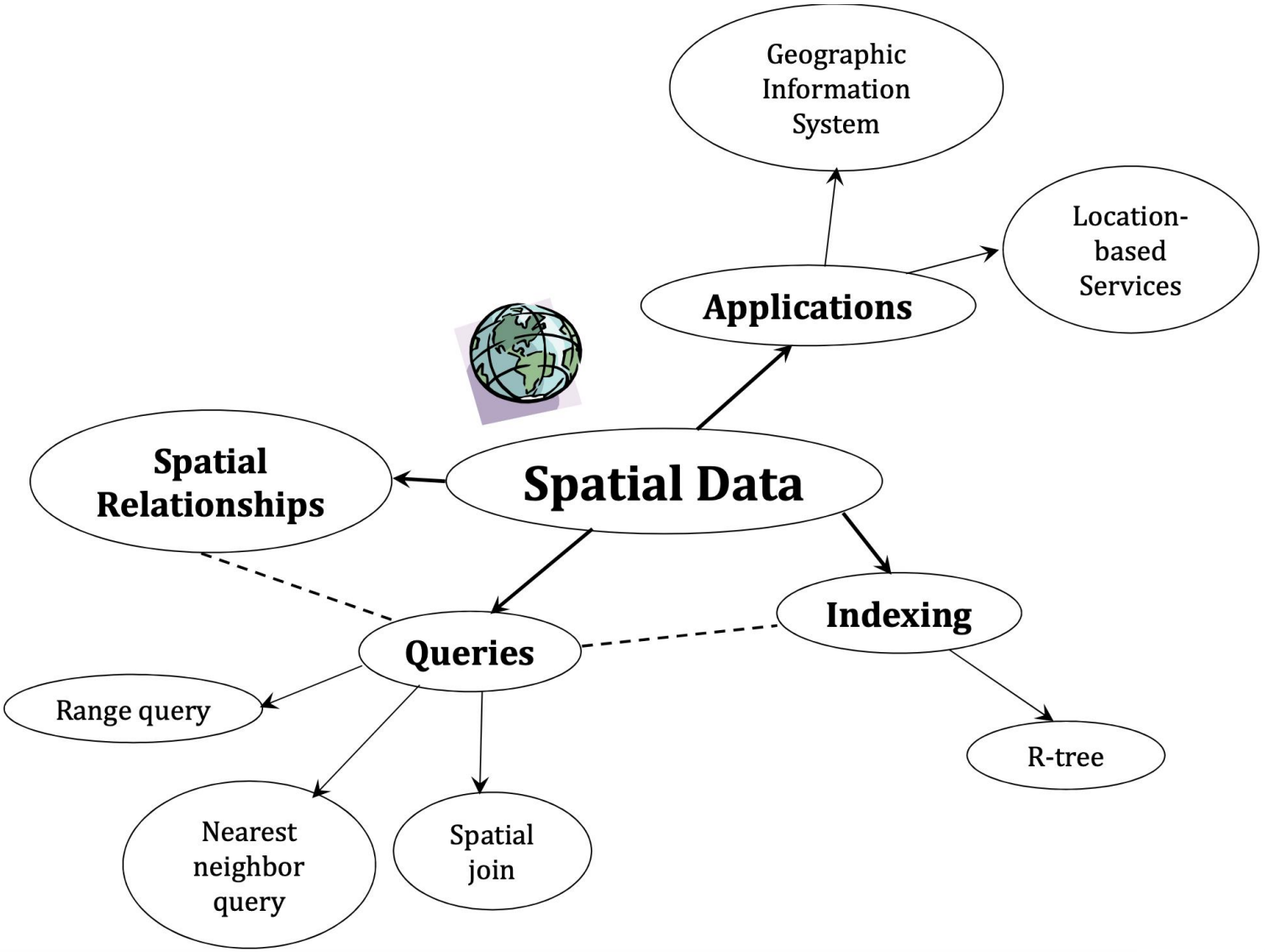
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DBGroup

Spatial Databases I

南方科技大学
唐　博
tangb3@sustech.edu.cn

The Concepts



Spatial Data Management

- Spatial Data, Relationships, and Queries
- Challenges of Indexing Spatial Data
- Filter-Refinement Processing
- The R-tree

Spatial Data Management

- A ***spatial object*** has some spatial attribute that describes its location and geometry
  - Typically 2-dimensional or 3-dimensional data
- A ***spatial relation*** (dataset) is a collection of spatial objects of the same entity (e.g., rivers, cities, roads)
- What applications use spatial data?
  - Geographic information system (GIS), location-based
  - services (LBS), computer-aided design (CAD)

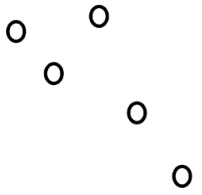
Road segments of a region

ID	Name	Type	Polyline
1	Boulevard	avenue	(10023,1094), (9034,1567), (9020,1610)
2	Leeds	highway	(4240,5910), (4129,6012), (3813,6129), (3602,6129)
…	…	…	…

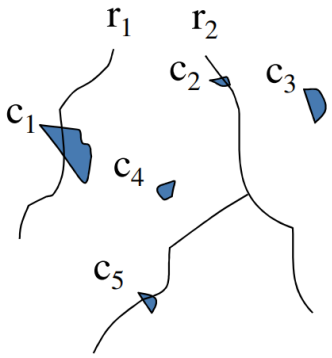
A spatial relation

Types of Spatial Data

- Points
  - Cities in a large-scale map



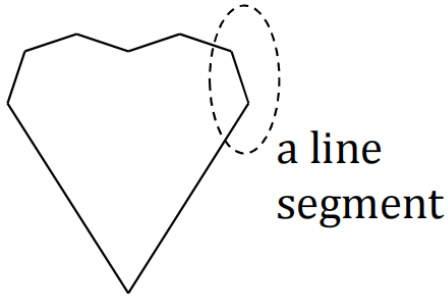
- Objects with extent
  - include rivers, forests, districts, buildings in a city, etc



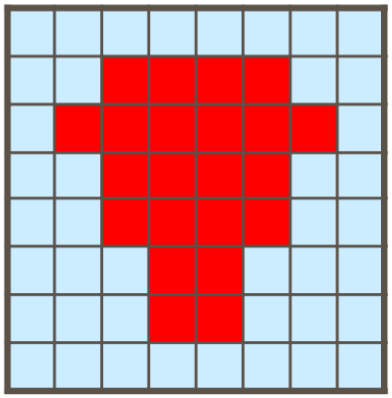
Spatial Data with Extent

- **Vector representation**: approximation by geometric objects, e.g., polygons, poly-lines, rectangles, etc.
- **Raster representation**: divide the space by a grid and approximate the objects by a set of pixels in this grid

