INFS4205/7205 Advanced Techniques for High Dimensional Data

Spatial Data Organization 2

Semester 1, 2021

University of Queensland

+ Advanced Techniques for High Dimensional Data

- Course Introduction
- Introduction to Spatial Databases
- Spatial Data Organization
- Spatial Query Processing
- Managing Spatiotemporal Data
- Managing High-dimensional Data
- Other High-dimensional Data Applications
- When Spatial Temporal Data Meets AI
- Route Planning
- Trends and Course Review

+ Spatial Indexing

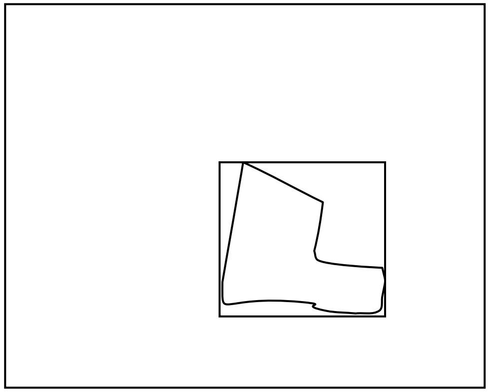
- Purpose:
  - Efficiency in processing spatial selection, join and other spatial operations
- Two strategies to organize space and objects
  - Map spatial objects into 1D space and use a standard index structure (B-tree)
  - Dedicated external data structures
- Basic ideas
  - Approximation
    - Bounding box, Grids
  - Hierarchical Data Organization

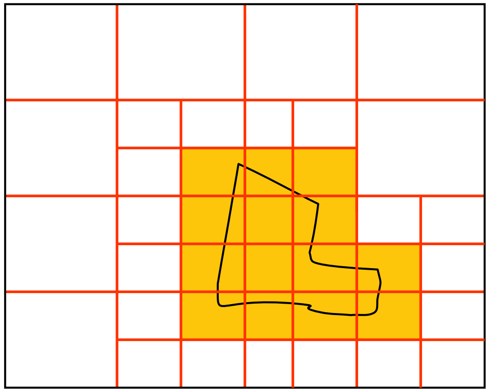
+ Object Approximation

- A fundamental idea of spatial indexing is the use of approximation

- Continuous Approximation
  - Object centric
  - Example:
    - Use of MBRs (Minimum Bounding Rectangles)
    - R-Tree

- Grid Approximation
  - Space centric
    - Faster mapping
    - Uniform / Non-uniform
    - High-D?
  - Example:
    - Quadtree



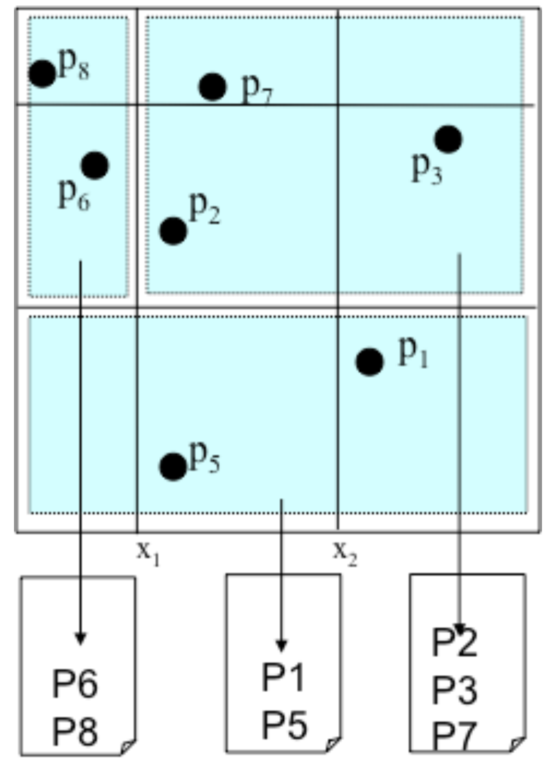


+ Data Access Methods

- One Dimensional
  - Hashing and B-Trees
- Line Data
  - Interval Tree, Segment Tree
- Point Data
  - Hashing: GRID and EXCELL
  - Hierarchical
    - Quadtree: Point and Region Quadtrees
    - kd-Tree
    - Z-values and B-tree
- Polygon Data
  - Transformation: End point mapping and Z-values
  - Overlapping: R-tree and R*-tree
  - Clipping: R^+-tree

+ Grid File

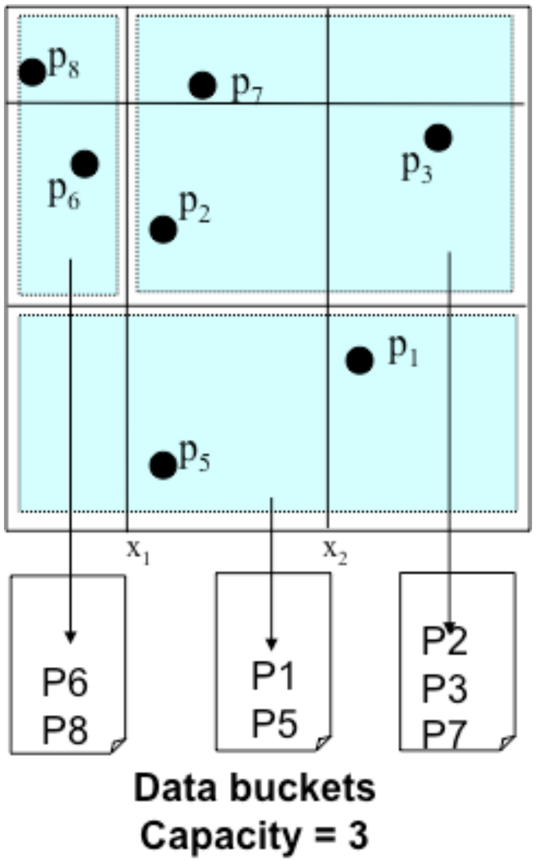
- Basic idea
  - Superimpose a k-dimensional grid on the space
    - Only scan the data in a grid
    - Essentially, a 2D hash function
  - Cells can be of varying sizes
  - Cell-to-bucket mapping: many-to-1
    - *What about 1-to-many?*
  - The grid definition (scales) is kept in memory
  - The grid directory is kept on disk
- Motivation
  - Fixed-grid not suitable for non-uniformly distributed data



Data buckets
Capacity = 3

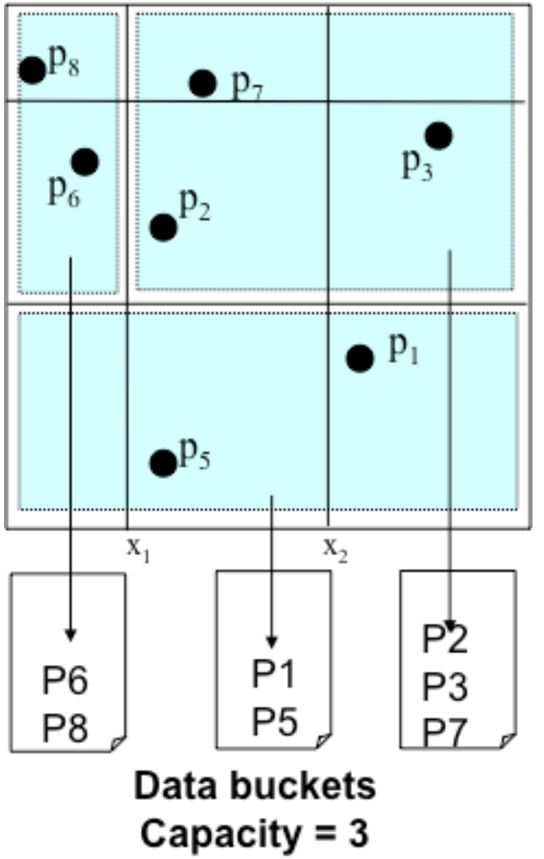
+ Grid File

- To answer a point query:
  - Use the scales / definition to locate the cell
  - Read the cell from disk
  - The loaded cell contains a reference (pointer) to data bucket
  - Read data bucket
  - On average two disk accesses
- To answer a range query:
  - Examine all cells that overlap the search region
  - Read the corresponding data buckets(s)



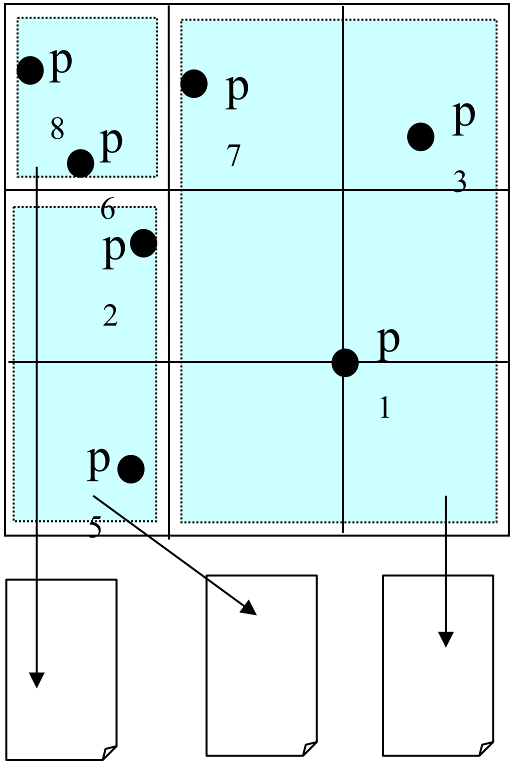
+ Grid File

- To insert a point:
  - Search (point query) the matching cell and data bucket
  - If there is sufficient space, insert into data bucket
  - Else: add a vertical or horizontal line to split, if necessary, and move data accordingly
- To delete a point:
  - Search …
  - Merge if necessary
- Problems:
  - Have to remember all the definitions of the grids
  - Why not uniformly?



+ EXCELL

- EXtendible CELL
- Motivation
  - Fixed grid is easier to manage and more efficient to use
- Basic Ideas
  - All cells are of the same size
    - No need to keep grid definition (scales) in memory
    - But it's still necessary to remember how to map grid cells to buckets
  - Somewhere splits, everywhere splits



Data buckets
Capacity = 3

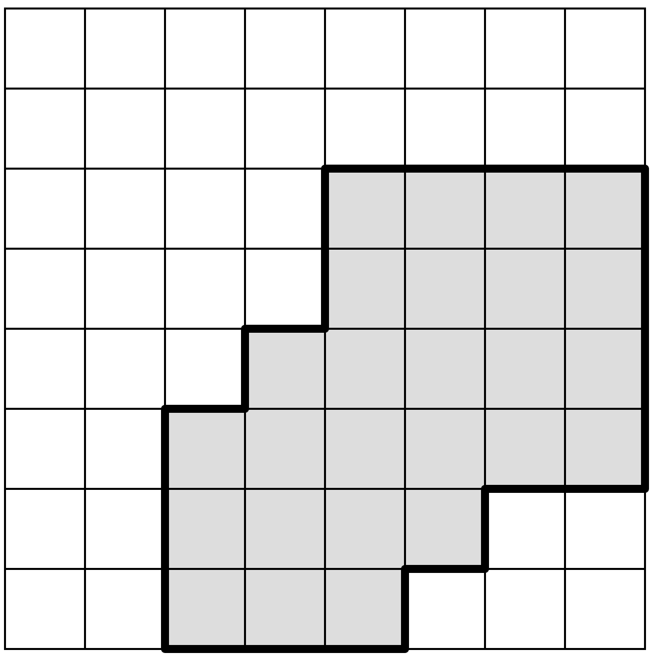
+ Data Access Methods

- One dimensional
  - Hashing and B-Trees
- Line Data
  - Segment Tree, Interval Tree
- Point data
  - Hashing: GRID and EXCELL
  - Hierarchical
    - Quadtree: Point and Region Quadtrees
    - kd-Tree
    - Z-values and B-tree
- Polygon data
  - Transformation: End point mapping and z-values
  - Overlapping: R-tree and R*-tree
  - Clipping: R^+-tree

+ Uniform Decomposition

Recursive decomposition of space

Resolution: max. level of decomposition, leading to $2^n \times 2^2$ cells



To have 1 × 1 cm cells, what is required resolution for:

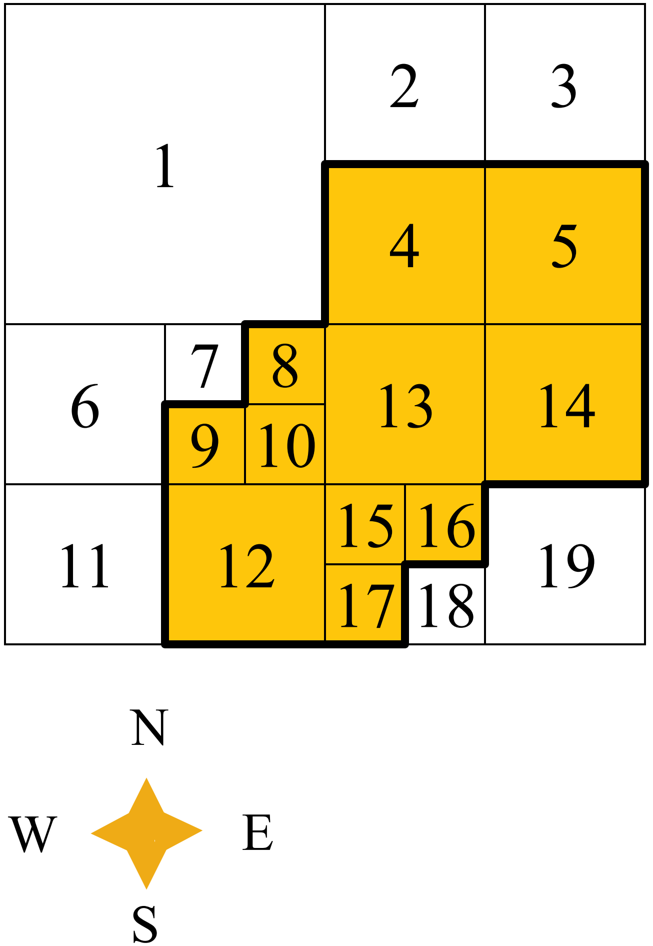
- An area of 5000 x 5000 km^2?
- An area of 300 x 300 km^2?

