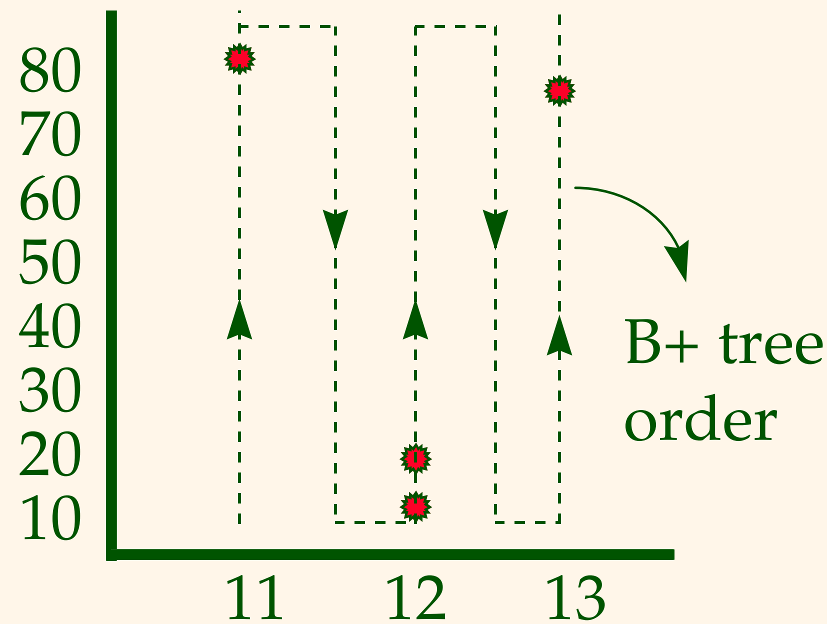
Multidimensional Indexing
The R Tree

Module 7, Lecture 1

Single-Dimensional Indexes

- B+ trees are fundamentally single-dimensional indexes.
- When we create a composite search key B+ tree, e.g., an index on <age, sal>, we effectively linearize the 2-dimensional space since we sort entries first by age and then by sal.

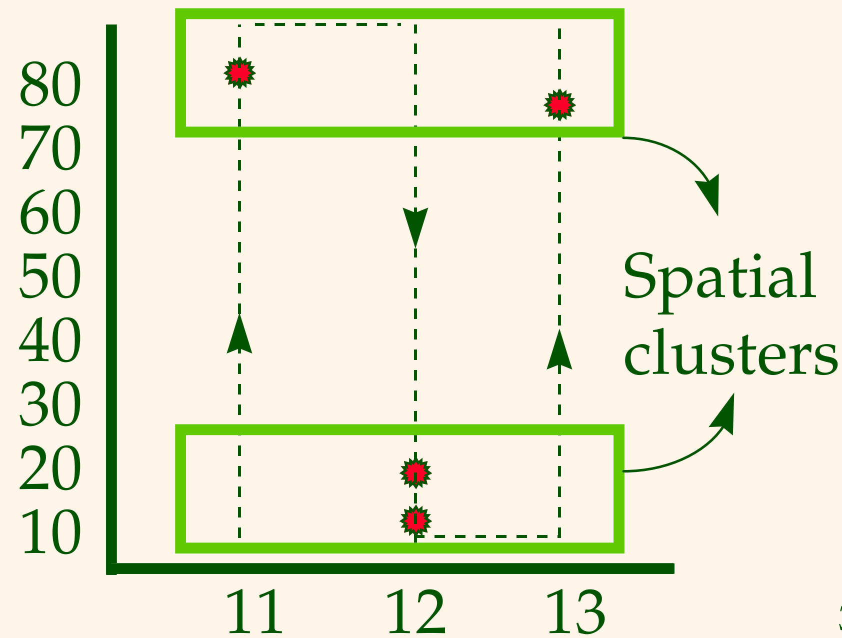
Consider entries:
<11, 80>, <12, 10>
<12, 20>, <13, 75>



Multidimensional Indexes

- A multidimensional index clusters entries so as to exploit “nearness” in multidimensional space.
- Keeping track of entries and maintaining a balanced index structure presents a challenge!

Consider entries:
<11, 80>, <12, 10>
<12, 20>, <13, 75>



Motivation for Multidimensional Indexes

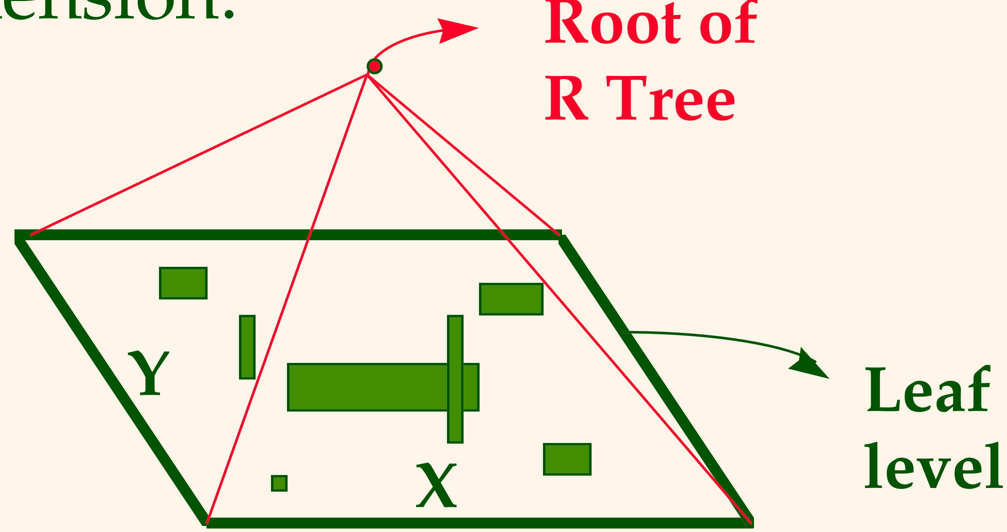
- Spatial queries (GIS, CAD).
  - Find all hotels within a radius of 5 miles from the conference venue.
  - Find the city with population 500,000 or more that is nearest to Kalamazoo, MI.
  - Find all cities that lie on the Nile in Egypt.
  - Find all parts that touch the fuselage (in a plane design).
- Similarity queries (content-based retrieval).
  - Given a face, find the five most similar faces.
- Multidimensional range queries.
  - 50 < age < 55 AND 80K < sal < 90K

What's the difficulty?

- An index based on spatial location needed.
  - One-dimensional indexes don't support multidimensional searching efficiently. (Why?)
  - Hash indexes only support point queries; want to support range queries as well.
  - Must support inserts and deletes gracefully.
- Ideally, want to support non-point data as well (e.g., lines, shapes).
- The R-tree meets these requirements, and variants are widely used today.

The R-Tree

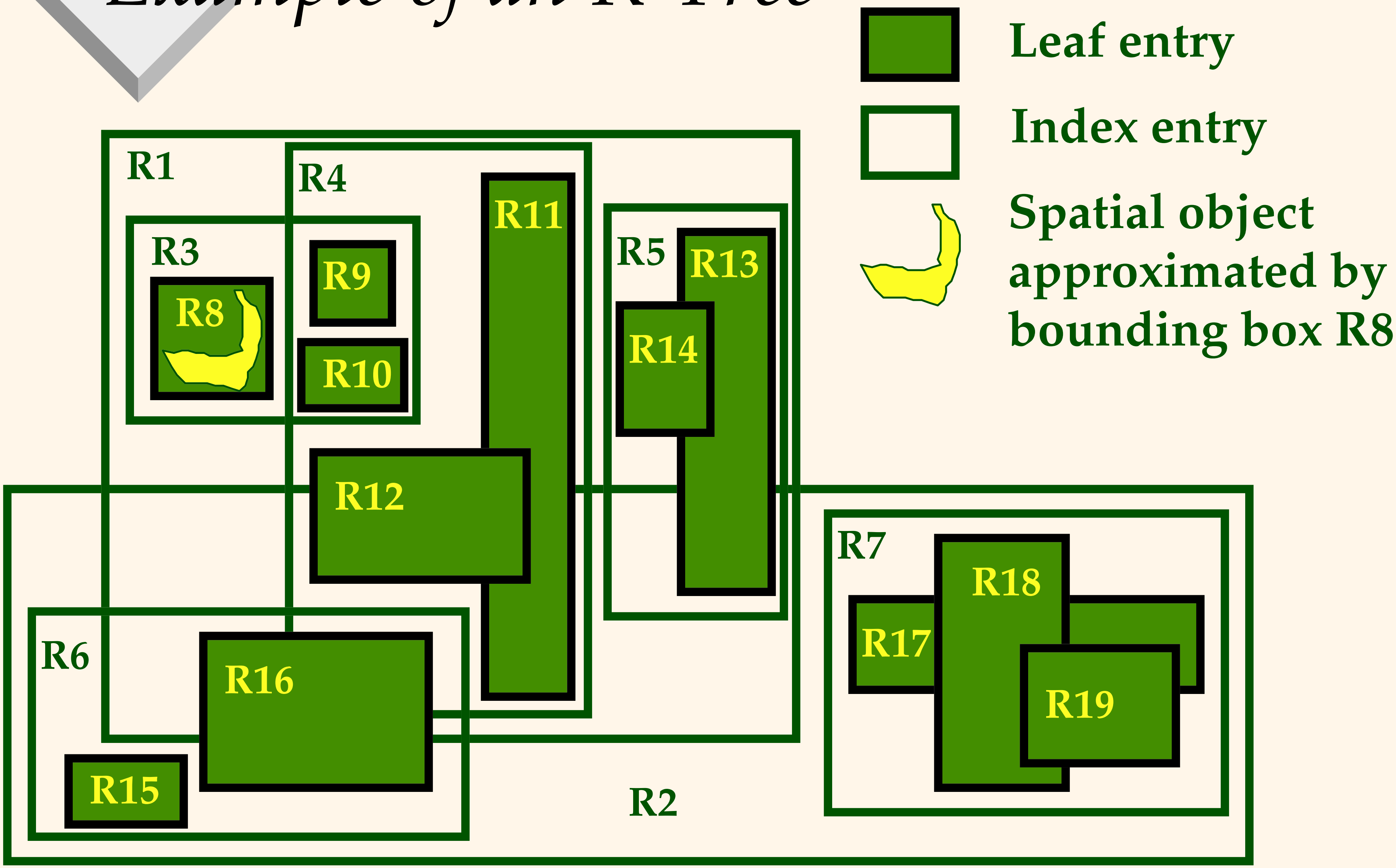
- The R-tree is a tree-structured index that remains balanced on inserts and deletes.
- Each key stored in a leaf entry is intuitively a box, or collection of intervals, with one interval per dimension.
- Example in 2-D:



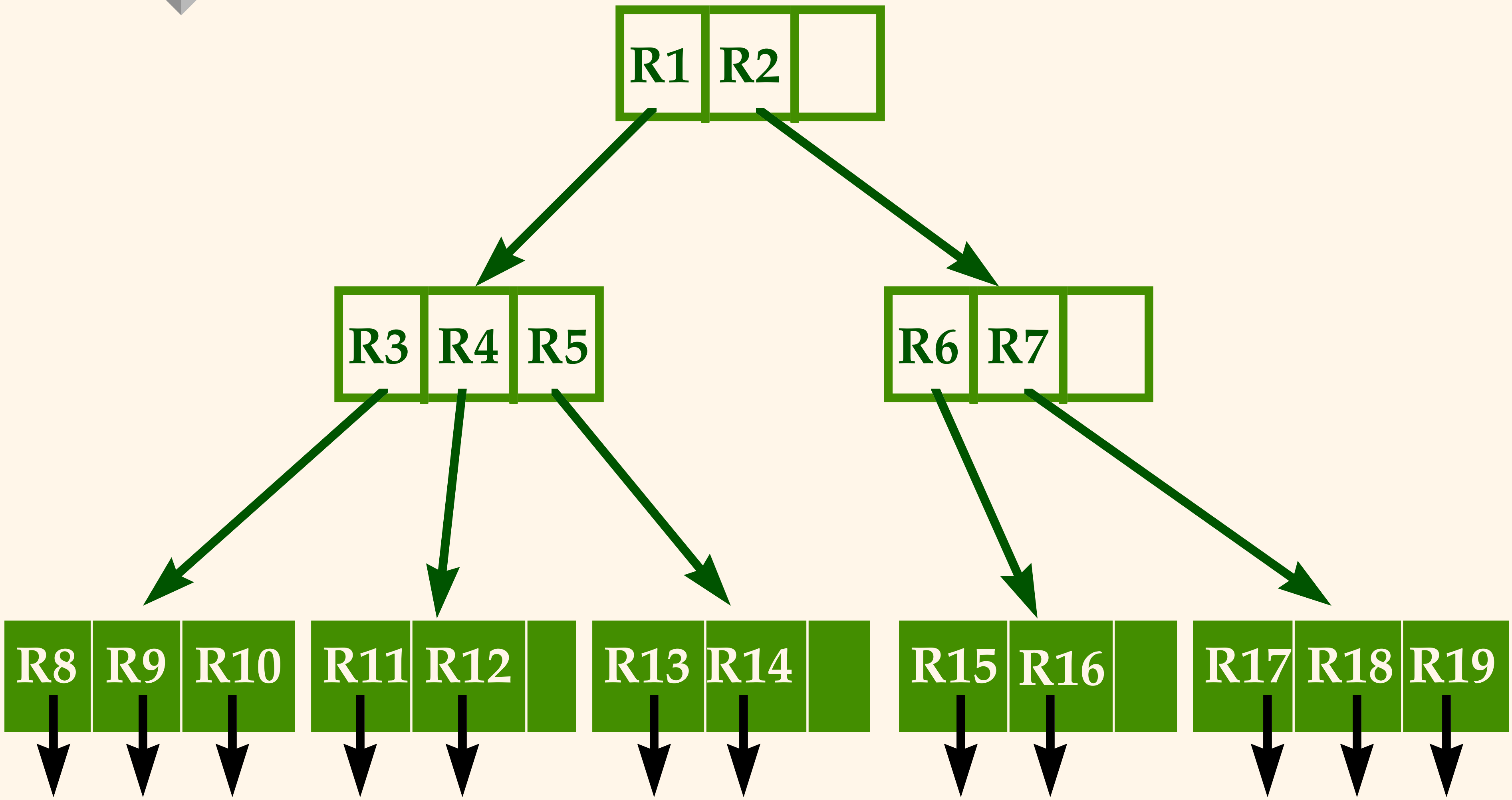
R-Tree Properties

- Leaf entry = < n-dimensional box, rid >
  - This is Alternative (2), with *key value* being a box.
  - Box is the tightest bounding box for a data object.
- Non-leaf entry = < n-dim box, ptr to child node >
  - Box covers all boxes in child node (in fact, subtree).
- All leaves at same distance from root.
- Nodes can be kept 50% full (except root).
  - Can choose a parameter *m* that is <= 50%, and ensure that every node is at least *m*% full.

Example of an R-Tree



Example R-Tree (Contd.)



Search for Objects Overlapping Box Q

Start at root.

1. If current node is non-leaf, for each entry <E, ptr>, if ***box*** E overlaps Q, search subtree identified by ptr.
2. If current node is leaf, for each entry <E, rid>, if E overlaps Q, rid identifies an object that might overlap Q.

Note: May have to search several subtrees at each node!
(In contrast, a B-tree equality search goes to just one leaf.)

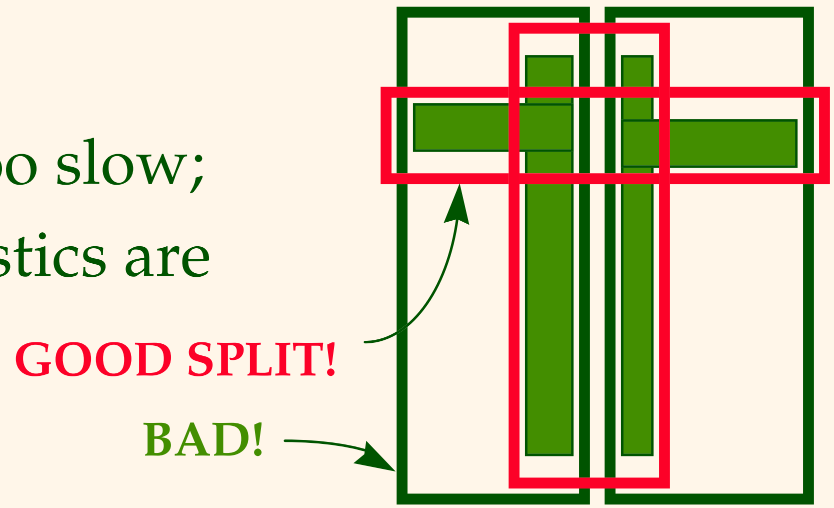
Insert Entry <*B, ptr*>

- Start at root and go down to "best-fit" leaf L.
  - Go to child whose box needs least enlargement to cover B; resolve ties by going to smallest area child.
- If best-fit leaf L has space, insert entry and stop. Otherwise, split L into L1 and L2.
  - Adjust entry for L in its parent so that the box now covers (only) L1.
  - Add an entry (in the parent node of L) for L2. (This could cause the parent node to recursively split.)

Splitting a Node During Insertion

- The entries in node L plus the newly inserted entry must be distributed between L1 and L2.
- Goal is to reduce likelihood of both L1 and L2 being searched on subsequent queries.
- Idea: Redistribute so as to minimize area of L1 plus area of L2.

Exhaustive algorithm is too slow; quadratic and linear heuristics are described in the paper.



Comments on R-Trees

- Deletion consists of searching for the entry to be deleted, removing it, and if the node becomes underfull, deleting the node and then re-inserting the remaining entries.
- Overall, works quite well for 2 and 3 D datasets. Several variants (notably, R+ and R* trees) have been proposed; widely used.
- Can improve search performance by using a convex polygon to approximate query shape (instead of a bounding box) and testing for polygon-box intersection.