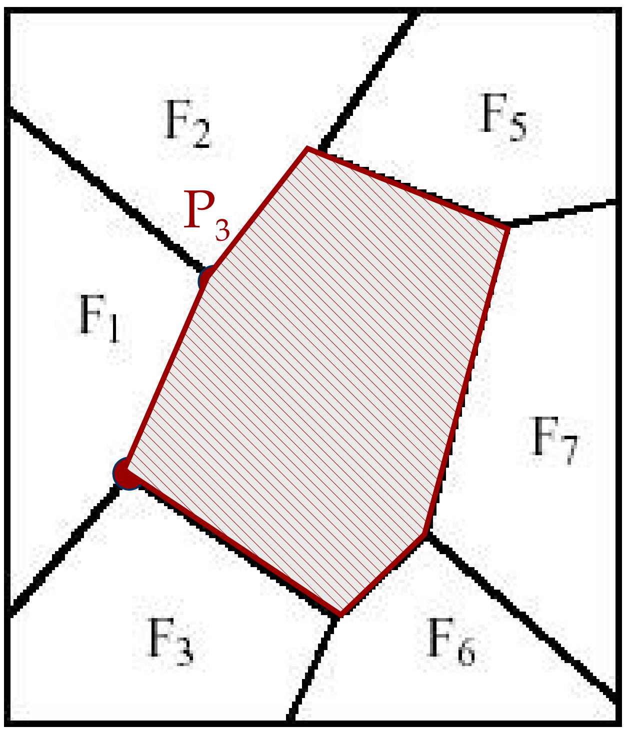
Spatial Data Management

12

Relational Representation of Spatial Data

- *Example*: Representation of geometric objects (here: fields of land) in normalized relations. Redundancy free representation requires distribution of information over 3 tables!



Fields

FID	BID
F_1	B_1
F_1	B_2
F_1	B_3
F_1	B_4
F_4	B_2
F_4	B_5
F_4	B_6
F_4	B_7
F_4	B_8
F_4	B_9
F_7	B_{10}
F_7	B_{11}
F_7	B_{12}

Borders

BID	PID_1	PID_2
B_1	P_1	P_2
B_2	P_2	P_3
B_3	P_3	P_4
B_4	P_4	P_1
B_5	P_2	P_5
B_6	P_5	P_6
B_7	P_6	P_7
B_8	P_7	P_8
B_9	P_8	P_3
B_{10}	P_6	P_9
B_{11}	P_9	P_{10}
B_{12}	P_{10}	P_7

Points

PID	XCoord	YCoord
P_1	X_{P1}	Y_{P1}
P_2	X_{P2}	Y_{P2}
P_3	X_{P3}	Y_{P3}
P_4	X_{P4}	Y_{P4}
P_5	X_{P5}	Y_{P5}
P_6	X_{P6}	Y_{P6}
P_7	X_{P7}	Y_{P7}
P_8	X_{P8}	Y_{P8}
P_9	X_{P9}	Y_{P9}
P_{10}	X_{P10}	Y_{P10}

Relational Representation of Spatial Data

- For (spatial) queries involving fields it is necessary to reconstruct the spatial information from the different tables
- E.g.: if we want to determine if a given point P is inside field F_2, we have to find all corner-points of parcel F_2 first

```
SELECT Points.PID, XCoord, YCoord
     FROM Fields, Border, Points
     WHERE FID = 'F2' AND
     Fields.BID = Borders.BID AND
     (Borders.PID1 = Points.PID OR
     Borders.PID2 = Points.PID)
```

- Even this simple query requires expensive joins of 3 tables
- Querying the geometry is not directly supported (P in F_2?)

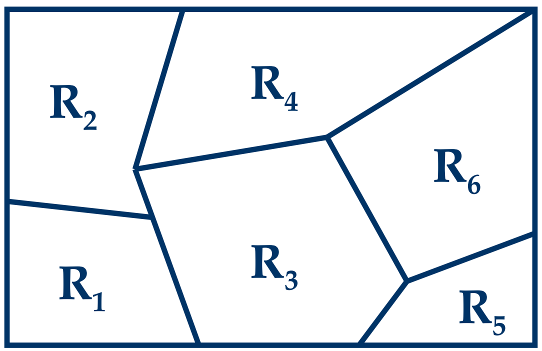
Extension of Relational Model for Spatial Data

- Integration of spatial data types and operations into the core of a DBMS (→ object-oriented and object-relational databases)
  - Data types such as *Point*, *Line*, *Polygon*
  - Operations such as *ObjectIntersect*, *RangeQuery*, etc.
- Advantages:
  - Natural extension of the relational model and query languages
  - Facilitates design and querying of spatial databases
  - Spatial data types and operations can be supported by spatial index structures and efficient algorithms, implemented in the core of a DBMS
- All major database vendors today implement support for spatial data and operations in their database systems via object-relational extensions

Extension of Relational Model for Spatial Data

- Example:

ForestZones(Zone:*Polygon*,ForestOfficial:*String*,Area:*Cardinal*)



ForestZones		
Zone	ForestOfficial	Area (m^2)
R_1	**Stevens**	**3900**
R_2	**Behrens**	**4250**
R_3	**Lee**	**6700**
R_4	**Goebel**	**5400**
R_5	**Jones**	**1900**
R_6	**Kent**	**4600**

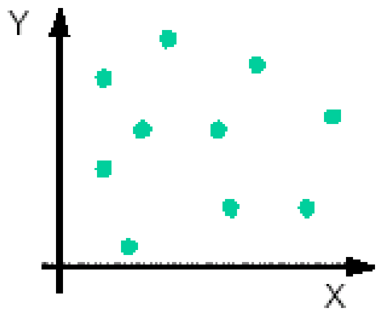
- The province decides that a reforestation is necessary in an area described by a polygon S. Find all forest officials affected by this decision.

```
SELECT ForestOfficial
     FROM ForestZones
     WHERE ObjectIntersects (S,Zone)
```

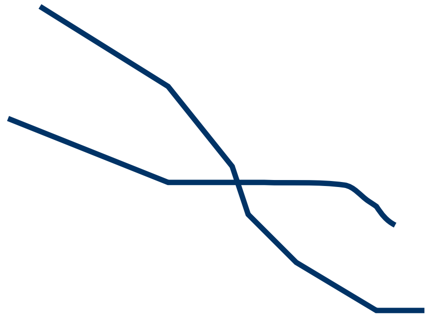
Data Types for Spatial Objects

- Spatial objects are described by
  - Spatial Extent
    - *location* and/or *boundary* with respect to a reference point in a coordinate system, which is at least 2-dim.
    - Basic object types: *Point, Lines, Polygon*
  - Other Non-Spatial Attributes
    - Thematic attributes such as height, area, name, etc.

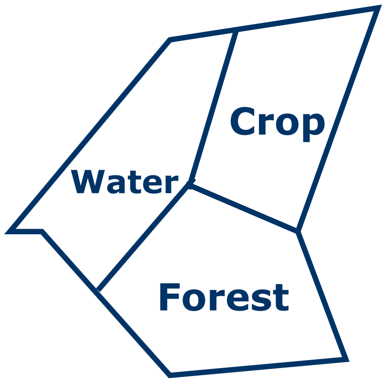
2-dim. points



2-dim. lines



2-dim. polygons

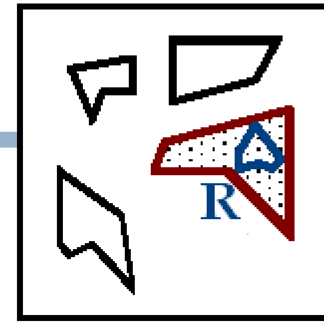


Spatial Query Processing

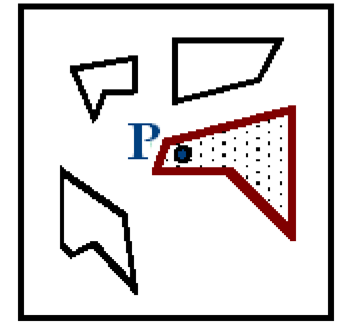
- DBMS has to support two types of operations
  - Operations to retrieve certain subsets of spatial object from the database
    - "Spatial Queries/Selections", e.g., window query, point query, etc.
  - Operations that perform basic geometric computations and tests
    - E.g., point in polygon test, intersection of two polygons etc.
- Spatial selections, e.g. in geographic information systems, are often supported by an interactive graphical user interface

Basic Spatial Queries

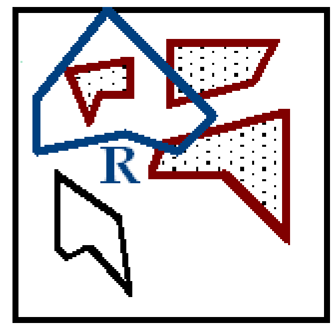
- ***Containment Query***: Given a spatial object R, find all objects that completely contain R. If R is a point: ***Point Query***
- ***Region Query***: Given a region R (polygon or circle), find all spatial objects that intersect with R. If R is a rectangle: ***Window Query***
- ***Enclosure Query***: Given a polygon region R, find all objects that are completely contained in R
- ***K-Nearest Neighbor Query***: Given an object P, find the k objects that are closest to P (typically for points)



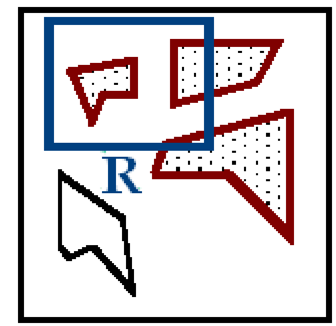
Containment Query



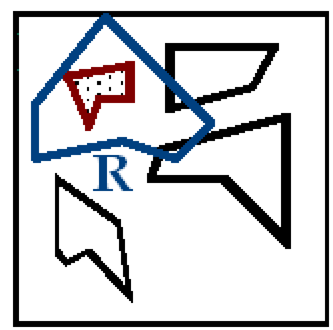
Point Query



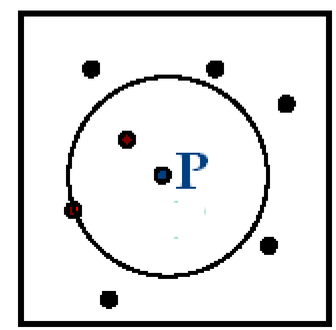
Region Query



Window Query



Enclosure Query



2-nn Query

Basic Spatial Queries - Spatial Join

- Given two sets of spatial objects (typically minimum bounding rectangles)

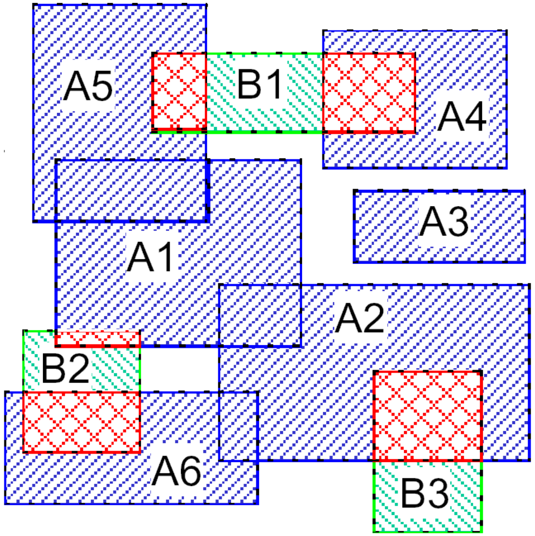
$$S_1 = \{R_1, R_2, \ldots, R_m\} \text{ and } S_2 = \{R'_1, R'_2, \ldots, R'_n\}$$

- Spatial Join: Compute all pairs of objects (R, R′) such that

$$R \in S_1, R' \in S_2 \text{ and } R \text{ intersects } R' \ (R \cap R' \neq \emptyset)$$

- Spatial predicates other than intersection are also possible, e.g. all pairs of objects that are within a certain distance from each other

$$\{A_1, A_2, .., A_6\} \otimes \{B_1, \ldots, B_3\}$$



Spatial Join

Answer set

(A_5, B_1)

(A_4, B_1)

(A_1, B_2)

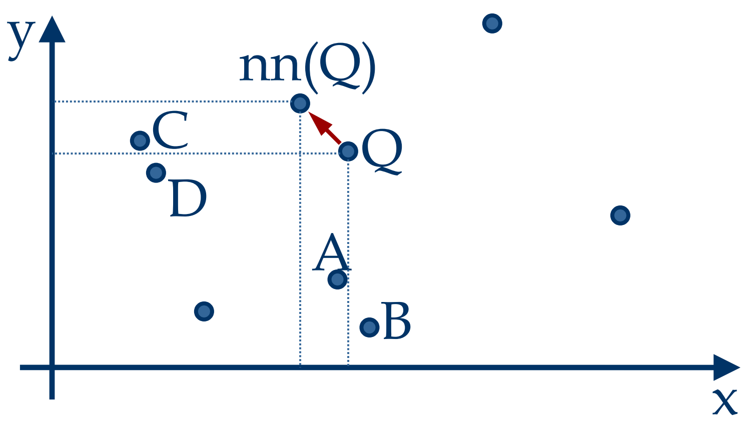
(A_6, B_2)

(A_2, B_3)

Index Support for Spatial Queries

- Conventional index structures such as B-trees are not designed to support spatial queries
  - Group objects only along one dimension
  - Do not preserve spatial proximity
- E.g. nearest neighbor query:

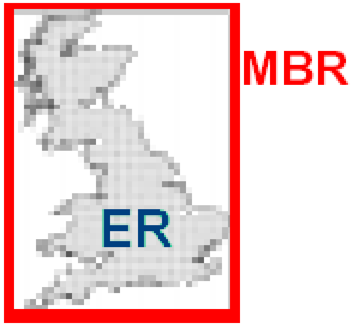
  Nearest neighbor of Q is typically not the nearest neighbor in any single dimension

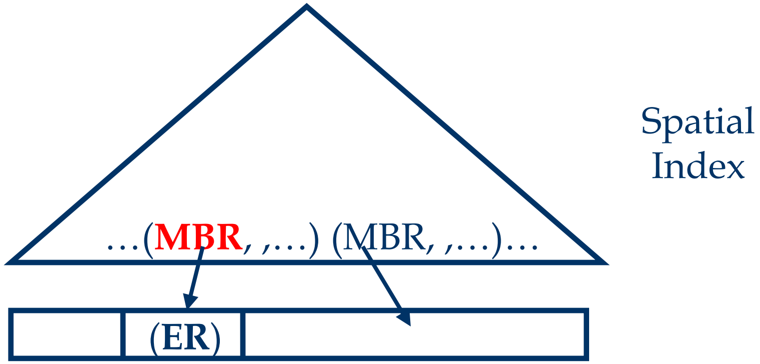


A and B closer in the X dimension;
C and D closer in the Y dimension.

Index Support for Spatial Queries (cont)

- Spatial index structures try to preserve spatial proximity
  - Group objects that are close to each other on the same data page
  - Problem: the number of bytes to store extended spatial objects (lines, polygons) varies
- Solution:
  - Store ***Approximations*** of spatial objects in the index structure, typically axis-parallel minimum bounding rectangles (MBR)
  - Exact object representation (ER) stored separately; pointers to ER in the index

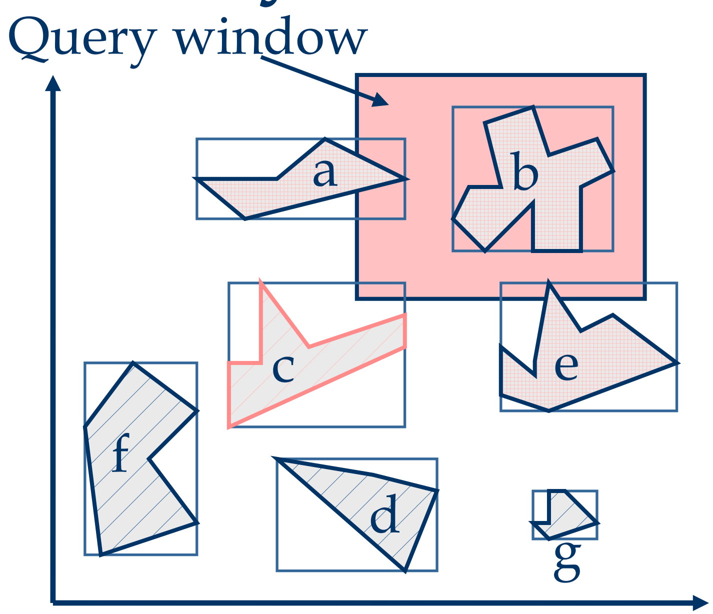




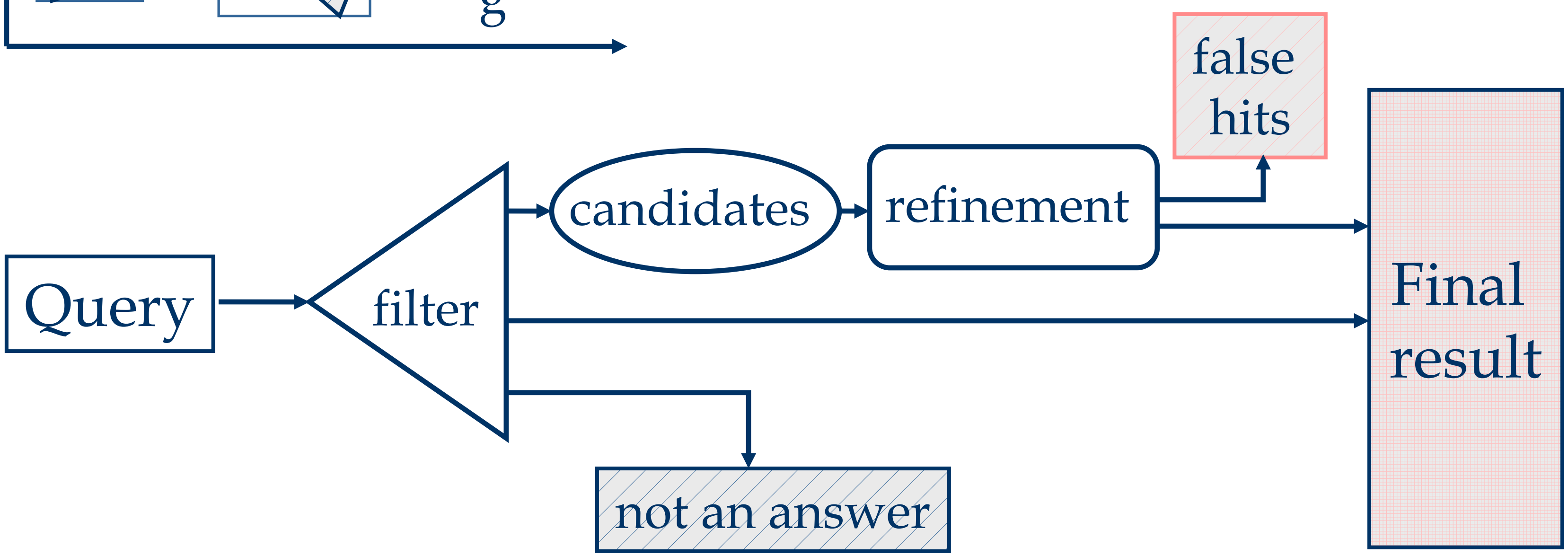
Query Processing Using Approximations

- Two-step procedure:
  1. Filter Step:
     - Use the index to find all approximations that satisfy the query
     - Some objects already satisfy the query based on the approximation, others have to be checked in the refinement step → *Candidate Set*
  2. Refinement Step:
     - Load the exact object representations for candidates left after the filter step and test whether they satisfies the query

Query Processing Using Approximations



- *a* and *b* are certain answers
- *f*, *d*, *g* are certainly not answers
- *c* and *e* are candidates
- *c* is a false hit



Embedding of the 2-dim. space into a 1-dim space

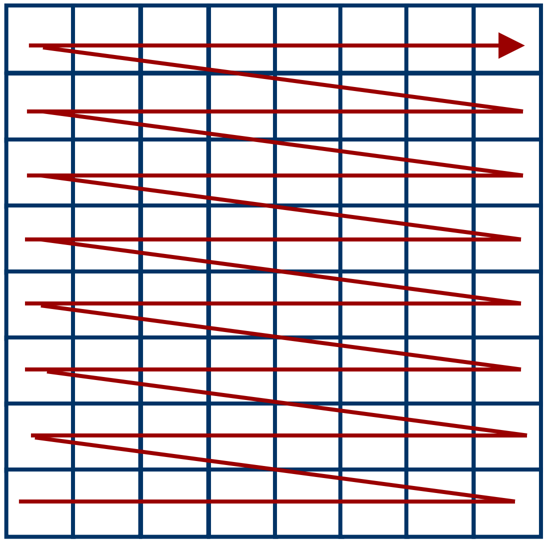
Basic Idea:

- The data space is partitioned into rectangular cells.
- Use a space filling curve to assign cell numbers to the cells (define a linear order on the cells)
  - The curve should preserve spatial proximity as good as possible
  - Cell numbers should be easy to compute
- Objects are approximated by cells.
- Store the cell numbers for objects in a conventional index structure with respect to the linear order

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

Space Filling Curves

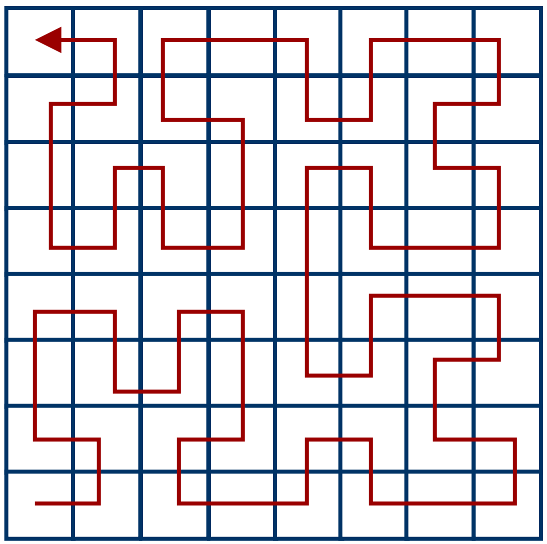
- Lexicographic order



56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Space Filling Curves (cont)

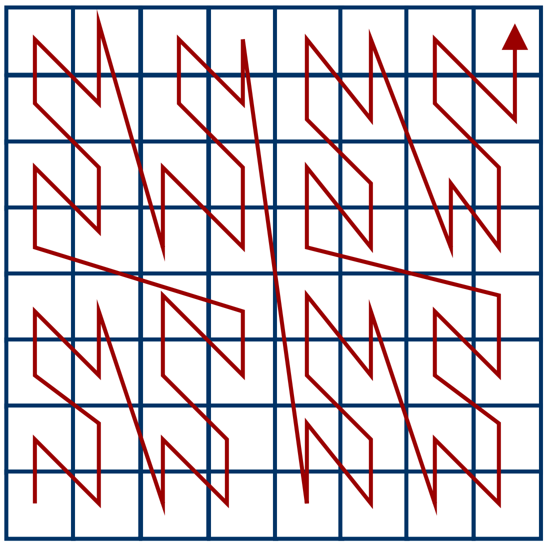
- Hilbert Curve



63	62	49	48	47	44	43	42
60	61	50	51	46	45	40	41
59	56	55	52	33	34	39	38
58	57	54	53	32	35	36	37
5	6	9	10	31	28	27	26
4	7	8	11	30	29	24	25
3	2	13	12	17	18	23	22
0	1	14	15	16	19	20	21

Space Filling Curves

- Z-Order

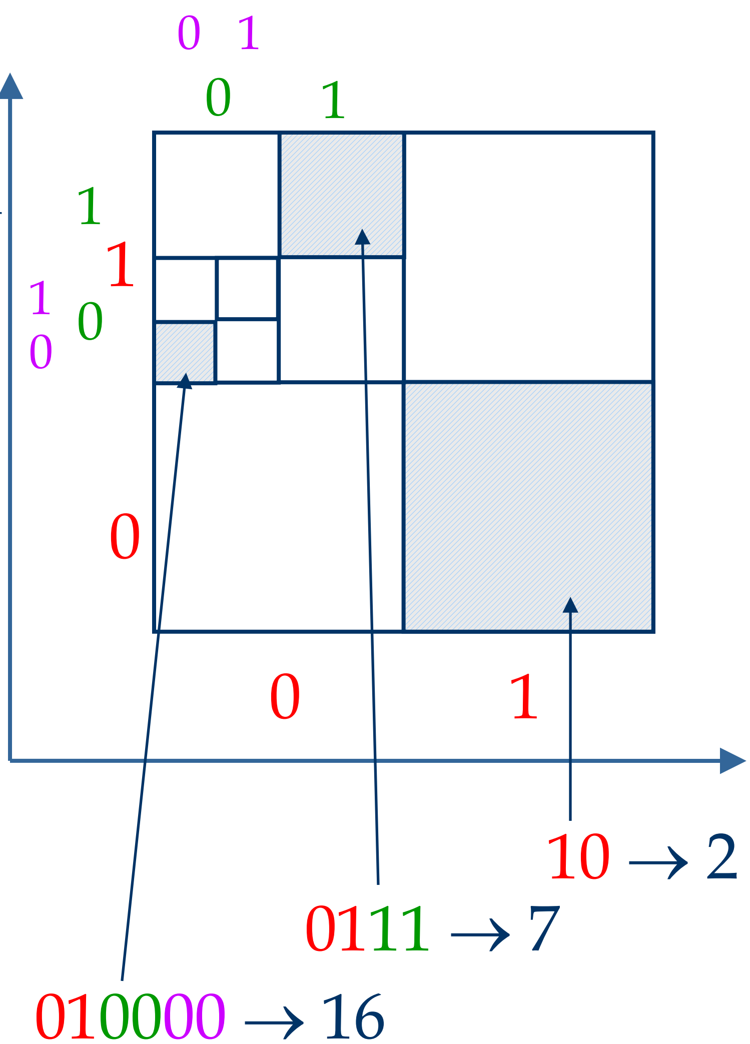


21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

- Z-Order preserves spatial proximity relatively good
- Z-Order is easy to compute

Z-Order – Z-Values

- Coding of Cells
  - Partition the data space recursively into two halves
  - Alternate X and Y dimension
  - Left/bottom → 0
  - Right/top → 1



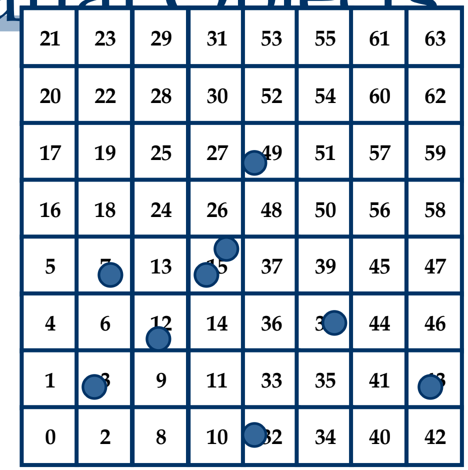
- **Z-Value: (*c*, *l*)**

  c = decimal value of the bit string

  l = level (number of bits)

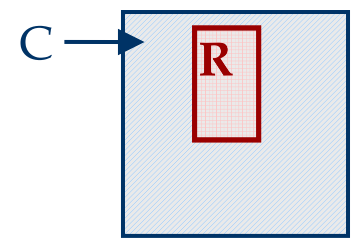
  - if all cells are on the same level, then *l* can be omitted

Z-Order - Representation of Spatial Objects

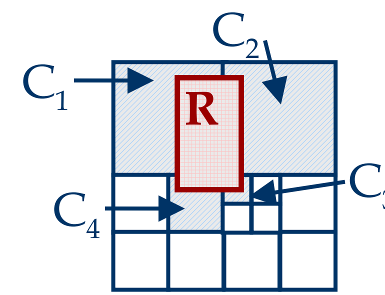


- For Points
  - Use a fixed a resolution of the space in both dimensions, i.e., each cell has the same size
  - Each point is then approximated by one cell

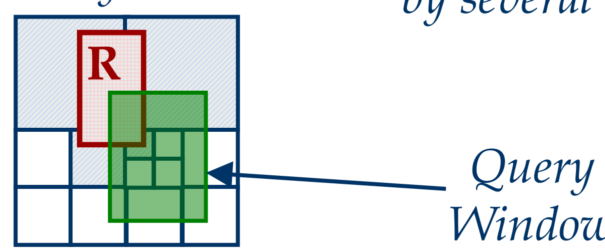
- For extended spatial object
  - minimum enclosing cell
    - Problems with cells that intersect the first partitions already
  - improvement: use several cells
    - Better approximation of the objects
    - Redundant storage
    - Redundant retrieval in spatial queries



Coding of R by one cell



Coding of R by several cells



Query returns the same answer several times

Z-Order – Mapping to a B^+-Tree

- *Linear order* for Z-values to store them in a B+-tree:

Let (c_1, l_1) and (c_2, l_2) be two Z-Values and let $l = \min\{l_1, l_2\}$.

The order relation $\leq_Z$ (that defines a linear order on Z-values) is then defined by:

$$(c_1, l_1) \leq_Z (c_2, l_2) \text{ iff } (c_1 \text{ div } 2^{(l1-l)}) \leq (c_2 \text{ div } 2^{(l2-l)})$$

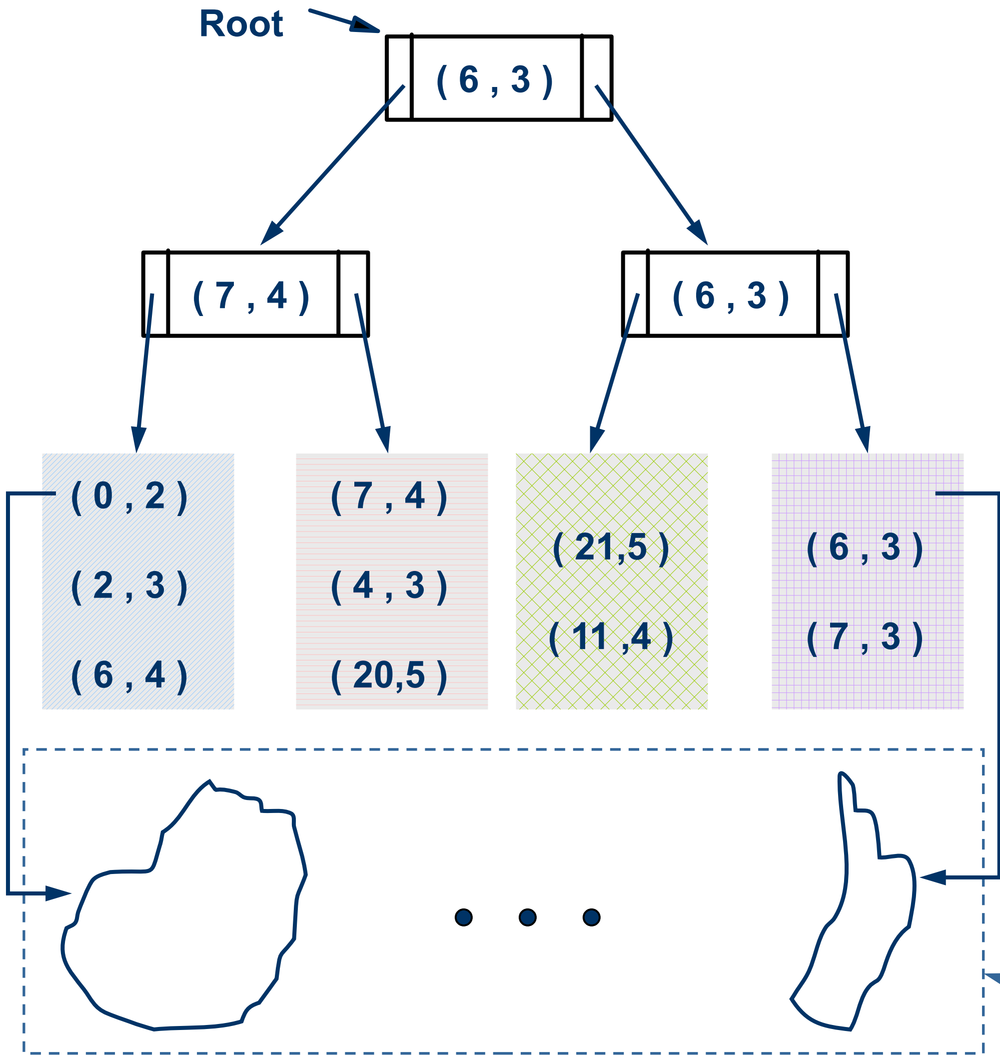
Examples:

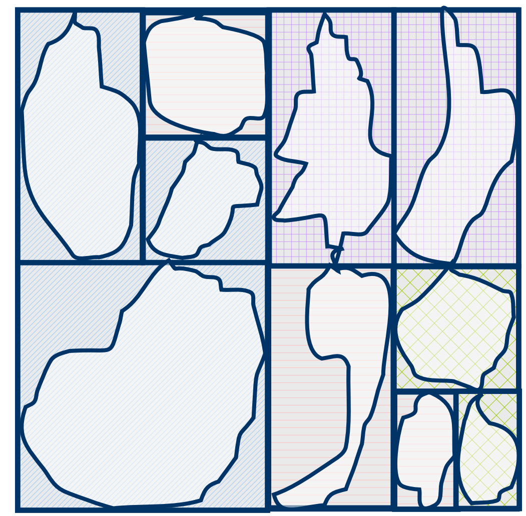
$(1,2) \leq_Z (3,2)$,

$(3,4) \leq_Z (3,2)$,

$(1,2) \leq_Z (10,4)$

Mapping to a B+-Tree - Example





(0 , 2) ≤ (7 , 4) ≤ (7 , 4) ≤ (6 , 3)

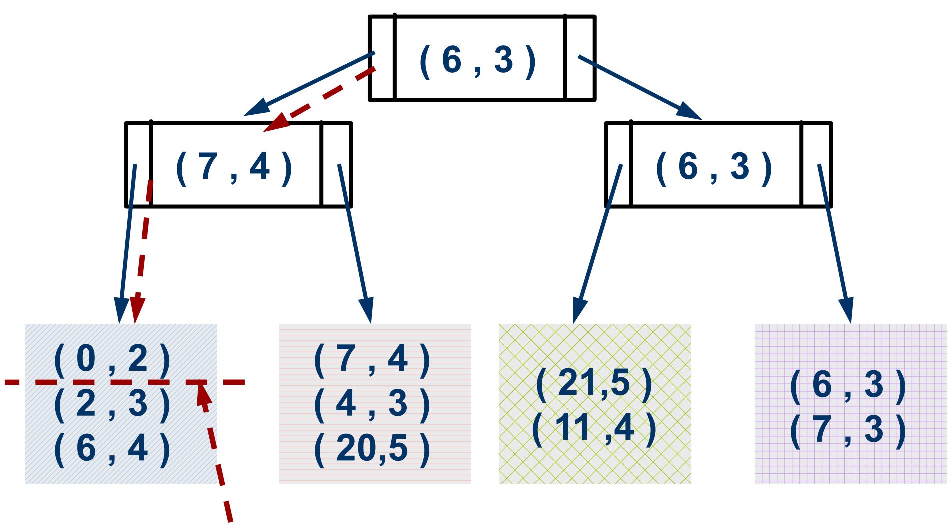
(2 , 3) ≤ (7 , 4) ≤ (4 , 3) ≤ (6 , 3)

(6 , 4) ≤ (7 , 4) ≤ (20 , 5) ≤ (6 , 3)

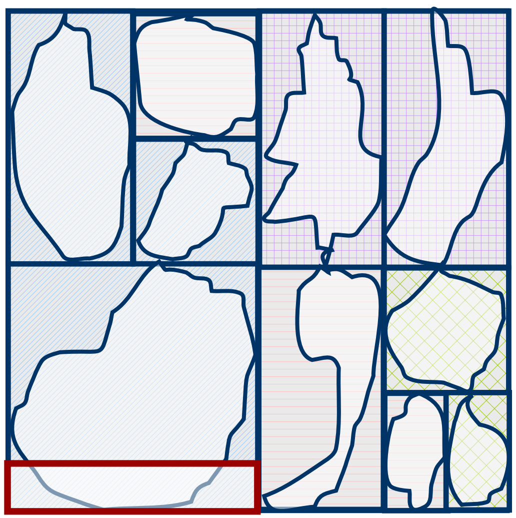
Exact representations stored in a different location

Mapping to a B$^+$-Tree - Window Query

- Window Query → Range Query in the B$^+$-tree
  - find all entries (Z-Values) in the range [*l*, *u*] where
    - *l* = smallest Z-Value of the window (bottom left corner)
    - *u* = largest Z-Value of the window (top right corner)
    - *l* and *u* are computed with respect to the maximum resolution/length of the Z-values in the tree (here: 6)



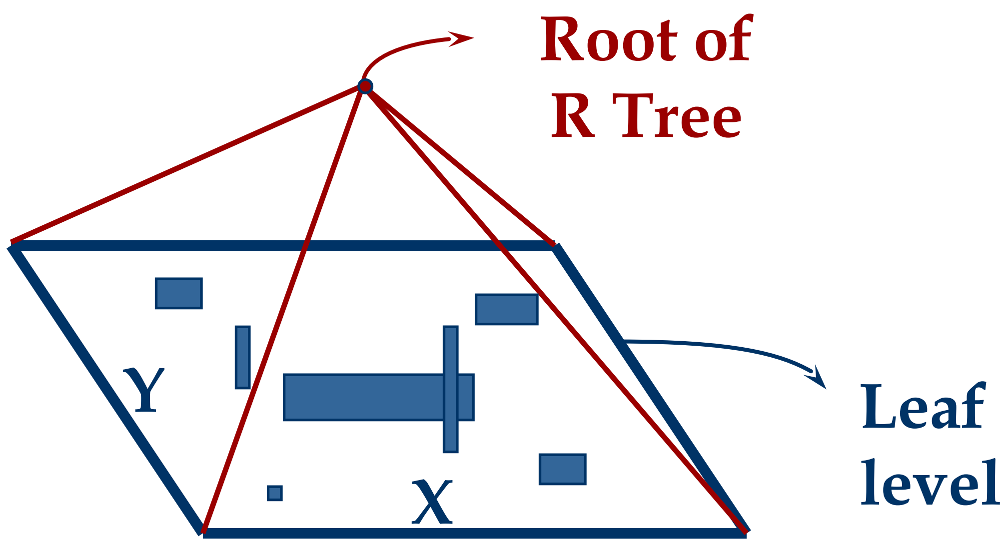
(10,6) ≤ (2,3) Result (0,2)



Window: Min = (0,6), Max = (10,6)

R-Trees

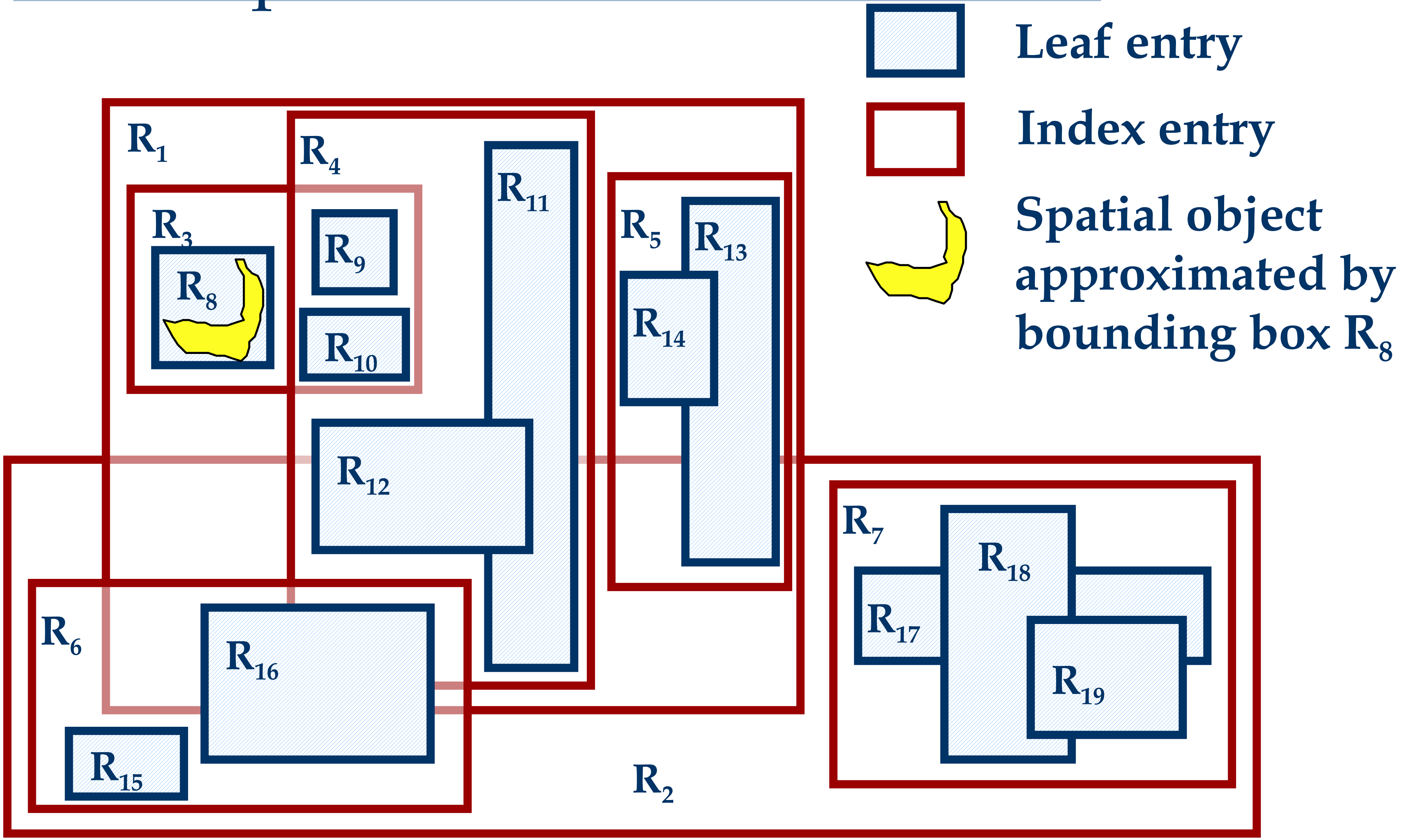
- The R-tree is a tree-structured index that remains balanced on inserts and deletes.
- Each key stored in a leaf entry is intuitively a box, or collection of intervals, with one interval per dimension.
- Example in 2-D:



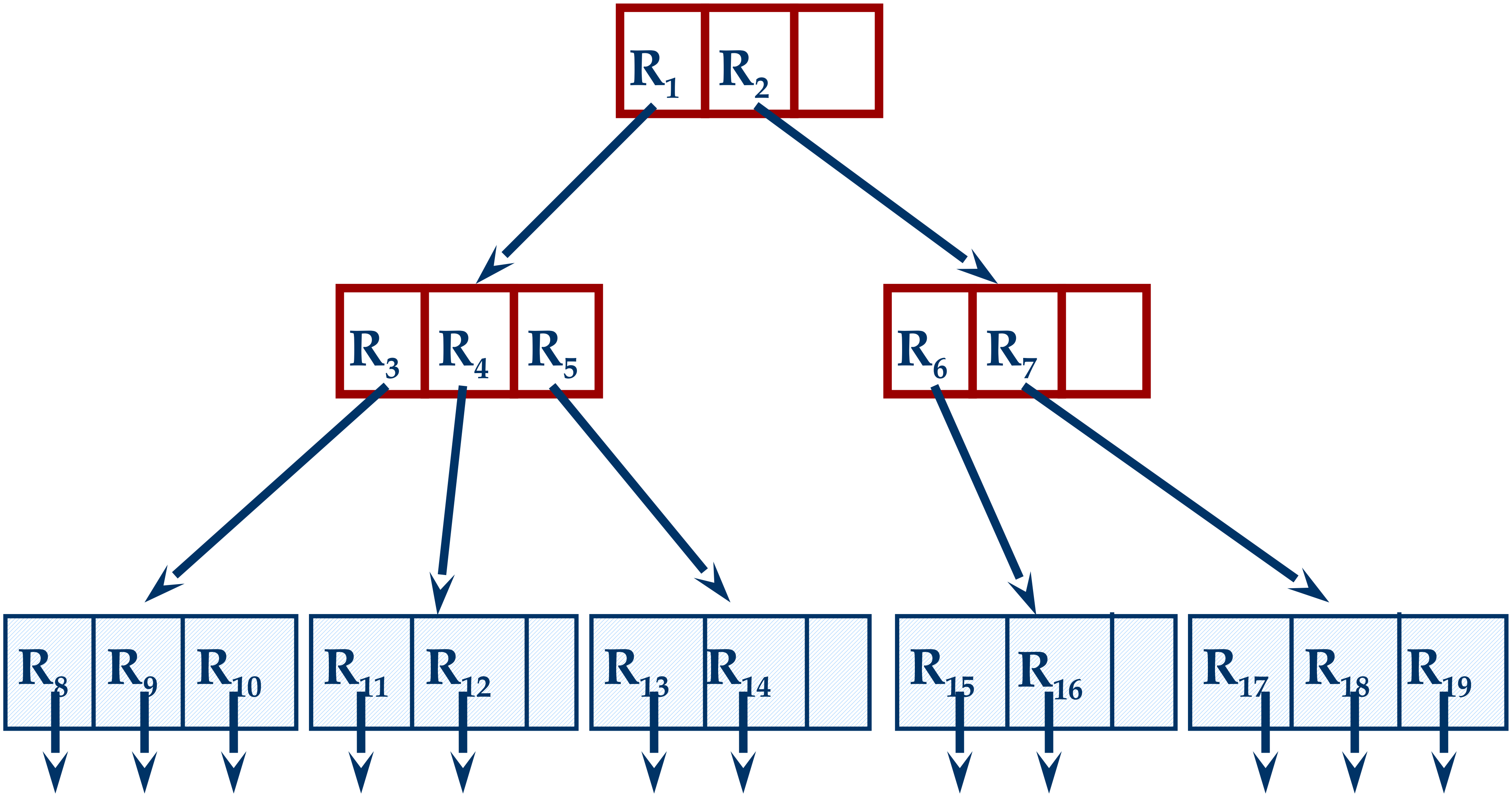
R-Tree Properties

- *Leaf entry* = < n-dimensional box, rid >
  - *key value* being a box.
  - Box is the tightest bounding box for a data object.
- *Non-leaf entry* = < n-dim box, ptr to child node >
  - Box covers all boxes in child node (in fact, subtree).
- All leaves at same distance from root.
- Nodes must be kept at least 50% full (except root).
  - Can choose a parameter *m* that is <= 50%, and ensure that every node is at least *m*% full.

Example of an R-Tree



Example R-Tree (cont.)



Search for Objects Overlapping Box Q

Start at root.

1. If current node is non-leaf, for each entry <E, ptr>, if ***box*** E overlaps Q, search subtree identified by ptr.
2. If current node is leaf, for each entry <E, rid>, if E overlaps Q, rid identifies an object that might overlap Q.

Note: May have to search several subtrees at each node!
(In contrast, a B-tree equality search goes to just one leaf.)

Improving Search Using Constraints

- It is convenient to store boxes in the R-tree as approximations of arbitrary regions, because boxes can be represented compactly.
- But why not use convex polygons to approximate query regions more accurately?
  - Will reduce overlap with nodes in tree, and reduce the number of nodes fetched by avoiding some branches altogether.
  - Cost of overlap test is higher than bounding box intersection, but it is a main-memory cost, and can actually be done quite efficiently. Generally a win.

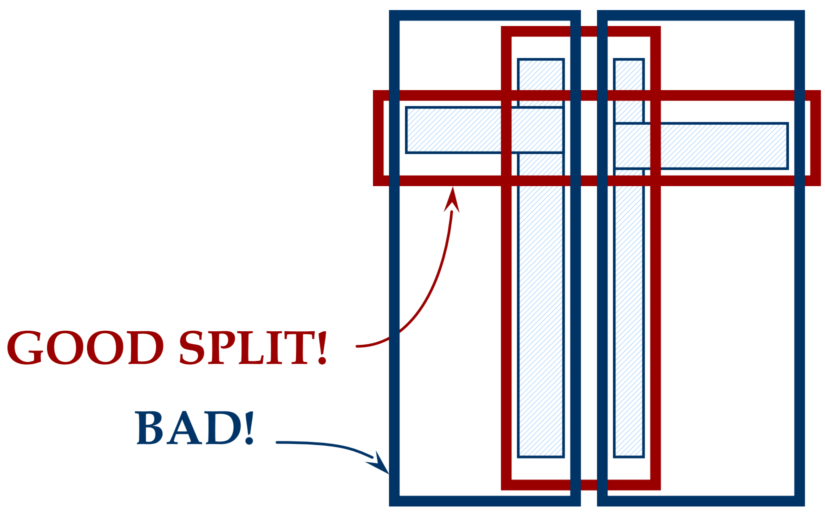
Insert Entry <B, ptr>

- Start at root and go down to "best-fit" leaf L.
  - Go to child whose box needs least enlargement to cover B; resolve ties by going to smallest area child.

- If best-fit leaf L has space, insert entry and stop. Otherwise, split L into L_1 and L_2.
  - Adjust entry for L in its parent so that the box now covers (only) L_1.
  - Add an entry (in the parent node of L) for L_2. (This could cause the parent node to recursively split.)

Splitting a Node During Insertion

- The entries in node L plus the newly inserted entry must be distributed between L_1 and L_2.
- Goal is to reduce likelihood of both L_1 and L_2 being searched on subsequent queries.
- Idea: Redistribute so as to minimize area of L_1 plus area of L_2.



R-Tree Variants

- The R* tree uses the concept of forced reinserts to reduce overlap in tree nodes. When a node overflows, instead of splitting:
  - Remove some (say, 30% of the) entries and reinsert them into the tree.
  - Could result in all reinserted entries fitting on some existing pages, avoiding a split.
- R* trees also use a different heuristic, minimizing box perimeters rather than box areas during insertion.
- Another variant, the R+ tree, avoids overlap by inserting an object into multiple leaves if necessary.
  - Searches now take a single path to a leaf, at cost of redundancy.

GiST

- The Generalized Search Tree (GiST) abstracts the "tree" nature of a class of indexes including B+ trees and R-tree variants.
  - Striking similarities in insert/delete/search and even concurrency control algorithms make it possible to provide "templates" for these algorithms that can be customized to obtain the many different tree index structures.
  - B+ trees are so important (and simple enough to allow further specialization) that they are implemented specially in all DBMSs.
  - GiST provides an alternative for implementing other tree indexes in an ORDBS.

Indexing High-Dimensional Data

- Typically, high-dimensional datasets are collections of points, not regions.
  - E.g., Feature vectors in multimedia applications.
  - Very sparse
- Nearest neighbor queries are common.
  - R-tree becomes worse than sequential scan for most datasets with more than a dozen dimensions.
- As dimensionality increases contrast (ratio of dist. between nearest and farthest points) usually decreases; "nearest neighbor" is not meaningful.
  - In any given data set, advisable to empirically test contrast.

Summary

- Spatial data management has many applications, including GIS, CAD/CAM, multimedia indexing.
  - Point and region data
  - Overlap/containment and nearest-neighbor queries

- Many approaches to indexing spatial data
  - R-tree approach is widely used in GIS systems
  - Other approaches include Grid Files, Quad trees, and techniques based on “space-filling” curves.
  - For high-dimensional datasets, unless data has good “contrast”, nearest-neighbor may not be well-separated

Comments on R-Trees

- Deletion consists of searching for the entry to be deleted, removing it, and if the node becomes under-full, deleting the node and then re-inserting the remaining entries.
- Overall, works quite well for 2 and 3 D datasets. Several variants (notably, R+ and R* trees) have been proposed; widely used.
- Can improve search performance by using a convex polygon to approximate query shape (instead of a bounding box) and testing for polygon-box intersection.