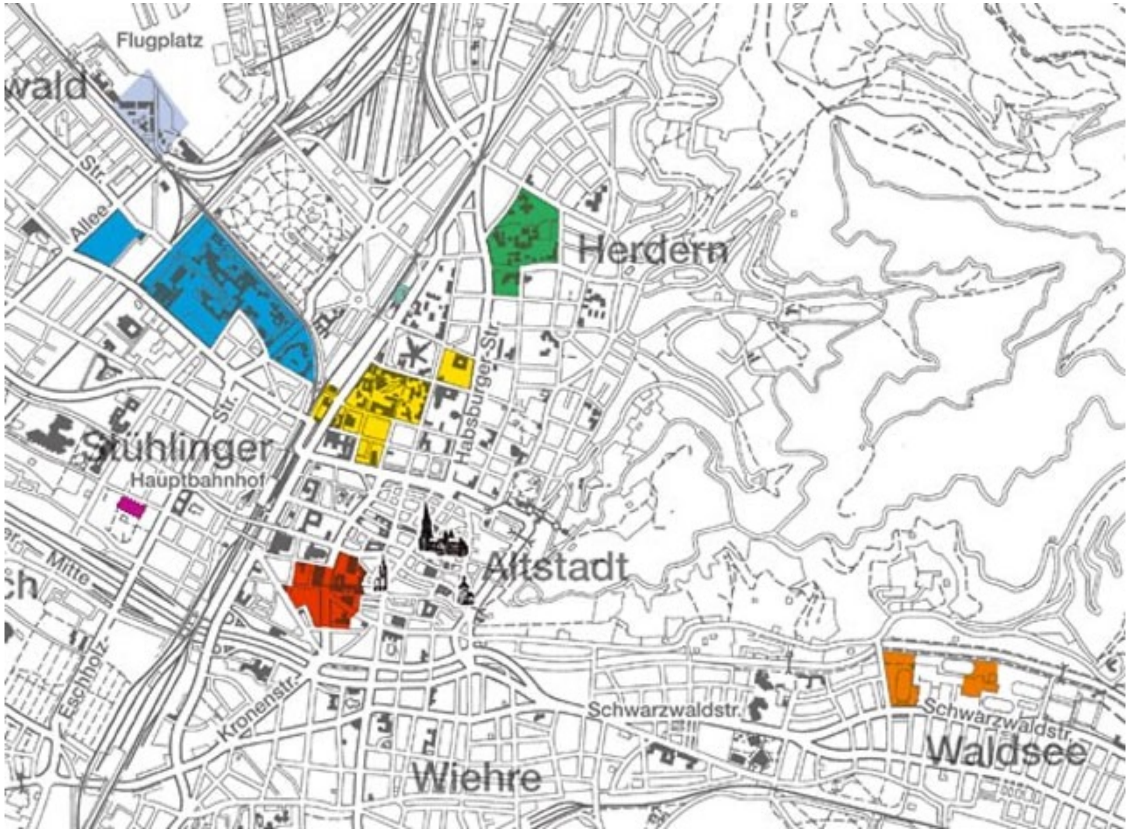
R-Trees

Outline

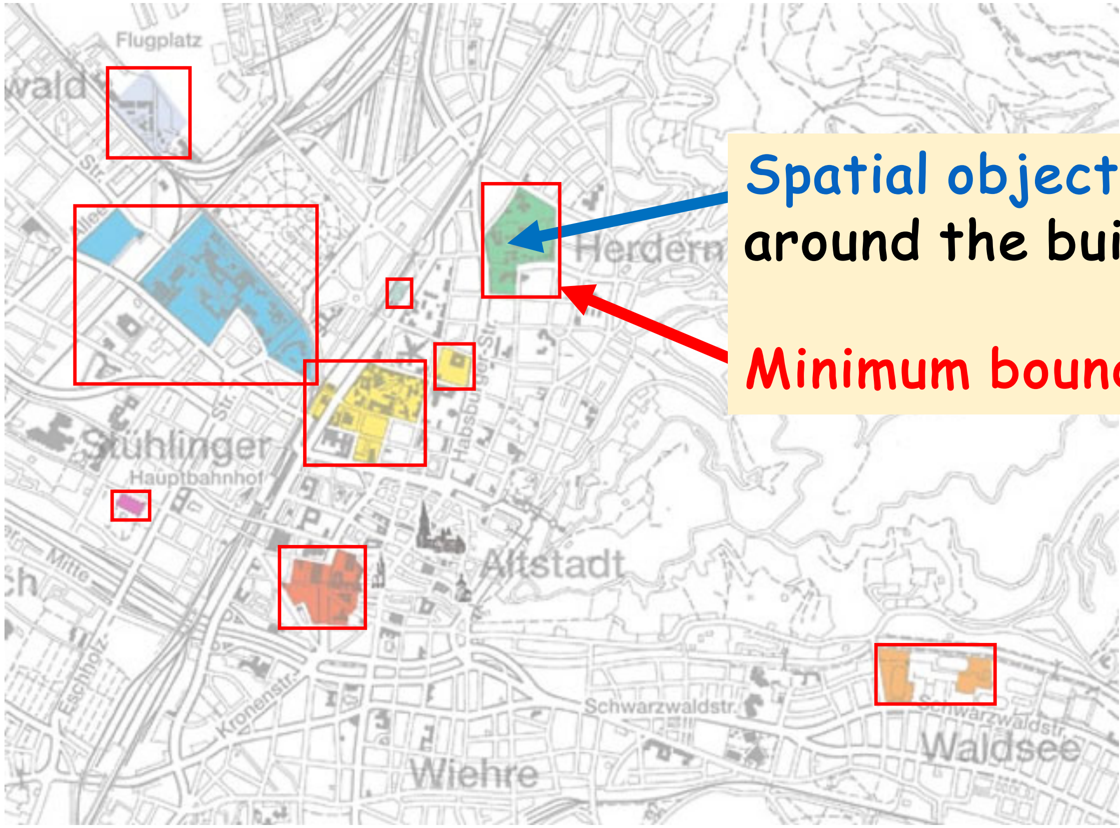
- Spatial data
- Shall have balanced-tree (B-tree) knowledge
- R-tree data structure
- Unique issues in R-tree (vs B-tree)
  - Set of continuous data items (vs a single data item)
  - Representing continuous data and optimizing their placements

Spatial Database



Given a city map, "index" all university buildings in an efficient structure for quick topological search.

