Kapitel 8: Zugriffsstrukturen

8.1 Indizes für Exact−Match−, Präfix−Match− und Bereichsanfragen

Anlegen von Indizes mit
CREATE [UNIQUE] INDEX index−name ON table (column {”,” column}) ...

Ein Index über den Attributen A1, A2, ..., Ak erlaubt die effiziente Bestimmung aller Treffer für Exact−Match− und Präfix−Match−Prädikate der Form
A1=wert1 AND A2=wert2 AND ... Aj=wertj (für ein beliebiges j mit $1 \leq j \leq k$)
oder Bereichsprädikate der Form
u1<=A1 AND u2<=A2 AND ... uj<=Aj AND
A1<=o1 AND A2 <=o2 AND ... Aj<=oj
(für ein beliebiges j mit $1 \leq j \leq k$ und beliebige Konstanten u1, ..., uj, o1, ..., oj)
sowie einen Relationen−Scan, bei dem die Tupel sortiert nach
A1 || A2 || ... || Aj (für ein beliebiges j mit $1 \leq j \leq k$)
zurückgeliefert werden.

Implementierung von Indizes durch Mehrwegbäume

B−Baum:

balancierter Mehrwegbaum für effizientes Suchen, Einfügen und Löschen auf Externspeicher.
Ein Knoten des Baums entspricht einer Seite und beinhaltet k Datensätze sowie − bei Nichtblattknoten − k+1 Sohnzeiger.
Worst−Case−Komplexität (Anzahl der Seitenzugriffe) für Suchen, Einfügen und Löschen in einem B−Baum mit n Datensätzen: $O(\log_k n)$.
Mittlere Speicherplatzauslastung bei zufälligem Einfügen: $\ln 2 \approx 0.69$.

Formale Definition:
Ein Mehrwegbaum heißt B−Baum der Ordnung m ($m \geq 1$), wenn gilt:

1) Jeder Knoten außer der Wurzel enthält mindestens m Schlüssel.
   und höchstens 2m Schlüssel.
2) Ein Nichtblattknoten mit k Schlüsseln $x_1, ..., x_k$ hat genau k+1 Söhne $t_1, ..., t_{k+1}$, so daß
   für alle Schlüssel s im Teilbaum t_i ($2 \leq i \leq k$) gilt $x_{i-1} < s < x_i$ und
   für alle Schlüssel s im Teilbaum t_1 gilt $s < x_1$ und
   für alle Schlüssel s im Teilbaum t_{k+1} gilt $x_k < s$.
3) Alle Blätter haben dasselbe Niveau.

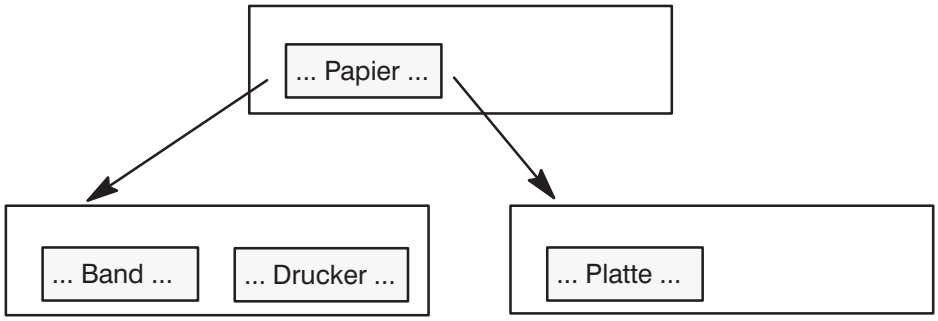
Pseudocode für das Suchen nach einem Schlüssel s in einem B−Baum mit Wurzel t:

t habe k Schlüssel x_1, ..., x_k und k+1 Söhne t_1, ..., t_{k+1}.
Bestimme den kleinsten Schlüssel x_i, so daß $s \leq x_i$.
Falls $s = x_i$ (für $i \leq k$)
dann Schlüssel gefunden
sonst
  Falls t kein Blatt ist
  dann
    Falls $s < x_i$ (für $i \leq k$)
    dann Suche s im Teilbaum t_i
    sonst Suche s im Teilbaum t_{k+1}
  sonst Schlüssel nicht vorhanden

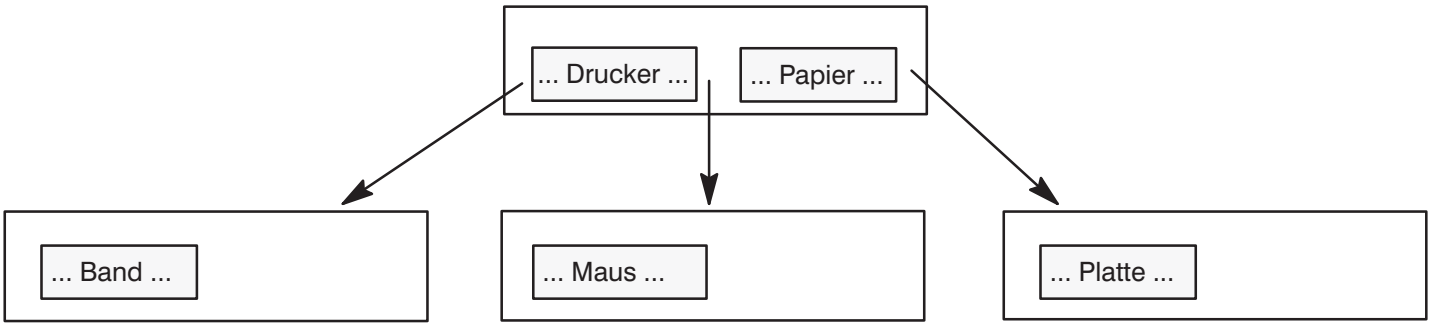
Pseudocode für das Einfügen eines Schlüssels e in einen B−Baum:

Suche nach dem einzufügenden Schlüssel e
Falls der Schlüssel noch nicht vorhanden ist
dann
  Sei t der Blattknoten, bei dem die Suche erfolglos geendet hat
  Wiederhole
    Falls t weniger als 2m Schlüssel enthält
    dann Füge e ein
    sonst
      "Knotenspaltung" ("Node Split"):
      − Sortiere den Schlüssel e in die 2m Schlüssel von t ein und
        bestimme den mittleren Schlüssel s dieser 2m+1 Schlüssel (den "Median")
      − Spalte t auf, d.h. erzeuge einen Bruderknoten t' und speichere
        die kleinsten m Schlüssel von t mit den entsprechenden Sohnzeigern in t sowie
        die größten m Schlüssel von t mit den entsprechenden Sohnzeigern in t'
      Falls t die Wurzel ist
      dann
        Erzeuge einen neuen Wurzelknoten r mit dem Schlüssel s und Zeigern auf t und t'
      sonst
        Betrachte den Vater von t als neues t und s als neues e
  bis keine Knotenspaltung mehr erfolgt ist

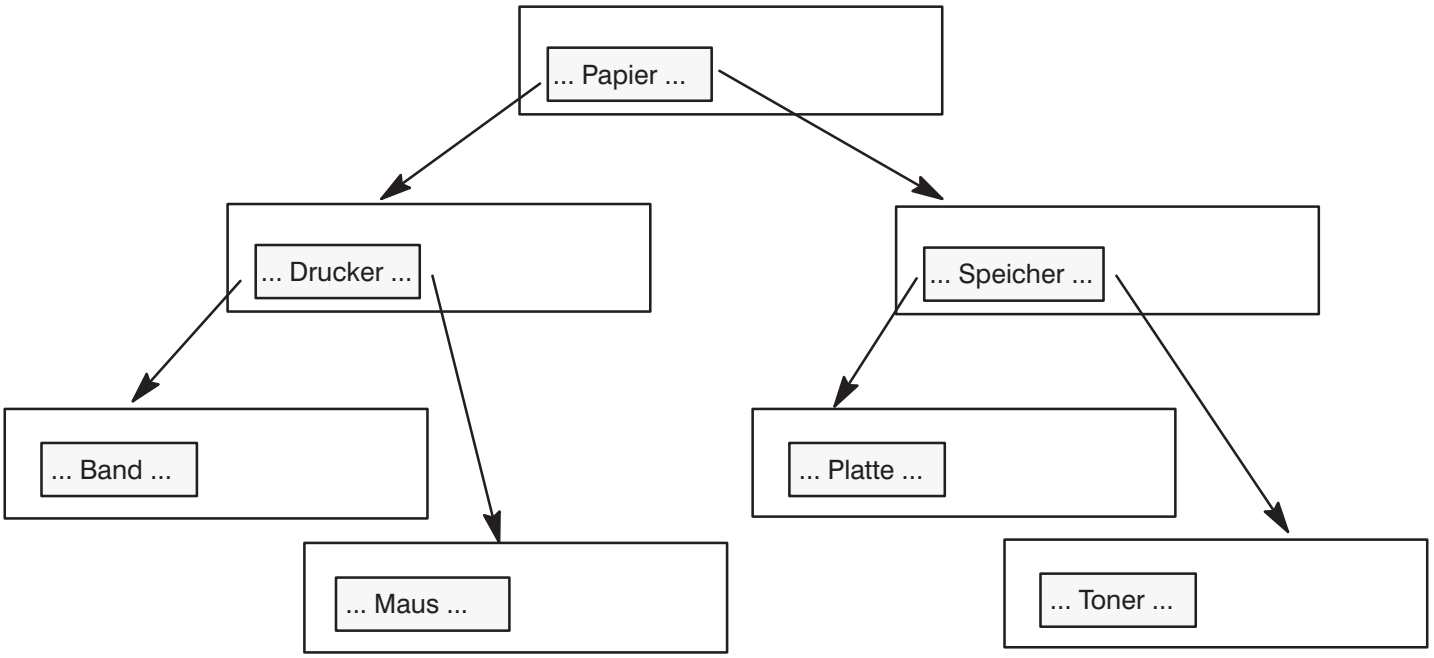
Beispiel: B−Baum−Index für Produkte.Bez



Einfügen von ”Maus”:
Wachstum des Baums durch Teilung von Knoten (”Split”):



Einfügen von ”Toner” und ”Speicher”:
Wachstum des Baums durch wiederholte Teilung von Knoten (incl. der Wurzel)
und Erzeugung einer neuer Wurzel:



B*−Baum (engl.: B^+−tree):

B−Baum−Variante, bei der Datensätze nur in den Blättern gespeichert werden und Nichtblattknoten nur "Wegweiser" (Suchschlüssel plus Sohnzeiger) enthalten.
DBS verwenden B*−Bäume wegen der folgenden Vorteile gegenüber B−Bäumen:

- Da in den Nichtblattknoten nur Wegweiser gespeichert werden, erreichen diese einen höheren Verzweigungsgrad, so daß die Baumhöhe potentiell geringer ist.
  Der Verzweigungsgrad kann durch geschickte Komprimierung der Wegweiser sogar noch weiter erhöht werden (wie z.B. bei Präfix−B*−Bäumen).
- Ein Scan in Sortierreihenfolge erfordert nur eine Traversierung der (verketteten) Blätter (gegenüber einer Traversierung aller Knoten beim B−Baum).

Der Verzweigungsgrad (engl.: fan−out) eines B*−Baums liegt typischerweise zwischen 50 und 400.

Formale Definition:
Ein Mehrwegbaum heißt B*−Baum der Ordnung (m, m*) ($m \geq 1$, $m^* \geq 1$), wenn gilt:

1) Jeder Nichtblattknoten außer der Wurzel enthält mindestens m und höchstens 2m Schlüssel.
2) Ein Nichtblattknoten mit k Schlüsseln $x_1, ..., x_k$ hat genau k+1 Söhne $t_1, ..., t_{k+1}$, so daß
   für alle Schlüssel s im Teilbaum t_i ($2 \leq i \leq k$) gilt $x_{i-1} < s \leq x_i$ und
   für alle Schlüssel s im Teilbaum t_1 gilt $s \leq x_1$ und
   für alle Schlüssel s im Teilbaum t_{k+1} gilt $x_k < s$.
3) Alle Blätter haben dasselbe Niveau.
4) Jedes Blatt enthält mindestens m* und höchstens 2m* Schlüssel.

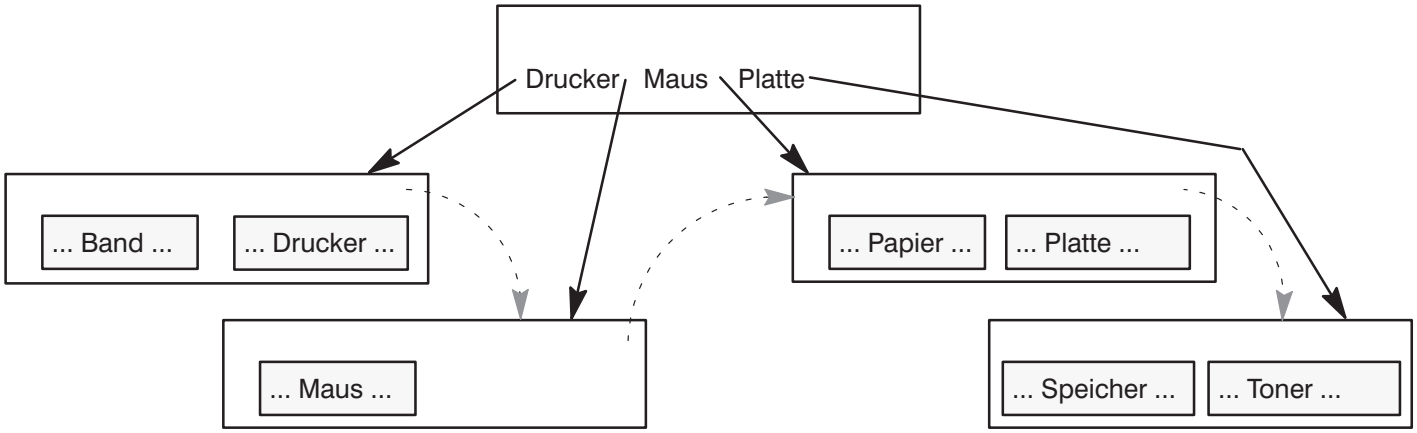
Pseudocode für das Suchen nach einem Schlüssel s in einem B−Baum mit Wurzel t:*

t habe k Schlüssel x_1, ..., x_k und k+1 Söhne t_1, ..., t_{k+1}.
Bestimme den kleinsten Schlüssel x_i, so daß $s \leq x_i$.
Falls $s = x_i$ (für $i \leq k$) und t ist ein Blatt
dann Schlüssel gefunden
sonst
  Falls t kein Blatt ist
  dann
    Falls $s \leq x_i$ (für $i \leq k$)
    dann Suche s im Teilbaum t_i
    sonst Suche s im Teilbaum t_{k+1}
  sonst Schlüssel nicht vorhanden

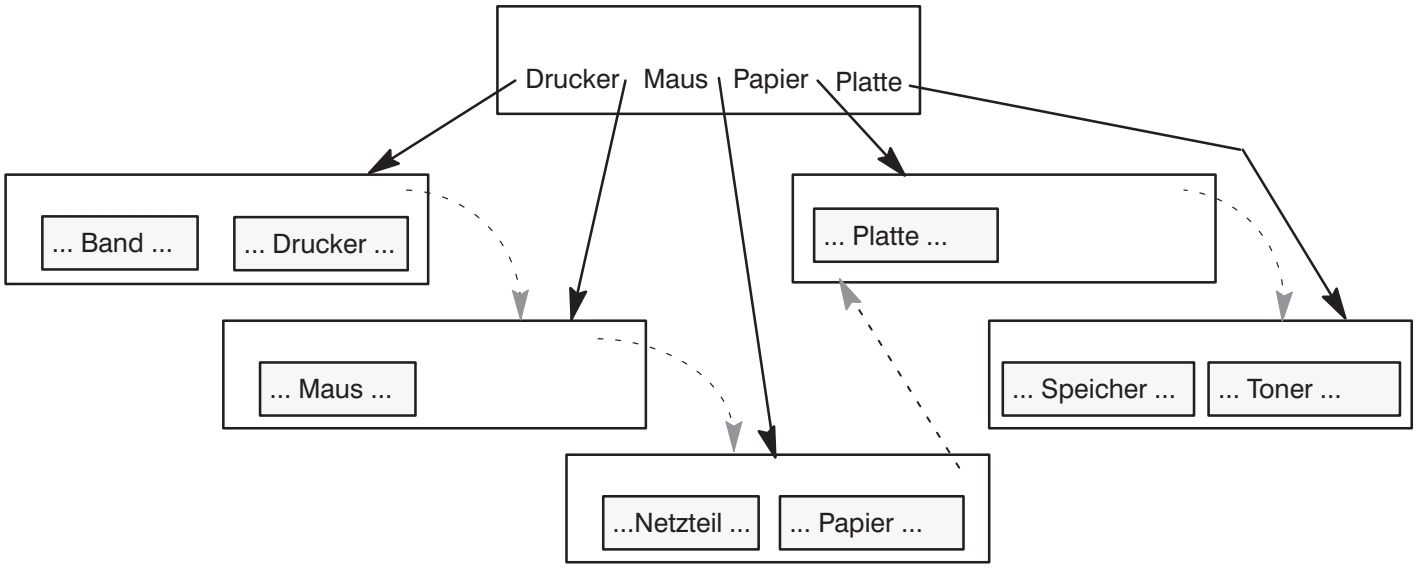
Pseudocode für das Einfügen eines Schlüssels e in einen B−Baum:*

Suche nach dem einzufügenden Schlüssel e
Falls der Schlüssel noch nicht vorhanden ist
dann
  Sei t der Blattknoten, bei dem die Suche erfolglos geendet hat
  Wiederhole
    Falls t ein Blattknoten ist und weniger als 2m* Schlüssel enthält oder
        t ein Nichtblattknoten ist und weniger als 2m Schlüssel enthält
    dann Füge e ein
    sonst
      ”Knotenspaltung” (”Node Split”):
        − Sortiere den Schlüssel e in die 2m* bzw. 2m Schlüssel von t ein und bestimme den Median s dieser 2m*+1 bzw. 2m+1 Schlüssel
        − Erzeuge einen Bruderknoten t’
        − **Falls** t ein Blattknoten ist
          dann
            Speichere die kleinsten m*+1 Schlüssel von t mit den entsprechenden Sohnzeigern in t sowie die größten m* Schlüssel von t mit den entsprechenden Sohnzeigern in t’
          sonst
            Speichere die kleinsten m Schlüssel von t mit den entsprechenden Sohnzeigern in t sowie die größten m Schlüssel von t mit den entsprechenden Sohnzeigern in t’
      Falls t die Wurzel ist
      dann
        Erzeuge einen neuen Wurzelknoten r mit dem Schlüssel s und Zeigern auf t und t’
      sonst
        Betrachte den Vater von t als neues t und s als neues e
  bis keine Knotenspaltung mehr erfolgt ist

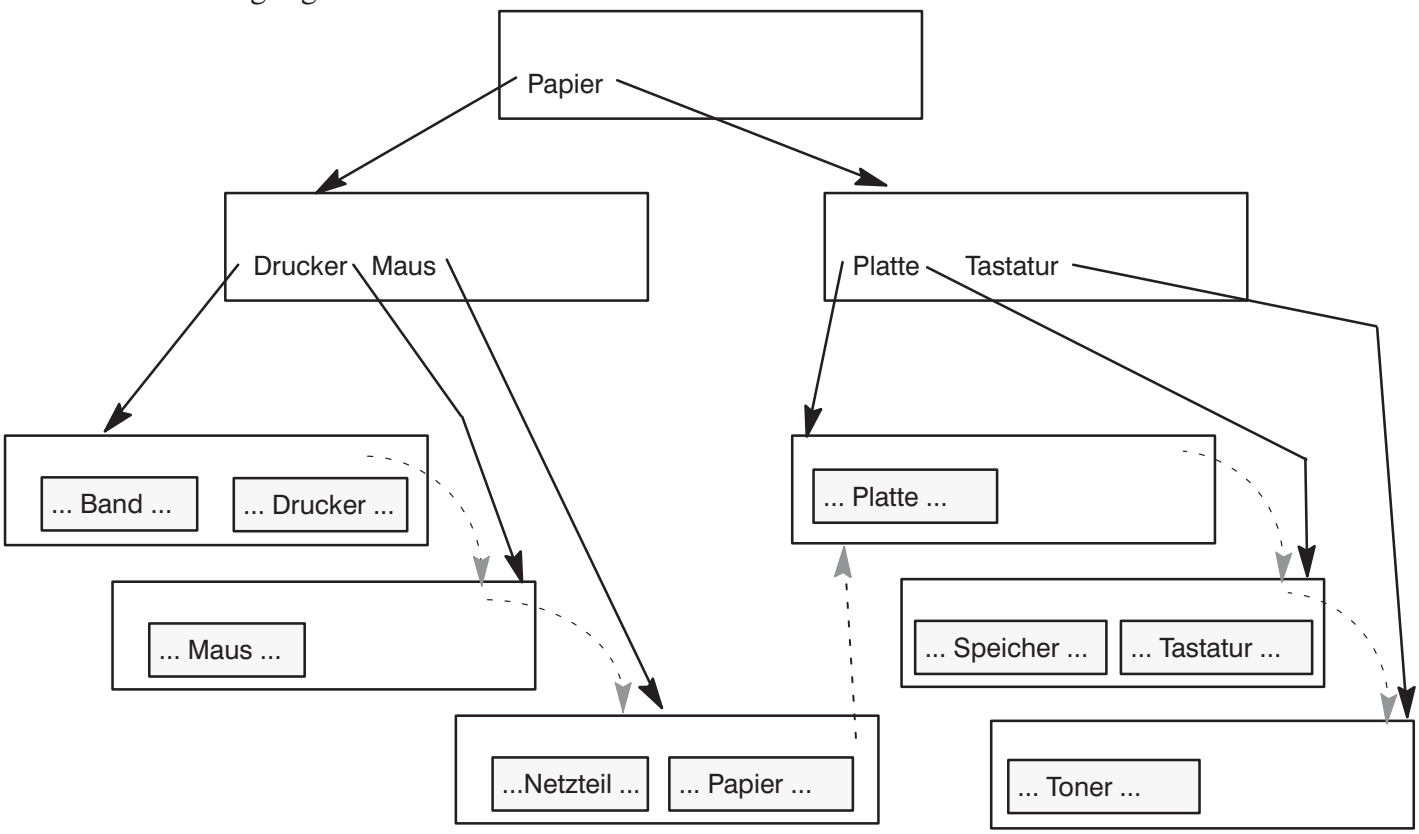
Beispiel: B*−Baum−Index für Produkte.Bez



Einfügen von ”Netzteil”:
Wachstum des Baums durch Teilung von Knoten (”Split”)



Einfügen von ”Tastatur”:
Wachstum des Baums durch wiederholte Teilung von Knoten (incl. der Wurzel)
und Erzeugung einer neuer Wurzel

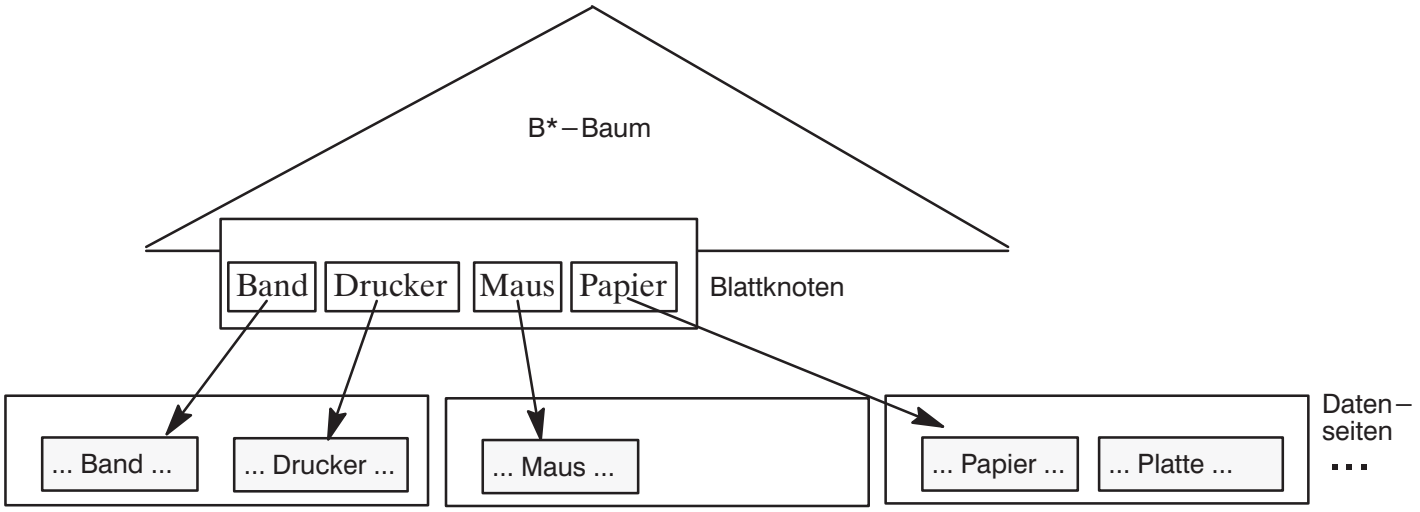


Varianten von (B*−Baum−)Indizes

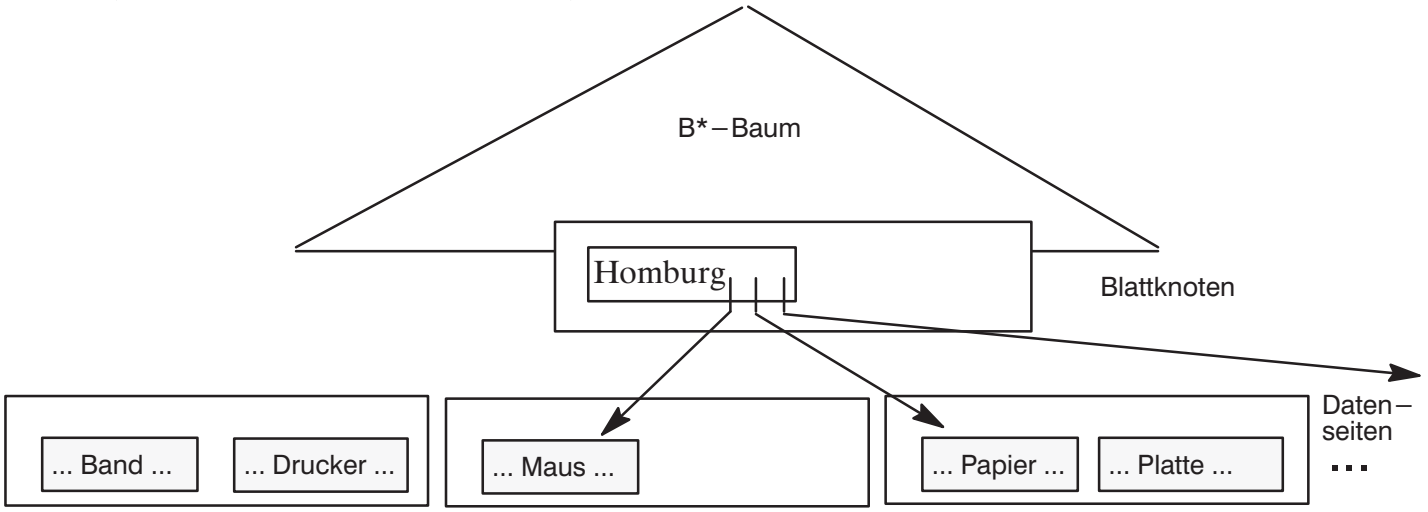
Unmittelbarer Index (direkter Index; engl.: sparse index):
die Blätter enthalten direkt die Datensätze.

Mittelbarer Index (indirekter Index; engl.: dense index):
die Blätter enthalten Suchschlüssel und Zeiger (TIDs) auf die Datensätze.

Beispiel für einen mittelbaren Index über Produkte.Bez:



Beispiel für einen mittelbaren Index über Produkte.Lagerort
(mit Duplikaten für Suchschlüssel):



Bei Suchschlüsseln mit Duplikaten wird im Blattknoten beim Suchschlüssel eine
TID−Liste gespeichert.

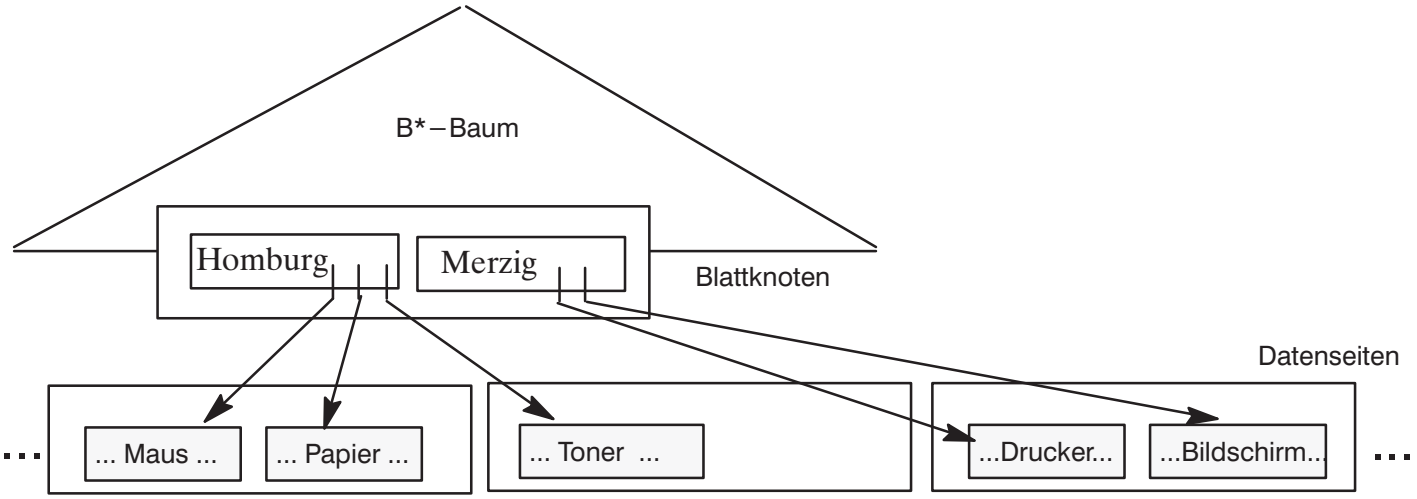
Bemerkung:
Für eine Relation kann es maximal einen unmittelbaren Index,
aber beliebig viele mittelbare Indizes geben.

Clustered Index:

Datensätze mit demselben Suchschlüssel liegen im selben Block oder in physisch benachbarten Blöcken.

Datensätze sind − nach dem Laden − nach dem Suchschlüssel sortiert gespeichert.

Beispiel für einen mittelbaren Index über Produkte.Lagerort:



Anlegen eines Clustered Index in Oracle:

CREATE CLUSTER cluster−name “(” key−column−list “)” [SIZE num−bytes] ...

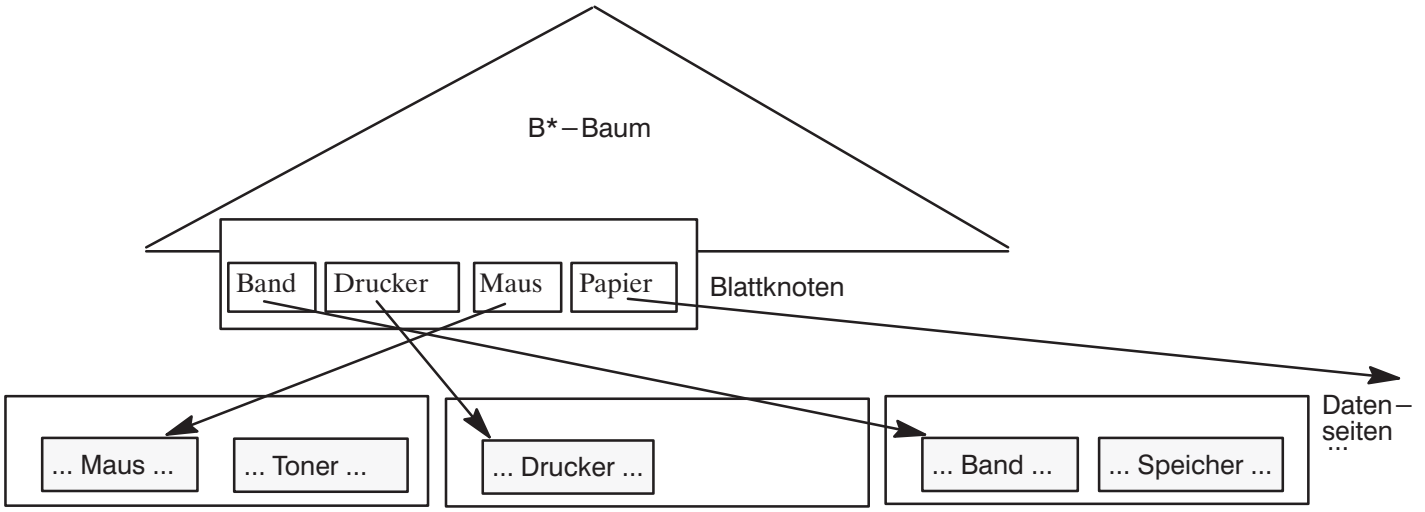
CREATE INDEX index−name ON CLUSTER cluster−name

CREATE TABLE table−name “(” column−spec “)” CLUSTER cluster−name

Unclustered Index:

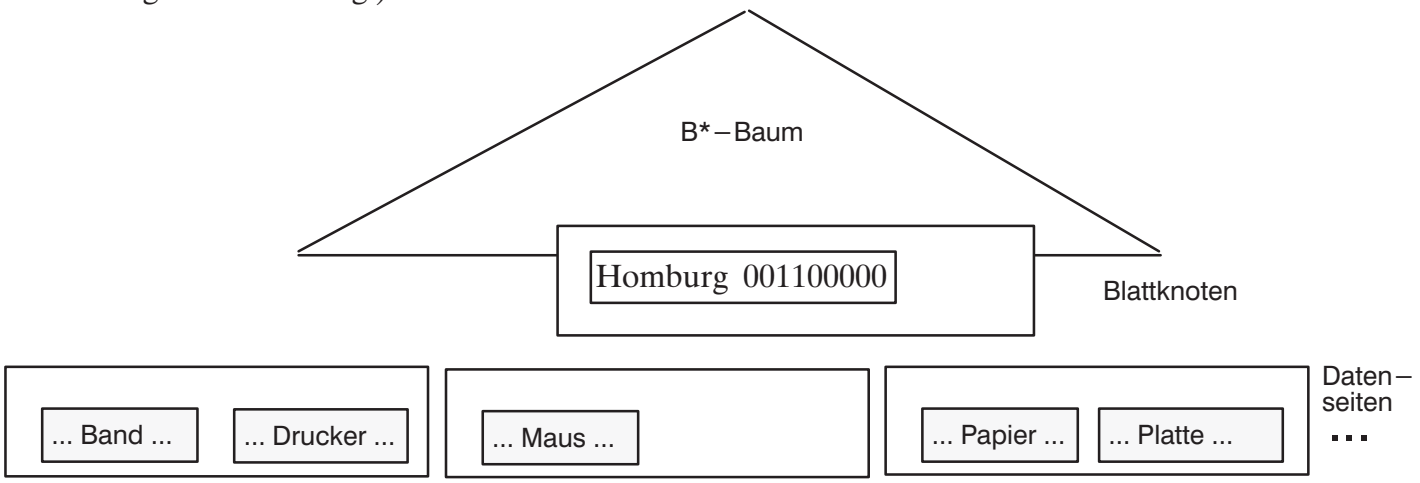
Es gibt keinen Zusammenhang zwischen dem Suchschlüssel des Index und der Speicherungsreihenfolge der Datensätze.

Beispiel:



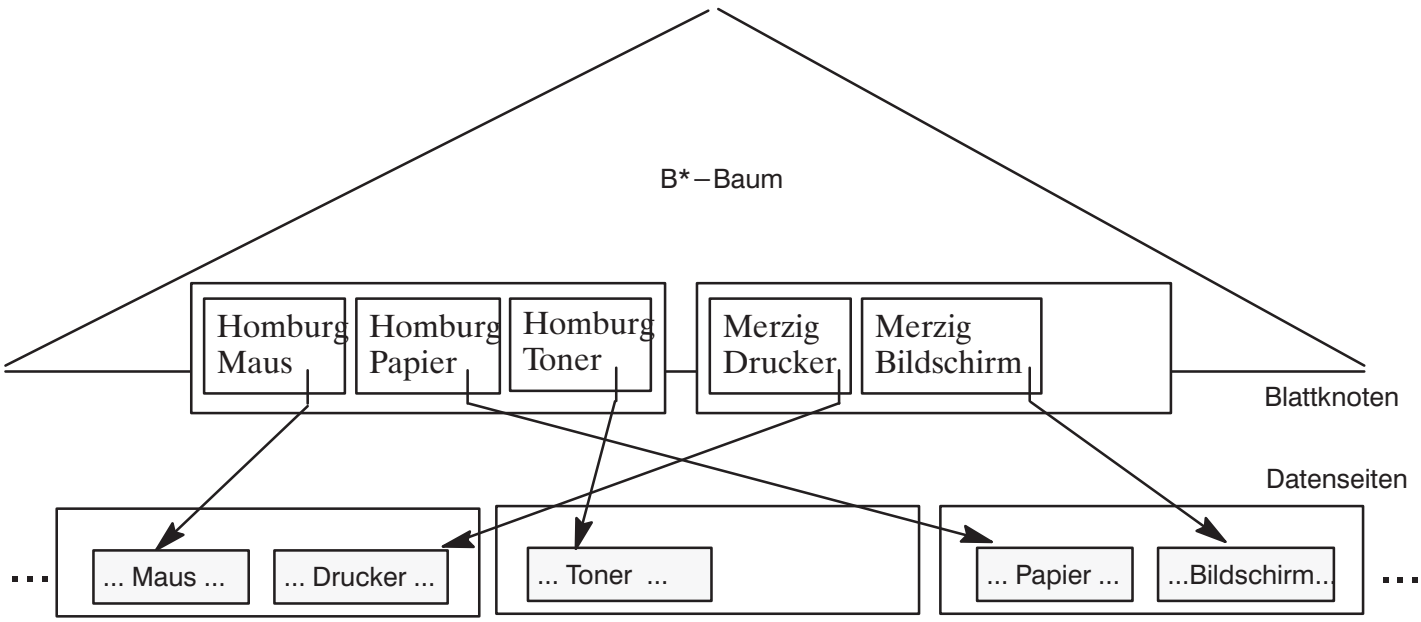
Bitlistentechnik:
Bei mittelbaren Indizes mit vielen Duplikaten kann man in den Blättern an Stelle der TID−Listen für jeden Schlüssel eine Bitliste speichern, bei der eine 1 an der Bitposition i ausdrückt, daß der Datensatz mit dem Database−Key (oder dem TID) i den entsprechenden Schlüsselwert für das indizierte Attribut enthält. Die Anzahl der überhaupt vergebbaren DBKs (oder TIDs) legt die Länge der Bitlisten (in Bits) fest.

Beispiel (Annahme: Es gebe maximal 10 DBKs. Die Sätze mit DBK 3 und 4 gehören zum Lagerort Homburg.):



Index über mehrere Attribute:
Der Suchschlüssel wird durch die Konkatenation mehrerer Attribute gebildet.

Beispiel für einen mittelbaren Index über Produkte.(Lagerort || Bez):



Speicher− und Zugriffskosten eines mittelbaren B*−Baum−Index

Beispiel: Index über Bestellungen.KNr

Annahmen:

Seitengröße: 4 KBytes (mit 96 Bytes Header)
Füllgrad einer Seite: 70%
Anzahl der Tupel in Bestellungen: 10 000 000
Anzahl verschiedener KNr−Werte in Bestellungen: 100 000
Für jeden KNr−Wert gibt es gleichviele Tupel in Bestellungen.
Länge eines KNr−Werts: 24 Bytes
Länge eines TIDs: 6 Bytes
Länge eines Sohnzeigers im Baum: 6 Bytes

Folgerungen:

Länge einer TID−Liste: 24 + 100*6 Bytes = 624 Bytes
Anzahl der TID−Listen in einer Indexblattseite: $\lfloor 0.70*(4096–96) / 624 \rfloor = 4$
Anzahl der Indexblattseiten: $\lceil 100000 / 4 \rceil = 25000$
Länge eines Wegweisers in einem Nichtblattknoten: 24 + 6 Bytes = 30 Bytes
Anzahl der Wegweiser in einem Nichtblattknoten (Verzweigungsgrad):
$\lfloor 0.70*(4096–96) / 30 \rfloor = 93$
Höhe des Index: $\lceil \log_{93} 25000 = \rceil + 1 = 4$ (3 Niveaus für Nichtblattknoten und das Blattniveau)
Der Index besteht also aus 25000 Blättern (Niveau 4),
$\lceil 25000 / 93 \rceil = 269$ Nichtblattknoten auf Niveau 3,
$\lceil 269 / 93 \rceil = 3$ Nichtblattknoten auf Niveau 2 sowie
1 Wurzelknoten (Niveau 1)
Der Index belegt 25270 * 4 KBytes $\approx$ 100 MBytes.
Eine Suche nach allen Bestellungen für eine KNr erfordert 4 + 100 = 104 Seitenzugriffe.

8.2 Hash−Indizes für Exact−Match−Queries

Abbildung von Schlüsseln auf eine Menge von N *Buckets* mittels einer *Hashfunktion*, so daß auf jeden Bucket möglichst gleichviele Schlüssel abgebildet werden.
Beispiel einer simplen Hashfunktion für numerische Schlüssel x:
h(x) = (ax+b) mod N mit Konstanten a, b.
Beispiel einer simplen Hashfunktion für Zeichenketten $z=z_1...z_n$:

$$h(z) = \left(\sum_{i=1}^{n} a_i * Ordinalzahl(z_i) \right) \bmod N$$ mit Konstanten $a_1, ..., a_n$.

Gute Hashfunktionen sind nicht nachbarschaftserhaltend, sondern haben vielmehr die Eigenschaft, Schlüssel quasi−zufällig auf Buckets abzubilden (engl.: ”randomization”). Daraus folgt, daß Hash−Indizes nur Exact−Match−Anfragen und weder Präfix−Match−Anfragen noch Bereichsanfragen und auch keine sortierten Scans unterstützen.

Jedem Bucket ist ein Primärblock zugeordnet, in dem eine feste Maximalzahl von Sätzen (bei unmittelbarem Index) oder Schlüsseln und TID−Listen (bei mittelbarem Index) gespeichert werden kann. Falls auf einen Bucket mehr Schlüssel abgebildet werden, als dessen Primärblock aufnehmen kann, wird ein Überlaufblock alloziert und mit dem Primärblock verkettet. Bei Bedarf können mehrere Überlaufblöcke für einen Bucket alloziert werden, die zusammen mit dem Primärblock eine verkettete Liste bilden.

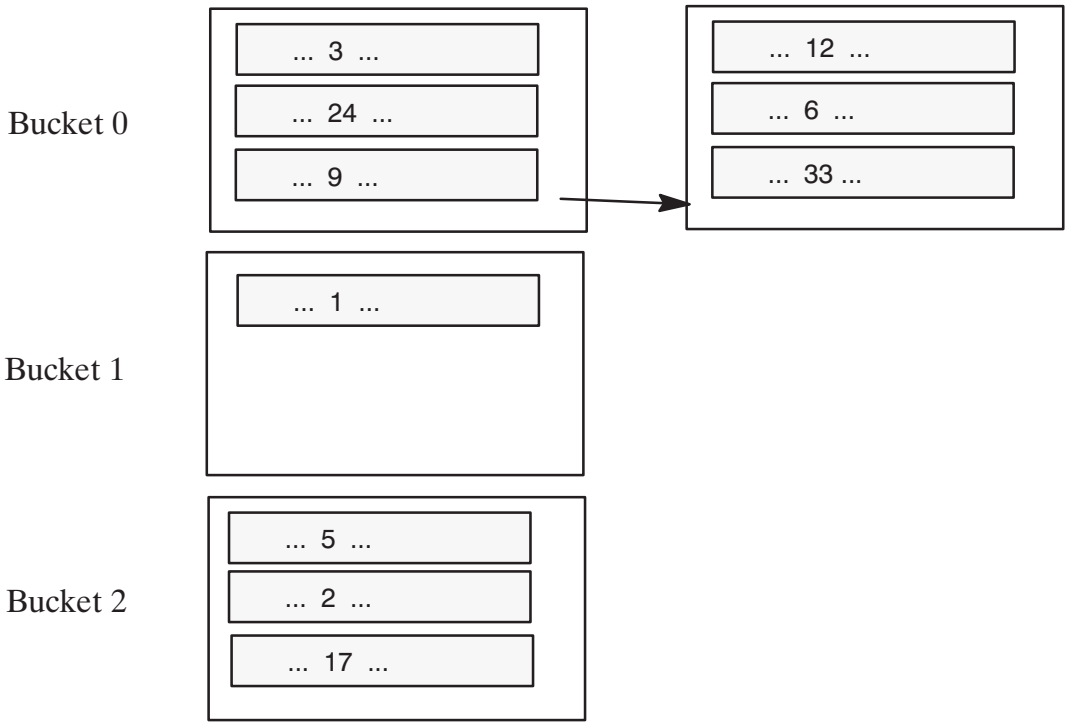
Ein Hash−Index über den Attributen A1, A2, ..., Ak erlaubt die effiziente Bestimmung aller Treffer für Exact−Match−Prädikate der Form
A1=wert1 AND A2=wert2 AND ... Ak=wertk.

Anlegen von Hash−Indizes in Oracle:

```
CREATE CLUSTER    cluster−name “(” key−column “)” ... SIZE m  ...
                  HASH IS key−column [ HASHKEYS n ]
CREATE TABLE table−name “(” column−spec “)” CLUSTER cluster−name
```

Vorteile von Hash−Indizes gegenüber B*−Bäumen:
Exact−Match−Anfragen erfordern in der Regel nur 1 Plattenzugriff
Nachteile:
keine dynamische Reorganisation, so daß die Effizienz linear entarten kann

Beispiel: Hash−Index für Bestellungen.KNr mit 3 Buckets und der Hashfunktion
h(k) = k mod 3



Analyse der Anzahl von Buckets mit Überlaufblöcken:

Gegeben: N Buckets
n Schlüssel
Blockkapazität: b Schlüssel pro Block

$$Prob[Bucket\ i\ hat\ k\ Schluessel] = \binom{n}{k} * \left(\frac{1}{N}\right)^{k} * \left(\frac{N-1}{N}\right)^{n-k}$$

$$Prob[Bucket\ i\ hat\ mehr\ als\ b\ Schluessel] = 1 - \sum_{k=0}^{b} \binom{n}{k} * \left(\frac{1}{N}\right)^{k} * \left(\frac{N-1}{N}\right)^{n-k}$$

$$E[Anzahl\ der\ Buckets\ mit\ Ueberlaufbloecken] = N * Prob[Bucket\ i\ hat\ mehr\ als\ b\ Schluessel]$$

Dynamische Hashverfahren

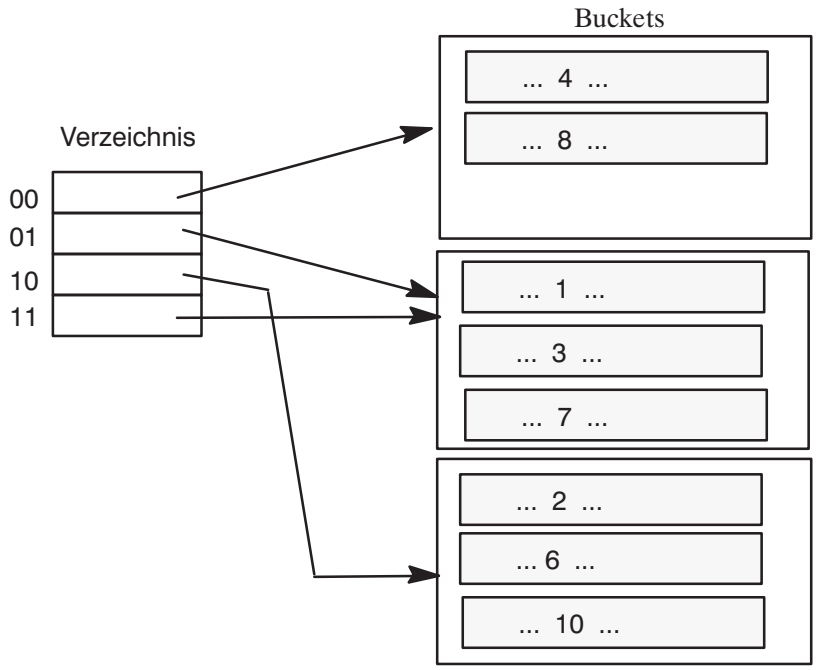
Ziel:
Die Anzahl der Buckets (und damit der Bildbereich der Hashfunktion) kann dynamisch erweitert werden, so daß bei einer Erweiterung nur ein kleiner Teil der Schlüssel umgespeichert werden muß.

Beispiel eines dynamischen Hashverfahrens: Erweiterbares Hashing (Extendible Hashing)

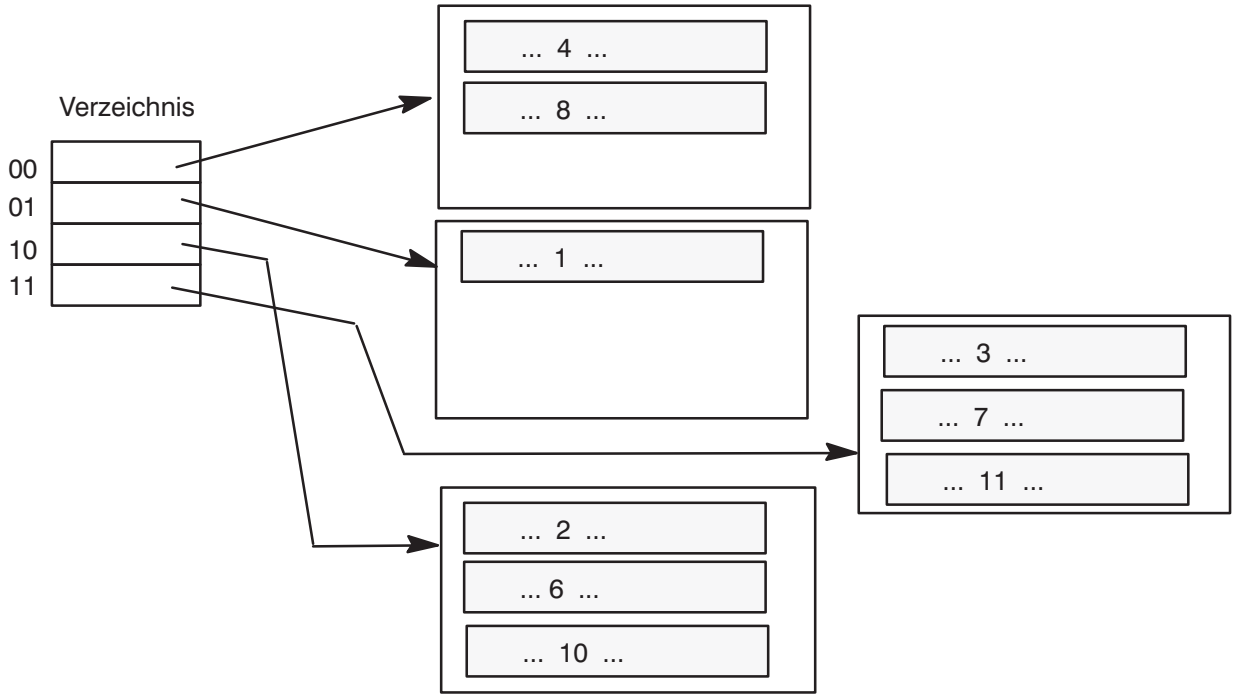
Ein *erweiterbarer Hash−Index der (globalen) Tiefe d* besteht aus einem Verzeichnis ("Directory") mit 2^d Einträgen und $N \leq 2^d$ Buckets. Die Einträge des Verzeichnisses verweisen auf jeweils einen Bucket, wobei auf denselben Bucket 2^{d-i} verschiedene Einträge verweisen können mit $0 \leq i \leq d$. Die Zahl i wird als *lokale Tiefe* eines Buckets bezeichnet.
Ein Schlüssel x wird in demjenigen Bucket gespeichert, auf den der Verzeichniseintrag mit der binären Nummer $h_{d-1} ... h_0$ verweist, wobei $h_{d-1} ... h_0$ die d hintersten Bits des Wertes h(x) der Hashfunktion h sind.

Wenn ein Bucket b überläuft bzw. dessen Überlaufkette zu lang wird, wird eine Teilung (*Split)* dieses Buckets durchgeführt.
Fall 1:
Wenn b eine lokale Tiefe $i < d$ hat, wird ein neuer Bucket b' alloziert und in den Verzeichniseinträgen e eingetragen, für die gilt: $e \bmod 2^{i+1} = (b+2^i) \bmod 2^d$.
Die lokale Tiefe des Buckets b und des neuen Buckets b' wird jeweils auf i+1 gesetzt.
Ein bislang b zugeordneter Schlüssel x wird nach b' umgespeichert, wenn das (i+1)−letzte Bit von h(x) den Wert 1 hat; ansonsten bleibt der Schlüssel in b.
Fall 2:
Wenn die lokale Tiefe von b gleich der globalen Tiefe des Verzeichnisses ist, wird zunächst das Verzeichnis durch Kopieren verdoppelt und die globale Tiefe um 1 erhöht.
Danach wird wie in Fall 1 verfahren.

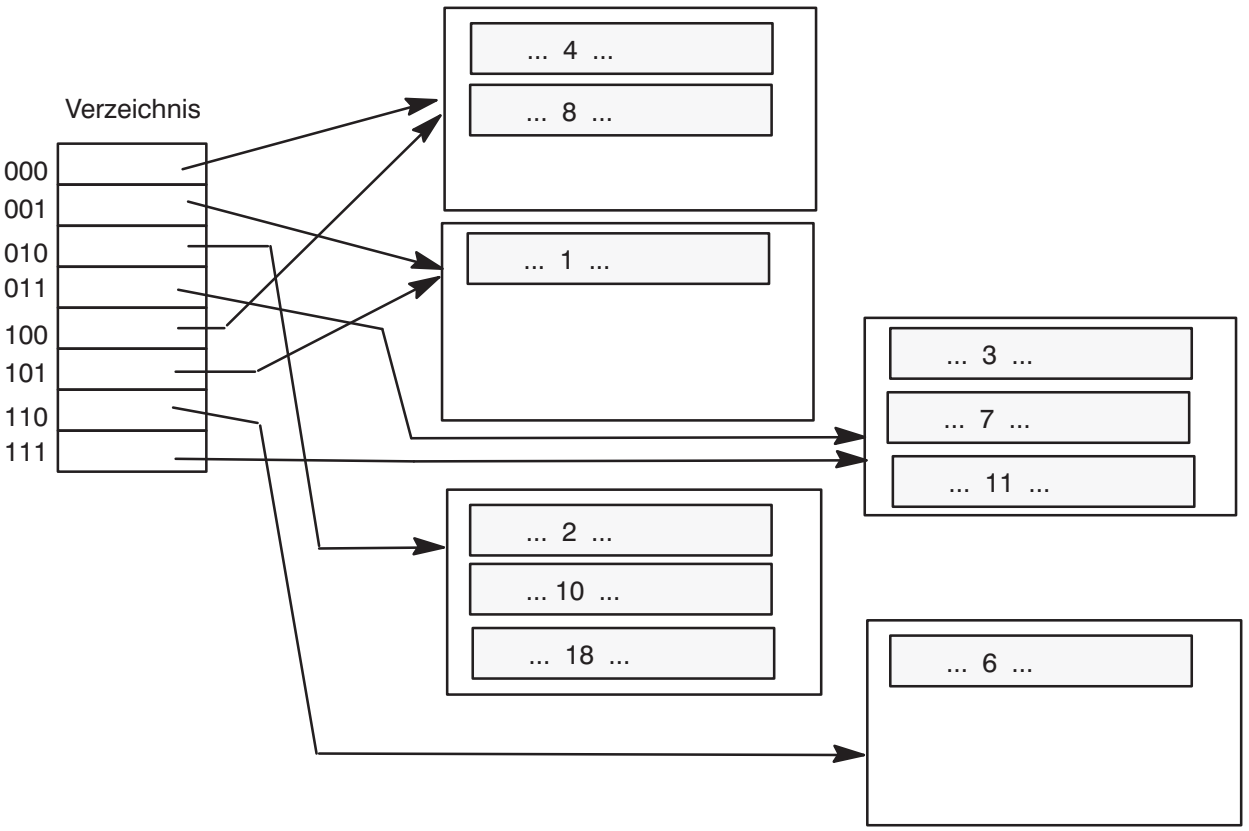
Beispiel: Erweiterbarer Hash−Index für Bestellungen.KNr mit h(k)=k, d=2, N=3:



Einfügen einer neuen Bestellung mit KNr 11:



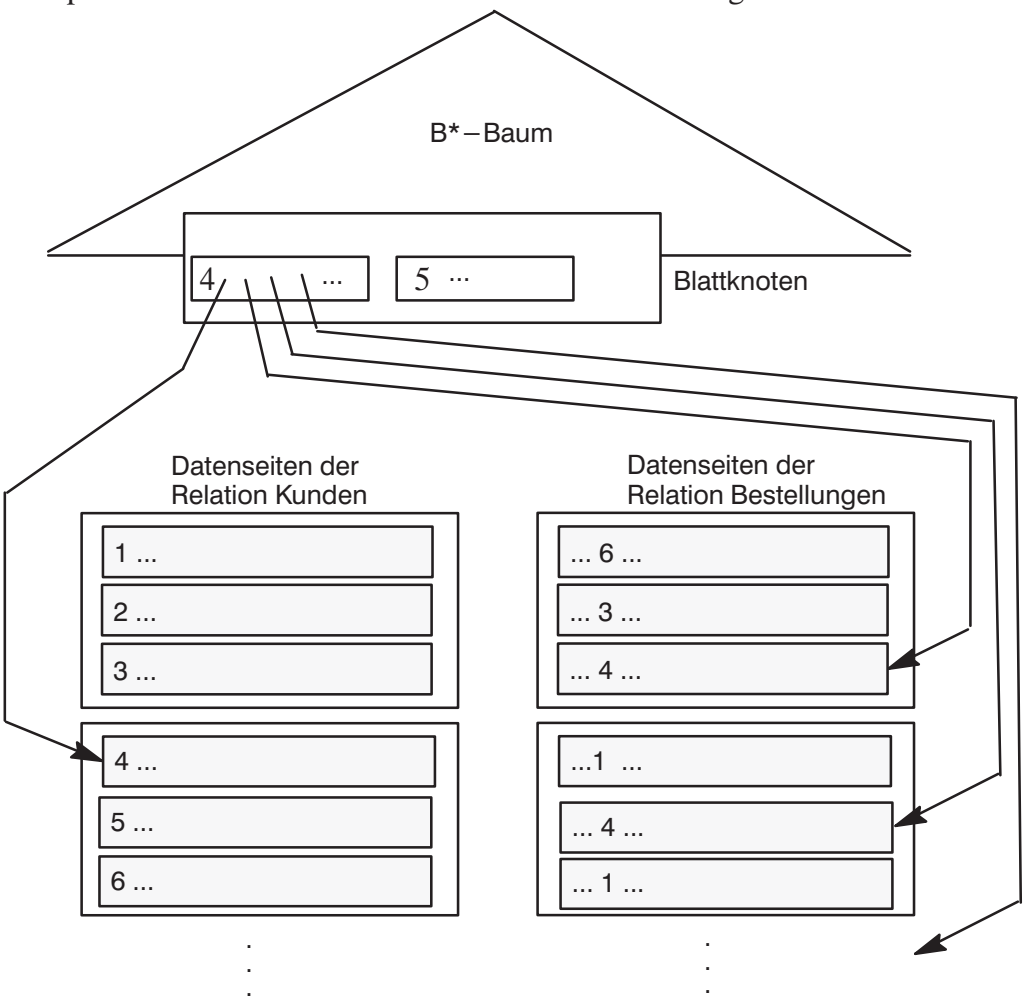
Einfügen einer neuen Bestellung mit KNr 18:



8.3 Indizes für Joins

Join−Index (Verallgemeinerter Zugriffspfad):
Mittelbarer B*−Baum−Index über dem Joinattribut A zweier Relationen R und S, bei dem in den Blättern für jeden Wert von A sowohl eine entsprechende R−Tupel−Verweisliste als auch eine S−Tupel−Verweisliste gespeichert ist.

Beispiel: Join−Index für Kunden.KNr und Bestellungen.KNr:



In Oracle realisierbar durch:

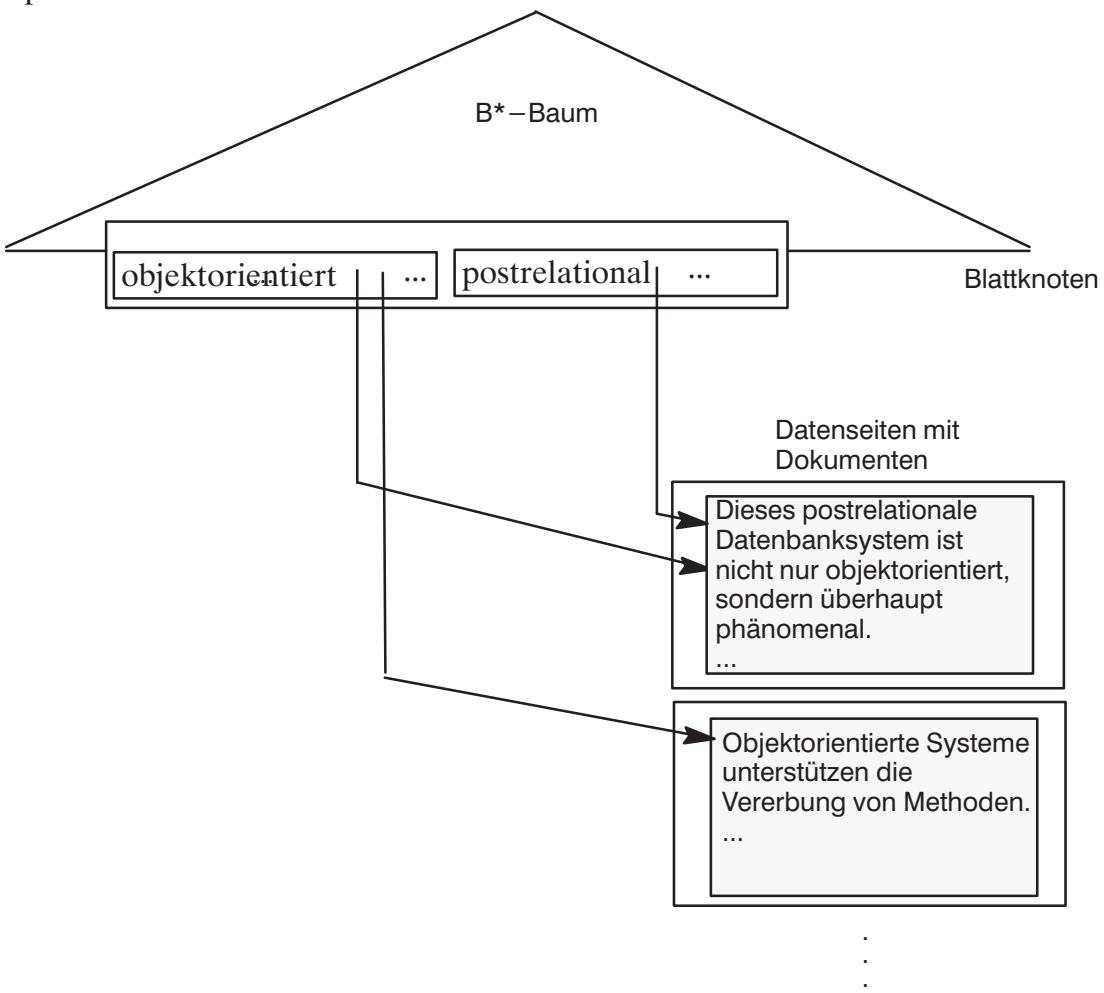
```
CREATE CLUSTER K−B−KNr−Cluster (KNr)
CREATE INDEX K−B−Join−Index ON CLUSTER K−B−KNr−Cluster
CREATE TABLE Kunden ( ... ) CLUSTER K−B−KNr−Cluster
CREATE TABLE Bestellungen ( ... ) CLUSTER K−B−KNr−Cluster
```

8.4 Indizes für Textsuche

Zweck:
Unterstützung von Information−Retrieval−Anfragen wie beispielsweise
(objektorientiert OR postrelational OR intelligent) AND
(Datenbank OR Informationssystem)
oder SQL−Anfragen mit Prädikaten vom Typ
Attribut LIKE '%Wert%'

Textindex:
Mittelbarer B*−Baum−Index für Deskriptoren von Dokumenten oder für Textfragmente (z.B. Wörter) in Textattributwerten

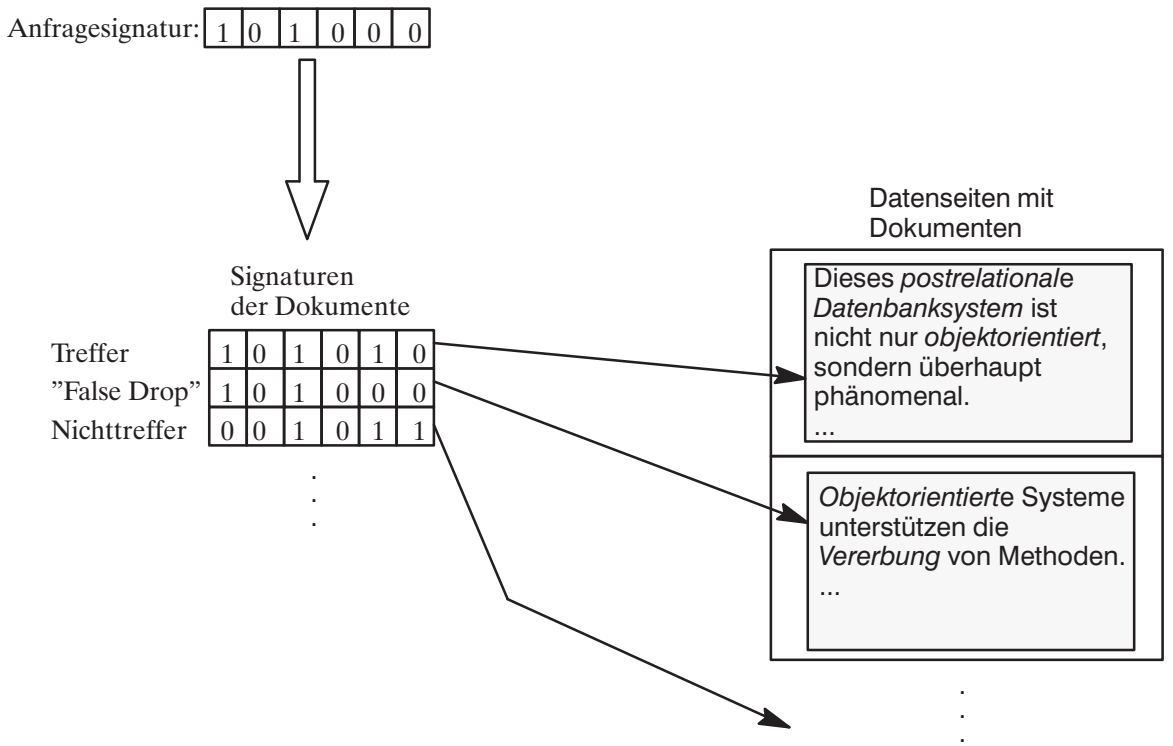
Beispiel:



Signaturen:
Jedes Dokument wird durch einen Bitvektor fester Länge, eine sogenannte Signatur, repräsentiert. Die in dem Dokument vorkommenden Deskriptoren bzw. Textfragmente werden per Hashing auf die Bitpositionen des Bitvektors abgebildet.
Die Signaturen aller Dokumente werden sequentiell (oder in einem speziellen Signaturbaum) gespeichert.

Anfragen vom Typ
{Dokumente d | d enthält die Deskriptoren d1, ..., dk}
werden durch Hashing von d1, ..., dk auf eine Anfragesignatur abgebildet, die mit allen Signaturen der gespeicherten Dokumente verglichen wird. Ein Dokument qualifiziert sich zunächst, wenn seine Signatur die Anfragesignatur enthält. Wegen der Nichtinjektivität der Signaturgenerierungsfunktion sind die so ausgewählten potentiellen Trefferdokumente nochmals genau zu überprüfen, ob sie wirklich alle Deskriptoren d1, .., dk enthalten (Elimination von “false drops”).

Beispiel:
Suche nach {Dokumente d | d enthält “objektorientiert” und “postrelational”)
(Annahme: Die Signaturgenerierungsfunktion bilde “objektorientiert” auf Bitposition 0 ab, “postrelational” auf 2, “Datenbanksystem” auf 4 und “Vererbung” auf 2.)



Weitere Techniken zur effizienten Textsuche basieren u.a. auf Digitalbäumen (sogenannten Tries), bei denen die Suche nicht auf kompletten Schlüsselvergleichen beruht, sondern auf positionsweisen Vergleichen (also z.B. bezüglich des $i-$ten Buchstaben in einem Wort).

8.5 Indizes für objektorientierte DBS

Materialisierung und Indizierung von Methodenresultaten

Wenn die Resultate von Methoden sehr häufig abgefragt werden und sich nur relativ selten aufgrund von Änderungsoperationen auf den Objekten verändern, kann es sinnvoll sein, das Resultat eines Methodenaufrufs im Objekt selbst zu speichern, so daß bei einer entsprechenden Abfrage nur das Resultat gelesen werden muß und die Methode selbst gar nicht ausgeführt wird. Diese "*Materialisierung*" des Methodenresultats ist nach außen transparent und kann somit jederzeit von Objekt zu Objekt wieder aufgehoben werden. Insbesondere muß ein materialisiertes Methodenresultat bei einer Änderung des Objekts (oder evtl. auch referenzierter Objekte) invalidiert oder aktualisiert werden. Beim Zugriff auf invalidierte, materialisierte Methodenresultate muß dann die Methode wieder ausgeführt und das Resultat ggf. rematerialisiert werden.

Eine besondere Form der Materialisierung von Methodenresultaten ist der Aufbau eines Index für die Methode. Dabei wird für jedes Objekt der indizierten Klasse die Methode ausgeführt, und ihr Resultat wird zusammen mit der OID des Objekts in einem Index gespeichert. Dieser Schritt muß zur Wartung des Index bei jeder relevanten Objektänderung wiederholt werden.

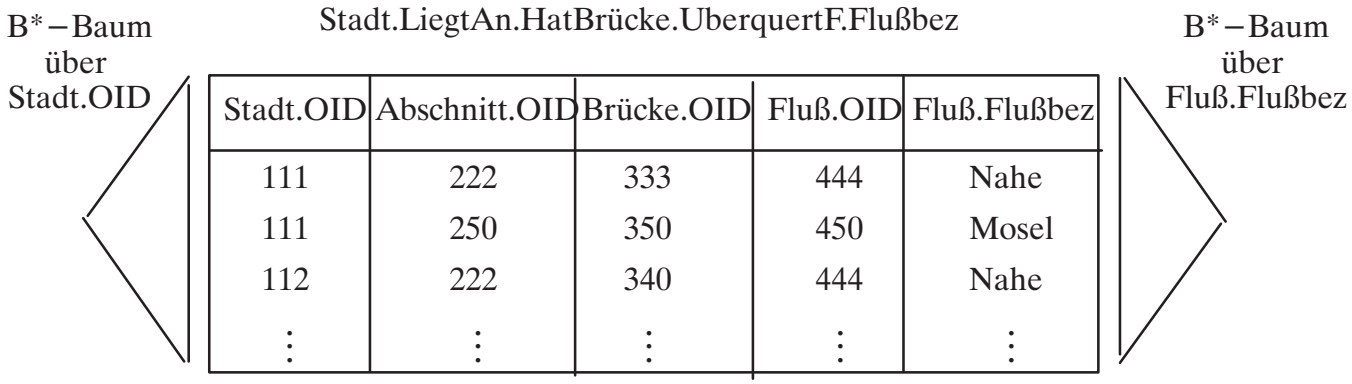
Materialisierung von Methodenresultaten ist vor allem dann vorteilhaft, wenn die Methodenausführung sehr aufwendig ist und sich die materialisierten Resultate nur selten ändern. Indizierung von Methodenresultaten ist darüber hinaus dann vorteilhaft, wenn die Methode häufig in Suchprädikaten zeitkritischer Anfragen verwendet wird.

Beispiel:
Materialisierung und ggf. Indizierung für die Methode "Fläche" auf der Klasse "Polygon".

Pfadindizes: Unterstützung von Pfadausdrücken in Anfragen

Wenn Attribute, die am Ende eines Referenzenpfades stehen, indiziert werden, kann es vorteilhaft sein, im Index nicht nur die Attributwerte selbst, sondern den gesamten Pfad in Form der OIDs der auf dem Pfad liegenden Objekte zusammen mit dem Attributwert zu speichern. Für solche Pfade können z.B. B*−Baum−Strukturen sowohl für den Anfang des Pfades (also die erste OID eines Pfades) als auch für den am Pfadende stehenden Attributwert aufgebaut werden. Diese Art von *Pfadindizes* verallgemeinert den sog. Join−Index, der in einigen relationalen DBS realisiert ist.

Beispiel:



Dieser Pfadindex unterstützt Queries der Typen:

- Welche Flüsse werden von den Straßenabschnitten überquert, an denen eine bestimmte Stadt liegt?
- Welche Städte liegen an Straßenabschnitten, die einen bestimmten Fluß überqueren?

8.6 Indizes für Partial−Match− und mehrdimensionale Bereichsanfragen

Ein mehrdimensionaler Index über den Attributen A1, ..., Ak erlaubt die effiziente Bestimmung aller Treffer für Prädikate der Form

A_{i_1}=wert1 AND ... AND A_{i_j}=wertj

oder

u1 <= A_{i_1} AND ... AND uj <= A_{i_j} AND

A_{i_1} <= o1 AND ... AND A_{i_j} <= oj

für alle Teilmengen $\{A_{i_1}, ..., A_{i_j}\} \subseteq \{A1, ..., Ak\}$ und beliebige Konstanten u1, ..., uj, o1, ..., oj.

Beispiel:
Ein mehrdimensionaler Index auf der Produkte−Relation über der Attributmenge {Preis, Gewicht, Lager} unterstützt Anfrageprädikate der Typen:

Preis = 50
Gewicht > 100
Lager = 'Homburg'
Preis=50 AND Gewicht > 100
Lager = 'Homburg' AND Gewicht > 100
Preis=50 AND Lager = 'Homburg'
Preis=50 AND Gewicht > 100 AND Lager = 'Homburg'

Mehrdimensionale Indizes spielen eine wichtige Rolle u.a. für die Verwaltung räumlicher Daten, bei denen die indizierten Attribute den kartesischen Raumkoordianten von geometrischen Objekten (Punkte, Polygone, ...) entsprechen.

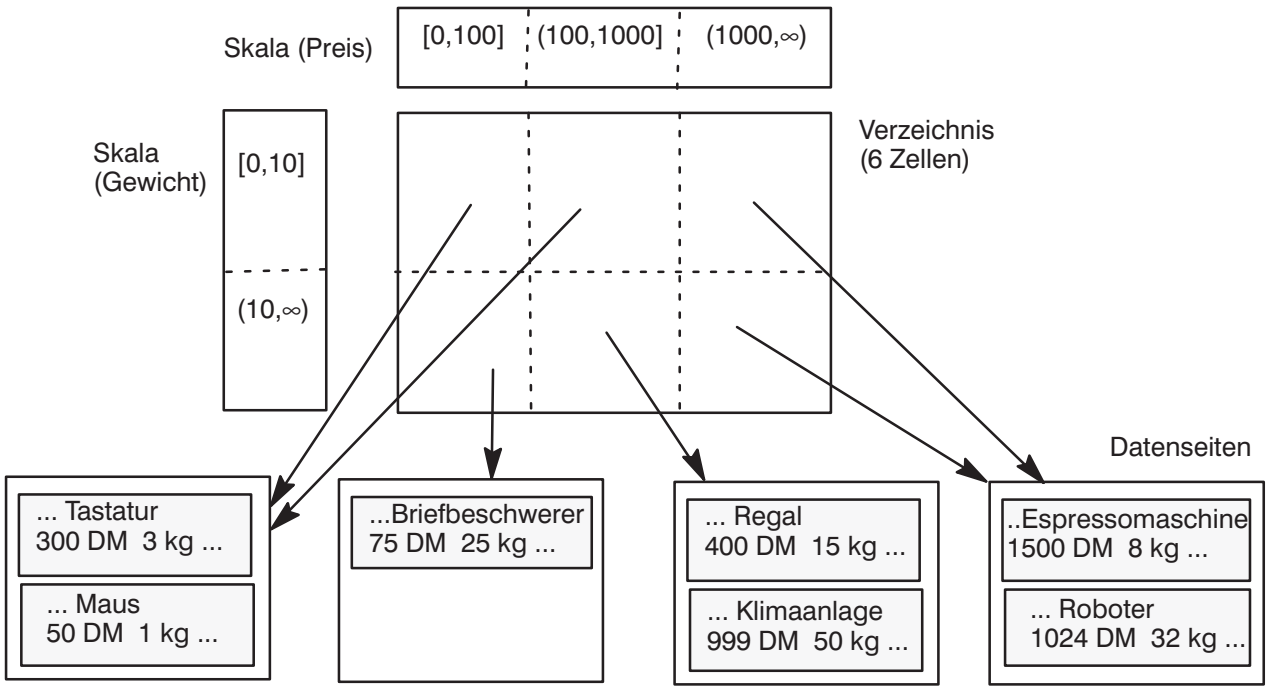
Implementierung eines mehrdimensionaler Index durch das Grid−File

Ein Grid−File für Attribute A1, ..., Ak besteht aus

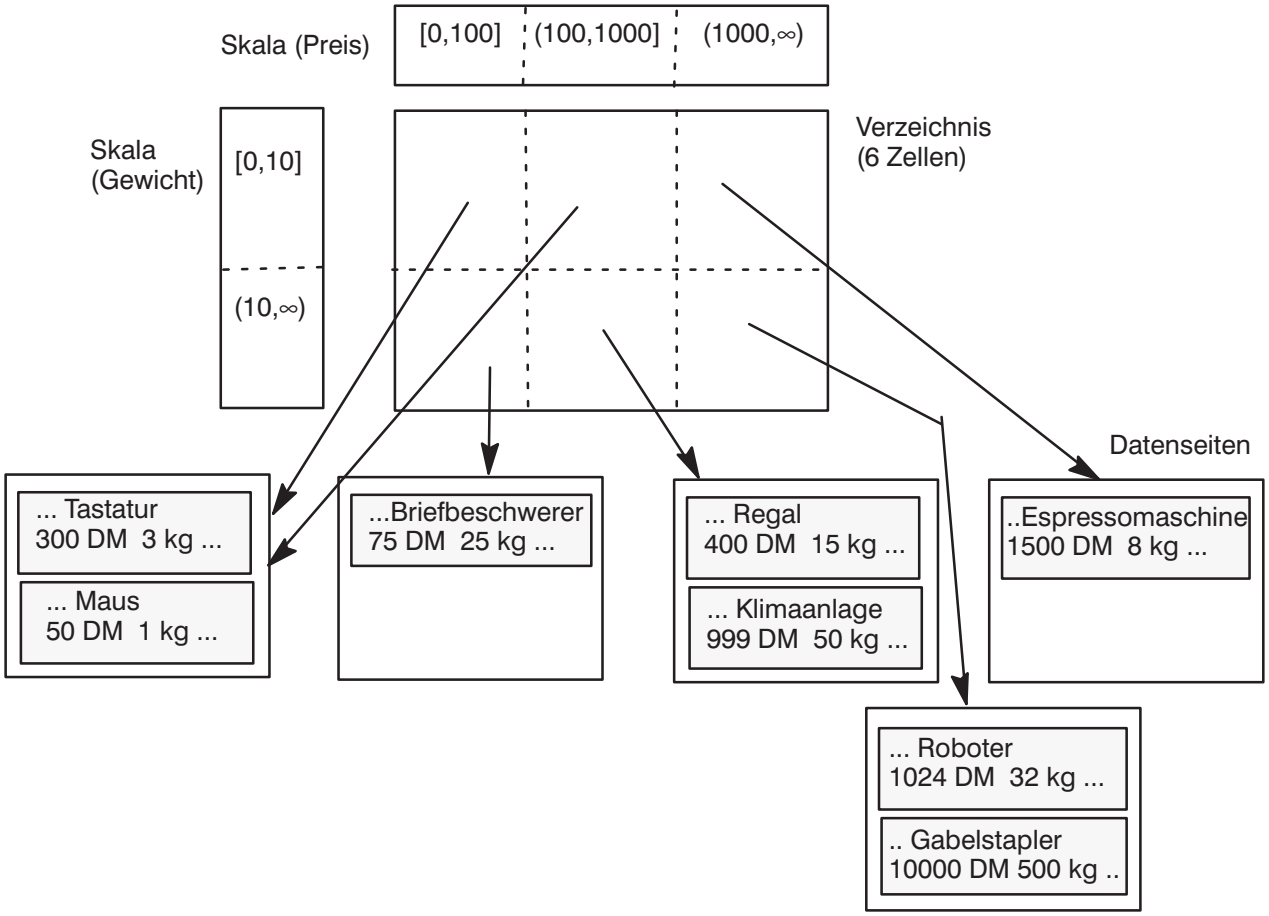
- k *Skalen (engl.: scales)*, die jeweils für ein Attribut Aj eine Partitionierung von Dom(Aj) in Form von m_j Werteintervallen spezifizieren,
- einem gitterförmigen *Verzeichnis (engl.: directory)* von $\prod_{j=1}^{k} m_j$ *Zellen*, die durch die k Skalen wie in einem k−dimensionalen Array indiziert werden, wobei die Zelle mit den Indizes $i_1, ..., i_k$ auf einen Datenblock verweist, der alle Datensätze enthält, deren Aj−Wert in das i_j−te Werteintervall fällt (für alle j: $1 \leq j \leq k$), und nur diese Sätze.

Effiziente Verwaltung des Directories auf Sekundärspeicher:
→ Baumstrukturen (z.B. kdB−Bäume, hB−Bäume) oder
hierarchische Grid−Files (z.B. BANG−File)

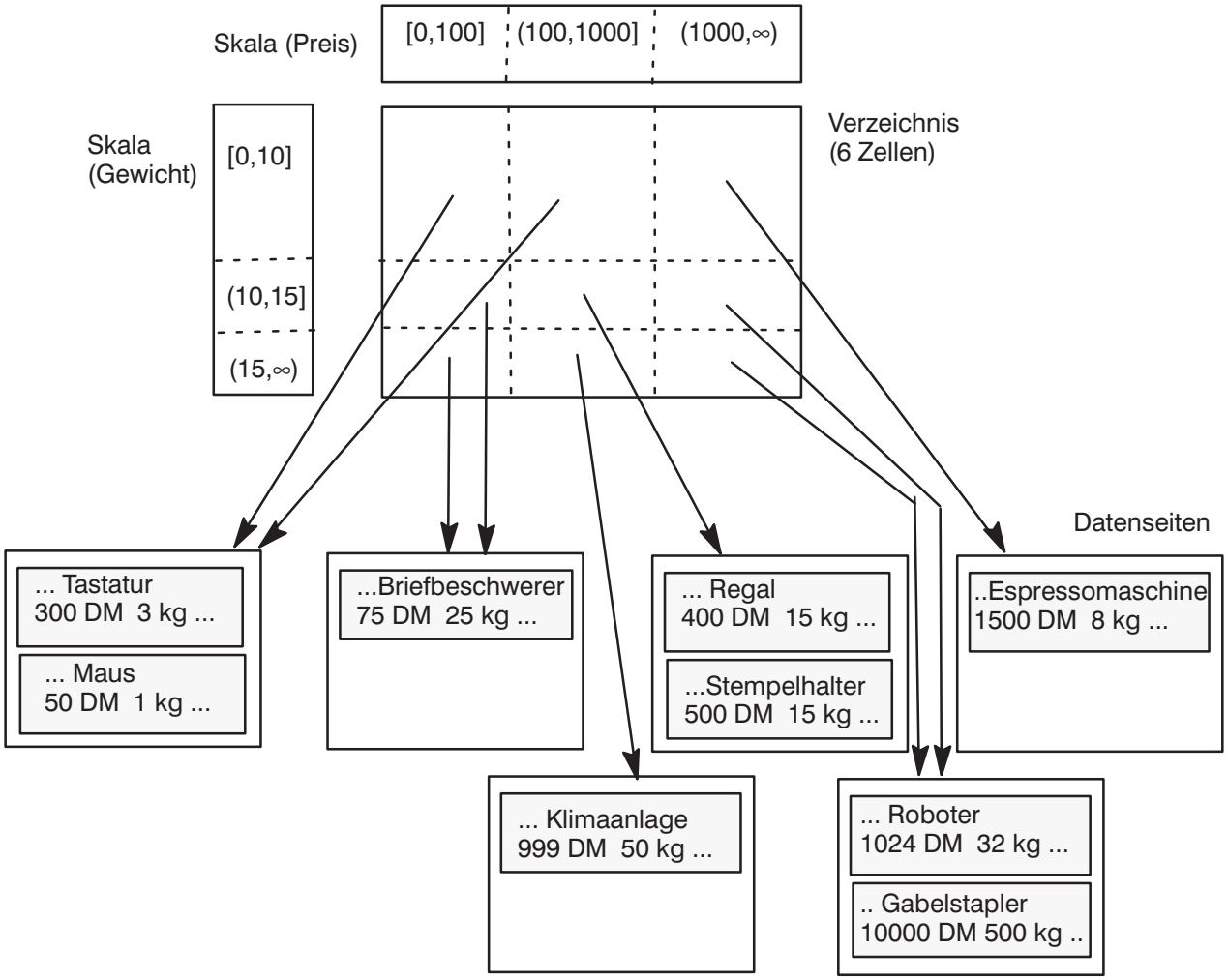
Beispiel: Grid−File für Produkte.{Preis, Gewicht}



Einfügen von (... Gabelstapler, 10000 DM, 500 kg, ...):



Einfügen von (... Stempelhalter, 500 DM, 15 kg, ...):



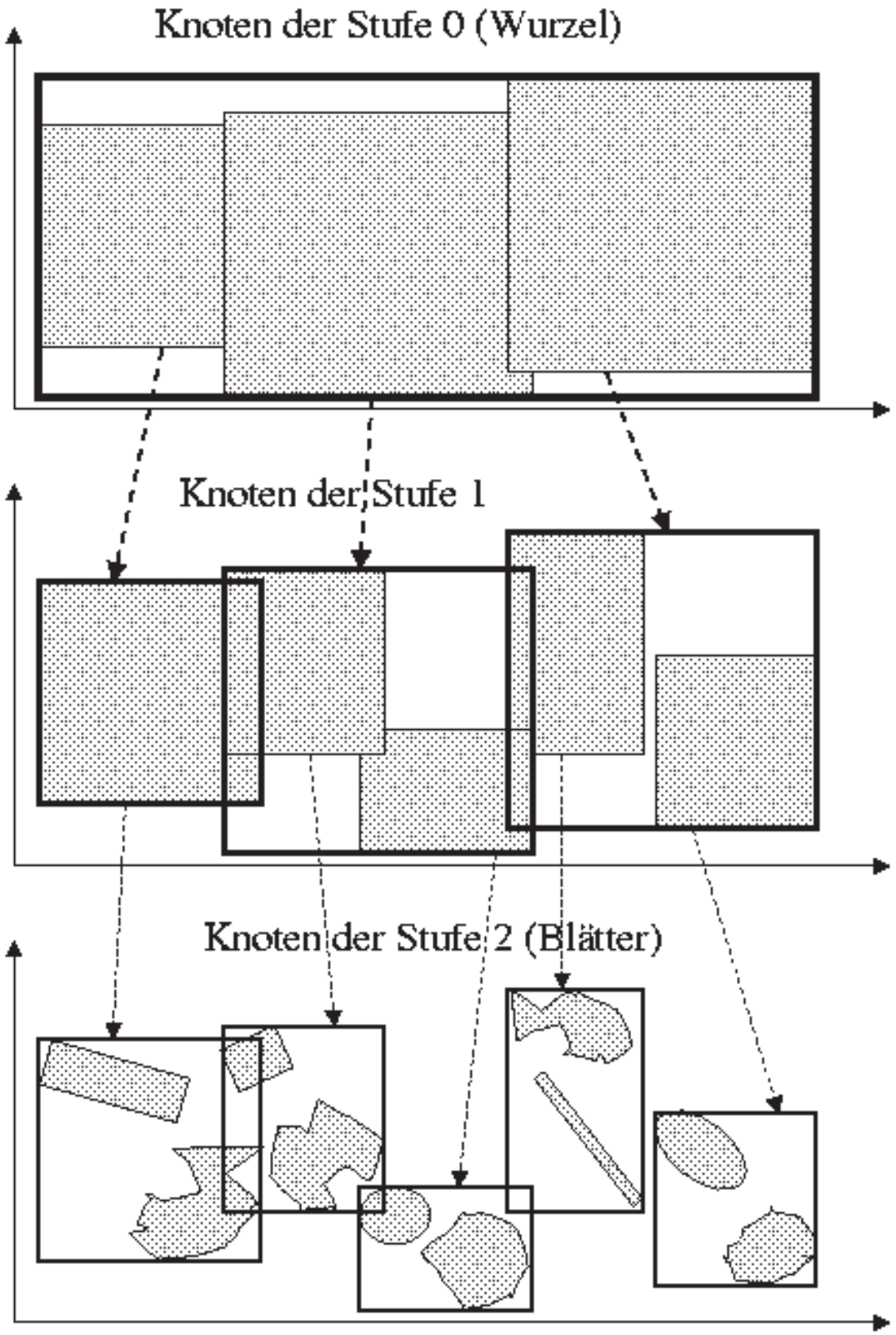
Implementierung eines mehrdimensionalen Index für räumliche Objekte durch den R−Baum

Räumliche Objekte werden durch ihr kleinstes umschreibendes, achsenparalleles Rechteck (minimum bounding rectangle = MBR) approximiert.
Ein R−Baum ist ein hohler Mehrwegebaum über solchen MBRs, bei dem ein Nichtblattknoten v einen Verweis auf einen Knoten s mit dem MBR aller Einträge von s assoziiert. MBRs werden dabei durch die Koordinaten des linken unteren und des rechten oberen Eckpunkts repräsentiert.

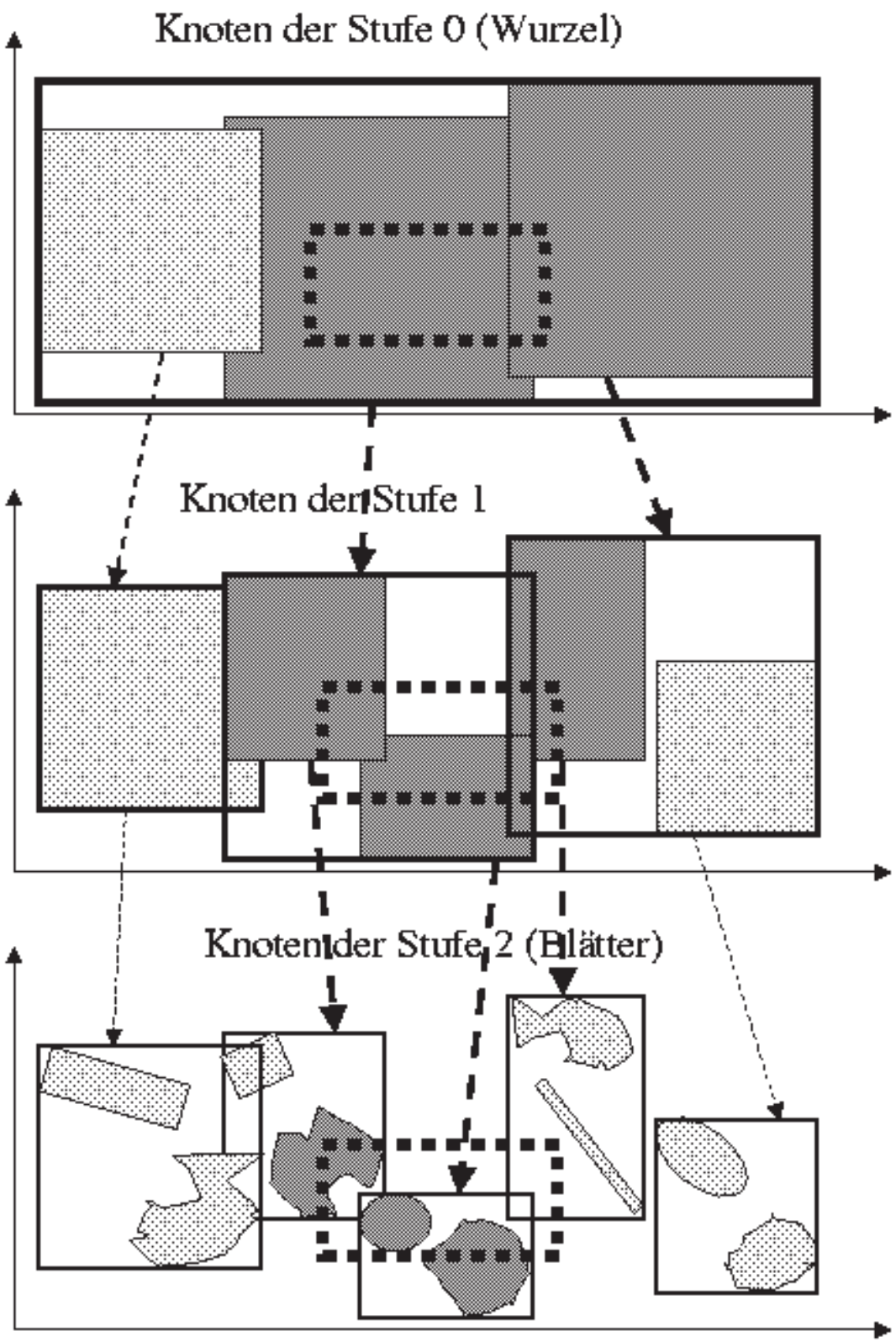
Zur Suche nach allen Objekten, die ein vorgegebenes Such−MBR schneiden, in einem R−Baum mit Wurzel t sind alle Teilbäume von t zu durchsuchen, deren MBR das Such−MBR schneidet. Ähnlich wie bei Signaturen sind die so bestimmten potentiellen Treffer nochmals bezüglich des Anfrageprädikats zu prüfen, und Nichttreffer sind zu eliminieren.

Beim Einfügen in einen R−Baum ist die Überschneidung der MBRs verschiedener Bruderknoten möglichst gering zu halten. Dazu werden Heuristiken verwendet, z.B. Einfügen in denjenigen Knoten einer Baumebene, dessen Umfang oder Fläche durch das neue Objekt minimal vergrößert wird. Beim Einfügen in einen vollen Knoten erfolgt ein Split analog zum B*−Baum.

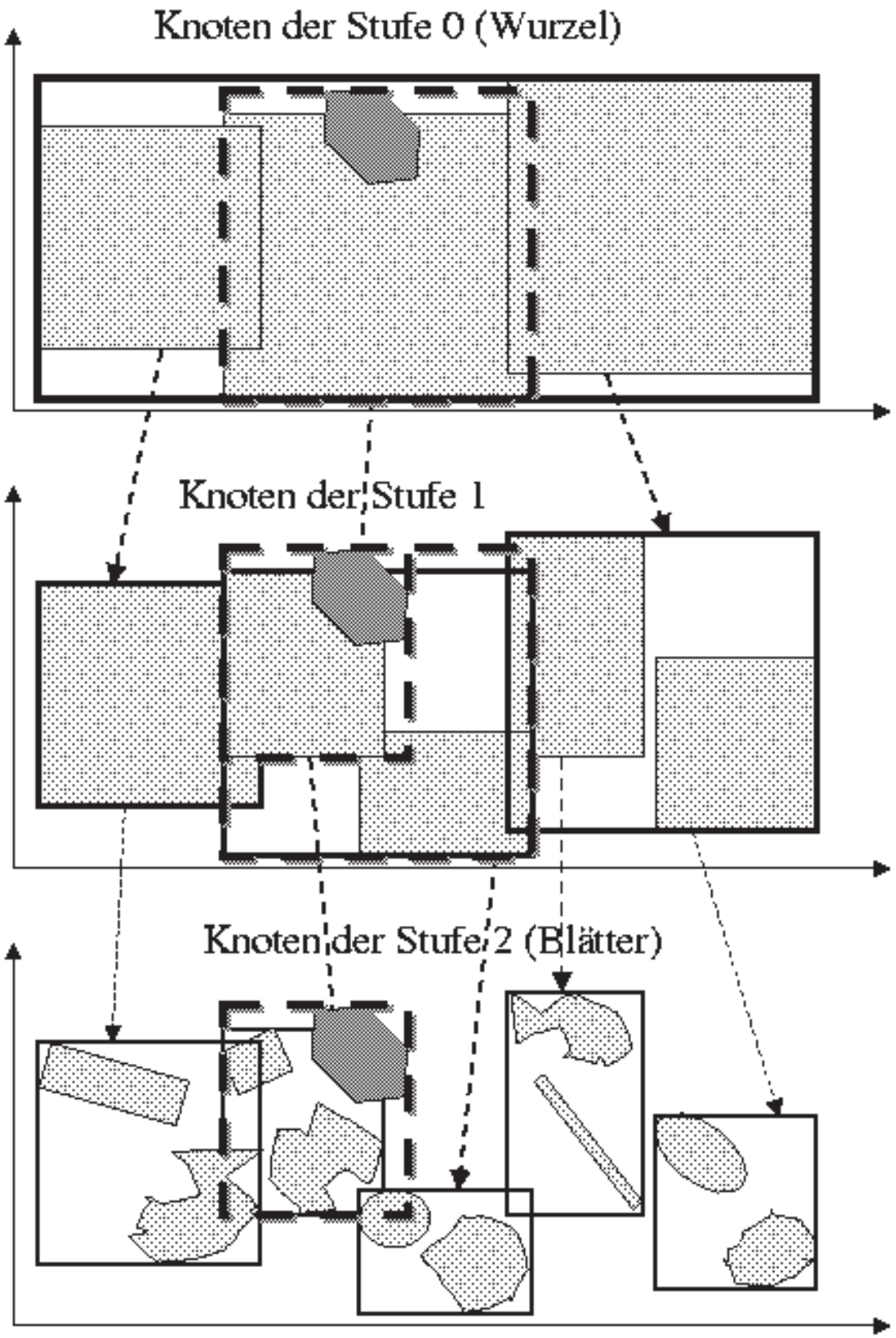
Beispiel eines R−Baums der Höhe 3:



Bearbeitung einer Rechteck−Anfrage für das R−Baum−Beispiel:



Bearbeitung einer Einfügung für das R−Baum−Beispiel:



8.7 Problem des Physischen Datenbankentwurfs

Gegeben:
- Datenbankschema (als Resultat des ”logischen” Datenbankentwurfs)
- Lastprofil:
  - Anfragetypen und deren Häufigkeiten
  - Änderungsoperationen und deren Häufigkeiten
- Kostenmäßige Restriktionen (z.B. bzgl. Speicherplatz)

Gesucht:
- Festlegung von Speicherungsstrukturen und Zugriffspfaden
  (das ”physische” Datenbankschema), so daß
  die mittlere Antwortzeit bzw. die mittleren Zugriffskosten minimiert werden.

→ Aufgabe des Datenbankadministrators bzw. Performance−Spezialisten
→ mathematische Optimierungsmodelle, ”Tuning−Werkzeuge”

Ergänzende Literatur zu Kapitel 8:

T. Härder, E. Rahm, Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer–Verlag, 1998

J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993, Chapter 15

T. Härder, Realisierung von operationalen Schnittstellen, in: P.C. Lockemann, J.W. Schmidt (Hrsg.), Datenbank–Handbuch, Springer–Verlag, 1987

B. Salzberg, File Structures: An Analytic Approach, Prentice Hall, 1988

T.J. Teorey, J.P. Fry, Design of Database Structures, Prentice Hall, 1982

A.L. Tharp, File Organization and Processing, John Wiley & Sons, 1988

R.J. Enbody, H.C. Du, Dynamic Hashing Schemes, ACM Computing Surveys Vol.20 No.2, 1988, pp. 85–113

C. Faloutsos, Access Methods for Text, ACM Computing Surveys Vol.17 No.1, 1985, pp. 49–74

P. Widmayer, Datenstrukturen für Geodatenbanken, in: G. Vossen, K.–U. Witt (Hrsg.), Entwicklungstendenzen bei Datenbanksystemen, Oldenbourg–Verlag, 1991

H. Samet, The Design and Analysis of Spatial Data Structures, Addison–Wesley, 1989