
Non-Standard-Datenbanken

Multidimensionale Indizierung

Prof. Dr. Ralf Möller

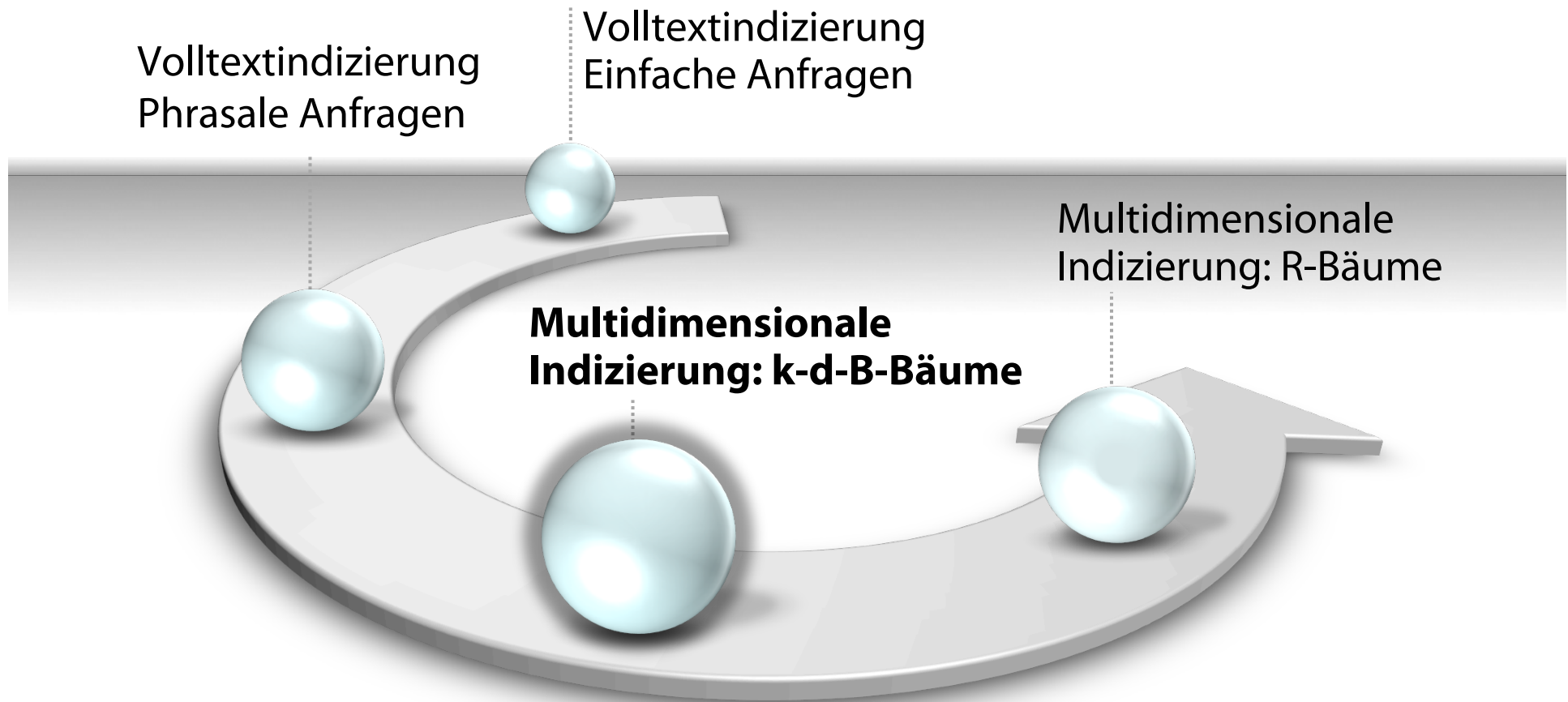
Universität zu Lübeck

Institut für Informationssysteme



Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung



Danksagung

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

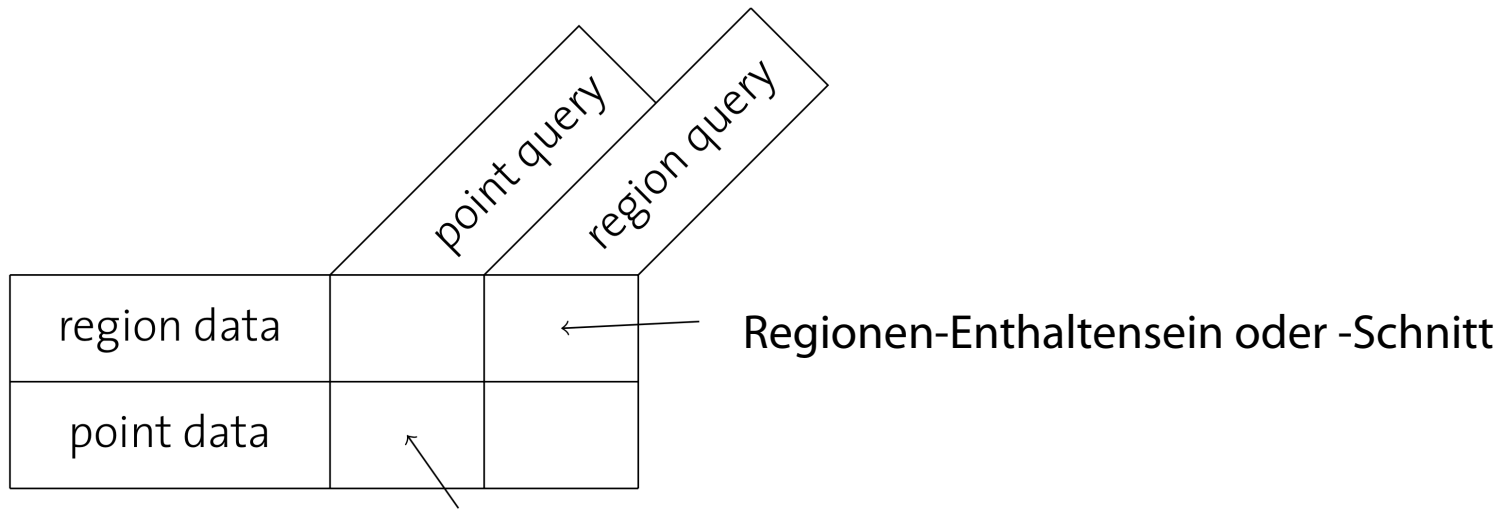
Mehr Dimensionen ...

```
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

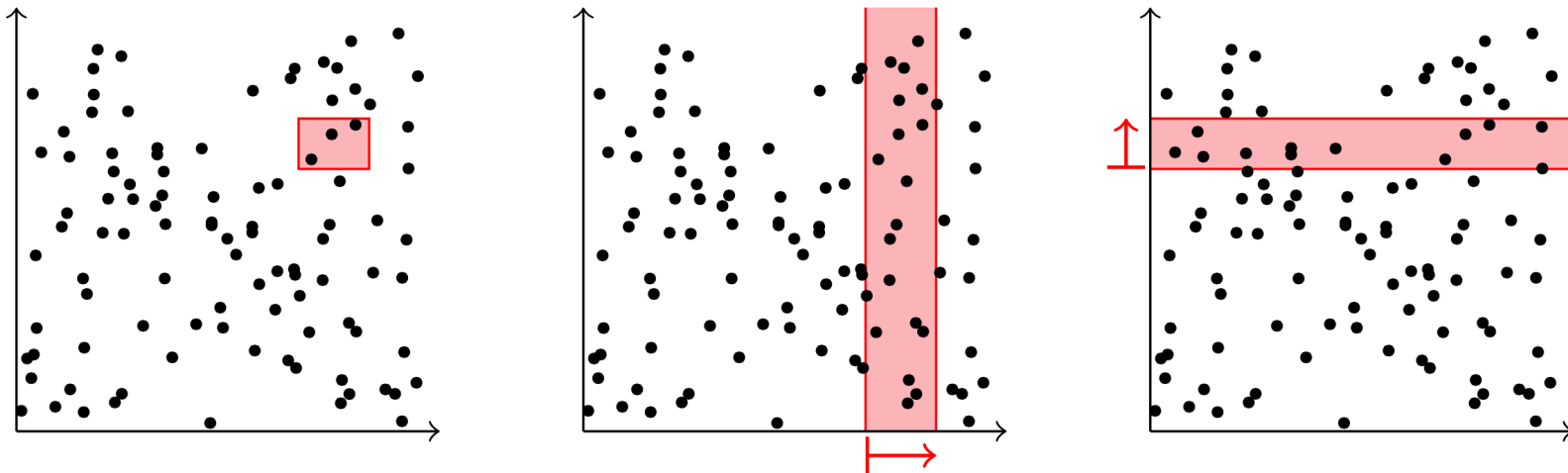


... und es gibt noch viele weitere interessante Anfragetypen für multidimensionale Daten

NB: Anfragen mit Gleichheit lassen sich in eindimensionale Anfragen zerlegen

Können wir nicht einfach B⁺-Bäume verwenden?

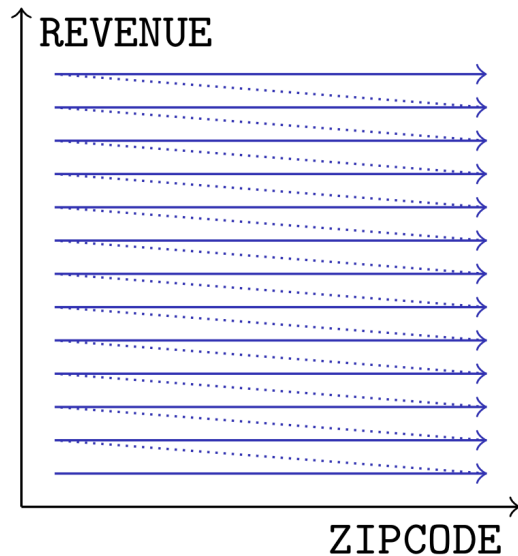
Vielleicht zwei B⁺-Bäume für ZIPCODE und REVENUE?



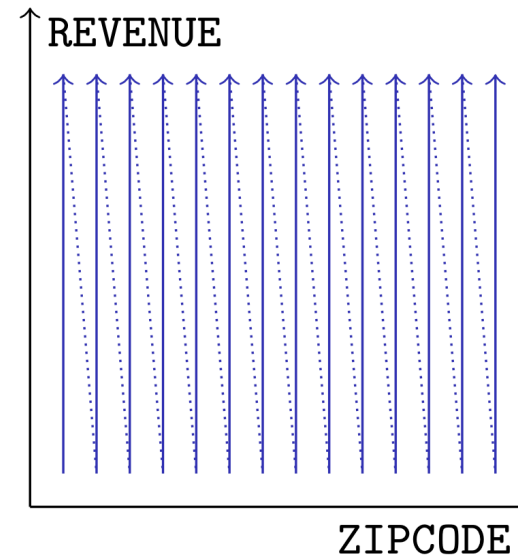
Man kann **pro Dimension** nur über **einen** Index laufen und hat viele **falsche Treffer**

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

Oder zusammengesetzte Schlüssel?



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



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

Gleiche Situation!

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

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

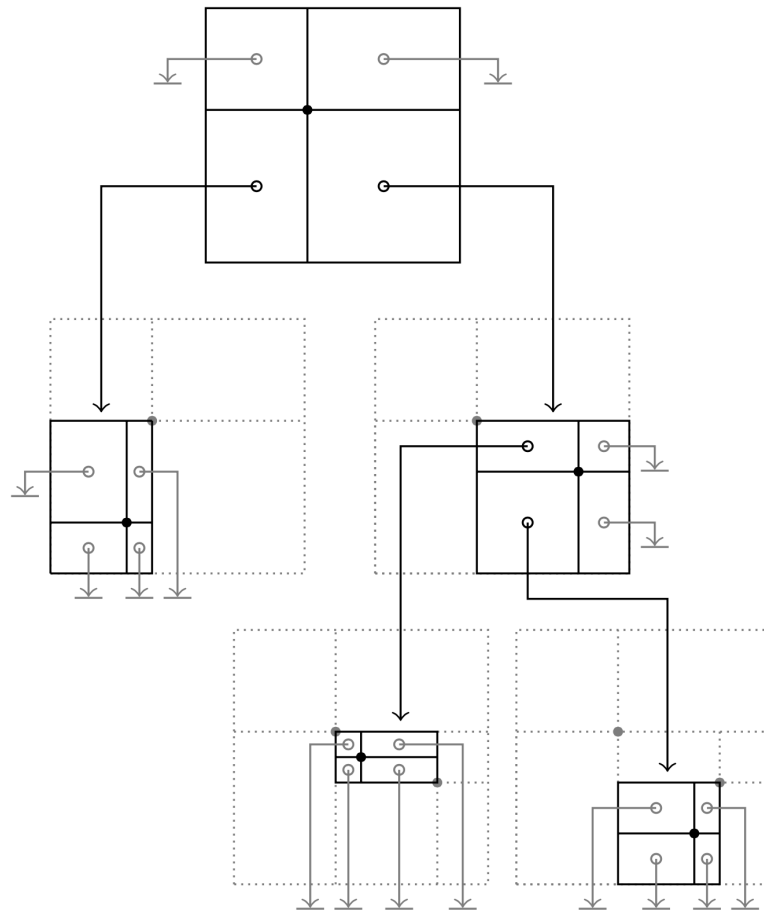
Multidimensionale Indexstrukturen

- B⁺-Bäume unterstützen nur **eindimensionale** Anfragen¹
- 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

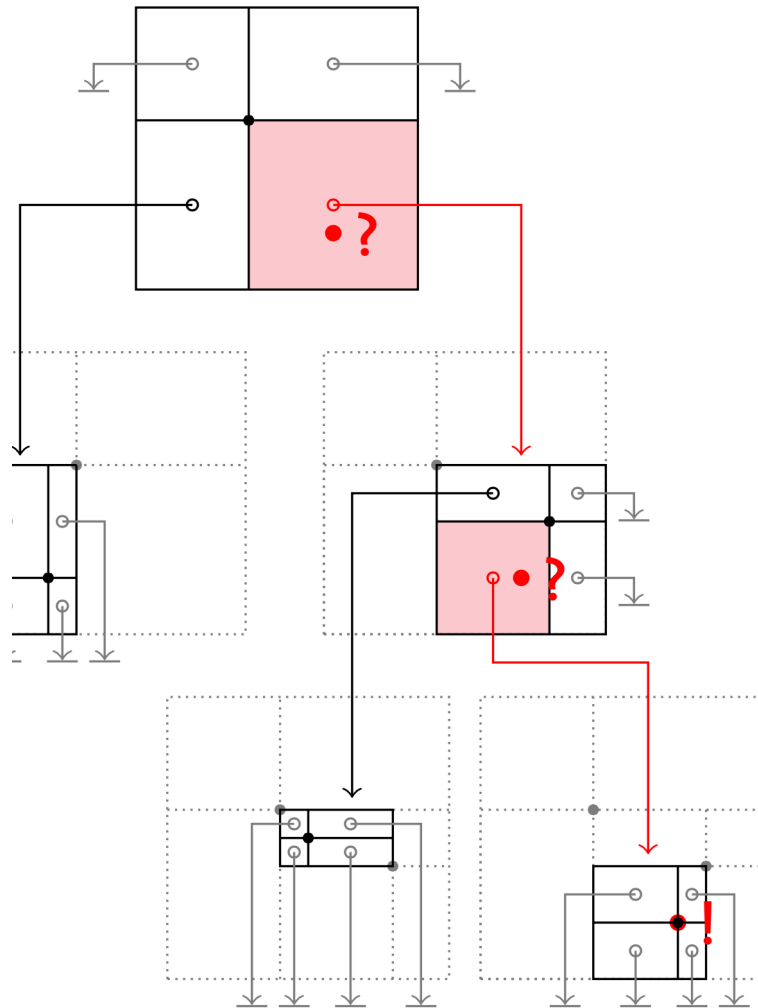
„Binärer“ Suchbaum

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 **Nullzeiger**
- Eine solche Datenstruktur heißt **Punkt-Quad-Baum**

Suche in einem Punkt-Quad-Baum



```

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

```

```

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

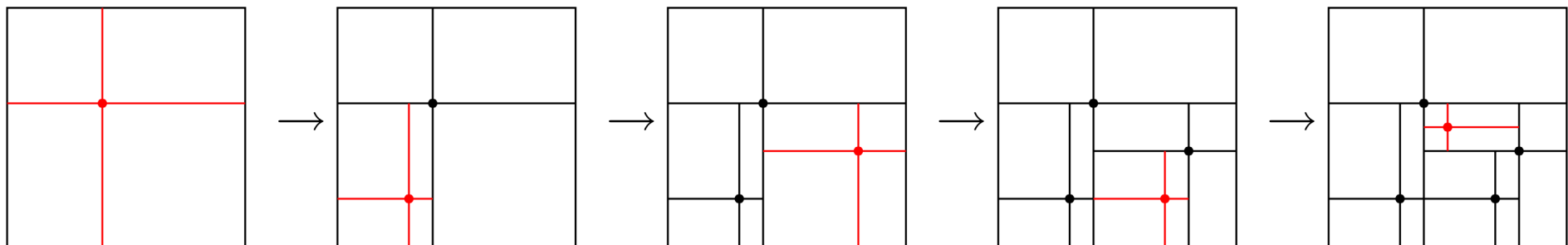
```

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**



Bereichsanfragen

Um eine Bereichsanfrage² 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   | append data point to result ;
4 foreach partition  $P$  in  $node$  that intersects with  $Q$  do
5   |  $node' \leftarrow$  node pointed to by  $P$  ;
6   | r_search ( $Q$ ,  $node'$ ) ;
```

```
1 Function: regionsearch ( $Q$ )
2 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

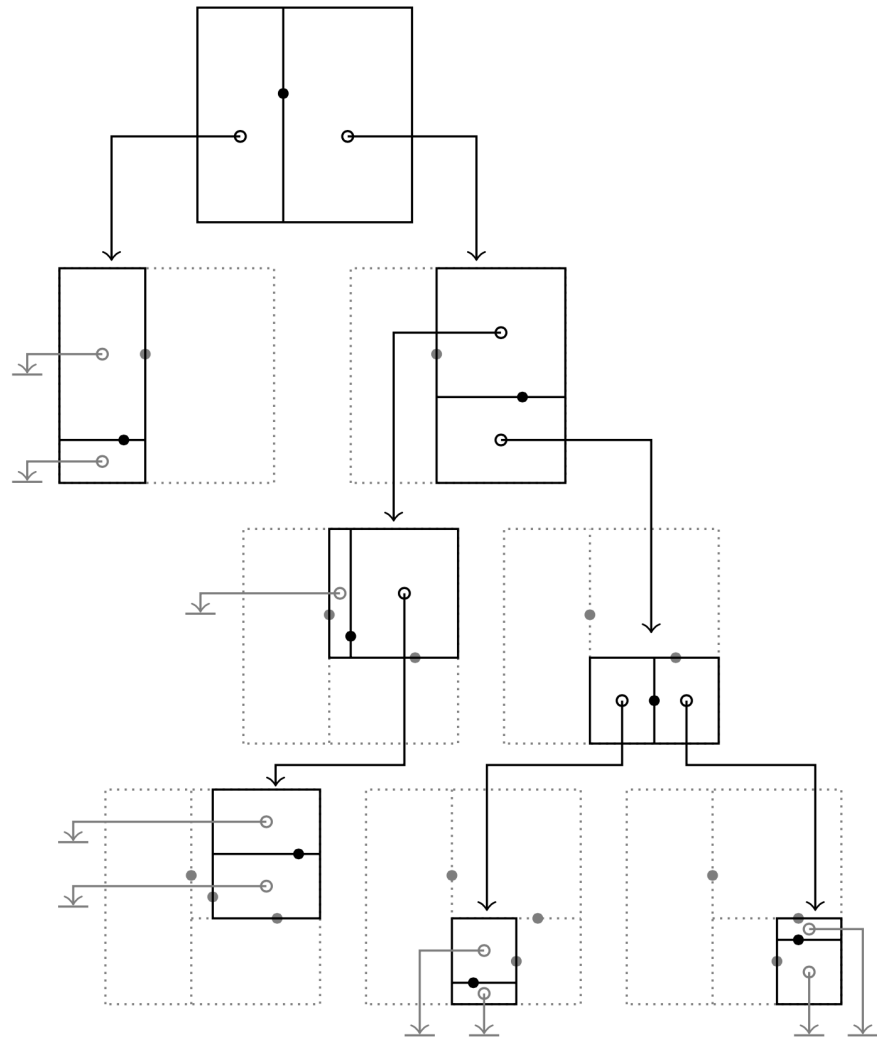
- die Form hängt von der **Einfügereihenfolge** ab
(im **schlimmsten** Fall entsteht eine **verkettete Liste**)
- **Nullzeiger** sind speicher**ineffizient** (ins. bei großem k)

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**

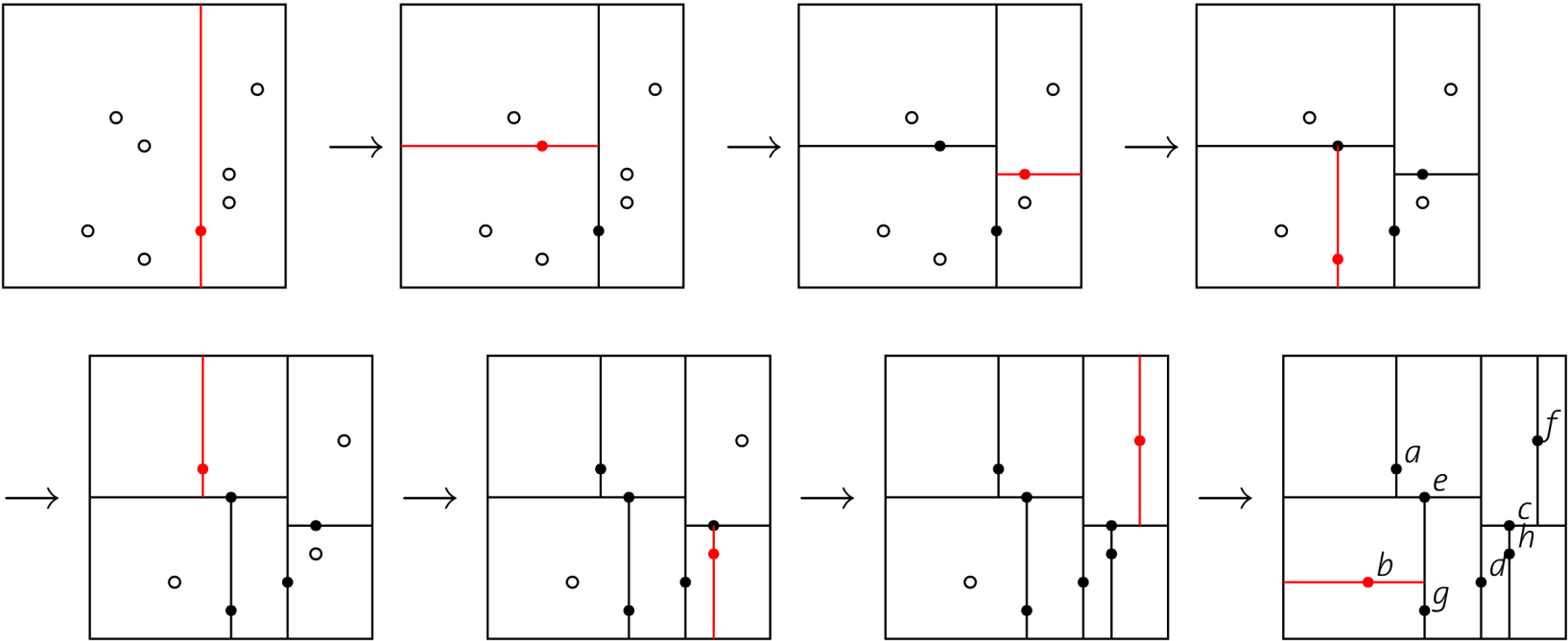
k-d-Bäume

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

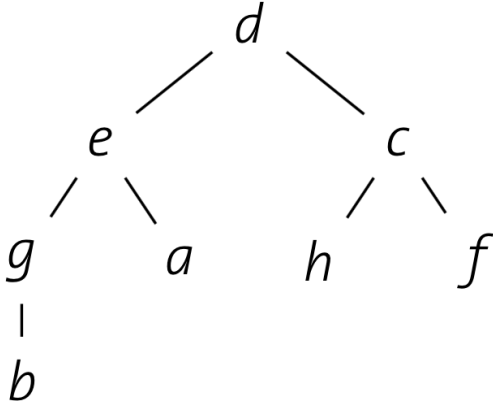
Für eine gegebene Punktmenge kann ein **balancierter** k-d-Baum konstruiert werden³

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

Balancierte *k*-d-Baum-Konstruktion



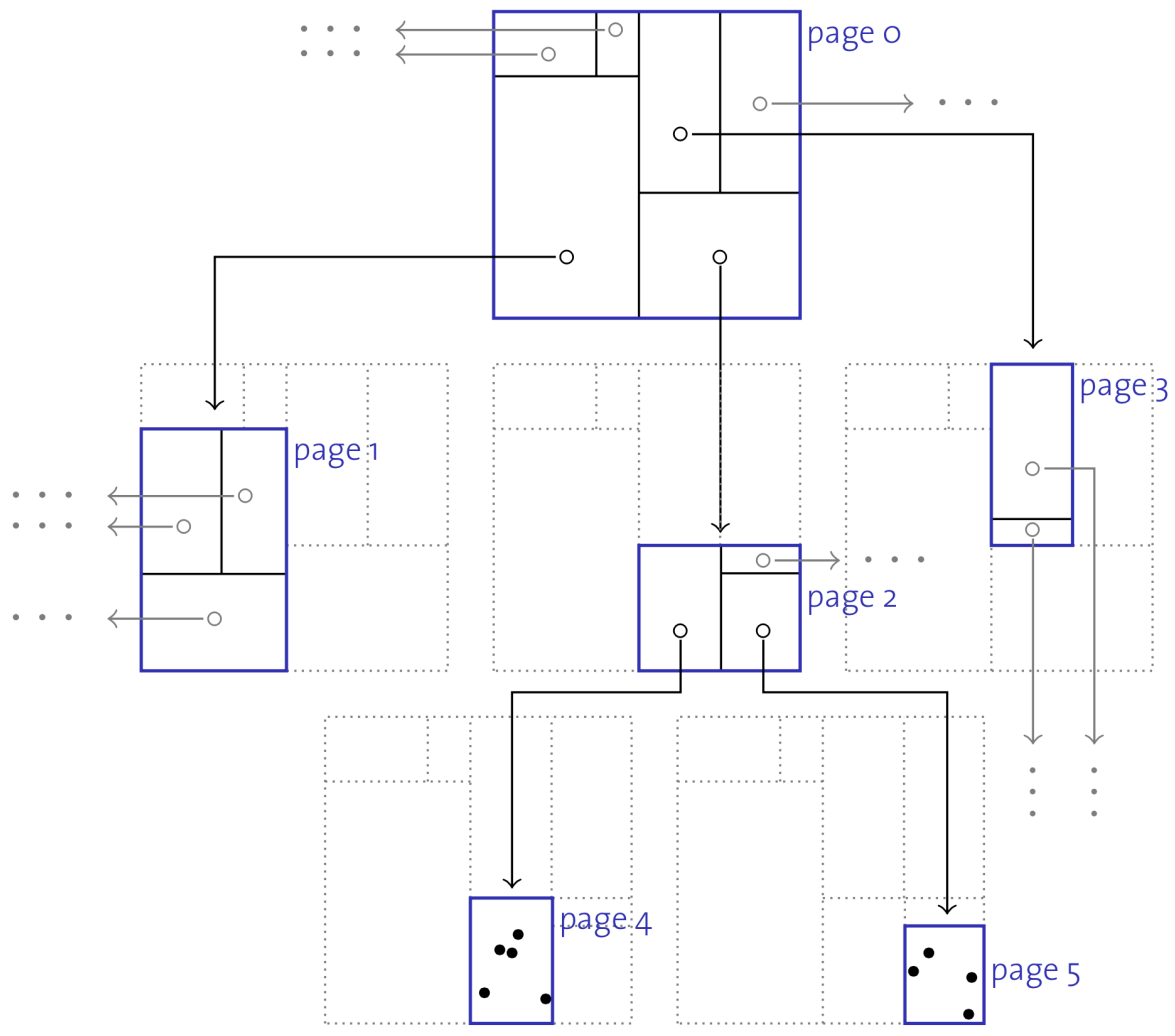
Ergebnis



k-d-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-Bäume: Zentrale Idee



Regionenseiten

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

Punktseiten

- enthalten Einträge $\langle \text{point, rid} \rangle$
- \rightarrow 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

Denkaufgabe: Vektorfeld

Wie sollen wir k nächste Nachbarn bestimmen?



Operationen auf k -d-B-Bäumen

- Beim **Einfügen** wird der Baum **balanciert** wie beim **B⁺-Baum**
 - Füge Eintrag $\langle \text{region, pageID} \rangle$ ($\langle \text{point, rid} \rangle$) in eine Regionenseite (Punktseite) ein, sofern **genügend Platz** vorhanden
 - **Sonst: Splitte** Seite auf

Aufsplittung einer Punktseite

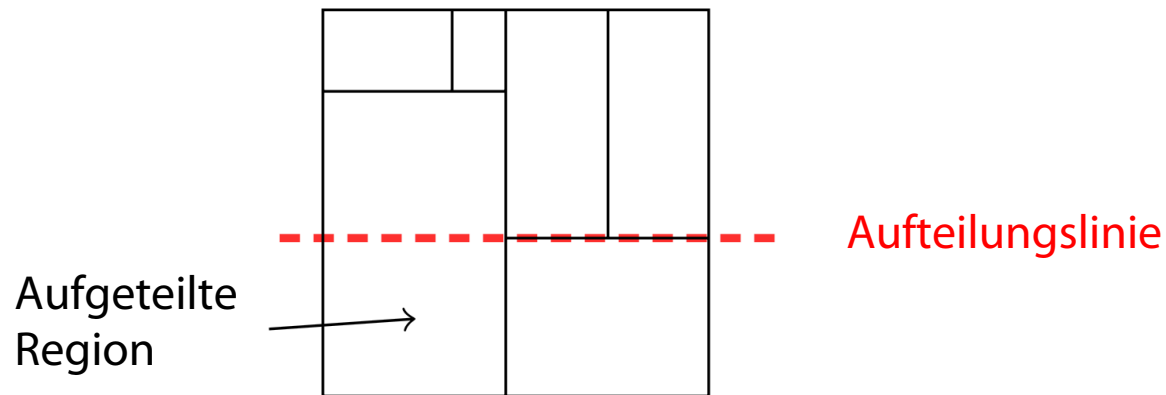
Aufteilung einer Seite p

- 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**

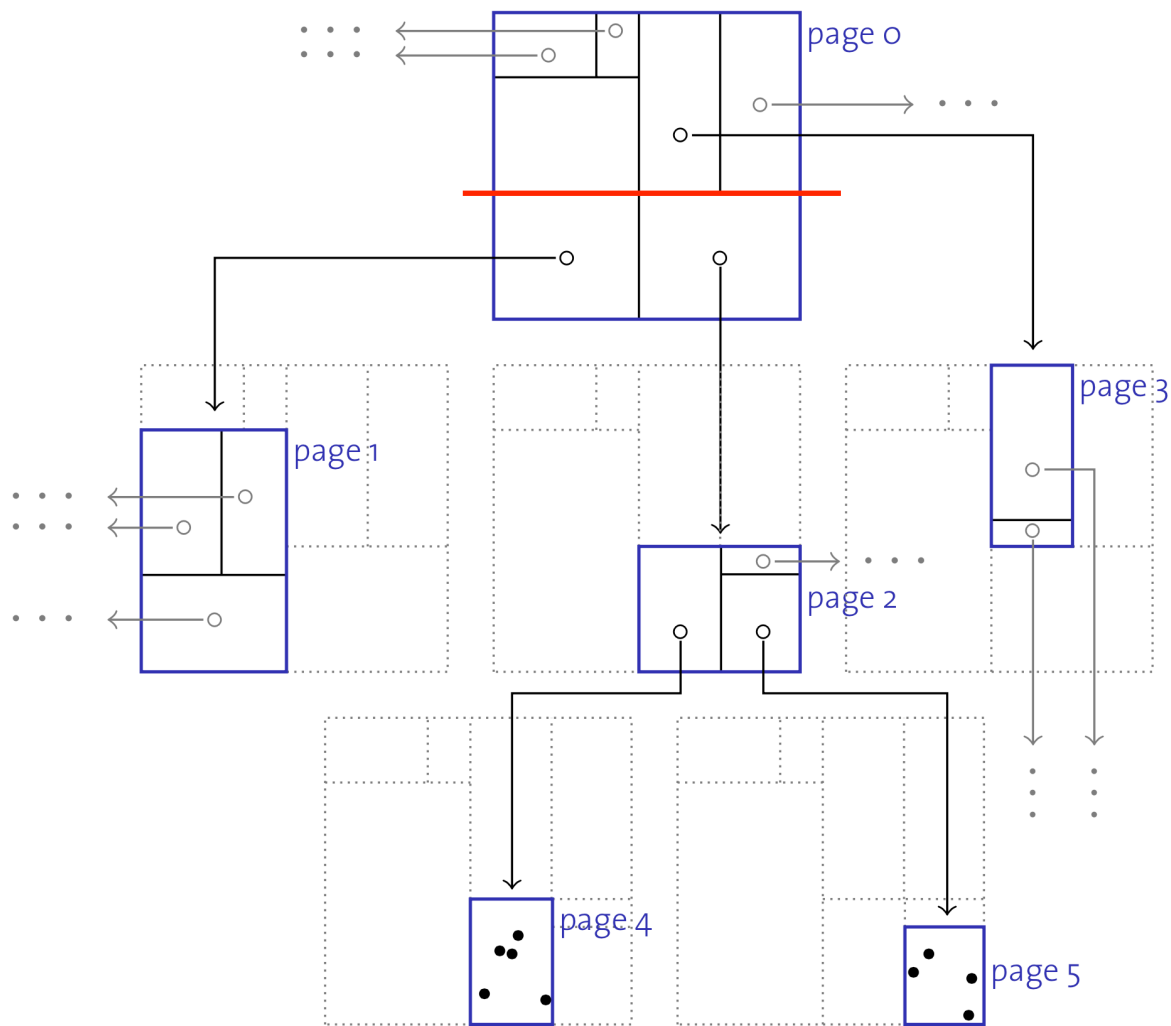
Aufspaltung einer Regionenseite

- Aufspaltung einer **Punktseite** und Verschiebung der Datenpunkt ist recht **einfach**
- Im Falle einer **Regionenaufspaltung** können bei einigen **Regionen auf beiden** Seite des Splitts liegen



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

Beispiel noch einmal



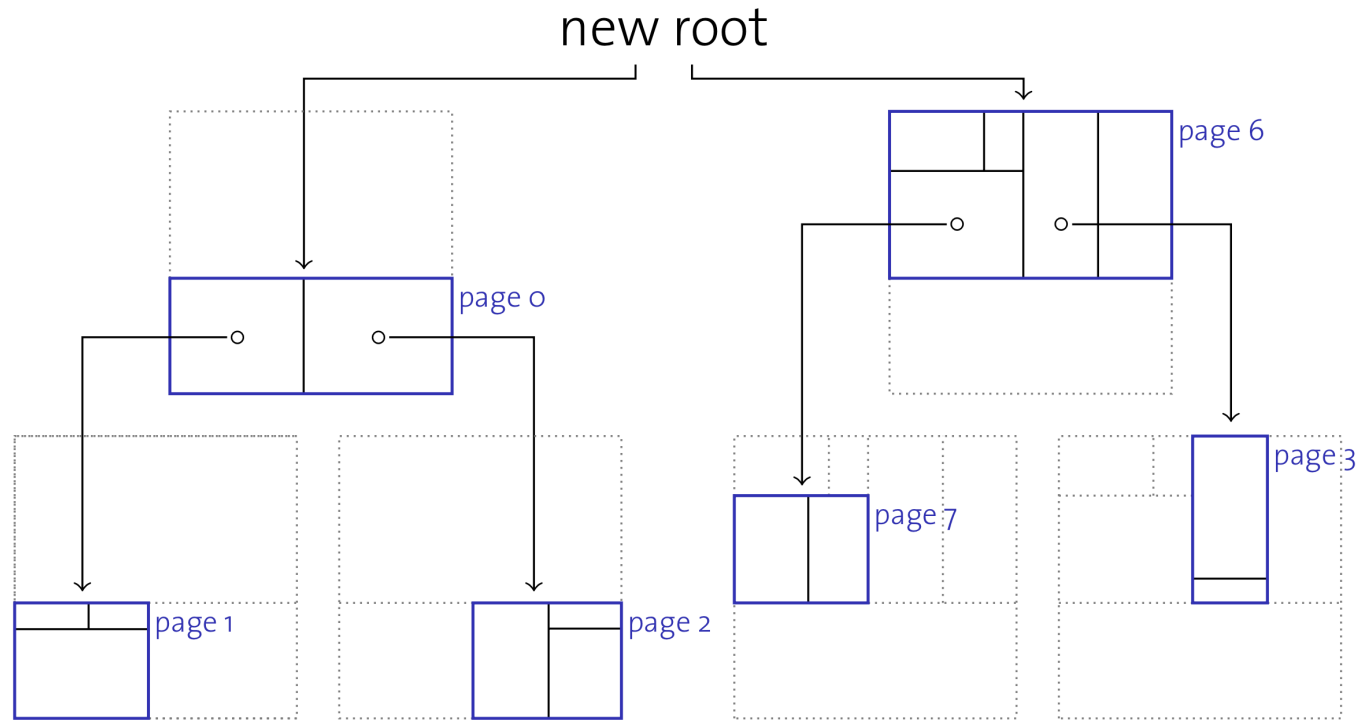
Regionenseiten

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

Punktseiten

- enthalten Einträge $\langle \text{point, rid} \rangle$
- \rightarrow 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-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:

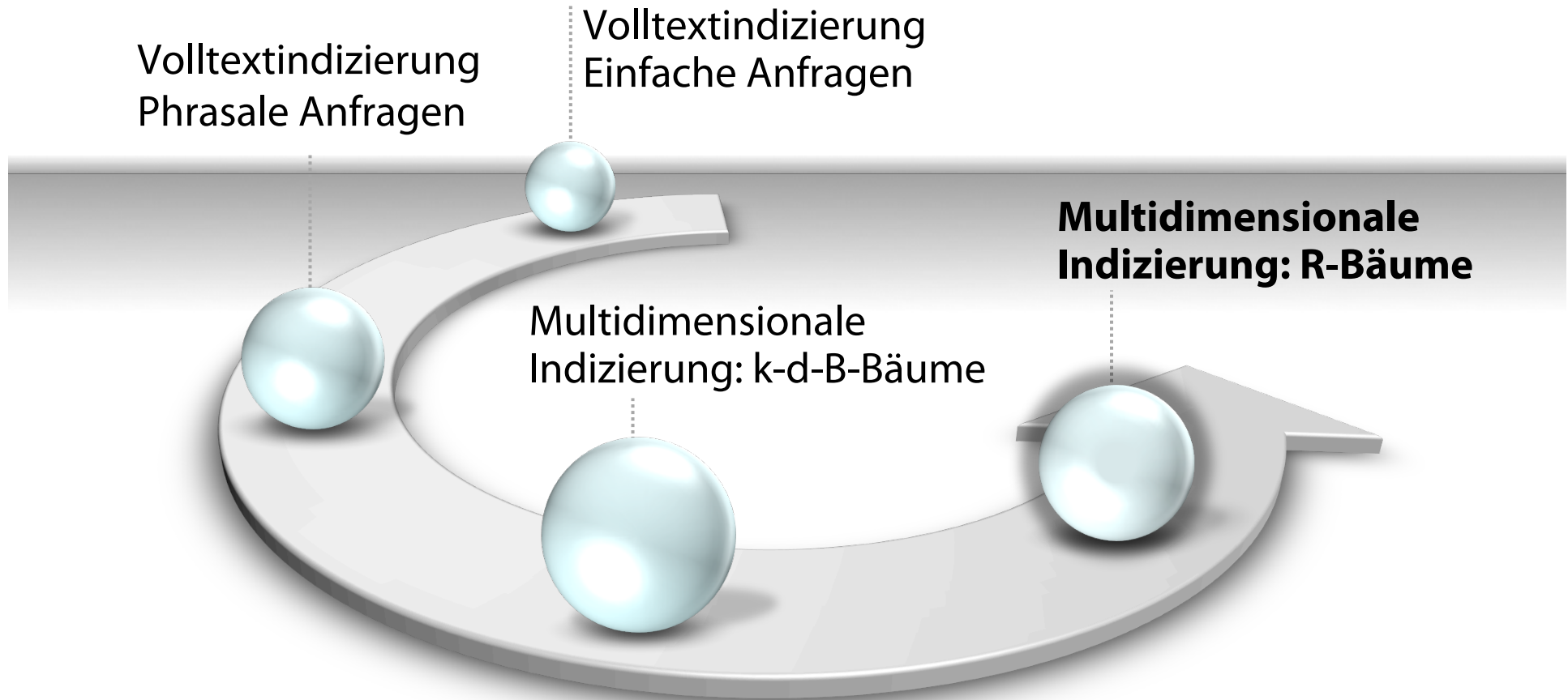
- **Keine Regionendaten**
- **Löschoperationen nicht** (dynamisch) **unterstützt**

Datenraum wird partitioniert, so dass

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

Non-Standard-Datenbanken

Von semistrukturierten Datenbanken zur Volltextsuche



R-Bäume

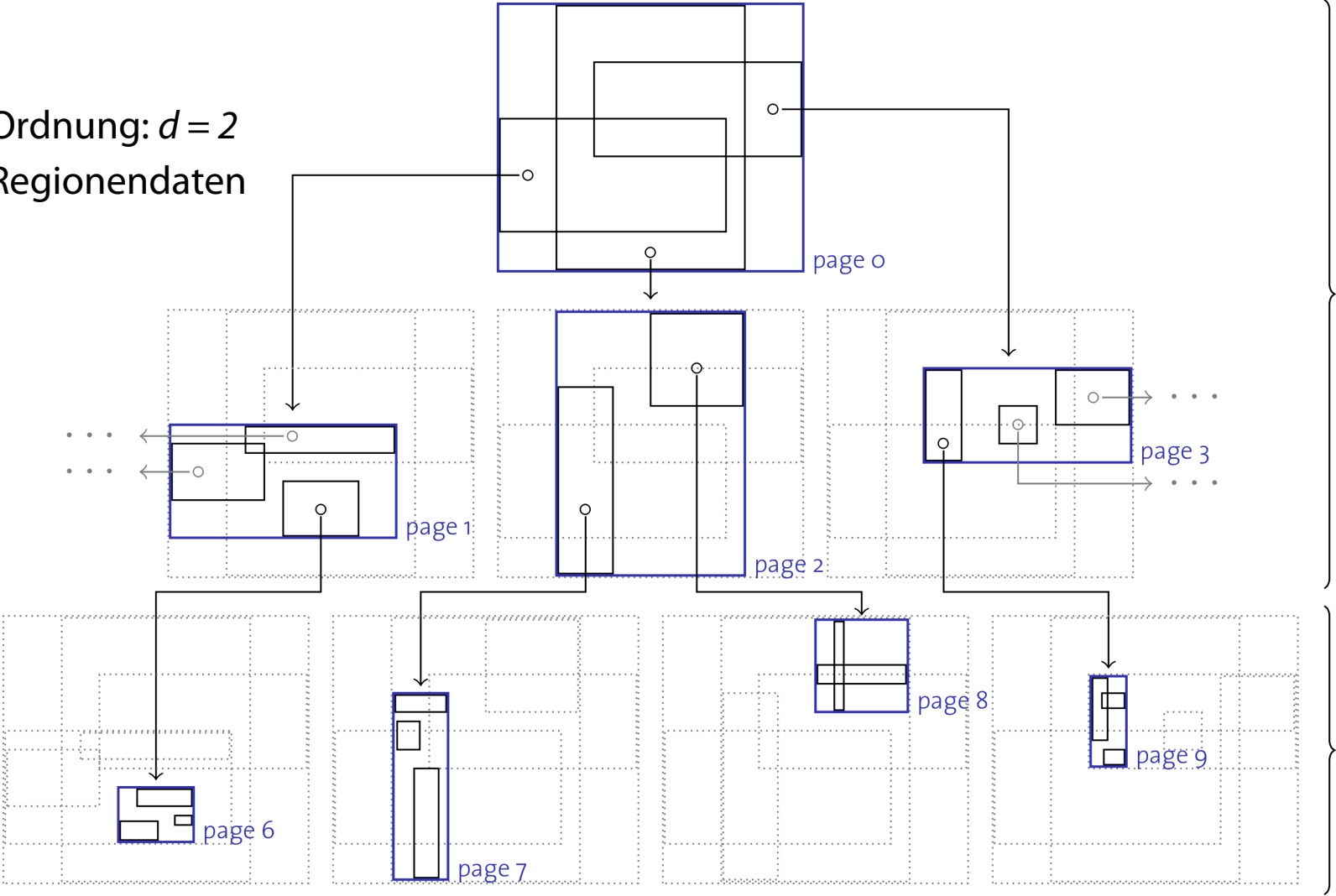
Regionen können sich in dieser Struktur überlappen

- Innere Knoten enthalten $\langle region, pageID \rangle$ Einträge, Blattknoten enthalten Einträge der Form $\langle region, rdf \rangle$, 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

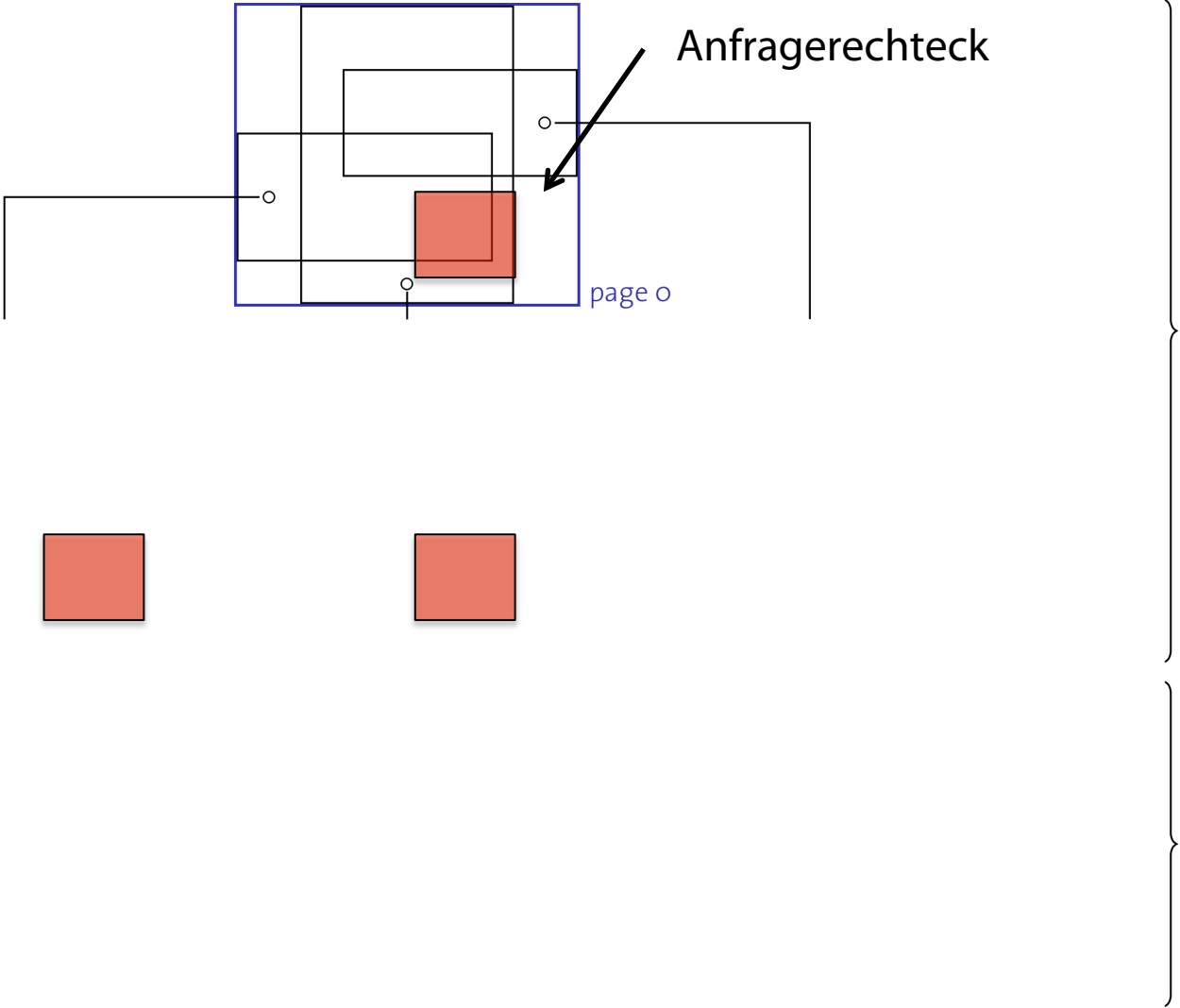


Innere Knoten

Blattknoten

R-Baum: Regionen-anfrage (Schnitt)

- Ordnung: $d = 2$
- Regionendaten



Innere Knoten

Blattknoten



R-Baum: Suchen und Einfügen

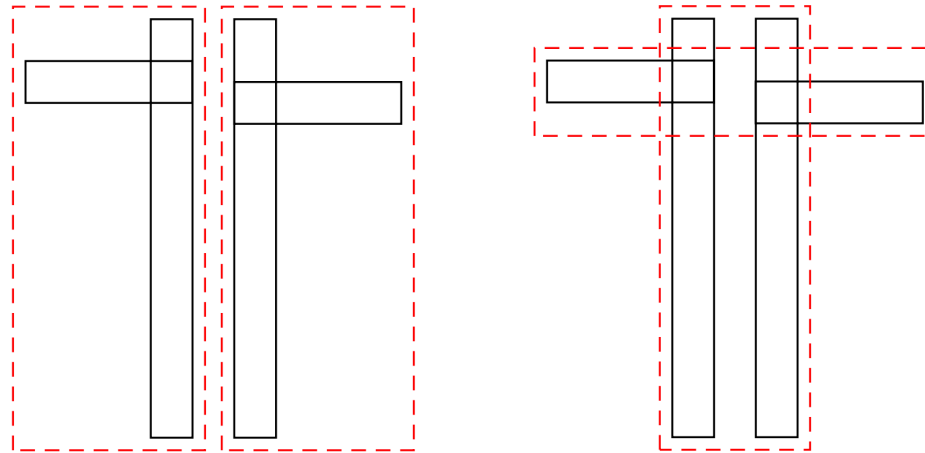
Während der **Suche** müssen ggf. mehrere Kinder betrachtet werden (gilt für Punkt- **und** Regionenanfragen)

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**

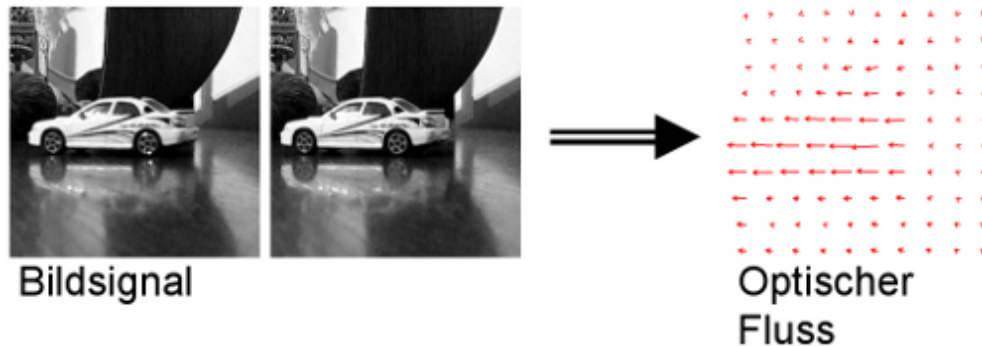
1. Falls ein Knoten n zu leer wird (weniger als d Einträge nach einer Löschoperation) wird der Knoten gelöscht
2. Und die Einträge werden auf andere Knoten verteilt

Der erste Schritt kann zur Löschung des Elternknoten führen

- Löschen ist eine aufwändige Operation in R-Bäumen

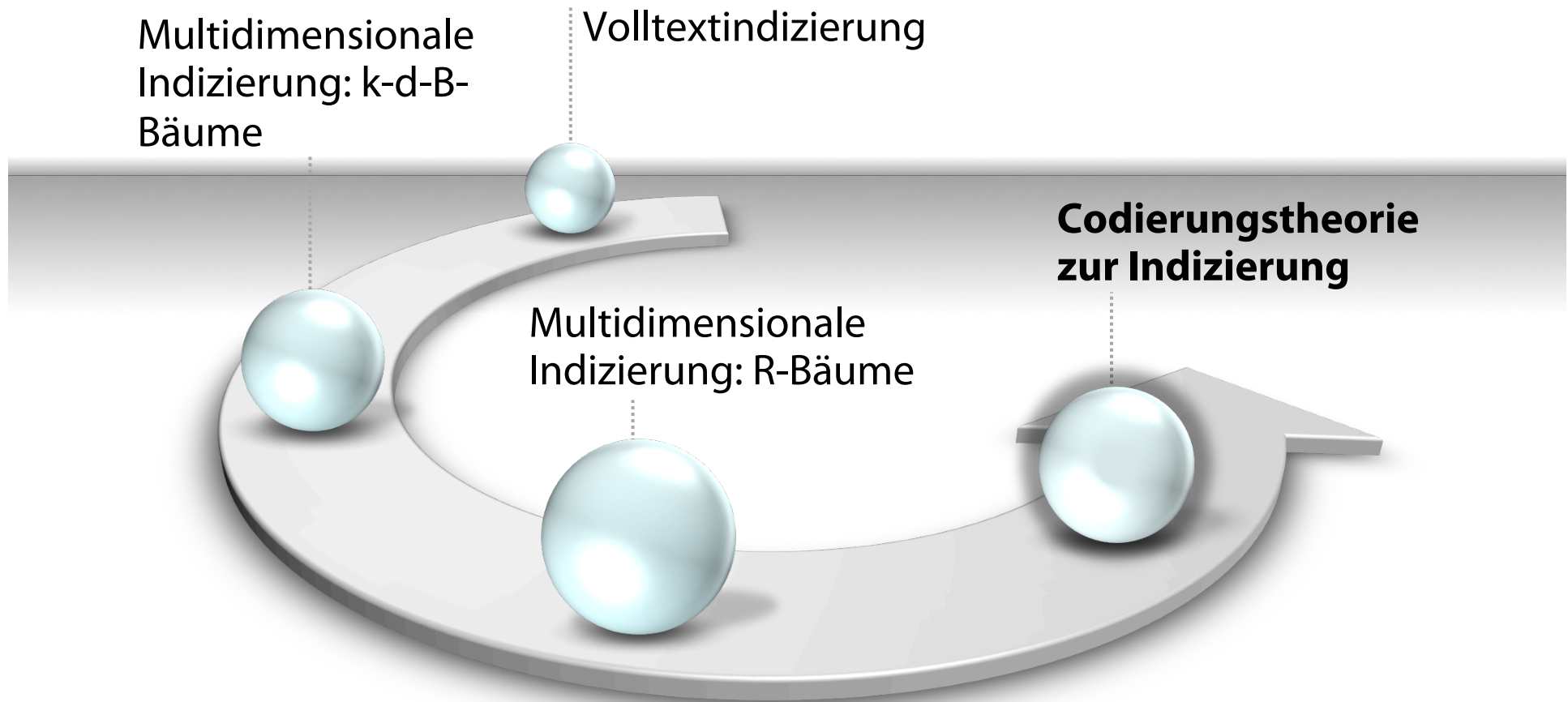
Anwendung von R-Bäumen

- 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



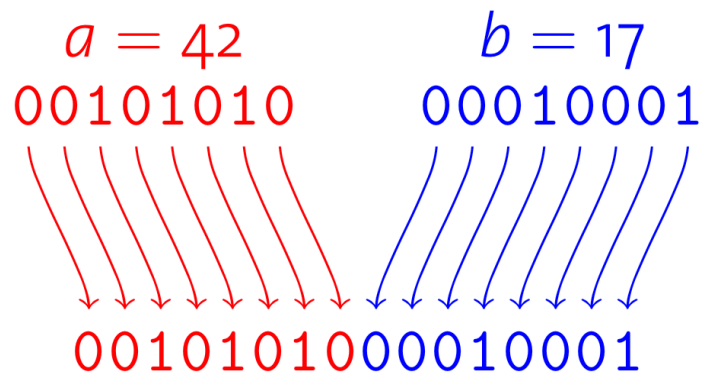
Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung

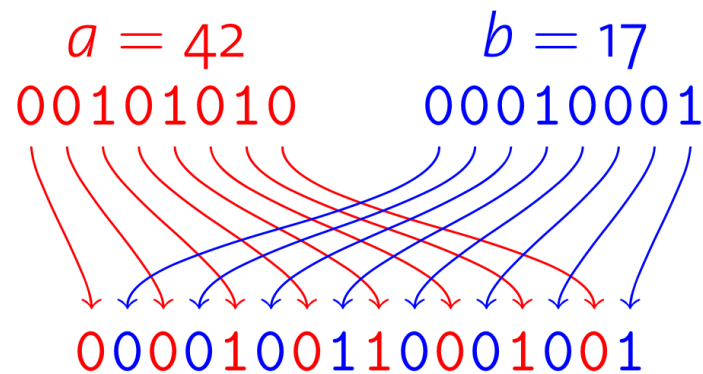


Bit-Verschränkung

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

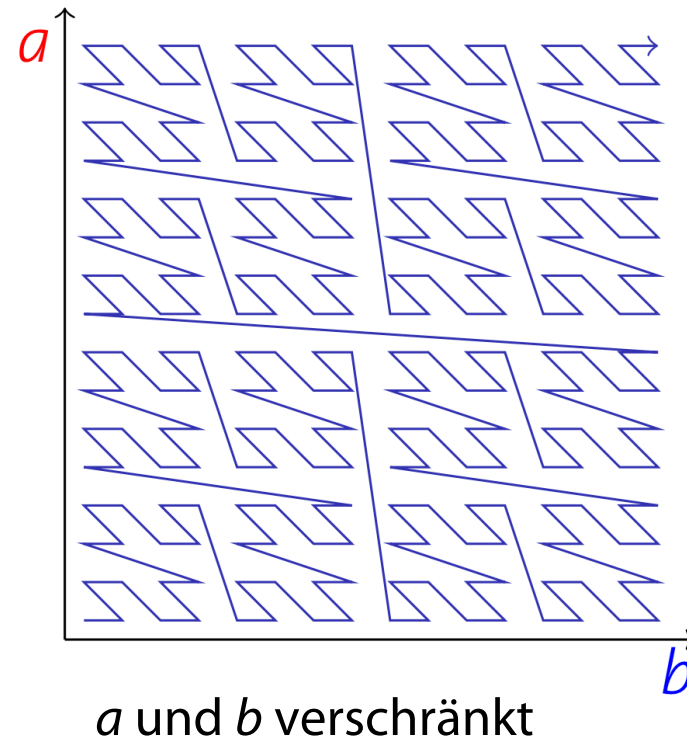
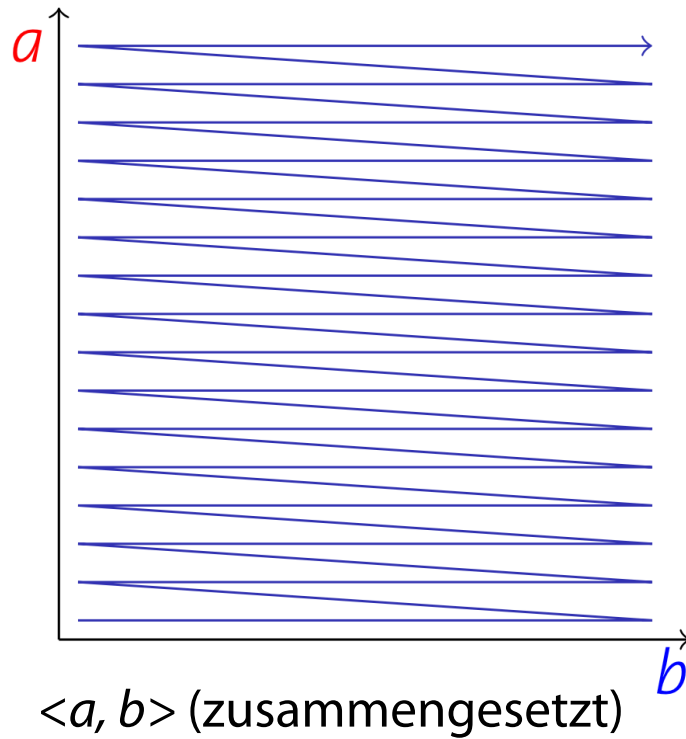


$\langle a, b \rangle$ (zusammengesetzt)



a und b verschränkt

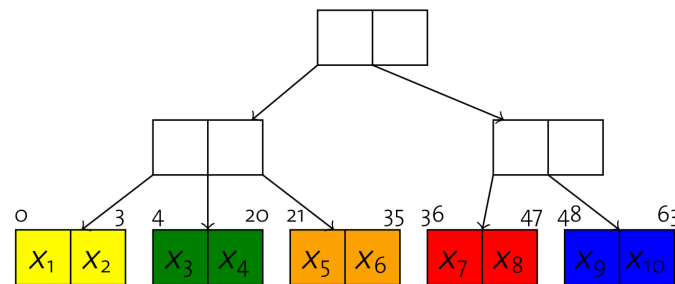
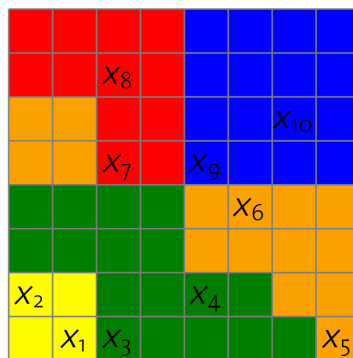
Z-Ordnung



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

B⁺-Bäume über Z-Ordnungen

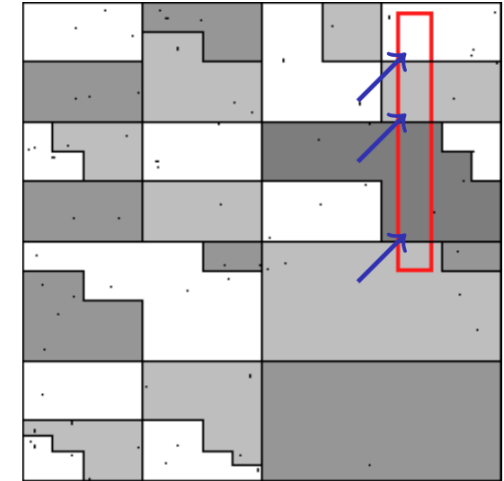
- Verwendung eines **B⁺-Baumes** um Z-Kodes des multidimensionalen Raums zu indizieren
- Blatt im B⁺-Baum beschreibt **Intervall** im **Z-Raum**
- Jedes dieser Intervall beschreibt eine **Region** im multidimensionalen Datenraum



- 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

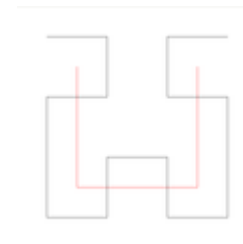
Nach jeder verarbeiteten Seite erfolgt Index-Rescan um neue Seite zu finden, die sich mit Anfragerechteck Q schneidet



```
1 Function: ub_range ( $Q$ )
2  $cur \leftarrow z(Q_{\text{bottom, left}})$ ;
3 while true do
4     // search  $B^+$ -tree page containing  $cur$  ( $\nearrow$  slide 55)
5      $page \leftarrow \text{search}(cur)$ ;
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  $Q_{\text{top, right}}$  then
10            break;
11        // compute next Z-address using  $Q$  and data on current page
12         $cur \leftarrow \text{get\_next\_z\_address}(Q, page)$ ;
```

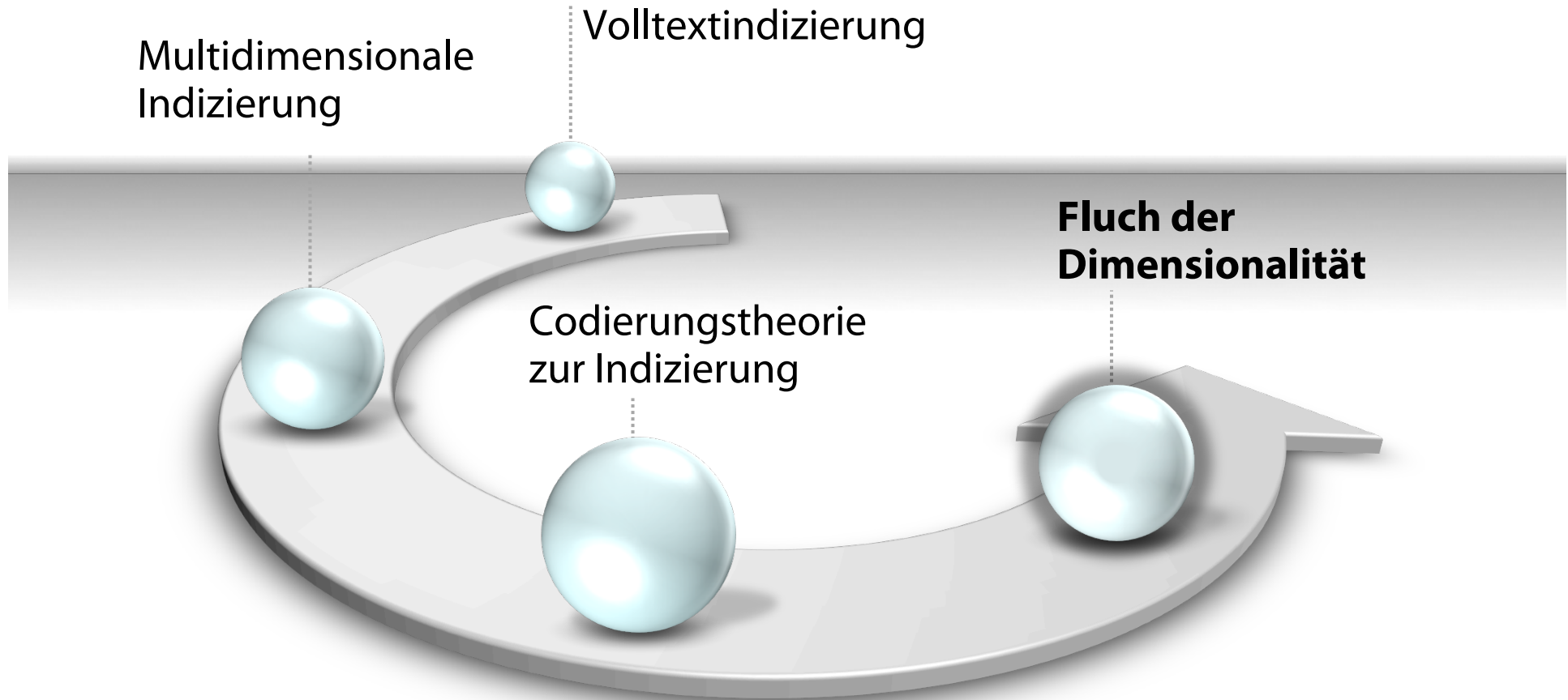
UB-Bäume – Diskussion

- 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)

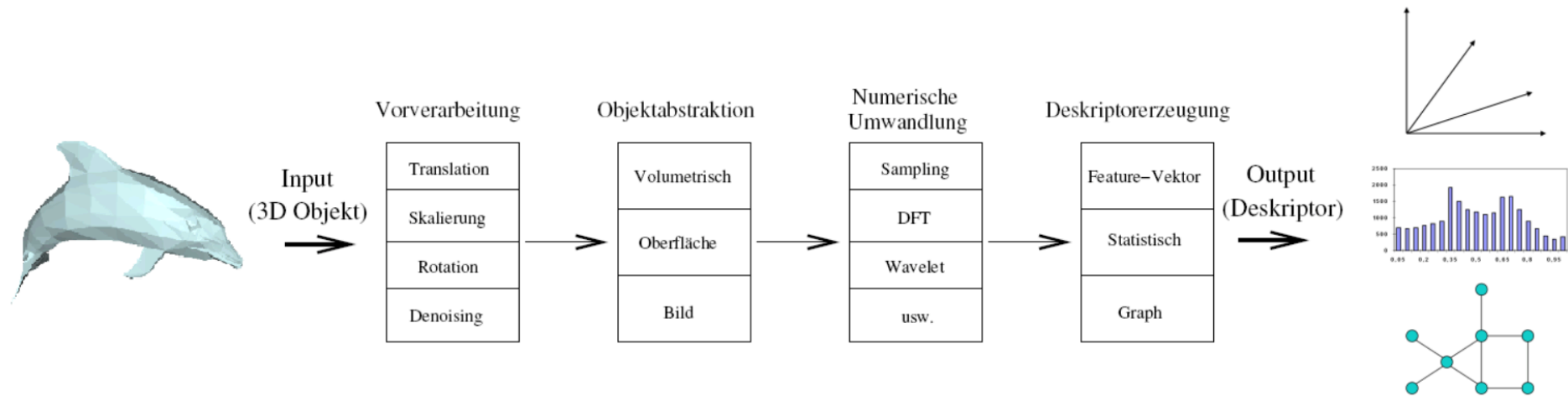


Non-Standard-Datenbanken

Von der Volltextsuche zur multidimensionalen Indizierung



Anwendungen: Multimedia-Datenbanken

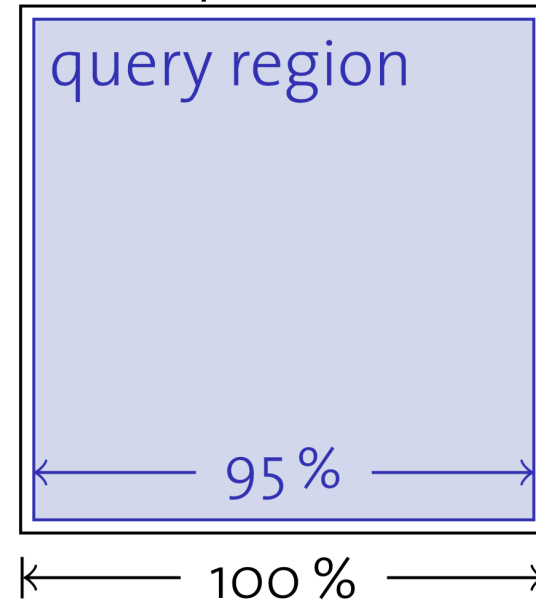


- Inhaltsbasierte Suche
- Große Merkmalsvektoren
- Hochdimensionale Räume

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

