[544] Spark MLlib 2

Tyler Caraza-Harter

Distribued ML Outline

Background: Tree-based Machine Learning Methods

Training Trees:

- in memory
- PLANET algorithm

Model Deployment

- UDF
- Streaming
- Server

Demos

Decision Trees

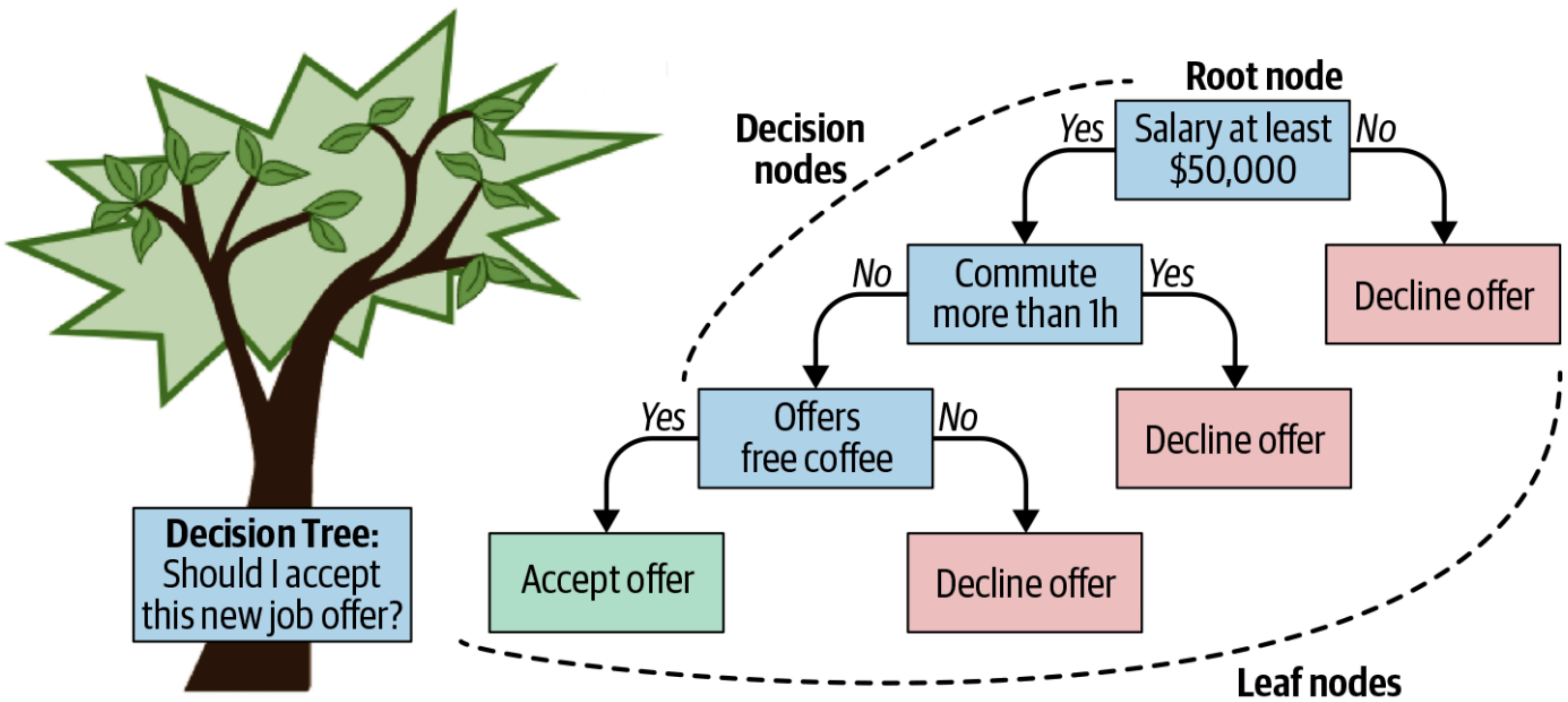
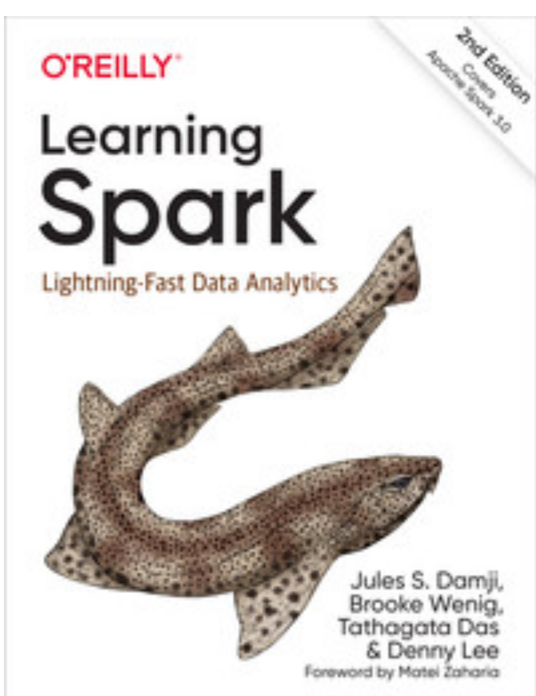


Figure 10-9. Decision tree example

decision trees are like nested if/else statements

features and labels can be numeric or categorical




```
def predict(row):
  if row.salary < 50K:
    return False
  else:
    if row.commute > 1h:
      return False
    else:
      if row.coffee == "free":
        return True
      else:
        return False
```

Ensemble Methods

Ensemble: many simple models vote. Many simple decision trees (each trained on subset of rows/columns) together are often better than one big tree. Examples:

- random forest
- gradient-boosted trees

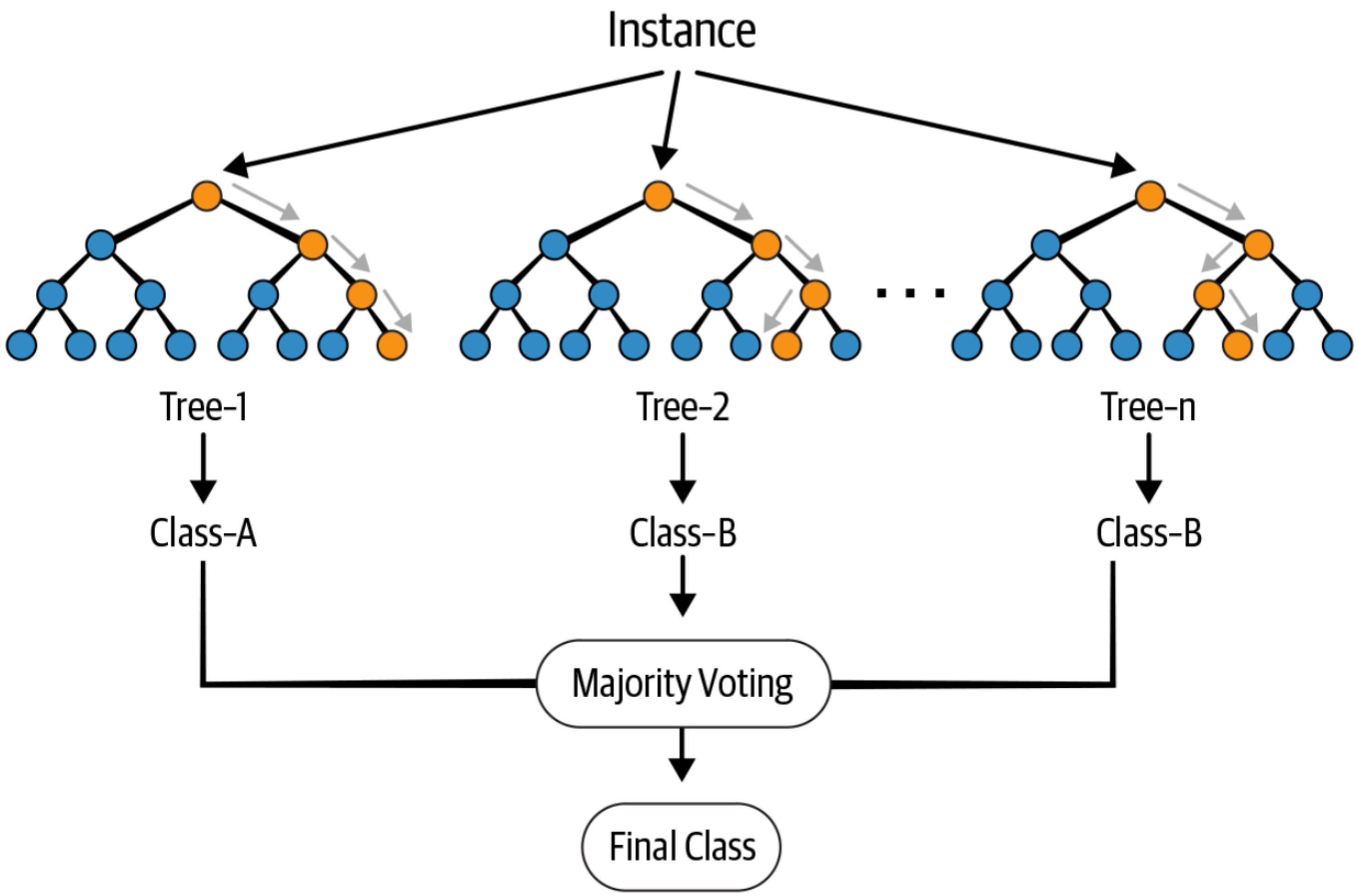
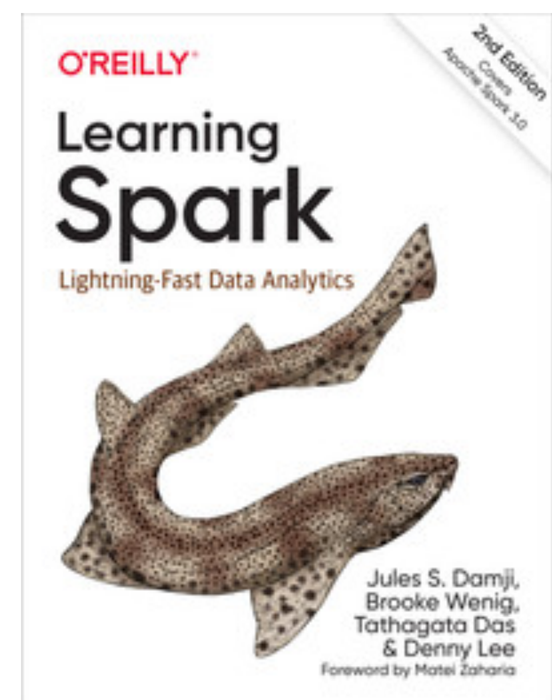
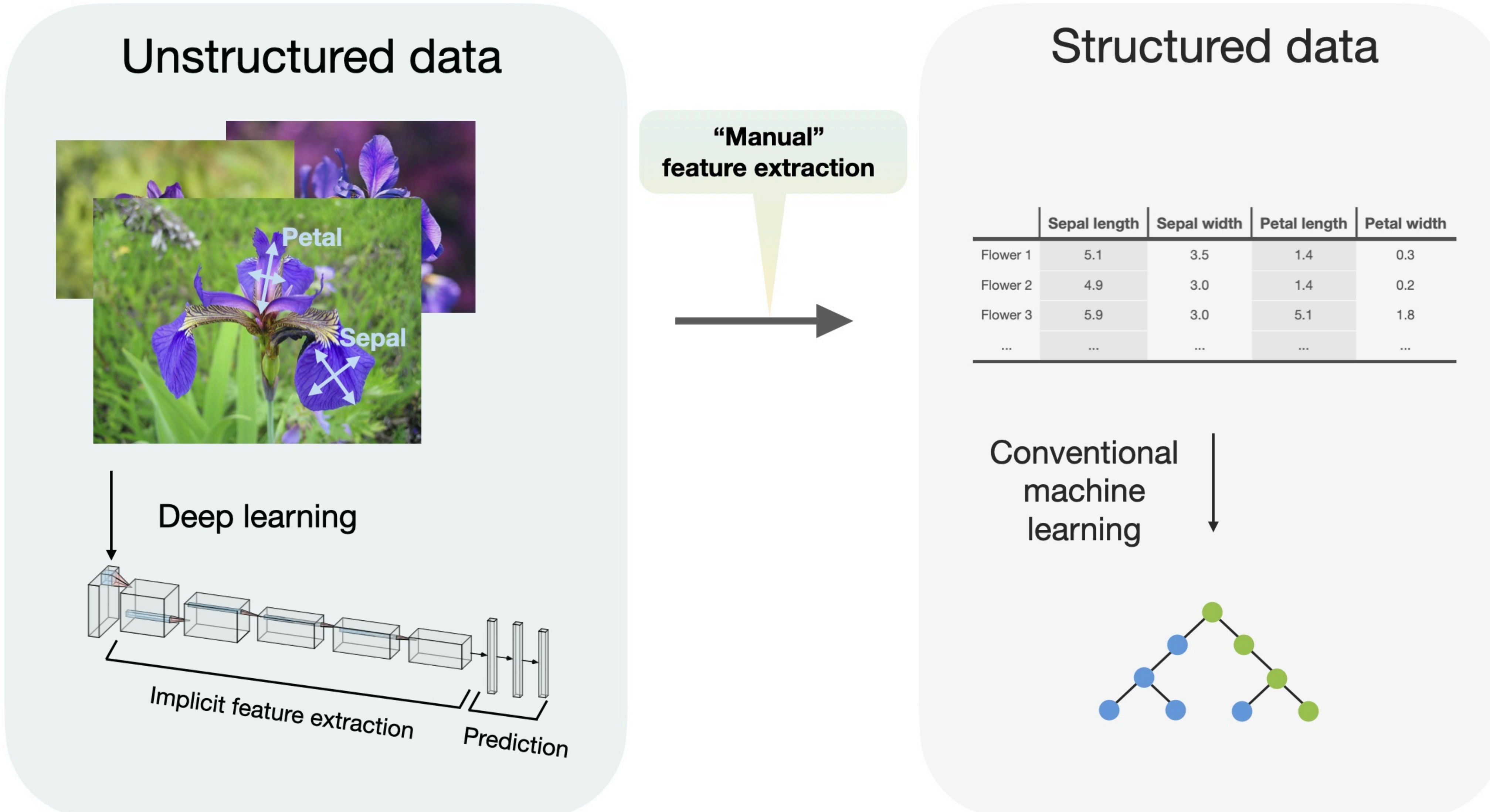


