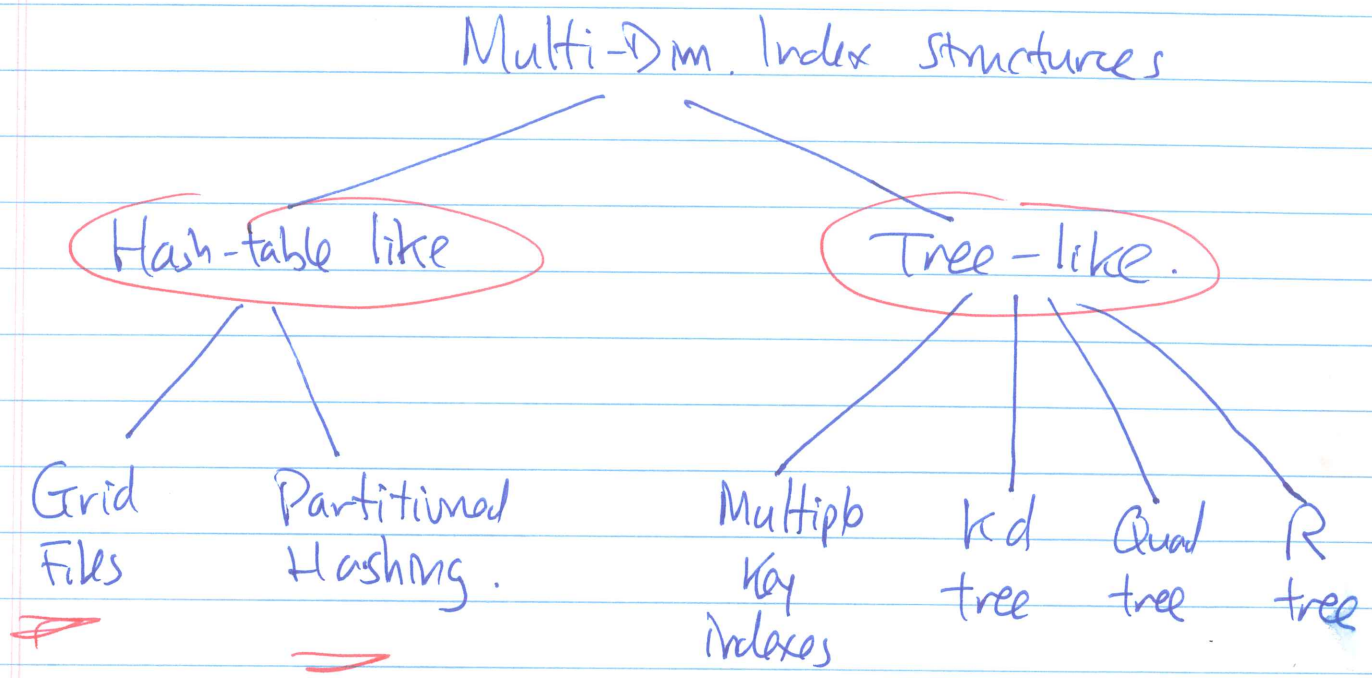
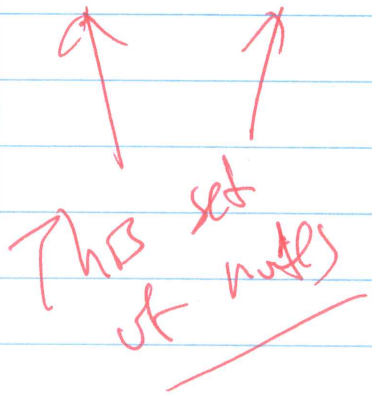
Grid Index Files + Partitioned Hash Functions

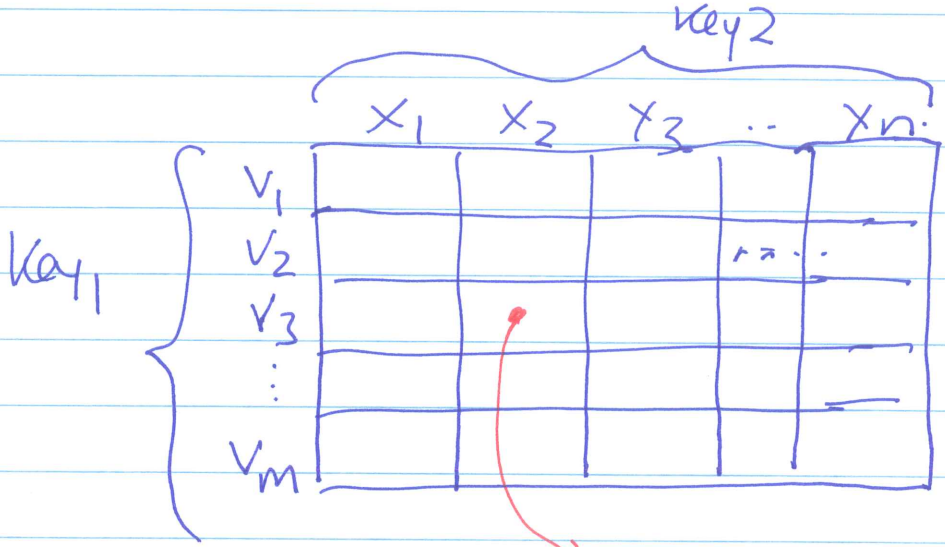
Multi-Dim. Index Structures:



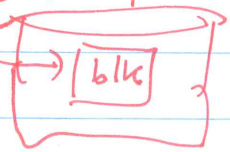


Grid Index (Grid Files)

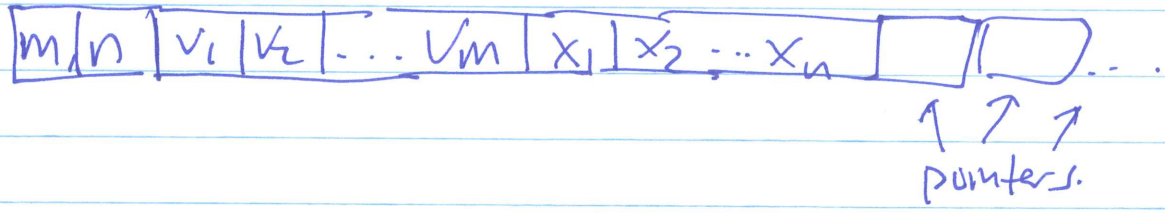
- A Grid Index (File) is an index organized into a 2-dimensional structure:



pointer to record(s) with $(key_1 = V_3, key_2 = X_2)$.



This is a file and saved eg. as follows:

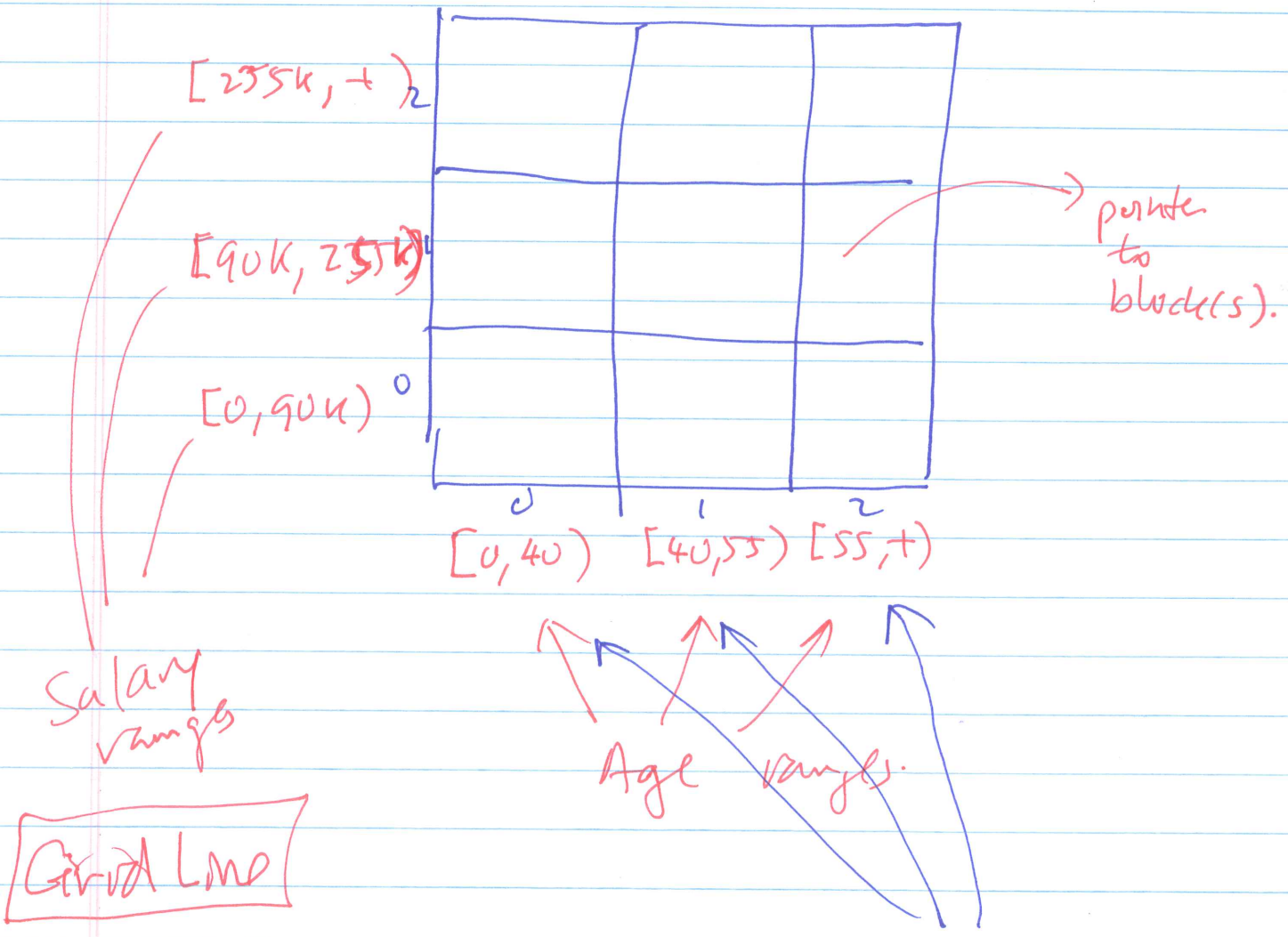


In general, the keys are

ranges

or "buckets".

