Database Systems Architecture
Q&A session - 12 december 2014

Stijn Vansummeren

General Course Information

Objective:

To obtain insight into the internal operation and implementation of database systems.

- Storage management
- Query processing
- Transaction management

General Course Information

What you may expect on the exam (1/2)

Upon successful completion of this course, the student should master the following competences:

1. Translating a given SQL expression into the Relational Algebra
2. Improving a relational algebra expression by, where possible, removing redundant joins in select-project-join subexpressions
3. Improving a relational algebra expression by, where possible, (a) replacing cartesian products by joins; and (b) pushing selections and projections
4. Describing and being able to implement traditional secondary-memory index structures (BTrees, Hashing)
5. Being able to describe and demonstrate the shortcomings of traditional index structures with respect to multi-dimensional search keys. In addition, explaining the studied multi-dimensional indexes by means of an example
6. Describing the most important implementation algorithms (one-pass, sorting, hashing, index) for each of the relational algebra operators, as well as judging the cost of each operator, and knowing their limitations of applicability

General Course Information

What you may expect on the exam (2/2)

Upon successful completion of this course, the student should master the following competences:

7. Given a logical query plan and given base statistics about the size and distributions of the database relations, constructing a heuristically optimal physical query plan, by estimating the sizes of the intermediate results and correspondingly comparing the possible implementations. When joins can be reordered, choosing the order with the least cost.
8. Solving exercises on logging
9. Solving exercises on concurrency control
10. Solving exercises on recoverability
11. Being able to reconstruct the studied proofs

General Course Information

Exam

- Written, closed-book
- Example exam on website
- Final grade: on 20 (14 pts exam, 6 pts project)
- You need a score ≥ 10 to pass.

Exam date

- Monday, January 12 from 13h00-16h00, S.UB2.147

Exam visit

- TBD: Friday 23 jan?

Translation of SQL into relational algebra

Question

How do you normalize a SQL query with a subquery having the inequalities($<$,$>$) not proceeded by an ALL or ANY?

Answer. Consider

```
SELECT name FROM MovieExec
WHERE netWorth >
  (SELECT DISTINCT AVG(E.netWorth)
   FROM MovieExec E)
```

$\Rightarrow$???

Translation of SQL into relational algebra

Question

How do you normalize a SQL query with a subquery having the inequalities($<$,$>$) not proceeded by an ALL or ANY?

Answer. Consider

```
SELECT name FROM MovieExec
WHERE netWorth >
  (SELECT DISTINCT AVG(E.netWorth)
   FROM MovieExec E)
```

$\Rightarrow$

```
SELECT name FROM MovieExec
WHERE EXISTS
  (SELECT DISTINCT AVG(E.netWorth)
   FROM MovieExec E
   HAVING netWorth > AVG(E.netWorth))
```

Translation of SQL into relational algebra

Question

Could you explain again the step of decorrelation when translating SQL queries into relational algebra, specially on which cases it's possible or not to do the simplification of removing one of the parts of the join, and which attributes from the subquery need to be explicit on the Join during the decorrelation

Answer.

- First translate the subquery. Let this translation be E_Q. Then translate the FROM clause of the outer query (taking into account context relations if necessary). Let this be E.
- For an EXISTS subquery the decorrelation is given by $\hat{E} \bowtie \pi_L(E_Q)$, where L is the list of all parameters of the subquery and $\hat{E}$ is the from clause where we omit relations that are context relations in E_Q. (EXISTS queries hence allow the optimization)
- For a NOT EXISTS subquery the decorrelation is given by $E \bowtie \pi_L(E_Q)$, where L is as before.. (NOT EXISTS queries hence do not allow the optimization)

Translation of SQL into relational algebra

Question

Would it be possible to solve this example from the exercises that you provided:

```
SELECT F.budget, E.eid
FROM Emp E, Dept D, Finance F
WHERE E.did = D.did AND D.did = F.did
  AND E.hobby = 'yodeling'
  AND D.floor NOT IN
  ( SELECT D2.floor FROM Dept D2, Finance F2
    WHERE NOT D2.dname = 'CID'
      OR  (F2.did = D2.did AND F2.expenses >= ALL
              (SELECT MAX(F3.expenses)
               FROM Finance F3
               WHERE F3.budget = F.budget)
          )
  )
```

Translation of SQL into relational algebra

Question

Would it be possible to solve this example from the exercises that you provided:

Answer

See separate slides. (+ questions below)

Optimization of conjunctive queries

Question

For me it is not clear enough how we found maximal select-project-join subexpressions. Also how we got from the minimal relational algebra expression the optimal query plan.

Answer

By means of the above exercise.

Optimization of conjunctive queries

Question

Q_1 is contained in Q_2. Could you give small example of a canonical db and how we mapped the values?

Answer

Consider

- $Q_1(t) \leftarrow \texttt{MovieStar}(n, a, g, 1960), \texttt{StarsIn}(t, y, n)$
- $Q_2(t) \leftarrow \texttt{MovieStar}(n, a, g, 1960), \texttt{StarsIn}(t, y_2, n_2), \texttt{StarsIn}(t, y, n)$

To check whether $Q_1 \subseteq Q_2$:

- Construct canonical db for Q_1:

$$D_1 = \{\texttt{MovieStar}(x_n, x_a, x_g, 1960), \texttt{StarsIn}(x_t, x_y, x_n)\}$$

- Check whether $x_t \in Q_2(D_1)$. (Illustration on blackboard)

Optimization of logical plans

Question

Would you please kindly explain us the most important algebraic laws that we need to take into account to improve a query plan like the selection-pushing procedure.

Answer (1/3

You only need to consider three laws:

- Recognize joins. E.g., if you have relations $R(A, B)$, $S(B, C)$, $T(C, D)$ then

$$\sigma_{S.C=T.C}\left(\sigma_{R.A=S.B}(R \times S) \times T\right)$$

becomes

$$(R \underset{R.A=S.B}{\bowtie} S) \bowtie T$$

Optimization of logical plans

Question

Would you please kindly explain us the most important algebraic laws that we need to take into account to improve a query plan like the selection-pushing procedure.

Answer (2/3

You only need to consider three laws:

- Push selections. E.g., if you have relations $R(A, B)$, $S(B, C)$, $T(C, D)$ then

$$\sigma_{R.A>50}\left(\sigma_{R.A=S.B}(R \times S) \times T\right)$$

becomes

$$\sigma_{R.A=S.B}(\sigma_{R.A>50}(R) \times S) \times T$$

which becomes (recognizing joins)

$$(\sigma_{R.A>50}(R) \underset{R.A=S.B}{\bowtie} S) \times T$$

Optimization of logical plans

Question

Would you please kindly explain us the most important algebraic laws that we need to take into account to improve a query plan like the selection-pushing procedure.

Answer (3/3

You only need to consider three laws:

- Add projections where possible. E.g., if you have relations $R(A, B)$, $S(B, C)$, $T(C, D)$ then

$$\pi_{R.A,R.B}\left((\sigma_{R.A>50}(R) \underset{R.A=S.B}{\bowtie} S) \times T\right)$$

becomes

$$\pi_{R.A,R.B}\left(\left(\sigma_{R.A>50}(R) \underset{R.A=S.B}{\bowtie} \pi_{S.B}(S)\right) \times \pi_{T.C}(T)\right)$$

Indexes

Question

Insertion into BTree. Could you please show one more example for non-trivial cases: non-leaf overflow, new root case.

Answer

Re-explanation of corresponding slides. (Slide 58 and onwards)

