Non-Standard-Datenbanken

Multidimensionale Indizierung

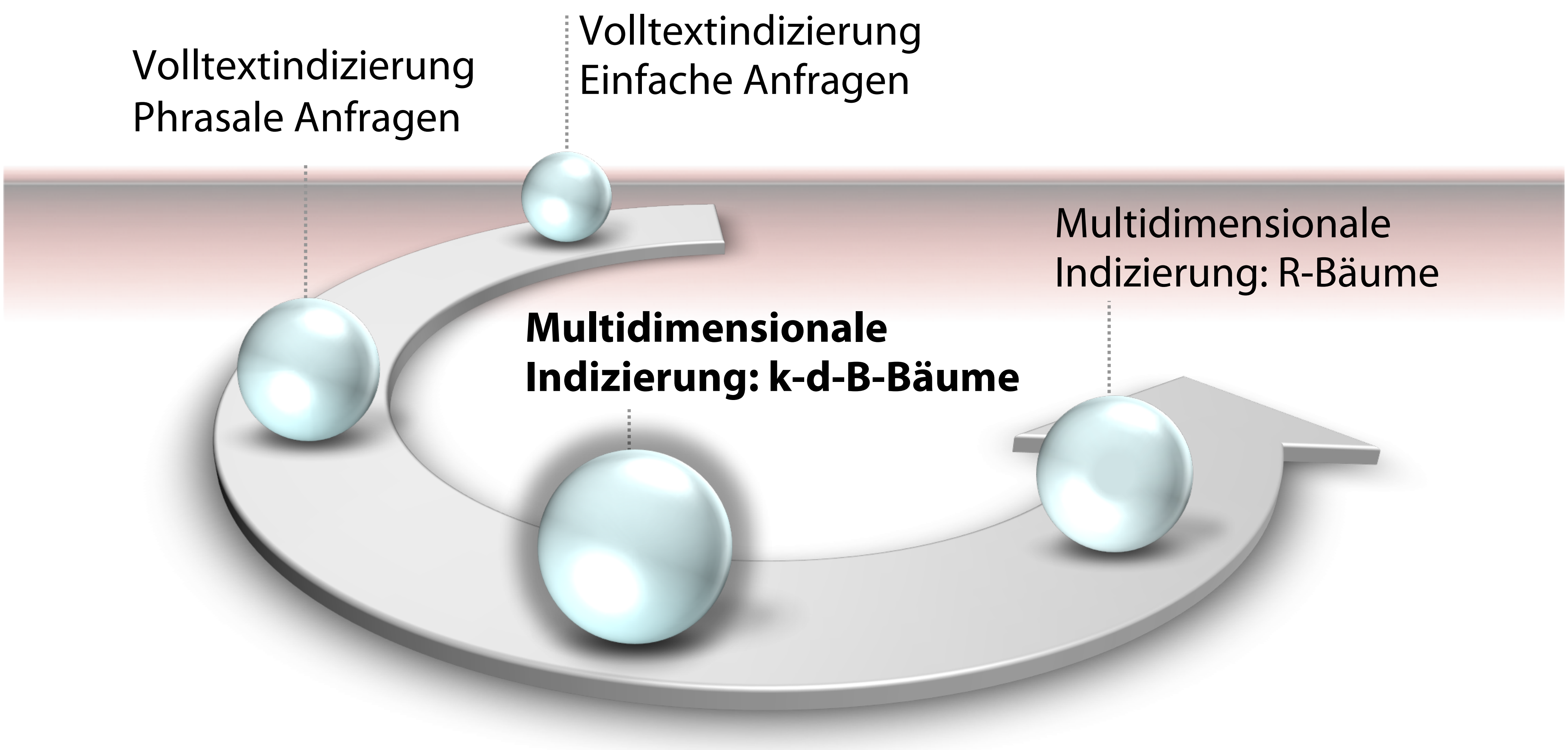
Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung



Danksagung

Die nachfolgenden Präsentationen sind motiviert durch Materialen einer Vorlesung von Jens Teubner

Insbesondere die Bilder habe ich übernommen

Ich bedanke mich für die Bereitstellung des Materials

Mehr Dimensionen ...

```
SELECT *
  FROM CUSTOMERS
 WHERE ZIPCODE BETWEEN 8000 AND 8999
   AND REVENUE BETWEEN 3500 AND 6000
```

- Anfrage beinhaltet Bereichsprädikat definiert über zwei Dimensionen, die nicht Primärschlüssel sind
- Typische Anwendungsfälle mit multidimensionalen Daten:
  - Online Analytical Processing (OLAP)
  - Geographische Informationssysteme
  - Multimedia-Systeme (Bilder- und Video-Suche)

... weitere Herausforderungen

Anfragen und Daten können Punkte oder Regionen sein

	point query	region query
region data		
point data		

Regionen-Enthaltensein oder -Schnitt

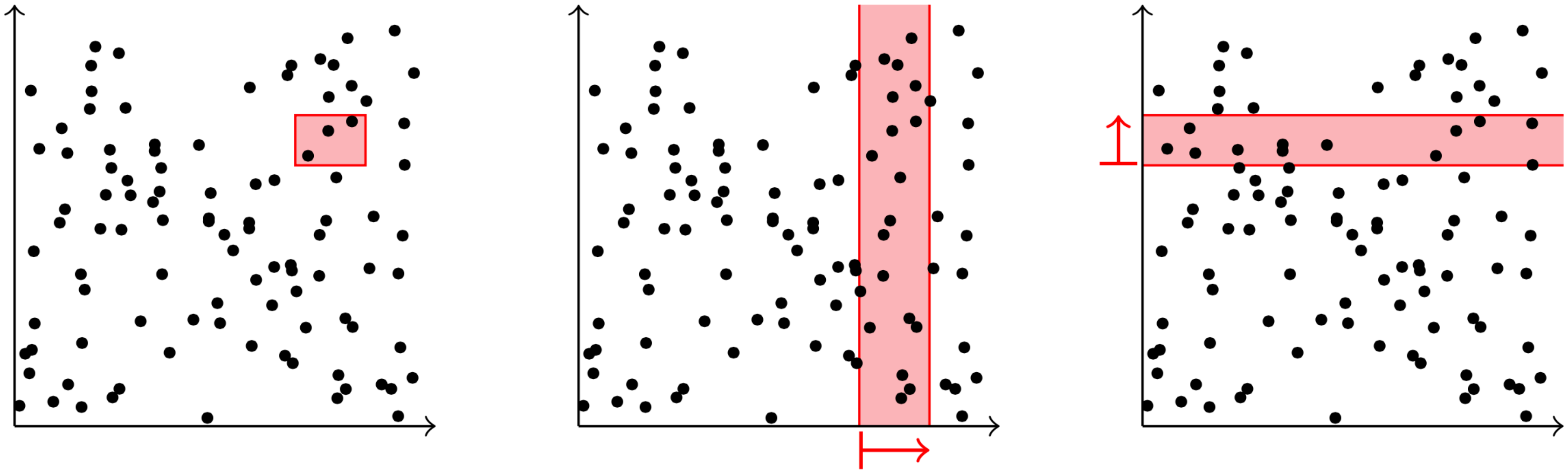
K-Nächste-Nachbarn-Suche (k-NN)

... und es gibt noch viele weitere interessante Anfragetypen für multidimensionale Daten

NB: Anfragen mit Gleichheit lassen sich in eindimensionale Anfragen zerlegen

Können wir nicht einfach B$^+$-Bäume verwenden?

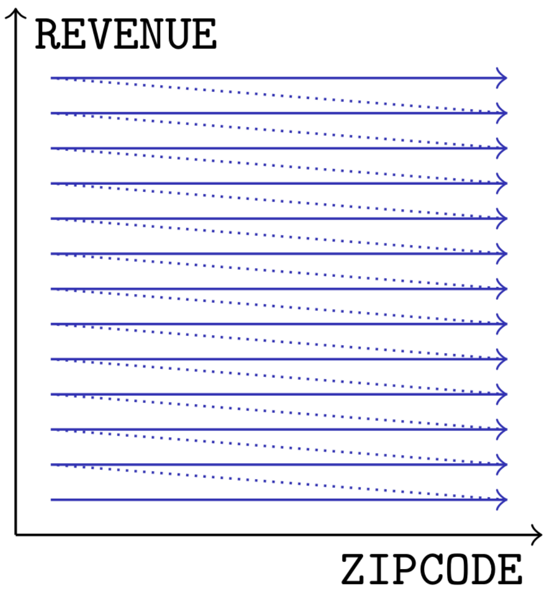
Vielleicht zwei B$^+$-Bäume für ZIPCODE und REVENUE?



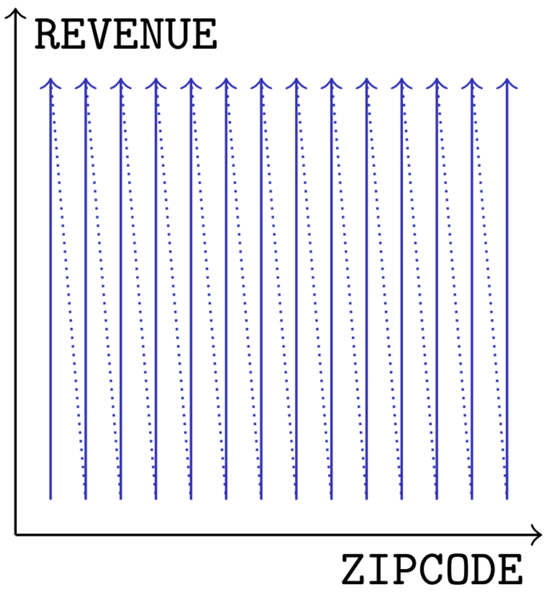
Man kann **pro Dimension** nur über **einen** Index laufen und hat viele **falsche Treffer**

Einige Datenbanken (z.B. DB2) bieten Konjunktion über Indexeinträge als Nicht-Standard-Erweiterung

Oder zusammengesetzte Schlüssel?



⟨REVENUE, ZIPCODE⟩ index



⟨ZIPCODE, REVENUE⟩ index

Gleiche Situation!

Indizes über zusammengesetzte Schlüssel sind **nicht symmetrisch**.

Das Hauptattribut dominiert die Organisation des B^+-Baums

Immerhin kann man ggf. auf dem Index arbeiten und irrelevante Einträge eliminieren

Multidimensionale Indexstrukturen

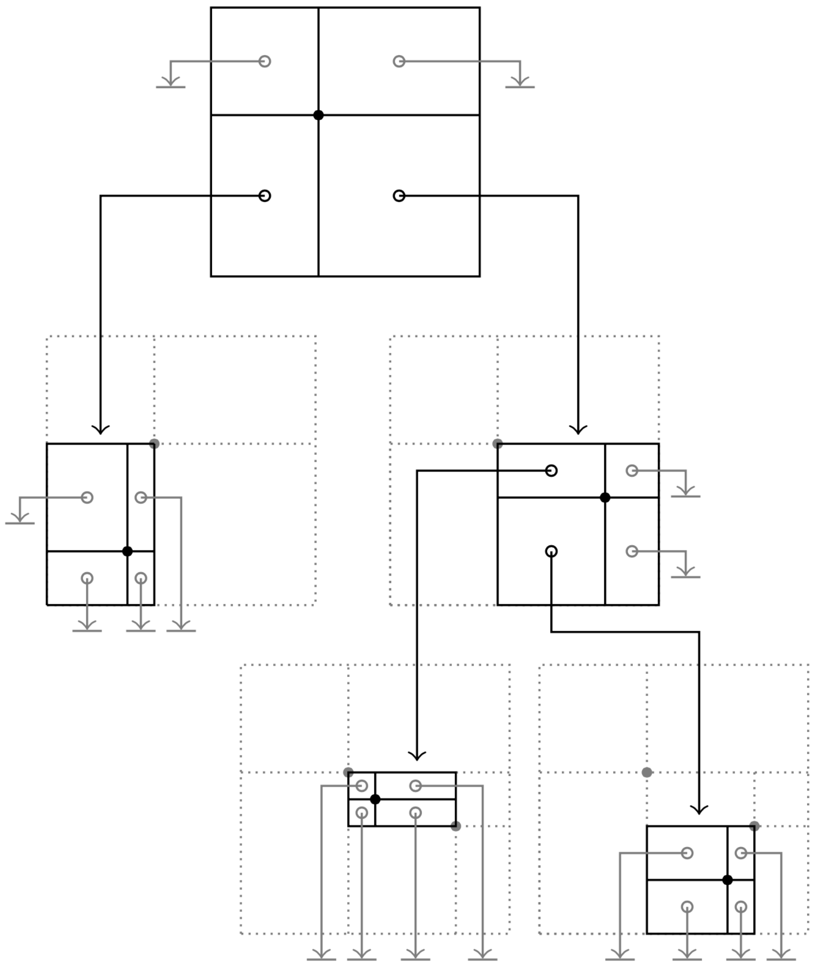
- B^+-Bäume unterstützen nur **eindimensionale** Anfragen[1]
- Wir suchen multidimensionale Indexstrukturen mit folgenden Eigenschaften
  - Symmetrie in allen Dimensionen
  - Raumorientierte Gruppierung von Daten
  - Dynamisch in Bezug auf Schreiboperationen
  - Unterstützung von häufigen Anfragen
- Erst Hauptspeicherdatenstrukturen,
  dann Erweiterungen für Sekundärspeicherbetrieb

---

[1] Am Ende betrachten wir mit UB-Bäumen noch eine elegante Kodierung, die auch bei B-Bäumen mehrdimensionale Anfragen recht gut unterstützt

„Binärer" Suchbaum

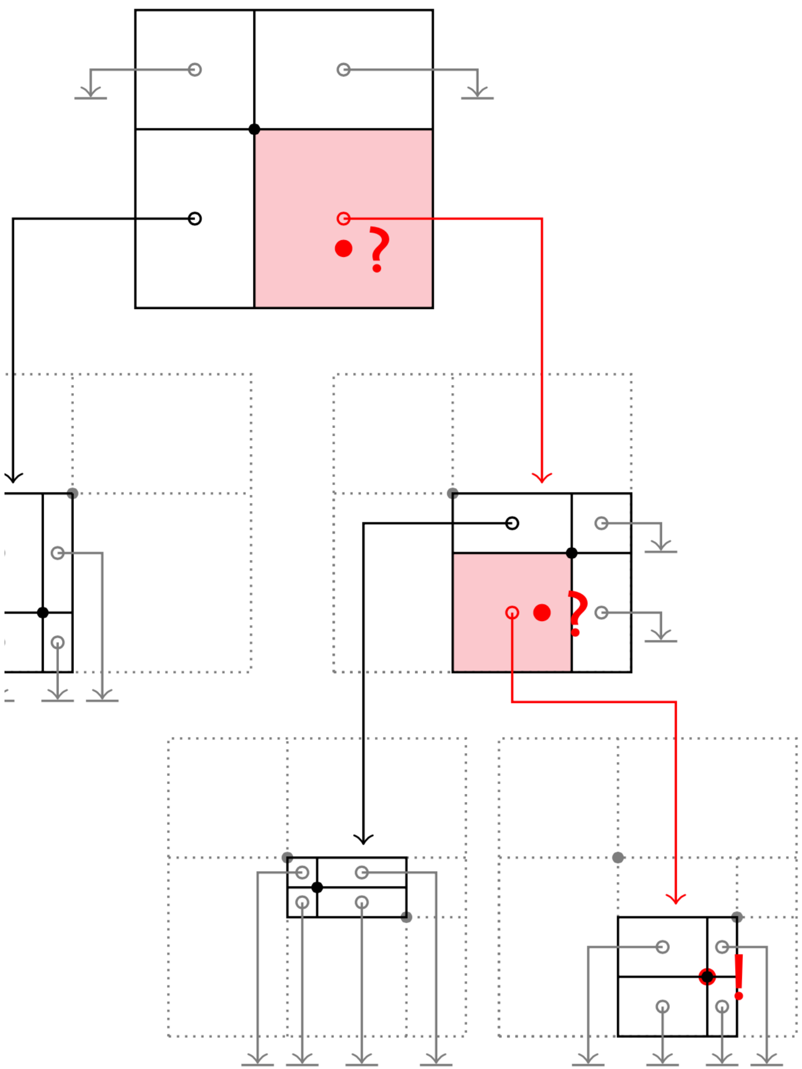
Für k Dimensionen wird aus dem Binärbaum ein 2^k-ärer Baum



- Jeder Datenpunkt **partitioniert** den Datenraum in 2^k **disjunkte** Regionen
- In einem Knoten zeigt jede Region auf einen neuen Knoten (zur Partitionierung) oder auf einen speziellen **Null**zeiger
- Eine solche Datenstruktur heißt **Punkt-Quad-Baum**

Finkel and Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Informatica, vol. 4, 1974.

Suche in einem Punkt-Quad-Baum



```
Function: p_search (q, node)
if q matches data point in node the
  | return data point ;
else
  | P ← partition containing q ;
  | if P points to null then
  |   | return not found ;
  | else
  |   | node′ ← node pointed to by
  |   | return p_search (q, node′)
```

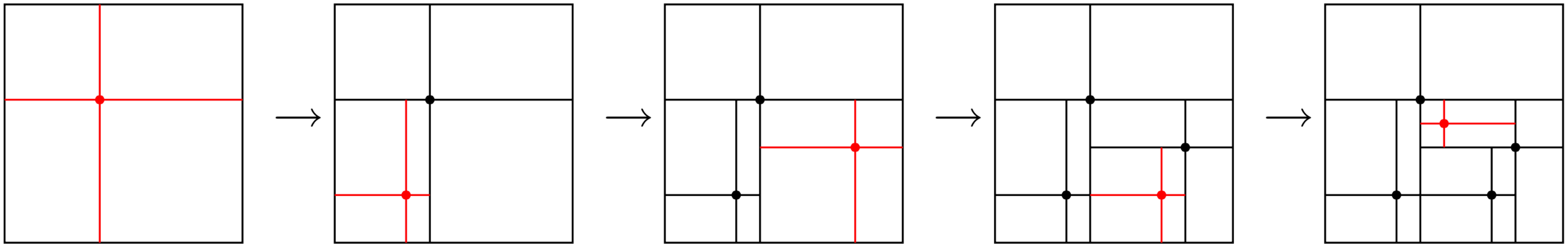
```
Function: pointsearch (q)
return p_search (q, root) ;
```

Einfügen in einen Punkt-Quad-Baum

Einfügen eines Punktes *qnew* in einen Quad-Baum funktioniert wie das Einfügen in einen Binärbaum

1. **Traversiere** den Baum, so als suche man nach *qnew* bis eine Partition *P* mit einem **Null**zeiger erreicht ist
2. Erzeuge **neuen Knoten** *n'*, der die Region *P* aufspannt und durch *qnew* partitioniert wird (mit **Null** für alle Subpartitionen)
3. Lasse *P* auf *n'* zeigen

Leider bleibt der Baum **nicht immer balanciert**



Bereichsanfragen

Um eine Bereichsanfrage[2] zu evaluieren, müssen ggf. mehrere Regionen verfolgt werden

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

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

[2] Wir betrachten rechteckige Region, ggf. sind Umgebungsboxen zu betrachten und die Antworten nachzuarbeiten

Punkt-Quad-Bäume – Diskussion

Punkt-Quad-Bäume

✓ sind **symmetrisch** in Bezug auf alle Dimensionen

✓ und unterstützen **Punkt-** und **Regionen-**Anfragen

Aber

– die Form hängt von der **Einfügereihenfolge** ab
(im **schlimmsten** Fall entsteht eine **verkettete Liste**)

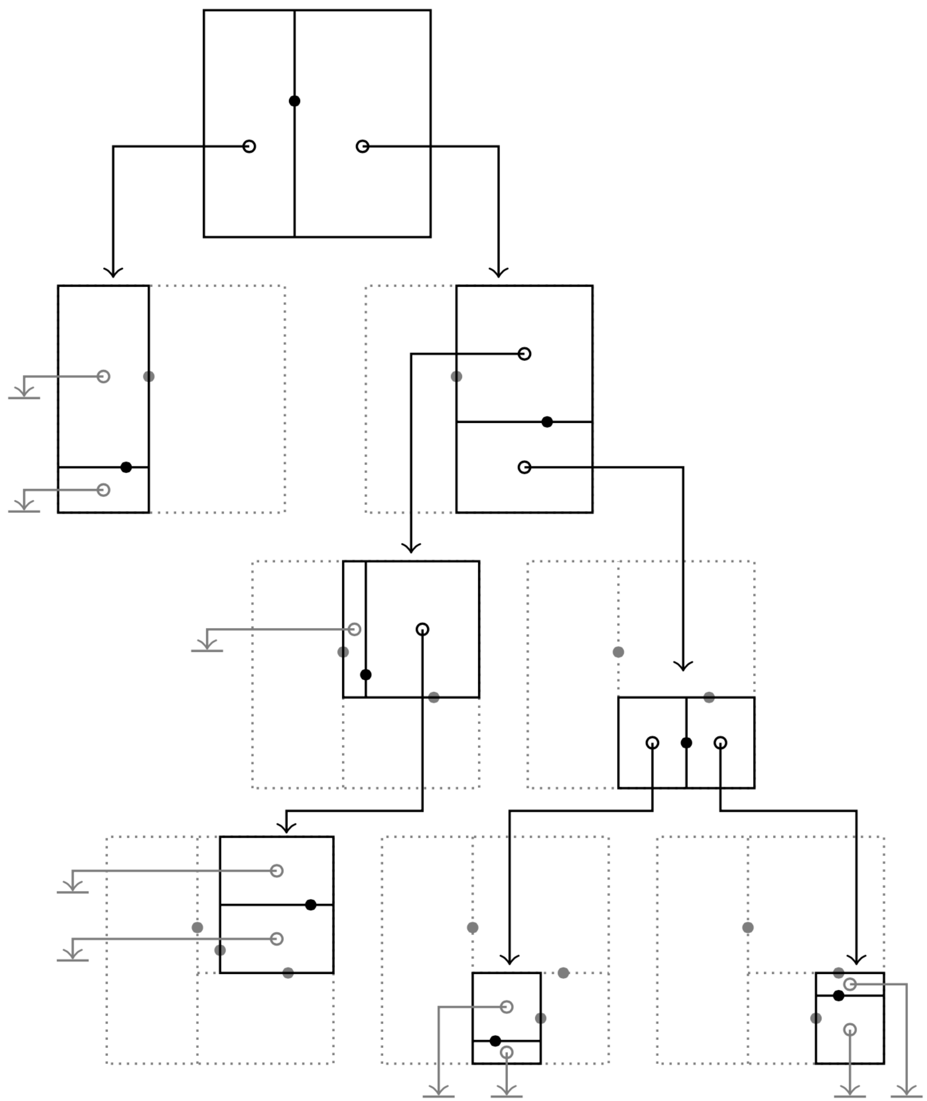
– **Null**zeiger sind speicher**ineffizient** (ins. bei großem k)

Und

o nur **Punktdaten** können gespeichert werden

NB: Punkt-Quad-Bäume sind für Hauptspeicher gedacht

k-d-Bäume



- Indiziere *k*-dimensionale Daten, aber halte den Baum binär
- Verwende für jede Baumebene *l* eine andere Dimension d_l als Diskriminator zur Partitionierung
  - Schema: Round-Robin
- Man erhält einen *k*-**d-Baum**

Bentley. Multidimensional Binary Search Trees Used for Associative Searching. Comm. ACM, vol. 18, no. 9, Sept. **1975**.

k-d-Bäume

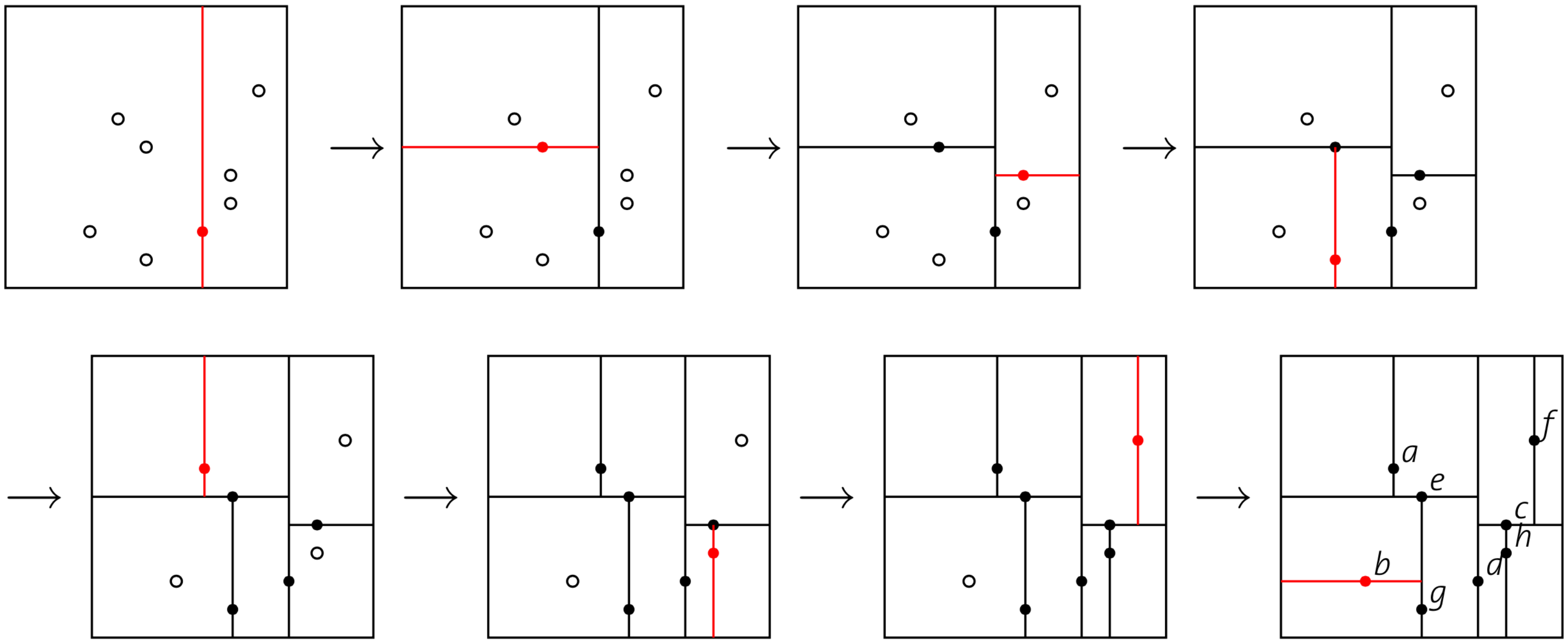
k-d-Bäume übernehmen die positiven Eigenschaften von Punkt-Quad-Bäumen, sind aber **speichereffizienter**

Für eine gegebene Punktmenge kann ein **balancierter** *k*-d-Baum konstruiert werden[3]

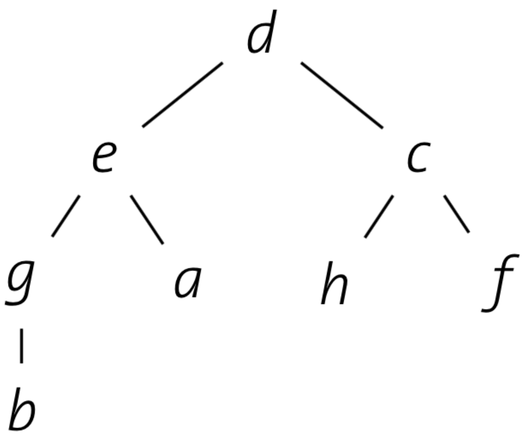
Function: `kdtree` (*pointset*, *level*)
if *pointset* is empty **then**
    return null ;
else
    $p \leftarrow$ median from *pointset* (along d_{level}) ;
    $points_{\text{left}} \leftarrow \{v \in pointset \text{ where } v_{d_{level}} < p_{d_{level}}\}$;
    $points_{\text{right}} \leftarrow \{v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

