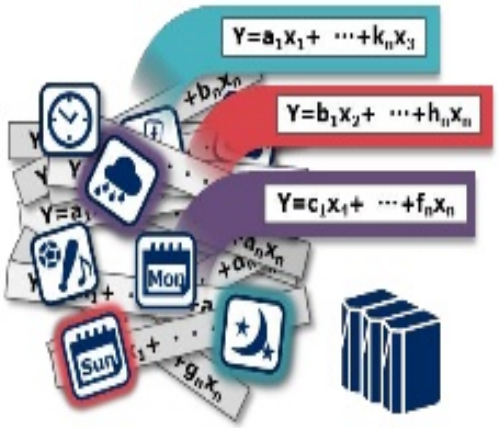
\Orchestrating a brighter world NEC

# Distributed Heterogeneous Mixture Learning on Spark

Masato Asahara and Ryohei Fujimaki
NEC Data Science Research Labs.
Jun/08/2016 @Spark Summit 2016

# Agenda

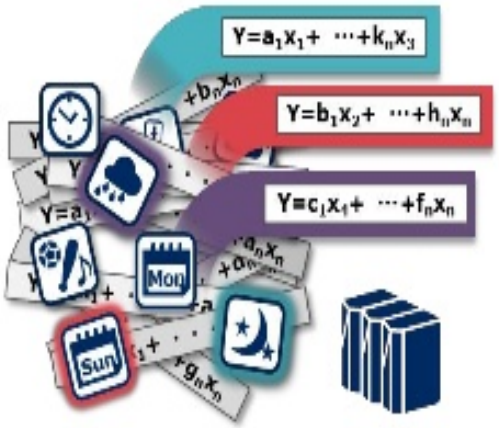
# Agenda




HML

Spark

# Agenda

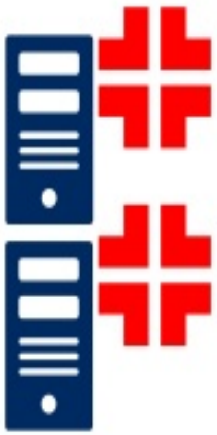


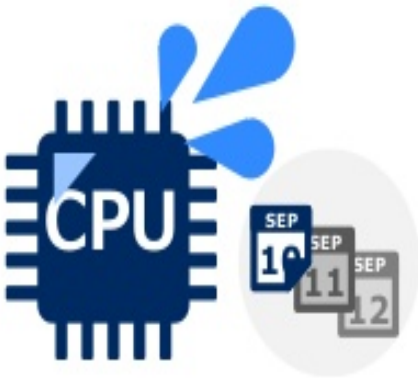

HML


**Data Shuffling**

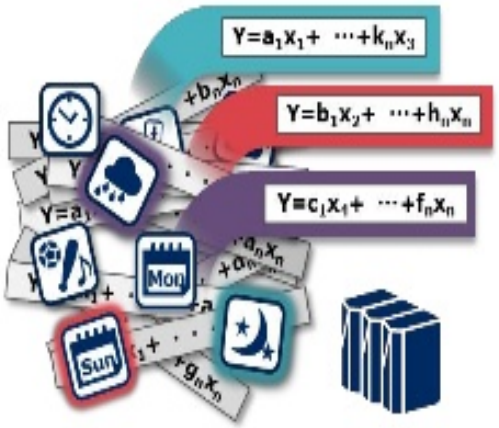
**MapReduce Synchronization**

**Matrix Computation**





# Agenda




HML

Spark


# NEC's Predictive Analytics and Heterogeneous Mixture Learning

# Enterprise Applications of HML

**Energy/Water Operation Mgmt.**

**Sales Optimization**

**Product Price Optimization**


**Driver Risk Assessment**

**Predictive Maintenance**

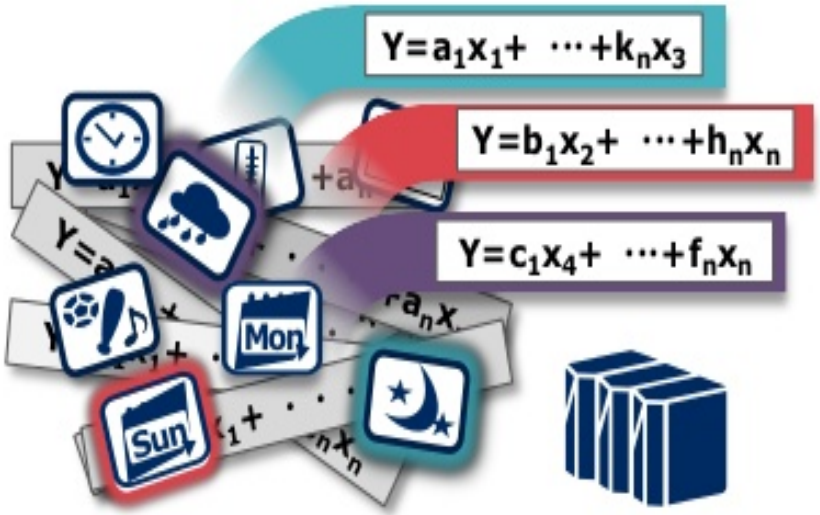
**Churn Retention**

**Inventory Optimization**

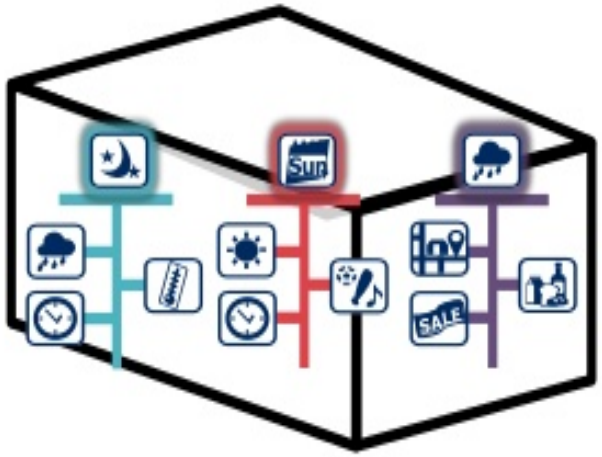
# NEC's Heterogeneous Mixture Learning (HML)

NEC's machine learning that automatically derives "accurate" and "transparent" formulas behind Big Data.


**Heterogeneous mixture data**

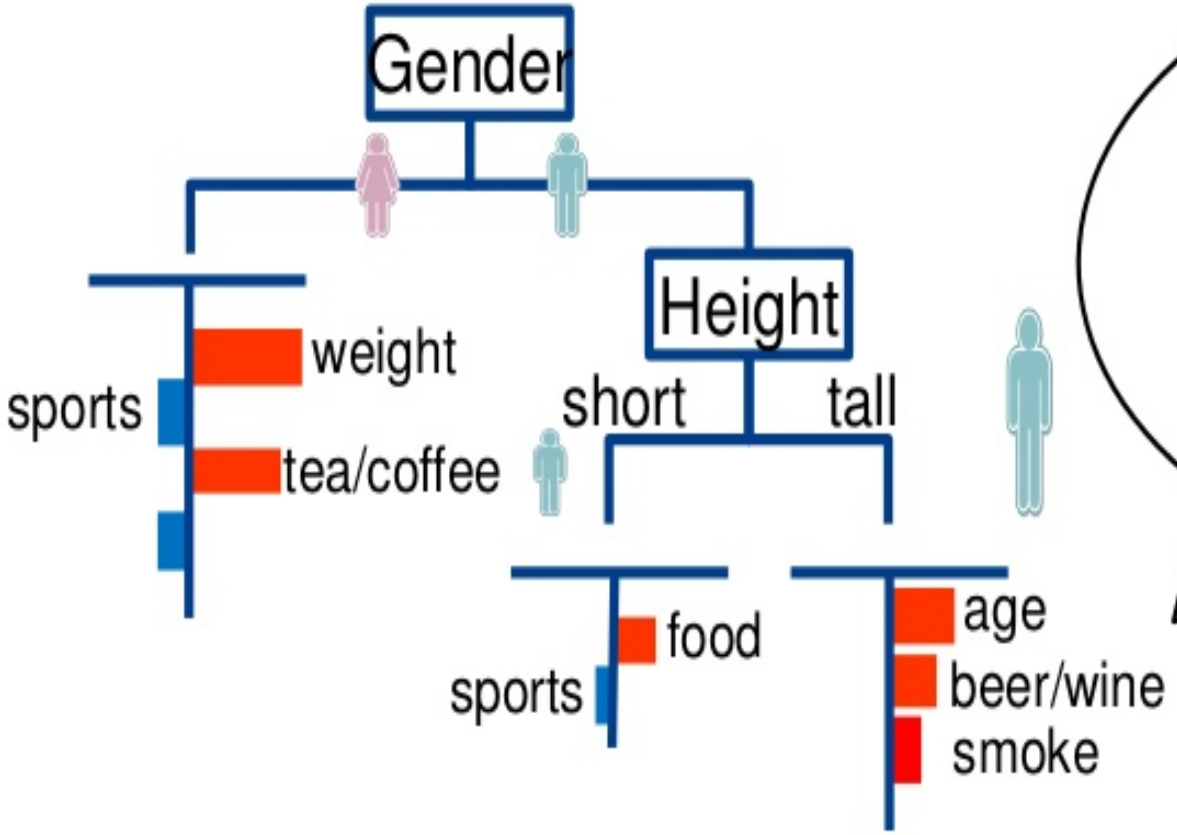



**Explore Massive Formulas**




**Transparent data segmentation and predictive formulas**

# The HML Model

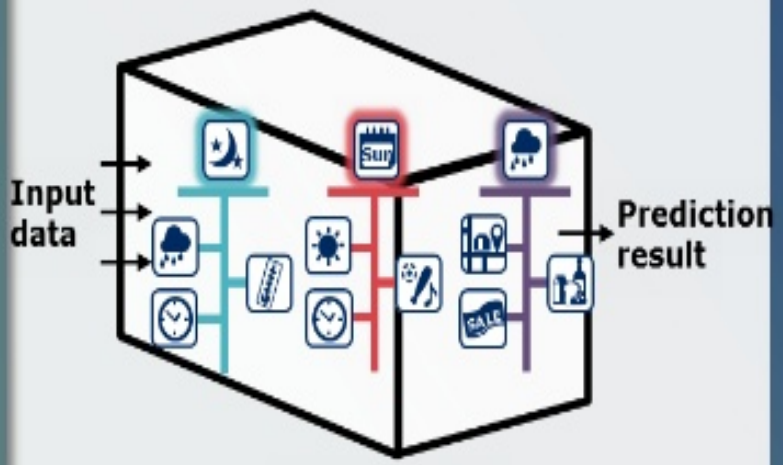



**The health risk of a tall man can be predicted by age, alcohol and smoke!**

# HML (Heterogeneous Mixture Learning)

## HML

Produces accurate predictions by switching optimal formulas. Transparent "reasons" support decision making based on predictions.



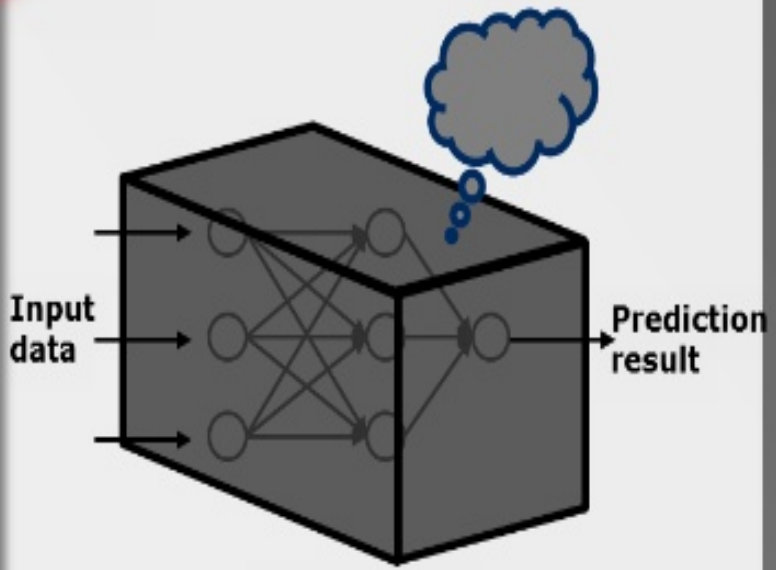

White box

**Actionable Intelligence**

**Accountable Intelligence**

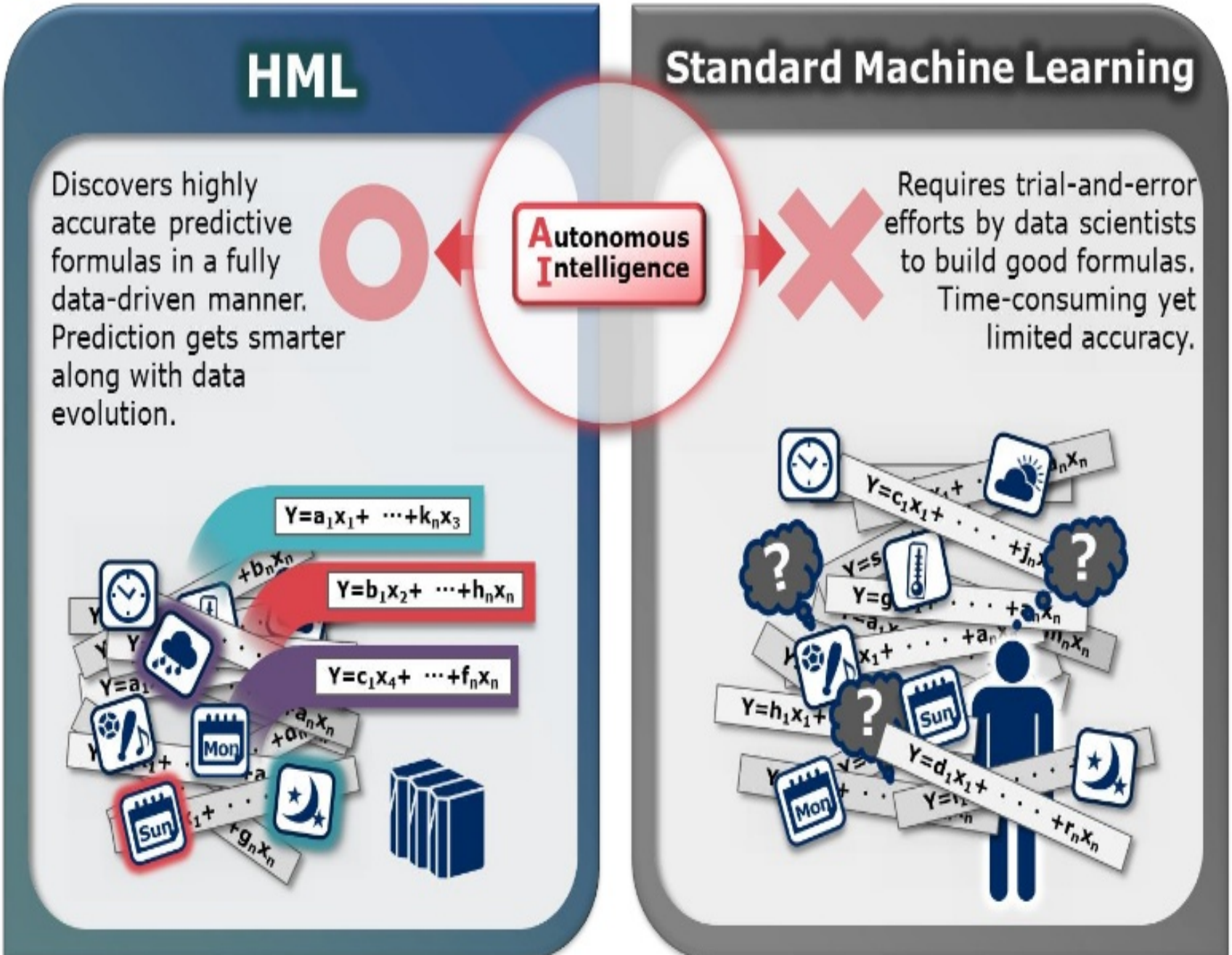
## Deep Learning

Produces accurate predictions by complex non-linear formula. Lack of transparency in predictions.



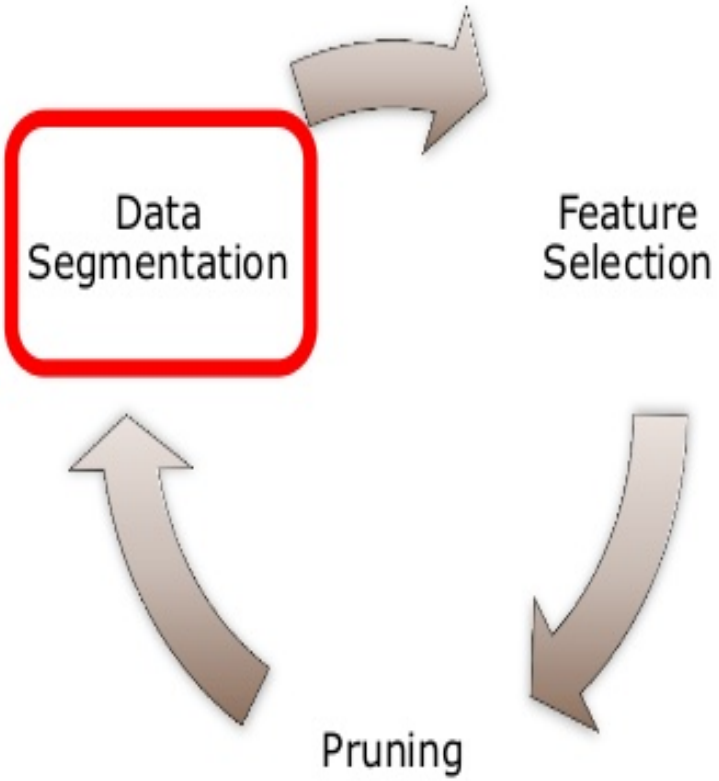

Black box

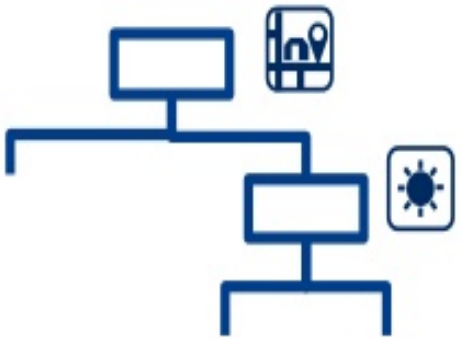
# HML (Heterogeneous Mixture Learning)

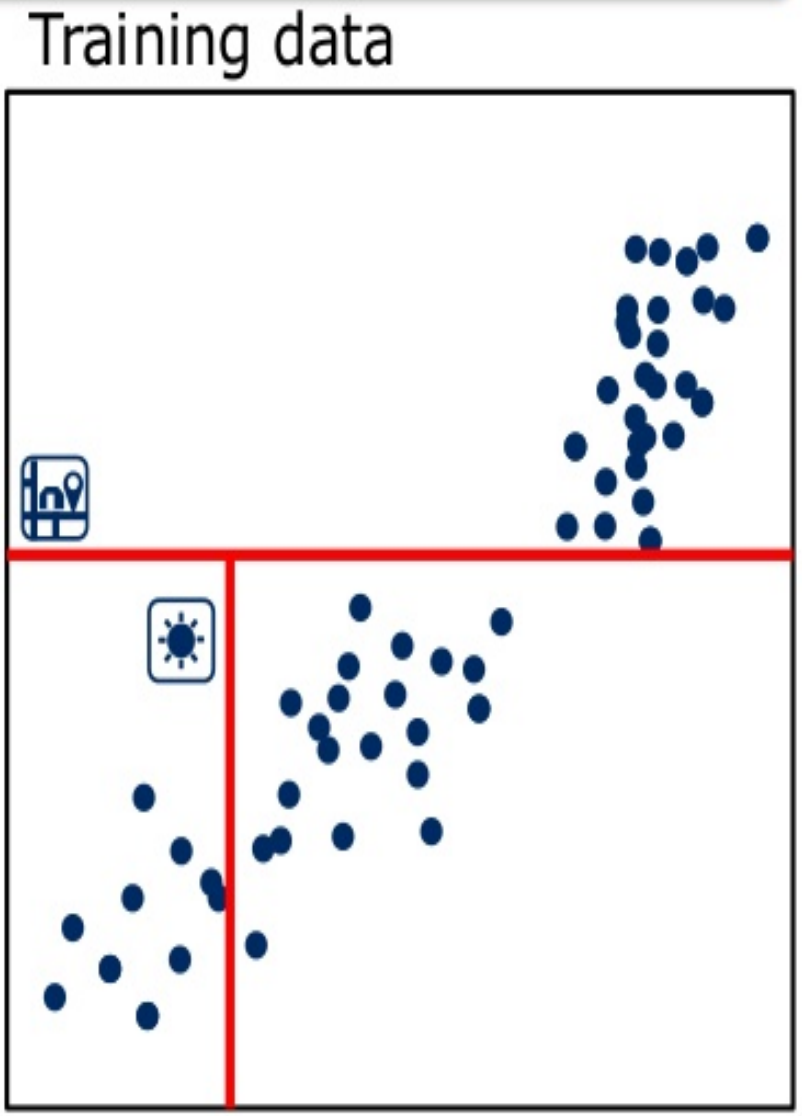


# HML Algorithm

## Segment data by a rule-based tree

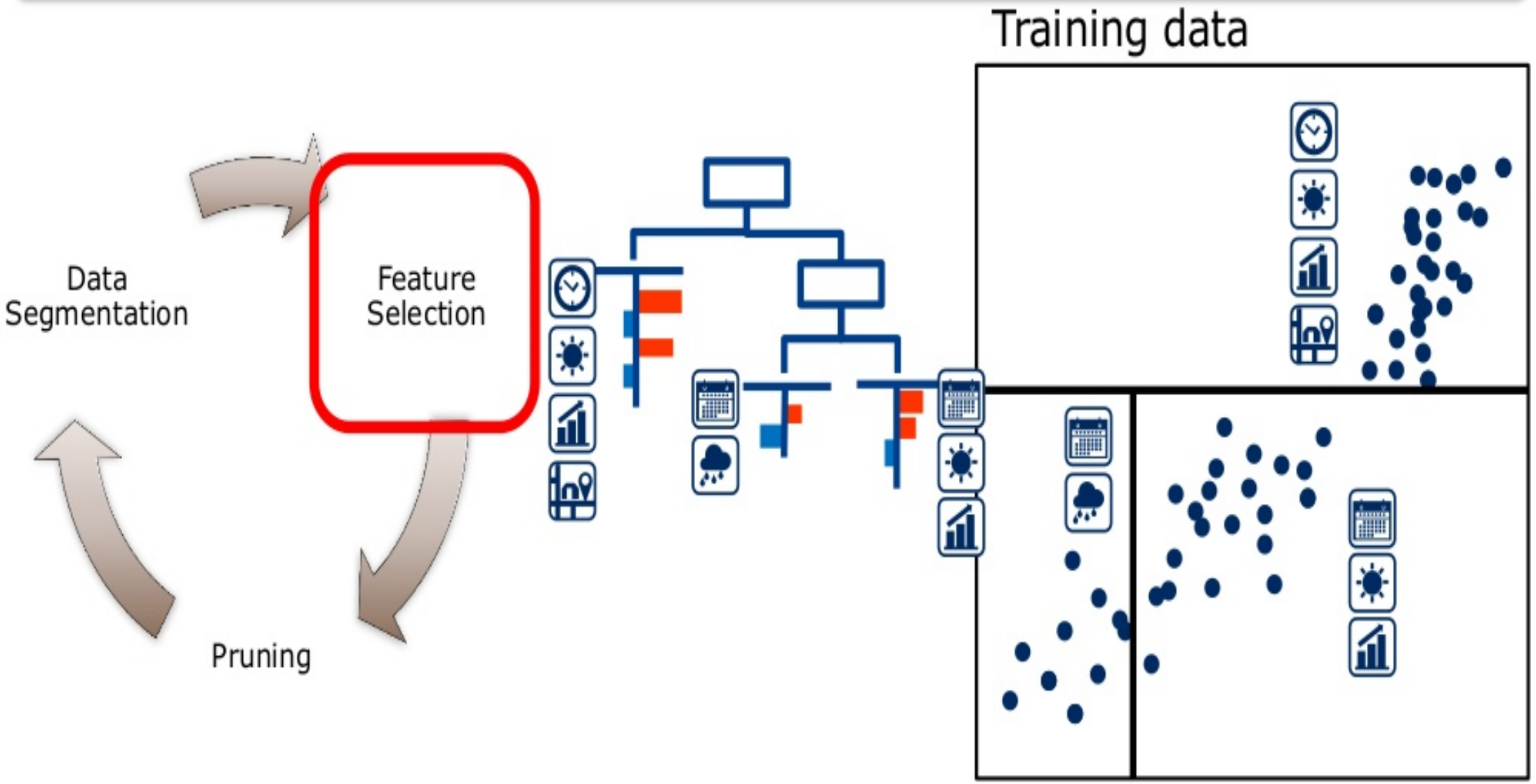




# HML Algorithm

## Select features and fit predictive models for data segments



# Enterprise Applications of HML

**Energy/Water Operation Mgmt.**

**Sales Optimization**

**Product Price Optimization**


**Driver Risk Assessment**

**Predictive Maintenance**

**Churn Retention**

**Inventory Optimization**

# Demand for Big Data Analysis

**Energy/Water Operation Mgmt.**

**Sales Optimization**

**Product Price Optimization**


**24(hour)×365(days)×3(year)×1000(shops)**
**~26,000,000 training samples**

# Demand for Big Data Analysis

**5 million(customers)×12(months)**
**=60,000,000 training samples**

**Driver Risk Assessment**

**Predictive Maintenance**

**Churn Retention**

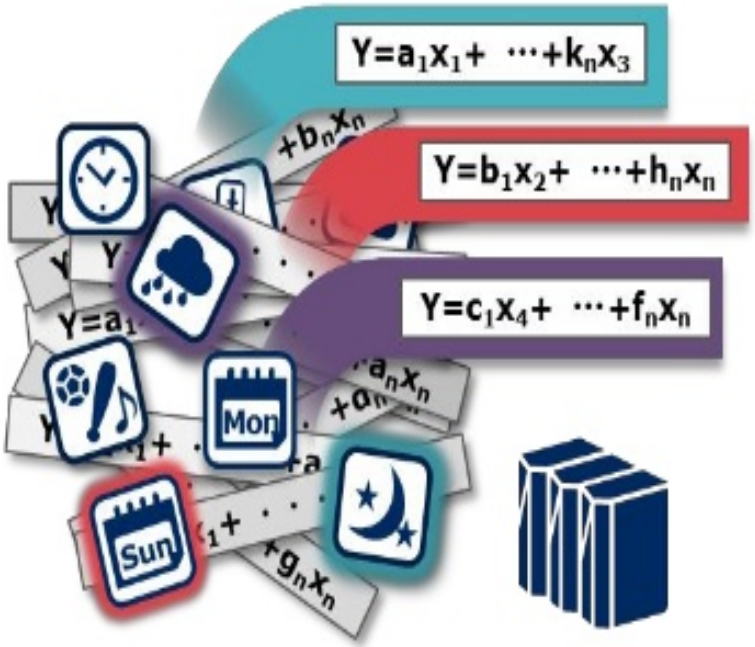
**Inventory Optimization**

# Demand for Big Data Analysis

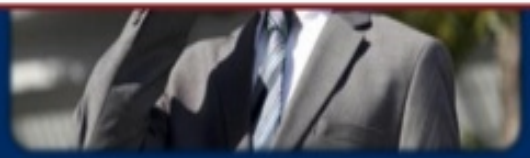
**Energy/Water Demand Forecasting**

**Sales Forecasting**

**Product Price Optimization**



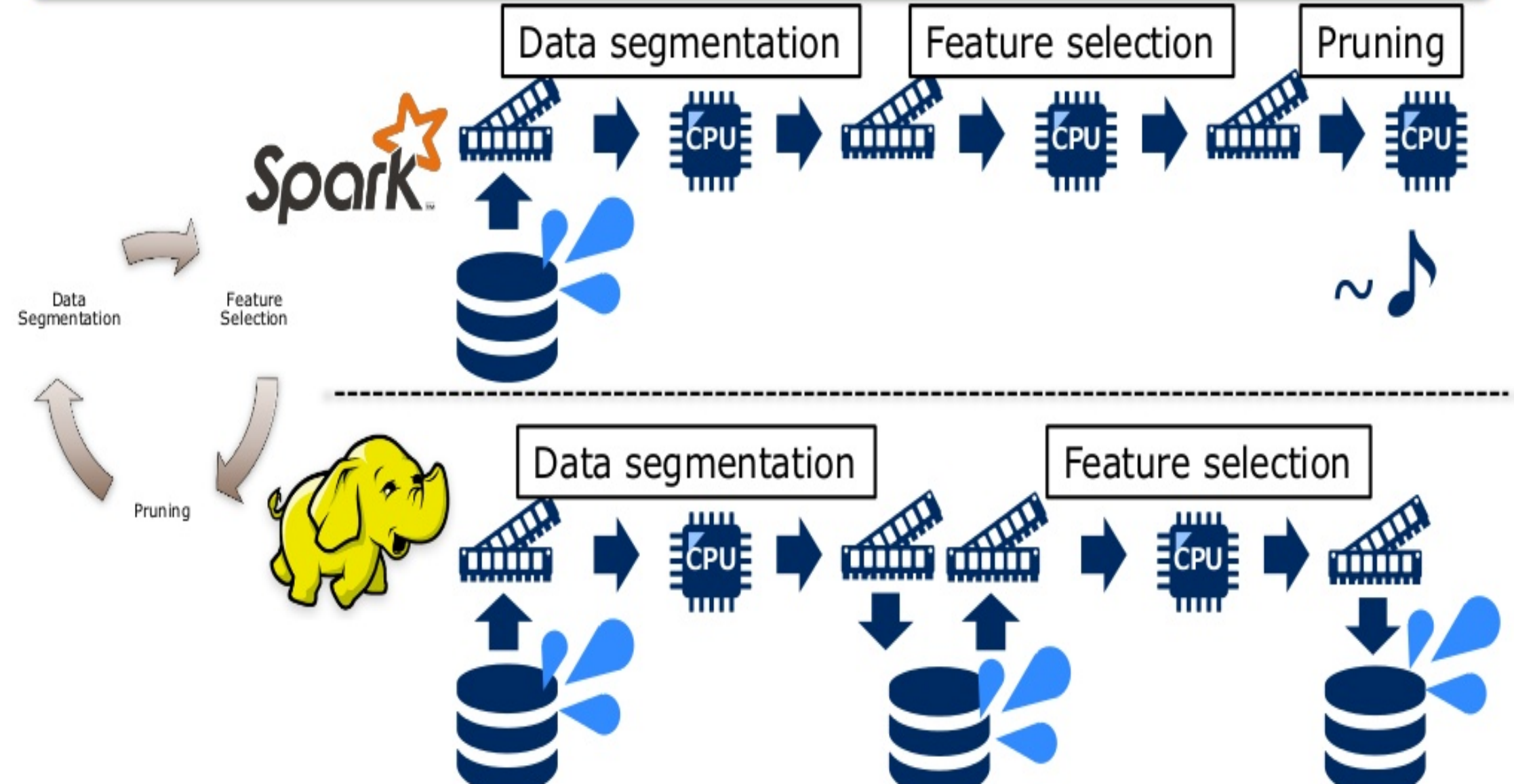

Spark



# Distributed HML on Spark

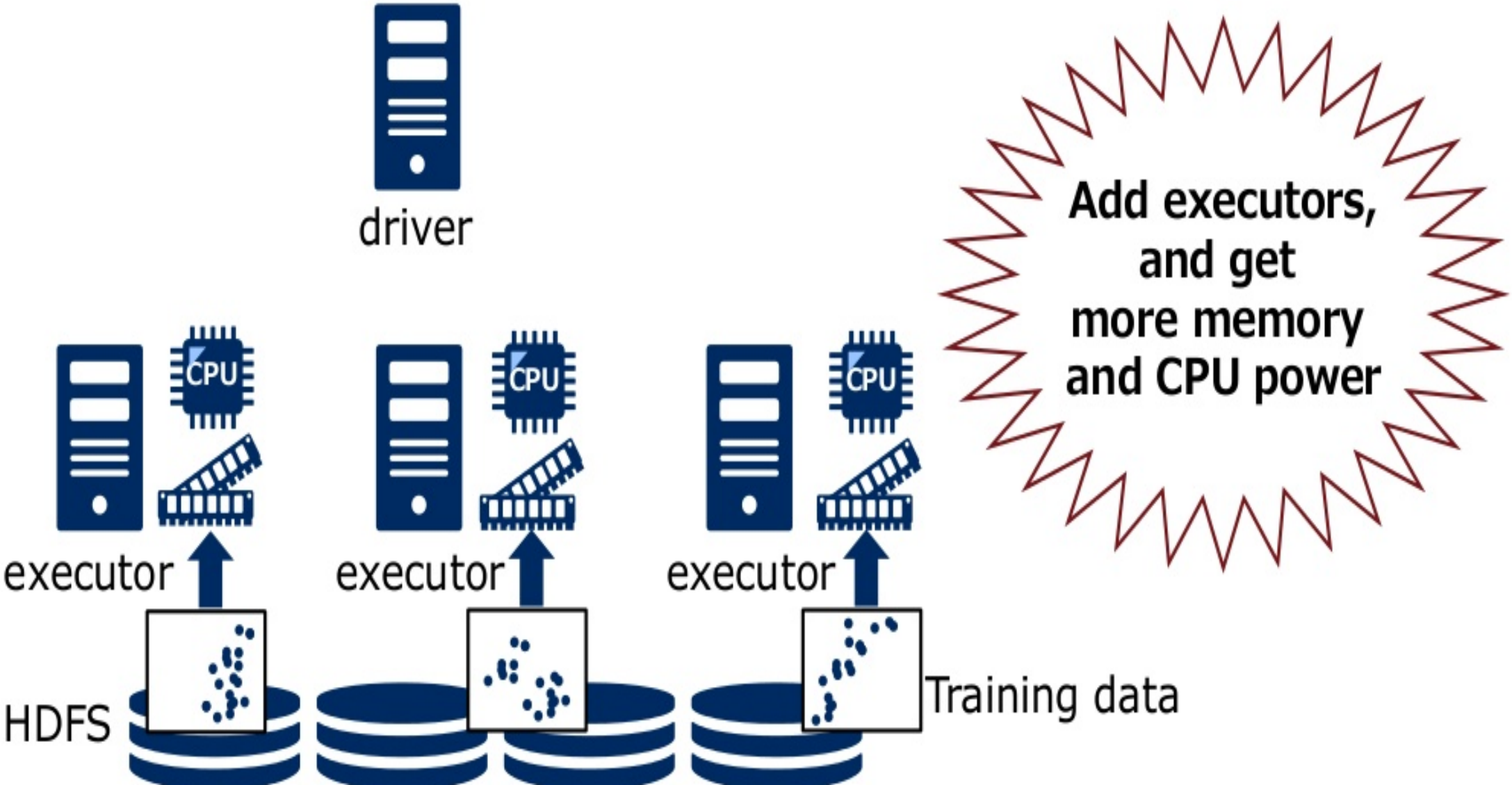
# Why Spark, not Hadoop?

## Because Spark's in-memory architecture can run HML faster

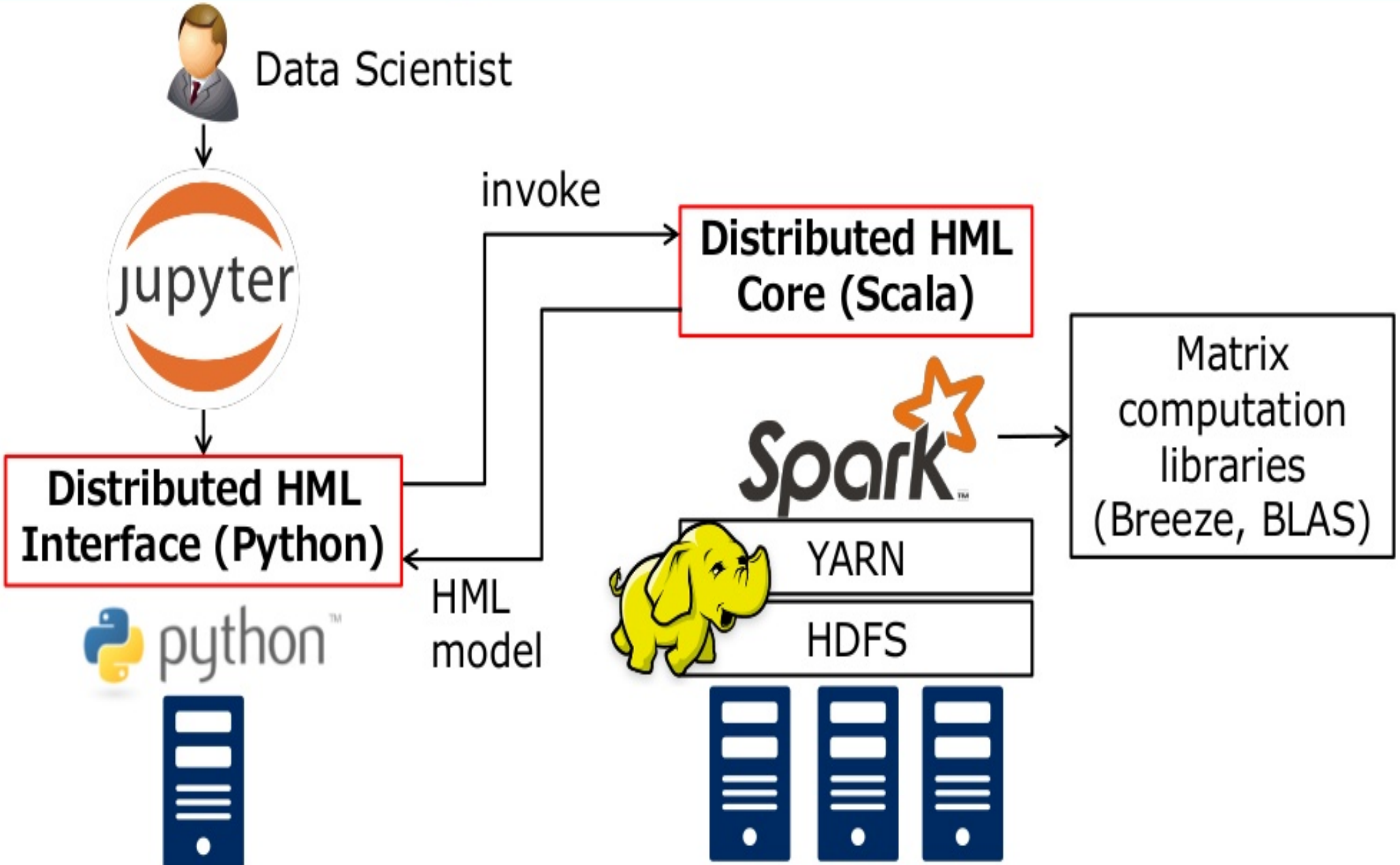


# Data Scalability powered by Spark

## Treat unlimited scale of training data by adding executors



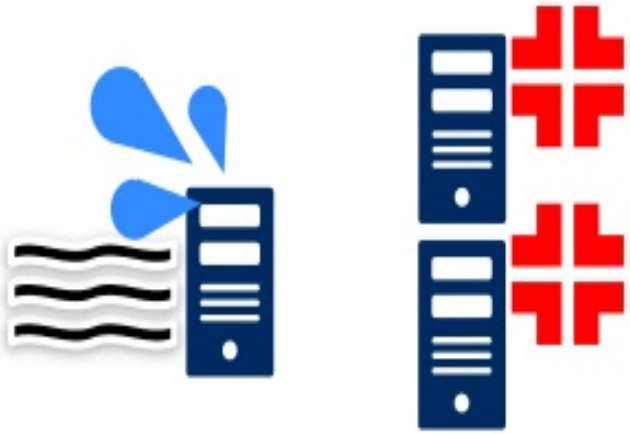
# Distributed HML Engine: Architecture



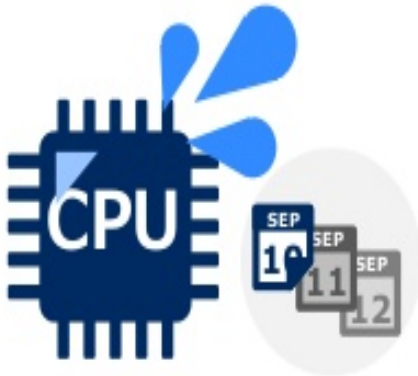
# 3 Technical Key Points to Fast Run ML on Spark

**Data Shuffling**

**MapReduce Synchronization**



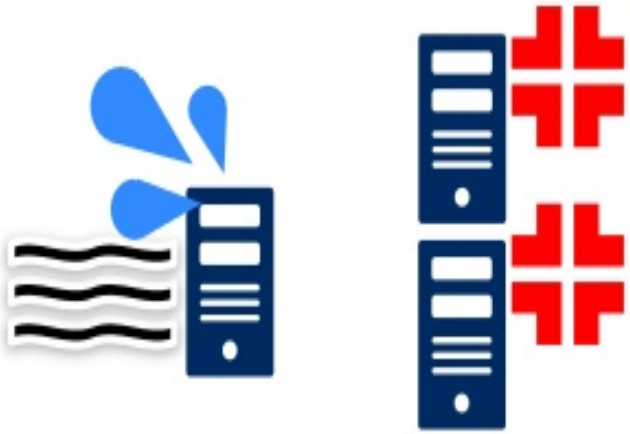
**Matrix Computation**



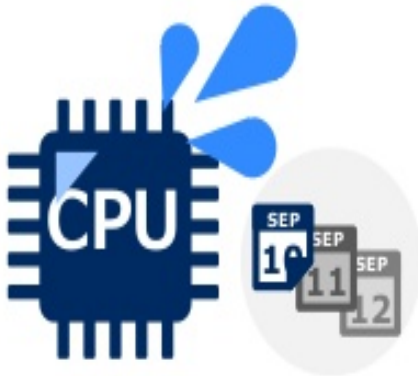
# 3 Technical Key Points to Fast Run ML on Spark

**Data Shuffling**

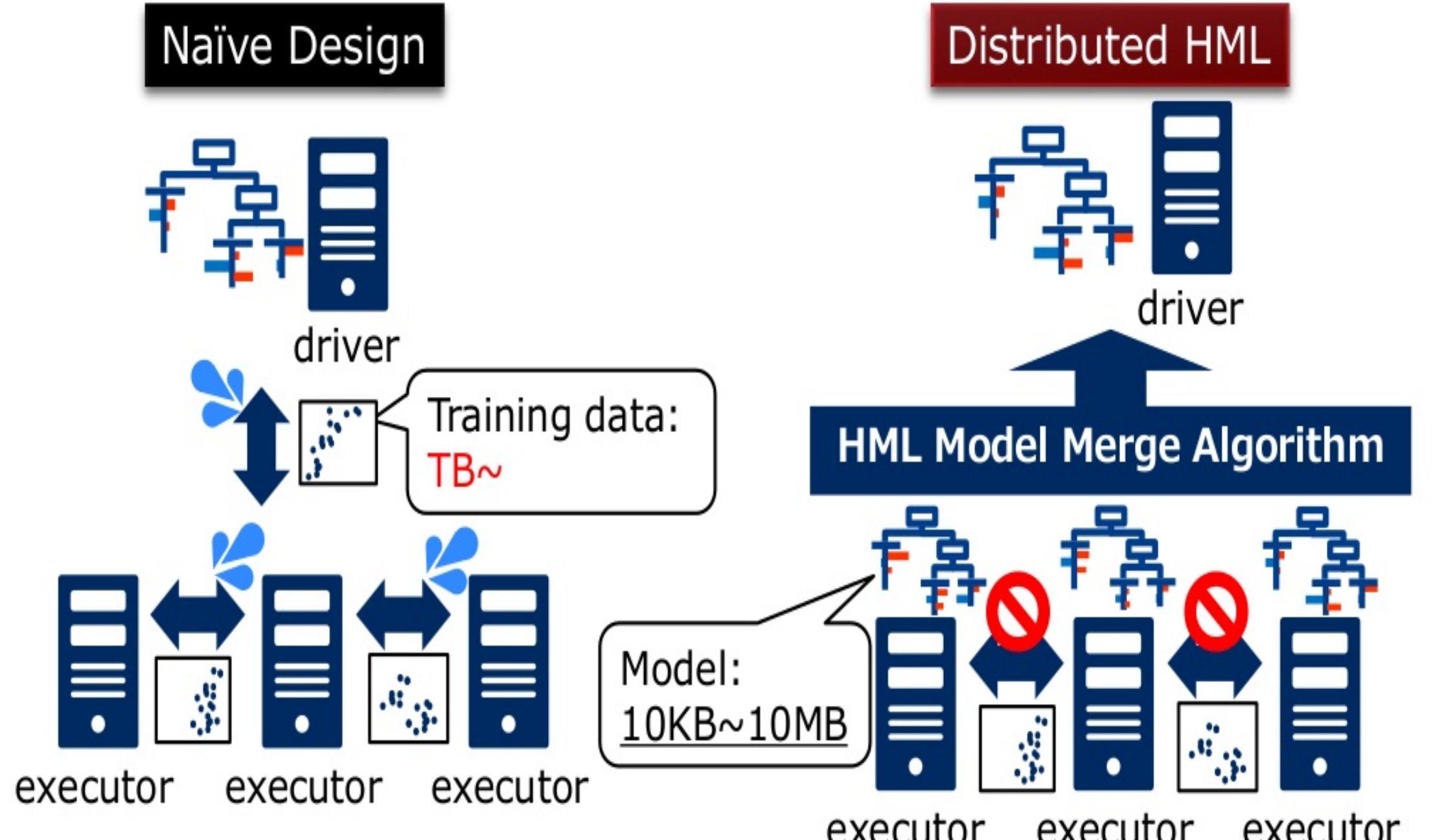
**MapReduce Synchronization**



**Matrix Computation**

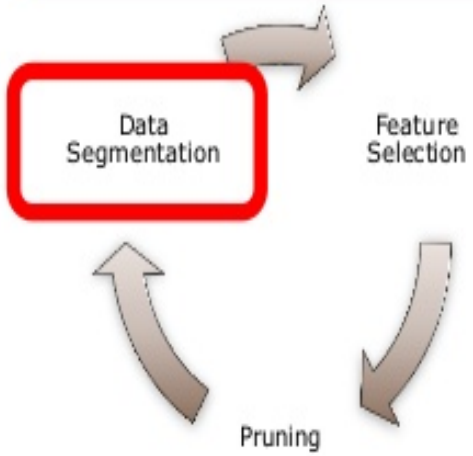


# Challenge to Avoid Data Shuffling



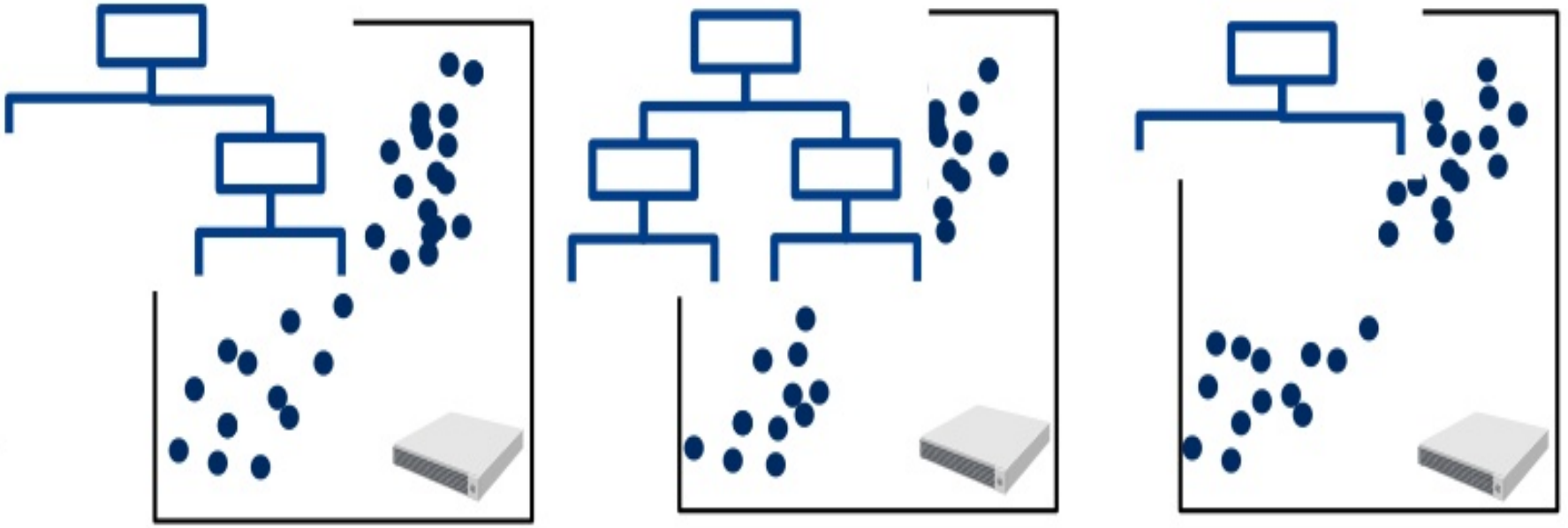
# Execution Flow of Distributed HML

## Executors build a rule-based tree from their local data in parallel



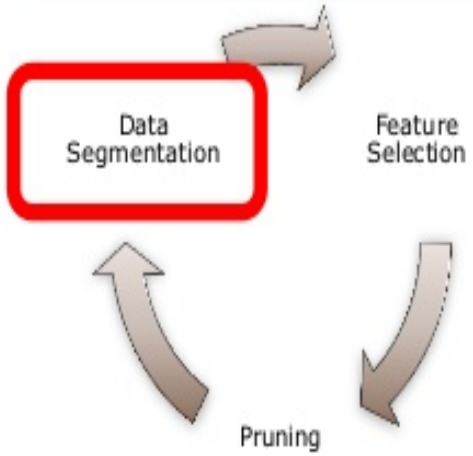

Driver

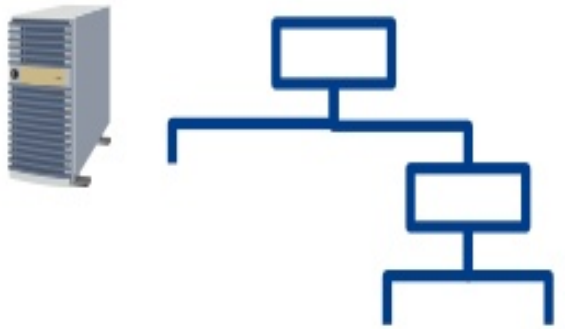
Executor



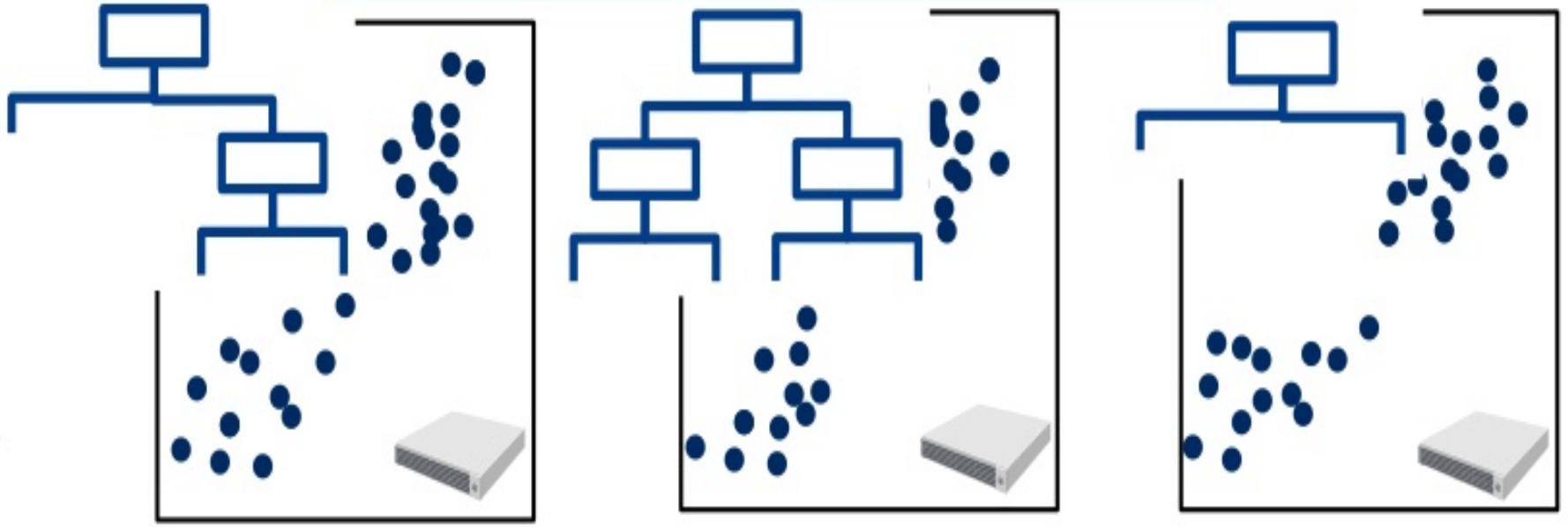
# Execution Flow of Distributed HML

## Driver merges the trees



**HML Segment Merge Algorithm**

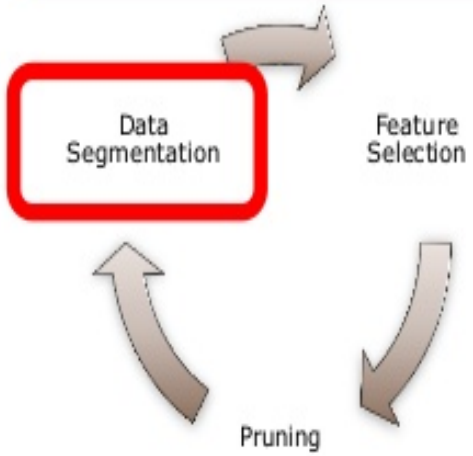


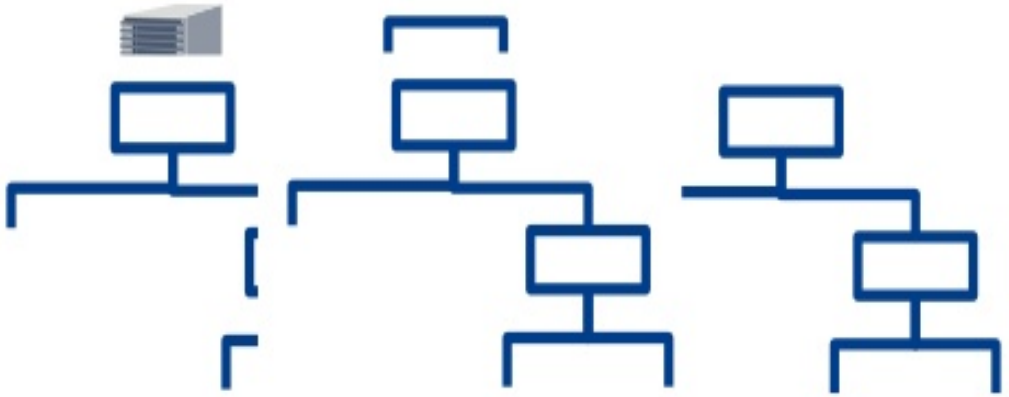
Driver

Executor

# Execution Flow of Distributed HML

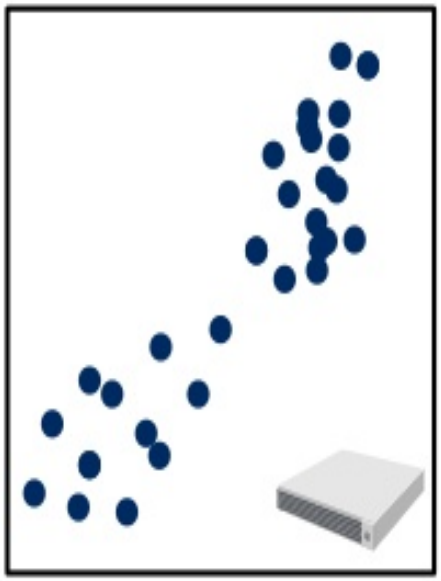
## Driver broadcasts the merged tree to executors

