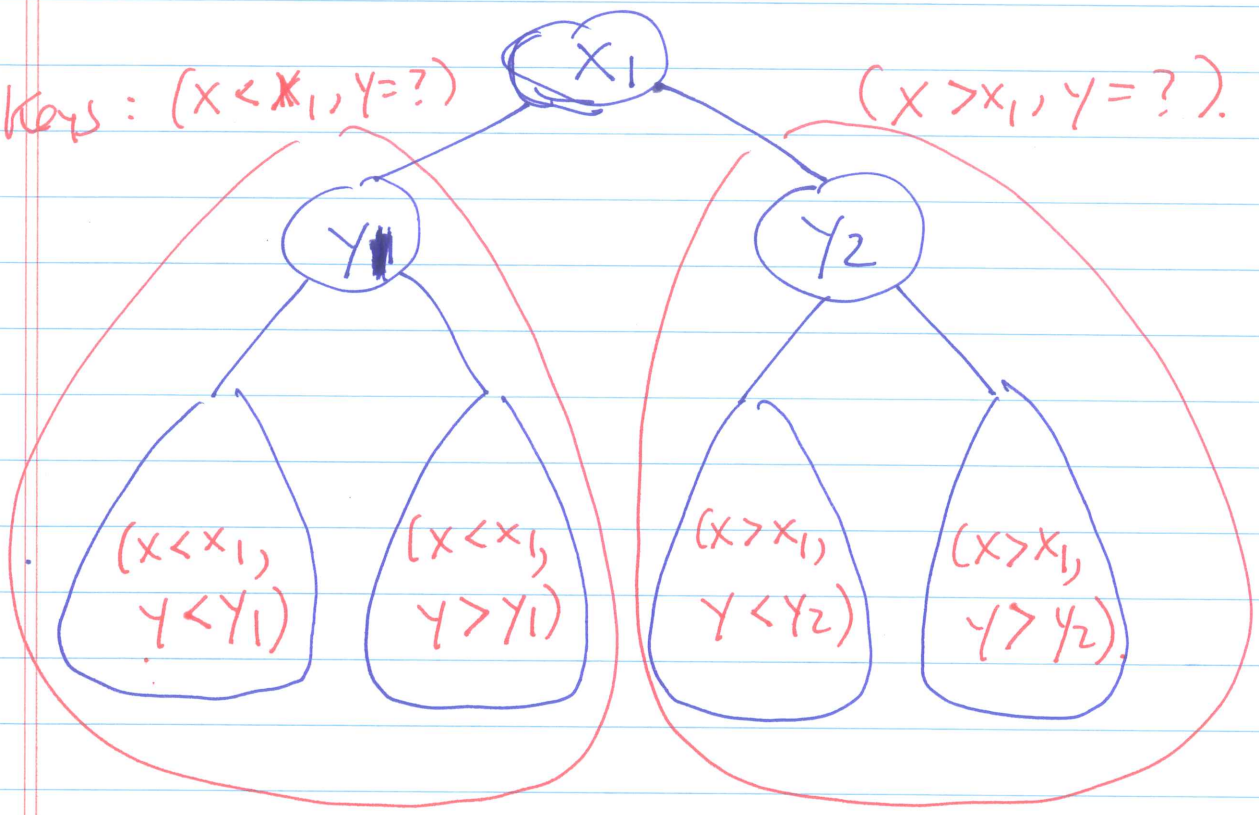
Multi-dim Indexes (Tree)

- **Kd-tree**: eg: 2 dimension. (x, y).



Note: In the classical kd-tree ÷

data points (multi-dim. values) are placed at the nodes of the kd-tree.

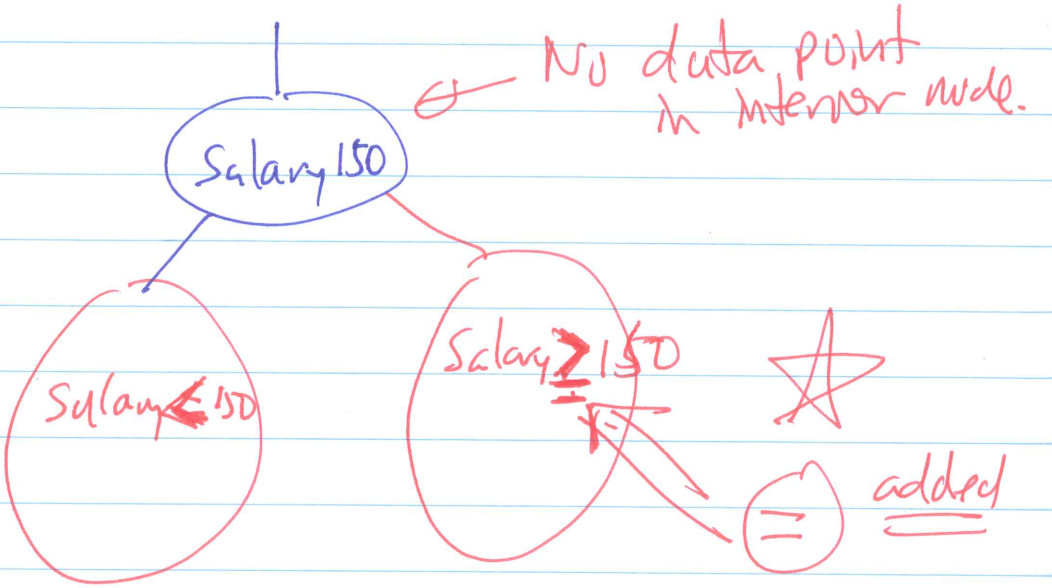
Modifications needed to simply storing kd-tree on disk.

Modifications to classical kd-tree:

(1) Interior nodes do NOT store data.

Interior nodes store:

(A) attribute (name)
(B) dividing value.



(2) Only the leaf blocks (nodes) contains data.

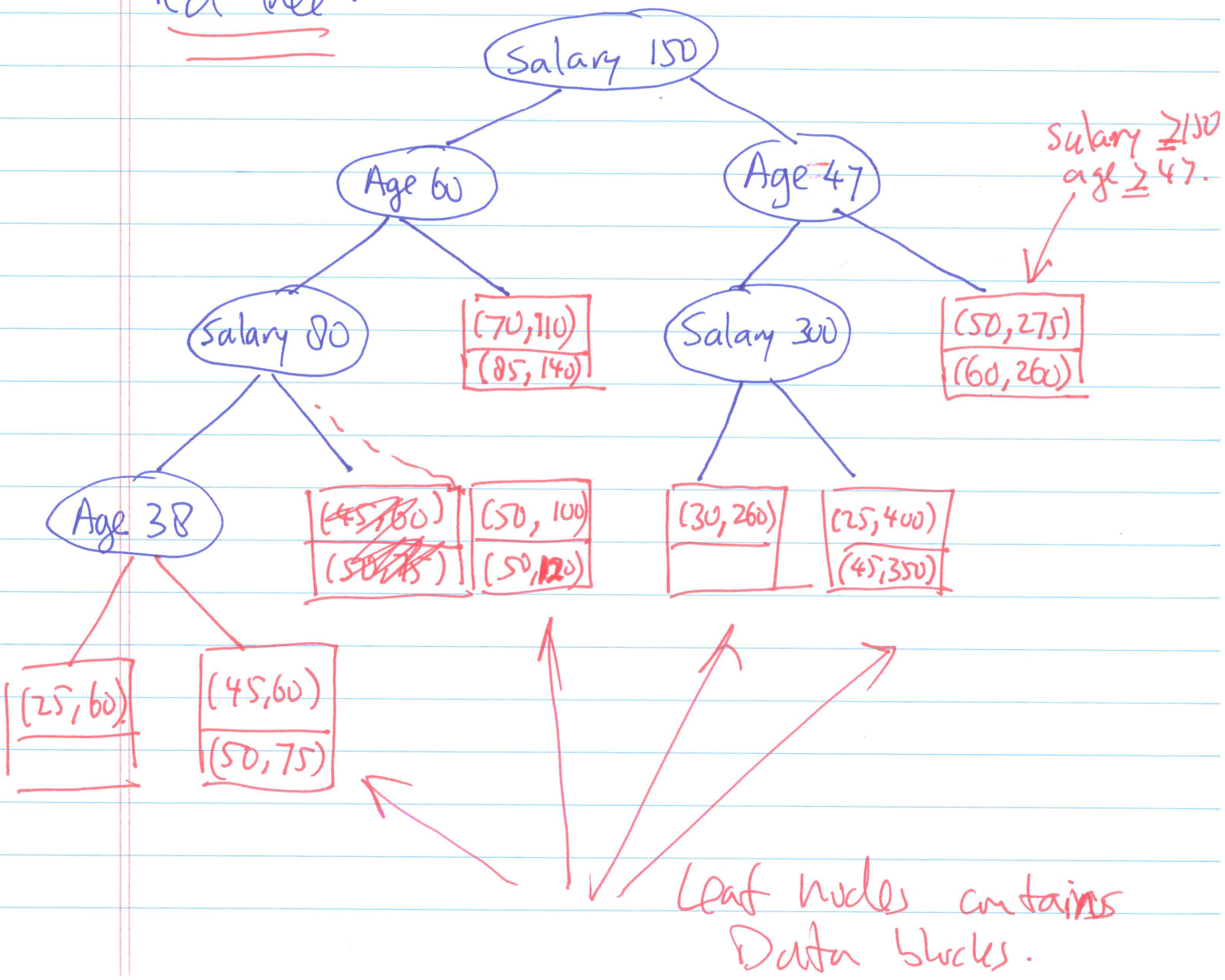
Example:

Input data:

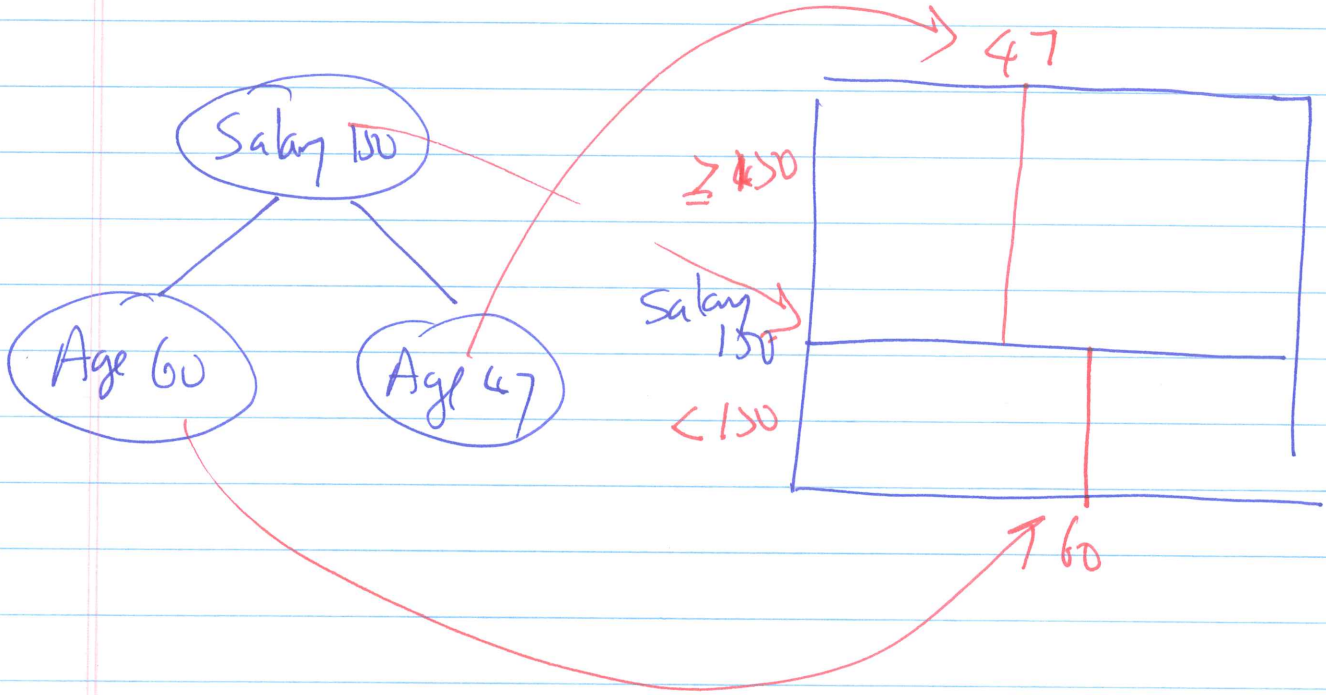
Age Salary

(25, 60)	(45, 60)	(50, 75)	(50, 100)
(50, 120)	(70, 110)	(85, 140)	(30, 260)
(25, 400)	(43, 350)	(50, 275)	(60, 260)

Kd-tree:

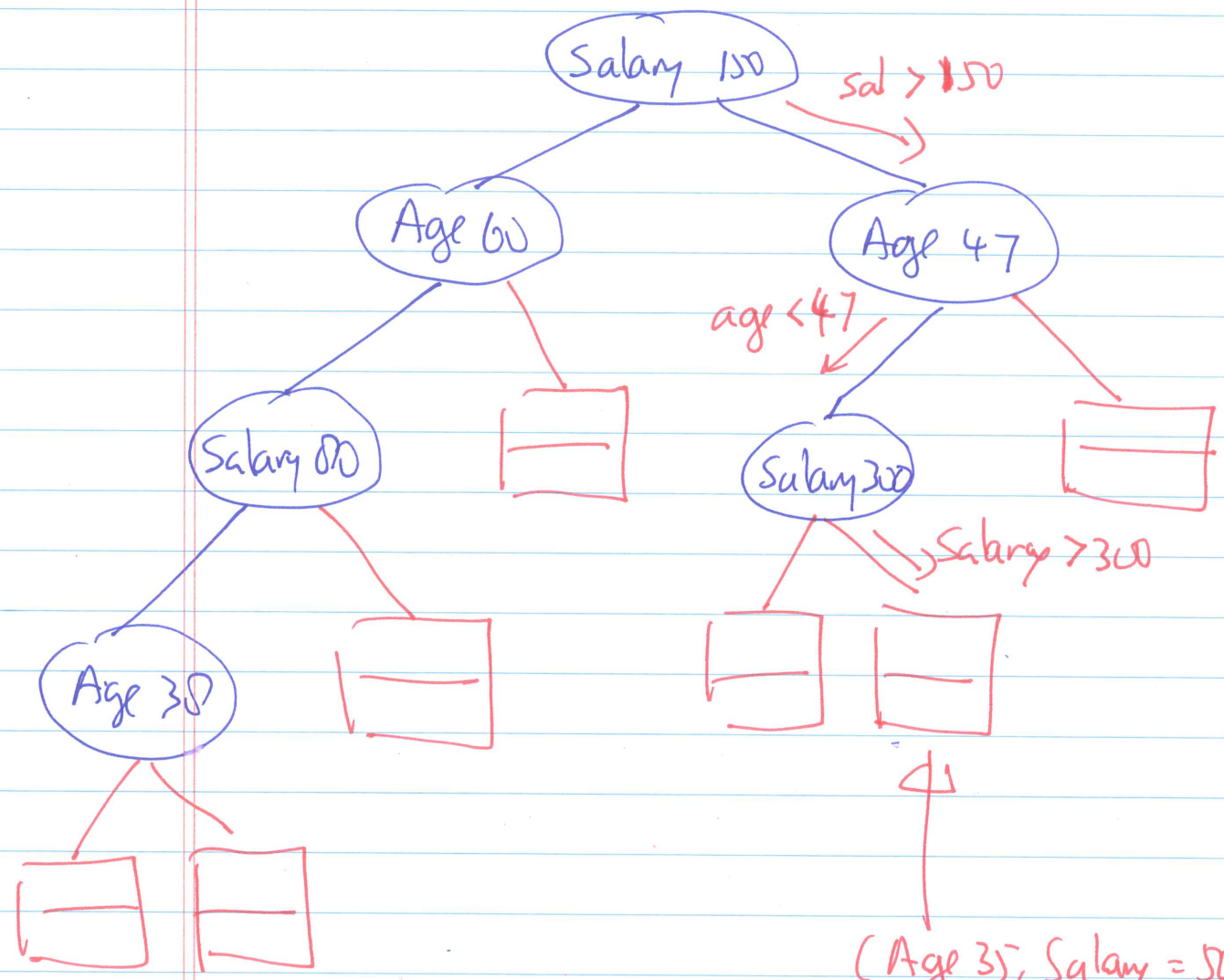


Pictorial Representation of a 2-dim kd tree:



Lookup in a kd-tree:

Lookup: (Age = 35, Salary = 500k)



(Age 35, Salary = 500)

must be store HERE

(search in block)

Insert into a kd-tree.

(1) do a lookup first – to find the leaf blk

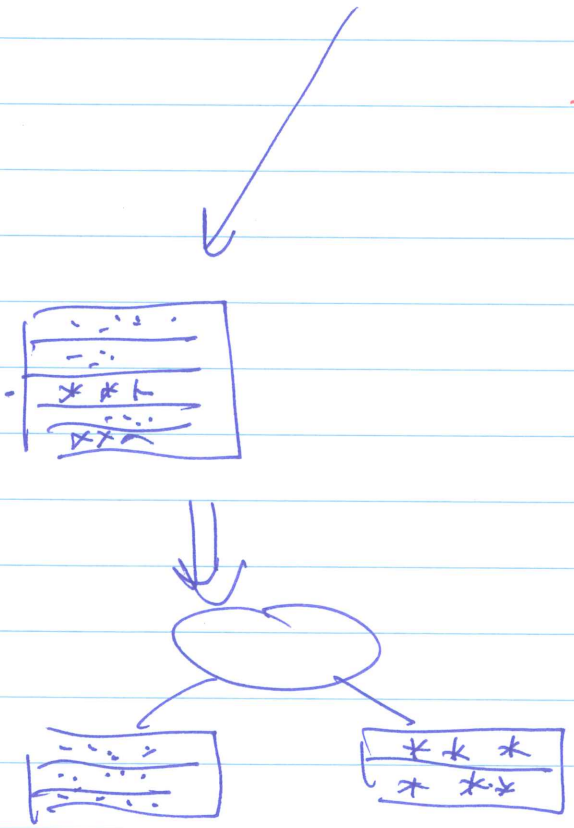
(2) If leaf block has room for record

→ insert in block.

