# Mehrwertige Abhängigkeiten

## multivalued dependencies (MVDs)

$\alpha \twoheadrightarrow \beta$

"Halb-formal":

- Seien $\alpha$ und $\beta$ disjunkte Teilmengen von $\mathcal{R}$
- und $\gamma = (\mathcal{R} \backslash \alpha) \backslash \beta$
- dann ist $\beta$ mehrwertig abhängig von $\alpha$ ($\alpha \twoheadrightarrow \beta$), wenn in jeder gültigen Ausprägung von $\mathcal{R}$ gilt:
- Bei zwei Tupeln mit gleichem $\alpha$-Wert kann man die $\beta$-Werte vertauschen, und die resultierenden Tupel müssen auch in der Relation enthalten sein.

Wichtige Eigenschaften:

- Jede FD ist auch eine MVD (gilt i.A. nicht umgekehrt)
- wenn $\alpha \twoheadrightarrow \beta$, dann gilt auch $\alpha \twoheadrightarrow \gamma$ (Komplementregel)
- $\alpha \twoheadrightarrow \beta$ ist trivial, wenn $\beta \subseteq \alpha$ ODER $\alpha \cup \beta = \mathcal{R}$ (also $\gamma = \emptyset$)

$\emptyset \twoheadrightarrow \emptyset$

# Beispiel: Mehrwertige Abhängigkeiten

Beispiel: $R = \{\text{Professor, Vorlesung, Assistent}\}$

| ProfessorIn | Vorlesung | AssistentIn |
|---|---|---|
| K | GDB | Linnea |
| K | WebDB | Linnea |
| K | GDB | Lukas |
| K | WebDB | Lukas |
| K | ERDB | Linnea |
| K | ERDB | Lukas |
| N | OO | Moritz |
| | | |
| | | |
| | | |
| | | |
| | | |

Professor ⇒ Vorlesung

Professor ⇒ Assistent

# Normalformen: 1NF $\supset$ 2NF $\supset$ 3NF $\supset$ BCNF $\supset$ 4NF

- **1. NF**: Attribute haben nur atomare Werte, sind also nicht mengenwertig.
- **2. NF**: Jedes Nichtschlüsselattribut (NSA) ist voll funktional abhängig von jedem Kandidatenschlüssel.
  - $\beta$ hängt **voll funktional** von $\alpha$ ab ($\alpha \overset{\bullet}{\rightarrow} \beta$), gdw. $\alpha \rightarrow \beta$ und es existiert kein $\alpha' \subset \alpha$, so dass $\alpha' \rightarrow \beta$ gilt.
- **3. NF**: Für alle geltenden nicht-trivialen FDs $\alpha \rightarrow \beta$ gilt entweder
  - $\alpha$ ist ein Superschlüssel, oder
  - jedes Attribut in $\beta$ ist in einem Kandidatenschlüssel enthalten
- **BCNF**: Die linken Seiten $(\alpha)$ aller geltenden nicht-trivalen FDs sind Superschlüssel.
- **4. NF**: Die linken Seiten $(\alpha)$ aller geltenden nicht-trivalen MVDs sind Superschlüssel.

# Übung: Höchste NF bestimmen

$$\mathcal{R} : \{[\ A, B, C, D, E\ ]\}$$

$$\begin{aligned} A &\rightarrow BCDE \\ AB &\rightarrow C \end{aligned}$$

- ☑ 1. NF
- ☑ 2. NF
- ☑ 3. NF
- ☑ BCNF
- ☑ 4. NF
- ○ keine der angegebenen

# Übung: Höchste NF bestimmen (2)

$\mathcal{R} : \{[\ A, B, C, D, E\ ]\}$

$A \rightarrow BCDE$

$B \rightarrow C$

- [x] 1. NF
- [x] 2. NF
- [ ] 3. NF
- [ ] BCNF
- [ ] 4. NF
- [ ] keine der angegebenen

# Schema in 3. NF überführen

## Synthesealgorithmus

- Eingabe:
  - **Kanonische Überdeckung** $\mathcal{F}_c$
    - Linksreduktion
    - Rechtsreduktion
    - FDs der Form $\alpha \to \emptyset$ entfernen (sofern vorhanden)
    - FDs mit gleicher linke Seite zusammenfassen
- Algorithmus:
  1. Für jede FD $\alpha \to \beta$ in $\mathcal{F}_c$ forme ein Unterschema $\mathcal{R}_\alpha = \alpha \cup \beta$, ordne $\mathcal{R}_\alpha$ die FDs $\mathcal{F}_\alpha := \{\alpha' \to \beta' \in \mathcal{F}_c \mid \alpha' \cup \beta' \subseteq \mathcal{R}_\alpha\}$ zu
  2. Füge ein Schema $\mathcal{R}_\kappa$ mit einem Kandidatenschlüssel hinzu
  3. Eliminiere redundante Schemata, d.h. falls $\mathcal{R}_i \subseteq \mathcal{R}_j$, verwerfe $\mathcal{R}_i$
- Ausgabe:
  - Eine Zerlegung des unsprünglichen Schemas, in der alle Schemata in 3.NF sind.
  - Die Zerlegung ist **abhängigkeitsbewahrend** und **verlustlos**.

# Übung: Synthesealgorithmus

$$\mathcal{R} : \{[\ A\,,\, B\,,\, C\,,\, D\,,\, E\,,\, F\ ]\}$$

$$\begin{aligned} B &\rightarrow A\cancel{C}\cancel{D}EF \\ EF &\rightarrow BC \\ A &\rightarrow D \end{aligned}$$

# Schema in BCNF überführen

**BCNF-Dekompositionsalgorithmus** (nicht abhängigkeitsbewahrend)

- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein $\mathcal{R}_i \in Z$ gibt, das nicht in BCNF ist:
  - Finde eine FD $(\alpha \to \beta) \in F^+$ mit
    - $\alpha \cup \beta \subseteq \mathcal{R}_i$ (FD muss in $\mathcal{R}_i$ gelten)
    - $\alpha \cap \beta = \emptyset$ (linke und rechte Seite sind disjunkt)
    - $\alpha \to \mathcal{R}_i \notin F^+$ (linke Seite ist kein Superschlüssel)
  - Zerlege $\mathcal{R}_i$ in $\mathcal{R}_{i.1} := \alpha \cup \beta$ und $\mathcal{R}_{i.2} := \mathcal{R}_i - \beta$
  - Entferne $\mathcal{R}_i$ aus $Z$ und füge $\mathcal{R}_{i.1}$ und $\mathcal{R}_{i.2}$ ein, also $Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i.1}\} \cup \{\mathcal{R}_{i.2}\}$

# Schema in 4.NF überführen

**4NF-Dekompositionsalgorithmus** (nicht abhängigkeitsbewahrend)

- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein $\mathcal{R}_i \in Z$ gibt, das nicht in 4NF ist:
  - Finde eine MVD $\alpha \twoheadrightarrow \beta \in \mathcal{F}^+$ mit
    - $\alpha \cup \beta \subset \mathcal{R}_i$ (FD muss in $\mathcal{R}_i$ gelten und nicht-trivial sein)
    - $\alpha \cap \beta = \emptyset$ (linke und rechte Seite sind disjunkt)
    - $\alpha \rightarrow \mathcal{R}_i \notin \mathcal{F}^+$ (linke Seite ist kein Superschlüssel)
  - Zerlege $\mathcal{R}_i$ in $\mathcal{R}_{i.1} := \alpha \cup \beta$ und $\mathcal{R}_{i.2} := \mathcal{R}_i - \beta$
  - Entferne $\mathcal{R}_i$ aus $Z$ und füge $\mathcal{R}_{i.1}$ und $\mathcal{R}_{i.2}$ ein, also $Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i.1}\} \cup \{\mathcal{R}_{i.2}\}$

