Lecture 07

Indexes

Summary – last week

- Physical Level
  - Relational Implementation through:
    - Star schema: improves query performance for often-used data
    - Snowflake schema: reduce the size of the dimension tables
  - Array based storage
    - How to perform linearization
  - MOLAP, ROLAP, HOLAP

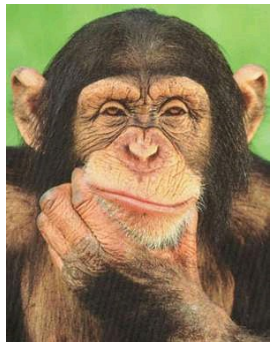
This week:

- Indexes
  - Tree based indexes
  - Bitmap indexes

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Indexes

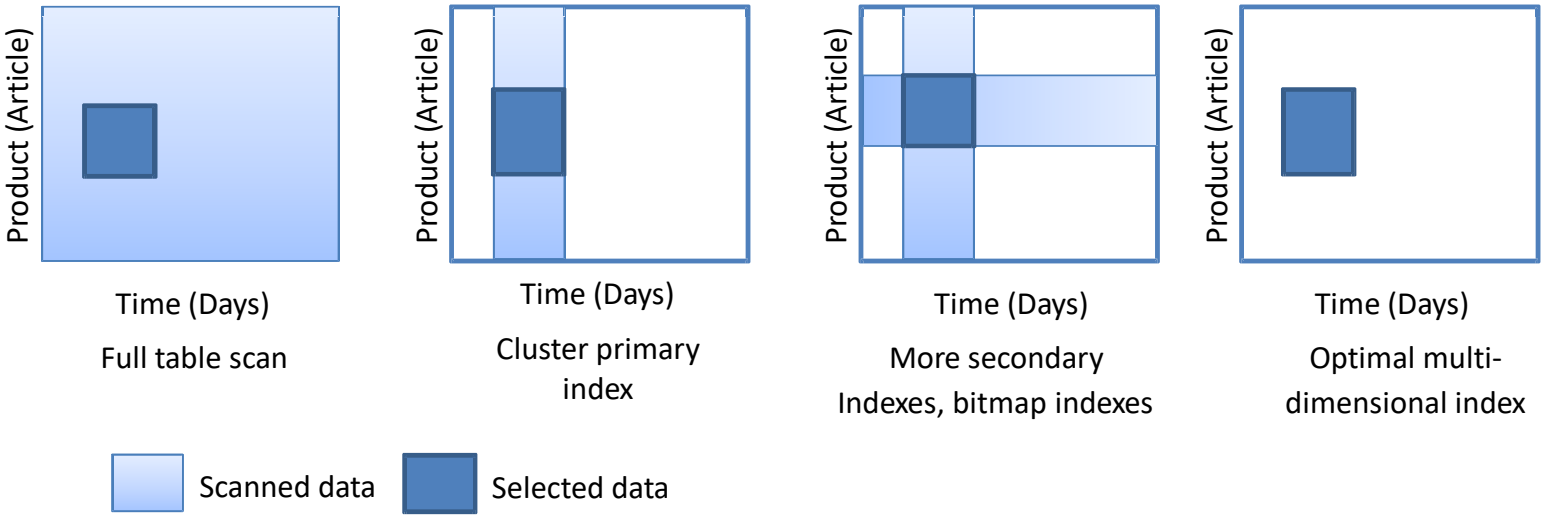
- Why index?
  - Consider a 100 GB table; at 100 MB/s read speed we need 17 minutes for a full table scan
  - Consider an OLAP query: the number of Bosch S500 washing machines sold in Germany last month?
    - Applying restrictions (product, location, and time) the selectivity would be strongly reduced
      - If we have 30 location, 10000 products and 24 months in the DW, the selectivity is
        1/30 * 1/ 10000 * 1/24 = 0,00000014
  - So…we read 100 GB for 1.4KB of data
    …not very smart



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Indexes (cont'd.)

- Reduce the size of read pages to minimum with indexes

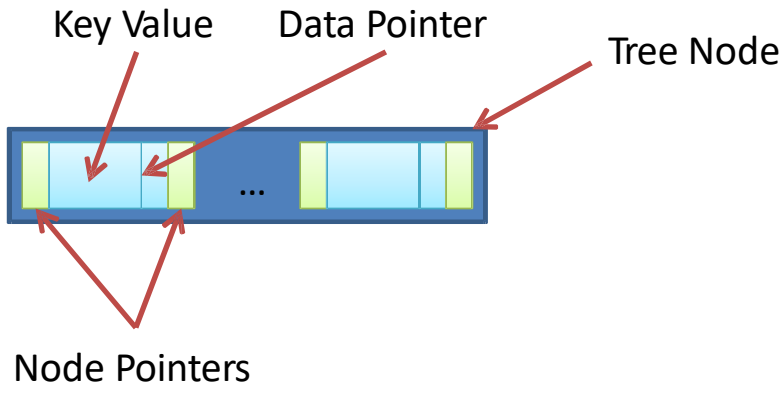


Indexes generate some overhead but in DB not much in DW.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

