Parallel Binary Decision Tree in *Regent*

Stanford CS315B Course Project

Li Deng

12.07.2017

Background

Features

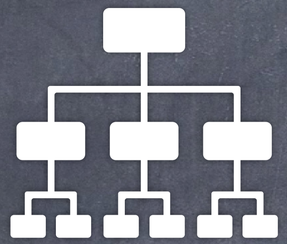
Row	*f1*	*f2*	*f3*	*Label*
1	...	...	...	0
2	...	...	...	1
3	...	...	...	1
4	...	...	...	0

(X_train, y_train)

X_test

Training

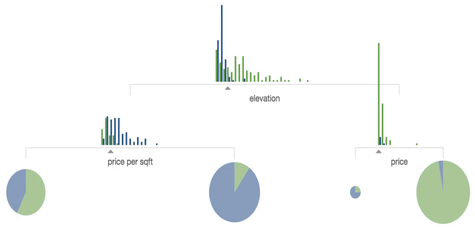
Testing

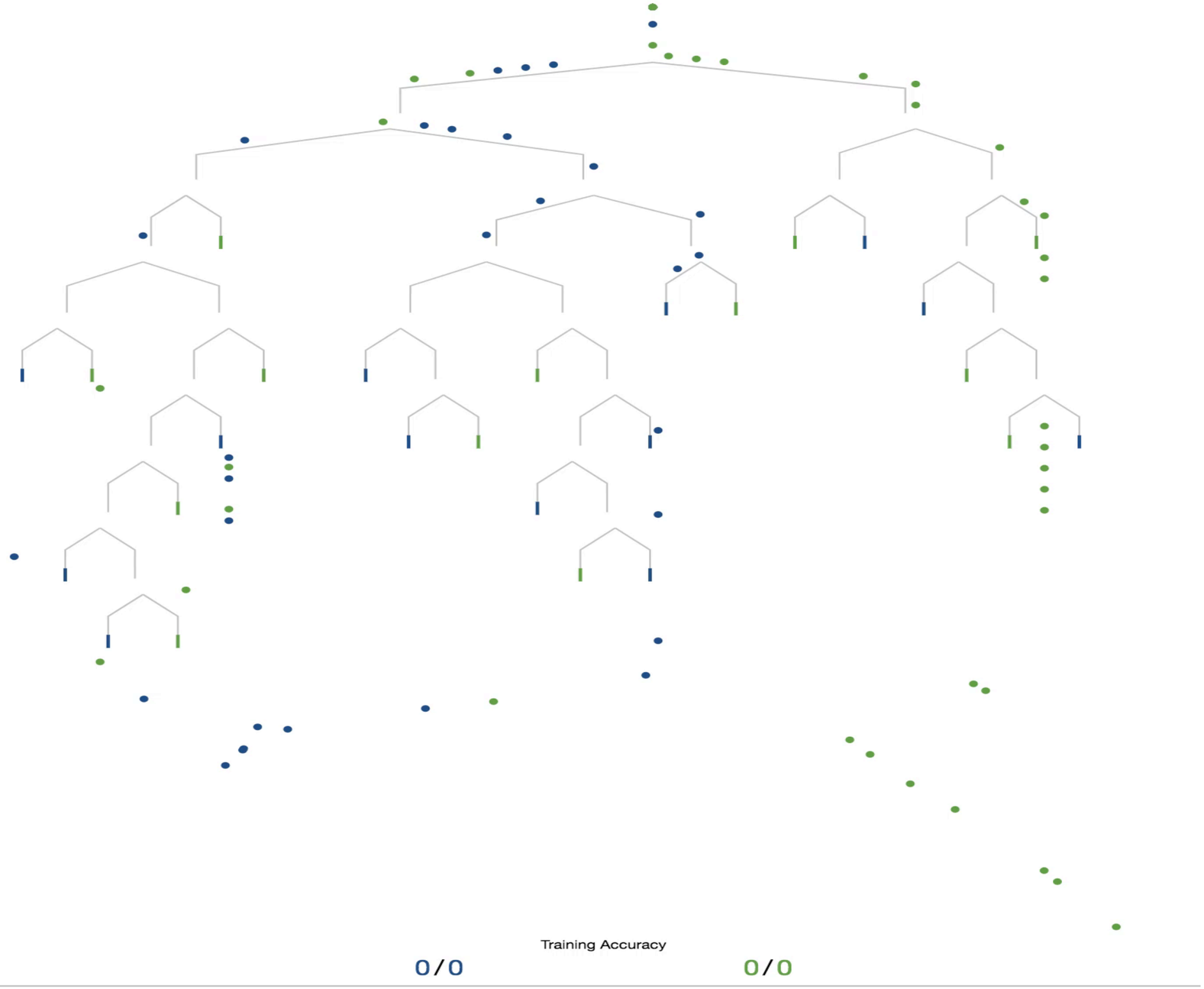


y'
predictions

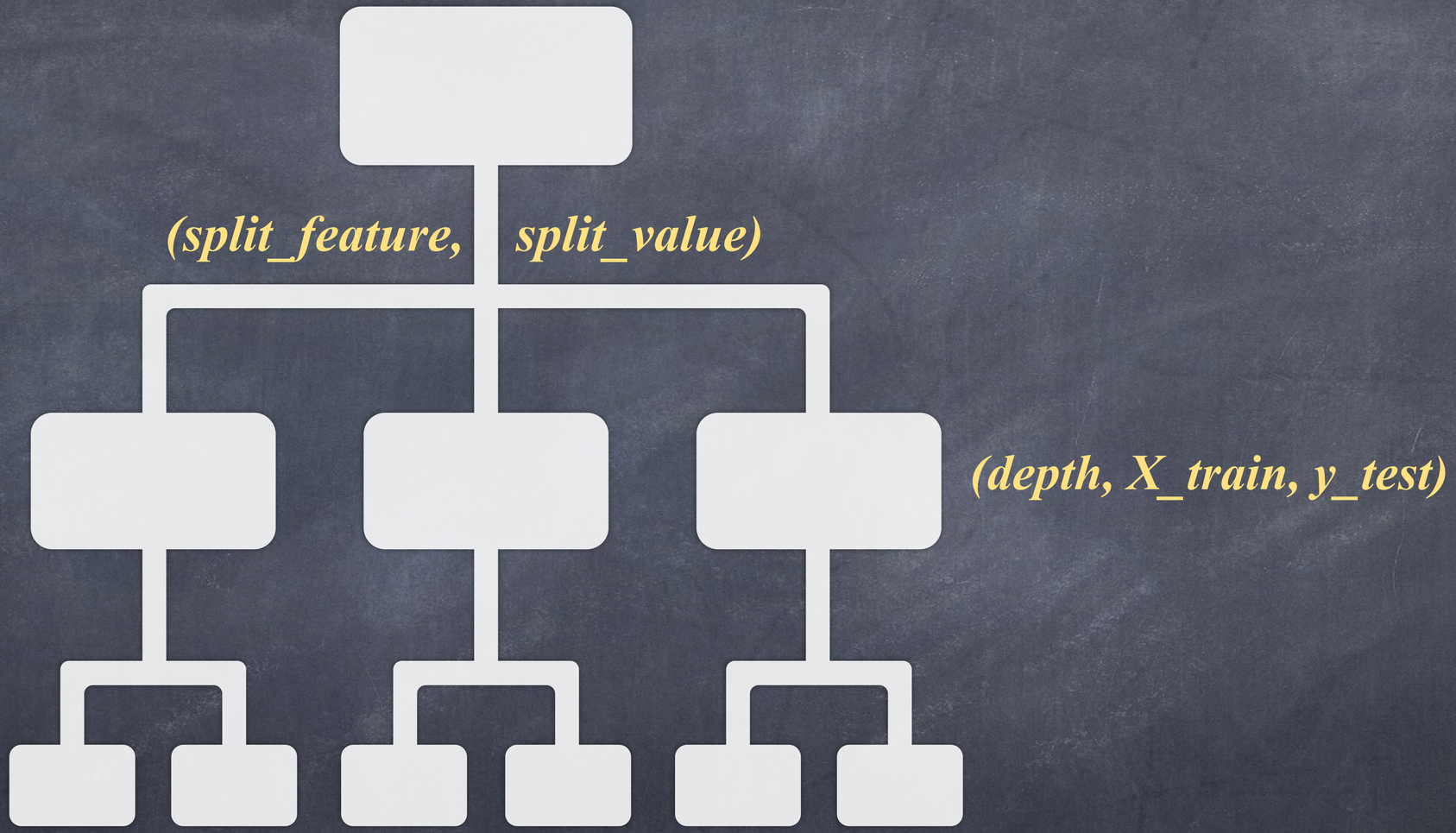
y_test

Accuracy





Source: D2R3


(split_feature, split_value)
(depth, X_train, y_test)

Algorithm

Training

- ***Init**: Tree Root = (d = 0, D = X_train)*
- ***While** (depth <= max_depth && purity > 0)*
  - *For each feature*
    - *For each value*
      - *Take it as split value, compute impurity*
    - *Compute best impurity for this feature*
  - *Pick the best feature in terms of purity as split feature and its associated best value as split value*
  - *Split current node into two*
- ***Label** current node by majority*

Testing

- ***For a certain data point, starting from root***
- ***While** (label == ?)*
  - *Use split_feature of current node to split*
  - ***If** f <= split_value : go to left subtree*
  - ***Else** go to right subtree*
- ***Reach Leaf Node, use its label as prediction***

Scope

- Binary Decision
- Fixed Number of Features
- Numeric Features
- Impurity Measure: *Gini Impurity*

$$I_G(p) = 1 - \sum_{i=1}^{J} {p_i}^2$$

Datasets

- ***Iris***
    - Toy Dataset: (*100* rows, *4* features)

- ***Adult Income***
    - Medium Dataset: (*7000* rows, *3* features)
    - Large Dataset: (*30000* rows, *7* features)