# Übung: BCNF-Dekompositionsalgorithmus

$\mathcal{R} = \{A, B, C, D, E, F\}$

$F_{\mathcal{R}} = \{B \to AD, DEF \to B, C \to AE\}$

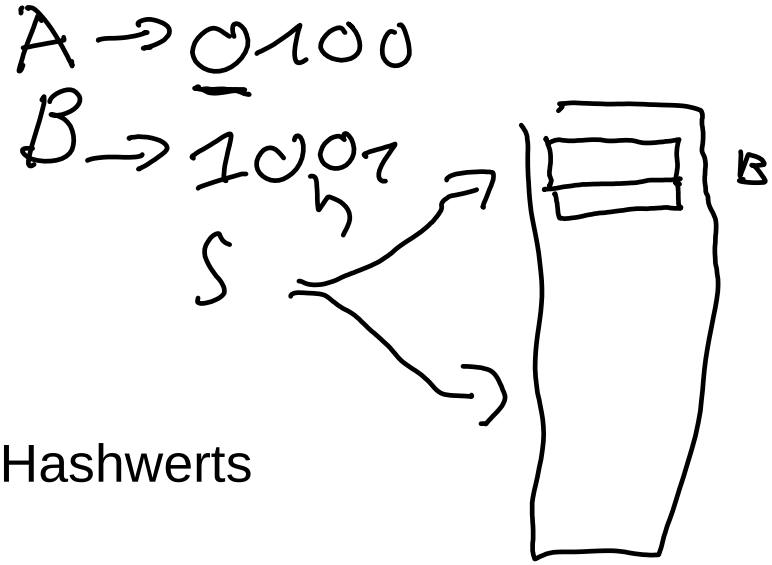
$R_1 = \{A, B, D\}$

$R_2 = \{B, C, E, F\}$

$R_{21} = \{C, E\}$

$R_{22} = \{B, C, F\}$

# Erweiterbares Hashing




Hashfunktion ***h*: *S*** → ***B***

Schlüssel — Bucket

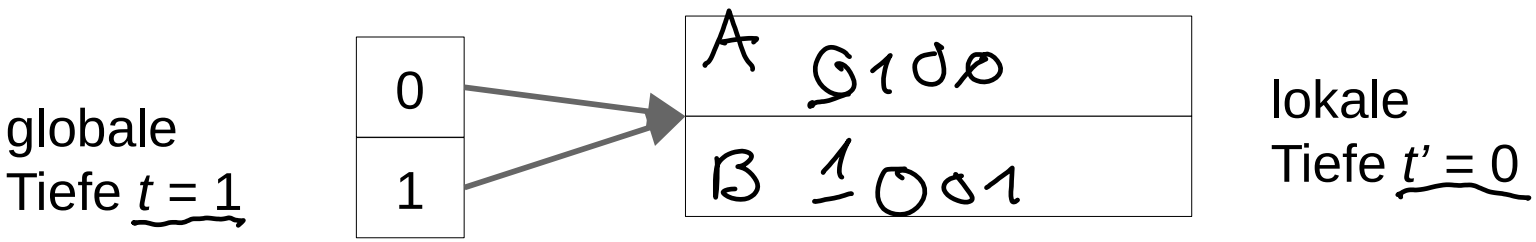
wir betrachten die **Binärdarstellung** des Hashwerts

*h(x) = dp*

unbenutzte Bits

Anzahl betrachteter Bits
= globale Tiefe des Dictionaries

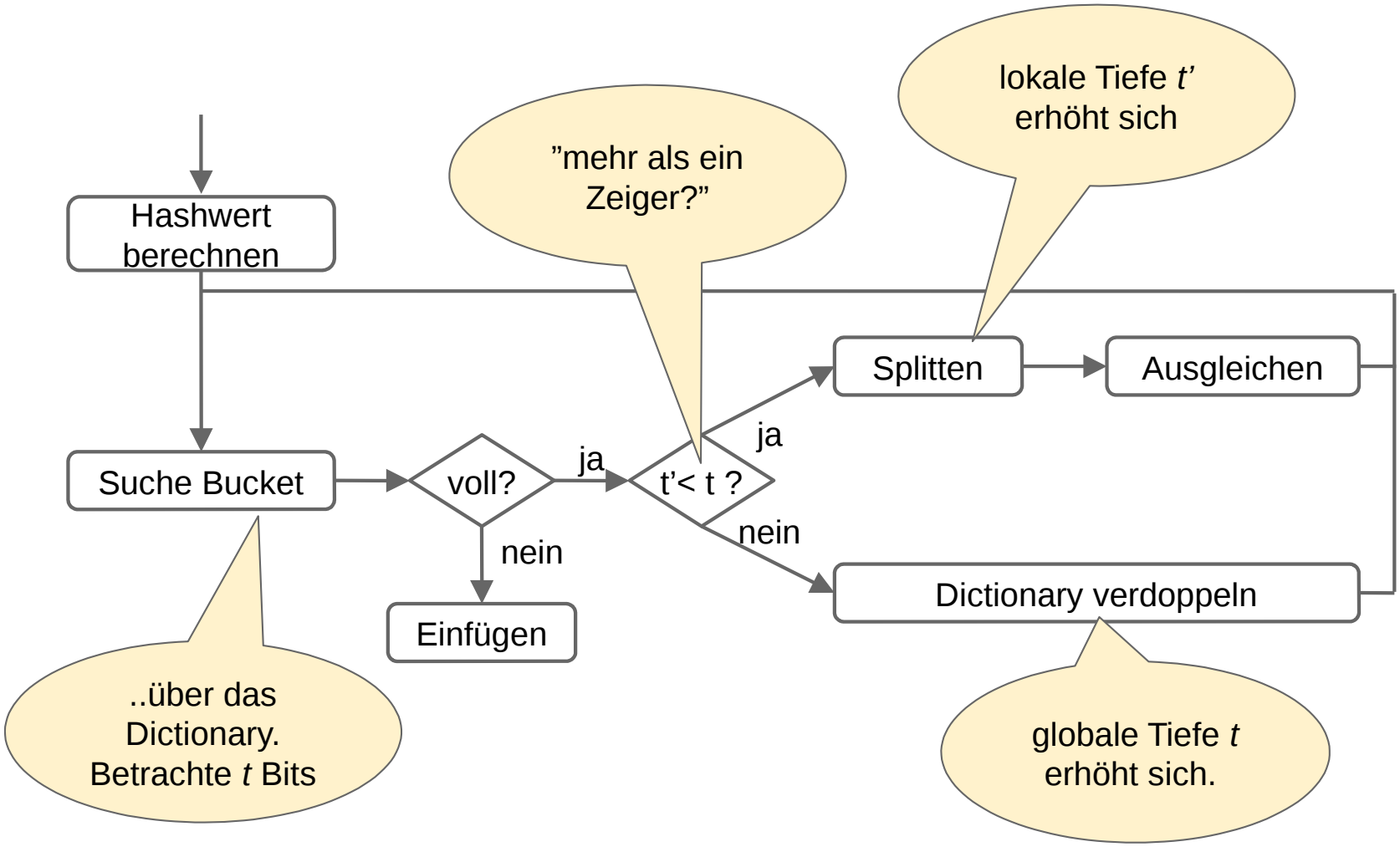
---




Dictionary
(Verzeichnis)

