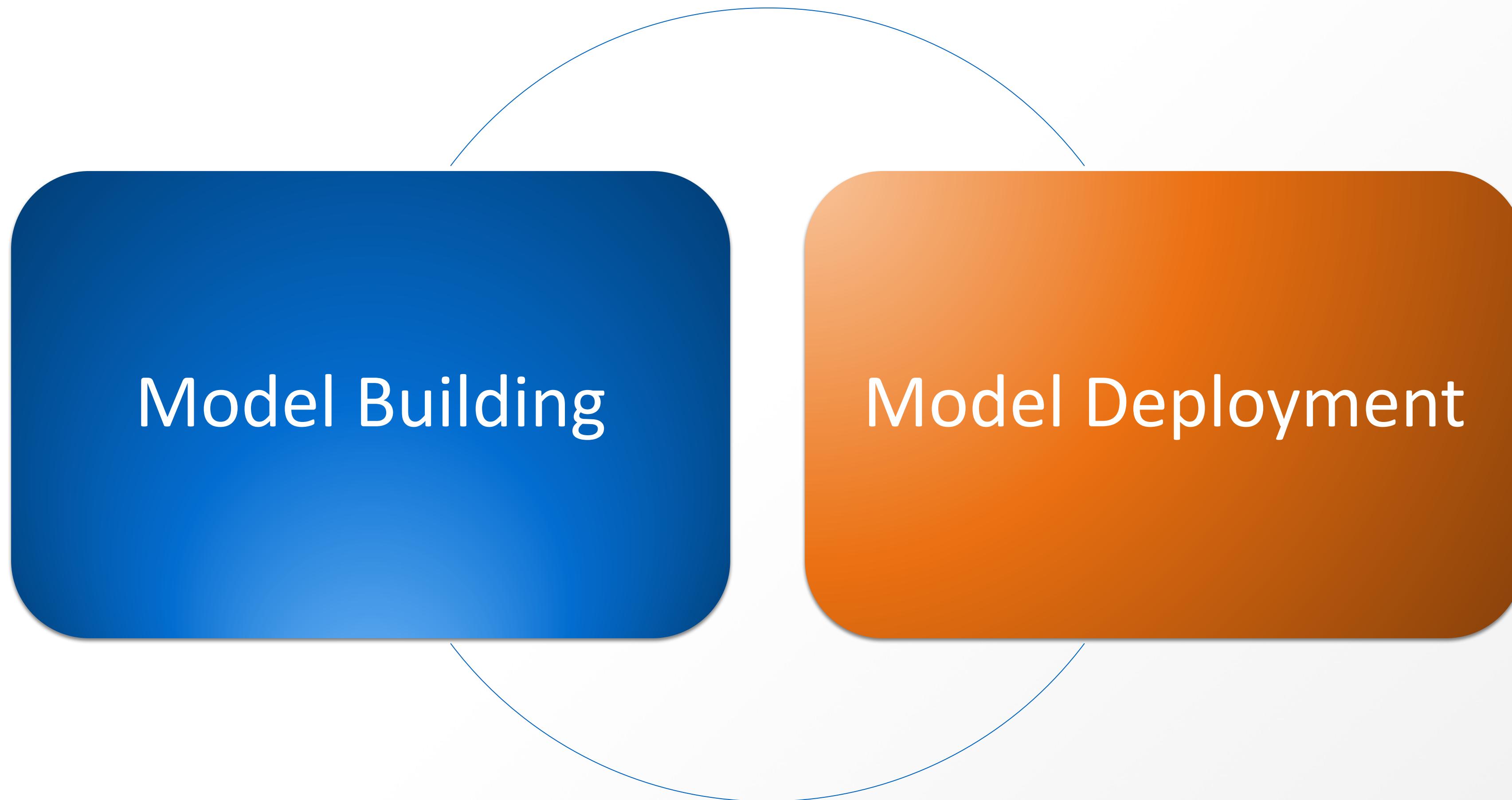


Model Deployment in H2O

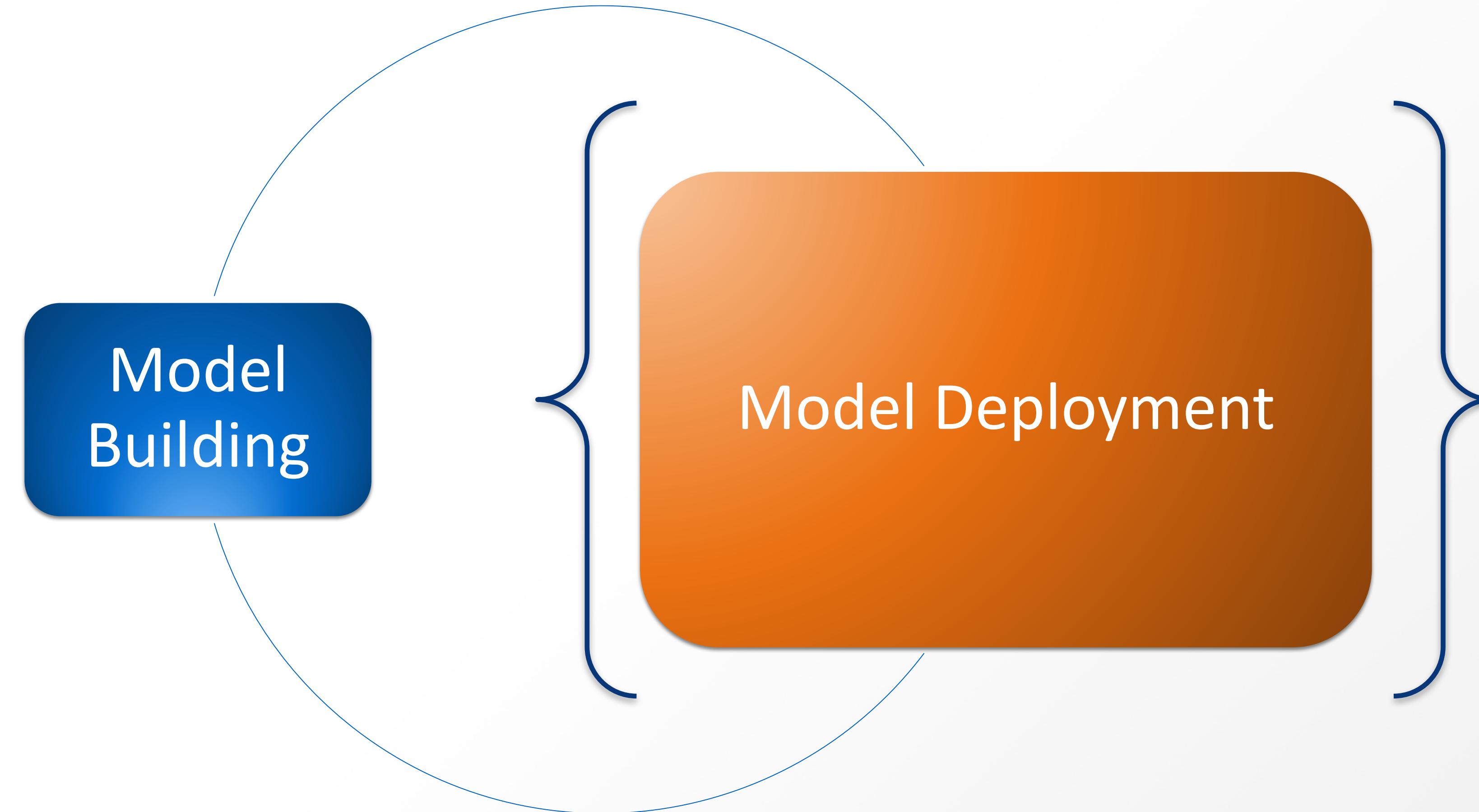
(H2O Models in Production)



Machine Learning Process



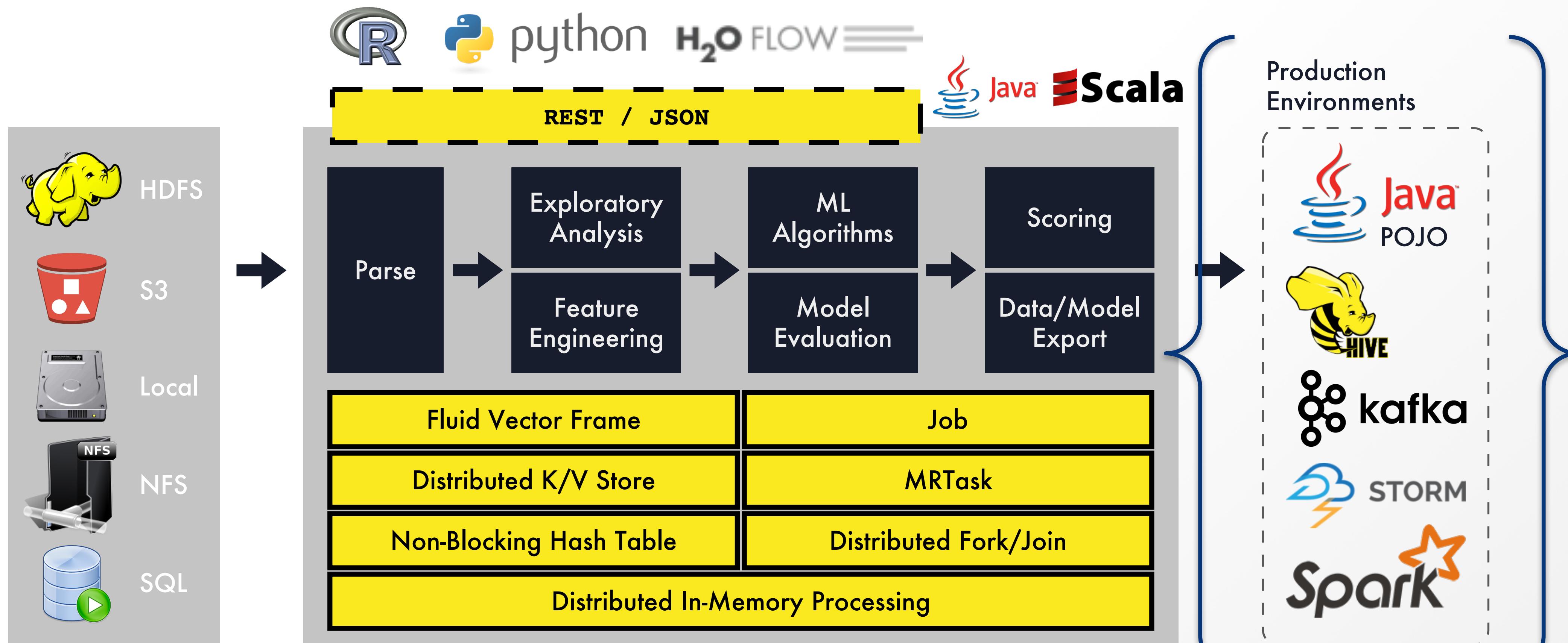
Machine Learning Process



So what is model deployment?



Core H2O Architecture



cloudera Hortonworks
MAPR

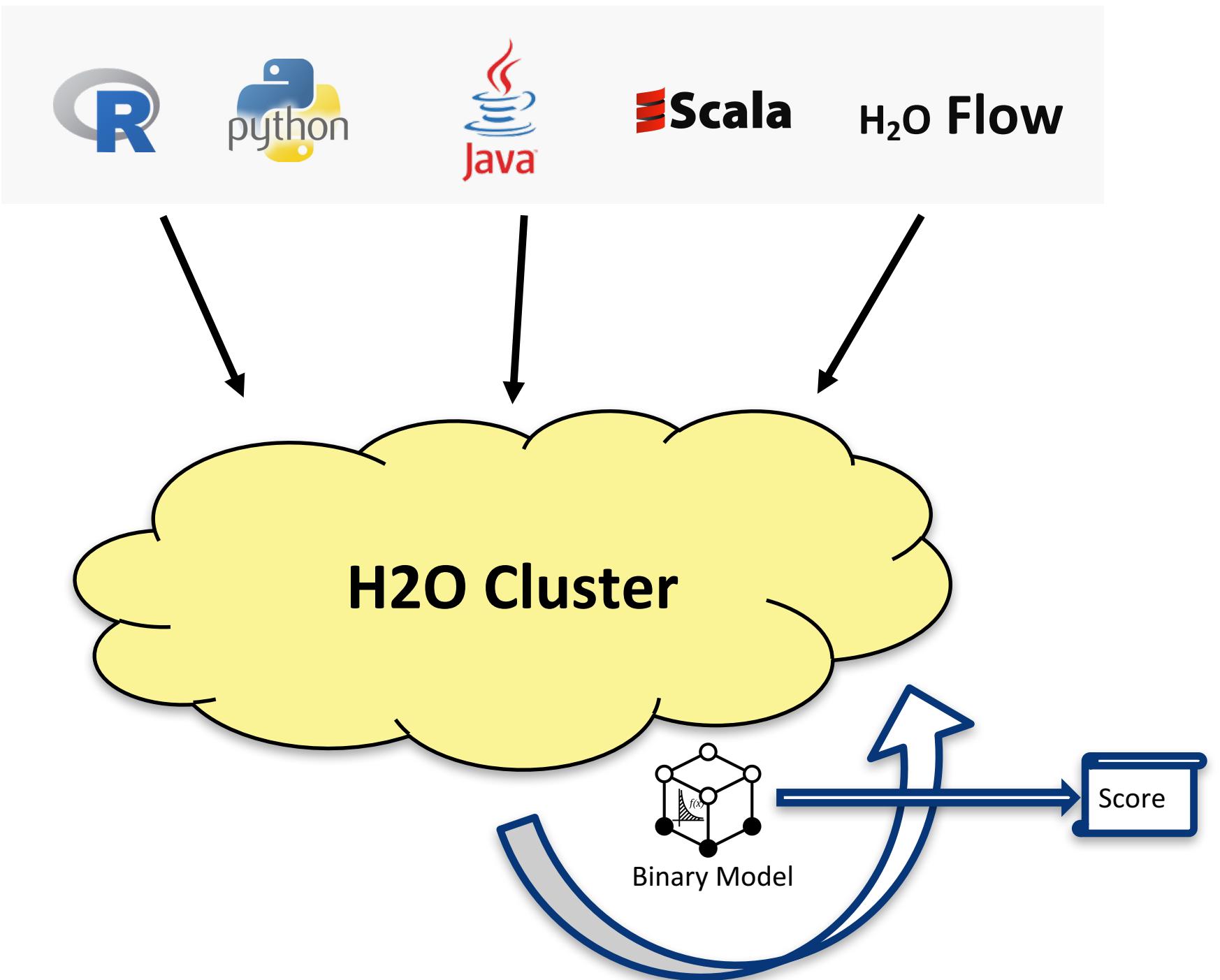
Spark + H₂O
SPARKLING
WATER

H₂O.ai

Model Deployment

- In-Cluster Scoring (Requires H2O cluster)
 - **Binary Model**
- Out-cluster Scoring (Independent to H2O)
 - Plain Old Java Object (POJO)
 - Model Object Optimized (MOJO)
 - Java MOJO
 - C++ MOJO

Machine Learning Model Lifecycle



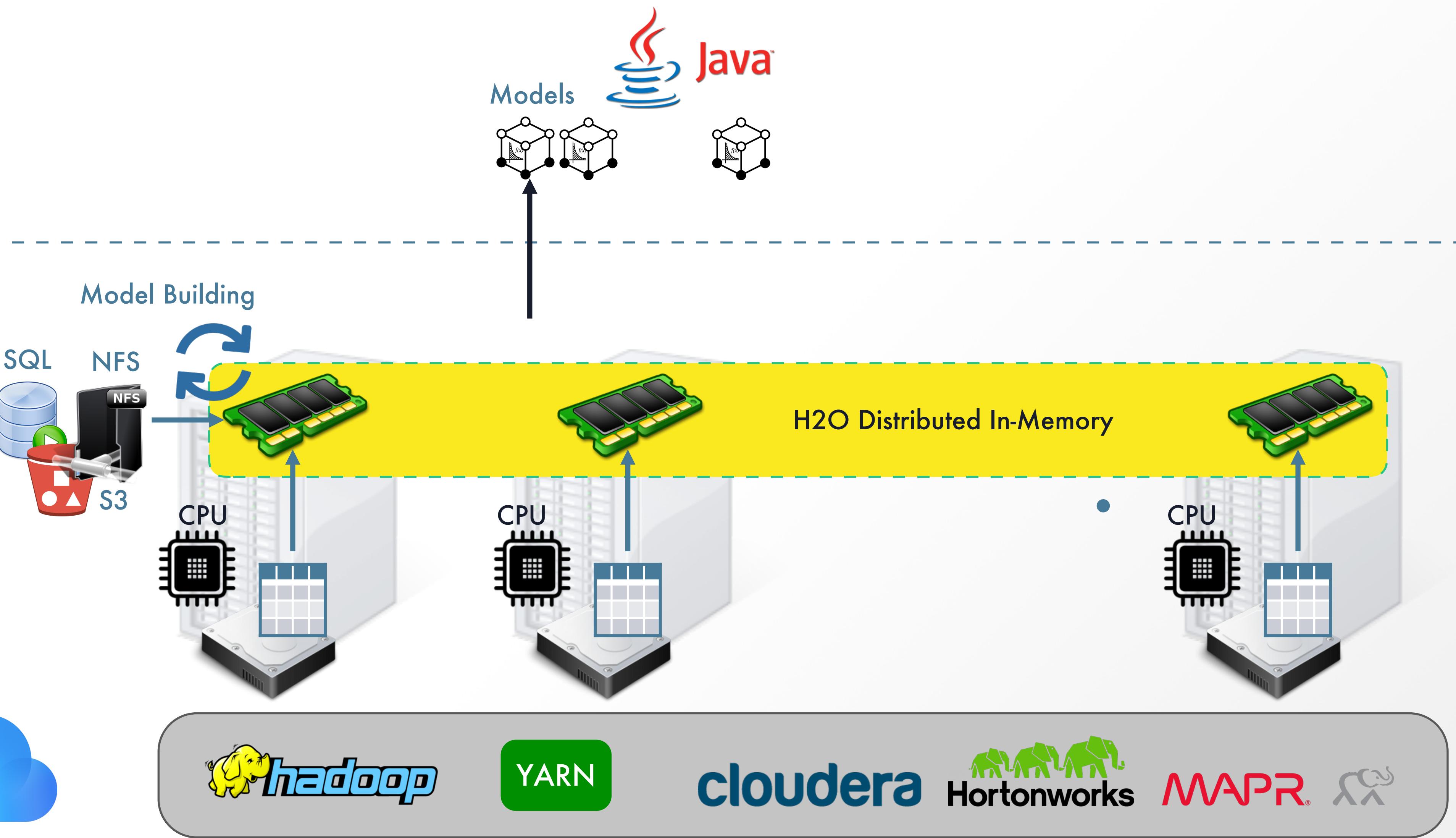
(H₂O can run anywhere: desktop, cloud, on-prem;
Hadoop and Spark environments supported)

Model Training

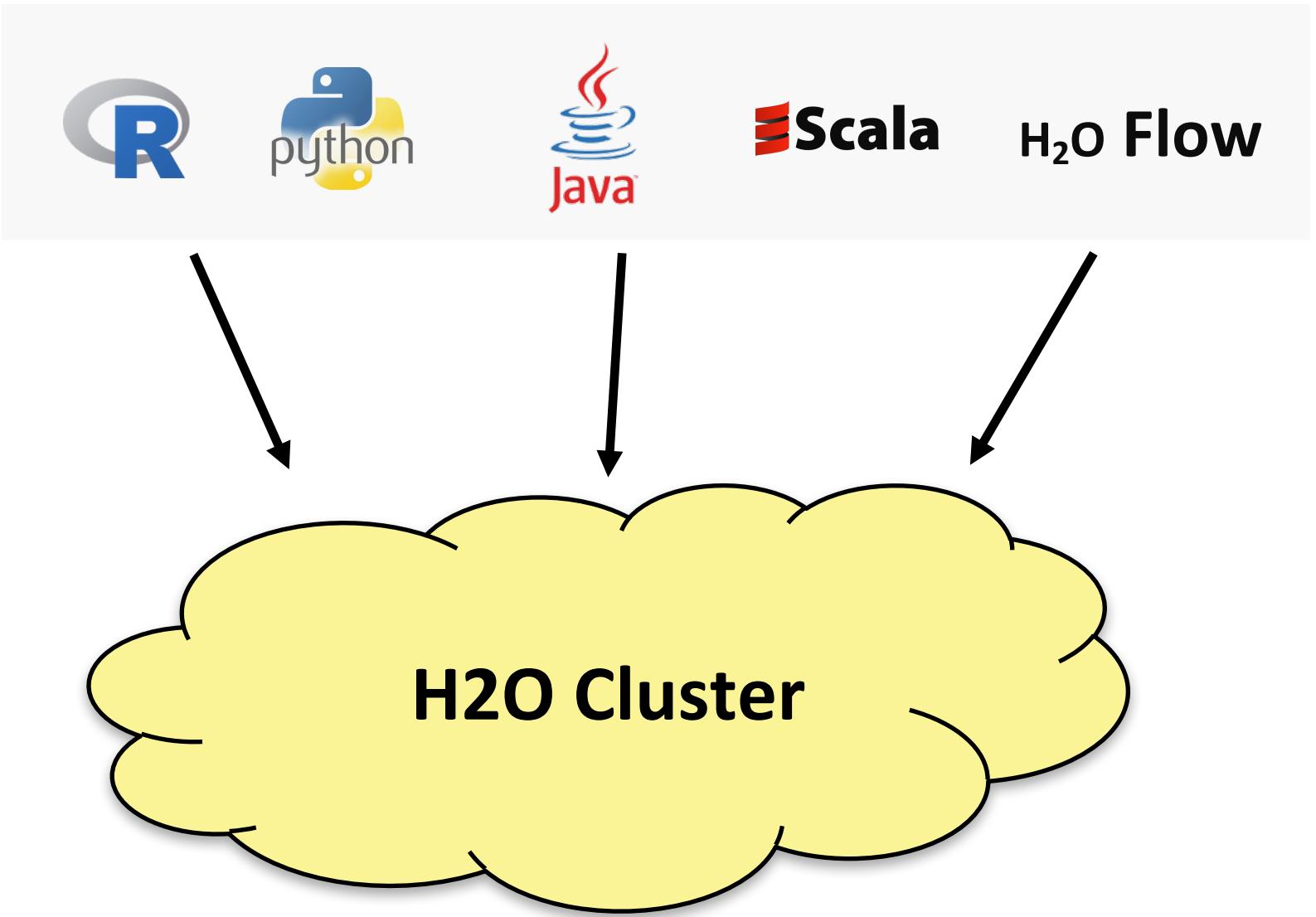
In-Cluster Scoring

- Model Persistence
- R API
 - `h2o.saveModel(object, path, force=FALSE)`
 - `h2o.loadModel(path)`
 - `h2o.predict(object, newdata)`
- Python API
 - `h2o.save_model(model, path, force=False)`
 - `h2o.load_model(path)`
 - `H2OModel.predict(newdata)`

Deployment Code



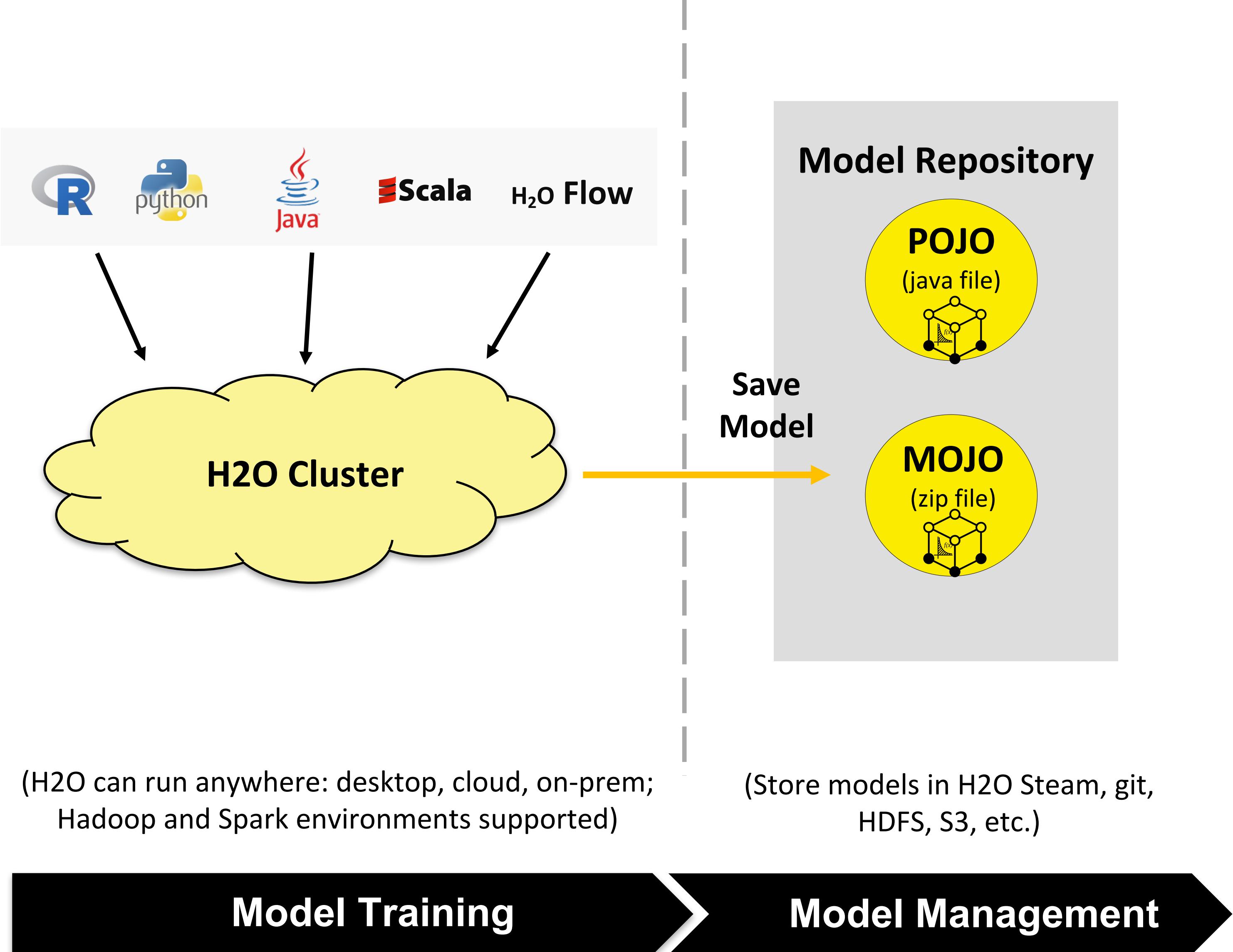
Machine Learning Model Lifecycle



(H2O can run anywhere: desktop, cloud, on-prem;
Hadoop and Spark environments supported)

Model Training

Machine Learning Model Lifecycle

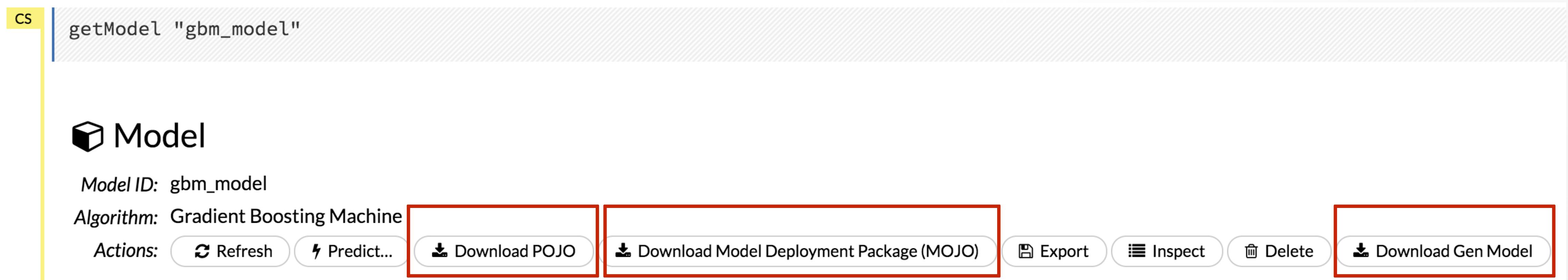


What is POJO/MOJO

- Java Code file
 - POJO – Source Code
 - Source code managed
 - MOJO – Compressed Java
 - Un-readable as it is compressed
- H2O provides supported
 - H2o-genmodel.jar
 - Must required with both POJO and MOJO

Exporting POJO/MOJO

- **FLOW**



- R : The file system will be used for the environment where R is launched
 - `h2o.download_mojo(gbm_model, path='/tmp/', get_genmodel_jar = TRUE)`
 - `h2o.download_pojo(gbm_model, path='/tmp/')`
- Python: The file system will be used for the environment where R is launched
 - `gbm_model.download_mojo(path='/tmp/', get_genmodel_jar=True)`
 - `gbm_model.download_pojo(path='/tmp/', get_genmodel_jar=True)`

Plain Old Java Object (POJO) Code

```
/*
 Licensed under the Apache License, Version 2.0
 http://www.apache.org/licenses/LICENSE-2.0.html

 AUTOGENERATED BY H2O at 2017-01-18T16:30:55.681-08:00
 3.10.2.2

 Standalone prediction code with sample test data for GBMModel named gbm_prostate

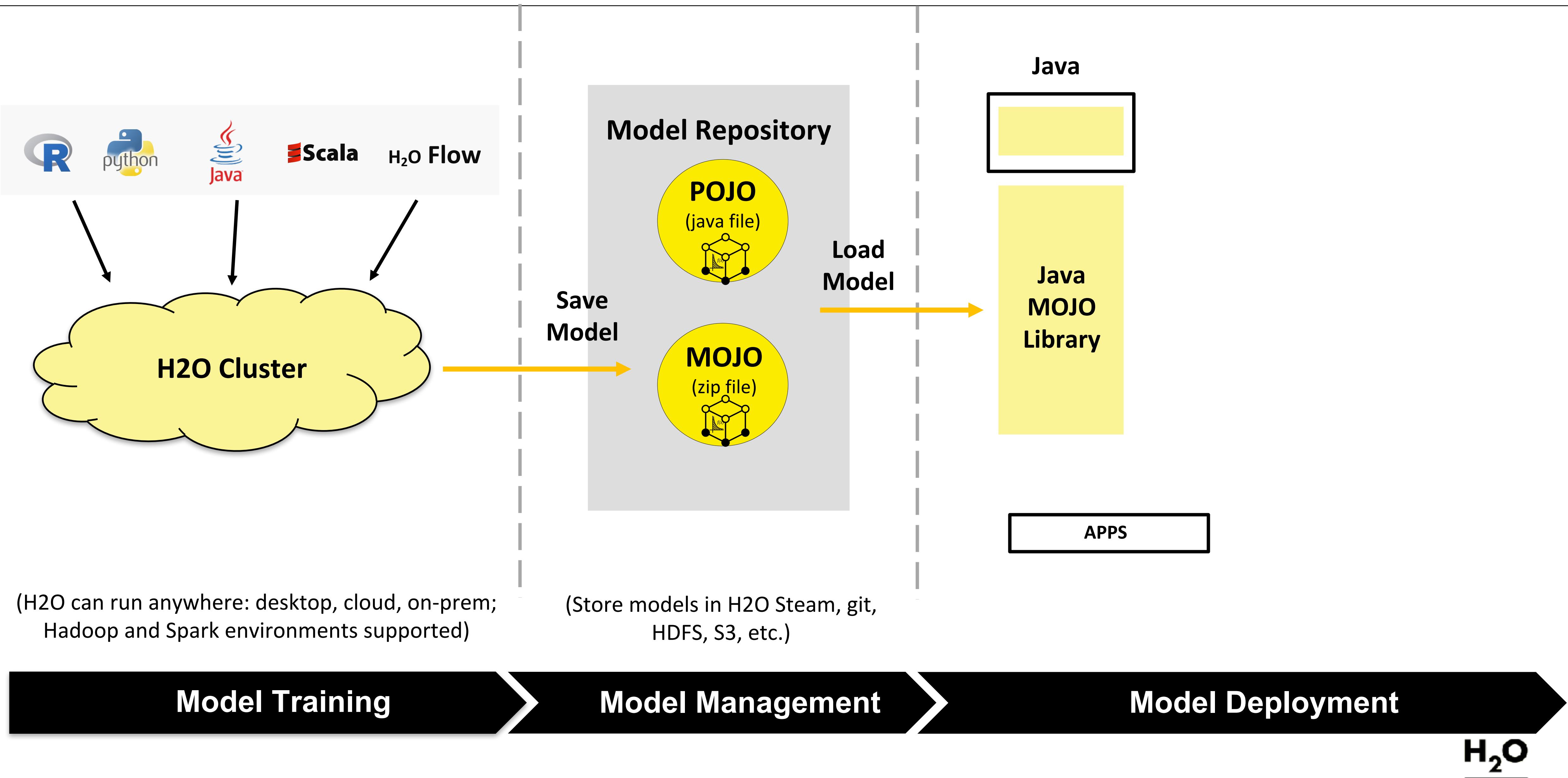
 How to download, compile and execute:
 mkdir tmpdir
 cd tmpdir
 curl http://10.0.1.13:54321/3/h2o-genmodel.jar > h2o-genmodel.jar
 curl http://10.0.1.13:54321/3/Models.java/gbm_prostate > gbm_prostate.java
 javac -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m gbm_prostate.java

 (Note: Try java argument -XX:+PrintCompilation to show runtime JIT compiler behavior.)
*/
import java.util.Map;
import hex.genmodel.GenModel;
import hex.genmodel.annotations.ModelPojo;

@ModelPojo(name="gbm_prostate", algorithm="gbm")
public class gbm_prostate extends GenModel {
    public hex.ModelCategory getModelCategory() { return hex.ModelCategory.Binomial; }

    public boolean isSupervised() { return true; }
    public int nfeatures() { return 5; }
    public int nclasses() { return 2; }

    // Names of columns used by model.
    public static final String[] NAMES = NamesHolder_gbm_prostate.VALUES;
    // Number of output classes included in training data response column.
    public static final int NCLASSES = 2;
```



POJO vs MOJO

	Java POJO	MOJO / Java Runtime
LANGUAGES	<ul style="list-style-type: none"> Java 	<ul style="list-style-type: none"> Java
ALGORITHMS	<ul style="list-style-type: none"> All 	<ul style="list-style-type: none"> GLM, DRF, GBM, GLRM (but eventually all)
COMPILING	<ul style="list-style-type: none"> Compiled Large POJO Compilation Errors 	<ul style="list-style-type: none"> Interpreted No compile errors
CLASS LOADING	<ul style="list-style-type: none"> Java class loading issues for many models <ul style="list-style-type: none"> — must tune MaxPermSize 	<ul style="list-style-type: none"> No Java class loading issues (very few fixed number of classes)
ALLOCATION	<ul style="list-style-type: none"> Models use PermGen storage 	<ul style="list-style-type: none"> Models stored in the Java heap
PERFORMANCE	<ul style="list-style-type: none"> Faster for small trees (about 1.5x-2x) 	<ul style="list-style-type: none"> Faster for large trees (about 20x)
BACKWARD COMPATIBILITY	<ul style="list-style-type: none"> POJO genmodel backward compatibility unsupported Difficult to run old and new versions side-by-side (requires multiple library versions and classloading tricks) 	<ul style="list-style-type: none"> Strong MOJO genmodel backward compatibility Easy to run old and new versions side-by-side Same prediction results with future genmodels for a saved MOJO
SIZE	<ul style="list-style-type: none"> ~10x larger than MOJO, so ~10x larger than .zip file 	<ul style="list-style-type: none"> Size is 1/10th of a POJO
EASY WRAPPER	<ul style="list-style-type: none"> Works with Java EasyPredict API 	<ul style="list-style-type: none"> Works with Java EasyPredict API
DECISION VISIBILITY	<ul style="list-style-type: none"> Decisions are visible in generated code 	<ul style="list-style-type: none"> Opaque (but open source) binary format Visualization tools provided (e.g. tree viewer)

POJO - main.java sample for regression

```
import java.io.*;
import java.util.Arrays;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;

public class main {
    private static String modelClassName = "glm_model";

    public static void main(String[] args) throws Exception {

        hex.genmodel.GenModel rawModel;
        rawModel = (hex.genmodel.GenModel) Class.forName(modelClassName).newInstance();

        System.out.println("isSupervised : " + rawModel.isSupervised());
        System.out.println("Model UUID : " + rawModel.getUUID());
        System.out.println("Column Names : " + Arrays.toString(rawModel.getNames()));
        System.out.println("Response ID : " + rawModel.getResponseIdx());
        System.out.println("Number of columns : " + rawModel.getNumCols());
        //System.out.println("Response Name : " + rawModel.getResponseName());
        System.out.println("isClassifier : " + rawModel.isClassifier());
        if (rawModel.isClassifier()) {
            System.out.println("Response Classes : " + rawModel.getNumResponseClasses());
        }

        for (int i = 0; i < rawModel.getNumCols(); i++) {
            String[] domainValues = rawModel.getDomainValues(i);
            System.out.println(Arrays.toString(domainValues));
        }

        EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);

        RowData row = new RowData();
        row.put("AGE", "68");
        row.put("RACE", "2");
        row.put("DCAPS", "2");
        row.put("VOL", "0");
        row.put("GLEASON", "6");

        RegressionModelPrediction p = model.predictRegression(row);
        System.out.println("Values: " + p.value);
        System.out.println("");
    }
}
```

POJO - main.java sample for classification

```
import java.io.*;
import java.util.Arrays;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;

public class main {
    private static String modelClassName = "gbm_model";

    public static void main(String[] args) throws Exception {

        hex.genmodel.GenModel rawModel;
        rawModel = (hex.genmodel.GenModel) Class.forName(modelClassName).newInstance();

        System.out.println("isSupervised : " + rawModel.isSupervised());
        System.out.println("Column Names : " + Arrays.toString(rawModel.getNames()));
        System.out.println("Response ID : " + rawModel.getResponseIdx());
        System.out.println("Number of columns : " + rawModel.getNumCols());
        //System.out.println("Response Name : " + rawModel.getResponseName());
        for (int i = 0; i < rawModel.getNumCols(); i++) {
            String[] domainValues = rawModel.getDomainValues(i);
            System.out.println(Arrays.toString(domainValues));
        }

        EasyPredictModelWrapper model = new EasyPredictModelWrapper(rawModel);
        RowData row = new RowData();
        row.put("AGE", "68");
        row.put("RACE", "2");
        row.put("DCAPS", "2");
        row.put("VOL", "0");
        row.put("GLEASON", "6");

        BinomialModelPrediction p = model.predictBinomial(row);
        System.out.println("Has penetrated the prostatic capsule (1=yes; 0=no): " + p.label);
        System.out.print("Class probabilities: ");
        for (int i = 0; i < p.classProbabilities.length; i++) {
            if (i > 0) {
                System.out.print(",");
            }
            System.out.print(p.classProbabilities[i]);
        }
        System.out.println("");
    }
}
```

MOJO - main.java sample for Regression

```
import java.io.*;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;
import hex.genmodel.MojoModel;
import java.util.Arrays;

public class main {
    public static void main(String[] args) throws Exception {
        EasyPredictModelWrapper model = new EasyPredictModelWrapper(MojoModel.load("glm_model.zip"));

        RowData row = new RowData();
        row.put("AGE", "68");
        row.put("RACE", "2");
        row.put("DCAPS", "2");
        row.put("VOL", "0");
        row.put("GLEASON", "6");

        RegressionModelPrediction p = model.predictRegression(row);
        System.out.println("Values: " + p.value);
        System.out.println("");
    }
}
```

MOJO – main.java sample for Classification

```
import java.io.*;
import hex.genmodel.easy.RowData;
import hex.genmodel.easy.EasyPredictModelWrapper;
import hex.genmodel.easy.prediction.*;
import hex.genmodel.MojoModel;
import java.util.Arrays;

public class main {
    public static void main(String[] args) throws Exception {
        EasyPredictModelWrapper model = new EasyPredictModelWrapper(MojoModel.load("gbm_model.zip"));

        RowData row = new RowData();
        row.put("AGE", "68");
        row.put("RACE", "2");
        row.put("DCAPS", "2");
        row.put("VOL", "0");
        row.put("GLEASON", "6");

        BinomialModelPrediction p = model.predictBinomial(row);
        System.out.println("Has penetrated the prostatic capsule (1=yes; 0=no): " + p.label);
        System.out.print("Class probabilities: ");
        for (int i = 0; i < p.classProbabilities.length; i++) {
            if (i > 0) {
                System.out.print(",");
            }
            System.out.print(p.classProbabilities[i]);
        }
        System.out.println("");
    }
}
```

Java Prediction Examples

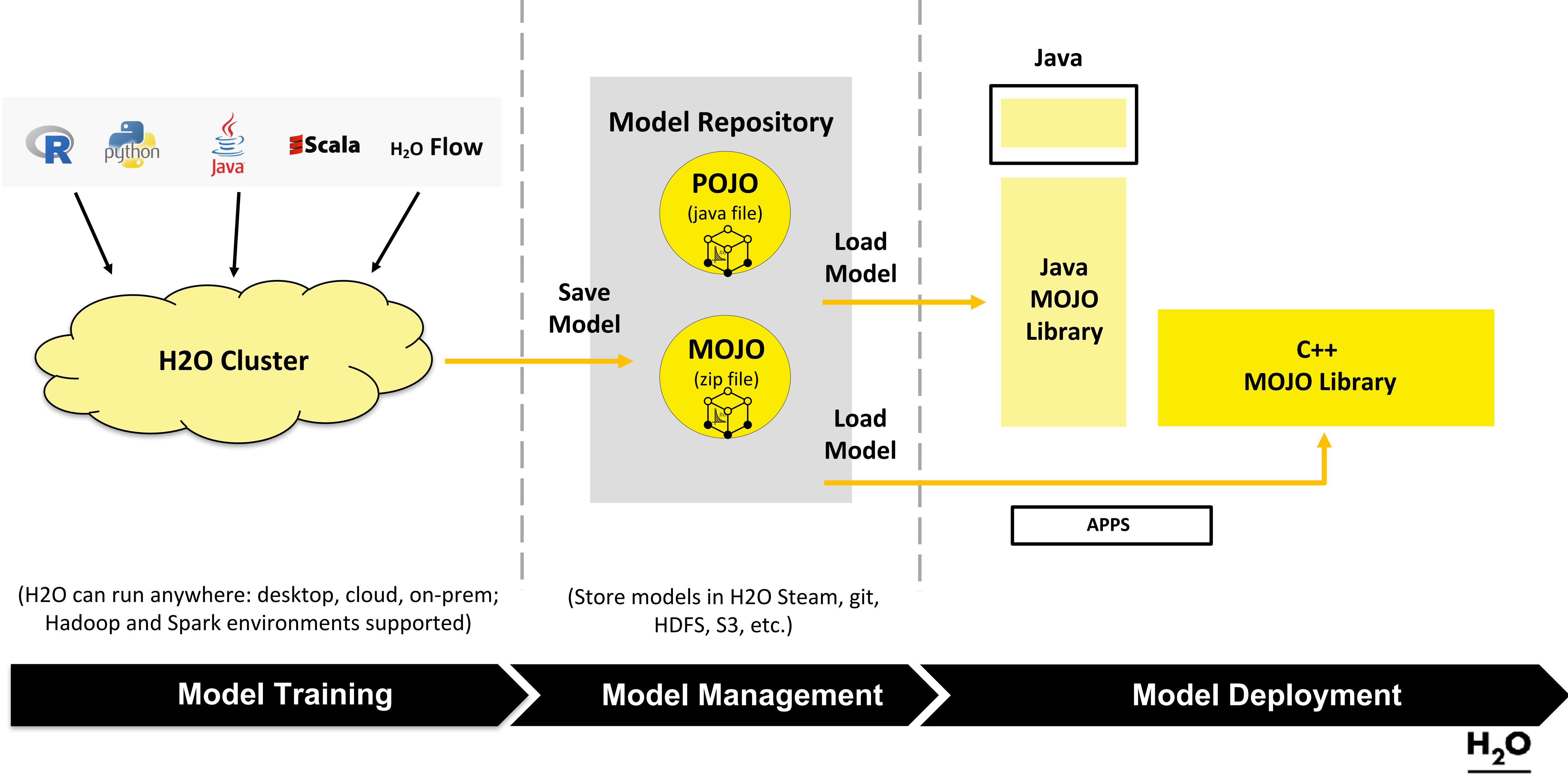
- POJO
 - Compile:
 - `$ javac -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m gbm_prostate.java main.java`
 - Run:
 - `$ java -cp .:h2o-genmodel.jar main`

