

# Database Management Systems

Winter 2004

## CMPUT 391: Spatial Data Management

Dr. Osmar R. Zaiane



University of Alberta

## Objectives of Lecture 10

### Spatial Data Management

- Discuss limitations of the relational data model and briefly introduce the Extended-Relational Model.
- This lecture will give you a basic understanding of spatial data management
  - What is special about spatial data
  - What are spatial queries
  - How do typical spatial index structures work

## Spatial Data Management



- Shortcomings of Relational Databases
- Modeling Spatial Data
- Spatial Queries
- Space-Filling Curves + B-Trees
- R-trees

## The Need for a DBMS

- On one hand we have a tremendous increase in the amount of data applications have to handle, on the other hand we want a reduced application development time.
  - Object-Oriented programming
  - DBMS features: query capability with optimization, concurrency control, recovery, indexing, etc.
- Can we merge these two to get an object database management system since data is getting more complex?

## Manipulating New Kinds of Data

- A television channel needs to store video sequences, radio interviews, multimedia documents, geographical information, etc., and retrieve them efficiently.
- A movie producing company needs to store movies, frame sequences, data about actors and theaters, etc. (textbook example)
- A biological lab needs to store complex data about molecules, chromosomes, etc, and retrieve parts of data as well as complete data.
- Think about NHL data and commercial needs.

## What are the Needs?

- Images
- Video
- Multimedia in general
- Spatial data (GIS)
- Biological data
- CAD data
- Virtual Worlds
- Games
- List of lists
- User defined data types



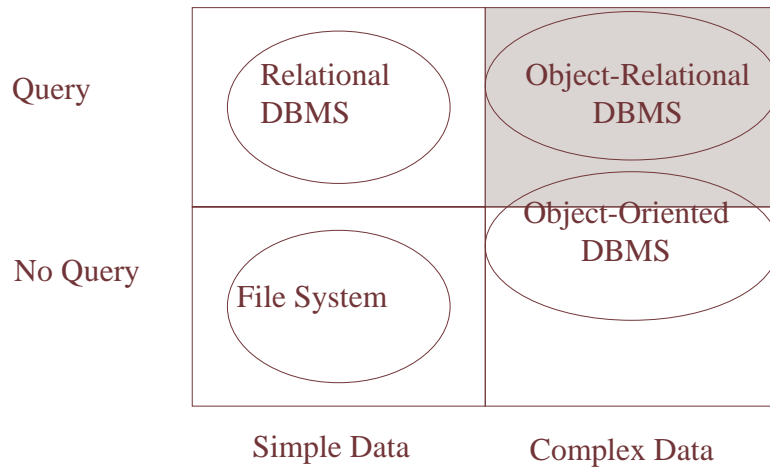
## Shortcomings with RDBMS

- Supports only a small fixed collection of relatively simple data types (integers, floating point numbers, date, strings)
  - No set-valued attributes (sets, lists,...)
  - No inheritance in the Is-a relationship
  - No complex objects, apart from BLOB (binary large object) and CLOB (character large object)
  - Impedance mismatch between data access language (declarative SQL) and host language (procedural C or Java): programmer must explicitly tell how things to be done.
- ➔ Is there a different solution?

## Existing Object Databases

- Object database is a persistent storage manager for objects:
  - Persistent storage for object-oriented programming languages (C++, SmallTalk, etc.)
  - Object-Database Systems:
    - Object-Oriented Database Systems: alternative to relational systems
    - Object-Relational Database Systems: Extension to relational systems
- Market: RDBMS ( \$8 billion), OODMS (\$30 million) world-wide
- OODB Commercial Products: ObjectStore, GemStone, Orion, etc.

