R-TREE Data Structure for Spatial Indexing

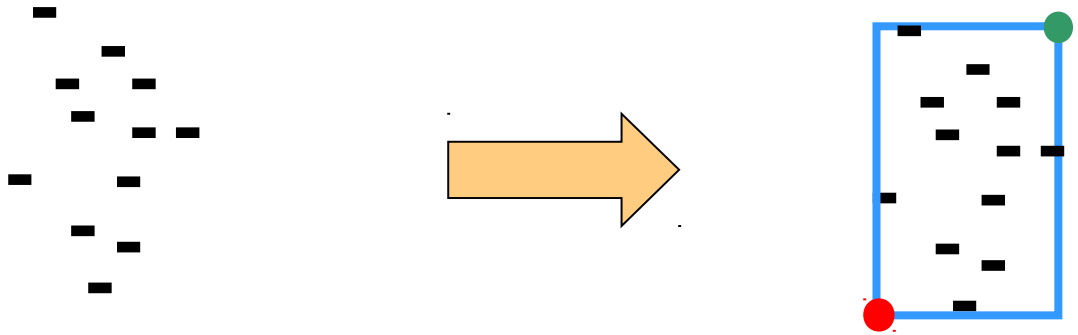
Sulaiman Muhammad (17IT143)

Spatial Database

- Database that is optimized for storing and quering geometrical data
- Represents objects defined in Geometric Space
- Like points, geographical co-ordinates, rectangles, polygons or 3-D objects
- Spatial Indices are used by Spatial Database to optimize spatial queries
- Queries like nearest neighbour, range, topological, nearest surrounder
- R-Trees are the preferred method to implement Spatial Indexing

Suppose we have a cluster of points in 2-D space...

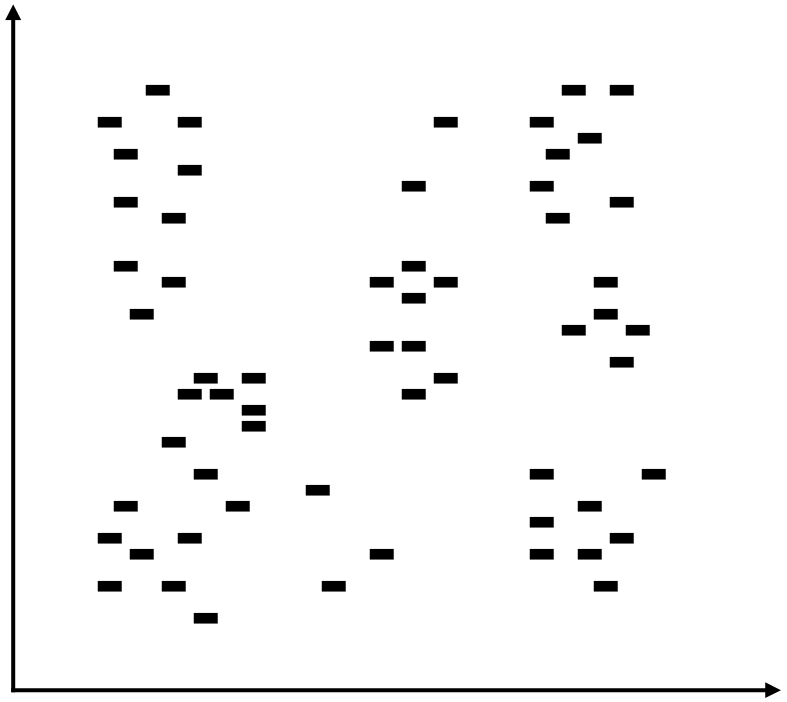
We can build a “box” around points. The smallest box (which is axis parallel) that contains all the points is called a Minimum Bounding Rectangle (MBR)

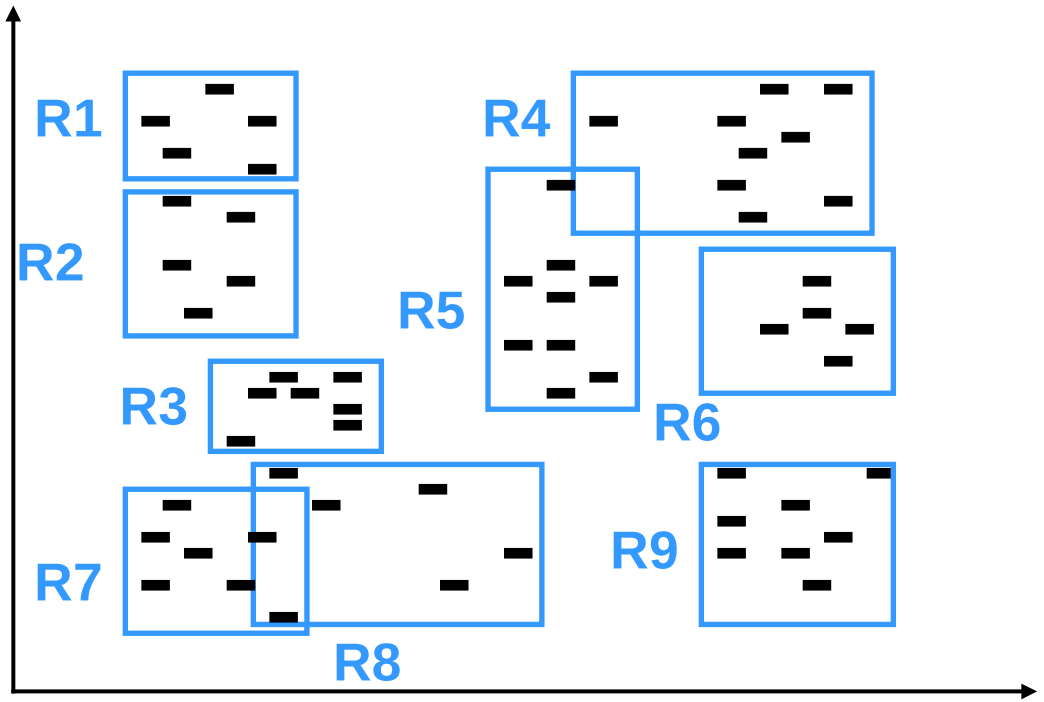


MBR = {(L.x,L.y)(U.x,U.y)}

Note that we only need two points to describe an MBR, we typically use lower left, and upper right.

We can visualize the locations of interest simply as points in space...



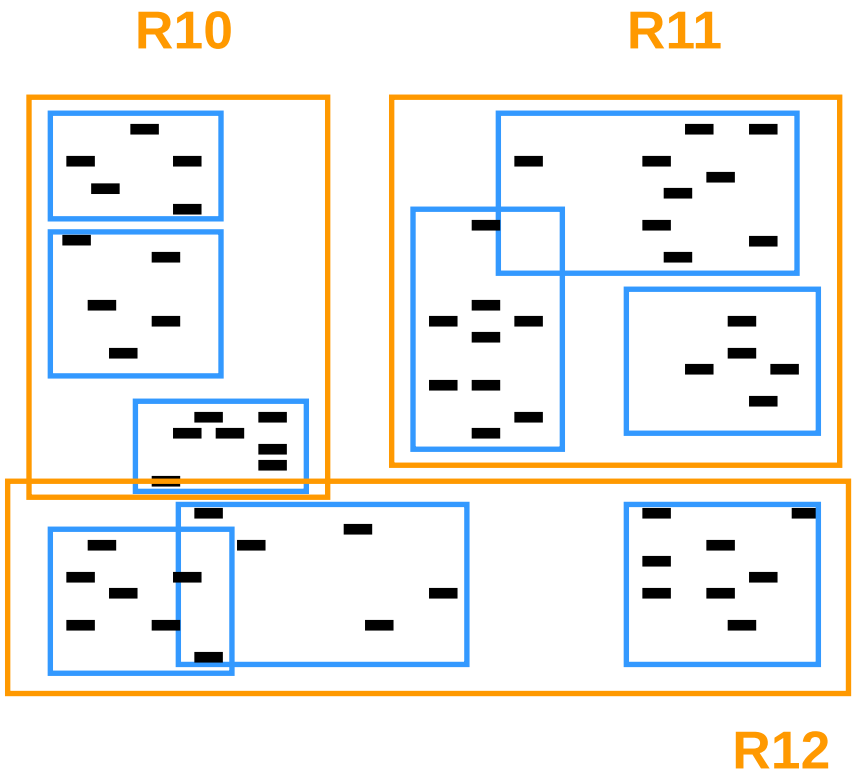


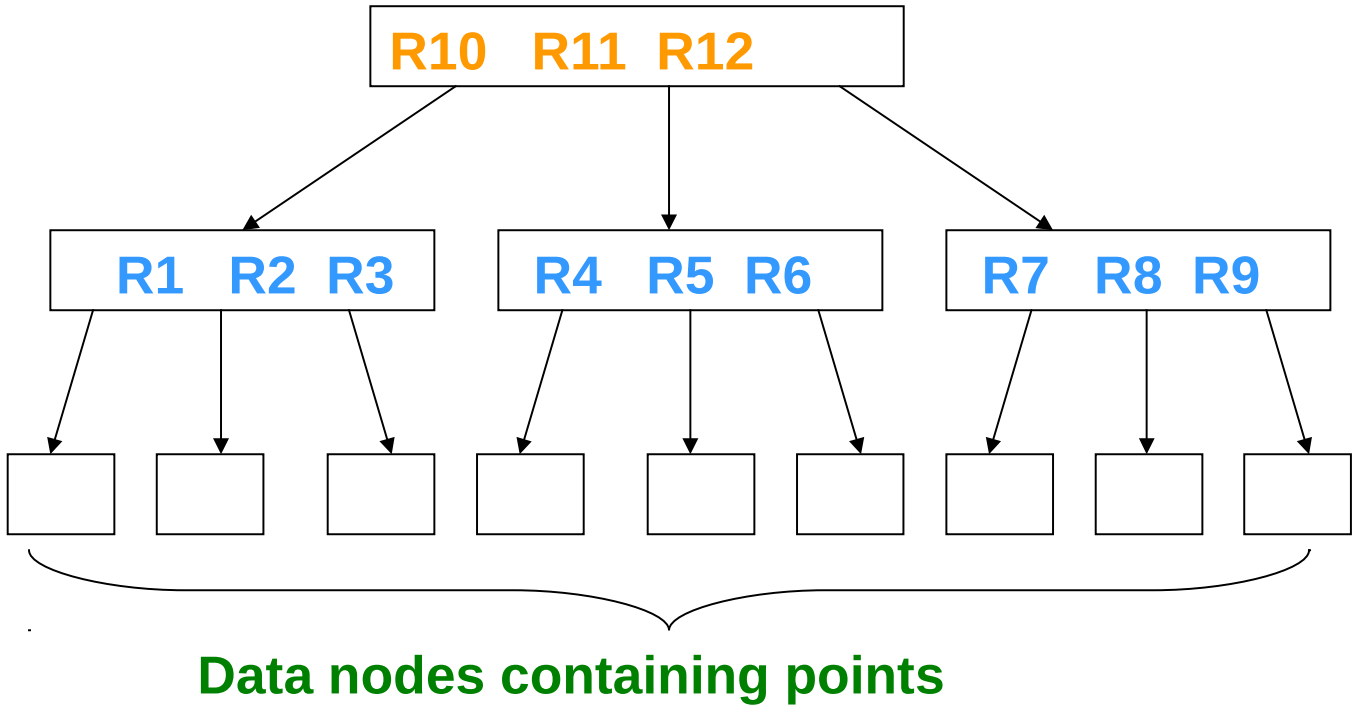
We can group clusters of datapoints into “boxes”, called Minimum Bounding Rectangles (MBRs).

Each MBR can be represented with just two points. The lower left corner, and the upper right corner.

We can further recursively group MBRs into larger MBRs….

…these nested MBRs are organized as a tree (called a spatial access tree or a multidimensional tree).





R-Trees

- Assume:
  1. M = Maximum number of entries in a node.
  2. m <= M/2
  3. N = Number of records
- R-tree has the following properties:
  - Every leaf node contains between m and M index records. Root node is the exception.
  - For each index record (I, tuple-identifier) in a leaf node, I is the smallest rectangle that spatially contains the n dimensional data object represented in the indicated tuple.
  - Every non-leaf node has between m and M children. Root node is the exception.
  - For each entry (I, child-pointer) in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node.
  - The root node has at least two children unless it is a leaf.
  - All leaves appear on the same level.
  - Height of a tree = $\log_m N-1$
  - Worst case utilization for all nodes except the root is m/M.

