Scalable Distributed Decision Trees in Spark MLlib

Manish Amde, Origami Logic
Hirakendu Das, Yahoo! Labs
Evan Sparks, UC Berkeley
Ameet Talwalkar, UC Berkeley

Who is this guy?

- Ph.D. in ECE from UC San Diego
- Currently, data science at Origami Logic
- Origami Logic
  - Search-based marketing intelligence platform
  - Work with global brands on large, unstructured marketing datasets
  - Using Spark for analytics

Overview

- Decision Tree 101
- Distributed Decision Trees in MLlib
- Experiments
- Ensembles
- Future work

Supervised Learning

Train

Predict

Classification / Regression

- Classification
  - Labels denote classes

- Regression
  - Labels denote real numbers

Car mileage from 1971!

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high
76	high	low
88	high	low

Learn a model to predict the mileage
(Binary classification!)

Let's Learn: Rule 1

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high
76	high	low
88	high	low

Let's Learn: Rule 1

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high
76	high	low
88	high	low

Let's Learn: Rule 1

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high
76	high	low
88	high	low

If weight is high, mileage is low

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

$labels : \{high : 4, low : 2\}$

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

$labels : \{high : 4, low : 2\}$

Training Data

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

$labels : \{high : 4, low : 2\}$

Split Candidates

Training Data

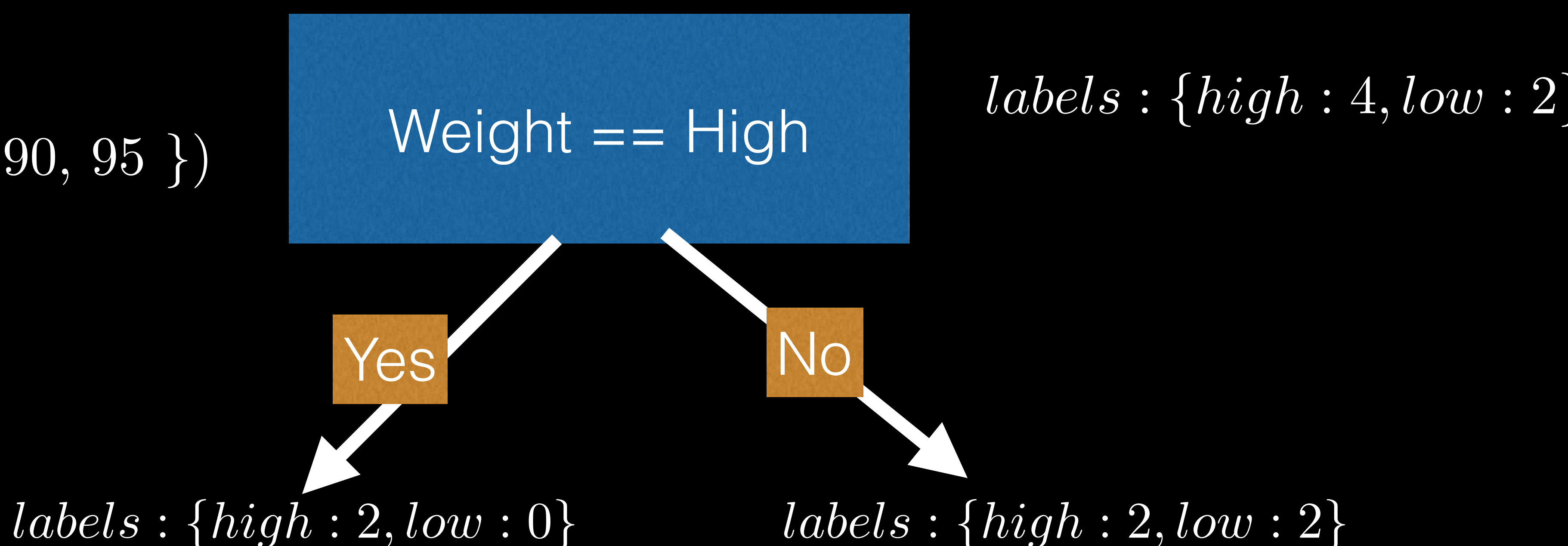
Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

$labels : \{high : 4, low : 2\}$

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })



Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

Weight == High

$labels : \{high : 4, low : 2\}$

Yes

$labels : \{high : 2, low : 0\}$

No

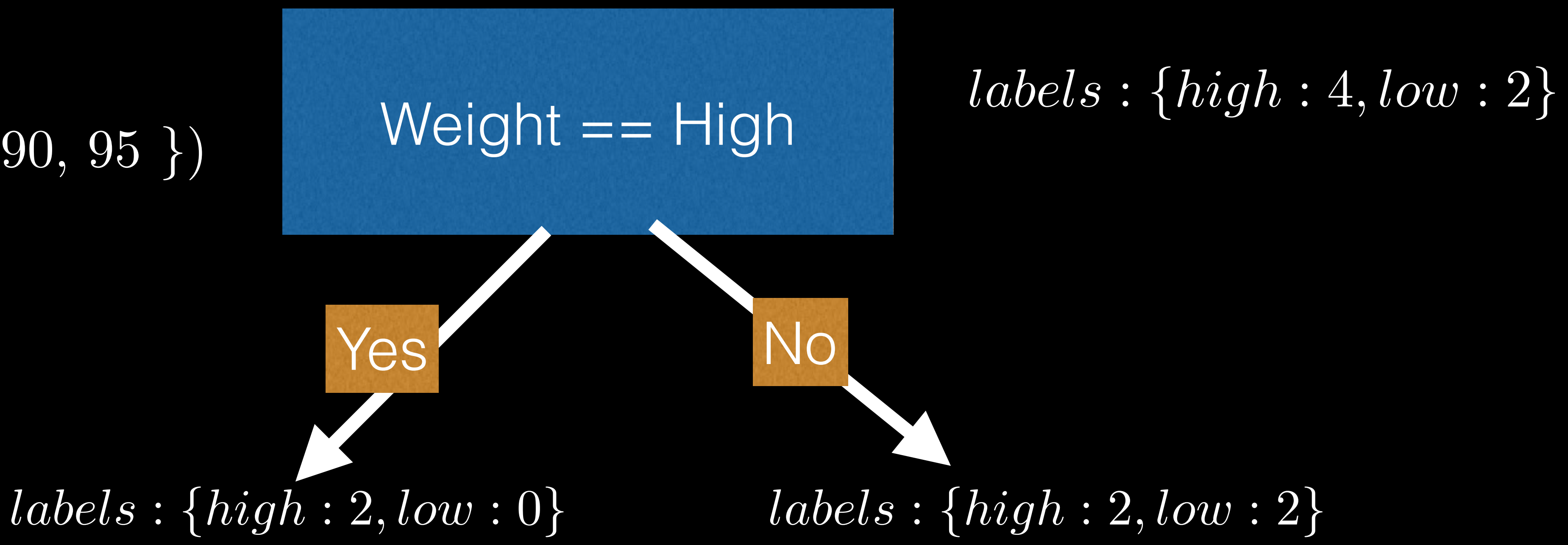
$labels : \{high : 2, low : 2\}$

Chose a split that causes maximum reduction in the label variability

Chose a split that maximizes "information gain"

Find Best Split

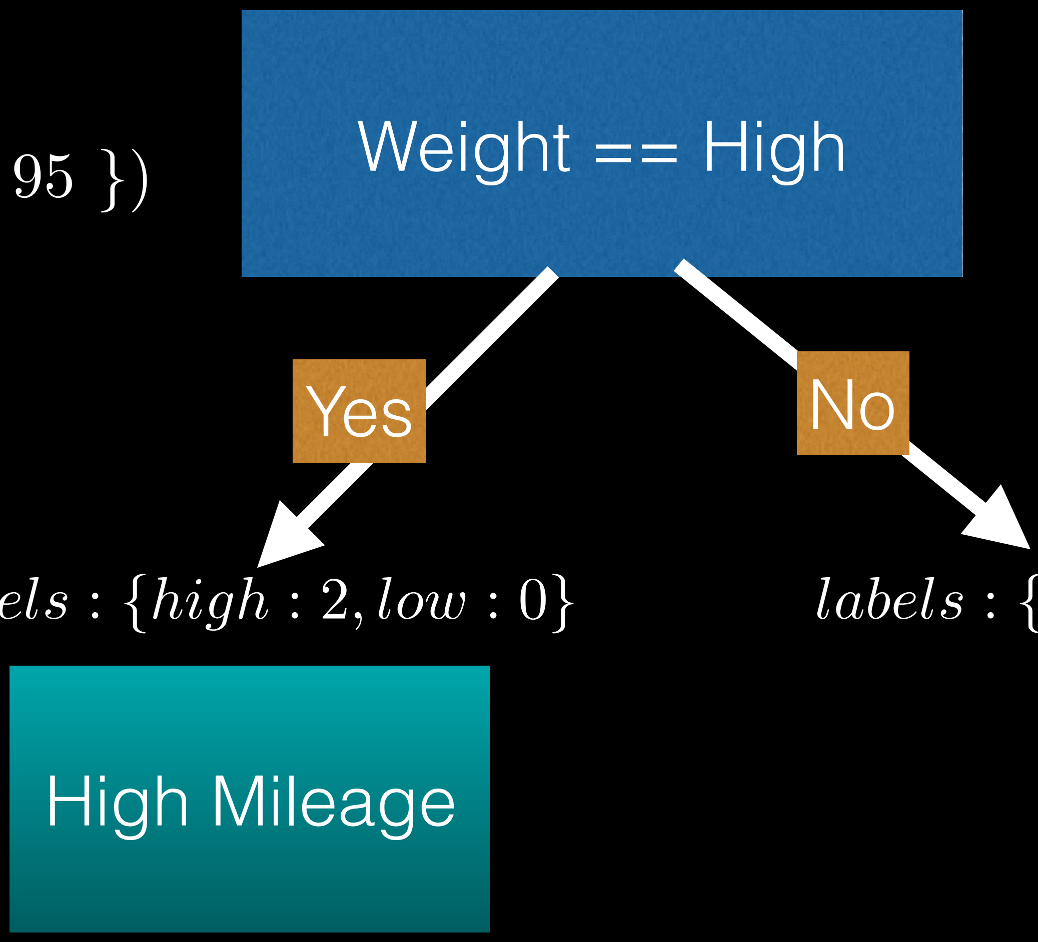
(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })



No increase in information gain possible

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })



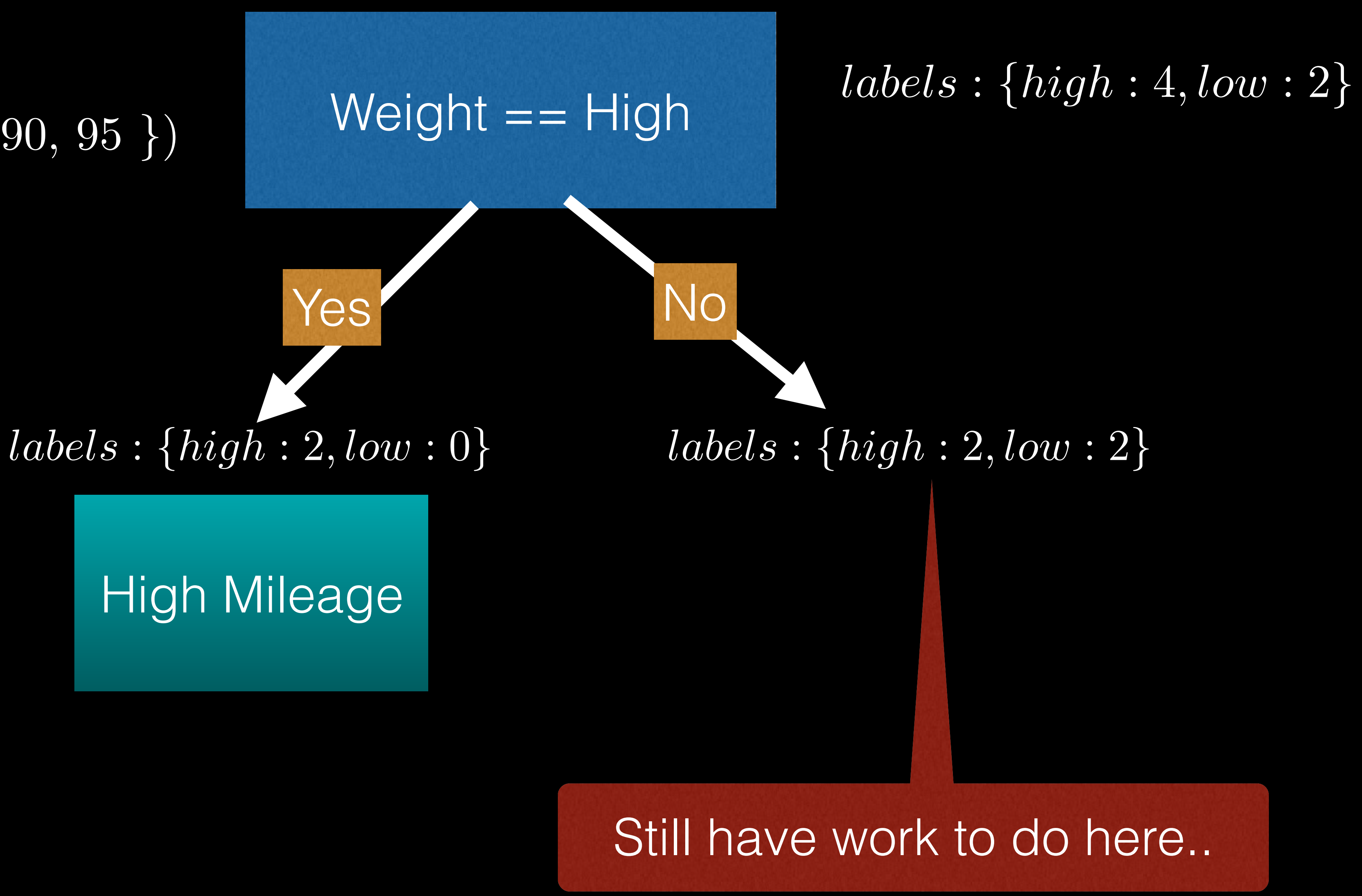
$labels : \{high : 4, low : 2\}$

$labels : \{high : 2, low : 0\}$

$labels : \{high : 2, low : 2\}$

Find Best Split

(weight, {low,high})
(hp, {70, 76, 86, 88, 90, 95 })

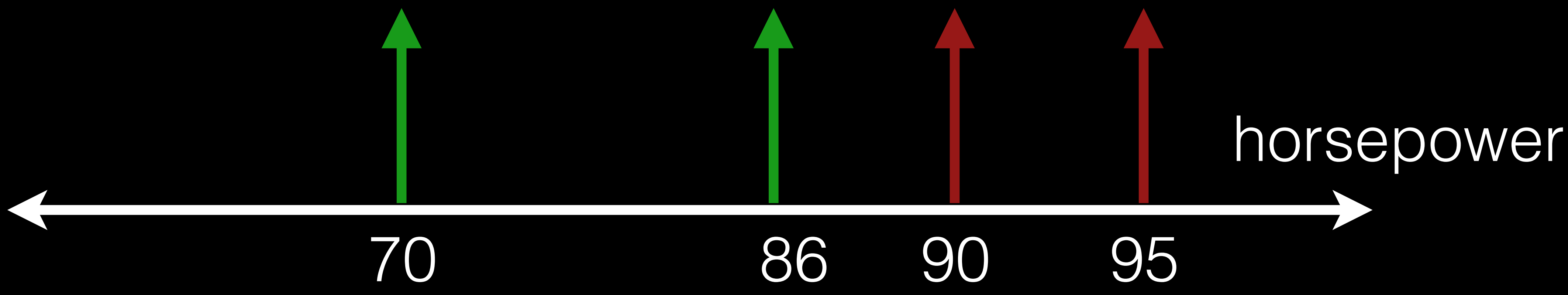


Let's Learn: Rule 2

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high

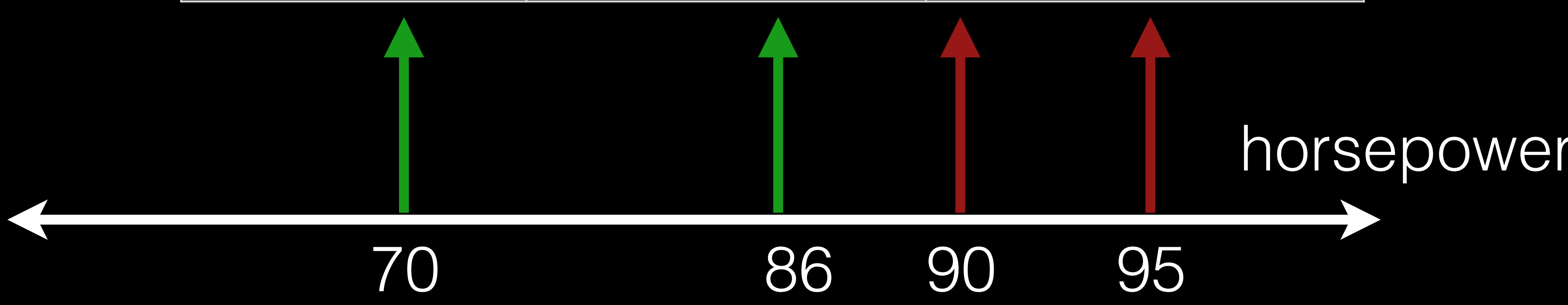
Let's Learn: Rule 2

horsepower	weight	mileage
95	low	low
90	low	low
70	low	high
86	low	high



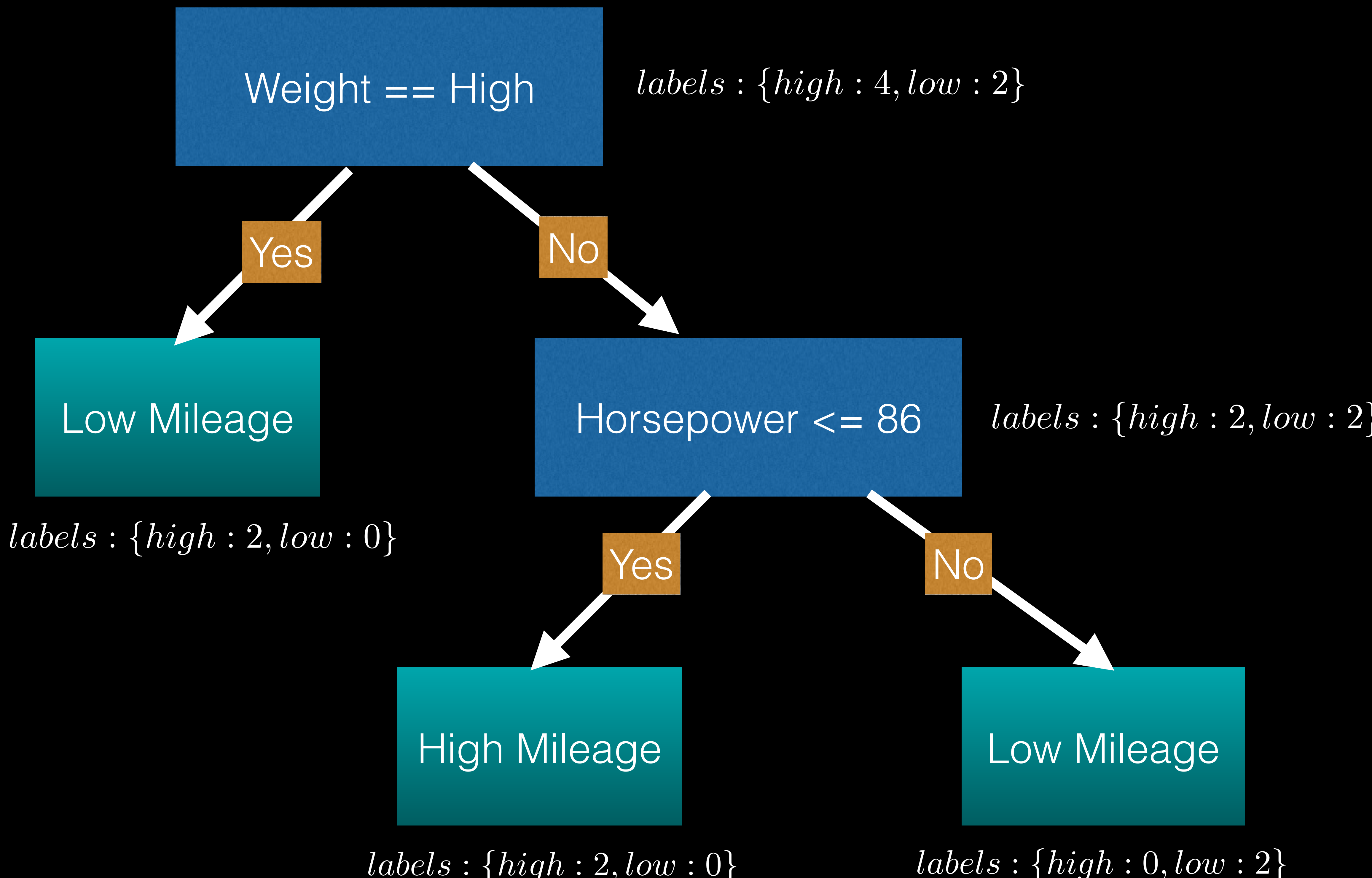
Let’s Learn: Rule 2

horsepower	weight	mileage
95	**low**	**low**
90	**low**	**low**
70	**low**	**high**
86	**low**	**high**



If horsepower <= 86, mileage is high. Else, it’s low.

