# Outline

- Searching in Scientific Databases: Introduction
- Feature spaces and proximity measures
- Feature transformation for text data
- Algorithmic Paradigms for Similarity Query Processing

# Algorithmic Paradigms for Similarity Query Processing in Scientific DBs

- Recap: One of the main considerations:
  - Generality: Avoid the necessity to develop similarity search algorithms for each application separately
  - Efficiency: Similarity search methods should allow efficient query processing
- We need appropriate data organization and query search concepts that meet the above two aims, generality and efficiency.

→ two algorithmic paradigms for similarity query processing:

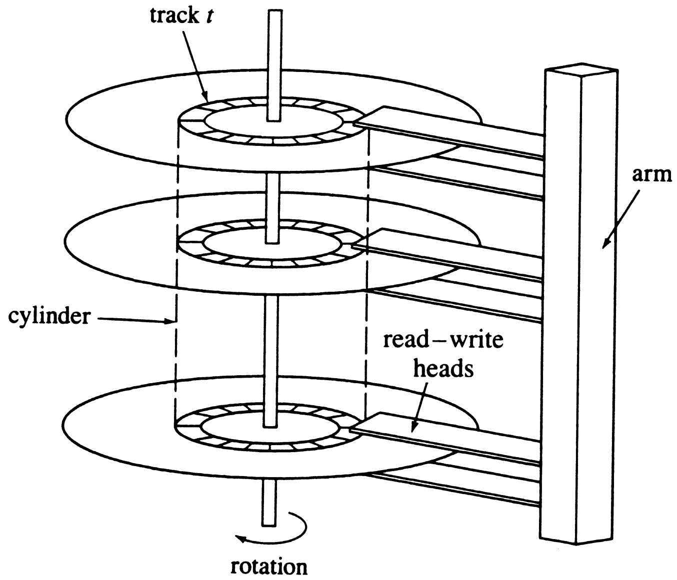
  - Indexing: Using spatial (or metric) indexes to organize the data in an efficient way and allow efficient similarity query processing
  - Multi-Step Query Processing: Use fast approximative search methods as a filter to retrieve a potentially small set of candidates to be refined afterwards.

# Index-based Similarity Search

- Principle of Indexes in Databasesystems

  [Böhm, Berchtold, Keim. ACM Computing Surveys, 2001]

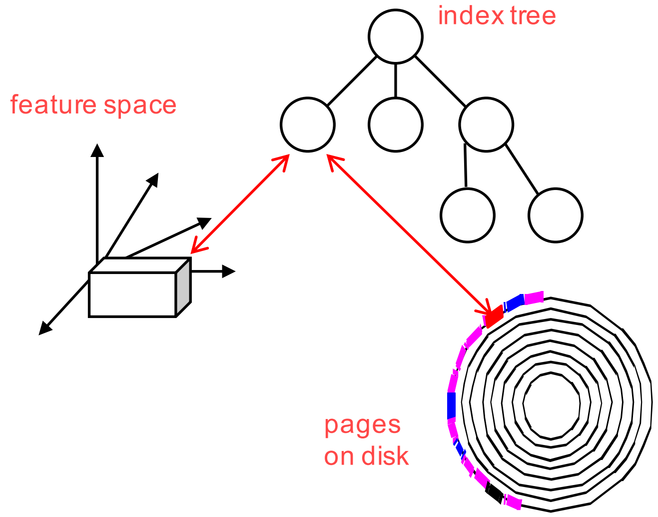
  - We assume that objects are stored on a disk memory.
  - The disk memory technically covers multiple disks.
  - Each disk has a number of tracks.
  - Each track consists of a bit sequence organized as sequence of bytes.




  - Disks (and also other data storage devices) are page oriented organized.
  - Organize DB objects such that only the those pages that store the "relevant" objects are loaded from disk.

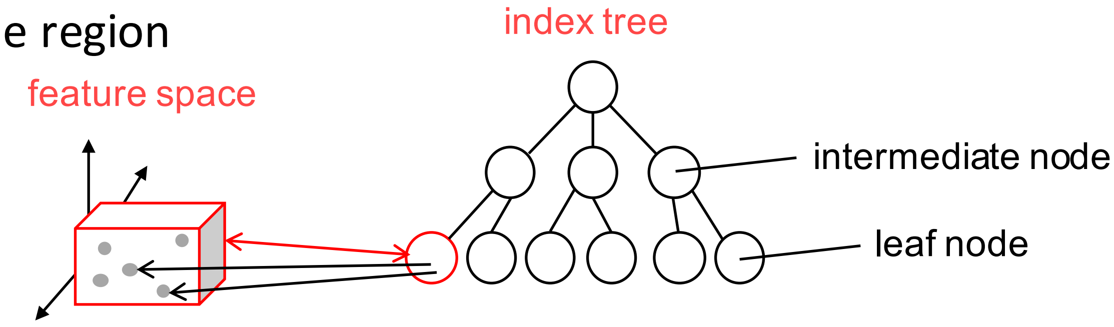
# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - Following the feature-based similarity search paradigm, similarity queries are efficiently supported by spatial index structures.
  - Tree-like organization:
    each node of the tree corresponds to a
    - page of the disk
    - region of the feature space (spatial **key**)
  - The capacity of a node/page corresponds to the block size of the hard disk.
  - Two types of nodes (= “index entries”)
    - Leaf (data) nodes
    - Inner (intermediate/directory) nodes
  - Note: In the following, both terms “page” and “node” have the same meaning and are interchangeable.