## DBMS Classification Matrix



## Object-Relational Features of Oracle

### Methods

```
CREATE TYPE Rectangle_typ AS OBJECT (
    len NUMBER,
    wid NUMBER,
    MEMBER FUNCTION area RETURN NUMBER,
);
```

```
CREATE TYPE BODY Rectangle_typ AS
    MEMBER FUNCTION area RETURN NUMBER IS
BEGIN
    RETURN len * wid;
END area;
END;
```

## Object-Relational Features of Oracle

### Collection types / nested tables

```
CREATE TYPE PointType AS OBJECT (
    x NUMBER,
    y NUMBER);
CREATE TYPE PolygonType AS TABLE OF PointType;
CREATE TABLE Polygons (
    name VARCHAR2(20),
    points PolygonType)
    NESTED TABLE points STORE AS PointsTable;
```

The relations representing individual polygons are not stored directly as values of the points attribute; they are stored in a single table, PointsTable

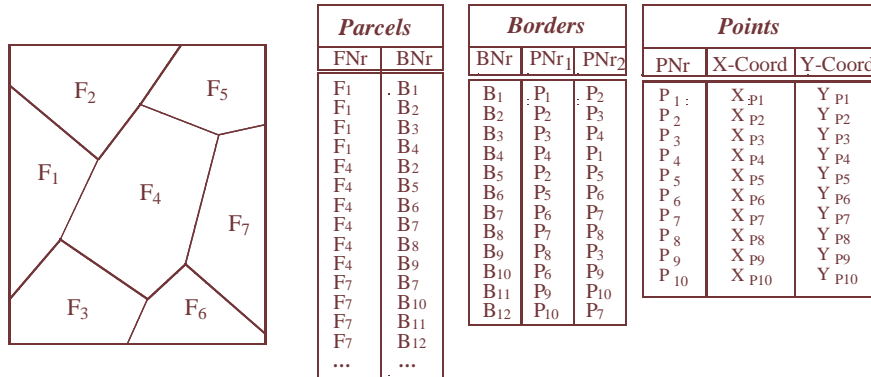
## Spatial Data Management



- Shortcomings of Relational Databases
- Modeling Spatial Data
- Spatial Queries
- Space-Filling Curves + B-Trees
- R-trees

## Relational Representation of Spatial Data

- Example: Representation of geometric objects (here: parcels/fields of land) in normalized relations



Redundancy free representation requires distribution of the information over 3 tables: *Parcels*, *Borders*, *Points*

## Relational Representation of Spatial Data

- For (spatial) queries involving parcels it is necessary to reconstruct the spatial information from the different tables
  - E.g.: if we want to determine if a given point P is inside parcel F<sub>2</sub>, we have to find all corner-points of parcel F<sub>2</sub> first

```
SELECT Points.PNr, X-Coord, Y-Coord
```

```
FROM Parcels, Border, Points
```

```
WHERE FNr = 'F2' AND
```

```
Parcel.BNr = Borders.BNr AND
```

```
(Borders.PNr1 = Points.PNr OR
```

```
Borders.PNr2 = Points.PNr)
```

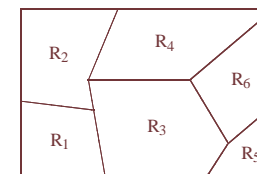
- Even this simple query requires expensive joins of three tables
- Querying the geometry (e.g., P in F<sub>2</sub>?) is not directly supported.

## Extension of the Relational Model to Support Spatial Data

- Integration of spatial data types and operations into the core of a DBMS (→ object-oriented and object-relational databases)
  - Data types such as *Point*, *Line*, *Polygon*
  - Operations such as *ObjectIntersect*, *RangeQuery*, etc.
- Advantages
  - Natural extension of the relational model and query languages
  - Facilitates design and querying of spatial databases
  - Spatial data types and operations can be supported by spatial index structures and efficient algorithms, implemented in the core of a DBMS
- All major database vendors today implement support for spatial data and operations in their database systems via object-relational extensions

## Extension of the Relational Model to Support Spatial Data – Example

Relation: **ForestZones**(Zone: *Polygon*, ForestOfficial: *String*, Area: *Cardinal*)



ForestZones		
Zone	ForestOfficial	Area (m <sup>2</sup> )
R <sub>1</sub>	Stevens	3900
R <sub>2</sub>	Behrens	4250
R <sub>3</sub>	Lee	6700
R <sub>4</sub>	Goebel	5400
R <sub>5</sub>	Jones	1900
R <sub>6</sub>	Kent	4600

- The province decides that a reforestation is necessary in an area described by a polygon S. Find all forest officials affected by this decision.