Mileage Classification Tree



Let's predict

horsepower	weight	mileage prediction
90	**high**	
80	**low**	
70	**high**	

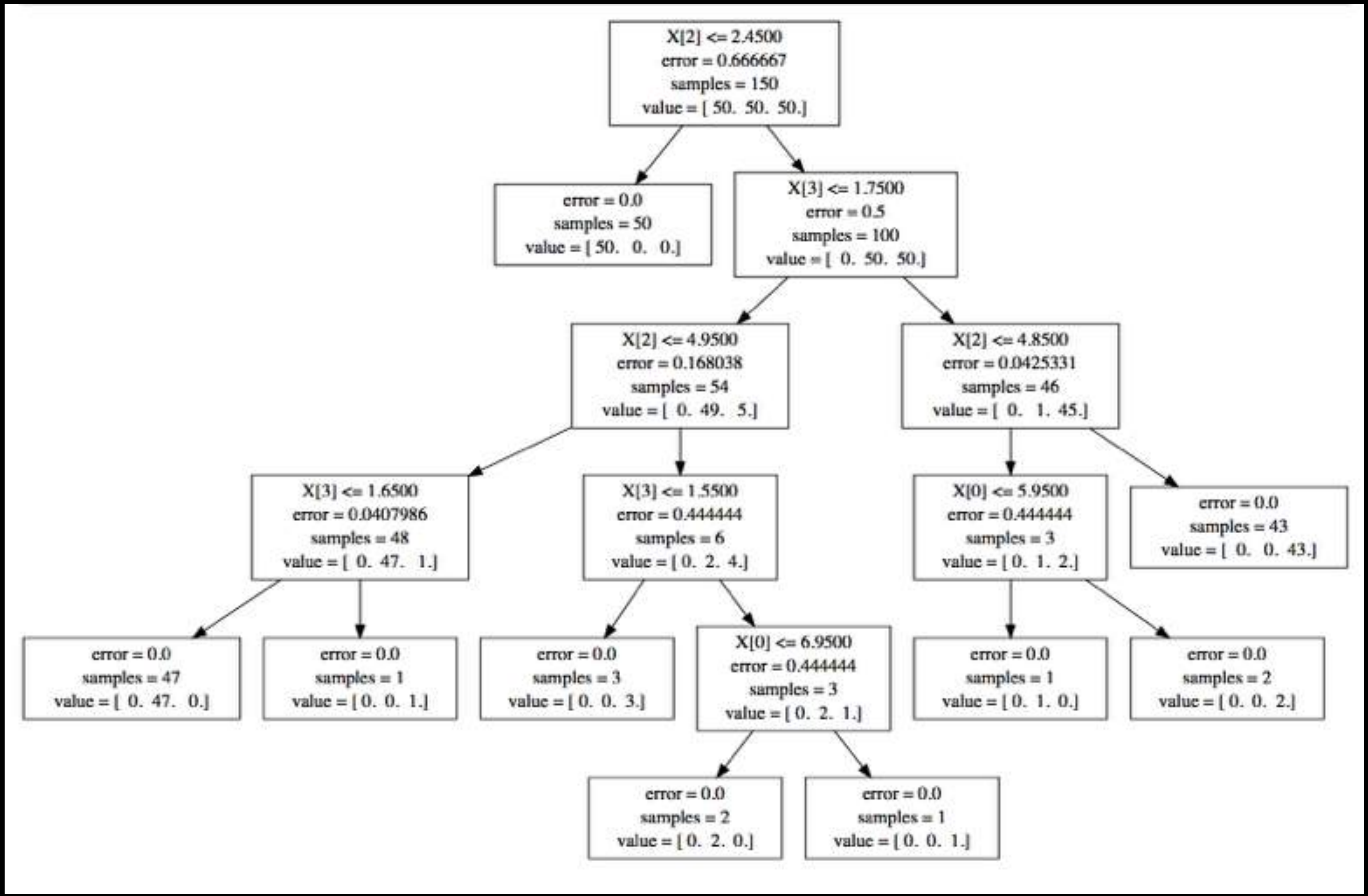
Let's predict

horsepower	weight	mileage prediction
90	high	low
80	low	low
70	high	high

Let’s predict

horsepower	weight	mileage prediction	
90	**high**	**low**	Correct!
80	**low**	**low**	Correct!
70	**high**	**high**	Wrong!

Complex in Practice



Why Decision Trees?

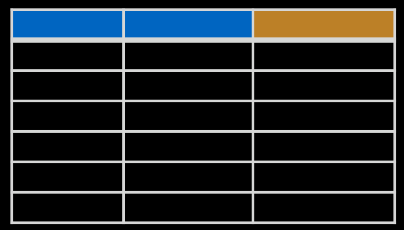
- Easy to interpret
- Handle categorical variables
- (Multi-class) classification and regression
- No feature scaling
- Capture non-linearities and feature interactions
- Handle missing values
- Ensembles are top performers

Overview

- Decision Tree 101
- Distributed Decision Trees in MLlib
- Experiments
- Ensembles
- Future work

Dataset: Single Machine

- Typically, dataset is loaded in memory as a matrix or dataframe
- Perform multiple passes over the data
- R, scikit-learn, …



Distributed Dataset



hp	weight	mileage
95	low	low
90	low	low

hp	weight	mileage
70	low	high
86	low	high

hp	weight	mileage
76	high	low
88	high	low

Distributed Dataset

Learn multiple models and combine them

hp	weight	mileage
95	low	low
90	low	low

hp	weight	mileage
70	low	high
86	low	high

hp	weight	mileage
76	high	low
88	high	low

Distributed Dataset

Learn multiple models and combine them

hp	weight	mileage
95	low	low
90	low	low

hp	weight	mileage
70	low	high
86	low	high

hp	weight	mileage
76	high	low
88	high	low

Does not work well for all data partitioning
Still need inter-machine communication to combine models

Distributed Dataset

Distributed Dataset

- Hadoop MapReduce
  - No implementations when we started
  - Currently: RHadoop, Oryx, OxData,….

Distributed Dataset

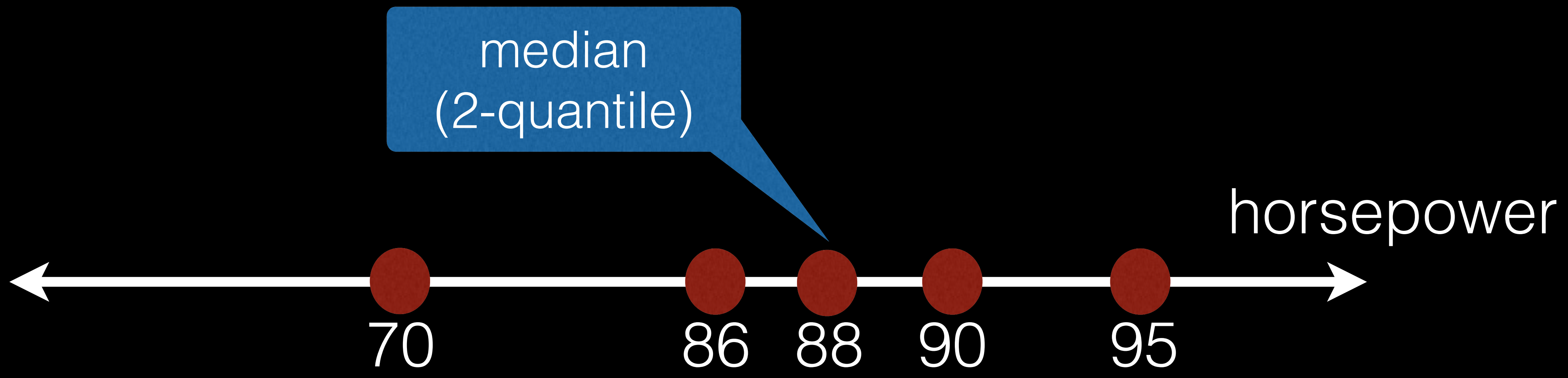
- Hadoop MapReduce
  - No implementations when we started
  - Currently: RHadoop, Oryx, OxData,….
- PLANET
  - Decision trees using MapReduce
  - Not open source
  - Extend with several optimizations

Distributed Dataset

- Hadoop MapReduce
  - No implementations when we started
  - Currently: RHadoop, Oryx, OxData,….
- PLANET
  - Decision trees using MapReduce
  - Not open source
  - Extend with several optimizations
- Spark
  - Iterative machine learning
  - No trees support in initial versions

Split Candidates for Distributed Implementation

- Splits candidates for continuous features
  - Costly to find all unique feature values
  - Sorted splits desirable for fast computation
  - High cardinality of splits leads to significant computation and communication overhead
- Approximate quantiles (percentiles by default)



Typical MapReduce Implementation: Algorithm

flatMap
input: instance
output: list(split, label)

reduceByKey
input: split, list(label)
output: split, labelHistograms

Typical MapReduce Implementation: Example

flatMap

reduceByKey

Typical MapReduce Implementation: Example

flatMap

hp	weight	mileage
76	**high**	**low**

reduceByKey

Typical MapReduce Implementation: Example

flatMap

hp	weight	mileage
76	**high**	**low**

$$(\text{weight}, \text{high}), \text{low}$$
$$(\text{hp}, 76), \text{low}$$
$$(\text{hp}, 86), \text{low}$$
$$(\text{hp}, 88), \text{low}$$
$$(\text{hp}, 90), \text{low}$$
$$(\text{hp}, 95), \text{low}$$

reduceByKey

Typical MapReduce Implementation: Example

flatMap

hp	weight	mileage
76	**high**	**low**

$$(\text{weight},\ \text{high}),\ \text{low}$$
$$(\text{hp},\ 76),\ \text{low}$$
$$(\text{hp},\ 86),\ \text{low}$$
$$(\text{hp},\ 88),\ \text{low}$$
$$(\text{hp},\ 90),\ \text{low}$$
$$(\text{hp},\ 95),\ \text{low}$$

reduceByKey

$$(\text{weight},\ \text{high}),\ [\text{low},\ \text{low}]$$

Typical MapReduce Implementation: Example

flatMap

hp	weight	mileage
76	**high**	**low**

$$(\text{weight, high}), \text{low}$$
$$(\text{hp}, 76), \text{low}$$
$$(\text{hp}, 86), \text{low}$$
$$(\text{hp}, 88), \text{low}$$
$$(\text{hp}, 90), \text{low}$$
$$(\text{hp}, 95), \text{low}$$

reduceByKey

$(\text{weight, high}), [\text{low, low}]$ $(\text{weight, high}), \{\text{low: } 2, \text{ high: } 0\}$

Typical MapReduce Implementation: Example

flatMap

hp	weight	mileage
76	high	low

$$(\text{weight, high}), \text{low}$$
$$(\text{hp}, 76), \text{low}$$
$$(\text{hp}, 86), \text{low}$$
$$(\text{hp}, 88), \text{low}$$
$$(\text{hp}, 90), \text{low}$$
$$(\text{hp}, 95), \text{low}$$

reduceByKey

$$(\text{weight, high}), [\text{low, low}]$$

$$(\text{weight, high}), \{\text{low: } 2, \text{ high: } 0\}$$

$$(\text{weight, !high}), \{\text{low: } 2, \text{ high: } 2\}$$

Typical MapReduce Implementation: Issues

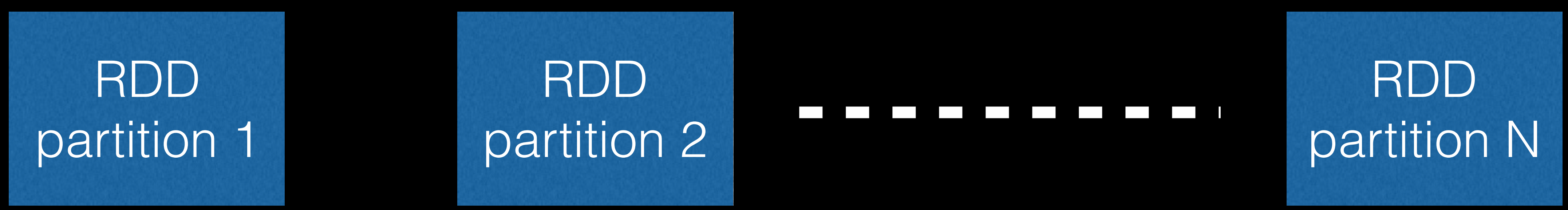
- For k features, m splits/feature and n instances, the map operation *emits* O(k*m*n) values per best split computation at a node
  - Communication overhead
- Can we do better?

Avoiding Map in MapReduce

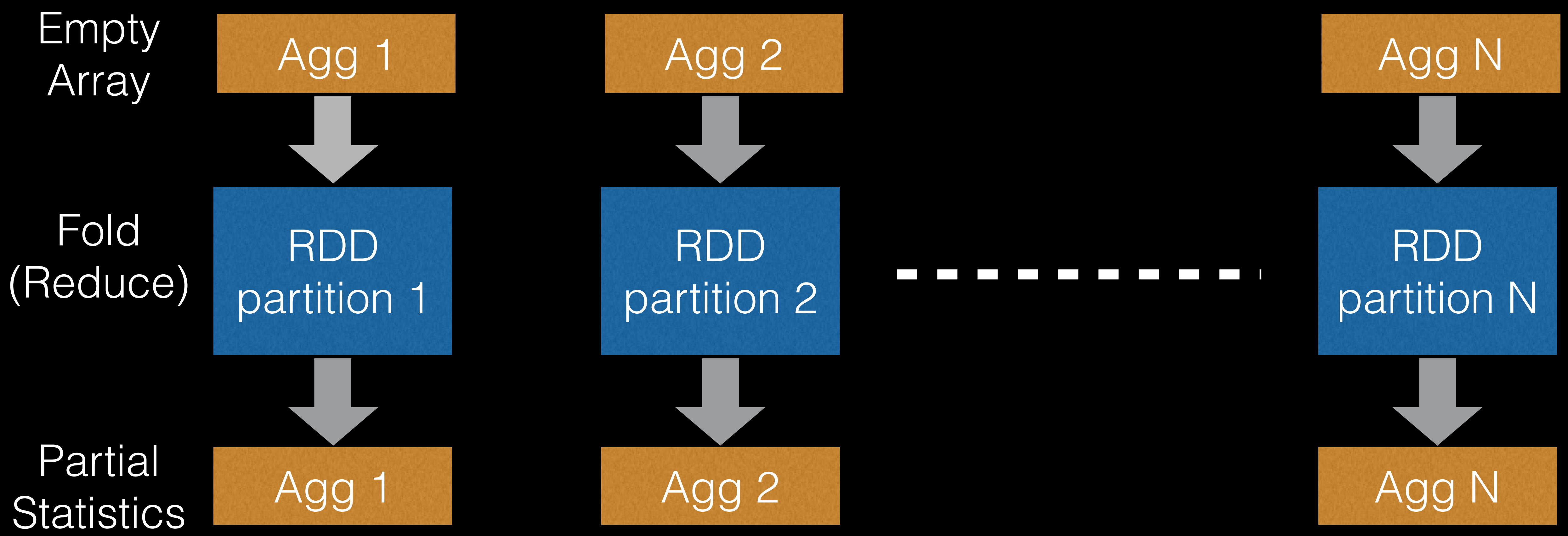
- Map operation essential when keys not known
  - For e.g., words in word count
  - Splits known in advance
- No map
  - avoids object creation overhead
  - avoids communication overhead due to shuffle

Optimization 1:
Aggregate (Distributed Fold)