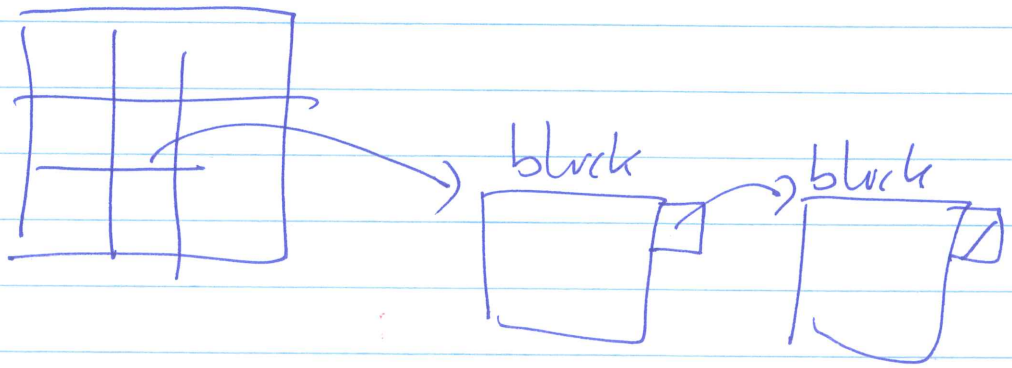
E.g.:



- Ranges are called "Grid lines"

Grid Index file:

- Uses an array whose dimension is the same as for the data file.
- Store - for each dimension - the list of values at which the "grid lines" occur.
- Uses overflow buckets: blocks.



overflow block.

Example index:

- Suppose we have a bunch of records of people buying jewelry
- Assume the only relevant attributes are:

(name, address, age, salary)

Important properties to categorize buyers

- We want to index the information on:

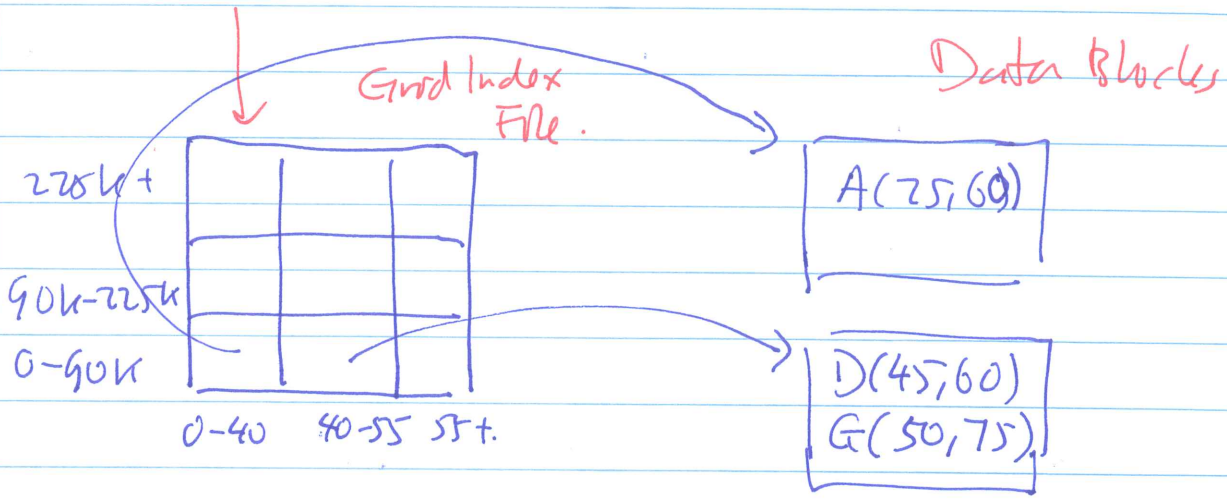
(age, salary).

- Data (input):

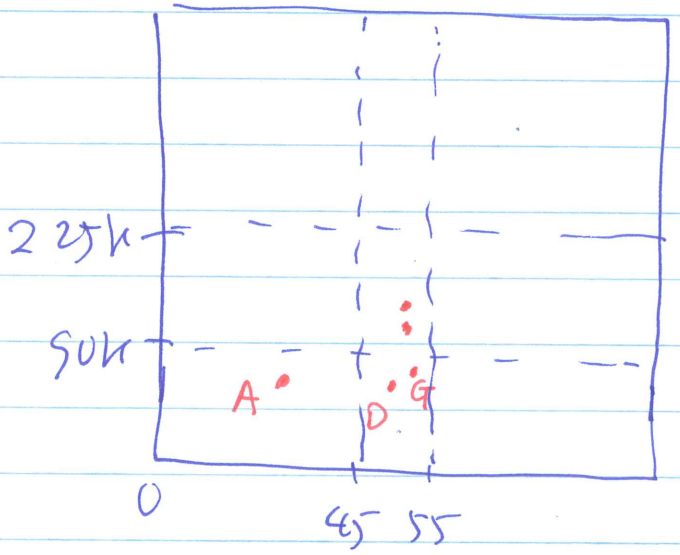
A(25, 6)	D(45, 60)	G(50, 75)	J(50, 100)
B(50, 120)	E(70, 110)	H(85, 140)	K(30, 260)
C(25, 400)	F(45, 350)	I(50, 275)	L(60, 260)