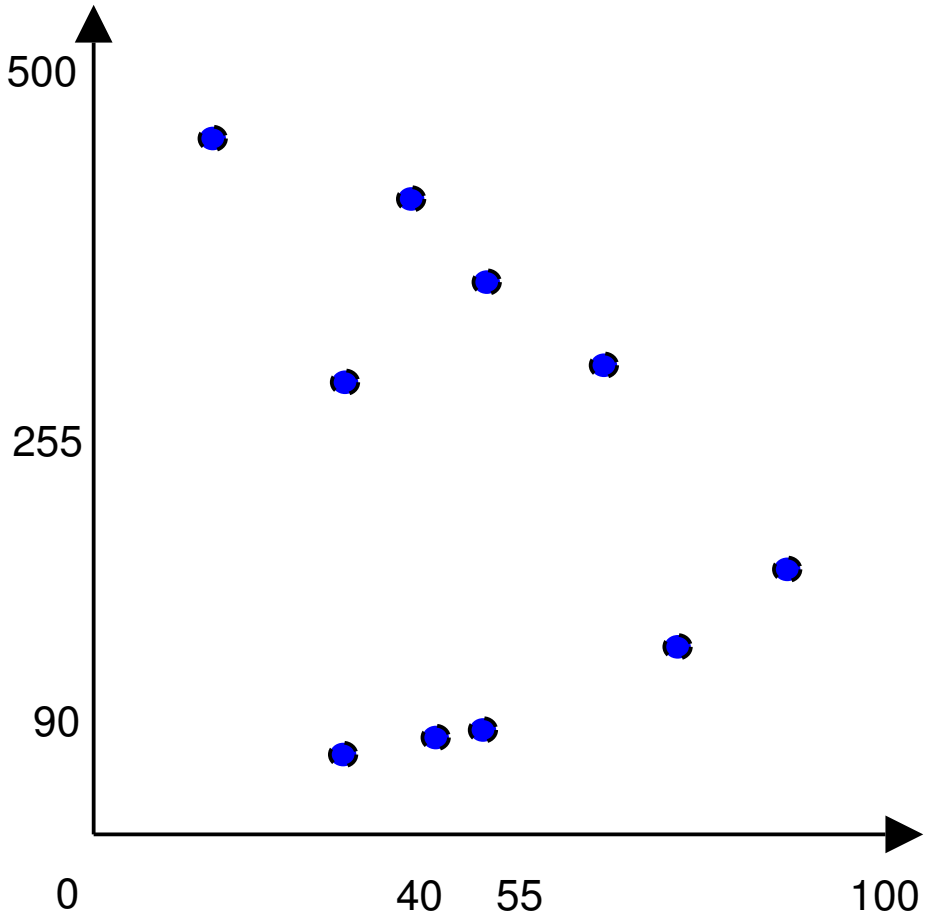
Multidimensional Indexes

Question

Could you please give us some examples related to multi-dimensional indexes using grid files and partitioned hashing, in terms of its performance. I have read the book, but it's still not clear for me.

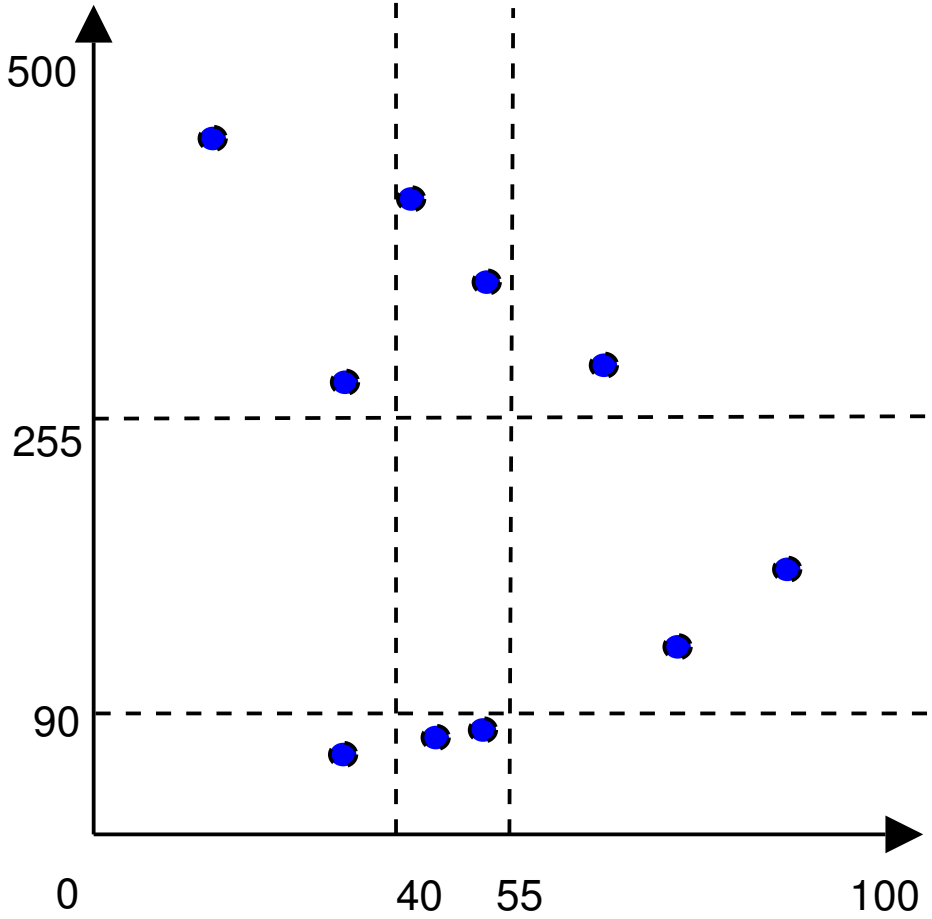
Multidimensional Indexes

Grid files: a variant on hashing



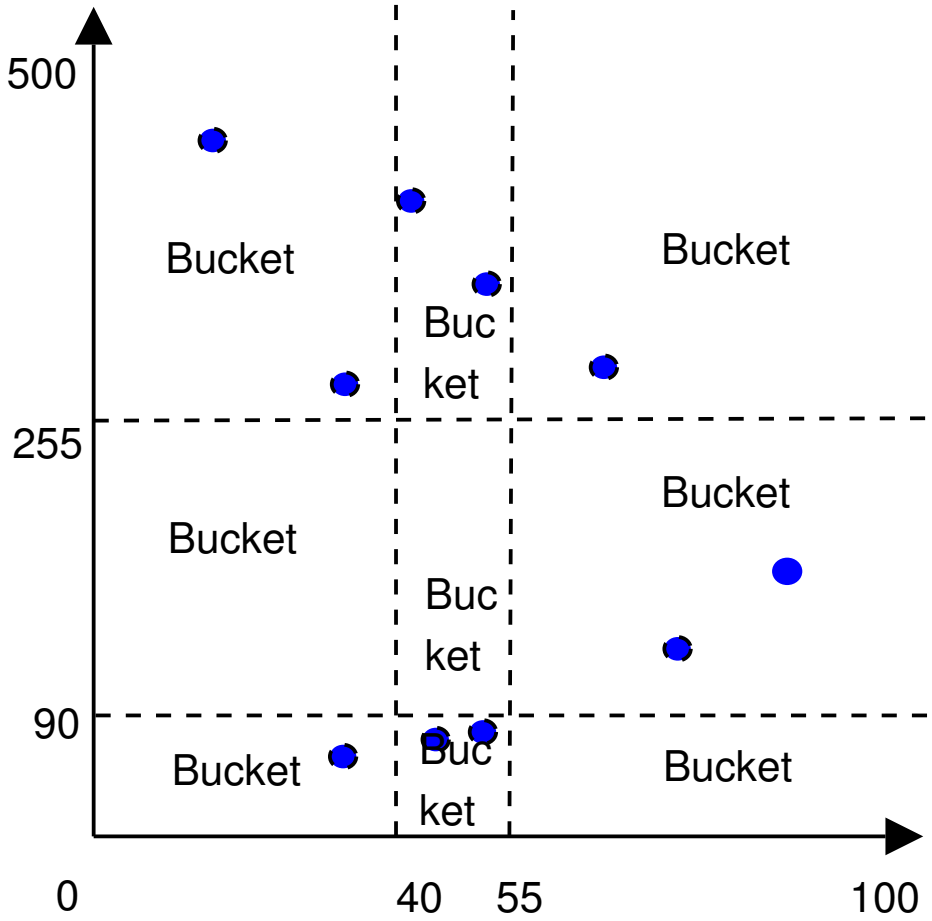
Multidimensional Indexes

Grid files: a variant on hashing



Multidimensional Indexes

Grid files: a variant on hashing



- Insert: find the corresponding bucket, and insert.

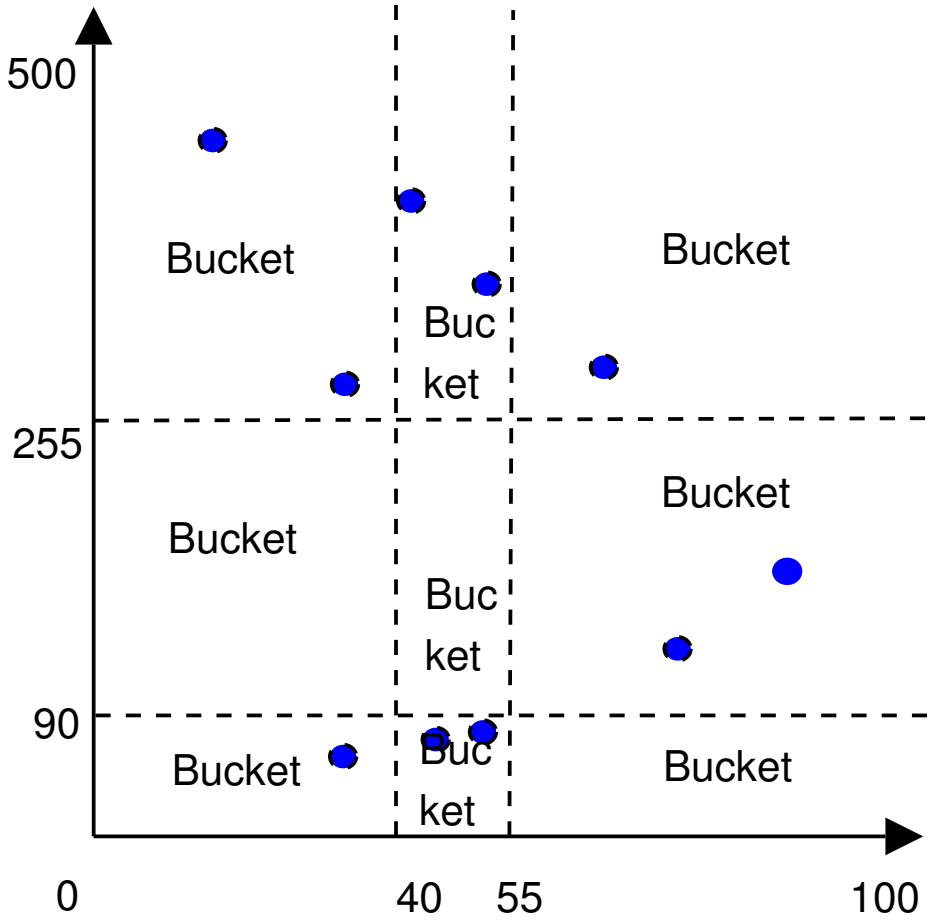
  If the block is full: create overflow blocks or split by creating new separator lines (difficult).

