5. 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!

    ➥aber 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**

  A) überlappende Regionen: R-Bäume, R*-Bäume

  B) disjunkte Regionen ➥ Clipping: R^+-Bäume

  C) Transformationsansätze

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

R-Bäume

- **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 Robotik

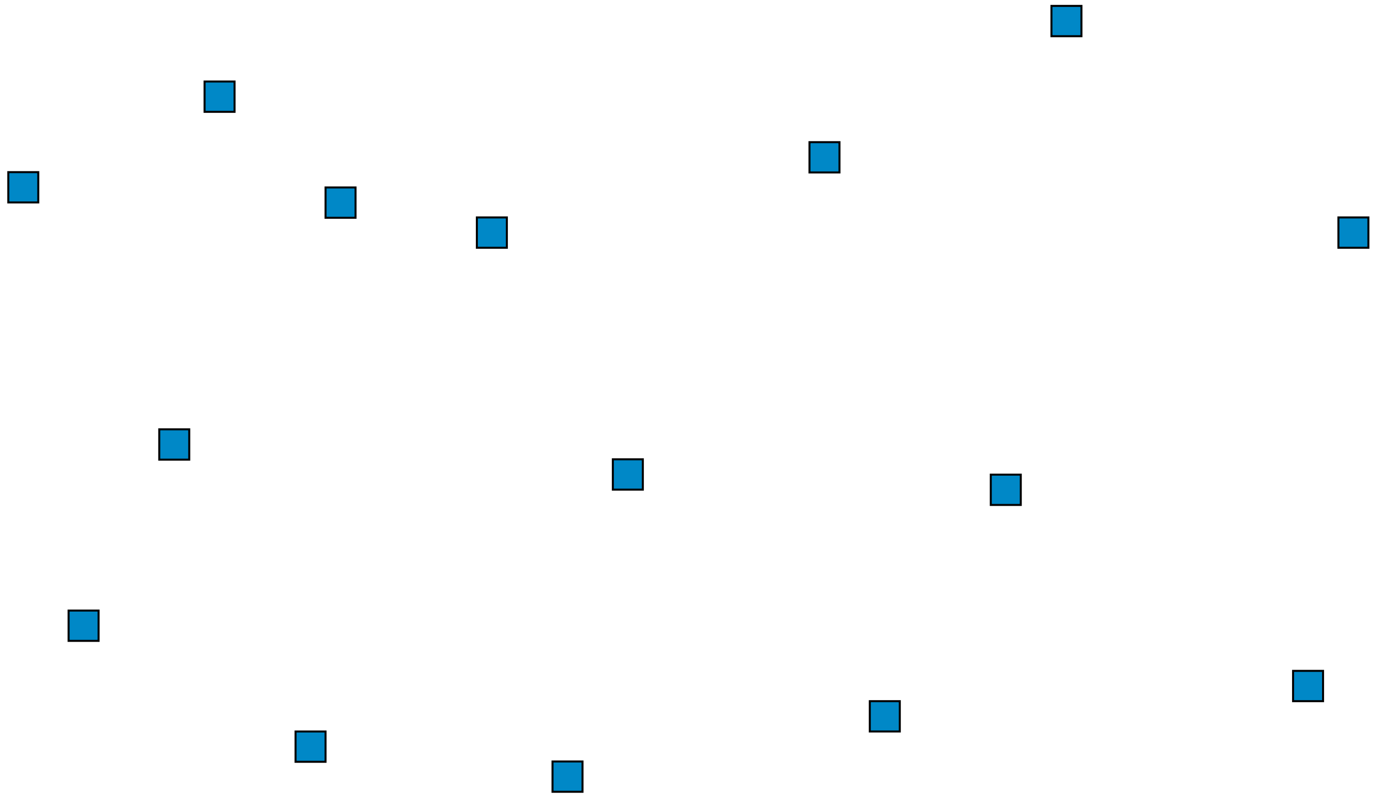
- **Ansatz:** Speicherung und Suche von achsenparallelen Rechtecken
  - Objekte werden durch Datenrechtecke repräsentiert und müssen durch kartesische Koordinaten beschrieben werden
  - Repräsentation im R-Baum erfolgt durch minimale begrenzende (k-dimensionale) Rechtecke/Regionen
  - Suchanfragen beziehen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

---

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

Beispiel für einen R-Baum*

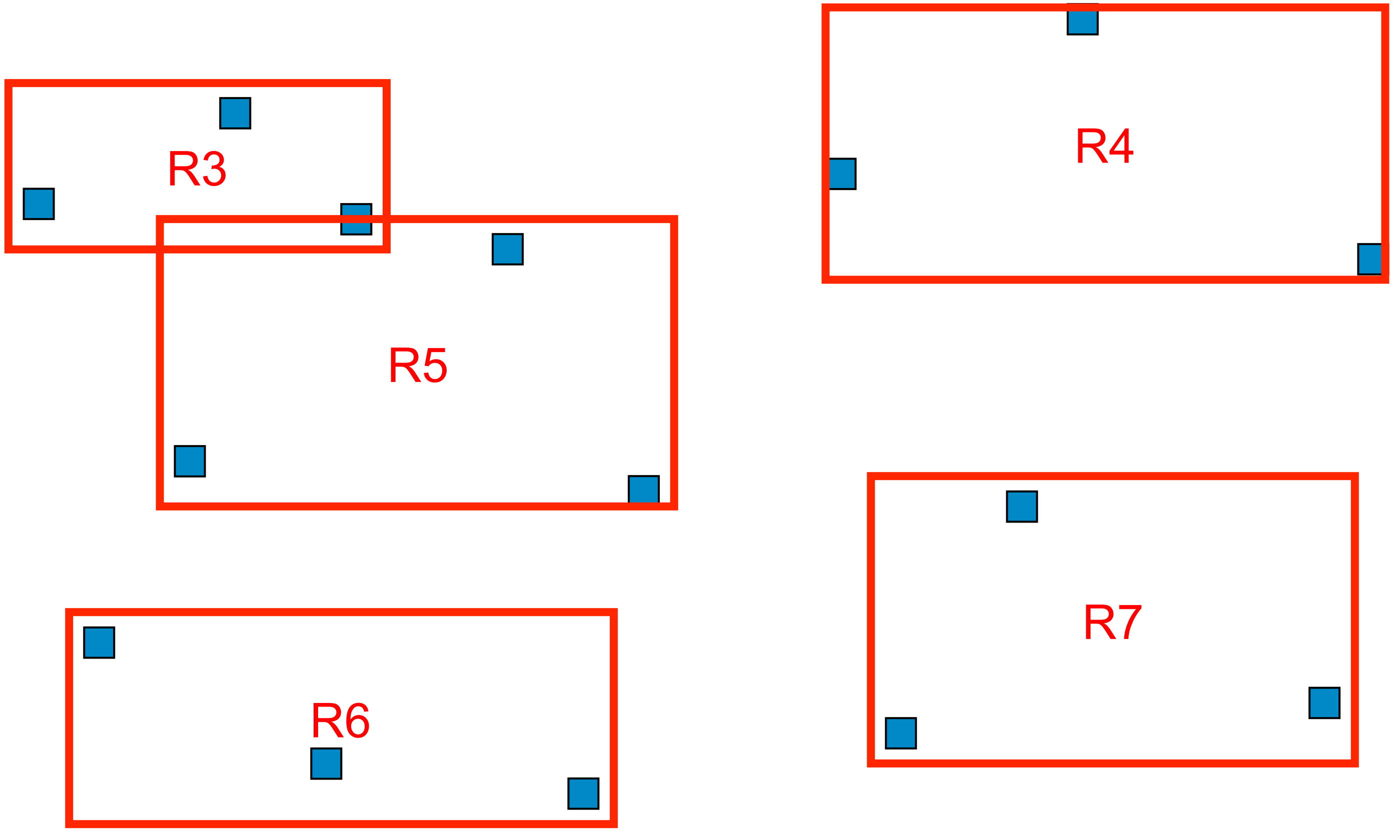
Abzuspeichern: folgende Rechteckmenge; min./max. Füllzahl pro Knoten = 2/3



*) Folien von M.Wimmer, TU München

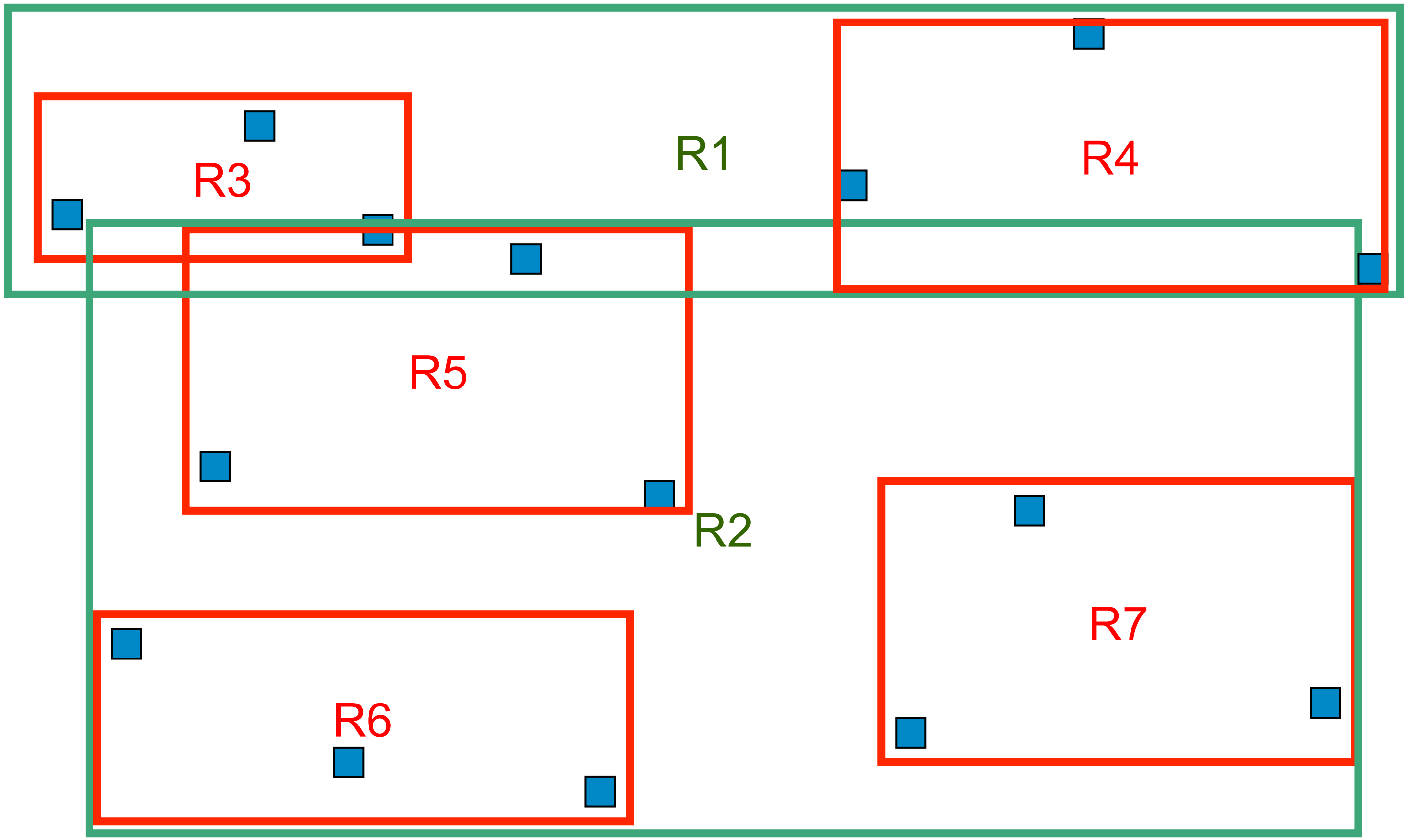
Baumstruktur

(Die jeweils größeren Rechtecke sind kleinste umgebende Rechtecke der ganz innenliegenden Rechtecke.)

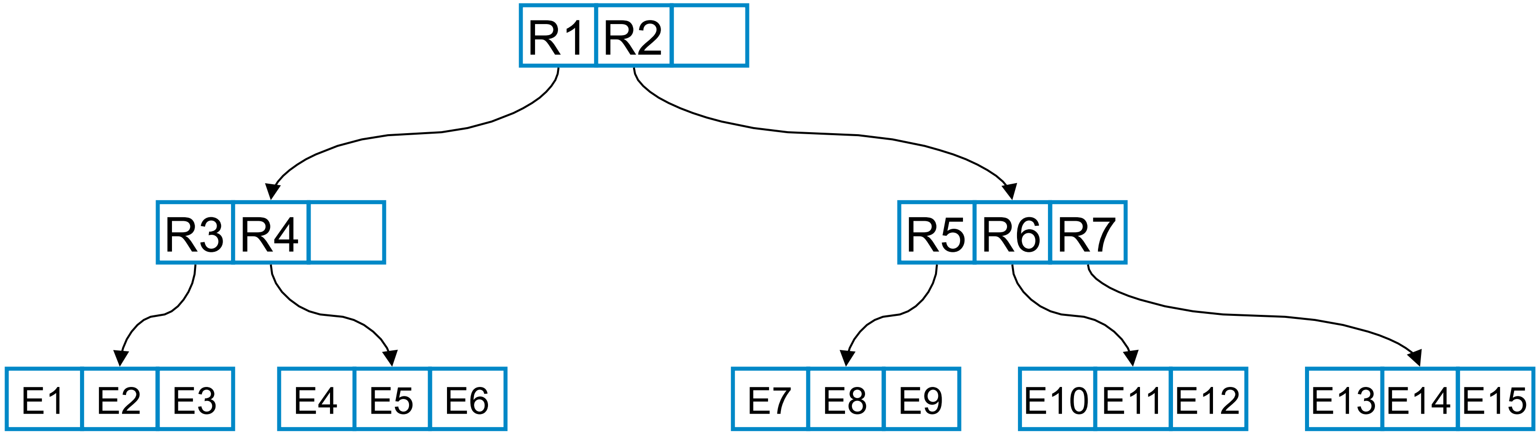


Baumstruktur (Forts.)

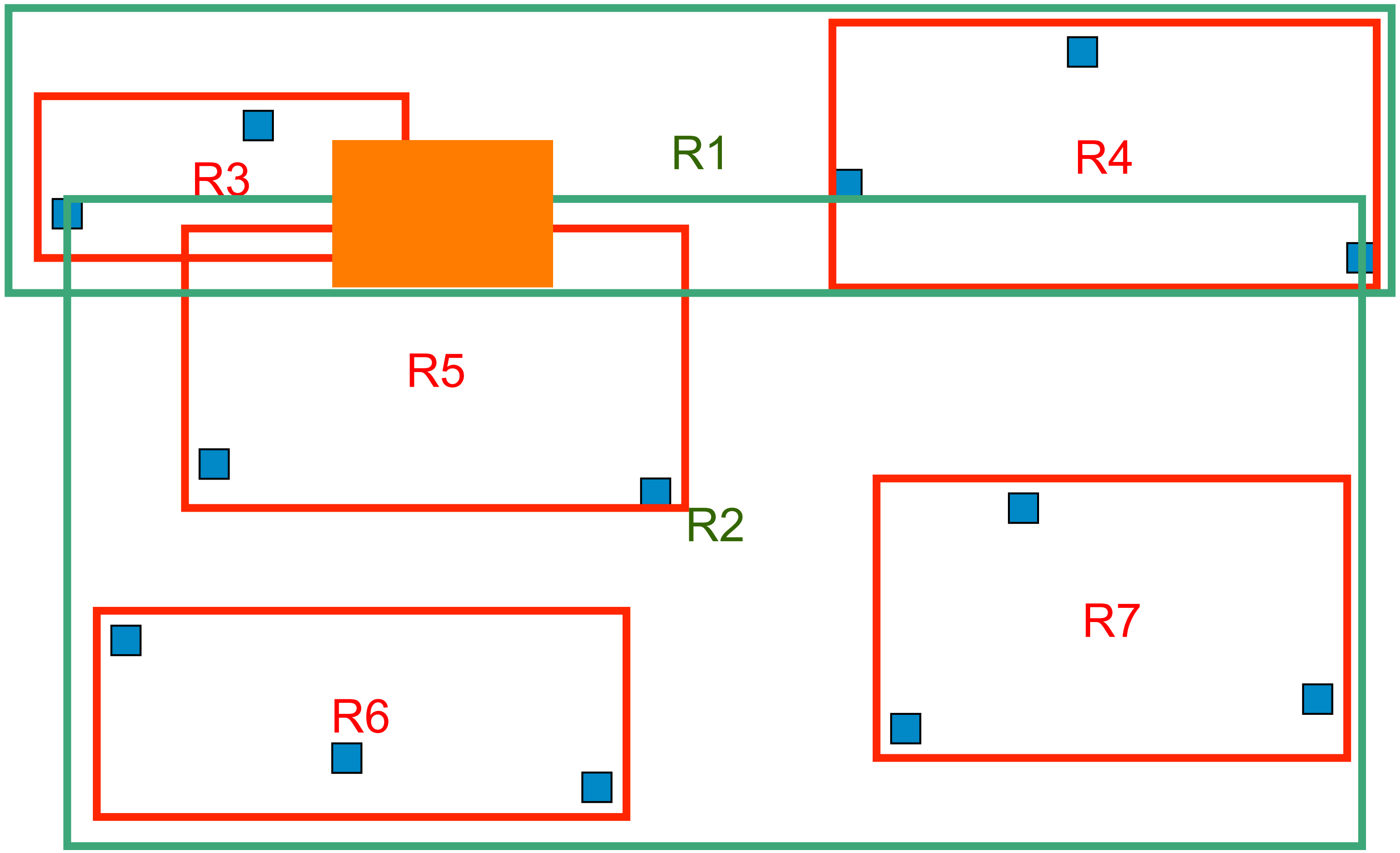
(Die jeweils größeren Rechtecke sind kleinste umgebende Rechtecke der ganz innenliegenden Rechtecke.)



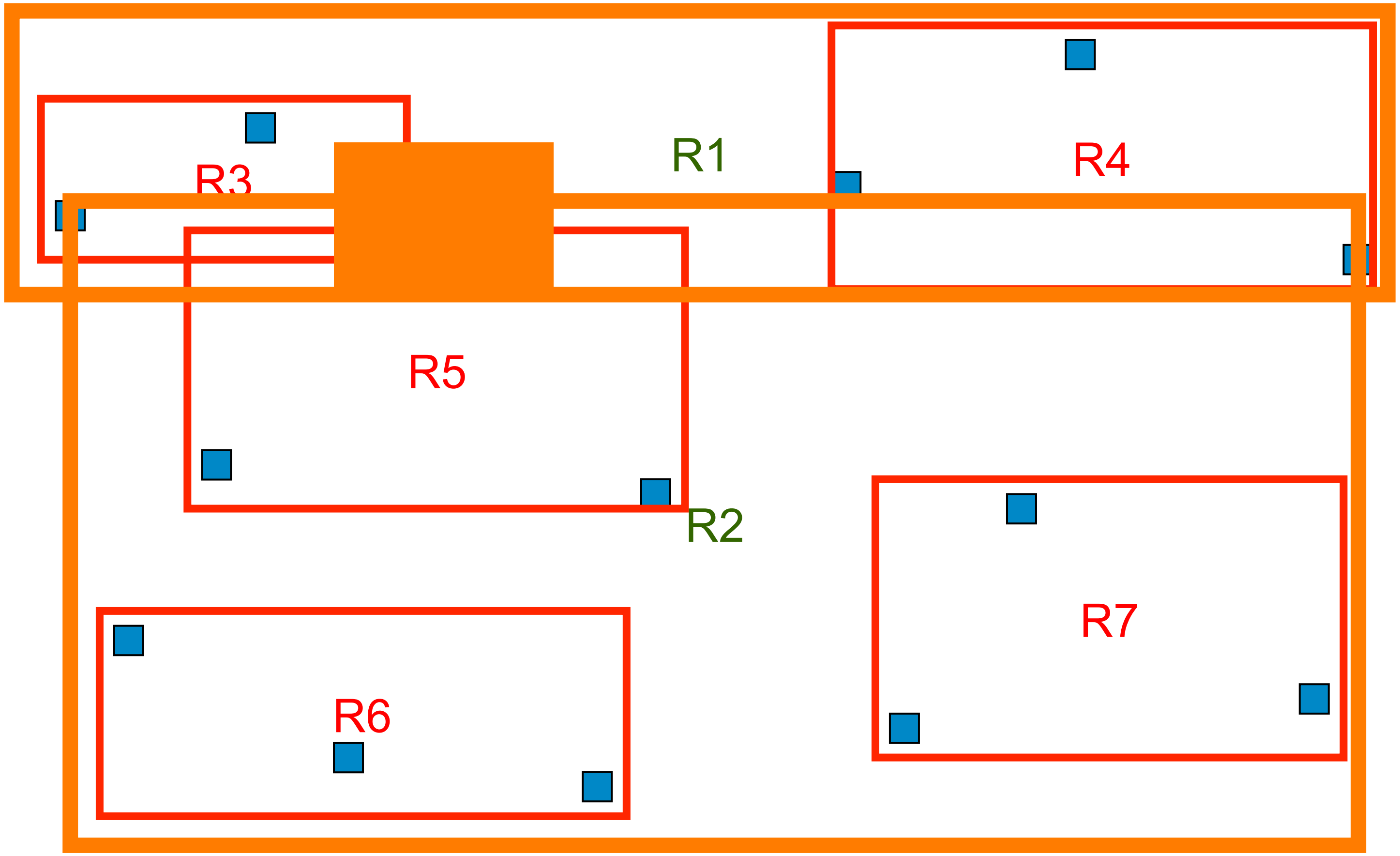
Baumstruktur (Forts.)



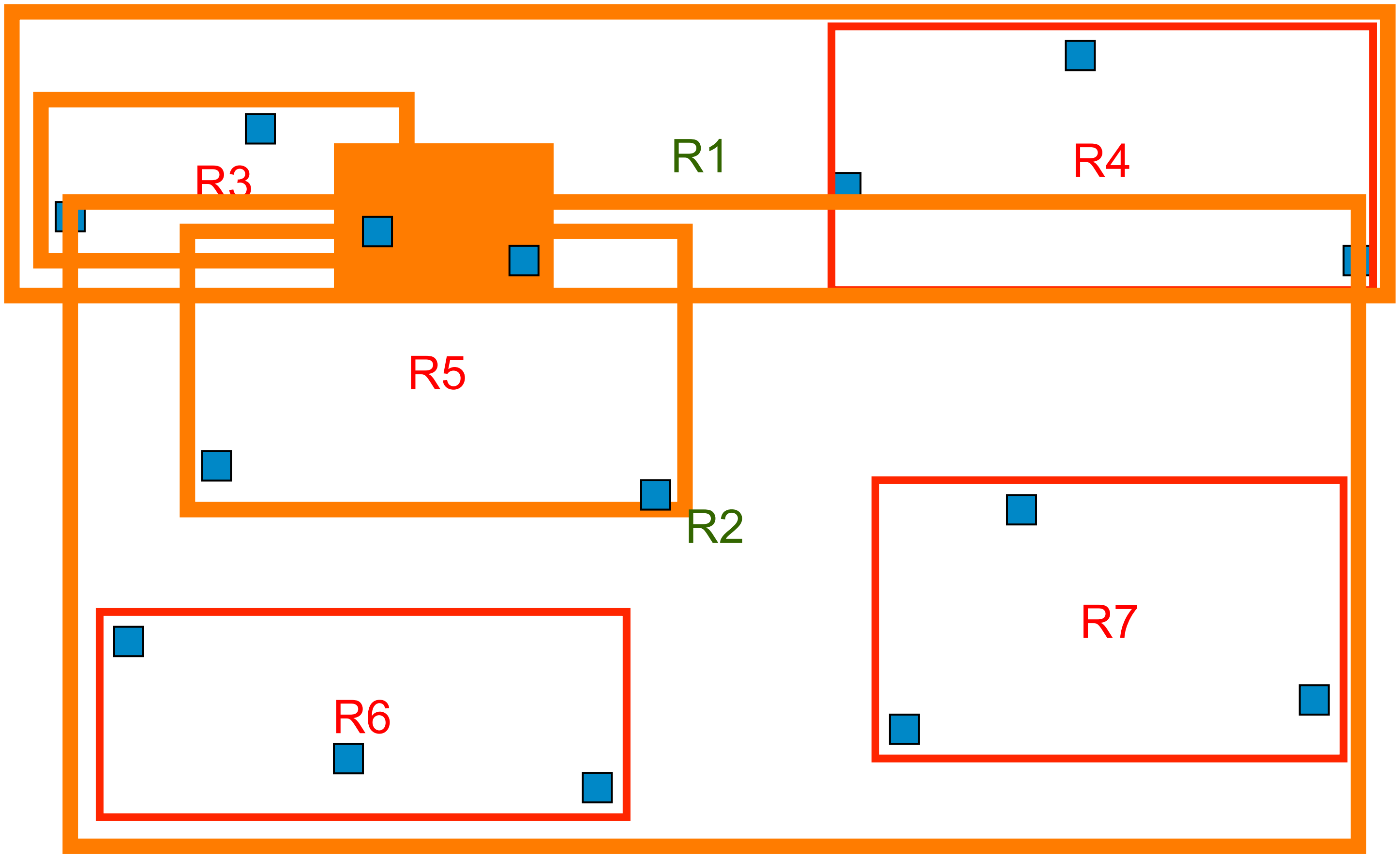
Erfolgreiche Suche



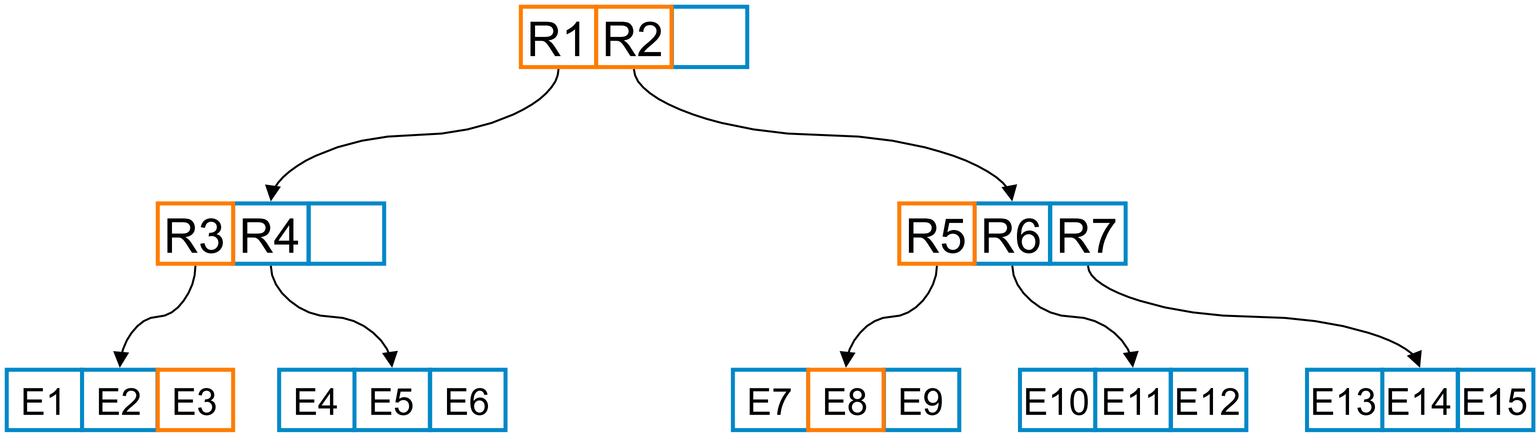
Erfolgreiche Suche (Forts.)



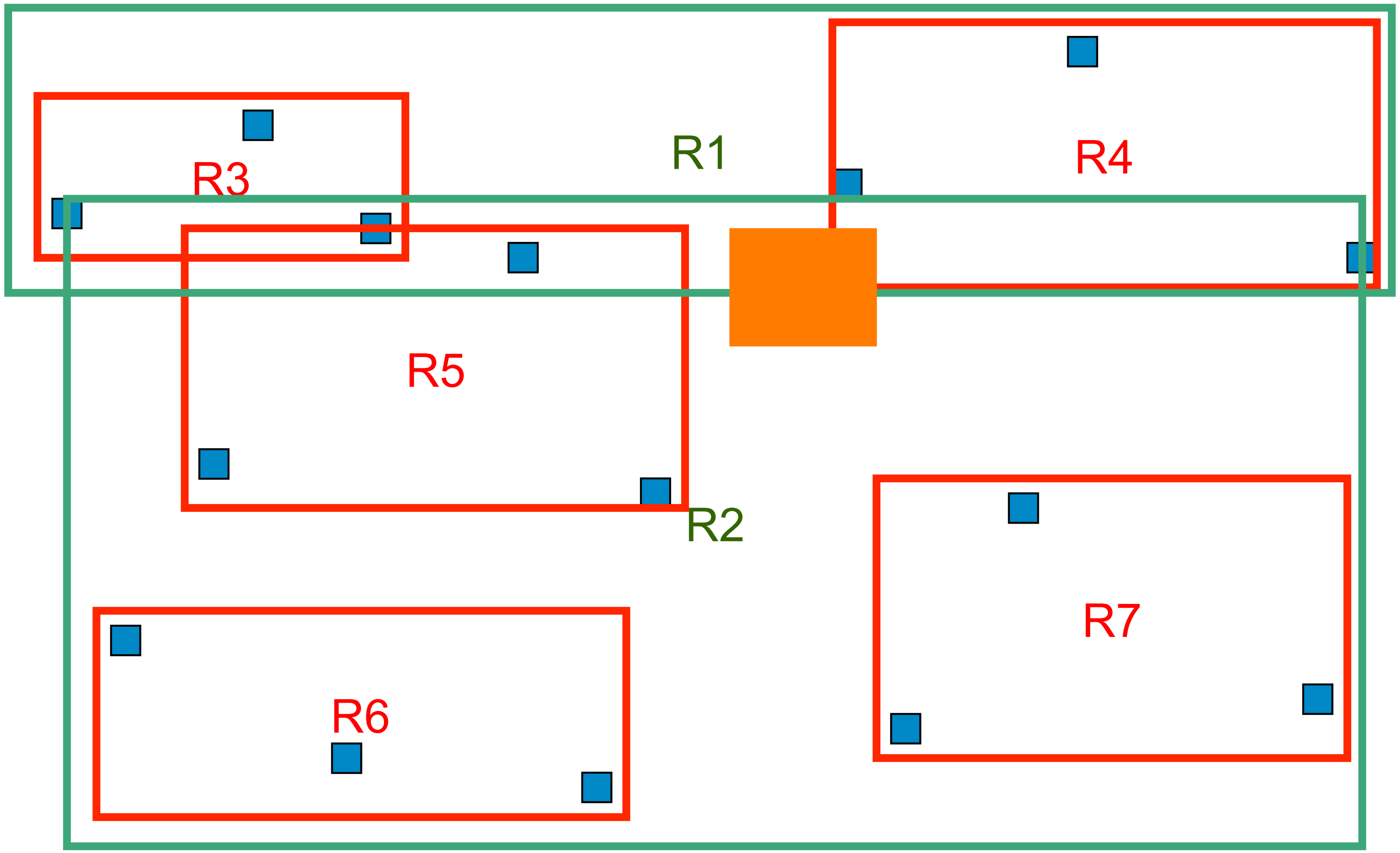
Erfolgreiche Suche (Forts.)



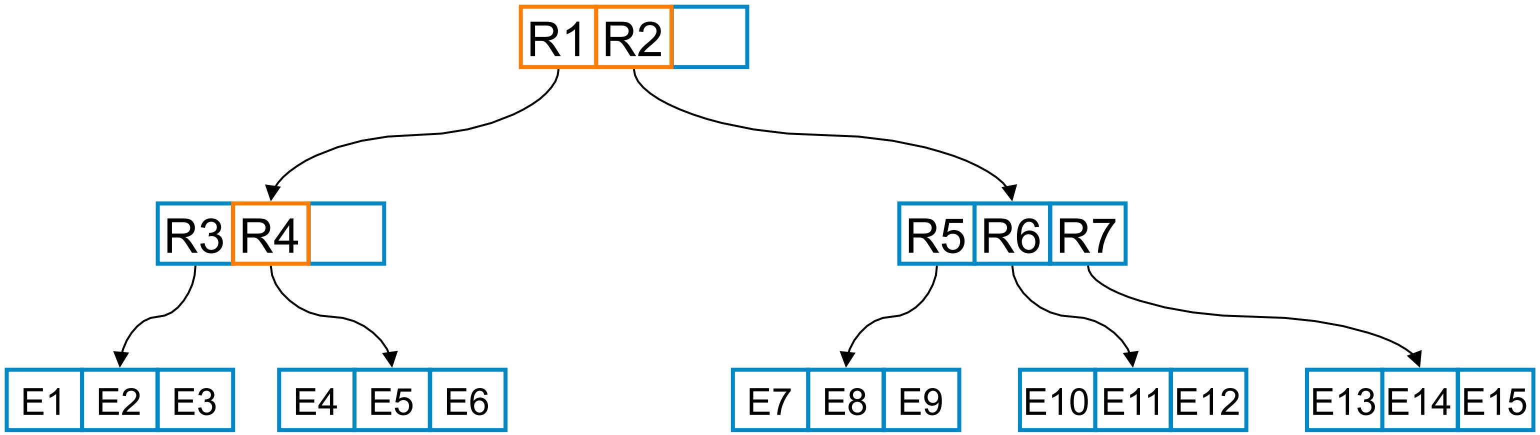
Erfolgreiche Suche (Forts.)



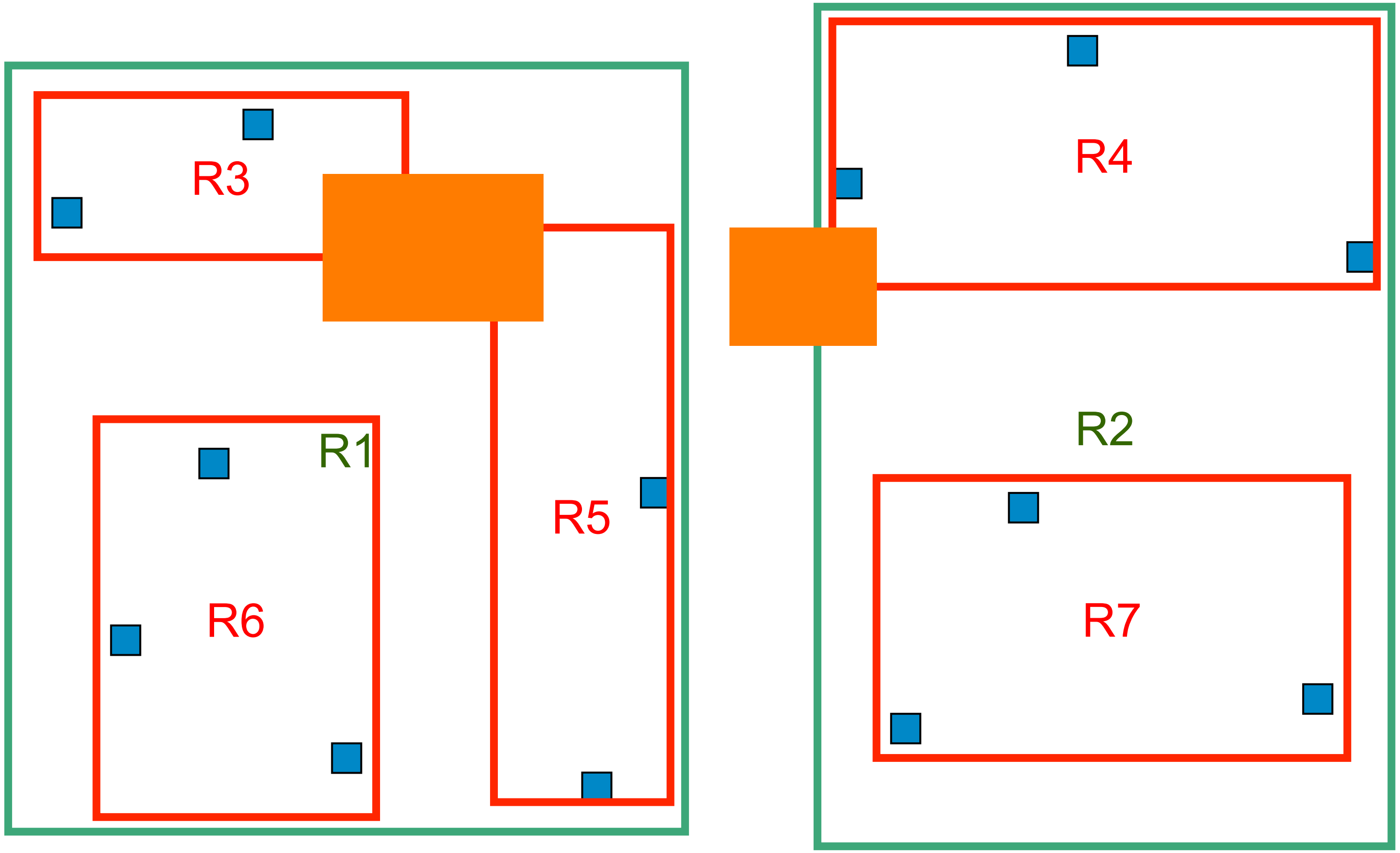
Erfolglose Suche



Erfolglose Suche (Forts.)



Bessere Aufteilung ?



R-Bäume (Forts.)

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

Blattknoteneintrag:

I_1	I_2	• • •	I_k	OID

kleinstes umschreibende Region/
Rechteck für Objekt OID
(Datenrechteck)

Zwischenknoteneintrag:

I_1	I_2	• • •	I_k	succ

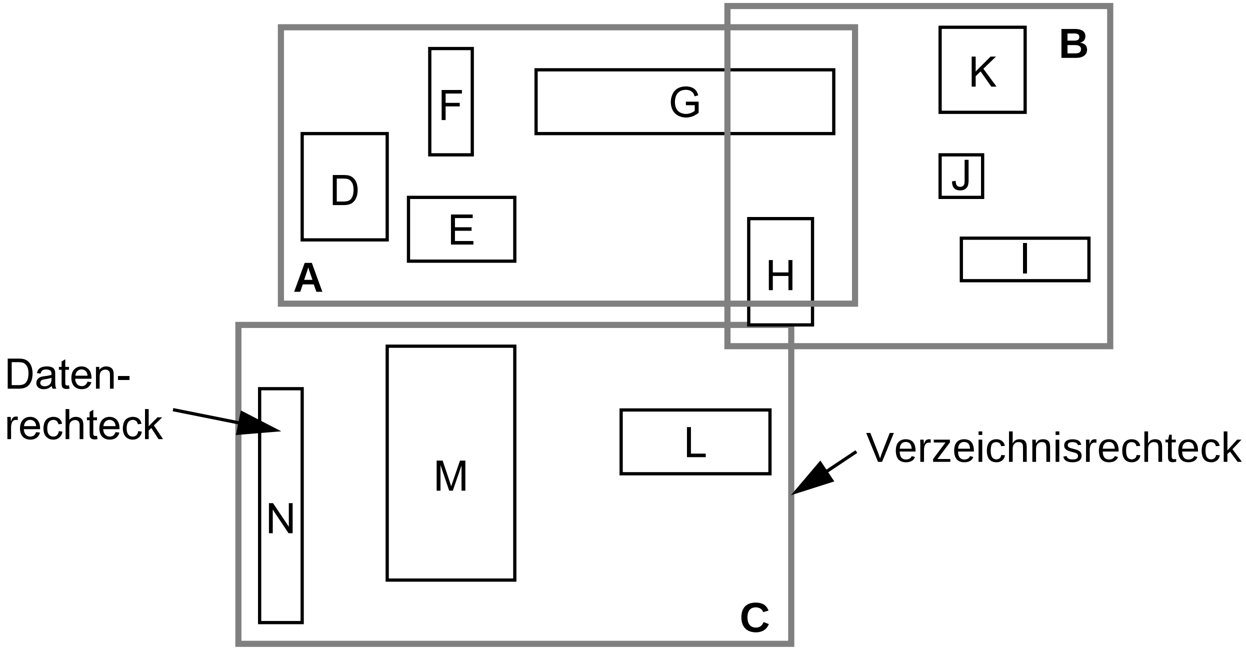
Intervalle beschreiben kleinste um-
schreibende Region/Rechteck für
alle im Teilbaum 'succ' enthaltenen
Objekte (Verzeichnisrechteck)

I_j = geschlossenes Intervall bzgl. Dimension j

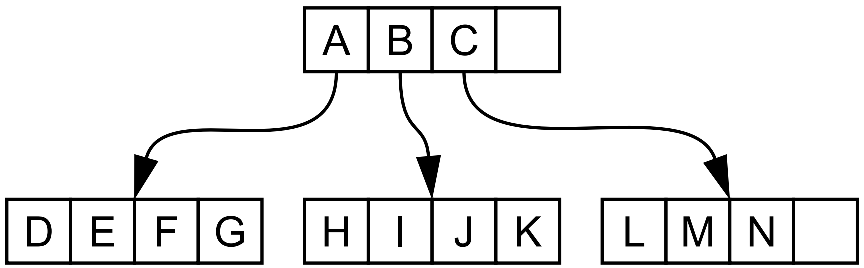
OID: Verweis auf Objekt   succ: Verweis auf Nachfolger/Kind

- **Eigenschaften von R-Bäumen**
  - 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 (siehe unten)

Beispiel 1:

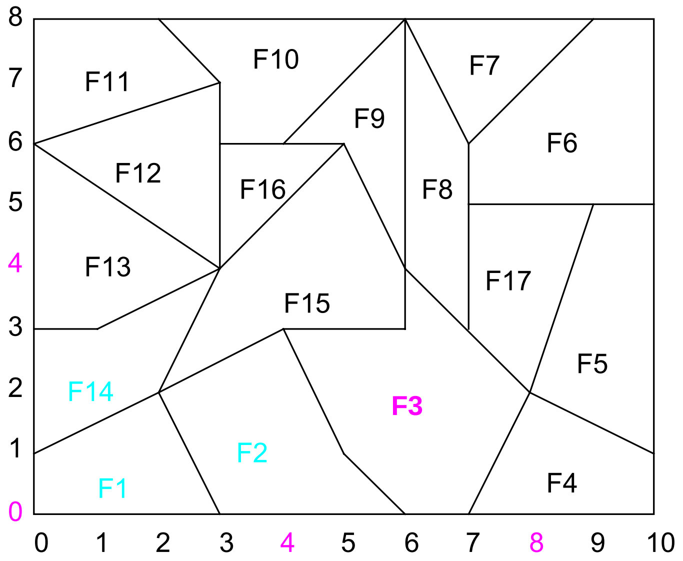


zugehöriger R-Baum:

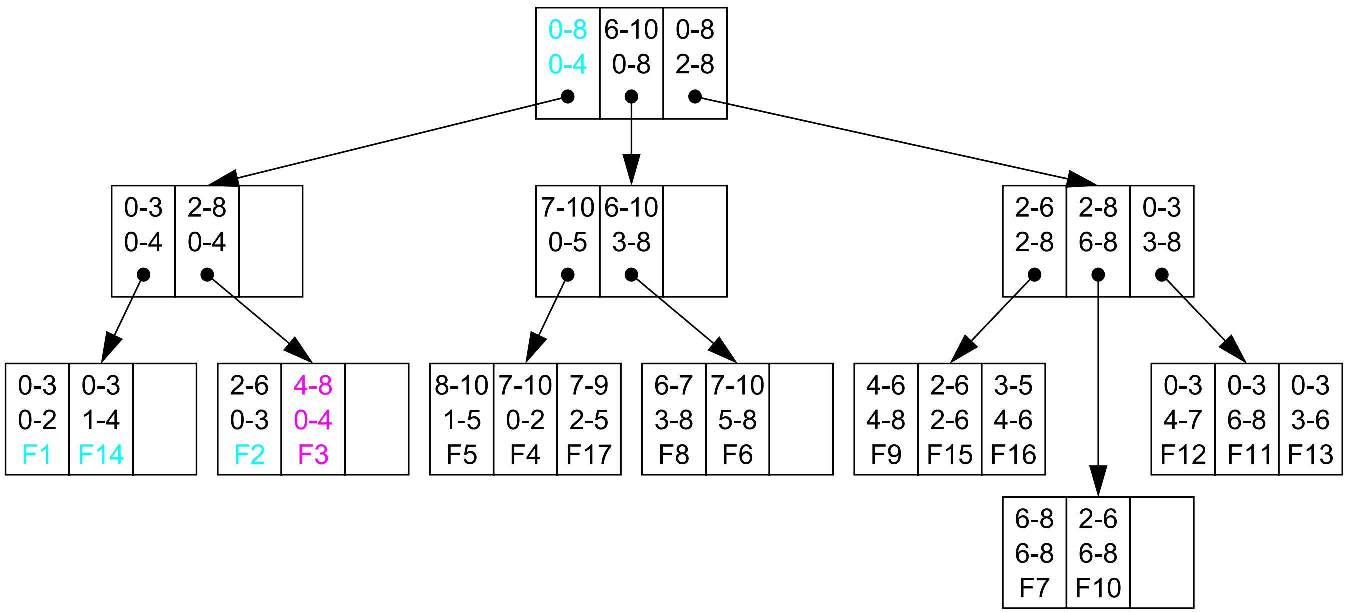


Beispiel 2:

- **Abzuspeichernde Flächenobjekte**



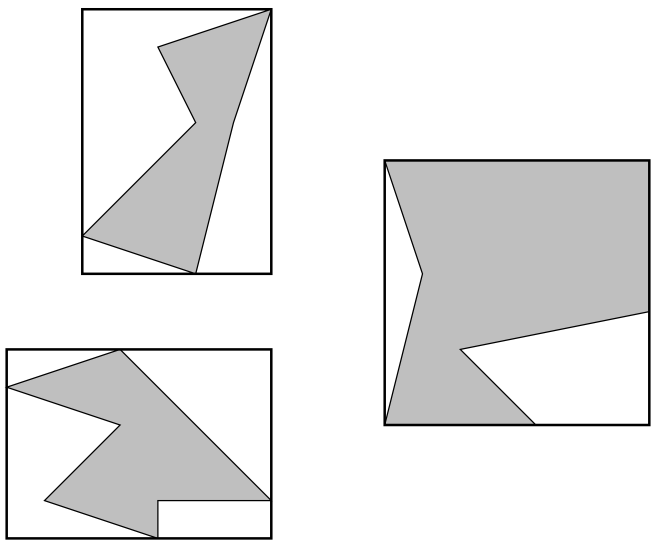
- **Zugehöriger R-Baum**



R-Baum-Algorithmen

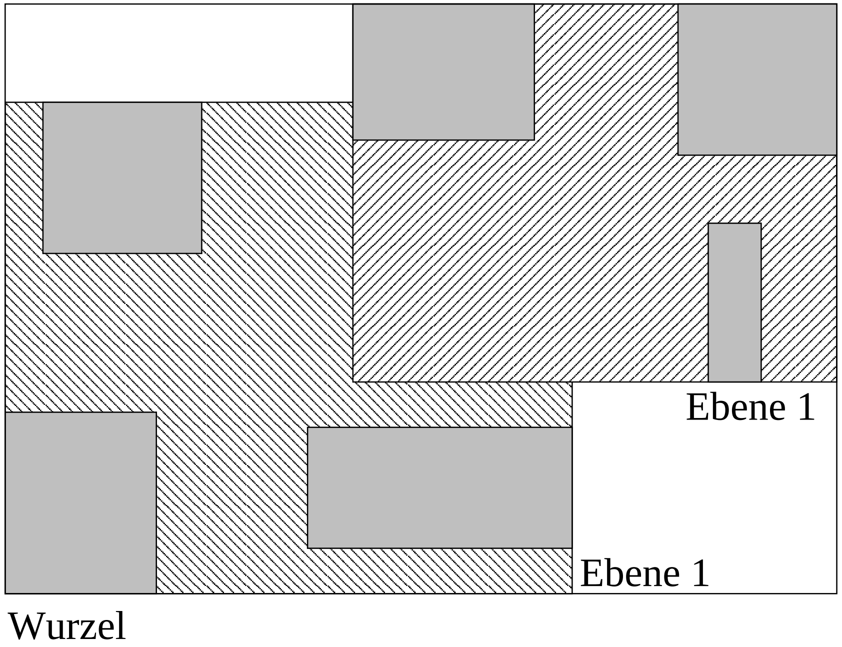
Aufbau eines R-Baums

- Blätter in einem R-Baum haben Einträge der Form *(dataRect, OID)*. Hierbei ist *dataRect* das Datenrechteck *(data rectangle)* des Objekts *OID*, d.h. im zweidimensionalen Fall das kleinste, das Objekt umgebende Rechteck.

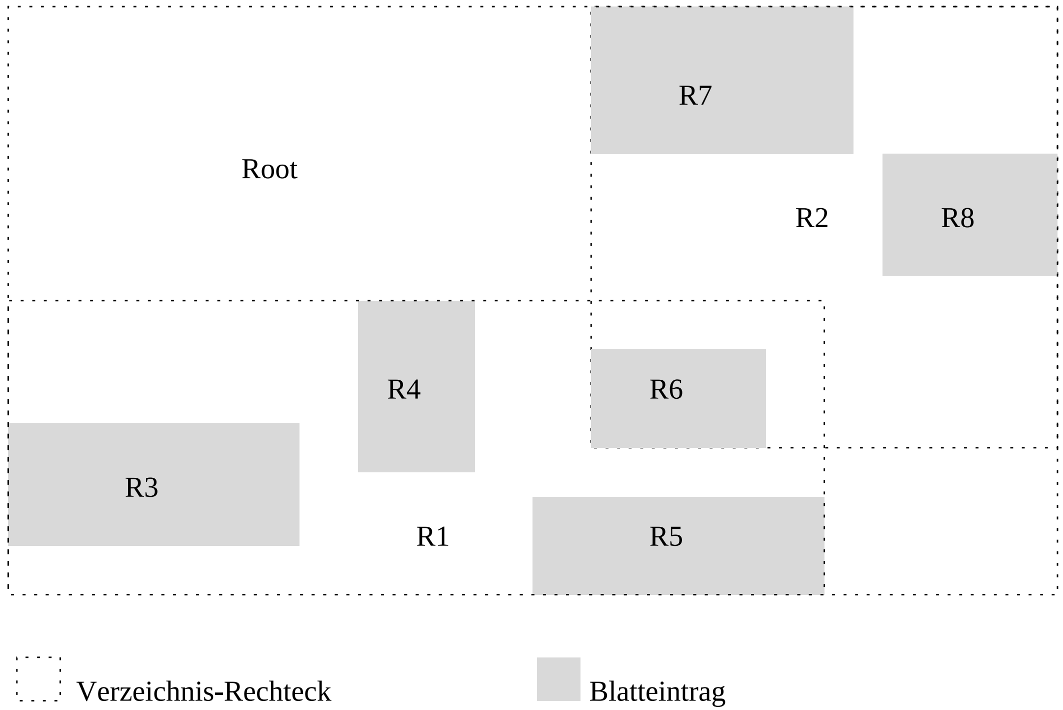


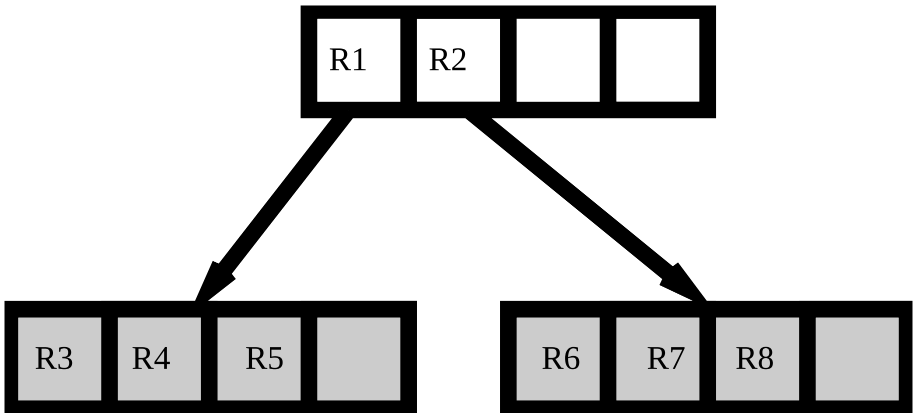
Aufbau eines R-Baums (Forts.)

- Innere Knoten haben Einträge der Form *(dirRect, succ)*. Hierbei ist *dirRect* das Verzeichnisrechteck des Nachfolgerknotens *succ*. Das Verzeichnisrechteck eines Knotens ist das kleinste umgebende Rechteck um die Rechtecke aller seiner Einträge.



Beispiel:


R7
Root
R2
R8
R4
R6
R3
R1
R5
Verzeichnis-Rechteck
Blatteintrag


R1
R2
R3
R4
R5
R6
R7
R8

Weitere R-Baum-Spezifikationen

1. Außer der Wurzel haben alle Knoten $m \le i \le M$ Einträge ($m \approx \frac{M}{2}$).
2. Die Wurzel hat mindestens 2 Nachfolger oder ist ein Blatt.
3. Alle Blätter liegen auf der gleichen Stufe.

Folgerung: Die Höhe eines R-Baums mit N Index-Einträgen ist höchstens $\lceil \log_m N \rceil$.

R-Baum-Algorithmen

Suchen im R-Baum

Suchanfragen werden im R-Baum mit einem Suchrechteck gestellt. Zu finden sind alle Blatteinträge, deren Verzeichnisrechteck das Suchrechteck schneidet, enthält oder in ihm enthalten ist.

Dass eine Ergebnismenge leer ist, kann im besten Fall schon auf höheren Stufen entschieden werden.

Meist muss in mehrere Zweige abgestiegen werden, um alle Ergebnisse aufzusammeln, da Verzeichnisrechtecke nicht disjunkt sein brauchen. Deshalb kann keine nicht-triviale Abschätzung für das Laufzeitverhalten im *worst-case* angegeben werden.

R-Baum-Algorithmen (Forts.)

Einfügen *(Insert)*

Algorithmus zum Einfügen eines Eintrags E:

I1 Suche mit *ChooseLeaf* ein Blatt L, in welches E eingefügt werden kann.

I2 Wenn $\#L < M$ dann füge E ein, sonst rufe *SplitNode* auf, um zwei Blattknoten $L1$ und $L2$ zu erhalten, welche E und alle alten Einträge von L enthalten.

I3 Rufe *AdjustTree* mit der Information "$L1$ und $L2$ statt L" auf.

I4 Wenn die Wurzel gespalten werden muß, erzeuge eine neue Wurzel, deren Kinder die beiden Knoten der aufgespaltenen alten Wurzel sind.

Einfügen *(ChooseLeaf)*

Steige im Baum bis zu einem Blatt ab; dabei:

Wähle den Nachfolger zu dem Verzeichnisrechteck, welches die kleinste Vergrößerung benötigt (bestenfalls keine Vergrößerung), um den neuen Eintrag E zu überdecken.

Wenn das nicht eindeutig ist, dann wähle das Rechteck mit der kleinsten Fläche.

Einfügen *(AdjustTree)*

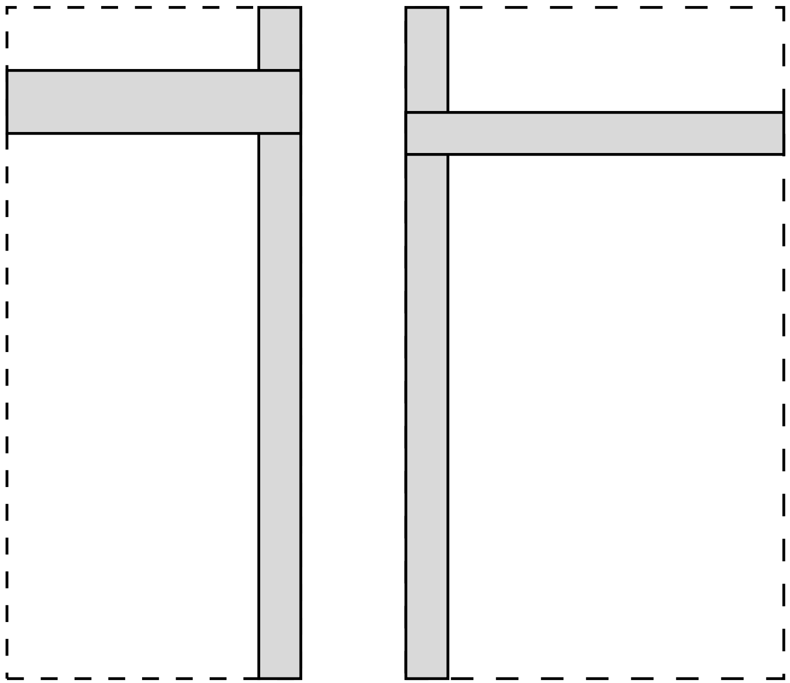
Steige im Baum auf, solange sich etwas ändert; dabei:

Passe das Verzeichnisrechteck des Vorgängereintrags an die Änderungen im Nachfolgerknoten an.

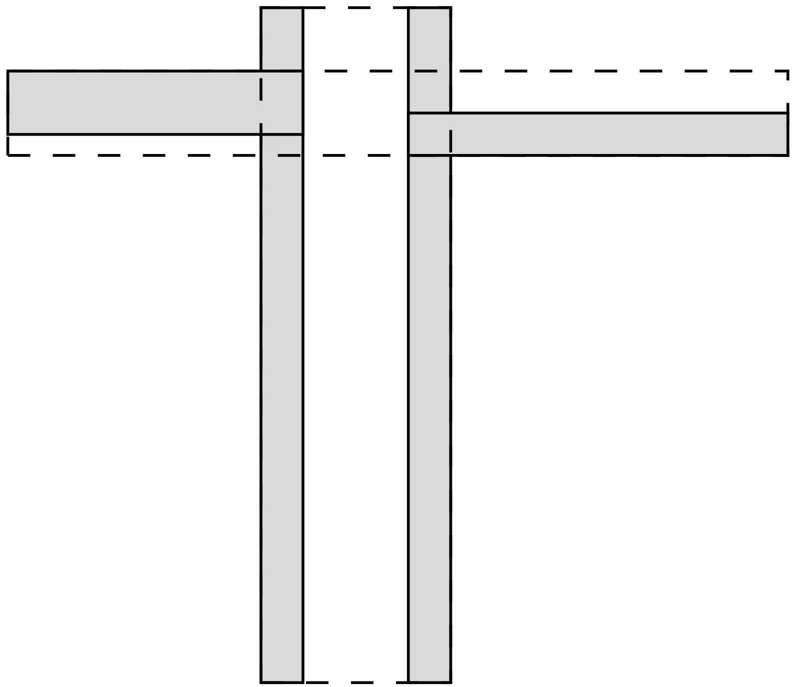
Wenn der Nachfolgerknoten gespalten wurde, füge einen neuen Eintrag in den Vorgängerknoten ein. Wenn dieser mehr als M Einträge enthält, so spalte auch diesen mit *SplitNode* (analog oben).

Einfügen *(SplitNode)*

Wenn ein Knoten $M + 1$ Einträge enthält, muss er gespalten und seine Einträge neu verteilt werden.



Schlechte Aufspaltung



Gute Aufspaltung

Einfügen *(SplitNode)* (Forts.)

Der triviale Algorithmus: Erzeuge alle möglichen Gruppierungen (=Teilmengen und ihr Komplement) und wähle die beste. $\Longrightarrow$ Laufzeit $O(2^M)$

Besser: **Einfügen *(QuadraticSplit = SplitNode mit* $O(M^2)$*)***

QS1 Wähle mit *PickSeeds* die zwei ersten Elemente der beiden Knoten.

QS2 Ende, wenn alle Einträge zugewiesen wurden, oder alle verbleibenden Einträge einem Knoten zugewiesen werden müssen, damit dieser mindestens m Einträge hat.

QS3 Wähle mit *PickNext* das nächste Element. Weise es

1. dem Knoten zu, dessen Verzeichnisrechteck am wenigsten vergrößert werden muß, sonst (falls nicht eindeutig)
2. dem Knoten zu, dessen Verzeichnisrechteck kleiner ist, sonst
3. dem Knoten zu, der weniger Elemente enthält, sonst
4. einem beliebigen Knoten zu.

Weiter mit **QS2**.

Einfügen *(SplitNode)* (Forts.)

Einfügen *(PickSeeds)*

PS1 Berechne die verschwendete Fläche f für jedes Paar (a, b) von Einträgen.

$$f = \text{Fläche}(\boxed{a, b}) - \text{Fläche}(a) - \text{Fläche}(b)$$

PS2 Wähle das Paar mit dem größten f als Startknoten.

Einfügen *(PickNext)*

PN1 Für alle verbliebenen Einträge berechne den Flächenzuwachs d_1 und d_2, wenn man es der ersten bzw. der zweiten Gruppe zuordnen würde.

PN2 Wähle das Element mit der größten Differenz zwischen d_1 und d_2.

Einfügen *(SplitNode)* (Forts.)

Oder: Einfügen (*LinearSplit = SplitNode mit* $O(M)$)

LinearSplit ist analog zu *QuadraticSplit*, benutzt aber ein anderes *PickSeeds* und statt *PickNext* wird ein beliebiger verbliebener Eintrag gewählt.

LinearPickSeeds

LPS1 Für jede Dimension: Finde die zwei Rechtecke, welche die kleinste Obergrenze und die größte Untergrenze annehmen.

LPS2 Normalisiere die Abstände der gefundenen Rechtecke durch Division mit der Strecke zwischen Minimal- und Maximalwert in der jeweiligen Dimension.

LPS3 Wähle das Paar mit dem größten normalisierten Abstand in einer Dimension.

R-Baum-Algorithmen (Forts.)

Löschen *(Delete)*

Löschen eines (Blatt-) Eintrags E:

D1 Finde Blattknoten K mit Eintrag E.

D2 Entferne E aus K.

D3 Solange K noch nicht Wurzel:

D3.1 Falls K nun noch $> m$ Einträge enthält, passe überdeckendes Rechteck im Vorgänger V an;
sonst entferne gesamten Knoten (!) sowie dessen überdeckendes Rechteck aus V, aber merke zugehörige Einträge (ggfs. mit deren Teilbäumen).

D3.2 $K := V$

D4 Falls K Wurzel und K nur noch ein Kind hat, mache dieses Kind zur neuen Wurzel.

Löschen *(Delete)* (Forts.)

D5 Füge gemerkte Einträge wieder ein: *("forced reinsert")*

Lokalisiere passende Blätter bzw. innere Knoten (auf gleicher Höhe wie vorher!) und füge gemerkte Einträge dort ein;

dann korrigiere / balanciere Baum jeweils nach oben hin wie beim Einfügen üblich.

R-Baum-Algorithmen (Forts.)

Optimierungskriterien

K1 Flächenminimierung der Verzeichnisrechtecke

K2 Minimale Überlappung der Verzeichnisrechtecke

K3 Randminimierung der Verzeichnisrechtecke

K4 Optimierung der Speicherausnutzung

In den klassischen R-Baum-Algorithmen wird überwiegend nach **K1** optimiert.

Die erfolgreichste Variante von R-Bäumen wurde i.w. durch experimentelle Neukombination dieser Optimierungskriterien gefunden: **R*-Bäume**

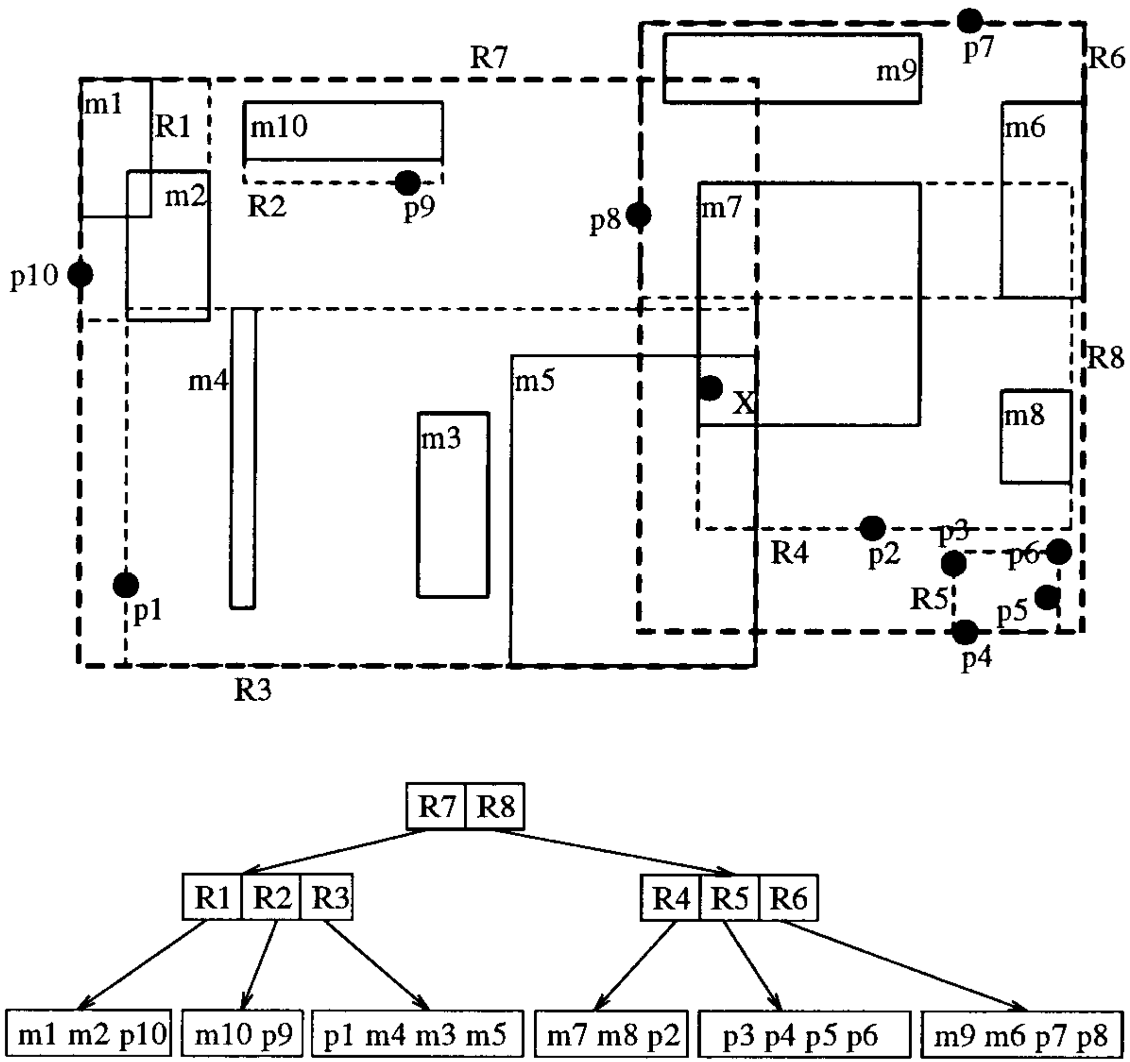


Fig. 3.26. R-tree

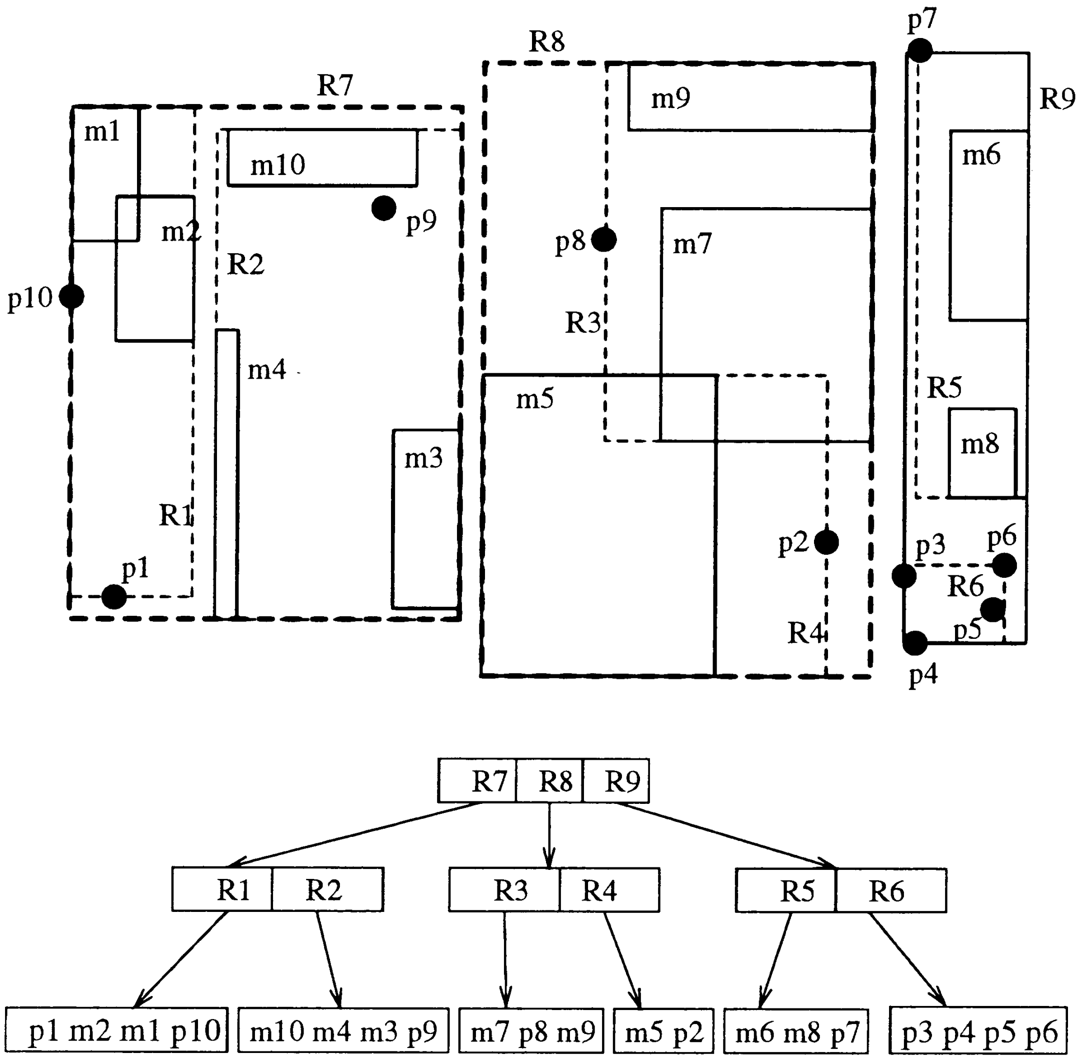


Fig. 3.27. R*-tree

Genaueres zu R*-Bäumen

- R*-Bäume sind eine anerkannte Verbesserung von **R**-Bäumen.
- Einige der folgenden Größen (insbes. p, q, r) bzw. Entscheidungen erscheinen zwar plausibel, sind aber letztlich durch (bestätigte) Experimente bestimmt[1]
- *ChooseSubtree/Leaf:*

  Definiere für einen Eintrag E_k in einem Knoten mit Einträgen $E_1, \ldots, E_n$:

  $$\text{Überlappung}(E_k) := \sum_{l=1\ldots n,\ k\neq l} \text{Fläche}(\boxed{E_k} \cap \boxed{E_l})$$

  Bei Wahl eines Nachfolgers direkt oberhalb Blattebene:

  wähle Eintrag, der geringste Überlappungsvergrößerung (**) bewirkt;
  falls nicht eindeutig: Eintrag mit geringster Flächenvergrößerung

  bei Wahl eines Nachfolgers höher im Baum:

  wähle Eintrag, mit geringster Flächenvergrößerung
- aber (**) würde quadratischen Rechenaufwand erfordern, weshalb nur ein Anteil von p Rechtecken (mit geringsten Flächenvergrößerungen) untersucht werden soll

[1]Beckmann, N./Kriegel, H.-P./Schneider, R./Seeger, B.: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. SIGMOD 1990, pp. 322-331

Genaueres zu R*-Bäumen (Forts.)

- *Splitten – Wahl einer Dimension:*

  betrachte alle Aufteilungen der Rechteckmenge
  (sortiert nach linken, dann rechten Grenzen)
  in eine linke und eine rechte Gruppe G_l, G_r einer Mindestgröße q;

  definiere $\text{Flächenmaß}(G_l, G_r) := \text{Fläche}(\boxed{G_l}) + \text{Fläche}(\boxed{G_r})$

  $\text{Umfangsmaß}(G_l, G_r) := \text{Umfang}(\boxed{G_l}) + \text{Umfang}(\boxed{G_r})$

  $\text{Überlappungsmaß}(G_l, G_r) := \text{Fläche}(\boxed{G_l} \cap \boxed{G_r})$

  wähle Dimension mit geringster Summe aller Umfangsmaße der Aufteilungen[2]

  entlang dieser Dimension wähle Aufteilung mit minimalem Überlappungsmaß

- *Overflowbehandlung:*

  beim ersten Overflow auf einer Ebene *forced reinsert* (erst später Splitting):

  entferne Einträge mit r größten Distanzen (zwischen den Mittelpunkten ihres Rechtecks und des überdeckenden Rechtecks),
  und passe überdeckendes Rechteck an,

  danach füge Einträge in ansteigender Reihenfolge der Distanzen wieder ein[3]

[2] weil dann eher quadratische Überdeckungen entstehen, besser wegen geringeren Durchmessers bzw. geringerer maximaler Distanz

[3] tauscht anfangs Einträge zwischen Nachbarknoten aus, verbessert also zunächst die Speicherauslastung, bevor Splits greifen; führt ebenfalls zu eher quadratischen Überdeckungen

Suchoptimierung durch R$^+$-Bäume

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

$\Rightarrow$ Effiziente Suche erfordert minimale Überdeckung und Überlappung !

- 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).