Sequential Implementation

Sequential Implementation

- First implemented in *python*, for validation and performance comparison

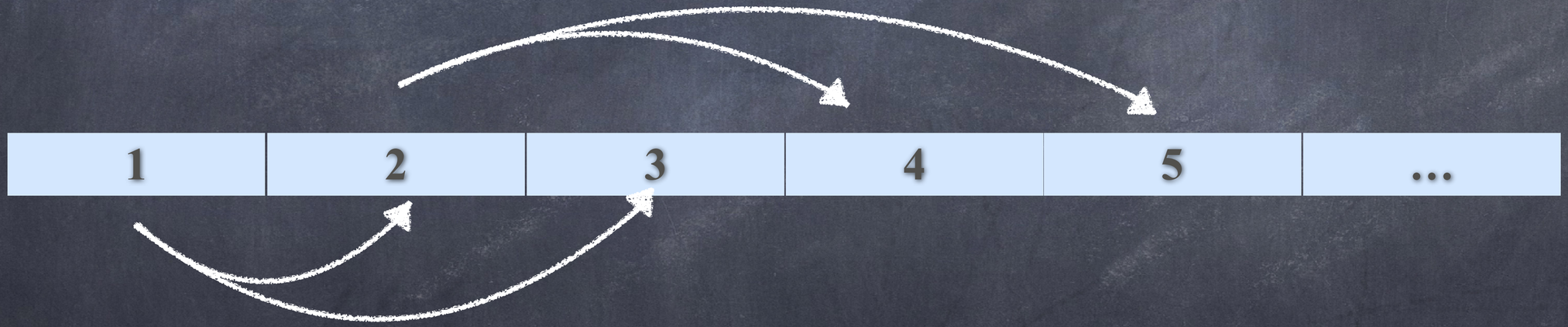
- *Regent:*

```
-- Field Space for each data point
-----------------------------------
fspace DataPoint {
  -- row number, i.e. ID of this data point
  row        : uint64;
  -- classification label
  label      : uint32;
  -- features: an array of cells
  features   : float[num_feature];
}
```

```
-- Field Space for decision tree node
-----------------------------------
fspace Tree{
    left : uint32,
    right: uint32,
    depth: uint32,    -- depth
    max_depth: uint32,    -- depth
    -- for leaf node only, {-1: non-labeled | 0/1: label}
    label         : int32,
    -- index of splitting feature | -1 for not splitting
    split_feature : int32,
    split_val     : float;    -- value of splitting feature
    gini          : float;    -- gini index
    n             : uint64;    -- number of data points in this node
    data          : uint64[max_row]
}
```

How to represent recursive structure of tree nodes?

Preallocate Region of Tree Nodes

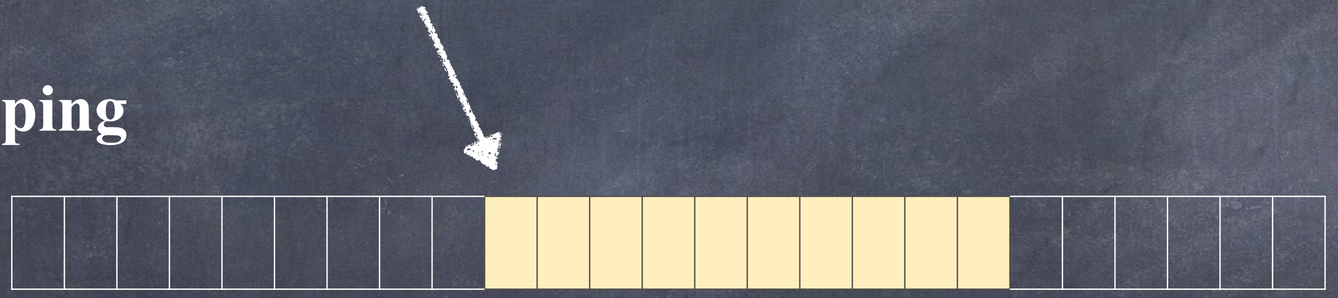


Static Array Too Slow!

Create Indirect Mapping Region

- Region of Tree Nodes

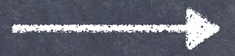
- Region of Mapping



- Region of Data

Indirect Mapping

```
-- Field Space for decision tree node
-----------------------------------
fspace Tree{
    left : uint32,
    right: uint32,
    depth: uint32,            -- depth
    max_depth: uint32,        -- depth
    -- for leaf node only, {-1: non-labeled | 0/1: label}
    label         : int32,
    -- index of splitting feature | -1 for not splitting
    split_feature : int32,
    split_val     : float;    -- value of splitting feature
    gini          : float;    -- gini index
    n             : uint64;   -- number of data points in this node
    mapping_head  : uint64;
}
```