Figure 10-12. Random forest predictions



A Spark cluster can train many trees in a random forest simultaneously!

Tree methods vs. Deep Learning



	Sepal length	Sepal width	Petal length	Petal width
Flower 1	5.1	3.5	1.4	0.3
Flower 2	4.9	3.0	1.4	0.2
Flower 3	5.9	3.0	5.1	1.8
...	...	...	...	...

Image from Blog Post: A Short Chronology Of Deep Learning For Tabular Data
Sebastian Raschka
https://sebastianraschka.com/blog/2022/deep-learning-for-tabular-data.html

Tree-based methods are still relevant in the age of deep learning because there are many important tabular datasets.

Is a Tree Good?

data:

x1	x2	y
1	-1	5
2	2	8.6
3	3	8.8
4	-2	5.2

$\boxed{x1 > 2.5}$

no

x1	x2	y
1	-1	5
2	2	8.6

yes

x1	x2	y
3	3	8.8
4	-2	5.2

$\boxed{x2 > 0}$

no

x1	x2	y
1	-1	5
4	-2	5.2

yes

x1	x2	y
2	2	8.6
3	3	8.8

which tree asks better questions about x values if we want to predict y?

Impurity

data:

x1	x2	y
1	-1	5
2	2	8.6
3	3	8.8
4	-2	5.2

$x1 > 2.5$

no

x1	x2	y
1	-1	5
2	2	8.6

high variance

yes

x1	x2	y
3	3	8.8
4	-2	5.2

high variance

better tree

$x2 > 0$

no

x1	x2	y
1	-1	5
4	-2	5.2

low variance

yes

x1	x2	y
2	2	8.6
3	3	8.8

low variance

impurity measures (like variance) measure how non-uniform label (y) values are in leaves

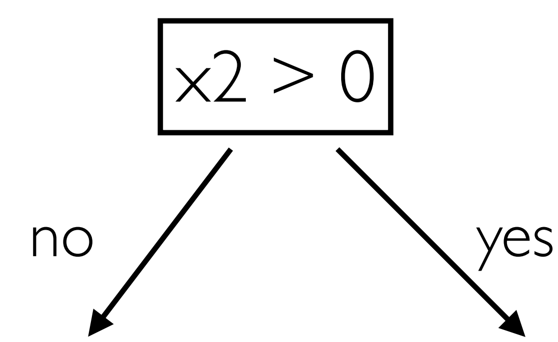
Predictions

data:

x1	x2	y
1	-1	5
2	2	8.6
3	3	8.8
4	-2	5.2
3	50	????

if a new data point lands in a leaf, assume it is similar to other rows in that leaf...

better tree



x1	x2	y
1	-1	5
4	-2	5.2

prediction: 5.1

x1	x2	y
2	2	8.6
3	3	8.8

prediction: 8.7

Distribued ML Outline

Background: Tree-based Machine Learning Methods

Training Trees:
- in memory
- PLANET algorithm

Model Deployment
- UDF
- Streaming
- Server

Demos

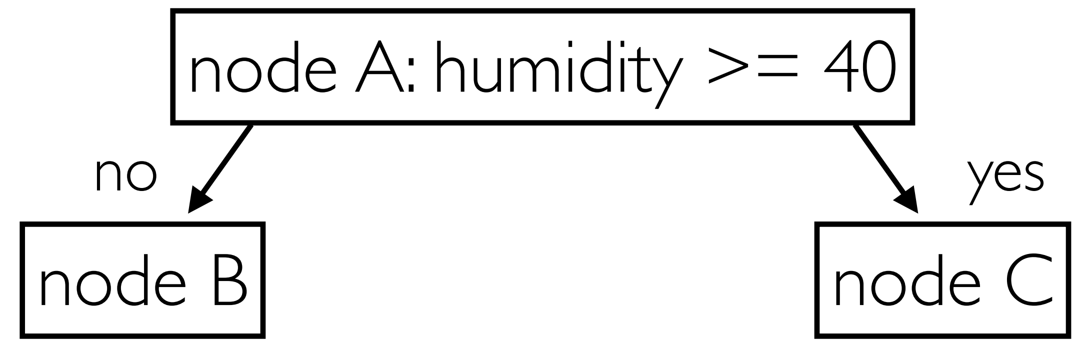
Splitting Nodes

node A

temp	humidity	rain
80	45	no
50	40	yes
45	20	no
70	25	no

probability: 25%

split



node B

temp	humidity	rain
45	20	no
70	25	no

probability: 0%

node C

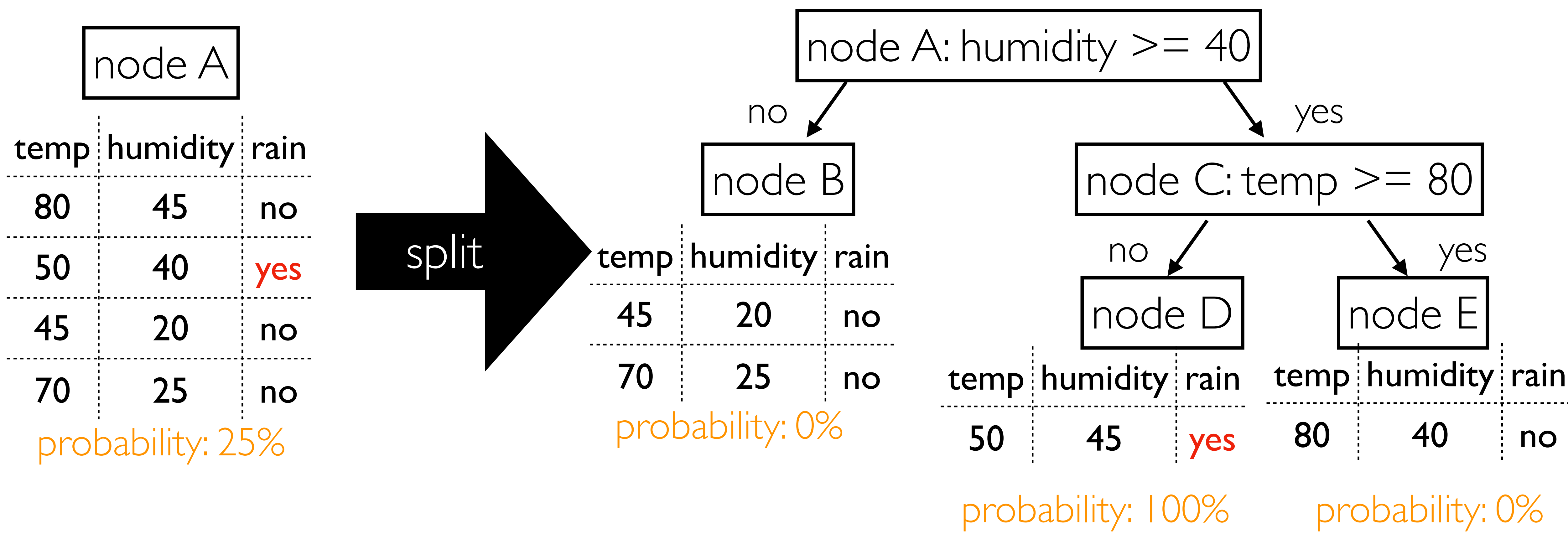
temp	humidity	rain
80	40	no
50	45	yes

probability: 50%

Algorithm

- start with one node with all data
- find split point in some column to create two children
- identify another node, recursively split
- eventually stop

