	-	-1	Summary...
AirTime	2026	2026			-1420	1583	101.48011717394411	83.23174396230985	Int	-	-1	Summary...
ArrDelay	2026	5846			-68	1625	7.08004164370139	33.91493836410402	Int	-	-1	Summary...
DepDelay	1830	19626			-1191	1639	8.524264031781604	31.692212029843702	Int	-	-1	Summary...
Origin		58				280	137.55906	75.21116804779798	enum	281	-1	Summary...
Dest		58				280	138.02559	75.31124466497435	enum	281	-1	Summary...
Distance					31	4962	725.8401	573.482730722045	Int	-	-1	Summary...
TaxiIn		2053				1460	7.57375	44.09759842575039	Int	-	-1	Summary...
TaxiOut		1840				251	15.42969	10.651373198861354	Int	-	-1	Summary...
Cancelled		98170				1	0.0183	0.13403465840176484	Int	-	-1	Summary...
CancellationCode	98170	98962				3	0.7448087431693989	0.7408363762609962	enum	4	-1	Summary...

Use Spark API for Data Filtering

Create a cheap wrapper around H₂O DataFrame

```
// Create RDD wrapper around DataFrame  
val airlinesTable : RDD[Airlines]  
    = asRDD[Airlines](airlinesData)
```

```
// And use Spark RDD API directly  
val flightsToORD = airlinesTable  
    .filter( f => f.Dest == Some("ORD") )
```

Regular Spark
RDD call

Use Spark SQL to Data Join

```
import org.apache.spark.sql.SQLContext
// We need to create SQL context
implicit val sqlContext = new SQLContext(sc)
import sqlContext._

flightsToORD.registerTempTable("FlightsToORD")
weatherTable.registerTempTable("WeatherORD")
```



Make context implicit to share it with h2oContext

Join Data based on Flight Date

```
val joinedTable = sql(
  """SELECT
    | f.Year, f.Month, f.DayofMonth,
    | f.CRSDepTime, f.CRSArrTime, f.CRSElapsedTime,
    | f.UniqueCarrier, f.FlightNum, f.TailNum,
    | f.Origin, f.Distance,
    | w.TmaxF, w.TminF, w.TmeanF,
    | w.PrcpIn, w.SnowIn, w.CDD, w.HDD, w.GDD,
    | f.ArrDelay
    | FROM FlightsToORD f
    | JOIN WeatherORD w
    | ON f.Year=w.Year AND f.Month=w.Month
    |    AND f.DayofMonth=w.Day""".stripMargin)
```

Split data

```
import hex.splitframe.SplitFrame
import hex.splitframe.SplitFrameModel.SplitFrameParameters

val sfParams = new SplitFrameParameters()
sfParams._train = joinedTable
sfParams._ratios = Array(0.7, 0.2)
val sf = new SplitFrame(sfParams)

val splits = sf.trainModel().get._output._splits
val trainTable = splits(0)
val validTable = splits(1)
val testTable = splits(2)
```

Result of
SQL query is
implicitly
converted
into H2O
DataFrame

Launch H₂O Algorithms

```
import hex.deeplearning._
import hex.deeplearning.DeepLearningModel
    .DeepLearningParameters

// Setup deep learning parameters
val dlParams = new DeepLearningParameters()
dlParams._train = trainTable
dlParams._response_column = 'ArrDelay'
dlParams._valid = validTable
dlParams._epochs = 100
dlParams._reproducible = true
dlParams._force_load_balance = false

// Create a new model builder
val dl = new DeepLearning(dlParams)

val dlModel = dl.trainModel.get
```



Make a prediction

```
// Use model to score data  
val dlPredictTable = dlModel.score(testTable)('predict')  
  
// Collect predicted values via RDD API  
val predictionValues = toShemaRDD(dlPredictTable)  
                        .collect  
                        .map (row =>  
  if (row.isNullAt(0))  
    Double.NaN  
  else  
    row(0))
```

Hands-On #2

Can I access results from R?

YES!

