Grundlagen Datenbanken: Übung 09

Tanmay Deshpande

Gruppe 20 & 21

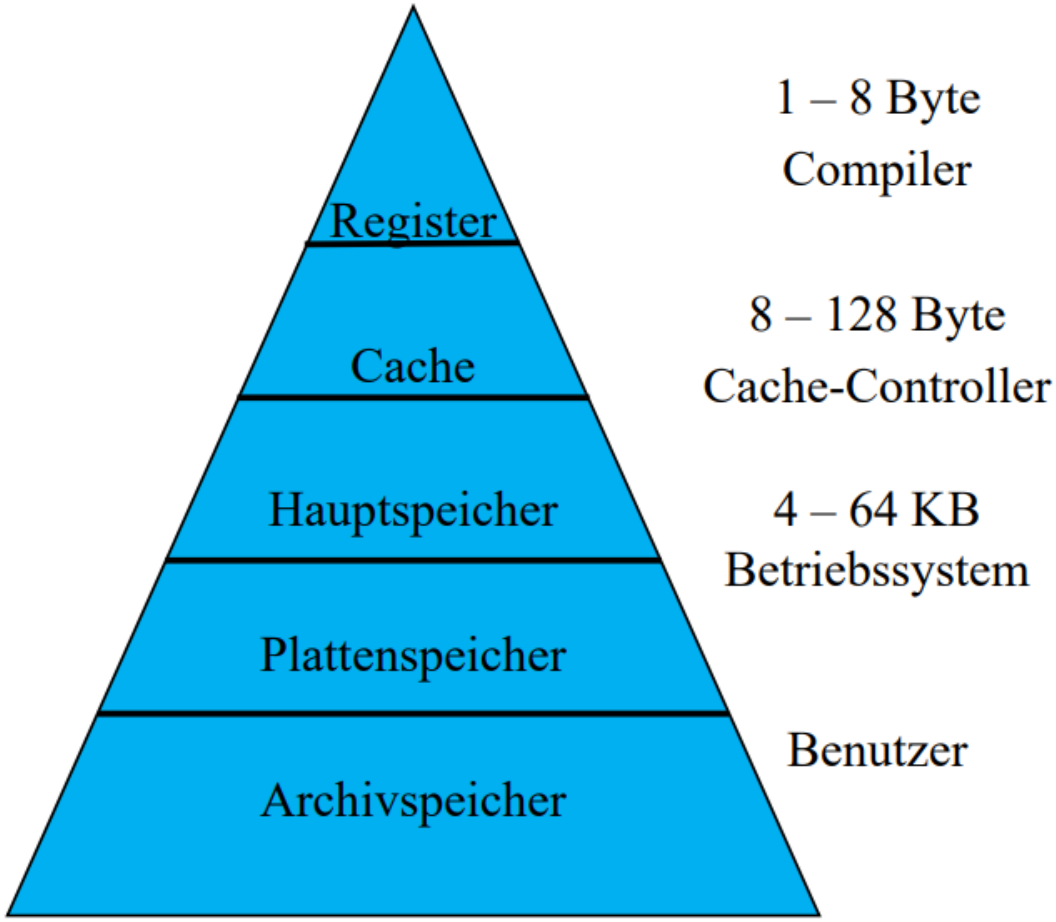
ge94vem@mytum.de

QR-Code für die Folien

Wiederholung

Woche 09

Speicherhierarchie



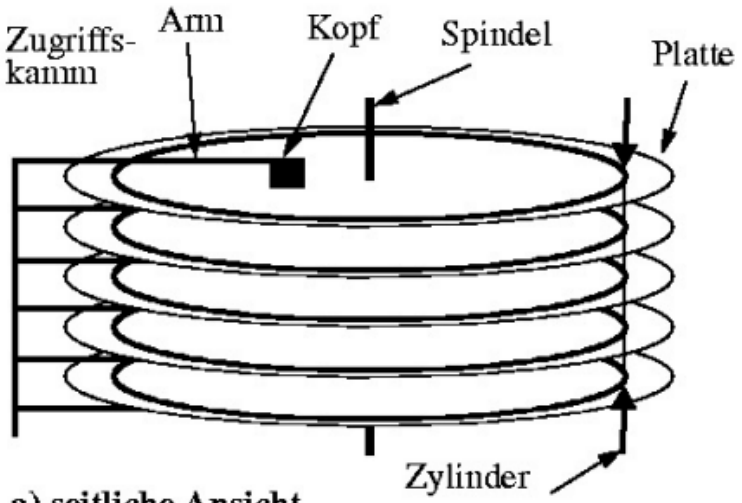
Magnetplattenspeicher

Aufbau

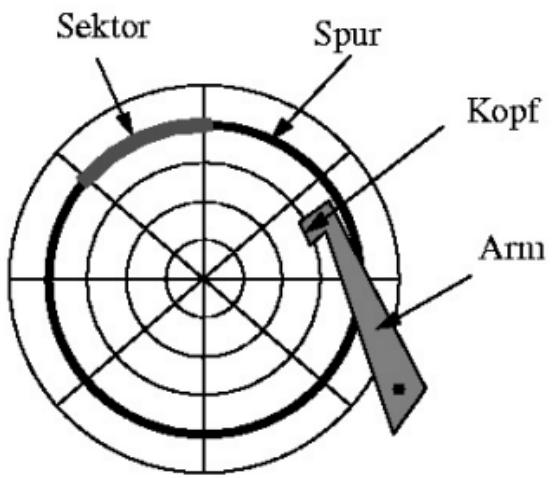
- mehrere gleichförmig rotierende Platten, für jede Plattenoberfläche ein Schreib-/Lesekopf
- jede Plattenoberfläche ist eingeteilt in Spuren
- die Spuren sind formatiert als Sektoren fester Größe (Slots)
- Sektoren (typischerweise 1 - 8 KB) sind die kleinste Schreib-/Leseeinheit auf einer Platte

Adressierung

- Zylindernummer, Spurnummer, Sektornummer
- jeder Sektor speichert selbstkorrigierende Fehlercodes; bei nicht behebbaren Fehlern erfolgt automatische Abbildung auf Ersatzsektoren



a) seitliche Ansicht



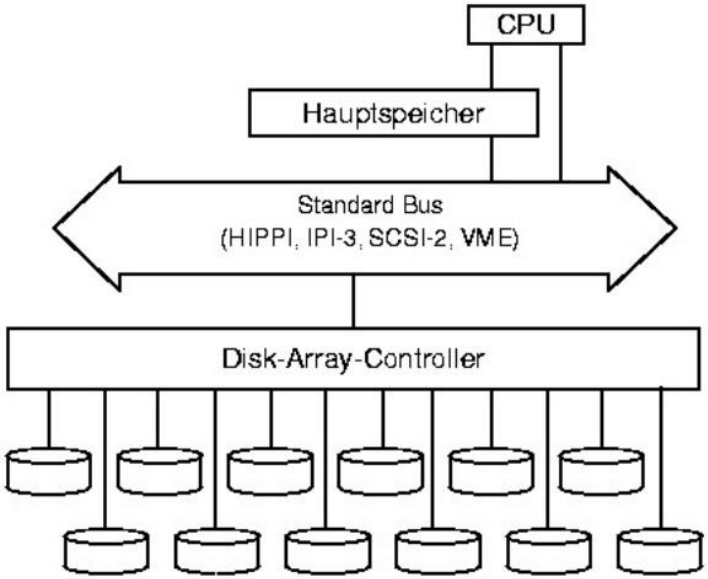
b) Draufsicht

Magnetplattenspeicher

- **Seek Time:** Zeit, die verbraucht wird, um den Schreib-/Lesekopf (Arm) auf dem richtigen Spur zu positionieren
- **Latenzzeit:** Zeit, die die Festplatte für Rotation braucht, damit der Schreib-/Lesekopf auf dem richtigen Sektor zugreifen kann.
  Beträgt im Durchschnitt ½ * Plattenumdrehung
- **Datentransfer von der Festplatte zu Hauptspeicher**
- **Random I/O:**
  Daten nicht notwendigerweise benachbart
  Jedes Mal wird den Arm neu positioniert und die Festplatte umgedreht
  Gesamtzeit = (Seek Time + Latenzzeit) * #Zugriffe + Transferzeit
- **Chained I/O:**
  Daten liegen benachbart auf der Festplatte
  Arm einmal positionieren, dann „von der Platte kratzen“
  Gesamtzeit = Seek Time + Latenzzeit + Transferzeit

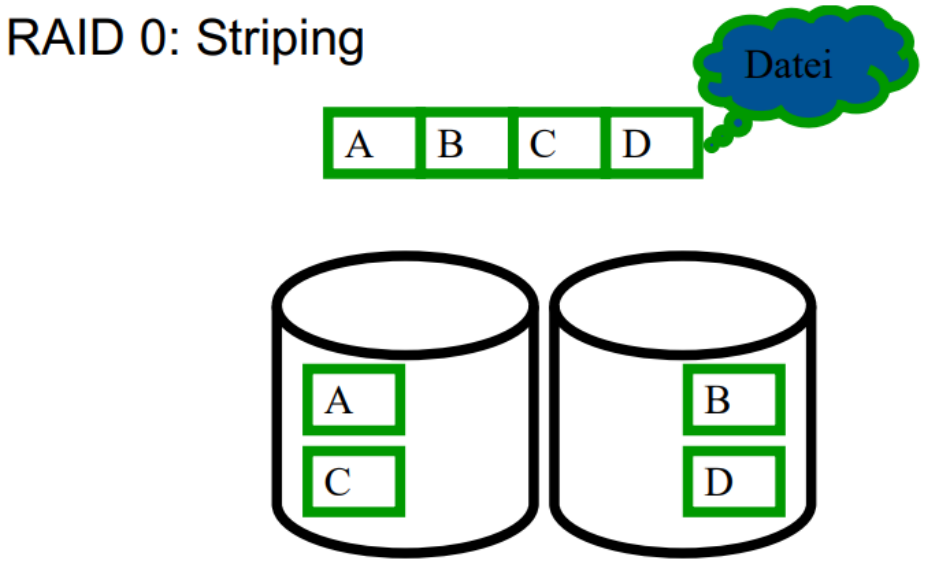
RAID

- RAID = Redundant Array of Independant Disks oder Redundant Array of Inexpensive Disks
- Verknüpft mehrere Festplatten zu einer logischen Einheit
- Ziel: Redundanz um Ausfallssicherheit zu erhöhen, Performanz verbessern
- Verschiedene RAID-Levels, die diese Ziele in unterschiedlichem Maße erreichen



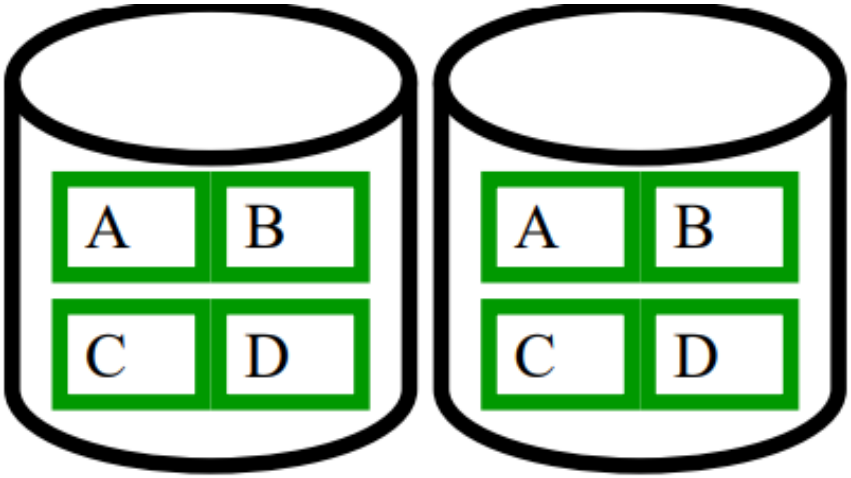
RAID 0: Striping auf Block-Ebene

- Lastbalancierung wenn alle Blöcke gleich häufig genutzt
- Doppelte Bandbreite beim sequentiellen Lesen der Datei bestehend aus ABCD
- Datenverlust wird wahrscheinlicher, je mehr Platten man verwendet



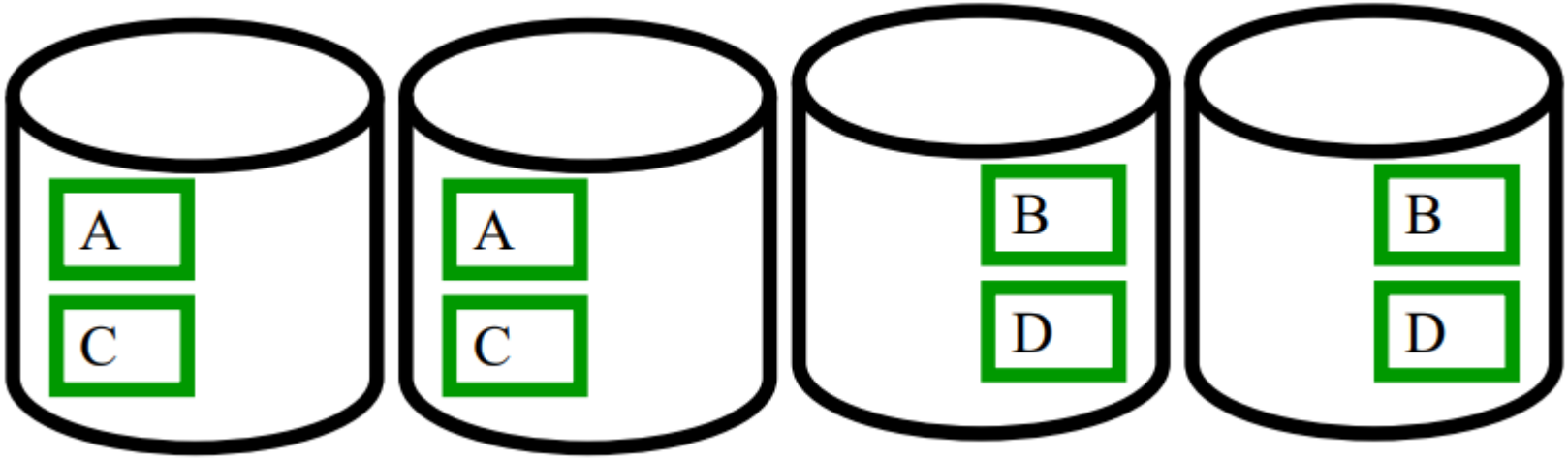
RAID 1: Spiegelung

- Erhöhte Datensicherheit wegen Redundanz
- Doppelter Speicherbedarf
- Lastbalancierung beim Lesen
- Beim Schreiben müssen beide Platten geändert werden. Das kann allerdings parallel geschehen und dauert nicht doppelt so lange wie das Schreiben von einem Block



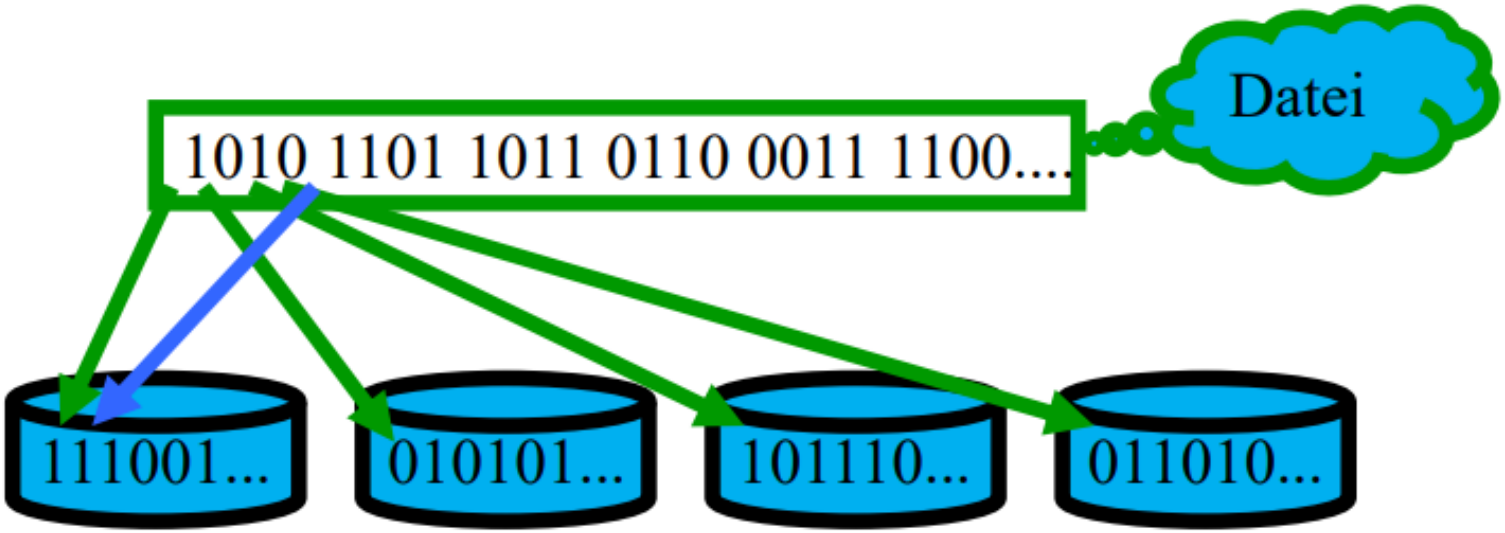
RAID 0 + 1: Striping + Spiegelung

- Kombiniert RAID 0 und RAID 1. Auch RAID 10 genannt
- Doppelter Speicherbedarf und Redundanz
- Lastbalancierung beim Lesen
- Beim Schreiben müssen beide Platten geändert werden. Das kann allerdings parallel geschehen und dauert nicht doppelt so lange wie das Schreiben von einem Block



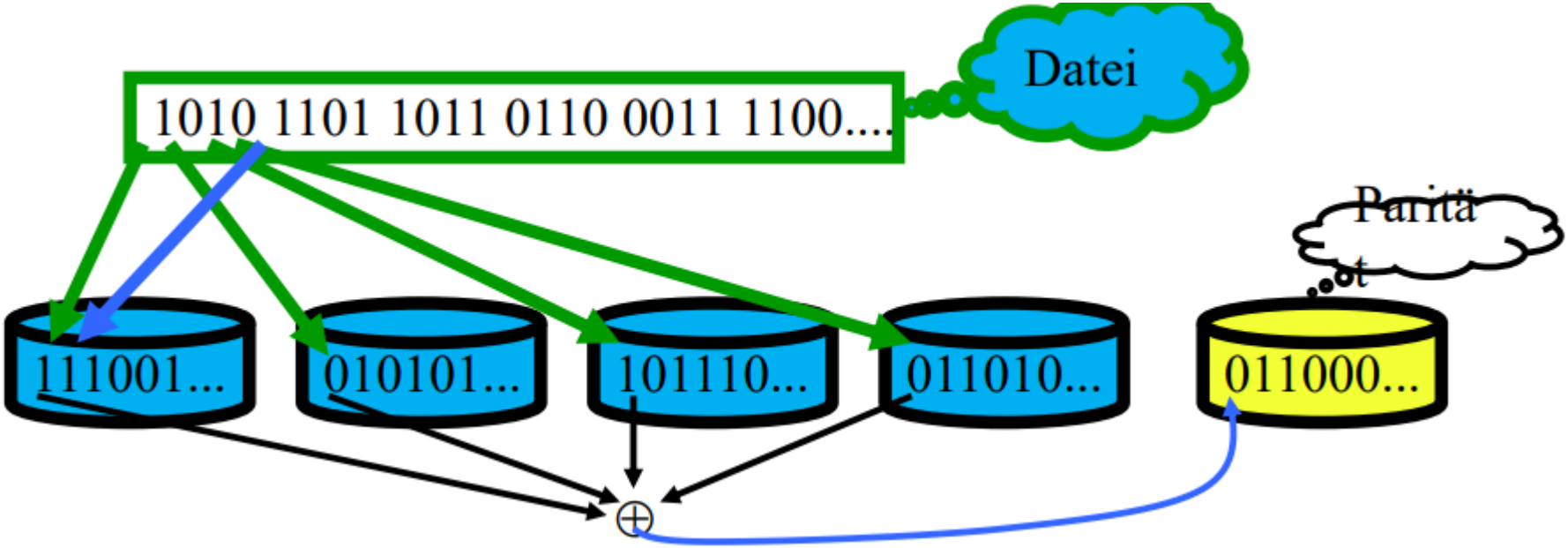
RAID 2: Striping auf Bit-Ebene

- Anstatt ganze Blöcke erfolgt Striping auf Bit-Ebene
- Theoretisch können Fehlererkennungscodes und Korrekturcodes auf jede Platte gespeichert werden
- In der Praxis nicht eingesetzt, weil Platten schon Fehlererkennungscodes verwalten



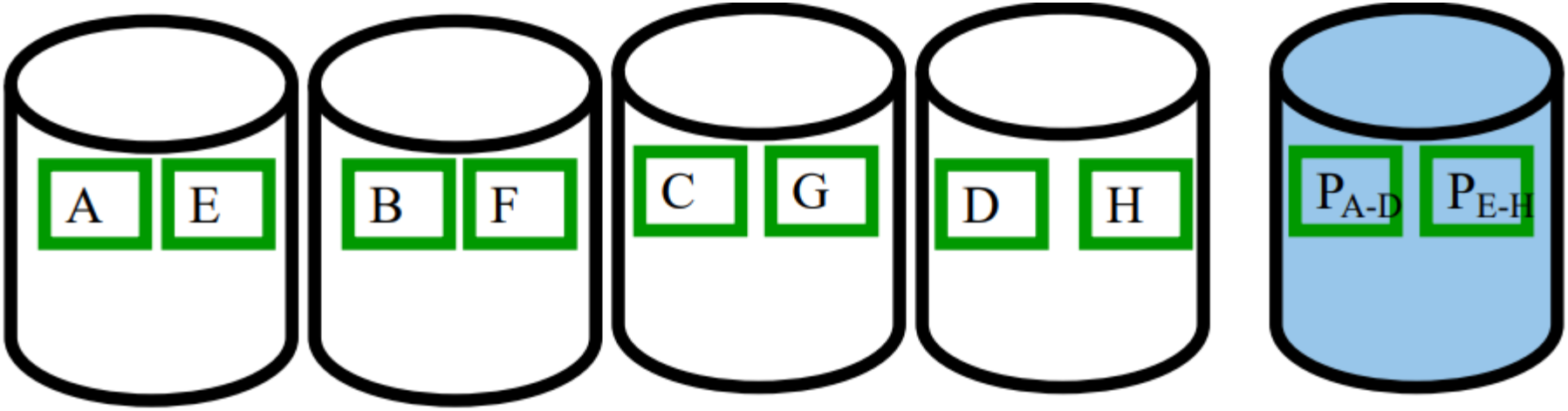
RAID 3: Striping auf Bit-Ebene + Paritätsplatte

- Bitweises Striping + zusätzliche Paritätsplatte
- Lesen eines Blocks erfordert Zugriff auf allen Platten
- Parität = Bitweises XOR. Kann der Ausfall von einer Platte kompensieren



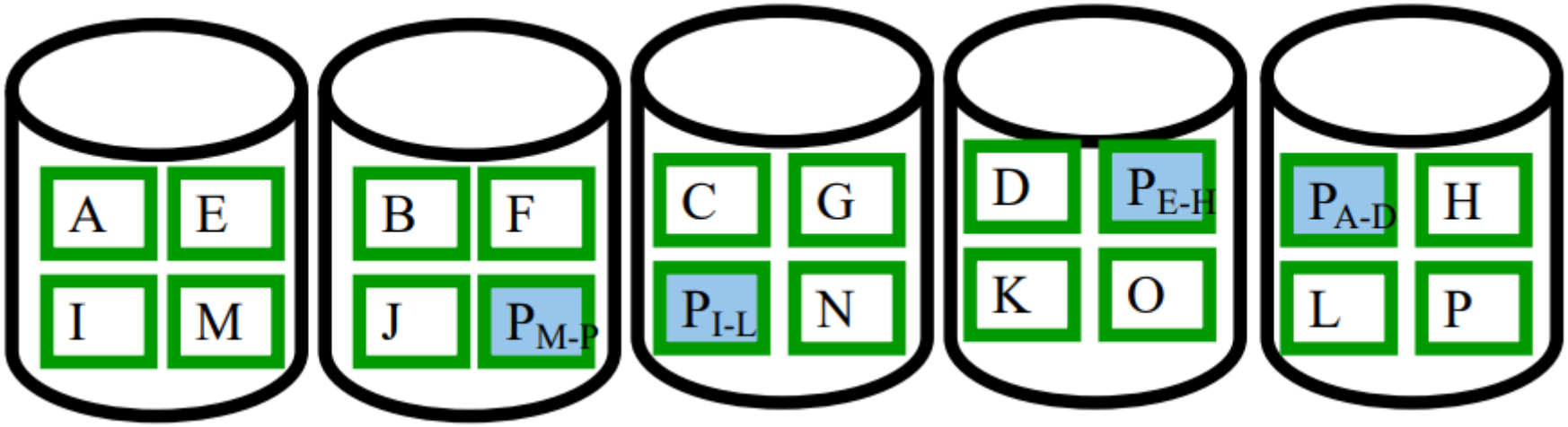
RAID 4: Striping auf Block-Ebene + Paritätsplatte

- Bessere Lastbalancierung als RAID 3
- Paritätsplatte bildet Flaschenhals – muss bei jeder Schreiboperation aktualisiert werden
- Modifikation der Paritätsplatte bei Änderung in Block A: $P'_{A\text{-}D} := P_{A\text{-}D} \oplus A \oplus A'$



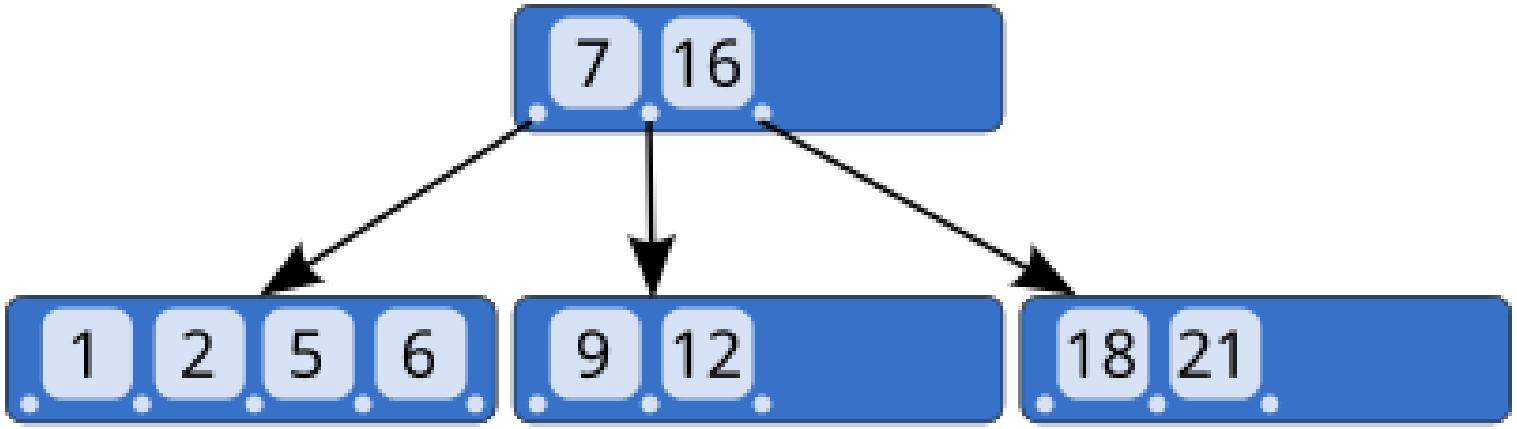
RAID 5: Striping auf Block-Ebene + verteilte Parität

- Bessere Lastbalancierung als RAID 4
- Paritätsplatte bildet keinen Flaschenhals mehr
- In der Praxis häufig eingesetzt
- Guter Ausgleich zwishen Platzbedarf und Leistungsfähigkeit



B-Bäume

- Effiziente Datenstruktur zur Speicherung von Daten
- (a, b)-Baum mit 2a = b
- Ein B-Baum mit Grad k bedeutet, a=k und b=2k
- Baum ist sortiert, links kleiner, rechts großer
- Pfad zur Wurzel von jedem Blatt aus ist gleich lang (balanciert)
- Jeder Knoten mindestens halb voll (außer Wurzel)



B-Bäume: Einfügen

1. Suche nach dem Schlüssel. Die Suche endet an der Einfügestelle

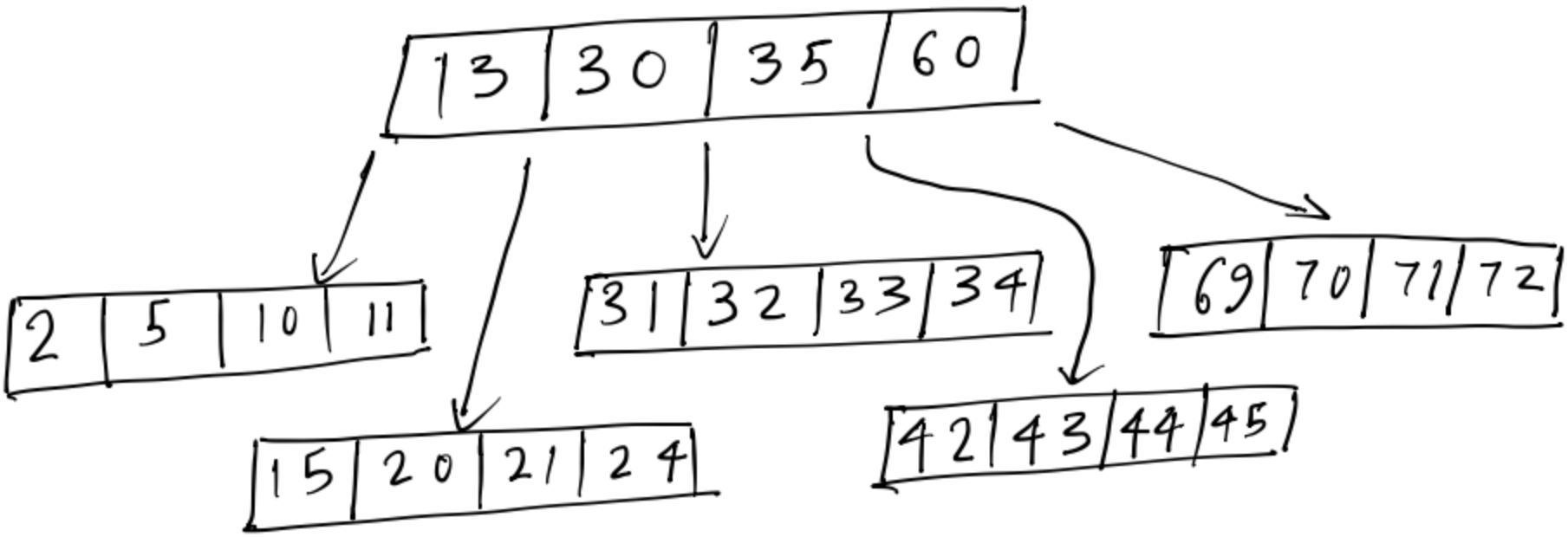
2. Füge den Schlüssel dort ein

3. Falls der Knoten jetzt überfüllt sein soll, teile ihn:
- Neuer Knoten anlegen und alle Einträge rechts vom mittleren Eintrag dort übernehmen
- Mittlerer Eintrag im Vaterknoten hinzufügen
- Im Vaterknoten, Verweis rechts vom mittleren Eintrag zu dem neuen Knoten verbinden

4. Falls Vaterknoten jetzt überfüllt sein soll:
- Falls es Wurzel ist, lege eine neue Wurzel an
- Sonst 3. wiederholen

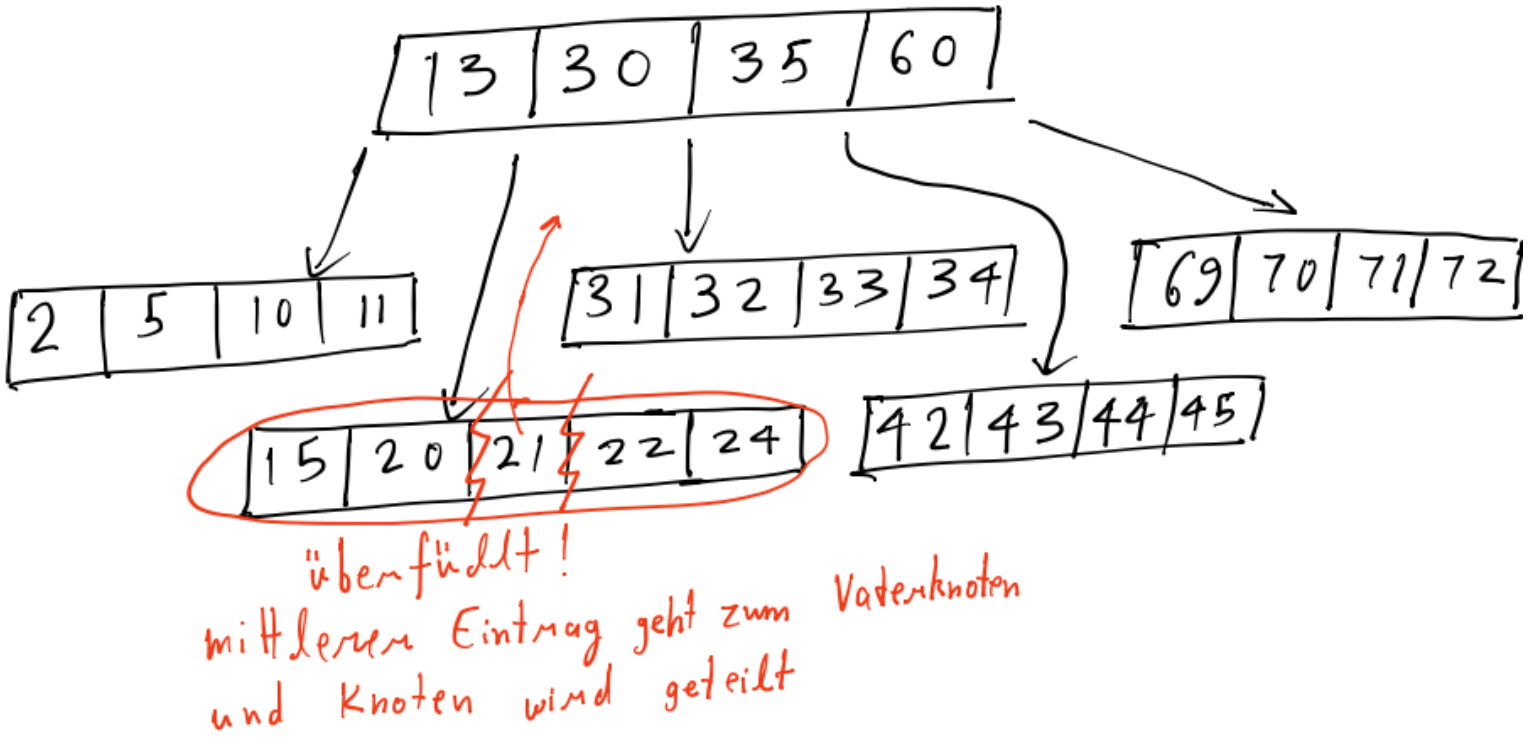
B-Bäume: Einfügen (Beispiel)

Wir wollen 22 einfügen



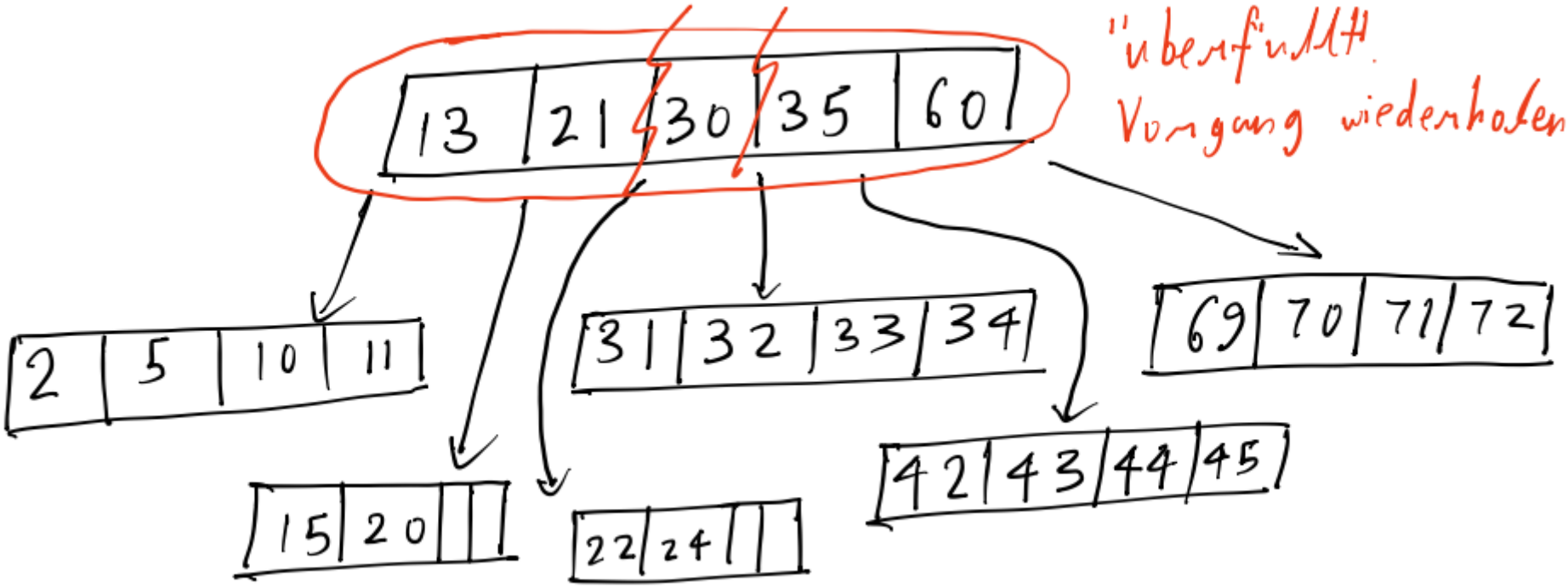
B-Bäume: Einfügen (Beispiel)

Wir wollen 22 einfügen



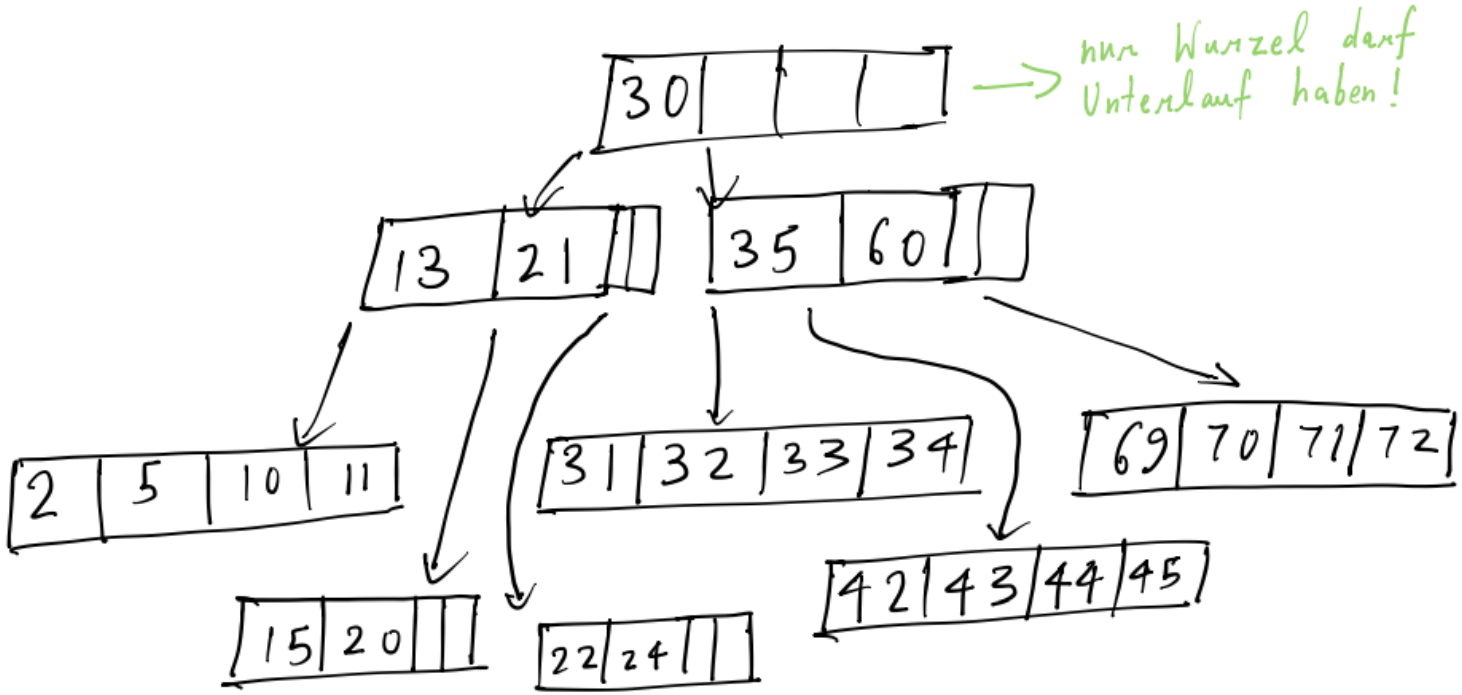
B-Bäume: Einfügen (Beispiel)

Wir wollen 22 einfügen



B-Bäume: Einfügen (Beispiel)

Wir wollen 22 einfügen



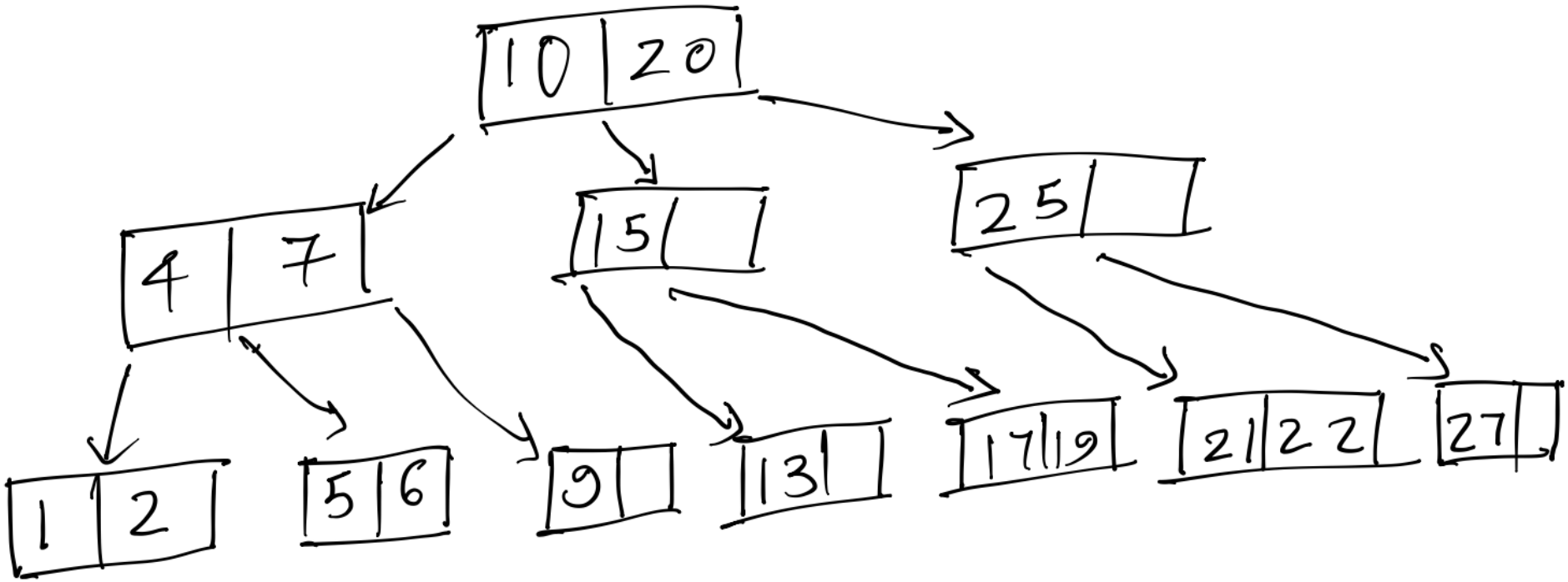
B-Bäume: Entfernen

- Zu entfernendes Element ist im Blatt:
  1. Entfernen
  2. Falls jetzt zu wenig im Knoten drin ist, von rechts oder links ein Element „klauen“, falls geht und Trenner beachten. Sonst mit rechts oder links und dem Trenner von Vaterknoten verschmelzen.
  3. Wiederhole falls nötig

- Zu entfernendes Element ist nicht im Blatt:
  1. Suche nach dem Nachfolger
  2. Tausche die beiden
  3. Entfernen
  4. Falls jetzt zu wenig im Knoten drin ist, von rechts oder links ein Element „klauen“, falls geht und Trenner beachten. Sonst mit rechts oder links und dem Trenner von Vaterknoten verschmelzen.
  5. Wiederhole falls nötig

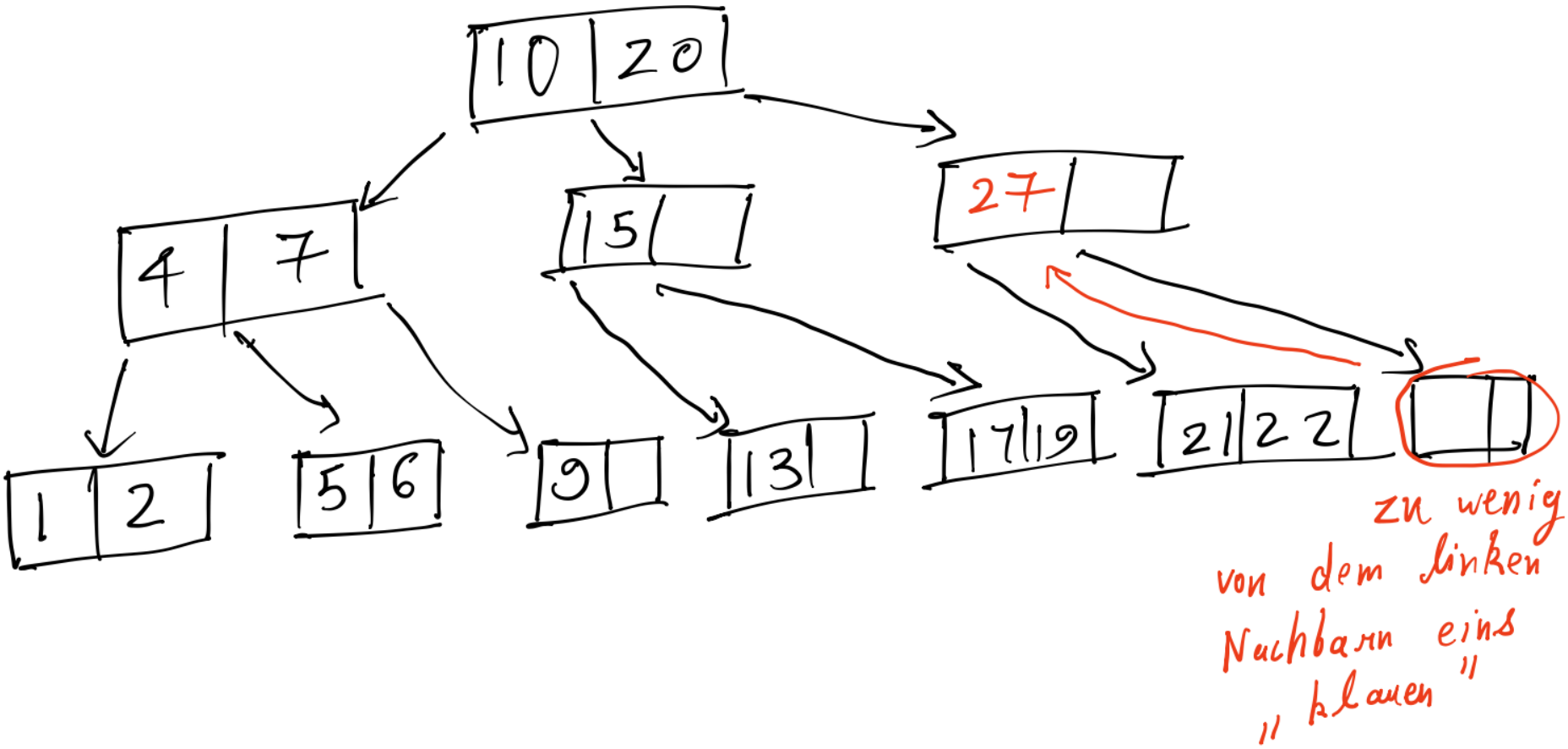
B-Bäume: Entfernen (Beispiel)

Wir wollen die 25 entfernen



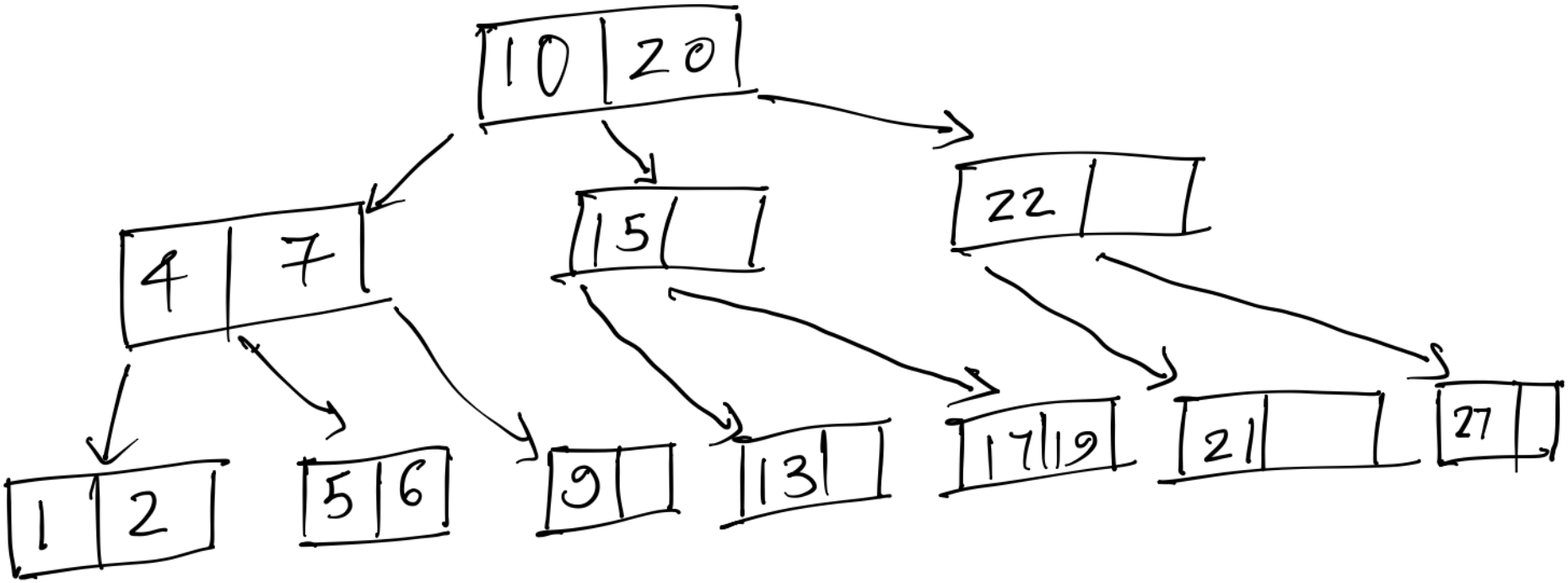
B-Bäume: Entfernen (Beispiel)

Wir wollen die 25 entfernen



B-Bäume: Entfernen (Beispiel)

Wir wollen die 25 entfernen



Aufgaben

Woche 09

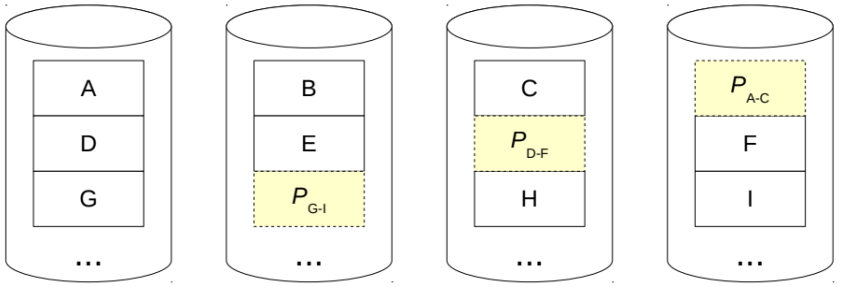
Aufgabe 01 a-d

a) Um welches RAID-Level handelt es sich?

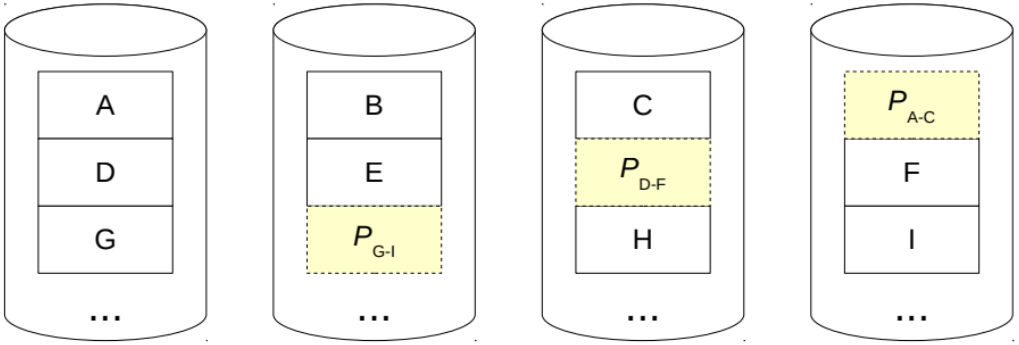
b) Wieviele Festplatten können ausfallen, ohne dass mit Datenverlust zu rechnen ist? Geben Sie eine allgemeine Lösung für einen Verbund bestehend aus n Festplatten an.

c) Kann die Ausfallsicherheit erhöht werden? Begründen Sie!

d) Welchen weiteren Vorteil bietet das gezeigte RAID-System neben der Ausfallsicherheit?



Lösungsvorschlag 01 a-d



- a) Um welches RAID-Level handelt es sich?
- RAID 5

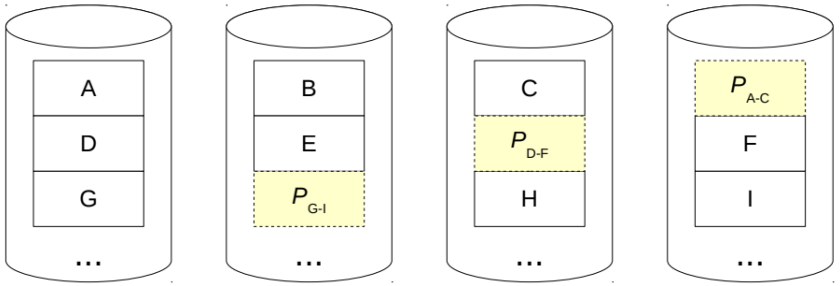
- b) Wieviele Festplatten können ausfallen, ohne dass mit Datenverlust zu rechnen ist? Geben Sie eine allgemeine Lösung für einen Verbund bestehend aus n Festplatten an.
- 1, unabhängig von n

- c) Kann die Ausfallsicherheit erhöht werden? Begründen Sie!
- Ja, durch RAID 6 (kann Ausfall zweier Platten kompensieren) oder RAID 15 (RAID 5 zusätzlich gespiegelt)

- d) Welchen weiteren Vorteil bietet das gezeigte RAID-System neben der Ausfallsicherheit?
- Höher Datendurchsatz

Aufgabe 01 e

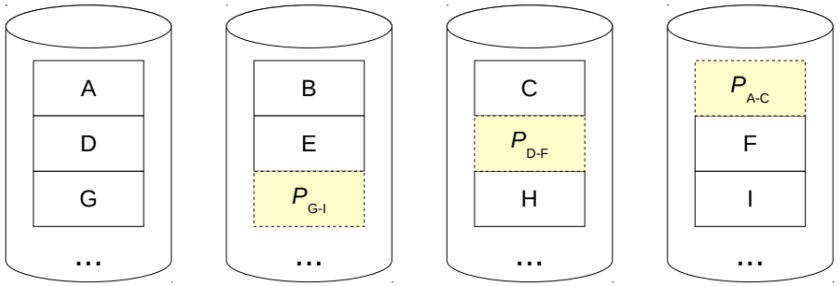
e) Nach einem Festplattendefekt enthalten die Datenblöcke die folgenden Binärdaten. Rekonstruieren Sie die Datenblöcke der $Disk_2$ mithilfe der XOR-Verknüpfung.



$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
$A = 1111$	$B = 1001$	$C = \_\_\_\_$	$P_{A-C} = 1110$
$D = 0101$	$E = 1100$	$P_{D-F} = \_\_\_\_$	$F = 1100$
$G = 0011$	$P_{G-I} = 1110$	$H = \_\_\_\_$	$I = 0011$

Lösungsvorschlag 01 e

e) Nach einem Festplattendefekt enthalten die Datenblöcke die folgenden Binärdaten. Rekonstruieren Sie die Datenblöcke der $Disk_2$ mithilfe der XOR-Verknüpfung.



$Disk_0$	$Disk_1$	$Disk_2$	$Disk_3$
$A = 1111$	$B = 1001$	$C = \mathbf{1000}$	$P_{A-C} = 1110$
$D = 0101$	$E = 1100$	$P_{D-F} = \mathbf{0101}$	$F = 1100$
$G = 0011$	$P_{G-I} = 1110$	$H = \mathbf{1110}$	$I = 0011$

Aufgabe 02

Gegeben sei ein Array von 1.000.000.000 8-Byte-Integer-Werten und ein Programm, das alle Werte aufsummiert.

Das Programm wird auf einem System mit 16 GB Hauptspeicher und einer herkömmlichen Magnetfestplatte (Größe 1 TB), auf der alle Werte sequentiell gespeichert sind, ausgeführt. Ein Random Access auf die Festplatte dauert 10 ms, beim sequentiellen Lesen hat sie einen Durchsatz von 160 MB/s. Das Summieren zweier Werte im Hauptspeicher dauert 1 ns.

(1 MB = 10^6 B und 1 TB = 10^{12} B)

a) Gehen Sie davon aus, dass alle Werte bereits im Hauptspeicher liegen. Wie lange läuft das Programm?

b) Nun liegen alle Werte ausschließlich auf der Festplatte. Wie lange läuft das Programm jetzt?

c) Auf der Festplatte liegt jetzt zusätzlich nach jedem 100.000. Wert die Summe der 100.000 davorliegenden Werte. Wie lange läuft das Programm, wenn es nur diese Summen aufsummiert?

Lösungsvorschlag 02 a) und b)

a) - Jede Zahl wird auf eine laufende Gesamtsumme addiert.
- Insgesamt müssen also 10^9 Additionen ausgeführt werden.
- Bei einer Zeit von 1 ns pro Addition ergibt dies eine Gesamtlaufzeit von $10^9 \cdot 10^{-9}$ s = 1s

b) Da die Werte sequentiell auf der Festplatte liegen, muss der Lesekopf nicht jeden Wert einzeln ansteuern sondern nur einmal zum ersten Wert finden und dann die Daten sequentiell auslesen. Hier wird die Lesezeit also vom maximalen Durchsatz der Platte dominiert. Insgesamt ergibt sich also:

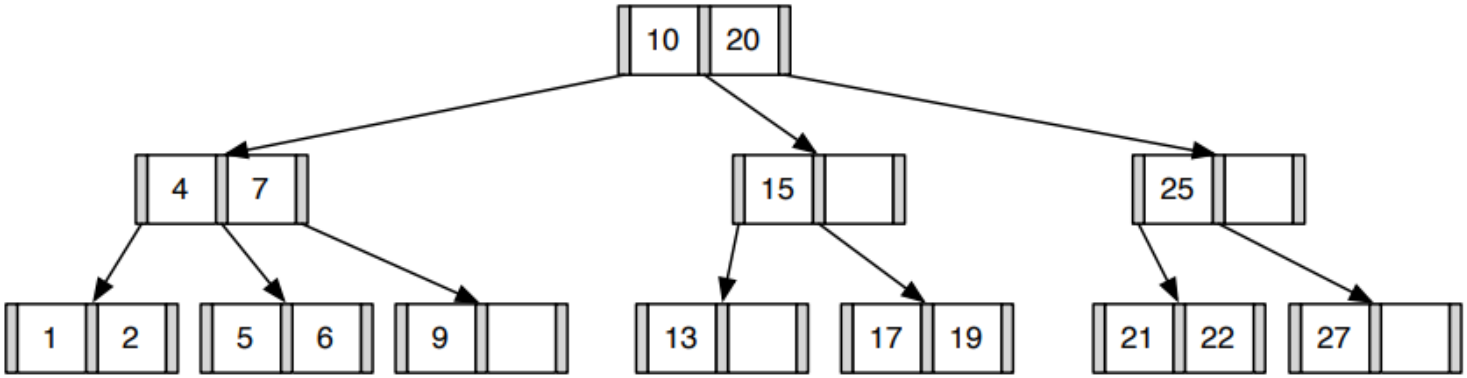
$$
\begin{aligned}
t_{total} &= t_{seek} + t_{read} \\
&= 10\ \text{ms} + \frac{10^9 \cdot 8\ \text{B}}{160 \cdot 10^6\ \frac{\text{B}}{s}} \\
&= 10\ \text{ms} + \frac{8 \cdot 10^9\ \text{B}}{16 \cdot 10^7\ \frac{\text{B}}{s}} \\
&= 10\ \text{ms} + 50\ \text{s} \\
&\approx 50\ \text{s}
\end{aligned}
$$

Dazu kommt noch die in Teilaufgabe a) berechnete Additionszeit selbst. Die Gesamtlaufzeit beträgt also 51 Sekunden.

Lösungsvorschlag 02 c)

- Hier kann sich das Programm die Einzeladditionen sparen und muss lediglich die $10^9/10^5$ = 10000 Zwischenwerte addieren.
- Diese liegen nun allerdings nicht mehr sequentiell hintereinander, sondern 100000·8 B = 800 KB auseinander. Das ist wesentlich größer als ein typischer Festplattensektor. Jeder Lesezugriff auf einen solchen Zwischenwert ist also ein Random Access.
- Wir erhalten so eine aufsummierte Zugriffszeit von 10000 · 10 ms = 100 s. Die Additionszeit mit 10000 ns = 10 µs und die Übertragungszeit mit (8 · 10000 B)/(160 · 10^6 B/s) = 0.5 ms ist vernachlässigbar.
- Insgesamt läuft das Programm also ungefähr 100 s. Trotz der „Optimierung“ hat diese Variante also eine längere(!) Laufzeit, als der „naive“ Ansatz aus Teilaufgabe b).

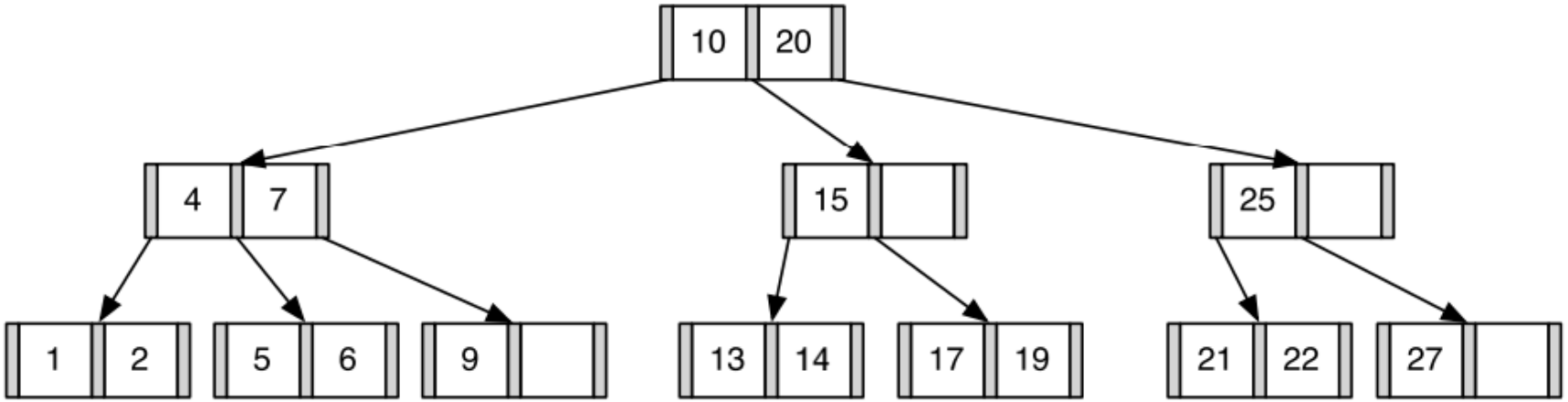
Aufgabe 03



Füge 14, 18 und anschließend 3 in den abgebildeten B-Baum ein.

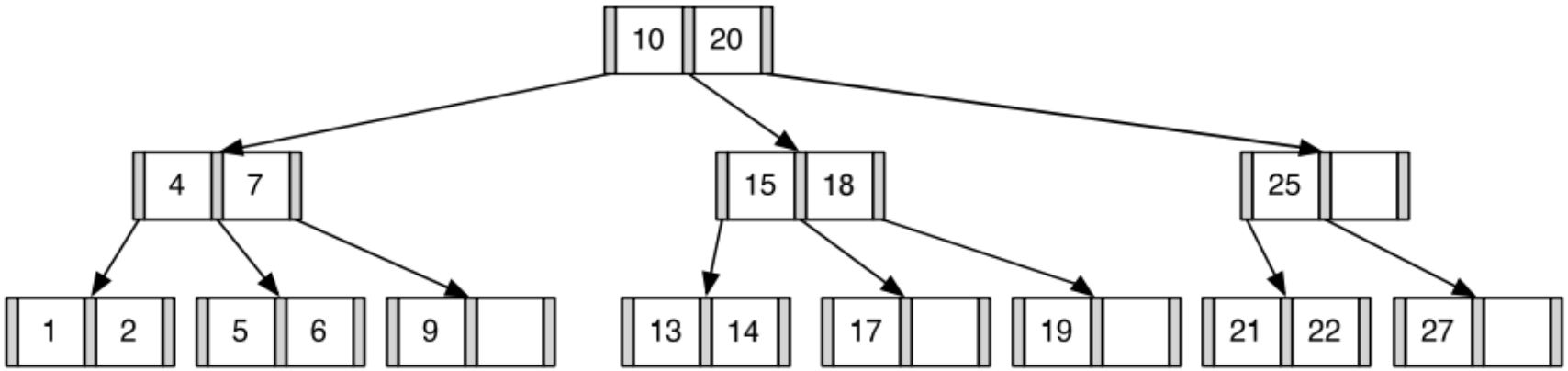
Lösungsvorschlag 03

Nach dem Einfügen von 14:



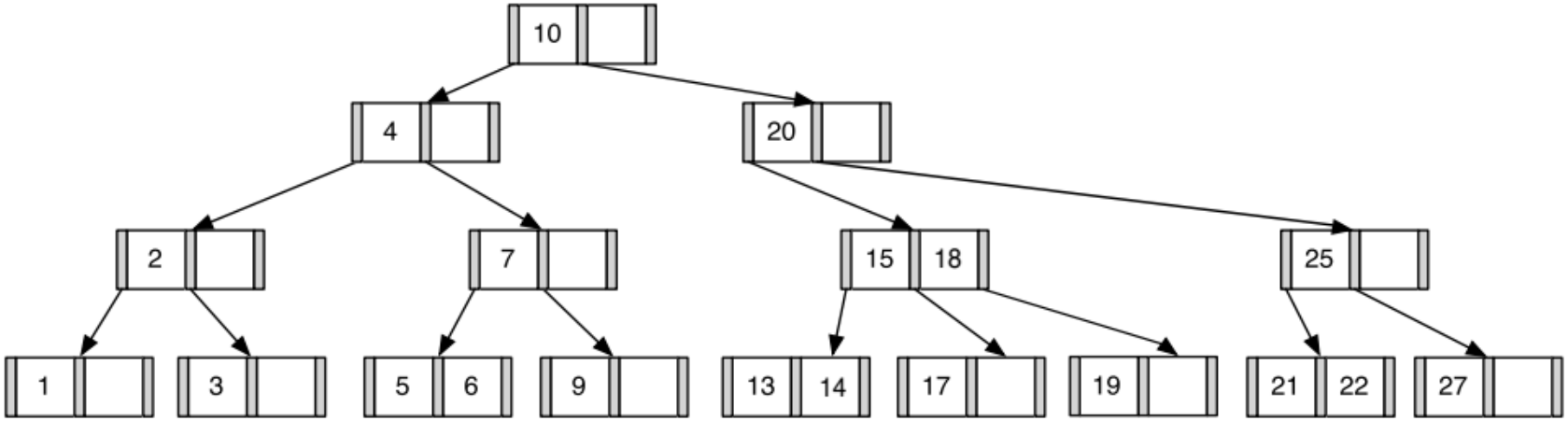
Lösungsvorschlag 03

Nach dem Einfügen von 18:



Lösungsvorschlag 03

Nach dem Einfügen von 3:



Aufgabe 04

Bestimmen Sie k für einen B-Baum, der die folgenden Informationen aller Menschen auf der Erde (ca. 10 Milliarden) enthalten soll: Namen, Land, Stadt, PLZ, Straße und Hausnummer (insgesamt ca. 100 Byte).

Dabei ist die Steuernummer eindeutig und 64 Bit lang und wird im B-Baum als Suchschlüssel verwendet.

Gehen Sie bei der Berechnung davon aus, dass eine Speicherseite 16 KiB groß ist und ein Knoten des B-Baums möglichst genau auf diese Seite passen sollte. (1 KiB = 2^{10} Byte)

Lösungsvorschlag 04

- Annahme: 64-Bit Architektur (Zeigerlänge = 8 Byte)
- Bei Grad k, hat der Knoten 2k Schlüssel-Daten Paare aber 2k + 1 Verweise:

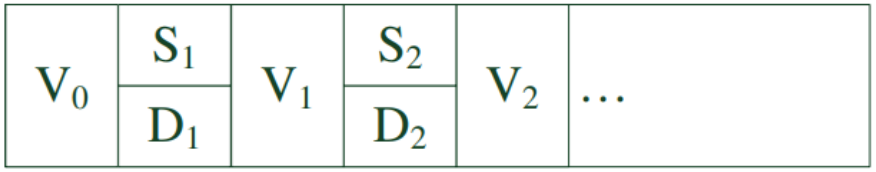


Abbildung 1: Struktur eines B-Baum Knotens

- Die Berechnung ergibt sich wie folgt:

$$
\begin{aligned}
2k \cdot (\text{Schlüsselgröße} + \text{Datengröße}) + (2k+1) \cdot \text{Zeigergröße} &\leq 16\ \text{KiB} \\
2k \cdot (8\ \text{Byte} + 100\ \text{Byte}) + (2k+1) \cdot 8\ \text{Byte} &\leq 16384\ \text{Byte} \\
2k \cdot 116\ \text{Byte} + 8\ \text{Byte} &\leq 16384\ \text{Byte} \\
2k &\leq 16376\ \text{Byte}/116\ \text{Byte} \\
k \leq (16376\ \text{Byte}/116\ \text{Byte})/2 &\approx 70,59
\end{aligned}
$$

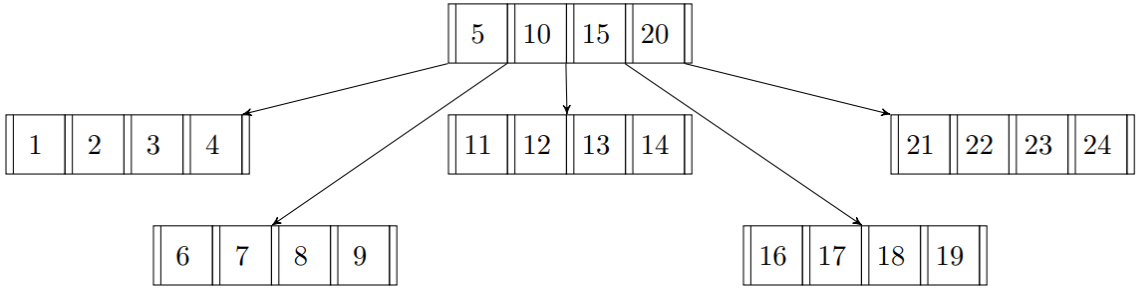
Aufgabe 05

Geben Sie eine Permutation der Zahlen 1 bis 24 an, so dass beim Einfügen dieser Zahlenfolge in einen (anfangs leeren) B-Baum mit Grad k = 2 ein Baum minimaler Höhe entsteht. Skizzieren Sie den finalen Baum

Lösungsvorschlag 05

- Alle Knoten müssen möglichst voll sein, damit die Höhe möglichst klein ist
- Mit Grad k=2, wäre es ideal, wenn wir insgesamt 6 Knoten mit jeweils 4 Elemente hätten (d.h. 1 Wurzel + 5 Blätter)

- Der einzig mögliche minimaler Baum ist der folgende:



- Damit die Zahlen 5, 10, 15 und 20 in die Wurzel geschrieben werden, muss die Einfügereihenfolge so gewählt werden, dass beim Teilen eines Knotens wegen Überlauf genau diese vier Zahlen als mittlere Elemente verwendet werden.
- Außerdem muss darauf geachtet werden, dass die Knoten nach dem Aufteilen wieder voll gefüllt werden.
- Eine mögliche Permutation: 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17, 20, 21, 22, 3, 4, 8, 9, 13, 14, 18, 19, 23, 24.