actual object



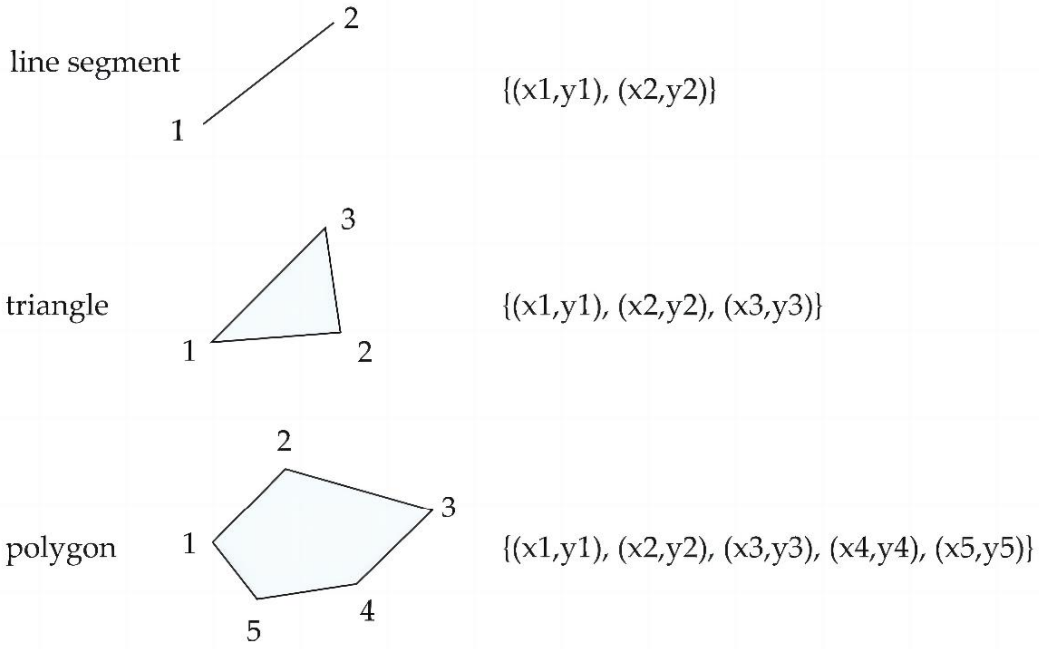
vector approximation



raster approximation

Vector Representation

Line Segments and Polygons



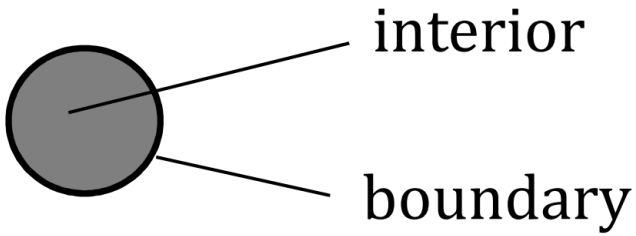
Spatial Relationships

- A **spatial relationship** links two objects based on their relative location and extent in space
  - Example: "My house is close to Central Park"
  - Different from the term "spatial relation", which means a dataset of spatial objects

- Types of spatial relationships
  - topological relationships
  - distance relationships
  - directional relationships

Topological Relationships

- Each object is characterized by the space it occupies
  - a set of pixels (finite or infinite)
- Each object has a boundary and an interior
  - boundary: the set of object pixels that are adjacent to at least one pixel not occupied by the object
  - interior: the set of object pixels which are not boundary
- Some objects may not have interior
  - e.g., points, line segments



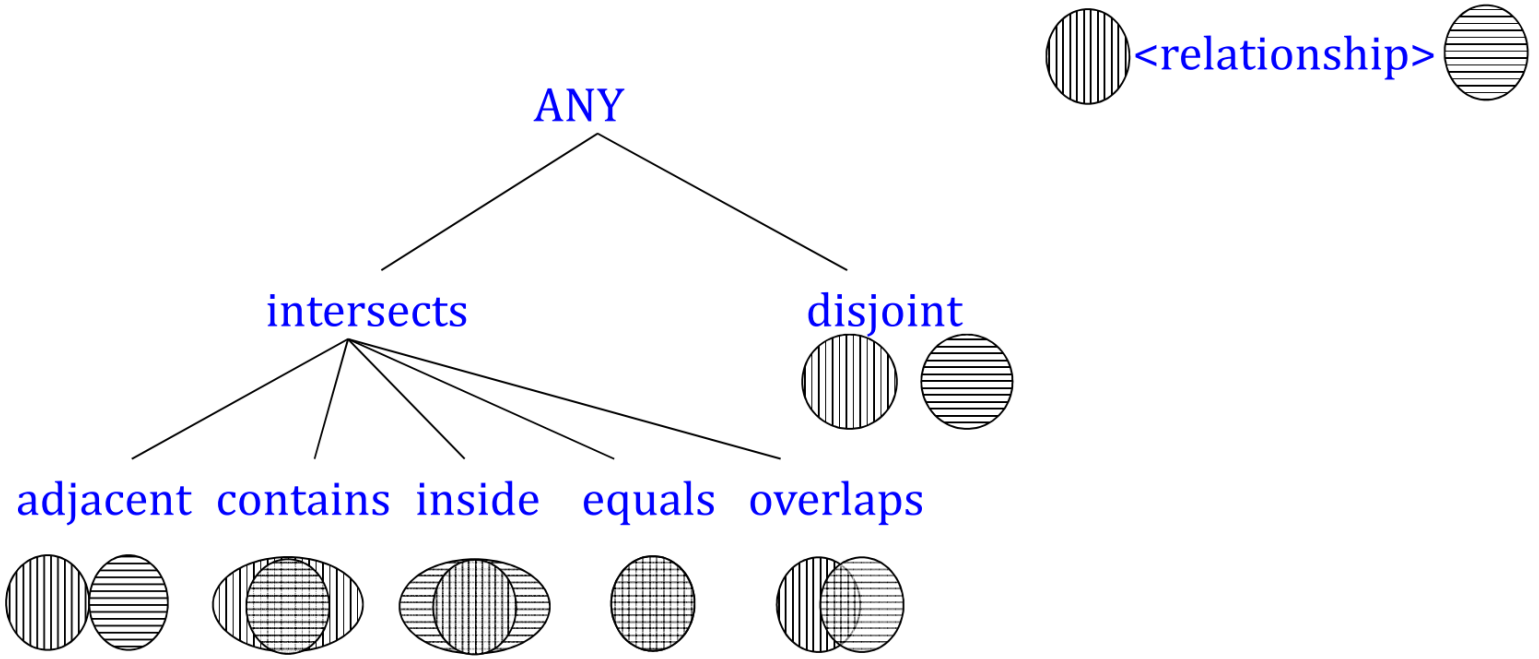
Topological Relationships

- A topological relationship between two objects is defined by a set-based relationship between their boundaries and interiors
  - E.g., o_1 is inside o_2 if $interior(o_1) \subset interior(o_2)$
- intersects includes: equals, inside, contains, adjacent, overlaps
- intersects $\Leftrightarrow \neg$disjoint

Topological relationship	equivalent boundary/interior relationships
$disjoint(o_1, o_2)$	$(interior(o_1) \cap interior(o_2) = \emptyset) \wedge (boundary(o_1) \cap boundary(o_2) = \emptyset)$
$intersects(o_1, o_2)$	$(interior(o_1) \cap interior(o_2) \neq \emptyset) \vee (boundary(o_1) \cap boundary(o_2) \neq \emptyset)$
$equals(o_1, o_2)$	$(interior(o_1) = interior(o_2)) \wedge (boundary(o_1) = boundary(o_2))$
$inside(o_1, o_2)$	$interior(o_1) \subset interior(o_2)$
$contains(o_1, o_2)$	$interior(o_2) \subset interior(o_1)$
$adjacent(o_1, o_2)$ (or $meets(o_1, o_2)$)	$(interior(o_1) \cap interior(o_2) = \emptyset) \wedge (boundary(o_1) \cap boundary(o_2) \neq \emptyset)$
$overlaps(o_1, o_2)$	$(interior(o_1) \cap interior(o_2) \neq \emptyset) \wedge (\exists p \in o_1 : p \notin interior(o_2) \wedge p \notin boundary(o_2))$ $\wedge (\exists p \in o_2 : p \notin interior(o_1) \wedge p \notin boundary(o_1))$

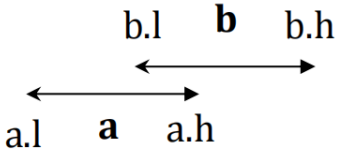
Topological Relationships

Hierarchy of topological relationships

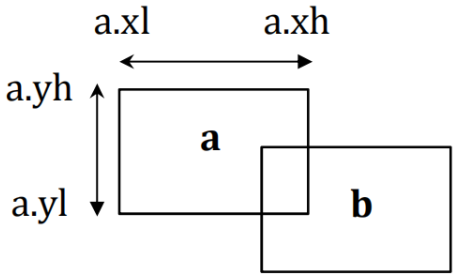


Intersection of Rectangles

- **intersects:** frequently-used topological relationship
- **rectangles:** used to represent objects/queries
- *Question*: how to check the "intersects" predicate without enumerating pixels?