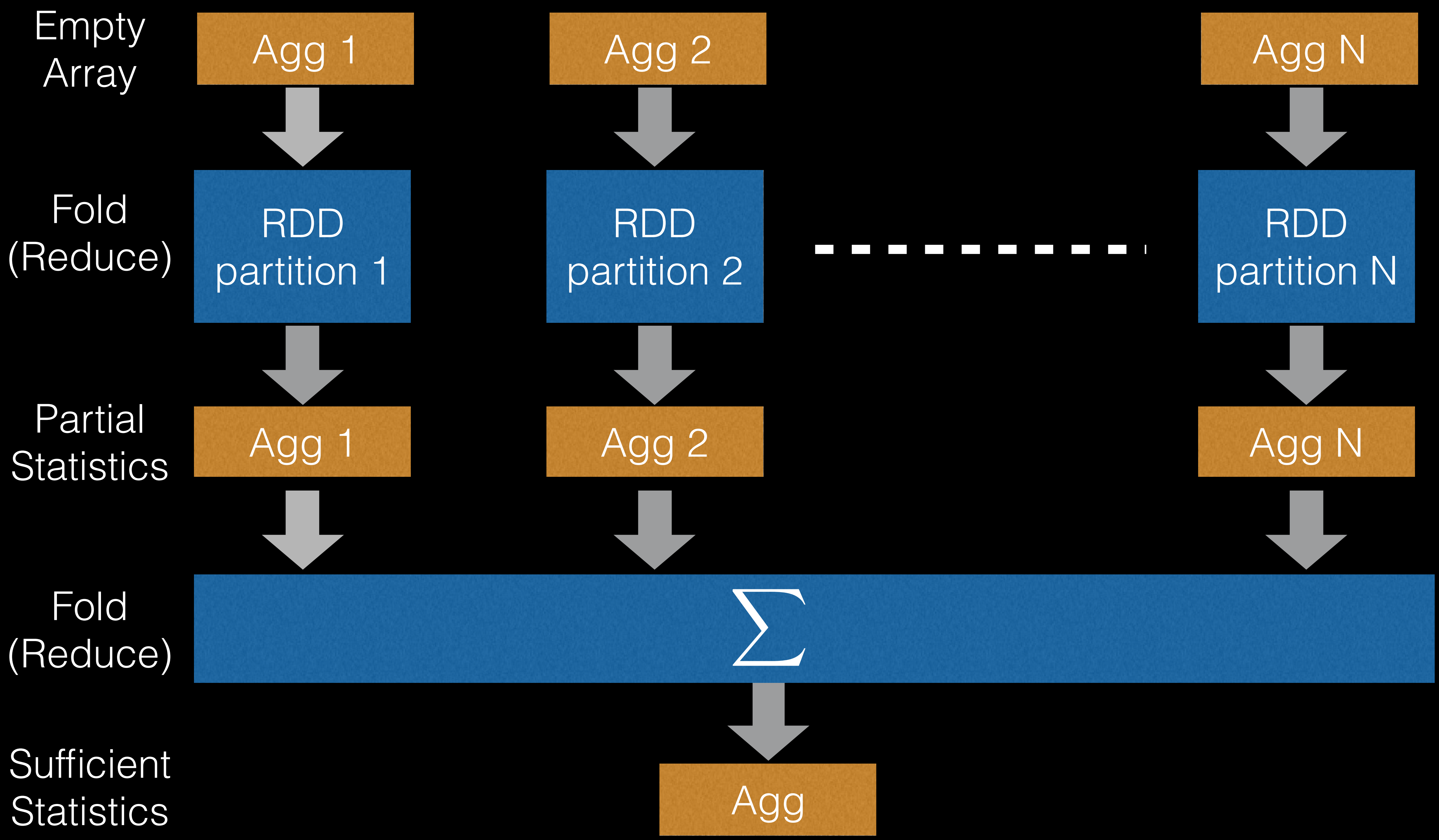
Optimization 1: Aggregate (Distributed Fold)



Optimization 1: Aggregate (Distributed Fold)



Optimization 1: Aggregate (Distributed Fold)

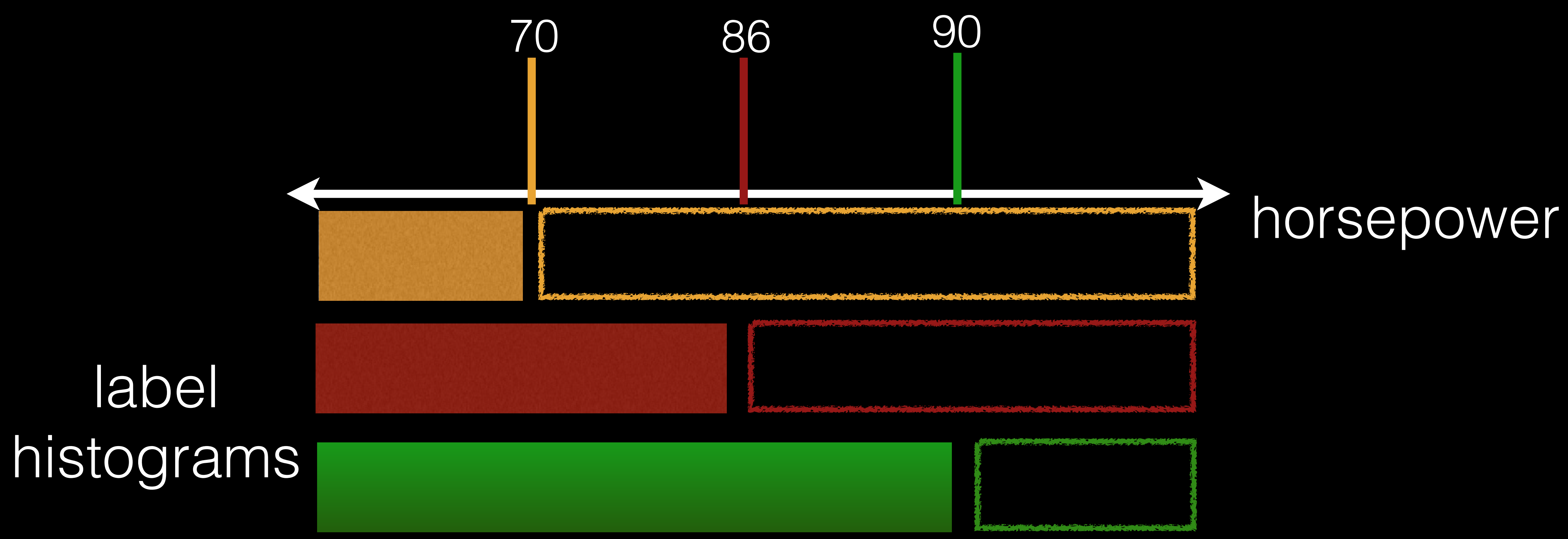


Sufficient Statistics

- Left and right child node statistics for each split
- Classification: label counts
- Regression: count, sum, sum^2

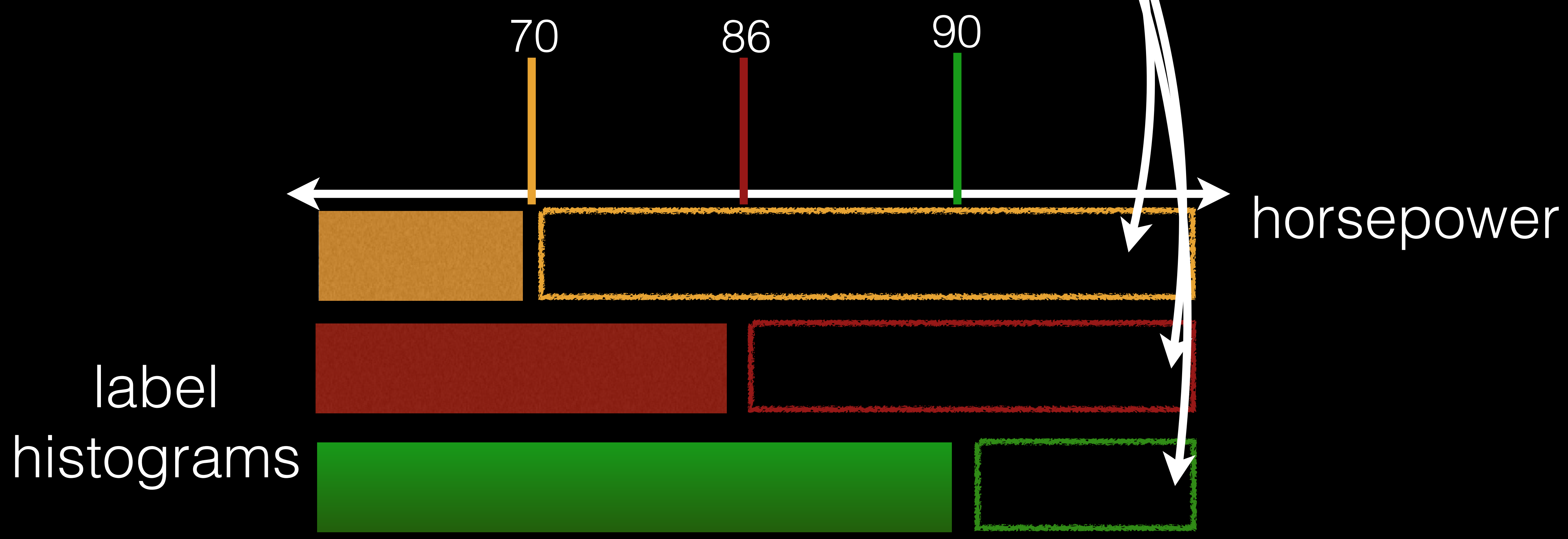
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



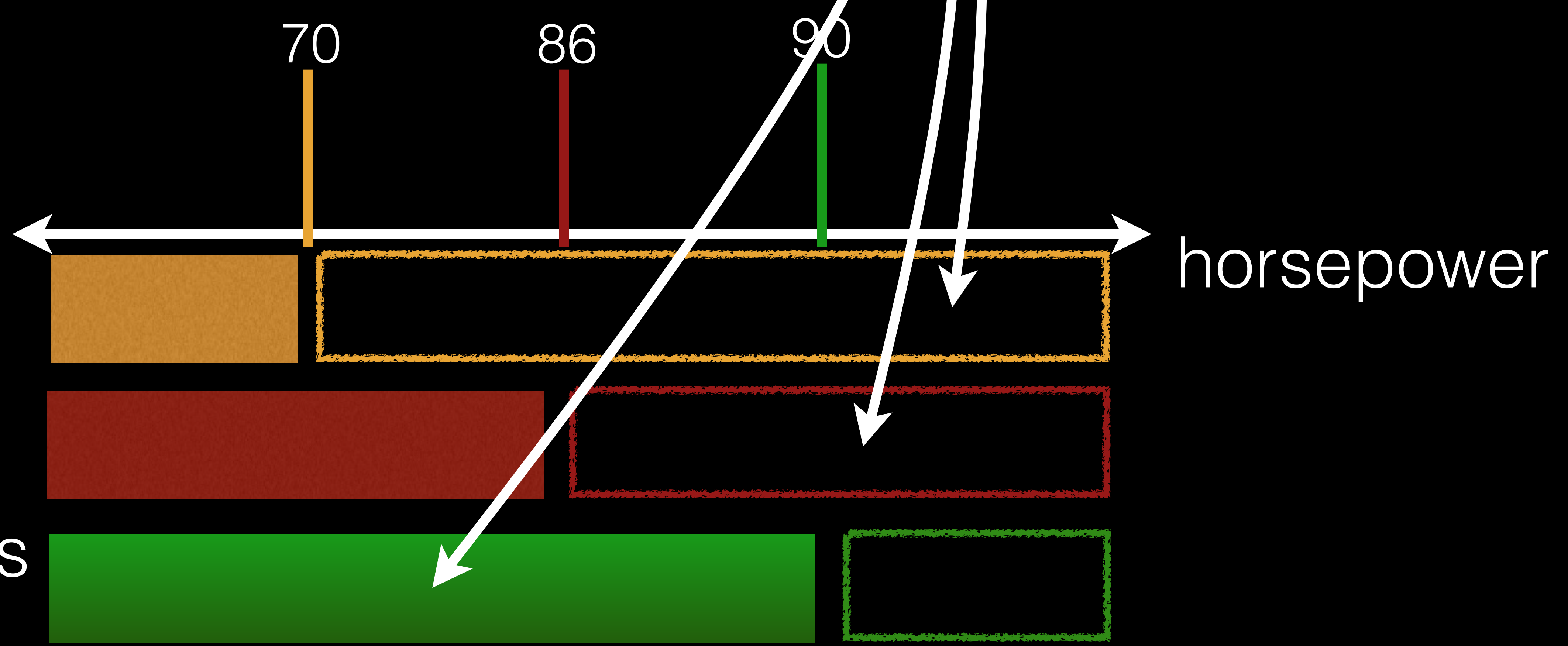
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



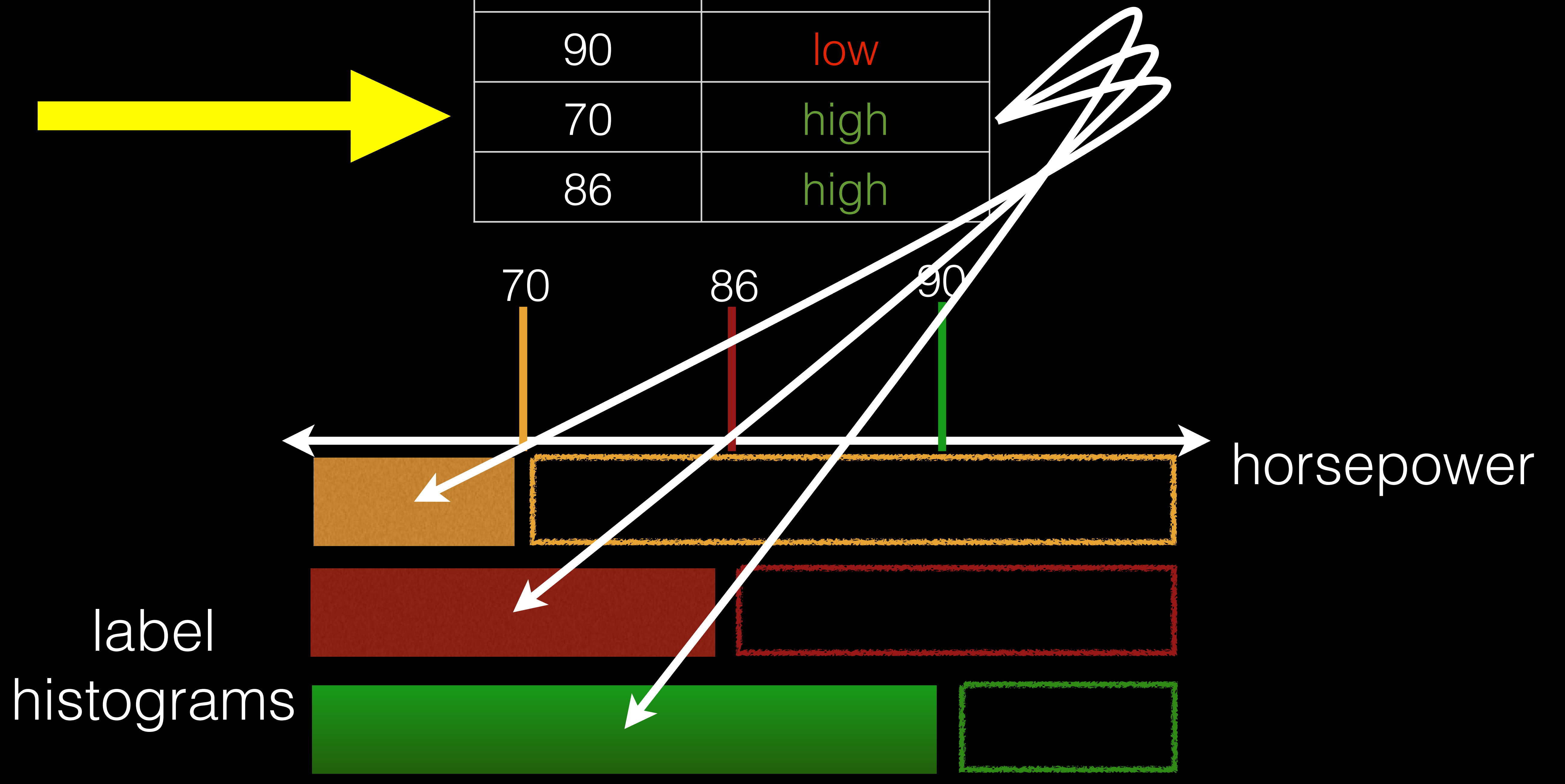
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



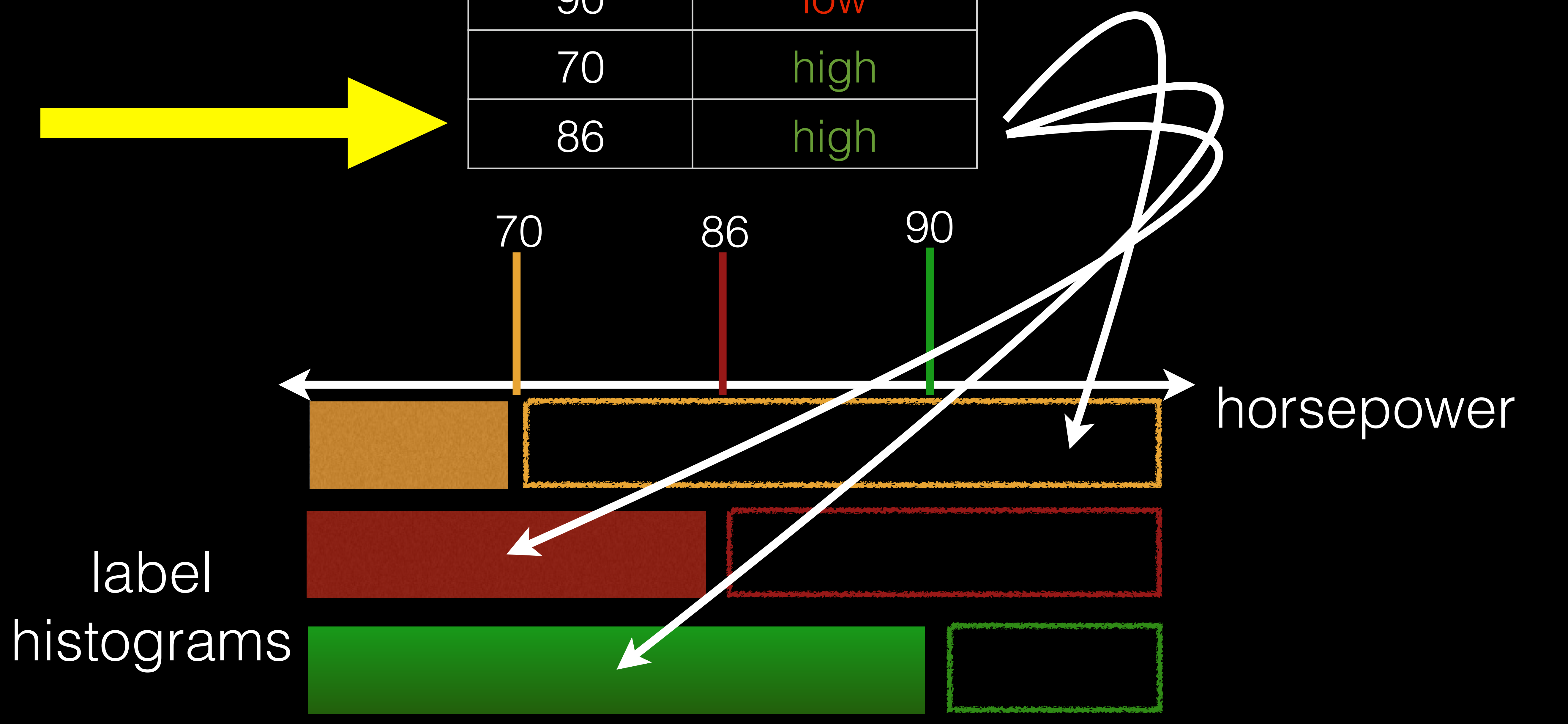
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



