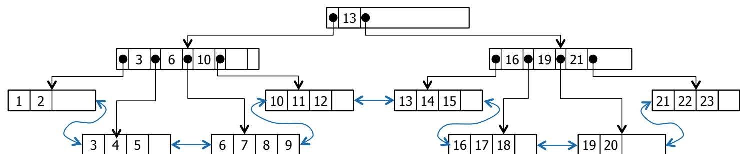
Sommersemester 2016

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

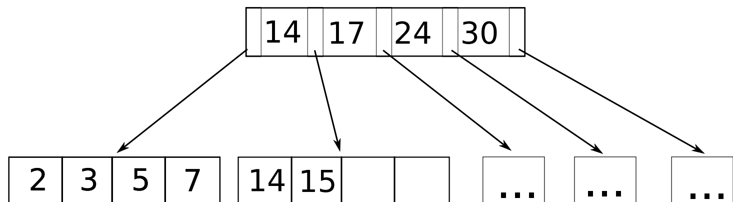
B+ Baum



- Jeder Knoten hat die Größe eines I/O Blocks.
- Ein Knoten ist zu mindestens 50% gefüllt.
- Das Lesen eines Knotens verursacht typischerweise einen wahlfreien Zugriff auf Festplatte
- Ein B+ Baum ist i.d.R. sehr flach, d.h. Suche verursacht nur wenige wahlfreie Zugriffe.
- Zudem, die ersten 1-2 Ebenen des Baums sind i.d.R. im Hauptspeicher "gecached".

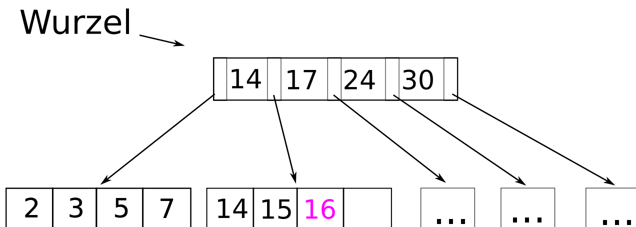
B+ Baum: Komplexeres Einfügebeispiel

Einfügen von 16 und 8



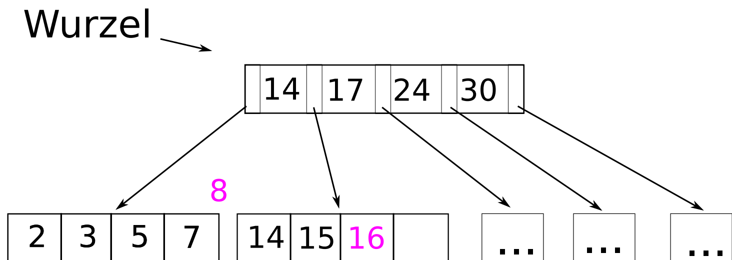
B+ Baum: Komplexeres Einfügebeispiel

Einfügen von 16 und 8



B+ Baum: Komplexeres Einfügebeispiel

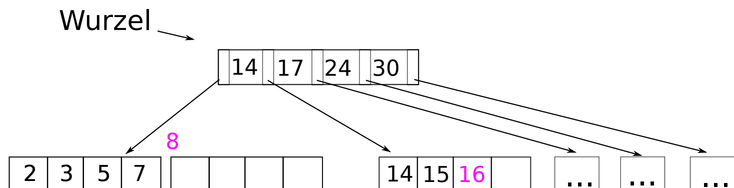
Einfügen von 16 und 8



Zum Einfügen der 8 in den Blattknoten ist kein Platz vorhanden.

B+ Baum: Komplexeres Einfügebeispiel

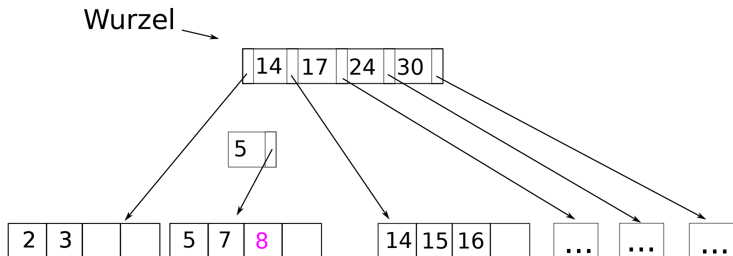
Einfügen von 16 und 8



Es wird also ein neuer Blattknoten erzeugt.

B+ Baum: Komplexeres Einfügebeispiel

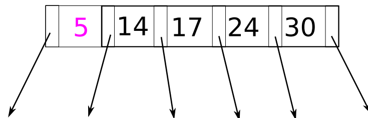
Einfügen von 16 und 8



... und der mittlere Wert nach oben kopiert.

B+ Baum - Komplexeres Einfügebeispiel

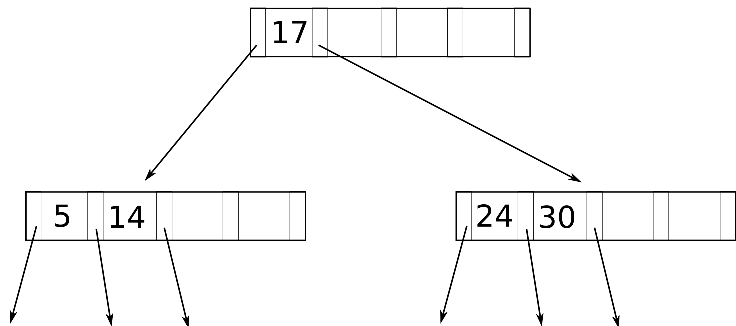
Einfügen von 16 und 8



Dieser innere Knoten ist nun voll und muss ebenfalls gesplittet werden. Dabei wird der mittlere Wert (17) in einen neuen inneren Knoten eingefügt.

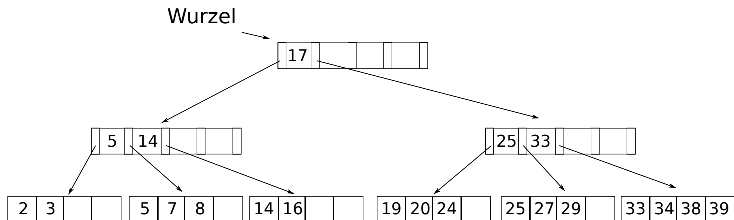
B+ Baum: Komplexeres Einfügebeispiel

Einfügen von 16 und 8



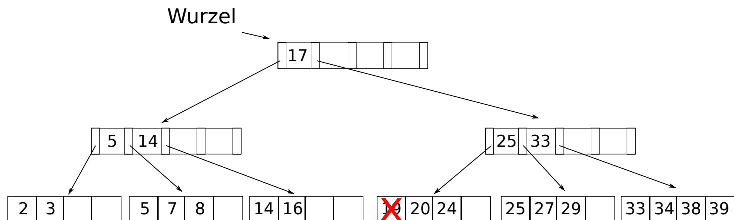
B+ Baum: Löschen Beispiel

Löschen von 19 und 20



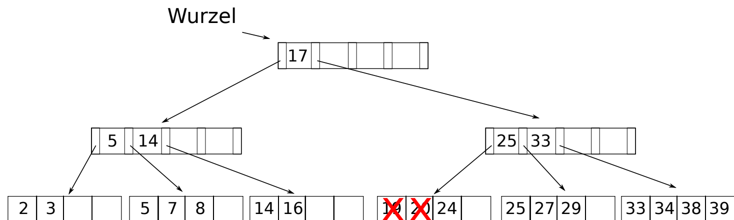
B+ Baum: Löschen Beispiel

Löschen von 19 und 20



B+ Baum: Löschen Beispiel

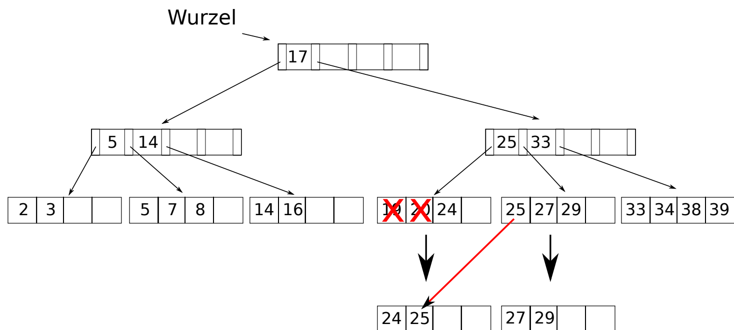
Löschen von 19 und 20



Verletzung von Bedingung, dass jedes Blatt mindestens $k^* = 2$ Einträge haben muss. Also muss ausgeglichen werden.

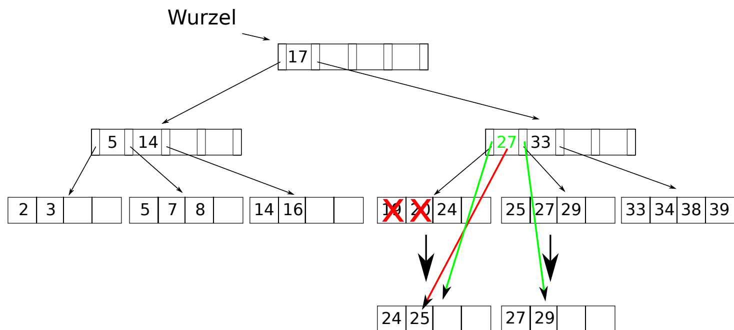
B+ Baum: Löschen Beispiel

Löschen von 19 und 20



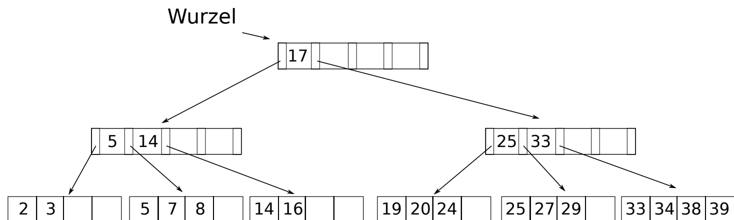
B+ Baum: Löschen Beispiel

Löschen von 19 und 20



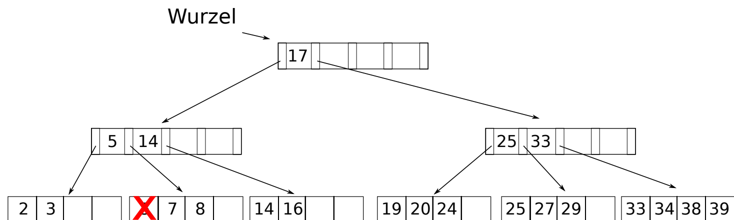
B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



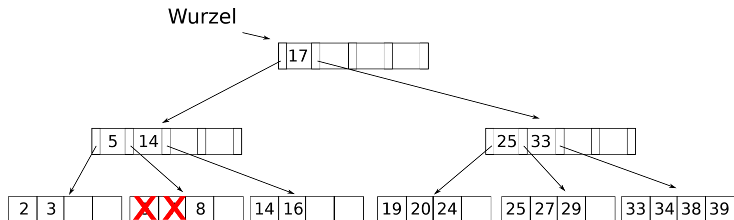
B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



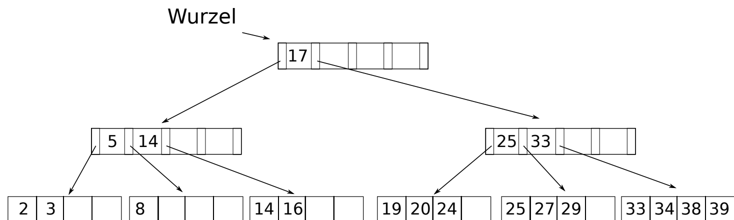
B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



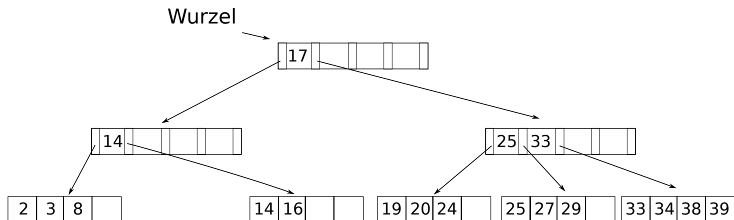
B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



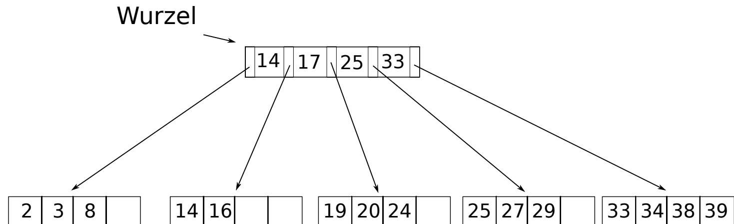
B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



B+ Baum: Löschen Beispiel 2

Löschen von 5 und 7



B* Baum

Statt wie im B+ Baum eine volle Seite auf zwei Seiten aufzuteilen, werden im B* Baum die Datensätze aus m Geschwisterknoten gleichmässig auf $(m+1)$ Seiten aufgeteilt.

Dies **verbessert die Speicherplatzausnutzung (SPAN)**:

SPAN	$m = 1$	$m = 2$	m
worst case:	$\frac{1}{1+1}$	$\frac{2}{2+1}$	$\frac{m}{m+1}$
avg. case:	$ln2\ (= 69\%)$	81%	$m * ln(\frac{m+1}{m})$

Allerdings werden die Kosten für das Einfügen erhöht, so dass in der Praxis $m \leq 3$.

Präfix B-Baum

Beobachtung

- Die Einträge in den inneren Knoten eines B+ Baums werden nur zur Navigation während der Suche benutzt.
- Einträge bestehen aus Verweisen (Pointern) und aus Schlüsseln.
- Pointer sind nur ein paar Bytes groß, aber Schlüssel können sehr lang sein (z.B. Donaudampfschifffahrtsgesellschaftskapitän)
- Wenn Platz gespart sind, passen mehr Einträge in einen Knoten. D.h. Baum wird flacher!

Idee

- Verwende nicht die gesamten Schlüssel, wie sie im Fall eines B+ Baums anfallen würden, sonder geeignete Präfixe.
- Ein Präfix B Baum ist ein B+ Baum, bei dem der Index-Teil des B+ Baums durch einen Index-Teil Präfix B-Baum ersetzt wurde.

Literatur: R. Bayer und K. Unterauer. Prefix B-trees. TODS, 1977.

Beispiel Split und Separator

Gegeben folgender volle Blattknoten eines B+ Baums:

Bigbird, Burt, Cookiemonster, Ernie, Snuffleopogus

Um den Schlüssel "Grouch" in diese Seite einzufügen, muss sie wie folgt in zwei Seiten aufgespalten werden:

Bigbird, Burt, Cookiemonster

Ernie, Grouch, Snuffleopogus

Statt nun "Ernie" als Schlüssel im Index zu benutzen, würde es auch ausreichen "D" oder "E" für den gleichen Zweck zu benutzen.

Separator

Im Allgemeinen kann man jeden String s benutzen, falls er die Eigenschaft hat, dass

$$\text{Cookiemonster} < s \leq \text{Ernie}$$

s kann dann im Index gespeichert werden, um die obigen beiden Seiten zu trennen.

Idee

Wenn man den möglichst kürzesten Separator benutzt, geht nur ein Minimum an Platz innerhalb einer Seite verloren. Der Baum wird flacher (weil höherer "Fan-Out")

Präfix-Eigenschaft

- Sind die Schlüssel Worte über einem Alphabet und ist die Ordnung der Schlüssel in alphabetischer (lexikographischer) Ordnung, dann gilt folgende Eigenschaft:

Präfix-Eigenschaft

Seien x und y zwei Schlüssel, so dass $x < y$. Dann gibt es einen eindeutigen Präfix $\hat{y}$ von y mit:

(a) $\hat{y}$ ist ein Separator zwischen x und y und

(b) kein anderer Separator zwischen x und y ist kürzer als $\hat{y}$.

Es kann auch Sinn machen Index-Knoten nicht genau in der "Mitte" zu splitten, sondern leichte Abweichung davon zuzulassen. Wieso? Was ist ein guter Separator zwischen Donaudampfschifffahrtsdampferhersteller und Donaudampfschifffahrtsgesellschaft?

Bulkloading

Problemstellung

- Gegeben eine große Menge an Datensätzen
- Wie kann ein B+ Baum darüber aufgebaut werden?
- Das Problem ist natürlich allgemeiner und tritt auch bei anderen Bäumen/Indexstrukturen auf.

Naive Möglichkeit

- Datensätze einzeln nach und nach in B+ Baum einfügen
- D.h. es wird immer von der Wurzel ab nach der passenden Einfügestelle gesucht
- Nicht sehr effizient (auch wenn die oberen Ebenen in einem DB-Puffer sind)

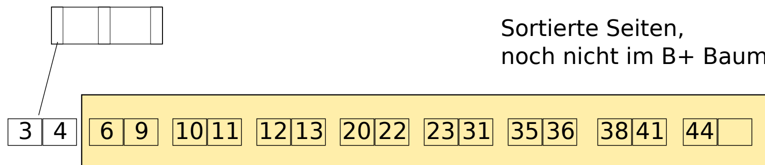
Bulkloading

Idee

- Sortiere Daten vor
- Dann baue Baum auf, indem Datensätze der Reihe nach hinzugefügt werden

Ablauf

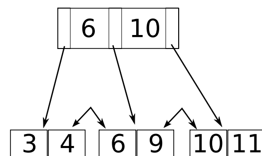
Der B+ Baum kann 2 Einträge pro Seite speichern. Wir fügen je ganze Seiten ein, hier zuerst die Seite mit Einträgen 3 und 4.



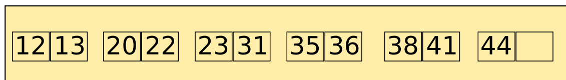
Buch: Ramakrishnan & Gehrke. Database Management Systems.

Bulkloading - Beispiel (2)

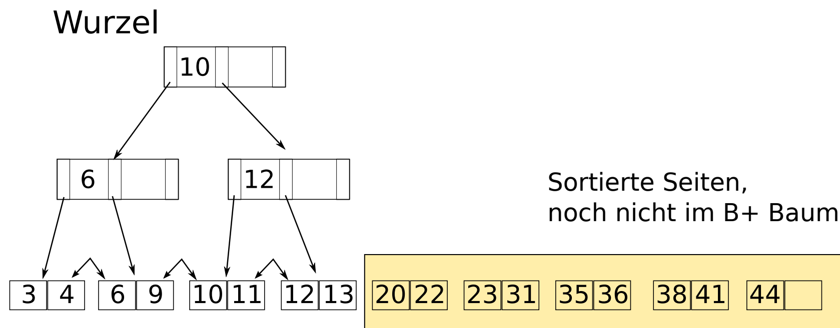
- Nachdem die beiden folgenden Seiten eingefügt worden sind ist die Wurzel nun voll.
- Bei dem Einfügen von (12,13) muss die Wurzel also aufgeteilt werden.



Sortierte Seiten,
noch nicht im B+ Baum



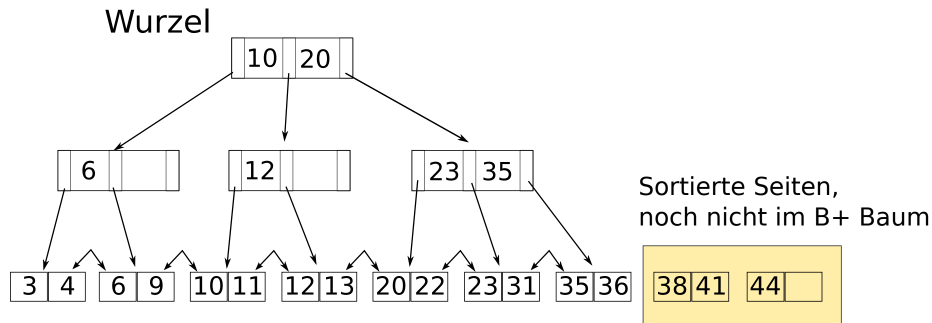
Bulkloading - Beispiel (3)



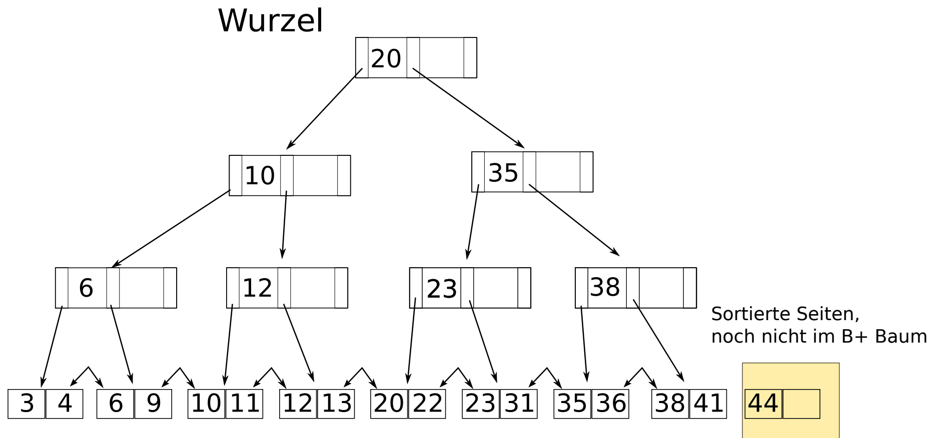
- Beim Aufteilen der Seiten wurden diese hier gleichmäßig auf die neuen Seiten unter der Wurzel aufgeteilt
- Im Allgemeinen hätte man auch anders aufteilen können.
- Z.B. nach einem bestimmen Füllgrad (z.B. 80%) oder man hätte auch alle alten Einträge in der linken Seite belassen können.

Bulkloading - Beispiel (4)

- Index-Einträge für die Blätter werden immer in den am weitesten rechts stehenden Index-Knoten direkt über der Blatt-Ebene eingefügt.
- Sollte dieser voll sein, so muss aufgeteilt werden.



Bulkloading - Beispiel (5)



Hashverfahren

- Gegeben: Domäne der Schlüssel $S = \{0,...M-1\}$
- M kann sehr groß sein
- Anfragen der Form: welche Datensätze haben den Schlüssel x?
- Sogenannte **Punktanfragen**

Grundidee

- **Abbildung auf Adressbereich** A, $|A| << |S|$
- $h : S \rightarrow A$ ordnet jedem Schlüssel einen Wert in A zu
- Gewünschte Eigenschaften:
  - Verteile Schlüssel gleichmäßig über Adressbereich
  - **Vermeide Kollisionen**
  - Kollision: verschiedene Werte in S werden auf den gleichen Wert in A abgebildet

Einheiten im Adressbereich werden auch Eimer (Englisch: **Buckets**) genannt.

Hashverfahren (2)

Man nimmt an, dass die Zahl der benutzten Schlüssel K viel kleiner ist als die Domäne S.
Abbildung h muss also nur die benutzten Schlüssel in K ordentlich abbilden.

Gefahr von Kollisionen

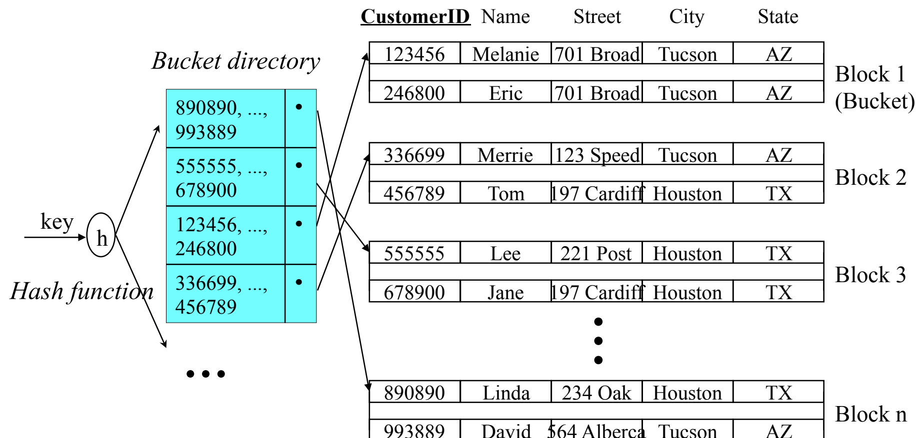
Möglichkeiten die Hashfunktion zu wählen

- $h(k) = k \, mod \, q$, q bestimmt Größe des Adressbereichs
- $h(k) = k^2$ oder $h(k) = c * k$, dann Auswahl gewisser Bitpositionen, um gültigen Adressbereich zu erhalten.

Fortgeschrittene Hashmethoden ($\Rightarrow$ DBS VL im Winter)

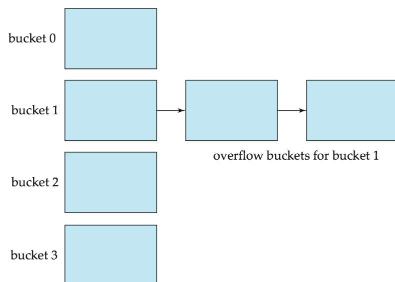
- Diese Verfahren sind sogenannte statische Verfahren
- Es gibt aber auch dynamische Verfahren, die die Hashfunktion an den Bedarf anpassen (Erweiterbares Hashing).

Statischer Hash Index: Beispiel



Statischer Hash Index

- Suche (Lookup)
  - Ein Zugriff auf auf das Verzeichnis (directory)
  - Ein Zugriff auf die eigentliche Datei
- Performance hängt von Wahl der Hash-Funktion ab
- Überlauf von Buckets
  - Zu viele verschiedene Schlüssel-Werte werden auf gleichen Bucket abgebildet
  - Lösung: **Überlaufbehandlung** mittels Verkettung von Überlauf-Buckets



Indexe in Postgres: B+ Baum

```
create table MeineTabelle (
        ID int,
        major int,
        minor int,
        name varchar
);
```

B+ Baum mit Schlüssel ID

```
create index MeineTable_i1 on MeineTabelle ID;
```

B+ Baum mit Schlüssel (major, minor)

```
create index MeineTable_i2 on MeineTabelle (major, minor);
```

Indexe über mehrere Spalten

Sogenannte **Composite-Key Indexe**.
Angabe einer Reihe (**Achtung! Reihenfolge ist wichtig!**) von Spalten.

create index indexName **on** MeineTabelle (att1 , att2, att3);

Tupel werden dann im Index sortiert anhand dieser Attribute, "Lexikographische Ordnung": att1 ist primäres Kriterium, gefolgt von att2, etc.

Optional mit Ordnung ASC oder DESC der einzelnen Attribute. Default ist ASC. Zum Beispiel:

create index indexName **on** MeineTabelle (att1 **DESC** , att2, att3);

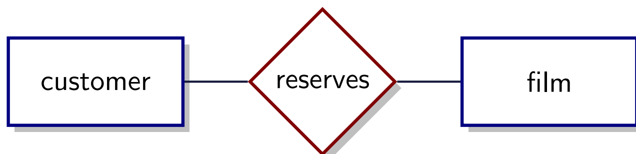
Indexe in Postgres: Hash

- **create index** name **on** table **using** hash (column);

Ein Hash-Index macht Sinn, falls:

- **Keine Bereichsanfragen** benötigt werden
- Order by (Schlüssel) nicht benötigt wird
- Keine oder nur kleine Joins über den Schlüsseln notwendig sind

Beispiel Datenbank



- **customer** (customerID, name, street, city, state)
- **reserved** (customerID, filmID, resDate)
- **film** (filmID, title, kind, rentalPrice)

Verschiedene Selektionen

- Primärschlüssel, Punktanfrage

  $$\sigma_{filmID=2}(film)$$

- Punktanfrage

  $$\sigma_{title='Terminator'}(film)$$

- Bereichsanfrage

  $$\sigma_{1<rentalPrice<4}(film)$$

- Konjunktion (d.h. logisches und)

  $$\sigma_{kind='F' \ \wedge \ rentalPrice=4}(film)$$

- Disjunktion (d.h. logisches oder)

  $$\sigma_{rentalPrice<2 \ \vee \ kind='D'}(film)$$

Ziel

Ersetze die Blätter des Anfrageplans durch spezifische Zugriffs-Methoden, d.h. kann/soll ich einen Index benutzen oder besser einen Sequenziellen-Scan der Datei?

Strategien für konjunktive Anfragen

```
SELECT *
FROM customer
WHERE name = 'Jensen' AND street = 'Elm'
    AND state = 'Arizona'
```

- Können die Indexe auf (name) und (street) benutzt werden?
- Kann der Index auf (name, street, state) benutzt werden?
- Kann der Index auf (name, street) benutzt werden?
- Kann der Index auf (name, street, city) benutzt werden?
- Kann der Index auf (city, name, street) benutzt?

Strategien für konjunktive Anfragen

```
SELECT *
FROM customer
WHERE name = 'Jensen' AND street = 'Elm'
    AND state = 'Arizona'
```

- Können die Indexe auf (name) und (street) benutzt werden? Ja
- Kann der Index auf (name, street, state) benutzt werden? Ja
- Kann der Index auf (name, street) benutzt werden? Ja
- Kann der Index auf (name, street, city) benutzt werden? Ja
- Kann der Index auf (city, name, street) benutzt? Nein

Mehrdimensionale Indexstrukturen

Bislang: Eindimensionale Schlüssel.

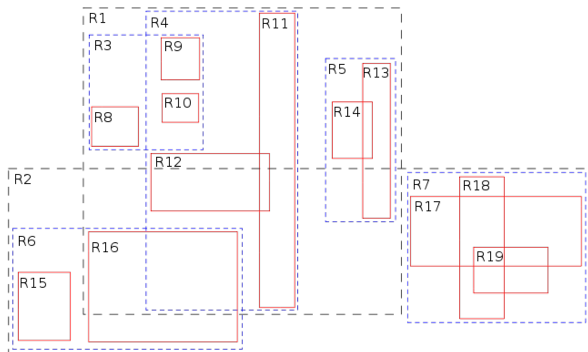
In vielen Anwendungen ist die Anzahl der Dimensionen höher, was tun?

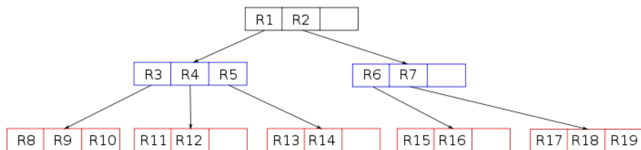
R Baum

- R steht für Rechteck/Rectangle
- Knoten definieren minimale Rechtecke, die die enhaltenen Rechtecke umschließen
- Balanciert. Aufbau (Algorithmus) ähnlich zum B+ Baum
- Überlappung der MBRs (=Minimum Bounding Rectangles)
- Überlappung führt zu Ineffizienz während der Anfrage.

R Baum: Beispiel

Quelle: wikipedia





Nachteile von Indexen

- Platzverbrauch
- Jedes insert/delete/update muss in den Indexen nachgepflegt werden
- Jede Änderungsoperation kostet etwas: CPU und/oder I/O
- Menge der Änderungsoperation kann soviel kosten, dass diese Kosten den Nutzen des Index überwiegen

Grundregel: je mehr Leseoperationen desto mehr lohnen sich Indexe

Genauere Regeln zum "Index Tuning", Anwendbarkeit und weitere Details zur physischen Organisation von Tupeln bzw. Referenzen auf Tupel in Indexen werden in der VL Datenbanksysteme vorgestellt.