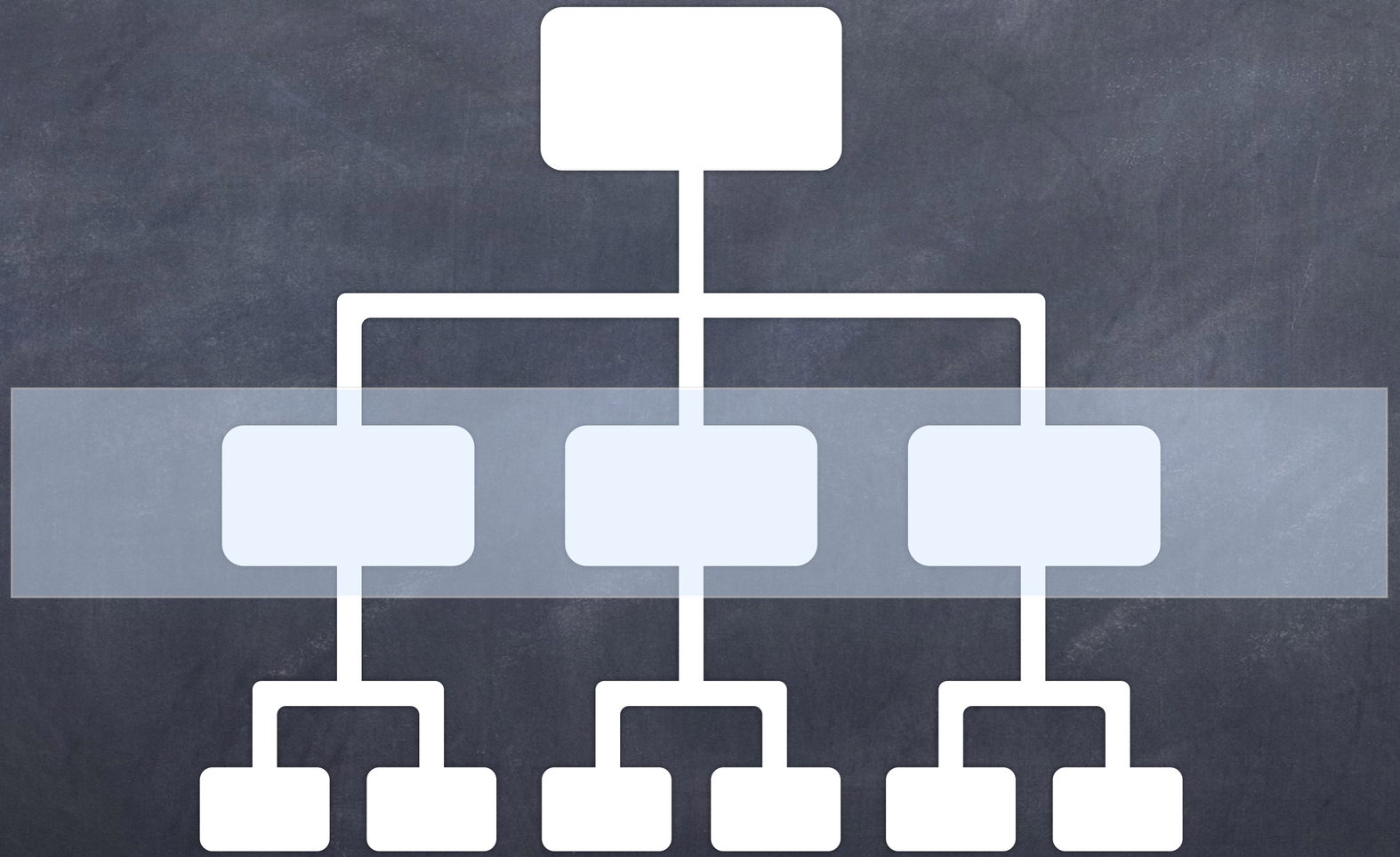
```
-- Field Space for mapping
-----------------------------------
fspace Mapping {
  -- row number, i.e. ID of this data point
  row         : uint64;
}
```

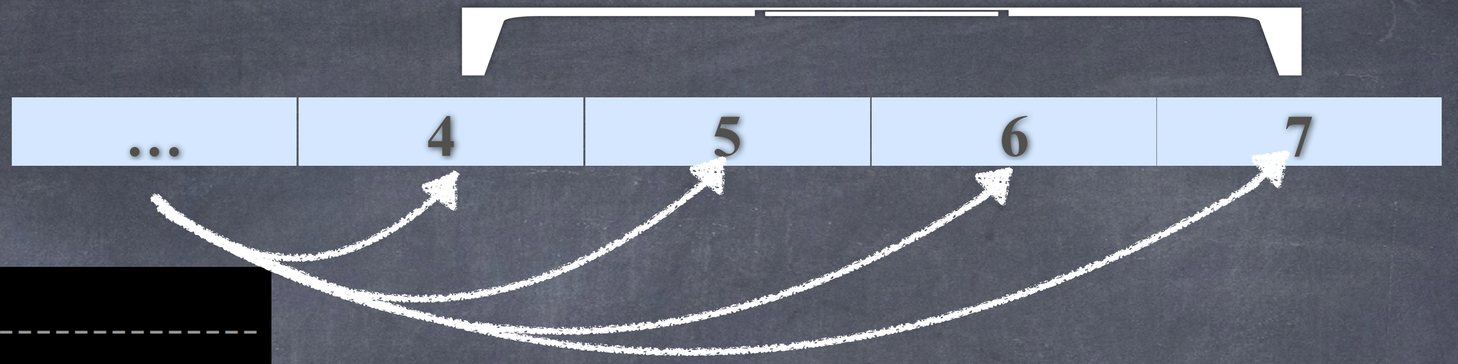
```
-- Field Space for each data point
-----------------------------------
fspace DataPoint {
  -- row number, i.e. ID of this data point
  row         : uint64;
```

Parallel Implementation

Parallel Training

Training of subtrees at the same level are independent






```
-- train a tree on data points
-------------------------------------------------------------------
task train(r_trees : region(ispace(int1d), Tree),
     |    |  r_data_points : region(ispace(int1d),DataPoint),
     |    |  r_mapping : region(ispace(int1d),Mapping),
     |    |  num_tree : int)
```

```
-- create coloring of tree region
var tree_coloring = ispace(int1d, num_tree)
-- create a partition of tree
var tree_partition = partition(equal, r_trees, tree_coloring)

-- create coloring of map
var map_coloring = ispace(int1d, num_tree)
-- create a partition of map
var map_partition = partition(equal, r_mapping, map_coloring)
```

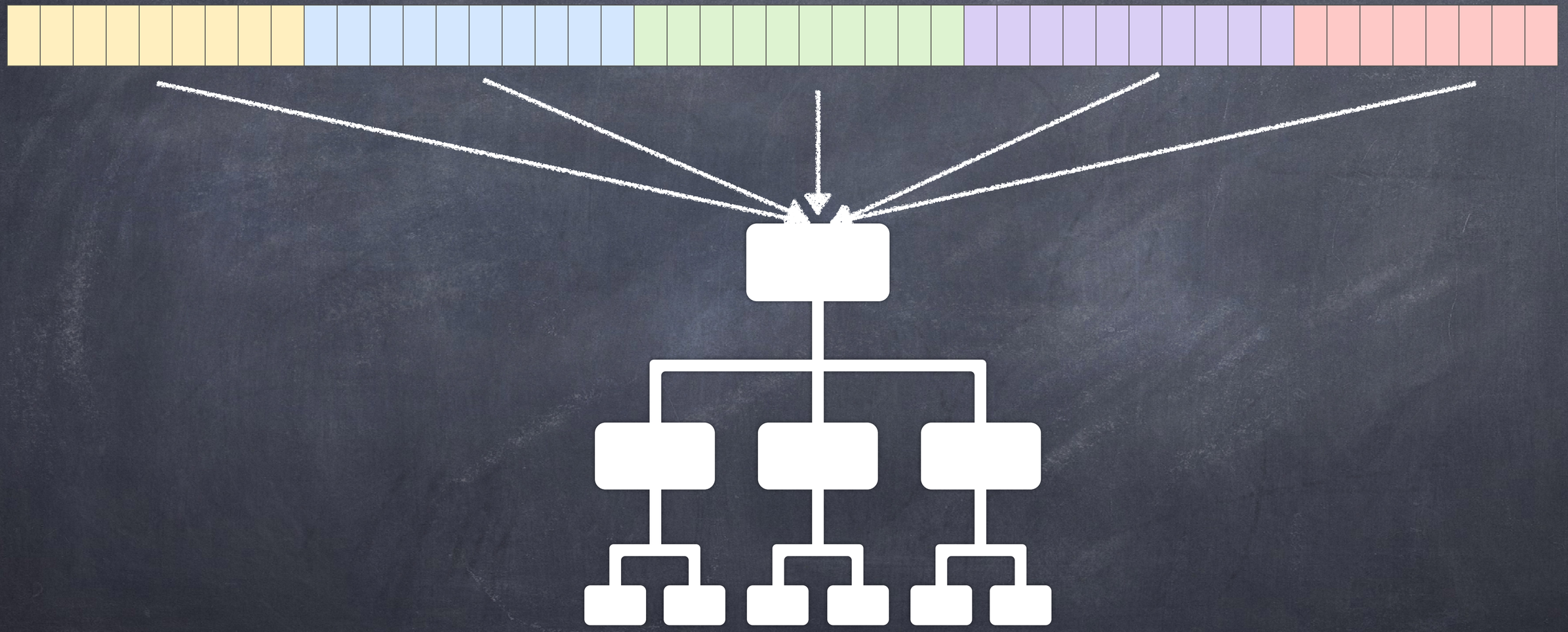
Partition of Mapping and Data

```
while start < num_tree do
|    __demand(__parallel)
|    for t_index = start, start + scaler do
|    |    split_node(tree_partition[t_index], r_data_points, map_partition[t_index], t_index)
|    end
|    start += scaler
|    scaler *= 2
end
```