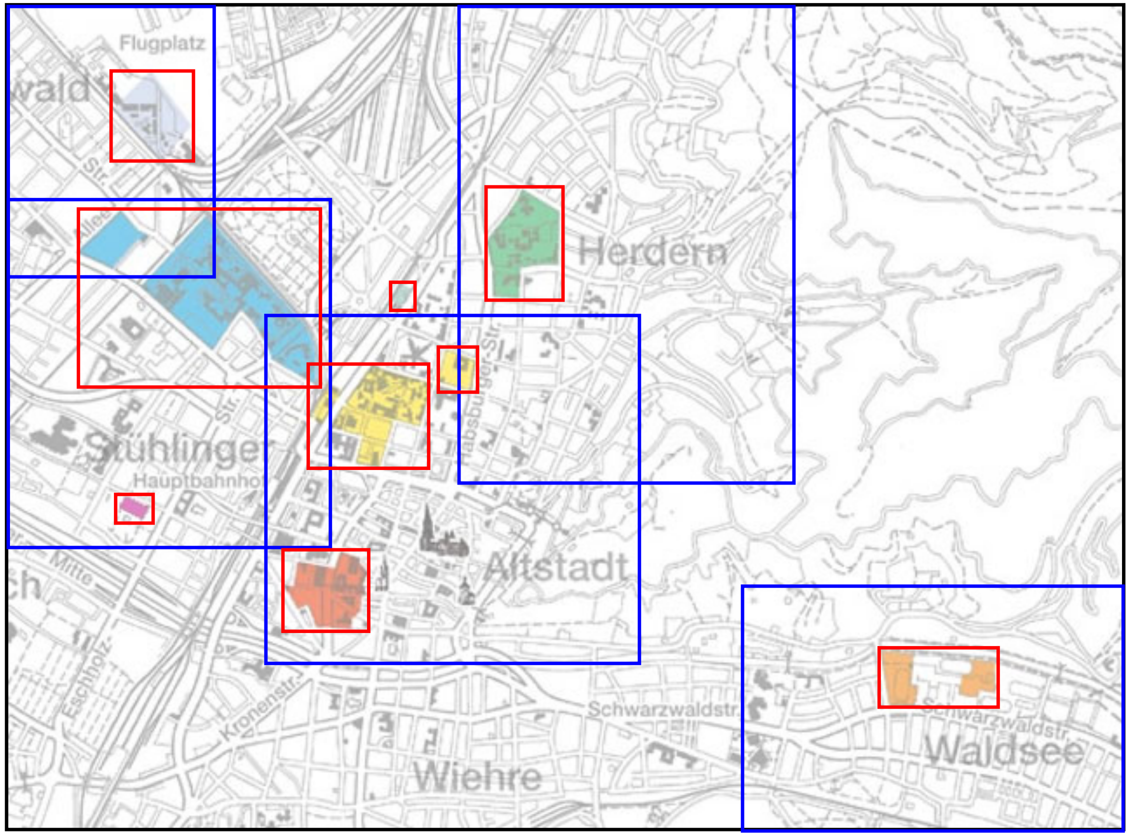
Spatial Database (cont'd)



Spatial object: Contour (outline) of the area around the building(s).

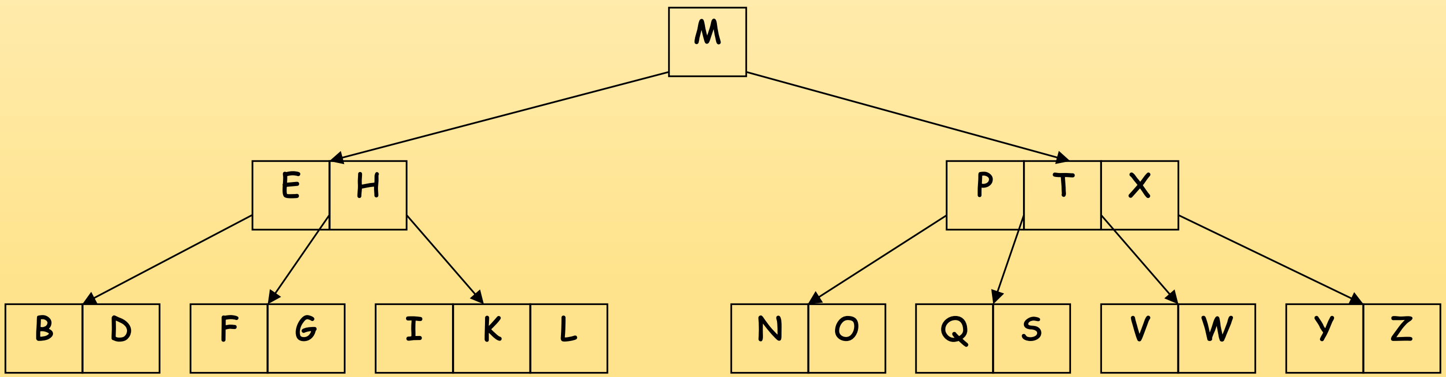
Minimum bounding region (MBR) of the object.

Spatial Database (cont'd)



Recall: B-tree

- A B-Tree is an ordered, dynamic, multi-way structure of order *m* (i.e. each node has at most *m* children).
- The keys and the subtrees are arranged in the fashion of a search tree.



Implementation in B-Tree

- Insert
  - Combine
  - Split

- Delete
  - Rotate
  - Combine
  - Split

The R-Tree Index Structure

- An R-Tree is a height-balanced tree.
  - Leaves in the structure all appear on the same level.
- Index records in the leaf nodes contain pointers to the actual spatial-objects they represent. (akin to B+-tree)
- A spatial database consists of a collection of tuples representing spatial objects, known as Entries.
  - Each Entry has a unique identifier that points to one spatial object, and its MBR; i.e. Entry = (MBR, pointer).

value

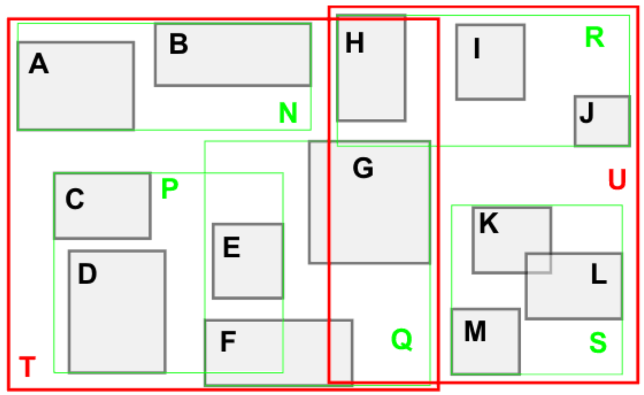
key

R-Tree Index Structure – Leaf *Entries*

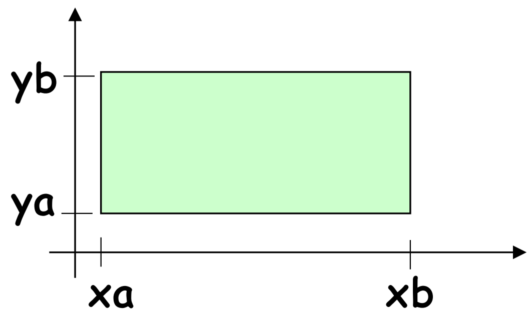
- An entry *E* in a *leaf* node is defined as:

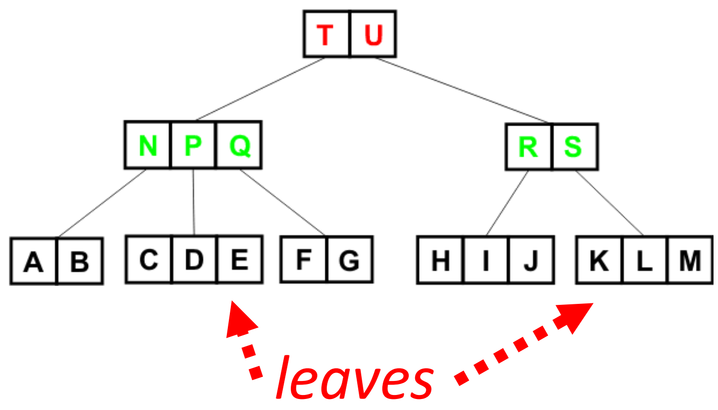
  E = (*I*, *tuple-identifier*)

  - *I* (or key) refers to the smallest binding *n*-dimensional region (MBR) that encompasses the spatial data pointed to by its *tuple-identifier* (or value).
  - *I* is a series of closed-intervals that make up each dimension of the binding region.



- **Example**. In 2D, *I* = (*Ix*, *Iy*), where *Ix* = [*xa*, *xb*], and *Iy* = [*ya*, *yb*].