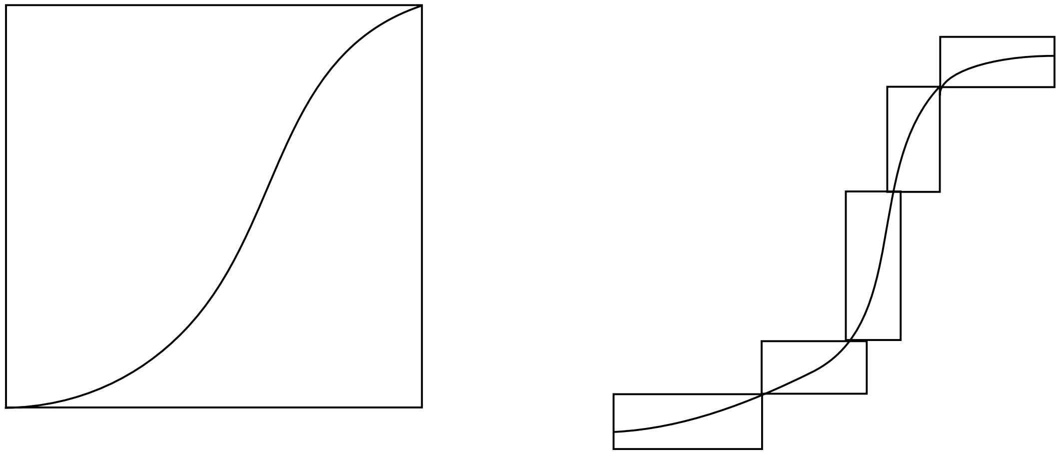
- **Idee R^+-Baum:**

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

  ➥ Vermeidung von Überlappungen bei Zwischenknoten

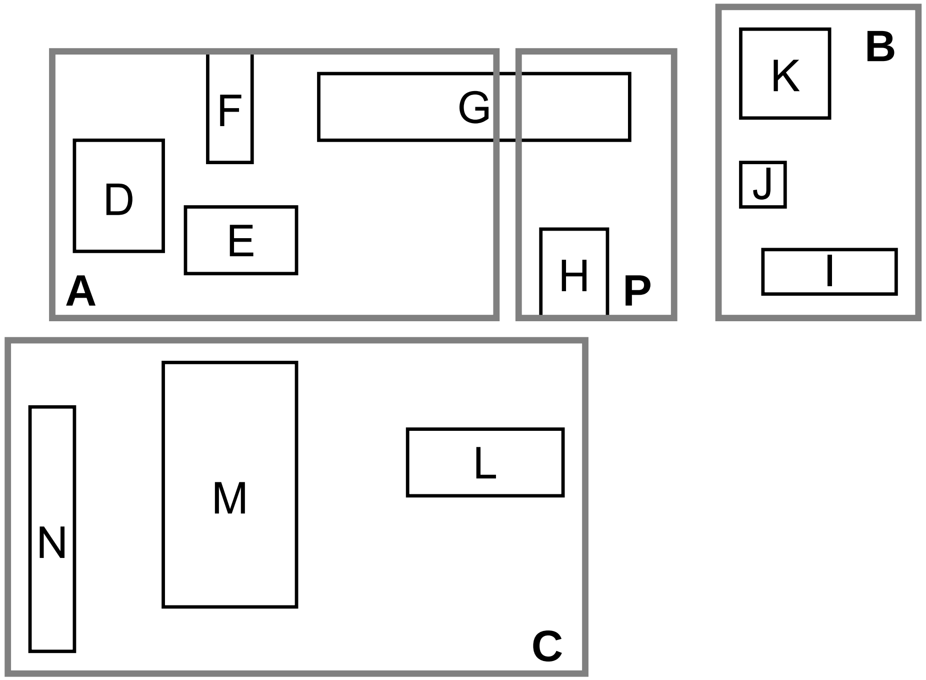
- **Konsequenz:**

  Ein Datenrechteck wird ggf. in eine Sammlung von disjunkten Teilrechtecken zerlegt und auf der Blattebene in verschiedenen Knoten gespeichert.



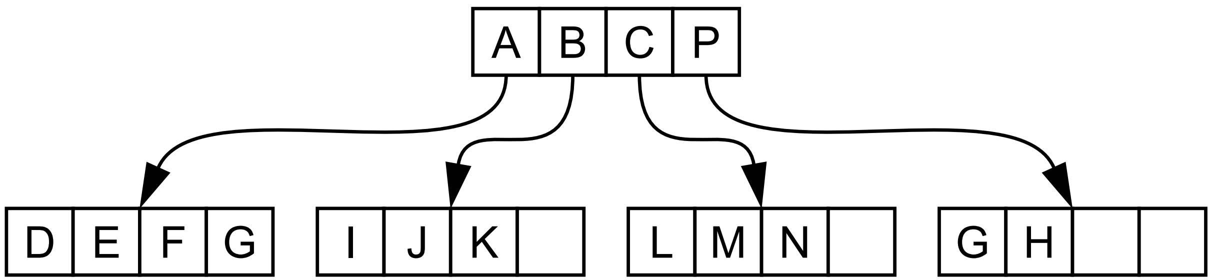
Aufteilungsmöglichkeiten eines langen Linienobjektes im R^+-Baum zum früheren Beispiel 1:

- **Aufteilung des Datenraumes**



... höhere Flexibilität durch Partitionierung von Datenrechtecken

- Zugehöriger **R^{+}-Baum**:



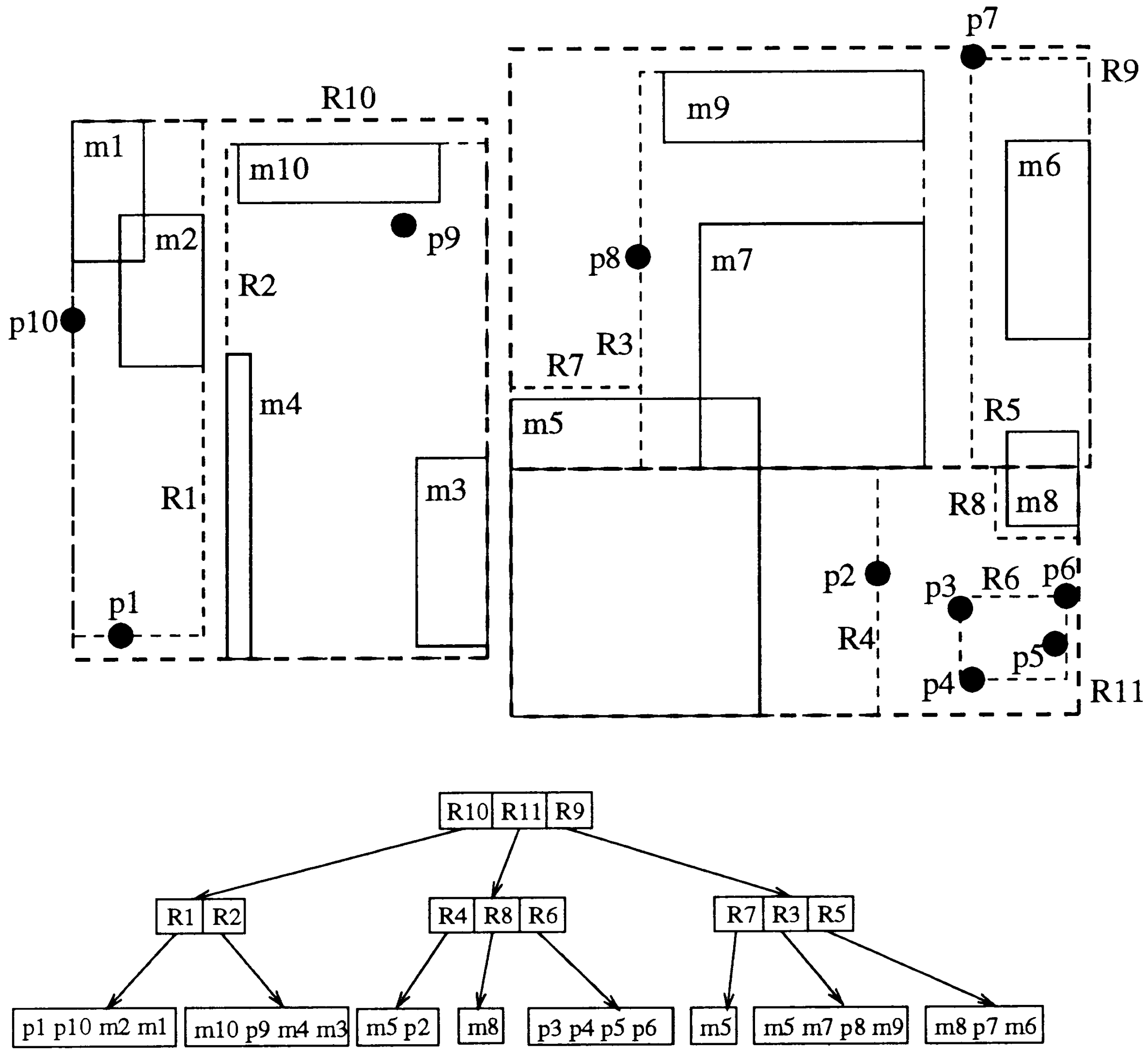


Fig. 3.29. R^+-tree

Genaueres zu R^+-Bäumen [4]

- *Vor-/Nachteile:*
  - \+ keine Überlappungen ⇒ nur ein Pfad bei exakten Anfragen bzw. weniger Pfade bei manchen Bereichsanfragen
  - – Anfragen mit mehreren Pfaden erfordern Eliminierung von Duplikaten bzw. Zusammensetzung von Objektfragmenten (je nach Abspeicherung geclipter Objekte)
  - – Einfügen algorithmisch komplizierter (s.u.);
    insbesondere kann es mehrere Pfade erfordern
  - \+ Regionen bilden vollständige Cluster der in ihnen enthaltenen Objekte bzw. Objektfragmente; Nutzdaten, auf die aus mehreren Regionen verwiesen werden kann, sind aber evtl. nicht mehr regionenbezogen clusterbar

- *Einfügen eines Eintrags E in einen inneren Knoten:*

  falls ein oder mehrere Regionen E insgesamt ganz überdecken

  (*) füge E in alle zugehörigen Teilbäume ein,
  jeweils geeignet geclipt, aber mind. mit Verweis auf Gesamtobjekt

  sonst erweitere Regionen zuerst (s.u.) und dann (*)

[4] Sellis,T. et. al: The R^+-Tree. a Dynamic Index for Multi-Dimensional Objects, in: Proc. 14Th VLDB Conf. 1987, pp. 507-518

Genaueres zu R$^+$-Bäumen (Forts.)

- *Erweitern von Regionen bzgl. eines Eintrags E:*

  erweitere möglichst nur eine Region wie beim R-Baum
  auf den noch nicht überdeckten Teil von *E*;

  falls *E* nicht durch Erweiterung vorhandener Regionen insgesamt ganz überdeckt werden kann ("Deadlock"), müssen einige Regionen vorher gesplittet werden (mit einem *forced split*)

- *Overflowbehandlung beim Einfügen:*

  Splitten typischerweise durch Einziehen einer Split-Linie
  wie beim k-d-B-Baum und Weitergabe nach oben;

  kann Clipping von Regionen und *forced splits* nach unten erfordern.

- *Splitkriterien:*

  - möglichst gleichmäßige Aufteilung
  - minimale Anzahl von Regionensplits
  - minimale Überdeckung von ungenutzten Flächen durch die Regionen

Weitere Varianten:

- $R^{\cdots}$-Bäume eignen sich auch zur Abspeicherung von Punktmengen oder gemischten Mengen, da Punkte spezielle (flächenleere) Rechtecke sind. Es ergibt sich eine bessere (gezieltere) Flächenüberdeckung als in k-d-B-Bäumen.

- $R^{\cdots}$-Bäume können auch zur Abspeicherung von Mengen eindimensionaler (z.B. temporaler) Intervalle verwendet werden. Am besten lassen sich die obigen Begriffe ("Flächenvergrösserung" usw.) anwenden, wenn eine triviale zweite Dimension in Einheitsgröße angenommen wird[5].

- Statt umgebenden Rechtecken sind auch umgebende Kreise (→ *sphere trees*) oder umgebende konvexe Polygone, evtl. konvexe Hüllen (→ *cell trees*) nutzbar. Erstere eignen sich gut für Nachbarschaftsanfragen.

---

[5] vgl. Teil I dieser Vorlesung, dortige Seite 6.4