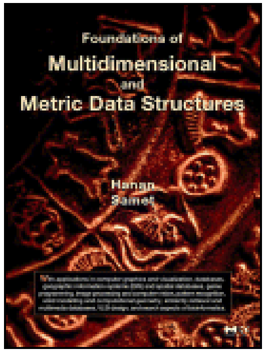
Multimedia Databases
Access Structures

Prof (FH) PD Dr. Mario Döller

Literature

- Books
  - Hanan Samet; *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers, 2006, ISBN: 978-0-12-369446-1.



  - K. Selcuk Candan, Maria Luisa Sapino; Data Management for Multimedia Retrieval, Campridge University Press, 2010, ISBN: 978-0-521-88739-7

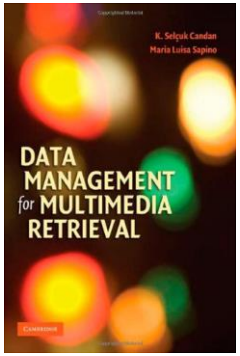


Table of Contents

Access Structures

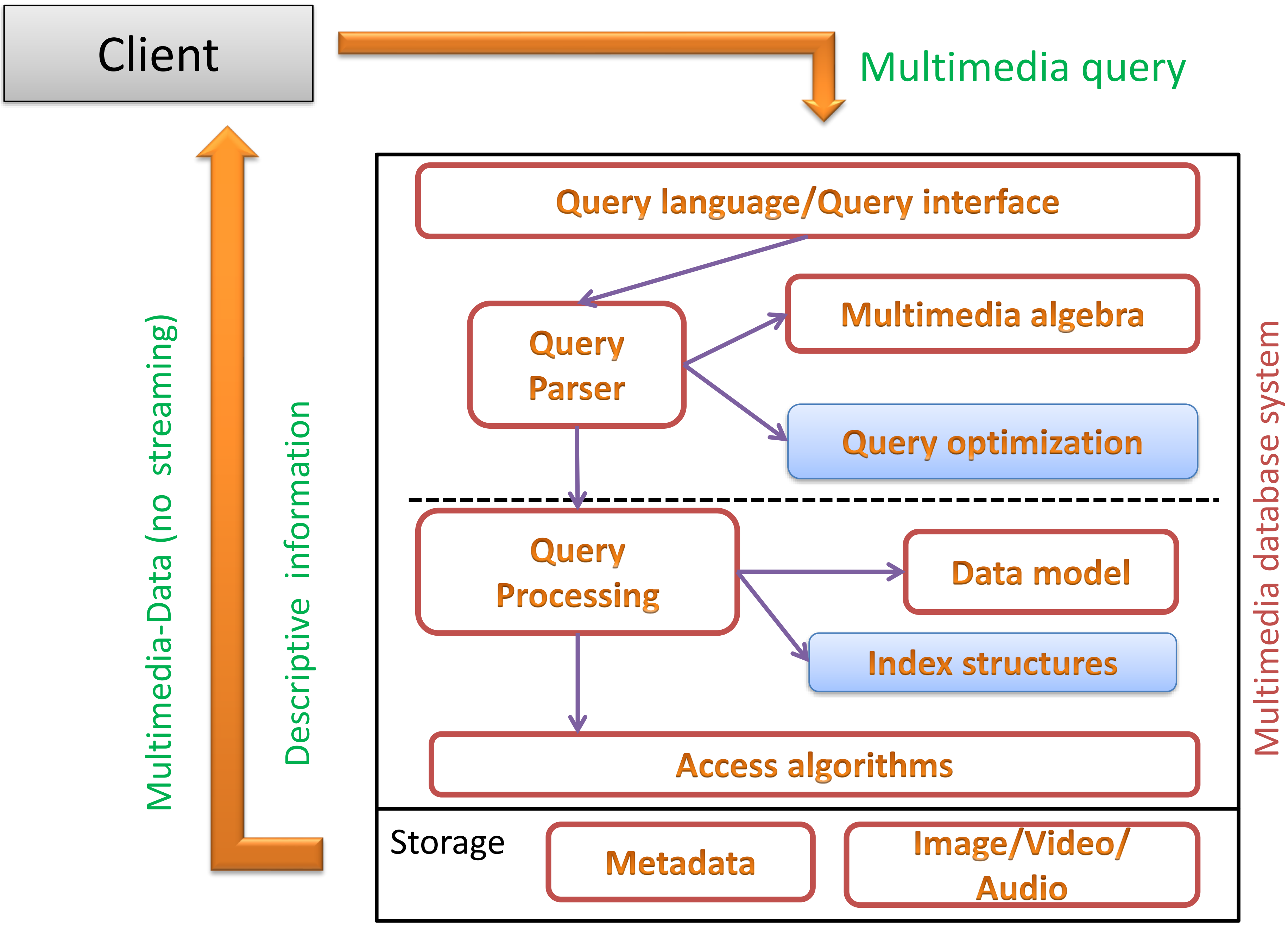

Client
Multimedia query
Multimedia-Data (no streaming)
Descriptive information
Query language/Query interface
Query Parser
Multimedia algebra
Query optimization
Query Processing
Data model
Index structures
Access algorithms
Multimedia database system
Storage
Metadata
Image/Video/ Audio

Table of Contents

Access Structures

Introduction

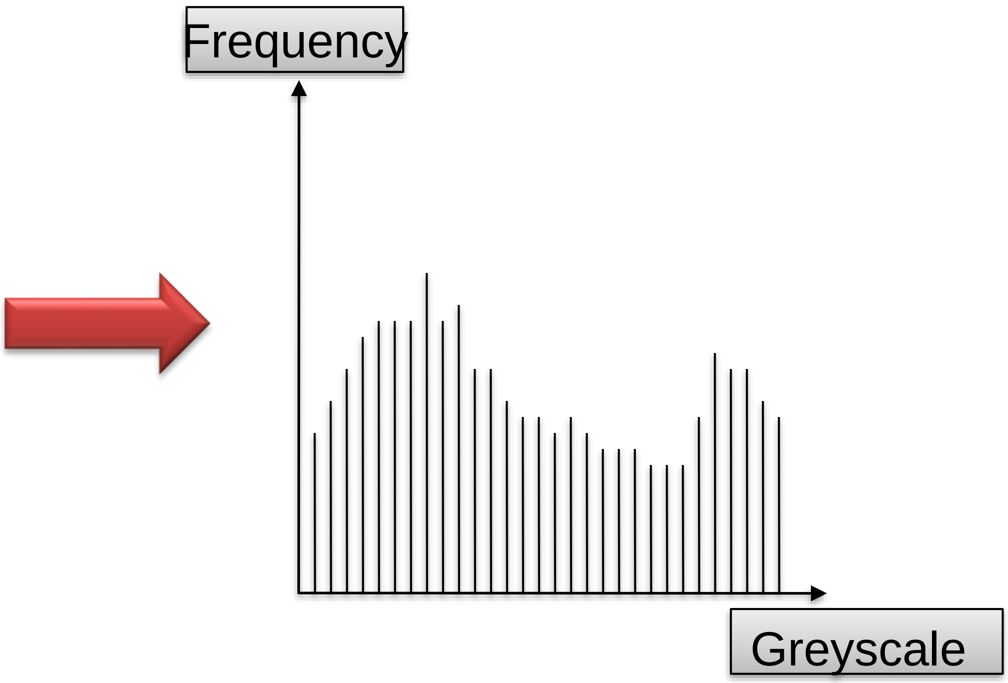
- Content-based retrieval in Multimedia database systems bases on the use of signature vectors.

- Low-level features of an image
(ex.: color distributions) are described by signature vectors.

- Signature vectors are automatically computed from the data.

Signature Vectors

- Example of Signature vector: greyscale-histogram of an image



Computation of Index Structures

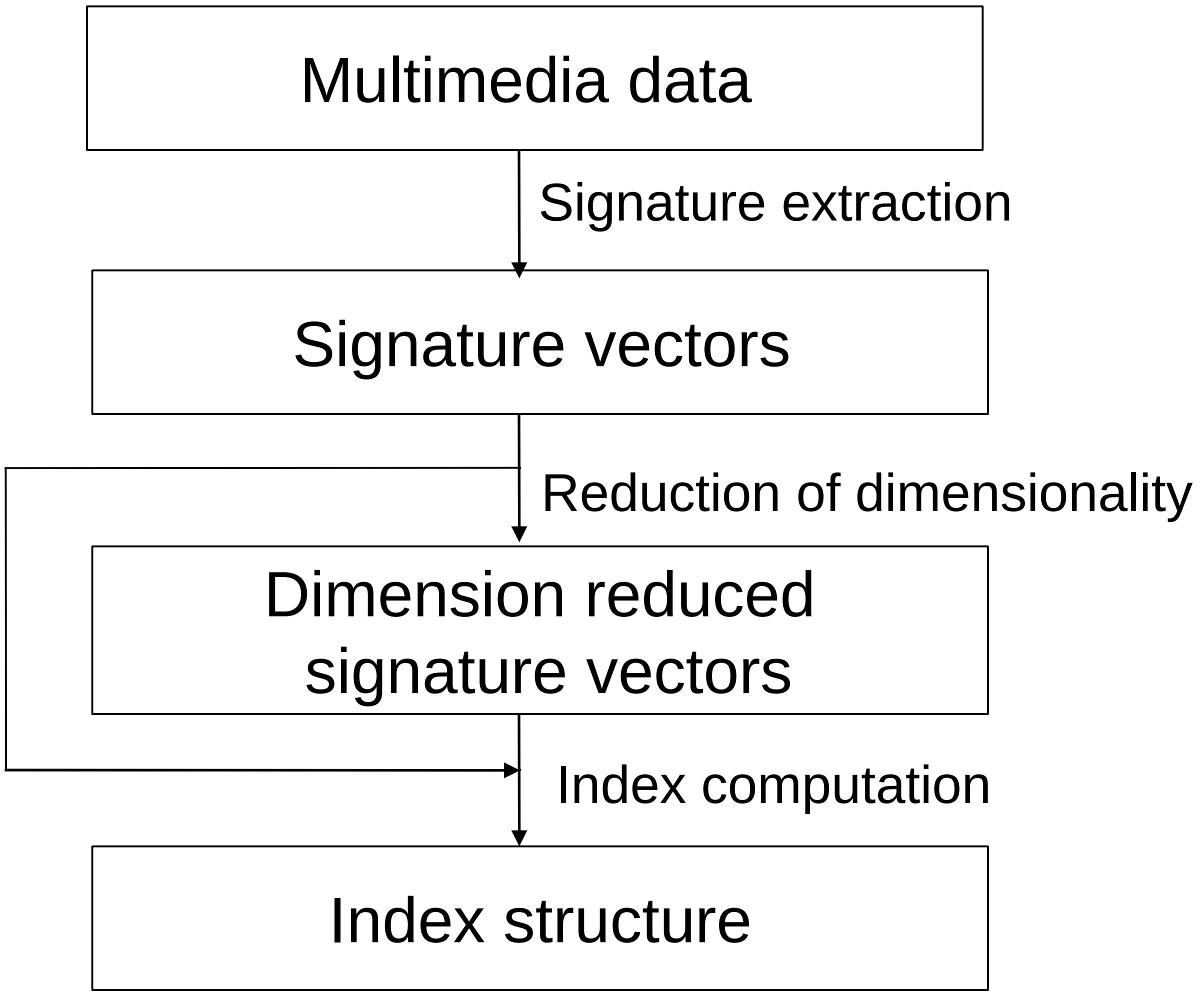


Table of Contents

Access Structures

Reduction of Dimensionality

- Goal: Signature vector with fewer dimensions but preserving the distance.
- Why?: efficiency of the index structures decreases with the dimensionality:
  - Curse of Dimensionality: sequential scan is faster than searching in the index structure.
- Means:
  - Transformations
  - Space filling curves

Reduction of Dimensionality through Transformations I

- Change the basis of the vectors
- Conversion in orthonormal vectors
- After transformation, possible to distinguish between coefficients with high and low influence
- The non important coefficients are deleted
- A signature vector with less dimensions is obtained