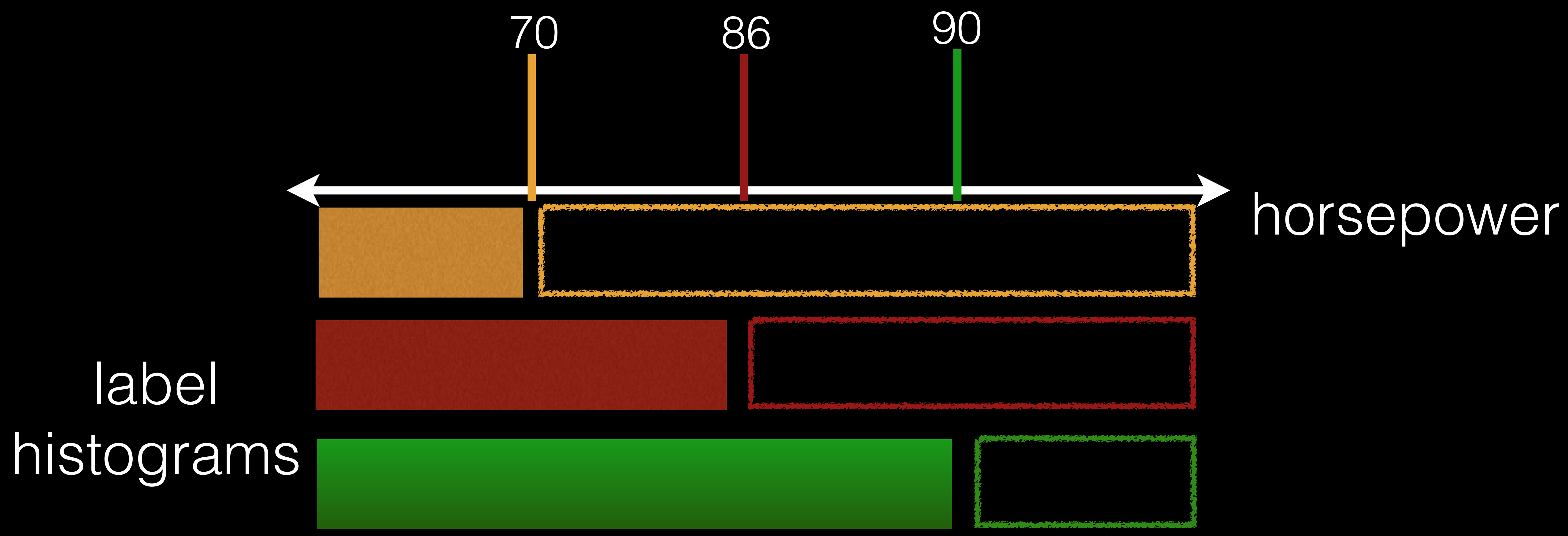
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



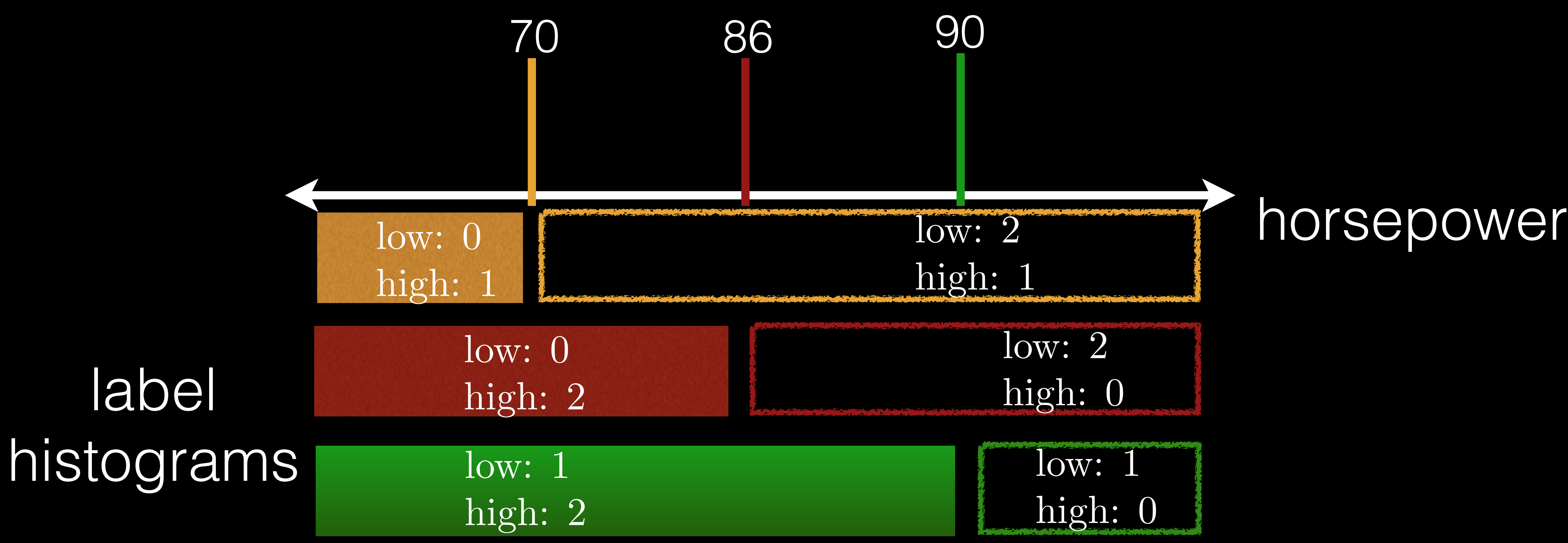
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



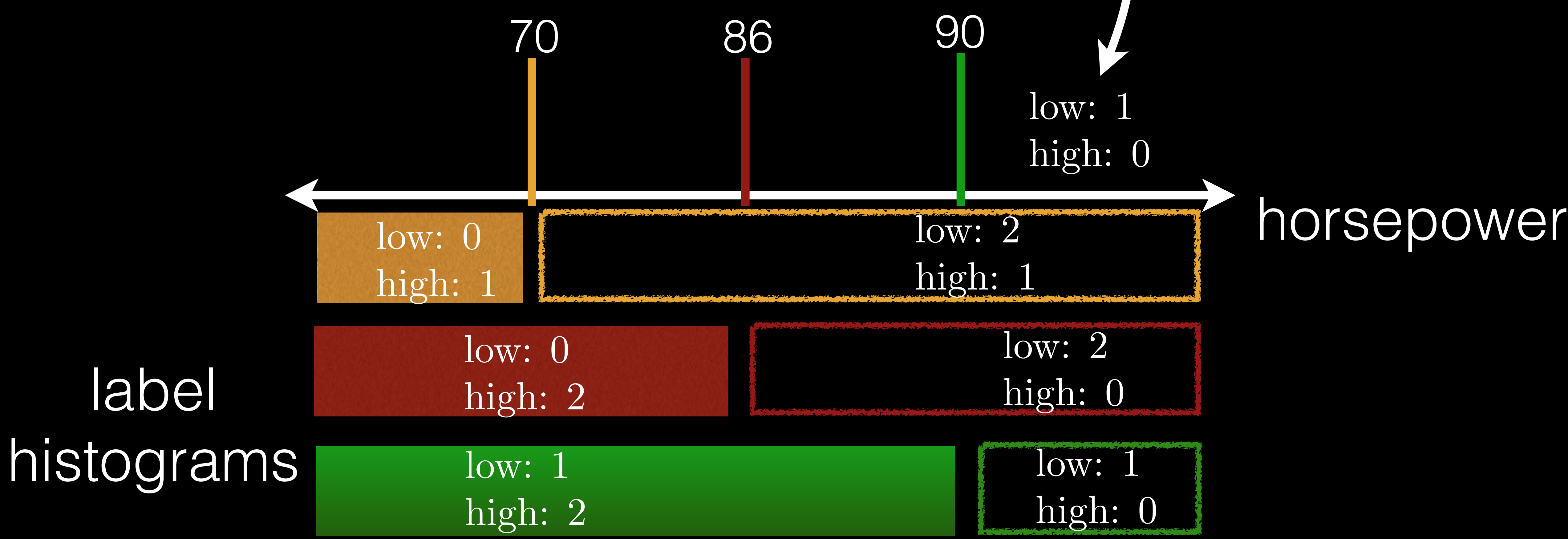
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



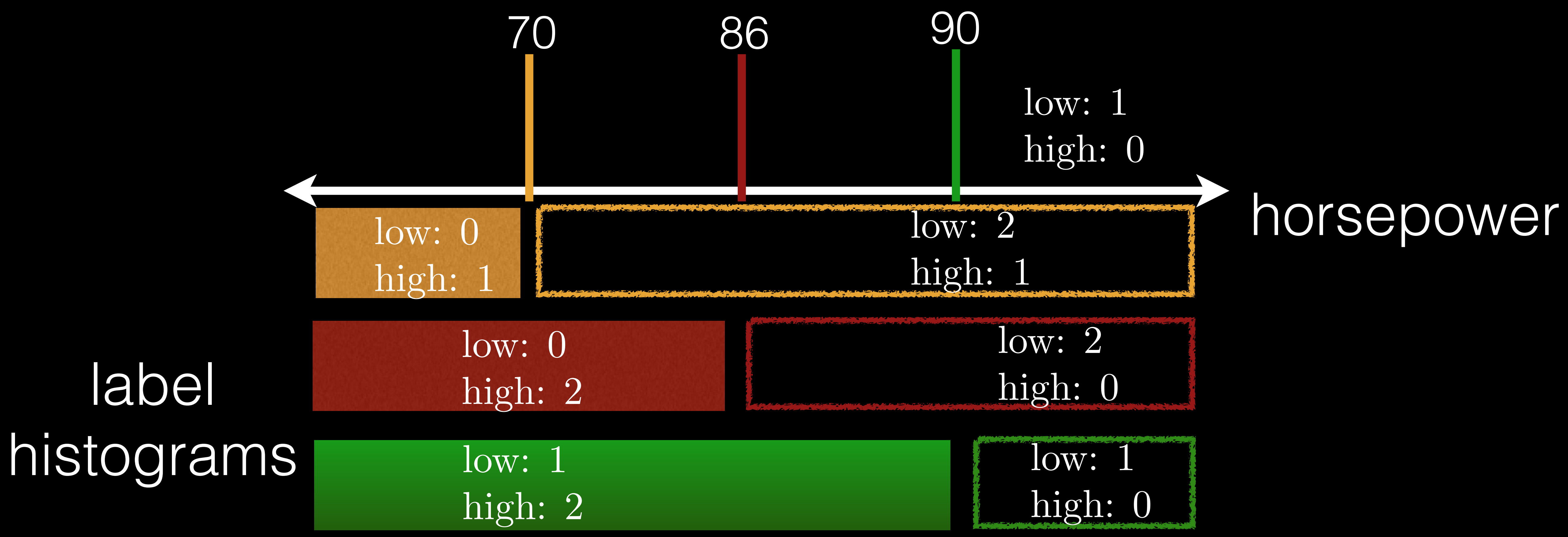
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



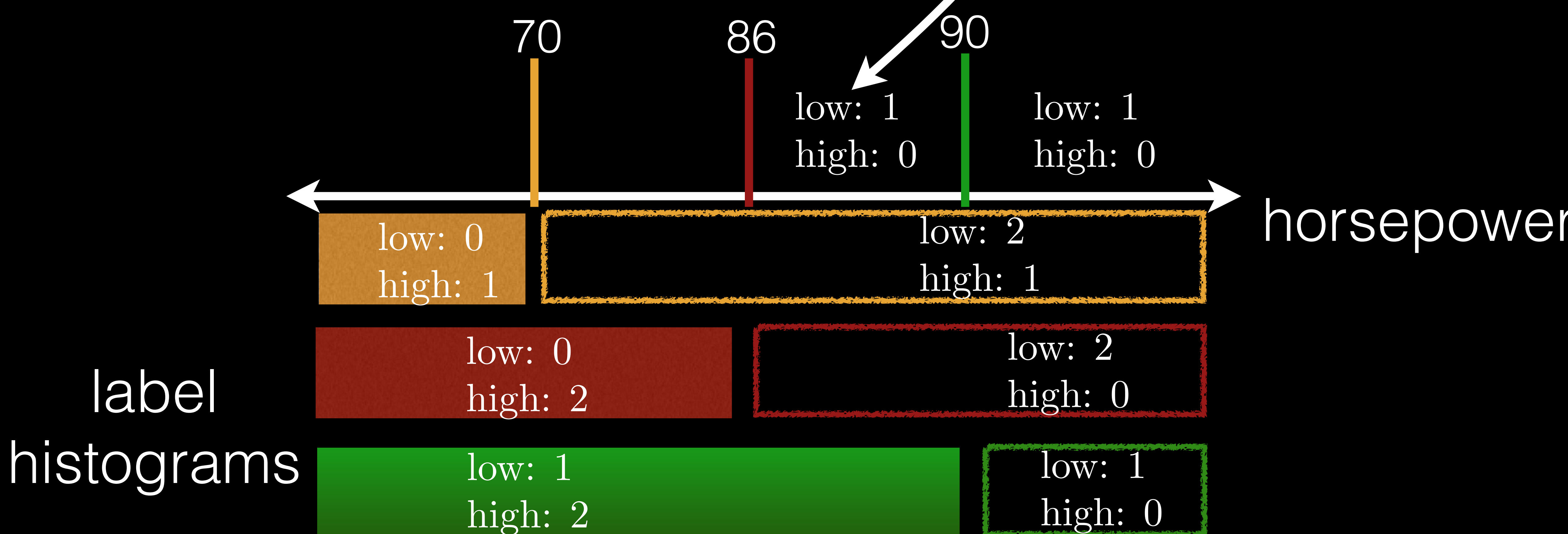
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



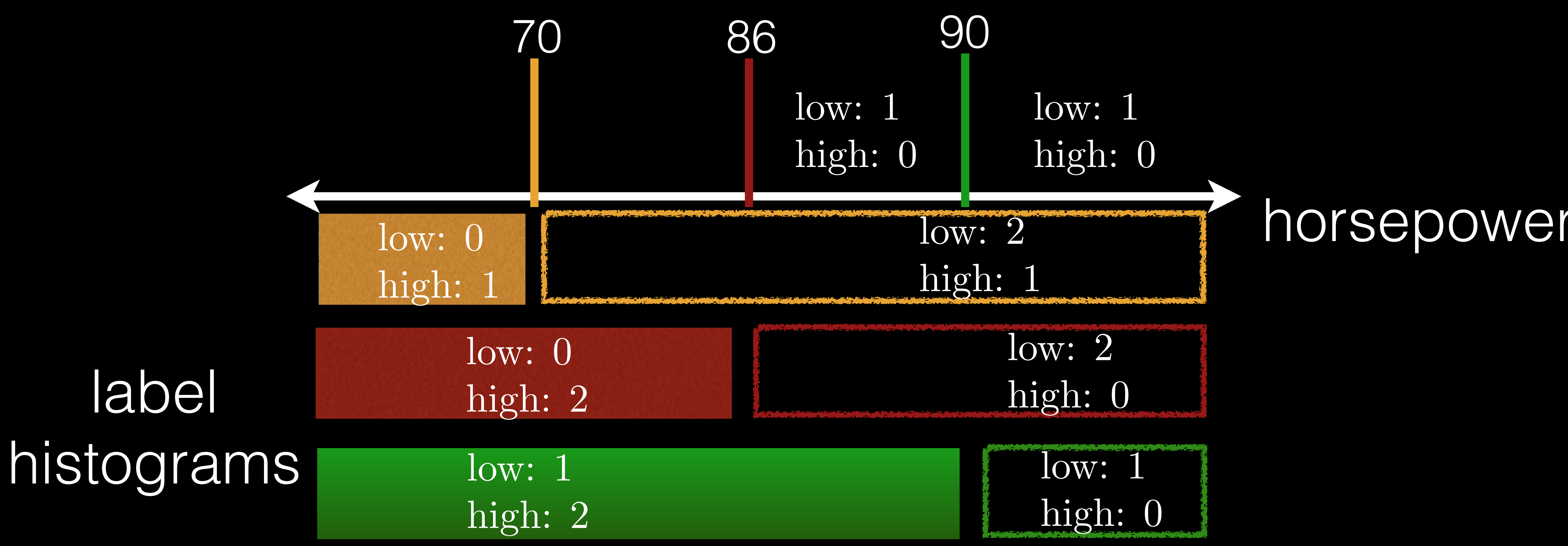
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



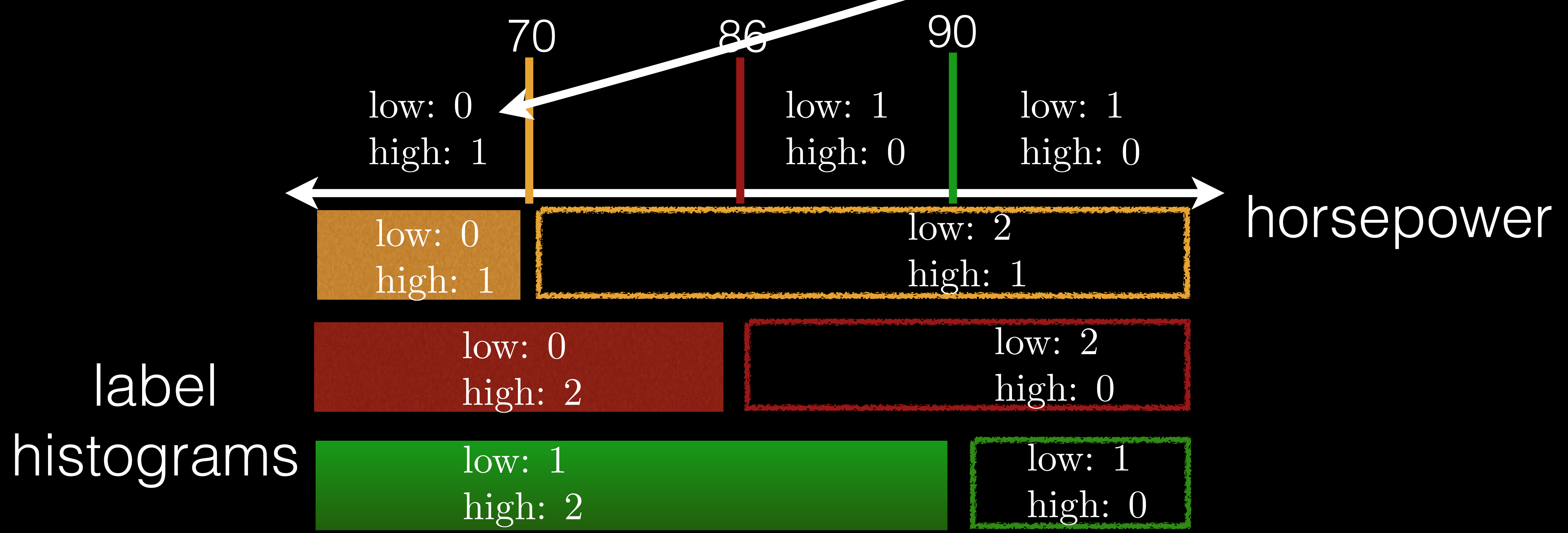
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