At each level

Parallel Testing

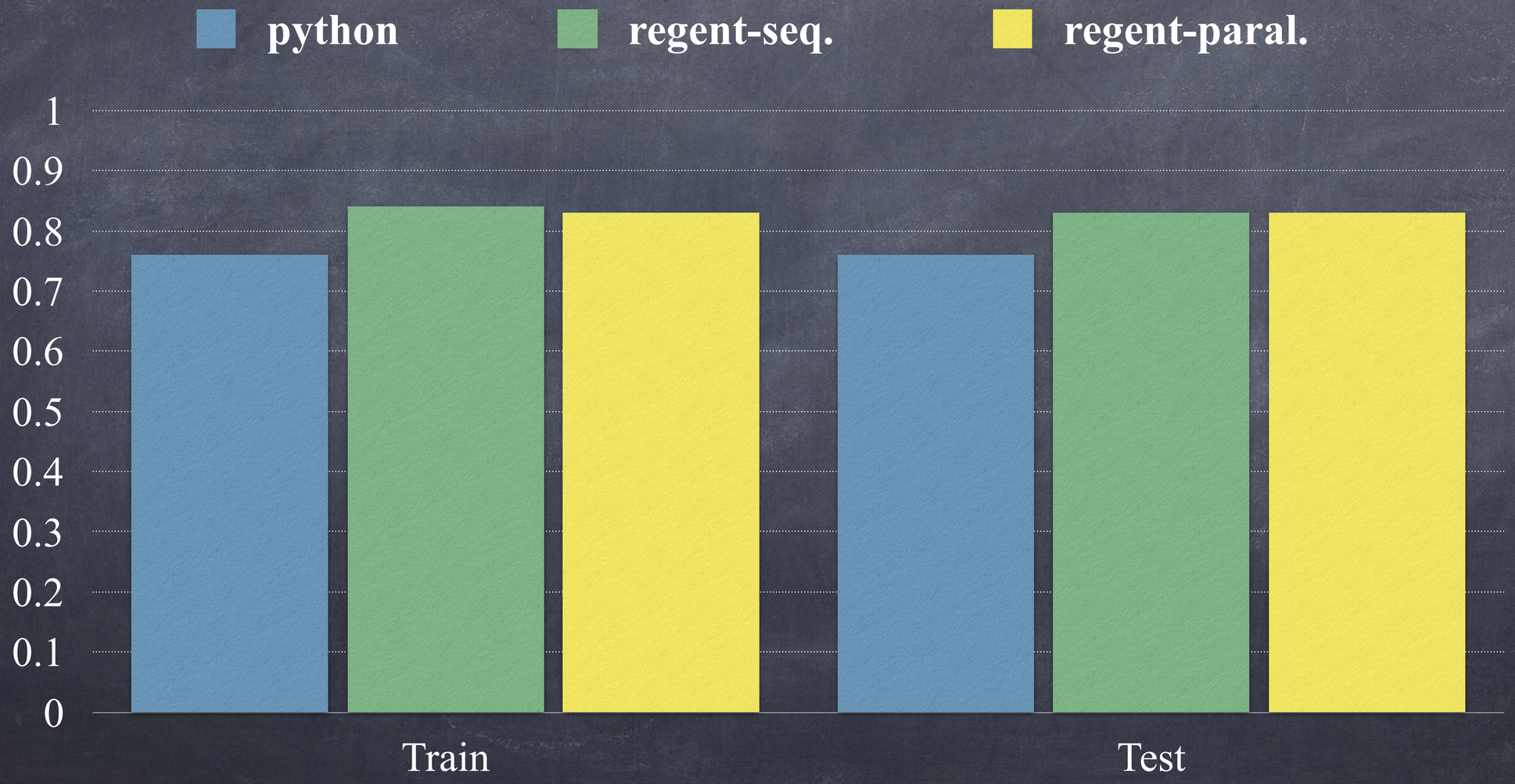
Testing of each data point are independent



Result

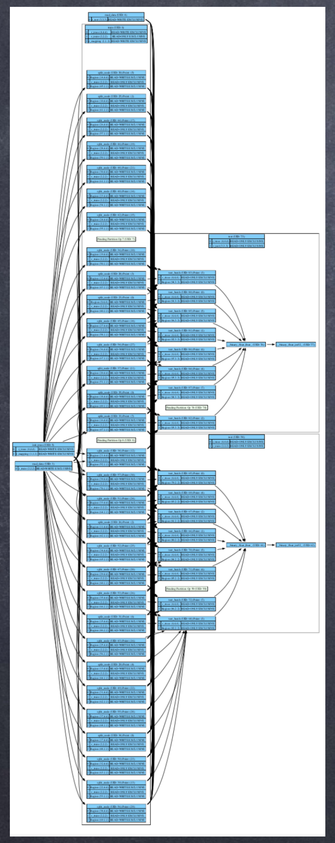
Accuracy Comparison

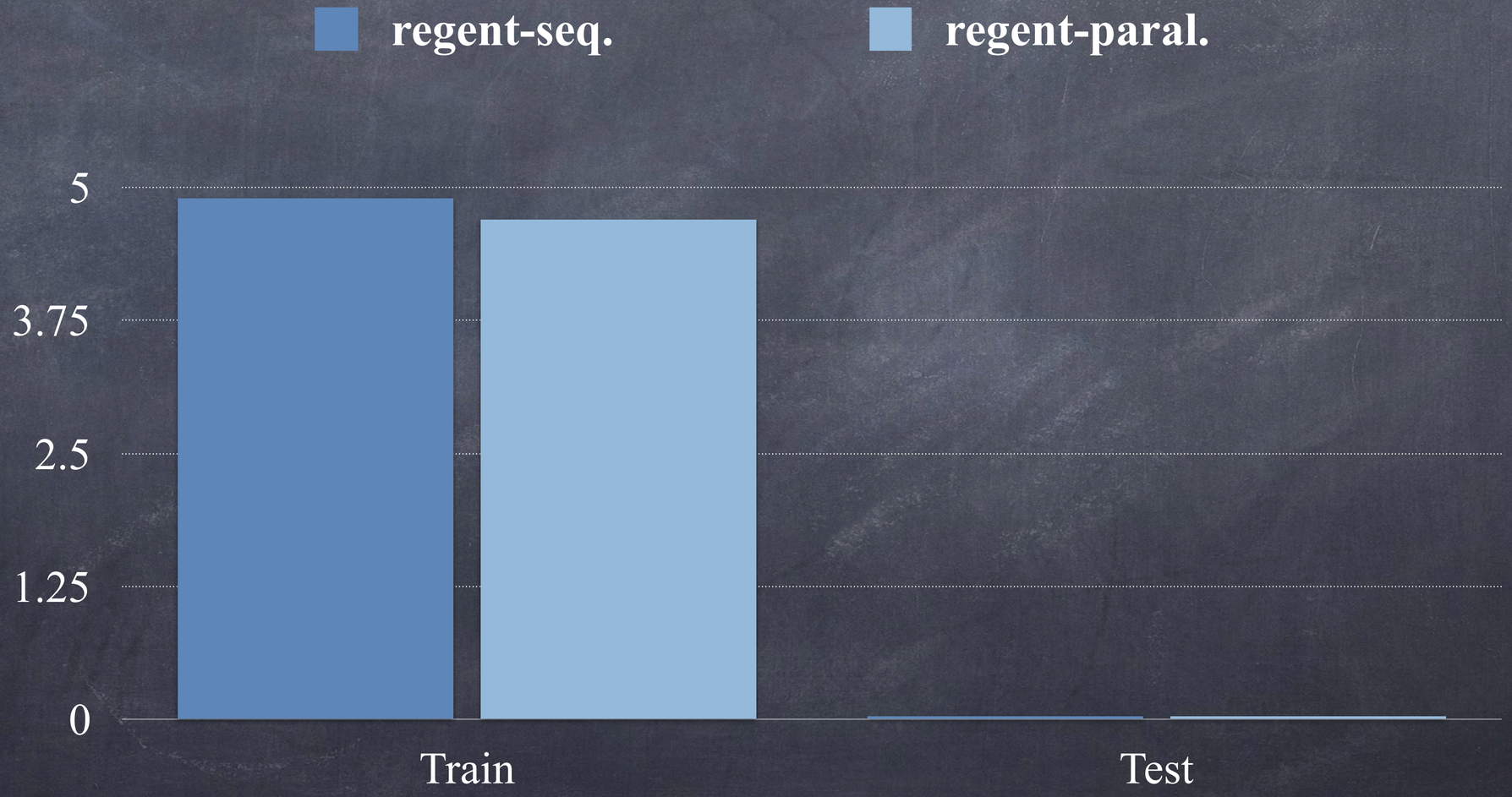
- *Adult Income Dataset*: (30000 Training | 10000 Testing | 7 Features)



Performance Comparison

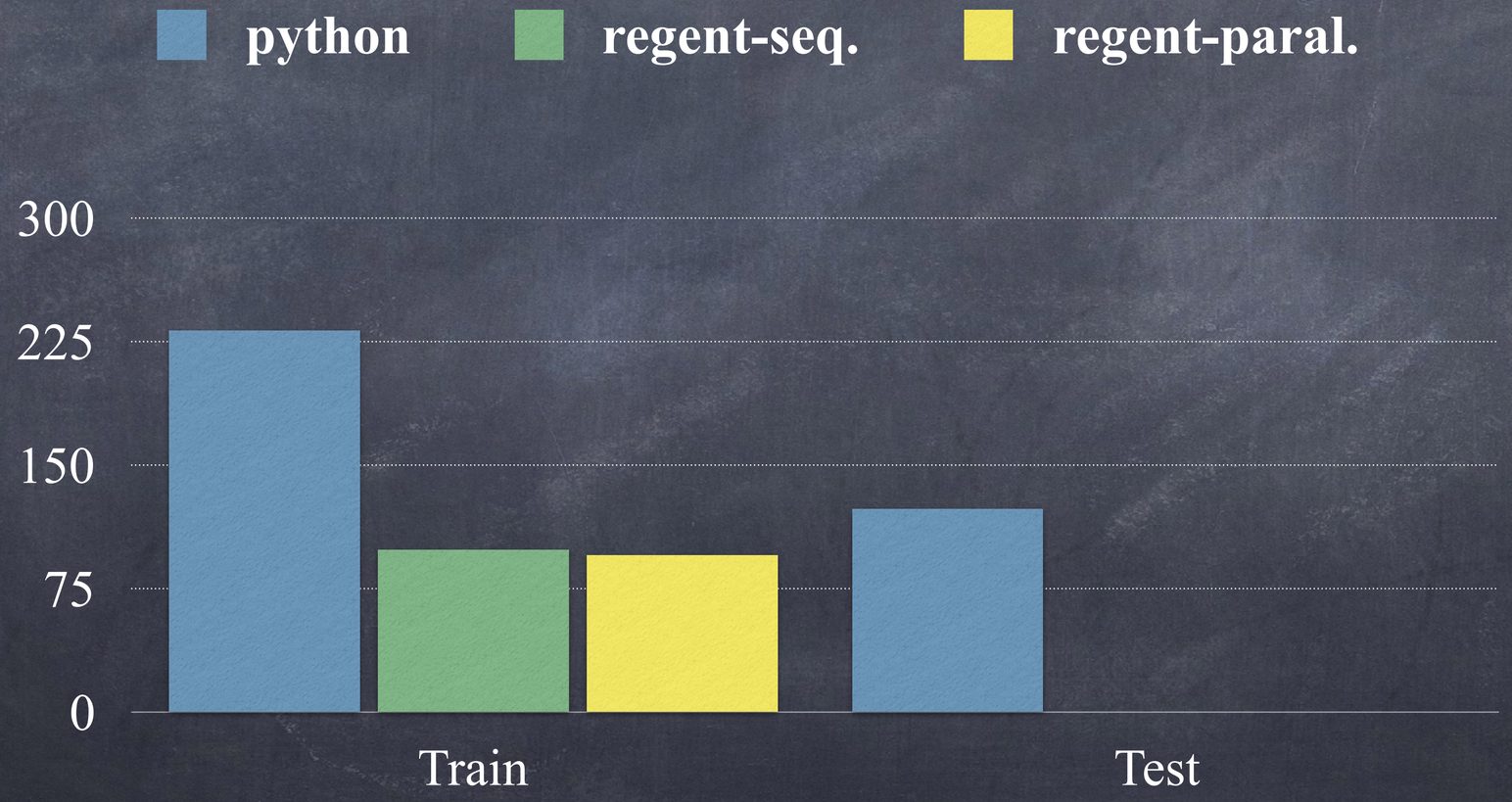
- *Adult Income Dataset*: (7000 Training | 3000 Testing | 7 Features)
- *Max Tree Depth = 6*





