

---

# **Non-Standard-Datenbanken und Data Mining**

Multidimensionale Indizierung

Prof. Dr. Ralf Möller

**Universität zu Lübeck**

**Institut für Informationssysteme**



# Übersicht

---

- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- **Mehrdimensionale Indexstrukturen**
- Cluster-Bildung
- Einbettungstechniken
- First-n-, Top-k-, und Skyline-Anfragen
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell,
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining, Probabilistische Skylines
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, Blockchain-Datenbanken

# Übersicht

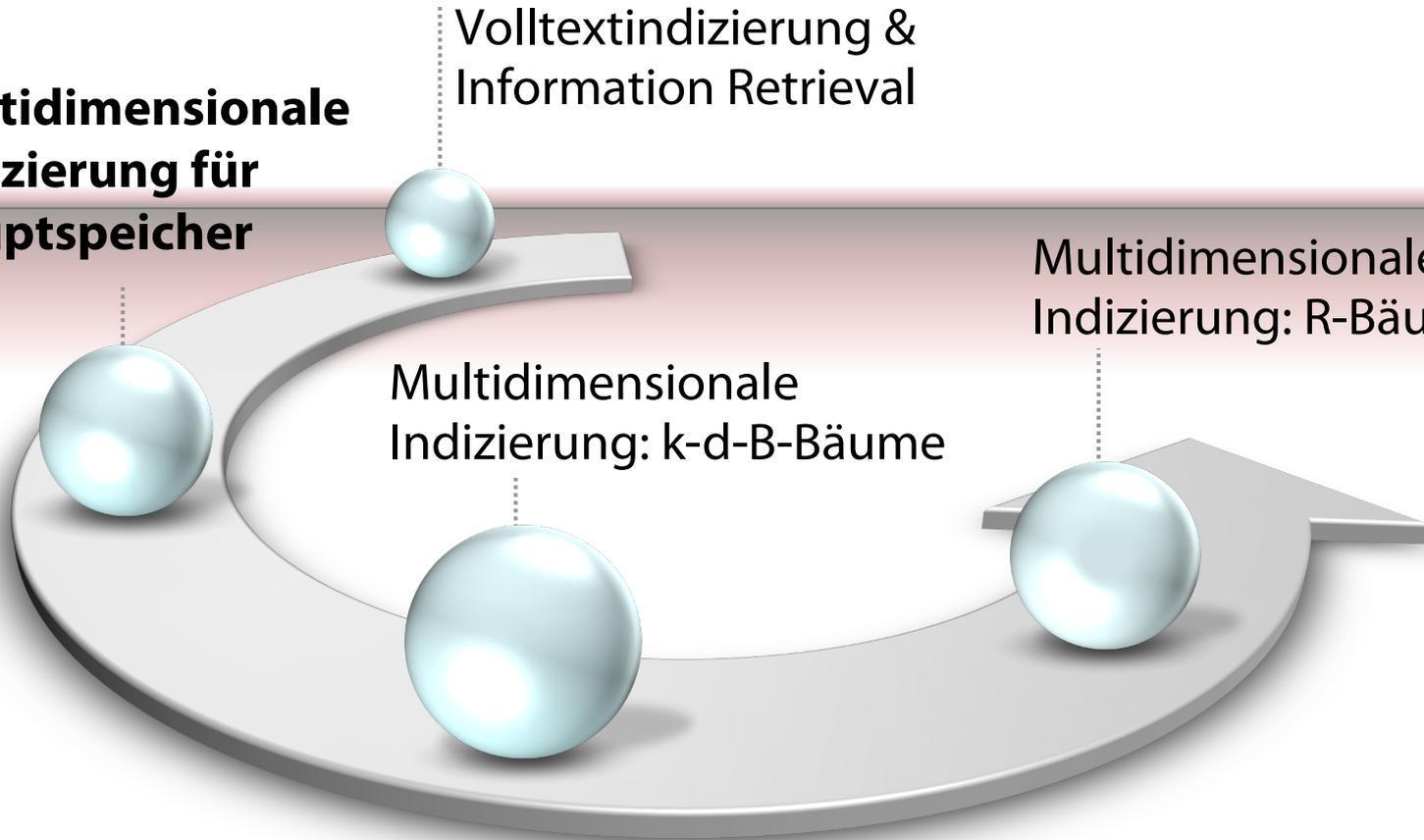
---

**Multidimensionale  
Indizierung für  
Hauptspeicher**

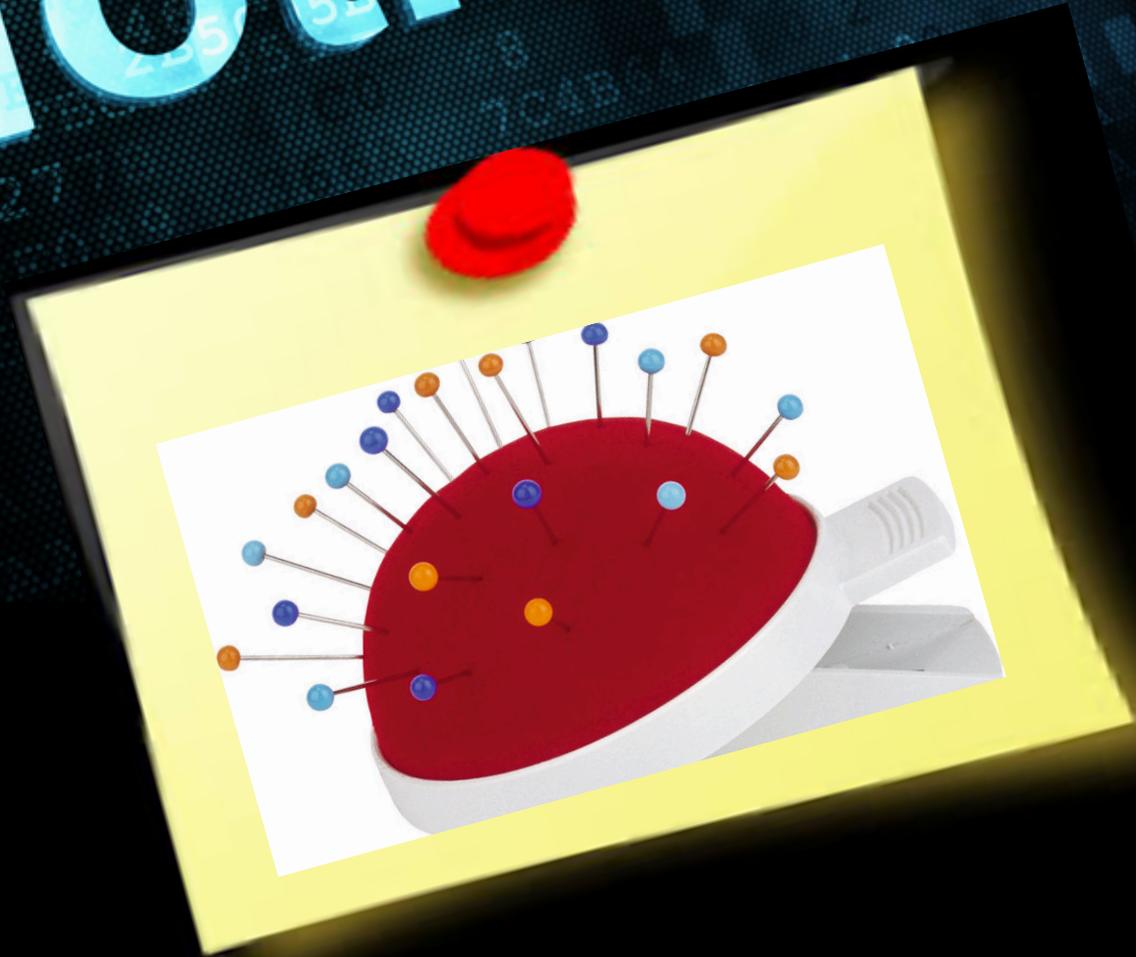
Volltextindizierung &  
Information Retrieval

Multidimensionale  
Indizierung: R-Bäume

Multidimensionale  
Indizierung: k-d-B-Bäume



# Motivation



# Danksagung

---

Die nachfolgenden Präsentationen sind motiviert durch  
Materialien einer Vorlesung von Jens Teubner  
Insbesondere die Bilder habe ich übernommen  
Ich bedanke mich für die Bereitstellung des Materials