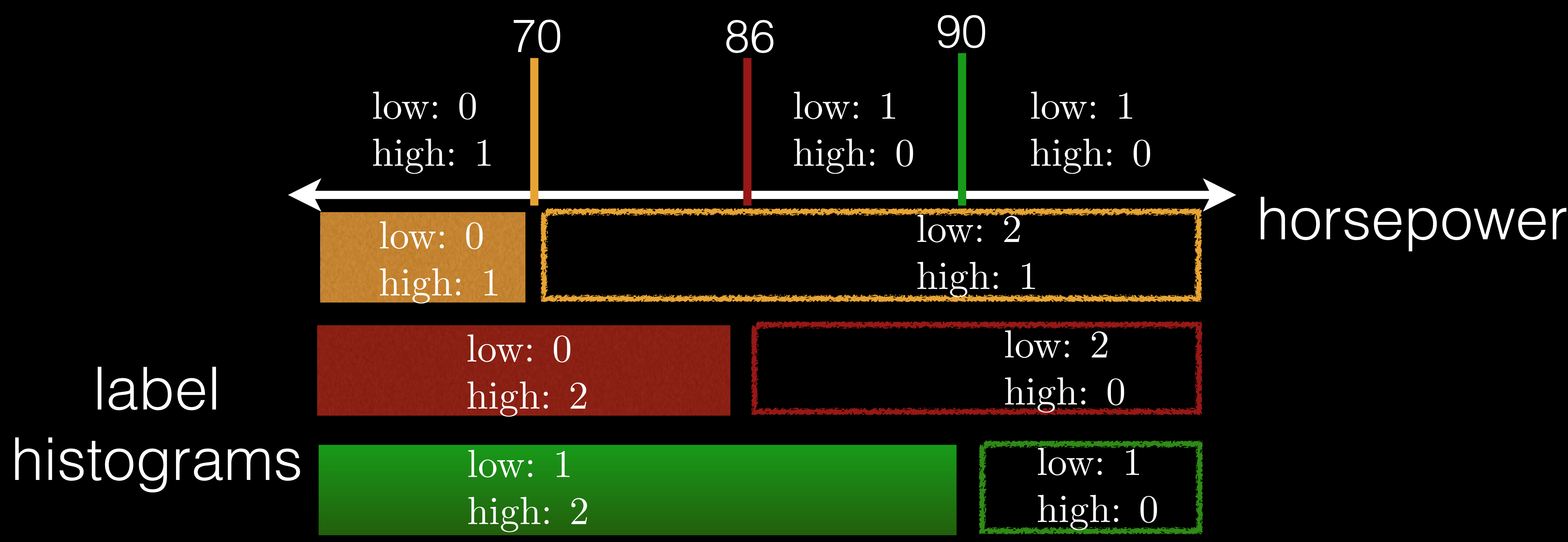
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



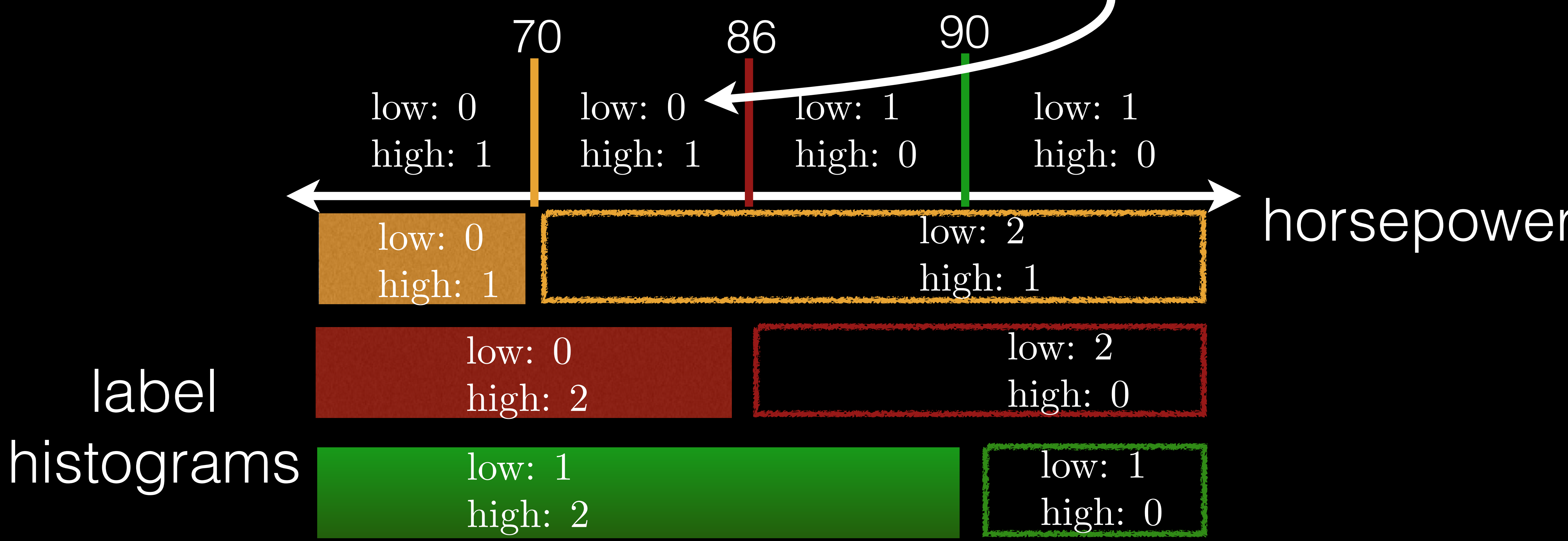
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



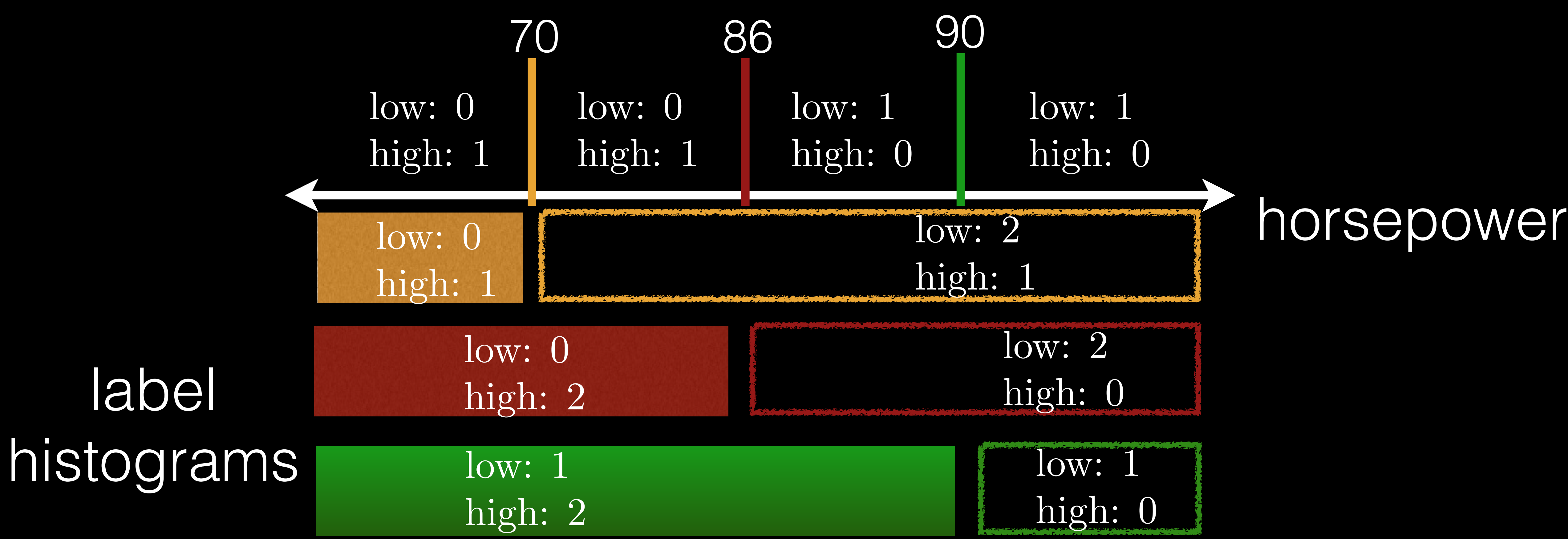
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



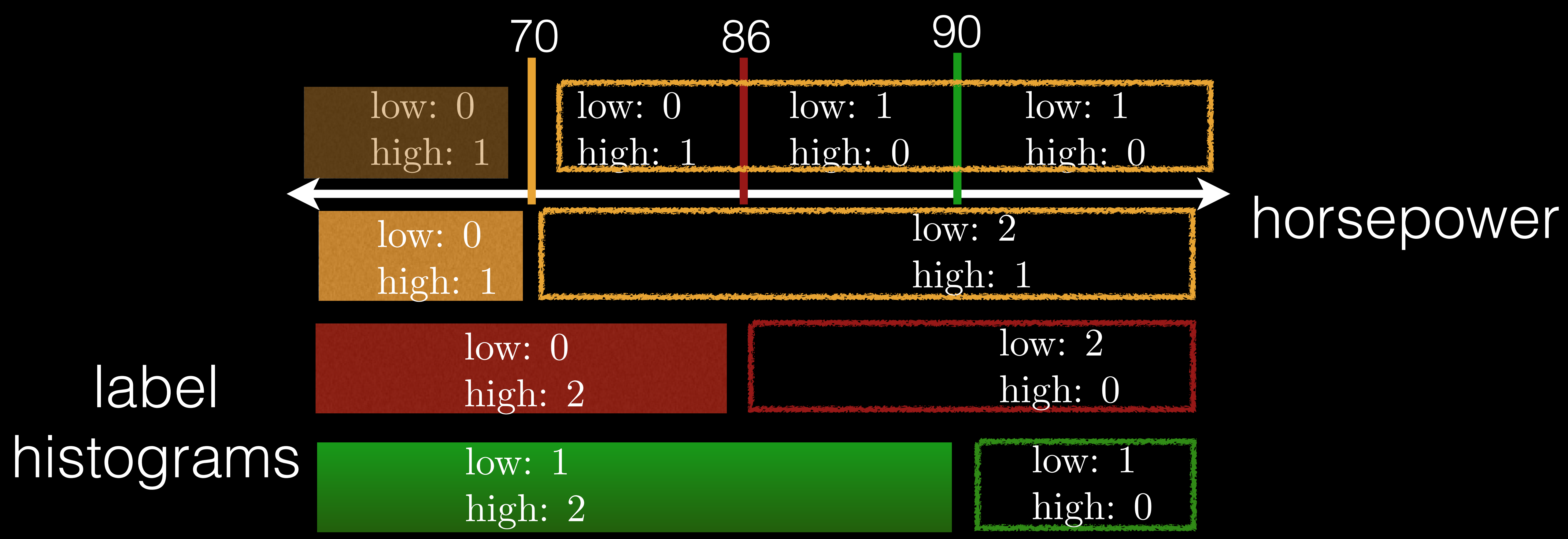
Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high



Optimization 2: Binning

hp	mileage
95	low
90	low
70	high
86	high

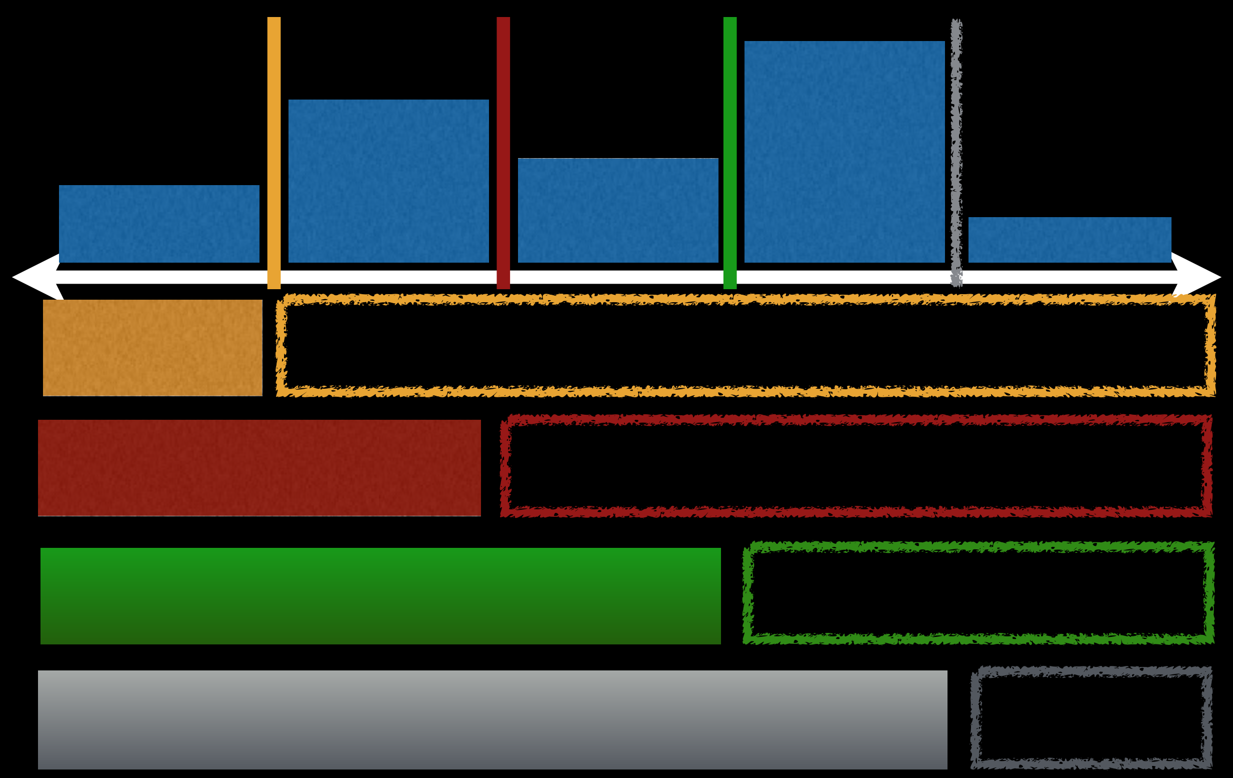


Bin-based Info Gain

m splits per feature

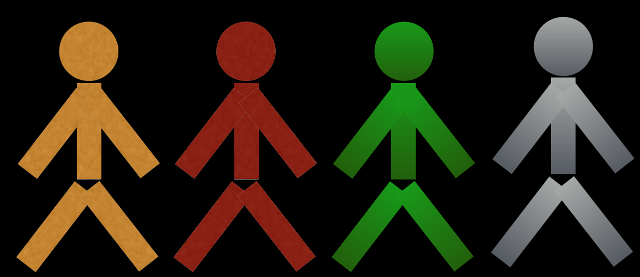
Binning using binary search
log(m) versus m

