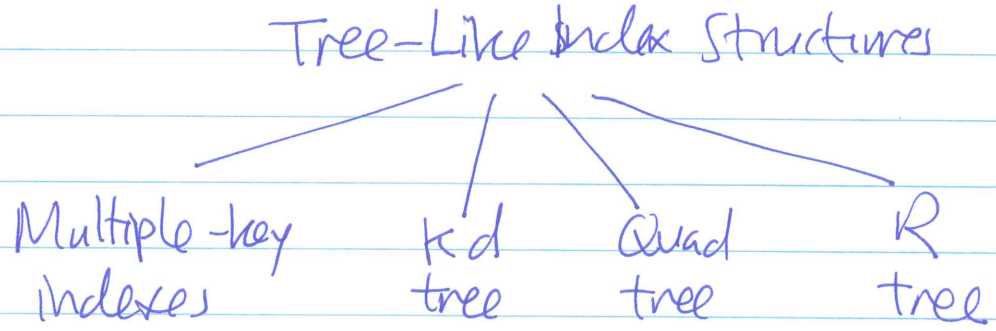
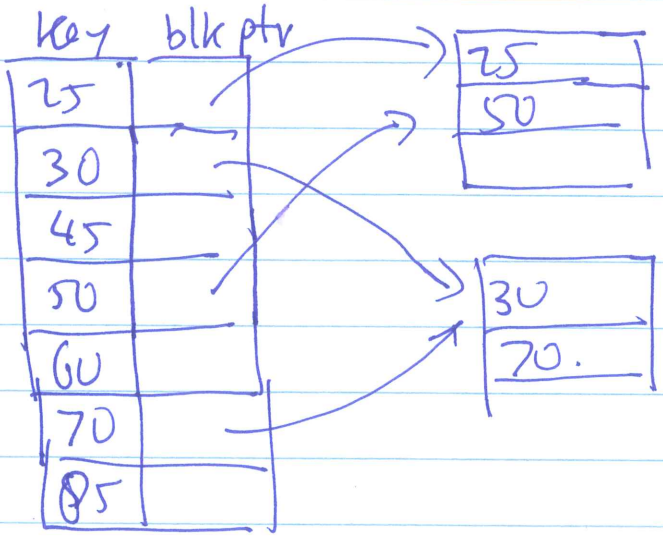
# Multi-Dim Indexes: Tree-Like Index structures.



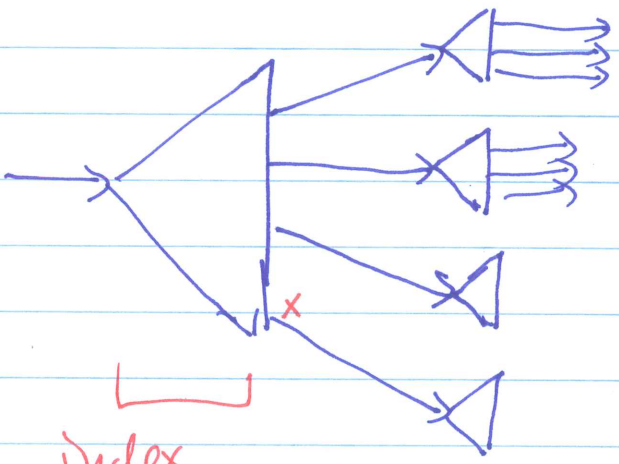

- Recall: line (single-dim) index:



# Multiple-key Indexes.

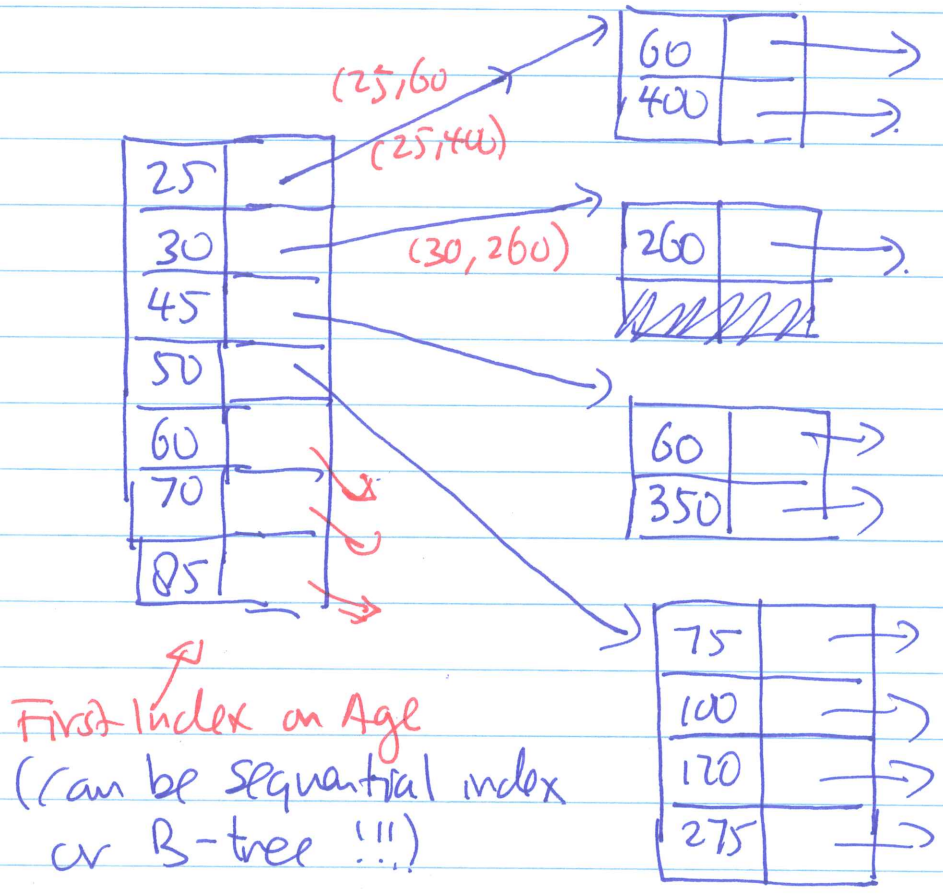
Data: (25, 60) (45, 60) (50, 75) (50, 100)
(50, 120) (70, 110) (85, 140) (30, 260)
(25, 400) (45, 350) (50, 275) (60, 260)

- Structure of a Multiple-key index:




Can be quite small !!!

Example : (2-level index on (Age, Salary)

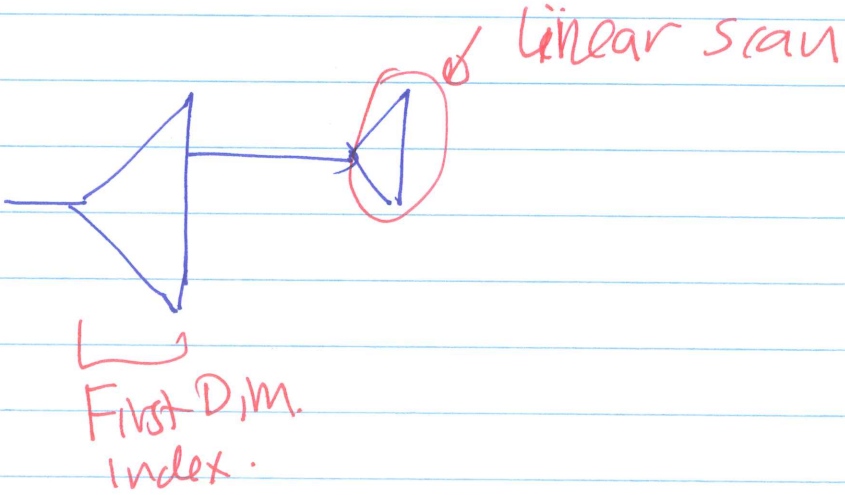


# Performance of Multiple-key Index:

- Look up a specific "point" (record):
  - Look up key in first dimension
    ⇒ Find the next index (tree).
  - Lookup key in $2^{nd}$ dimension
    etc.

