DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

Algorithmen und Datenstrukturen

Teil 11:
Suchen (II) – B-Bäume

DHBW Stuttgart Campus Horb
Fakultät Technik
Studiengang Informatik
Dozent: Olaf Herden
Stand: 06/2020

Gliederung

- Motivation und Definition
- Operationen auf B-Bäumen
- Beispiel
- B*-Bäume

Motivation (I)

- Bisherige Suchstrukturen Hauptspeicher-basiert
- Praxisanforderung:
  - Effiziente Speicherung und Reorganisation sehr großer Datenmengen
  - Datenvolumen größer als Hauptspeicher
- Ziele:
  - Reduzierung der Plattenspeicherzugriffe (Operationen etwa um Faktor 10^6 langsamer als Hauptspeicherzugriffe)
  - Plattenplatz „möglichst gut“ ausnutzen (Verhältnis tatsächlich genutzter Platz zu reserviertem Platz auf der Platte sollte über 50% liegen)
- Zur Technologie:
  - Tausch zwischen Haupt- und Plattenspeicher:
    - Nicht einzelne Datensätze
    - Blöcke oder Seiten
  - Typische Größen: 2 kB, 4 kB, 8 kB, 16 kB oder 32 kB

Motivation (II)

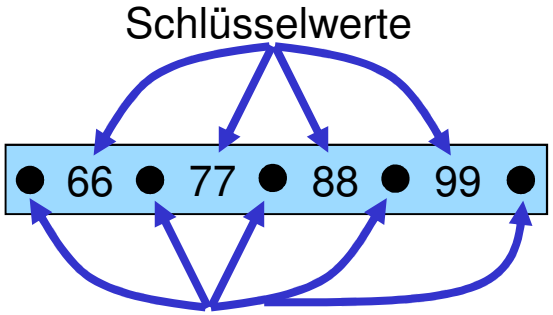
- Erste Idee:
  - Direkte Übertragung binärer Suchbäume
  - Verweise auf linken und rechten Nachfolger zeigen nicht auf Hauptspeicheradressen, sondern auf Blöcke des Plattenspeichers
  - Schlechte Ausnutzung des Plattenspeichers:
    - Blockgröße 4kB (4096 Byte)
    - Platzbedarf Zeiger (8 Byte)
    - Platzbedarf Schlüsselwert (32 Byte)
    - Platzbedarf restlicher Datensatzinhalt (200 Byte)
    - Jeder Block wäre mit 248/4096 ≈ 6% belegt
    - Vorgehensweise viel zu ineffizient
- Problem: Schlechter Füllungsgrad der Blöcke
- Idee zur Verbesserung: Mehrere Datensätze und Verweise in einem Knoten des Baums (d.h. in einem Block) speichern
- Strukturen werden als Mehrwege- oder Vielwege-Suchbäume bezeichnet

Motivation (III)

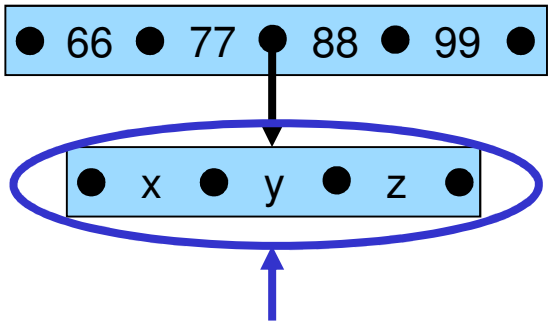
- Bekanntester Vertreter dieser Kategorie sind B-Bäume:
  - Verwalten in einem Knoten mehrere Datensätze und Verweise
  - Baumhöhe ist ausgeglichen, d.h. alle Pfade von der Wurzel zu einem Blatt sind gleich lang
  - Füllungsgrad eines Knoten ist $\geq 50\%$
  - Kein innerer Knoten hat leeren Unterbaum
  - Verzweigungsgrad kann variieren
- B-Baum wäre vollständig ausgeglichen, wenn
  - Alle Wege von der Wurzel zu den Blättern gleich lang
  - Jeder Knoten hat gleich viele Einträge
  - Reorganisationsaufwand hierfür aber viel zu groß
- B-Baum-Kriterium: Jeder Knoten (außer der Wurzel) enthält zwischen m und 2m Schlüsselwerte
- Anmerkung: B steht für balanciert und nicht für binär!

Definition

- Ein Baum heißt B-Baum (der Ordnung m), g.d.w.
  (1) Jeder Knoten außer der Wurzel enthält mindestens m Datensätze
  (2) Jeder Knoten enthält höchstens 2m Datensätze
  (3) Knoten mit k Datensätzen, der kein Blatt ist, besitzt genau k+1 Nachfolger
  (4) Alle Blätter besitzen das gleiche Niveau, d.h. sie stehen auf einer Ebene
  (5) Sind S_1, S_2, ..., S_k, mit $m \leq k \leq 2m$ die Schlüssel eines Knotens x, dann sind alle Schlüssel des ersten (d.h. linkesten) Nachfolgers von x kleiner als S_1, alle Schlüssel des (k+1)-ten Nachfolger größer als S_k und alle Schlüssel des i-ten Nachfolgers mit $1 < i < k+1$ größer als S_{i-1} und kleiner als S_i.
- Struktur einer Knotens:



Verweise auf Nachfolger



Werte in diesem Knoten liegen zwischen 77 und 88

Berechnung der Ordnung

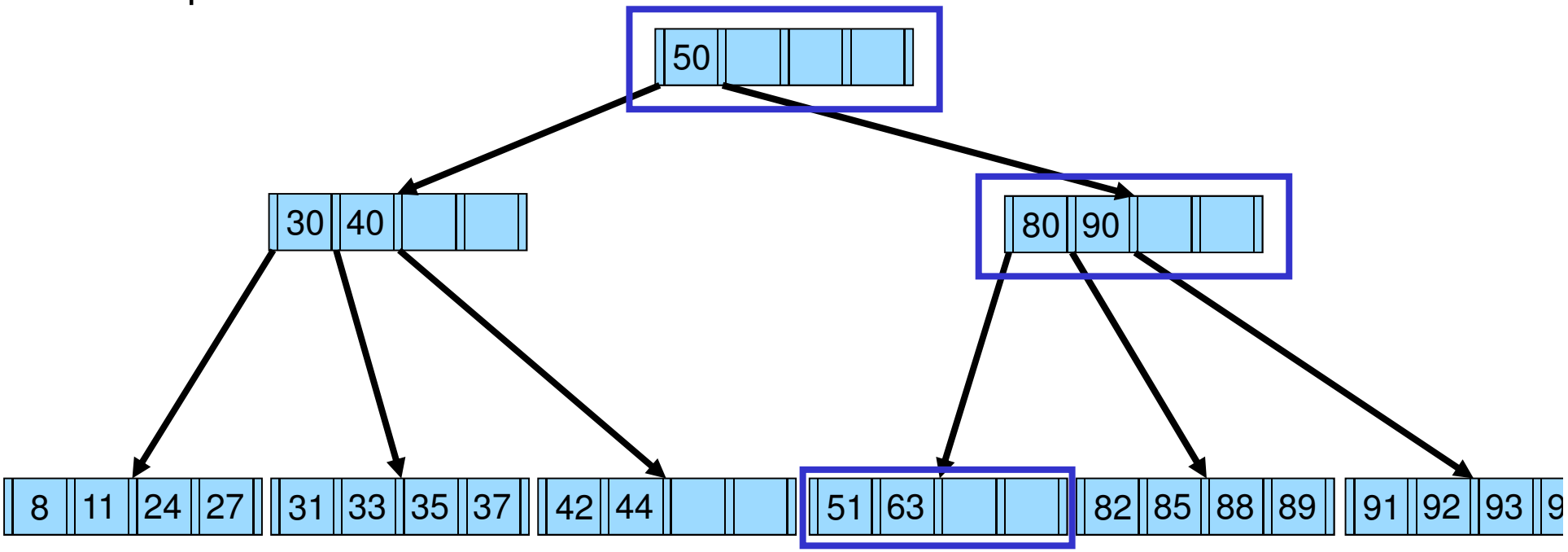
- Anzahl x der Einträge pro Block kann wie folgt berechnet werden:
  - Blockgröße b
  - Platzbedarf Zeiger z
  - Platzbedarf Schlüsselwert s
  - Platzbedarf restlicher Datensatzinhalt d
  - Es gilt: $x \cdot (s+d) + (x+1) \cdot z = b$
  - $\left\lfloor \frac{b-z}{s+d+z} \right\rfloor$ Einträge passen in einen Knoten,
    wobei $\lfloor\ \rfloor$ nächst kleinere ganze Zahl berechnet (Floor-Funktion)
- Beispiel:
  - Im einleitenden Beispiel ergeben sich 17 Werte
  - D.h. B-Baum der Ordnung 8 ist zu wählen

Übersicht

- Motivation und Definition
- Operationen auf B-Bäumen
- Beispiel
- B*-Bäume

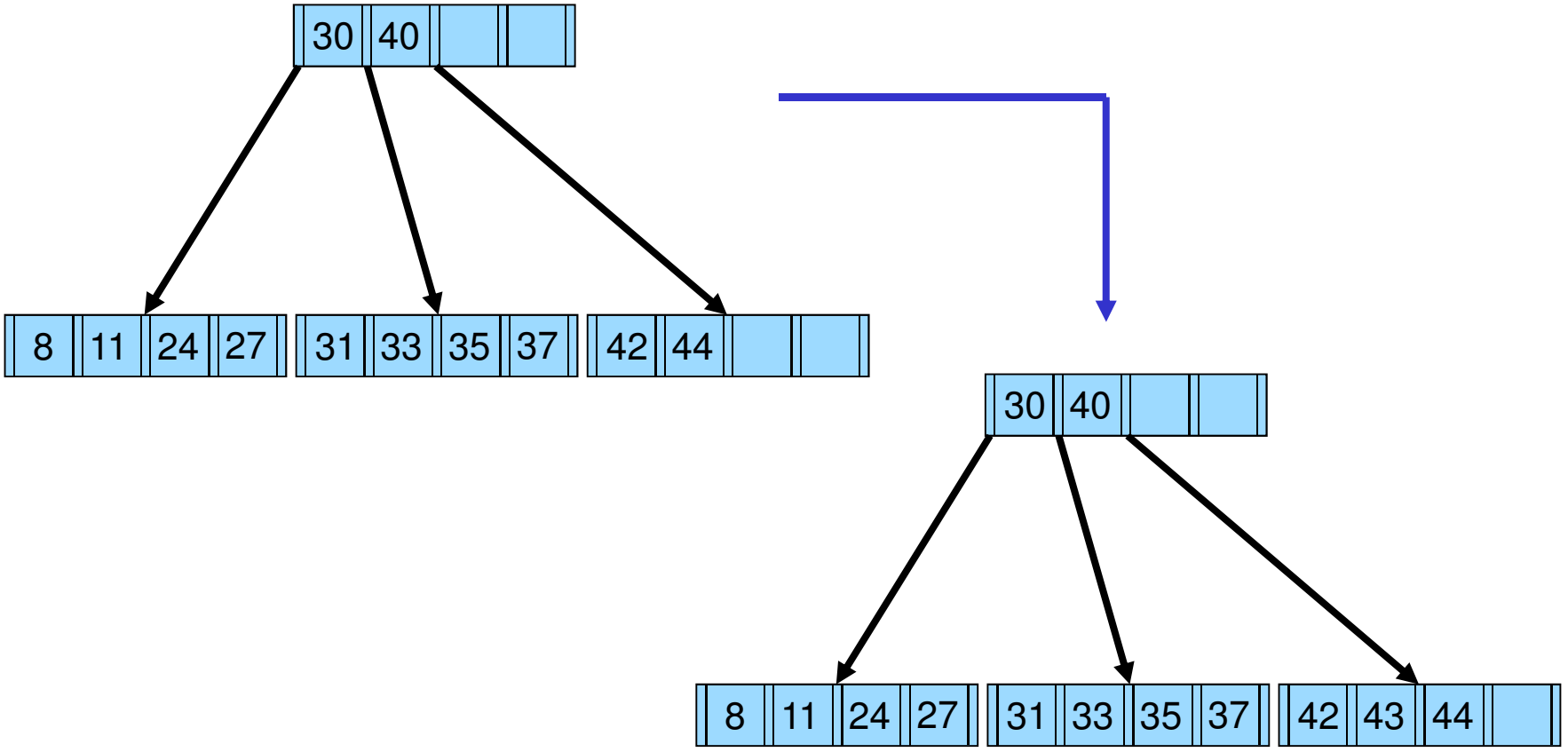
Suchen

- Suchen eines Elements:
  - Einstieg über Wurzel
  - Schlüsselwerte eines Knotens zerlegen Wertebereich in Intervalle
  - Gehe in Nachfolger, dessen Verweis im entsprechenden Intervall liegt
  - Fahre fort, bis Schlüsselwert gefunden oder Blatt erreicht
- Beispiel: Suche Element 63



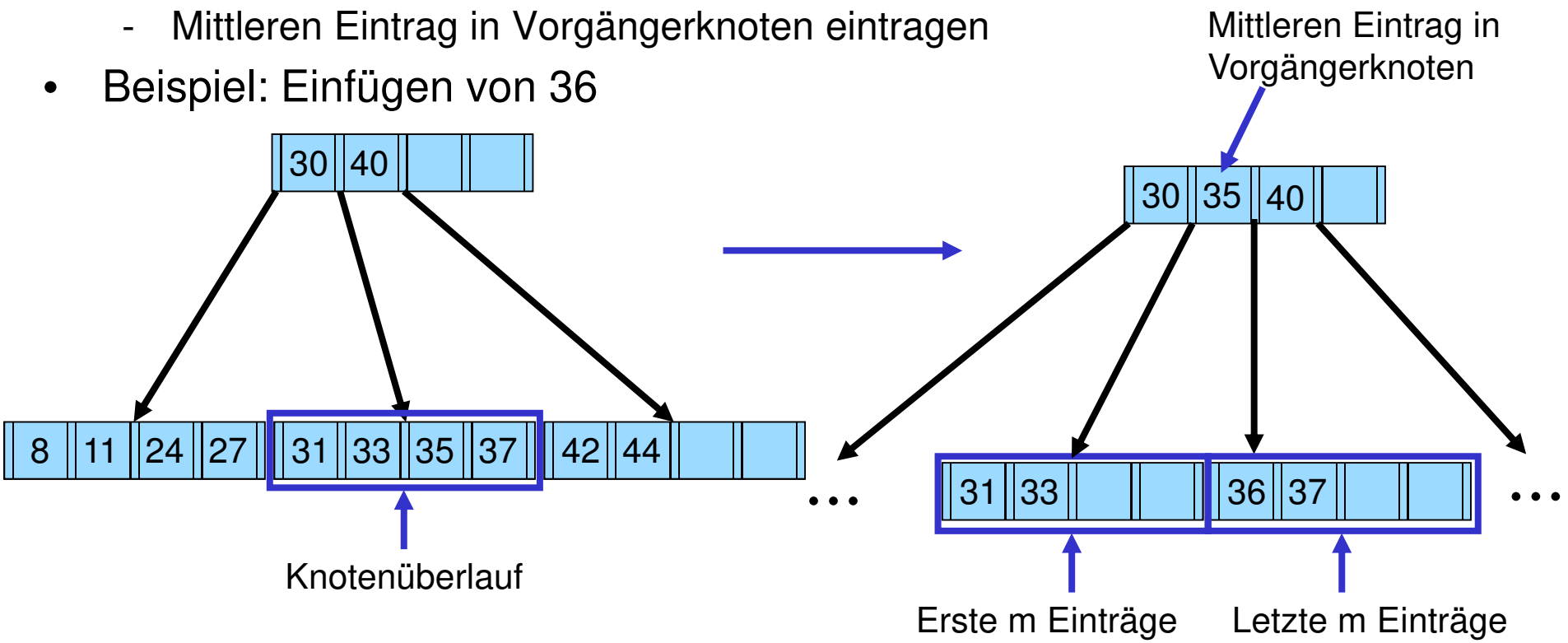
Einfügen (I)

- Entsprechende Blattseite suchen
- Blattseite hat < 2m Einträge ⇒ Neuen Wert einfügen
- Beispiel: Einfügen von 43



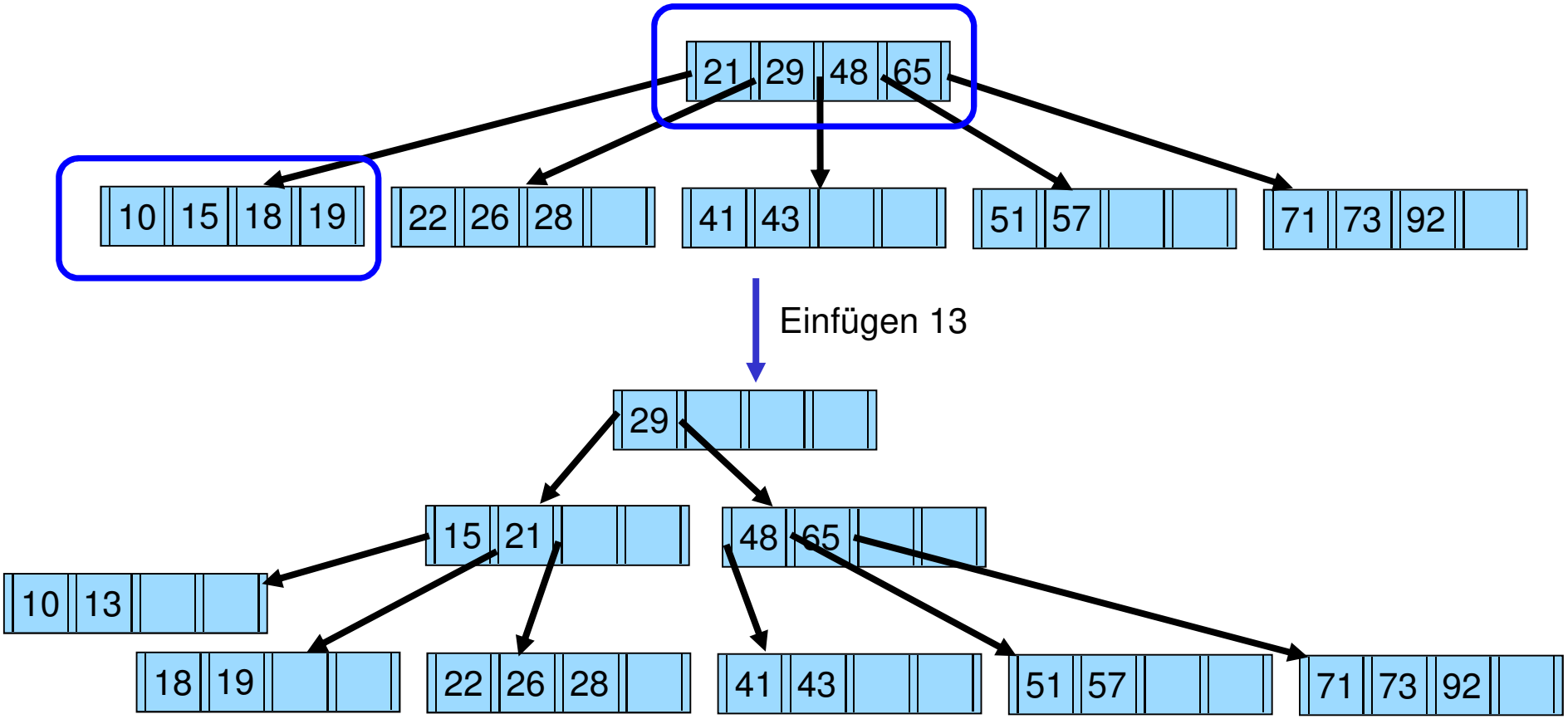
Einfügen (II)

- Blattseite hat 2m Einträge $\Rightarrow$ Überlauf des Knotens
- Behandlung Überlauf:
  - Erzeugen eines neuen Knotens unter Verwendung des Vorgängers
  - Erste m Einträge verbleiben im Originalknoten
  - Letzte m Einträge kommen in neuen Knoten
  - Mittleren Eintrag in Vorgängerknoten eintragen
- Beispiel: Einfügen von 36



Einfügen (III)

- Bei dieser Vorgehensweise kann auch der Vorgängerknoten überlaufen
- Knotenaufspalten rekursiv weiter nach oben fortgesetzen, evtl. bis zur Wurzel
- D.h.: B-Bäume wachsen an der Wurzel, z.B.



Löschen (I)

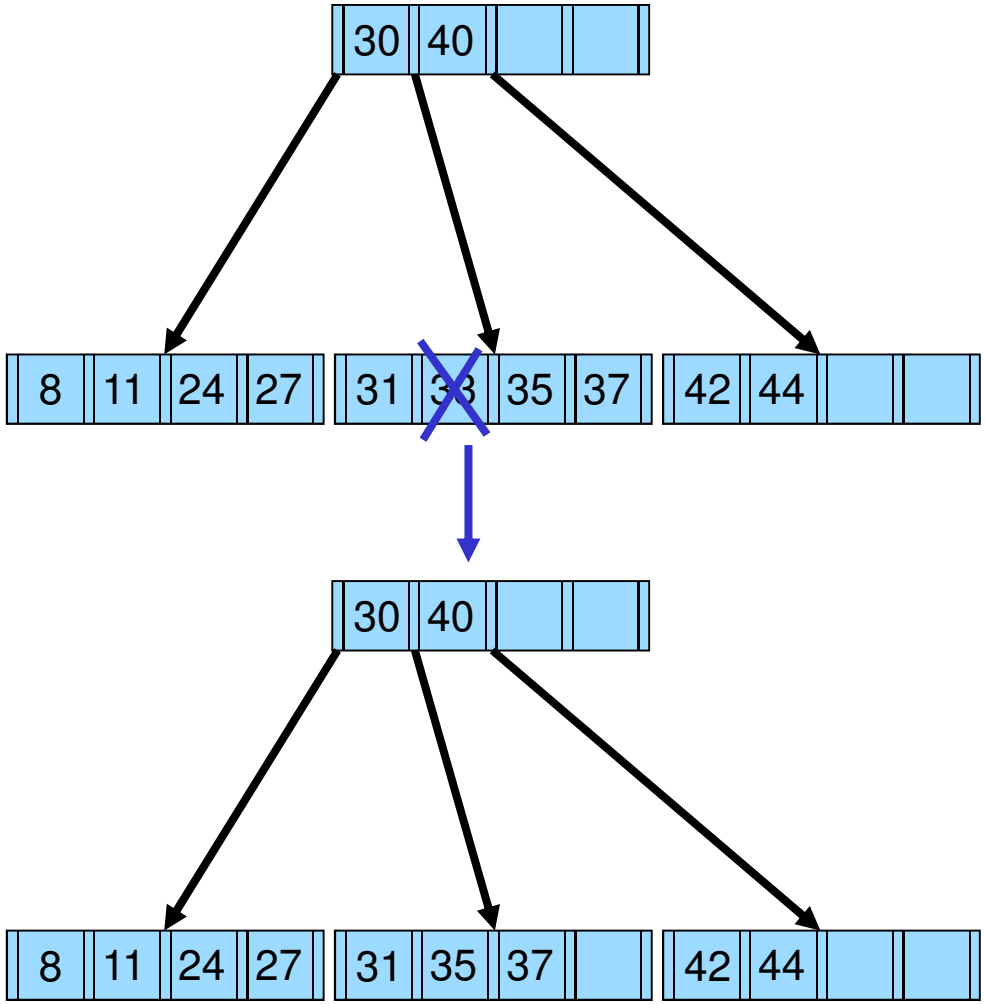
- Gegenteiliges Problem zum Einfügen kann auftreten: Unterlauf, wenn Knoten nur m Einträge besitzt
- Vorgehen:
  - Entsprechenden Knoten k suchen
  - Ist k Blatt, dann Wert löschen, evtl. Unterlauf korrigieren
  - Ist k nicht auf Blattebene, dann analog zu binärem Suchbaum k mit Inorder-Nachfolger tauschen, evtl. Unterlauf korrigieren
- Behandlung Unterlauf:

  (1) Besitzt Nachbarknoten mindestens m+1 Schlüssel, wird unter Beteiligung des Vorgängers ein Schlüssel zur Seite hinzugefügt und ein Schlüssel beim Nachbarn entfernt

  (2) Besitzen der/die Nachbarn nur m Schlüssel, wird ein Nachbarknoten mit aktuellem Knoten unter Hinzunahme des entsprechenden Schlüssels aus dem Vorgänger verschmolzen und beim Vorgänger wird rekursiv geprüft, ob dieser noch mindestens m Schlüssel besitzt, evtl. setzt sich diese Rekursion bis zur Wurzel fort

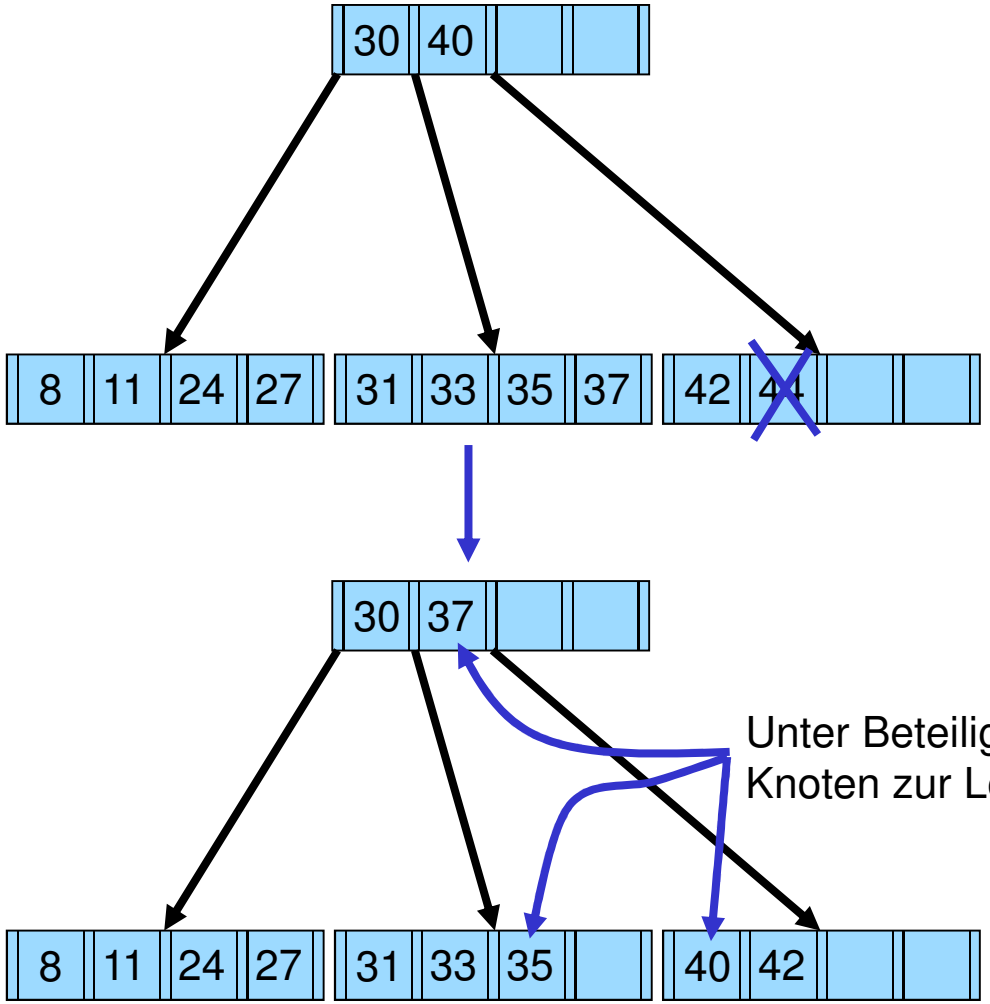
Löschen (II)

- Beispiel: Löschen ohne Unterlauf
  - Löschen von Element 33



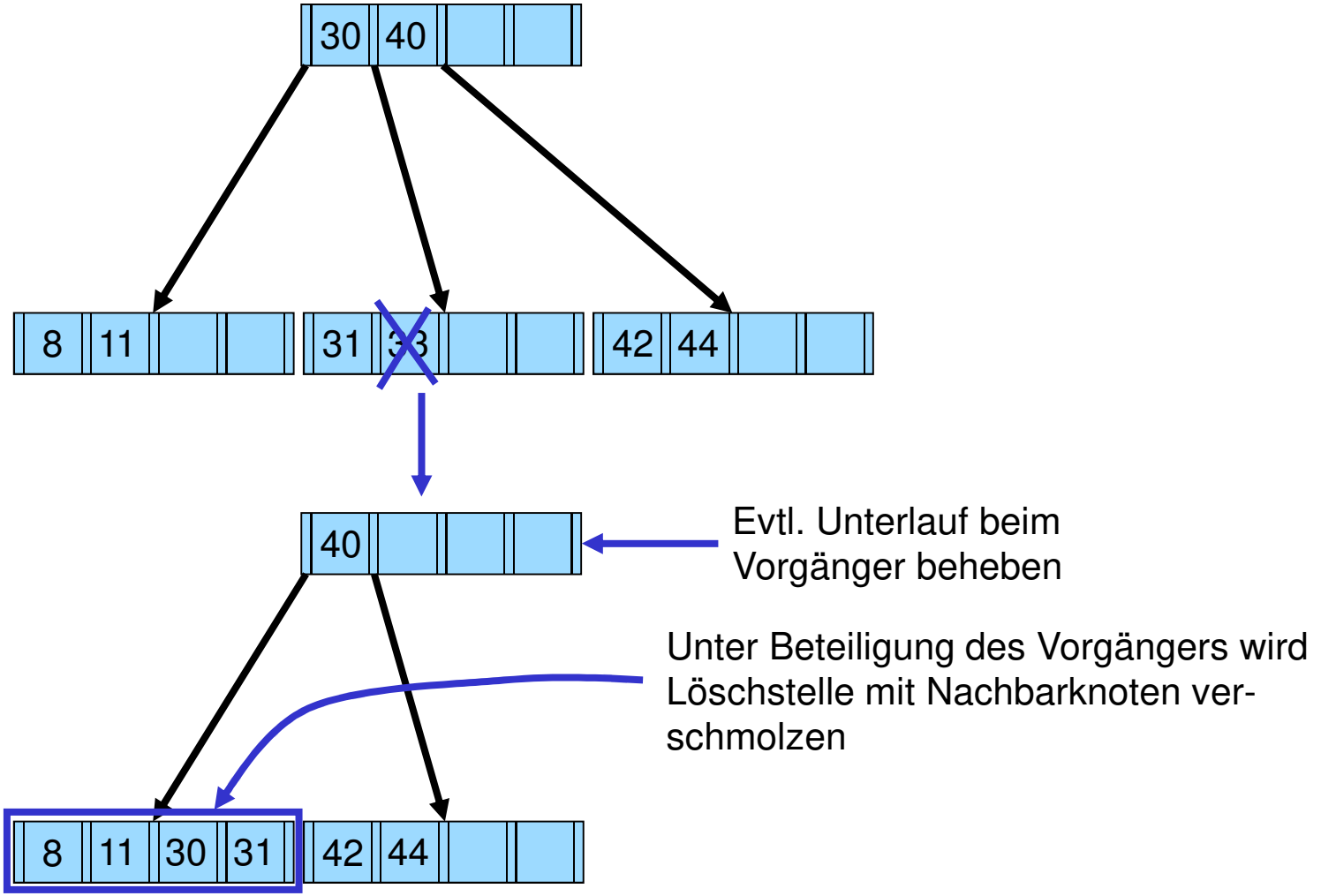
Löschen (III)

- Beispiel: Löschen mit Unterlauf, Nachbar hat mehr als m Einträge
  - Löschen von Element 44



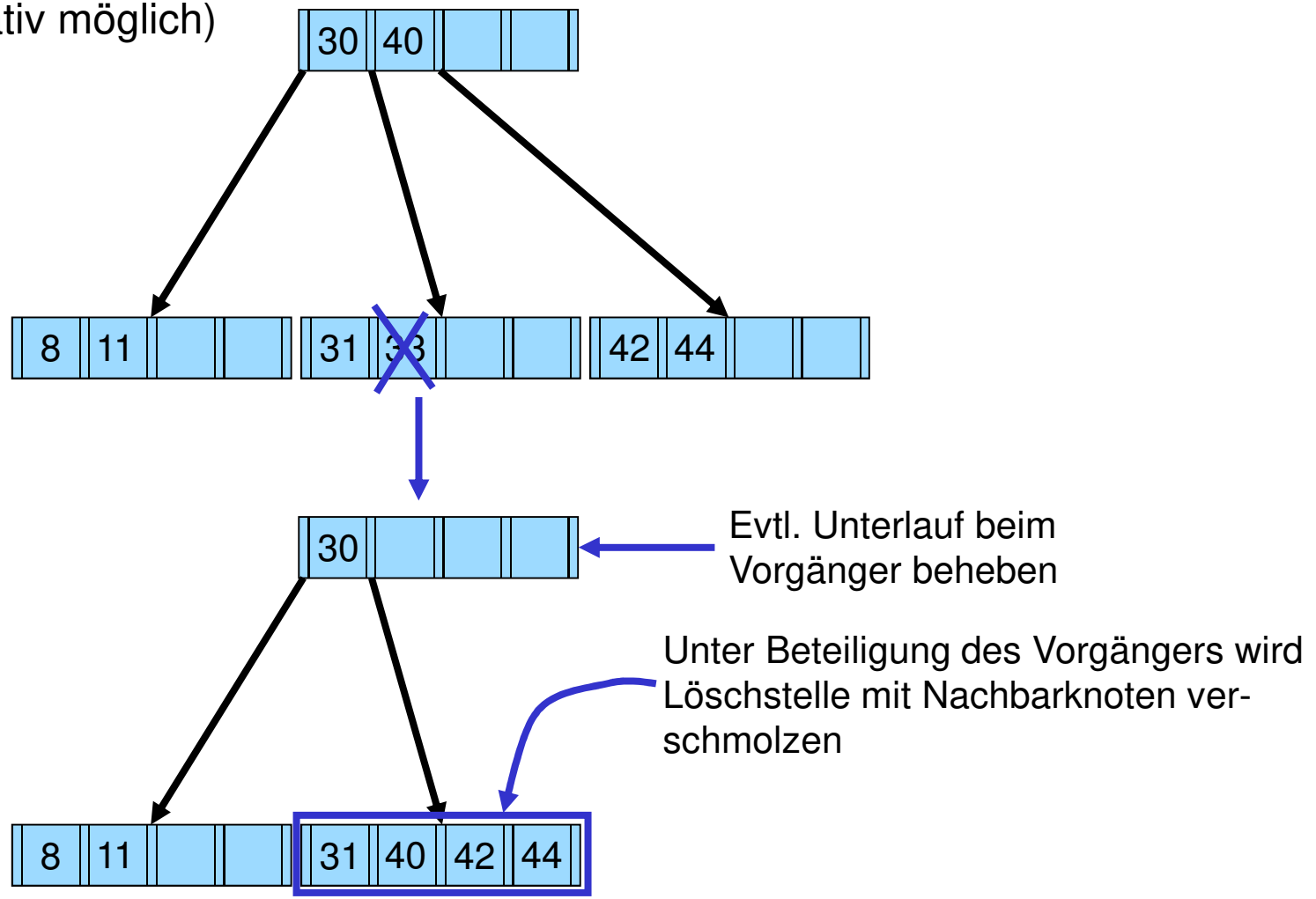
Löschen (IV)

- Beispiel: Löschen mit Unterlauf, Nachbar hat nur m Einträge
  - Löschen von Element 33



Löschen (V)

- Beispiel (Fortsetzung):
  - Löschen von Element 33; Verschmelzen mit rechtem Nachbarknoten (Alternativ möglich)



Aufwand der Operationen

- Komplexität beim Einfügen, Suchen und Löschen in einem B-Baum verlangt immer $O(\log_m(n))$ Operationen
- Entspricht Höhe des Baumes
- Ein paar konkrete Werte:
  - Größe eines Knotens sollte ca. der Seitengröße des Hintergrundspeichers entsprechen
  - B-Baum der Ordnung 8 (aus letztem Beispiel)

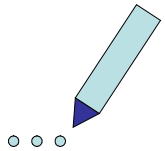
Datensätze n	Baumhöhe = Seitenzugriffe ($\log_8(n)$)
1.000	4
10.000	5
100.000	6
1.000.000	7
2.000.000	7
5.000.000	8
10.000.000	8

Übersicht

- Motivation und Definition
- Operationen auf B-Bäumen
- Beispiel
- B*-Bäume

Beispiel

- Bauen Sie einen B-Baum der Ordnung 2 mit den Werten 10, 11, 22, 17, 12, 13, 27, 25, 18, 15, 16, 19, 26 auf!
- Löschen Sie anschließend die Werte 25, 15, 13, und 22 aus dem B-Baum!



Übersicht

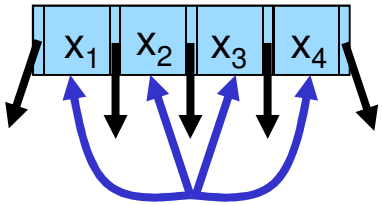
- Motivation und Definition
- Operationen auf B-Bäumen
- Beispiel
- B*-Bäume

B*-Bäume (I)

- Suchen in B-Bäumen: Durchlaufen aller Ebenen
- Pro Ebene Übertragung eines Blocks zwischen Haupt- und Plattenspeicher
- Wichtigstes Optimierungsziel: Reduzierung Anzahl der Übertragungen
- Hierzu: Höhe des Baums reduzieren, d.h. Verzweigungsgrad (und damit die Ordnung m) erhöhen
- B*-Bäume: Innere Knoten speichern Schlüsselwerte ohne ihre Restdaten
- Komplette Datensätze (Schlüssel + Restdaten) stehen nur noch in den Blättern

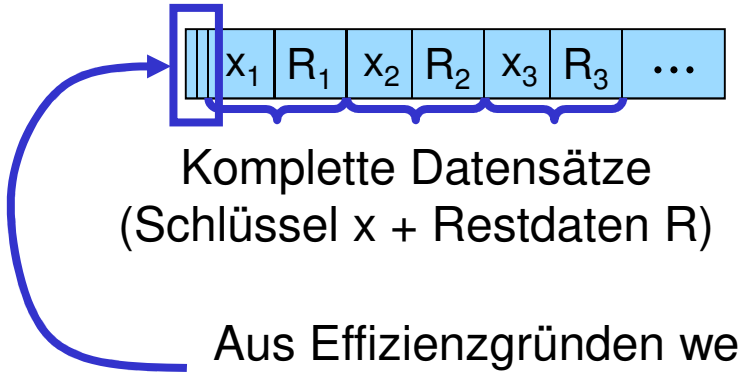
B*-Bäume (II)

- B*-Bäume besitzen dabei zwei unterschiedliche Arten von Knoten:
  - Innere Knoten:



x_1 bis x_4 sind Schlüsselwerte

  - Blattknoten:

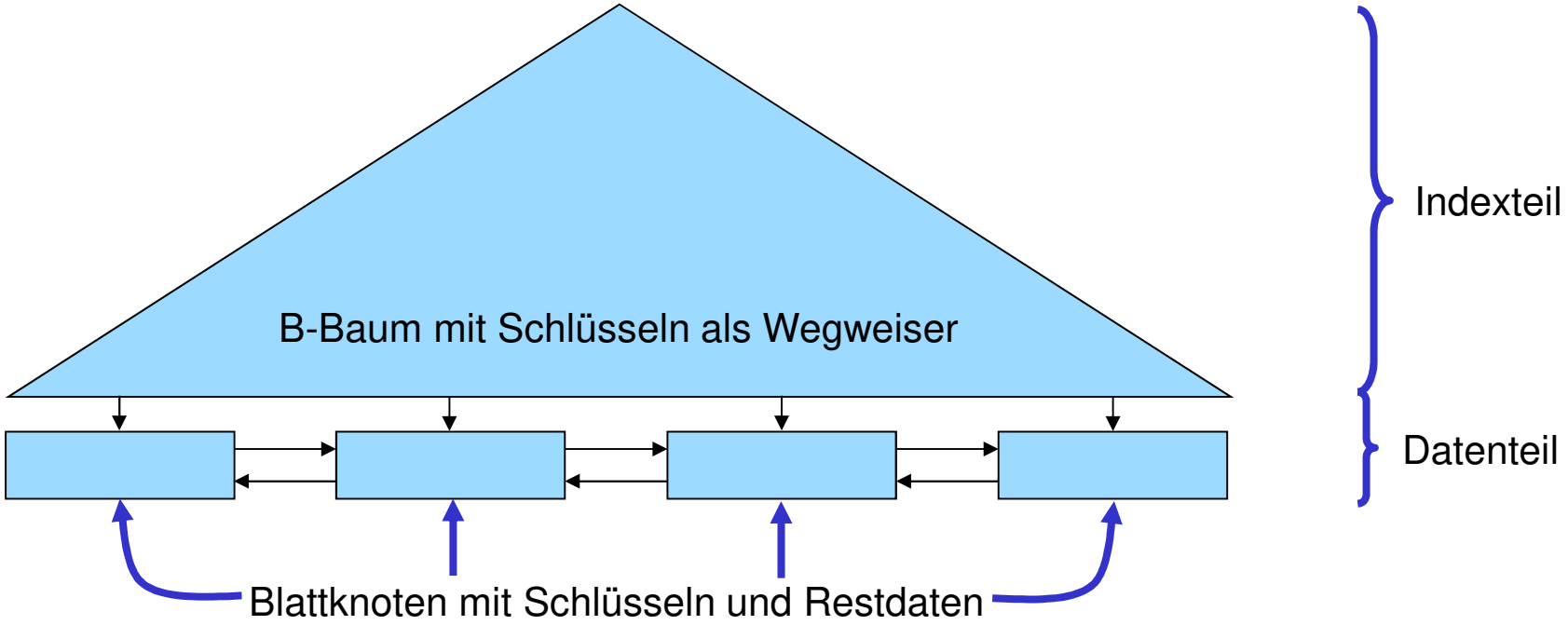


Komplette Datensätze
(Schlüssel x + Restdaten R)

Aus Effizienzgründen werden nebeneinander liegende Blätter häufig miteinander verzeigert

B*-Bäume (III)

- Insgesamt ergibt sich für B*-Bäume damit folgende Struktur:



- Folgende Variante ist (z.B. in Datenbanken) weit verbreitet:
  - Blätter beinhalten nur Schlüssel plus Verweis auf eigentlichen Datensatz

B*-Bäume (IV)

- Entscheidender Vorteil von B*-Bäumen: Erheblich mehr Einträge passen in einen Knoten
- Es passen $\left\lfloor \frac{b-z}{s+z} \right\rfloor$ Einträge in einen Knoten (d=0)
- Im Beispiel von vorhin: 102 Einträge möglich, d.h. Ordnung 51 anstelle 8
- Angewendet auf die Höhe des Baums ergibt sich:

Datensätze n	Baumhöhe Ordnung 51	Baumhöhe Ordnung 8
1.000	2	4
10.000	3	5
100.000	3	6
1.000.000	4	7
2.000.000	4	7
5.000.000	4	8
10.000.000	5	8

- Baumhöhe und damit Anzahl der Plattenzugriffe lässt sich fast halbieren

B*-Bäume (V)

- Anzahl der Nicht-Blattknoten in B*-Bäumen gegenüber B-Bäumen erheblich reduziert
- Dadurch können häufig alle inneren Knoten im Hauptspeicher gepuffert werden
- Dadurch brauchen bei der Suche keine Indexknoten übertragen werden
- Nur noch Übertragung eines einzigen Knotens (Blocks) notwendig, der die Daten enthält
- Dies ist für den Fall über den Hauptspeicher hinausgehender Datenmengen nicht mehr zu verbessern
- Ein weiterer wichtiger Vorteil von B*-Bäumen ist die leichte Handhabbarkeit von Datensätzen variabler Länge
- Aus all diesen Gründen werden in der Praxis praktisch nur B*-Bäume verwendet

Zusammenfassung

- Motivation und Definition:
  - Datenstrukturen für effizientes Suchen außerhalb des Hauptspeichers
  - Einfaches Übertragen der Suchbaum-Konzepte zu ineffizient
  - Mehrwege-Suchbäume/B-Bäume
- Operationen auf B-Bäumen:
  - Suchen
  - Einfügen
  - Löschen
  - Aufwand
- Beispiel
- B*-Bäume
  - Optimierung durch Trennen von Schlüsseln und Restdaten
  - Werden in der Praxis fast ausschließlich eingesetzt

Übungen (I)

- **Aufgabe 1**

  Gegeben sei die folgenden Zahlenfolge: 23, 66, 42, 11, 5, 77, 69, 55, 16, 84.

  a) Bauen Sie schrittweise einen B-Baum der Ordnung 2 auf!

  b) Löschen Sie aus dem B-Baum nacheinander die Einträge 66, 42 und 23.

- **Aufgabe 2**

  Erläutern Sie die Idee des B*-Baums und geben Sie an, warum mit dieser Datenstruktur gegenüber gewöhnlichen B-Bäumen eine Verbesserung erreicht wird.

- **Aufgabe 3**

  Es liegen die folgenden Daten vor: Das zugrunde liegende Betriebssystem hat eine Blockgröße von 8 kB und es sollen Datensätze verwaltet werden, die aus 16 Byte breiten Schlüssel und 400 Byte Restdaten bestehen. Ermitteln Sie die optimale Ordnung des B-Baums. Geben Sie außerdem für 1.000, 10.000, 100.000, 1.000.000, 2.000.000, 5.000.000 und 10.000.000 Datensätze die maximale Anzahl an Plattenzugriffen an!

  Wie verändern sich Ordnung und Anzahl der Zugriffe, wenn ein B*-Baum zum Einsatz kommt?

Übungen (II)

- **Aufgabe 4**

	wahr	falsch
Plattenspeicherzugriffe sind etwa halb so schnell wie Hauptspeicherzugriffe.		
Eine typische Seitengröße auf dem Plattenspeicher ist 4 kB.		
In B-Baum steht das „B“ für binär.		
B-Bäume sind eine spezielle Form von Mehrwege-Suchbäumen.		
Suchen in B-Bäumen ist von der Baumhöhe abhängig.		
Einfügen in B-Bäumen ist vom Füllungsgrad der Knoten abhängig.		
Im Gegensatz zu binären Suchbäumen wird bei B-Bäumen nicht im Blatt gelöscht.		