Bin histogram update
factor of m savings

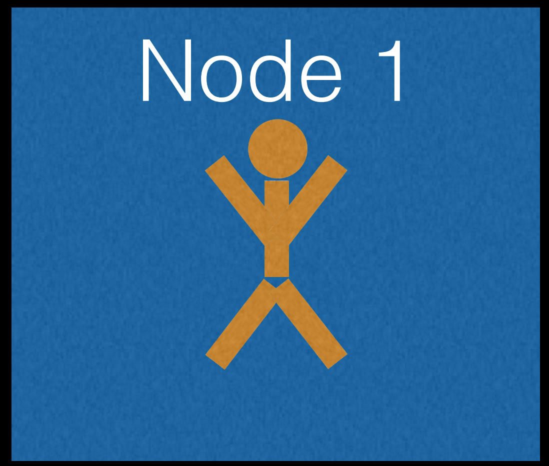


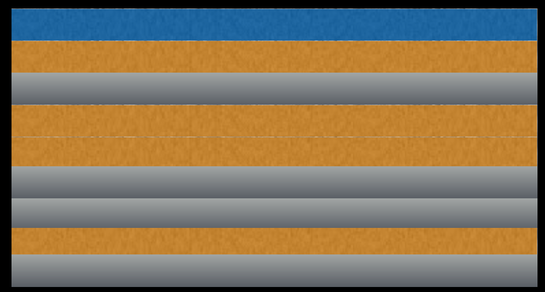
Significant savings in computation

Optimization 3: Level-wise training



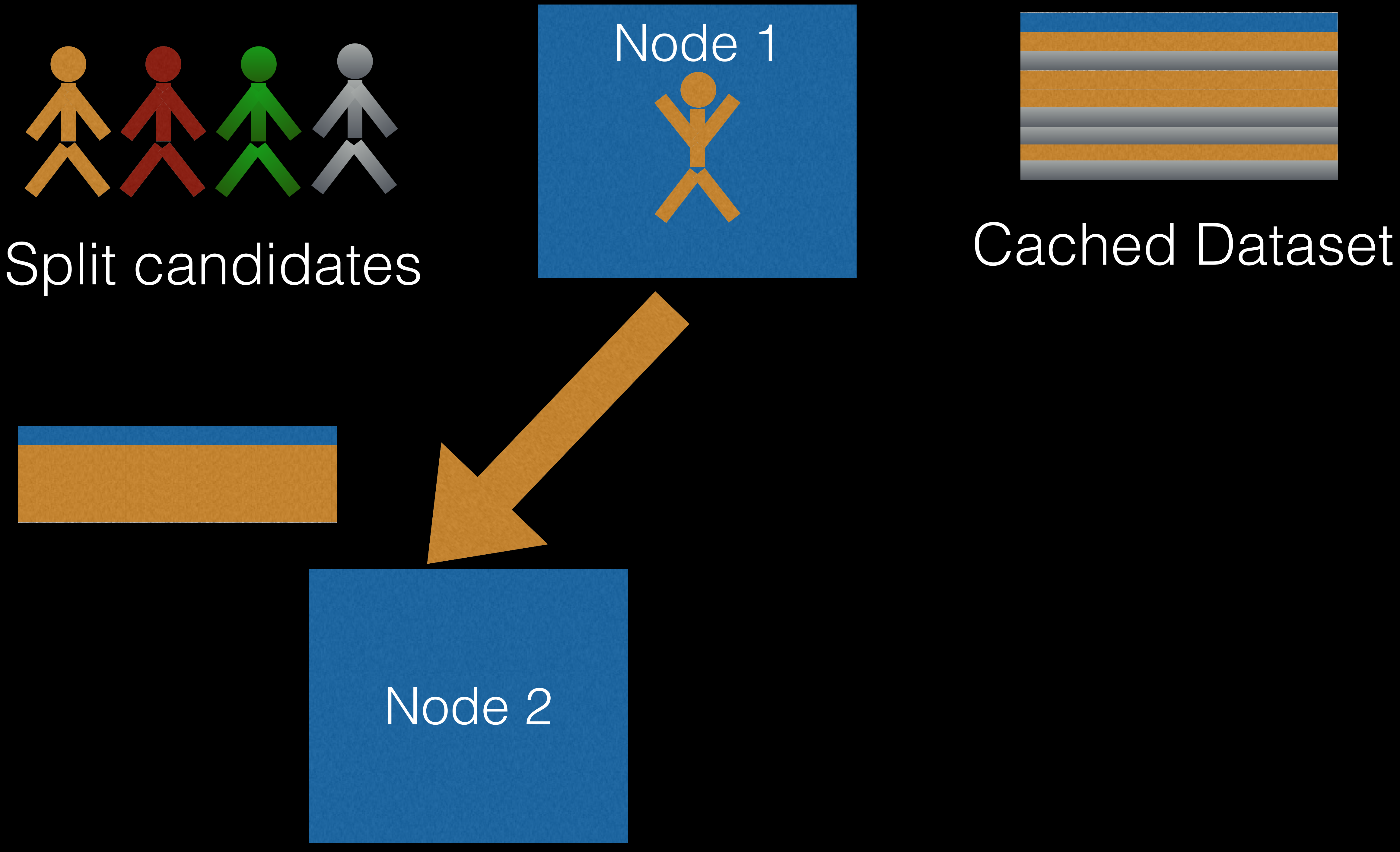
Split candidates



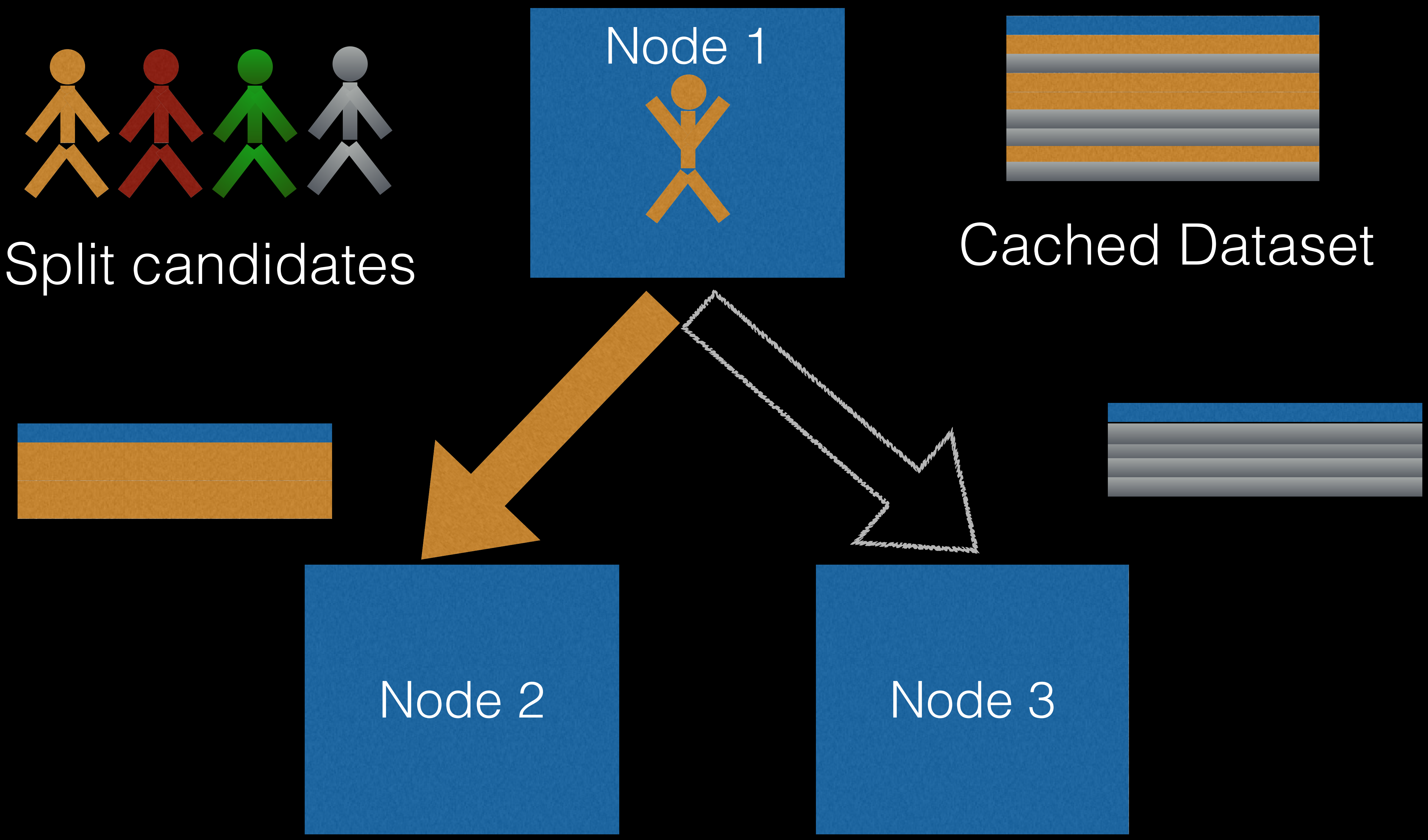


Cached Dataset

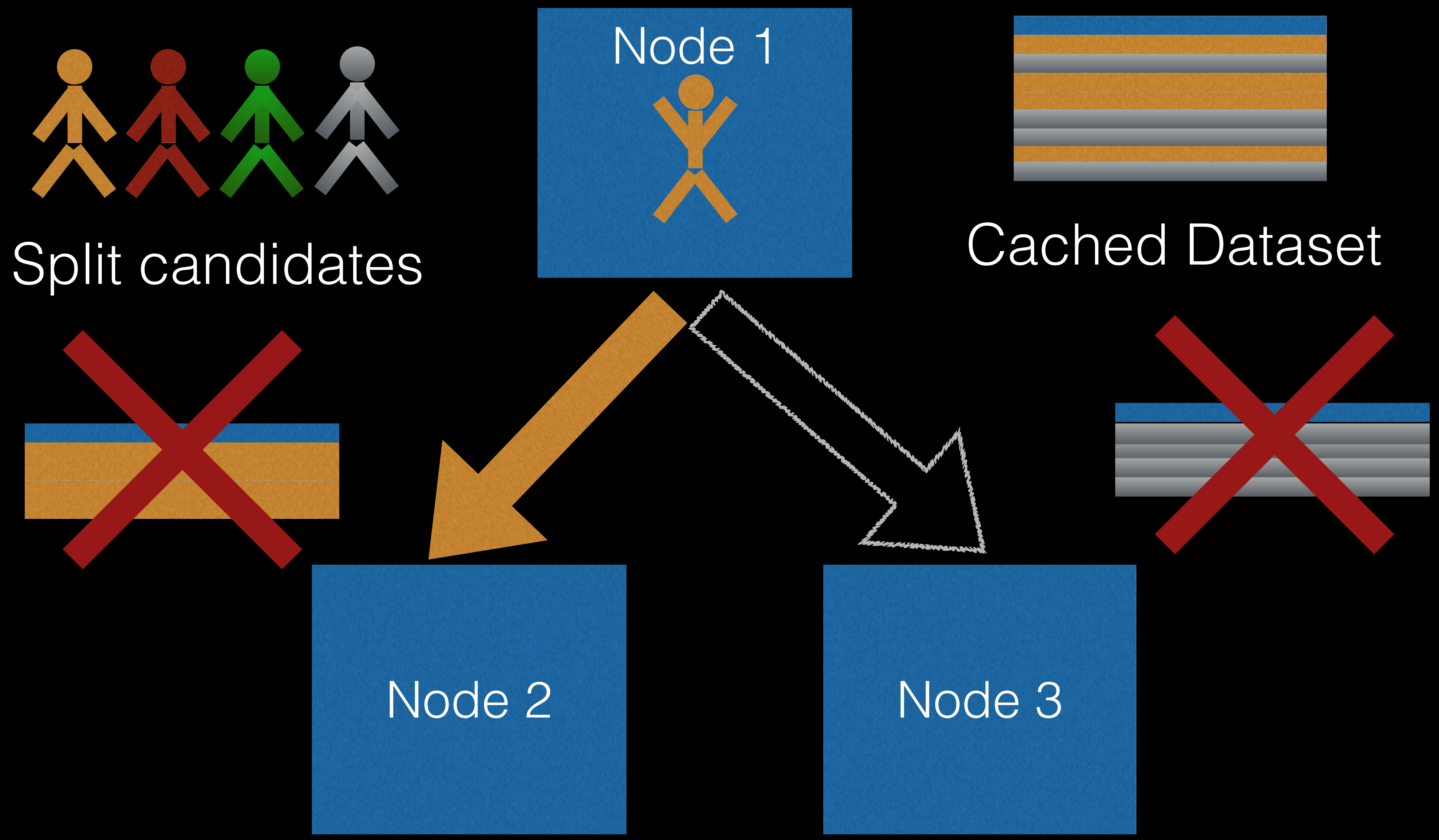
Optimization 3: Level-wise training



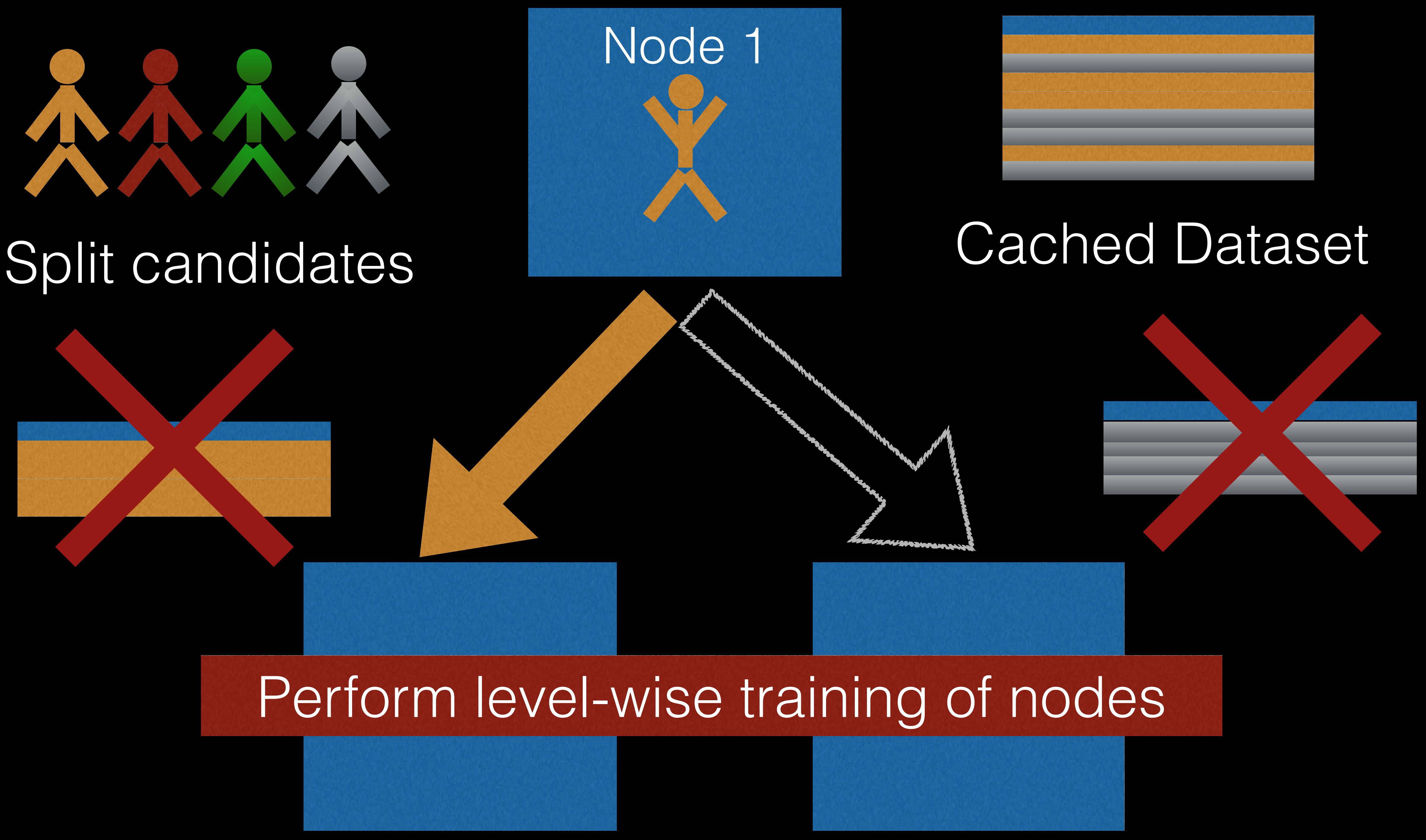
Optimization 3: Level-wise training



Optimization 3: Level-wise training

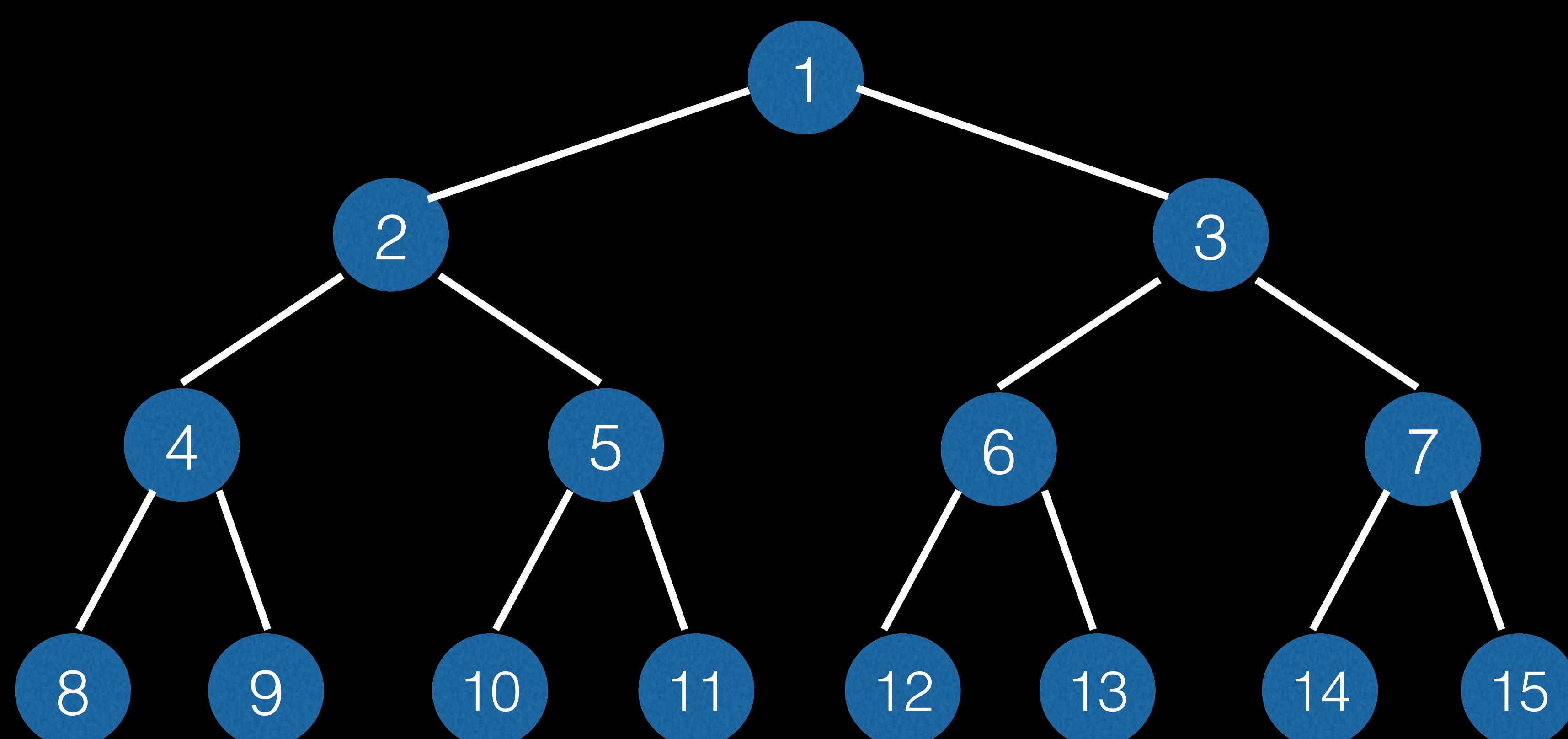


Optimization 3: Level-wise training



Level-wise training

- L passes instead of $2^L - 1$ for full tree
- Depth 4: 4 passes instead of 15
- Depth 10: 10 passes instead of 1023
- ….



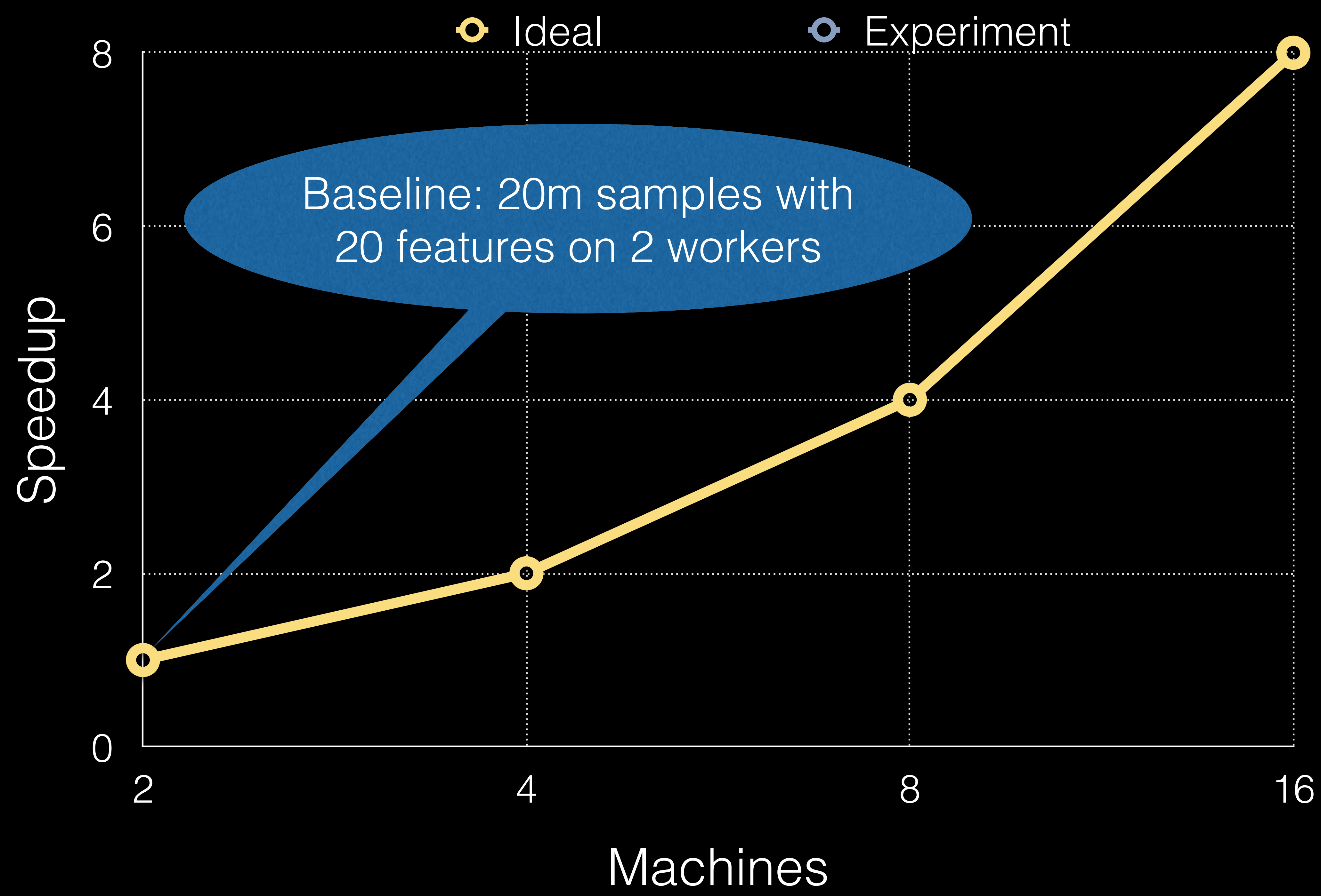
MLlib decision tree features

- Binary classification and regression (1.0)
- Categorical variable support (1.0)
- Arbitrarily deep trees (1.1)
- Multiclass classification* (under review for 1.1)
- Sample weights* (under review for 1.1)

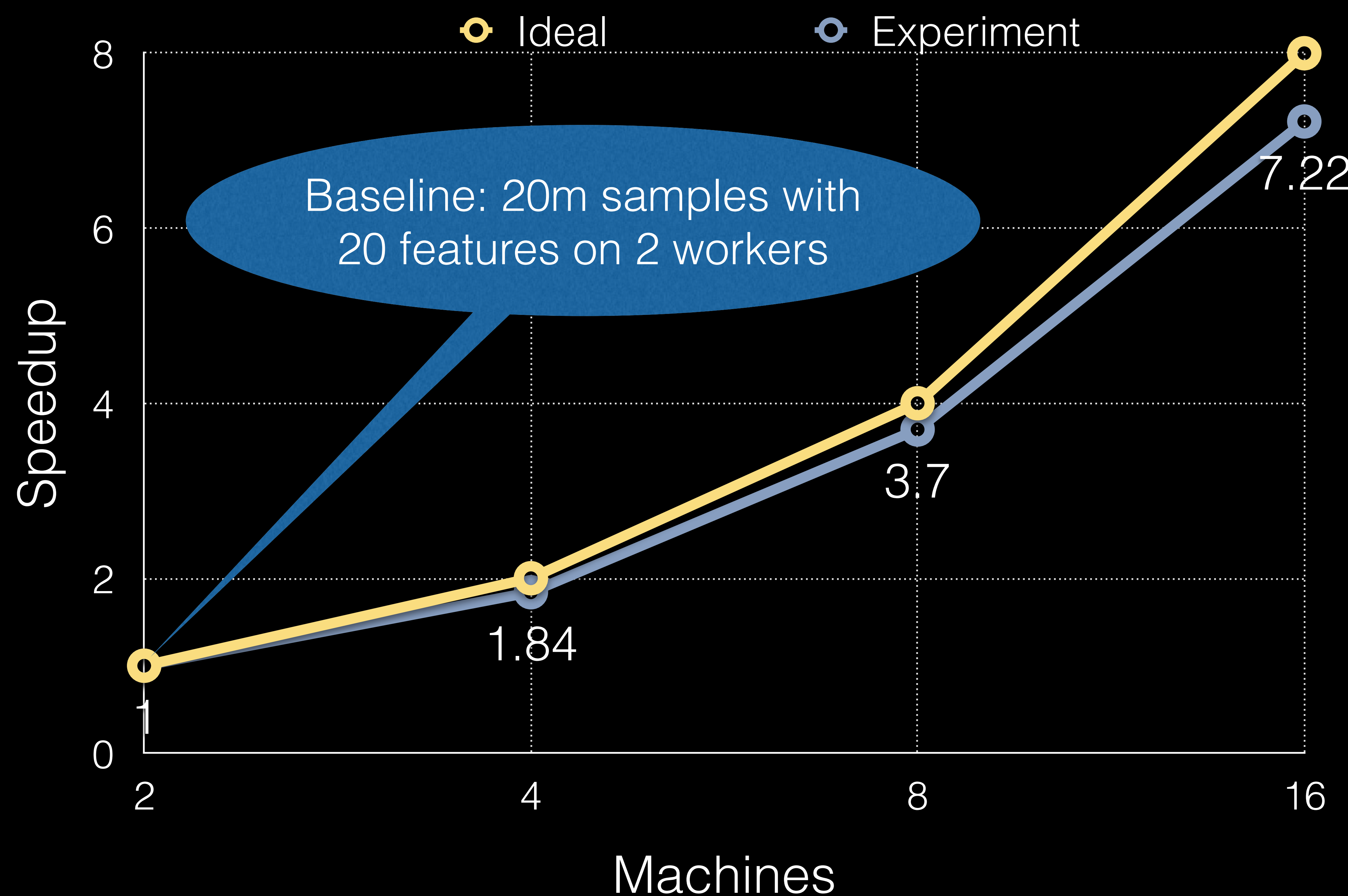
Overview

- Decision Tree 101
- Distributed Decision Trees in MLlib
- Experiments
- Ensembles
- Future work

Strong Scaling Experiment



Strong Scaling Experiment

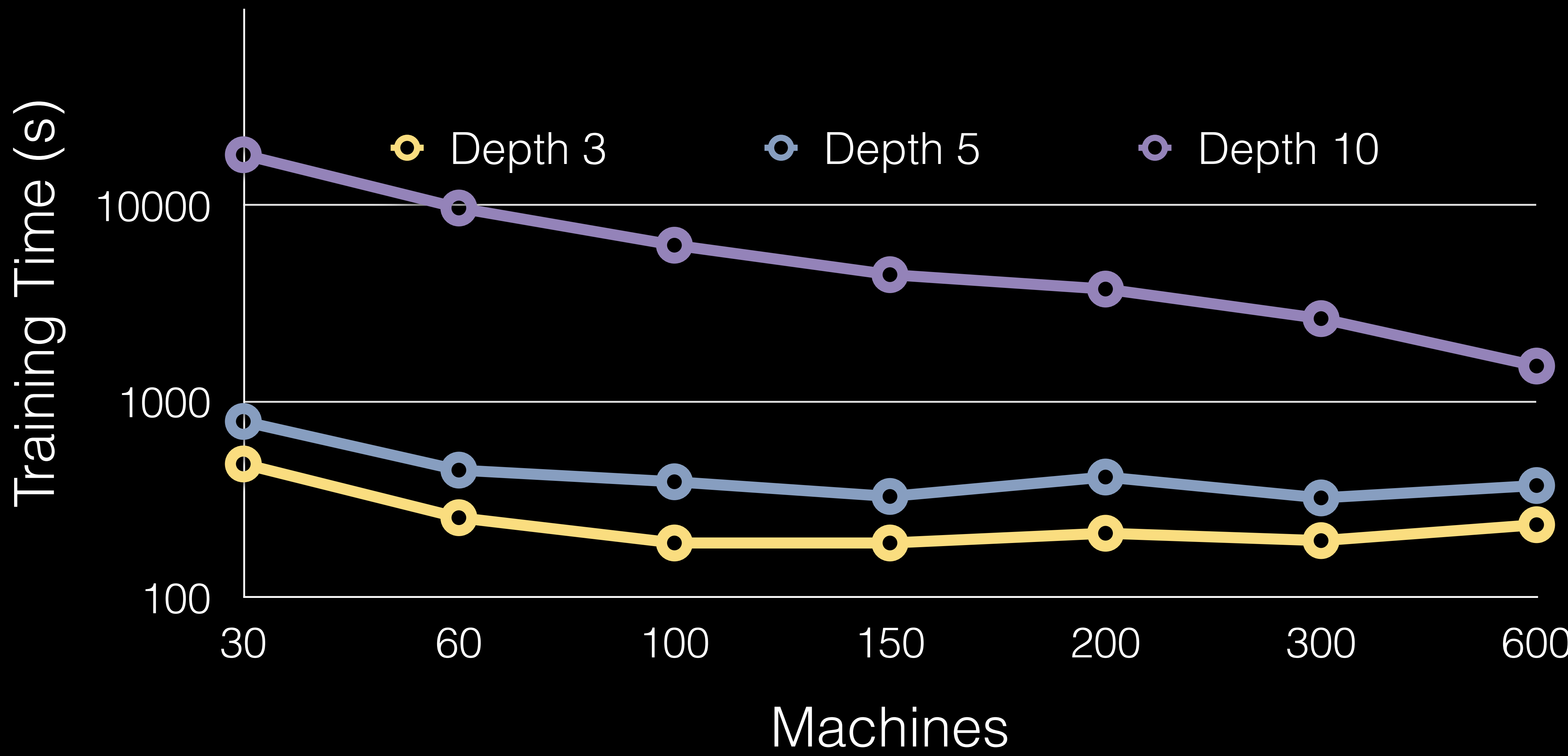


Strong Scaling Results

- Synthetic dataset
- 10 to 50 million instances
- 10 to 50 features
- 2 to 16 machines
- 700 MB to 18 GB dataset
- Average speedup from 2 to 16 machines was 6.6X!

Large-scale experiment

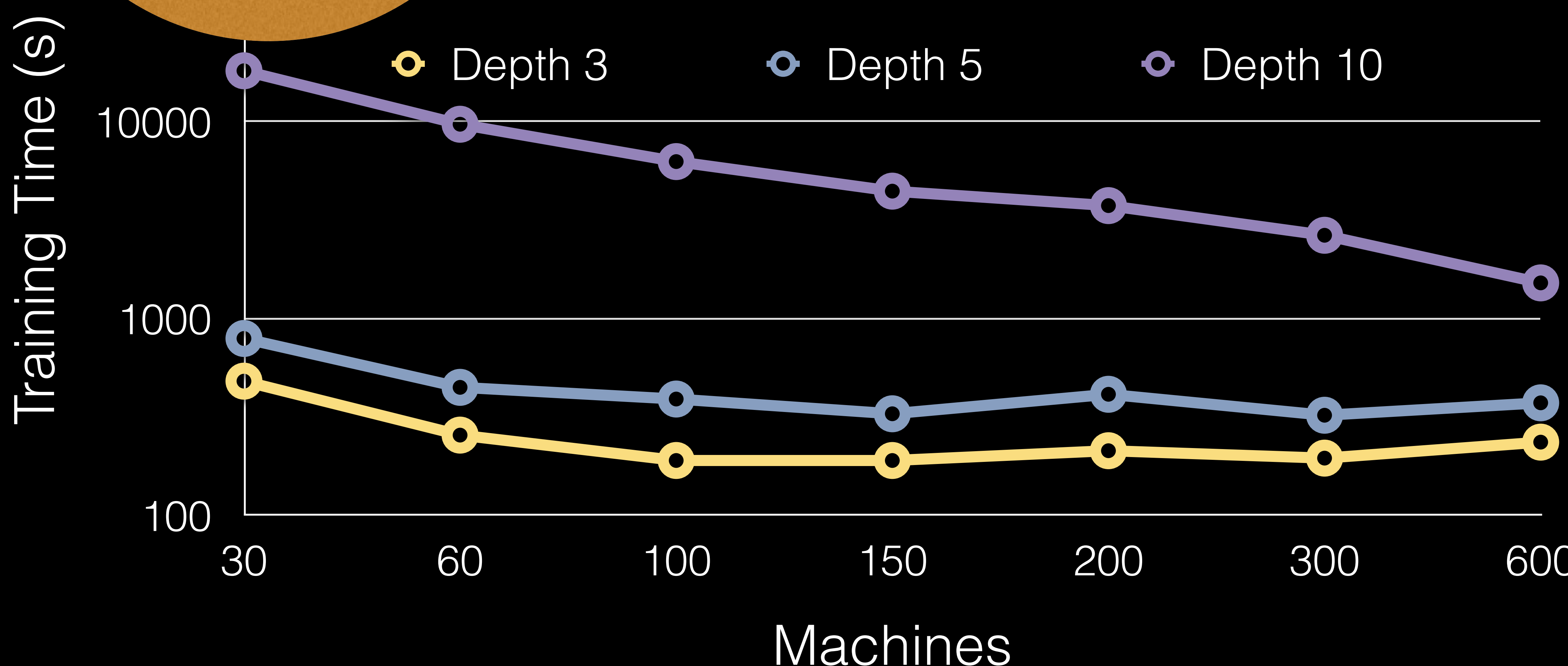
0.5 billion instances, 20 features, 90 GB dataset



Large-scale experiment

Works on large datasets

nces, 20 features, 90 GB dataset

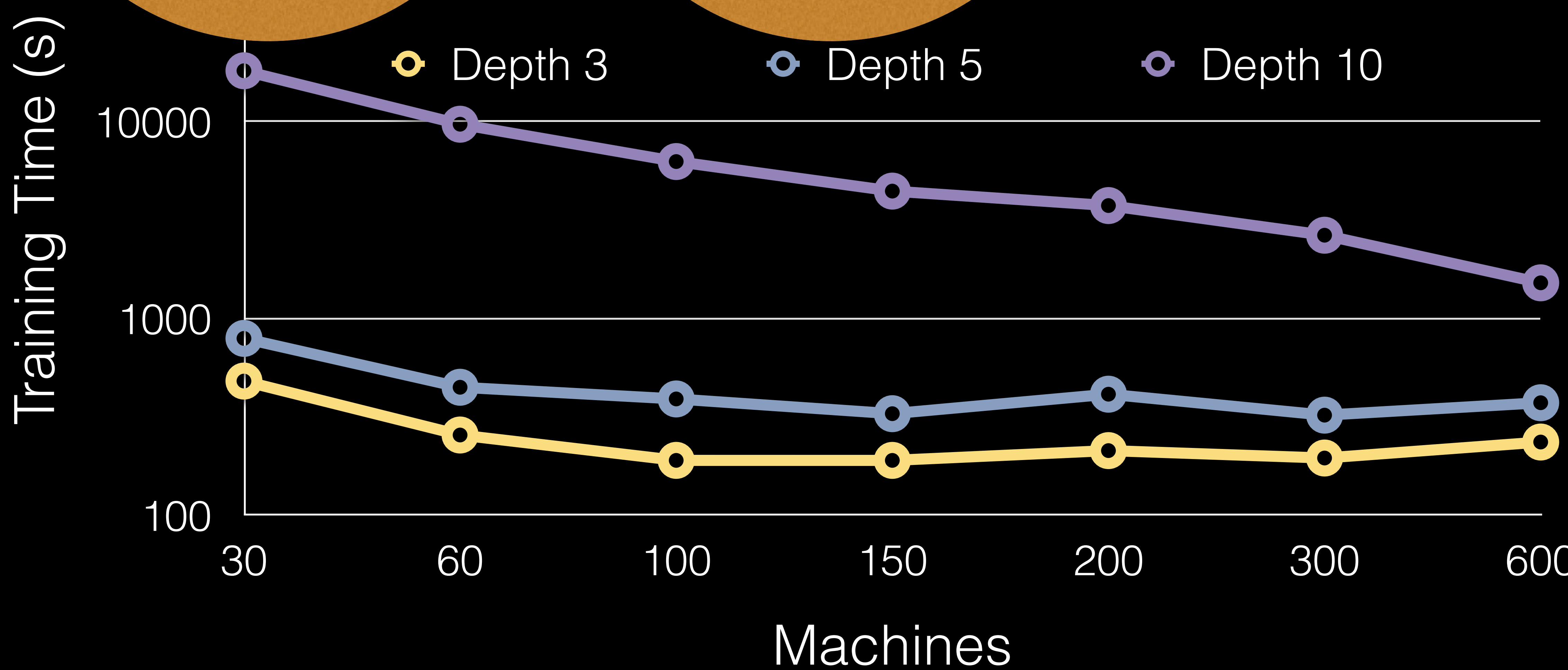


Large-scale experiment

Works on large datasets

Deep trees require more comp.

nc … 90 GB dataset

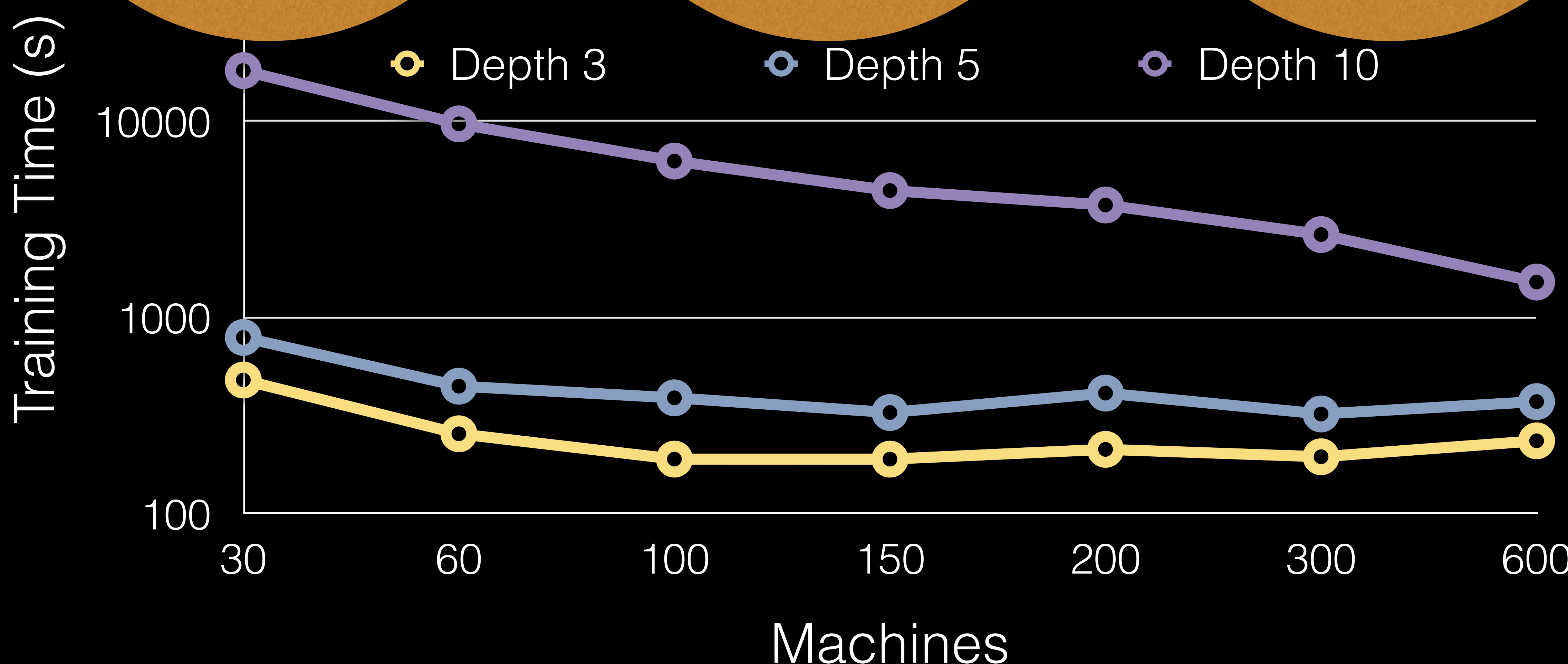


Large-scale experiment

Works on large datasets

Deep trees require more comp.

Comp. vs Comm.

