Indexing

- Extract specific information from data and access data through it
- attributes, attribute vectors
- Two step retrieval:
  1) *hypothesis*: search through the index returns all qualifying documents plus some false alarms
  2) *verification*: the answer is examined to eliminate false alarms

Database Indexing Methods

- Indexing based on
  - *primary key*: single attribute, no duplicates
  - *secondary keys*: one or more attributes
    - duplicates are allowed
    - indexing in *M*-dimensional feature spaces
- Data and queries are vectors
  - retrieval: two step search approach

Primary Key Indexing

- Dynamic indexing: the file grows or shrinks to adapt to the volume of data
  - good space utilization and good performance
- Methods:
  - *B*-trees and variants (*B*+-trees, *B**-trees)
  - Hashing and variants (linear hashing, spiral etc.)
  - hashing is faster, *B*-trees preserve order of keys
- *B*-trees, hashing are the industry work-horses

Secondary Key Indexing

- Much interest in multimedia
  - signals are represented by feature vectors
  - *feature extraction* computes feature vectors from signals
- The index organizes the feature space so that it can answer queries on any attribute

Query Types

- **Exact match**: all attribute values are specified
  - name = "smith" and salary = 30,000
- **Partial match**: not all attribute values are specified
  - name="smith" and salary = *
- **Range queries**: range of attribute values are specified
  - name="smith" and (20,000 <= salary <= 30,000)
  - find images within distance T
- **Nearest Neighbor (*NN*)**: find the *K* best matches
  - find the 10 most similar images
- **Spatial join queries**: find pairs of attributes satisfying a common constraint
  - find cities within 10km from a lake

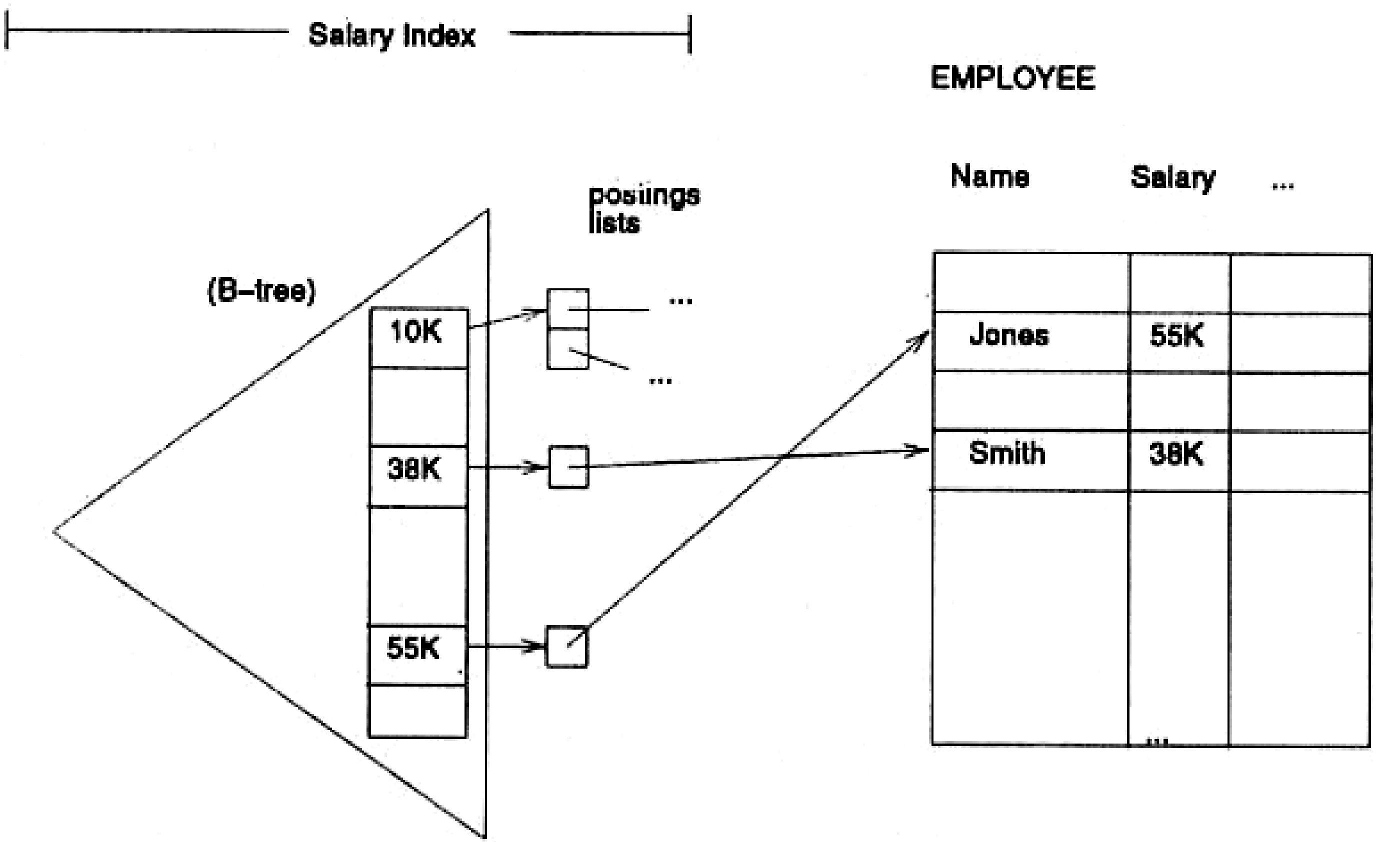
Index Structures

- Inverted files: each attribute points to a list of documents
- Point Access Methods (PAMs): data are points in an *M*-dimensional space
  - Grid file, *k-d*-tree, *k-d-B*-tree, *hB*-tree, ...
- Spatial Access Methods (SAMs): data are lines, rectangles, other geometric objects in high dimensional spaces
  - *R*-trees and variants, space filling curves

