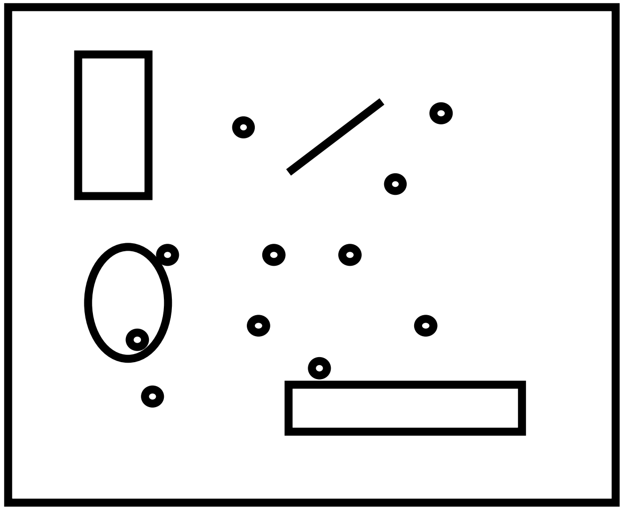
Spatial Indexing I

R-trees

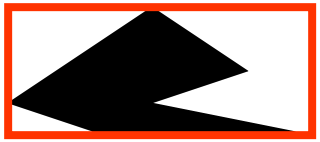
Problem

- Given a collection of geometric objects (points, lines, polygons, ...)
- organize them on disk, to answer efficiently spatial queries (range, nn, etc)



R-tree

- In multidimensional space, there is no unique ordering! Not possible to use B+-trees☹
- [Guttman 84] R-tree!
- Group objects close in space in the same node
  - => guaranteed page utilization
  - => easy insertion/split algorithms.
  - (only deal with Minimum Bounding Rectangles - **MBR**s)

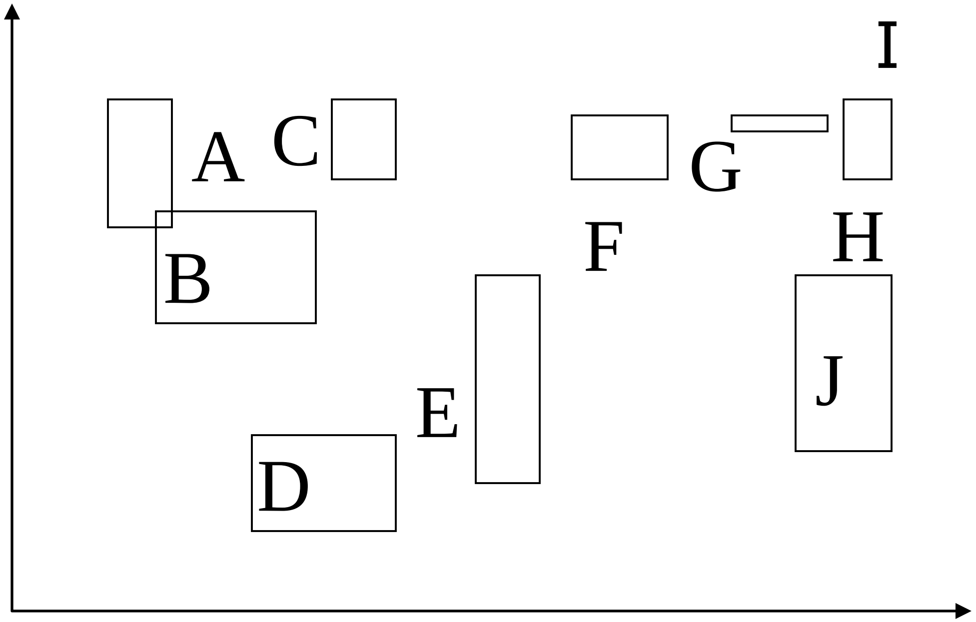


R-tree

- A multi-way external memory tree
- Index nodes and data (leaf) nodes
- All leaf nodes appear on the same level
- Every node contains between m and M entries
- The root node has at least 2 entries (children)

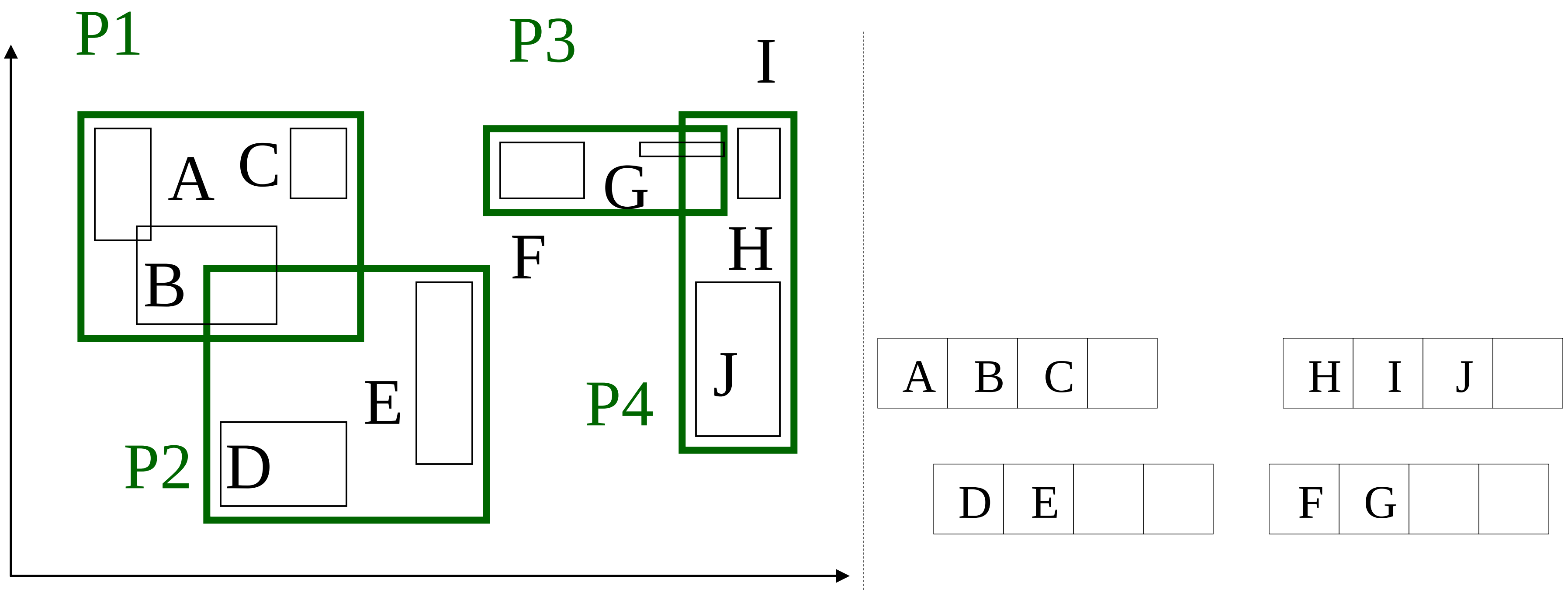
Example

- eg., w/ fanout 4: group nearby rectangles to parent MBRs; each group -> disk page



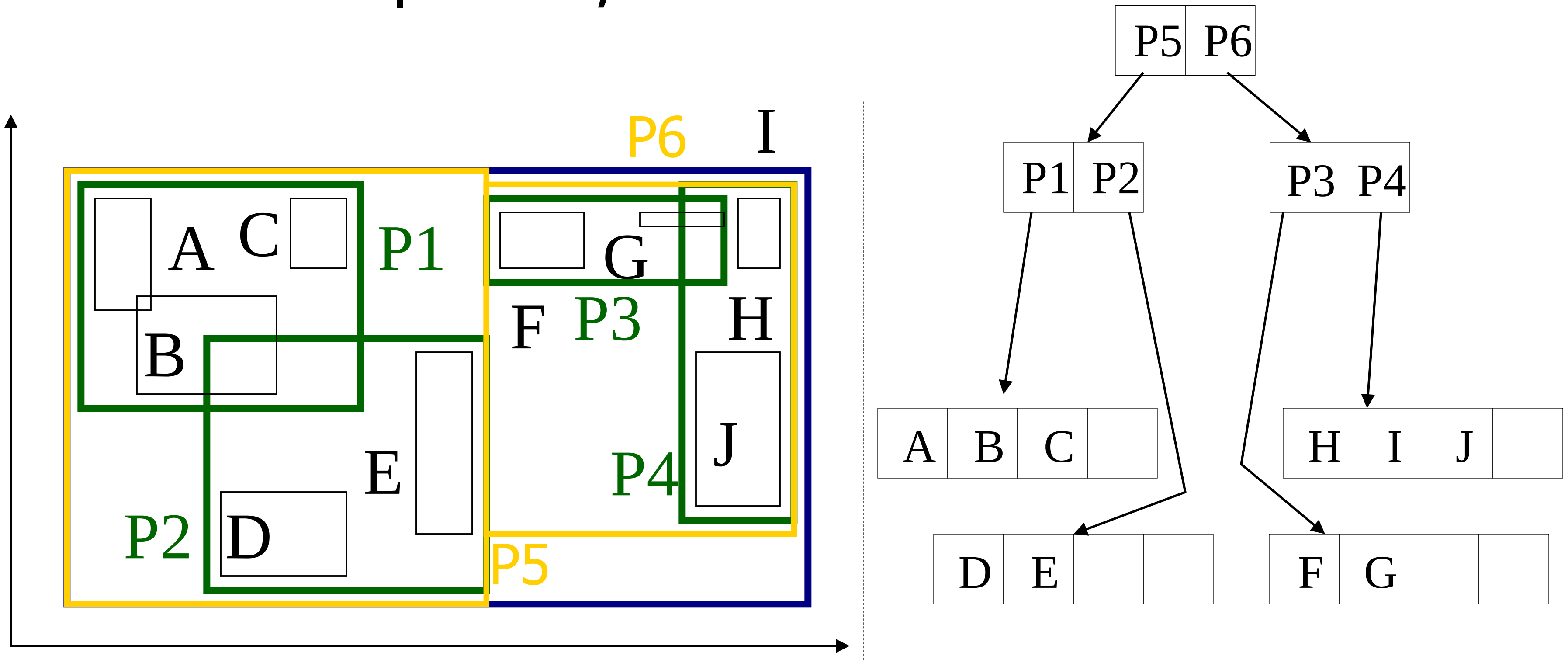
Example

- F=4



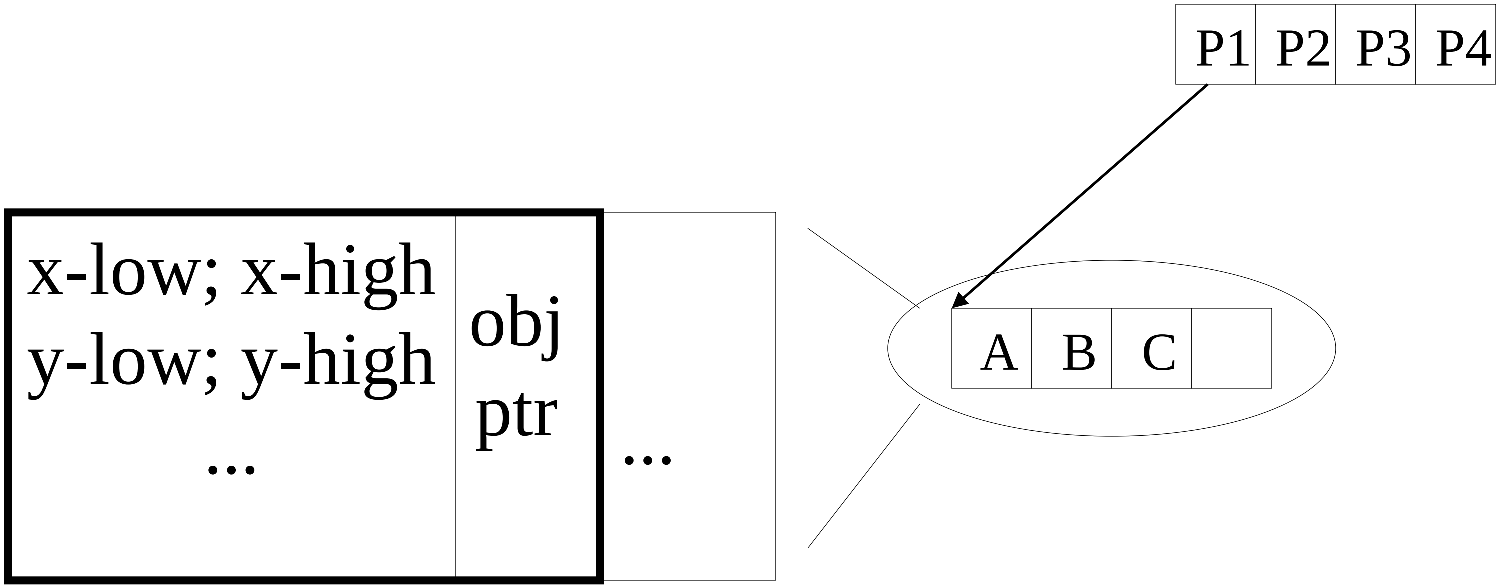
Example

- F=4| m=2, M=4



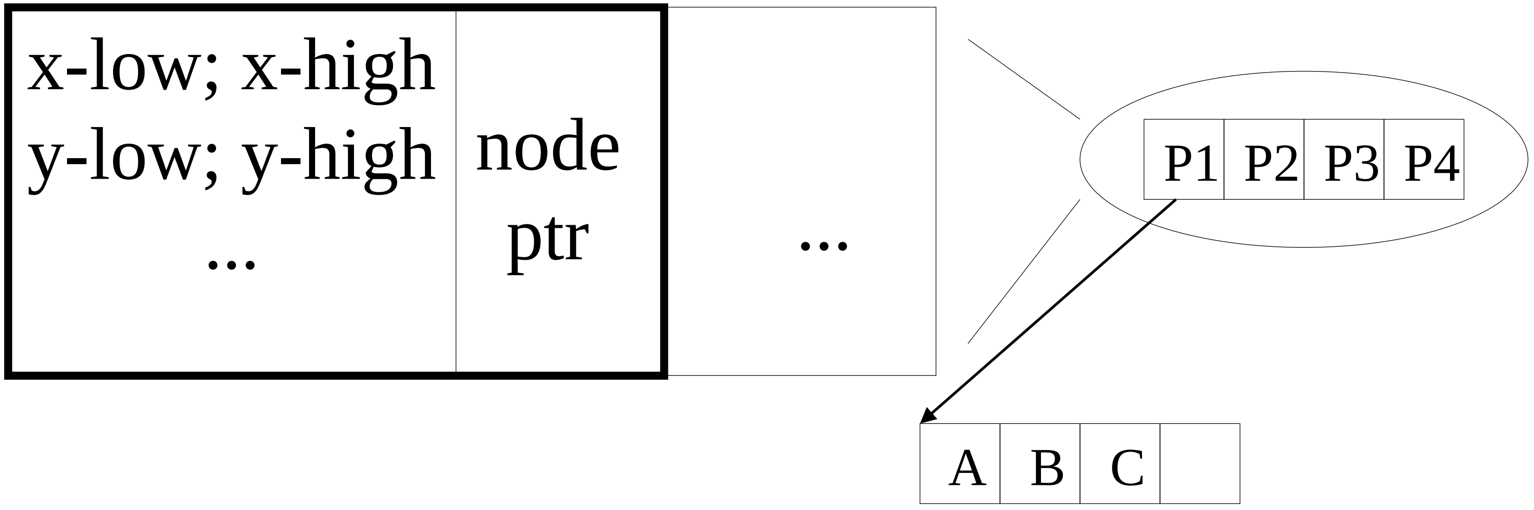
R-trees - format of nodes

- {(MBR; obj_ptr)} for leaf nodes

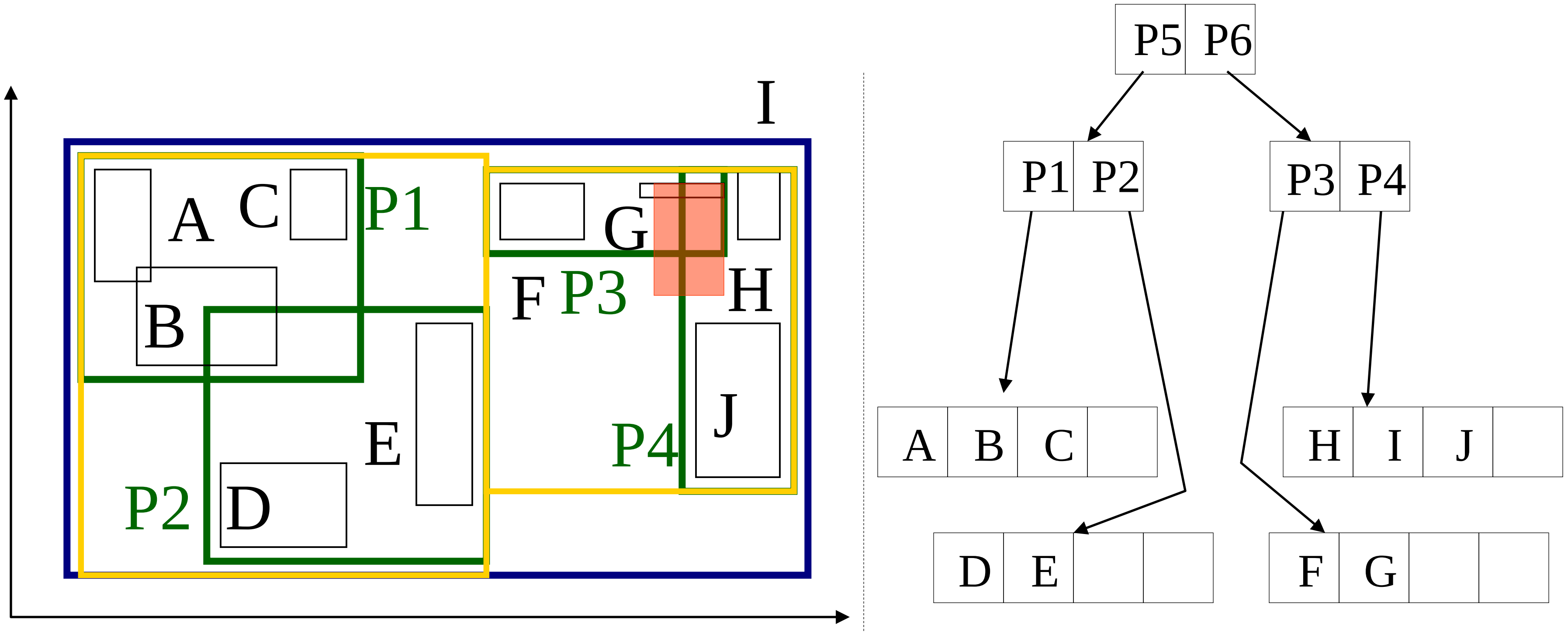


R-trees - format of nodes

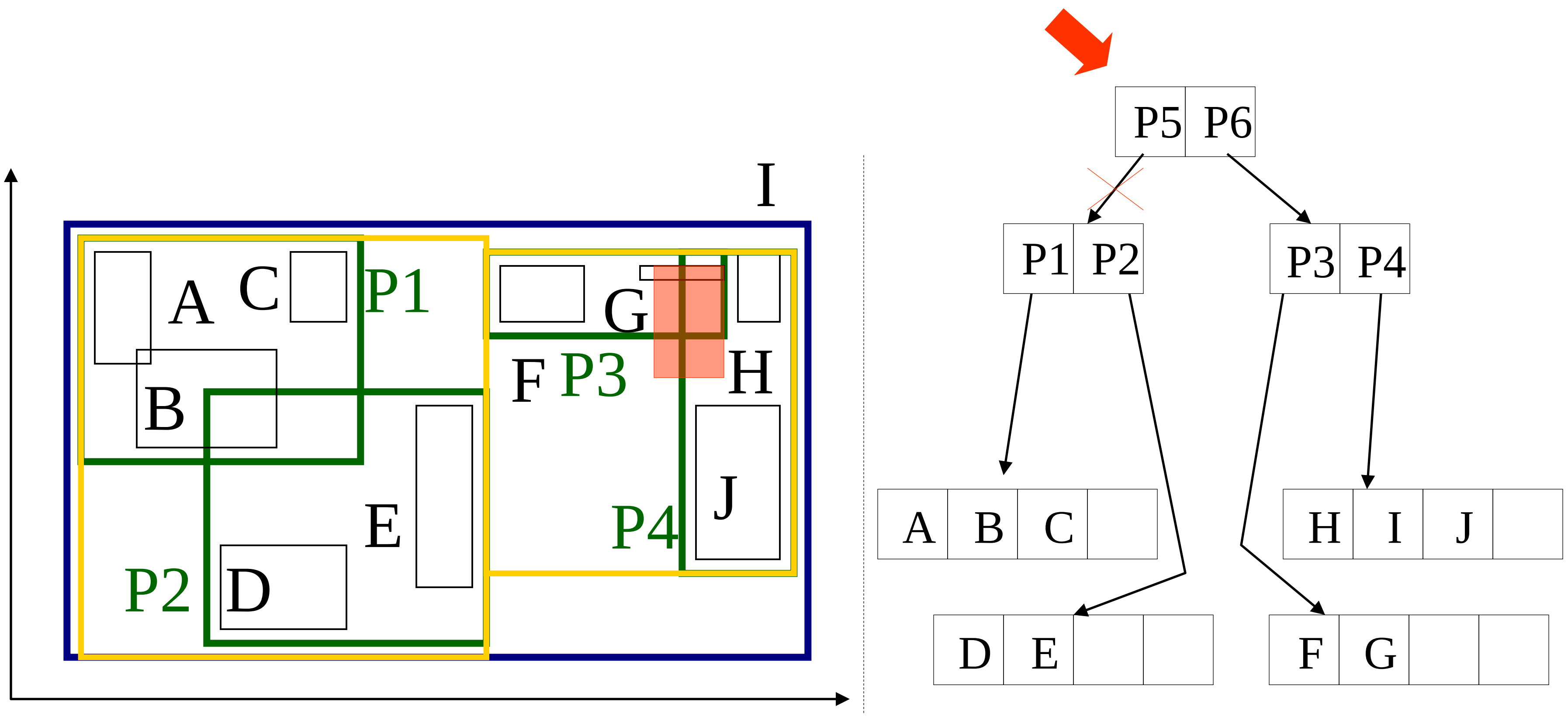
- {(MBR; node_ptr)} for non-leaf nodes



R-trees:Search