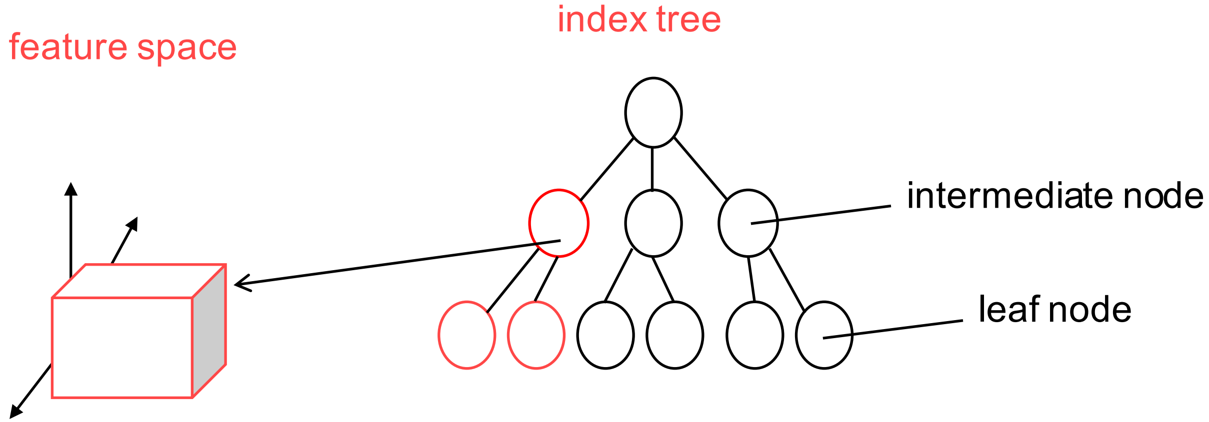
# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - Leaf (data) nodes correspond to data pages that store the objects on disk
  - Inner (intermediate) nodes are called directory nodes and store directory entries including for each child node C:
    - a pointer to C's address on the disk
    - a description of the region of the feature space represented by C
  - Each object o (or a reference to o) is stored in exactly one data node (a node in the region of which this object is contained in)



# Index-based Similarity Search

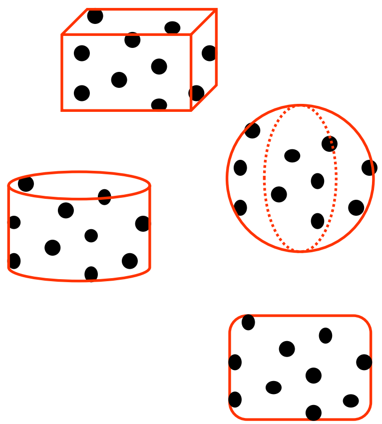
- Principle of Spatial Indexes in Databasesystems
  - Each region r (spatial key) is stored in exactly one intermediate node (a node in the region of which region r is contained in).



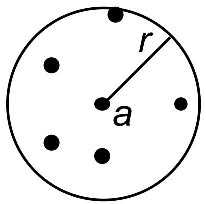

  - Similar objects, (i.e. nearby objects) are stored in the same (or nearby) region/subtree.
  - Each node (page) is associated with a (spatial) region (spatial approximation) called page region. A page region serves as spatial key and covers all objects organized in the subtree below the corresponding node (or directly within the corresponding leaf node).

# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - How do regions (spatial keys) look like?
    - In the case of Euclidean vector data:
      - Axis-parallel minimum bounding rectangles (MBR)
      - (R-Tree, R*-Tree, X-Tree, …)
      - Minimum bounding spheres (SS-Tree, …)
      - Minimum bounding cylinder (TV-Tree, …)
      - Combinations thereof (SR-Tree, …)



    - In the case of general metric (non-vector) data:
      - Reference object + covering radius/distance threshold (e.g. M-tree)

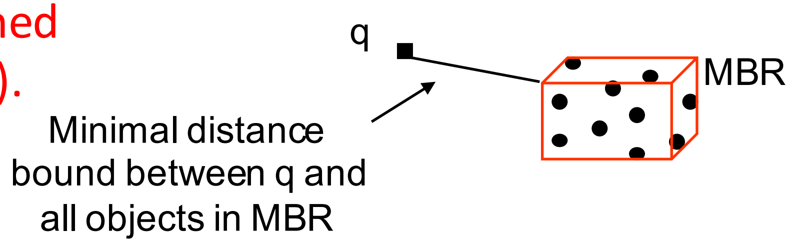



  - Regions achieve that similar objects are mostly stored on similar pages.

# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - A page region associated with a page (node) P always covers entirely all objects stored in P (if P is a leaf node) or all objects stored in the subtree below P (if P is a directory page (node)).
  - This is an important requirement to guaranty that reported result sets are complete, i.e. no false drops.
  - Conservative approximation, lower-bounding property: The distance between an object (e.g. query object) and all objects covered by a page region can always be lower bounded, e.g. the minimal distance between a point q and the MBR.

