MISTRAL

Processing Relational Queries Using a Multidimensional Access Method

Volker Markl
Rudolf Bayer

http://mistral.in.tum.de

FORWISS

(Bayerisches Forschungszentrum für Wissensbasierte Systeme)

Staff Members

MISTRAL Project Management

Prof. Rudolf Bayer, Ph.D. (FORWISS Knowledge Bases Group Head)

Dr. Volker Markl (MISTRAL Project Leader, Deputy Research Group Head)

MISTRAL Research Assistants

Dipl. Inform. Robert Fenk

Dipl. Inform. Roland Pieringer

Frank Ramsak, M.Sc.

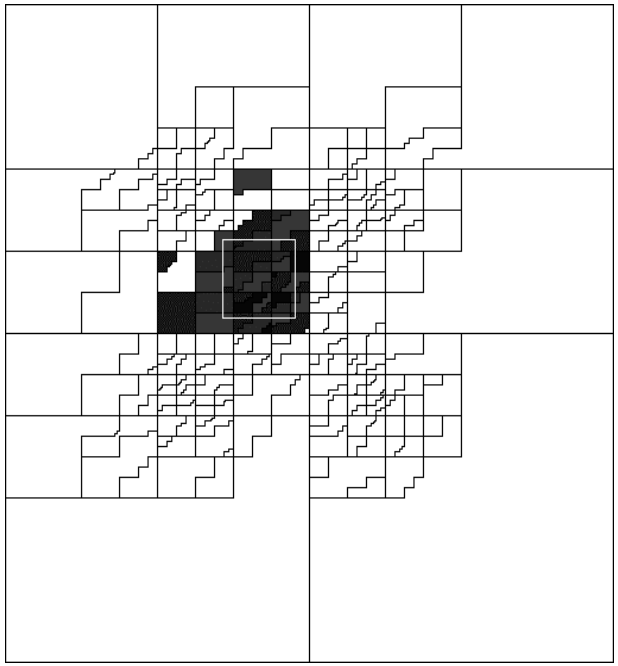
Dipl. Inform. Martin Zirkel

MISTRAL Master Students and Interns

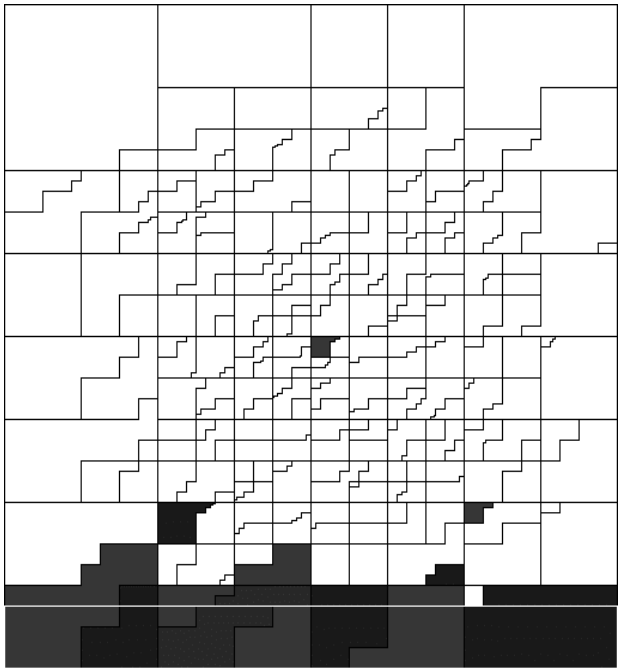
Ralf Acker, Bulent Altan, Sonja Antunes, Michael Bauer, Sascha Catelin, Naoufel Boulila, Nils Frielinghaus, Sebastian Hick, Stefan Krause, Jörg Lanzinger, Christian Leiter, Yiwen Lue, Stephan Merkel, Nasim Nadjafi, Oliver Nickel, Daniel Ovadya, Markus Pfadenauer, Timka Piric, Sabine Rauschendorfer, Antonius Salim, Maximilian Schramm, Michael Streichsbier, Anton Tichatschek

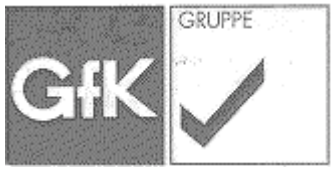
http://mistral.in.tum.de

Range Queries



Tetris Algorithm



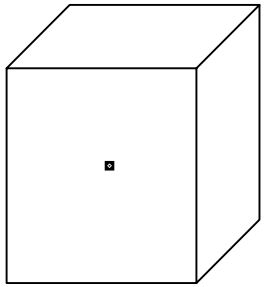


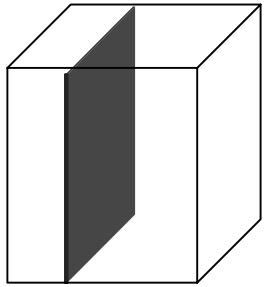
Overview

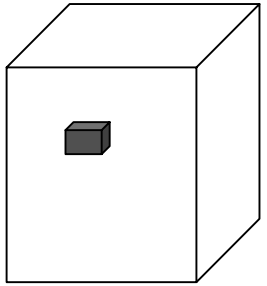
1. Concept of the UB-Tree: Z-Regions
2. Insertion
3. Range Query Algorithm
4. Tetris Algorithm
5. Kernel Integration
6. Performance Overview

Relations and MD Space

- Decision Support Relation (similar to TPC-D)
  - Fact(customer, product, time, Sales)
  - → defines a three dimensional cube
- Point Query
  - All sales for one customer for one specific product on a certain day
- Partial Match Query
  - All sales for product X
- Range Query
  - All sales for year 1999 for a specific product group for a specific customer group





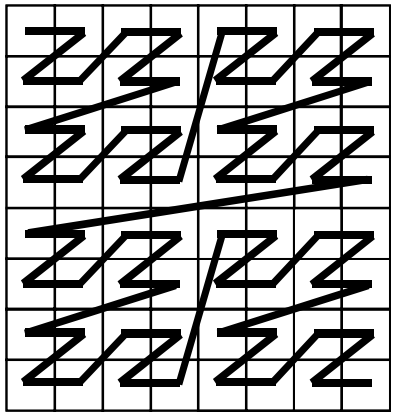


Design Goals

- clustering tuples on disk pages while preserving spatial proximity
- efficient incremental organization
- logarithmic worst-case guarantees for insertion, deletion and point queries
- efficient handling of range queries
- good average memory utilization

Z-Ordering

$$Z(x) = \sum_{i=0}^{s-1} \sum_{j=1}^{d} x_{j,i} \cdot 2^{i \cdot d + j - 1}$$



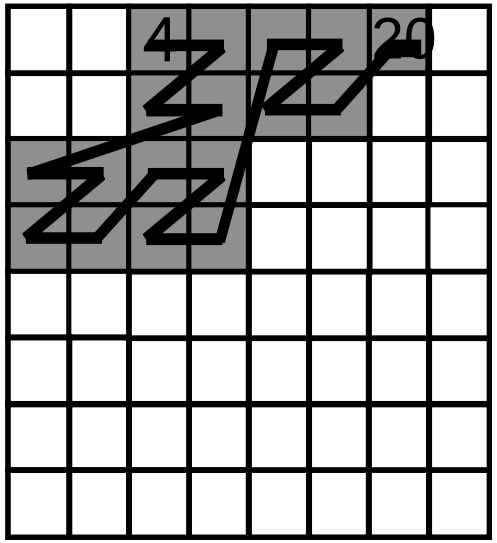
(a)

	0	1	2	3	4	5	6	7
0	0	1	4	5	16	17	20	21
1	2	3	6	7	18	19	22	23
2	8	9	12	13	24	25	28	29
3	10	11	14	15	26	27	30	31
4	32	33	36	37	48	49	52	53
5	34	35	38	39	50	51	54	55
6	40	41	44	45	56	57	60	61
7	42	43	46	47	58	59	62	63

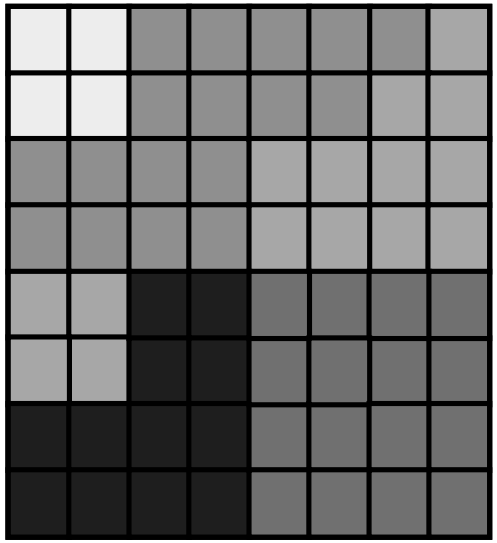
(b)

Z-regions/UB-Trees

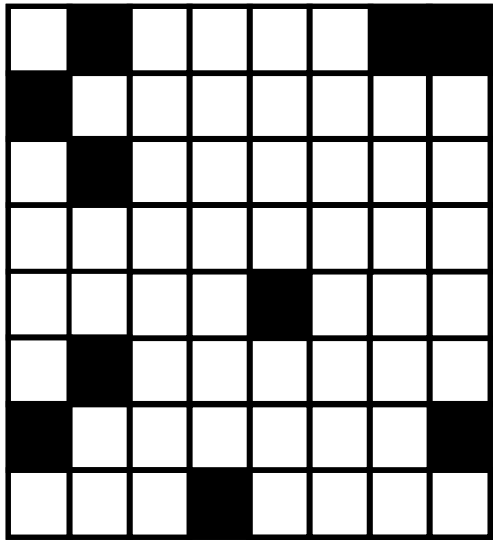
A *Z-region* [α : β] is the space covered by an interval on the Z-curve and is defined by two Z-addresses α and β.



Z-region [4 : 20]

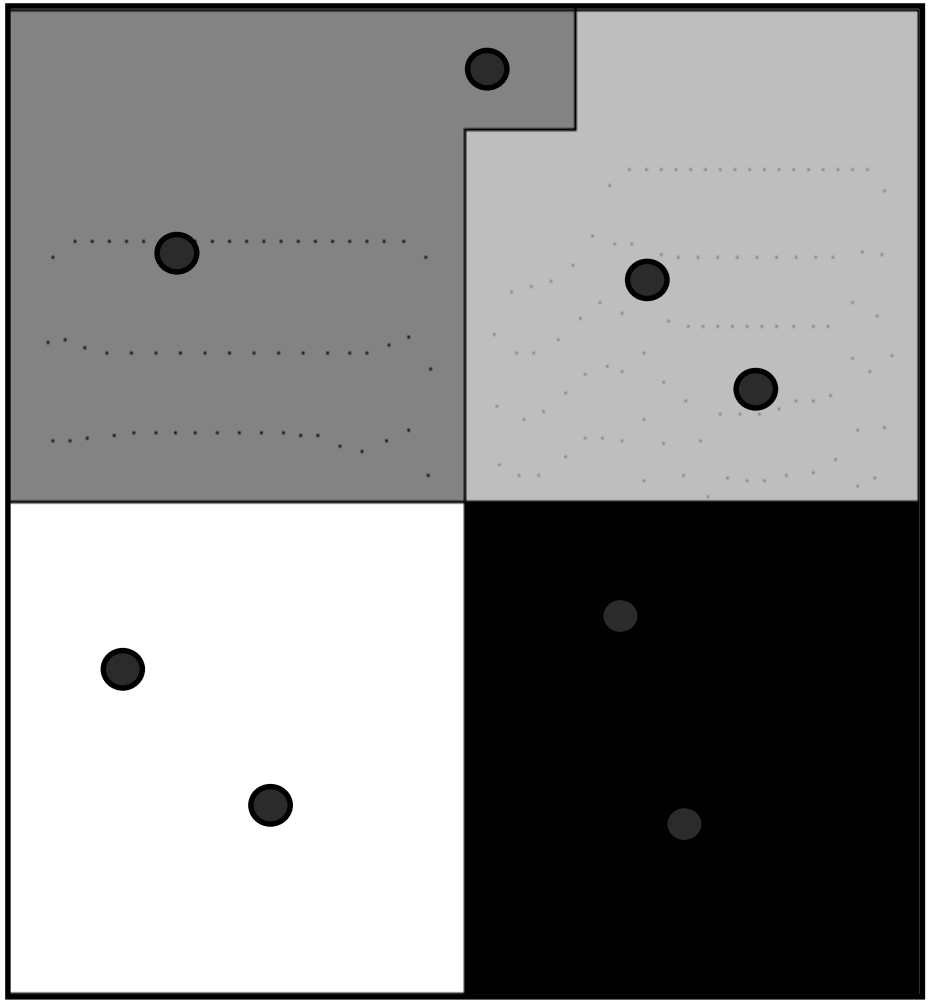


UB-Tree partitioning: [0 : 3],[4 : 20], [21 : 35], [36 : 47], [48 : 63]

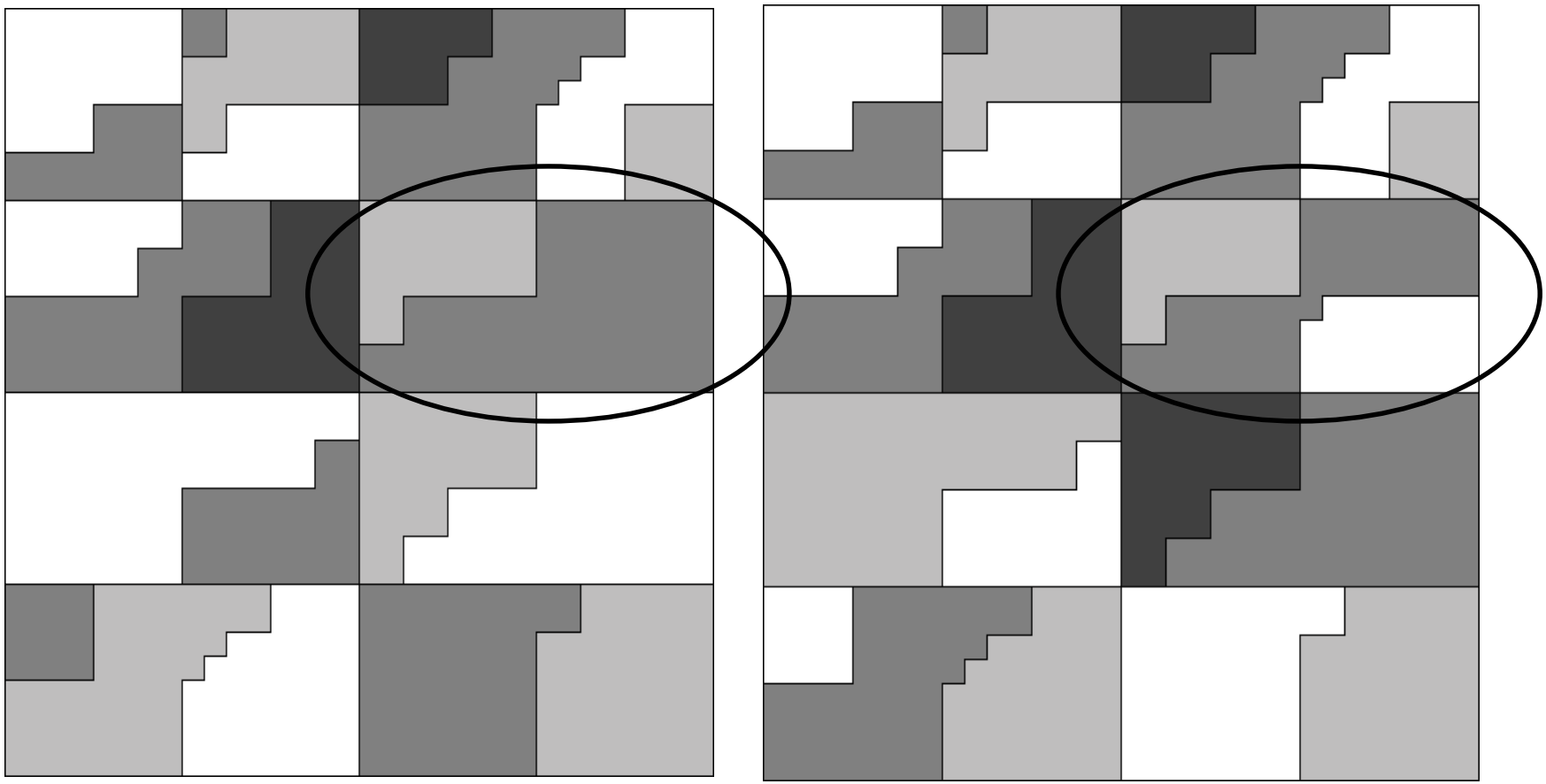


point data creating the UB-Tree on the left for a page capacity of 2 points

UB-Tree Insertion 1/2/3/4



UB-Tree Insertion 18/19

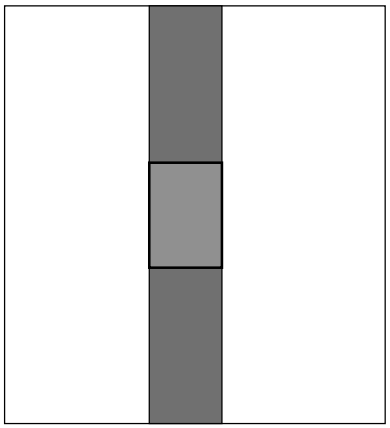


Multidimensional Range Query

SELECT * FROM table

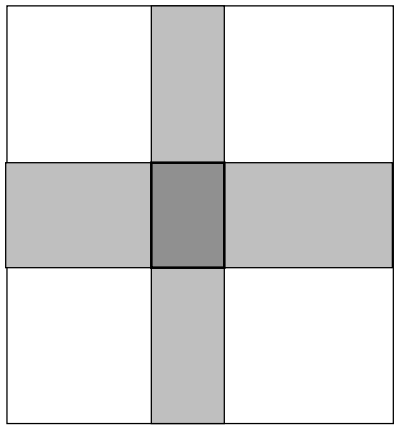
WHERE (A_1 BETWEEN a_1 AND b_1) AND
(A_2 BETWEEN a_2 AND b_2) AND
.....
(A_n BETWEEN a_n AND b_n)

Theoretical Comparison of the Rangequery Performance



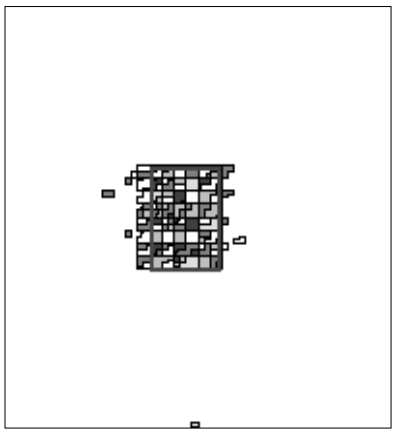
composite key clustering B-Tree

$s_1 * P$



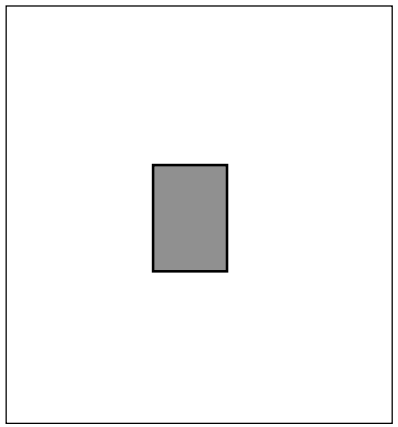
multiple B-Trees, bitmap indexes

$s_1 * l_1 + s_2 * l_2 + s_1 * s_2 * T$



multidimensional index

$s_1^{\uparrow} * s_2^{\uparrow} * P$



ideal case

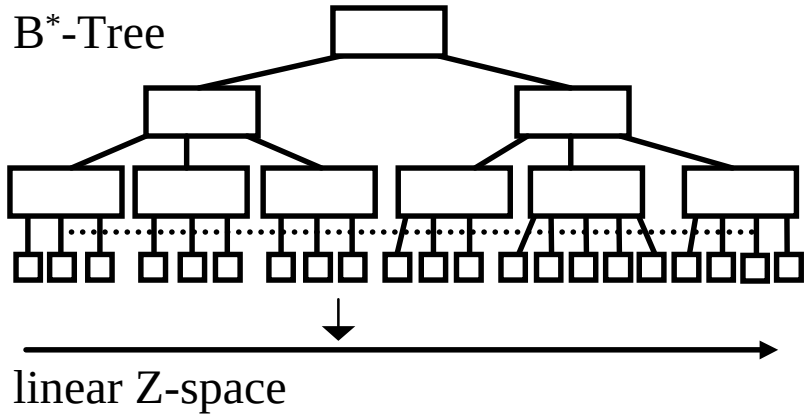
$s_1 * s_2 * P$

```
rangeQuery(Tuple ql, Tuple qh)
{
  Zaddress start  = Z(ql);
  Zaddress cur    = start;
  Zaddress end    = Z(qh);
  Page page = {};

  while (1)
  {
    cur = getRegionSeparator(cur);
    page = getPage(cur);
    outputMatchingTuples(page, ql, qh);
    if ( cur >= end ) break;
      cur = getNextZAddress(cur, start, end);
  }
}
```

UB-Tree

B*-Tree

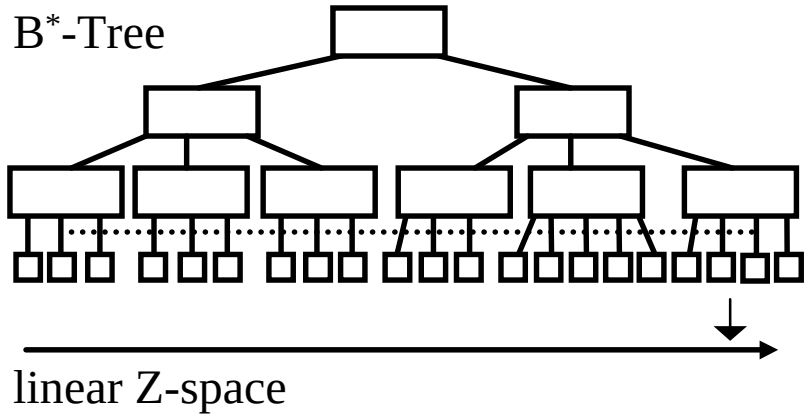


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

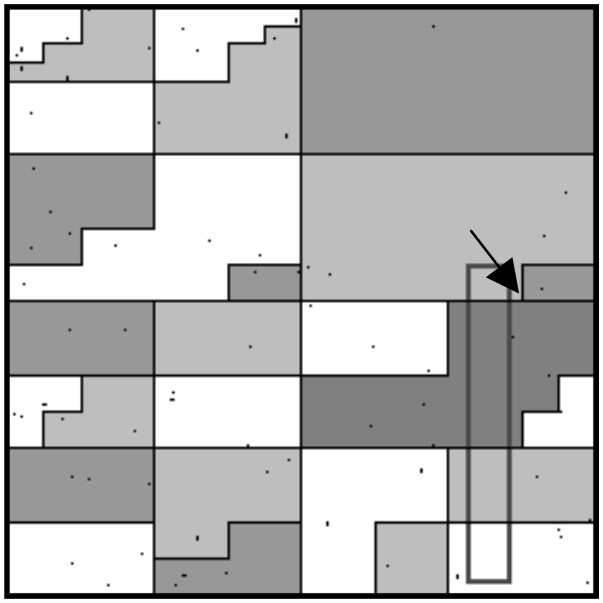
B*-Tree



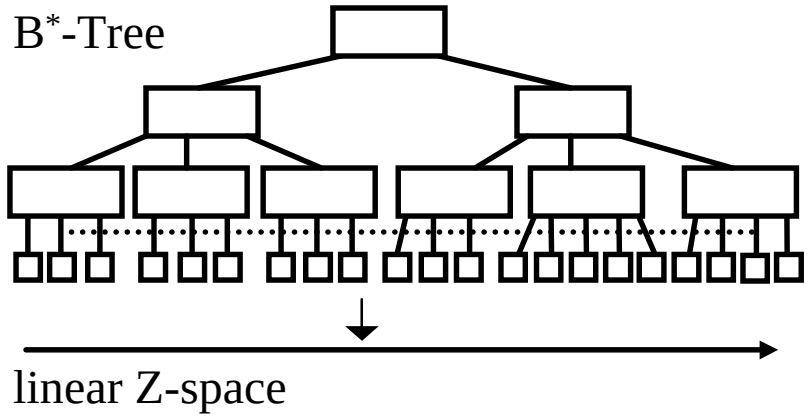
```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
    cur = getRegionSeparator(cur);
    page = getPage(cur);
    outputMatchingTuples(page, ql, qh);
    if ( cur >= end ) break;
      cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree



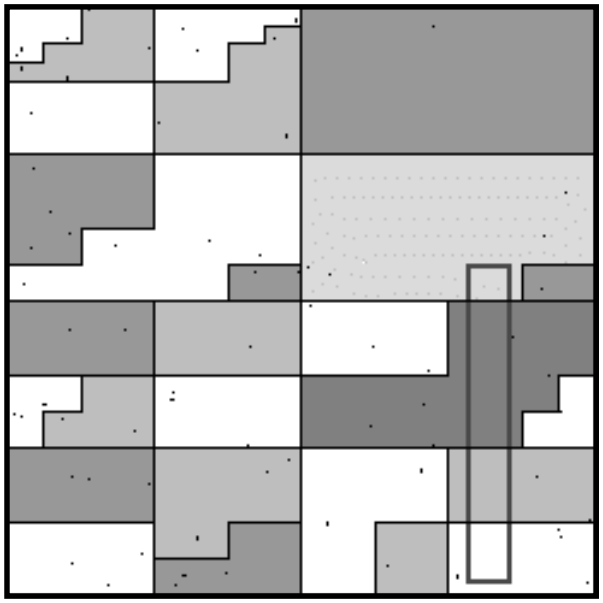
B*-Tree



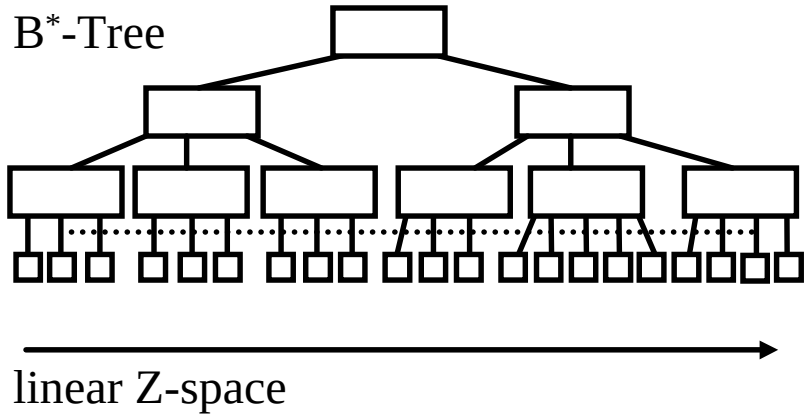
```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree



B*-Tree

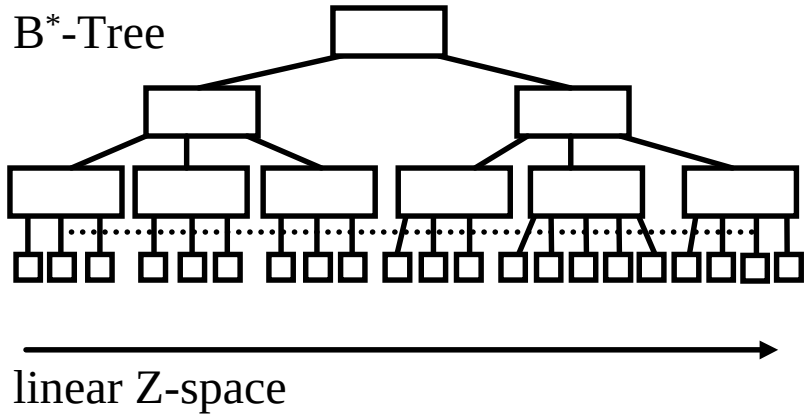


```
rangeQuery(Tuple ql, Tuple qh)
{
    Zaddress start  = Z(ql);
    Zaddress cur    = start;
    Zaddress end    = Z(qh);
    Page page = {};

    while (1)
    {
      cur = getRegionSeparator(cur);
      page = getPage(cur);
      outputMatchingTuples(page, ql, qh);
      if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
    }
}
```

UB-Tree

B*-Tree

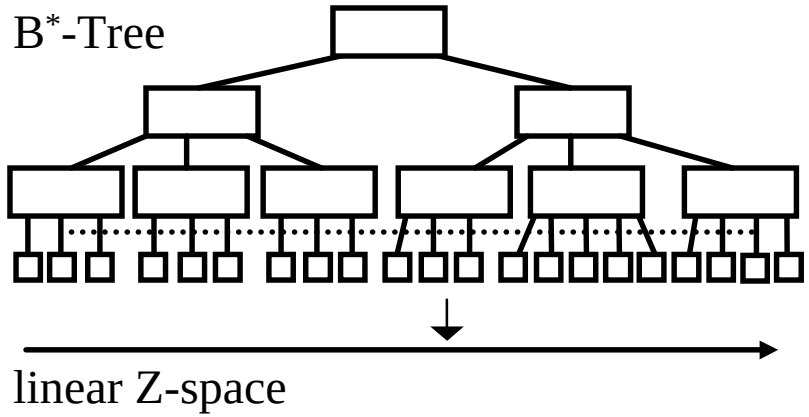


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
     cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

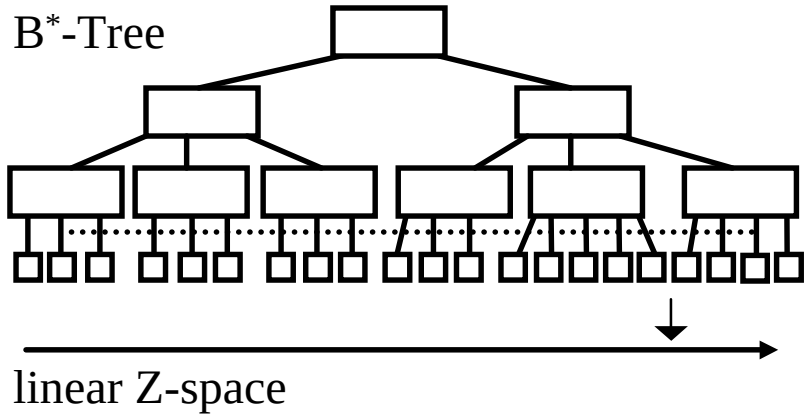


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
   cur = getRegionSeparator(cur);
    page = getPage(cur);
    outputMatchingTuples(page, ql, qh);
    if ( cur >= end ) break;
      cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

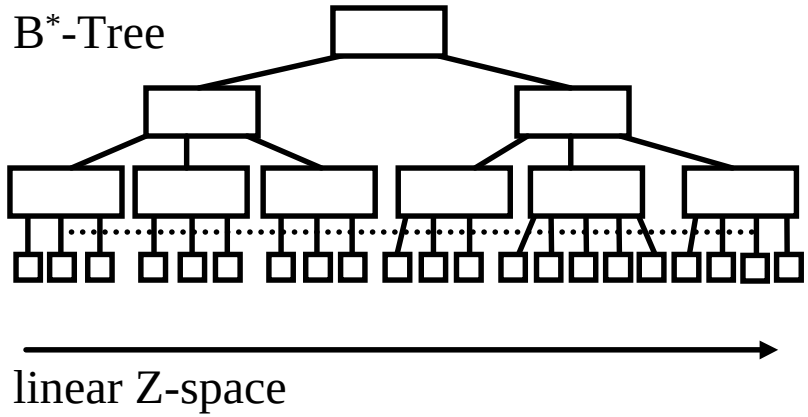


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

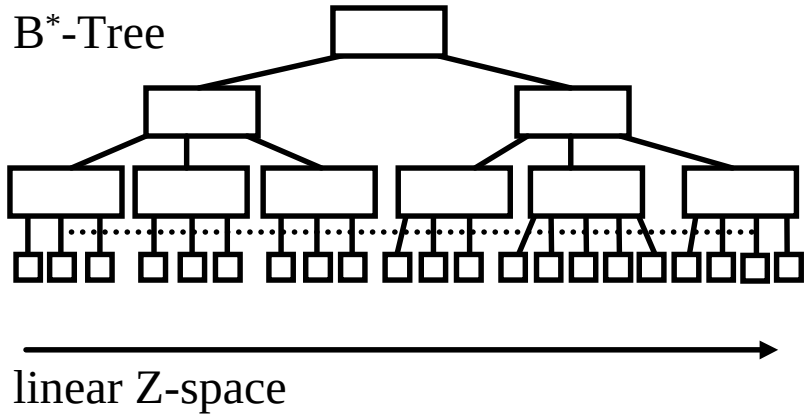


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

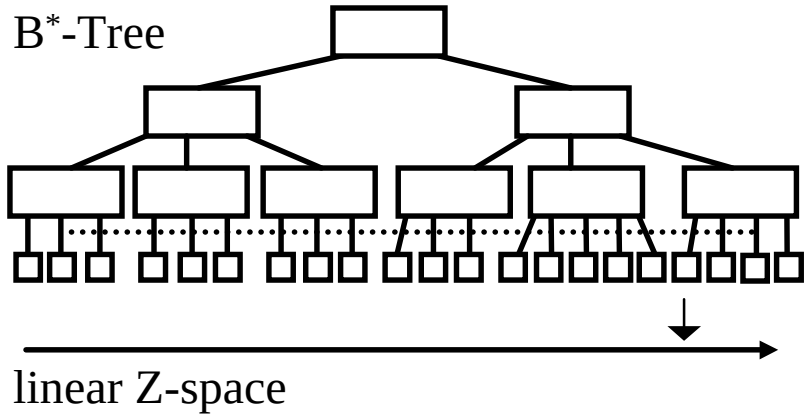


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
     cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

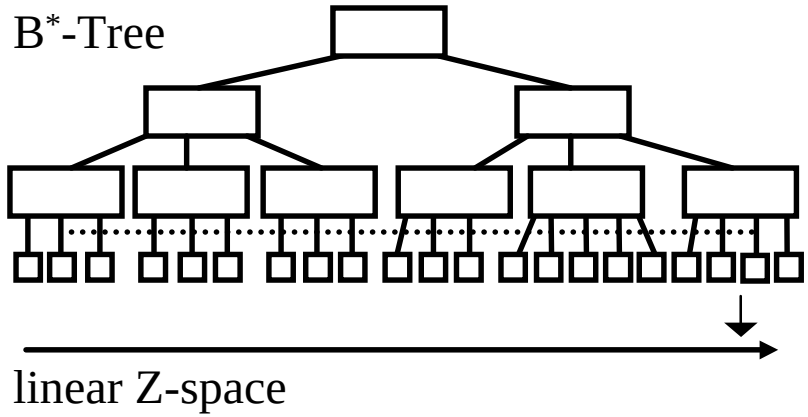


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
    cur = getRegionSeparator(cur);
    page = getPage(cur);
    outputMatchingTuples(page, ql, qh);
    if ( cur >= end ) break;
      cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

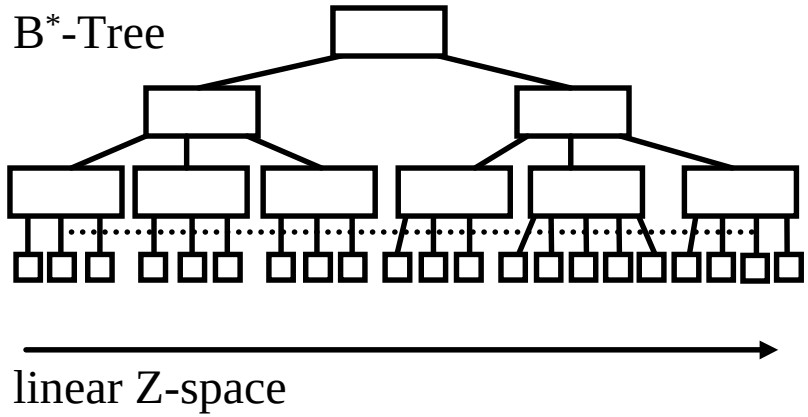


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

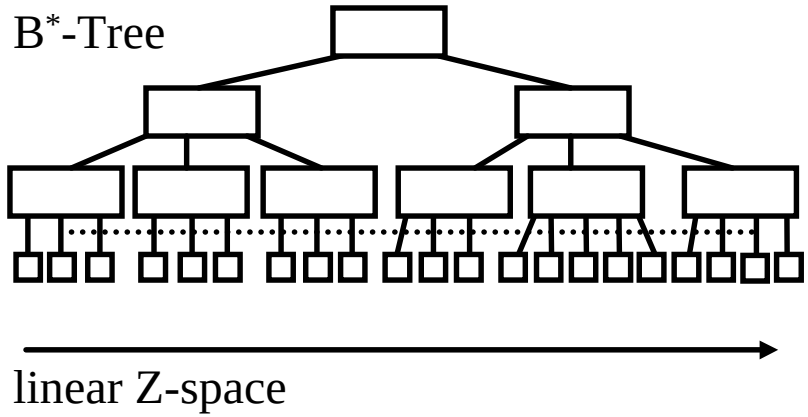


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree

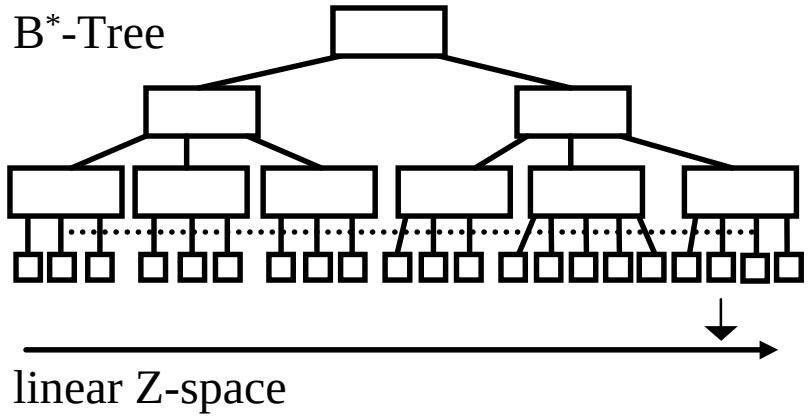


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
     cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

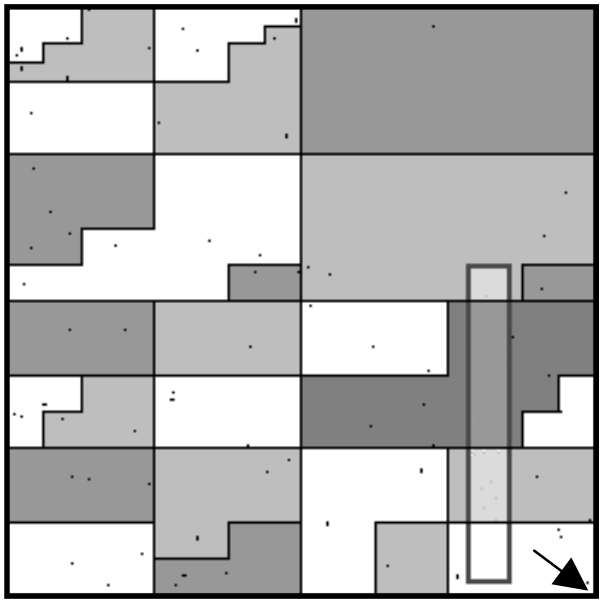
B*-Tree



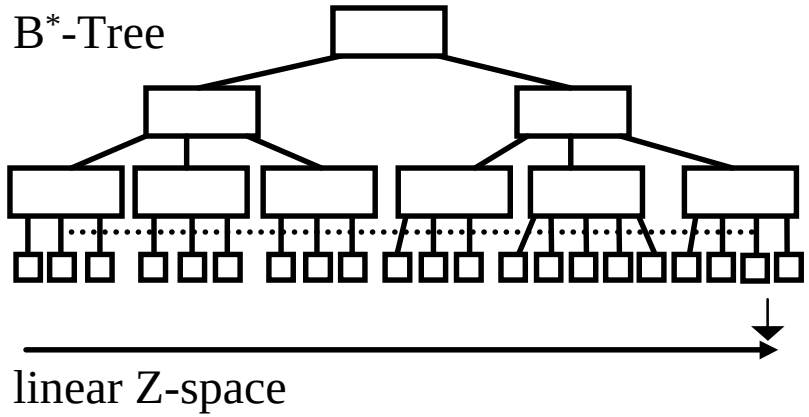
```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
    cur = getRegionSeparator(cur);
    page = getPage(cur);
    outputMatchingTuples(page, ql, qh);
    if ( cur >= end ) break;
      cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree



B*-Tree

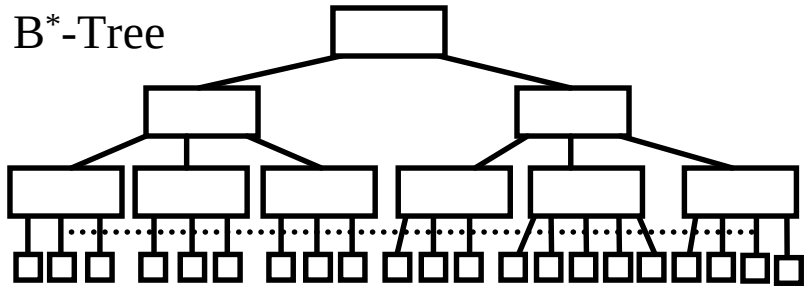


```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

B*-Tree



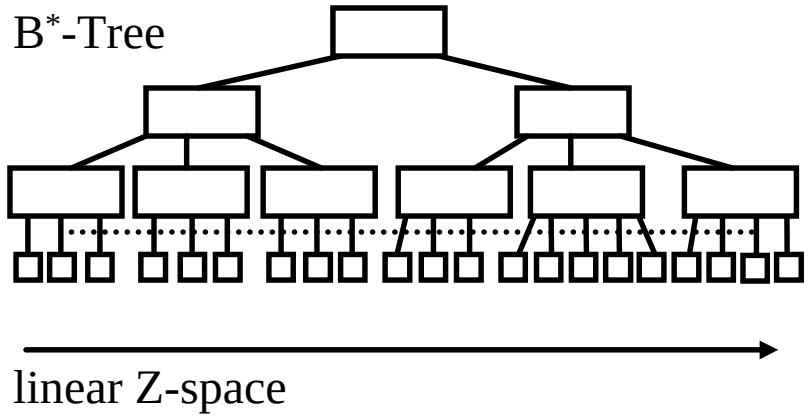
linear Z-space

```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
        cur = getNextZAddress(cur, start, end);
   }
}
```

UB-Tree

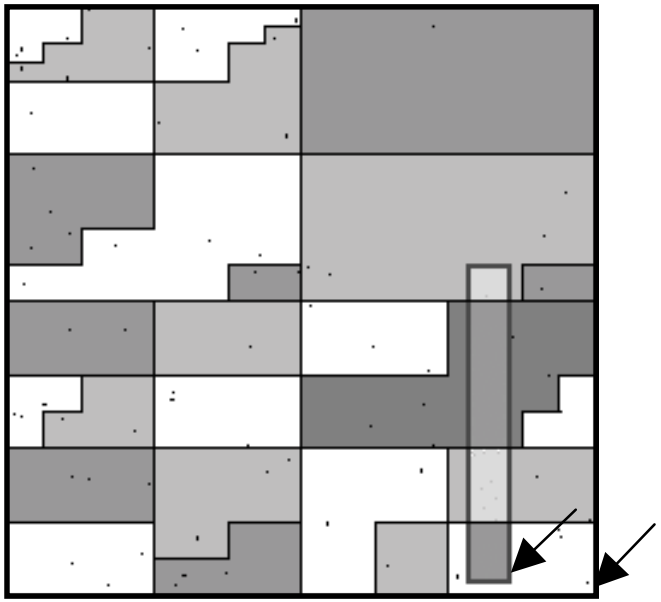
B*-Tree



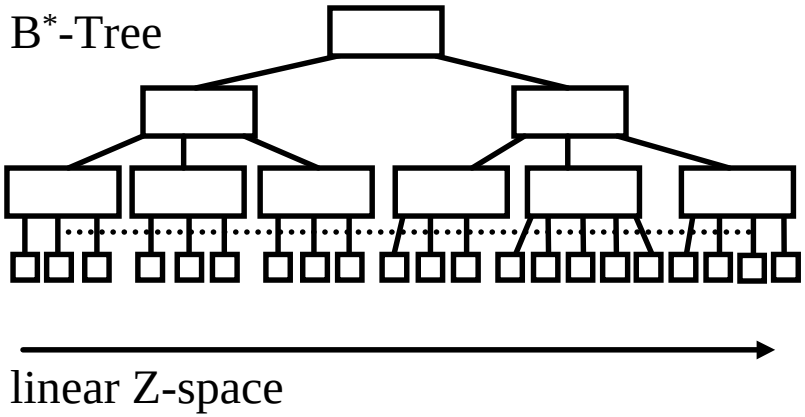
```
rangeQuery(Tuple ql, Tuple qh)
{
   Zaddress start  = Z(ql);
   Zaddress cur    = start;
   Zaddress end    = Z(qh);
   Page page = {};

   while (1)
   {
     cur = getRegionSeparator(cur);
     page = getPage(cur);
     outputMatchingTuples(page, ql, qh);
     if ( cur >= end ) break;
       cur = getNextZAddress(cur, start, end);
   }
}
```

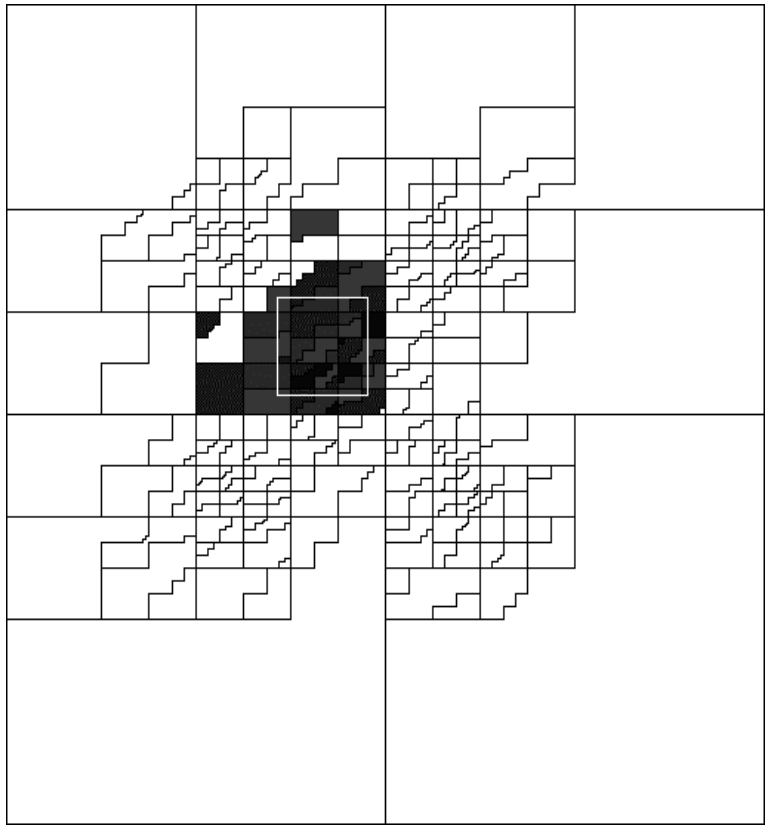
UB-Tree

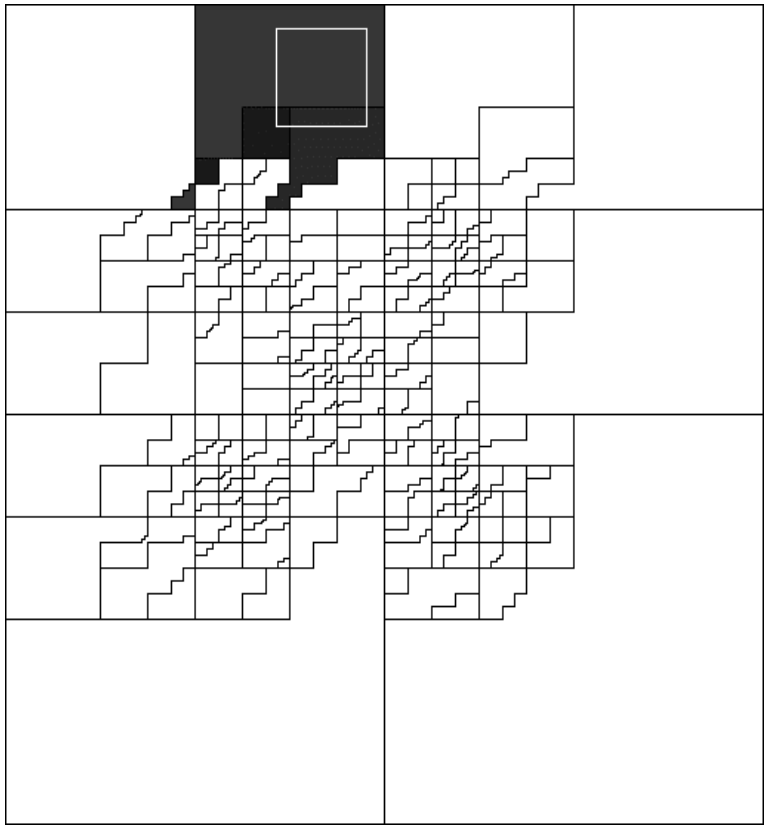


B*-Tree

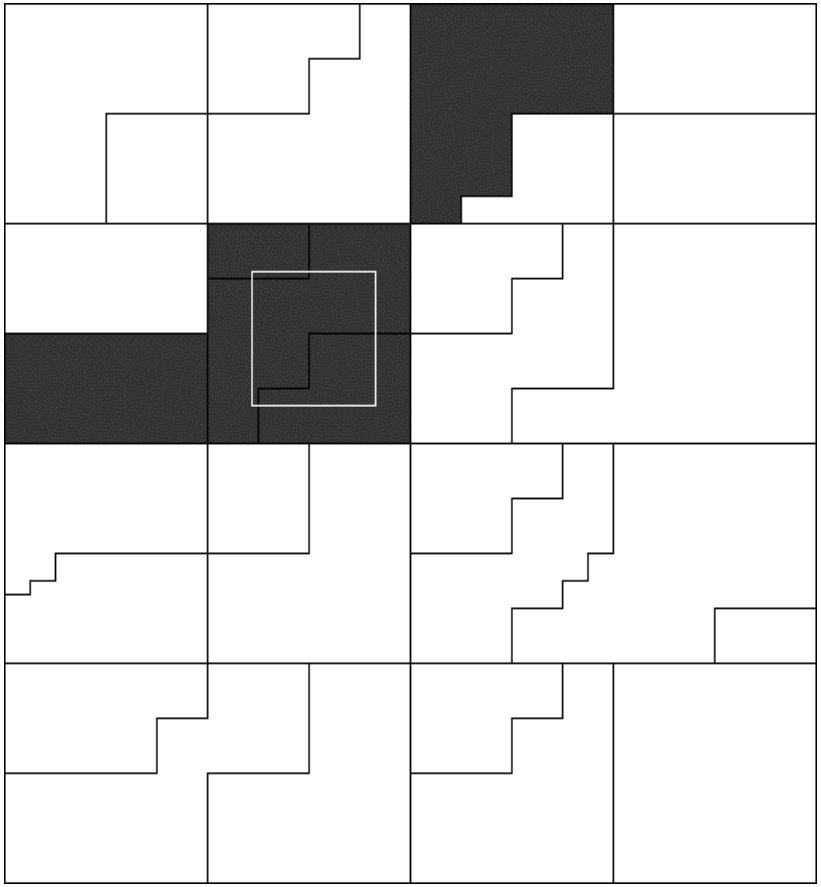


Range Queries and Data Distributions

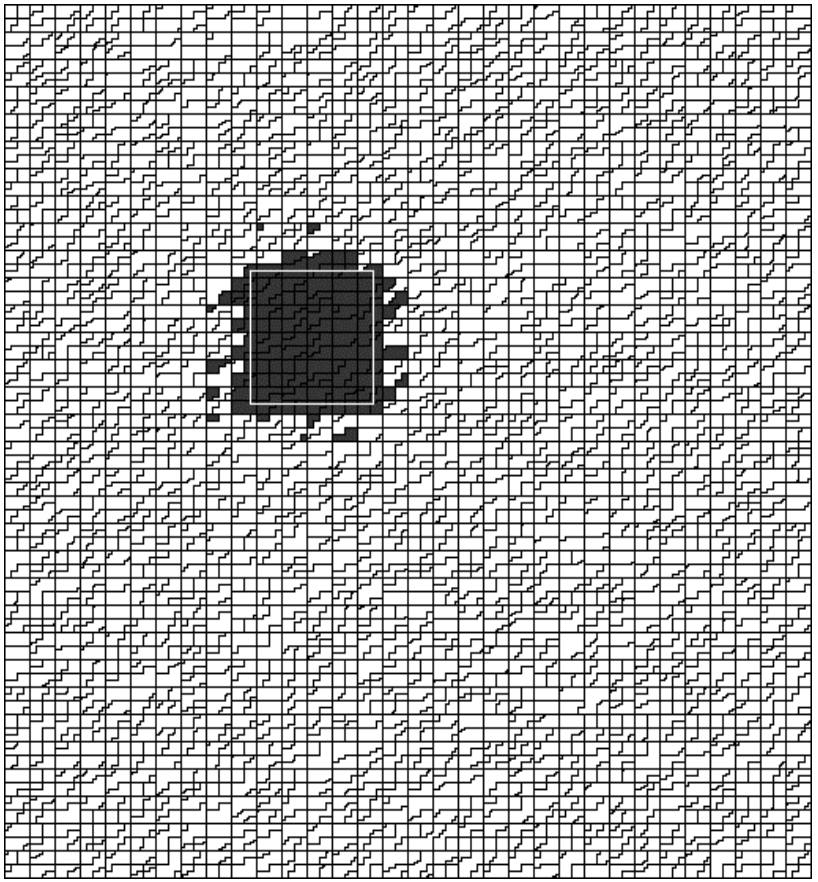




Growing Databases



1000 tuples



50 000 tuples

Summary UB-Trees

- 50% storage utilization, dynamic updates
- Efficient Z-address calculation (bit-interleaving)
- Logarithmic performance for the basic operations
- Efficient range query algorithm (bit-operations)
- Prototype UB/API above RDBMS (Oracle 8, Informix, DB2 UDB, TransBase, MS SQL 7.0) using ESQL/C
- **Patent application**

Standard Query Pattern

SELECT * FROM table

WHERE (A_1 BETWEEN a_1 AND b_1) AND

(A_2 BETWEEN a_2 AND b_2) AND

.....

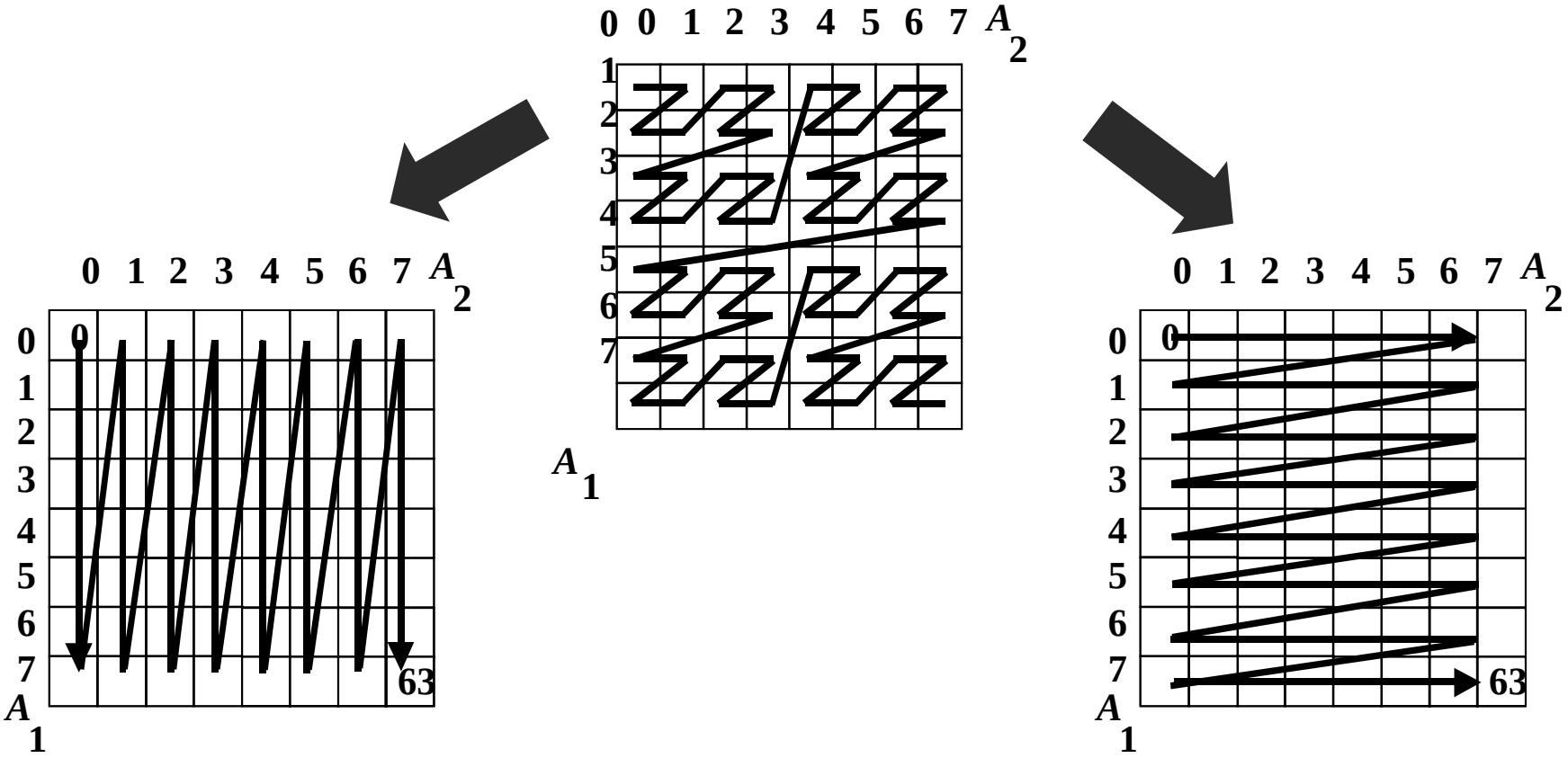
(A_n BETWEEN a_n AND b_n)

ORDER BY A_i, A_j, A_k, ...

(GROUP BY A_i, A_j, A_k, ...)

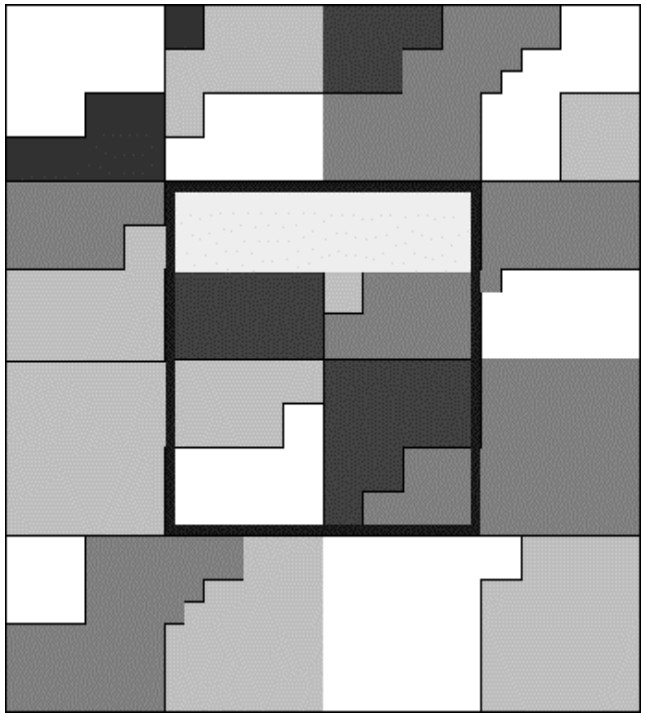
Z-Order/Tetris Order

$$T_j(x) = x_j \circ Z(x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_d)$$



The Tetris Algorithm

sort direction



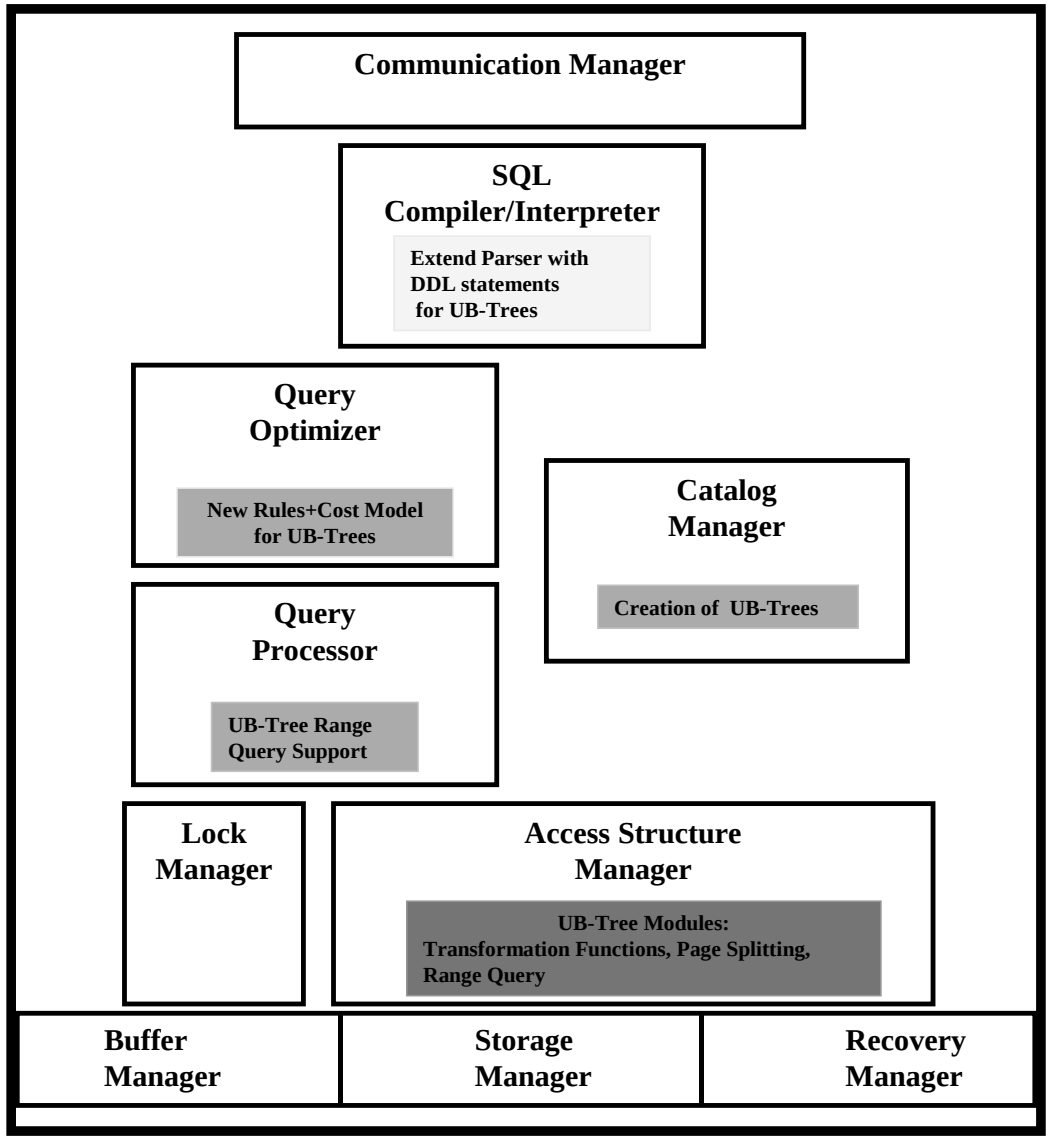
Summary Tetris

- Combines sort process and evaluation of multi-attribute restrictions in one processing step
- I/O-time linear w.r. to result set size
- temporary storage sublinear w.r. to result set size
- Sorting no longer a “blocking operation”

✍ **Patent application**

Integration Issues

- Starting point with TransBase:
  - clustering B*-Tree
  - appropriate data type for Z-values: variable bit strings
- Modifications to B*-Tree in TransBase:
  - support for computed keys:
    - » Z-values are only stored in the index, not together with the tuples
    - » tuples are stored in Z-order
  - generalization of splitting algorithm:
    - » computed page separators for improved space partitioning



- Minor extensions:
- Major extensions:
- New modules:
- **NO changes for:**
  - DML
  - Multi-user support, i.e., locking, logging facilities
    → handled by underlying B*-Tree

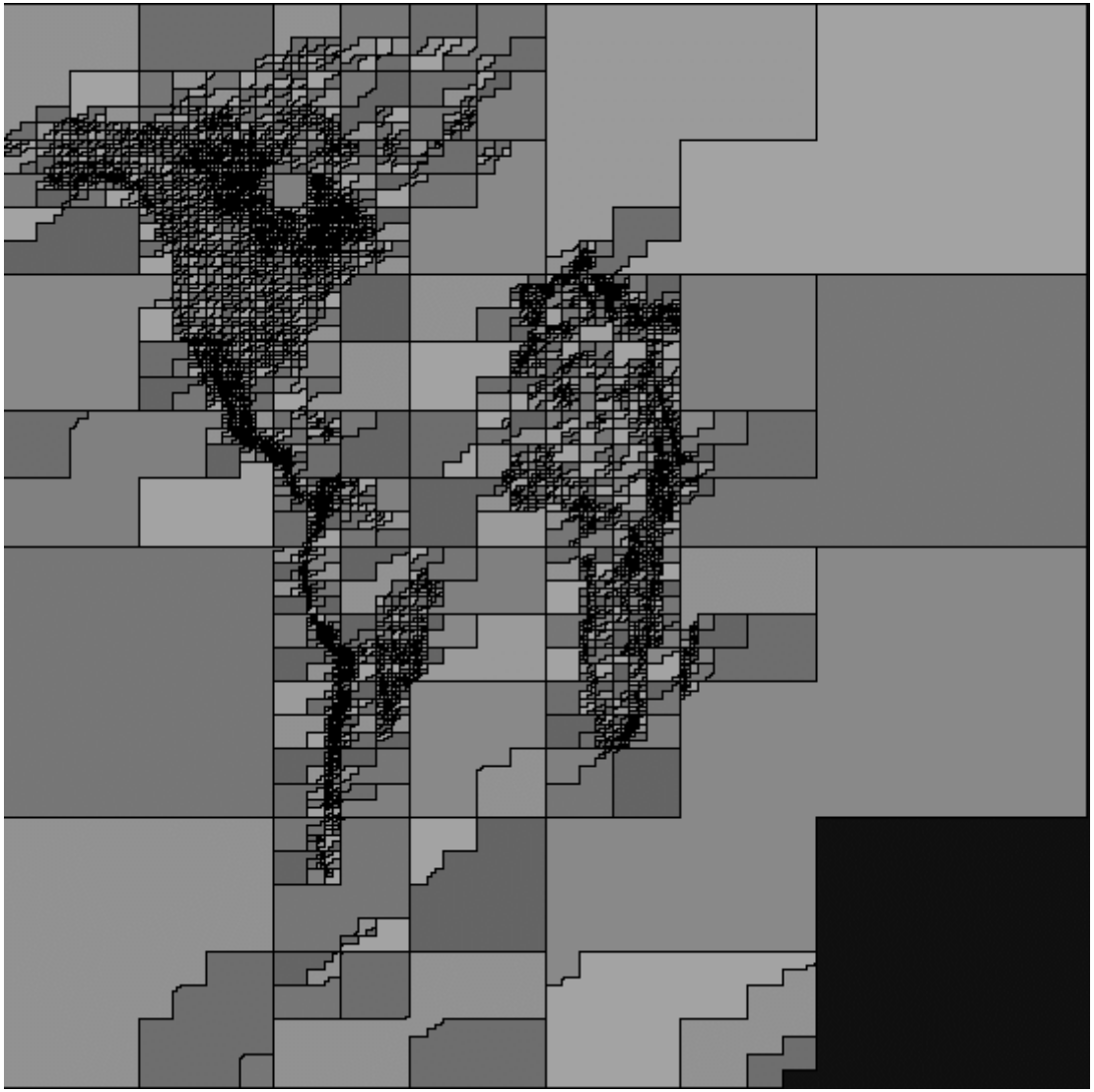
Summary Integration

- Integration of the UB-Tree has been achieved within one year
- TransBase HyperCube is shipping since Systems 1999 and was awarded the 2001 IT-Prize by EUROCASE and the European Commission
- UB-Trees speed up relational DBMS for multidimensional applications like Geo-DB and data warehouse up to two orders of magnitude
- Speedup is even more dramatic for CD-ROM databases (archives)

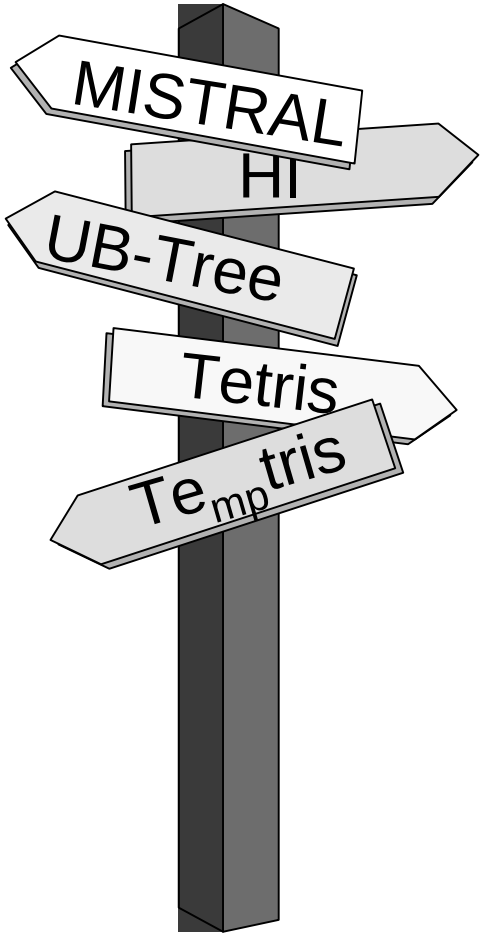
Application Fields of the UB-Tree

- Data Warehouses
  - Measurements with SAP BW Data
    - » UB-Tree/API for Oracle
    - » UB-Tree ***on top of*** Oracle outperforms conventional B-Tree and Bitmap indexes ***in*** Oracle!
  - Measurements with the GfK Data Warehouse
    - » UB-Tree in TransBase HyperCube
    - » significant performance increases (Factor of 10)
- Geographic Databases
- „Multidimensional Problems“
  - Archiving Systems, Lifecycle-Management, Data Mining, OLAP, OLTP, etc.

The UB-Tree



Further Information



http://mistral.in.tum.de

mistral@in.tum.de