On Building Predictive Models With H2O

H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Administration
5. Sparkling Water
6. Productionalizing Models

Company Overview

Company

- Team: 80. Founded in 2012, Mountain View, CA
- Stanford Math & Systems Engineers

Product

- Open Source Leader in Machine & Deep learning
- Ease of Use and Smarter Applications
- R, Python, Spark & Hadoop Interfaces
- Expanding Predictions to Mass Analyst markets


Executive Team

Sri Satish Ambati

CEO & Co-founder

DataStax

Tom Kraljevic

VP of Engineering

Abrizio, Intel

Arno Candel

Chief Architect

HPC, CERN

Board of Directors

Jishnu Bhattacharjee // Nexus Ventures
Ash Bhardwaj // Flextronics

Scientific Advisory Council

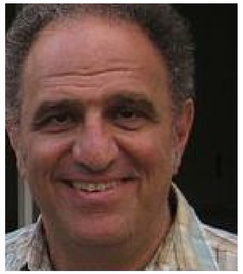
Trevor Hastie
Stephen Boyd
Rob Tibshirani

H2O.ai
Machine Intelligence

cientific Advisory Council

Dr. Trevor Hastie

- PhD in Statistics, Stanford University
- John A. Overdeck Professor of Mathematics, Stanford University
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Co-author with John Chambers, *Statistical Models in S*
- Co-author, *Generalized Additive Models*
- 108,404 citations (via Google Scholar)



Dr. Rob Tibshirani

- PhD in Statistics, Stanford University
- Professor of Statistics and Health Research and Policy, Stanford University
- COPPS Presidents' Award recipient
- Co-author, *The Elements of Statistical Learning: Prediction, Inference and Data Mining*
- Author, *Regression Shrinkage and Selection via the Lasso*
- Co-author, *An Introduction to the Bootstrap*

Dr. Stephen Boyd

- PhD in Electrical Engineering and Computer Science, UC Berkeley
- Professor of Electrical Engineering and Computer Science, Stanford University
- Co-author, Convex Optimization
- Co-author, Linear Matrix Inequalities in System and Control Theory
- Co-author, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*

What is H2O?

Math Platform

Open source in-memory prediction engine

- Parallelized and distributed algorithms making the most use out of multithreaded systems
- GLM, Random Forest, GBM, PCA, etc.

API

Easy to use and adopt

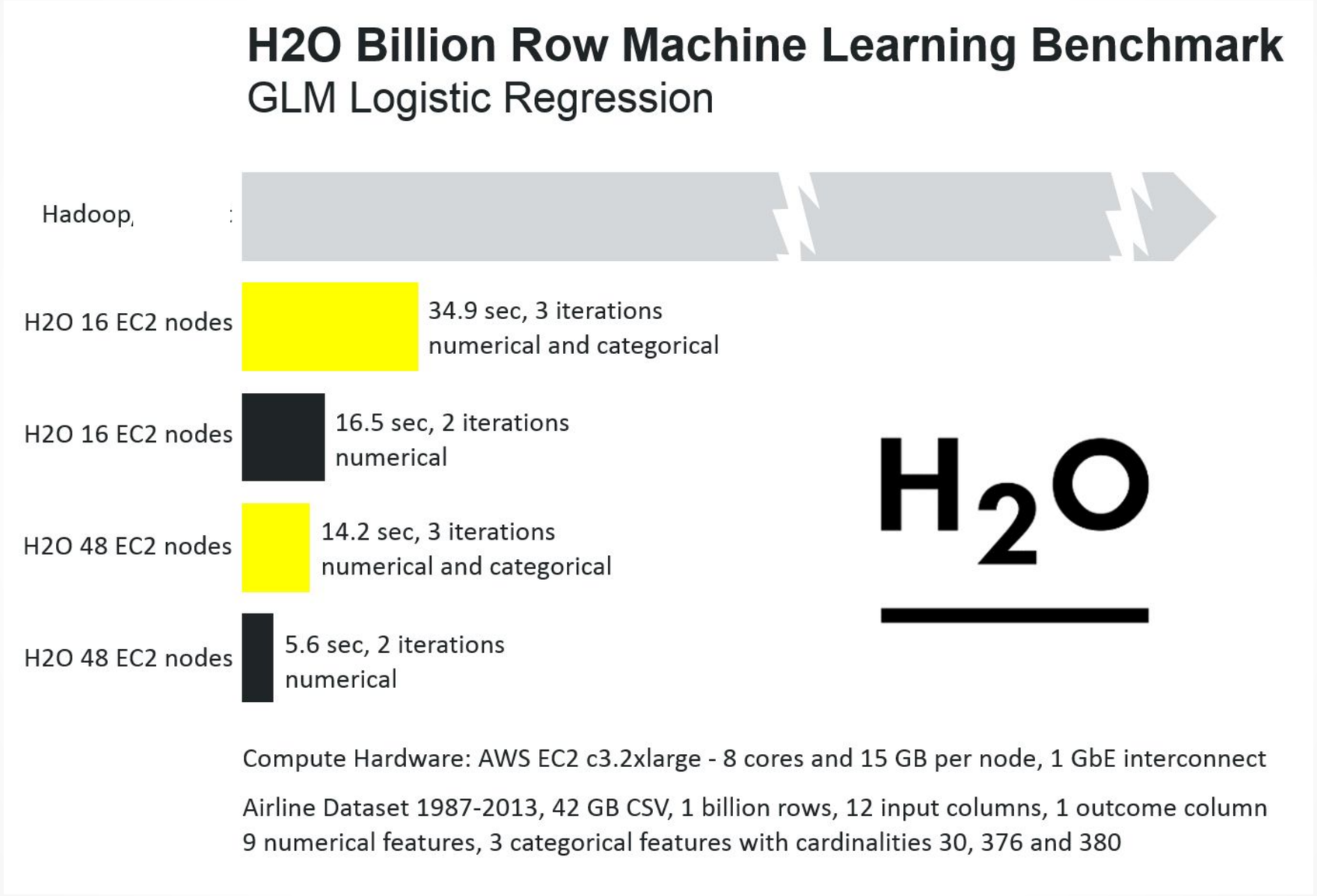
- Written in Java – perfect for Java Programmers
- REST API (JSON) – drives H2O from R, Python, Excel, Tableau

Big Data

More data? Or better models? BOTH

- Use all of your data – model without down sampling
- Run a simple GLM or a more complex GBM to find the best fit for the data
- More Data + Better Models = Better Predictions

Performance



Algorithms on H_2O

Supervised Learning

Statistical Analysis

- **Generalized Linear Models**: Binomial, Gaussian, Gamma, Poisson and Tweedie
- **Naïve Bayes**

Ensembles

- **Distributed Random Forest**: Classification or regression models
- **Gradient Boosting Machine**: Produces an ensemble of decision trees with increasing refined approximations

Deep Neural Networks

- **Deep learning**: Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

Algorithms on H_2O

Unsupervised Learning

Clustering

- **K-means**: Partitions observations into k clusters/groups of the same spatial size

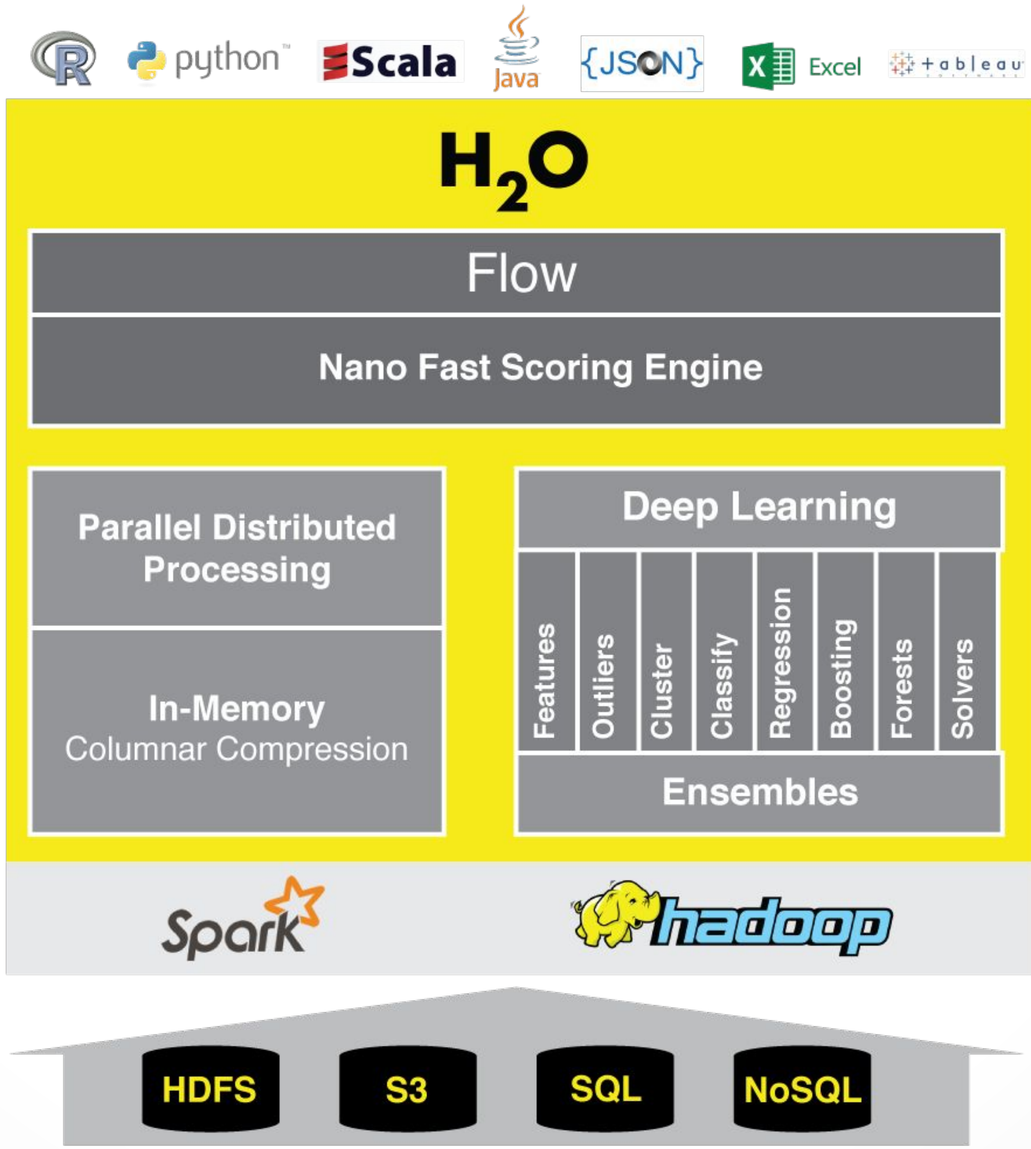
Dimensionality Reduction

- **Principal Component Analysis**: Linearly transforms correlated variables to independent components

Anomaly Detection

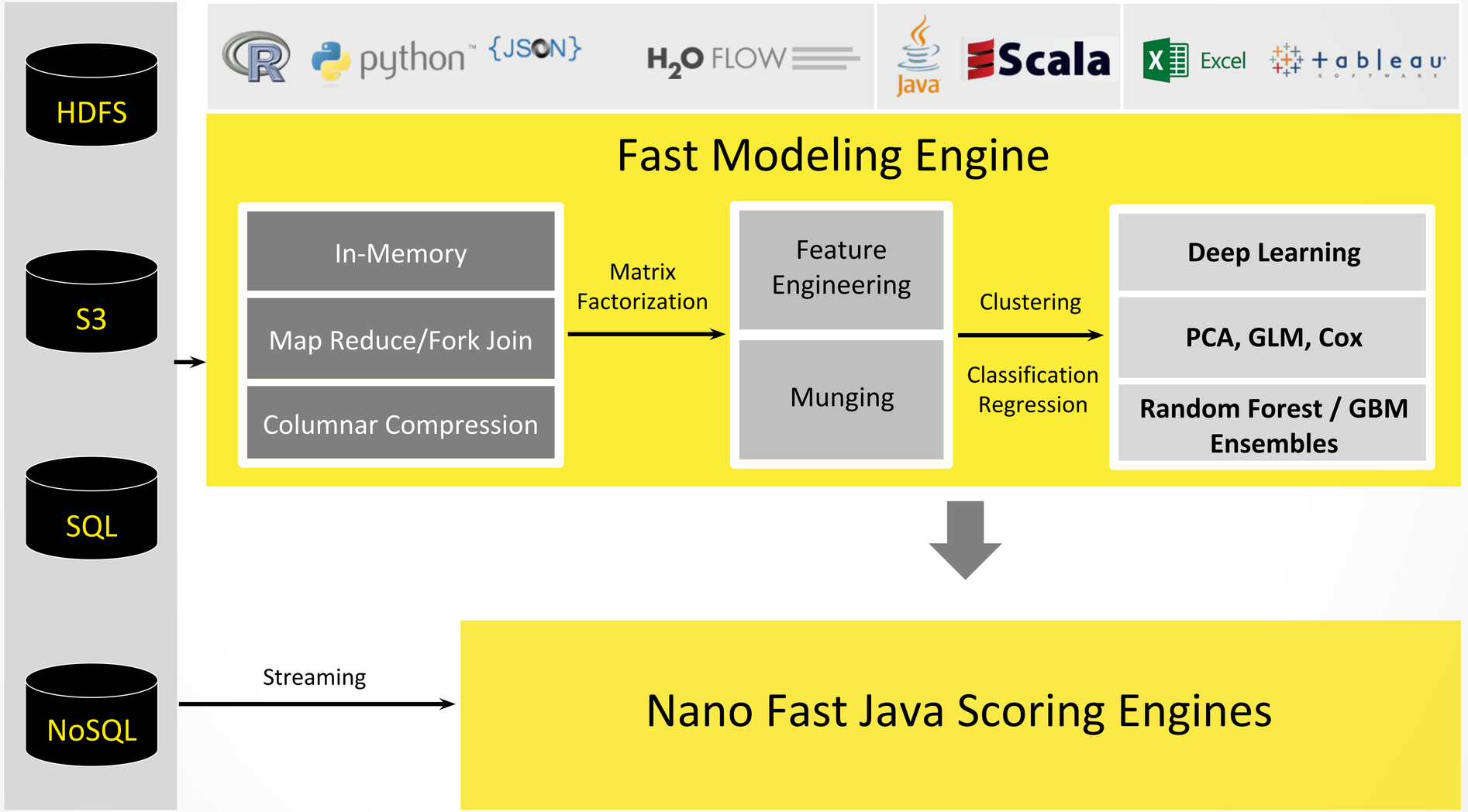
- **Autoencoders**: Find outliers using a nonlinear dimensionality reduction using deep learning