=> Objects in (or below) a page (node) can be safely pruned (i.e. ignored) without missing correct results (false drops).

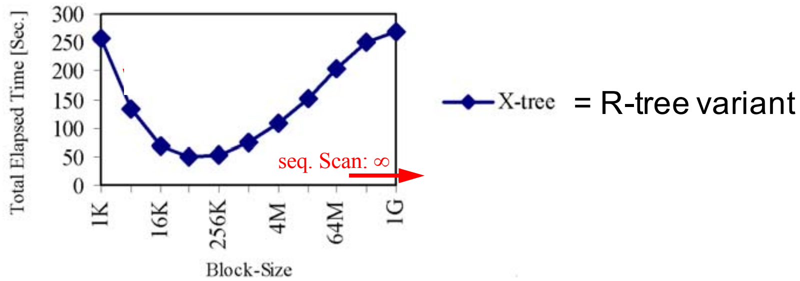


# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - Principal Goal
    - If the tree is balanced it has a height of O(log n), where n = number of objects in database. [which means $\exists c \in \mathbb{R}$, s.t. $\forall 0<n<\infty$: height of tree < c log n]
    - In the best case, only one node (page region) at each level of the tree qualifies for the query predicate
    - => the traversal of the index tree for answering the query is limited to one path from the root node to a leaf node
    - Costs: O(log n · <costs for evaluating the query predicate>)
    - Worst-case: the complete tree is traversed (even worse than seq. scan)
    - Efficient update of the tree (insert, delete, update), i.e. updates affects as few pages per level as possible (aim: one page per level).

# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - Parameters that affect the performance
    - Overlap of page regions: the higher the overlap the higher the possibility that more search paths need to be traversed
    - Node capacity:
      - Low capacity results in a high tree, i.e., long search paths
      - High capacity results in a flat tree, i.e. short search paths, but
        - High cost for accessing each node/page (loading a page is expensive).
        - Many, potentially irrelevant, entries have to be evaluated per page access → degenaration to sequential scan
      - A good trade-off is mandatory.
    - How to determine an appropriate node capacity? empirically

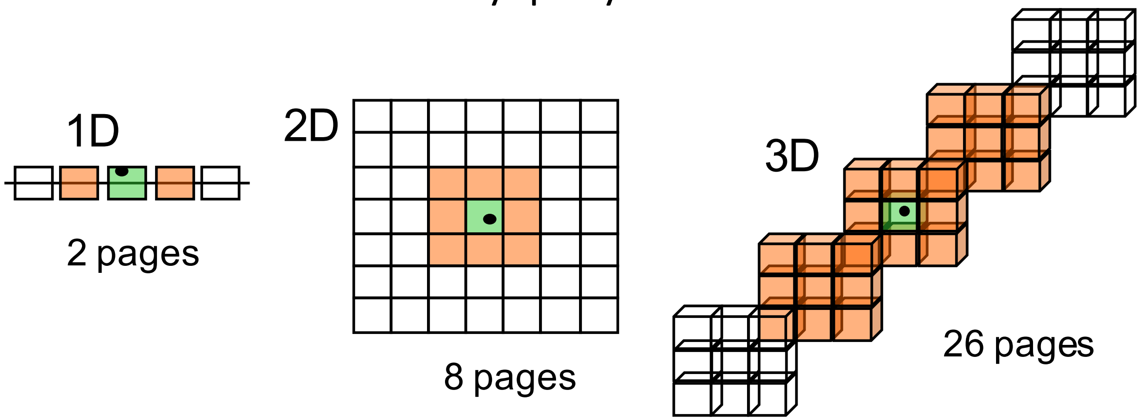


# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - Parameters that affect the performance
    - Selectivity of the query predicate:
      A large number of qualifying objects implies a large number of qualifying nodes so usually multiple search path need to be traversed.
      Note: Selectivity σ := $\frac{|result|}{|DB|}$ => small result ~ high selectivity
    - High selectivity is a characteristic for similarity queries.
      (usually only holds for low dimensional feature spaces, up to 20 dimensions, see "curse of dimensionality").

## Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems
  - The *Curse of Dimensionality*
    - Performance of spatial index supported queries degenerates with increasing dimensionality.
    - Generally holds for high-dimensional (>50 dimensions) vector spaces.
    - Observation: Number of pages to be accessed for a similarity query



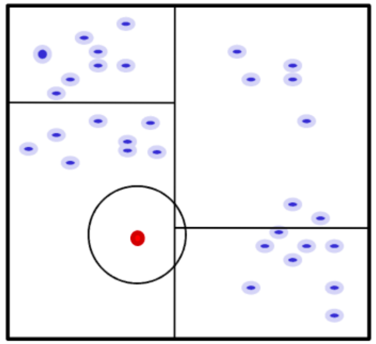

    - Generally: $3^d$-1 pages
    - Higher page region overlap
    - Consequence: often the whole directory of the index accessed

# Index-based Similarity Search

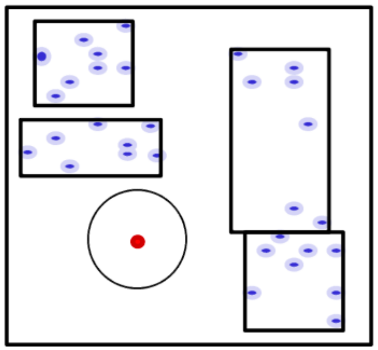
- Principle of Spatial Indexes in Databasesystems (cont.)

