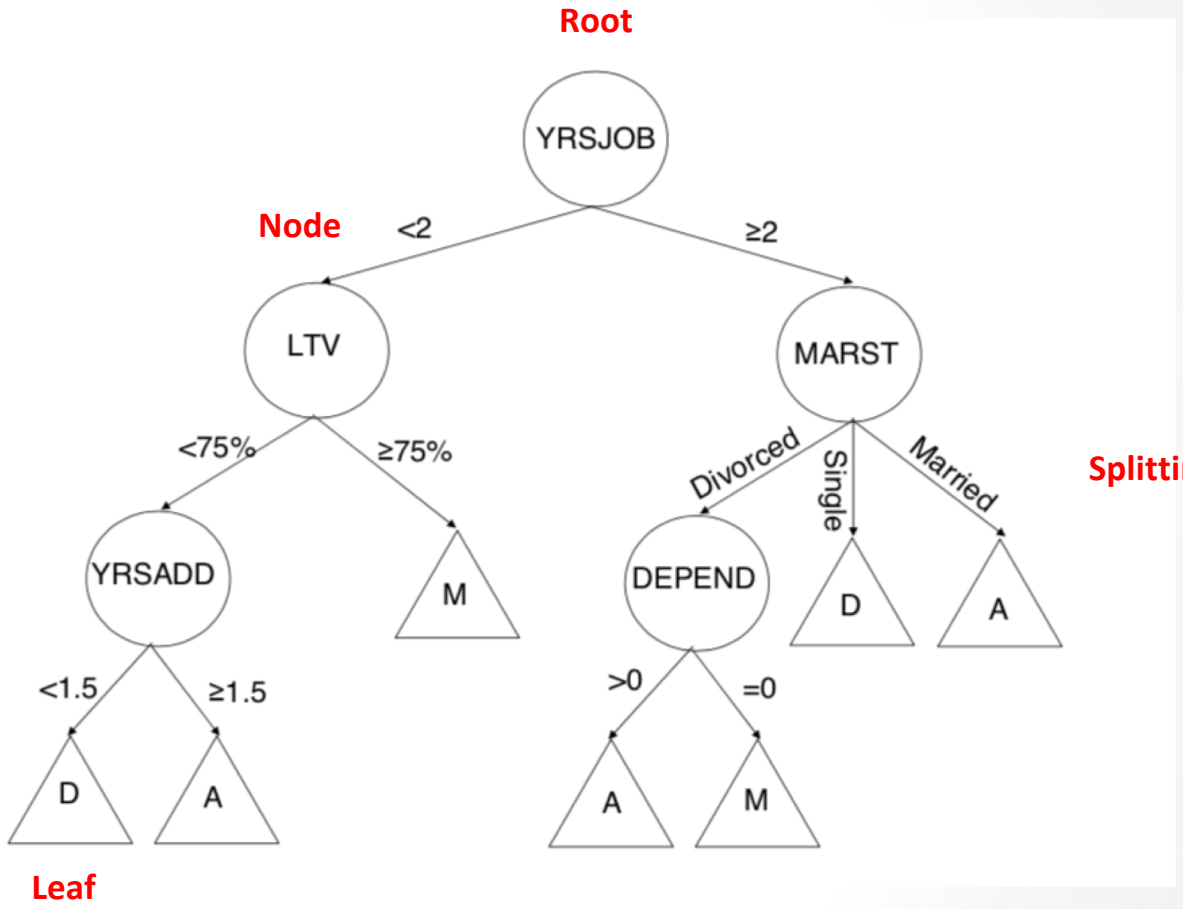
Supervised Learning:

DECISION TREES (RECURSIVE PARTITIONING)

Decision Trees

- supervised learning
- classification or regression
- traditionally created manually
- tree-growing algorithms to optimize decision