Accuracy with Speed and Scale



H2O.ai
Machine Intelligence

Accuracy with Speed and Scale



H2O.ai
Machine Intelligence

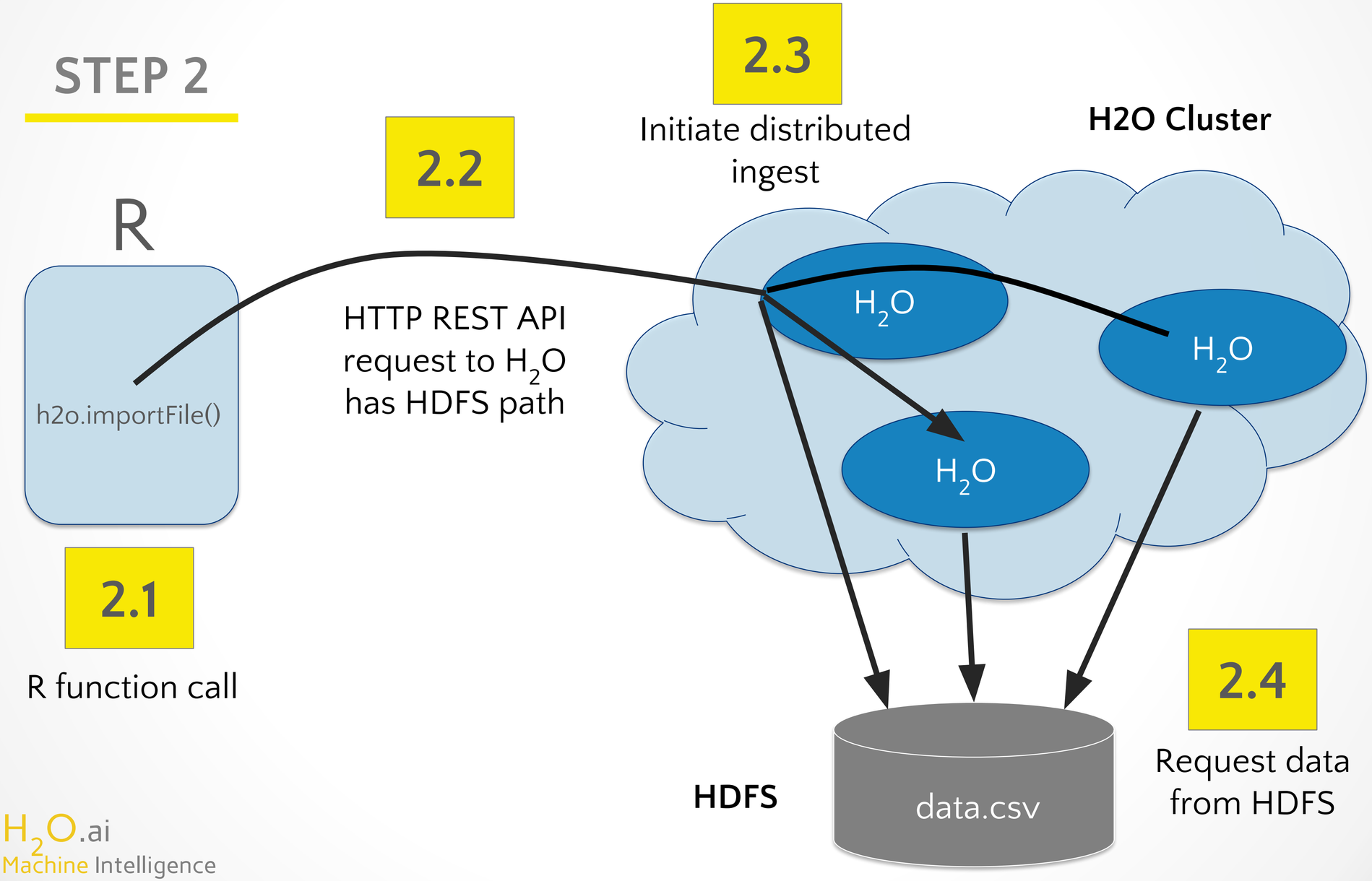
Reading Data into H2O with R

STEP 1

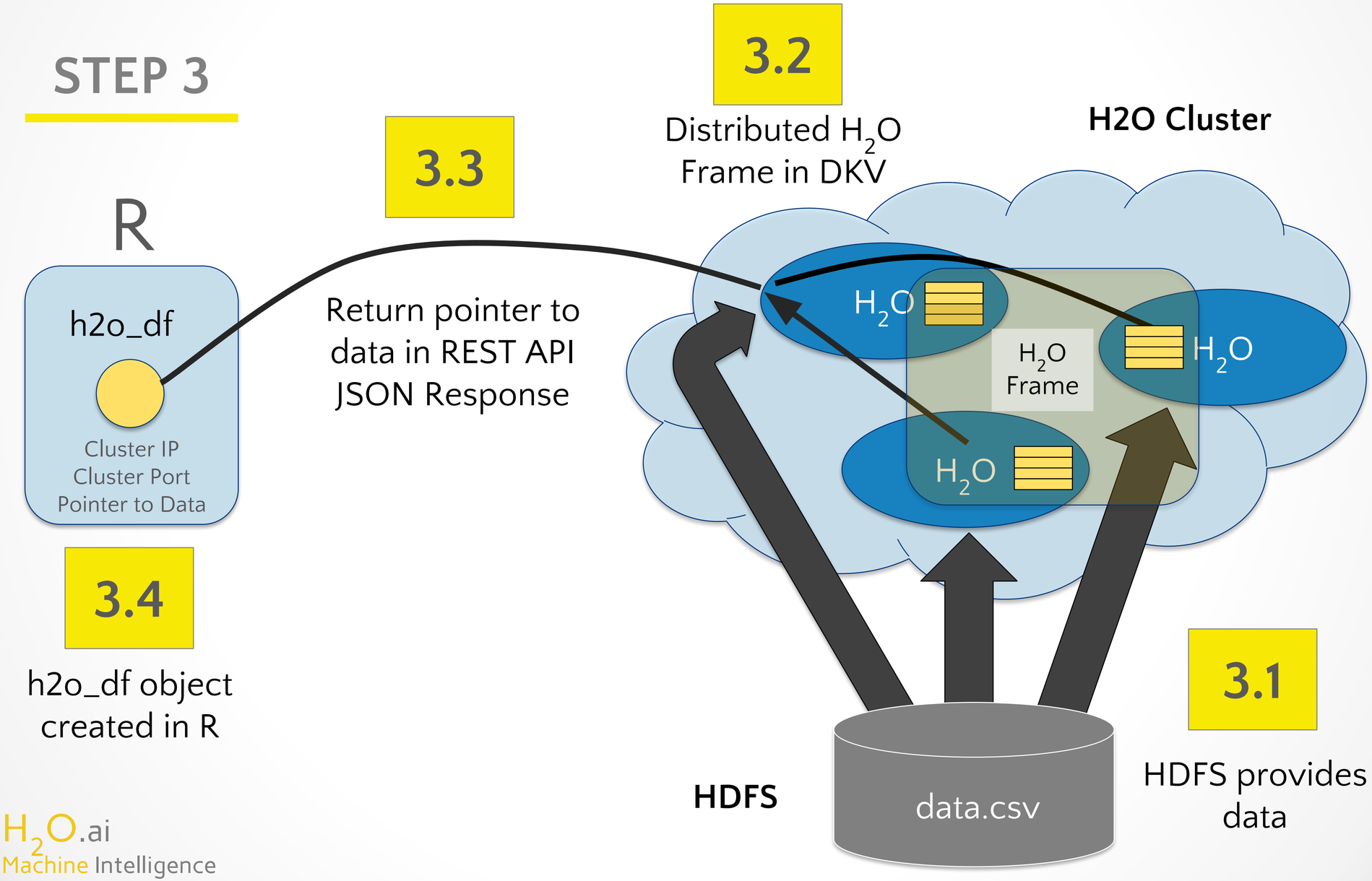
h2o_df = h2o.importFile("../data/allyears2k.csv")

R user

Reading Data from HDFS into H2O with R



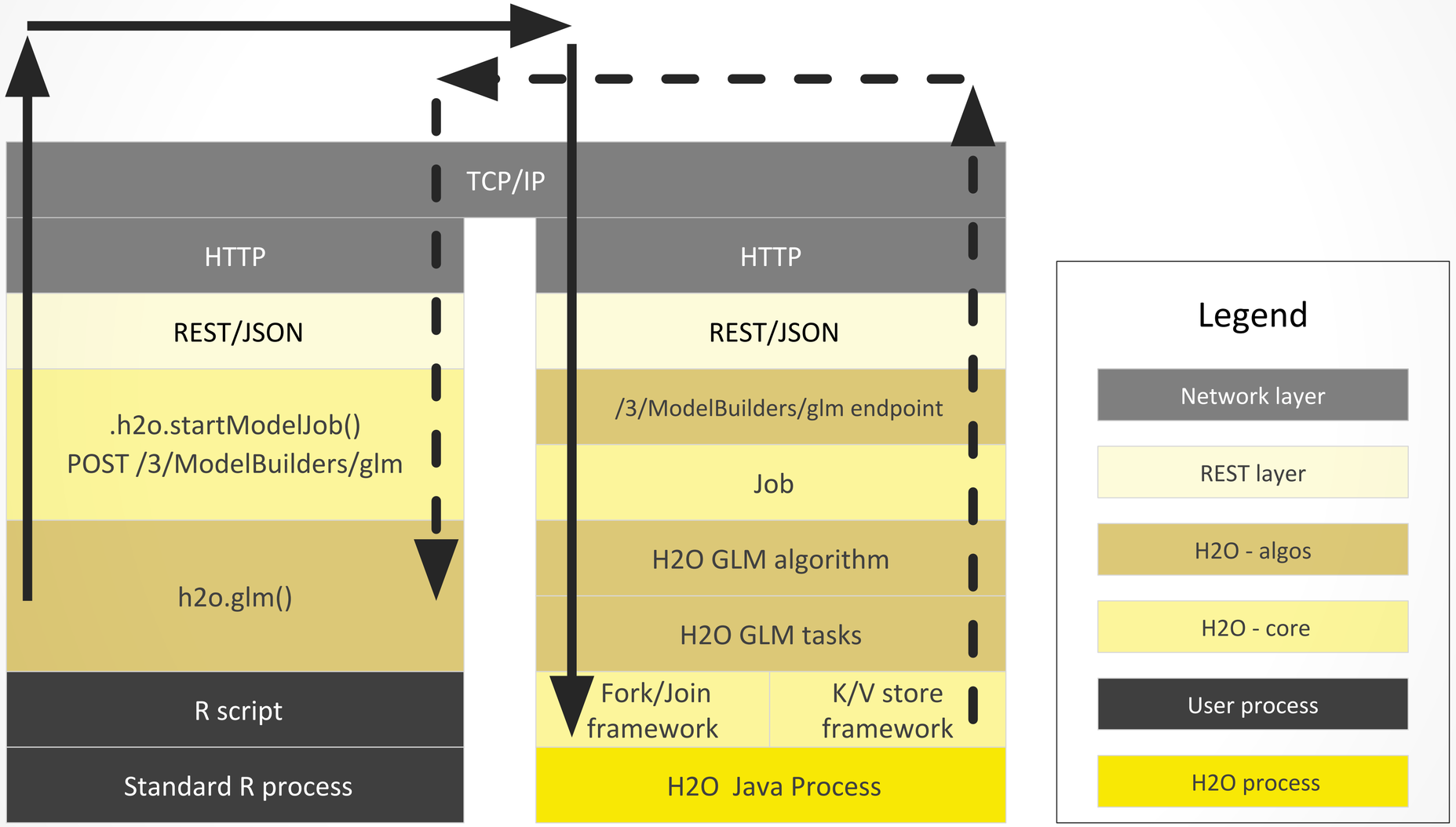
Reading Data from HDFS into H2O with R



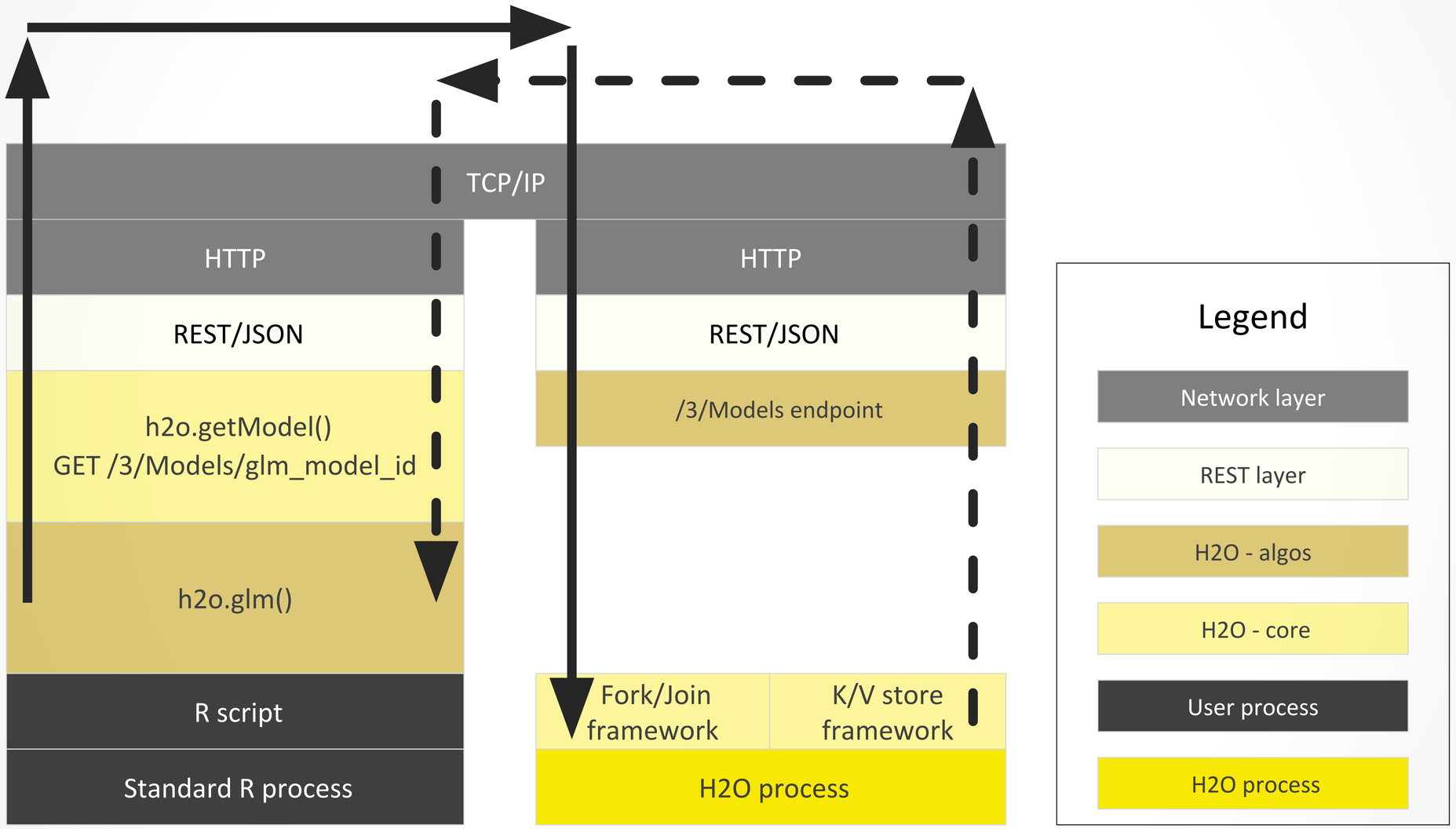
Data Munging in R

	Standard R	R on H2O
Reading in data	read.csv, read.table, etc	h2o.importFile
Summarizing data	summary	summary, h2o.summary
Combining rows or columns	cbind, rbind	h2o.cbind, h2o.rbind
Unary or Binary Operations	+, -, *, /, ^, %%, %/%, etc	+, -, *, /, ^, %%, %/%, etc
Building GLM model	glm, glmnet	h2o.glm
Predict with Model	predict	h2o.predict
Obtaining Metrics	auc, mse, logLoss, etc	h2o.auc, h2o.mse, h2o.logloss
Unsupported Functions	smooth	UNSUPPORTED

R Script Starting H2O GLM



R Script Retrieving H2O GLM Result



H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Administration
5. Sparkling Water
6. Productionalizing Models

`as.h2o()` vs. `as.data.frame()`

`as.h2o()`

- Turns R data.frame into an H2OFrame
- Saves data.frame to CSV and then performs `h2o.upload()` to H2O cluster

`as.data.frame()`

- Turns H2OFrame to R data.frame
- Does `h2o.downloadCSV` and then `read.csv`, and finally turns it into data.frame

Note: Be mindful of `as.data.frame()`, it is difficult to make a large H2OFrame into a R data.frame

Column Types: Enum, Numeric, Dates

Enum vs. Numeric vs. Dates

Enum	**Enum:** Also known as enumerated variable. Represents a categorical variable
Numeric	**Numeric:** Variable that holds a numeric value
Dates	**Date:** %M%D%Y is automatic format that H2O reads in

Enum vs. Numeric vs. Dates

Enum

`as.factor(x)`

Used to convert given variable to an enumerated type variable

Numeric

`as.numeric(x)`

Used to convert given variable to an numeric variable

Dates

`h2o.year(x)`
Convert the entries of an H2OFrame object from milliseconds to years, indexed starting from 1900.

`h2o.month(x)`
Converts the entries of an H2OFrame object from milliseconds to months (on a 1 to 12 scale).

Filters & Logics

Filters & Logics

```
h2o.ifelse(test, yes, no)
```

Arguments

`test`	A logical description of the condition to be met (>, <, =, etc…)
`yes`	The value to return if the condition is TRUE.
`no`	The value to return if the condition is FALSE.

Note: Only numeric values can be tested, and only numeric results can be returned.

Summary & Aggregation

Summary & Aggregation

```
h2o.group_by(data, by, ..., gb.control =
list(na.methods = NULL, col.names = NULL))
```

Arguments

`data`	an H2OFrame object.
`by`	a list of column names
`gb.control`	a list of how to handle NA values in the dataset as well as how to name output columns
`. . .`	Any supported aggregated function: `mean, min, max, sum, sd, nrow`

Summary & Aggregation

```
h2o.table(x, y = NULL, dense = TRUE)
```

Arguments

x	An H2OFrame object with at most two columns
y	An H2OFrame similar to x, or NULL
dense	A logical for dense representation, which lists only non-zero counts, 1 combination per row. Set to FALSE to expand counts across all combinations.

Value: Returns a tabulated H2OFrame Object

String Munging

String Munging

```
h2o.impute(data, column = 0, method = c(“mean", "median",
“mode"), combine_method c("interpolate","average", "lo", "hi"),
by = NULL, groupByFrame = NULL, values = NULL)
```

Arguments

data	The dataset containing the column to impute.
column	The column to impute.
method	“mean" replaces NAs with the column mean; "median" replaces NAs with the column median; "mode" replaces with the most common factor (for factor columns only)
combine_method	If method is "median", then choose how to combine quantiles on even sample sizes. This parameter is ignored in all other cases
by	Group by columns
groupByFrame	Impute the column col with this pre-computed grouped frame.
values	A vector of impute values (one per column). NaN indicates to skip the column

String Munging

- Substitute pattern match w/replacement
  - `h2o.gsub()`
  - `h2o.sub()`

- String Cleaning
  - `h2o.substring()`
  - `h2o.strsplit()`
  - `h2o.trim()`
  - `h2o.rstrip() & h2o.lstrip()`
  - `h2o.interaction()`

Demo Time

H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Administration
5. Sparkling Water
6. Productionalizing Models

1. Train vs Test

Training Set vs. Test Set

- Partition the original data (randomly or stratified) into a **training** set and a **test** set. (e.g. 70/30)

Training Error vs. Test Error

- It can be useful to evaluate the training error, but you should not look at training error alone.
- Training error is not an estimate of **generalization error** (on a test set or cross-validated), which is what you should care more about.
- Training error vs test error over time is an useful thing to calculate. It can tell you when you start to overfit your model, so it is a useful metric in supervised machine learning.

2. Train vs Test vs Valid

Training Set vs. Validation Set vs. Test Set

- If you have “enough” data and plan to do some model tuning, you should really partition your data into three parts — Training, Validation and Test sets.
- There is **no general rule** for how you should partition the data and it will depend on how strong the signal in your data is, but an example could be: 50% Train, 25% Validation and 25% Test

Validation is for Model Tuning

- The validation set is used **strictly for model tuning** (via validation of models with different parameters) and the test set is used to make a final estimate of the generalization error.

3. Model Performance

Test Error

- Partition the original data (randomly) into a training set and a test set. (e.g. 70/30)
- Train a model using the training set and evaluate performance (a single time) on the test set.

K-fold Cross-validation

- Train & test K models as shown.
- Average the model performance over the K test sets.
- Report cross-validated metrics.

Performance Metrics

- Regression: R^2, MSE, RMSE
- Classification: Accuracy, F1, H-measure, Log-loss
- Ranking (Binary Outcome): AUC, Partial AUC

4. Class Imbalance

Imbalanced Response Variable

- A dataset is said to be **imbalanced** when the binomial or multinomial response variable has one or more classes that are underrepresented in the training data, with respect to the other classes.

Very common

- This is incredibly common in real-word datasets.
- In practice, balanced datasets are the rarity, unless they have been artificially created.
- There is **no precise definition** of what defines an imbalanced vs balanced dataset — the term is vague.
- My rule of thumb for binary response: If the minority class makes <10% of the data, this can cause issues.

Industries

- Advertising — Probability that someone clicks on ad is very low… very very low.
- Healthcare & Medicine — Certain diseases or adverse medical conditions are rare.
- Fraud Detection — Insurance or credit fraud is rare.

4. Remedies

Artificial Balance

- You can **balance** the training set using sampling.

Potential Pitfalls

- Notice that we don't say to balance the test set. The test set represents the true data distribution. The only way to get "honest" model performance on your test set is to use the original, unbalanced, test set.
- The same goes for the hold-out sets in cross-validation. For this, you may end up having to write custom code, depending on what software you use.

Solutions

- H2O has a "balance_classes" argument that can be used to do this properly & automatically.
- You can manually **upsample (or downsample)** your minority (or majority) class(es) set either by duplicating (or sub-sampling) rows, or by using row weights.

5. Categorical Data

Real Data

- Most real world datasets contain categorical data.

Too Many Categories

- Problems can arise if you have **too many categories**.
- A lot of ML software will place limits on the number of categories allowed in a single column (e.g. 1024) so you may be forced to deal with this whether you like it or not.
- When there are high-cardinality categorical columns, often there will be many categories that only occur a small number of times (not very useful).

Solutions

- If you have some hierarchical knowledge about the data, then you may be able to reduce the number of categories by using some sensible higher-level mapping of the categories.
- Example: ICD-9 codes — thousands of unique diagnostic and procedure codes. You can map each category to a higher level super-category to reduce the cardinality.

6. Missing Data

Types of Missing Data

- Unavailable: Valid for the observation, but not available in the data set.
- Removed: Observation quality threshold may have not been reached, and data removed
- Not applicable: measurement does not apply to the particular observation (e.g. number of tires on a boat observation)

What to Do

- It depends! Some options:
- Ignore entire observation.
- Create an binary variable for each predictor to indicate whether the data was missing or not
- Segment model based on data availability.
- Use alternative algorithm: decision trees accept missing values; linear models typically do not.

7. Outliers/Extreme Values

Types of Outliers

- Outliers can exist in response or predictors
- Valid outliers: rare, extreme events
- Invalid outliers: erroneous measurements

What Can Happen

- Outlier values can have a disproportionate weight on the model.
- MSE will focus on handling outlier observations more to reduce squared error.
- Boosting will spend considerable modeling effort fitting these observations.

What to Do

- Remove observations.
- Apply a transformation to reduce impact: e.g. log or bins.
- Choose a loss function that is more robust: e.g. MAE vs MSE.
- Impose a constraint on data range (cap values).
- Ask questions: Understand whether the values are valid or invalid, to make the most appropriate choice.

8. Data Leakage

What Is It

- Leakage is allowing your model to use information that will not be available in a production setting.
- Obvious example: using the Dow Jones daily gain/loss as part of a model to predict individual stock performance

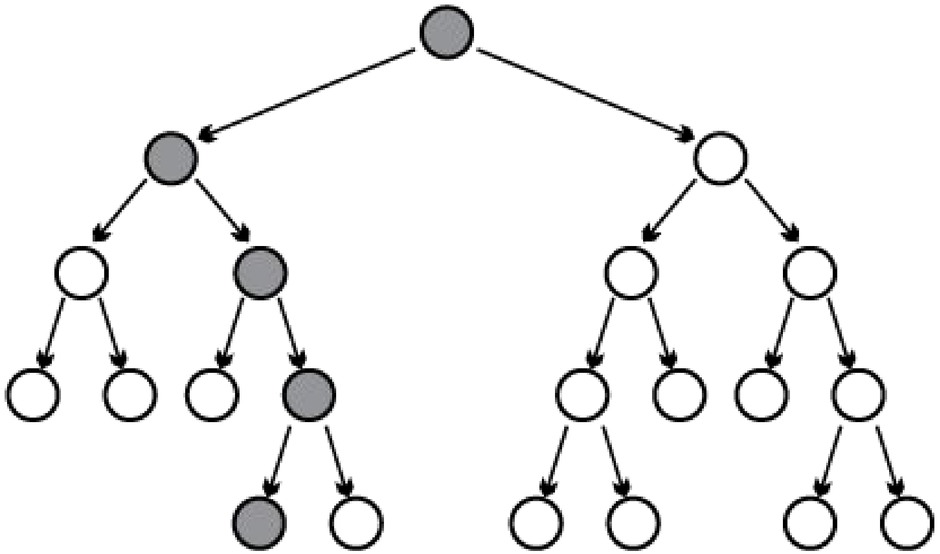
What Happens

- Model is overfit.
- Will make predictions inconsistent with those you scored when fitting the model (even with a validation set).
- Insights derived from the model will be incorrect.

What to Do

- Understand the nature of your problem and data.
- Scrutinize model feedback, such as relative influence or linear coefficient.

Algorithm Spotlight: Gradient Boosting Machine



When to use GBM?

What is GBM?

An ensemble of weak learners. The decision trees in GBM are built consecutively, with each new layer solving for the net loss of prior trees. This means the algorithm is inherits the pros of building decision trees while also making up for poor accuracy by creating an ensemble of trees that learn from prior trees.

Strengths

- Nonlinear models
- Robust to correlated features
- Robust to feature distributions
- Robust to missing values

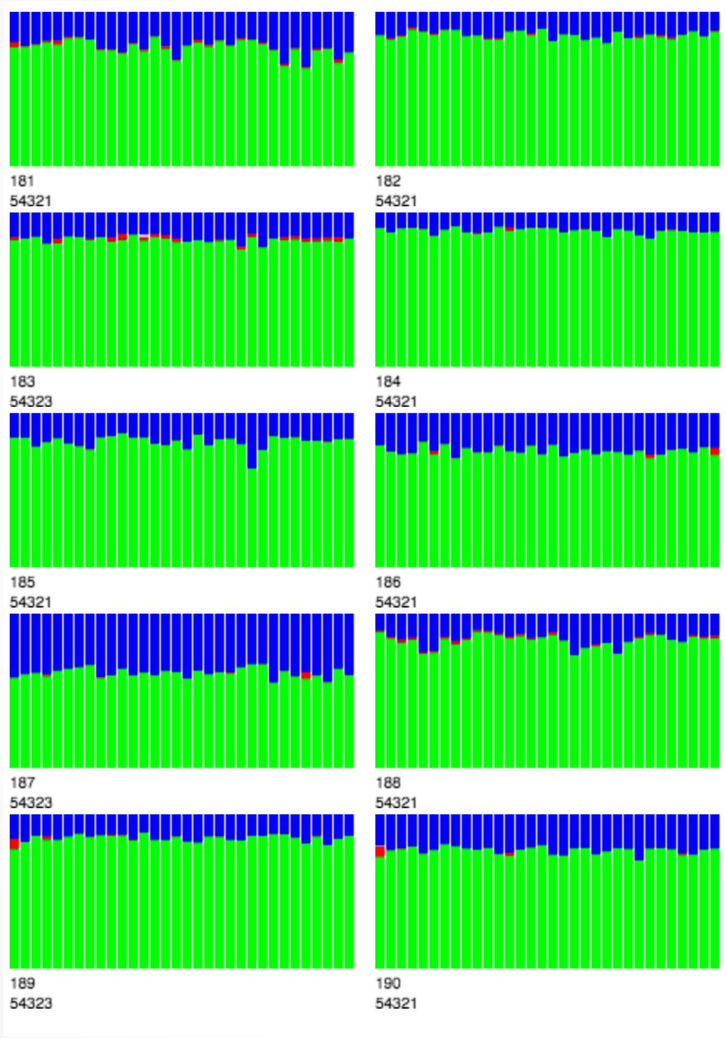
Weaknesses

- Tends to overfit on the training set
- Sensitive to noise and outliers

GBM Functionalities in H2O

- **Regression and Classification Models** Exponential distributions including Poisson, Gamma, and Tweedie are available in addition to Bernoulli, multinomial, and gaussian distributions.
- **Fast and CPU Efficient** Parallel and distributed computation across multiple nodes and many cores.
- **Grid Search** Hyperparameter optimization allows the user to run through many parameters before selecting the best models.
- **Early Stopping** The user can specify the metric and the incremental change in the metric as convergence. If additional trees no longer provide an increase in AUC prune the tree early.
- **Stochastic** User can specify the sample rate that the algorithm will sample the column and row by for better generalization.
- **Model Output** The model is exportable as Java code and if you find the model overfitted after a certain number of trees, it is easy to reduce the number of trees in a POJO before putting it in production without rerunning model build.

Implementation in H2O



1. Map vectors to leaves of a tree by assigning a Node ID to each row in the dataset.
2. We make a split point decision that will move rows of the data down two internal leave nodes.
3. On each pass through the data, each observation starts at an internal tree node. As we move the observation down the decision tree into a new leaf we record the new Node ID for next pass.
4. We then collect the summaries from all the split decisions and pick the feature and the split point with the least variance.
5. By repeating this process we move all the data down the decision trees assigning it new Node IDs along the way and collecting histograms for all the rows. At the end we can split each current leaf and finish building a layer and we repeat for each layer. After one tree is built we start on the next tree.
6. We run completely parallel and distributed within a single tree level with an upper bound on the tree building rate which is capped at the latency to build a single level as described above.

Parameter Explanation

Binning

- Traditionally split points are chosen by sorting the each feature and inspecting an induced split.
- For big data even when running parallel and distributed this can be computationally expensive so we approximate sorting with binning.
- **More Bins, More Accurate** The number of bins can be specified by the user and it is the minimum number of bins required in a histogram built for each feature.

Binning Categorical Features

- **Too Many Levels** We run the issue of slow model builds by inducing splits by each level so again we choose to bin the levels in a categorical column.
- To try each and every level in a column you will need to specify bins_cat to be the cardinality of the column.
- **More Bins, More Likely To Overfits** Increasing the number of bins can lead to perfect splits by a single level, this reduces the randomness of grouped factors. The model can split more finely then it should on the training set and start to overfit on the validation set.

Generalized Linear Models

- Well known statistical/machine learning method
- Fits a linear model
  - link(y) = c1*x1 + c2*x + … + cn*xn + intercept
  - easy to fit
  - easy to understand and interpret
  - well understood statistical properties
- Regression problems
  - gaussian, poisson, gamma, tweedie
- Classification
  - binomial, multinomial
- Requires good features
  - generally not as powerful “out of the box” as other models (GBM, Deep Learning)

Penalized Generalized Linear Models

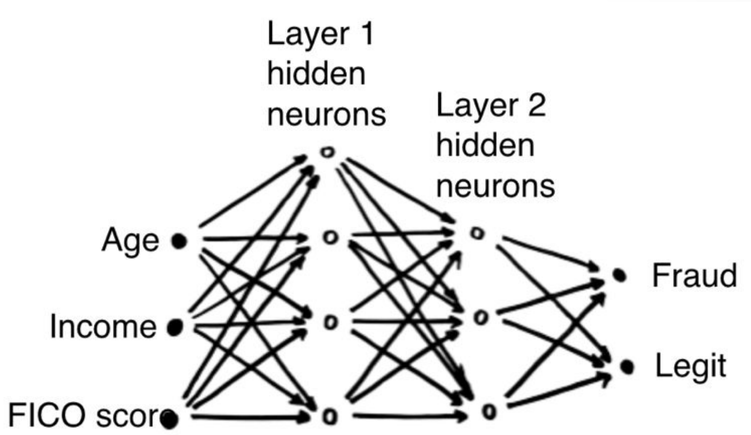
- Plain GLM is nice, but
  - can overfit (test performance way worse than training)
  - correlated variables - solution not unique, how to pick?
- Solution - Add Regularization (aka be more conservative)
  - add penalty to reduce size of the vector
  - L1 or L2 norm of the coefficient vector
- L1 versus L2
  - L1 sparse solution
    - picks one correlated variable, others discarded
  - L2 dense solution
    - correlated variables coefficients are pushed to the same value
- Elastic Net
  - combination of L1 and L2
  - sparse solution, correlated variables grouped, enter/ leave the model together

Summary - Best Practices

- Regularization selection
  - if not sure, try both dense and sparse (l1, no l1)
  - don't need to grid search alpha, only 2 or 3 values would do (e.g. 0.0, .5,.99)
  - always include some l2, i.e. set alpha to .99 instead of 1
- Wide datasets
  - datasets with ~ 10s of thousand of predictors or more
  - standard IRLSM solver does not work (can use L-BFGS)
  - IRLSM + lambda search works and is recommended!
    - (with alpha >> 0!)
    - can get solution up to low thousands of non-zeros
    - efficient way to filter out unused coefficients
  - L-BFGS + l2 penalty works
    - L-BFGS + l1 may work, but can take very long time
- Grid Search - available, not really needed
  - lambda search with 2 or 3 values of alpha is enough

What is Deep Learning?

- Deep Learning learns a **hierarchy of non-linear transformations**
- Neurons transform their input in a non-linear way
- **Black-box, brute-force method, really good at pattern recognition**
- Deep Learning got a boost in the last decade due to **faster hardware** and **algorithmic advances**



Deep Learning model
=
set of connecting weights
+
type of non-linearity

H2O

Deep Learning: Practical Use

strengths

- non linear
- robust to correlated features
- conceptually simple
- learned features can be extracted
- can stop training at any time
- can be fine-tuned with more data
- great ensemble member
- efficient for multi-class problems
- world-class at pattern recognition

weaknesses

- slow to train
- slow to score
- not interpretable
- results not fully reproducible
- theory not well understood
- overfits, needs regularization
- many hyper-parameters
- expands categorical variables
- must impute missing values

What is H2O Deep Learning?

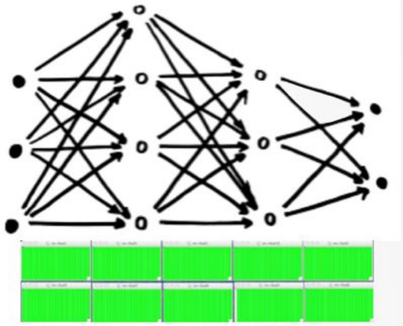
H2O Deep Learning:

Multi-layer fully-connected feed-forward Neural Network

+ **distributed** processing on multi-node clusters

+ **multi-threaded** speedup on multi-core CPUs

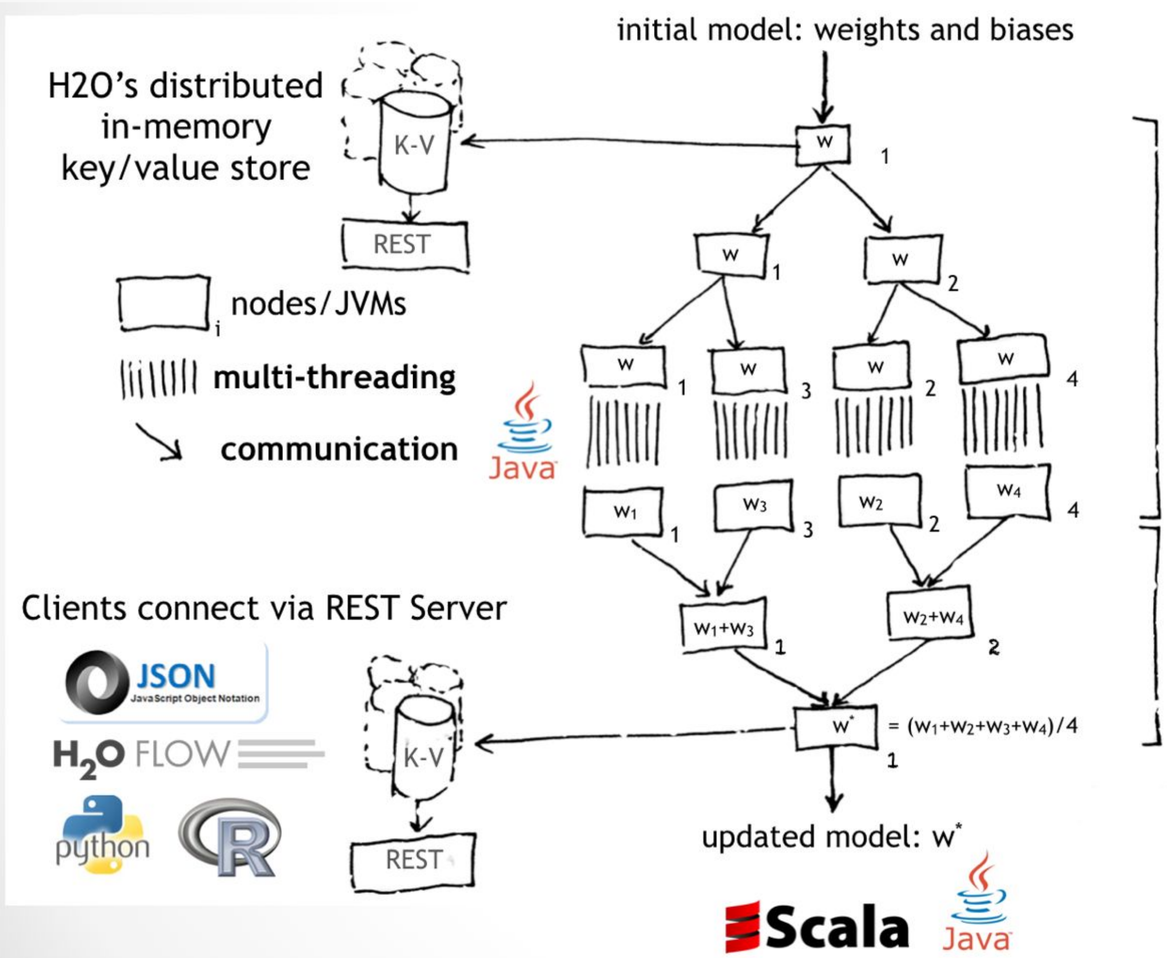
+ **fully featured** for **fast** & **accurate** results
  (automatic standardization, automatic handling of categorical and missing values, train/test data adaptation, model initialization, activation functions, multiple loss functions, autoencoder, load balancing, auto-tuning, adaptive learning rate, rate decay, momentum, L1/L2 penalty, dropout, hyper-parameter search, N-fold cross-validation, checkpointing, early stopping, variable importances, feature extraction, realtime model inspection, optimizations for sparse data and networks, etc.)



all 320 cores on 10 nodes maxed out

= **Easy-to-use scalable Deep Learning for large real-world datasets**
(insurance, healthcare, finance, fraud, churn, risk, IoT, etc.)

H2O Deep Learning Architecture



Map/Reduce Iterations

Map: training (fprop+bprop)
parallelized across all nodes and all CPU cores

Reduce: model aggregation
for global convergence

Shown here: Model averaging
New: Elastic averaging

Communication frequency is auto-tuned and user-controllable (affects convergence)

H2O Deep Learning on MNIST - World-record results

```
> library(h2o)
> h2oServer <- h2o.init(ip="mr-0xd1", port=53322)
> train_hex <- h2o.importFile(h2oServer, "mnist/train.csv.gz")
> test_hex  <- h2o.importFile(h2oServer, "mnist/test.csv.gz")
> train_hex[,785] <- as.factor(train_hex[,785])
> test_hex[,785] <- as.factor(test_hex[,785])
> dl_model <- h2o.deeplearning(x=c(1:784), y=785, training_frame=train_hex, l1=1e-5,
                               activation="RectifierWithDropout", input_dropout_ratio=0.2,
                               classification_stop=-1,hidden=c(1024,1024,2048), epochs=8000)
|=================================================================================================|
==================================|  100%
> h2o.confusionMatrix(dl_model, test_hex)
Confusion Matrix - (vertical: actual; across: predicted):
         0    1    2    3   4   5   6    7   8    9   Error          Rate
0      974    1    1    0   0   0   2    1   1    0 0.00612 =     6 / 980
1        0 1135    0    1   0   0   0    0   0    0 0.00088 =    1 /1,135
2        0    0 1028    0   1   0   0    3   0    0 0.00388 =   4 / 1,032
3        0    0    1 1003   0   0   0    3   2    1 0.00693 =   7 / 1,010
4        0    0    1    0 971   0   4    0   0    6 0.01120 =    11 / 982
5        2    0    0    5   0 882   1    1   1    0 0.01121 =    10 / 892
6        2    3    0    1   1   2 949    0   0    0 0.00939 =     9 / 958
7        1    2    6    0   0   0   0 1019   0    0 0.00875 =   9 / 1,028
8        1    0    1    3   0   4   0    2 960    3 0.01437 =    14 / 974
9        1    2    0    0   4   3   0    2   0  997 0.01189 =  12 / 1,009
Totals 981 1142 1038 1013 977 891 956 1031 964 1007 0.00830 = 83 / 10,000
```

1-liner: call h2o.deeplearning() in R

full run: 10 hours on 10-node cluster
2 hours on desktop gets to 0.9% test set error

Just supervised training
on original 60k/10k dataset:
No data augmentation
No distortions
No convolutions
No pre-training
No ensemble
0.83% test set error:
current world record

Table 1: *Classification error rate comparison: DBN vs. DCN*

DBN [3] (Hinton's)	DBN (MSR's)	DCN (Fine-tuning)	DCN (no Fine-tuning)	Shallow (D)CN (Fine-tuned single la
1.20%	1.06%	0.83%	0.95%	1.10%

H2O

H2O Deep Learning Performance Benchmark

MNIST hand-written digits: 28x28=784 features - 10-class classification (60k/10k train/test)
100 epochs, ReLU, 50% hidden dropout, SGD (constant rate=0.01, no momentum), no L1/L2

Network Topology (neurons per layer)	Keras/Theano GTX 980 Ti GPU mini-batch 32	H2O Deep Learning 1 Xeon server* mini-batch 1	H2O Deep Learning 10 Xeon servers** mini-batch 1
784,**128,128,128**,10	**300 secs** 0.144 train loss 0.106 test loss	**65 secs** 0.018 train loss 0.125 test loss	**12 secs** 0.053 train loss 0.102 test loss
784,**512,512,512**,10	**410 secs** 0.038 train loss 0.064 test loss	**420 secs** 0.001 train loss 0.115 test loss	**56 secs** 0.016 train loss 0.075 test loss
784,**2048,2048,2048**,10	**770 secs** 0.012 train loss 0.058 test loss	**4800 secs** 0.000 train loss 0.105 test loss	**680 secs** 0.013 train loss 0.078 test loss

Note: different mini-batch sizes - GPU: **32** -> faster (approximate) vs H2O/CPU: **1** -> fits better

*Dual E5-2650 (8 cores, 2.6GHz), 10GbE **10 model averaging steps

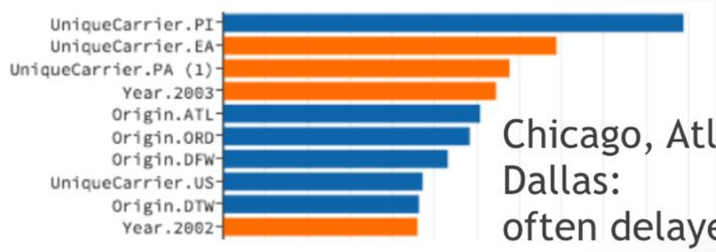
Significant Performance Gains with Deep Learning

Predict departure delay (Y/N) on 20 years of airline flight data
(116M rows, 31 cols, categorical + numerical data with missing values)

H2O GLM: 30 secs
alpha=0.5, lambda=1.379e-4 (auto)

MSE	0.226182
r2	0.074236
logloss	0.643529
AUC	0.657474

AUC: 0.657



Chicago, Atlanta, Dallas: often delayed

Feature importances

H2O Deep Learning: 100 secs
4 hidden ReLU layers of 20 neurons, 1 epoch

MSE	0.215205
r2	0.119492
logloss	0.619137
AUC	0.699005

AUC: 0.699

Features have non-linear impact

8-node EC2 cluster: 64 virtual cores, 1GbE network

H2O

H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Administration
5. Sparkling Water
6. Productionalizing Models

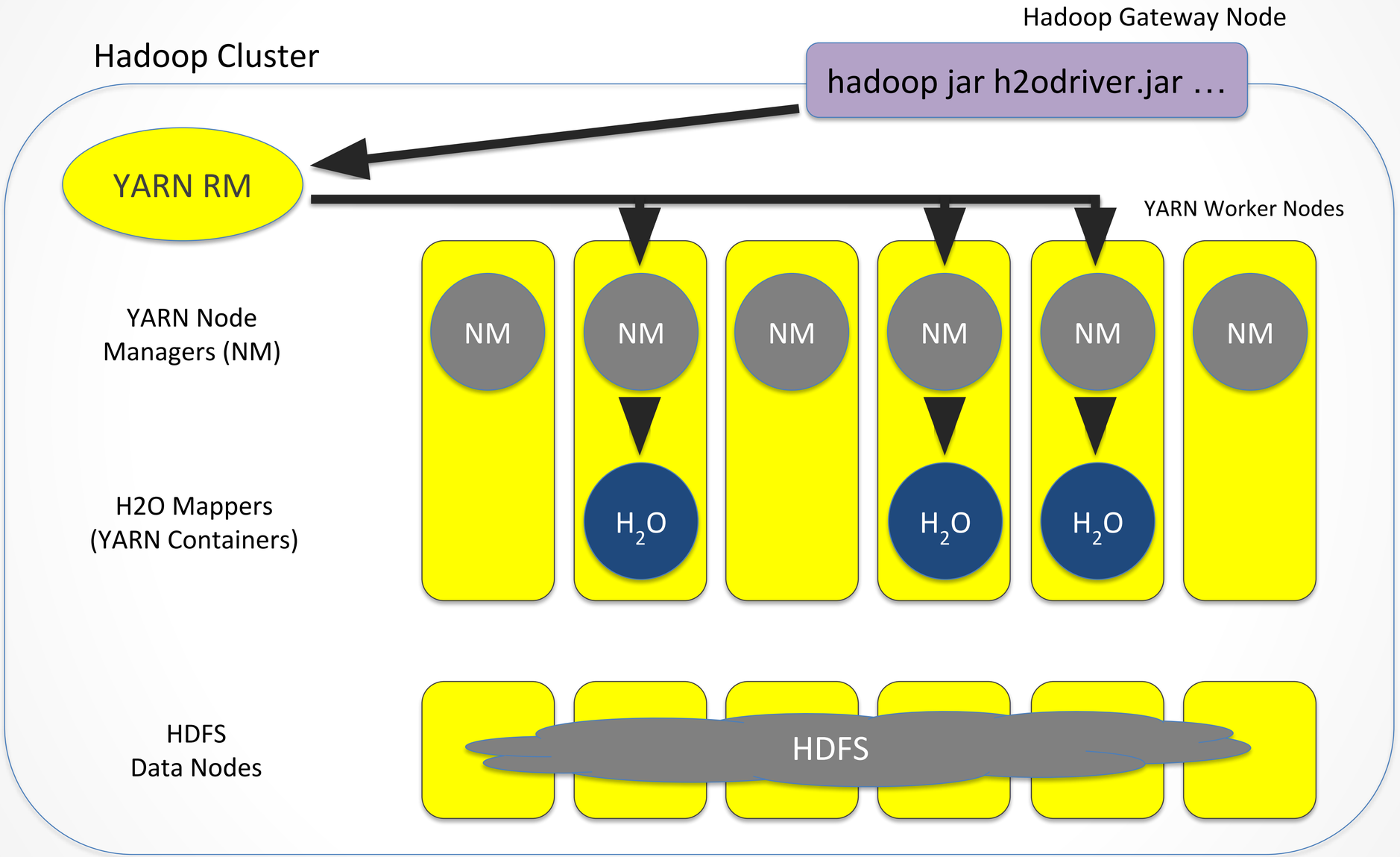
H2O on Hadoop (and Yarn)

- **On Demand** H2O launches as a mapper task that never reduces. User only needs permission to launch map reduce task in order to spin up their own H2O cluster:

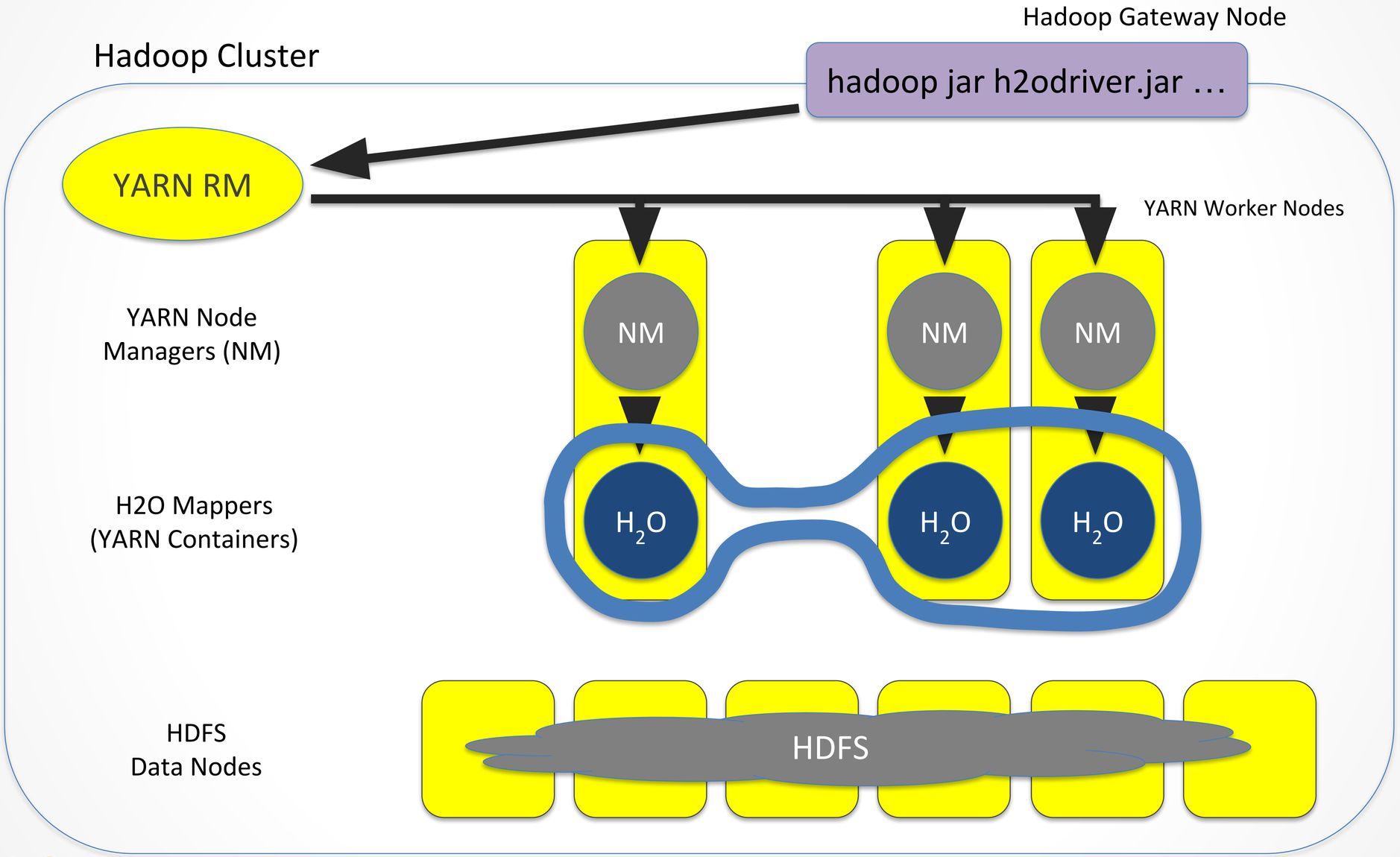
  hadoop jar h2odriver.jar -nodes 10 -mapperXmx 50g

- **Update frequently** The installation of H2O is essentially a jar file so in order to upgrade the user only needs to download a new driver jar file.
- **Yarn to manage resources** Special queues can be allocated to users to manage the containers and memory allowed.
- **Access to HDFS and Hive Tables** H2O can be built with any distribution's class libraries to allow access to HDFS whether launched with Yarn or in standalone mode.

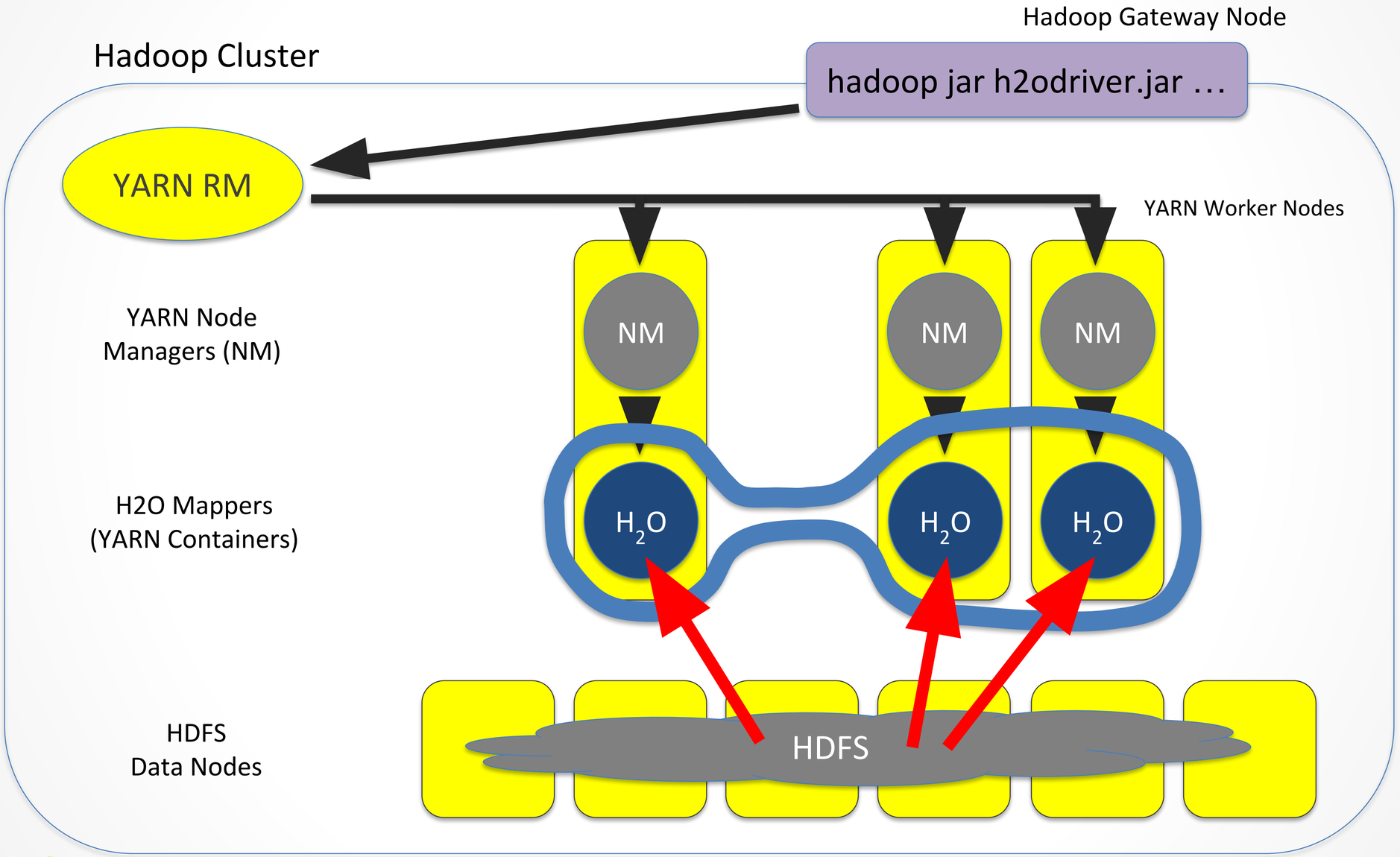
H2O on YARN Deployment



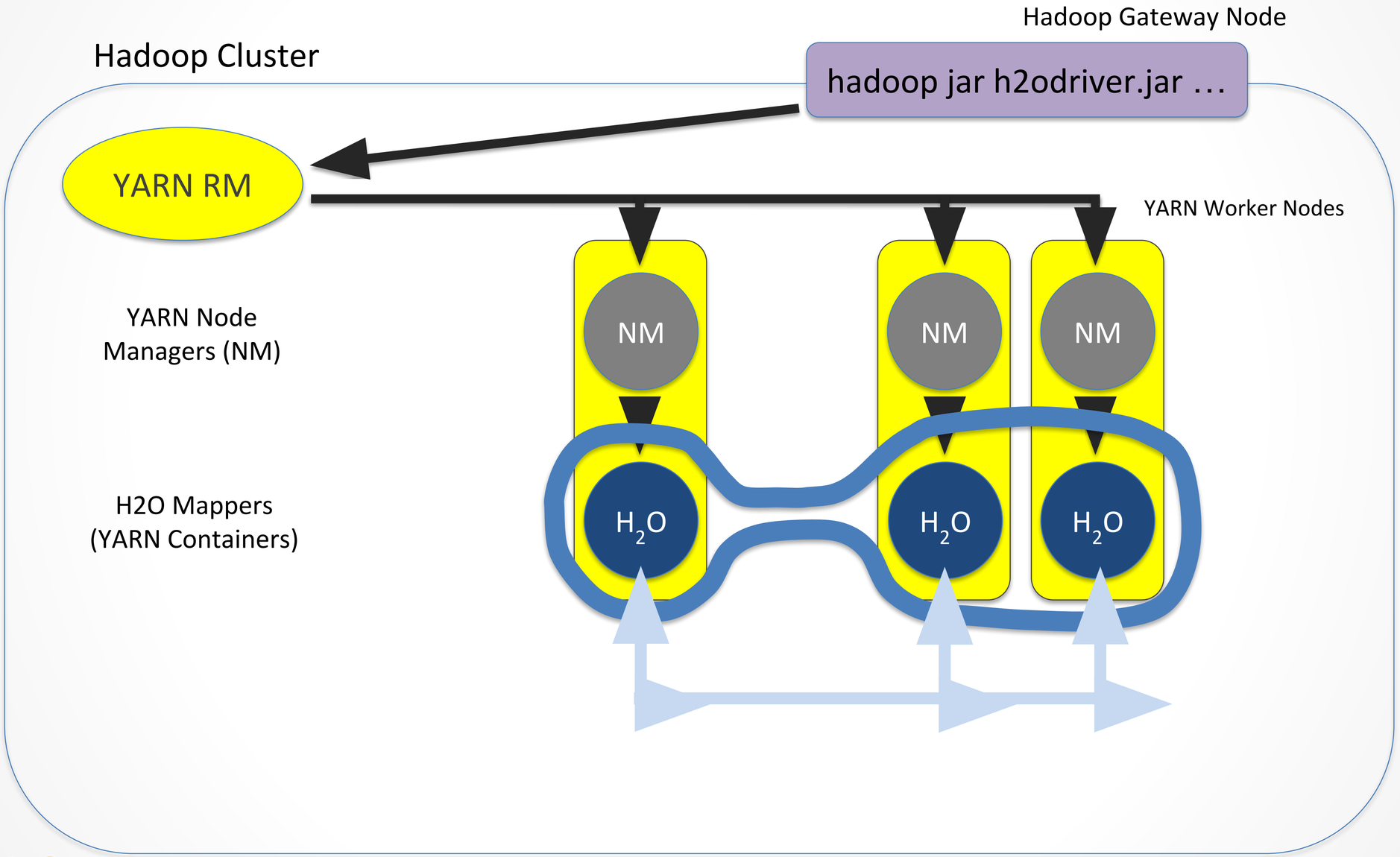
Now You Have an H2O Cluster



Read Data from HDFS *Once*



Build Models *in-Memory*



H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Adminstration
5. Sparkling Water
6. Productionalizing Models

What is Sparkling Water?

Provides

Transparent integration of H2O with Spark ecosystem

Transparent use of H2O data structures and algorithms with Spark API

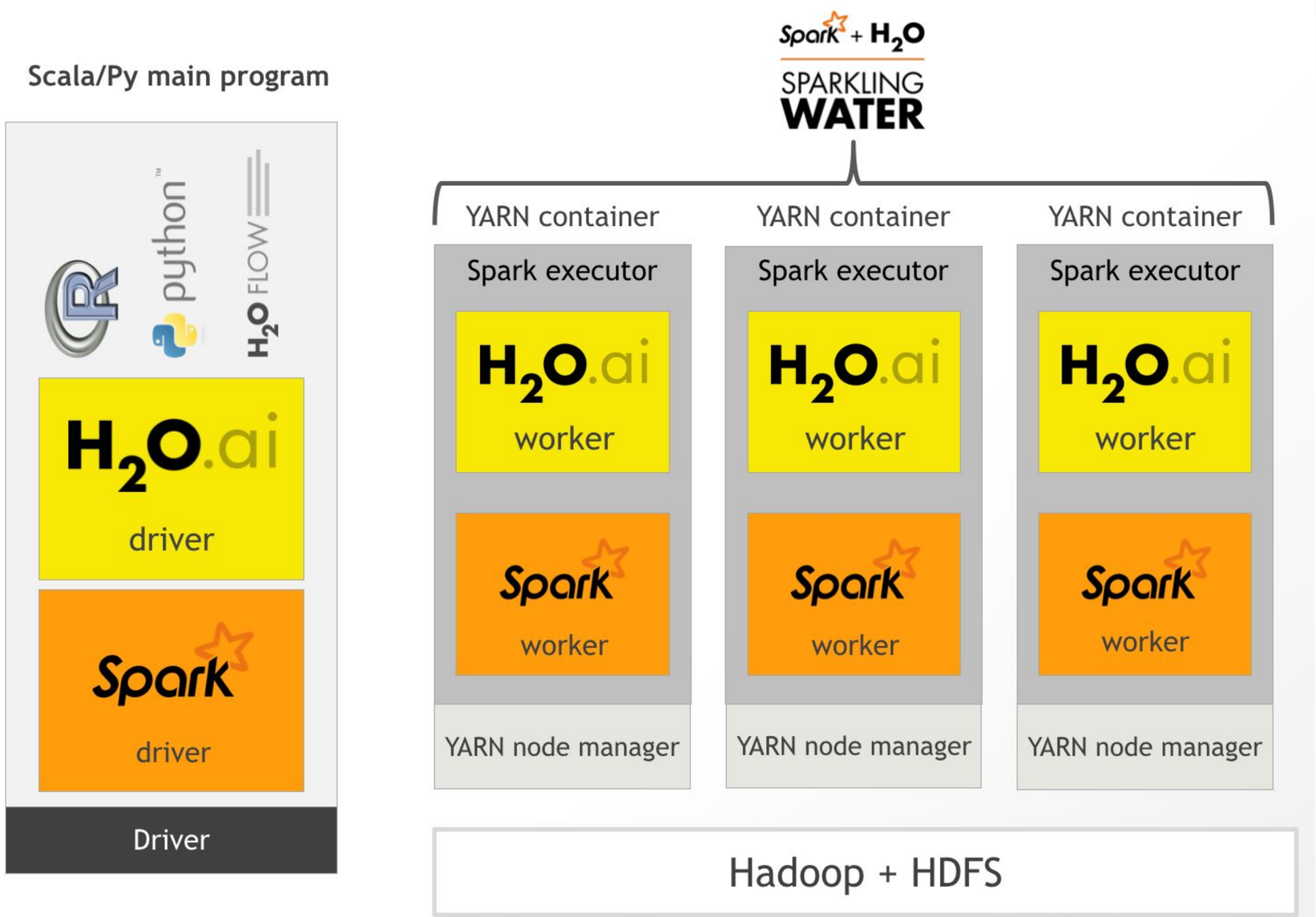
Excels in existing Spark workflows requiring advanced Machine Learning algorithms

Platform for building

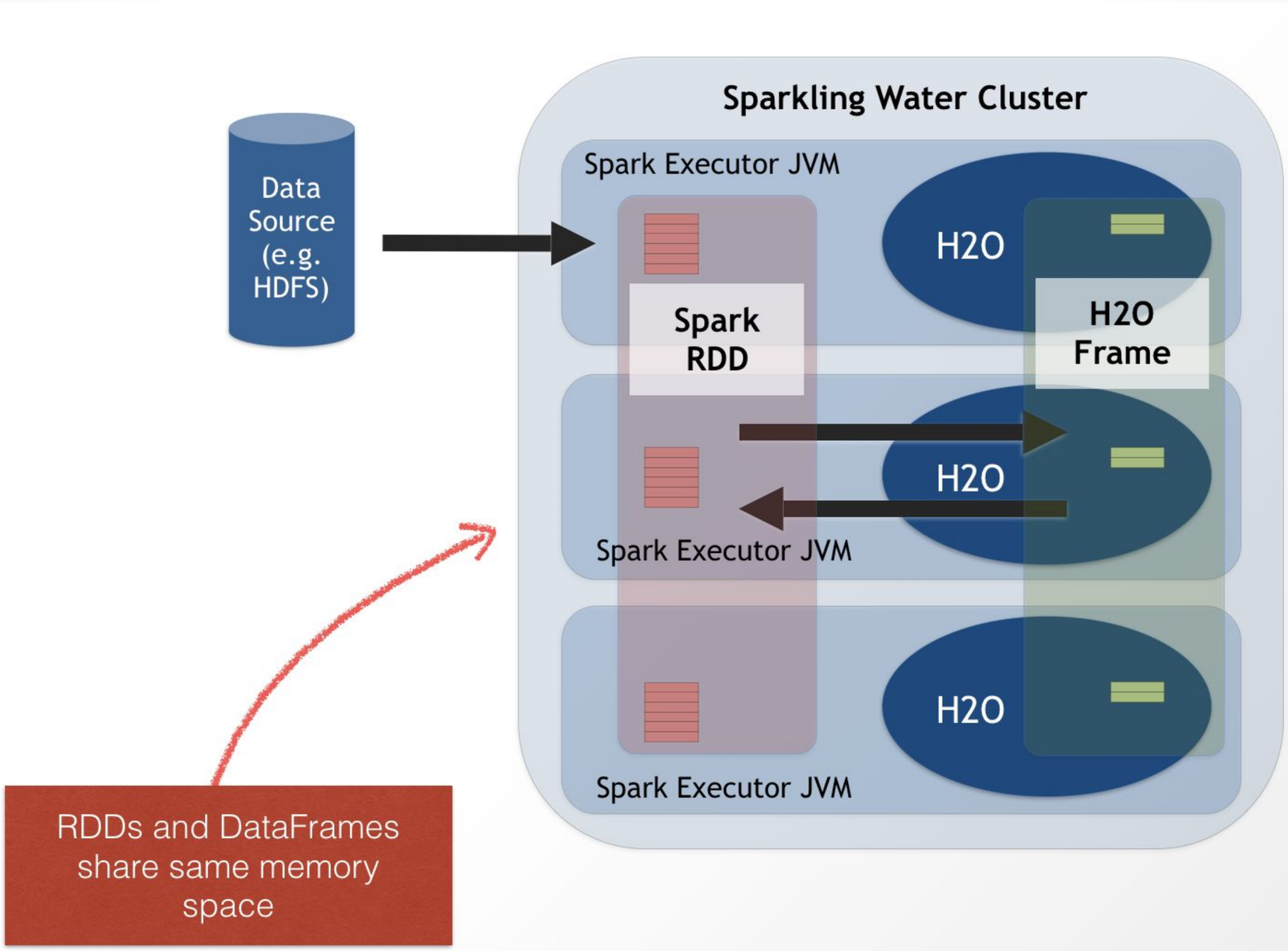
Smarter ML Applications

H_2O

What is Sparkling Water?



Spark and H2O Data Sharing



Simple Spam Detector

OR

Detect spam text messages

H2O

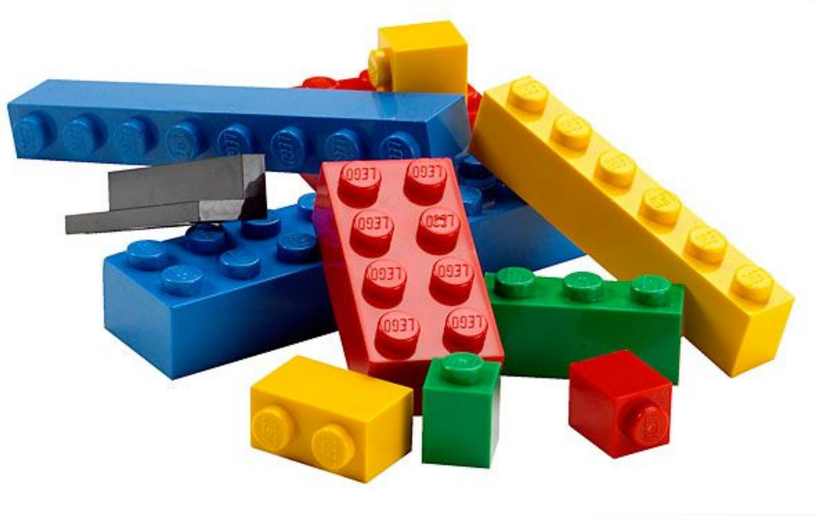
Data example

	A	B
1	ham	Ok... But they said i've got wisdom teeth hidden inside n mayb need 2 remove.
2	ham	U thk of wat to eat tonight.
3	ham	I dunno until when... Lets go learn pilates...
4	spam	Someonone you know is trying to contact you via our dating service! To find out who it could be call from your mobile or landline 09064015307 BOX334SK38ch
5	ham	Ok c u then.
6	spam	URGENT! We are trying to contact U. Todays draw shows that you have won a £800 prize GUARANTEED. Call 09050003091 from land line. Claim C52. Valid12hrs only
7	spam	Not heard from U4 a while. Call 4 rude chat private line 01223585334 to cum. Wan 2C pics of me gettin shagged then text PIX to 8552. 2End send STOP 8552 SAM xxx
8	ham	staff.science.nus.edu.sg/~phyhcmk/teaching/pc1323
9	ham	Thank god they are in bed!
10	ham	Hey tmr meet at bugis 930 ?
11	spam	You are a winner you have been specially selected to receive £1000 cash or a £2000 award. Speak to a live operator to claim call 087123002209am-7pm. Cost 10p
12	spam	URGENT! Your Mobile No. was awarded £2000 Bonus Caller Prize on 5/9/03 This is our final try to contact U! Call from Landline 09064019788 BOX42WR29C, 150PPM
13	spam	Loan for any purpose £500 - £75,000. Homeowners + Tenants welcome. Have you been previously refused? We can still help. Call Free 0800 1956669 or text back 'help'
14	ham	Haha... Sounds crazy, dunno can tahan anot...
15	spam	You have won ?spam 000 cash or a ?2,000 prize! To claim, call09050000327
16	ham	Sorry i din lock my keypad.
17	ham	Thanx but my birthday is over already.
18	spam	FREE for 1st week! No1 Nokia tone 4 ur mobile every week just txt NOKIA to 8077 Get txting and tell ur mates. www.getzed.co.uk POBox 36504 W45WQ 16+ norm150p/tone
19	spam	Congratulations - Thanks to a good friend U have WON the £2,000 Xmas prize. 2 claim is easy, just call 08712103738 NOW! Only 10p per minute. BT-national-rate
20	ham	Me n him so funny...
21	spam	pdate_Now - Double mins and 1000 txts on Orange tariffs. Latest Motorola, SonyEricsson & Nokia & Bluetooth FREE! Call MobileUpd8 on 08000839402 or call2optout/!YHL
22	ham	Ok...
23	ham	Yup no more already... Thanx 4 printing n handing it up.
24	ham	Anything lor. Juz both of us lor.
25	ham	It's é only $140 ard...É rest all ard $180 at least...Which is é price 4 é 2 bedrm ($900)
26	ham	Oh oh... Den muz change plan liao... Go back have to yan jiu again...
27	ham	Ok lor then we go tog lor...
28	ham	Okay lor... Wah... like that def they wont let us go... Haha... What did they say in the terms and conditions?
29	ham	Dunno lei... I thk mum lazy to go out... I neva ask her yet...
30	ham	THATS ALRITE GIRL, U KNOW GAIL IS NEVA WRONG!!TAKE CARE SWEET AND DONT WORRY.C U L8TR HUN!LOVE Yaxxx

ML Workflow

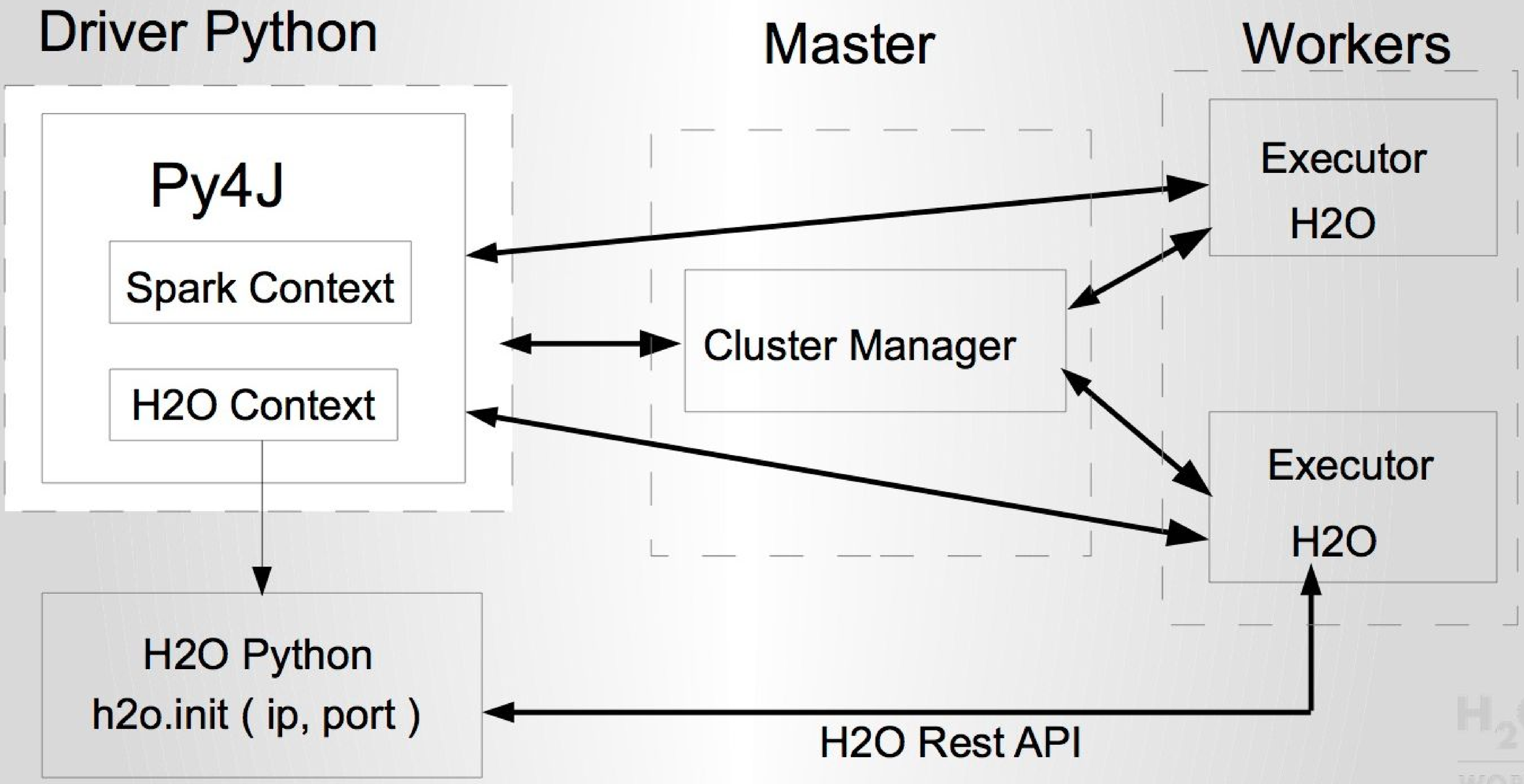
Goal: For a given text message identify if it is spam or not

1. Extract data
2. Transform, tokenize messages
3. Build Tf-IDF model
4. Create and evaluate Deep Learning model
5. Use the model to detect spam



Demo:
Sparkling Shell
Sparkling Water in Flow

PySparkling Architecture



Why use PySparkling

- Automatic Parallelization and less lines of code
- Much Faster on big data - uses H2O's rest API calls to connect to H2O Cluster

Demo:
PySparkling With Chicago Crime Data

H2O Training

1. Introduction to H2O
2. H2O Best Practices [Munging]
3. H2O Best Practices [Modeling]
4. Hadoop Adminstration
5. Sparkling Water
6. Productionalizing Models

The Problem

- Goal:
  - Move from prototype to production

- Road block:
  - Prototyping Environment Cages Your:
    - Feature preprocessing
    - Models
    - Ideas

Lending Club Dataset

- Loan data from 2007 up until 2015 including rejected applications and accepted applications.
- Of the 500k accepted applicants about 160k loans have either been completely paid off or defaulted.
- There are about 4million applicants in the rejected loans dataset.
- **Use Case 1** Predict the likelihood of a user defaulting based on the information supplied when applying for a loan.
- **Use Case 2** Determine the interest rate Lending Club would have offered the user based on the information supplied when applying for a loan.
- **Use Case 3** Based on the subset of information available in both rejected and accepted loans, predict the likelihood Lending Club will approve your loan request.

Data Dictionary

Predictor Variable	Description	Units
loan_amnt	Requested loan amount	US dollars
term	Loan term length	months
emp_length	Employment length	years
home_ownership	Housing status	categorical
annual_inc	Annual income	US dollars
verification_status	Income verification status	categorical
purpose	Purpose for the loan	categorical
addr_state	State of residence	categorical
dti	Debt to income ratio	%
delinq_2yrs	Number of delinquencies in the past 2 years	integer
revol_util	Revolving credit line utilized	%
total_acc	Total accounts (number of credit lines)	integer
longest_credit_length	Age of oldest active account	years

Response Variable	Description	Model Category
bad_loan	Is this loan likely to be bad?	Binomial classification
int_rate	What should the interest rate be?	Regression
app_status	Was the application approved?	Binomial classification

Example Model Output

Bad Loan Model

Algorithm: GBM
Model category: Binary Classification
ntrees: 100
max_depth: 5
learn_rate: 0.05

AUC on valid: .685
max F1: 0.202

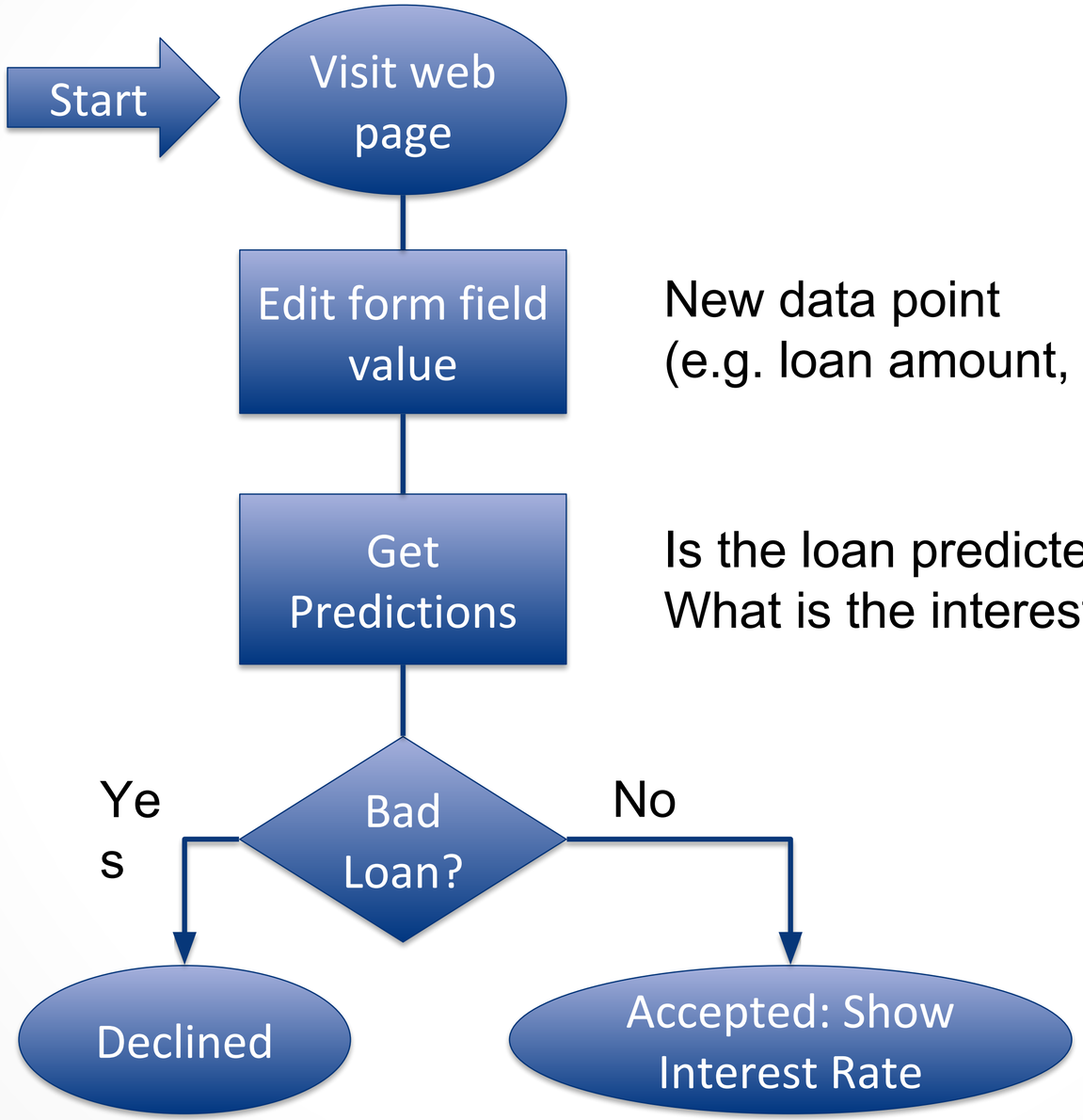
Interest Rate Model

Algorithm: GBM
Model category: Regression

ntrees: 100
max_depth: 5
learn_rate: 0.05

MSE: 11.1
R2: 0.424

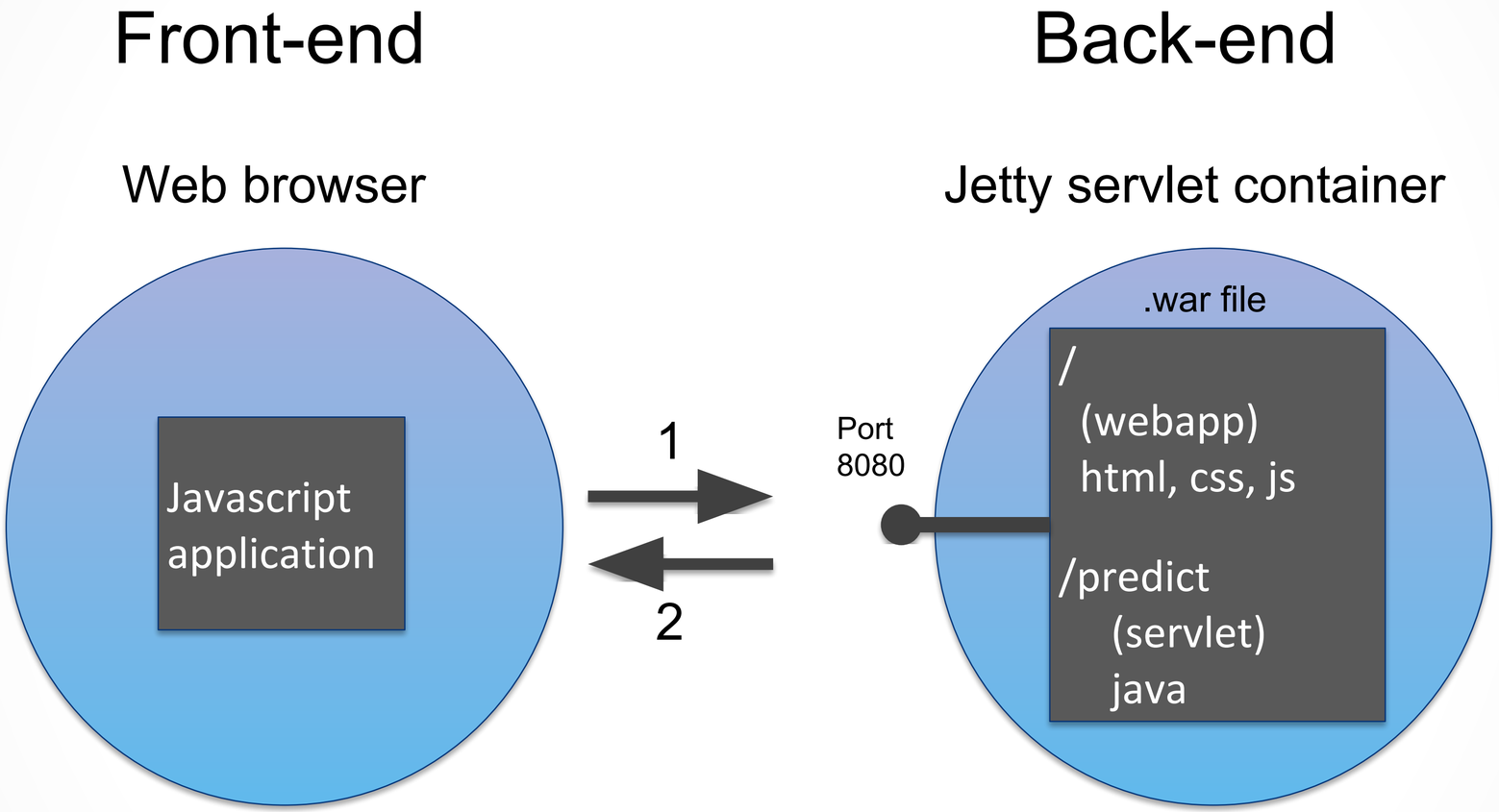
WORKFLOW FOR THIS APP



New data point
(e.g. loan amount, income)

Is the loan predicted to be bad?
What is the interest rate?

APP ARCHITECTURE DIAGRAM



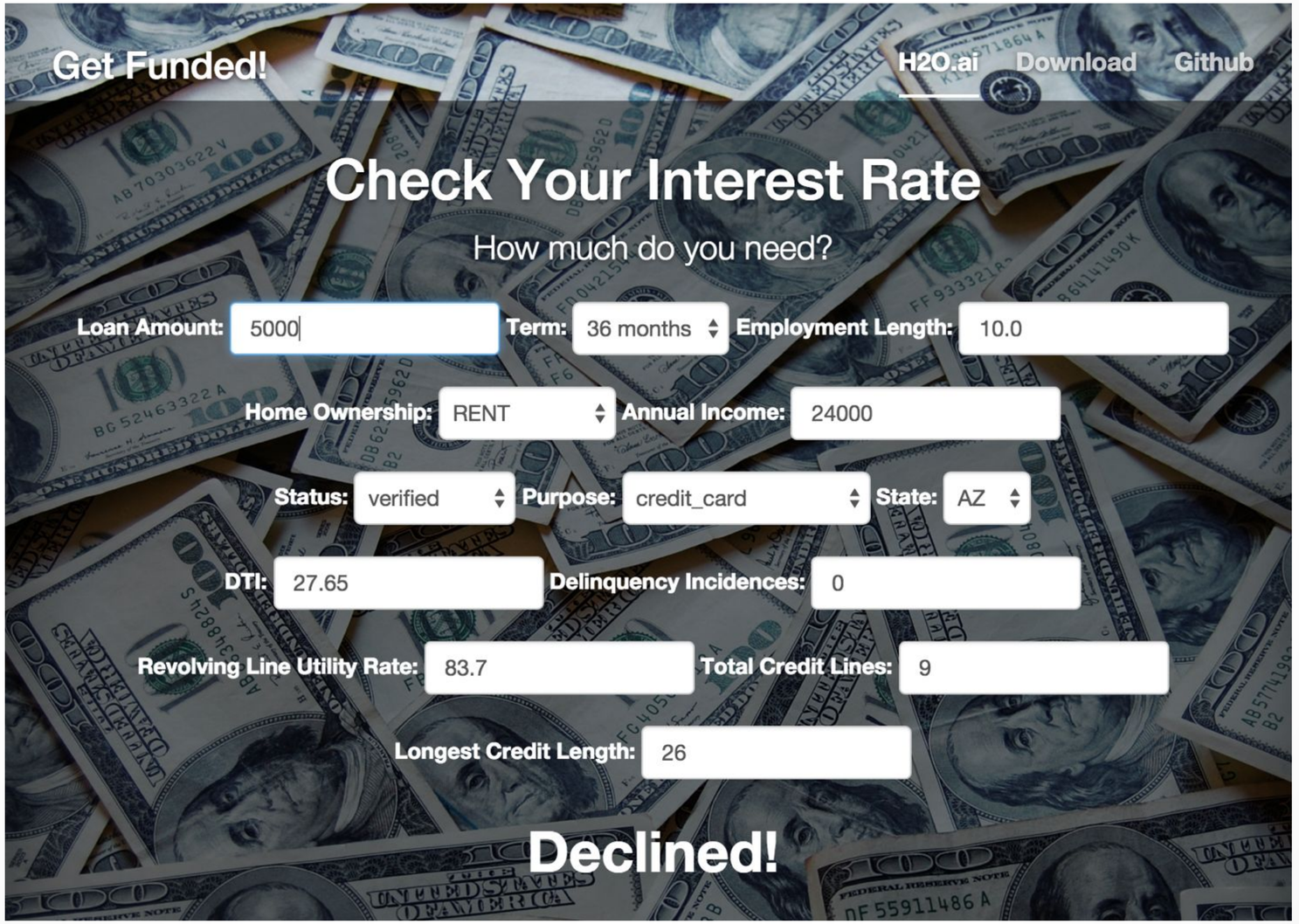
1. HTTP GET with query parameters (loan_amt, annual_inc, etc.)
2. JSON response with predictions

SOFTWARE PIECES

- Offline
  - R + H2O (model building)

- Online
  - Front-end
    - Web browser
    - JavaScript application (run in the browser)
  - Back-end
    - Jetty servlet container
    - H2O-generated model POJO (hosted by servlet container)

Predict Loan Defaults



H2O.ai
Machine Intelligence