Intersection condition for 1-dimensional intervals:
(b.l ≤ a.h) *and* (a.l ≤ b.h)

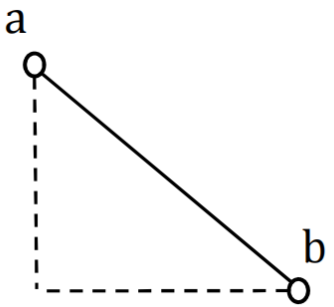


Intersection condition for 2-dimensional rectangles:
(b.xl ≤ a.xh) *and* (a.xl ≤ b.xh) *and*
(b.yl ≤ a.yh) *and* (a.yl ≤ b.yh)

Distance Relationships

- Distance relationships associate two objects based on how far they are

- We focus on the Euclidean distance
  - $dist(a,b) = \sqrt{(a.x - b.x)^2 - (a.y - b.y)^2}$

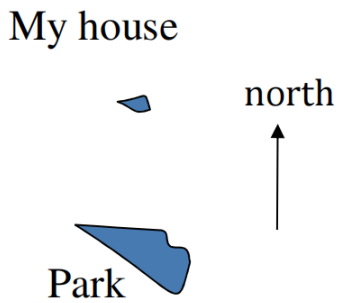


- Another distance measure: the shortest path distance on a road network

Directional Relationships

- Directional relationships associate two objects based on their relative **orientation** according to a global reference system
  - My house is north of the park

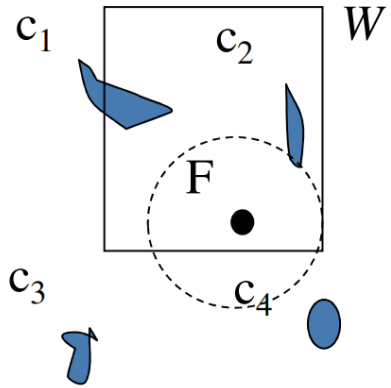
- Examples of directional relationships:
  - north, south, east, west, etc.
  - left, right, above, below, front, behind, etc.

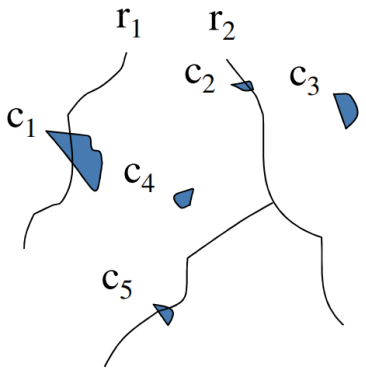


- Combinations of topological, distance, and directional relationships can be used to describe a pair of spatial objects:
  - My house is **disjoint** with the park, **100 meters north of** it

Spatial Queries

- Input: spatial relation(s)
- Output: objects (or combinations of objects) satisfying some spatial relationships
  - with a reference query object, or
  - between two spatial datasets
- **Range query** (spatial selection, window query)
  - e.g., find all cities that intersect window W
  - *Answer set*: {c1, c2}
- **Nearest neighbor query**
  - e.g., find the city closest to the forest F
  - *Answer*: c2
- **Spatial join**
  - e.g., find all pairs of cities and rivers that intersect
  - *Answer set*: {(r1,c1), (r2,c2), (r2,c5)}





History

- Spatial index developed for range query
  - 1974-1975: point access method (quad-tree, kd-tree)
  - 1981: Space ordering method (B+-tree over the Z-order)
  - 1984: R-tree
  - 1990: R*-tree

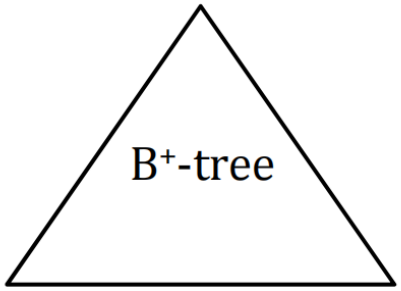
- Query types on R-trees
  - 1984: Range query
  - 1993: Spatial join
  - 1995: Nearest neighbor query
  - 1998: Closest pair

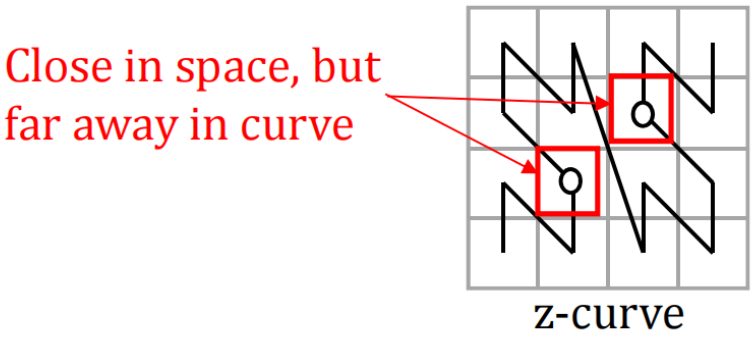
History

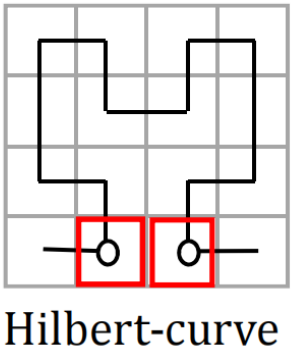
Commercial products

- Oracle Spatial (and Graph)
  - 1995: Point data only
  - 1996-2000: Lines, polygons, ……
  - 2002: Quadtree, R-tree index
  - https://docs.oracle.com/cd/B10500_01/appdev.920/a96630/sdo_index_query.htm
- SQL Server, Spatial Support
  - 2008: spatial data types (for flat/round Earth models)
  - 2012: geometry hierarchy, complicated curves
  - Index: B+-tree over grid index
  - https://msdn.microsoft.com/en-us/library/bb895265.aspx

Space Ordering Method





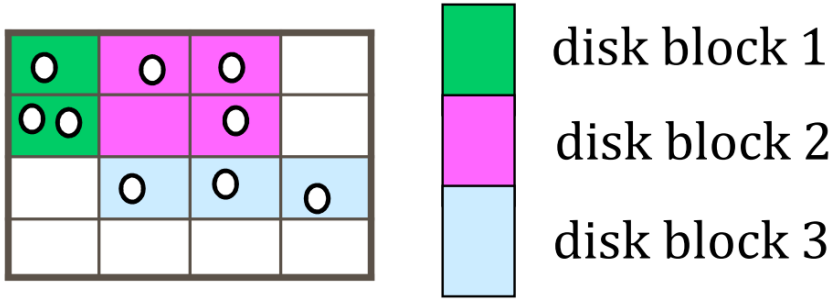


- Space ordering method
  - Use a **space filling curve** to convert each point to a value
  - Index the values of these points by a B+-tree
  - Range query: Decompose the query into multiple interval queries
- Problems
  - Dimensionality: No space ordering preserves spatial proximity
  - Complex spatial extent: Redundancy for assigning objects into disk pages

Point Access Methods

- Point access methods divide the space into disjoint regions and group the points by their regions
  - Grid-file
  - kd-tree
  - Quad-tree

Example of a grid-file:

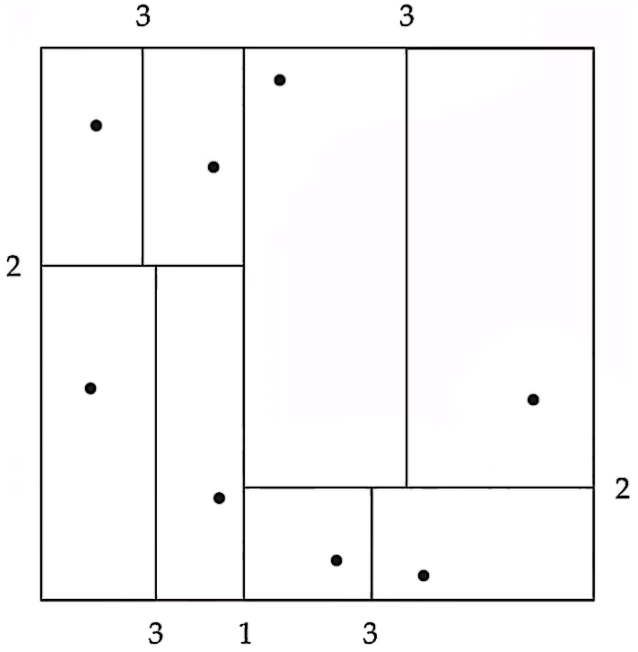


Point Access Method: k-d Tree

k-d tree

- Each tree level partitions the space into two regions with
- (almost) equal number of points
  - choose one dimension for partitioning at the root level
  - choose another dimension for partitioning at the next level and so on
- Partitioning stops when a node has less than a fixed maximum number of points