  Two classes of spatial index structures:

  - **Space Partitioning Index Methods**
    - Page regions built by recursive decomposition of the object space using split planes (binary or n-ary splits)
    - Usually used as main memory index structure (low branching factor of the tree nodes).

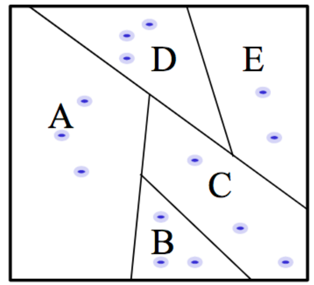


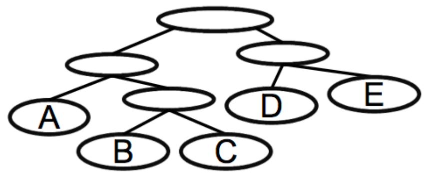
  - **Data (Object) Partitioning Index Methods**
    - Page regions built by minimal spatial covers of groups of objects that are derived by decomposition of the set of objects into n groups.
    - Appropriate for secondary memory index method.

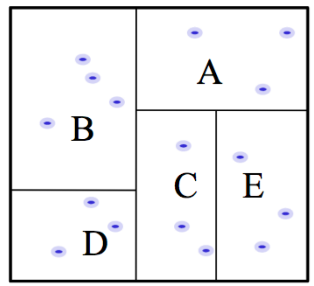


# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Space Partitioning Index Methods
    - Binary Space Partitioning Tree (BSP-Tree):
      - Root covers entire object space
      - Each inner node has two children nodes
      - Data (objects) only stored at leaf level
    - Prominent variant: kD-Tree
      - max. M entries and min. M/2 entries in each page
      - Page overflow => Axis parallel split with alternating split axis
      - Balanced split, i.e. split s.t. 50% of objects on each side of the split
      - Page underflow => Merging sibling nodes

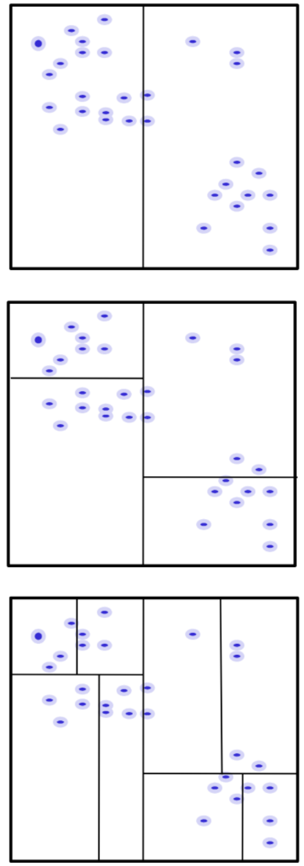


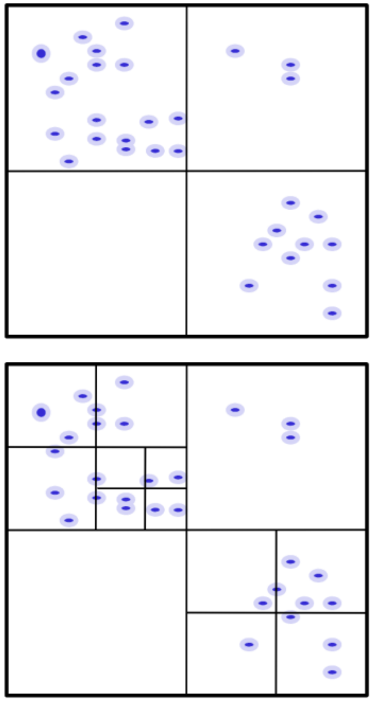
# Index-based Similarity Search

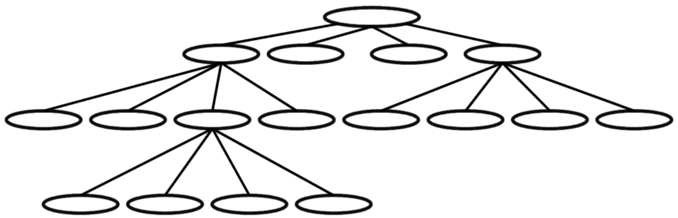
- Principle of Spatial Indexes in Databasesystems (cont.)
  - Space Partitioning Index Methods: BSP-Tree
    - Problems:
      - No balanced tree (degeneration of the tree if objects are not equally distributed), i.e. neighboring sub-trees have the same (or similar) height.
      - Balancing strategies in general possible but very expensive, i.e. high cost for tree organization.
    - Bulk-Load (BSP-Tree):
      - Assumption: Static database, i.e. all objects are known.
      - The tree can be build bottom-up by recursive balanced splits until each leaf node contains at most M entries.
      - Bulk-load always yields a balanced tree structure.



# Index-based Similarity Search

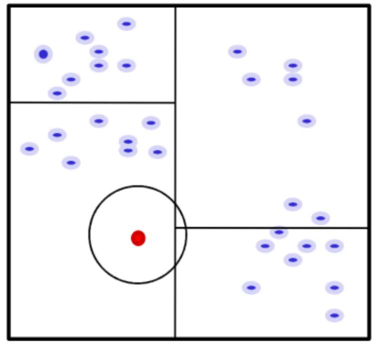
- Principle of Spatial Indexes in Databasesystems (cont.)
  - Space Partitioning Index Methods: Quad-Tree
    - Quad-Tree for 2D data:
      - Root node covers the whole object space.
      - Each inner node has 4 child nodes, decomposing the space into 4 equally sized partitions.
      - Quad-trees are usually not balanced.
      - Objects only stored at leaf nodes
      - Leaf nodes have a maximal capacity but no constraint on the minimal number of contained objects.

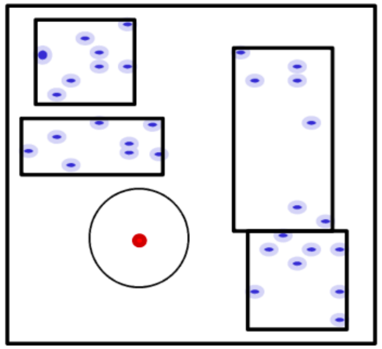




# Index-based Similarity Search

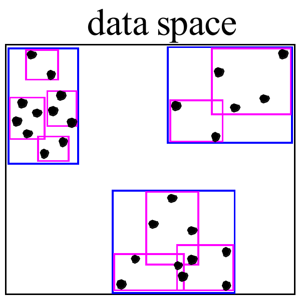
- Principle of Spatial Indexes in Databasesystems (cont.)
  - Space Partitioning Index Methods: Pros and Cons
    - (+) Allows efficient (re-)organization of the tree structure
    - (+) Page regions do not overlap => Point queries result in one search path from root to leaf node.
    - (-) Page regions could have large dead space => bad spatial query performance
  - Data Partitioning Index Methods: Pros and Cons
    - (+) Less dead space covered by spatial regions => better query performance if objects are not equally distributed.
    - (+) Full balanced tree which is appropriate for secondary storage
    - (-) Page regions may overlap => (Point query) search could lead to multiple search paths from root to leaf node.

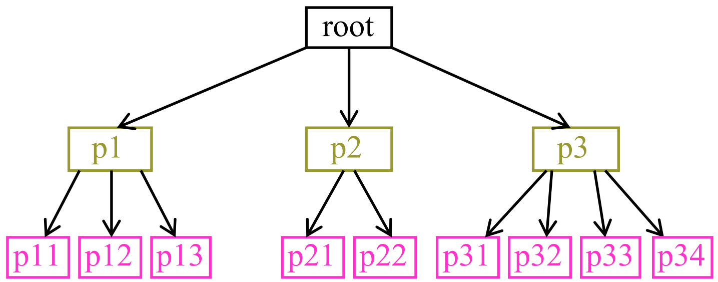




# Index-based Similarity Search

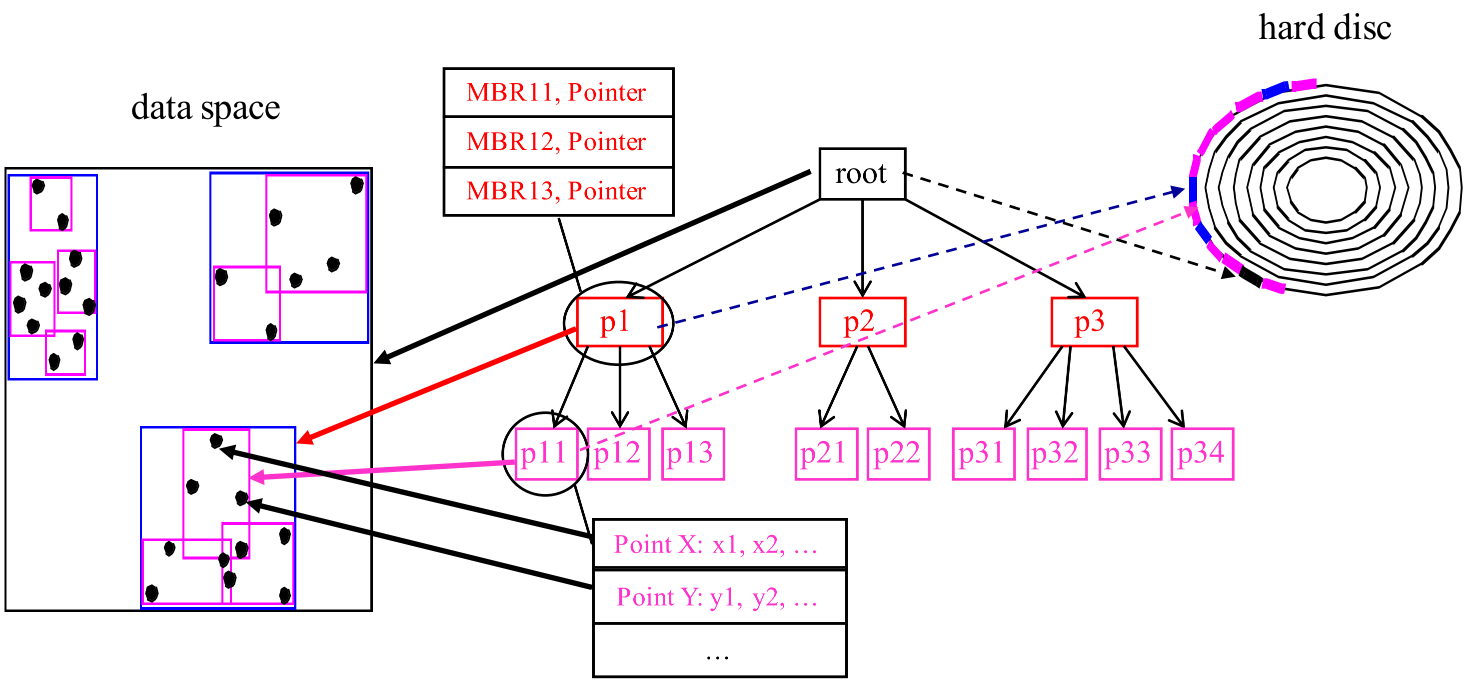
- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Spatial Index: R-tree
    - Designed for 2D rectangles but often used for multi-dimensional vector data (points are rectangles without extend)
    - Design
      - Balanced tree
      - Data pages and directory pages have the same capacity
      - Rectangular page regions (MBRs)
      - Split and insert operations try to minimize:
        - page region overlap
        - dead space



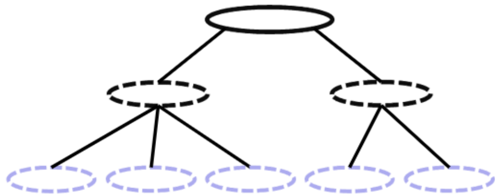
# Index-based Similarity Search

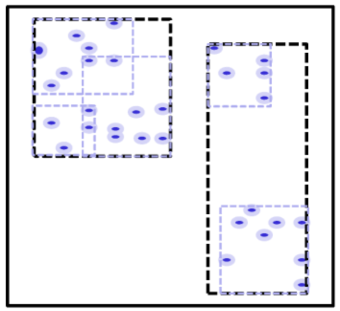
- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Spatial Index: R-tree (cont.)



# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Spatial Index: R-tree (cont.)
    - Further notes to the structure of an R-tree:
      - Root node covers entire object space and has max. M entries. (Entry := (MBR, pointer to child node))
      - Inner nodes have between m and M entries, where $m \leqq M/2$.
      - The MBR associated with a node (subtree), entirely covers all MBRs associated with nodes of the subtree.
      - All objects are stored in leaf nodes and all Leaf nodes are on the same tree level (called leaf level)
      - The R-tree can be used to manage rectangle objects or point objects





# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Spatial Index: R-tree (cont.)
    - Inserting a new entry (object) x in an R-tree:
      - Each insert procedure needs a prior search of the leaf node where the object has to be inserted, starting at the root node.
      - Because of overlapping page regions, at each node we have to consider 3 cases:
        - Case 1: x is entirely covered by exactly one page region R
          => Insert x into the subtree associated with R.
        - Case 2: x is entirely covered by multiple page regions $R_1,\ldots,R_n$
          => Instert x into the subtree of the region with the smallest volume.
        - Case 3: x is not entirely covered by any page region
          => Instert x into the subtree of the page region having the smallest increase in volume to cover x.
      - If x is inserted in a leaf node N, N could exceed his capacity (overflow) => split N into two nodes.

# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Spatial Index: R-tree (cont.)
    - Splitting a node in an R-tree:
      - If an overflow of a node N occurs, N has to be split in two nodes N1 and N2, and the entries of N has to be distributed among N1 and N2 s.t.
        - at least m entries are in each of these two nodes.
        - the overlap, volume, and dead space of the resulting MBRs associated with N1 and N2 is minimized.
      - There are a multiple split strategies proposed with the R-tree and the R*-tree (variant of R-Tree showing the best split performance).

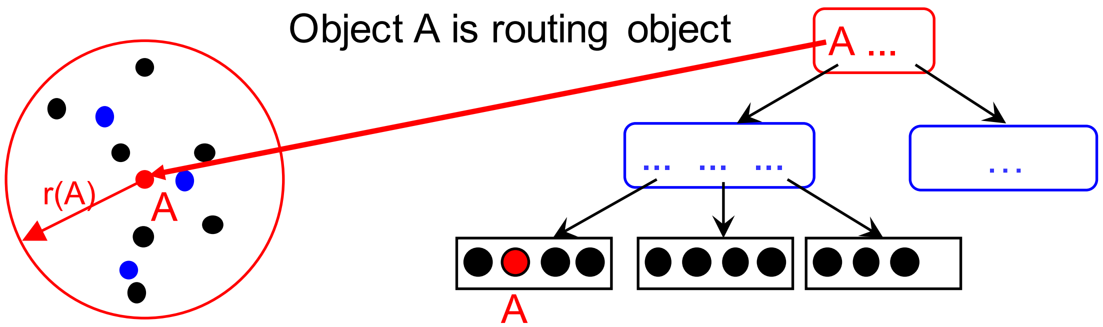
    Homework: For the different split strategies, read the article about R-trees which is available in Blackboard!!

# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Metric Index: M-tree
    - Dynamic index structure for general metric spaces.
    - The distance function for the computation of the similarity between two objects must be a metric (i.e. fulfill metric properties).
    - Design of the index structure:
      - Balanced hierarchical index structure with uniquely sized data and directory pages.
      - Separation between directory nodes and data nodes (similar to R-tree)
      - The indexed database objects (or direct links to the objects) will be stored (organized) in the leaf nodes.

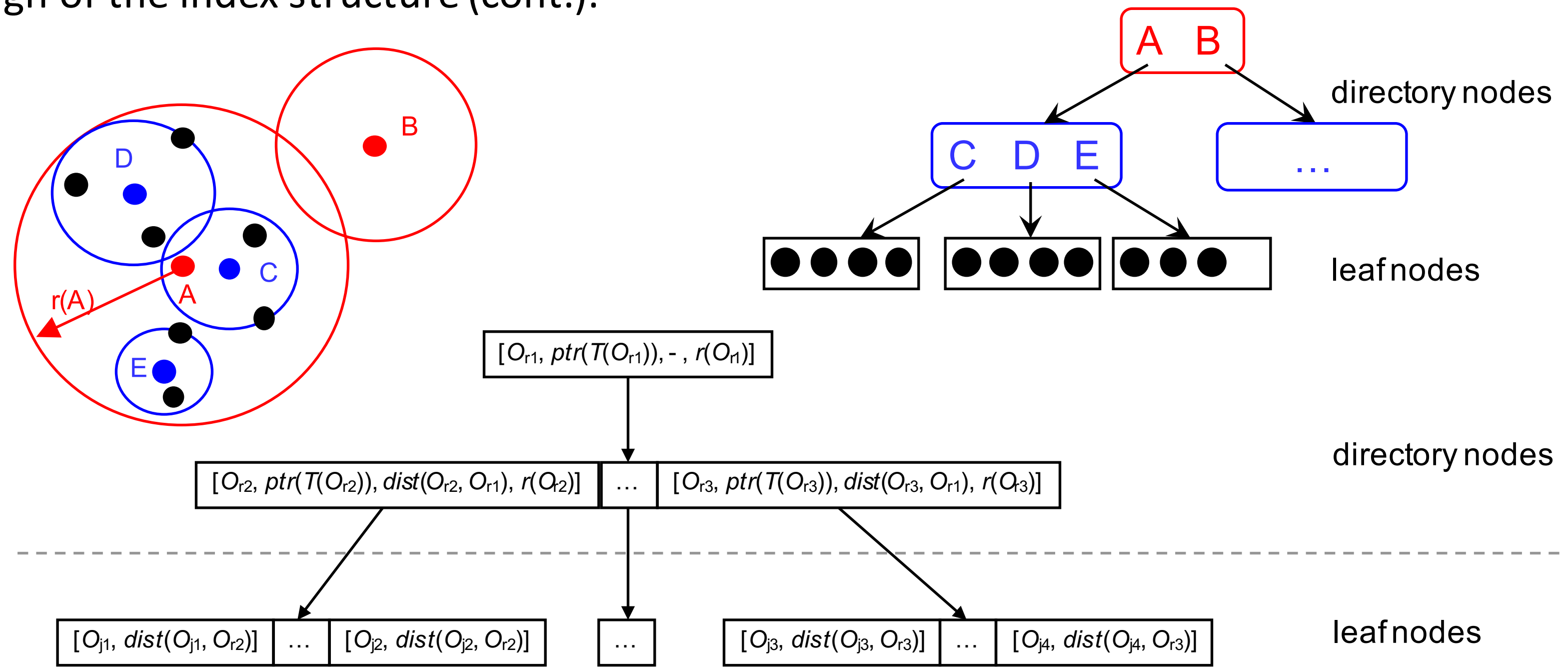
# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Metric Index: M-tree (cont.)
    - Design of the index structure (cont.):
      - Directory nodes contain *Routing Objects*.
      - Routing Objects are database objects that got the role to serve as reference for other objects.
      - A Routing Object RO contains a pointer to its subtree (in addition to its exact object description) and a distance radius r(RO), that bounds the distance between RO to all objects organized in the subtree below RO.



# Index-based Similarity Search

- Principle of Spatial Indexes in Databasesystems (cont.)
  - Example Metric Index: M-tree (cont.)
    - Design of the index structure (cont.):

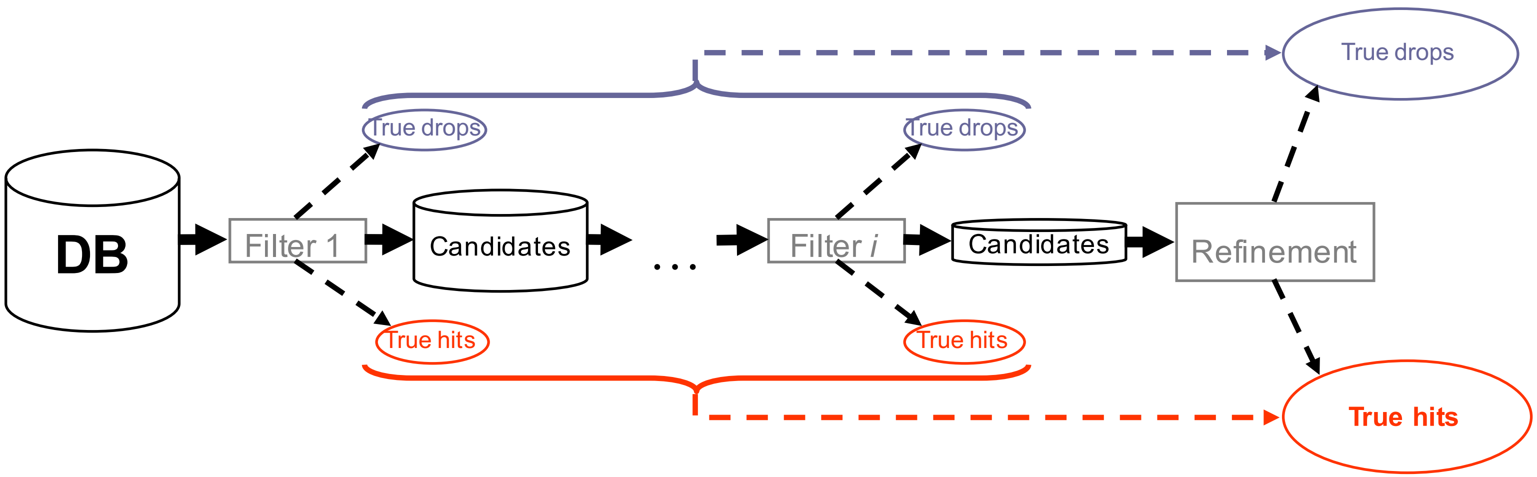


# Similarity Search based on Multi-Step Query Processing

- Principals of Multi-step query processing (a.k.a. filter/refinement query processing)
  - Motivation
    - In many applications, the evaluation of the query predicate becomes the bottleneck of query processing
      - Costly distance functions (e.g., edit-distance on sequences, area of overlap of polygons, distance computation in road network graphs)
      - High-dimensional feature vectors ("curse of dimensionality")
    - Solution:
      - Use a filter distance (usually based on low dimensional vectors) or a filter predicate which is much less costly to compute in order to filter out potential hits and drops
      - Set of remaining candidates need to be refined, i.e., the exact predicate is evaluated

# Similarity Search based on Multi-Step Query Processing

- Principals of Multi-step query processing (cont.)
  - General schema
    - One or more (cascading) filter steps successively shrink down the set of potential candidates.
    - Refinement step evaluates the exact query predicate for remaining candidates



# Similarity Search based on Multi-Step Query Processing

- Principals of Multi-step query processing (cont.)
  - Properties of the filter (sequence of filters)
    - The filter should be very selective in order to keep the set of candidates small.
    - In case of consecutive filters,
      - the selectivity of the filters should increase.
      - first filters should be cheaper (in terms of the cost for the query predicate evaluation) than the remaining filters.
    - The filters should work in a conservative way, i.e. the filter result (candidates) reported by the filter is a superset of the exact result, i.e. all query results appear either as true hits or candidates. => no false drops.

# Similarity Search based on Multi-Step Query Processing

- Principals of Multi-step query processing (cont.)
  - Relationship between filter step(s) and refinement step
    - To avoid false dismissals/drops
      - the filter should be conservative, i.e., only true drops are eliminated

        or
      - the filter step should be progressive, i.e., only true hits are identified
    - Lower bounding filter FLB

$$\forall x, y \in DB : dist_{F_{LB}}(F_{LB}(x), F_{LB}(y)) \leq dist(x, y)$$

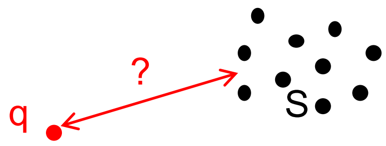
      - yields a conservative approximation (superset) of the result set
      - can be used to identify true drops.

# Similarity Search based on Multi-Step Query Processing

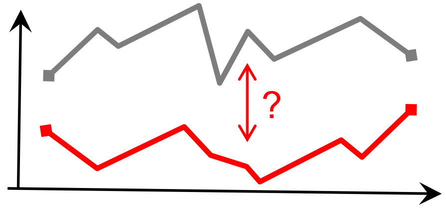
- Principals of Multi-step query processing (cont.)
  - Relationship between filter step(s) and refinement step (cont.)
    - Upper bounding filter $F_{UB}$

$$\forall x, y \in DB : dist_{F_{UB}}(F_{UB}(x), F_{UB}(y)) \geq dist(x, y)$$

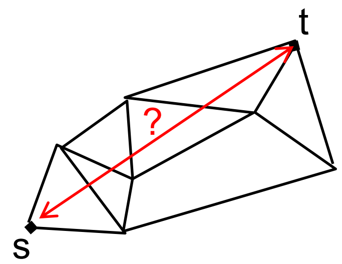
    - yields a progressive approximation (subset) of the result set
    - can be used to identify true hits
    - Examples: Appropriate lower/upper bounding filter distances?




point ←→ point sets




Time series




points in traffic networks

## Similarity Search based on Multi-Step Query Processing

- In the following, we will review these different algorithmic approaches, i.e.,
  - naïve search (sequential scan)
  - index-based search
  - multi-step query processing
- for different query types including
  - distance range queries
  - nearest neighbor queries
  - …
- If not explicitly specified, we assume Euclidean distance between objects in feature space.