Teil III

Von Datenbanken und ihren Systemen

Robert Hartmann (SoSe 2024)

**basierend auf Folien von
Prof. Dr. Harm Knolle**

**Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg**

- Kapitel 8 - Speicherstrukturen -

Inhalt

0 - Vorbemerkungen
Teil I - Von EDV-Anwendungen und Ihren Anforderungen
1 - Einführung
Teil II - Von Daten und ihren Modellen
2 - Prozess des Datenbankentwurfs
3 - Semantische Datenmodelle
4 - Logische Datenmodelle
5 - Datenbankmodelle
6 - Datenanfrage und Datenänderung
Teil III - Von Datenbanken und ihren Systemen
7 - Datenbanksysteme
8 - Speicherstrukturen
9 - Ausblick

Inhalt

- Physische Entwurf der Datenbank
- Interne Datensätze
- Zugriffsverfahren für Primärschlüssel

- Index-Sequentielle Organisation -

Inhalt

- Physische Entwurf der Datenbank
- Interne Datensätze
- Zugriffsverfahren für Primärschlüssel
  - Primär- und Sekundärschlüssel
  - Sequentielle Organisation
  - **Index-Sequentielle Organisation**
  - Gestreute Organisation
  - Sortiert-Logisch-Sequentielle Organisation
  - Baumverfahren
  - Diskussion der Verfahren

Überblick

- Definition Index
- Eigenschaften von Indexen
- Vorgehen der Suche mit Indexen
- Probleme beim Einfügen von Datensätzen
- Überlauf und Reorganisation

Index-Sequentielle Organisation

- Definition Index -

Organisation

- die Datensätze werden in Datenblöcken fester Größe abgelegt (z.B. 8 KB)
- die Datensätze werden wie bei der sequentiellen Organisation nach dem Primärschlüssel aufsteigend abgespeichert
- reicht ein Datenblock zur Speicherung der Daten nicht mehr auf, dann werden weitere Datenblöcke benötigt und sequentiell verkettet
- werden sehr viele Datenblöcke benötigt, so muss bei der Suche nach einem Datensatz
  - zuerst der Datenblock gefunden werden
  - und in dem Datenblock dann der Datensatz
- zur Unterstützung der Suche wird zusätzlich eine Art Inhaltsverzeichnis (ein Index) aufgebaut
- gesucht wird nun nicht in den Datenblöcken
  - sondern erst im Index
  - und dann in den Datensätzen

Index

- ein Index ist eine Ansammlung von Datensätzen eines neuen Typs, die ausschließlich für die internen Zwecke der Datenverwaltung verwendet werden
- der Index wird in einem oder mehreren Blöcken verwaltet

Struktur des Index

- jeder Satz des Index ist vom Typ
  - (s,b) mit
  - s einem Primärschlüssel und
  - b einem Blockidentifikator
- jeder Index-Block enthält eine Vielzahl (s,b)-Index-Einträgen
  - Blockgröße z.B. 8 KB
  - s: z.B. 2 Bytes, b: z.B. 8 Bytes
  - Block enthält ca. 800 Index-Einträge

Index-Sequentielle Organisation

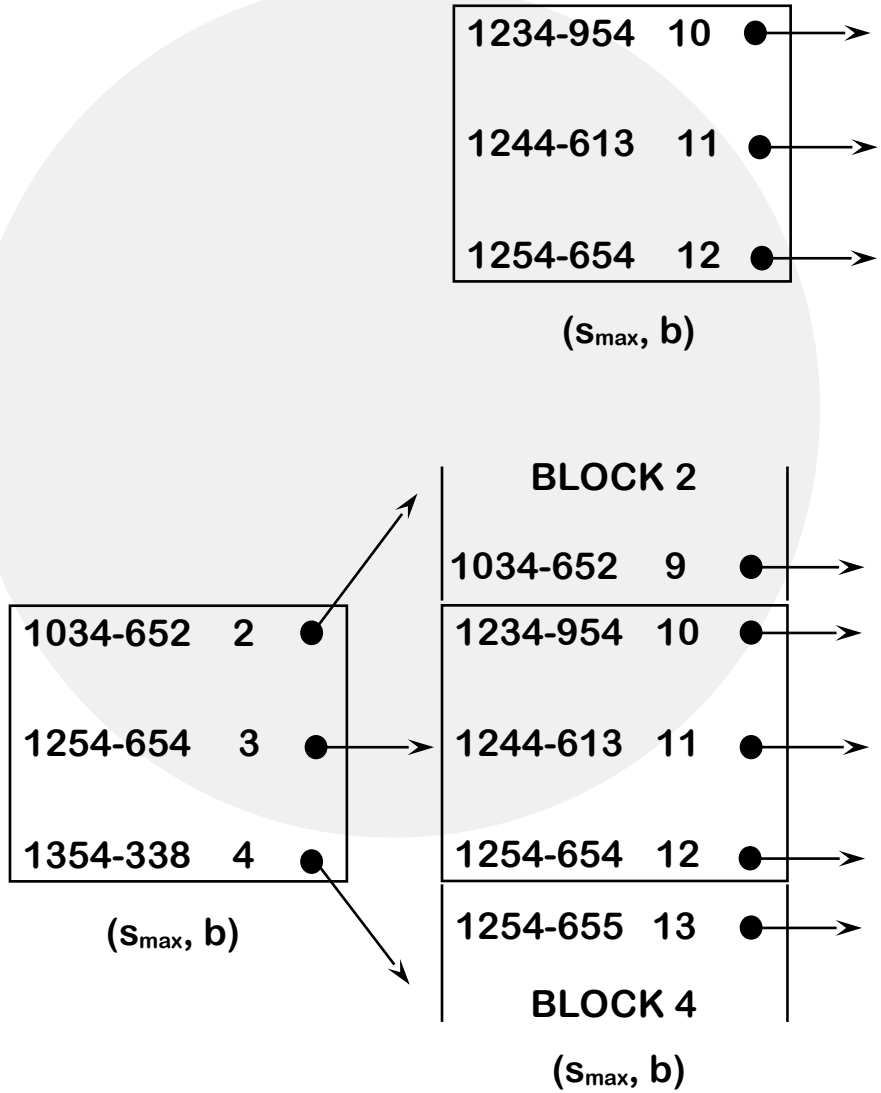
- Eigenschaften von Indexen -

Dichte

- Indexe sind in der Regel nicht dicht, d.h. sie enthalten nicht alle Schlüsselwerte des Primärschlüssels
- Mögliche Indexeinträge sind:
  - (s_{max},b) oder (s_{min},b)
  - mit s_{max}
    = größter Schlüsselwert in Block b oder
  - mit s_{min}
    = kleinster Schlüsselwert in Block b

Mehrstufigkeit

- Indexe können kaskadiert werden
  - … d.h. bei großen Indexen, die über mehrere Blöcke gehen, wird ein Index für den Index aufgebaut
- diese Technik ist beliebig wiederholbar



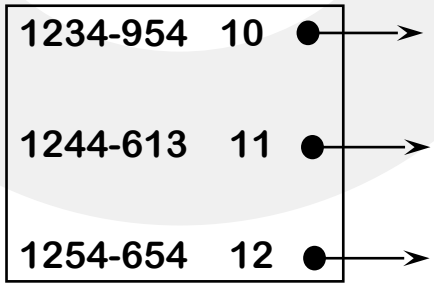
- Vorgehen der Suche mit Indexen (I) -

Aufgabe

- gesucht wird ein Datensatz D mit dem Primärschlüsselwert s aus der Menge der Primärschlüsselwerte S

Organisation

- es existiert ein einstufiger Index für die Menge der Primärschlüsselwerte S über den Datensätzen mit den Paaren (s_{max},b)



(s_{max}, b)

Vorgehen einstufiger Index

- Schritt 1: suche im Index einen Eintrag für den gilt: $s <=_{min} s_{max}$

  $<=_{min}$: es existiert kein weiteres s_{max}, das größer s ist und noch näher zu s liegt oder aber gleich s ist