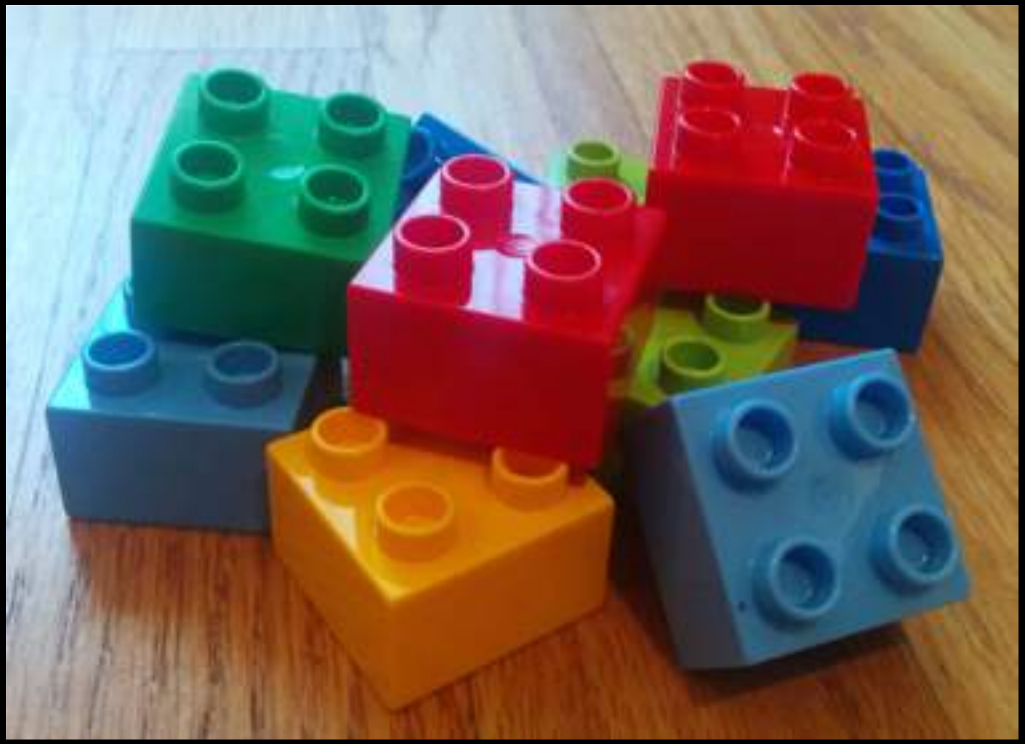


Overview

- Decision Tree 101
- Distributed Decision Trees in MLlib
- Experiments
- Ensembles
- Future work

Tree Ensembles

- Decision trees are building blocks
- Boosting
  - sequential
  - sample weight
- Random Forests
  - parallel construction
  - level-wise training extension to multiple trees



AdaBoost wrapper

```
// SAMME
var weightedInput = input
// 2. For m = 1 to M:
var m = 0
while (m < M) {
  // (a) Fit a classifier T(m)(x) to the training data using weights wi.
  trees(m) = new DecisionTree(strategy).train(weightedInput)
  // (b) Compute err(m)
  val weightedTotalError
    = weightedInput.map(x => x.weight * unequalIdentity(trees(m).predict(x.features),
    x.label)).sum()
  val totalWeight = weightedInput.map(x => x.weight).sum()
  val err = weightedTotalError / totalWeight
  // (c) Compute alpha(m)
  alphas(m) = math.log((1- err)/err) + math.log(K - 1)
  // (d) Set weights
  weightedInput
    = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight * alphas(m) * unequalIdentity(trees(m).predict(x.features), x.label)))
  // (e) Renormalize weights
  val totalWeightAfterReweighing = weightedInput.map(x => x.weight).sum()
  weightedInput = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight / totalWeightAfterReweighing ))
  m += 1
}
```

AdaBoost wrapper

```scala
// SAMME
var weightedInput = input
// 2. For m = 1 to M:
var m = 0
while (m < M) {
  // (a) Fit a classifier T(m)(x) to the training data using weights wi.
  trees(m) = new DecisionTree(strategy).train(weightedInput)
  // (b) Compute err(m)
  val weightedTotalError
    = weightedInput.map(x => x.weight * unequalIdentity(trees(m).predict(x.features),
    x.label)).sum()
  val totalWeight = weightedInput.map(x => x.weight).sum()
  val err = weightedTotalError / totalWeight
  // (c) Compute alpha(m)
  alphas(m) = math.log((1- err)/err) + math.log(K - 1)
  // (d) Set weights
  weightedInput
    = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight * alphas(m) * unequalIdentity(trees(m).predict(x.features), x.label)))
  // (e) Renormalize weights
  val totalWeightAfterReweighing = weightedInput.map(x => x.weight).sum()
  weightedInput = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight / totalWeightAfterReweighing ))
  m += 1
}
```

AdaBoost wrapper

```scala
// SAMME
var weightedInput = input
// 2. For m = 1 to M:
var m = 0
while (m < M) {
  // (a) Fit a classifier T(m)(x) to the training data using weights wi.
  trees(m) = new DecisionTree(strategy).train(weightedInput)
  // (b) Compute err(m)
  val weightedTotalError
    = weightedInput.map(x => x.weight * unequalIdentity(trees(m).predict(x.features),
    x.label)).sum()
  val totalWeight = weightedInput.map(x => x.weight).sum()
  val err = weightedTotalError / totalWeight
  // (c) Compute alpha(m)
  alphas(m) = math.log((1- err)/err) + math.log(K - 1)
  // (d) Set weights
  weightedInput
    = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight * alphas(m) * unequalIdentity(trees(m).predict(x.features), x.label)))
  // (e) Renormalize weights
  val totalWeightAfterReweighing = weightedInput.map(x => x.weight).sum()
  weightedInput = weightedInput.map(x => WeightedLabeledPoint(x.label, x.features,
    x.weight / totalWeightAfterReweighing ))
  m += 1
}
```

Overview

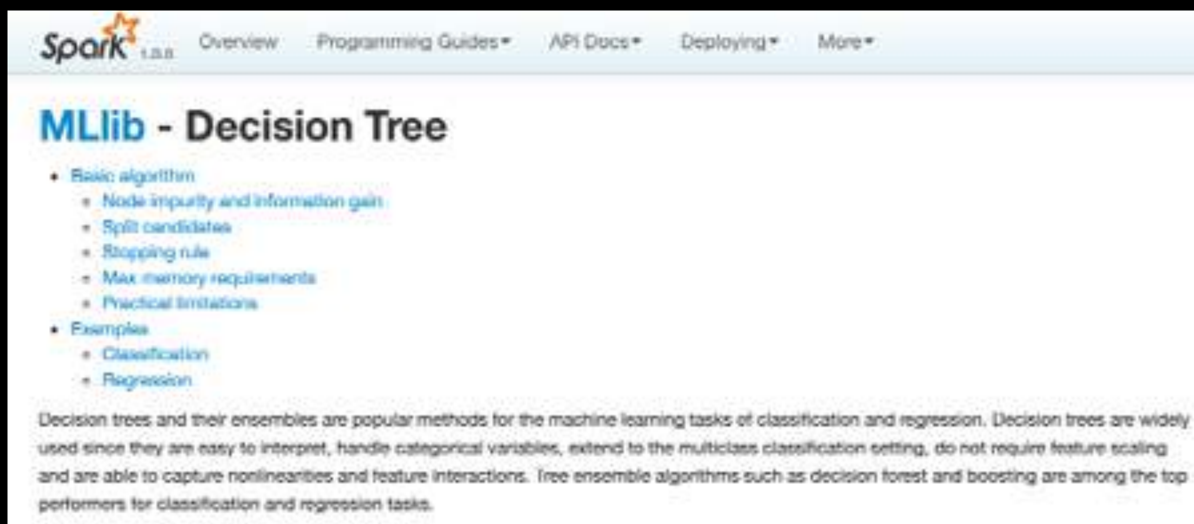
- Decision Tree 101
- Distributed Decision Trees in MLlib
- Experiments
- Ensembles
- Future work

Future Work

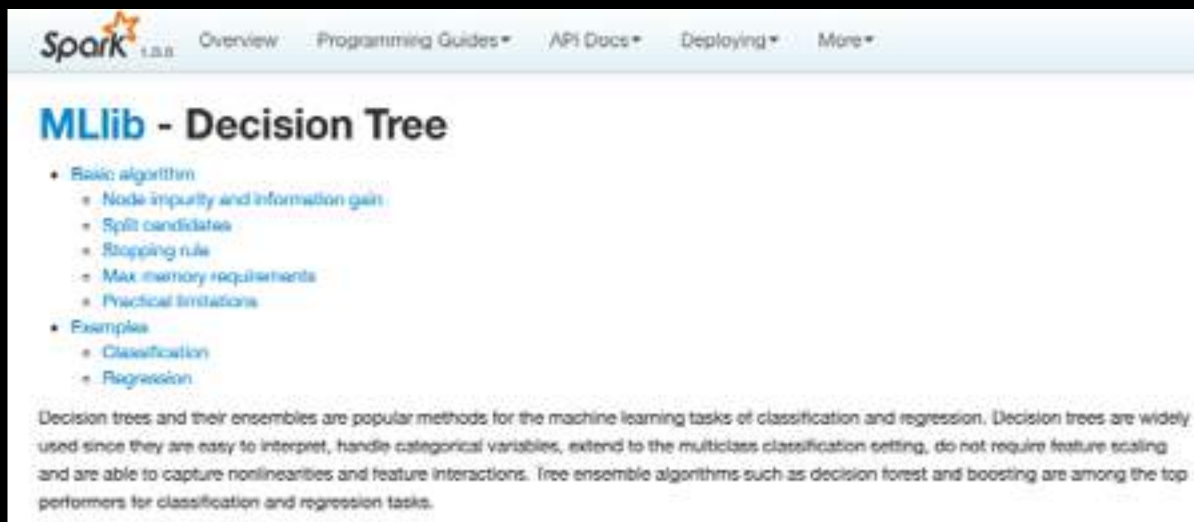
- Ensembles (stretch goal for 1.1)
- Feature importances
- Decision tree visualizations
- Testing over a variety of user datasets

Requests

Requests


Spark
Overview
Programming Guides
API Docs
Deploying
More
MLlib - Decision Tree
Basic algorithm
Node impurity and information gain
Split candidates
Stopping rule
Max memory requirements
Practical limitations
Examples
Classification
Regression
Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical variables, extend to the multiclass classification setting, do not require feature scaling and are able to capture nonlinearities and feature interactions. Tree ensemble algorithms such as decision forest and boosting are among the top performers for classification and regression tasks.

Requests

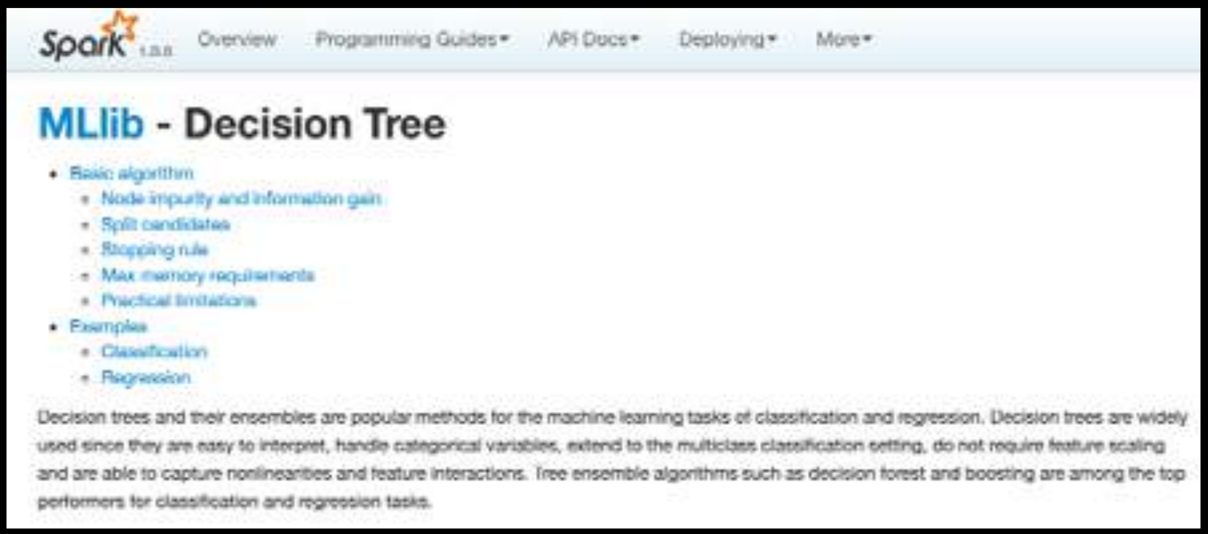

Spark Overview Programming Guides API Docs Deploying More

MLlib - Decision Tree

- Basic algorithm
  - Node impurity and information gain
  - Split candidates
  - Stopping rule
  - Max memory requirements
  - Practical limitations
- Examples
  - Classification
  - Regression

Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical variables, extend to the multiclass classification setting, do not require feature scaling and are able to capture nonlinearities and feature interactions. Tree ensemble algorithms such as decision forest and boosting are among the top performers for classification and regression tasks.

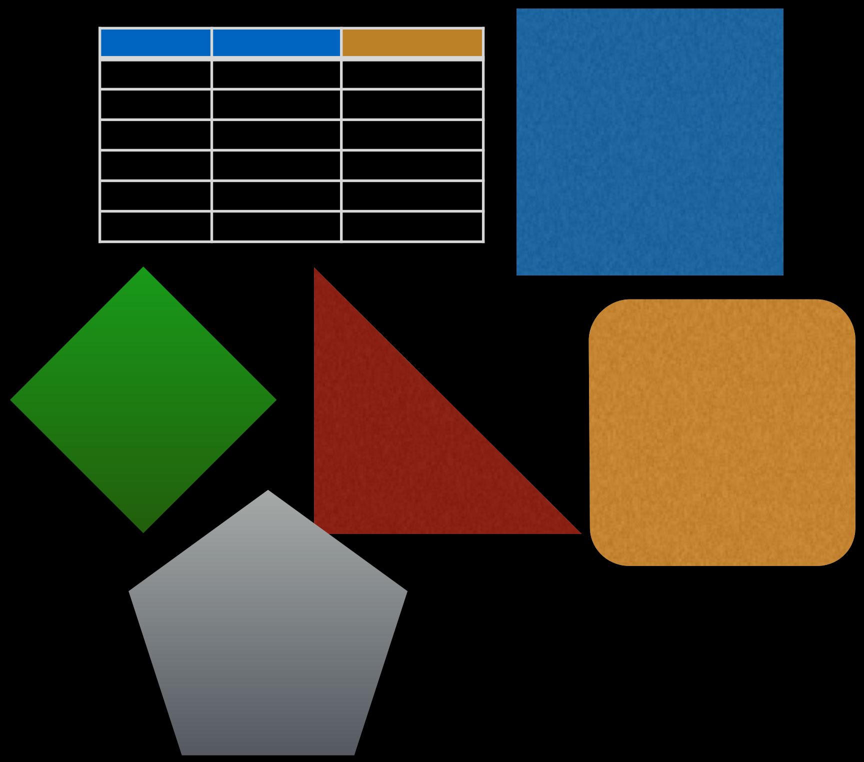
Requests


Spark 1.0.0 Overview Programming Guides API Docs Deploying More

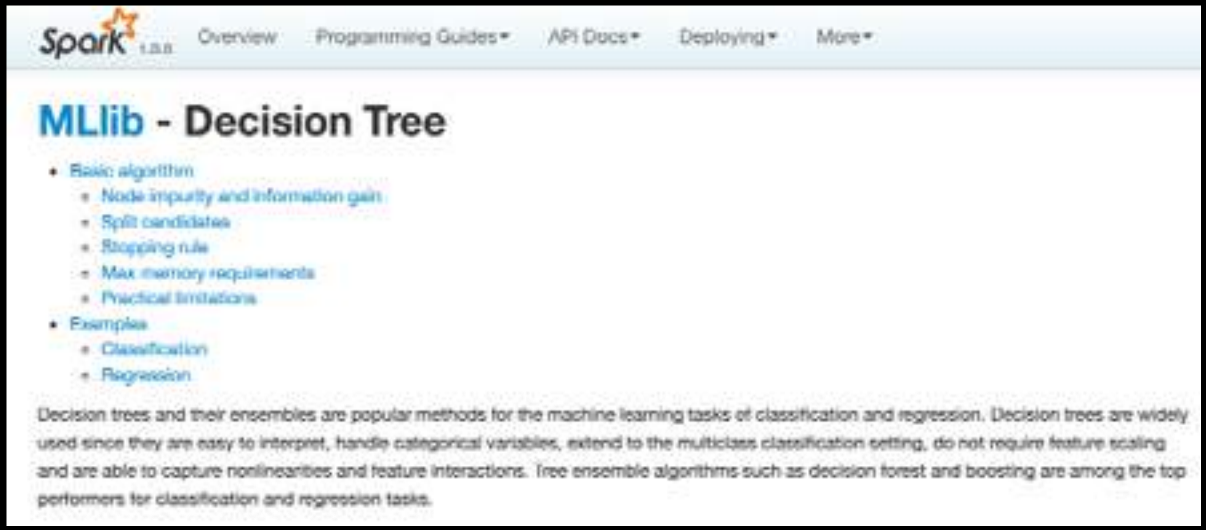
MLlib - Decision Tree

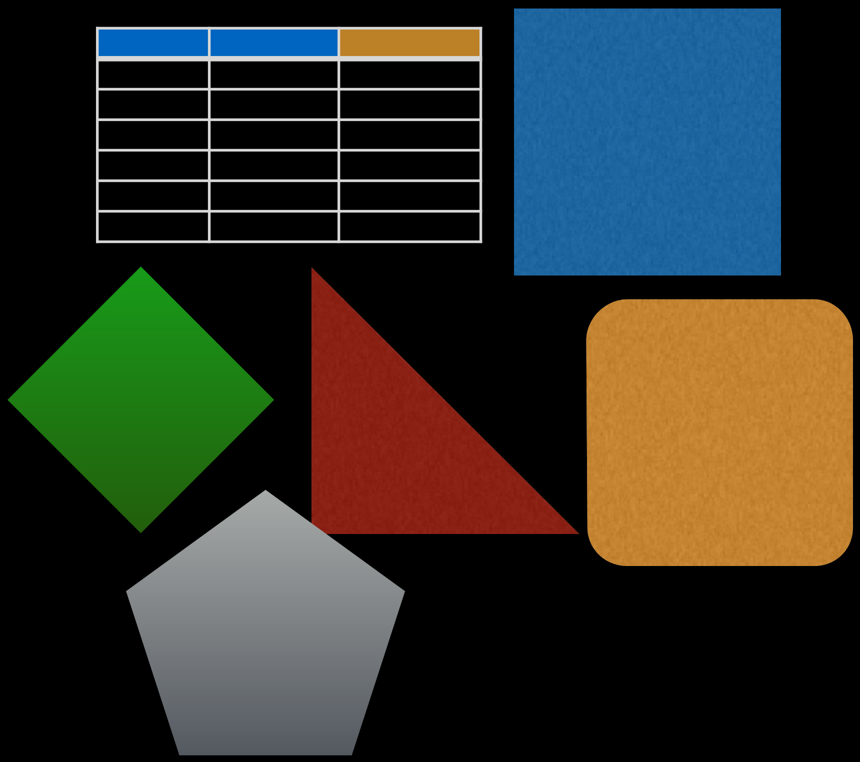
- Basic algorithm
  - Node impurity and information gain
  - Split candidates
  - Stopping rule
  - Max memory requirements
  - Practical limitations
- Examples
  - Classification
  - Regression

Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical variables, extend to the multiclass classification setting, do not require feature scaling and are able to capture nonlinearities and feature interactions. Tree ensemble algorithms such as decision forest and boosting are among the top performers for classification and regression tasks.



Requests


Spark 1.0.0
Overview
Programming Guides
API Docs
Deploying
More
MLlib - Decision Tree
Basic algorithm
Node impurity and information gain
Split candidates
Stopping rule
Max memory requirements
Practical limitations
Examples
Classification
Regression
Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical variables, extend to the multiclass classification setting, do not require feature scaling and are able to capture nonlinearities and feature interactions. Tree ensemble algorithms such as decision forest and boosting are among the top performers for classification and regression tasks.




FEATURE

Thanks