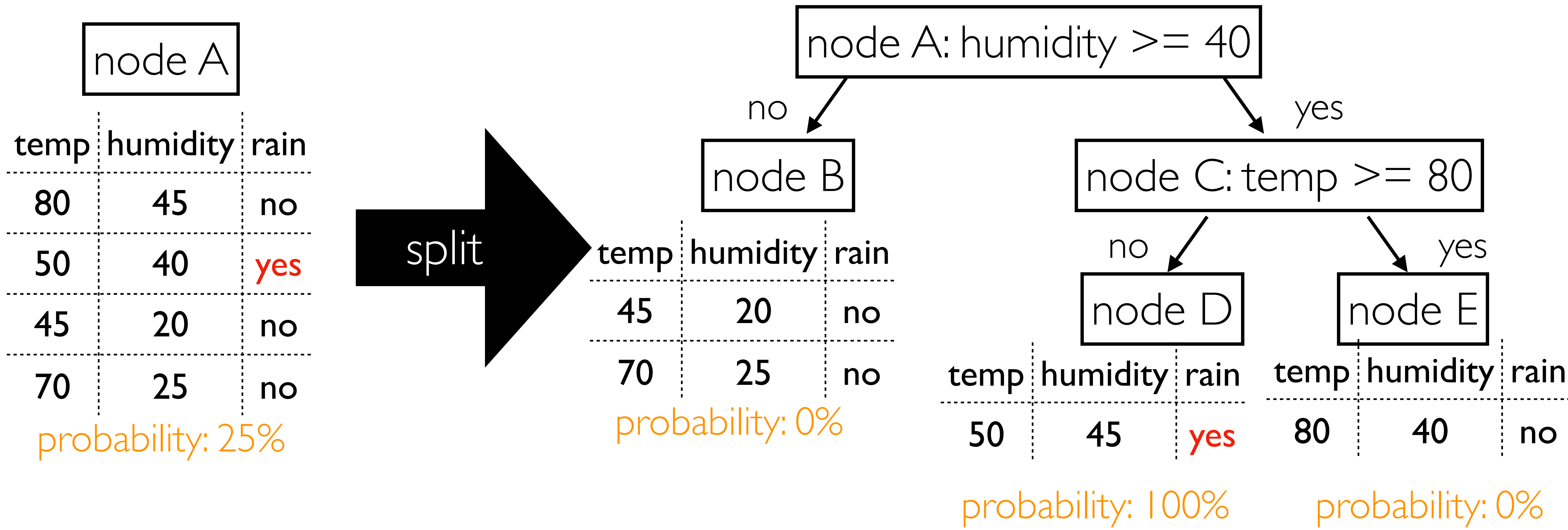
Splitting Nodes



Algorithm

- start with one node with all data
- find split point in some column to create two children
- identify another node, recursively split
- eventually stop

When to Stop Splitting?



Some Approaches

- set maximum tree height
- set minimum number of rows in node required for split
- prune tree later to get rid of unhelpful/excessive splitting

Choosing Splits

node A

temp	humidity	rain
80	45	no
50	40	yes
45	20	no
70	25	no

probability: 25%

split

Which node to split?

- 2 feature columns
- 3 ways to divide 4 rows into big small
- 2*3 = 6 choices
- try all, choose one that reduces impurity the most!
- how to do so efficiently?

Choosing Splits

node A

temp	humidity	rain
80	45	no
50	40	yes
45	20	no
70	25	no

sort by each column

temp	humidity	rain
45	20	no
50	40	yes
70	25	no
80	45	no

calculate impurity for each split (one pass)

left no	left yes	right no	right yes	impurity
1	0	2	1	...
1	1	2	0	...
2	1	1	0	...

Observation: we can incrementally compute impurity for each split point by looking at just more row of data. Don't need to loop over all rows for every possible split point.

Choosing Splits

node A

temp	humidity	rain
80	45	no
50	40	yes
45	20	no
70	25	no

sort by each column

temp	humidity	rain
45	20	no
50	40	yes
70	25	no
80	45	no

temp	humidity	rain
45	20	no
70	25	no
50	40	yes
80	45	no

calculate impurity for each split (one pass)

left no	left yes	right no	right yes	impurity
1	0	2	1	...
1	1	2	0	...
2	1	1	0	...

left no	left yes	right no	right yes	impurity
1	0	2	1	...
2	0	1	1	...
2	1	1	0	...

choose best

Challenge: Big Data

node A

temp	humidity	rain
80	45	no
50	40	yes
45	20	no
70	25	no

sort by each column

temp	humidity	rain
45	20	no
50	40	yes
70	25	no
80	45	no

...

What if rows for a node are too big to fit in RAM on one worker?

- partitioned across many Spark workers
- maybe fits in cumulative RAM of many workers (maybe not)
- each sort would be expensive (network shuffle/exchange)
- as looping over every possible split point, we'll be computing on one worker at any given time (the one that has data around the split point). Not parallel!

Distribued ML Outline

Background: Tree-based Machine Learning Methods

Training Trees:

- in memory
- PLANET algorithm

Model Deployment

- UDF
- Streaming
- Server

Demos

PLANET Algorithm

PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo
Google, Inc.
[bpanda, jsherbach, sugato]@google.com, bayardo@alum.mit.edu

ABSTRACT

Classification and regression tree learning on massive datasets is a common data mining task at Google, yet many state of the art tree learning algorithms require training data to

plexities such as data partitioning, scheduling tasks across many machines, handling machine failures, and performing inter-machine communication. These properties have motivated many technology companies to run MapReduce

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36296.pdf

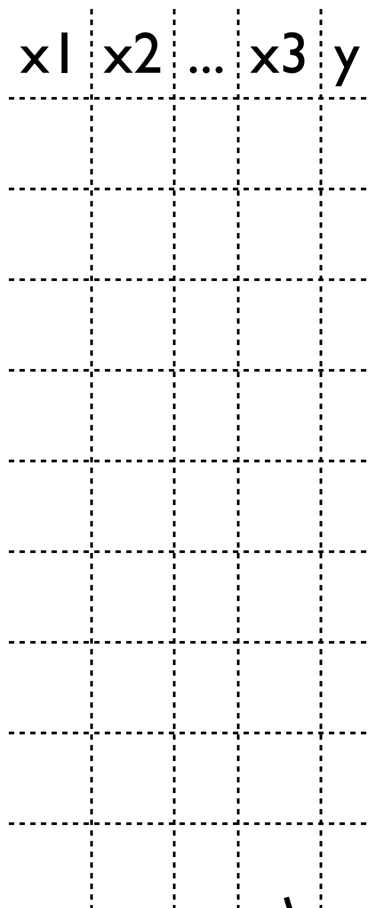
PLANET: Parallel Learner for Assembling Numerous Ensemble Trees

- originally implemented as MapReduce jobs
- Spark DecisionTreeRegressor and DecisionTreeClassifier use it too

Hybrid Approach

- in-memory splitting for nodes with few enough rows to fit in worker memory
- simplified distributed approach for nodes with lots of data

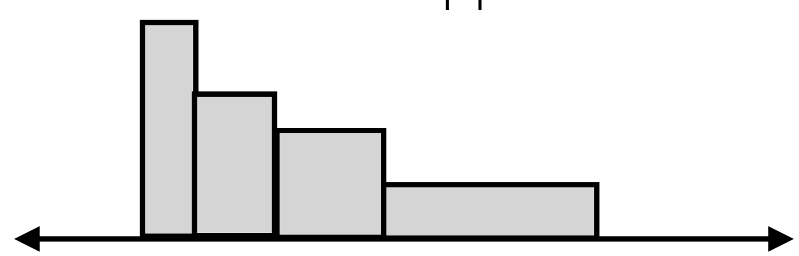
Step 1: Compute Equi-Depth Histograms



Warning: number of bins must be greater than biggest number of unique values in any categorical column

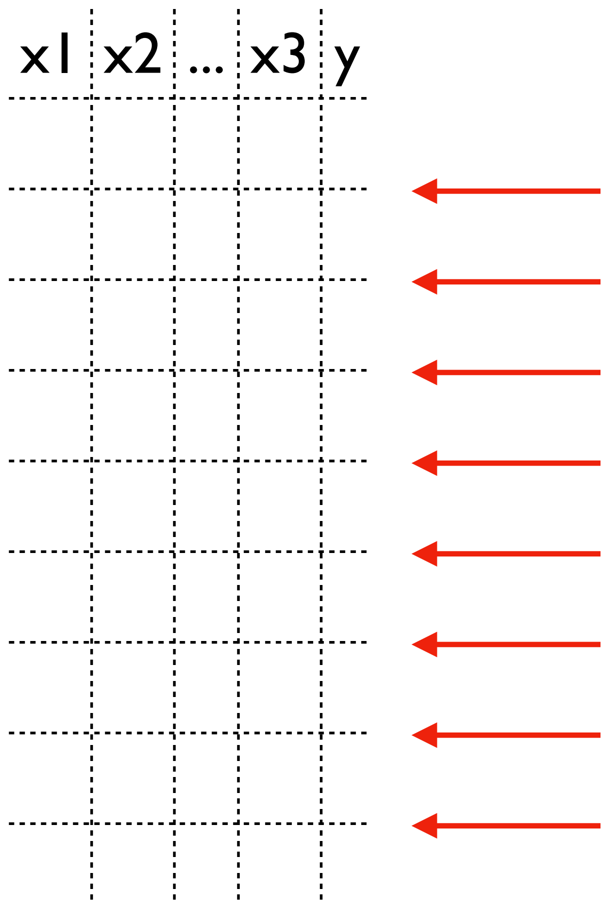
```
dt = DecisionTreeClassifier(labelCol="y")
dt.setMaxBins(4)
```

equi-depth histogram for x3 feature
(each bucket has approx same number of samples



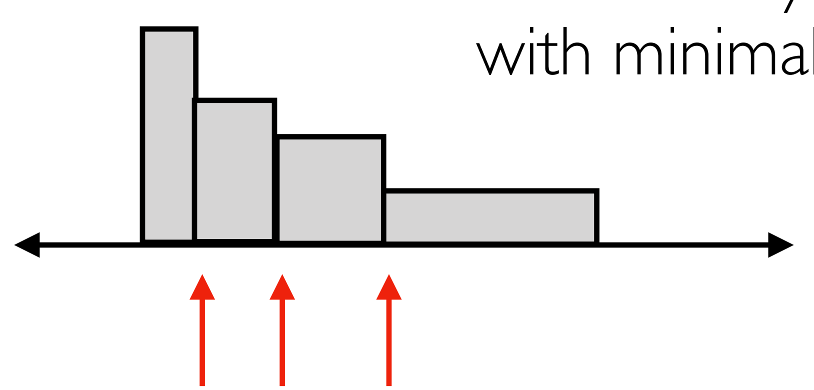
histograms for every feature...