Ranges: Age: 0-40, 40-55, 55+
Salary: 0-90K, 90K-225K, 225K+

Grid Index File for these records:



The book represents (it) as follows:



Lookup in a Grid Index file

Recall: Grid Index file contains:

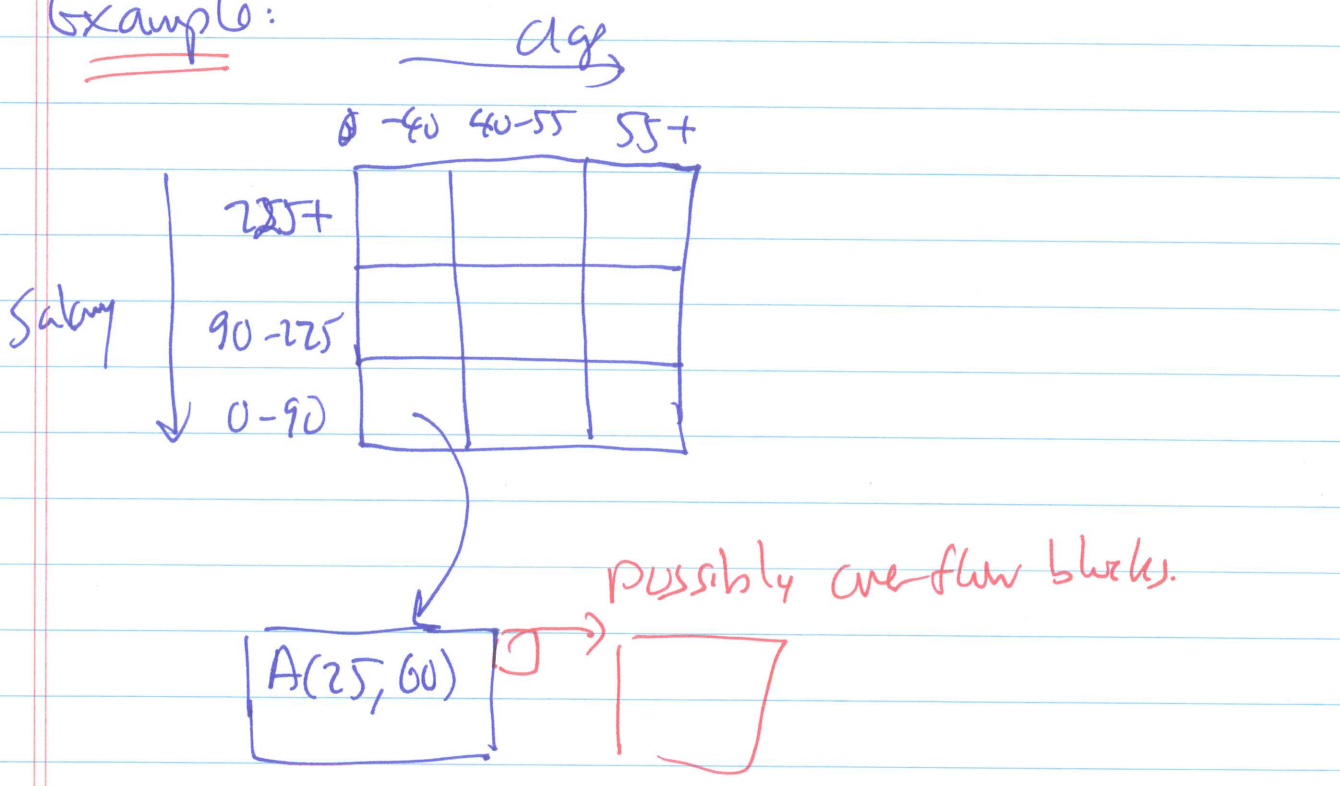
(1) Number of dimensions

(2) List of values at which the grid lines occur in each dimension.

Given: a "point" = value in N-dimension $(v_1, v_2, v_3 \ldots, v_N)$.

$\Rightarrow$ determine the "position" of the "point" in the grid for each dimension.

Example:



Given point:

age = 35 → 0-40
salary = 60 → 0-90

→ read bkt ptr.

Inserting into a Grid Index File

(1) Do Lookup first

→ find blk pointer.

(2) If there is room in data block

⇒ insert

Overflow:

(A) Add overflow block

(B) Reorganize the grid lines.

It's like "dynamic hashing"

but more complex.