Reduction of Dimensionality through Transformations II

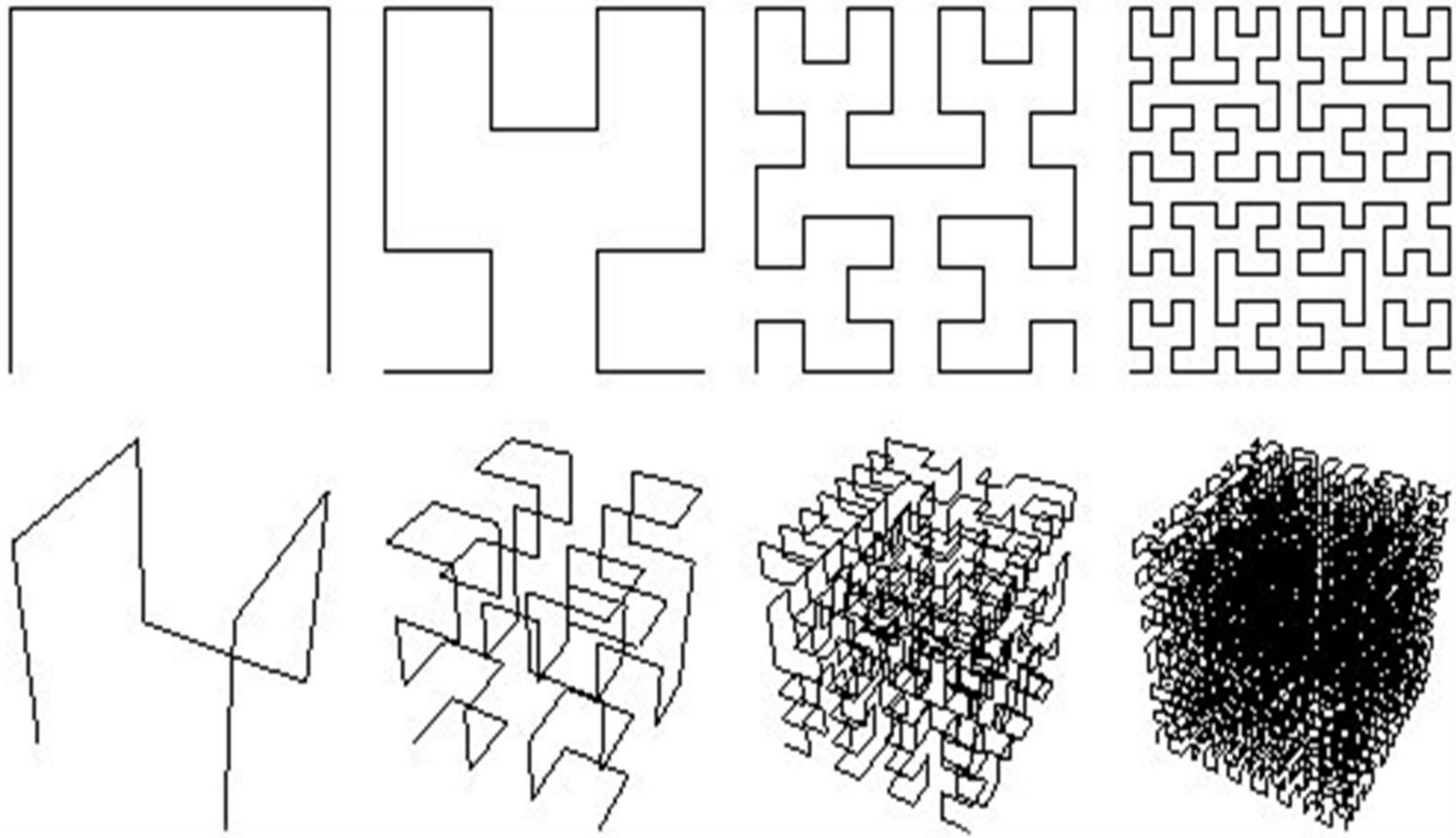
Transformation	Application
Karhunen-Loeve	clustered data
Fourier bzw. FFT	periodic data
Wavelet	discrete data
DCT	locally correlated data

Reduction of Dimensionality through Space Filling curves I

- Multidimensional space represented by a single curve.
- Create a d-dimensional point in a one-dimensional space, such that the multidimensional order is kept (as much as possible)
- Enables the use of a one-dimensional index structure.
- Examples:
  - Hilbert curve
  - Z-Ordering

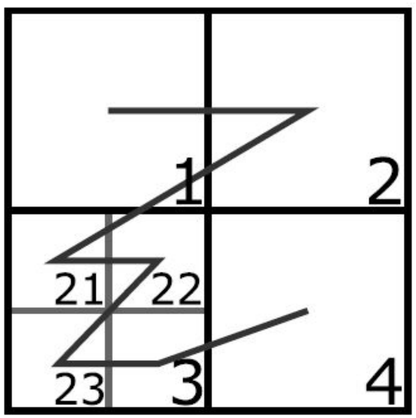
Reduction of Dimensionality through Space Filling curves II

Example: Hilbert curves

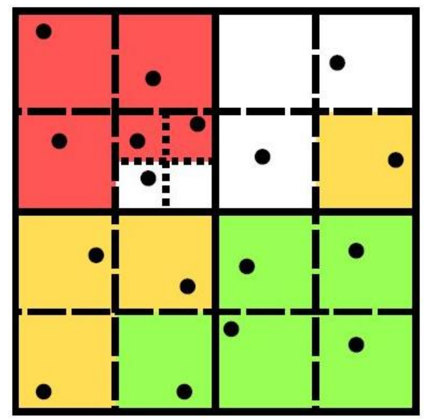


Reduction of Dimensionality through Space Filling curves III

- Example: Z-curves



Division of the space in regions



Store the four regions into the pages of a B*-tree

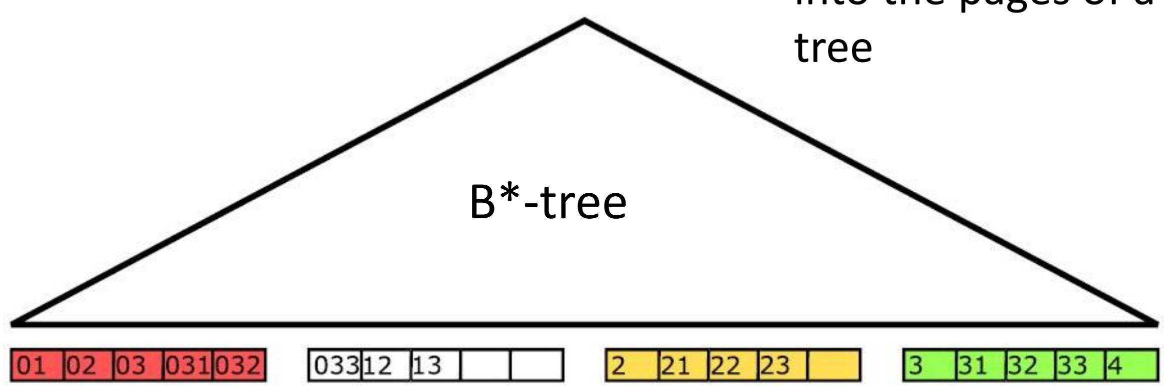


Table of Contents

Access Structures

Multi-dimensional access structures
Classification

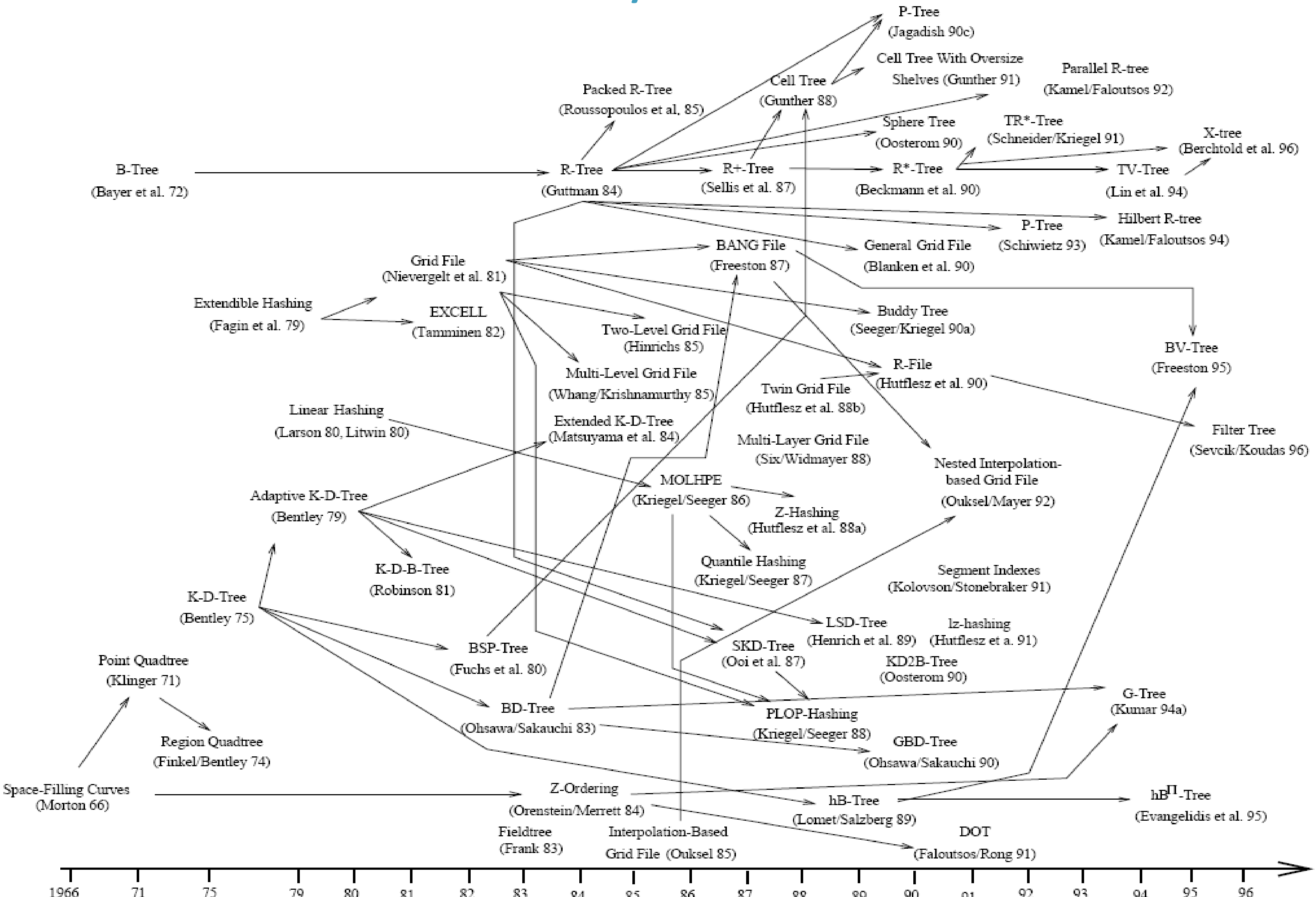
- Secondary storage algorithms:
  - R-tree (and its variants)
  - VA-file

- Main memory algorithms:
  - quadtrees
  - kd-trees
  - Locality Sensitive Hashing

Multidimensional Access Structures

- R-tree and its variants are among the most effective index trees for multidimensional index structures.
- R-tree is a generalization of the B-trees for multidimensional spaces.
- Each node is described by its MBR (Minimum Bounding Rectangle).
- A MBR contains all objects inside a node (signature vectors and sub-trees)

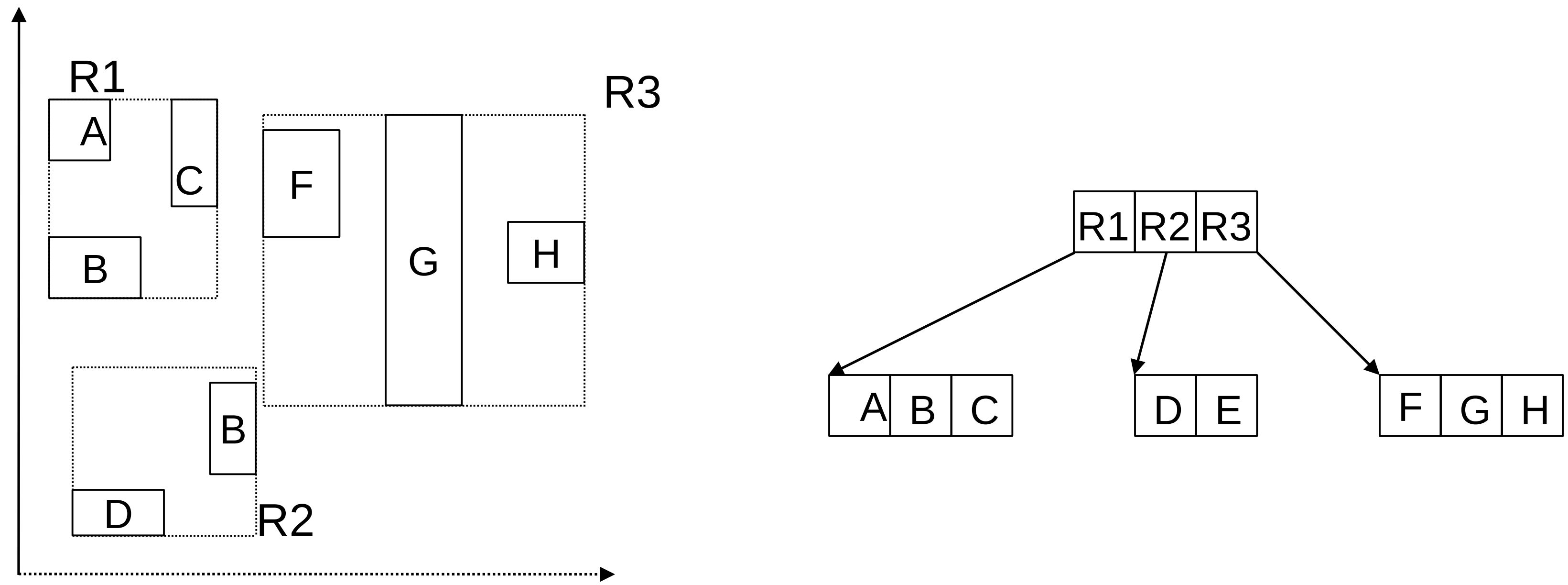
Access Structures: History



(c) Volker Gaede, Oliver Günther, Multidimensional Access Methods, ACM Computing Surveys, 30(2):170-231, 1998

Secondary Storage Algorithms

R-tree I



Secondary Storage Algorithms
R-tree II

- Overlapping MBRs reduce the efficiency of R-trees.

- Variants of the R-tree:
  - R+-tree
  - R*-tree
  - SS-tree
  - SR-tree
  - TV-tree
  - X-tree

Secondary Storage Algorithms

R+ - tree

- No overlap of MBRs allowed in R+-tree.

- An object is added to all nodes which MBR it overlaps.

- Improved search efficiency
  (compared to R-tree).

- Height of the tree is higher compared to R-tree.

Secondary Storage Algorithms

R* - tree

- Overlap of MBRs allowed.
- More efficient than R-tree.
- Reason: modified add- and split strategies.
- If a node is full: forced re-add.
- Split algorithm not called directly, instead: try to delete then re-add existing entries.

Secondary Storage Algorithms

SS-tree

- R*-tree and SS-tree very similar.

- Bounding box is a circle.

- By adding, the tree is ordered on the basis of the similarity of circular bounding boxes.

- Better performance than the R*-tree.

Secondary Storage Algorithms

SR-tree

- Enhancement of the R*-trees and of the SS-trees.
- Bounding Box is a combination of rectangles (cf. R*-tree) and circles (cf. SS-tree).
- Divide the objects in smaller, disjoint regions.
- Efficient search.
  - Details see later.

Secondary Storage Algorithms

TV-tree I

- Varying number of dimensions for indexing.
- Used dimensions depend on:
  - number of the objects to index.
  - current tree height.
- Node closer to the root: consider less dimensions.
- Actually used dimensions computed with telescope function.

Secondary Storage Algorithms
TV-tree II

- Overlaps allowed
  - Problems for nodes that use dimensions for indexing
- Next problem: first dimensions often contain the same value

Secondary Storage Algorithms

X-tree

- eXtended tree.
- Splitting is avoided as much as possible.
- Made possible by supernode -> double capacity from „normal“ node.
- Splitting using split history -> find split with minimal overlap.
- More efficient than TV-tree and R*-tree

Secondary Storage Algorithms
Multi-Feature Access Structures

- Previously introduced access structures target on one feature space (e.g. feature vector for color or texture).
- By indexing several feature vectors several access structures are needed.

- Multi-feature access structure support indexing several feature vectors in one access structure
  - M-tree and M2-tree[1,2]

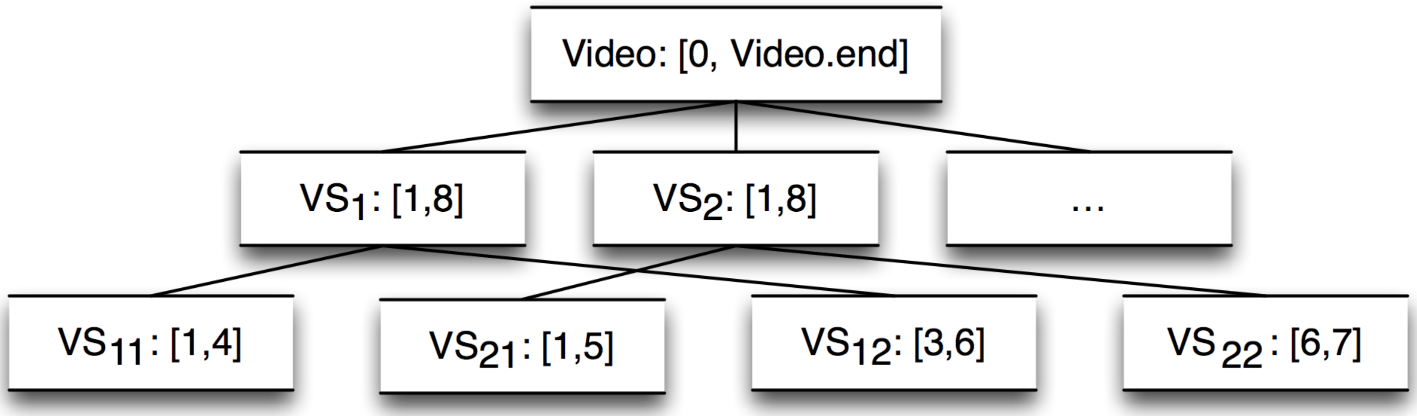
[1] P. Ciaccia and M. Patella. The M2-Tree: Processing Complex Multi-Feature Queries with Just One Index. In Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, Switzerland, 2000.
[2] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. Similarity Search the Metric Space Approach, volume 32 of Advances in Database Systems, chapter 3. Springer Verlag, 2006.

Multi-Feature Access Structures

TempoM2

- TempoM2: Access structure for multifeature temporal search in video repositories

- Data model: a Video is defined as: $V = \{VS_1, VS_2, \dots, VS_n\}$, $n \in \mathbb{N}$ and features a partial order over the video segments: $VS_x, VS_y \in V \ \land VS_x \neq VS_y | VS_x \preccurlyeq VS_y$ (due to overlapping and temporal wholes).



Multi-Feature Access Structures

TempoM2 [3]

- Consists of two levels:
  - First level bases on multi-feature M^2-tree
    - Indexing multiple low-level features of VS_x.
    - Balanced tree where all VS_x are leaf nodes.
    - Used for content-based search.
  - Second level represents temporal relations among the video segments.
    - Requires total order over video segments

[3] Mario Döller, Florian Stegmaier, Simone Jans and Harald Kosch. **T empoM2: A Multi Feature Access Method for Temporal Video Search.** *In Proceedings of the 18th International Conference on Multimedia Modelling.* January 4-6, Klagenfurt, Austria, 2012.

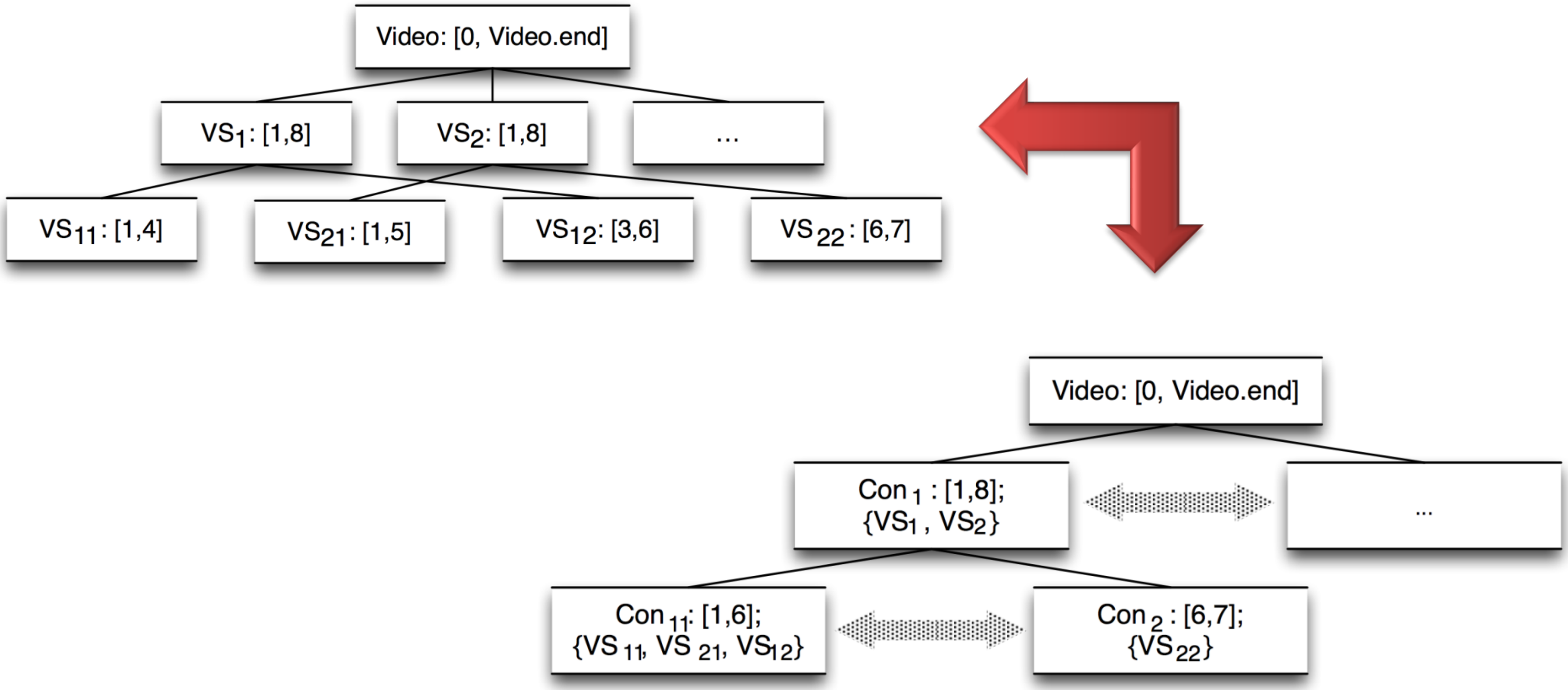
Multi-Feature Access Structures

TempoM2 [3]

- Total order over video segments is realized by introducing a container node Con_x with the following characteristic:
  - Every container node has a time interval $[Con_x.start, Con_x.end]$
  - For all VS_j covered by Con_x the following must hold:
    $\forall VS_j \epsilon\ Con_x\colon\ VS_j.start\ \geq\ Con_x.start\ \wedge VS_j.end\ \leq\ Con_x.end$
  - Relation to neighboring container:
    - Let Con_i be the parent node of Con_x, then: $Con_i.start\ \leq\ Con_x.start\ \wedge Con_i.end \geq Con_x.end$
    - Let Con_i be the left neighbor node of Con_x at the same level then: $Con_i.end \leq Con_x.start$

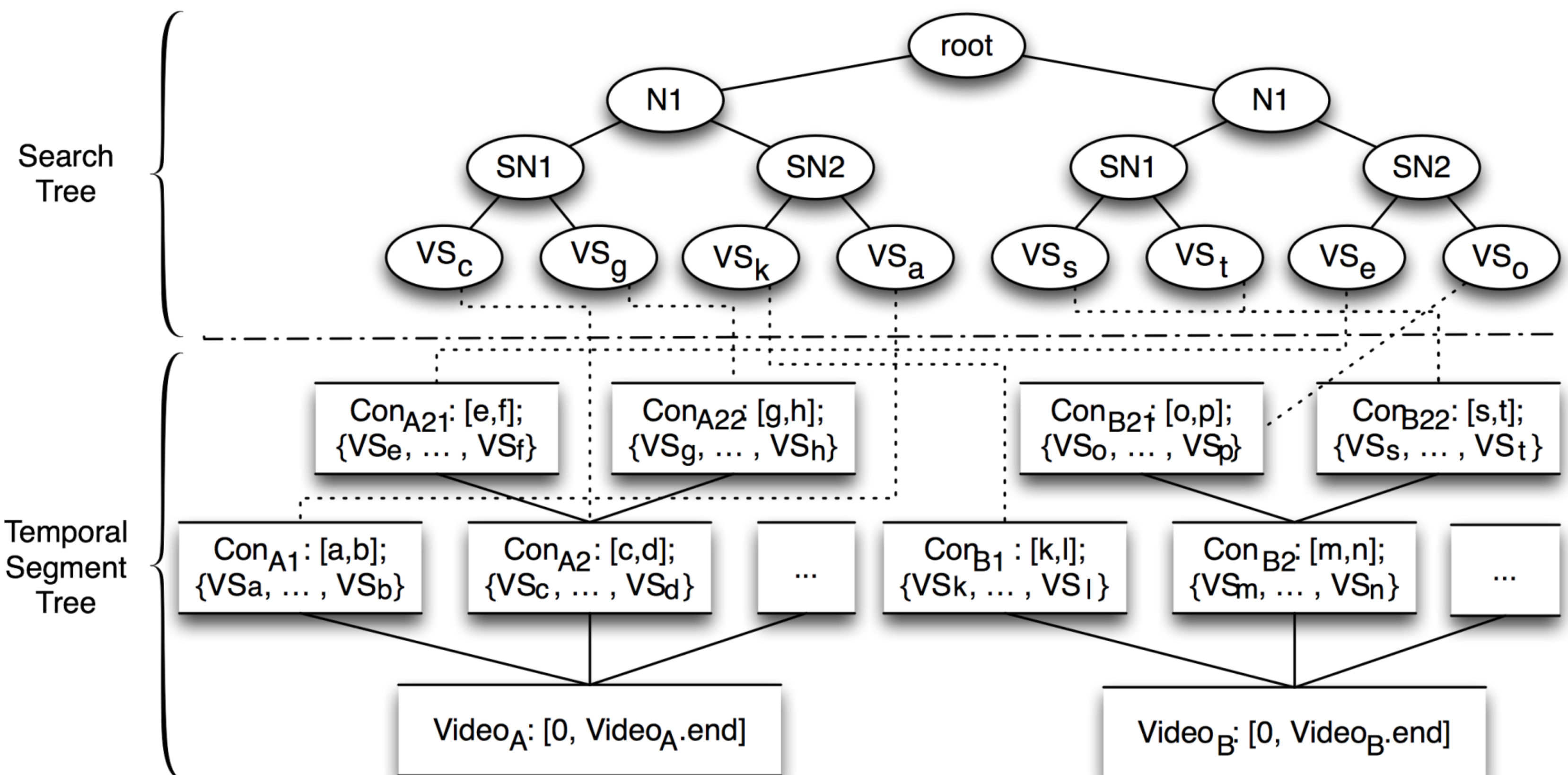
Multi-Feature Access Structures
TempoM2 [3]

- Total order over video segments for example video of previous slide:



Multi-Feature Access Structures

TempoM2 [3]



Multi-Feature Access Structures

TempoM2 - Search

- Is triggered by three input parameters according to the TemporalQuery type of the MPEG Query Format: *sourceResource, targetResource, relation*

- Example Query:

  freeKicks -> precedes-> goalScene

Note: the targetResource element is optional

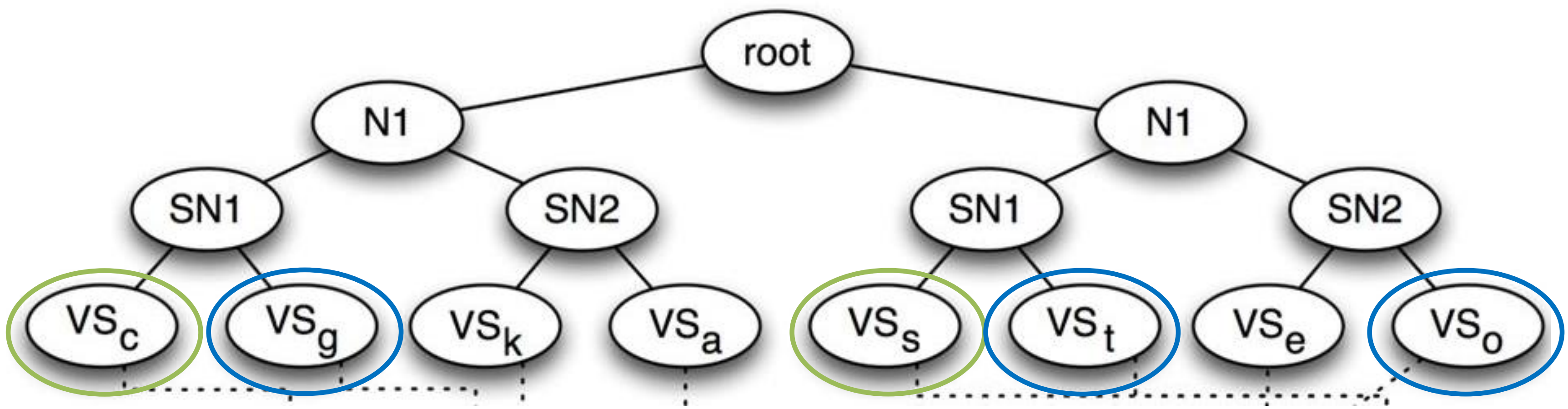
Multi-Feature Access Structures

TempoM2 - Search

- Evaluation of Temporal Search

    I. Similarity Search over the M^2-tree to receive two video segment lists: $\mathrm{S} = \{\mathrm{VS_i}, \dots, \mathrm{VS_m}\}$ for the sourceRelation and $\mathrm{T} = \{\mathrm{VS_j}, \dots, \mathrm{VS_n}\}$ for the targetRelation.

Example: $S = \{VS_c, VS_s\}$ *and* $T = \{VS_g, VS_t, VS_o\}$



Multi-Feature Access Structures

TempoM2 - Search

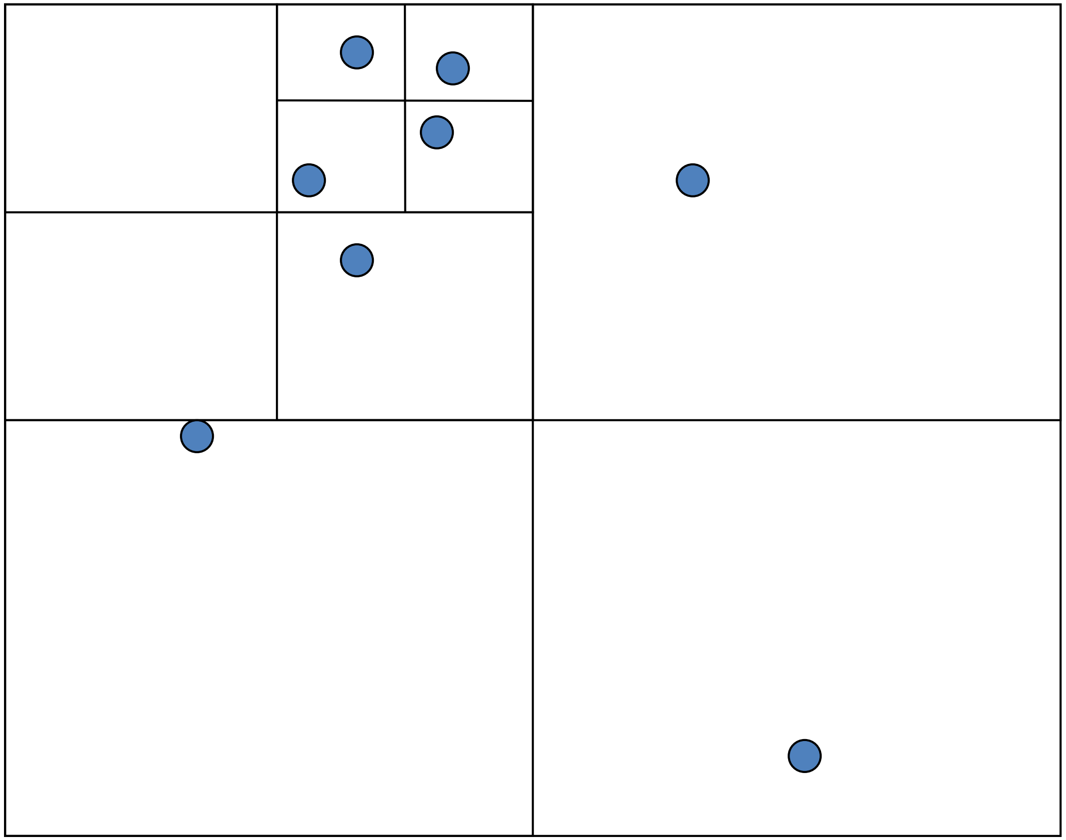
II. Temporal Search for all elements of the set S according to the direction of the temporal relation until an element of set T is found.

   I. Due to total ordering navigation direction is either horizontal (e.g., follows, meets, precedes, etc.) or vertical (e.g., starts, finishes, contains, etc.)

   II. Due to total ordering the following pruning criteria (excerpt) can be applied, where container node Con_N and $VS_x \in S$:
      - During: $Con_N.end \leq VS_x.start \lor Con_N.start \geq VS_x.end$
      - Overlaps: $VS_x.end \geq Con_x.end$
      - Starts: $VS_x.end \geq Con_x.end \lor VS_x.start \leq Con_x.start$
      - …

Main memory algorithms
Quadtree

- Simplest spatial structure on Earth !



Main memory algorithms
Quadtree

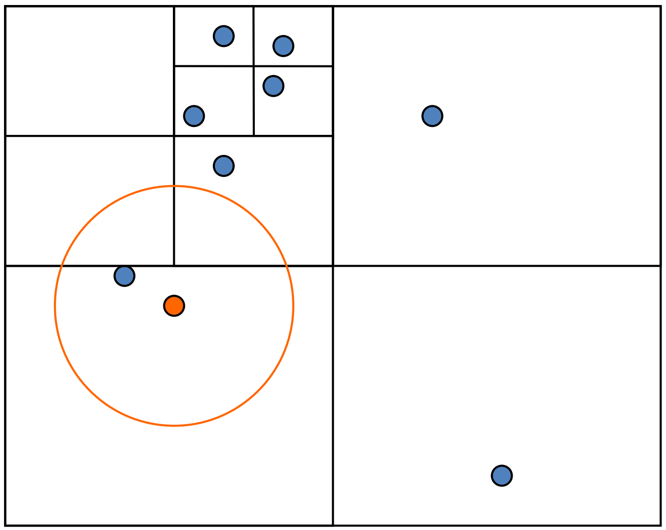
- Split the space into 2^d equal subsquares

- Repeat until done:
  - only one pixel left
  - only one point left
  - only a few points left

- Variants:
  - split only one dimension at a time
  - k-d-trees (in a moment)

Main memory algorithms
Quadtree – Range Search

- Near neighbor (range search):
  - put the root on the stack
  - repeat
    - pop the next node *T* from the stack
    - for each child *C* of *T*:
      - if *C* is a leaf, examine point(s) in *C*
      - if *C* intersects with the ball of radius *r* around *q*, add *C* to the stack



Main memory algorithms
Quadtree - Problems

- Simple data structure

- Versatile, easy to implement

- Problems?
  - Empty spaces: if the points form sparse clouds, it takes a while to reach them
  - Space exponential in dimension
  - Time exponential in dimension, e.g., points on the hypercube

Main memory algorithms

K-d tree [4]

[4] J. L. Bentley: *Multidimensional binary search trees used for associative searching*. Communications of the ACM 18, 9 (September 1975), S. 509–517.

- Main ideas:
  - only one-dimensional splits
  - instead of splitting in the middle, choose the split “carefully” (many variations)
  - near(est) neighbor queries: as for quadtrees
- Advantages:
  - no (or less) empty spaces
  - only linear space
- Exponential query time still possible

Main memory algorithms

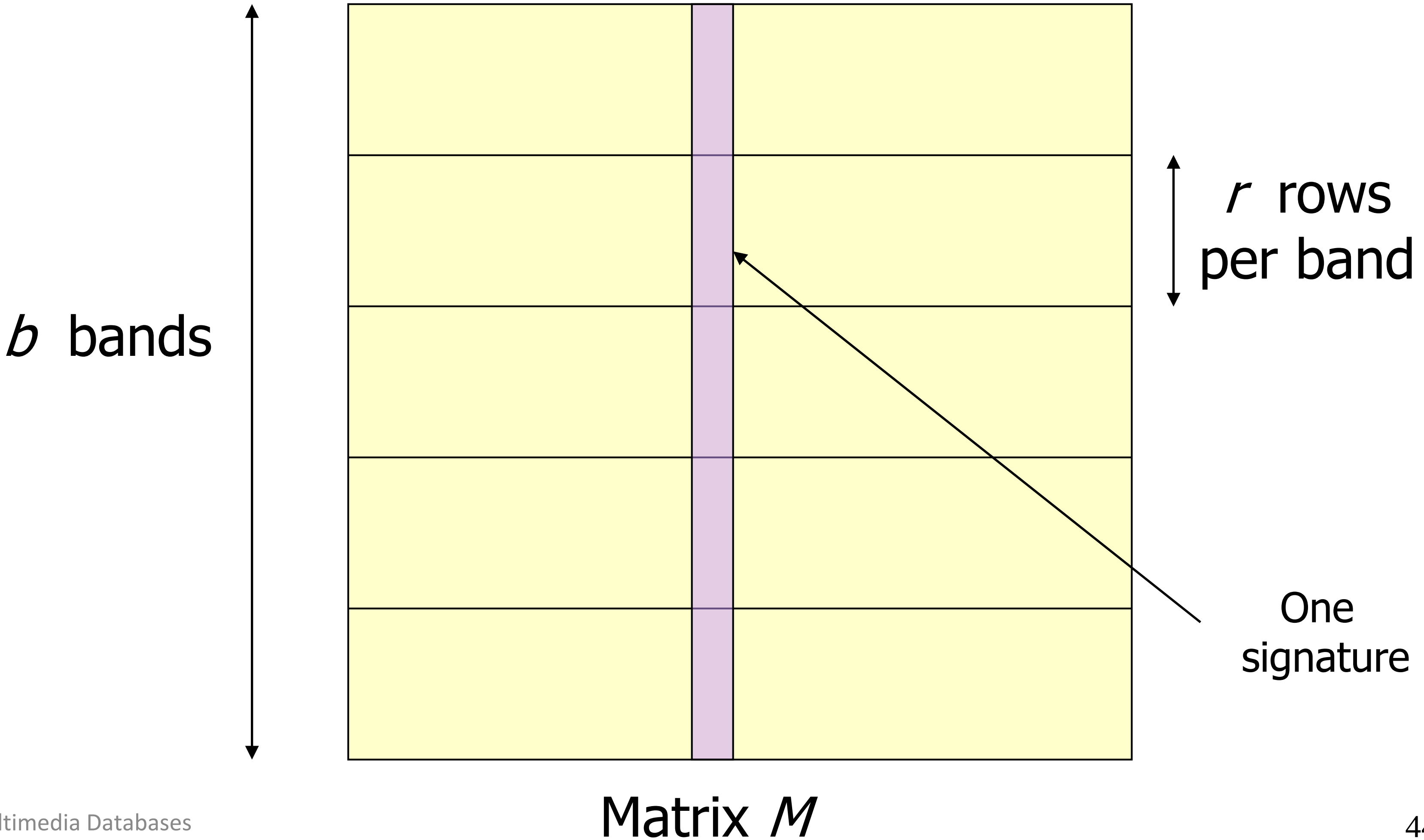
Locality-Sensitive Hashing [5]

- Big idea: hash columns of signature matrix *M* several times.
- Arrange that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.
- has been shown to be the most promising solution to Approximate NN search

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in
high dimensions via hashing. In VLDB, pages 518-529,
1999.

Main memory algorithms
Locality-Sensitive Hashing

- Partition into Bands.

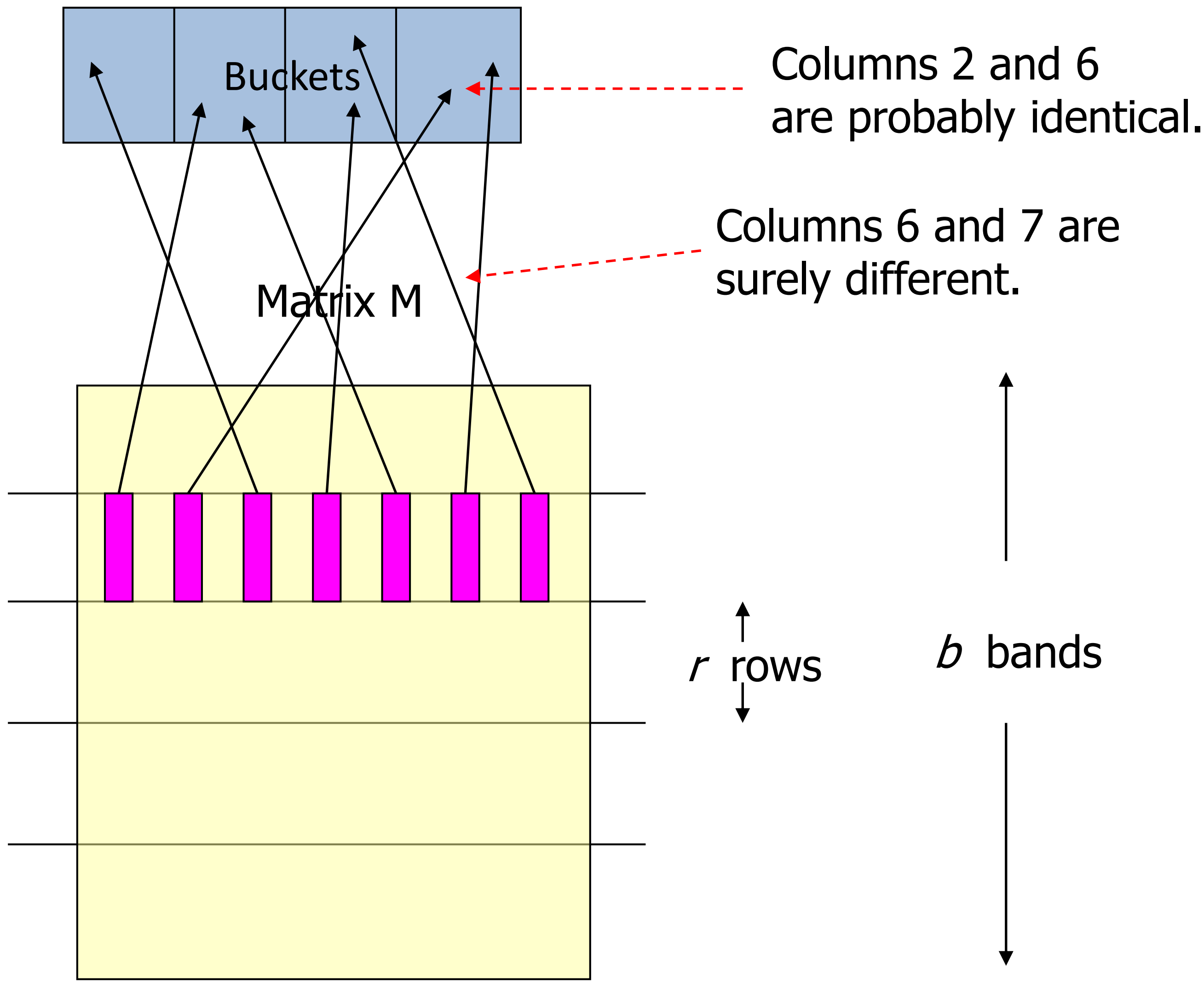


Main memory algorithms
Locality-Sensitive Hashing

- Partition into Bands
  - Divide matrix M into b bands of r rows.
  - For each band, hash its portion of each column to a hash table with k buckets.
    - Make k as large as possible.
  - *Candidate* column pairs are those that hash to the same bucket for $\geq$ 1 band.
  - Tune b and r to catch most similar pairs, but few non similar pairs.

Main memory algorithms

Locality-Sensitive Hashing

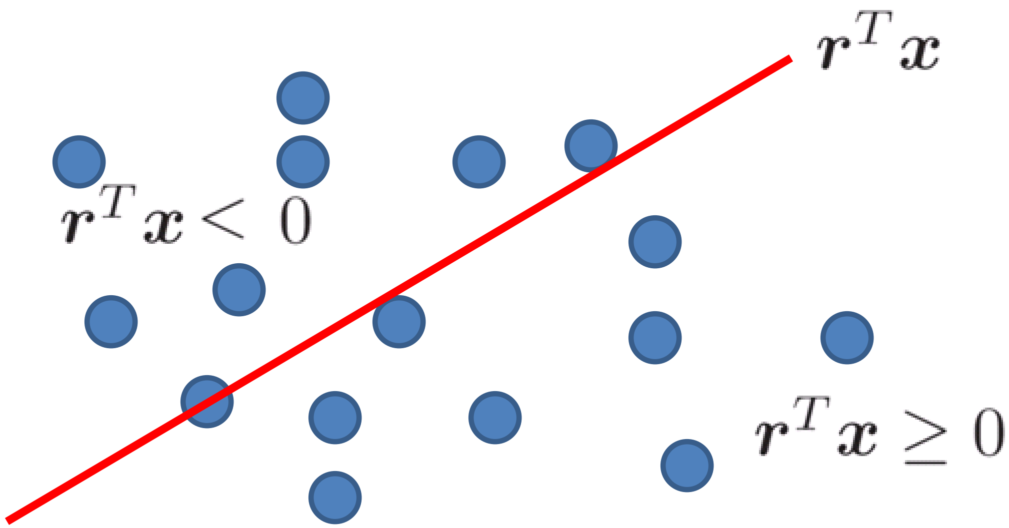


Main memory algorithms
Locality-Sensitive Hashing

The hashing function of LSH to produce Hash Code

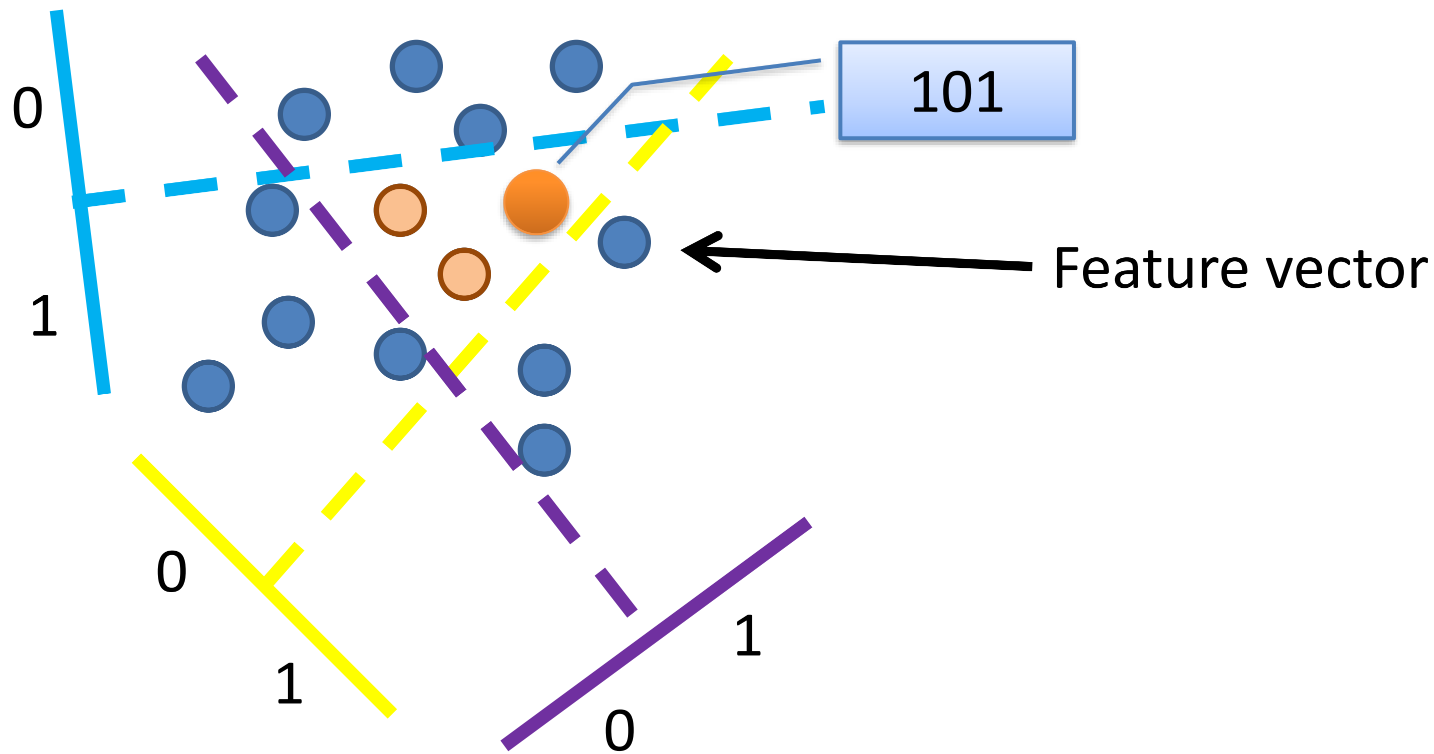
$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$\boldsymbol{r}^T\boldsymbol{x} \geq 0$ is a hyperplane separating the space (next page for example)

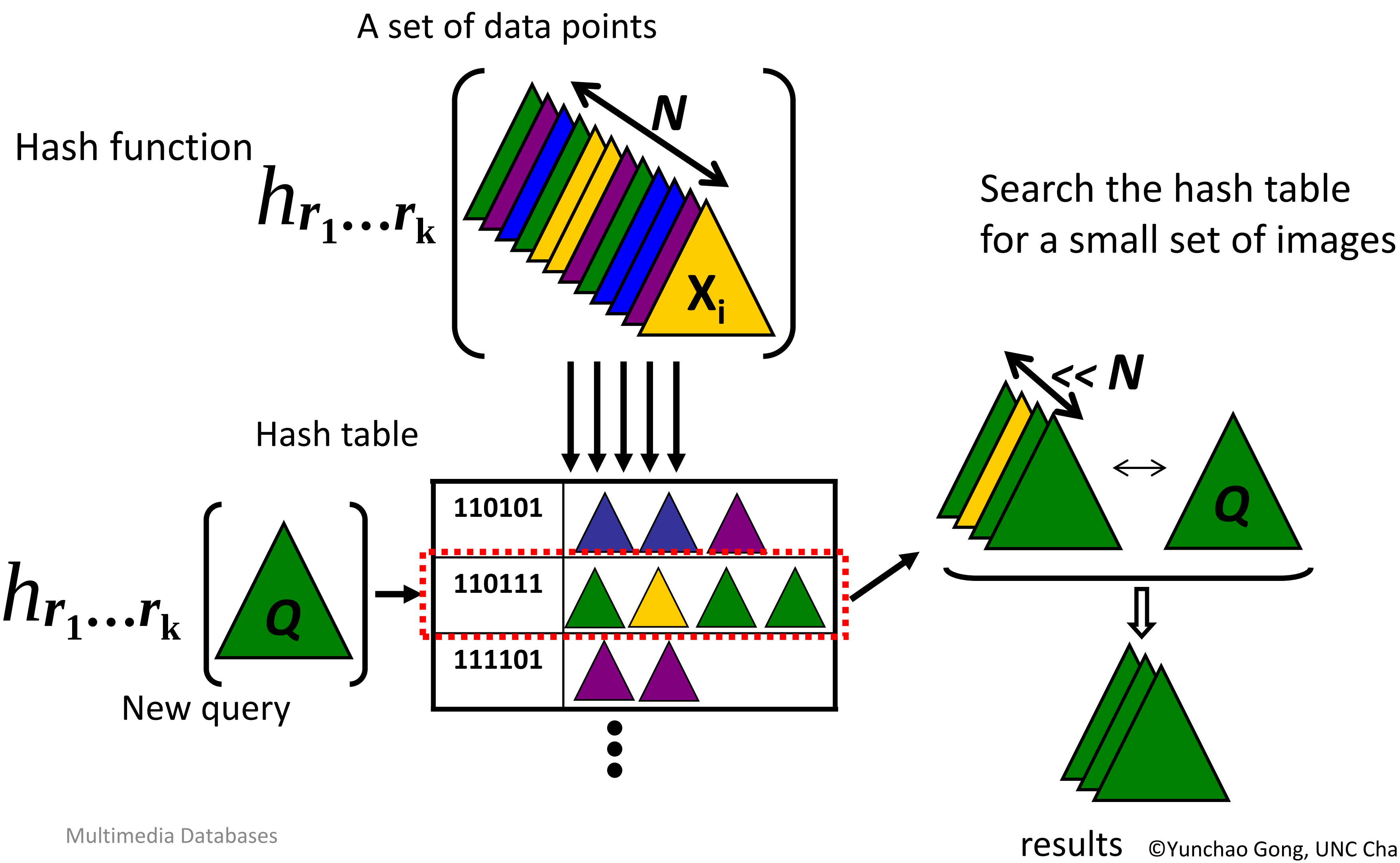


Main memory algorithms
Locality-Sensitive Hashing

- Take random projections of data $r^T x$
- Quantize each projection with few bits



Main memory algorithms
Locality-Sensitive Hashing - Search



Main memory algorithms

Locality-Sensitive Hashing – State of the Art

- Drawback of original LSH
  - hundreds of or even more hash tables are constructed, which causes a huge space requirement
- To reduce the number of hash tables, Multi-Probe LSH [6] was proposed, which could find more similar points from a single hash table by exploring the buckets near the one into which the query point falls
- C2LSH [7] proposes an interesting method to collect candidate points, called dynamic collision counting.
- SortingKeys-LSH (SK-LSH) [8], which verifies candidates in the unit of disk page. As points with close compound hash keys are arranged together in the disk space, only a small number of disk page accesses are required

Main memory algorithms
Locality-Sensitive Hashing – State of the Art

[6] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. In ACM Multimedia, pages 209-218, 2008.

[7] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality sensitive hashing scheme based on dyanmic collision counting. In SIGMOD, pages 541-552, 2012.

[8] Yingfan Liuz, Jiangtao Cuiz, Zi Huangx, Hui Liz, Heng Tao Shen. SKLSH: An Efficient Index Structure for Approximate Nearest Neighbor Search. In Proceedings of the VLDB Endowment, 7(9):745-756, 2014.

Content-based Queries

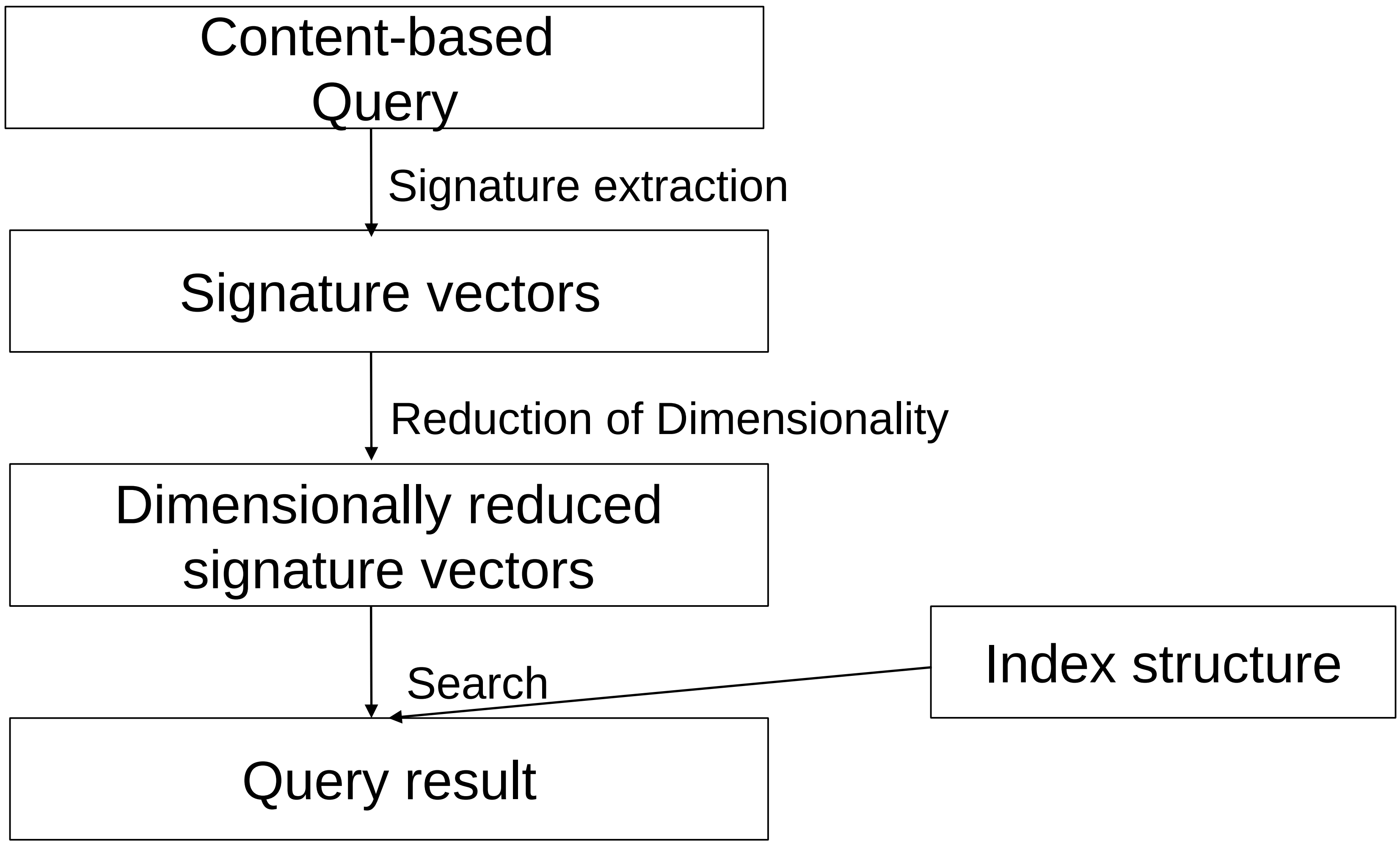


Table of Contents

Access Structures

Types of Queries for MMDB I

- Similarity query:
  - For a given object
  - k most similar searched
  - generalization of the k-Nearest-Neighbor-Search (k-NN Search)
  - Algorithms correspond to those for NN-Search
  - But additional option: ignore a node if its distance to the query vector is too high

Types of Queries in MMDB II

- Range query:
  - For a given region
  - Search all objects that intersect with the region
  - Reduce the computation time by ignoring nodes
    -> nodes which intersection with query space is lower than a threshold

  - Near neighbor (range search): find one/all points in P within distance r from q
  - Approximate near neighbor: find one/all points p' in P, whose distance to q is at most $(1+\varepsilon)$ times the distance from q to its nearest neighbor

Nearest-Neighbor-Search I

- Consider as a type of range query.
- Variable radius.
- Condition: the nodes are enclosed in Minimum-Bounding-Rectangles (MBR).
- Given for R-trees (and variations).
- Fast search requires fast reduction of the radius.

Nearest-Neighbor-Search II

- Reduction of the radius:
  - Order not yet visited nodes in priority queue
  - First visit nodes in first positions

- Priority determined from:
  - minimal distance between query point and node's center
  - minimal distance between query point and node
  - minimizing the maximal distance

Table of Contents

Access Structures

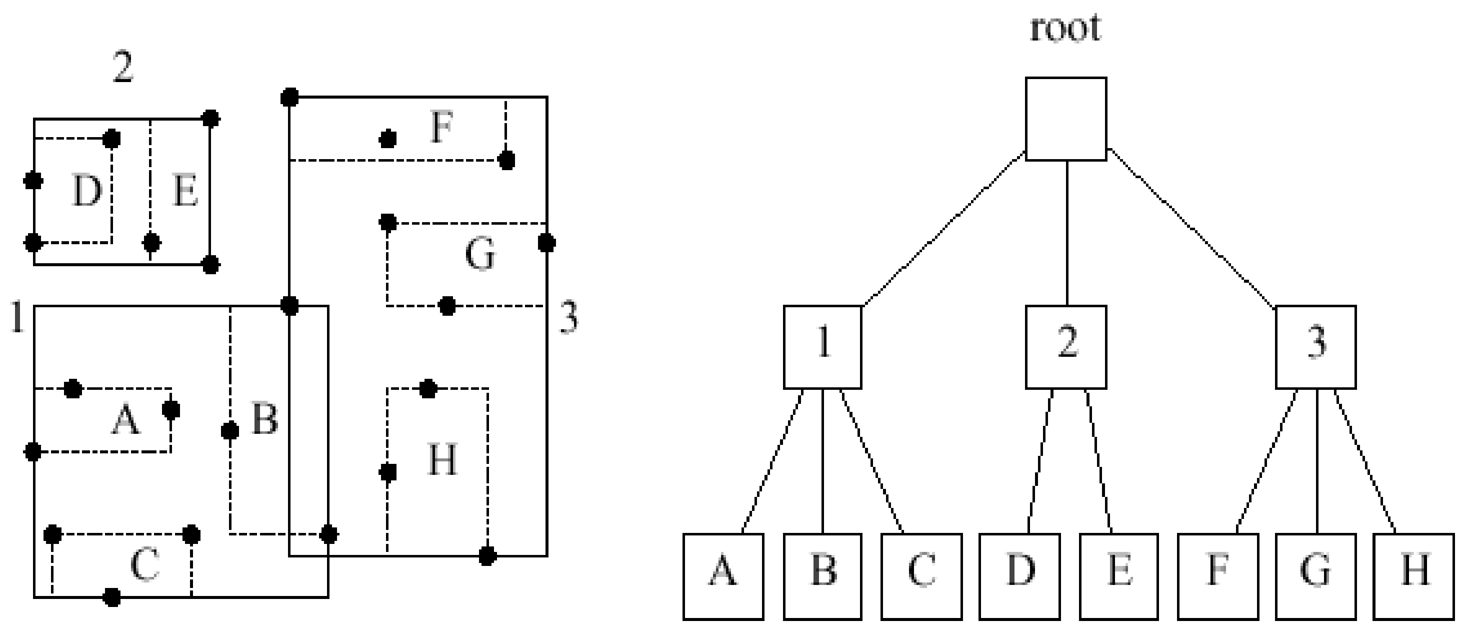
Case study: The SR-tree as an Example of Index Structure in Multimedia Databases

- The SR-tree is an index structure for high-dimensional Nearest-Neighbor-search

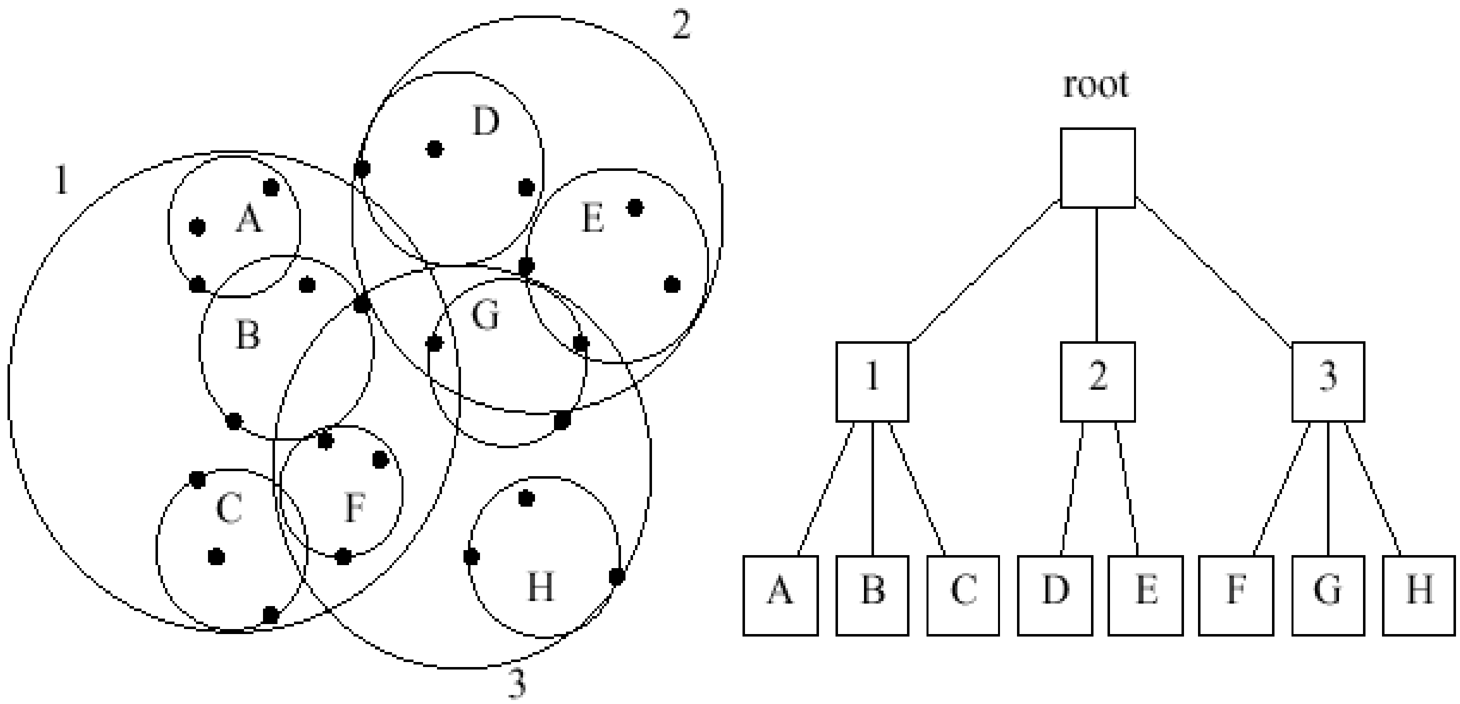
SR-tree : Introduction

- SR-tree stands for "Sphere/Rectangle-tree".
- is an extension of the R*-trees and of the SS-trees.
- The regions (bounding boxes) of the SR-tree correspond to intersections of rectangles (bounding rectangles) and circles (bounding spheres).

Structure of R*-trees



Structure of SS-trees



Regions of an SR-tree

- The diameter of a region in an SR-tree is comparable with:
  - The diameter of a bounding sphere of SS-trees
  - The diagonal of a bounding rectangles of R*-trees

Properties of SR-trees I

- Bounding rectangles divide the points in regions with lower volumes. They usually have a higher diameter than bounding spheres, especially in high-dimensional spaces.

- Bounding spheres divide the points in regions with lower diameters. They usually have a higher volume than bounding rectangles.

Properties of the SR-trees II

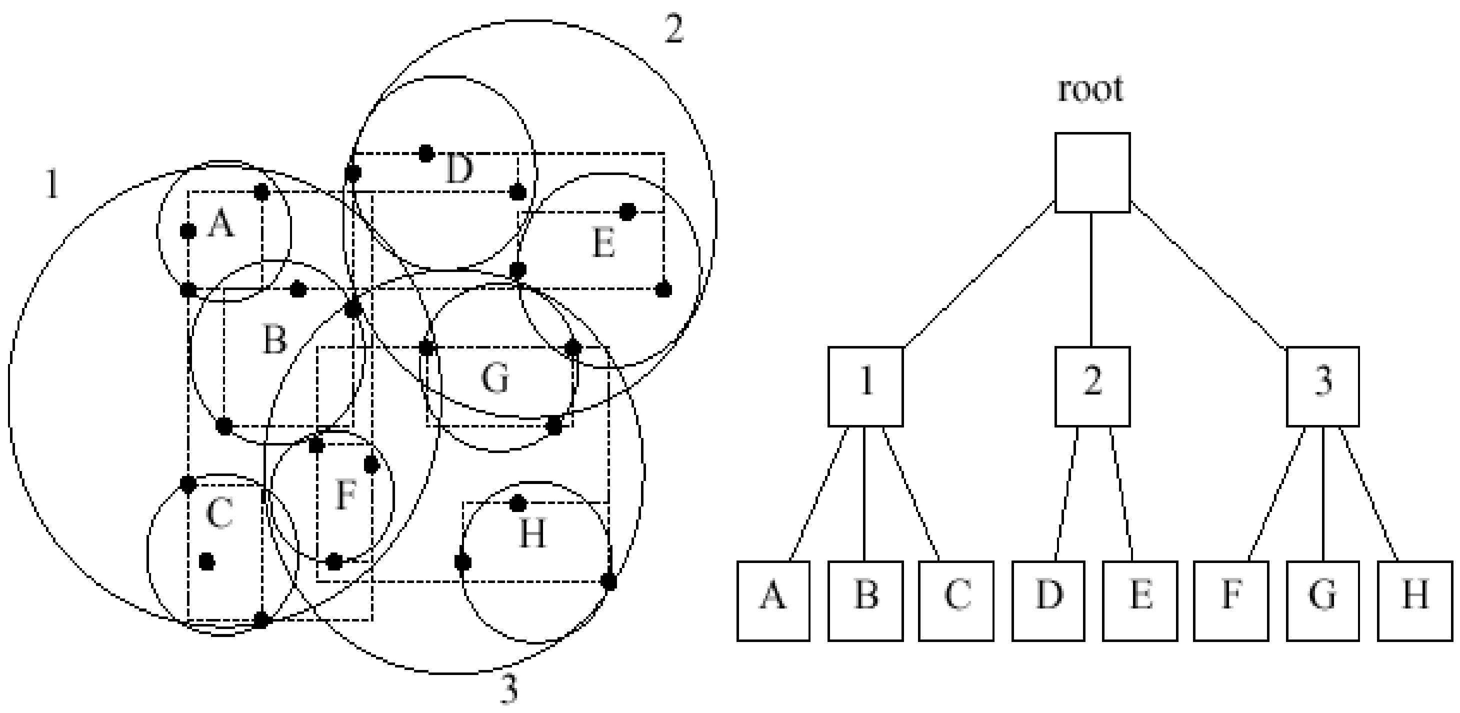
- SR-trees combine the use of bounding spheres and bounding rectangles
- The properties of both types of regions are complementary, thus allowing their diameters to divide points in regions with lower volumes and diameters

Comparison of R*-tree, SS-tree and SR-tree

	R*-tree	SS-tree	SR-tree
Shape of region	Rectangle	Circle	Intersection of rectangle and circle
Tree construction strategy	Minimize volumes and overlaps	Minimize diameters	Minimize diameters
Diameter	4	1	1
Volume	60	10^9	1

*** The values for diameter and volume were determined from a test performed with 16-dimensional data points.**

The structure of SR-trees



Insertion Algorithm

- The insertion algorithm of SR-trees is based on that of the SS-trees, which makes use of the center of the bounding spheres.
- By searching, when going down in the data structure, the subtree with the most similar center to the new entry is selected.
- In the case of SR-trees, both regions (bounding spheres and bounding rectangles) are updated.

Delete Algorithm

- The delete algorithm of SR-trees is similar to that of R-trees.
- When removing an entry that causes no underfilling of leaves/nodes, the entry is simply removed without any other action.
- Otherwise, the underfilled leaves/nodes are removed and all corresponding entries are added again.

Nearest-Neighbor-Search with SR-tree

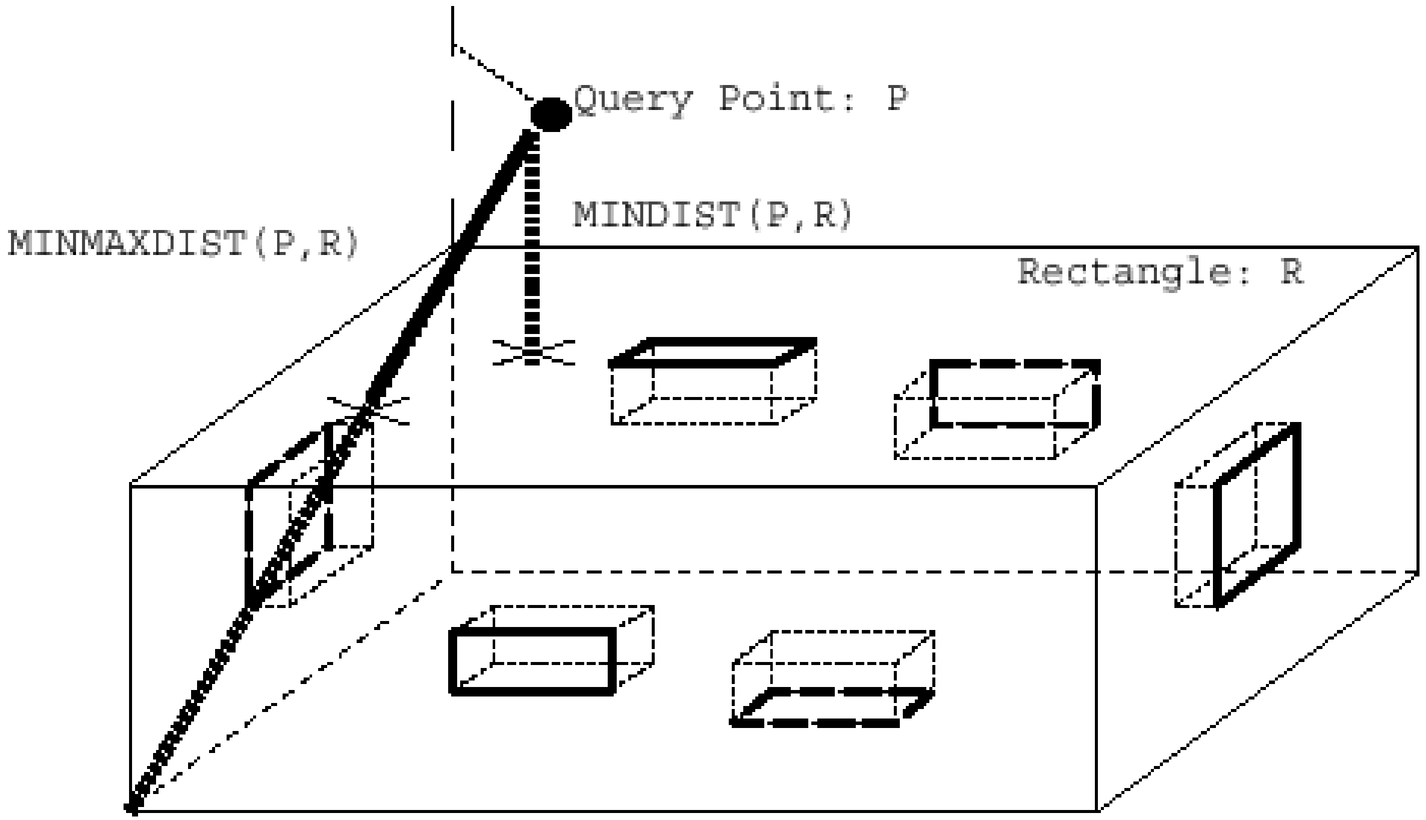
- The algorithm performs an ordered depth search.
- A number of points that are the closest to the query point are found, and a set of candidates is built.
- The set of candidates is inspected by visiting each leaf which region overlaps the set of candidates.
- Once all leaves have been visited, the last candidate is returned as the result.

Nearest-Neighbor-Search – Definition I

- Minimal distance (MINDIST):
  Euclidian distance between the query point and the region

- Minimax-distance (MINMAXDIST):
  minimal value of all maximal distances between the query point and the corresponding points on all n axes

Nearest-Neighbor-Search – Definition II



Pruning When Searching in SR-trees I

- Pruning enables to identify subtrees that cannot possibly contain the searched object. They are then excluded from further processing in order not to be searched in vain.

- A region R1 is excluded if its MINDIST is higher than the MINMAXDIST of another region R2: indeed it cannot possibly contain the nearest neighbor (downward pruning).

Pruning When Searching in SR-trees II

- If the actual distance from query point P to a given object O is higher than the MINMAXDIST of a region, the object is excluded (upward pruning).

- A region with a MINDIST higher than the actual distance from query point P to an object O is excluded (upward pruning).

Recursive Procedure Nearest-Neighbor-Search I (Leaf Node)

```
If Node.type = LEAF then
   For I := 1 to Node.count

      /* Compute distance of points to region */
      dist := objectDist(Pt, Node.region)

      /* if the computed distance is lower than the distance to the
previous region, the leaf is nearest neighbour */
      if (dist < Nearest.dist) then
            Nearest.dist := dist
            Nearest.region := Node.region
```

Recursive Procedure Nearest-Neighbour-Search II (Inner Nodes)

```
/* Not a leaf node =>
   order, sort, apply pruning & search next node */

Else
   createSonsList(Point, node, sonsList)

/* Sort list, so that the next son is first selected */
   sortSonsList(sonsList)

   /* Apply downward pruning */
   number = pruneSonsList(node, point, Next, sonsList)
   For I := 1 to number
        nodeNew := node.currentSon

        /* Continue searching in the branch of the next son */
        nearestNeighborSearch(nodeNew, point, Next)

        /* Apply upward pruning */
     number := pruneSonsList(node, point, Next, sonsList)
```

Strengths of SR-trees

- SR-tree divides points in regions with small volume and low diameter.

- Putting the points in smaller regions improves their disjunctivity.

- Smaller volumes and diameters increase the performance of the Nearest-Neighbor-Search

Weaknesses of SR-trees

- Higher cost of creation.

- The size of the nodes increases with the dimensionality.

- Reducing the bifurcations can require reading more nodes when executing queries.

- Performance of query execution may suffer as a result.

Table of Contents

Access Structures

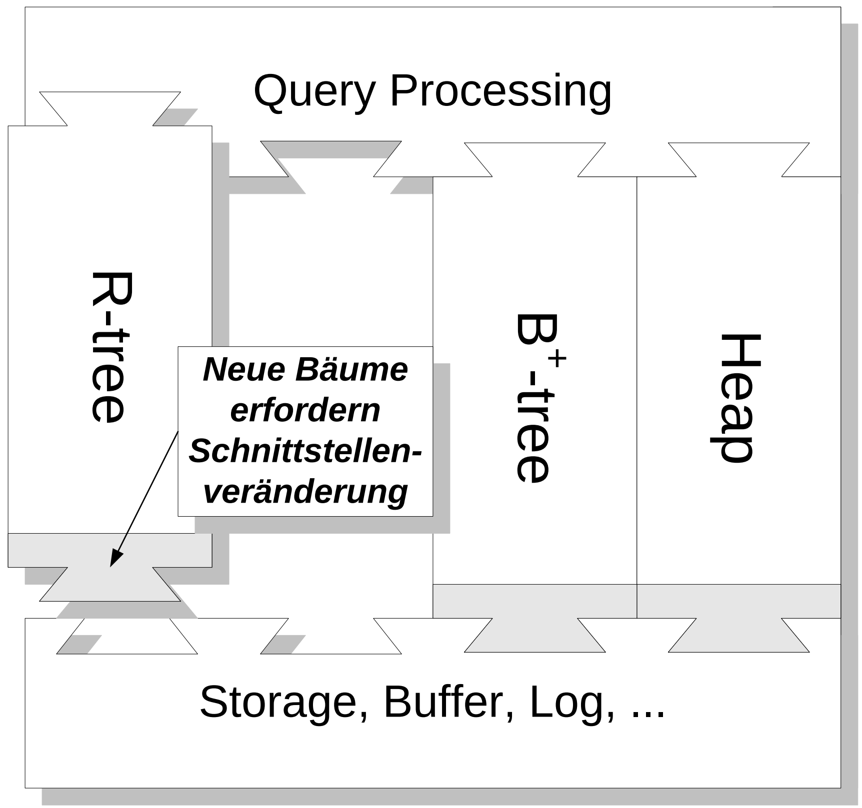
GiST – Generalized Search Tree Framework

- http://gist.cs.berkeley.edu/

- The Generalized Search Tree (GiST) provides an abstraction of the "type of tree" actually used (from the previously presented B+ tree and R-tree variants).
  - Similarities in insert/delete/search and even „concurrency control“ enable the use of "templates".
  - B+ trees are extremely important (and simple enough to be specialized), in practice they are available in all commercial DBMSs.
  - GiST offers an *alternative for the integration of various types of trees in an ORDBMS*.

Starting Position

- Problems of previous solutions: “Concurrency” and “Recovery” must be re-implemented for each new search tree.

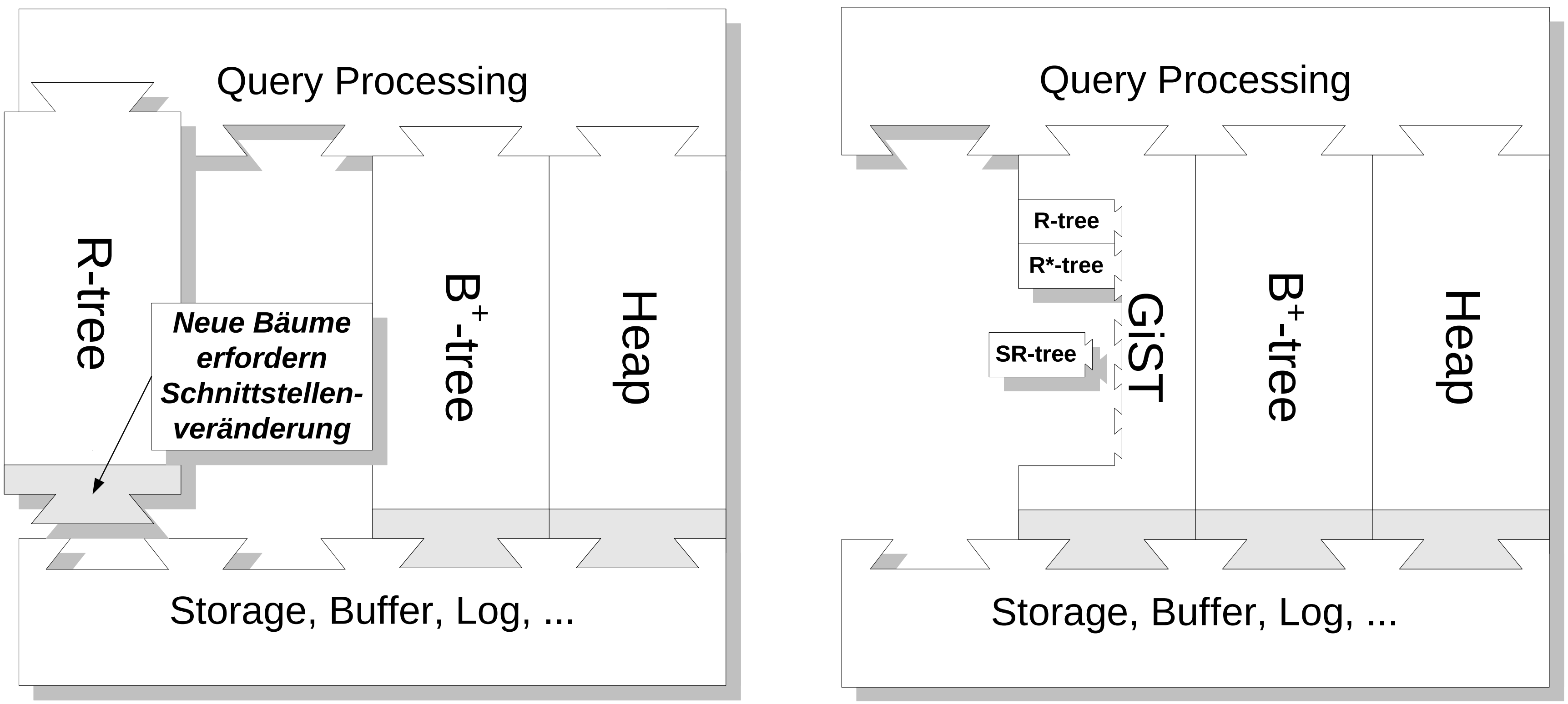


Generalized Search Tree: Overview

- Generalized Search Tree (GiST) = “template index structure”
  - Extensible set of datatypes and queries
  - Examples available in downloadable package:
    B-trees, R-trees, SR-tress …
  - Details: Hellerstein, Naughton, Pfeffer, VLDB ’95

- GiST offers:
  - A “basic structure”: height-balanced tree
  - Template algorithms: search, insert and delete
  - No limitation wrt. keys and their distribution on a node

Generalized Search Tree: Comparison



Current research paper in this direction:

1. Stefan Sprenger, Patrick Schäfer, Ulf Leser; BB-Tree: A main-memory index structure for static multidimensional workloads; In Proceedings of the Int. Conf. on Data Engineering; Macau, China; 2019
2. Angjela Davitkova, Evica Milchevski, Sebastian Michel; The ML-Index: A Multidimensional, Learned Index for Point, Range, and Nearest-Neighbor Queries; In Proceedings of the23rd International Conference on Extending Database Technology (EDBT); 2020
3. Stefan Sprenger, Patrick Schäfer, Ulf Leser; Multidimensional range queries on modern hardware; In Proceedings of the 30th International Conference on Scientific and Statistical Database Management; 2018

The end