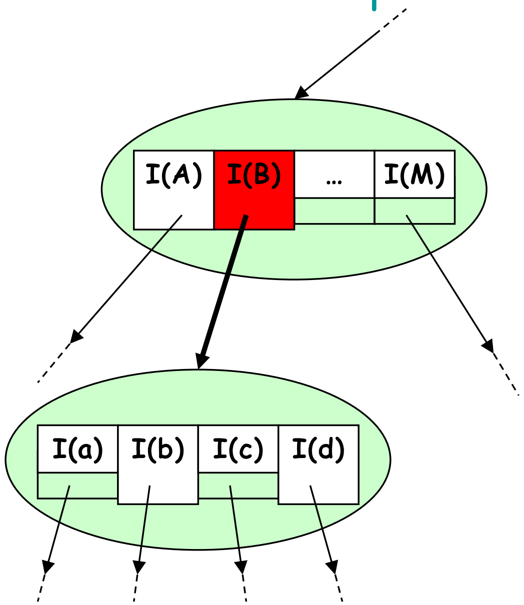


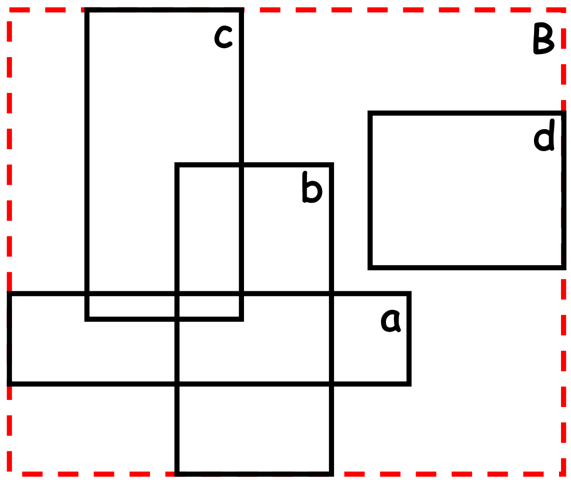
R-Tree Index Structure - Non-Leaf *Entries*

- An entry *E* in a *non-leaf* node is defined as:

$$E = (I, \textit{child-pointer})$$

where the *child-pointer* points to the child of this node, and *I* is the MBR that "encompasses" all the regions in the child-node's pointer's entries.





Properties (B-tree, Essentially)

- Let M be the maximum number of entries that will fit in one node.
- Let $m \leq M/2$ be a parameter specifying the minimum number of entries in one node.
  - Our previous lecture regarding B-tree lets m = ceil[M/2].
- Then an R-Tree must satisfy the following properties:
  1. Every leaf node contains between *m* and *M* index records, unless it is the root.
  2. For each index-record Entry (*I*, *tuple-identifier*) in a leaf node, *I* is the MBR that spatially contains the *n*-dimensional data object represented by the *tuple-identifier*.
  3. Every non-leaf node has between *m* and *M* children, unless it is the root.
  4. For each Entry (*I*, *child-pointer*) in a non-leaf node, *I* is the MBR that spatially contains the regions in the child node.
  5. The root has two children unless it is a leaf.
  6. All leaves appear on the same level.