- Schritt 2: lies den Block, dessen Anfangsadresse durch b gegeben ist

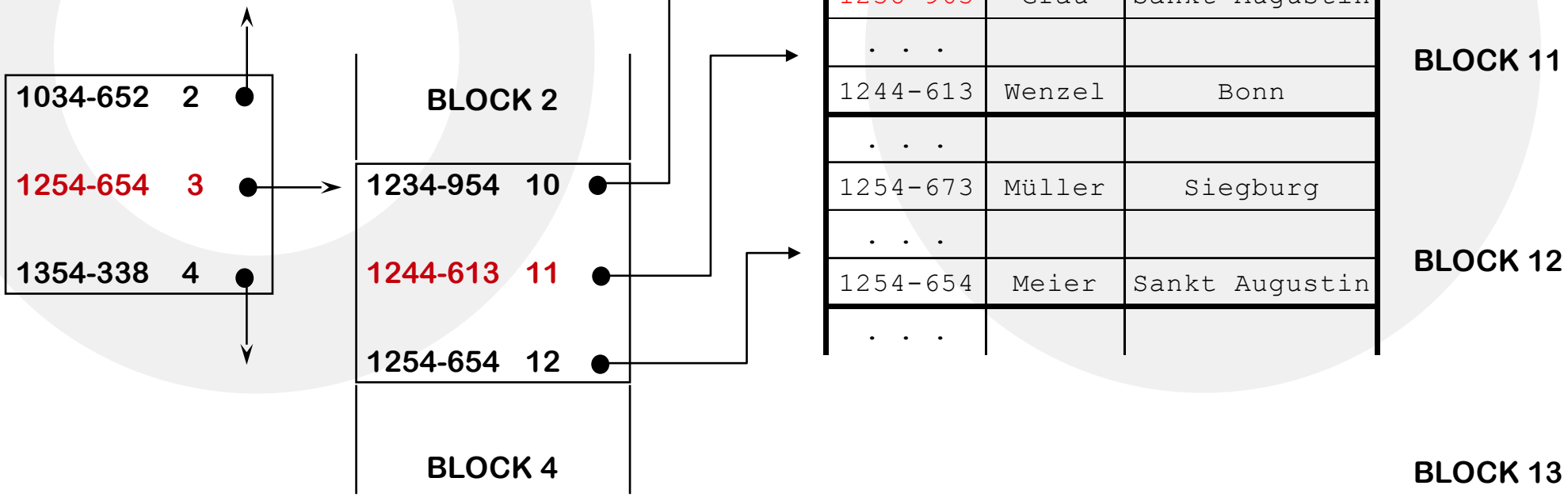
- Schritt 3: suche (sequentiell oder binär) in diesem Block nach dem Datensatz mit dem Primärschlüsselwert s

Index-Sequentielle Organisation

- Vorgehen der Suche mit Indexen (II) -

Beispiel: Index-sequentielle Ordnung

- zweistufiger Index
- suche Datensatz 1238-963
- Organisation mit (s_{max}, b)



Index-Sequentielle Organisation

- Probleme beim Einfügen von Datensätzen -

Speicherreserve

- bei vollständig gefüllten Blöcken der Datensatzdatei führt bereits das Einfügen eines neuen Datensatzes zu umfangreichen Datenverschiebungen
- daher lässt man auf den Blöcken häufig eine Speicherreserve
  (also z.B.: nur zu 60% gefüllte Blöcke)
- ist die Speicherreserve erschöpft, müssen Überlaufblöcke angelegt werden

Überlaufblöcke

- sind eine Art künstlicher Verlängerung eines Blockes
- werden durch „Verzeigerung“ mit den zu verlängernden Blöcken verbunden

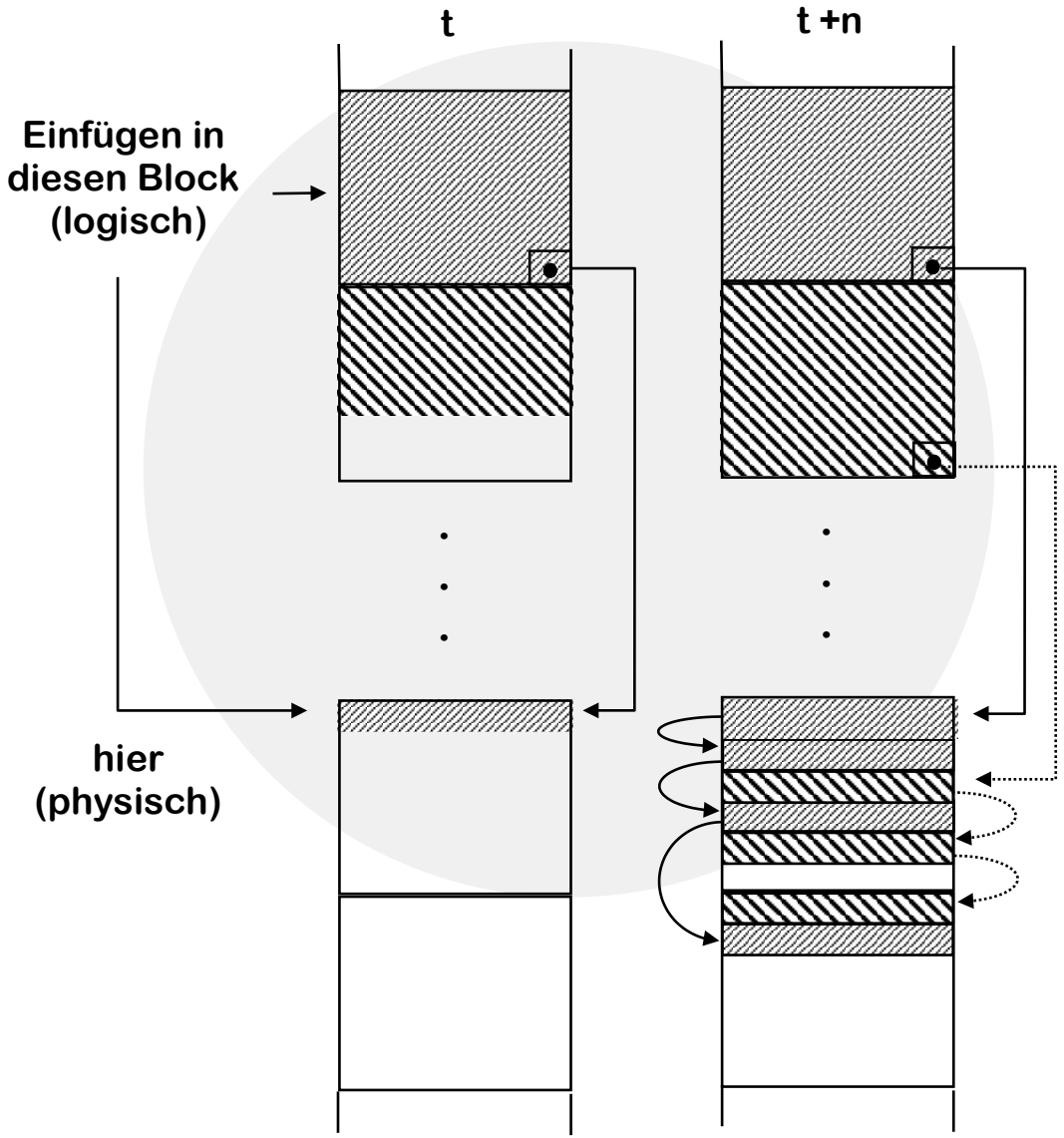
Organisation

- entweder pro übergelaufenem Block eine eigene Verkettung von Überlaufblöcken
- oder ein Überlaufbereich, in dem sog. Überlaufsätze sequentiell abgelegt werden (hier werden dann einzelne Sätze verkettet)

- Überlauf und Reorganisation -

Problem beim Überlauf

- die Verteilung von Datensätzen erfordert von Zeit zu Zeit eine Reorganisation der Datenbank



- Gestreute Organisation -

Inhalt

- Physische Entwurf der Datenbank
- Interne Datensätze
- Zugriffsverfahren für Primärschlüssel
  - Primär- und Sekundärschlüssel
  - Sequentielle Organisation
  - Index-Sequentielle Organisation
  - **Gestreute Organisation**
  - Sortiert-Logisch-Sequentielle Organisation
  - Baumverfahren
  - Diskussion der Verfahren

Überblick

- Hash-Verfahren
- Beispiel: Divisions-Rest-Verfahren
- Kollisionsauflösung bei Hash-Verfahren

- Hash-Verfahren -

Hash-Verfahren

- mit Hilfe einer Funktion, deren Eingabewerte die Primärschlüsselwerte sind, wird eine Speicheradresse errechnet, auf der der betreffende Datensatz dann abgelegt wird
- das Verfahren verstreut die Datensätze im zur Verfügung stehenden Speicherbereich (daher auch Streuspeicherverfahren genannt)
- Speicheradressen können beispielsweise Blocknummern sein

Primärschlüsselwerte

- sei S die Menge der existierenden Primär-schlüsselwerte und τ die Menge der grundsätzlich möglichen gültigen Primärschlüsselwerte, dann ist
  - $S \subseteq \tau$

Beziehung Primärschlüssel / Speicherplatz

- sei N die Menge der verfügbaren Speicheradressen, dann ordnet
  - $h: \tau \rightarrow N$
- jedem Schlüsselwert $s \in \tau$ eine Speicheradresse $\in \mathbb{N}$ zu

Bedingung

- card(S) » card(N) und damit auch
- card(τ) » card(N)
  - »: „deutlich“ größer

Folge

- mehreren Sätzen können gleiche Adressen zugewiesen werden (gewollt),
- was allerdings zu Kollisionen führt

Gestreute Organisation

- Beispiel: Divisions-Rest-Verfahren -

Voraussetzung

- der Primärschlüssel S sei numerisch
- zur Verfügung stehen Z Speicherplätze

Funktion

- h(s) = s modulo Z
- h(s) ist der Rest, der sich bei der ganzzahligen Division von s durch Z ergibt

Annahme im Beispiel

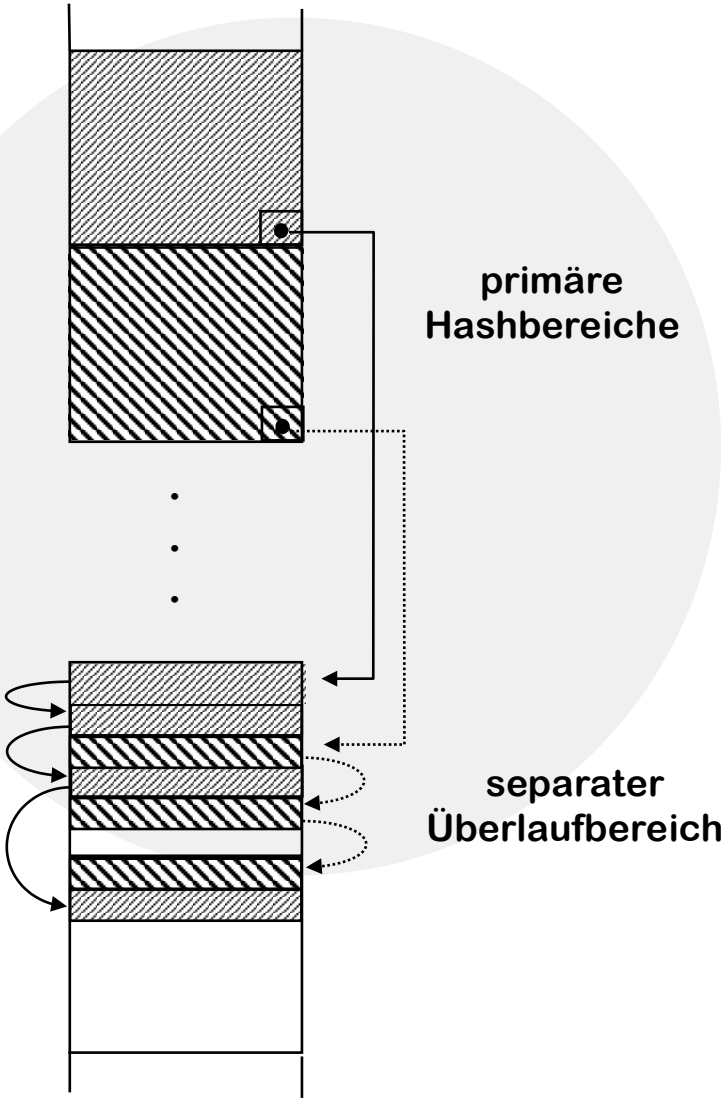
- es stehen 100 Speicherblöcke zur Verfügung (Z = 100)
- die Primärschlüsselwerte müssen zuvor in einen numerischen Wert umgewandelt werden (Problem Bindestrich)
- 10 Kollisionen (K) pro Speicherplatz sind problemlos möglich

Speicher-platz			
	. . .		
	. . .		
13	1234-713	Dillo	Sankt Augustin
73	1234-873	Wilke	Bonn
54	1234-954	Schmidt	Siegburg
	. . .		
63	1238-963	Grau	Sankt Augustin
	. . .		
K → 13	1244-613	Wenzel	Bonn
	. . .		
K → 73	1254-673	Müller	Siegburg
	. . .		
K → 54	1254-654	Meier	Sankt Augustin
	. . .		

- Kollisionsauflösung bei Hash-Verfahren -

Methoden

- Kollisionsbehandlung mit Überlaufbereich
  - Reservierung bestimmter Blöcke für den Überlauf (siehe Skizze)
  - Möglichkeiten der Kollisionsbehandlung ohne Überlaufbereich
    - Nutzung des ersten freien Speicherplatzes im Hash-Bereich nach dem durch die Hash-Funktion h bestimmten Platz h(s)
    - Abspeicherung im ersten freien Speicherplatz nach einer Folge von Speicherplätzen, deren Adressen über Zufallszahlen berechnet werden (Schlüssel = Eingabe für Zufallszahlengenerator)
    - Einführung einer zweiten Hash-Funktion für die kollidierten Datensätze



- Sortiert-Logisch-Sequentielle Organisation -

Listen

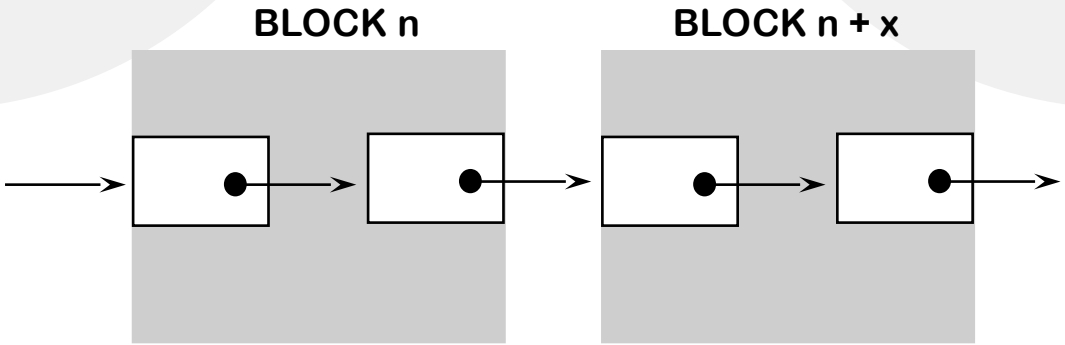
- Verkettung der Datensätze durch Zeiger
- Blöcke sind nicht physisch benachbart

Vorteil

- Vollständige Ordnung der Datensätze
- Ordnung nach dem Sortierkriterium des Schlüssels möglich
- unabhängig von der physischen Speicherung ergibt sich immer eine logische Ordnung

Nachteil

- bei m Datensätzen sind zum Suchen
  - durchschnittlich m/2 Datensätze zu lesen, da Ordnung nur logisch
  - die sich auf (m/2/Anzahl Datensätze pro Block) Blockzugriffe verteilen
- Löschen und Einfügen können eine Liste sehr ungünstig über den gesamten Speicherbereich verteilen, so dass es zu einer erheblichen Verlangsamung von Schreibzugriffen kommen kann



- Baumverfahren -

Inhalt

- Physische Entwurf der Datenbank
- Interne Datensätze
- Zugriffsverfahren für Primärschlüssel
  - Primär- und Sekundärschlüssel
  - Sequentielle Organisation
  - Index-Sequentielle Organisation
  - Gestreute Organisation
  - Sortiert-Logisch-Sequentielle Organisation
  - **Baumverfahren**
  - Diskussion der Verfahren

Überblick

- Baumorganisation
- Verzeigerung von B-Bäumen
- Operationen auf B-Bäumen
- Einfügen in B-Bäume
- B^+-Bäume
- Ausprägung B^+-Baum

- Baumorganisation -

Bäume

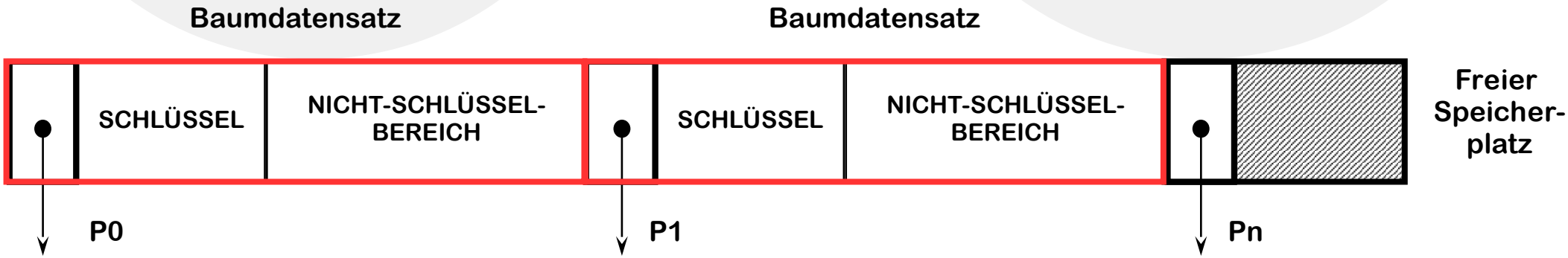
- spezielle Form von Bäumen für Daten,
- … die wie Tabellen auf Sekundärspeichern gelagert sind und
- … die erforderlichen Segmente bei Bedarf in den Arbeitsspeicher geladen werden

Arten

- B-Baum und B^+-Baum
  (können als baumartig kaskadierter Index angesehen werden)

Definition

- die Knoten eines B-Baumes sind Speicherblöcke fester Länge
- jeder Block kann 2*k B-Baumdatensätze gleicher Länge aufnehmen
  (k = beliebige Zahl)
- jeder B-Baum Datensatz besteht aus drei Teilen:
  - einen Zeiger auf einen weiteren Knoten
  - einen Schlüssel
  - einen Nichtschlüsselbereich (z.B.: Zeiger auf Sekundärspeicherblock)



- Verzeigerung von B-Bäumen -

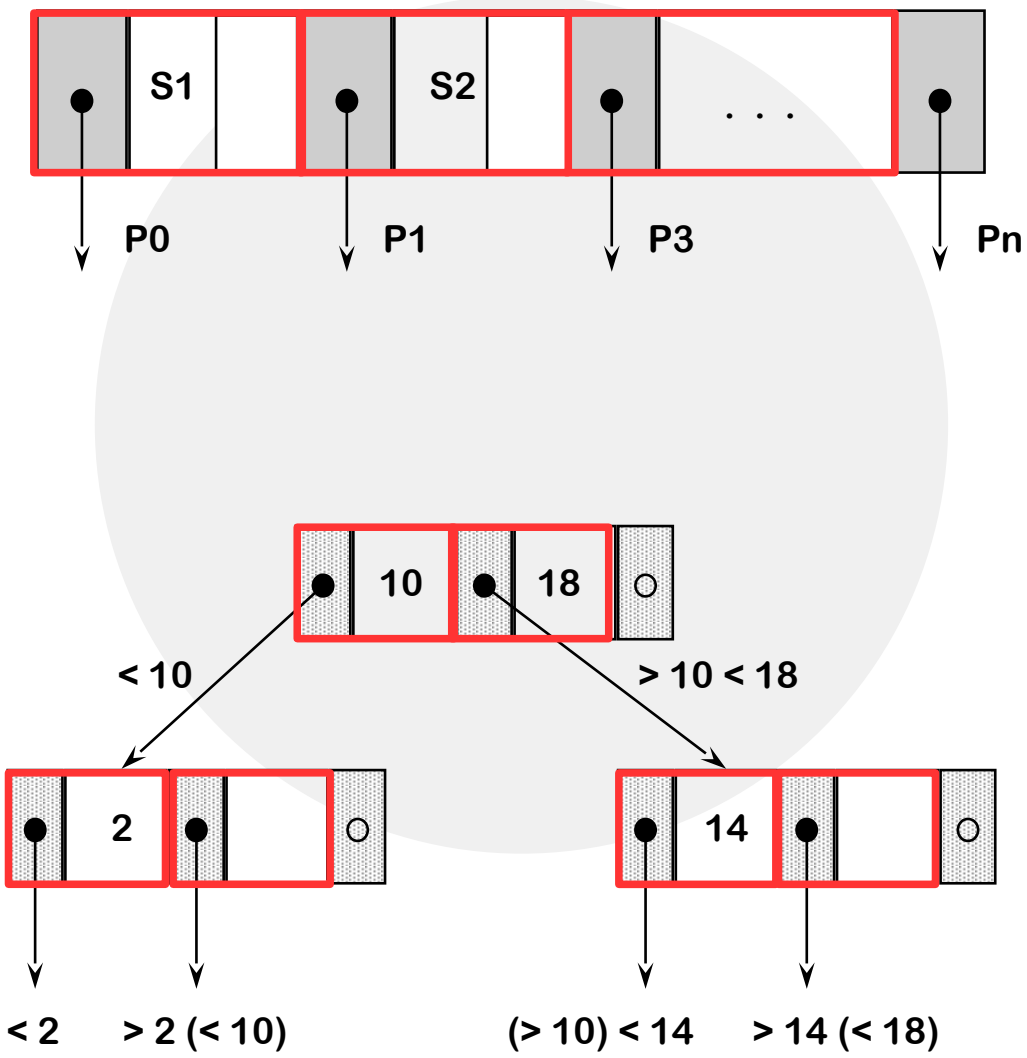
Vorgehensweise

- P0 zeigt auf einen Teilbaum (Block), dessen Schlüssel alle kleiner sind als S1
- P1 zeigt auf einen Teilbaum (Block), dessen Schlüssel alle zwischen S1 und Sn liegen
- Pn zeigt auf einen Teilbaum (Block), dessen Schlüssel alle größer als Sn-1 sind
- in den Blattknoten sind die Zeiger nicht definiert
- nicht alle Zeiger müssen benutzt werden

Unterschied zur Index-Sequentiellen Organisation

- keine Unterscheidung zwischen Index- und Datenblöcken
- B-Baum ist ausgeglichen („balanciert“)

B für „balanciert“ ; nicht für „binär“ !



Tafelbild: Abbildung der Daten auf den Sekundärspeicher - B-Baum-organisiert

123	XYZ	ADC	987

Beispieldatensatz mit 80 Bytes Länge

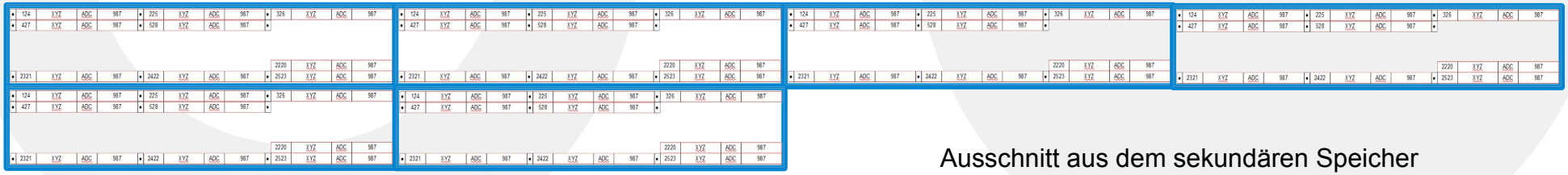
Pointer mit 8 Bytes Länge (bei 64-Bit-Adressen)

•

•	124	XYZ	ADC	987	•	225	XYZ	ADC	987	•	326	XYZ	ADC	987
•	427	XYZ	ADC	987	•	528	XYZ	ADC	987	•				

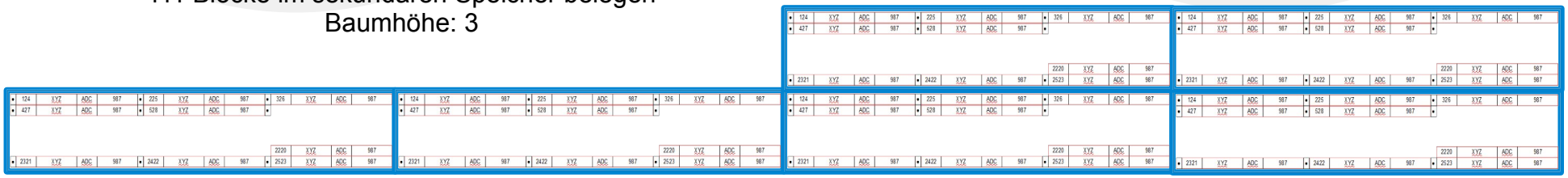
B-Baum-Block des sekundären Speichers mit 8 KB
(Füllgrad ca. 90 Beispieldatensätze zu je 80 Bytes Daten + 8 Bytes-Pointer

											2220	XYZ	ADC	987
•	2321	XYZ	ADC	987	•	2422	XYZ	ADC	987	•	2523	XYZ	ADC	987



Ausschnitt aus dem sekundären Speicher
(1 MB entsprechen ca. 122 Blöcke zu je 8 KB)

10.000 Datensätze zu je 88 Bytes (mit Pointer) würden
unter Ausnutzung des Füllgrads als B-Baum
111 Blöcke im sekundären Speicher belegen
Baumhöhe: 3



Tafelbild: Abbildung der Daten auf den Sekundärspeicher - B-Baum-Höhe

123	XYZ	ADC	987

•

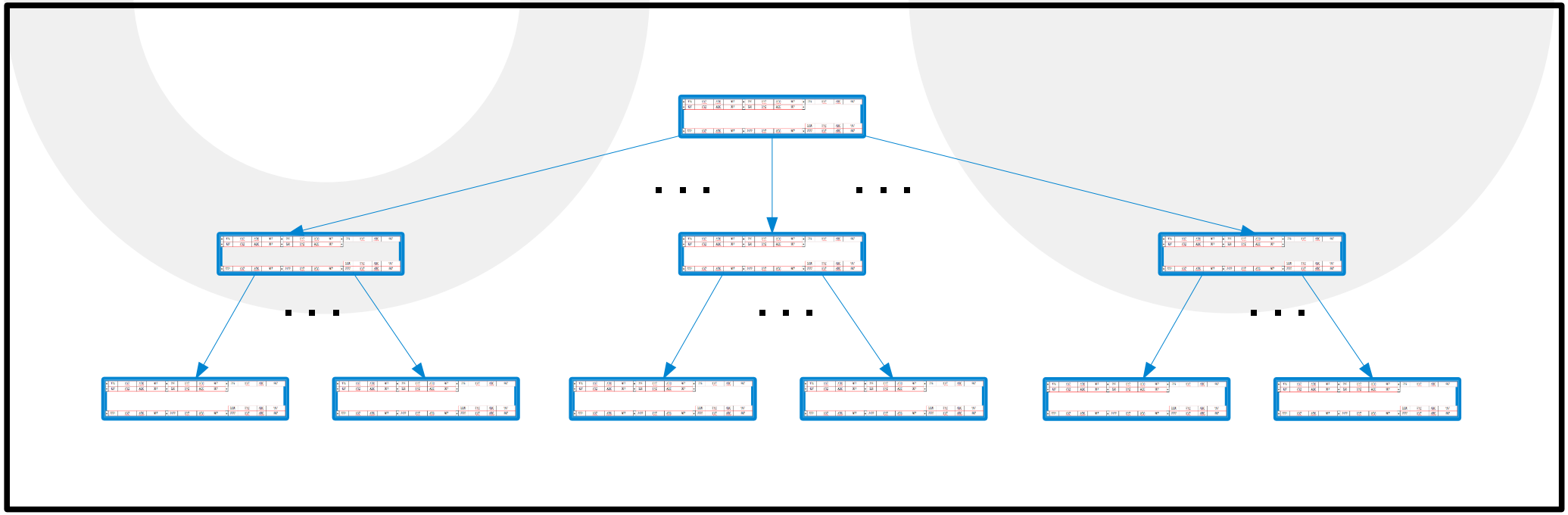
Beispieldatensatz mit 80 Bytes Länge

Pointer mit 8 Bytes Länge (bei 64-Bit-Adressen)

Beispiel 10.000 Datensätze

Ebene 1: ca. 90 Datensätze (10.000 werden benötigt)
Ebene 2: plus weitere 90 * 90 Datensätze = 90 + 8.100 = 8.190 Datensätze (10.000 werden benötigt)
Ebene 3: plus weitere 90 * 90 * 90 Datensätze = 8.190 + 729.0000 = 737.190 Datensätze (10.000 werden nur benötigt)

Jeder Datensatz wird mit maximal 3 Blockzugriffen gefunden



- Operationen auf B-Bäumen -

Einfügen und Löschen (Bedingungen)

- B-Bäume sind immer ausgeglichen (alle Blätter haben dieselbe Höhe h)
- jeder Knoten (außer der Wurzel) enthält mindestens k Schlüssel
- die Wurzel des B-Baumes hat entweder keinen Sohn oder mindestens zwei Söhne
- jeder Knoten (außer der Wurzel und den Blättern) hat m+1 Söhne wenn m die Anzahl der im Knoten enthaltenen Schlüssel ist

Suchen in B-Bäumen

- der Kenntnis der Verzeigerung entsprechend wird der Baum durchlaufen indem der zu suchende Schlüssel mit den Schlüsseln in den Knoten verglichen wird
- Start bei der Wurzel
- Ende wenn Schlüssel gefunden oder kein Blatt den Schlüssel enthält (erfolglose Suche)

Größe eines B-Baumes

- Größe Baumdatensatz
  = Größe Zeiger + Datensatz
- Anzahl Datensätze pro Block
  = Blockgröße / Größe Baumdatensatz

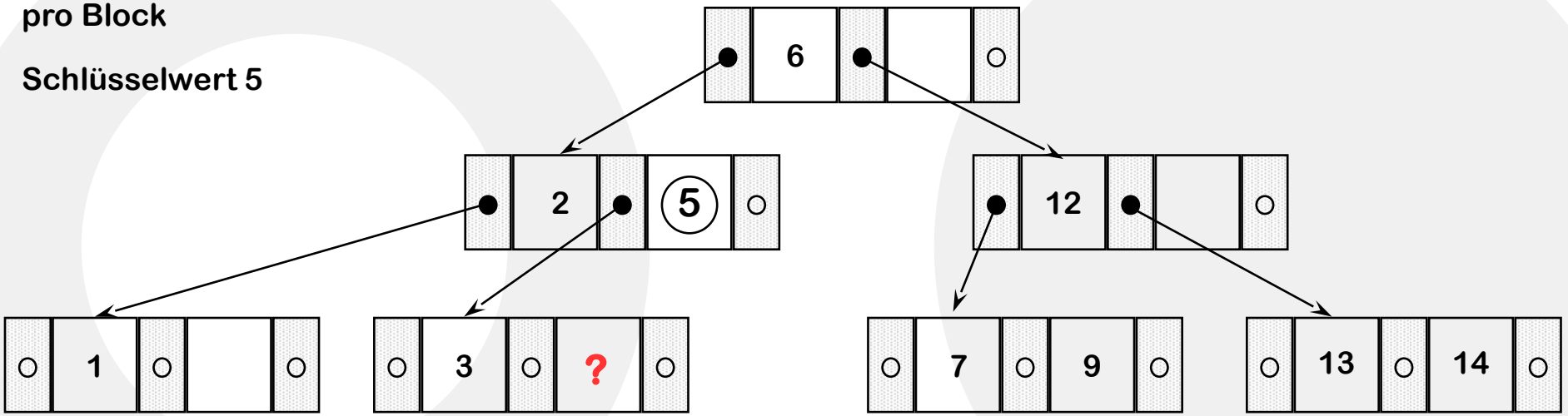
Beispiel

- bei einer
  - Blockgröße von z.B. 8 KB,
  - Zeiger von z.B. 8 Bytes und
  - Datensätzen von z.B. jeweils 72 Bytes
  - kann ein Knoten ca. 100 Baumdatensätze aufnehmen
- ein B-Baum mit zwei Ebenen kann in diesem Beispiel
- 100 + (100 * 100) = 10.100 Datensätze verwalten,
- die sich mit maximal 2 Blockzugriffen finden lassen

- Einfügen in B-Bäume (I) -

Beispiel:

- zwei Datensätze pro Block
- Schlüsselwert 5

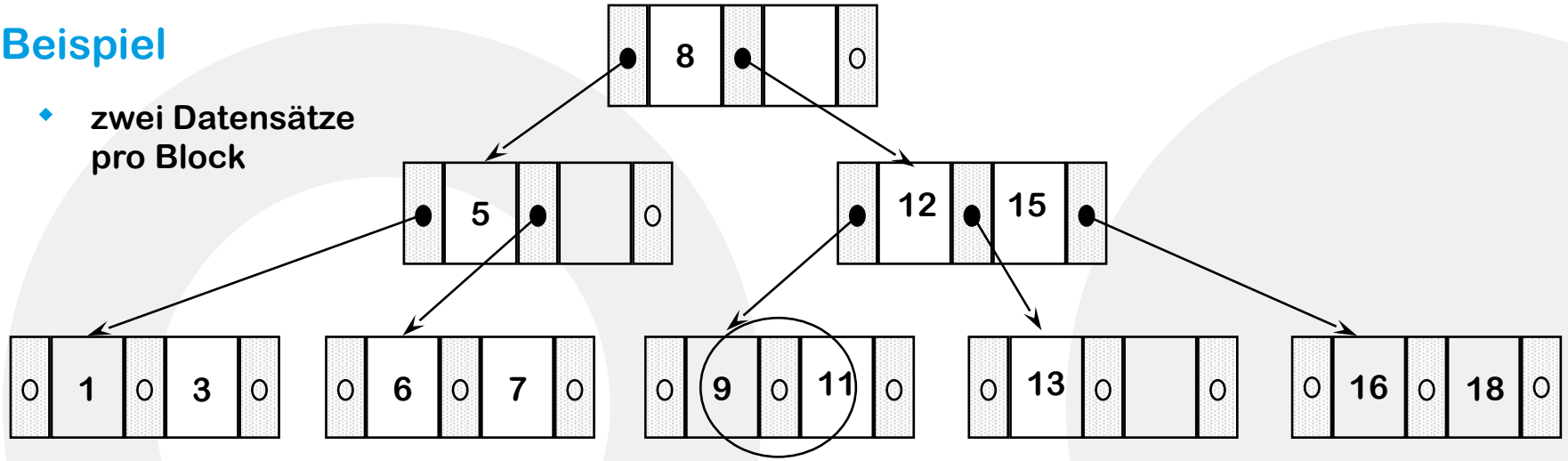


Im Beispiel werden nur die Schlüsselwerte dargestellt

- Einfügen in B-Bäume (II) -

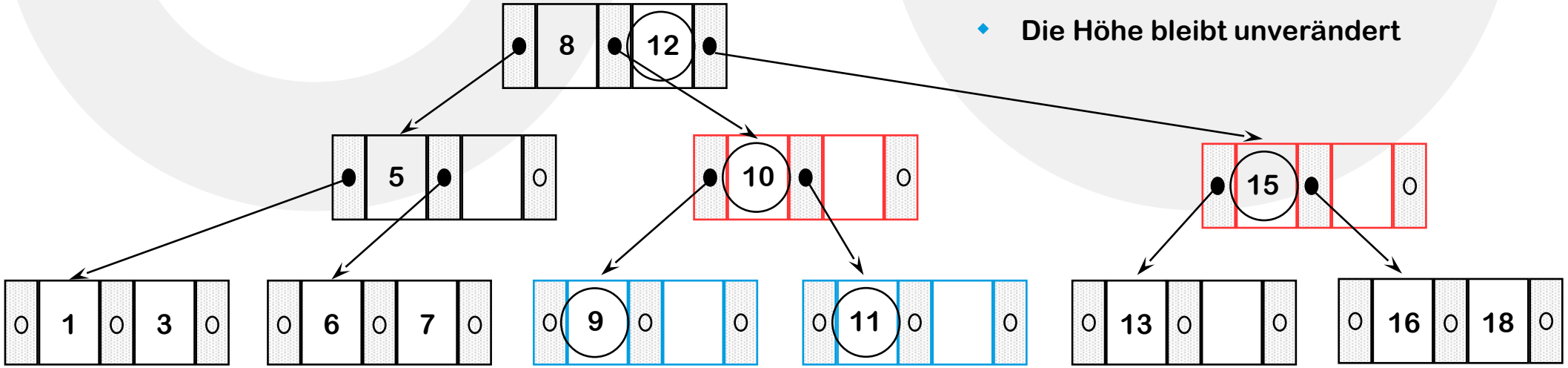
Beispiel

- zwei Datensätze pro Block



- Schlüsselwert 10

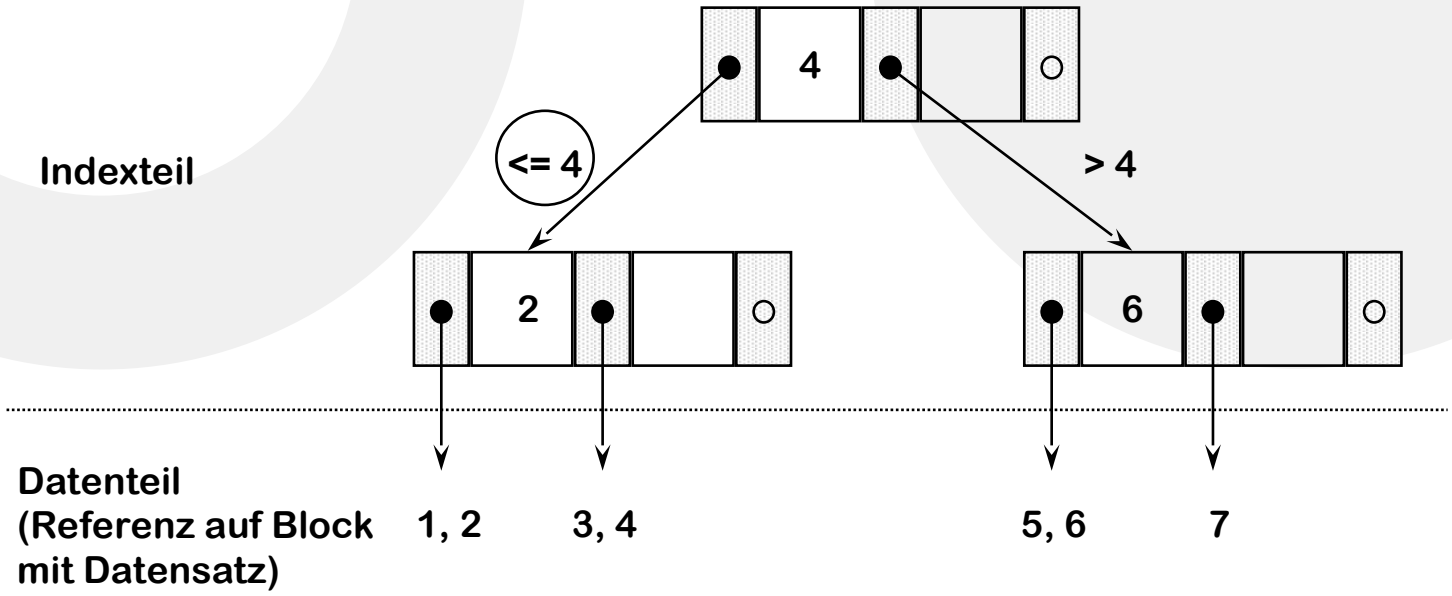
- Zwei Blöcke müssen gesplittet werden
- Die Höhe bleibt unverändert



- B⁺-Bäume (I) -

Unterschied zu B-Bäumen

- B+-Bäume enthalten die Satzinformation nur (!) in den Blättern
- nur der Baum der Höhe h-1 ist als Index zu verstehen
- oftmals auch als B*-Baum bezeichnet
  - B*-Baum stellt eine Spezialisierung dar
- die Satzstruktur des B⁺-Baumes (Indexteil) enthält im Unterschied zum B-Baum keinen (!) Nicht-Schlüsselteil
- es sind nicht mehr alle Schlüssel im Indexteil des Baumes enthalten
- die Datensätze in den Blättern befindet sich daher außerhalb des eigentlichen Baums



Beispiel mit maximal 2 Indexdatensätzen pro Block

- B⁺-Bäume (II) -

Größe eines B⁺-Baumes

- Größe Indx-Baumdatenatze
  ca. Größe Zeiger + Größe Primärschlüssel
- Anzahl Index-Baumdatensätze pro Block
  ca. Blockgröße / Größe Index-Baumdatensatz

Beispiel

- bei einer
  - Blockgröße von z.B. 8 KB
  - Zeiger von z.B. 8 Bytes und
  - davon Primärschlüssel mit 2 Bytes
  - kann ein Index-Block
    ca. 800 Index-Baumdatensätze aufnehmen
- bei einer
  - Datensatzgröße von z.B. jeweils 100 Bytes
  - Blockgröße von 8 KB
  - kann ein Datenblock ca. 80 Datensätze aufnehmen

Berechnung der Zugriffe

- für 80 Datensätze (1 Block)
  - 1 bis 80 Datensätze
    verteilen sich über 1 Datenblock
  - wird kein Index benötigt
  - ein Zugriff (direkter Zugriff) möglich
- für 81 bis (800 * 80) Datensätze
  - 81 bis 64.000 Datensätze
    verteilen sich über (bis zu) 800 Datenblöcke
  - wird eine Index-Ebene benötigt
  - zwei Zugriffe sind erforderlich
- für 64000 bis (800 * 800 * 80) Datensätze
  - 64.001 bis 51.200.000 Datensätze
    verteilen sich über (bis zu) 640.000
    Datenblöcke
  - werden zwei Indexebenen benötigt
  - drei Zugriffe sind erforderlich

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Tafelbild: Abbildung der Daten auf den Sekundärspeicher - B⁺-Baum-organisiert

123	XYZ	ADC	987

Beispieldatensatz mit 80 Bytes Länge, davon Schlüssel mit 8 Bytes

•	123

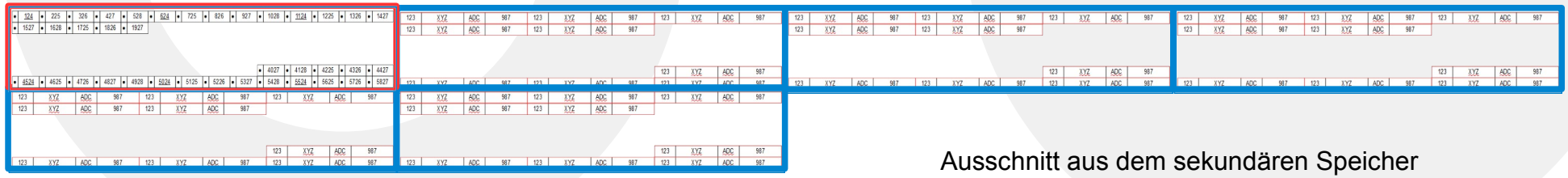
Index-Datensatz: Schlüssel mit 8 Bytes plus Pointer mit 8 Bytes Länge (bei 64-Bit-Adressen)

•	124	•	225	•	326	•	427	•	528	•	624	•	725	•	826	•	927	•	1028	•	1124	•	1225	•	1326	•	1427
•	1527	•	1628	•	1725	•	1826	•	1927																		

B⁺-Baum-Block des sekundären Speichers mit 8 KB
Füllgrad: ca. 500 Indexdatensätze zu je 16 Bytes

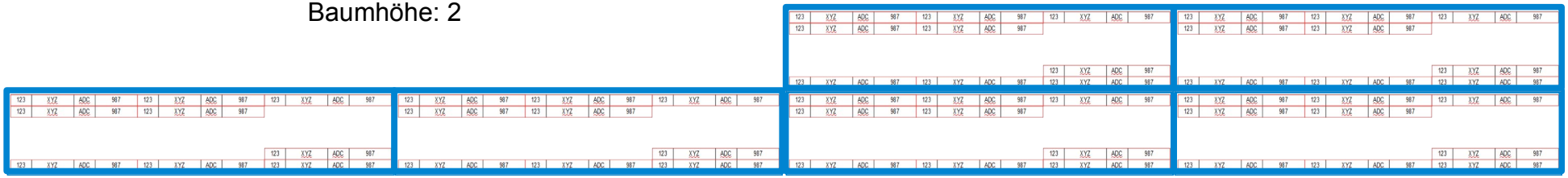
Damit lassen sich
ca. 500 Daten-Blöcke
oder 500 weitere Index-Datenblöcke adressieren

•	4027	•	4128	•	4225	•	4326	•	4427
•	4524	•	4625	•	4726	•	4827	•	4928	•	5024	•	5125	•	5226	•	5327	•	5428	•	5524	•	5625	•	5726	•	5827
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Ausschnitt aus dem sekundären Speicher
(1 MB entsprechen ca. 122 Blöcke zu je 8 KB)

10.000 Datensätze zu je 80 Bytes würden als B⁺-Baum
unter Ausnutzung des Füllgrads 100 Daten-Blöcke plus
1 Index-Block (20 % gefüllt) im sekundären Speicher belegen
Baumhöhe: 2



Tafelbild: Abbildung der Daten auf den Sekundärspeicher - B⁺-Baum-Höhe

123	XYZ	ADC	987

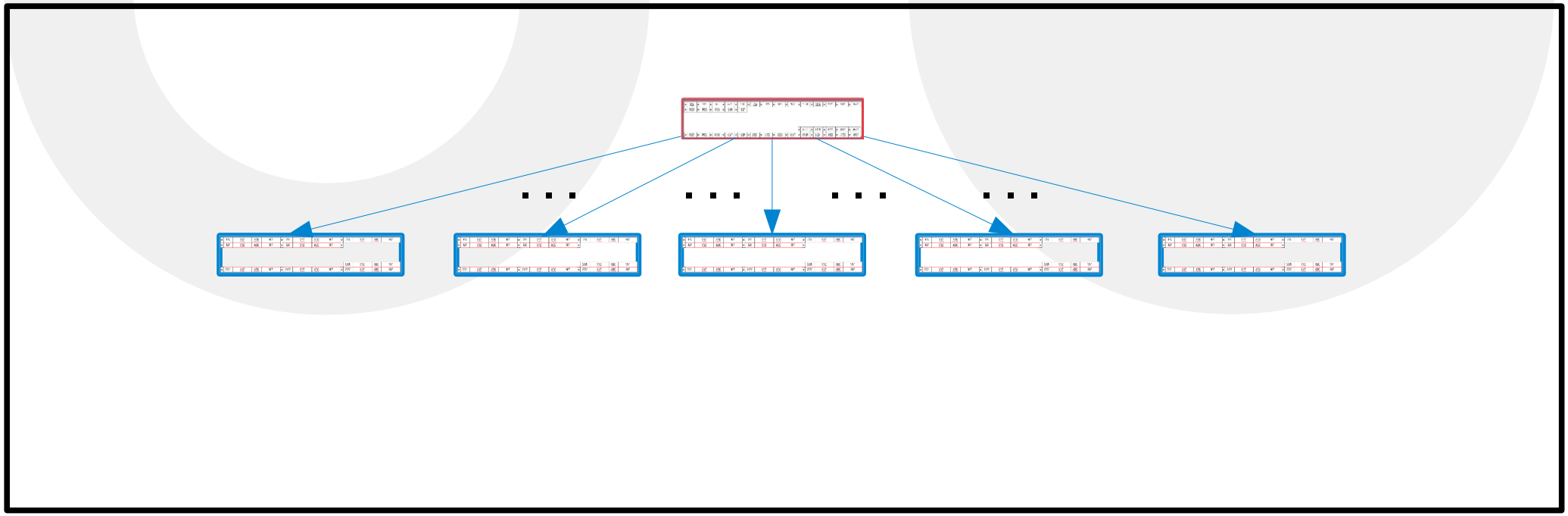
•	123

Beispieldatensatz mit 80 Bytes Länge, davon Schlüssel mit 8 Bytes

Beispiel 10.000 Datensätze

Ebene 1: maximal 500 Index-Datensätze (100 werden nur benötigt)
Ebene 2: maximal 500 * 100 Datensätze = 50.000 Datensätze (10.000 werden nur benötigt)

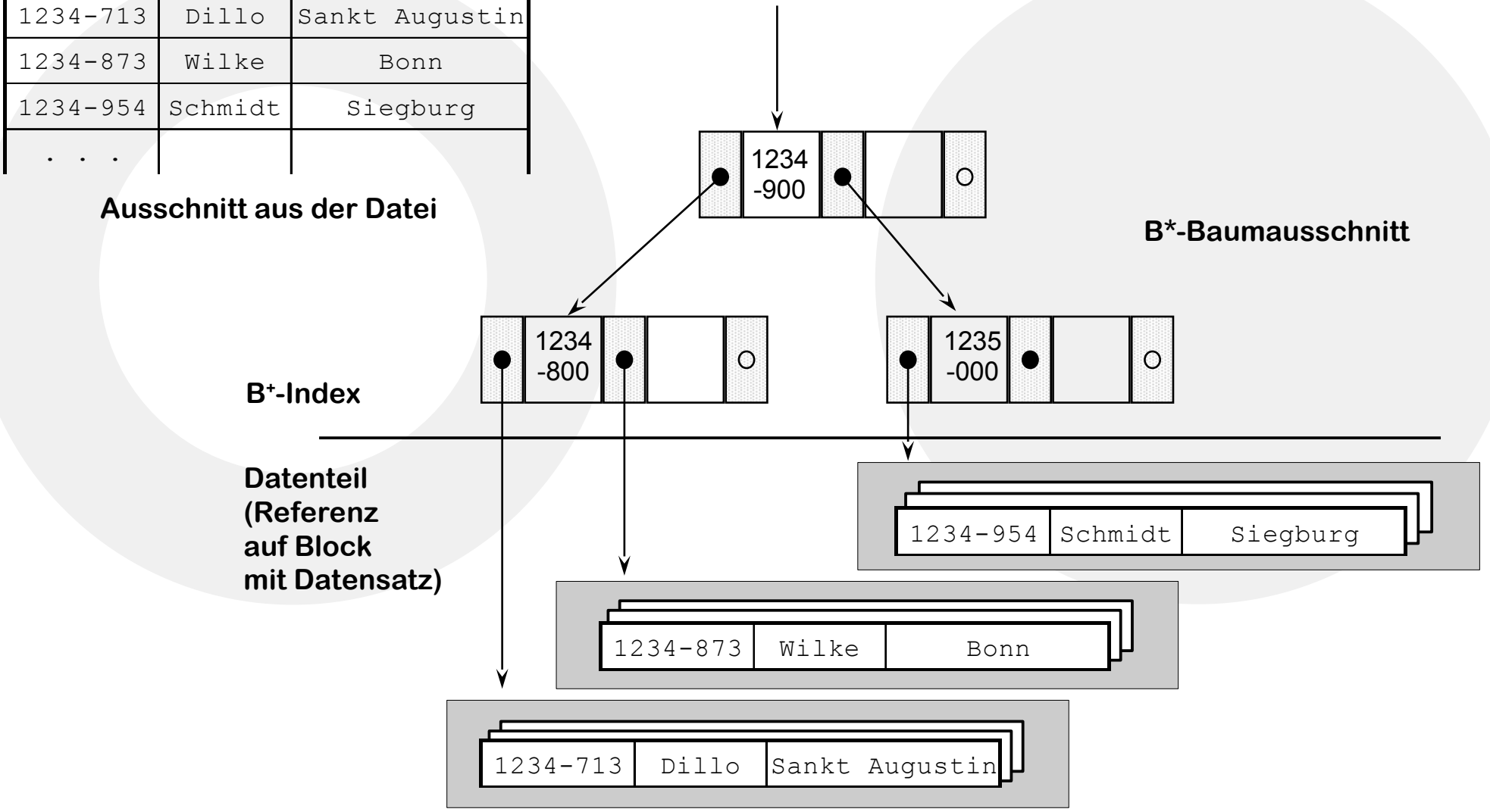
Jeder Datensatz wird mit genau 2 Blockzugriffen gefunden: einem Indexblockzugriff und einem Datenblockzugriff



- Ausprägung B⁺-Baum -

. . .		
1234-713	Dillo	Sankt Augustin
1234-873	Wilke	Bonn
1234-954	Schmidt	Siegburg
. . .		

Ausschnitt aus der Datei



Zugriffsverfahren für Primärschlüssel

- Diskussion der Verfahren -

Sequentielle Organisation

- nur bei sehr kleinen Datenmengen zu empfehlen

Index-Sequentielle Verfahren

- von Zeit zur Zeit ist Reorganisation erforderlich
  da die sequentielle Verwaltung statisch ist
- zusätzliche Zugriffe auf den Überlaufbereich
  verschlechtern die Performance

Gestreute Verfahren

- nur nach umfangreicher Reorganisation an
  veränderte Datenmenge anpassbar
- direkter Zugriff
  in der Sortierreihenfolge nicht möglich
- sehr schnell, wenn ohne Kollisionen
- kein Speicheraufwand für Indexführung
- Anwendung nur für Sätze fester Länge

Sortiert-Logisch-Sequentielle Verfahren

- hoher Suchaufwand
- werden im Allgemeinen
  nicht für Primärschlüssel verwendet

Baumverfahren

- einfach anpassbar an die Datenmenge
- Zugriff in der Sortierreihenfolge
  der Schlüssel möglich
- Aufwand exakt vorhersagbar
- anwendbar für beliebige Datensätze

Tafelübung

Ausgangssituation

- Block: 8 KBytes
- Datensatz: 80 Bytes (mit Primärschlüssel)
- Primärschlüssel: 12 Bytes
- Pointer: 4 Bytes
- Anzahl Datensätze: 10.000.000

Anzahl Blöcke zum Speichern (ungefähr)

- erforderlicher Speicherplatz:
  - 10.000.000 * 80 Bytes
    = 800.000.000 Bytes = 800.000 KBytes
- erforderliche Blöcke:
  - 800.000 KBytes / 8 KBytes / Block
    = 100.000 Blöcke
  - 100.000 Blöcke müssen sequentiell durchsucht werden, um gezielt einen Datensatz zu finden

Berechnung des B⁺-Index

- Datensätze pro Datenblock:
  - 8.000 Bytes / 80 Bytes / Satz = 100 Sätze
- Indexeintrag (Referenz):
  - 12 Bytes + 4 Bytes = 16 Bytes
  - 8.000 KBytes / 16 Bytes / Indexeintrag
    = 500 Indexeinträge pro Indexblock
- eine Ebene: 1 Indexblock
  - Datensätze: 500 * 100 = 50.000 Datensätze
- zwei Ebenen: 1 + 500 Indexblöcke
  - Datensätze: 500 * 500 * 100
    = 25.000.000 Datensätze
- jeder der 10.000.000 Datensätze,
  die sich auf 100.000 Datenblöcke verteilen,
  lässt sich mit nur drei Blockzugriffen finden
  - zwei Indexblöcke und ein Datenblock