Split Points: In-Mem vs. Distributed



split points for one-node, in-memory algorithm

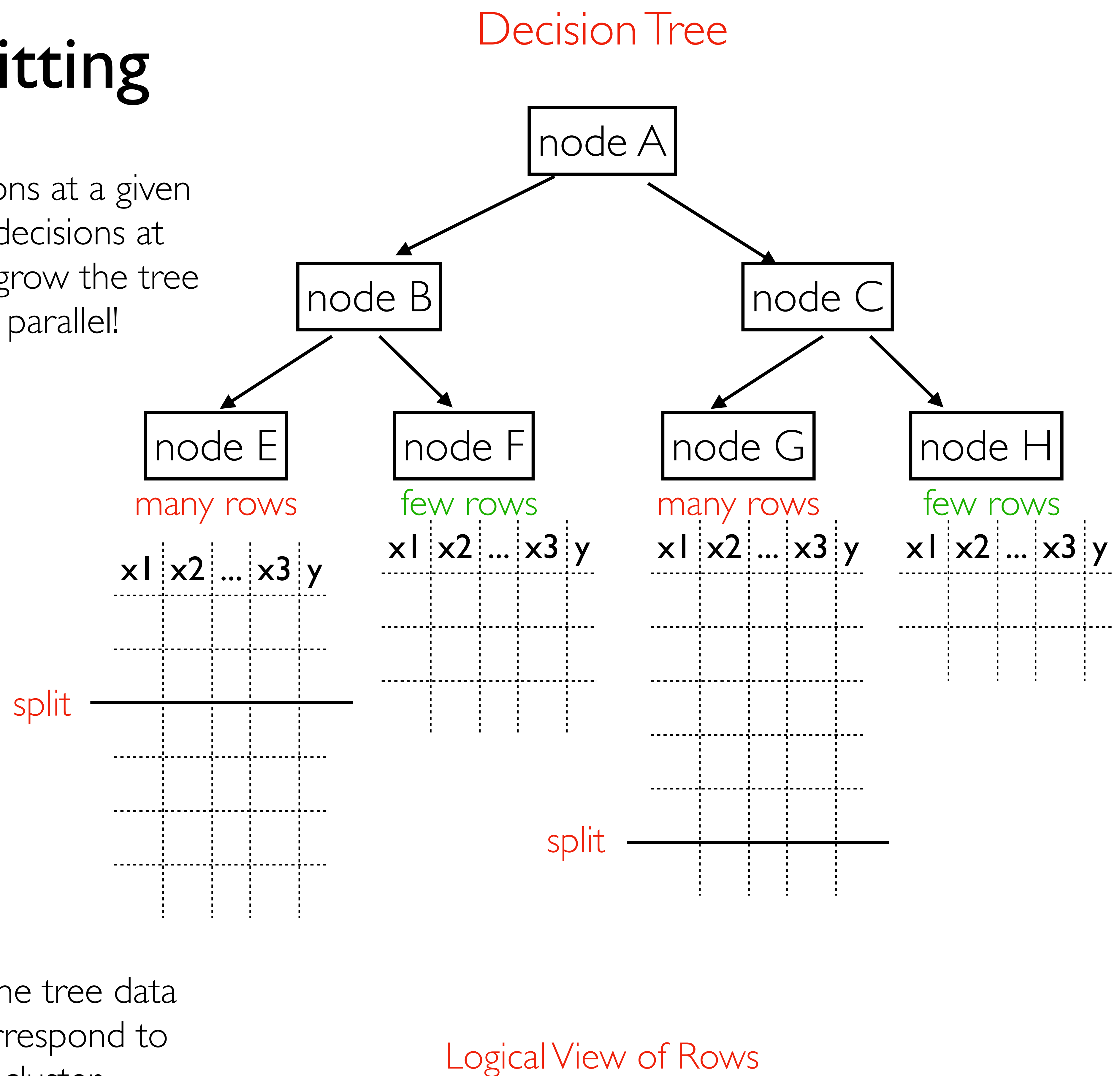
- splits based on initial data
- every task has split info for every column
- small number of pre-defined split points make it easy for executors to collaborate with minimal coordination



split points for distributed algorithm

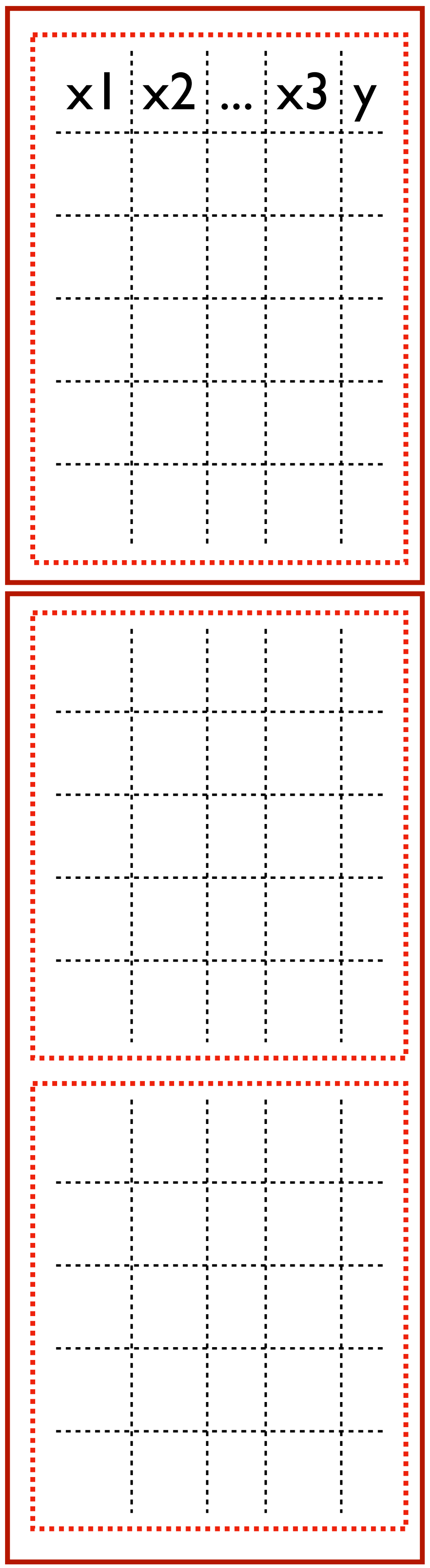
Parallel Splitting

Observation: split decisions at a given node are unrelated to decisions at other nodes. It's easy to grow the tree at different places in parallel!

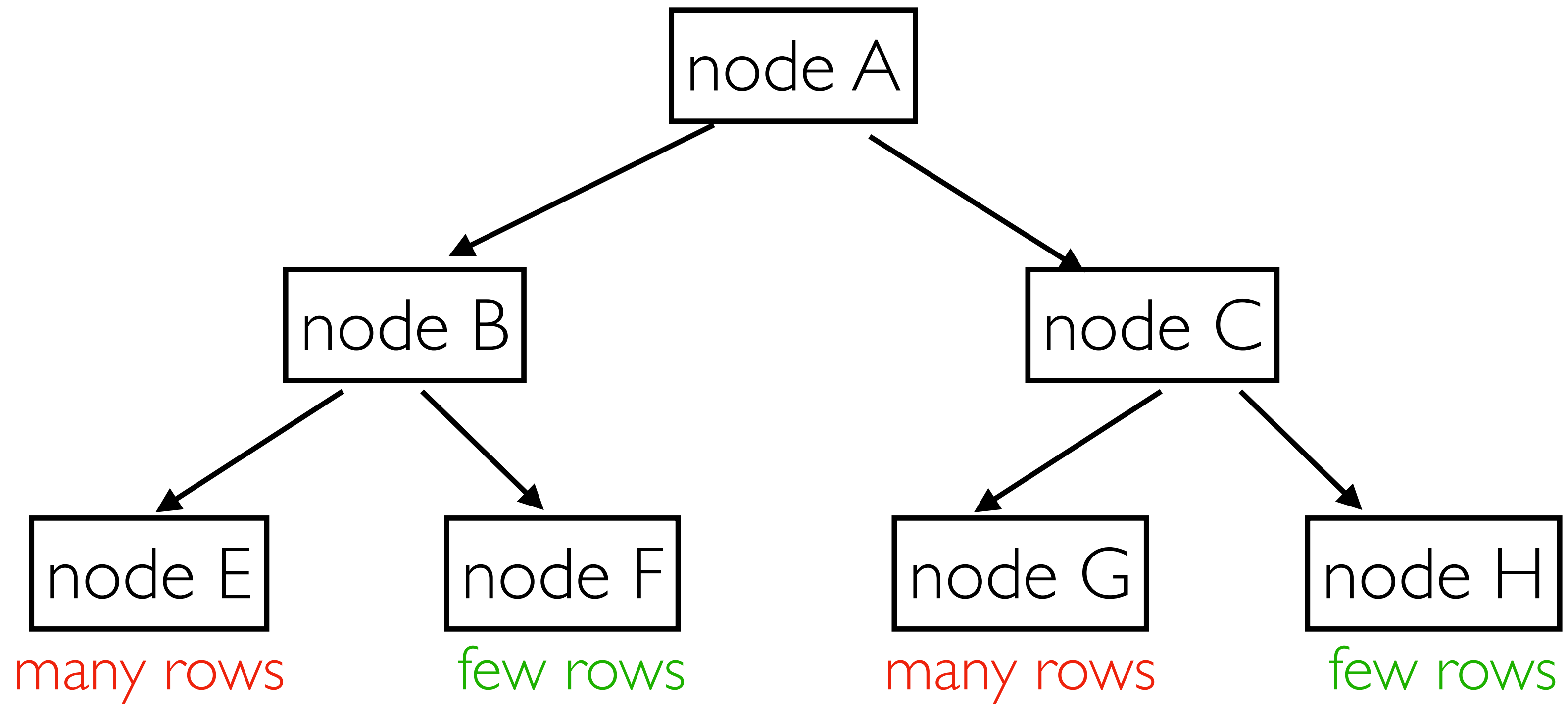


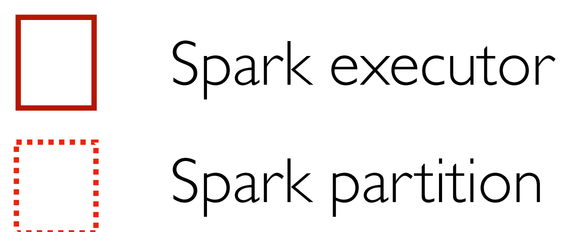
Clarification: nodes in the tree data structure DO NOT correspond to nodes in the Spark cluster.

Physical Layout



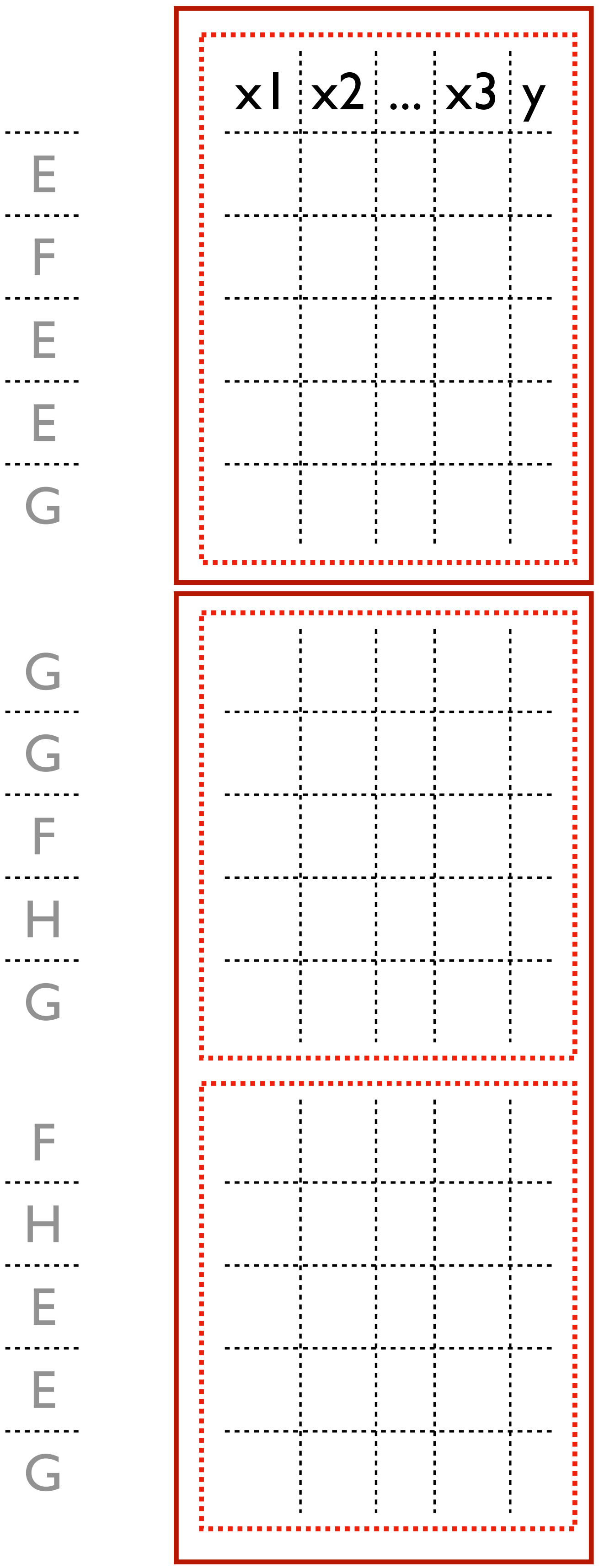
Decision Tree



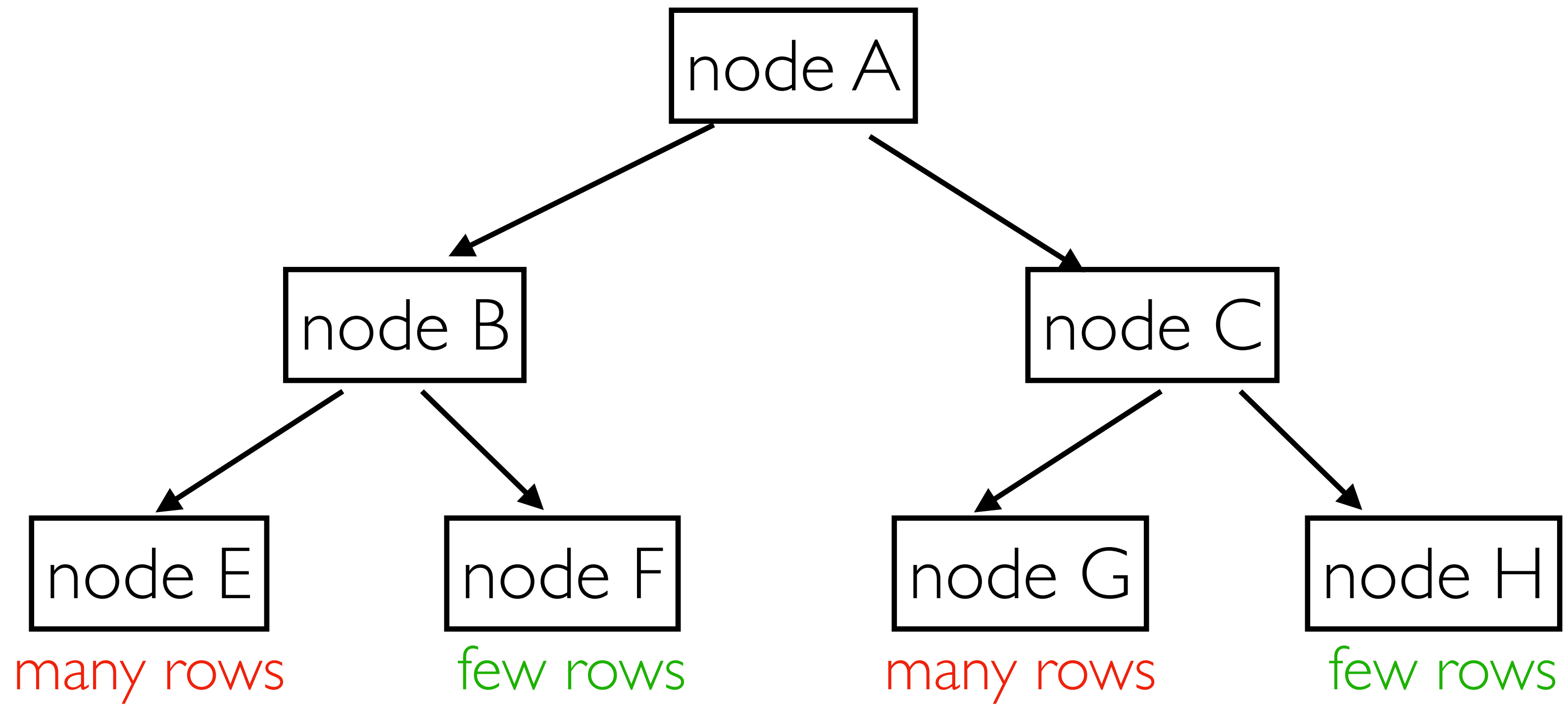


- all rows are in one big Spark DataFrame
- no particular order

Physical Layout



Decision Tree

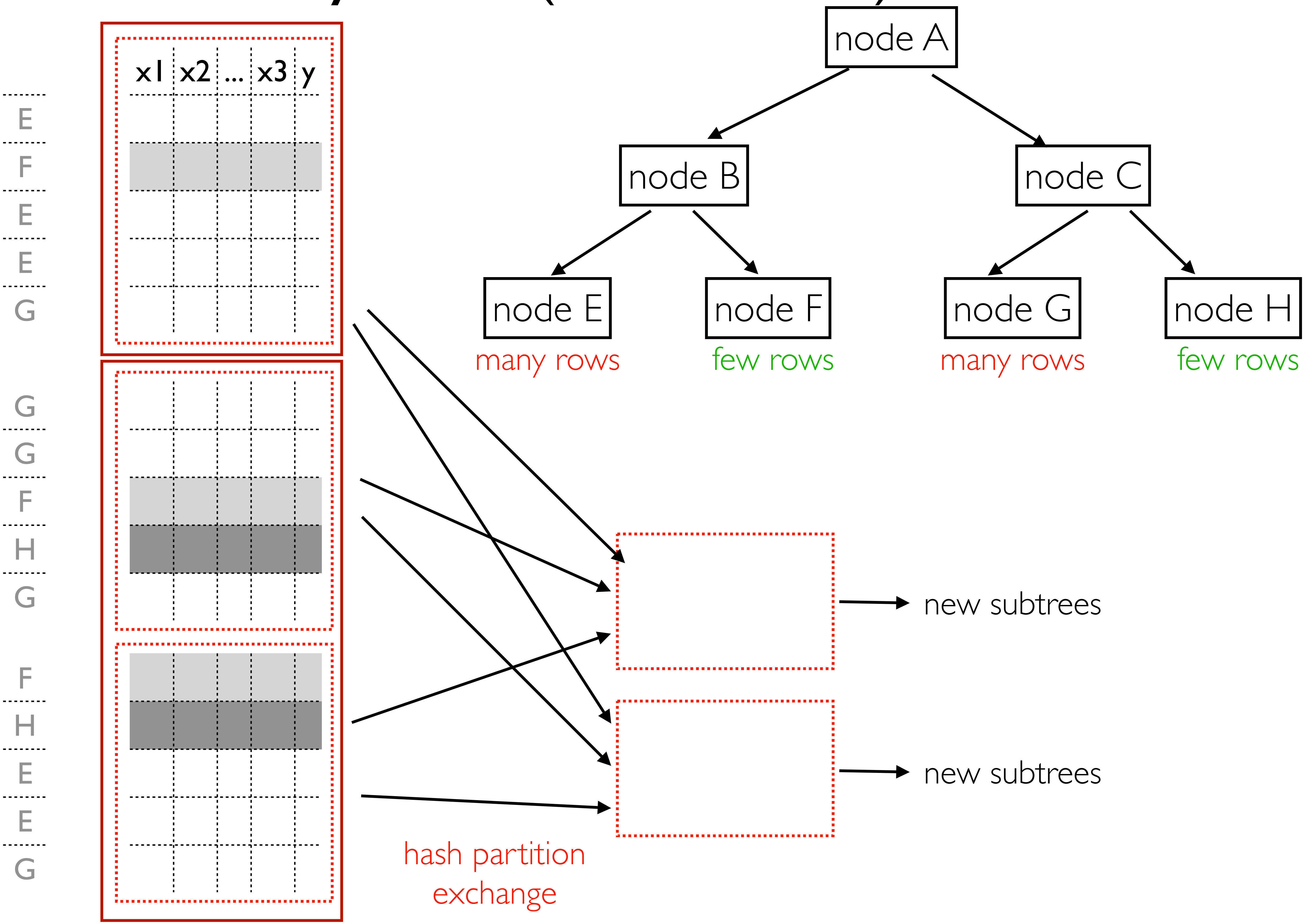


Spark executor

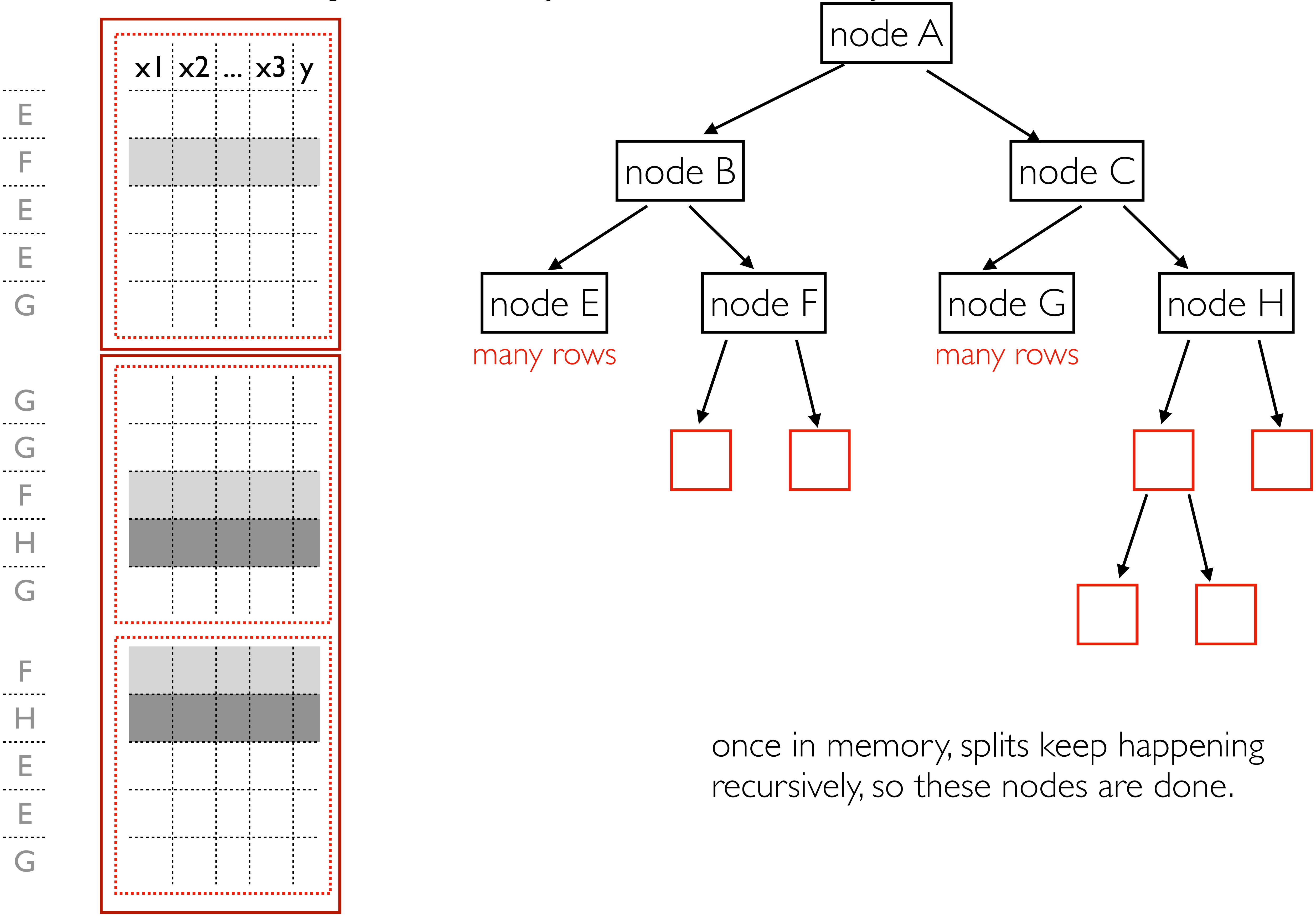
Spark partition

- all rows are in one big Spark DataFrame
- no particular order
- given current tree and x1...xN values, we can infer what leaf node in the tree owns each row

In Memory Build (small nodes)

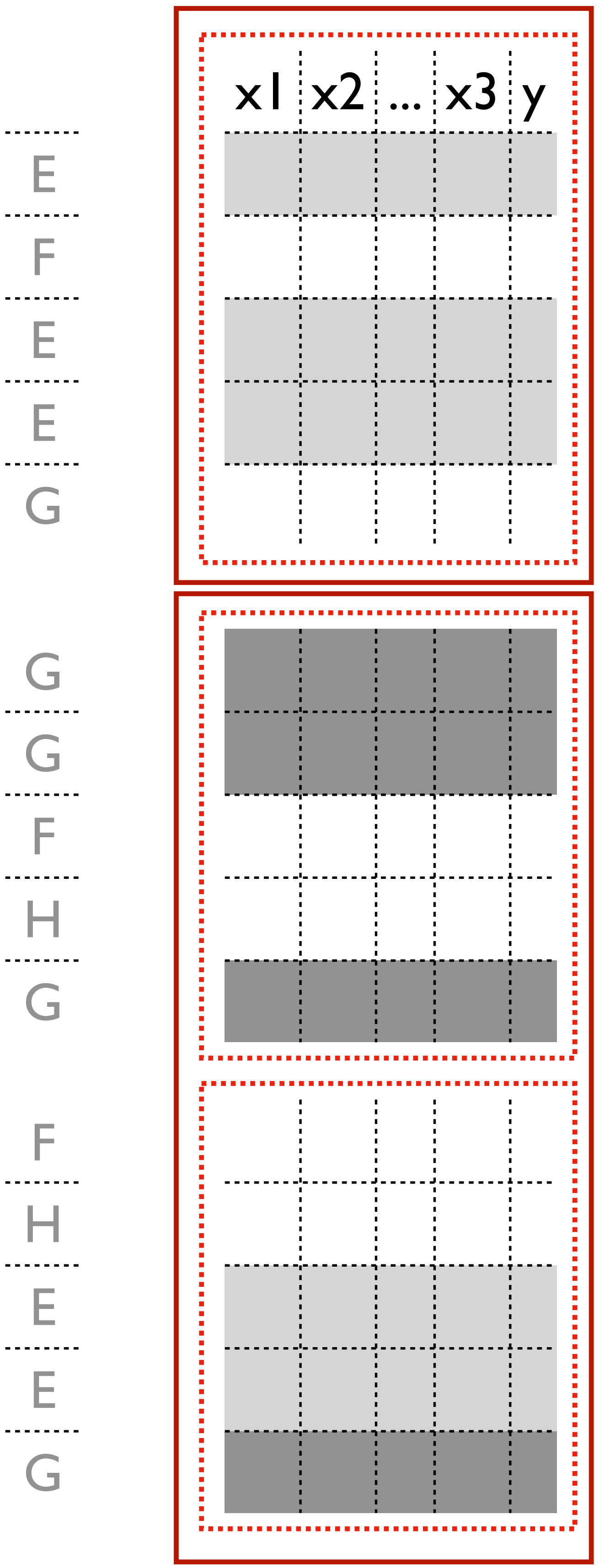


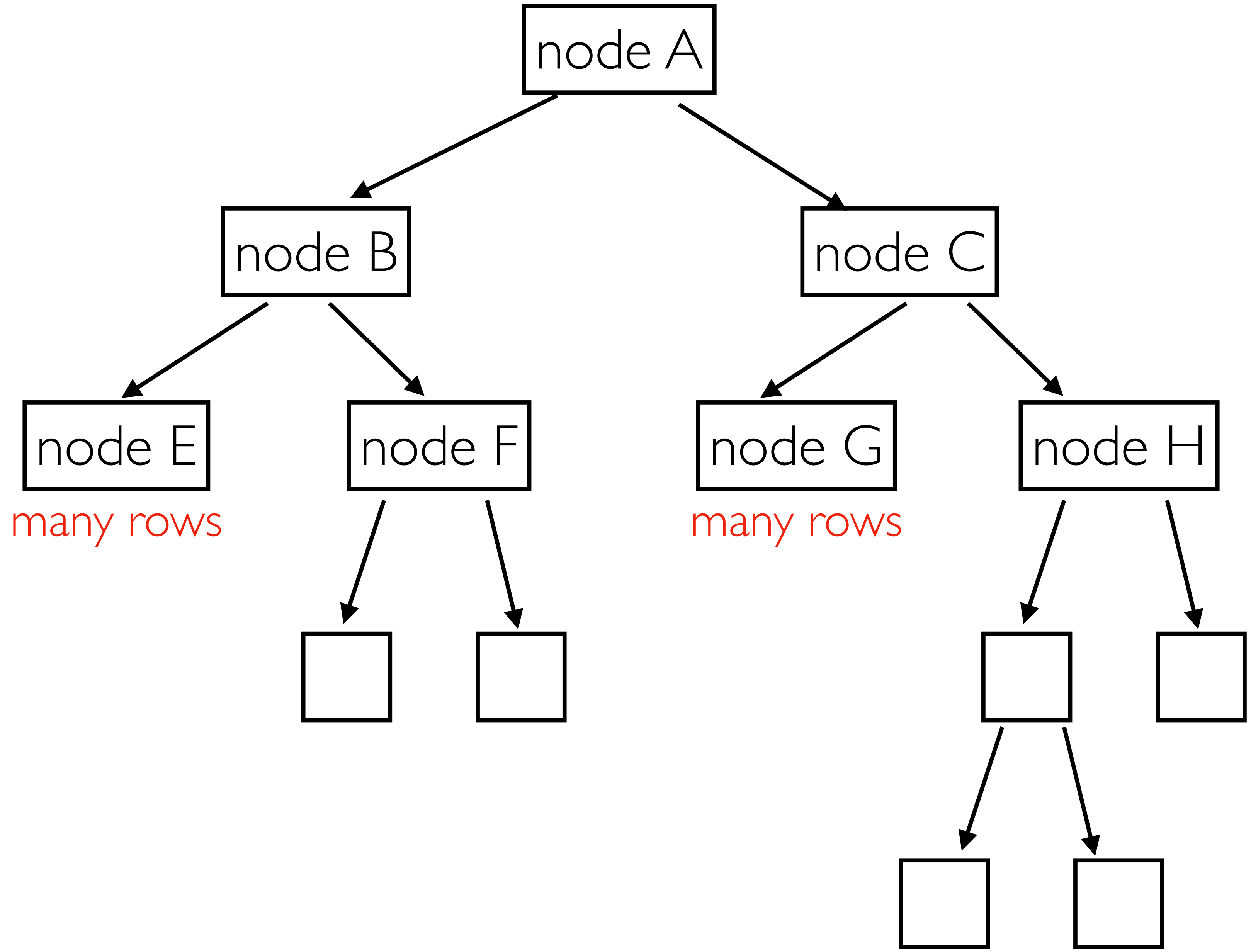
In Memory Build (small nodes)



once in memory, splits keep happening recursively, so these nodes are done.