R-trees:Search

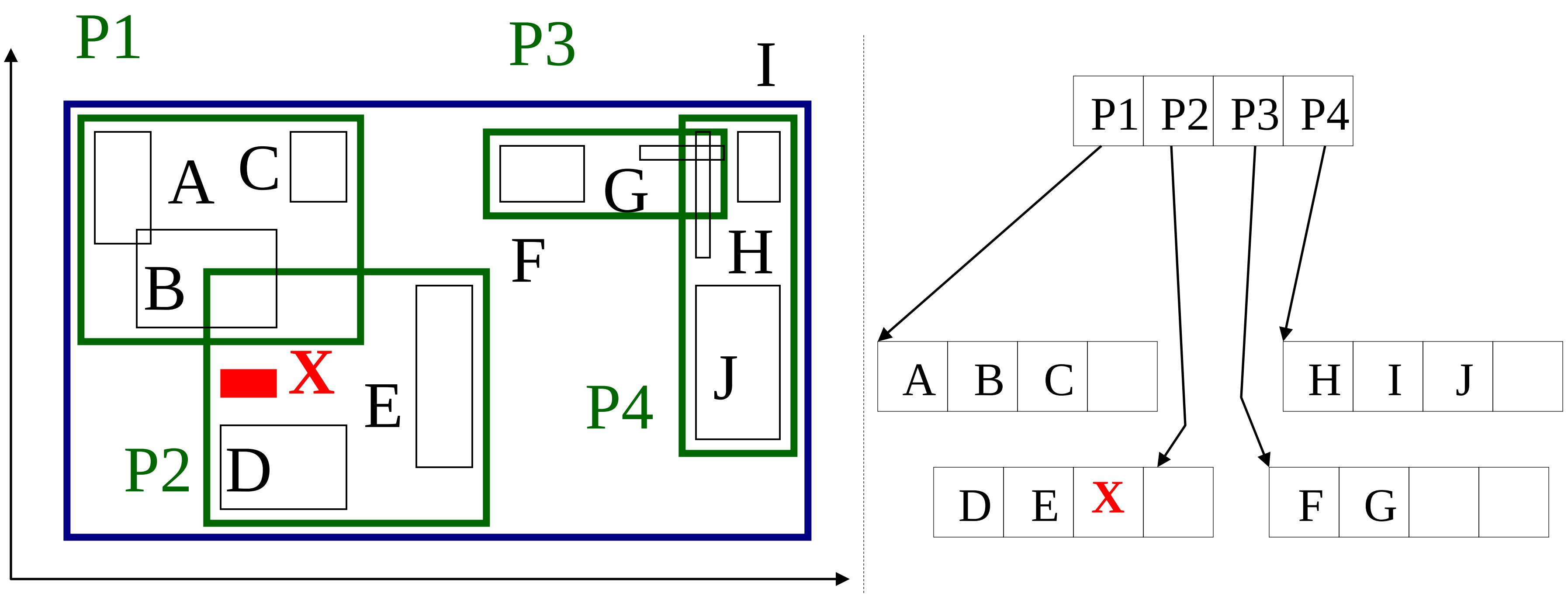


R-trees:Search

- Main points:
  - every parent node completely covers its 'children'
  - nodes in the same level may overlap!
  - a child MBR may be covered by more than one parent - it is stored under ONLY ONE of them. (ie., no need for dup. elim.)
  - a point query may follow multiple branches.
  - everything works for **any(?)** dimensionality