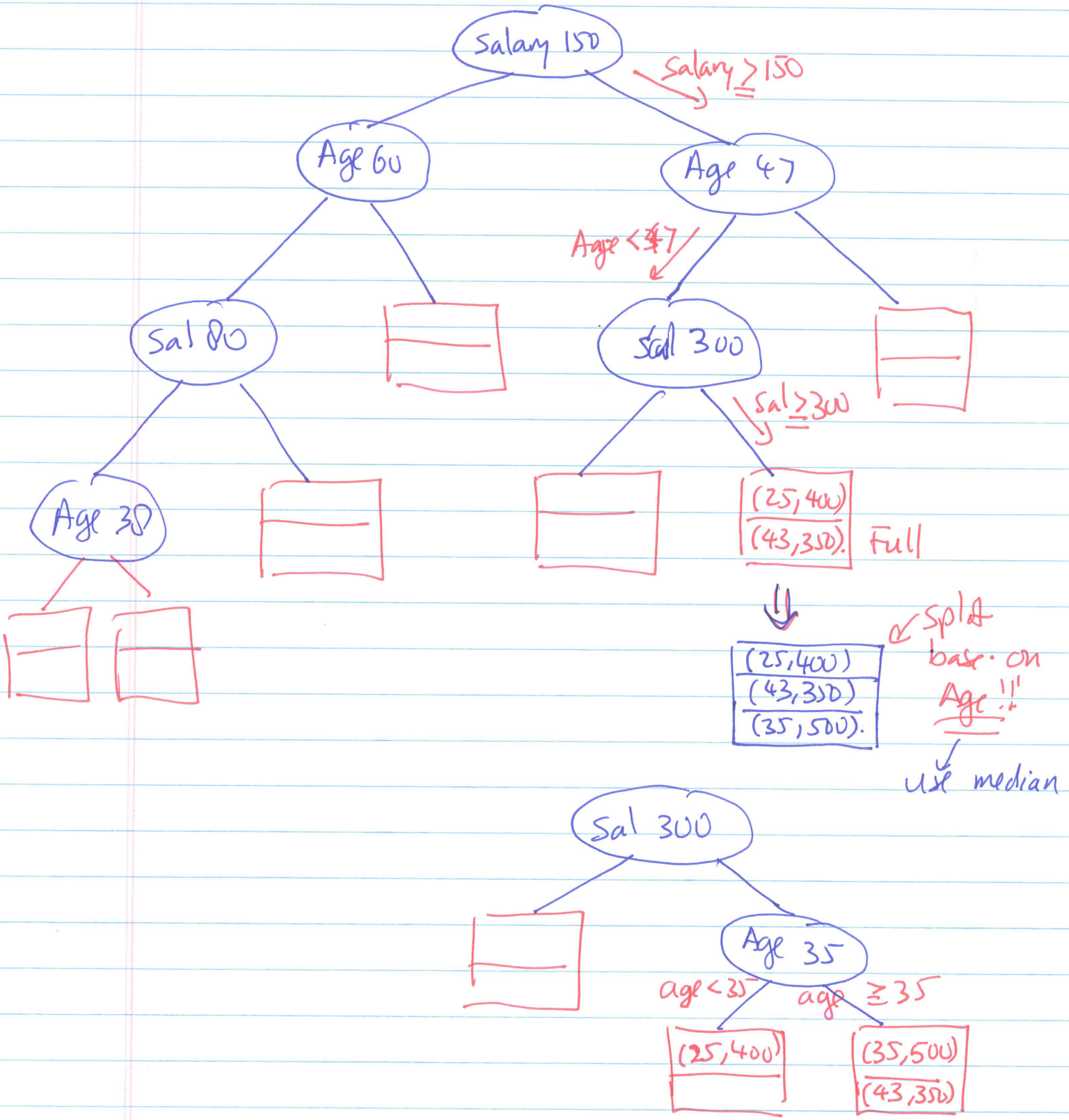
If leaf block if full

→ Split block. (make an interior node).

the dimension of the attribute used to split depends on the level of the leaf block!!



Example: insert (Age=35, salary=500K)



Delete operation

- Delete is very complicated. (especially takes many blocks accesses.)
- So instead of Delete,

Database systems usually

Rebuilds the indexes

from scratch.

- take down the database at night.
- Rebuild index files.

Kd-trees. performance

- Partial Match query:

(1) for the dimension that has a

given search value

↓

take the (one) branch down. (correct)

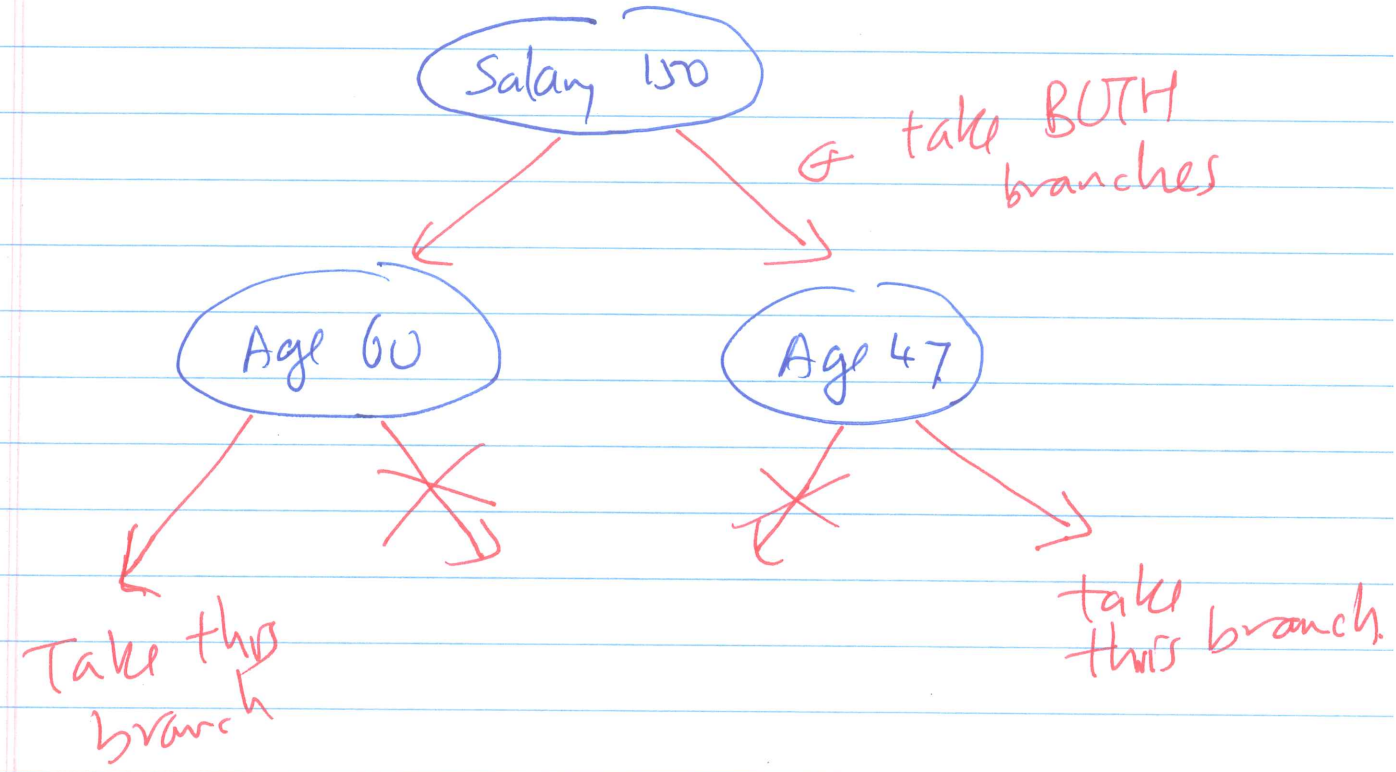
(2) for the dimension that has

NO search value

↓

take BOTH branches down.

Example: (age = 50, salary = *)



And so on.

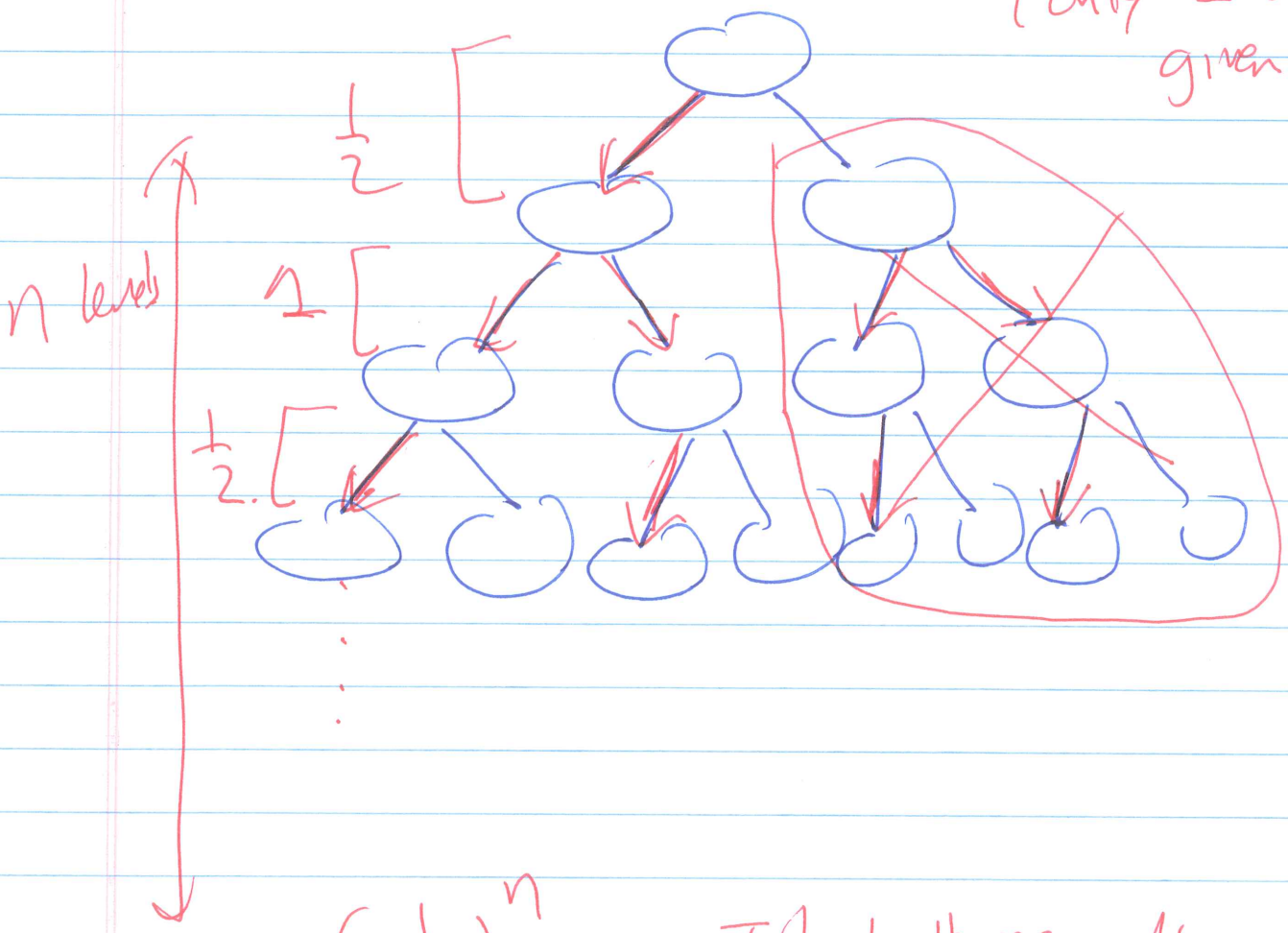
Question: How much did this improve?

↓

How much did the kd-tree reduce the search space ?!!

Assume: kd-tree is balanced
(same # height every where)

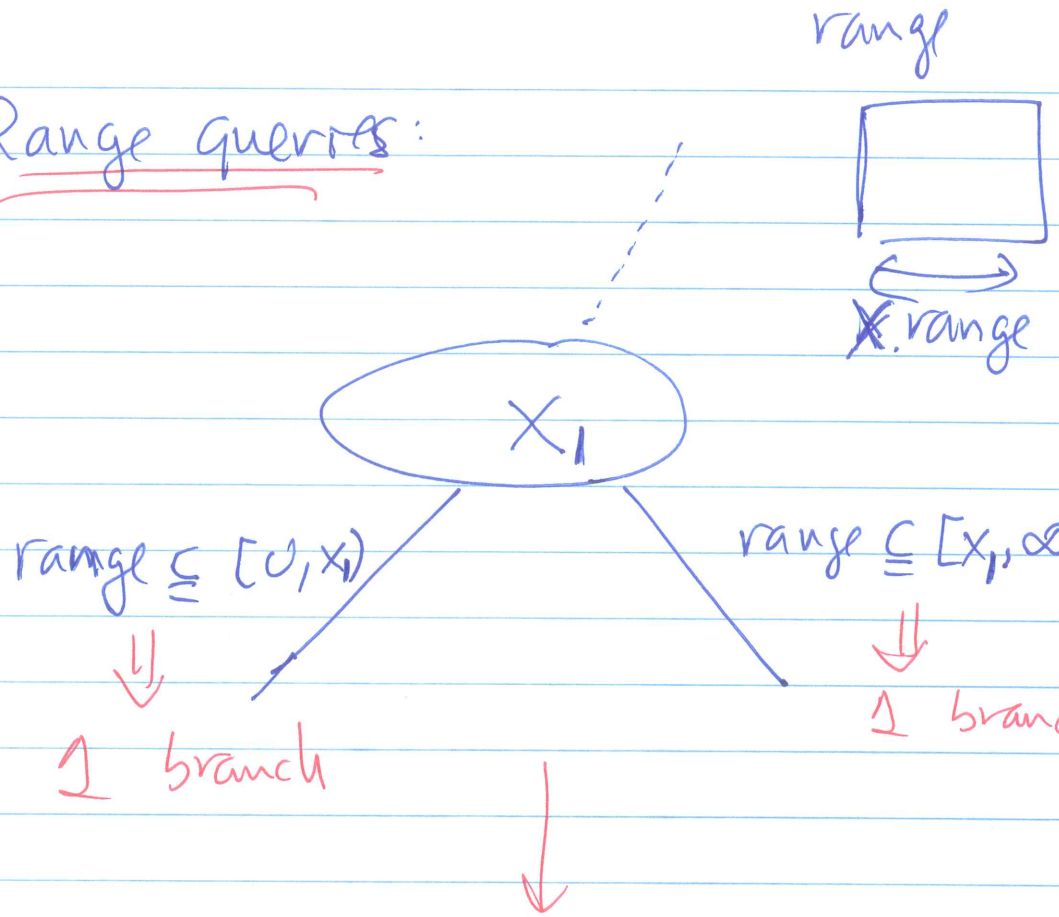
2n levels.



$\left(\frac{1}{2}\right)^n$ * total # records

(pretty good)

* Range queries:



range saddles (splits) at X_1

search both branches.

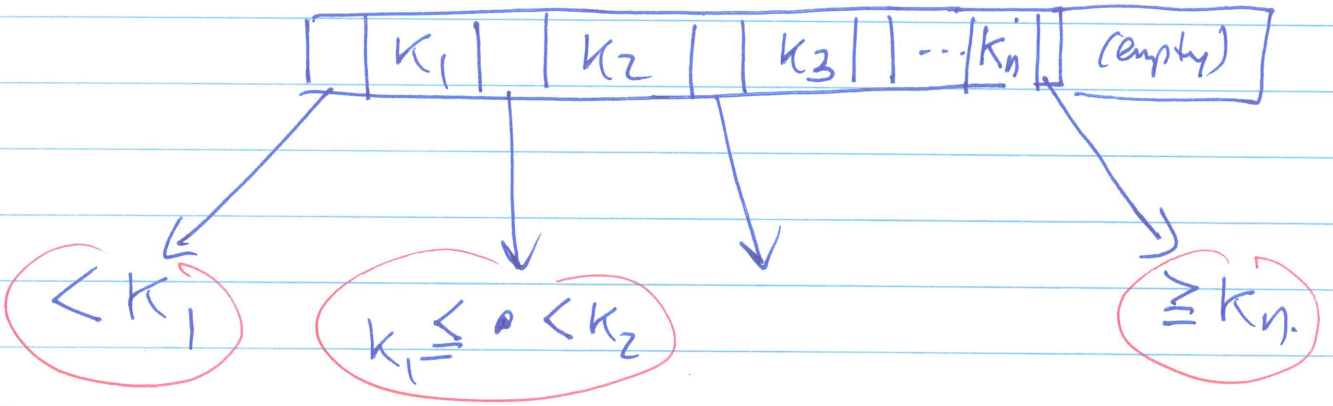
(same decision in the second dimension)

- Nearest Neighbour: ~~No use at all....~~

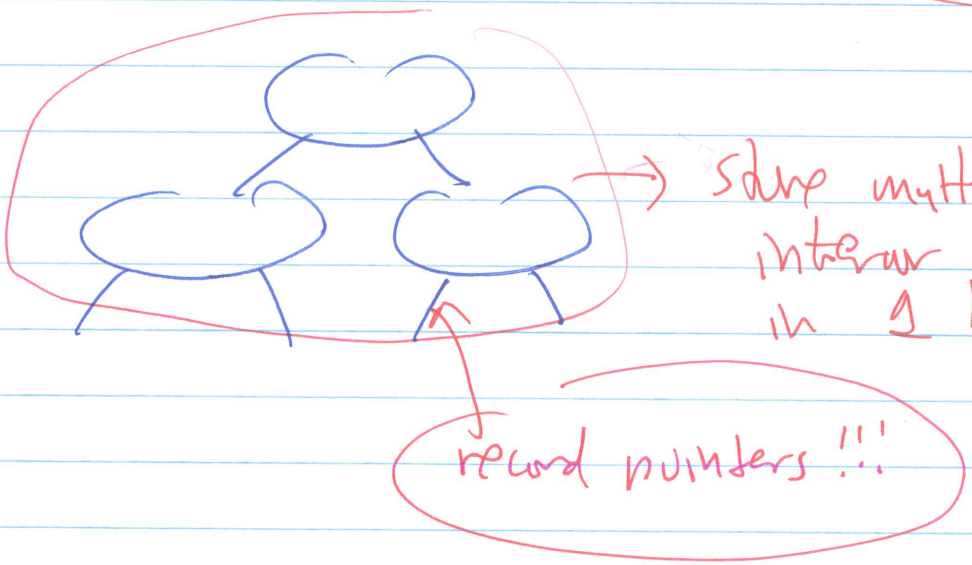
tough ... (not discussed in book)

Adapting kd-tree to Secondary Storage

(1) Use multi-way branches at interior nodes. (like a B-tree)



(2) Group interior nodes into one block



The pointers uses are

record pointers.

Use pointer swizzling when

the block is read into memory!!