# Indexe für mehr Dimensionen ...

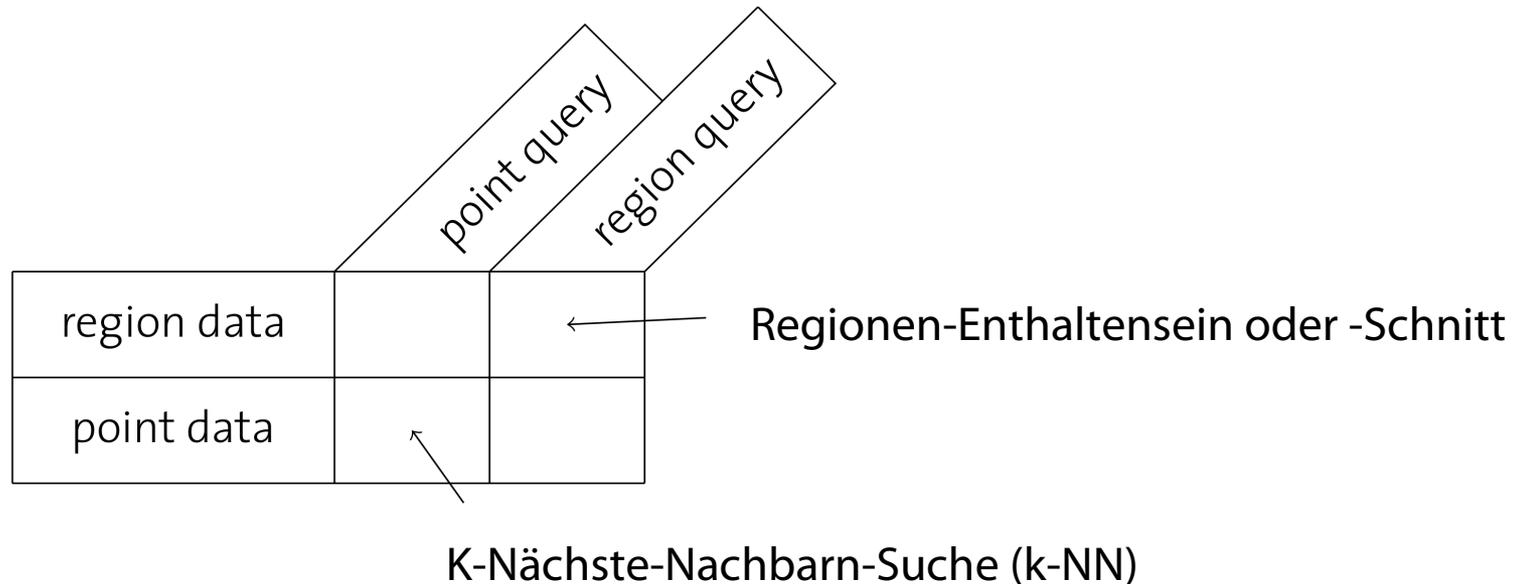
---

```
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
  - Information-Retrieval und Multimedia-Systeme (Dokumentsuche)

# ... weitere Herausforderungen

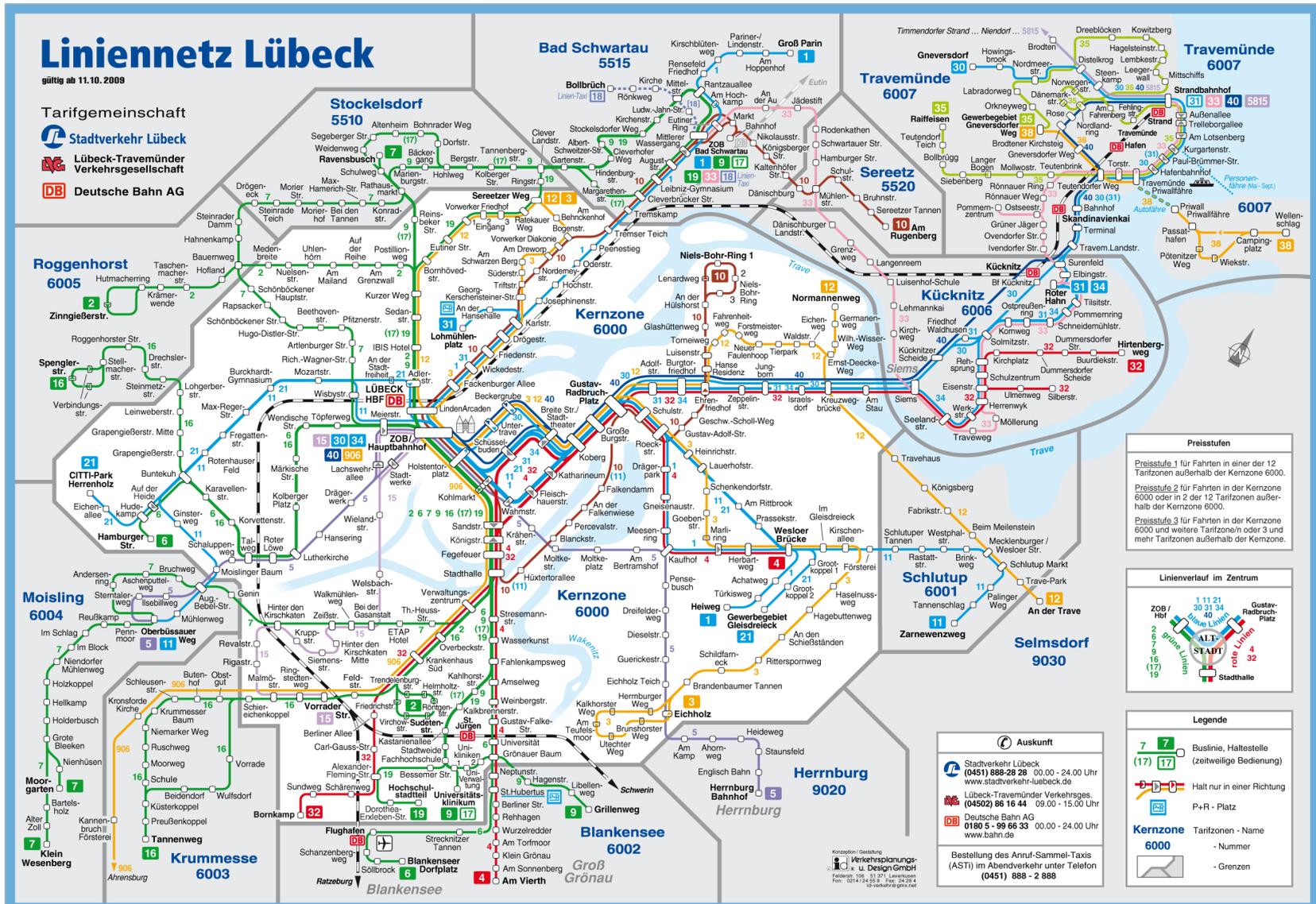
Anfragen und Daten können Punkte oder Regionen sein



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

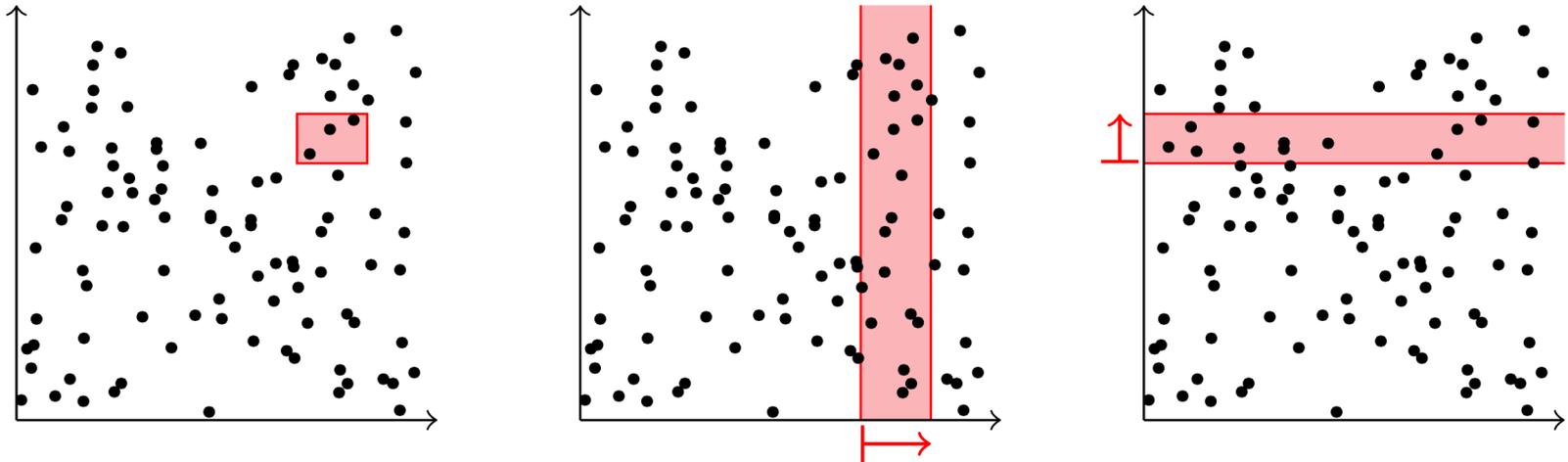
NB: Anfragen mit Gleichheit lassen sich in eindimensionale Anfragen zerlegen

# Punkte, Linien und Regionen



# Können wir nicht einfach B<sup>+</sup>-Bäume verwenden?

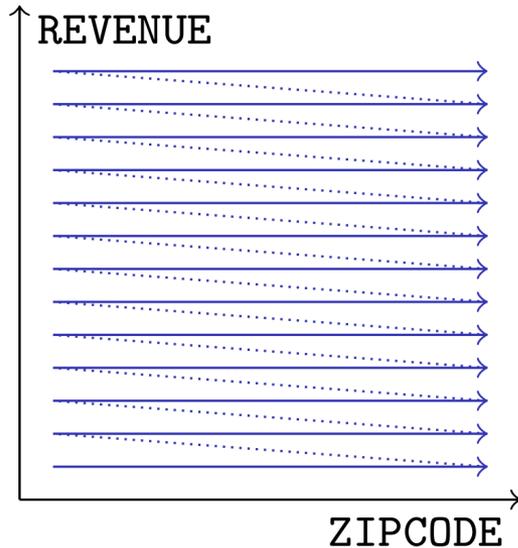
Vielleicht zwei B<sup>+</sup>-Bäume für ZIPCODE und REVENUE?



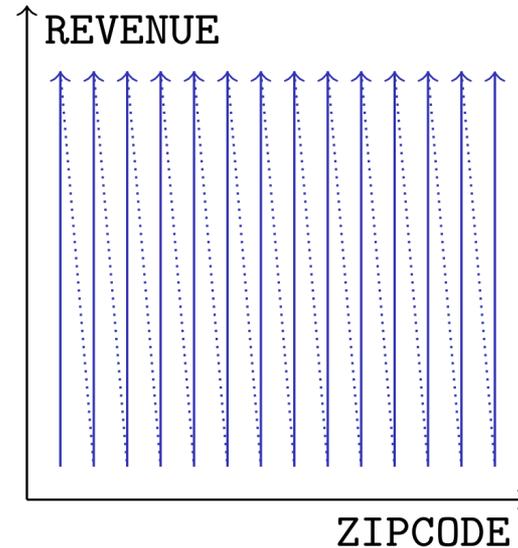
Man kann **pro Dimension** nur über **einen** Index laufen und hat viele **falsche Treffer**

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

# Oder zusammengesetzte Schlüssel?



$\langle \text{REVENUE}, \text{ZIPCODE} \rangle$  index



$\langle \text{ZIPCODE}, \text{REVENUE} \rangle$  index

## Gleiche Situation!

Indizes über zusammengesetzte Schlüssel sind **nicht symmetrisch**.  
Das Hauptattribut dominiert die Organisation des B<sup>+</sup>-Baums

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

# Multidimensionale Indexstrukturen

---

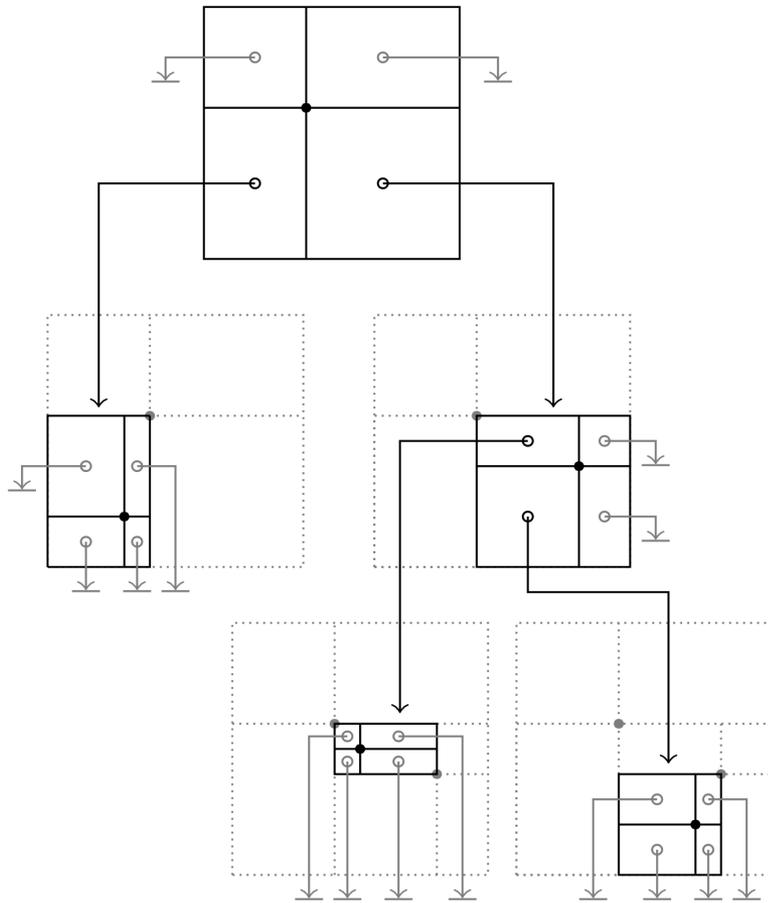
- B<sup>+</sup>-Bäume unterstützen nur **eindimensionale** Anfragen<sup>1</sup>
- Wir suchen multidimensionale Indexstrukturen mit folgenden Eigenschaften
  - Symmetrie in allen Dimensionen
  - Raumorientierte Gruppierung von Daten
  - Dynamisch in Bezug auf Schreiboperationen
  - Unterstützung von häufigen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

---

<sup>1</sup> 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 **Nullzeiger**
- Eine solche Datenstruktur heißt **Punkt-Quad-Baum**

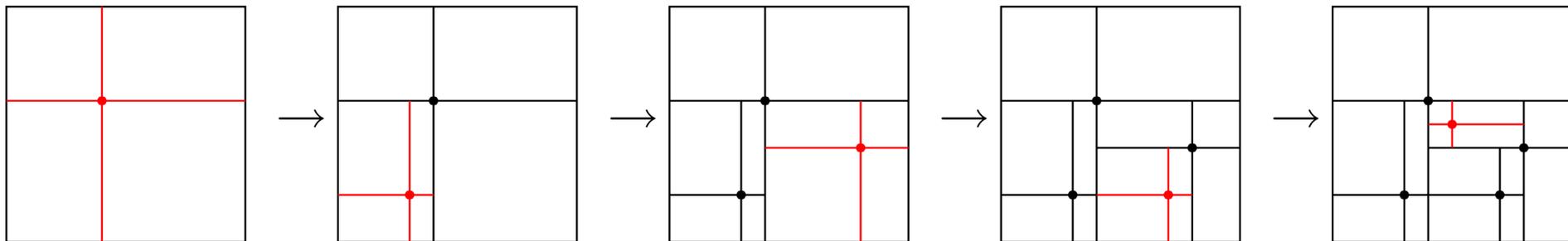


# Einfügen in einen Punkt-Quad-Baum

**Einfügen** eines Punktes  $q_{new}$  in einen Quad-Baum funktioniert wie das Einfügen in einen Binärbaum

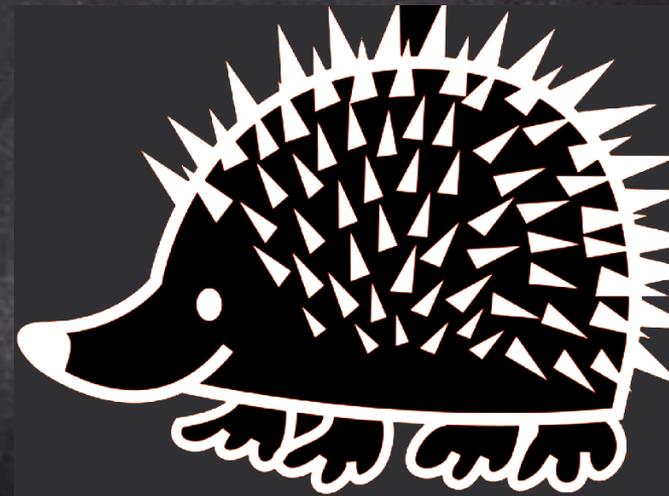