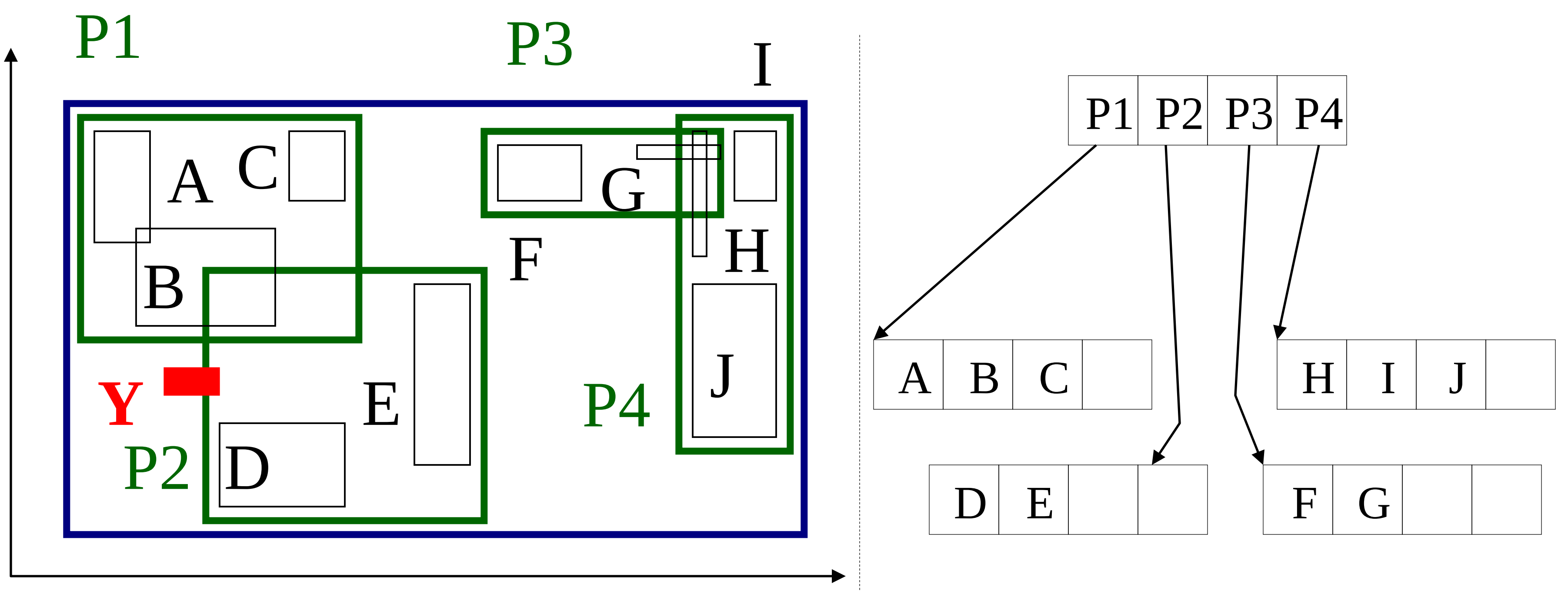
R-trees:Insertion

Insert X



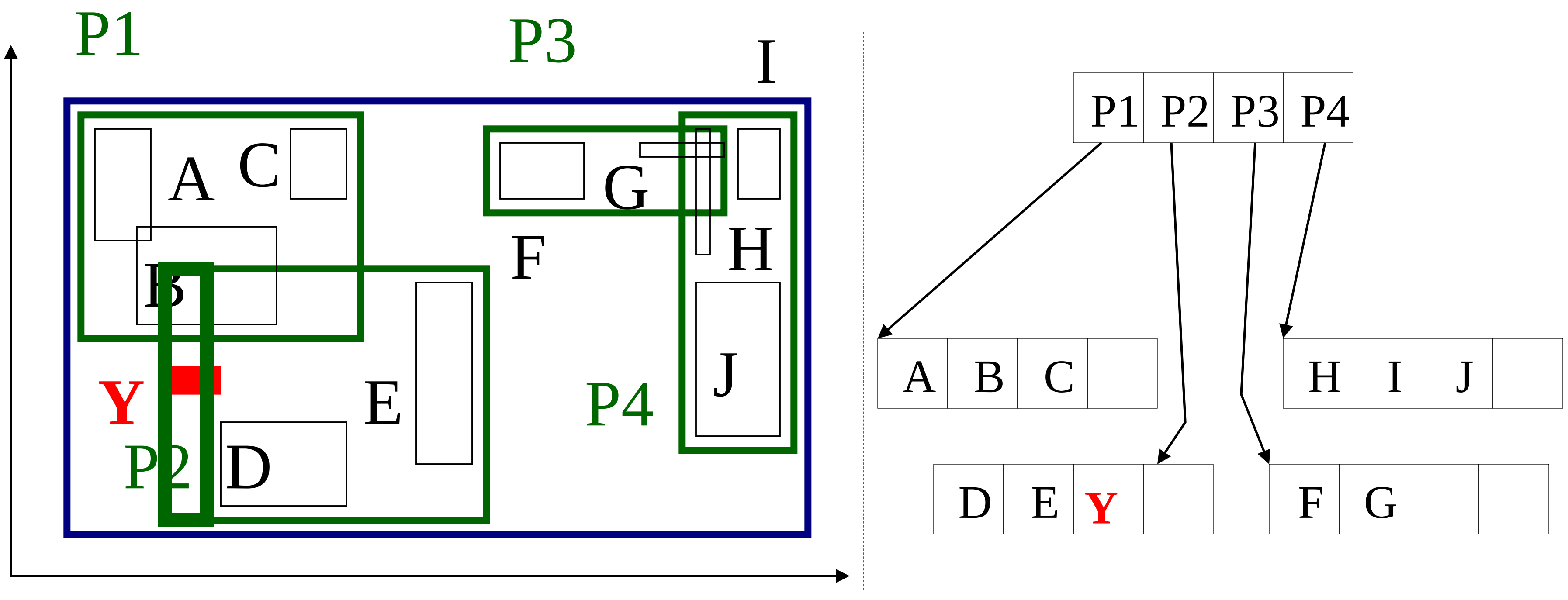
R-trees:Insertion

Insert Y



R-trees:Insertion

- Extend the parent MBR

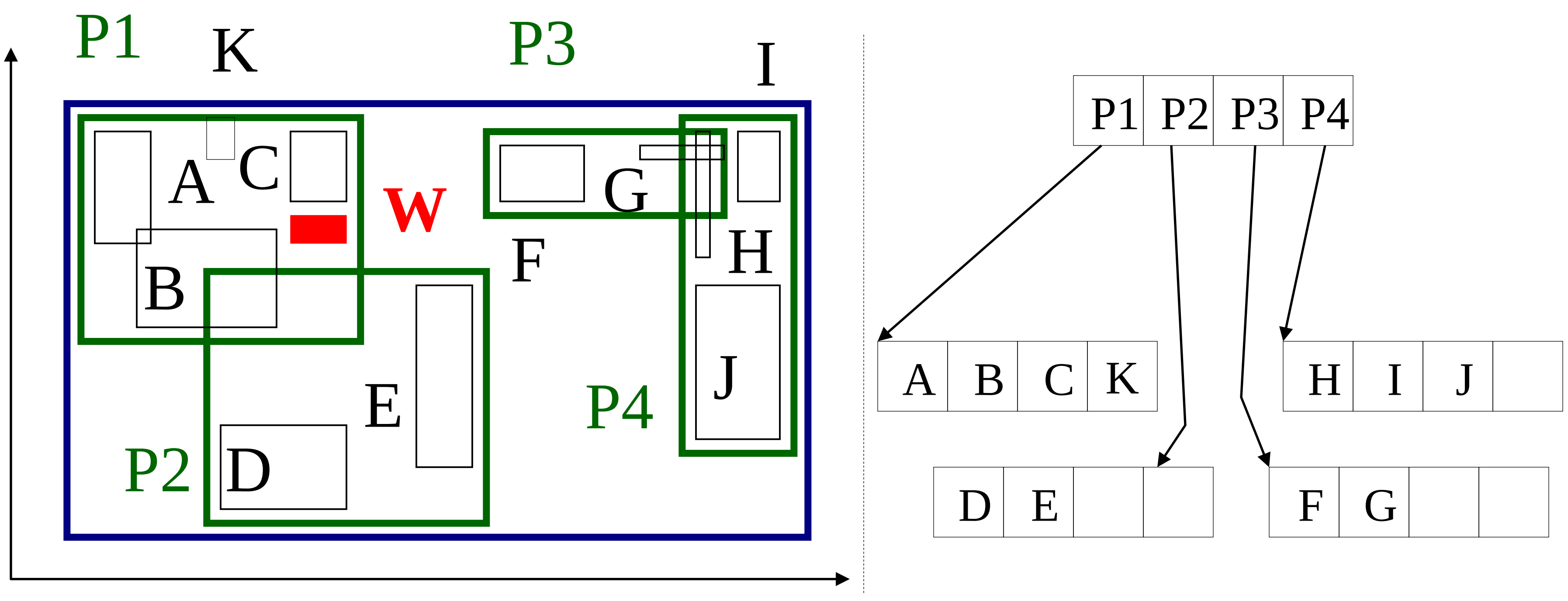


R-trees:Insertion

- How to find the next node to insert a new object Y?
  - Using ChooseLeaf: Find the entry that needs the least enlargement to include Y. Resolve ties using the area (smallest)
- Other methods (later)