$$T : \mathbf{x} \rightarrow y$$

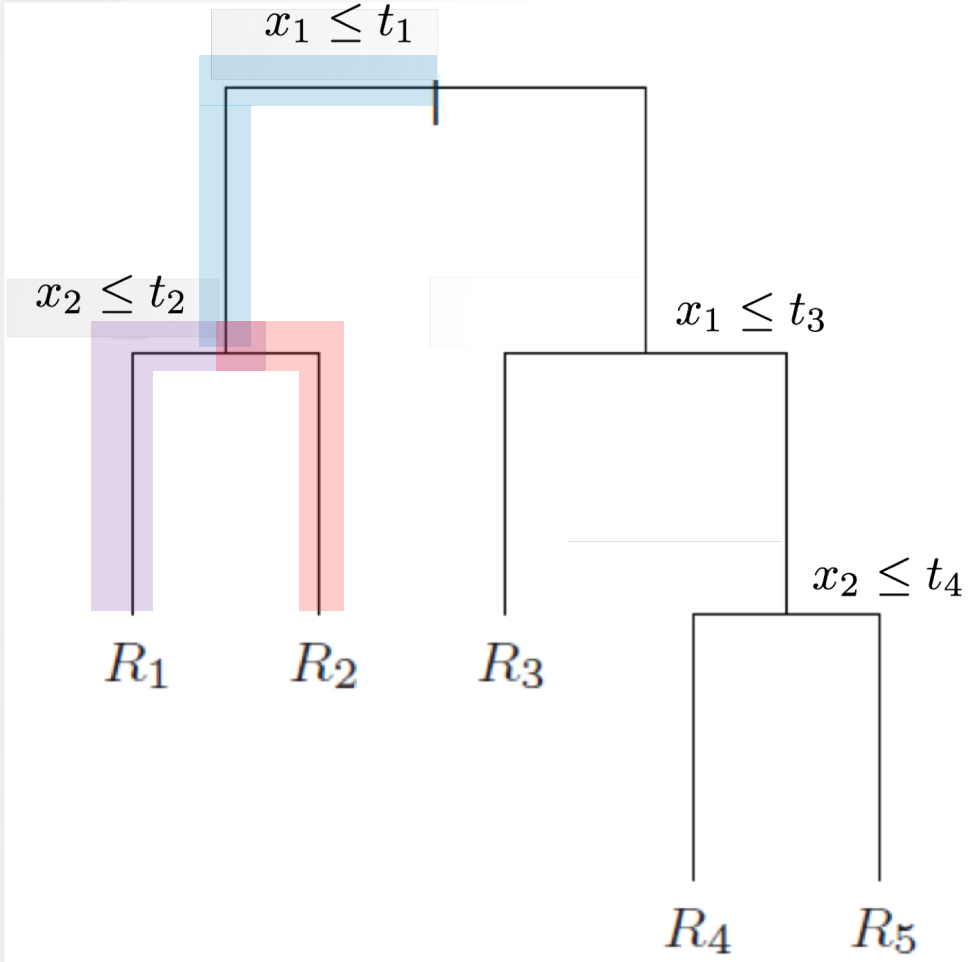


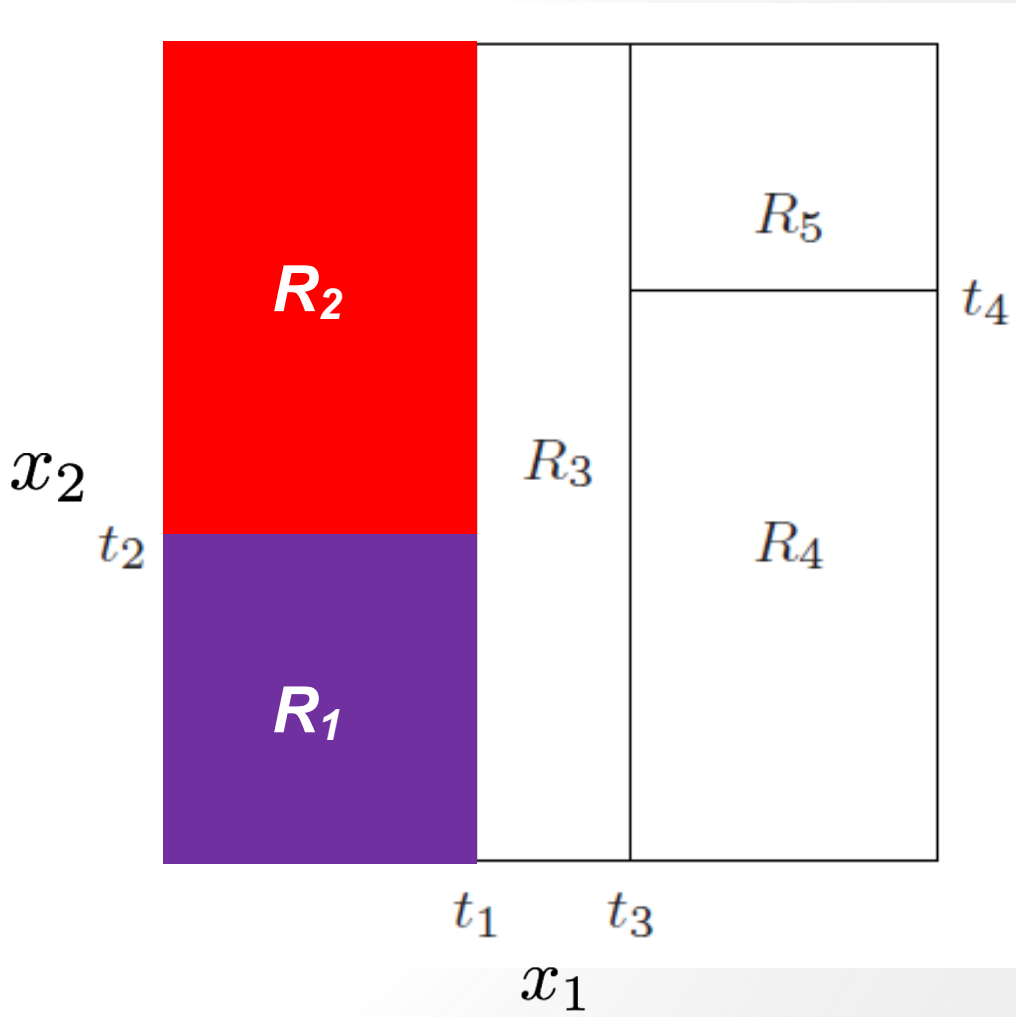
Example: Processing mortgage applications
YRSJOB: Years at current job
YRSADD: Years at current address
DEPEND: # of dependents
MARST: marital status
LTV: loan-to-value ratio

Rokach, Maimon. Data Mining with Decision Trees

Decision Trees

- Decision tree is a recursive partition of the input space
- Most are *binary*





Hastie, Tibshirani, Friedman. Elements of Statistical Learning

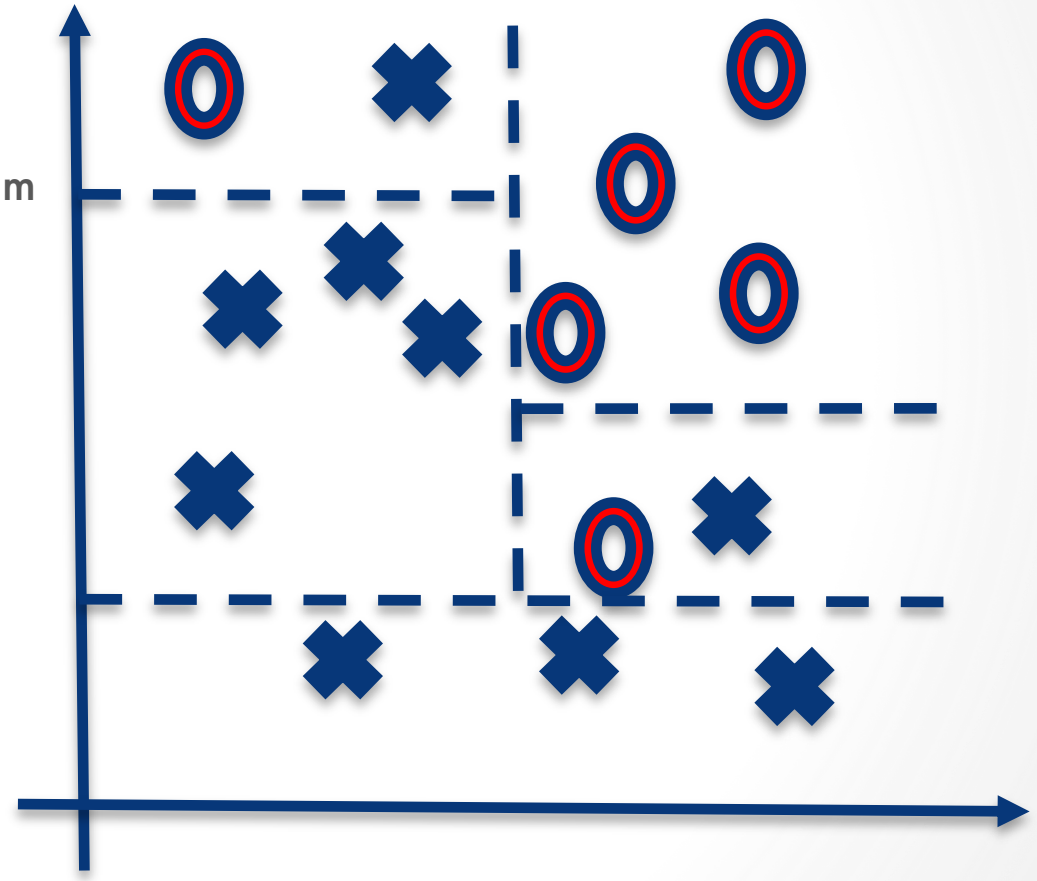
Growing a Tree: Stopping Criteria

1) All instances in leaf have the same y value

2) A maximum tree depth has been reached

3) Number of instances in leaf is below some minimum

4) Splitting criteria is below some threshold



Missing and Unseen Values

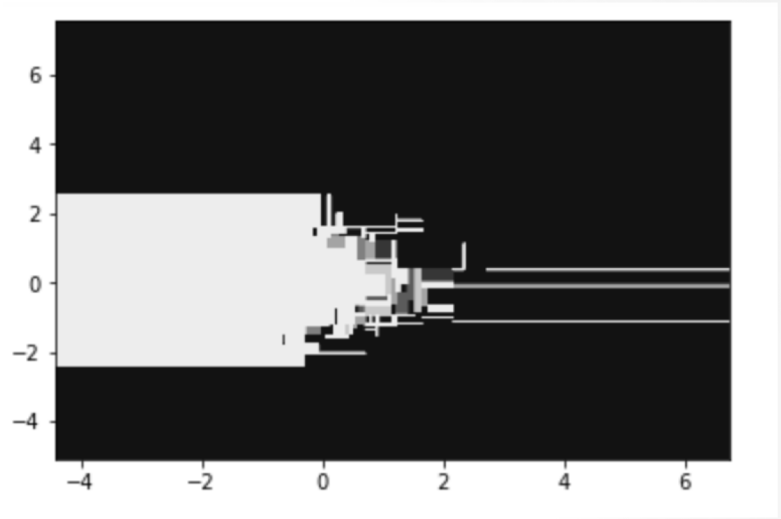
- Missing Data in Training Predictors: Treated as a “level”
- Missing Value in Training Target: Ignored
- Unseen Categorical Levels During Scoring: Treated as an “NA” value and classified with outliers

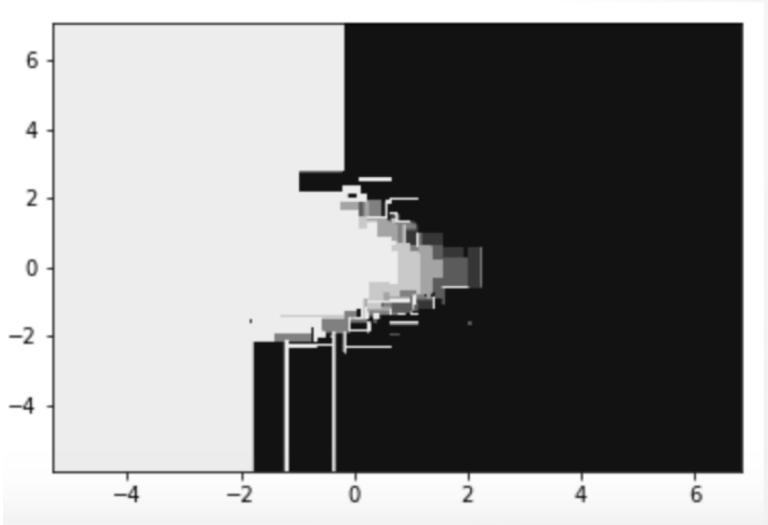
Pros/Cons of Decision Trees

simple to understand/interpret
little data prep
- natural handling of "mixed" data types
- handling of missing values
- handing of multi-class outputs
robustness to outliers in input space
insensitive to monotonic transformations of inputs
computational scalability (large N) $\mathcal{O}(n \log n)$

automatically ignores irrelevant inputs

✗ weak predictors
✗ can be unstable to small variations in the data
✗ poor ability to extract linear relationships
✗ can create biased trees if classes unbalanced