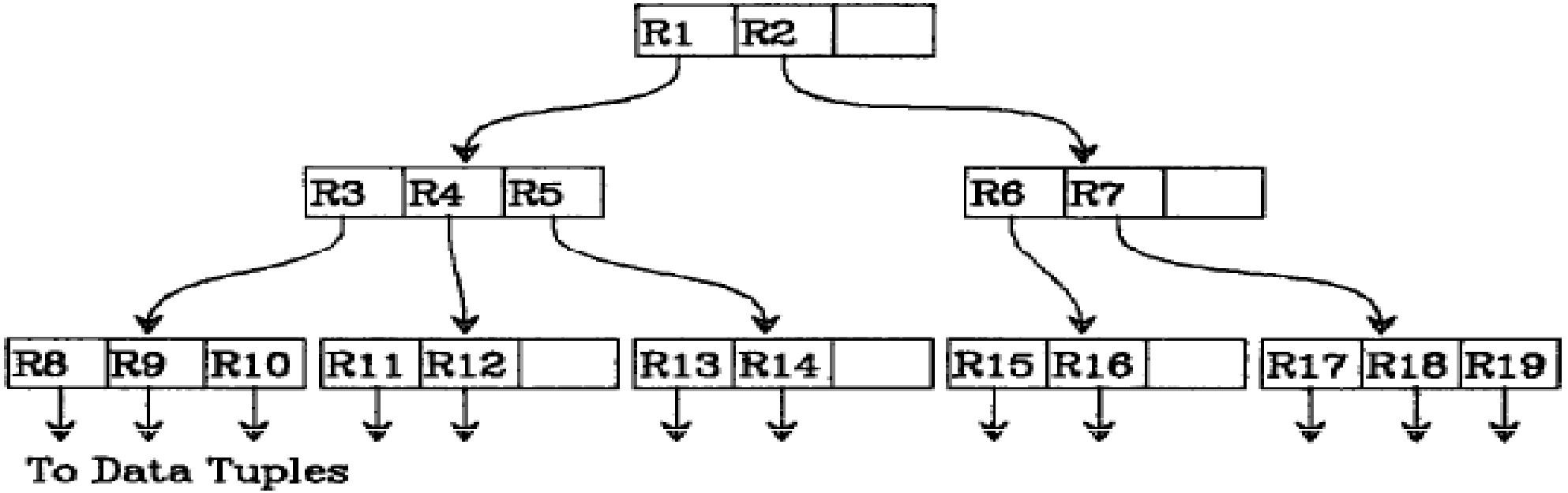
R-Tree: Leaf Nodes

- Leaf nodes contain index records:
  - (I, tuple-identifier)
- tuple-identifier is RID,
- I is an n-dimensional rectangle that bounds the indexed spatial object
- $I = (I_0, I_1, \ldots, I_{n-1})$ where n is the number of dimensions.
- I_i is a closed bounded interval [a,b] describing the extent of the object along dimension i.
- Values for a and b might be infinity, indicating an unbounded object along dimension i.

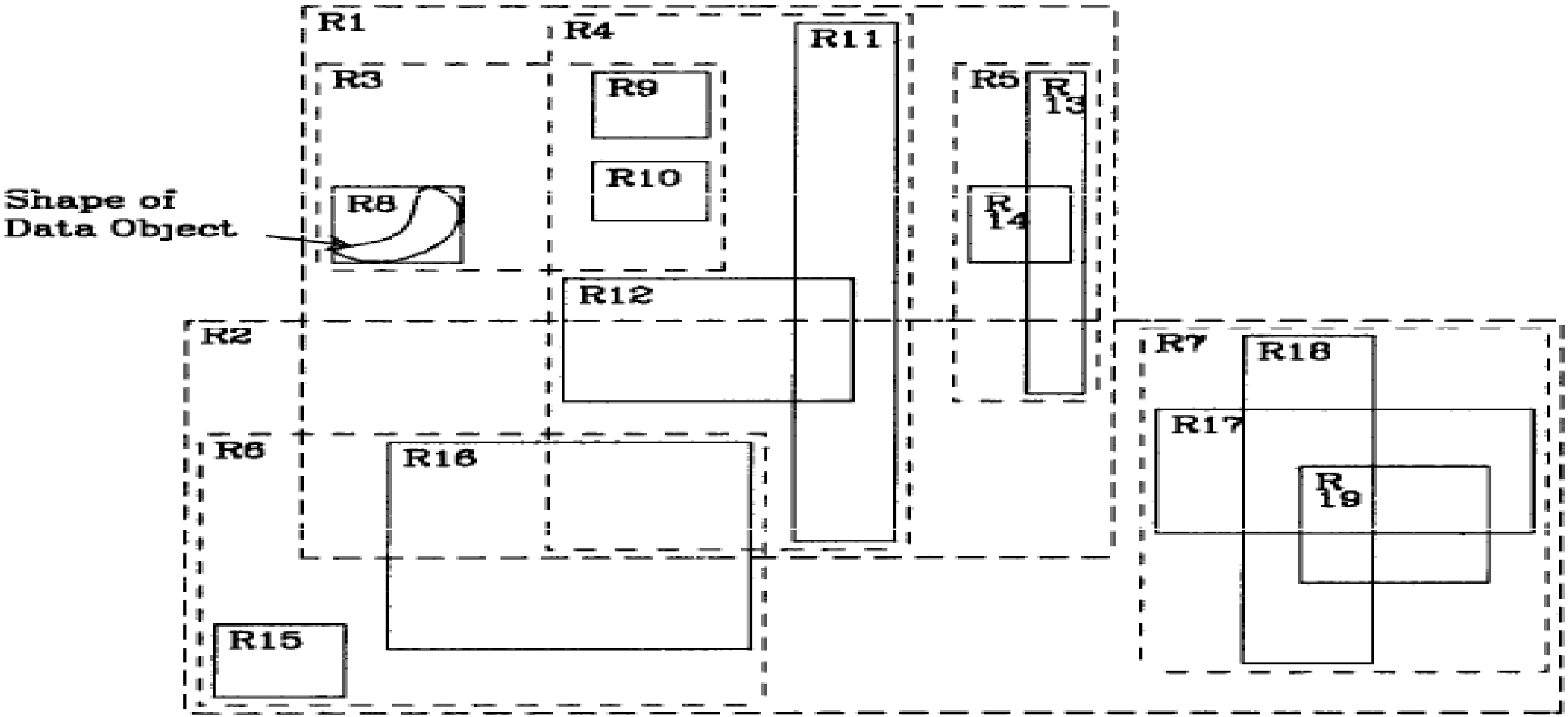
R-Tree: Non-leaf nodes

- Non-leaf nodes contain entries of the form:
- (I, child-pointer)
- Child-pointer is the address of a lower node in the R-Tree.
- I covers all rectangles in the lower node’s entries.

Example


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
To Data Tuples

Example


R1
R4
R11
R3
R9
R5
R13
R10
Shape of
Data Object
R8
R14
R12
R2
R7
R18
R17
R6
R16
R19
R15