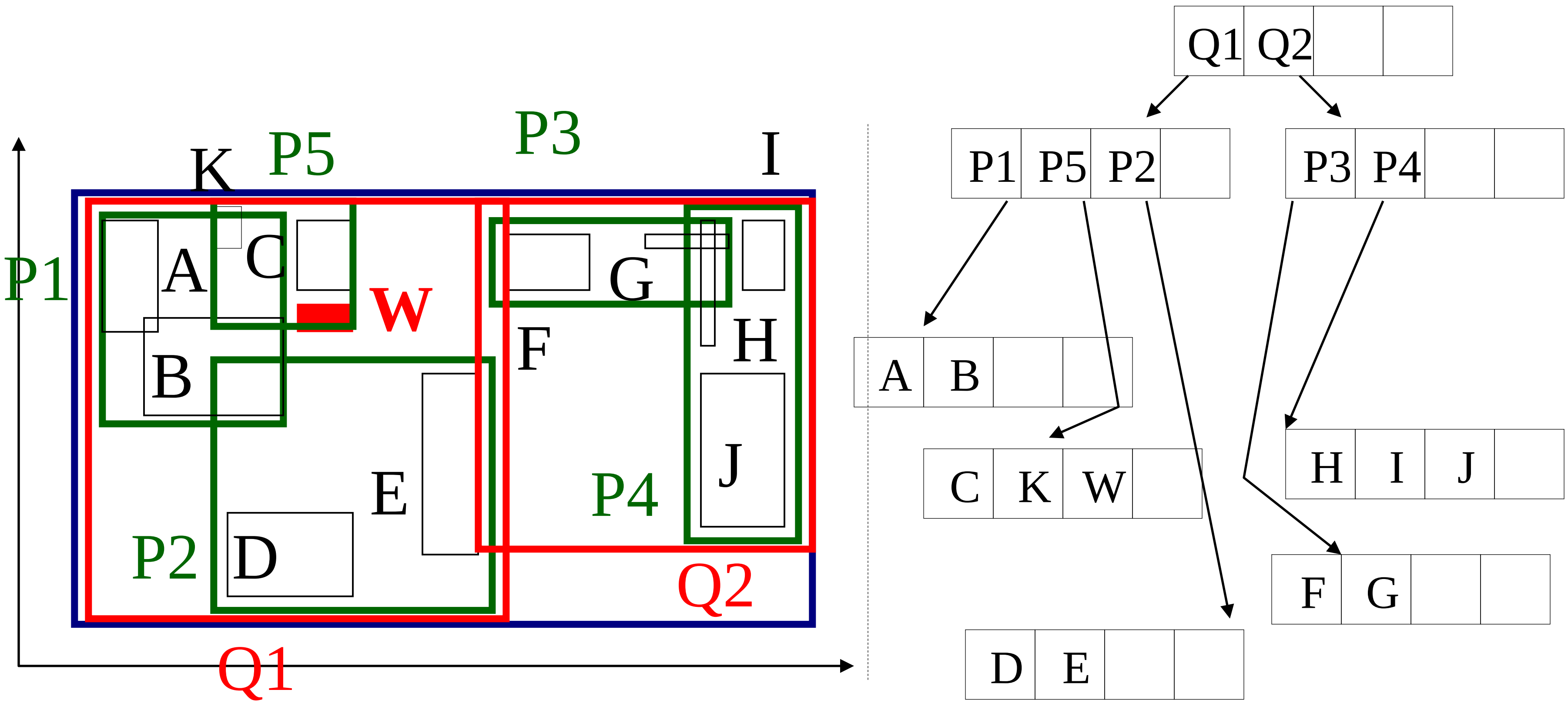
R-trees:Insertion

- If node is full then Split : ex. Insert w



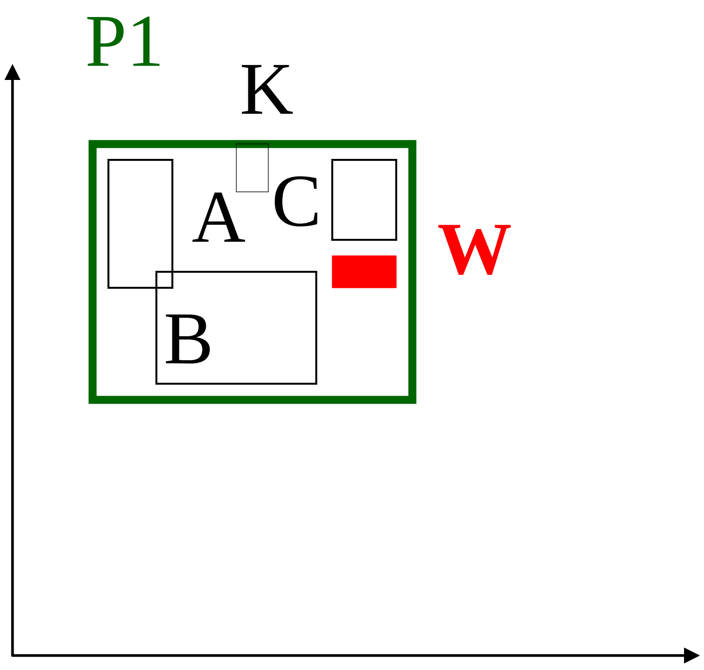
R-trees:Insertion

- If node is full then Split : ex. Insert w



R-trees:Split

- Split node P1: partition the MBRs into two groups.



- (A1: plane sweep,

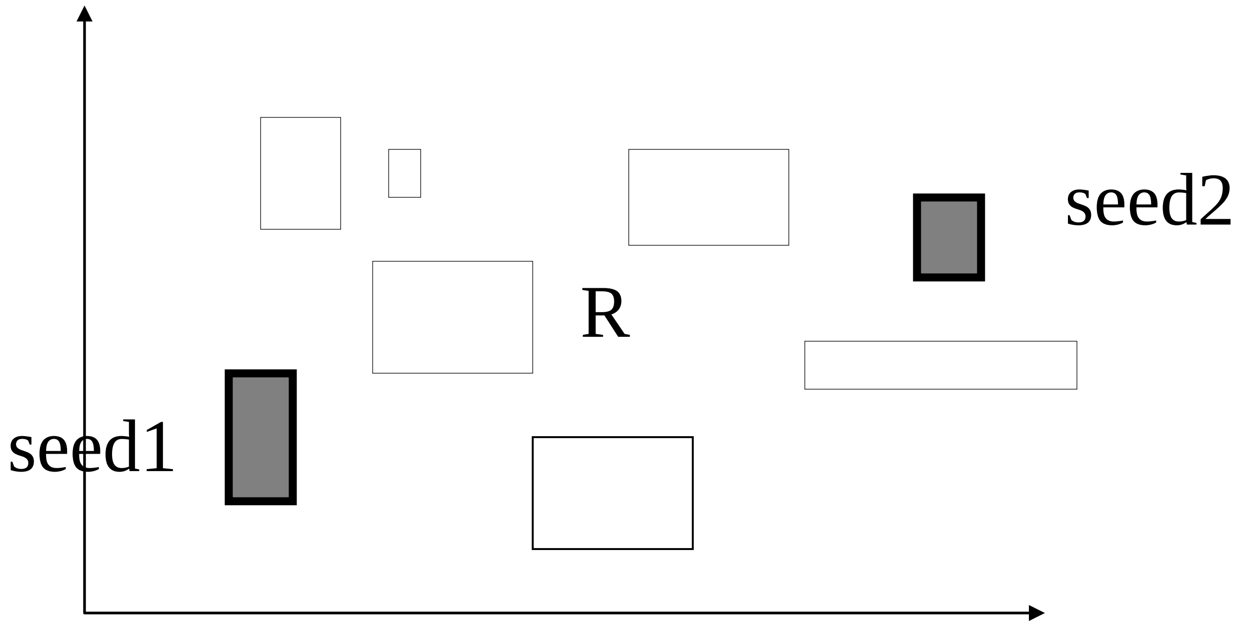
  until 50% of rectangles)

- A2: ‘linear’ split
- A3: quadratic split
- A4: exponential split:

  2^{M-1} choices

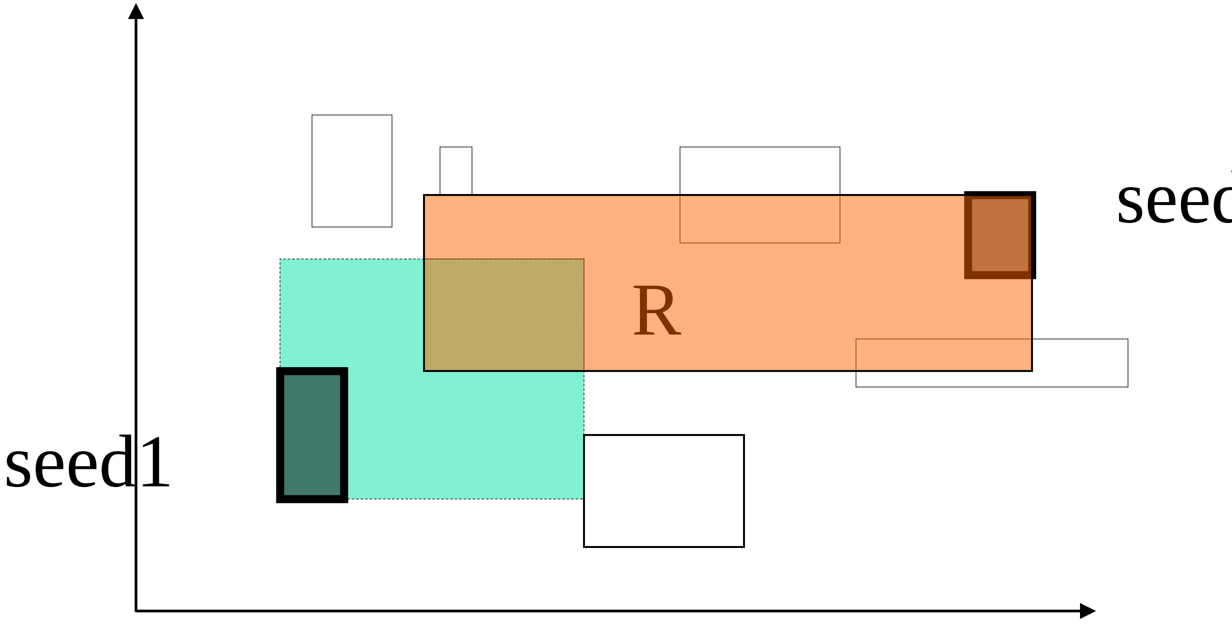
R-trees:Split

pick two rectangles as ‘seeds’ for group 1 and group 2;