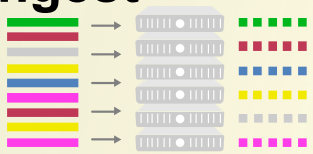




Exercise 1: Creating Decision Tree in sklearn

Scalable Implementation in H2O

1 Parallel Data Ingest



Data is stored in-memory on all cluster compute nodes

- **Rows are evenly distributed across the cluster**
- **Columns are stored separately and compressed**

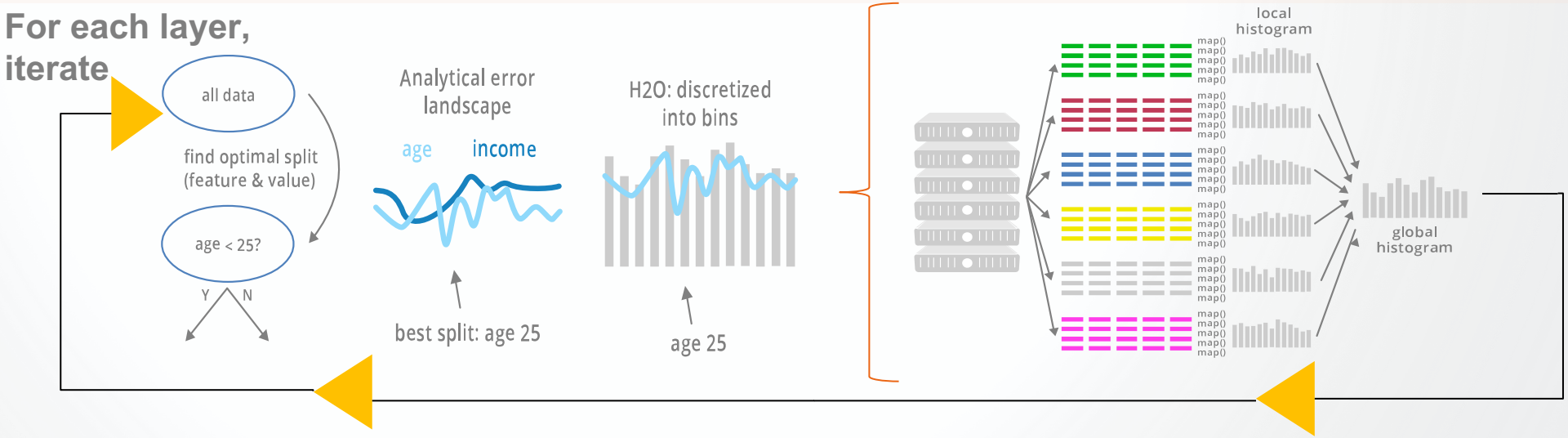
Basis for fine-grain Map/Reduce for histogram calculation

2 Distributed Tree Building via Fine-Grain Map/Reduce to find optimal split points of data layer by layer

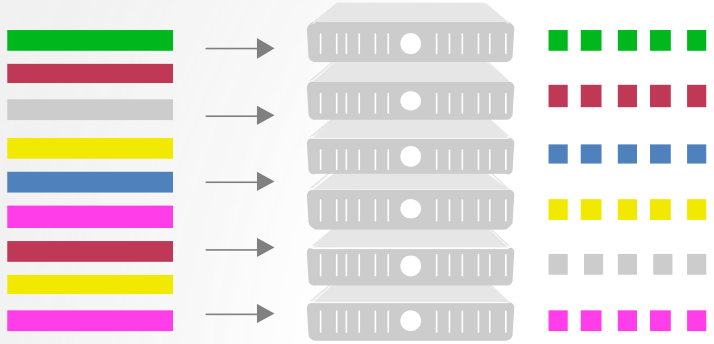
Start with root node and build layers of tree nodes **[ILLUSTRATION BELOW]**

For each layer, repeat the following:

- **For a set of features, split the data at every possible split point**
- **Find the split that leads to best model improvement**
- **Use discretization to limit the number of potential splits**
  - **To find the split, local histograms are calculated on each node and then aggregated into a global histogram**
  - **From the global histogram, the best split column is chosen**



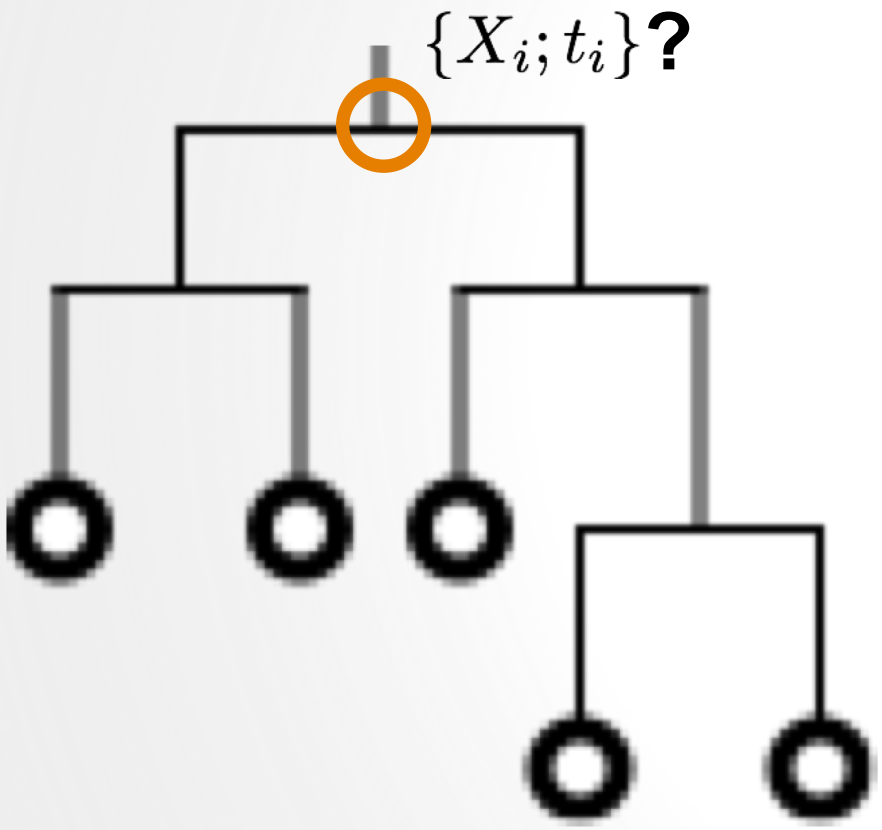
Scalable Implementation in H2O



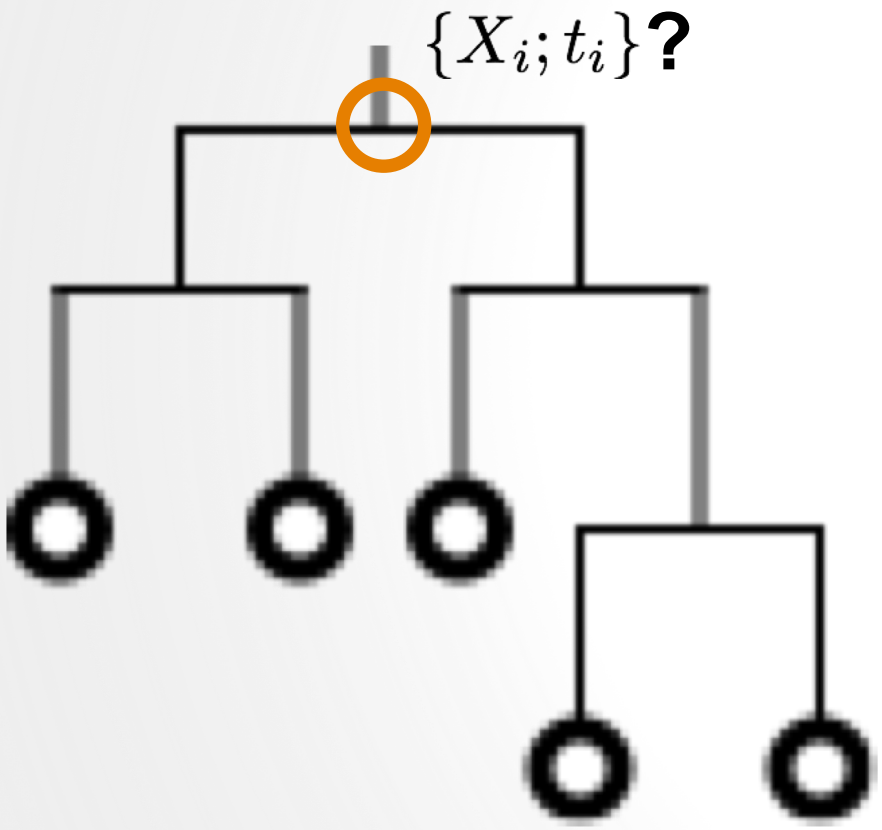
Parallel Parse into **Distributed Rows**

Fine Grain Map Reduce Illustration: Scalable Distributed Histogram Calculation for GBM

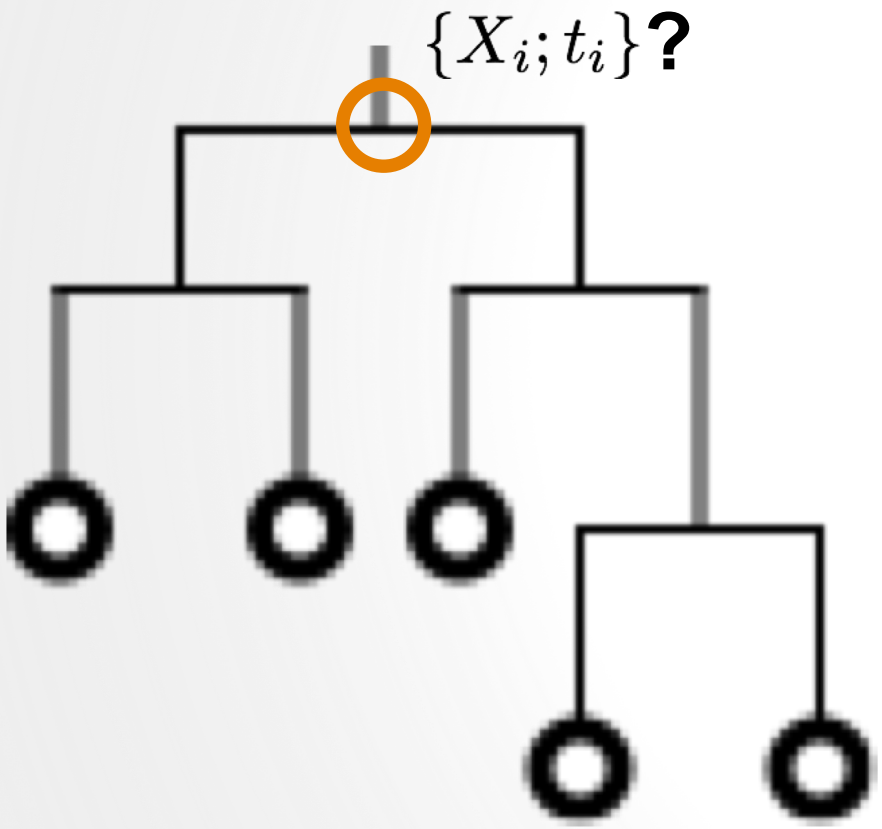
Tree Growth with Data Parallelism

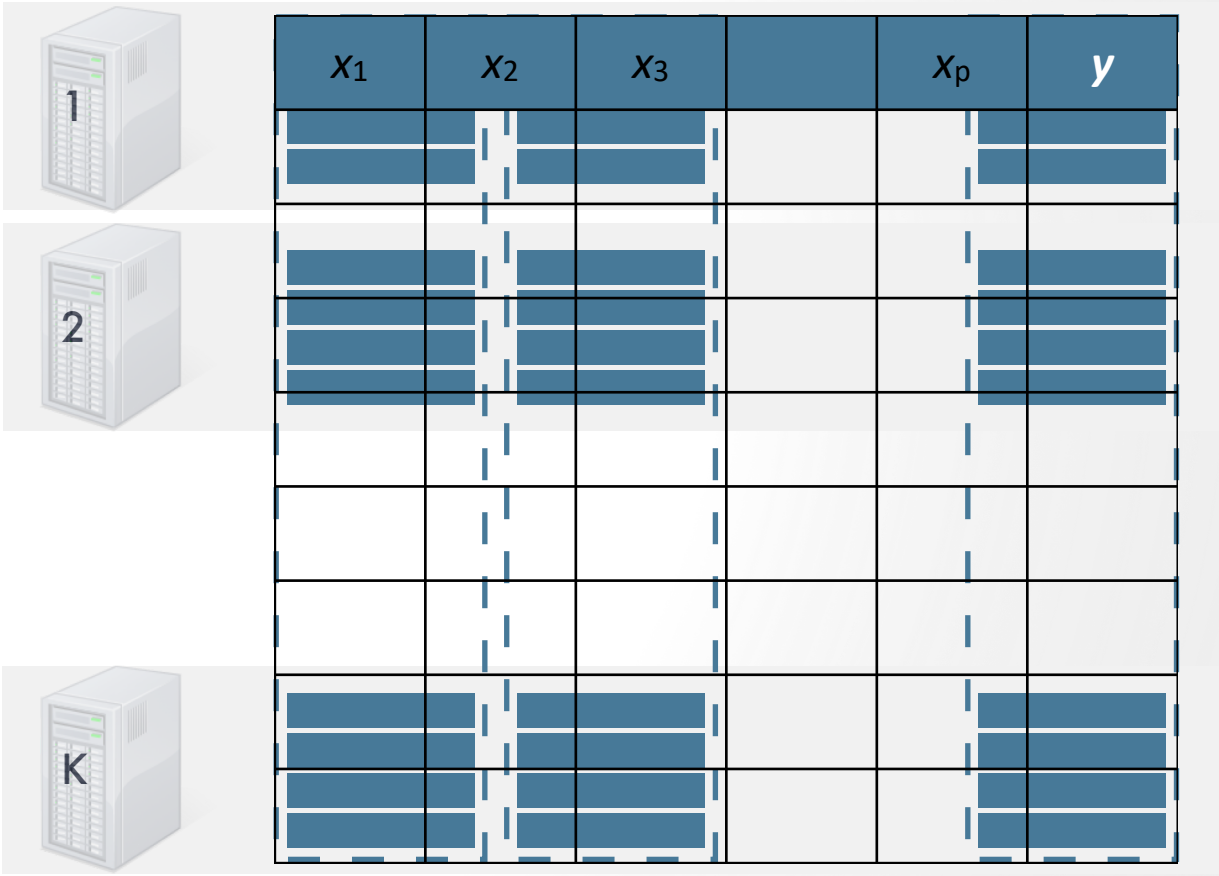


Tree Growth with Data Parallelism

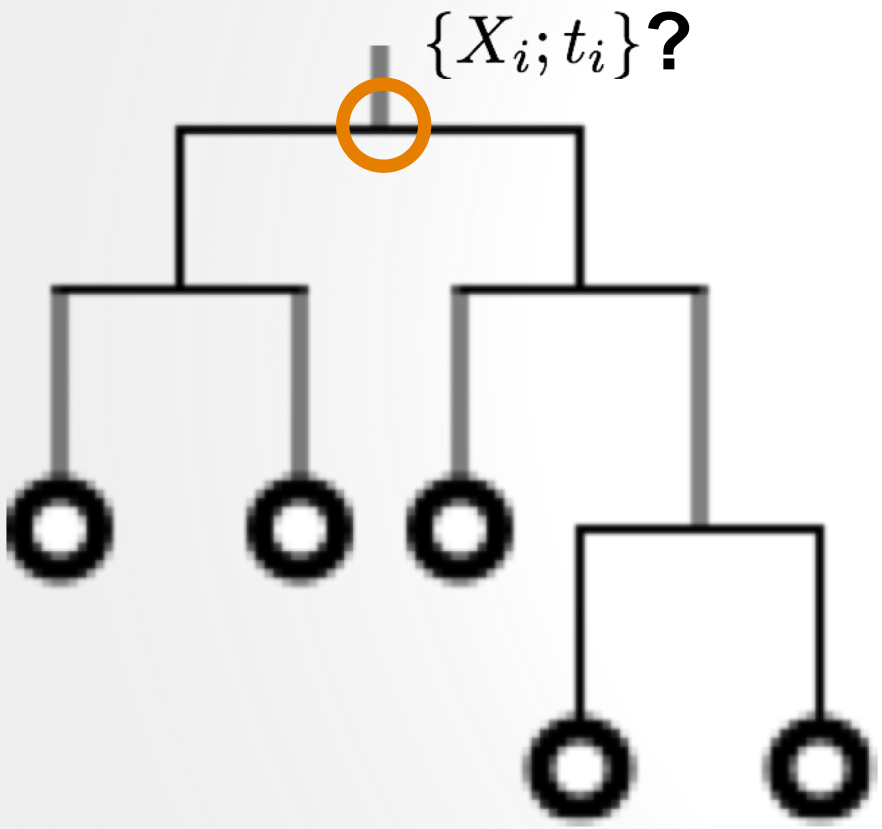


Tree Growth with Data Parallelism

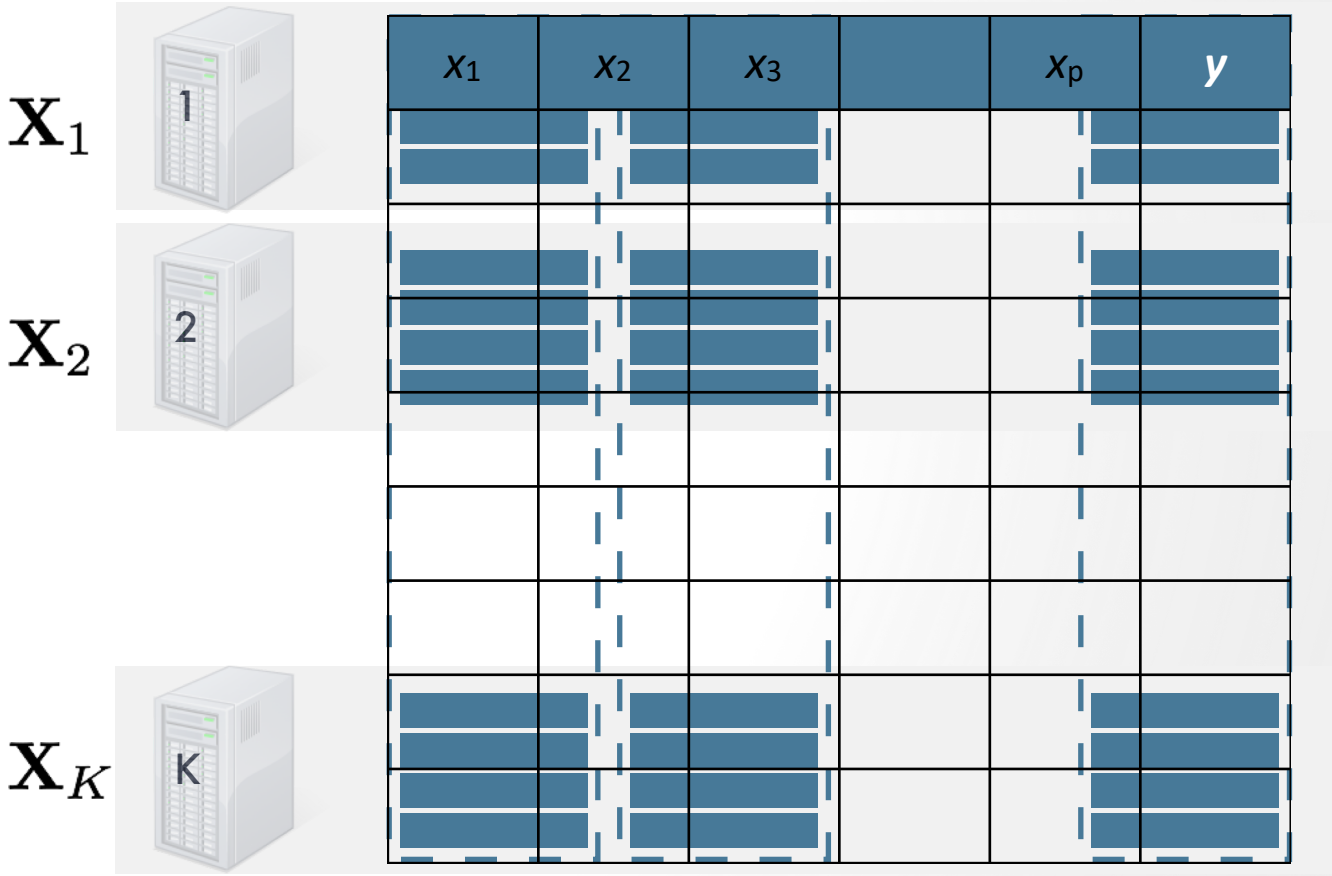




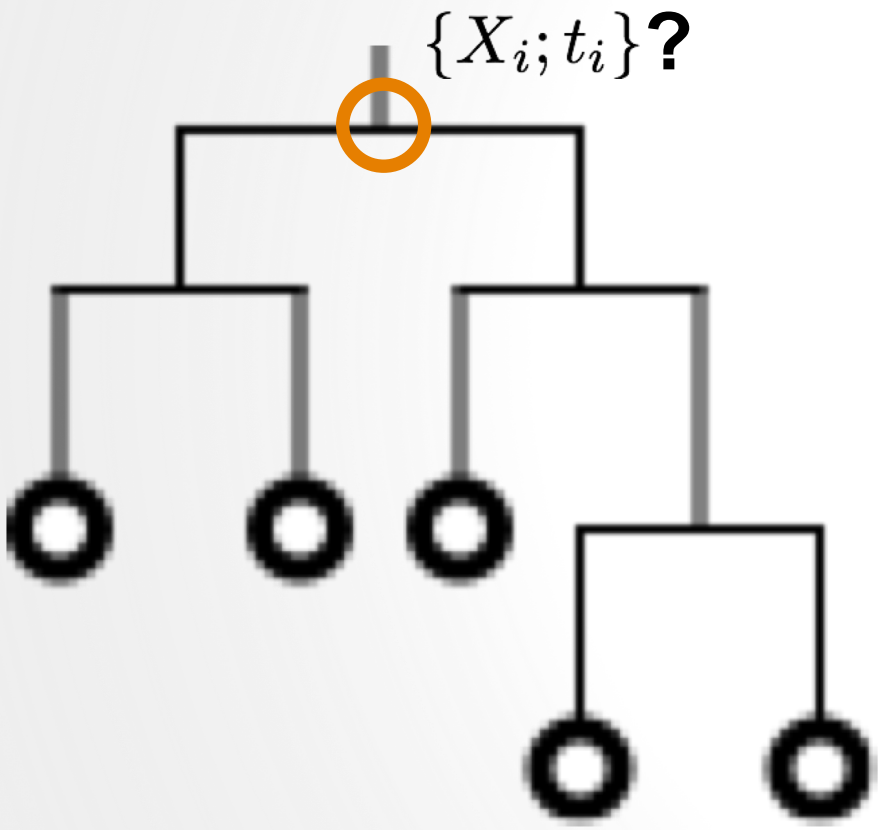
Tree Growth with Data Parallelism



$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

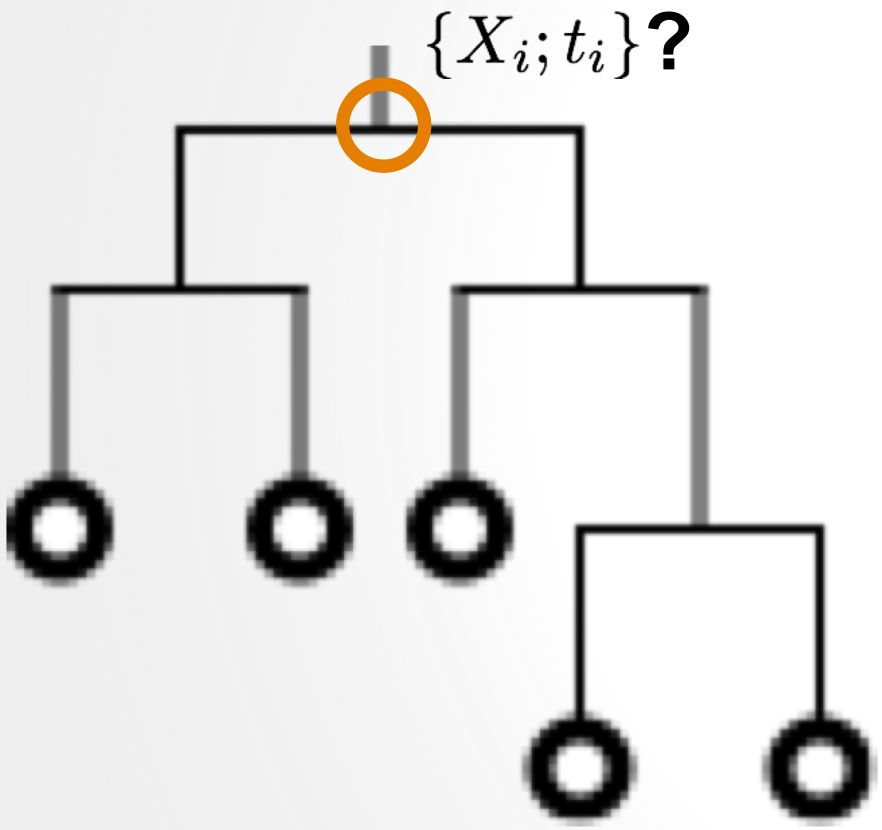


Tree Growth with Data Parallelism



$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

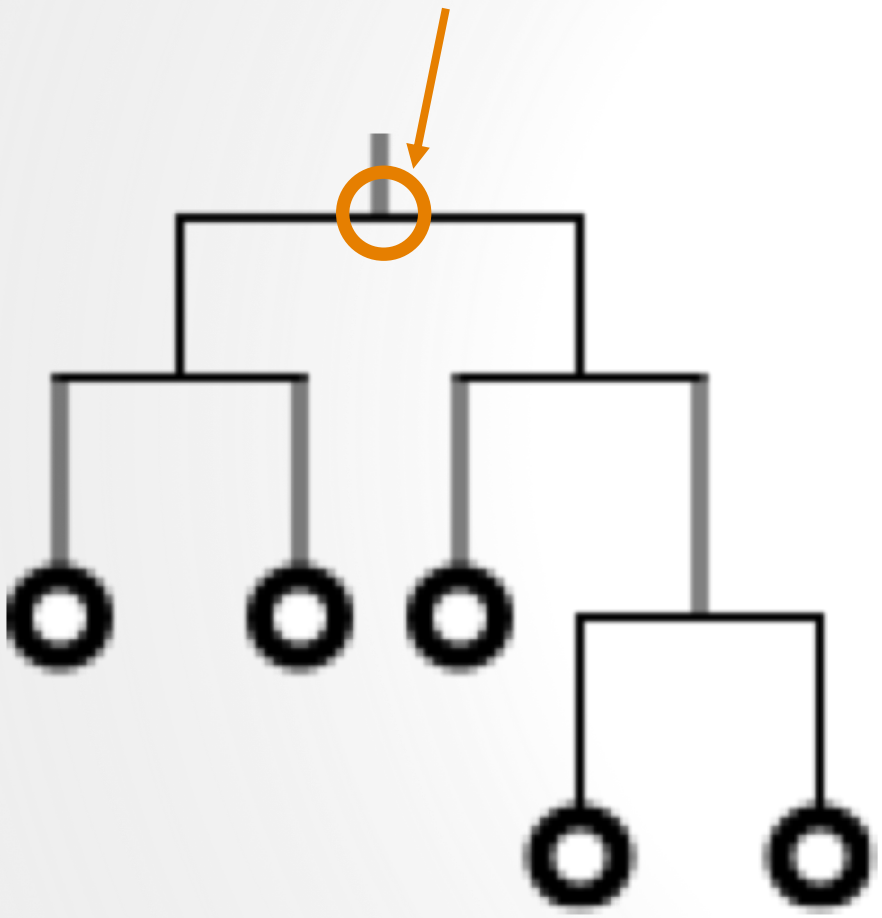
Tree Growth with Data Parallelism



$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$

Tree Growth with Data Parallelism

$$\{X_i; t_i\} = f(\mathbf{math}\,(\mathbf{X_1}), \ldots, \mathbf{math}\,(\mathbf{X_K}))$$

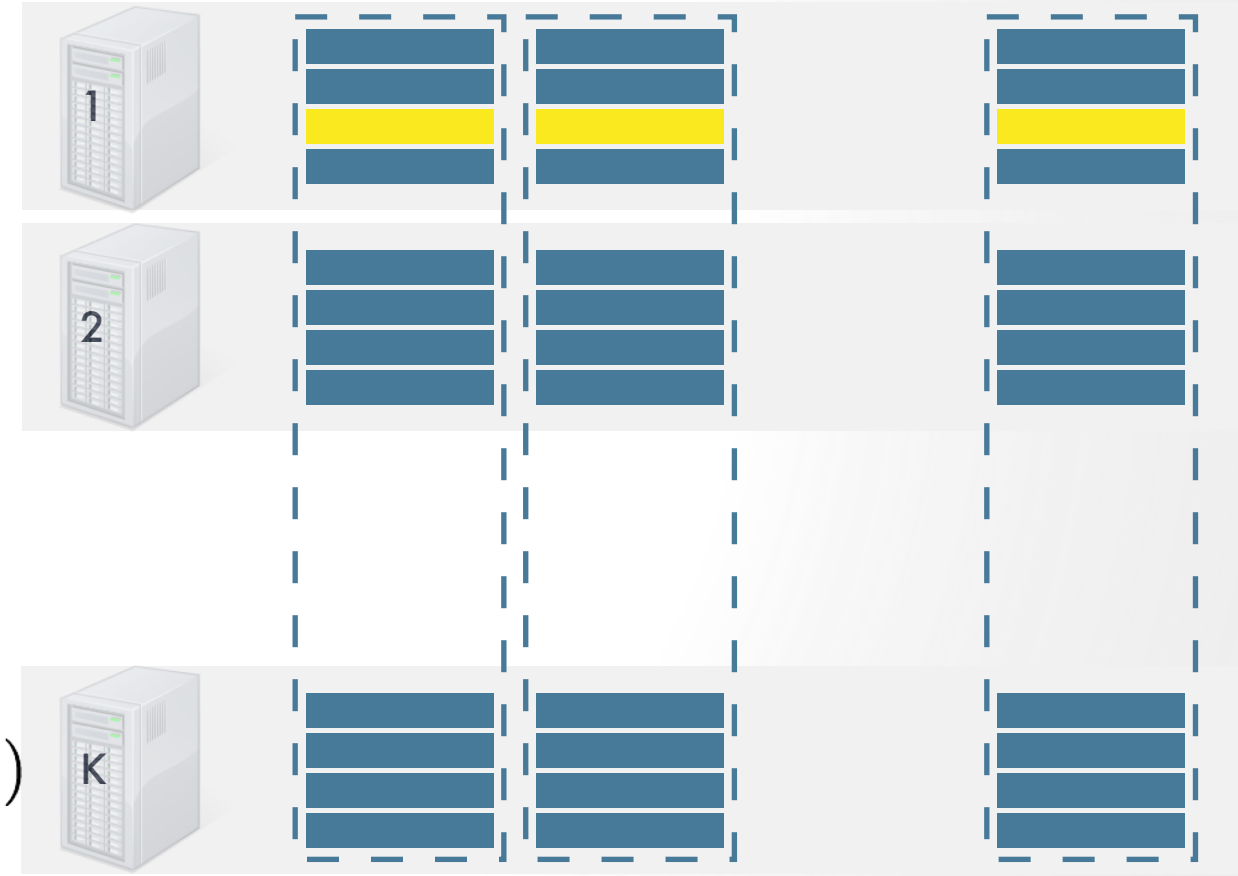


$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

$\mathbf{math}\,(\mathbf{X_1})$

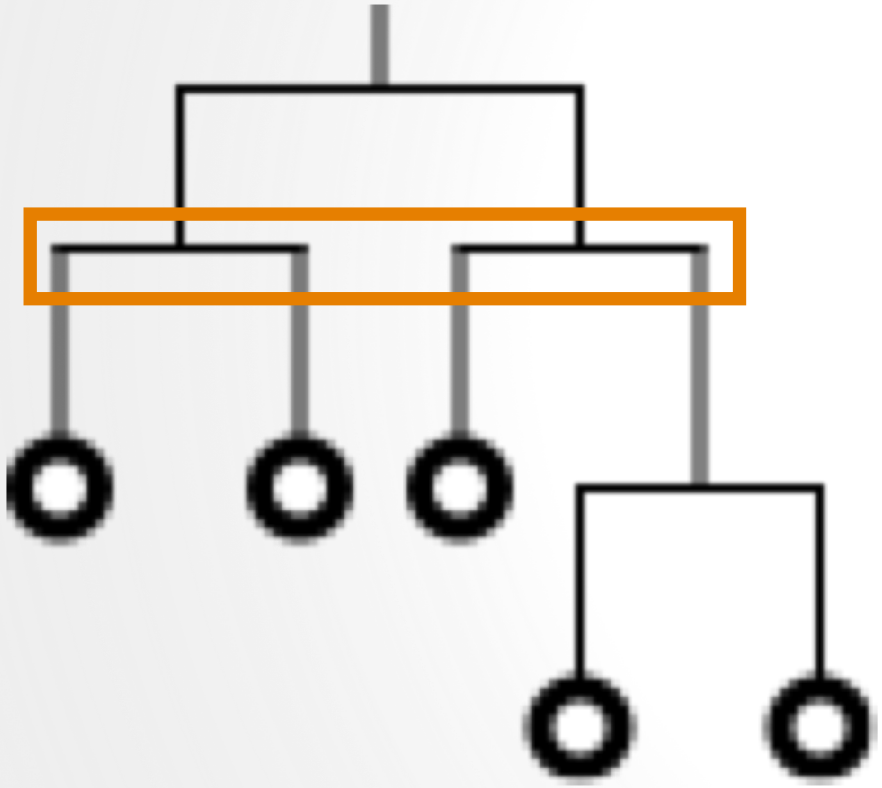
$\mathbf{math}\,(\mathbf{X_2})$

$\mathbf{math}\,(\mathbf{X_K})$



Tree Growth with Data Parallelism

$$\{X_i; t_i\} = f(\mathbf{math}(\mathbf{X_1}), \ldots, \mathbf{math}(\mathbf{X_K}))$$

