B-Bäume, Hashtabellen, Cloning/Shadowing, Copy-on-Write

Sommersemester 2013

Thomas Maier

Proseminar: Ein- / Ausgabe – Stand der Wissenschaft

Gliederung

1. Hashtabelle

- verwaltet Daten.

- Hashfunktion bildet Datei auf Integer ab.

- Wert entspricht Arrayindex.

- Direktes Einfügen und Löschen möglich.

- $Aufwand$: $O(1)$.

Idealfall: Eine Datei pro Index

$$Hashfunktion\ f(\mathrm{x}) = \begin{cases} 0, & falls\ x = a \\ 1, & falls\ x = b \\ 2, & falls\ x = c \end{cases}$$

Nachteil: Viele Kollisionen
$Aufwand \rightarrow O(n)$.

Index $f(x)$	**Datei**
0	a
1	b
2	c

2. B-Baum

- Daten- / Indexstruktur.

- Wird häufig in Datenbanken eingesetzt.

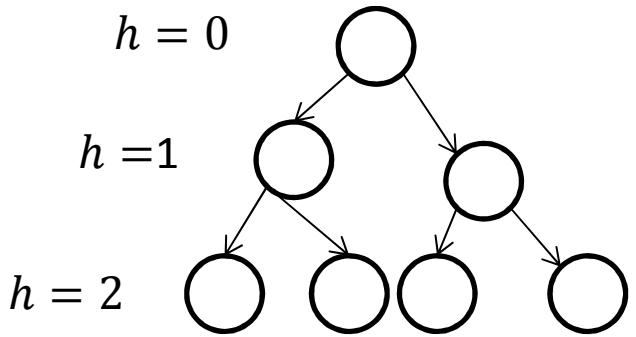
- 1972 von Rudolf Bayer und Edward M. McCreight entwickelt.

- Vollständig balancierter Baum.

2.1 Motivation

Binärer Baum:
- Jeder Knoten hat höchstens 2 Pfade.

- $Aufwand\ O(h)$

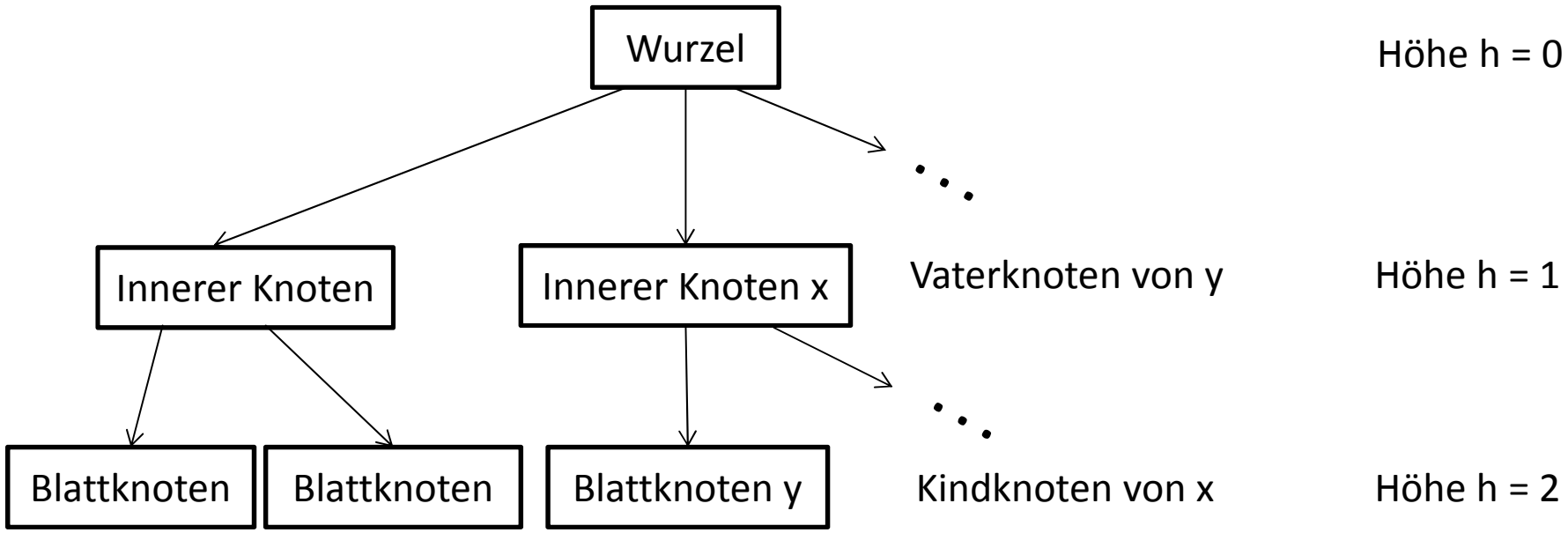


$$h \geq \log_2(n+1) - 1$$

Problem: Zu langsam.

2.2 Definition (allg.)

Allgemein:



2.2 Definition (B-Baum)

- Variable Anzahl s von Schlüssel pro Knoten.

- Innere Knoten haben $s + 1$ verweise auf Kindknoten.

Beispiel eines Knoten:

$\|\|$	k_1	$\|\|$	k_2	$\|\|$	k_3	$\|\|$

k_i *der größe nach geordnet*

Ordnung des Baumes

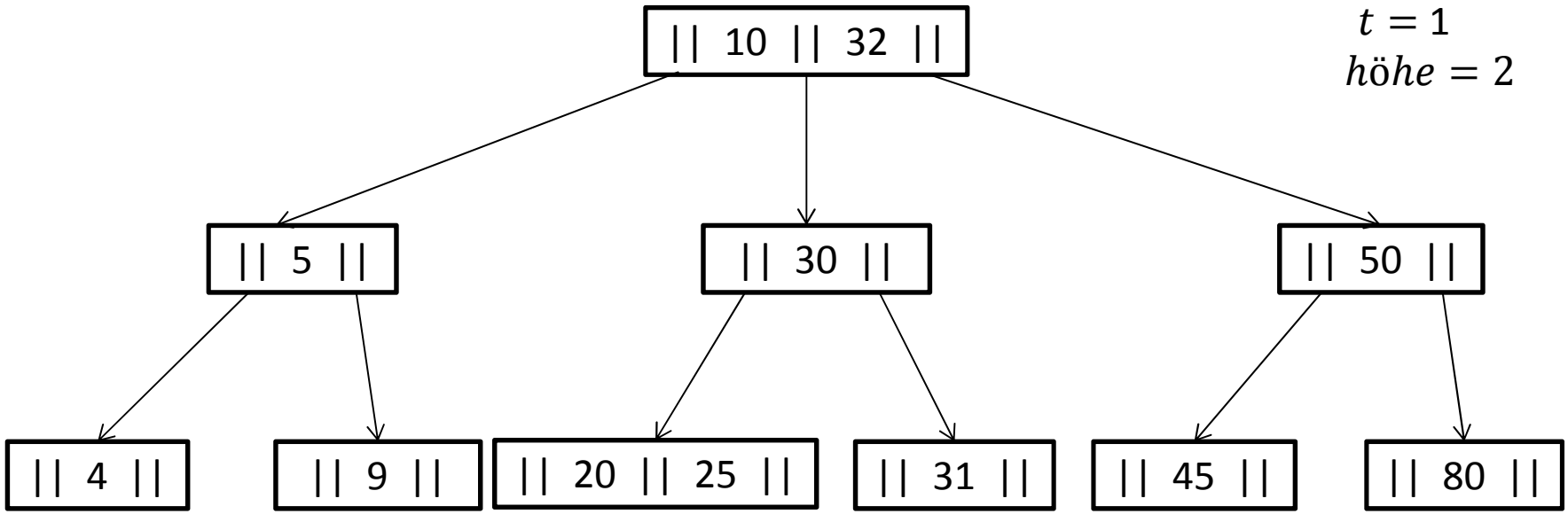
Für jeden Knoten außer der Wurzel gilt:
$t \leq Anzahl\ der\ Schlüssel \leq 2t$

Für die Wurzel gilt:
$1 \leq Anzahl\ der\ Schlüssel \leq 2t$

2.2 Definition (B-Baum)

Vollständig ausbalanciert.

Beispiel:



$t = 1$
$höhe = 2$

2.3 Suchen von Daten

Suche Schlüssel k.

- Start in der Wurzel

Sucht kleinsten Schlüssel k_i | $k \leq k_i$.

1. Fall: $k = k_i$
Schlüssel k wurde gefunden.

2. Fall: $k < k_i$
Gehe in Unterbaum, links von k_i und wiederhole.

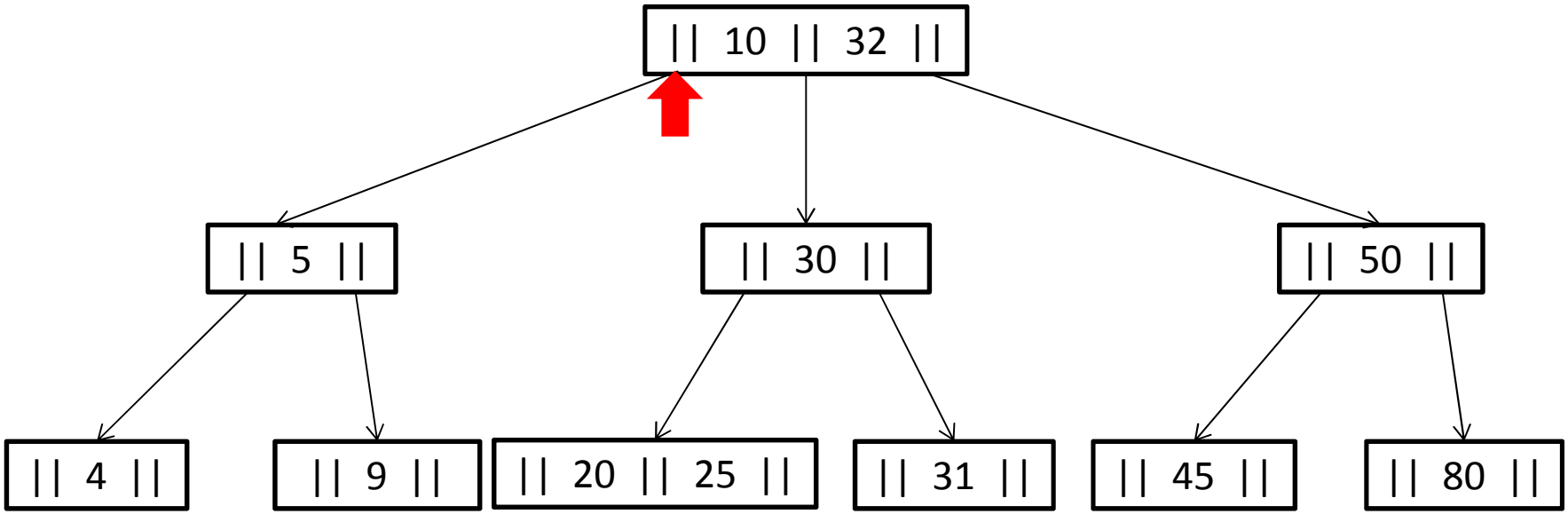
3. Fall: Es existiert kein solches k_i
Gehe in Unterbaum ganz rechts und wiederhole.

- Endet erfolglos, wenn 2. / 3. Fall eintritt und kein Unterbaum existiert.

2.3 Suchen von Daten(Beispiel)

Suche Schlüssel k = 31.

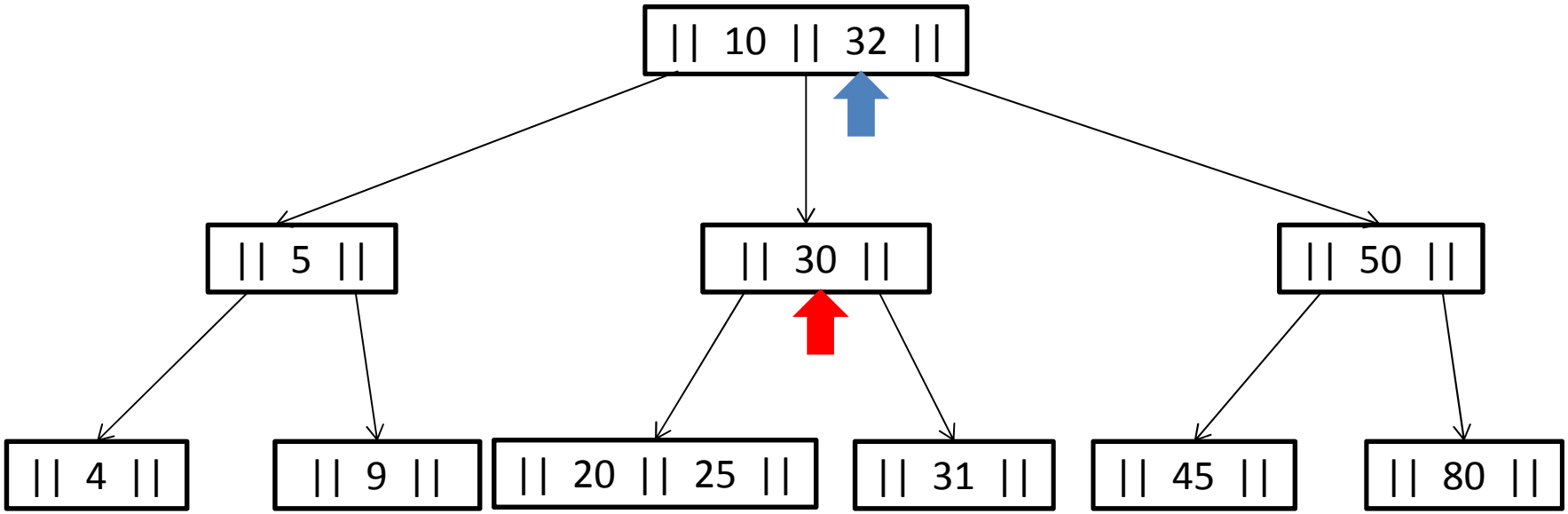
Starten in der Wurzel (roter Pfeil)



2.3 Suchen von Daten(Beispiel)

Suche Schlüssel k = 31.

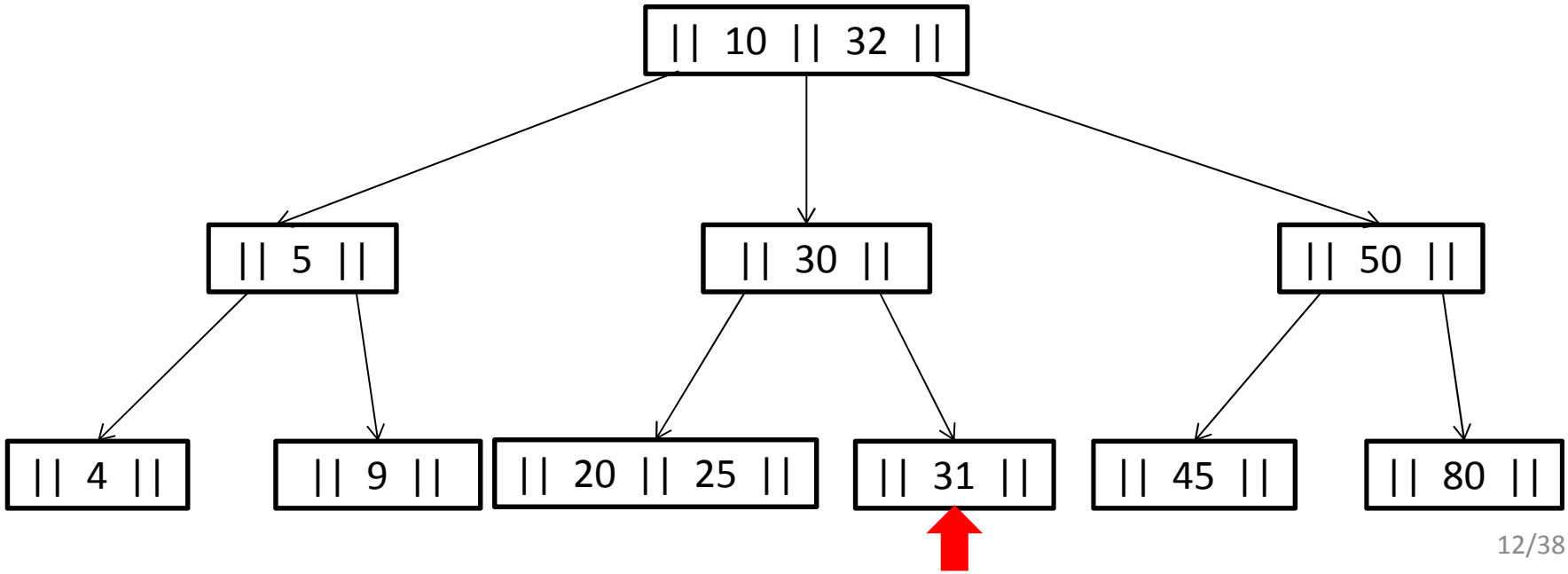
$Suchen\ k_i\ mit\ der\ Eigenschaft\ k \leq k_i$ (blauer Pfeil)
2. Fall: $k < k_i$ (roter Pfeil)



2.3 Suchen von Daten(Beispiel)

Suche Schlüssel k = 31.

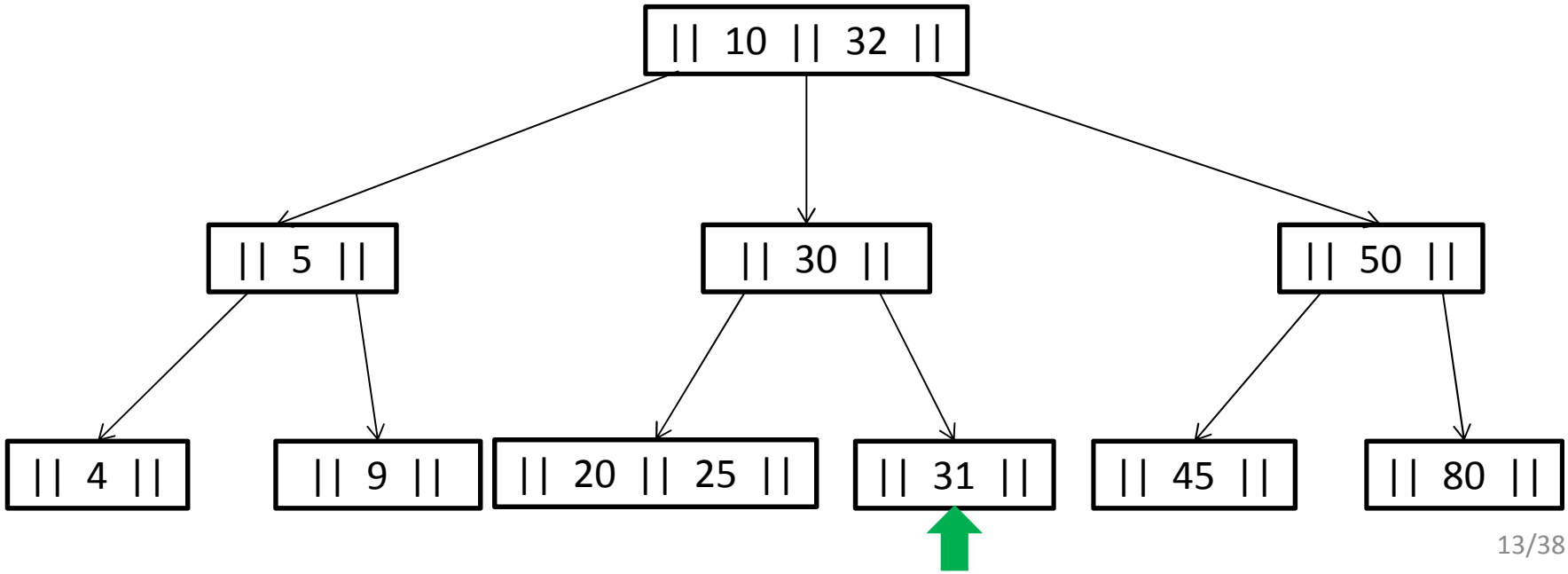
3. Fall: Es existiert kein solches k_i



2.3 Suchen von Daten(Beispiel)

Suche Schlüssel k = 31.

1. Fall: $k = k_i$



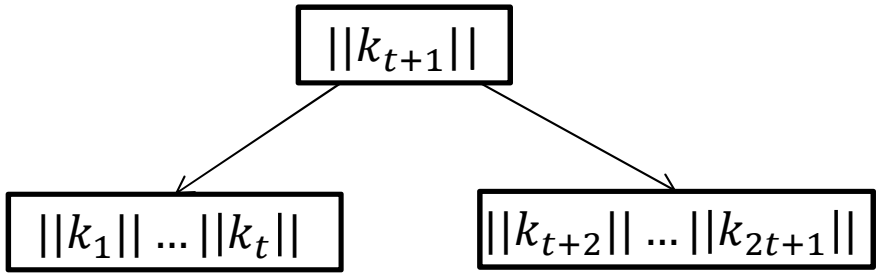
2.4 Einfügen von Daten

Füge k ein.

1. Schritt: Suche k wie in 2.3

1. Fall: k gefunden
⇒ Einfügen abbrechen.

2. Fall: k nicht gefunden
⇒ Suche endet im Blattknoten.



Abbildung

2. Schritt:

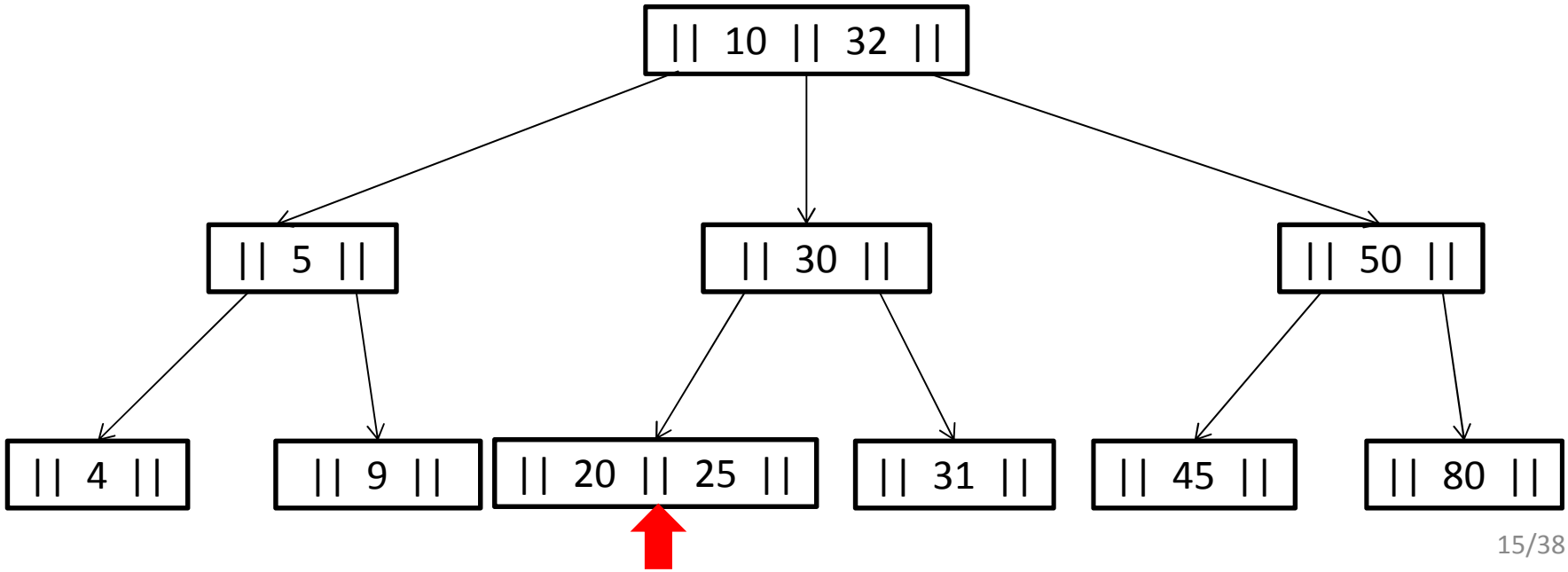
1. Fall: Knoten nicht voll
⇒ füge k ein.

2. Fall: Knoten voll (2t Schlüssel).
⇒ 1. Füge k ein.
2. Splitte Knoten wie in *Abbildung*.
3. Füge k_{t+1} rekursiv in Vaterknoten ein.

2.4 Einfügen von Daten(Beispiel)

Füge $k = 23$ ein.

1. Schritt: Suche k wie in 1.2.

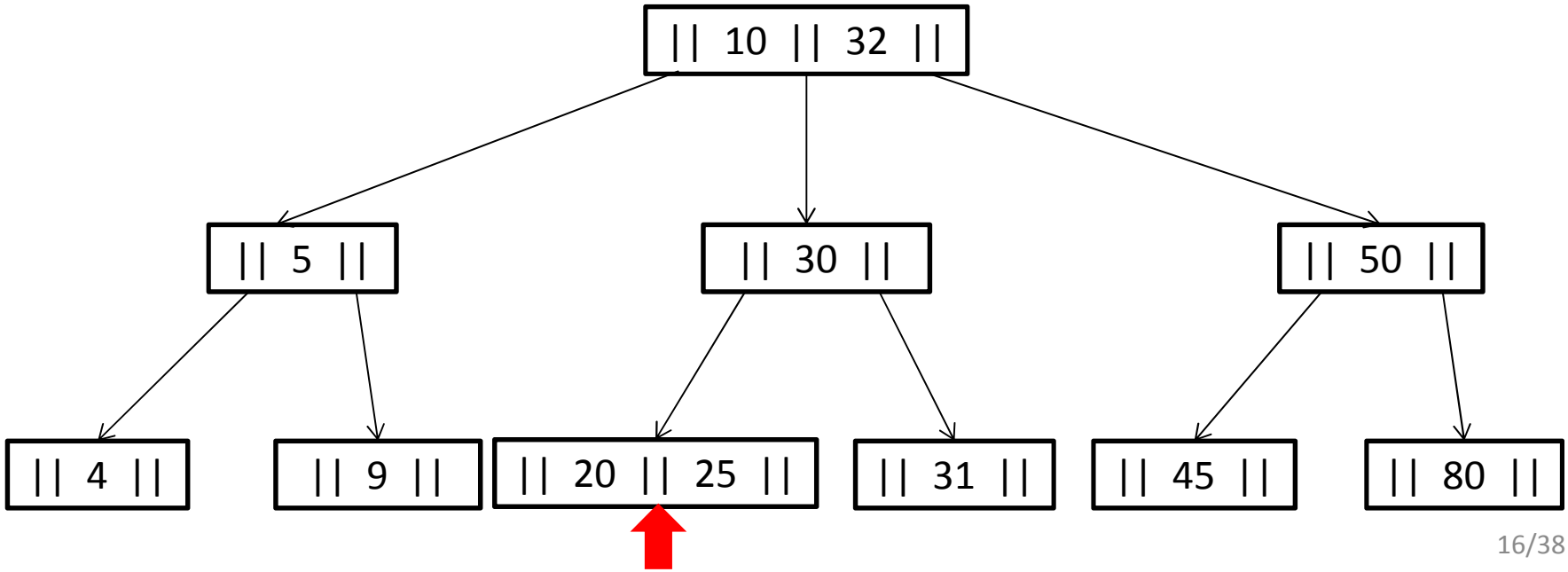


2.4 Einfügen von Daten(Beispiel)

Füge $k = 23$ ein.

2. Schritt:

2. Fall: Knoten voll.



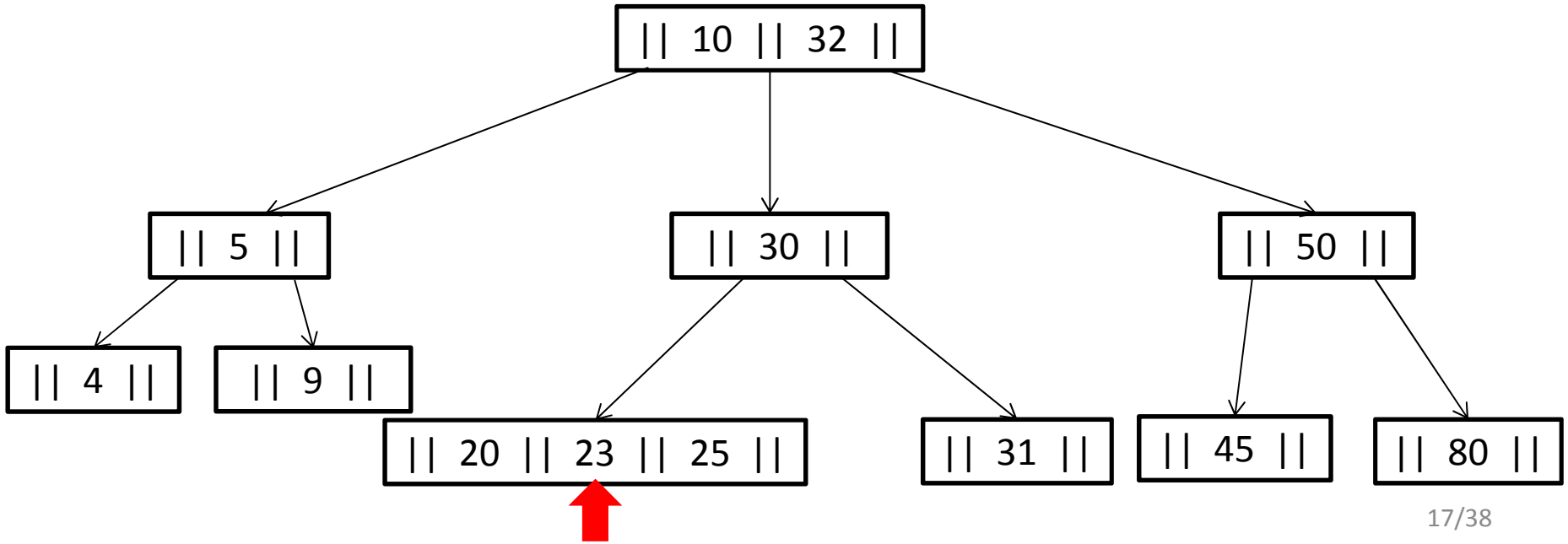
2.4 Einfügen von Daten(Beispiel)

Füge $k = 23$ ein.

2. Schritt:

2. Fall:

1. Füge 23 ein



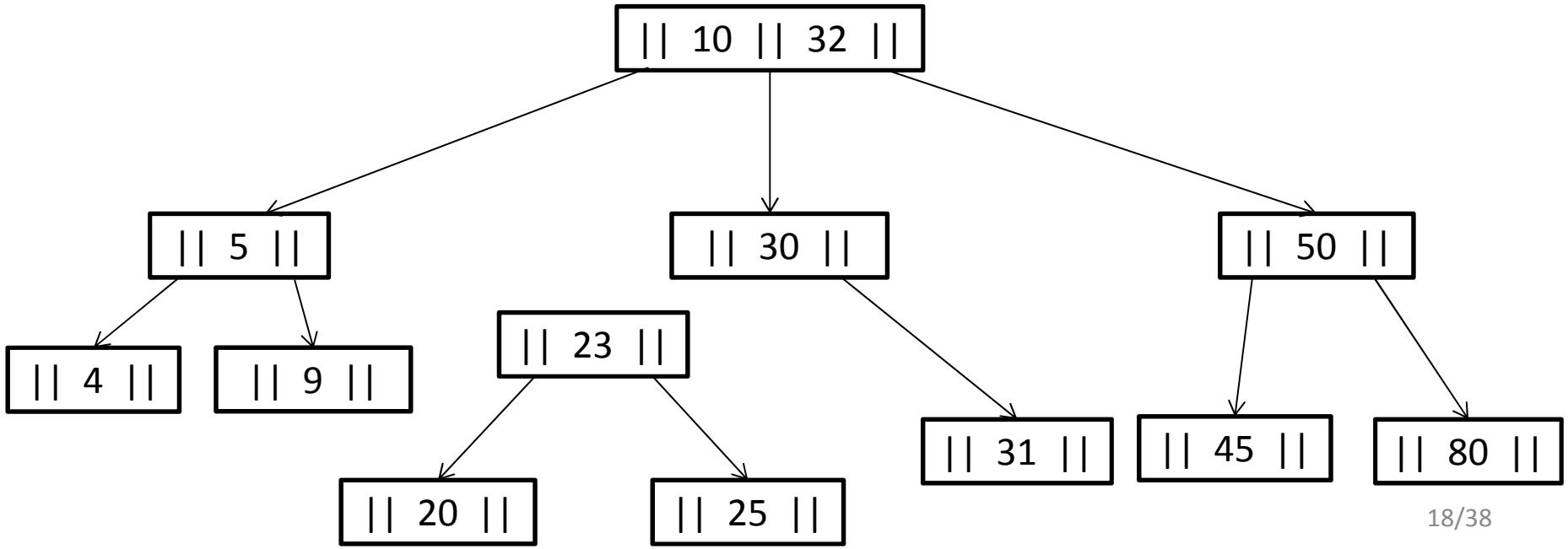
2.4 Einfügen von Daten(Beispiel)

Füge $k = 23$ ein.

2. Schritt:

2. Fall:

2. Splitte Knoten wie in *Abbildung*.



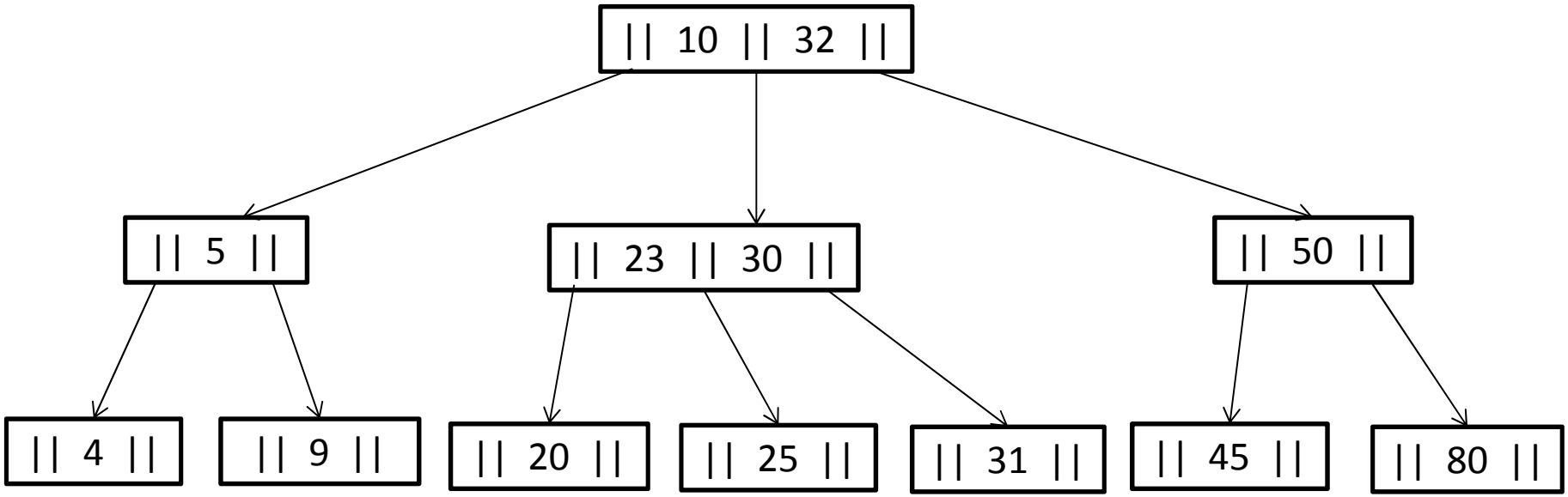
2.4 Einfügen von Daten(Beispiel)

Füge $k = 23$ ein.

2. Schritt:

2. Fall:

3. Füge k_{t+1} rekursiv in Vaterknoten ein.



2.5 Löschen von Daten

- Komplexe Operation.

- Unterscheidet 2 Fälle:

    1. Fall: Schlüssel wird aus einem Blatt gelöscht.

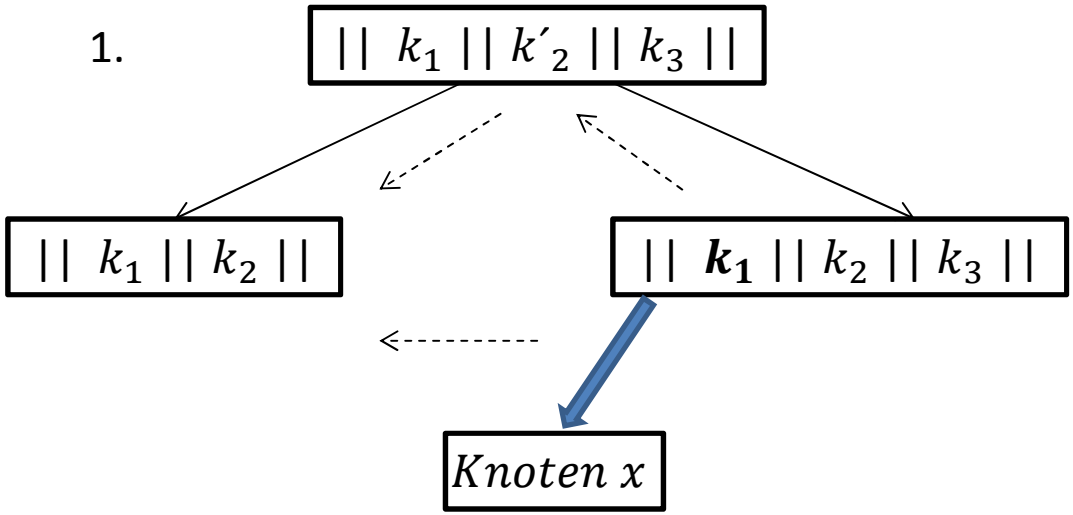
    2. Fall: Schlüssel wird aus einem inneren Knoten gelöscht.

- Nach dem Löschen muss die Ordnung des Baumes wiederhergestellt werden

2.5.1 Verschieben von Schlüsseln

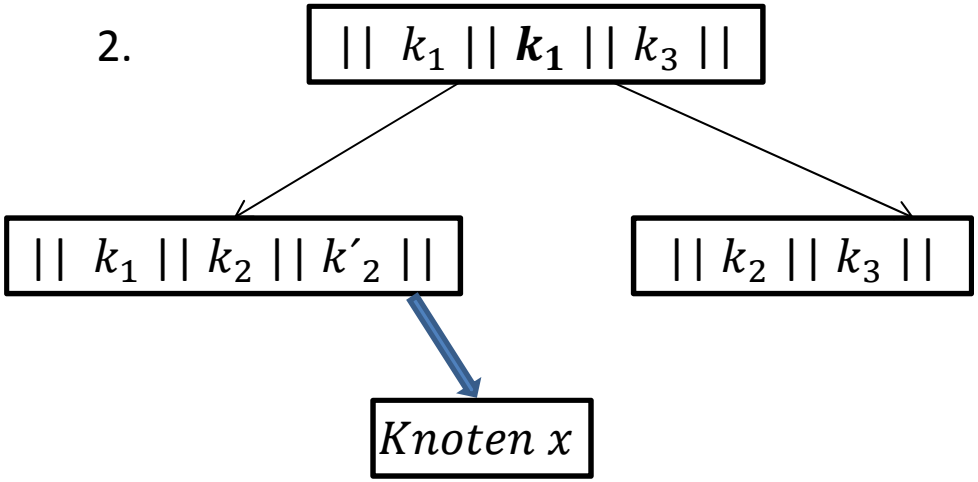
$t = 2$

1.



$\forall k \in x : k'_2 \leq k \leq \boldsymbol{k_1}$

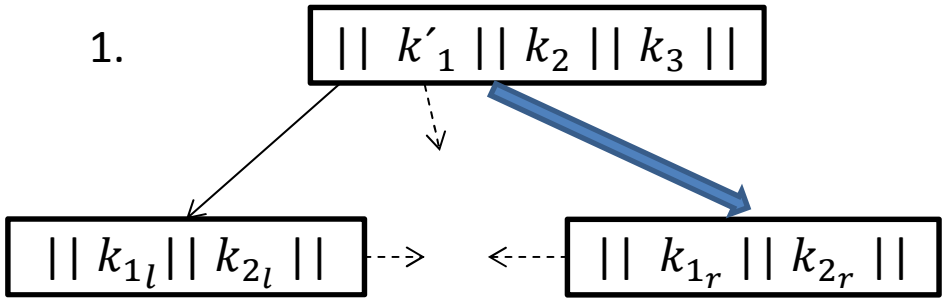
2.



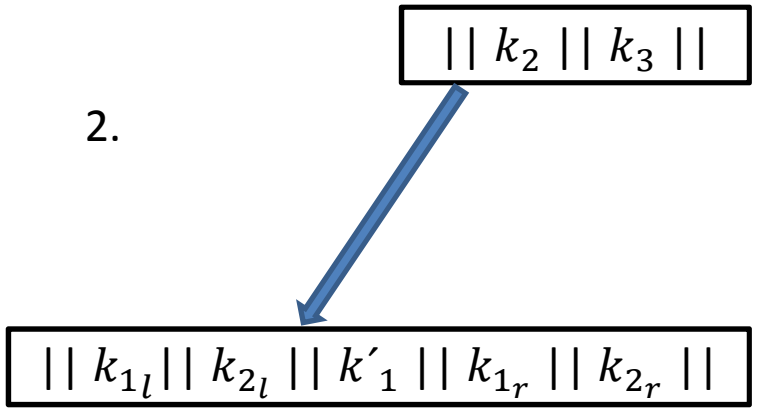
2.5.2 Verschmelzung von Knoten

$t = 2$

1.



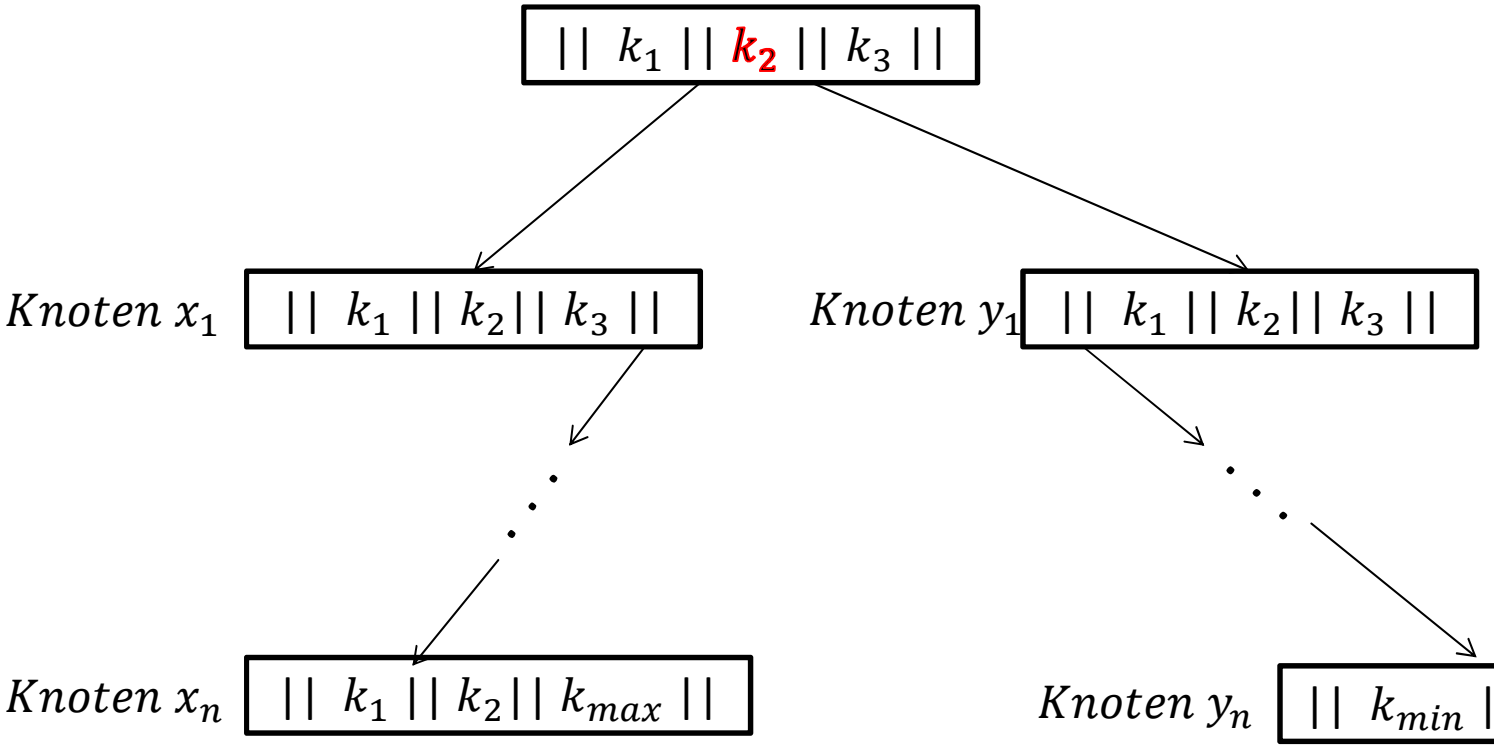
2.



Hier muss ein Schlüssel gelöscht werden.

2.5.3 Löschen aus inneren Knoten

k_2 soll gelöscht werden.



$\forall k_i \in x_i\ (1 \leq i \leq n) : k_i \leq k_{max} < k_2$

$k_{max} =$ *symmetrischer Vorgänger*

$\forall k_i \in y_i\ (1 \leq i \leq n) : k_i \geq k_{min} > k_2$

$k_{min} =$ *symmetrischer Nachfolger*

2.5.3 Löschen aus inneren Knoten

k_2 soll gelöscht werden.

1. Schritt: symmetrischer Vorgänger und Nachfolger werden gesucht*.

2. Schritt: Ersetzte k_2 durch Vorgänger bzw. Nachfolger und
lösche Vorgänger bzw. Nachfolger

*Entscheidung ob symmetrischer Vorgänger oder Nachfolger
Gewählt wird, hängt von s der jeweiligen Knoten x_n bzw. y_n ab.

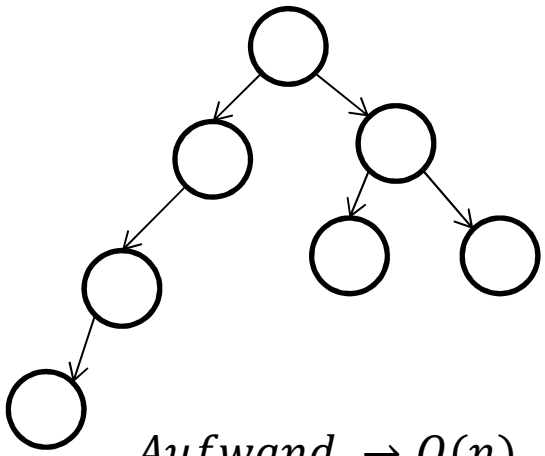
2.6 Aufwand

$Aufwand$: $O(\log_t n)$.

Nachteil:
Reorganisation nach Löschen oder Einfügen kommt oft vor.

Vorteil:
B-Bäume entarten nicht.

entarteter Baum:



$Aufwand \rightarrow O(n)$

2.7 Beispiel(B^+-Baum)

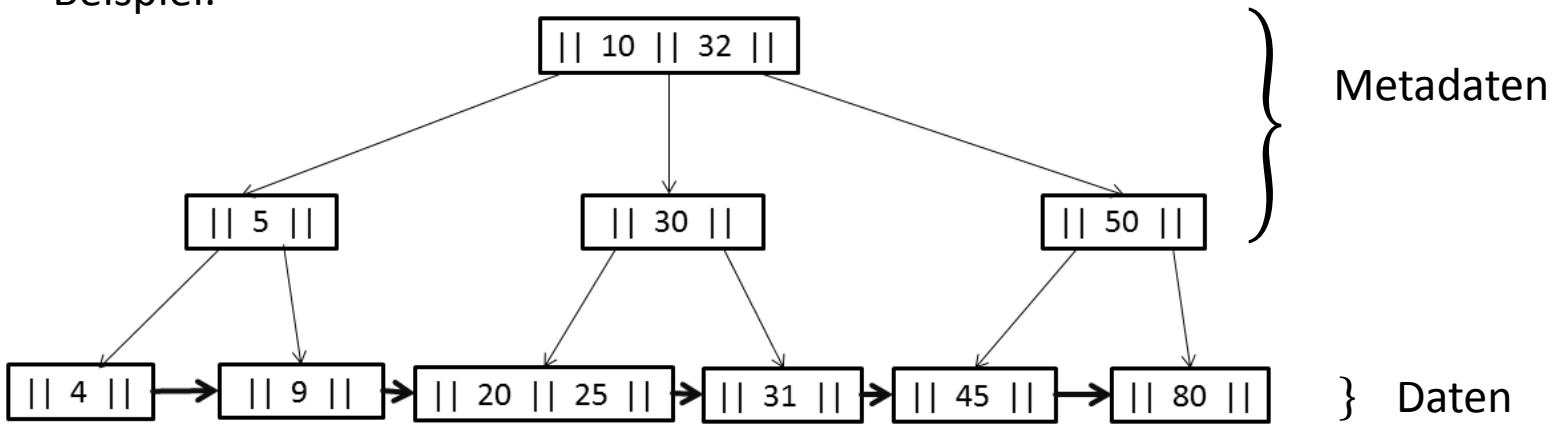
- Innere Knoten verweisen nicht auf Daten (Metadaten).

- Blätter verweisen auf Daten.

- Blätter von links nach rechts verkettet.

- In der Praxis häufig verwendet.

Beispiel:

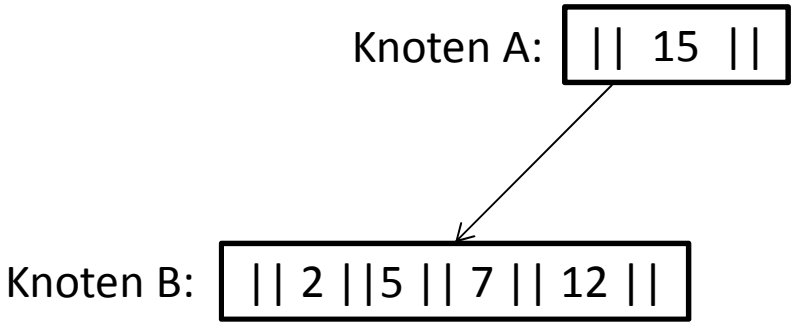


2.8 Parallele Operationen auf B-Bäume

Problem: 2 Prozesse greifen gleichzeitig auf B-Baum zu.

Prozess 1: sucht Datei(12)

Prozess 2: fügt Datei(3) ein.

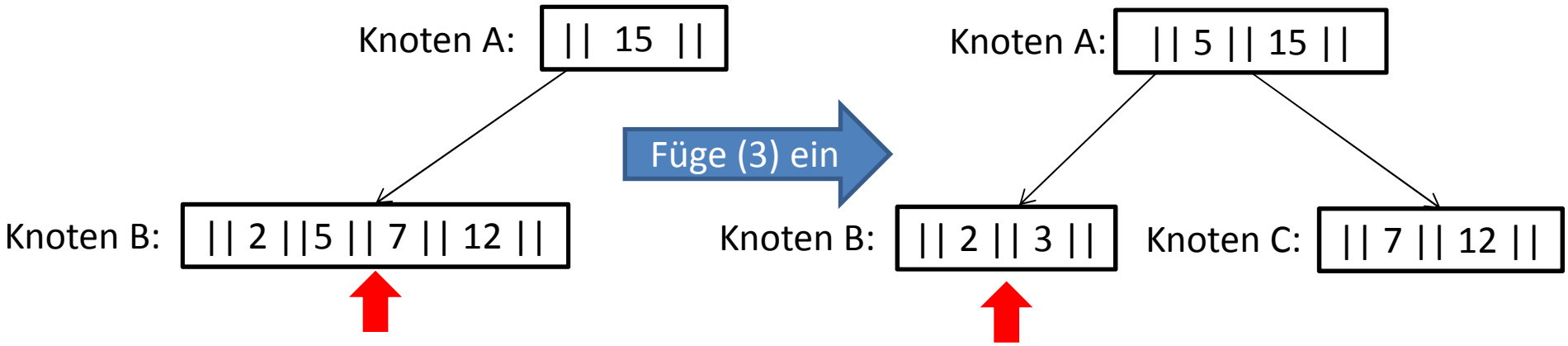


Modifiziertes Beispiel von http://wwwbayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf

2.8 Parallele Operationen auf B-Bäume

Zeit

Prozess 1	**Prozess 2**
Geht in den Unterbaum B	
	Spaltet B in B und C
Sucht in B nach Datei(12)	



Modifiziertes Beispiel von http://wwwbayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf

2.8 Parallele Operationen auf B-Bäume

Einfachste Lösung: Jeder Prozess markiert Wurzel mit Sperre für andere Prozesse.

- Leseprozess sperrt mit S.
  $\Rightarrow$ nur Lesezugriffe gestattet.

- Schreibprozess sperrt mit X.
  $\Rightarrow$ kein anderer Prozess gestattet.

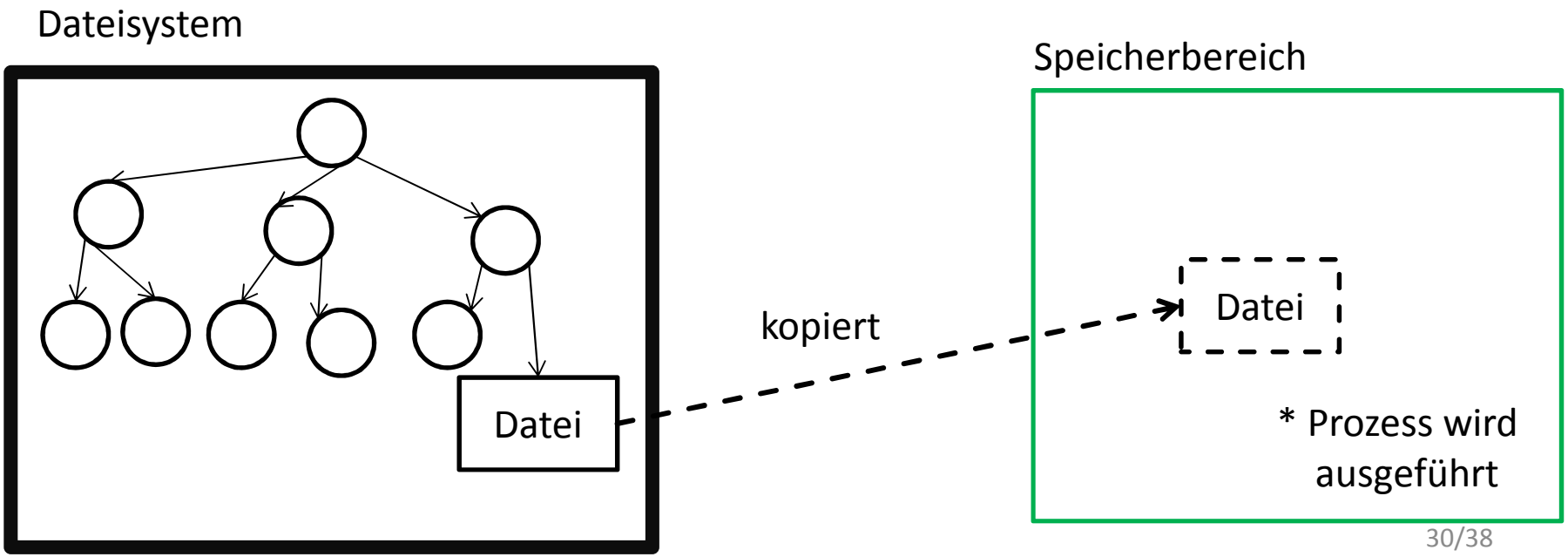
Neues Problem: Wenn sich immer ein Leseprozess im Baum befindet, kann nie geschrieben werden (Schreibprozess verhungert).

3. Copy-on-Write

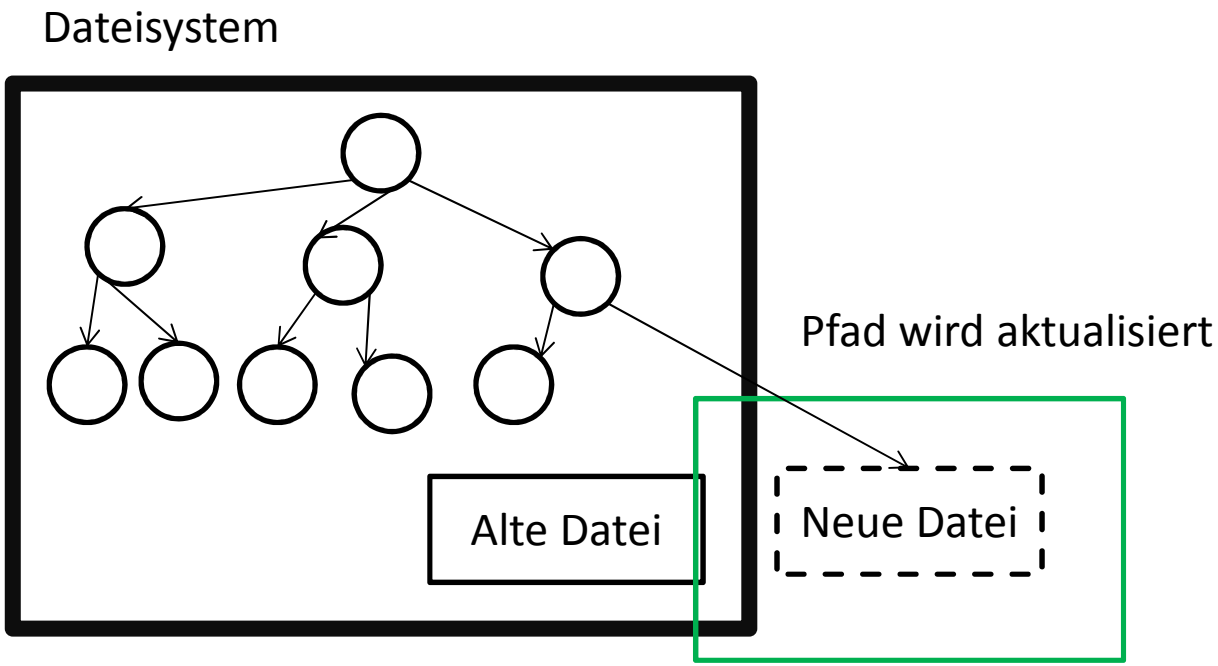
Zu deutsch: Kopieren-beim-Schreiben.

- Lesende Prozesse greifen alle auf das Originale Datensystem zu.

- Schreibende Prozesse schreiben auf Schattenkopie



3. Copy-on-Write



4. Shadowing/Cloning

- Erstellt exakte live Kopie der originalen Datenbank auf anderen Serverinstanz.

- Im Notfall ersetzen diese schnell die Produktionsdatenbank.

Vorteile	Nachteile
zusätzlicher Schutz zu Backups.	Schützt nicht vor Benutzerfehler.
Kann mit minimalem Zeitverlust aktiviert werden.	Shadowkopie kann Datenbank nicht zurücksetzen.
Läuft ohne Verwaltungsaufwand im Hintergrund.	Verursacht kleinen Leistungsabfall.
Kleiner als Logdateien.	Ersetzt kein Sicherheitssystem.

5. Stand der Wissenschaft (BTRFS)

- Copy-on-Write Dateisystem von Oracle Corporation.

- seit 2007 in der Entwicklung für Linux Betriebssysteme.

- soll ext3(bzw. ext4)-Dateisystem ersetzen.

- Speichert effizienter als ext3 / ext4.

- erweiterter Speicherbereich von $2^{64}\ byte = 16\ Exbibyte$.

- Defragmentierung und Datenkompression während des Betriebs.

- optimiert für Solid-State-Drives.

- findet selbstständig Fehler und behebt diese.

5. Stand der Wissenschaft (BTRFS)

- BTRFS speichert in 2 Baumstrukturen.
  Eine für Verzeichnis- und Dateinamen und eine für die Datenblöcke.

Performance Werte auf dem gleichem Testrechner:

	Ext3	**Ext4**	**Btrfs**
Anlegen von acht Dateien á 1 GByte			
Zeit	155,9s	145,1s	120,6s
Durchsatz beim Schreiben	55,4 MByte	59,3 MByte	68,5 MByte
Löschen von acht Dateien á 1 GByte			
Zeit	11,87 s	0,33s	0,63s
10 000 zufällige lese-Schreiboperationen in 8 GByte			
Operationen/ s	80,0	80,7	115,2

Werte von http://www.heise.de/open/artikel/Snapshots-Subvolumes-Performance-224650.html

5. Stand der Wissenschaft (BTRFS)

Am 9. Januar 2009 erstmals in Linuxkernel 2.6.29 aufgenommen.

stabile Version 1.0 war 2008 zur Veröffentlichung geplant,
wurde aber bisher nicht Veröffentlicht(Stand 29.04.2013)
(Quelle: http://www.golem.de/news/linux-kernel-3-9-viel-grafisches-und-mit-goldfischen-1304-98997.html (6.5.2013))

Fazit:

- hohes Potenzial
- noch nicht für produktiven Einsatz gedacht

6. Zusammenfassung

- Hashtabelle Aufwand $O(1)$.
  → Nachteil: artet schnell aus.

- B-Baum ist vollständig ausbalanciert.
  → Kann nicht ausarten

- $\mathrm{B}^{+} - \mathrm{Baum}$ werden in der Praxis oft verwendet.
  → Nur in den Blättern sind Daten.
    → von links nach rechts verkettet.

- Copy-on-Write ermöglicht mehrere parallele Zugriffe auf ein Dateisystem.

- Shadowing/Cloning live Kopien, welche zur Laufzeit erstellt werden.

- Btrfs soll ext3/ext4 ersetzen.
  → seit 2007 in der Entwicklungsphase.
  → innovatives System mit vielen Extras zur Optimierung.
  → sehr hohe Speicherverwaltung.

Quellen

B-Baum

- http://de.wikipedia.org/wiki/B-Baum (19.04.2013)
- http://en.wikipedia.org/wiki/B-tree (19.04.2013)
- http://martin-thoma.com/b-baume/ (19.04.2013)
- http://wwwbayer.in.tum.de/lehre/WS2001/HSEM-bayer/BTreesAusarbeitung.pdf (26.04.2013)

Shadowing/Cloning

- http://www.ibexpert.net/ibe_de/index.php?n=Doku.DatenbankShadow (19.04.2013)
- http://www.searchdatacenter.de/themenbereiche/storage/storage-management/articles/153256/index2.html (19.04.2013)
- http://msdn.microsoft.com/de-de/library/ms189852.aspx#Overview (26.04.2013)
- http://www.searchsecurity.de/themenbereiche/applikationssicherheit/datenbank-sicherheit/articles/119874/ (3.5.2013)
- http://winswitch.org/documentation/shadow.html (3.5.2013)

Quellen

Hashtabelle

- http://de.wikipedia.org/wiki/Hashtabelle (23.04.2013)
- SE 1 Skript

Copy-on-Write

- http://de.wikipedia.org/wiki/Copy-On-Write (3.5.2013)
- http://en.wikipedia.org/wiki/Copy-on-write (3.5.2013)
- http://www.it-business.de/glossar/Kopieren%20beim%20Schreiben/articles/194011/ (03.05.2013)

BTRFS

- http://de.wikipedia.org/wiki/Btrfs (6.5.2013)
- http://www.oracle.com/us/technologies/linux/btrfs-features-1405320.pdf (6.5.2013)
- https://btrfs.wiki.kernel.org/index.php/Main_Page (6.5.2013)
- http://www.heise.de/open/artikel/Snapshots-Subvolumes-Performance-224650.html (6.5.2013)