Using one-dimensional indexes

- We could use one-dimensional indexes for such data
  - However, we will see that this may be inefficient
  - Or rather, a different access method may be more suitable to the data
- Example 1: Range Query (also called a Window Query)
  - Recall: $WQ(I^d) = \{o \mid I^d \cap o.G \neq \emptyset\}$
  - A one-dimensional range query (d=1) specifies a range of values (low and high)
  - Two-dimensional query (d=2) specifies ranges along the x-dimension and y-dimension
  - Idea: Using ranges for both dimensions, compute pointers for tuples in each dimension, then return their intersection

1D index: range query

- In our example, suppose:
  - We have 1,000,000 points
  - These are distributed randomly
  - $0 \leq x \leq 1000$; $0 \leq y \leq 1000$
  - 100 point tuples fit on a disk block
  - Our B+-tree indexes for the table has 200 key-pointer pairs (we have an index for x-coordinates and one for y-coordinates)
- Our query will be this:
  - "Return all points in the centre of the space such that $450 \leq x \leq 550$ and $450 \leq y \leq 550$."

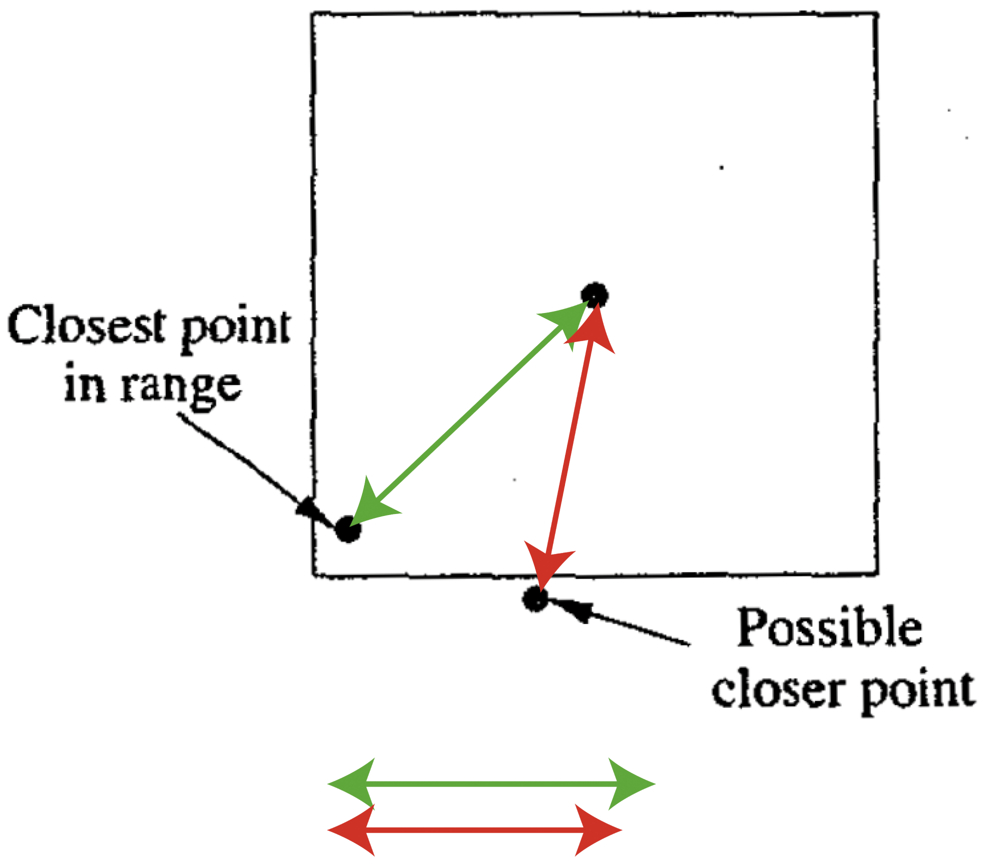
1D index: range query

- Using the B+-tree index for x:
  - Assuming the 1,000,000 points are distributed randomly...
  - ... and knowing that our range is 10% of the total x-dimension...
  - ... then we can estimate there will be 100,000 pointers returned from the index.
- Same reasoning applies for B+-tree index for y:
  - We will have 100,000 pointers
- Approximately 10,000 pointers will be in the range
- Disk I/Os:
  - 500 for x-index (i.e., 100,000 / 200 pointers per block)
  - 500 for y-index (ditto)
  - Assume one interior node also traversed for each index
  - Total: 1002 disk block I/Os just for index

Using one-dimensional indexes

- Example 2: Nearest-Neighbour query
  - Recall: NNQ(o') = {o | $\forall$ o" : dist(o'.G, o.G) $\leq$ dist(o'.G, o".G)}
- Idea here:
  - First perform a range query around the point...
  - ... then find closest point within the range.
- What could possibly go wrong?
  1. No point within the selected range
  2. Closest point in the range might not actually be closest point!

"Closest point" ain't the closest!



- Modified scheme is needed
  - Begin by estimating range.
  - If no points found, expand the range
  - If a point is found, check if a closer one is in a region adjacent to the range
    - If so, expand the range once more, retrieve all points, and check

Hashing for multidimensional data

- (We will look at graph schemes later)
- **Grid Files**
  - Values along dimensions are not hashed.
  - Rather, each dimension is partitioned
  - (That is, values are sorted along dimensions)
- Intuition:
  - Think of space of points as being partitioned in some grid.
  - Lines across the grid (vertical and horizontal) make up those partitions.
  - Viewed from one dimension along, the structure is made up of **stripes**
  - Number of grid lines in each dimension can vary.
  - An (x,y) value therefore hashes into a grid partition (in which we then search for the actual tuple)

Running example

Who buys a new game console every year?

(Maybe doing some market analysis for EBGames or somesuch.)

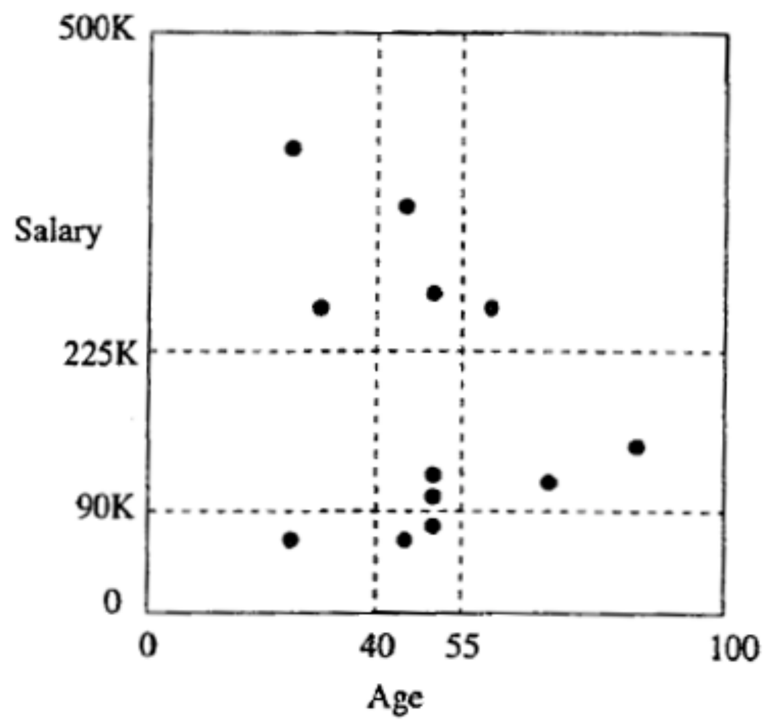
Only concerned here with a customer's age and yearly salary/wage.

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

A grid file

- Twelve points from example are plotted in the graph.
- Some grid lines have been selected for each dimension
  - Notice that the number of points per partition is low with these lines.
- Central rectangle represents customers satisfying:

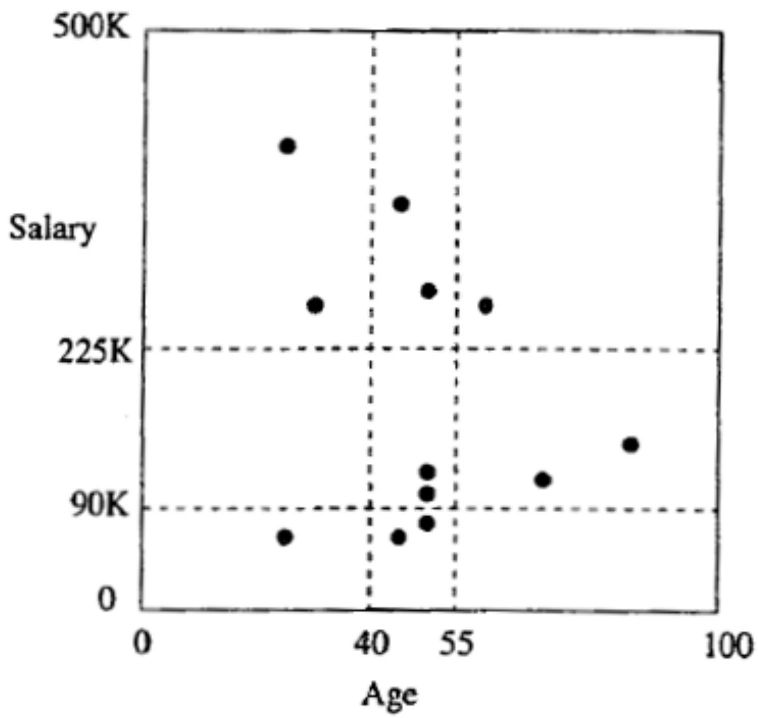
  $40 \leq$ age < 55
  $90 \leq$ salary < 225

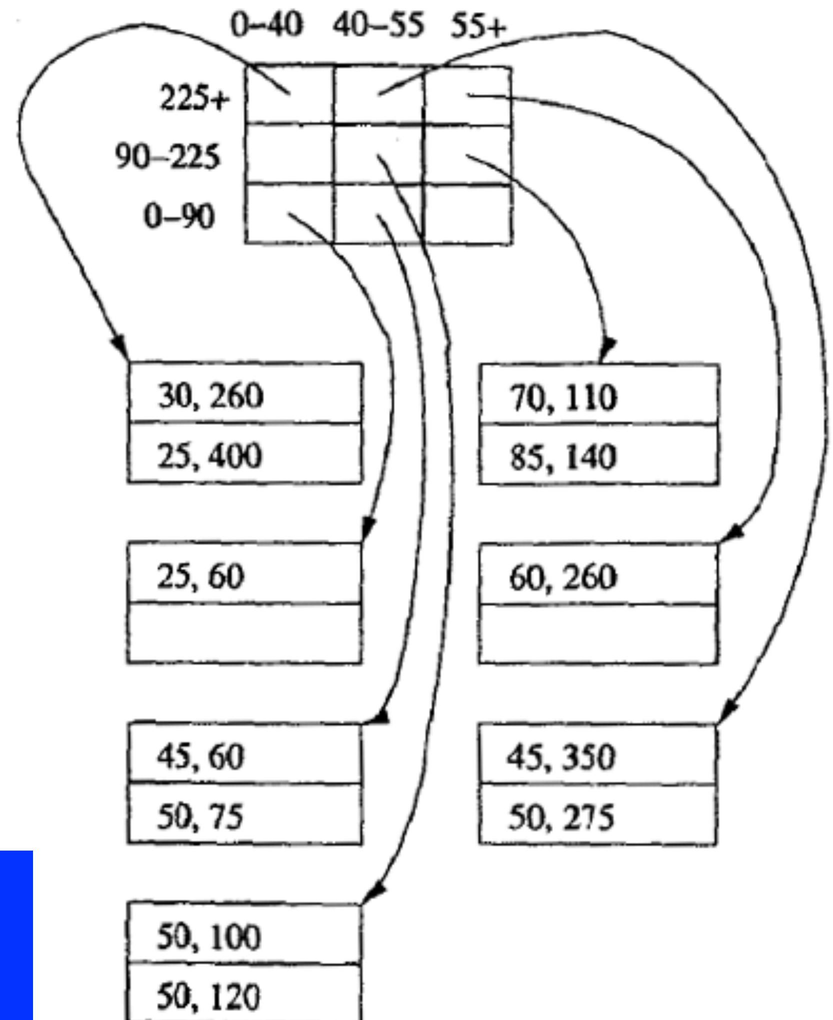


Grid file: lookup

- Each partition behaves, in essence, like a hash bucket
  - Each point in the partition will be stored in block corresponding to that bucket.
  - Overflow blocks can be used as needed.
- Regular hashing using a one-dimensional array of buckets.
- Here we use a two-dimensional bucket
  - Number of entries in each dimension corresponds to number of partitions in each dimension
  - To locate a point's bucket, we need to know the grid line positions for each dimension

Grid file (showing array & buckets)





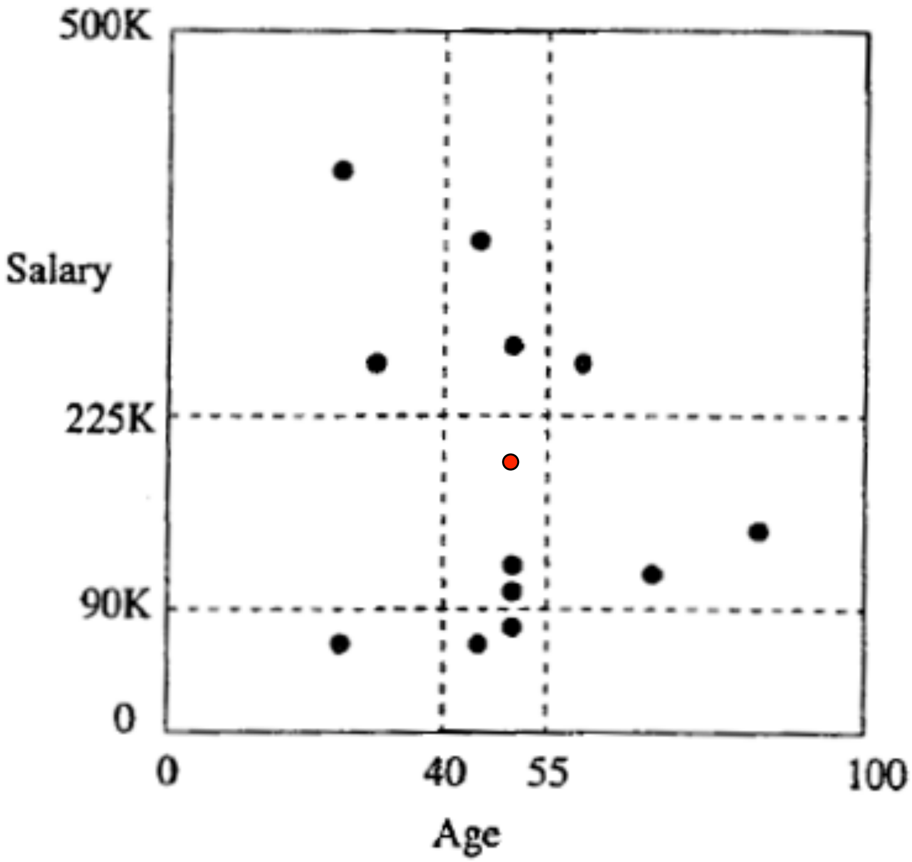
Note that two buckets are empty:
salary $\leq$ 90K $\wedge$ 55 $\leq$ age
90K $\leq$ salary < 225K $\wedge$ 0 $\leq$ age < 40

Grid file: insertion

- To insert, we initially perform the bucket lookup.
- If there is room in the bucket, place new tuple in the bucket.
- If there is no room in the bucket, then a decision must be made:
  1. Add overflow blocks to the bucket, or
  2. Change the partitioning of the file by adding or moving grid lines
- Example: add a customer of age 52 and salary 200K.
  – This would result in a bucket moving from two to three entries.
  – Let us suppose we want to keep buckets at two entries.
  – We must add a grid line.

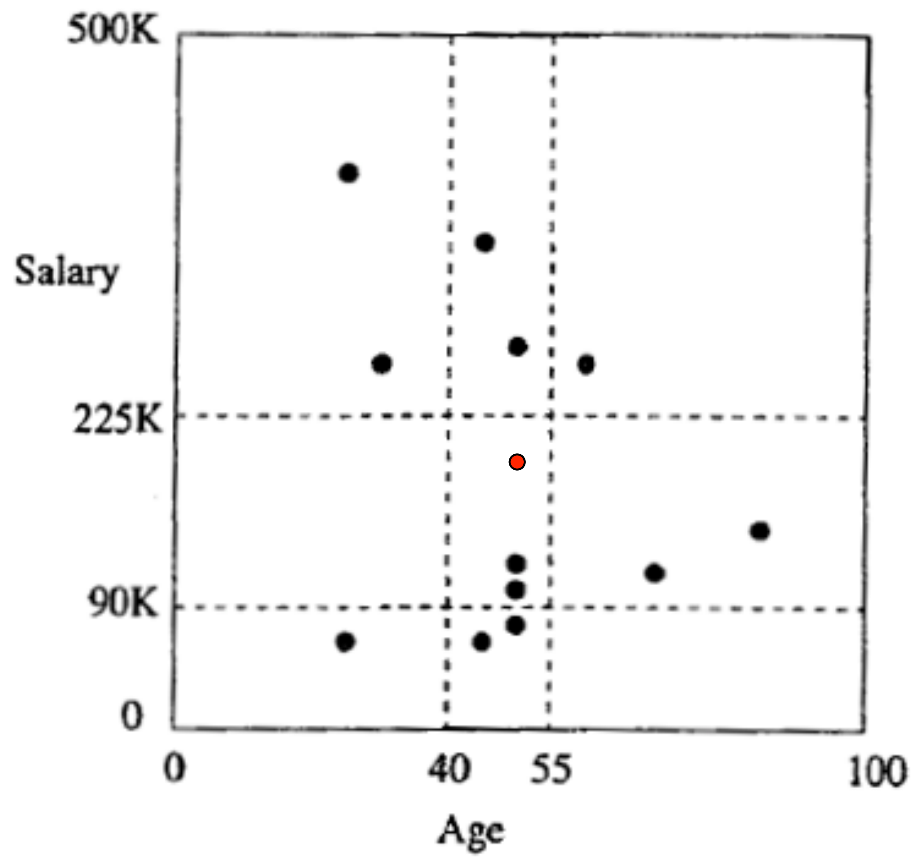
Grid line addition

- We want to split the middl region such that there are two points in one partition and one in another.
- Current data in middle partition:
  - (50,120), (50,100)
- Want to add:
  - (52, 200)
- There are three ways here introduce a grid line
  - Vertical line at 51
  - Horizontal line at, say, 200
  - Horizontal line at, say, 115



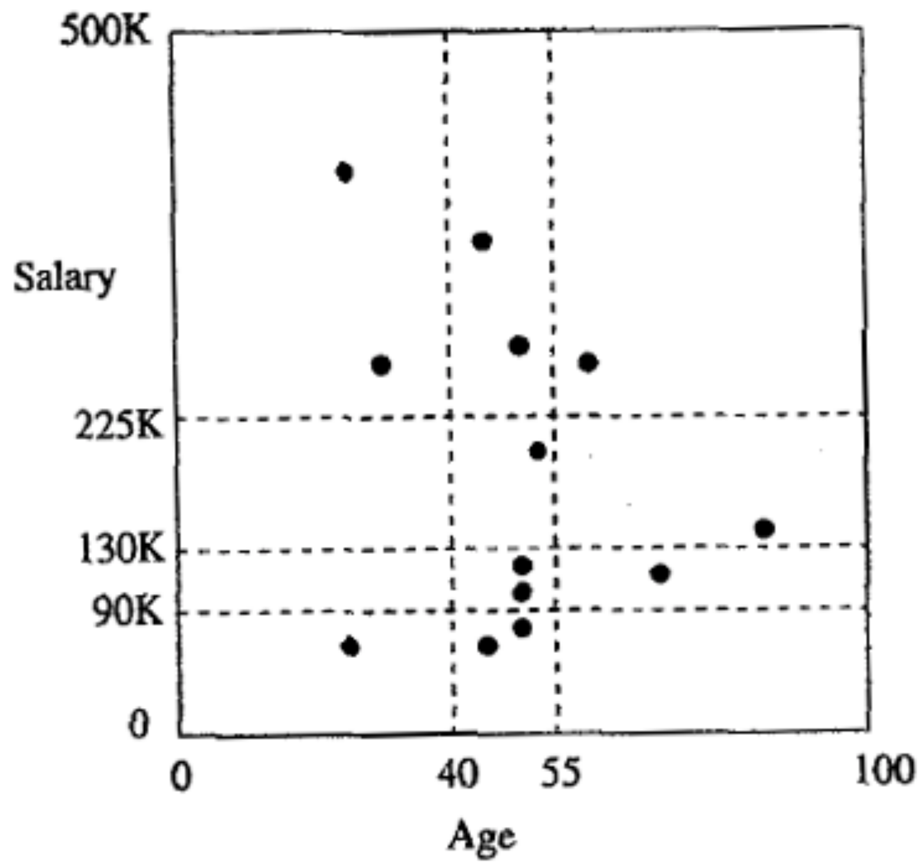
Grid line addition: Some observations

- We want to split the middle region such that there are two points in one partition and one in another.
- A vertical line at age 51 doesn't help us much for buckets above or below (i.e., other buckets' points to left f 51).
- Horizontal line at salary 200 would do the trick, but if we move it down even further (around 130K), we can split the right bucket
- Horizontal line at salary 100 could work, although we should try to split the bucket to the right.
- We'll choose the second option.



Grid line addition: Some observations

- We want to split the middle region such that there are two points in one partition and one in another.
- A vertical line at age 51 doesn't help us much for buckets above or below (i.e., other buckets' points to left f 51).
- Horizontal line at salary 200 would do the trick, but if we move it down even further (around 130K), we can split the right bucket
- Horizontal line at salary 100 could work, although we should try to split the bucket to the right.
- We'll choose the second option.



Grid file: performance

- We make a few assumptions regarding when grid files make sense, regardless of # of dimensions
  - Number of grid buckets can be kept low
  - Can keep indexes in memory...
  - ... and if that isn't possible, at least keep grid-line values in memory (i.e., binary search)
  - Typical buckets contain few overflow blocks
- Operations to consider:
  - Lookup a point
  - Partial-Match query
  - Range query
  - Nearest-Neighbour query

Operations to consider: # of disk I/Os

- Lookup a point
  - Need only find the bucket
  - Read: one bucket
  - Insert/delete: at least two buckets
- Partial-Match query
  - i.e., "Find all console purchasers aged 25"; "find all purchasers with income of $80k"
  - Look at all buckets in a row or a column
  - Number of I/Os can be low or high depending on number of grid lines (but still a fraction of all buckets)

Operations to consider: # of disk I/Os

- Range query (i.e., WG)
  - In our example, d=2, and therefore range is a rectangular area
  - Find all buckets in that area
  - E.g., "Find all customers aged 20 to 35 with a salary of $40K to $90K"
- Nearest-Neighbour query

Operations to consider (contd)

- Range query (i.e., WG)
  - In our example, d=2, and therefore range is a rectangular area
  - Find all buckets in that area
  - E.g., "Find all customers aged 25 to 45 with a salary of $80k to $150k"
  - Must look at buckets surrounded in red
  - If some buckets are completely interior to a search region, then all points belong to the search result.
- Could have lots of I/Os

Operations to consider (contd)

- Nearest-Neighbour query
  - Given search point P...
  - ... search the bucket in which that point belongs.
  - Find candidates in the bucket
- But as we saw earlier, the answer might be in a neighboring bucket
  - Is the distance from P to a border less than P to the closet intra-bucket candidate?
  - Might end up searching many other buckets...