4. Verfahren zur Organisation des umgebenden Datenraums

– möglichst nicht entlang der Objekte selber –

A) Lokale Verfeinerung ("Divide and Conquer")

Å Zerlegungsprinzip von D

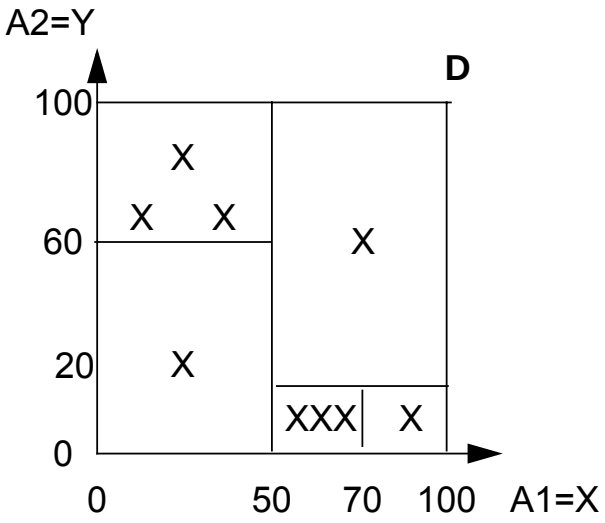
- D wird dynamisch in Zellen aufgeteilt
- die Objekte einer Zelle werden als Sätze in je einem Bucket abgelegt
- bei Bucket-Überlauf: lokale Zellverfeinerung

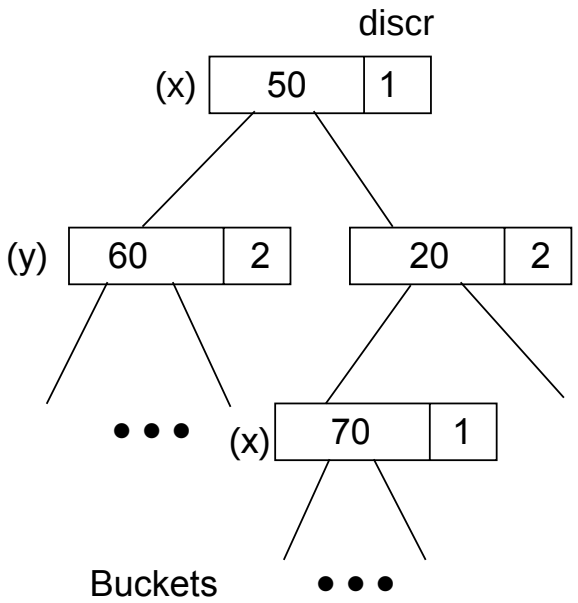
  ➥ "Divide and Conquer"

- also abschnittsweise Erhaltung der Topologie (Clusterbildung)

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

  Baum als Zugriffsstruktur hat nur Wegweiserfunktion für die Buckets (=Blätter)





Å **Eigenschaften**:

- kein Balancierungsmechanismus eingebaut, i.w. interne Datenstruktur

\+ exakte Anfrage einfach (ein Pfad)

\+ partielle / Bereichsanfrage: rekursiv, Suchbedingung aufteilen (mehrere Pfade)

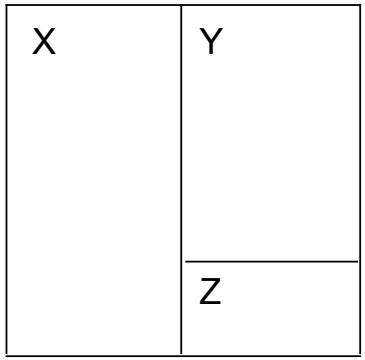
\+ nächster Nachbar?: analog zu Bereichsanfragen (Ähnlichkeitsbereich), steuern über vermutete maximale Entfernung als globalen Parameter, unter Annahme, dass Buckets nicht leer

- **k-d-B-Baum**

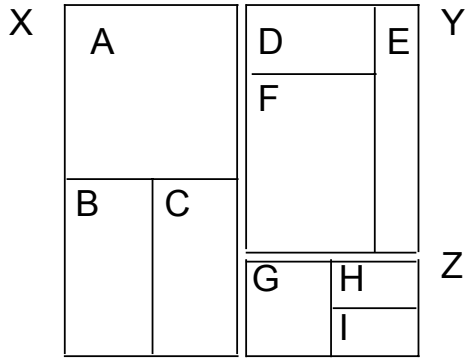
Å Idee: Kombination von k-d- und B*-Baum

- k-d-B-Baum "paginiert" k-d-Bäume und ordnet ihren Knoten 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 Blätter-Buckets gespeichert
- die inneren Knoten (Index- oder Directory-Seiten) fungieren nur als Wegweiser und speichern Beschreibungen mehrerer Zellen
- operationales Verhalten wie beim B*-Baum

Å Beispiel:



Ebene 0 (Wurzel),
eine Indexseite



Ebene 1,
drei Indexseiten X, Y, Z

Ebenenweise Organisation der Regionen von D (k=2)

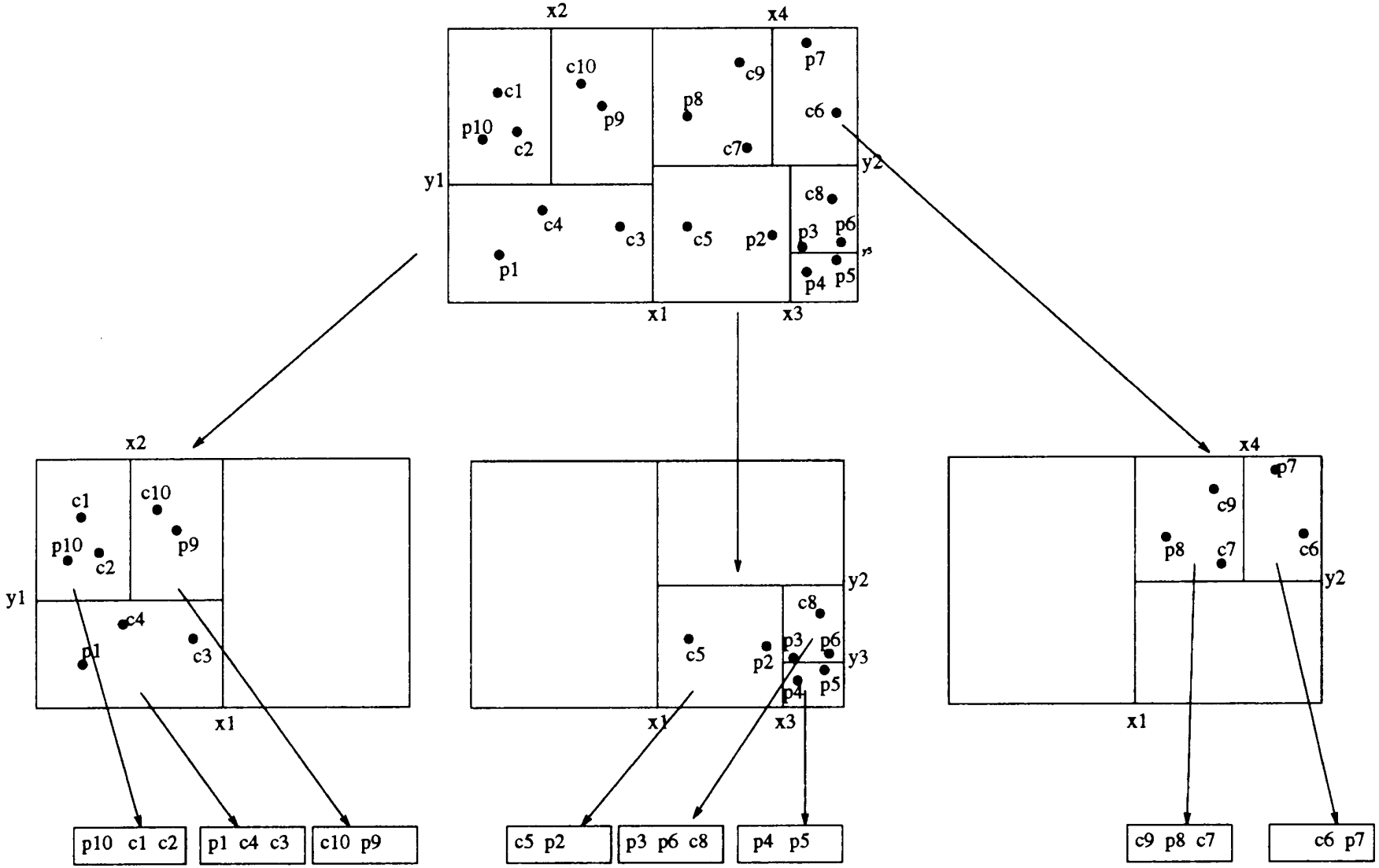
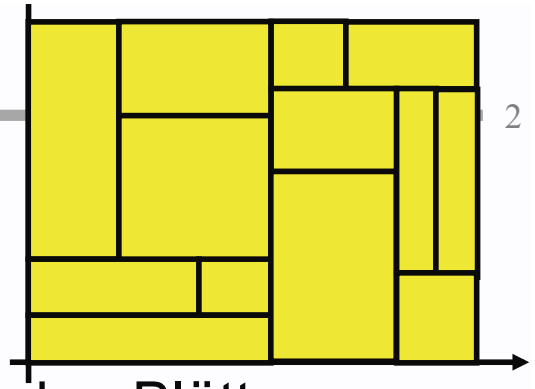


Figure 19. k-d-B-tree.

k-d-B-Baum

- Ein k-d-B-Baum besteht aus einer Menge von Seiten und einem Verweis auf die Wurzel-Seite.
- 2 Arten von Seiten:
  - *R*-Seiten (Indexseiten) enthalten eine Menge von Paaren (region, nachf)
  - *P*-Seiten (Datenseiten) enthalten eine Menge von Paaren (punkt, location)
  - region: Spezifikation einer Region
  - nachf: Verweis auf Nachfolger-Seite im Baum
  - punkt: Spezifikation eines Punktes
  - location: Verweis auf zugehörigen Datensatz
    (location wird im folgenden vereinfachend weggelassen)
- Die *P*-Seiten sind die Blätter des Baums mit der Punktemenge.
- Die *R*-Seiten definieren die Aufteilung des Datenraums in Regionen (für k=2 Dimensionen: Rechtecke).

k-d-B-Baum



Definierende Eigenschaften

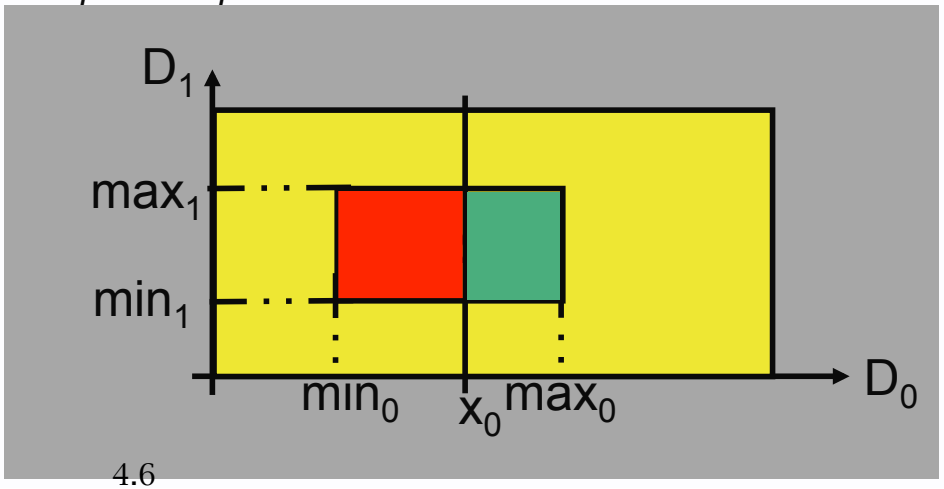
1. *R*-Seiten enthalten keine Null-Zeiger und sind nicht leer.
2. Die Längen aller Pfade von der Wurzel zu den Blättern (*P*-Seiten) sind identisch.
3. Alle Regionen in einer *R*-Seite sind disjunkt und ihre Vereinigung ergibt wieder eine Region.
4. Falls die Wurzel-Seite eine *R*-Seite ist, ergibt die Vereinigung ihrer Regionen den gesamten Datenraum.
5. Falls ein Eintrag (region, nachf) in einer *R*-Seite vorkommt,
   a) und nachf ist eine *R*-Seite, dann ist region die Vereinigung aller Regionen in nachf.
   b) und nachf ist eine *P*-Seite, dann liegen alle Punkte von nachf in region.

k-d-B-Baum

Sei $x_i \in D_i$ und $R = I_0 \times \ldots \times I_{k-1}$ eine Region mit $I_i = [\min_i, \max_i)$, $0 \le i \le k-1$.

Falls $x_i \in I_i$, definiere

- linke Teilregion von R bezüglich x_i:
  leftregion $(R, x_i) := I_0 \times \ldots \times [\min_i, x_i) \times \ldots \times I_{k-1}$
- rechte Teilregion von R bezüglich x_i:
  rightregion $(R, x_i) := I_0 \times \ldots \times [x_i, \max_i) \times \ldots \times I_{k-1}$
- Falls $x_i \notin I_i$, dann gilt
  - R liegt links von x_i gdw. $\max_i \le x_i$
  - R liegt rechts von x_i gdw. $x_i < \min_i$
- Sei $p = (p_0, \ldots, p_{k-1}) \in D$ ein Punkt. Dann gilt
  - p liegt links von x_i gdw. $p_i < x_i$
  - p liegt rechts von x_i gdw. $p_i \ge x_i$

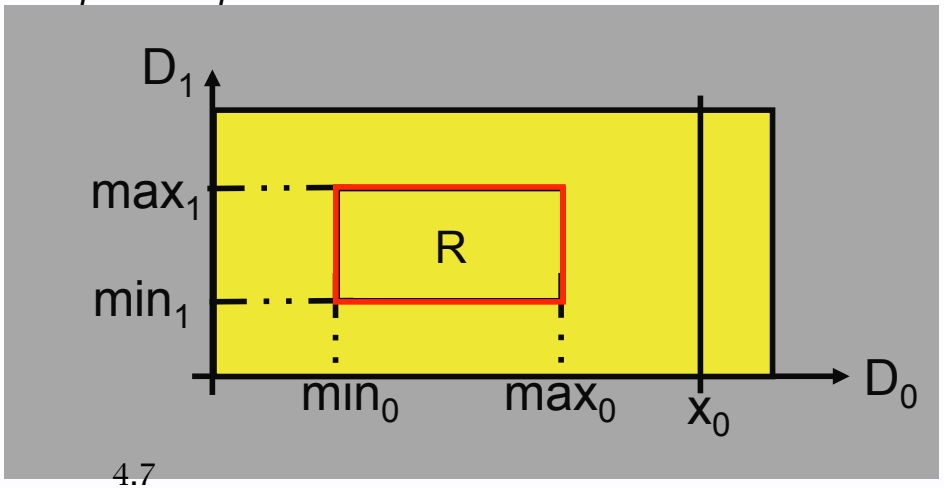


k-d-B-Baum

Sei $x_i \in D_i$ und $R = I_0 \times \ldots \times I_{k-1}$ eine Region mit $I_i = [\min_i, \max_i)$, $0 \leq i \leq k-1$.

Falls $x_i \in I_i$, definiere

- linke Teilregion von R bezüglich x_i:
  leftregion $(R, x_i) := I_0 \times \ldots \times [\min_i, x_i) \times \ldots \times I_{k-1}$
- rechte Teilregion von R bezüglich x_i:
  rightregion $(R, x_i) := I_0 \times \ldots \times [x_i, \max_i) \times \ldots \times I_{k-1}$
- Falls $x_i \notin I_i$, dann gilt
  - R liegt links von x_i gdw. $\max_i \leq x_i$
  - R liegt rechts von x_i gdw. $x_i < \min_i$
- Sei $p = (p_0, \ldots, p_{k-1}) \in D$ ein Punkt. Dann gilt
  - p liegt links von x_i gdw. $p_i < x_i$
  - p liegt rechts von x_i gdw. $p_i \geq x_i$

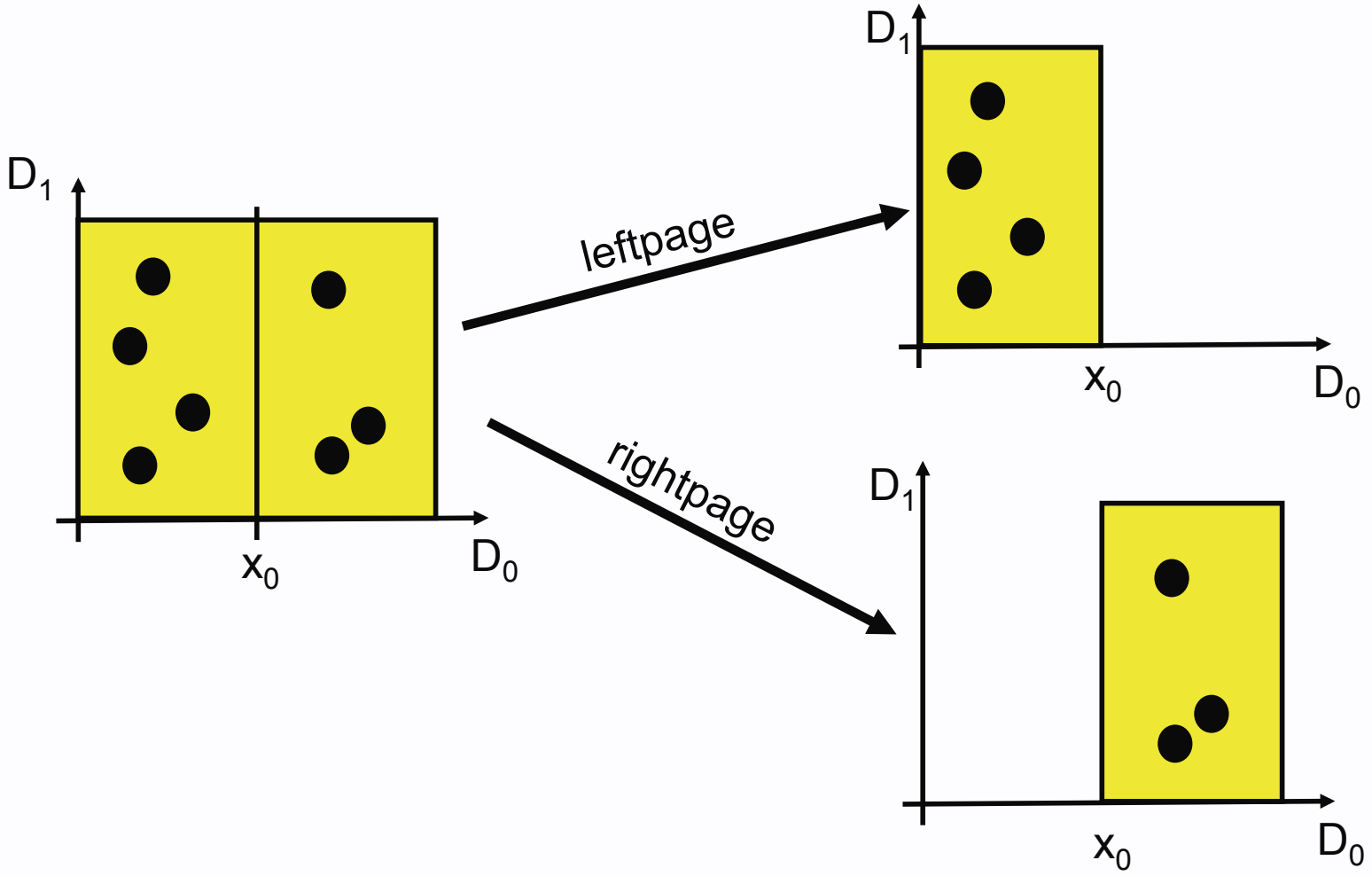


Spalten von Seiten im k-d-B-Baum

Split (page, x_i):
  if page verweist auf eine R-Seite
    then return (R-Split (page x_i))
    else return (P-Split (page, x_i));

P-Split (page, x_i):
Erzeuge neue P-Seiten leftpage und rightpage;
forall Punkte *p* in page **do**
    if *p* links von x_i **then** füge *p* in leftpage ein
    else füge *p* in rightpage ein;
Lösche page;
return (leftpage, rightpage);

Spalten von P-Seiten im k-d-B-Baum

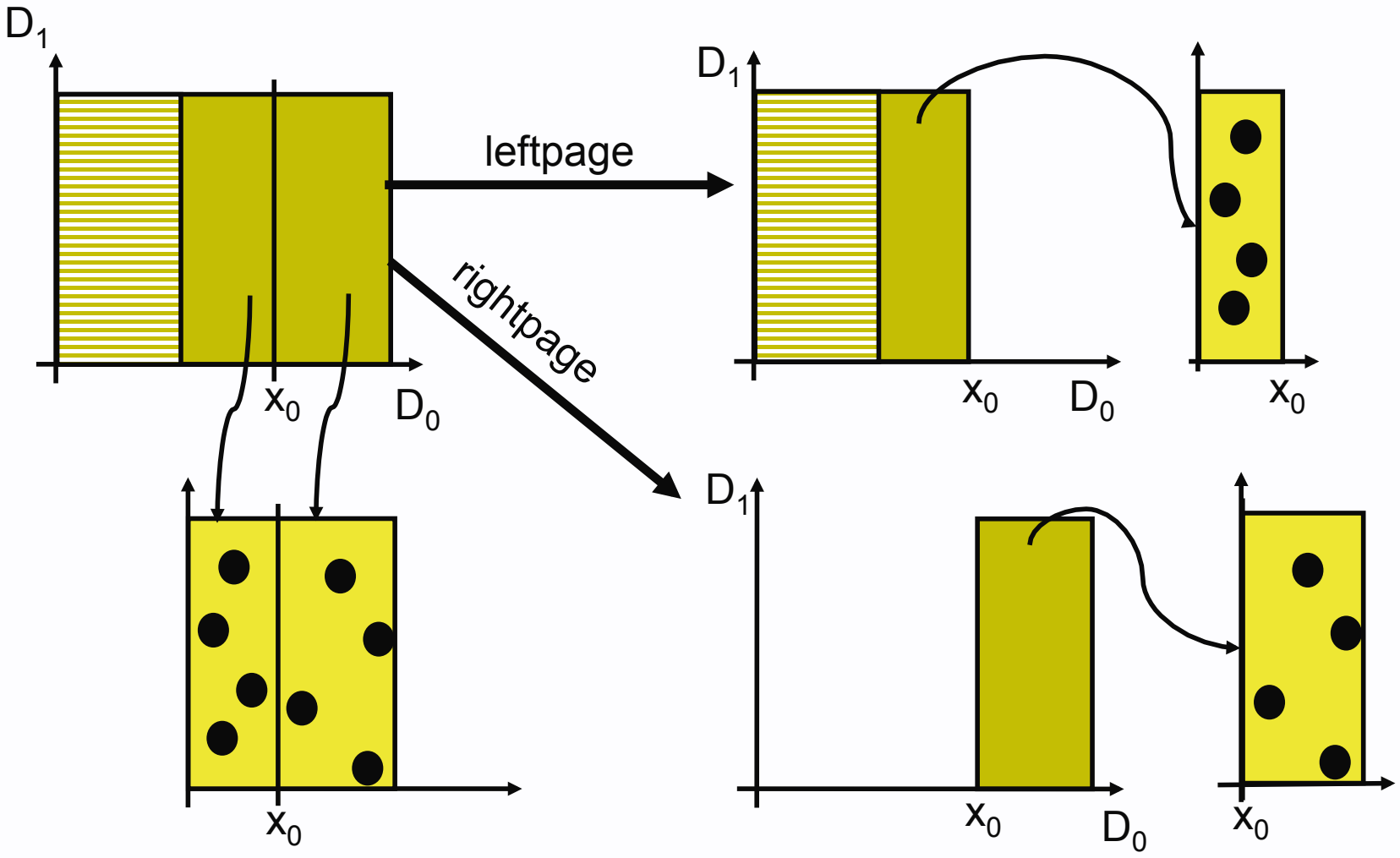


Spalten von R-Seiten im k-d-B-Baum

R-Split (page, x_i):
Erzeuge neue R-Seiten leftpage und rightpage;
forall Paare (region, nachf) in page:
    if region links von x_i **then**
        füge (region, nachf) in leftpage ein
    else if region rechts von x_i **then**
        füge (region, nachf) in rightpage ein
    else begin
        (leftpage', rightpage'):= Split (nachf,x_i); *forced split!*
        füge (leftregion(region, x_i), leftpage') in leftpage ein;
        füge (rightregion(region, x_i), rightpage') in rightpage ein;
    end;
Lösche page;
return (leftpage, rightpage);

Da durch die Wahl von x_i (s.u.) eine Neuaufteilung der Regionen erforderlich ist, wird ein sich meist bis zu den Blattknoten fortpflanzender Split erwungen..

Spalten von R-Seiten im k-d-B-Baum



Einfügen von Datensätzen

Insert (*ws*, *p*): // *ws*: Wurzelseite, *p*: ein Punkt //

```
if ws=null then begin
     erzeuge neue P-Seite page;
     füge p in page ein;
     setze ws := page
end
else begin
    führe Exact Match Query nach p durch;
    Ergebnis: Verweis auf Seite page, in die p eingefügt werden soll;
    if p schon in page then begin Sonderbehandlung; return end;
    füge p in page ein;
    while Überlauf von page do ...
end;
```

Einfügen von Datensätzen

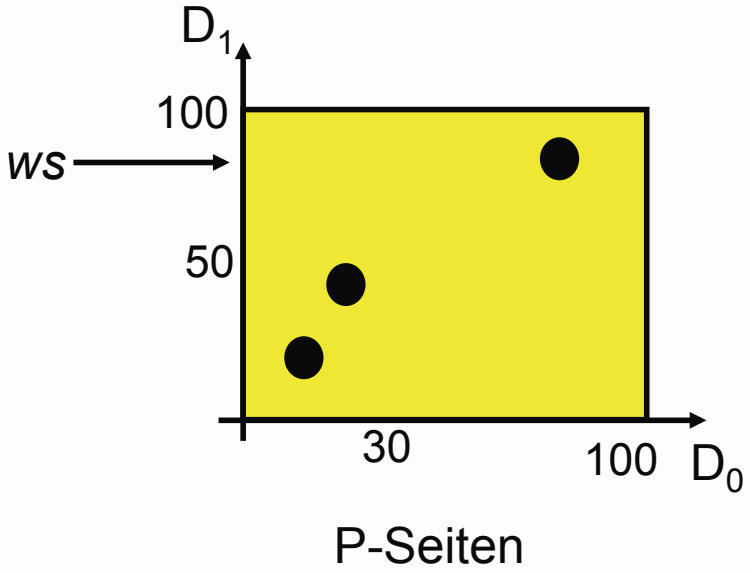
while Überlauf von page **do begin**
  Wähle (erneut) eine Dimension D_i und einen Wert $x_i \in D_i$ so, dass
  durch das Splitten von page entlang x_i zwei nicht volle Seiten entstehen;
  (leftpage, rightpage) := Split (page, x_i)
  if page = Wurzel-Seite *ws* **then begin**
    erzeuge neue R-Seite newpage mit den Einträgen
      ($D_0 \times \ldots \times [-\infty_i, x_i) \times \ldots \times D_{k-1}$, leftpage)
      und ($D_0 \times \ldots \times [x_i, \infty_i) \times \ldots \times D_{k-1}$, rightpage);
    setze ws := newpage;
    return
  end else begin
    Sei parentpage die Elternseite von page;
    Ersetze in parentpage den Eintrag (region, page) durch die Einträge
      (leftregion(region, x_i), leftpage)
      und (rightregion(region, x_i), rightpage);
    page := parentpage
  end
end;

Splits aufgrund von Überläufen können sich wie in B-Bäumen bis zur Wurzel fortsetzen.

Einfügen von Sätzen

D_0=D_1=[0,..., 100), Seitenkapazität = 3

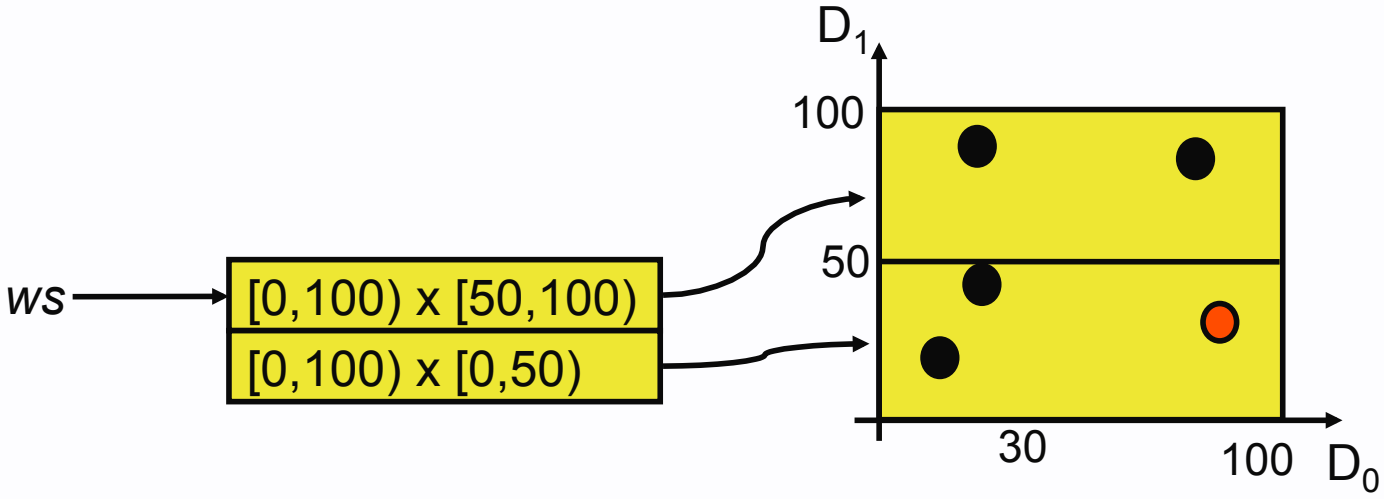
Insert(ws,(75,80))
Insert(ws,(5,15))
Insert(ws,(20,40))



Einfügen von Sätzen

D_0=D_1=[0,..., 100), Seitenkapazität = 3

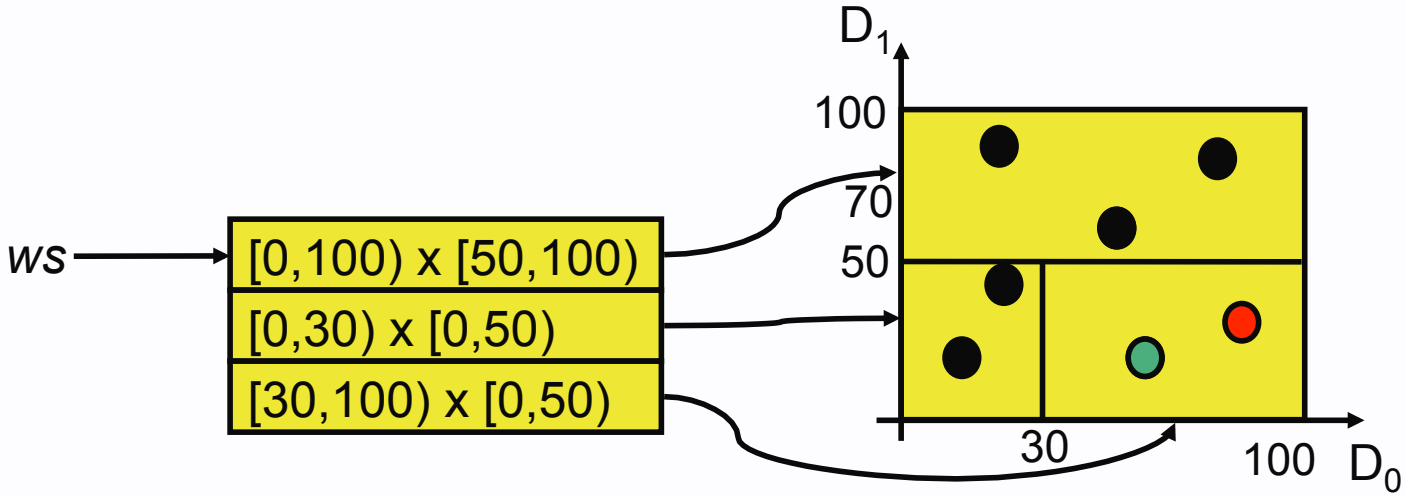
Insert(ws,(75,80))
Insert(ws,(5,15))
Insert(ws,(20,40))
Insert(ws,(80,30))
Insert(ws,(20,90))



Einfügen von Sätzen

D_0=D_1=[0,..., 100), Seitenkapazität = 3

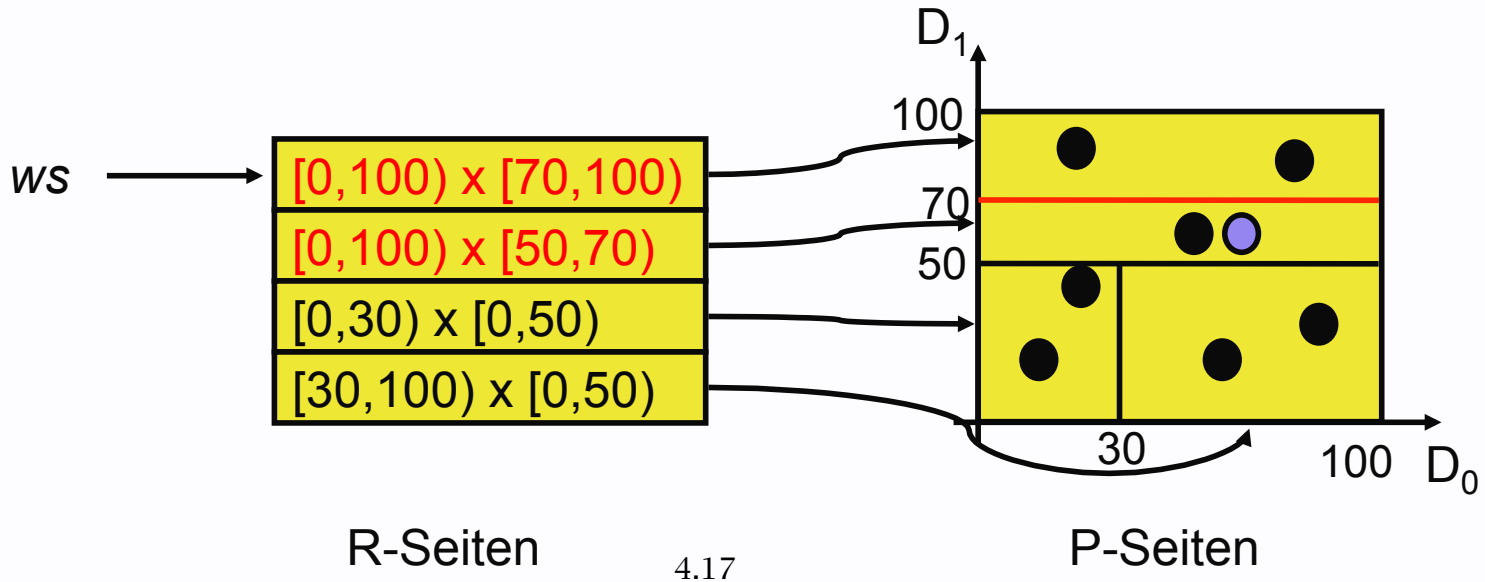
Insert(ws,(75,80))
Insert(ws,(5,15))
Insert(ws,(20,40))
Insert(ws,(80,30))
Insert(ws,(20,90))
Insert(ws,(55,20))
Insert(ws,(50,60))



Einfügen von Sätzen

D_0=D_1=[0,..., 100), Seitenkapazität = 3

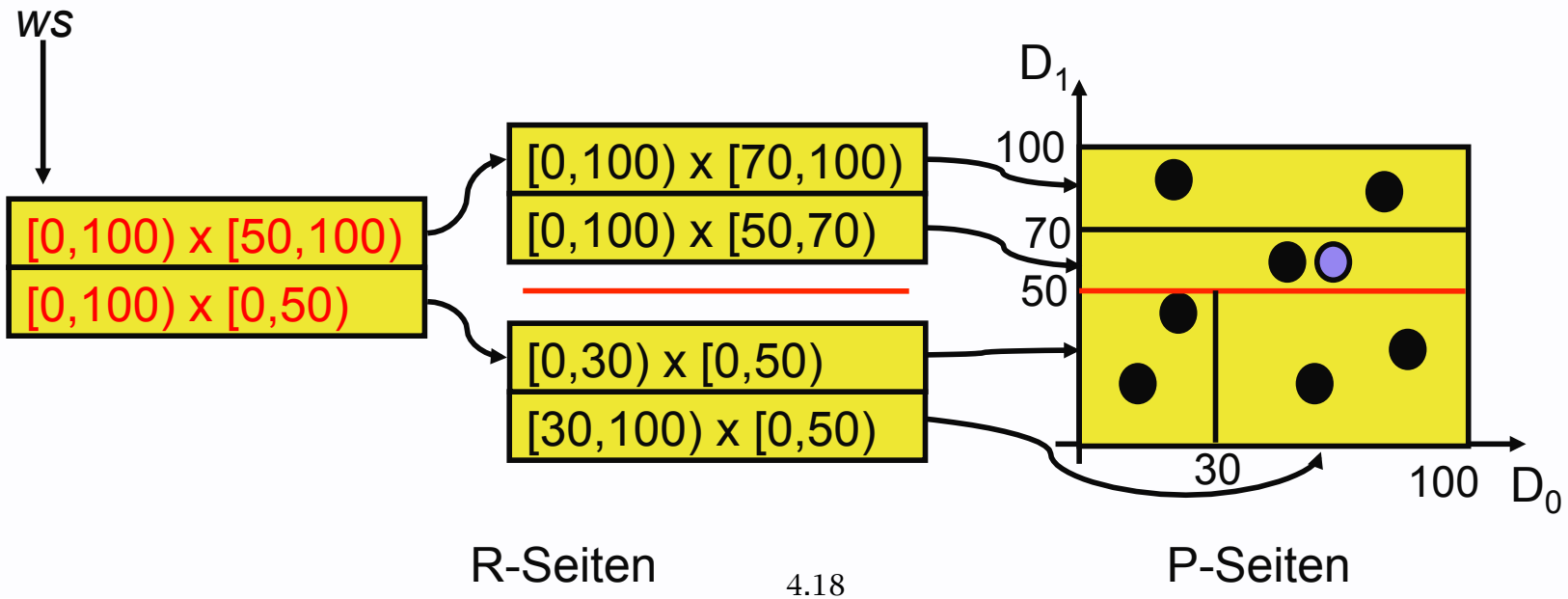
...
Insert(ws,(60,60))



Einfügen von Sätzen

D_0=D_1=[0,..., 100), Seitenkapazität = 3

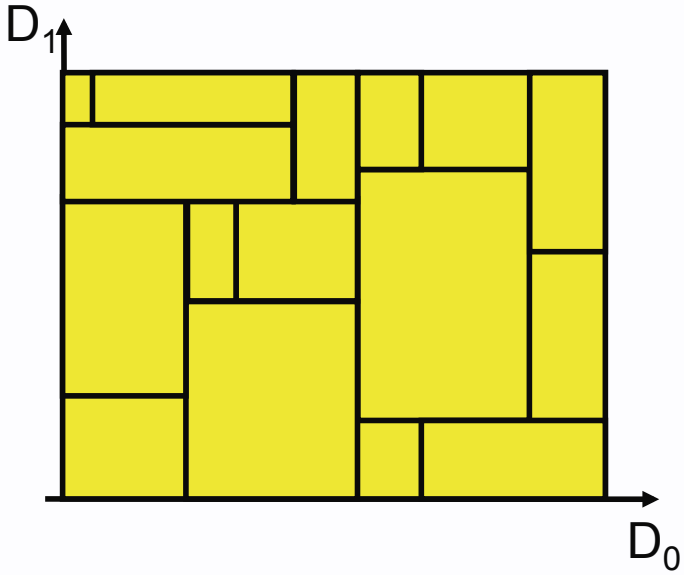
...
Insert(ws,(60,60))



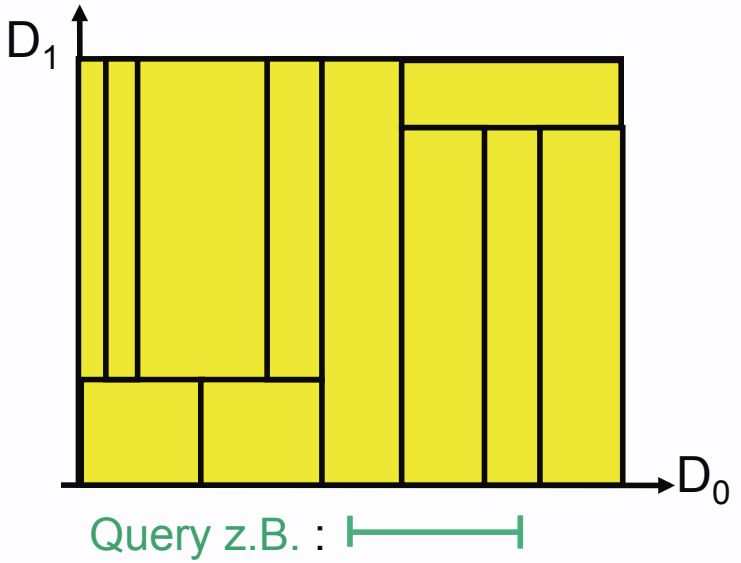
Strategien für das Spalten

Aufteilung des Datenraums:

Zyklisches Spalten von Seiten:



Wenn Partial Match Queries auf D_0 bevorzugt unterstützt werden sollen: Spalten entlang D_0 wann immer möglich.



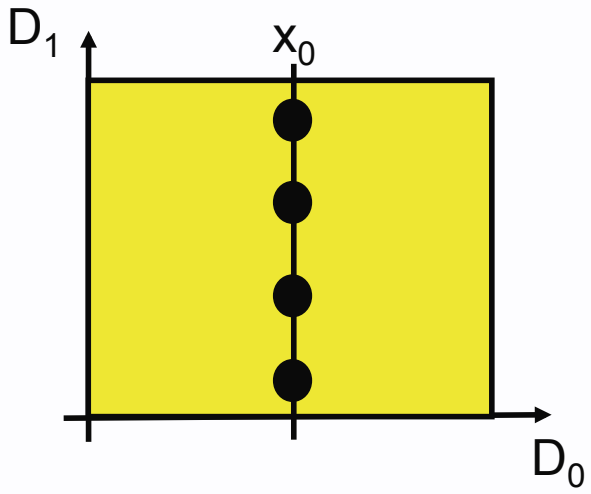
Falls die Verteilung der Daten in $D = D_0 \times \ldots \times D_{k-1}$ bekannt ist: Definiere die Länge eines Intervalls $I_i \subset D_i$ als Häufigkeit, dass die Komponente k_i eines Schlüssels $k \in D$ in I_i liegt. Wähle beim Splitten einer Region $I_0 \times \ldots \times I_{k-1}$ immer das/die in diesem Sinne längste Intervall/Dimension.

Strategien für das Spalten

Auswahl eines Wertes x_i :

- Wähle x_i so, dass in der zu splittenden Seite gleich viele Punkte links und rechts von x_i liegen, z.B. den Median der vorkommenden x-Koordinaten.

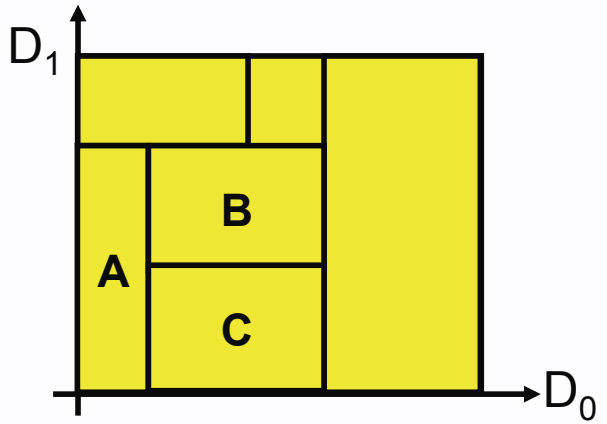
- u.U. muss eine Neuwahl der Spalt-Dimension erfolgen:



Eigenschaften des k-d-B-Baumes

- Bearbeitung von Range Queries: Rekursives Durchlaufen der Äste des Baumes, deren Knoten Regionen enthalten, die geschnitten mit der Anfrage-Region nicht die leere Menge ergeben.
- Für den Seitenfüllgrad kann im k-d-B-Baum keine untere Grenze garantiert werden.
- Um die Degeneration des Baumes zu vermeiden, muss er reorganisiert werden.
- Es existieren mehrere mögliche Reorganisations-Maßnahmen:
  - *Beispiel:* Zusammenlegen benachbarter Seiten.
  - *Problem:* Die Vereinigung der Regionen der beiden Seiten muss wieder eine Region ergeben. Dies ist nicht immer der Fall!

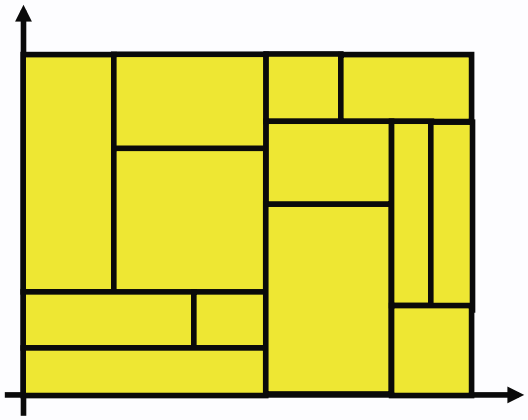
A *kann weder mit* B *noch mit* C *zusammengelegt werden.*



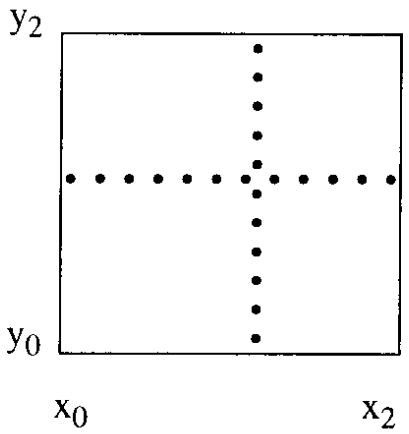
Einordnung

Mehrdimensionale Zugriffsstrukturen können gemäß der Art der Aufteilung des Datenraums in Gebiete charakterisiert werden, z.B. k-d-B-Baum:

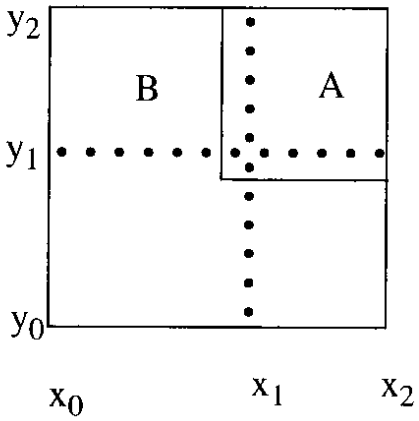
1. atomare Gebiete (beschreibbar durch je ein Rechteck)
2. vollständig (die Vereinigung aller Gebiete ergibt den gesamten Datenraum)
3. disjunkt (die Gebiete überlappen nicht)

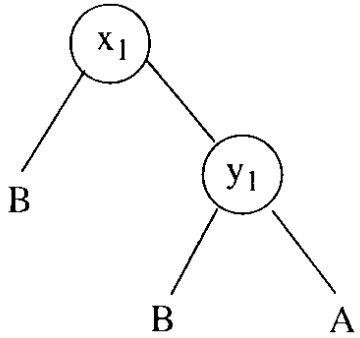


- **hB-Baum**
  - Variante von k-d-B-Bäumen
  - Indexknoten sind als k-d-Bäume (bzw. -DAGs) organisiert
  - Aufspaltung kann mehr als eine Dimension einbeziehen
    ⇒ nicht nur Aufteilung in Rechtecke (quasi in Backsteine/*bricks*),
    sondern auch Vereinigung oder Differenz von Rechtecken
    (quasi Backsteine mit Löchern/*holed bricks*)



a) Split-Linie bei Punktmengen
auf orthogonalen Linien?





b) „Backstein mit Loch“
und der zugehörige modifizierte k-d-Baum

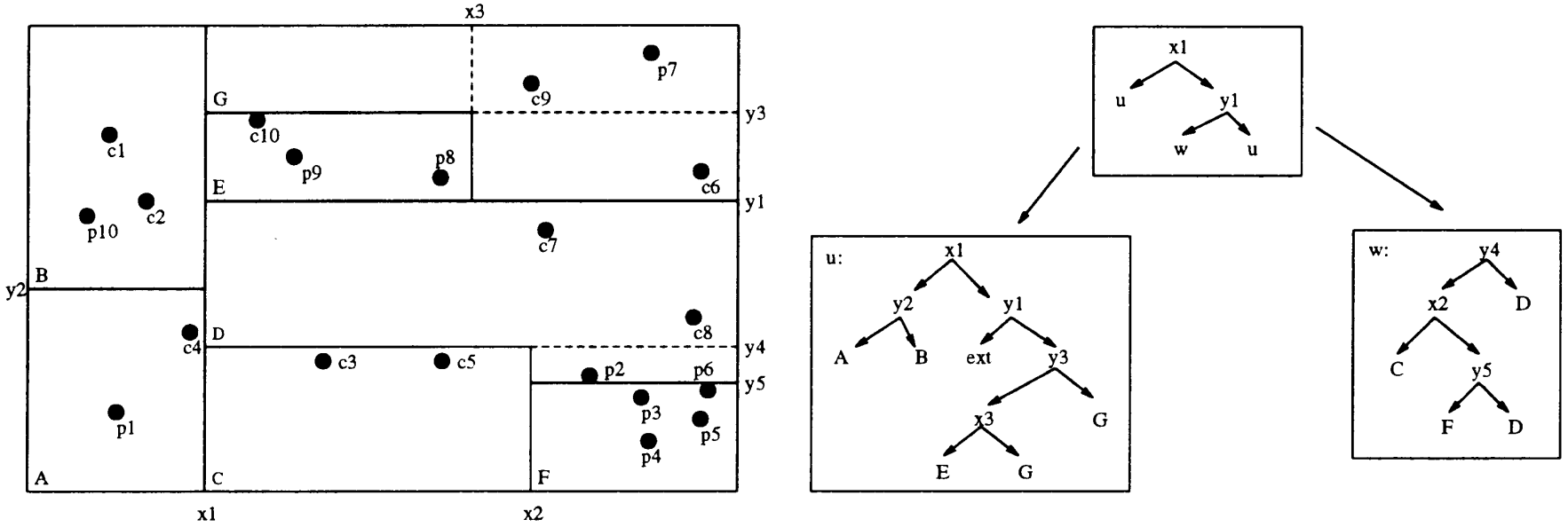


Figure 23. hB-tree.

- geänderter Split-Vorgang
  (siehe folg. Abb.; 'ext' = Verweis, der zum dargestellten Teilbereich "extern" ist)
- garantiert im worst case (1/3 : 2/3)-Aufteilung
- keine forced splits mehr

⇒ guter Kandidat für Einsatz in DBS

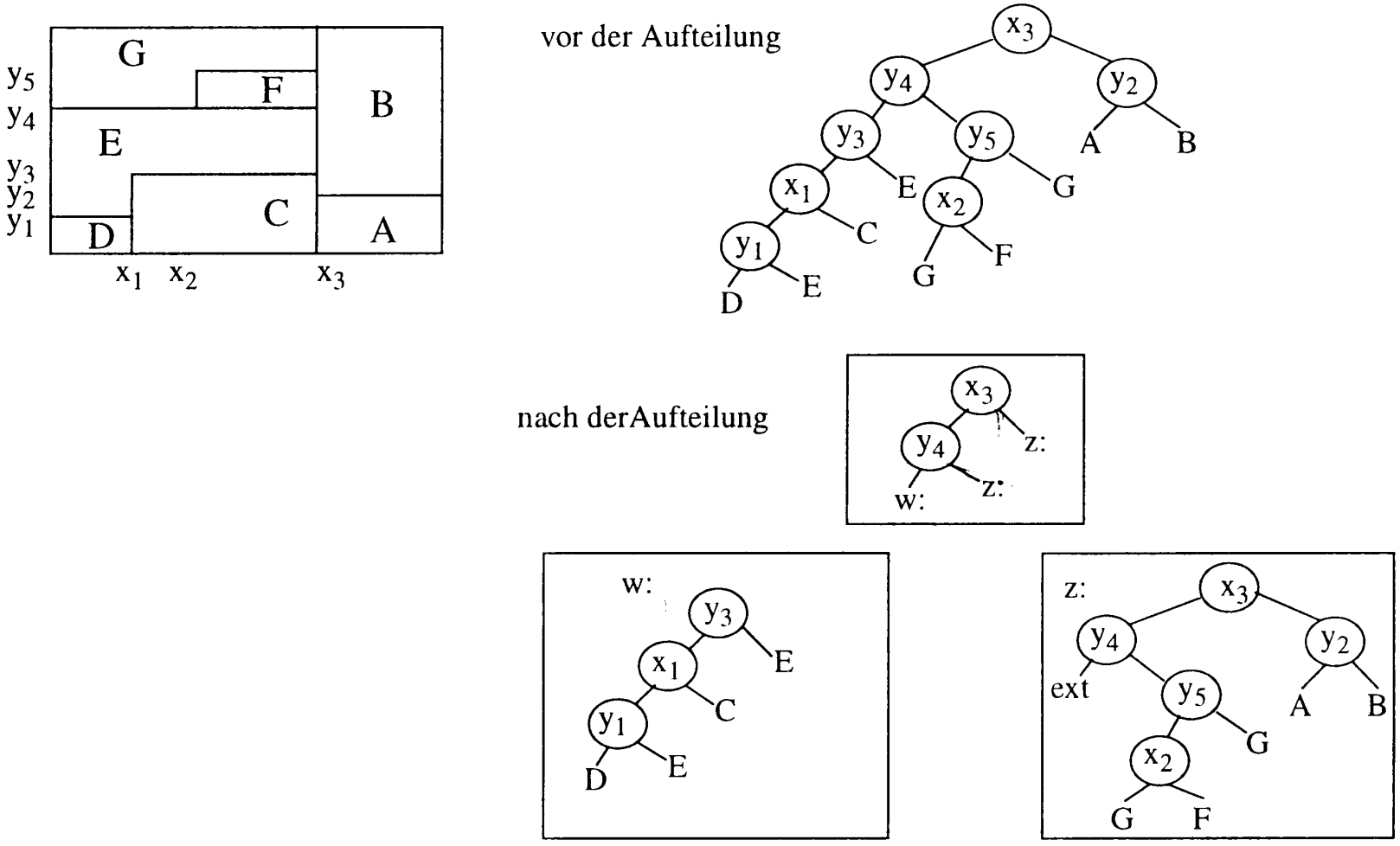


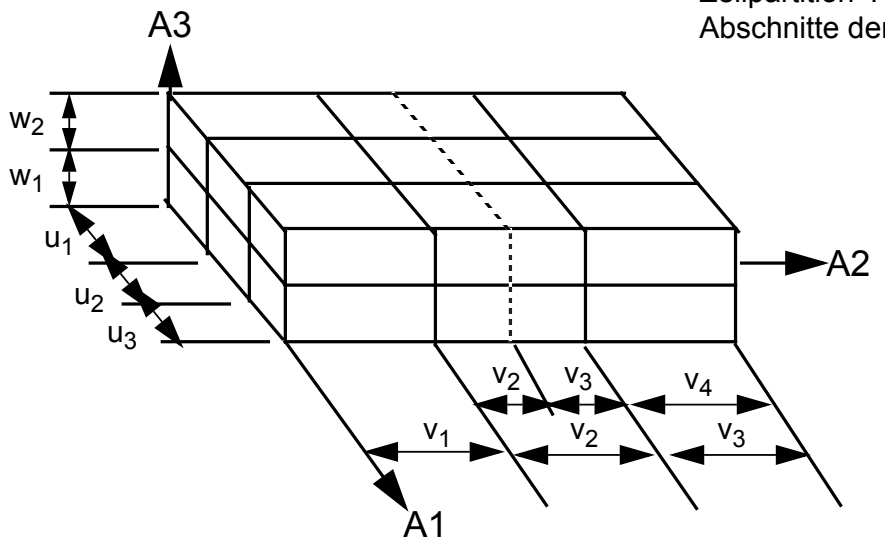
Abb. 9.12: Flexibler Split-Vorgang in einem hB-Baum

B) 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 Buckets zugeordnet.
  - 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 für Dimensionsverfeinerung**

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



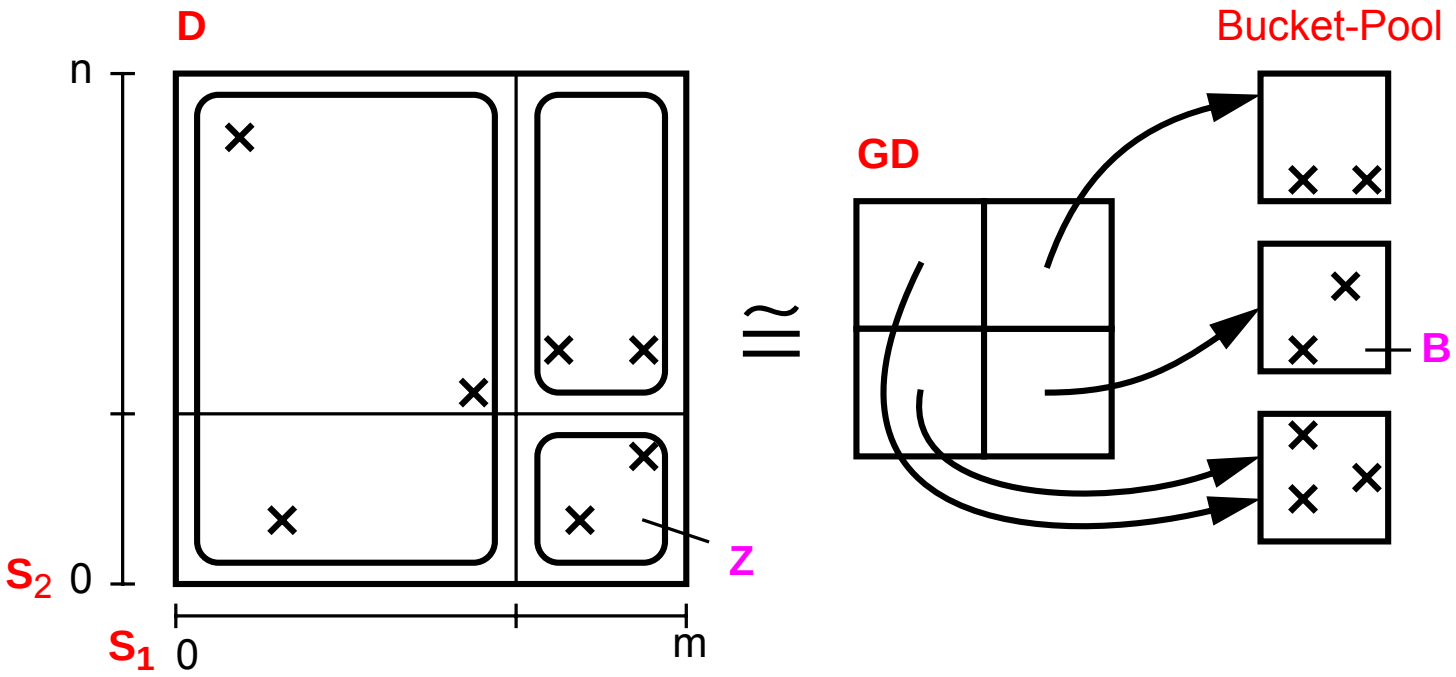
Dreidimensionaler Datenraum D mit Zellpartition P;
Veranschaulichung eines Split-Vorganges im Intervall v

- **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?

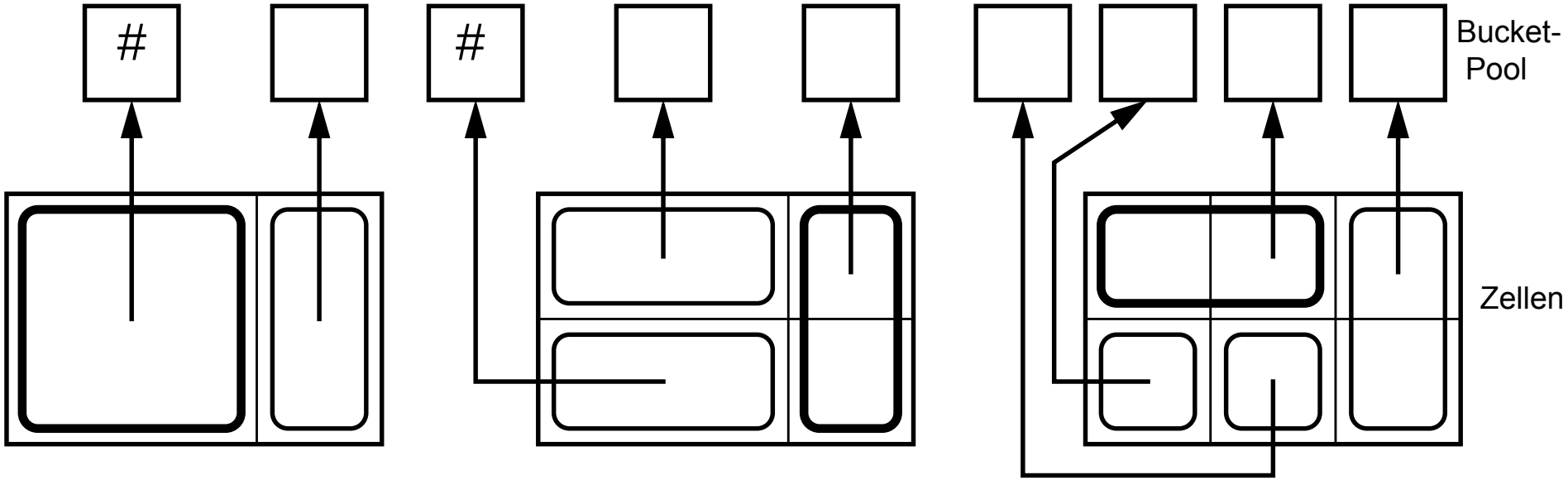
Grid-Files: Idee

- **Zerlegungsprinzip von D: Dimensionsverfeinerung**



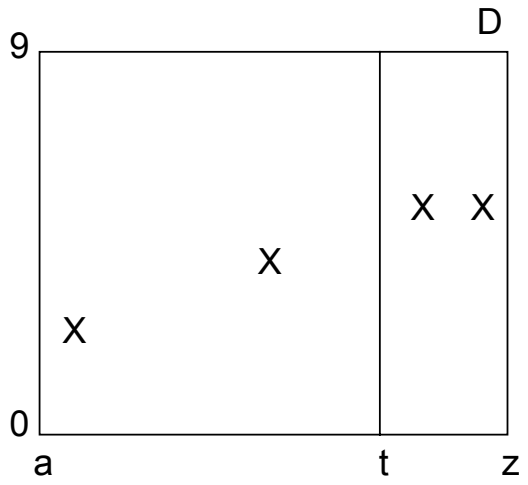
- **Komponenten**
  - k Skalierungsvektoren (Scales) S_i definieren die Zellen (Grid) auf dem k-dim. Datenraum D
  - Zell- oder Grid-Directory GD: dynamische k-dim. Matrix zur Abbildung von D auf die Menge der Buckets
  - Bucket: Speicherung der Objekte einer oder mehrerer Zellen (Bucket-Pool)
- **Eigenschaften**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

Schachtelförmige Zuweisung von Zellen zu Buckets

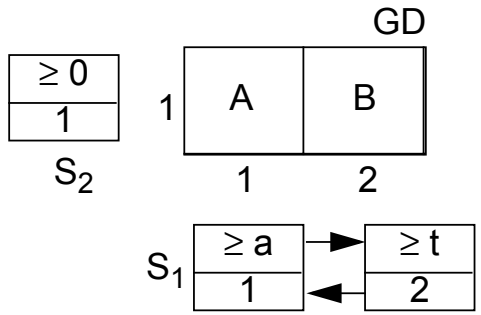


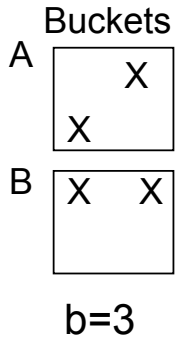
(zeigt auch Entwicklung des Grid-Files bei fortgesetzten Einfügungen in Buckets #)

Schrittweise Entwicklung eines Grid-Files - am Beispiel



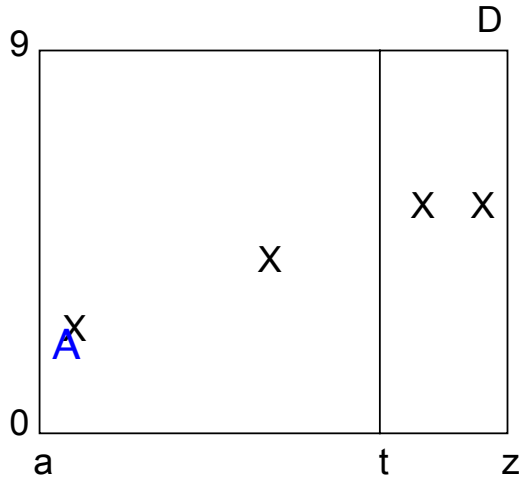
Situation a

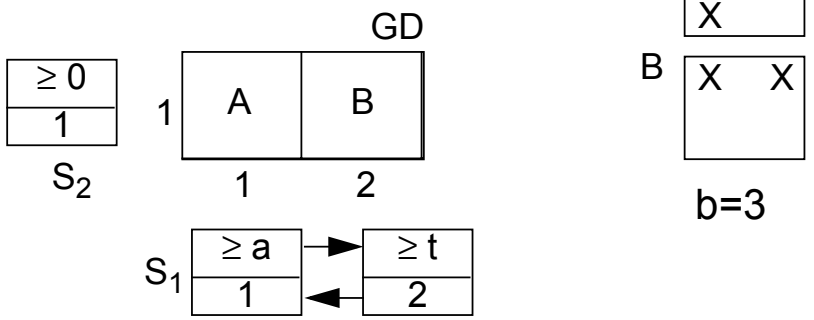




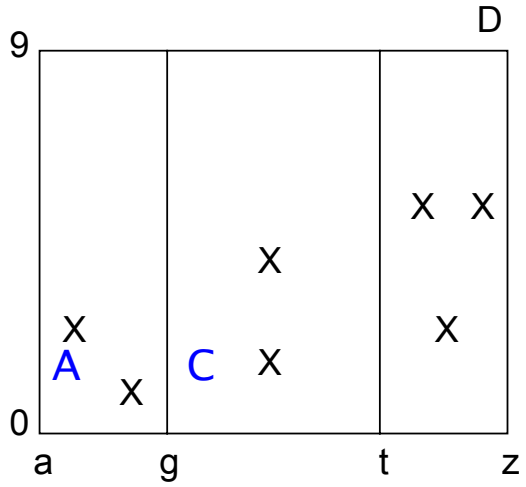
beachte: Skalierungsvektoren als zweifach verkettete Listen

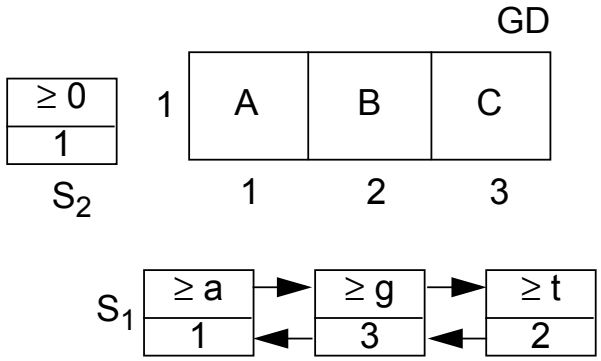
- Indirektion erlaubt es, das GD an den Rändern wachsen zu lassen.
- Minimierung des Änderungsdienstes von GD
- Stabilität der GD-Einträge

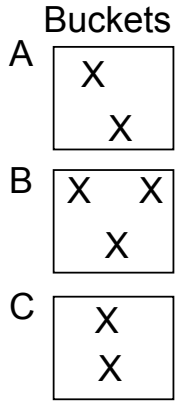




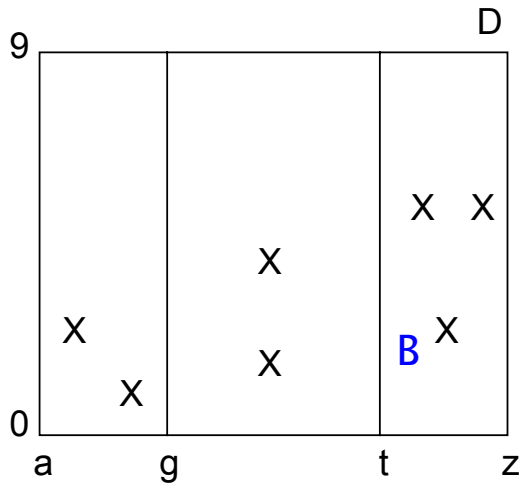
Situation a

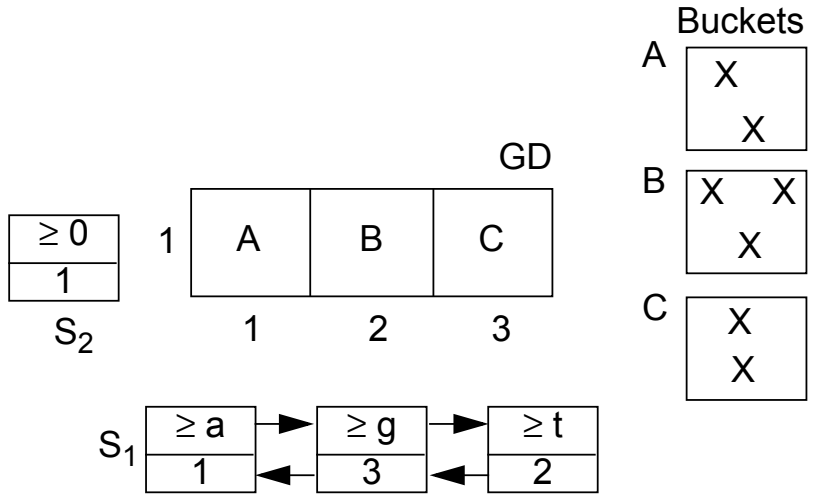




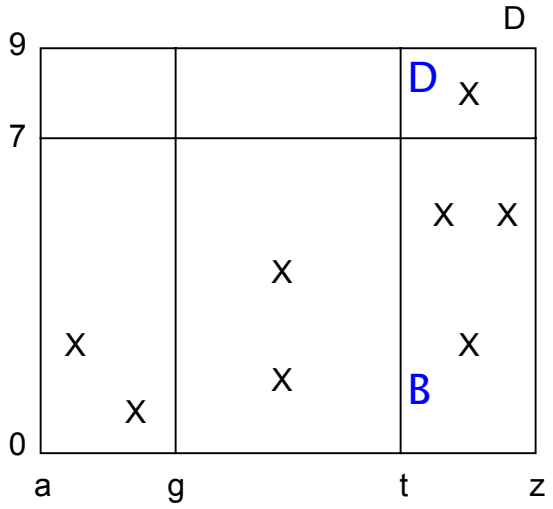


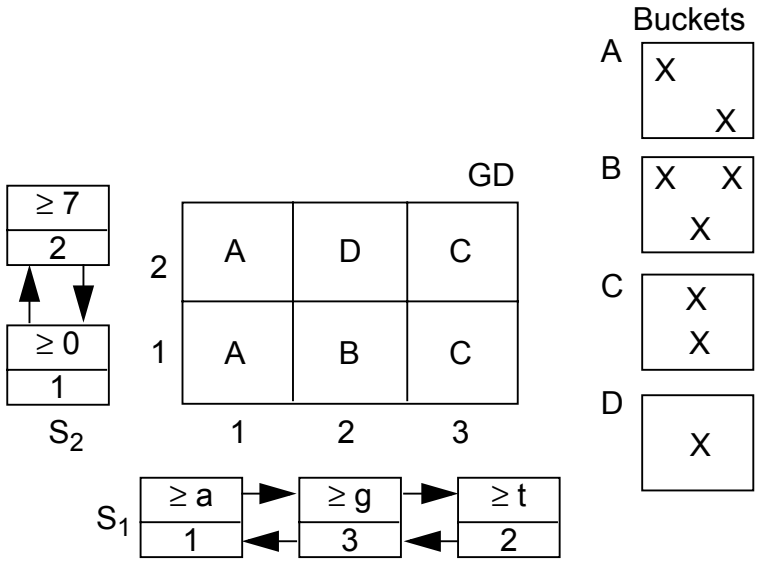
Situation b



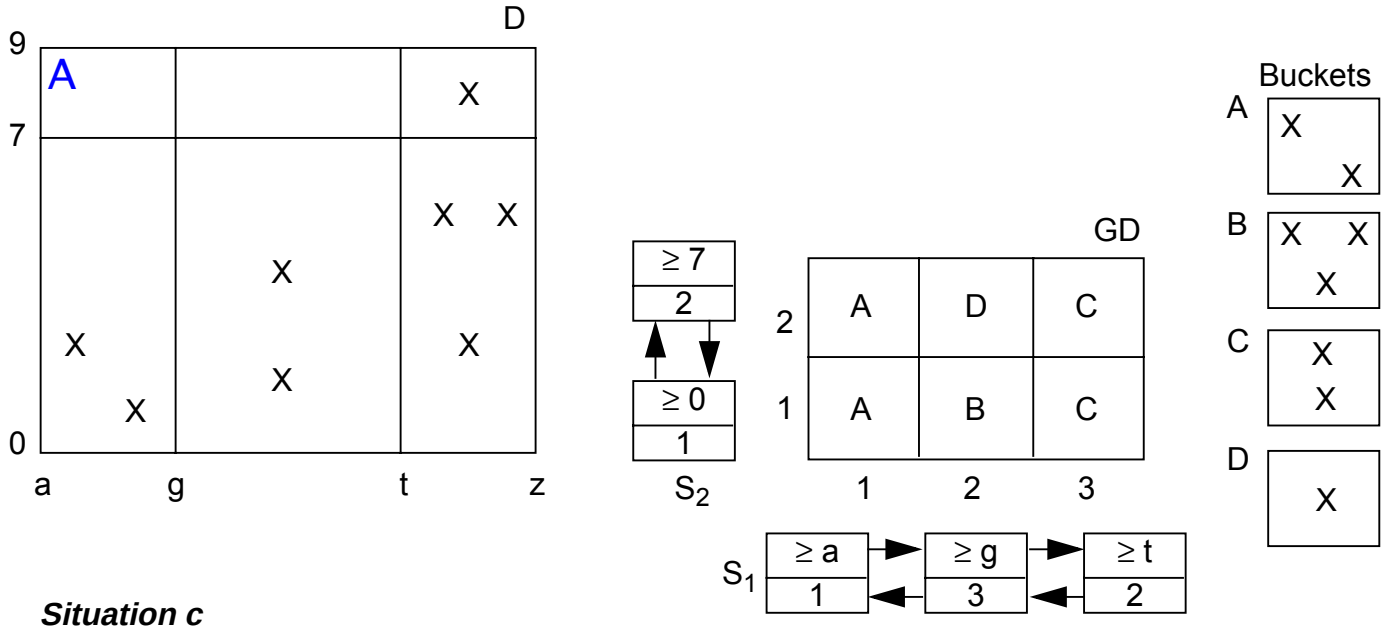


Situation b

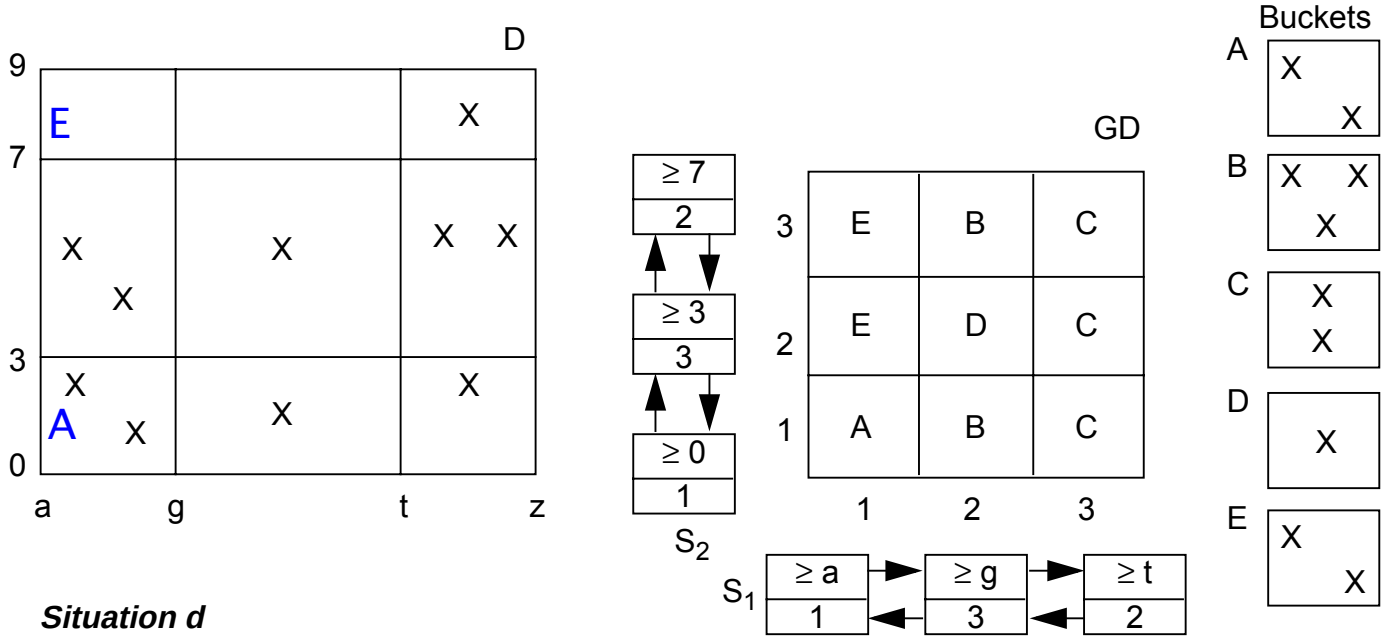




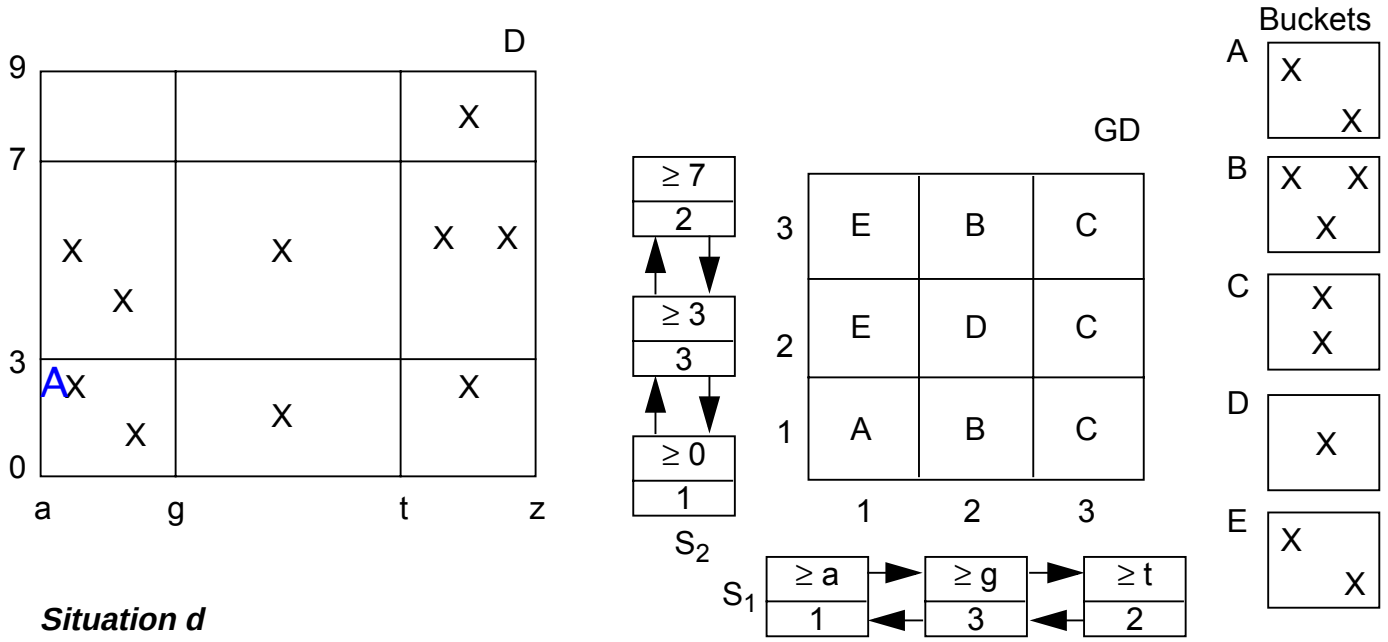
Situation c



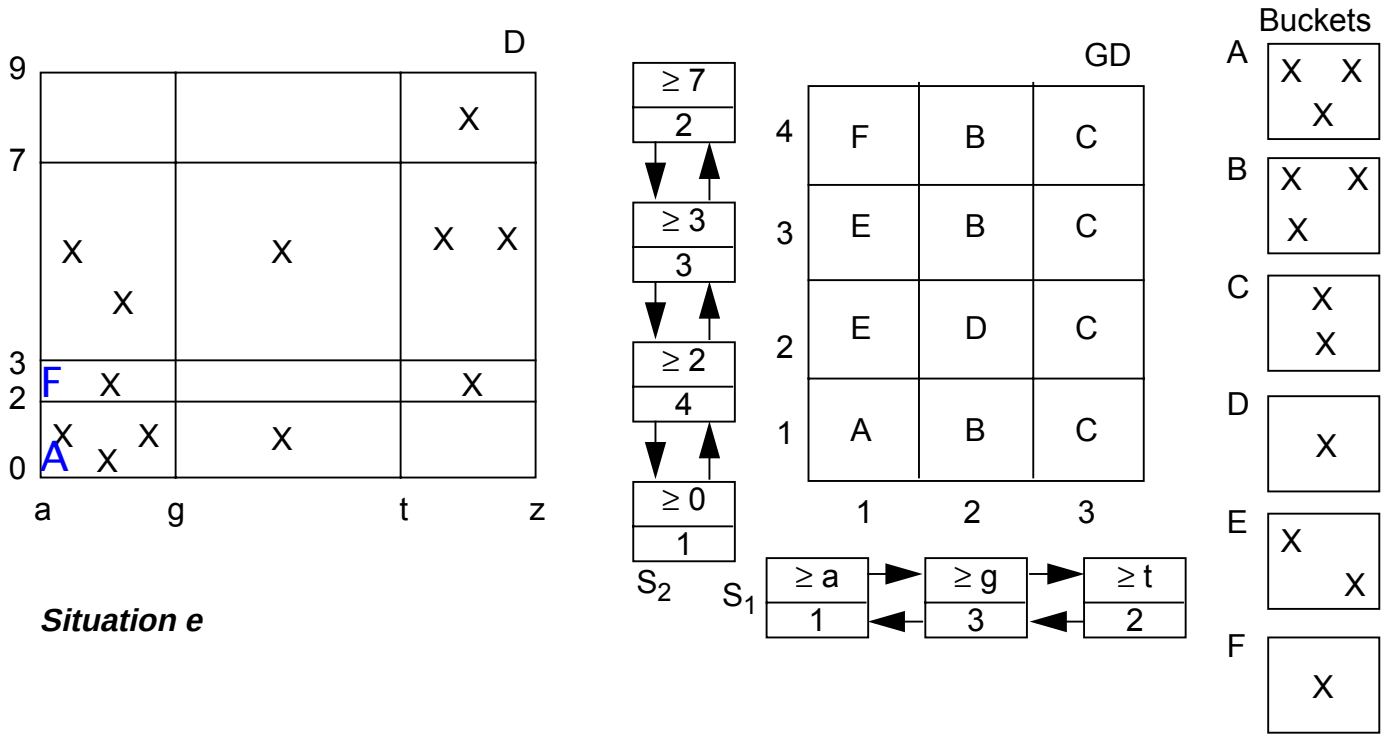
Situation c



Situation d



Situation d



Situation e

- **Charakteristika**
  - Prinzip der zwei Plattenzugriffe:
    unabhängig von Werteverteilungen, Operationshäufigkeiten
    und Anzahl der gespeicherten Sätze
  - Split- und Mischoperationen jeweils nur auf 2 Buckets (+1 Directoryseite)
  - Speicherplatzbelegung
    - durchschnittliche Belegung der Buckets nicht beliebig klein
    - schiefe Verteilungen vergrößern nur GD

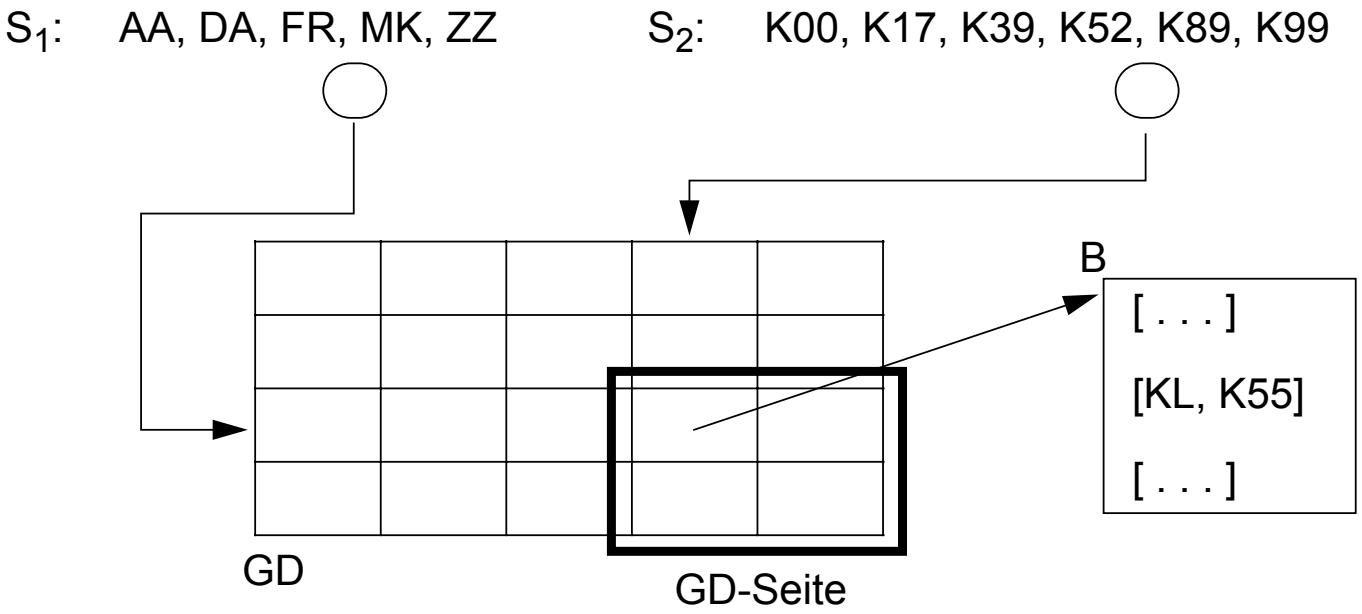
- **Speicherung:**
  - dynamische k-dim. Matrix GD (auf Externspeicher; seitenweise einlagern)
  - k eindim. Vektoren S_i (unbedingt im Hauptspeicher!)

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

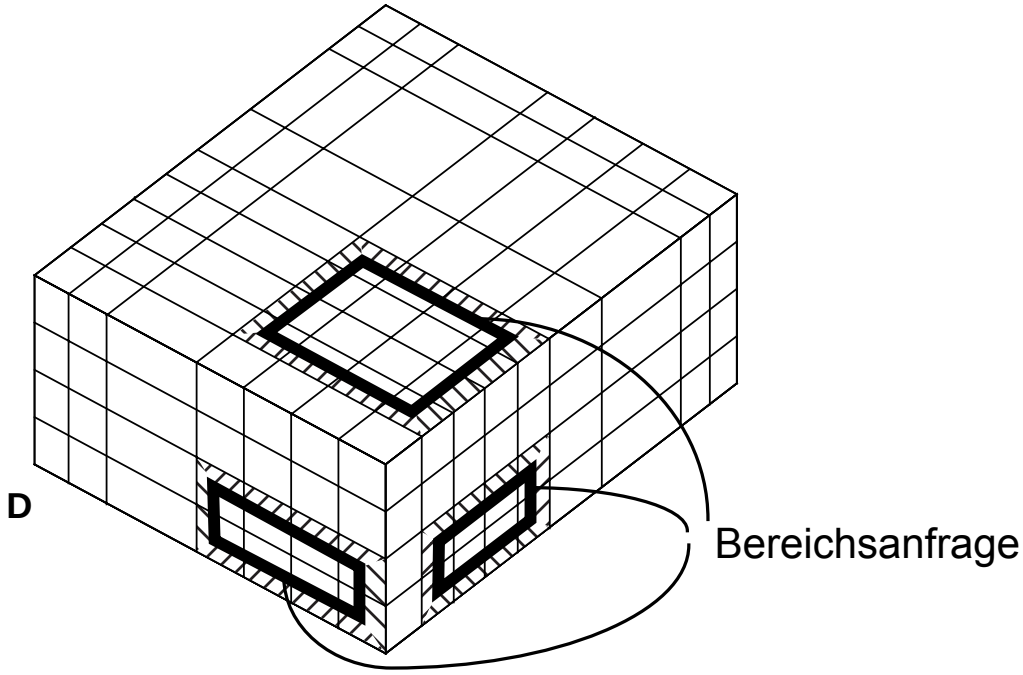
Grid-File - Suchfragen

- **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



Weitere Anmerkungen zu Grid-Files

- *Vorsicht:* Die obige Beispielserie zeigt in jedem Schritt eine (teure !) Dimensionsverfeinerung. Dies ist kein typisches Verhalten.

  Häufig brauchen nach Einfügungen die Buckets gar nicht oder nur innerhalb der schon vorhandenen Directory-Struktur gesplittet werden (nur Splitten von Zellen, aber keine Dimensionsverfeinerung, hier z.B. nach weiteren Einfügungen in C).

- Verfeinerungen werden durch eine einzige Zelle verursacht und können für benachbarte Zellen auch in der Zukunft nutzlos bleiben.

- Grid-Directories wachsen bei gleichförmigen Verteilungen "nur" superlinear ($O(N^{1+\frac{k-1}{k \cdot \text{Bucketgröße}}})$), aber im worst case polynomial ($O(N^k)$), falls $N = \#\text{Datensätze}$.

- Die Auswahl von Verfeinerungsdimensionen und -grenzen kann analog zu Splitdimensionen und -grenzen in k-d-B-Bäumen erfolgen:
  - z.B. alternierend, "längstes" Intervall
  - z.B. Mitte oder lokaler Median

- Falls Datensätze in Buckets nur durch Zeiger (TIDs) repräsentiert werden, entstehen bei der Auswertung von Bereichsanfragen i.a. zu grosse Obermengen, die viele Extra-Seitenzugriffe zur Überprüfung benötigen. Deshalb sollten die Buckets die originale Schlüsselinformation mit enthalten.

Weitere Anmerkungen zu Grid-Files (Forts.)

- Das Verschmelzen von Buckets (und ggfs. von Zellen) funktioniert idealerweise mit dem früheren Splitpartner ("Buddy"), falls dieser noch nicht aufgeteilt wurde. Dazu muss ein binärer Buddy-Baum mitgeführt und aktualisiert werden.

  Auch möglich, aber aufwändiger ist die Verschmelzung mit Nachbarzellen oder -buckets, so dass ein Rechteck entsteht.

  Dimensionsverfeinerungen werden normalerweise nicht zurückgenommen.

- Es gibt viele Weiterentwicklungen, u.a.:
  - Da eine Bereichsanfrage an ein Grid-File auf eine Bereichsanfrage an das Grid-Directory führt: Hierarchie von Grid-Files (**two-level grid files**)

    $\rightsquigarrow$ mehr Zugriffe im Directory-Baum vs. geringeres Directory-Wachstum
  - Zusammenspiel von zwei GridFiles mit abweichenden Partitionierungen, nur Mittensplits, ggfs. Datenverlagerungen zwischen Files (**twin grid files**)

    $\rightsquigarrow$ geringere #Buckets, bessere Speicherplatzausnutzung
  - wie hB-Baum keine rechteckigen Zellen (**BANG-File**, balanced nested grid file)

    $\rightsquigarrow$ besser an Objektverteilung anpassbar, nur linear wachsendes Directory, aber Suchen aufwändiger (max. #Seitenzugriffe = Directorygröße)

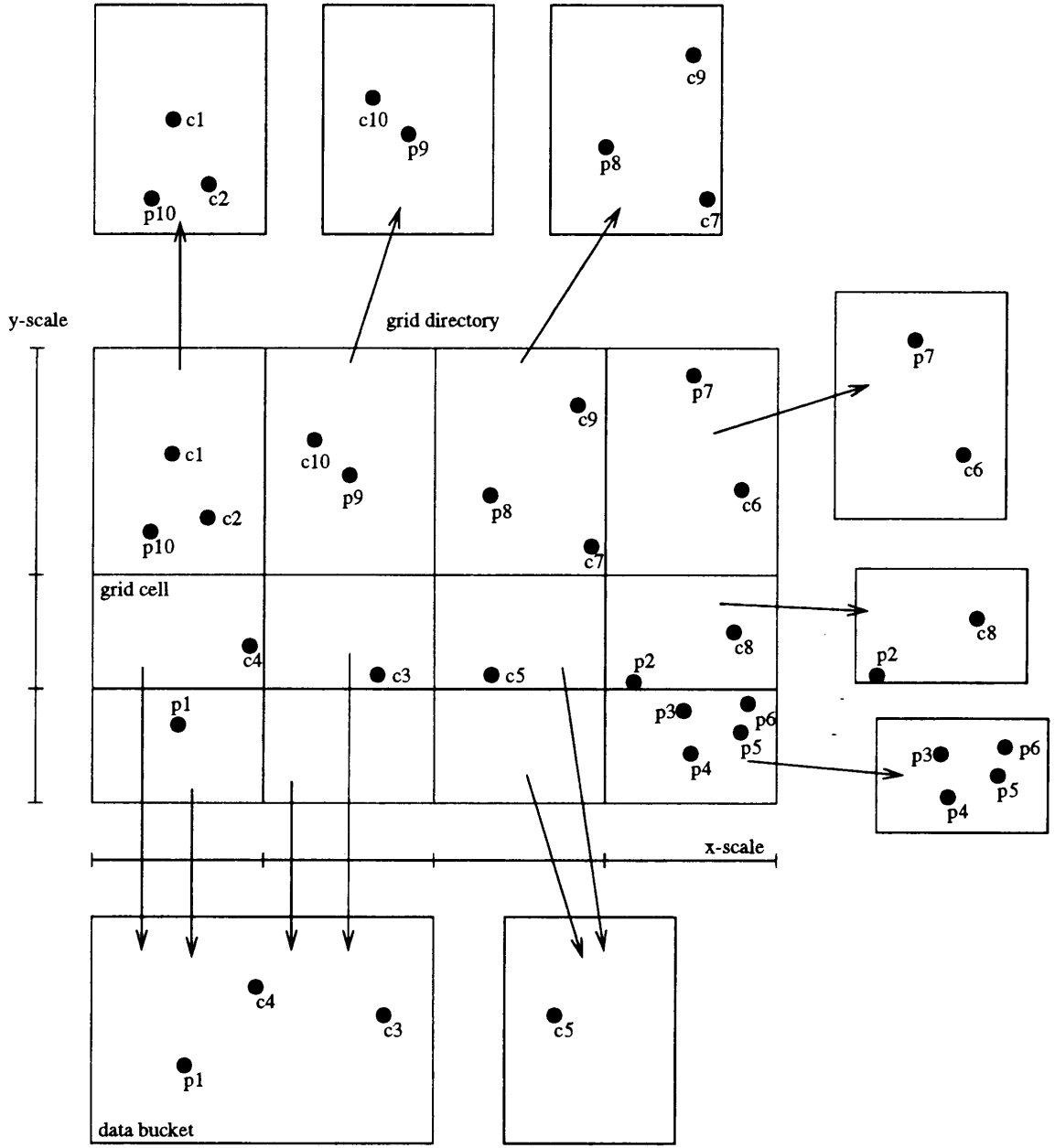


Figure 16. Grid file.

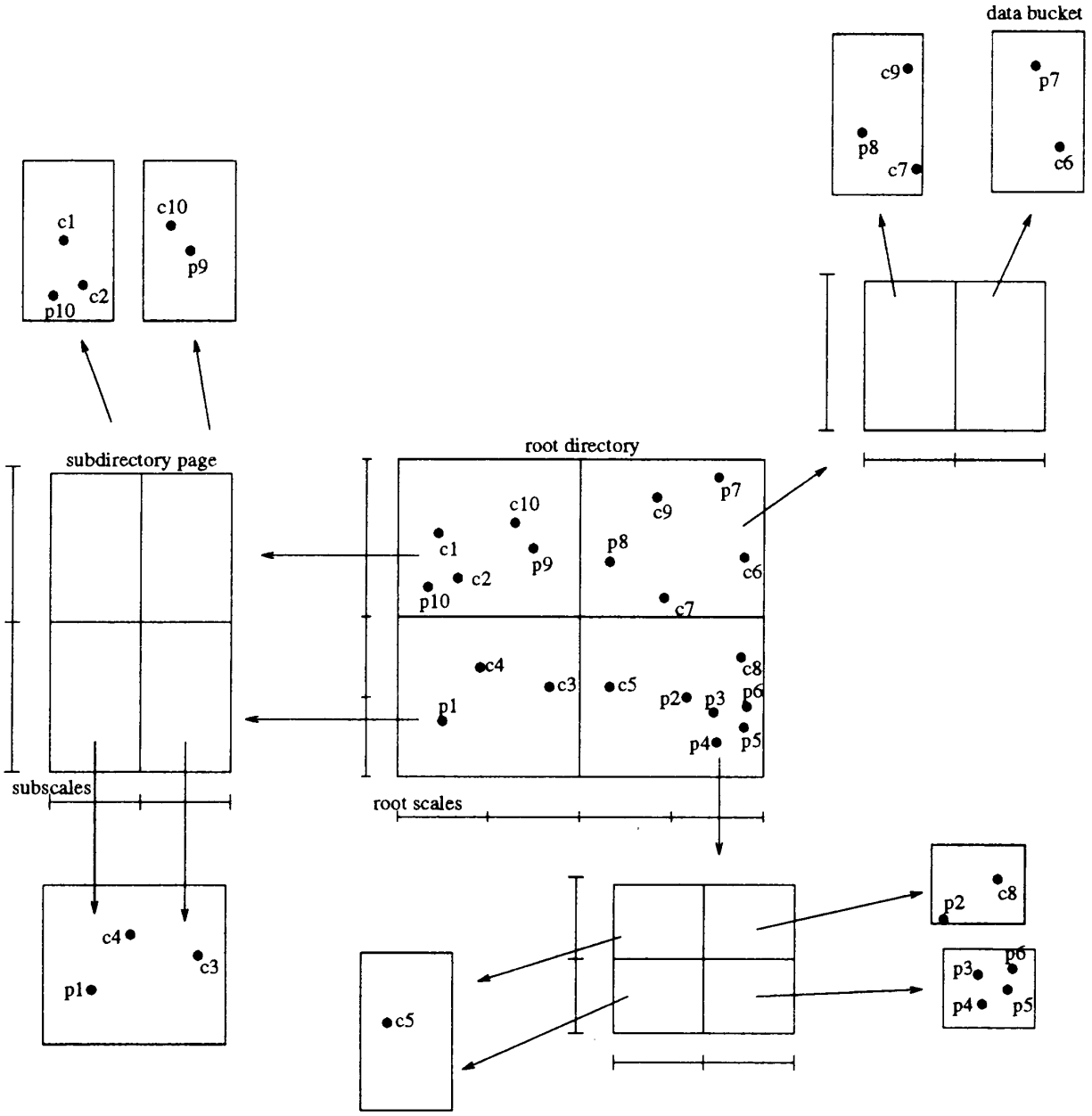


Figure 17. Two-level grid file.

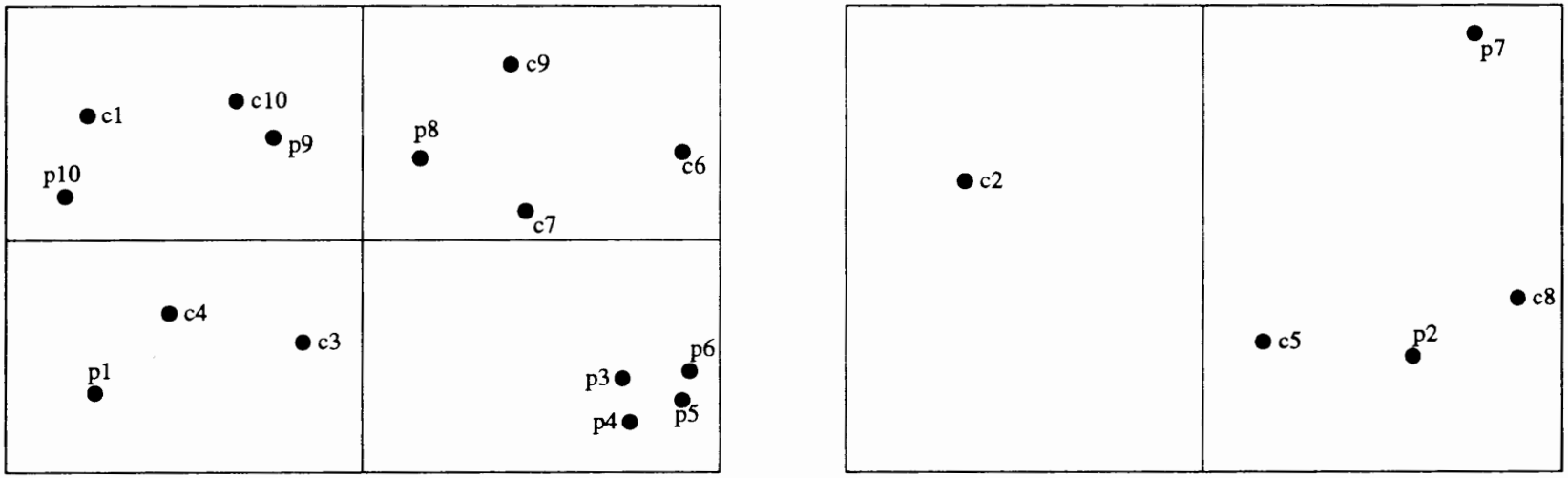
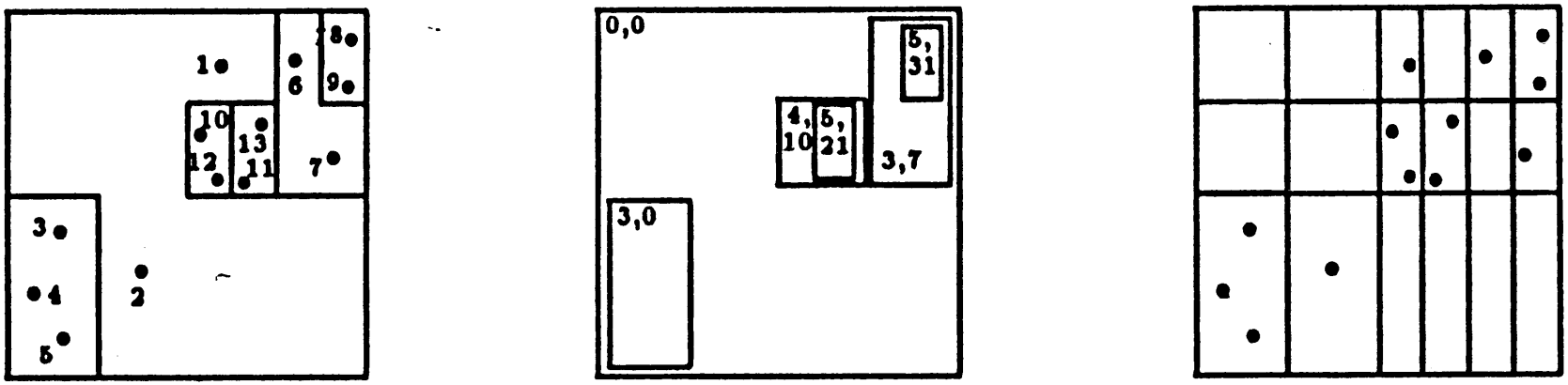


Figure 18. Twin grid file.



Eine BANG-File-Einteilung (a), ihre Verwaltung (b) und eine Gridfile-Einteilung (c) für $c = 3$

Alle Zellen werden fortgesetzt halbiert, bei alternierender Splitdimension.
$(i, j),\ 0 \le j < 2^i$, bezeichnet die j-te Zelle auf der i-ten Halbierungsstufe (auch darstellbar als Binärzahl j mit i Bits). Das Directory ist ein Baum von nichtleeren Zellen bis zur aktuell maximalen Halbierungsstufe, mit (0,0) als Wurzel und den jeweils enthaltenen nichtleeren Zellen als Nachfolger.
Eine Zelle (= Bucket) speichert alle in ihr enthaltenen Datensätze ohne die in den Nachfolgern enthaltenen Datensätze.