EXERCISE 9

Multimedia Databases (Indexing)

Prof. Dr. Harald Kosch/ Prof. Dr. Mario Döller
Tutor: Kanishka Ghosh Dastidar, Dr. Alaa Alhamzeh

Task 1: Index structures in general

- 1. What are the general functions of index structures in databases?
  - increase the execution speed of retrieval queries by enabling fast access to given database objects.
    - improving the query optimizer ability to access the data quickly.

Drawbacks of using many indexes:

- They consume additional storage space.
- They slow down the operations of inserting, deleting and updating data, since the indexes have to be recomputed at each operation.

Task 1: Index structures in general

- Which other requirements does CBIR have on index structures?
  - Methods for reduction of dimensionality
  - Specific indexing data structures.

Task 1: Index structures in general

- Among the frequently-used index structures, we list the B-tree, Hash, K-d tree and Point Quadtree. For each one, explain its structure and its functioning.

  - B-trees
  - Hashes
  - K-d trees
  - Point-Quadtrees

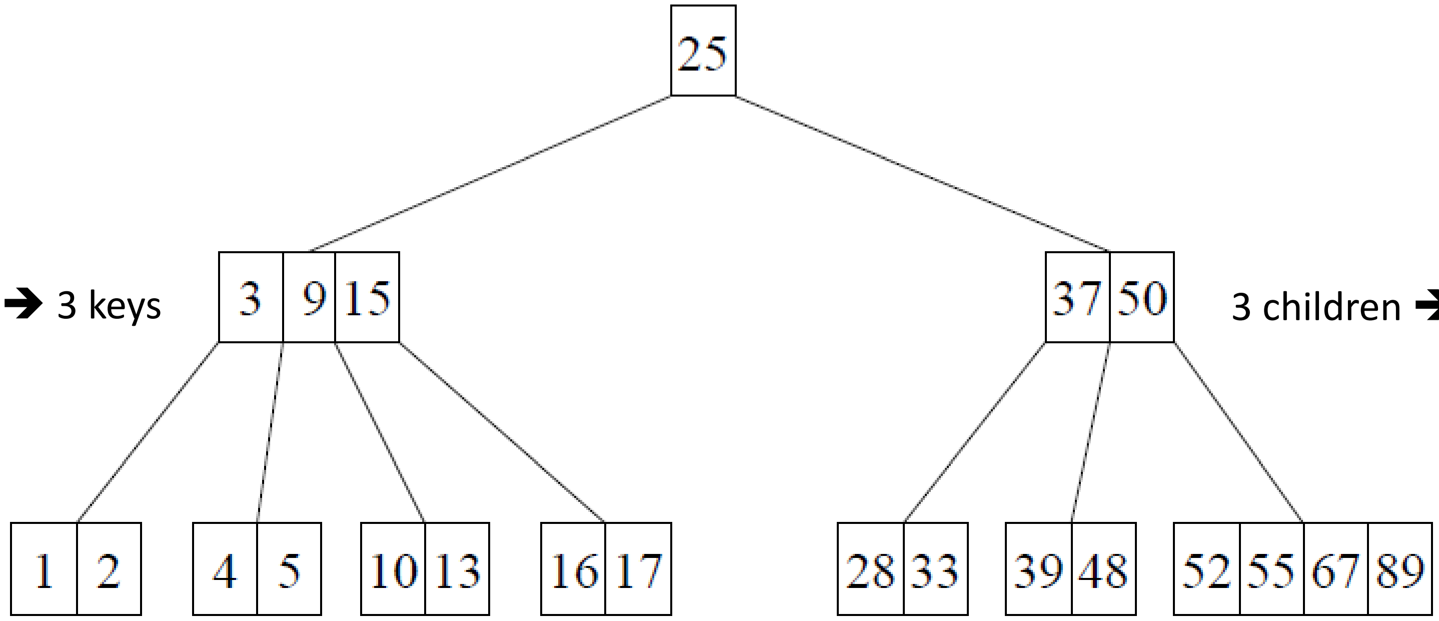
Task 1: Index structures in general

- **B-tree: (Balanced tree)**
- According to Knuth's definition [Knuth 1998], a B-tree of order *m* is a tree, which satisfies the following properties:
  - Every node has at most m children
  - The root has at least 2 children if it is not a leaf node, and it has at most m children.
  - Each non-leaf node (except root) has at least $\lceil m/2 \rceil$ children and at most m children.
  - A non-leaf node with m/2<=k<=m children contains k-1 keys.
  - Each non-leaf node contains at most a number of (m-1) keys and m pointers to children nodes. Each leaf contains between [m/2]-1 and m-1 keys.
  - All leaves are at the same level of the tree, and carry information.

Task 1: Index structures in general

- **Example of B-tree of order 5**

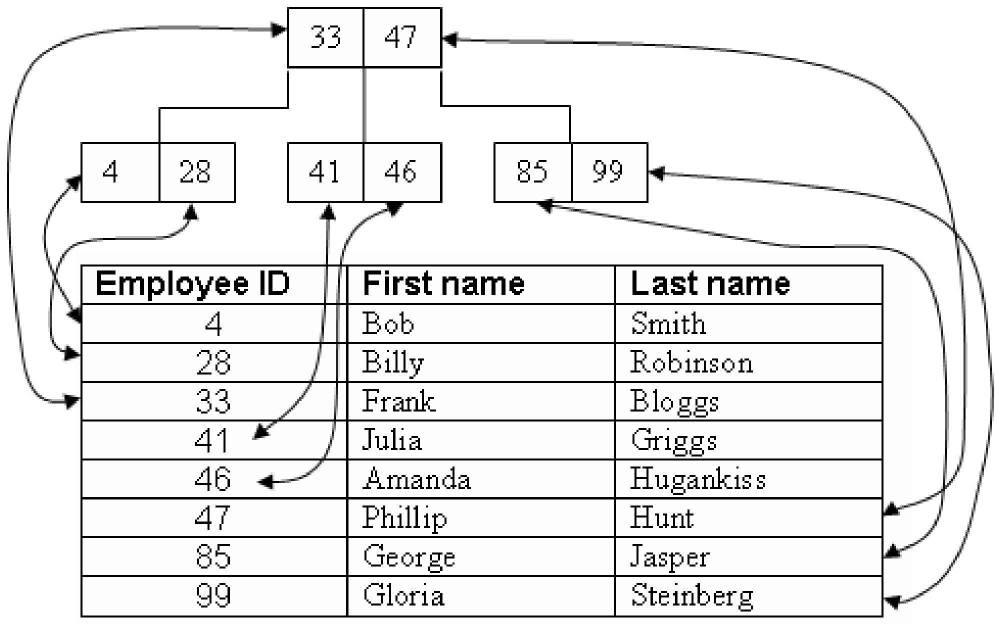
4 children ➔ 3 keys



3 children ➔ 2 keys

Task 1: Index structures in general

- **Example of B-tree**



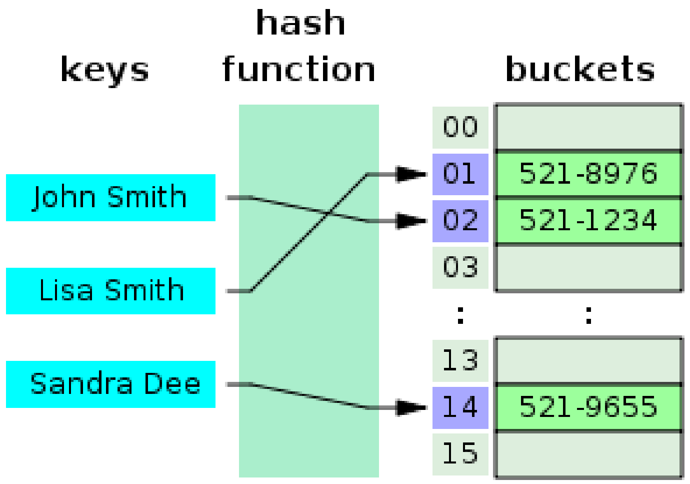
Employee ID	First name	Last name
4	Bob	Smith
28	Billy	Robinson
33	Frank	Bloggs
41	Julia	Griggs
46	Amanda	Hugankiss
47	Phillip	Hunt
85	George	Jasper
99	Gloria	Steinberg

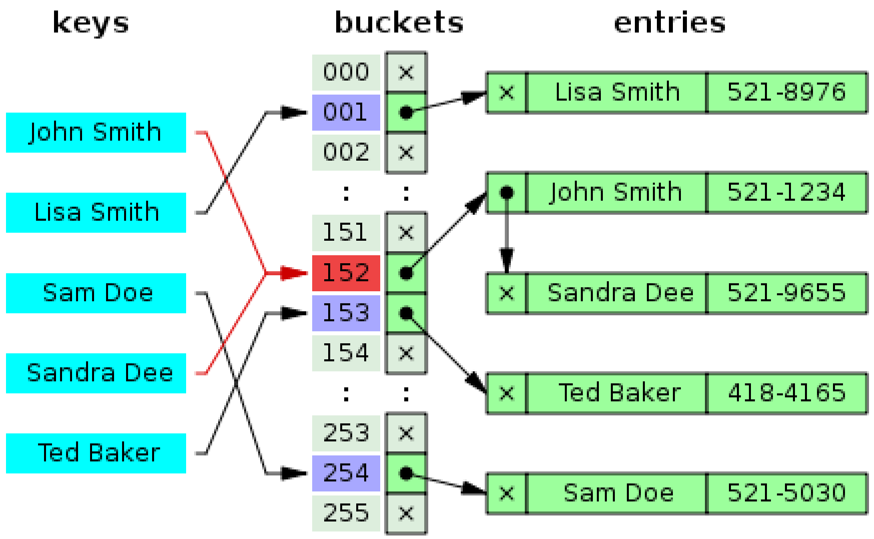
Task 1: Index structures in general

- **B-tree**
  - The insert and delete algorithms guarantee that the tree remains balanced.
  - Search, deletion and insertion all run in logarithmic time.

Task 1: Index structures in general

- **Hashes**

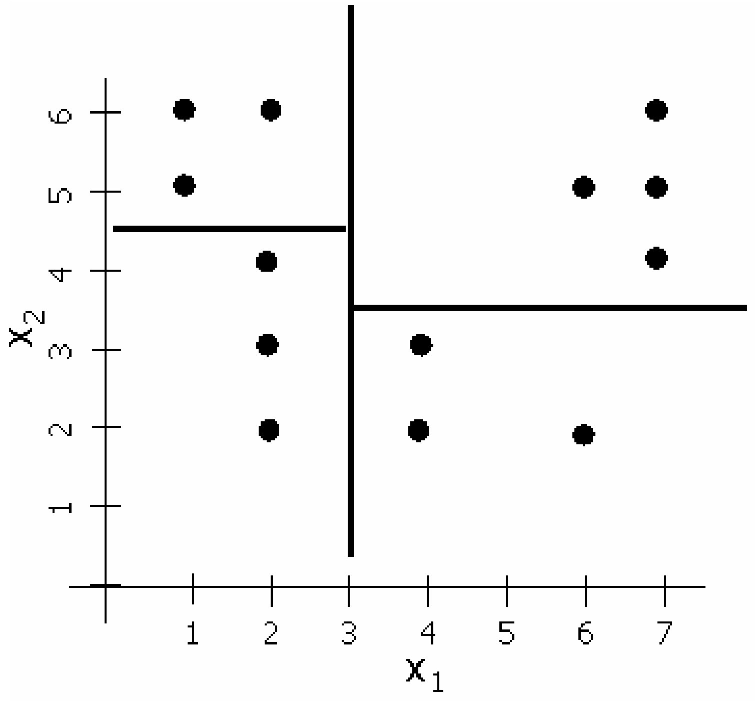


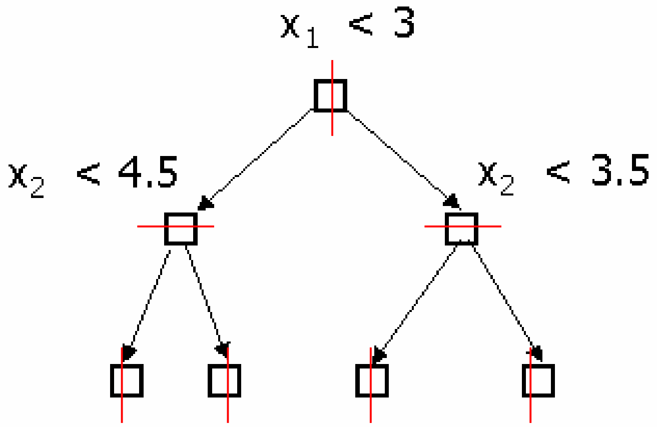


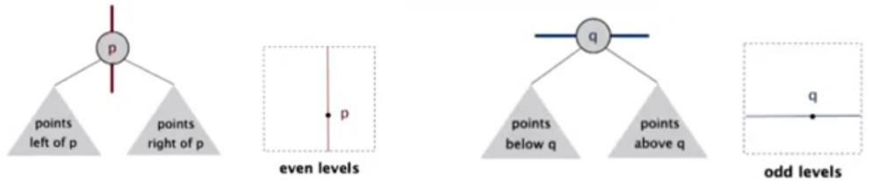
Hash collision resolved by separate chaining.

Task 1: Index structures in general

- **Kd-tree: 2D example**



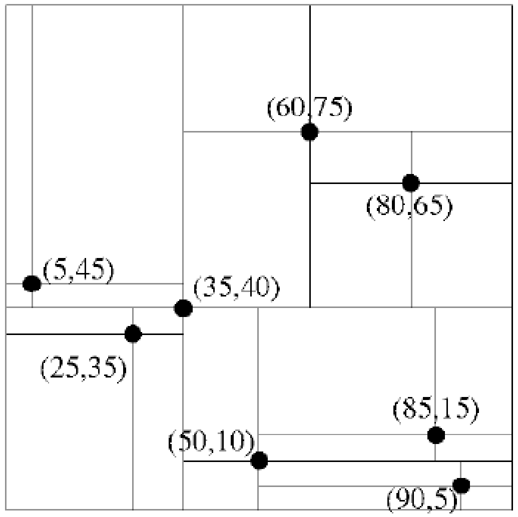


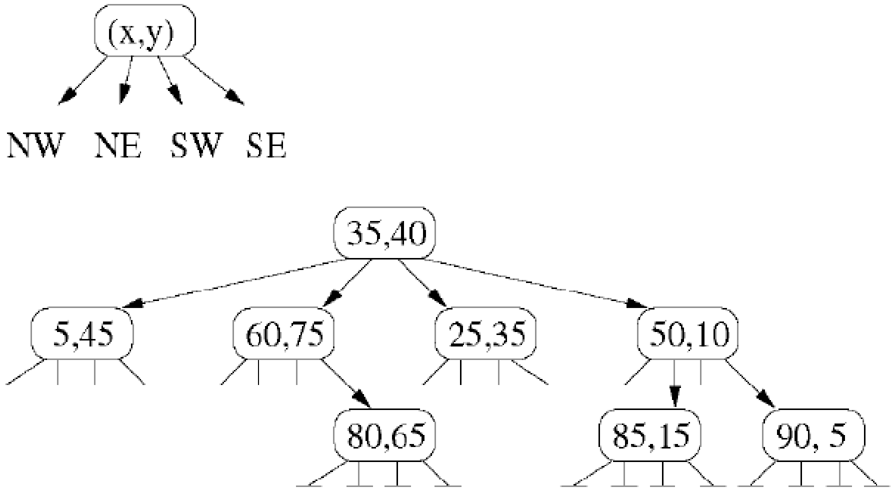


Task 1: Index structures in general

- **Point-Quad tree**

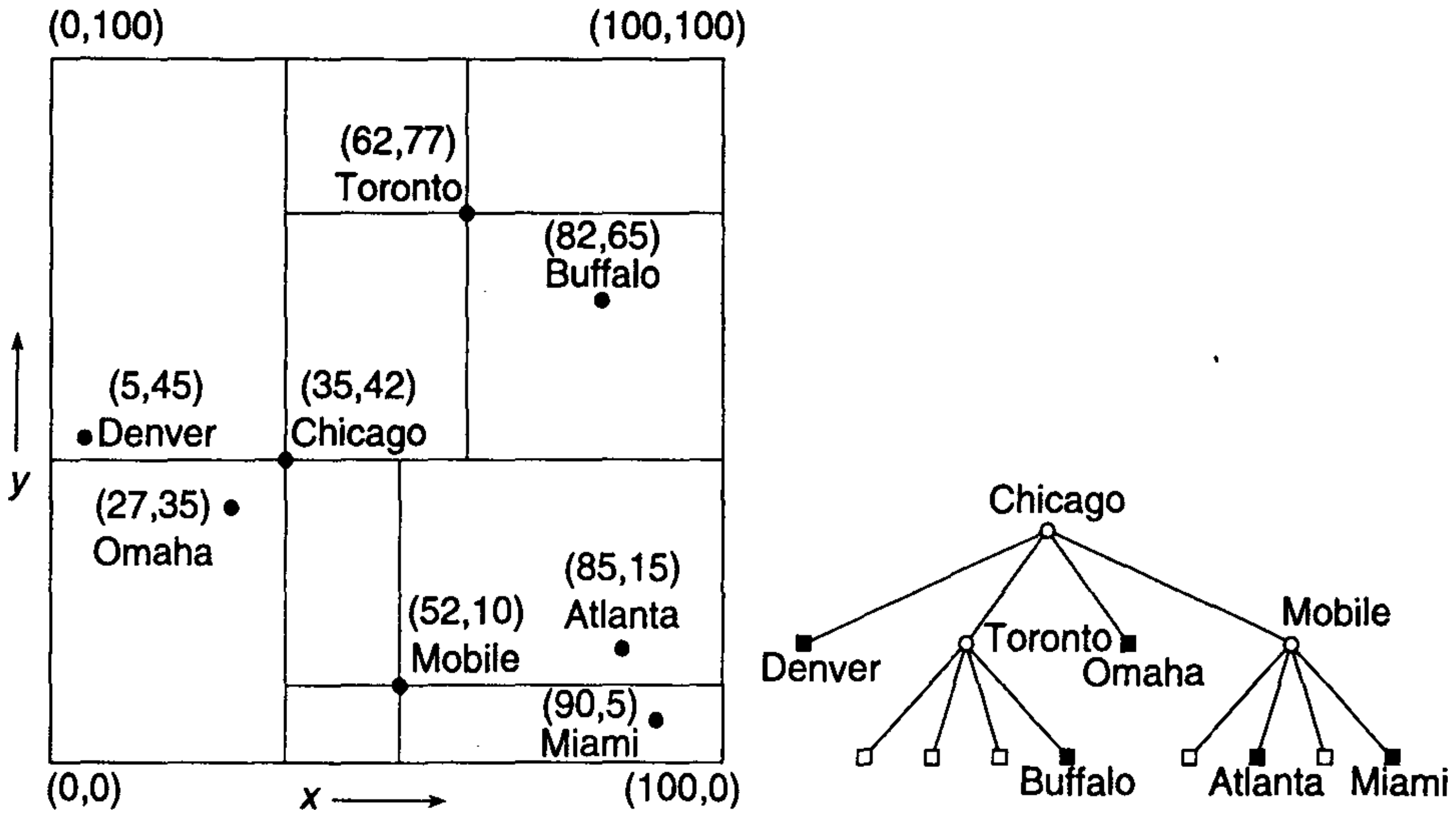
$$(35, 40), (50, 10), (60, 75), (80, 65), (85, 15), (5, 45), (25, 35), (90, 5).$$





Task 1: Index structures in general

- **Point-Quad tree**



Task 2: Index structures for n-dimensional data: R-tree

- 1. Describe the most important characteristics and the structure of an R-tree.
- 2. Explain the functioning (Inserting, Deleting) of an R-tree-index. Illustrate the single steps with the help of examples.

The R–tree

Definition, Properties, Methods

Adapted from Albuni et al., Universität Wien

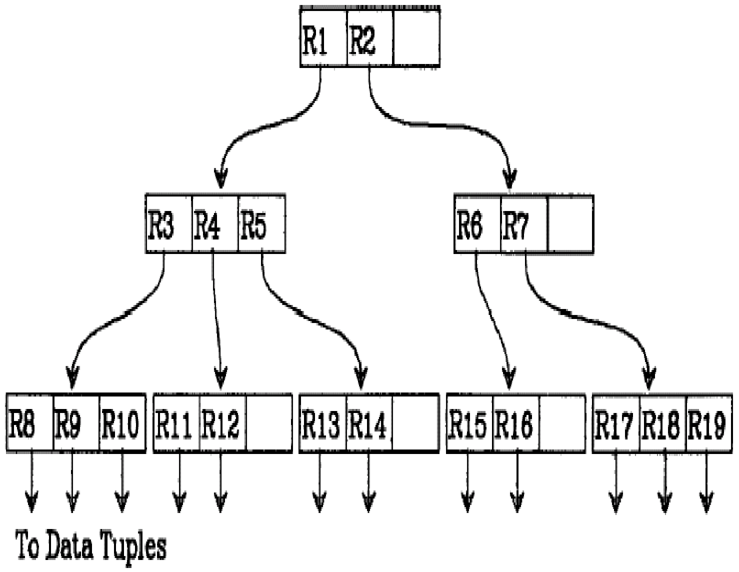
Definition

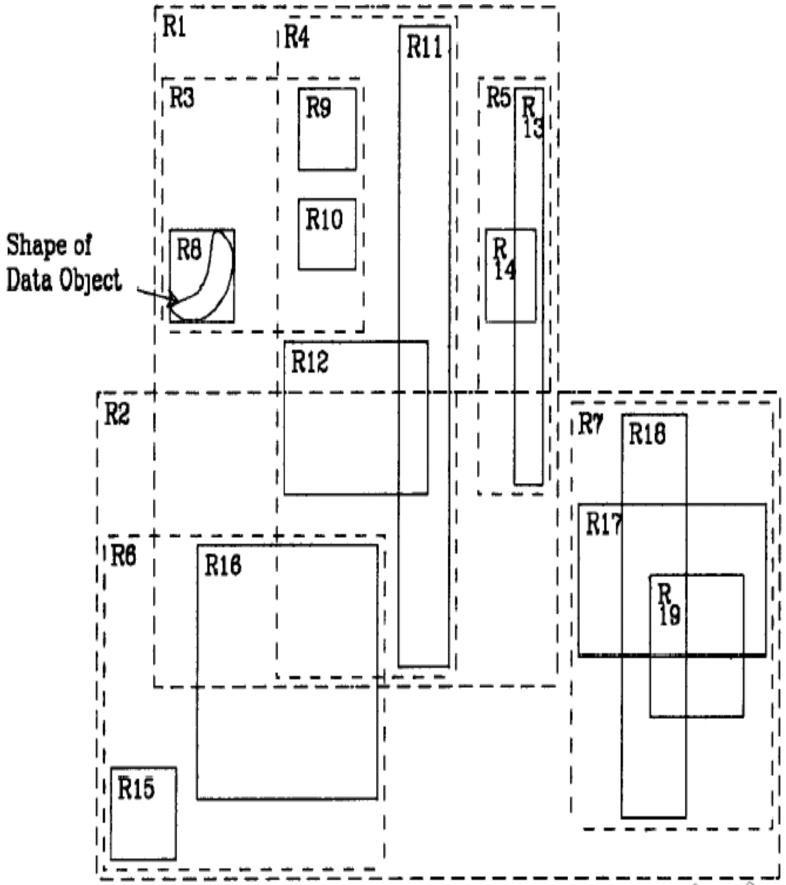
- An R-tree is a height-balanced tree similar to a B-tree but with only index records in its leaf nodes containing pointers to data objects.

(Guttman 1984)

- Each node is described by its MBB or also known as MBR (Minimum Bounding Rectangle) .

Example



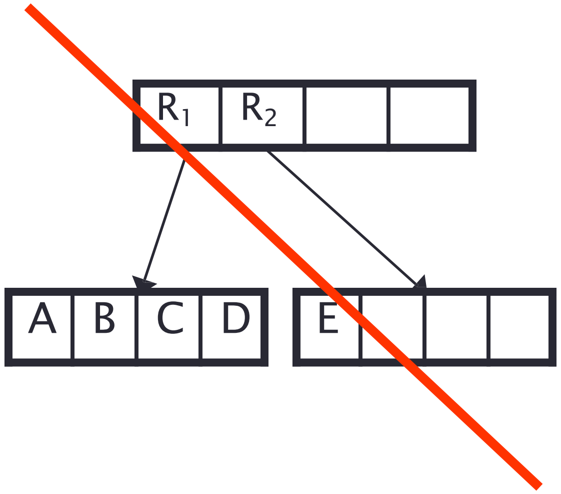


⬚ Internal node

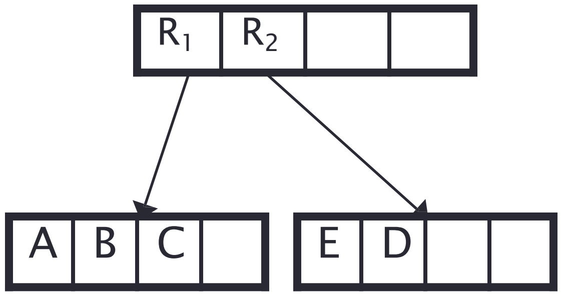
▭ Leaf

Properties

- Let M be the maximum number of entries that will fit in one node and let m≤M/2 be a parameter specifying the minimum number of entries in a node. The R-tree must satisfy the following properties:
- P1) Every **leaf node** contains **‘between m and M index records** unless it is the root.

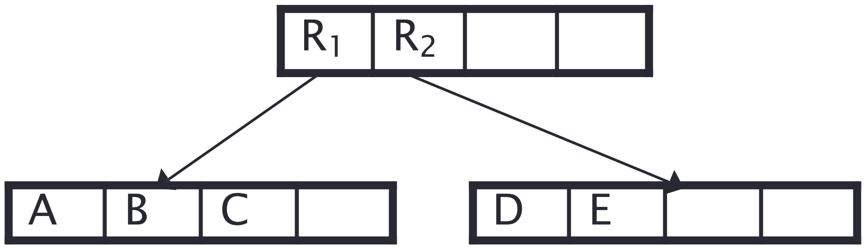


$M=4$
$m=2$
$m \leq M/2$



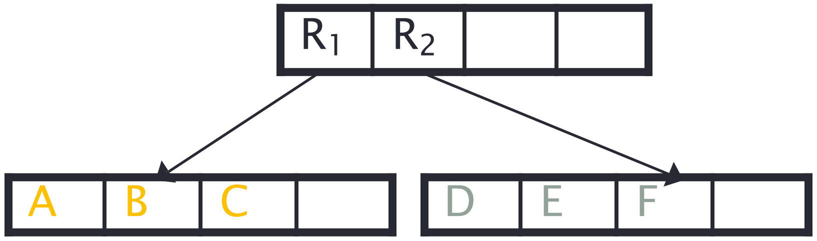
Properties

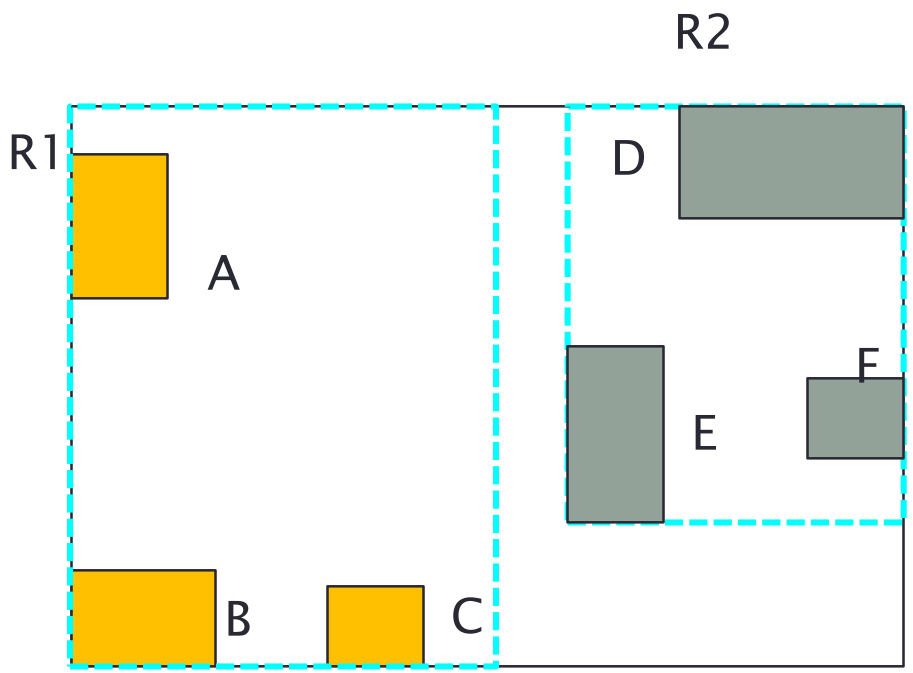
- P2) For each entry **(I, objectpointer)** in a **leaf**, I is the MBB (Minimum Bounding Box). of the object contained in the index.



Properties

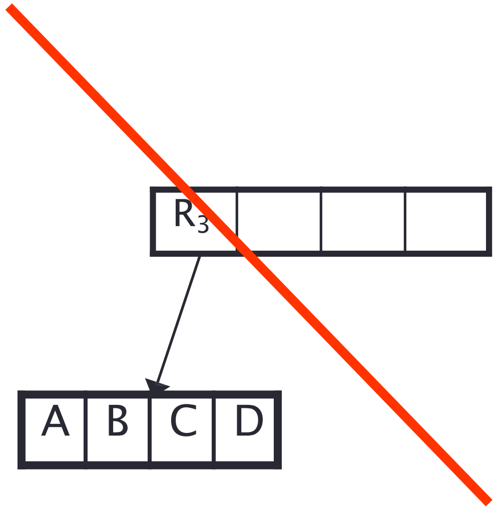
- P3) For each entry E **(I, childpointer)** in a **non-leaf,** I is the MBB of the MBBs of its children.

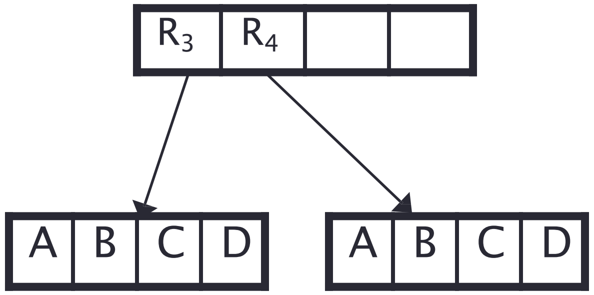




Properties

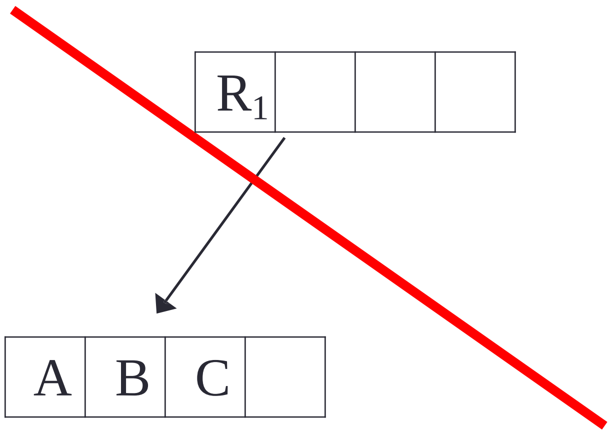
- P4) Each **non-leaf** that is not the root **has between m and M children**.

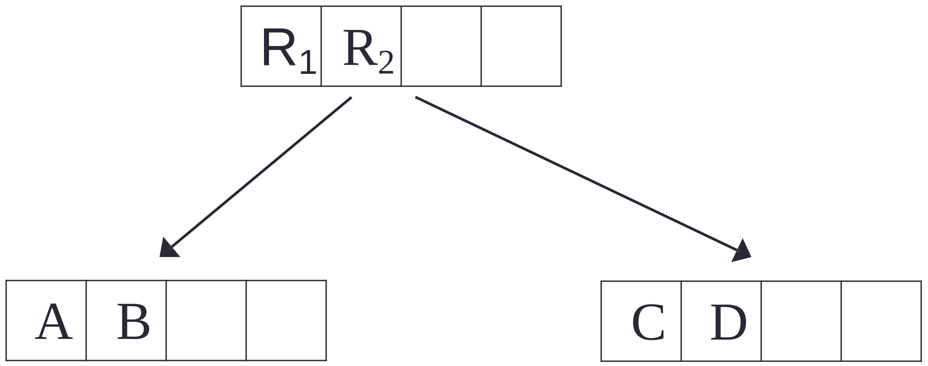




Properties

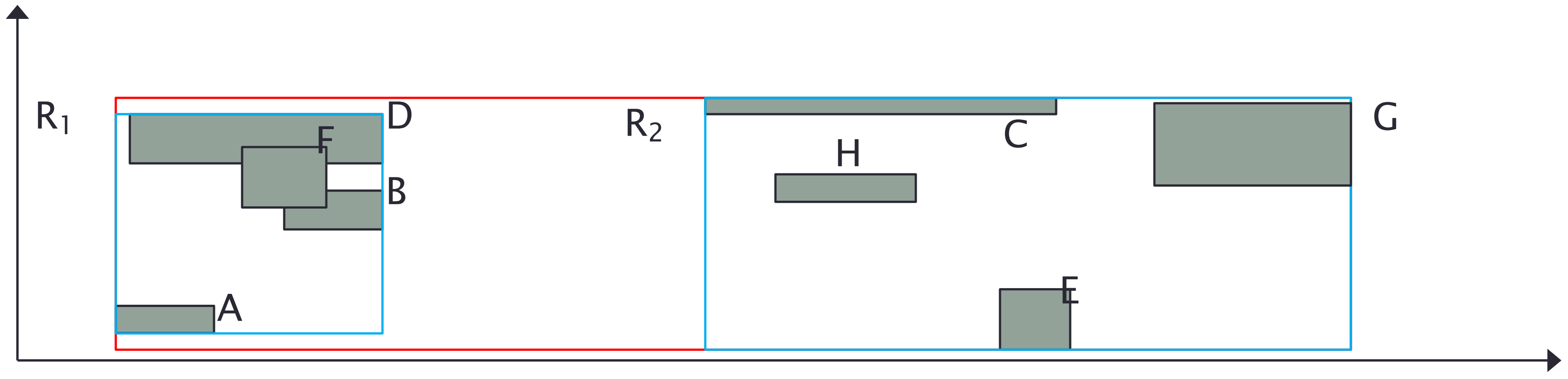
- P5) The **root** node has **at least two children** unless it is a leaf





Properties

- P6) All leaves appear on the same level



Task 2: Index structures for n-dimensional data: R-tree

- **Insertion:** General Algorithm to insert an entry E in an R-Tree
  - I1 ChooseLeaf [choose the leaf that should receive the new entry] -> Leaf L
  - I2 Add E in L (if leaf node is full split L in L and LL)
  - I3 AdjustTree [go up from leaf to root, updates the MBBs and make further splits if needed.]
    - May have to **enlarge** bounding boxes on path to leaf
    - May have to **rebalance** the tree

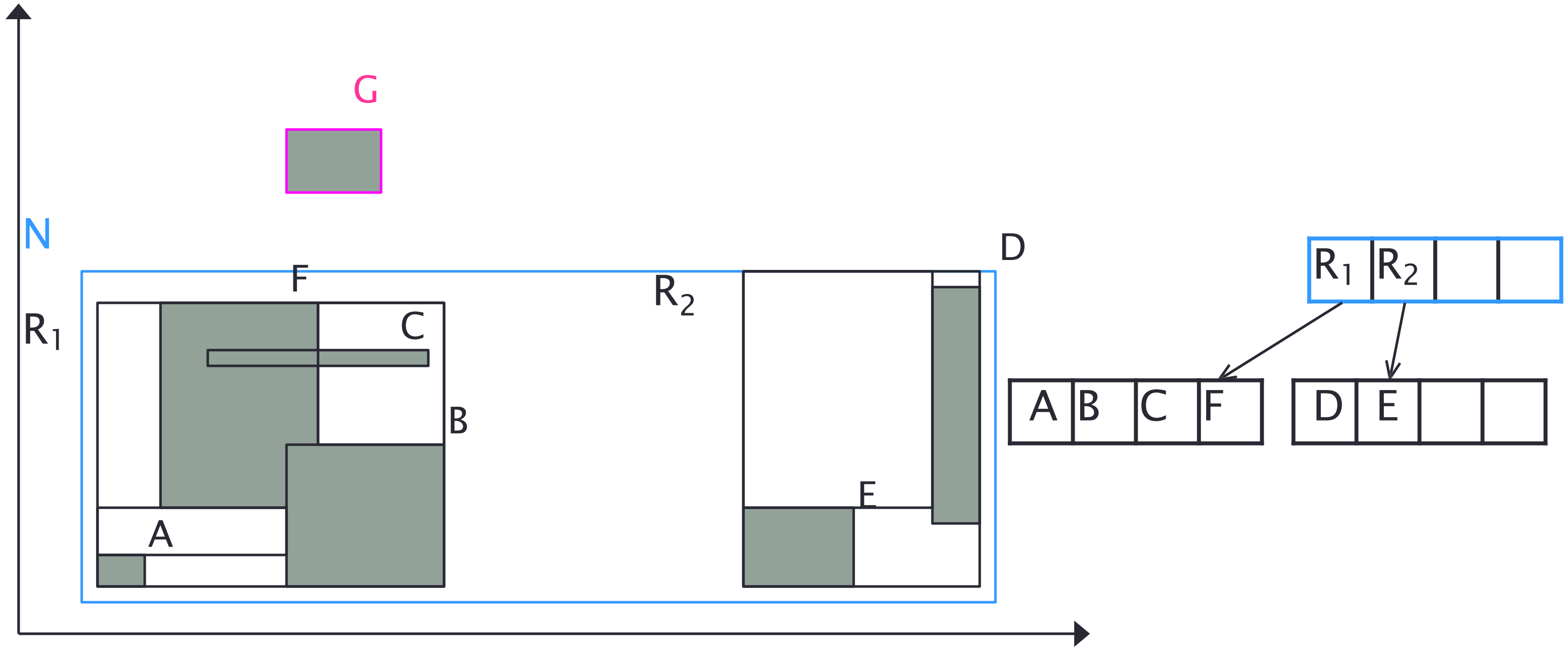
Insertion

Algorithm ChooseLeaf

To choose the leaf that should receive the new entry E.

CL1 [Initialisation]: Let N be the root.

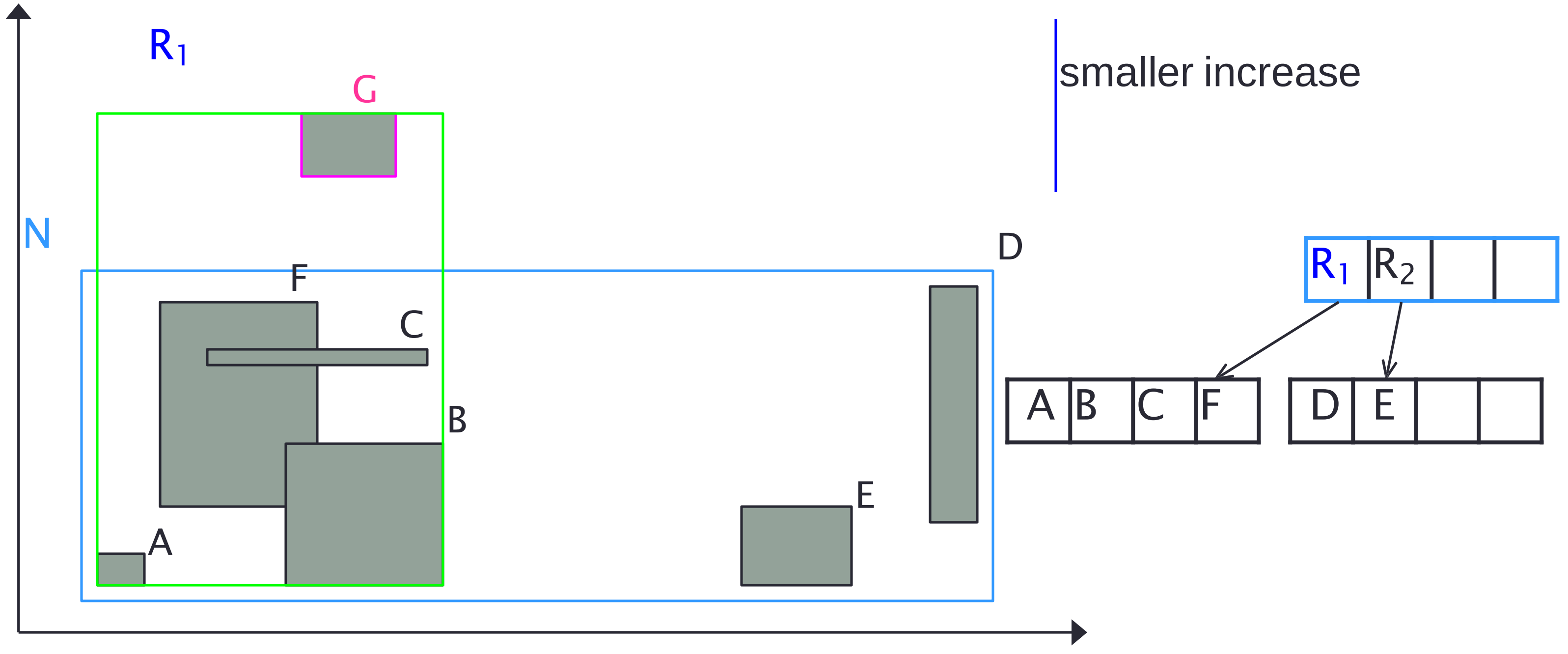
CL2 [Check leaf] :If N is a leaf, return N.



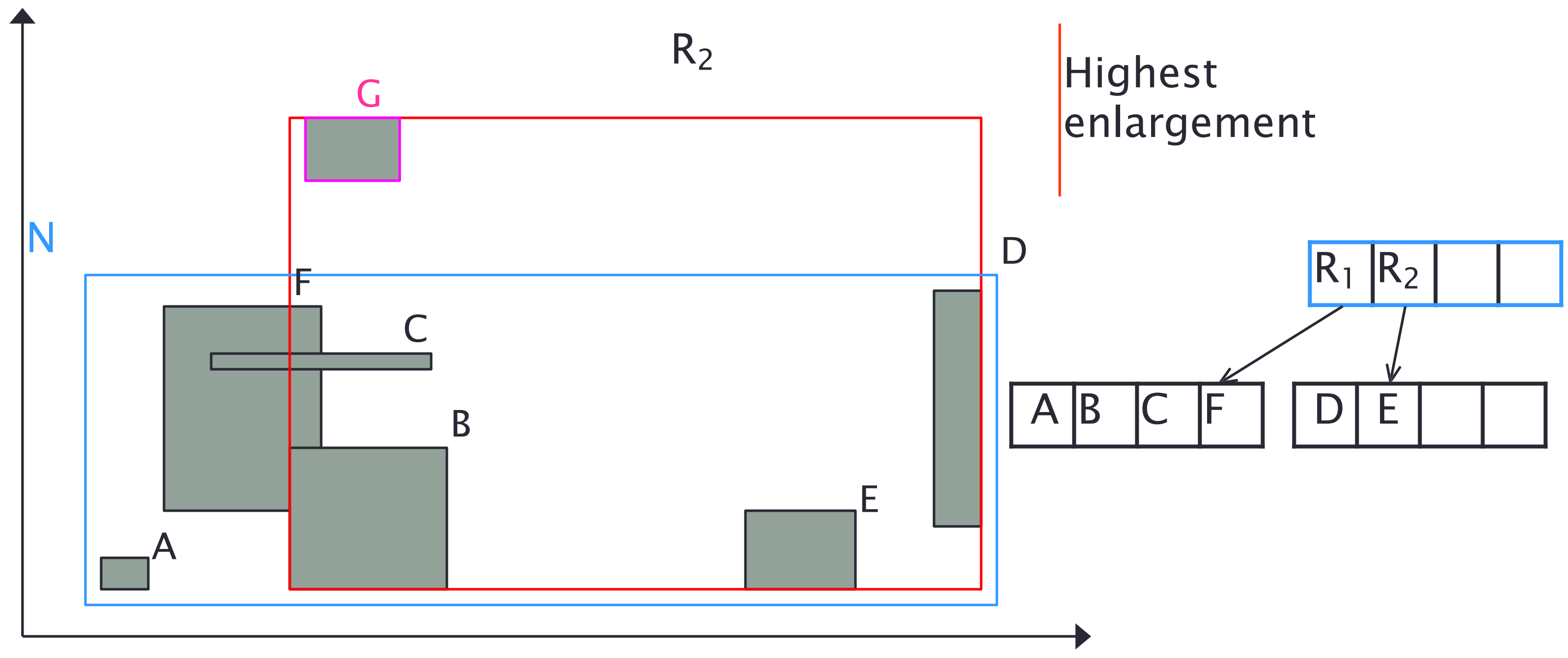
Insertion

CL3 [Choose subtree] If N is not a leaf, let F be the entry in N, which MBB F.I the smallest size increase needs in order to integrate E.I . In case of equality choose the one with the smallest volume.

CL4 [Go downwards until reaching the leaf] Let N be the child pointed to by F.p and continue from CL2.



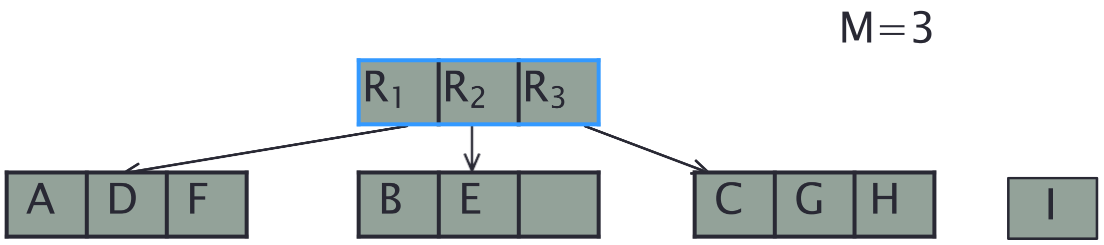
Insertion



Insertion

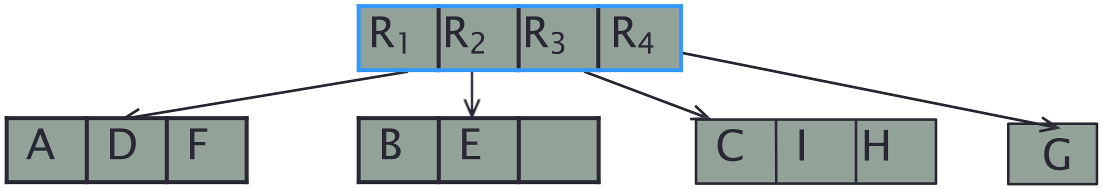
Algorithm AdjustTree: Example with recursive splits

New node I is to be inserted in R3.



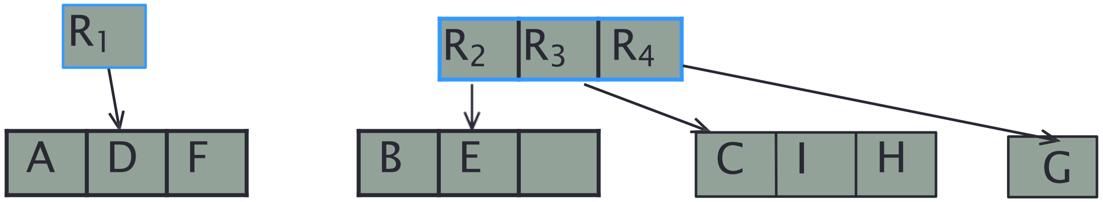
Insertion

R3 is split in R3 and R4; the root has to be split



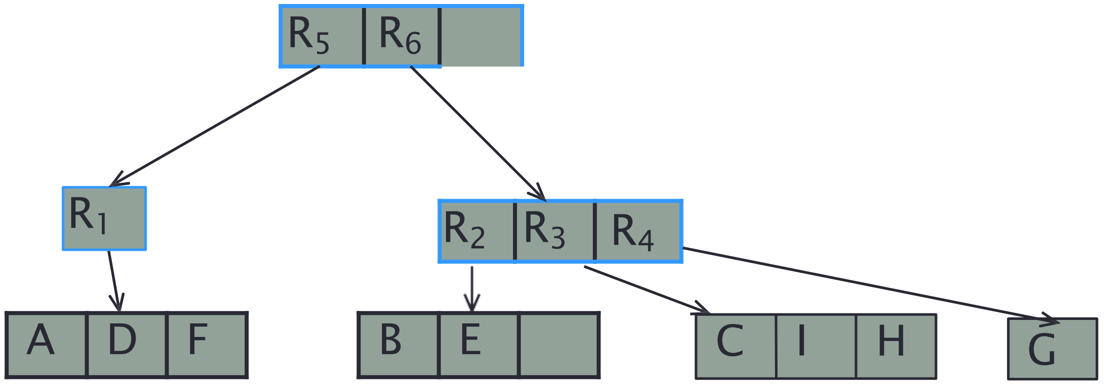
Insertion

Result of the split



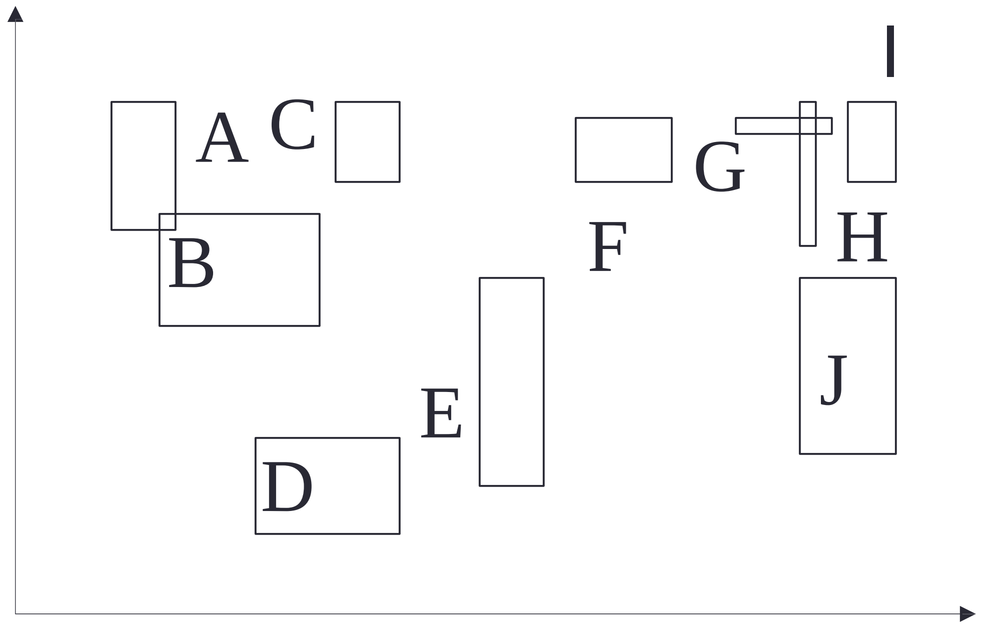
Insertion

A new level is created.



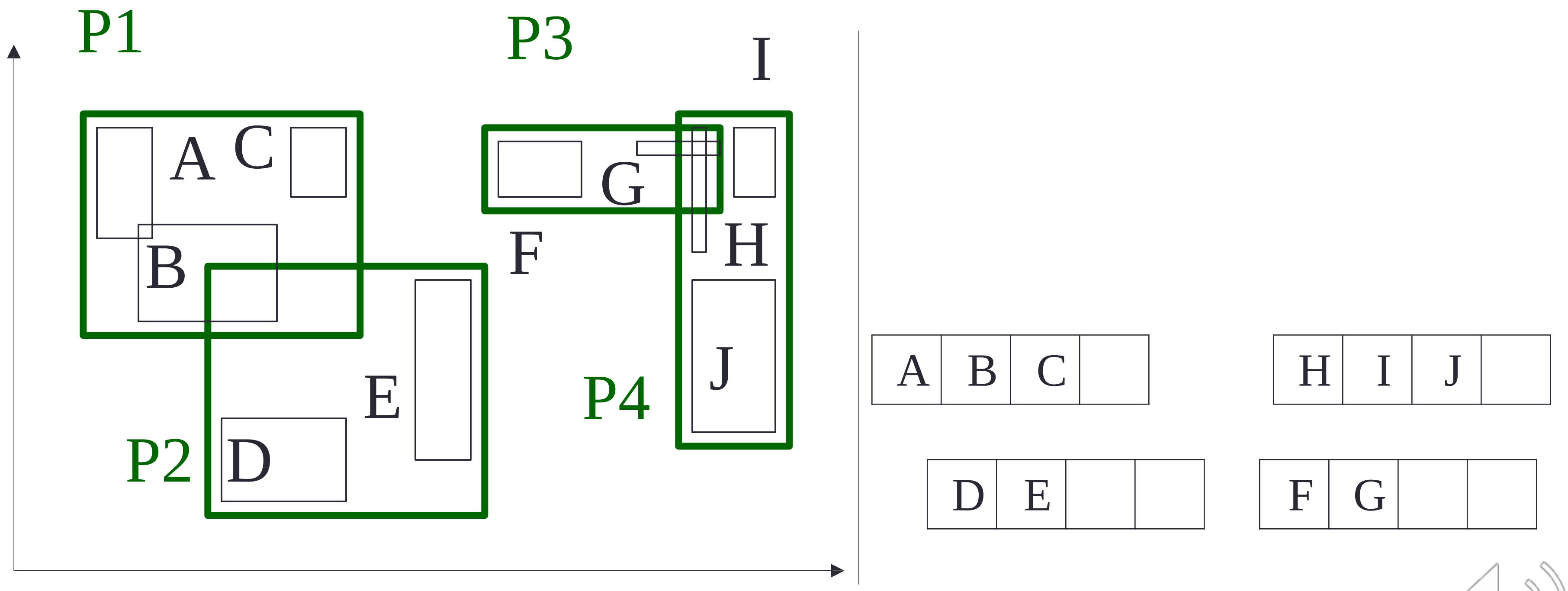
Task 2: Index structures for n-dimensional data: R-tree

- Do the splitting and give the arrangement of the Minimum Bounding Box (MBB), such that the maximum number of entries is equal to 4 (M=4).



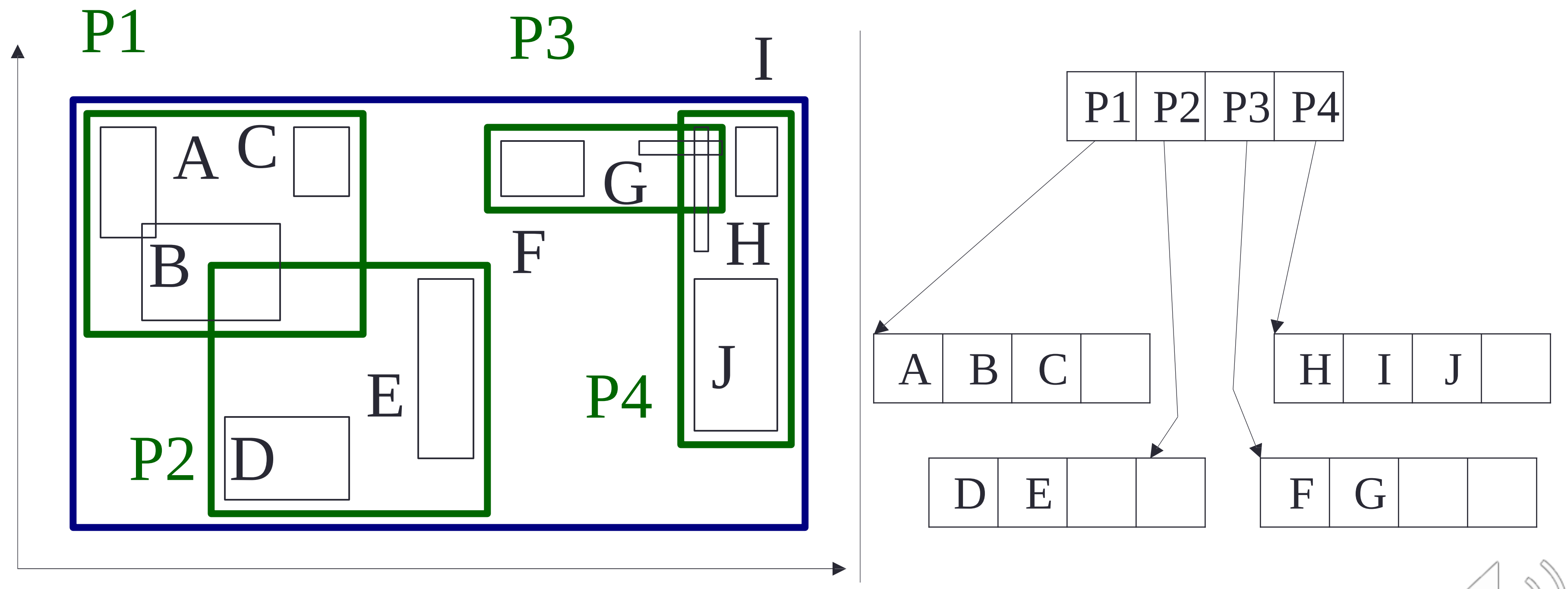
Task 2: Index structures for n-dimensional data: R-tree

- M=4



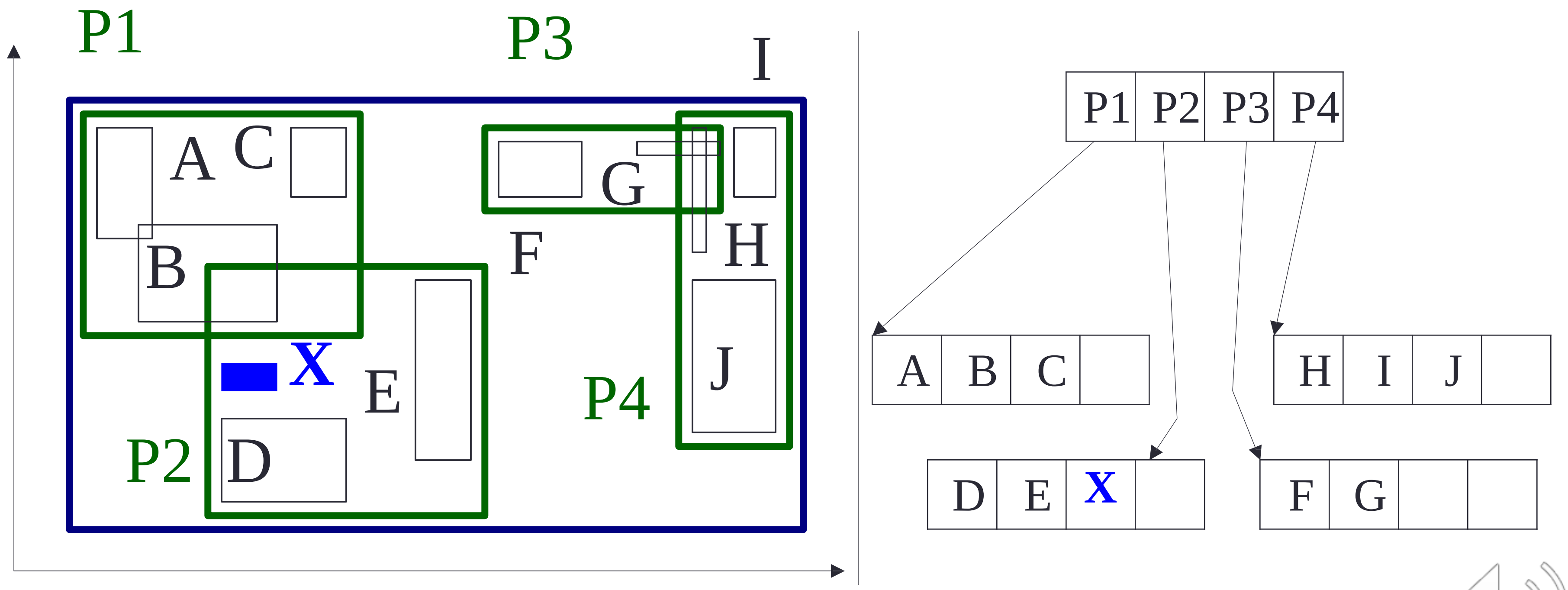
Task 2: Index structures for n-dimensional data: R-tree

- M=4



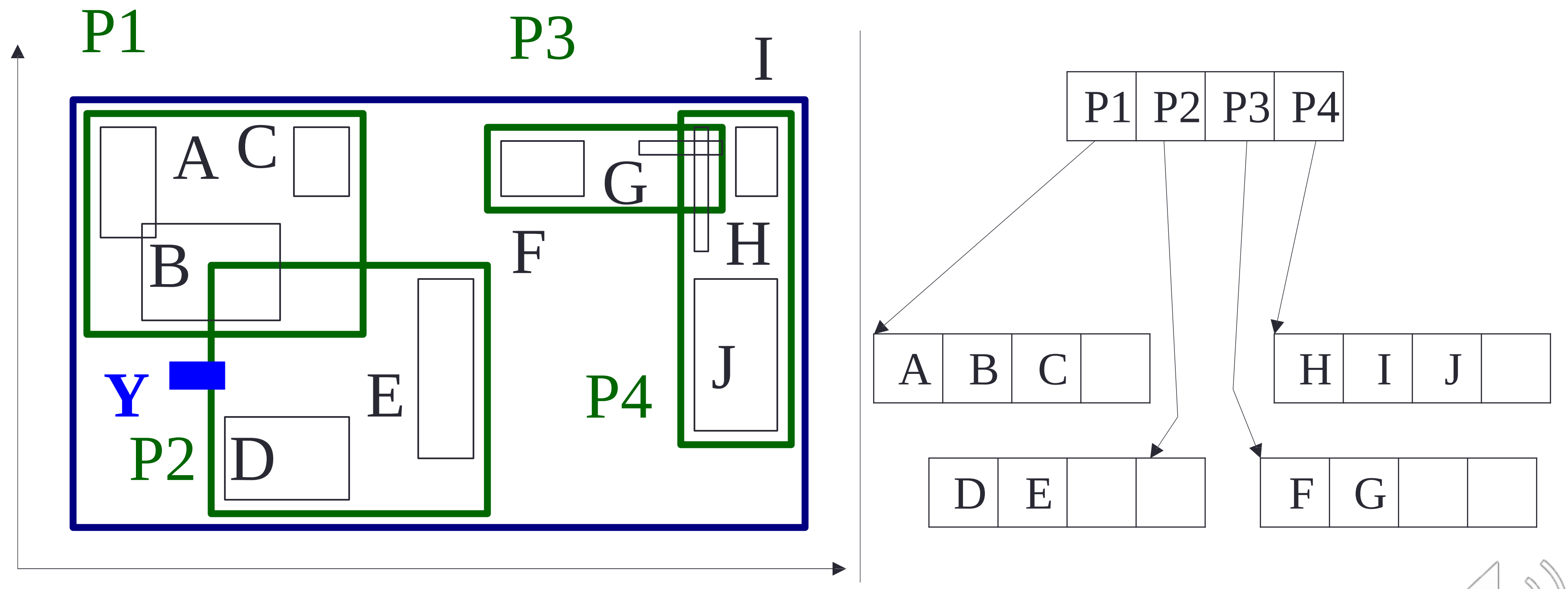
Task 2: Index structures for n-dimensional data: R-tree

Insert X



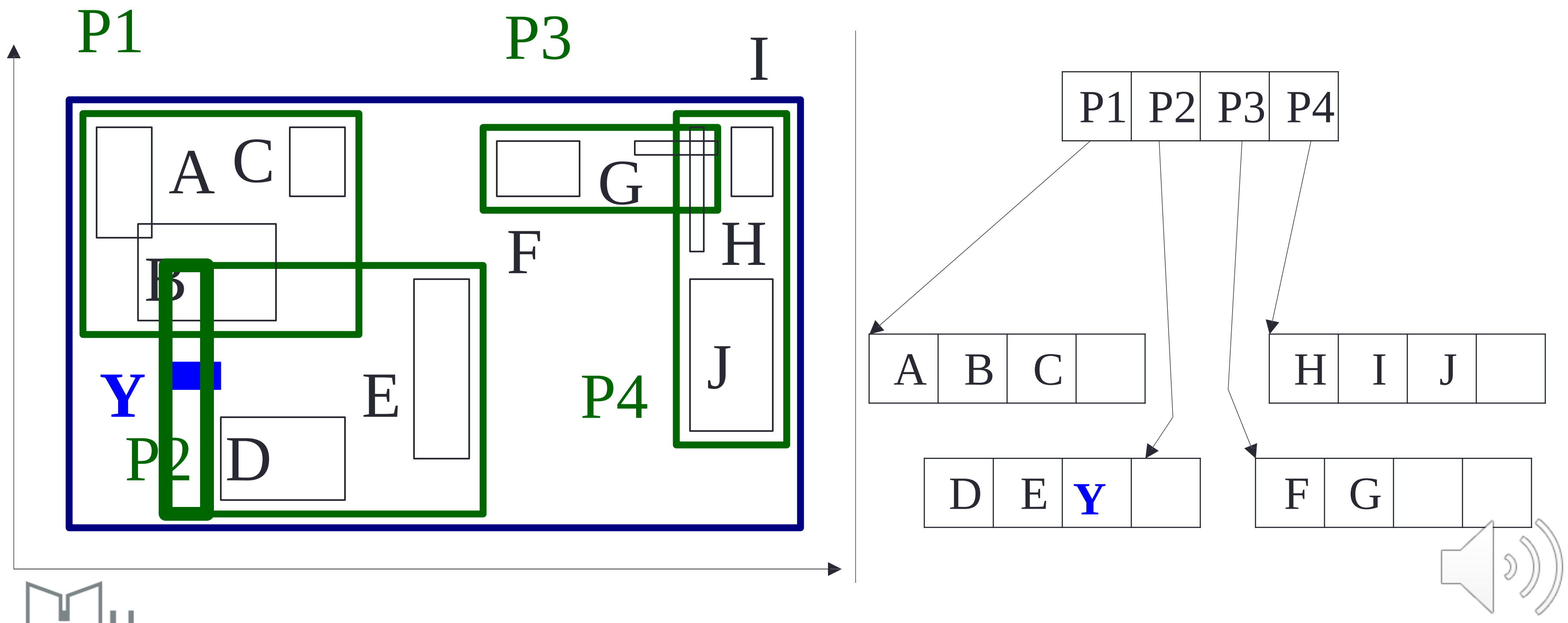
Task 2: Index structures for n-dimensional data: R-tree

Insert Y



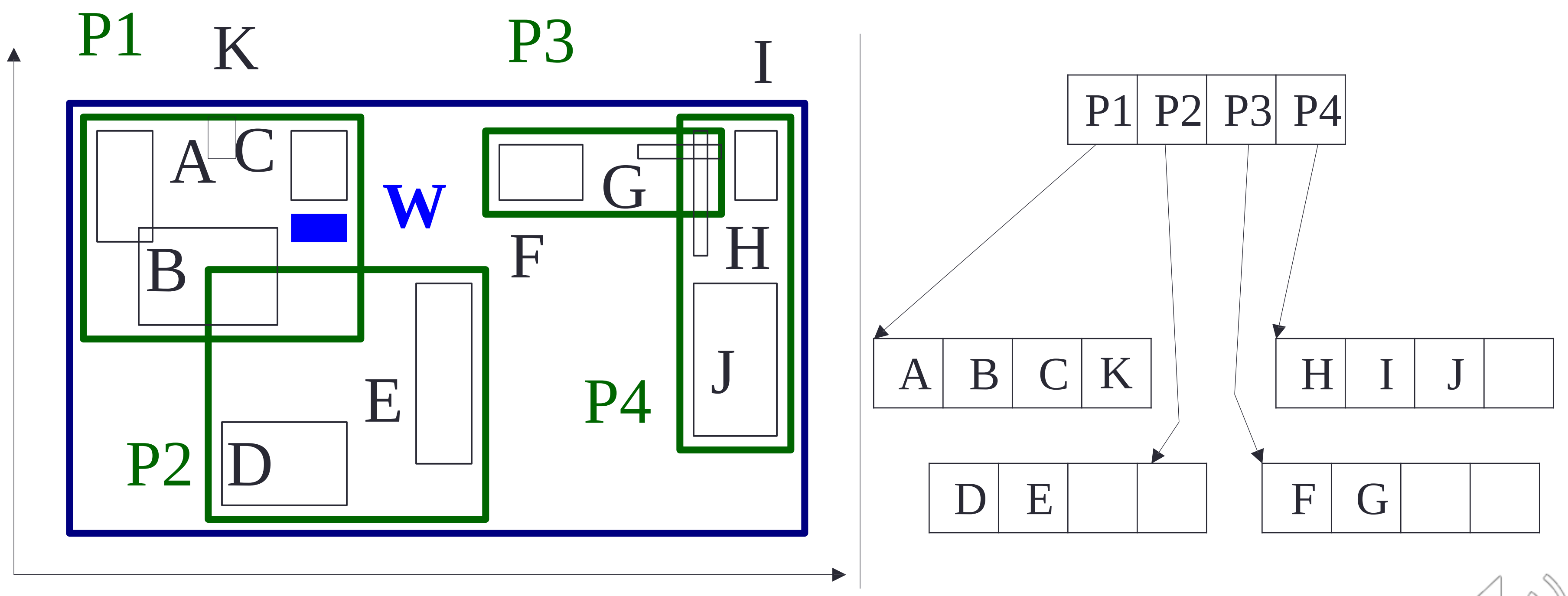
Task 2: Index structures for n- dimensional data: R-tree

- How to find the next node to insert the new object Y?
  - Using ChooseLeaf: Find the entry that needs the least enlargement to include Y and extend the parent MBR.



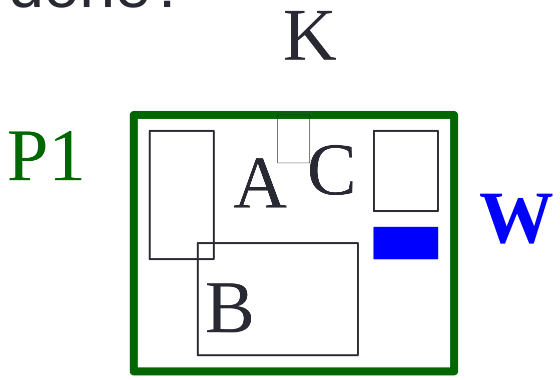
Task 2: Index structures for n-dimensional data: R-tree

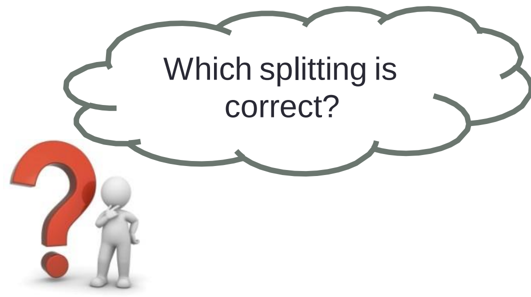
- Since the node is full, in order to insert W, we have to split. How the splitting is done?

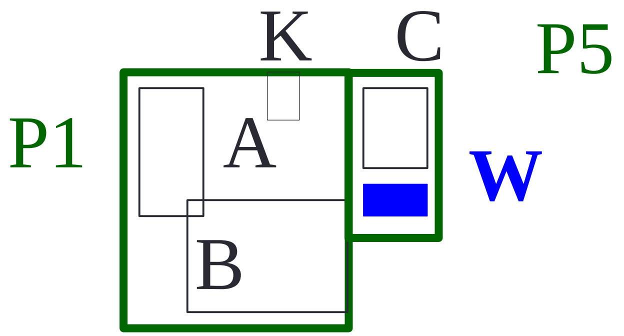


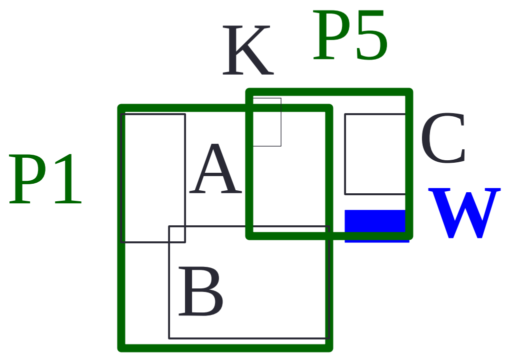
Task 2: Index structures for n-dimensional data: R-tree

- Since the node is full, in order to insert W, we have to split. How the splitting is done?



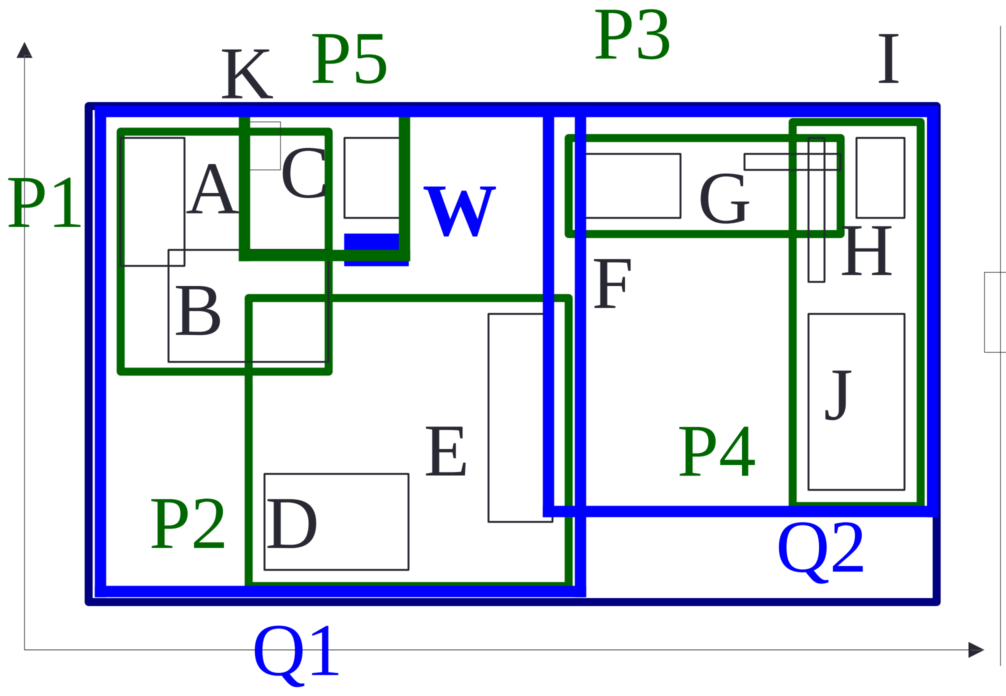


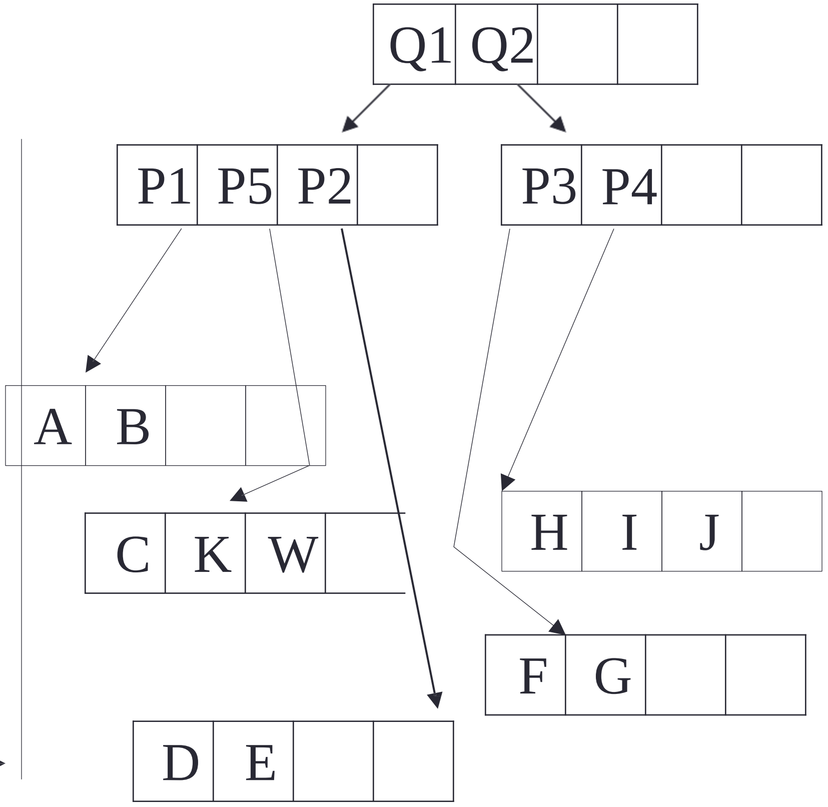




Task 2: Index structures for n-dimensional data: R-tree

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





Task 2: Index structures for n-dimensional data: R-tree

Delete Algorithm: Remove Index-Entry E from an R-tree

- D1 [find node to which E belongs]:
  using the algorithm „Findleaf“, find the leaf L, which contains E;
  quit if E is not found

- D2 [Delete E]:
  Delete E from L

- D3 [Propagate changes]:
  call condense-tree with L as Parameter

- D4 [shorten tree]:
  if the root now only has one child C after the adaptation of the tree, make C the new root of the tree

Task 2: Index structures for n-dimensional data: R-tree

Algorithm Findleaf

- Let T be the root of the considered R-tree; goal: find the leaf containing the entry E

- FL1 [search subtrees]:
  If T is not a leaf, inspect each entry F in T to check if FI and EI overlap. For each such entry, apply Findleaf to the corresponding subtree (updating T with the root of the subtree) until E is found or all entries have been checked

- FL2 [search leaf nodes]:
  If T is a leaf, compare each of its entries with E. Return T once E has been found

Task 2: Index structures for n-dimensional data: R-tree

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.

- C1 Initialization: Set N=L. Set Q, the-set of eliminated nodes, to be empty.
- C2 [Remove nodes containing too few entries]:

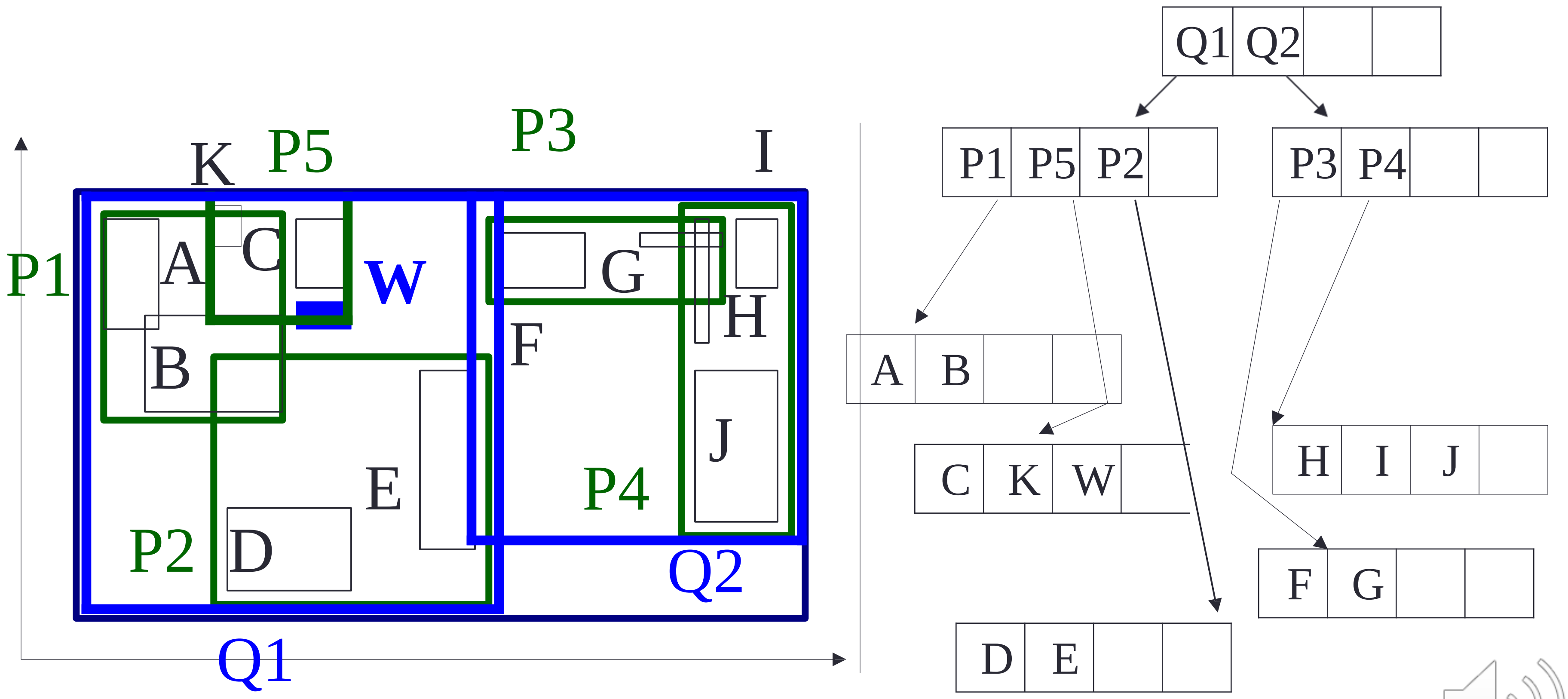
  Let P be the parent of N. If N contains too few entries, remove the entry for N in P and add all entries of N to Q.
- C3 [adjust covering MBB]:

  If N has enough entries, adjust the MBB of N in P so that it tightly contains all entries remaining in N.
- C4 [Re-insert orphaned entries]:

  If P is not the root, apply condensetree to P. Otherwise, apply insert to all entries left in Q.

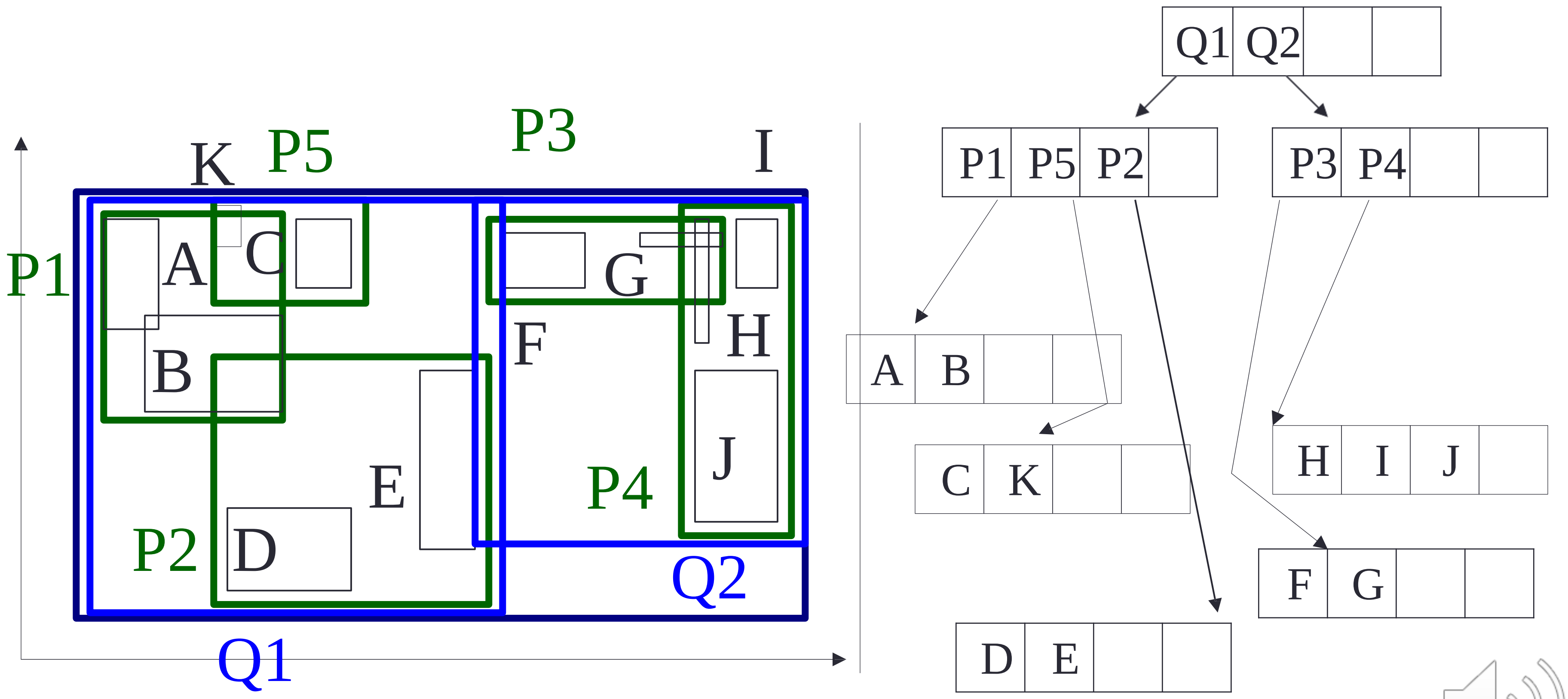
Task 2: Index structures for n-dimensional data: R-tree

- Remove w



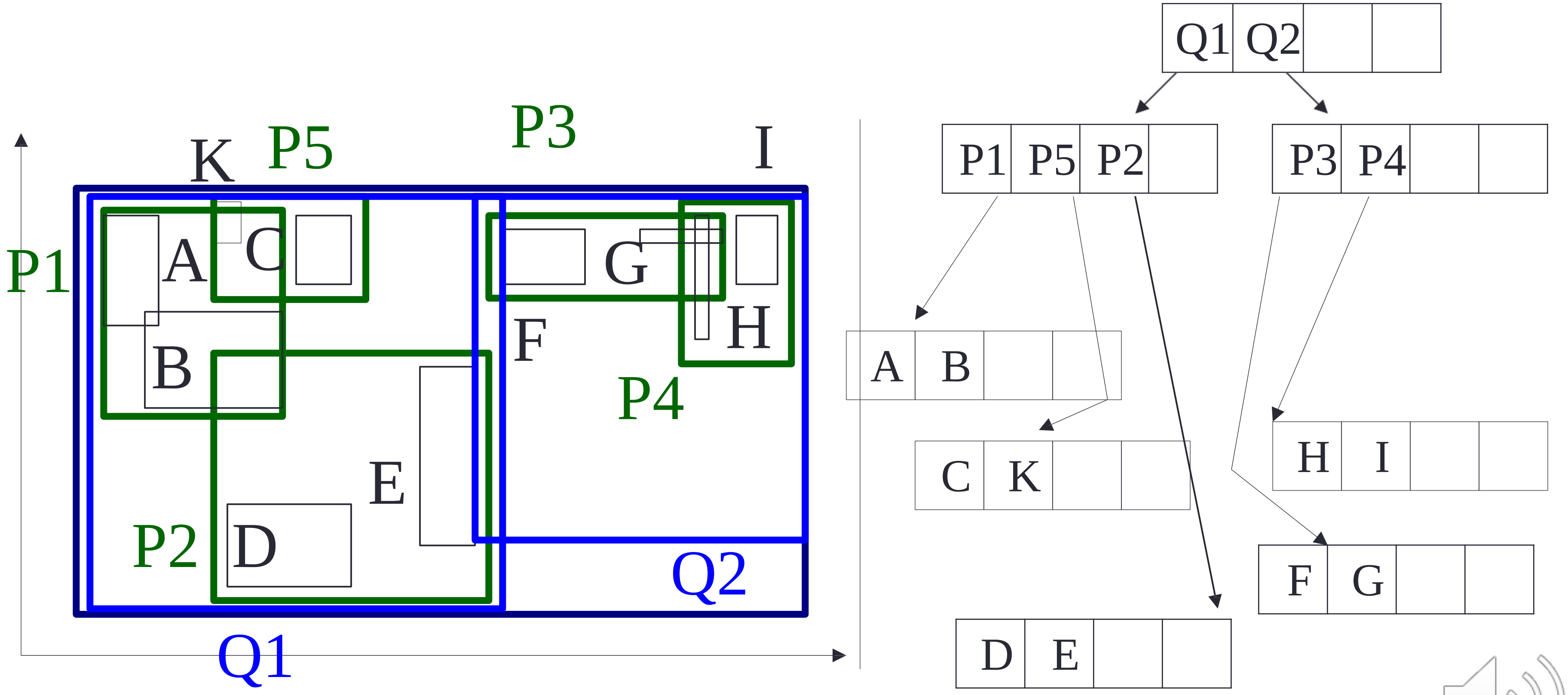
Task 2: Index structures for n-dimensional data: R-tree

- Remove w



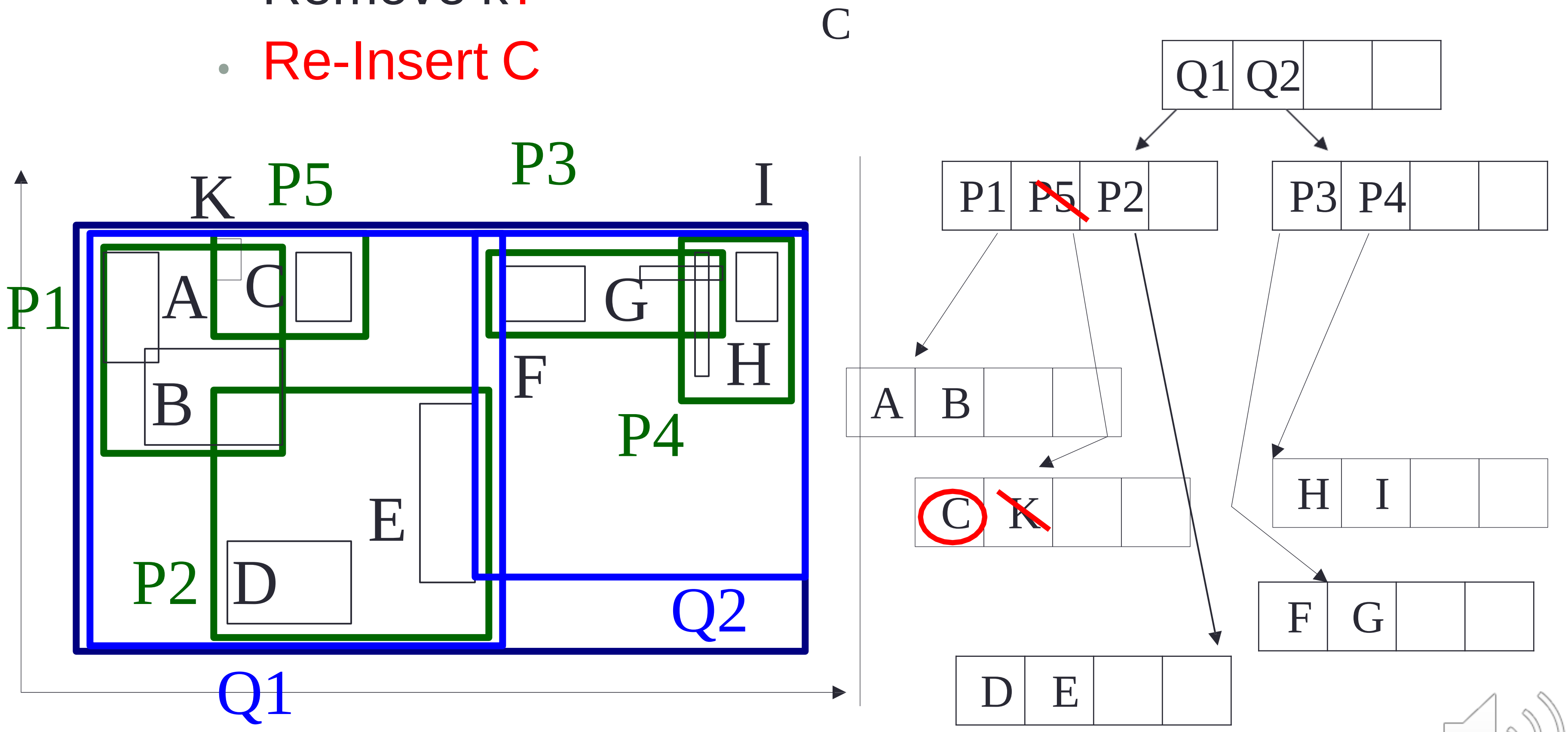
Task 2: Index structures for n-dimensional data: R-tree

- Remove j



Task 2: Index structures for n-dimensional data: R-tree

- Remove k?
- Re-Insert C



For next week

Task 3 and Exam preparation (of my sessions)

The week after is dedicated to Kanishka’s part!

Please put your questions on StudIP till Monday

Task 3: Search algorithms for n-dimensional objects in databases

- Describe the following algorithms for similarity search with and without index:
  - 1. Exact queries
  - 2. Range queries
  - 3. Nearest neighbor queries

Task 3: Search algorithms for n-dimensional objects in databases

- In multimedia databases, we are commonly interested in objects similar to a given one. The result of a similarity match depends on:
    - used distance metric (e.g.; Euclidean metric, Manhattan metric, Maximum metric)
    - used query type.

Task 3: Search algorithms for n-dimensional objects in databases

- **Exact query:** it is the simplest query type which retrieves all points in the database with **identical** feature vectors. It has a single parameter that is the query point Q.

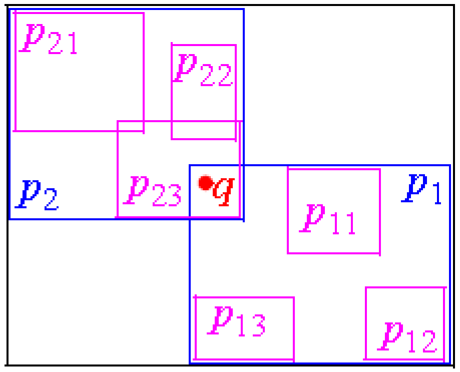
$$PointQuery(DB, Q) = \{P \in DB | P = Q\}$$

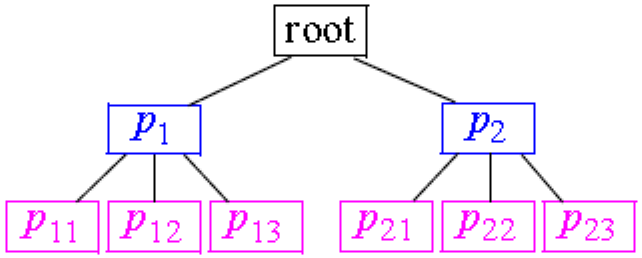
- Without index: the algorithm is simply a sequential search in the DB.

Task 3: Search algorithms for n-dimensional objects in databases

- With Index:

```
function ExactMatchQuery ($q$: Point, $pa$: DiskAddress): Set of Point
    $result := \emptyset$ ;
    $p :=$ LoadPage ($pa$) ;
    if IsDataPage ($p$) then
        for $i := 0$ to $p.num\_objects$ do
            if $q = p.object\,[i]$ then
                $result := result \cup p.object\,[i]$ ;
    else
        for $i := 0$ to $p.num\_objects$ do
            if IsPointInRegion ($q$, $p.region\,[i]$) then
                $result := result \cup$ ExactMatchQuery ($q$, $p.childpage\,[i]$) ;
    return $result$ ;
```





Task 3: Search algorithms for n-dimensional objects in databases

- **Range query:** it returns all points P that have a smaller or equal distance r from the query point Q with respect to the used metric M.

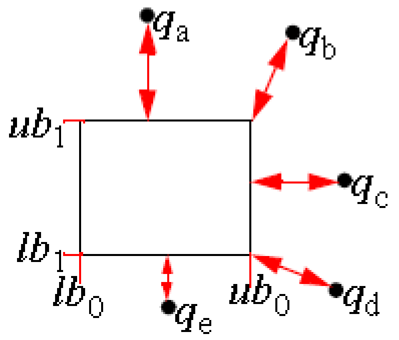
- Without index: similar to the exact query algorithm, it is simply a sequential search in the DB.

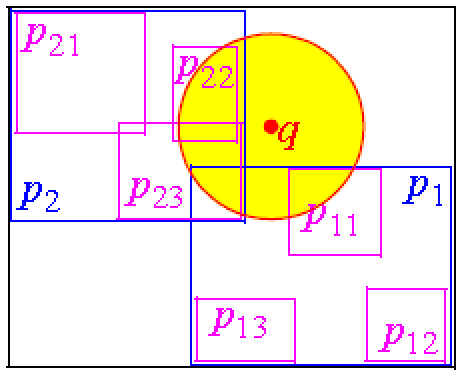
$$RangeQuery(DB, Q, r, M) = \{P \in DB | \delta_M(P, Q) \leq r\}$$

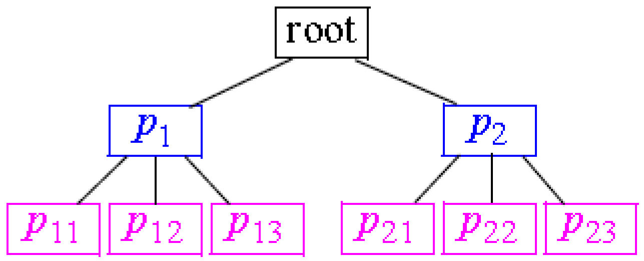
Task 3: Search algorithms for n-dimensional objects in databases

- With Index:

```
function RangeQuery ($q$: Point; $\varepsilon$: Real; $pa$: DiskAddress): Set of Point
    $result := \emptyset$ ;
    $p :=$ LoadPage ($pa$) ;
    if IsDataPage ($p$) then
        for $i := 0$ to $p.num\_objects$ do
            if distance ($q$, $p.object$ [$i$]) $\leq \varepsilon$ then
                $result := result \cup p.object$ [$i$] ;
    else
        for $i := 0$ to $p.num\_objects$ do
            if MINDIST ($q$, $p.region$ [$i$]) $\leq \varepsilon$ then
                $result := result \cup$ RangeQuery ($q$, $p.childpage$ [$i$]) ;
    return $result$ ;
```







Task 3: Search algorithms for n-dimensional objects in databases

- **Nearest neighbor query (**NNQ)**.**
- In contrast to the Range Query, NNQ returns exactly one result, namely the most similar (with the lowest distance) to the query point Q.

$$NNQ(DB,Q,M)=\{P\in DB|\forall P'\in DB:\delta_M(P,Q)\leq\delta_M(P',Q)\}$$

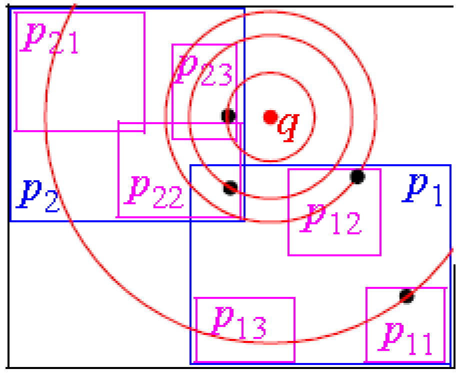
- A relative to this type of query is the k-Nearest Neighbor Query(k-NNQ) which returns k nearest points.

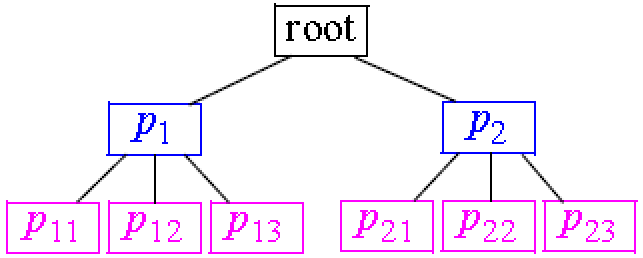
$$k-NNQ(DB,Q,k,M)=\{P_1...P_k\in DB|\neg\exists P'\in DB\backslash\{P_1...P_k\}$$
$$\wedge\neg\exists i,1\leq i\leq k:\delta_M(P_i,Q)>\delta_M(P',Q)\}$$

Task 3: Search algorithms for n-dimensional objects in databases

- With Index:

```
procedure SimpleNNQuery (q: Point; pa: DiskAddress)
    p := LoadPage (pa) ;
    if IsDataPage (p) then
        for i := 0 to p.num_objects do
            if distance (q, p.object [i]) ≤ resultdist then
                result := p.object [i] ;
                resultdist := distance (q, p.object [i]) ;
    else
        for i := 0 to p.num_objects do
            if MINDIST (q, p.region [i]) ≤ resultdist then
                SimpleNNQuery (q, p.childpage [i]) ;
```





Resultdist has to be initialized with infinity
MINDIST is the function to calculate the used metric