Inverted Files

- Maintain a posting list per attribute
- A posting list points to records that have the same value
- A directory for each distinct attribute value
  - sorted
  - organized as a *B*-tree *or* as a hash table
- Boolean queries are resolved by merging posting lists

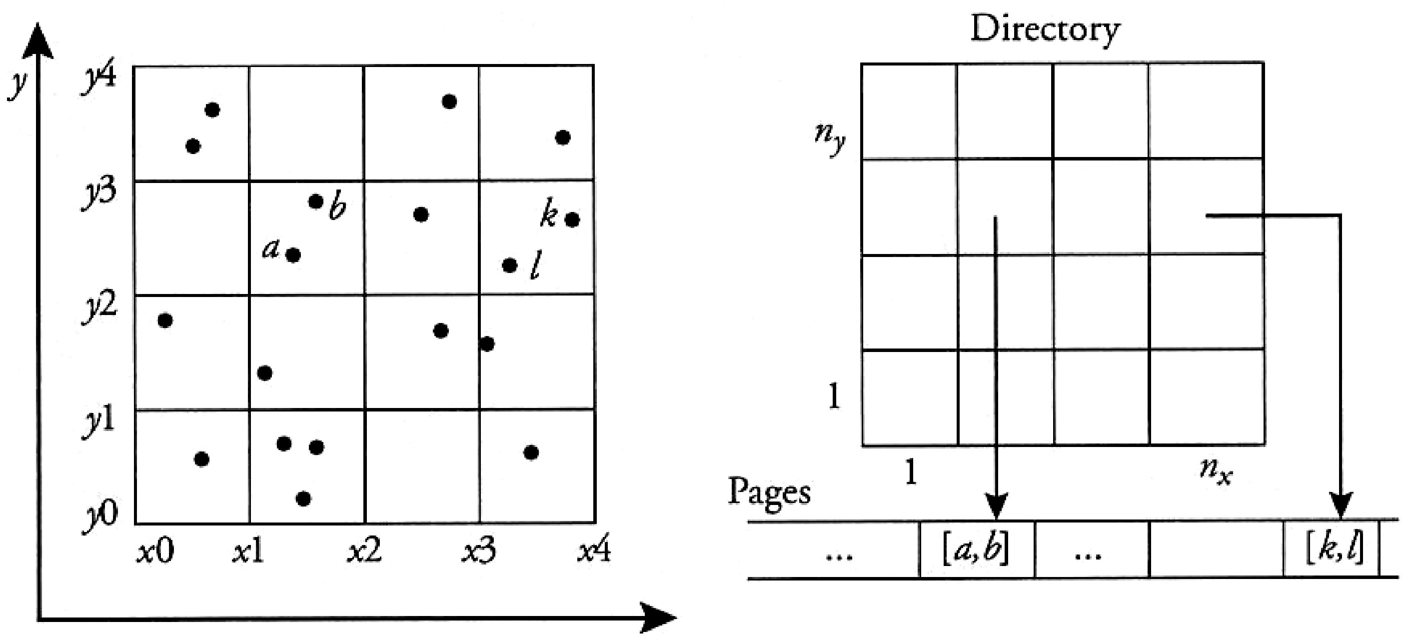
Inverted file with B-tree



Grid File

- Imposes a grid on the address space
  - the grid adapts to the data density by introducing more divisions on areas with high data density
  - grid cells correspond to disk pages
  - two or more cells may share a page
  - the cuts are allowed on predefined points (e.g., 1/2, 1/4, 3/4) on each axis
  - *M*-dim. directory for *M*-dim. data
  - directory: one entry for each cell and a pointer to a disk page

2D Grid on 2D Space



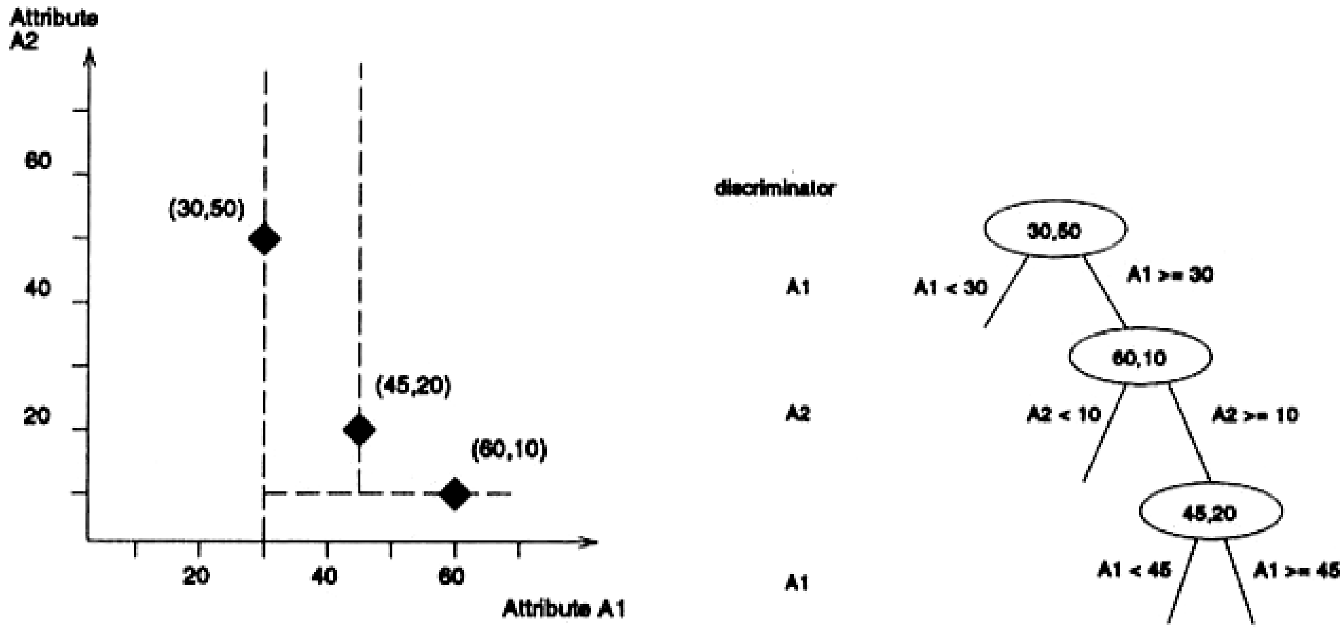
Comment on Grid File

- *Pros*:
  - two disk accesses for exact match
  - symmetric with respect to the attributes
  - adapts to non-uniform data distributions
  - good for low dimensionality spaces
- *Cons*:
  - not good for correlated attributes
  - large directory for many dimensions

k-d-trees

- Divides the address space into disjoint regions through cuts on alternating dimensions (attributes)
  - binary tree
  - A different attribute as discriminator at each level
  - the left sub-tree contains records with smaller values of that attribute
  - the right sub-tree keeps records with greater values

k-d-tree with 3 Records, 2 Attributes



(a) (b)

a) the divisions of the address space
b) the tree

Comments on k-d-tree

- *Pros*:
  - elegant and intuitive algorithms
  - good performance thanks to the efficient pruning of the search space
  - Good for exact, range and nearest-neighbor queries
- *Cons*:
  - main memory access method

Extensions of k-d-trees

- *k-d-B*-trees [Robinson 81]:
  - divides the address space into *m* intervals for every node (not just 2 as the *k-d*-tree)
  - Always balanced, disk access method
- *hB*-tree [Lomet & Salzberg 90]:
  - divides the address space into regions
  - the regions may have holes
  - nodes (disk pages) are organized as *B*-trees
  - disk access method

Spatial Access Methods (SAMs)

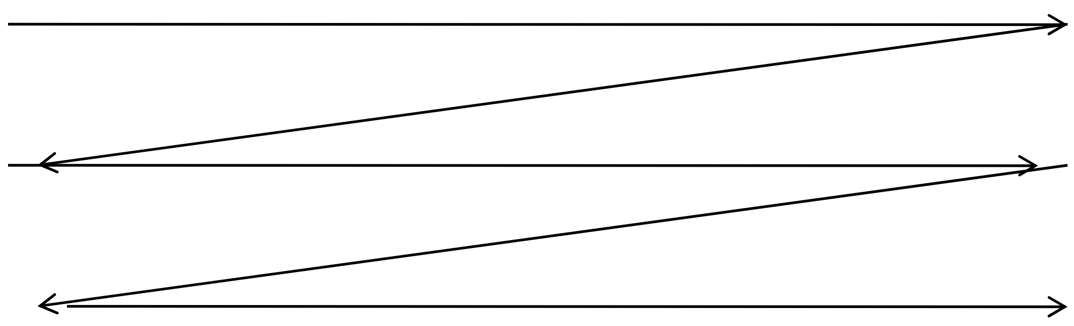
- File structures that handle points, lines, rectangles, general geometric objects in high dimensional spaces
- Two classes of SAMs:
  - *space filling curves*: *Z*, Gray, Hilbert curves
  - *tree structures*: *R*-trees and its variants
- Common query types:
  - point queries: find the nearest rectangles containing it
  - window queries: find intersecting rectangles

Space Filling Curves

- Mapping of multi-dimensional space to one dimension
  - visit all data points in space in some order
  - this order defines an *1D* sequence of points
  - points which are close together in the multi-dimensional space must be assigned similar values in the 1D sequence
  - A B^+-tree for indexing

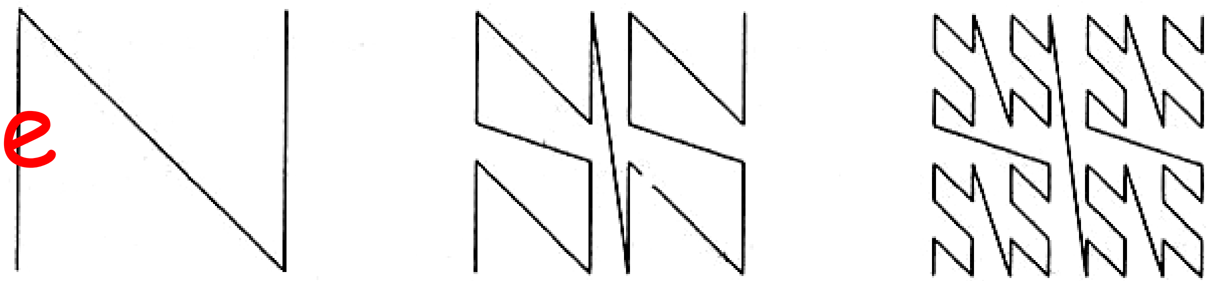
Space Traversal

- Visit the pixels in row-wise order
  - Tends to create large gaps between neighboring points
- Better ideas: Z-curves, Hilbert curves

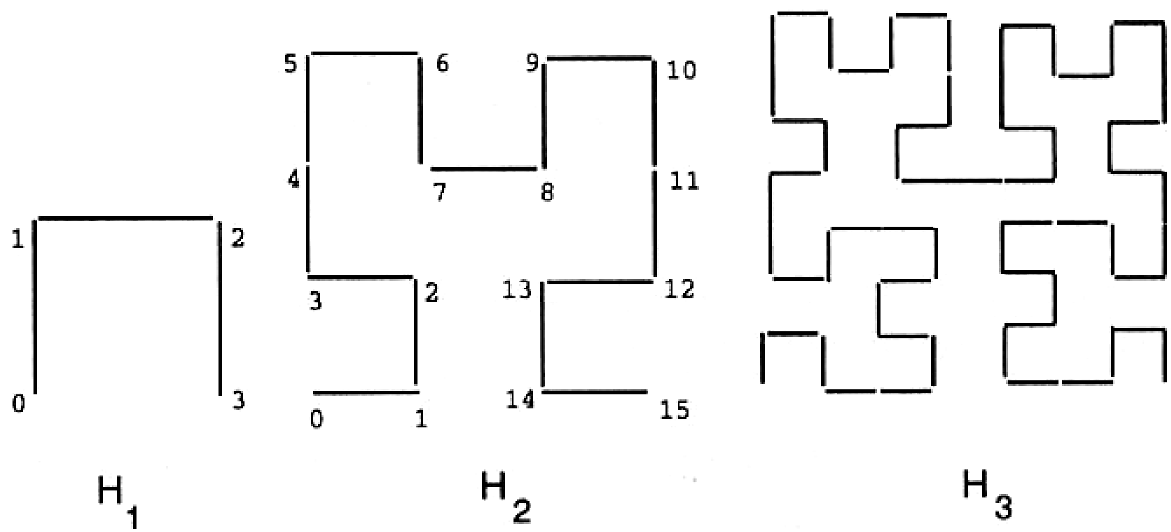


Two Common Curves

Z-curve



Hilbert

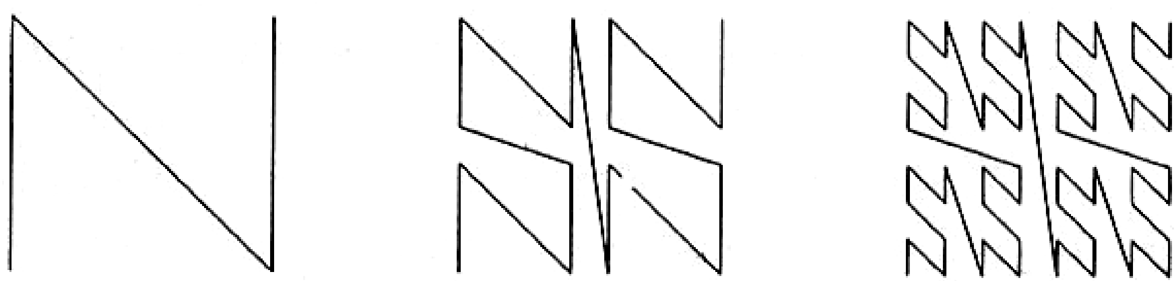


Creating Indices

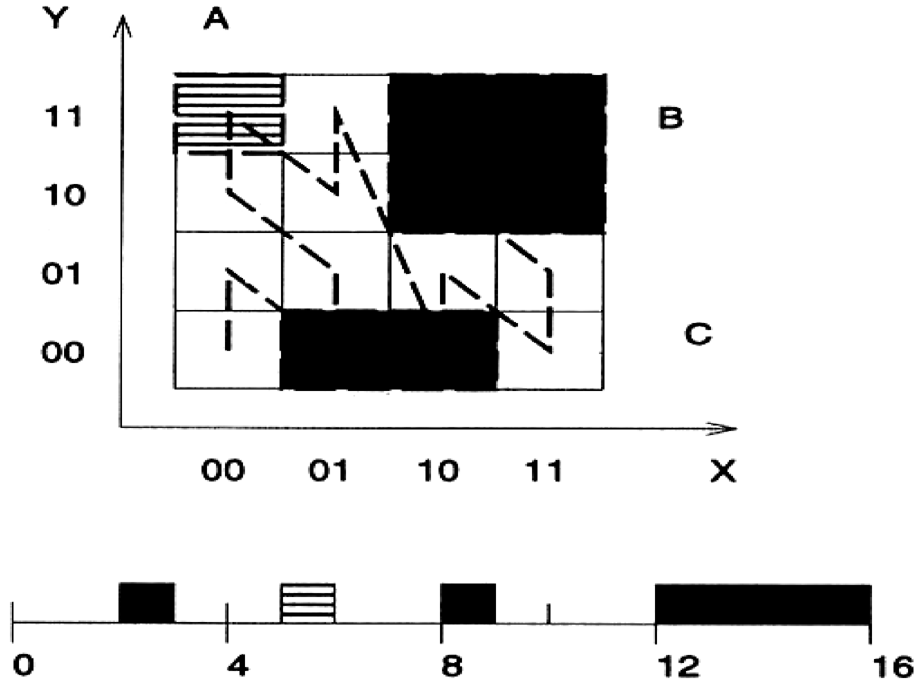
- Bit interleaving:
  - Assign *k*-bits per axis (2^k values)
  - Take the *x*, *y*, … coordinates of each pixel in binary form
  - Shuffle bits in some order
  - Each pixel takes the value of the resulting binary number
- The order with which the pixels are taken produces a mapping to 1D space

Z-Order

- Shuffle bits from each of the *M* dimensions in a round-robin fashion
  - *2D space*: "*12*" take bit from x coordinate first, then bit from y coordinate
- Visiting all pixels in ascending *Z*-value order creates a self-similar trail of *N* shapes
  - the trail can be defined on different size grids



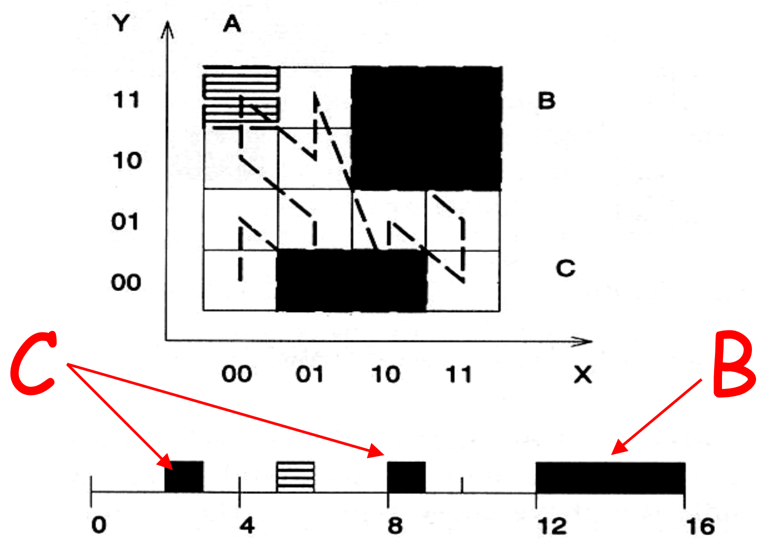
k=2 bits
per pixel



Pixel A=(0,3)=(00,11): shuffle(1212,00,11)=0101=5

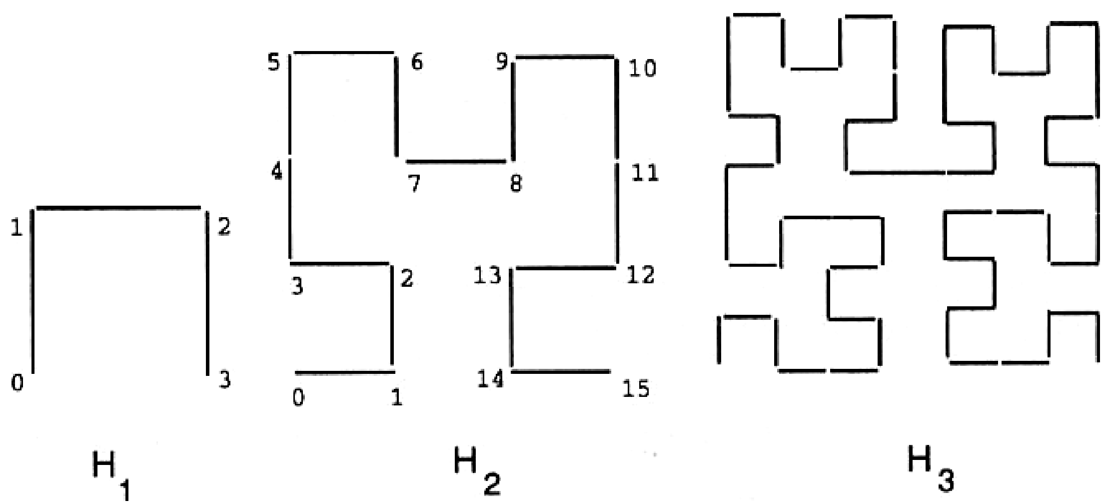
Z Regions

- A region breaks into one or more pieces each of which is described by a *Z*-value
  - region **C** breaks into 2 pixels: with *Z* values 0010=2 and 1000=8
  - region **B** consists of 4 pixels with common prefix 11 which is taken to be *Z*-value of the **C** region



Hilbert Curve

- Better clustering than *Z*-ordering
  - Less abrupt jumps
  - better distance preserving properties

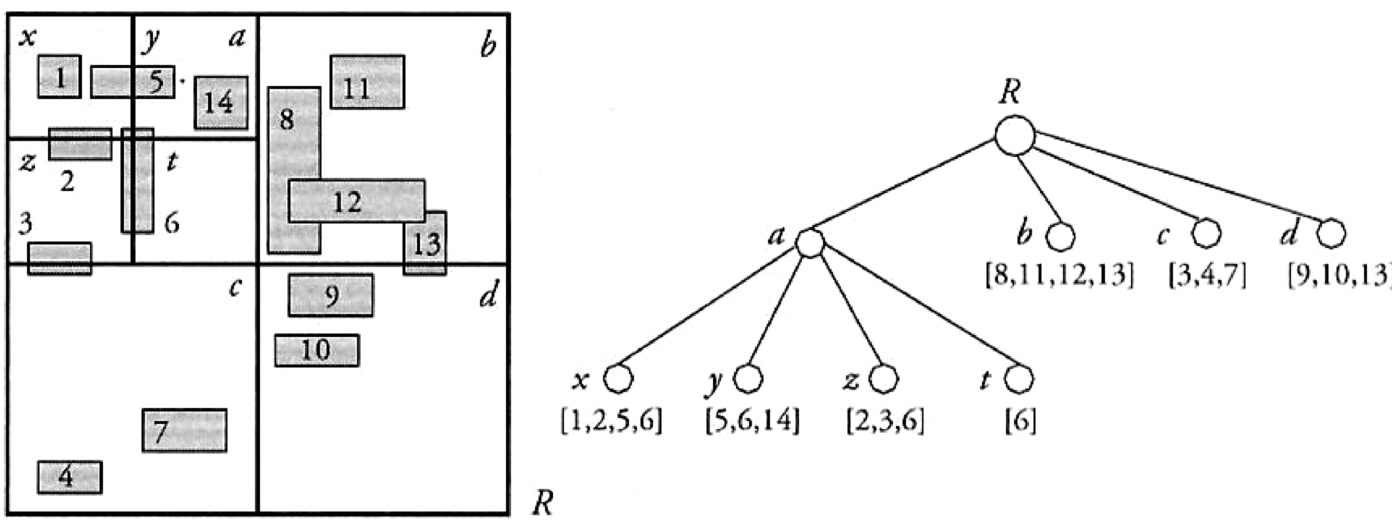


Tree SAMs

- Quadtree: space driven access method
  - good for main memory
  - Linear Quadtree: combines *Z*-ordering with quadtrees, good for main memory and disk
- *R*-tree: data driven access method
  - good for main memory and disk
  - *R*+-tree, *R**-tree, *SS*-tree, *SR*-tree etc.

Quadtree

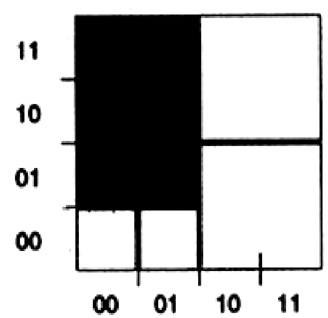
- Recursive decomposition of space into quadrants
  - decompose until a criterion is satisfied
  - the index is a quaternary tree
  - each node contains the rectangles it overlaps



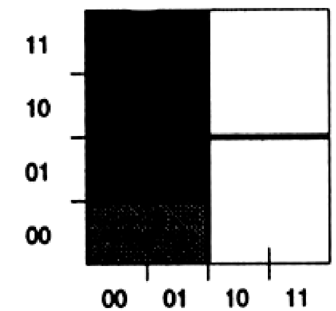
Linear Quadtree

NW 1 0	NE 1 1
SW 0 0	SE 0 1

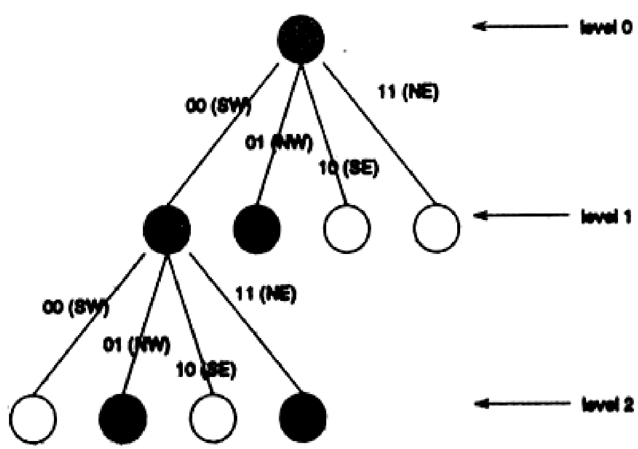
- Good for disk storage
  - nodes: *NW, NE, SW, SE*
  - *0*: *S or W*,
  - *1*: *N or E*
  - each edge has a 2-bit label (e.g., *NW*: 10)
  - *Z*-value of a node: concatenate *Z*-values from root (e.g., shaded rectangle: 0001)
  - *Z*-values are inserted into a *B+*-tree

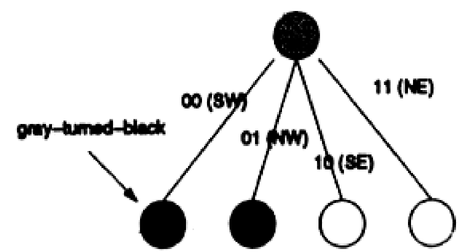


(a) a spatial object



(d) the corresponding approximate object



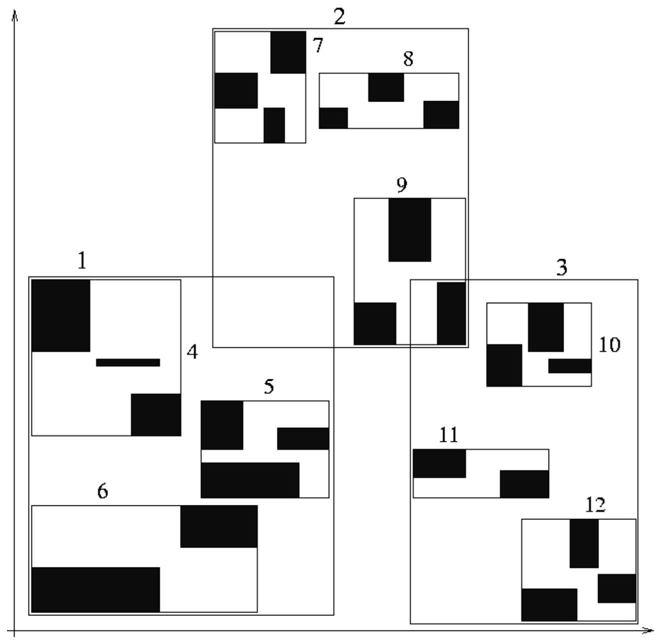


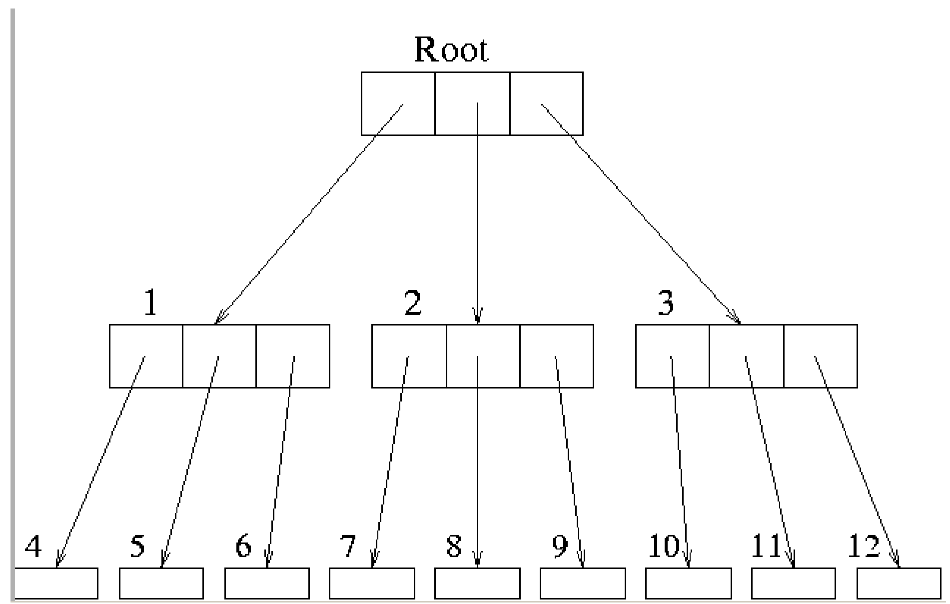
shaded rectangle: 3 blocks

approximation contains the shaded region

R-tree [Guttman 84]

- The most successful SAM
- Balanced, as a *B+* tree for many dimensions
- Objects are approximated by MBRs
- Non-leaf nodes contain entries *(ptr, R)*
  - *ptr*: pointer to children node
  - *R*: *MBR* that covers all rectangles in child node
- leaf-nodes contain entries *(obj-id, R)*
  - *obj-id*: pointer to object
  - *R*: *MBR* that covers all objects in child node
- parent nodes are allowed to overlap





rectangles organized as an R-tree (fanout: 3)

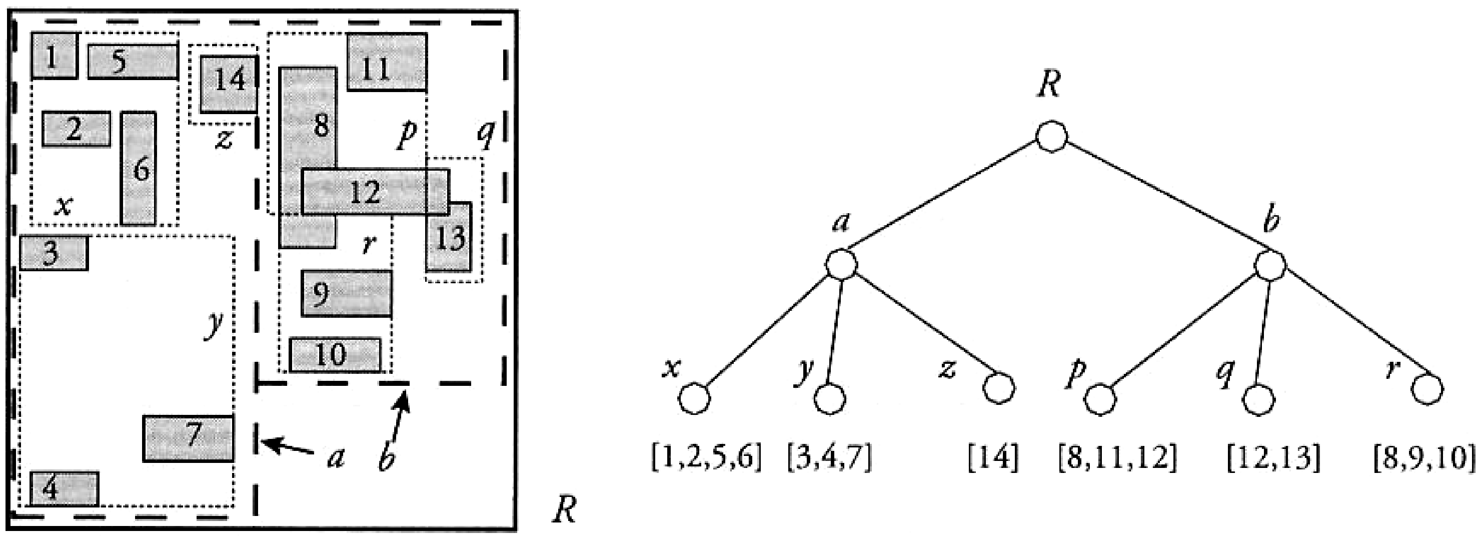
R-tree leaf nodes correspond to disk pages

Algorithms for R-trees

- Nodes overlap leads to searching along multiple paths and recursive algorithms
- *Insertion*: traverse tree, put in suitable node
  - split if necessary
  - *R**-tree*: differ splitting
  - changes propagate upwards
  - *R*-tree is always balanced
- *Range queries*: traverse tree, compare query with node *MBR*, prune non-intersecting nodes
- *NN queries*: more complex, branch and bound technique [Roussopoulos 95]

R-trees and Variants

- *R**-tree: differ splits to achieve better utilization in a better structured *R*-tree
  - when a node overflows, some of its children are deleted and reinserted
  - outperforms *R*-tree by 30% (?)
- *R+*-tree: nodes are not allowed to overlap
  - no good space utilization, larger trees, rectangles can be duplicated, complex algorithms
  - outperform *R*-trees for point queries: a single path is followed from the root a leaf
  - *R*-trees outperform *R+*-trees for range queries



R+-tree, objects 8,12 are referenced twice

Recent R-tree Variants

- Different space decomposition schemes
  - e.g., bounding spheres (*BS*) instead of rectangles
  - *BSs* reduce overlapping of *MBRs*
  - minimum unused space inside *BSs*
  - *BSs* divide space into short-diameter regions
  - *BSs* tend to have larger volumes than *MBRs* and contain more points
  - less flexible: only radius varies instead of length, width
  - more complex algorithms

SS-tree

- Similar to *R**-tree
  - uses *spheres* instead of rectangles
  - good performance for point queries
- *SR-tree* combines the structure of the *R**-tree and of the *SS*-tree
  - A bounding region is defined by the intersection of a *sphere* and an *MBR*
  - good for *NN*-queries

Metric Trees

- Consider only relative distances of objects rather than their absolute positions in space for indexing
  - Require that distance *d* is a *metric*
    - *d(a,b) = (b,a)*
    - *d(a,b) >= 0* for *a < > 0* and *d(a,b) = for a = b*
    - *d(a,c) <= d(a,b) + d(b,c)*
  - *triangle inequality* for pruning the search space

VP-Tree [Yanilos 93]

- Divides the space using a distance from a selected vantage point
  - *root*: entire space (all database objects)
  - *left subtree*: points with the less distance
  - *right subtree*: points with greater distance
  - recursive processing at each node
  - a binary tree is formed
  - logarithmic search time
  - static, good for main memory
  - *m-vp-tree:* multiple vantage points

M-Tree [Ciaccia 97]

- Combines SAMs and metric trees
- Balanced tree, good for disk
- Routing objects: internal nodes
- Leaf nodes: actual objects
- Routing objects point to covering sub-trees
- Objects in a covering sub-tree are within distance *r* from the routing object
- A routing object is associated with a distance *p* from its parent object

Performance

- Dimensionality curse: as dimensionality grows the performance drops
  - even worst than sequential scanning
- *R*-trees and variants: up to 20-30 dims for point objects, 20 dims for rectangles
  - more dimensions, larger space for MBRs, fanout decreases, taller and slower tree
- Fractals: good performance for 2-3 dims
- *M*-trees: good performance for up to 10 dims

References

- John Louis Bentley, Jerome H. Friedman, "Data Structures for Range Searching". Computing Surveys, Vol. 11, No 4, December 1979
- Antonin Guttman, "R-trees: A Dynamic index Structure for Spatial Searching". Proceedings ACM SIGMOD International Conference on the Management of Data, 1984.
- Timos Sellis, Nick Roussopoulos and Christos Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects". Proceedings of the 13th VLDB Conference, Brighton 1987.
- Norbert Beckmann, Hans-Peter Kriegel, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". Proceedings ACM SIGMOD International Conference on the Management of Data, Atlantic City, NJ, May 1990.
- David Lomet, "A Review of Recent Work on Multi-attribute Access Methods". SIGMOD RECORD, Vol. 21, No 3,September 1992.
- Peter N. Yianilos, "Data Structures and Algorithms for the Nearest Neighbor Search in General Metric Spaces". Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Austin-Texas, Jan. 1993.

References

- Paolo Ciaccia, Marco Patella, Pavel Zezula, "M-tree: An Efficient Method for Similarity Search in Metric Space". Proceedings of the 23rd VLDB Conference, Athens Greece, 1997.
- Volker Gaede, Oliver Gunther, "Multidimensional Access Methods". ACM Computing Surveys, Vol.30, No 2, June 1998.
- Joseph M. Hellerstein, Avi Pfeffer, "The RD-tree: An Index Structure for sets". University of Wisconsin, Computer Science Technical report 1252, November 1994.
- N. Katayama and S. Satoh. The SR-tree: "An Index Structure for High-Dimensional Nearest Neighbor Queries, In *Proc. of ACM SIGMOD*, pages 369–380, 1997.
- C. Faloutsos and S. Roseman: "Fractals for Secondary Key Retrieval", In Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 247-252, Philadelphia, Pennsylvania, March 29-31, 1989