↙

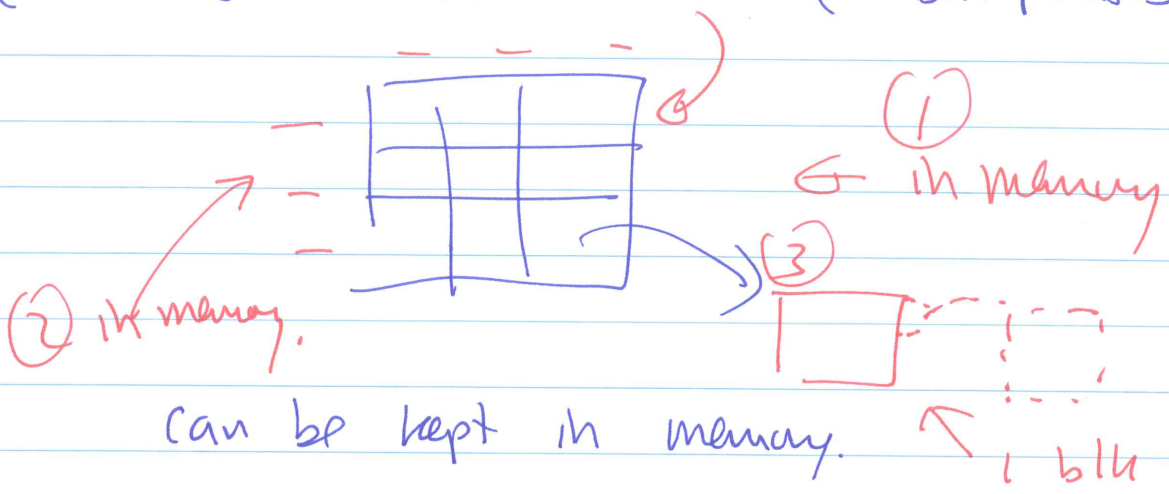
New grid line will divide MANY buckets!!

(Could make many small/empty buckets!).

Performance of Grid Index

Assumptions:

(1) The bucket matrix: (grid of blk pointers)



(2) Values of the grid lines

can also be kept in memory.

(3) Few overflow blocks.

(so access to data block ≈ 1 blk)

Performance of Grid Index – numbers.

(1) Look up a specific point $(v_1, v_2 \ldots, v_N)$

∅ blk access to get blk pointer.

1 blk access to get data block.

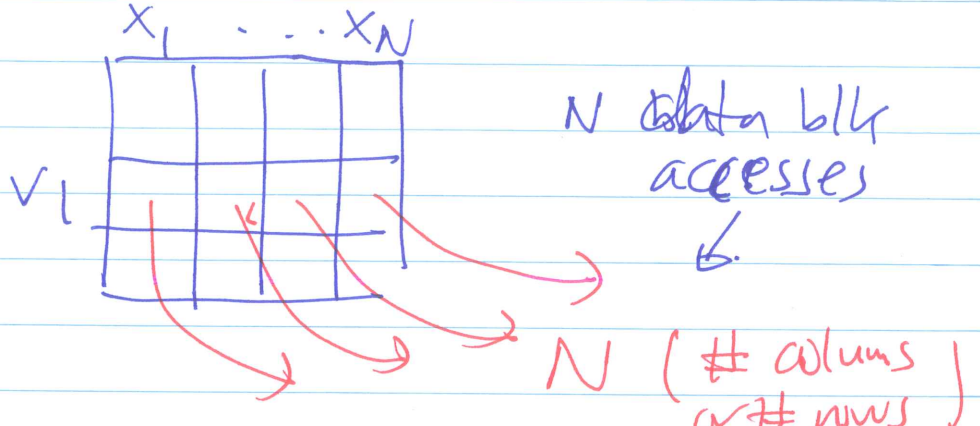
Insert / Delete:

may require an additional Write blk.

(or overflow).

(2) Partial Match query: $(v_1, *)$

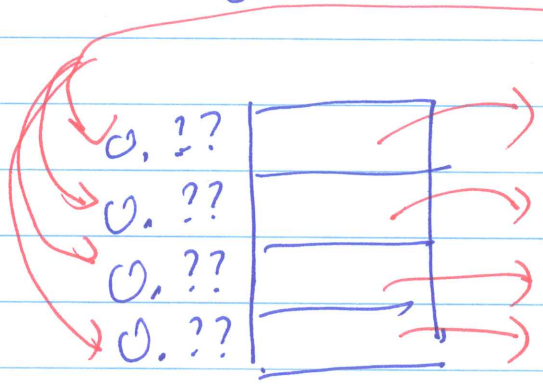
∅ blk access to get blk pointers



- Partial Match search: (example)

age = 50 (given) Salary = * (unknown.)

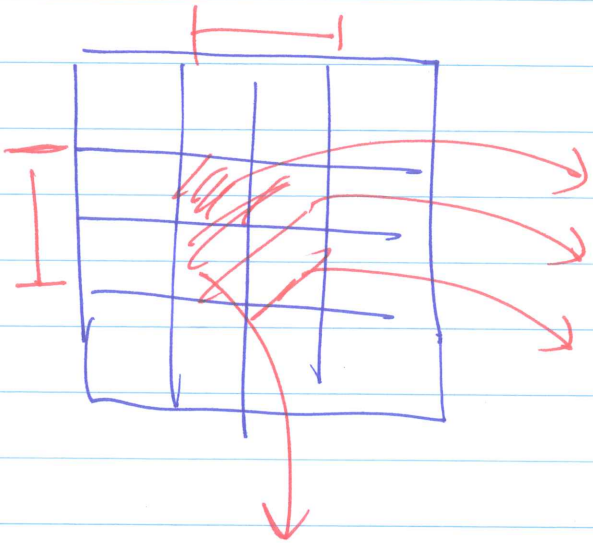
Records for age = 50 are found here:



- Look up specific point: very fast

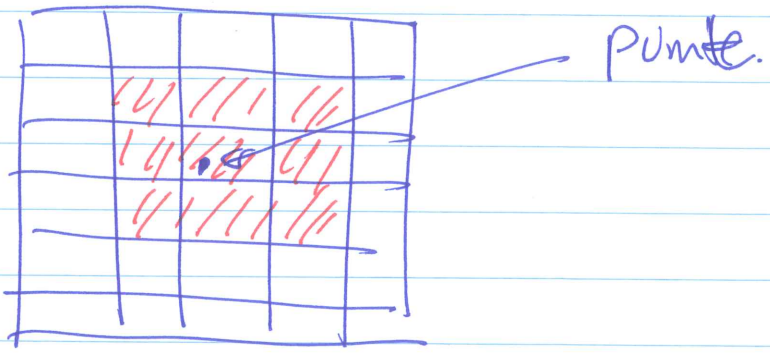
(It's hashing, after all...)

(3) Range Queries:



All data blocks in the range.

(4) Nearest Neighbor query:

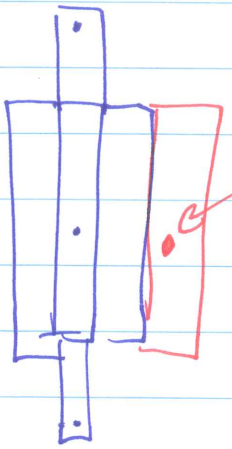


About: 9 data blocks

CAVEAT

It actually depends on the geometry

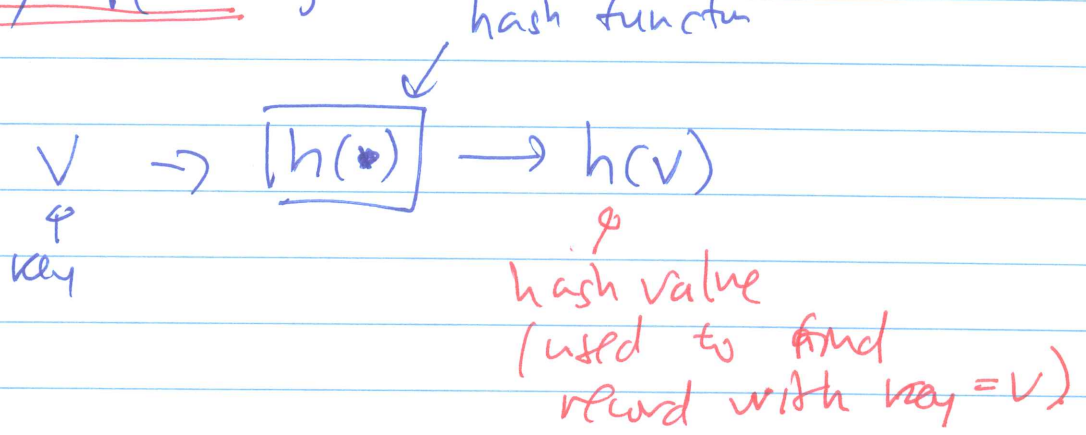
If the Grid is very Stretched:



Nearest Neighbour !!!

Partitioned Hashing:

- Ordinary Hashing:



- Partitioned Hashing:

Key is composite: $(v_1, v_2, \ldots, v_n)$.

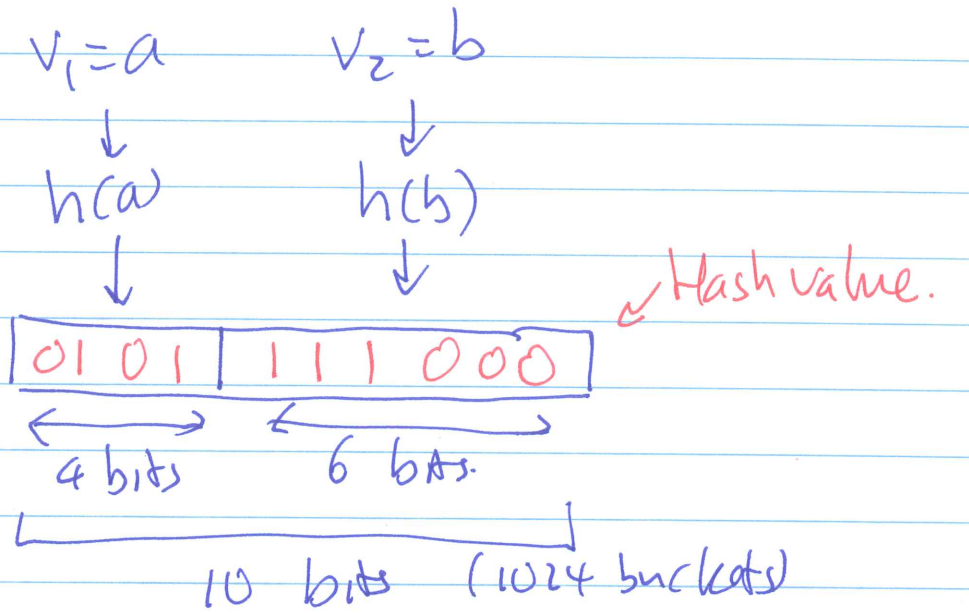
Uses (n) hash functions:

$h_1(\cdot), h_2(\cdot), \ldots h_n(\cdot)$

hash value = concatenation of

$h_1(v_1) \cdot h_2(v_2) \cdot \ldots h_n(v_n)$.

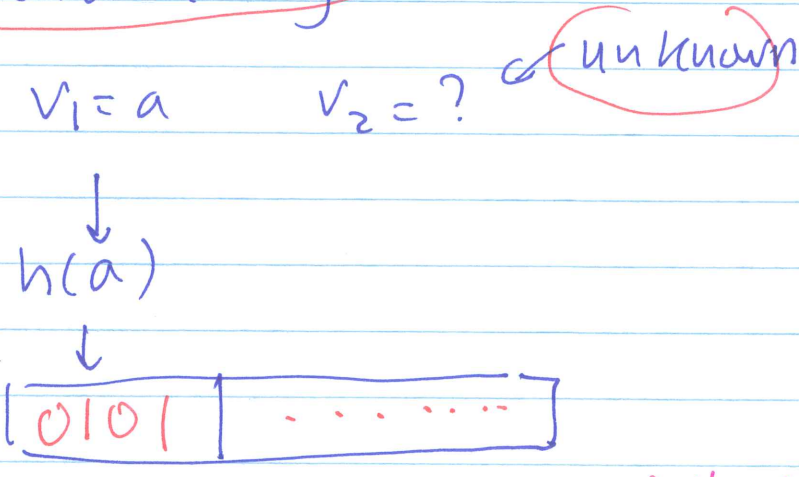
Example:



- Difference between Ordinary Hashing vs Partitioned Hashing

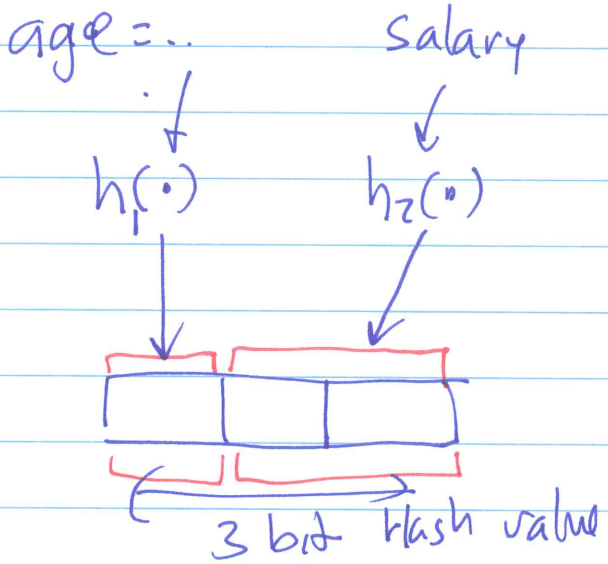
(1) Ordinary Hashing cannot hash when one (or more) keys are missing.

Partitioned Hashing:



↳ you can still get a RANGE (subset) of buckets to search!

