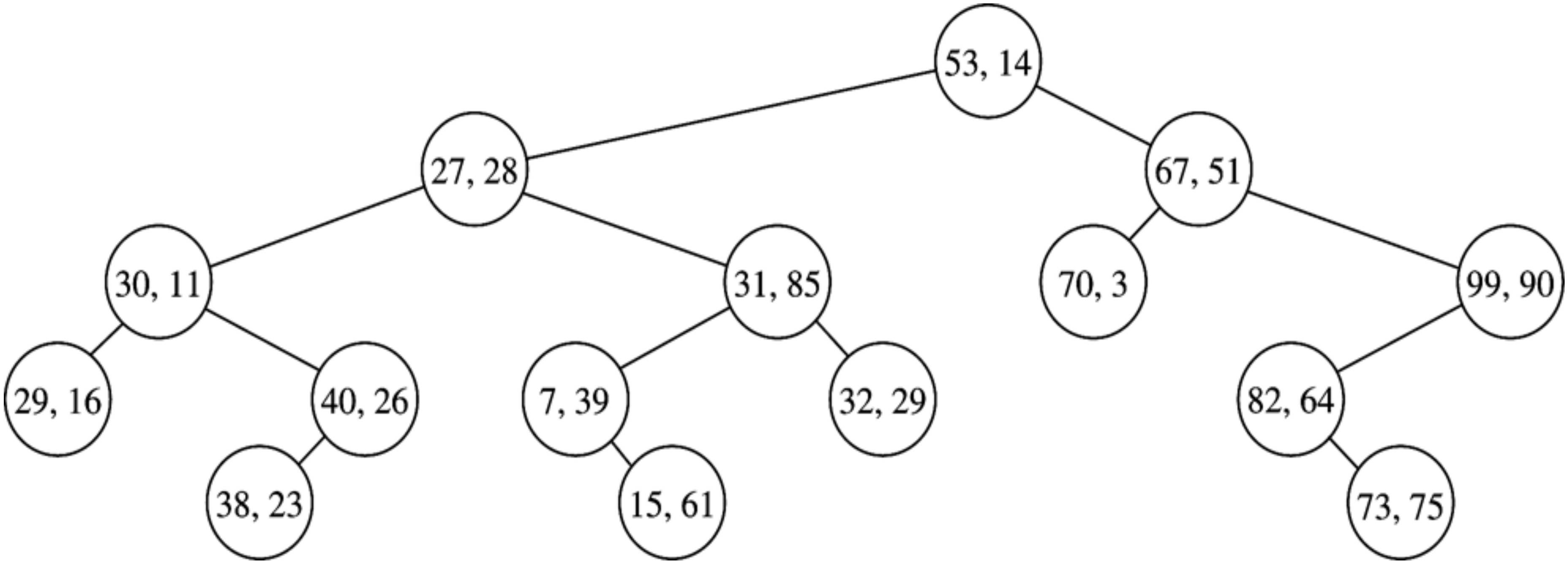
K-D Trees

Based on materials by Dennis Frey, Yun Peng, Jian Chen, Daniel Hood, and Jianping Fan

K-D Tree

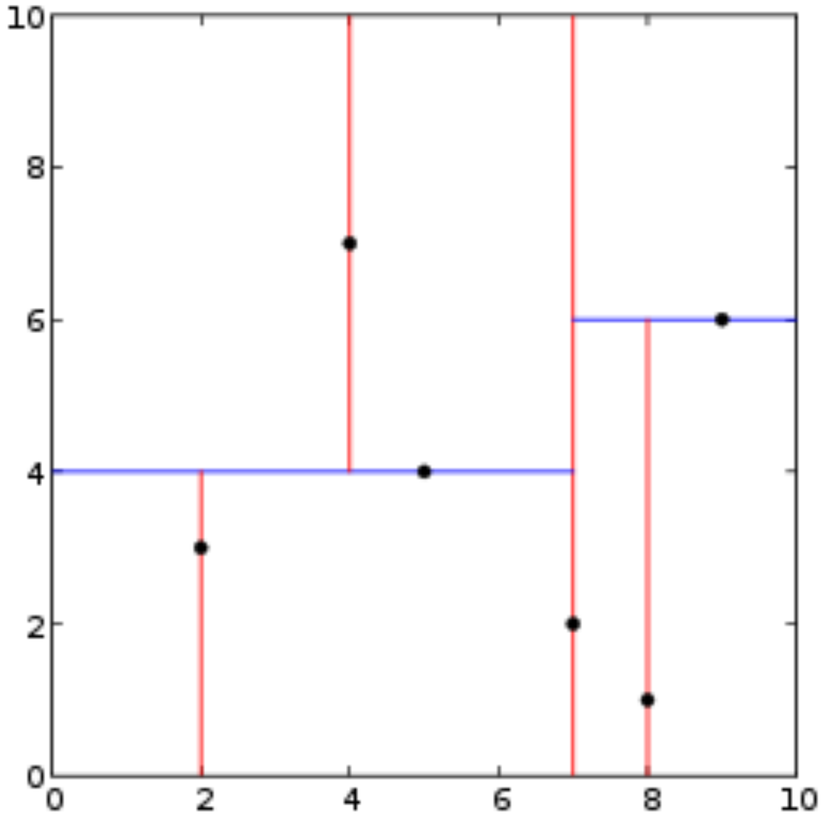
- Introduction
  - Multiple dimensional data
    - Range queries in databases of multiple keys:
      Ex. find persons with
      $34 \le \textit{age} \le 49$ and $\$100\text{k} \le \textit{annual income} \le \$150\text{k}$
    - GIS (geographic information system)
    - Computer graphics
  - Extending BST from one dimensional to k-dimensional
    - **It is a binary tree**
    - Organized by levels (root is at level 0, its children level 1, etc.)
    - Tree branching at level 0 according to the first key, at level 1 according to the second key, etc.
- KdNode
  - Each node has a vector of keys, in addition to the pointers to its subtrees.

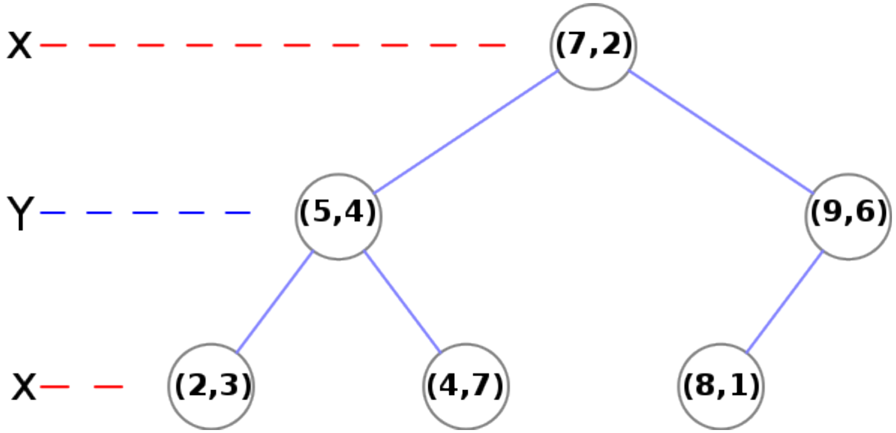
K-D Tree



- A 2-D tree example

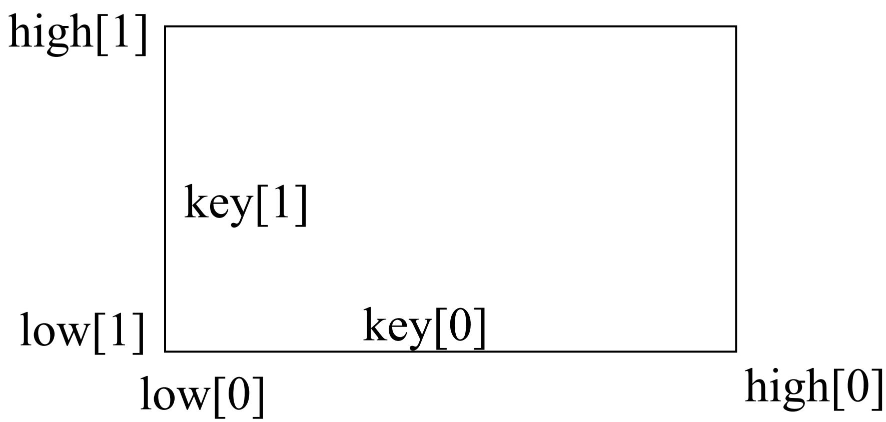
K-D tree decomposition for the point set (2,3), (5,4), (9,6), (4,7), (8,1), (7,2).





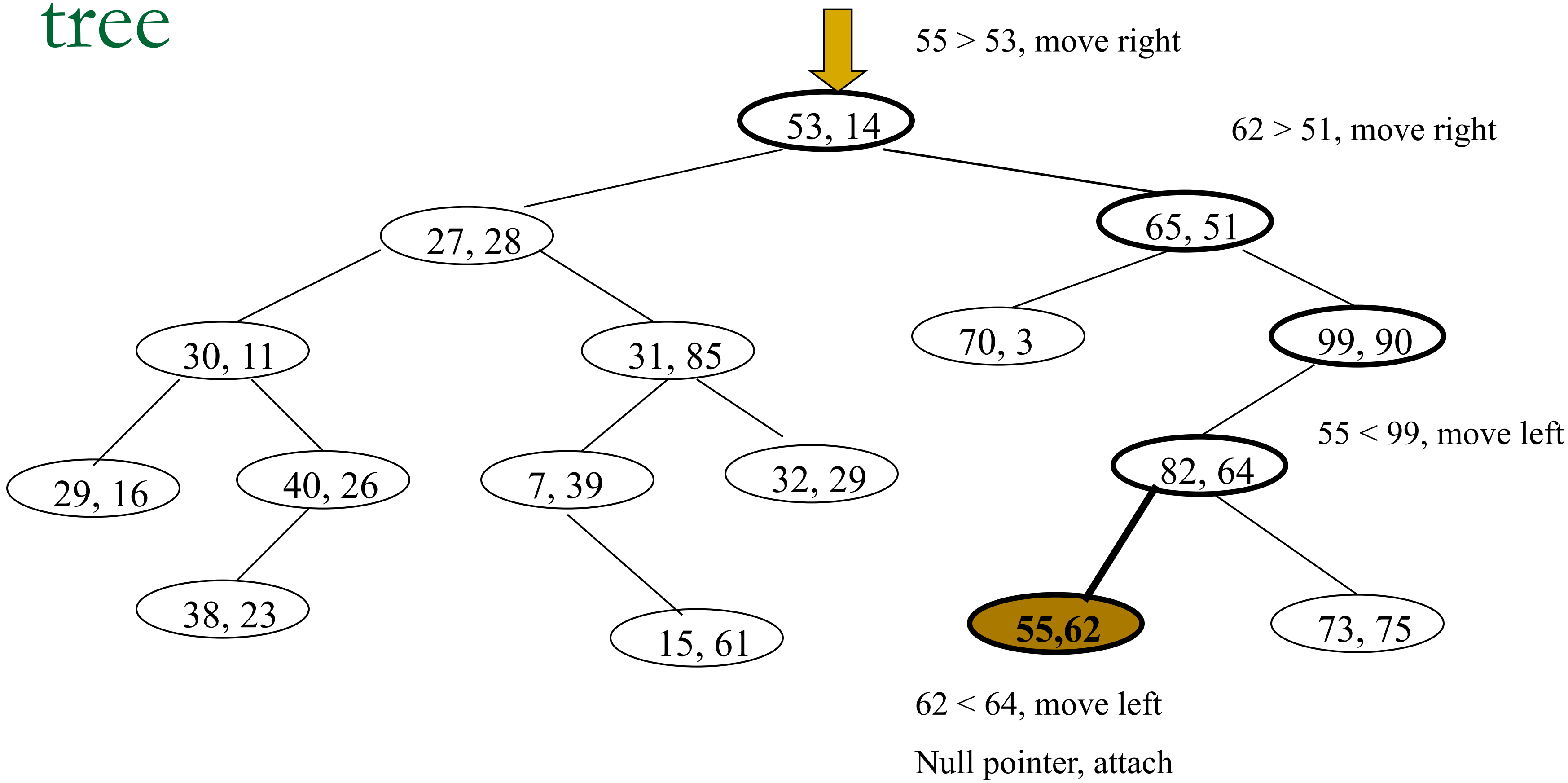
2-D Tree Operations

- Insert
  - A 2-D item (vector of size 2 for the two keys) is inserted
  - New node is inserted as a leaf
  - Different keys are compared at different levels
- Find/print with an orthogonal (rectangular) range



  - exact match: insert (low[level] = high[level] for all levels)
  - partial match: (query ranges are given to only some of the k keys, other keys can be thought in range ± ∞)

Insert (55, 62) into the following 2-D tree

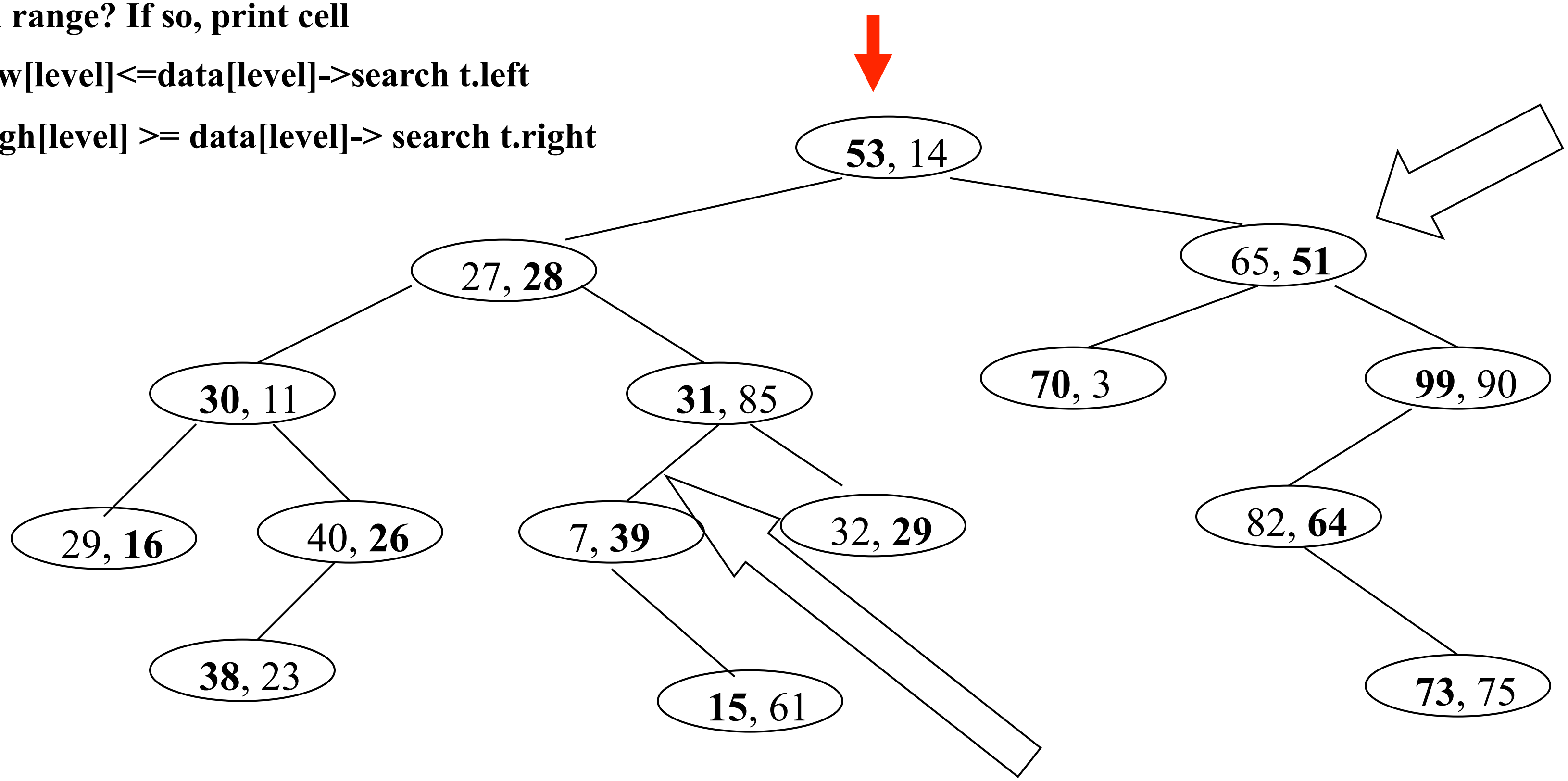


printRange in a 2-D Tree

In range? If so, print cell

low[level]<=data[level]->search t.left

high[level] >= data[level]-> search t.right



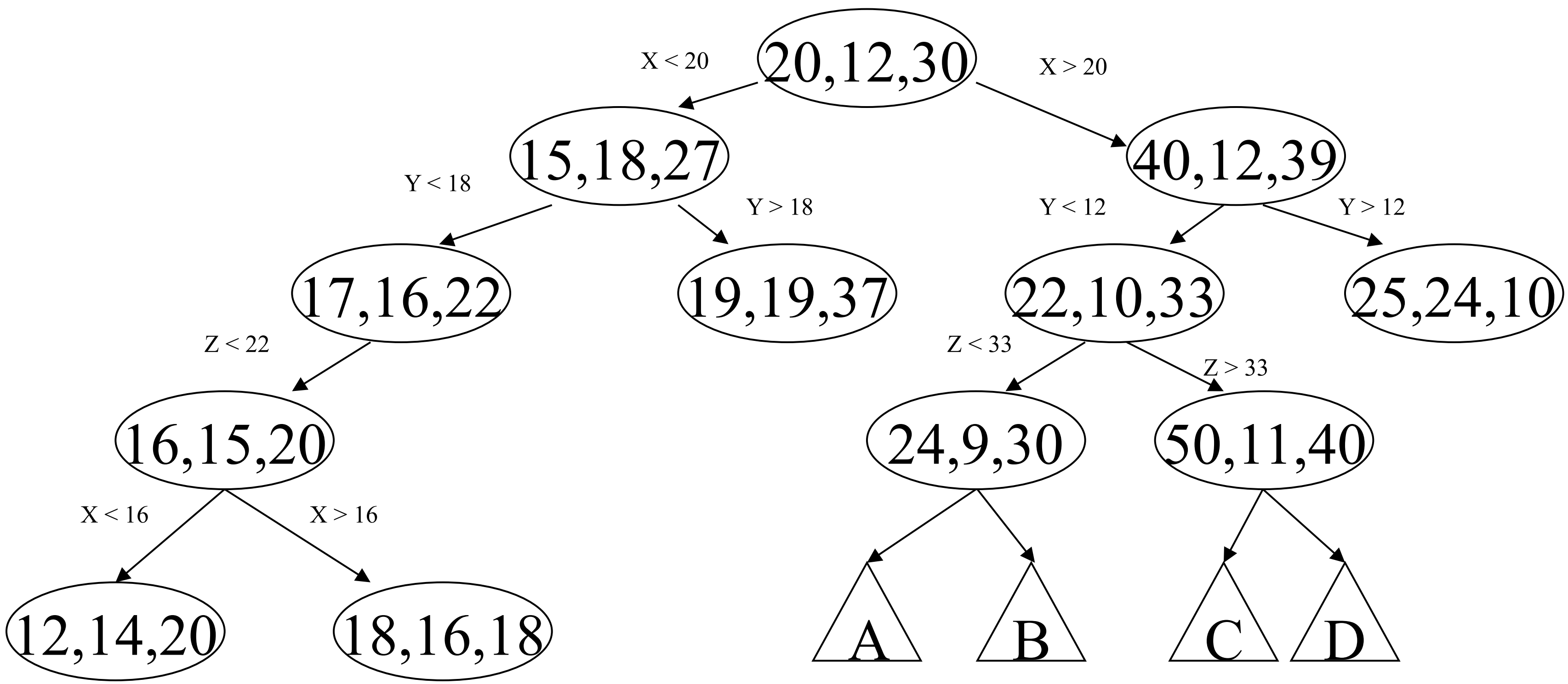
low[0] = 35, high[0] = 40;

low[1] = 23, high[1] = 30;

This sub-tree is never searched.

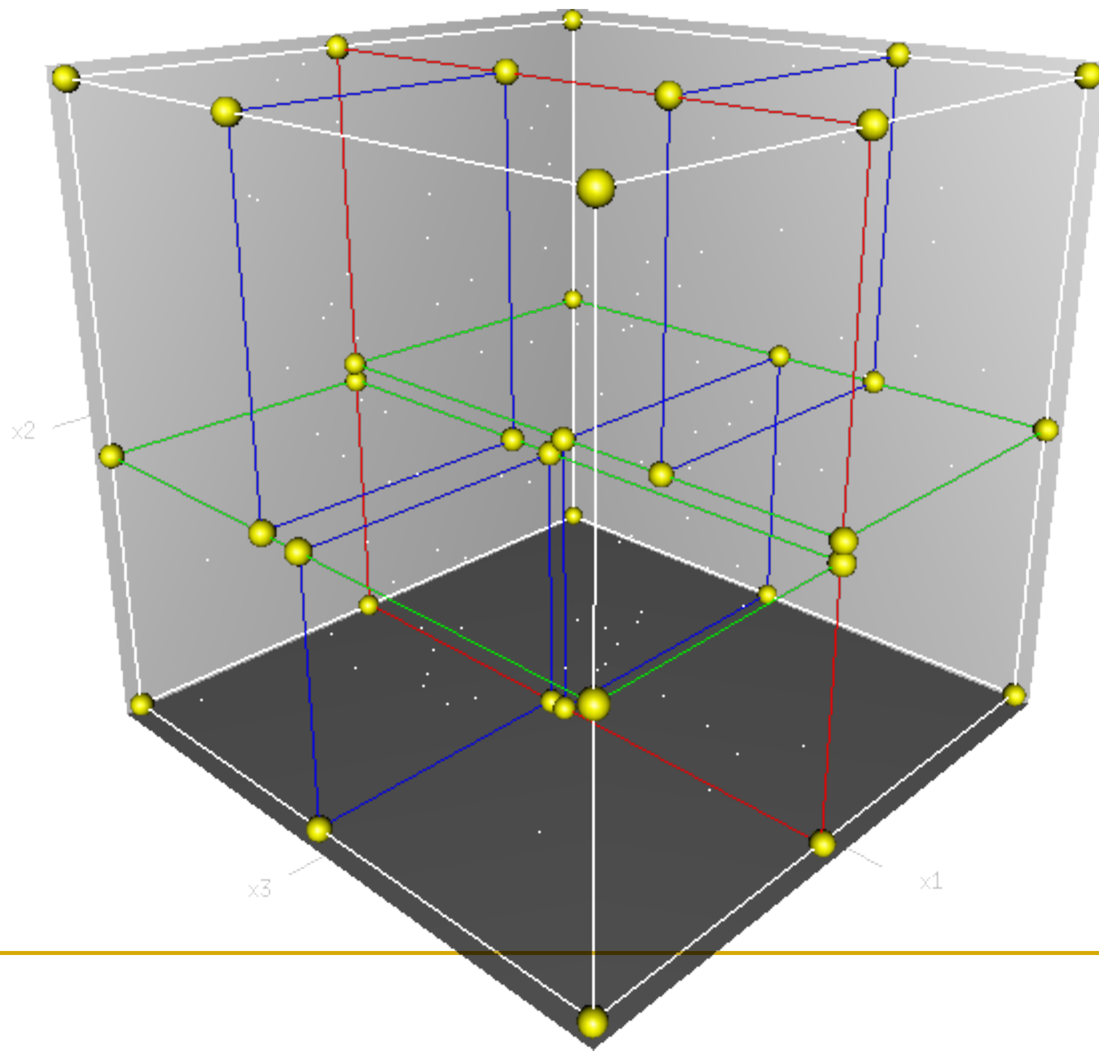
Searching is "preorder". Efficiency is obtained by "pruning" subtrees from the search.

3-D Tree example



What property (or properties) do the nodes in the subtrees labeled A, B, C, and D have?

3-D Tree



K-D Operations

- Modify the 2-D insert code so that it works for K-D trees.
- Modify the 2-D printRange code so that it works for K-D trees.

K-D Tree Performance

- Insert
  - Average and balanced trees: O(lg N)
  - Worst case: O(N)
- Print/search with a square range query
  - Exact match: same as insert (low[level] = high[level] for all levels)
  - Range query: for M matches
    - Perfectly balanced tree:

      K-D trees: $O(M + kN^{(1-1/k)})$

      2-D trees: $O(M + \sqrt{N})$
    - Partial match

      in a random tree: $O(M + N^{\alpha})$ where $\alpha = (-3 + \sqrt{17}) / 2$

K-D Tree Performance

- More on range query in a perfectly balanced 2-D tree:
  - Consider one boundary of the square (say, low[0])
  - Let T(N) be the number of nodes to be looked at with respect to low[0]. For the current node, we may need to look at
    - One of the two children (e.g., node (27, 28)), and
    - Two of the four grand children (e.g., nodes (30, 11) and (31, 85)).
  - Write T(N) = 2 T(N/4) + c, where N/4 is the size of subtrees 2 levels down (we are dealing with a perfectly balanced tree here), and c = 3.
  - Solving this recurrence equation:

$$T(N) = 2T(N/4) + c = 2(2T(N/16) + c) + c$$

$$\ldots$$

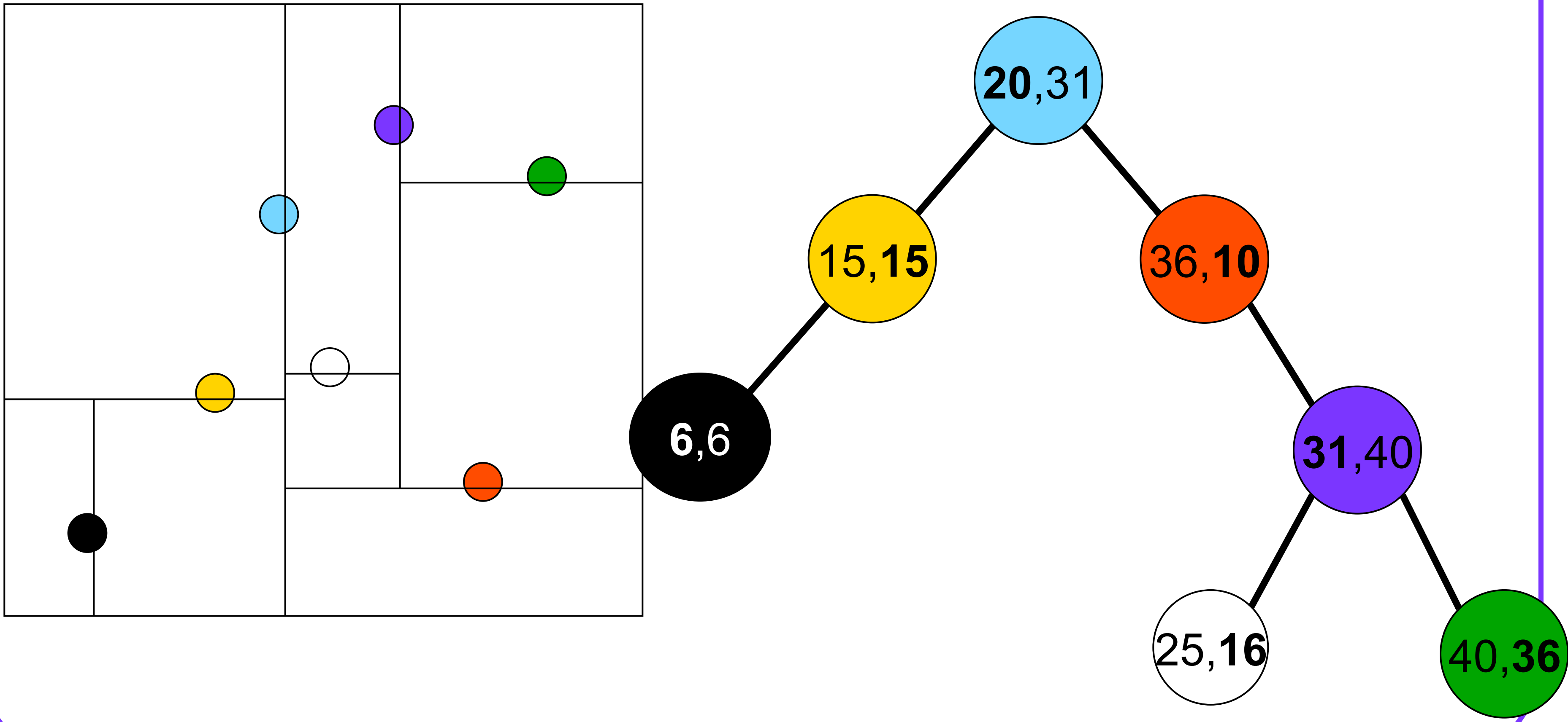
$$= c(1 + 2 + \cdots + 2^{\wedge}(\log_4 N) = 2^{\wedge}(1+ \log_4 N) - 1$$

$$= 2*2^{\wedge}(\log_4 N) - 1 = 2*2^{\wedge} ((\log_2 N)/2) - 1 = O(\sqrt{N})$$

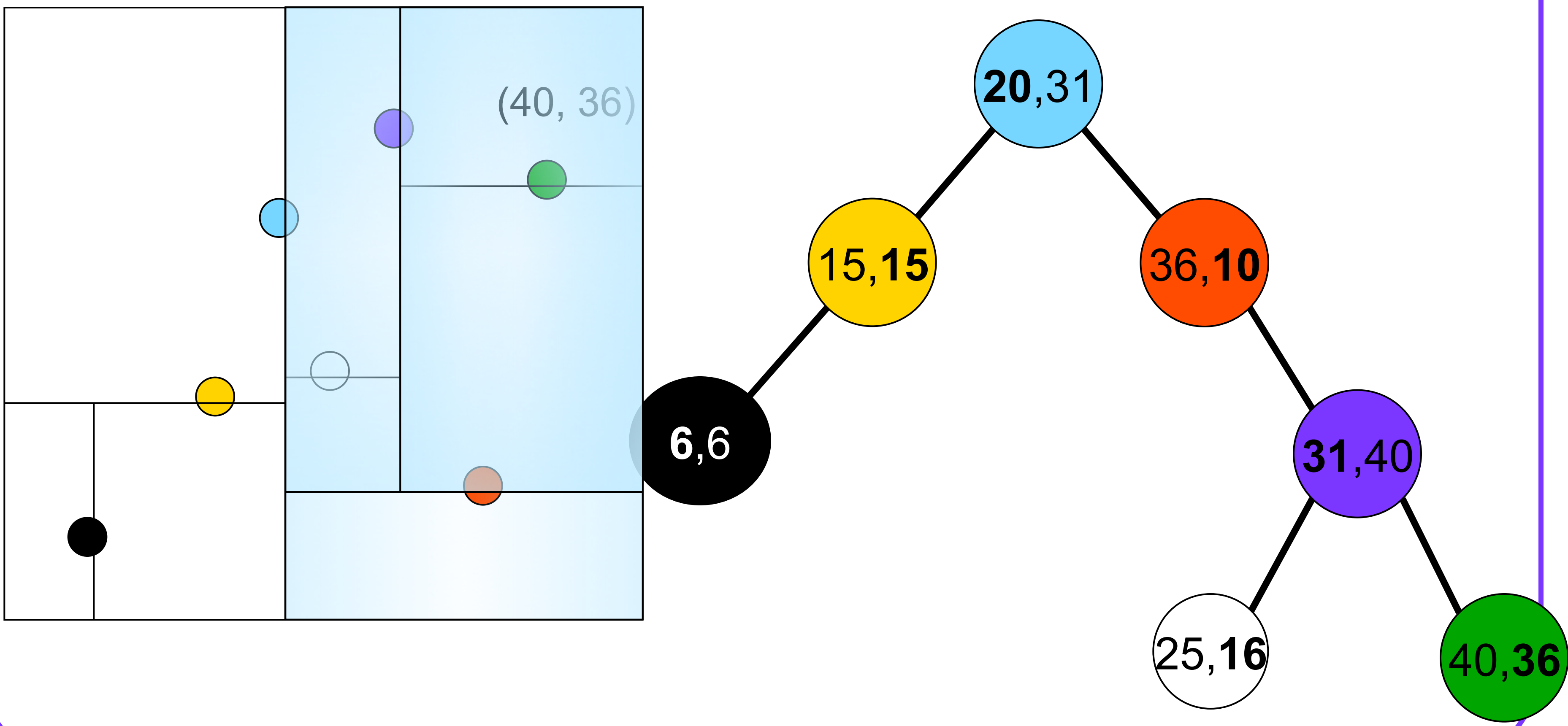
K-D Tree Remarks

- Remove
  - No good remove algorithm beyond lazy deletion
    (mark the node as removed)
- Balancing K-D Tree
  - No known strategy to guarantee a balanced 2-D tree
  - Periodic re-balance
- Extending 2-D tree algorithms to k-D
  - Cycle through the keys at each level

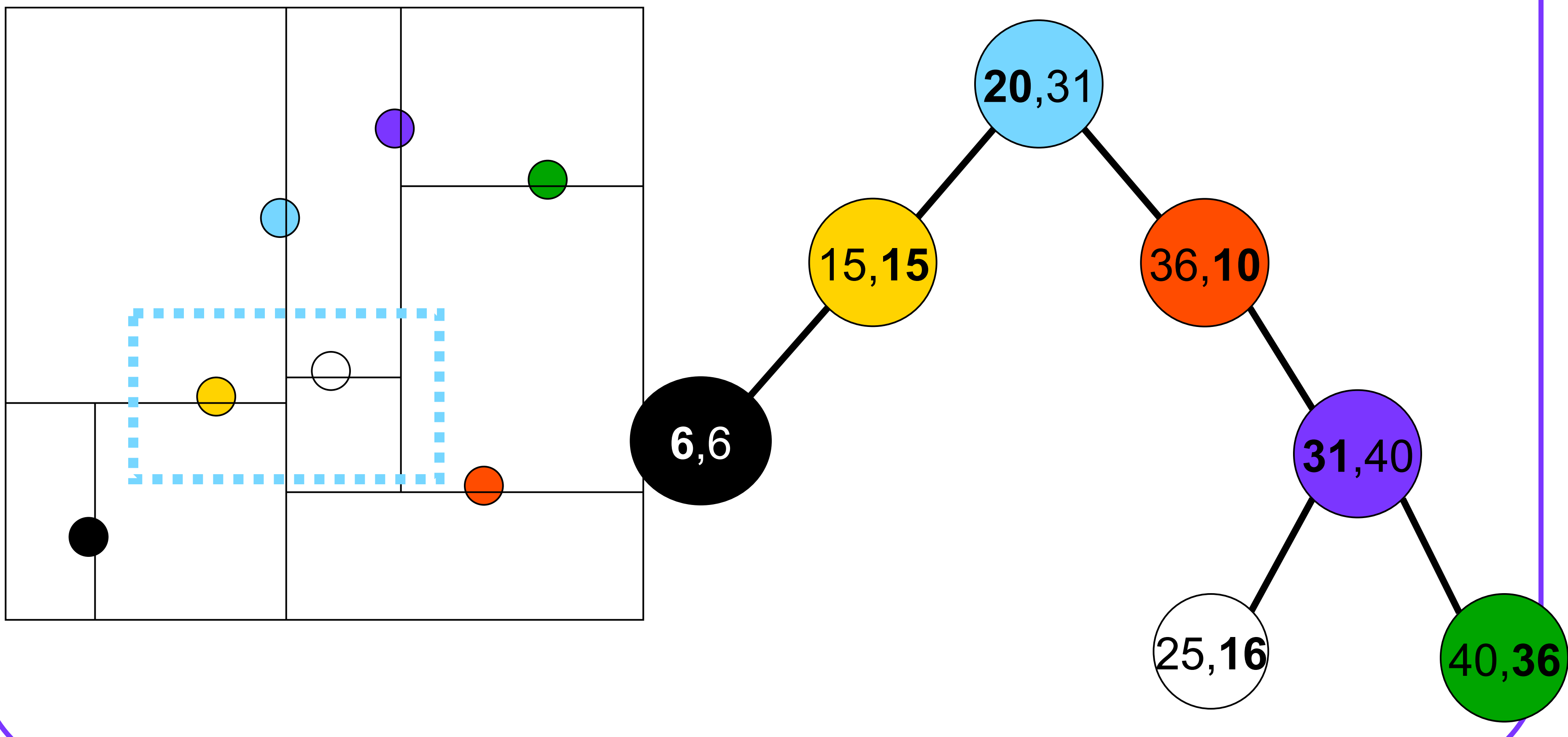
Insertion



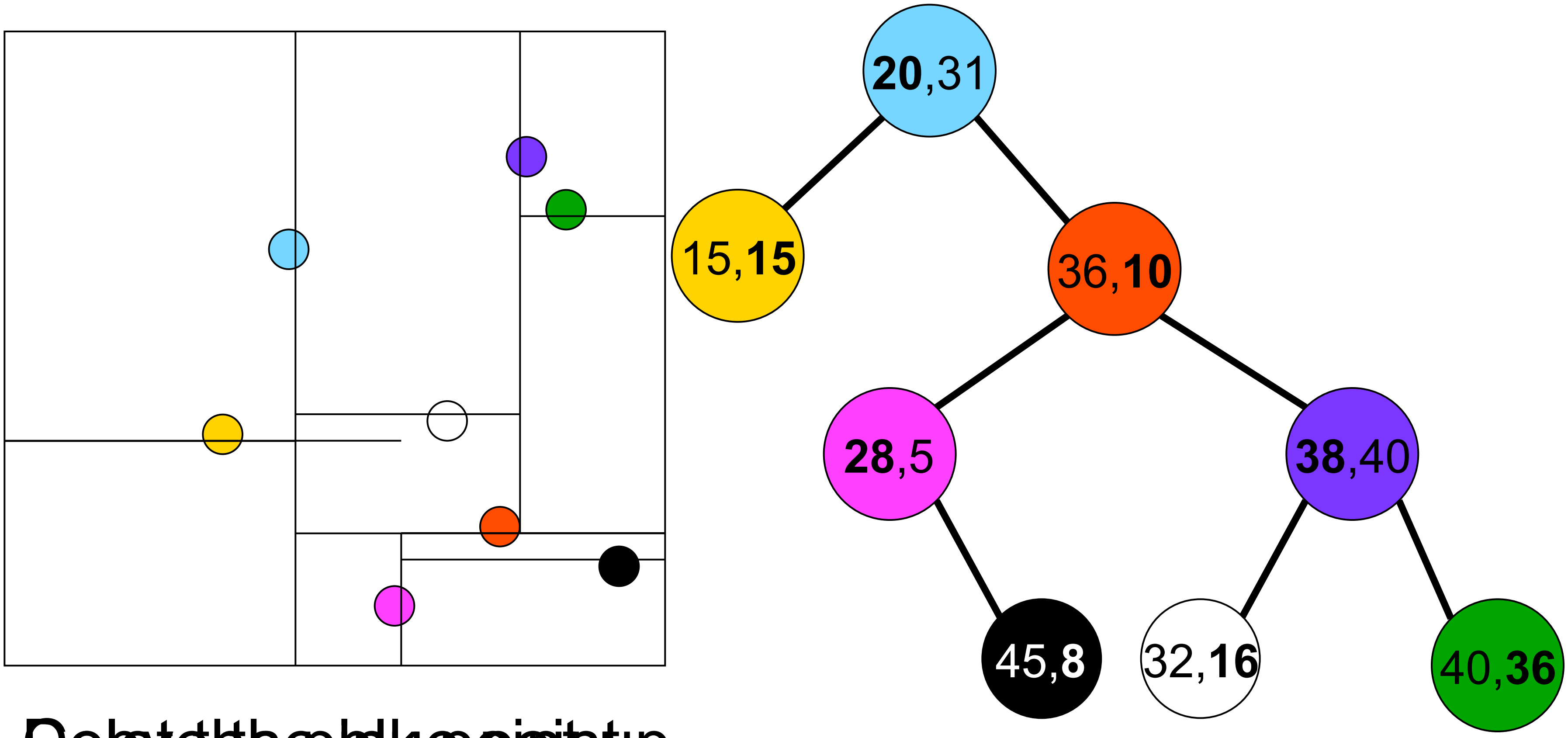
Exact Search



Range search



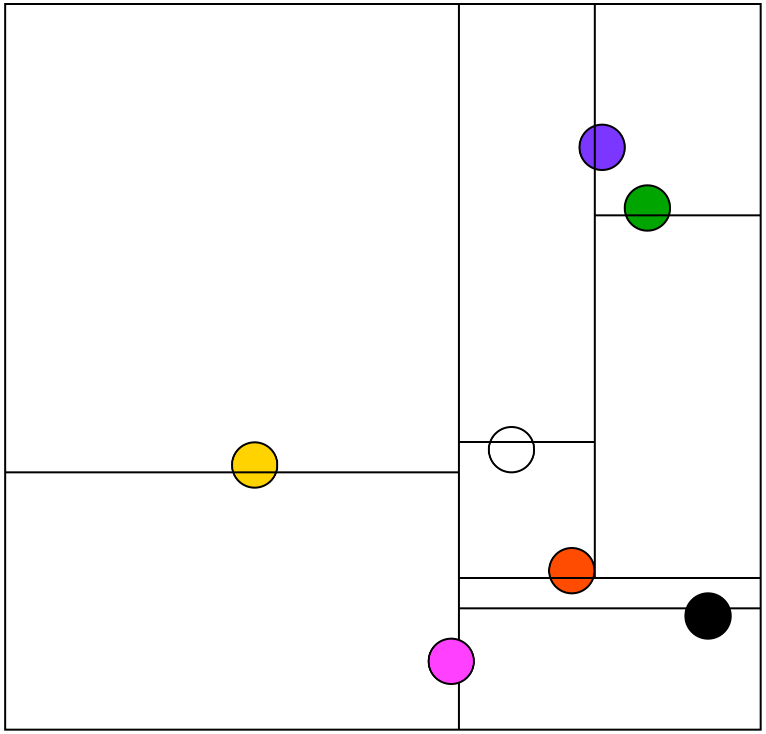
Deletion

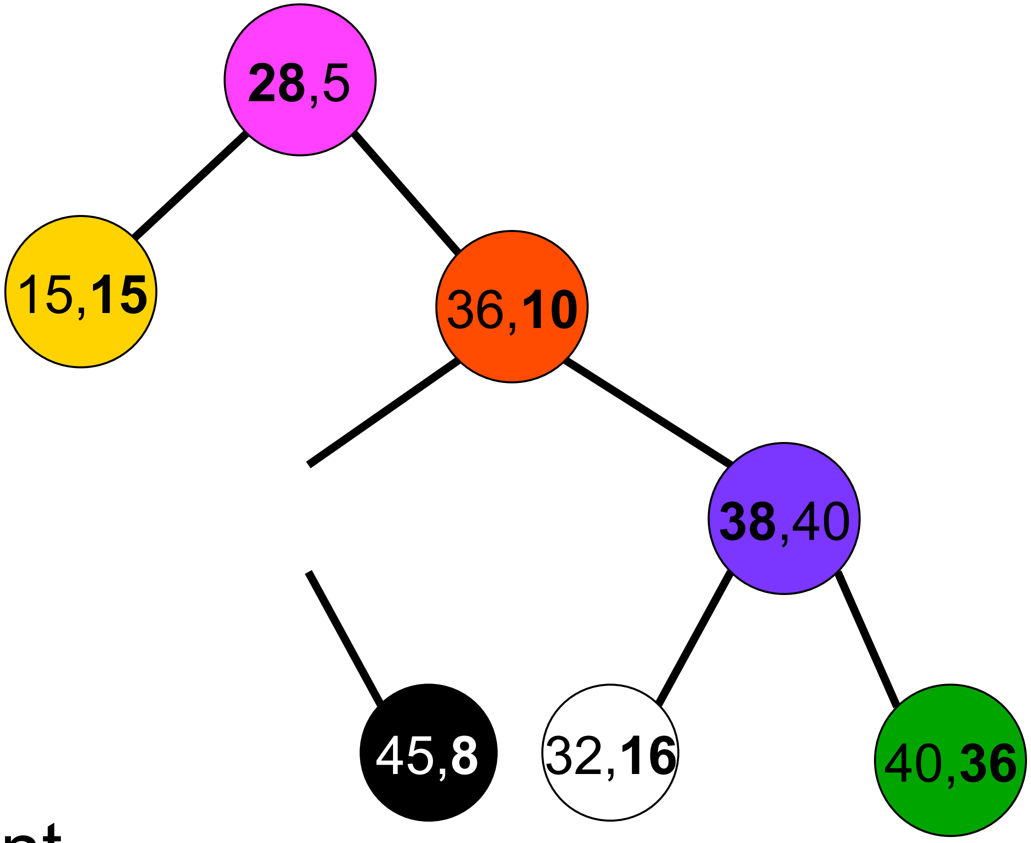


Delete the pink point

Copy the black point up

Deletion



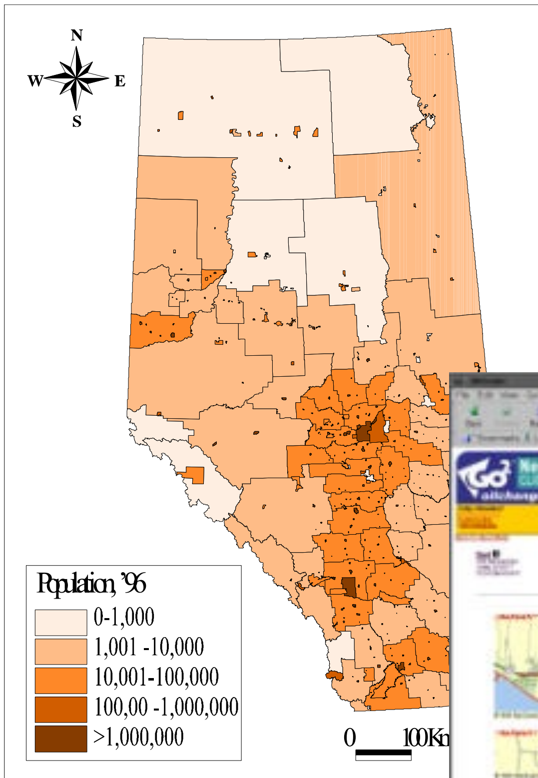


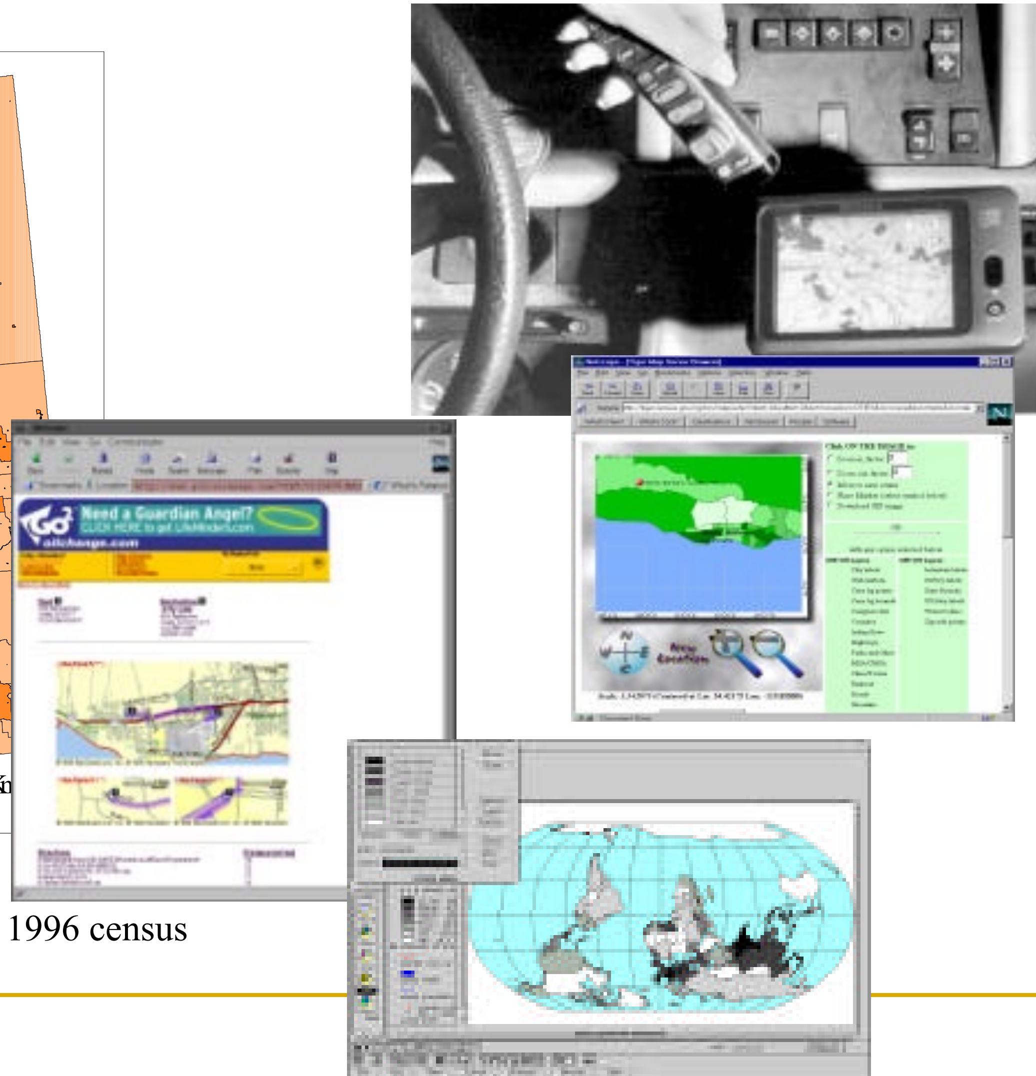
Delete the old pink point

Applications

- Query processing in sensor networks
- Nearest-neighbor searchers
- Optimization
- Ray tracing
- Database search by multiple keys

Examples of applications

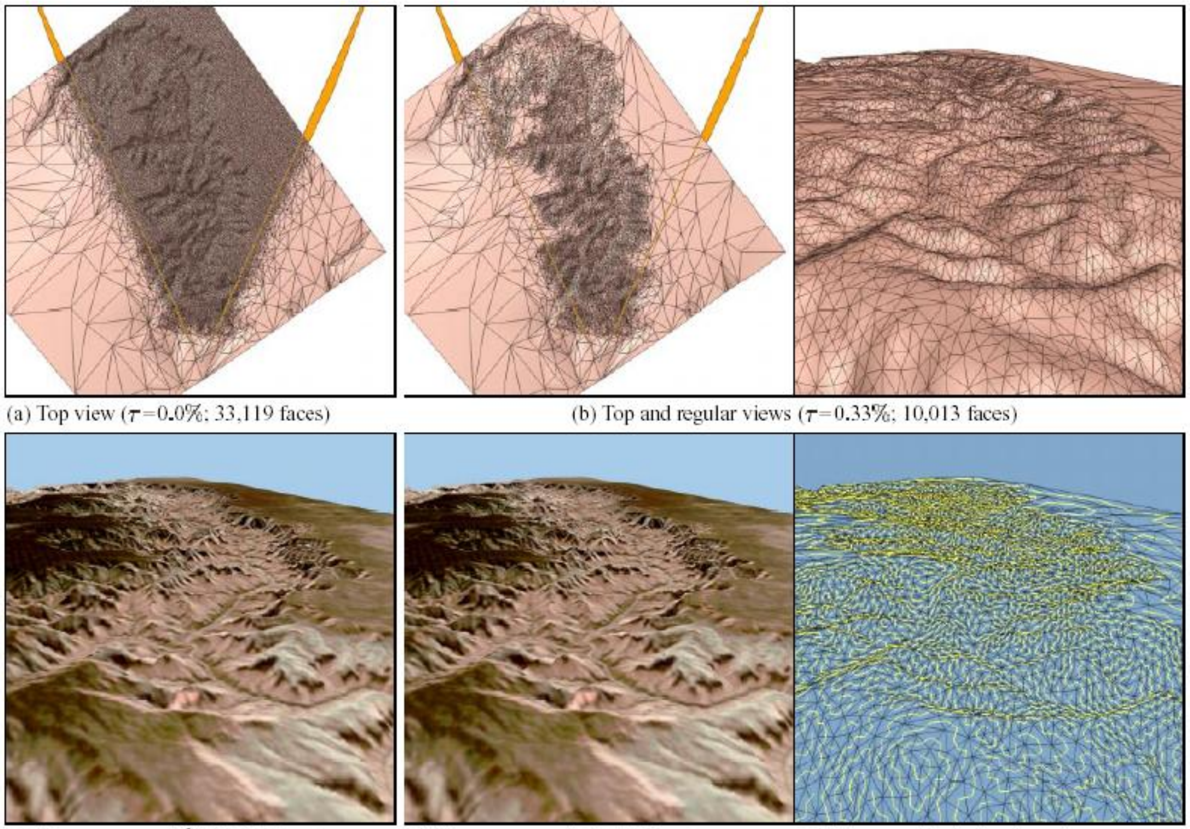




Population Distribution in Alberta, 1996 census

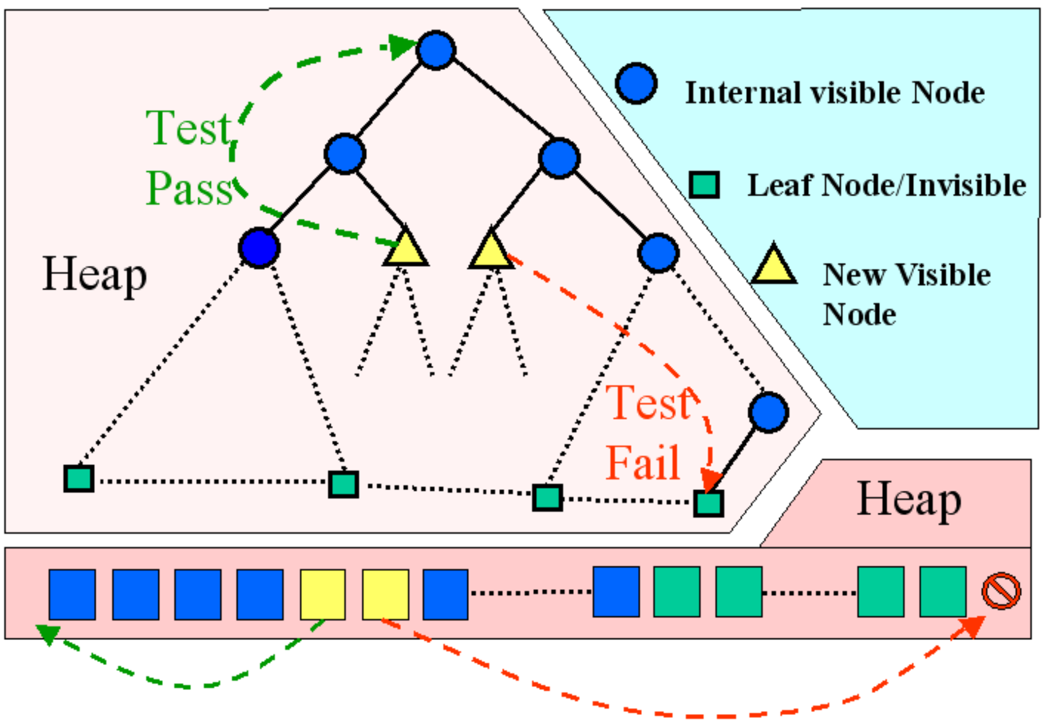
Progressive Meshes

Developed by Hugues Hoppe, Microsoft Research Inc. Published first in SIGGRAPH 1996.

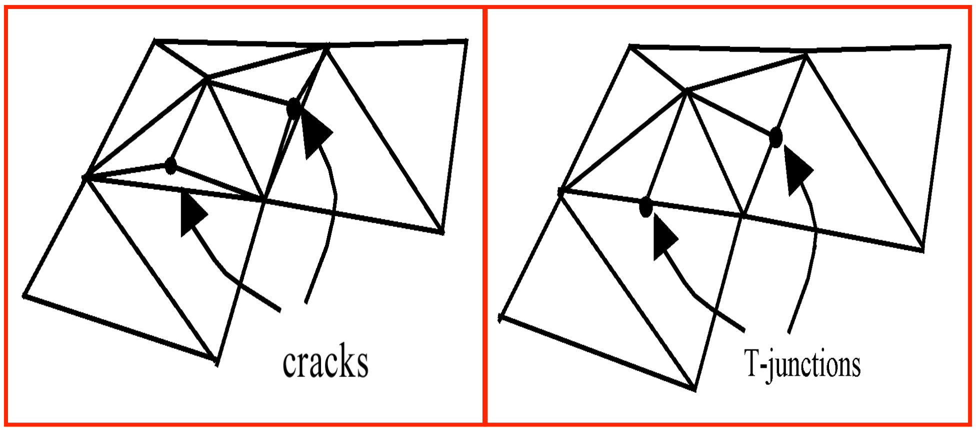


(a) Top view (τ=0.0%; 33,119 faces)
(b) Top and regular views (τ=0.33%; 10,013 faces)
(c) Texture mapped $\hat{M}$ (79,202 faces)
(d) Texture mapped (10,013 faces)
(e) 764 generalized triangle strips

Terrain visualization applications

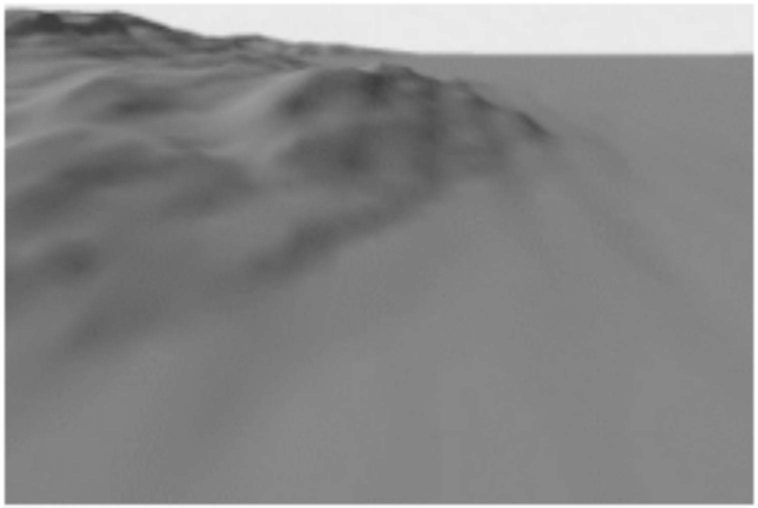


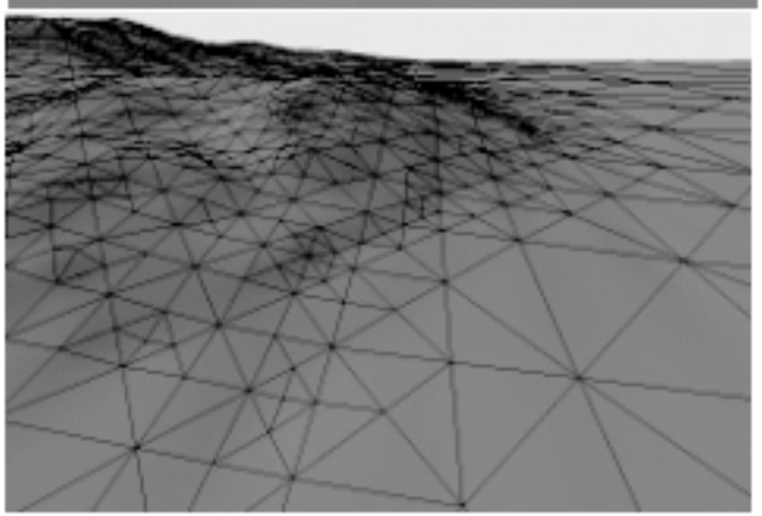
Geometric subdivision

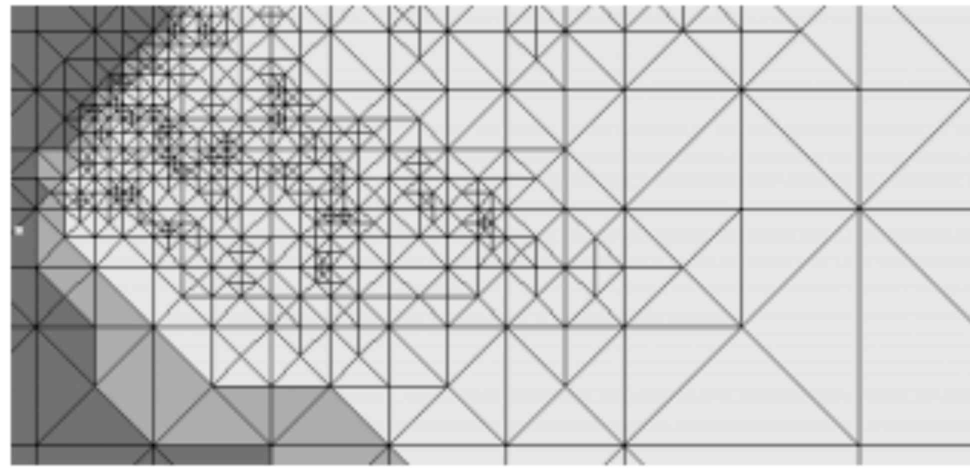


Problems with Geometric Subdivisions

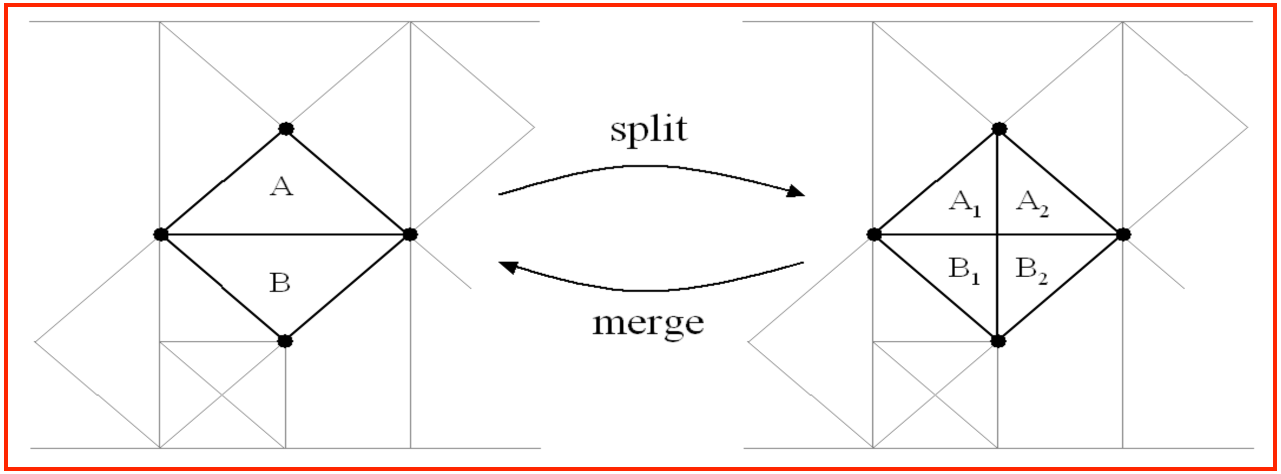
Real-time Optimally Adapting Meshes





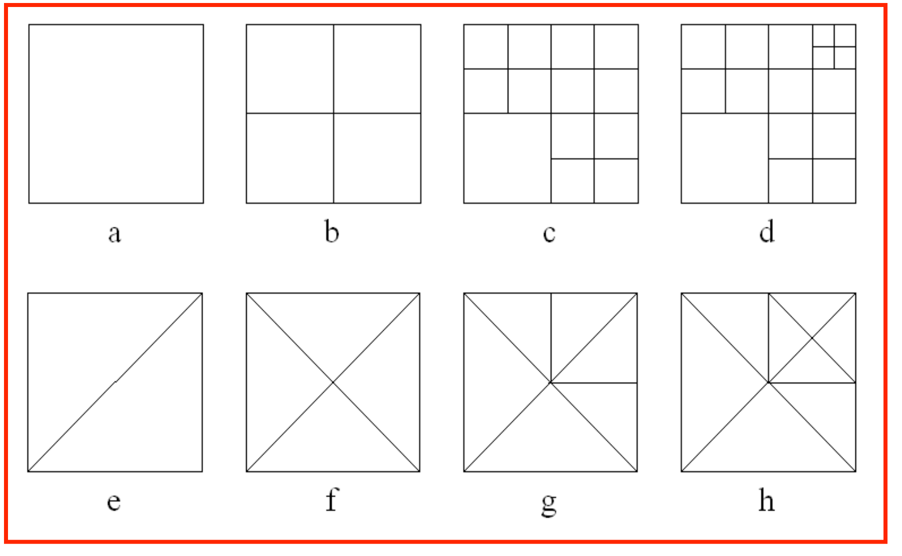


ROAM principle



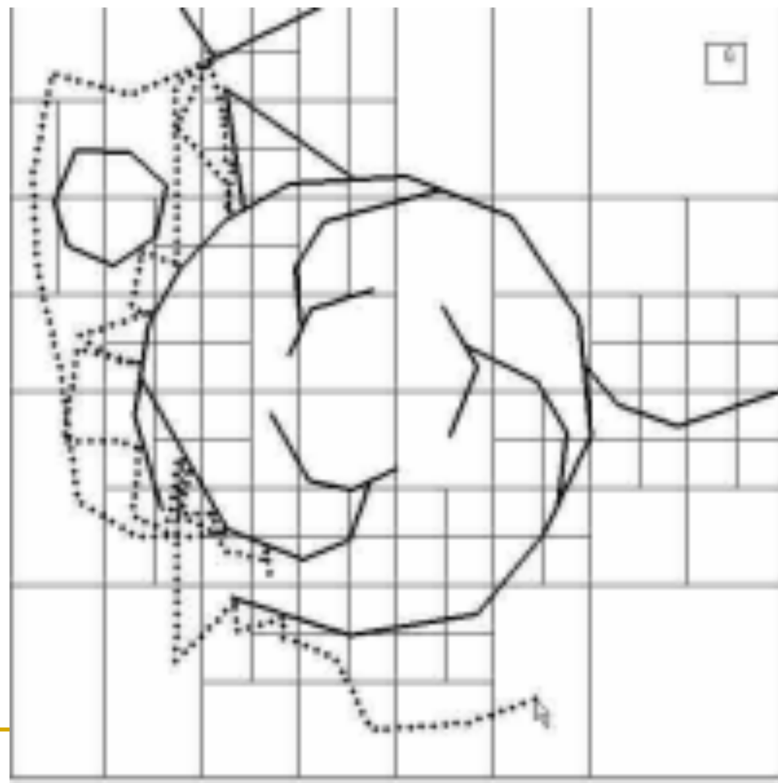
The basic operating principle of ROAM

Quad-tree and Bin-tree for ROAM (real-time adaptive mesh)



Parti-Game Reinforcement Learning

- http://www.autonlab.org/autonweb/14745/version/1/part/4/data/partigame-demo.mpg



Decision Tree

- Database indexing structure is built for decision making and tries to make the decision as fast as possible!