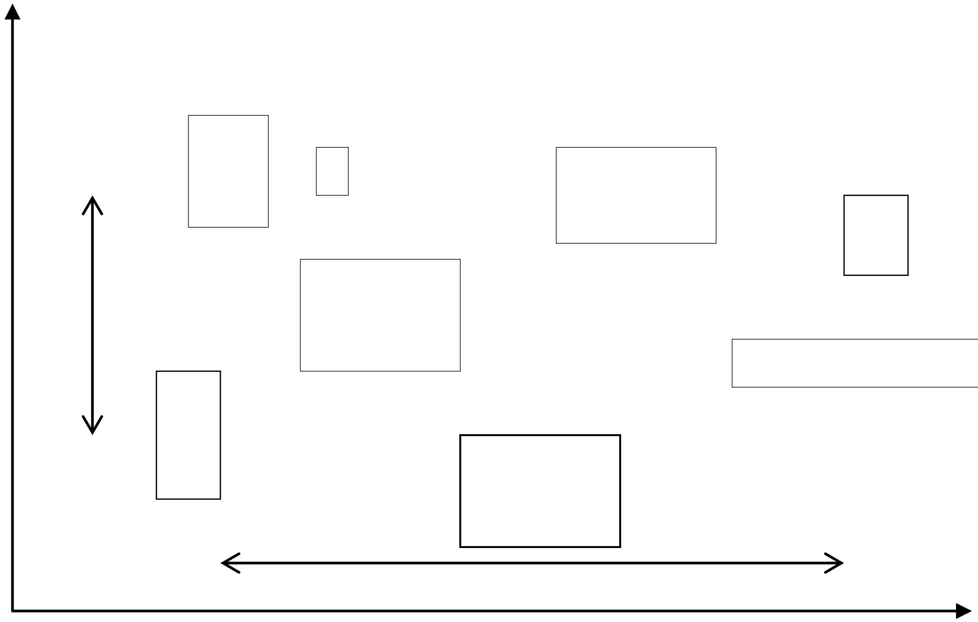
R-trees:Split

- pick two rectangles as ‘seeds’ for group 1 and group 2;
- assign each rectangle ‘R’ to the ‘closest’ ‘group’:
- ‘closest’: the smallest increase in area



R-trees:Linear Split

- How to pick Seeds:
  - Find the rects with the highest low and lowest high sides in each dimension
  - Normalize the separations by dividing by the width of all the rects in the corresponding dim
  - Choose the pair with the greatest normalized separation
- PickNext: pick one of the remaining rects and add it to the closest group



R-trees: Quadratic Split

- How to pick Seeds:
  - For each pair E1 and E2, calculate the rectangle J=MBR(E1, E2) and d= J-E1-E2. Choose the pair with the largest d
- PickNext:
  - For each remaining rect calculate the area increase to include it in group d1 and d2
  - Choose rect with highest difference: |d1-d2|
  - Assign to closest group

R-trees:Insertion

- Use the **ChooseLeaf** to find the leaf node to insert an entry E
- If leaf node is full, then **Split,** otherwise insert there
  - Propagate the split upwards, if necessary
- Adjust parent nodes

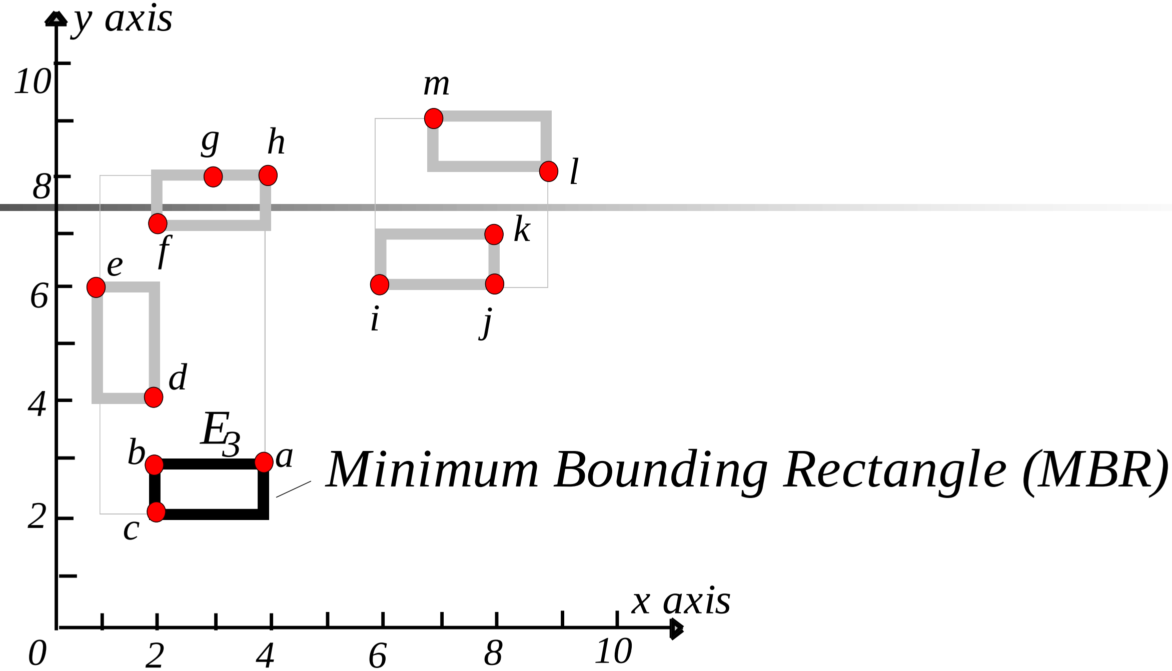
R-Trees:Deletion

- Find the leaf node that contains the entry E
- Remove E from this node
- If underflow:
  - Eliminate the node by removing the node entries and the parent entry
  - Reinsert the orphaned (other entries) into the tree using **Insert**

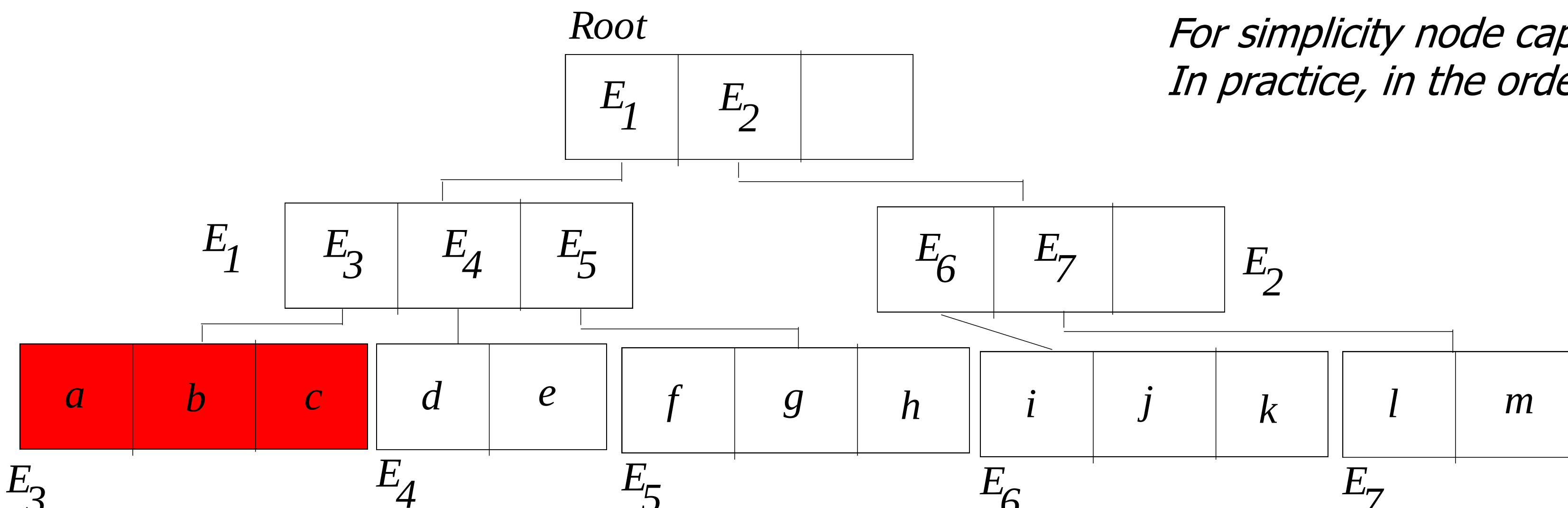
- Other method **(later)**

R-trees: Variations

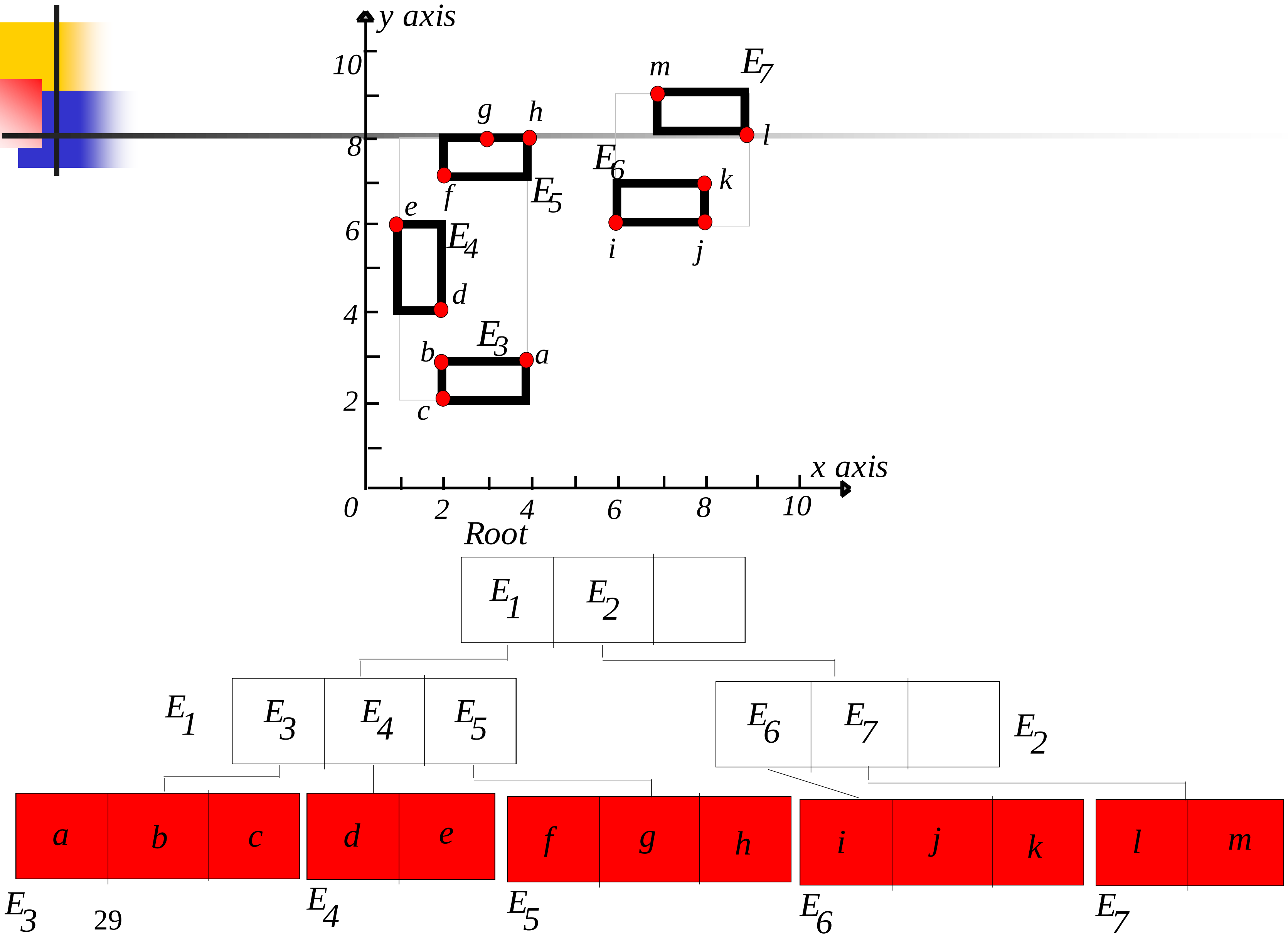
- R+-tree: DO not allow overlapping, so split the objects (similar to z-values)
- R*-tree: change the insertion, deletion algorithms (minimize not only area but also perimeter, forced re-insertion)



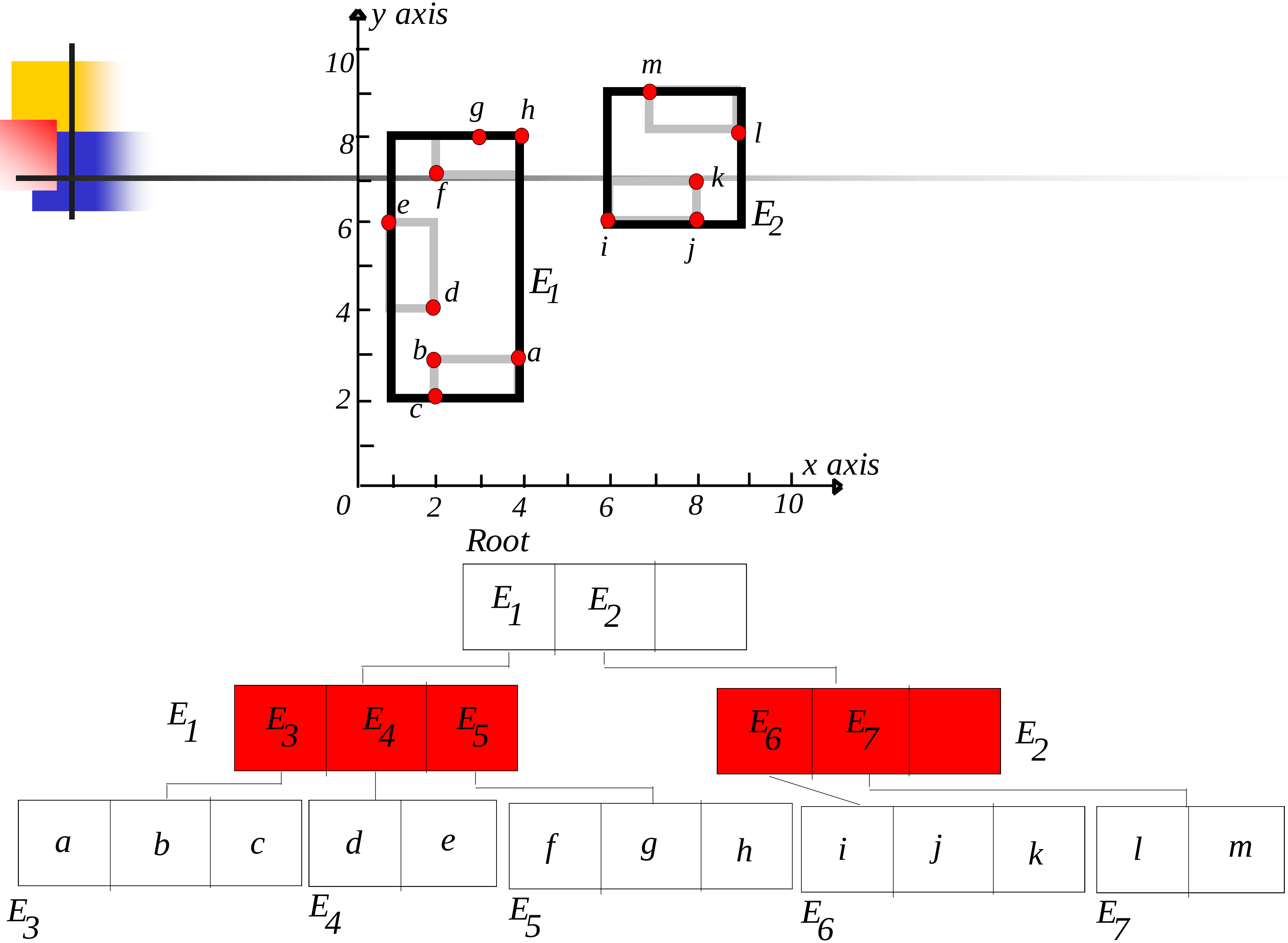
For simplicity node capacity = 3
In practice, in the order of 100



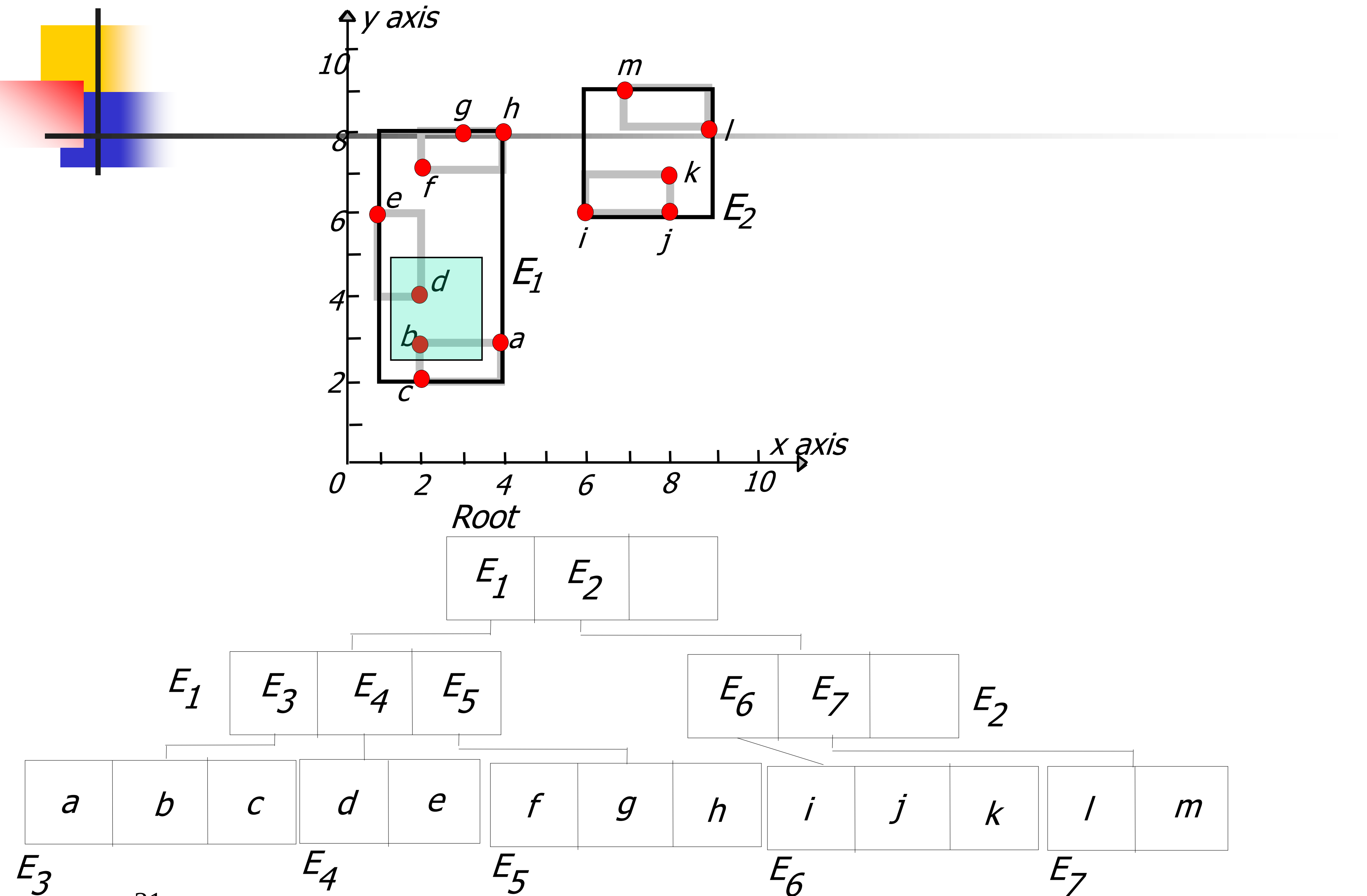
R-Tree, Leaf Nodes



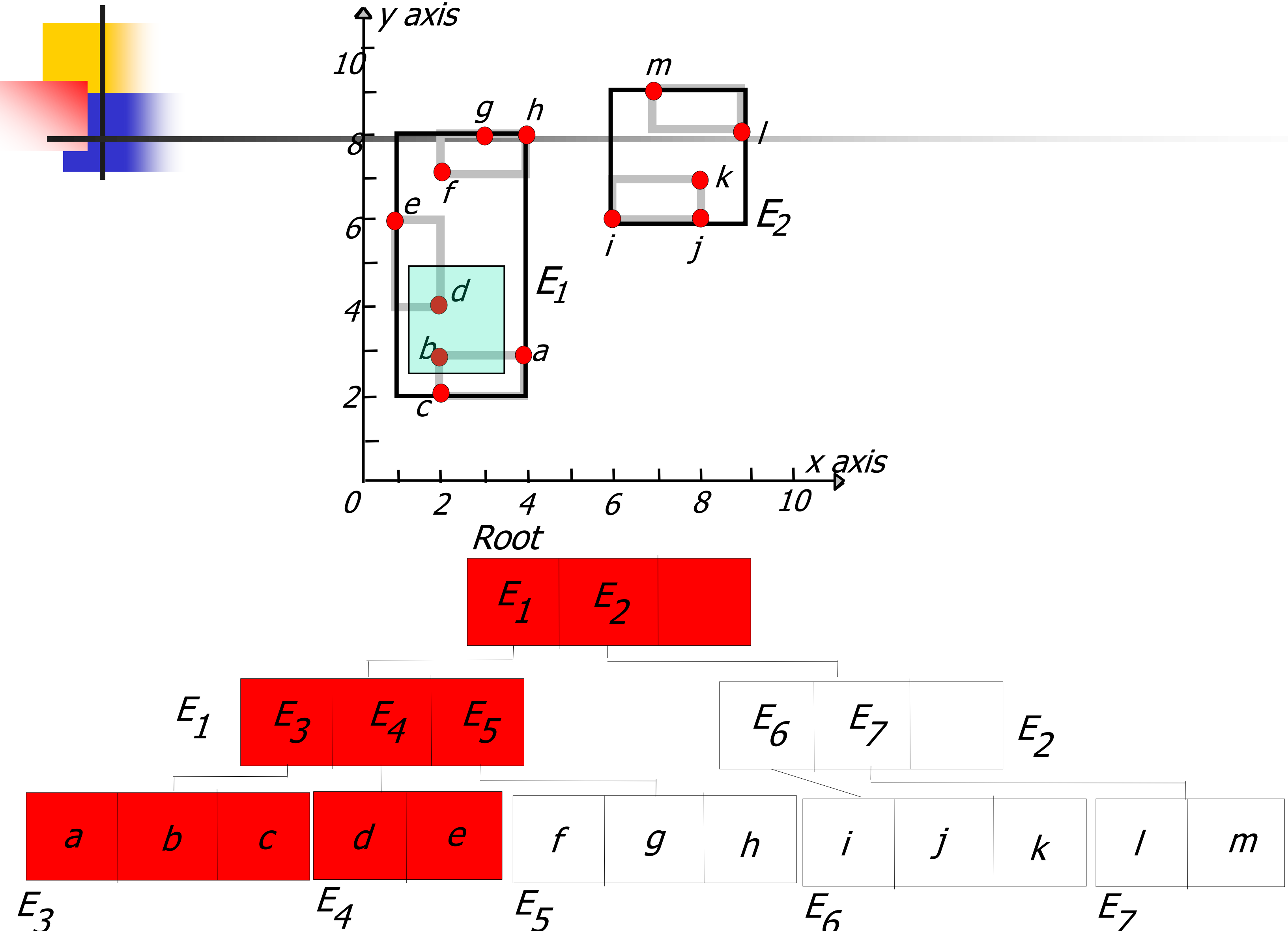
R-Tree – Intermediate Nodes



R-tree, Range Query



Range Query



y axis
x axis
0
2
4
6
8
10
m
g
h
l
k
f
e
E2
i
j
d
E1
b
a
c
Root
E1
E2
E1
E3
E4
E5
E6
E7
E2
a
b
c
d
e
f
g
h
i
j
k
l
m
E3
E4
E5
E6
E7

R-tree

- The original R-tree tries to minimize the area of each enclosing rectangle in the index nodes.
- Is there any other property that can be optimized?

R*-tree → Yes!

R*-tree

- Optimization Criteria:
  - (O1) Area covered by an index MBR
  - (O2) Overlap between directory MBRs
  - (O3) Margin of a directory rectangle
  - (O4) Storage utilization
- Sometimes it is impossible to optimize all the above criteria at the same time!

R*-tree

- ChooseLeaf:
  - If next node is not a leaf node, choose the node using the following criteria:
    - Least overlap enlargement
    - Least area enlargement
    - Smaller area
  - Else
    - Least area enlargement
    - Smaller area

R*-tree

- SplitNode
  - Choose the axis to split
  - Choose the two groups along the chosen axis
- ChooseSplitAxis
  - Along each axis, sort rectangles and break them into two groups (M-2m+2 possible ways where one group contains at least m rectangles). Compute the sum S of all <u>margin-values (perimeters)</u> of each pair of groups. Choose the one that minimizes S
- ChooseSplitIndex
  - Along the chosen axis, choose the grouping that gives the minimum <u>overlap-value</u>

R*-tree

- Forced Reinsert:
  - defer splits, by forced-reinsert, i.e.: instead of splitting, temporarily delete some entries, shrink overflowing MBR, and re-insert those entries
- Which ones to re-insert?
  - The ones that are further way from the center
- How many? A: 30%