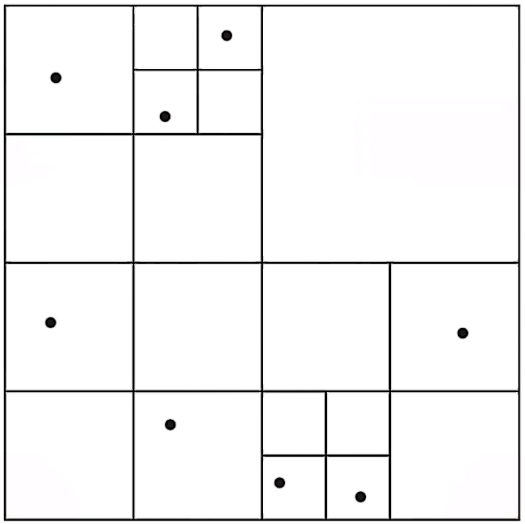
- Example figure of k-d tree
  - Each line (except the outside box) corresponds to a tree node
    - Each leaf node contains 1 point
  - Tree level: the numbering of the lines



Point Access Method: Quad-tree

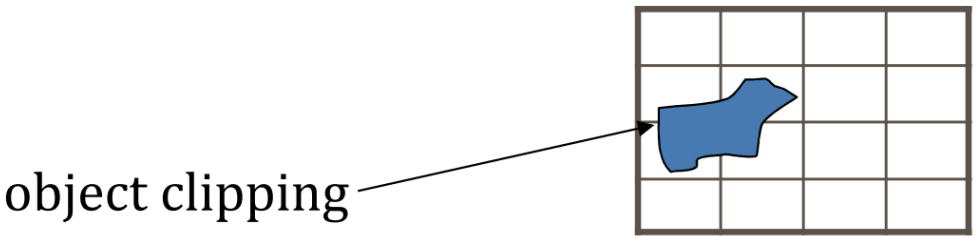
Quadtrees

- The top node is associated with the entire space
- Each non-leaf node divides its region into four rectangles with equal area
  - Each such node has four corresponding child
- Partitioning stops when a node has less than a fixed maximum number of points



Drawbacks of Point Access Methods

- Extensions of *k-d* trees and quadtrees can be used to index line segments and polygons
  - Require splitting segments/polygons into pieces at partitioning boundaries
  - The same segment/polygon may be stored at several leaf nodes
- Drawback: point access methods are not effective for extended objects
  - Objects may be *clipped* into several parts ➔ data redundancy and affects performance badly



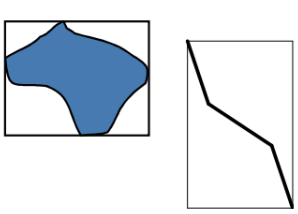
Spatial Access Methods

- **Large** spatial dataset, stored on disk
  - Query processing: I/O time is much larger than CPU time
- Spatial Access Methods (SAM): Indexing spatial data
  - aim at minimizing the expected I/O cost
- R-tree is the most popular spatial access method
  - Use the filter-refinement approach for processing queries

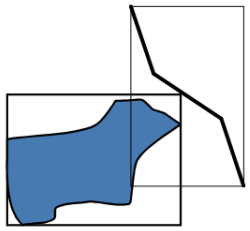
Filter-refinement Query Processing

- It is slow to evaluate spatial relationships on geometric data; thus, we approximate a *spatial object* by its **minimum bounding rectangle** (MBR)
- The spatial query is processed in two steps:
  1. Filter step: test the MBR against the query predicate → Fast!
  2. Refinement step: [if passed the filter step] then test the exact geometry of objects → Slow!

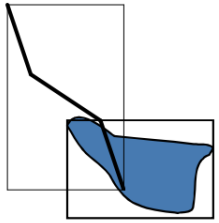
Example for spatial intersection joins:



filtered pair



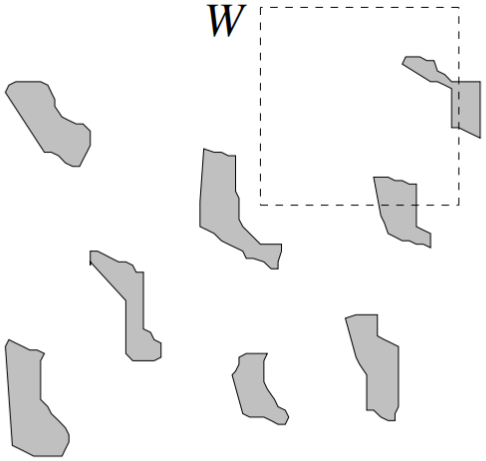
non-qualifying pair that passes the filter step (false hit)



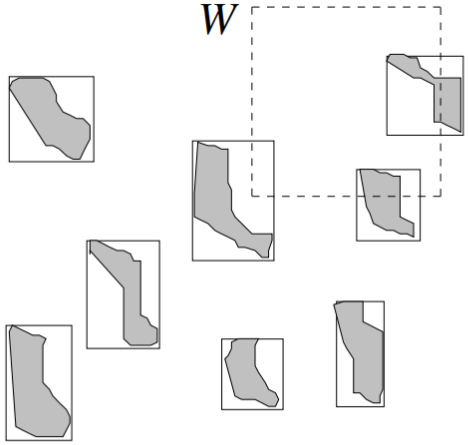
qualifying pair

Filter-refinement Query Processing

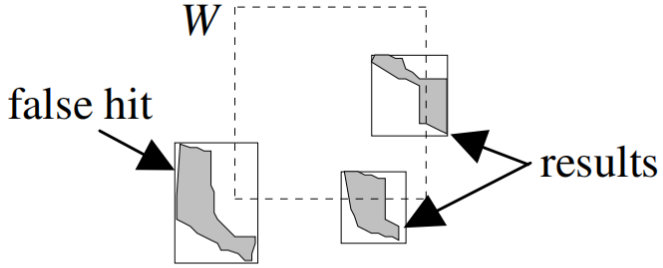
- Problem: Find all objects that intersect the query window W
- Comparison cost ≈ number of examined vertices
- Suppose that each polygon has 100 vertices
  - Comparisons in Fig. (a) = (100+4)*8 = 832
  - Comparisons in Fig. (b) = (4+4)*8 + (100+4)*3 = 376