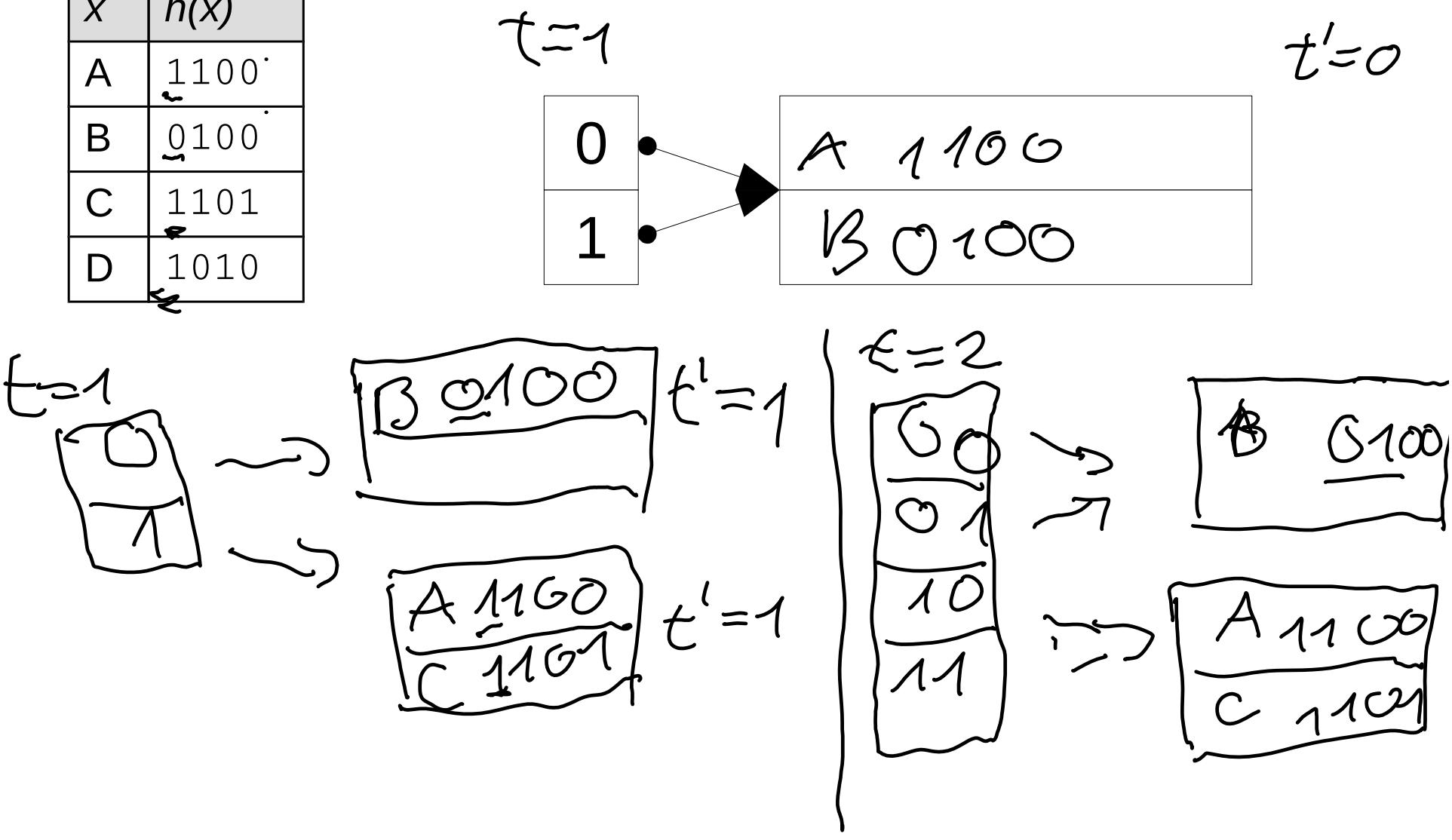
Bucket (Behälter)
Platz für *n* Einträge
hier *n*=2

# Erweiterbares Hashing / Einfügen



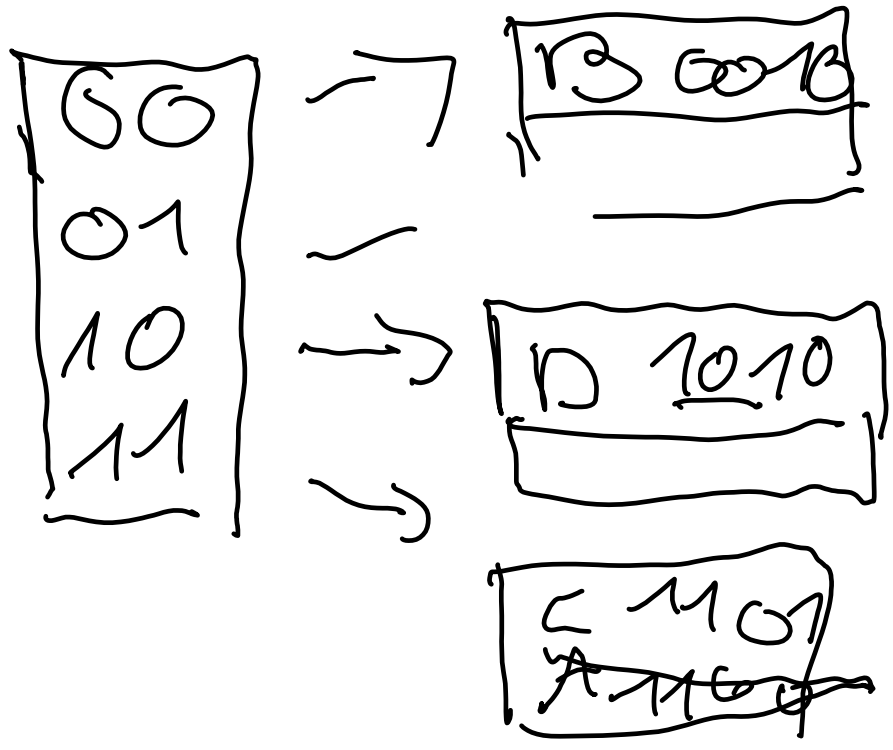
# Übung: Erweiterbares Hashing / Einfügen

