6. Mehrdimensionale Zugriffspfade

- **Klassifikation der Anfragen**
- **Grundprobleme**
  - Organisation räumlicher Daten
  - Erhaltung der Topologie (Clusterbildung)
  - Objektdarstellung
- **Organisation der Datensätze**
  - Quad-Tree
  - Multi-Key-Hashing
- **Organisation des umgebenden Datenraums**
  - k-d-Baum und k-d-B-Baum
  - Grid-File
- **Zugriffspfade für ausgedehnte räumliche Objekte**
  - R-Baum
  - R^+-Baum
- **Objekttransformation**
  am Beispiel eindimensionaler Intervalle

Klassifikation der Anfragetypen

Definitionen:

- Eine Datei ist eine Sammlung von N Sätzen des Typs $R = (A_1, ..., A_n)$, wobei jeder Satz ein Punktobjekt durch ein geordnetes n-Tupel $t = (a_1, a_2, ..., a_n)$ von Werten darstellt. Die Attribute $A_1, ..., A_k$ ($k \leq n$) seien Schlüssel.
- Eine Anfrage Q spezifiziert einige Bedingungen, die von den Schlüsselwerten der Sätze in der Treffermenge erfüllt sein müssen.
- <u>Schnittbildende Anfragen</u> (*intersection queries*): sich qualifizierende Objekte überlappen mit dem Anfragebereich
- <u>Enthaltenseins- oder Umschließungsanfragen</u> (*containment , enclosure queries*): sich qualifizierende Objekte sind ganz im Anfragebereich enthalten oder enthalten den Anfragebereich vollständig

Klassifikation der schnittbildenden Anfragen

1. ***Exakte Anfrage*** *(exact match query)*:
   spezifiziert für jeden Schlüssel einen Wert

   $$Q = (A_1 = a_1) \wedge (A_2 = a_2) \wedge \ ... \wedge (A_k = a_k)$$

2. ***Partielle Anfrage*** *(partial match query):*
   spezifiziert s < k Schlüsselwerte

   $$Q = (A_{i1} = a_{i1}) \wedge (A_{i2} = a_{i2}) \wedge \ ... \wedge (A_{is} = a_{is})$$
   $$\text{mit } 1 \leq s < k \text{ und } 1 \leq i_1 < i_2 < ... < i_s \leq k$$

3. ***Bereichsanfrage*** *(range query)*:
   spezifiziert einen Bereich $r_i = [l_i \leq a_i \leq u_i]$ für jeden Schlüssel A_i

   $$\begin{aligned} Q &= (A_1 = r_1) \wedge \ ... \wedge (A_k = r_k) \\ &\equiv (A_1 \geq l_1) \wedge (A_1 \leq u_1) \wedge \ ... \wedge (A_k \geq l_k) \wedge (A_k \leq u_k) \end{aligned}$$

4. ***Partielle Bereichsanfrage*** *(partial range query):*
   spezifiziert für s < k Schlüssel einen Bereich

   $$Q = (A_{i1} = r_{i1}) \wedge \ ... \wedge (A_{is} = r_{is})$$
   $$\text{mit } 1 \leq s < k \text{ und } 1 \leq i_1 < ... < i_s \leq k \text{ und } r_{ij} = [l_{ij} \leq a_{ij} \leq u_{ij}],\ 1 \leq j \leq s$$

Klassifikation der Anfragetypen (2)

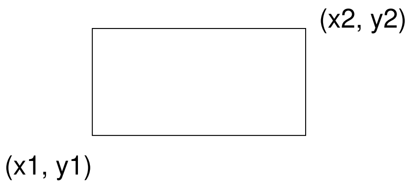
- **Allgemeine Bereichsanfrage**
  - genauer Bereich $[l_i = a_i = u_i]$
  - unendlicher Bereich $[-\infty \leq a_i \leq \infty]$

➥ Alle 4 Fragetypen der schnittbildenden Anfragen lassen sich als allg. Bereichsfrage ausdrücken

- **Klassifikation der Enthaltenseins-Anfragen**
  - ***Punktanfrage*** (*point query*): Gegeben ist ein Punkt im Datenraum D. Finde alle Objekte, die ihn enthalten.
  - ***Gebietsanfrage*** (*region query*): Gegeben ist ein Anfragegebiet. Finde alle Objekte, die es schneiden (es umschließen, in ihm enthalten sind).

- **Beispiele: Suche nach Rechtecken**

RECTANGLES (x1, y1, x2, y2)



a) Bestimmung aller Rechtecke, die den Punkt (2,5) enthalten

```
SELECT   x1, y1, x2, y2
FROM     RECTANGLES
WHERE    x1 <= 2 AND  x2 >= 2 AND  y1 <= 5  AND  y2 >= 5
```

b) Bestimmung der Rechtecke mit Punkt (1,3) als Eckpunkt links unten

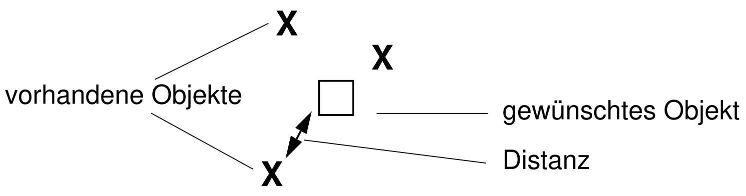
```
SELECT   x1, y1, x2, y2
FROM     RECTANGLES
WHERE    x1 = 1 AND y1 = 3
```

➥ Das sind alles harmlose Fragen, die sogar im Relationenmodell mühelos beantwortet werden können

Klassifikation der Anfragetypen (3)

- **Nächster-Nachbar-Anfragen** (*best match query*, *nearest neighbor query*)
  - gewünschtes Objekt nicht vorhanden

  ➥ Frage nach <u>möglichst ähnlichen</u> Objekten



- "best" wird bestimmt über verschiedene Arten von Distanzfunktionen

- Beispiele:
  - Objekt erfüllt nur 8 von 10 geforderten Eigenschaften
  - Objekt ist durch Synonyme beschrieben

- **Bestimmung des nächsten Nachbar:**

D = Distanzfunktion

B = Sammlung von Punkten im k-dim. Raum

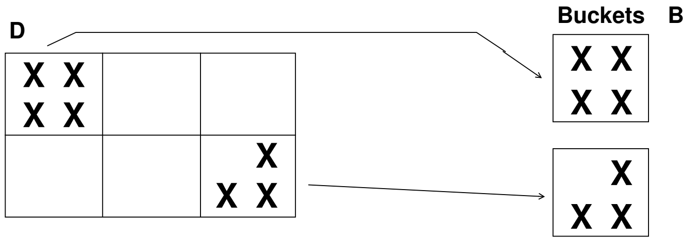
Gesucht: nächster Nachbar von p (in B)

Der nächste Nachbar ist q, wenn

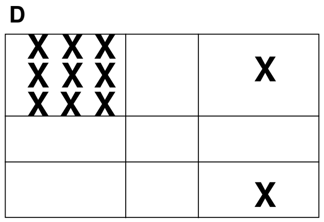
$$(\forall\, r \in B)\ \{r \neq q \ \Rightarrow\ [D(r,p) \geq D(q,p)]\}$$

Grundprobleme

1. **Erhaltung der topologischen Struktur**



2. **Stark variierende Dichte der Objekte**



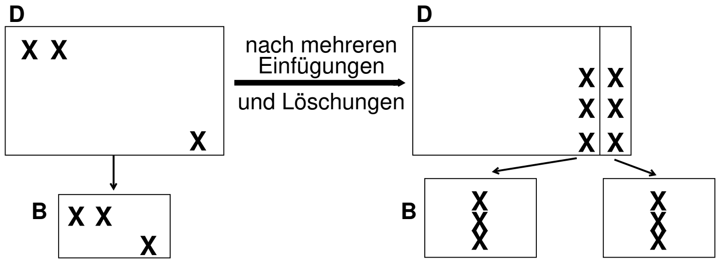
Starke Änderung der räumlichen Belegung über die Zeit

↳ <u>keine</u> regelmäßige Aufteilung von D

↳ gleiche Bucketgröße

3. **Objektdarstellung**
   - Punktobjekte
   - Objekte mit Ausdehnung

4. **Dynamische Reorganisation**

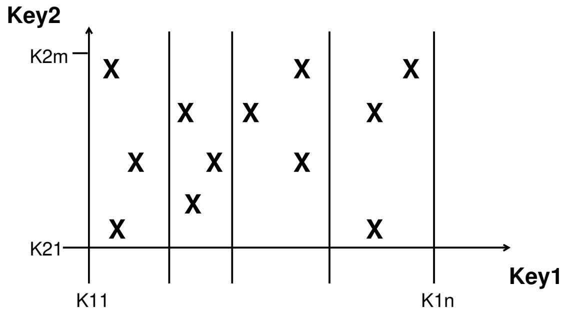


5. **Balancierte Zugriffsstruktur**
   - beliebige Belegungen und Einfüge-/Löschreihenfolgen
   - Garantie eines gleichförmigen Zugriffs
     ➥ 2 oder 3 Externspeicherzugriffe

Nutzung eindimensionaler Zugriffspfade

- **Bisher:**
  Indexierung (Invertierung) einer Dimension, z. B. B*-Baum

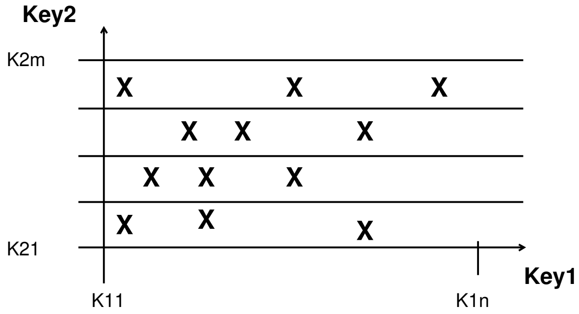
- **Zerlegungsprinzip des Schlüsselraumes beim B*-Baum (2-dim.)**



B*-Baum (Key1)

➥ Partitionierung des Raumes nach Werten von Key1

- **Zusätzlicher B*-Baum (Key2) möglich:**



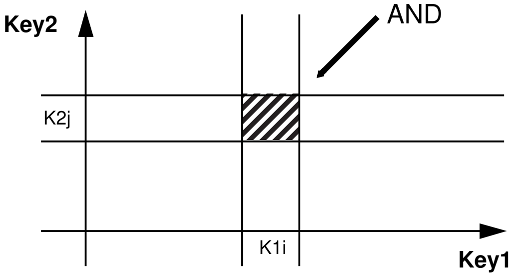
B*-Baum (Key2)

➥ Partitionierung des Raumes nach Werten von Key2

Mehrattributsuche

- **Zugriff nach** $(\text{Key1} = \text{K1i}) \left\{ \begin{matrix} \text{OR} \\ \text{AND} \end{matrix} \right\} (\text{Key2} = \text{K2j})$
  - Zeigerliste für K1i : aus B*-Baum (Key1)
  - Zeigerliste für K2j : aus B*-Baum (Key2)

  ➥ **Mischen + Zugriff auf Ergebnistupel**



  ➥ große Zeigerlisten und Zwischenergebnisse !

- **Simulation des mehrdimensionalen Zugriffs mit <u>einem B*-Baum</u> ?**

Idee: Konkatenierte Schlüssel: **Key1 | Key2**
Konkatenierte Werte:
K11 | K21
K11 | K22
⋮
K11 | K2m
K12 | K21
K12 | K22
⋮
K12 | K2m
K13 | K21
⋮
K1n | K2m

Suchoperationen:

- (Key1 = K1i) AND (Key2 = K2j) ?
- Key2 = K2j ?
- Key1 = K1i ?
- OR-Verknüpfung ?

Vergleich	Quad-Tree	Multi-Key Hashing	k-d-Baum, k-d-B-Baum	Grid-File	R-Baum
Dynamische Reorganisation					
Erhaltung der topologischen Struktur (Clusterbildung)					
Bestimmung der Nachbarschaft: Bereichsanfragen, best match					
Trennung von Zugriffsstruktur und Daten,					
Exakte Anfrage (balanc. Struktur)					
Objektdarstellung					

Quad-Tree (Quadranten-Baum)

- **Speicherungsstruktur für 2-dimensionalen Mehrattributzugriff**

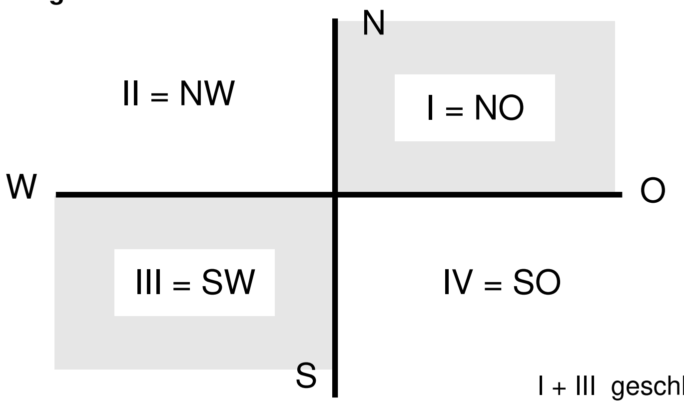
- **Zerlegungsprinzip des Datenraumes D:**

  ➥ rekursive Partitionierung durch Quadranten

- **Realisierung als Generalisierung des Binärbaumes:**
  - jeder Knoten enthält einen Satz
  - Außengrad eines Knotens: max. 4
  - Wurzel teilt 2-dim. Raum in 4 Quadranten auf
  - rekursive Aufteilung jedes Quadranten durch Wurzel eines Unterbaumes
  - i-ter Unterbaum eines Knotens enthält Punkte im i-ten Quadranten

- **Ziel: Berücksichtigung von Nachbarschaftsbeziehungen**
  Bsp.: geographische Daten: x, y - Koordinaten

- **Aufteilung:**



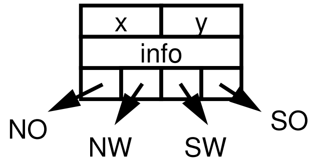
I + III geschlossen
II + IV offen

- **Verallgemeinerung:**

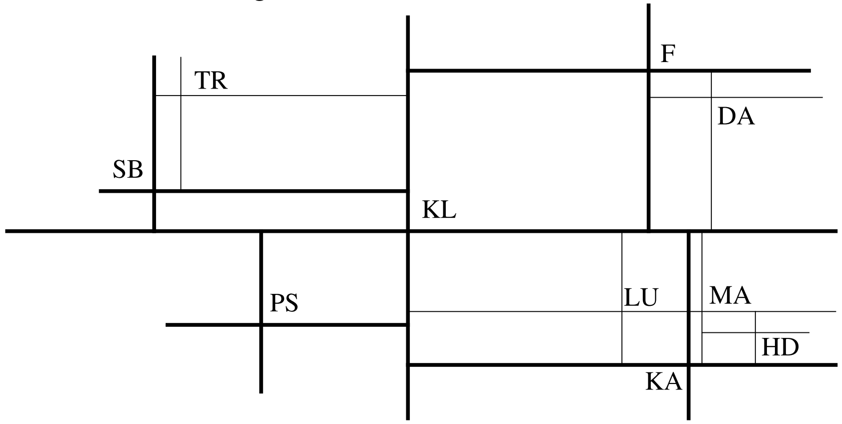
  k-dimensionaler Schlüssel ⇒ Außengrad jedes Knotens: 2^k
  k=3: Octtree, k=4: Hextree

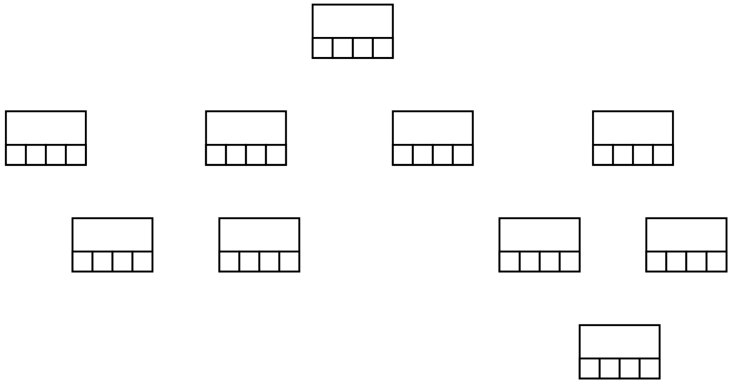
Quad-Tree (2)

- **Knotenformat**



- **Räumliche Aufteilung**





- **Eigenschaften**
  - Baumstruktur abhängig von Einfügereihenfolge (unbalanciert)
  - aufwendiges Löschen (Neueinfügen der Unterbäume)
  - keine Abbildung auf Seiten

Multi-Key-Hashing (partitioned hashing)

- **Zerlegungsprinzip von D:**

  ➥ Partitionierung durch Hashfunktionen in jeder Dimension:
  wird realisiert durch Aufteilung der Bucketadresse in k Teile
  (k = Anzahl der Schlüssel)

- **Für jeden Schlüssel i:**
  eigene Hash-Funktion h_i, die Teil der Bucketadresse bestimmt

  - Anzahl der Buckets sei 2^B (Adresse = Folge von B Bits)
  - jede Hash-Funktion h_i liefert b_i Bits mit $B = \sum_{i=1}^{k} b_i$
  - Satz $t = (a_1, a_2, ..., a_k, ...)$ gespeichert in Bucket mit

    Adr. $= h_1(a_1) \mid h_2(a_2) \mid ... \mid h_k(a_k)$

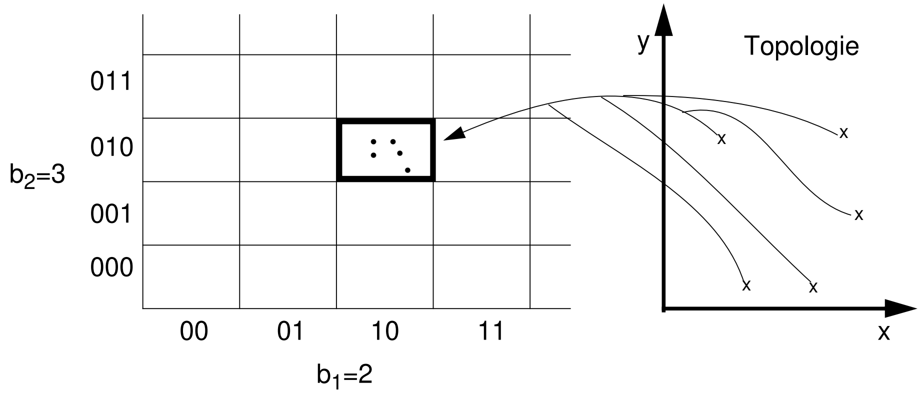
- **Vorteile:**
  - kein Index, geringer Speicherplatzbedarf und Änderungsaufwand
  - Exact-Match-Anfragen: Gesamtadresse bekannt

    ➥ Zugriff auf 1 Bucket
  - Partial-Match-Anfrage: Eingrenzung des Suchraumes
    $(A_i = a_i)$: Anzahl zu durchsuchender Buckets reduziert sich um 2^{b_i}

    ➥ $N_B = 2^B / 2^{b_i} = 2^{B-b_i}$

- **Nachteile** / **Probleme:**
  - topologische Struktur der Daten bleibt *nicht* erhalten
  - keine Unterstützung von Bereichs- und Best-Match-Anfragen
  - Optimale Zuordnung der b_i zu A_i abhängig von Fragehäufigkeiten

Multi-Key-Hashing: Beispiel



Bucket mit Adresse ’10010’ enthält alle Sätze mit

$h_1(a_1) =$ ’10’ und $h_2(a_2) =$ ’010’

- **Anwendungsbeispiel:**

Pnr:	INT (5)	$b_1 = 4$	
Svnr:	INT (9)	$b_2 = 3$	
Aname:	CHAR (10)	$b_3 = 2$	➥ B=9 (512 Buckets)

$h_1(Pnr) = Pnr \bmod 16$	$h_1(58651) = 11$	$\rightarrow$ 1011
$h_2(Svnr) = Svnr \bmod 8$	$h_2(130326734) = 6$	$\rightarrow$ 110
$h_3(Aname) = L(Aname) \bmod 4$	$h_3(XYZ55) = 1$	$\rightarrow$ 01

➥ B-Adr. = 101111001

- **Anzahl der Zugriffe**

Exact-Match-Anfragen: Zugriff auf 1 Bucket

Partial-Match-Anfragen:

Pnr = 58651 ➥ $N_B = 2^9/2^4 = 32$

(Pnr = 73443) AND (Svnr = 2332) ➥ $N_B = 2^9/ 2^{4+3} = 4$

Mehrdimensionale binäre Suchbäume

- **Erweiterung des Binärbaumes**
  - Berücksichtigung von k Schlüsseln: k-d-Baum
  - alle Datensätze werden mit Hilfe der Baumstruktur organisiert: knotenorientiert (homogen)
  - Wartungsoperationen wie beim binären Suchbaum, aber
  - **auf jeder Ebene erfolgt Schlüsselvergleich für einen der k Schlüssel**

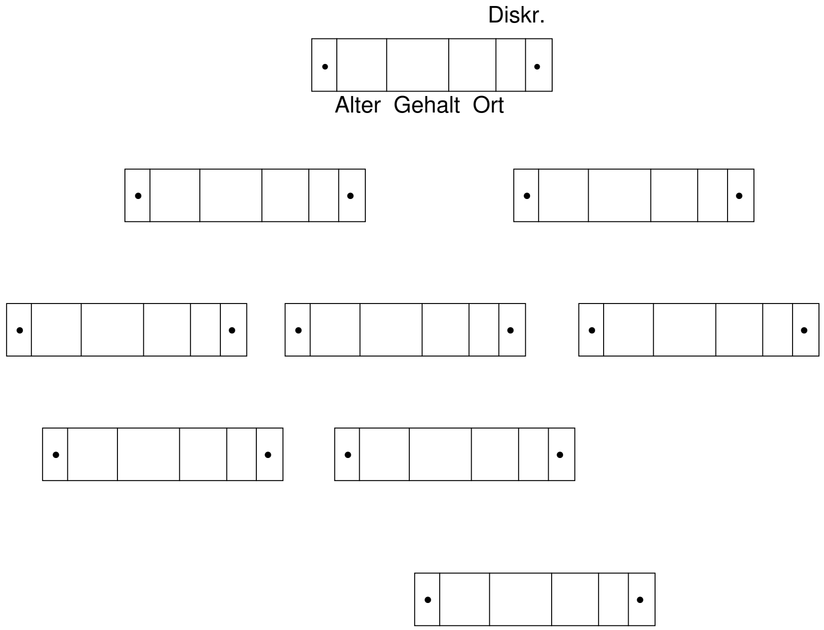
- **Diskriminator legt Schlüssel auf jeder Ebene fest**
  - zyklische Variation des Diskriminators d:
    für alle Knoten der Baumebene i gilt: d = (i mod k) + 1
  - der linke (rechte) Nachfolger zu einem Knoten enthält alle Sätze mit kleineren (größeren) Werten für das Diskriminatorattribut
    $\forall$ Q $\in$ LOSON(P): $K_d(Q) \leq K_d(P)$
    $\forall$ R $\in$ HISON(P) : $K_d(R) > K_d(P)$

- Baumstruktur ist nicht balanciert - verschiedene Erweiterungsvorschläge
  - zu einem Satz werden m-1 nachfolgende Ebenen des Baumes komplett in eine Seite abgebildet:
    - Seite muß bis zu 2^m - 1 Sätze aufnehmen können
    - jeder Satz auf Ebene 1, m+1, 2m+1, ... bildet Wurzel eines Teilbaumes, der in einer Seite liegt

- **Weitere Variante des k-d-Baumes**
  - blattorientiert (heterogen, Speicherung der Sätze in Buckets)

Beispiel: 3-d-Baum (knotenorientiert)

Alter	Gehalt	Ort
35	17K	KL
28	40K	F
29	15K	DA
25	45K	KL
40	12K	SB
29	16K	DA
30	17K	F
42	100K	F
29	14K	DA



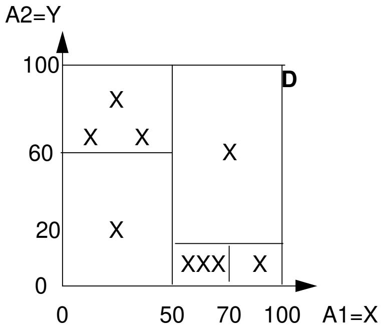
- Baumstruktur abhängig von Einfügereihenfolge (unbalancierter Baum)
- Eingrenzung des Suchraumes für Partial-Match- und Bereichsanfragen
- Löschen ist sehr schwierig

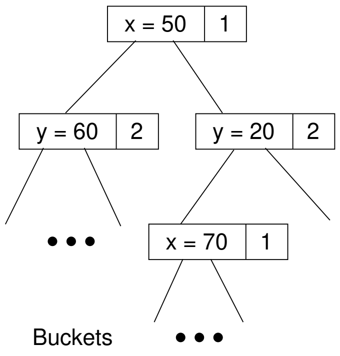
Organisation des umgebenden Datenraums - Divide and Conquer

- **Zerlegungsprinzip von D**
  - D wird dynamisch in Zellen aufgeteilt
  - Die Objekte einer Zelle werden als Sätze in Buckets abgelegt
  - bei Bucket-Überlauf: lokale Zellverfeinerung

    ➥ Divide and Conquer
  - abschnittsweise Erhaltung der Topologie (Clusterbildung)
  - Baum als Zugriffsstruktur für die Buckets hat nur Wegweiserfunktion

- **Beispiel: Heterogener k-d-Baum - k=2:**



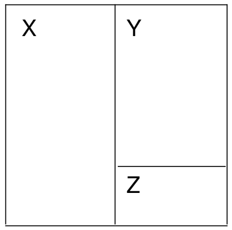


- **Eigenschaften des k-d-Baumes**
  - Clusterbildung durch Buckets ist Voraussetzung für praktischen Einsatz
  - Eingebauter Balancierungsmechanismus ist nicht vorhanden
  - Wie werden Aktualisierungsoperationen (Löschen!) durchgeführt?
  - Wie werden die verschiedenen Anfragetypen unterstützt?

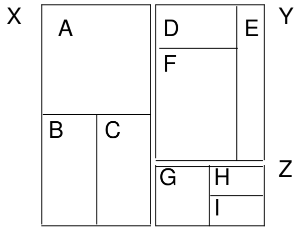
k-d-B-Baum[1]

- **Lösungsidee: Kombination von k-d- und B*-Baum**
  - k-d-B-Baum paginiert k-d-Bäume und ordnet ihren Teilbäumen Buckets (Seiten) zu, wie das bei B*-Bäumen der Fall ist
  - auf jeder Baumebene wird der k-dimensionale Datenraum in schachtelförmige Zellen oder Regionen (bei k=2 in Rechtecke) partitioniert, wobei eine Region jeweils die Zellen/Regionen eines Knotens der darunterliegenden Ebene zusammenfaßt und repräsentiert
  - alle Datensätze sind in den Blättern (Buckets) gespeichert
  - die inneren Knoten (Index- oder Directory-Seiten) besitzen nur Wegweiser
  - operationales Verhalten wie beim B*-Baum

- **Beispiel:**



Ebene 0 (Wurzel), eine Indexseite



Ebene 1, drei Indexseiten X, Y, Z

Ebenenweise Organisation der Regionen von D (k=2)

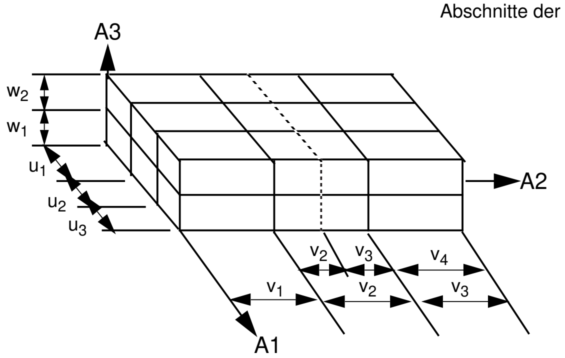
- **Sehr komplexe Regeln für die dynamische Reorganisation**
  - Bucket-Überlauf erzwingt Split der Zelle - optimierte Aufteilung der Zelle möglich (z. B. Median-Split)
  - Fortpflanzung des Split zur Wurzel hin möglich
  - bei Split von Indexseiten ist manchmal eine Neuaufteilung von Regionen erforderlich (sonst können sich sehr ungünstige Strukturen ergeben), was einen sich zu den Blattknoten hin fortpflanzenden Split (forced split) auslöst!

1. Robinson, J.T.: The k-d-B-Tree: a search structure for large multidimensional dynamic indexes, in: Proc. ACM SIGMOD Conf., 1981, pp. 10-18

Organisation des umgebenden Datenraums - Dimensionsverfeinerung

- **Prinzip**
  - Datenraum D wird dynamisch durch **ein orthogonales Raster** (grid) partitioniert, so daß k-dimensionale Zellen (Grid-Blöcke) entstehen
  - Die in den Zellen enthaltenen Objekte werden in Buckets gespeichert
  - Eine Zelle ist deshalb eindeutig einem Bucket zuzuordnen
  - Die klassenbildende Eigenschaft dieser Verfahren ist das Prinzip der Dimensionsverfeinerung, bei dem ein Abschnitt in der ausgewählten Dimension durch einen vollständigen Schnitt durch D verfeinert wird
- **Beispiel**

Datenraum D = A1 x A2 x A3
Zellpartition P = U x V x W
Abschnitte der Partition $U = (u_1, u_2, \ldots u_1)$
$V = (v_1, v_2, \ldots, v_m)$
$W = (w_1, w_2, \ldots, w_n)$

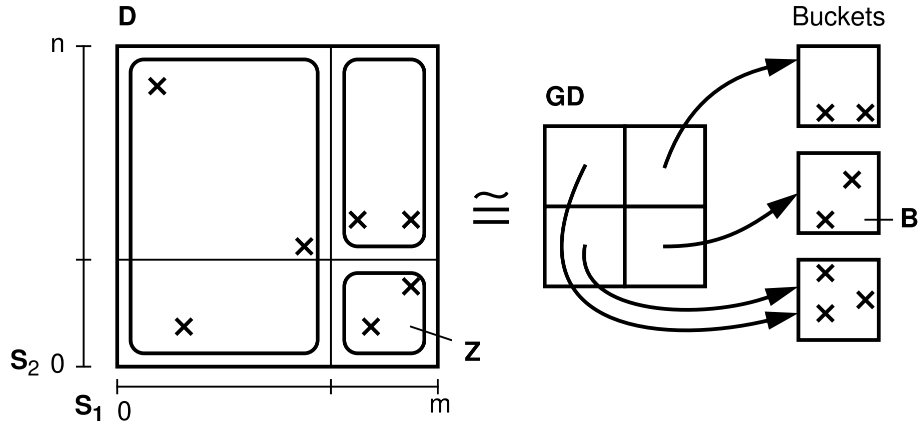


Dreidimensionaler Datenraum D mit Zellpartition P; Veranschaulichung eines Split-Vorganges im Intervall v_2

- **Probleme der Dimensionsverfeinerung**
  - Wieviele neue Zellen enstehen jedesmal?
  - Was folgt für die Bucketzuordnung?
  - Welche Abbildungsverfahren können gewählt werden?
  - Gibt es Einschränkungen bei der Festlegung der Dimensionsverfeinerung?

Grid-File[1]: Idee

- **Zerlegungsprinzip von D: Dimensionsverfeinerung**



- **Komponenten**
  - k Skalierungsvektoren (Scales) definieren die Zellen (Grid) auf k-dim. Datenraum D
  - Zell- oder Grid-Directory GD: dynamische k-dim. Matrix zur Abbildung von D auf die Menge der Buckets
  - Bucket: Speicherung der Objekte einer oder mehrerer Zellen (Bucketbereich BB)
- **Eigenschaften**
  - 1:1-Beziehung zwischen Zelle Z_i und Element von GD
  - Element von GD = Ptr. zu Bucket B
  - n:1-Beziehung zwischen Z_i und B
- **Ziele**
  - Erhaltung der Topologie
  - effiziente Unterstützung aller Fragetypen
  - vernünftige Speicherplatzbelegung

1. Nievergelt, J. et al.: The Grid File: An Adaptable, Symmetric Multikey File Structure, ACM Trans. Database System, 1984, pp. 38-71

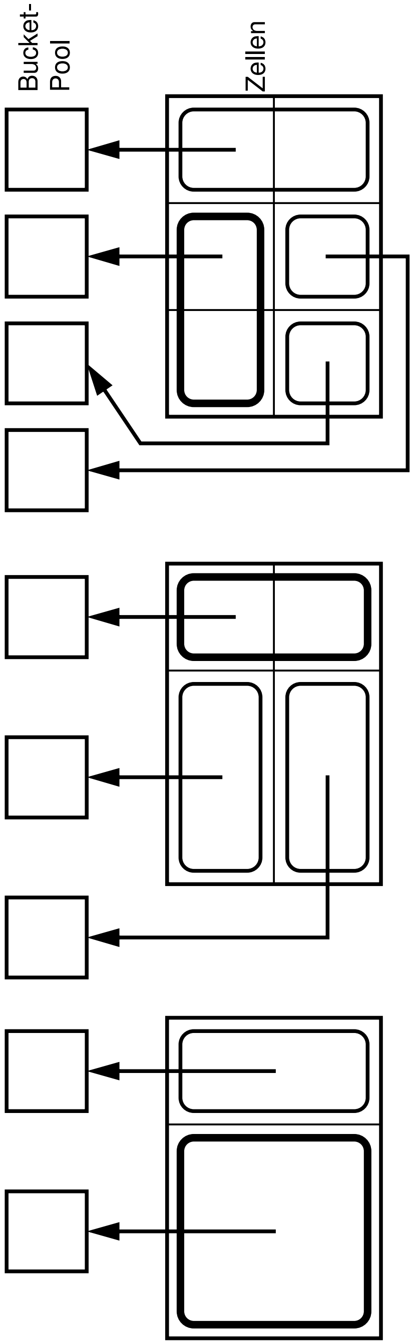
Zentrale Datenstruktur: Grid-Directory

- **Anforderungen**
  - Prinzip der zwei Plattenzugriffe
    unabhängig von Werteverteilungen, Operationshäufigkeiten
    und Anzahl der gespeicherten Sätze
  - Split- und Mischoperationen jeweils nur auf zwei Buckets
  - Speicherplatzbelegung
    - durchschnittliche Belegung der Buckets nicht beliebig klein
    - schiefe Verteilungen vergrößern nur GD

- **Entwurf einer Directory-Struktur**
  - dynamische k-dim. Matrix GD (auf Externspeicher)
  - k eindim. Vektoren S_i (im Hauptspeicher)

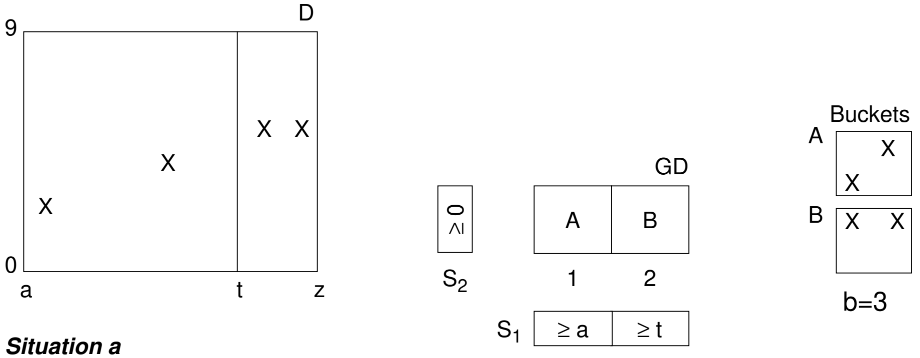
- **Operationen auf GD**
  - direkter Zugriff auf einen GD-Eintrag
  - relativer Zugriff (NEXTABOVE, NEXTBELOW)
  - Mischen zweier benachbarter Einträge einer Dimension
    (mit Umbenennung der betroffenen Einträge)
  - Splitten eines Eintrages einer Dimension (mit Umbenennung)

Schachtelförmige Zuweisung von Zellen zu Buckets

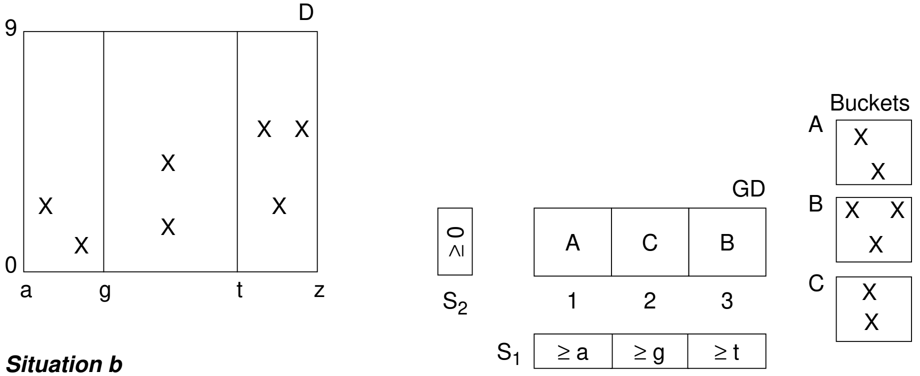


Grid-File - Beispiel

- Grid-File (GF) besteht aus
  - einer Menge von Buckets
  - k(=2) Skalierungsvektoren als sequentielle Listen
  - k(=2)-dimensionalem GD, das die Zellen von D repräsentiert und Verweise auf die zugeordneten Buckets besitzt (A = Adresse von Bucket A)
- **Schrittweise Entwicklung eines GF**



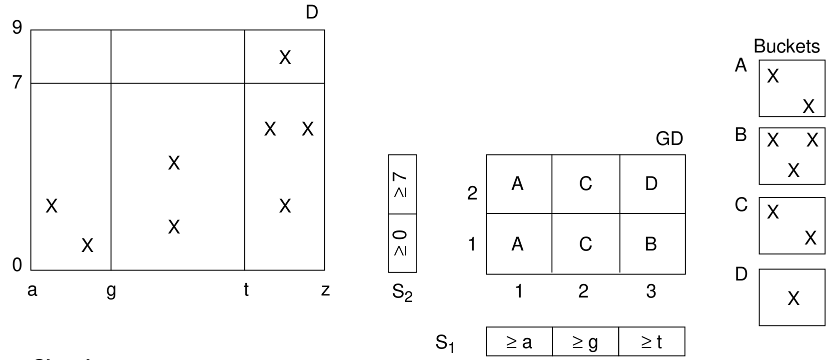
Situation a



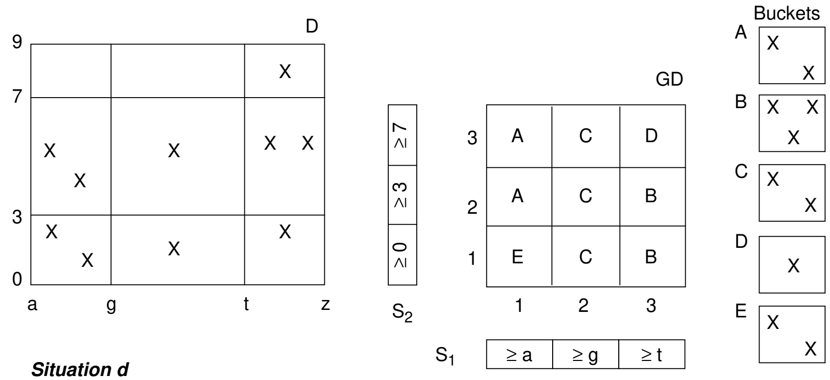
Situation b

Grid-File - Beispiel (2)

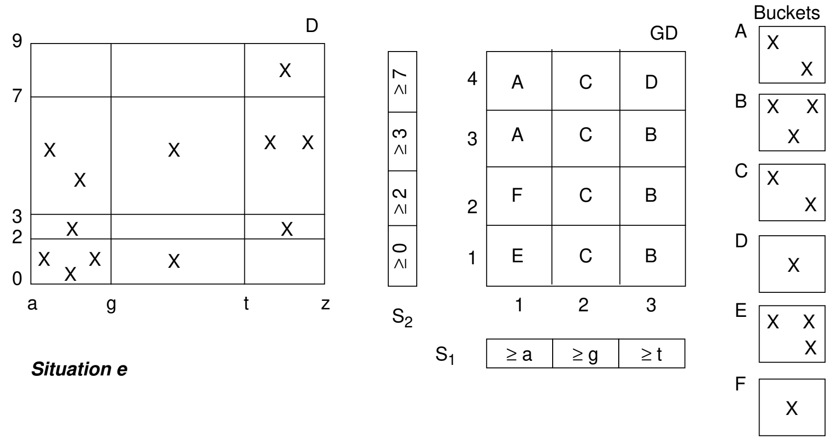
- **Verfeinerung des Datenraums und zugehörige Entwicklung des GF**



Situation c



Situation d

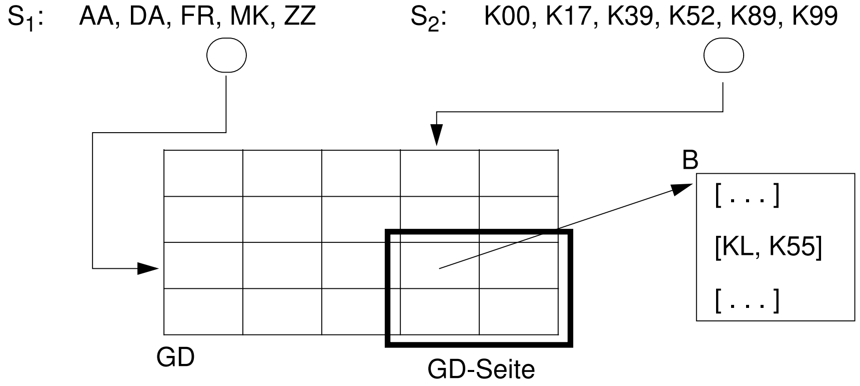


Situation e

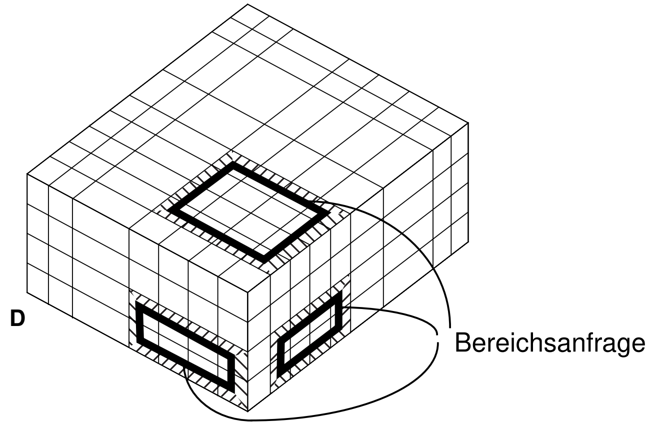
Grid-File - Suchfragen

- **Exakte Anfrage (exact match)**

  SELECT *
  FROM PERS
  WHERE ORT = ‘KL’ AND ANR = ‘K55’



- **Bereichsanfrage**
  - Bestimmung der Skalierungswerte in jeder Dimension
  - Berechnung der qualifizierten GD-Einträge
  - Zugriff auf die GD-Seite(n) und Holen der referenzierten Buckets



Zugriffspfade für ausgedehnte räumliche Objekte

- **Ausgedehnte räumliche Objekte besitzen**
  - allgemeine Merkmale wie Name, Beschaffenheit, . . .
  - Ort und Geometrie (Kurve, Polygon, . . .)

- **Indexierung des räumlichen Objektes**
  - genaue Darstellung?
  - Objektapproximation durch schachtelförmige Umhüllung - effektiv!

    ➥ dadurch werden Fehltreffer möglich

- **Probleme**
  - neben Objektdichte muß Objektausdehnung bei der Abbildung und Verfeinerung berücksichtigt werden
  - Objekte können andere enthalten oder sich gegenseitig überlappen

- **Klassifikation der Lösungsansätze**
  - sich gegenseitig überlappende Regionen (R-Baum)
  - Clipping (R^+-Baum)
  - Transformationsansatz

    ➥ bildet ausgedehnte räumliche Objekte funktional auf höherdimensionale Punkte ab - begrenzte Anwendbarkeit und Tauglichkeit!

R-Baum[1]

- **Ziel: Effiziente Verwaltung räumlicher Objekte**
  (Punkte, Polygone, Quader, ...)

- **Anwendungen:**
  - Kartographie:
    Speicherung von Landkarten, effiziente Beantwortung "geometrischer" Fragen
  - CAD:
    Handhabung von Flächen, Volumina und Körpern
    (z.B. Rechtecke beim VLSI-Entwurf)
  - Computer-Vision und Robotics

- **Hauptoperationen:**
  - Punktanfragen (point queries):
    Finde alle Objekte, die einen gegebenen Punkt enthalten
  - Gebietsanfragen (region queries):
    Finde alle Objekte, die mit einem gegebenen Suchfenster überlappen (es umschließen, in ... vollständig enthalten sind)

- **Ansatz:** Speicherung und Suche von **achsenparallelen Rechtecken**
  - Objekte werden durch Datenrechtecke repräsentiert und müssen durch kartesische Koordinaten beschrieben werden
  - Repräsentation im R-Baum erfolgt durch minimale begrenzende (k-dimensionale) Rechtecke/Regionen
  - Suchanfragen beziehen sich ebenfalls auf Rechtecke/Regionen

1. A. Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching, in: Proc. ACM SIGMOD Conf., 1984, pp. 47-57

R-Baum (2)

- **R-Baum ist höhenbalancierter Mehrwegbaum**
  - jeder Knoten entspricht einer Seite
  - pro Knoten maximal M, minimal m (>= M/2) Einträge

Blattknoteneintrag:

I_1	I_2	• • •	I_k	TID

kleinstes umschreibendes Rechteck (Datenrechteck) für TID

Zwischenknoteneintrag:

I_1	I_2	• • •	I_k	PID

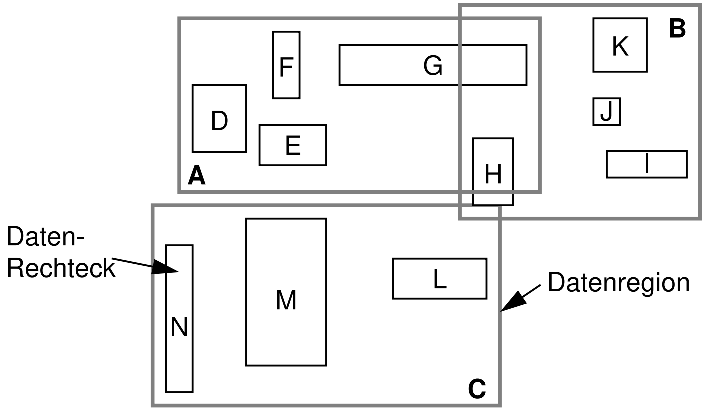
Intervalle beschreiben kleinste umschreibende Datenregion für alle in PID enthaltenen Objekte

I_j = geschlossenes Intervall bzgl. Dimension j
TID: Verweis auf Objekt PID: Verweis auf Sohn

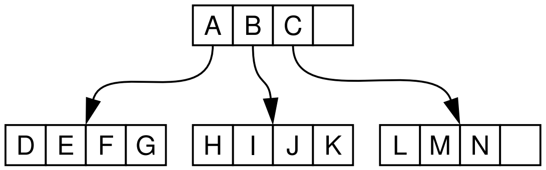
- **Eigenschaften**
  - starke Überlappung der umschreibenden Rechtecke/Regionen auf allen Baumebenen möglich
  - bei Suche nach Rechtecken/Regionen sind ggf. mehrere Teilbäume zu durchlaufen
  - Änderungsoperationen ähnlich wie bei B-Bäumen

Abbildung beim R-Baum

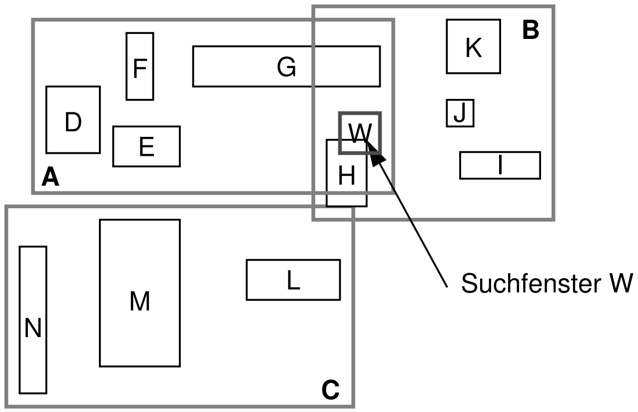
- **Aufteilung des Datenraumes**



- **R-Baum**

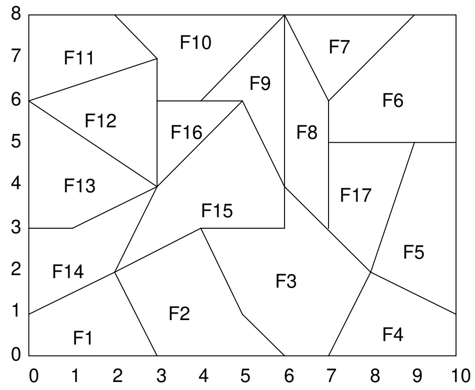


- **Beispiel für ein „schlechtes“ Suchfenster**

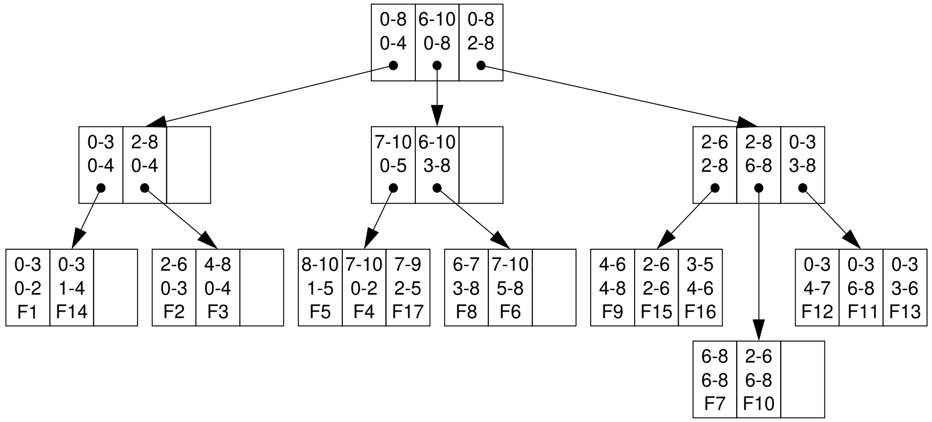


R-Baum: Beispiel

- **Abzuspeichernde Flächenobjekte**



- **Zugehöriger R-Baum**



Suchoptimierung durch den R^+-Baum

- **Überdeckung und Überlappung** bei einer Ebene des R-Baumes
  - **Überdeckung** (coverage) ist der gesamte Bereich, um alle zugehörigen Rechtecke zu überdecken
  - **Überlappung** (overlap) ist der gesamter Bereich, der in zwei oder mehr Knoten enthalten ist
  - Minimale Überdeckung reduziert die Menge des „toten Raumes“ (leere Bereiche), der von den Knoten des R-Baumes überdeckt wird.
  - Minimale Überlappung reduziert die Menge der Suchpfade zu den Blättern (noch kritischer für die Zugriffszeit als minimale Überdeckung)

➥ Effiziente Suche erfordert minimale Überdeckung und Überlappung

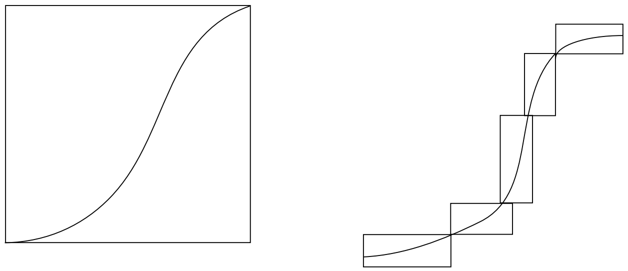
- **Idee**

  Es sind Partitionierungen erlaubt, die Datenrechtecke „zerschneiden” (Clipping)

➥ Vermeidung von Überlappungen bei Zwischenknoten

- **Konsequenz**

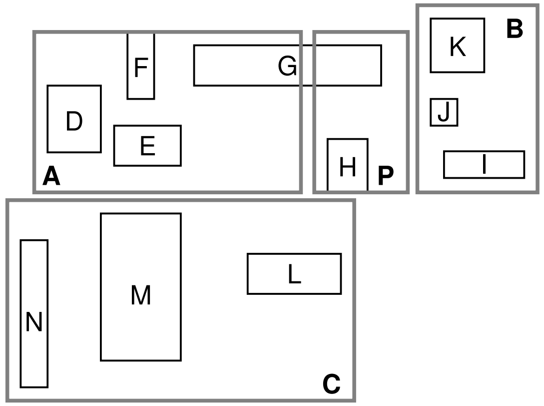
  Daten-Rechteck wird ggf. in eine Sammlung von disjunkten Teilrechtecken zerlegt und auf der Blattebene in verschiedenen Knoten gespeichert.



Aufteilungsmöglichkeiten eines langen Linienobjektes im R^+-Baum

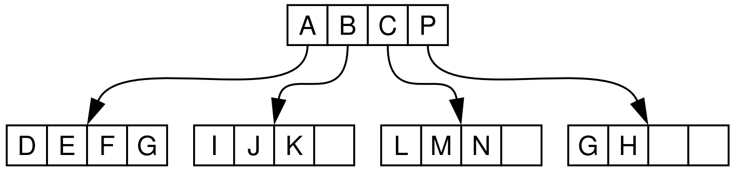
R^+-Baum[1]

- **Aufteilung des Datenraumes**



➥ **Höhere Flexibilität durch Partitionierung von Daten-Rechtecken**

- **R^+-Baum**



- **Eigenschaften**
  - Überlappung von Datenregionen wird vermieden
  - Überdeckungsproblematik wird wesentlich entschärft
  - Clusterbildung der in einer Region zusammengefaßten Objekte kann durch Clipping verhindert werden
  - komplexere Algorithmen für bestimmte Anfragen (Enthaltensein)
  - schwierigere Wartung, keine Leistungsvorteile gegenüber R-Baum

1. Sellis, T. et a.: The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects, in: Proc.14th Int. Conf. on Very Large Data Bases, Brighton,1987, pp. 507-518

Anfragen an räumliche Objekte - Aspekte der Erweiterung der DB-Sprache

- **Kein Raumbezug und keine räumlichen Operatoren** im Relationenmodell
  - Hohe Komplexität bei einfachen Beispielen
  - **Beispiel:** Darstellung beliebig gelagerter Rechtecke in der Ebene

    R-ECK (RE-NR, X1, Y1, X2, Y2, X3, Y3, X4, Y4)

  Anfrage: Finde alle Rechtecke, die ein Rechteck mit den Punkten (PA, PB, PC, PD) echt umschließen.

```
SELECT RE-NR FROM R-ECK
WHERE   XA > X1 AND YA < G(P1, P2, XA)
           AND YA > G(P4, P1, XA) AND
        XB < Y2 AND YB < G(P1, P2, XB)
           AND XB < G(P2, P3, XB) AND
        XC < X3 AND YC < G(P2, P3, XC)
           AND YC > G(P3, P4, XC) AND
        YD > Y4 AND YD > G(P3, P4, XD)
           AND YD > G(P1, P4, XD);
```

  mit Abkürzungen: z.B. G(P1, P2, XA) für

  ((Y1 - Y2) ∗ XA) / (X1 - X2) + (Y2 ∗ X1 - Y1 ∗ X2) / (X1 - X2)

- **Objekt-relationale Datenmodelle - Schlüsseleigenschaft Erweiterbarkeit mit benutzerdefinierten Funktionen und Operatoren:** Anfragen mit Operatoren wie Überlappung, Schnitt, Entfernung, Enthaltensein («) ...

  ➥ SELECT RE-NR FROM R-ECK X
  WHERE [PA, PB, PC, PD] « X;

Zusammenfassung

- **Wichtige Eigenschaften mehrdimensionaler Zugriffspfade**
  - Erhaltung der topologischen Struktur des Datenraumes
  - Adaption an die Objektdichte
  - Dynamische Reorganisation
  - Balancierte Zugriffsstruktur
  - Unterstützung von verschiedenen Anfragetypen (*intersection queries*, *best match queries*)
- **Darstellung von räumlichen Objekten**
  - Abstraktion zu punktförmiger Repräsentation ist Regelfall
  - „Ausgedehnte“ Darstellung nur in grober Annäherung:

    ➥ Verarbeitungsprobleme
- **Vorteile neuerer Konzepte**
  - Organisation des umgebenden Datenraumes - Grid File
  - Darstellung räumlich ausgedehnter Objekte - R- und R^+-Baum

    ➥ **Jedoch**: Es existieren sehr viele Vorschläge und Konzepte, die sehr speziell und praktisch nicht erprobt sind
- **Notwendigkeit der DBS-Integration** von „konventionellen“ (eindimensionalen) und mehrdimensionalen Zugriffspfaden
  - Zugriffsmöglichkeiten über räumliche und zeitliche Dimensionen
  - Unterstützung von „Data Warehouse“-Anwendungen, Geographischen Informationssystemen, . . .
- **Erweiterung objekt-relationaler DBS** durch mehrdimensionale Zugriffspfadstrukturen (vor allem Grid File und R-Baum)

Eindimensionale Intervalle

- **Anwendungsbeispiele für Intervallanfragen**
  - Temporale Datenbanken
  - Fuzzy-Logik/Matching
  - Intervallmodellierung ist oft hilfreich bei Profilabgleich (Personalisierung von Web-Seiten) oder Qualitätsklassen (Qualitätskriterien in Intervalldarstellung)

- **Mehrdimensionale Indexstrukturen sind nicht spezialisiert** auf die
  - effiziente Verwaltung von 1D-Objekten bzw. Intervallen
  - optimierte Auswertung entsprechender Anfragen

- **Wie lassen sich Intervalle darstellen?**
  - in B*-Bäumen, Quad-Tree, Grid-File, ...
  - **Idee:** Endpunkt-Transformation resultiert in einem Parameterraum mit 2 Dimensionen von gleichen Wertebereichen

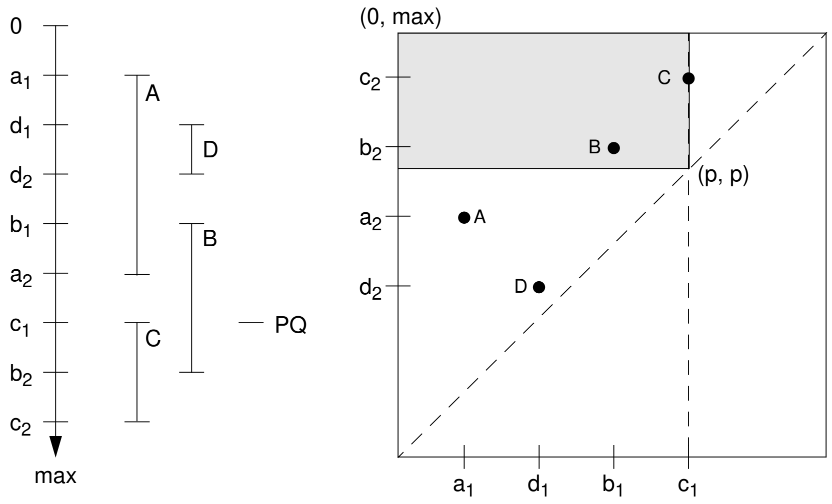
$$[13, 27] \xrightarrow[\text{Transformation}]{\text{Parameterraum}} (13, 27)$$

1D-Intervall | 2D-Punkt

- **Welche Anfragetypen sind zu unterstützen?**
  - Enthaltensein eines Punktes (PQ)
  - Überlappung von Intervallen
    - IQ: Intersection Query (Durchschnitt)
    - EQ: Enclosure Query (Umschließung)
    - CQ: Containment Query (Enthaltensein)

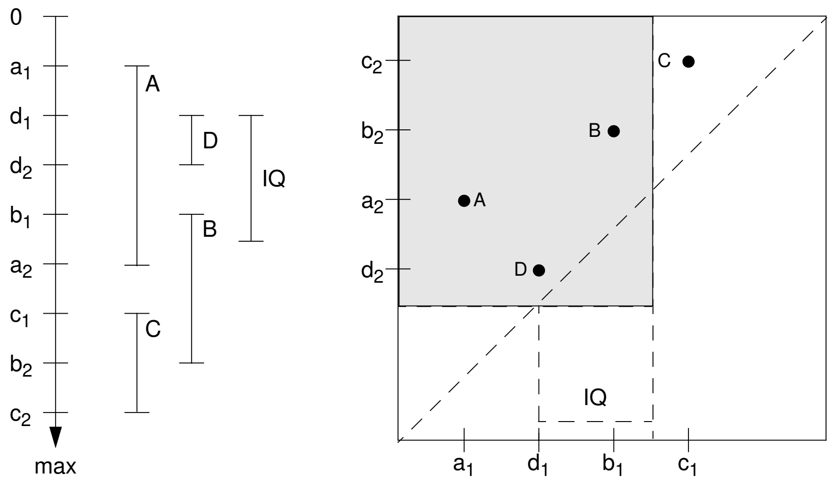
Eindimensionale Intervalle (2)

- **Wertebereich für Intervalle U = [0, max]**
- **Point Query (PQ): [p, p] mit (p ∈ U)**



  - Anfragebereich: [(0, max), (p, p)]

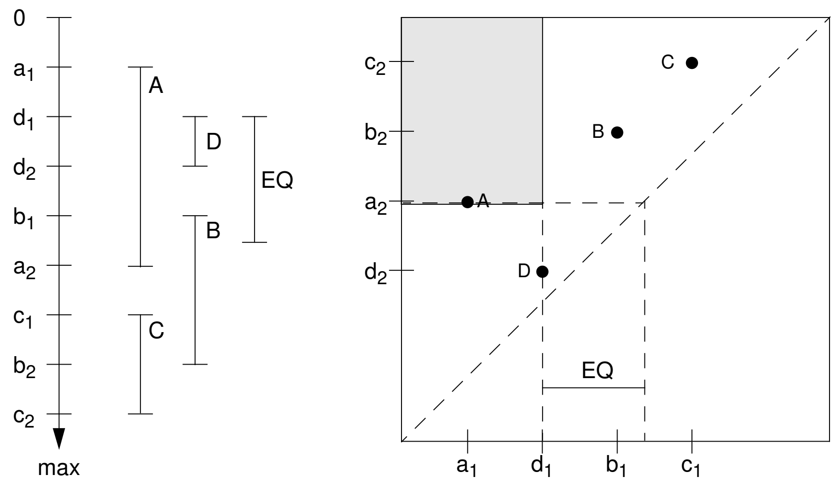
- **Intersection Query (IQ): [s, e] mit (s, e ∈ U)**



  - Anfragebereich: [(0, max), (e, s)]

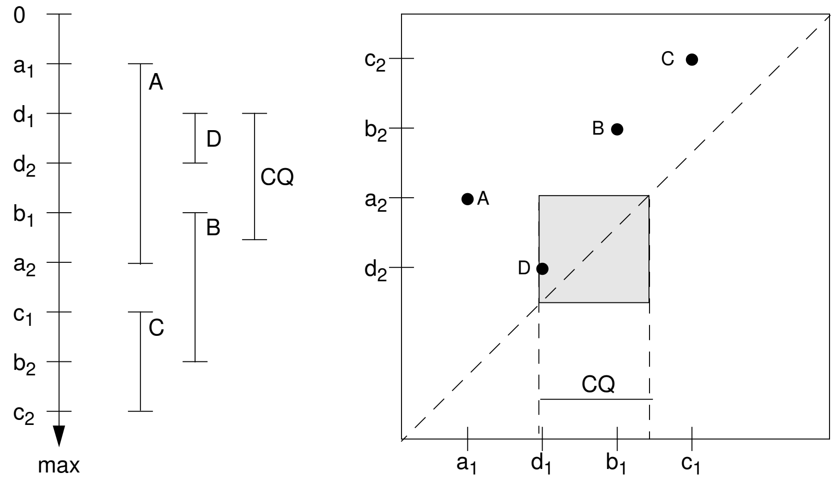
Eindimensionale Intervalle (3)

- **Enclosure Query (EQ): [s, e] mit (s, e ∈ U)**



- Anfragebereich: [(0, max), (s, e)]

- **Containment Query (CQ): [s, e] mit (s, e ∈ U)**

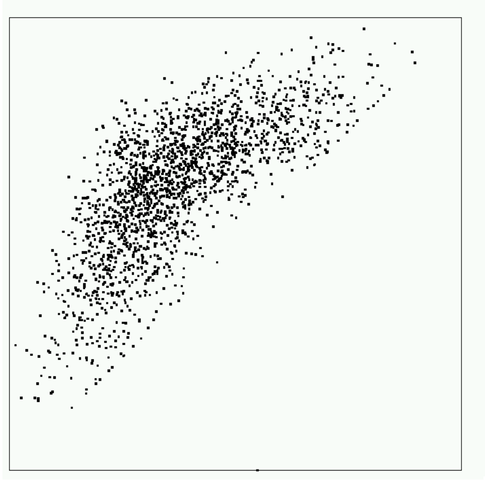


- Anfragebereich: [(s, e), (e, s)]

Eindimensionale Intervalle (4)

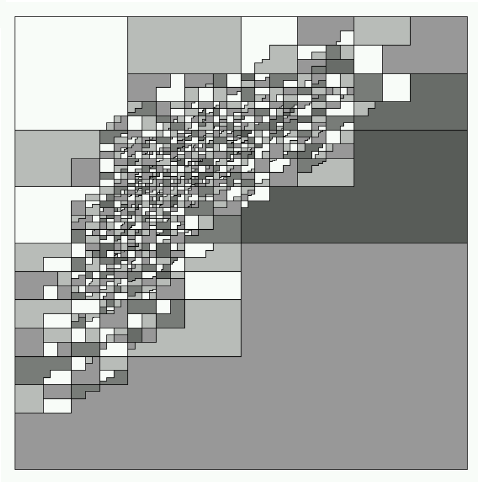
- **Abbildung auf UB-Baum ➥ Vorteile des B*-Baumes**
  - hervorragende Externspeicherverwaltung, Skalierbarkeit
  - logarithmische ‘worst case’-Garantien für alle wichtigen Operationen
  - Antwortzeit ist linear zur Größe der Ergebnismenge
  - Unabhängigkeit von Eingabedaten und Einfügereihenfolge
  - geringe Speicherplatzkosten für den Index

- **Beispiel mit 2000 Gauß-verteilten Intervallen**



Parameterraum

UB-Baum



Bayer, R.:
The Universal B-Tree for Multidimensional Indexing: General Concepts, in: Proc. Int. Conf. on Worldwide Computing and its Applications, LNCS 1274, Springer, 1997, pp. 198-209