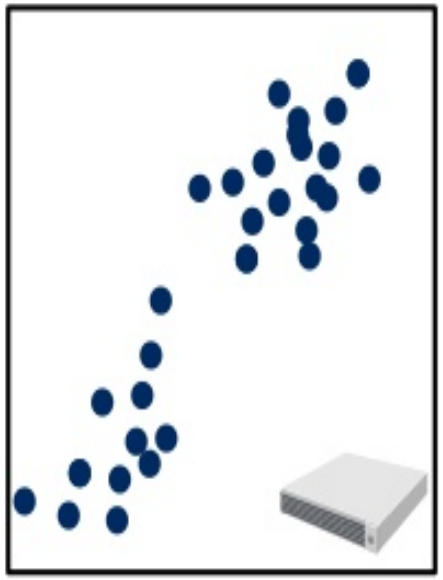


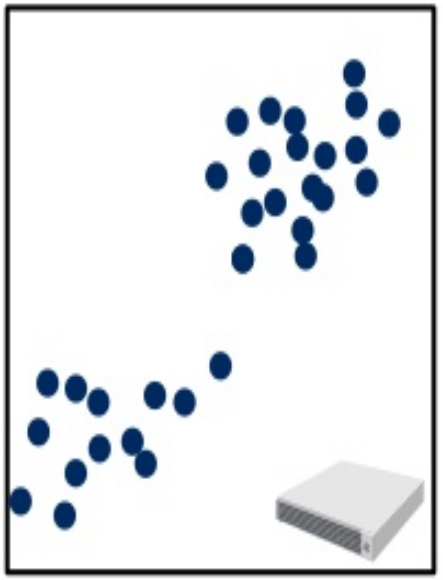



Driver

Executor

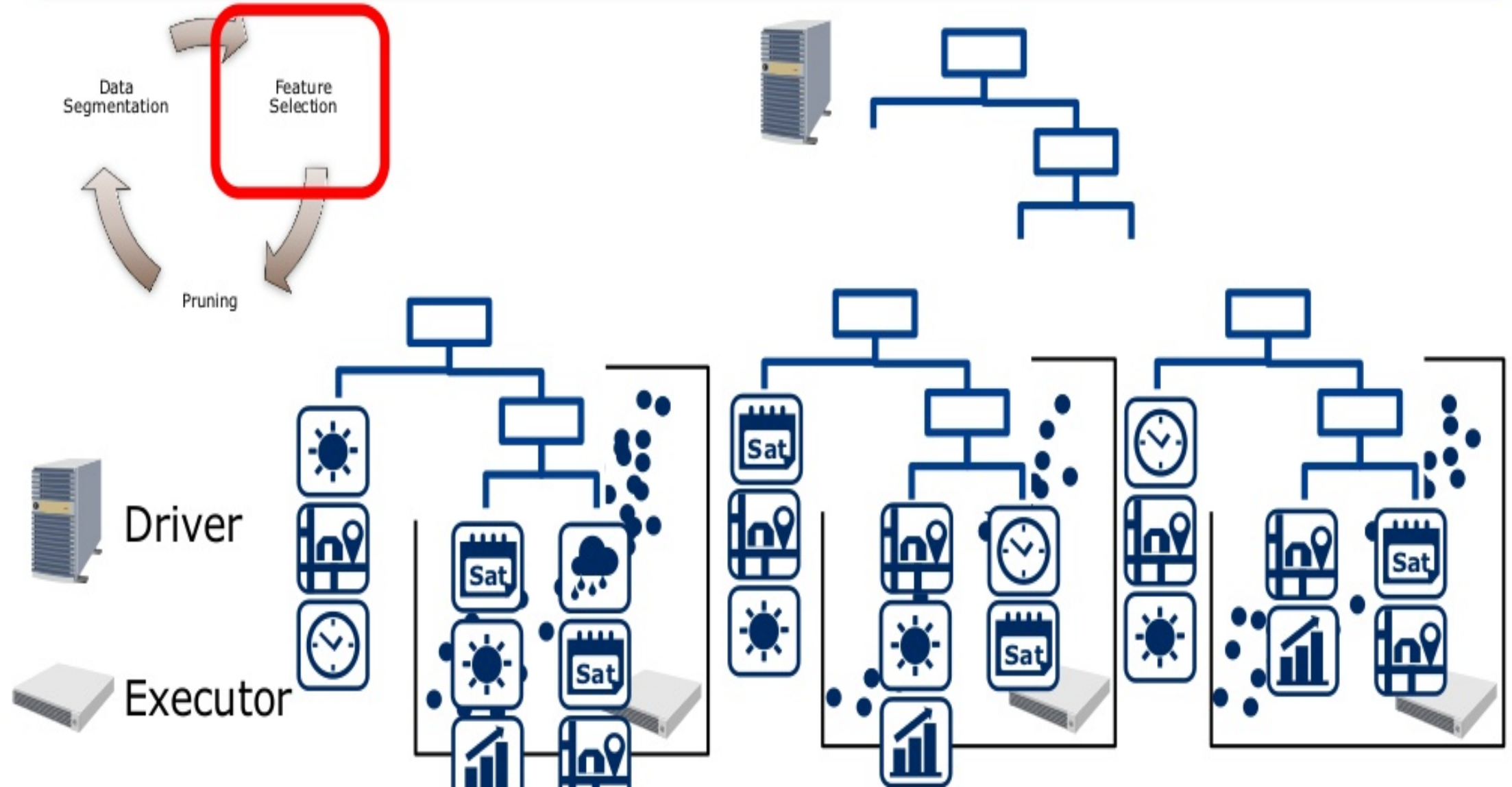






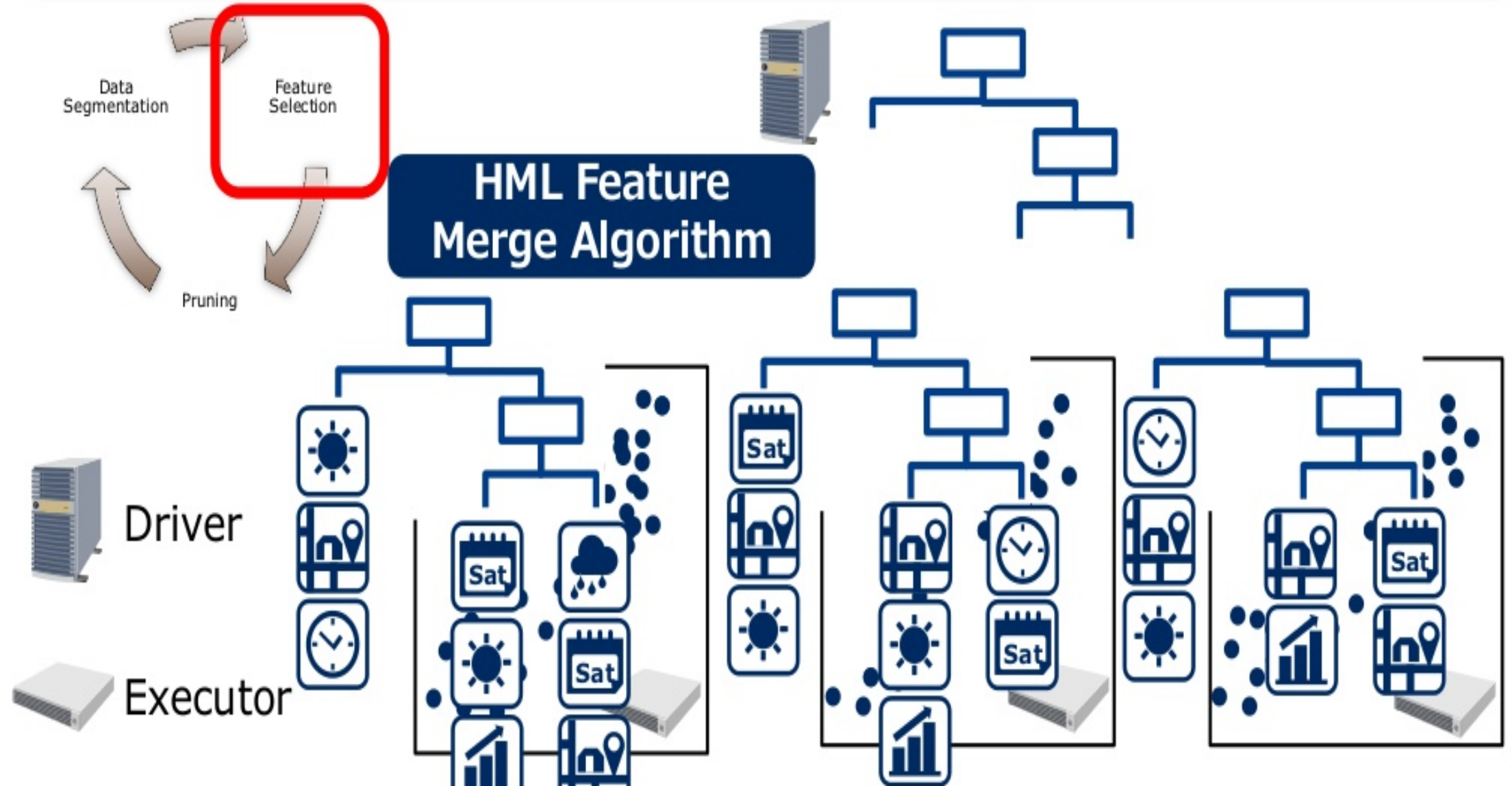
# Execution Flow of Distributed HML

## Executors perform feature selection with their local data in parallel



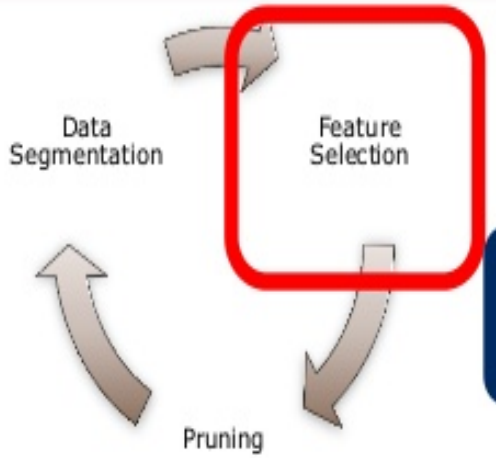
# Execution Flow of Distributed HML

## Driver merges the results of feature selection

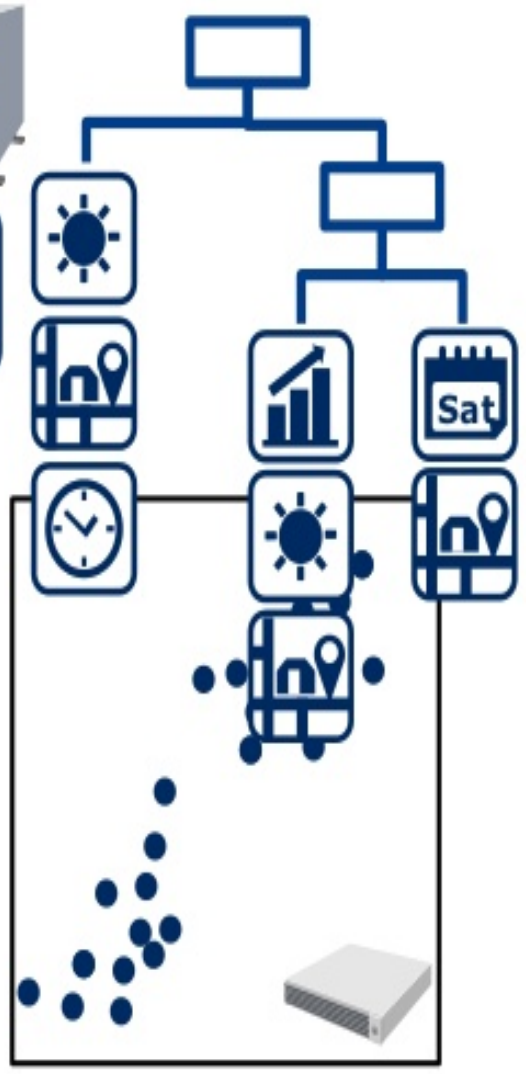


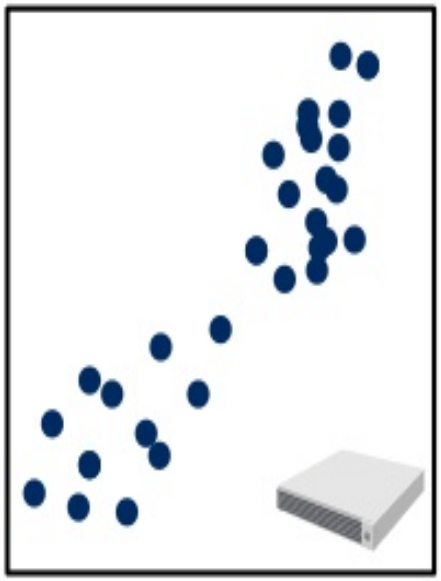
# Execution Flow of Distributed HML

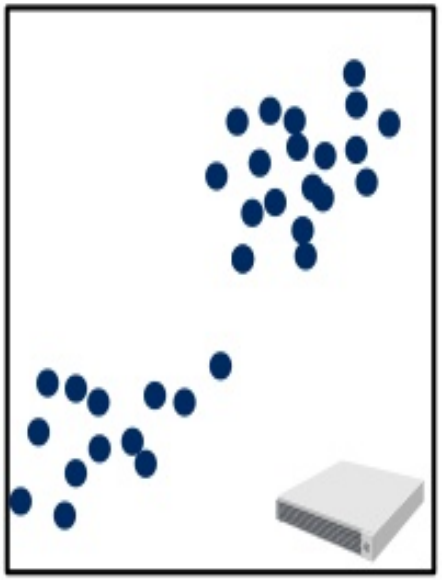
## Driver merges the results of feature selection




**HML Feature Merge Algorithm**



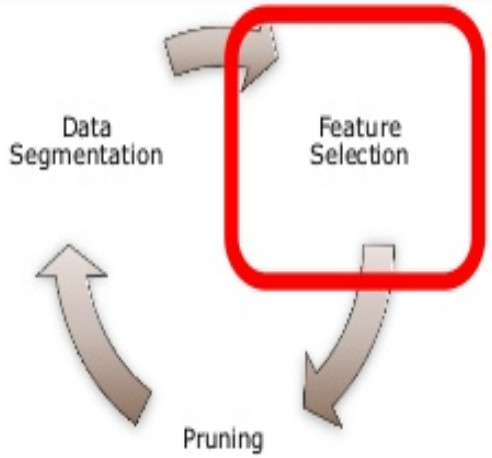


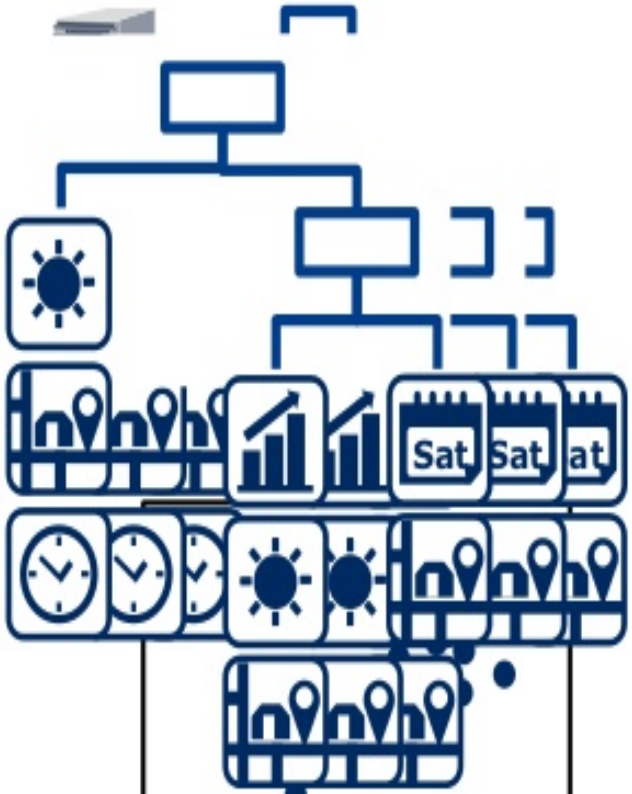
Driver

Executor

# Execution Flow of Distributed HML

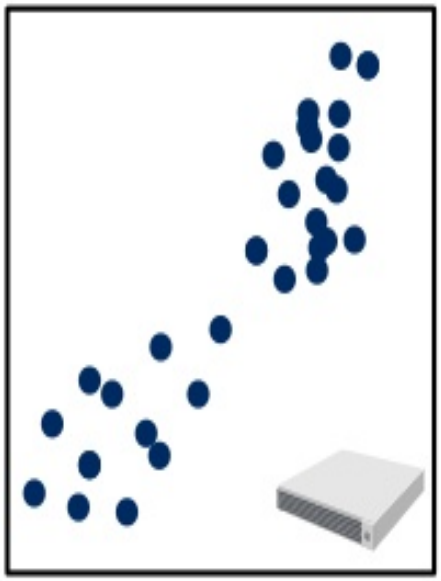
## Driver broadcasts the merged results of feature selection to executors



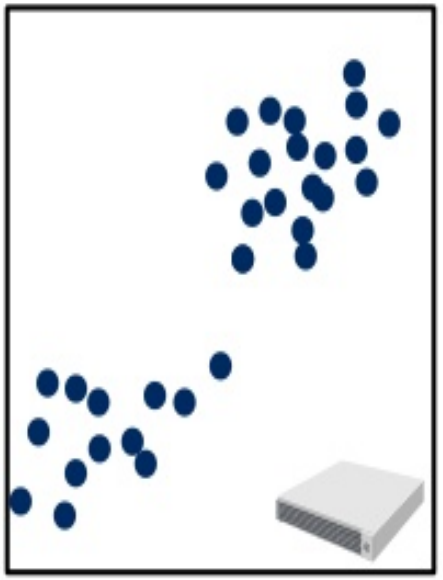




Driver

Executor



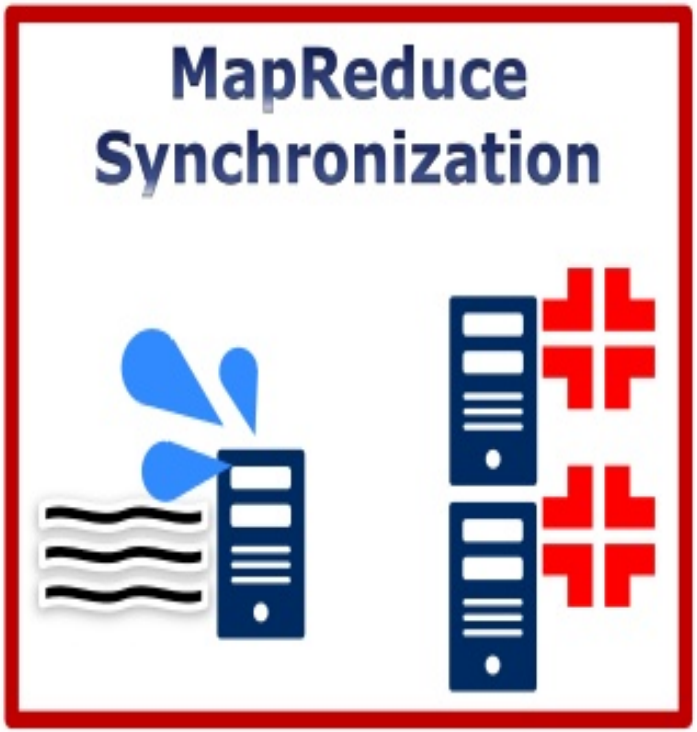




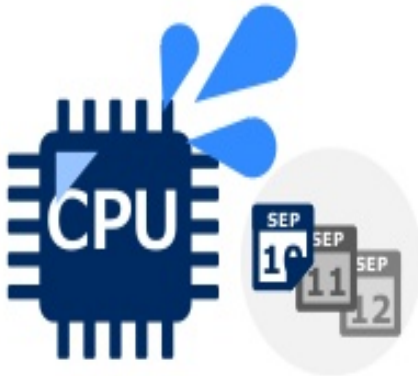
# 3 Technical Key Points to Fast Run ML on Spark

**Data Shuffling**

**MapReduce Synchronization**

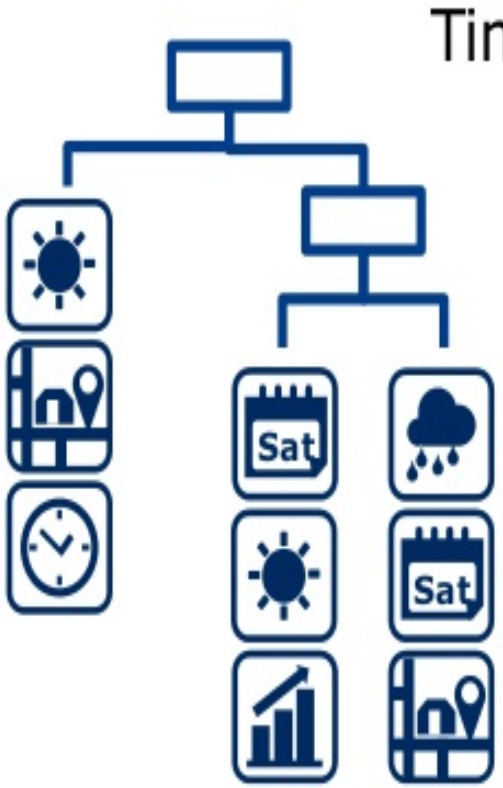


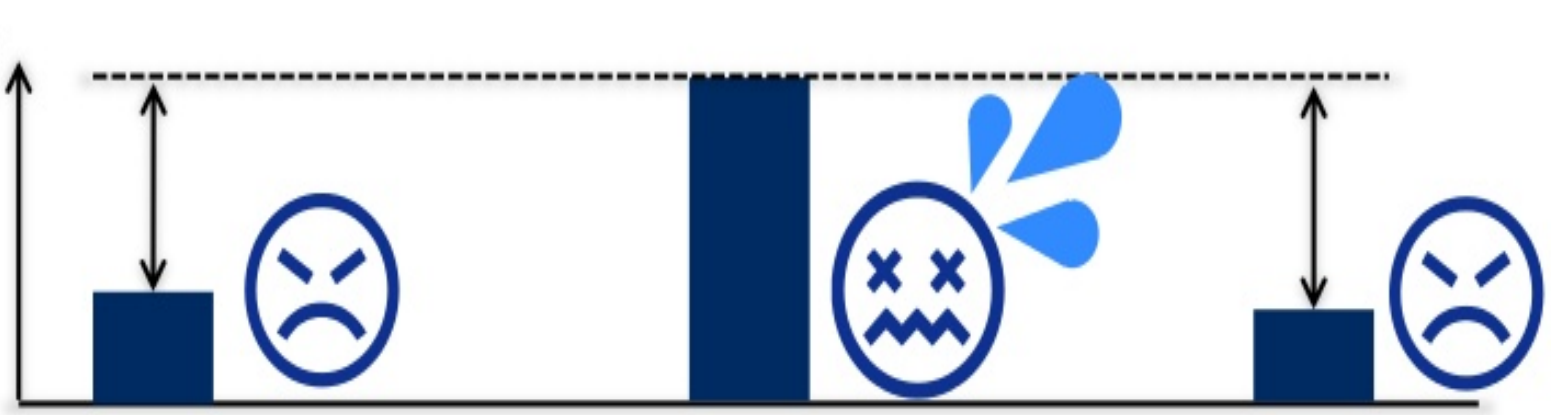
**Matrix Computation**



# Wait Time for Other Executors Delays Execution

## Machine learning is likely to cause unbalanced computation load




Executor

Executor

Executor

# Balance Computational Effort for Each Executor

Executor optimizes all predictive formulas with equally-divided training data



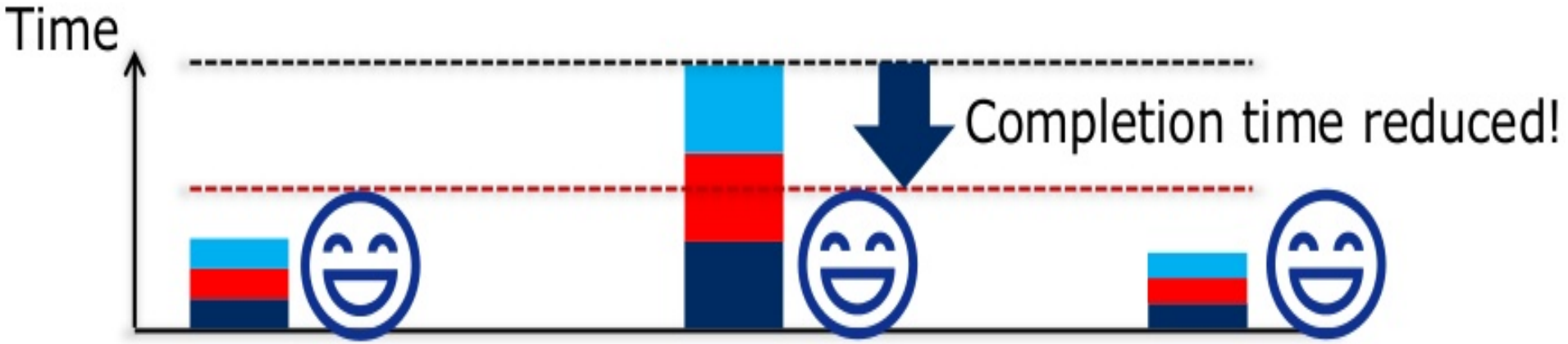
Executor

Executor

Executor

# Balance Computational Effort for Each Executor

Executor optimizes all predictive formulas with equally-divided training data



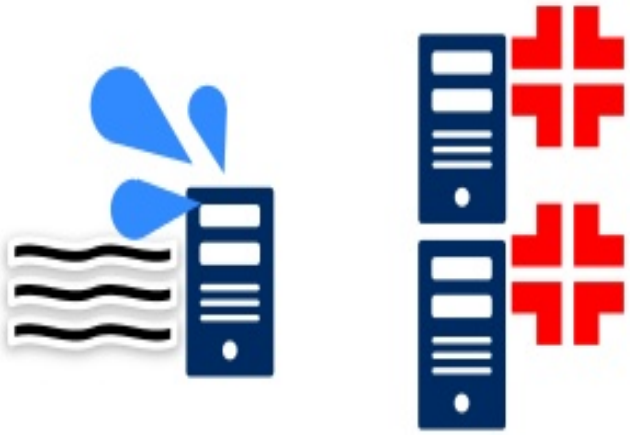


Executor Executor Executor

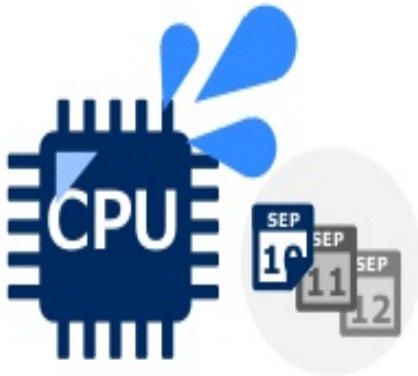
# 3 Technical Key Points to Fast Run ML on Spark

**Data Shuffling**

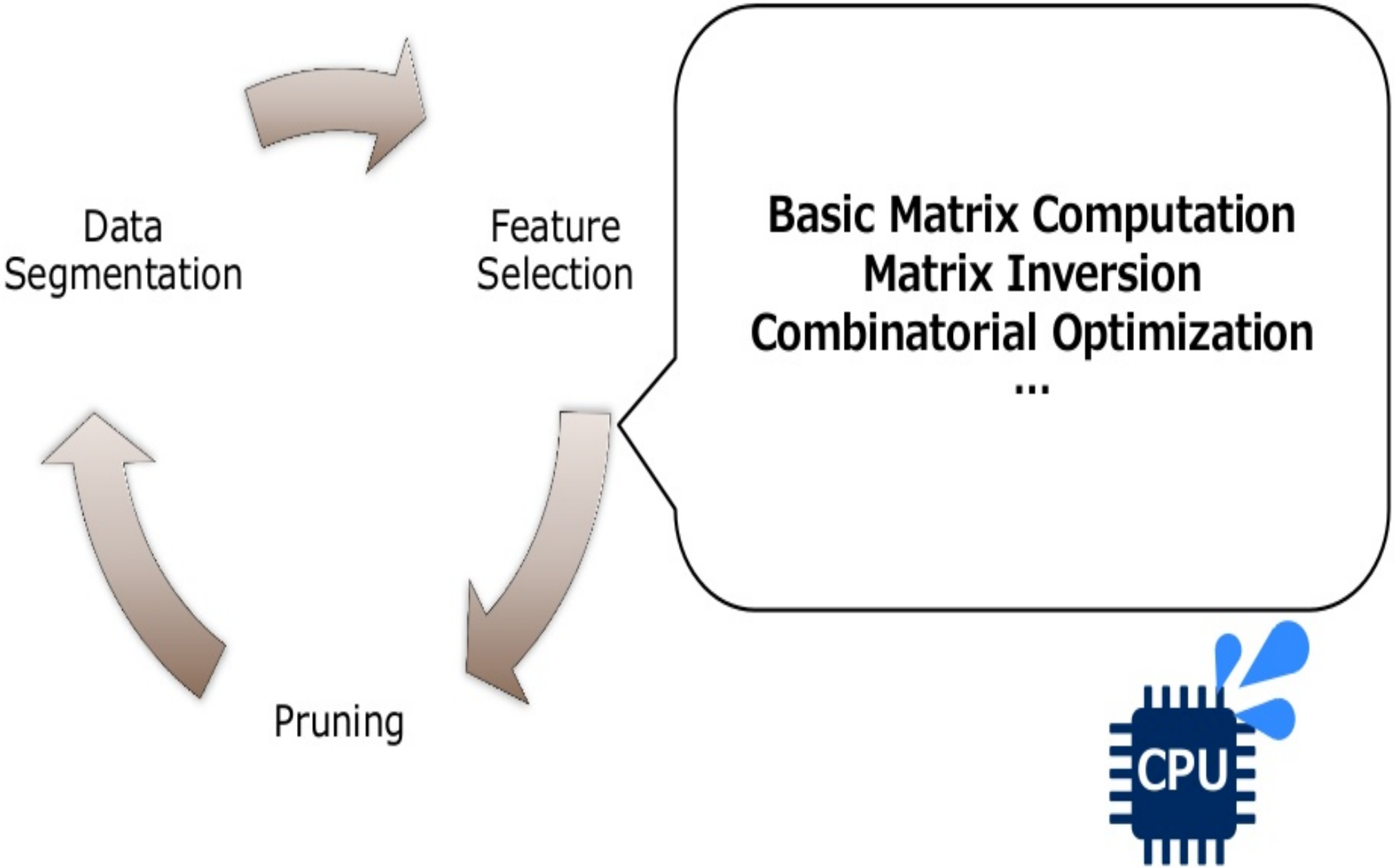

**MapReduce Synchronization**



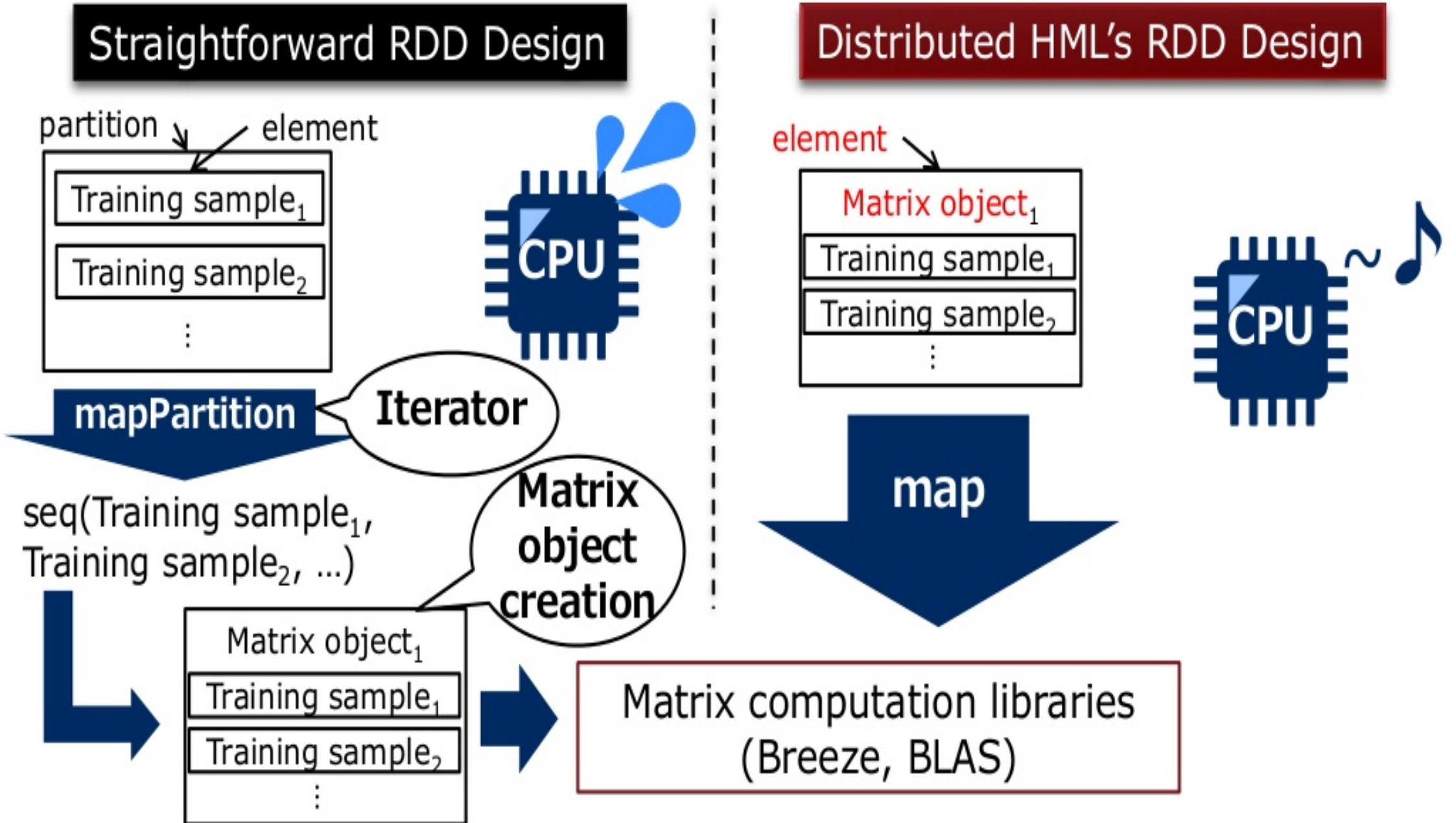
**Matrix Computation**



# Machine Learning Consists of Many Matrix Computations

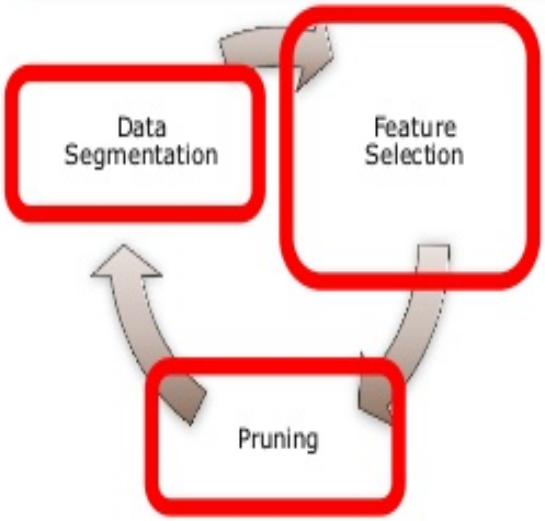


# RDD Design for Leveraging High-Speed Libraries



# Performance Degradation by Long-Chained RDD

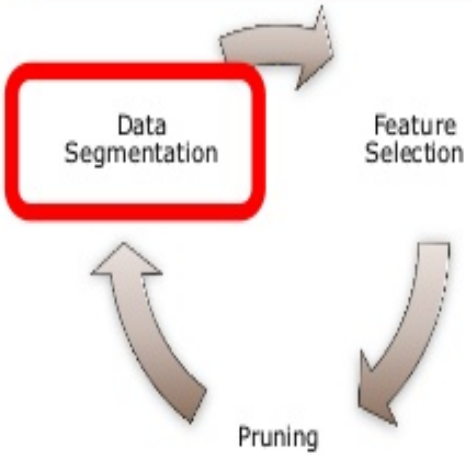
## Long-chained RDD operations cause high-cost recalculations



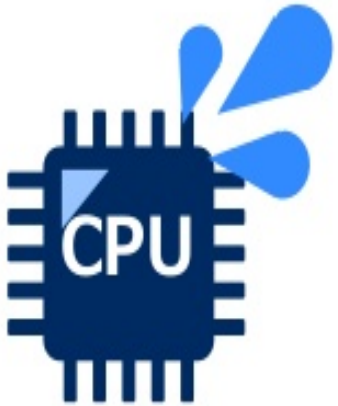

```
RDD.map(TreeMapFunc)
   .reduce(TreeReduceFunc)
   .map(FeatureSelectionMapFunc)
   .reduce(FeatureSelectionReduceFunc)
   .map(PruningMapFunc)
   .map(TreeMapFunc)
   .reduce(TreeReduceFunc)
   .map(FeatureSelectionMapFunc)
   .reduce(FeatureSelectionReduceFunc)
```

# Performance Degradation by Long-Chained RDD

## Long-chained RDD operations cause high-cost recalculations

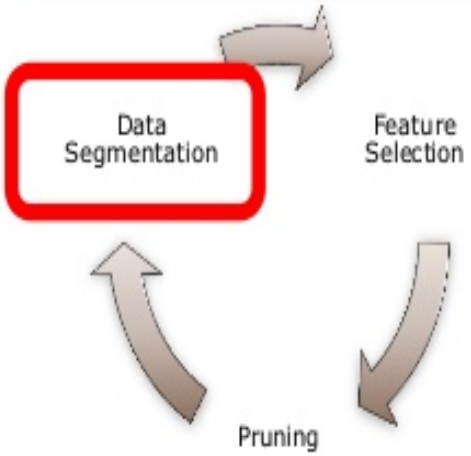



```
RDD.map(TreeMapFunc)
   .reduce(TreeReduceFunc)
   .map(FeatureSelectionMapFunc)
   .reduce(FeatureSelectionReduceFunc)
   .map(PruningMapFunc)
   .map(TreeMapFunc)
   .reduce(TreeReduceFunc)
   .map(FeatureSelectionMapFunc)
   .reduce(FeatureSelectionReduceFunc)
   .map(TreeMapFunc)
```

# Performance Degradation by Long-Chained RDD

## Cut long-chained operations periodically by checkpoint()




```
savedRDD = RDD.map(TreeMapFunc)
           .reduce(TreeReduceFunc)
           .map(FeatureSelectionMapFunc)
           .reduce(FeatureSelectionReduceFunc)
           .map(PruningMapFunc)
           .map(TreeMapFunc)
           .reduce(TreeReduceFunc)
           .map(FeatureSelectionMapFunc)
           .reduce(FeatureSelectionReduceFunc)
           .checkpoint()
           .map(TreeMapFunc)
```


# Benchmark Performance Evaluations

# Eval 1: Prediction Error (vs. Spark MLlib algorithms)

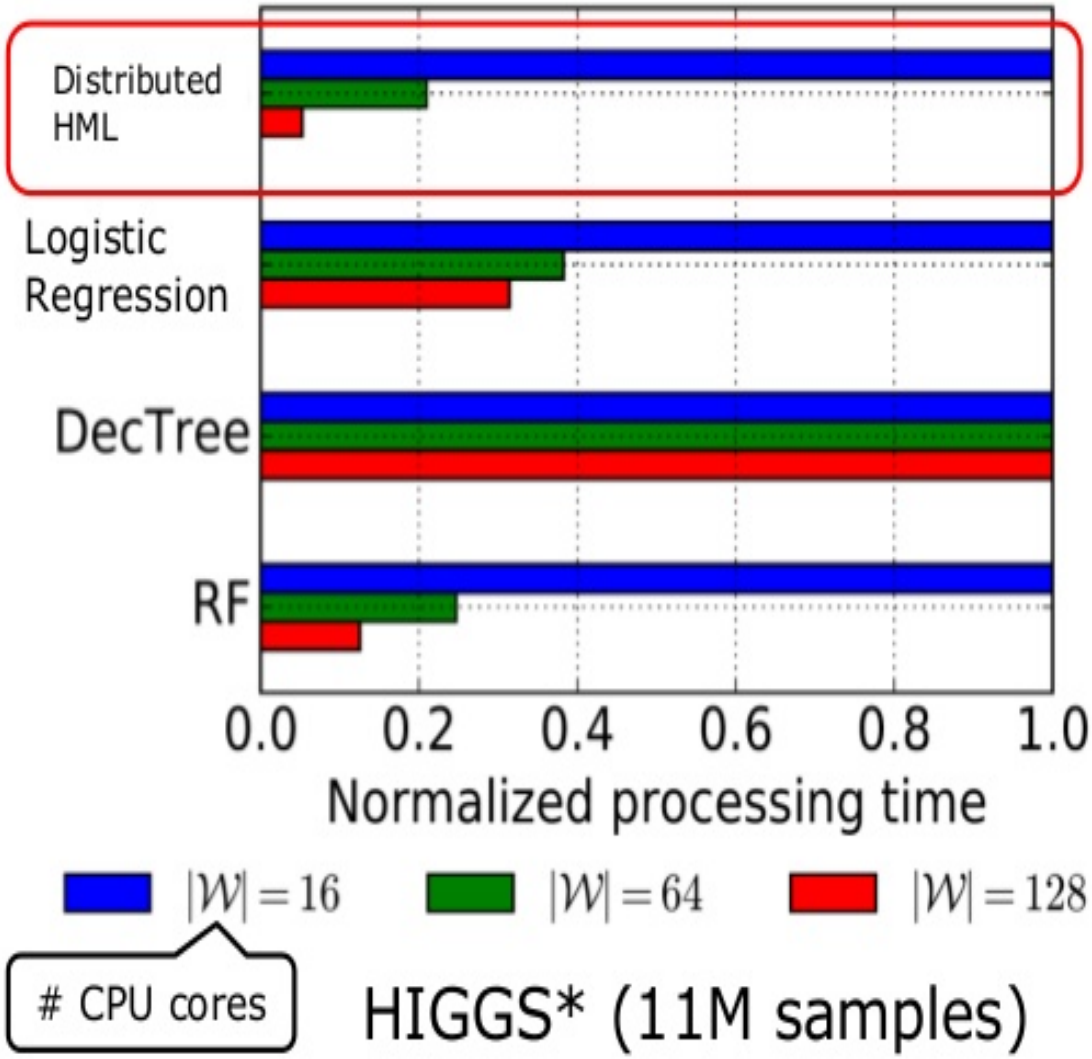
Distributed HML achieves low error competitive to a complex model

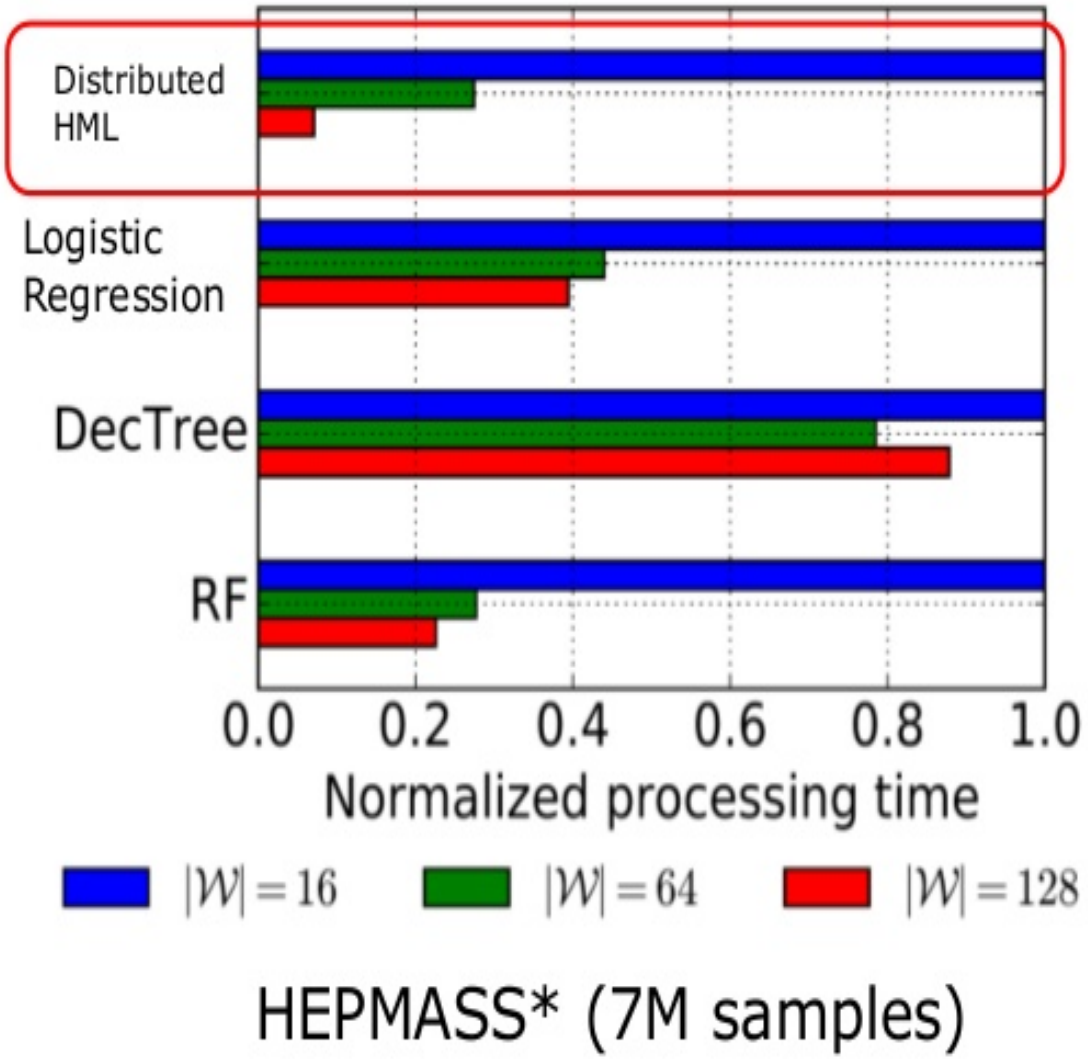
| data | # samples | Distributed HML | Logistic Regression | Decision Tree | Random Forests |
|---|---|---|---|---|---|
| gas sensor array (CO)* | 4,208,261 | 1st 0.542 | 0.597 | 0.587 | 0.576 |
| household power consumption * | 2,075,259 | 1st 0.524 | 0.531 | 0.529 | 0.655 |
| HIGGS* | 11,000,000 | 2nd 0.335 | 0.358 | 0.337 | 0.317 |
| HEPMASS* | 7,000,000 | 1st 0.156 | 0.163 | 0.167 | 0.175 |

* UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/)

# Eval 2: Performance Scalability (vs. Spark MLlib algos)

## Distributed HML is competitive with Spark MLlib implementations.




* UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/)

# Evaluation in Real Case

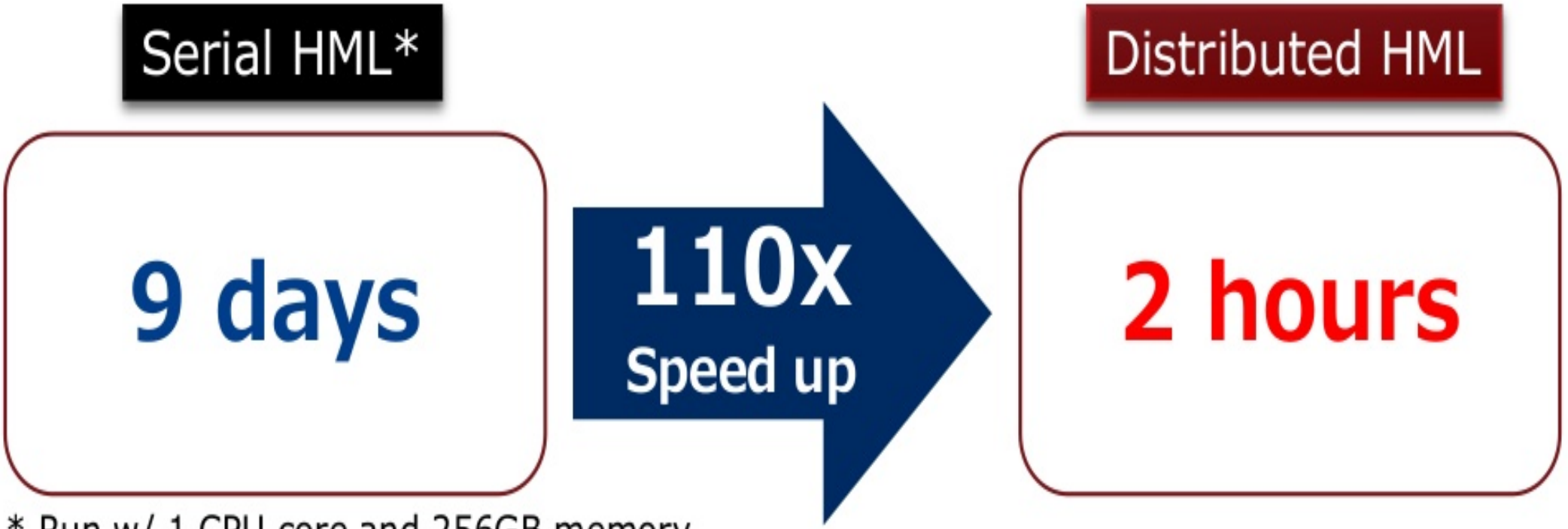
# ATM Cash Balance Prediction


Around 20,000 ATMs

Much cash → High inventory cost

Little cash → High risk of out of stock

# Training Speed




* Run w/ 1 CPU core and 256GB memory

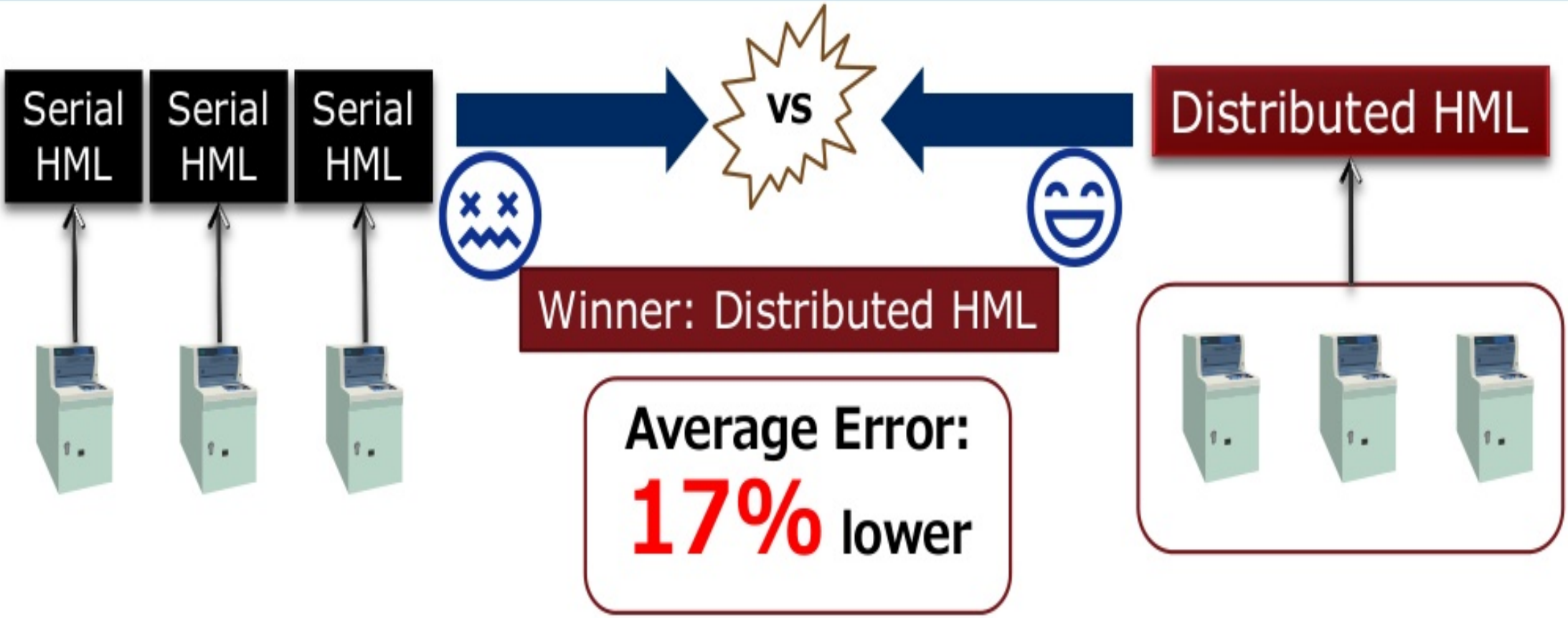
## Summary of data

- # ATMs: around 20,000 (in Japan)
- # training samples: around 10M

## Cluster spec (10 nodes)

- # CPU cores: 128
- Memory: 2.5TB
- Spark 1.6.1,
  Hadoop 2.7.2 (HDP 2.3.2)

# Prediction Error



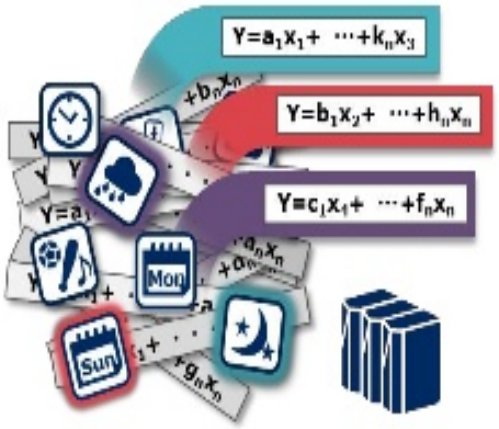

## Summary of data

- # ATMs: around 20,000 (in Japan)
- # training samples: around 23M

## Cluster spec (10 nodes)

- # CPU cores: 128
- Memory: 2.5TB
- Spark 1.6.1,
  Hadoop 2.7.2 (HDP 2.3.2)

# Summary

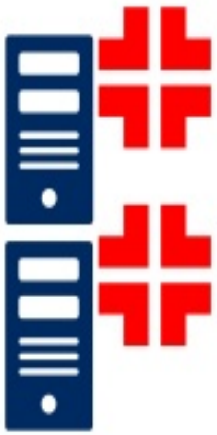


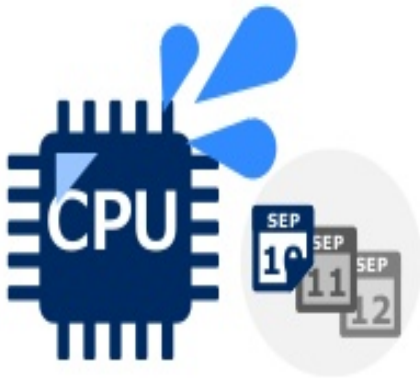


**Data Shuffling**

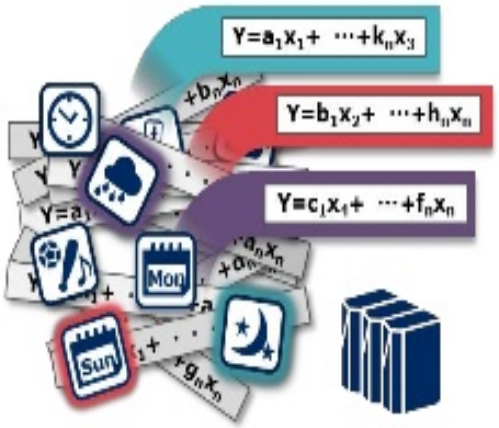
**MapReduce Synchronization**

**Matrix Computation**





# Summary




Spark










Orchestrating a brighter world
NEC