(a) objects and a query



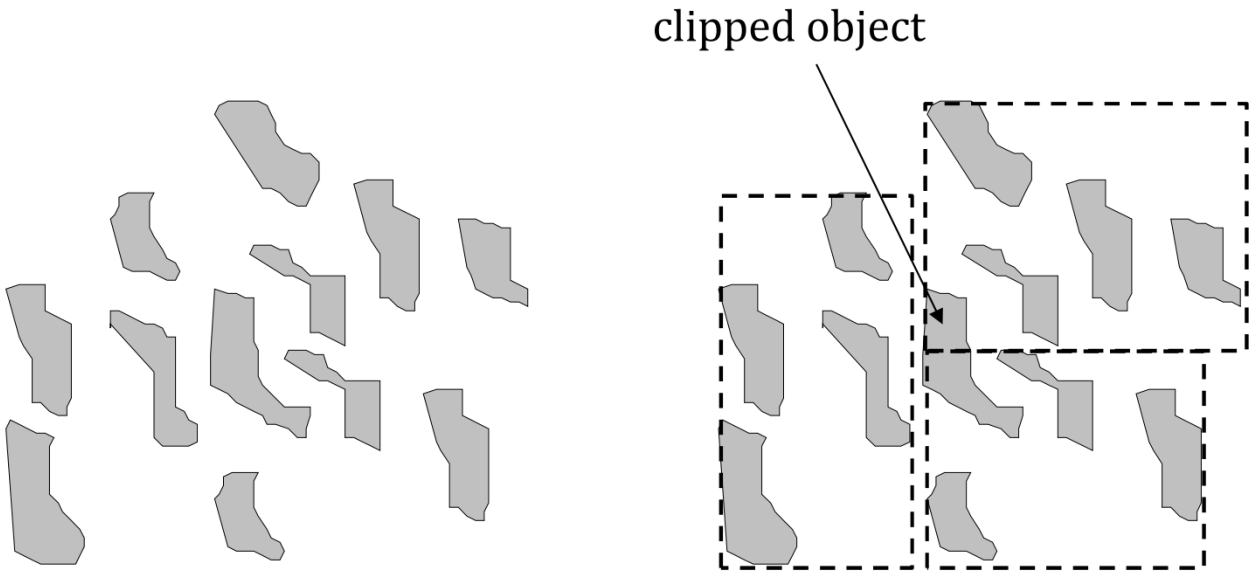
(b) object MBRs



(c) candidates and results

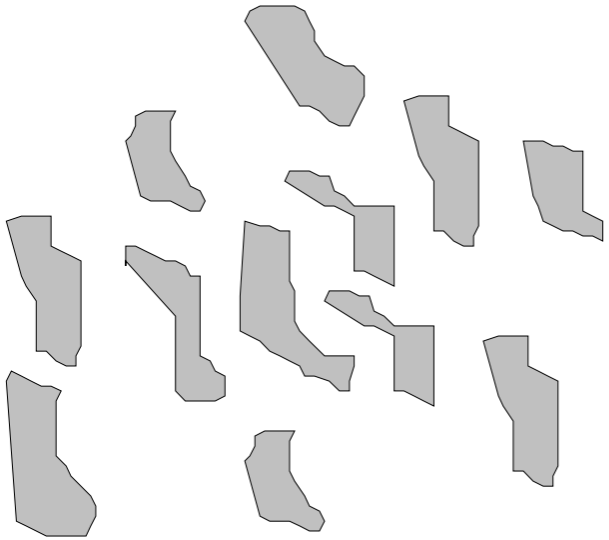
Spatial Access Methods

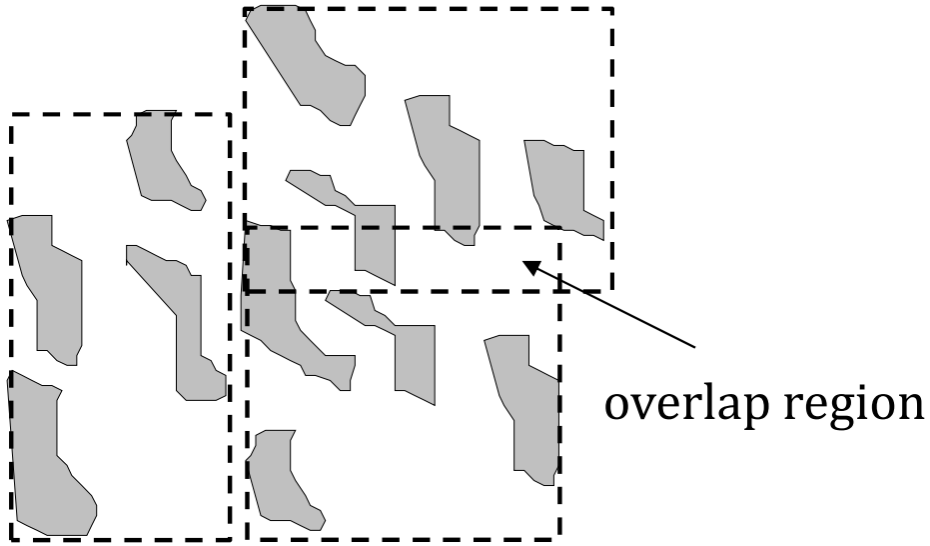
- Problem:
  - Group the objects below into 3 groups of 4 objects each such that the MBRs of the groups do not overlap
  - not feasible in this example
- Avoid object clipping by allowing regions of object groups to overlap



Spatial Access Methods

- Revised problem:
  - Group the objects below into 3 groups of 4 objects each such that the MBRs of the groups have the **minimum overlap**
  - Hard optimization problem
  - Use a fast, suboptimal solution instead (discussed later)





Spatial Indexing Methods

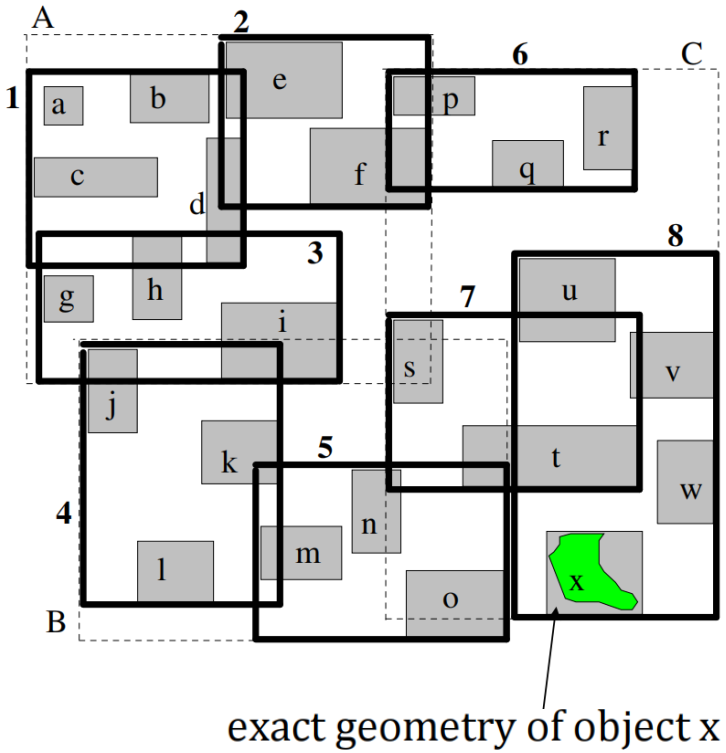
Indexing method	Space ordering method (B^+-tree over space filling curve)	Point access methods (k-D tree, quad tree)	**R-tree**
Point object	yes	yes	yes
Object with extent	no	Use clipping (but it leads to redundant data and high query cost)	yes
Point query	yes	yes	yes
Range query	High cost	reasonable	Low cost

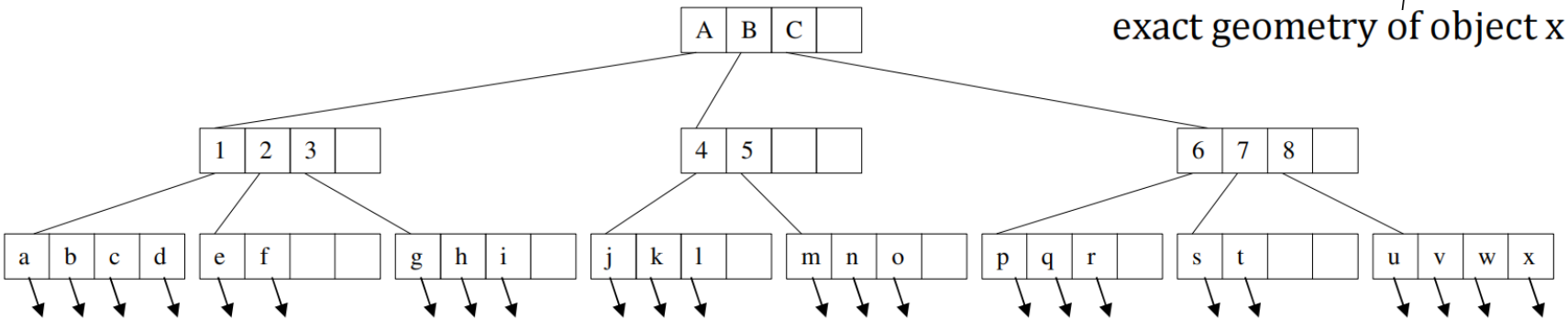
The R-tree

- Groups object MBRs to disk blocks
  - Each group of objects (a disk block) is a leaf of the tree
  - Objects in the same leaf node should be close together
- The MBRs of the leaf nodes are grouped to form nodes at the next level
  - MBRs located close together are grouped
- Grouping is recursively applied at each level until a single group (the root) is formed

The R-tree

- Leaf node entries
  - <MBR, object-id>
- Non-leaf node entries
  - <MBR, ptr>
  - Its MBR = MBR of all entries in its child node
- Properties like B+-tree
  - balanced tree
  - 1 node → 1 disk block
  - Minimum fanout: 0.4 of capacity