- Delete: find the corresponding bucket, and delete.

  Reorganize if desired

Multidimensional Indexes

Grid files: a variant on hashing



- Good support for point queries
- Good support for partial match queries
- Good support for range queries

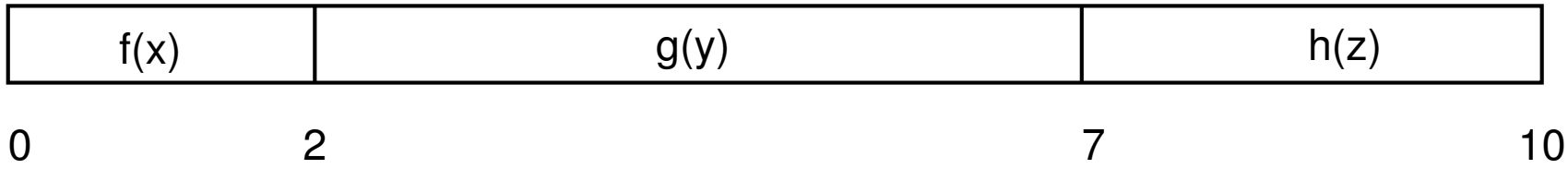
  $\rightarrow$ Lots of buckets to inspect, but also lots of answers
- Reasonable support for nearest-neighbour queries

  $\rightarrow$ By means of neighbourhood searching
- But: many empty buckets when the data is not uniformly distributed

Multidimensional Indexes

Partitioned Hash Functions

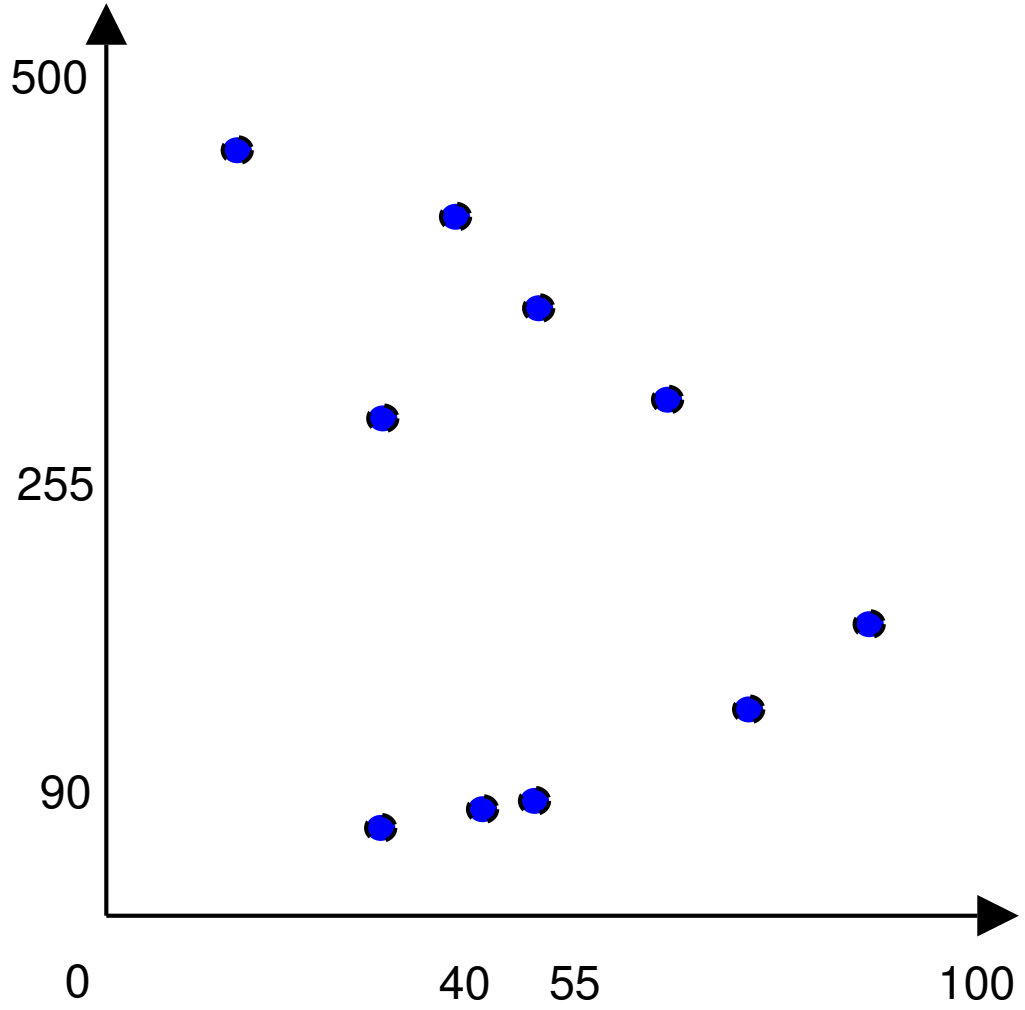
Assume that we have 1024 buckets available to build a hashing index for (x, y, z). We can hence represent each bucket number using 10 bits. Then we can determine the hash value for (x, y, z) as follows:



- Good support for point queries
- Good support for partial match queries
- No support for range queries
- No support for nearest-neighbour queries
- Less wasted space than grid files

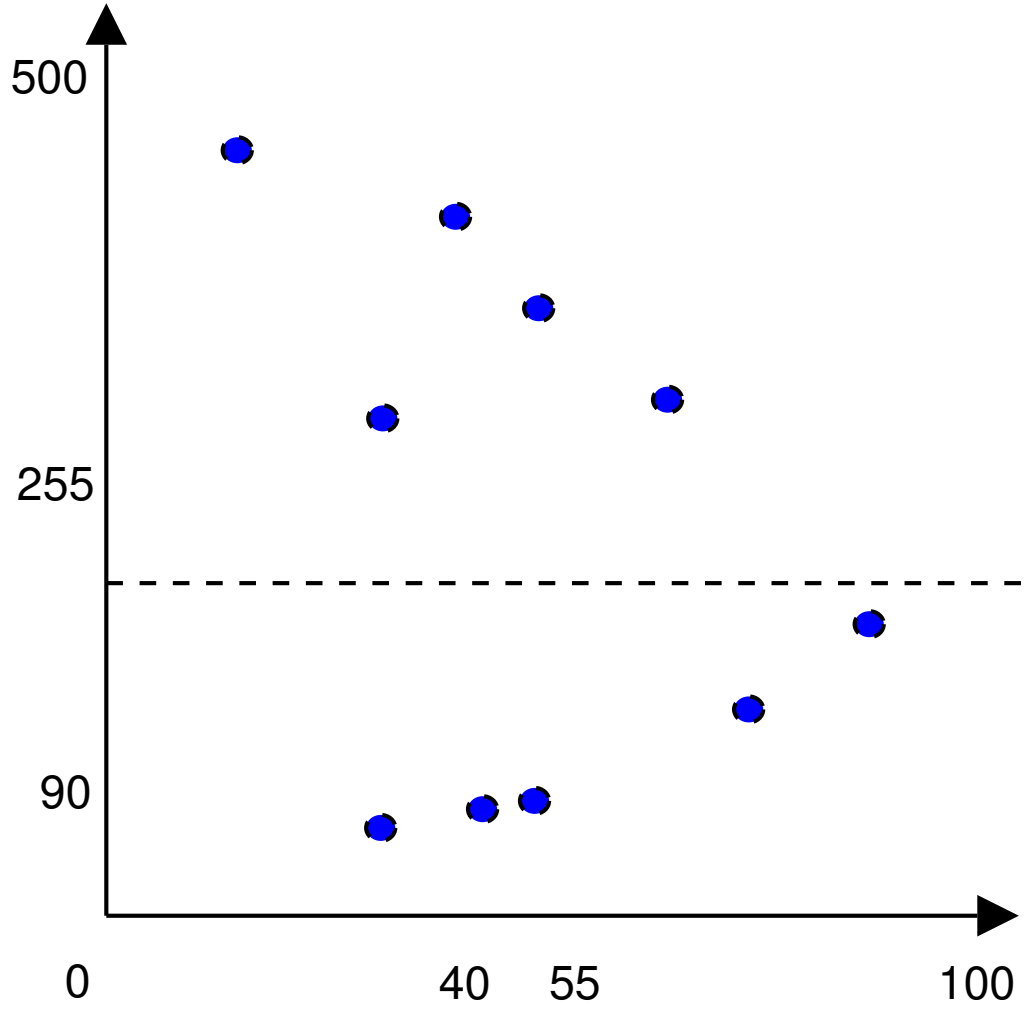
Multidimensional Indexes

kd-Trees



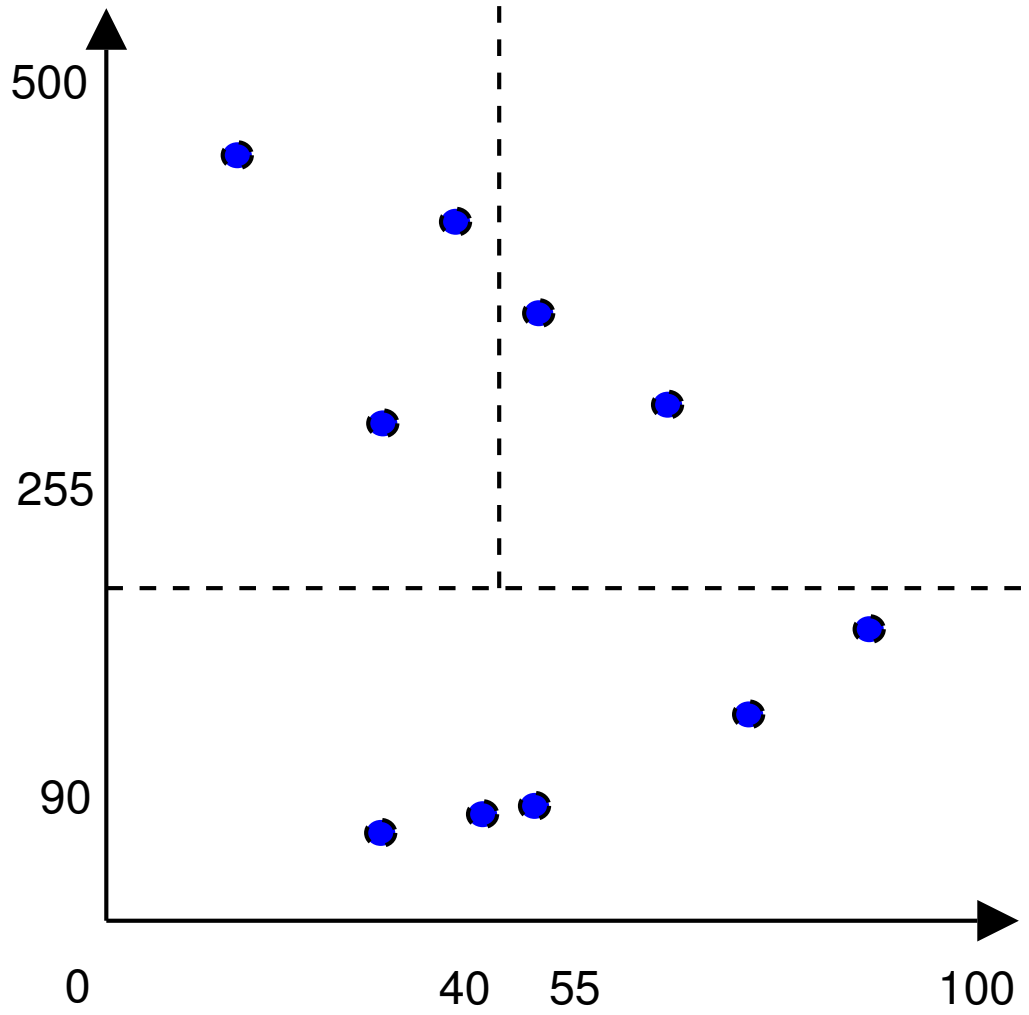
Multidimensional Indexes

kd-Trees



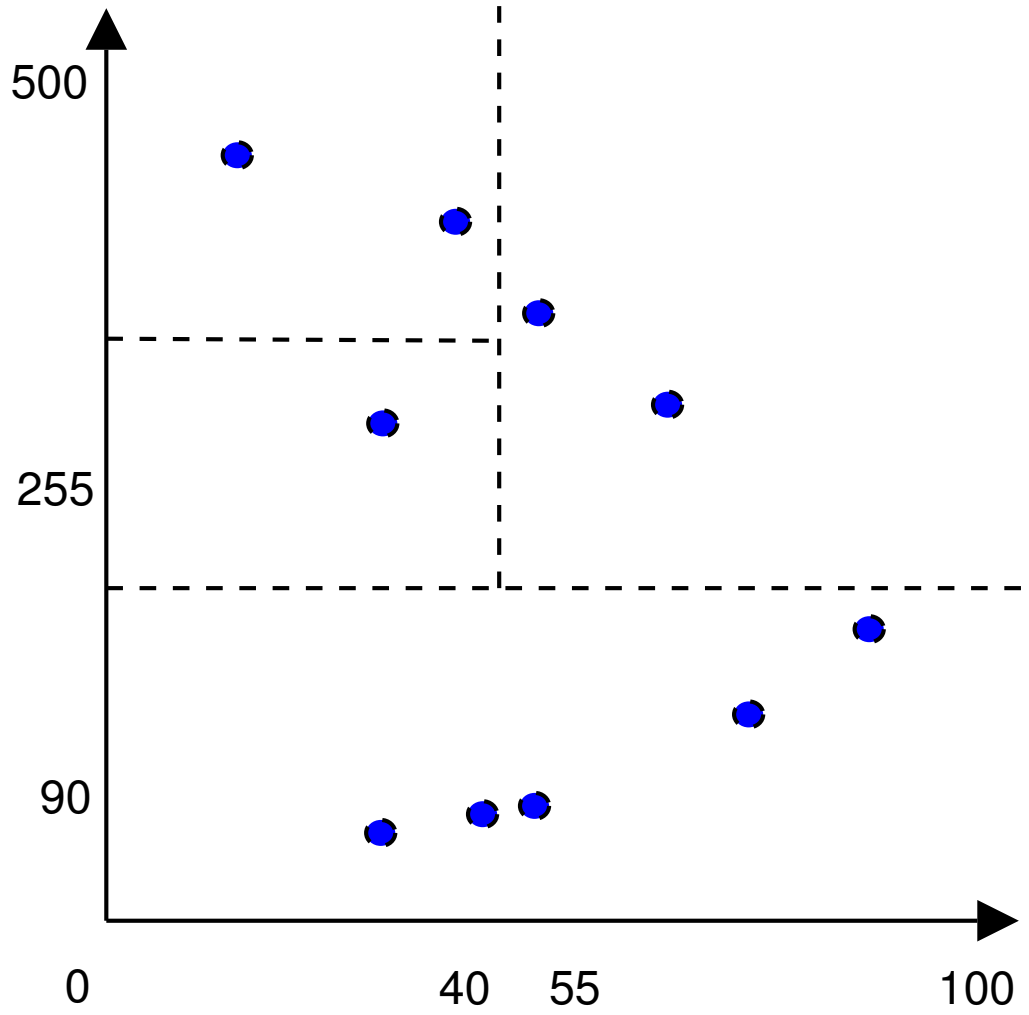
Multidimensional Indexes

kd-Trees



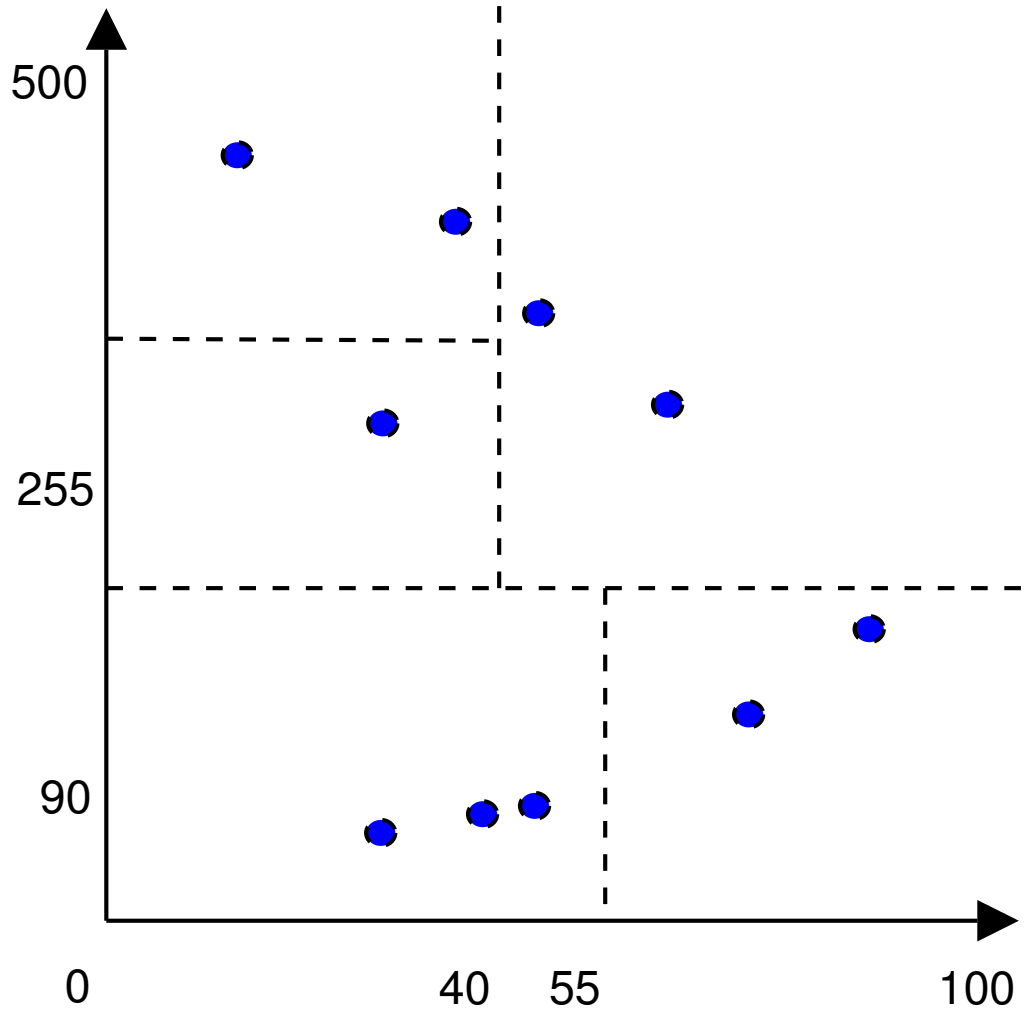
Multidimensional Indexes

kd-Trees



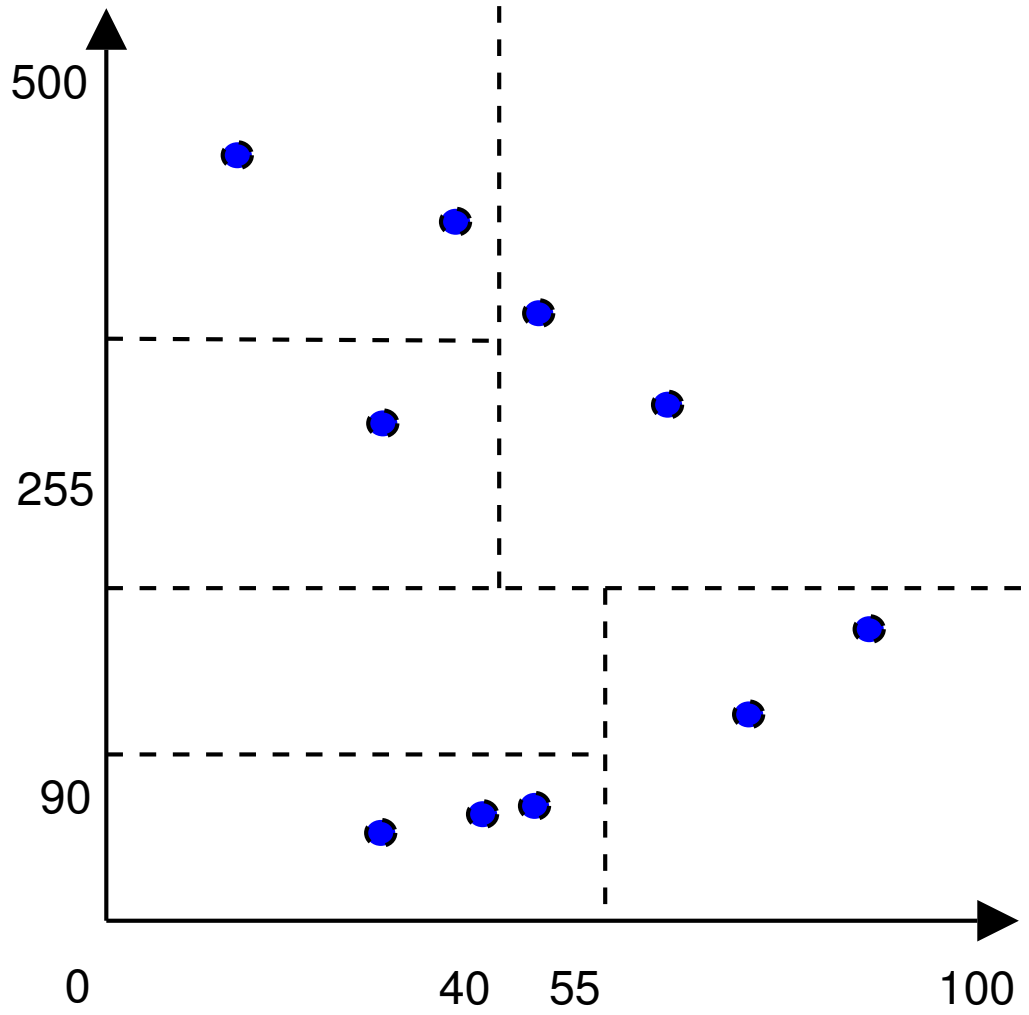
Multidimensional Indexes

kd-Trees



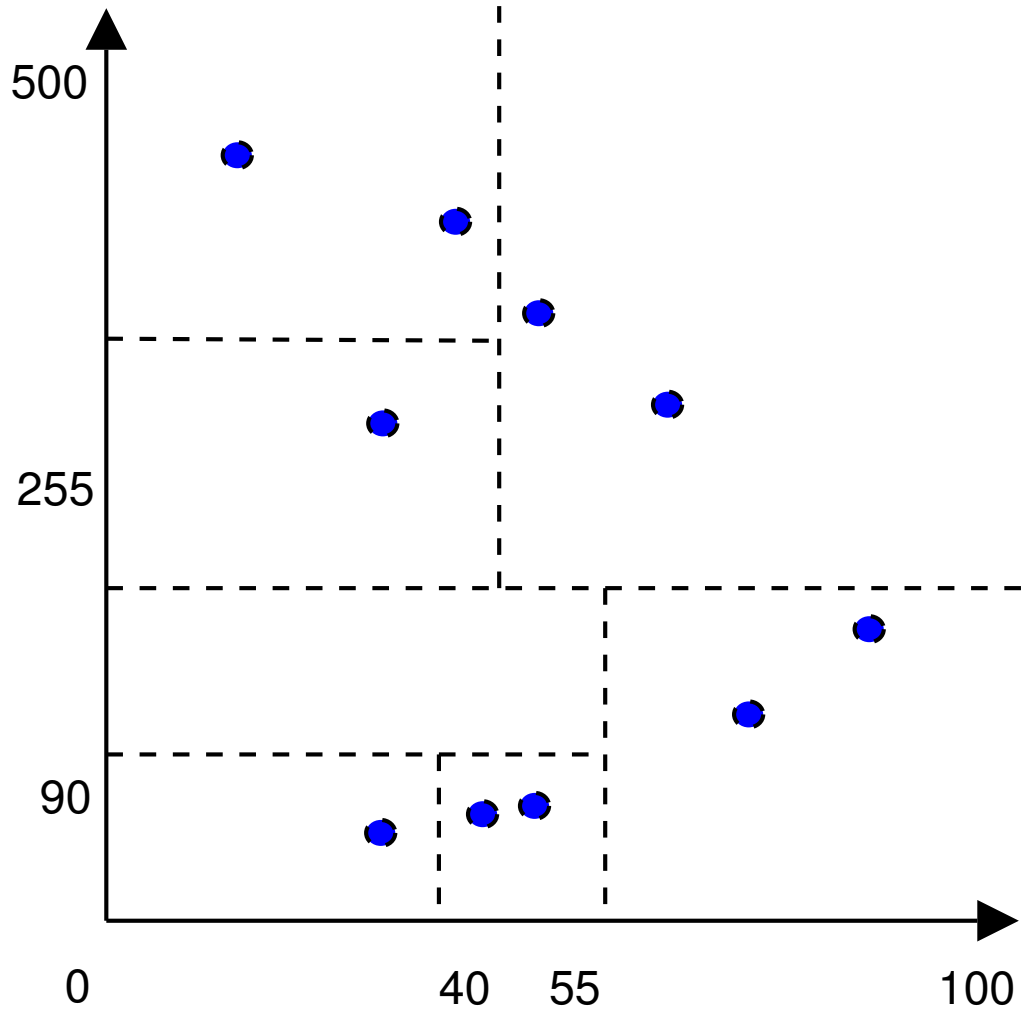
Multidimensional Indexes

kd-Trees



Multidimensional Indexes

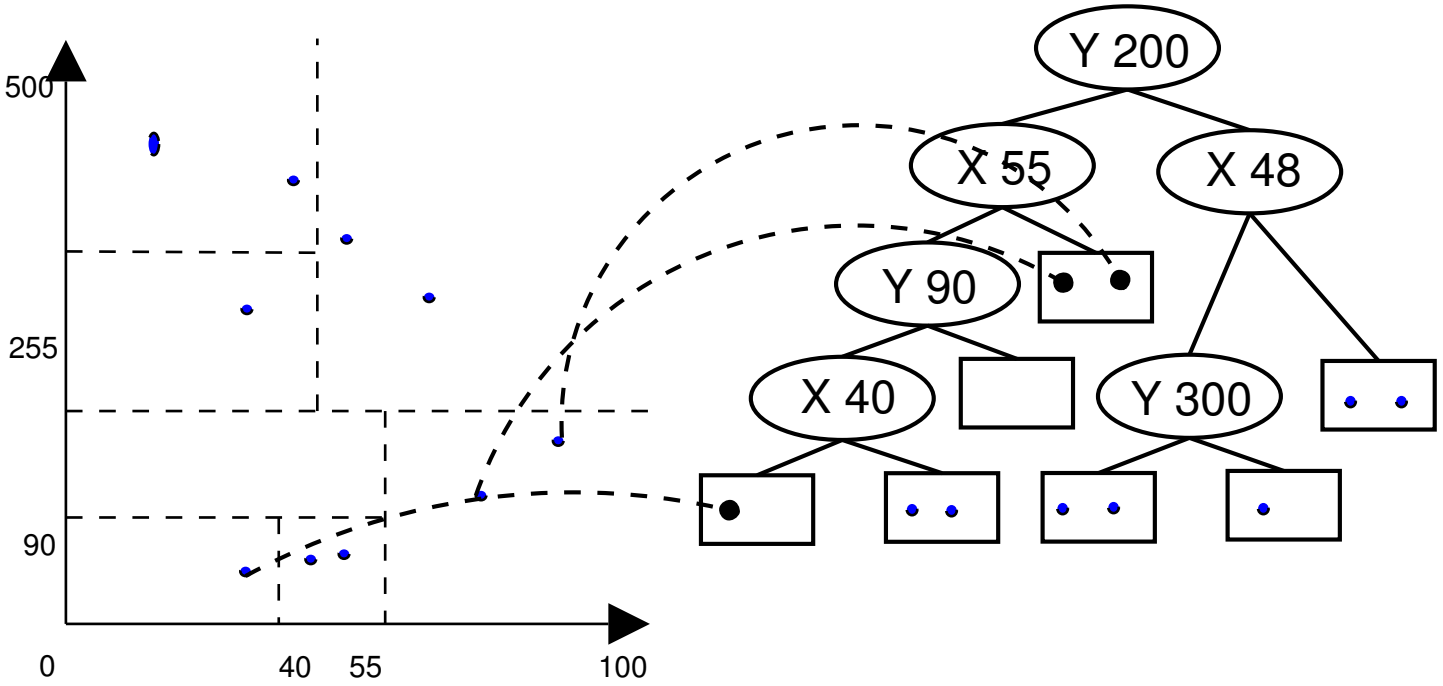
kd-Trees



Multidimensional Indexes

kd-Trees

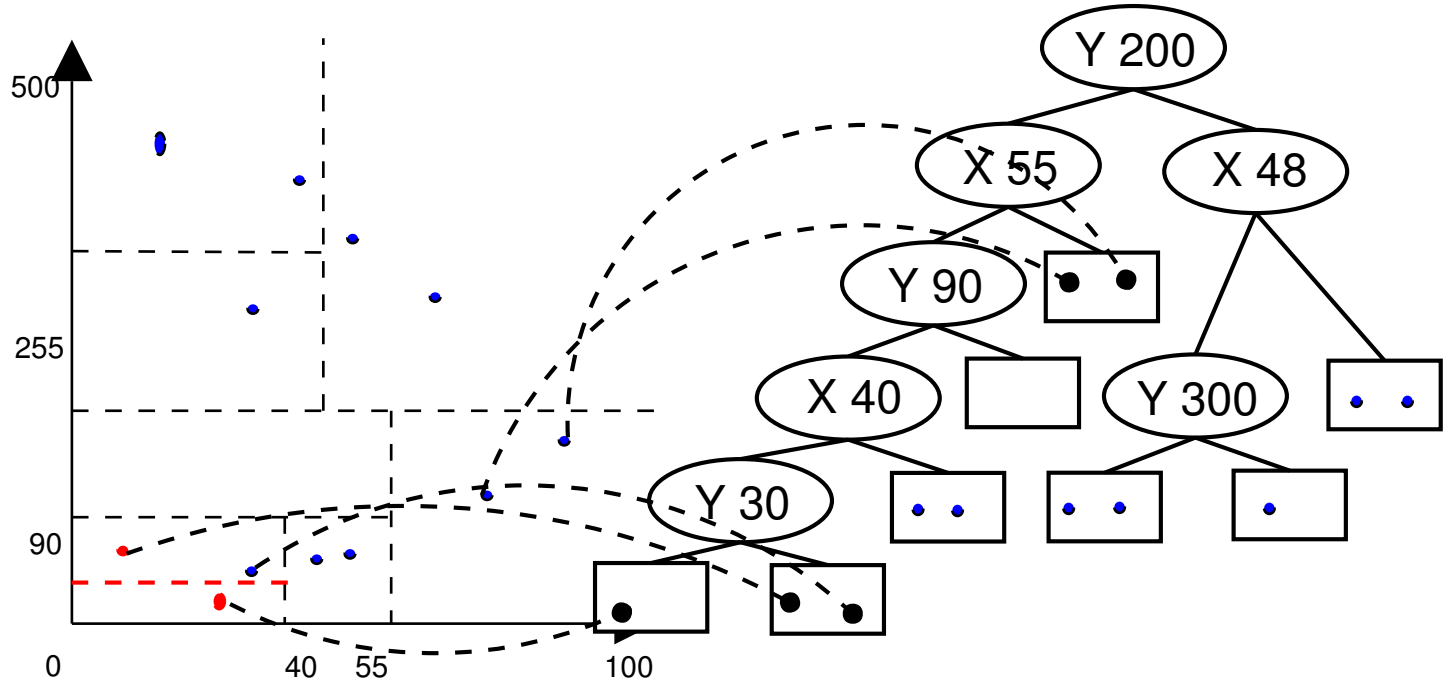
We can look at this as a tree as follows:



Multidimensional Indexes

kd-Trees

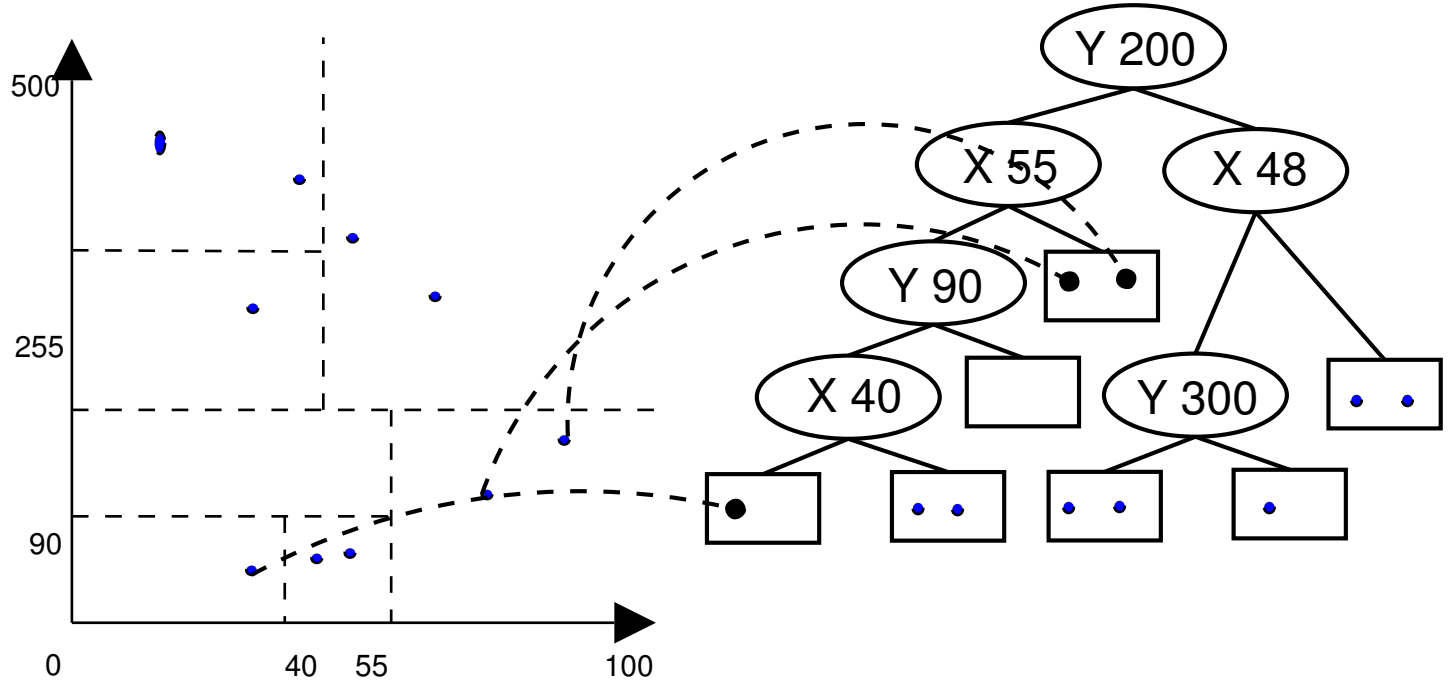
We continue splitting after new insertions:



Multidimensional Indexes

kd-Trees

- Good support for point queries
- Good support for partial match queries: e.g., $(y = 40)$
- Good support for range queries $(40 \leq x \leq 45 \wedge y < 80)$
- Reasonable support for nearest neighbour



Multidimensional Indexes

kd-Trees for secondary storage

- Generalization to n children for each interal node (cf. BTree).

  But it is difficult to keep this tree balanced since we cannot merge the children

- We limit ourselves to two children per node (as before), but store multiple nodes in a single block.

Physical Operators

Question

Could you please explain about TPMMS (Two-Phase, Multiway Merge-Sort) algorithm. Some examples might be helpful for me to understand that, because it might be confusing for some of us.

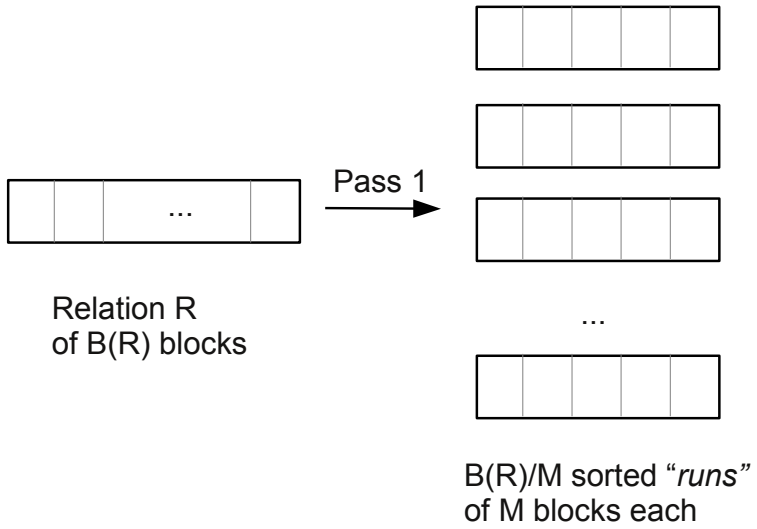
Answer

- (Re-explanation using following slides)
- Question: Why restrict to two-phase?
- Note: difference between project and description in book!
- Example: on blackboard for dataset containing $[1, 21]$, $M = 3$ (integers).

Physical Operators

Sort-based set union

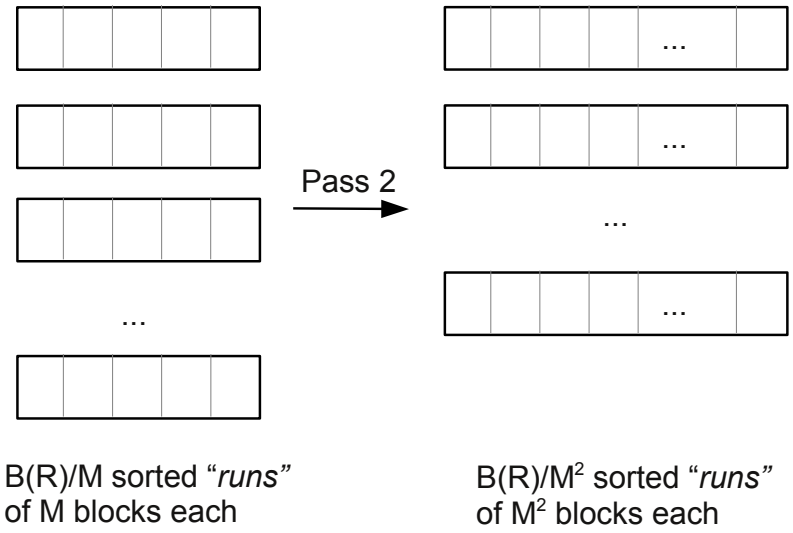
- Sorting can in principle be done by any suitable algorithm, but is usually done by Multiway Merge-Sort:
  - In the first pass we read M blocks at the same time from the input relation, sort these by means of a main-memory sorting algorithm, and write the sorted resulting sublist to disk. After the first pass we hence have $B(R)/M$ sorted sublists of M blocks each.



Physical Operators

Sort-based set union

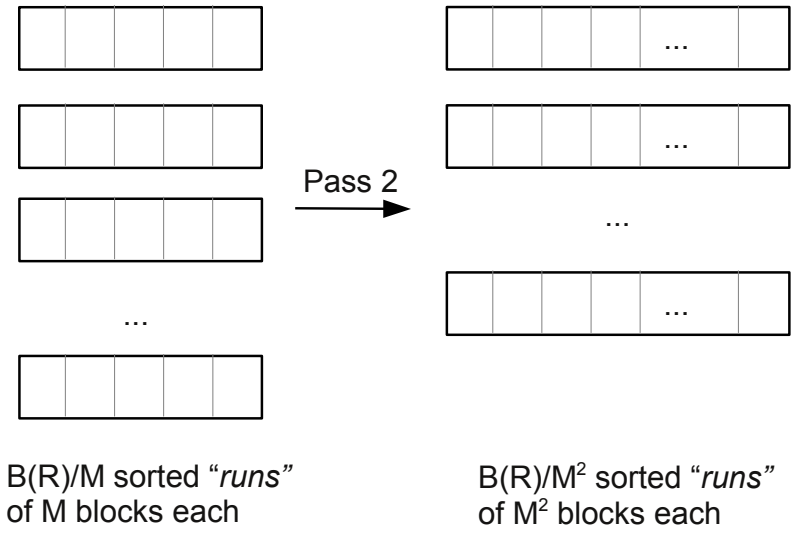
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - In the 2nd pass, we merge the first M sublists from the first pass into a single sublist of M^2 blocks. We do so by iterating synchronously over these M sublists, keeping 1 block of each list into memory during this iteration.



Physical Operators

Sort-based set union

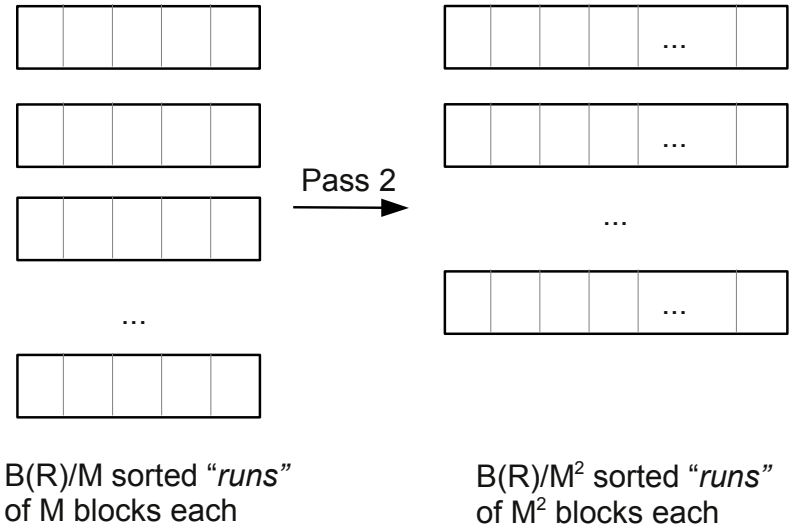
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - We then merge the next M sublists into a single sublist, and continue until we have treated each sublist resulting from the first pass.



Physical Operators

Sort-based set union

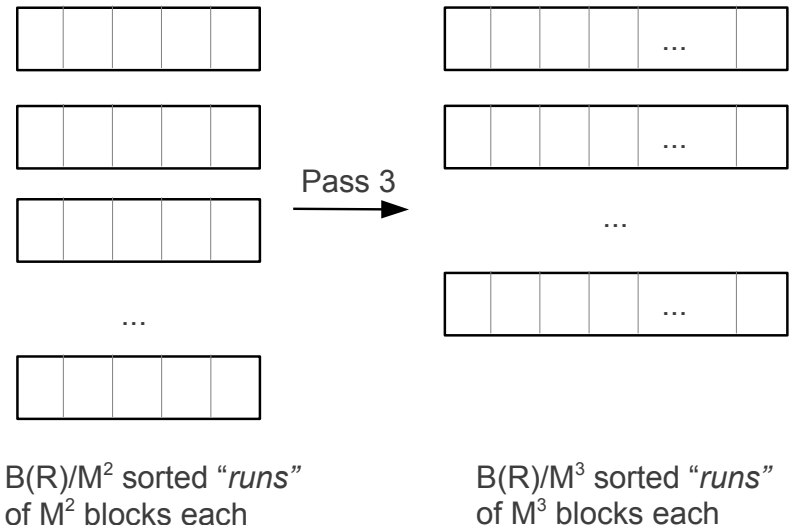
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - After the second pass we hence have $B(R)/M^2$ sorted sublists of M^2 blocks each.



Physical Operators

Sort-based set union

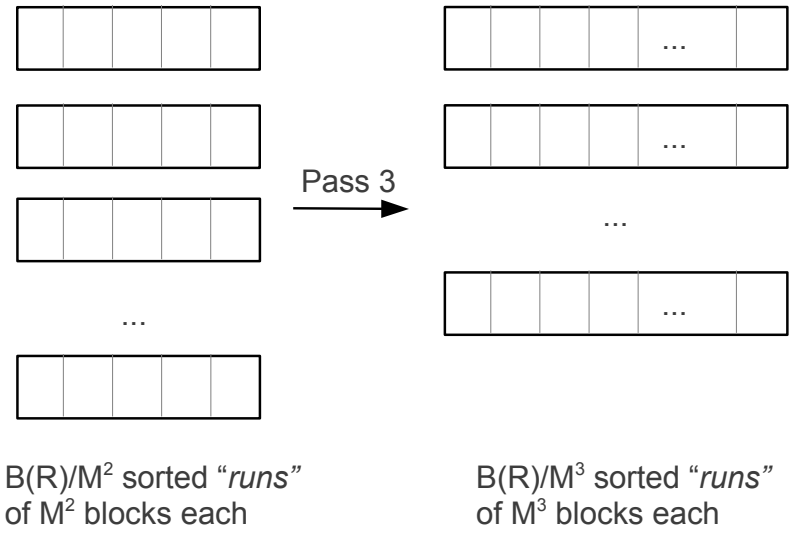
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - In the 3rd pass, we merge the first M sublists from the 2nd pass (each of M^2 blocks) into a single sublist of M^3 blocks. We do so by iterating synchronously over these M sublists, keeping 1 block of each list into memory during this iteration.



Physical Operators

Sort-based set union

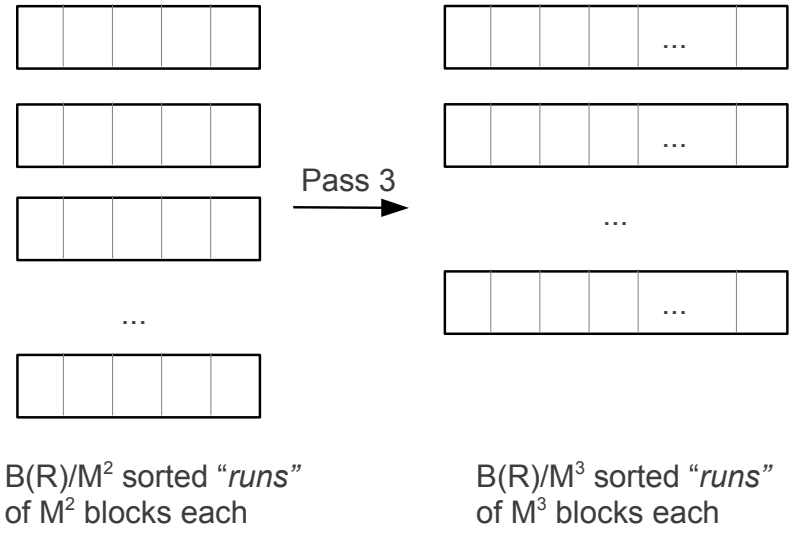
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - We then merge the next M sublists into a single sublist, and continue until we have treated each sublist resulting from the 2nd pass .



Physical Operators

Sort-based set union

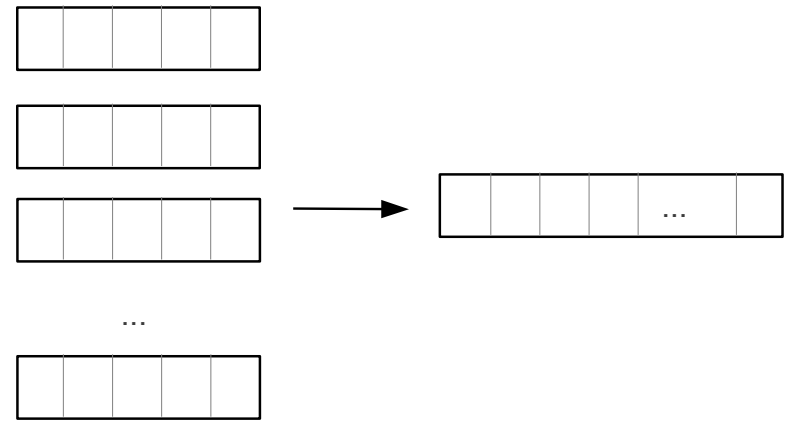
- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - After the 3rd pass we hence have $B(R)/M^3$ sorted sublists of M^3 blocks each.



Physical Operators

Sort-based set union

- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  - We keep doing new passes until we reach a single sorted list.



Physical Operators

Sort-based set union

- Sorting can in principle be done by suitable algorithm, but is usually done by Multiway Merge-Sort:
  1. In the first pass we read M blocks at the same time from the input relation, sort these by means of a main-memory sorting algorithm, and write the sorted resulting sublist to disk. After the first pass we hence have $B(R)/M$ sorted sublists of M blocks each.
  2. In the following passes we keep reading M blocks from these sublists and merge them into larger sorted sublists. (After the second pass we hence have $B(R)/M^2$ sorted sublists of M^2 blocks each, after the third pass $B(R)/M^3$ sorted sublists, ...)
  3. We repeat until we obtain a single sorted sublist.
- What is the complexity of this?
  1. In each pass we read and write the entire input relation exactly once.
  2. There are $\lceil \log_M B(R) \rceil$ passes
  3. The total cost is hence $2B(R) \lceil \log_M B(R) \rceil$ I/O operations.

Logging

Question

Could you give one more example for undo-redo log with checkpoints (like in ex. 17.4.5). It is still a little bit confusing.

Answer

Consider

$\langle \text{START } S\rangle\langle S, A, 60, 61\rangle\langle \text{COMMIT } S\rangle\langle \text{START } T\rangle\langle T, A, 10, 11\rangle$
$\langle \text{START } U\rangle\langle U, B, 20, 21\rangle\langle \text{CKPT START } (T, U)\rangle\langle T, C, 30, 31\rangle$
$\langle \text{START } V\rangle\langle \text{CKPT END}\rangle\langle U, D, 40, 41\rangle\langle V, F, 70, 71\rangle\langle \text{COMMIT } U\rangle\langle T, E, 50, 51\rangle$

A crash occurs right after $\langle T, E, 50, 51\rangle$ record. How should we recover from the crash?

Concurrency control

Question

So, the topic I feel a bit confused is the "Two Phase Locking",it would be great if you could take some examples to explain how the two phased scheduler is actually working.

Answer

A two-phase locked schedule is a schedule in which lock actions ($l_T(X)$) and unlock actions ($u_T(X)$) can also occur and:

1. before a transaction reads or writes to X, it must have the lock on X;
2. if a transaction T asks a lock on X which is held by another transaction, then the scheduler pauses T until the lock is released;
3. once a transaction does an unlock, it cannot do any more locks.

Concurrency control

Answer (cont)

- In practice transactions will only make read and write requests. They do not make lock and unlock requests. It is the task of the scheduler to add the latter to the schedule

  cf. Section 18.5 in book

- In particular, the scheduler (Part I) inserts the required lock requests.
- When a transaction commits or aborts, all of its locks are released by the scheduler (Part I)