- MOJO
 - Compile:
 - `$ javac -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m main.java`
 - Run:
 - `$ java -cp .:h2o-genmodel.jar main`

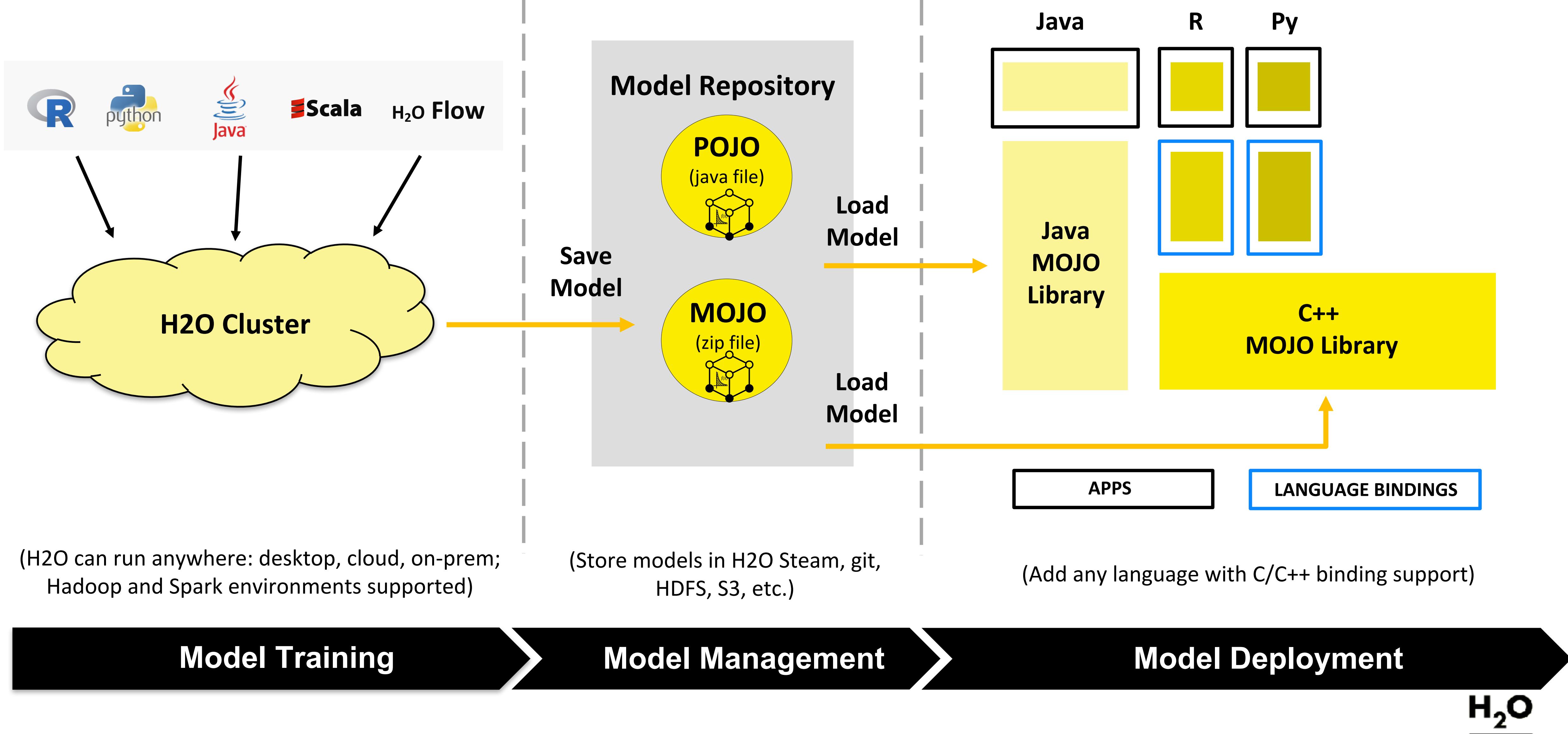
Python MOJO Scoring

- Look or the API
 - Available into H2O 3.14.0.3 onwards ONLY
 - `H2o.predict_json()`

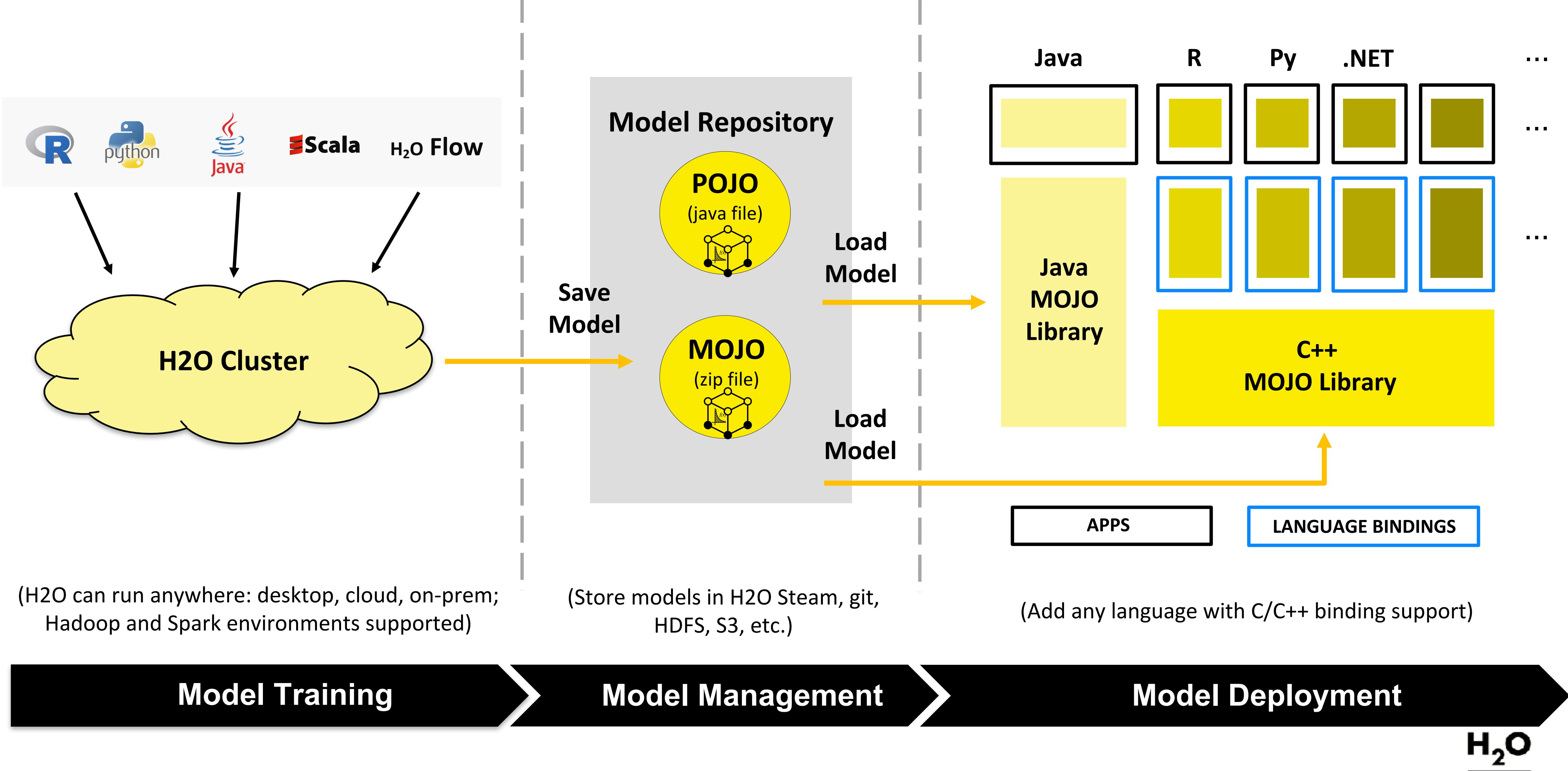
Machine Learning Model Lifecycle



Machine Learning Model Lifecycle



MOJOs Enable Machine Learning Model Predictions in Most Languages



POJO vs MOJO

	Java POJO	MOJO / Java Runtime	MOJO / C++ Runtime
LANGUAGES	<ul style="list-style-type: none"> Java 	<ul style="list-style-type: none"> Java 	<ul style="list-style-type: none"> C++, R, Python, .NET, other
ALGORITHMS	<ul style="list-style-type: none"> All 	<ul style="list-style-type: none"> GLM, DRF, GBM, GLRM (but eventually all) 	<ul style="list-style-type: none"> GBM (but eventually all)
COMPILING	<ul style="list-style-type: none"> Compiled Large POJO Compilation Errors 	<ul style="list-style-type: none"> Interpreted No compile errors 	<ul style="list-style-type: none"> Same as Java
CLASS LOADING	<ul style="list-style-type: none"> Java class loading issues for many models — must tune MaxPermSize 	<ul style="list-style-type: none"> No Java class loading issues (very few fixed number of classes) 	<ul style="list-style-type: none"> (Does not apply)
ALLOCATION	<ul style="list-style-type: none"> Models use PermGen storage 	<ul style="list-style-type: none"> Models stored in the Java heap 	<ul style="list-style-type: none"> Models stored in the C heap
PERFORMANCE	<ul style="list-style-type: none"> Faster for small trees (about 1.5x-2x) 	<ul style="list-style-type: none"> Faster for large trees (about 20x) 	<ul style="list-style-type: none"> Same as Java
BACKWARD COMPATIBILITY	<ul style="list-style-type: none"> POJO genmodel backward compatibility unsupported Difficult to run old and new versions side-by-side (requires multiple library versions and classloading tricks) 	<ul style="list-style-type: none"> Strong MOJO genmodel backward compatibility Easy to run old and new versions side-by-side Same prediction results with future genmodels for a saved MOJO 	<ul style="list-style-type: none"> Same as Java
SIZE	<ul style="list-style-type: none"> ~10x larger than MOJO, so ~10x larger than .zip file 	<ul style="list-style-type: none"> Size is 1/10th of a POJO 	<ul style="list-style-type: none"> Same as Java
EASY WRAPPER	<ul style="list-style-type: none"> Works with Java EasyPredict API 	<ul style="list-style-type: none"> Works with Java EasyPredict API 	<ul style="list-style-type: none"> Works with C++ EasyPredict API
DECISION VISIBILITY	<ul style="list-style-type: none"> Decisions are visible in generated code 	<ul style="list-style-type: none"> Opaque (but open source) binary format Visualization tools provided (e.g. tree viewer) 	<ul style="list-style-type: none"> Same as Java

C++ MOJO (MOCO) Scoring Demo

```
In [17]: import unittest  
import os  
import sys  
import h2omojo
```

```
In [18]: mojo_model = h2omojo.load_mojo_model(mojo_path='/Users/avkashchauhan/learn/customers/mojo-cpp/airline_unzipped')
```

```
In [19]: def raw(mojo_model):  
    row2 = []  
    n = mojo_model.nfeatures()  
    for i in range(n):  
        row2.append(0.0)  
    row2[0] = 68  
    row2[1] = 2  
    row2[2] = 2  
    row2[3] = 0  
    row2[4] = 6  
  
    pred2 = mojo_model.score0(row2)  
  
    n = mojo_model.get_preds_size()  
    if n != len(pred2):  
        raise Exception("Size mismatch")  
  
    if True:  
        for i in range(len(pred2)):  
            print(pred2[i])  
  
    print("")
```

```
In [21]: h2omojo.set_verbosity(0)  
raw(mojo_model)
```

```
1.0  
0.522506084464  
0.477493915536
```

Spark UDF Example

- Here is the Sample Code Snippet

```
import _root_.hex.genmodel.GenModel
import _root_.hex.genmodel.easy.{EasyPredictModelWrapper, RowData}
import _root_.hex.genmodel.easy.prediction
import _root_.hex.genmodel.MojoModel
import _root_.hex.genmodel.easy.RowData

// Load Mojo val mojo = MojoModel.load("/Users/avkashchauhan/learn/customers/mojo_bin/gbm_model.zip")
val easyModel = new EasyPredictModelWrapper(mojo)

// Get Mojo Details
var features = mojo.getNames.toBuffer

// Creating the row
val r = new RowData
r.put("AGE", "68")
r.put("RACE", "2")
r.put("DCAPS", "2")
r.put("VOL", "0")
r.put("GLEASON", "6")

// Performing the Prediction
val prediction = easyModel.predictBinomial(r).classProbabilities
```

- Visit: https://github.com/Avkash/mldl/blob/master/code/scala/spark_udf_score.md

Hive UDF Example

- Visit github: https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/hive_udf_template
 - Hive POJO UDF
 - https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/hive_udf_template/hive_udf_pojo_template
 - Hive MOJO UDF
 - https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/hive_udf_template/hive_udf_mojo_template
 - Hive Multi MOJO UDF
 - https://github.com/h2oai/h2o-tutorials/tree/master/tutorials/hive_udf_template/hive_multimojo_udf_template

Hive MOJO UDF Demo

- **Preparation to have test data on Hive**
 - Copying dataset to HDFS
 - Adding test dataset into Hive
- **Your Models**
 - Get your MOJO model (model.zip) and h2o-genmodel.jar
- **Working on Repo**
 - Clone the repo
 - Copy the model and genmodel from exported location to into the Hive UDF project
 - Update the pom.xml to Reflect Hadoop and Hive Versions
 - Now Compiling the UDF Jar File
 - Copying the final builds files to HDFS
 - Adding JAR files to Hive
 - Validating the UDF function is in Hive
 - Testing the solution in Hive
- Visit: https://github.com/Avkash/mldl/blob/master/orgs/h2o/hive_udf_prostate.md

Handling unseen categorical with MOJO/POJO

- Question:
 - What happens when you try to predict on a categorical level not seen during training?
- If given data has unseen categorical
 - Unseen categorical levels are turned into NAs,
 - Follow the same behavior as an NA
 - When training data has NO - NA
 - Unseen categorical levels in the test data follow the majority direction
 - » the direction with the most observations
 - When training data has NA
 - Unseen categorical levels in the test data follow the direction that is optimal for the NAs of the training data



avkash@h2o.ai