[3] v_i: Koordinate i von Punkt v

Balancierte *k*-d-Baum-Konstruktion

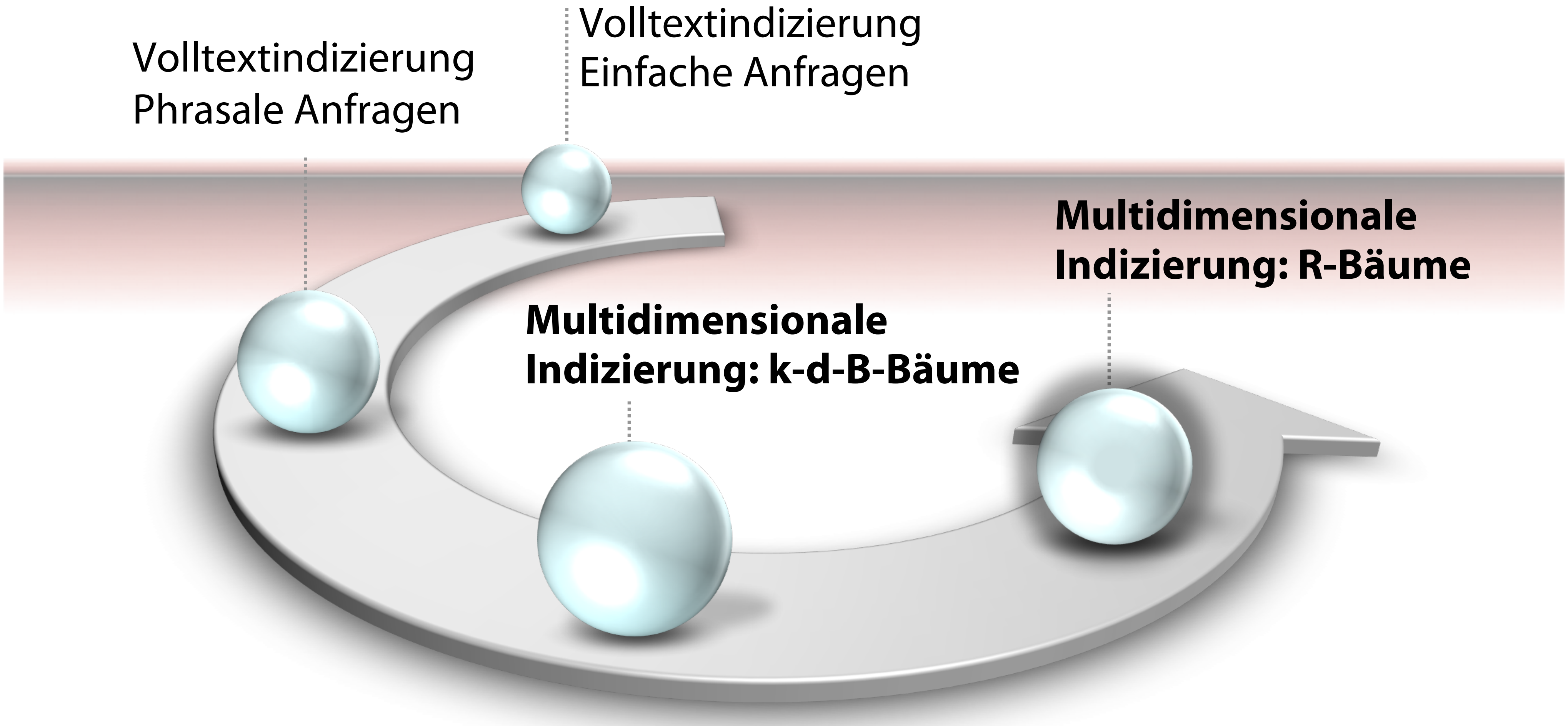


Ergebnis



Non-Standard-Datenbanken

Multidimensionale Indizierung

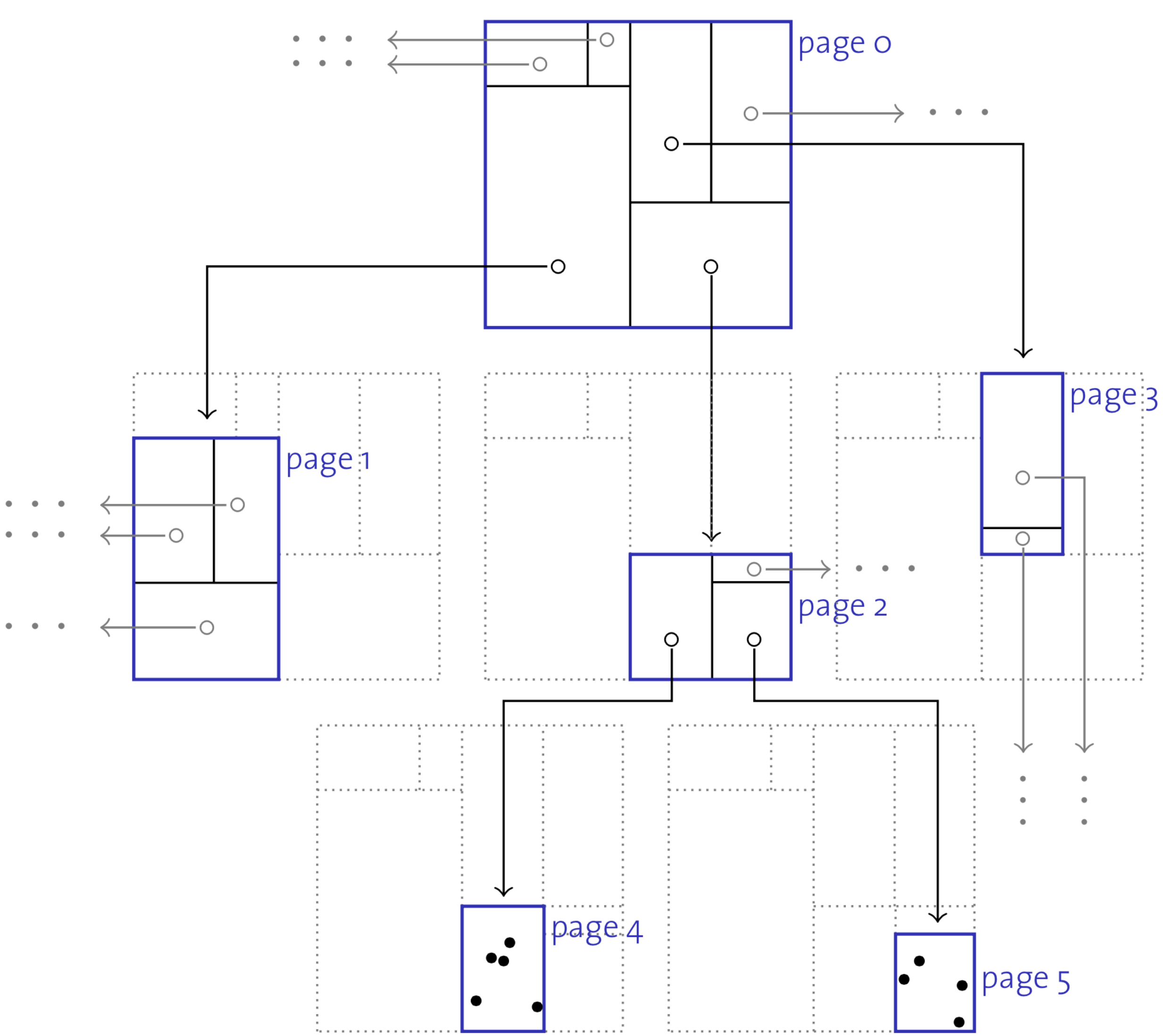


k-d-B-Bäume

- k-d-Bäume auf Sekundärspeichern
- Verwendung von Seiten als organisatorische Einheiten
  - Jeder Knoten in einem k-d-B-Baum füllt eine Seite
- k-d-Baum-Layout für jede Seite

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

k-d-B-Bäume: Zentrale Idee



Regionenseiten

- enthalten Einträge <region, pageID>
- keine **Null**zeiger
- bilden **balancierten** Baum
- alle Regionen **disjunkt** und **rechteckig**

Punktseiten

- enthalten Einträge <point, rid>
- → Blattknoten B^+-Baum

Operationen auf k-d-B-Bäumen

- **Suche** in einem k-d-B-Baum läuft wie folgt:
  - Auf jeder Seite bestimme die Region R_i, die Anfragepunkt q enthält (oder sich mit der Anfrageregion Q schneidet)
  - Für jedes solche R_i bestimme die Seite und wende Suche rekursiv an
  - Auf Punktseiten hole jeden Punkt p_i, der auf Anfrage passt und gebe ihn zurück

Operationen auf *k*-d-B-Bäumen

- Beim **Einfügen** wird der Baum **balanciert** wie beim **B^+-Baum**
  - Füge Eintrag <region, pageID> (<point, rid>) in eine Regionenseite (Punktseite) ein,
    sofern **genügend Platz** vorhanden
  - **Sonst: Splitte** Seite auf

Aufsplittung einer Punktseite

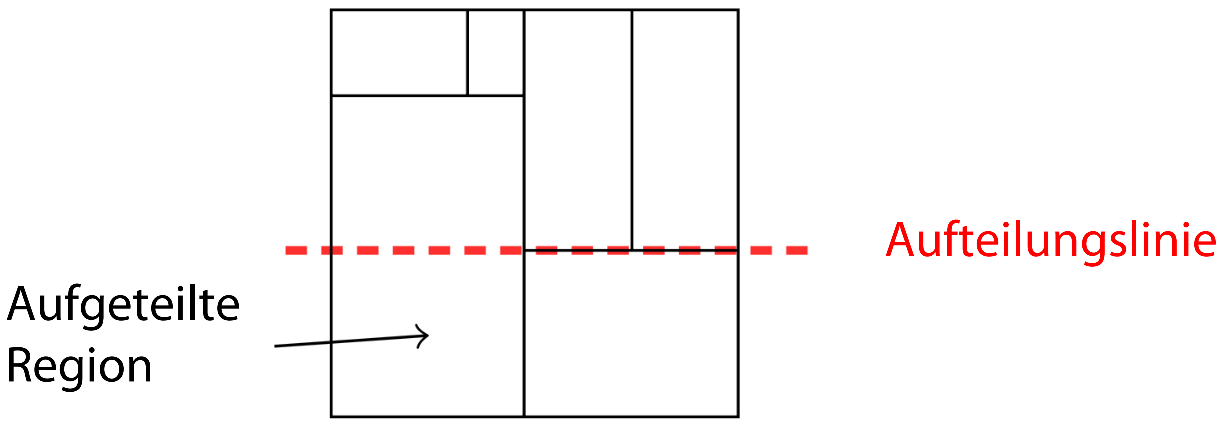
Aufteilung einer Seite *p*

1. **Wähle Dimension** *i* und eine *i*-Koordinate x_i entlang derer die Aufteilung erfolgen soll, so dass die Teilung zwei nicht übervolle Seiten erzeugt
2. **Schiebe** Datenpunkte entsprechend auf neue Seiten p_{links} oder p_{rechts} sofern $p_i < x_i$ oder $p_i \geq x_i$
3. **Ersetze** *<region, p>* auf der **Elternseite** durch *<linke-region,* p_{links}*>* und *<rechte-region,* p_{rechts}*>*

Der 3. Schritt kann zu einem **Überlauf** der Elternseite führen und damit zu einem **Aufspalten** einer **Regionenseite**

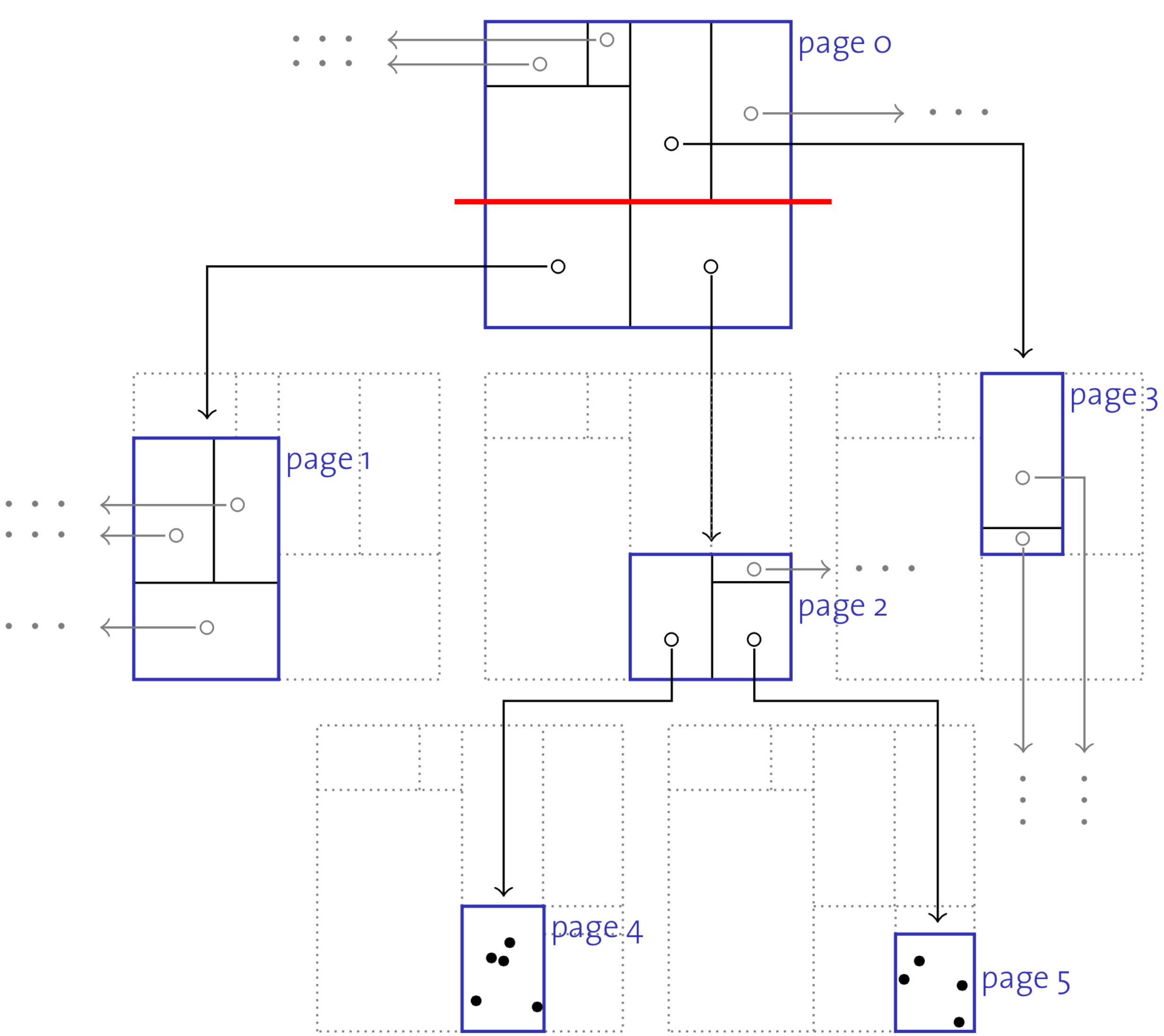
Aufspaltung einer Regionenseite

- Aufspaltung einer **Punktseite** und Verschiebung der Datenpunkte ist recht **einfach**
- Im Falle einer **Aufspaltung** können einige **Regionen auf beiden** Seite der Aufteilungslinie liegen



- Diese Regionen müssen **aufgeteilt** werden
- Mögliche Folge: **Rekursives Aufteilen nach unten**

Beispiel noch einmal



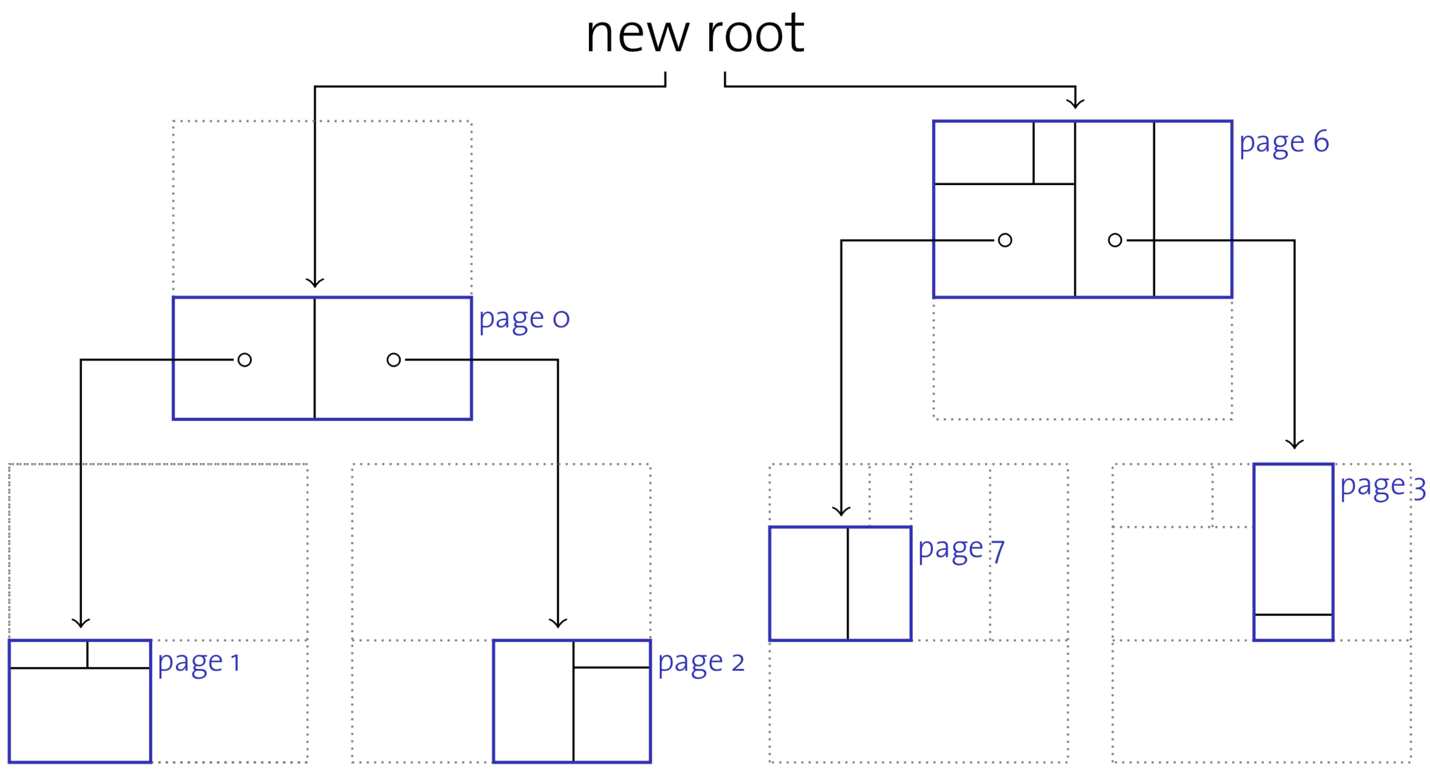
Regionenseiten

- enthalten Einträge <region, pageID>
- keine **Null**zeiger
- bilden **balancierten** Baum
- alle Regionen **disjunkt** und **rechteckig**

Punktseiten

- enthalten Einträge <point, rid>
- → Blattknoten B^+-Baum

Beispiel: Aufspaltung von Seite 0



Wurzelseite 0 → Seiten 0 und 6 (neue Wurzel erzeugen)

Regionenseite 1 → Seiten 1 und 7

(Punktseiten nicht gezeigt)

k-d-B-Bäume – Diskussion

- ✓ **Symmetrie** in Bezug auf alle Dimensionen
- ✓ **Räumliche** Gruppierung von Daten in **seitenorientierter** Weise
- ✓ **Dynamisch** in Bezug auf **Schreib**operationen
- ✓ Unterstützung von **Punkt** und **Regionen**anfragen

Aber:

- ○ **Keine Regionendaten**
- ○ **Löschoperationen nicht** (dynamisch) **unterstützt**

Datenraum wird partitioniert, so dass

- jede Region **rechteckig** ist und
- sich Regionen **nicht überlappen**

R-Bäume

Regionen können sich in dieser Struktur überlappen

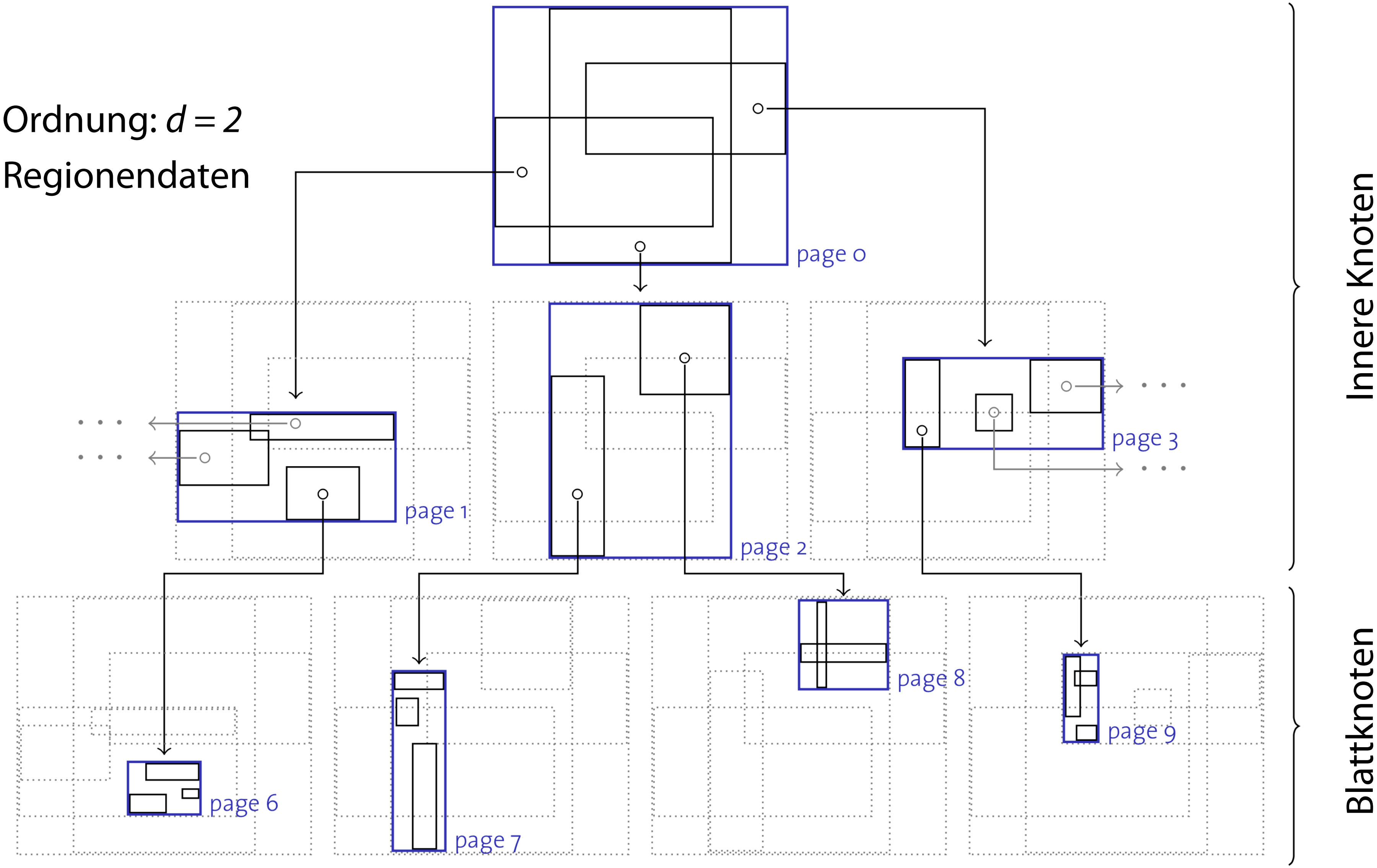
- Innere Knoten enthalten <region, pageID> Einträge, Blattknoten enthalten Einträge der Form <region, rid>, wobei region das **minimale Umgebungsrechteck** der Datenelemente, die über den Zeigern erreichbar sind
- Jeder Knoten enthält zwischen d und 2d Elemente (→ B^+-Baum). Die Wurzel kann weniger als d Elemente enthalten, sofern weniger als d Elemente im Baum sind.
- **Einfüge**- und **Löschalgorithmen** halten den R-Baum **balanciert**

Es können sowohl **Punkte** als auch **Regionen gespeichert** werden

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

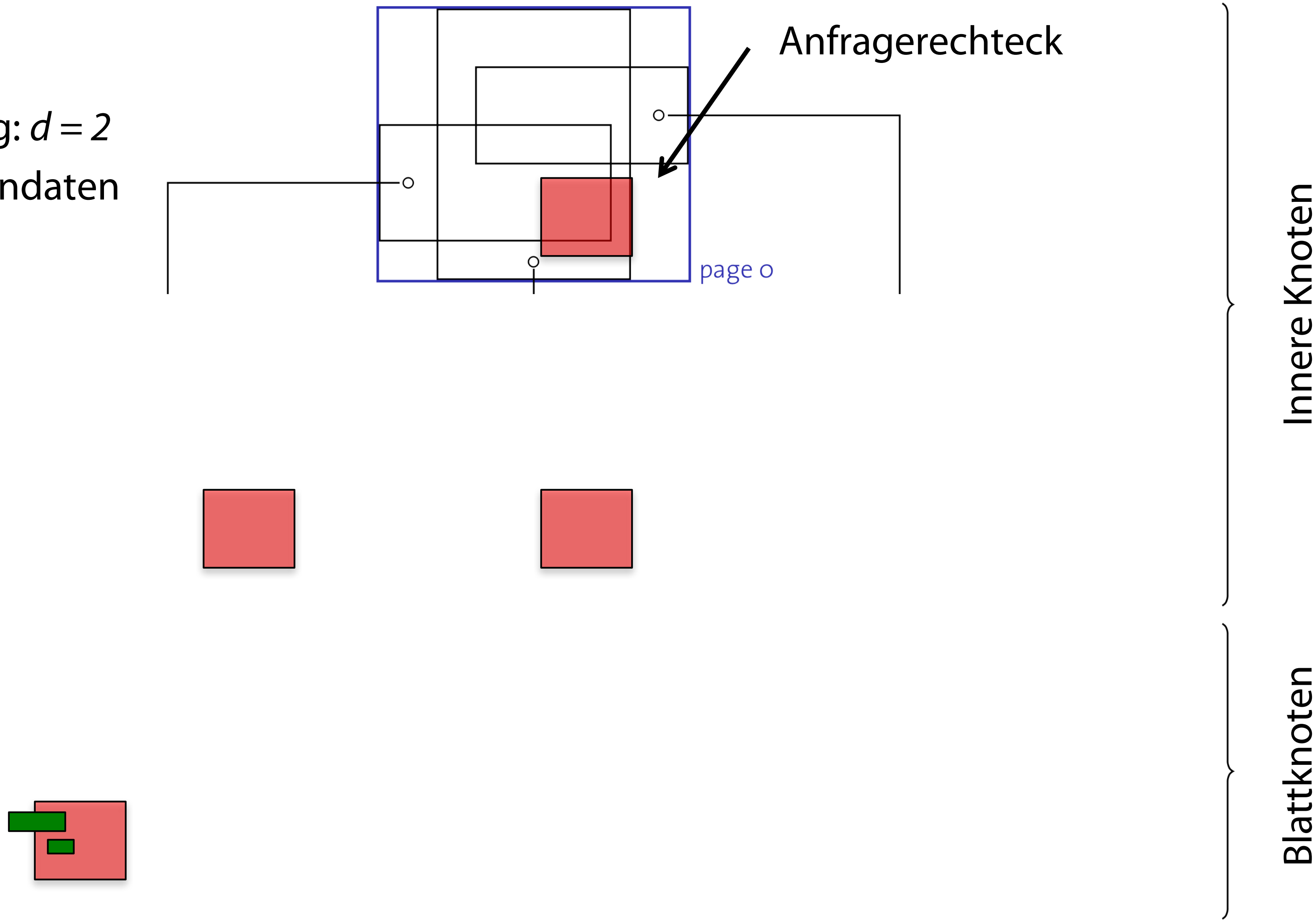
R-Baum: Beispiel

- Ordnung: $d = 2$
- Regionendaten



R-Baum: Regionenanfrage (Schnitt)

- Ordnung: $d = 2$
- Regionendaten



R-Baum: Suchen und Einfügen

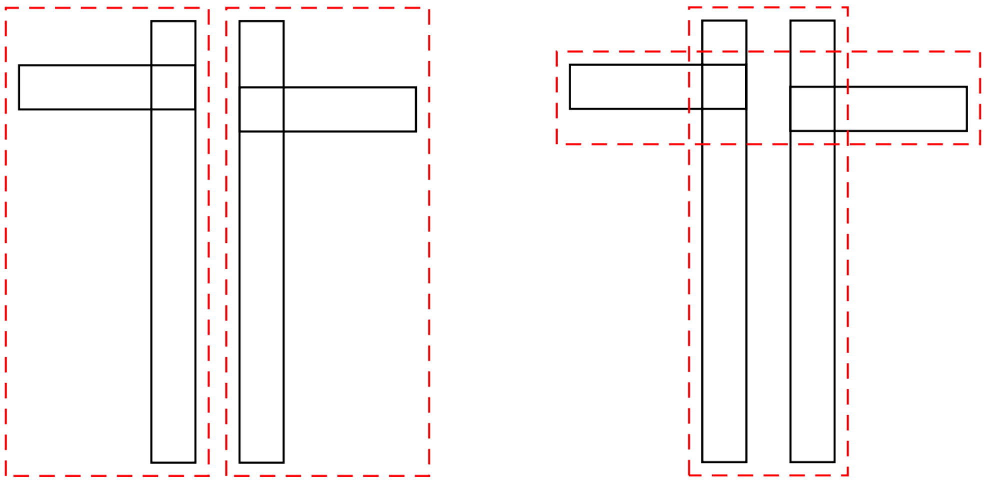
Während der **Suche** müssen ggf. mehrere Kinder betrachtet werden (gilt für Punkt- **und** Regionenanfragen)

Einfügen erfolgt wie in einem B^+-Baum

1. **Wähle** richtigen Blattknoten *n* für die Einfügung (versuche entstehende neue Rechtecke zu minimieren)
2. Falls *n* **voll** ist, **spalte** ihn auf (wir haben *n* und *n'*) und verteile alte Einträge auf *n* und *n'*
   – Aufspaltungen können nach oben propagieren und erreichen ggf. die Wurzel
3. Nach dem Einfügung müssen Regionen im Vorgänger-knoten angepasst werden (Umgebungsrechtecke)

Aufspaltung von Knoten im R-Baum

Mehrere Möglichkeiten



Schlechte Aufspaltung   Gute Aufspaltung

Heuristik: Minimiere überdeckte Fläche

Bestimmung der besten Aufteilung i.a. zu kombinatorisch

Das originale Guttman-Papier stellt Approximation vor

Verbessert in Nachfolgepapieren (R^*-Baum, ...)

Löschoperationen

R-Baum-**Invarianten** bei jeder Operation **beibehalten**

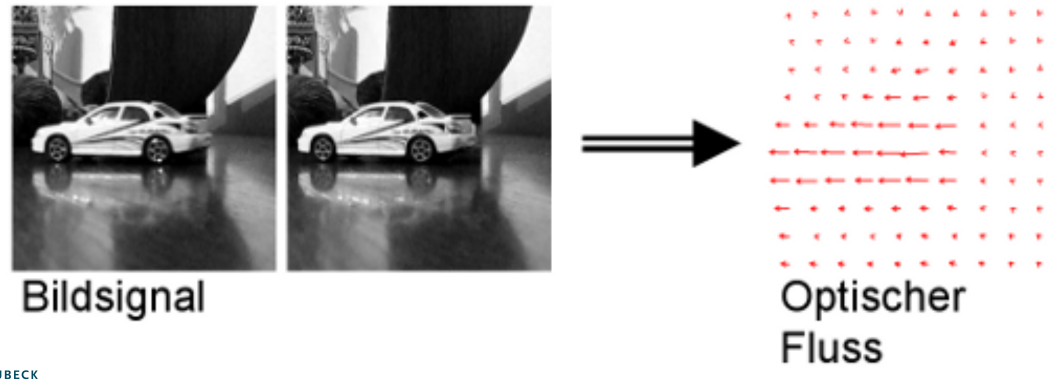
1. Falls ein Knoten *n* zu leer wird (weniger als *d* Einträge nach einer Löschoperation) wird der Knoten gelöscht
2. Und die Einträge werden auf andere Knoten verteilt