| $x$ | $h(x)$ |
|---|---|
| A | 1100 |
| B | 0100 |
| C | 1101 |
| D | 1010 |



# Übung: Erweiterbares Hashing / Einfügen

| $x$ | $h(x)$ |
|---|---|
| A | 1100 |
| B | 0100 |
| C | 1101 |
| D | 1010 |

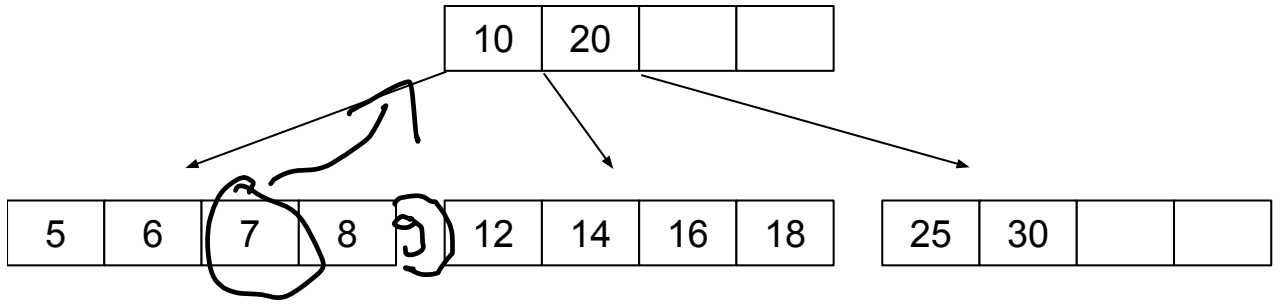


# B-Baum

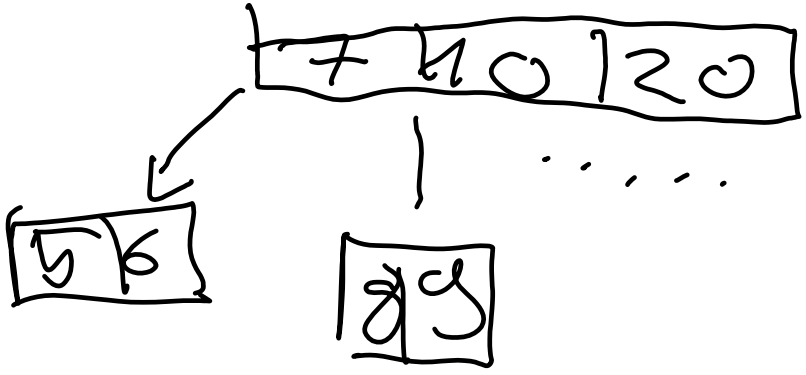
- Für Hintergrundspeicher konzipiert (1 Knoten $=$ 1 Seite, Knotenkapazität $>$ 1000)
- Hat Grad $k$: Jeder Knoten (außer Wurzel) mind. $k$, maximal $2k$ Elemente
  - Bei Überfüllung nach Einfügen: Aufteilen in $2$ Knoten.
  - Bei Unterbelegung nach Löschen: Ausgleich mit Nachbarn, oder Verschmelzung wenn Nachbar minimal belegt.
  - Aufteilen oder Verschmelzen setzt sich rekursiv bis zur Wurzel fort.
- B+-Bäume sind *hohl*, haben Werte nur in den Blättern:
  - hat daher Grad $(k, k^*)$. $k$ für innere Knoten, $k^*$ für Blattknoten.

# Übung: B-Baum (1)

Fügen Sie die Zahl **9** in den folgenden B-Baum ein.

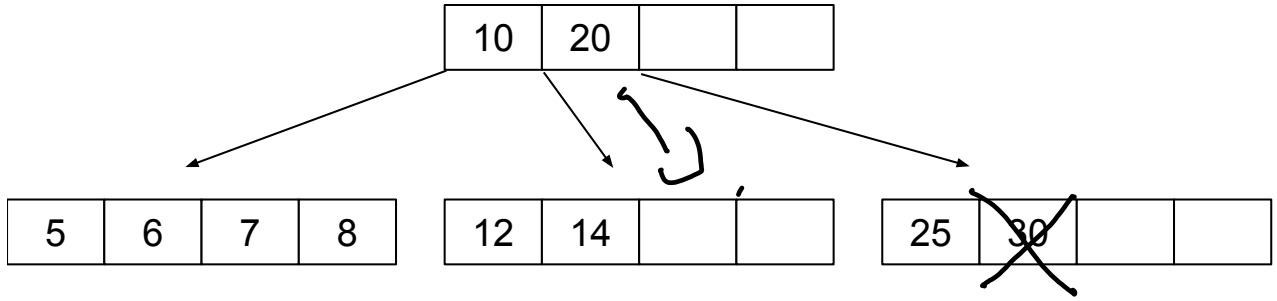



Zeichnen Sie den vollständigen, resultierenden Baum inklusive aller Referenzen.

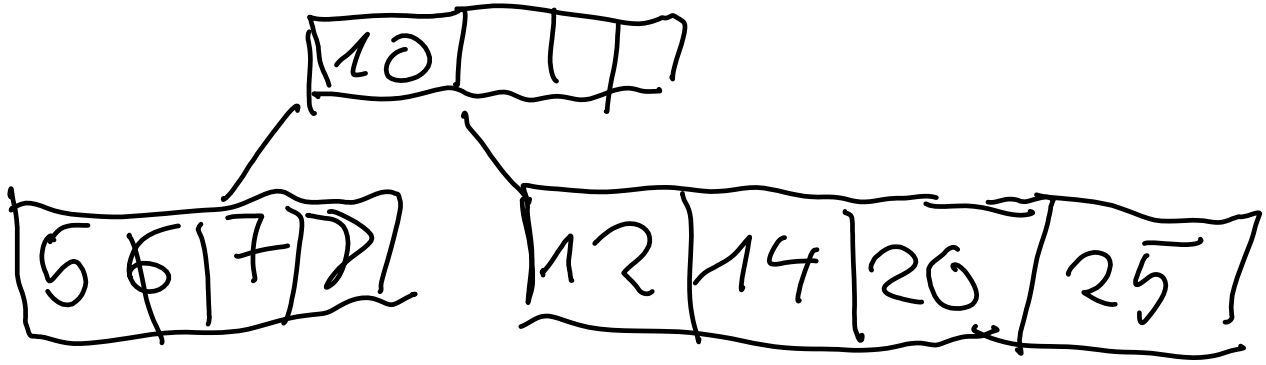


# Übung: B-Baum (2)

Löschen Sie die Zahl **30** im folgenden B-Baum. $k = 2$




Zeichnen Sie den vollständigen, resultierenden Baum inklusive aller Referenzen.

