

**B+-trees...**

... over composite keys

**Point Quad Trees**

Point (Equality) Search

Inserting Data

Region Queries

**k-d Trees**

Balanced Construction

**K-D-B-Trees**

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

**R-Trees**

Searching and Inserting

Splitting R-Tree Nodes

**UB-Trees**

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

**Spaces with High  
Dimensionality**

**Wrap-Up**

# Chapter 5

## Multi-Dimensional Indexing

Indexing Points and Regions in  $k$  Dimensions

*Architecture and Implementation of Database Systems*

Winter 2010/11

Torsten Grust  
Wilhelm-Schickard-Institut für Informatik  
Universität Tübingen



# More Dimensions...

## One SQL query, many range predicates

```
1 SELECT *
2 FROM CUSTOMERS
3 WHERE ZIPCODE BETWEEN 8880 AND 8999
4 AND REVENUE BETWEEN 3500 AND 6000
```

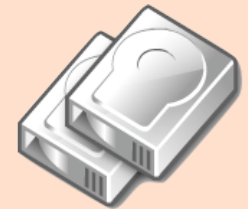
This query involves a **range predicate** in **two** dimensions.

Typical use cases for **multi-dimensional data** include

- **geographical data** (GIS: Geographical Information Systems),
- **multimedia retrieval** (e.g., image or video search),
- **OLAP** (Online Analytical Processing),
- queries against **encoded data** (e.g., XML)

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

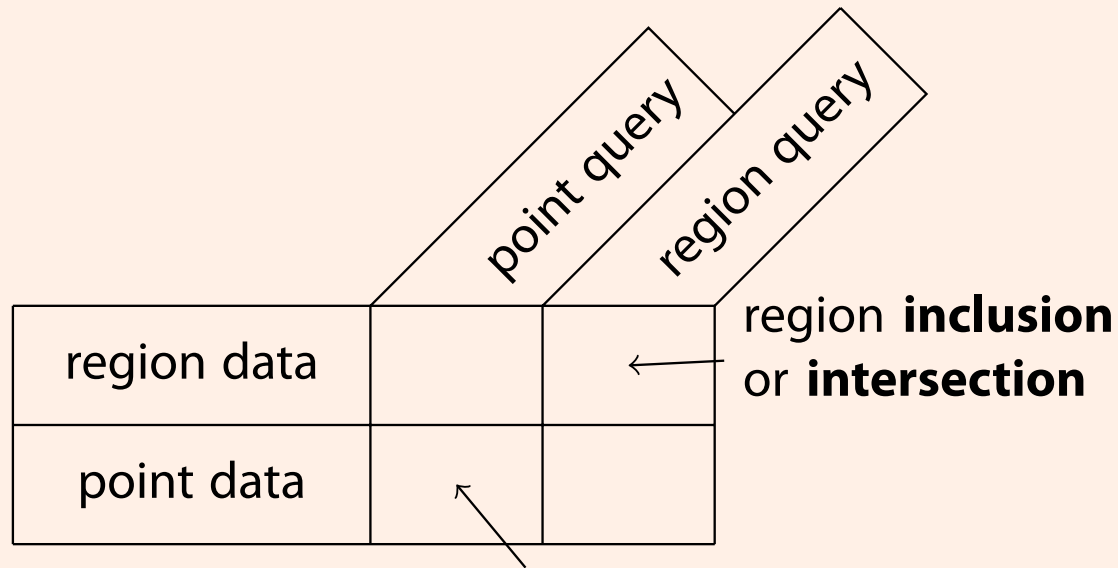
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

## ... More Challenges...

### Queries and data can be points or regions



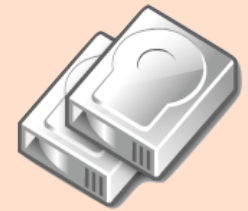
most interesting: ***k*-nearest-neighbor search (*k*-NN)**

... and you can come up with many more meaningful types of queries over multi-dimensional data.

Note: All equality searches can be reduced to one-dimensional queries.

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

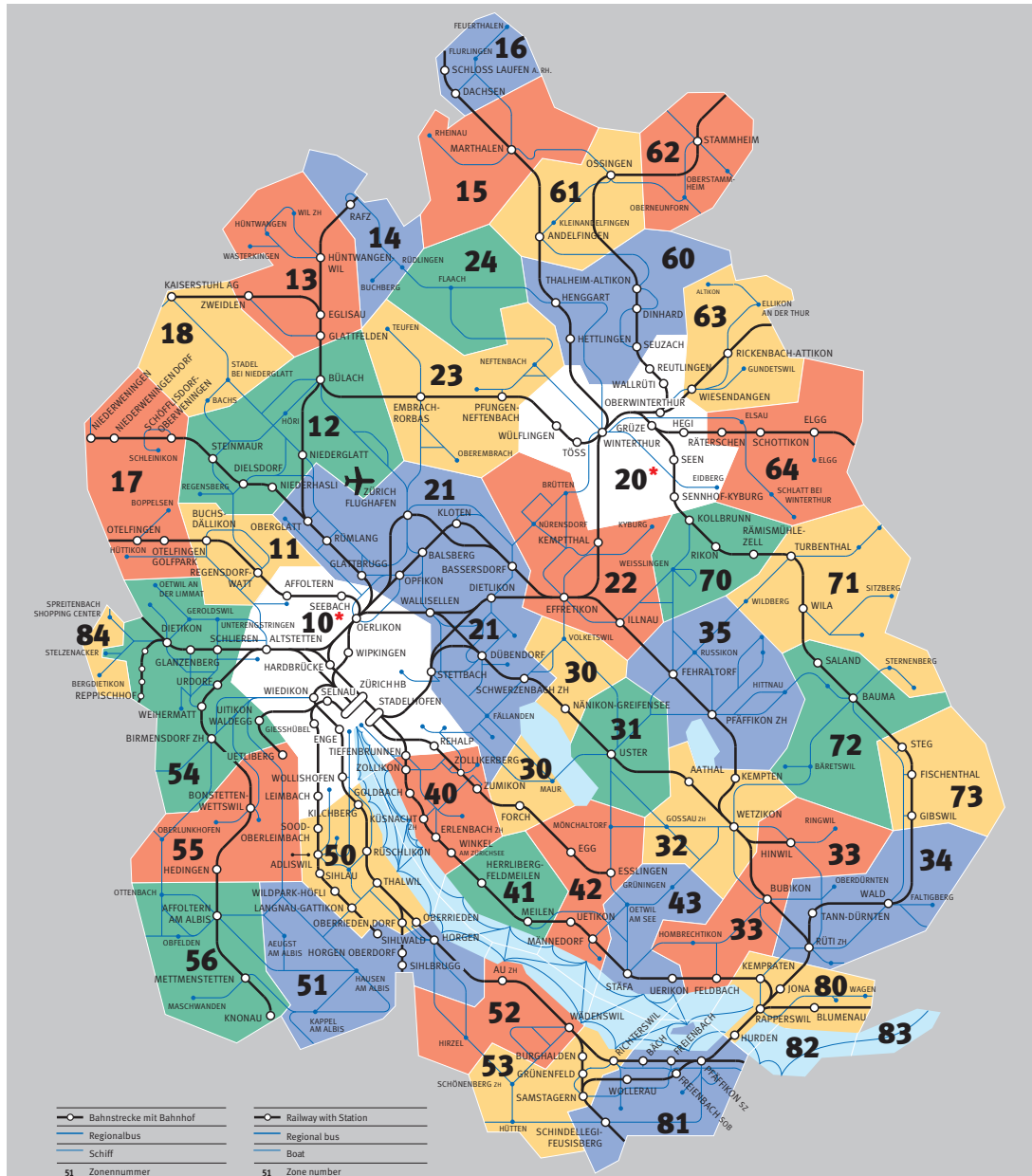
### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

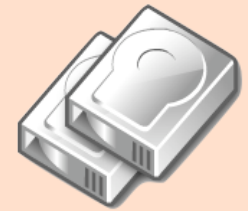
### Wrap-Up

# Points, Lines, and Regions



## Multi-Dimensional Indexing

Torsten Grust



B+-trees...

... over composite keys

Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

k-d Trees

Balanced Construction

K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

R-Trees

Searching and Inserting

Splitting R-Tree Nodes

UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

Spaces with High Dimensionality

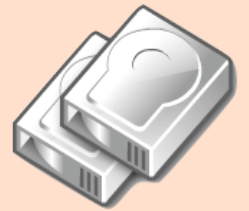
Wrap-Up

## ... More Solutions

### Recent proposals for multi-dimension index structures

Quad Tree [Finkel 1974]	K-D-B-Tree [Robinson 1981]
R-tree [Guttman 1984]	Grid File [Nievergelt 1984]
R <sup>+</sup> -tree [Sellis 1987]	LSD-tree [Henrich 1989]
R*-tree [Geckmann 1990]	hB-tree [Lomet 1990]
Vp-tree [Chiueh 1994]	TV-tree [Lin 1994]
UB-tree [Bayer 1996]	hB- $\Pi$ -tree [Evangelidis 1995]
SS-tree [White 1996]	X-tree [Berchtold 1996]
M-tree [Ciaccia 1996]	SR-tree [Katayama 1997]
Pyramid [Berchtold 1998]	Hybrid-tree [Chakrabarti 1999]
DABS-tree [Böhm 1999]	IQ-tree [Böhm 2000]
Slim-tree [Faloutsos 2000]	Landmark file [Böhm 2000]
P-Sphere-tree [Goldstein 2000]	A-tree [Sakurai 2000]

None of these provides a “fits all” solution.



#### B+-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

#### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

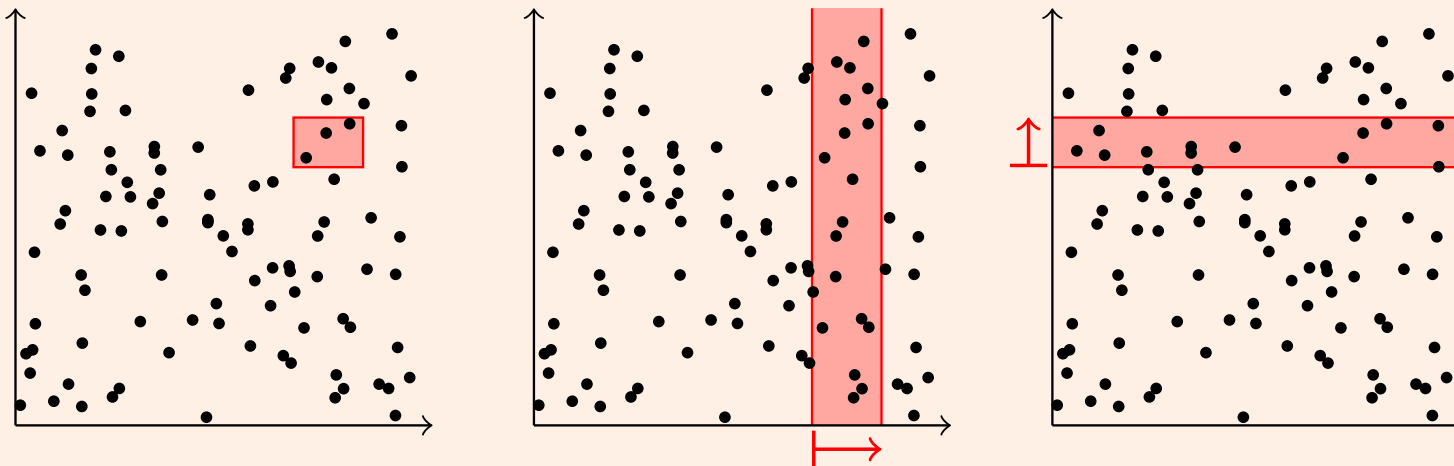
#### Spaces with High Dimensionality

#### Wrap-Up

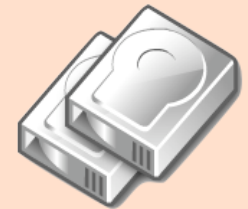
## Can We Just Use a B<sup>+</sup>-tree?

- Can two B<sup>+</sup>-trees, one over ZIPCODE and over REVENUE, adequately support a two-dimensional range query?

### Querying a rectangle



- Can only scan along **either** index at once, and both of them produce many **false hits**.
- If all you have are these two indices, you can do **index intersection**: perform both scans in separation to obtain the *rids* of candidate tuples. Then compute the intersection between the two *rid* lists (DB2: IXAND).



#### B<sup>+</sup>-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

#### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering  
B<sup>+</sup>-Trees over Z-Codes  
Range Queries

#### Spaces with High Dimensionality

#### Wrap-Up

# IBM DB2: Index Intersection

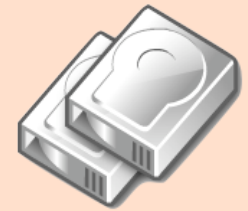
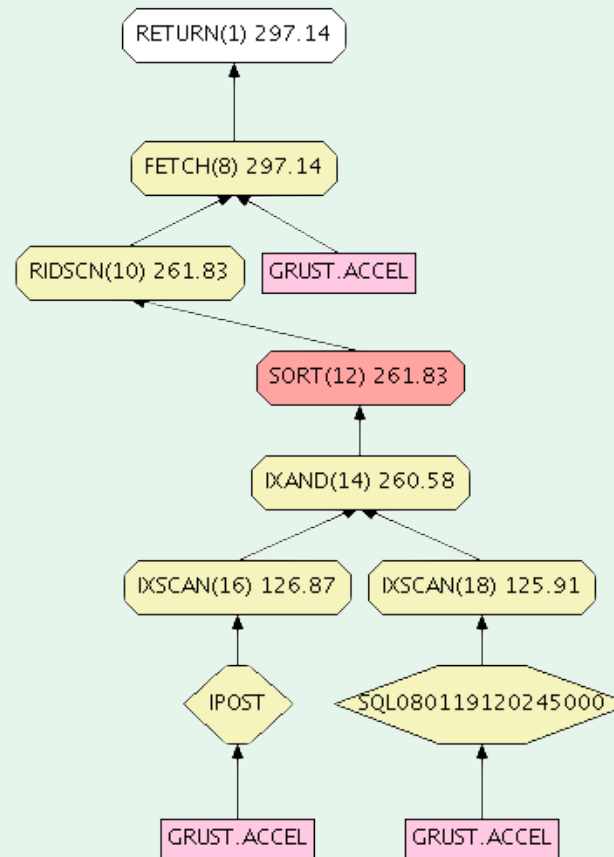
## DB2<sup>®</sup> IBM DB2 uses index intersection (operator IXAND)

```
1 SELECT *
2 FROM ACCEL
3 WHERE PRE BETWEEN 0 AND 10000
4 AND POST BETWEEN 10000 AND 20000
```

Relevant indexes defined over table ACCEL:

- 1 IPOST over column POST
- 2 SQL0801192024500 over column PRE (primary key)

## DB2<sup>®</sup> Plan selected by the query optimizer



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

### UB-Trees

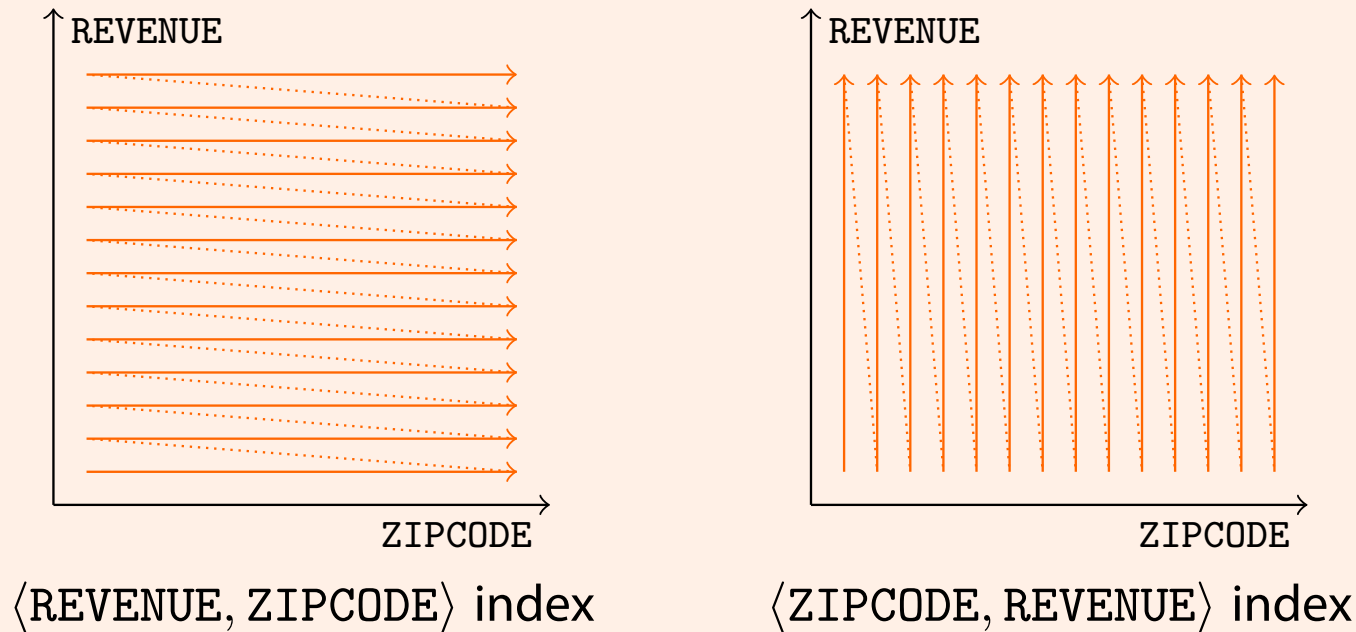
Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Can Composite Keys Help?

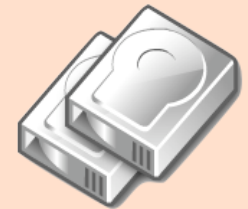
## Indexes with composite keys



- **Almost the same thing!**  
Indices over composite keys are **not symmetric**: The major attribute dominates the organization of the  $B^+$ -tree.
- **But:** Since the minor attribute is also stored in the index (part of the keys  $k$ ), we may discard non-qualifying tuples **before** fetching them from the data pages.

## Multi-Dimensional Indexing

Torsten Grust



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

## R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

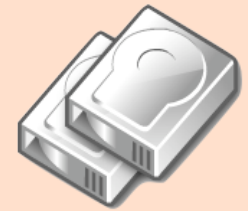
## UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

## Spaces with High Dimensionality

## Wrap-Up

# Multi-Dimensional Indices



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

## Spaces with High Dimensionality

## Wrap-Up

- B<sup>+</sup>-trees can answer **one-dimensional** queries **only**.<sup>1</sup>
- We would like to have a **multi-dimensional index structure** that
  - is **symmetric** in its dimensions,
  - **clusters** data in a space-aware fashion,
  - is **dynamic** with respect to updates, and
  - provides good support for useful queries.
- We will start with data structures that have been designed for **in-memory** use, then tweak them into **disk-aware** database indices (e.g., organize data in a page-wise fashion).

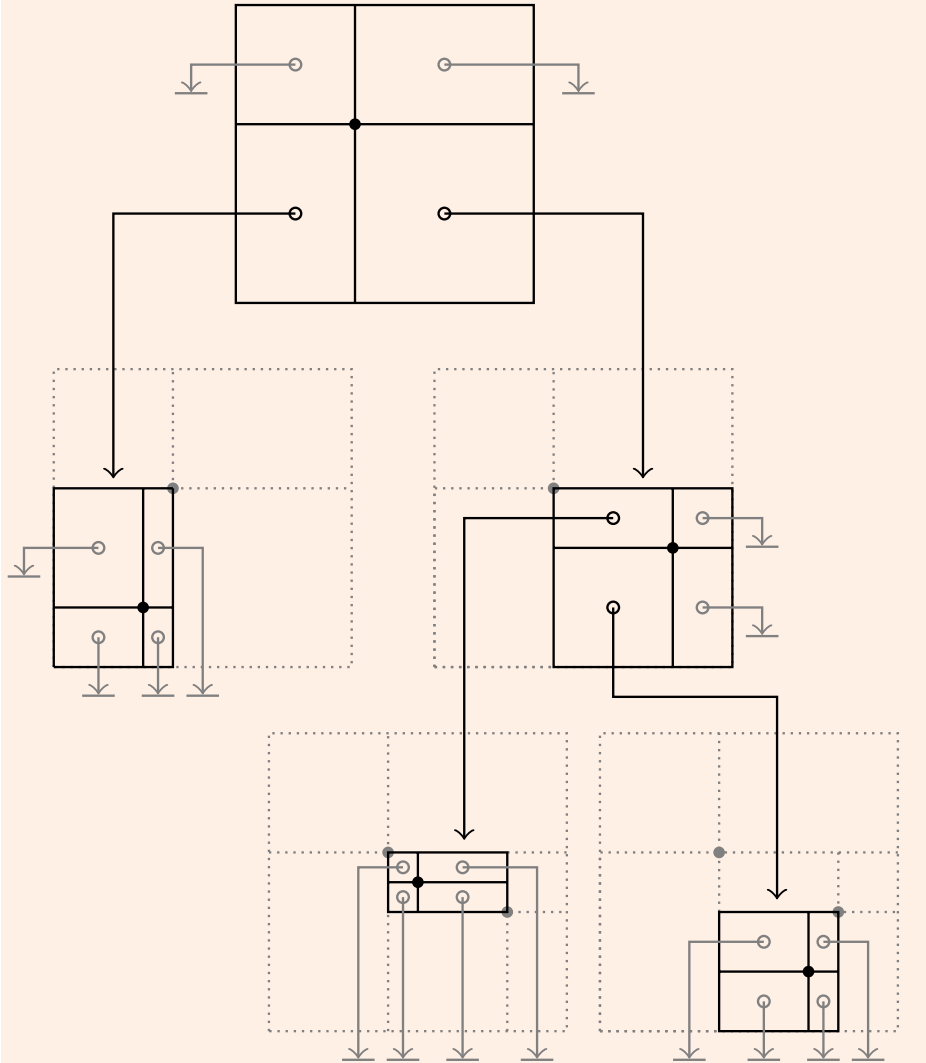
---

<sup>1</sup>Toward the end of this chapter, we will see UB-trees, a nifty trick that uses B<sup>+</sup>-trees to support some multi-dimensional queries.

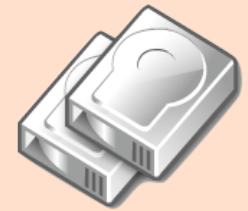
# “Binary” Search Tree

In  $k$  dimensions, a “**binary tree**” becomes a  $2^k$ -ary tree.

## Point quad tree ( $k = 2$ )



- Each data point **partitions** the data space into  $2^k$  **disjoint regions**.
- In each node, a region points to another node (representing a refined partitioning) or to a special **null** value.
- This data structure is a **point quad tree**.  
↗ Finkel and Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, vol. 4, 1974.



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

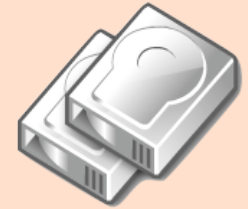
### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

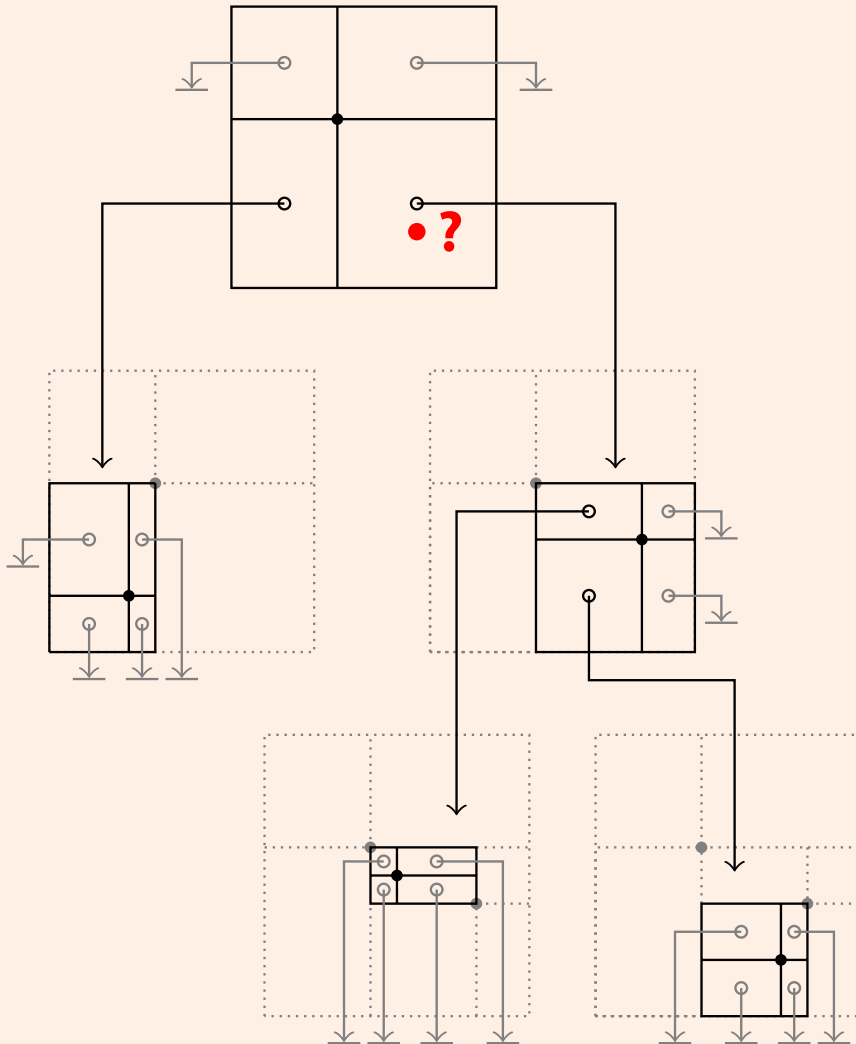
### Spaces with High Dimensionality

### Wrap-Up

# Searching a Point Quad Tree



## Point quad tree ( $k = 2$ )



## Point quad tree search

```
1 Function: p_search( $q$ ,  $node$ )
2 if  $q$  matches data point in  $node$  then
3   | return data point;
4 else
5   |  $P \leftarrow$  partition containing  $q$ ;
6   | if  $P$  points to null then
7     | return not found;
8   | else
9     |  $node' \leftarrow$  node pointed to by  $P$ ;
10    | return p_search( $q$ ,  $node'$ );
```

```
1 Function: pointsearch( $q$ )
2 return p_search( $q$ ,  $root$ );
```

### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

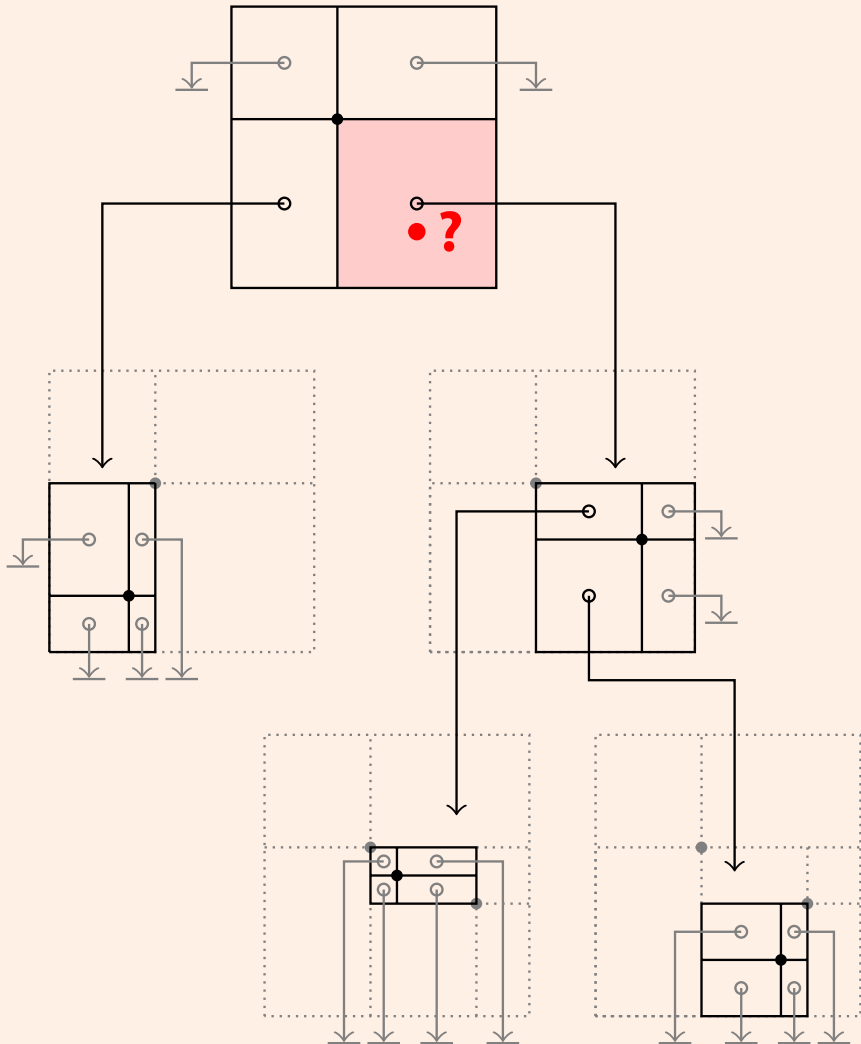
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Searching a Point Quad Tree