Der erste Schritt kann zur Löschung des Elternknoten führen

- Löschen ist eine aufwändige Operation in R-Bäumen

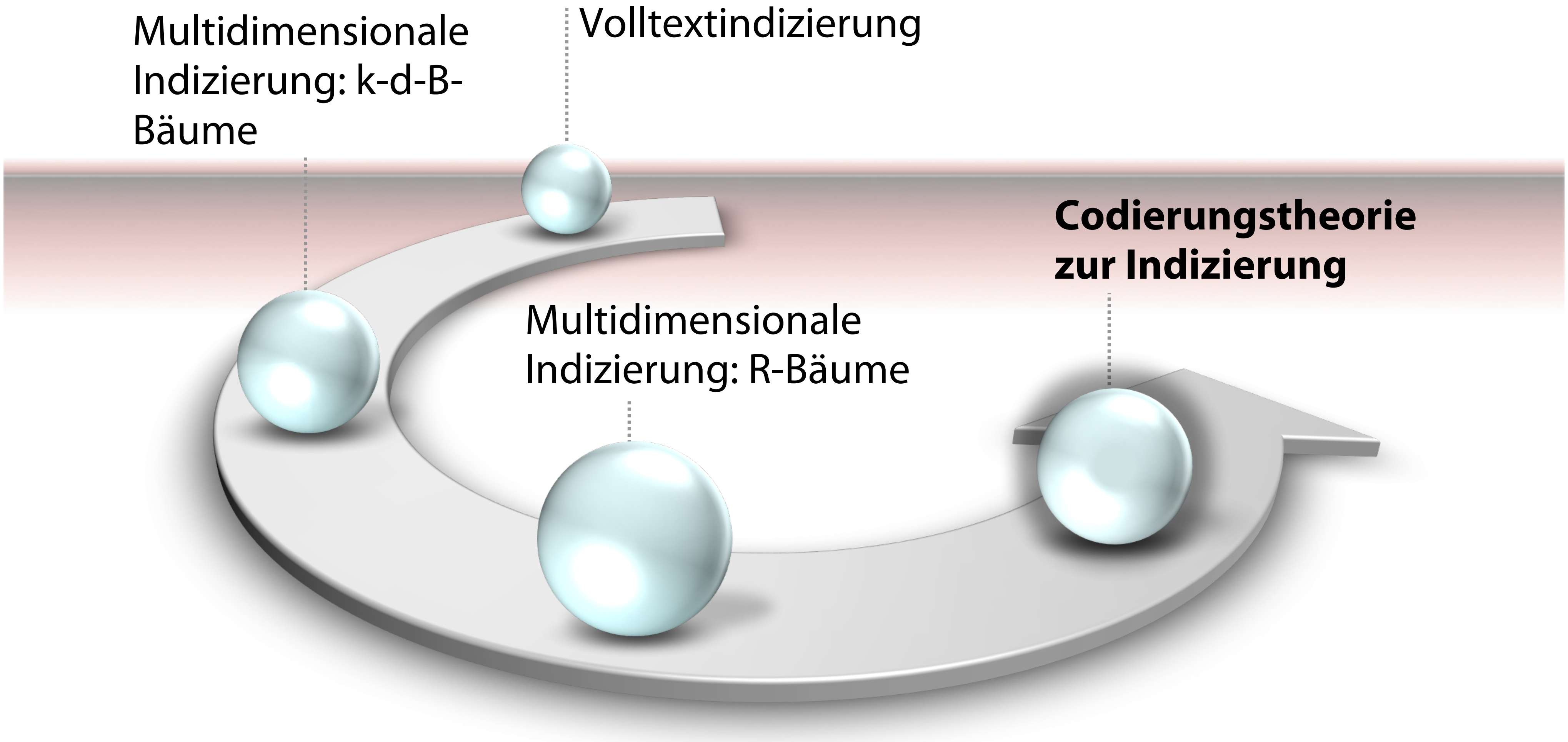
Anwendung von R-Bäumen

- Geographische Informationssysteme
  - Meist ein Zusatzmodul bei Datenbanksystemen
  - Integriert in PostgreSQL
- Multimedia-Datenbanksysteme
  - Extraktion von Merkmalen aus Bildern
    (→ hochdimensionaler Merkmalsvektor)
  - Topologische Beziehungen in räumlichen Anfragen
  - Mehrdimensional z.B. auch bei Trajektorien



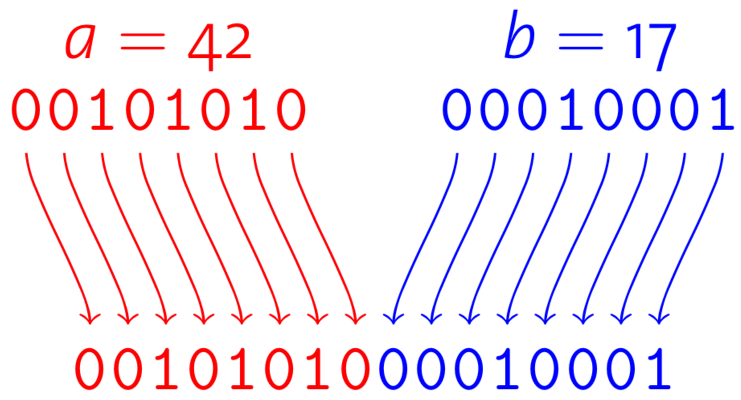
Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung

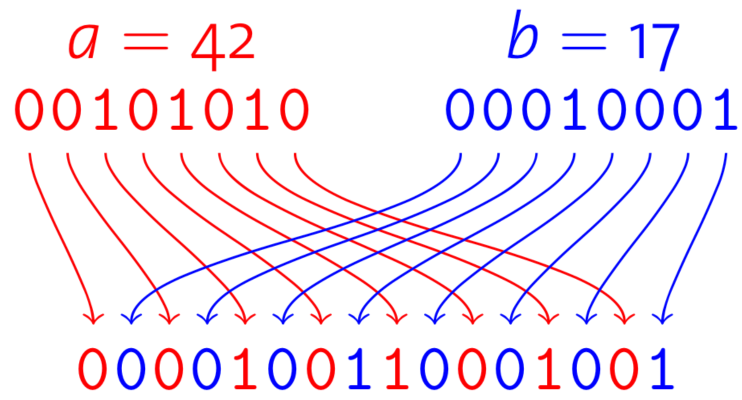


Bit-Verschränkung

- Zusammengesetzte Schlüssel <*a*, *b*> wegen Asymmetrie nicht direkt hilfreich für den effizienten Zugriff
- Was passiert, wenn die Bits von a und b verschränkt werden (und damit „symmetrischer“)?

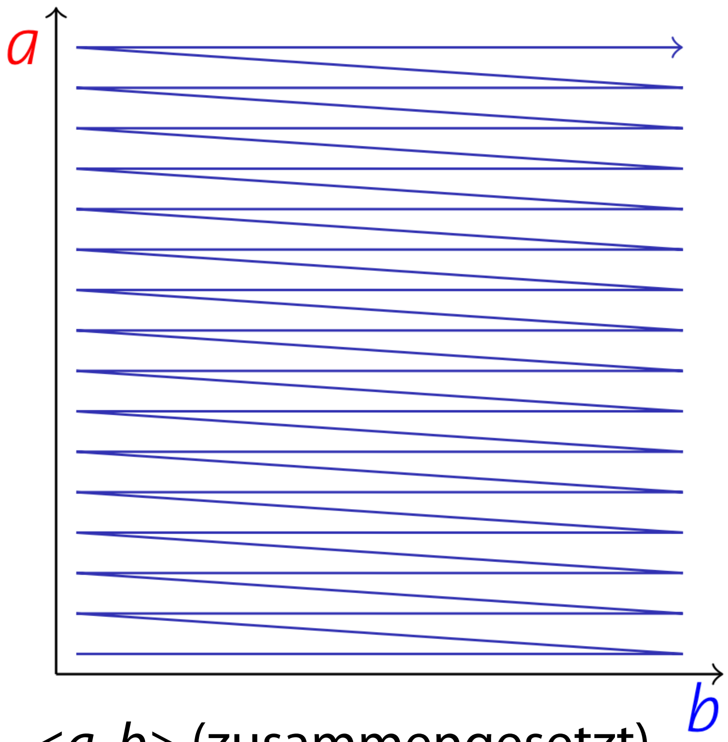


<*a*, *b*> (zusammengesetzt)

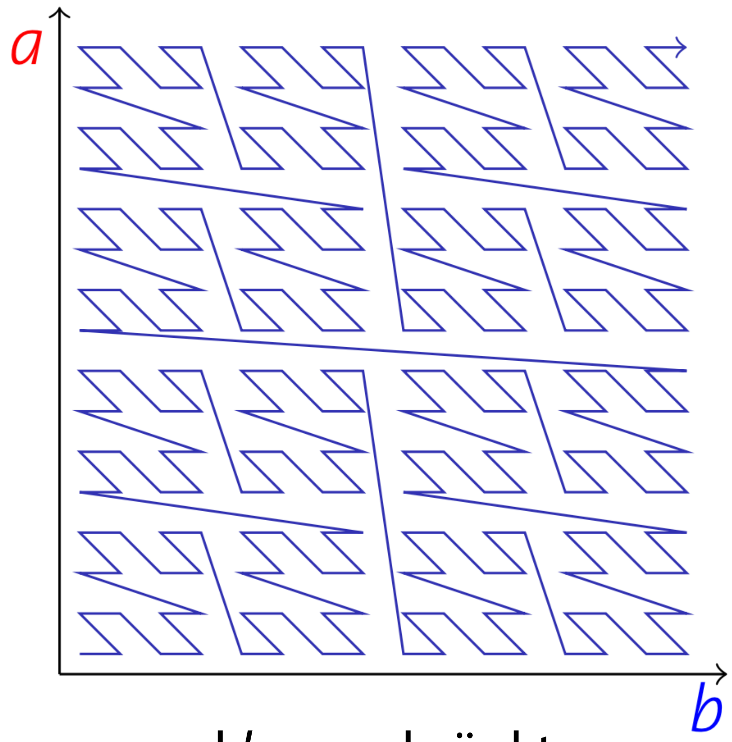


a und *b* verschränkt

Z-Ordnung



<*a*, *b*> (zusammengesetzt)

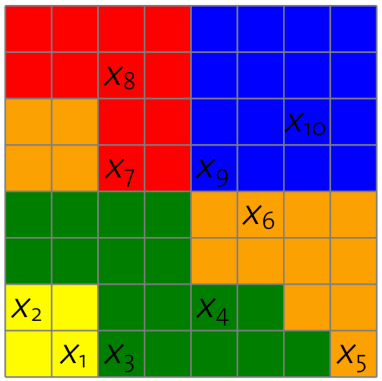


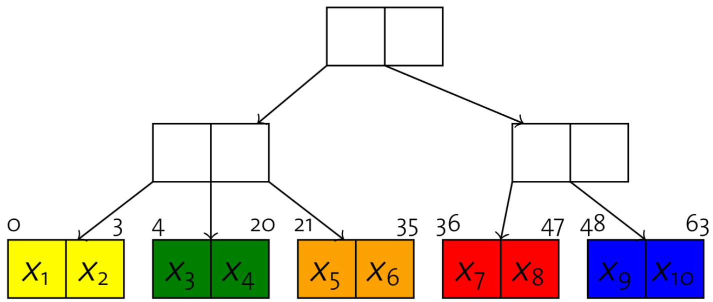
a und *b* verschränkt

- Beide Ansätze **linearisieren** die Koordinaten im Wertebereich nach einer festgelegten Ordnung
- Bitverschränkung erzeugt die **Z-Ordnung**
- Durch die Z-Ordnung erfolgt **räumliche Gruppierung**

B^+-Bäume über Z-Ordnungen

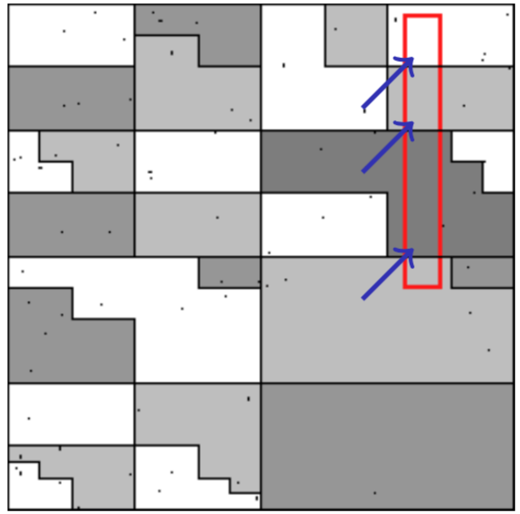
- Verwendung eines **B^+-Baumes** um Z-Kodes des multidimensionalen Raums zu indizieren
- Blatt im B^+-Baum beschreibt **Intervall** im **Z-Raum**
- Jedes dieser Intervalle beschreibt eine **Region** im multidimensionalen Datenraum





- Um alle Datenpunkte für eine Anfrage *Q* zu finden, sollen nur solche Blattseiten betrachtet werden, die Regionen enthalten, die sich mit *Q* **schneiden**

UB-Baum-Bereichsanfragen

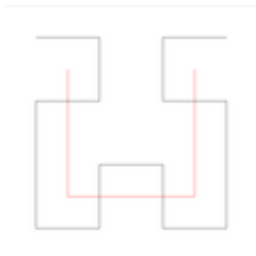


Nach jeder verarbeiteten Seite erfolgt Index-Rescan um neue Seite zu finden, die sich mit Anfragerechteck Q schneidet

```
Function: ub_range (Q)
cur ← z(Q_bottom,left) ;
while true do
    // search B+-tree page containing cur
    page ← search (cur);
    foreach data point p on page do
        if p is in Q then
            append p to result ;
    if region in page reaches beyond Q_top,right then
        break ;
    // compute next Z-address using Q and data on current page
    cur ← get_next_z_address (Q, page);
```

UB-Bäume – Diskussion

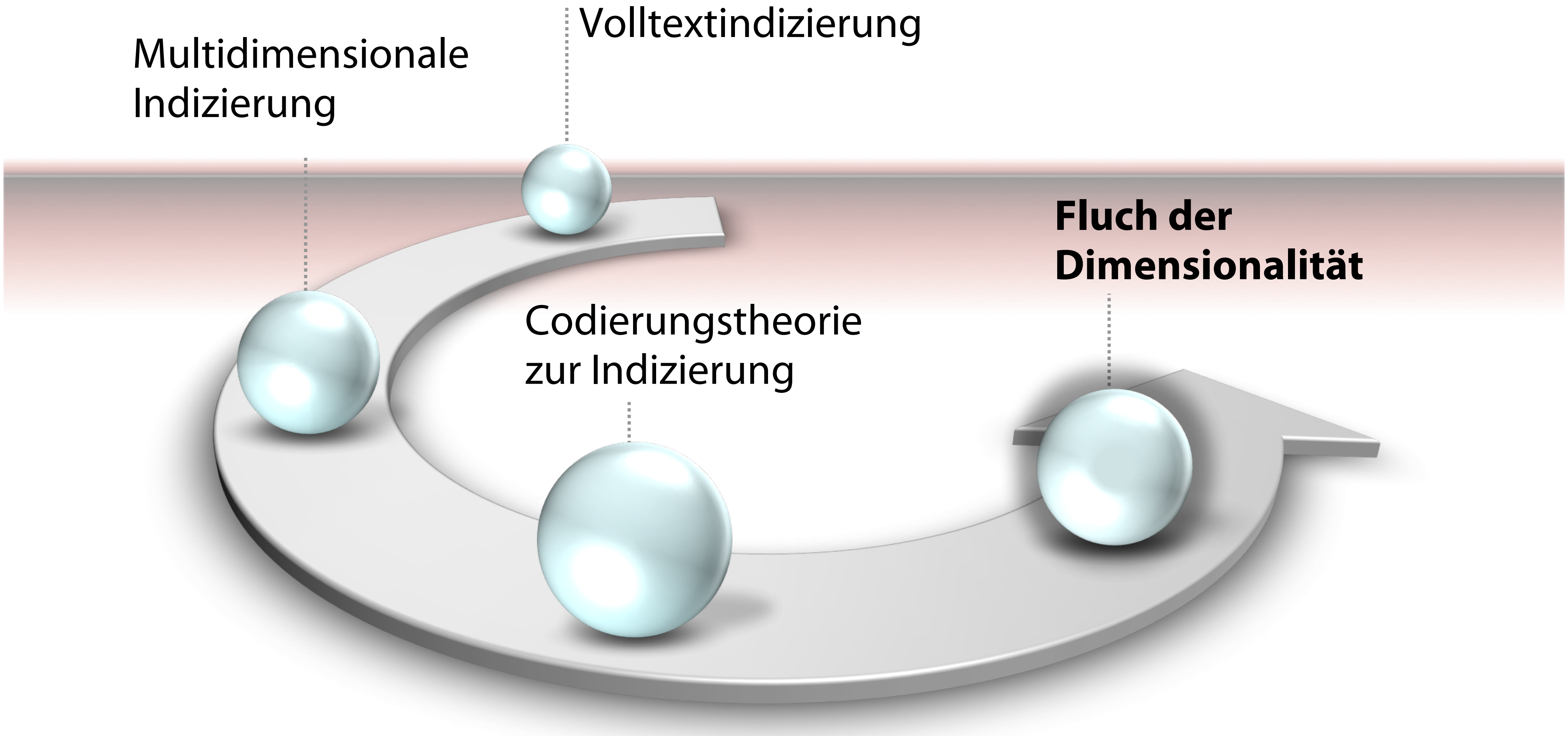
- UB-Bäume sind dynamisch in Bezug auf Änderungen (bedingt durch die zugrundeliegenden B-Bäume)
- Kommerzielle Verwendung im Transbase Datenbanksystem
- Raumfüllende Kurven vieldiskutiert in der Literatur (z.B. Hilbert-Kurven)



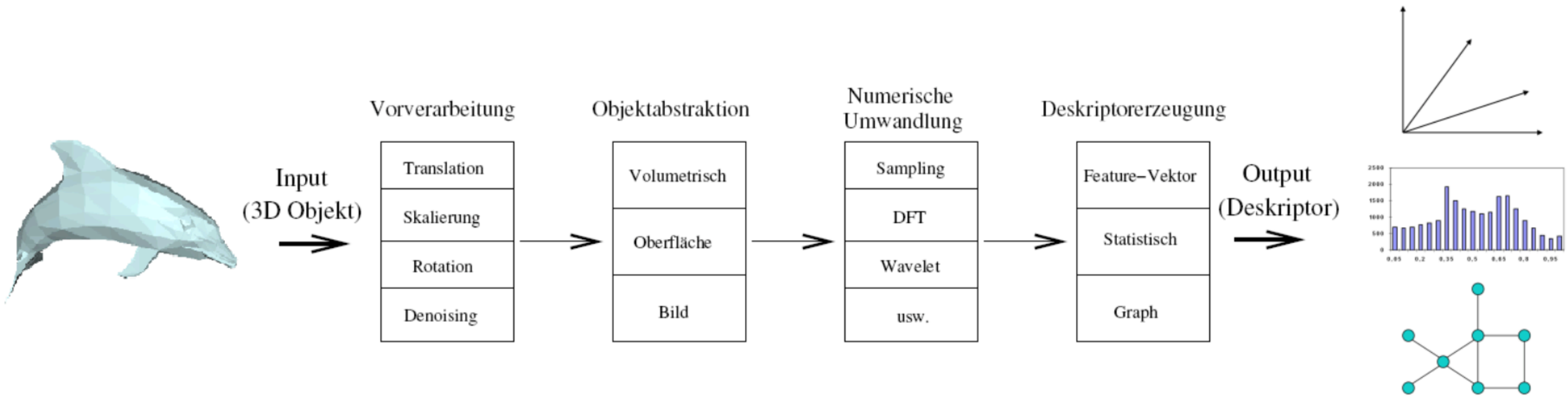
F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, R. Bayer, Integrating the UB-Tree into a Database System Kernel, In Proc. 26th International Conference on Very Large Data Bases, pp. 263-272, **2000**

Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung



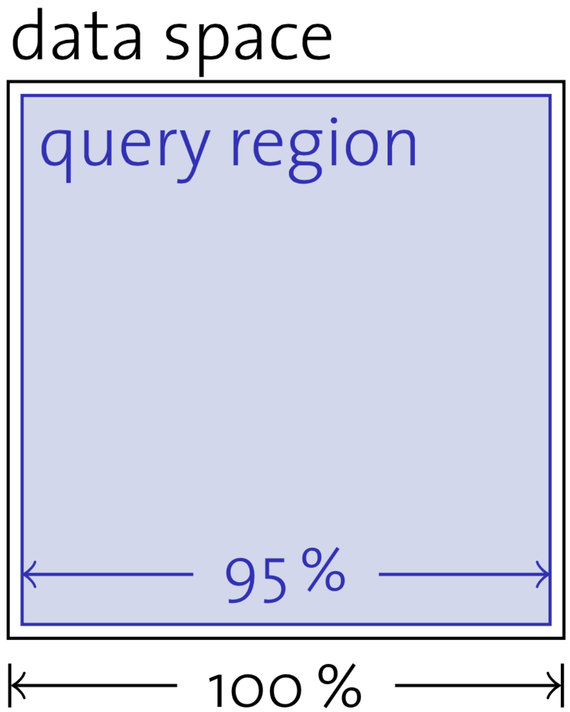
Anwendungen: Multimedia-Datenbanken



- Inhaltsbasierte Suche
- Viele Merkmalsvektoren
- Hochdimensionale Räume

Fluch der Dimensionalität

- **Für große Werte von *k*** sind die diskutierten Techniken **wenig effektiv**
  - Für k=100 ergeben sich $2^{100} \approx 10^{30}$ Partitionen pro Knoten in einem Punkt-Quad-Baum
  - Selbst bei Milliarden von Datenpunkten sind fast alle Partitionen leer
  - Betrachten wir eine sehr große Region („Würfel") mit einer Abdeckung von 95% der Region in jeder Dimension



Für k = 100 ergibt sich eine Wahrscheinlichkeit von $0.95^{100} \approx 0{,}59\%$, dass ein Punkt in dieser Region liegt

Multidimensionale Indizierung