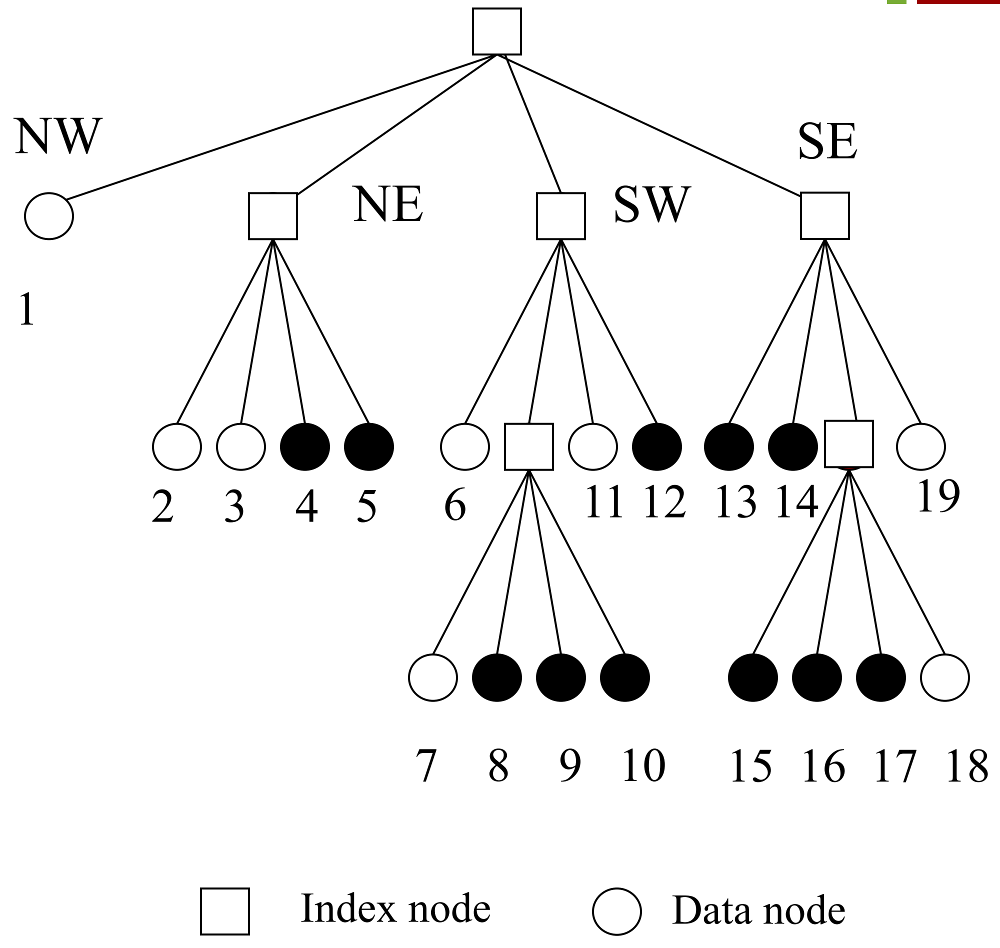
n=29 (25)

How many times do you need to fold a piece of paper to make it reach the moon?

Average thickness of paper sheet = 0.1mm
Distance between earth & moon = 384,403km

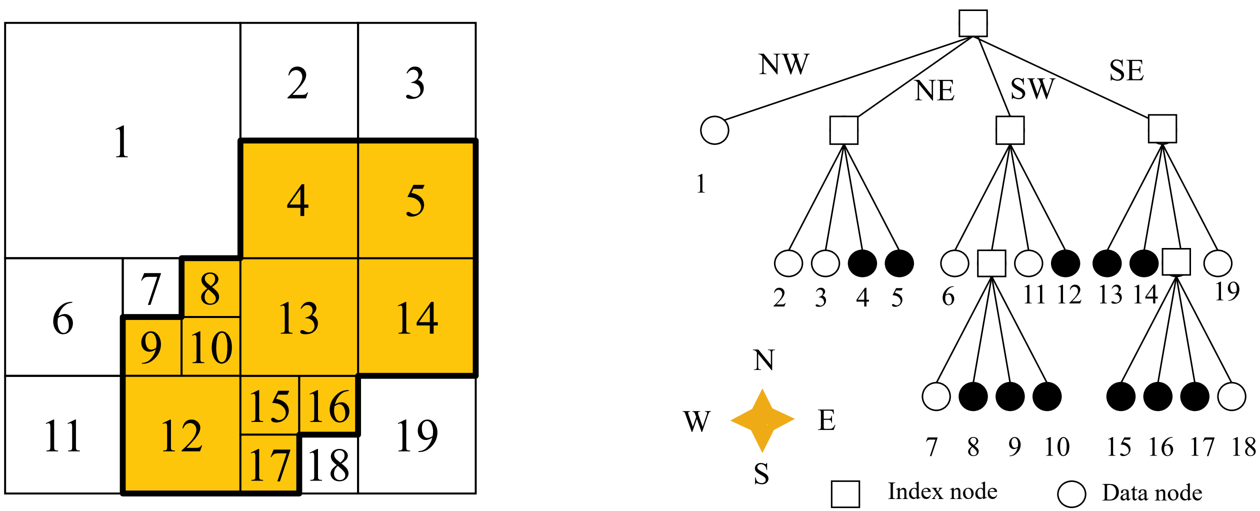
+ Quadtree – Basic Idea





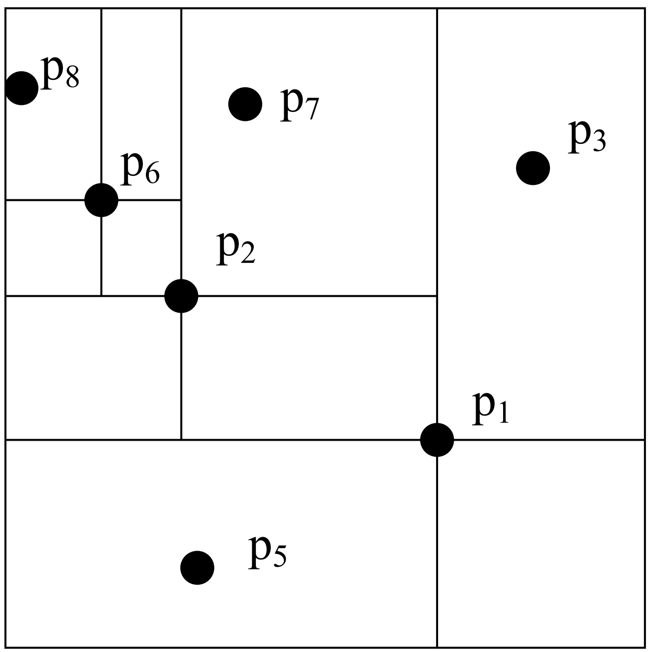
+ Quadtree – Basic Idea

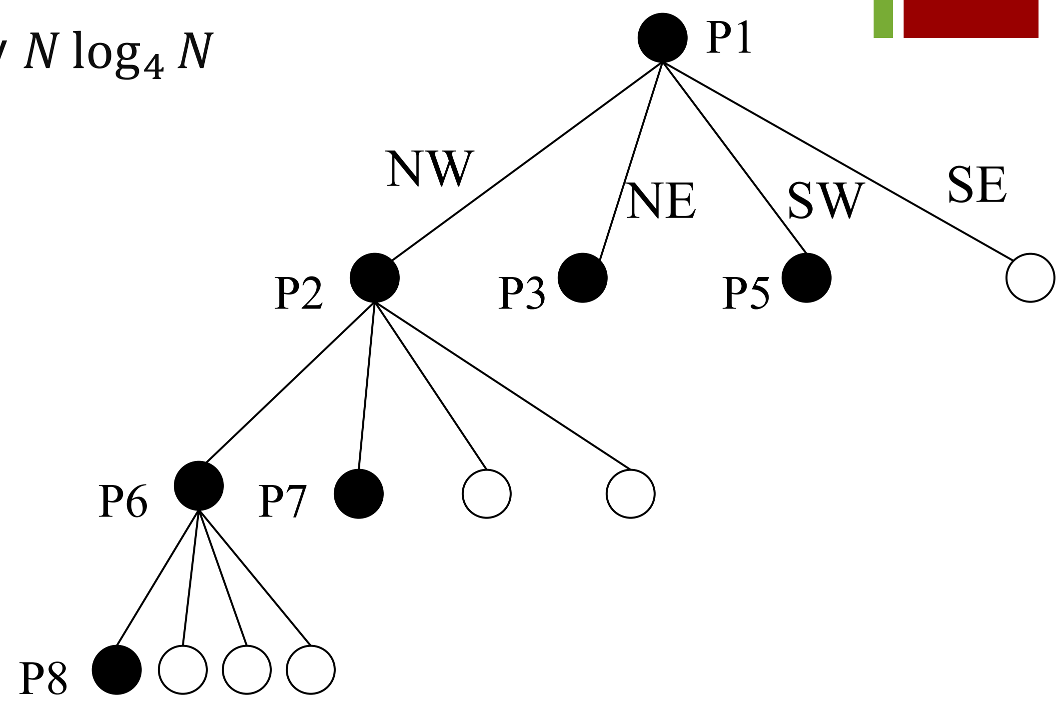
- Not a binary tree
  - Four-way comparison instead of two in 2D
  - In d dimensions, inner node has 2^d children
  - Not necessarily balanced
    - Tree shape depends on the data distribution / insertion order



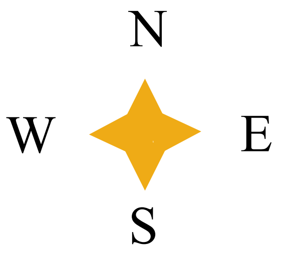
+ Point Quadtree Construction

- Insertion
  - Random insertion roughly $N \log_4 N$





- When is the worst case?
  - Insertion takes $N(N-1)/2$

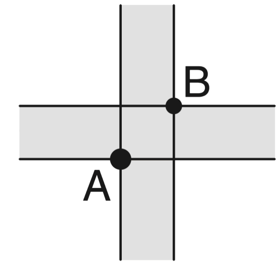


+ Point Quadtree Construction

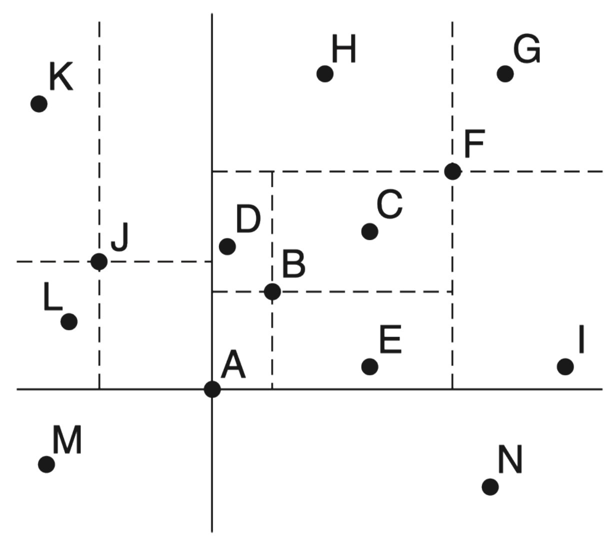
- Optimized Point Quadtree
  - For any tree node A, no subtree of A has more than one-half of the points in the tree rooted at A
  - When all the points are known a priori
  - Sort the points primarily by one key and secondarily by the other
    - Root A takes the median value of the points
    - For example, x is the sorting primary key
      - All the points larger than A lie in NE and SE
      - All the points smaller than A lie in NW and SW
  - How to achieve it dynamically?

+ Point Quadtree Deletion

- Re-insert all points of the sub-tree rooted at the deleted point
  - Simple but expensive

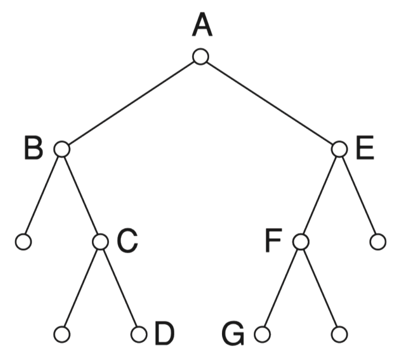


- Replace the deleted node $A\ (x_A, y_A)$
  - With a node $B(x_B, y_B)$ such that the regions between x_A and x_B, and y_A and y_B are empty (hatched)
    - Can be replaced directly without changing the tree structure
    - Large amount of search
    - Sometimes does no exist

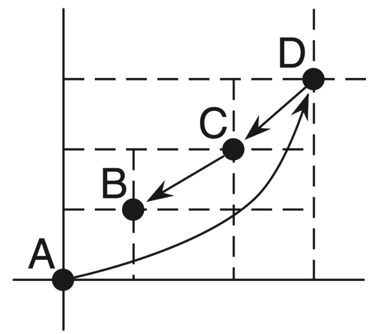


+ Point Quadtree Deletion

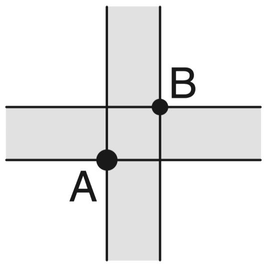
- When delete A in a binary search tree
  - It can be replaced by D or G
    - The two closest nodes in value



- Determine four candidates for the replacement
  - One for each quadrant
  - Opposite quads of children
    - For NE, goes the all the way down by SW
  - Select the “best” candidate
    1. Closer to each of its bordering axes than the others
    2. Minimum L_1 metric value (sum of the distance to the axes)