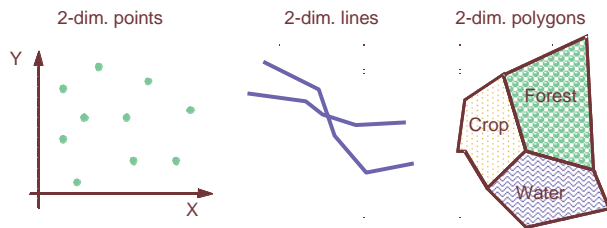
```
SELECT ForestOfficial
```

```
FROM ForestZones
```

```
WHERE ObjectIntersects (S, Zone)
```

## Data Types for Spatial Objects

- Spatial objects are described by
  - Spatial Extent
    - location and/or boundary with respect to a reference point in a coordinate system, which is at least 2-dimensional.
    - Basic object types: *Point, Lines, Polygon*
  - Other Non-Spatial Attributes
    - Thematic attributes such as height, area, name, land-use, etc.



## Spatial Data Management



- Shortcomings of Relational Databases
- Modeling Spatial Data
- Spatial Queries
- Space-Filling Curves + B-Trees
- R-trees

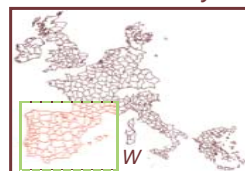
## Spatial Query Processing

- DBMS has to support two types of operations
  - Operations to retrieve certain subsets of spatial object from the database
    - “Spatial Queries/Selections”, e.g., window query, point query, etc.
  - Operations that perform basic geometric computations and tests
    - E.g., point in polygon test, intersection of two polygons etc.
- Spatial selections, e.g. in geographic information systems, are often supported by an interactive graphical user interface

Point Query

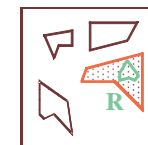


Window Query

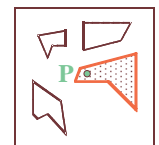


## Basic Spatial Queries

- Containment Query:** Given a spatial object R, find all objects that completely contain R. If R is a Point: **Point Query**
- Region Query:** Given a region R (polygon or circle), find all spatial objects that intersect with R. If R is a rectangle: **Window Query**
- Enclosure Query:** Given a polygon region R, find all objects that are completely contained in R
- K-Nearest Neighbor Query:** Given an object P, find the k objects that are closest to P (typically for points)



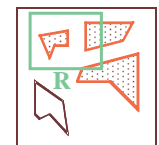
Containment Query



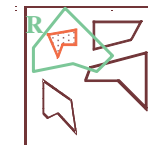
Point Query



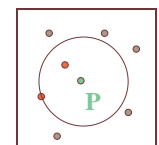
Region Query



Window Query



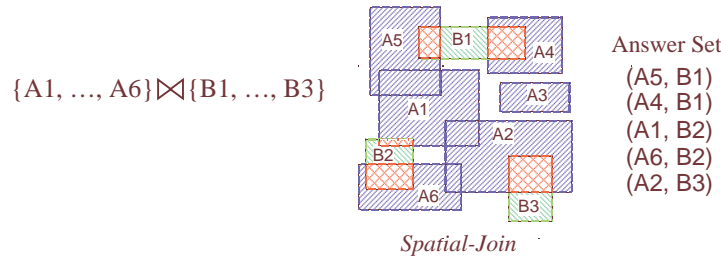
Enclosure Query



2-nn Query

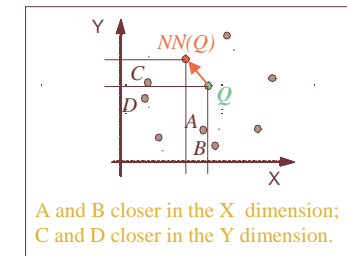
## Basic Spatial Queries – Spatial Join

- Given two sets of spatial objects (typically minimum bounding rectangles)
  - $S_1 = \{R_1, R_2, \dots, R_m\}$  and  $S_2 = \{R'_1, R'_2, \dots, R'_n\}$
- Spatial Join: Compute all pairs of objects  $(R, R')$  such that
  - $R \in S_1, R' \in S_2$ ,
  - and  $R$  intersects  $R'$  ( $R \cap R' \neq \emptyset$ )
  - Spatial predicates other than intersection are also possible, e.g. all pairs of objects that are within a certain distance from each other



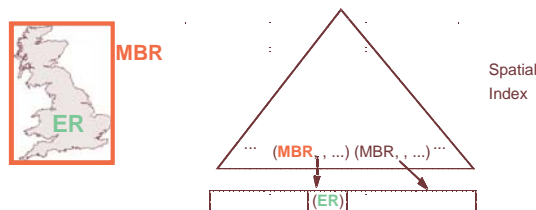
## Index Support for Spatial Queries

- Conventional index structures such as B-trees are not designed to support spatial queries
  - Group objects only along one dimension
  - Do not preserve spatial proximity
    - E.g. nearest neighbor query:
      - Nearest neighbor of  $Q$  is typically not the nearest neighbor in any single dimension



## Index Support for Spatial Queries

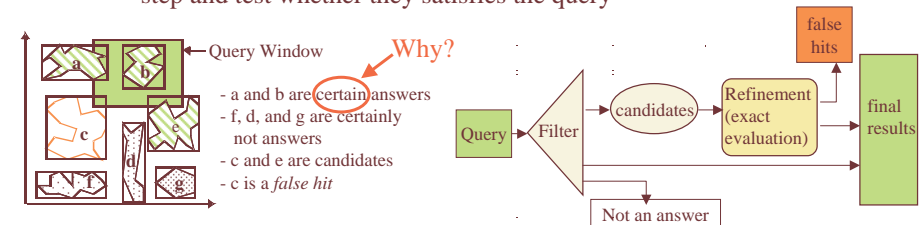
- Spatial index structures try to preserve spatial proximity
  - Group objects that are close to each other on the same data page
  - Problem: the number of bytes to store extended spatial objects (lines, polygons) varies
  - Solution:
    - Store *Approximations* of spatial objects in the index structure, typically axis-parallel minimum bounding rectangles (MBR)
    - Exact object representation (ER) stored separately; pointers to ER in the index



## Query Processing Using Approximations

### Two-Step Procedure

- Filter Step:
  - Use the index to find all approximations that satisfy the query
  - Some objects already satisfy the query based on the approximation, others have to be checked in the refinement step  $\rightarrow$  Candidate Set
- Refinement Step:
  - Load the exact object representations for candidates left after the filter step and test whether they satisfies the query



# Spatial Data Management



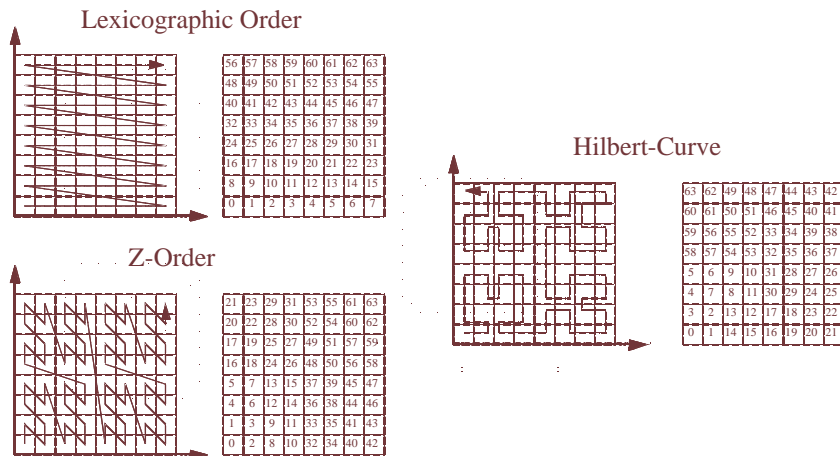
- Shortcomings of Relational Databases
- Modeling Spatial Data
- Spatial Queries
- Space-Filling Curves + B-Trees
- R-trees

# Embedding of the 2-dimensional space into a 1 dimensional space

- Basic Idea:
  - The data space is partitioned into rectangular cells.
  - Use a space filling curve to assign cell numbers to the cells (define a linear order on the cells)
    - The curve should preserve spatial proximity as good as possible
    - Cell numbers should be easy to compute
  - Objects are approximated by cells.
  - Store the cell numbers for objects in a conventional index structure with respect to the linear order

21	23	29	31	53	55	61	63
20	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

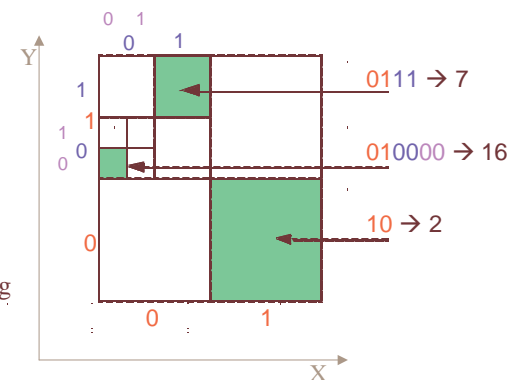
## Space Filling Curves



- Z-Order preserves spatial proximity relatively good
- Z-Order is easy to compute

## Z-Order – Z-Values

- Coding of Cells
  - Partition the data space recursively into two halves
  - Alternate X and Y dimension
  - Left/bottom  $\rightarrow$  0
  - Right/top  $\rightarrow$  1
- **Z-Value:  $(c, l)$**   
 $c$  = decimal value of the bit string  
 $l$  = level (number of bits)



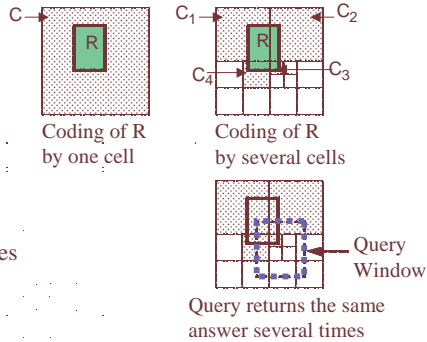
if all cells are on the same level, then  $l$  can be omitted

# Z-Order – Representation of Spatial Objects

- For Points
  - Use a fixed resolution of the space in both dimensions, i.e., each cell has the same size
  - Each point is then approximated by one cell

21	23	29	31	53	55	61	63
30	22	28	30	52	54	60	62
17	19	25	27	49	51	57	59
16	18	24	26	48	50	56	58
5	7	13	15	37	39	45	47
4	6	12	14	36	38	44	46
1	3	9	11	33	35	41	43
0	2	8	10	32	34	40	42

- For extended spatial object
  - minimum enclosing cell
    - Problems with cells that intersect the first partitions already
  - improvement: use several cells
    - Better approximation of the objects
    - Redundant storage
    - Redundant retrieval in spatial queries



# Z-Order – Mapping to a B<sup>+</sup>-Tree

- Linear Order for Z-values to store them in a B<sup>+</sup>-tree:
  - Let  $(c_1, l_1)$  and  $(c_2, l_2)$  be two Z-Values and let  $l = \min\{l_1, l_2\}$ .

The order relation  $\leq_Z$  (that defines a linear order on Z-values) is then defined by

$$(c_1, l_1) \leq_Z (c_2, l_2) \text{ iff } (c_1 \text{ div } 2^{(l_1-l)}) \leq (c_2 \text{ div } 2^{(l_2-l)})$$

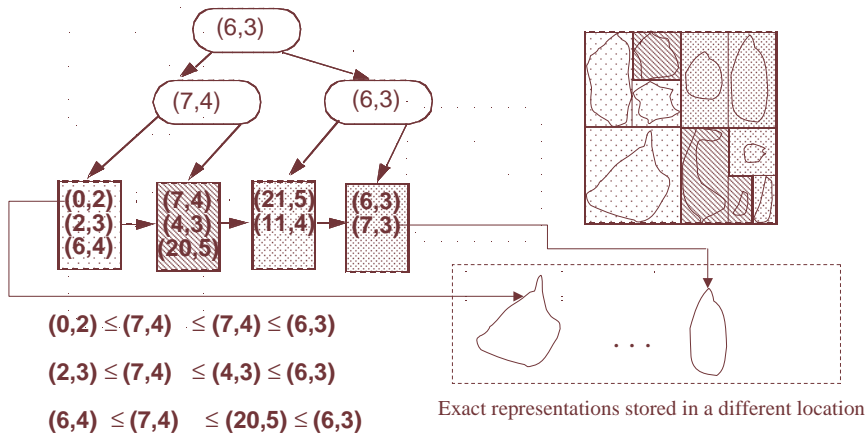
Examples:

$$(1,2) \leq_Z (3,2),$$

$$(3,4) \leq_Z (3,2),$$

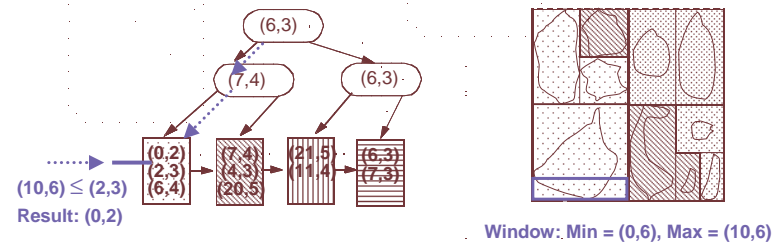
$$(1,2) \leq_Z (10,4)$$

# Mapping to a B<sup>+</sup>-Tree - Example



# Mapping to a B<sup>+</sup>-Tree – Window Query

- Window Query  $\rightarrow$  Range Query in the B<sup>+</sup>-tree
  - find all entries (Z-Values) in the range  $[l, u]$  where
    - $l$  = smallest Z-Value of the window (bottom left corner)
    - $u$  = largest Z-Value of the window (top right corner)
    - $l$  and  $u$  are computed with respect to the maximum resolution/length of the Z-values in the tree (here: 6)





# Spatial Data Management

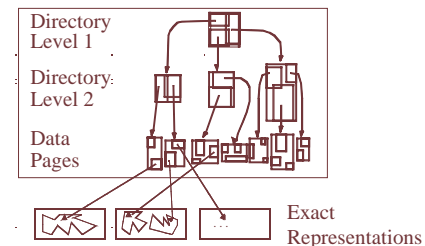


- Shortcomings of Relational Databases
- Modeling Spatial Data
- Spatial Queries
- Space-Filling Curves + B-Trees
- R-trees

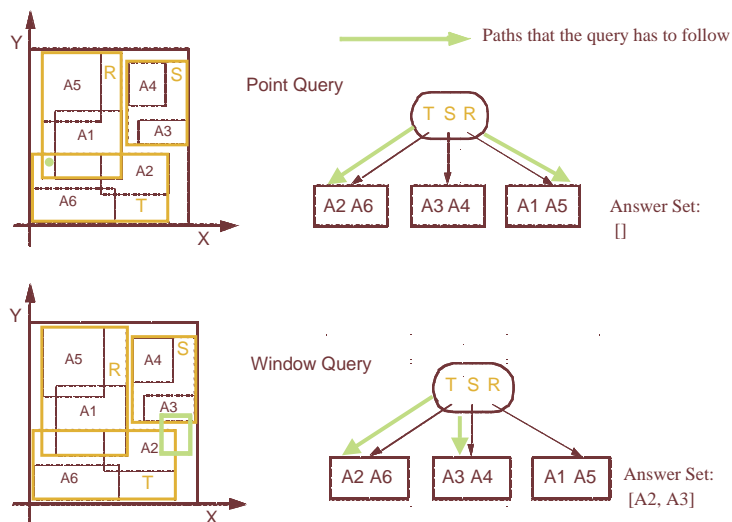
# The R-Tree – Properties

- Balanced Tree designed to organize rectangles [Gut 84].
- Each page contains between  $m$  and  $M$  entries.
- Data page entries are of the form  $(MBR, PointerToExactRepr)$ .
  - MBR is a minimum bounding rectangle of a spatial object, which PointerToExactRepr is pointing to
- Directory page entries are of the form  $(MBR, PointerToSubtree)$ .
  - MBR is the minimum bounding rectangle of all entries in the subtree, which PointerToSubtree is pointing to.
- Rectangles can overlap
- The height  $h$  of an R-Tree for  $N$  spatial objects:

$$h \leq \lceil \log_m N \rceil + 1$$



# The R-Tree – Queries



# The R-Tree – Queries

**PointQuery** (Page, Point);

```

FOR ALL Entry ∈ Page DO
  IF Point IN Entry.MBR THEN
    IF Page = DataPage THEN
      PointInPolygonTest (load(Entry.ExactRepr), Point)
    ELSE
      PointQuery (Entry.Subtree, Point);
  