Requirements

R 3.1.2+

RStudio

H2O R package

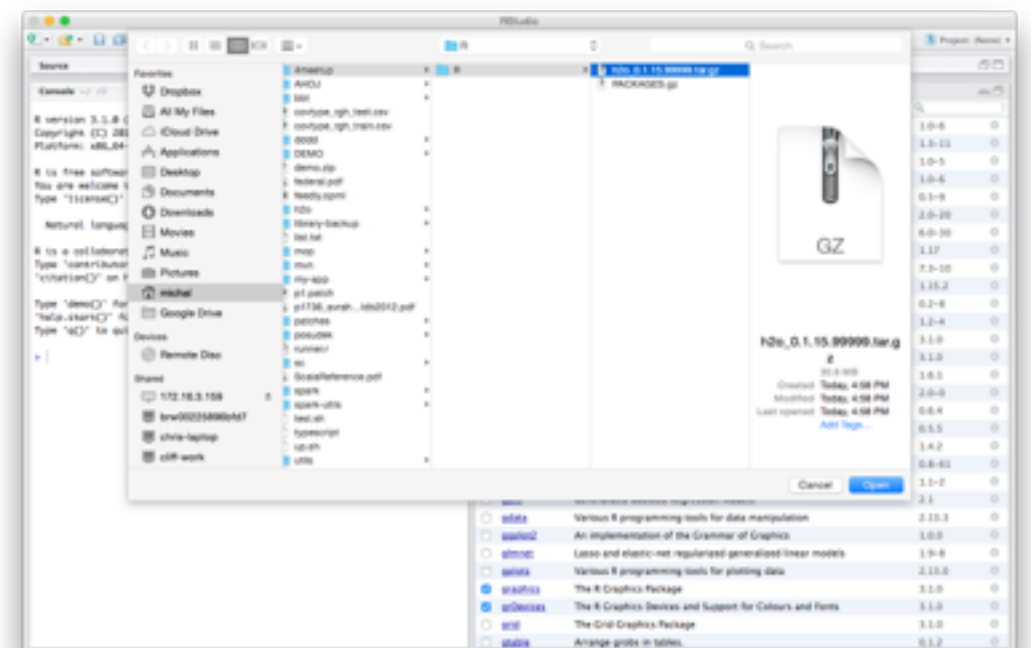
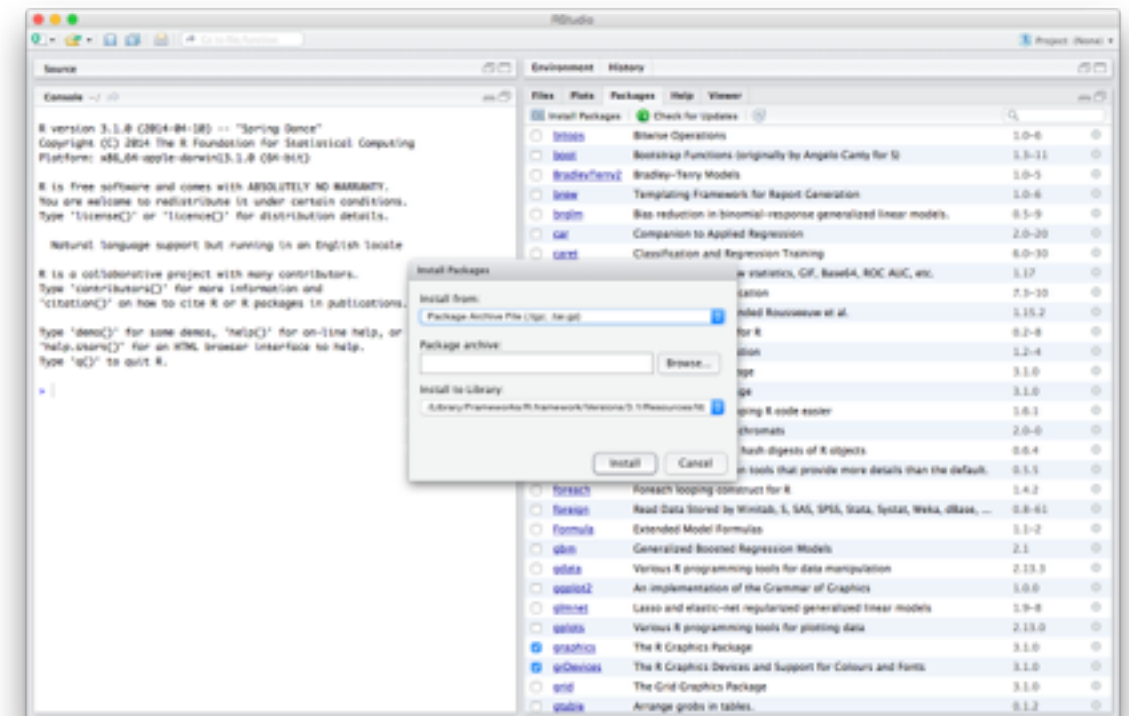
Install R package