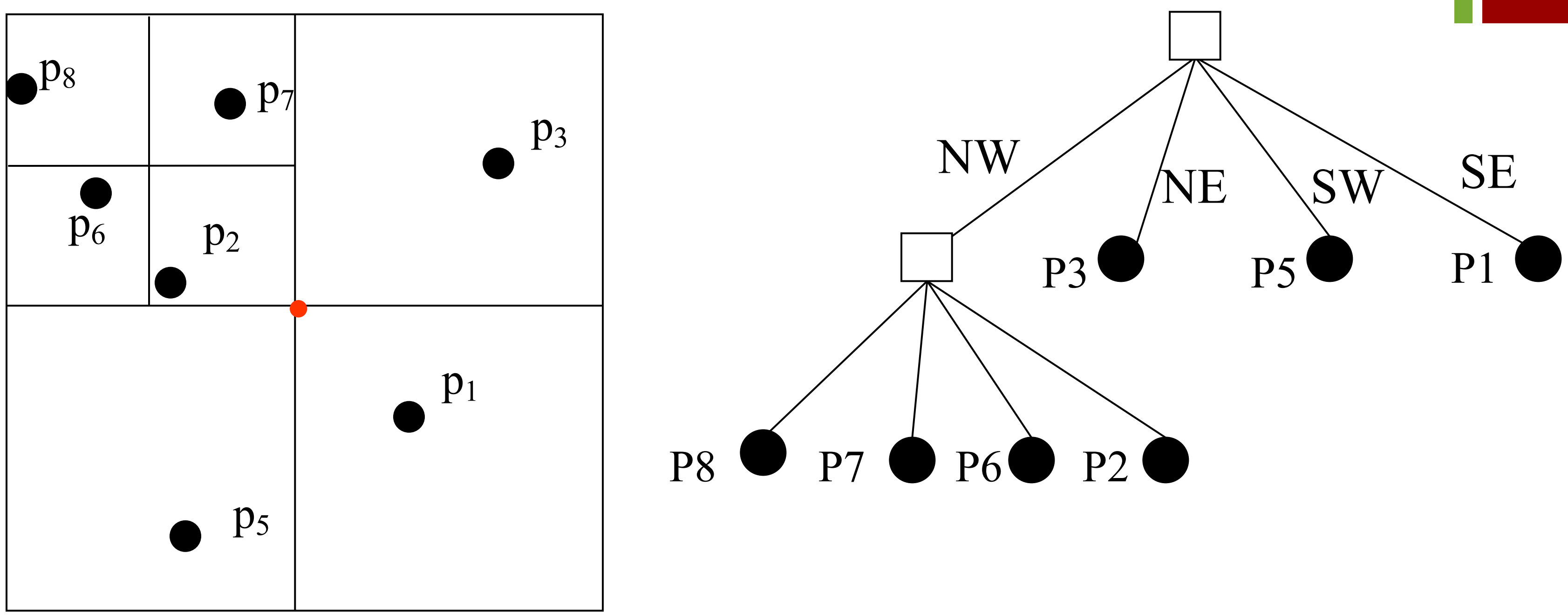


- Re-insert the affected areas



+ Region Quadtree

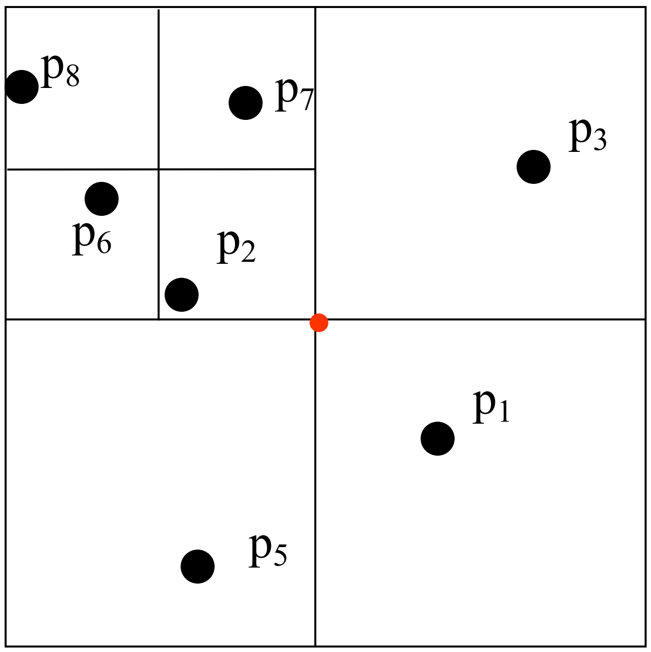
- PR Quadtree

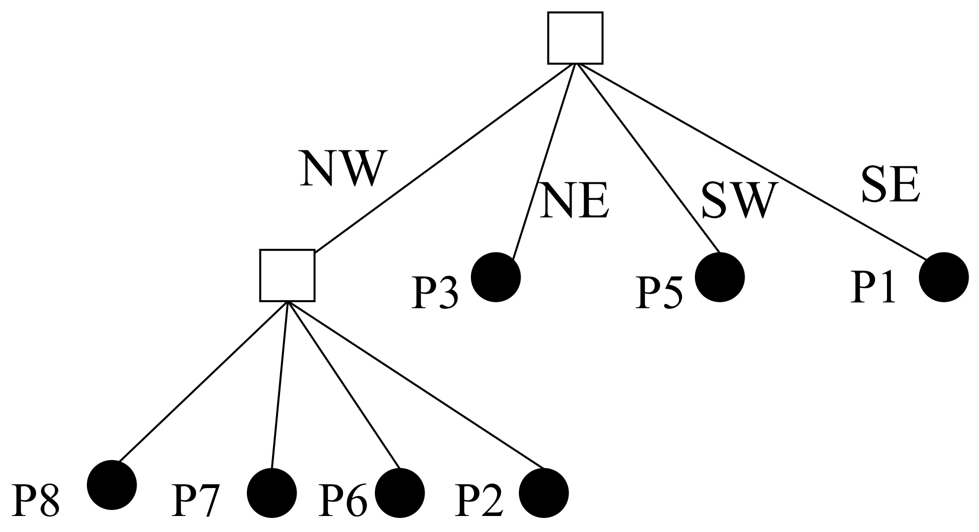


- Based on regular decomposition of the universe
  - Recursively decomposing a region into four congruent blocks
  - Only leaves contain data

Region Quadtree

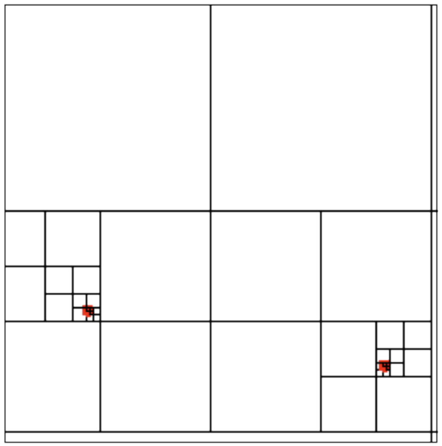
- PR Quadtree Deletion
  - After deletion, if at most one siblings has a point
    - Merge the siblings

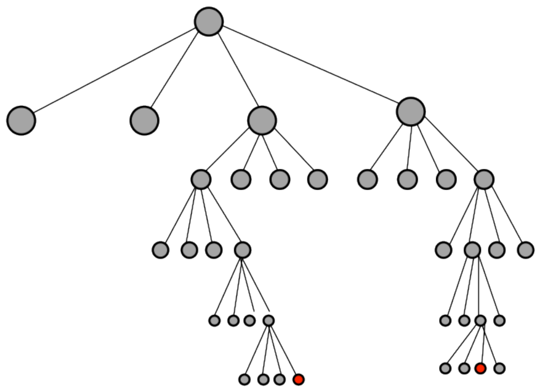




+ Region Quadtree

- MX Quadtree
  - The domain of the point is discrete
    - The overall region is $2^k \times 2^k$
  - All the values are discrete, have the same type and range
    - All coordinates ranging from 0 to $2^n - 1$
    - Treat them as pixels
      - Good for image data
    - Points are always at leaves
    - All the data are at the same level
- How about the order?

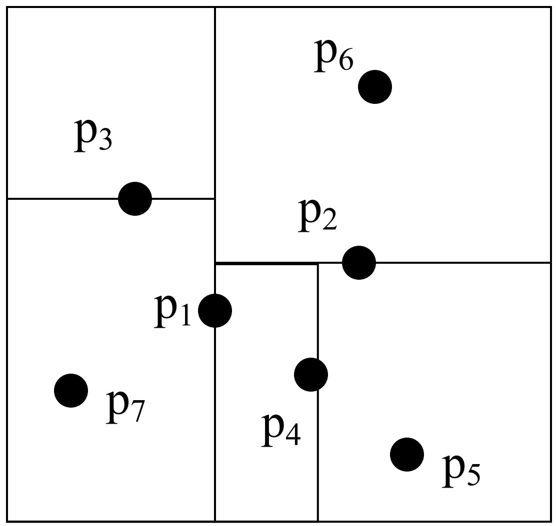


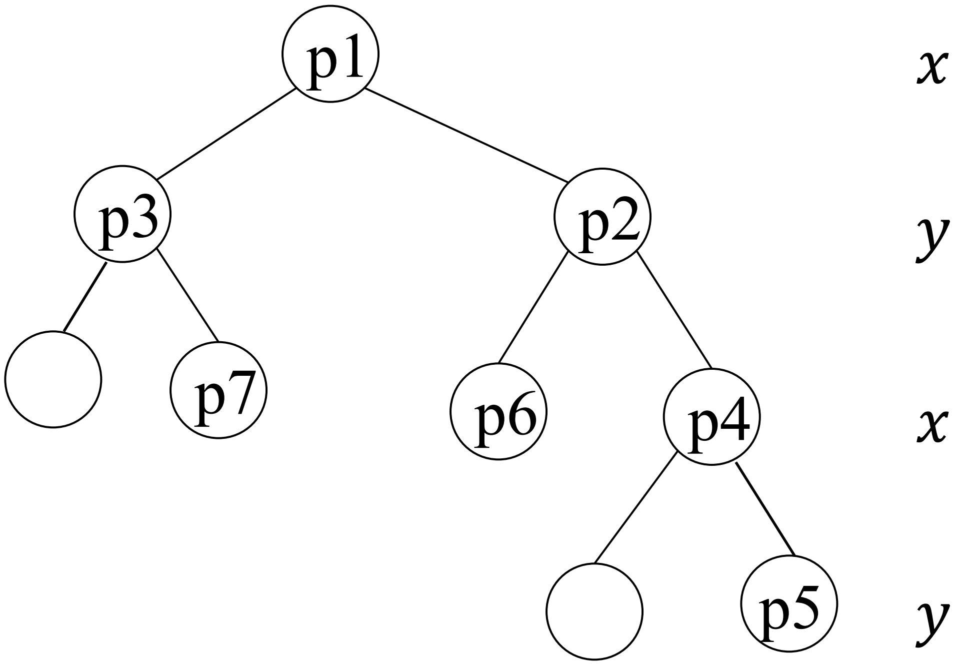


+ kd-Tree (k-dimensional Tree)

- Decomposition at data points (like Point Quadtree)
- Motivation
  - Point (& Region) Quadtree
    - Store k pointers and compares k values for a k-Dimensional space
  - kd-Tree: compare one dimension a time
- Basic Idea
  - Select a dimension, split according to this dimension and do the same recursively with the two new sub-partitions
  - Fan-out is constant (=2) for arbitrary number of dimensions
  - Number of comparisons at each node is constant (=1)
    - Binary search tree
- Problem
  - The resulting binary tree is not adequate for secondary storage (i.e, *one data item per node*)

+ kd-Tree Construction





Depends on the order of insertion (not robust for sorted data).

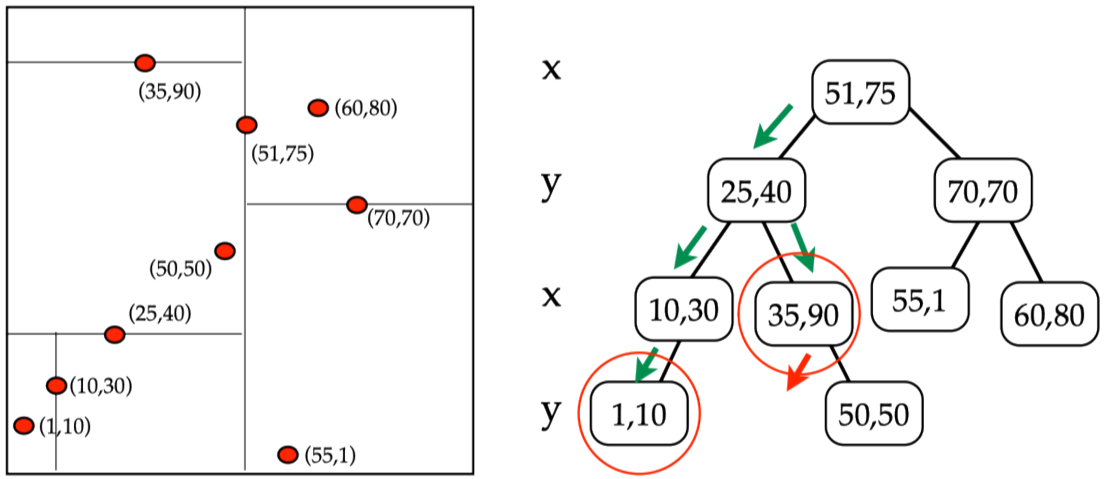
Variations: non-alternative, data at leaves only, representing regions etc.

+ kd-Tree FindMin

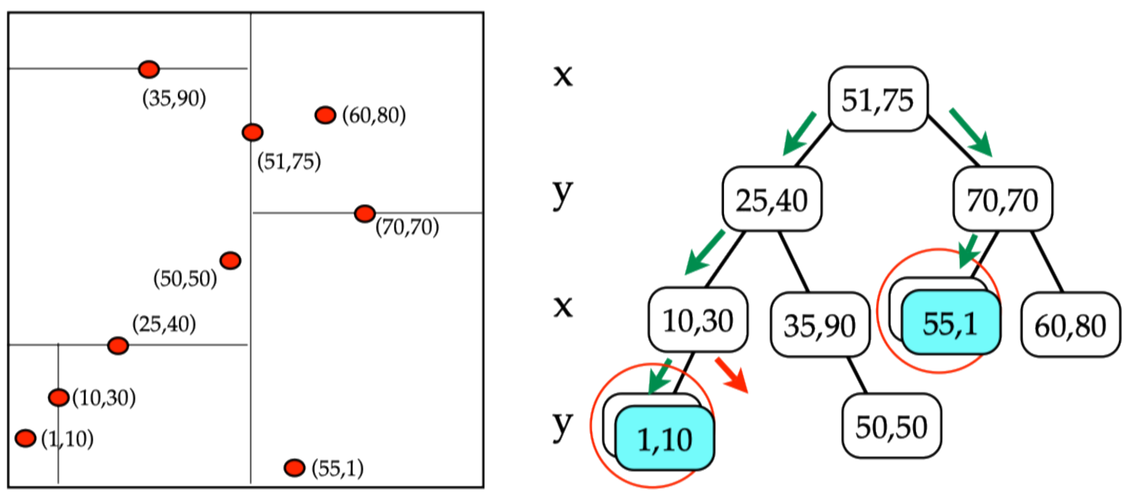
- $FindMin(d)$
  - Find the point with the smallest value in the d^{th} dimension
  - If $cutDim(node) = d$
    - The minimum can't be in the right subtree
      - Recurse on the left subtree
    - If no left subtree
      - The current node is the min for the tree rooted at this node
  - If $cutDim(node) \neq d$
    - The minimum could be in either subtree
      - Recurse on both subtrees

+ kd-Tree FindMin

- $FindMin(x)$



- $FindMin(y)$



+ kd-Tree: Deletion

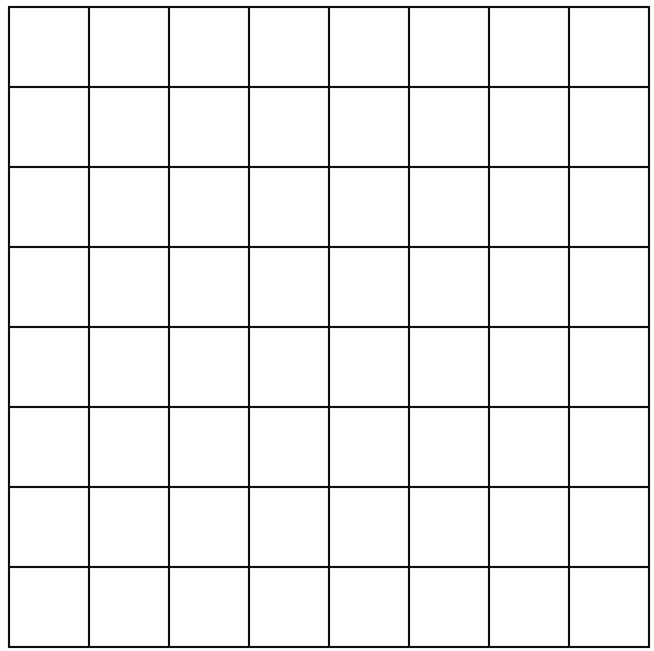
- To remove a node on a level with discriminator along dimension j (suppose $<$ go to left and $\geq$ go to right)
  - If the node is a leaf, remove it
  - Else if node has right subtree
    - Find the j-minimum node in the right subtree
      - Replace node with j-minimum node and repeat until you reach a leaf, then remove the leaf
  - Else find the j-maximum node in the left subtree, replace, repeat, remove ?
    - This will cause problems if there are duplicate coordinates in p's left subtree
      - Compute the j-minimum from left subtree as replacement
      - Make left subtree the new right subtree

Example in Tutorial

+ Multidimensional Data

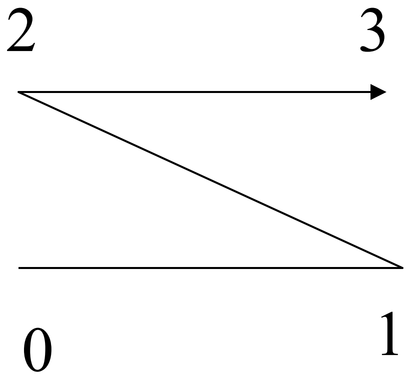
- There is no total order that preserves spatial proximity
- **Solution:**
  - Find heuristic solutions: total orders that preserve spatial proximity to some extent
- **Idea:**
  - If two objects are located close together in original space (k-dimensional), they should be close together in one-dimensional space (with high probability)
  - Balancing for one-dimensional data is well known (B/B+ tree)

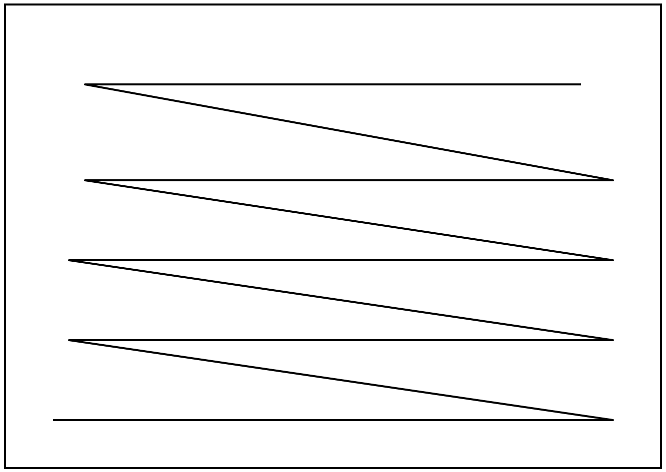
+ Space-Filling Curves (I)



+ Space-Filling Curves (II)

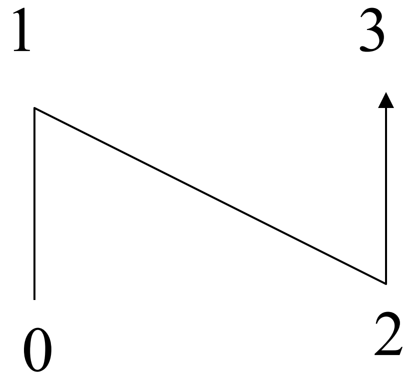
- Row-order

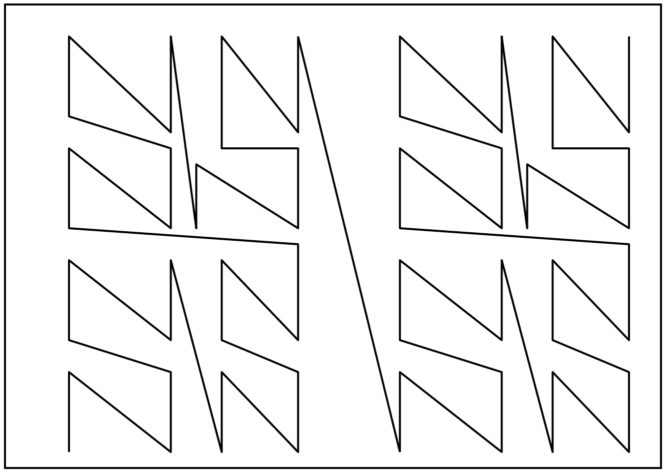




+ Space-Filling Curves (III)

- Z-order (Peano Order)





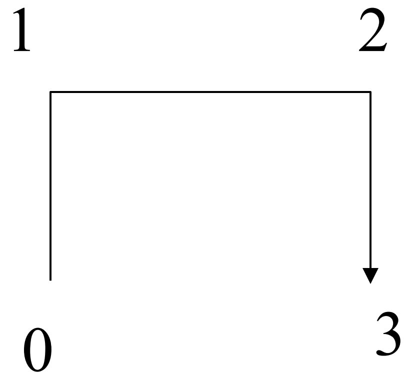
- Easy and elegant way to encode cells.

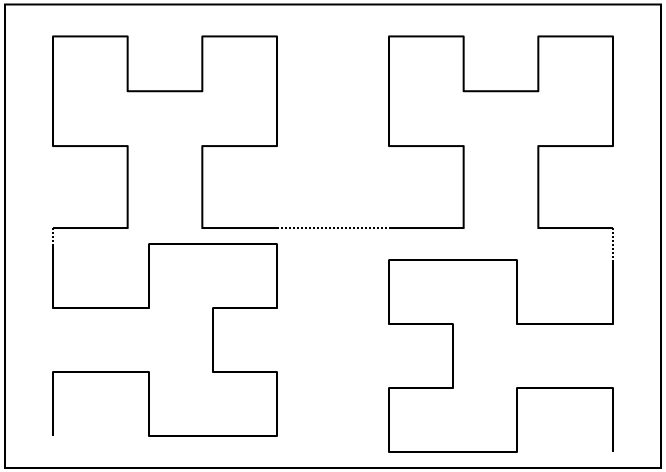
- SIRO-DBMS (SDM) and Oracle use this order.

Giuseppe Peano (/piˈɑːnoʊ/) was an Italian mathematician (1858 – 1932).

+ Space-Filling Curves (IV)

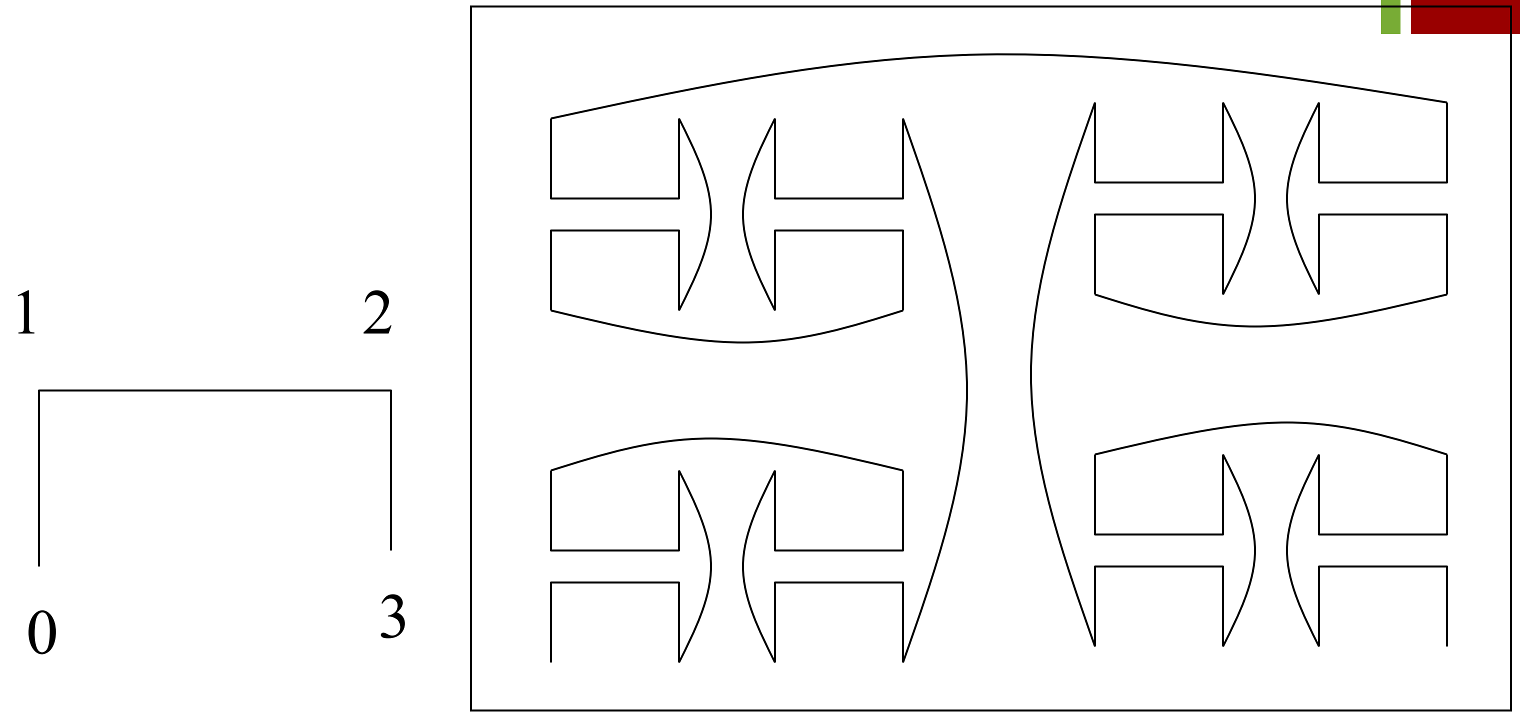
- Hillbert Order





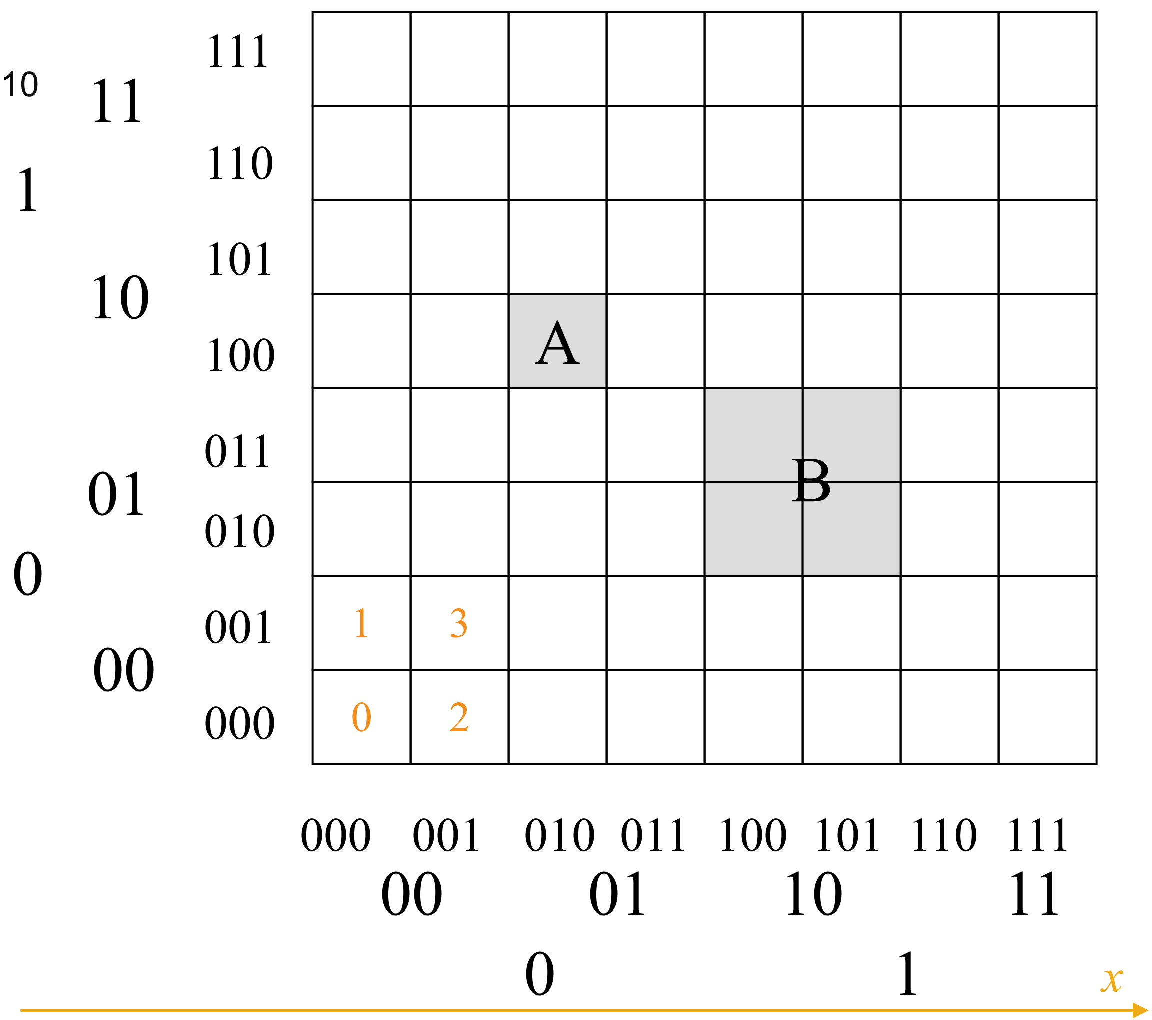
+ Space-Filling Curves (V)

- Gray order



+ Z-Order

- How to obtain the z-order?
  1. Counting: A is 24
  2. Quaternary: $(120)_4 = (24)_{10}$
  3. Bit-Interleaving
     - $x_0 y_0 x_1 y_1 \dots$
     - $(011000)_2 = (24)_{10}$
     - Works fine with varying resolutions
     - B: $(21)_4$
     - $(1001)_2$



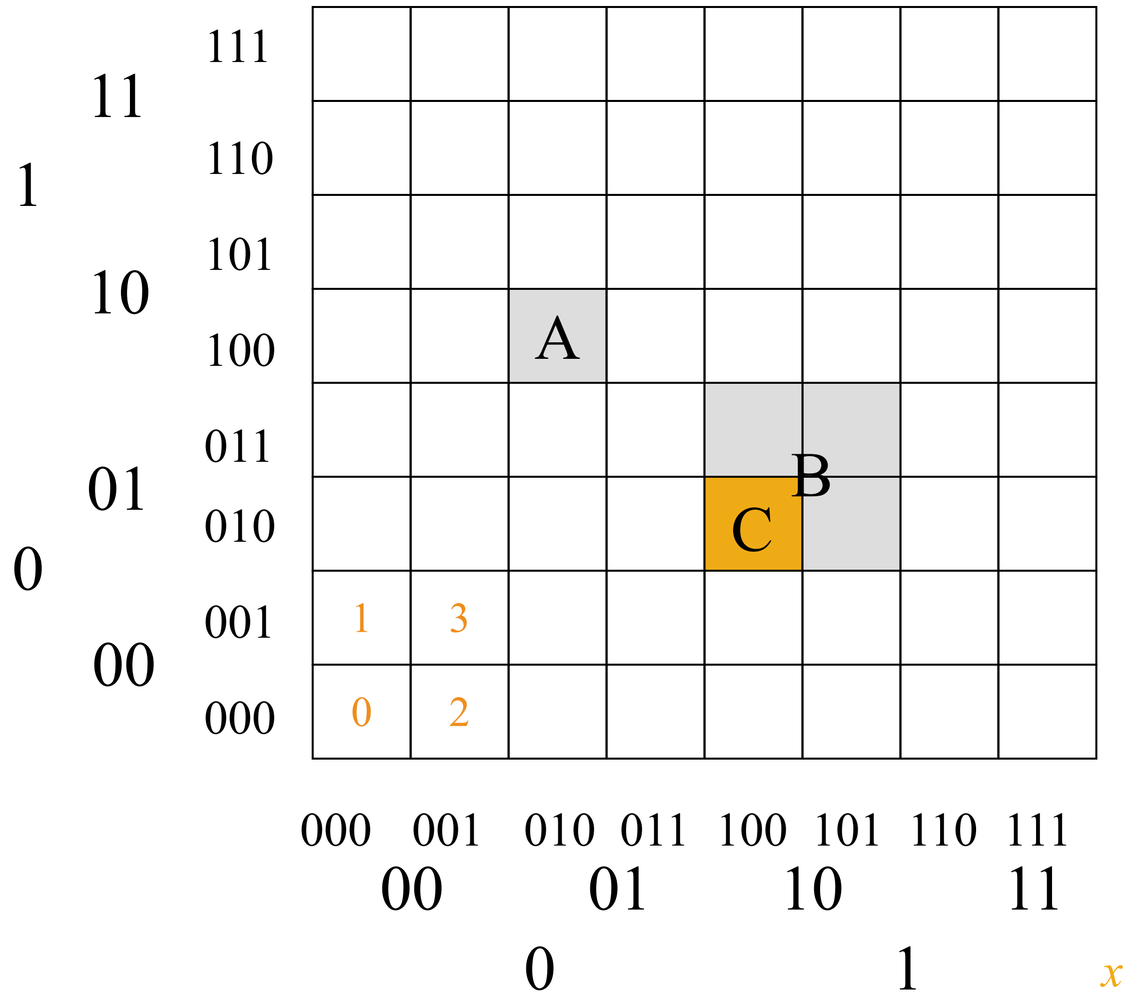
Some Properties of Z-Values

- Variable length
  - Approximate at different levels
  - Appending ‘0’s at the end to unify z-value length
    - Ambiguous!
    - From base4 to base5
      - 0123 to 12340

- Nesting Peano Cells
  - $a = a_1 a_2 a_3 .. a_n$
  - $b = b_1 b_2 b_3 \ldots b_n$
  - a is nested inside b if and only if
    - $length(a) \geq length(b), length(a)$ is the number of non-zero digits in z-value *a*
    - let $k = length(a), a_i = b_i, 1 \leq i \leq k$

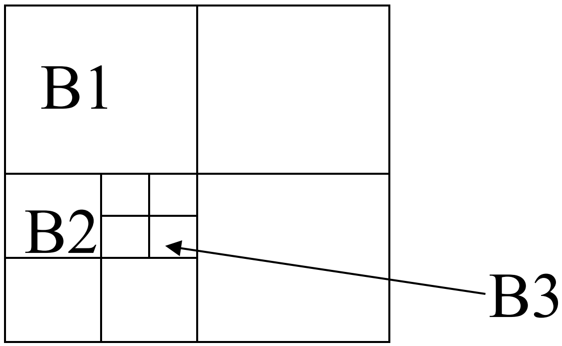
+ Z-Value Example

- B Covers C (Base 4)
  - B: 21
  - C: 210
- B covers C (Base 5)
  - B: 320
  - C: 321



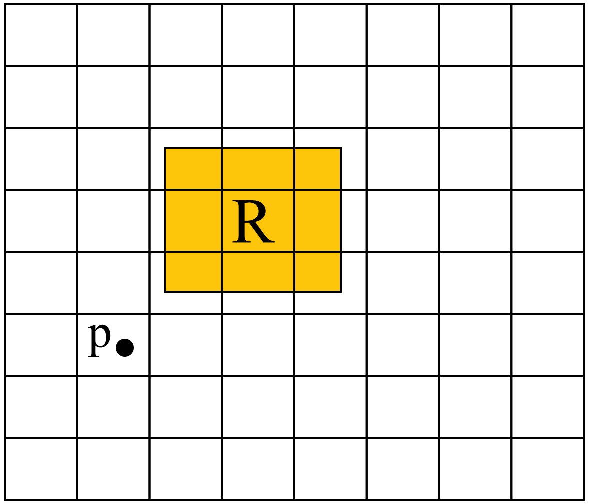
+ Using Z-Values and B-Tree (I)

- Motivation
  - Use standard B-tree to manage multidimensional data
- Basic Idea
  - A Peano cell corresponds to a bucket
  - Peano cells are of varying sizes
  - Z-values are managed by B-tree



+ Using Z-Values and B-Tree (II)

- Search
  - Point query: find the z-value for the unit Peano cell containing point p
  - Range query: find the min and max z-values for rectangle R *(or the z-values approximating R)*
- Insertion and deletion
- Compatibility of z-value indices
  - Origin and orientation
  - Spatial extent
  - Resolution

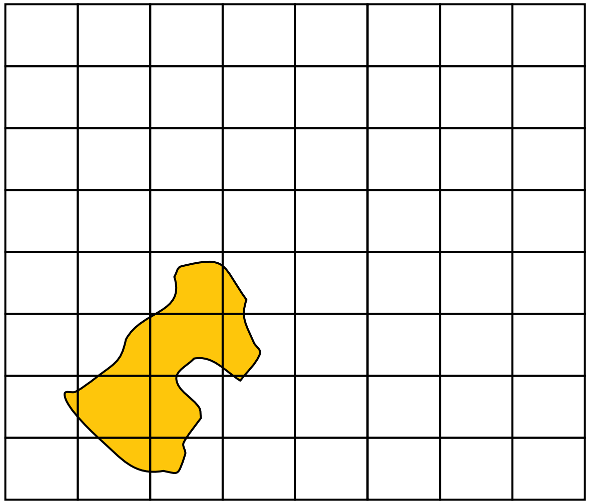


+ Data Access Methods

- One dimensional
  - Hashing and B-Trees
- Line Data
  - Interval Tree, Segment Tree
- Point data
  - Hashing: GRID and EXCELL
  - Hierarchical
    - Quadtree: point and region quadtrees
    - kd-Tree
    - Z-values and B-tree
- Polygon data
  - Transformation: End point mapping and Z-values
  - Overlapping: R-tree and R*-tree
  - Clipping: R^+-tree

+ Indexing Objects with Spatial Extent

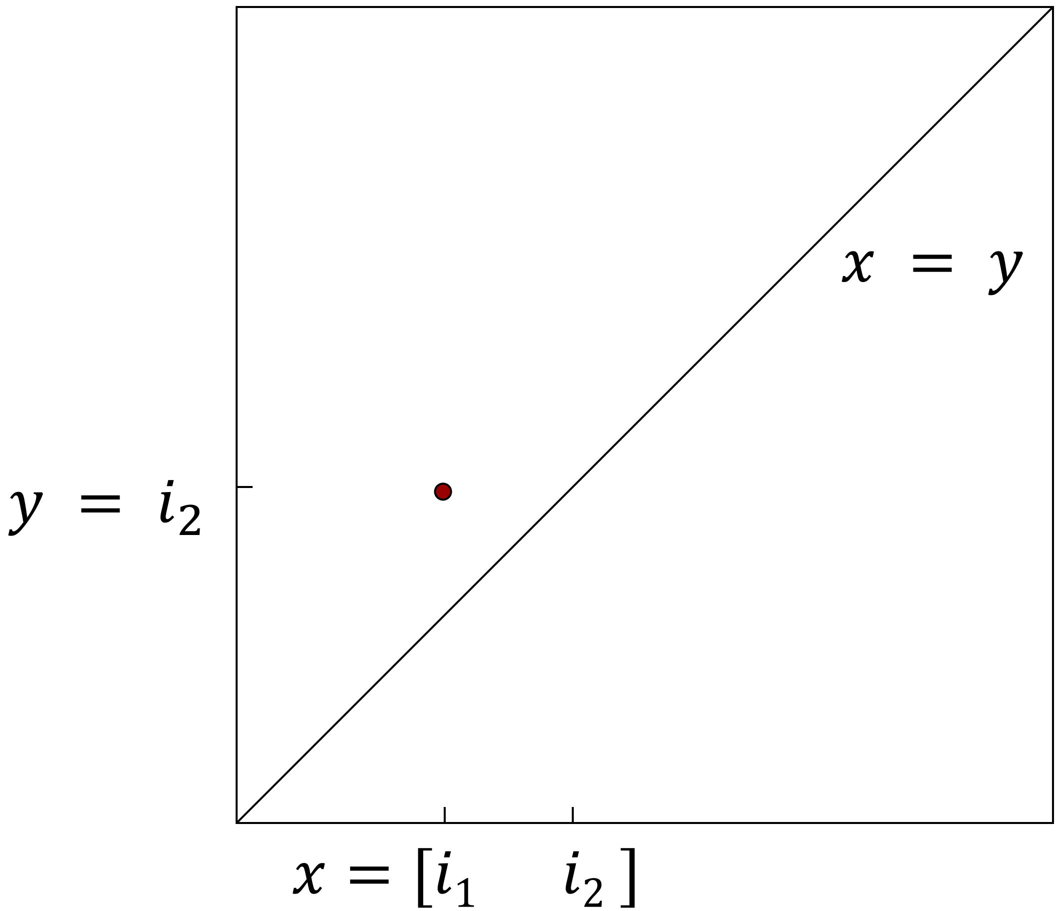
- Rectangles more difficult than points as they do not fall into a single cell of a bucket partition.
- Three strategies
  - Transformation: End point mapping and Z-values
  - Overlapping: R-tree and R*-tree
  - Clipping: R⁺-tree



+ Transformation: High Dimensional Points

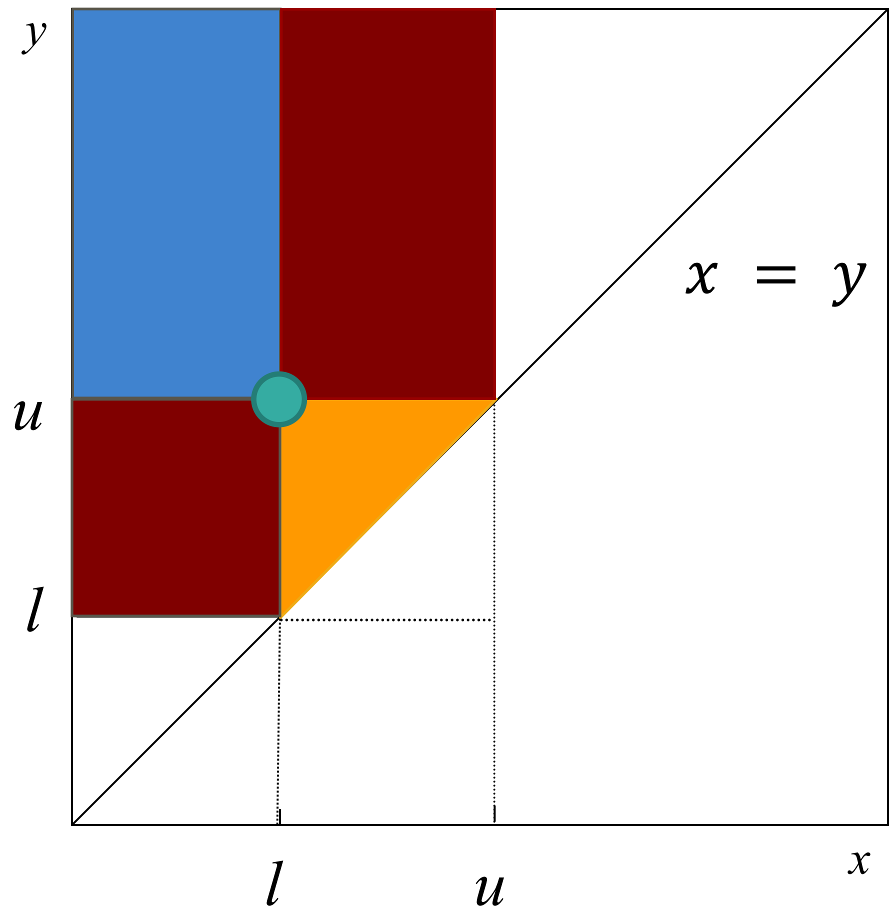
- Motivation
  - Points are easy to manage
- Basic Ideas
  - A rectangle in 2-D space can be mapped to a point in 4-D space
  - Using point access methods
- Two methods
  - Endpoint mapping, or midpoint mapping
    - $(x_{low}, y_{low},)\ (x_{high}, y_{high}) \rightarrow (x_{low}, x_{high}, y_{low}, y_{high})$
    - $(x_{center}, x_{ext}), (y_{center}, y_{ext}) \rightarrow (x_{center}, x_{ext}, y_{center}, y_{ext})$

+ Endpoint Mapping



- Every range in 1-D space becomes a point in 2-D space
- $x \leq y$ for the ranges, no point in the lower triangle

+ Query Processing Using Endpoint Mapping

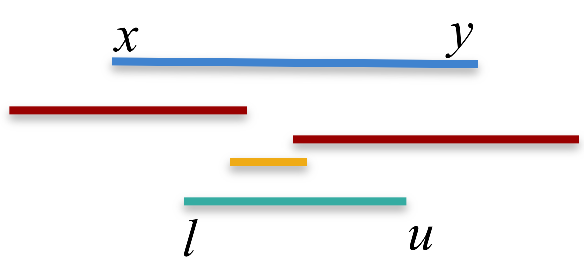


Given x-interval (l, u):

1) *Intersection query*: find all x-intervals overlapping with (l, u)

2) *Containment query*: find all x-intervals inside (l, u)

3) *Enclosure query*: find all x-intervals enclosing (l, u)



Very intuitive, but…

+ Problems with Endpoint Mapping

- Points in the higher-D space are highly skewed
  - Not distributed evenly in the space
    - Only half, mapped into higher but smaller space
- Almost no relationship between the distances of two objects in the original space and the higher-D space
- A simple, intuitive query in the original space becomes complex and difficult to understand in the higher-D space
- Query processing in the higher-D space less efficient

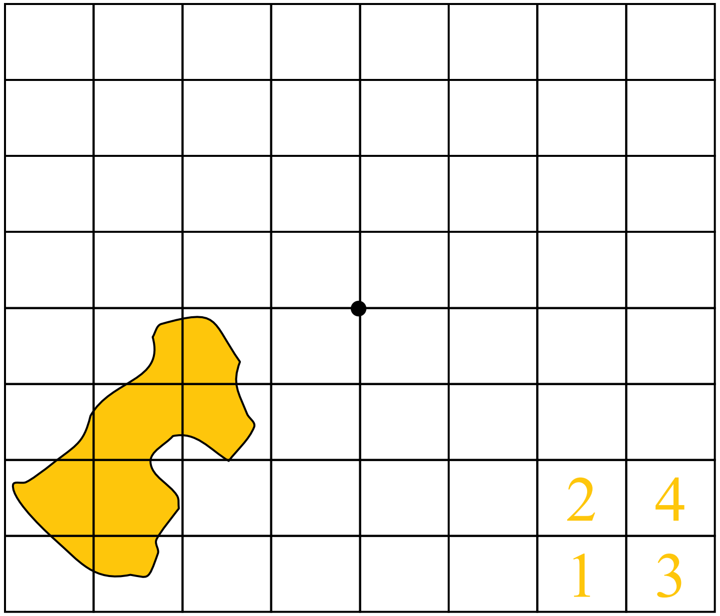
...conceptually very simple, but...

+ Transformation: Using Z-Ordering

- Motivation
  - Still using point access methods, but without drawbacks of the previous approach
- Basic Ideas
  - Instead of mapping a polygon into a point, decompose a polygon into a set of Peano cells and map each Peano cell into a number (i.e., z-value)
  - Reverse of end point mapping: Higher D to lower D

+ Transformation: Using Z-Ordering

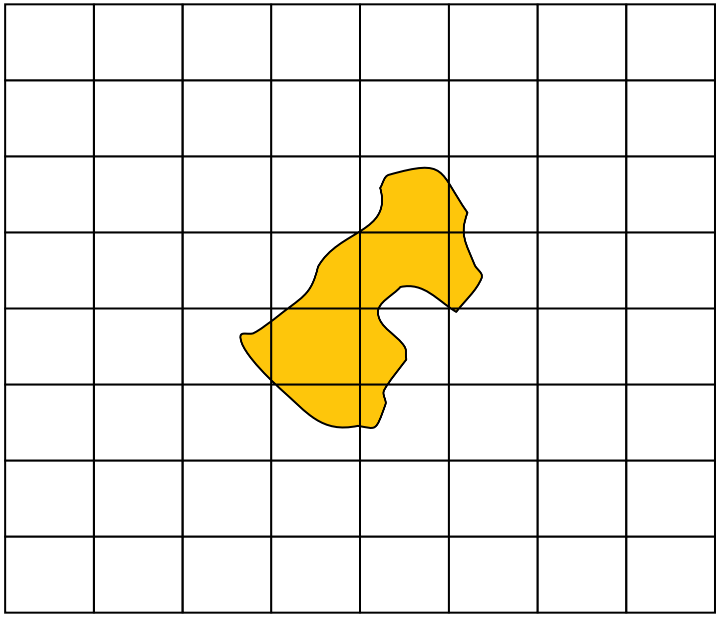
- Granularity
  - {11}, or {111, 112, 114}, or {111, 1121, 1123, 1124, 1141, 1142}
- When decomposition stops
  - Current cell either fully out or in the polygon
  - Reached the "resolution"



...the entire space is 1.

+ Redundancy in Z-Ordering

- Finer granularity
  - √ Improves approximation accuracy
  - √ Can reduce the number of “false hits”
  - × Too many index entries degrade query performance because of inflated index table
  - × May identify the same object multiple times in spatial query processing

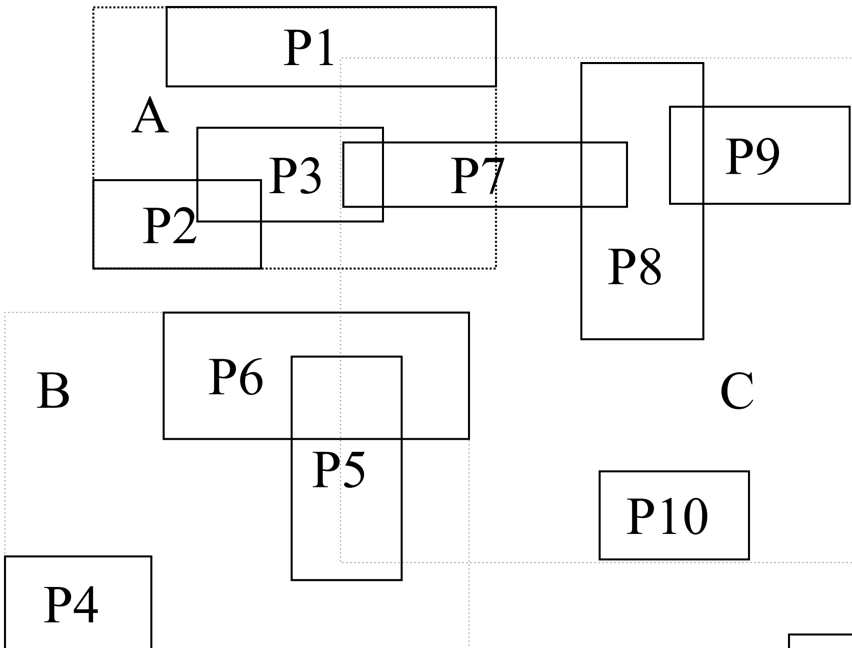


- The 4-Key method (Compromise)
  - Any objects, use no more than 4 values
    - Intuitively, a small object in the very middle, what happens?

+ Overlapping Regions

- Motivation
  - Single index entry for a polygon
- Basic ideas
  - One object (or its key) in one bucket only
  - Cell boundary calculated according to polygons inside the cell
  - Allow overlapping cells: *inevitable*!
- Problems
  - Multiple cells need to be examined to search an object
  - Where to insert?

+ R-Tree and R*-Tree

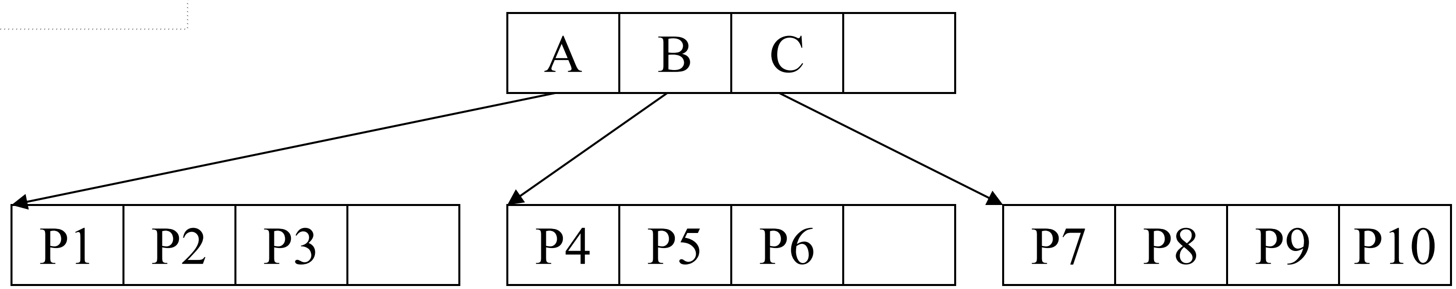


A node must have more than m, less than M elements.

Many different strategies for:

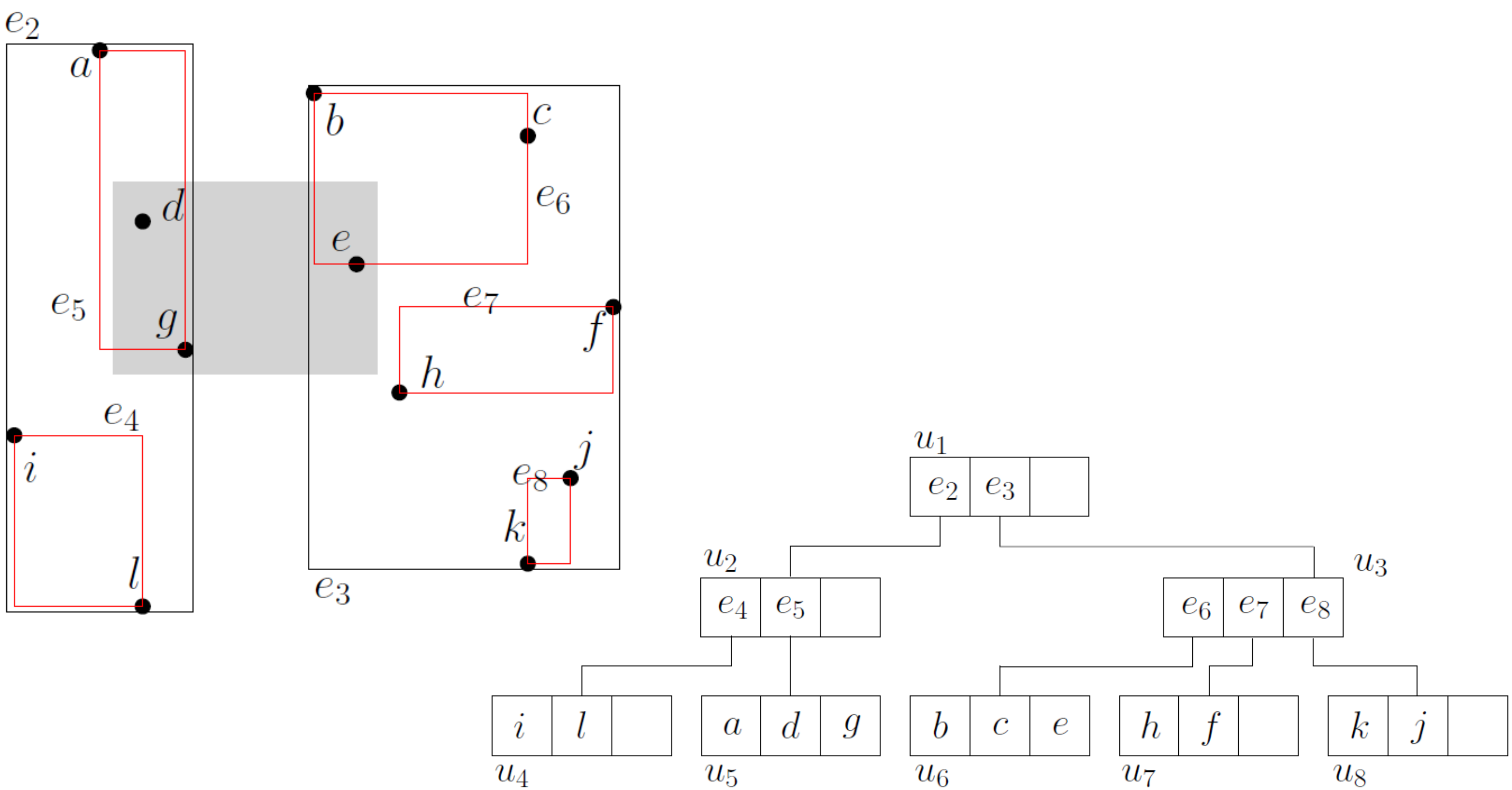
- Insertion → Split
- Deletion → Reinsert

what info recorded in a node?



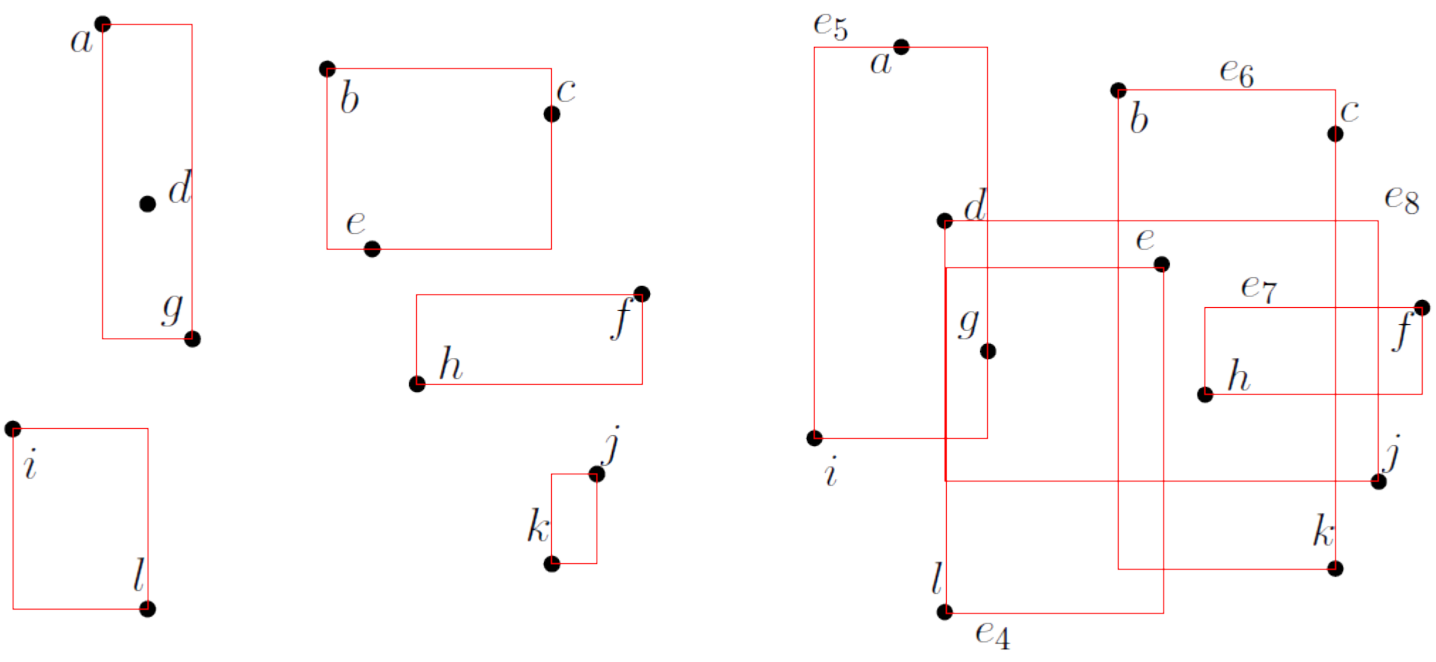
+ R-Tree Range Query

- u_1, u_2, u_3, u_5, u_6 are accessed



+ R-Tree Construction

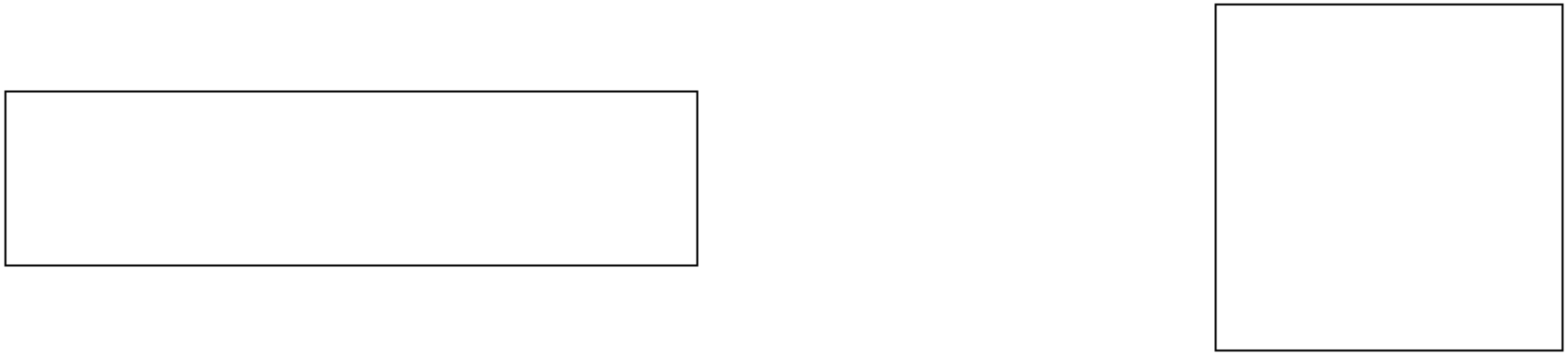
- R-Tree construction can be "arbitrary"
  - Bottom-up
  - No formal constraint on the grouping of data into nodes



- The left tree has a smaller perimeter sum than the right one

+ R-Tree Construction

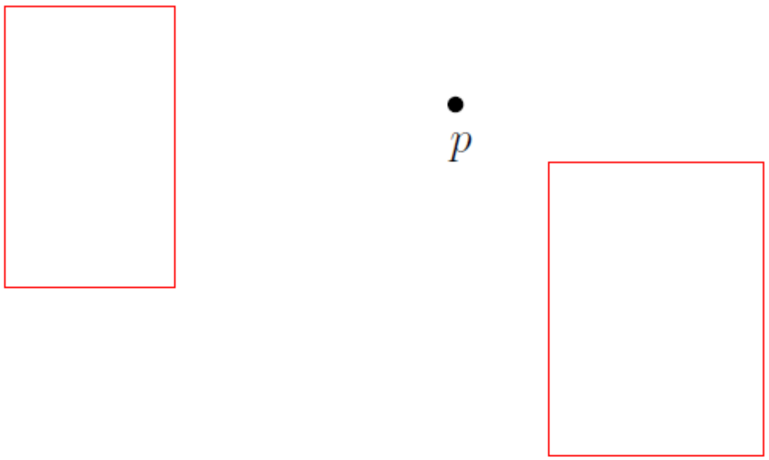
- Why not minimize the area?
  - A rectangle with a smaller perimeter usually has a smaller area, but not the vice versa



+ R-Tree Insertion

- $Insert(u, p)$

1. If u is a leaf node
2.   add p to u
3.   if u overflows
4. $handle - overflow(u)$
5. else
6. $v \leftarrow choose - subtree(u, p)$
7. $insert(v, p)$



- Which MBR would you insert p into?
  - The MBR with the minimum increase
- How to handle the overflow?

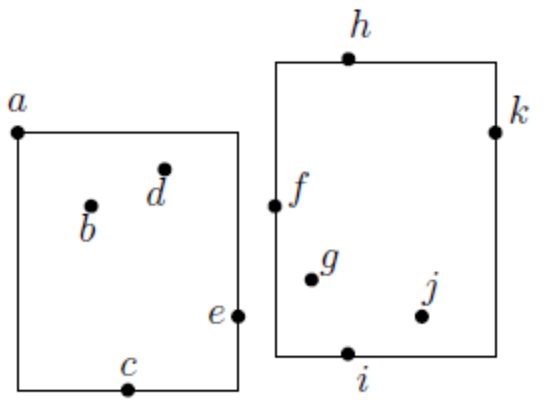
+ R-Tree Handle Overflow

- $handle - overflow(u)$
  1. $Split(u)$ into u and u'
  2. If u is the root
  3. create a new root with u and u' as its child nodes
  4. else
  5. $w \leftarrow$ the parent of u
  6. update $MBR(u)$ in w
  7. add u' as a child of w
  8. if w overflows
  9. $handle - overflow(w)$

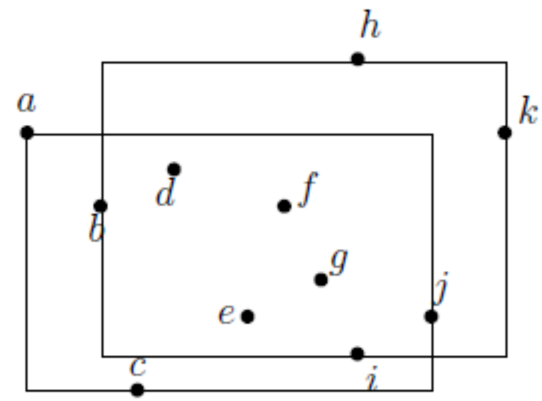
- How to split?

+ R-Tree Splitting a Leaf

- Let S be a set of $B + 1$ points
  - Divide S into two disjoint sets S_1 and S_2 to minimize the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$
  - $|S_1| \geq 0.4B$, $|S_2| \geq 0.4B$



$$S_1 = \{a, b, c, d, e\}$$
$$S_2 = \{f, g, h, i, j, k\}$$

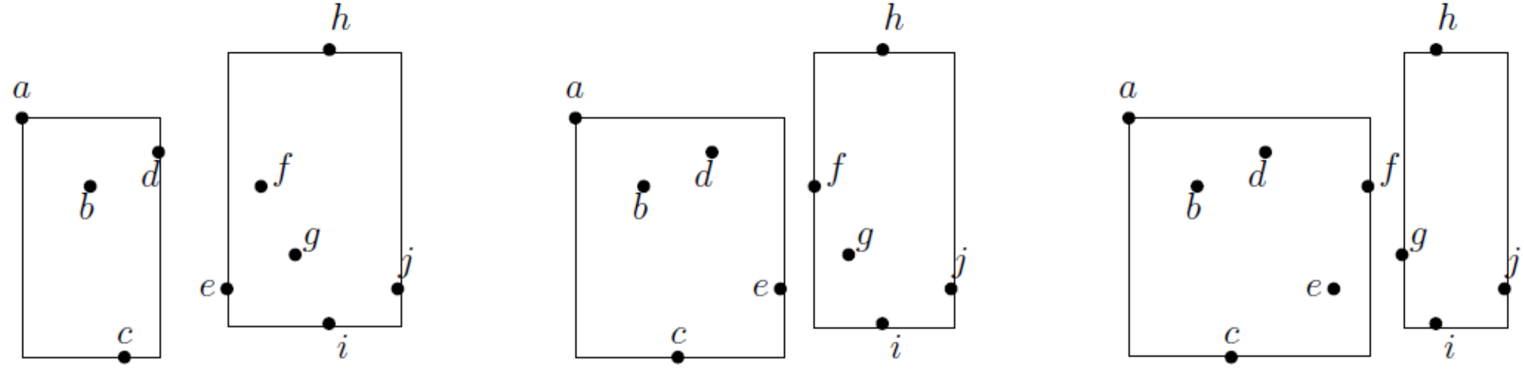


$$S_1 = \{a, d, e, g, j\}$$
$$S_2 = \{b, c, f, h, i, k\}$$

+ R-Tree Splitting a Leaf

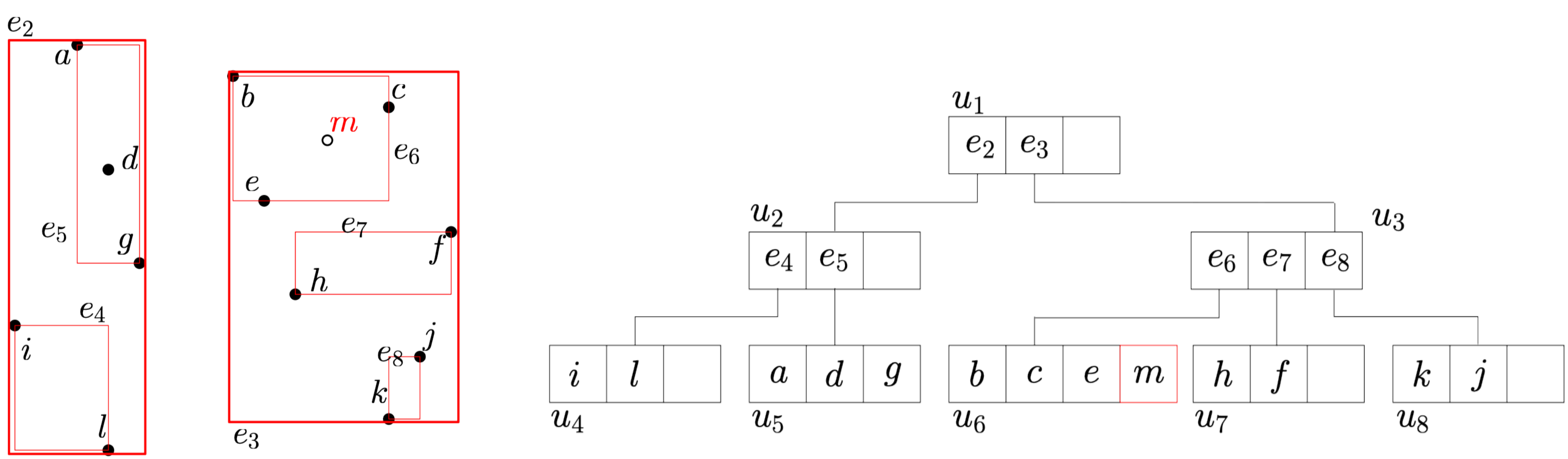
- $split(u)$

1. m: number of points in u
2. Sort the points of u on x-dimension
3. For $i = \lceil 0.4B \rceil$ to $m - \lceil 0.4B \rceil$
4. $S_1 \leftarrow$ the set of the first i points in the list
5. $S_2 \leftarrow$ the set of the other i points in the list
6. Calculate the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$
7. Repeat 2-6 with respect to y-dimension
8. Return the best split found



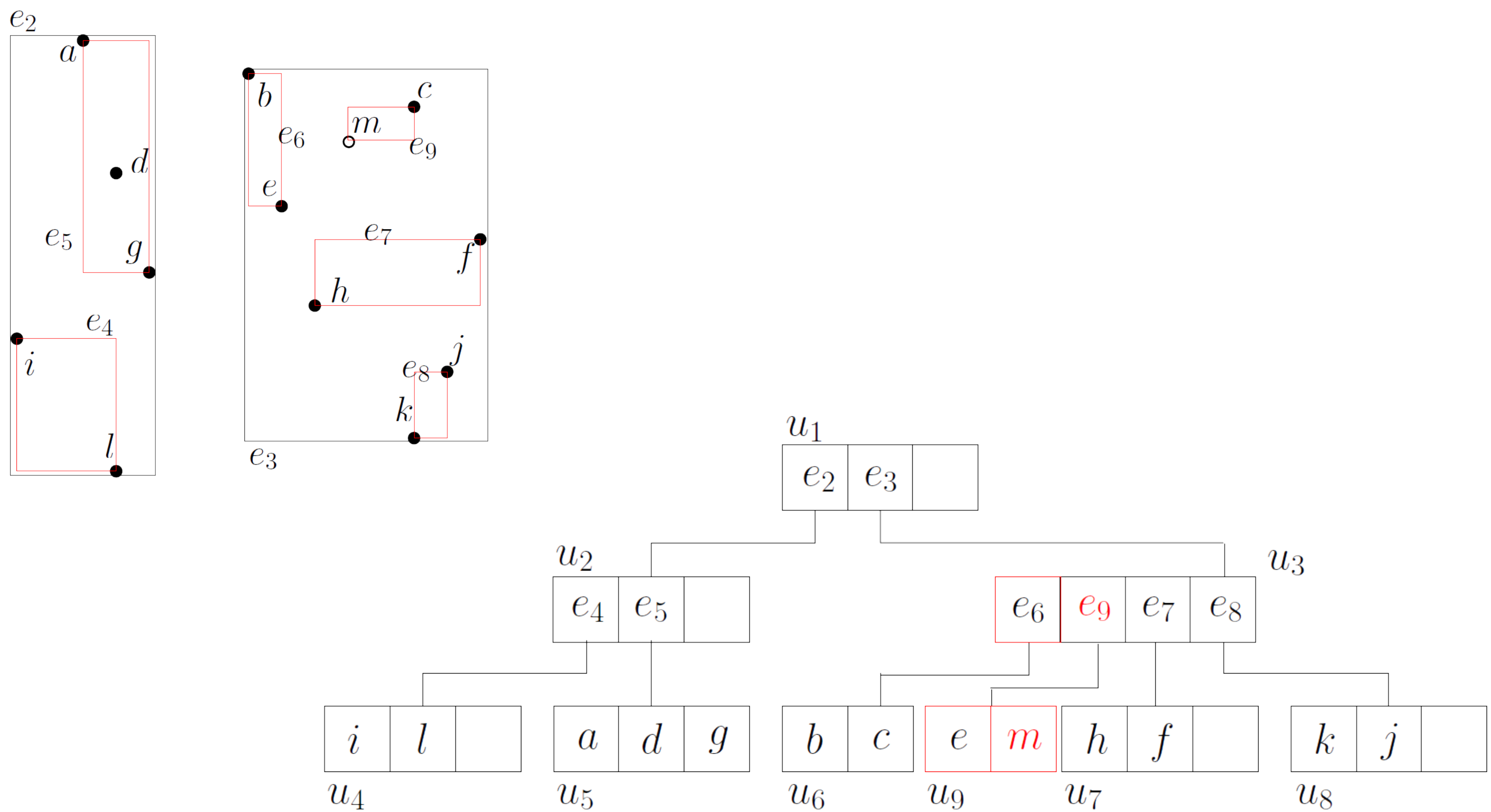
+ R-Tree Insertion Example

- Let S be a set of $B+1$ rectangles
  - Divide S into two disjoint sets S_1 and S_2 to minimize the perimeter sum of $MBR(S_1)$ and $MBR(S_2)$
  - $|S_1| \geq 0.4B$, $|S_2| \geq 0.4B$
  - Node u_6 splits, generating u_9



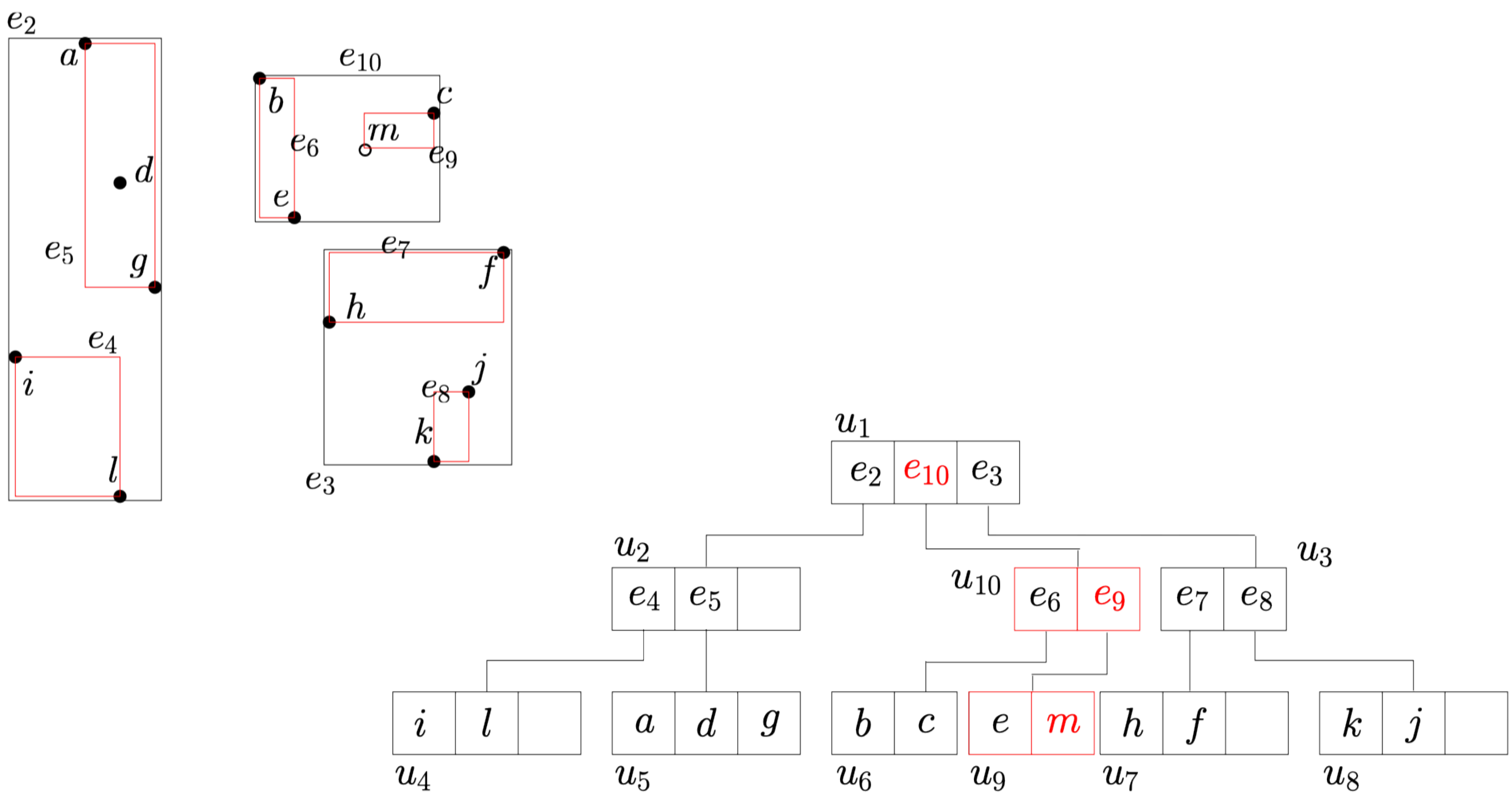
+ R-Tree Insertion Example

- Adding u_9 as a child of u_3 causes u_3 to overflow



+ R-Tree Insertion Example

- Node u_3 splits, generating u_{10} as a child of the root

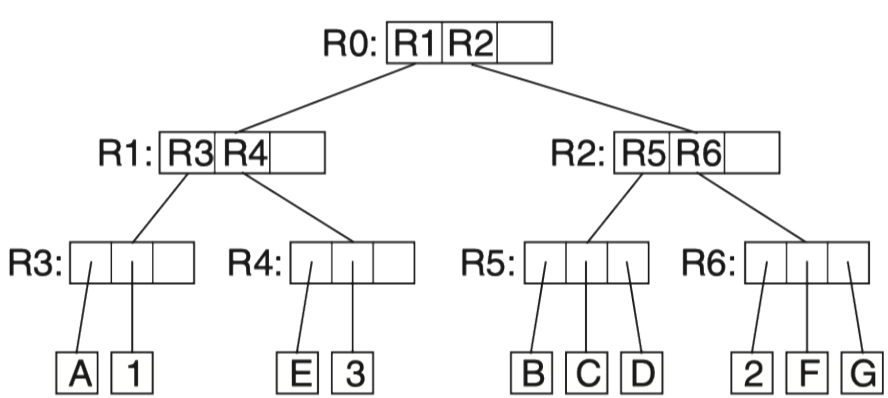


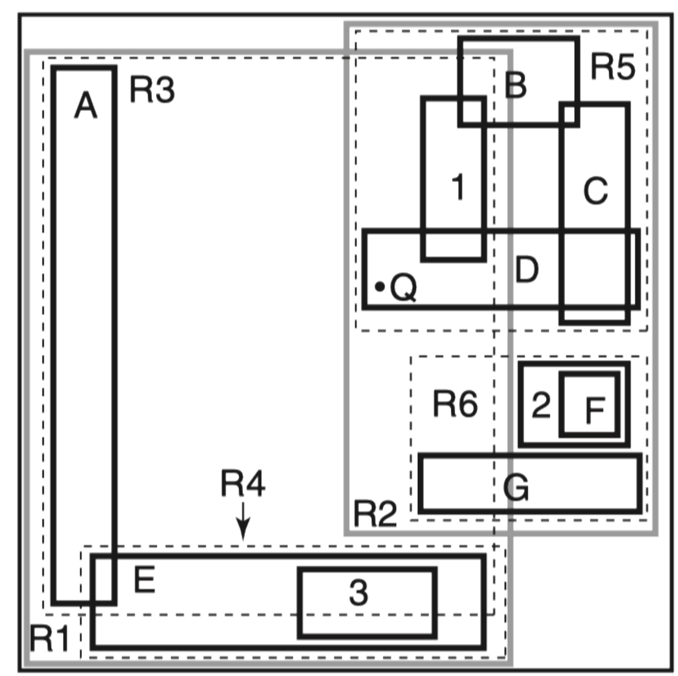
+ Query Processing Using R-Trees

- A node records the MBR of all objects in the subtree rooted from the node
  - Point query
  - Window query
  - Spatial join query
  - Nearest Neighbor query
  - Skyline query
  - …

+ Clipping

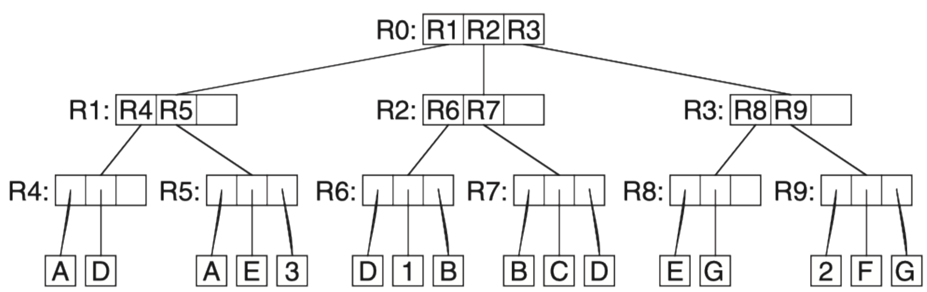
- Motivation
  - R-Tree: May examine all the MBRs at all levels
    - Because the MBR may overlap, the space is not disjointly decomposed
      - Query point Q in the example
  - Single search path for a point query

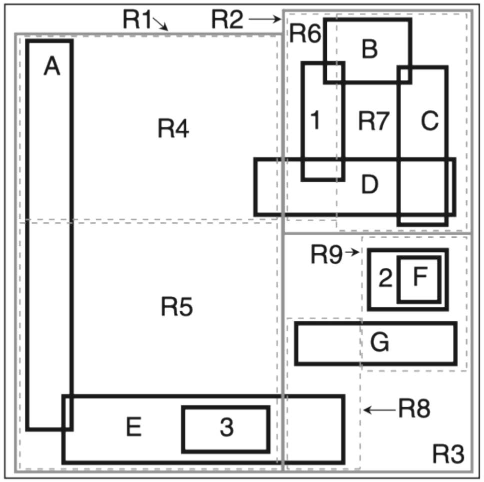




+ R^+-Tree

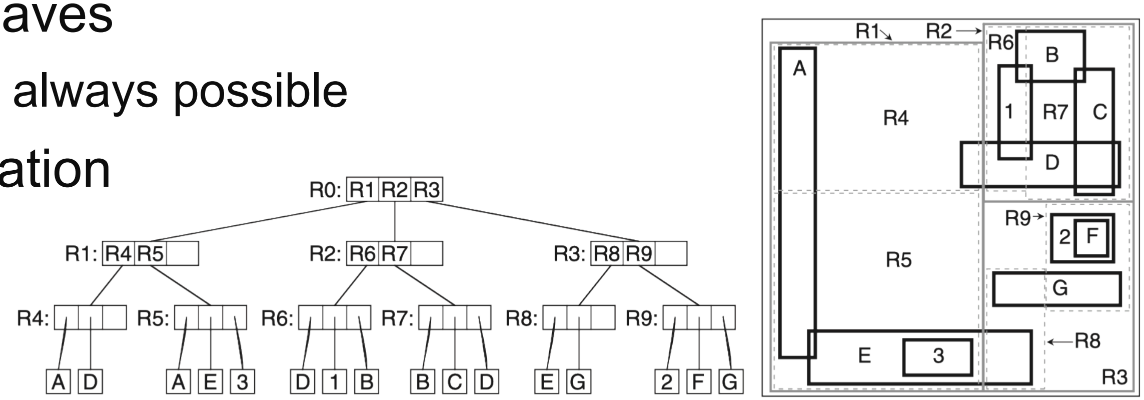
- Basic ideas
  - A hierarchy of overlapping MBRs → A hierarchy of disjoint MBRs
    - Regular grid / Irregular grid
  - Clipping polygon at cell boundaries
    - Whenever an MBR at a lower level overlaps with another MBR, decompose it into a collection of non-overlapping sub-MBRs
  - Allowing one polygon in multiple cells
    - Non-overlapping is achieved at the cost of space





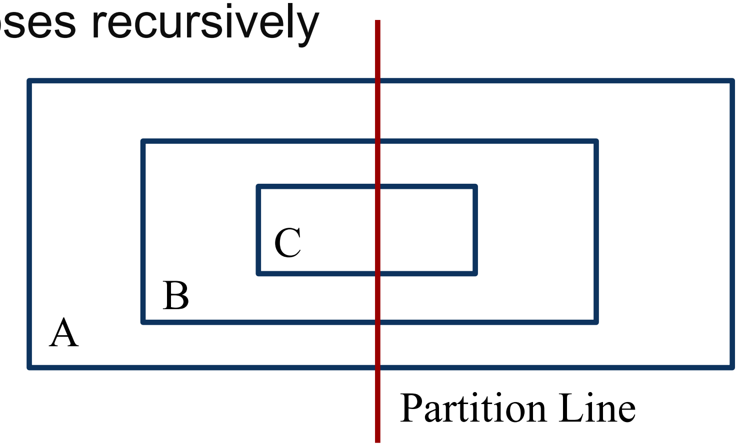
+ R⁺-Tree

- Insertion
  - Insert an object's MBR into all of the leaf nodes that overlap it
    - Find all the intersected nodes and clipped the object's MBR
  - Overflow
    - Propagate to the parents, like R-Tree
    - Propagate to the children
      - A split of the parent node may introduce a space partition that affects the children nodes
- Deletion
  - Remove from all the leaves
    - Lead to Merge, but not always possible
  - Periodically re-organization



+ R⁺-Tree

- Split
  - When A is split, its child B has to be split, and B's child C has to be split…
  - Split the children might further require the split the parent
    - Split parent may require split the children (repartitioning creates overflow)
    - Up and down and up and down…
    - Deadlock!
      - Upper-bound is M MBRs
      - A node contains M+1 MBRs that encloses recursively



+ R$^+$-Tree

- Problems
  - Multiple index entries for an object
    - Increased search time (return the same object more than once for window query)
    - Overflow more likely
  - Cascading splitting
  - Deadlock

+ Query Processing Using R^+-Trees

- Point query
- Window query
- Within buffer / distance
- Spatial join query

+ Data Access Methods

- One dimensional
  - Hashing and B-Trees
- Line Data
  - Interval Tree, Segment Tree
- Point data
  - Hashing: GRID and EXCELL
  - Hierarchical
    - Quadtree: point and region quadtrees
    - kd-Tree
    - Z-values and B-tree
- Polygon data
  - Transformation: End point mapping and Z-values
  - Overlapping: R-tree and R*-tree
  - Clipping: R^+-tree

+ Indexing High Dimensional Data

- GIS applications in 2- or 3-D only
- Multimedia DB can have data with several hundred dimensions.
- While point/polygon access methods can be generalized for higher-D applications, they may be not efficient
- High-D indexing is a hard problem

+ Advanced Techniques for High Dimensional Data

- Course Introduction
- Introduction to Spatial Databases
- Spatial Data Organization
- Spatial Query Processing
- Managing Spatiotemporal Data
- Managing High-dimensional Data
- Other High-dimensional Data Applications
- When Spatial Temporal Data Meets AI
- Route Planning
- Trends and Course Review