```

First call: Page = Root of the R-tree

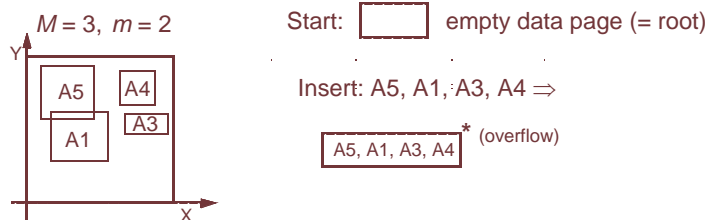
**Window Query** (Page, Window);

```

FOR ALL Entry ∈ Page DO
  IF Window INTERSECTS Entry.MBR THEN
    IF Page = DataPage THEN
      Intersection (load(Entry.ExactRepr), Window)
    ELSE
      WindowQuery (Entry.Subtree, Window);
  
```

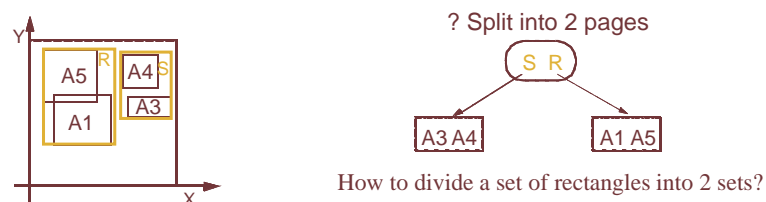
## R-Tree Construction – Optimization Goals

- Overlap between the MBRs
  - ⇒ spatial queries have to follow several paths
  - ⇒ try to minimize overlap
- Empty space in MBR
  - ⇒ spatial queries may have to follow irrelevant paths
  - ⇒ try to minimize area and empty space in MBRs



## R-Tree Construction – Important Issues

- Split Strategy



- Insertion Strategy

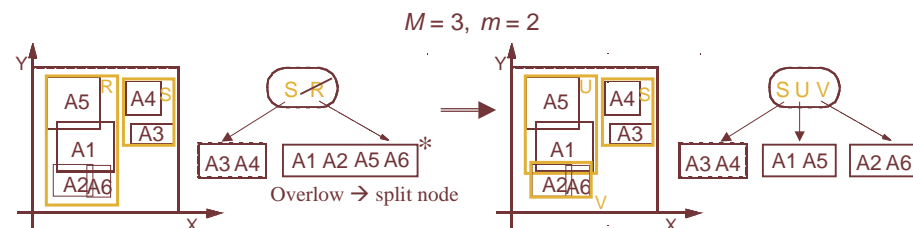


## R-Tree Construction – Insertion Strategies

- Dynamic construction by insertion of rectangles  $R$ 
  - Searching for the data page into which  $R$  will be inserted, traverses the tree from the root to a data page.
  - When considering entries of a directory page  $P$ , 3 cases can occur:
    1.  $R$  falls into exactly one  $Entry.MBR$ 
      - follow  $Entry.Subtree$
    2.  $R$  falls into the MBR of more than one entry  $e_1, \dots, e_n$ 
      - follow  $E_i.Subtree$  for entry  $e_i$  with the smallest area of  $e_i.MBR$ .
    3.  $R$  does not fall into an  $Entry.MBR$  of the current page
      - check the increase in area of the  $MBR$  for each entry when enlarging the  $MBR$  to enclose  $R$ . Choose  $Entry$  with the minimum increase in area (if this entry is not unique, choose the one with the smallest area); enlarge  $Entry.MBR$  and follow  $Entry.Subtree$
- Construction by “bulk-loading” the rectangles
  - Sort the rectangles, e.g., using Z-Order
  - Create the R-tree “bottom-up”

## R-Tree Construction – Split

- Insertion will eventually lead to an overflow of a data page
  - The parent entry for that page is deleted.
  - The page is split into 2 new pages - according to a *split strategy*
  - 2 new entries pointing to the newly created pages are inserted into the parent page.
  - A now possible overflow in the parent page is handled recursively in a similar way; if the root has to be split, a new root is created to contain the entries pointing to the newly created pages.

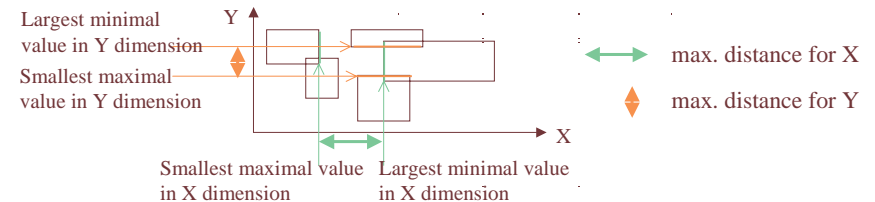


## R-Tree Construction – Splitting Strategies

- Overflow of node  $K$  with  $|K| = M+1$  entries  $\rightarrow$  Distribution of the entries into two new nodes  $K_1$  and  $K_2$  such that  $|K_1| \geq m$  and  $|K_2| \geq m$
- *Exhaustive algorithm:*
  - Searching for the “best” split in the set of all possible splits is too expensive ( $O(2^M)$  possibilities!)
- *Quadratic algorithm:*
  - Choose the pair of rectangles  $R_1$  and  $R_2$  that have the largest value  $d(R_1, R_2)$  for empty space in an MBR, which covers both  $R_1$  and  $R_2$ .  
 $d(R_1, R_2) := \text{Area}(\text{MBR}(R_1 \cup R_2)) - (\text{Area}(R_1) + \text{Area}(R_2))$
  - Set  $K_1 := \{R_1\}$  and  $K_2 := \{R_2\}$
  - Repeat until STOP
    - if all  $R_i$  are assigned: STOP
    - if all remaining  $R_i$  are needed to fill the smaller node to guarantee minimal occupancy  $m$ : assign them to the smaller node and STOP
    - else: choose the next  $R_i$  and assign it to the node that will have the smallest increase in area of the MBR by the assignment. If not unique: choose the  $K_i$  that covers the smaller area (if still not unique: the one with less entries).

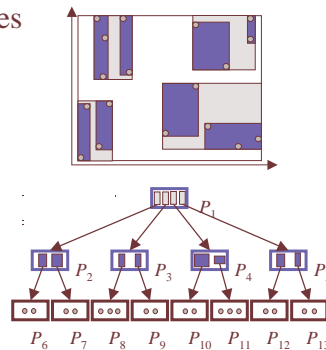
## R-Tree Construction – Splitting Strategies

- *Linear algorithm:*
  - Same as the quadratic algorithm, except for the choice of the initial pair: Choose the pair with the largest distance.
    - For each dimension determine the rectangle with the largest minimal value and the rectangle with the smallest maximal value (the difference is the *maximal distance/separation*).
    - Normalize the maximal distance of each dimension by dividing by the sum of the extensions of the rectangles in this dimension
    - Choose the pair of rectangles that has the greatest normalized distance.  
 Set  $K_1 := \{R_1\}$  and  $K_2 := \{R_2\}$ .



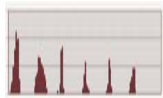


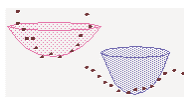


## R-Trees – Variants

- Many variants of R-trees exist,
  - e.g., the R\*-tree, X-tree for higher dimensional point data, ...
  - For further information see <http://www.cs.umd.edu/~hjs/rtrees/index.html> (includes an interactive demo)
- R-trees are also efficient index structures for point data since points can be modeled as “degenerated” rectangles
  - Multi-dimensional points, where a distance function between the points is defined play an important role for similarity search in so-called “feature” or “multi-media” databases.



## Examples of Feature Databases

- Measurements for celestial objects (e.g., intensity of emission in different wavelengths)
    - 
  - Colour histograms of images
    -   $\rightarrow$  
  - Documents, shape descriptors, ...
    -   
- $n$   $d$ -dimensional feature vectors  
 $(o1_1, o1_2, \dots, o1_d)$   
 $(o2_1, o2_2, \dots, o2_d)$   
 $\dots$   
 $(on_1, on_2, \dots, on_d)$

# Feature Databases and Similarity Queries

- Objects + Metric Distance Function

- The distance function measures (dis)similarity between objects

- Basic types of similarity queries

- range queries with range  $\epsilon$ 
  - Retrieves all objects which are similar to the query object up to a certain degree  $\epsilon$
- $k$ -nearest neighbor queries
  - Retrieves  $k$  most similar objects to the query