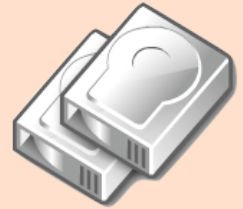
## Point quad tree ( $k = 2$ )



## Point quad tree search

```
1 Function: p_search( $q$ ,  $node$ )  
2 if  $q$  matches data point in  $node$  then  
3   | return data point;  
4 else  
5   |  $P \leftarrow$  partition containing  $q$ ;  
6   | if  $P$  points to null then  
7     | return not found;  
8   | else  
9     |  $node' \leftarrow$  node pointed to by  $P$ ;  
10    | return p_search( $q$ ,  $node'$ );
```

```
1 Function: pointsearch( $q$ )  
2 return p_search( $q$ ,  $root$ );
```



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

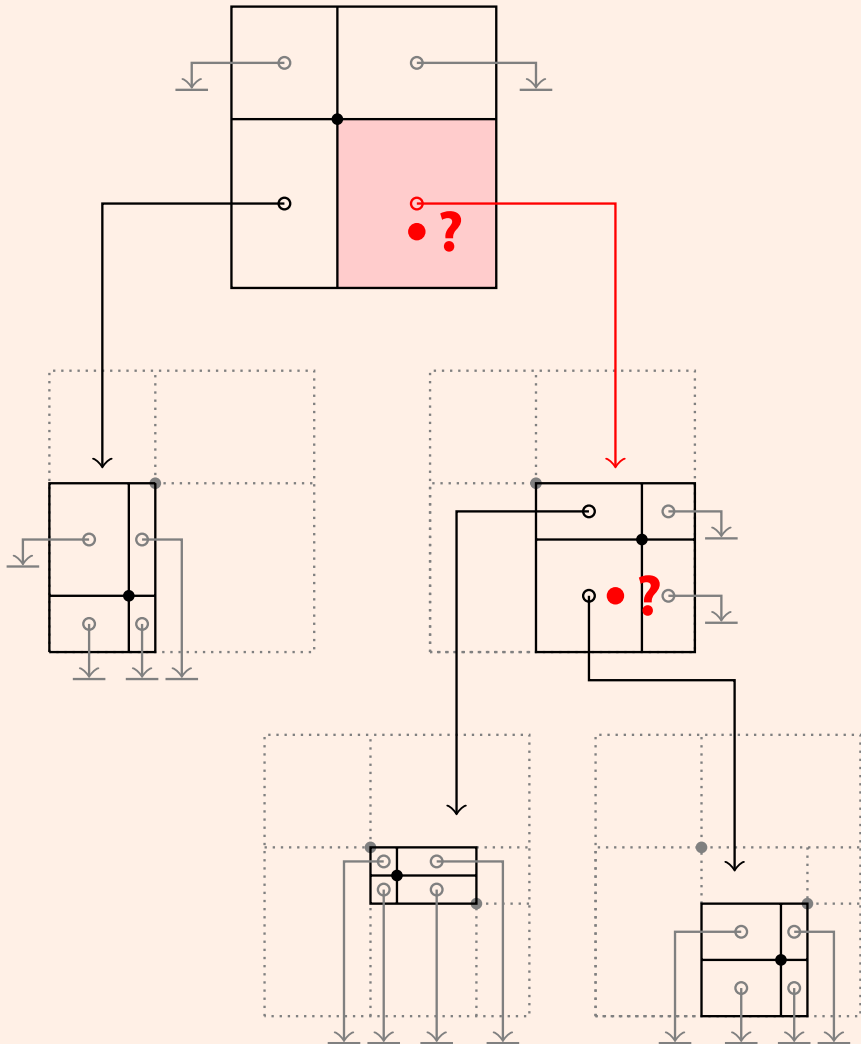
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Searching a Point Quad Tree

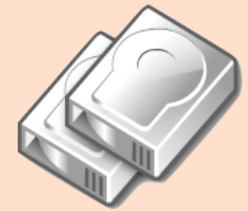
## Point quad tree ( $k = 2$ )



## Point quad tree search

```
1 Function: p_search( $q$ ,  $node$ )  
2 if  $q$  matches data point in  $node$  then  
3   | return data point;  
4 else  
5   |  $P \leftarrow$  partition containing  $q$ ;  
6   | if  $P$  points to null then  
7     | return not found;  
8   | else  
9     |  $node' \leftarrow$  node pointed to by  $P$ ;  
10    | return p_search( $q$ ,  $node'$ );
```

```
1 Function: pointsearch( $q$ )  
2 return p_search( $q$ ,  $root$ );
```



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

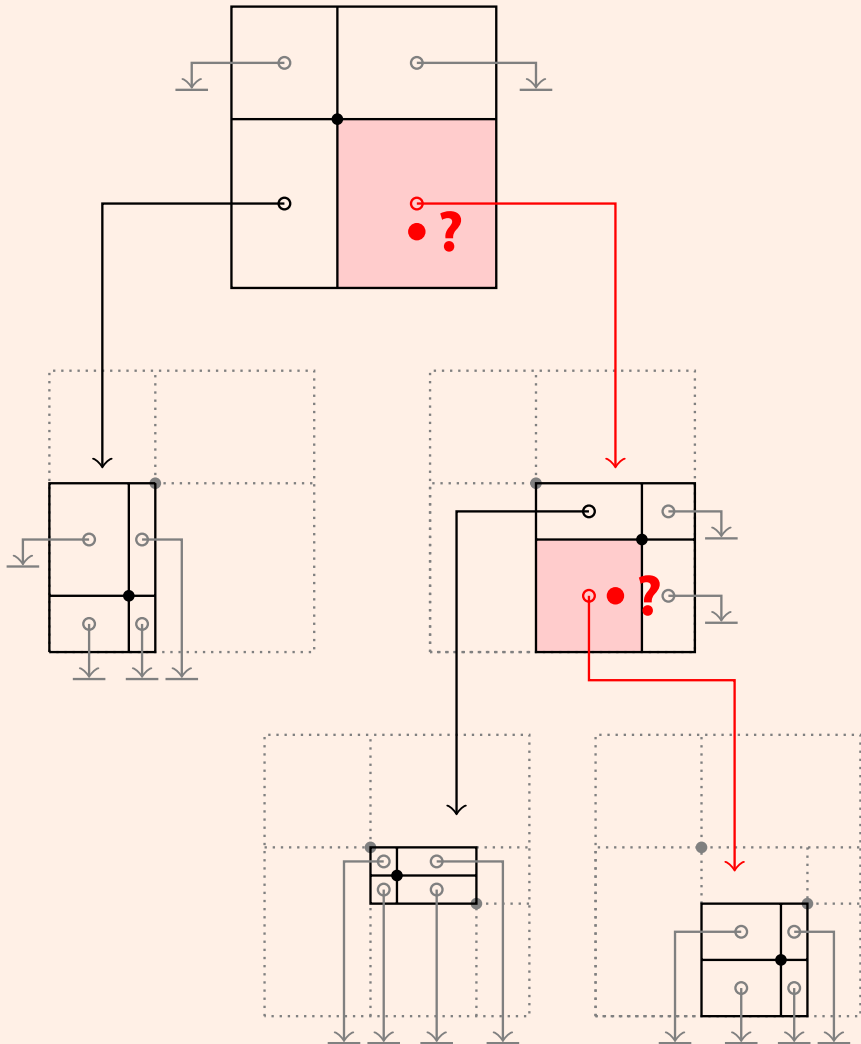
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Searching a Point Quad Tree

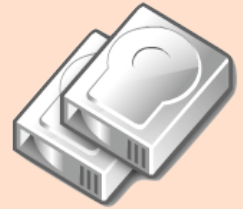
## Point quad tree ( $k = 2$ )



## Point quad tree search

```
1 Function: p_search( $q$ ,  $node$ )  
2 if  $q$  matches data point in  $node$  then  
3   | return data point;  
4 else  
5   |  $P \leftarrow$  partition containing  $q$ ;  
6   | if  $P$  points to null then  
7     | return not found;  
8   | else  
9     |  $node' \leftarrow$  node pointed to by  $P$ ;  
10    | return p_search( $q$ ,  $node'$ );
```

```
1 Function: pointsearch( $q$ )  
2 return p_search( $q$ ,  $root$ );
```



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

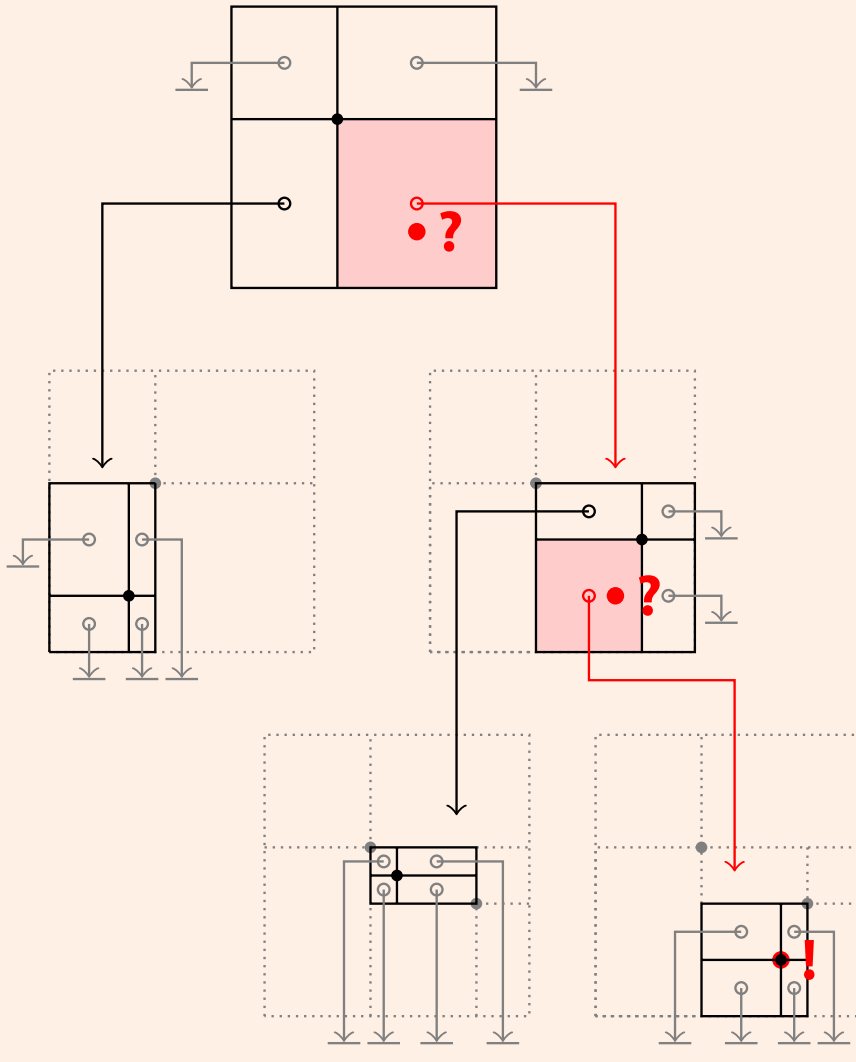
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Searching a Point Quad Tree

## Point quad tree ( $k = 2$ )



## Point quad tree search

```

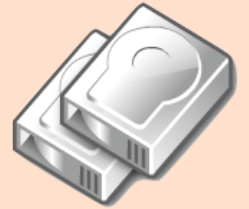
1 Function: p_search( $q$ ,  $node$ )
2 if  $q$  matches data point in  $node$  then
3   | return data point;
4 else
5   |  $P \leftarrow$  partition containing  $q$ ;
6   | if  $P$  points to null then
7     | return not found;
8   | else
9     |  $node' \leftarrow$  node pointed to by  $P$ ;
10    | return p_search( $q$ ,  $node'$ );

```

```

1 Function: pointsearch( $q$ )
2 return p_search( $q$ ,  $root$ );

```



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

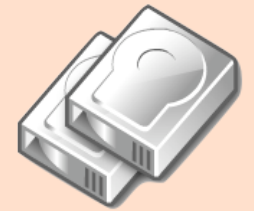
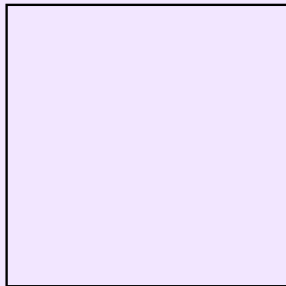
## Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

### Example (Insertions into an empty point quad tree)



#### B+-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

#### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

#### Spaces with High Dimensionality

#### Wrap-Up

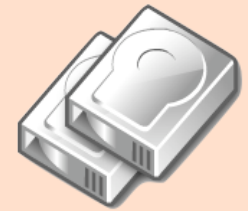
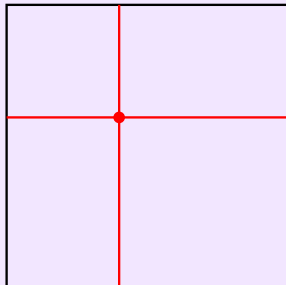
## Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

### Example (Insertions into an empty point quad tree)



#### B+-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

#### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

#### Spaces with High Dimensionality

#### Wrap-Up

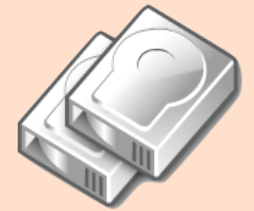
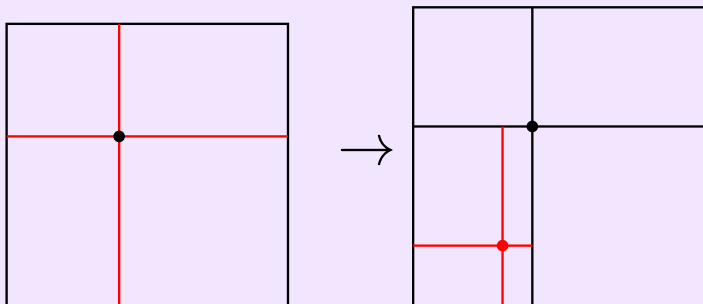
## Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

### Example (Insertions into an empty point quad tree)



#### B+-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

#### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

#### Spaces with High Dimensionality

#### Wrap-Up

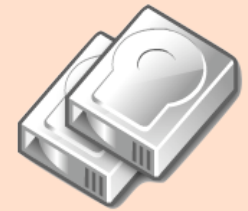
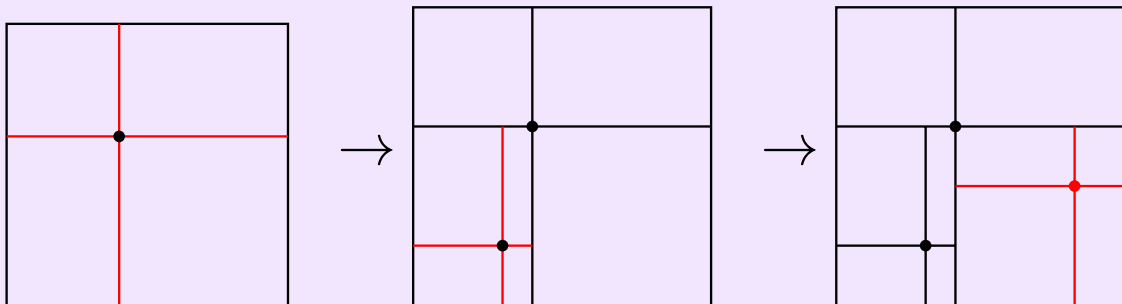
## Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

### Example (Insertions into an empty point quad tree)



#### B+-trees...

... over composite keys

#### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

#### k-d Trees

Balanced Construction

#### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

#### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

#### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

#### Spaces with High Dimensionality

#### Wrap-Up

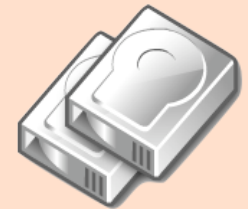
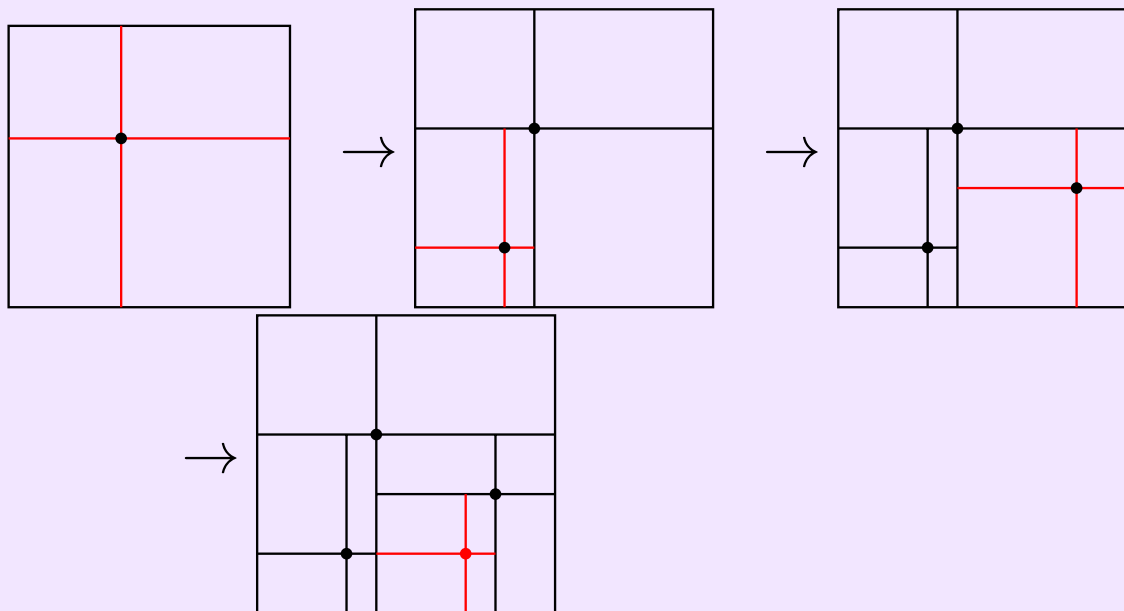
# Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

## Example (Insertions into an empty point quad tree)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

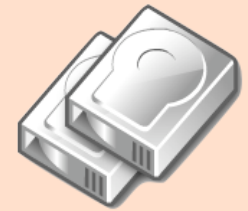
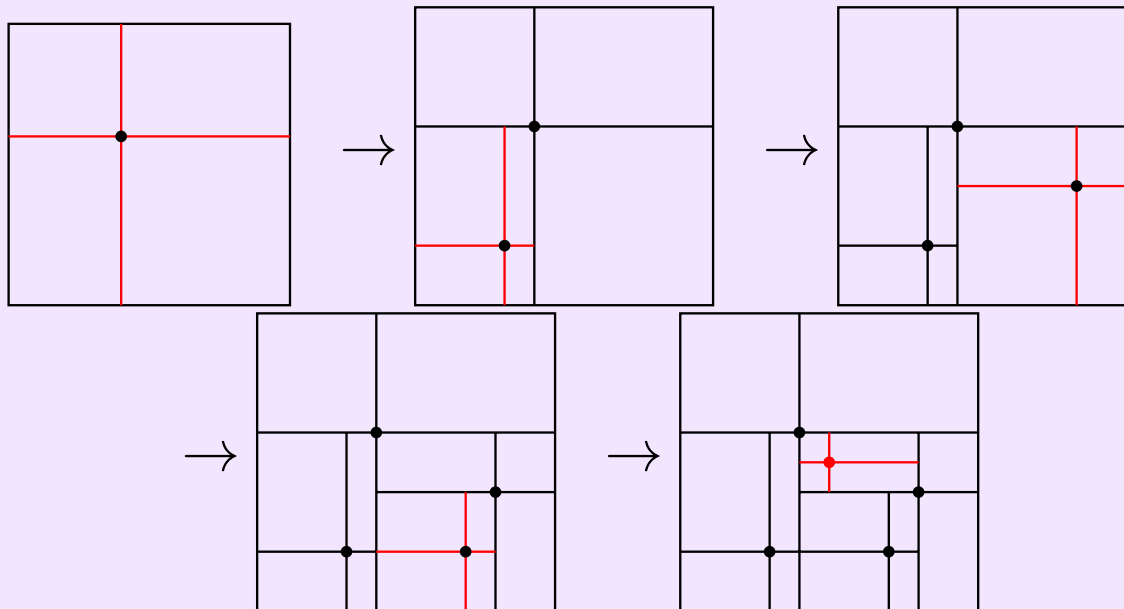
# Inserting into a Point Quad Tree

**Inserting** a point  $q_{\text{new}}$  into a quad tree happens analogously to an insertion into a binary tree:

- 1 **Traverse** the tree just like during a search for  $q_{\text{new}}$  until you encounter a partition  $P$  with a **null** pointer.
- 2 Create a **new node**  $n'$  that spans the same area as  $P$  and is partitioned by  $q_{\text{new}}$ , with all partitions pointing to **null**.
- 3 Let  $P$  point to  $n'$ .

Note that this procedure does **not** keep the tree **balanced**.

## Example (Insertions into an empty point quad tree)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Range Queries

To evaluate a **range query**<sup>2</sup>, we may need to follow **several** children of a quad tree node *node*:

## Point quad tree range search

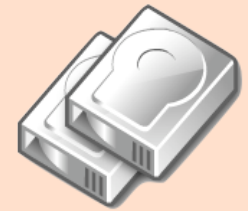
```
1 Function: r_search (Q, node)
2 if data point in node is in Q then
3   | append data point to result ;
4 foreach partition P in node that intersects with Q do
5   | node' ← node pointed to by P ;
6   | r_search (Q, node') ;
```

---

```
1 Function: regionsearch (Q)
2 return r_search (Q, root) ;
```

---

<sup>2</sup>We consider **rectangular** regions only; other shapes may be answered by querying for the **bounding rectangle** and post-processing the output.



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# Point Quad Trees—Discussion

## Point quad trees

- ✓ are **symmetric** with respect to all dimensions and
- ✓ can answer **point queries** and **region queries**.

## But

- ✗ the shape of a quad tree depends on the **insertion order** of its content, in the worst case **degenerates** into a **linked list**,
- ✗ **null** pointers are **space inefficient** (particularly for large  $k$ ).

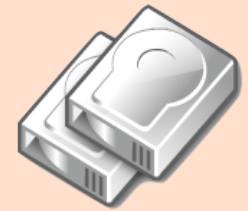
In addition,

- they can only store **point data**.

Also remember that quad trees were designed for **main memory** operation.

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

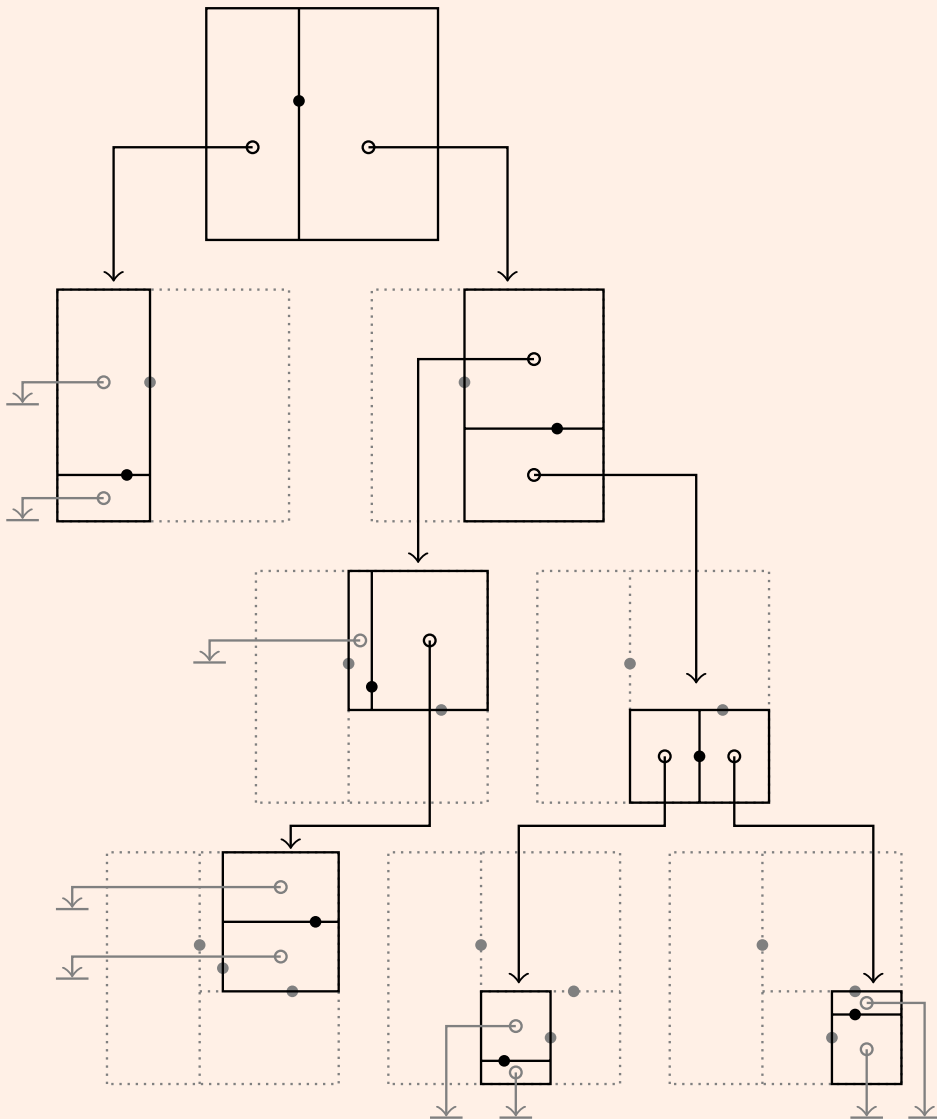
Range Queries

### Spaces with High Dimensionality

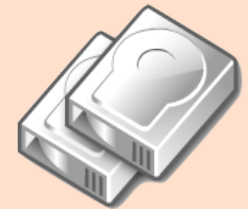
### Wrap-Up

# k-d Trees

## Sample k-d tree ( $k = 2$ )



- Index  $k$ -dimensional data, but keep the tree **binary**.
- For each **tree level**  $l$  use a different **discriminator dimension**  $d_l$  along which to **partition**.
  - Typically: **round robin**
- This is a **k-d tree**.
  - ↗ Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Comm. ACM*, vol. 18, no. 9, Sept. 1975.



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

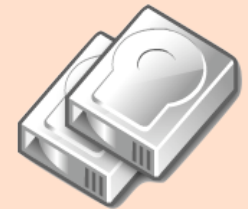
# k-d Trees

k-d trees inherit the positive properties of the point quad trees, but improve on **space efficiency**.

For a given point set, we can also construct a **balanced** k-d tree:<sup>3</sup>

## k-d tree construction (Bentley's OPTIMIZE algorithm)

```
1 Function: kdtree (pointset, level)
2 if pointset is empty then
3   return null ;
4 else
5    $p \leftarrow$  median from pointset (along  $d_{level}$ ) ;
6    $points_{left} \leftarrow \{v \in pointset \text{ where } v_{d_{level}} < p_{d_{level}}\}$ ;
7    $points_{right} \leftarrow \{v \in pointset \text{ where } v_{d_{level}} \geq p_{d_{level}}\}$ ;
8    $n \leftarrow$  new k-d tree node, with data point  $p$  ;
9    $n.left \leftarrow$  kdtree ( $points_{left}$ ,  $level + 1$ ) ;
10   $n.right \leftarrow$  kdtree ( $points_{right}$ ,  $level + 1$ ) ;
11  return  $n$  ;
```



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

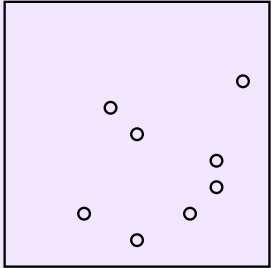
### Spaces with High Dimensionality

### Wrap-Up

<sup>3</sup> $v_i$ : coordinate  $i$  of point  $v$ .

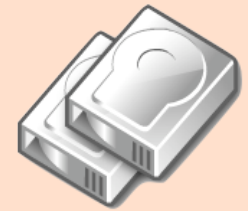
# Balanced $k$ -d Tree Construction

## Example (Step-by-step balanced $k$ -d tree construction)



### Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

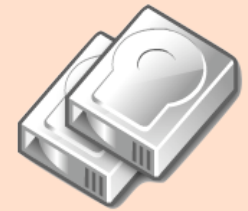
### Spaces with High Dimensionality

### Wrap-Up

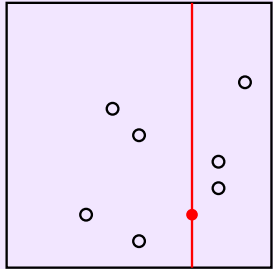
# Balanced $k$ -d Tree Construction

Multi-Dimensional  
Indexing

Torsten Grust



## Example (Step-by-step balanced $k$ -d tree construction)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

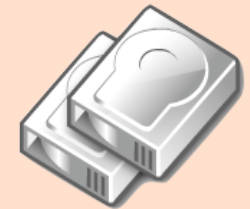
B+-Trees over Z-Codes

Range Queries

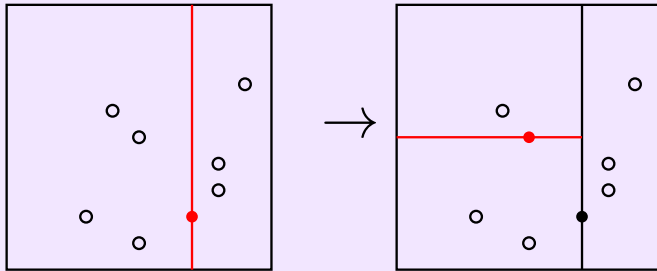
### Spaces with High Dimensionality

### Wrap-Up

# Balanced $k$ -d Tree Construction



## Example (Step-by-step balanced $k$ -d tree construction)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

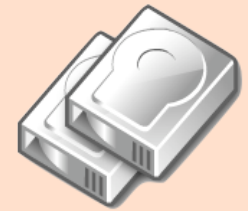
B+-Trees over Z-Codes

Range Queries

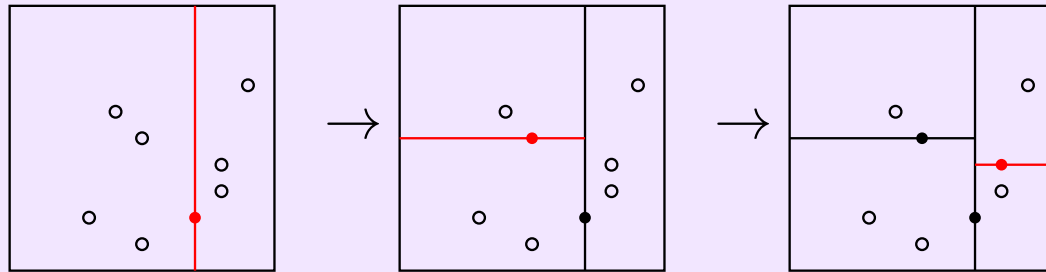
### Spaces with High Dimensionality

### Wrap-Up

# Balanced $k$ -d Tree Construction



## Example (Step-by-step balanced $k$ -d tree construction)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

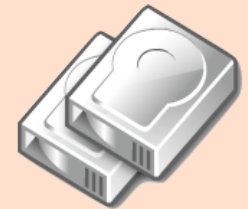
B+-Trees over Z-Codes

Range Queries

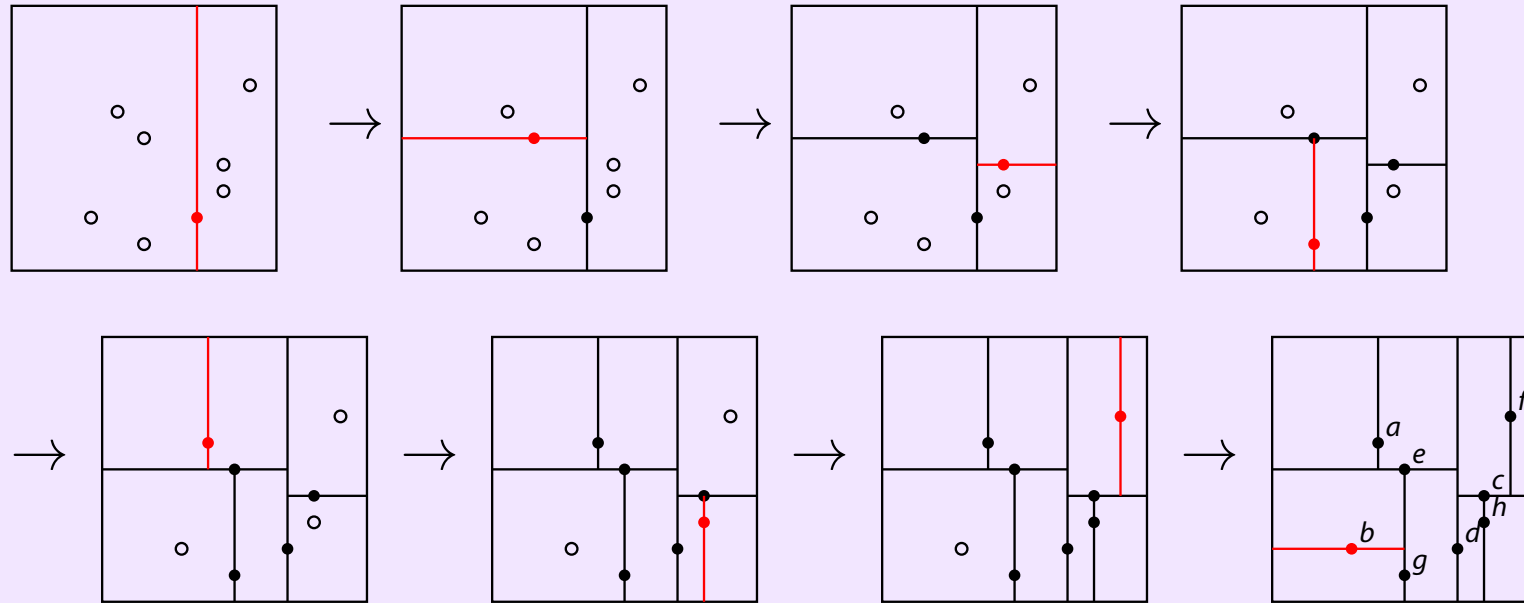
### Spaces with High Dimensionality

### Wrap-Up

# Balanced $k$ -d Tree Construction



## Example (Step-by-step balanced $k$ -d tree construction)



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

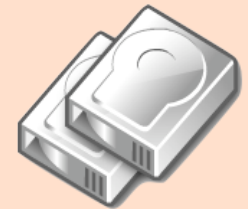
### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

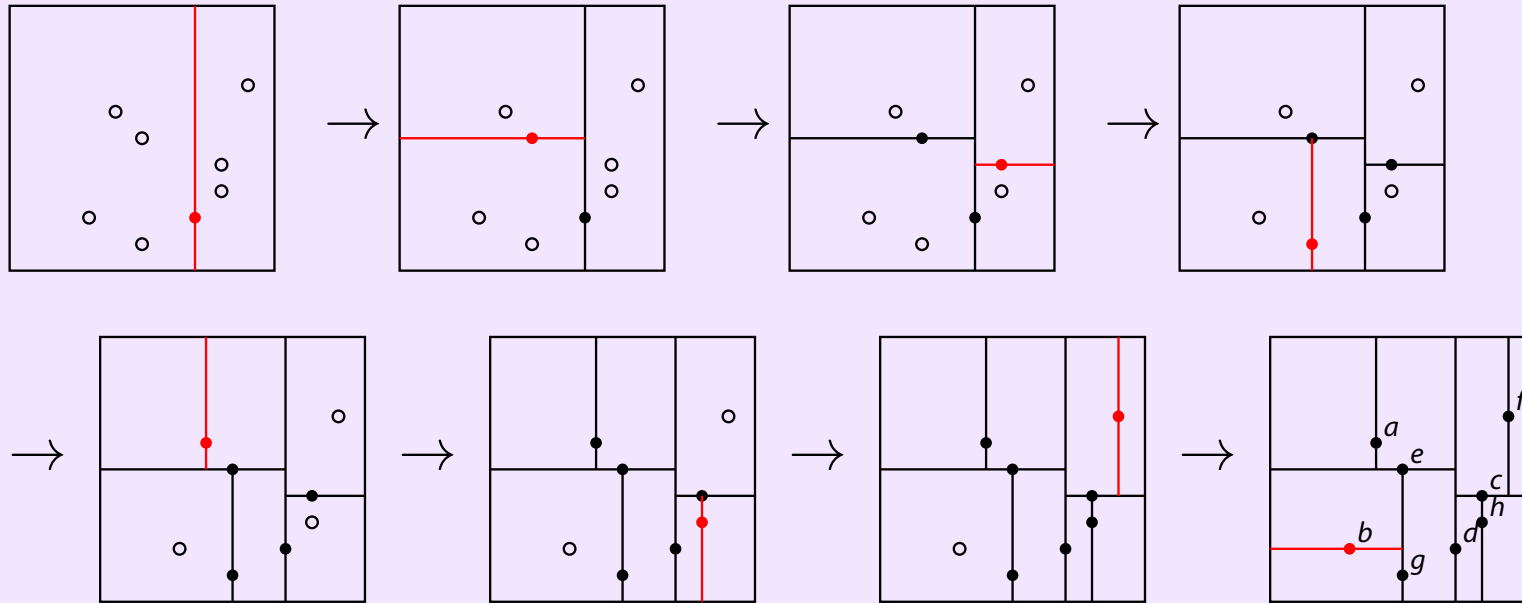
### Spaces with High Dimensionality

### Wrap-Up

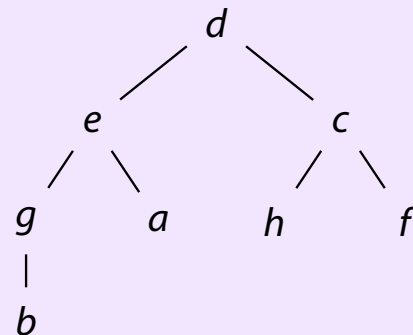
# Balanced $k$ -d Tree Construction



## Example (Step-by-step balanced $k$ -d tree construction)



### Resulting tree shape:



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# K-D-B-Trees

$k$ -d trees improve on some of the deficiencies of point quad trees:

- ✓ We can **balance** a  $k$ -d tree by **re-building** it.  
(For a limited number of points and in-memory processing, this may be sufficient.)
- ✓ We're no longer wasting big amounts of **space**.

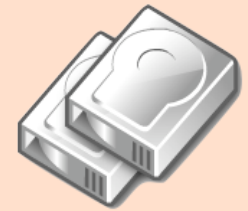
It's time to bring  $k$ -d trees to the disk. The **K-D-B-Tree**

- uses **pages** as an organizational unit  
(*e.g.*, each node in the K-D-B-tree fills a page) and
- uses a  **$k$ -d tree-like layout** to organize each page.

↗ John T. Robinsion. The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. *SIGMOD 1981*.

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

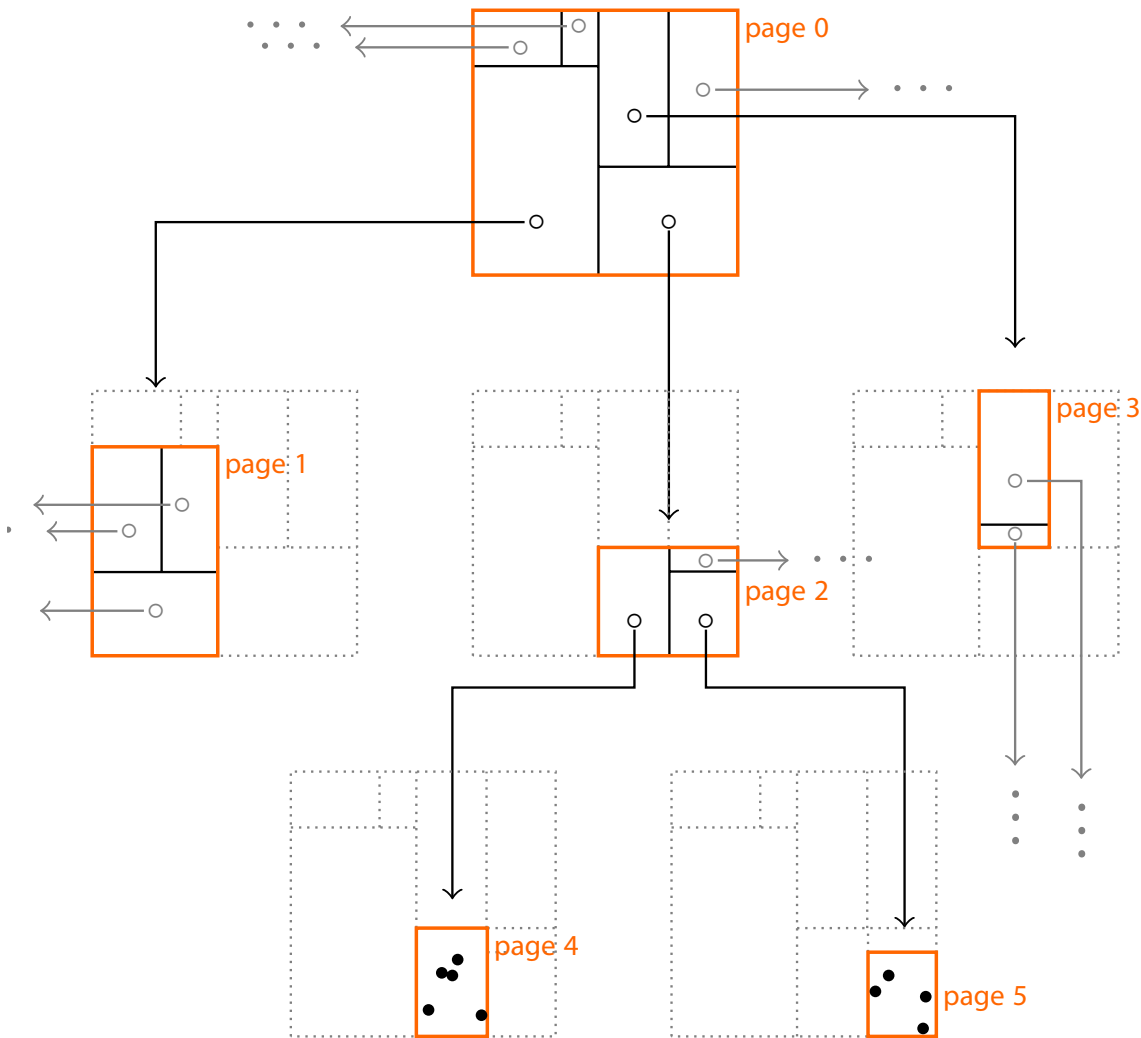
### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# K-D-B-Tree Idea

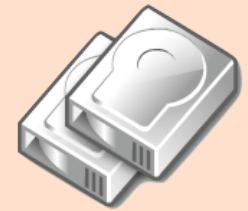


## Region pages:

- contain entries  $\langle \text{region}, \text{pageID} \rangle$
- no **null** pointers
- form a **balanced** tree
- all regions **disjoint** and **rectangular**

## Point pages:

- contain entries  $\langle \text{point}, \text{rid} \rangle$
- $\rightsquigarrow$  B<sup>+</sup>-tree leaf nodes



### B<sup>+</sup>-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

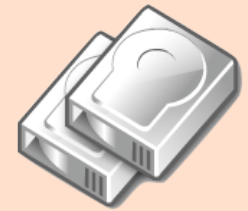
### UB-Trees

Bit Interleaving / Z-Ordering  
B<sup>+</sup>-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# K-D-B-Tree Operations



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

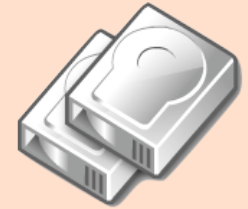
Range Queries

## Spaces with High Dimensionality

## Wrap-Up

- **Searching** a K-D-B-Tree works straightforwardly:
  - Within each page determine all regions  $R_i$  that contain the query point  $q$  (intersect with the query region  $Q$ ).
  - For each of the  $R_i$ , consult the page it points to and recurse.
  - On point pages, fetch and return the corresponding record for each matching data point  $p_i$ .
- When **inserting** data, we keep the K-D-B-Tree **balanced**, much like we did in the **B<sup>+</sup>-tree**:
  - Simply insert a  $\langle region, pageID \rangle$  ( $\langle point, rid \rangle$ ) entry into a region page (point page) if there's **sufficient space**.
  - **Otherwise, split** the page.

# K-D-B-Tree: Splitting a Point Page



## Splitting a point page $p$

- 1 **Choose a dimension**  $i$  and an  $i$ -coordinate  $x_i$  along which to split, such that the split will result in two pages that are not overfull.
- 2 **Move** data points  $p$  with  $p_i < x_i$  and  $p_i \geq x_i$  to new pages  $p_{\text{left}}$  and  $p_{\text{right}}$  (respectively).
- 3 Replace  $\langle \text{region}, p \rangle$  in the **parent** of  $p$  with  $\langle \text{left region}, p_{\text{left}} \rangle$  and  $\langle \text{right region}, p_{\text{right}} \rangle$ .

Step 3 may cause an **overflow** in  $p$ 's parent and, hence, lead to a **split** of a **region page**.

### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

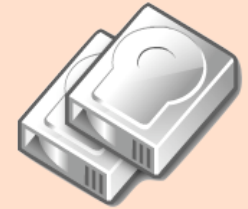
B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up

# K-D-B-Tree: Splitting a Region Page



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

## UB-Trees

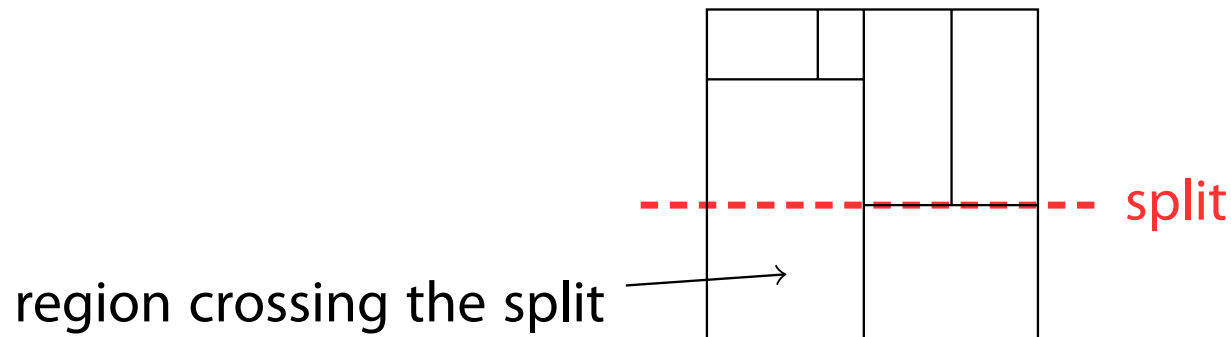
Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

## Spaces with High Dimensionality

## Wrap-Up

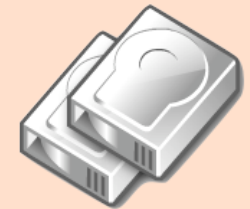
- Splitting a **point page** and moving its data **points** to the resulting pages is straightforward.
- In case of a **region page split**, by contrast, some **regions** may intersect with **both** sides of the split.

Consider, *e.g.*, page 0 on slide 19:

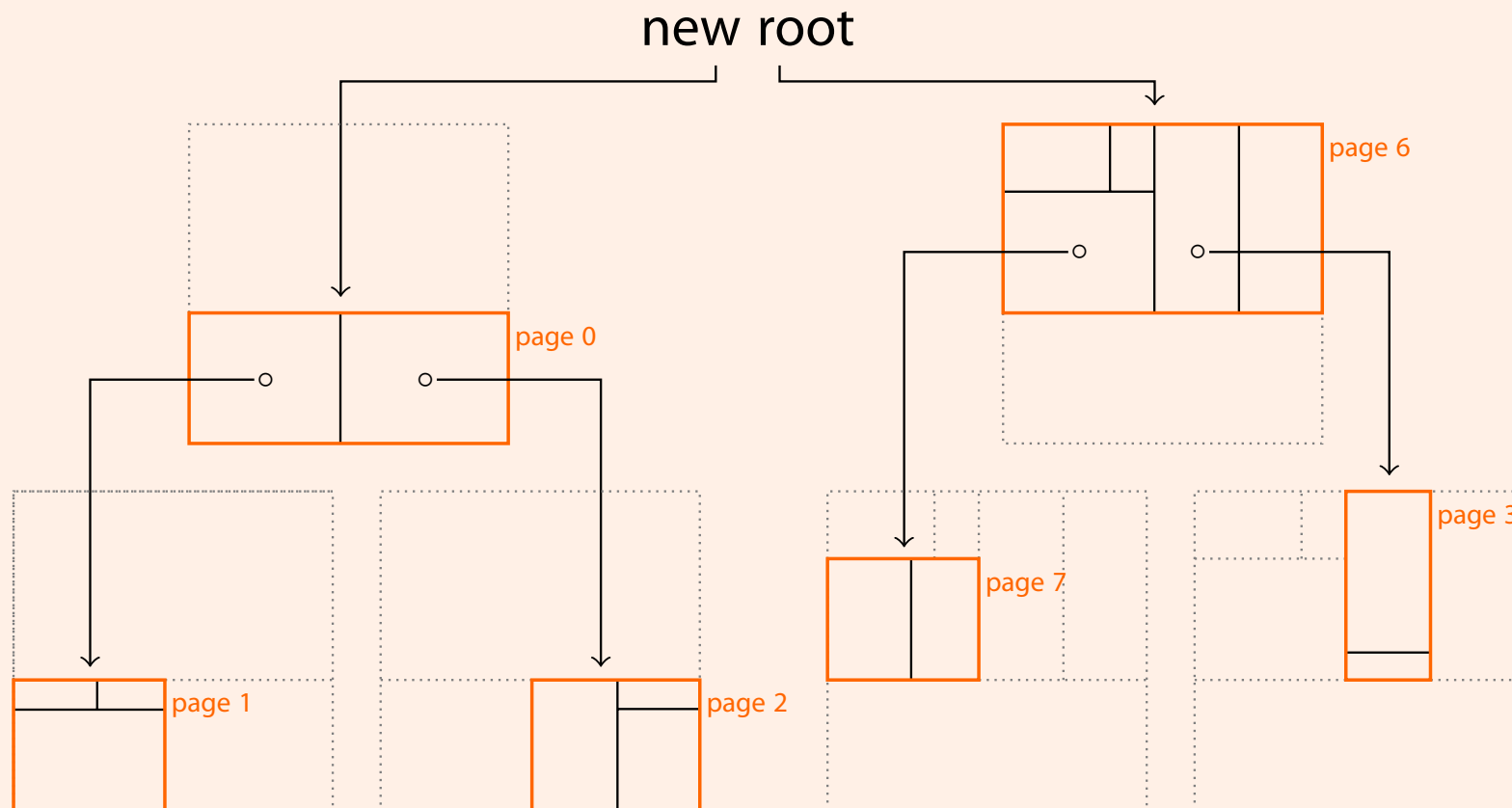


- Such regions need to be **split**, too.
- This can cause a **recursive** splitting **downward** (!) the tree.

# Example: Page 0 Split in K-D-B-Tree of slide 19



## Split of region page 0



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

- Root page 0  $\Rightarrow$  pages 0 and 6 ( $\rightsquigarrow$  creation of new root).
- Region page 1  $\Rightarrow$  pages 1 and 7 (point pages not shown).

# K-D-B-Trees: Discussion

## K-D-B-Trees

- ✓ are **symmetric** with respect to all dimensions,
- ✓ **cluster** data in a space-aware and page-oriented fashion,
- ✓ are **dynamic** with respect to updates, and
- ✓ can answer **point queries** and **region queries**.

## However,

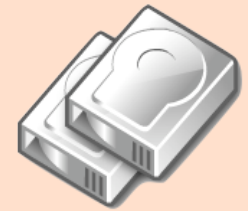
- we still don't have support for **region data** and
- K-D-B-Trees (like *k-d* trees) will not handle **deletes** dynamically.

This is because we always partitioned the data space such that

- every region is **rectangular** and
- regions never **intersect**.

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

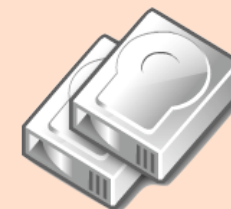
Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

### Wrap-Up



**R-trees** do not have the disjointness requirement:

- R-tree inner or leaf nodes contain  $\langle region, pageID \rangle$  or  $\langle region, rid \rangle$  entries (respectively).  
*region* is the **minimum bounding rectangle** that spans all data items reachable by the respective pointer.
- Every node contains between  $d$  and  $2d$  entries ( $\rightsquigarrow$  B<sup>+</sup>-tree).<sup>4</sup>
- **Insertion** and **deletion** algorithms keep an R-tree **balanced** at all times.

R-trees allow the storage of **point and region data**.

↗ Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD 1984*.

### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

### Spaces with High Dimensionality

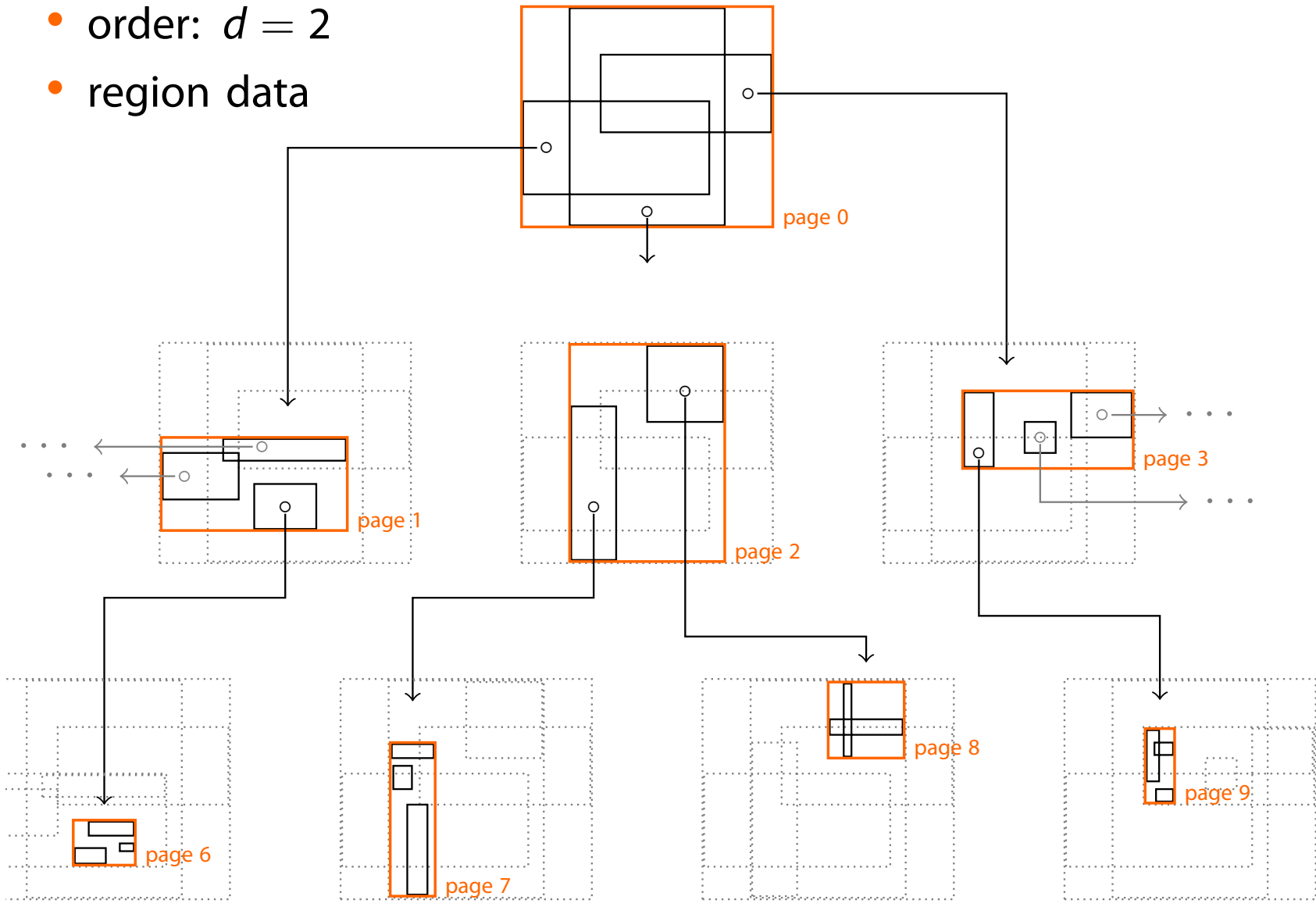
### Wrap-Up

---

<sup>4</sup>except the root node

# R-Tree: Example

- order:  $d = 2$
- region data

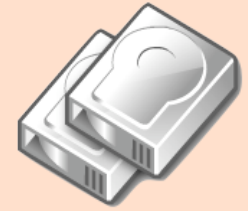


inner nodes

leaf nodes

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

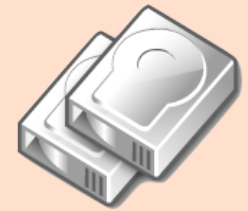
### Wrap-Up

# R-Tree: Searching and Inserting

While **searching** an R-tree, we may have to descend into more than one child node for point **and** region queries (↗ range search in point quad trees, slide 13).

## Inserting into an R-tree (cf. B<sup>+</sup>-tree insertion)

- 1 **Choose** a leaf node  $n$  to insert the new entry.
  - Try to minimize the necessary region enlargement(s).
- 2 If  $n$  is **full**, **split** it (resulting in  $n$  and  $n'$ ) and distribute old and new entries evenly across  $n$  and  $n'$ .
  - Splits may propagate bottom-up and eventually reach the root (↗ B<sup>+</sup>-tree).
- 3 After the insertion, some regions in the ancestor nodes of  $n$  may need to be **adjusted** to cover the new entry.



### B<sup>+</sup>-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

### R-Trees

Searching and Inserting

Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering

B<sup>+</sup>-Trees over Z-Codes

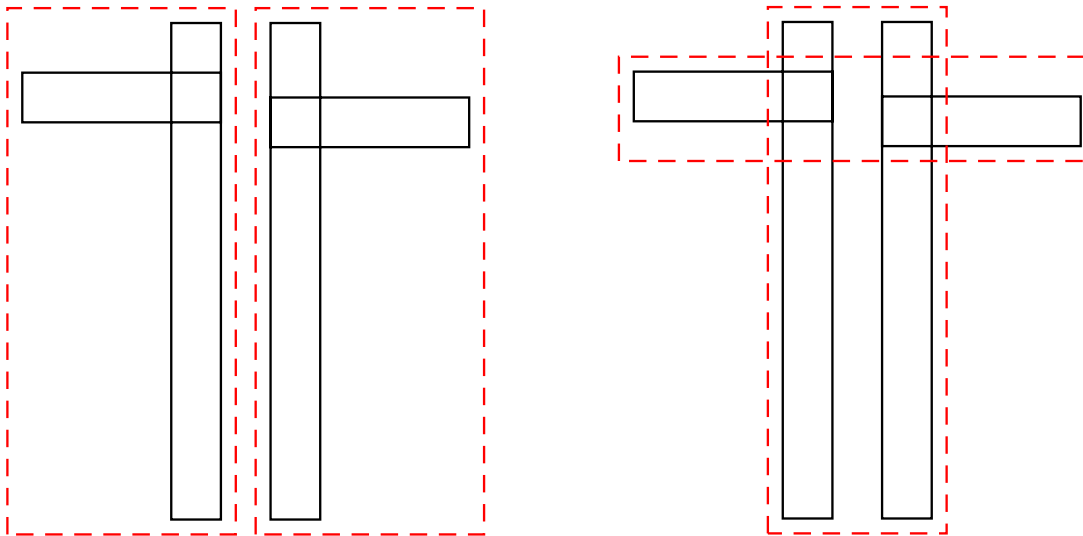
Range Queries

### Spaces with High Dimensionality

### Wrap-Up

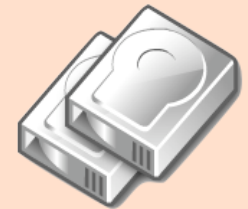
# Splitting an R-Tree Node

To **split** an R-tree node, we generally have more than one alternative:



**Heuristic: Minimize the totally covered area. But:**

- **Exhaustive** search for the best split is infeasible.
- Guttman proposes two ways to **approximate** the search.
- Follow-up papers (e.g., the R\*-tree) aim at improving the quality of node splits.



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

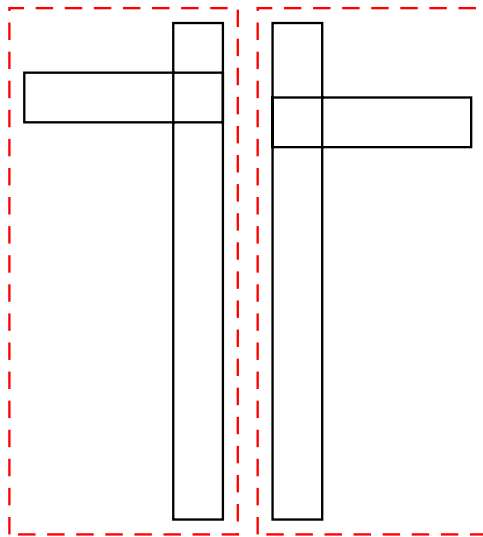
Range Queries

## Spaces with High Dimensionality

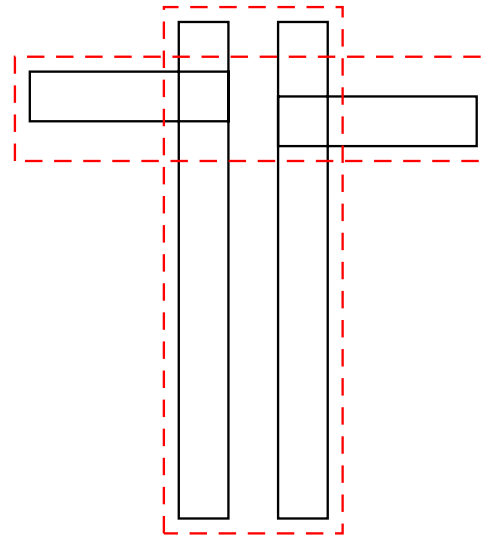
## Wrap-Up

# Splitting an R-Tree Node

To **split** an R-tree node, we generally have more than one alternative:



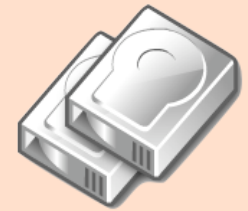
“bad” split



“good” split

**Heuristic: Minimize the totally covered area. But:**

- **Exhaustive** search for the best split is infeasible.
- Guttman proposes two ways to **approximate** the search.
- Follow-up papers (e.g., the R\*-tree) aim at improving the quality of node splits.



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

## Spaces with High Dimensionality

## Wrap-Up

# R-Tree: Deletes

All R-tree invariants (see 25) are maintained during **deletions**.

- 1 If an R-tree node  $n$  **underflows** (*i.e.*, less than  $d$  entries are left after a deletion), the whole node is **deleted**.
- 2 Then, all entries that existed in  $n$  are **re-inserted** into the R-tree.

Note that Step 1 may lead to a recursive deletion of  $n$ 's parent.

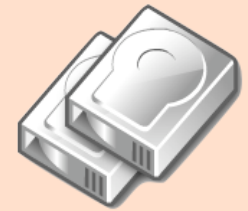
- Deletion, therefore, is a rather **expensive** task in an R-tree.

## Spatial indexing in mainstream database implementations

- Indexing in commercial database systems is typically based on **R-trees**.
- Yet, only few systems implement them out of the box (*e.g.*, PostgreSQL). Most require the licensing/use of specific extensions.

## Multi-Dimensional Indexing

Torsten Grust



### B+-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### k-d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

### UB-Trees

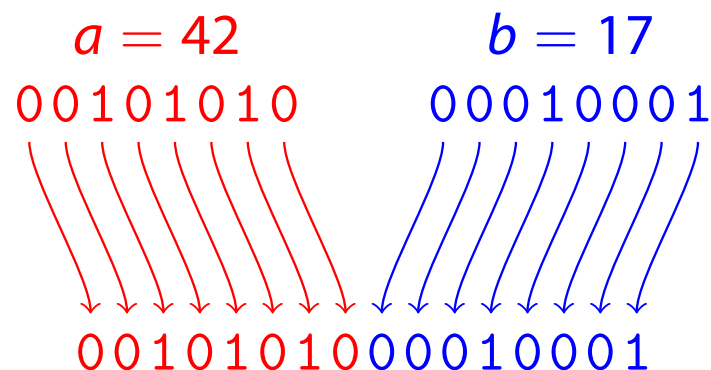
Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

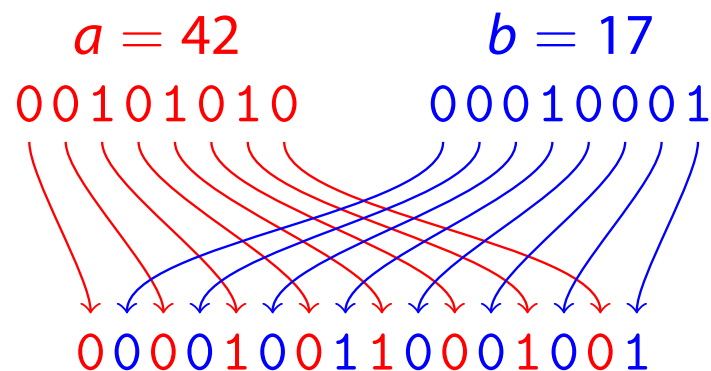
### Wrap-Up

# Bit Interleaving

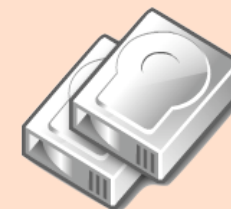
- We saw earlier that a B<sup>+</sup>-tree over **concatenated** fields  $\langle a, b \rangle$  does not help our case, because of the **asymmetry** between the role of  $a$  and  $b$  in the index key.
- What happens if we **interleave** the bits of  $a$  and  $b$  (hence, make the B<sup>+</sup>-tree “more symmetric”)?



$\langle a, b \rangle$  (concatenation)



$a$  and  $b$  interleaved



B<sup>+</sup>-trees...

... over composite keys

Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

k-d Trees

Balanced Construction

K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

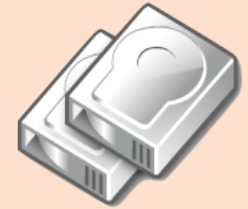
UB-Trees

Bit Interleaving / Z-Ordering  
B<sup>+</sup>-Trees over Z-Codes  
Range Queries

Spaces with High Dimensionality

Wrap-Up

# Z-Ordering



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

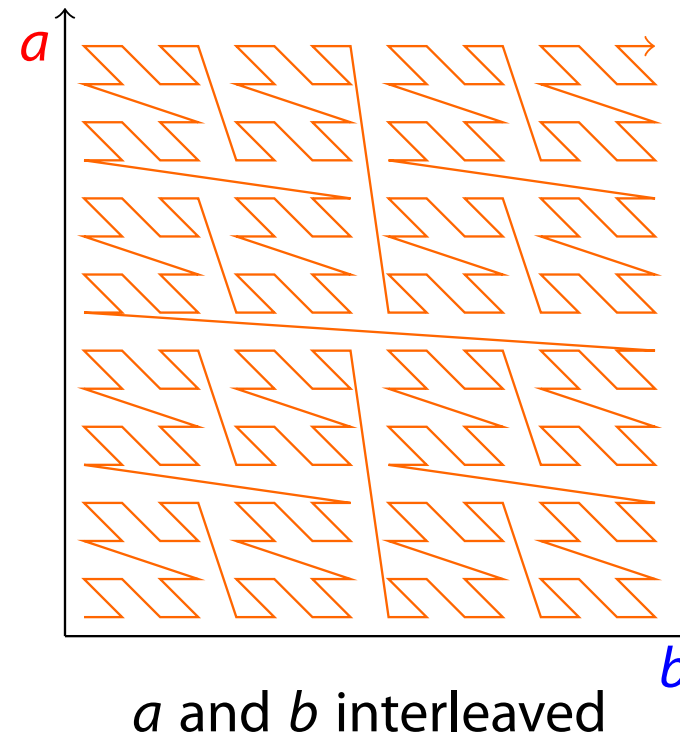
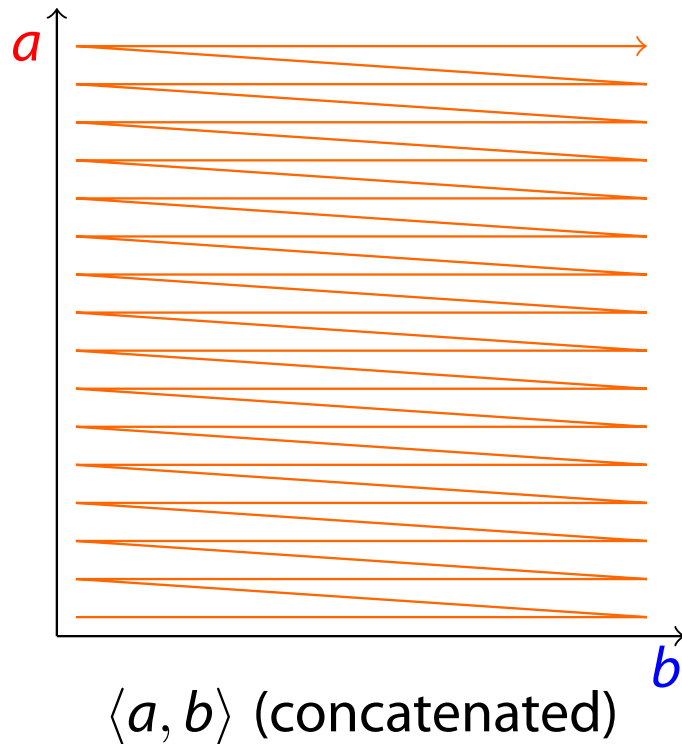
Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

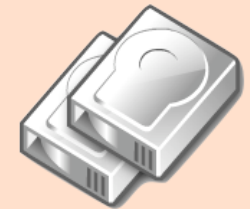
## Spaces with High Dimensionality

## Wrap-Up



- Both approaches **linearize** all coordinates in the value space according to some **order**. ↗ see also slide 8
- Bit interleaving leads to what is called the **Z-order**.
- The Z-order (largely) preserves spatial **clustering**.

# UB-Tree: B<sup>+</sup>-trees Over Z-Codes



## B<sup>+</sup>-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

## R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B<sup>+</sup>-Trees over Z-Codes

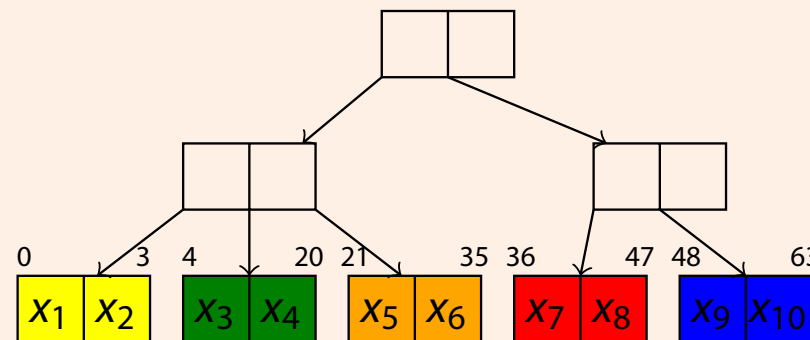
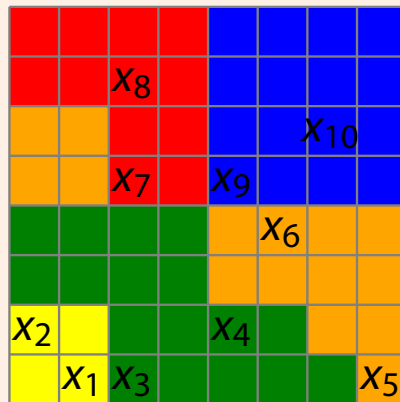
Range Queries

## Spaces with High Dimensionality

## Wrap-Up

- Use a **B<sup>+</sup>-tree** to index Z-codes of multi-dimensional data.
- Each leaf in the B<sup>+</sup>-tree describes an **interval** in the **Z-space**.
- Each interval in the Z-space describes a **region** in the multi-dimensional data space:

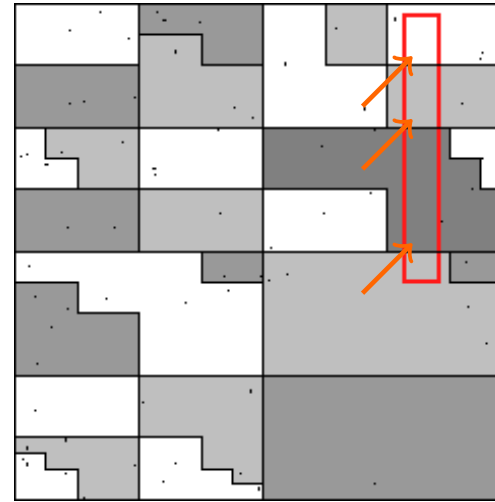
## UB-Tree: B<sup>+</sup>-tree over Z-codes



- To retrieve all data points in a query region  $Q$ , try to touch only those leaf pages that **intersect** with  $Q$ .

# UB-Tree: Range Queries

After each page processed, perform an **index re-scan** (↗) to fetch the next leaf page that intersects with  $Q$ .



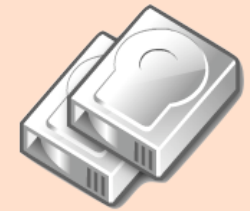
## UB-tree range search (Function `ub_range(Q)`)

```
1 cur ←  $z(Q_{\text{bottom, left}})$  ;
2 while true do
   // search B+-tree page containing cur (↗ slide 0.0)
3   page ← search(cur);
4   foreach data point p on page do
5     if p is in  $Q$  then
6       append p to result ;
7   if region in page reaches beyond  $Q_{\text{top, right}}$  then
8     break ;
   // compute next Z-address using  $Q$  and data on current
   page
9   cur ← get_next_z_address( $Q$ , page);
```

Example by Volker Markl and Rudolf Bayer, taken from <http://mistral.in.tum.de/>

## Multi-Dimensional Indexing

Torsten Grust



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

## R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

## UB-Trees

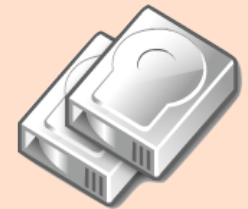
Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes

Range Queries

## Spaces with High Dimensionality

## Wrap-Up

# UB-Trees: Discussion



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search

Inserting Data

Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations

Splitting a Point Page

Splitting a Region Page

## R-Trees

Searching and Inserting

Splitting R-Tree Nodes

## UB-Trees

Bit Interleaving / Z-Ordering

B+-Trees over Z-Codes

Range Queries

## Spaces with High Dimensionality

## Wrap-Up

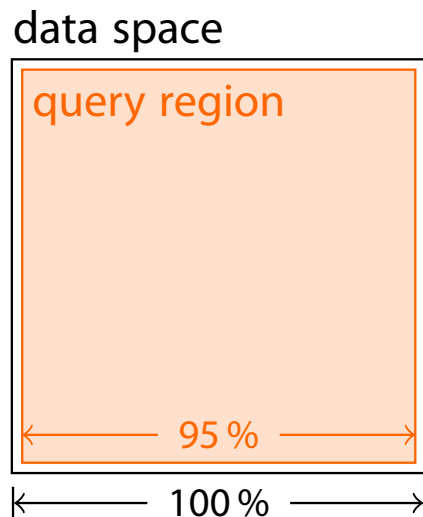
- Routine `get_next_z_address()` (return next Z-address lying in the query region) is non-trivial and depends on the shape of the Z-region.
- UB-trees are **fully dynamic**, a property inherited from the underlying B<sup>+</sup>-trees.
- The use of other **space-filling curves** to linearize the data space is discussed in the literature, too. *E.g.*, **Hilbert curves**.

 UB-trees have been commercialized in the Transbase<sup>®</sup> database system.

# Spaces with High Dimensionality

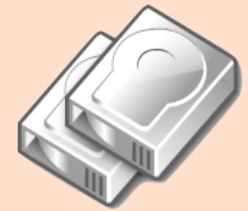
For large  $k$ , all techniques that have been discussed become **ineffective**:

- For example, for  $k = 100$ , we get  $2^{100} \approx 10^{30}$  partitions per node in a **point quad tree**. Even with billions of data points, **almost all** of these are empty.
- Consider a **really big** search region, cube-sized covering 95 % of the range along **each** dimension:



For  $k = 100$ , the probability of a point being in this region is still only  $0.95^{100} \approx 0.59\%$ .

- We experience the **curse of dimensionality** here.



**B+-trees...**

... over composite keys

**Point Quad Trees**

Point (Equality) Search  
Inserting Data  
Region Queries

**k-d Trees**

Balanced Construction

**K-D-B-Trees**

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

**R-Trees**

Searching and Inserting  
Splitting R-Tree Nodes

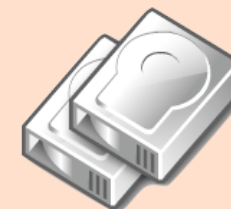
**UB-Trees**

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

**Spaces with High Dimensionality**

**Wrap-Up**

# Page Selectivity for $k$ -NN Search



## B+-trees...

... over composite keys

## Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

## k-d Trees

Balanced Construction

## K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

## R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

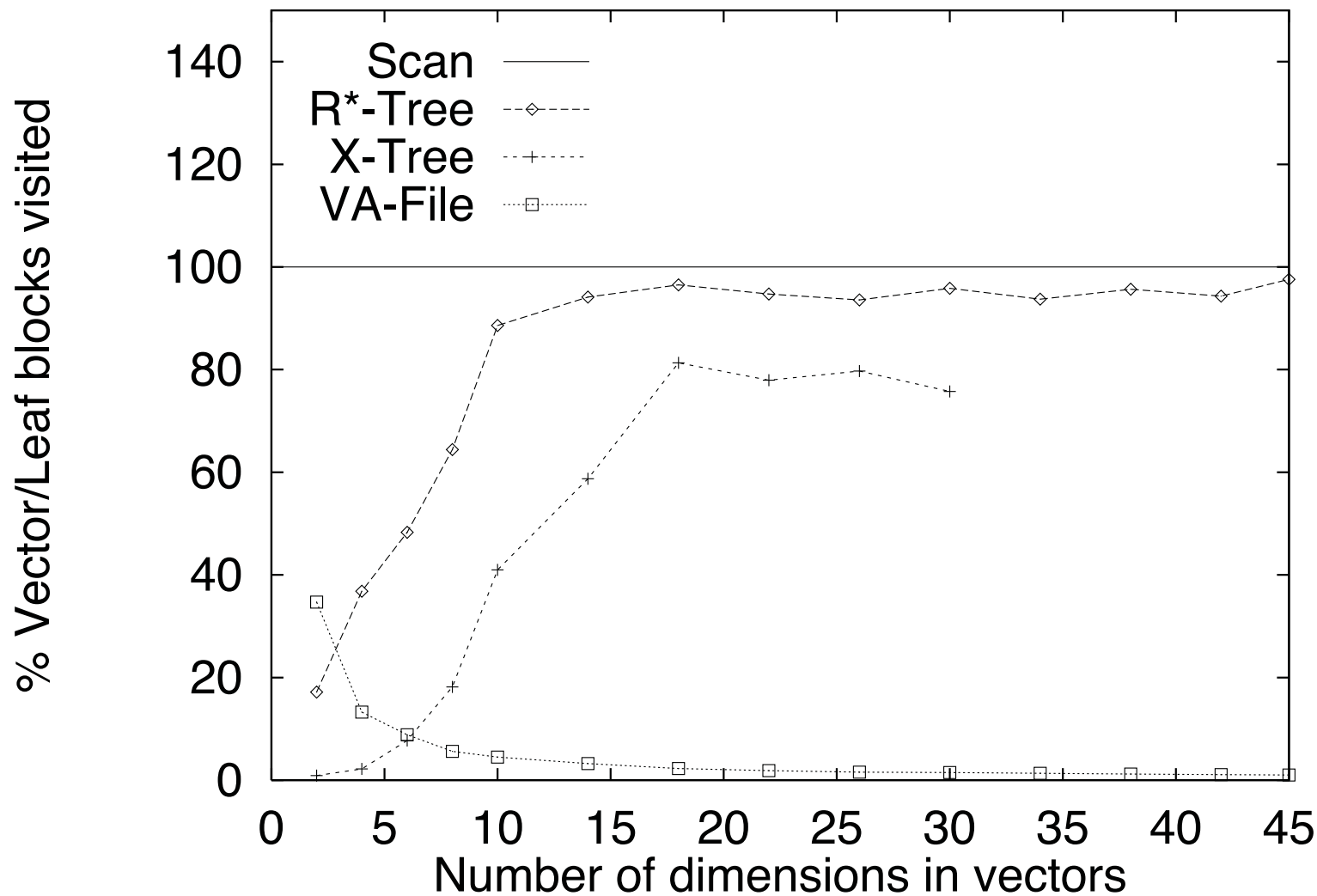
## UB-Trees

Bit Interleaving / Z-Ordering  
B+-Trees over Z-Codes  
Range Queries

## Spaces with High Dimensionality

## Wrap-Up

N=50'000, image database, k=10



Data: Stephen Bloch. What's Wrong with High-Dimensionality Search. VLDB 2008.

# Wrap-Up

## Point Quad Tree

$k$ -dimensional analogy to binary trees; main memory only.

## $k$ -d Tree, K-D-B-Tree

$k$ -d tree: partition space one dimension at a time (round-robin); K-D-B-Tree: B<sup>+</sup>-tree-like organization with pages as nodes, nodes use a  $k$ -d-like structure internally.

## R-Tree

regions within a node may overlap; fully dynamic; for point and region data.

## UB-Tree

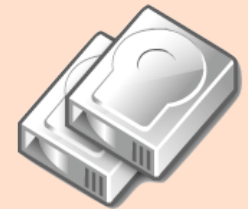
use space-filling curve (Z-order) to linearize  $k$ -dimensional data, then use B<sup>+</sup>-tree.

## Curse Of Dimensionality

most indexing structures become ineffective for large  $k$ ; fall back to sequential scanning and approximation/compression.

## Multi-Dimensional Indexing

Torsten Grust



### B<sup>+</sup>-trees...

... over composite keys

### Point Quad Trees

Point (Equality) Search  
Inserting Data  
Region Queries

### $k$ -d Trees

Balanced Construction

### K-D-B-Trees

K-D-B-Tree Operations  
Splitting a Point Page  
Splitting a Region Page

### R-Trees

Searching and Inserting  
Splitting R-Tree Nodes

### UB-Trees

Bit Interleaving / Z-Ordering  
B<sup>+</sup>-Trees over Z-Codes  
Range Queries

### Spaces with High Dimensionality

## Wrap-Up