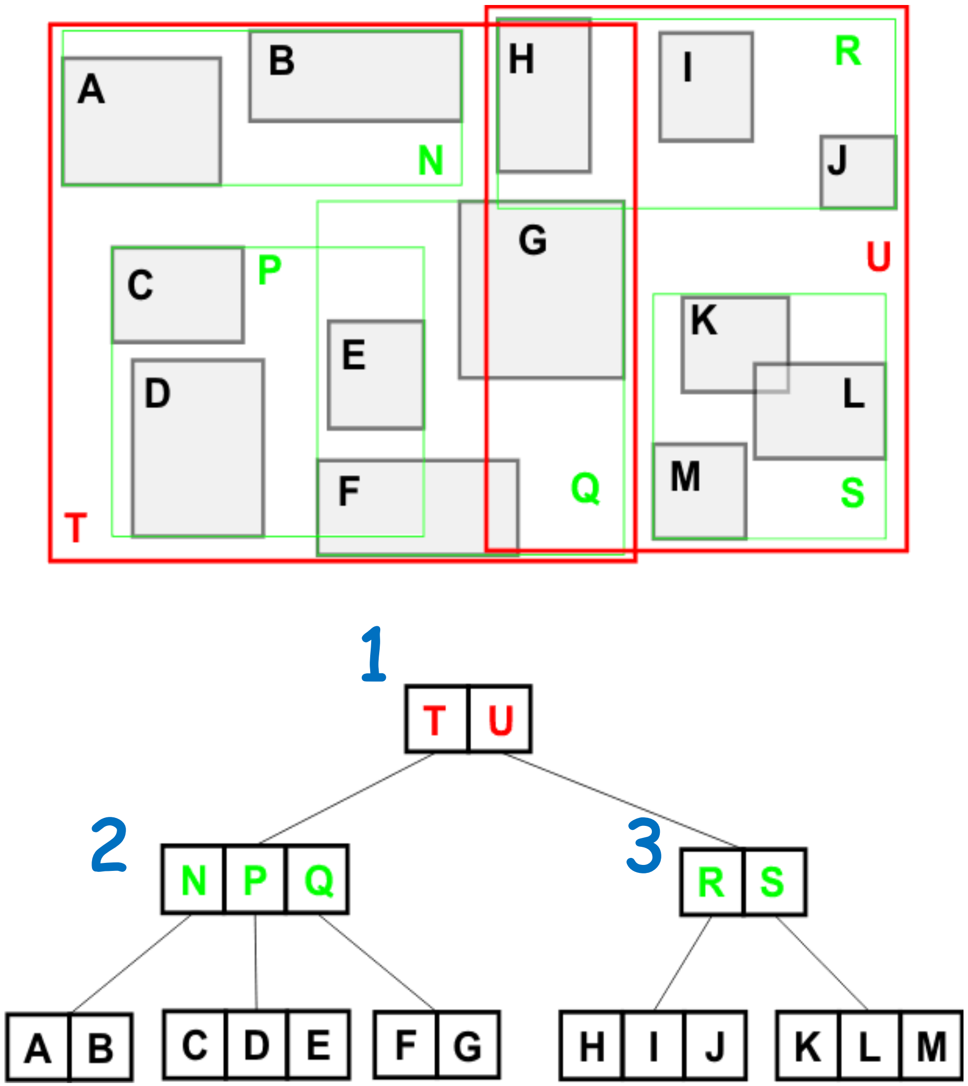
Node Overflow and Underflow (Akin to B-tree)

- Node overflow
  - A Node-Overflow happens when a new Entry is added to a fully packed node, causing the resulting number of entries in the node to exceed the upper-bound *M*.
  - The 'overflow' node must be split, and all its current entries, as well as the new one, consolidated for *local* optimum arrangement.

- Node underflow
  - A Node-Underflow happens when one or more Entries are removed from a node, causing the remaining number of entries in that node to fall below the lower-bound *m*.
  - The underflow node must be condensed, and its entries dispersed for *global* optimum arrangement.

Observations

- MBR's may overlap
  - in a single node
    - 1st level : (T, U) in node 1
    - 2nd level: (P, Q) in node 2
  - in multiple nodes
    - 2nd level: (Q, R) in nodes 2 and 3
    - 一parent node裡的MBR's有overlaps，distinct child nodes裡的MBR's就可能也會發生overlaps (overlaps是會傳遞到descendent nodes)



B-Tree vs R-Tree

- B-tree:
  - A key can only appear in a "single" key segment
  - Key segments are "disjoint"
- R-tree:
  - A key object (MBR) may overlap "multiple" range segments, though the range segments may be disjoint
  - Range segments may be "overlapped", however

Search Strategy in R-Tree

- Let Q be the query region.
- Let T be the root of the R-Tree.
- Search all entry-records whose regions overlaps Q.

- for (;;)
  - **Search sub-trees:**
    - If T is not leaf, then apply **Search** on ever child-node entry E whose I overlaps Q.
  - **Search leaf nodes:**
    - If T is leaf, then check each entry E in the leaf and return E if $E.I$ overlaps Q.

Search Issue

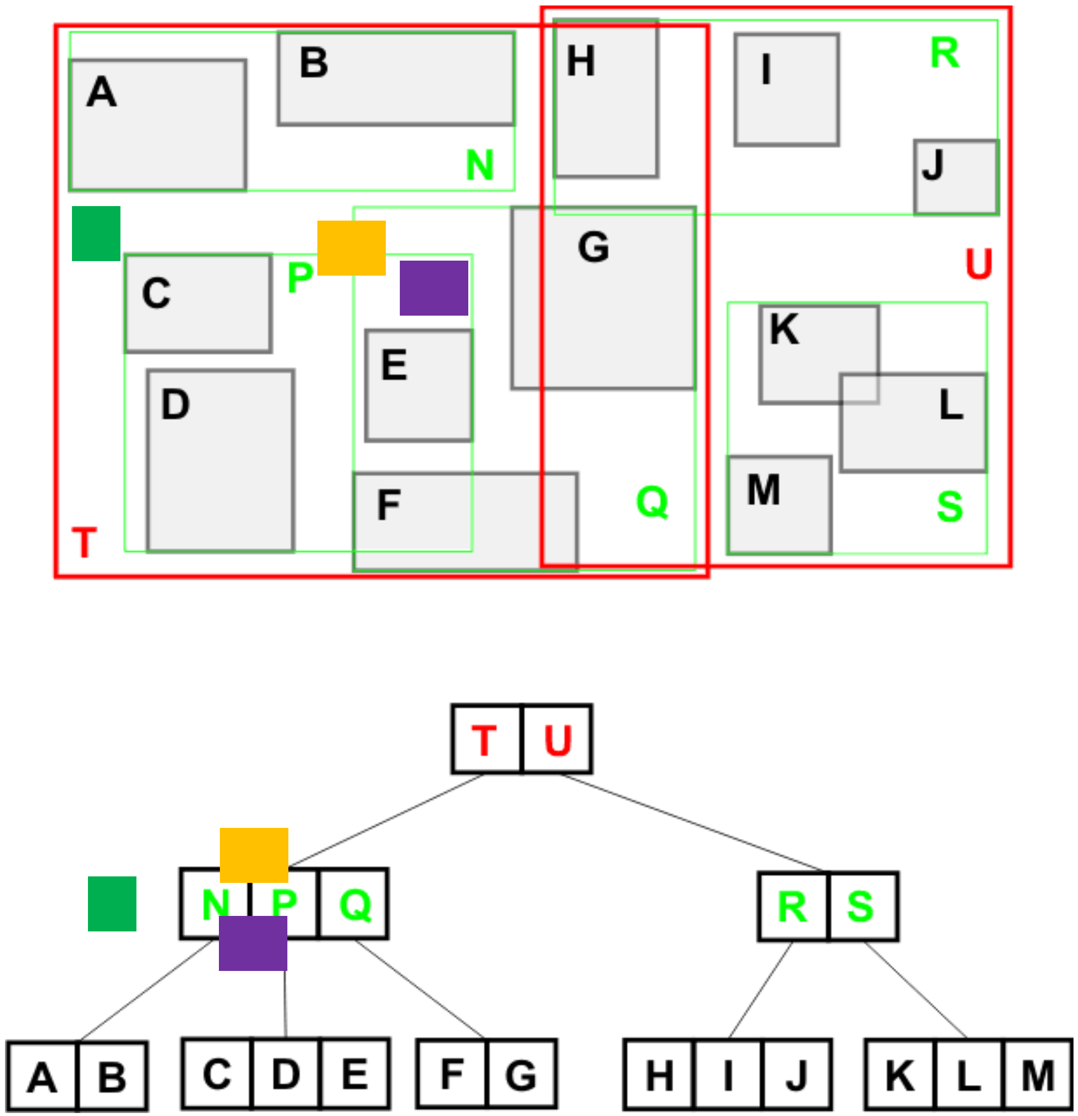
- The search algorithm descends the tree from the root.
- More than one subtree under a node visited may need to be searched.
- *Cannot* guarantee good worst-case performance.
  - Countered by the algorithms during insertion, deletion, and update that maintain the tree in a form that allows the search algorithm to eliminate irrelevant regions of the indexed space.
  - So that only data near the search area need to be examined.
  - Emphasis is on the optimal placement of spatial objects with respect to the spatial location of other objects in the structure.

Insertion in R-Tree

- Let $E = (I, \textit{tuple-identifier})$ be the new entry to be inserted.
- Let T be the root of the R-Tree.
  - Locate a leaf L starting from T to insert E.
  - Add E to L. If L is already full (overflow), split L into L and L'.
  - Propagate MBR changes (enlarged or reduced) upwards.
  - Grow tree taller if node split propagation causes T to split.

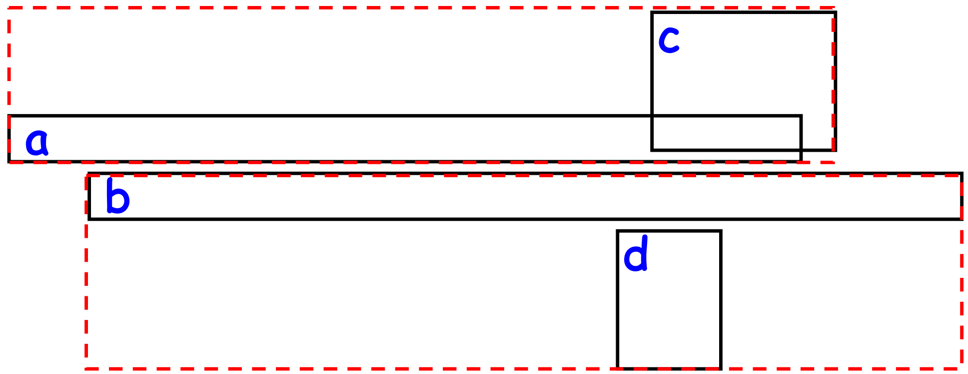
Insertion:

Which ways to Descend?

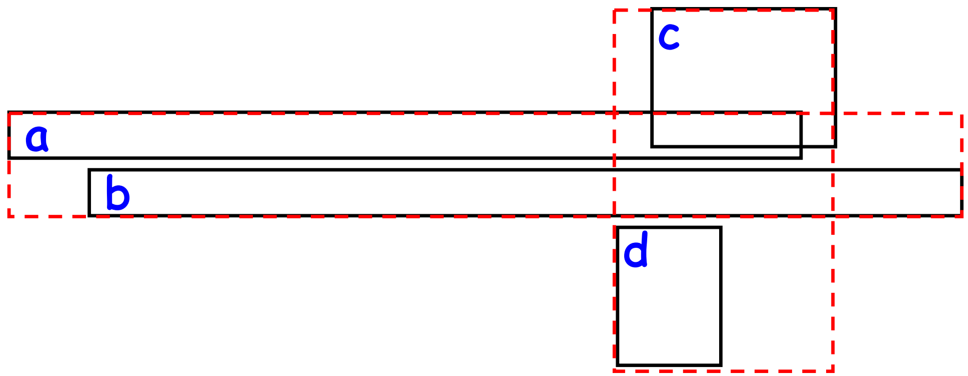


Node Splitting Issue

Bad?



Good?



Derivatives of the R-Tree

- May consider constraints:
  - Overlap: Total area contained within 2 or more MBRs (in a node).
  - Coverage: Total area of all the MBRs of all nodes (in a node).

- Overlap:
  - tighter/smaller MBRs -> more MBR records -> size of R tree increases (bad) and good for search, however
- Coverage:
  - Enlarged MBRs -> sparser in a MBR -> less MBR records -> search overhead increases as more MBR records visited

Variants

- R+-tree
- R*-tree
- ...