IMPLEMENTING R-TREE

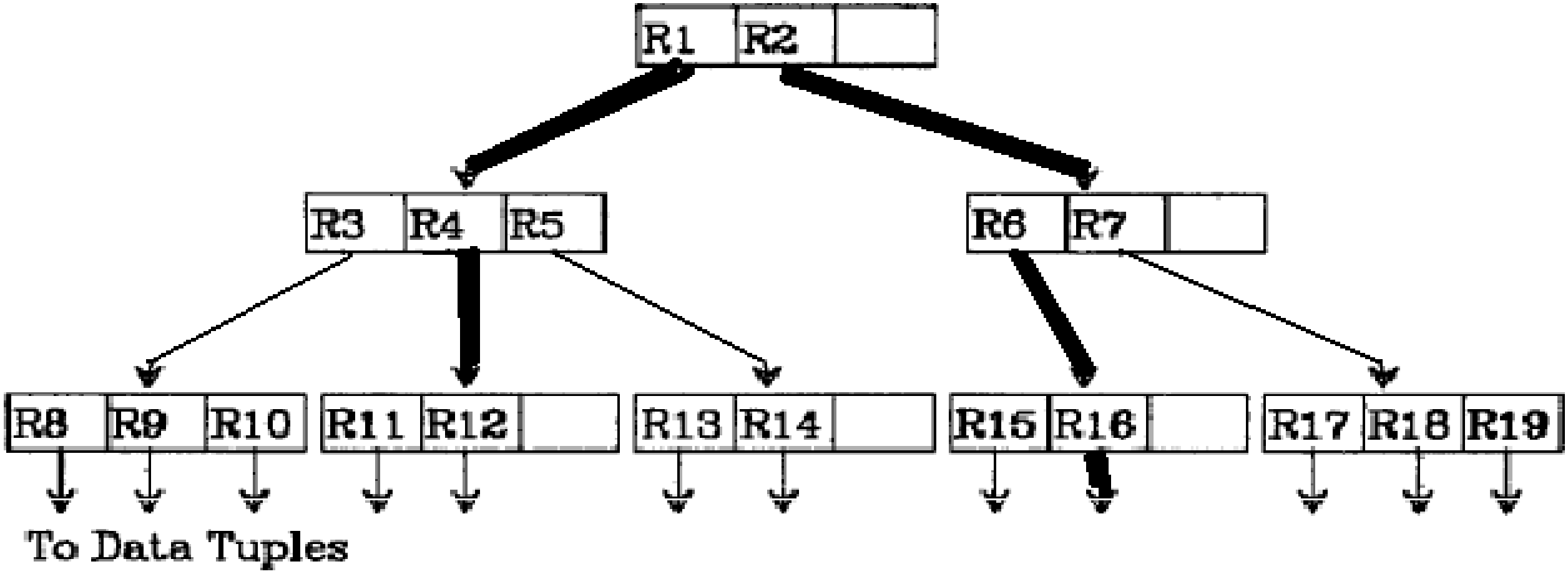
- LOOK UP
- INSERT
- DELETE

Guttman, Antonin. *R-trees: A dynamic index structure for spatial searching*. Vol. 14. No. 2. ACM, 1984.

LOOK UP :- Given an R-tree whose root node is T, find all the obj which contain the query pint/rectangle

- ALGORITHM
    - [Search subtrees].From root check if the point/rect is-present in the Entries T
    - [search leaf node]:If T is a leaf check if its present in T then T is a qualified node

S = R12 S=R16


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
To Data Tuples

INSERT:-

- ALGORITHM
  - [Find position for new record] Invoke **ChooseLeaf** to select a leaf node L in which to place E
  - [Add record to leaf node] If L has room for another entry, insert E Otherwise invoke **SplitNode** to obtain L and LL containing E and all the old entries of L
  - [Propagate changes upward] Invoke **AdjustTree** on L, also passing LL If a split was performed
  - [Grow tree taller] If node split propagation caused the root to split, create a new root whose children are the two resulting nodes

INSERT:

- **CHOOSE LEAF:-** Select a leaf node in which to place a new index entry E

    - [Initialize] Set N to be the root node
    - [Leaf check] If N is a leaf, return N.
    - [Choose subtree] If N is not a leaf, let F be the entry in N whose rectangle FI needs least enlargement to include EI Resolve ties by choosing the entry with the rectangle of smallest area
    - [Descend until a leaf is reached.] Set N to be the child node pointed to by Fp and repeat from the second step

INSERT:

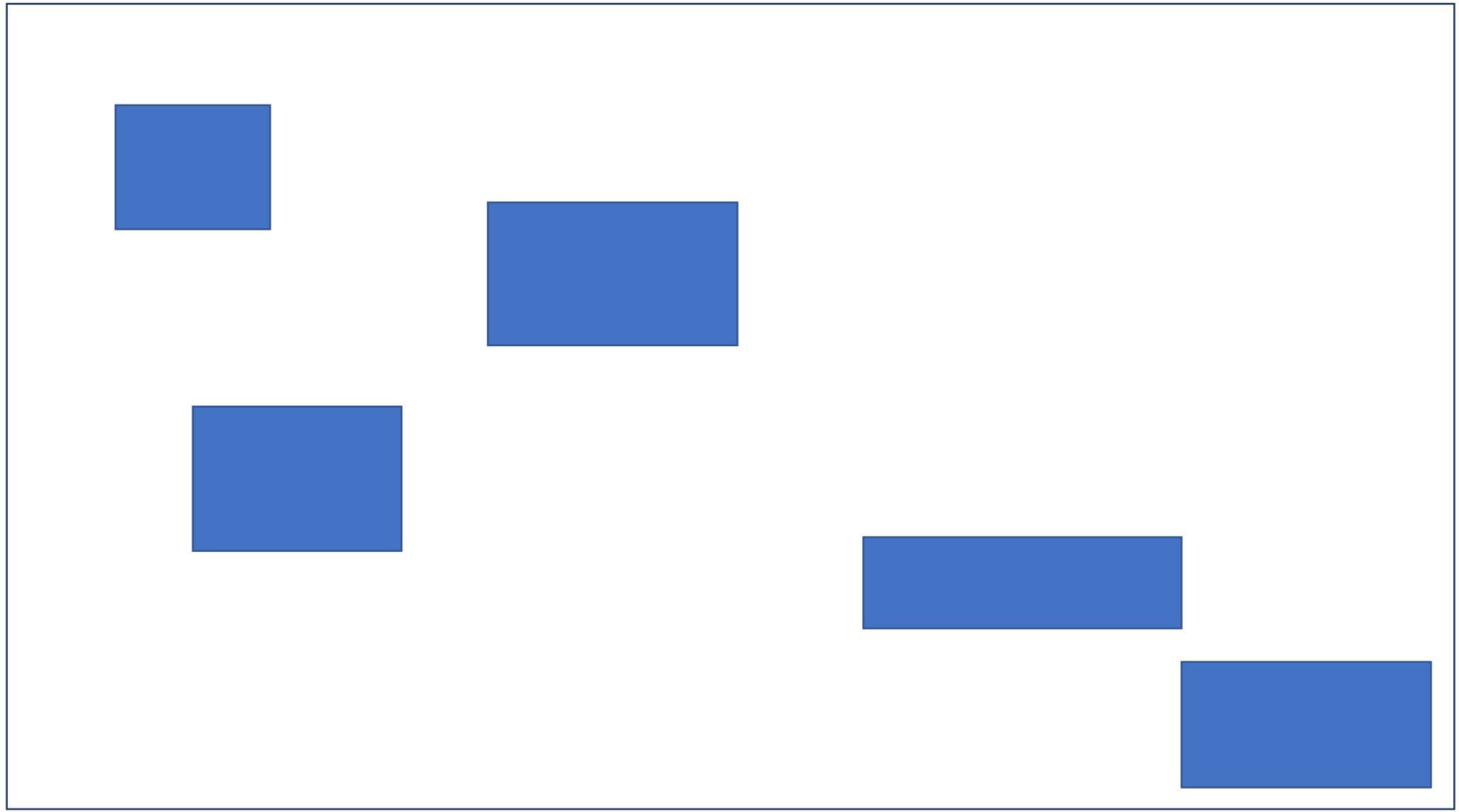
- **SPLIT-NODE:-**
- Quadratic Split:- Divide a set of M+1 index entries into two groups
  - [Pick first entry for each group] Apply Algorithm **PickSeeds** to choose two entries to be the first elements of the groups Assign each to a group
  - [Check If done] If all entries have been assigned, stop If one group has so few entries that all the rest must be assigned to it in order for it to have the minimum number m, assign them and stop
  - [Select entry to assign] Invoke Algorithm **PickNext** to choose the next entry to assign Add it to the group whose covering rectangle will have to be enlarged least to accommodate it, Resolve ties by adding the entry to the group with smaller area, then to the one with fewer entries, then to either Repeat from second step

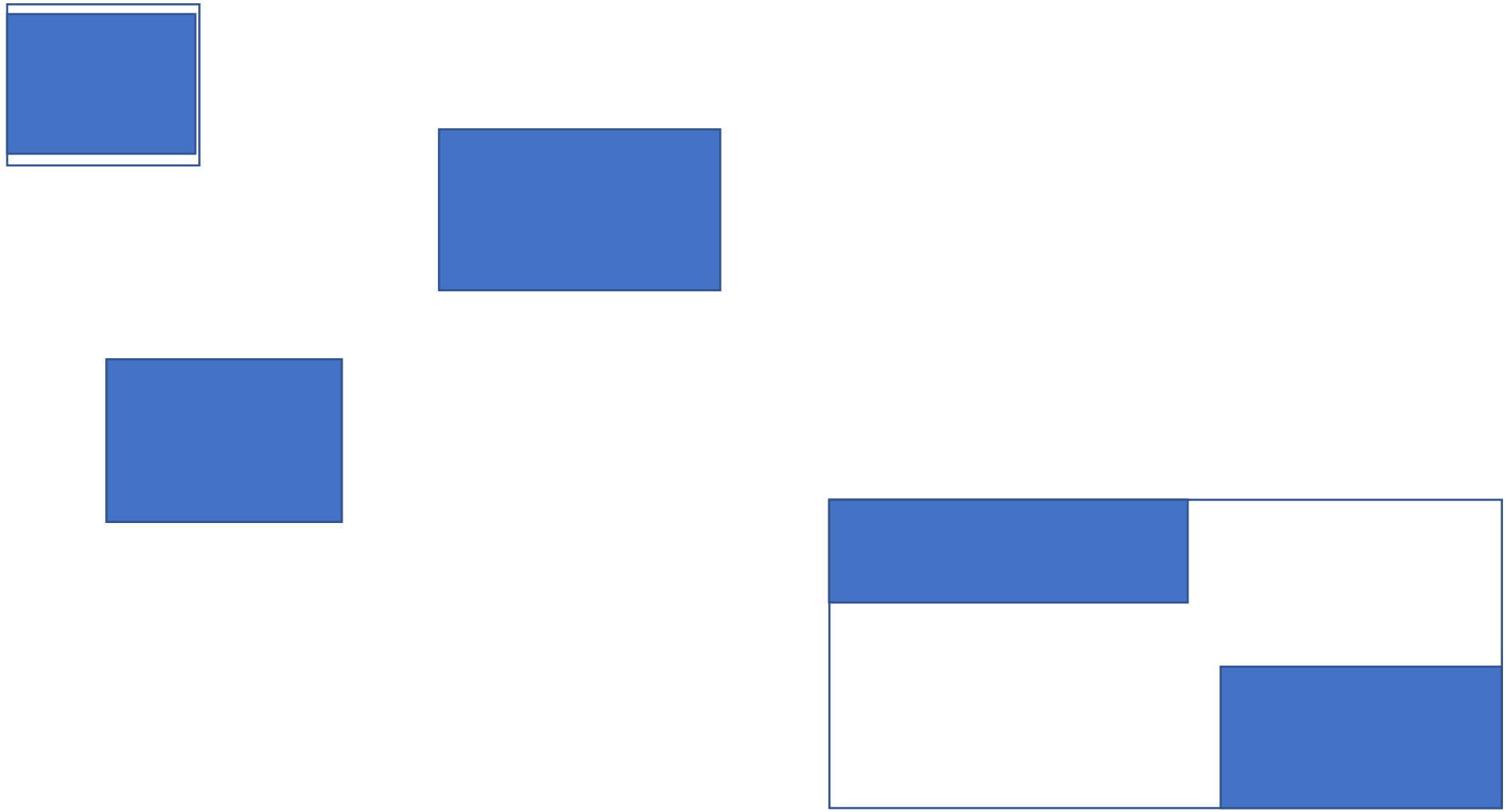
Pick seeds:-Select two entries to be the first elements of the groups

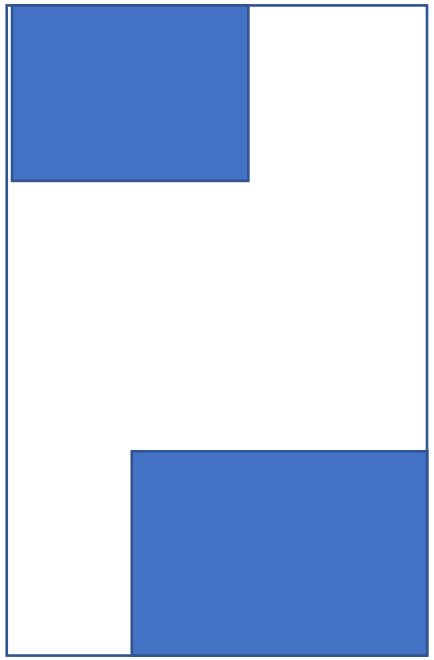
- ALGORITHM
- [Calculate inefficiency of grouping entries together] For each pair of entries E1 and E2, compose a rectangle J including E1-I and E2-I Calculate d = area(J) - area(El-I) - area(E2-I)
- [Choose the most wasteful pair] Choose the pair with the largest d

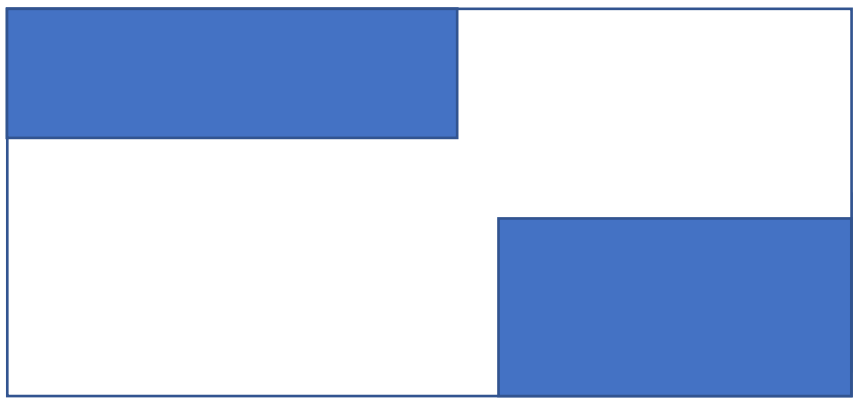
PickNext :- Select one remaining entry for classification m a group.

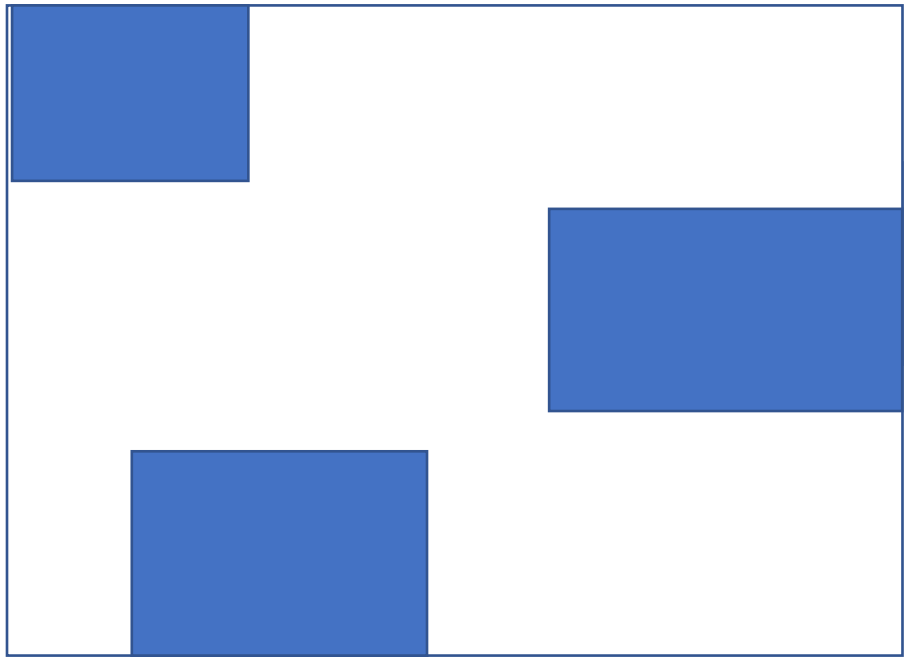
- ALGORITHM:
  - [Determine cost of putting each entry in each group] For each entry E not yet in a group, calculate d1= the area increase required in the covering rectangle of Group 1 to include EI Calculate d2 similarly for Group 2
  - [Find entry with greatest preference for one group] Choose any entry with the maximum difference between d 1 and d2

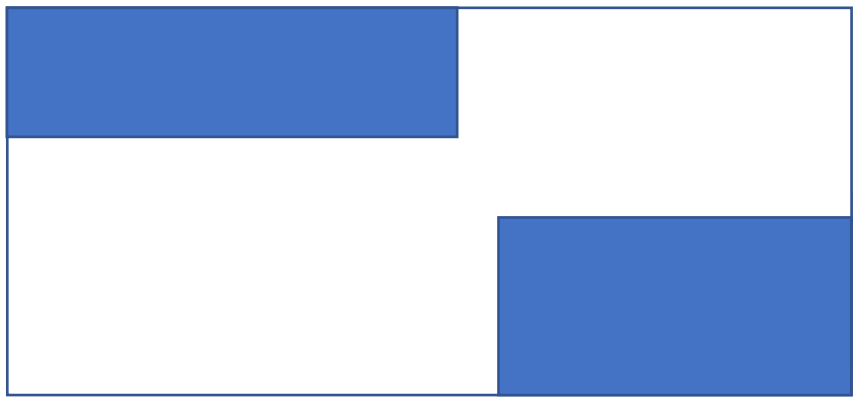








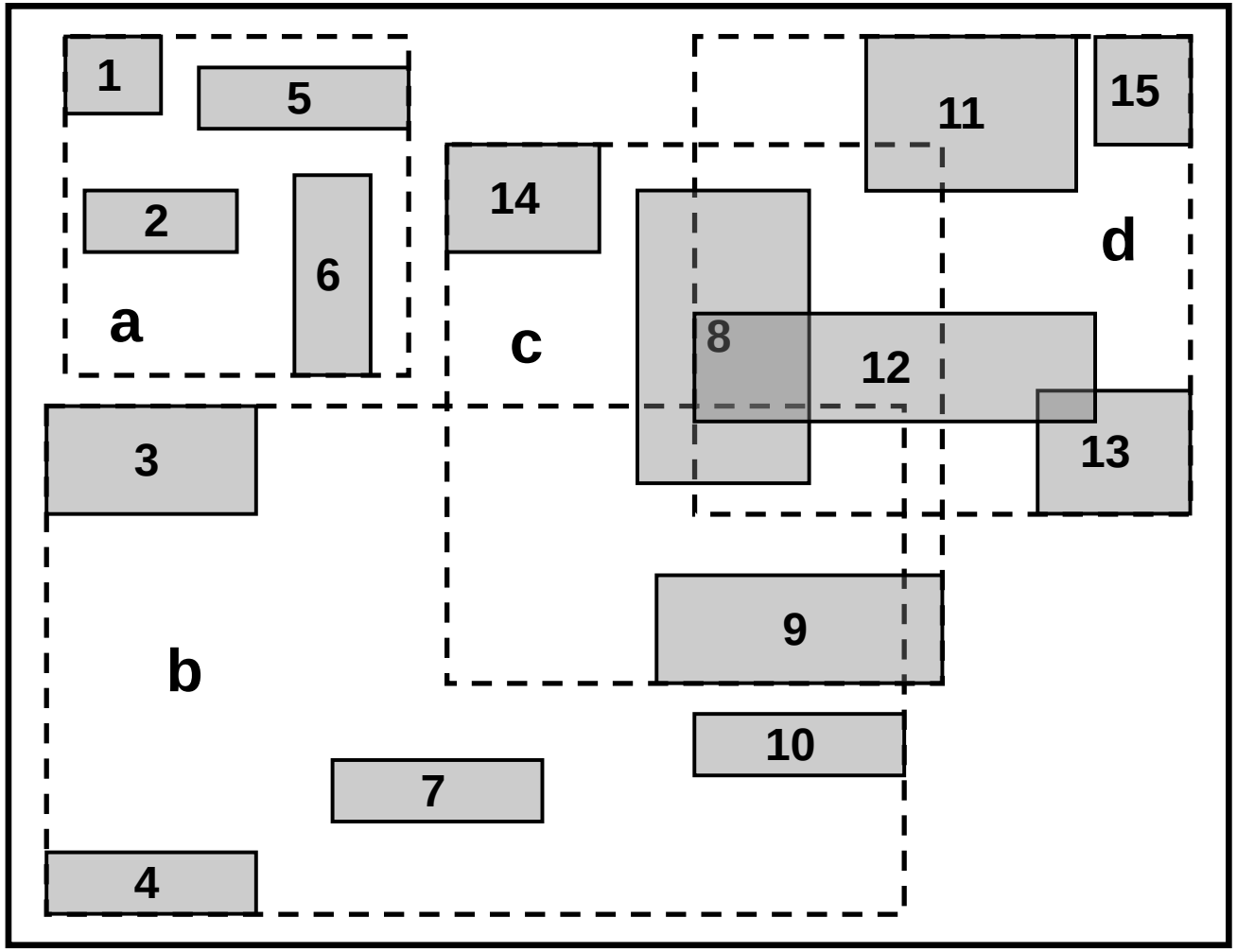




INSERT:

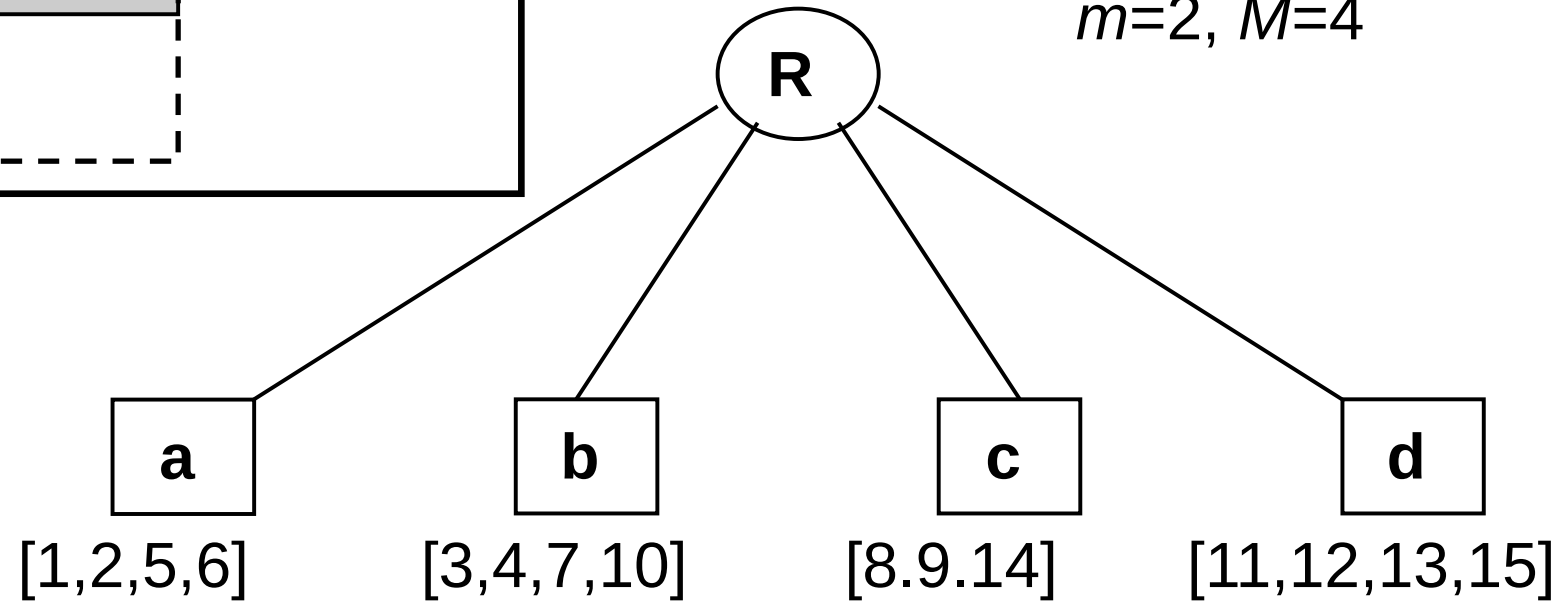
- **ADJUST-NODE:-** Ascend from a leaf node L to the root, adjusting covering rectangles and propagating node splits as necessary

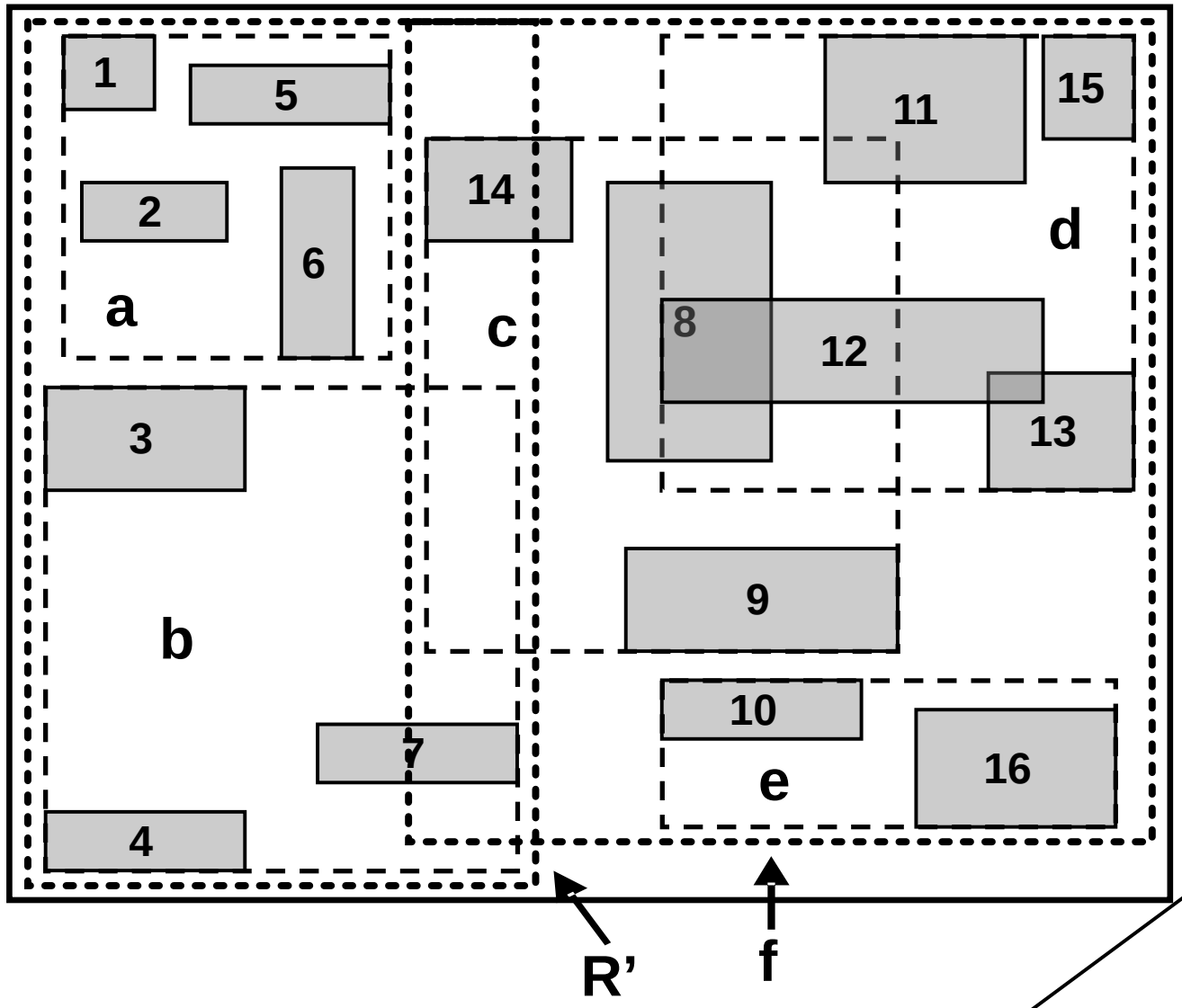
  - AT1[Initialize] Set N=L If L was split previously, set NN to be the resulting second node
  - AT2 [Check If done] If N is the root, stop
  - AT3 [Adjust covering rectangle in parent entry] Let P be the parent node of N, and let EN be N’s entry in P Adjust EN I so that it tightly encloses all entry rectangles in N.
  - AT4 [Propagate node split upward] If N has a partner NN resulting from an earlier split, create a new entry ENN with ENNp pointing to NN and ENNI enclosing all rectangles in NN Add ENN to P If there is room Otherwise, invoke SplitNode to produce P and PP containing ENN and all P’s old entrees
  - AT5 [Move up to next level.] Set N=P and set NN=PP If a split occurred, Repeat from AT2.



Insert object 15

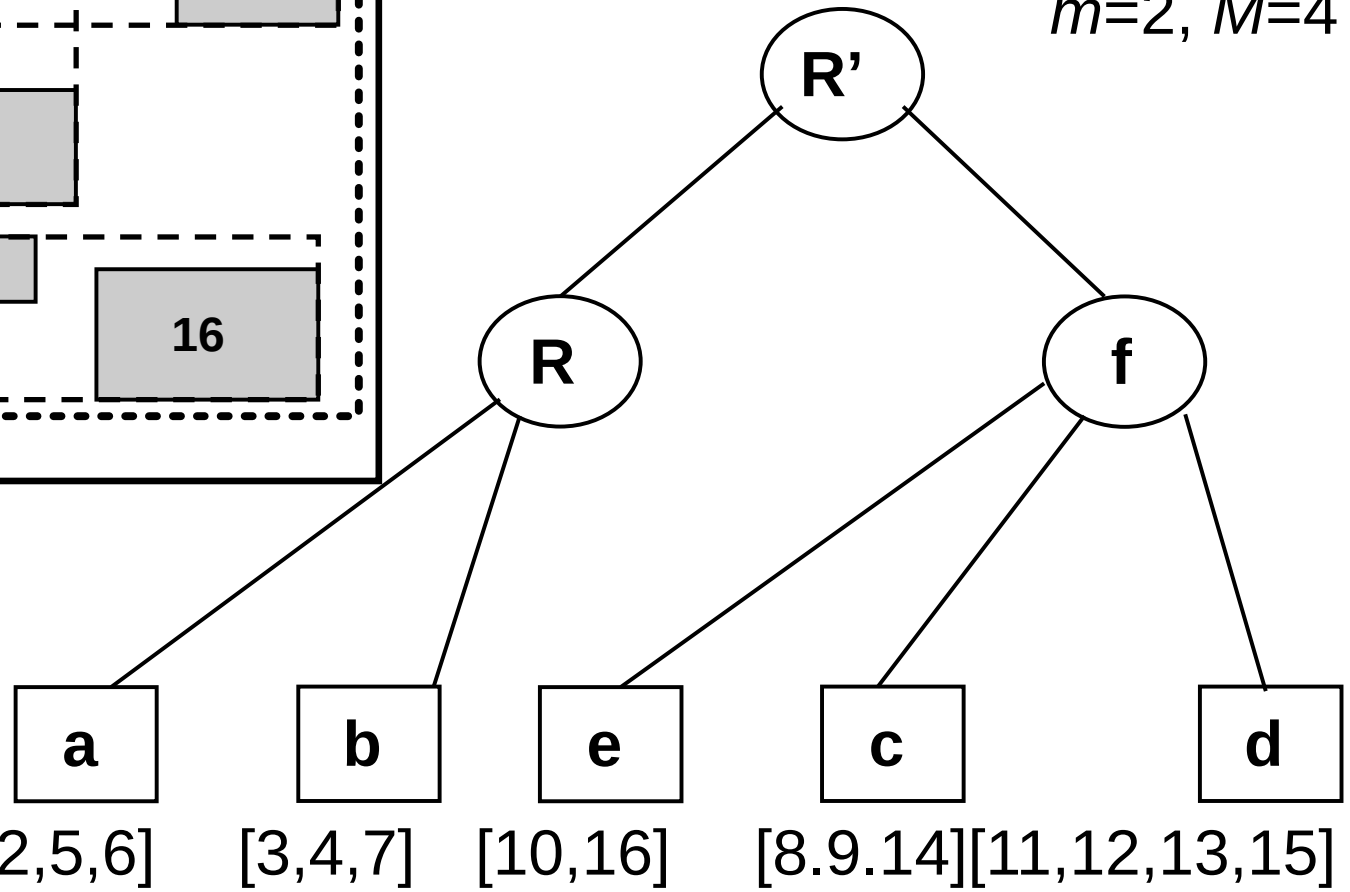
m=2, *M*=4





Insert object 16

m=2, *M*=4



DELETE:-Remove index record E from an R-tree

- ALGORITHM
  - [Find node containing record] Invoke **FindLeaf** to Locate the leaf node L containing E Stop if the record was not found.
  - [Delete record.] Remove E from L
  - [Propagate changes] **CondenseTree**, passing L.
  - [Shorten tree.] If the only one child after been adjusted, make the child the new root

DELETE:-

- FindLeaf:-Given an R-tree whose root node is T, find the leaf node containing the index entry E.
- ALGORITHM:-

    - [Search subtrees] If T is not a leaf, check each entry F in T to determine if F-I overlaps E-I For each such entry invoke FindLeaf on the tree whose root is pomoted to by Fp until E is found or all entries have been checked
    - [Search leaf node for record] If T is a leaf, check each entry to see if it matches E If E is found return T

CONDENSE TREE:-Given a leaf node L from which an entry has been deleted, eliminate the node If it has too few entries and relocate its entries Propagate node elimination upward as necessary. Adjust all covering rectangles on the path to the root, making them smaller If possible

1][Initialize] Set N=L Set Q, the set of eliminated nodes, to be empty

2][Find parent entry.] If N 1s the root, go to 6 , Otherwise let P be the parent of N, and let EN be N’s entry in P

3][Eliminate under-full node.] If N has fewer than 2 entries, delete EN from P and add N to set Q.

4][Adjust covering rectangle] If N has not been eliminated, adjust EN-I to tightly contain all entries in N

5][Move up one level m tree] Set N=P and repeat from 2-6

6][Re-insert orphaned entries] Re-insert all entries of nodes in set Q Entries from eliminated leaf nodes are re-inserted in tree leaves as described in Algorithm **Insert,** but entries from higher-level nodes must be placed higher in the tree, so that leaves of their dependent subtrees will be on the same level as leaves of the main tree

IMPLEMENTING KNN USING R-TREE

- KNN-GIVES YOU THE N NEAREST NEIGHBOURS
- USING KNN TO IMPLEMENT

    - a) NEAREST
    - b) NEARESTKEY
    - c)WHERE_AM_I

KNN:-

- ALGORITHM:
  - Calculate the MIN distance from the point to BB in that node
  - If Min is less than MaxMin ,Enter that node else check for the next BB
  - Calculate the distance of the entries inside that BB and and if found to be less than MaxMin then replace MaxMin’th neighbour with the current one
  - Update MaxMin to be furthest distance of those k neighbours

Implementation