Concrete Example: who buys jewelry data

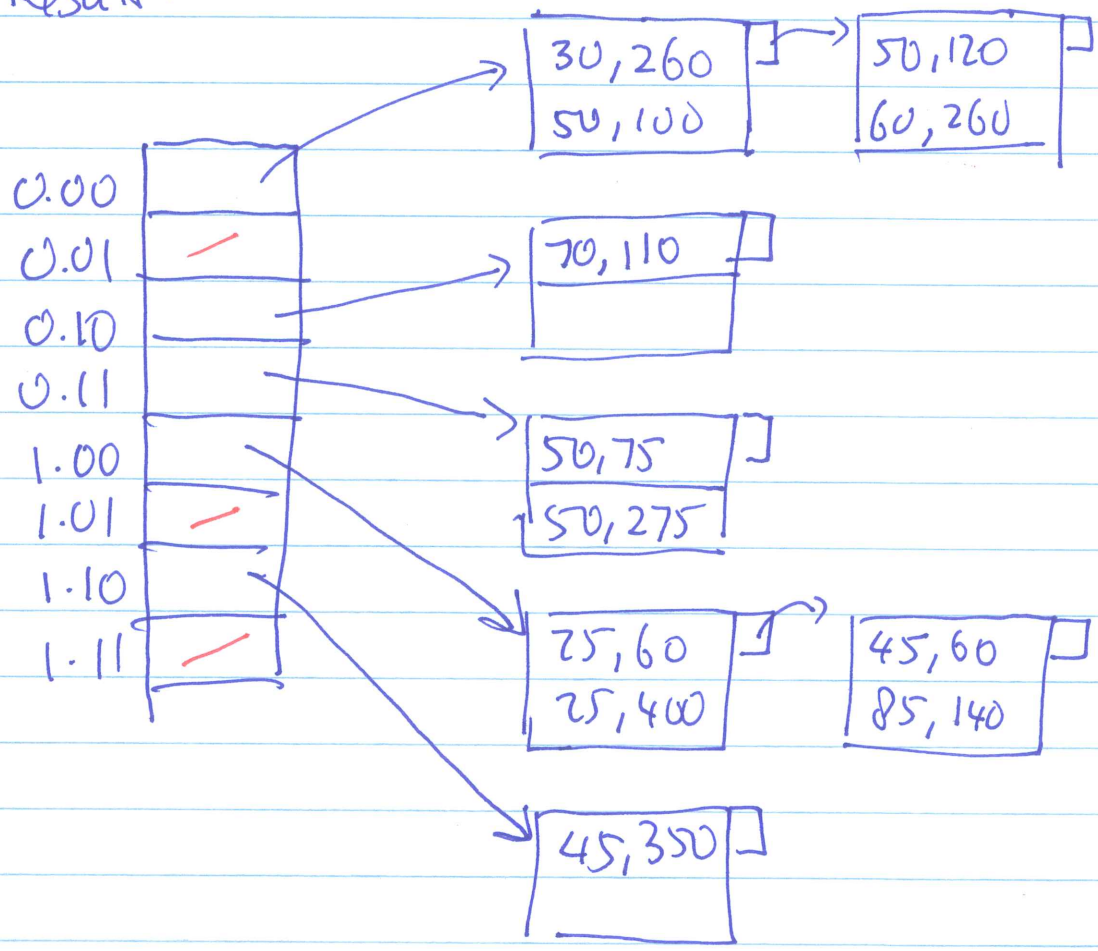


$h_1(\cdot) = age \% 2$

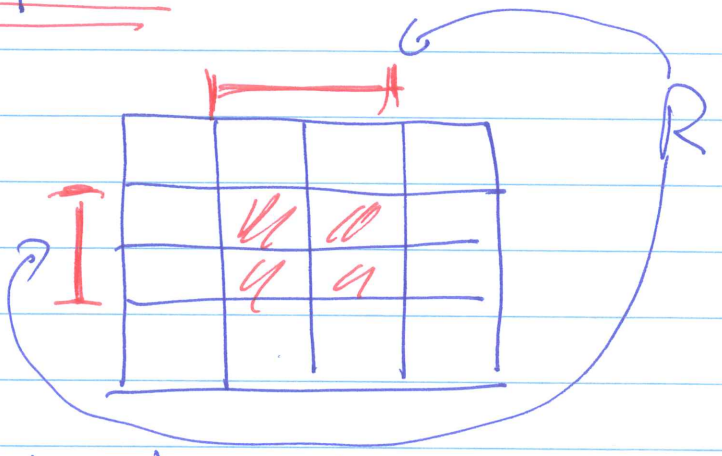
$h_2(\cdot) = (salary/1000) \% 4.$

Assume: 2 records per block

Result:



- Range queries:



  - Hash the values in the range $(V_1, V_2) \in R$ $\rightarrow h(V_1, V_2)$
  - Use the $h(V_1, V_2)$ to access the blocks.

- Nearest Neighbor:

  Partitioned ~~Index~~ Hashing is

  completely useless for Nearest Neighbor

  (because there is no notion of distance in the hash function.

  ↳ closeness of the bucket numbers

  ≠ physical distance between data "points" (records).

Advantage of Partitioned Hashing.

- Good hash function can randomize the records :-

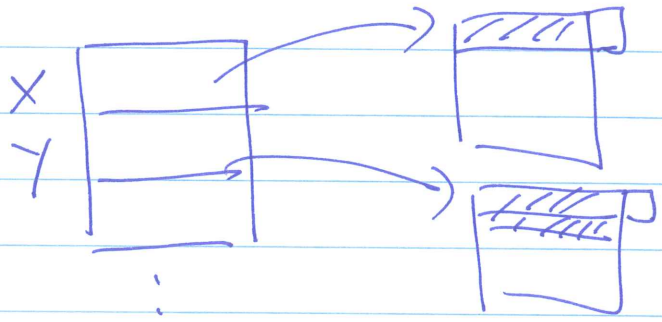
  good occupation rate
  (avg # records per bucket similar for all buckets.

- Grid files (especially with LARGE # dimensions) will have

  many empty buckets.

Handling Tiny Buckets:

Instead:



You can pack different buckets into one block:

Block header must contain information:

(1) bucket ID
(2) records in specific bucket.

Eg: