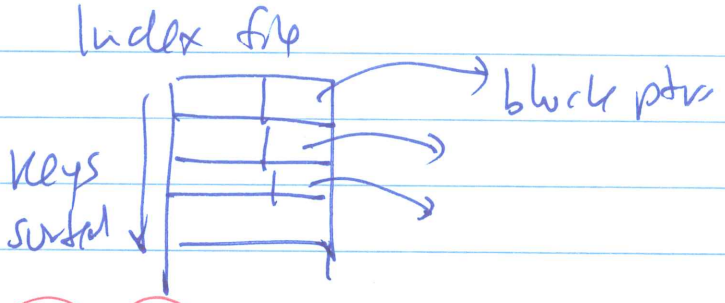
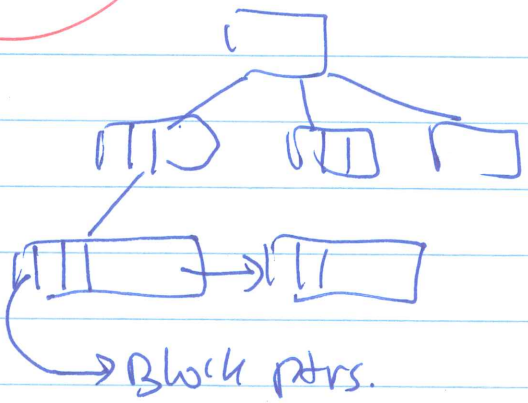
# Multi-dimensional Indexes (Intro + Motivation)

- The indexes studied so far:

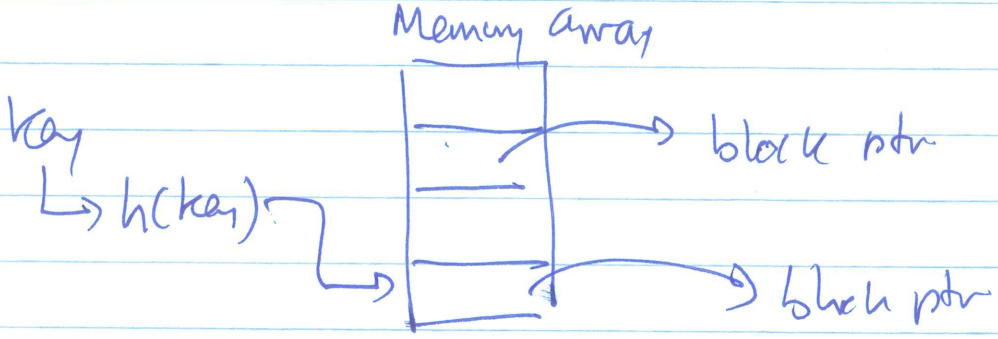
(1) Sorted indexes:




(2) $B^+$-tree




(3) Hashing



Common property:

All these index methods are "one-dimensional".

search key value:

→ ordered in one dimension.

(The Hash function value provides the one-dim. ordering).

- Argument:

(1) There are multi-dimensional information.

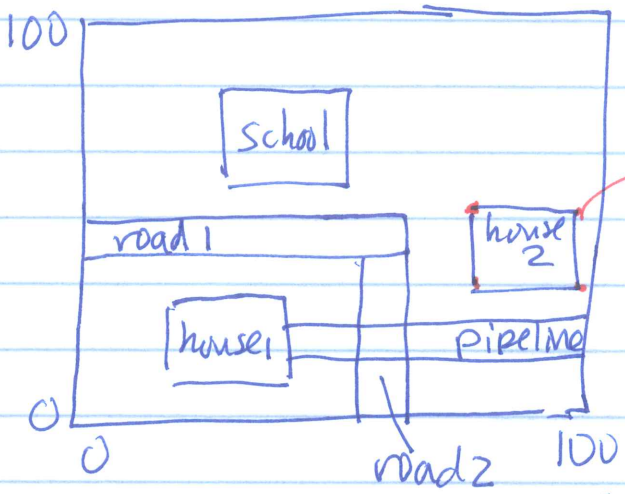
(+) queries.

(2) The index techniques (that we studied so far) are

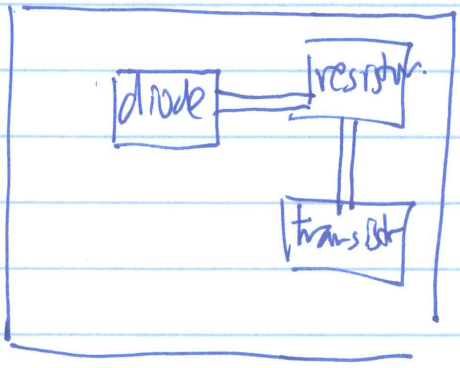
inadequate to "help" in speeding up accesses of multi-dim. info.

# Examples of multi-dim. information

- Example: geographic information:




- Example 2: circuit board.



# What multi-dim. queries look like

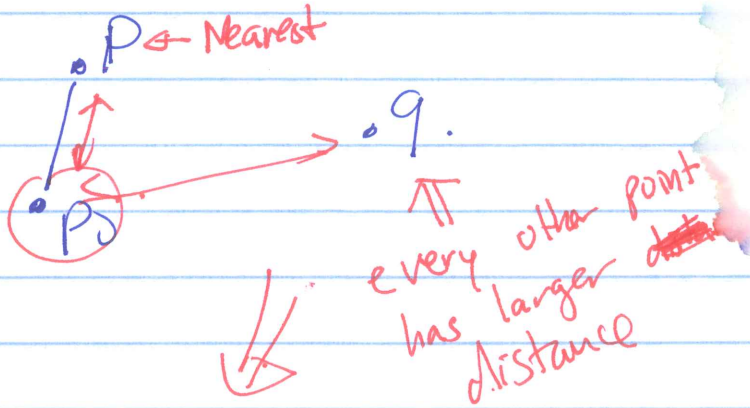
## Multi-Dim. Queries

- Suppose a point is represented by 2 coordinates $(x, y)$:

  point(x, y).

- Given a point $P_0(10, 20)$.
- We want ~~to~~ find points stored in the DB that is the closed to $P_0$.

Find nearest point




Solution:

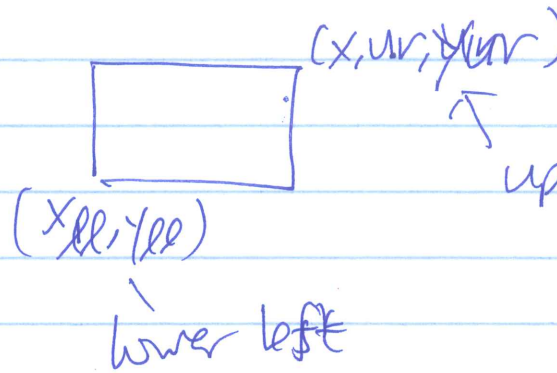
```
select *
from points p
where not exists
    (select *
    from points (q)
    where (distance(q, P0) < dist(p, P0))
```

or:

```
(select *
from points q
where (q.x − 10)^2 + (q.y − 20)^2 <
      (p.x − 10)^2 + (p.y − 20)^2 ).
```

Another multi-dim. query: is a point inside a rectangle

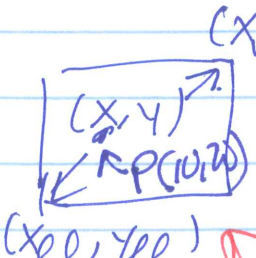
- Rectangle(id, $x_{ll}$, $y_{ll}$, $x_{ur}$, $y_{ur}$)




- point P (10, 20).

Question: find all rectangles that contains P.

select id
from Rectangles
where




graphically

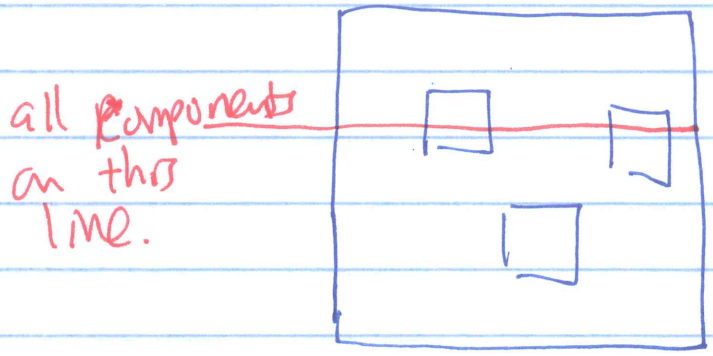
$x_{ll} \leq 10 \wedge y_{ll} \leq 20$

and $10 \leq x_{ur} \wedge 20 \leq y_{ur}$.

coordinate of the point P

# Common Query types of geographic information.

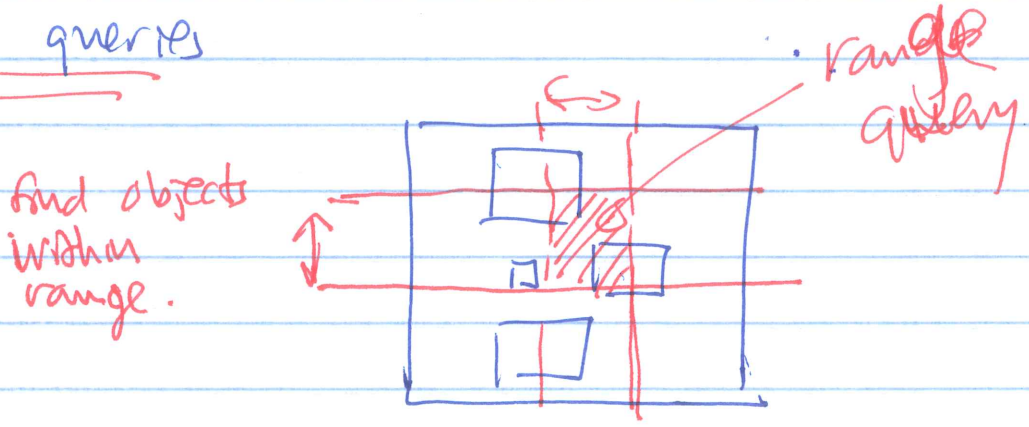
(1) Partial match queries. (Not all dimension are given)




Specify values for 1 or more dimension

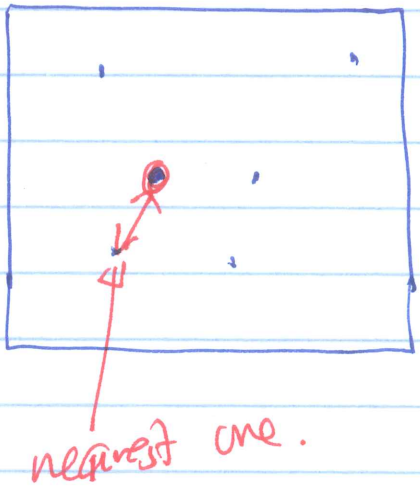
Find all points/objects matching specified values in given dim.

(2) Range queries

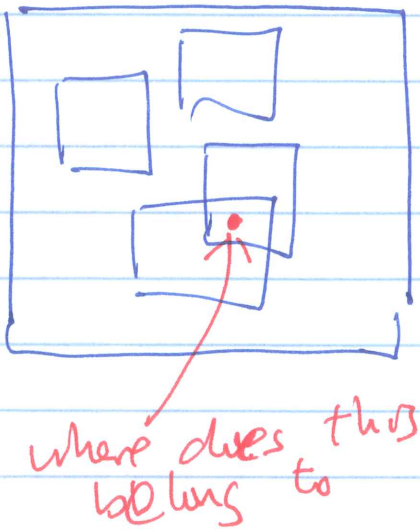



- partial or wholly within given range.

(3) Nearest-neighbor queries




(4) Where-am-I queries:




Given a point

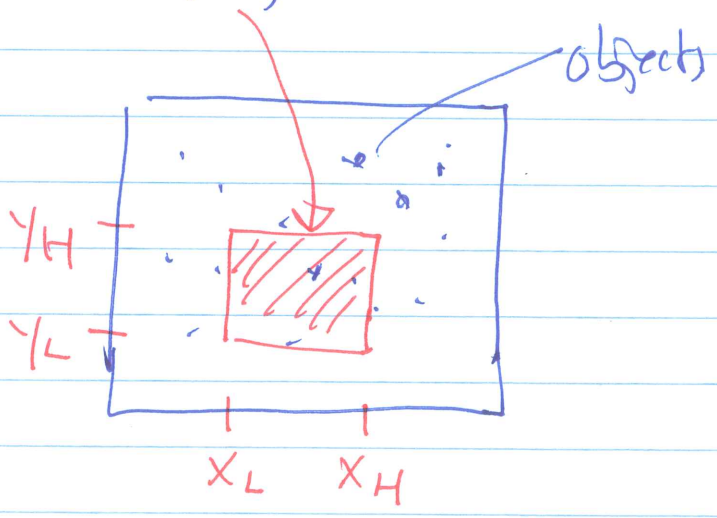
Which object(s) contains that point.?

## Motivation for developing "multi-dim index" techniques:

- $64,000 question: can our index technique support geometrical queries efficiently ??

- Case study: processing a range-query using a B-tree index.

Range query: Find all object in this rectangle:



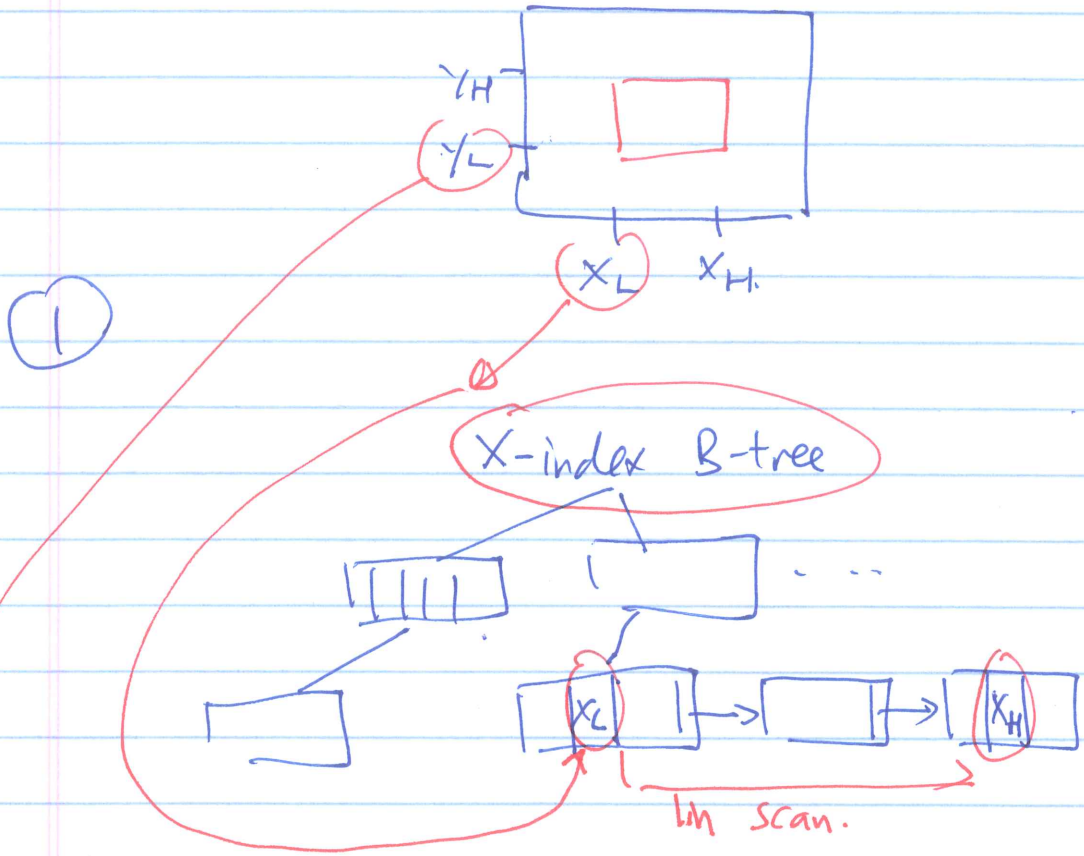

Suppose: we have B$^+$-tree index on:

- X-coordinates
- Y-coordinates

of all objects

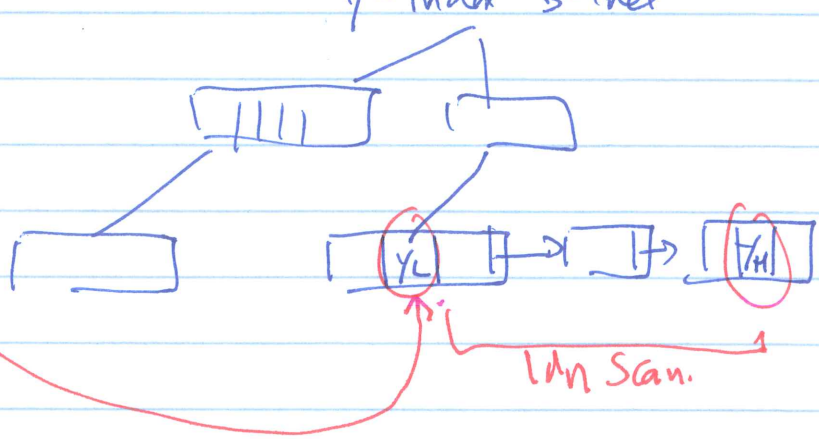
# Use x-index + y-index to process range query

How to use a $B^+$-tree index to process range query:




Use $x_L$ to find leaf node in X-$B^+$-tree.

2 Use $y_L$ to find leaf node in Y-$B^+$-tree



(3) Intersect the pointers from BOTH results.

(4) Retrieve records of the intersecting pointers.

BUT !!!

The procedure

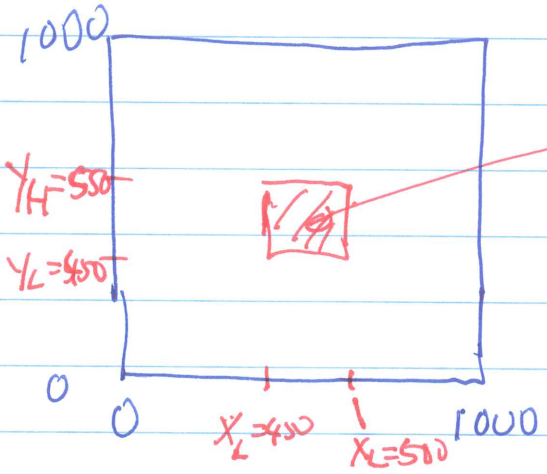
- It sounds good and may speed up.

But practically, it is surprisingly useless !!!

↓

~~Need~~ # blocks read will be about the same as reading the entire file !!

A concrete example: 1,000,000 points
- uniformly distributed




Approx: $\frac{1}{100} \cdot 1{,}000{,}000$
$= 10{,}000$ pts.

Statistics:

- Approx. $\frac{1}{10} \cdot (1{,}000{,}000) = 100{,}000$ pts
  with x-coord between [450, 550]
- Approx $\frac{1}{10}(1{,}000{,}000) = 100{,}000$ pts
  with y-coord between [450, 550]
- Approx $\frac{1}{100}(1{,}000{,}000) = 10{,}000$ pts
  in the intersection.

→ Answer of range query = 10,000 records

- We need some

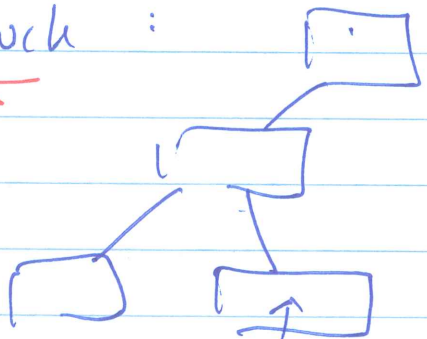
  Storage information

  to compute processing cost.

- Storage structure assumptions:

1 data block: | 100 points. |

1 B-tree block:



200 key-pointer pairs.

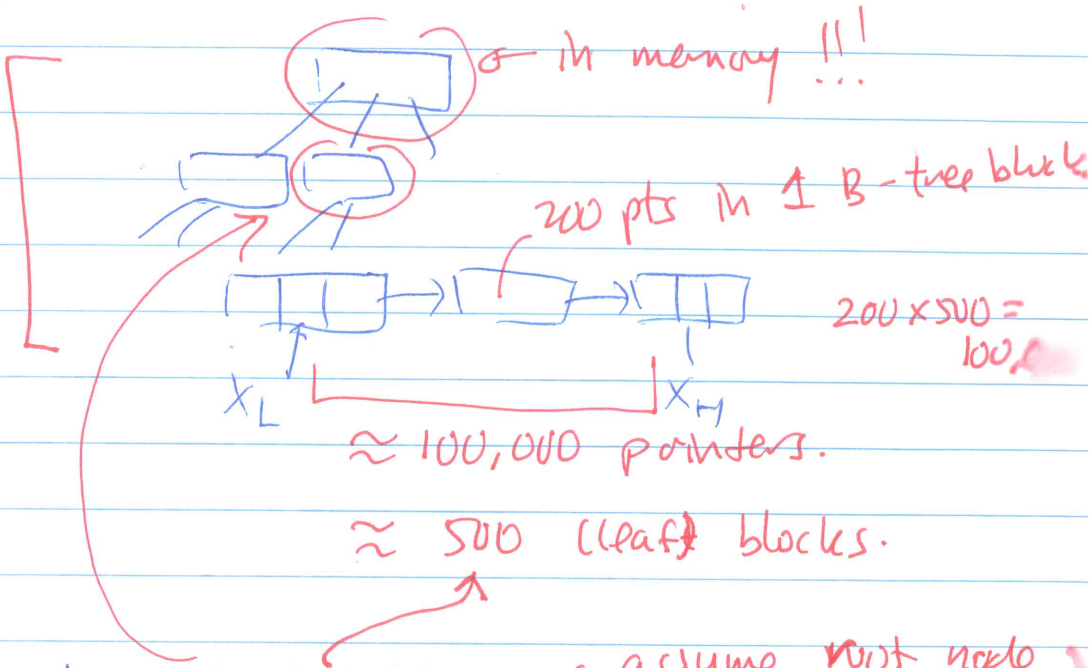
B-tree indexes exist on

X-coordinates

Y-coordinates.

Consider the processing cost using B-trees:

(1) Using the B-tree for x-coord:




# blk reads = 1 + 500 (assume root node is stored in memory)

(2) Use the B-tree to get y-coords:

Same statistics

# blk reads = 1 + 500 = 501.

⇒ # blk reads to get

Intersection = 1002

- After getting appr. 10,000 record ptrs, we still need to:

  access the data in blocks.

- Assuming the data is stored

  randomly

  the 10,000 records would in stored in

  ≈ 10,000 diff. blocks.

  (only 1% of the data, so it's rare they occupy same block).

- Total # blks read:

  100

  + ~ 10,000

  ---

  11,002 blks

Look at the data file:

- 1,000,000 points.
- 1 data blk: 100 pts.

SU: # blocks used = $\frac{1,000,000}{100}$

= 10,000 blks.

- In other words:

If we just SCAN the file and test for x, y coordinates, the # blks read used is:

10,000 blk.

⇒ Having index and limit to 1% of the points, give us no improvement!!!