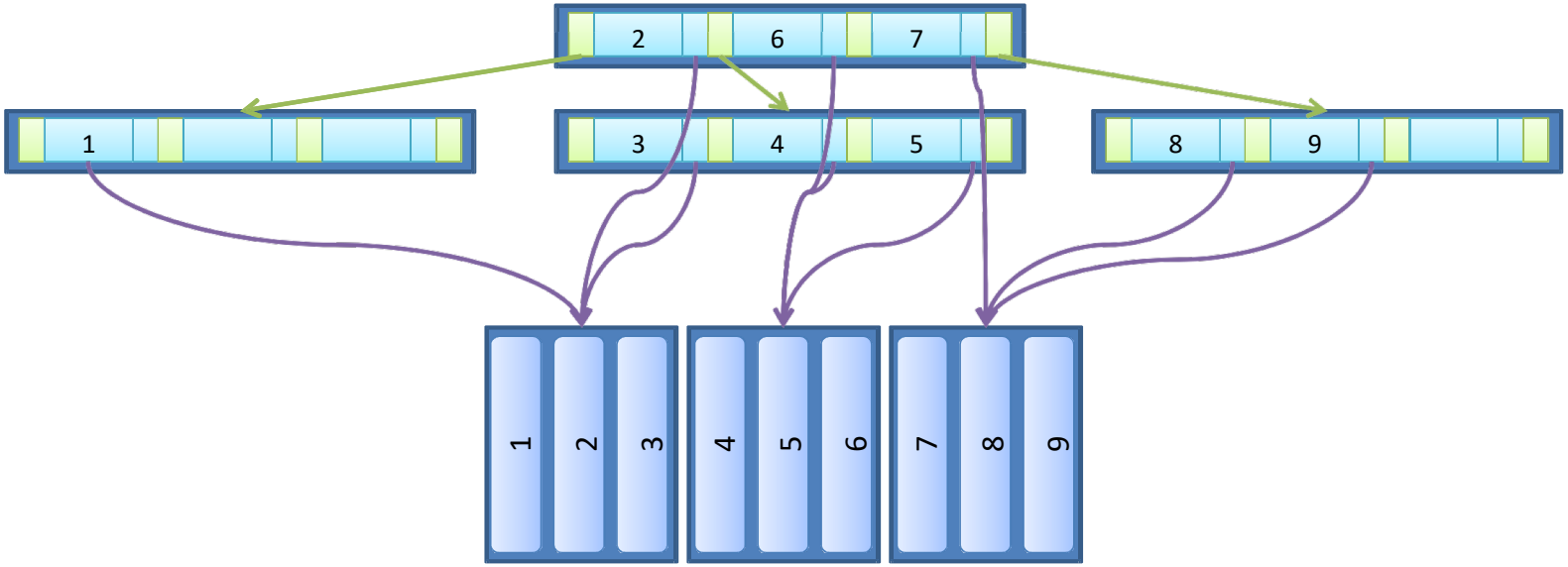
Tree Based Indexes

- In the beginning…there were B-Trees
  - Data structures for storing sorted data with amortized run times for insertion and deletion
  - Basic structure of a node



Tree Structures

- Search in database systems
  - B-tree structures allow exact search with logarithmic costs

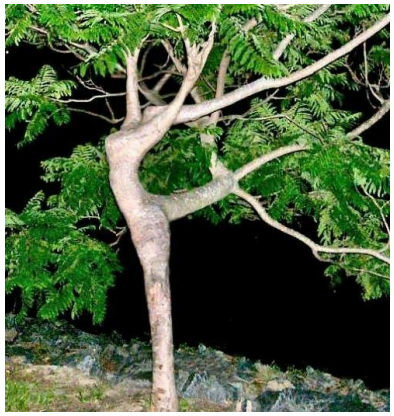


DBMS reads the whole block in memory even it needs only one data value and keep this block in memory (unless memory space is available) for the future purposes.

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Tree Structures (cont'd.)

- Search in DWs
  - The data is multidimensional, B-trees however, support only one-dimensional search
- Are there any possibilities to extend tree functionality for multidimensional data?



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

R-Tree

- The **R-tree** (Guttman, 1984) is the prototype of a multi-dimensional extension of the classical B-trees
- Frequently used for low-dimensional applications (used to about 10 dimensions), such as geographic information systems
- More scalable versions: R^+-Trees, R*-Trees and X-Trees (each up to 20 dimensions for uniform distributed data)

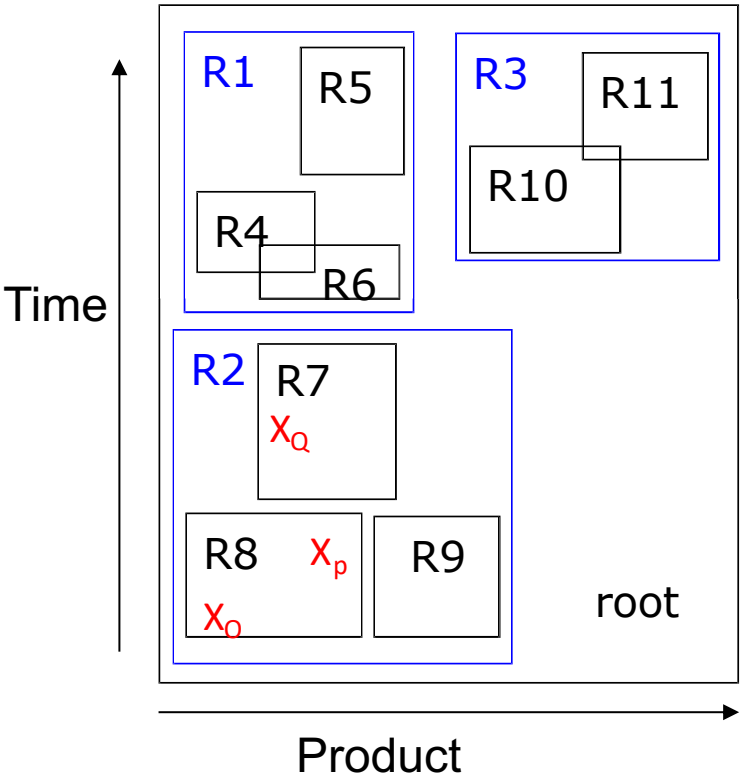
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

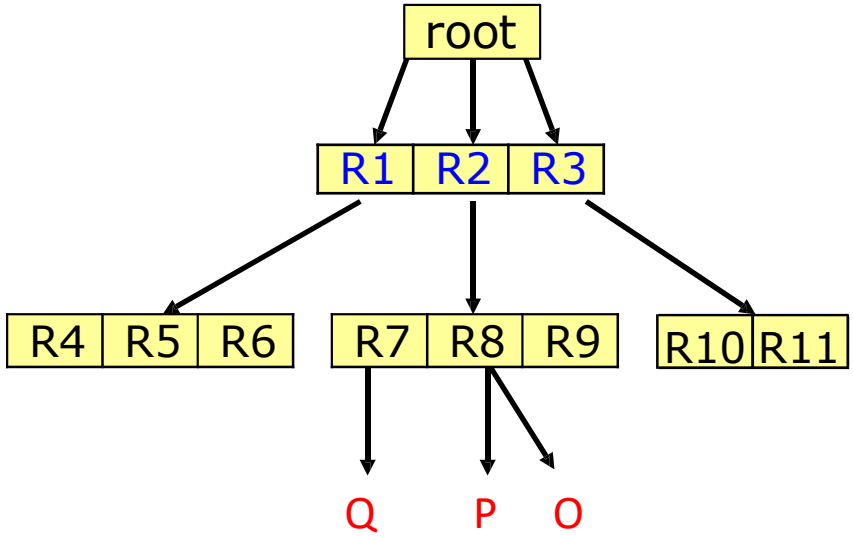
R-Tree Structure

- **Dynamic Index Structure**
  (insert, update and delete are possible)
- Data structure
  - **Data pages** are leaf nodes and store clustered point data and data objects
  - **Directory pages** are the internal nodes and store directory entries
  - Multidimensional data are structured with the help of **Minimum Bounding Rectangles** (MBRs)

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

R-Tree Example





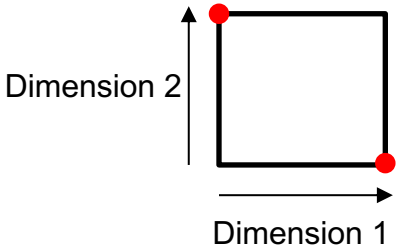
R-Tree Characteristics

- **Local grouping** for clustering – not effective for uniformly distributed data.
- **Overlapping clusters** (the more the clusters overlap the more **inefficient** is the index)
- **Height balanced** tree structure (therefore all the children of a node in the tree have about the same number of successors)
- Objects are **stored**, only in the **leaves**
  - Internal nodes are used for navigation – query can be answered without searching of all levels of tree
- MBRs are used as a **geometry**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

R-Tree Properties

- The root has **at least two children**
- Each internal node has between m and M children
- M and $m \leq M / 2$ are pre-defined parameters
- For each entry $(I, \text{child pointer})$ in an internal node, I is the **smallest rectangle** that contains the rectangles of the child nodes e.g. for two dimensions I should be $[(i_{d1s}\text{-}i_{d1e}), (i_{d2s}\text{-}i_{d2e})]$.



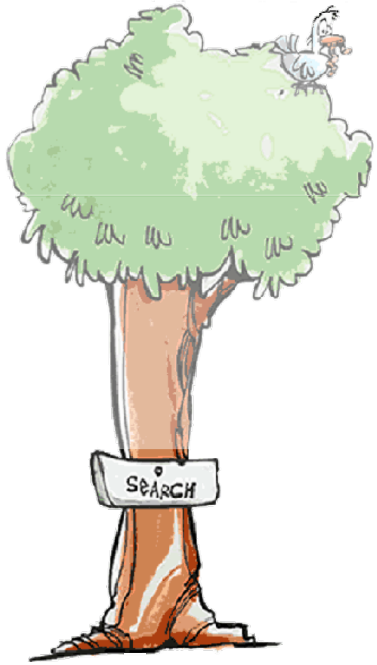
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Operations of R-Tree

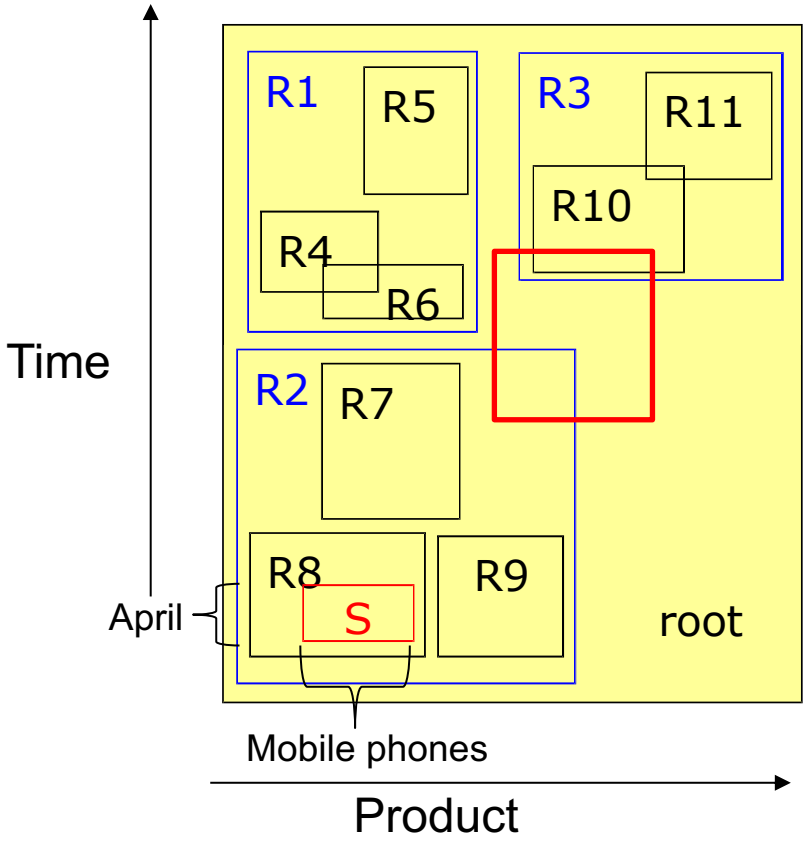
- The essential operations for the use and management of an R-tree are
  - Search
  - Insert
  - Updates
  - Delete
  - Splitting

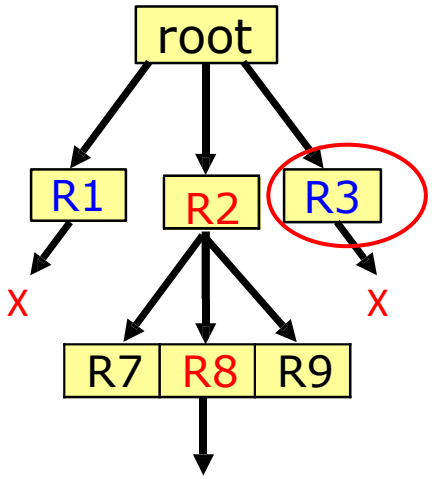
Searching in R-Trees

- The tree is searched **recursively** from the root to the leaves
  - One path is selected
  - If the requested record has not been found in that sub-tree, the next path is traversed
- The path selection is arbitrary



Example





- Check only 7 nodes instead of 12

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Searching in R-Trees (cont'd.)

- **No guarantee** for good performance
- In the worst case, all paths must traversed (due to overlaps of the MBRs)
- Search algorithms try to exclude as many irrelevant regions as possible ("pruning")

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Insert (or loading in DW)

- **Procedure**
  - The **best leaf** page is chosen (ChooseLeaf) considering the spatial criteria
    - Best leaf: the leaf that needs the smallest volume growth to include the new object
  - The object will be inserted there if there is enough room (number of objects in the node < M)

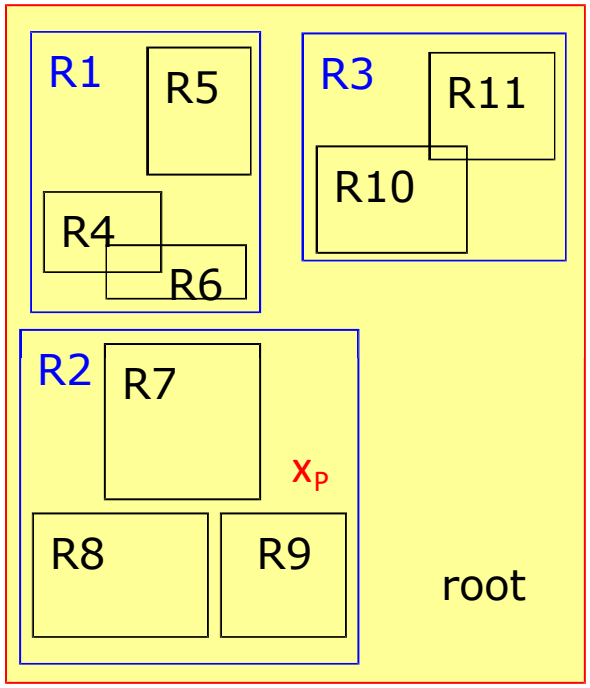
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

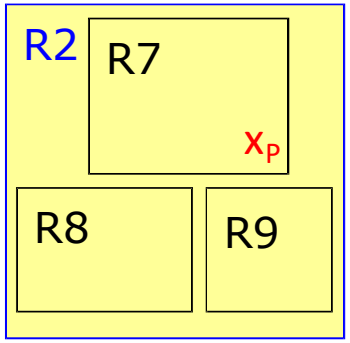
Insert (cont'd.)

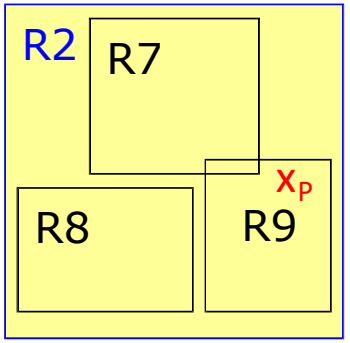
- If there is no more place left in the node, it is considered a case for **overflow** and the node is divided (SplitNode)
  - Goal of the split is to result in **minimal overlap** and as **small dead space as possible**
- Interval of the parent node must be adapted to the new object (AdjustTree)
- If the root is reached by division, then create a new root whose children are the two split nodes of the old root

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

R-Tree Insert Example







- Inserting P either in R7 or R9
- In R7, it needs more space, but does not overlap

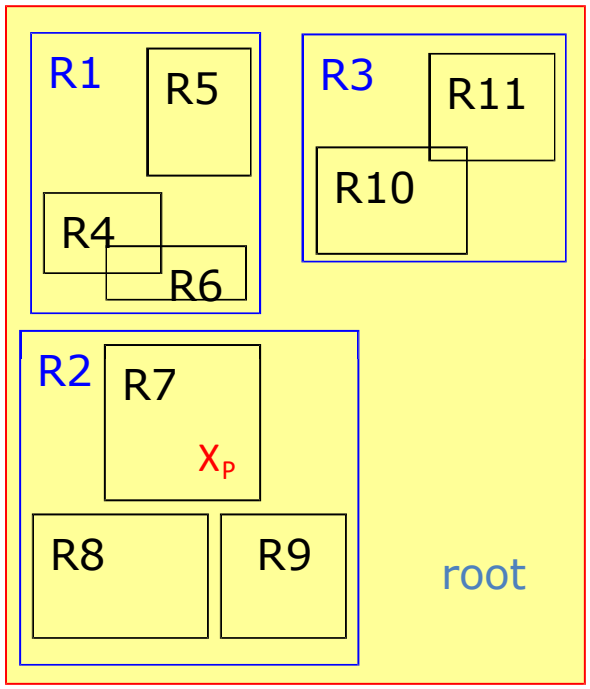
Heuristics

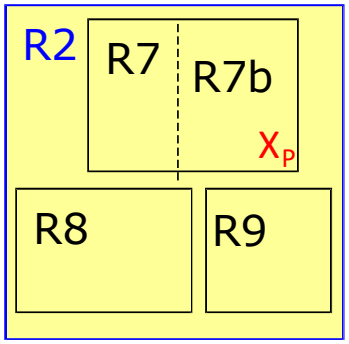
- An object is always inserted in the nodes, to which it produces the smallest increase in volume
- If it falls in the interior of a MBR no enlargement is need
- If there are several possible nodes, then select the one with the **smallest volume**

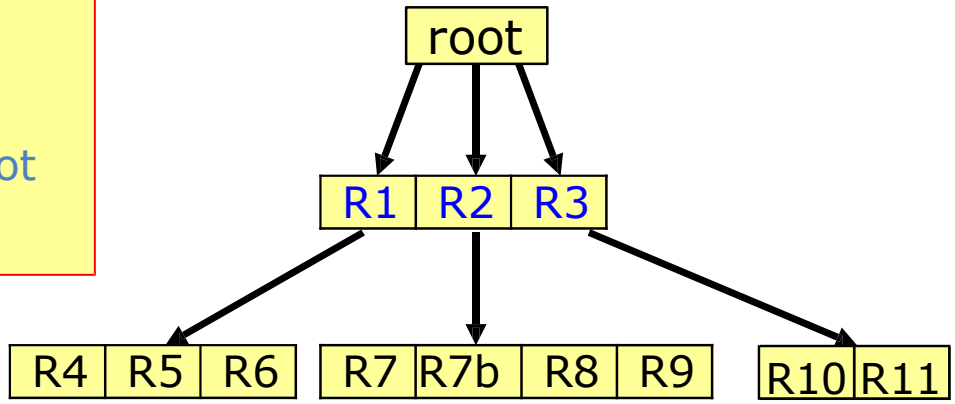
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Insert with Overflow

Overflow because M=3







Suppose R7 already have 3 elements.

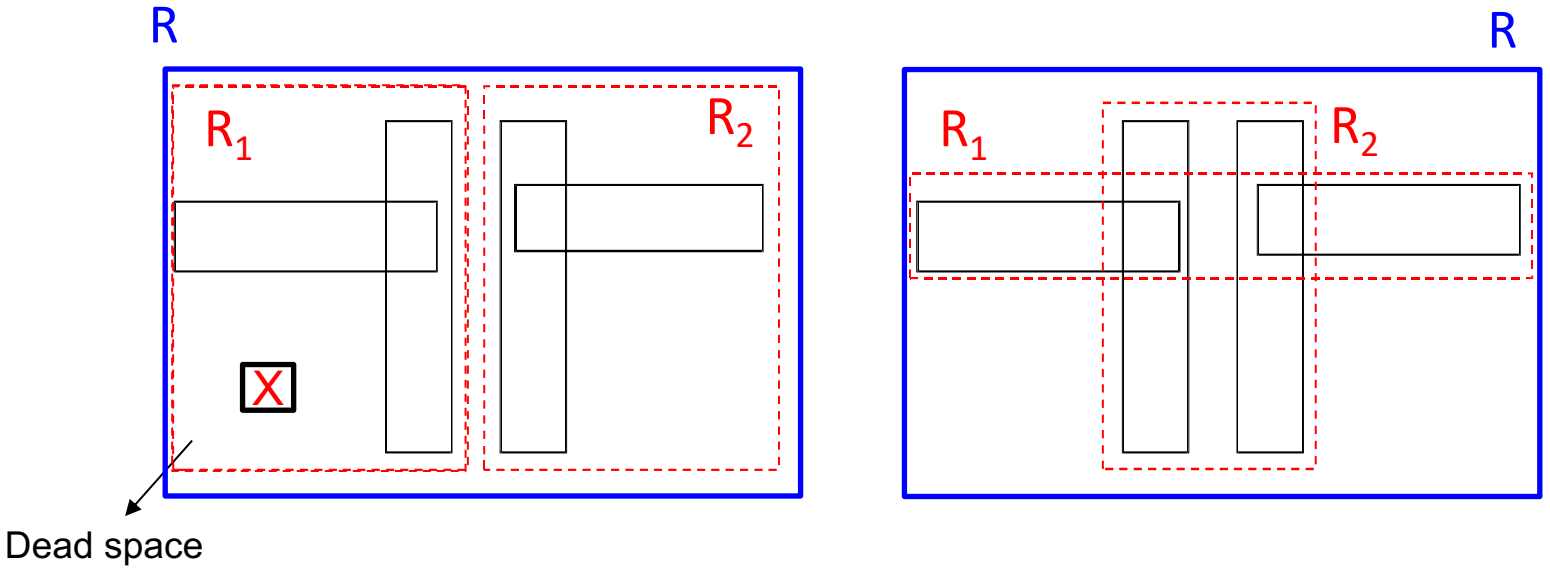
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

SplitNode

- If an object is inserted in a full node, then the $M+1$ objects will be divided among two new nodes
- The goal in splitting is that it should rarely be needed to traverse both resulting nodes on subsequent searches
  - Therefore use small MBRs. This leads to minimal overlapping with other MBRs

Split Example

- Calculate the minimum total area of two rectangles, and minimize the **dead space**



Bad split

Better Split

Overflow Problem

- Deciding on how exactly to perform the splits is **not trivial**
  - All objects of the old MBR can be divided in different ways on two new MBRs
  - The volume of both resulting MBRs should remain as small as possible

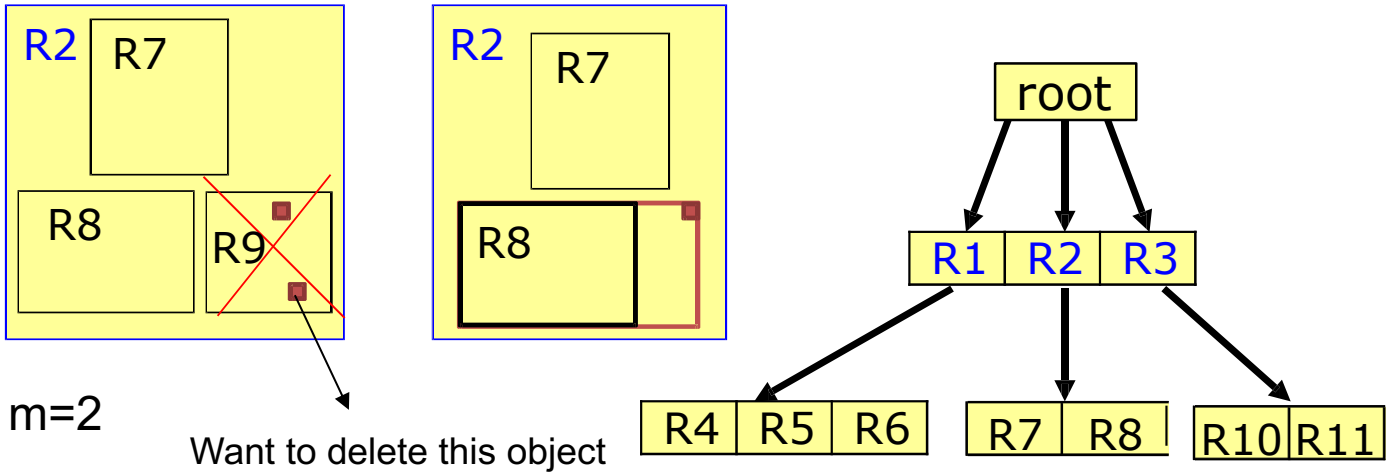
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Delete

- Procedure
  - Search the leaf node with the object to delete (FindLeaf)
  - Delete the object (deleteRecord)
  - The tree is condensed (CondenseTree) if the resulting node has < m objects
  - When **condensing**, a node is completely erased and the objects of the node which should have remained are **reinserted**
  - If the root remains with just one child, the child will become the new root

Example

- An object from R9 is deleted
  (1 object remains in R9, but m = 2)
  - Due to few objects R9 is deleted, and R2 is reduced (condenseTree)

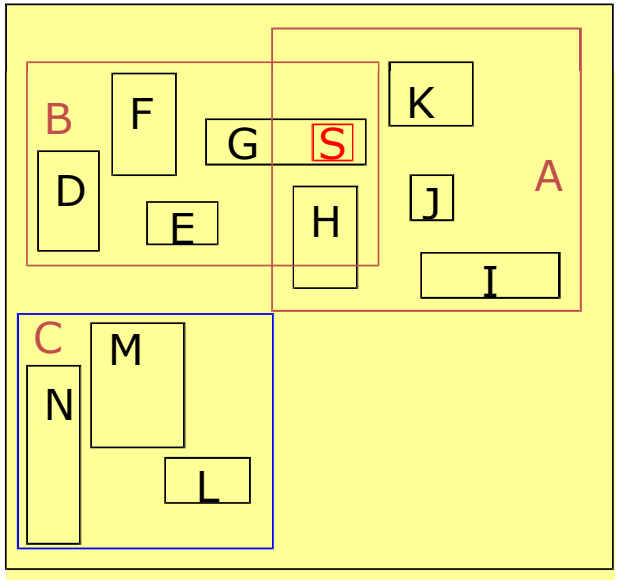


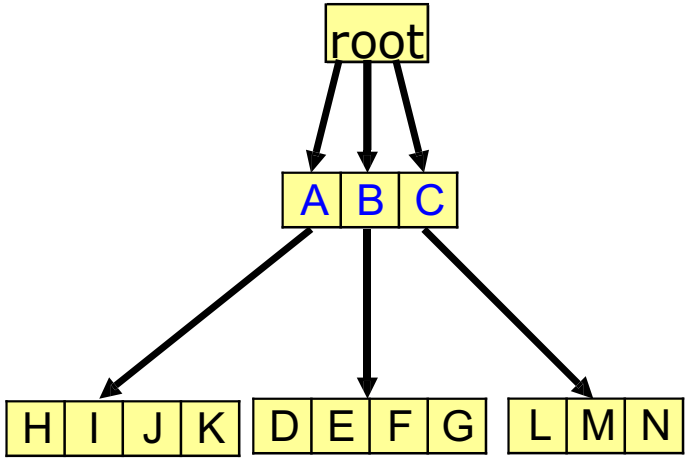
Update

- If a record is updated, its surrounding rectangle can change
- The index entry must then be deleted updated and then re-inserted

Block Access Cost

- The most efficient search in R-trees is performed when the **overlap** and the **dead space** are minimal





Avoiding overlapping is only possible if data points are known in advance

We did the traversing of extra 4 nodes which was not required if we knew that A and B are not overlapped.

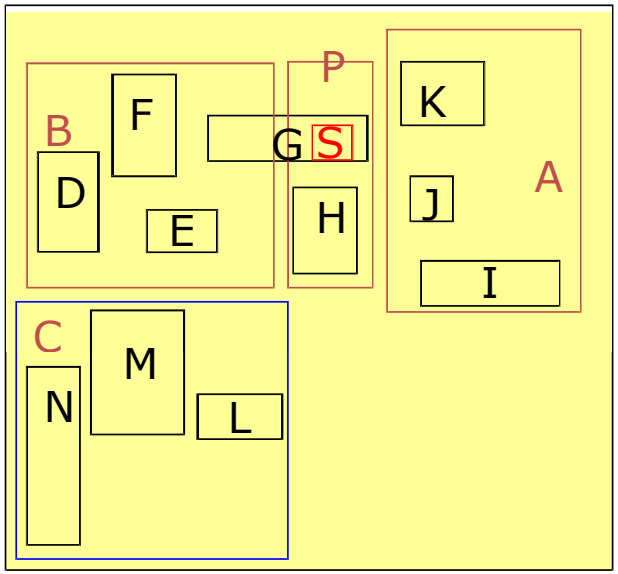
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

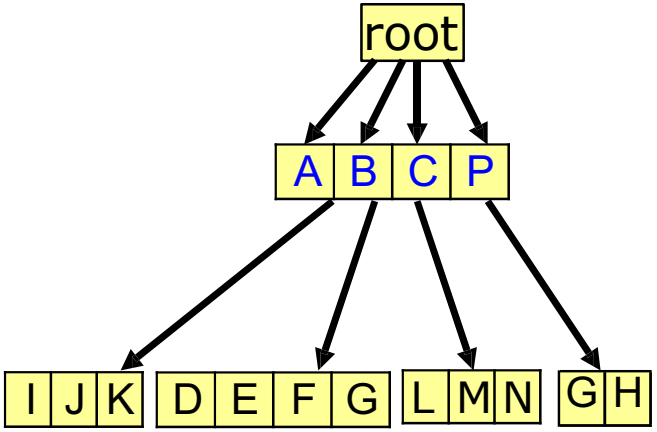
Improved Version of R-Tree

- Where are R-trees inefficient?
  - They **allow overlapping** between neighboring MBRs
- R^+-Trees (Sellis ua, 1987)
  - **Overlapping** of neighboring MBRs are prohibited
  - This may lead to identical leafs occurring more than once in the tree
  - Improve search efficiency, but similar scalability as R-trees

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

R+ -Tree





- Overlaps are not permitted (A and P)
- Data rectangles are divided and may be present (e.g., G) in several leafs

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Performance

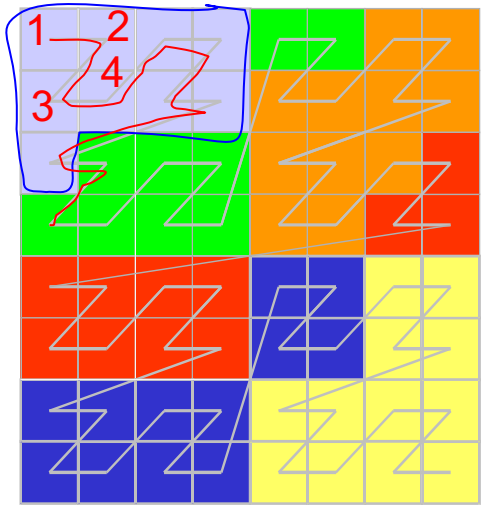
- The **main advantage** of R^+-trees is to improve the search performance
- Especially for point queries, this saves 50% of access time
- **Drawback** is the low occupancy of nodes resulting through many splits
- R^+-trees often **degenerate** with the increasing number of changes

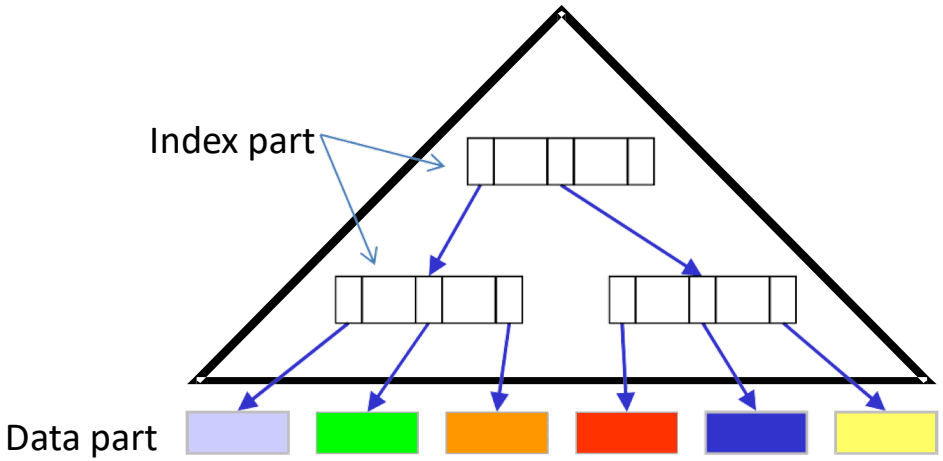
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees

- Combination of B*-Tree and Z-curve = Universal B-Tree (UB-tree)
  - Z-curve is used to map multidimensional points to one-dimensional values (Z-values)

    Z-values are used as keys in B*-Tree

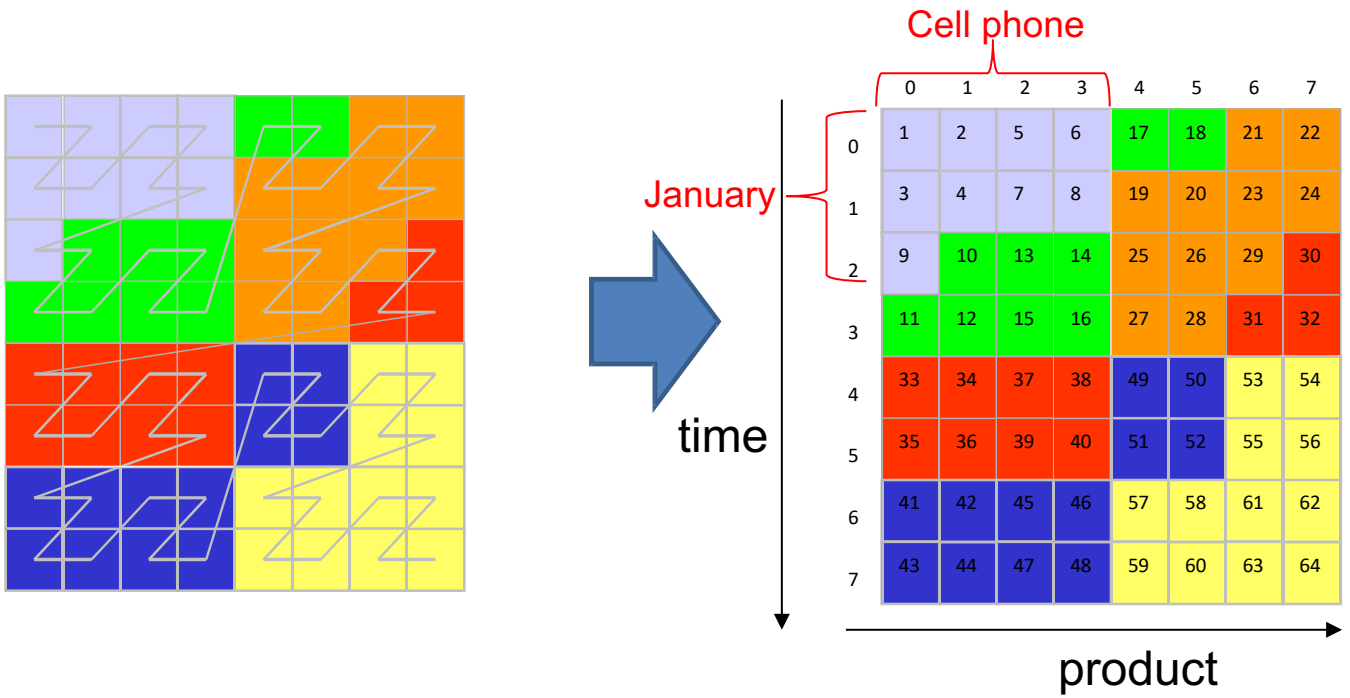




Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

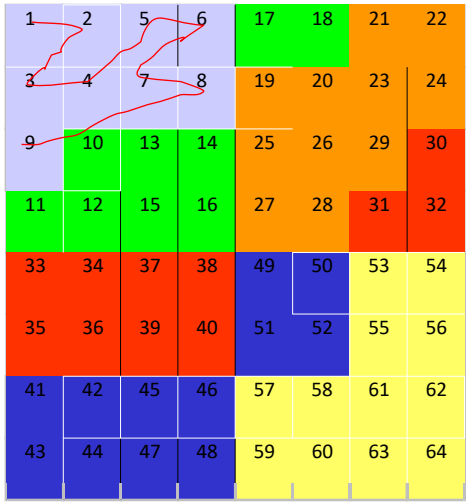
- Concept of Z-Regions
  - To create a disjunctive partitioning of the multidimensional space
  - This allows for very efficient processing of **multidimensional range queries**



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

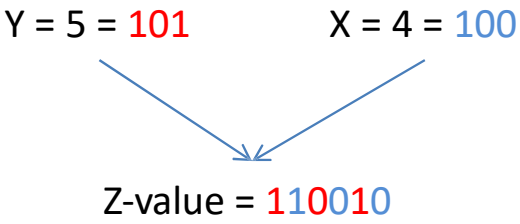
- Z-Regions
  - The space covered by an interval on the Z-Curve
  - Defined by two Z-Addresses a and b
    - We call it the region address of [a :b]
  - Each Z-Region maps exactly **onto one page** on secondary storage
    - I.e., to one leaf page of the B*-Tree
  - E.g., of Z-Regions
    - [1:9], [10, 18], [19, 28]…

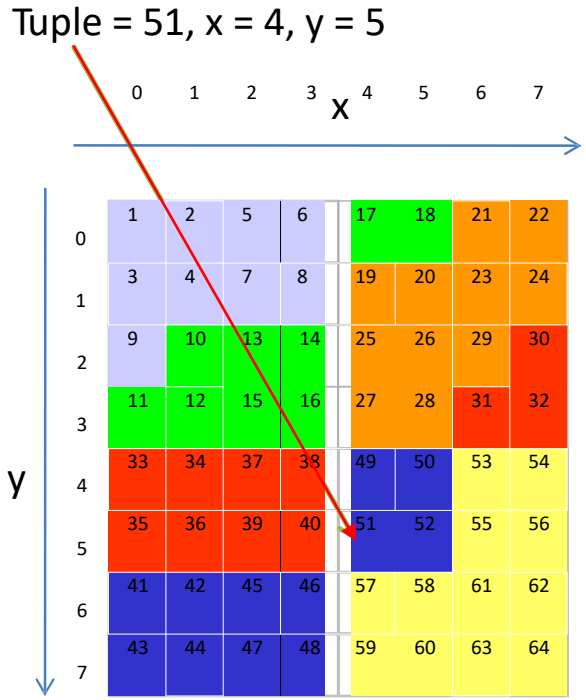


Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

- Z-Value address representation
  - Calculated through bit **interleaving** of the coordinates of the tuple





Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

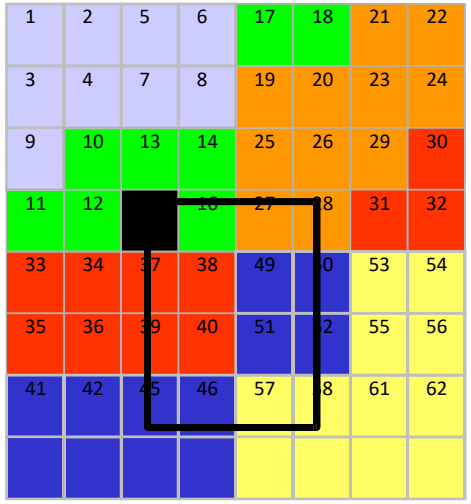
UB-Trees (cont'd.)

- Why Z-Values?
  - With Z-Values we reduce the dimensionality of the data to one dimension
  - Z-Values are then used as keys in B*-trees
  - Z-Value are very important for **range queries**!

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

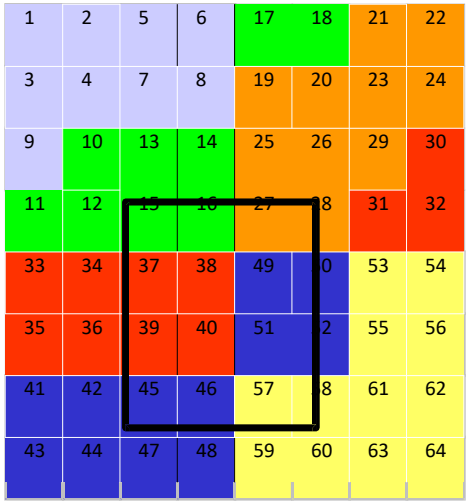
- Range queries (RQ) in UB-Trees
  - Each query can be specified by 2 coordinates
    - q_a (the upper left corner of the query rectangle)
    - q_b (the lower right corner of the query rectangle)
  - RQ-algorithm
    1. Starts with q_a and calculates its Z-Region
       1. Z-Region of q_a is [10:18]



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

- Range queries (RQ) in UB-Trees
  2. The corresponding page is loaded and filtered with the query predicate
     1. Tupels 15 and 16 fulfill the predicate
  3. The next region (inside the query rectangle) on the Z-curve is calculated
     1. The next jump point on the Z-curve is 27
  4. Repeat steps 2 and 3 until the end-address of the last filtered region is bigger than q_b



Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

UB-Trees (cont'd.)

- The critical part of the algorithm is **calculating the jump point** on the Z-curve which is inside the query rectangle
  - If this takes too long it eliminates the advantage obtained through optimized disk access
  - How is the jump point optimally calculated?
    - From 3 points: q_a, q_b and the current Z-Region
    - By performing bit operations

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes

- Bitmap Indexes
  - Lets assume a relation Expenses with three attributes: Nr, Shop and Sum
  - A bitmap index for attribute Shop looks like this

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Value	Vector
P&C	001000
Real	010101
Saturn	100010

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
~~5~~	~~Saturn~~	~~350~~
6	Real	80

- Handling modification
  - Assume record numbers are not changed
- Deletion
  - **Tombstone** replaces deleted record (6 doesn't become 5!)
  - Corresponding bit is set to 0

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	001000
Real	010101
Saturn	100000

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

- Inserted record is assigned the next record number
  - A bit of value 0 or 1 is appended to each bit vector
  - If new record contains a new value of the attribute, add one bit-vector
    - E.g., insert new record with REWE as shop

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80
7	REWE	23

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	0010000
Real	0101010
Saturn	1000100
REWE	0000001

Bitmap Indexes (cont'd.)

- Update
  - Change the bit corresponding to the old value of the modified record to 0
  - Change the bit corresponding to the new value of the modified record to 1
  - If the new value is a new value of A, then insert a new bit-vector: e.g., replace Shop for record 2 to REWE

Nr	Shop	Sum
1	Saturn	150
2	REWE	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Before

Value	Vector
P&C	001000
Real	010101
Saturn	100010

After

Value	Vector
P&C	001000
Real	000101
Saturn	100010
REWE	010000

Bitmap Indexes (cont'd.)

- Select
  - Basic AND, OR bit operations:
    - E.g., select the sums we have spent in Saturn and P&C

Nr	Shop	Sum
1	Saturn	150
2	Real	65
3	P&C	160
4	Real	45
5	Saturn	350
6	Real	80

Saturn	OR	P&C	=	Result
1		0		1
0		0		0
0		1		1
0		0		0
1		0		1
0		0		0

Value	Vector
P&C	001000
Real	010101
Saturn	100010

  - Bitmap indexes should be used when selectivity is **high**

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Bitmap Indexes (cont'd.)

- Advantages
  - Operations are efficient and easy to implement (directly supported by hardware)
- Disadvantages
  - For each new value of an attribute a new bitmap-vector is introduced
    - If we bitmap index an attribute like birthday (only day) we have 365 vectors: 365/8 bits ≈ 46 Bytes for a record, just for that
    - Solution to such problems is multi-component bitmaps
  - Not fit for range queries where many bitmap vectors have to be read
    - Solution: range-encoded bitmap indexes

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Summary

- B-Trees are not fit for multidimensional data
- R-Trees
  - MBR as geometry to build multidimensional indexes
  - Operations: select, insert, overflow problem, node splitting , delete
  - Inefficient because they allow overlapping between neighboring MBRs
  - R^+-trees - improve the search performance

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Summary (cont'd.)

- UB-Trees
  - Reduce multidimensional data to one dimension in order to use B-Tree indexes
  - Z-Regions, Z-Curve, use the advantage of bit operations to make optimal jumps
- Bitmap indexes
  - Great for indexing tables with set-like attributes e.g., Gender: Male/Female
  - Operations are efficient and easy to implement (directly supported by hardware)
  - Multi component reduce the storage while range encoded allow for fast range queries

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Next Lecture

- Optimization
  - Partitioning
  - Joins
  - MaterializedViews

Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides