5. Eindimensionale Zugriffspfade

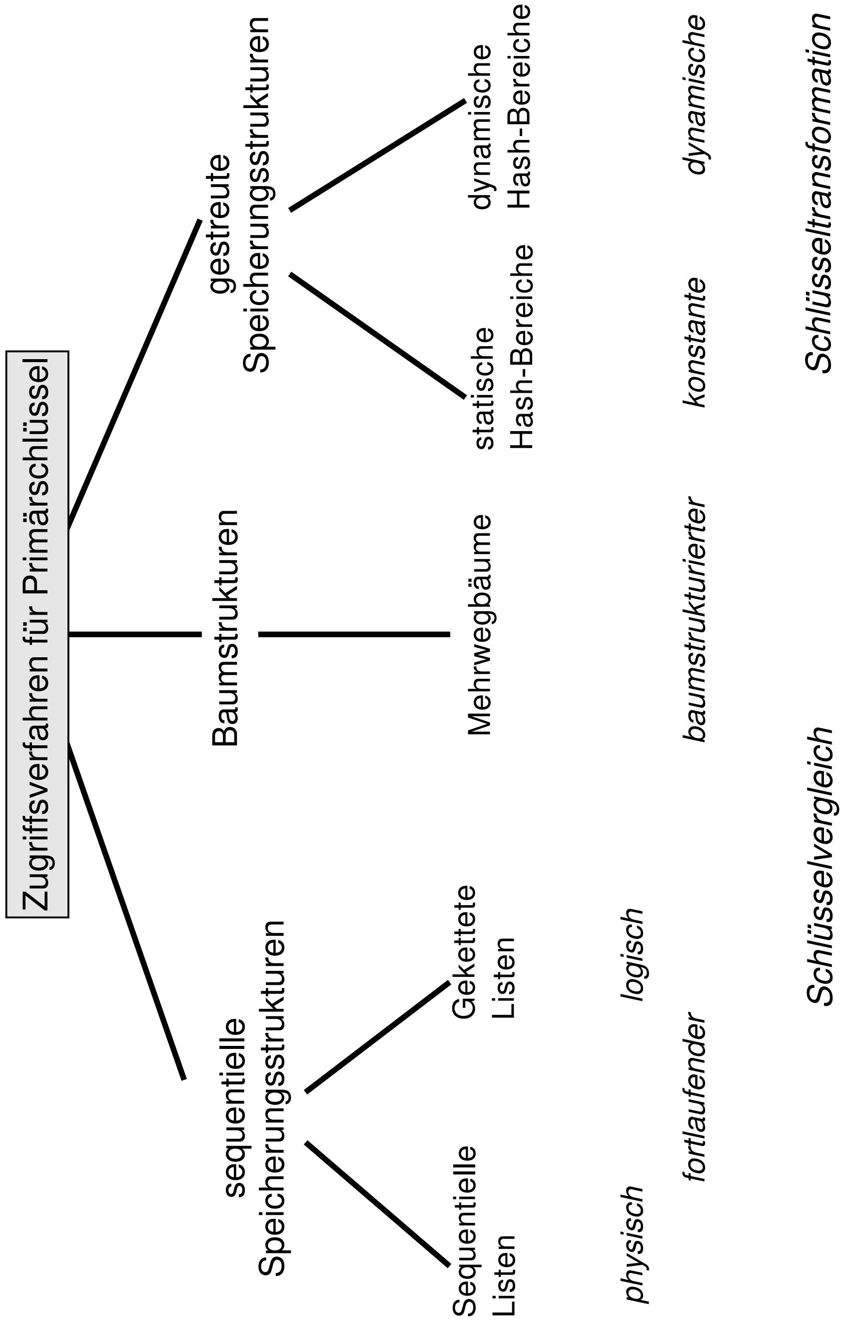
- **Anforderungen**
- **Zugriffspfade für Primärschlüssel**
  - Mehrwegbäume (B- und B*-Bäume)
  - Hash-Verfahren
    - Statische Verfahren
    - Erweiterbares Hashing
- **Zugriff über Sekundärschlüssel**
- **Hierarchische Zugriffspfade**
- **Verallgemeinerte Zugriffspfadstruktur**
- **Wichtige Größen:**
  - n = #Sätze eines Satztyps
  - b = mittlere #Sätze/Seite (Blockungsfaktor)
  - q = #Treffer der Anfrage
  - N_S = #Seitenzugriffe
  - N_B = #Blattseiten des B*-Baums
  - h_B = Höhe des B*-Baums

Anforderungen an Zugriffspfade

- **Folgende Arten von Zugriffen müssen unterstützt werden:**
  - Sequentieller Zugriff auf alle Sätze eines Satztyps (Scan)
  - Sequentieller Zugriff in Sortierreihenfolge eines Attributes
  - Direkter Zugriff über den Primärschlüssel
  - Direkter Zugriff über einen Sekundärschlüssel
  - Direkter Zugriff über zusammengesetzte Schlüssel und komplexe Suchausdrücke (Wertintervalle, ...)
  - Navigierender Zugriff von einem Satz zu einer dazugehörigen Satzmenge desselben oder eines anderen Satztyps

➥ **Wenn kein geeigneter Zugriffspfad vorhanden ist , sind alle Zugriffsarten durch fortlaufende Suche (Scan) abzuwickeln**

- **Scan**
  - wird von allen DBVS unterstützt
  - ist ausreichend / effizient bei:
    - kleinen Satztypen (z. B. ≤ 5 Seiten)
    - Anfragen mit großen Treffermengen (z. B. > 3 %)
- **DBVS kann Prefetching zur Scan-Optimierung nutzen**



Mehrwegbäume

- **Bezugsgröße: Seite = Transporteinheit zum Externspeicher**

  im Gegensatz zu Binärbäumen

- **Vorfahr: ISAM (statisch, periodische Reorganisation)**

- **Weiterentwicklung: B- und B*-Baum**
  - referenzierte und materialisierte Speicherung der Datensätze
  - dynamische Reorganisation durch Splitten und Mischen von Seiten

- **Funktion**
  - direkter Schlüsselzugriff
  - sortiert sequentieller Zugriff

- **Balancierte Struktur**
  - unabhängig von Schlüsselmenge
  - unabhängig von Einfügereihenfolge

- **Realisierung von Index-organisierten Tabellen**
  - oft nach Primärschlüssel geordnet
  - Cluster-Bildung durch eingebettete Datensätze

- **Verbesserung der Baumbreite (fan-out)**
  - Schlüsselkomprimierung
  - Nutzung von „Wegweisern” in B*-Bäumen
  - Präfix-B-Bäume

- **Verbesserung des Belegungsgrades**

  ➥ verallgemeinertes Splittingverfahren

B-Baum

Def.: Ein B-Baum vom Typ (k, h) ist ein Baum mit folgenden Eigenschaften

1. Jeder Weg von der Wurzel zum Blatt hat die Länge h
2. Jeder Zwischenknoten hat mindestens k+1 Söhne.
   Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne
3. Jeder Knoten hat höchstens 2k+1 Söhne

Seitenformat:

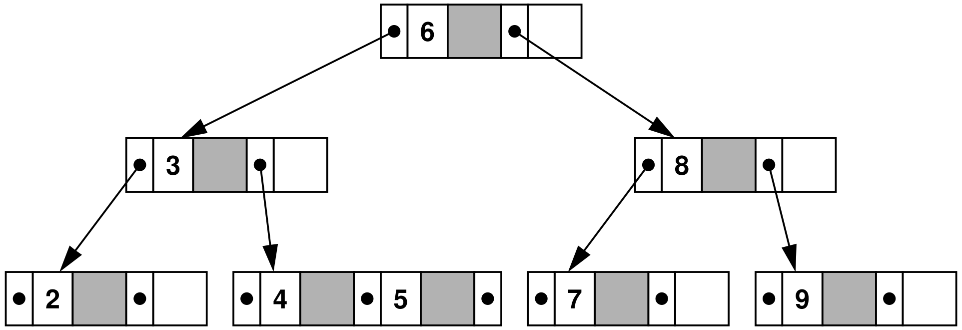
Z_0	S_1	D_1	Z_1	S_2	D_2	Z_2	• • •	S_m	Z_m	D_m	frei

Z_i = Zeiger Sohnseite

S_i = Schlüssel

D_i = Daten des Satzes oder Verweis auf den Satz
(materialisiert oder referenziert)

Beispiel:



bei 4 KB Seiten:

Z=4 B, S=4 B, D=92 B ⇒ 100 B pro Eintrag ⇒ ca. 40 Söhne

Z=4 B, S=4 B, D=4 B ⇒ 12 B pro Eintrag ⇒ ca. 330 Söhne

B*-Baum

Def.: Ein B*-Baum vom Typ (k, k*, h) ist ein Baum mit folgenden Eigenschaften

1. Jeder Weg von der Wurzel zum Blatt hat die Länge h
2. Jeder Zwischenknoten hat mindestens k+1 Söhne.
   Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne.
   Jedes Blatt hat mindestens k* Einträge.
3. Jeder Zwischenknoten hat höchstens 2k+1 Söhne.
   Jedes Blatt hat höchstens 2k* Einträge.

Zwischenknoten:

Z_0	S_1	Z_1	S_2	Z_2	• • •	S_m	Z_m	frei

Z_i = Zeiger Sohnseite

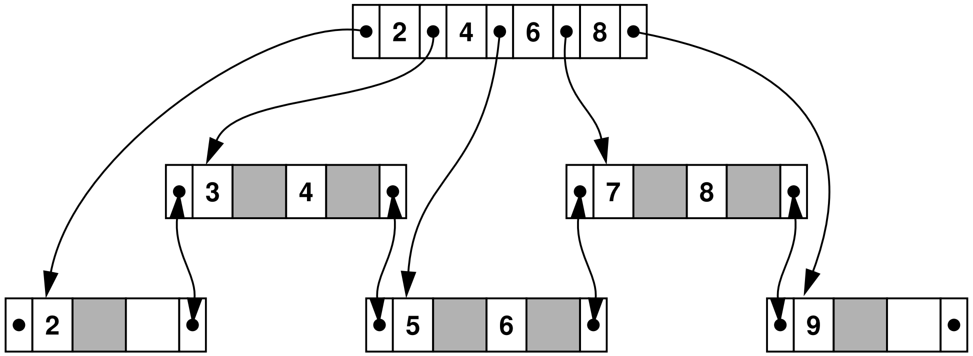
S_i = Schlüssel

Blattknoten:

V	S_1	D_1	S_2	D_2	• • •	S_m	D_m	frei	N

D_i = Verweis auf Satz (materialisiert oder referenziert)

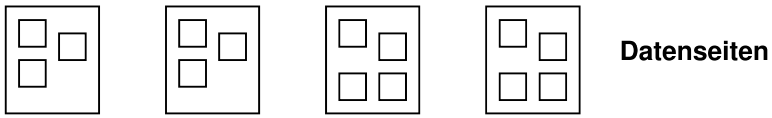
N = Nachfolger-Zeiger, V = Vorgänger-Zeiger



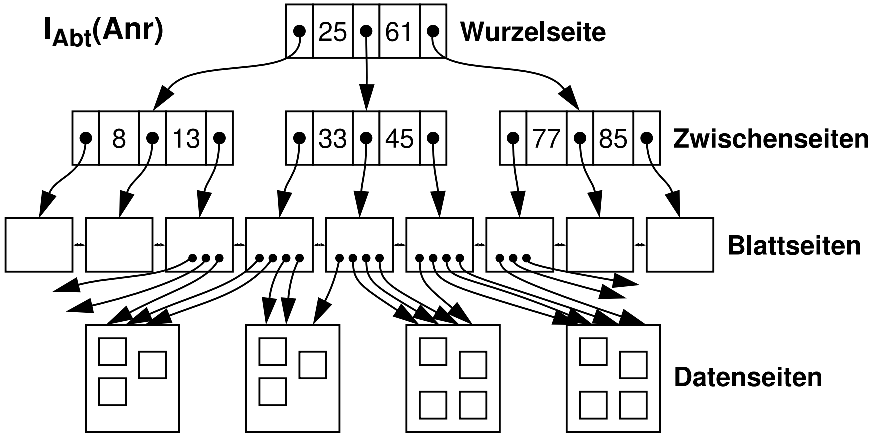
Z=4 B, S=4 B ⇒ 8 B pro Eintrag ⇒ ca. 500 Söhne bei 4 KB Seite

Zugriff zu allen Sätzen eines Satztyps

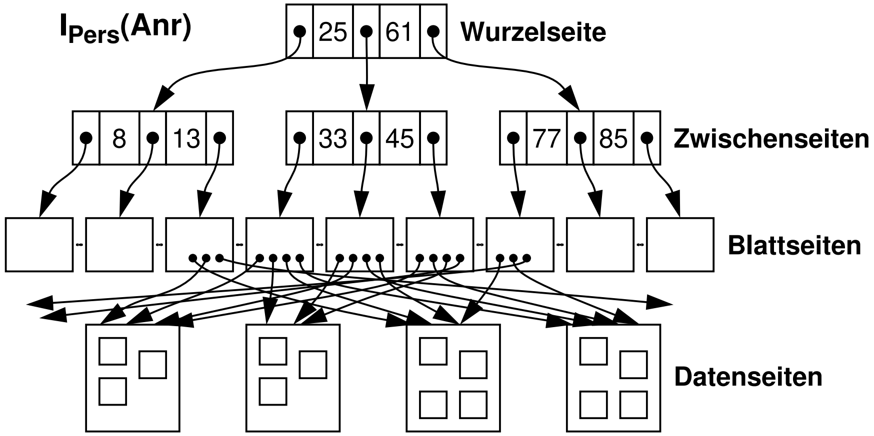
Tabellen-Scan



Index-Scan mit Cluster-Bildung



Index-Scan ohne Cluster-Bildung



Gestreute Speicherungsstrukturen (Hash-Verfahren)

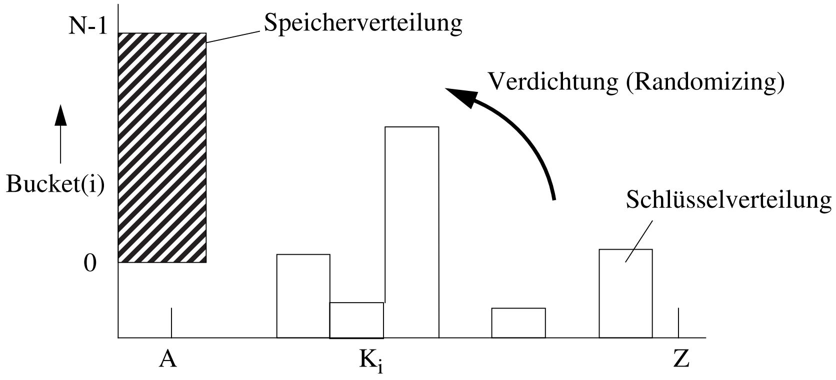
- **Direkte Berechnung** der Satzadresse über Schlüssel (Schlüsseltransformation)

- **Hash-Funktion**

  h: S $\rightarrow$ {0, 1, ..., N-1} S = Schlüsselraum
  N = Größe des statischen Hash-Bereiches in Seiten (***Buckets***)

- **Idealfall: h ist injektiv (keine Kollisionen)**
  - nur in Ausnahmefällen möglich ('dichte' Schlüsselmenge)
  - jeder Satz kann mit einem Seitenzugriff gefunden werden

- **Statische Hash-Bereiche mit Kollisionsbehandlung**
  - vorhandene Schlüsselmenge K (K $\subseteq$ S) soll möglichst gleichmäßig auf die N Buckets verteilt werden



  - Behandlung von Synonymen:
    - Aufnahme im selben Bucket, wenn möglich
    - ggf. Anlegen und Verketten von Überlaufseiten
  - typischer Zugriffsfaktor: 1.1 bis 1.4

- **Vielzahl von Hash-Funktionen anwendbar**

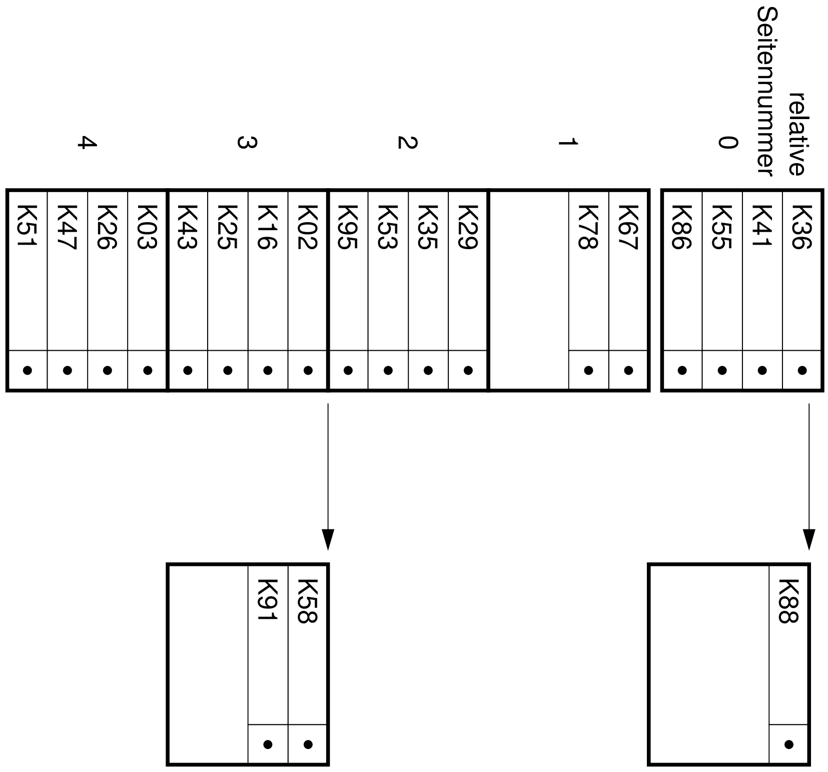
  z. B. Divisionsrestverfahren, Faltung, Codierungsmethode, ...

Statisches Hash-Verfahren mit Überlaufbereichen: Beispiel

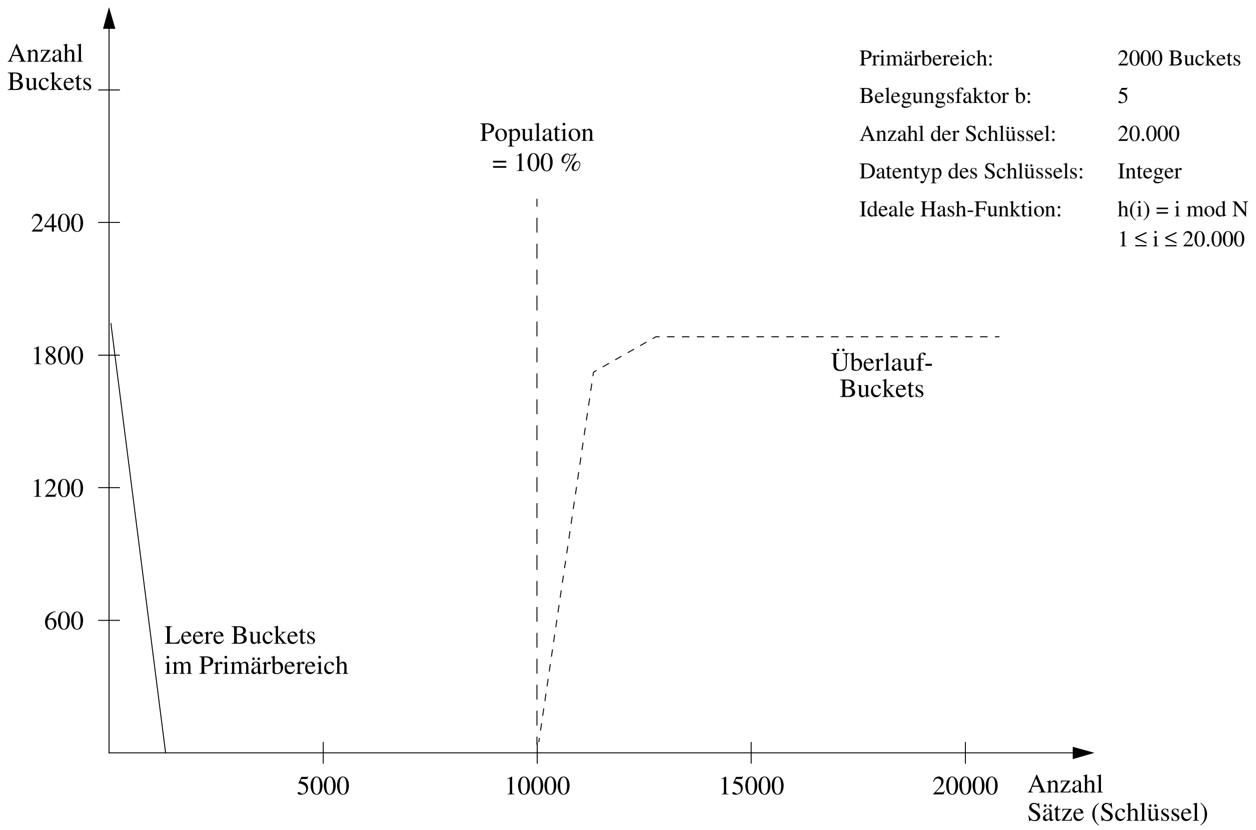
Adreßberechnung für Schlüssel K02:

$$
\begin{array}{r}
1101\ 0010 \\
\oplus\ 1111\ 0000 \\
\oplus\ 1111\ 0010 \\
\hline
1101\ 0000 = 208_{10}
\end{array}
$$

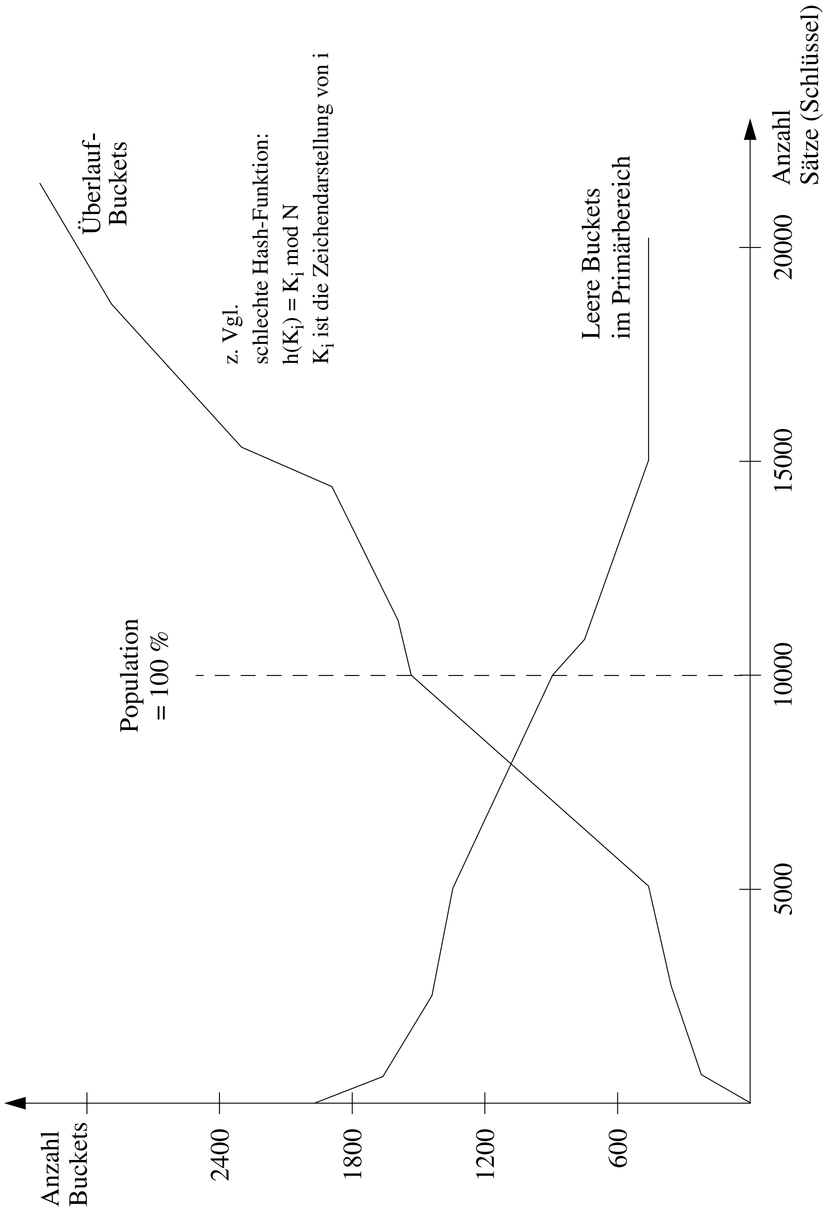
$208 \bmod 5 = 3$



Belegung von Hash-Bereichen – Messung (1)



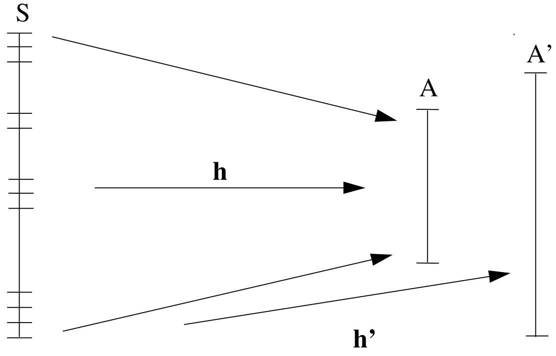
Belegung von Hash-Bereichen – Messung (2)



Dynamische Hash-Verfahren

- **Wachstumsproblem bei statischen Verfahren**
  - Statische Allokation von Speicherbereichen: Speicherausnutzung?
  - Bei Erweiterung des Adreßraumes: Rehashing

  ➥ Kosten, Verfügbarkeit, Adressierbarkeit



  ➥ Alle Sätze erhalten eine **neue Adresse**

- **Entwurfsziele**
  - Dynamische Struktur erlaubt Wachstum und Schrumpfung des Hash-Bereichs (Datei)
  - Keine Überlauftechniken
  - Zugriffsfaktor ≤ 2 für die direkte Suche

- **Viele konkurrierende Ansätze**
  - Extendible Hashing (Fagin et al., 1978)
  - Virtual Hashing und Linear Hashing (Litwin, 1978, 1980)
  - Dynamic Hashing (Larson, 1978)

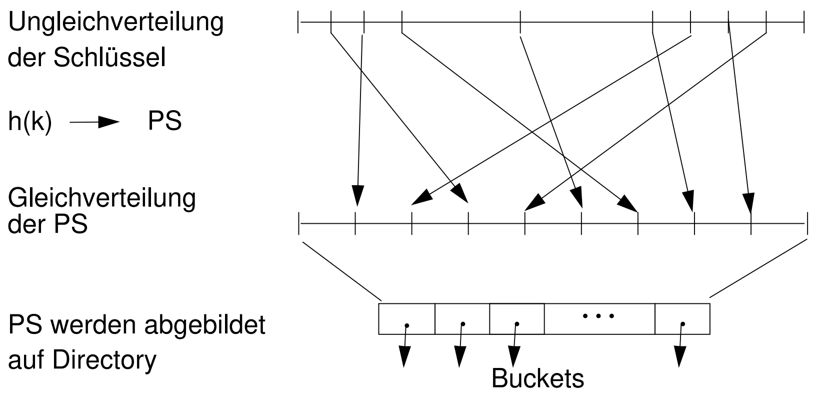
  ➥ Lösungsvorschläge **mit und ohne Index** (Hilfsdaten)
  „Any hashing which may dynamically change its hashing function”.

Erweiterbares Hashing

- **Lösungsidee: Verknüpfung der**
  - von den B-Bäumen bekannten Split- und Mischtechniken von Seiten zur Konstruktion eines dynamischen Hash-Bereichs mit der
  - von den Digitalbäumen her bekannten Adressierungstechnik zum Aufsuchen eines Speicherplatzes

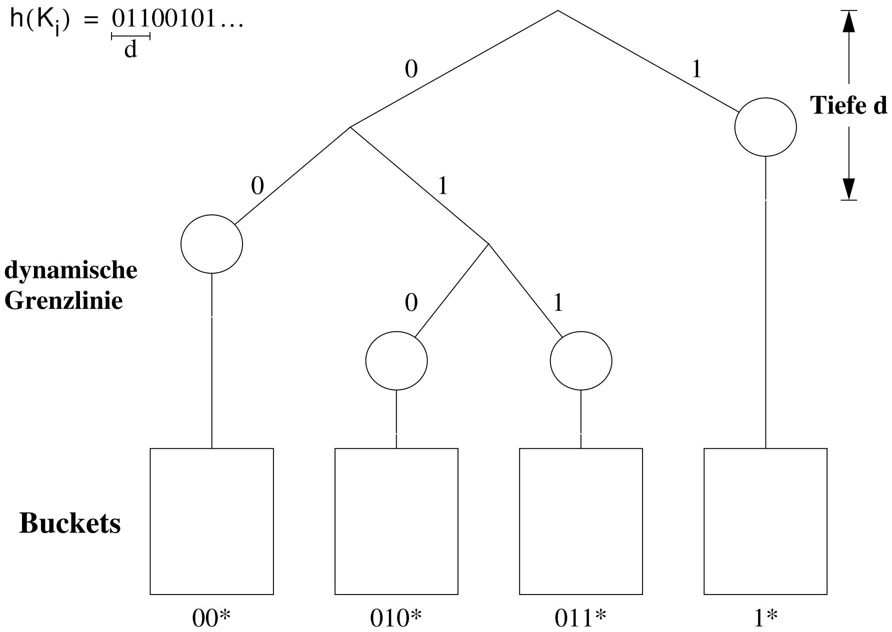
- **Prinzipielle Vorgehensweise**
  - Die einzelnen Bits eines Schlüssels steuern den Weg durch den zur Adressierung benutzten Digitalbaum.
  - $K_i = (b_0, b_1, b_2, \ldots)$. Es ist prinzipiell möglich, die Bitfolge von K_i direkt für die Adressierung heranzuziehen. Bei Ungleichverteilung der Schlüssel ist dann ein unausgewogener Digitalbaum zu erwarten.
  - Da Digitalbäume keinen Balancierungsmechanismus für ihre Höhe besitzen, muß die Ausgewogenheit “von außen” aufgezwungen werden.
  - $h(K_i) = (b_0, b_1, b_2, \ldots)$. Die Verwendung von $h(K_i)$ als sog. Pseudoschlüssel (PS) soll bessere Gleichverteilung gewährleisten.

- **Gleichverteilung der Pseudoschlüssel PS**
  impliziert minimale Höhe des Digitalbaumes



Erweiterbares Hashing[1] (3)

- **Prinzipielle Abbildung der Pseudoschlüssel**



  - Zur Adressierung eines Buckets sind d Bits erforderlich, wobei sich dafür i. allg. eine dynamische Grenzlinie variierender Tiefe ergibt.
  - Die Digitalbaum-Adressierung bricht ab, sobald ein Bucket den ganzen Teilbaum aufnehmen kann.
  - ➥ ausgeglichener Digitalbaum garantiert minimales d_{max}

- **Dynamisches Wachsen und Schrumpfen des Hash-Bereiches**
  - Buckets werden erst bei Bedarf bereitgestellt
  - Knoten unterschiedlicher Tiefe verweisen auf ein Bucket
  - ➥ **hohe Speicherplatzbelegung möglich**

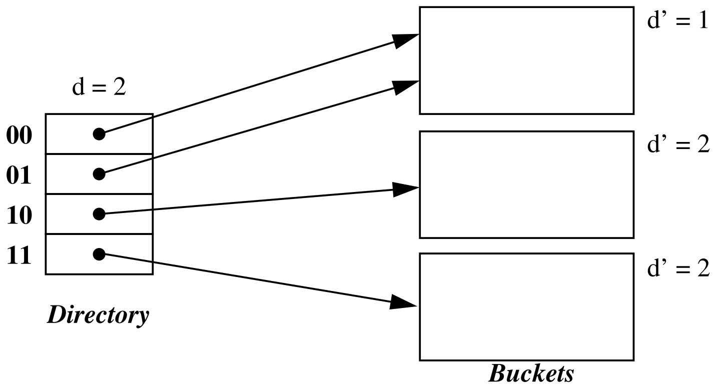
1. Fagin, R., et. al: Extendible hashing - a fast access method for dynamic files. ACM Trans. Database Syst. 4:3. 1979. 315-344

Erweiterbares Hashing (4)

- **Verfahren benötigt keine Überlaufbereiche, jedoch erfolgt Zugriff über *Directory* (Index)**
  - Binärer Digitalbaum der Höhe d wird durch einen (2^d)-Digitalbaum der Höhe 1 implementiert (Trie der Höhe 1 mit 2^d Einträgen)
  - d wird festgelegt durch den längsten Pfad im binären Digitalbaum
  - In einem Bucket werden nur Sätze gespeichert, deren PS in den ersten d' Bits übereinstimmen (d' = lokale Tiefe)
  - d = MAX (d'): d Bits des PS werden zur Adressierung verwendet (d = globale Tiefe)
  - Directory enthält 2^d Einträge

- **Speicherungsstruktur**

  Der Trie läßt sich als Directory oder Adreßverzeichnis auffassen. Die d Bits von $h(K_i)$ zeigen im Directory auf einen Eintrag mit der Adresse des Buckets, das den Schlüssel K_i enthält. Wenn d' < d, können (benachbarte) Einträge auf dasselbe Bucket verweisen.

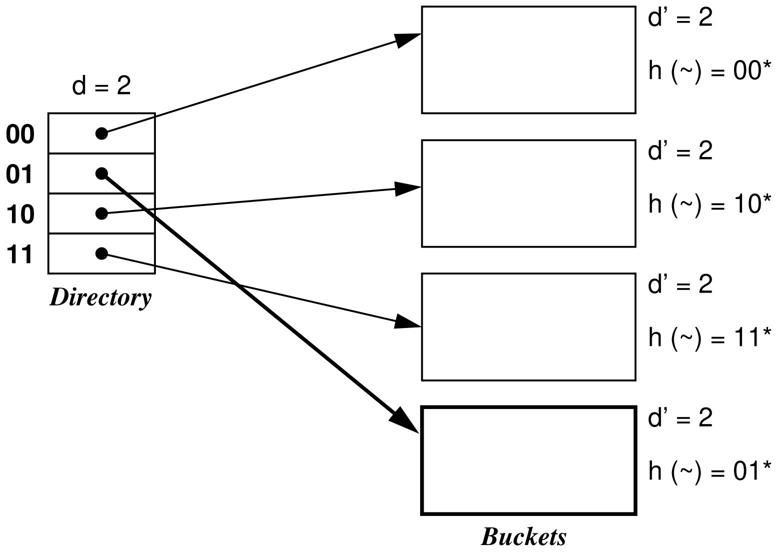


➥ Kosten der direkten Suche: **max. 2 Seitenzugriffe**

Erweiterbares Hashing: Splitting von Buckets (1)

- **Fall 1: Überlauf eines Buckets, dessen lokale Tiefe kleiner als die globale Tiefe d ist**

➥ Anlegen eines neuen Buckets (Split) mit
  - lokaler Neuverteilung der Daten
  - Erhöhung der lokalen Tiefe
  - lokaler Korrektur der Verweise im Directory

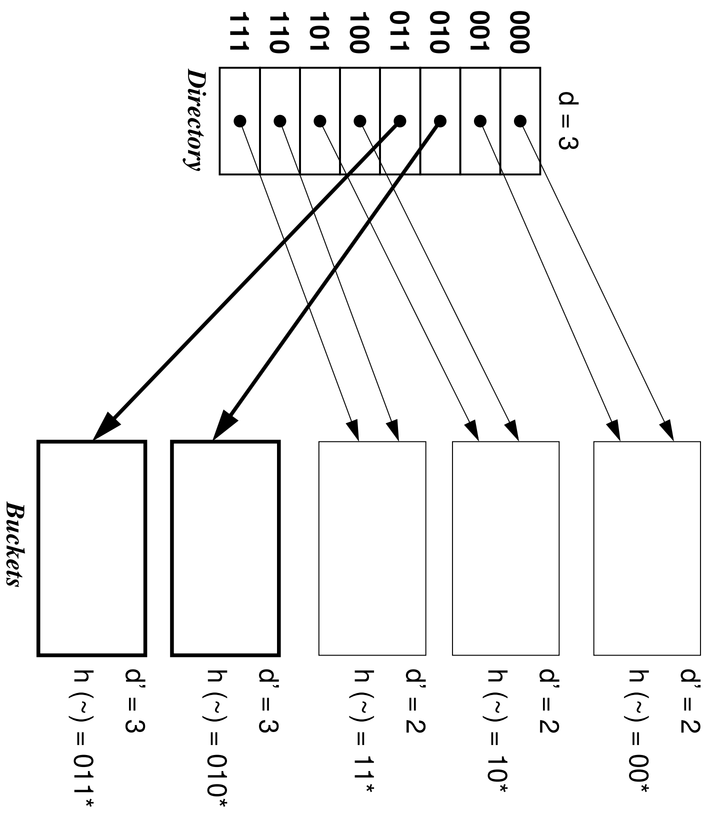


Erweiterbares Hashing: Splitting von Buckets (2)

- **Fall 2: Überlauf eines Buckets, dessen lokale Tiefe gleich der globalen Tiefe ist**

➡ Anlegen eines neuen Buckets (Split) mit

- lokaler Neuverteilung der Daten (Erhöhung der lokalen Tiefe)
- Verdopplung des Directories (Erhöhung der globalen Tiefe)
- globaler Korrektur/Neuverteilung der Verweise im Directory



Vergleich der wichtigsten Zugriffsverfahren

Zugriffsverfahren	Speicherungsstruktur	Direkter Zugriff	Sequentielle Verarbeitung	Änderungsdienst (Ändern ohne Aufsuchen)
Fortlaufender Schlüsselvergleich	Sequentielle Liste	O (n) ≈ 10^4	O (n) ≈ 2 · 10^4	O (1) ≤ 2
	Gekettete Liste	O (n) ≈ 5 · 10^5	O (n) ≈ 10^6	O (1) ≤ 3
Baumstrukturierter Schlüsselvergleich	Balancierte Binärbäume	O ($\log_2$ n) ≈ 20	O (n) ≈ 10^6	O (1) = 2
	Mehrwegbäume	O ($\log_k$ n) ≈ 3 - 4	O (n) ≈ 10^6*	O (1) = 2
Konstante Schlüsseltrans-formationsverfahren	Externes Hashing mit separatem Überlaufbereich	O (1) ≈ 1.1 - 1.4	O (n $\log_2$ n)**	O (1) ≈ 1.1
	Externes Hashing mit Separatoren	O (1) = 1	O (n $\log_2$ n)**	O (1) = 1 (+D)
Variable Schlüsseltrans-formationsverfahren	Erweiterbares Hashing	O (1) = 2	O (n $\log_2$ n)**	O (1) ≈ 1.1 (+R)
	Lineares Hashing	O (1) = 1	O (n $\log_2$ n)**	O (1) < 2

* Bei Clusterbildung bis zu Faktor 50 geringer

** Physisch sequentielles Lesen, Sortieren und sequentielles Verarbeiten der gesamten Sätze

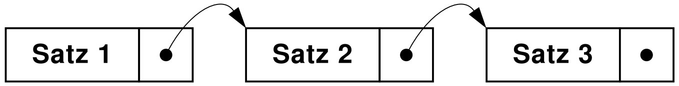
Beispielangaben für n = 10^6

Verknüpfungsstrukturen für Satzmengen

- **Materialisierte Speicherung**

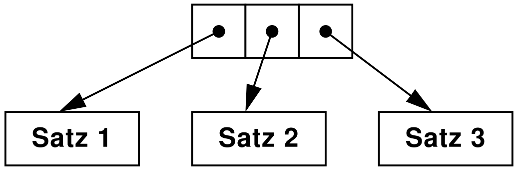
1. Physische Nachbarschaft der Sätze (Cluster-Bildung, Listen)

2. Verkettung der Sätze

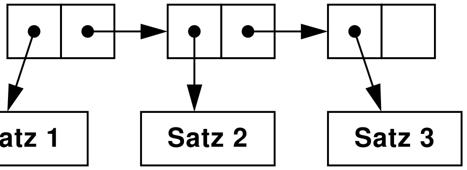


- **Referenzierte Speicherung**

3. Physische Nachbarschaft der Zeiger (Invertierung)



4. Verkettung der Zeiger



Zugriffspfade für Sekundärschlüssel

- **Suche nach Sätzen mit einem vorgegebenen Wert in einem nicht-identifizierenden Attribut (*Sekundärschlüssel*)**

- **Ergebnis ist Satzmenge**

Anr
A02
A03
A12
• • •

Pnr	Anr	W-Ort	Gehalt
12345	A02	KL	45000
23456	A02	MA	51000
34567	A03	KL	48000
45678	A02	MA	55000
56789	A03	F	65000
67890	A12	KL	50000

W-Ort
KL
MA
F
• • •

- **Realisierung: Einstiegsstruktur + Verknüpfungsstruktur**
  - Primärschlüssel-Zugriffspfade als Einstiegsstruktur auf Satzmengen anwendbar
  - Prinzipiell lassen sich alle Verknüpfungsstrukturen für Satzmengen heranziehen

- **Aber:**
  Referenzierte Speicherung erlaubt effiziente Unterstützung mengenalgebraischer Operationen

➥ Verwendung von Invertierungstechniken bzw. B*-Bäumen

Zugriffspfade für Sekundärschlüssel (2)

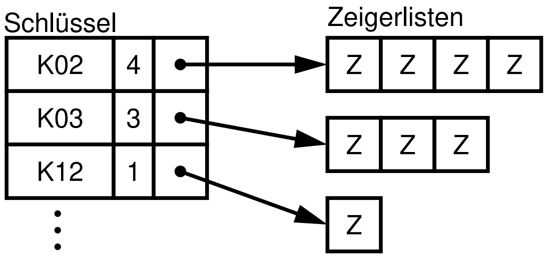
- **Häufige Realisierung: Invertierung**
  - Trennung der Zugriffspfaddaten von den Datensätzen (referenzierte Speicherung)
  - Verweis Z realisiert als TID, DBK/PPP, ...
  - Es kommen zwei Darstellungsmethoden in Betracht:

a) Gemeinsame Verwaltung der Suchstruktur und der Zeigerlisten

Schlüssel		Zeigerlisten			
K02	4	Z	Z	Z	Z
K03	3	Z	Z	Z	
K12	1	Z			
⋮					

➥ relativ kurze Zeigerlisten erforderlich!

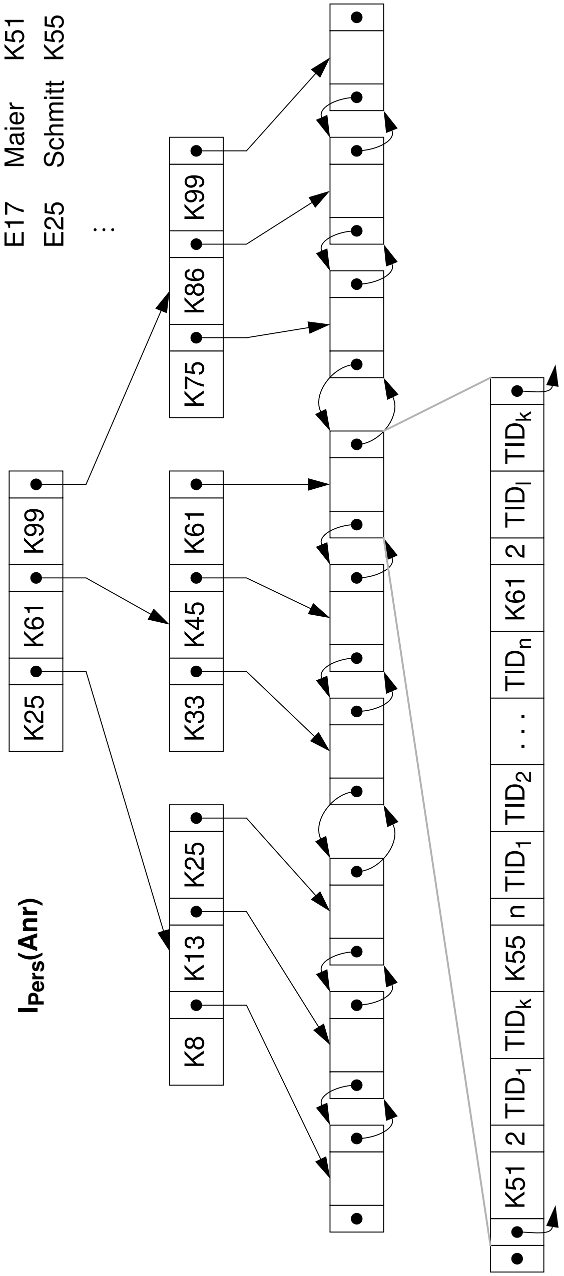
b) In der Suchstruktur ist (ähnlich wie bei Zugriffspfaden für Primärschlüssel) nur ein Verweis pro Schlüsselwert vorhanden, der zu einer Liste mit Satzverweisen führt (Zeigerliste)



➥ Zeigerlisten können in separaten „Behältern" abgelegt werden

Tabelle Pers

Pers (Pnr,	Name,	Anr,	...)
E1	Müller	K55	...
E17	Maier	K51	
E25	Schmitt	K55	
...			



B*-Baum

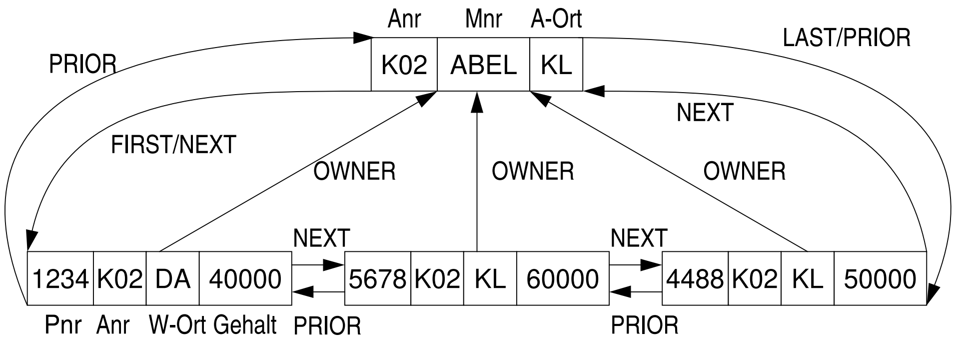
- als Zugriffspfad für Sekundärschlüssel Anr
- mit Sortierreihenfolge der Sekundärschlüssel (Bereichsfragen!) sowie Vorwärts- und Rückwärtsverkettung

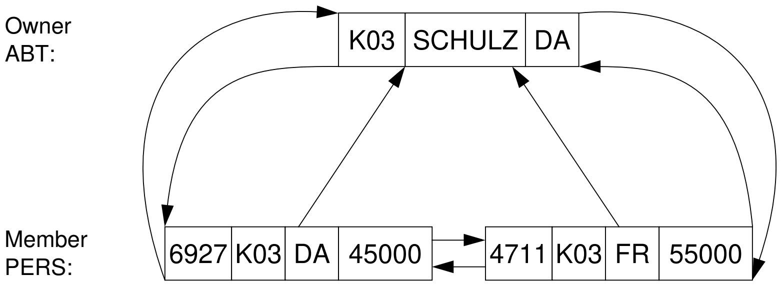
Hierarchische Zugriffspfade

- **Realisierung funktionaler Beziehungen zwischen zwei Satztypen**
  (Set-Typen nach dem Netzwerkmodell)

- **Jede Ausprägung einer Owner-Satzart wird mit 0..n Ausprägungen der Member-Satzart verknüpft**

Logische Sicht:
Darstellung der Navigationsmöglichkeiten

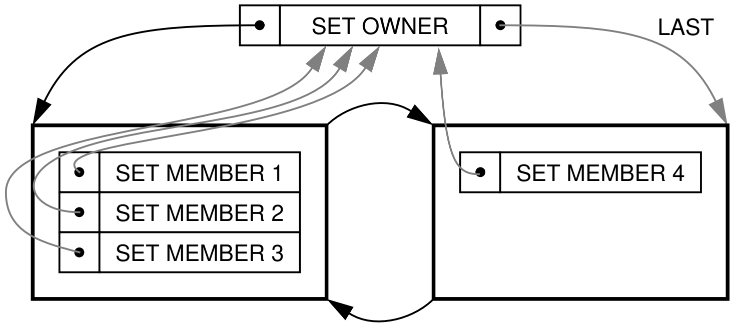




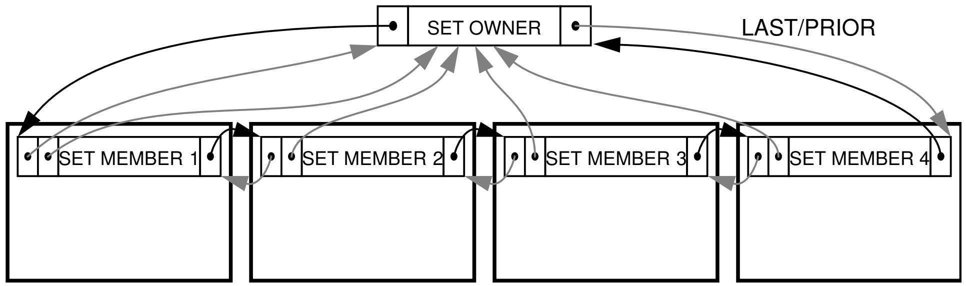
➥ drei Implementierungen für unterschiedliche Leistungsanforderungen

Hierarchische Zugriffspfade - Implementierung

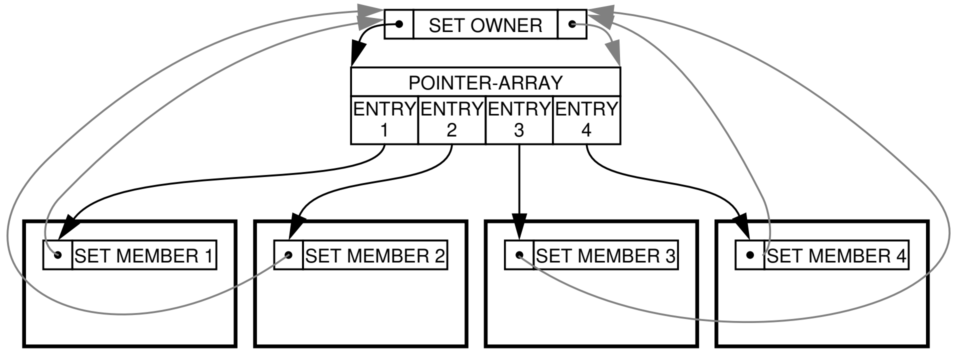
- **Sequentielle Liste auf Seitenbasis**



- **Gekettete Liste**



- **Pointer-Array-Struktur**



⟶ : optionaler Zeiger

Hierarchische Zugriffspfade - Bewertung der Implementierungstechniken

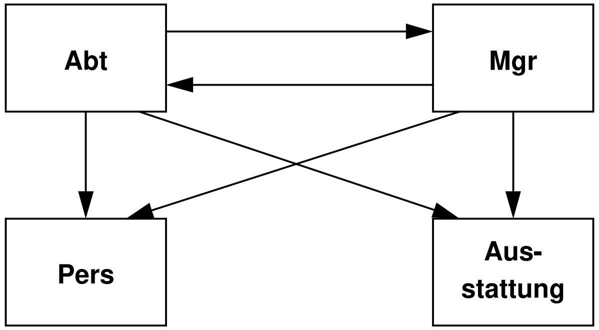
- **Pointer-Array**
  - stabiles Verhalten
  - Verhalten unabhängig vom Set-Wachstum und Set-Reihenfolge
  - ’Standard’-Verfahren bei unscharfen Informationen über Set-Größe und Zugriffshäufigkeit

- **Sequentielle Liste**
  - auf einen Set-Typ pro Member-Satztyp beschränkt (Cluster-Bildung)
  - schnelles Aufsuchen / Einfügen in Set-Reihenfolge
  - Ändern teurer als bei Pointer-Array

- **Gekettete Liste**
  - Vorteile bei Mitgliedschaft des Member-Satztyps in mehreren Sets
  - billiger Wechsel auf andere Set-Ausprägungen
  - sequentieller Zugriff schneller als bei Pointer-Array
  - nur gut in kleinen Set-Ausprägungen

Verallgemeinerte Zugriffspfadstruktur

- **Idee:**
  gemeinsame Verwendung einer Indexstruktur (B*-Baum) für mehrere Satztypen, für die Beziehungen (1:1, 1:n, n:m) über demselben Wertebereich (z. B. für Anr) definiert und durch Gleichheit von Attributwerten repräsentiert sind

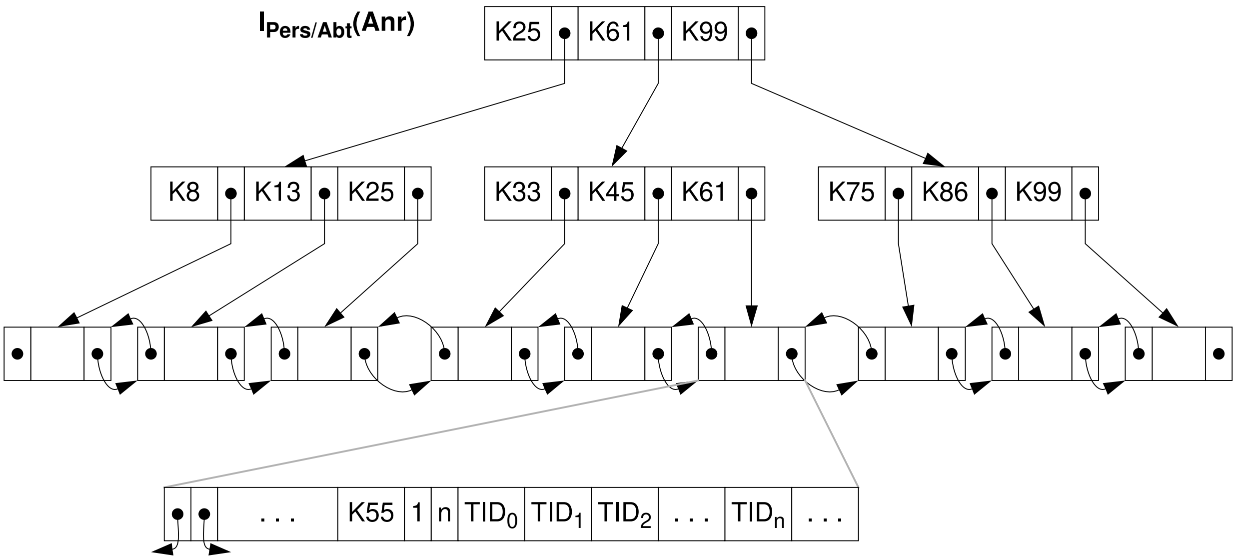


Alle Tabellen besitzen ein Attribut (z. B. Anr), das auf dem Wertebereich Abtnr definiert ist

- **Nutzung der Indexstruktur** für
  - Primärschlüsselzugriff z. B. als I_{Abt}(Anr)
  - Sekundärschlüsselzugriff z. B. als I_{Pers}(Anr)
  - hierarchischen Zugriff z. B. von Abt(Anr) nach Pers(Anr) oder in umgekehrter Richtung
  - Verbundoperationen (Join) z. B. von Abt.Anr = Pers.Anr

- **Kombinierte Realisierung** von Primärschlüssel-, Sekundärschlüssel- und hierarchischen Zugriffspfaden mit einem erweiterten B*-Baum
  - Innere Baumknoten bleiben unverändert
  - Blätter enthalten Verweise für primäre und sekundäre Zugriffspfade

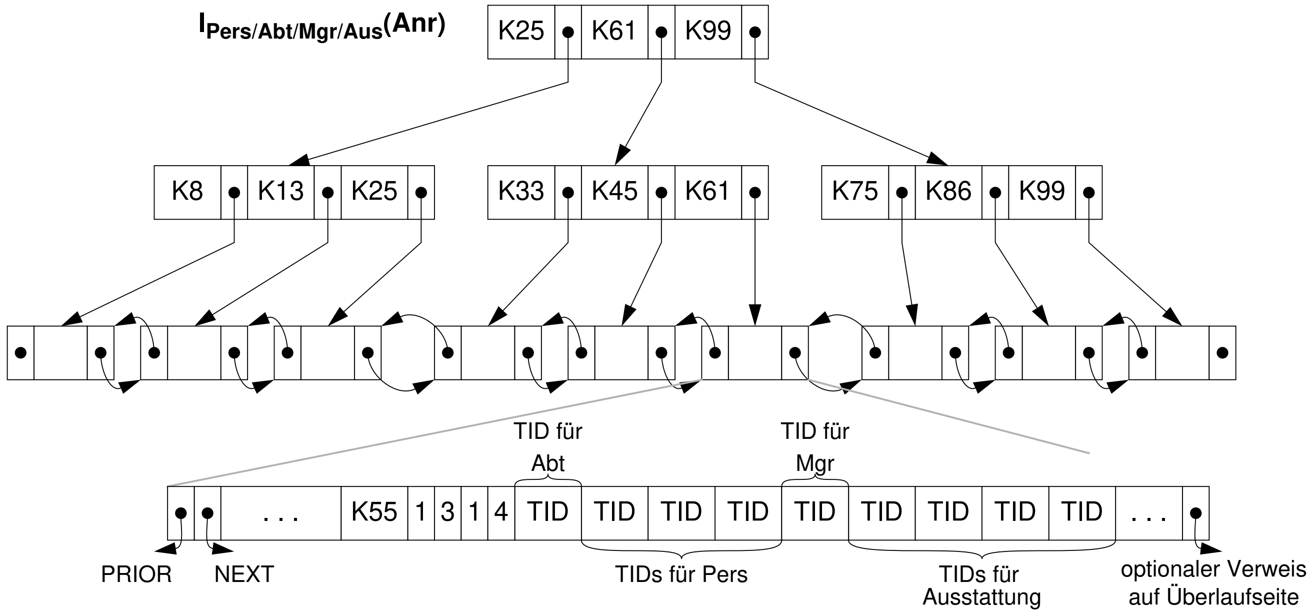
B*-Baum als kombinierte Zugriffspfadstruktur



Struktur enthält Index für Abt, Pers und
Link für Abt-Pers mit direktem Zugriff von

1. OWNER zu jedem MEMBER,
2. jedem MEMBER zu jedem anderen MEMBER,
3. jedem MEMBER zum OWNER

B*-Baum als verallgemeinerte Zugriffspfadstruktur



Zugriffspfadstruktur umfaßt

4 Index-Strukturen

6 Link-Strukturen

Verallgemeinerte Zugriffspfadstruktur: Bewertung

- **Schlüssel werden nur einmal gespeichert:**

  ➥ Speicherplatzersparnis

- **Einheitliche Struktur für alle Zugriffspfadtypen:**

  ➥ Vereinfachung der Implementierung

- **Unterstützung der Join-Operation sowie bestimmter statistischer Anfragen**

- **Einfache Überprüfung der referentiellen Integrität sowie weiterer Integritätsbedingungen**

  (z. B. Kardinalitätsrestriktionen)

- **Erhöhung der Anzahl der Blattseiten:**

  ➥ mehr Seitenzugriffe beim sequentiellen Lesen aller Sätze eines Satztyps in Sortierordnung

- **Höhe des Baumes bleibt meist erhalten:**

  ➥ ähnliches Leistungsverhalten beim Aufsuchen von Daten und beim Änderungsdienst

Physischer DB-Entwurf

- **Physischer DB-Entwurf**
  - Sorgfältige Wahl der wichtigen Attribute (Spalten), nach deren Werte Cluster-Bildung vorgenommen wird
  - Oft werden die Primärschlüsselattribute oder OWNER-MEMBER-Beziehungen für die Cluster-Bildung ausgewählt
  - Geeignete Wahl des Belegungsgrades beim Laden von Tabellen mit Cluster-Eigenschaft
  - Welche Attribute sollen indexiert werden?

- **Indexierungsheuristik**
  Indexstrukturen werden angelegt
  - auf allen Primär- und Fremdschlüsselattributen, was auch mit verallgemeinerten Zugriffspfadstrukturen erreicht werden kann
  - auf Attributen vom Typ DATE
  - auf Attributen, die in (häufigen) Anfragen in Gleichheits- oder IN-Prädikaten vorkommen

  ➥ Primär- und Fremdschlüsselindexierung erlaubt Navigation durch mehrere Tabellen ausschließlich mit Hilfe der Indexstrukturen

- **Alternative Indexierungsheuristik**
  - Indexstrukturen werden auf Primärschlüssel- und (möglicherweise) auf Fremdschlüsselattributen angelegt
  - Zusätzliche Indexstrukturen werden nur angelegt, wenn für eine aktuelle Anfrage der neue Index zehnmal weniger Sätze liefert als irgendein existierender Index

  ➥ Beide Heuristiken liefern fast die gleichen Indexstrukturen für die meisten Datenbanken und Arbeitslasten

Zugriffspfade in kommerziellen Datenbanksystemen

DB2 (IBM)	B*-Baum (clustered, non-clustered), partitionierte Tabellen, ...
Informix	B-Baum, statisches Hashing, ISAM, HEAP, ...
Oracle	B*-Baum (mit Präfix-/Suffix-Komprimierung), (Join-) Cluster-Bildung, ...
Sybase	B*-Baum (clustered, non-clustered), ...
RDB (DEC)	B*-Baum (clustered, non-clustered), Hashing, Join-Cluster-Bildung, ...
NonStop SQL (Tandem)	B*-Baum (clustered, non-clustered) mit Präfix-Komprimierung, ...
UDS (Siemens)	B*-Baum, statisches Hashing, Cluster-Bildung (LIST), Invertierung (Pointer-Array), Kettung (CHAIN)

Zusammenfassung

- **Cluster-Bildung optimiert (sortiert) sequentielle Zugriffe**
- **Standard-Zugriffspfadstruktur: B*-Baum** (the ubiquitous B*-tree)
  - materialisierte und referenzierte Speicherung der Datensätze
  - Index-organisierte Tabelle mit Cluster-Bildung
- **Schnellerer Schlüsselzugriff erfordert Hash-Verfahren**
  - (nur) direkter Zugriff (= 1 Seitenzugriff)
  - Erweiterbares Hashing unterstützt stark wachsende Datenbestände (≤ 2 Seitenzugriffe)
- **Zugriffspfade für Sekundärschlüssel**
  - Einstiegsstruktur: B*-Baum u.a.
  - Verknüpfungsstruktur: Zeigerlisten, Bitlisten

    ➥ Unterstützung mengentheoretischer Operationen
- **Hierarchische Zugriffspfade**
  - Unterstützung von Verbund-Operationen (Relationenmodell)
  - effiziente Abwicklung von Set-Operationen (NW-Modell)
  - Verknüpfungsstruktur: Ketten, Zeigerlisten, Listen
  - vielfältige Abstimmungsmöglichkeiten auf spezielle Arbeitslasten
- **Verallgemeinerte Zugriffspfadstruktur**
  - auch Join-Index genannt
  - Unterstützung von Primärschlüssel-, Sekundärschlüssel- und hierarchischen Zugriffen