B-Bäume I

Annahme:

- Sei n die Anzahl der Objekte und damit der Datensätze.
- Das Datenvolumen ist zu groß, um im Hauptspeicher gehalten zu werden, z.B. $n = 10^6$.
- Datensätze auf externen Speicher auslagern, z.B. Festplatte

Beobachtung:

- Plattenspeicher als Menge von Blöcken (mit 1 - 4 KByte).
- Ein Block wird durch das Betriebssystem komplett in den Hauptspeicher übertragen.
- Diese Übertragungseinheiten werden als **Seiten** bezeichnet.

B-Bäume II

Idee:

- Konstruiere eine Baumstruktur mit der Eigenschaft
  1 Knoten des Baumes = 1 (oder mehr) Seite(n) des Plattenspeichers
- Für binäre Baumstrukturen bedeutet das:
  left oder right-Zeiger folgen = 1 Plattenspeicherzugriff

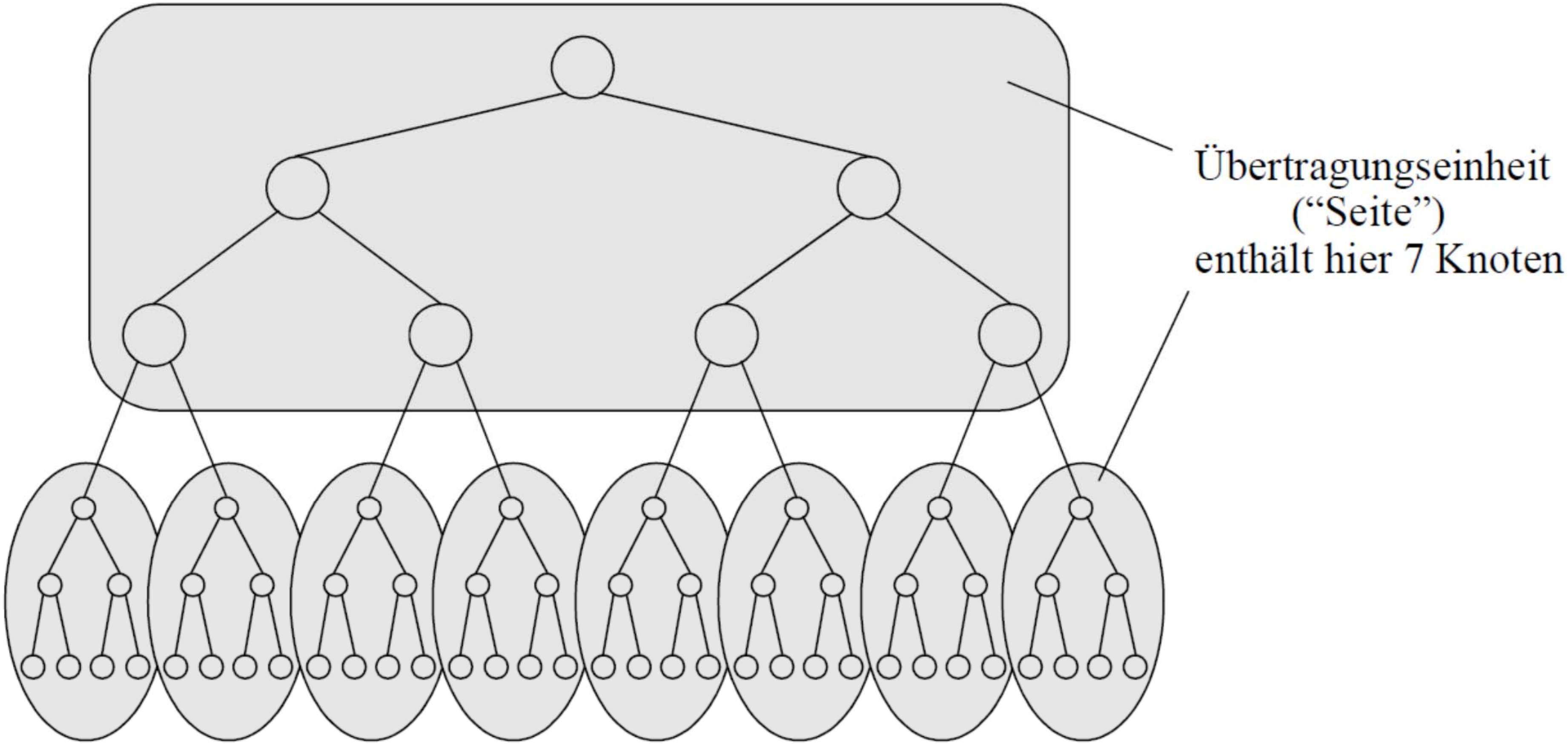
Beispiel:

Sei die Anzahl der Datensätze $n = 10^6$

$\rightarrow \log_2(10^6) = \log_2(10^3)^2 = 2 \cdot \log_2(10^3) \approx 20$ Plattenzugriffe

B-Bäume III

Idee: Zusammenfassen mehrerer binärer Knoten zu einer Seite



B-Bäume IV

Beispiel für einen B-Baum:

99 Knoten in einer Seite → 100-fache Verzweigung

→ $\log_{100}(10^6) = 3$ Plattenzugriffe

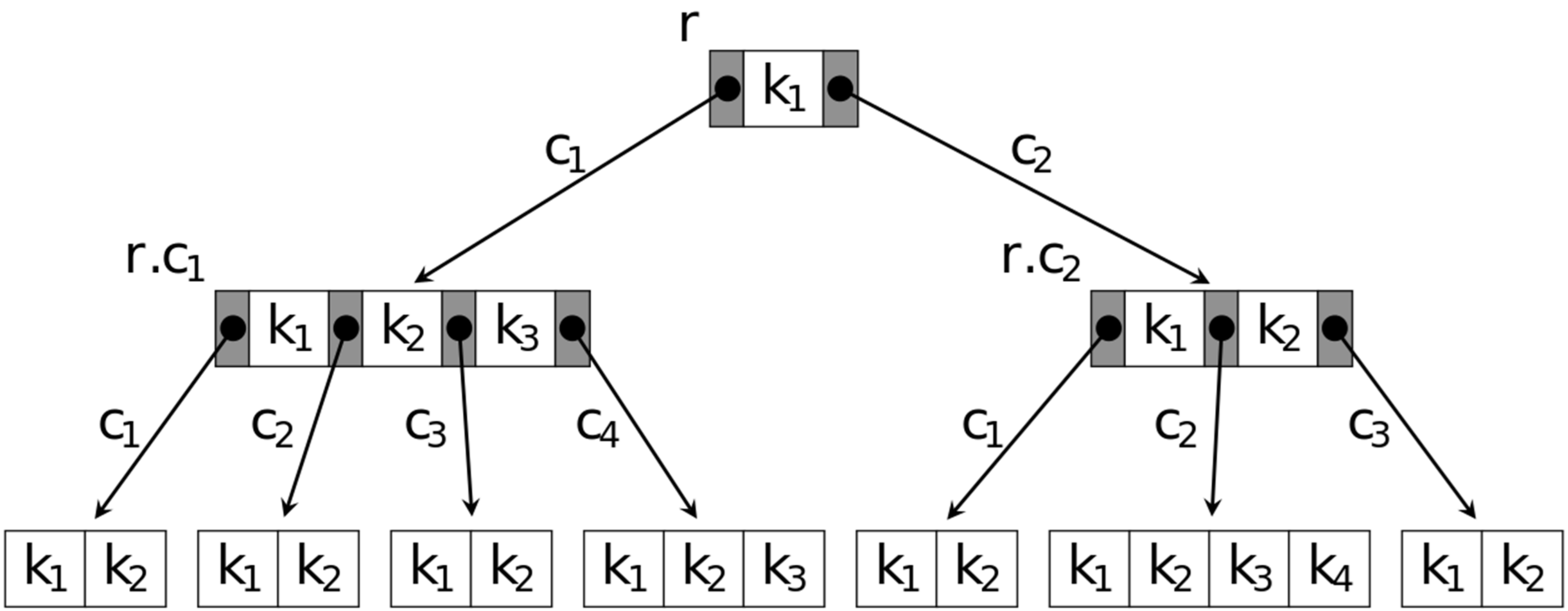
(reduziert auf 2, falls die Wurzelseite immer im Hauptspeicher liegt)

Definition: B-Baum der Ordnung m: (Bayer und McCreight 1972)

1. Jeder Knoten enthält höchstens $2m$ Schlüssel.
2. Jeder Knoten außer der Wurzel enthält mind. m Schlüssel.
3. Die Wurzel enthält mindestens einen Schlüssel.
4. Ein Knoten mit k Schlüsseln hat genau $k + 1$ Söhne.
5. Alle Blätter befinden sich auf demselben Level.

B-Bäume V

Beispiel für einen B-Baum der Ordnung 2



B-Bäume: Berechnung der maximalen Höhe I

Maximale Höhe h_{max} eines
B-Baumes der Ordnung m mit n Schlüsseln:

Level 1 hat $k_1 = 1$ Knoten.

Level 2 hat $k_2 \geq 2$ Knoten.

Level 3 hat $k_3 \geq 2(m+1)$ Knoten

…

Level $h+1$ hat $k_{h+1} \geq 2(m+1)^{h+1}$ (äußere, leere) Knoten.

B-Bäume: Berechnung der maximalen Höhe II

Ein B-Baum mit n Schlüsseln teilt den **Wertebereich** der Schlüssel in $n+1$ Intervalle. Es gilt also

$$k_{h+1} = n + 1 \geq 2(m+1)^{h+1}, \text{ d.h. } h \leq 1 + \log_{m+1}\left(\frac{n+1}{2}\right).$$

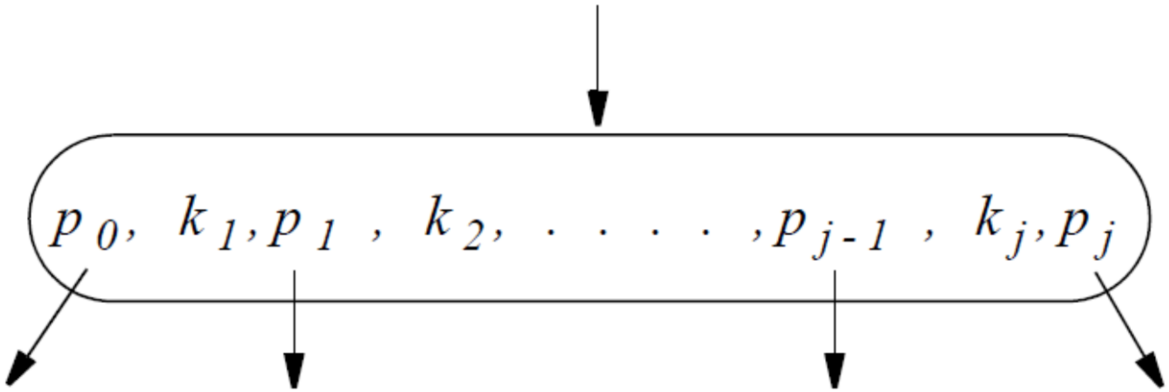
Da die Höhe immer ganzzahlig ist, und da diese Rechnung minimalen Füllgrad annimmt, folgt:

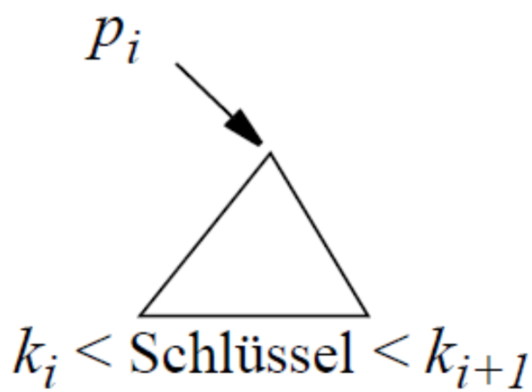
$$h \leq \left\lfloor \log_{m+1}\left(\frac{n+1}{2}\right) \right\rfloor + 1$$

Beobachtung:

Jeder Knoten (außer der Wurzel) ist mindestens mit der Hälfte der möglichen Schlüssel gefüllt. Die Speicherplatzausnutzung beträgt also mindestens 50%!

B-Bäume: Allgemeine Knotenstruktur I





wobei:

- $k_1 < k_2 < \cdots < k_j$ und $m \leq j \leq 2m$;
- p_1 zeigt auf den Teilbaum mit Schlüsseln zwischen k_i und k_{i+1}.

Anmerkung: Da die Schlüssel in jedem Knoten eines B-Baumes aufsteigend sortiert sind, kann ein Knoten nach Übertragung in den Hauptspeicher binär durchsucht werden.

B-Bäume: Allgemeine Knotenstruktur II

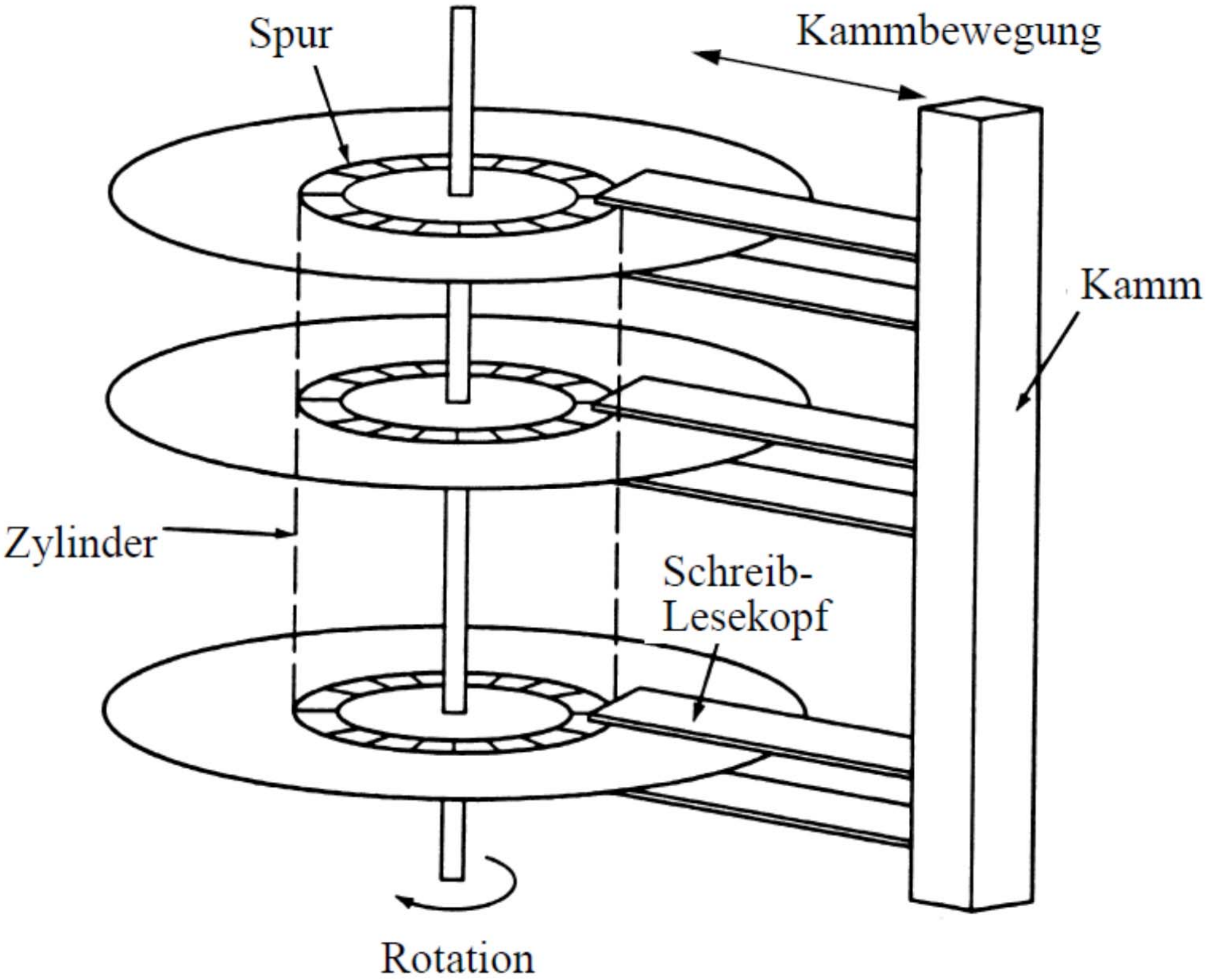
```
class Entry {
   public int key;
   public Page son;
   // ... Konstruktor ...
}
class Page {
   public int numberOfEntries;
   public Page firstSon;
   public Entry [] entries;
   // Im Konstruktor initialisieren mit new Entry(2*m + 1);
   ...
}
class Btree {
   protected Page root;
   protected int m;
   ...
}
```

B-Bäume: Reale Rechner und Plattenspeicher I

Welche Ordnung m von B-Bäumen ist auf realen Rechnern und Plattenspeichern günstig?

- Wir betrachten den physischen Aufbau eines Magnetplattenspeichers.
- Dieser besteht aus einer Reihe übereinanderliegender rotierender Magnetplatten, die in Zylinder, Spuren und Sektoren unterteilt sind.
- Der Zugriff erfolgt über einen Kamm mit Schreib-/Leseköpfen, der quer zur Rotation bewegt wird.

B-Bäume: Reale Rechner und Plattenspeicher II



B-Bäume: Reale Rechner und Plattenspeicher III

Diskussion: Flash-Speicher (z.B. SSD)

- Daten können nur blockweise gelöscht oder überschrieben werden
- Lebensdauer hängt von Anzahl der Löschvorgänge ab
  → copy-on-write B-trees
- Erase-block-size bei SSDs: 128kb – 512kb (!)

→ Random-reads sind schnell

→ Random-writes bedeuten, dass bis zu 512kb geschrieben werden müssen, um 1 bit zu löschen!

B-Bäume: Zugriffszeit I

Seitenzugriff erfolgt in mehreren Phasen

Phase	Hard Disk Drive		Solid State Disk	
	Ursache	**Zeit**	**Ursache**	**Zeit**
Positionierungszeit (PZ)	Kamm-bewegung	8 ms	Adress-berechnung	0.1 ms
Latenzzeit (LZ)	Warten auf Sektor	4 ms	---	---
Übertragungszeit (ÜZ)	Übertragung der Daten	$3{,}3 \cdot 10^{-6}$ ms / Byte	Übertragung der Daten	$1{,}0 \cdot 10^{-5}$ ms / Byte

→ **Zugriffszeit für eine Seite:** PZ + LZ + ÜZ · (Seitengröße)

B-Bäume: Zugriffszeit II

Sei die Größe eines Schlüssels durch α Bytes und die eines Zeigers durch β Bytes gegeben.

→ **Seitengröße** $\approx 2m(\alpha + \beta)$

→ **Zugriffszeit** pro Seite $\approx \text{PZ} + \text{LZ} + \text{ÜZ} \cdot 2m(\alpha + \beta) = a + b \cdot m$

mit $a = \text{PZ} + \text{LZ}$ und $b = 2(\alpha + \beta) \cdot \text{ÜZ}$

Andererseits ergibt sich für die **interne Verarbeitungszeit** pro Seite bei binärer Suche:

$c \cdot \log_2(m) + d$ für Konstanten c und d.

Die **gesamte Verarbeitungszeit** pro Seite ist damit:

$$a + b \cdot m + c \cdot \log_2(m) + d$$

(a: Suchen / $b \cdot m$: Lesen / $c \cdot \log_2(m) + d$: Verarbeiten)

B-Bäume: Zugriffszeit III

Maximale Anzahl von Seiten auf einem Suchpfad eines B-Baumes mit n Schlüsseln:

$$h_{max} = \left\lfloor \log_{m+1}\left(\frac{n+1}{2}\right) \right\rfloor + 1 = f \cdot \frac{\log_2\left(\frac{n+1}{2}\right)}{\log_2(m)} \text{ für eine Konstante } f.$$

Maximale Suchzeit MS ist damit gegeben durch die Funktion:

$$MS(m) = g \cdot \left(\frac{a+d}{\log_2(m)} + \frac{b \cdot m}{\log_2(m)} + c\right) \text{ mit } g = f \cdot \log_2\left(\frac{n+1}{2}\right).$$

Für die obigen Konstanten setzen wir beispielshaft die folgenden Werte (eines HDD) ein:

$$a = 0{,}012s,\ a + d \approx a = 0{,}012s = 12\, ms$$

$$\alpha = 8,\ \beta = 4 \rightarrow b = 24 \cdot 3,3 \cdot 10^{-6}\, ms = 79,2 \cdot 10^{-6} ms.$$

B-Bäume: Zugriffszeit IV

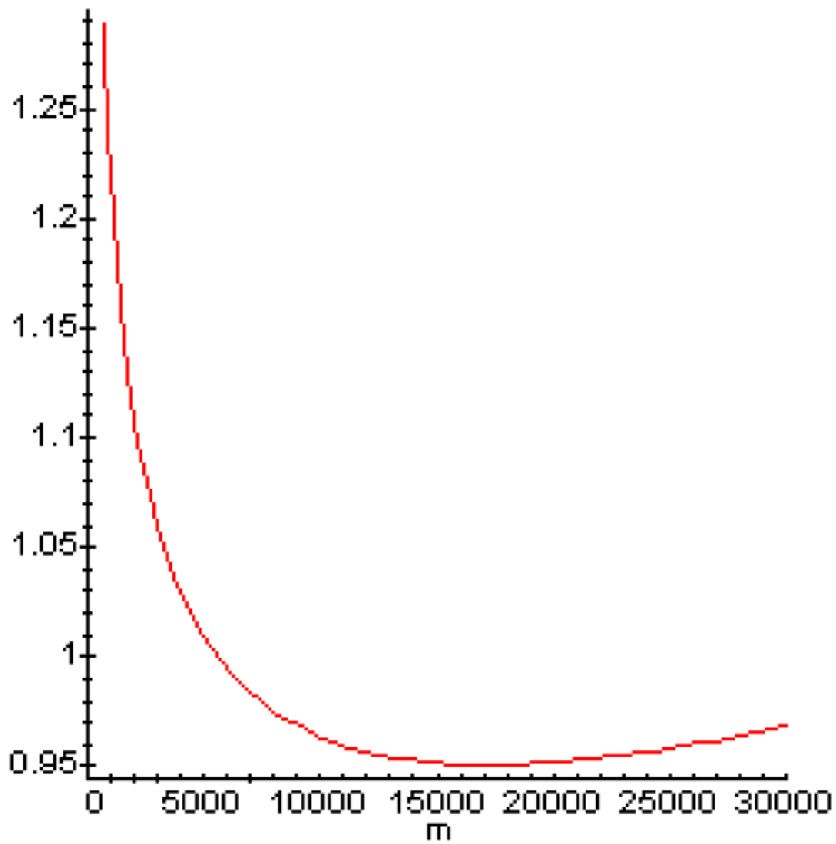
Damit ist folgende Funktion $\overline{MS(m)}$ zu minimieren:

$$\overline{MS(m)} = \left(\frac{12}{\log_2(m)} + \frac{0{,}0000792 \cdot m}{\log_2(m)}\right) ms \rightarrow \boldsymbol{MIN}$$

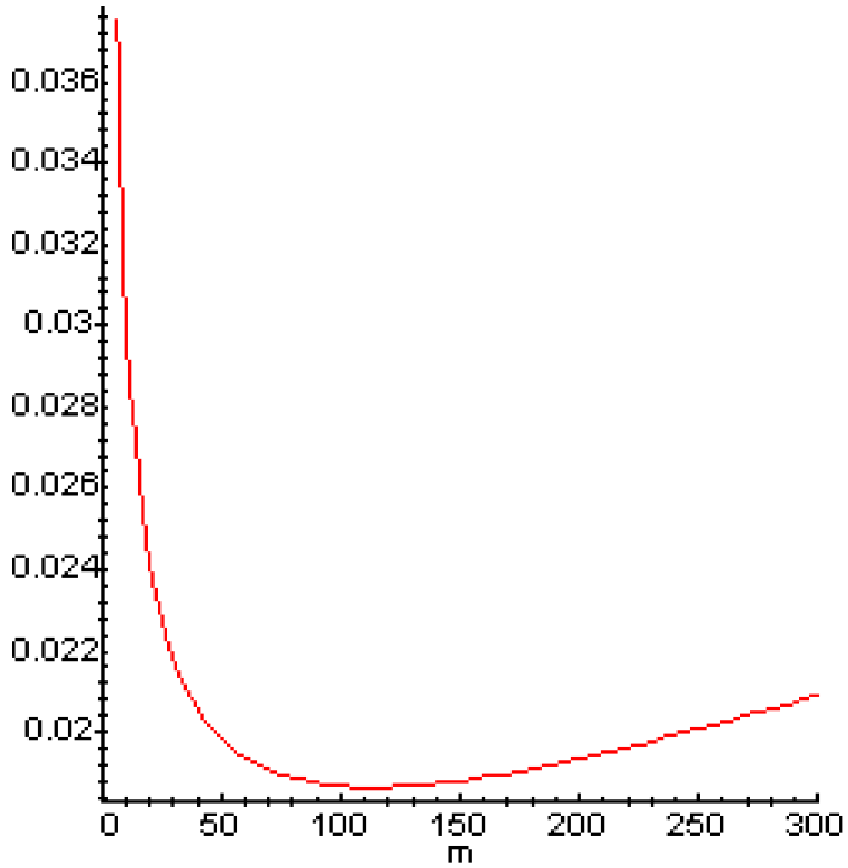
- Die maximale Suchzeit ist somit nahezu minimal für $16000 \leq m \leq 19000$.
- Dies entspricht in obigem Beispiel einer “optimalen” Seitengröße von 432 KByte.
- Für die exemplarischen Werte der Solid-State-Disk ergibt sich eine „optimale“ Seitengröße von 3,0 KB (für $m = 125$).

B-Bäume: Zugriffszeit V

$\overline{MS(m)}$ für Hard Disk Drive



$\overline{MS(m)}$ für Solid State Disk



B-Bäume: Einfügen I

Knoten K wird gesucht, in den der neue Schlüssel einzufügen ist. Dieser ist stets ein Blatt. Der Schlüssel wird in K an der entsprechenden Stelle eingefügt.
Sei s die Anzahl der Schlüssel in K nach dem Einfügen:

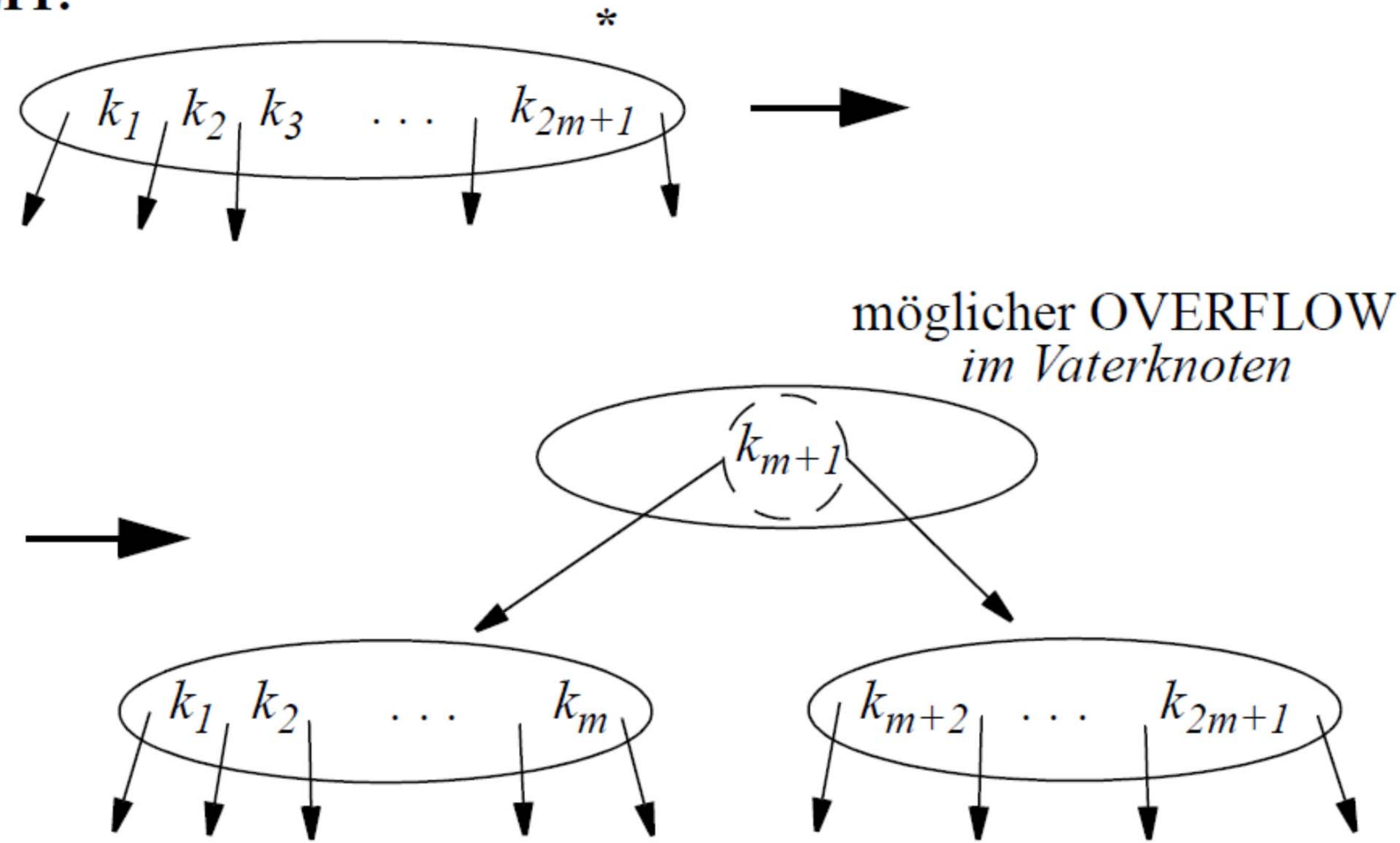
Fall 1: $s \leq 2m$: → **STOP**

Fall 2: $s = 2m + 1$ → **OVERFLOW**

- **OVERFLOW** wird behandelt durch Aufspalten (**SPLIT**) des Knotens.
- Dies kann einen OVERFLOW im Vaterknoten zur Folge haben.
- Ein OVERFLOW kann sich bis zur Wurzel des Baumes fortsetzen.
- Falls die Wurzel gesplittet wird, wächst die Höhe des Baumes um 1

B-Bäume: Einfügen II

SPLIT:



B-Bäume: Entfernen I

- Der zu entfernende Schlüssel *k* wird im Baum gesucht und aus dem gefundenen Knoten *K* gelöscht.
- Falls *K* kein Blatt ist, wird der entfernte Schlüssel durch den Schlüssel *p* ersetzt, der der kleinste Schlüssel im Baum ist, der größer als *k* ist.
- Sei *P* der Knoten, in dem *p* liegt. Dann ist *P* ein Blatt, aus dem *p* nun entfernt wird.
- Auf diese Weise wird der Fall "Löschen in einem inneren Knoten" auf den Fall "Löschen in einem Blatt" zurückgeführt.

Sei s die Anzahl der Schlüssel in K nach dem Entfernen:

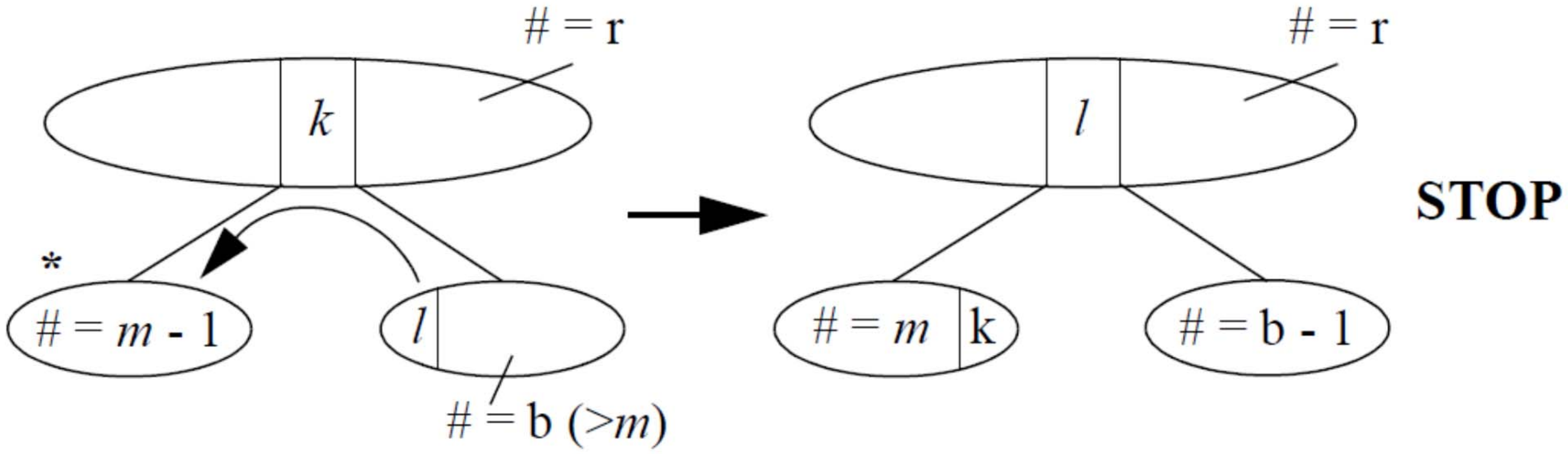
Fall 1: $s \geq m$: → **STOP**

Fall 2: $s = m - 1$ → **UNDERFLOW**

B-Bäume: Entfernen II

UNDERFLOW-Behandlung:

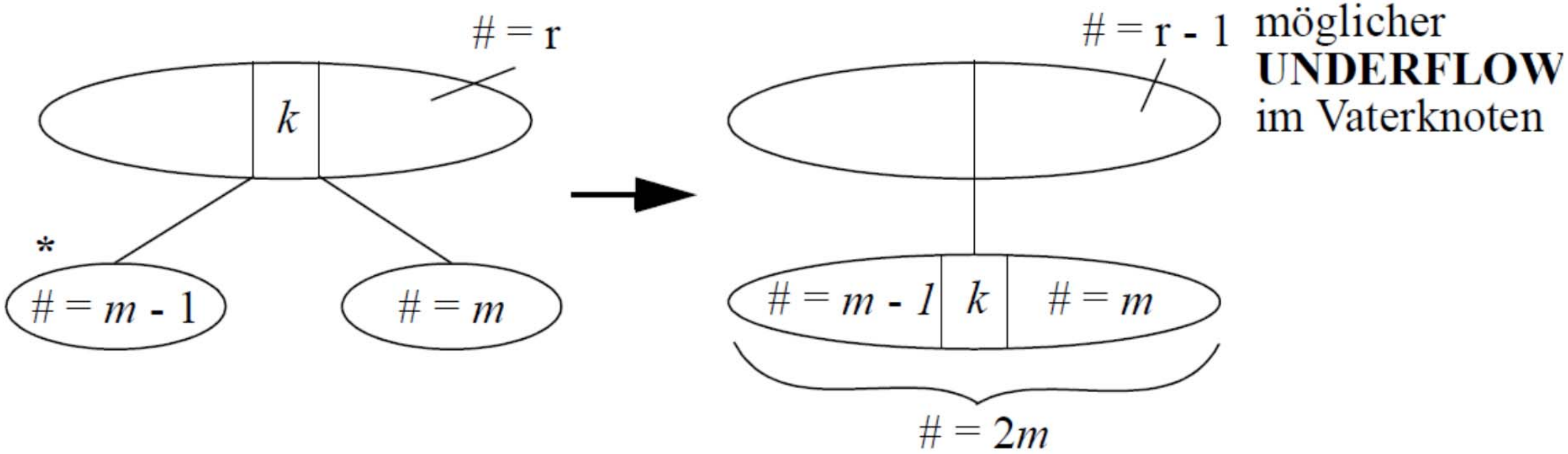
- **Fall 1:** Bruder hat $\geq m + 1$ Schlüssel
  $\rightarrow$ **Ausgleichen** mit dem Bruder



(Bruder bezeichnet dabei stets nur den rechten Bruder, falls vorhanden, andernfalls den linken Bruder!)

B-Bäume: Entfernen III

- **Fall 2:** Bruder hat m Schlüssel
  → **Verschmelzen** mit dem Bruder unter Hinzunahme des trennenden Vaterschlüssels
  → möglicher **UNDERFLOW** im Vaterknoten



- Auch die UNDERFLOW Behandlung bis zur Wurzel des Baumes fortsetzen.
- Wird aus der Wurzel des Baumes der letzte Schlüssel entfernt, so wird diese gelöscht → die Höhe des Baumes verringert sich um 1

B-Bäume: Aufwandsabschätzung I

Durchschn. Anzahl von Aufspaltungen (Splits) pro Einfügung:

Bezeichne s die durchschnittliche Anzahl der Knoten-Splits pro

Modell:

- Ausgehend von einem leeren Baum wird ein B-Baum der Ordnung m für die Schlüssel $k_1, k_2, \ldots, k_n$ durch n aufeinanderfolgende Einfügungen konstruiert.
- Sei t die Gesamtzahl der Aufspaltungen bei der Konstruktion dieses B-Baumes $\rightarrow s = \frac{t}{n}$
- Sei p die Anzahl der Knoten (Seiten) des B-Baumes mit $p \geq 3$ $\rightarrow t < p - 1$ (genauer: $t \leq p - 2$)
- Für die Anzahl n der Schlüssel in einem B-Baum der Ordnung m mit p Knoten gilt: $n \geq 1 + (p - 1) \cdot m$

B-Bäume: Aufwandsabschätzung II

- $\rightarrow p - 1 \leq \frac{n-1}{m} \rightarrow s = \frac{t}{n} < \frac{p-1}{n} \leq \frac{1}{n} \cdot \frac{n-1}{m} < \frac{1}{m}$

  wobei im allg.: $16000 \leq m \leq 19000$ (s.o.)

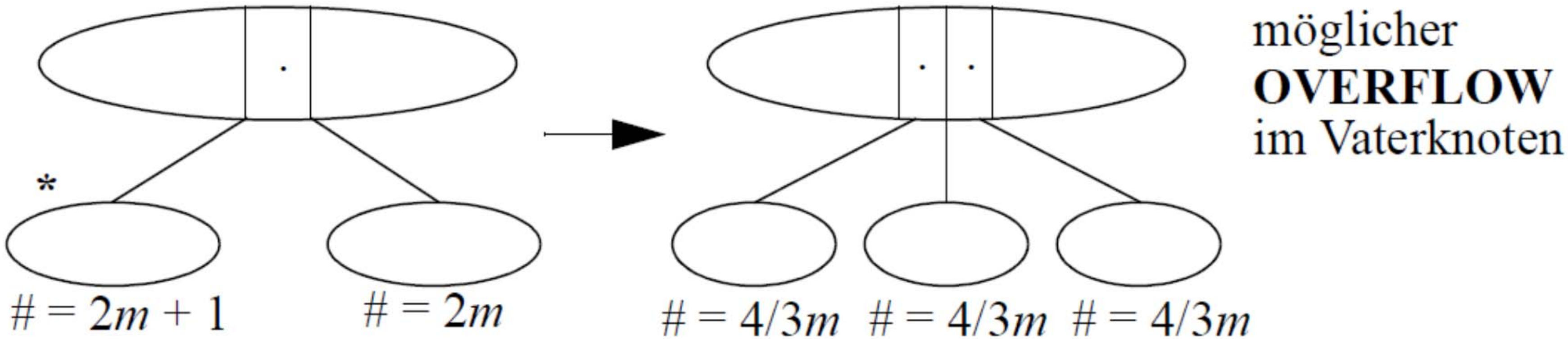
- Es gilt: $t \leq p - 2 \rightarrow t < p - 1$ für $p \geq 3$

  Denn:
  - Bei jeder Aufspaltung wird mindestens ein zusätzlicher Knoten geschaffen
  - Aufspalten der Wurzel → 2 zusätzliche Knoten
  - Wenn ein B-Baum mehr als einen Knoten hat, dann ist die Wurzel mindestens einmal aufgespalten worden
  - Dem ersten Knoten des B-Baums geht keine Aufspaltung voraus

B*-Bäume I

B*-Bäume verhalten sich im Wesentlichen wie B-Bäume, jedoch wird bei **OVERFLOW** eines Knotens dieser nicht gleich aufgespalten, sondern wie beim UNDERFLOW der Bruder betrachtet (der rechte, falls vorhanden):

- **Fall 1:** Bruder hat $b \leq 2m - 1$ Schlüssel
  → Ausgleichen mit dem Bruder
- **Fall 2:** Bruder hat $b = 2m$ Schlüssel
  → Verteilen auf drei Knoten



B*-Bäume II

Definition: B*-Baum der Ordnung m

m sei ein Vielfaches von 3.

1. Jeder Knoten außer der Wurzel enthält mindestens $\frac{4}{3}m$, höchstens $2m$ Schlüssel.
2. Die Wurzel enthält mindestens einen, höchstens $\frac{8}{3}m$ Schlüssel.
3. Ein Knoten mit k Schlüsseln hat genau $k+1$ Söhne.
4. Alle Blätter befinden sich auf dem selben Level.

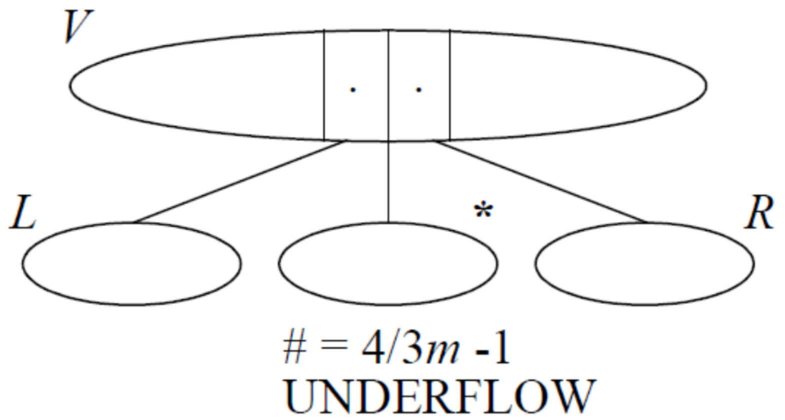
B*-Bäume besitzen eine **Speicherplatzausnutzung** von mindestens 66%.

B*-Bäume: Maximale Höhe

Nach einer zum B-Baum äquivalenten Berechnung ergibt sich für die maximale Höhe h_{max} eines B*-Baumes der Ordnung m mit n Schlüsseln:

$$h_{max} = \left\lfloor \log_{\frac{4}{3m}+1} \left(\frac{n+1}{2} \right) \right\rfloor + 1$$

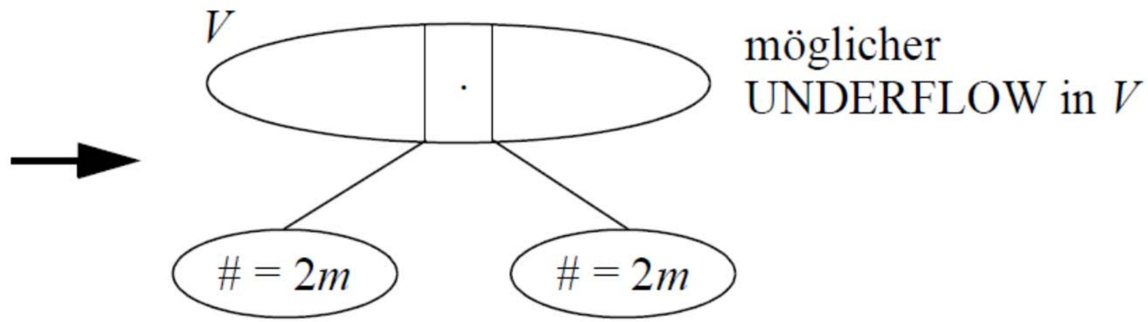
B*-Bäume: Entfernen I



Fall 1: R hat mehr als $\frac{4}{3}m$ Schlüssel → Ausgleichen von * und R

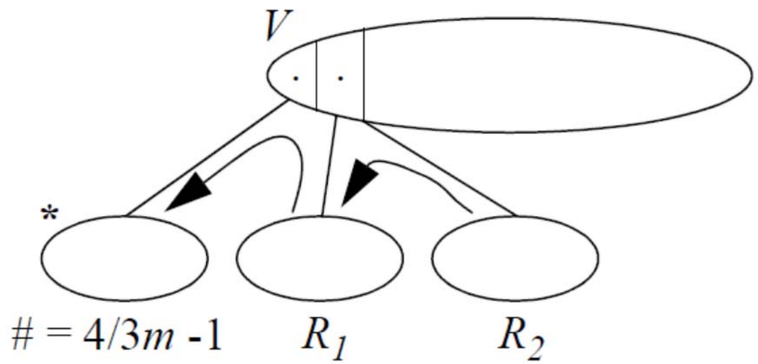
Fall 2: R hat $\frac{4}{3}m$ Schlüssel:

- **2.1:** L hat mehr als $\frac{4}{3}m$ Schlüssel → Ausgleichen von * und L
- **2.2:** L hat $\frac{4}{3}m$ Schlüssel:



B*-Bäume: Entfernen II

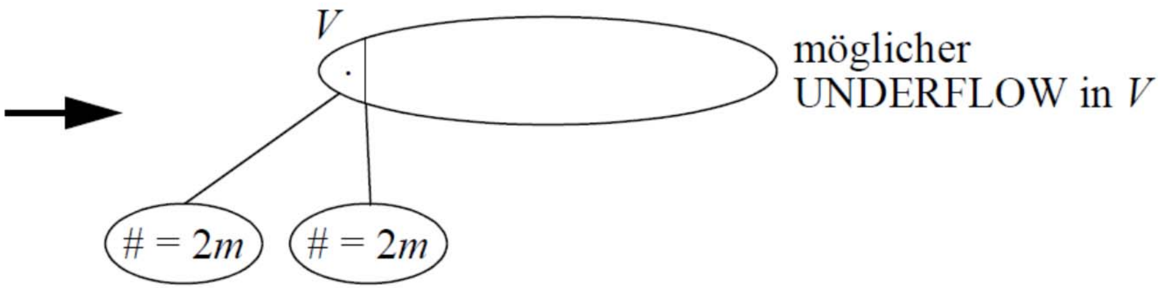
Fall 3: * hat nur einen direkten Bruder, aber weitere indirekte Brüder:



3.1: Falls R_1 mehr als $\frac{4}{3}m$ Schlüssel besitzt → Ausgleichen von * und R_1

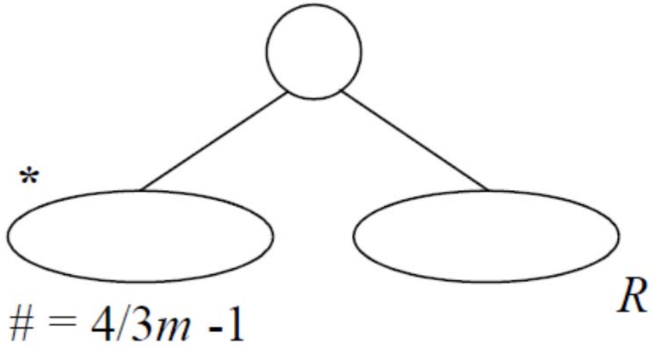
3.2: Falls R_1 genau $\frac{4}{3}m$ Schlüssel besitzt und R_2 mehr als $\frac{4}{3}m$ Schlüssel besitzt → zuerst Ausgleichen von R_1 und R_2, dann von * und R_1

3.3: Falls R_1 und R_2 jeweils genau $\frac{4}{3}m$ Schlüssel besitzen

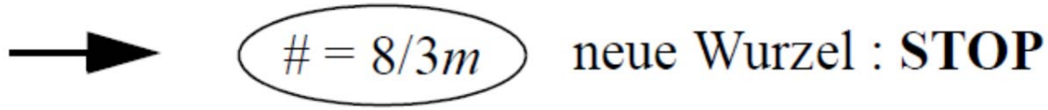


B*-Bäume: Entfernen III

Fall 4: * ist Sohn einer binären Wurzel



4.1: R hat $\frac{4}{3}m$ Schlüssel → Ausgleichen von * mit R



4.2: R hat mehr als $\frac{4}{3}m$ Schlüssel → Ausgleichen von * und R

B-Bäume und B*-Bäume

Korollar:

B-Bäume und B*-Bäume bilden eine Klasse balancierter Suchbäume.

Bemerkung:

- In fast allen gängigen Datenbanksystemen sind B-Bäume, B*-Bäume oder deren Varianten zur effizienten Ausführung von Suchoperationen implementiert.
- Für SSDs gibt es spezielle Weiterentwicklungen