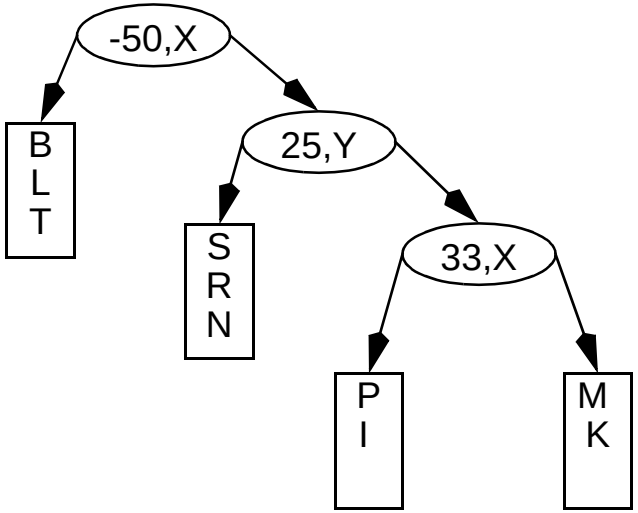
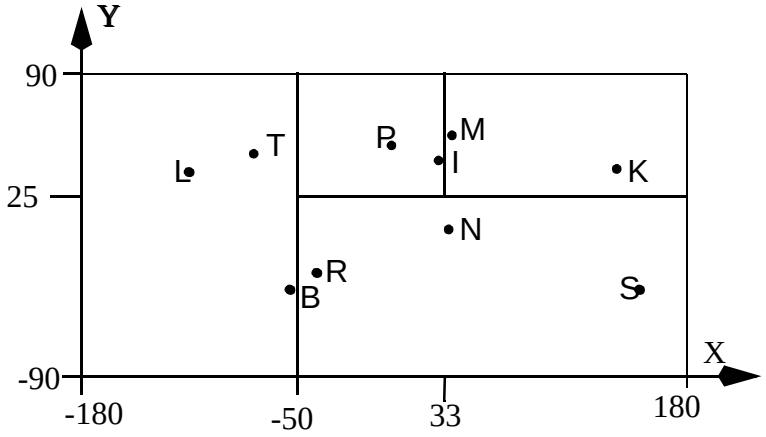
4.2.2 KDB-Baum

- basiert auf der Strategie des kd-Baums
- bildet die Zwischenknoten des kd-Baums auf Externspeicherseiten ab.

Beispiel:

- Annahme:
  höchstens 3 Referenzen können in einem Zwischenknoten des KDB-Baums sein

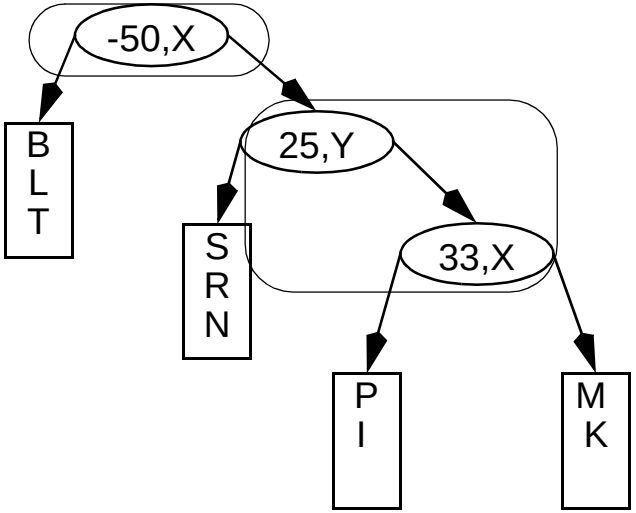




- Frage: Wie werden die 4 Seitenreferenzen auf zwei Seiten aufgeteilt?

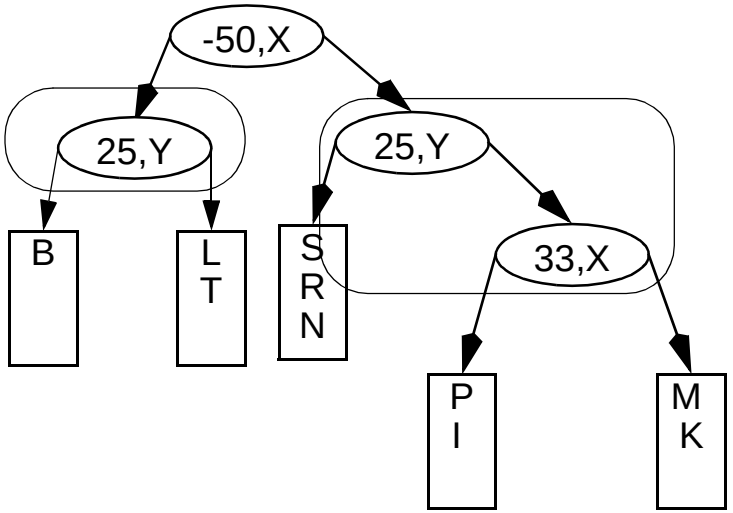
Lösung 1:
Benutze die Wurzel des kd-Baums für das Aufteilen der Referenzen:

- alle Referenzen im linken Teilbaum verbleiben in der alten Seite
- alle Referenzen im rechten Teilbaum werden in eine neue Seite umgespeichert
- Indexeintrag in der Wurzel des kd-Baums kommt in den Elternknoten.



- Nachteil: schiefe Aufteilung

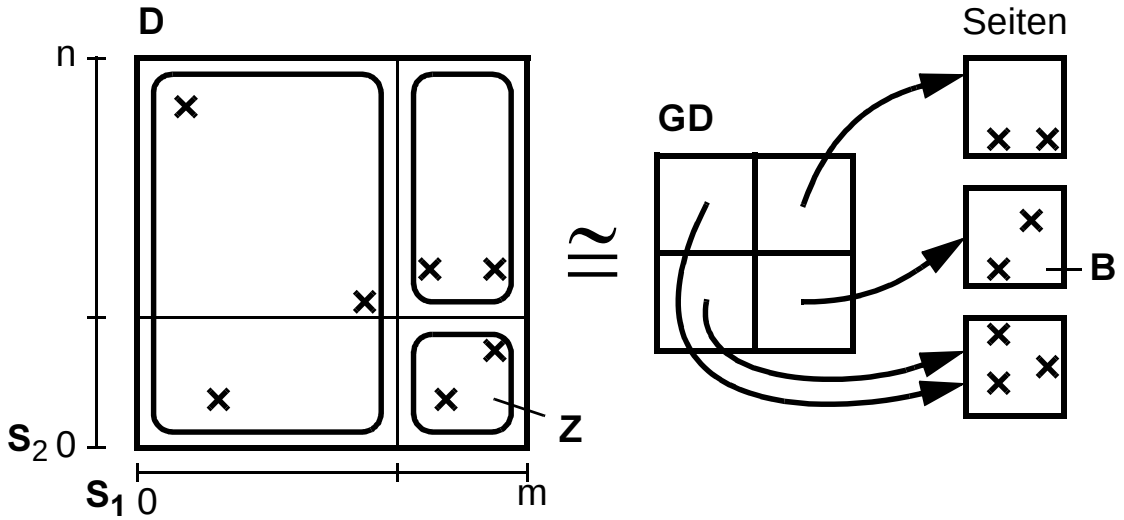
Lösung 2:
Erzeuge neue Knoten (und Referenzen), so daß eine gleichmäßige Aufteilung möglich ist.



- Nachteil: viele unterfüllte Seiten

4.2.3 Grid File

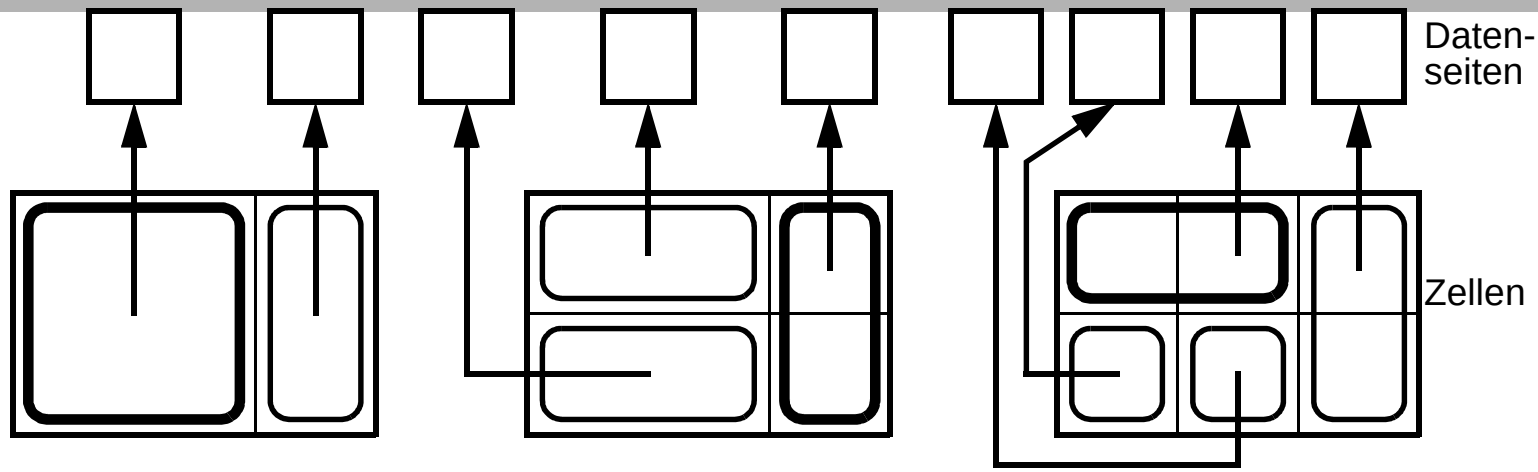
- verallgemeinert das Prinzip von EH für mehrdimensionale Daten
- Idee:



- 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 Seiten
  - Seiten: Speicherung der Objekte einer oder mehrerer Zellen (Seitenbereich SB)

- Eigenschaften
  - 1:1-Beziehung zwischen Zelle Z_i und Element von GD
  - Element von GD =Zelle zu Seite B
  - n:1-Beziehung zwischen Z_i und B
- Ziele
  - Erhaltung der Topologie
  - effiziente Unterstützung aller Fragetypen
  - vernünftige Speicherplatzbelegung
- Anforderungen
  - Prinzip der zwei Plattenzugriffe: unabhängig von Werteverteilungen, Operationshäufigkeiten und Anzahl der gespeicherten Sätze
  - Split- und Mischoperationen jeweils nur auf zwei Seiten
  - Speicherplatzbelegung:
    - durchschnittliche Belegung der Seiten 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)

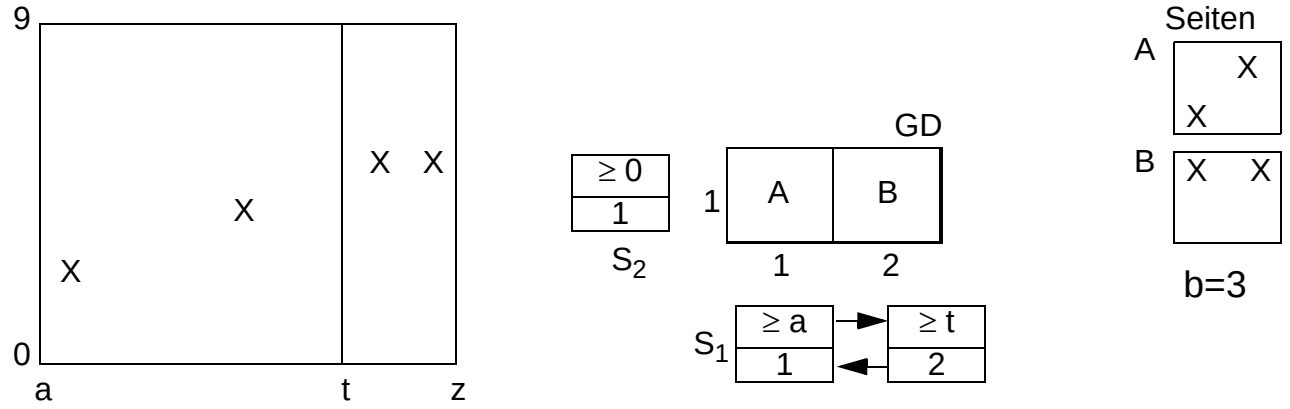
Zuweisung von Zellen zu Seiten



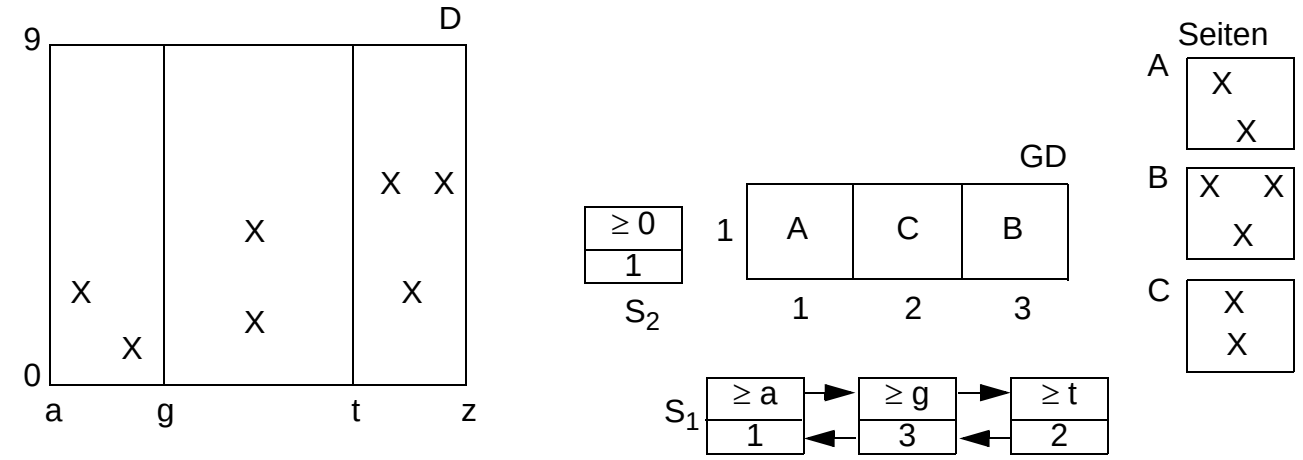
- Anforderung
  - Splitting überfüllter Seiten
  - Verschmelzen unterfüllter Seiten
- Seitenregionen (= Vereinigung aller Zellen einer Seite) sind rechteckig
  - diese Bedingung allein kann zu einer Partitionierung führen, die ein Verschmelzen von Seitenregionen nicht mehr unterstützt (Deadlock)
- Projektionen der Seitenregionen auf die Achsen sind binär konstruierbar
  - keine Deadlocks für k = 2

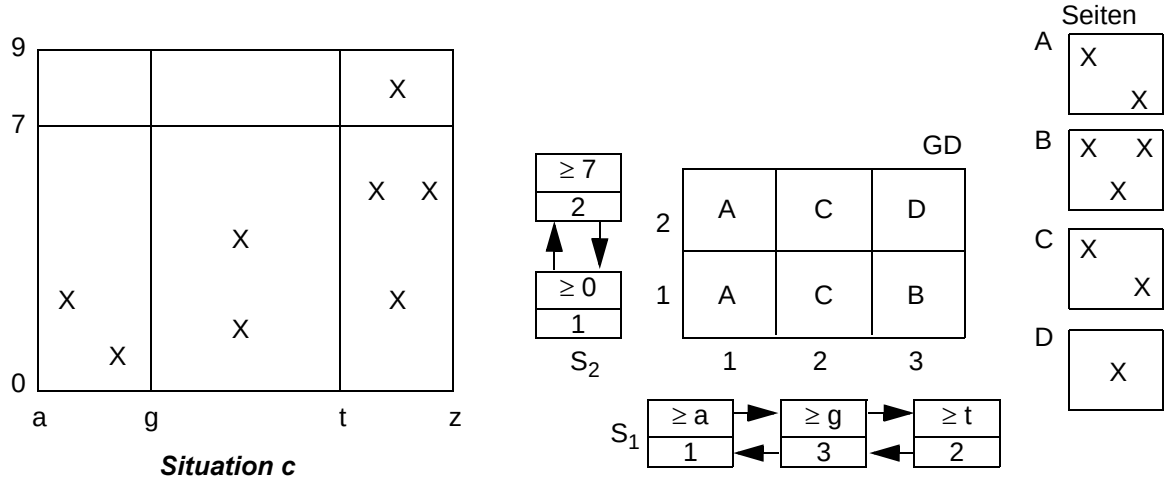
Grid File (Beispiel)

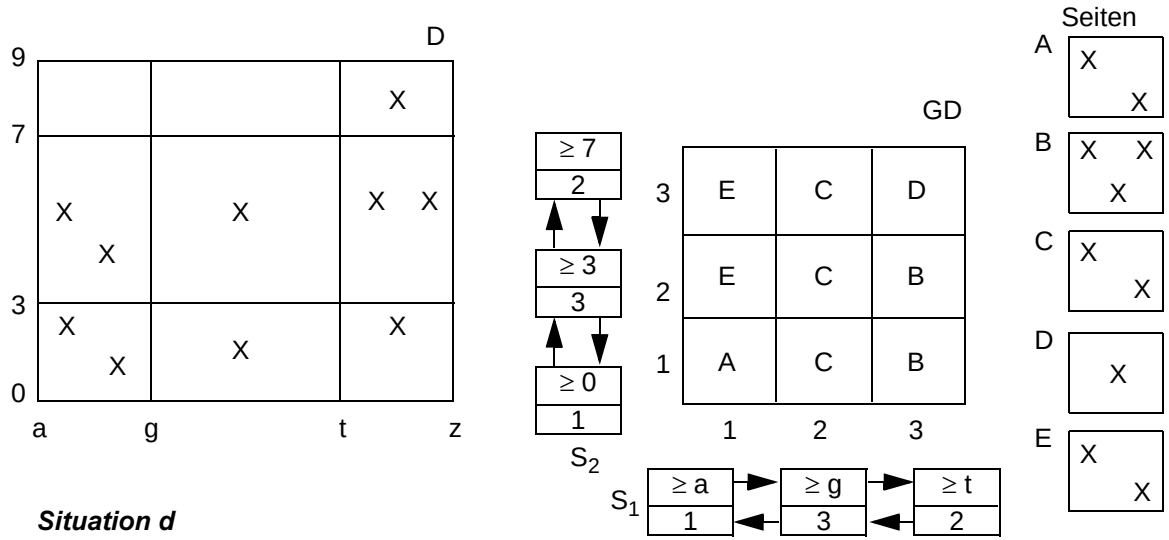
- nachdem 4 Datensätze eingefügt sind:



- nachdem 7 Datensätze eingefügt sind:

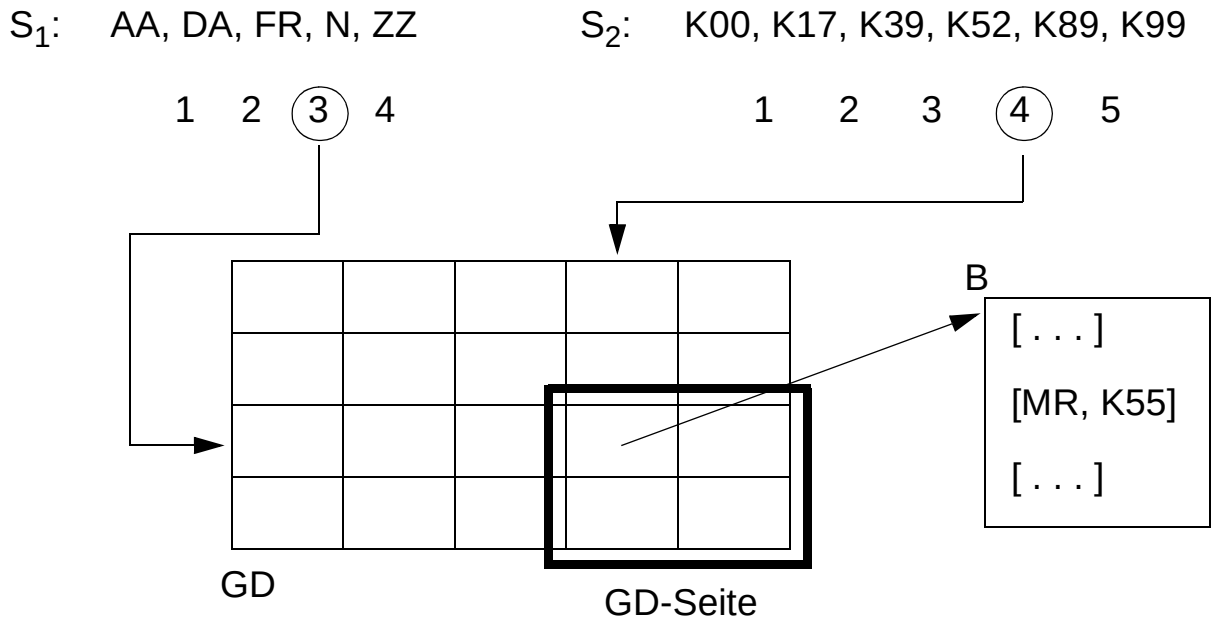



9
7
0
a
g
t
z
X
Situation c
GD
≥ 7
2
≥ 0
1
S2
2
1
A
C
D
A
C
B
1
2
3
S1
≥ a
1
≥ g
3
≥ t
2
Seiten
A
B
C
D


D
9
7
3
0
a
g
t
z
X
Situation d
GD
≥ 7
2
≥ 3
3
≥ 0
1
S2
3
2
1
E
C
D
E
C
B
A
C
B
1
2
3
S1
≥ a
1
≥ g
3
≥ t
2
Seiten
A
B
C
D
E

Exakte Anfrage

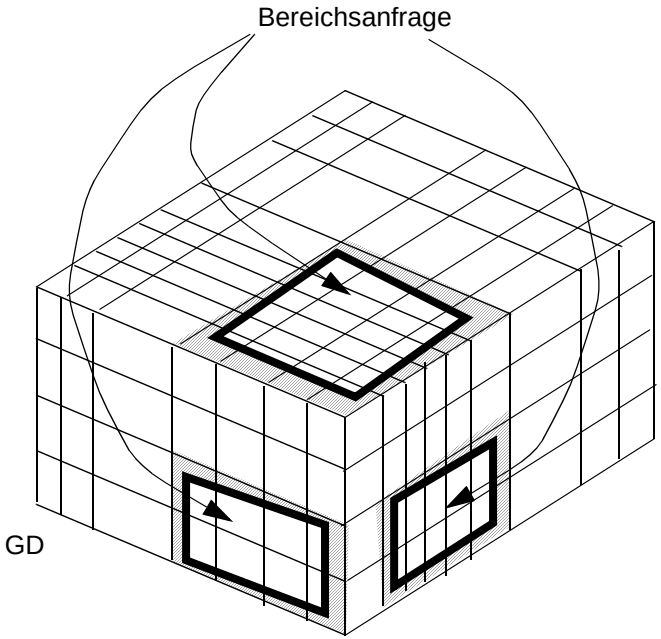
```
SELECT    *
FROM      PERS
WHERE     ORT = 'MR' AND ANR = 'K55'
```



- gegeben zwei Indexeinträge der Skalen
  - Wie berechnet man die Adresse der zugehörigen Zelle?

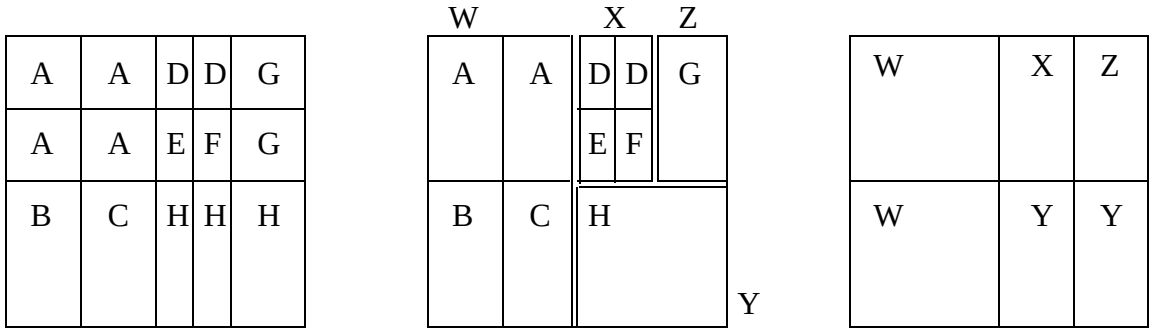
Bereichsanfrage

- Bestimmung der Skalierungswerte in jeder Dimension
- Berechnung der qualifizierten GD-Einträge
- Zugriff auf die GD-Zelle(n) und Holen der referenzierten Seiten



Leistungsbetrachtung

- Wachstum von GD ist superlinear
  - bei gleichförmigen Objektverteilungen: $O(N^{1+(k-1)/k*b})$
  - bei schiefen Objektverteilungen bis zu: $O(N^k)$
- Verfeinerung in Dimension k
  - Anzahl der hinzukommenden GD-Einträge: $n = (m_1 * m_2 * \ldots * m_{k-1})$
- Speicherplatzausnutzung für Seiten
- Organisation von GD als Grid-File
  - Regelfall: zweistufige Realisierung



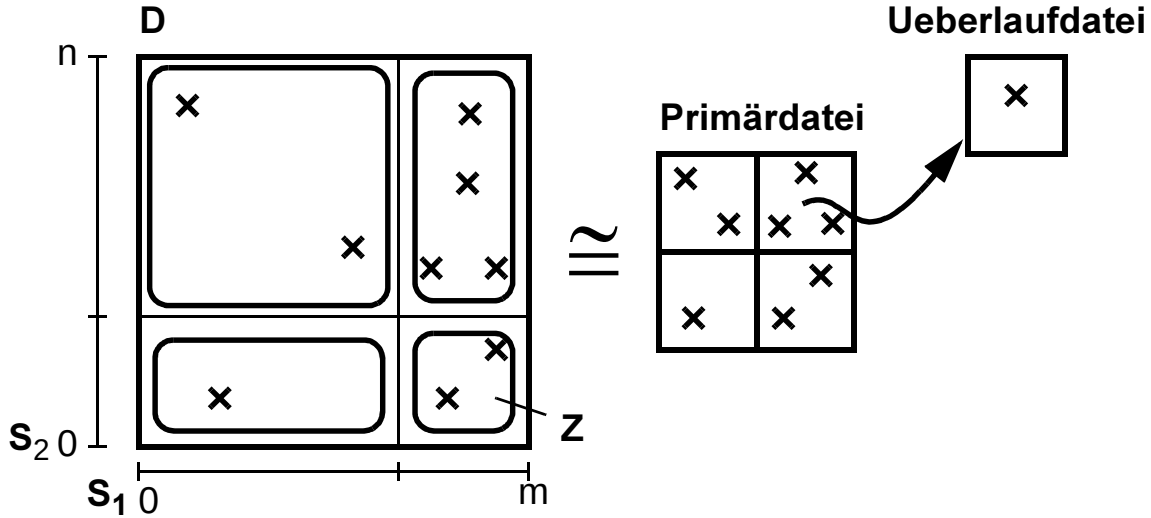
GD, einstufig

GD_{1i}, 4 Directory-Regionen

GD_0, intern

4.2.4 PLOP-Hashing

- PLOP = Piecewise Linear Order Preserving
- PLOP-Hashing verallgemeinert das Prinzip von LH für mehrdimensionale Daten



- Erweitern der Datei
  - Kontrollfunktion: Falls Belegunsfaktor in einer Scheibe größer 100 %
  - Lineares Aufspalten der Seiten in dieser Scheibe
- Verschmelzen von Seiten
  - Kontrollfkt.: Falls Belegung in zwei benachbarten Scheiben kleiner 40%

Leistungsbeurteilung

- im schlechtesten Fall:
  - extrem lange Ketten
  - schlechte SPAN
  - …
- Forderung für effiziente Verarbeitung des PLOP-Hashing (und Grid File):
  - Verteilung der Attribute müssen unabhängig voneinander sein.

  Unter diesen Voraussetzungen ist PLOP-Hashing sehr effizient:
  - hohe SPAN (76%)
  - exakte Anfrage (1.01 Zugriffen im Durchschnitt)
  - effiziente Unterstützung von Bereichsanfragen
  - geringer Aufwand in einem Reorganisationsschritt

4.3 Zugriffsstrukturen 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 aber Fehltreffer möglich
- Probleme
  - neben Position muß auch die Ausdehnung bei der Abbildung der Objekte auf Seiten berücksichtigt werden.
  - Objekte können andere enthalten oder sich gegenseitig überlappen
- Klassifikation der Lösungsansätze
  - sich gegenseitig überlappende Regionen (R-Baum)
  - Clipping (R^+-Baum)
  - Transformationsansatz

R-Baum

- Zugriffsstruktur für die effiziente Verwaltung von **achsenparallelen** Rechtecken
- basiert auf der Idee überlappender Seitenregionen
- verallgemeinert die Idee des B^+-Baums

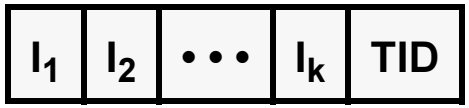
Definition

Ein *R-Baum* mit ganzzahligen Parameter *m* und *M*, $1 \leq m \leq \lceil M/2 \rceil$, organisiert eine Menge von Rechtecken in einem Baum mit folgenden Eigenschaften:

- Der Baum besteht aus Daten- und Directoryseiten. In einer *Datenseite* werden Rechtecke (plus TID) und in einer *Directoryseite* Indexeinträge der Form (*R*, *Ref*) gehalten. Hier bezeichnet *R* ein Rechteck und *Ref* eine Referenz auf einen Teilbaum.
- Jedes Rechteck eines Indexeintrags überdeckt die Datenrechtecke des zugehörigen Teilbaums minimal.
- Alle Datenseiten sind Blätter des Baums. Der Baum ist vollständig balanciert, d.h. alle Pfadlängen von der Wurzel zu einem Blatt sind gleich.
- Jede Seite besitzt maximal *M* Einträge und, mit Ausnahme der Wurzel, mindestens *m* Einträge.

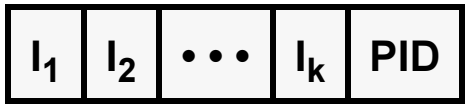
- R-Baum ist höhenbalancierter Mehrwegbaum
  - jeder Knoten entspricht einer Seite
  - pro Knoten maximal M, minimal m ($\leq$ M/2) Einträge

Blattknoteneintrag:



kleinstes umschreibendes Rechteck (Datenrechteck) für TID

Zwischenknoteneintrag:



Intervalle beschreiben kleinste umschreibende Datenregion für alle in PID enthaltenen Objekte

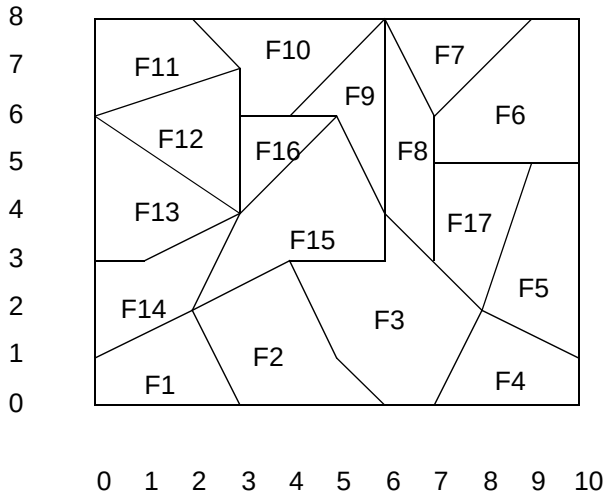
I_j = geschlossenes Intervall bzgl. Dimension j

TID: Verweis auf Objekt PID: Verweis auf Sohn

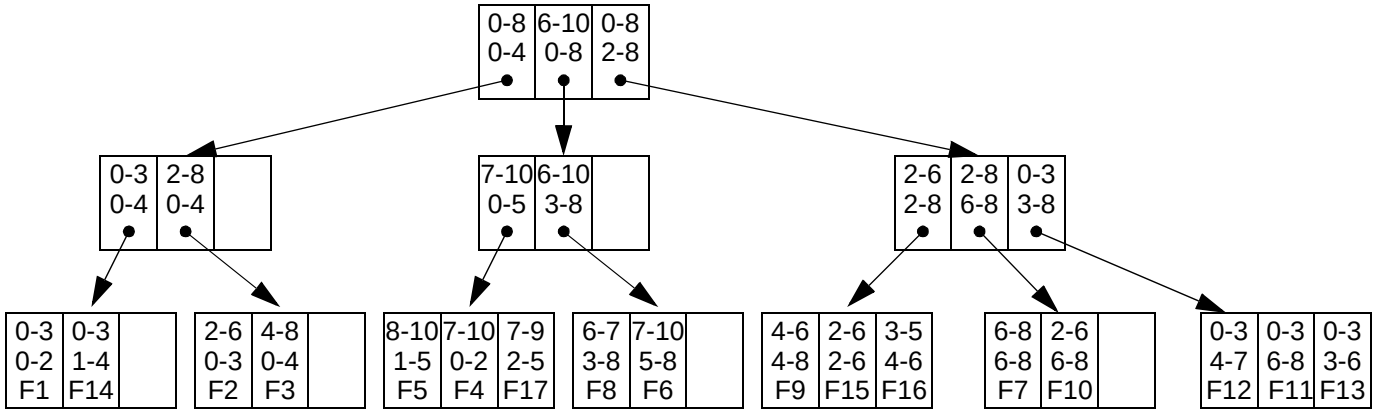
- Eigenschaften
  - starke Überlappung der umschreibenden Rechtecke/Regionen auf allen Baumebenen möglich
  - bei Suche nach Rechtecken/Regionen sind ggf. mehrere Teilbäume zu durchlaufen
  - Änderungsoperationen ähnlich wie bei B-Bäumen

Beispiel (Flächen)

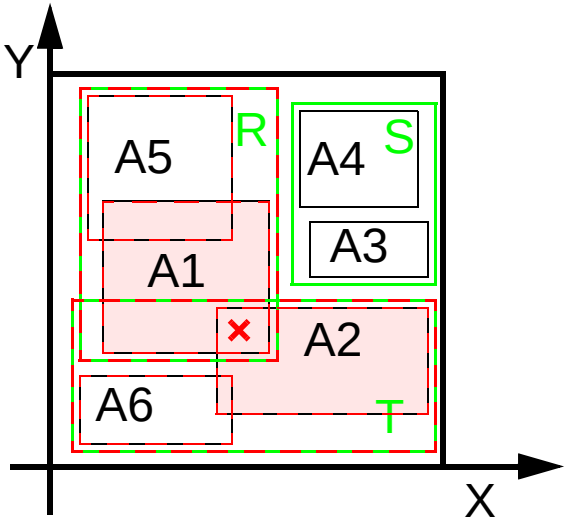
- Abzuspeichernde Flächenobjekte



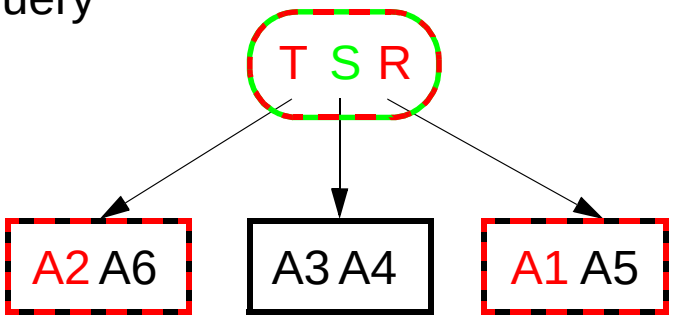
- Zugehöriger R-Baum

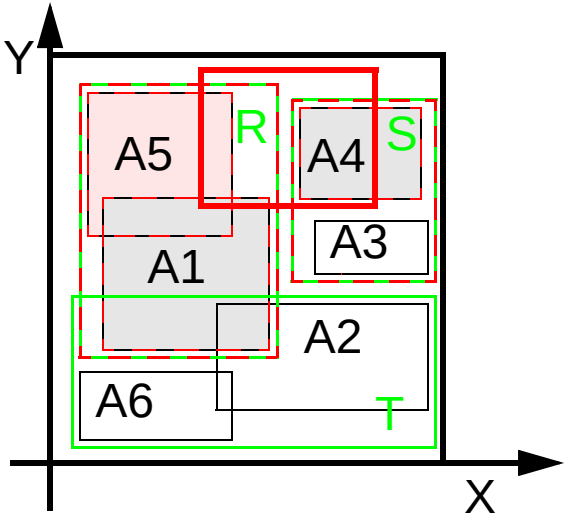


4.3.1 Suchanfragen im R-Baum

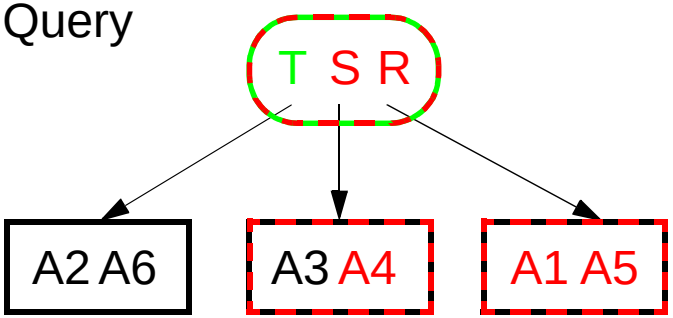


Point Query

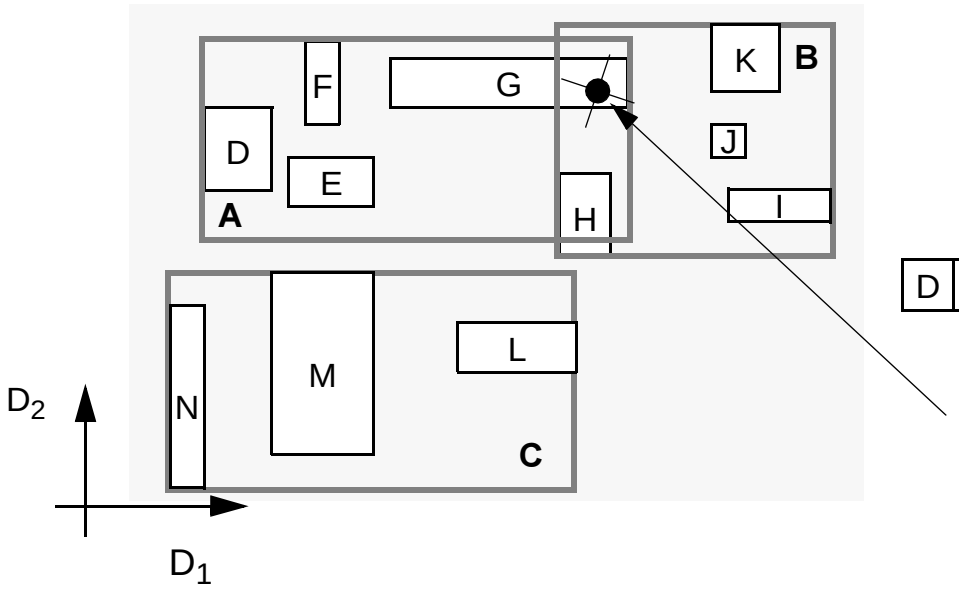


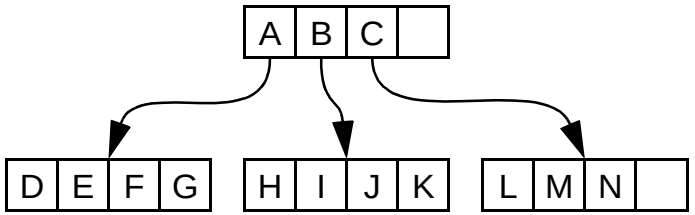


Window Query



Implementierung einer Anfragen



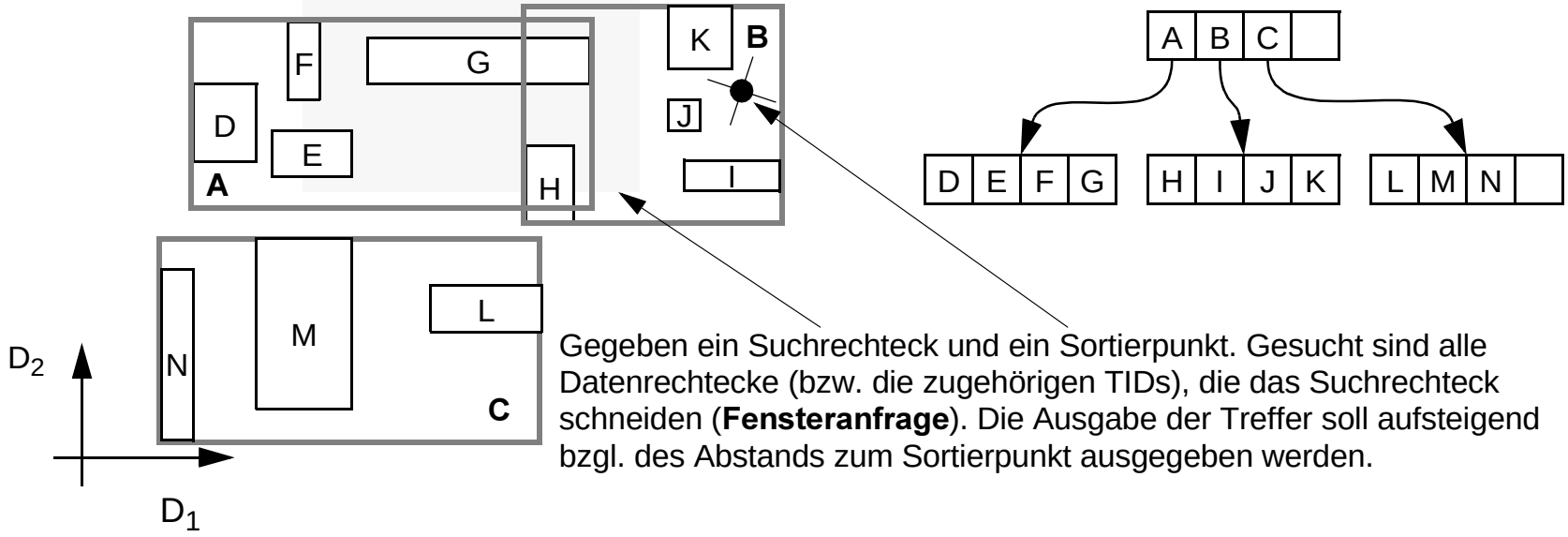


Gegeben ein Suchpunkt. Gesucht sind alle Datenrechtecke (bzw. die zugehörigen TIDs), die diesen Punkt enthalten (**Punktanfrage**)

Aufbau einer Anfrage (Tiefendurchlauf)

- Füge die Referenz des Wurzelknotens in einen Stapel ein.
- Solange der Stapel nicht leer ist:
  a) Nimm das oberste Element aus dem Stapel und lies den zugehörigen Knoten.
  b) Falls der Knoten ein Directoryknoten ist:
  Füge die Referenzen der **qualifizierenden** Einträge in den Stapel ein.
  c) Andernfalls:
  Gib die TIDs der qualifizierenden Datensätze aus.

Prioritätsgesteurte Bereichsanfragen

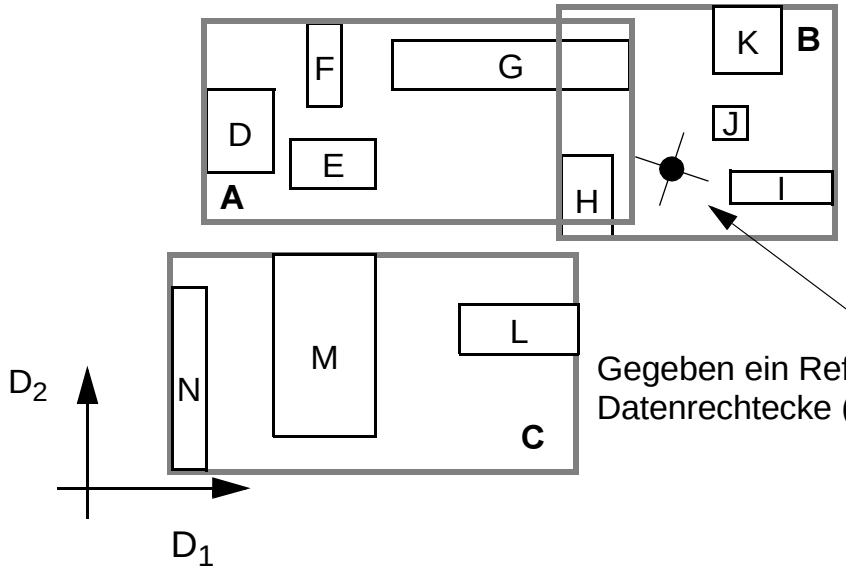


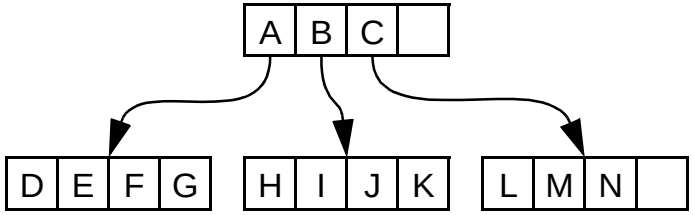
Gegeben ein Suchrechteck und ein Sortierpunkt. Gesucht sind alle Datenrechtecke (bzw. die zugehörigen TIDs), die das Suchrechteck schneiden (**Fensteranfrage**). Die Ausgabe der Treffer soll aufsteigend bzgl. des Abstands zum Sortierpunkt ausgegeben werden.

Aufbau der Anfrage

- Im Unterschied zu der zuvor beschriebenen Anfrage wird nun statt eines Stapels eine Prioritätswarteschlange benutzt.
- Priorität eines Rechtecks ist der minimale Euklidische Abstand zum Sortierpunkt.
- Datenrechtecke werden nun ebenfalls in die Prioritätswarteschlange eingefügt.

k-nächste-Nachbaranfrage





Gegeben ein Referenzpunkt. Gesucht sind die k nächsten Datenrechtecke (bzw. die zugehörigen TIDs) zum Referenzpunkt.

Aufbau der Anfrage

- Die Anfrage entspricht einer prioritätsgesteuerten Bereichsanfrage, wobei der Suchbereich dem gesamten Datenraum entspricht.
- Abbruch des Algorithmus nachdem k Ausgaben produziert wurden.
- Beispiel (k = 3): H, J, I

Kriterien für effiziente Anfragebearbeitung

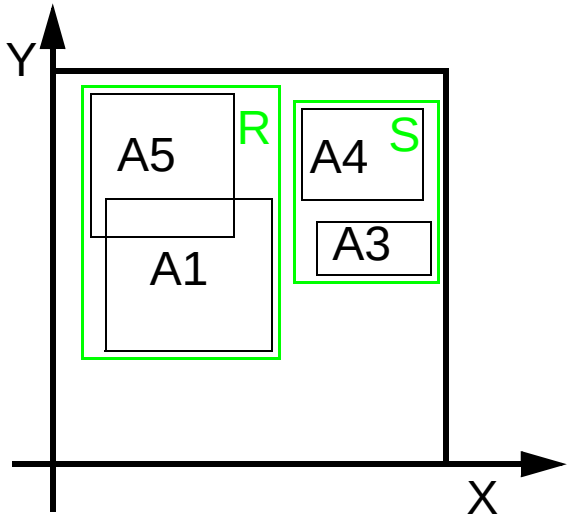
- **Überdeckung** (coverage) einer Ebene eines R-Baums
  - gesamter Bereich, um alle zugehörigen Rechtecke zu überdecken
- **Überlappung** (overlap) einer Ebene eines R-Baums
  - gesamter Bereich, der in zwei oder mehr Knoten enthalten ist
- **Umfang** einer Ebene eines R-Baums
  - Summe des Umfangs aller Rechtecke in einem Knoten

⇒ effiziente Suche erfordert minimale Überdeckung, Überlappung und Umfang

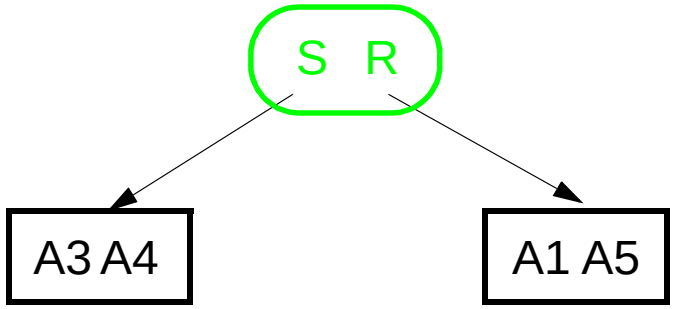
- minimale Überdeckung reduziert die Menge des “toten Raumes” (leere Bereiche), der von den Knoten des R-Baumes überdeckt wird.
- minimale Überlappung reduziert die Menge der Suchpfade zu den Blättern (noch kritischer für die Zugriffszeit als minimale Überdeckung)
- minimaler Umfang reduziert die Trefferquote bei einer Fensteranfrage

⇒ wird beim Splitting der Seiten versucht zu erreichen.

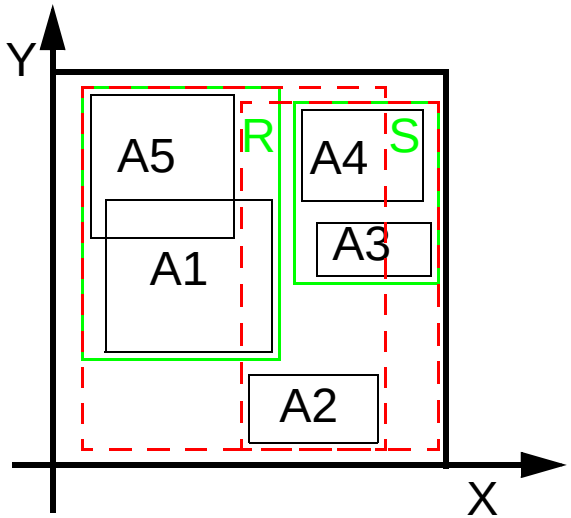
4.3.2 Dynamischer Aufbau

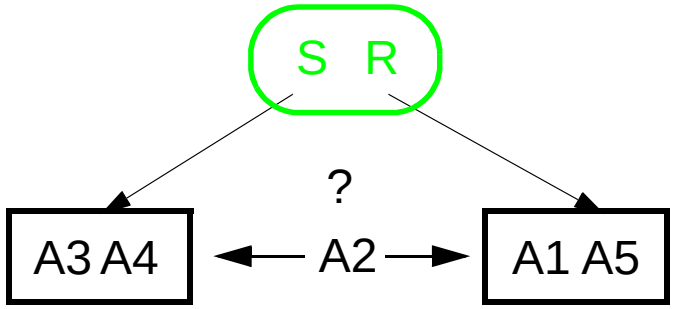


⇒ Split in 2 Seiten



Frage: Wie wird aufgeteilt? *(Splitstrategie)*





Frage: Wo wird eingefügt? *(Einfügestrategie)*

Einfügestrategie des R-Baums

- Wie beim B+-Baum soll das Einfügen auf genau einen Pfad beschränkt sein.
- Gegeben ein innerer Knoten. An welchen Zweig des Knotens soll die Einfügeoperation weitergeleitet werden?

Fälle

Fall 1: *R* fällt vollständig in genau ein Directory-Rechteck *D*

– Einfügen in Teilbaum von *D*

Fall 2: *R* fällt vollständig in mehrere Directory-Rechtecke D_1, ... , D_n

– Einfügen in Teilbaum von D_i, das die geringste Fläche aufweist

Fall 3: *R* fällt vollständig in kein Directory-Rechteck

– Einfügen in Teilbaum von *D*, das den geringsten Flächenzuwachs erfährt (in Zweifelsfällen: ... , das die geringste Fläche hat)

– *D* muß entsprechend vergrößert werden

Variationsmöglichkeiten

- Einbeziehen der entstehenden Überlappung, des Umfangs, des toten Raums

Splitstrategien für R-Bäume

- Der Knoten K läuft mit $|K| = M+1$ über:
  - Aufteilung auf zwei Knoten K_1 und K_2, s.d. $|K_1| \geq m$ und $|K_2| \geq m$

Erschöpfender Algorithmus

- Suche unter den $O(2^M)$ Möglichkeiten die "beste" aus $\Rightarrow$ sehr aufwendig ($M \approx 200$)
- Gibt es eine niedrigere obere Schranke?

Linearer Algorithmus (Greedy-Verfahren)

- Suche für jede Dimension die "Extrem"-Rechtecke
- Wähle das Paar von Rechtecken mit den größten (normalisierten) Abstand bezüglich einer Dimension; diese beiden Rechtecke bilden K_1 und K_2
- Durchlaufe alle verbliebenen Rechtecke R_i und weise sie dem Knoten K_j zu, der dadurch den geringsten Flächenzuwachs erfährt.
- Falls die Anzahl der verbliebenen Rechtecke = $m - |K_j|$, dann weise sie K_j zu.

Quadratischer Algorithmus (Verfeinertes Greedy-Verfahren)

- Wähle das Paar von Rechtecken R_1 und R_2 mit dem größten Wert von d als Ausgangsmenge für K_1 bzw. K_2; d := Fläche($R_1 \cup R_2$) - Fläche(R_1) - Fläche(R_2)
- Durchlaufe alle verbliebenen Rechtecke R_i in einer Reihenfolge, so daß immer das Rechteck R_i als nächstes zugeordnet wird, wo die Differenz zwischen Fläche($K_1 \cup R_i$)-Fläche (K_1) und Fläche($K_2 \cup R_i$)-Fläche (K_2) am größten ist

Split des R*-Baums (Plane-Sweep-Verfahren)

- Bestimmung der Splitdimension
  - Sortiere für jede Dimension die Rechtecke gemäß ihrer Extremwerte

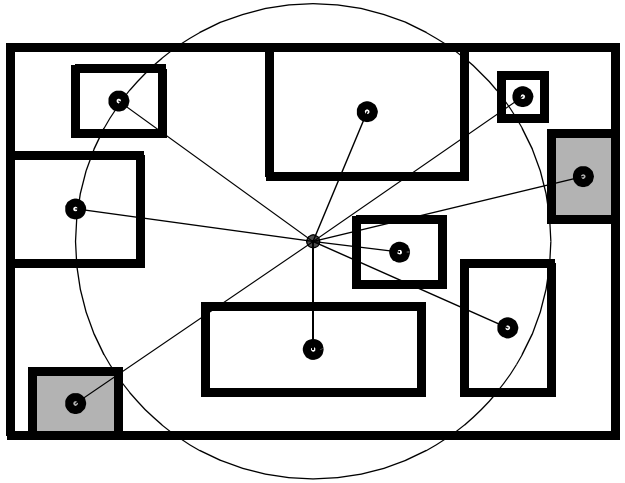
  Für jede Dimension:
  - Für jede der beiden Sortierungen werden M-2m+2 Aufteilungen der M+1 Rechtecke bestimmt, so daß die 1. Gruppe der j-ten Aufteilung m-1+j Rechtecke und die 2. Gruppe die übrigen Rechtecke enthält
  - *UG* sei die Summe aus dem Umfang der beiden MURs R_1 und R_2 um die Rechtecke der beiden Gruppen
  - *US* sei die Summe der *UG* aller berechneten Aufteilungen

  Es wird die Dimesion mit dem geringsten *US* als Splitdimension gewählt
- Bestimmung der Aufteilung
  - Es wird die Aufteilung der gewählten Splitdimension genommen, bei der R_1 und R_2 die geringste Überlappung haben
  - In Zweifelsfällen wird die Aufteilung genommen, bei der R_1 und R_2 die geringste Überdeckung von totem Raum besitzen

(Die besten Resultate haben sich bei Experimenten für m = 0,4*M ergeben)

Vermeidung von Splits (R*-Baum)

- Bevor eine Seite einem Split unterzogen wird, werden die am weitesten vom Zentrum des gemeinsamen minimalen Rechtecks entfernt liegenden Einträge (Einträge von Datensätzen oder von Teilbäumen) gelöscht und noch einmal in den R*-Baum eingefügt *(Forced Reinsert)*



- Ziele:
  - Vermeiden von Splits $\Rightarrow$ bessere Speicherplatzausnutzung
  - Anpassung des R*-Baums an die aktuelle Datenverteilung (bessere Unabhängigkeit von sortierten Einfügungen, Anteil der gelöschten und wieder eingefügten Rechtecke beim R*-Baum: 30%)

Vergleich der verschiedenen Strategien

- Resultate aus Experimenten mit verschiedenen Datenverteilungen (100000 Datensätze) und verschiedenen Anfragetypen.
- Keine Pufferung (mit Ausnahme eines Pfads)
- Experimente mit zweidimensionalen Punkten

	linearer Split	quadratischer Split	R*-Baum Split
rel. Anfrageleistung	233.1	175.9	100.0
Speicherplatzausnutzung	64 %	68%	71%
Einfügekosten	7.3	4.5	3.4

- Experimente mit zweidimensionalen Rechteckmengen

	linearer Split	quadratischer Split	R*-Baum Split
rel. Kosten einer Fensteranfrage (0.1 %)	189.8	126.4	100.0
Speicherplatzausnutzung	63 %	68%	73%
Einfügekosten	12.6	7.7	6.1

4.3.3 Massenaufbau eines R-Baums

Problem:

- Gegeben eine Datenmenge S. Erstelle mit möglichst *niedrigen Aufwand* einen guten R-Baum.

Modellannahmen

- Datenmenge S
  - N = #Datenobjekte in S
- Hauptspeicher
  - M = #Datenobjekte, die in den Haupstpeicher passen
- Ein Block ist die Transfereinheit zwischen Haupt- und Sekundärspeicher
  - B = page capacity
  - m = M/B (Ann.: M ist Vielfaches von B)
  - n = N/B (Ann.: N ist Vielfaches von B)

Überlegungen zu den Kosten

Objektweises Einfügen von S

- $O(N \log_B N)$ I/O

Massenaufbau eines (B+-Baums)

- Algorithmus
  1. Externes Sortieren
  2. Bottom-up Aufbau des B+-Baums
- Gesamtkosten
  - $\Theta(n \log_m n)$ I/O

Untere Schranke für den Massenaufbau eines R-Baums

- $\Omega(n)$ I/O (durch linearen Scan über die Datenmenge)
- i. A. führt dies aber zu R-Bäumen mit schlechter Suchqualität

Ziel

- Erzeugen von “guten” R-Bäumen mit dem gleichen Aufwand wie bei B+-Bäumen

4.3.3.1 Sortierbasierter Massenaufbau

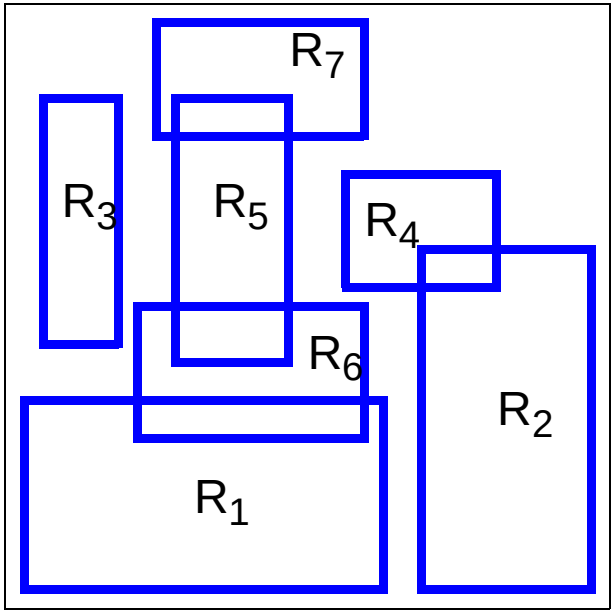
1. Schritt

- Sortieren der Daten bzgl. einer vorgegebenen Ordnung.
- Beispiele:
  - Minimum der Objekte in einer fest gewählten Dimension
  - Z-Wert der Rechtecksschwerpunkte
  - Hilbert-Wert der Rechtsschwerpunkte

2. Schritt

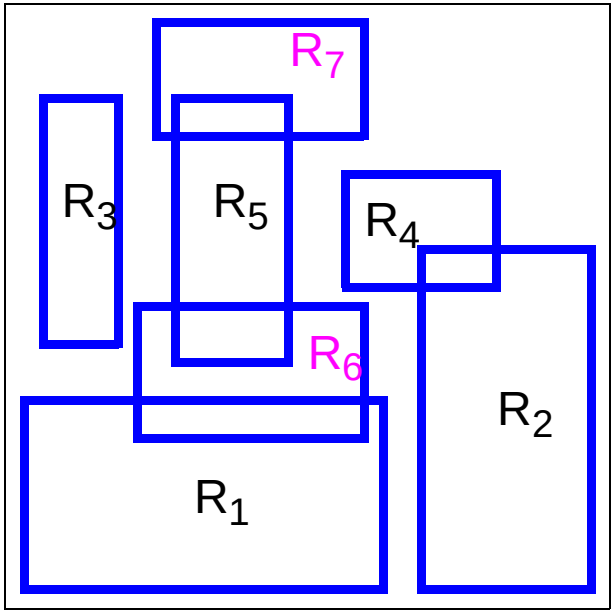
- Strategie A
  - Benutze einen LRU-Buffer der Größe m und füge objektweise in den R-Baum ein.
- Strategie B
  - Analog zum B+-Baum wird der R-Baum bottom-up über die sortierte Eingabemenge aufgebaut

Beispiel



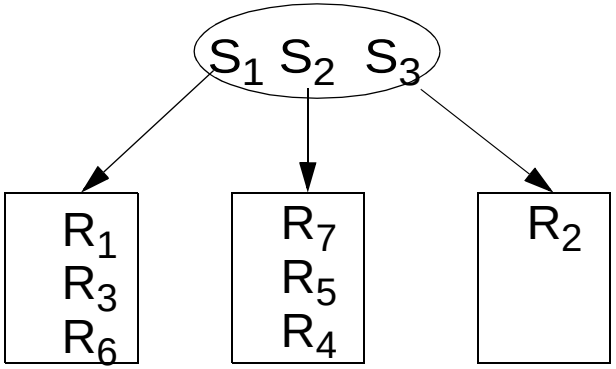
Sortieren nach 1. Dimension:
R_1, R_3, R_6. R_7, R_5, R_4, R_2

Beispiel

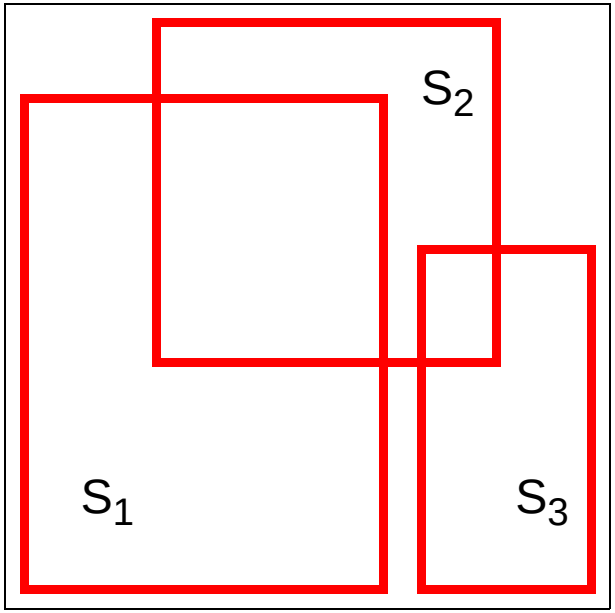


Sortieren nach 1. Dimension:
R_1, R_3, R_6. R_7, R_5, R_4, R_2

R-Baum



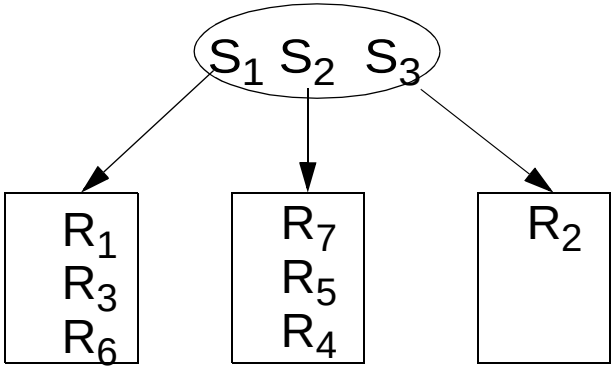
Beispiel



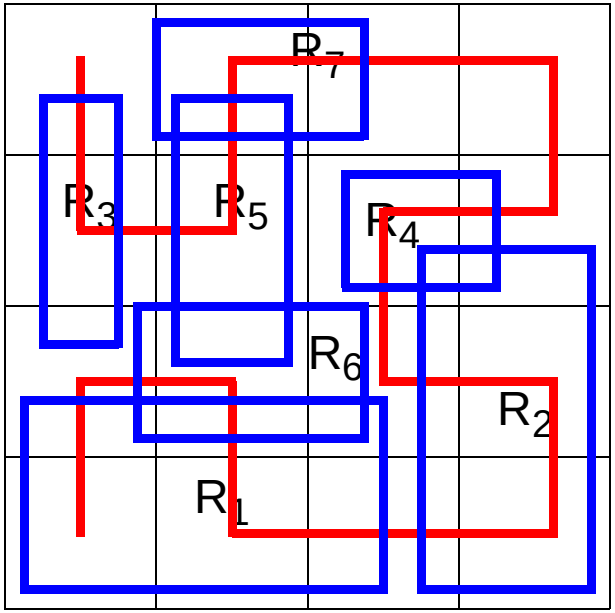
Partitionierung des Wurzelknotens

Sortieren nach 1. Dimension:
R_1, R_3, R_6. R_7, R_5, R_4, R_2

R-Baum

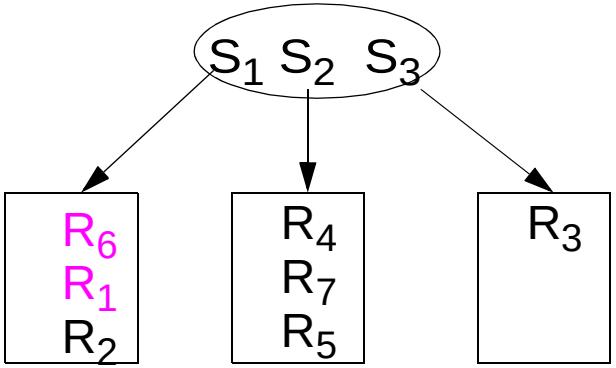


Beispiel

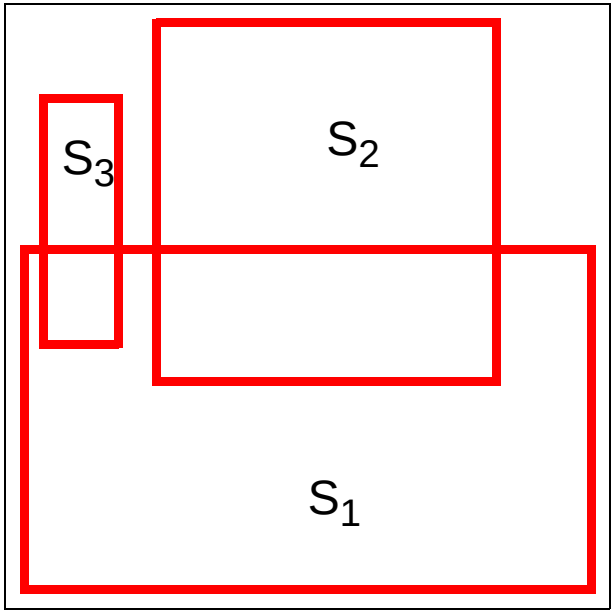


Sortieren nach Hilbertwert:
R_1, R_6, R_2. R_4, R_7, R_5, R_3

R-Baum



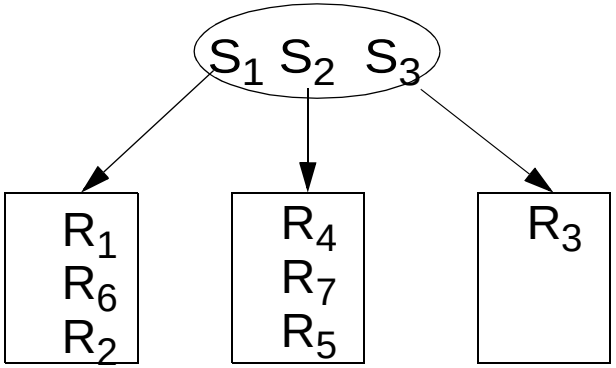
Beispiel



Partitionierung des Wurzelknotens

Sortieren nach Hilbertwert:
R_1, R_6, R_2. R_4, R_7, R_5, R_3

R-Baum



Sort-Tile-Recursive

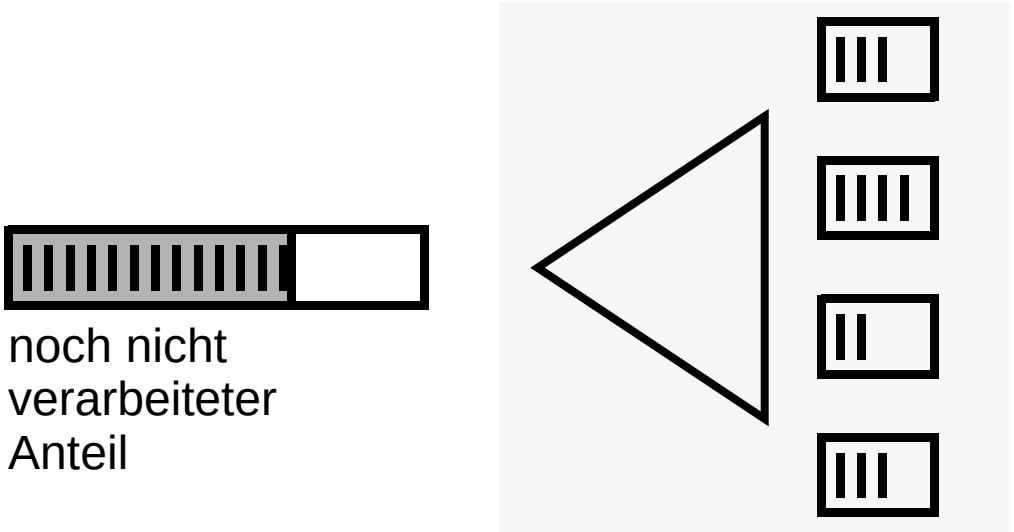
- Leutenegger, Lopez, Edgington (1997). Wird in Oracle für den Massenaufbau von R-Bäumen benutzt.
- Algorithmus (2-dimensionale Daten)
  - Sortiere die Rechtecken bzgl. der ersten Dimension Ihres Schwerpunktes und erzeuge Scheiben mit jeweils $\lceil n^{1/2} \rceil$ Daten (eine Scheibe besteht aus einer zusammenhängenden Teilfolge von Rechtecken).
  - Sortiere jeder dieser Scheiben bzgl. der zweiten Dimension des Schwerpunkts der Rechtecke und erzeuge Seiten mit jeweils B Rechtecken.
- Der Füllgrad einer Scheibe/Seite entspricht mit Ausnahme der Letzten dem gesetzten Wert.
- Verallgemeinerung für mehrdimensionale Daten
  - Iteratives Durchlaufen der einzelnen Dimensionen
  - Aufwand: d-mal Sortieren der gesamten Datenmenge
- Experimente (und Vergleich zum sortierbasierten Aufbau mit Hilbert-Werten) haben gezeigt, dass dieses Verfahren i. A. qualitativ bessere R-Bäume erzeugt

  ABER: es gibt keine Garantie für qualitativ bessere Bäume

4.3.3.2 Quickload

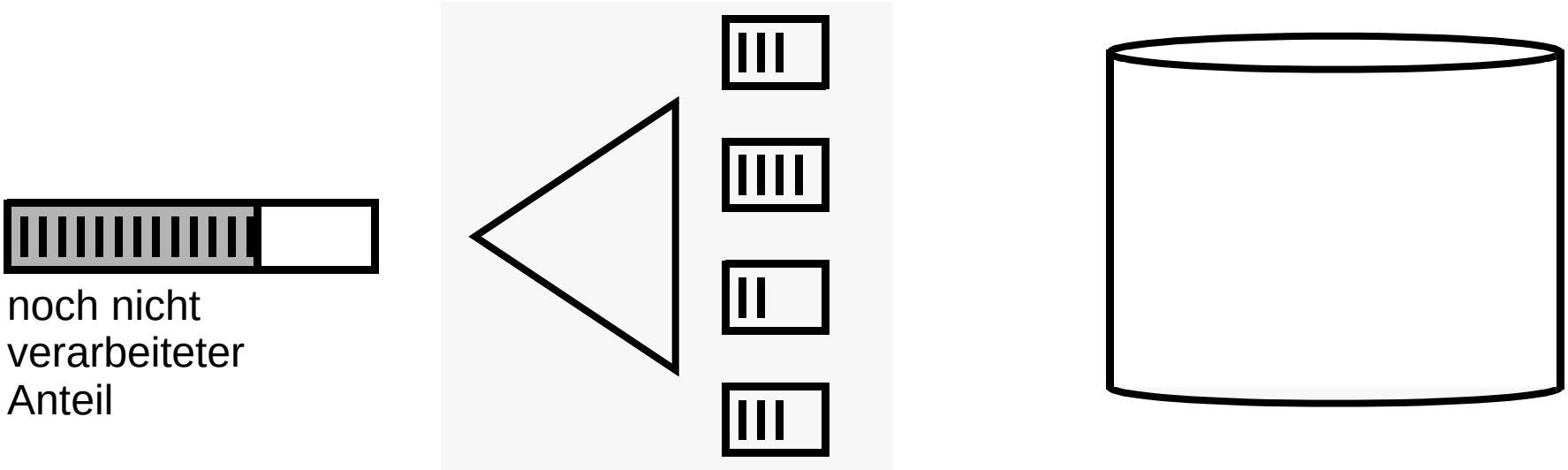
- Einfaches Verfahren zum schnellen Aufbau eines Index (Bercken & Seeger, 2001).
  - ähnliche Idee kann auch für Joins verwendet werden.
- Anwendbar für den R-Baum und viele andere Zugriffsstrukturen (Grow&Post-Tree). Folgende Routinen müssen implementiert sein:
  - **ChooseSubtree**:
    Zu einem Datenobjekt und einem internen Indexknoten wird der Kindknoten berechnet, an den der Datenobjekt weitergereicht wird.
  - **Grow**
    Zu einem Datenobjekt (Indexeintrag) und einem Blattknoten (Indexknoten) wird der Datenobjekt in den Knoten eingetragen.
  - **SplitNode**
    Ein Knoten wird in zwei aufgespalten.
  - **Post**
    Die durch ein Split erzeugten bzw. veränderten Indexeinträge werden an den Elternknoten weitergereicht.
- Der Einfachheit halber zeigen wir im Folgenden wie die Blattebene eines Index aufgebaut wird.

Skizze des Verfahrens:



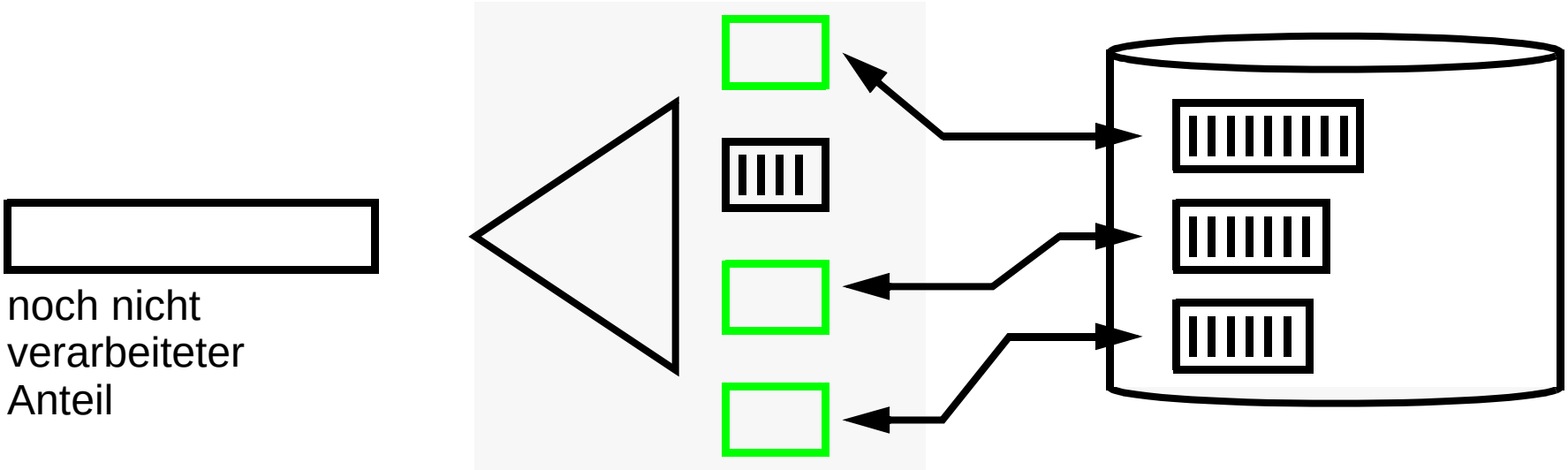
- Aufbau eines Index im Hauptspeicher durch eine Stichprobe der Relation

Skizze des Verfahrens:



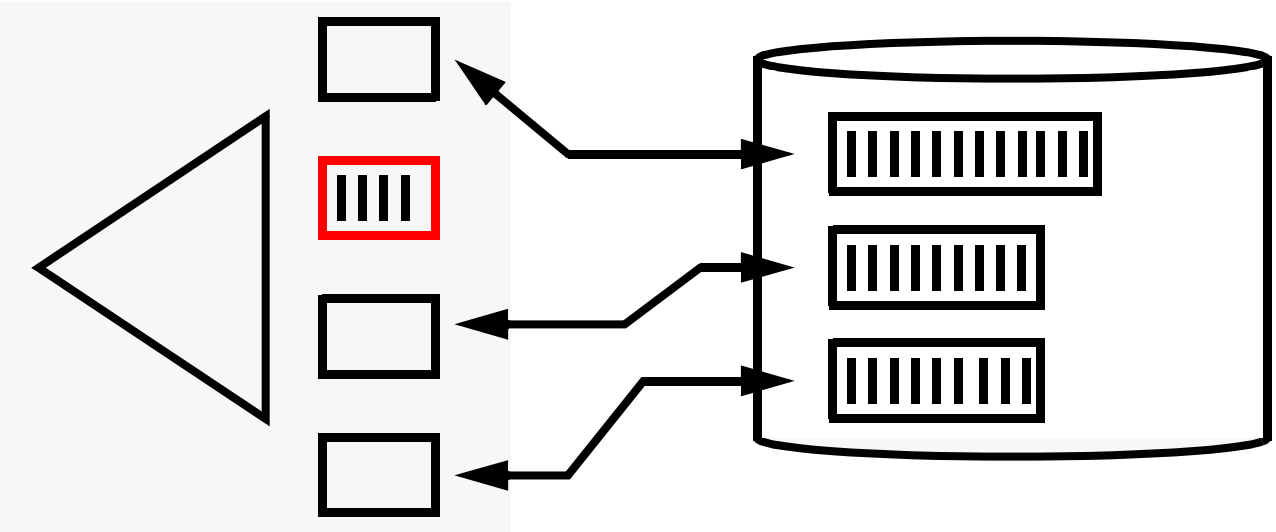
- Aufbau eines Index im Hauptspeicher durch eine Stichprobe der Relation
- Für jedes Blatt: Zuordnung eines externen Puffers.

Skizze des Verfahrens:



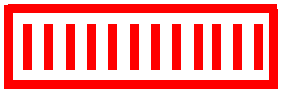
- Aufbau eines Index im Hauptspeicher durch eine Stichprobe der Relation
- Für jedes Blatt: Zuordnung eines externen Puffers
- Verteilen der noch nicht verarbeiteten Datenobjekte:
  - Falls ein Datenobjekt auf ein volles Blatt trifft, wird der Inhalt des Blatts in den assoziierten Puffer geschrieben und das Blatt gelöscht.
  - Falls das Blatt bereits gelöscht ist: Einfügen des Objekts in den Puffer.
  - Ansonsten: Einfügen des Objekts in das Blatt.

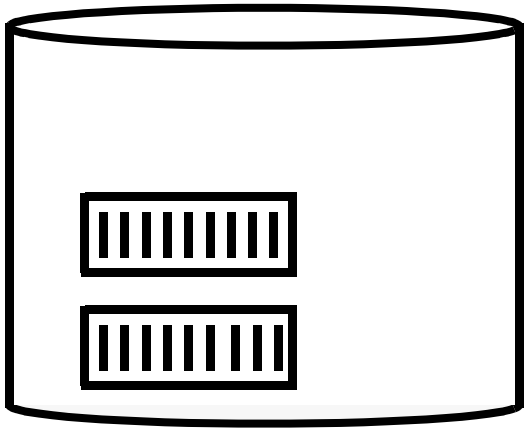
Skizze des Verfahrens:



- Aufbau eines Index im Hauptspeicher durch eine Stichprobe der Relation
- Verteilen der noch nicht verarbeiteten Datenobjekte:
  - Falls ein Datenobjekt auf ein volles Blatt trifft, wird der Inhalt des Blatts in den assoziierten Puffer geschrieben und das Blatt gelöscht.
  - Falls das Blatt bereits gelöscht ist: Einfügen des Objekt in den Puffer.
  - Ansonsten: Einfügen des Objekte in das Blatt.
- Falls der externe Puffer eines Blattknotens leer ist, wird der Knoten als Datenseite des Zielindex akzeptiert.

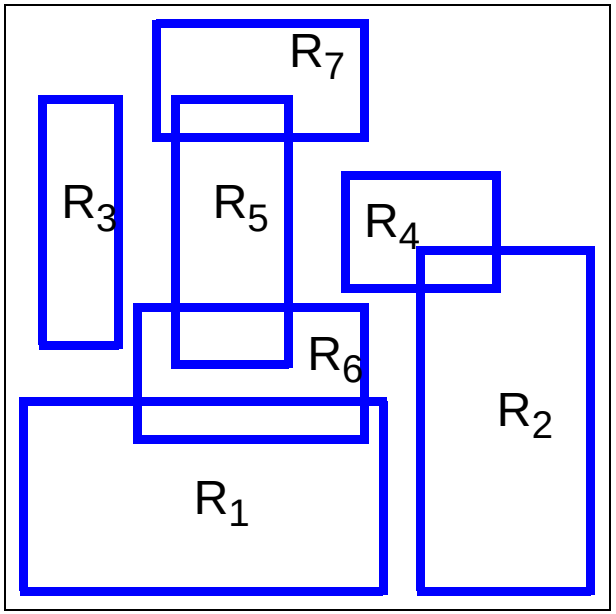
Skizze des Verfahrens:





- Aufbau eines Index im Hauptspeicher durch eine Stichprobe der Relation
  - Verteilen der noch nicht verarbeiteten Datenobjekte:
    Falls ein Datenobjekt auf ein volles Blatt trifft, wird der Inhalt des Blatts in den assoziierten Puffer geschrieben und das Blatt gelöscht.
  - Falls das Blatt bereits gelöscht ist: Einfügen des Objekt in den Puffer.
  - Ansonsten: Einfügen des Objekte in das Blatt.
- Falls der externe Puffer eines Blattknotens leer ist, wird der Knoten als Datenseite des Zielindex akzeptiert.
- Lösche den Index und wende das Verfahren rekursiv auf jeden nicht-leeren Puffer an.

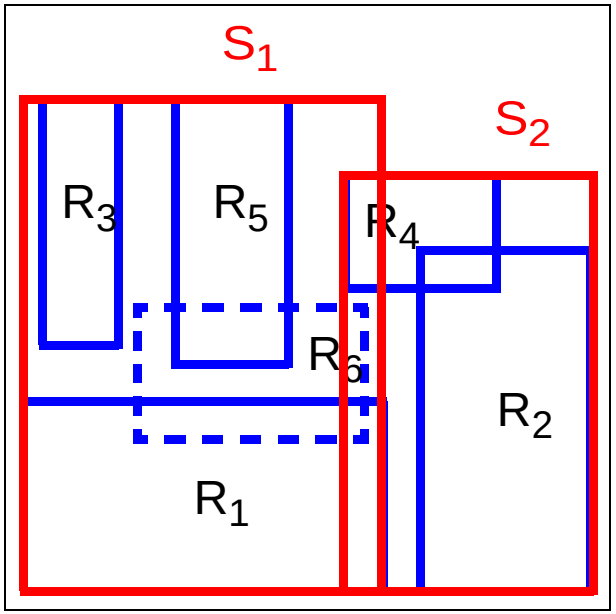
Beispiel (R-Baum):



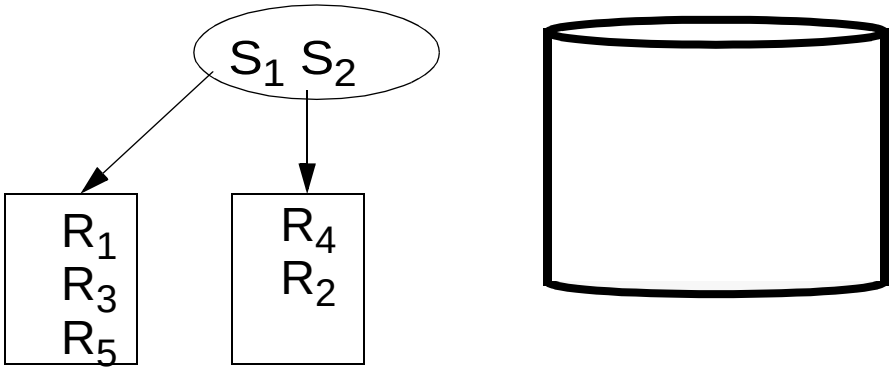
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



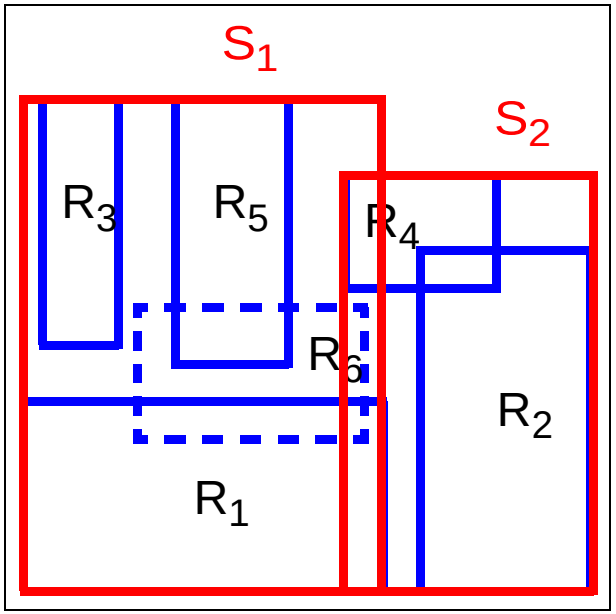
R-Baum



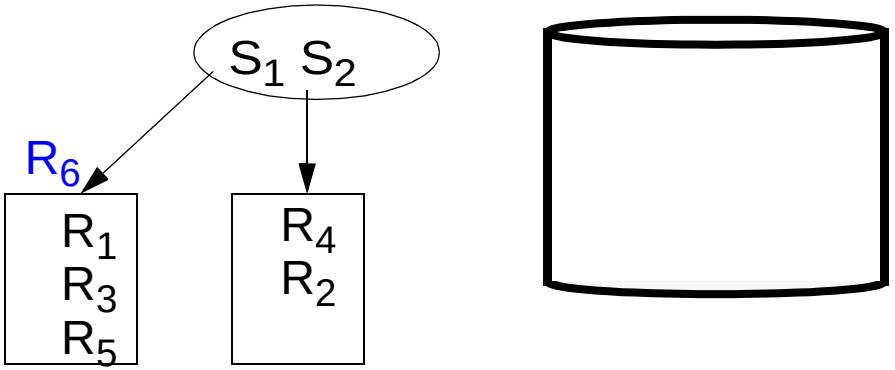
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



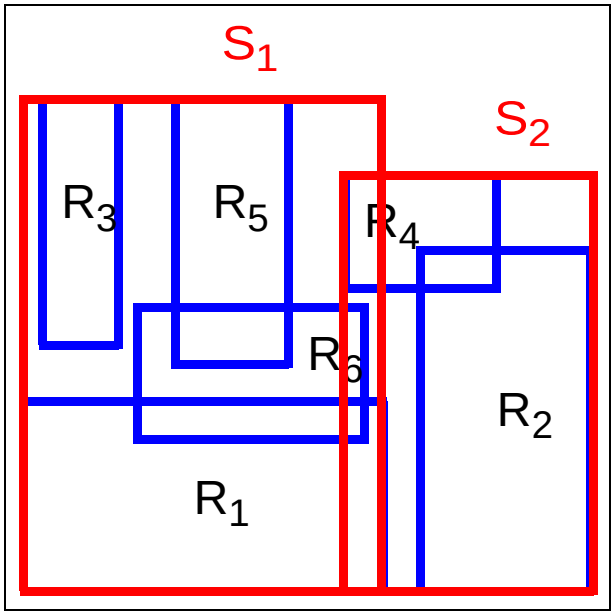
R-Baum



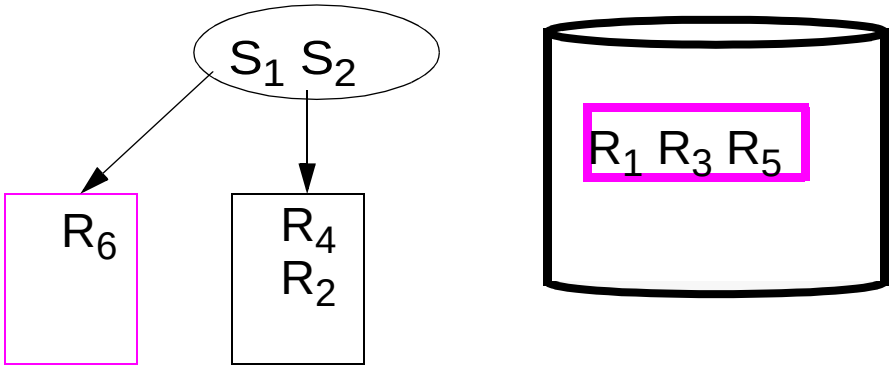
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



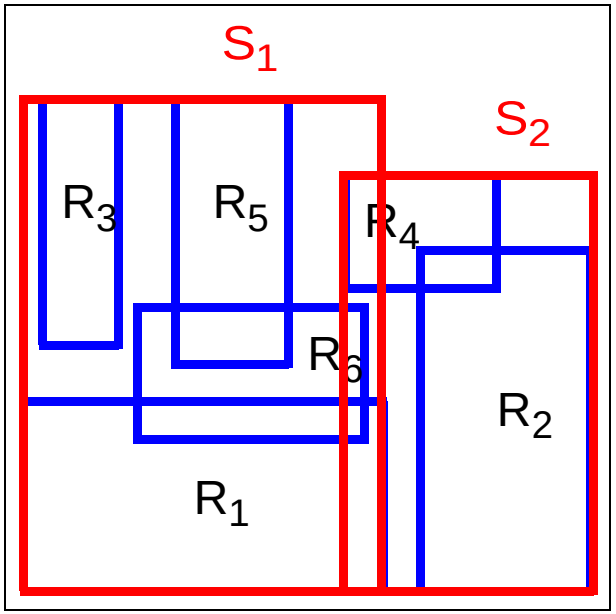
R-Baum



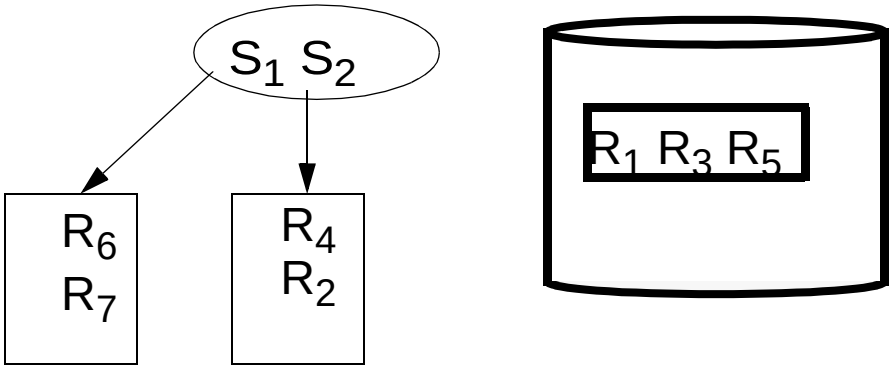
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



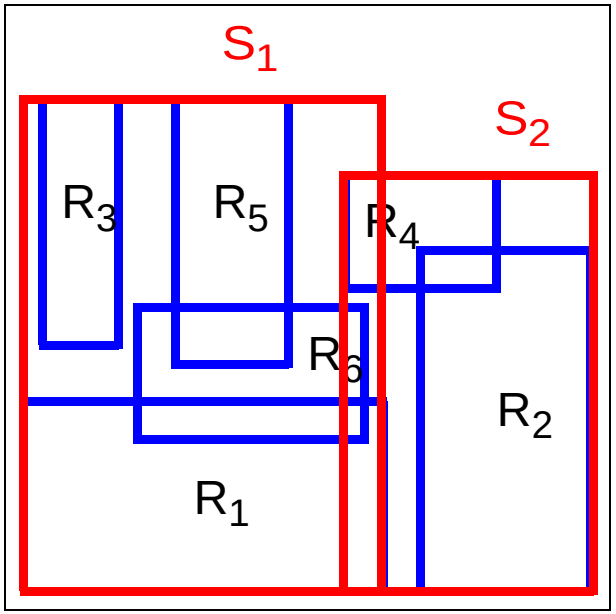
R-Baum



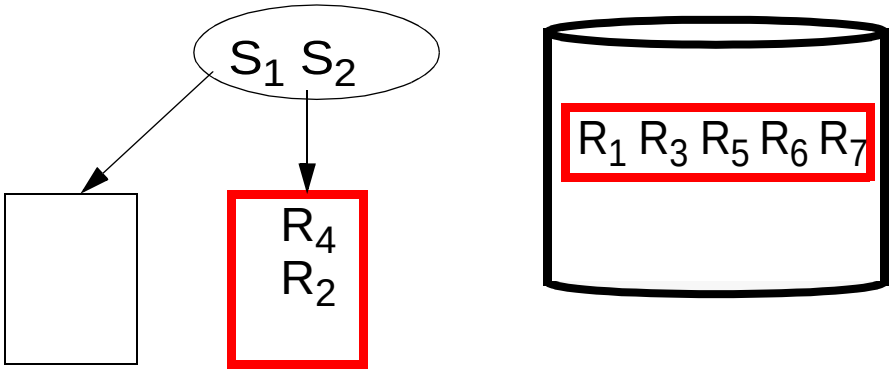
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



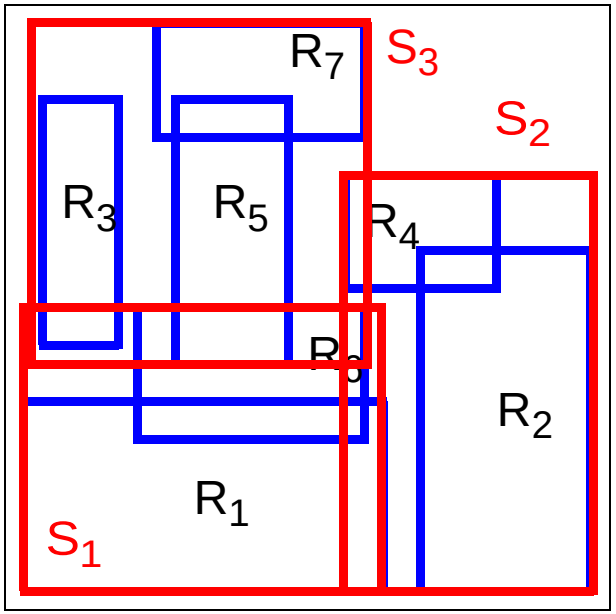
R-Baum



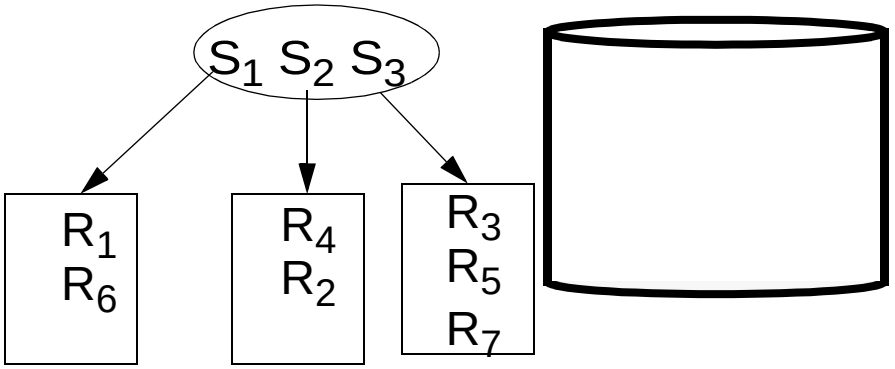
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



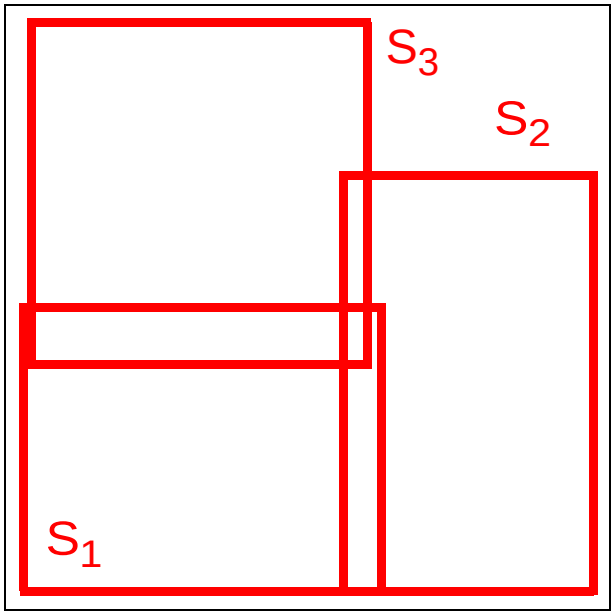
R-Baum



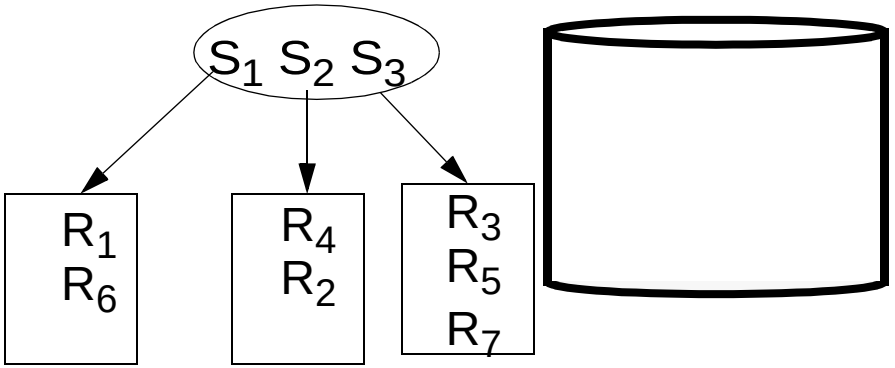
Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Beispiel (R-Baum):



R-Baum



Annahmen:

- Kapazität in einem Blatt: 3
- Verzweigungsgrad: 3
- Hauptspeicher bietet Platz für maximal zwei Blattknoten

Leistung

- Im schlimmsten Fall kann das Verfahren entarten, wobei nach dem Aufbau des Index im Hauptspeicher die restlichen Daten in einen Blattknoten des R-Baums eingefügt werden.

  Laufzeit des Verfahrens: $O(N^2/(M*B))$ I/O Zugriffe

- Bei einer gleichmäßigen Verteilung der Daten auf die Blattknoten beträgt die Anzahl der Zugriffe

  $$O(N/B \log_{M/B} N/B)$$

  Dieser Aufwand entspricht dann also dem von externen Sortieren.

- Speicherplatzausnutzung entspricht der eines R-Baums und ist damit wesentlich niedriger als beim Massenaufbau eines linearen R-Baums.

4.3.3.3 Pufferbaum

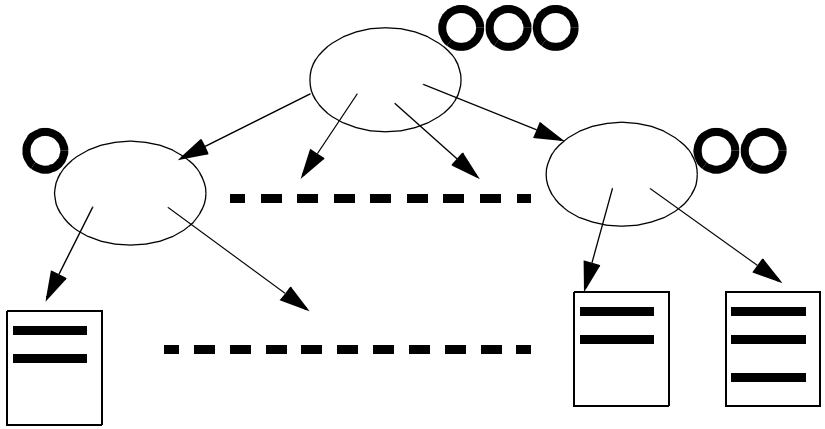
- Wie bereits bei Quickload ist das Verfahren für die Klasse der Grow&Post-Trees anwendbar.
  - Wir beschränken die folgende Diskussion auf den R-Baum.
- Das Verfahren nutzt den von Lars Arge 1995 vorgestellten Pufferbaum, siehe [Arg 95], zum Massenaufbau eines Index.

Wesentliche Idee

- Bündelung von mehreren Operationen mit einem gemeinsamen Einfügepfad
- Gute Ausnutzung des verfügbaren Hauptspeichers

Realisierung

- nebenläufige Verarbeitung von mehreren Einfügeoperationen
  - Blockieren einer Operation an einem Knoten
  - Zwischenspeicherung des Status einer blockierten Einfügeoperation
  - Reaktivierung aller wartender Einfügeoperationen eines Knotens, wenn sich an diesem Knoten genügend Masse angesammelt hat.



Zwei Ansätze

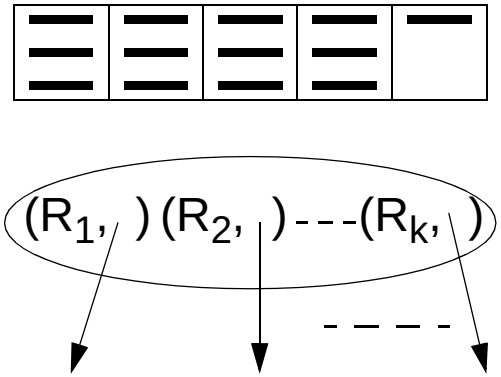
- Bercken, Seeger & Widmayer (1997)
  - Aufbau eines Index in verschienden Phase (pro Phase eine Ebene)
- Arge, Hinrichs, Vahrenhold, Vitter (1999)
  - Aufbau des Index, indem an allen $\lfloor \log_B M \rfloor$ Ebenen ein Puffer der Größe $\Theta(M)$ angehängt wird.
  - Vorteil dieses Ansatzes ist, dass auch Updates, Löschen und Anfragen als Massenoperationen unterstützt werden.
  - Für praktische Implementierungen: Einen Puffer an jedem Knoten
- Pufferbäume können auch zum Aufbau von B+-Bäumen benutzt werden (ohne die zugrundeliegende Datenmenge vorher zu sortieren)
  - Gesamtkosten entsprechen der unteren Schranke von externen Sortieren!

Pufferbaum

Definition:

Ein R-Baum implementiert einen Pufferbaum der Ordnung C, Pufferkapazität p und Knotenkapazität B, falls

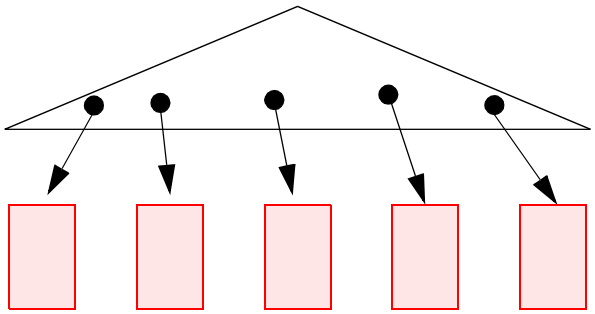
1. ein innerer Knoten höchstens C Kinder hat;
2. ein Block maximal B Datensätze aufnehmen kann;
3. ein innerer Knoten einen Puffer mit höchstens p Blöcken besitzt;
4. die Blöcke eines Puffers mit Ausnahme des letzten Blocks stets voll sind.



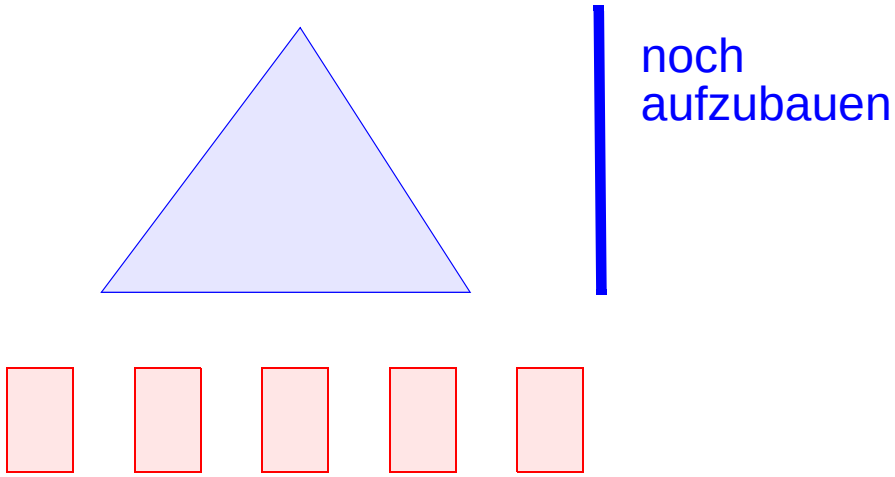
1. Phase: Aufbau der Blattebene

- Einfügen der Datensätze in einen Pufferbaum
- Blattknoten des Pufferbaums entsprechen den Knoten der Zielstruktur

Pufferbaum
(C = Verzweigungsgrad der inneren Knoten)



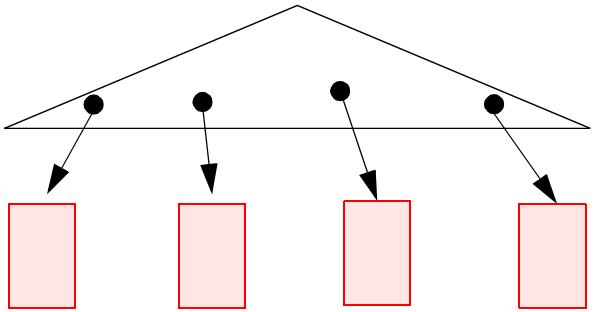
Zielstruktur



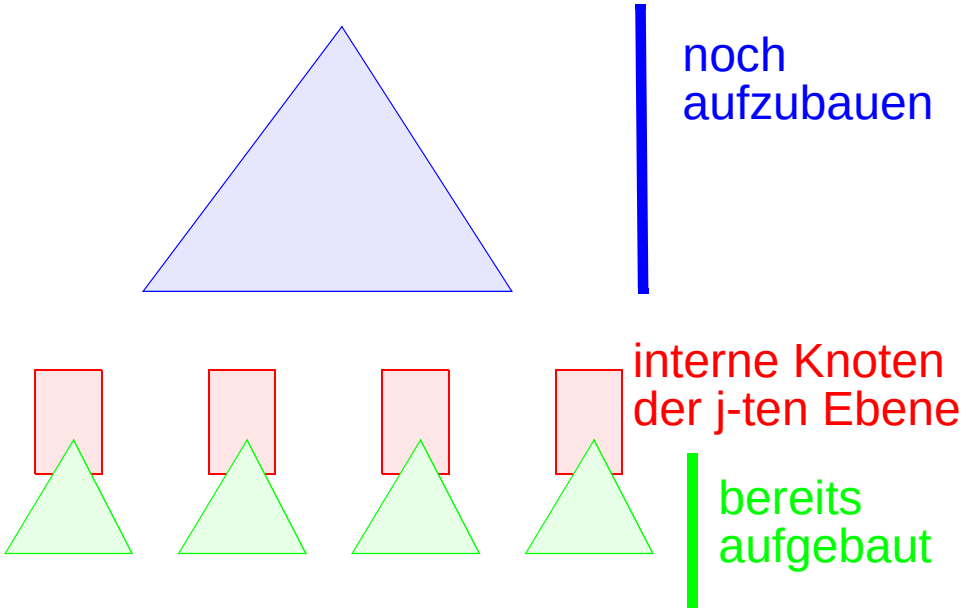
j-te Phase: Aufbau der j-ten Ebene

- Betrachte alle Einträge, die am Ende der (j-1)-ten Phase auf Blattknoten verweisen.
- Füge diese Einträge mit den MBR der Teilbäume in einen neuen Pufferbaum ein.
- Datenknoten des Pufferbaums entsprechen den internen Knoten der j-ten Ebene.

generischer Pufferbaum
(C = Verzweigungsgrad eines inneren Knotens)



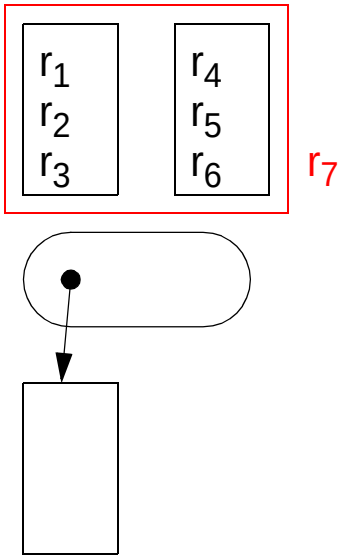
Zielstruktur



Beispiel

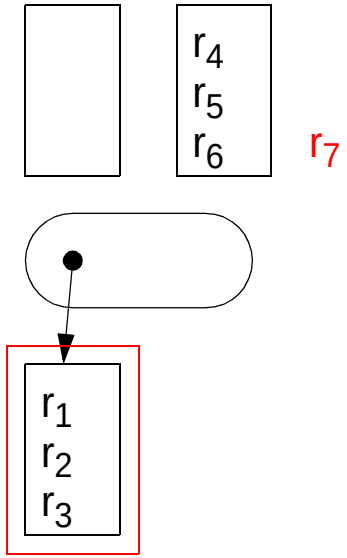
- Parameterwahl:
  - C = 3 (Verzweigungsgrad der inneren Knoten)
  - p = 2 (Pufferkapazität)
  - B = 3 (Kapazität eines Blattknotens)
- Einfügen von 25 Datensätzen $r_1,\ldots r_{25}$ in einen Pufferbaum

Situation am Anfang



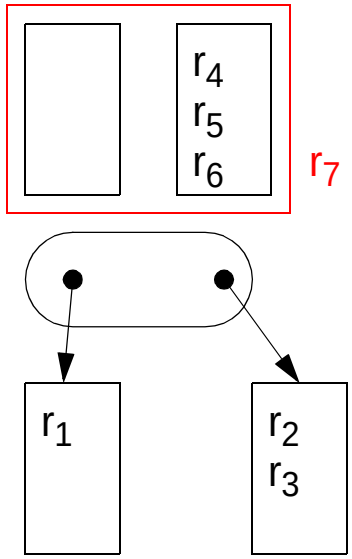
nächste Aktion: Puffer leeren

Situation am Anfang



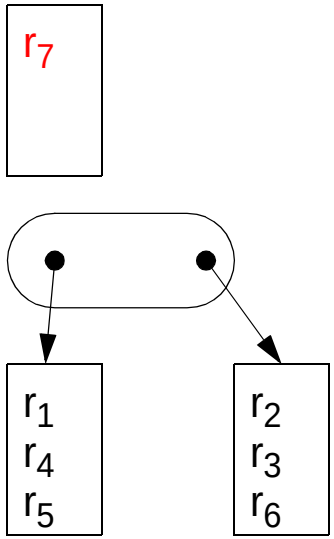
nächste Aktion: Split

Situation am Anfang



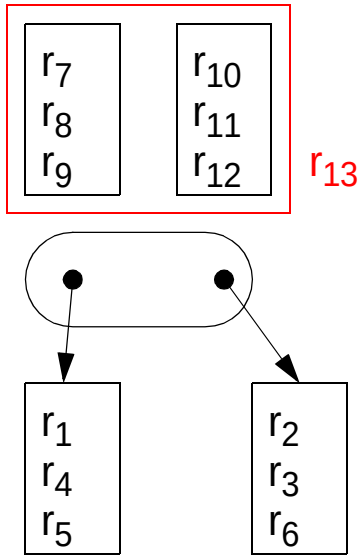
nächste Aktion: mit dem Leeren des Puffers fortfahren

Situation am Anfang



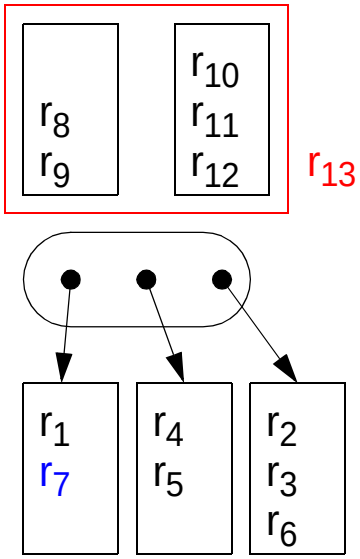
nächste Aktion: neue Einfügeoperationen starten

Leeren des Puffers eines inneren Blattknotens



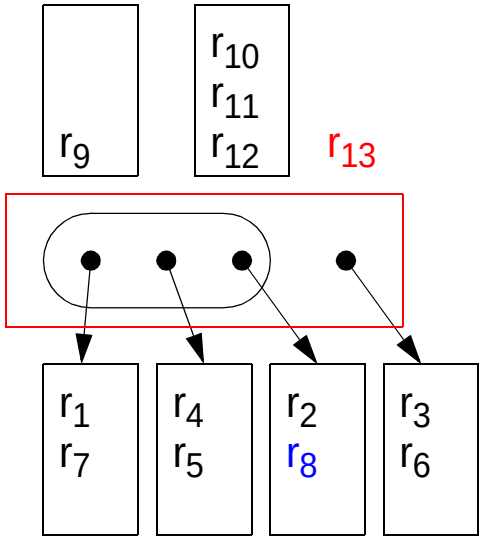
nächste Aktionen: Leeren des Puffers & Split

Leeren des Puffers eines inneren Blattknotens



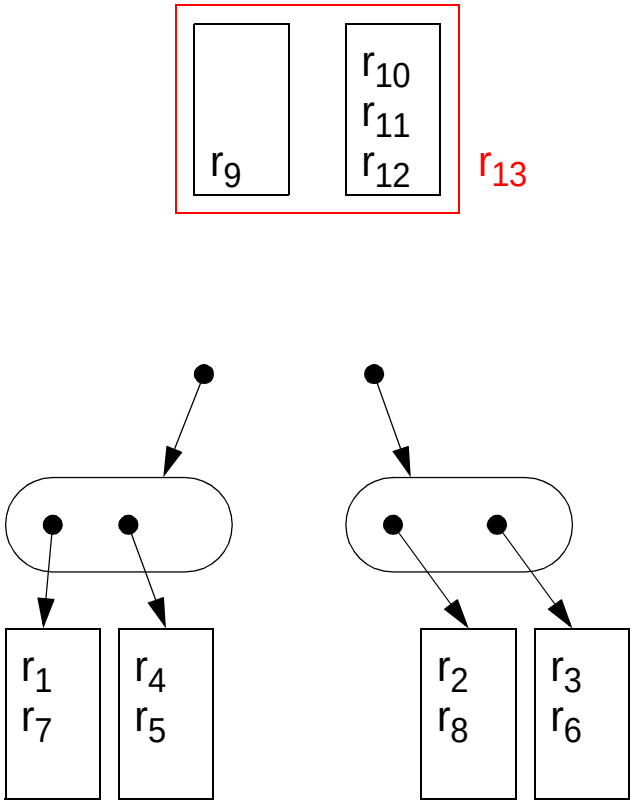
nächste Aktion: mit dem Leeren des Puffers fortfahren

Leeren des Puffers eines inneren Blattknotens



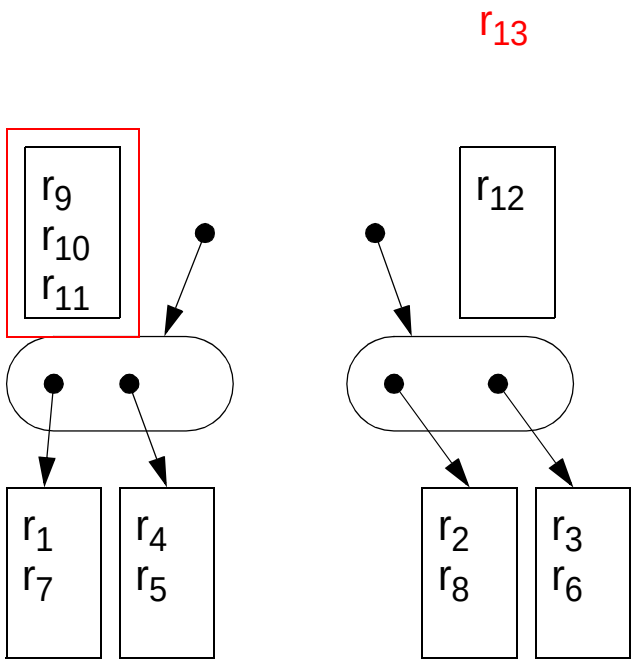
nächste Aktion: Split des inneren Knotens

Leeren des Puffers eines inneren Blattknotens



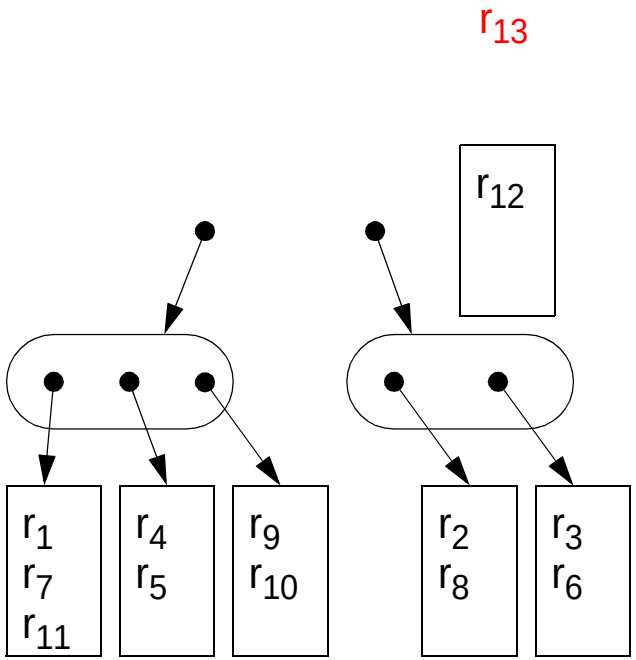
nächste Aktion: Split des Puffers

Leeren des Puffers eines inneren Blattknotens



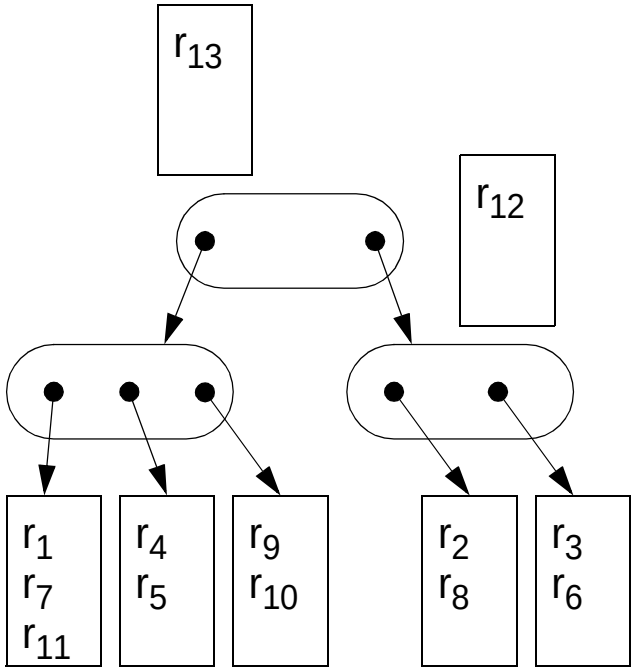
nächste Aktion: Leeren des Puffers

Leeren des Puffers eines inneren Blattknotens



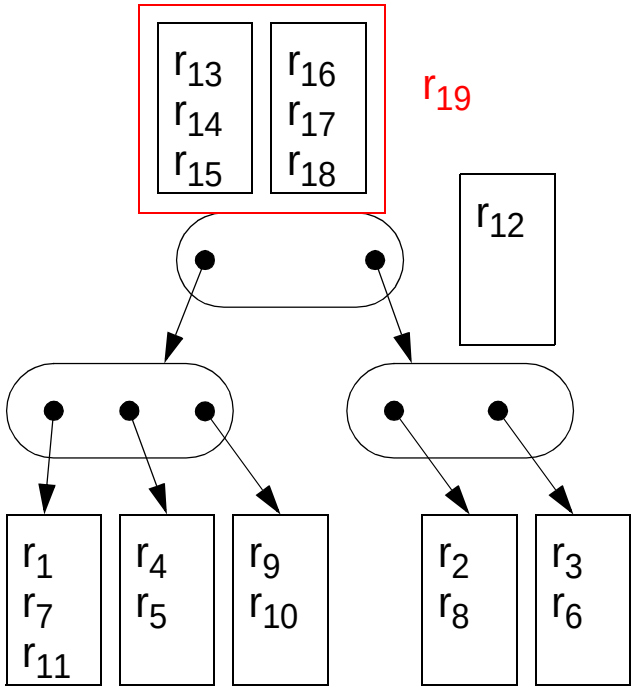
nächste Aktion: Erzeugen einer neuen Wurzel

Leeren des Puffers eines inneren Blattknotens



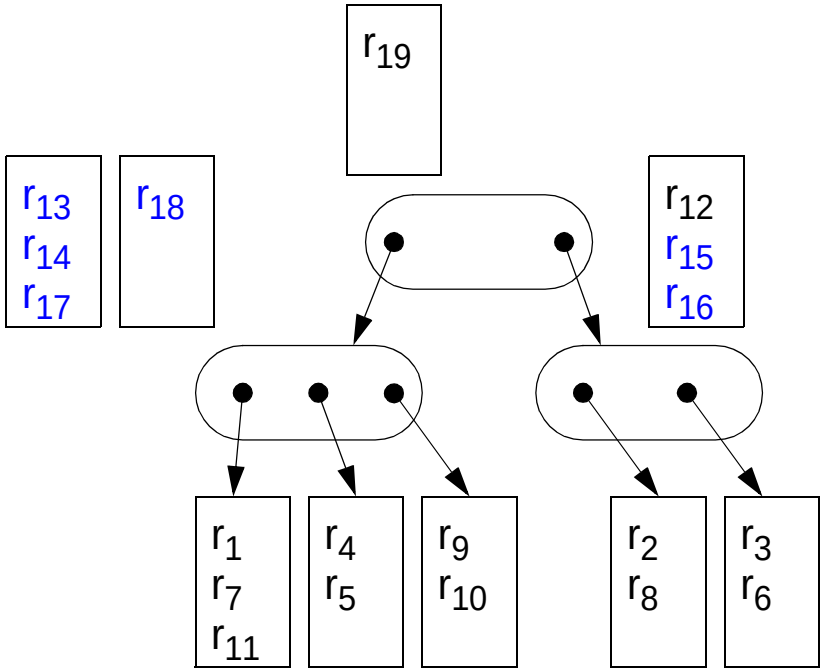
nächste Aktion: Starten einer neuen Einfügeoperation

Leeren des Puffers eines inneren Knotens



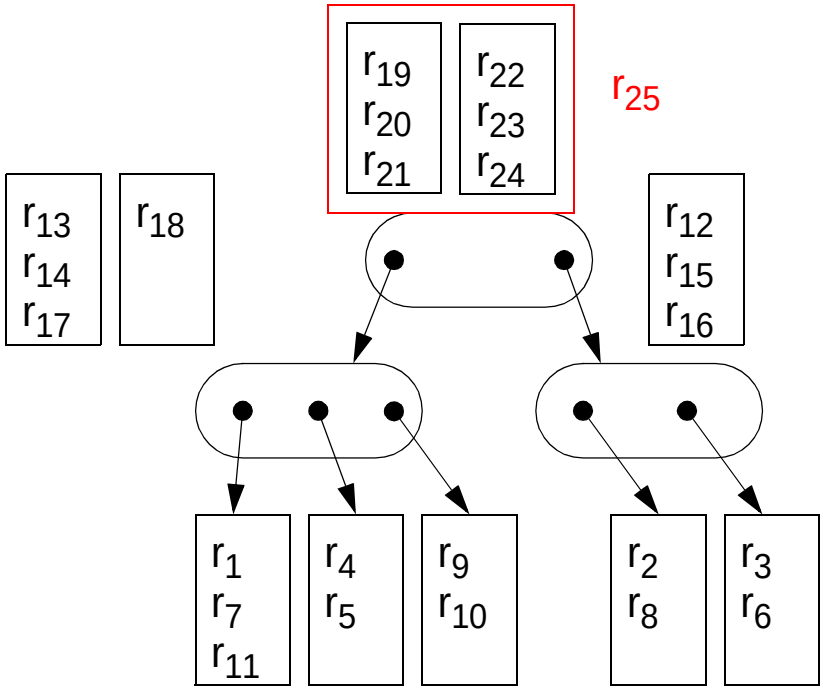
nächste Aktion: Leeren des Puffers

Leeren des Puffers eines inneren Knotens



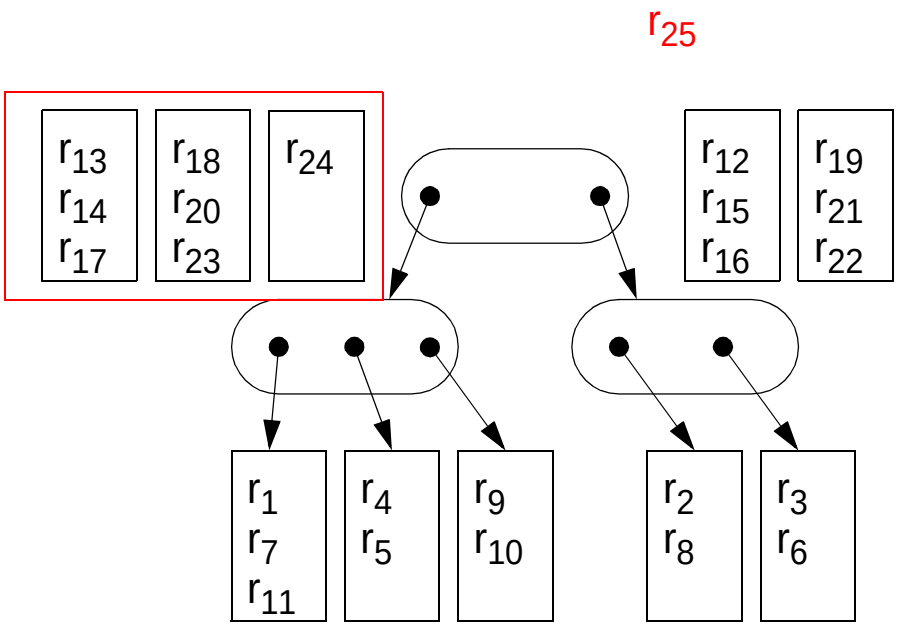
nächste Aktion: Starten von Einfügeoperationen

Leeren des Puffers eines inneren Knotens



nächste Aktion: Puffer leeren

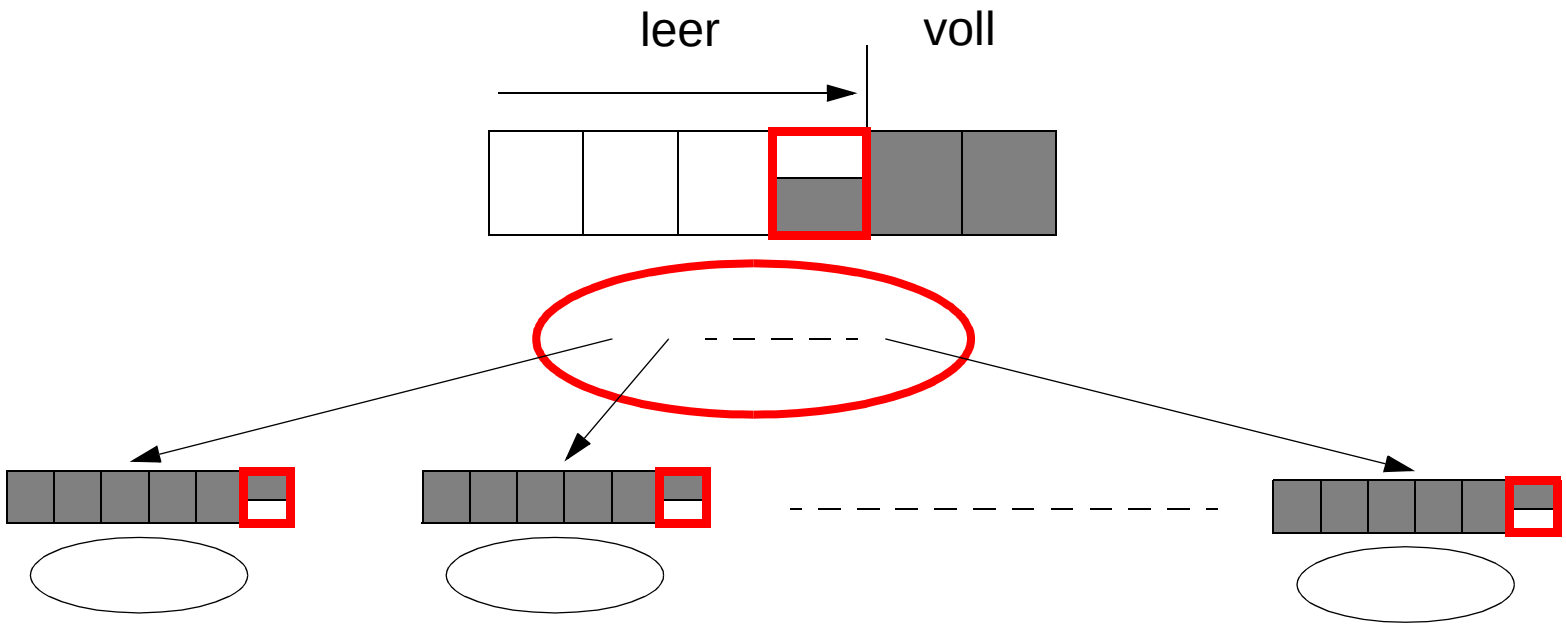
Leeren des Puffers eines inneren Knotens



nächste Aktion: Puffer leeren

Analyse

Wahl des Parameters C



Anforderung

- Jeder Block eines Puffers soll höchstens einmal gelesen und geschrieben werden.

$$B(C+1)+C \leq M$$

Wahl des Parameters p

- p kontrolliert die Größe der Eingabemenge, die in einen Teilbaum auf einmal hineinfließt.
  - p zu klein: Leeren eines Puffers wird zu teuer
  - p zu groß: Aufspalten eines Puffers wird zu teuer

Lemma:

Für $p = \Theta(C)$ kostet das Aufspalten eines Puffers O(*C*) und das Leeren eines Puffers $\Theta(C)$ Zugriffe.

Sind die Entwurfsziele erreicht worden?

Satz:

Für $p = \Theta(M/B)$ und $C = \Theta(M/B)$ gilt: Falls für eine einzelne Einfügeoperation $O(\log_B N)$ Zugriffe benötigt werden, so sind für den Massenaufbau

$$\Theta\left(\frac{N \log_{M/B} N/B}{B}\right)$$

Zugriffe notwendig.

- Beispiele für Zugriffsstrukturen:
  - R-Baum, B+-Baum
- Das Verfahren ist universell für Grow&Post-Trees anwendbar!

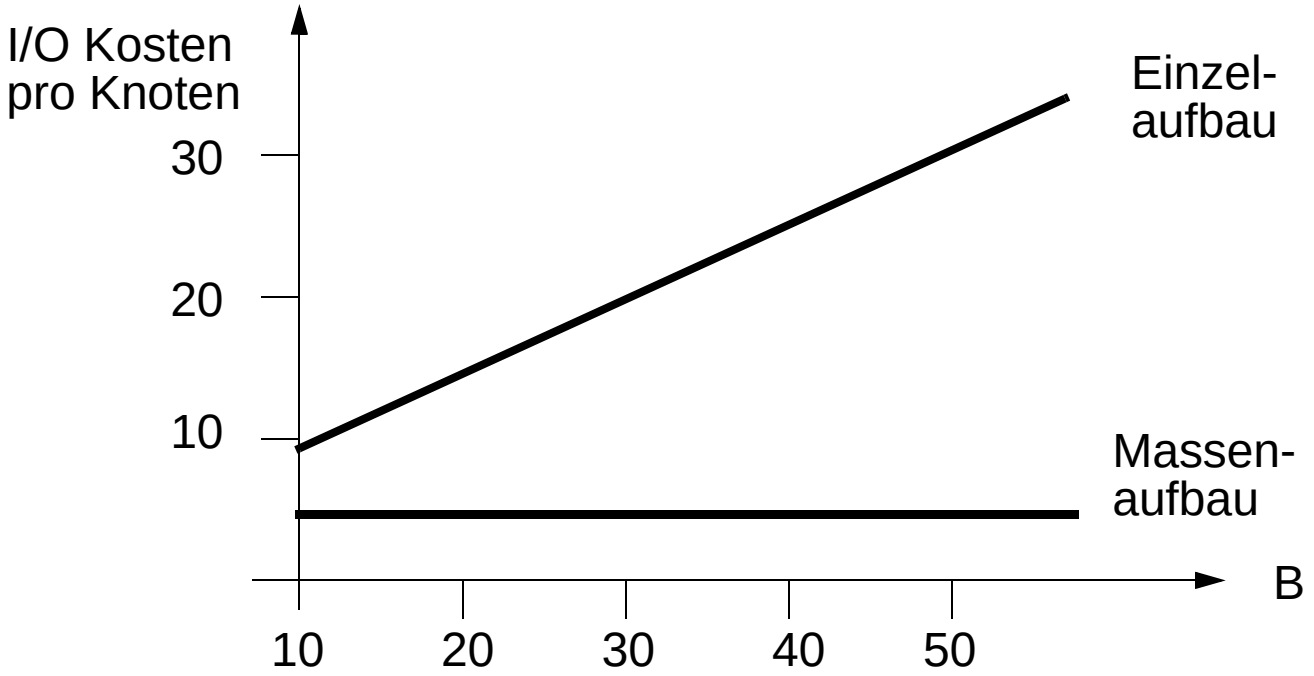
Experimentelle Ergebnisse

- Wahl der Parameter:

  N = 100,000

  M/B = 200

  C = p = 198



Zusammenfassung

- Wichtige Eigenschaften mehrdimensionaler Zugriffspfade
  - Erhaltung der topologischen Struktur des Datenraumes
  - Adaption an die Objektdichte
  - Dynamische Reorganisation
  - Balancierte Zugriffsstruktur
  - Unterstützung von verschiedenen Anfragetypen: Fenster- u. Nachbaranfrage
- Darstellung von räumlichen Objekten
  - Abstraktion zu punktförmiger Repräsentation ist Regelfall
  - "Ausgedehnte" Darstellung nur in grober Annäherung
- Notwendigkeit der Integration von "konventionellen" und mehrdimensionalen Zugriffspfaden in Nichtstandard-DBS
  - Anfrageoptimierung
  - Mehrbenutzerbetrieb
  - Ansätze existieren für den ZB+-Baum und den R-Baum
- Erprobung der praktischen Tauglichkeit von mehrdimensionalen Zugriffsstrukturen (R-Baum) steht noch aus.