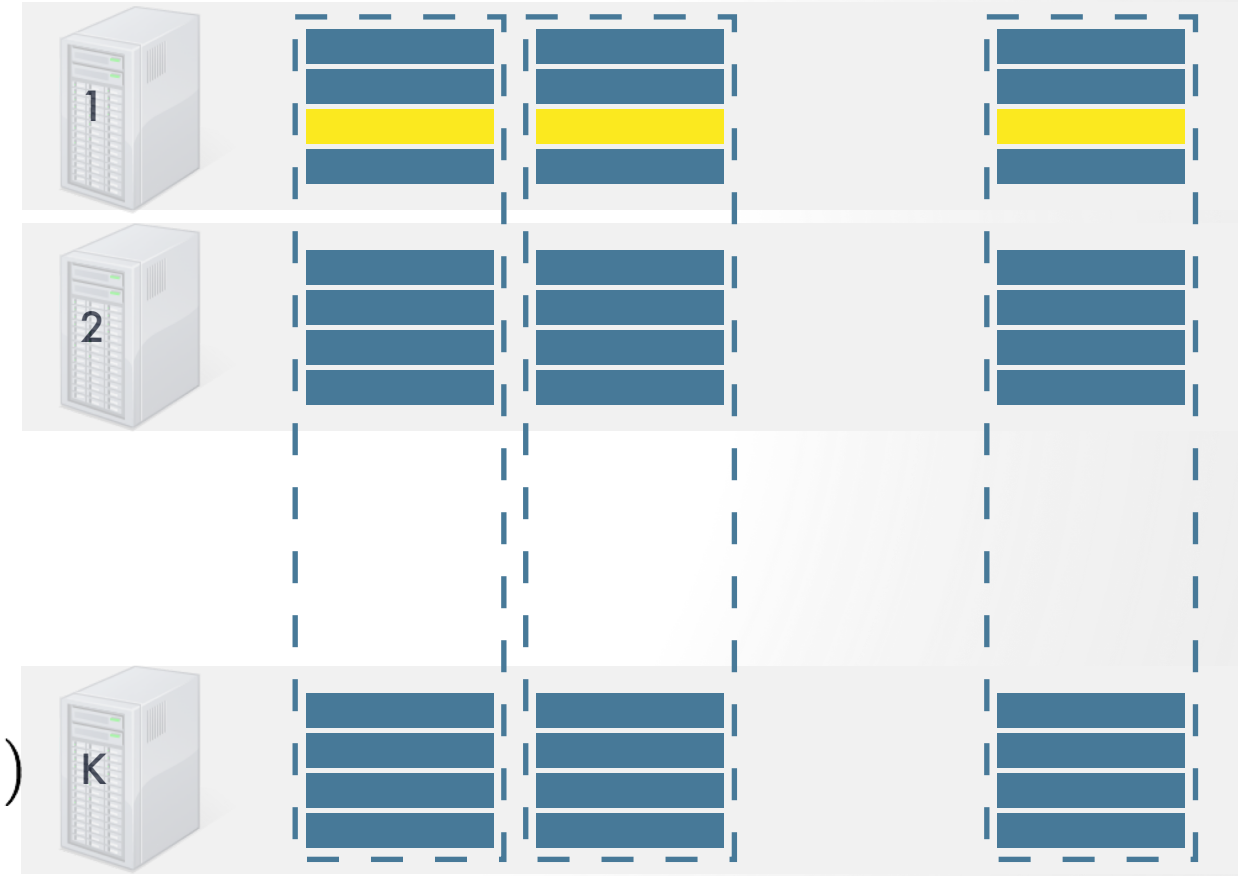


$$\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$$

$\mathbf{math}(\mathbf{X_1})$

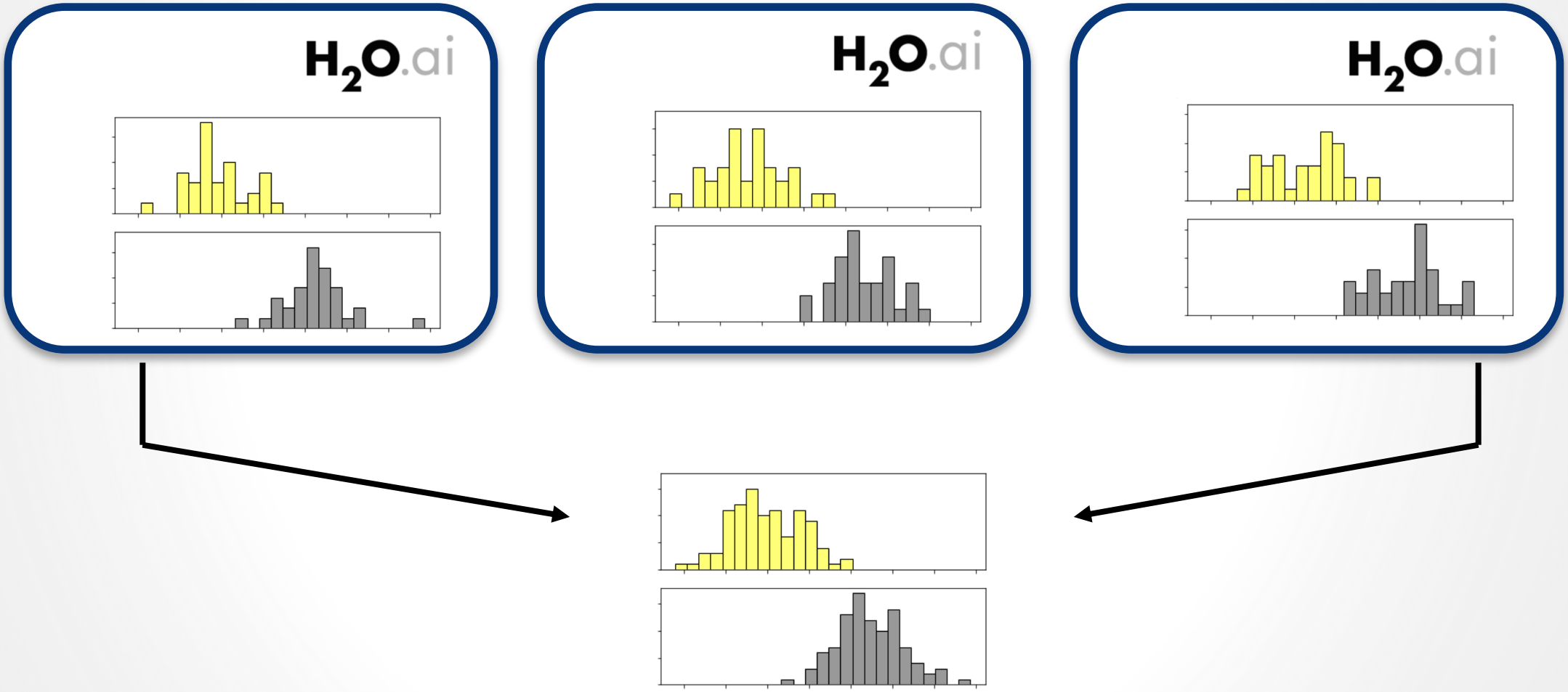
$\mathbf{math}(\mathbf{X_2})$

$\mathbf{math}(\mathbf{X_K})$

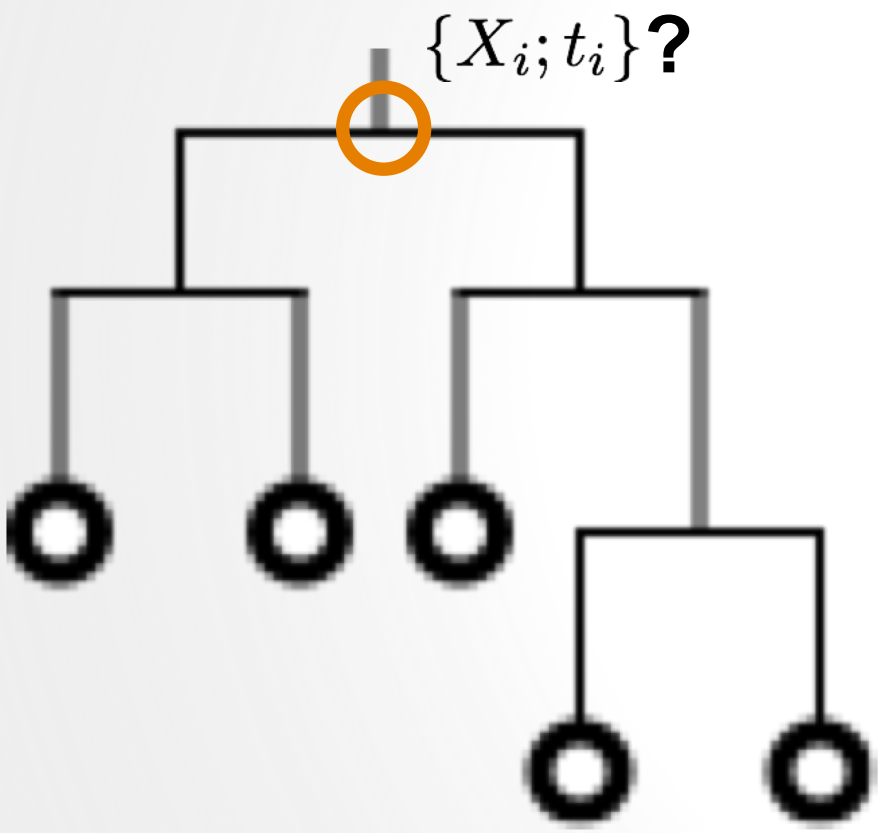


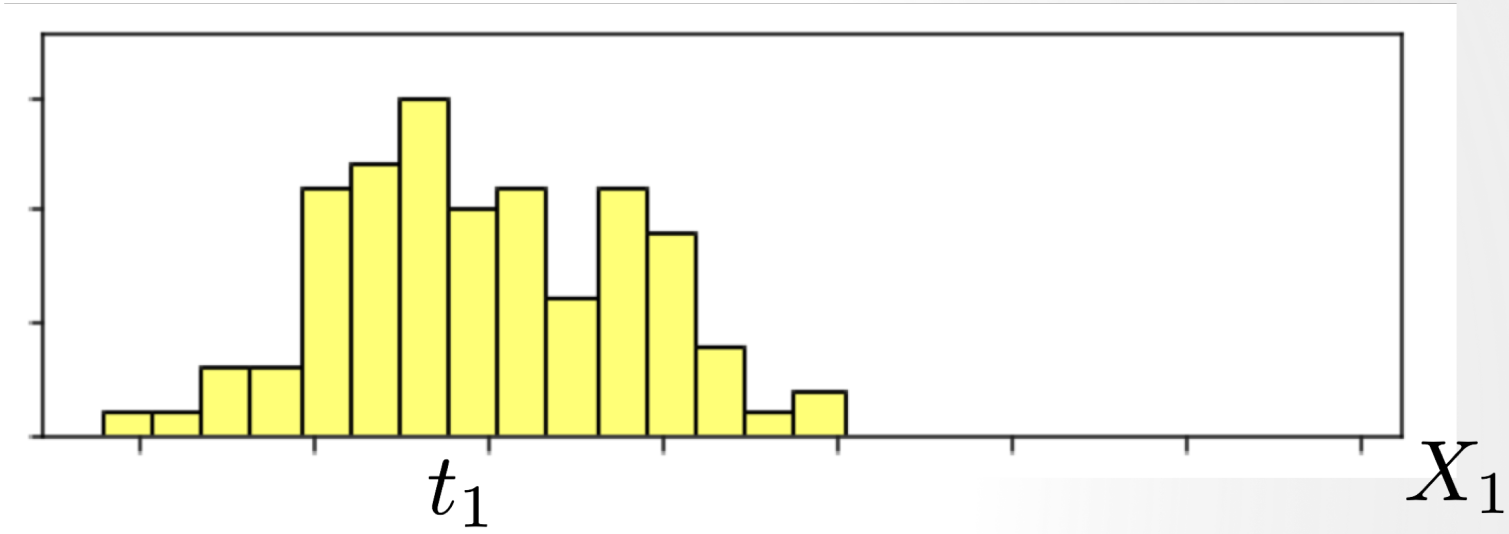
Full Data Parallelism for Each Level of Tree Growth!

Splitting with Distributed Data



Numerical Binning (Histogramming)





- **`nbins`**: number of bins in histogram
- **`nbins_top_level`**: number of bins to use at the top node, then halve at each ensuing level
- **`histogram_type`**: method for binning {Uniform Adaptive, Random, QuantilesGlobal, RoundRobin}

Categorical Binning

- Lexigraphical ordering (i.e. alphabetical ordering)

- Example: {Red, Blue, Yellow, Orange, Purple, Green}
  - Lexigraphical order: {Blue, Green, Orange, Purple, Red, Yellow}
  - **`nbin_cats = 2`**: {Blue, Green, Orange}, {Purple, Red, Yellow}
  - **`nbin_cats = 3`**: {Blue, Green}, {Orange, Purple}, {Red, Yellow}
  - **`nbin_cats >= 6`**: {Blue}, {Green}, {Orange}, {Purple}, {Red}, {Yellow}

Binning in H2O Decision Trees

- Binning Numeric Features
  - Traditionally split points are chosen by sorting the each feature and inspecting an induced split.
  - For big data even when running parallel and distributed this can be computationally expensive so we approximate sorting with binning.
  - **More Bins, More Accurate** The number of bins can be specified by the user and it is the minimum number of bins required in a histogram built for each feature.
- Binning Categorical Features
  - **High Cardinality Features** slow model builds by inducing splits by each level.
  - Bin the levels in a categorical column according to “bins_cat” parameter.
  - **More Bins, More Likely To Overfit** Increasing the number of bins can lead to splits on a single category, which can lead to overfitting.