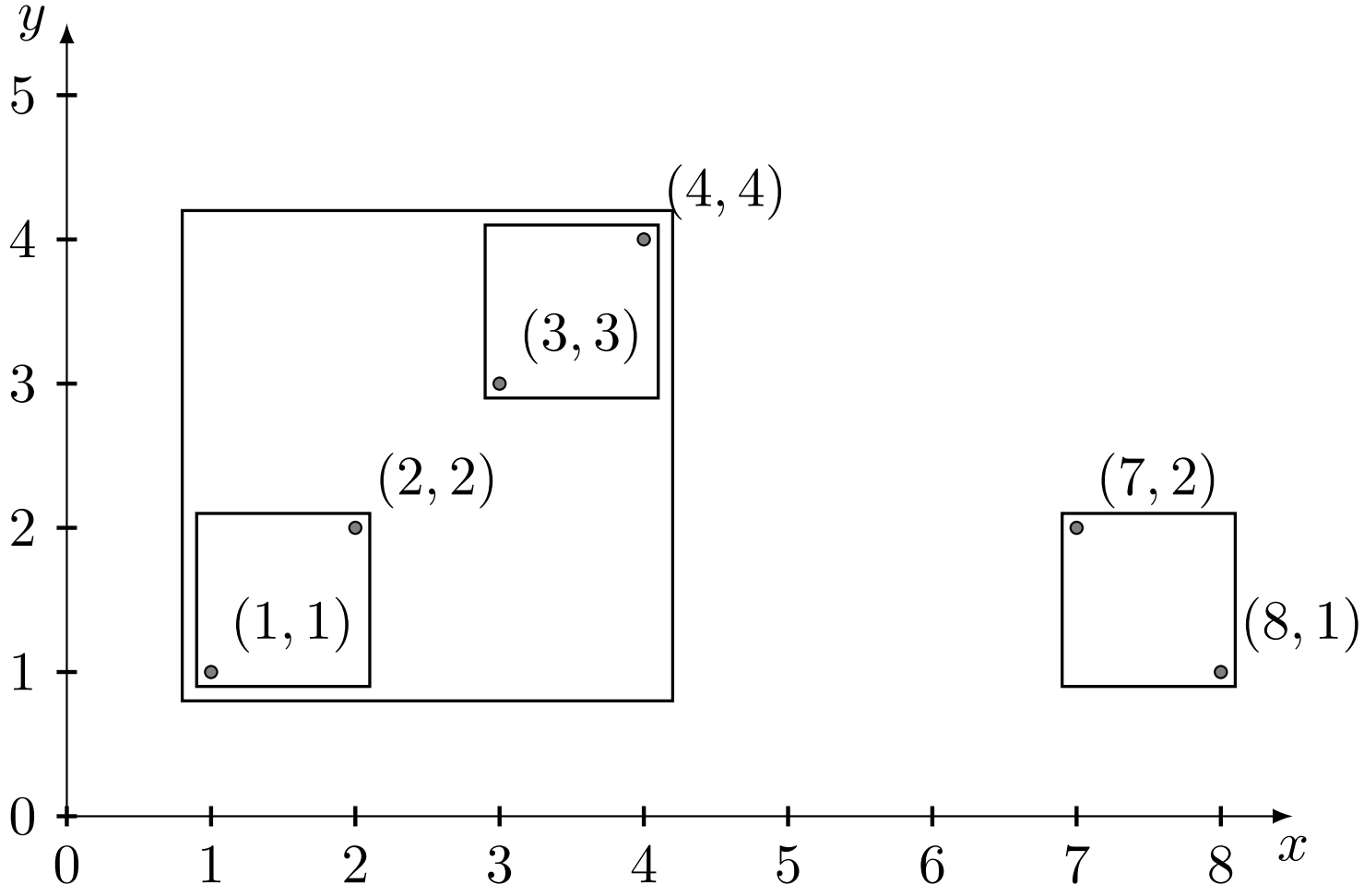


# R-Baum

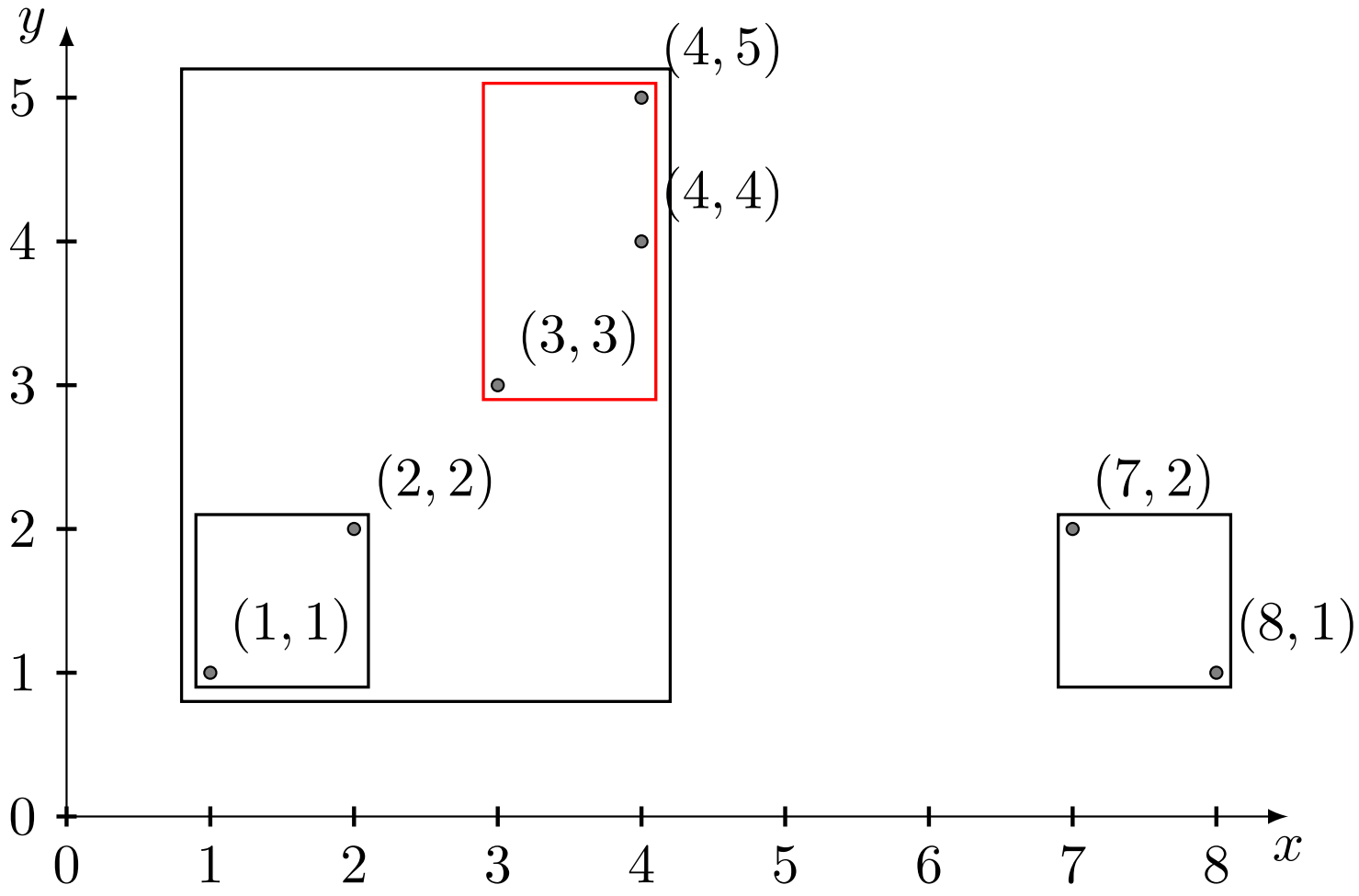
- Mehrdimensionale Indexstruktur: Punkte in n-dimensionalem Datenraum werden in Knoten aufgeteilt
- Funktionsweise ähnlich zu B+-Baum
- Split nicht eindeutig, da es nicht „die beste“ Möglichkeit gibt, Boxen aufzuteilen
- Punkt- und Bereichsabfragen müssen meist mehrere Pfade traversieren, da sich Boxen überschneiden können

# Übung: R-Baum

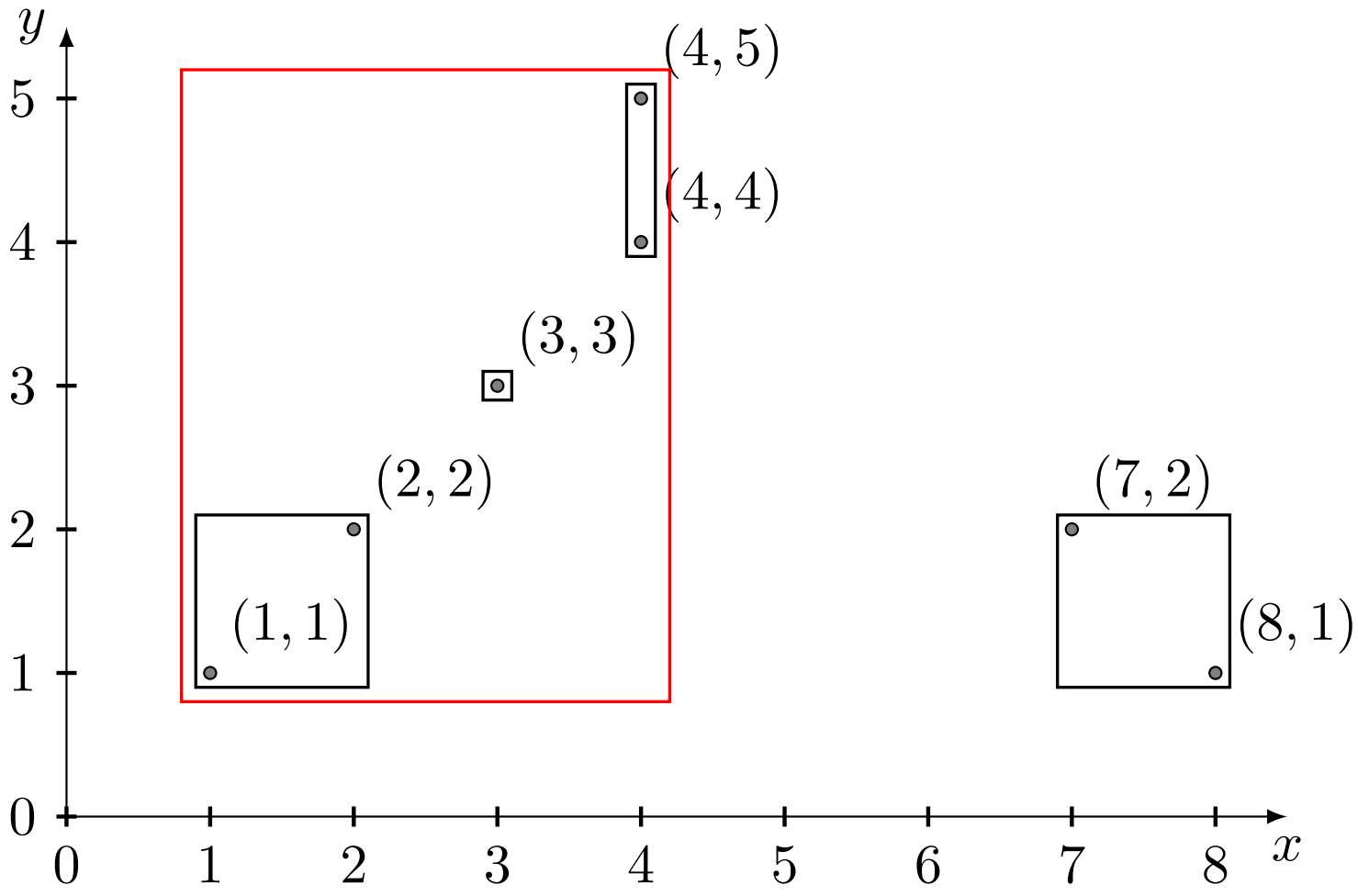
Fügen Sie den Wert $(4, 5)$ in den untenstehenden R-Baum mit der Knotenkapazität 2 ein und zeichnen Sie den resultierenden Datenraum.