1. **Traversiere** den Baum, so als suche man nach  $q_{new}$  bis eine Partition  $P$  mit einem **Nullzeiger** erreicht ist
2. Erzeuge **neuen Knoten**  $n'$ , der die Region  $P$  aufspannt und durch  $q_{new}$  partitioniert wird (mit **Null** für alle Subpartitionen)
3. Lasse  $P$  auf  $n'$  zeigen

Leider bleibt der Baum **nicht immer balanciert**



# Denkaufgabe: Vektorfeld

Wie sollten wir  $k$  nächste Nachbarn bestimmen?



# Bereichsanfragen

---

Um eine Bereichsanfrage<sup>2</sup> zu evaluieren, müssen ggf. mehrere Regionen verfolgt werden

- 1 **Function:**  $r\_search(Q, node)$
- 2 **if** data point in  $node$  is in  $Q$  **then**
- 3      $\lfloor$  **append** data point to result ;
- 4 **foreach** partition  $P$  in  $node$  that intersects with  $Q$  **do**
- 5      $\lfloor$   $node' \leftarrow$  node pointed to by  $P$  ;
- 6      $\lfloor$   $r\_search(Q, node')$  ;

- 
- 1 **Function:**  $regionsearch(Q)$
  - 2 **return**  $r\_search(Q, root)$  ;

---

<sup>2</sup> Wir betrachten rechteckige Regionen, bei allgemeinen Regionen sind Umschließungsboxen zu bilden 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

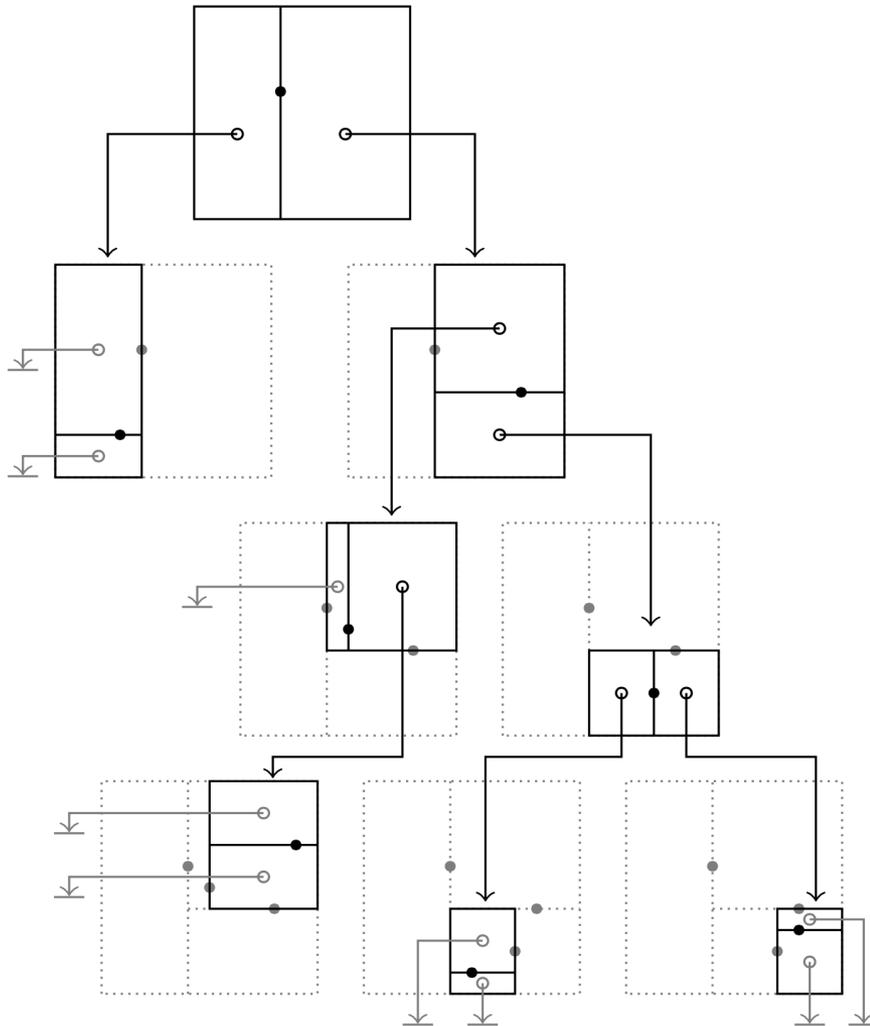
- die Form hängt von der **Einfügereihenfolge** ab  
(im **schlimmsten** Fall entsteht eine **verkettete Liste**)
- **Nullzeiger** sind speicher**ineffizient** (ins. bei großem  $k$ )

Und

- 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**

# k-d-Bäume

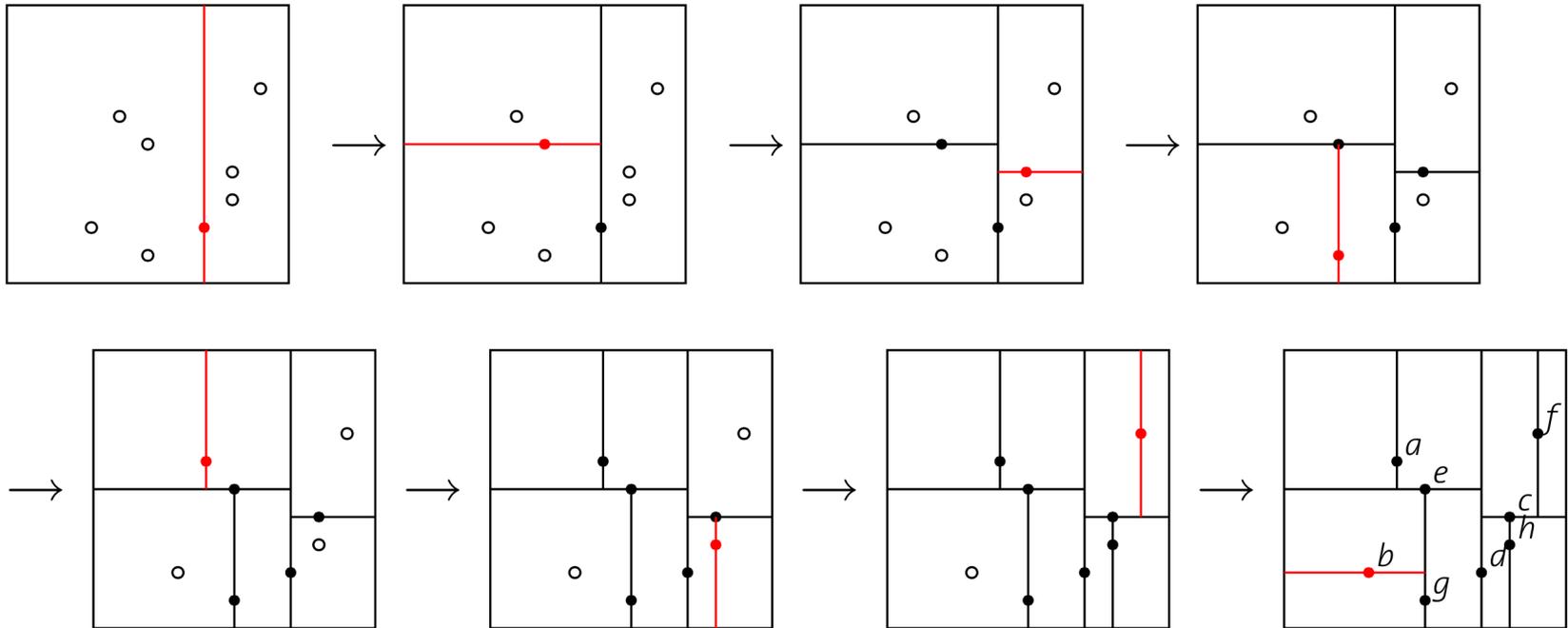
k-d-Bäume übernehmen die positiven Eigenschaften von Punkt-Quad-Bäumen, sind aber **speichereffizienter**

Für eine gegebene Punktmenge kann ein **balancierter** k-d-Baum konstruiert werden<sup>3</sup>

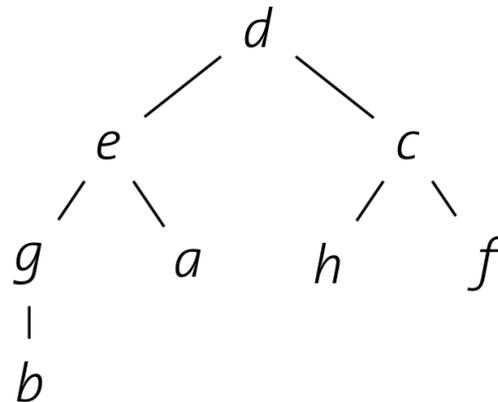
```
1 Function: kdtree (pointset, level)
2 if pointset is empty then
3   | return null ;
4 else
5   |  $p \leftarrow$  median from pointset (along  $d_{level}$ ) ;
6   |  $points_{left} \leftarrow \{v \in pointset \text{ where } v_{d_{level}} < p_{d_{level}}\}$ ;
7   |  $points_{right} \leftarrow \{v \in pointset \text{ where } v_{d_{level}} \geq p_{d_{level}}\}$ ;
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  $n$  ;
```

<sup>3</sup>  $v_i$ : Koordinate  $i$  von Punkt  $v$

# Balancierte $k$ -d-Baum-Konstruktion

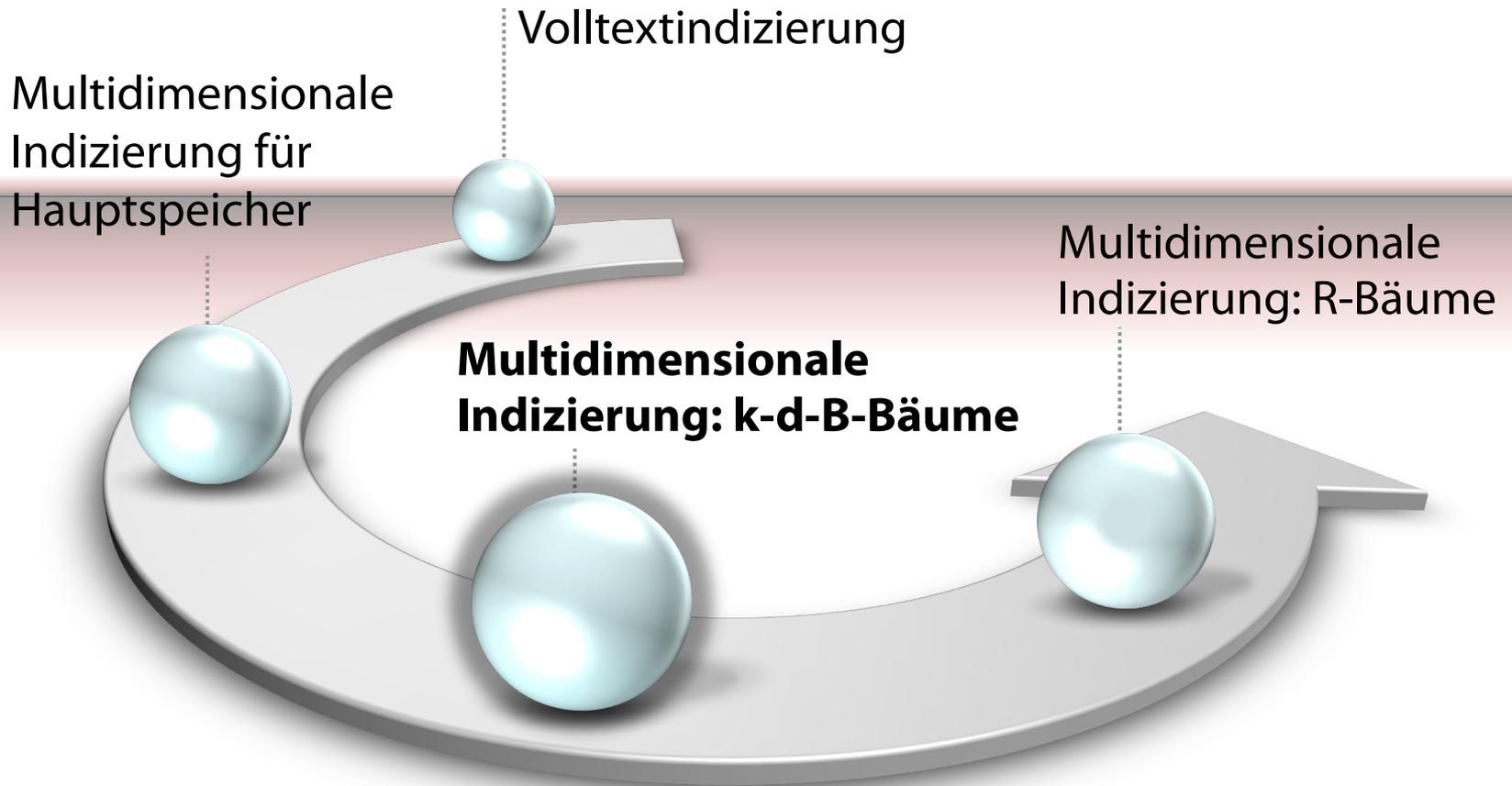


Ergebnis



# Non-Standard-Datenbanken

## Multidimensionale Indizierung

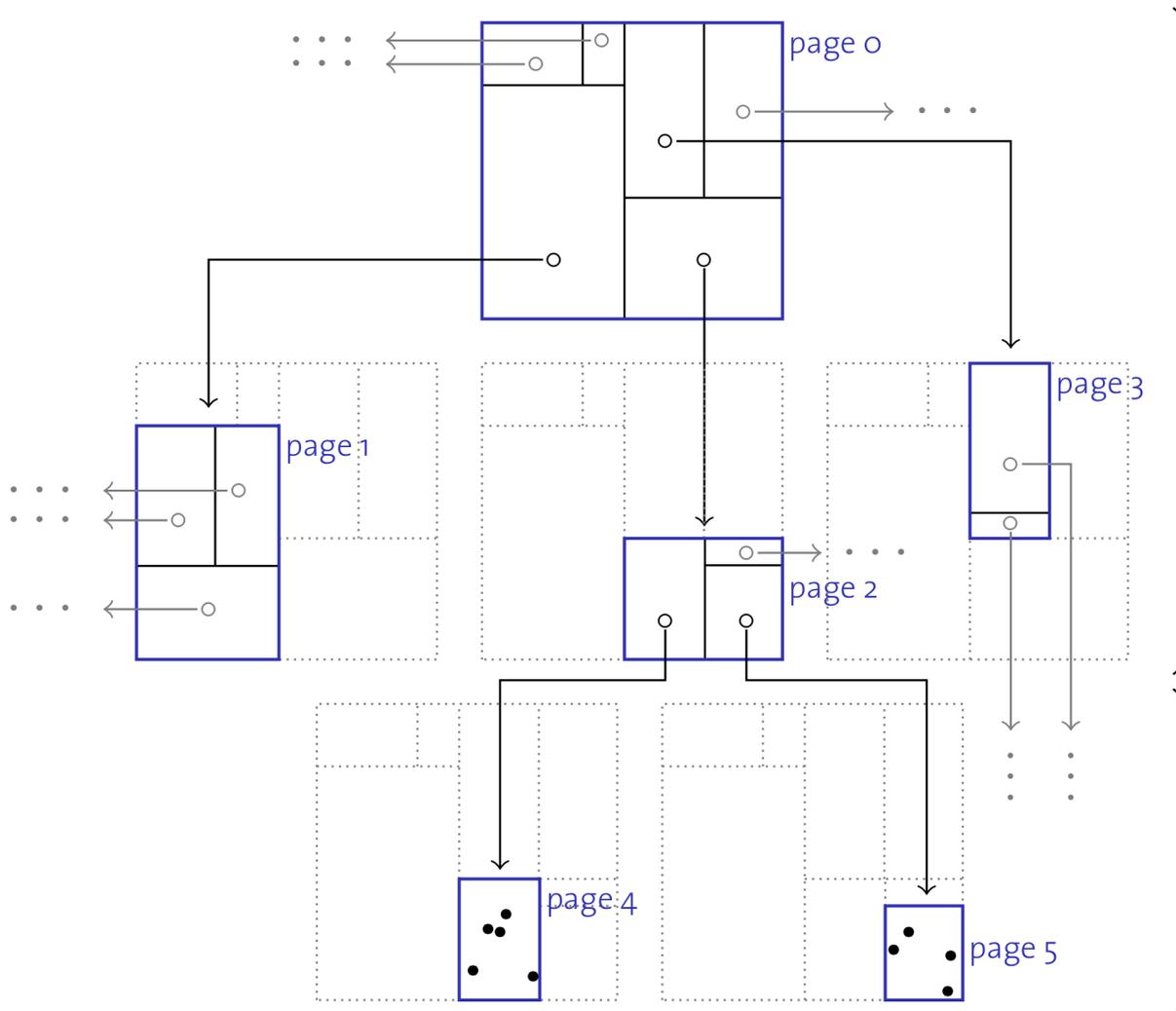


# k-d-B-Bäume

---

- 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

# k-d-B-Bäume: Zentrale Idee



## Regionenseiten

- enthalten Einträge  $\langle \text{region, pageID} \rangle$
- keine **Nullzeiger**
- bilden **balancierten** Baum
- alle Regionen **disjunkt** und **rechteckig**

## Punktseiten

- enthalten Einträge  $\langle \text{point, rid} \rangle$
- $\rightarrow$  Blattknoten B<sup>+</sup>-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<sup>+</sup>-Baum**
  - Füge Eintrag  $\langle \text{region}, \text{pageID} \rangle$  ( $\langle \text{point}, \text{rid} \rangle$ ) in eine Regionenseite (Punktseite) ein, sofern **genügend Platz** vorhanden
  - **Sonst: Splitte** Seite auf

# Aufsplittung einer Punktseite

---

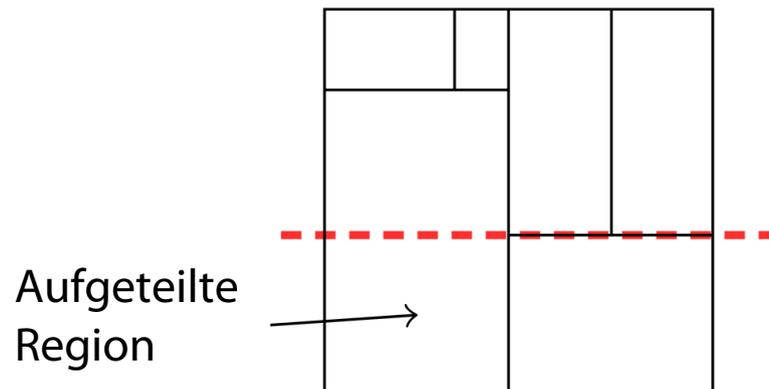
## Aufteilung einer Seite $p$

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**  $\langle region, p \rangle$  auf der **Elternseite** durch  $\langle linke-region, p_{links} \rangle$  und  $\langle rechte-region, p_{rechts} \rangle$

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

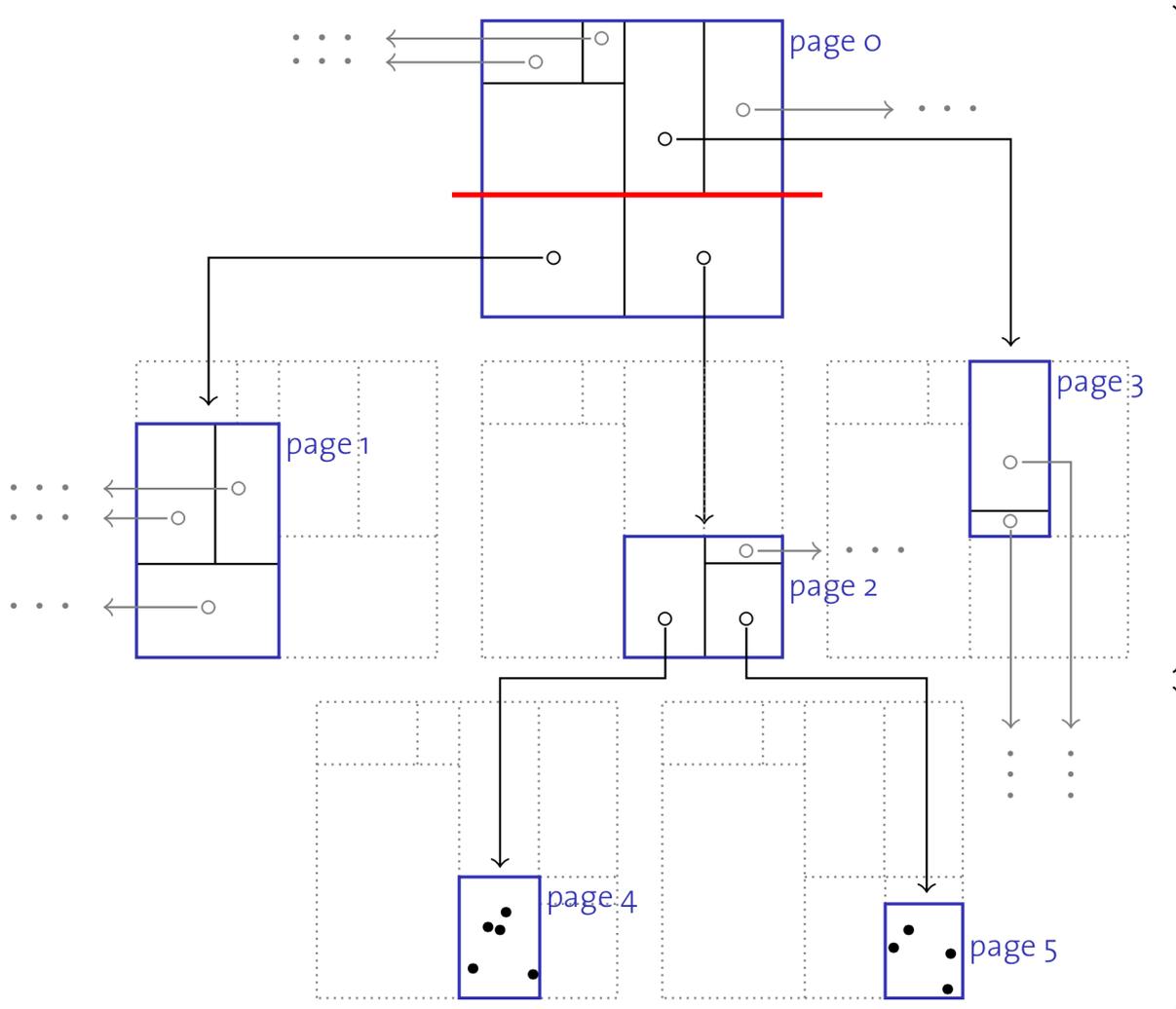
# Aufspaltung einer Regionenseite

- 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



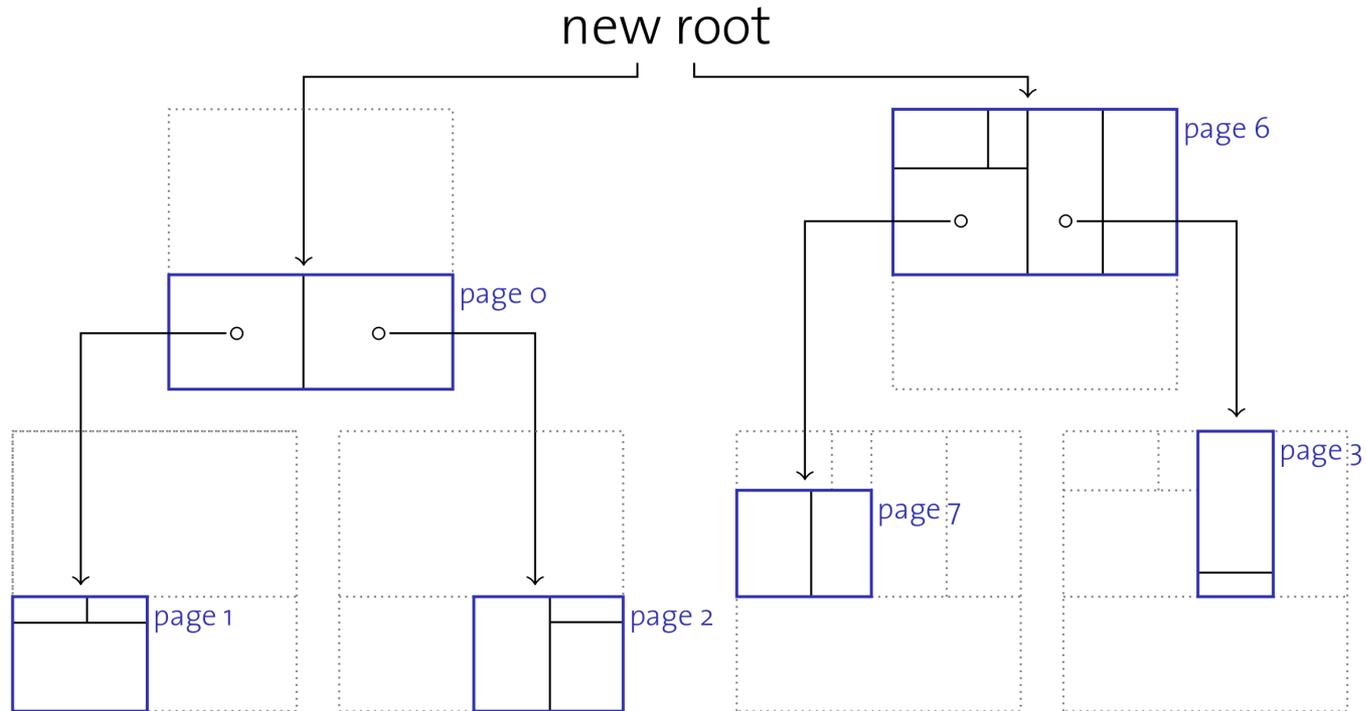
## Regionenseiten

- enthalten Einträge  $\langle \text{region, pageID} \rangle$
- **keine Nullzeiger**
- bilden **balancierten** Baum
- alle Regionen **disjunkt** und **rechteckig**

## Punktseiten

- enthalten Einträge  $\langle \text{point, rid} \rangle$
- $\rightarrow$  Blattknoten B<sup>+</sup>-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 für Regionen als Daten

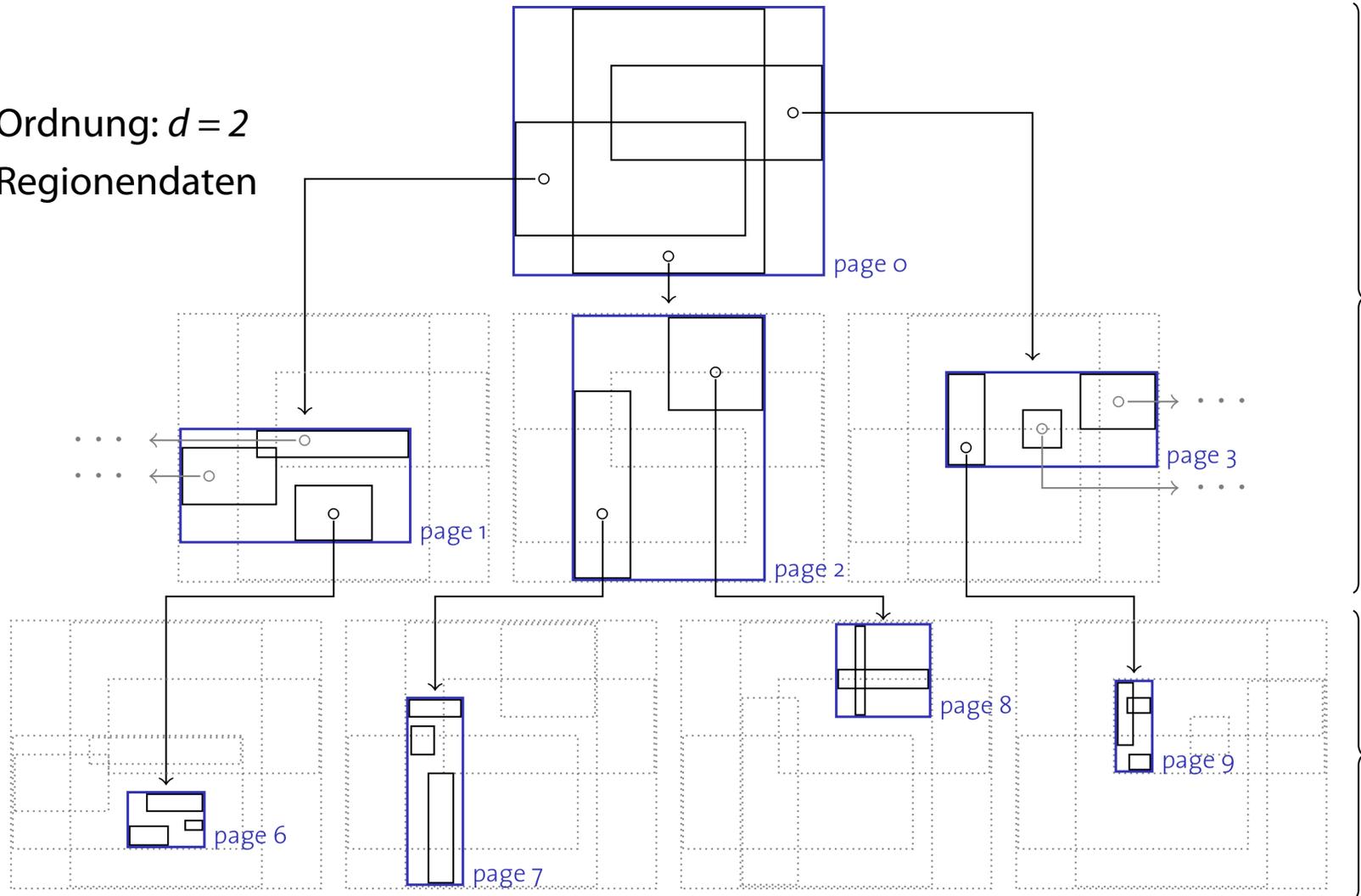
---

Regionen können sich in dieser Struktur überlappen

- Innere Knoten enthalten  $\langle \text{region, pageID} \rangle$  Einträge, Blattknoten enthalten Einträge der Form  $\langle \text{region, rid} \rangle$ , wobei region das **minimale Umgebungsrechteck** der Datenelemente, die über den Zeigern erreichbar sind
  - Jeder Knoten enthält zwischen  $d$  und  $2d$  Elemente ( $\rightarrow B^+$ -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

# R-Baum: Beispiel

- Ordnung:  $d = 2$
- Regionendaten

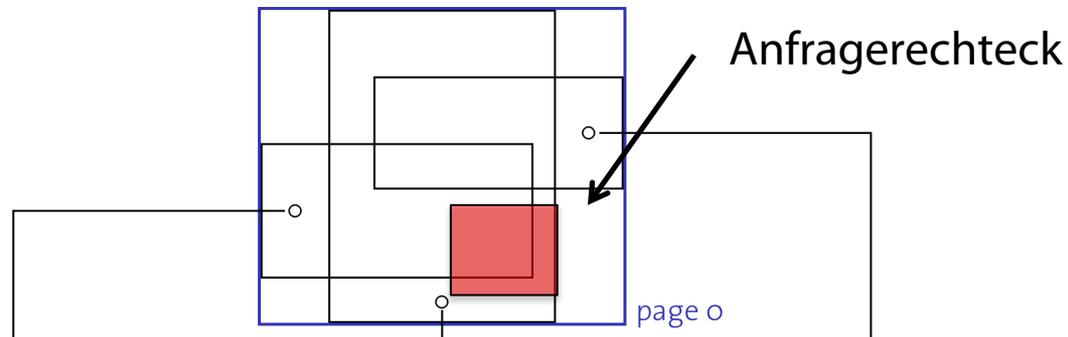


Innere Knoten

Blattknoten

# R-Baum: Regionenabfrage (Schnitt)

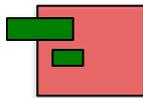
- Ordnung:  $d = 2$
- Regionendaten



Innere Knoten



Blattknoten



# R-Baum: Suchen und Einfügen

---

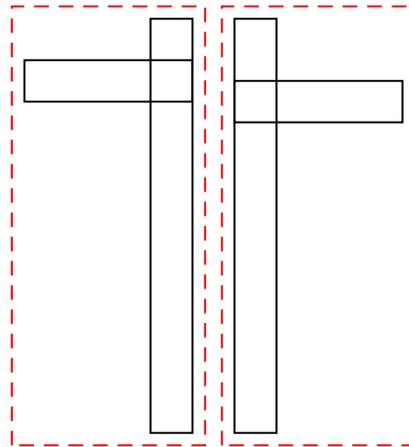
Während der **Suche** müssen ggf. mehrere Kinder betrachtet werden (gilt für Punkt- **und** Regionenanfragen)

**Einfügen** erfolgt wie in einem  $B^+$ -Baum

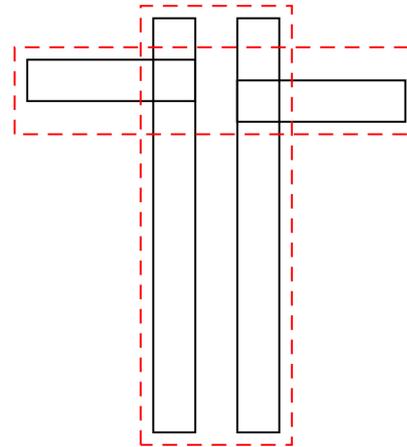
1. **Wähle** richtigen Blattknoten  $n$  für die Einfügung (versuche entstehende neue Rechtecke zu minimieren)
2. Falls  $n$  **voll** ist, **spalte** ihn auf (wir haben  $n$  und  $n'$ ) und verteile alte Einträge auf  $n$  und  $n'$ 
  - Aufspaltungen können nach oben propagieren und erreichen ggf. die Wurzel
3. Nach dem Einfügung müssen Regionen im Vorgängerknoten 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 Nachfolgebapieren ( $R^*$ -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 aufwendige Operation in R-Bäumen

---

# **Non-Standard-Datenbanken und Data Mining**

Multidimensionale Indizierung

Prof. Dr. Ralf Möller

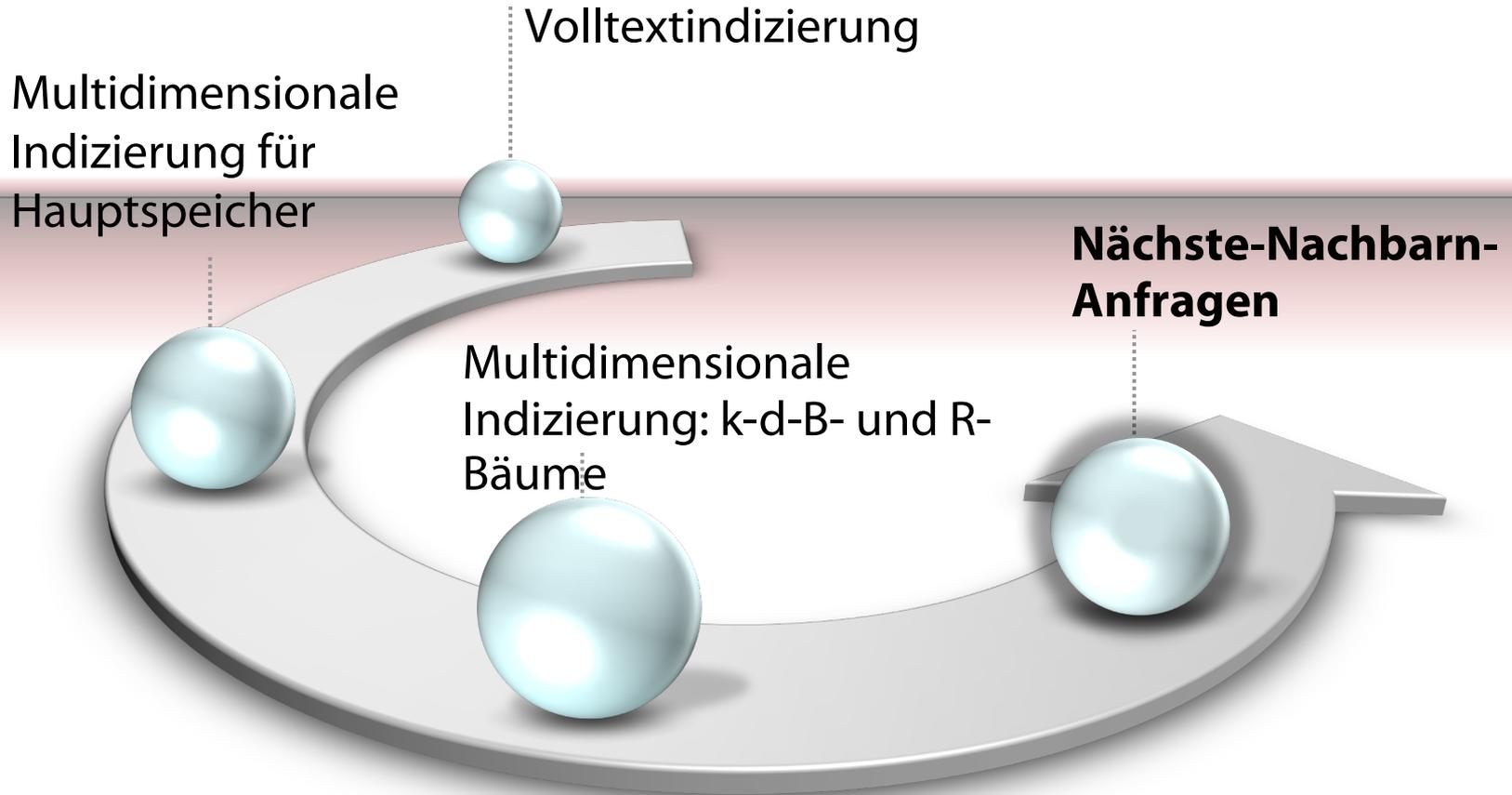
**Universität zu Lübeck**

**Institut für Informationssysteme**



# Non-Standard-Datenbanken

## Multidimensionale Indizierung



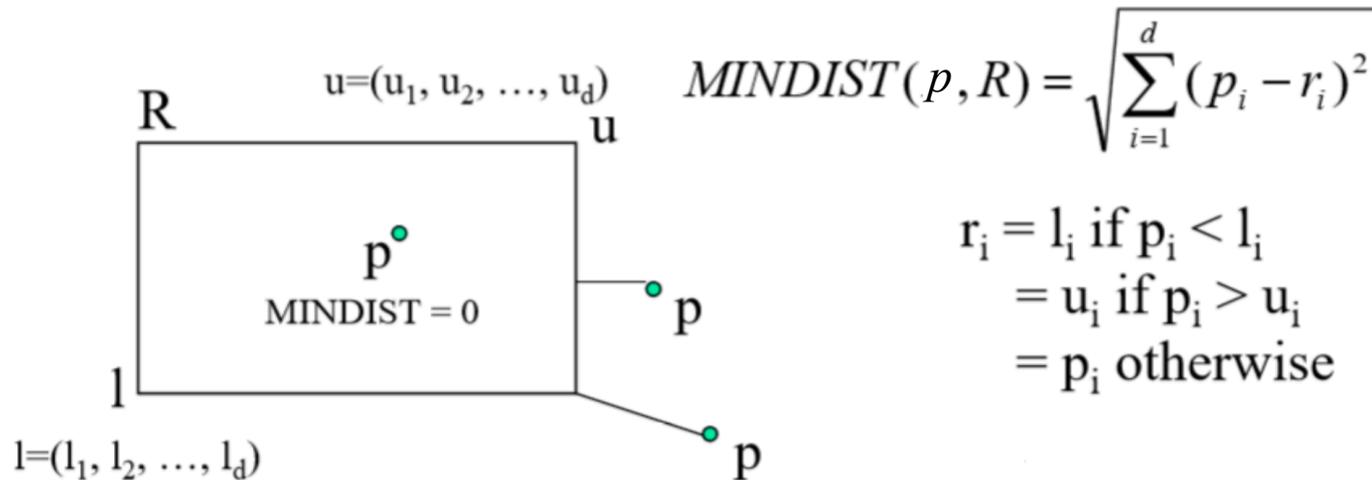
# K-Nächste-Nachbarn-Anfragen

- Sei  $M$  eine (Grund-)Menge,
- $q \notin M$  ein Anfragepunkt,
- $k \in \mathbb{N}$  die gewünschte Anzahl gesuchter nächster Nachbarn von  $q$  und  $d$  eine Abstandsfunktion
- und  $|M| \geq k$ ,
- dann ist die Menge  $NN(k, q, M)$  der  $k$  nächsten Nachbarn von  $q$  definiert als
  - diejenige Teilmenge  $S \subseteq M$ , mit  $|S| = k$
  - so dass es keine Menge  $S' \subseteq M$  mit  $S \cap S' = \emptyset$  gibt,
  - so dass ein  $p \in S$  und  $p' \in S'$  existiert und  $d(q, p) > d(q, p')$

$S'$  muss nicht  
notwendigerweise  
 $k$  Elemente haben.

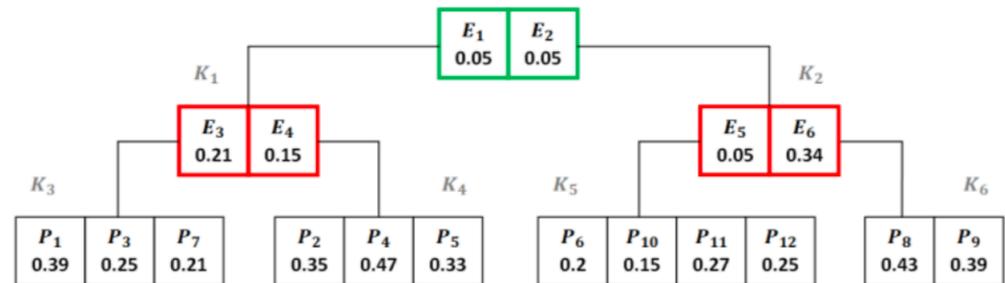
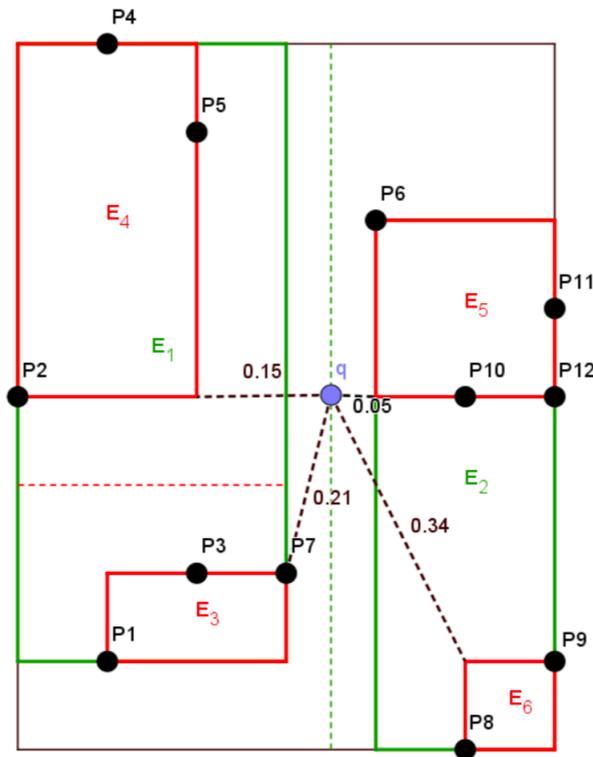
# MINDIST

- Nehmen wir an, wir haben einen Punkt  $p$  und ein Rechteck  $R$  definiert durch zwei Punkte  $l$  und  $u$ , dann sei die Funktion  $MINDIST(p, R)$  wie folgt definiert:

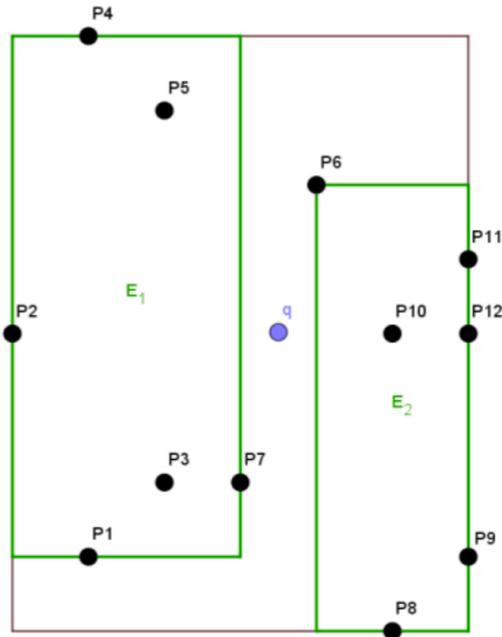


# Branch-and-Bound-Ranking-Search mit R-Baum

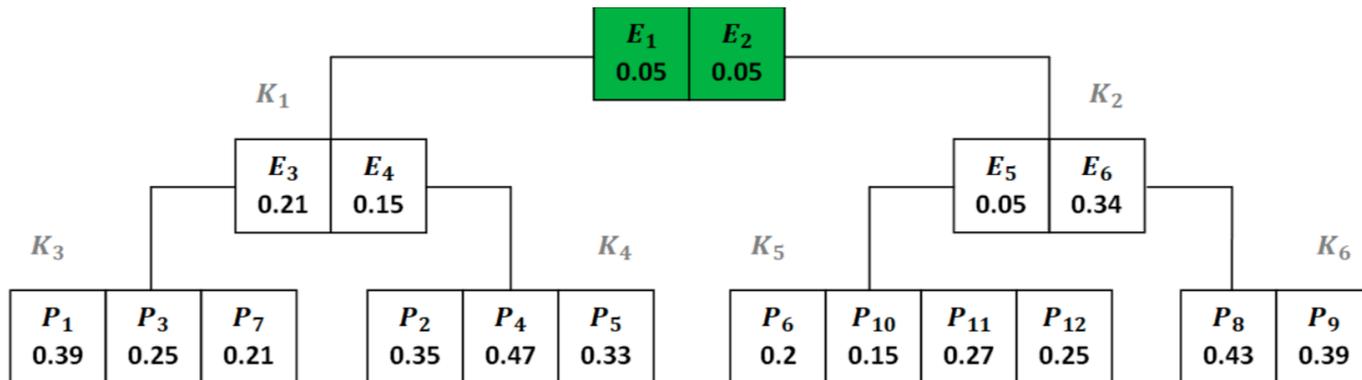
ID	1	2	3	4	5	6	7	8	9	10	11	12
Attribut 1	0,2	0,1	0,3	0,2	0,3	0,5	0,4	0,6	0,7	0,6	0,7	0,7
Attribut 2	0,2	0,5	0,3	0,9	0,8	0,7	0,3	0,1	0,2	0,5	0,6	0,5
Abstand	0,39	0,35	0,25	0,47	0,33	0,2	0,21	0,43	0,39	0,15	0,27	0,25

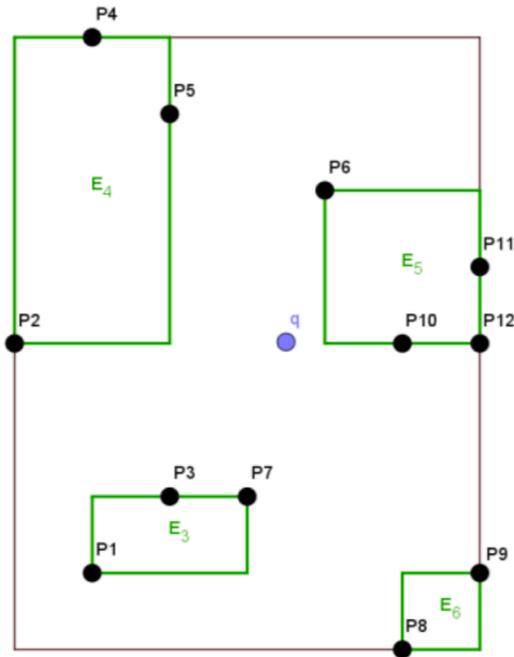


G. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," ACM TODS, 24(2), pp. 265- 318, Juni 1999

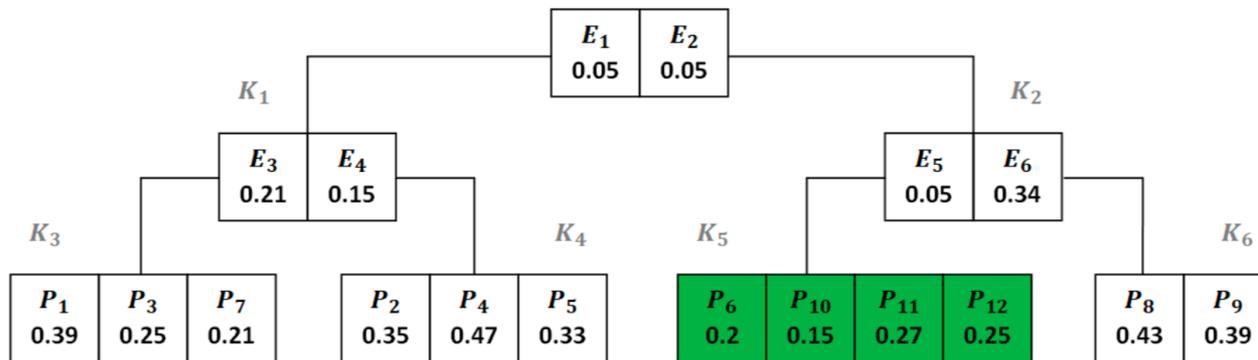


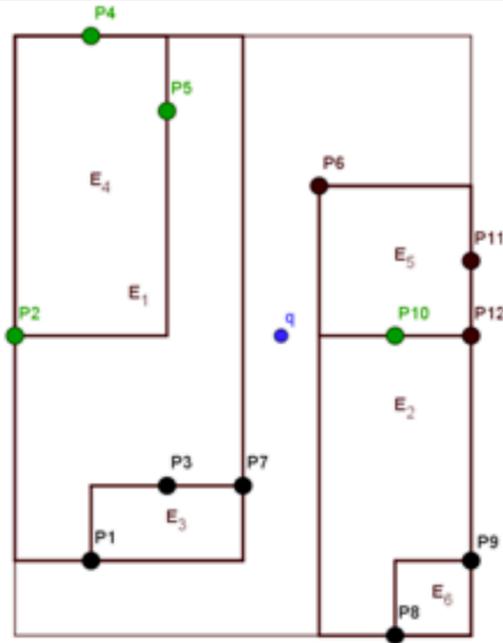
Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$



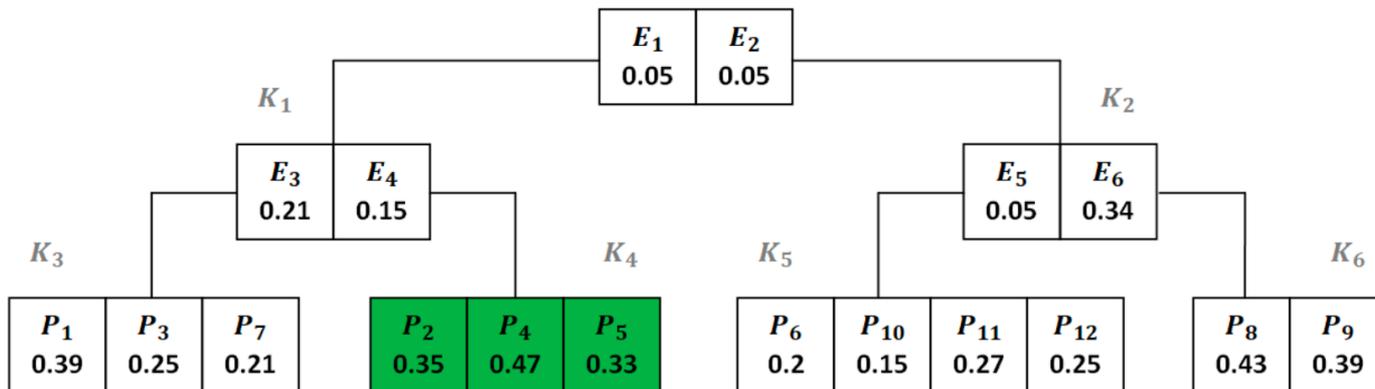


Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$
Folge $E_1$ und $E_2$	$(E_5, 0.05), (E_4, 0.15), (E_3, 0.21), (E_6, 0.34)$	$\emptyset$





Aktion	Warteschlange	Ergebnis
Lade Wurzel	$(E_1, 0.05), (E_2, 0.05)$	$\emptyset$
Folge $E_1$ und $E_2$	$(E_5, 0.05), (E_4, 0.15), (E_3, 0.21), (E_6, 0.34)$	$\emptyset$
Folge $E_5$	$(E_4, 0.15), (P_{10}, 0.15), (P_6, 0.2), (E_3, 0.21), (P_{12}, 0.25), (P_{11}, 0.27), (E_6, 0.34)$	$\emptyset$
Folge $E_4$	$(P_{10}, 0.15), (P_6, 0.2), (E_3, 0.21), (P_{12}, 0.25), (P_{11}, 0.27), (P_5, 0.33), (P_2, 0.35), (E_6, 0.34), (P_4, 0.47)$	$\{P_{10}\}$



# Branch-and-Bound-Ranked-Search (BRS)

```
algorithm BRS (rt, k, type, score, f, filter)
  // rt is an R-tree on the dataset, k denotes the number of data points to return
  // type can be either min or max, score computes a score for an rtree entry (MBR) based on f
  // f is a point scoring function, filter is a filter for the data points (where clause)
  let pq = build_priority_queue(type, {})
    result = {}
    n = 0
    object
  for e ∈ root(rt) do
    insert((e, score(mbr(e), f)), pq)
  while n < k and not empty?(pq) do
    object := delete_next(pq)
    if point?(object) then
      result := result ∪ {object}
      n := n + 1
    else
      if leaf?(object) then
        for p ∈ filter(points(mbr(object))) do
          insert((p, f(p)), pq)
      else
        for e ∈ children(object) do
          insert((e, score(mbr(e), f)), pq)
  return result
```

```
BRS(data-rtree, 3, 'min', MINDIST, f, identity)
```

# Non-Standard-Datenbanken

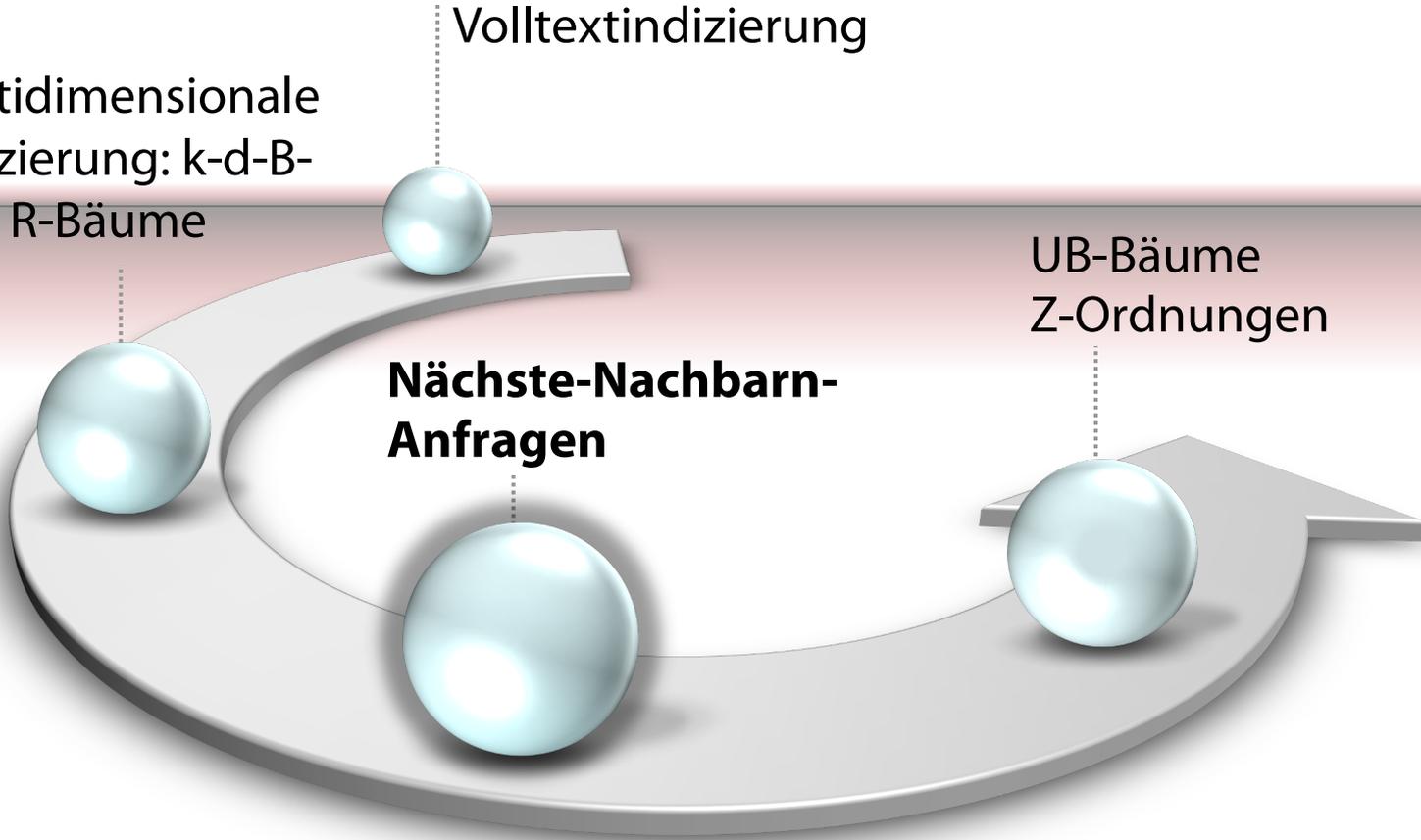
## Multidimensionale Indizierung

Multidimensionale Indizierung: k-d-B- und R-Bäume

Volltextindizierung

UB-Bäume  
Z-Ordnungen

**Nächste-Nachbarn-Anfragen**



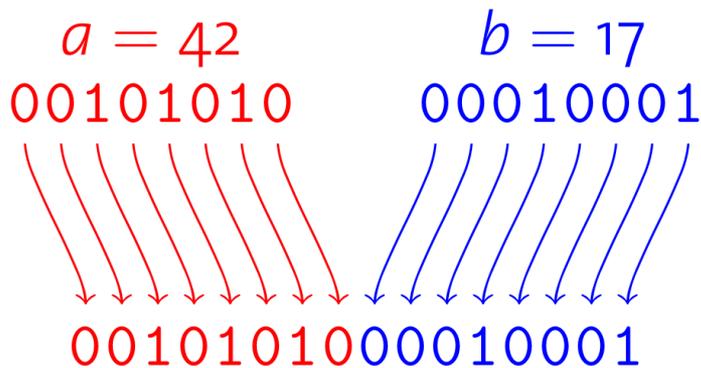
---

# Von Z-Ordnungen Zu UB-Bäumen

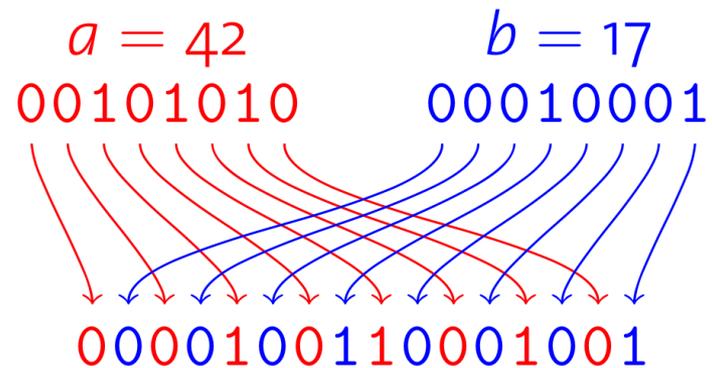


# Bit-Verschränkung

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

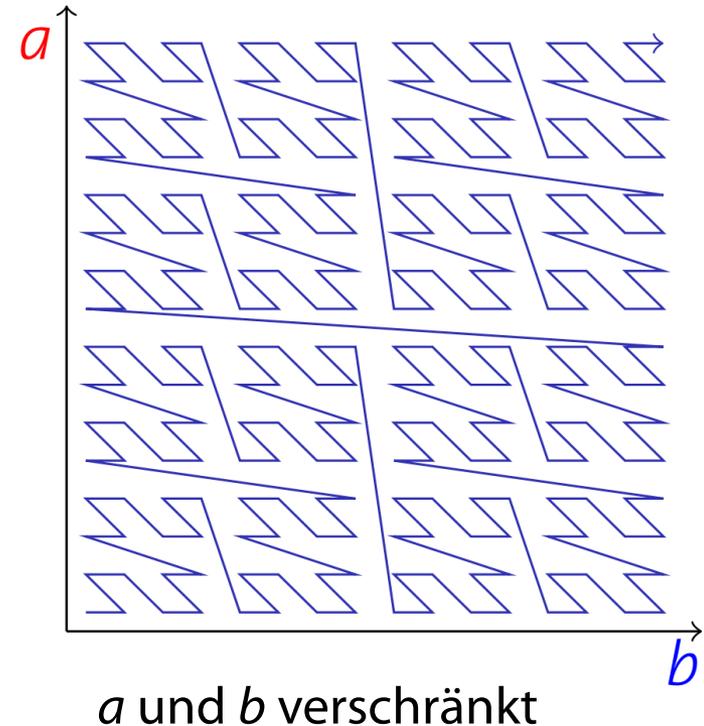
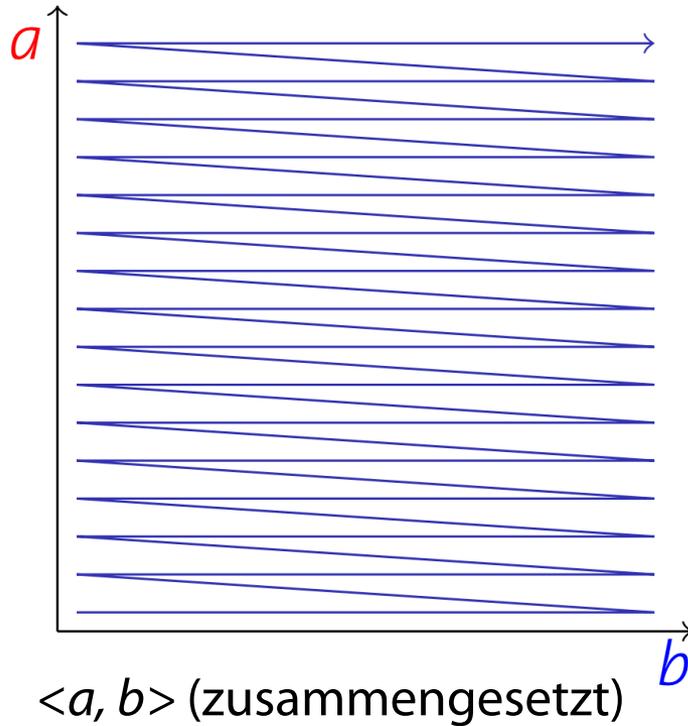


$\langle a, b \rangle$  (zusammengesetzt)



$a$  und  $b$  verschränkt

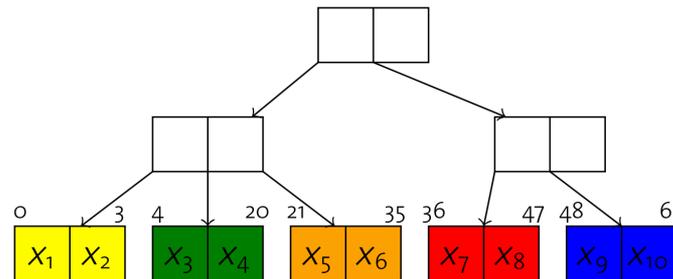
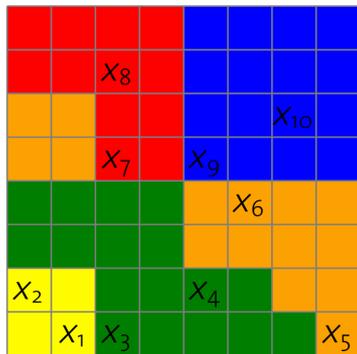
# Z-Ordnung



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

# B<sup>+</sup>-Bäume über Z-Ordnungen

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

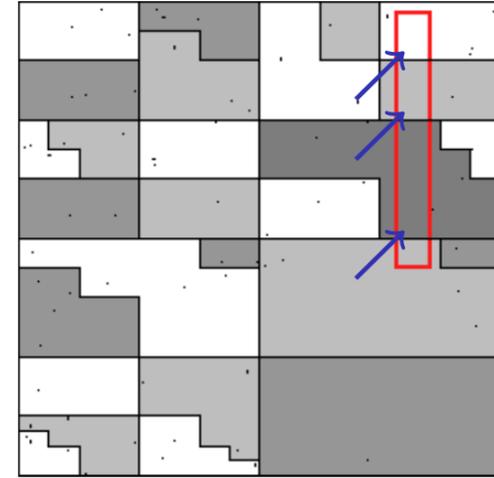


- Um alle Datenpunkte für eine Anfrage  $Q$  zu finden, sollen nur solche Blattseiten betrachtet werden, die Regionen enthalten, die sich mit  $Q$  **schneiden**

# UB-Baum-Bereichsanfragen

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

```
1 Function:  $ub\_range(Q)$ 
2  $cur \leftarrow z(Q_{bottom, left})$ ;
3 while true do
4     // search B+-tree page containing  $cur$ 
5      $page \leftarrow search(cur)$ ;
6     foreach data point  $p$  on  $page$  do
7         if  $p$  is in  $Q$  then
8             append  $p$  to result;
9         if region in  $page$  reaches beyond  $Q_{top, right}$  then
10            break;
11     // compute next Z-address using  $Q$  and data on current page
12      $cur \leftarrow get\_next\_z\_address(Q, page)$ ;
```





# Non-Standard-Datenbanken

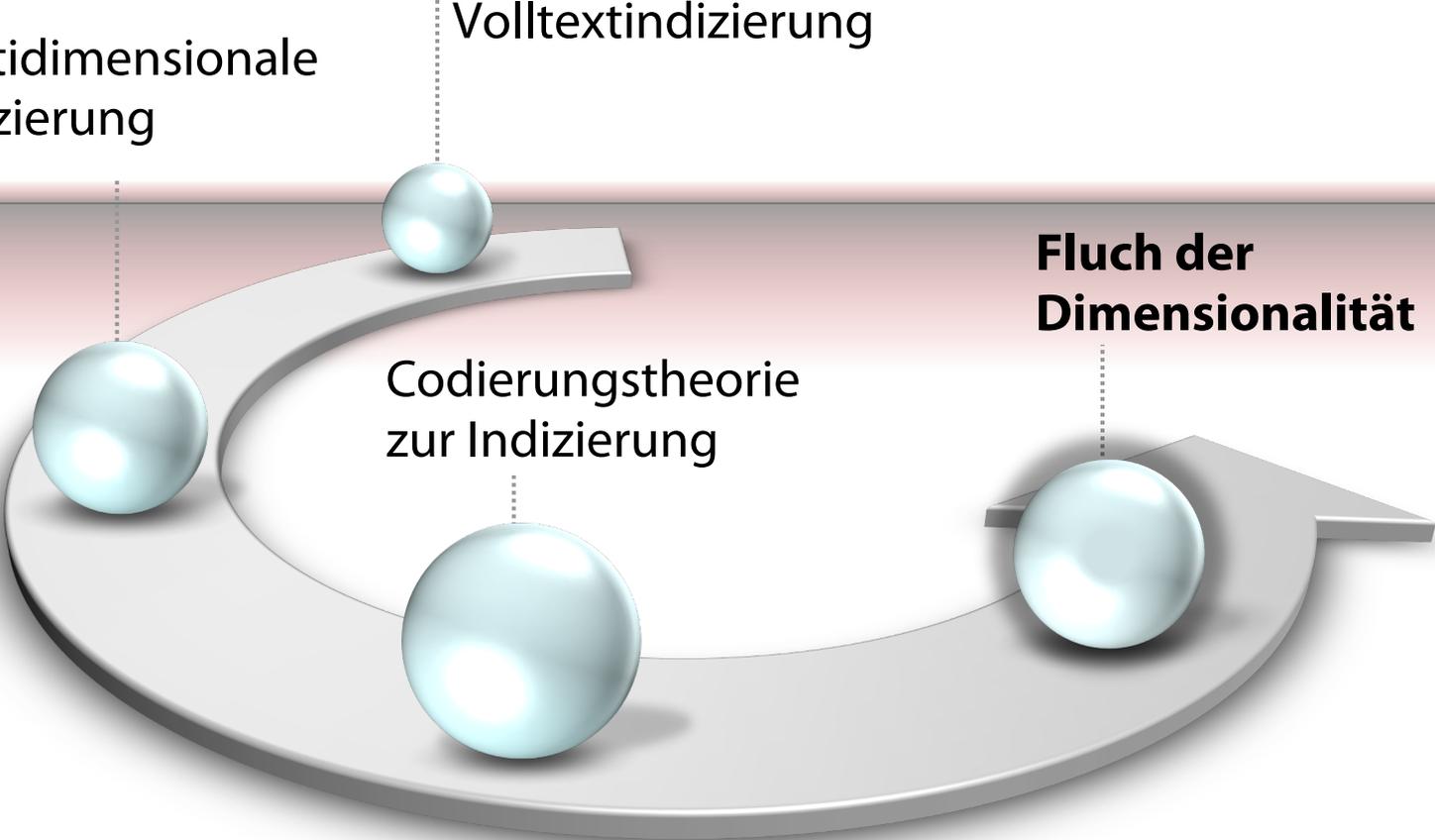
Von der Volltextsuche zur multidimensionalen Indizierung

Multidimensionale Indizierung

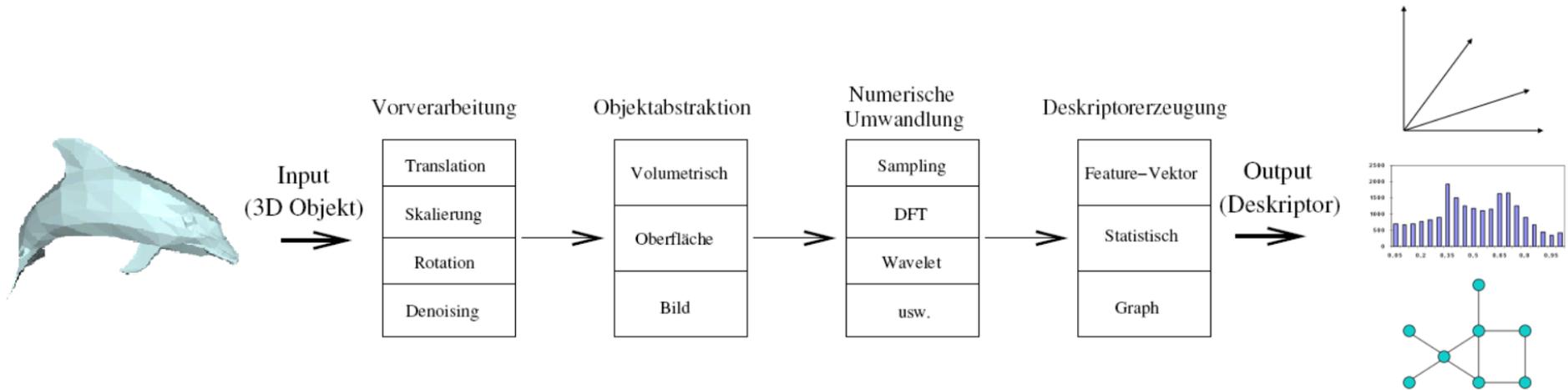
Volltextindizierung

**Fluch der Dimensionalität**

Codierungstheorie zur Indizierung



# Anwendungen: Multimedia-Datenbanken

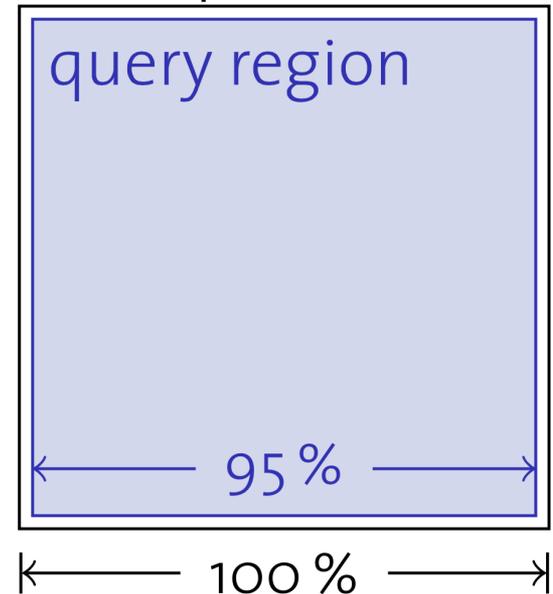


- Inhaltsbasierte Suche
- Viele Merkmalsvektoren
- Hochdimensionale Räume

# 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

data space



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

# Multidimensionale Indizierung: Überblick