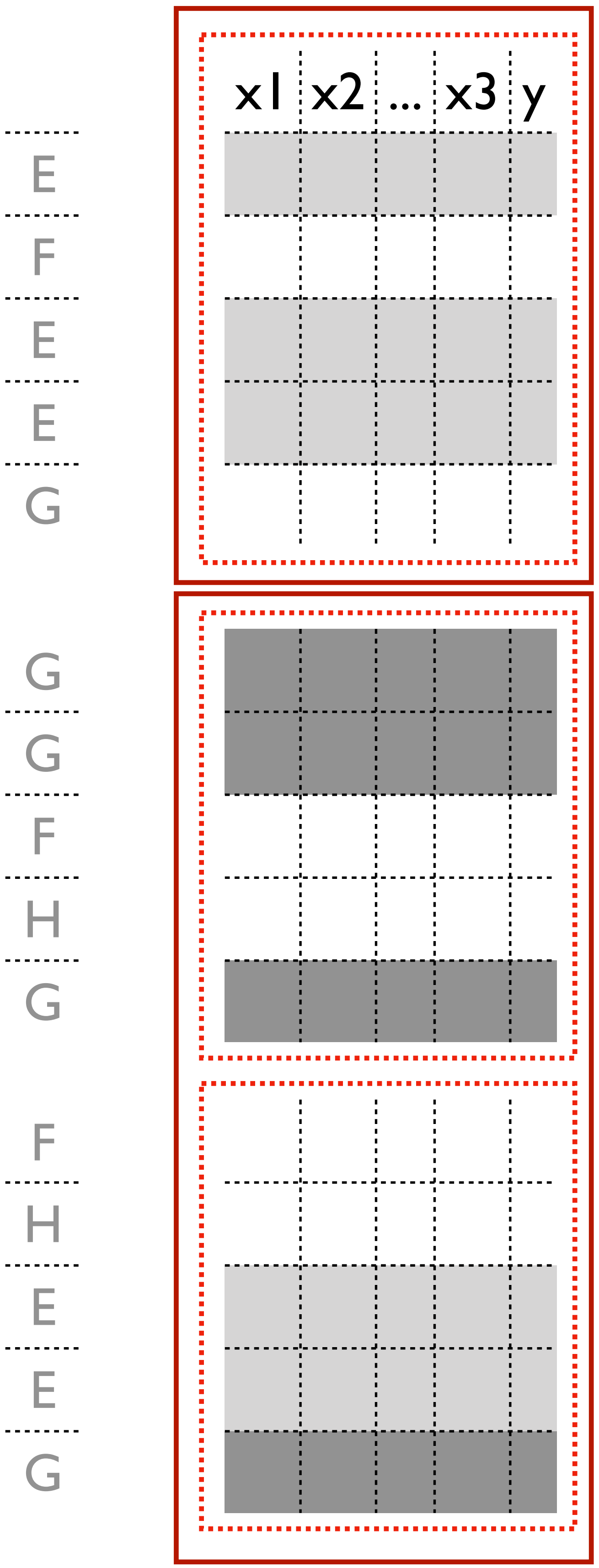
Big Nodes

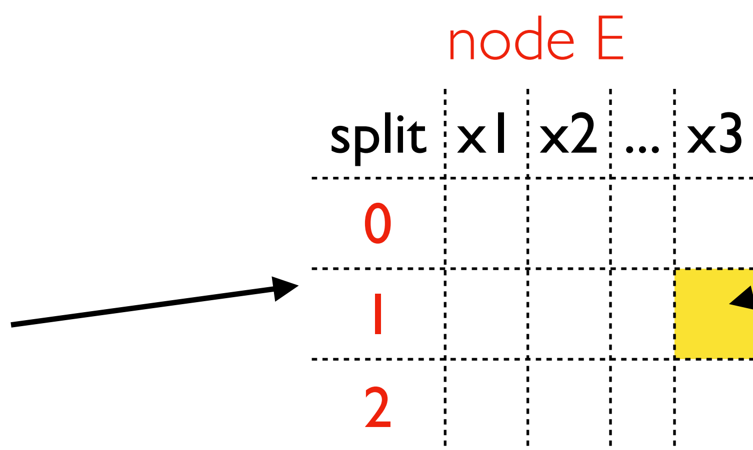




- don't move data!
- just output stats per partition for every split/feature option

Big Nodes



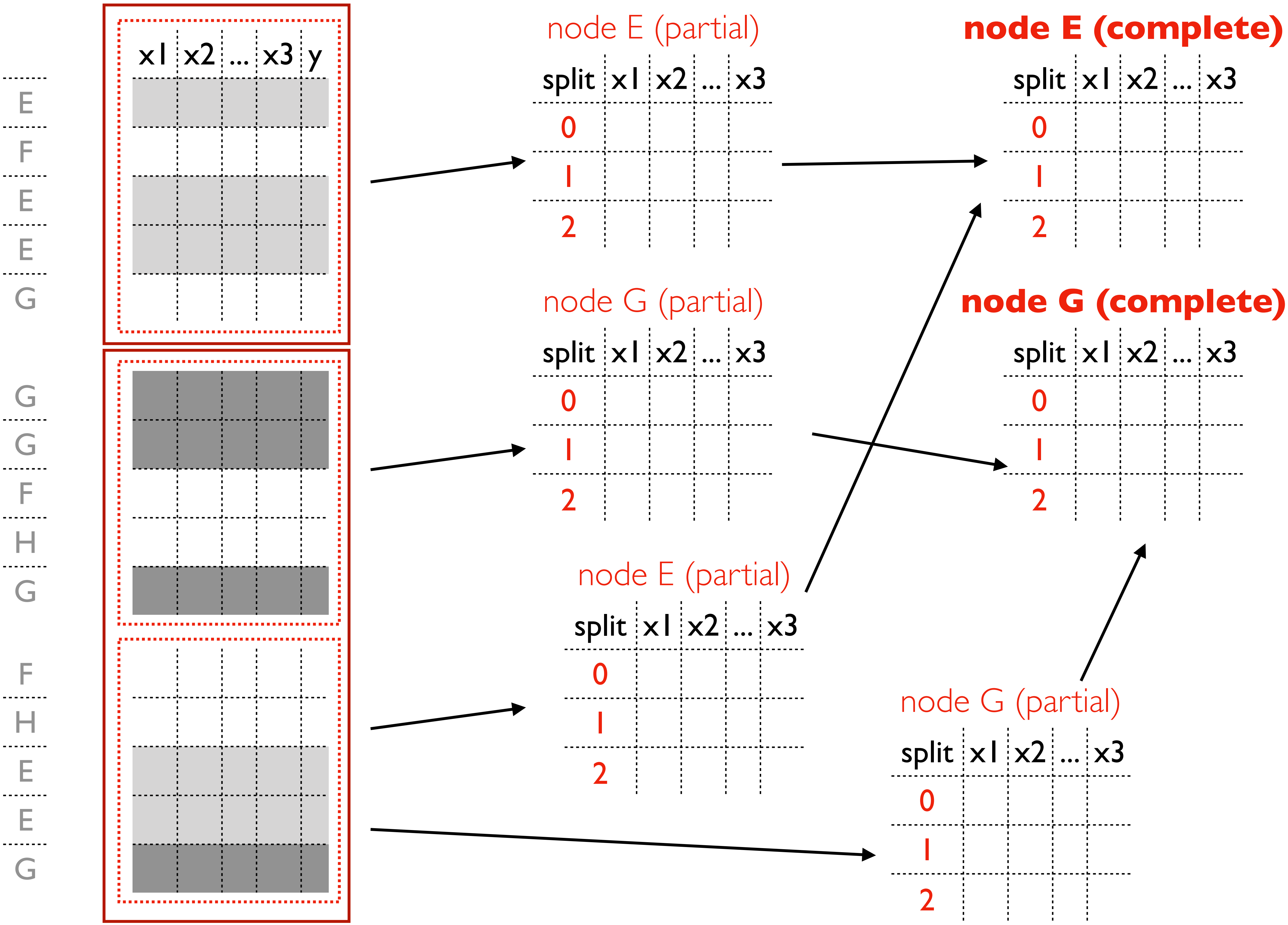


stats per feature/split combo

- left no: *number*
- left yes: *number*
- right no: *number*
- right yes: *number*

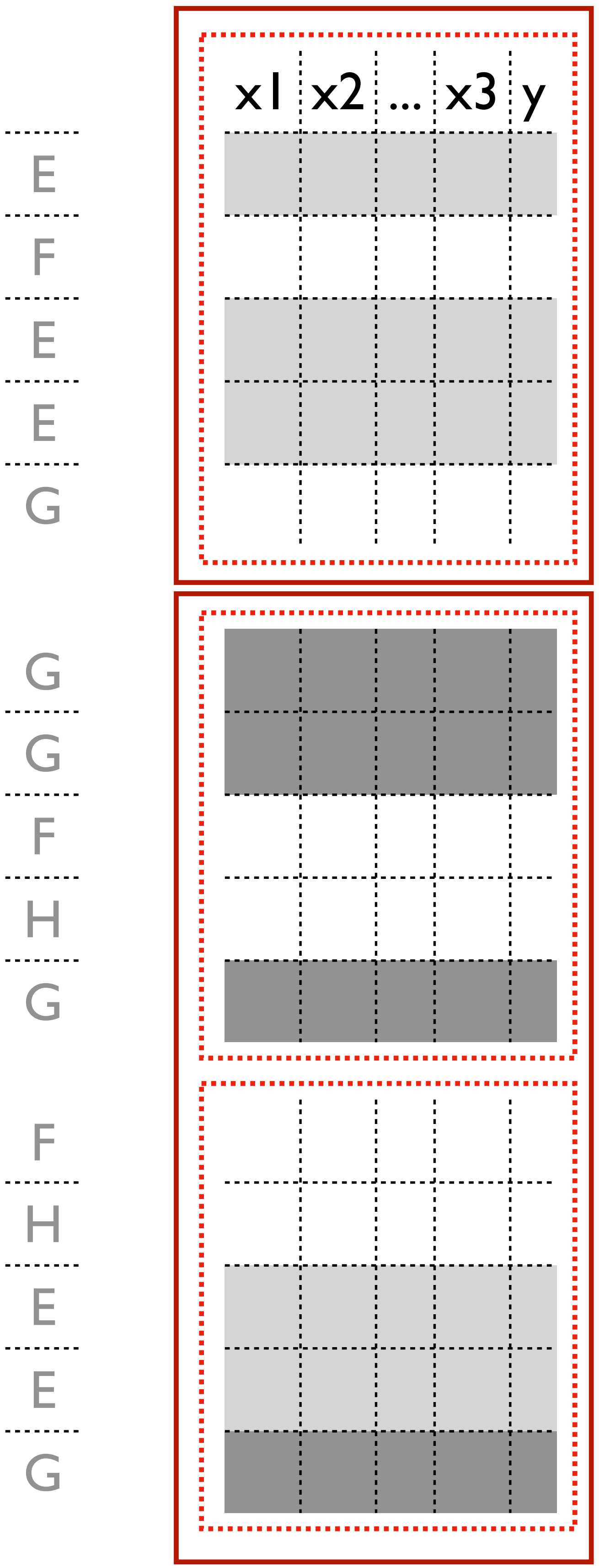
```
dt = DecisionTreeClassifier(labelCol="y")
dt.setMaxBins(3)
```

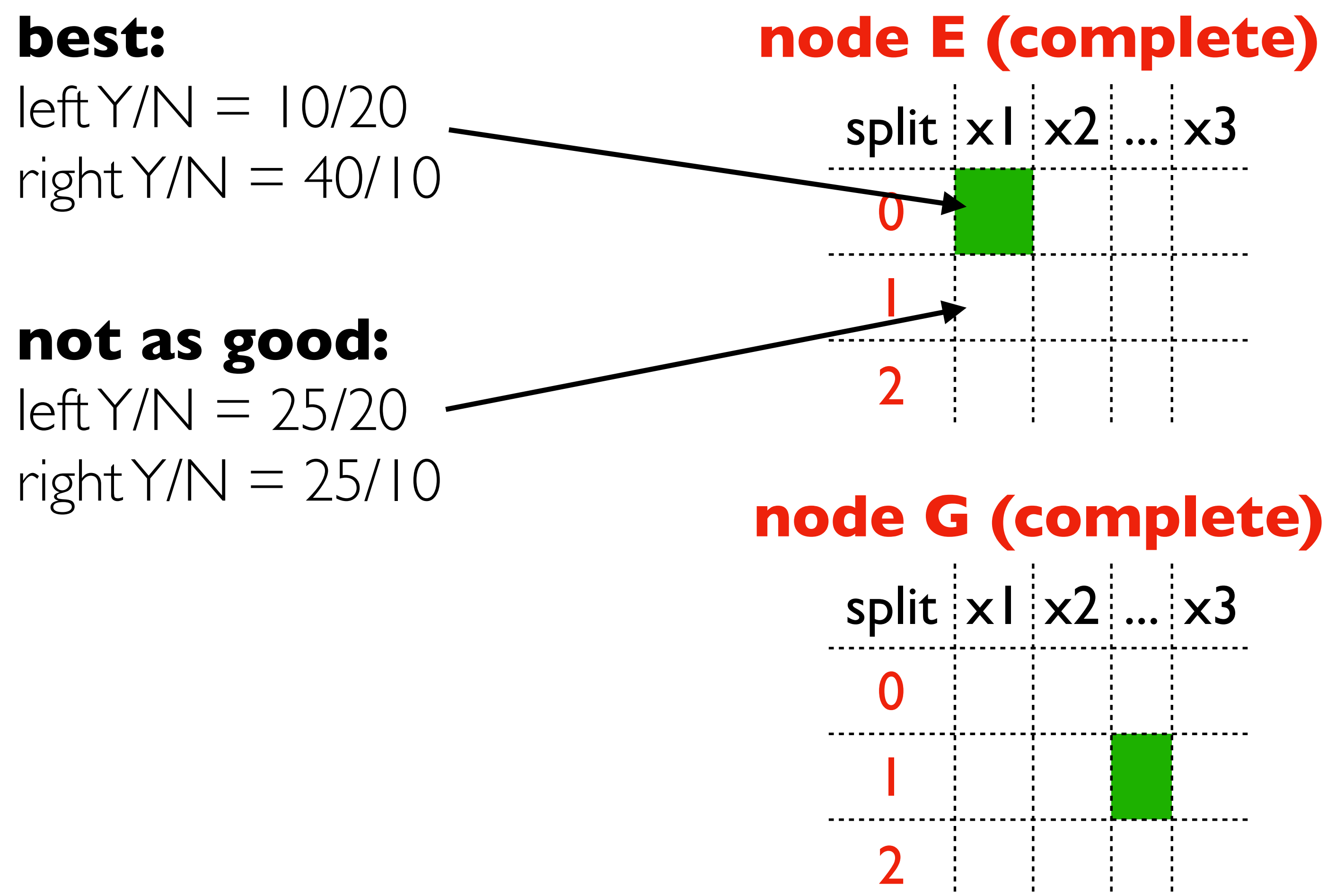
Big Nodes



x1 x2 ... x3 y
E
F
E
E
G
G
G
F
H
G
F
H
E
E
G
node E (partial)
split x1 x2 ... x3
0
1
2
node G (partial)
split x1 x2 ... x3
0
1
2
node E (partial)
split x1 x2 ... x3
0
1
2
node G (partial)
split x1 x2 ... x3
0
1
2
node E (complete)
split x1 x2 ... x3
0
1
2
node G (complete)
split x1 x2 ... x3
0
1
2

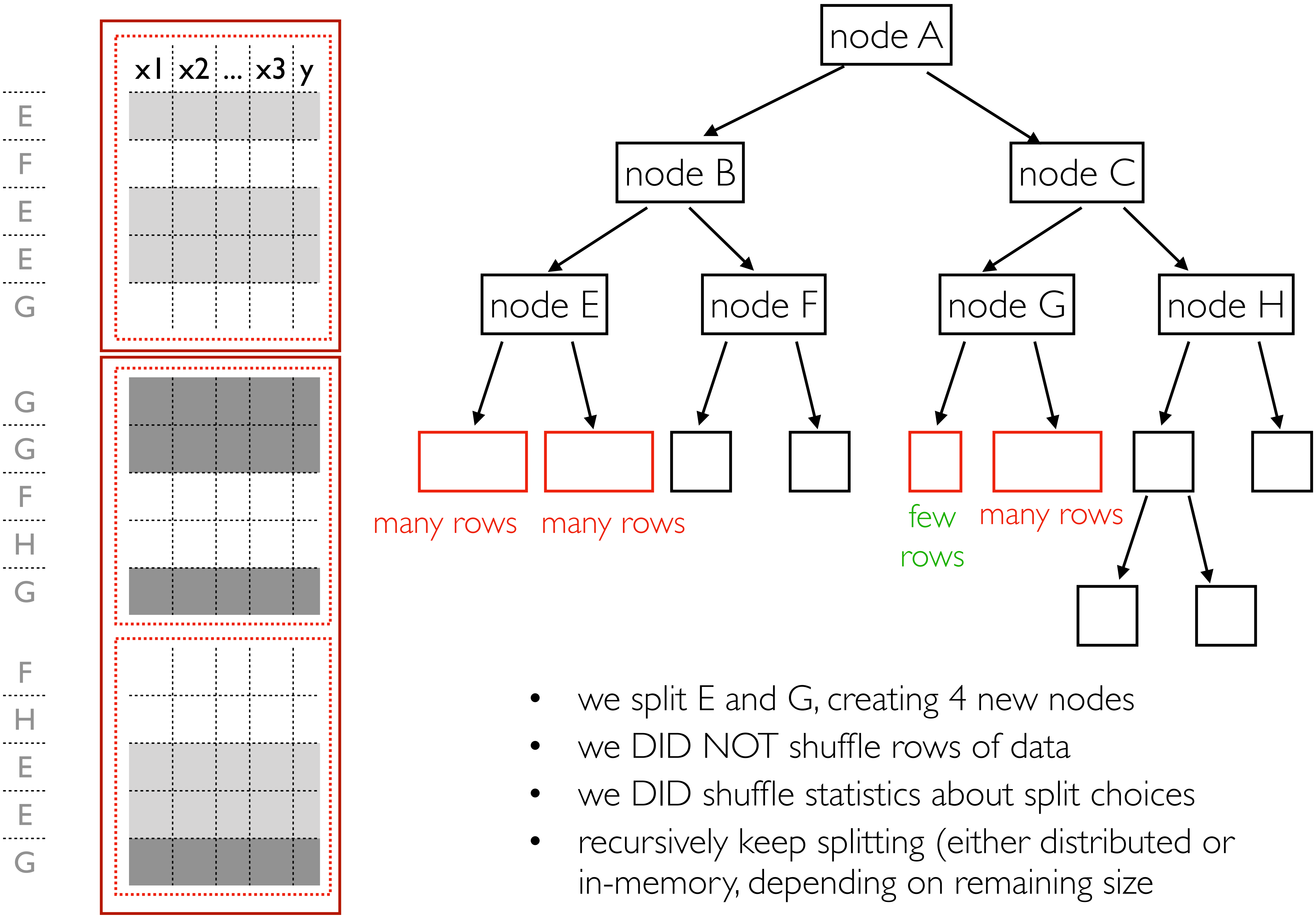
Big Nodes





- each table corresponds to a node we can split
  (we will choose best split for each node)
- each column represents a feature we could split on
- each row represents a threshold we could use for that split

New Nodes



- we split E and G, creating 4 new nodes
- we DID NOT shuffle rows of data
- we DID shuffle statistics about split choices
- recursively keep splitting (either distributed or in-memory, depending on remaining size

TopHat

Distribued ML Outline

Background: Tree-based Machine Learning Methods

Training Trees:

- in memory
- PLANET algorithm

Model Deployment

- UDF
- Streaming
- Server

Demos