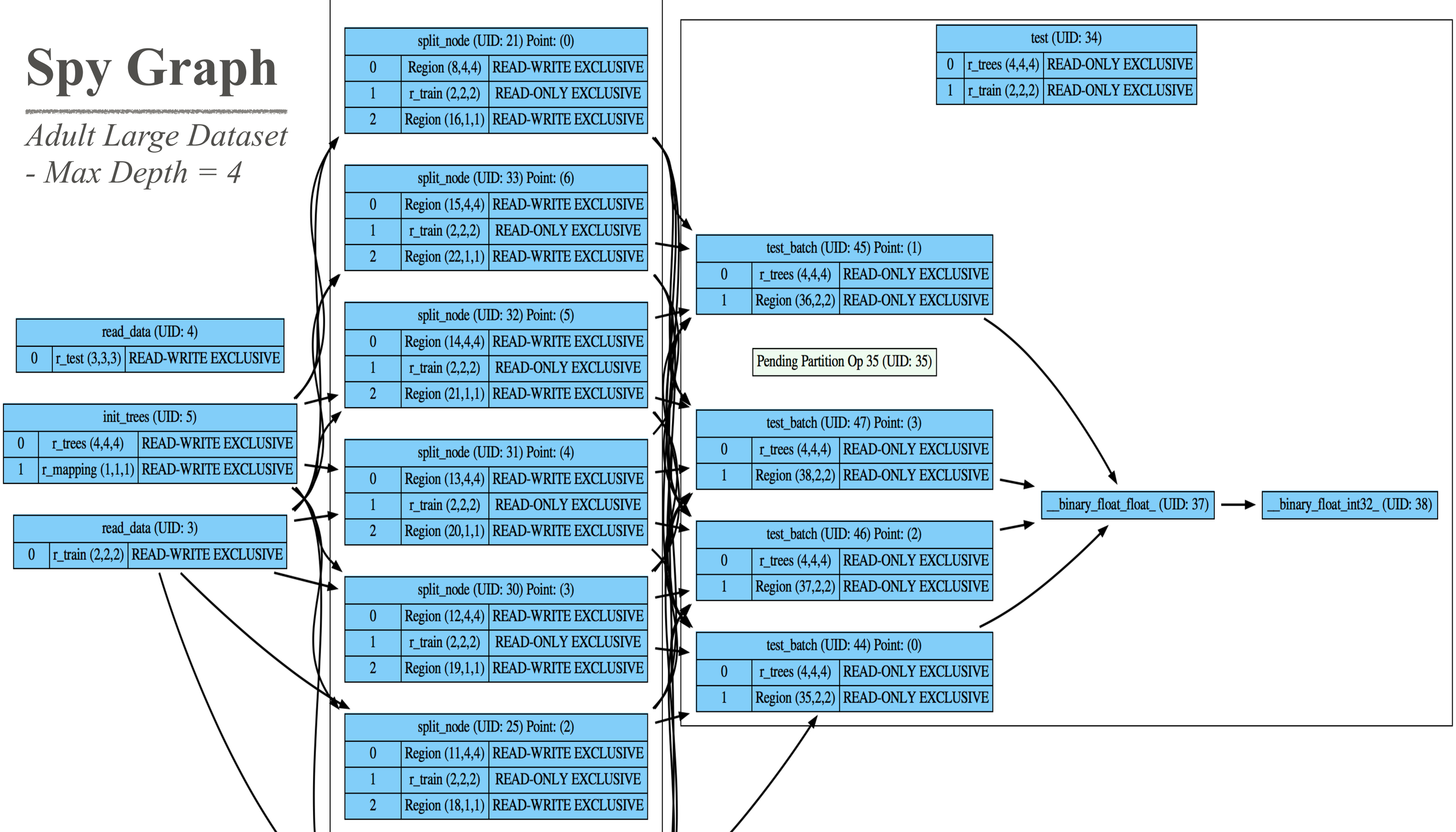
Performance Comparison

- *Adult Income Dataset*: (30000 Training | 10000 Testing | 7 Features)
- *Max Tree Depth = 6*
- From Spy: *Parallel Degrade to sequential on my Mac*



Spy Graph

Adult Large Dataset
- Max Depth = 4



Thanks!