Lecture 11

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

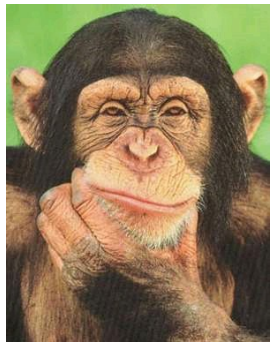
This week:

- Indexes
  - Tree based indexes
  - Bitmap indexes

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

Indexes

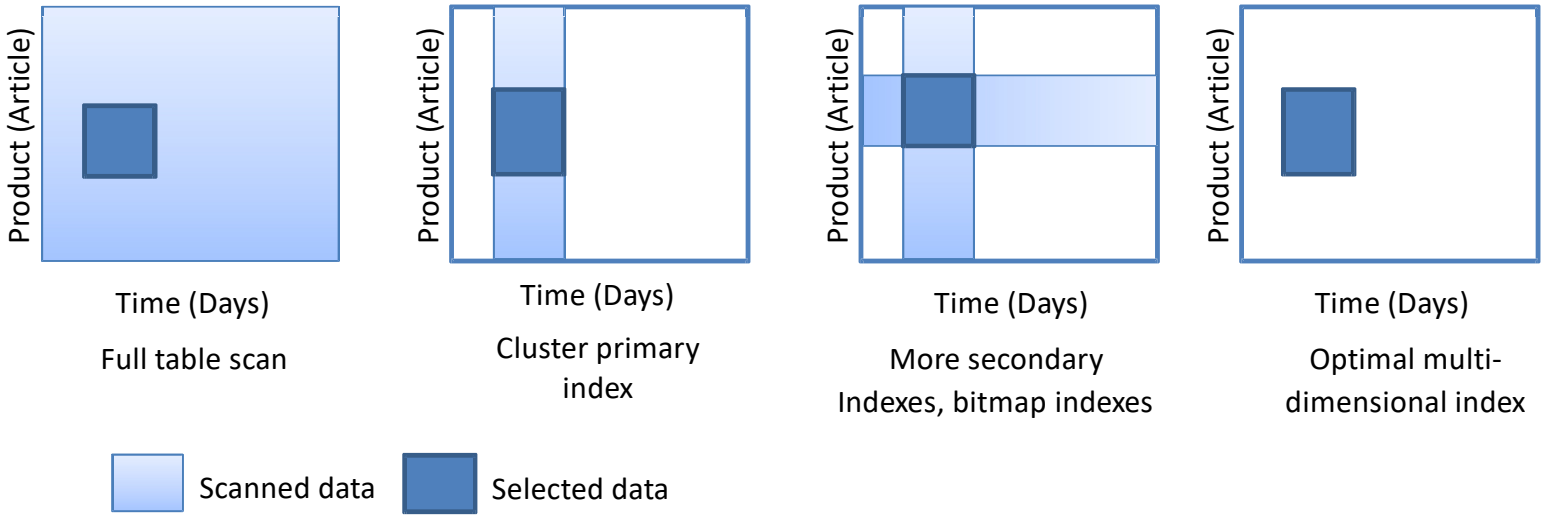
- Why index?
  - Consider a 100 GB table; at 100 MB/s read speed we need 17 minutes for a full table scan
  - Consider an OLAP query: the number of Bosch S500 washing machines sold in Germany last month?
    - Applying restrictions (product, location, and time) the selectivity would be strongly reduced
      - 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

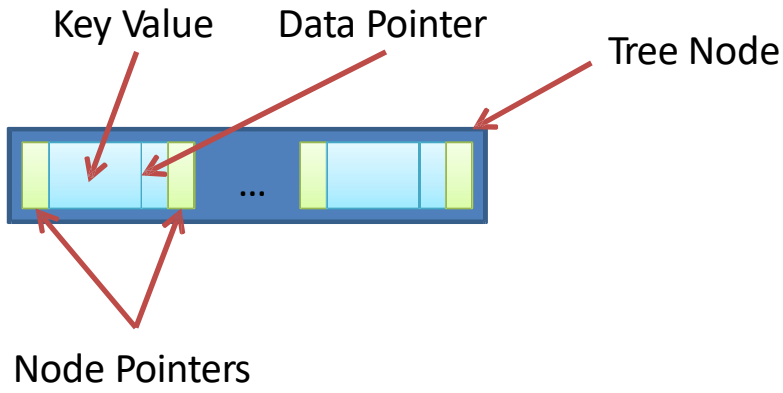


Indexes generate some overhead but in DB not much in DW.

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

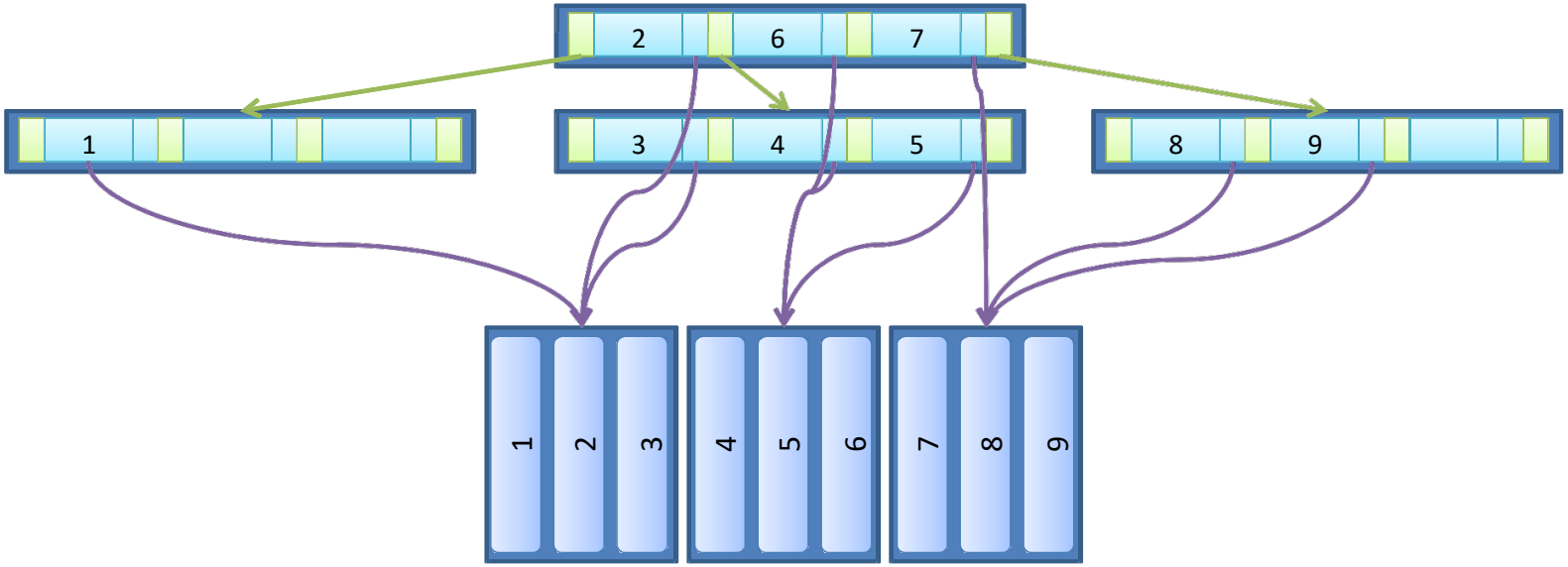
Tree Based Indexes

- 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

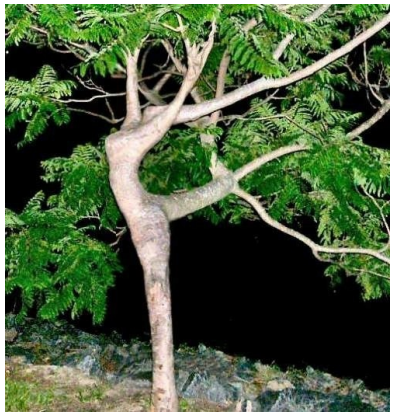


DBMS reads the whole block in memory even it needs only one data value and keep this block in memory (unless memory space is not available) for the future purposes.

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)

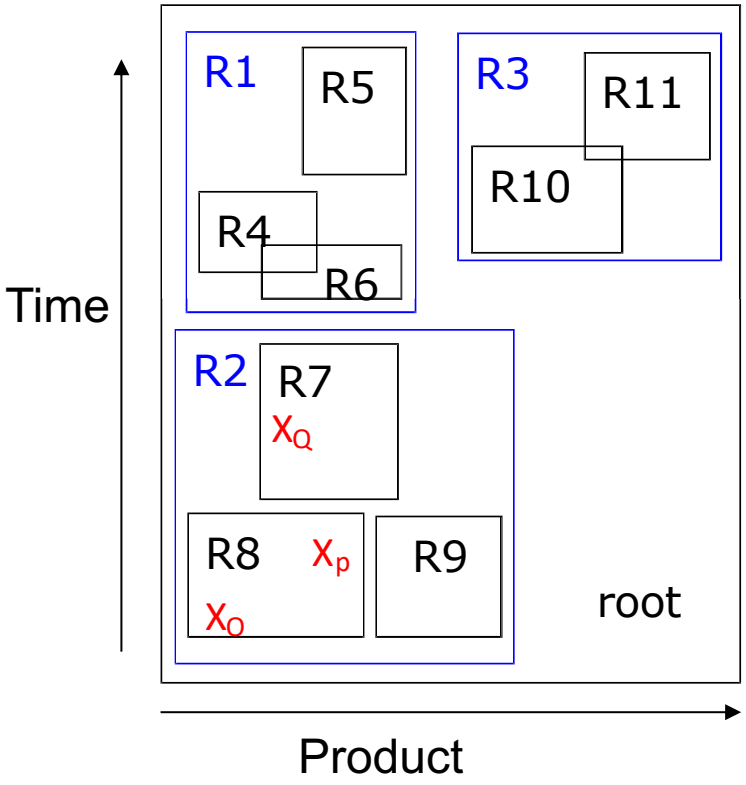
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

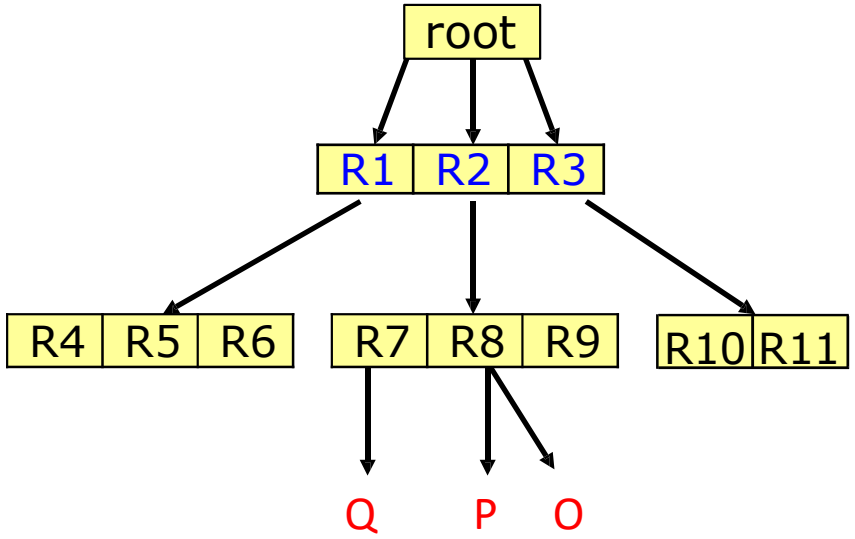
R-Tree Structure

- **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

Operations of R-Tree

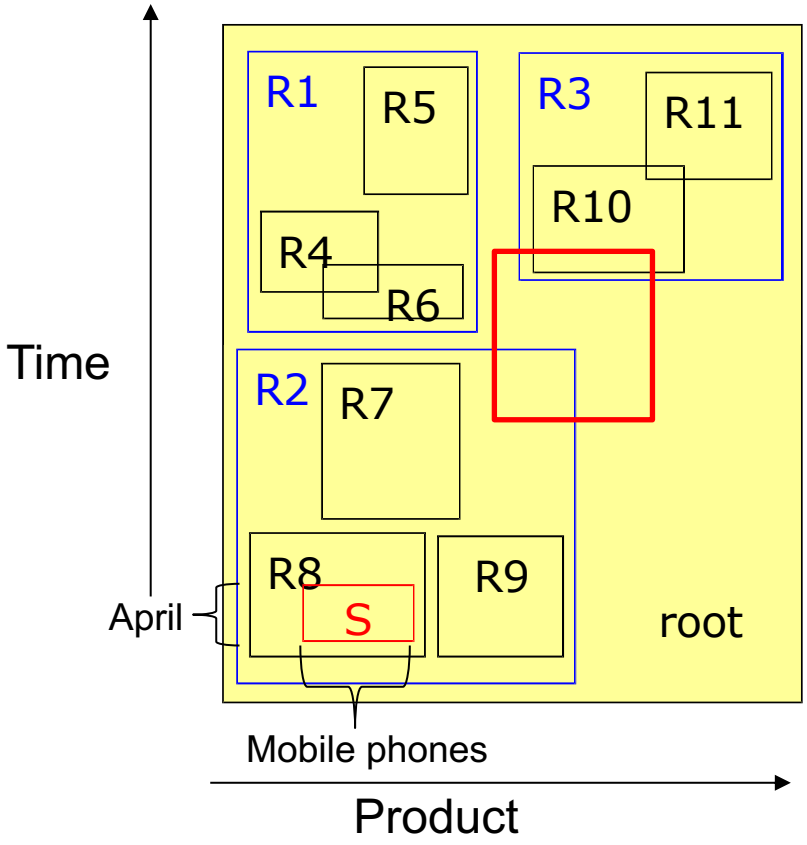
- The essential operations for the use and management of an R-tree are
  - Search
  - Insert
  - Updates
  - Delete
  - Splitting

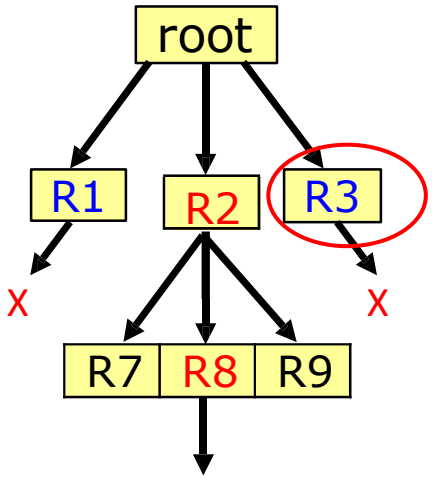
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

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)

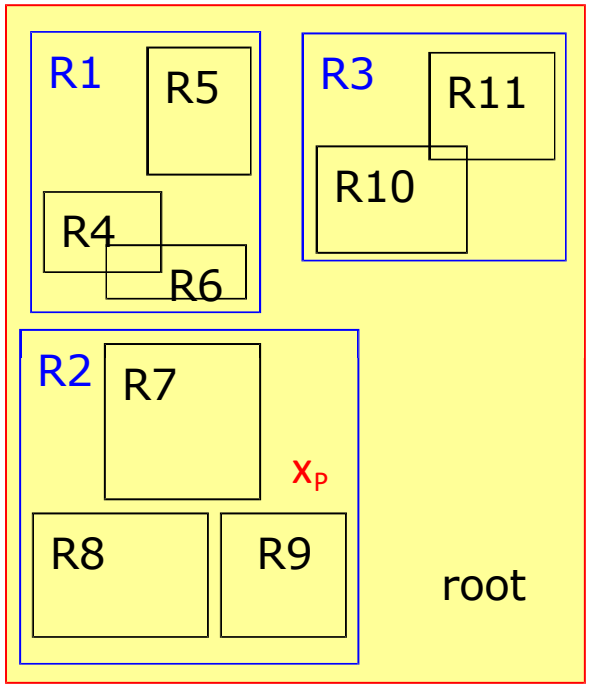
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

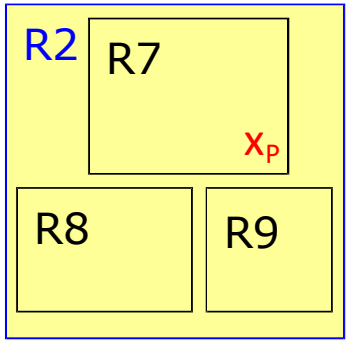
Insert (cont'd.)

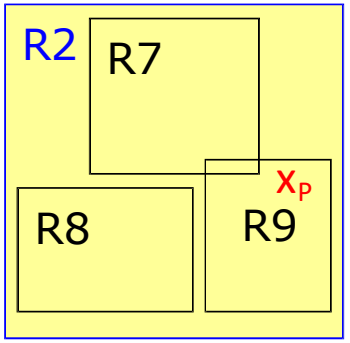
- If there is no more place left in the node, it is considered a case for **overflow** and the node is divided (SplitNode)
  - Goal of the split is to result in **minimal overlap** and as **small dead space as possible**
- Interval of the parent node must be adapted to the new object (AdjustTree)
- If 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

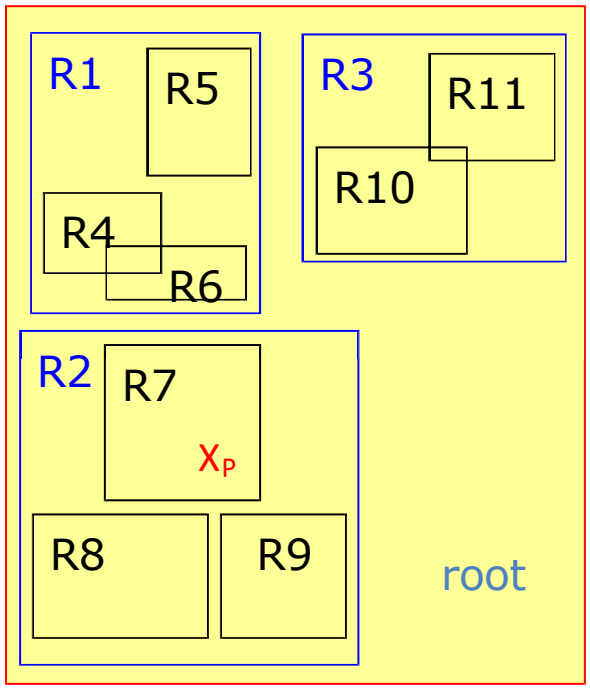
Heuristics

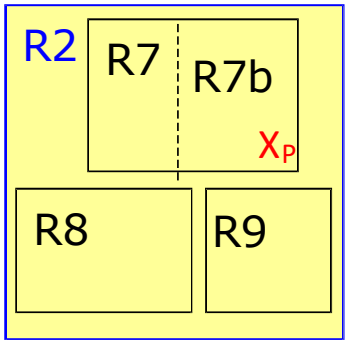
- An object is always inserted in the nodes, to which it produces the smallest increase in volume
- If it falls in the interior of a MBR no enlargement is need
- If there are several possible nodes, then select the one with the **smallest volume**

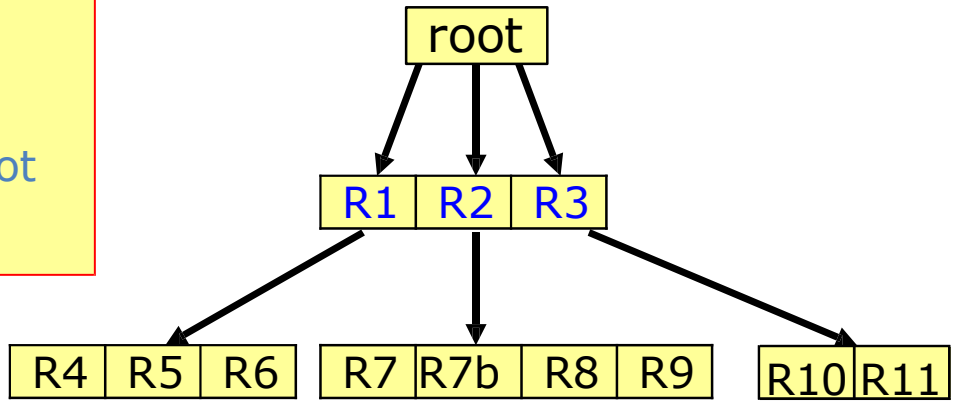
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

Insert with Overflow

Overflow because M=3







Suppose R7 already have 3 elements.

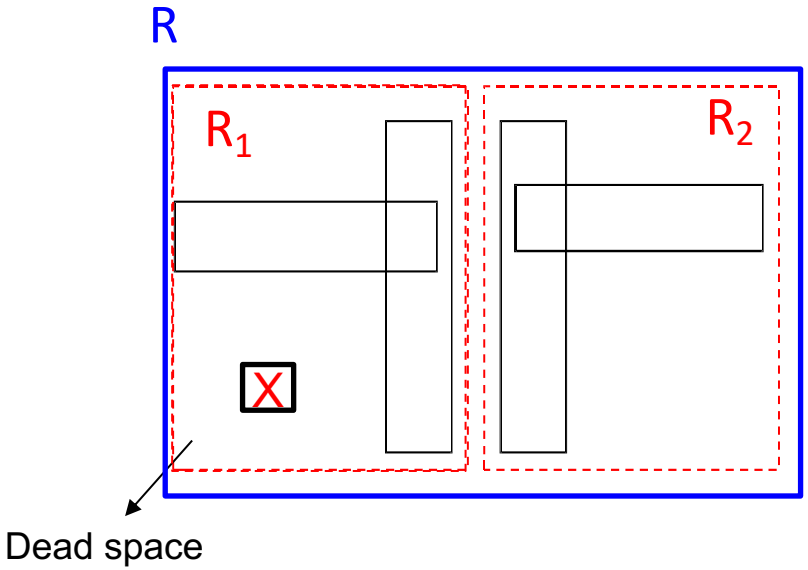
Acknowledgment - Thanks to Wolf-Tilo Balke and Silviu Homoceanu - TU Braunschweig for the slides

SplitNode

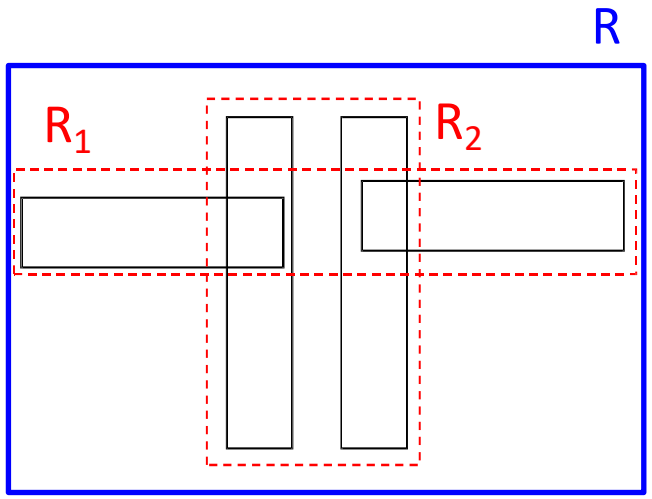
- If an object is inserted in a full node, then the $M+1$ objects will be divided among two new nodes
- The goal in splitting is that it should rarely be needed to traverse both resulting nodes on subsequent searches
  - Therefore use small MBRs. This leads to minimal overlapping with other MBRs

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

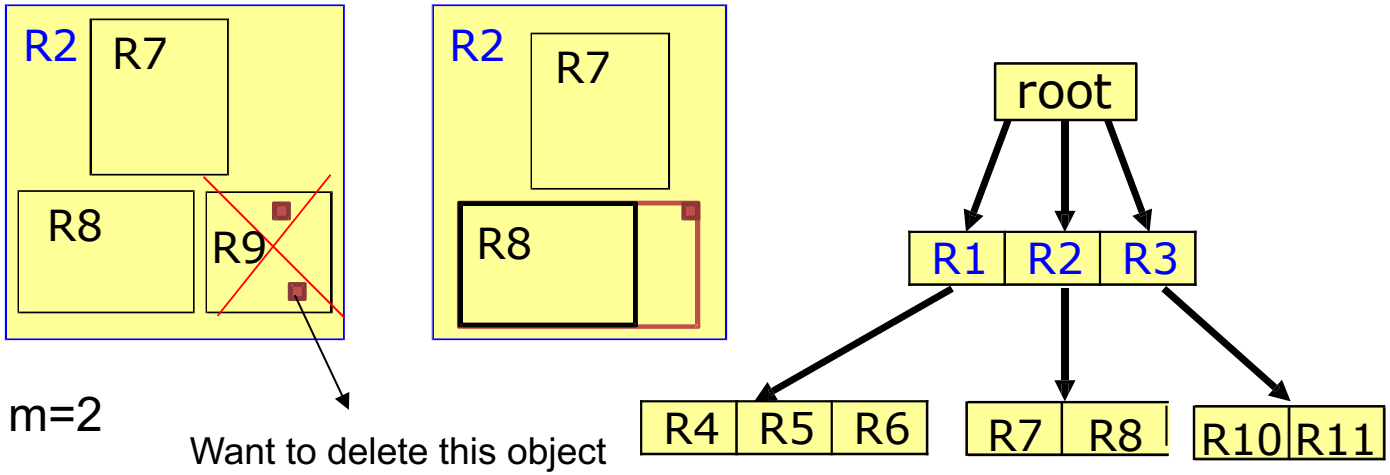
Delete

- 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

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

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)

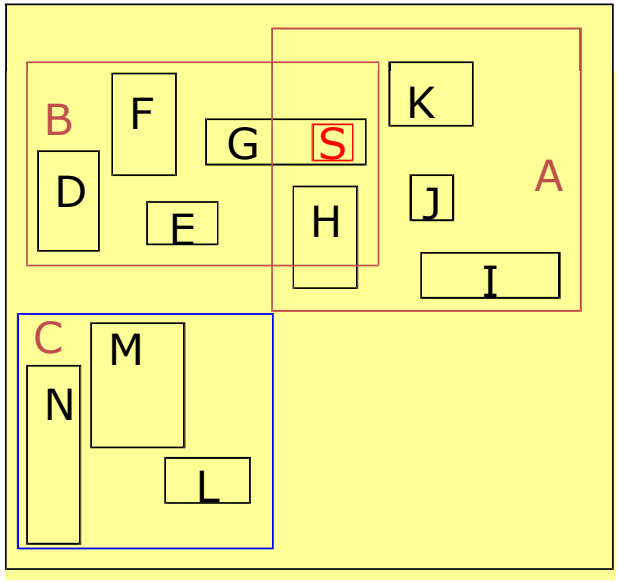


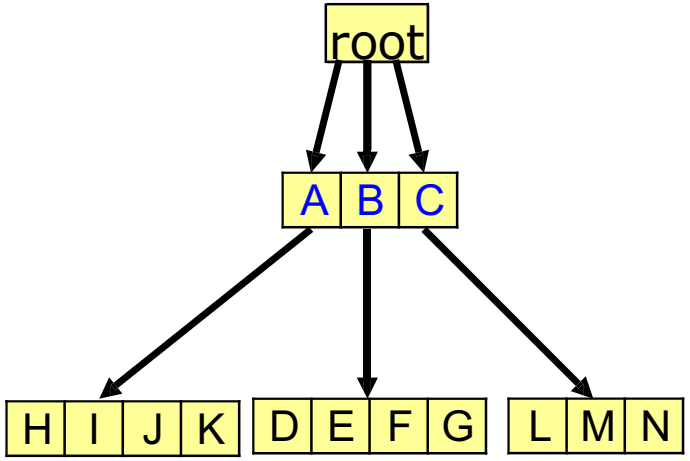
Update

- 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

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

Next Lecture

- Index continued