**You can find R package
on USB stick**

1. Open RStudio

**2. Click on
“Install Packages”**

**3. Select
h2o_0.1.20.99999.tar.gz
file from USB**

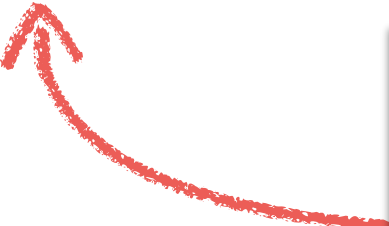


Generate R code

In Sparkling Shell:

```
import org.apache.spark.examples.h2o.DemoUtils.residualPlotRCode
```

```
residualPlotRCode(  
  predictionH2OFrame, 'predict',  
  testFrame, 'ArrDelay')
```



Utility generating
R code to show
residuals plot for
predicted and actual
values

Residuals Plot in R

```
# Import H2O library and initialize H2O client
library(h2o)
```

```
h = h2o.init()
```

```
# Fetch prediction and actual data, use remembered keys
pred = h2o.getFrame(h, "dframe_b5f449d0c04ee75fda1b9bc865b14a69")
act = h2o.getFrame(h, "frame_rdd_14_b429e8b43d2d8c02899ccb61b72c4e57")
```

```
# Select right columns
predDelay = pred$predict
actDelay = act$ArrDelay
```

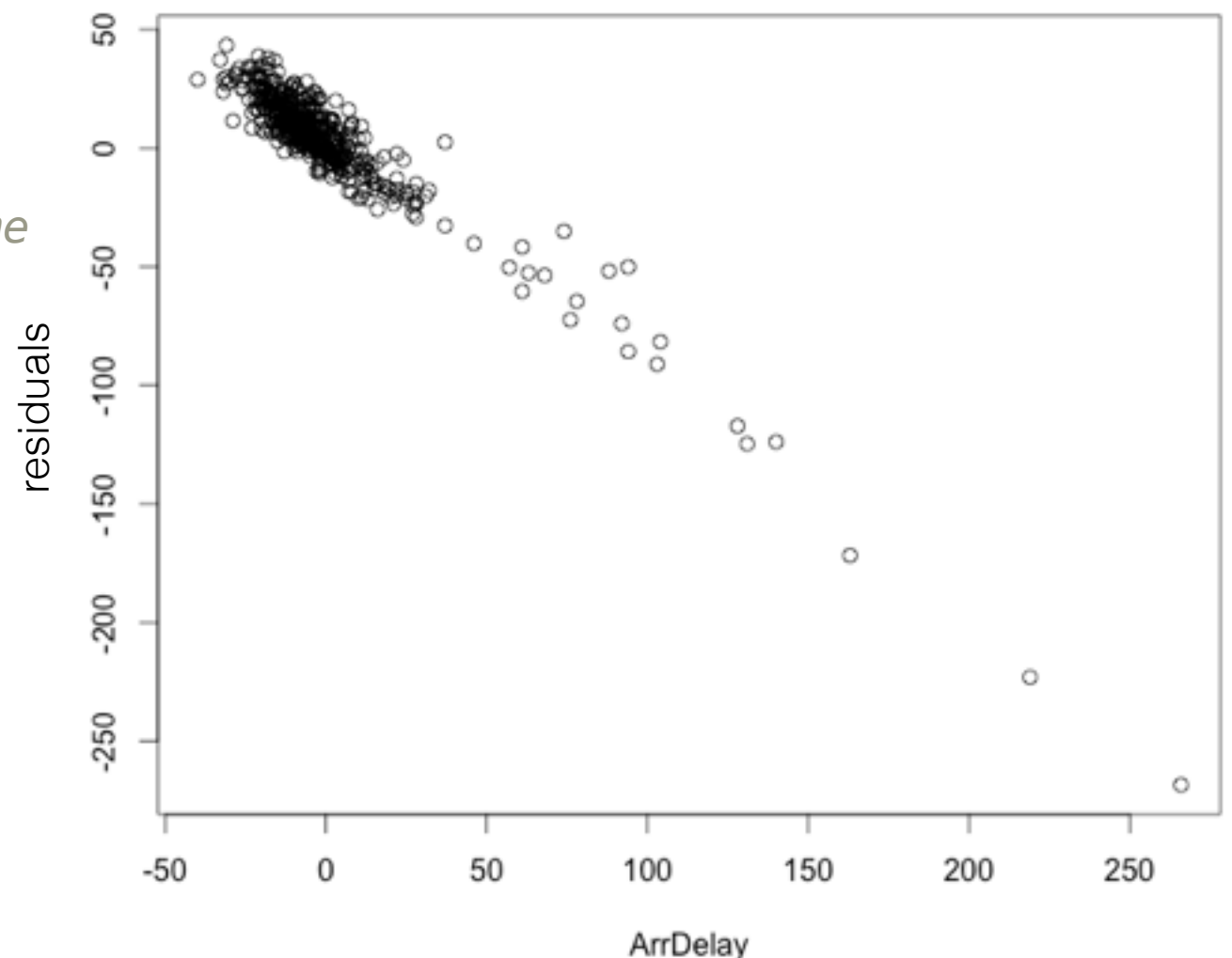
```
# Make sure that number of rows is same
nrow(actDelay) == nrow(predDelay)
```

```
# Compute residuals
residuals = predDelay - actDelay
```

```
# Plot residuals
compare = cbind(
  as.data.frame(actDelay$ArrDelay),
  as.data.frame(residuals$predict))
```

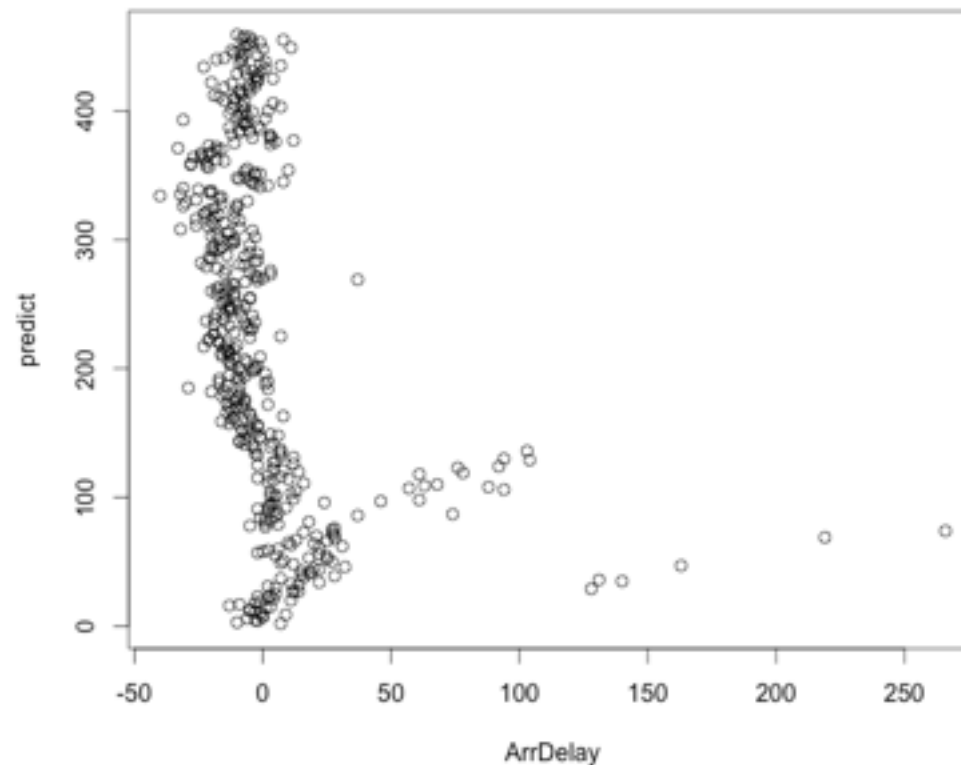
```
plot( compare[,1:2] )
```

References
of data



Warning!

If you are running R v3.1.0 you will see different plot:



Why? Float number handling was changed in that version. Our recommendation is to upgrade your R to the newest version.

Try GBM Algo

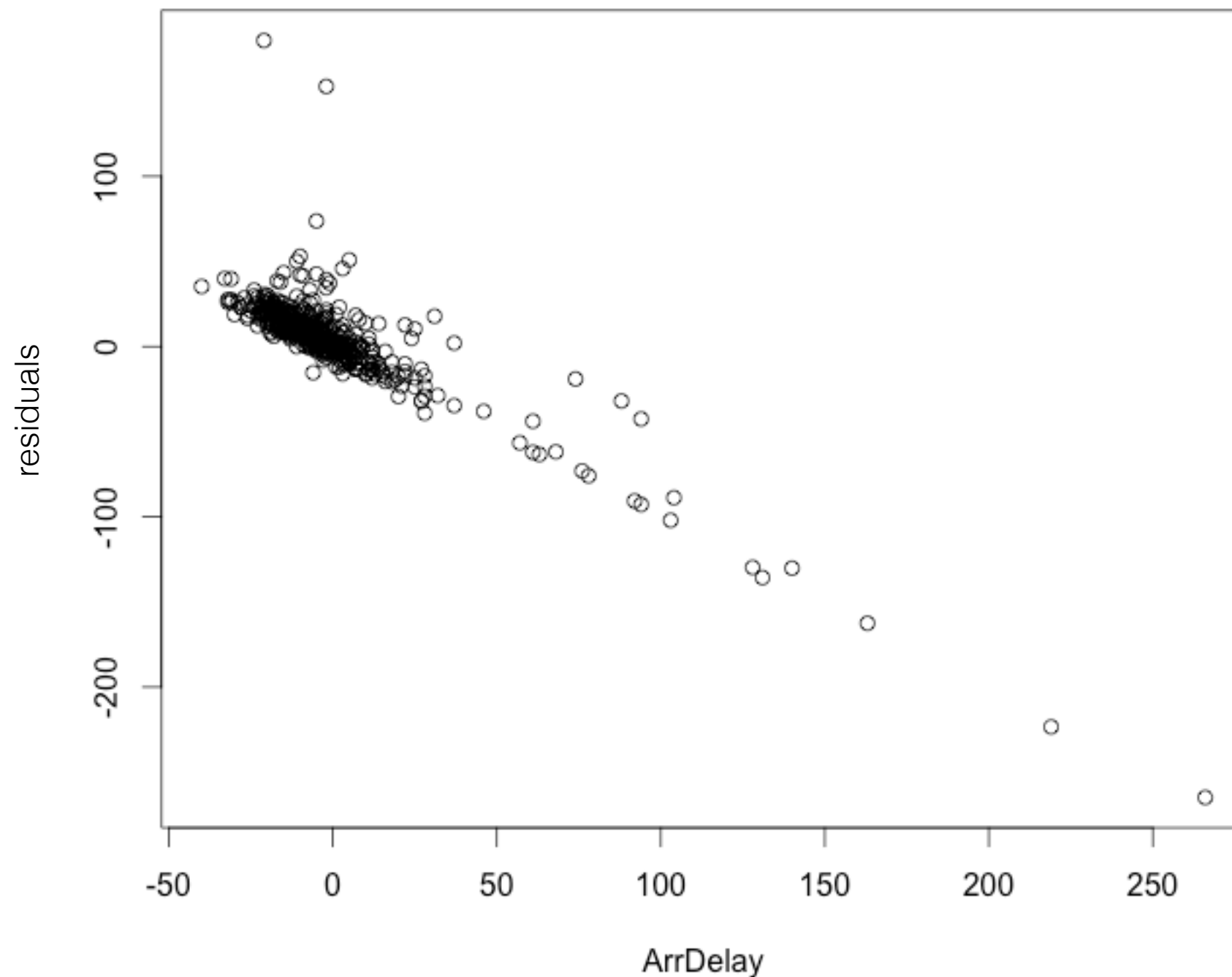
```
import hex.tree.gbm.GBM
import hex.tree.gbm.GBMModel.GBMParameters

val gbmParams = new GBMParameters()
gbmParams._train = trainTable
gbmParams._response_column = 'ArrDelay'
gbmParams._valid = validTable
gbmParams._ntrees = 100

val gbm = new GBM(gbmParams)
val gbmModel = gbm.trainModel.get

// Print R code for residual plot
val gbmPredictTable = gbmModel.score(testTable)('predict')
printf( residualPlotRCode(gbmPredictTable, 'predict', testTable,
'ArrDelay') )
```

Residuals plot for GBM prediction



Hands-On #3

**How Can I Develop and
Run Standalone App?**

Requirements

Idea or Eclipse

Git

Use Sparkling Water Droplet

Clone H2O Droplets repository

```
git clone https://github.com/h2oai/h2o-droplets.git  
cd h2o-droplets/sparkling-water-droplet/
```

Generate IDE project

For Idea

`./gradlew idea`

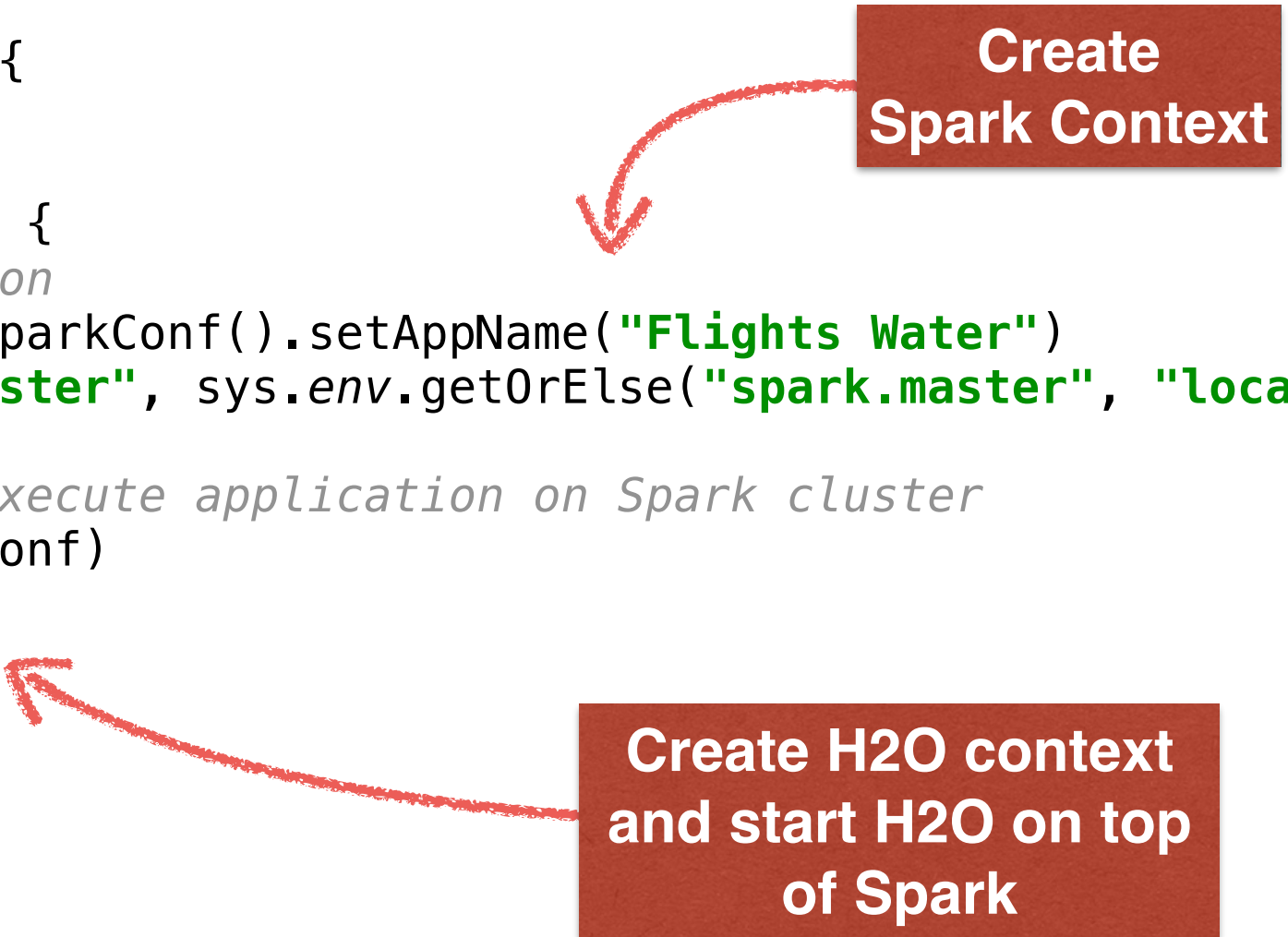
For Eclipse

`./gradlew eclipse`

... add import project into your IDE

Create An Application

```
object AirlinesWeatherAnalysis {  
  
  /** Entry point */  
  def main(args: Array[String]) {  
    // Configure this application  
    val conf: SparkConf = new SparkConf().setAppName("Flights Water")  
    conf.setIfMissing("spark.master", sys.env.getOrElse("spark.master", "local"))  
  
    // Create SparkContext to execute application on Spark cluster  
    val sc = new SparkContext(conf)  
    // Start H2O cluster only  
    new H2OContext(sc).start()  
  
    // User code  
    // . . .  
  }  
}
```



Create Spark Context

Create H2O context and start H2O on top of Spark

Build the Application

Build and test



`./gradlew build shadowJar`



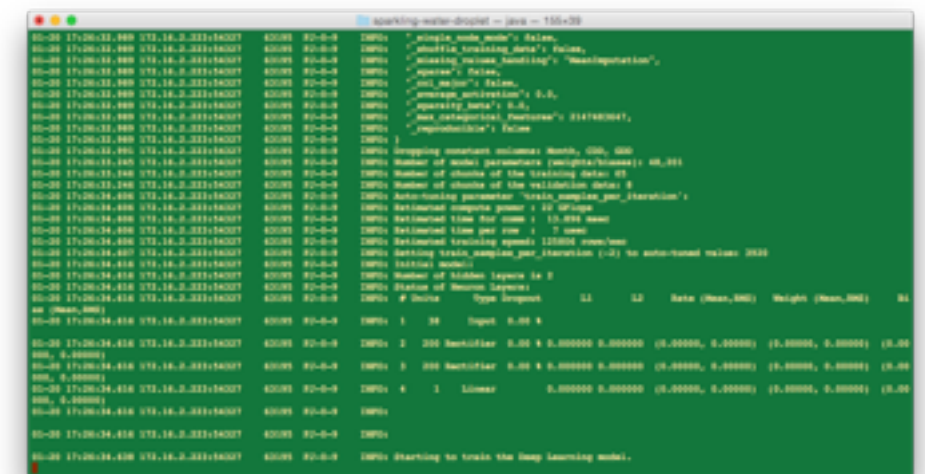
Create an assembly
which can be submitted
to Spark cluster

Run code on Spark

```
#!/usr/bin/env bash
```

```
APP_CLASS=water.droplets.AirlineWeatherAnalysis
FAT_JAR_FILE="build/libs/sparkling-water-droplet-app.jar"
MASTER=${MASTER:-"local-cluster[3,2,1024]"}
DRIVER_MEMORY=2g
```

```
$SPARK_HOME/bin/spark-submit "$@" \
  --driver-memory $DRIVER_MEMORY \
  --master $MASTER \
  --class "$APP_CLASS" $FAT_JAR_FILE
```



It is Open Source!

You can participate in

H2O Scala API

Sparkling Water testing

Mesos, Yarn, workflows ([PUBDEV-23,26,27,31-33](#))

Spark integration

MLLib Pipelines

Check out our JIRA
at <http://jira.h2o.ai>

Come to Meetup

<http://www.meetup.com/Silicon-Valley-Big-Data-Science/>



More info

Checkout **H2O.ai** Training Books

<http://learn.h2o.ai/>

Checkout **H2O.ai** Blog for Sparkling Water tutorials

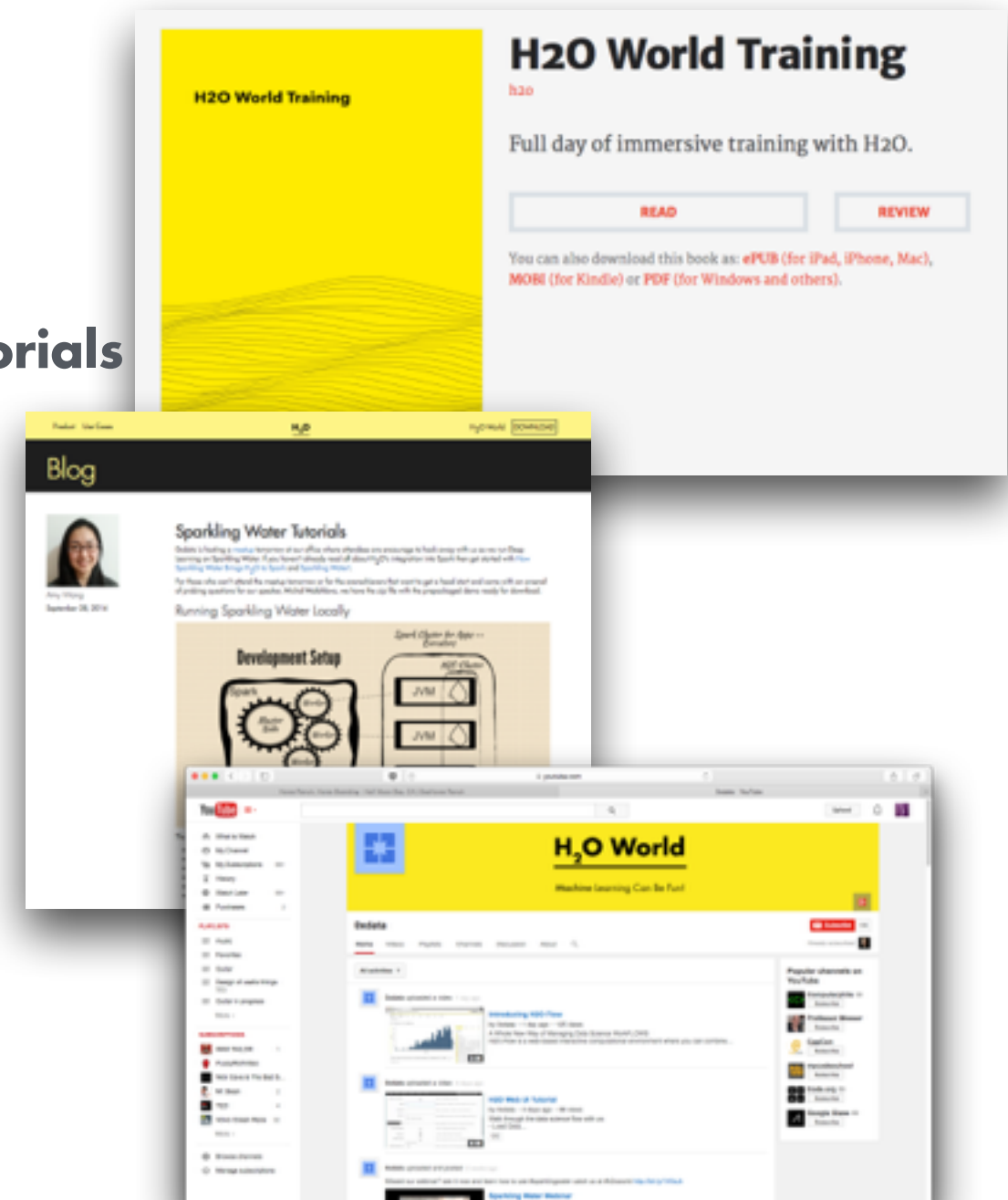
<http://h2o.ai/blog/>

Checkout **H2O.ai** Youtube Channel

<https://www.youtube.com/user/0xdata>

Checkout GitHub

<https://github.com/h2oai/sparkling-water>



Thank you!

Learn more about H₂O at
h2o.ai
or

```
> for r in sparkling-water; do  
git clone "git@github.com:h2oai/\$r.git"  
done
```

Follow us at @h2oai

And the winner is