- Partial Match:

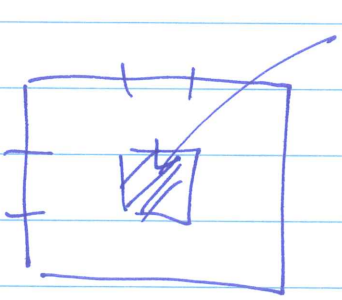
  If FIRST dim. attr. value is given, the access is efficient:

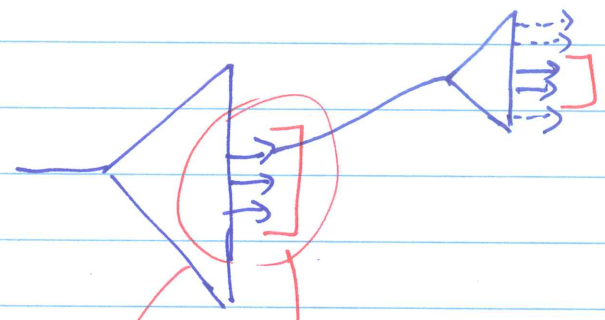


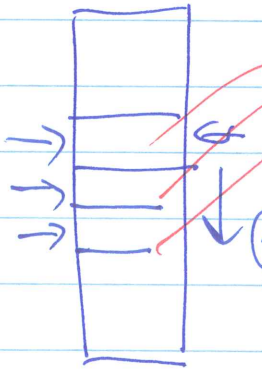

  If FIRST dim. attr. value is NOT given, we must search every sub-index....

  → Can be very time consuming.

# Range query:



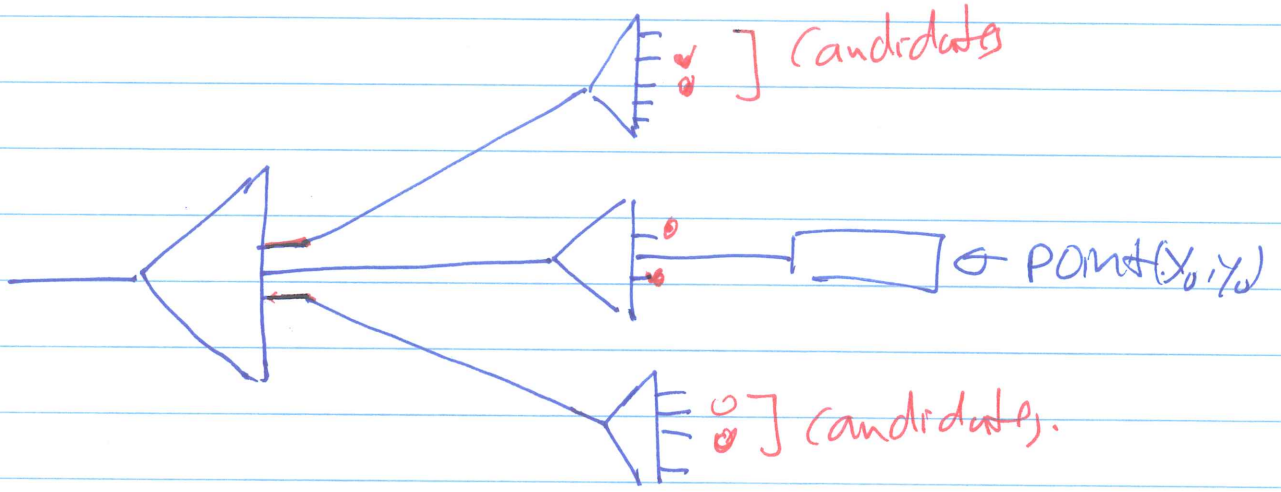



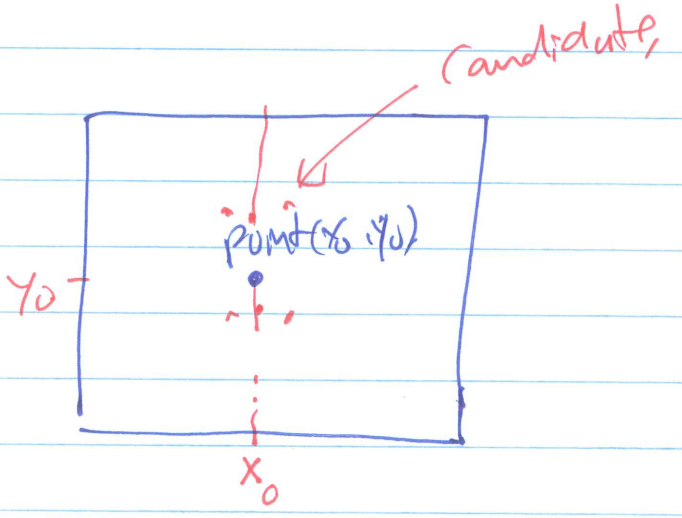
(1) Find the smallest value.

(2) Traverse (Scan) through the index file until you reach the upper value.

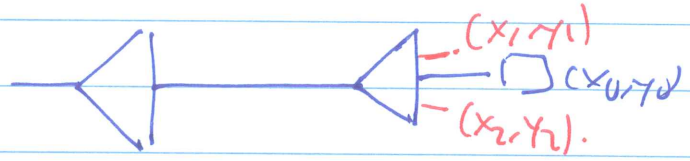
(3) Do the same for the 2nd dim. index (k times, once for each value in the range).

- Nearest-Neighbor Queries:
  - Can only be found by a search

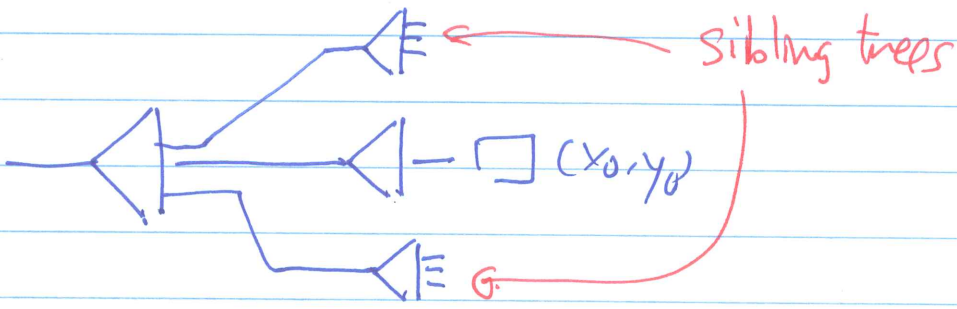


(1) Find a nearest point $(x_1, y_1)$ in your own subtree:



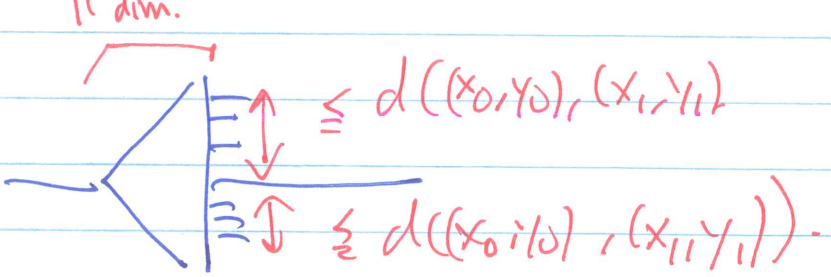

(2) If none exists, try in your sibling trees:



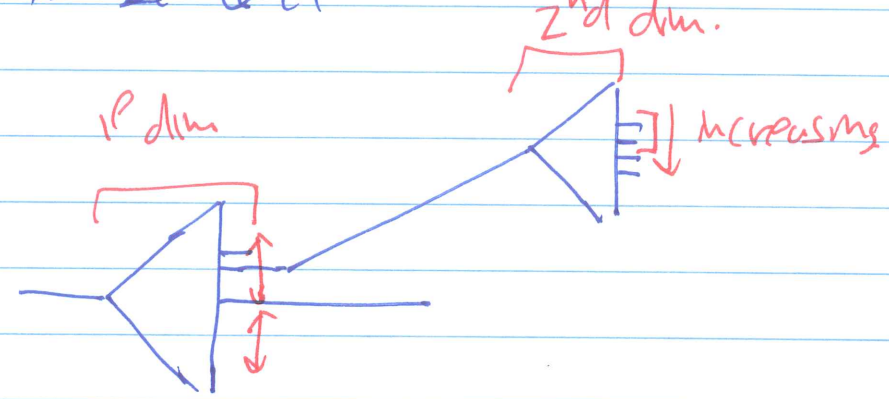

(3) This nearest point $(x_1, y_1)$ will establish a

~~lower bound~~ upper bound

on the search for more sibling trees:



(4) The distance $d((x_0, y_0), (x_1, y_1))$ is also used to limit search in 2nd level:

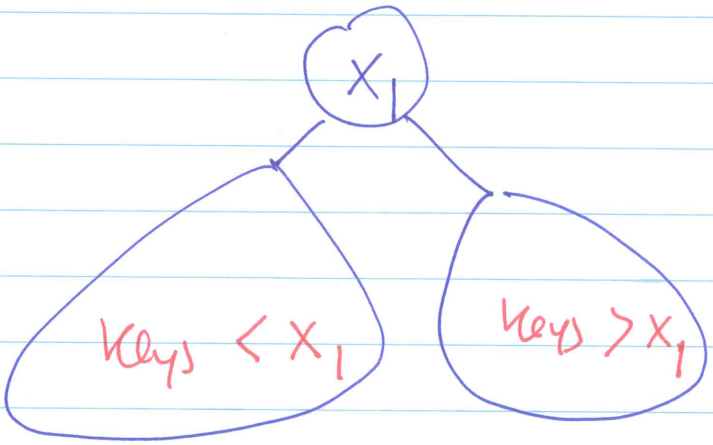


# Kd-trees (K-dimensional search tree).

- **Fact:**

  Kd-tree is a **main-memory** data structure that generalizes the binary search tree (BST) into multi-dimension.

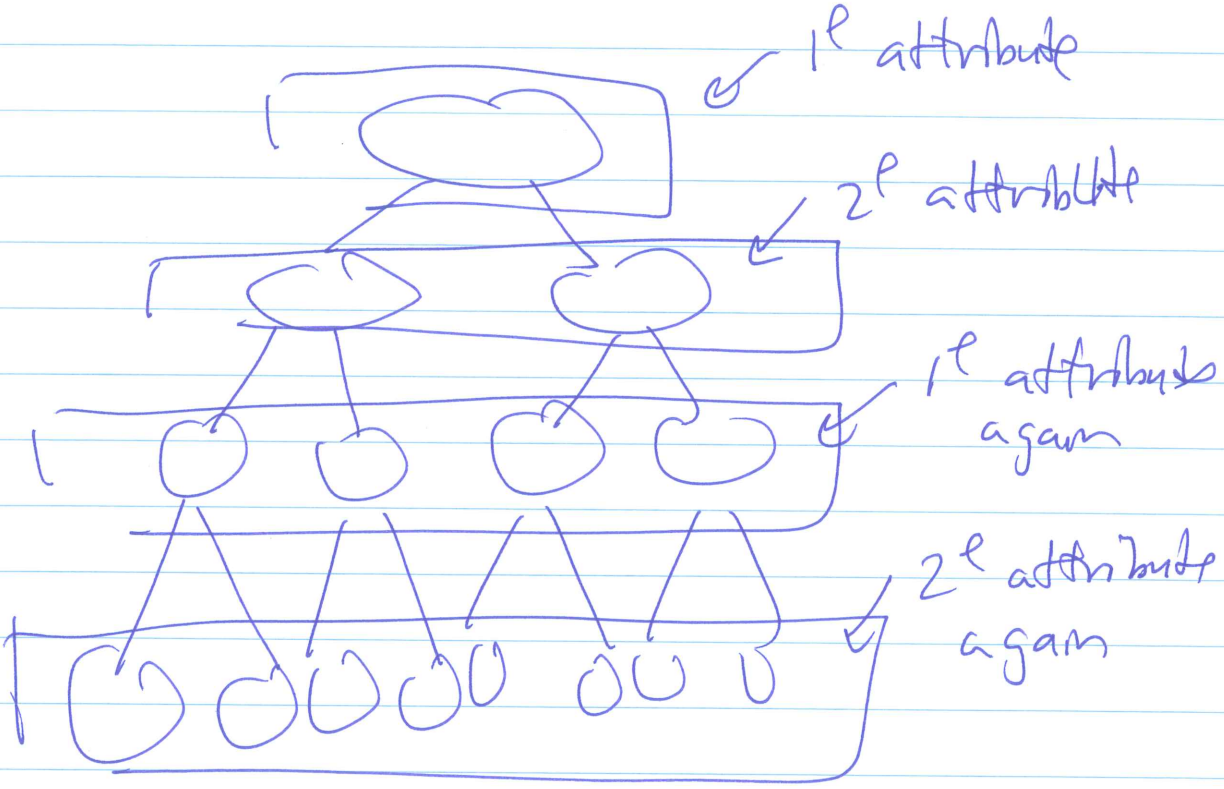
- **Recall:** Binary Search Tree:



Note: the dimensions

~~rotations~~ rotates

~~through~~ the levels of the kd-tree:




And so on !!!