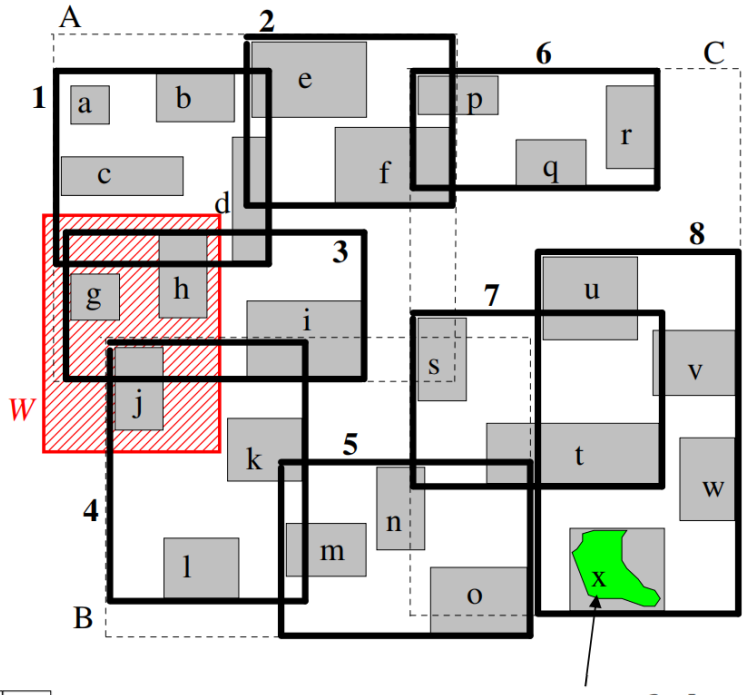


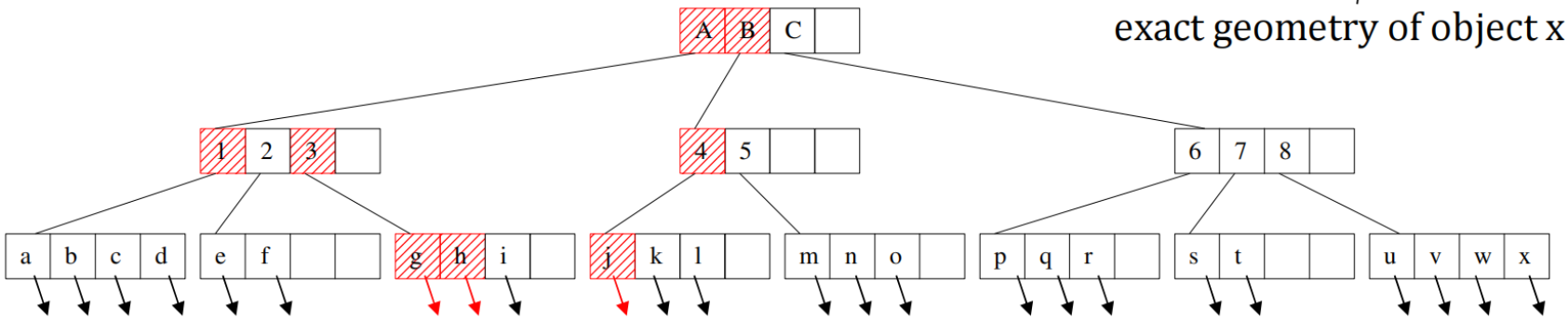


30

Searching

- W is a range query (window intersection query)
- Which nodes are visited during search?





Range searching on an R-tree

- **Range_Query**(query *W*, R-tree node *n*):

  If *n* is not a leaf node

    For each index entry e in n such that e.MBR intersects *W*

      visit node *n'* pointed by *e.ptr* ;

      Range_Query(*W, n'*) ;

  If n is a leaf node

    For each index entry *e* in *n* such that e.MBR intersects *W*

      visit object o pointed by *e.object-id*;

      test range query against exact geometry of o;

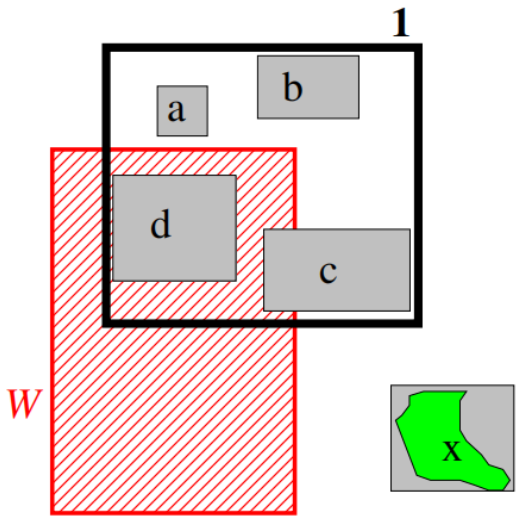
      if o intersects *W*, then report o;

- The search starts from the root node
- May follow *multiple paths* during search

Range search with other predicates

- The search algorithm considers only the intersection predicate
  - I.e., find all objects intersecting the query window W

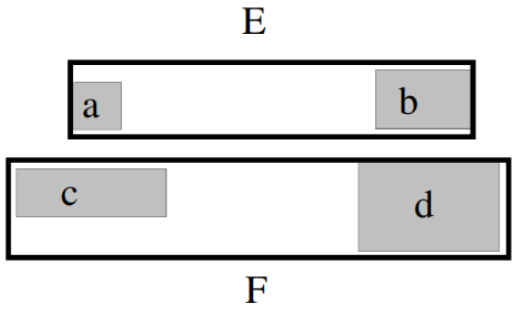
- How to modify the algorithm for other search predicates?
- E.g., find all objects that are **contained** in W
  - Search predicate for an object o
    - contains (W, o)
  - Search predicate for an object's MBR R_o
    - contains (W, R_o). Why?
  - Search predicate for a non-leaf MBR R_e
    - intersects (W, R_e). Why?

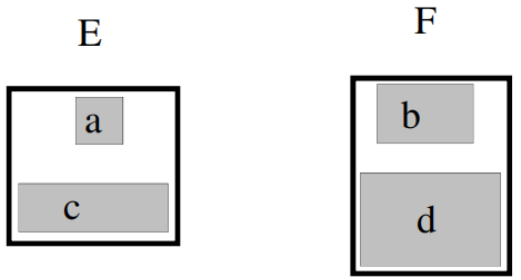


Construction of the R-tree

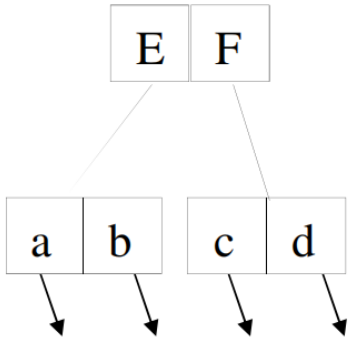
- Insertion similar to B+-tree
  - In addition, special optimization algorithms are designed for
    - choosing the path where a new MBR is inserted
    - splitting overflown nodes
- Underflows in deletions are handled by
  - deleting the underflown leaf node
  - re-inserting the remaining entries

Which R-tree is better? Why?

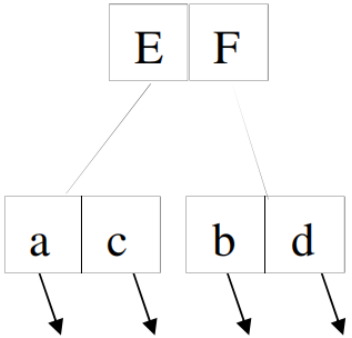




Tree #1



Tree #2



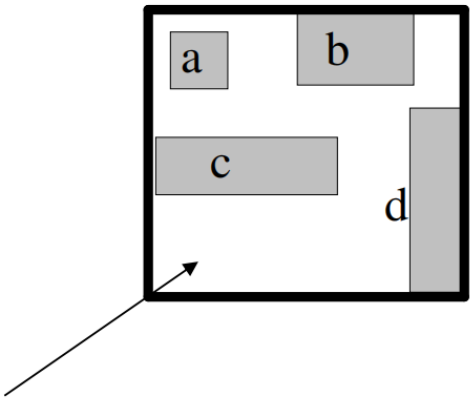
R*-tree

An optimized version of R-tree

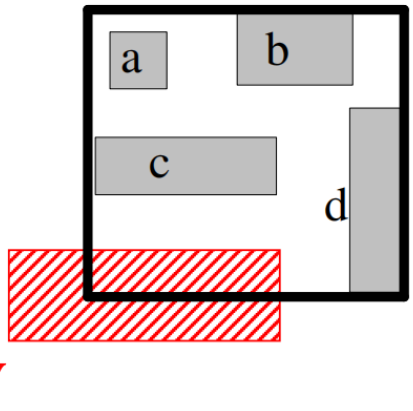
- The R-tree and the R*-tree differ only in the insertion algorithm
- The improved insertion algorithm aims at constructing a tree of high quality
- A "good" tree should have
  - nodes with small MBRs
  - nodes with small overlap
  - nodes that look like squares
  - nodes as full as possible

Optimization Criteria: Small Area

- Minimize the area covered by an index rectangle
  - small area means small dead space



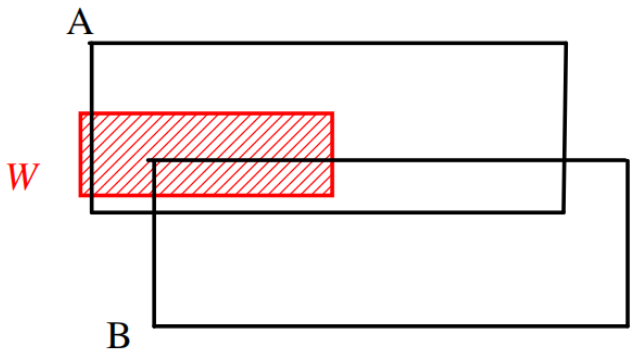
dead space



window query intersects node MBR, but no object!

Optimization Criteria: Small Overlap

- Minimize overlap between node MBRs
  - Minimizes the number of traversed paths

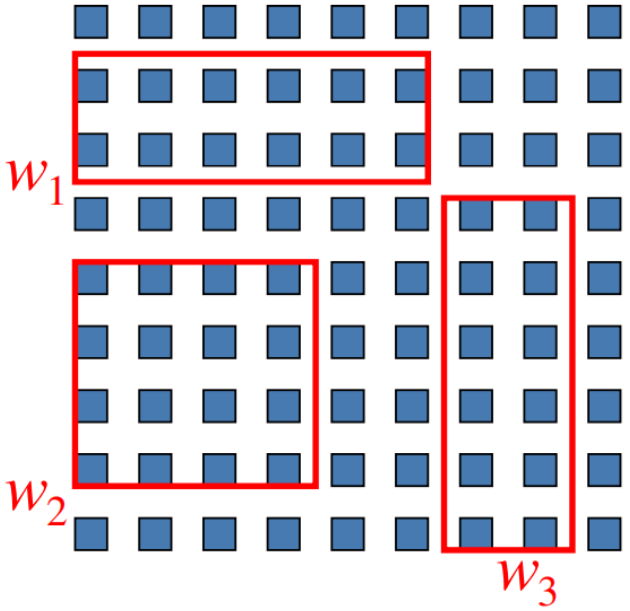


Both nodes intersect query!

Optimization Criteria: Small Margin

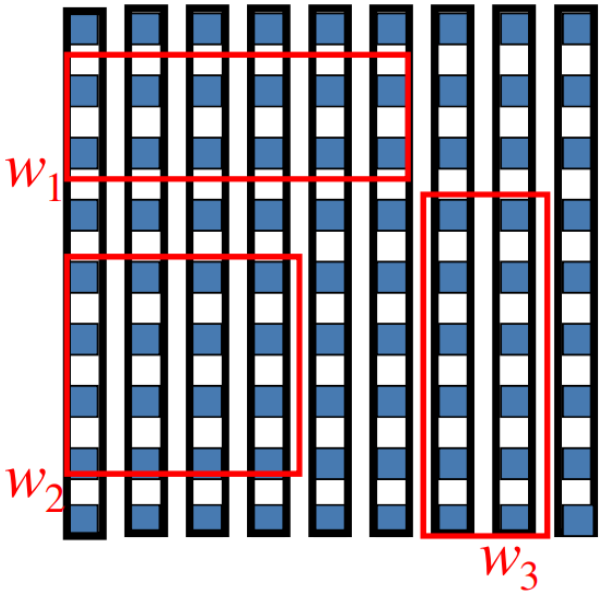
- Minimize the margins of node MBRs
  - Square-like nodes, smaller number of intersections for a random query, better structure

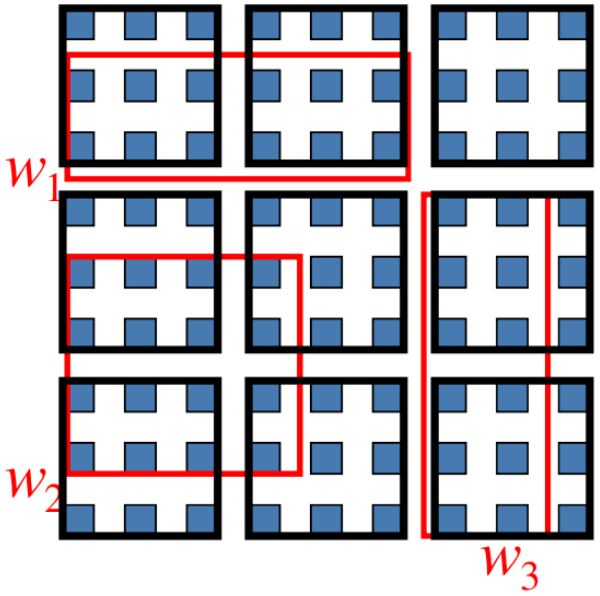
- Example: assign the rectangles (objects) into nodes (groups) of 9 rectangles,
- …… in a way such that the expected number of group MBRs intersected by a (random) query is minimized



Effect of Margins Minimization

- Grouping 1 (minimal areas of group MBRs)
  - bad grouping for queries w_1, w_2
- Grouping 2 (minimization of margins)
  - minimizes the expected number of groups touching a random query





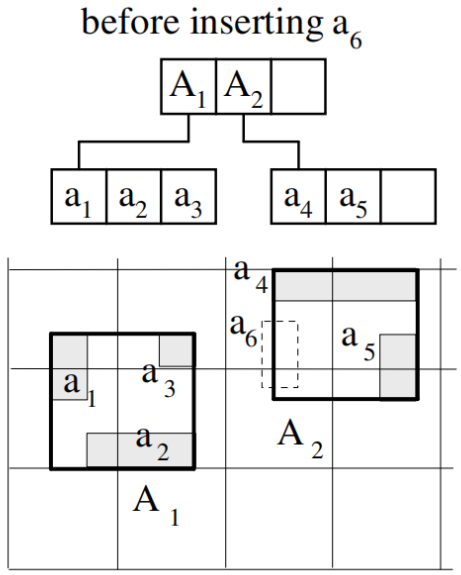
Optimization Criteria

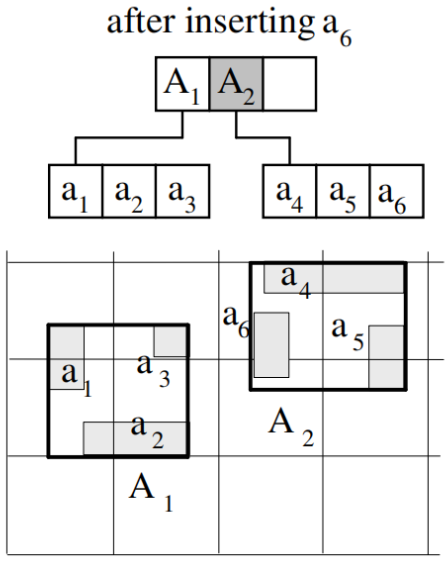
High Storage Utilization

- Optimize the storage utilization
  - Nodes in tree should be filled as much as possible
  - Minimizes tree height and potentially decreases dead space
  - This criteria may be in conflict with the first optimization criteria

Insertions in an R-tree

- What happens during an insertion?
  - Find an appropriate leaf node to store the new entry <MBR,object-id>
  - For all nodes along the path from the root to the leaf node, adjust (if necessary) their MBRs to fit the new MBR



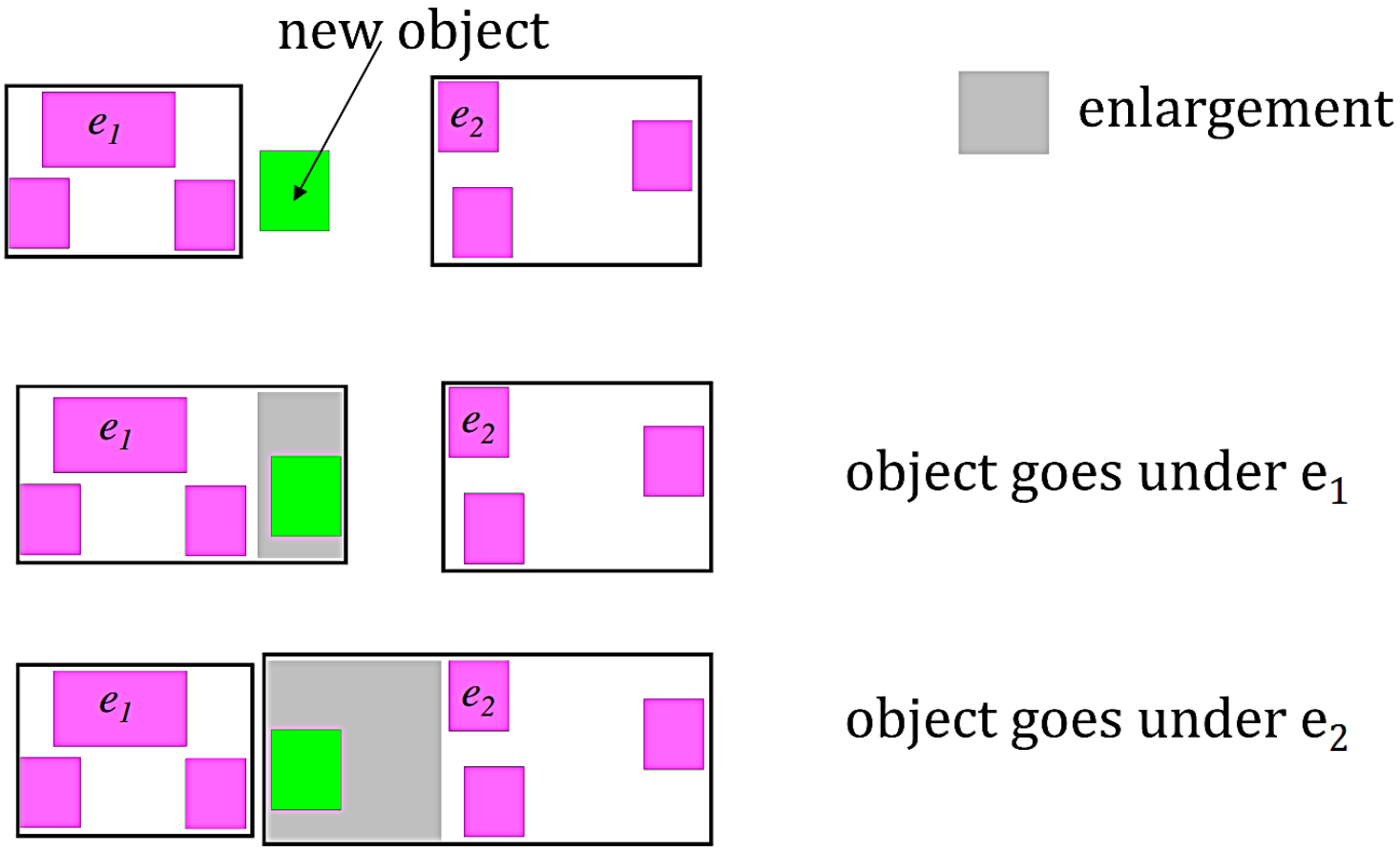


Insertion heuristics

- When inserting a new entry e into the tree, we follow one path from the root to a leaf
  - The entry is then inserted to the leaf
- Issue: How to choose the path?
- Let *N* be the current node, for inserting the new entry *e*:
  - Follow subtree pointed by entry *e'* of *N*:
    - whose MBR is **enlarged** the least after insertion
    - whose MBR enlargement will cause the **minimum overlap** with other entries of the same node
      - break any ties by choosing MBR with the **minimum area**

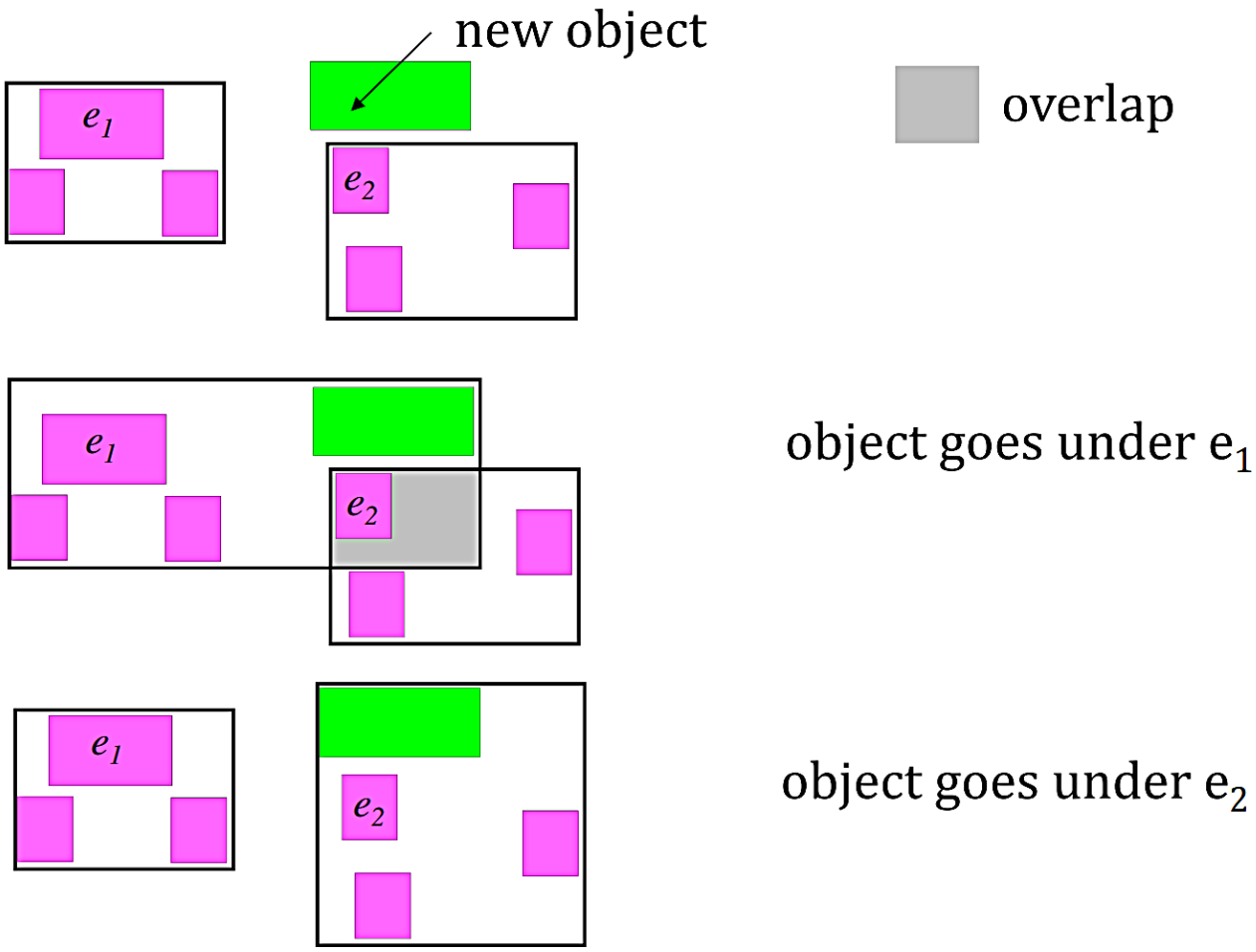
Insertion heuristics

MBR Enlargement



Insertion heuristics

MBR Overlap



Summary

- Spatial Data, Relationships, and Queries
  - Topological, distance, directional
- Challenges of Indexing Spatial Data
- Filter-Refinement Processing
- The R-tree
  - Tree structure
  - Searching, insertion

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DBGroup

谢谢！

DBGroup @ SUSTech

Dr. Bo Tang (唐博)

tangb3@sustech.edu.cn