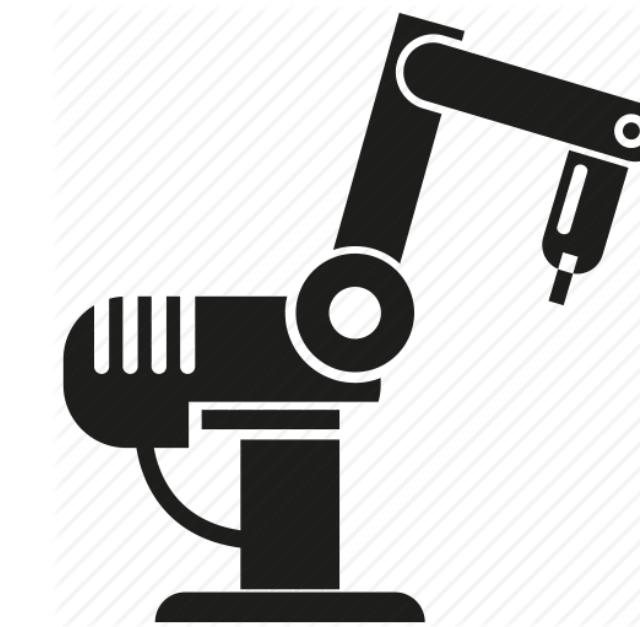


# Scalable Automatic Machine Learning in H2O



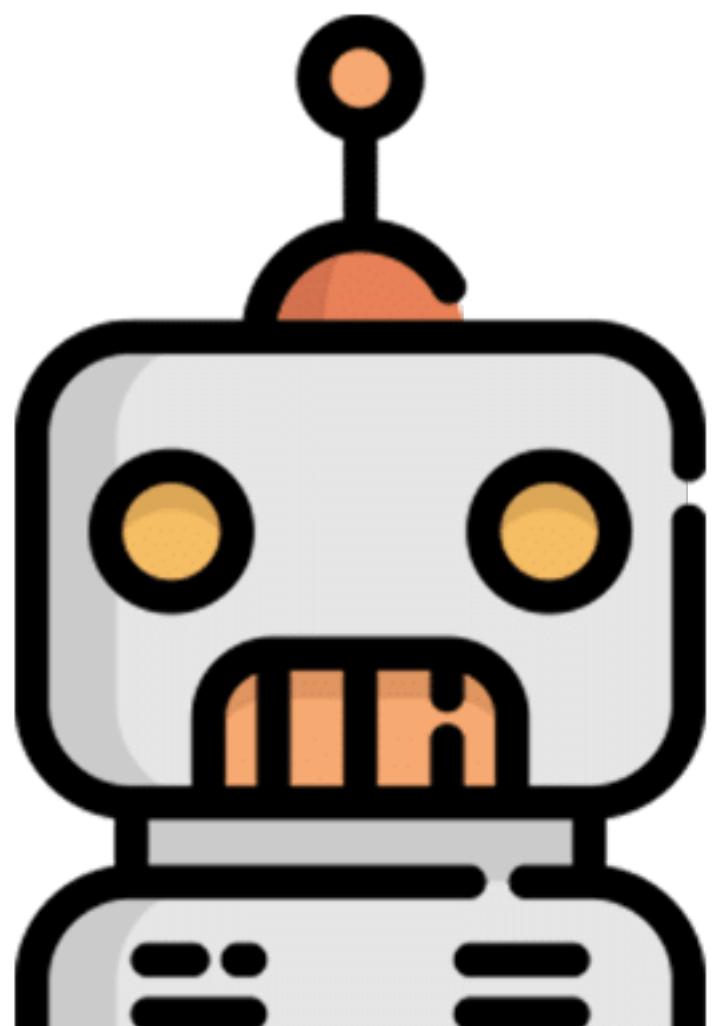
May 2018  
2018 Symposium on Data Science and Statistics



Navdeep Gill  
@Navdeep\_Gill\_

# Agenda

- H2O Platform
- Automatic Machine Learning (AutoML)
- H2O AutoML Overview
- AutoML Pro Tips
- Demos



# Company Overview

<b>Founded</b>	2011 Venture-backed, Debuted in 2012
<b>Products</b>	<ul style="list-style-type: none"><li>• <b>H<sub>2</sub>O Open Source In-Memory AI Prediction Engine</b></li><li>• Sparkling Water (H<sub>2</sub>O + Spark)</li><li>• H2O4GPU (H2O on GPUs)</li><li>• Enterprise Steam</li><li>• Driverless AI</li></ul>
<b>Mission</b>	Operationalize Data Science & Provide a Platform to Build Beautiful Data Products
<b>Team</b>	<p>75+ employees</p> <ul style="list-style-type: none"><li>• Distributed Systems Engineers doing Machine Learning</li><li>• World-class Visualization Designers</li></ul>
<b>Headquarters</b>	Mountain View, CA



3

# Scientific 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, *Generalized Additive Models*
- 108,404 citations (via Google Scholar)



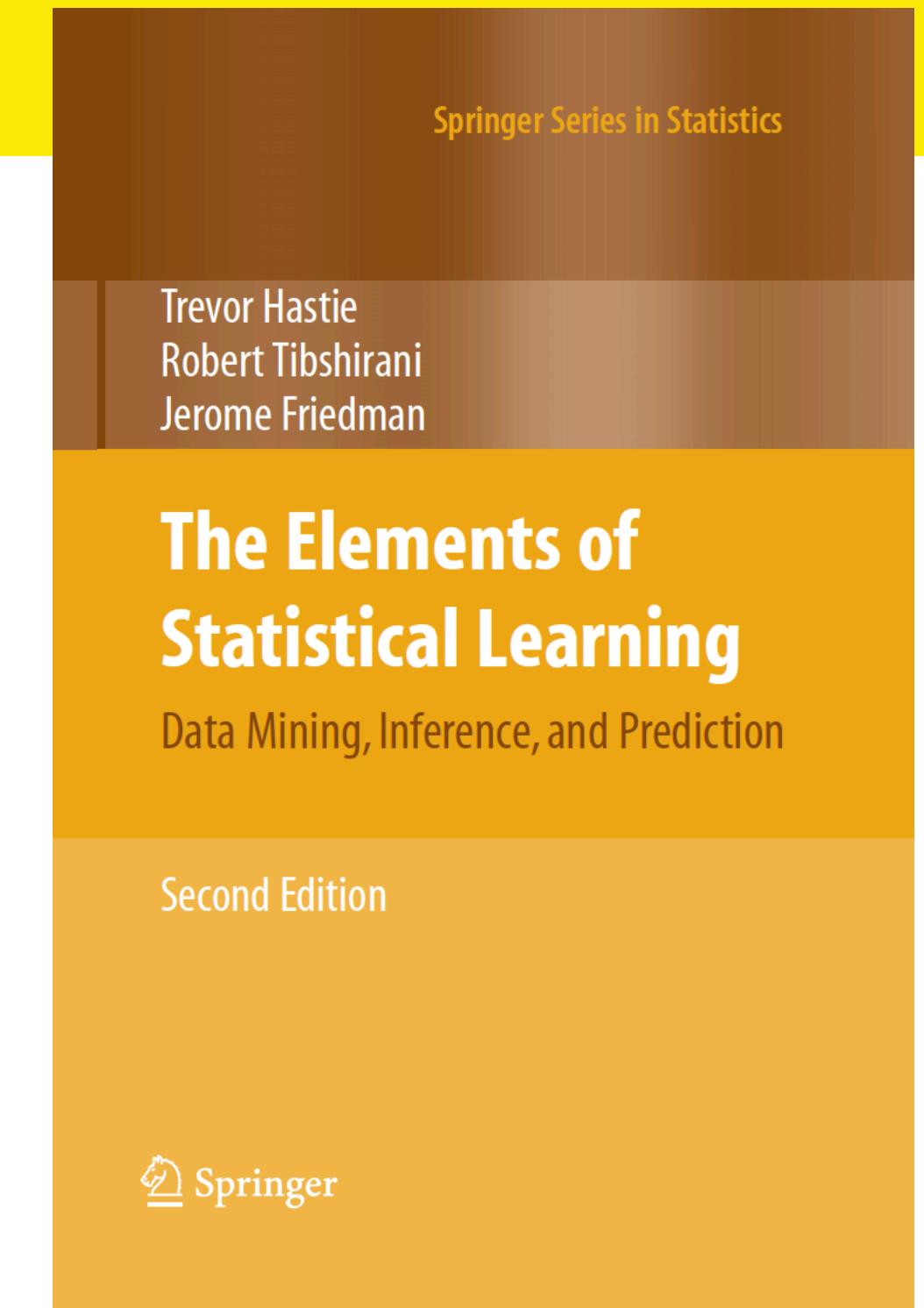
## Dr. Robert Tibshirani

- 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. Steven 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*



# The World's Best Machine Learning Platform

H2O.ai makes machine learning accessible and allows business users to extract insights from data, without needing expertise in deploying or tuning machine learning models.

[SEE HOW](#)

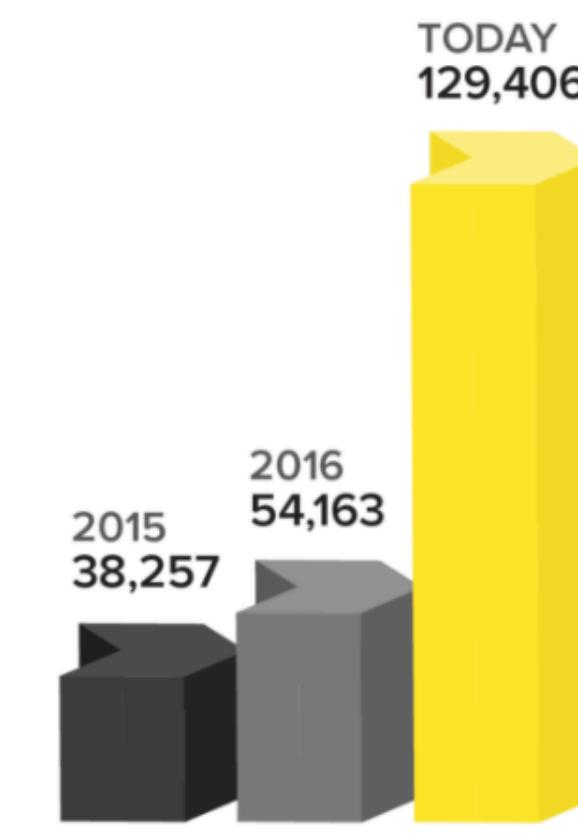
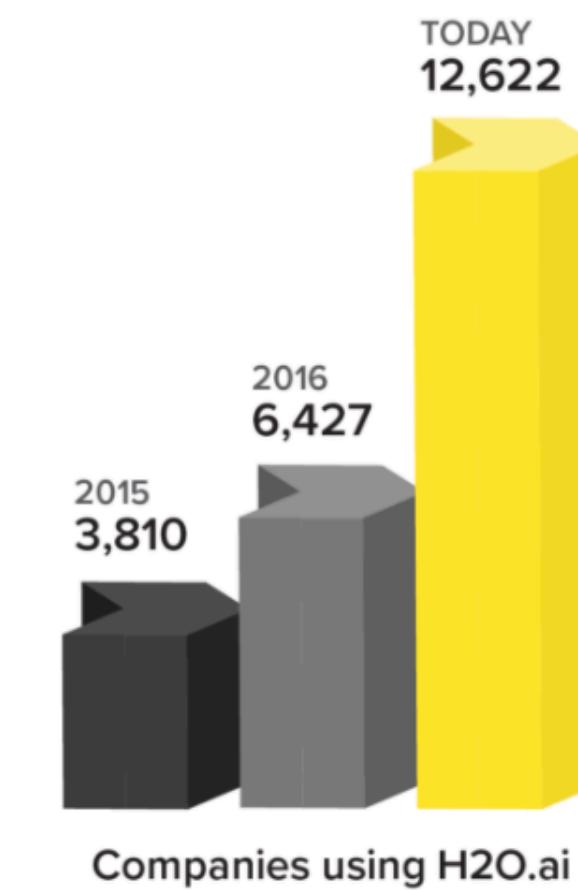
222 OF FORTUNE  
THE 500



8 OF TOP 10 BANKS

7 OF TOP 10 INSURANCE COMPANIES

4 OF TOP 10 HEALTHCARE COMPANIES



# What is H2O?

**Java-Based Software for In-Memory Data Modeling**

**Open Source**



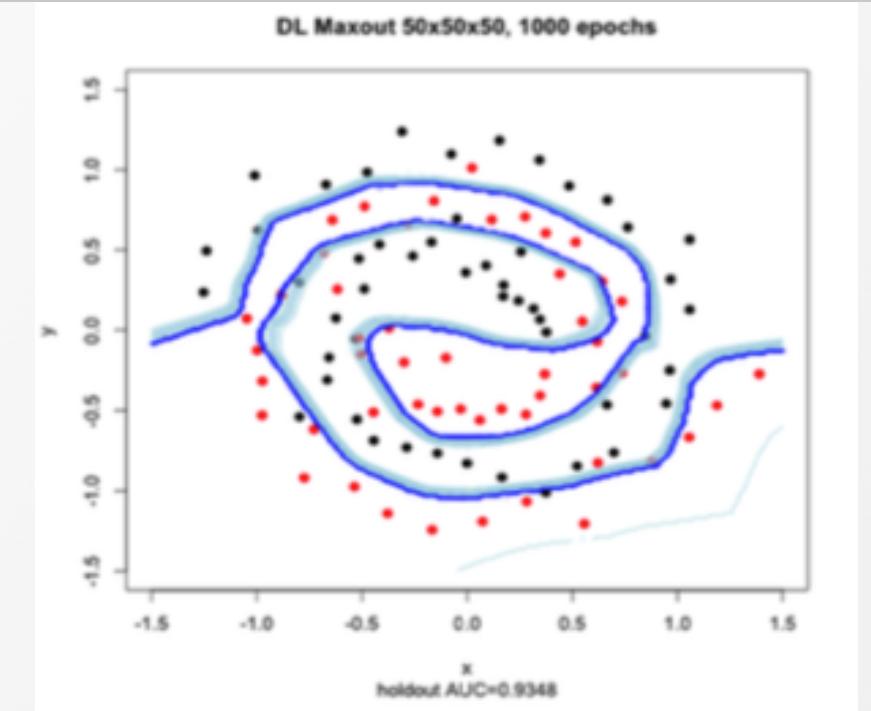
**Big Data Ecosystem**



**Flexible Interface**

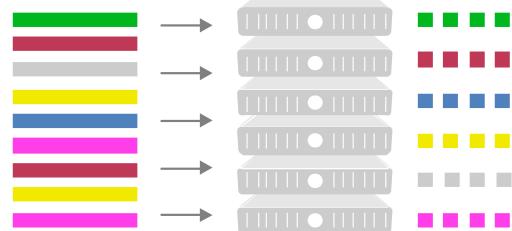
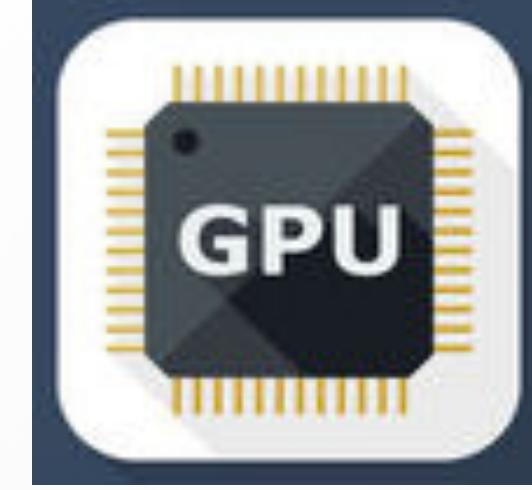


**Smart and Fast Algorithms**

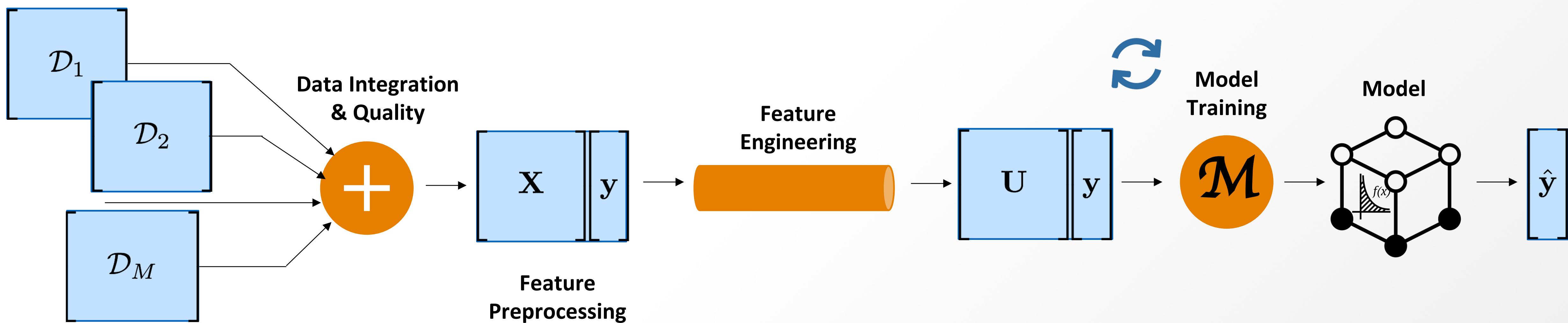


# What is H2O?

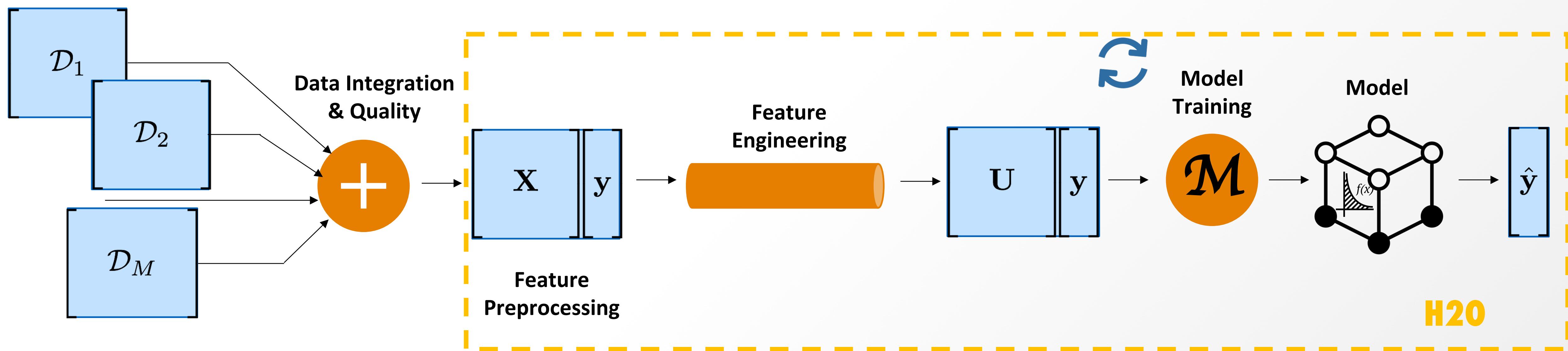
## Java-Based Software for In-Memory Data Modeling

Scalability and Performance	Rapid Model Deployment	GPU Enablement*	Cloud Integration
 <ul style="list-style-type: none"><li>Distributed In-Memory Computing Platform</li><li>Distributed Algorithms</li><li>Fine-Grain MapReduce</li></ul>	<ul style="list-style-type: none"><li>Highly portable models deployed in Java (POJO)</li><li>Automated and streamlined scoring service deployment with Rest API*</li></ul>		  

# The Machine Learning Pipeline



# Where H2O Fits



# Current Algorithm Overview

## Statistical Analysis

---

- Linear Models (GLM)
- Naïve Bayes

## Ensembles

---

- Random Forest
- Gradient Boosting Machine
- Stacking / Super Learner

## Deep Neural Networks

---

- MLP
- Autoencoder
  - Anomaly Detection
  - Deep Features

## Clustering

---

- K-Means (Auto-K)

## Dimension Reduction

---

- Principal Component Analysis
- Generalized Low Rank Models

## Word Embedding

---

- Word2Vec

## Time Series

---

- iSAX

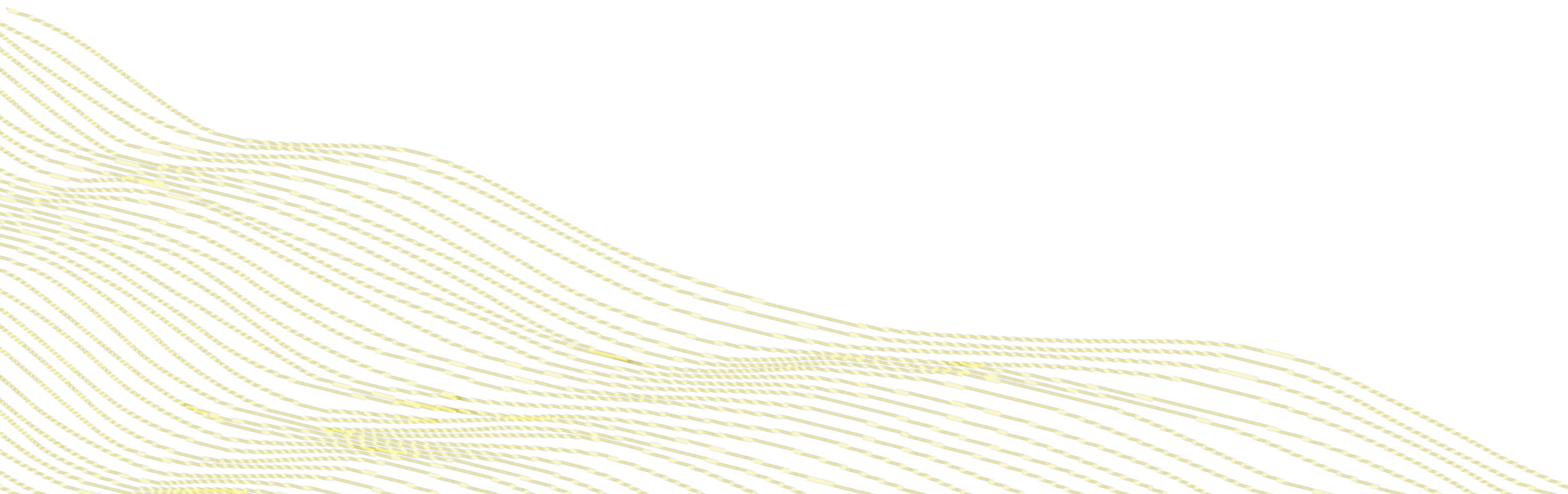
## Machine Learning Tuning

---

- Hyperparameter Search
- Early Stopping

# H2O Core

*Behind the Scenes*



# How H<sub>2</sub>O Core Works



# How H2O Core Works

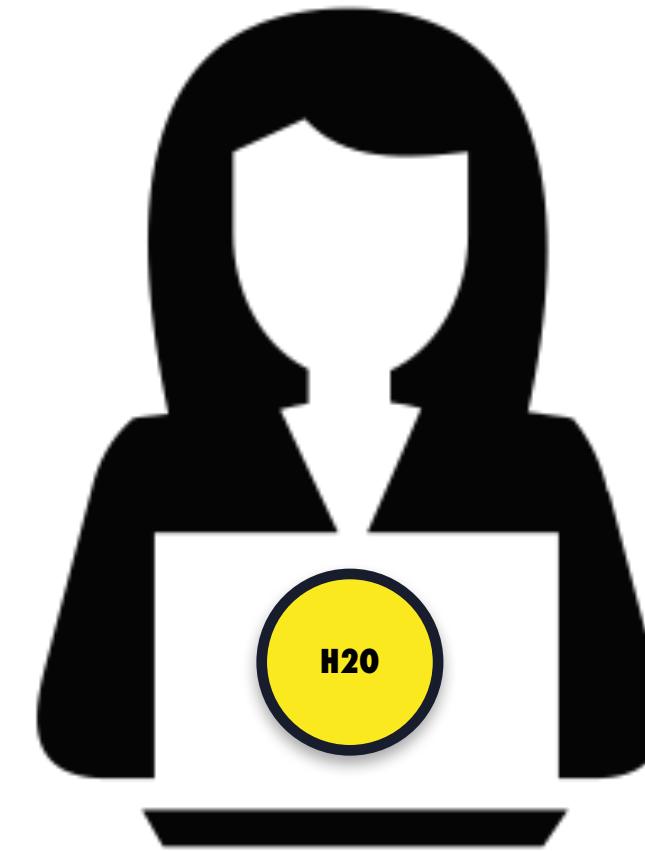


(just a java application)

# How H2O Core Works



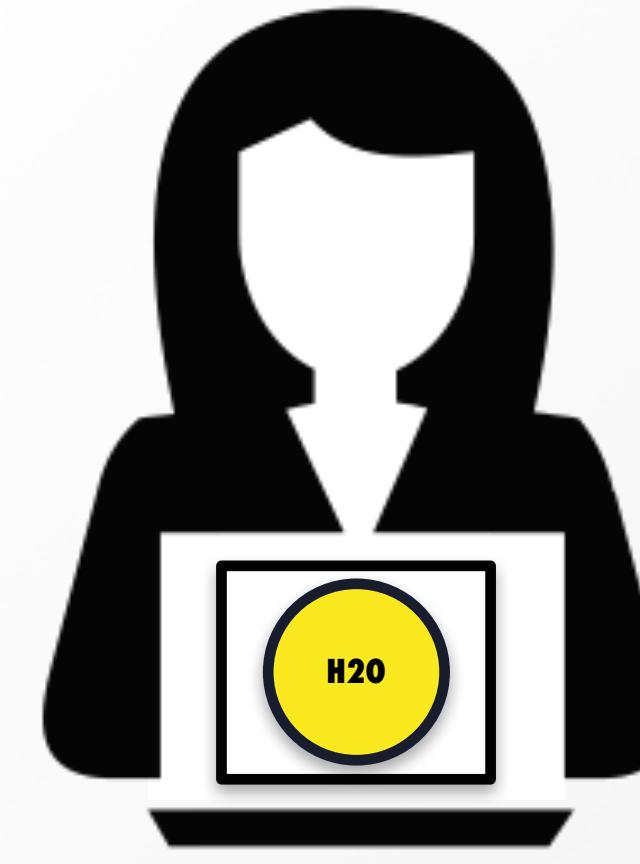
# How H2O Core Works



on a laptop

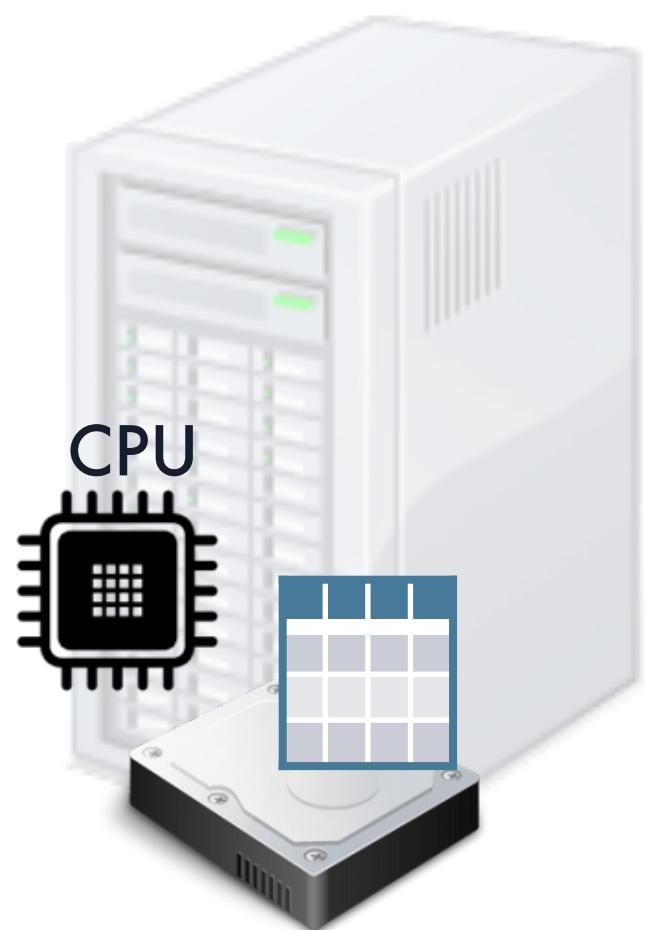


on a virtual machine

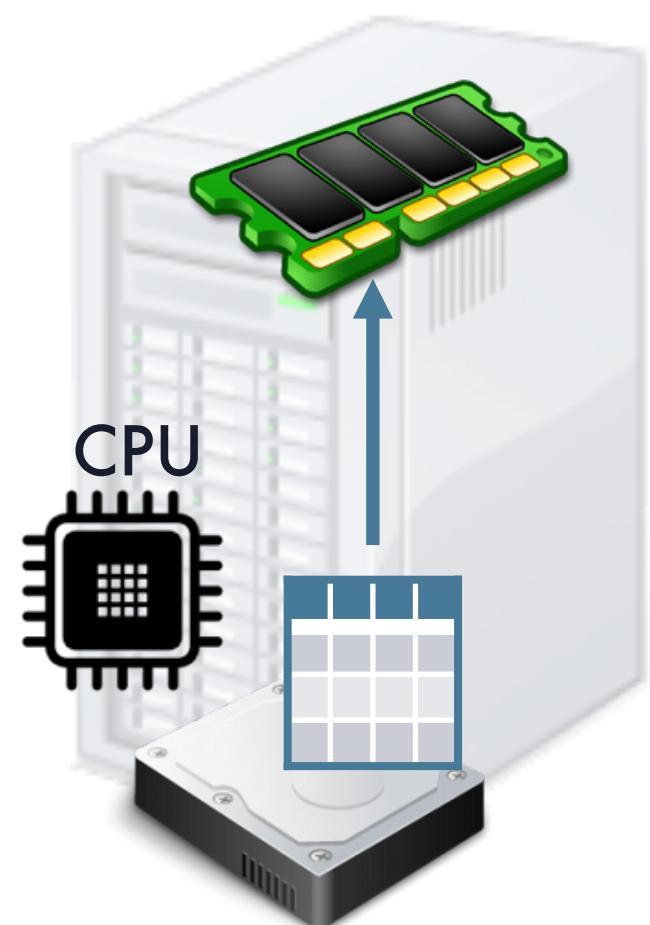


in a container

# H2O Core

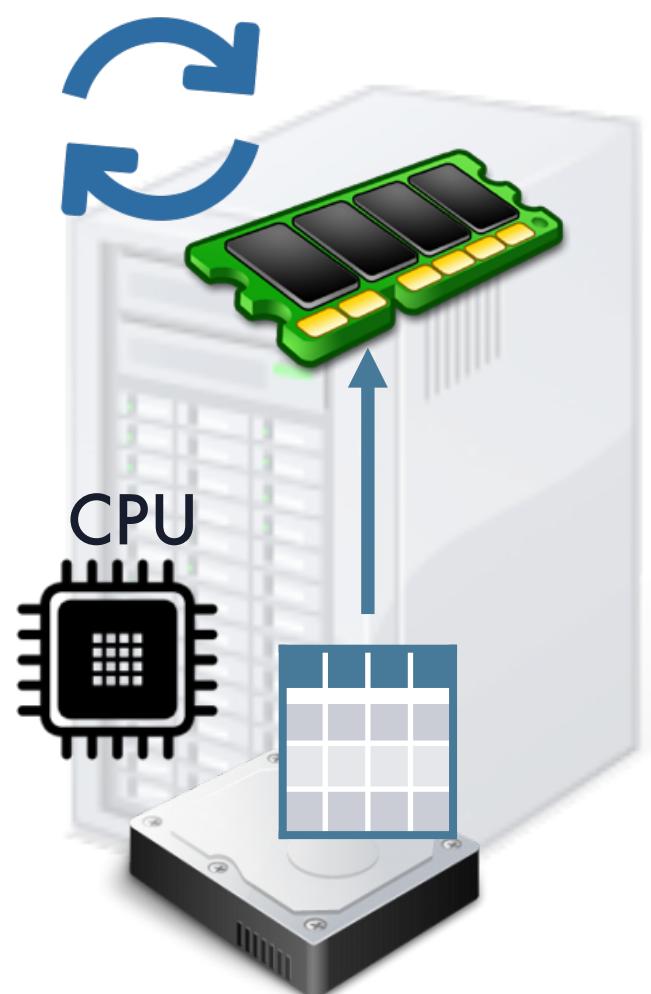


# H2O Core



# H2O Core

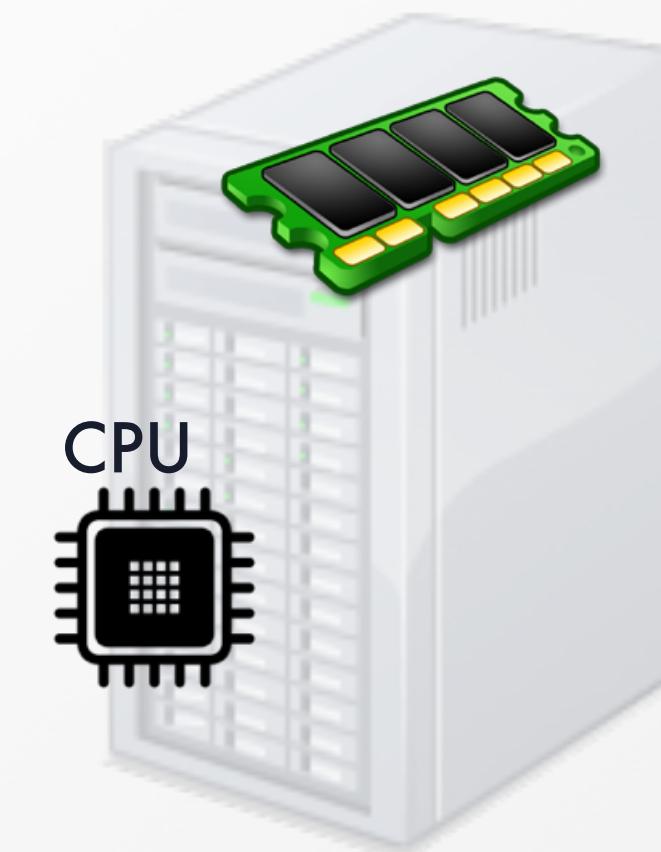
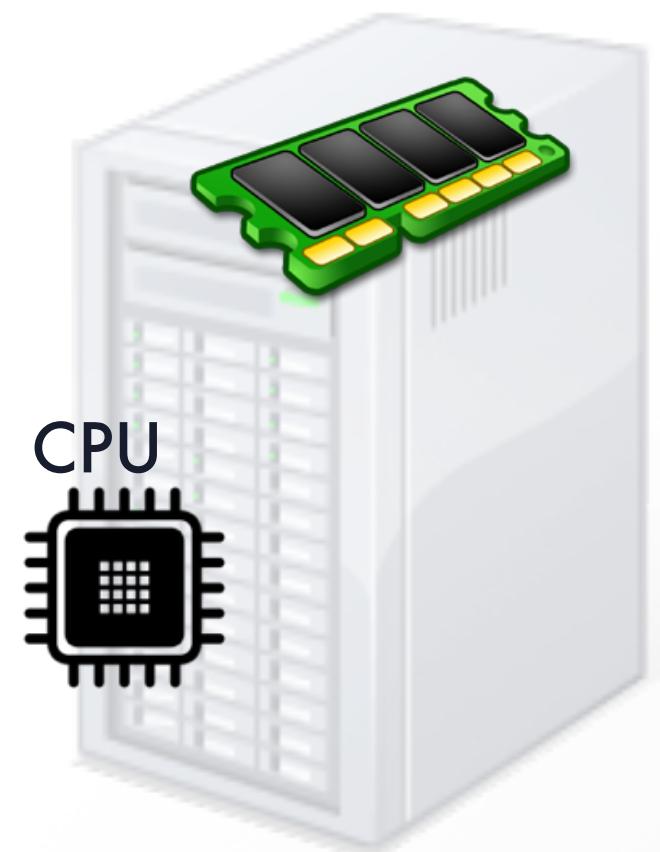
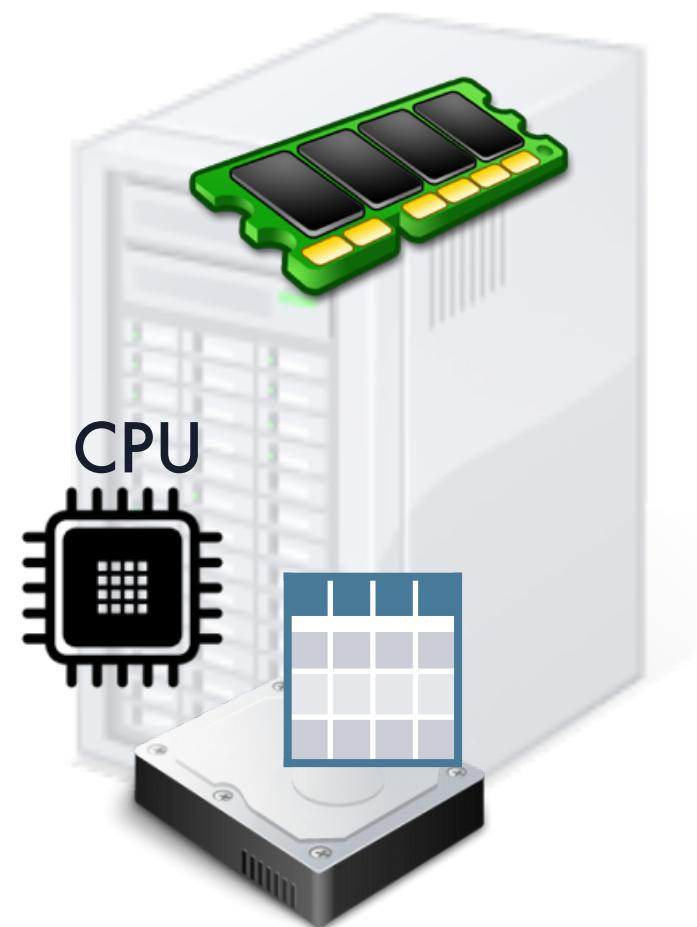
Model Building



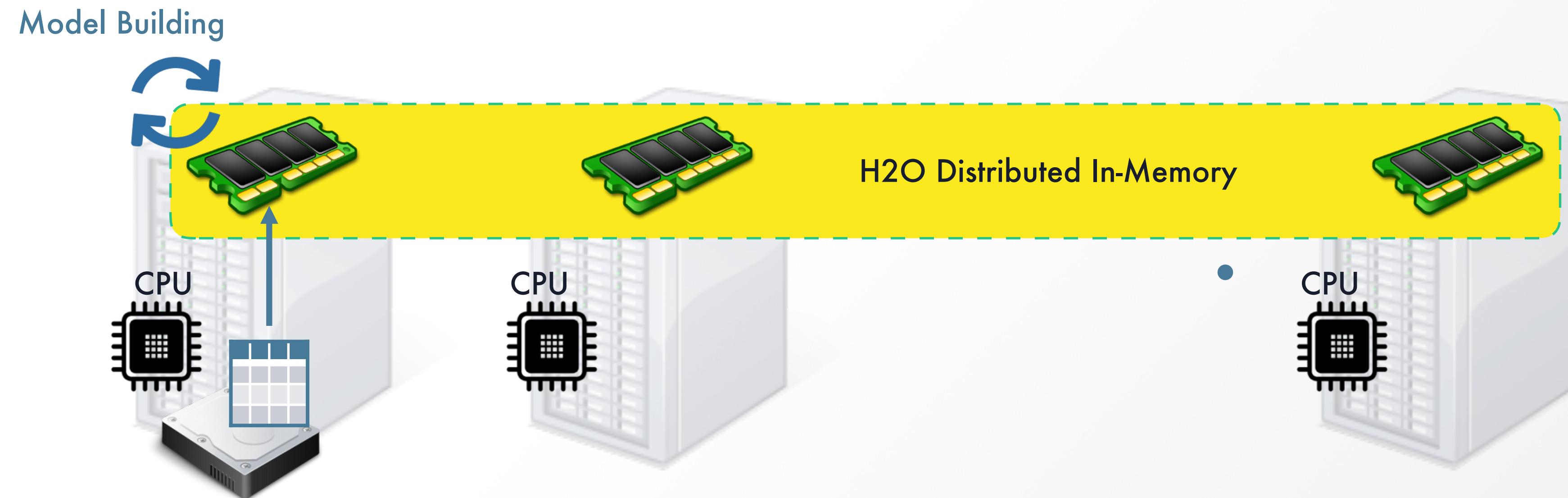
# H2O Core



# H2O Core

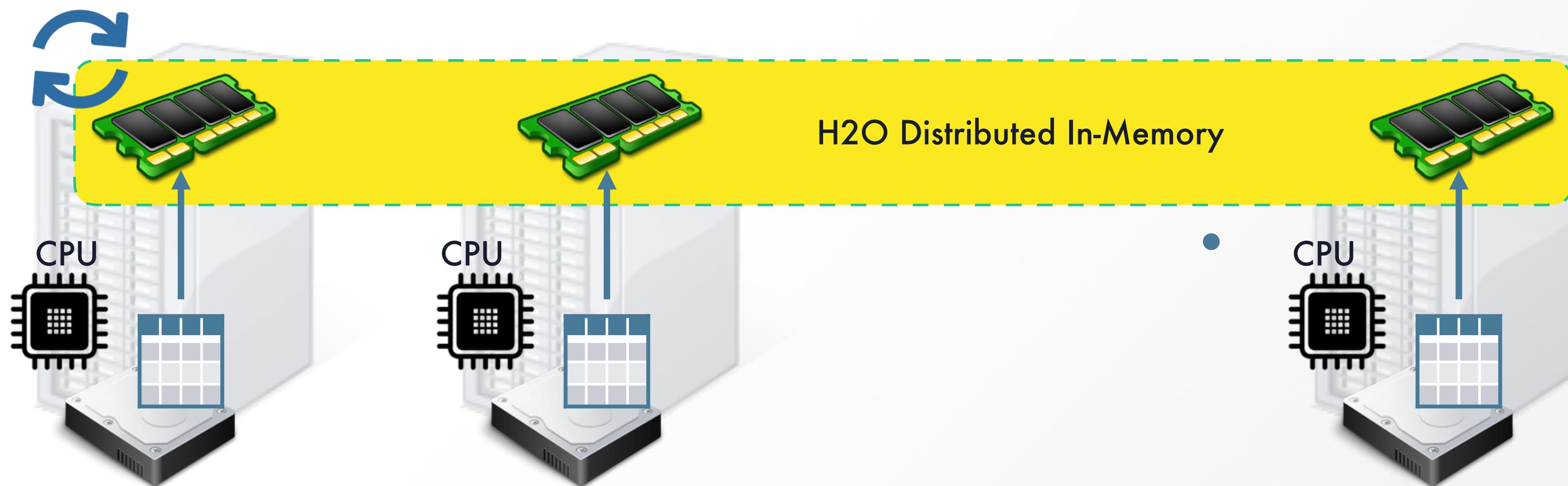


# H2O Core



# H2O Core with Hadoop

Model Building



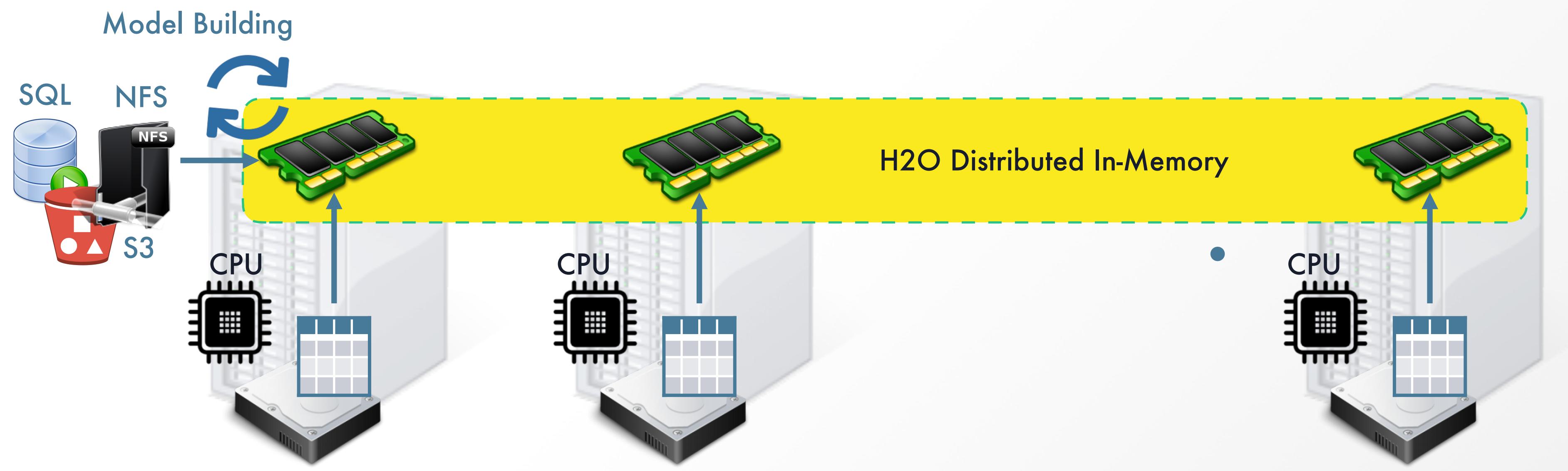
YARN

cloudera Hortonworks

MAPR

H<sub>2</sub>O  
WORLD

# H2O Core with Other Data Sources



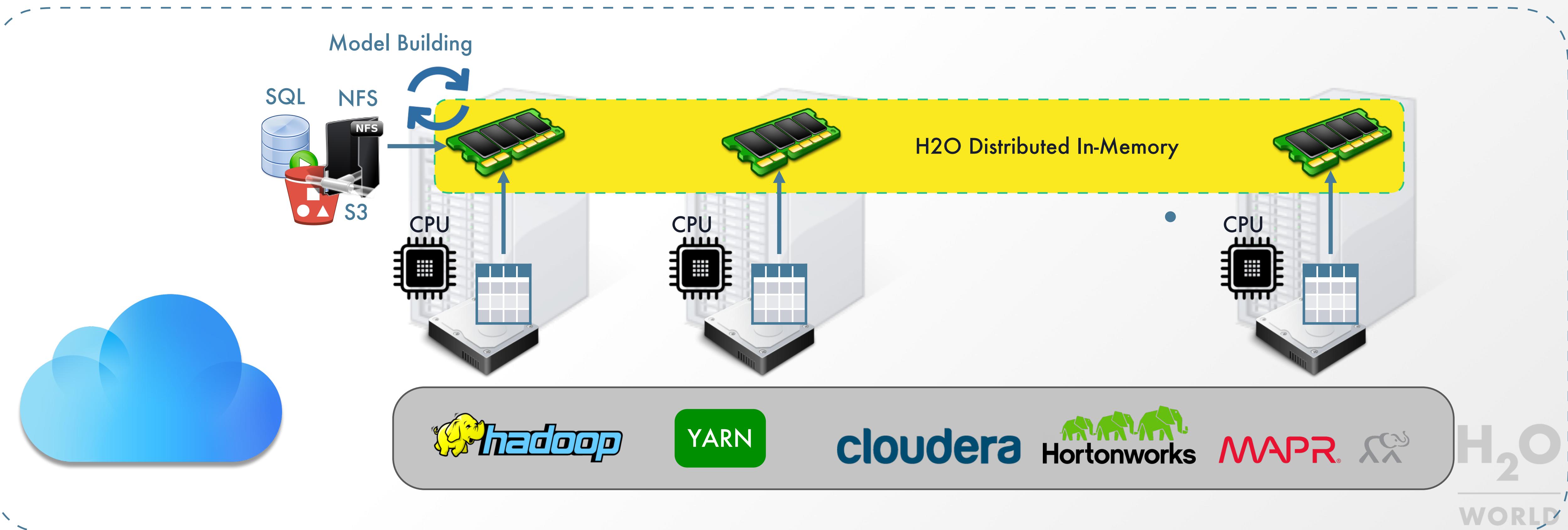
YARN

cloudera Hortonworks

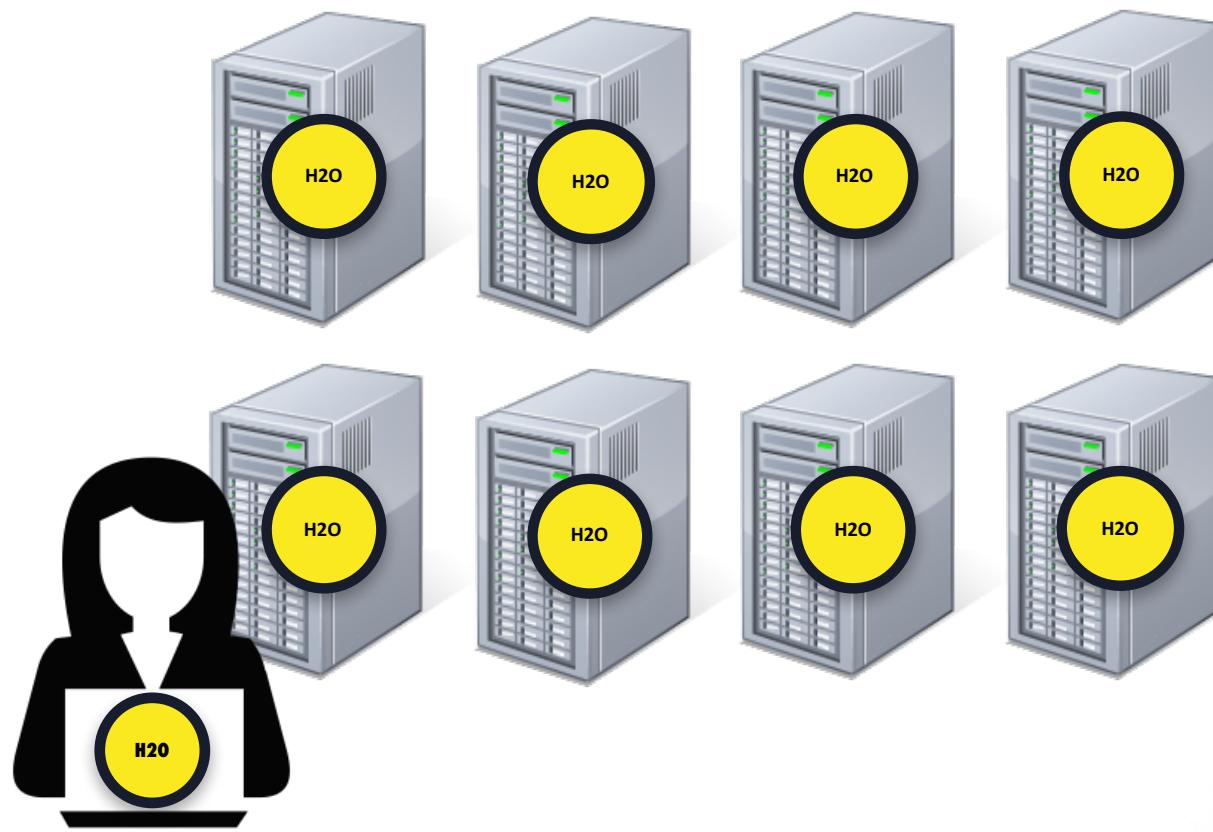
MAPR

H<sub>2</sub>O  
WORLD

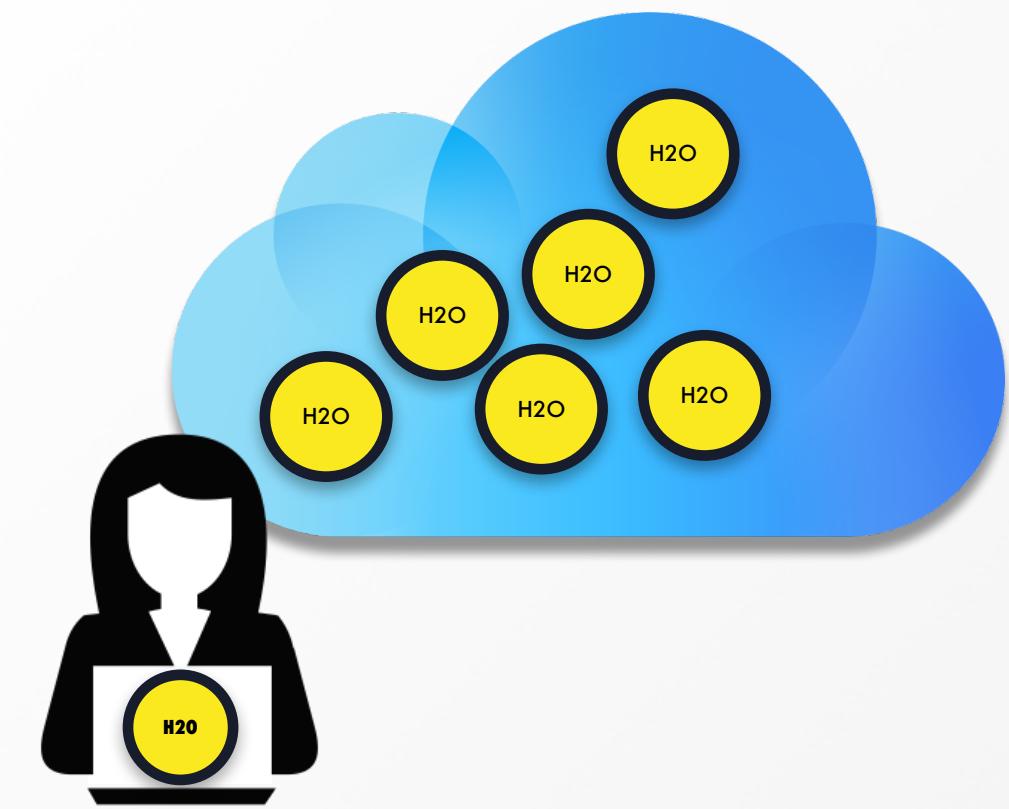
# H2O Core on the Cloud



# H2O Distributed Environments



distributed across multiple machines



distributed on the cloud

# The H2O Python/R API

*(how the client & cluster communicate)*



# H2O Python/R API



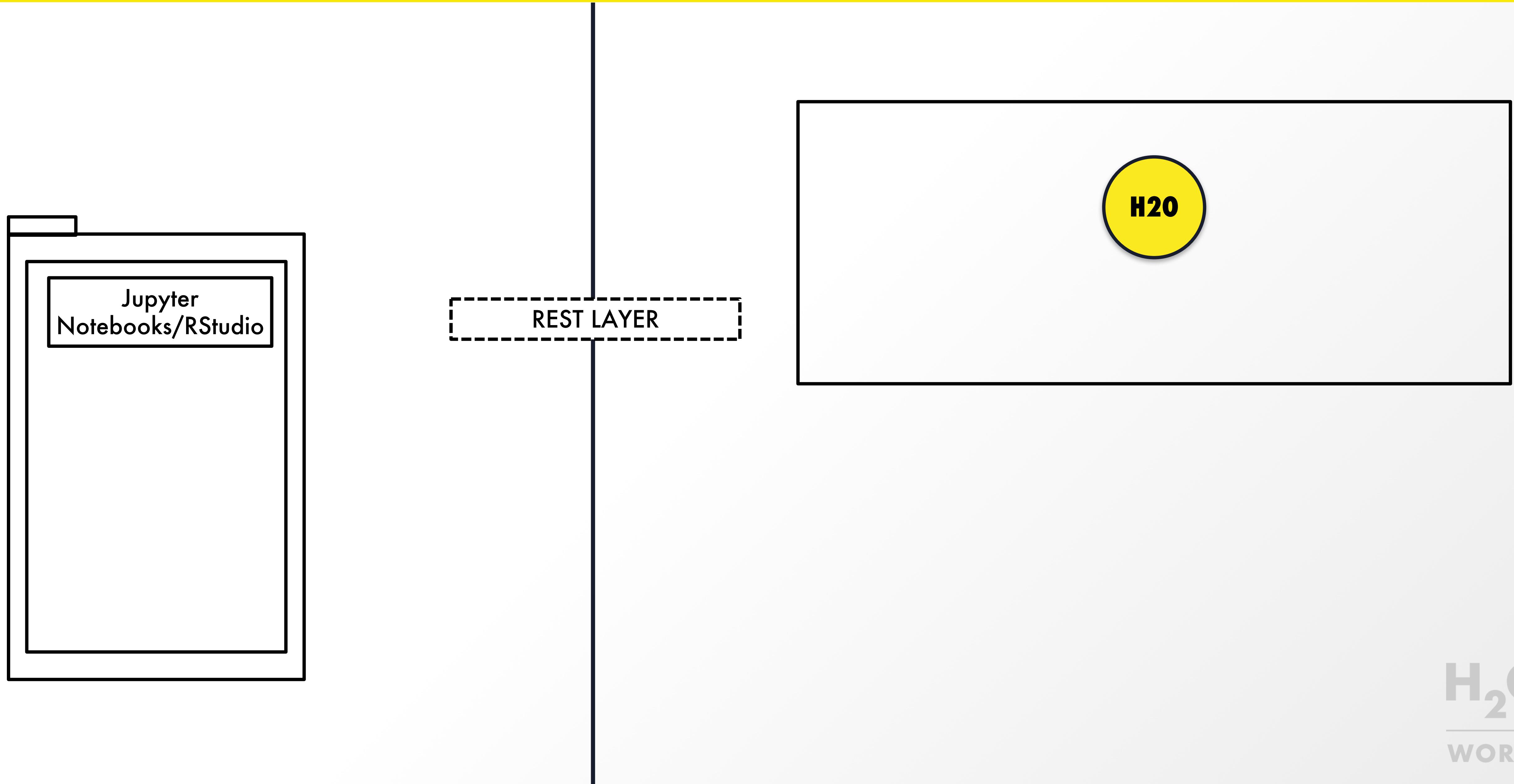
(ALL GREAT THINGS)

# H2O Python/R API

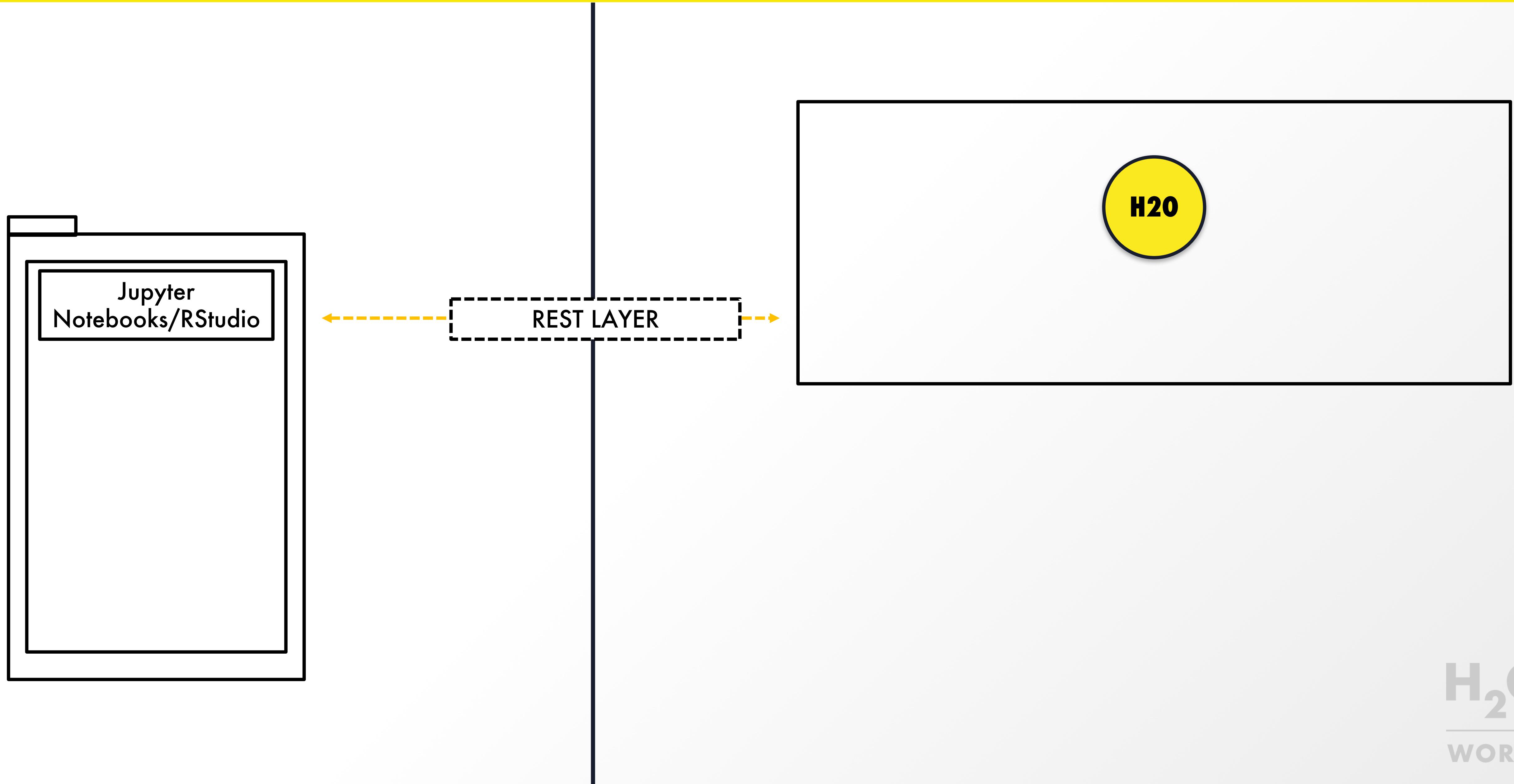


**parallelism & distribution of work** ✓

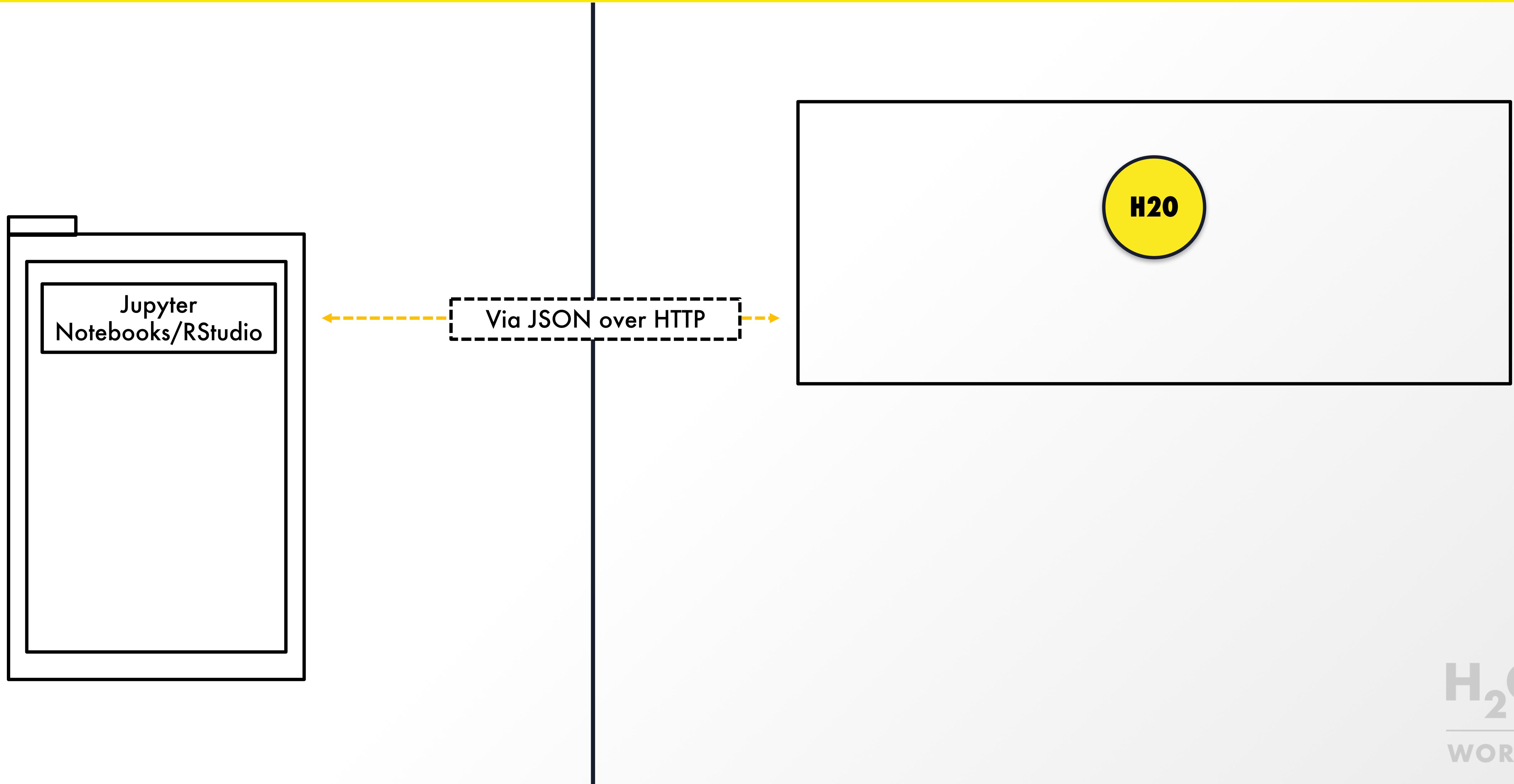
# H2O Python/R API



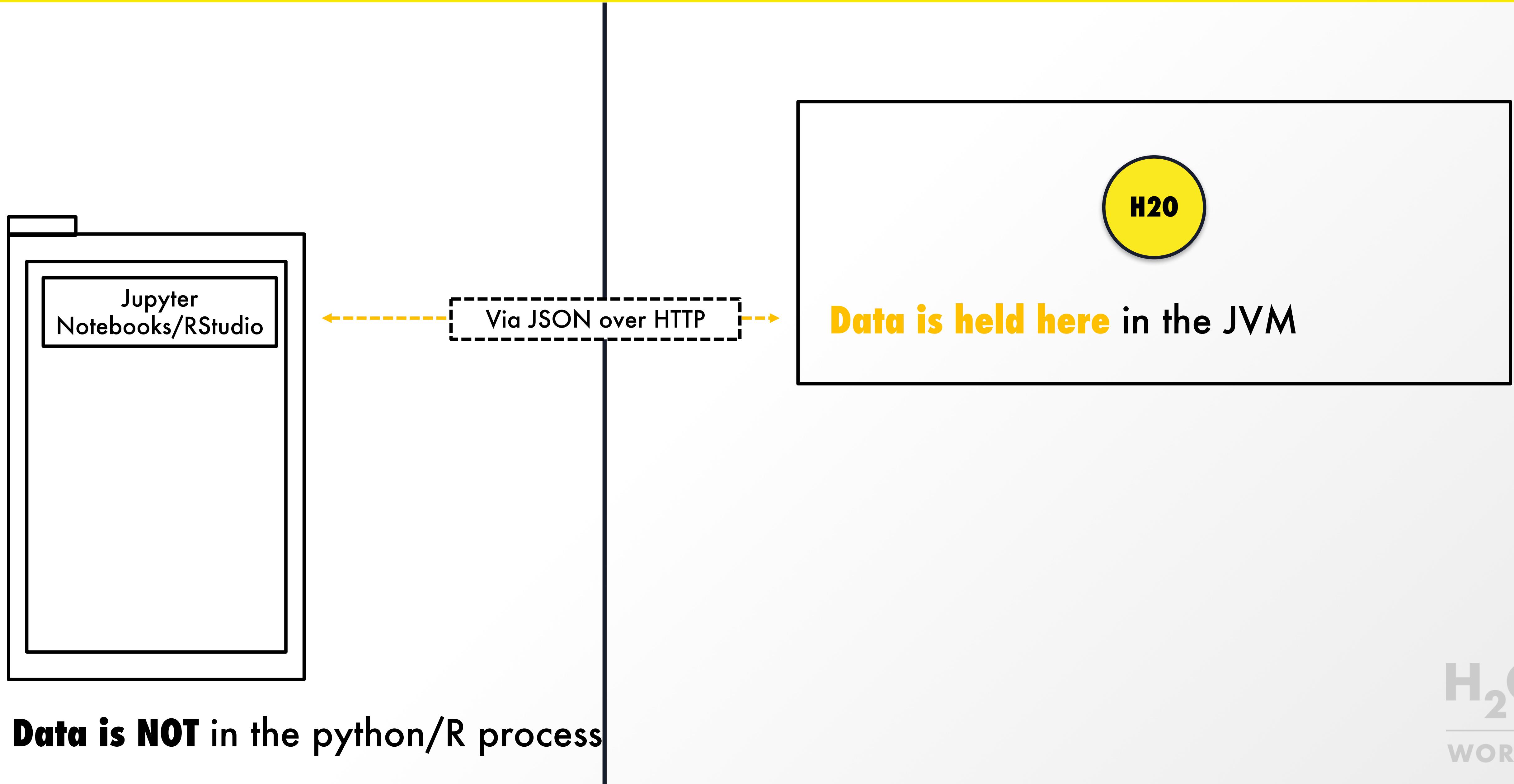
# H2O Python/R API



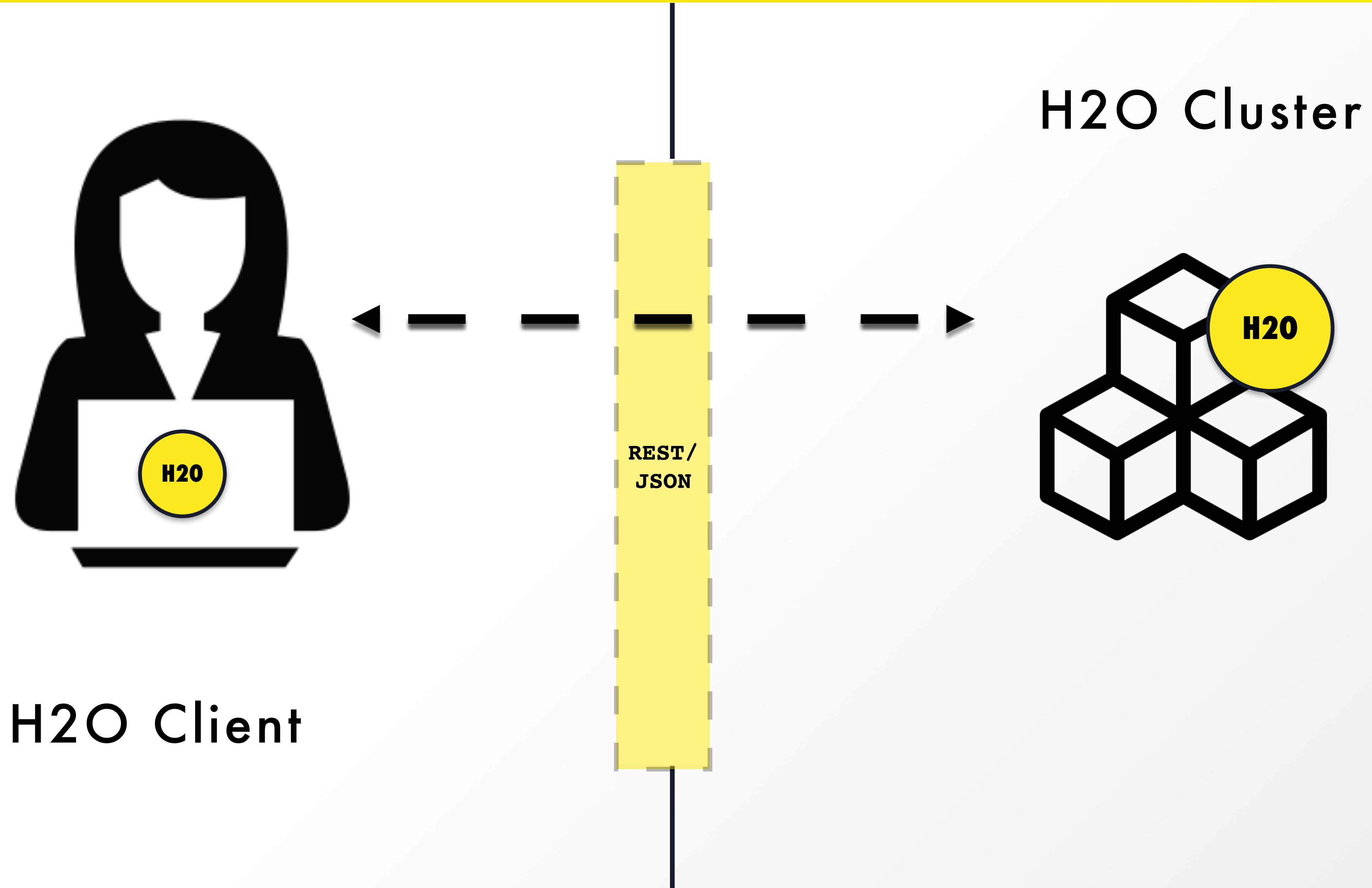
# H2O Python/R API



# H2O Python/R API



# Client & Cluster Communication



# Communication Layers: Interface

Standard Python Interface



the Client

## Jupyter Notebook Interface

localhost:8888/notebooks/Untitled.ipynb?kernel\_name=python3

jupyter Untitled Last Checkpoint: a few seconds ago (unsaved changes)

In [1]: `import h2o  
h2o.init()`

Checking whether there is an H2O instance running at <http://localhost:54321>....

H2O cluster uptime:	54 secs
H2O cluster version:	3.16.0.2
H2O cluster version age:	21 days, 21 hours and 30 minutes
H2O cluster name:	H2O_from_python_laurend_53px3f
H2O cluster total nodes:	1
H2O cluster free memory:	3.541 Gb
H2O cluster total cores:	4

## Terminal Interface

1. Shell

03:40:45 ~ python

Python 2.7.11 (default, Jan 22 2016, 08:29:18)  
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> import h2o

>>> h2o.init()

Checking whether there is an H2O instance running at http://localhost:54321....

Attempting to start a local H2O server...

Java Version: java version "1.8.0\_73"; Java(TM) SE Runtime Environment (build 1.8.0\_73-b02, mixed mode)

Starting server from /usr/local/lib/python2.7/site-packages/h2o/backend/bin/h2o

Ice root: /var/folders/k/\_kpp1czqs3957vq2pr5qngck00000gn/T/tmp0DqYLb

JVM stdout: /var/folders/k/\_kpp1czqs3957vq2pr5qngck00000gn/T/tmp0DqYLb/h2o\_laur

JVM stderr: /var/folders/k/\_kpp1czqs3957vq2pr5qngck00000gn/T/tmp0DqYLb/h2o\_laur

Server is running at http://127.0.0.1:54321

Connecting to H2O server at http://127.0.0.1:54321... successful.

-----

H2O cluster uptime: 03 secs

H2O cluster version: 3.16.0.2

H2O cluster version age: 21 days, 21 hours and 29 minutes

H2O cluster name: H2O\_from\_python\_laurend\_53px3f

H2O cluster total nodes: 1

H2O cluster free memory: 3.556 Gb

H2O cluster total cores: 4

H2O cluster allowed cores: 4

H2O cluster status: accepting new members, healthy

H2O connection url: http://127.0.0.1:54321

H2O connection proxy:

H2O internal security: False

H2O API Extensions: XGBoost, Algos, AutoML, Core V3, Core V4

Python version: 2.7.11 final

-----

>>> █

# Communication Layers: Code Script

## Python Notebook using H2O Package

Python Notebook



the Client

The screenshot shows a Python Notebook interface with a toolbar at the top and a code editor below. The code editor has a green border around the code area. The code itself is written in Python and uses the H2O package to load a Boston housing dataset, set predictor and response variables, split the data into training and validation sets, and train a generalized linear estimator. The code is annotated with comments explaining each step.

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

# Communication Layers: A Command

## Importing Big Data with Python Code

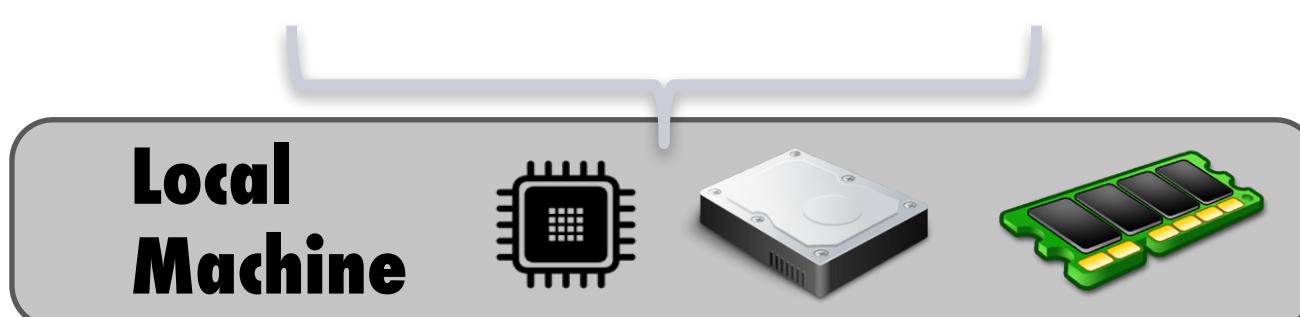
### Python Commands



the Client

In [ ]: `import h2o  
from h2o.estimators.glm import H2OGeneralizedLinearEstimator  
h2o.init()  
  
# import the boston dataset:  
# this dataset looks at features of the boston suburbs and predicts median housing prices  
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing  
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")  
  
# set the predictor names and the response column name  
predictors = boston.columns[:-1]  
# set the response column to "medv", the median value of owner-occupied homes in $1000's  
response = "medv"  
  
# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds  
# within river and 0 otherwise)  
boston['chas'] = boston['chas'].asfactor()  
  
# split into train and validation sets  
train, valid = boston.split_frame(ratios = [.8])  
  
# try using the `alpha` parameter:  
# initialize the estimator then train the model  
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)  
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)  
  
# print the mse for the validation data  
print(boston_glm.mse(valid=True))`

**h2o.import\_file(...)**



# Communication Layers: A Command

## Importing Big Data with Python Code

### Python Commands



the Client

A screenshot of a Jupyter Notebook interface. The title bar says "Python 3". The toolbar includes icons for file operations, cell selection, and cell toolbar. The code cell contains the following Python script:

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

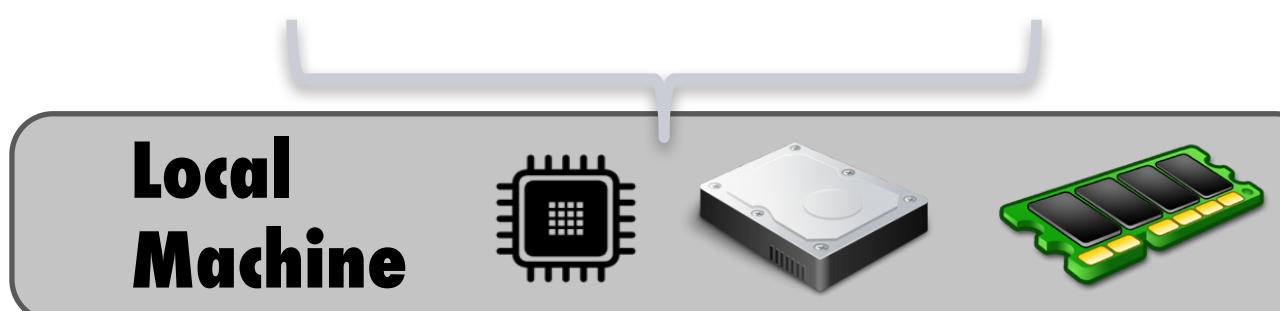
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# crosses river)
boston['chas'] = boston['chas'].asfactor()

# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



Path to Your Dataset

`h2o.import_file(...)`

# Fourth: Communicate



`h2o.import_file(...)`  
requests file import

```
In [ ]: import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

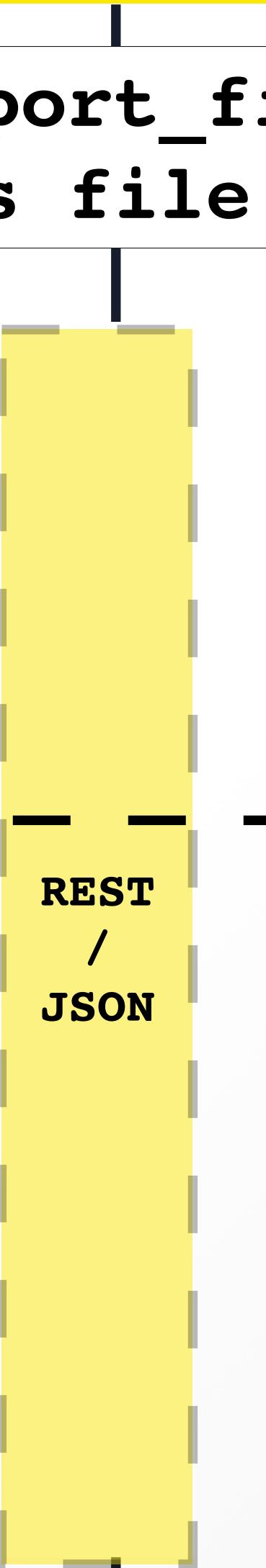
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
boston['chas'] = boston['chas'].asfactor()

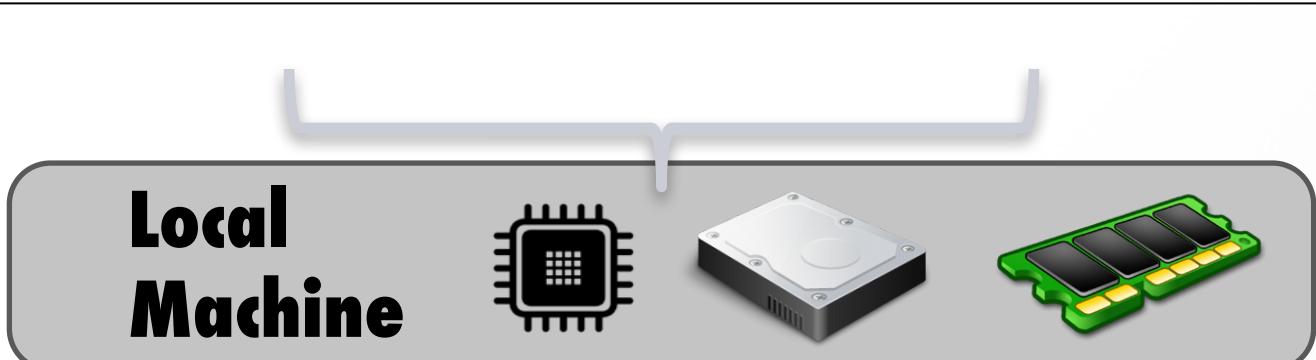
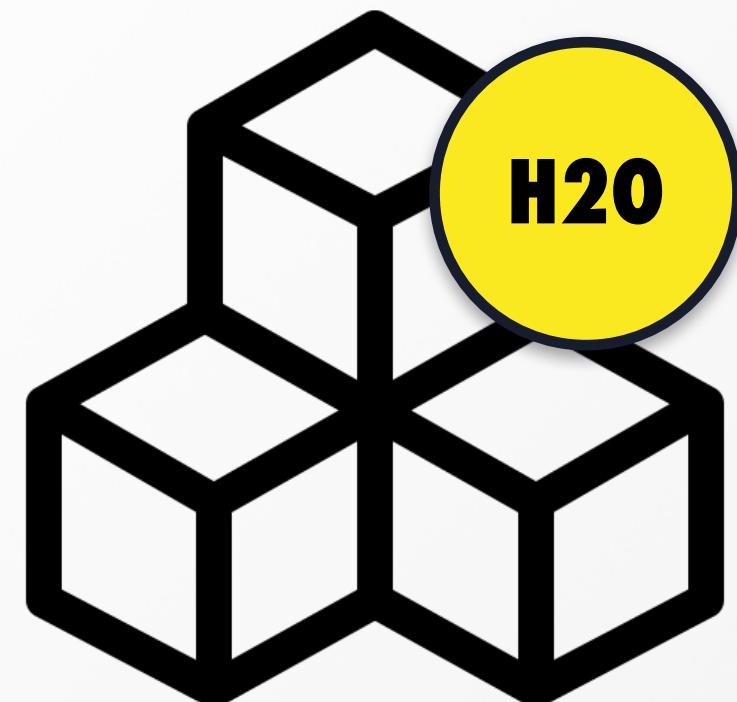
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```



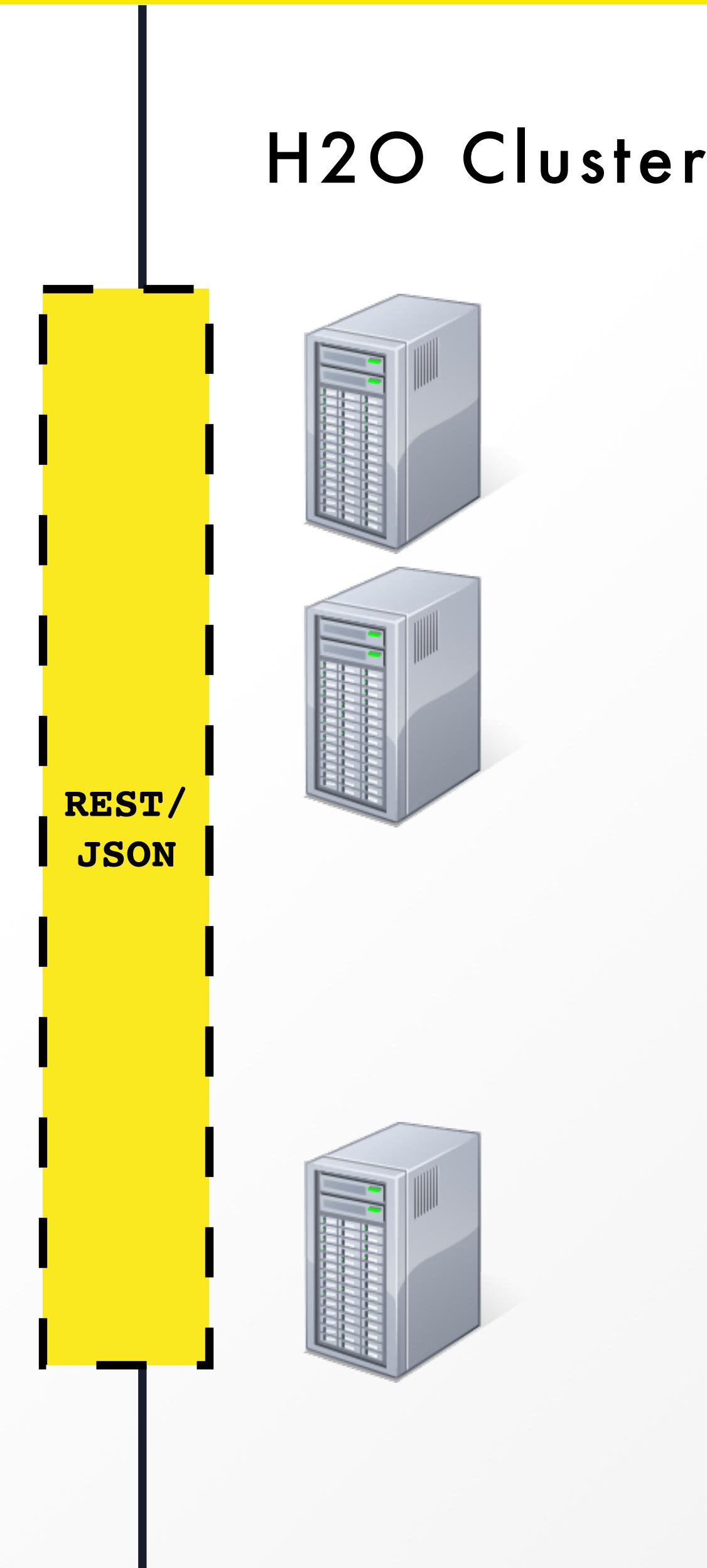
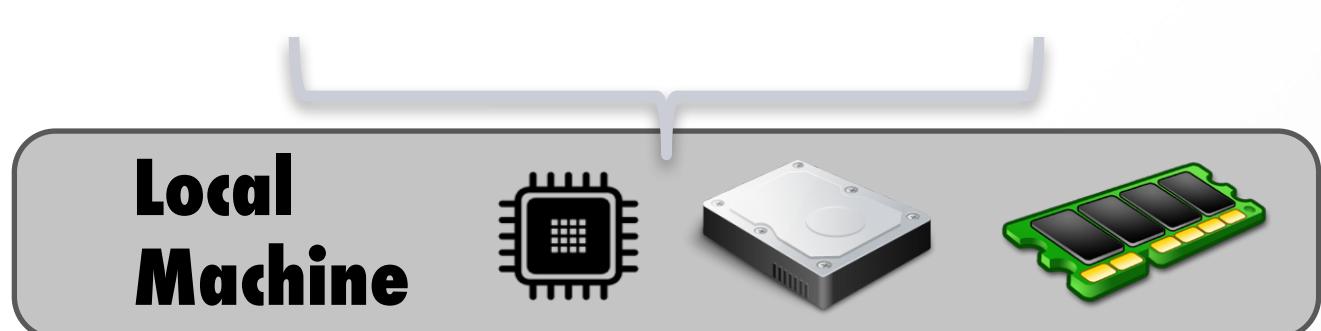
H2O Cluster



# Fifth: Cluster Does Heavy Lifting



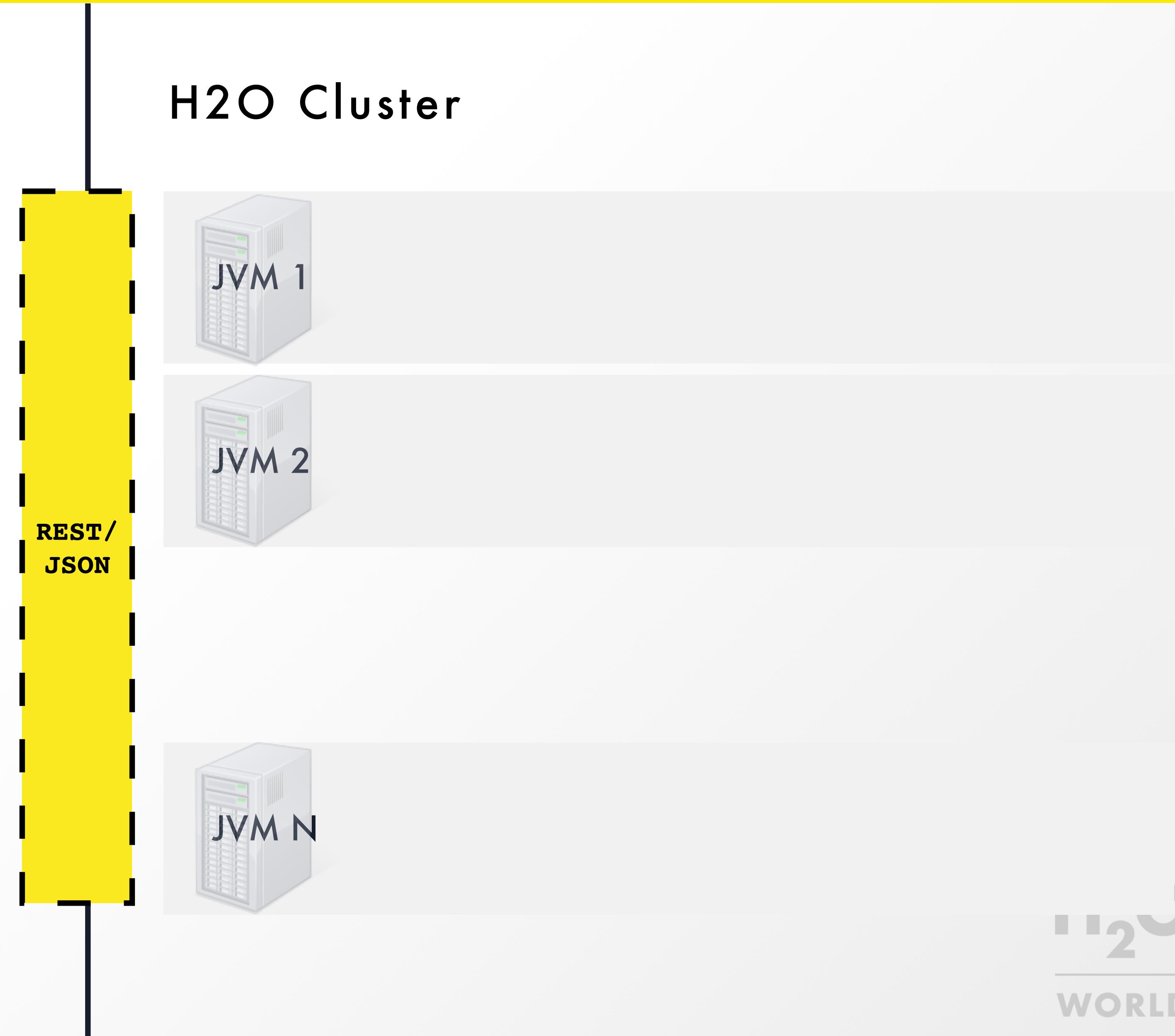
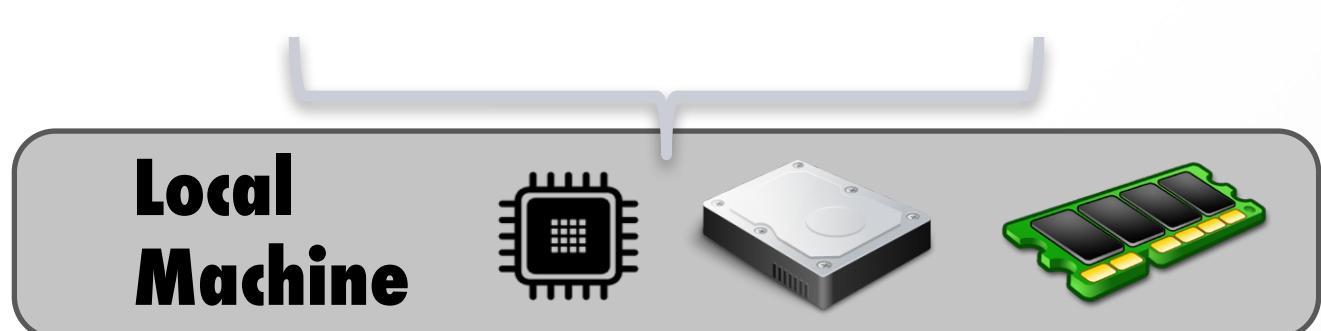
```
File Edit View Insert Cell Kernel Help Python 3 O  
Cell Toolbar  
  
In [ ]: import h2o  
from h2o.estimators.glm import H2OGeneralizedLinearEstimator  
h2o.init()  
  
# import the boston dataset:  
# this dataset looks at features of the boston suburbs and predicts median housing prices  
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing  
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")  
  
# set the predictor names and the response column name  
predictors = boston.columns[:-1]  
# set the response column to "medv", the median value of owner-occupied homes in $1000's  
response = "medv"  
  
# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds  
boston['chas'] = boston['chas'].asfactor()  
  
# split into train and validation sets  
train, valid = boston.split_frame(ratios = [.8])  
  
# try using the `alpha` parameter:  
# initialize the estimator then train the model  
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)  
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)  
  
# print the mse for the validation data  
print(boston_glm.mse(valid=True))
```



# Fifth: Cluster Does Heavy Lifting



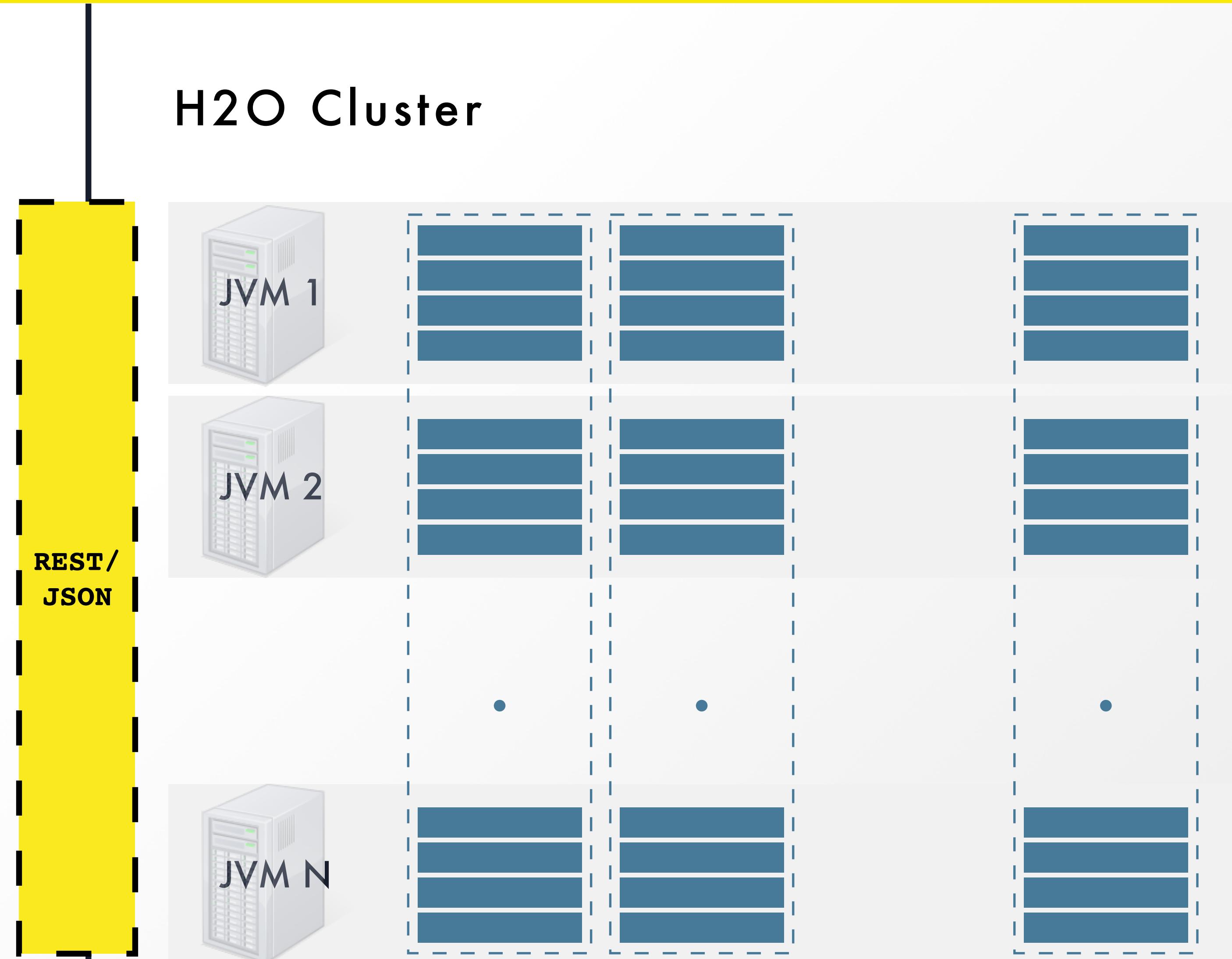
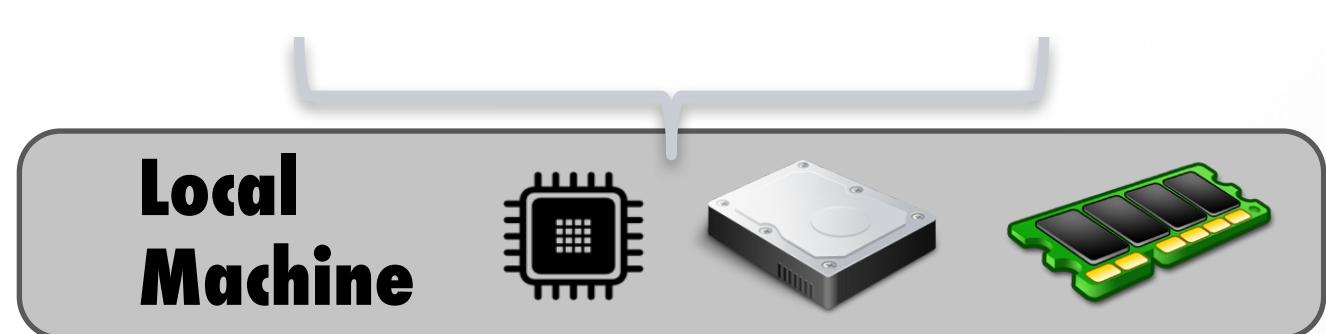
```
File Edit View Insert Cell Kernel Help Python 3 O  
Cell Toolbar  
In [ ]: import h2o  
from h2o.estimators.glm import H2OGeneralizedLinearEstimator  
h2o.init()  
  
# import the boston dataset:  
# this dataset looks at features of the boston suburbs and predicts median housing prices  
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing  
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/Bos  
  
# set the predictor names and the response column name  
predictors = boston.columns[:-1]  
# set the response column to "medv", the median value of owner-occupied homes in $1000's  
response = "medv"  
  
# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds  
boston['chas'] = boston['chas'].asfactor()  
  
# split into train and validation sets  
train, valid = boston.split_frame(ratios = [.8])  
  
# try using the `alpha` parameter:  
# initialize the estimator then train the model  
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)  
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)  
  
# print the mse for the validation data  
print(boston_glm.mse(valid=True))
```



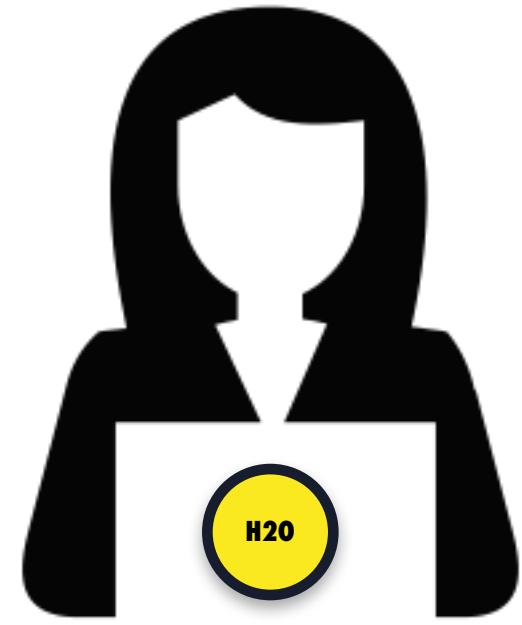
# Fifth: Cluster Does Heavy Lifting



```
File Edit View Insert Cell Kernel Help Python 3 O  
Cell Toolbar  
In [ ]: import h2o  
from h2o.estimators.glm import H2OGeneralizedLinearEstimator  
h2o.init()  
  
# import the boston dataset:  
# this dataset looks at features of the boston suburbs and predicts median housing prices  
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing  
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/Bos...  
  
# set the predictor names and the response column name  
predictors = boston.columns[:-1]  
# set the response column to "medv", the median value of owner-occupied homes in $1000's  
response = "medv"  
  
# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds  
boston['chas'] = boston['chas'].asfactor()  
  
# split into train and validation sets  
train, valid = boston.split_frame(ratios = [.8])  
  
# try using the `alpha` parameter:  
# initialize the estimator then train the model  
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)  
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)  
  
# print the mse for the validation data  
print(boston_glm.mse(valid=True))
```



# Fifth: Cluster Does Heavy Lifting



In [ ]:

```
import h2o
from h2o.estimators.glm import H2OGeneralizedLinearEstimator
h2o.init()

# import the boston dataset:
# this dataset looks at features of the boston suburbs and predicts median housing prices
# the original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Housing
boston = h2o.import_file("https://s3.amazonaws.com/h2o-public-test-data/smalldata/gbm_test/BostonHousing.csv")

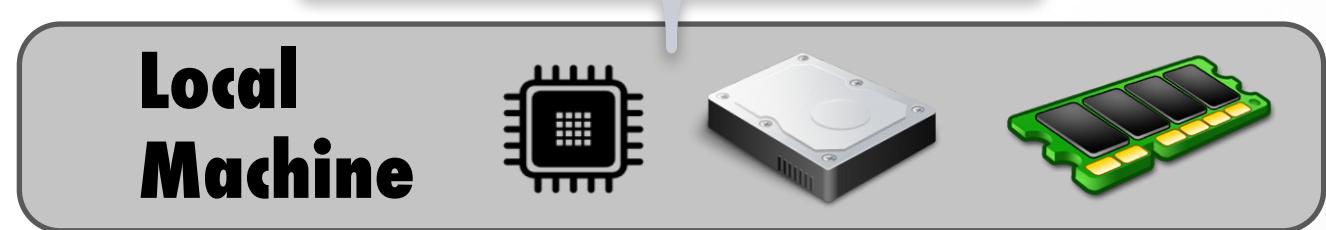
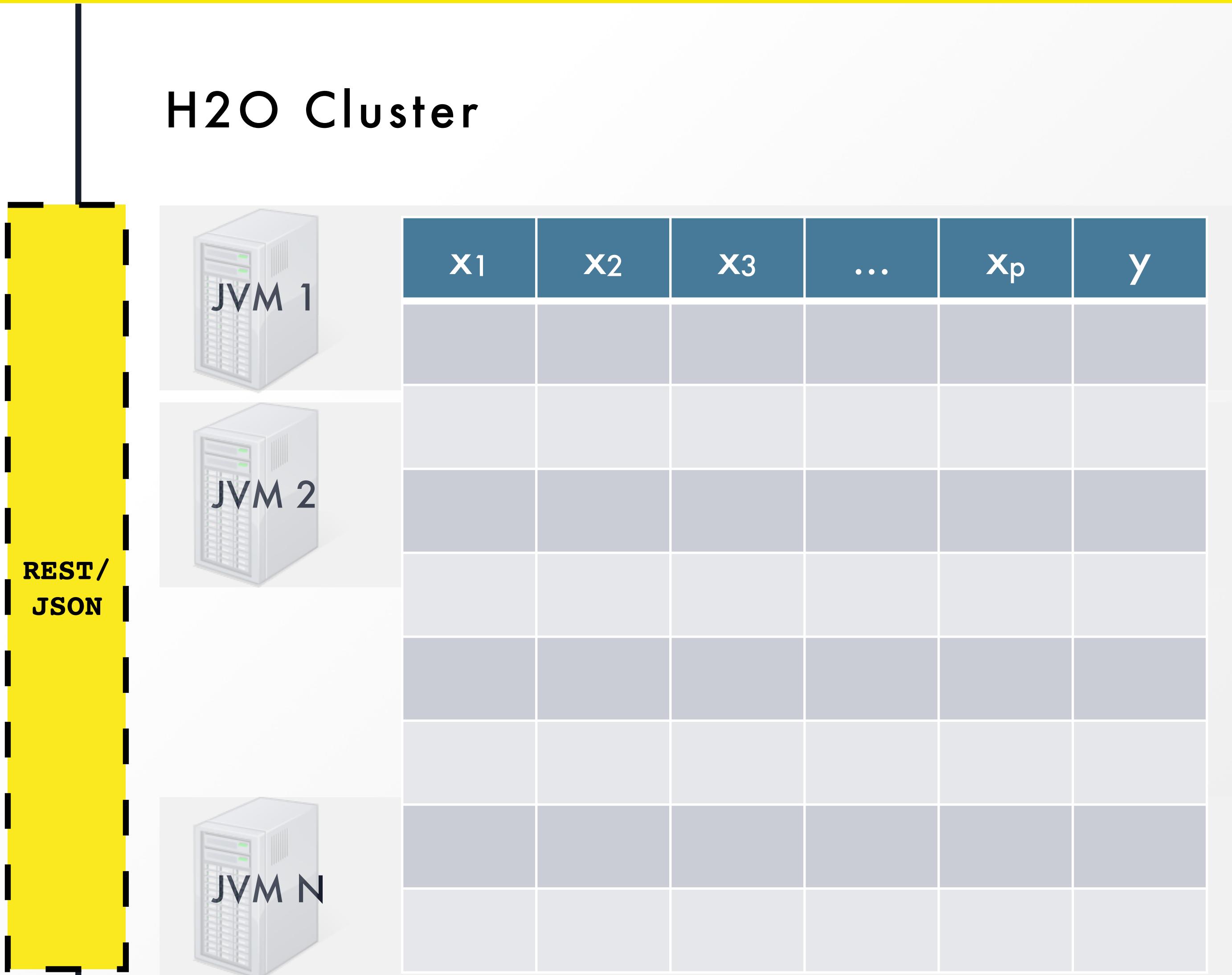
# set the predictor names and the response column name
predictors = boston.columns[:-1]
# set the response column to "medv", the median value of owner-occupied homes in $1000's
response = "medv"

# convert the chas column to a factor (chas = Charles River dummy variable (= 1 if tract bounds
# intersected Charles River))
boston['chas'] = boston['chas'].asfactor()

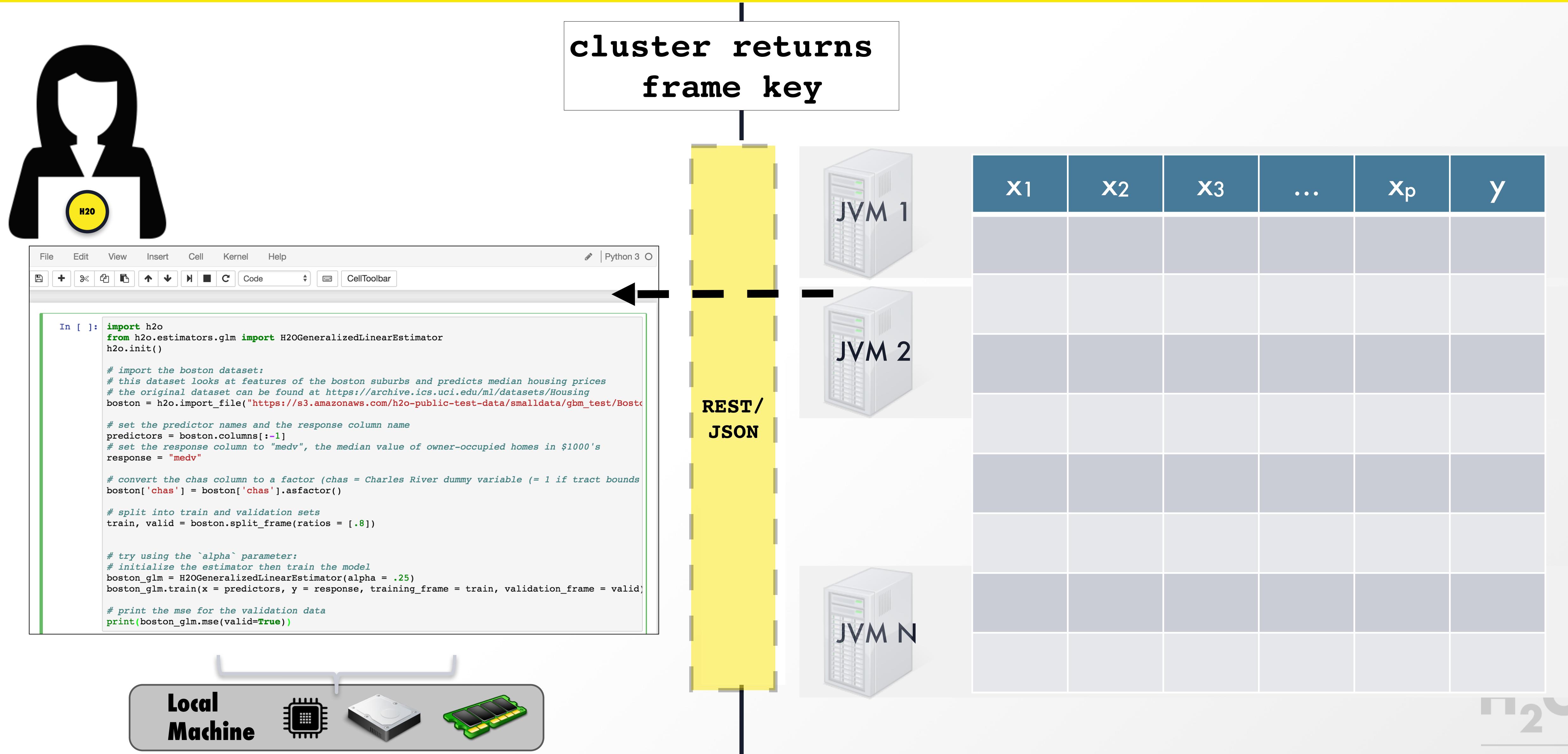
# split into train and validation sets
train, valid = boston.split_frame(ratios = [.8])

# try using the `alpha` parameter:
# initialize the estimator then train the model
boston_glm = H2OGeneralizedLinearEstimator(alpha = .25)
boston_glm.train(x = predictors, y = response, training_frame = train, validation_frame = valid)

# print the mse for the validation data
print(boston_glm.mse(valid=True))
```

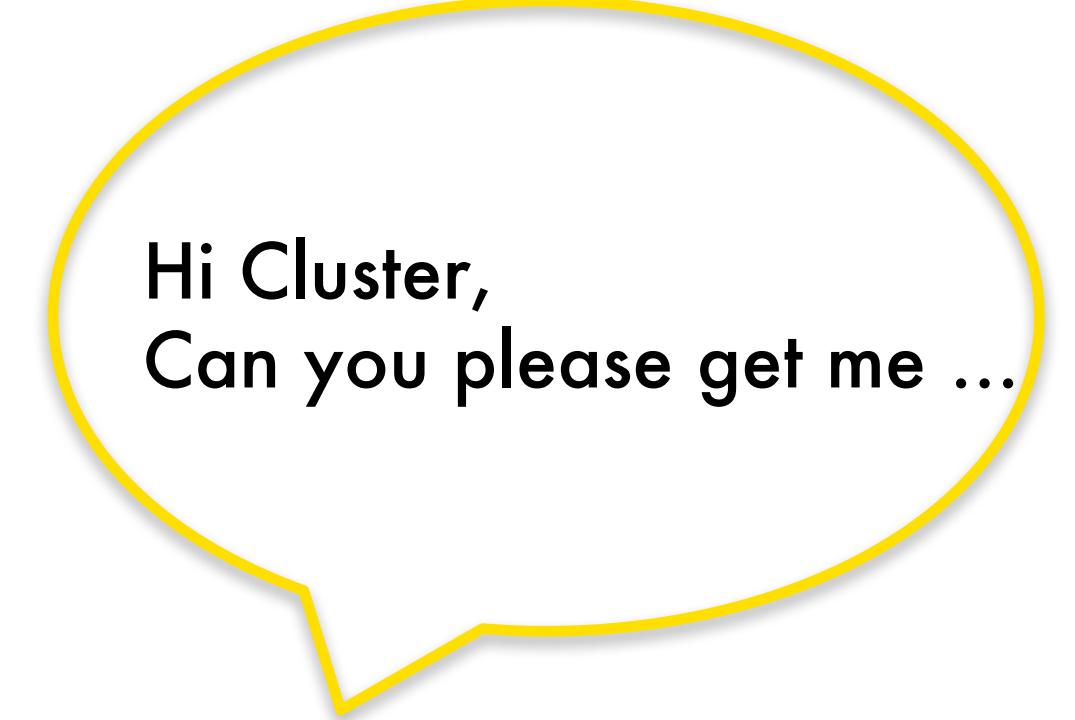


# H2O client



# What does the Client Do?

Clients Only Tells the Cluster What to Do

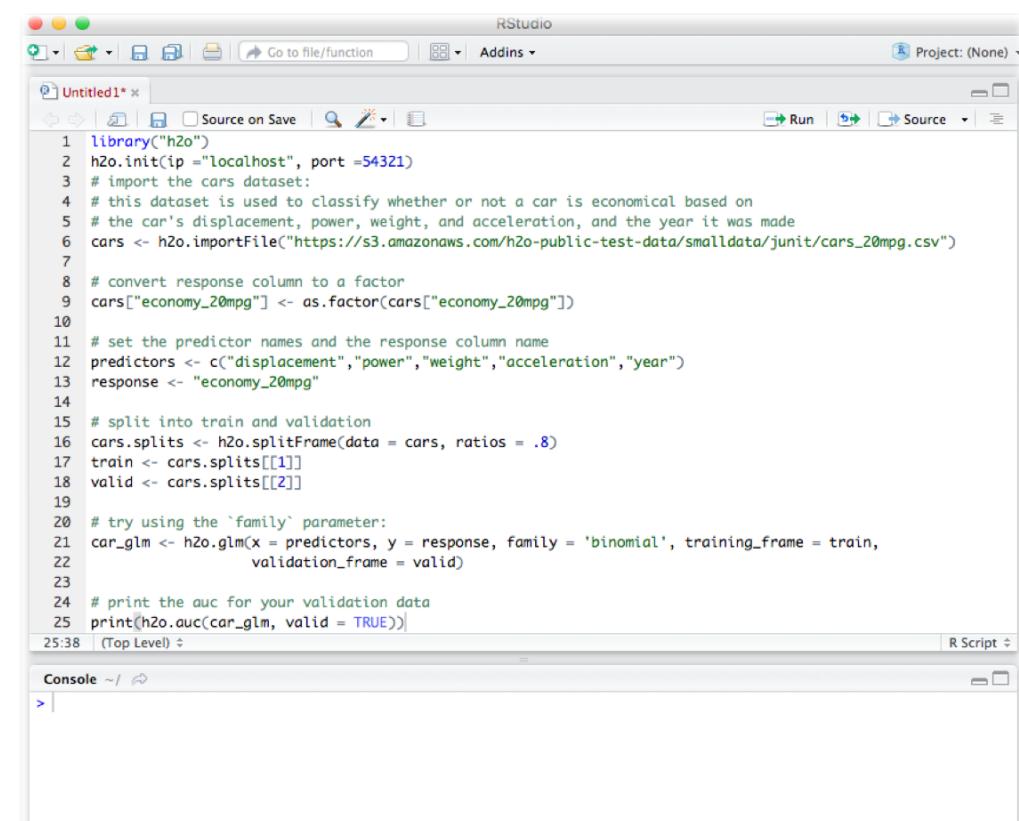


```
library("h2o")
h2o.init(ip = "localhost", port = 54321)
# import the cars dataset:
# this dataset is used to classify whether or not a car is economical based on
# the car's displacement, power, weight, and acceleration, and the year it was made
cars <- h2o.importFile("https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv")
# convert response column to a factor
cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
# set the predictor names and the response column name
predictors <- c("displacement","power","weight","acceleration","year")
response <- "economy_20mpg"
# split into train and validation
cars.splits <- h2o.splitFrame(data = cars, ratios = .8)
train <- cars.splits[[1]]
valid <- cars.splits[[2]]
# try using the `family` parameter:
cor_glm <- h2o.glm(x = predictors, y = response, family = 'binomial', training_frame = train,
validation_frame = valid)
# print the auc for your validation data
print(h2o.auc(cor_glm, valid = TRUE))
```

Hi Cluster,  
Can you please get me ...

# Client Only Passes Requests

Big Data Never Flows Through The Client  
Unless **Explicitly** Asked



The screenshot shows an RStudio interface with an 'Untitled1' script file open. The code performs the following steps:

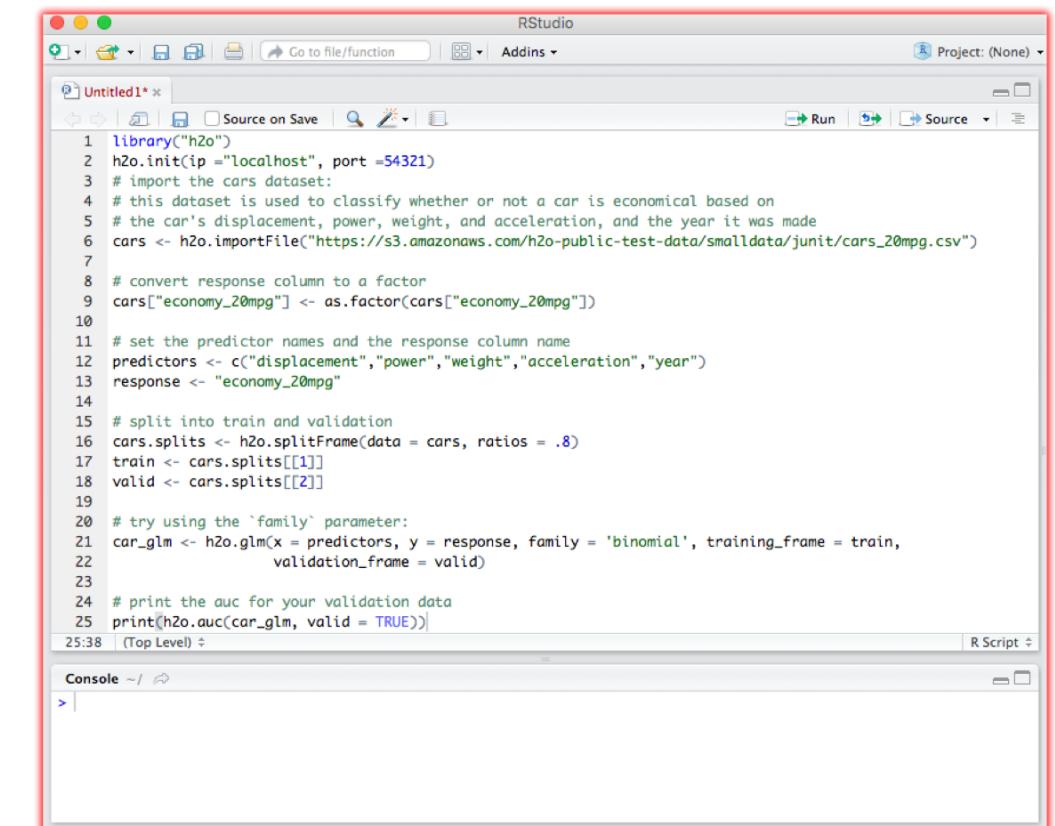
- Imports the 'h2o' library.
- Initializes H2O with port 54321.
- Imports the 'cars' dataset.
- Specifies the dataset is used to classify whether or not a car is economical based on its displacement, power, weight, and acceleration, and the year it was made.
- Imports the 'cars\_20mpg.csv' file from S3.
- Converts the 'economy\_20mpg' column to a factor.
- Sets predictor names and response column name.
- Creates splits for training and validation data.
- Attempts to use the 'family' parameter with 'h2o.glm'.
- Prints the AUC for the validation data.

No, you take care of the  
heavy lifting

# What if?

## Pulling Big Data into Python/R Can Overwhelm Your Session

**my\_big\_dataframe.as\_data\_frame()**

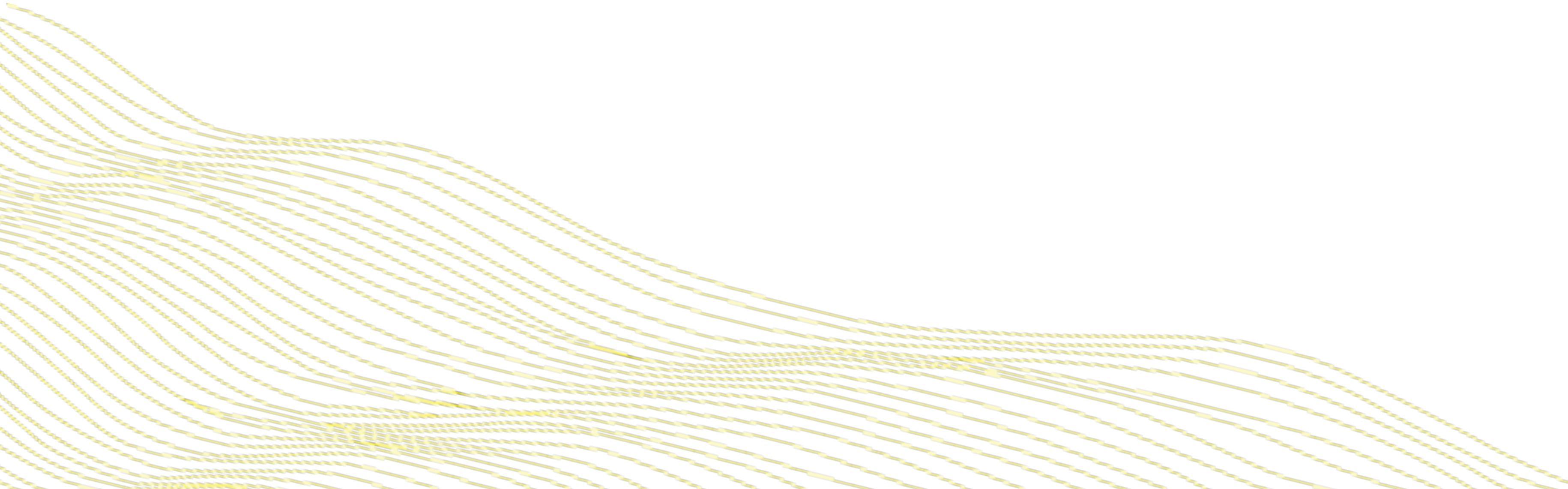


A screenshot of an RStudio interface. The top bar shows 'RStudio' and 'Untitled1\*'. The main area contains a large block of R code. The code starts with `library("h2o")` and ends with `print(h2o.auc(car\_glm, valid = TRUE))`. The code is written in a monospaced font. Below the code editor is a 'Console' window with a single line of text: '> |'. A red speech bubble with the word 'AH!' is overlaid on the right side of the RStudio window.

```
1 library("h2o")
2 h2o.init(ip = "localhost", port = 54321)
3 # import the cars dataset:
4 # this dataset is used to classify whether or not a car is economical based on
5 # the car's displacement, power, weight, and acceleration, and the year it was made
6 cars <- h2o.importFile("https://s3.amazonaws.com/h2o-public-test-data/smalldata/junit/cars_20mpg.csv")
7
8 # convert response column to a factor
9 cars["economy_20mpg"] <- as.factor(cars["economy_20mpg"])
10
11 # set the predictor names and the response column name
12 predictors <- c("displacement","power","weight","acceleration","year")
13 response <- "economy_20mpg"
14
15 # split into train and validation
16 cars.splits <- h2o.splitFrame(data = cars, ratios = .8)
17 trin <- cars.splits[[1]]
18 valid <- cars.splits[[2]]
19
20 # try using the 'family' parameter:
21 car_glm <- h2o.glm(x = predictors, y = response, family = 'binomial', training_frame = train,
22                      validation_frame = valid)
23
24 # print the auc for your validation data
25 print(h2o.auc(car_glm, valid = TRUE))
```

# **Automatic Machine Learning (AutoML)**

*Scalable Automatic Machine Learning*



# AutoML Agenda

- **The Machine Learning Pipeline**
- What is Automatic Machine Learning?
- How to Use H2O's AutoML

# AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning? (**Feature Preprocessing**)
- How to Use H2O's AutoML

# AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning? **(Feature Engineering)**
- How to Use H2O's AutoML

# AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning?**(Hyperparameter Search)**
- How to Use H2O's AutoML

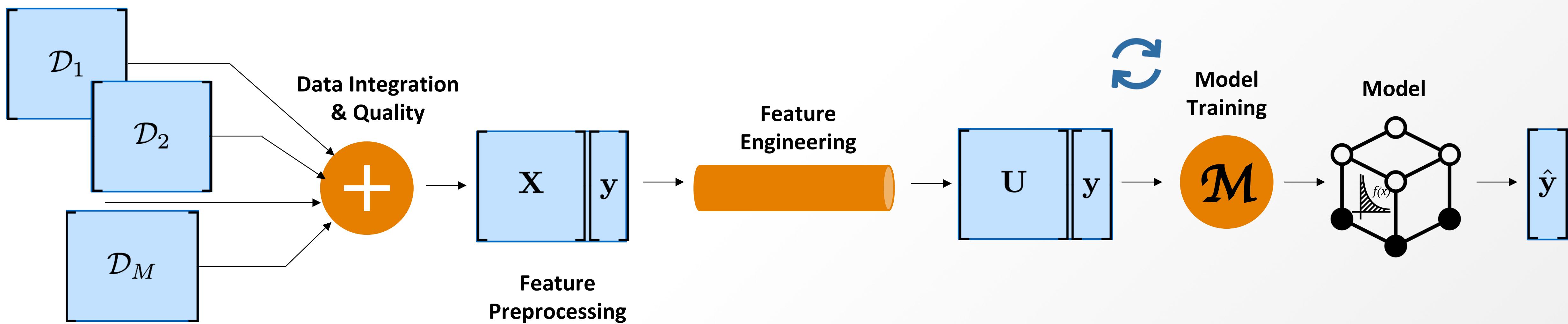
# AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning?**(Stacked Ensembles)**
- How to Use H2O's AutoML

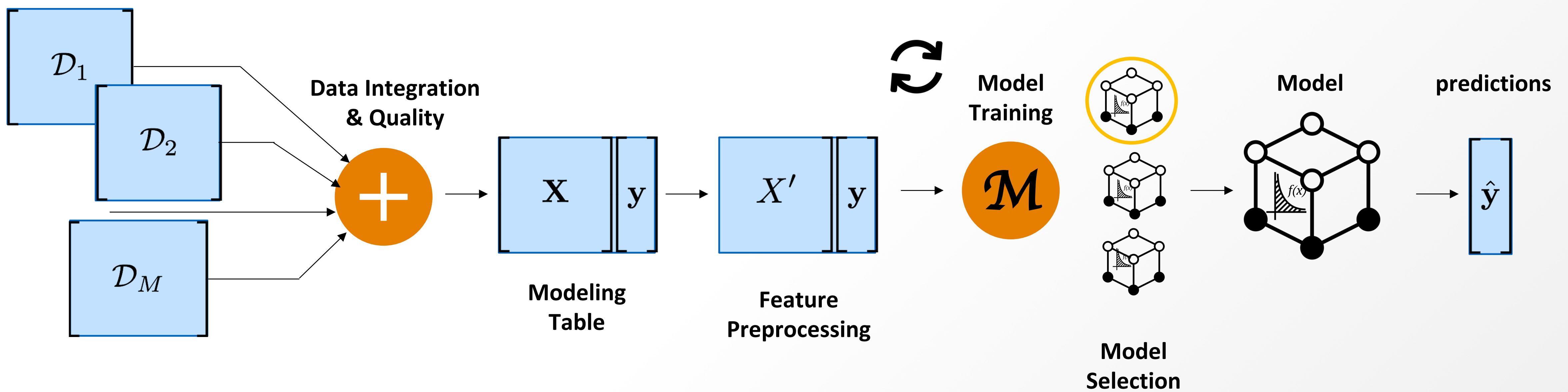
# AutoML Agenda

- The Machine Learning Pipeline
- What is Automatic Machine Learning?
- How to Use H2O's AutoML (more hands-on!)

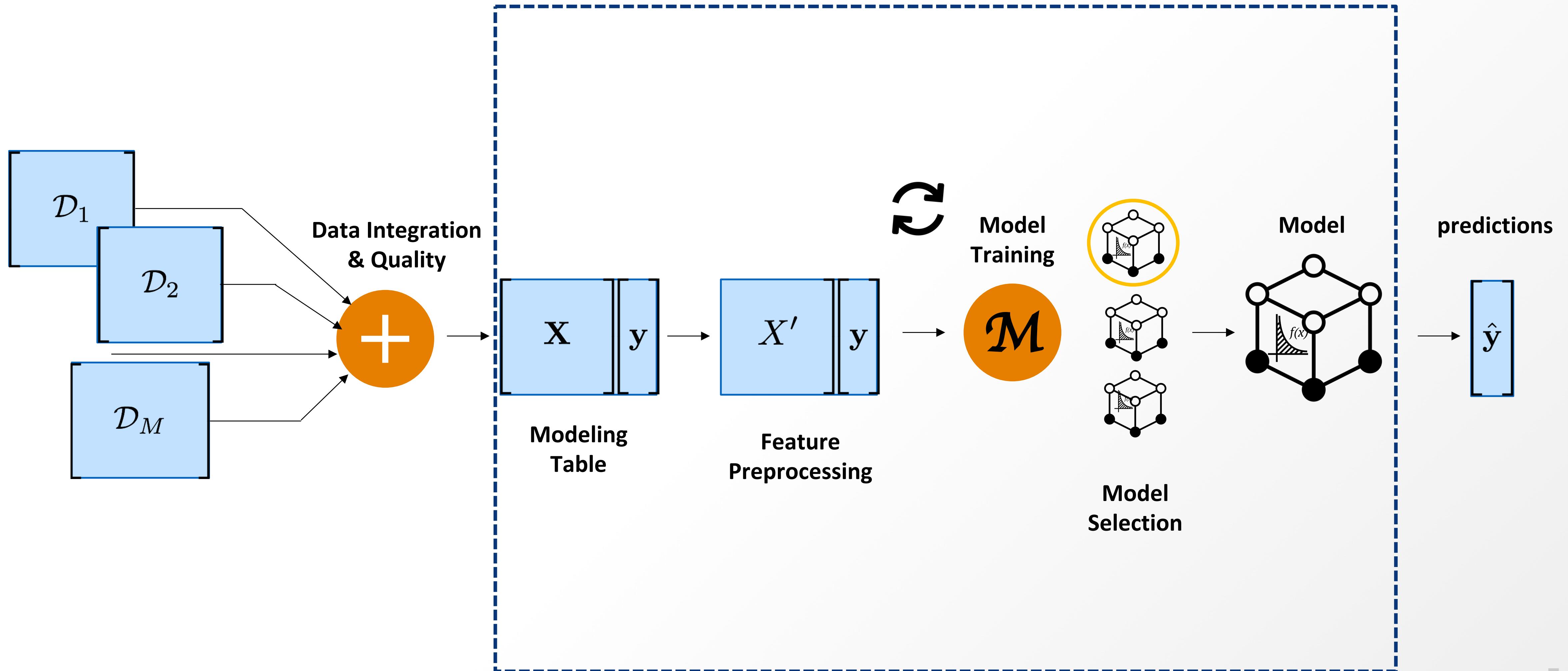
# The Machine Learning Pipeline



# Updated Machine Learning Pipeline



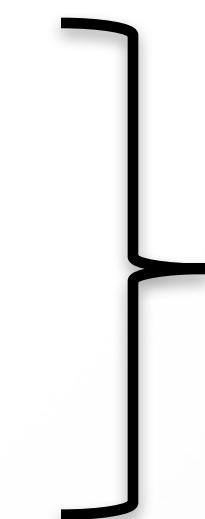
# Where Automatic Machine Learning Fits



# Automatic ML Overview

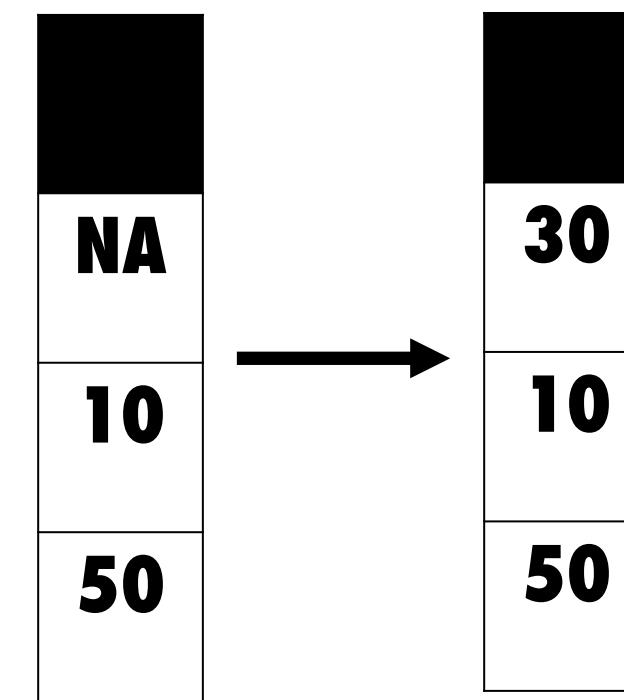
- **Feature Preprocessing**
- **Feature Engineering**
- Hyperparameter Search
- Ensembles

# Automatic ML Overview

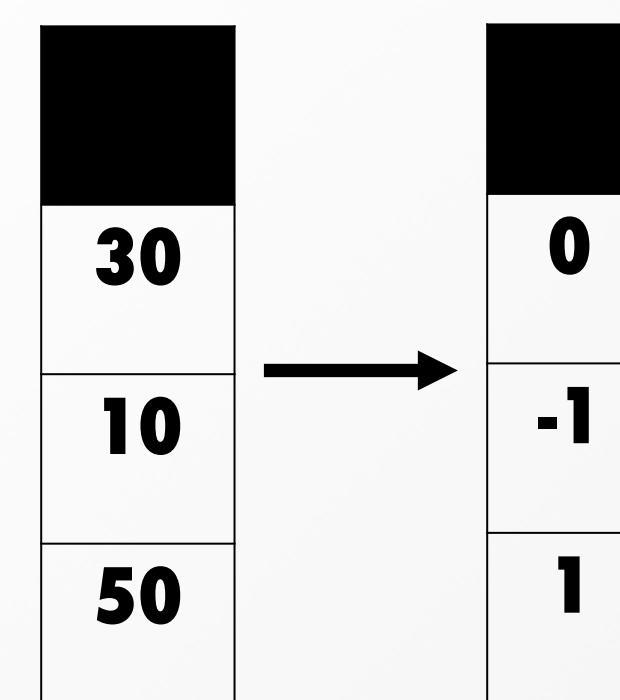
- Feature Preprocessing
  - Feature Engineering
  - Hyperparameter Search
  - Ensembles
- 
- Distinction  
Sometimes  
Fuzzy

# Feature Preprocessing

## Missing Values

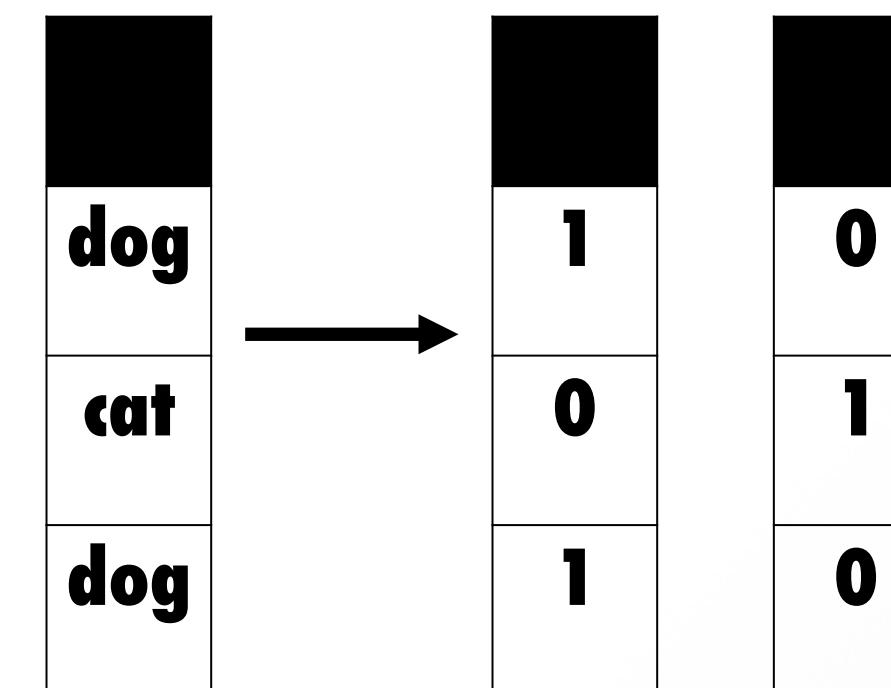


## Standardization

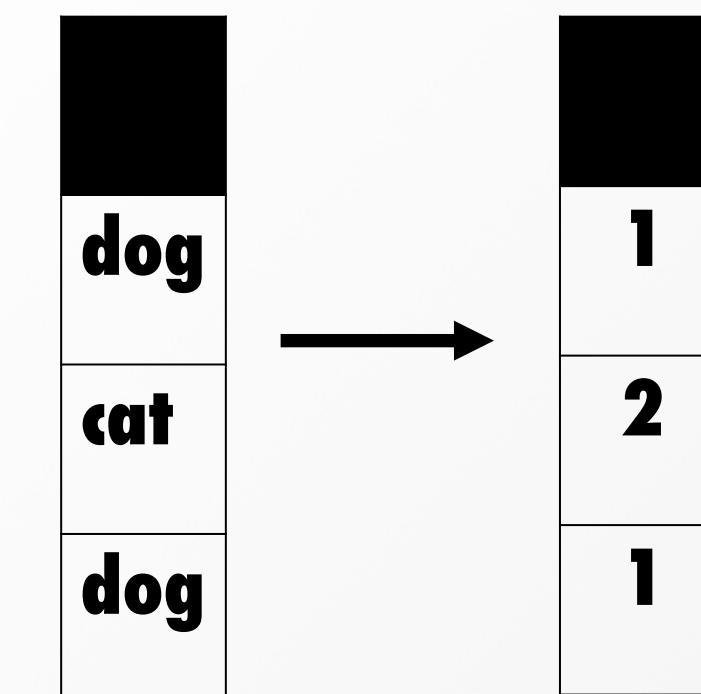


# Feature Preprocessing

## Handling Categorical Columns

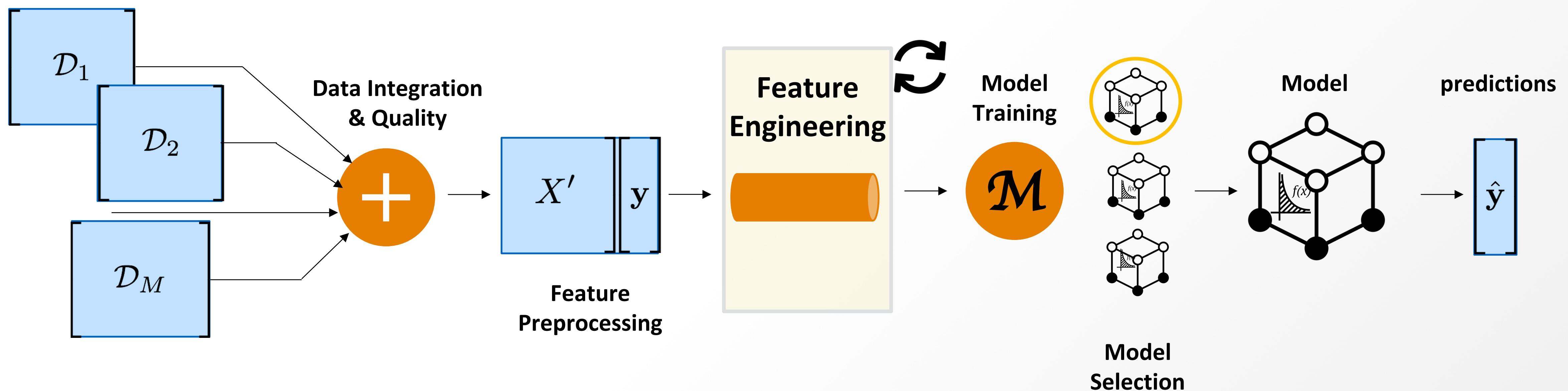


One-hot Encoding



Label Encoder

# Adding in Feature Engineering



# Feature Engineering



Domain Expertise

Timestamp
jan/02/2017
march/03/2019
dec/11/2018



Month	Day	Year
01	02	2017
03	03	2019
12	11	2018

Date Expansion

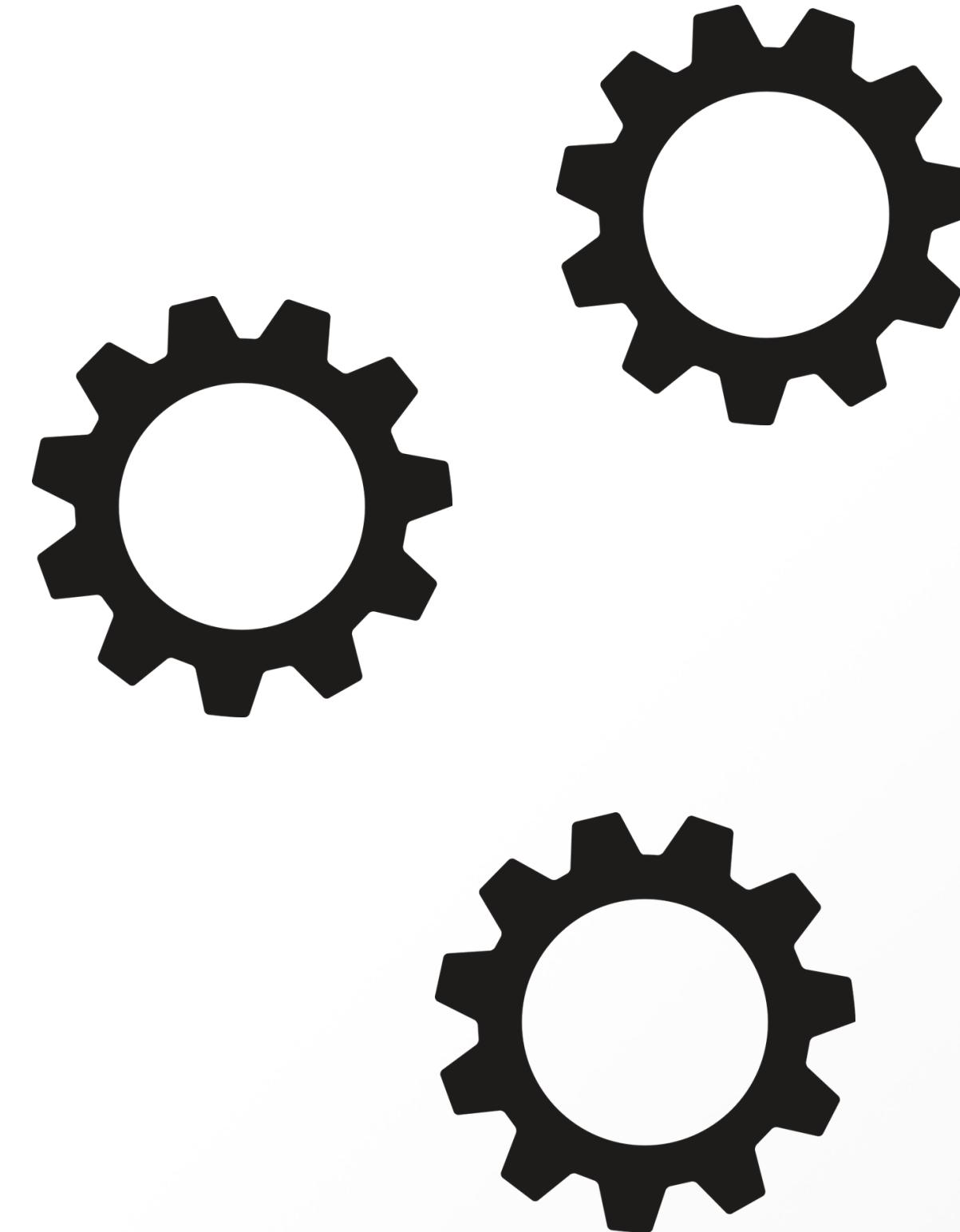
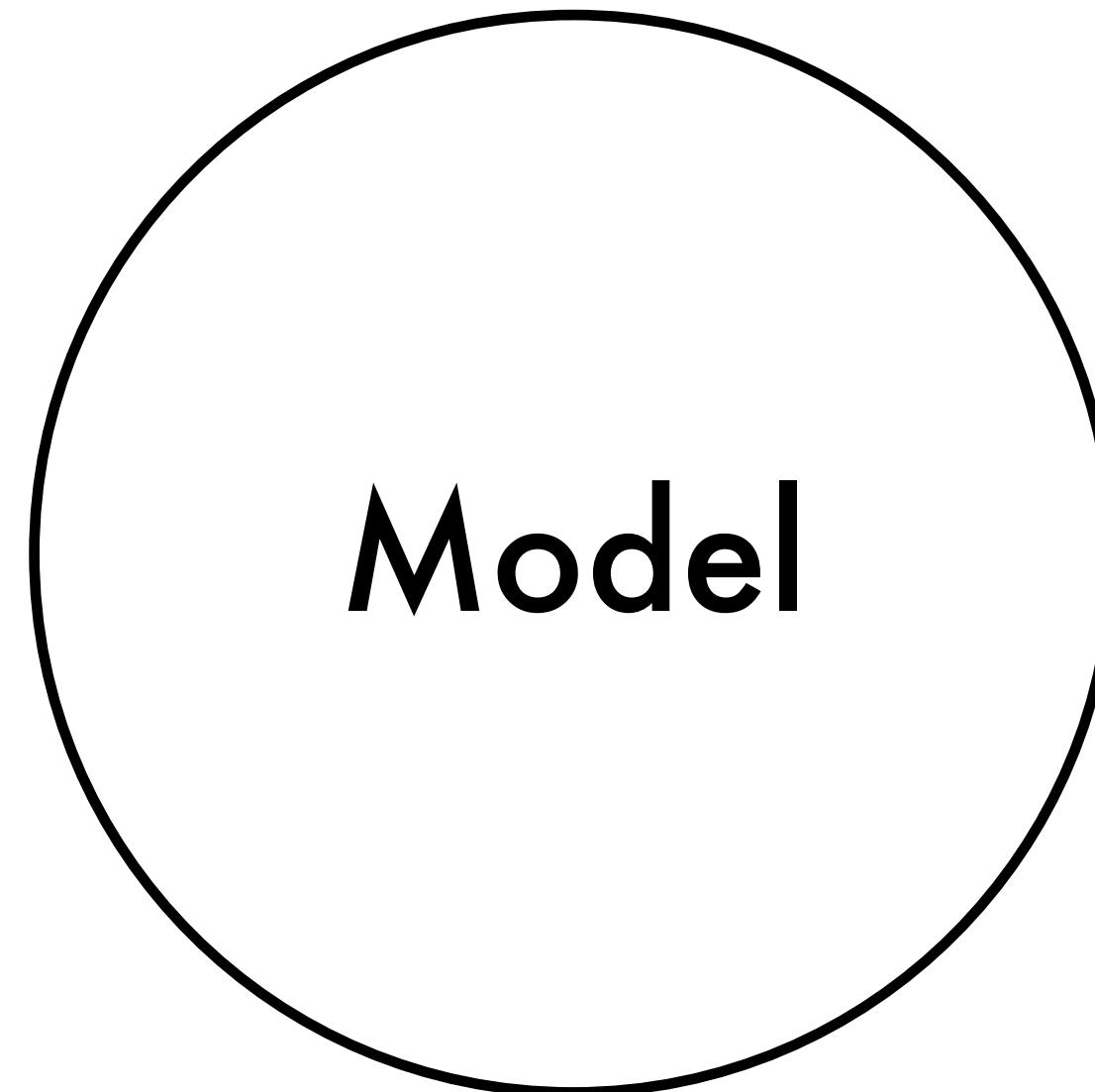
# Automatic ML Overview

- Feature Preprocessing
- Feature Engineering
- **Hyperparameter Search (part of H2O AutoML)**
- Ensembles

# What are Hyperparameters?

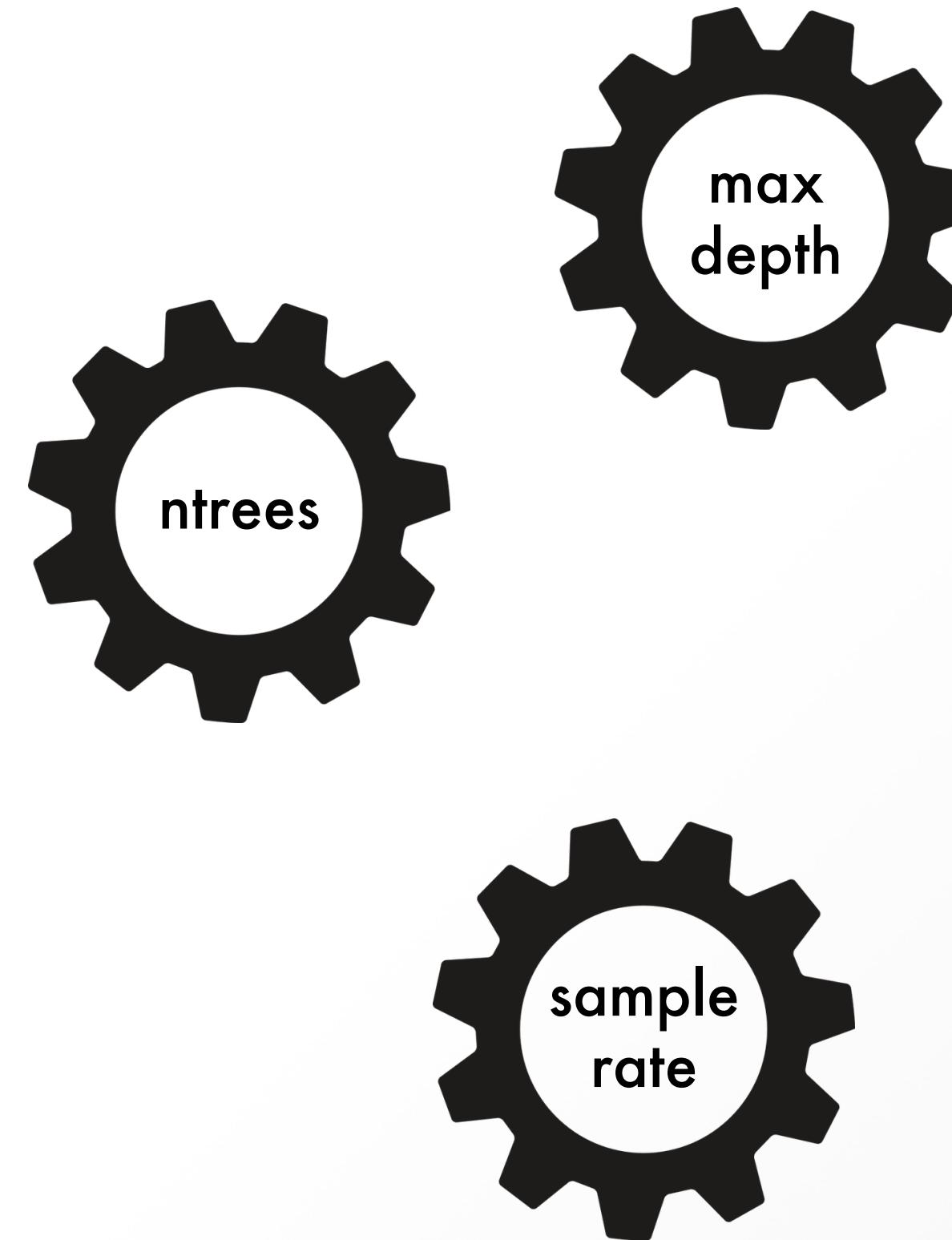
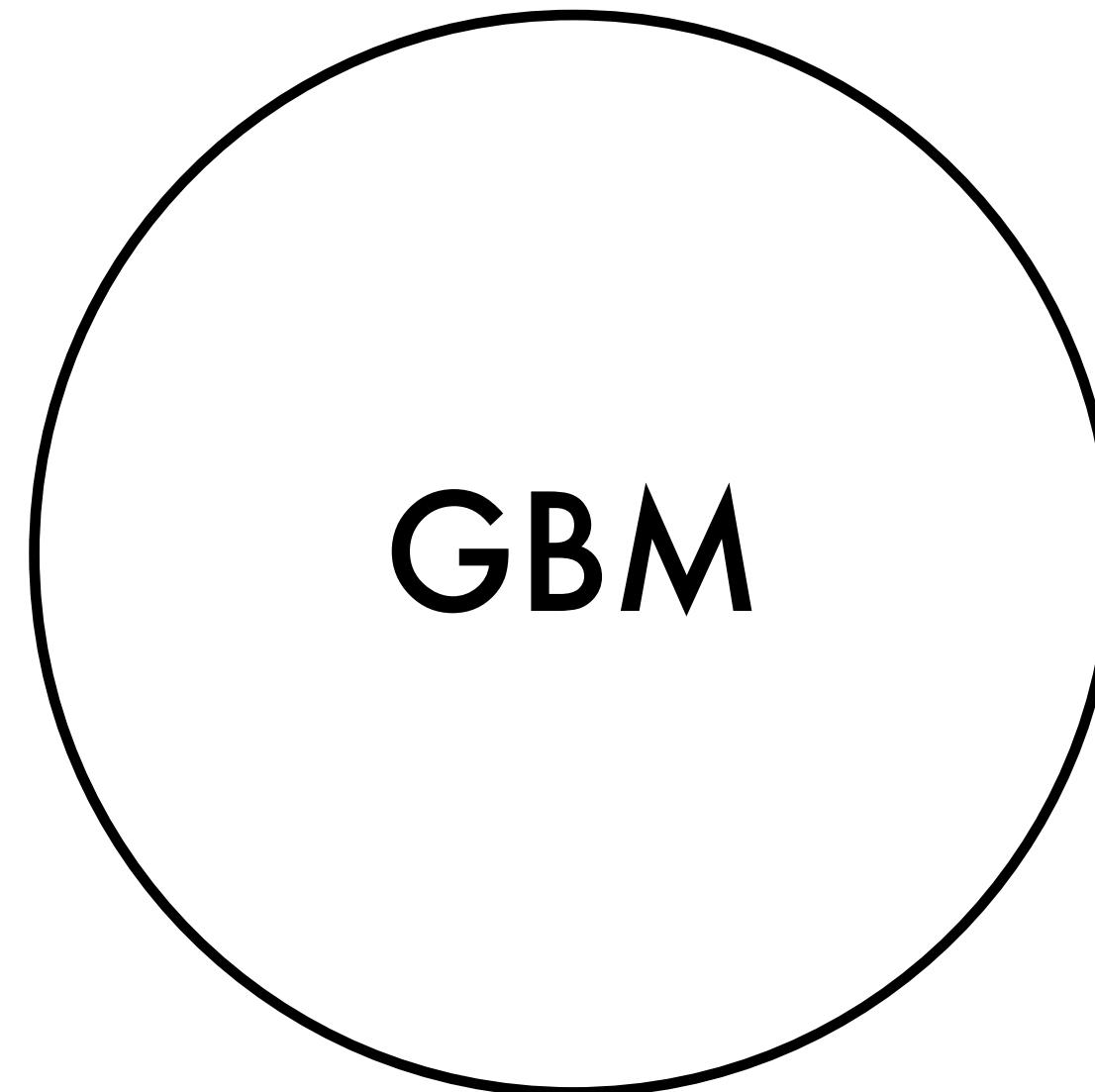
(a quick review)

# What are Model Hyperparameters?



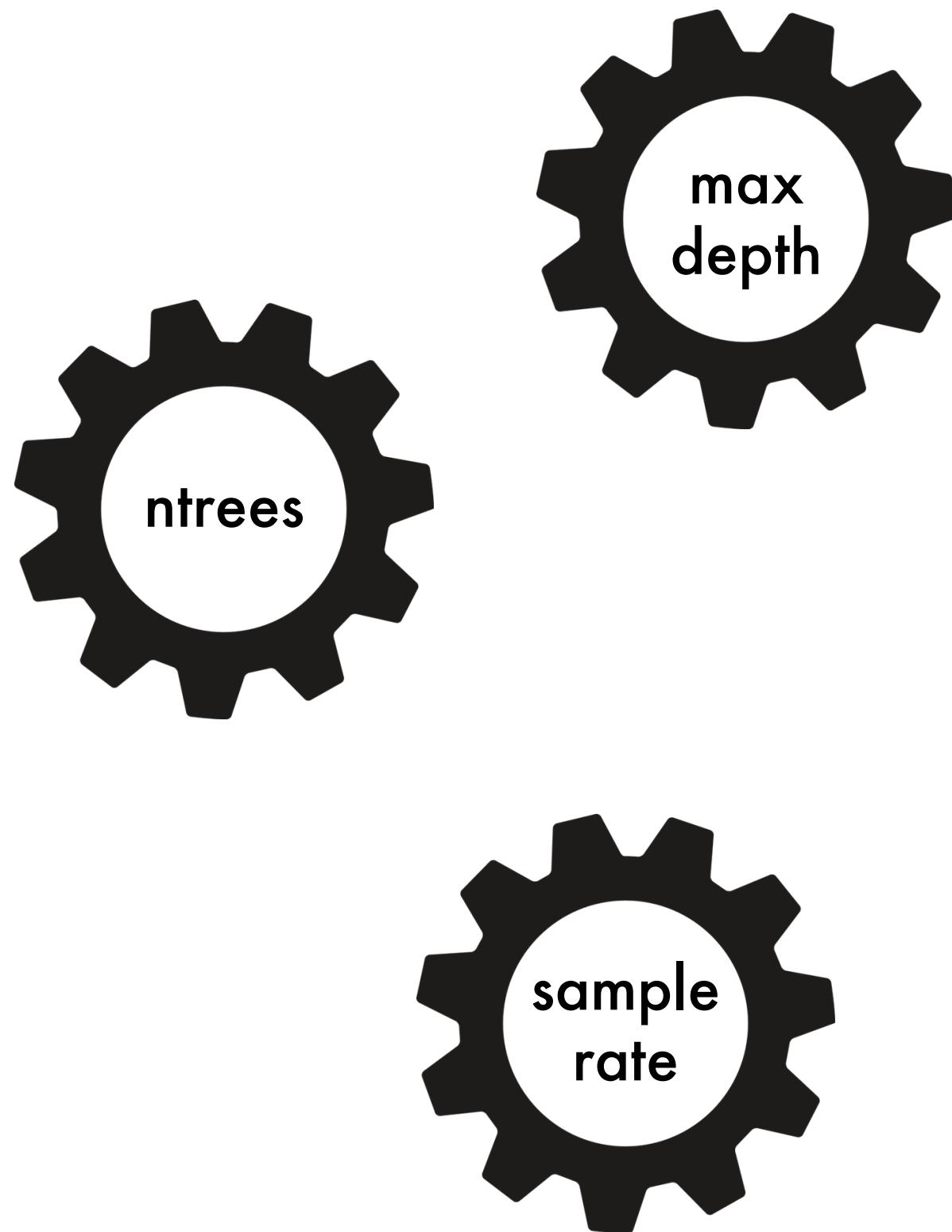
Every model comes with knobs

# What are Model Hyperparameters?



Every model comes with knobs

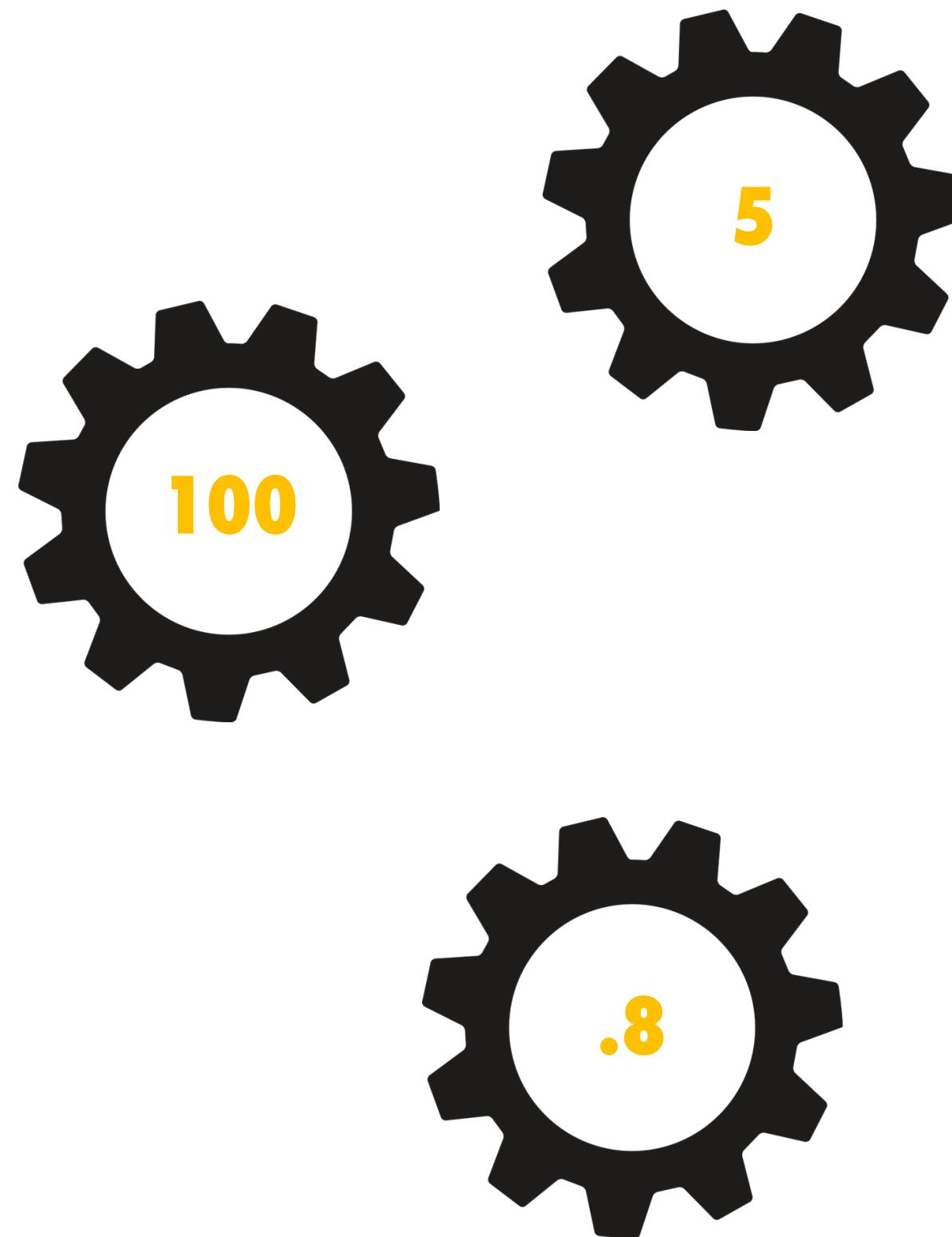
# What are Hyperparameter?



Unlike regular model parameters, hyperparameters are not learned from the data

The user has to set the hyperparameters **values**

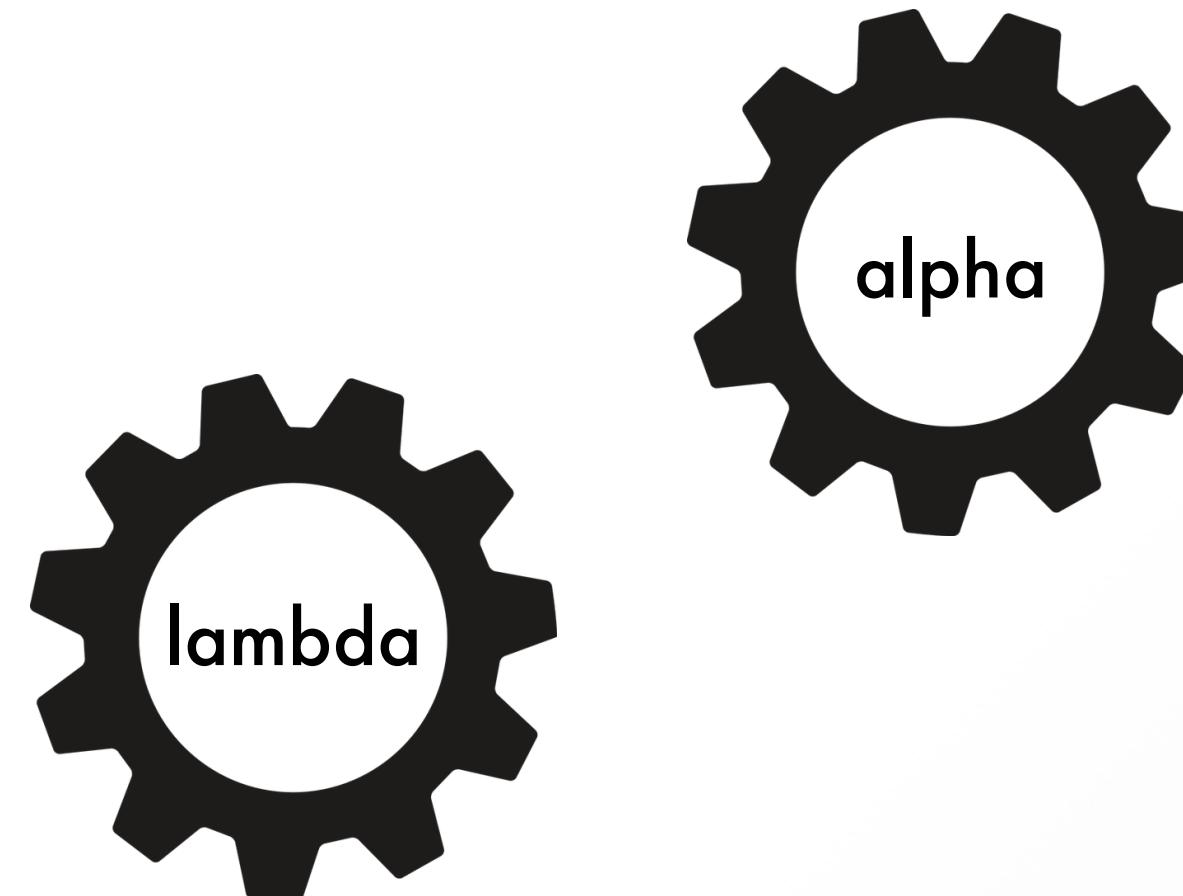
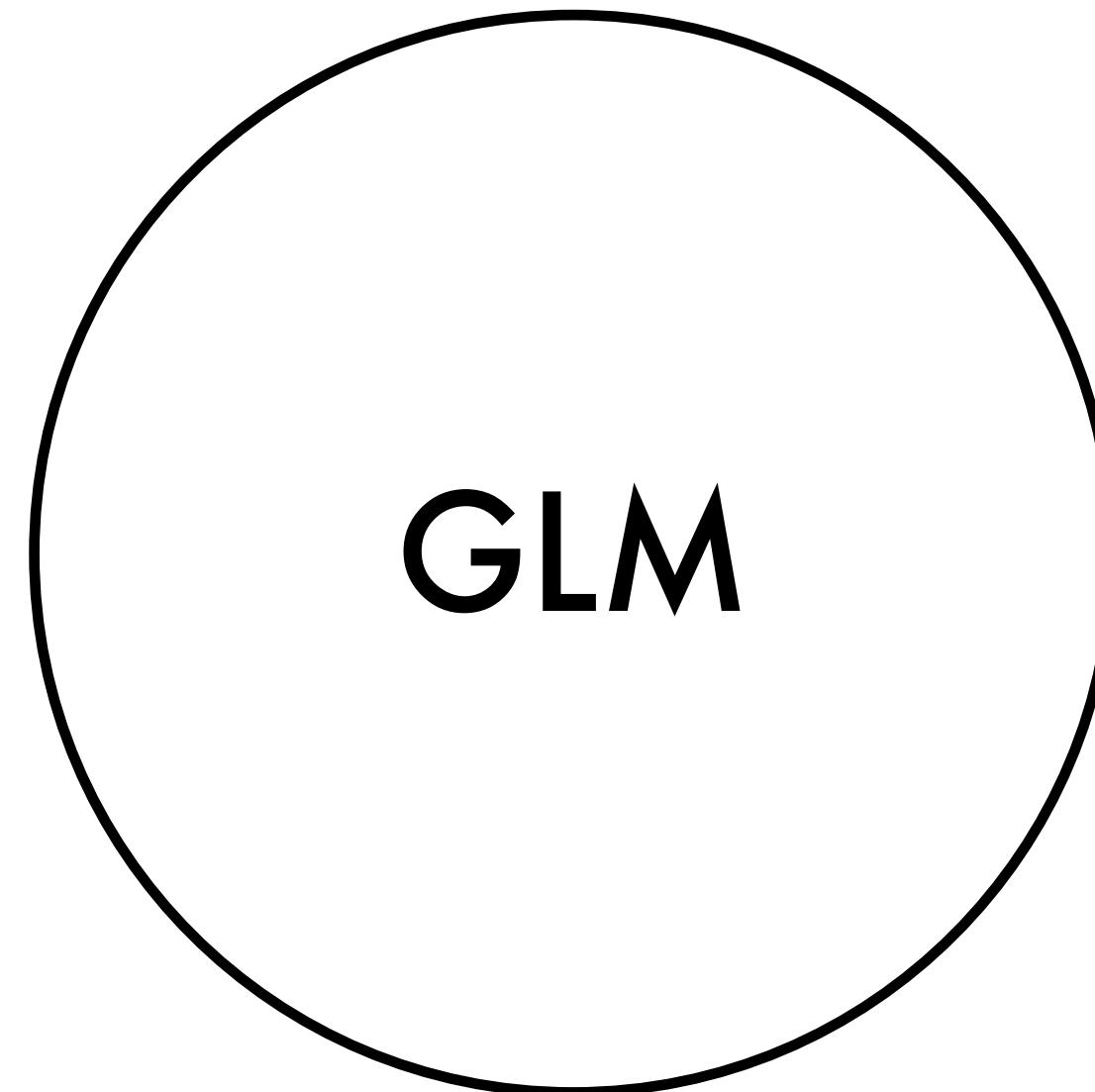
# What are Hyperparameter?



Unlike regular model parameters, hyperparameters are not learned from the data

The user has to set the hyperparameters **values**

# What are GLM's Hyperparameters?



# GLM Model Parameters

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left( \alpha \|\beta\|_1 + \frac{1}{2}(1 - \alpha) \|\beta\|_2^2 \right)$$

# GLM Model Parameters

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left( \alpha \|\beta\|_1 + \frac{1}{2}(1-\alpha) \|\beta\|_2^2 \right)$$

# GLM Hyperparameter

$$\max_{\beta, \beta_0} \sum_{i=1}^N \log f(y_i; \beta, \beta_0) - \lambda \left( \alpha \|\beta\|_1 + \frac{1}{2} (1 - \alpha) \|\beta\|_2^2 \right)$$

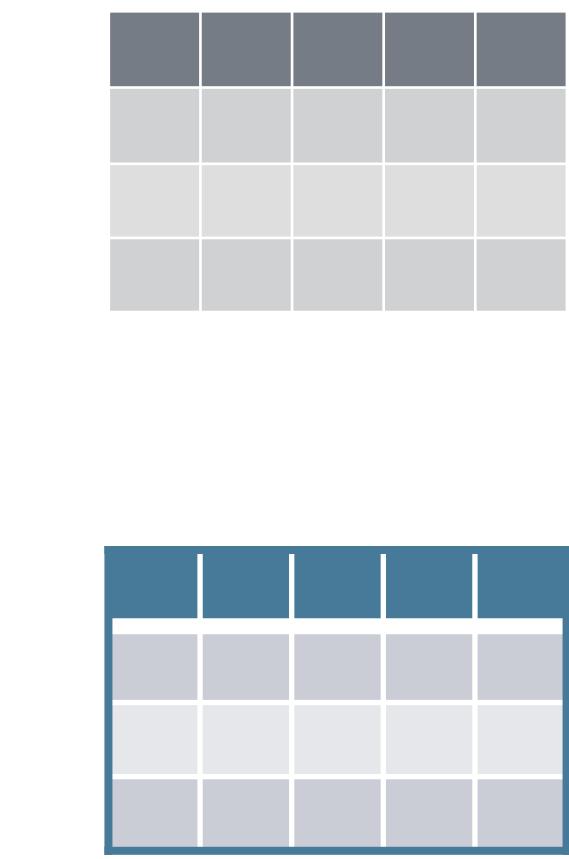
# Model Building: Hyperparameter Search

## 4 Main Ways:

- Manual Model Tuning
- Cartesian Grid Search
- Random Grid Search\*
- Bayesian Hyperparameter Optimization

\*What AutoML Uses

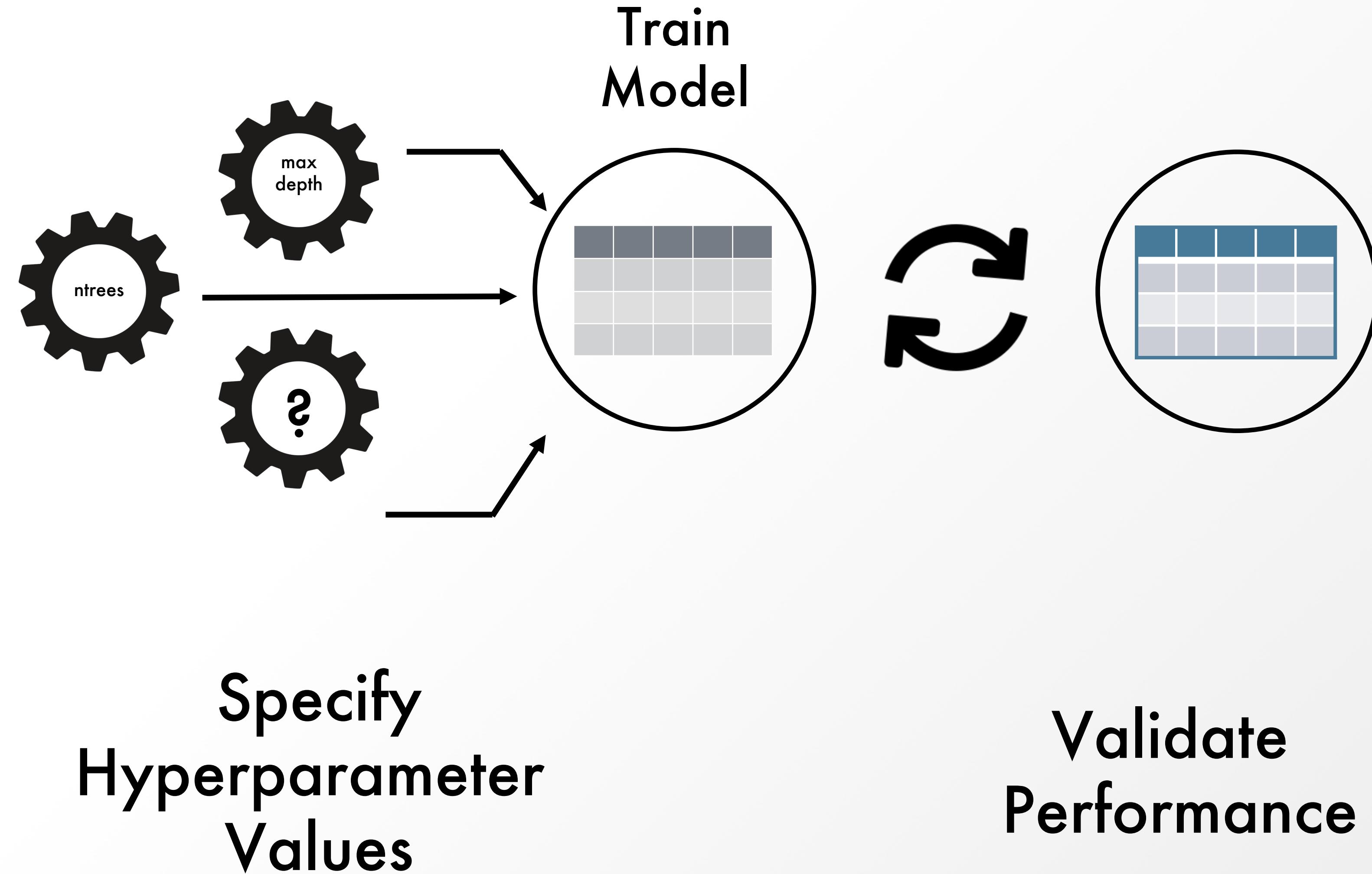
# Manual Model Tuning



Split Dataset

training  
data

validation  
data



# Manual Model Tuning

How did you do?

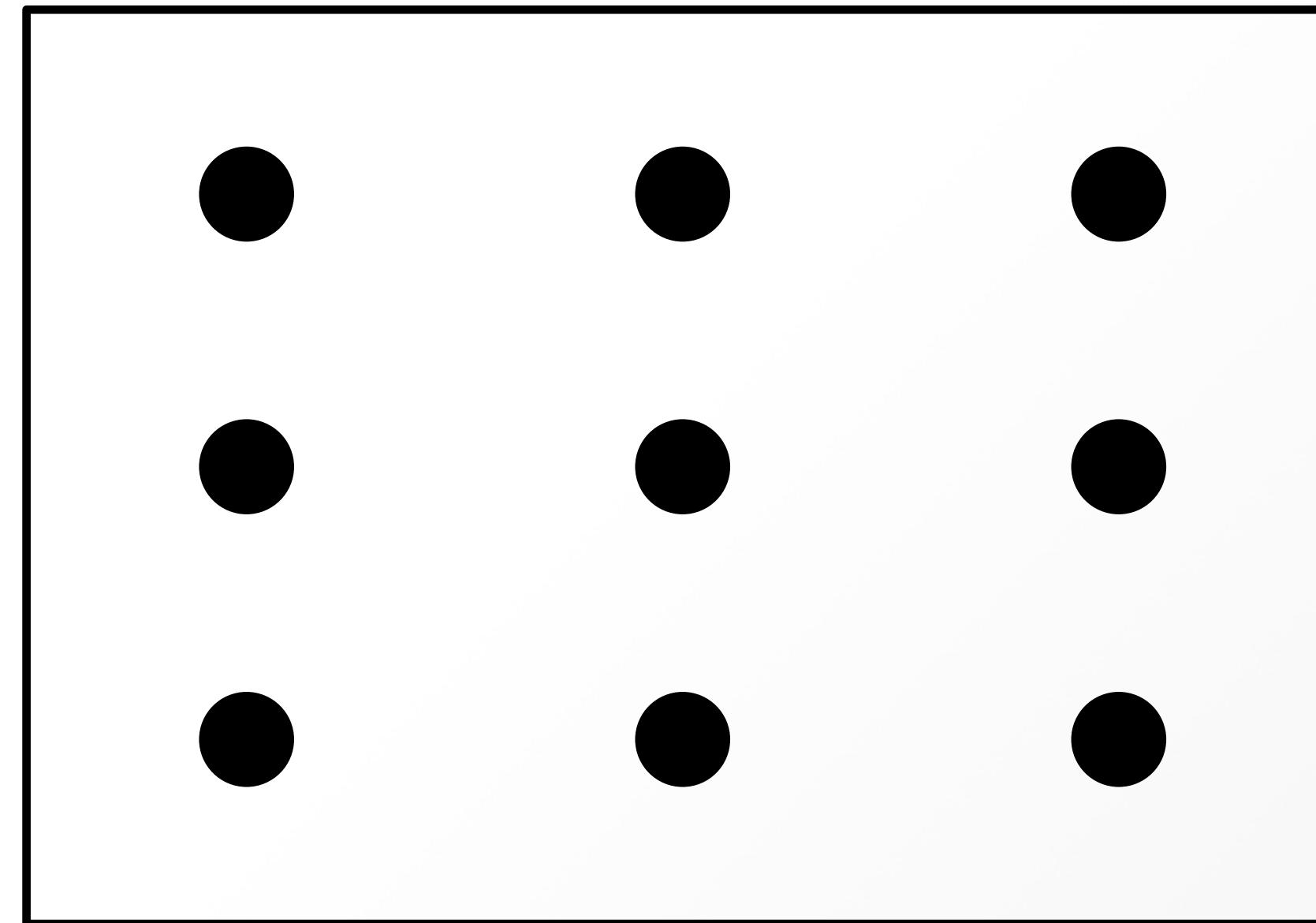
- Not so good: **Repeat** Process
- GOOD! : Stop

# **Cartesian Grid Search**

**AKA Exhaustive Search: Tries Every Combination**

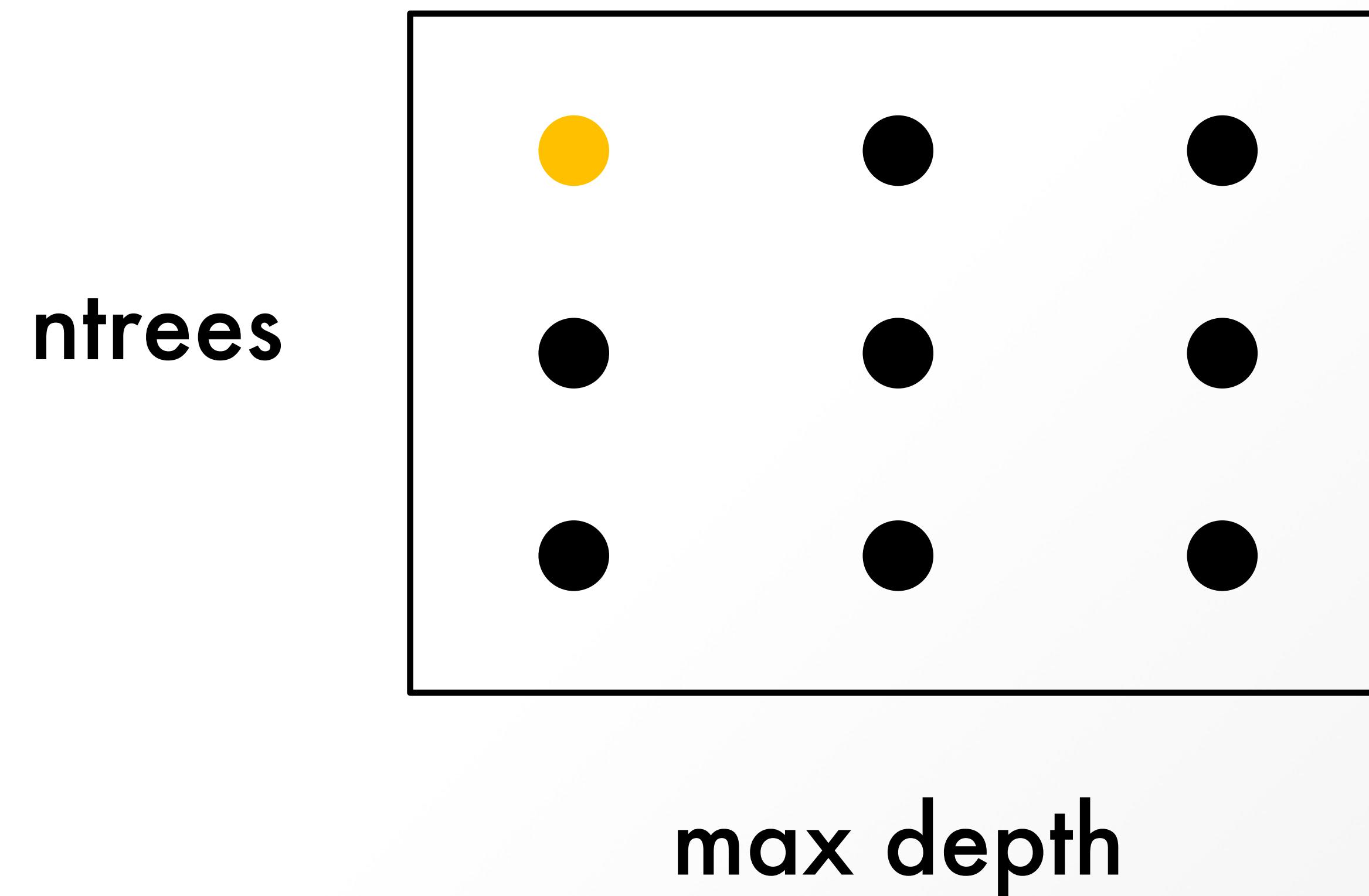
# Cartesian Grid Search

ntrees



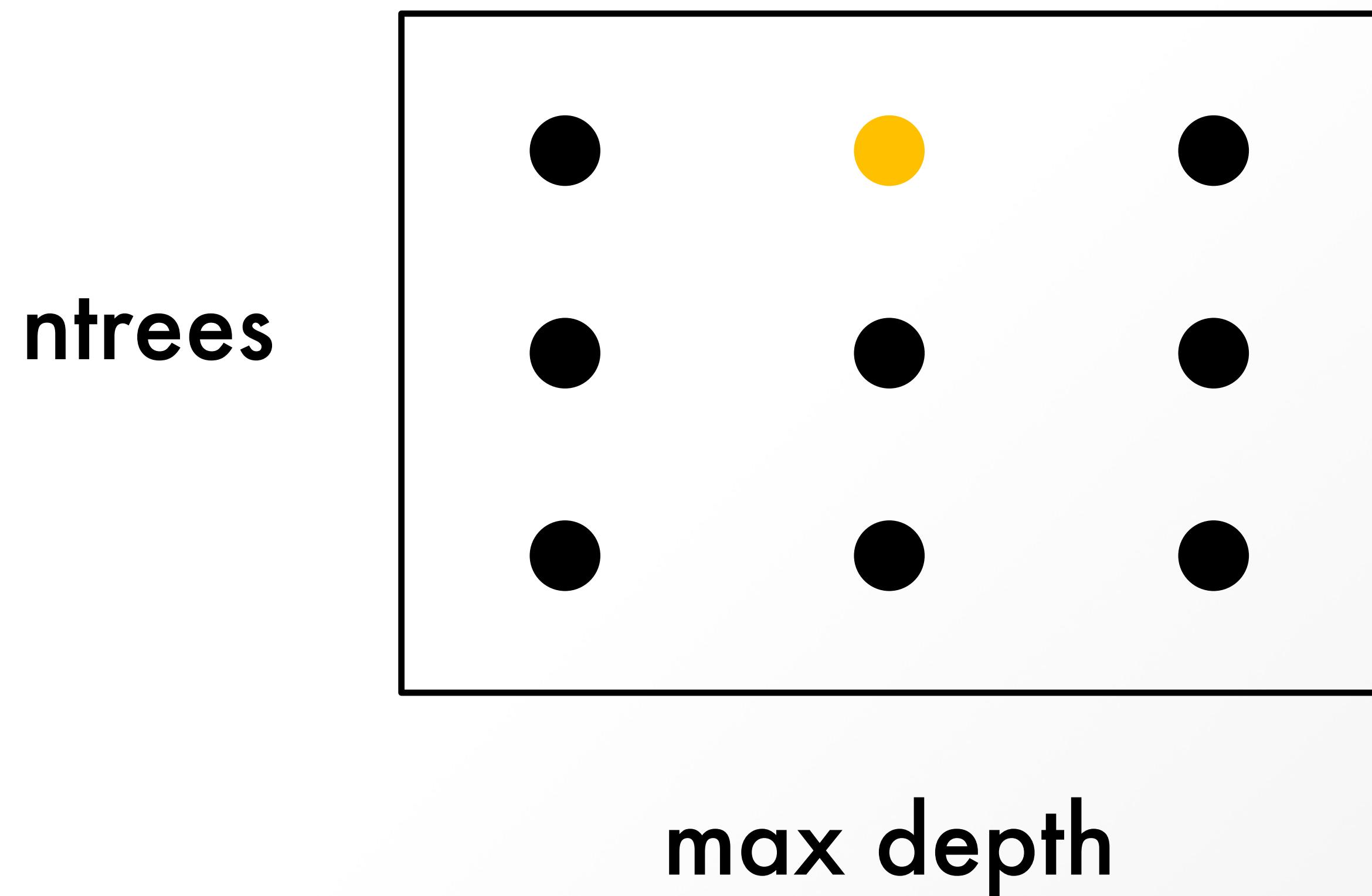
max depth

# Cartesian Grid Search



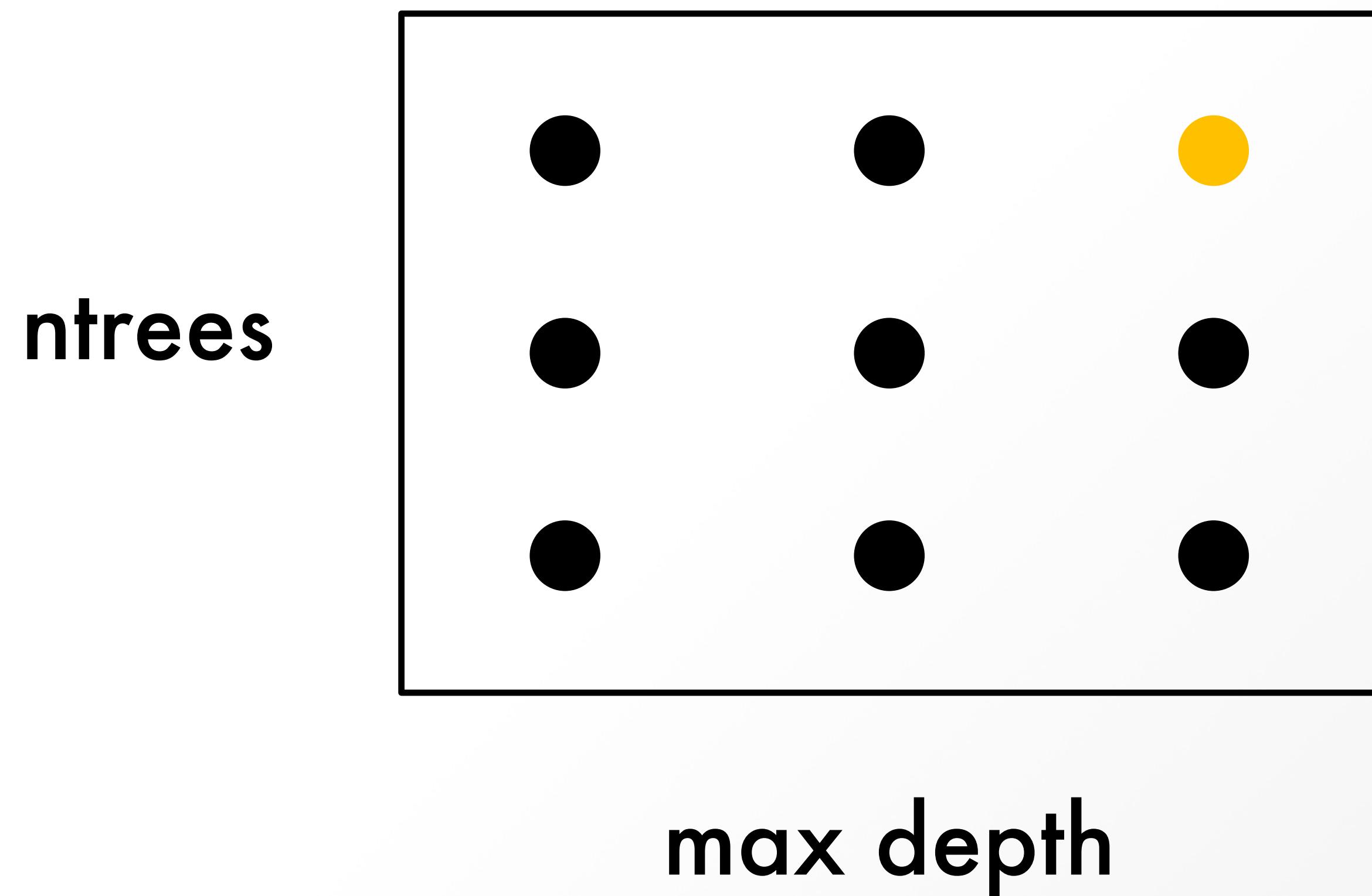
**single trial**  
ntrees = 1000  
max depth = 5

# Cartesian Search



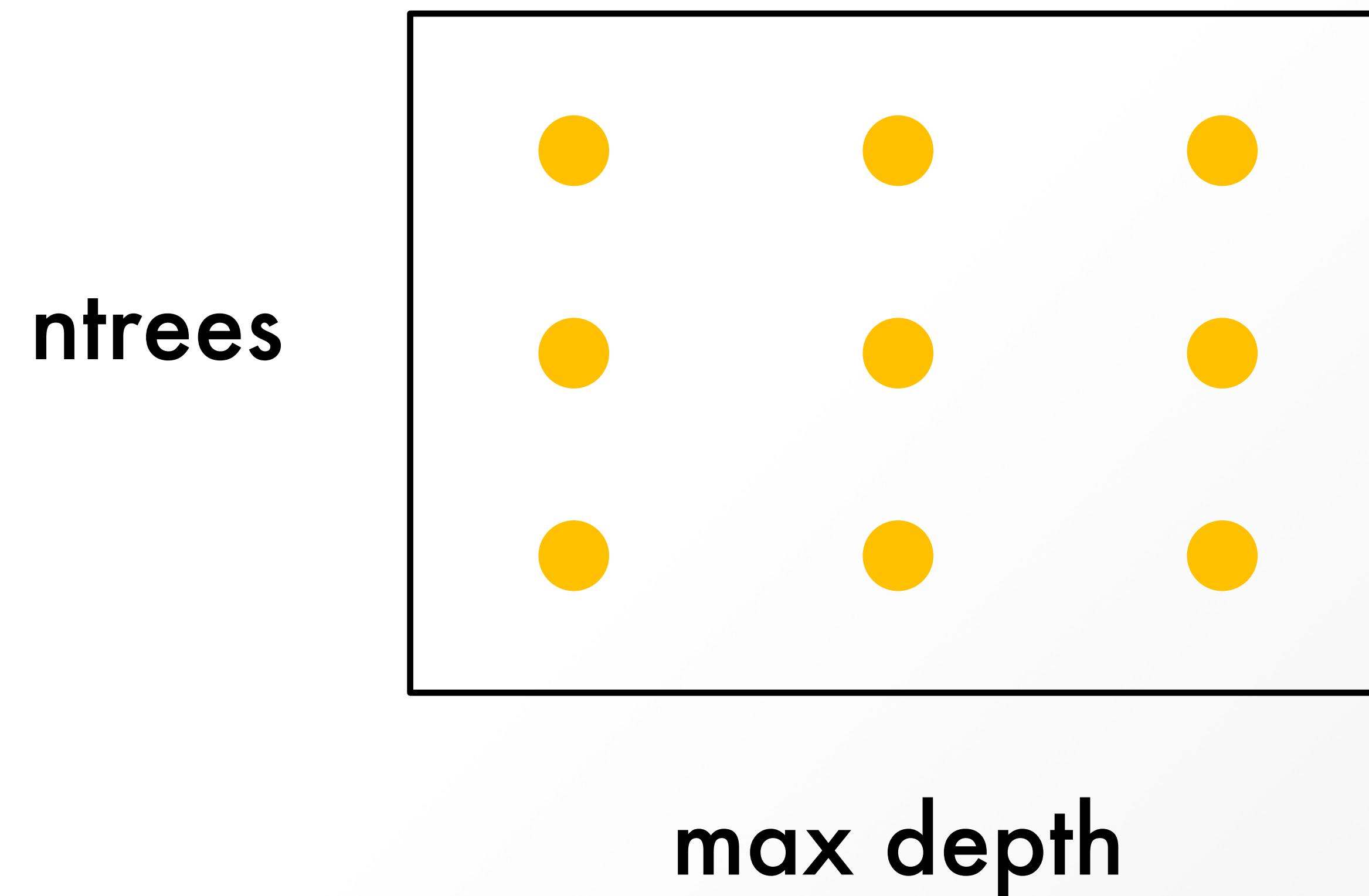
**single trial**  
ntrees = 1000  
max depth = 10

# Cartesian Search



**single trial**  
ntrees = 1000  
max depth = 15

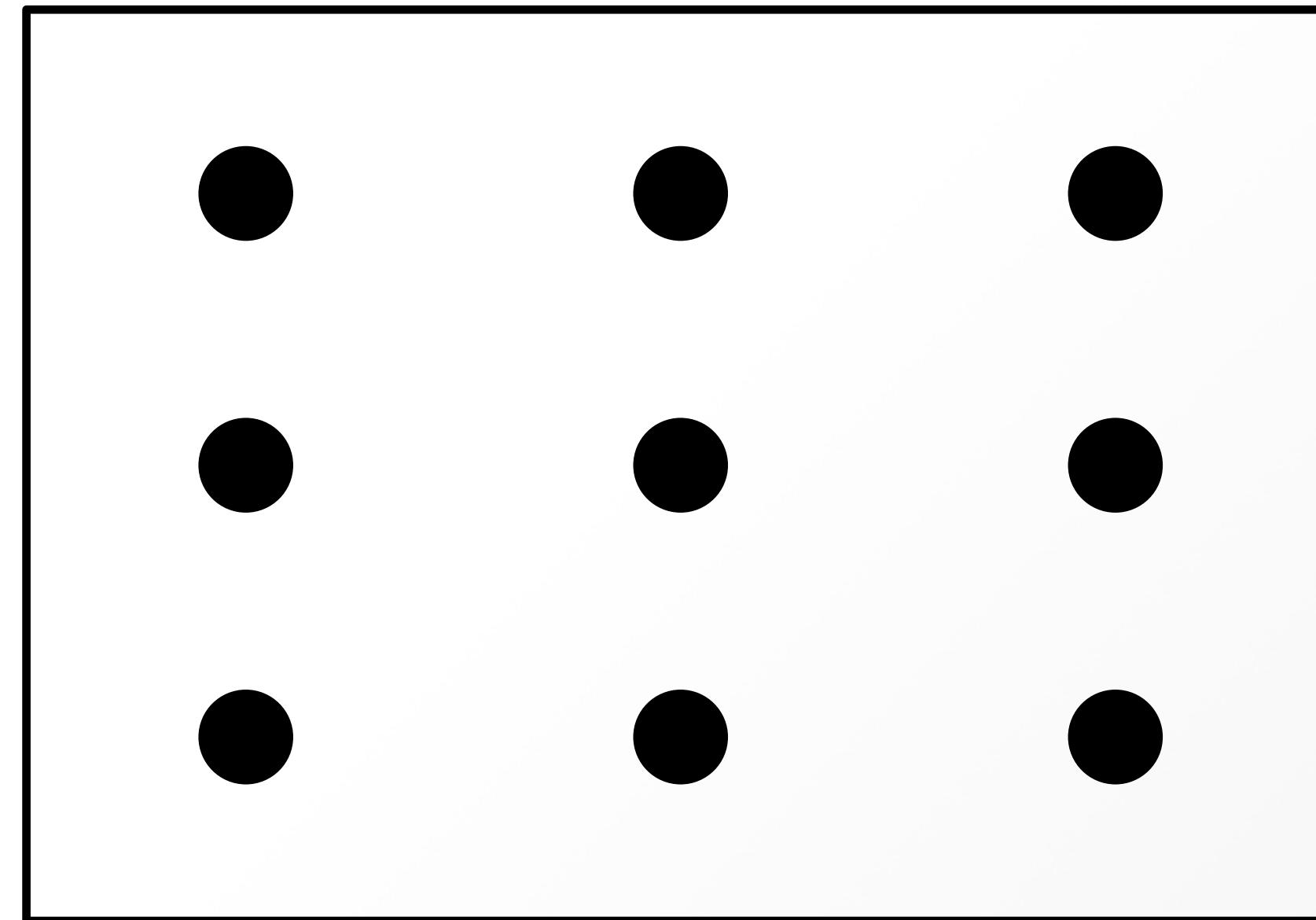
# Cartesian Grid Search



**Cartesian Grid  
Search**  
Complete Exploration

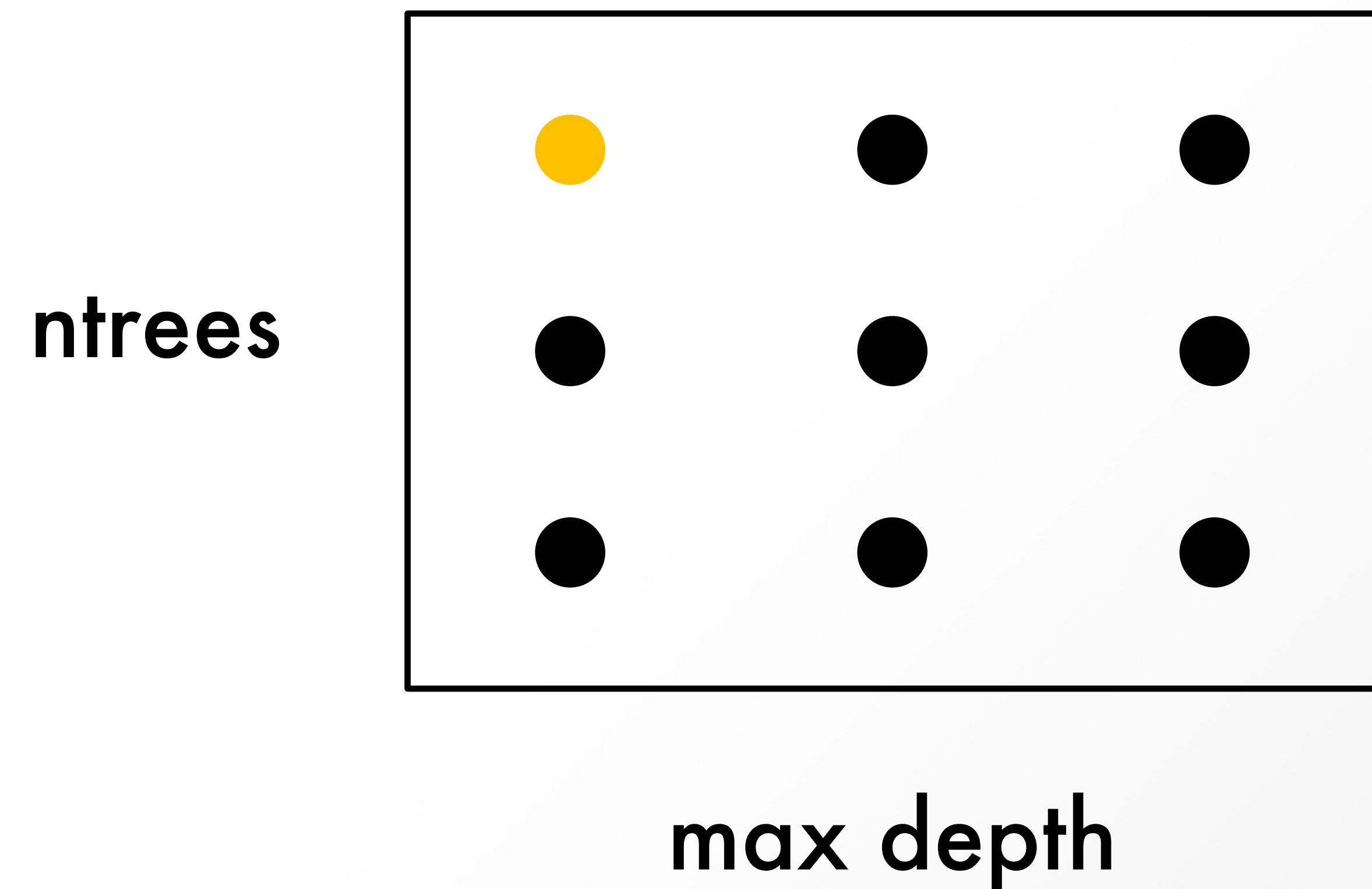
# Random Grid Search

ntrees



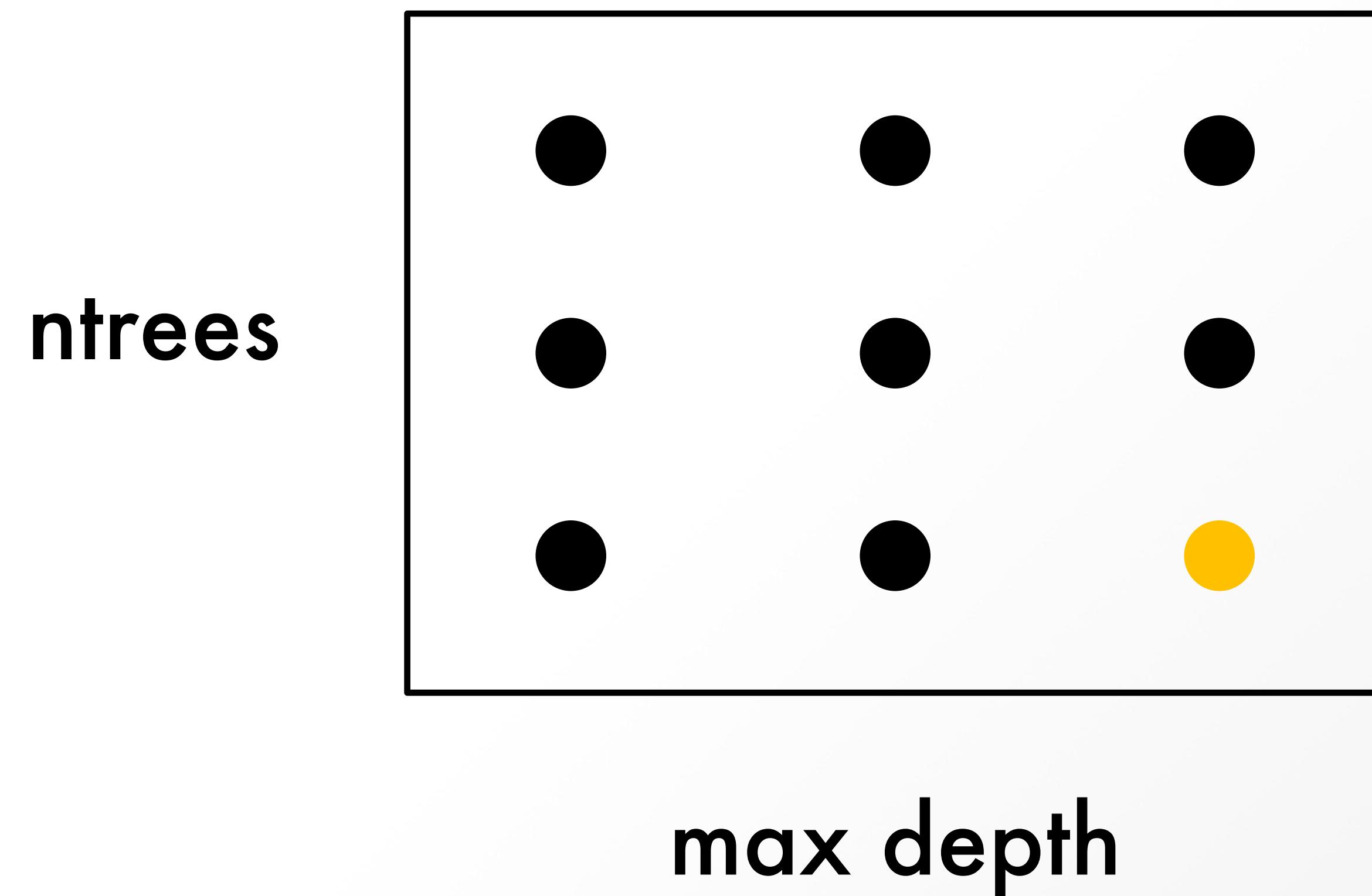
max depth

# Random Grid Search



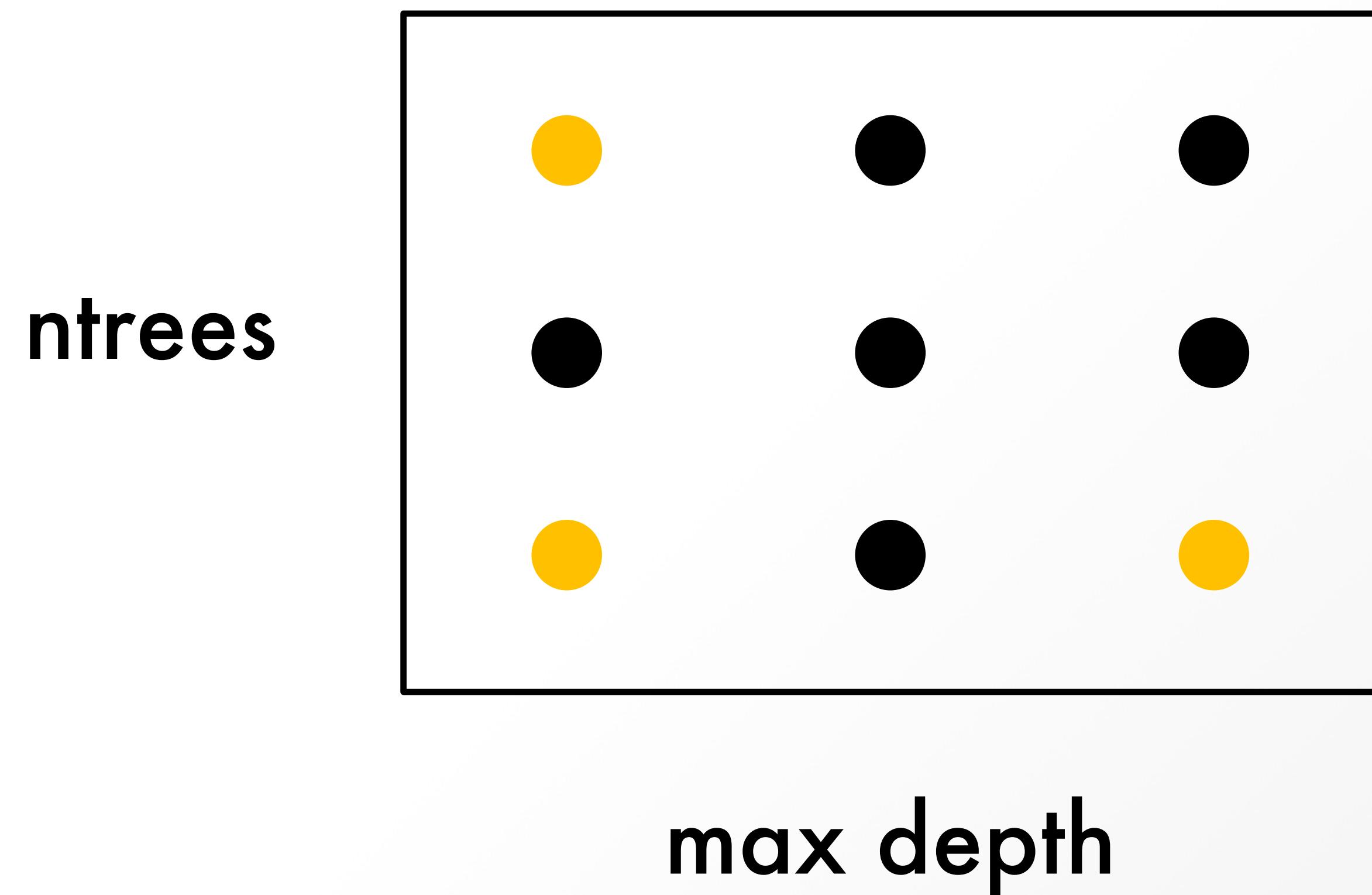
**single trial**  
`ntrees = 1000`  
`max depth = 5`

# Random Grid Search



**single trial**  
ntrees = 3000  
max depth = 15

# Random Grid Search

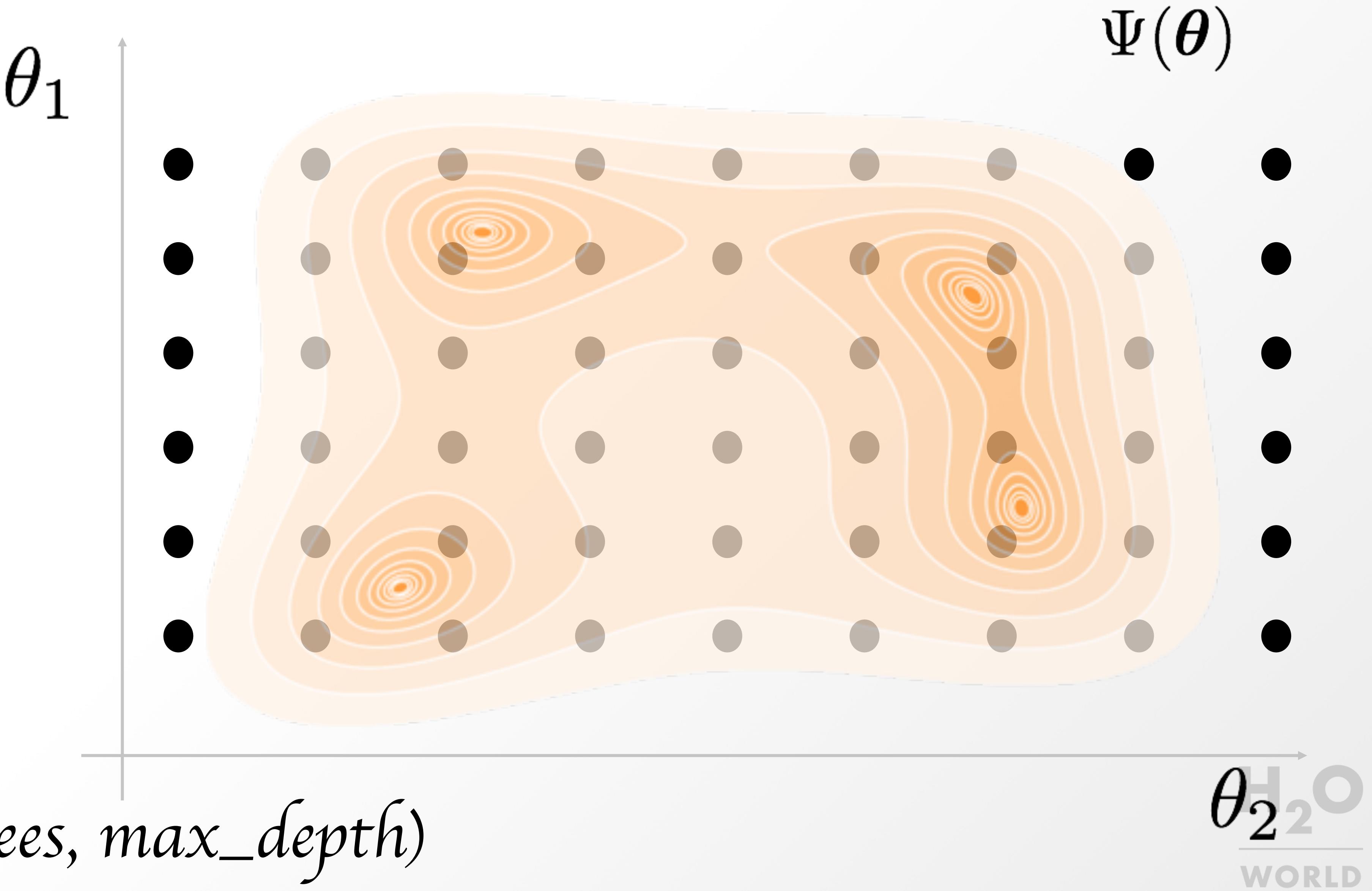


**Random Grid  
Search**  
explores more in  
less time

# Other ways to Optimize?

$$\theta^* = \arg \min_{\theta \in \Theta} \Psi(\theta)$$

$$\theta = (\theta_1, \theta_2) = (ntrees, max\_depth)$$



# Bayesian Optimization: An Analogy

Consider the following question:

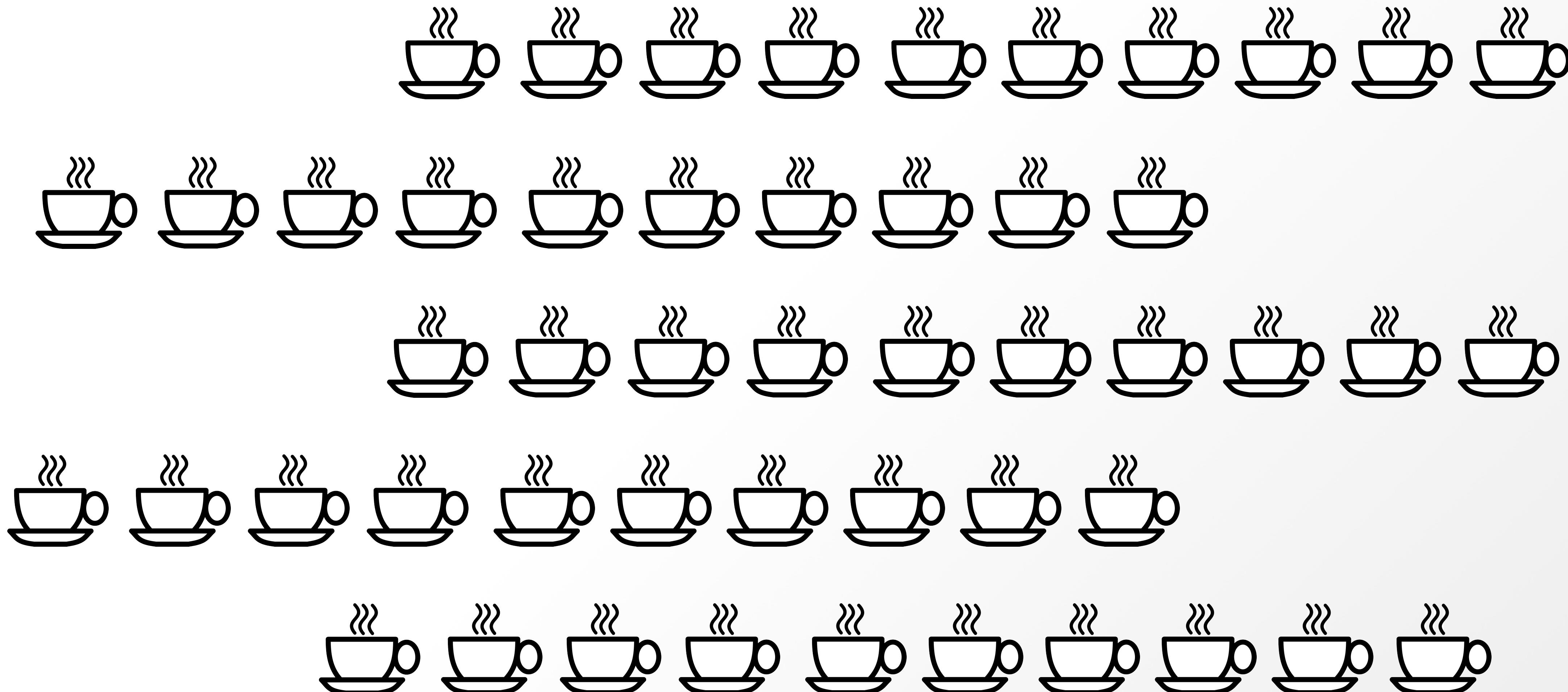
**How would you make the best cup of coffee in world?**

Given:

1. You have access to **every coffee** bean in the world
2. You have **every coffee instrument** ever invented
3. You only have **ONE WEEK**

# Bayesian Optimization: An Analogy

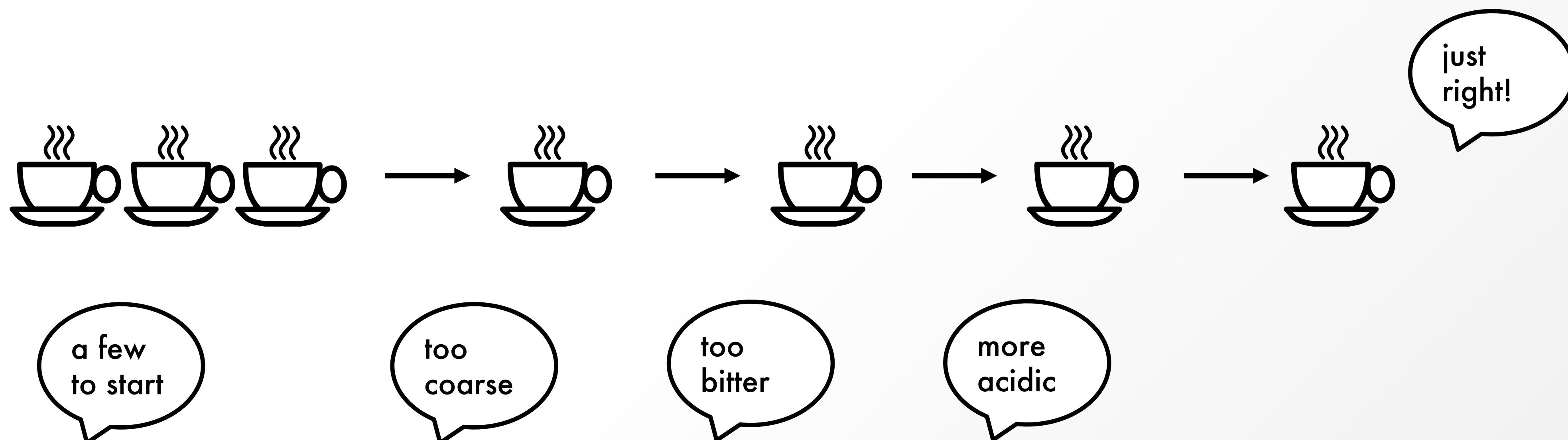
**you could brew all the coffee!**



# Bayesian Optimization: An Analogy

**What if you had a way to know how to adjust the settings**

**after every cup of coffee**

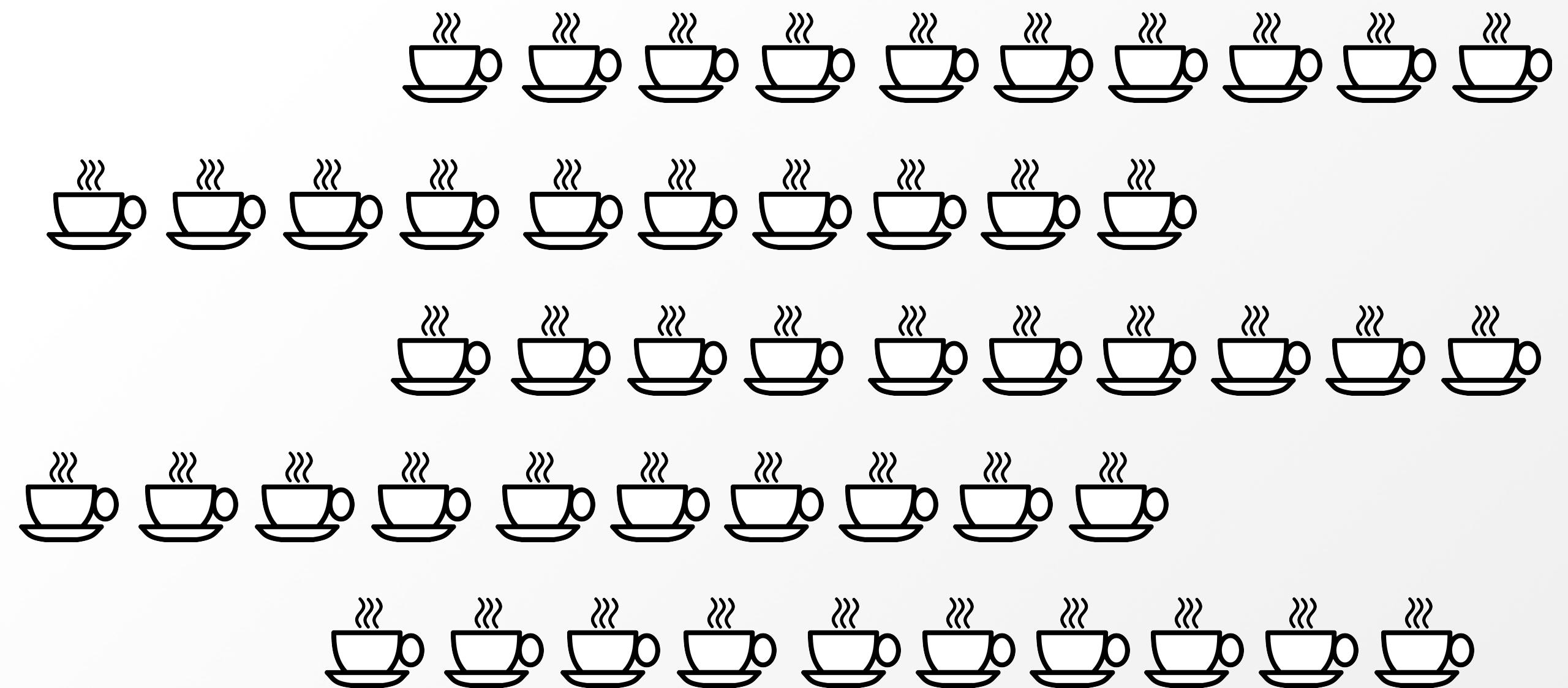


# Bayesian Optimization: An Analogy

More time drinking good coffee



VS



# Bayesian Optimization: An Analogy

**Same idea** applies to Bayesian Optimization for Modeling

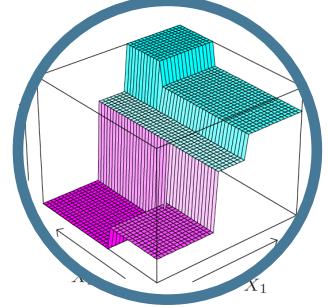
1. Try a few different hyperparameters => see what happens
2. get suggestion on what to try next
3. Now you don't have to train an infinite number of models

# Automatic ML Overview

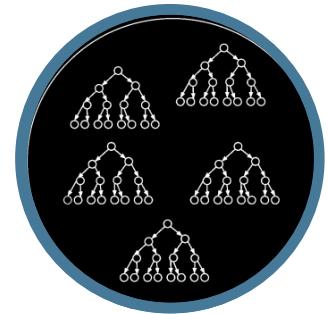
- Feature Preprocessing
- Feature Engineering
- Hyperparameter Search (part of H2O AutoML)
- Ensembles (**part of H2O AutoML**)

# Ensembles

## Vanilla Ensembles

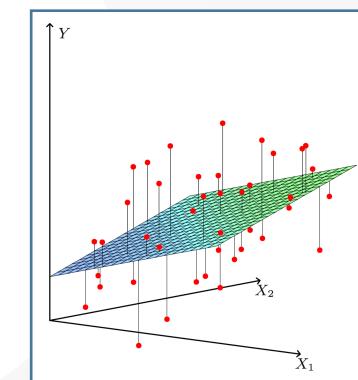
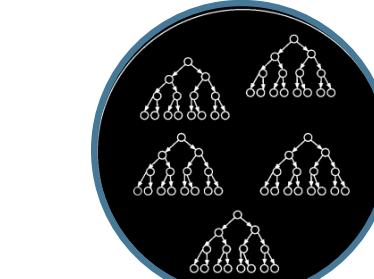
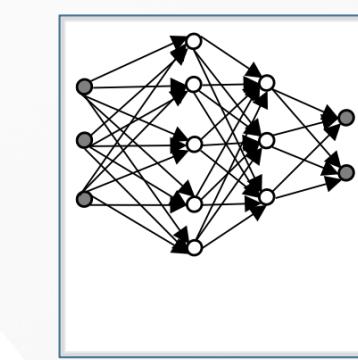
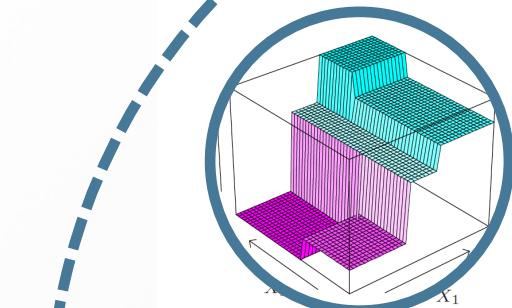


Gradient Boosting Machine



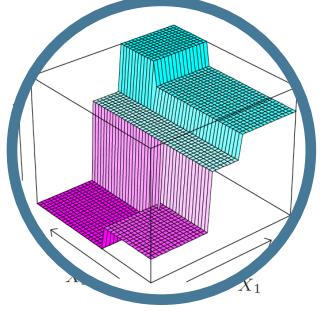
Random Forest

## Super Learning Ensembles

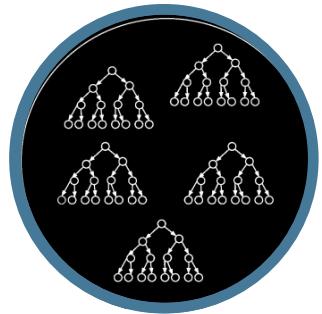


# Ensembles

## Vanilla Ensembles

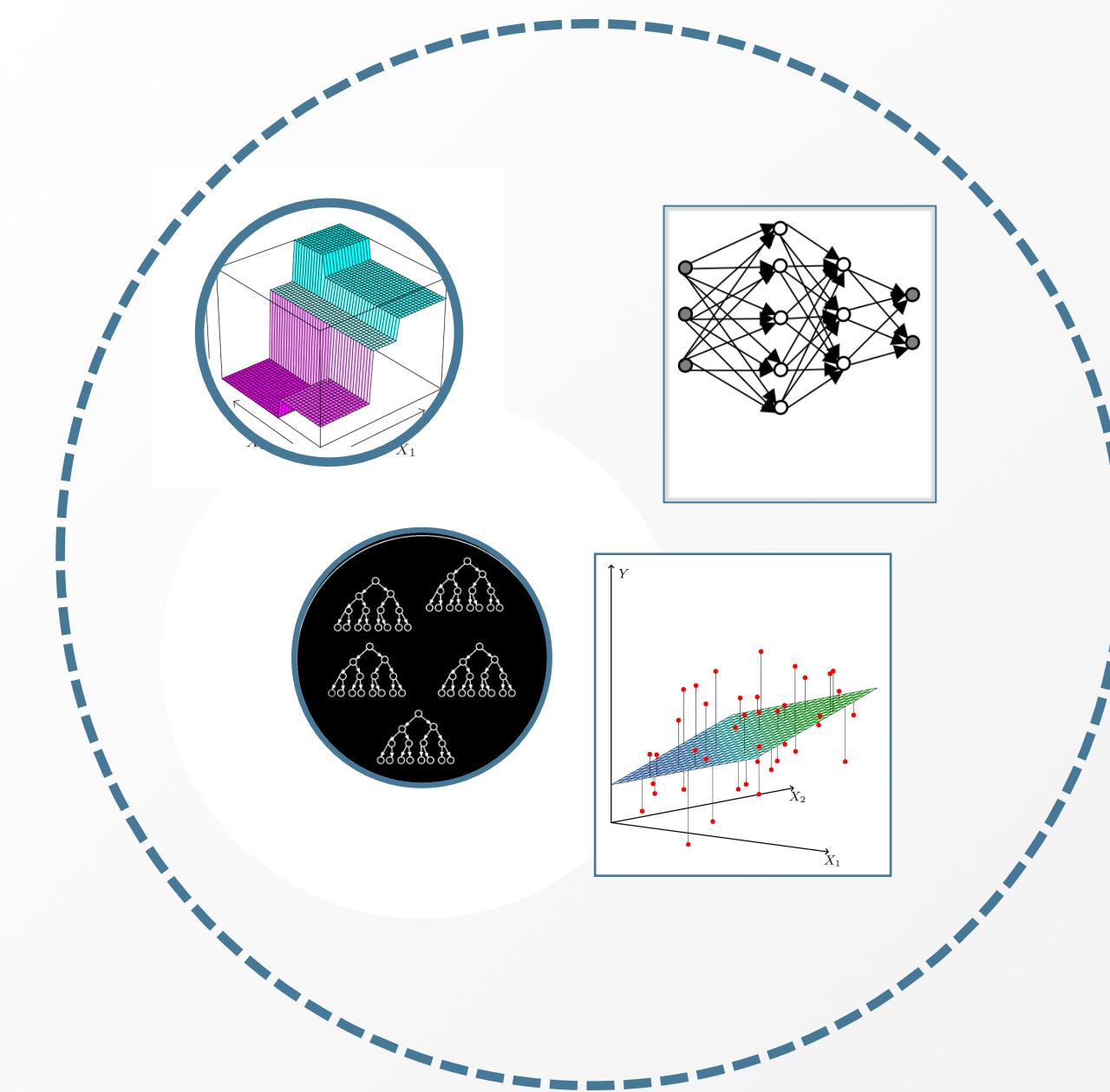


Gradient Boosting Machine



Random Forest

## (AKA Stacking)



# **More on Stacked Ensembles**

**“Stacking is an ensemble procedure where a second-level learner is trained on the output of a collection of base learners.”**

# More on Stacked Ensembles

“Stacking is an ensemble procedure where a **second-level** learner is trained on the output of a collection of base learners.”

# More on Stacked Ensembles

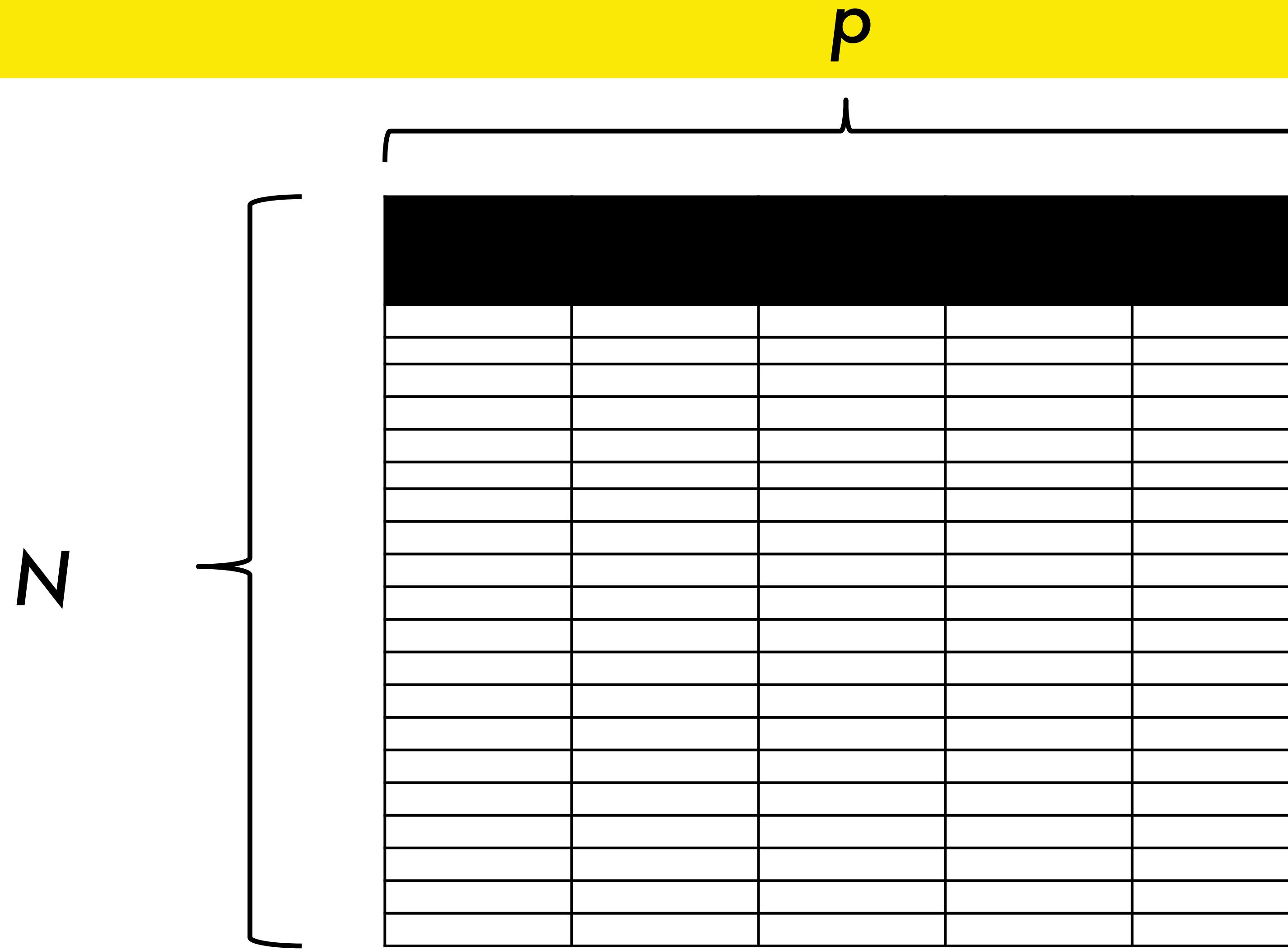
“Stacking is an ensemble procedure where a **metalearner** is trained on the output of a collection of base learners.”

# More on Stacked Ensembles

“Stacking is an ensemble procedure where a second-level learner is trained on the output of a collection of **base learners**.”

# **Stacked Ensemble Steps**

- 1. Specify original dataset for training**
2. Specify algorithms with which to train
3. Stack each algo's cross-validated predictions
4. Train final model on cross-validated results



**your original  
training data**

**“Level-zero” data**

# Stacked Ensemble Steps

1. Specify original dataset for training
2. **Specify algorithms with which to train**
3. Stack each algo's cross-validated predictions
4. Train final model on cross-validated results

**supervised algos used to train**

**GLM**

**DRF**

**XRT**

**GBM**

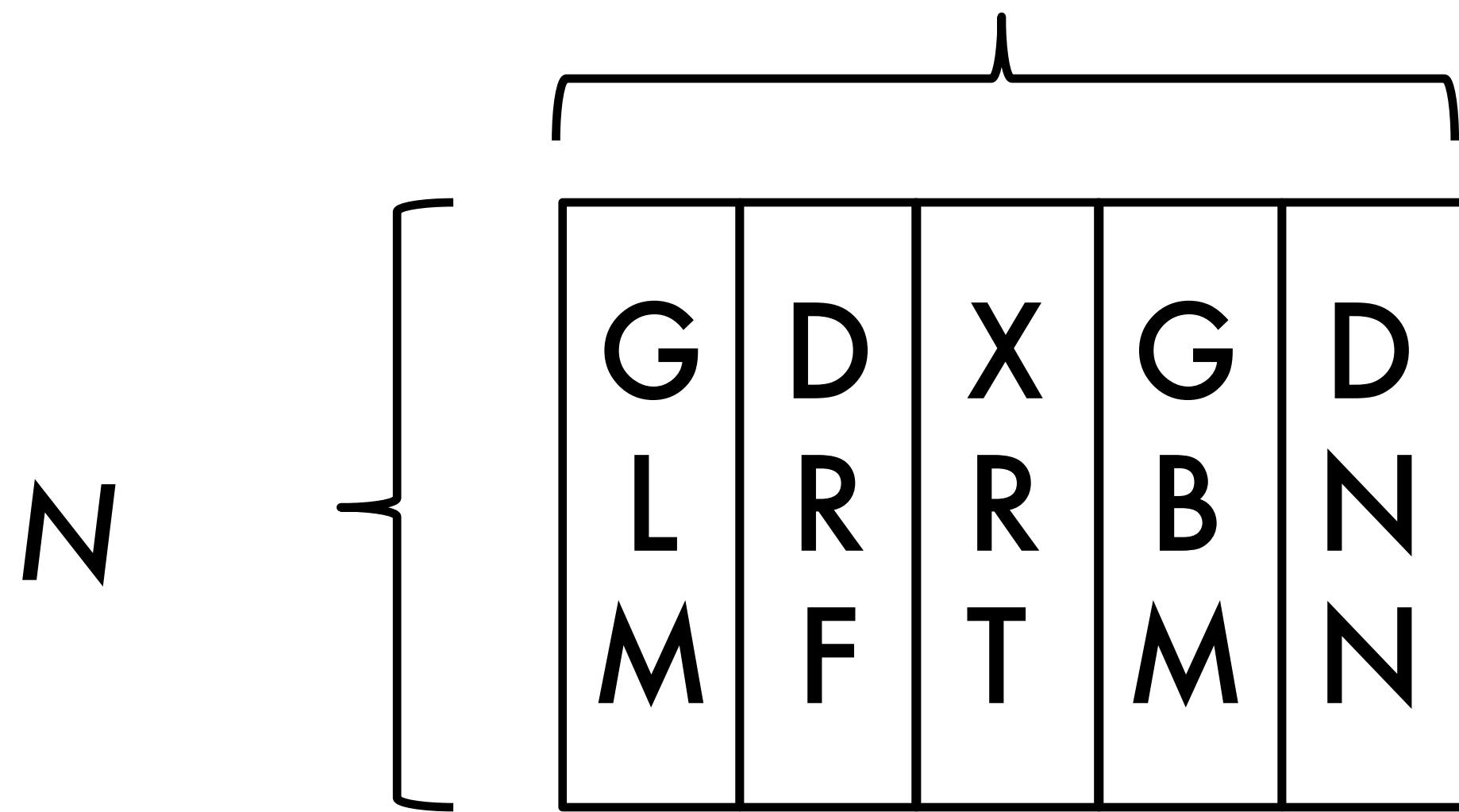
**DNN**

**base learners**

# Stacked Ensemble Steps

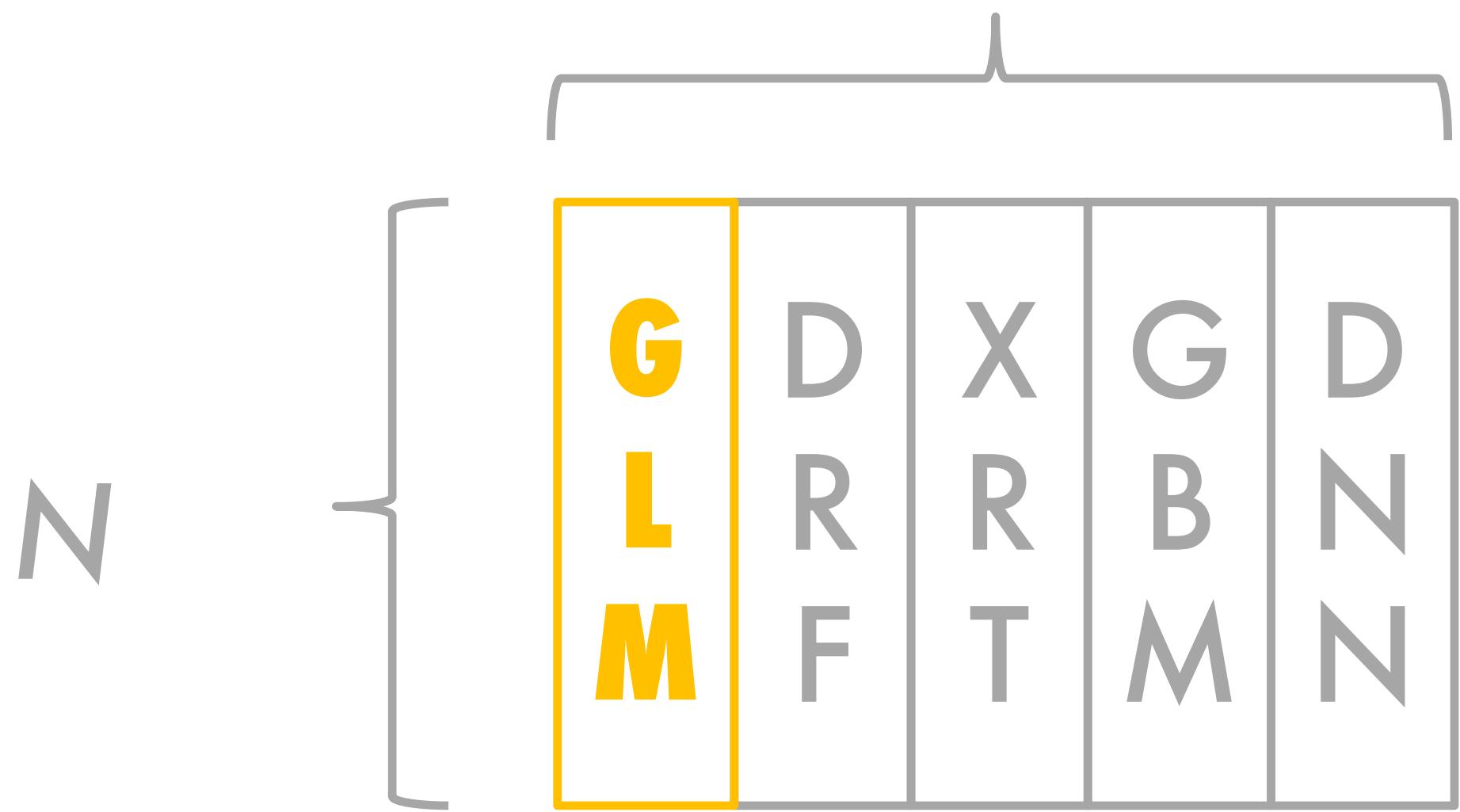
1. Specify original dataset for training
2. Specify algorithms with which to train
3. Stack each algo's cross-validated predictions
4. Train final model on cross-validated results

L base learners



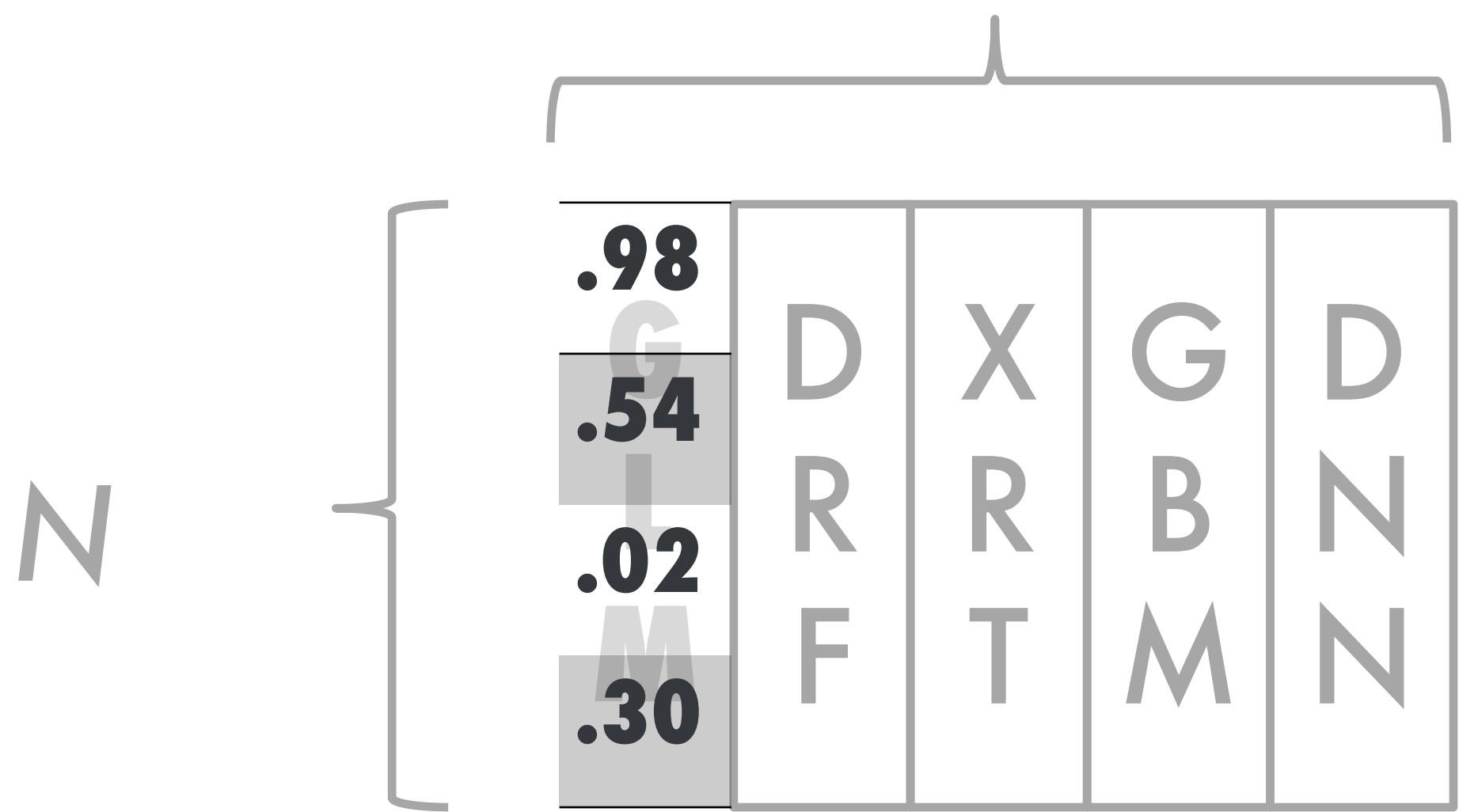
stacking your  
base learners

## L base learners



**meaning:**  
each column is an algorithm's  
**cross-validated predictions**

## L base learners



**meaning:**  
each column is an algorithm's  
**cross-validated predictions**

L base learners

N	10	D	X	G	D
G	5.5	R	R	B	N
R	7	T		M	N
F	M				
30					

meaning:  
each column is an algorithm's  
**cross-validated predictions**

How do we get the  
cross-validated predictions?

# **How do we get the cross-validated predictions in H2O?**

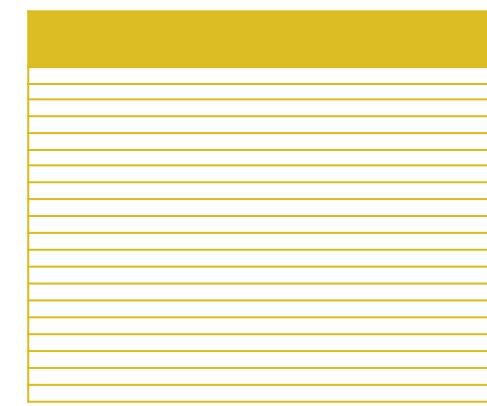
```
keep_cross_validation_predictions = True
```

# **what is cross-validation?**

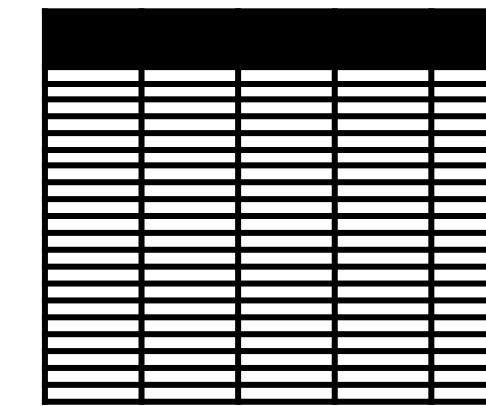
**quick review**

# Without K-Fold Cross Validation:

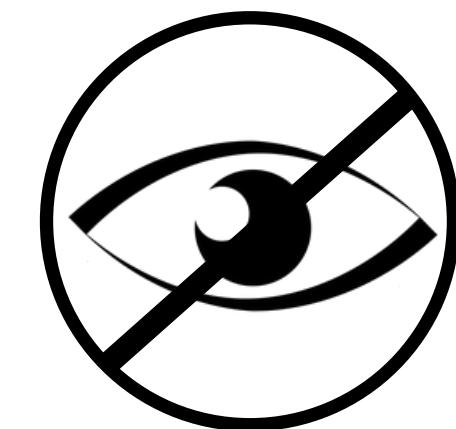
A)



Train

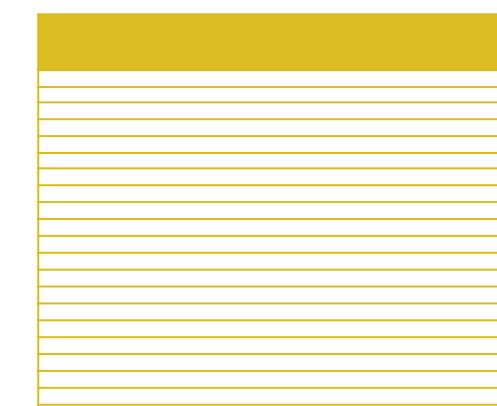


Test

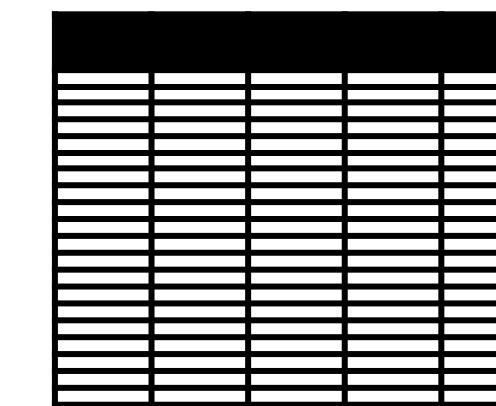


# Without K-Fold Cross Validation:

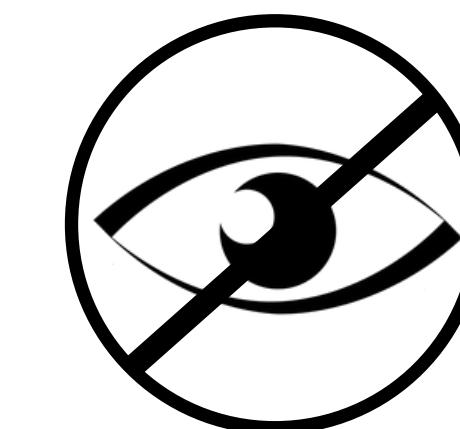
A)



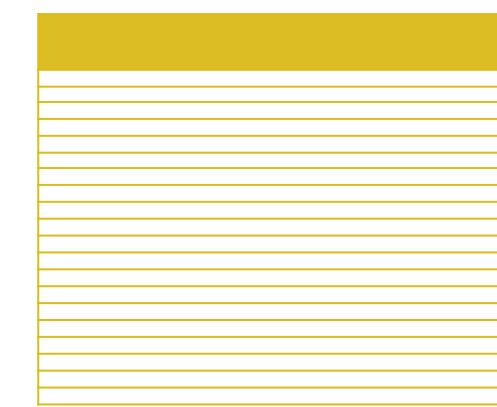
Train



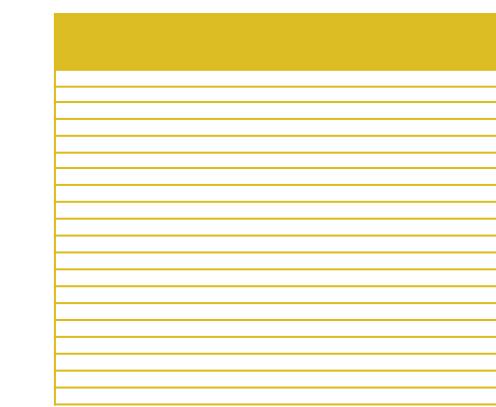
Test



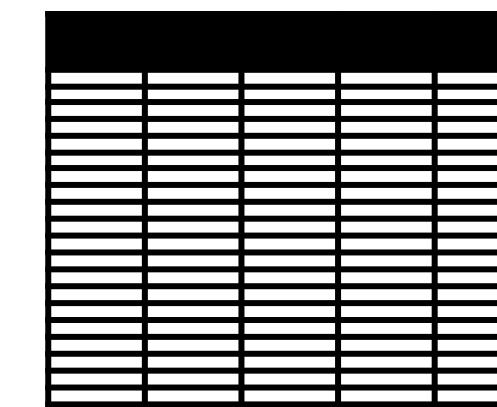
B)



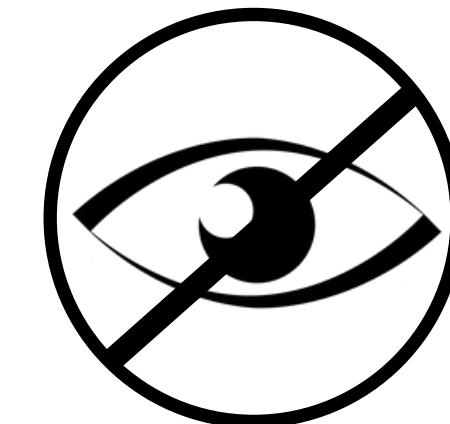
Train



Validate



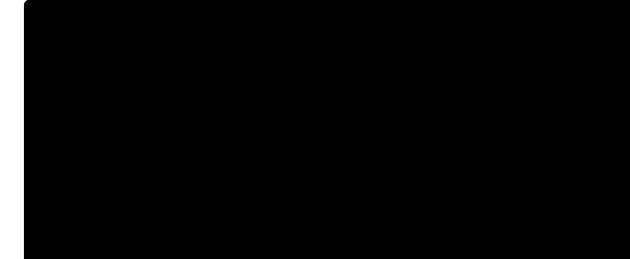
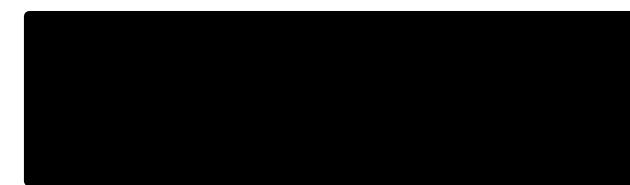
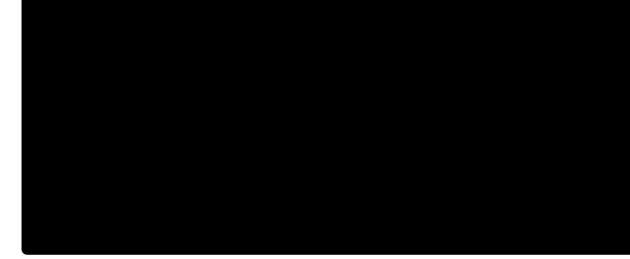
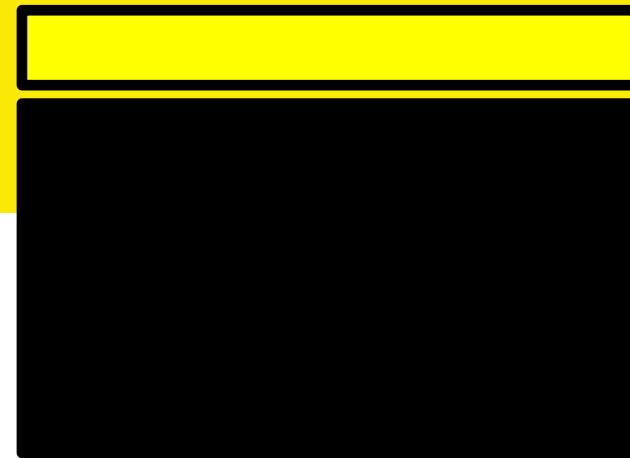
Test



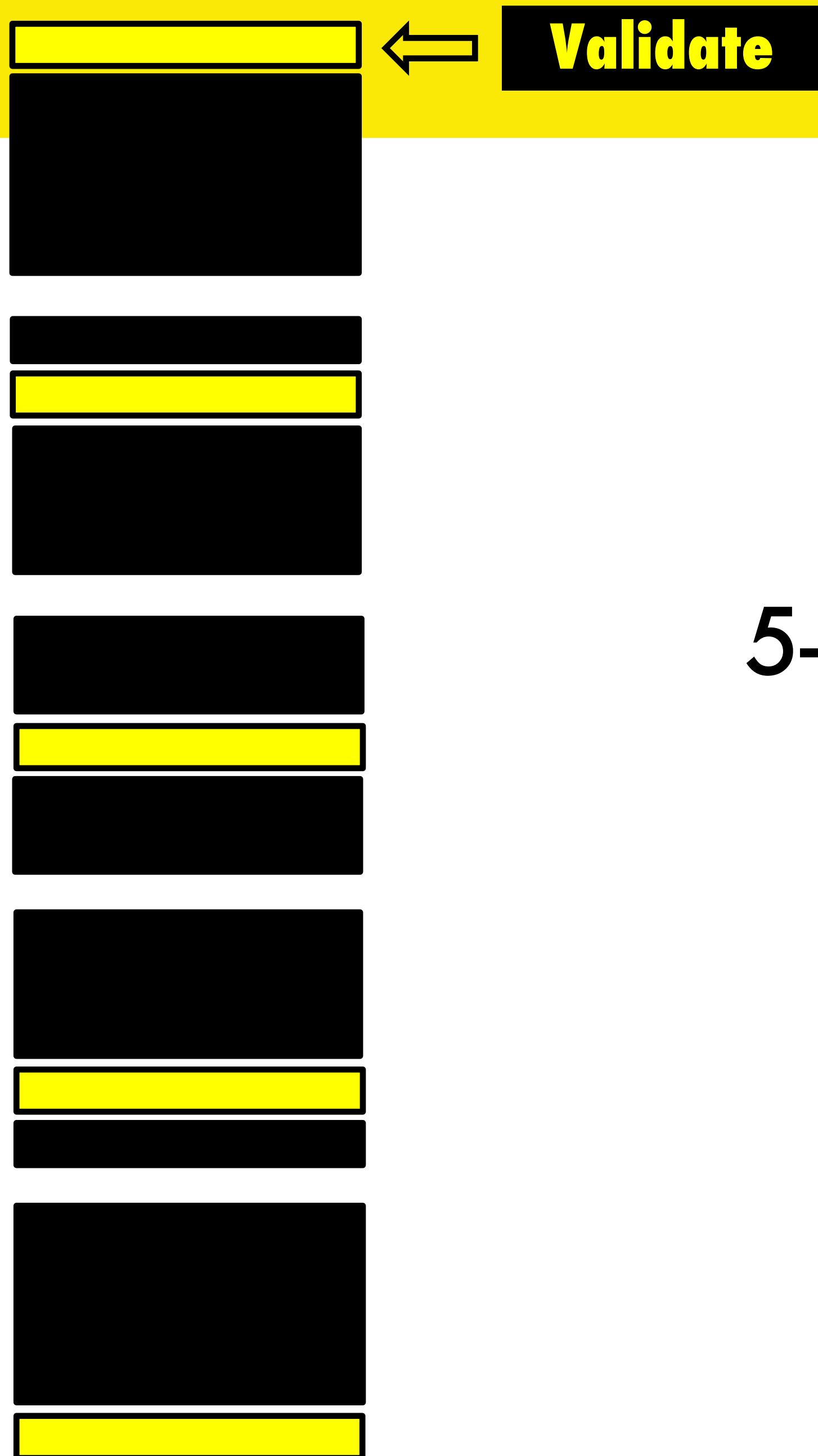
Model Selection

# With K-Fold Cross Validation

**5-fold cross validation  
for each algo**



**5-fold cross validation  
for each algo**



**5-fold cross validation  
for each algo**



← **Train**

← **Train**



**5-fold cross validation  
for each algo**





**5-fold cross validation  
for each algo**



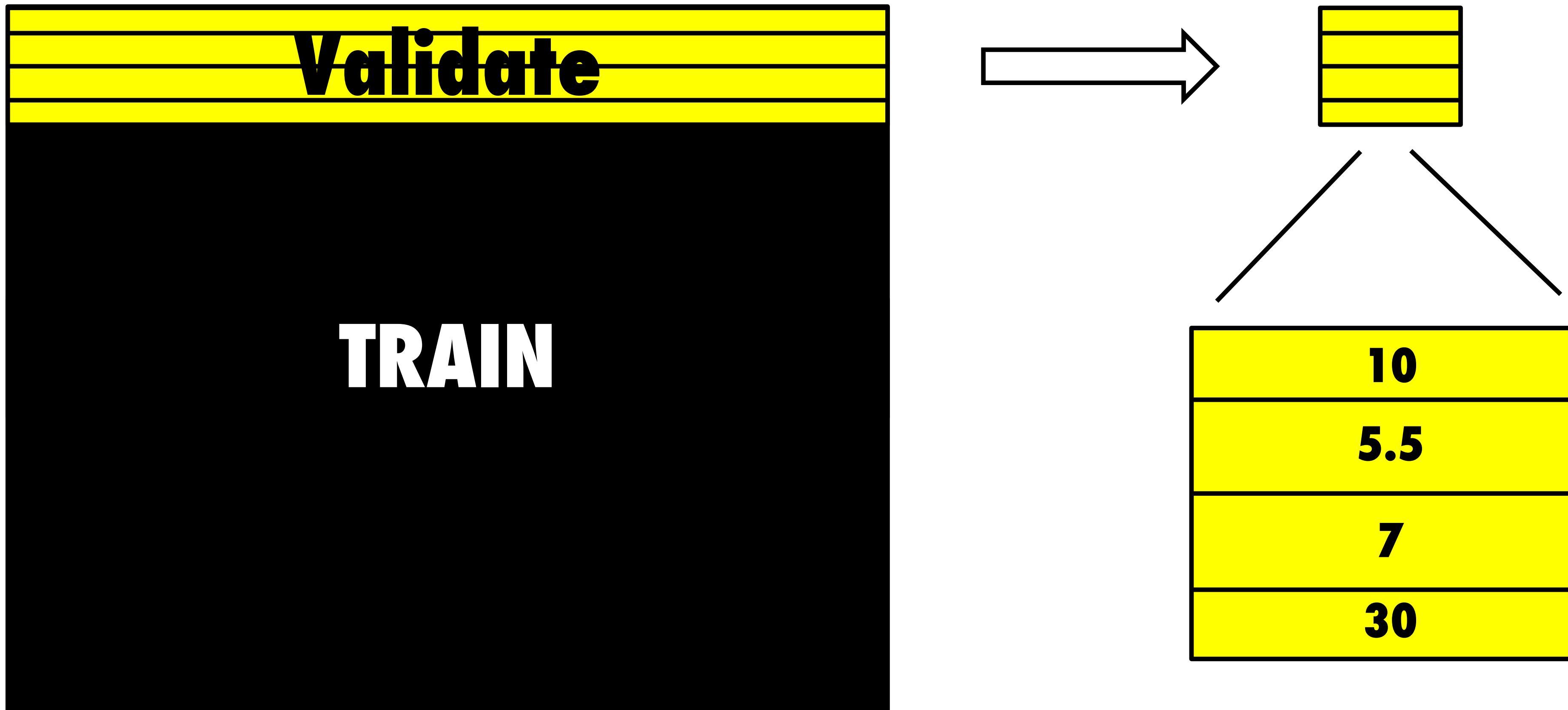
**Continue**

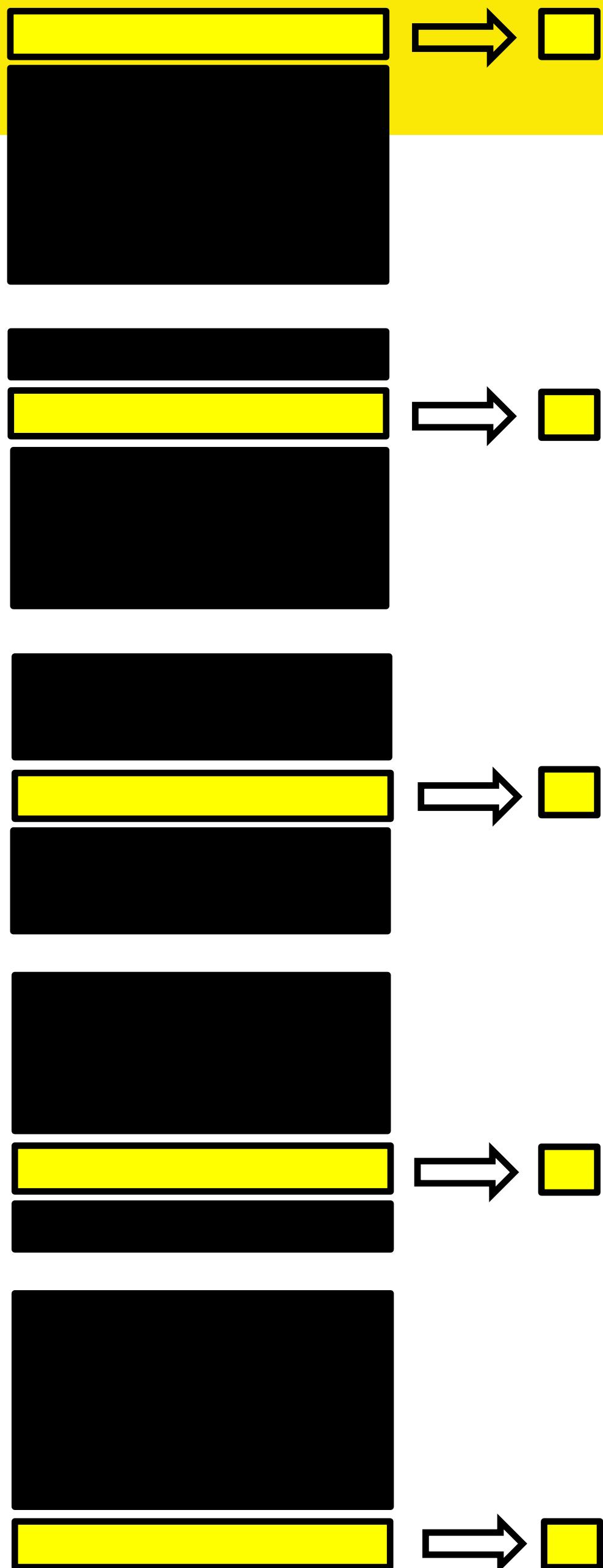
**Process**

**5-fold cross validation  
for each algo**

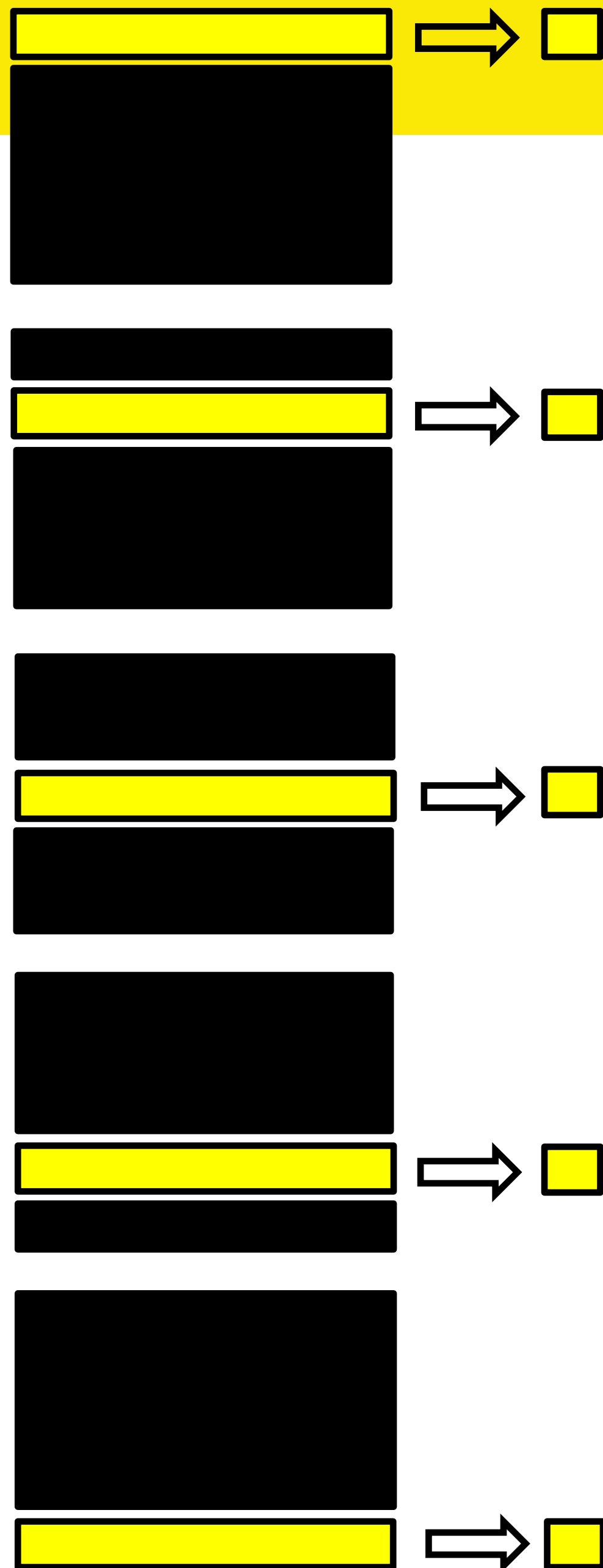
- 
- 
- 
- 
- 
- 
- 
- 
-

# Cross-Validated Prediction Values

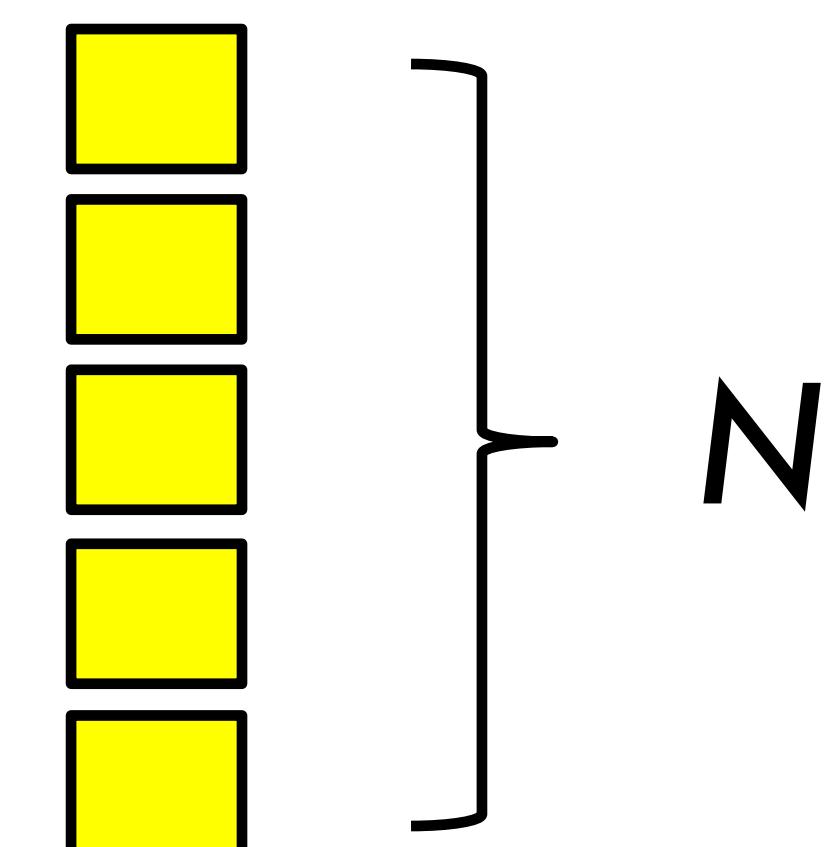




collect the  
cross-validated  
predictions



collect the  
cross-validated  
predictions



# Cross-Validation for Stacking

K=4

Fold : 2

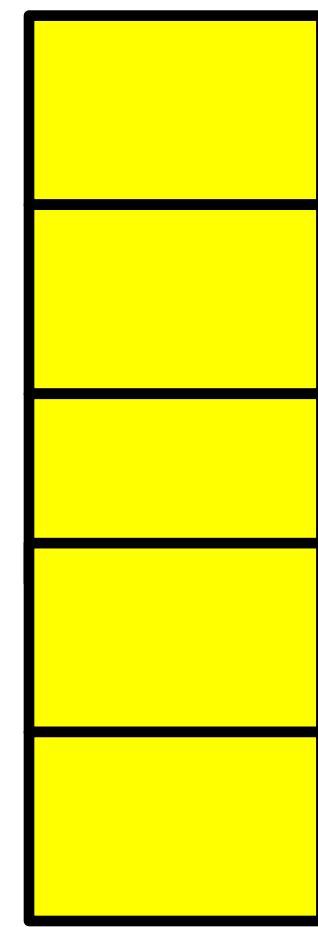
Validate  
store pred →

x0	x1	x2	x3	y
0.94	0.27	0.80	0.34	1
0.02	0.22	0.17	0.84	0
0.83	0.11	0.23	0.42	1
0.74	0.26	0.03	0.41	0
0.08	0.29	0.76	0.37	0
0.71	0.76	0.43	0.95	1
0.08	0.72	0.97	0.04	0
0.84	0.79	0.89	0.05	1

Train

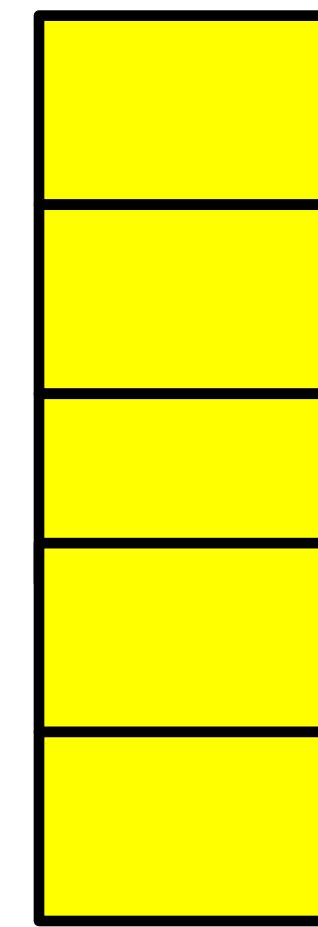
pred
0.96
0.03
0.90
0.02
0.03
0.07
0.08
0.90

**GLM**



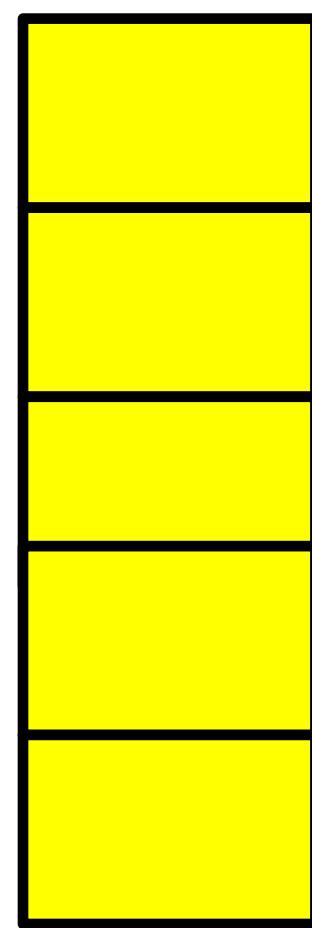
**results for each  
algorithm**

DRF



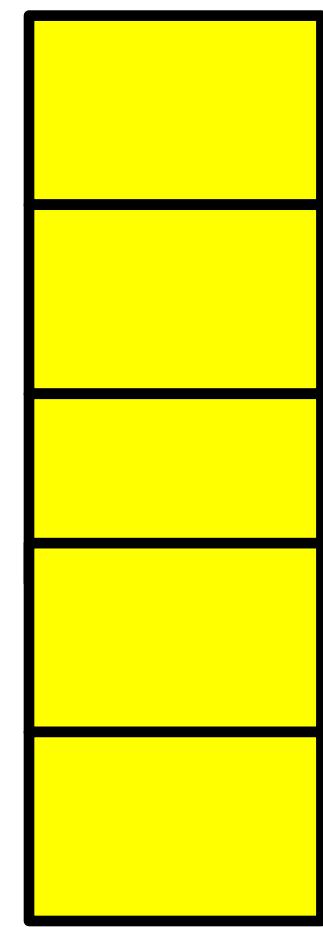
results for each  
algorithm

XRT

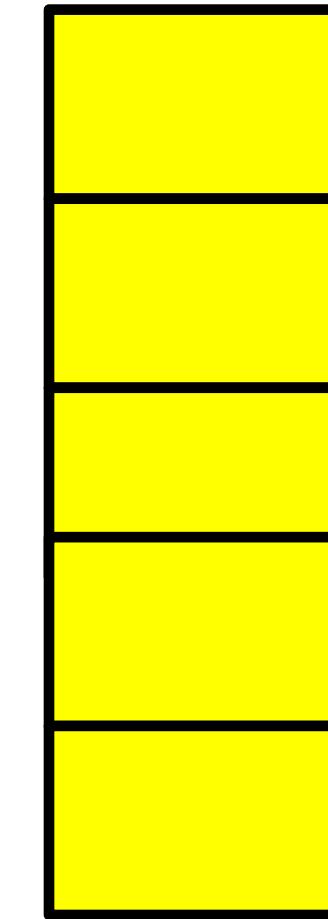
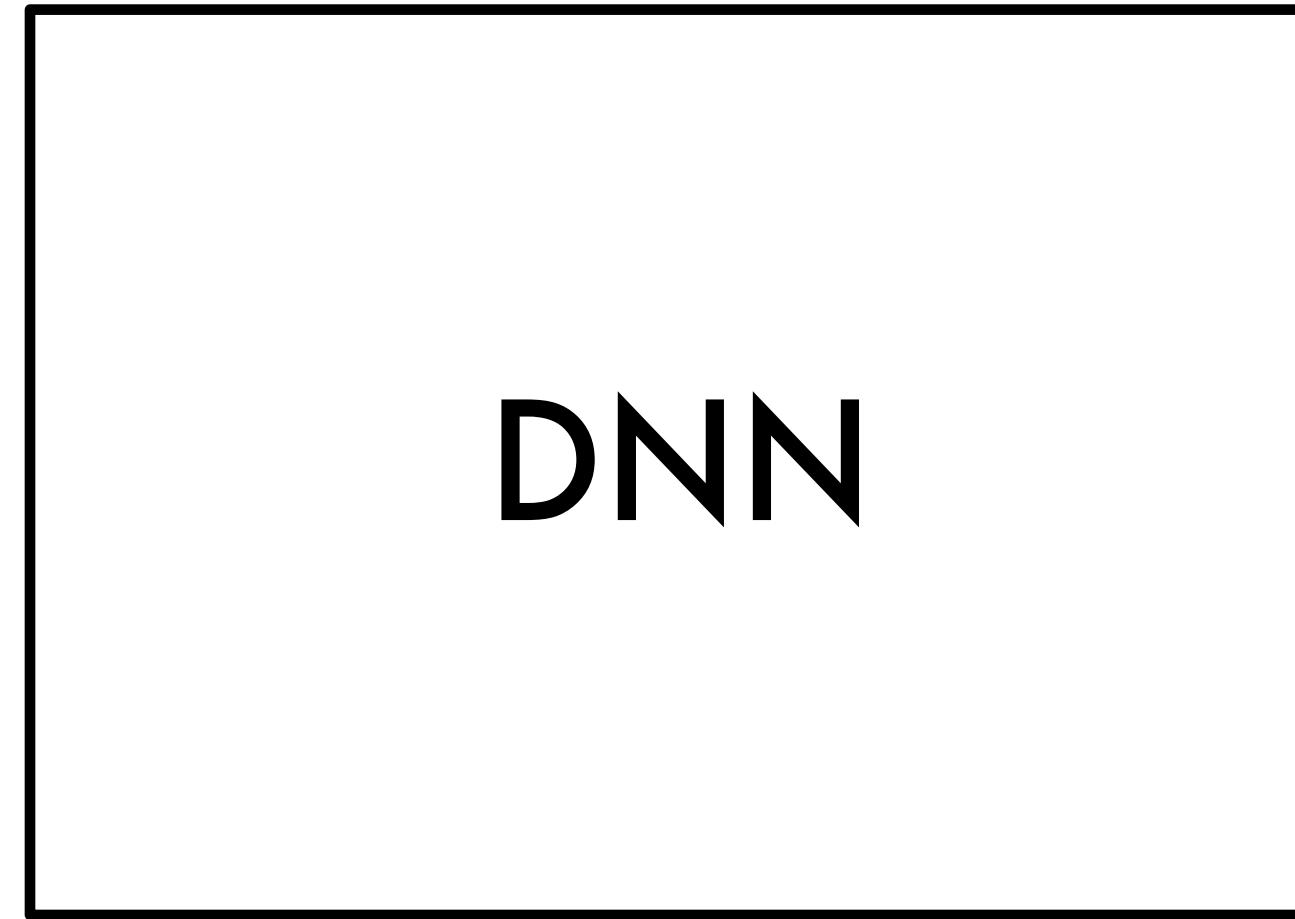


results for each  
algorithm

**GBM**



**results for each  
algorithm**

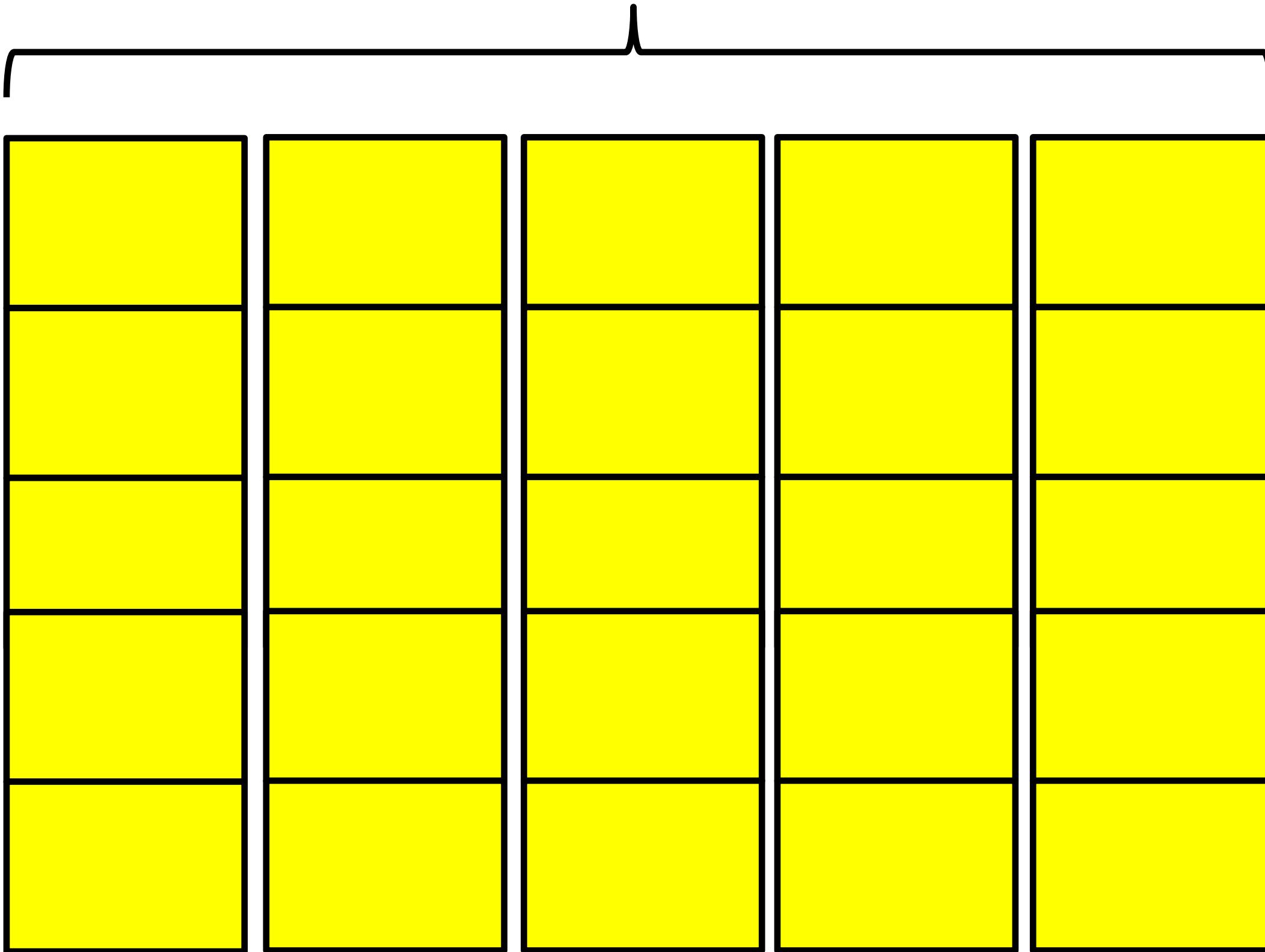


**results for each  
algorithm**

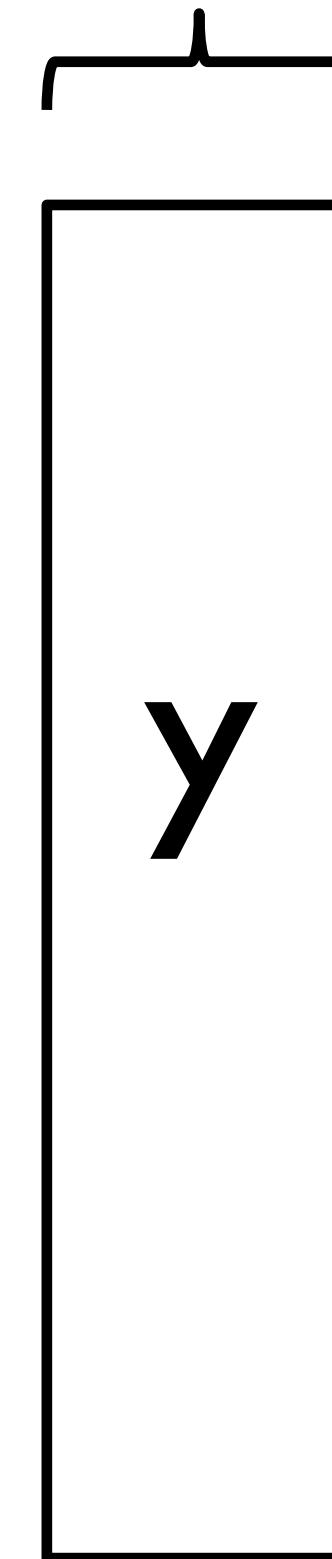
# Stacked Ensemble Steps

1. Specify original dataset for training
2. Specify algorithms with which to train
3. Stack each algorithm's cross-validated predictions
4. Train final model on cross-validated results

# Base Learners' CV Predictions

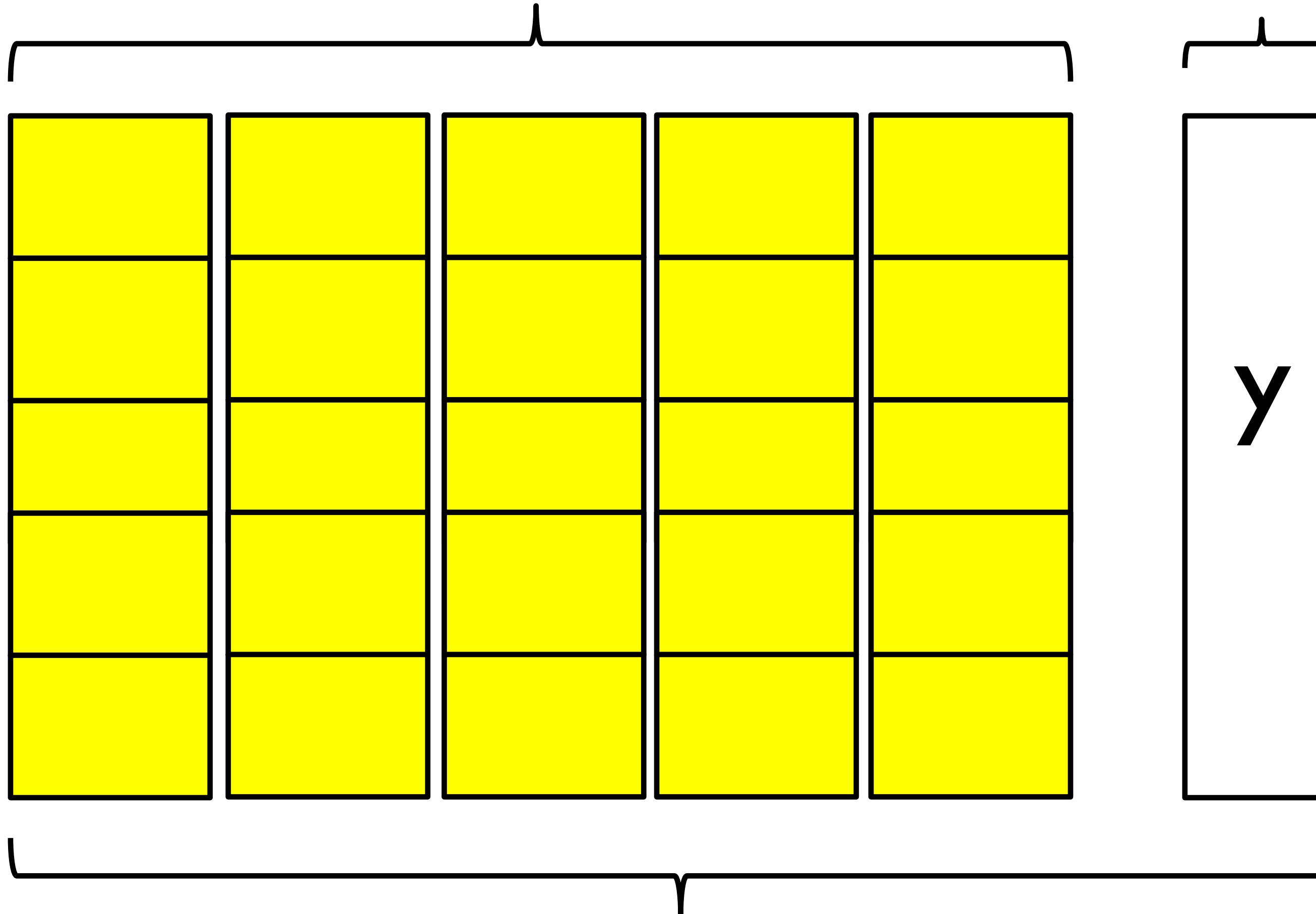


# Original Target

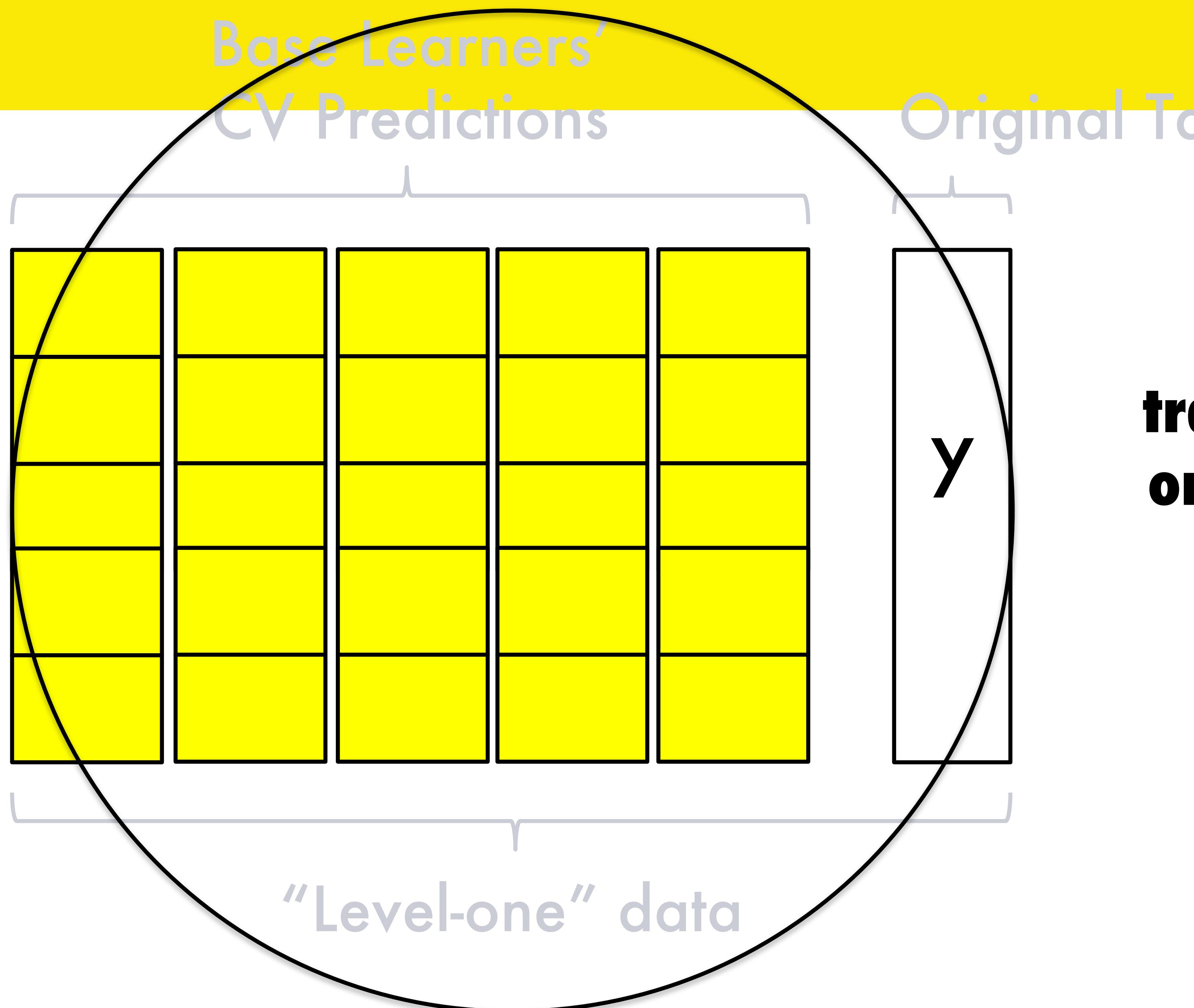


**Base Learners'  
CV Predictions**

**Original Target**



**"Level-one" data**



**train metalearner  
on level-one data**



train **metalearner**  
on level-one data

# Random Grid Search & Stacking

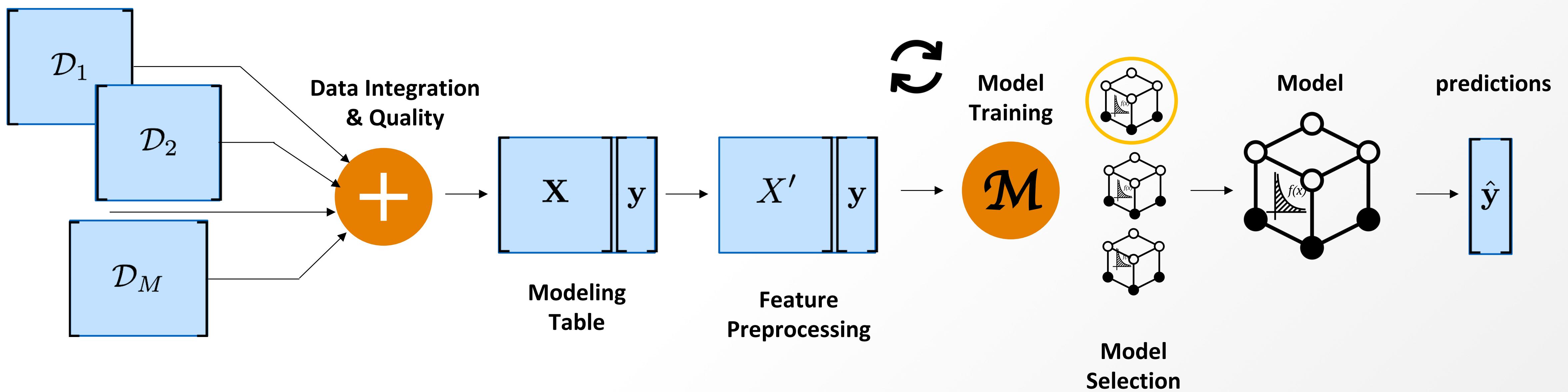
- Random Grid Search combined with Stacked Ensembles is a powerful combination.
- Ensembles perform particularly well if the models they are based on (1) are individually strong, and (2) make uncorrelated errors.
- Stacking uses a second-level metalearning algorithm to find the optimal combination of base learners.

# Agenda: Keeping Track

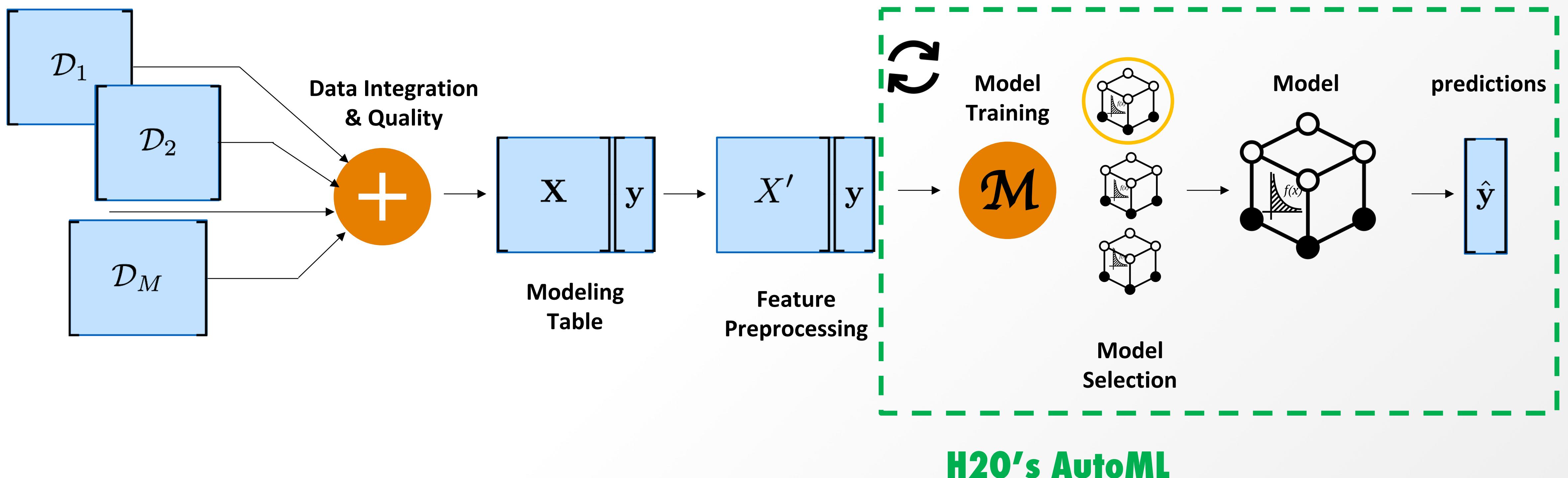
## Aspects of Automatic Machine learning:

- Feature Preprocessing
- Feature Engineering
- Hyperparameter Search
- Ensembles
- **Up Next: H2O's AutoML**

# Updated Machine Learning Pipeline



# Updated Machine Learning Pipeline



# H2O AutoML (current release)

## Data Preprocessing

- Imputation, one-hot encoding, standardization
  - Feature selection and/or feature extraction (e.g. PCA)
  - Count/Label/Target encoding of categorical features
- 

## Model Generation

- Cartesian grid search or random grid search
  - Bayesian Hyperparameter Optimization
  - Individual models can be tuned using a validation set
- 

## Ensembles

- Ensembles often out-perform individual models:
- Stacking / Super Learning (Wolpert, Breiman)
- Ensemble Selection (Caruana)

# H2O AutoML

- Basic data pre-processing (as in all H2O algos).
- Trains a random grid of GBMs, DNNs, GLMs, etc. using a carefully chosen hyper-parameter space
- Individual models are tuned using a validation set.
- Two Stacked Ensembles are trained (“All Models” ensemble & a lightweight “Best of Family” ensemble).
- Returns a sorted “Leaderboard” of all models.

Available in H2O >=3.14

# H2O AutoML Interface

- The H2O AutoML interface is designed to have as few parameters as possible so that all the user needs to do is point to their dataset, identify the response column and optionally specify a time constraint or limit on the number of total models trained.
- In both the R and Python API, AutoML uses the same data-related arguments, **x**, **y**, **training\_frame**, **validation\_frame**, as the other H2O algorithms.
- Most of the time, all you'll need to do is specify the data arguments.
- You can then configure values for **max\_runtime\_secs** and/or **max\_models** to set explicit time or number-of-model limits on your run

# H2O AutoML Interface

- **Required Parameters**
  - **Required Data Parameters**
    - **y**: This argument is the name (or index) of the response column.
    - **training\_frame**: Specifies the training set.
  - **Required Stopping Parameters**
    - One of the following stopping strategies (time or number-of-model based) must be specified. When both options are set, then the AutoML run will stop as soon as it hits one of either of these limits.
      - **max\_runtime\_secs**: This argument controls how long the AutoML run will execute for. This defaults to 3600 seconds (1 hour).
      - **max\_models**: Specify the maximum number of models to build in an AutoML run, excluding the Stacked Ensemble models. Defaults to NULL/None

# H2O AutoML Interface

- **Optional Parameters**
  - **Optional Data Parameters**
    - **x**: A list/vector of predictor column names or indexes. This argument only needs to be specified if the user wants to exclude columns from the set of predictors. If all columns (other than the response) should be used in prediction, then this does not need to be set.
    - **validation\_frame**: This argument is used to specify the validation frame used for early stopping of individual models and early stopping of the grid searches (unless **max\_models** or **max\_runtime\_secs** overrides metric-based early stopping).
    - **leaderboard\_frame**: This argument allows the user to specify a particular data frame use to score & rank models on the leaderboard. This frame will not be used for anything besides leaderboard scoring. If a leaderboard frame is not specified by the user, then the leaderboard will use cross-validation metrics instead (or if cross-validation is turned off by setting nfolds = 0, then a leaderboard frame will be generated automatically from the validation frame (if provided) or the training frame).
    - **fold\_column**: Specifies a column with cross-validation fold index assignment per observation. This is used to override the default, randomized, 5-fold cross-validation scheme for individual models in the AutoML run.
    - **weights\_column**: Specifies a column with observation weights. Giving some observation a weight of zero is equivalent to excluding it from the dataset; giving an observation a relative weight of 2 is equivalent to repeating that row twice. Negative weights are not allowed.
    - **ignored\_columns**: (Optional, Python only) Specify the column or columns (as a list/vector) to be excluded from the model. This is the converse of the x argument.

# H2O AutoML Interface

- **Optional Miscellaneous Parameters**
  - **nfolds**: Number of folds for k-fold cross-validation of the models in the AutoML run. Defaults to 5. Use 0 to disable cross-validation; this will also disable Stacked Ensembles (thus decreasing the overall best model performance).
  - **stopping\_metric**: Specifies the metric to use for early stopping of the grid searches and individual models. Defaults to "AUTO". The available options are:
    - AUTO: This defaults to logloss for classification, deviance for regression
    - deviance
    - logloss
    - mse
    - rmse
    - mae
    - rmsle
    - auc
    - lift\_top\_group
    - misclassification
    - mean\_per\_class\_error

# H2O AutoML Interface

- **Optional Miscellaneous Parameters cont.**

- **stopping\_tolerance:** This option specifies the relative tolerance for the metric-based stopping criterion to stop a grid search and the training of individual models within the AutoML run. This value defaults to 0.001 if the dataset is at least 1 million rows; otherwise it defaults to a bigger value determined by the size of the dataset and the non-NA-rate. In that case, the value is computed as  $1/\sqrt{nrows * \text{non-NA-rate}}$ .
- **stopping\_rounds:** This argument is used to stop model training when the stopping metric (e.g. AUC) doesn't improve for this specified number of training rounds, based on a simple moving average. In the context of AutoML, this controls early stopping both within the random grid searches as well as the individual models. Defaults to 3 and must be an non-negative integer. To disable early stopping altogether, set this to 0.
- **seed:** Integer. Set a seed for reproducibility. AutoML can only guarantee reproducibility if **max\_models** is used because **max\_runtime\_secs** is resource limited, meaning that if the available compute resources are not the same between runs, AutoML may be able to train more models on one run vs another. Defaults to NULL/None.
- **project\_name:** Character string to identify an AutoML project. Defaults to NULL/None, which means a project name will be auto-generated based on the training frame ID. More models can be trained and added to an existing AutoML project by specifying the same project name in multiple calls to the AutoML function (as long as the same training frame is used in subsequent runs).
- **exclude\_algos:** List/vector of character strings naming the algorithms to skip during the model-building phase. An example use is `exclude_algos = ["GLM", "DeepLearning", "DRF"]` in Python or `exclude_algos = c("GLM", "DeepLearning", "DRF")` in R. Defaults to None/NULL, which means that all appropriate H2O algorithms will be used, if the search stopping criteria allow. The algorithm names are:
  - GLM
  - DeepLearning
  - GBM
  - DRF (This includes both the Random Forest and Extremely-Randomized Trees (XRT) models. Refer to the Extremely Randomized Trees section in the DRF chapter and the `histogram_type` parameter description for more information.)
  - StackedEnsemble

# H2O AutoML Interface

- **Auto-Generated Frames**
  - If the user doesn't specify a **validation\_frame**, then one will be created automatically by randomly partitioning the training data. The validation frame is required for early stopping of the individual algorithms, the grid searches and the AutoML process itself.
  - By default, AutoML uses cross-validation for all models, and therefore we can use cross-validation metrics to generate the leaderboard. If the **leaderboard\_frame** is explicitly specified by the user, then that frame will be used to generate the leaderboard metrics instead of using cross-validation metrics.
  - For cross-validated AutoML, when the user specifies:
    - **training**: The **training\_frame** is split into training (80%) and validation (20%).
    - **training + leaderboard**: The **training\_frame** is split into training (80%) and validation (20%).
    - **training + validation**: Leave frames as-is.
    - **training + validation + leaderboard**: Leave frames as-is.
  - If not using cross-validation (by setting **nfolds** = 0) in AutoML, then we need to make sure there is a test frame (aka. the "leaderboard frame") to score on because cross-validation metrics will not be available. So when the user specifies:
    - **training**: The **training\_frame** is split into training (80%), validation (10%) and leaderboard/test (10%).
    - **training + leaderboard**: The **training\_frame** is split into training (80%) and validation (20%). Leaderboard frame as-is.
    - **training + validation**: The **validation\_frame** is split into validation (50%) and leaderboard/test (50%). Training frame as-is.
    - **training + validation + leaderboard**: Leave frames as-is.

# H2O AutoML in Flow GUI

H2O FLOW  Flow ▾ Cell ▾ Data ▾ Model ▾ Score ▾ Admin ▾ Help ▾

Untitled Flow



CS

runAutoML

26ms

## Run AutoML

Project Name:

Training Frame:

(Select) 

Seed:

Max models to build:

Max Run Time (sec):

Early stopping metric:

AUTO 

Early stopping rounds:

Stopping Tolerance:

nfold:

 Build Model

# H2O AutoML in R

## Example

```
library(h2o)  
h2o.init()  
  
train <- h2o.importFile("train.csv")  
  
aml <- h2o.automl(y = "response_colname",  
                    training_frame = train,  
                    max_runtime_secs = 600)  
  
lb <- aml@leaderboard
```

# H2O AutoML in Python

## Example

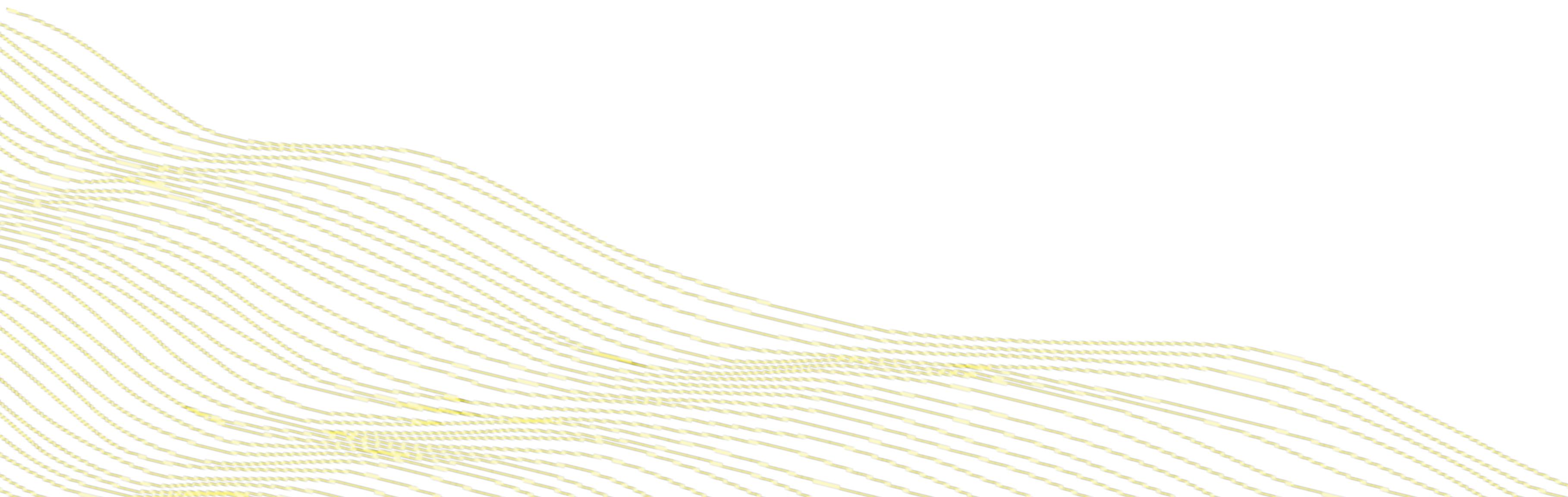
```
import h2o  
  
from h2o.automl import H20AutoML  
  
h2o.init()  
  
train = h2o.import_file("train.csv")  
  
aml = H20AutoML(max_runtime_secs = 600)  
aml.train(y = "response_colname",  
          training_frame = train)  
  
lb = aml.leaderboard
```

# H2O AutoML Leaderboard

model_id	auc	logloss
StackedEnsemble_AllModels_0_AutoML_20171121_012135	0.788321	0.554019
StackedEnsemble_BestOfFamily_0_AutoML_20171121_012135	0.783099	0.559286
GBM_grid_0_AutoML_20171121_012135_model_1	0.780554	0.560248
GBM_grid_0_AutoML_20171121_012135_model_0	0.779713	0.562142
GBM_grid_0_AutoML_20171121_012135_model_2	0.776206	0.564970
GBM_grid_0_AutoML_20171121_012135_model_3	0.771026	0.570270
DRF_0_AutoML_20171121_012135	0.734653	0.601520
XRT_0_AutoML_20171121_012135	0.730457	0.611706
GBM_grid_0_AutoML_20171121_012135_model_4	0.727098	0.666513
GLM_grid_0_AutoML_20171121_012135_model_0	0.685211	0.635138

Example Leaderboard for binary classification

# AutoML Pro Tips!



**Before you press the “red button”**



# AutoML Pro Tips: Input Frames

- Don't use `leaderboard_frame` unless you really need to; use cross-validation metrics to generate the leaderboard instead (default).
- If you only provide `training_frame`, it will chop off 20% of your data for a validation set to be used in early stopping. To control this proportion, you can split the data yourself and pass a `validation_frame` manually.

# AutoML Pro Tips: Exclude Algos

- If you have sparse, wide data (e.g. text), use the `exclude_algos` argument to turn off the tree-based models (GBM, RF).
- If you want tree-based algos only, turn off GLM and DNNs via `exclude_algos`.

# AutoML Pro Tips: Time & Model Limits

- AutoML will stop after 1 hour unless you change `max_runtime_secs`.
- Running with `max_runtime_secs` is not reproducible since available resources on a machine may change from run to run. Set `max_runtime_secs` to a big number (e.g. `9999999999`) and use `max_models` instead.

# AutoML Pro Tips: Cluster memory

- Reminder: All H2O models are stored in H2O Cluster memory.
- Make sure to give the H2O Cluster a lot of memory if you're going to create hundreds or thousands of models.
- e.g. `h2o.init(max_mem_size = "80G")`

**After you press the “red button”**



# AutoML Pro Tips: Early Stopping

- If you're expecting more models than are listed in the leaderboard, or the run is stopping earlier than `max_runtime_secs`, this is a result of the default "early stopping" settings.
- To allow more time, increase the number of `stopping_rounds` and/or decrease value of `stopping_tolerance`.

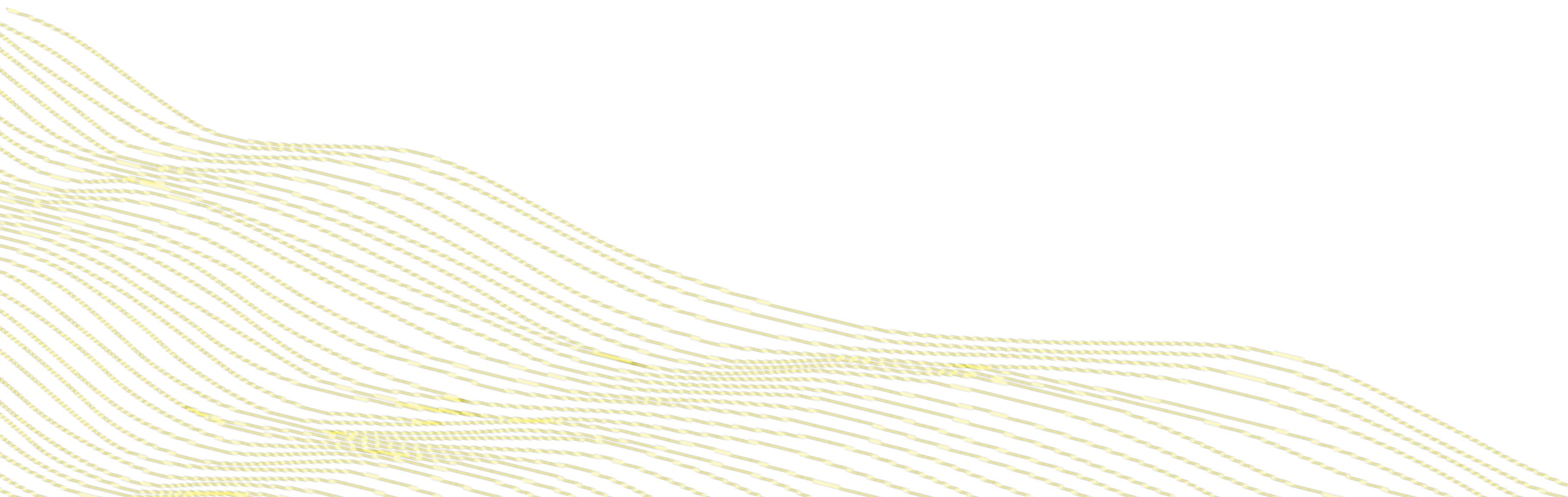
# AutoML Pro Tips: Add More Models

- If you want to add (train) more models to an existing AutoML project, just make sure to use the same training set and `project_name`.
- If you set the same seed twice it will give you identical models as the first run (not useful), so change the seed or leave it unset.

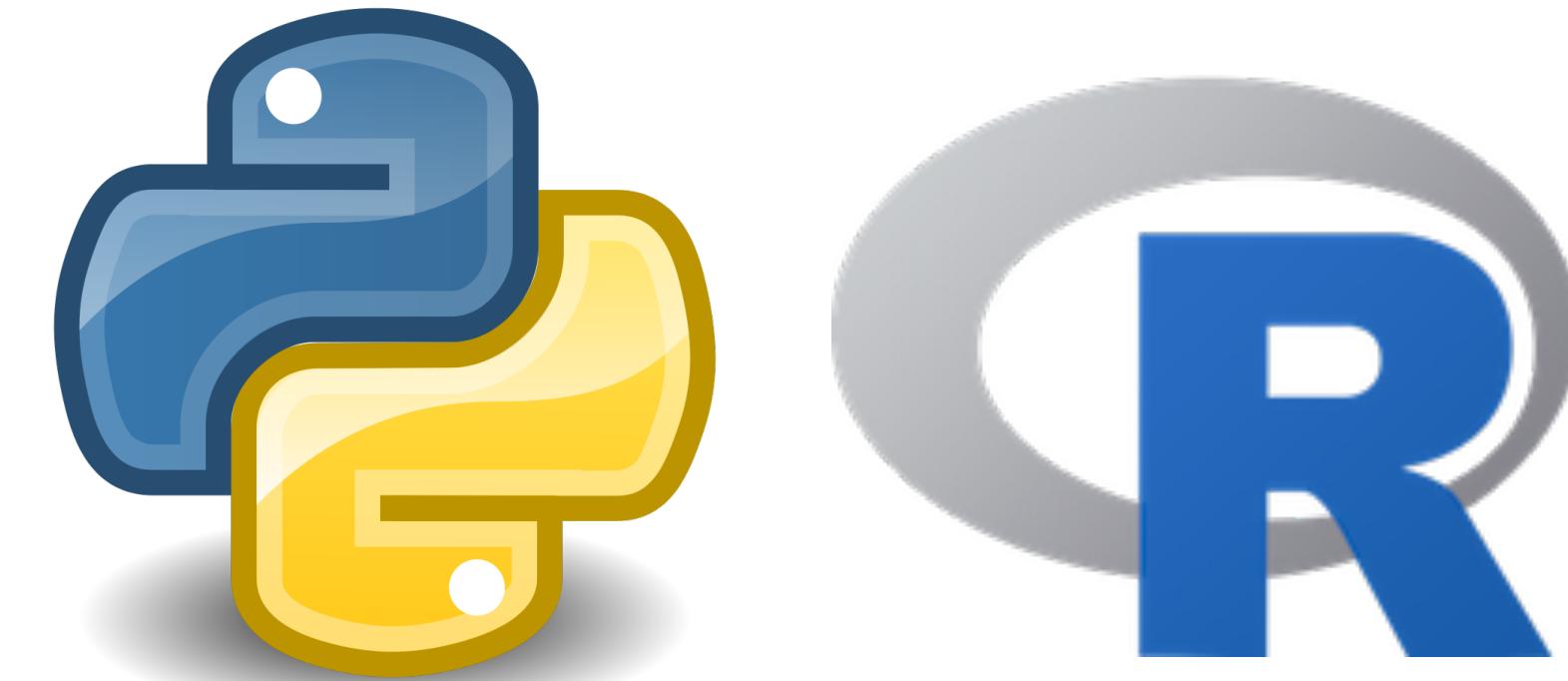
# AutoML Pro Tips: Saving Models

- You can save any of the individual models created by the AutoML run. The model ids are listed in the leaderboard.
- If you're taking your leader model (probably a Stacked Ensemble) to production, we'd recommend using "Best of Family" since it only contains 5 models and gets most of the performance of the "All Models" ensemble.

# H2O AutoML Demos



# H2O AutoML Demos



Code available here

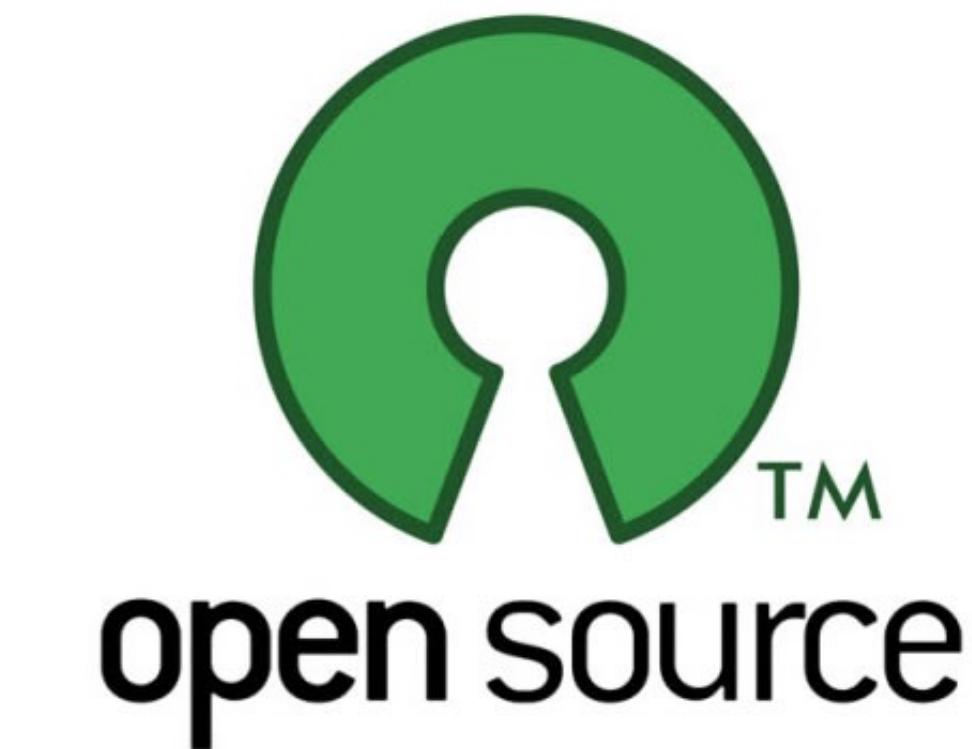
<https://github.com/navdeep-G/sdss-h2o-automl>

# H2O Resources

- Documentation: <http://docs.h2o.ai>
- Tutorials: <https://github.com/h2oai/h2o-tutorials>
- Slidedecks: <https://github.com/h2oai/h2o-meetups>
- Videos: <https://www.youtube.com/user/0xdata>
- Stack Overflow: <https://stackoverflow.com/tags/h2o>
- Google Group: <https://tinyurl.com/h2ostream>
- Gitter: <http://gitter.im/h2oai/h2o-3>
- Events & Meetups: <http://h2o.ai/events>



# Contribute to H2O!



Get in touch over email, Gitter or JIRA.

<https://github.com/h2oai/h2o-3/blob/master/CONTRIBUTING.md>

# Thank you!

@navdeep-G on Github

@Navdeep\_Gill\_ on Twitter

navdeep@h2o.ai