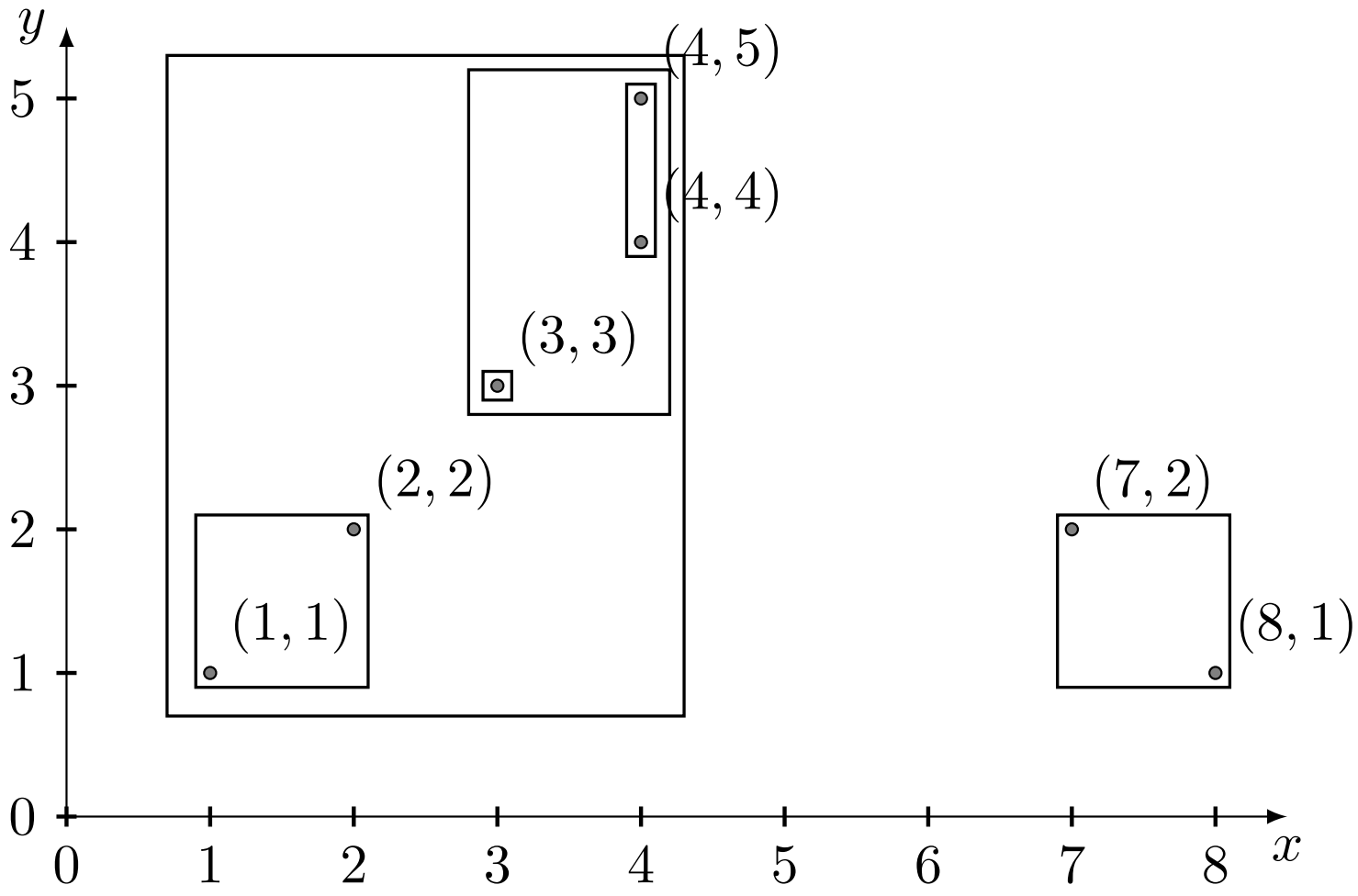
# Übung: R-Baum



# Übung: R-Baum



# Übung: R-Baum



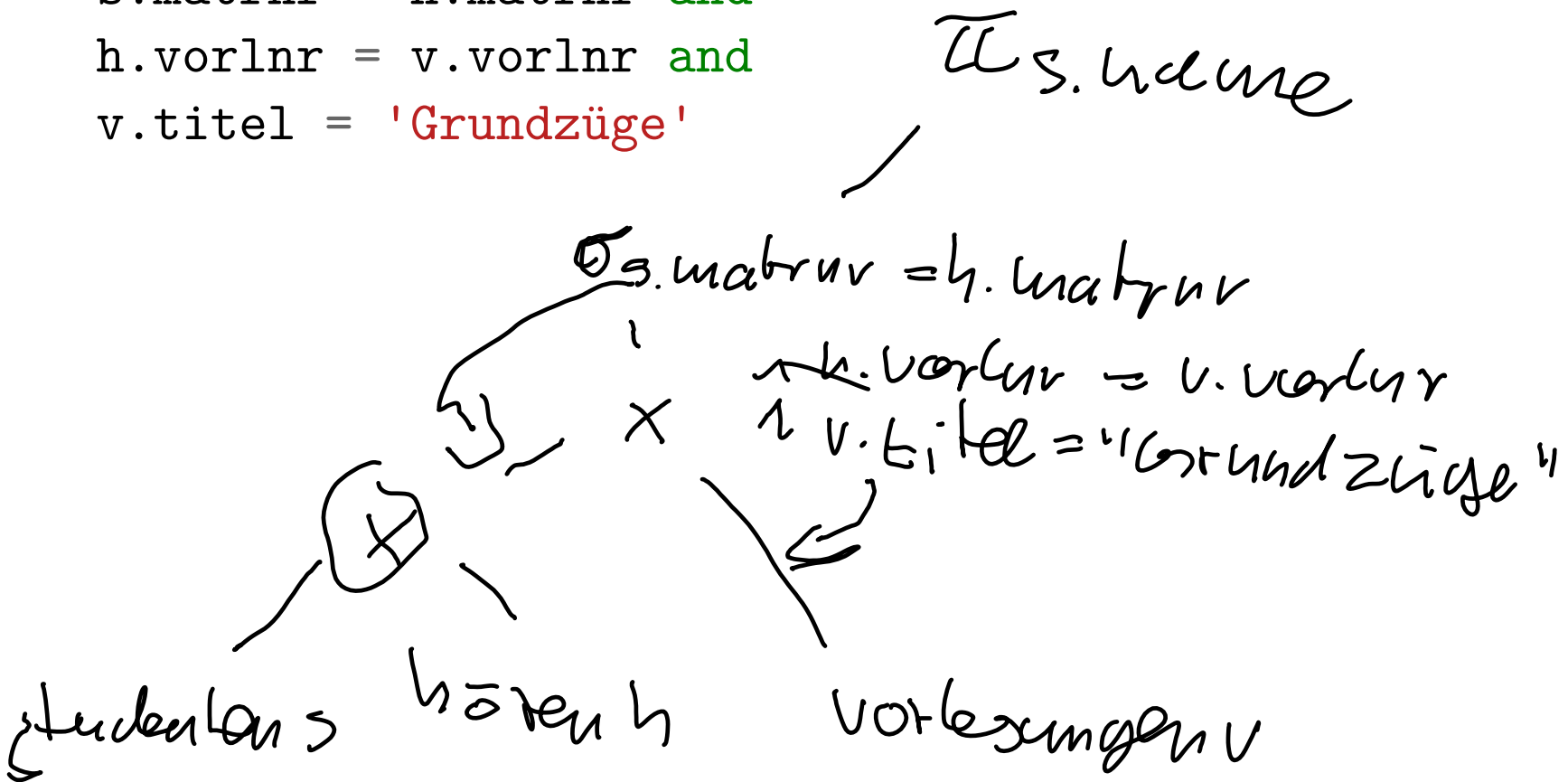
# Anfrageoptimierung

- Kanonische Übersetzung:
  - Projektion für alle Attribute in `select`
  - Selektion für `where`-Bedingung
  - Kreuzprodukte für alle Relationen in `from`
- Logische Optimierung, Ziel: Verringern der Zwischenergebnisse, Verwendung der Äquivalenzregeln:
  - Frühe Selektion („push down“)
  - Selektion + Kreuzprodukt ersetzen durch Join
  - Joinreihenfolge optimieren
- Physische Optimierung: Wahl des Joinalgorithmus:
  - Nested-Loop-Join
  - Sort-Merge-Join
  - Hash-Join
  - Index-Join

# Übung: Anfrageoptimierung

Geben Sie die kanonische Übersetzung der folgenden SQL-Anfrage an und optimieren Sie diese logisch:

```
select distinct s.name
from studenten s, hören h, vorlesungen v
where
    s.matrnr = h.matrnr and
    h.vorlnr = v.vorlnr and
    v.titel = 'Grundzüge'
```



# Übung: Anfrageoptimierung (2)

Angenommen

- $|s| = 10000$
- $|h| = 20 * |s| = 200000$
- $|v| = 1000$
- 10% der Studenten haben 'Grundzüge' gehört

Dann ergeben sich

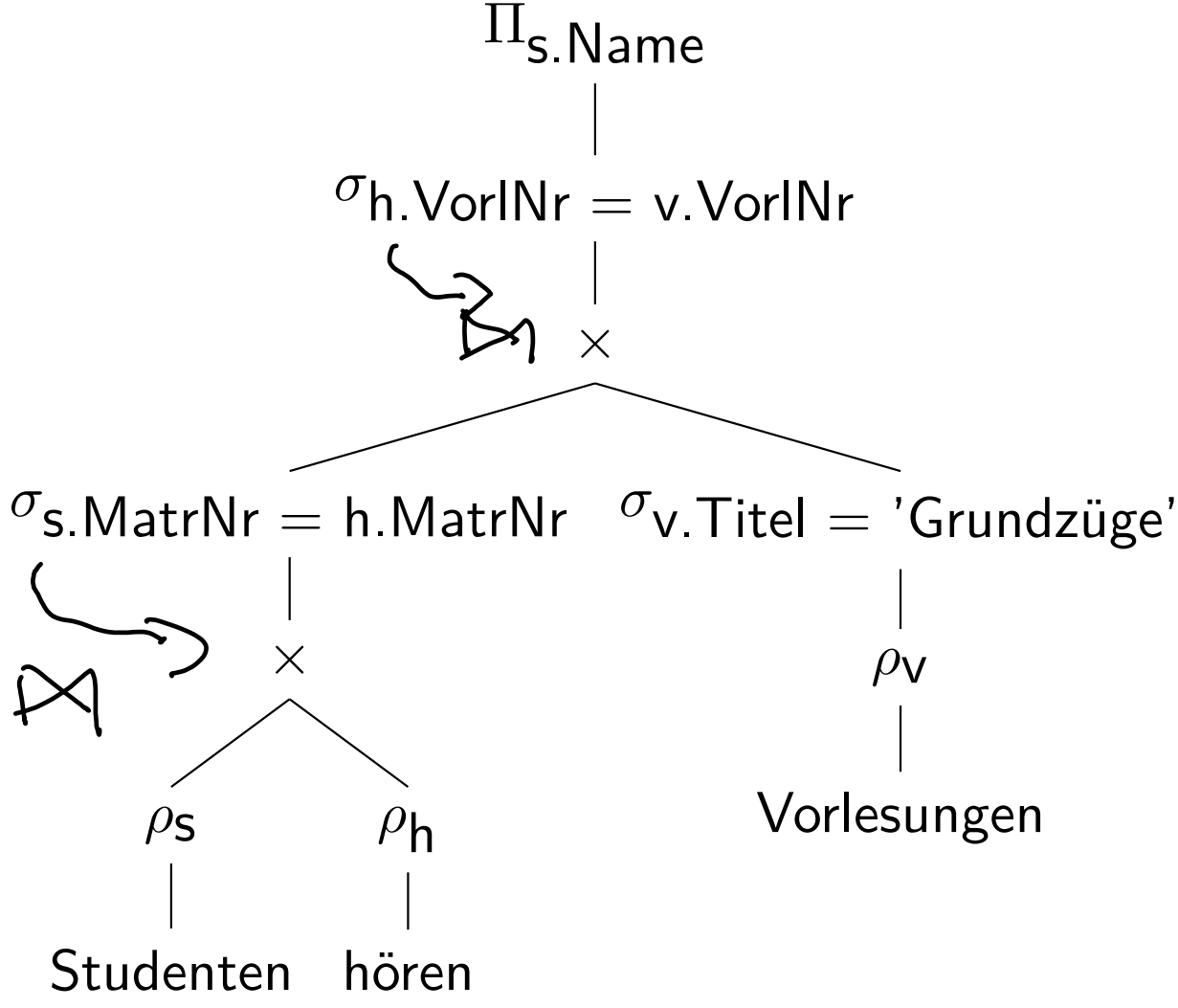
- $|s \times h \times v| = 10000 \cdot 20 \cdot 10000 \cdot 1000 = 2 \cdot 10^{12}$

Nach der Selektion verbleiben noch

- $|\sigma_p(s \times h \times v)| = 0,1 \cdot |s| = 1000$

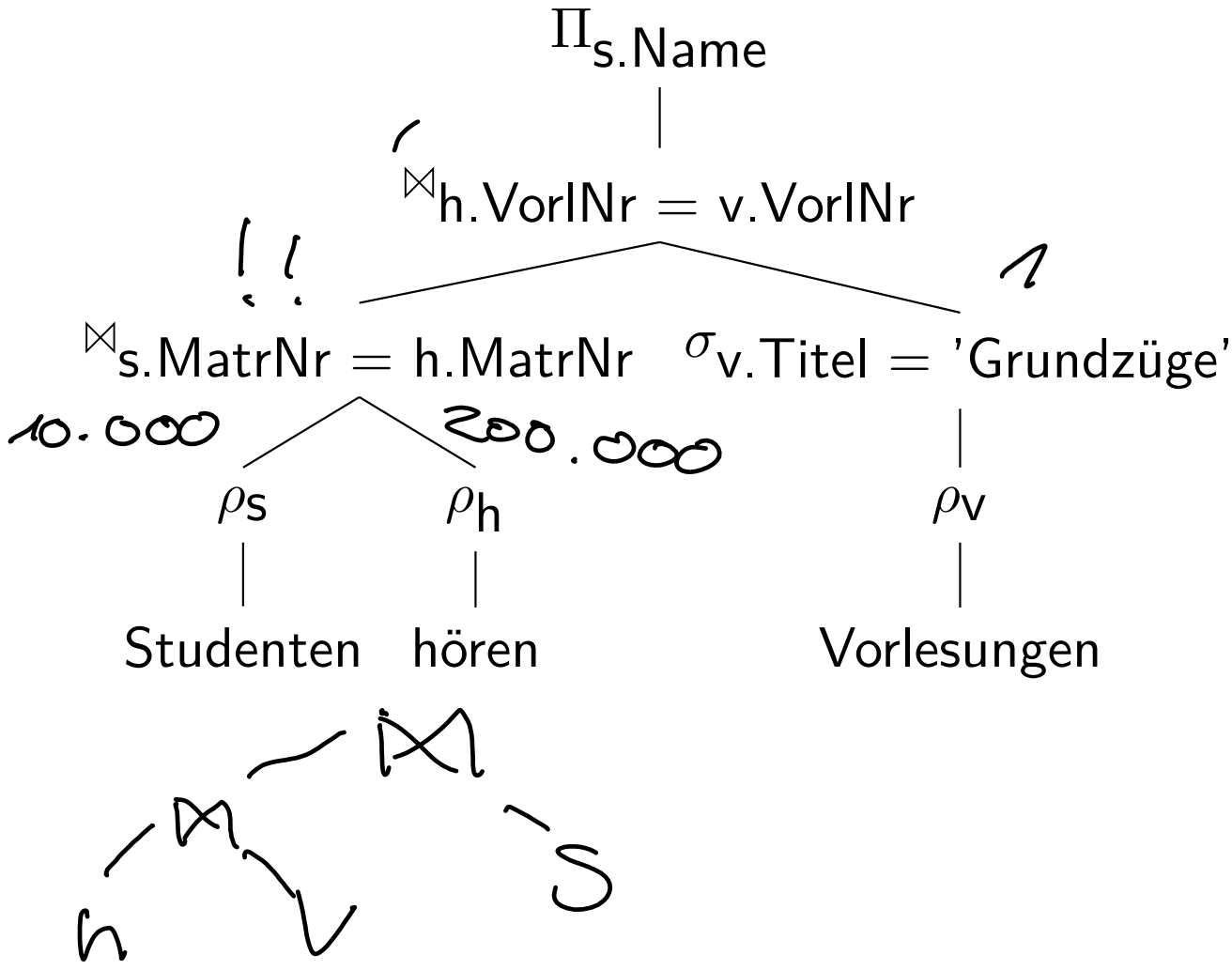
# Übung: Anfrageoptimierung (3)

**Optimierung 1**: Selektionen frühzeitig ausführen (*push selections*):



# Übung: Anfrageoptimierung (4)

**Optimierung 2**: Kreuzprodukte durch Joins ersetzen (*introduce joins*):



# Übung: Anfrageoptimierung (5)

**Optimierung 3**: Joinreihenfolge optimieren (*join order optimization*), so dass die Zwischenergebnismengen möglichst klein sind: