Datenbanksysteme

Eine Einführung

2. aktualisierte und erweiterte Auflage

Alfons Kemper und André Eickler

Universität Passau
Lehrstuhl für Dialogorientierte Systeme
Tel. (0851) 509-3060
Fax (0851) 598-3962
email: kemper/eickler@db.fmi.uni-passau.de
WWW: http://www.db.uni-passau.de/ kemper

Oldenbourg Verlag, München, 1997
ISBN: 3-486-24136-2

Literatur

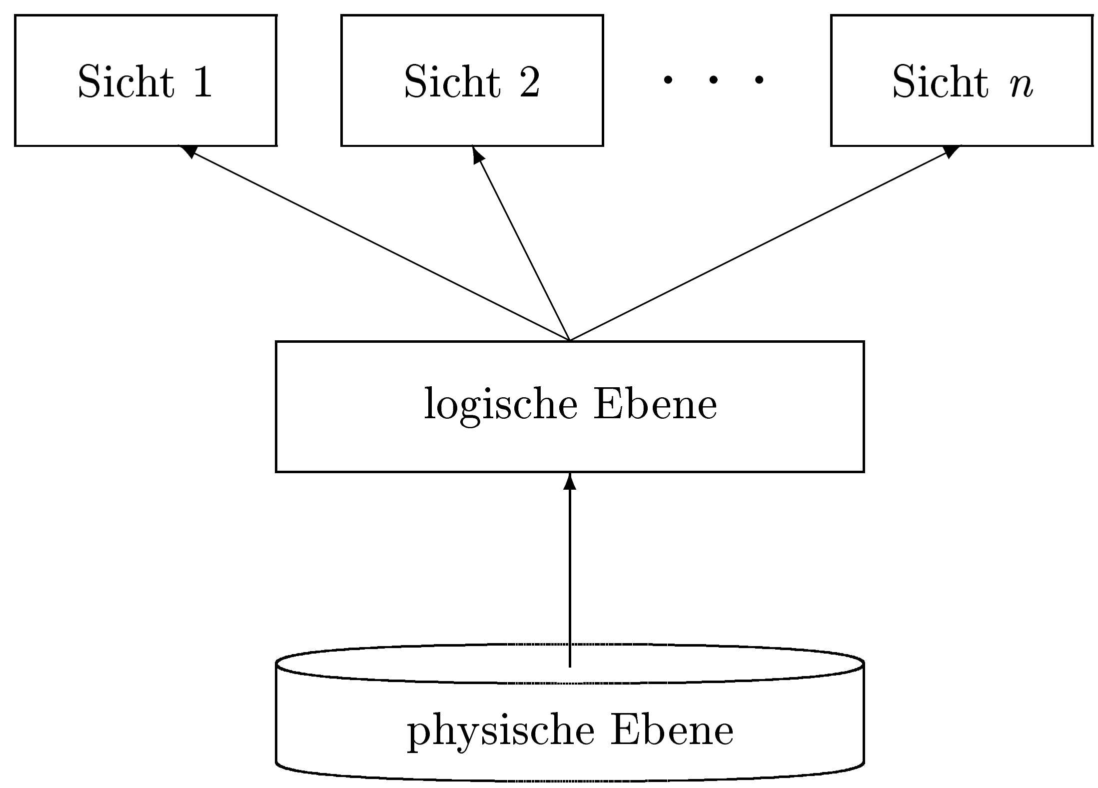
- A. Kemper, A. Eickler
  Datenbanksysteme – Eine Einführung. 2. aktualisierte und erweiterte Auflage. Oldenbourg Verlag, 1997.
- A. Silberschatz, H. F. Korth und S. Sudarshan
  Database System Concepts, 3. Auflage, McGraw Hill Book Co., 1997.
- R. Elmasri, S.B. Navathe: Fundamentals of Database Systems, Benjamin Cummings, Redwood City, Ca, USA, 2. Auflage, 1994.
- G. Vossen: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme. Addison Wesley, 1994.
- D. Maier
  The Theory of Relational Databases. Computer Science Press. 1983.
- S. M. Lang, P. C. Lockemann
  Datenbankeinsatz, Springer Verlag, 1995.
- C. Batini, S. Ceri, S. B. Navathe: Conceptual Database Design, Benjamin Cummings, Redwood City, Ca, USA 1992.
- C.J. Date
  An Introduction to Database Systems, 6. Auflage, Addison–Wesley Publishing Company, 1994.
- J.D. Ullman, J. Widom
  A First Course in Database Systems, McGraw Hill Book Co., 1997.
- A. Kemper, G. Moerkotte: Object-Oriented Database Management: Applications in Engineering and Computer Science, Prentice Hall, 1994.
- E. Rahm.
  Mehrrechner-Datenbanksysteme. Addison–Wesley, 1994.
- P. Dadam
  Verteilte Datenbanken und Client/Server Systeme,
  Springer Verlag,1996

Motivation für den Einsatz eines DBMS

Typische Probleme bei Informationsverarbeitung ohne DBMS:

- Redundanz und Inkonsistenz
- beschränkte Zugriffsmöglichkeiten
- Probleme beim Mehrbenutzerbetrieb
- Verlust von Daten
- Integritätsverletzung
- Sicherheitsprobleme
- hohe Entwicklungskosten für Anwendungsprogramme

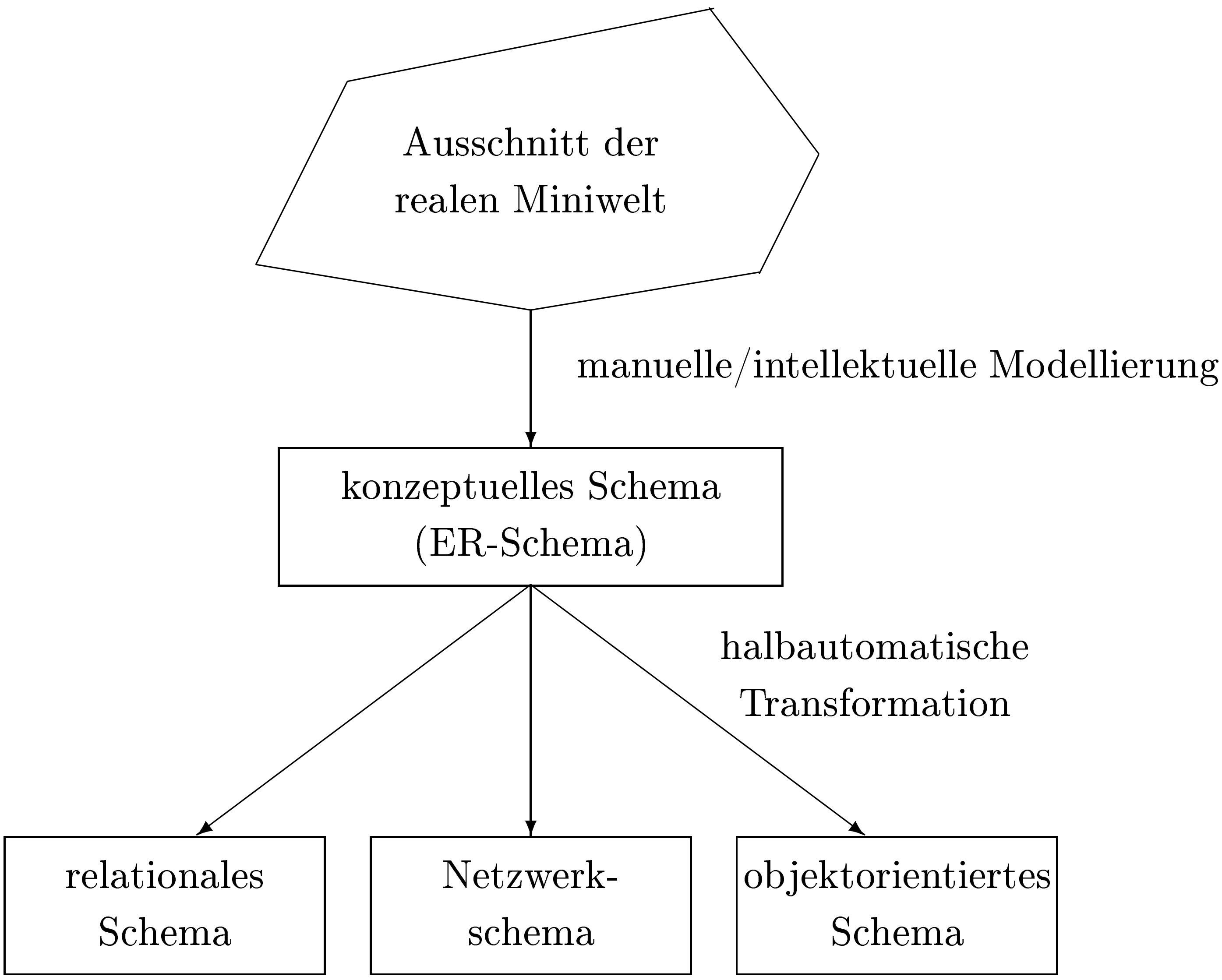
Die Abstraktionsebenen eines Datenbanksystems



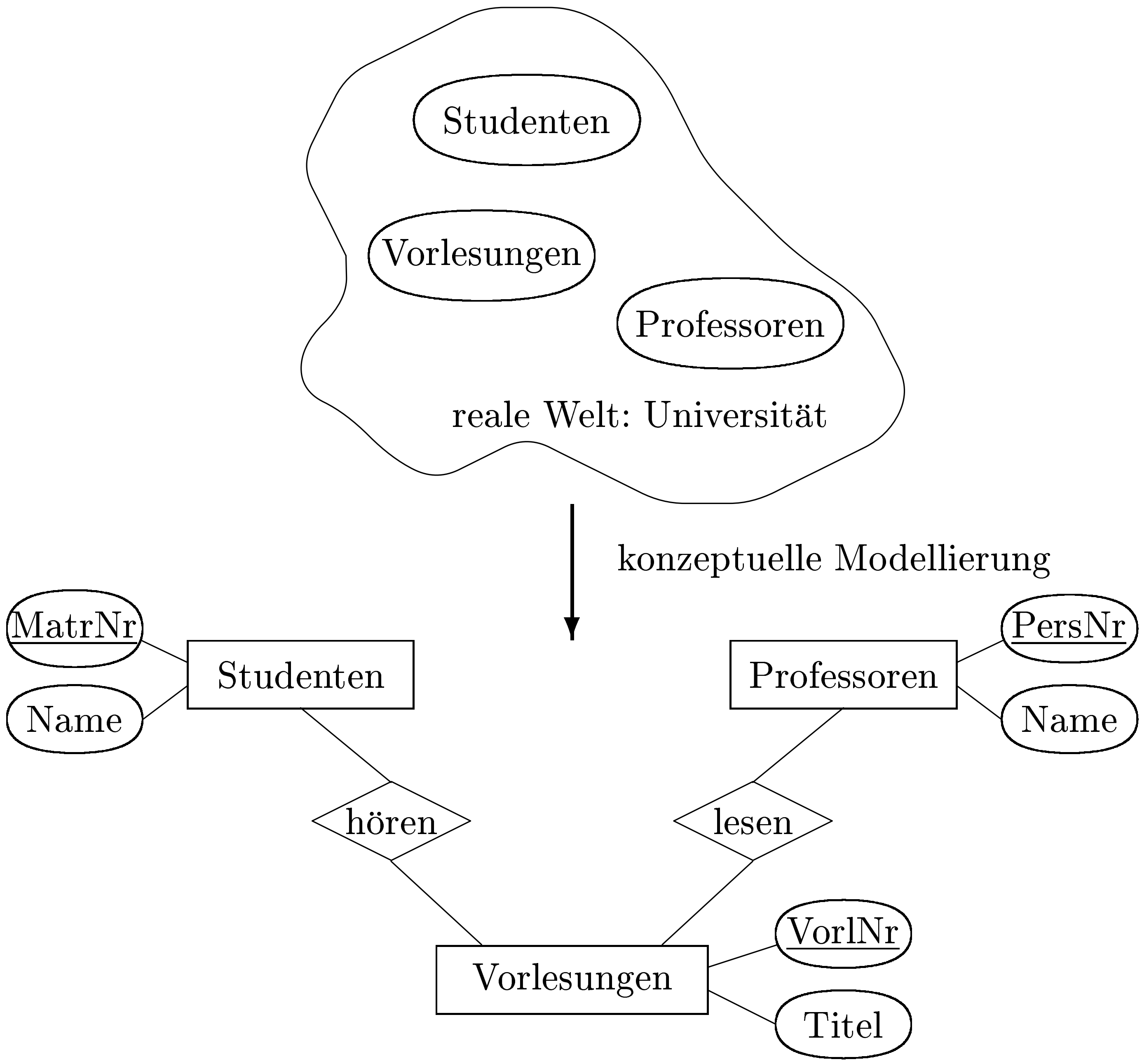
Datenunabhängigkeit:

- physische Datenunabhängigkeit
- logische Datenunabhängigkeit

Datenmodellierung



Modellierung einer kleinen Beispielanwendung



Logische Datenmodelle

- Netzwerkmodell
- hierarchisches Datenmodell
- relationales Datenmodell
- objektorientiertes Datenmodell
- deduktives Datenmodell

Das relationale Datenmodell

Studenten	
MatrNr	Name
26120	Fichte
25403	Jonas
...	...

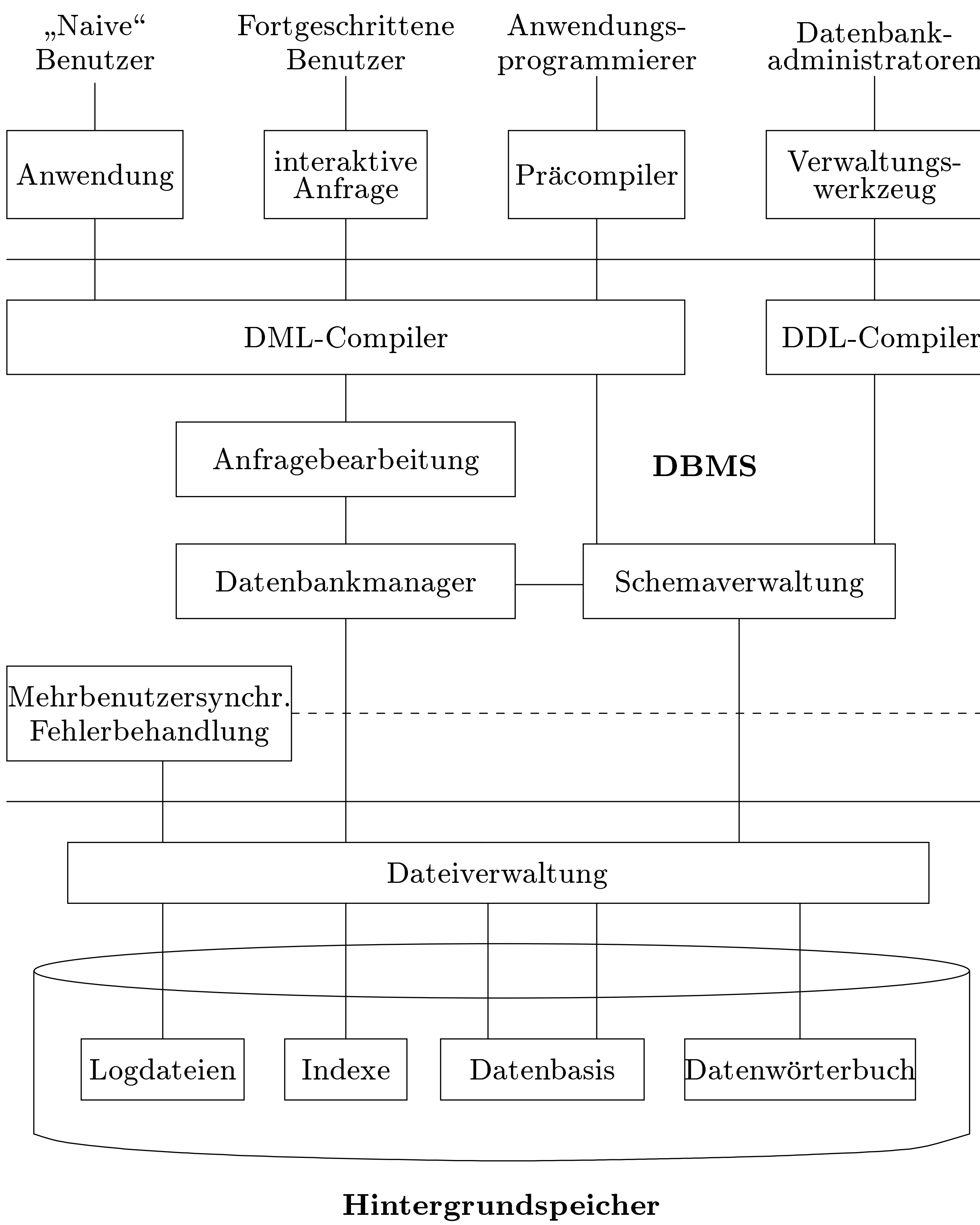
hören	
MatrNr	VorlNr
25403	5022
26120	5001
...	...

Vorlesungen	
VorlNr	Titel
5001	Grundzüge
5022	Glaube und Wissen
...	...

select Name
from Studenten, hören, Vorlesungen
where Studenten.MatrNr = hören.MatrNr **and**
hören.VorlNr = Vorlesungen.VorlNr **and**
Vorlesungen.Titel = 'Grundzüge';

update Vorlesungen
set Titel = 'Grundzüge der Logik'
where VorlNr = 5001;

Architekturübersicht eines DBMS



Datenbankentwurf

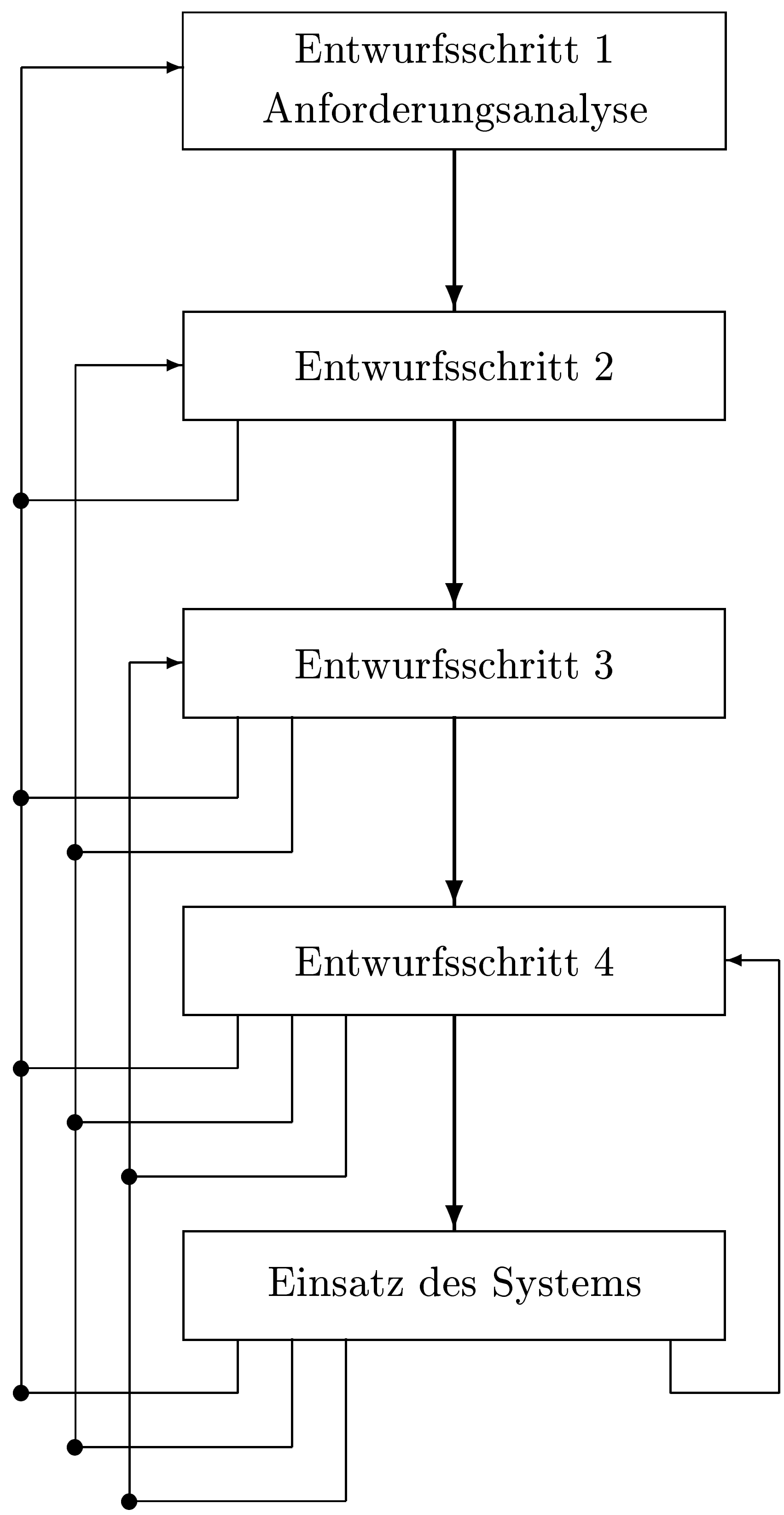
Abstraktionsebenen des Datenbankentwurfs

1. konzeptuelle Ebene

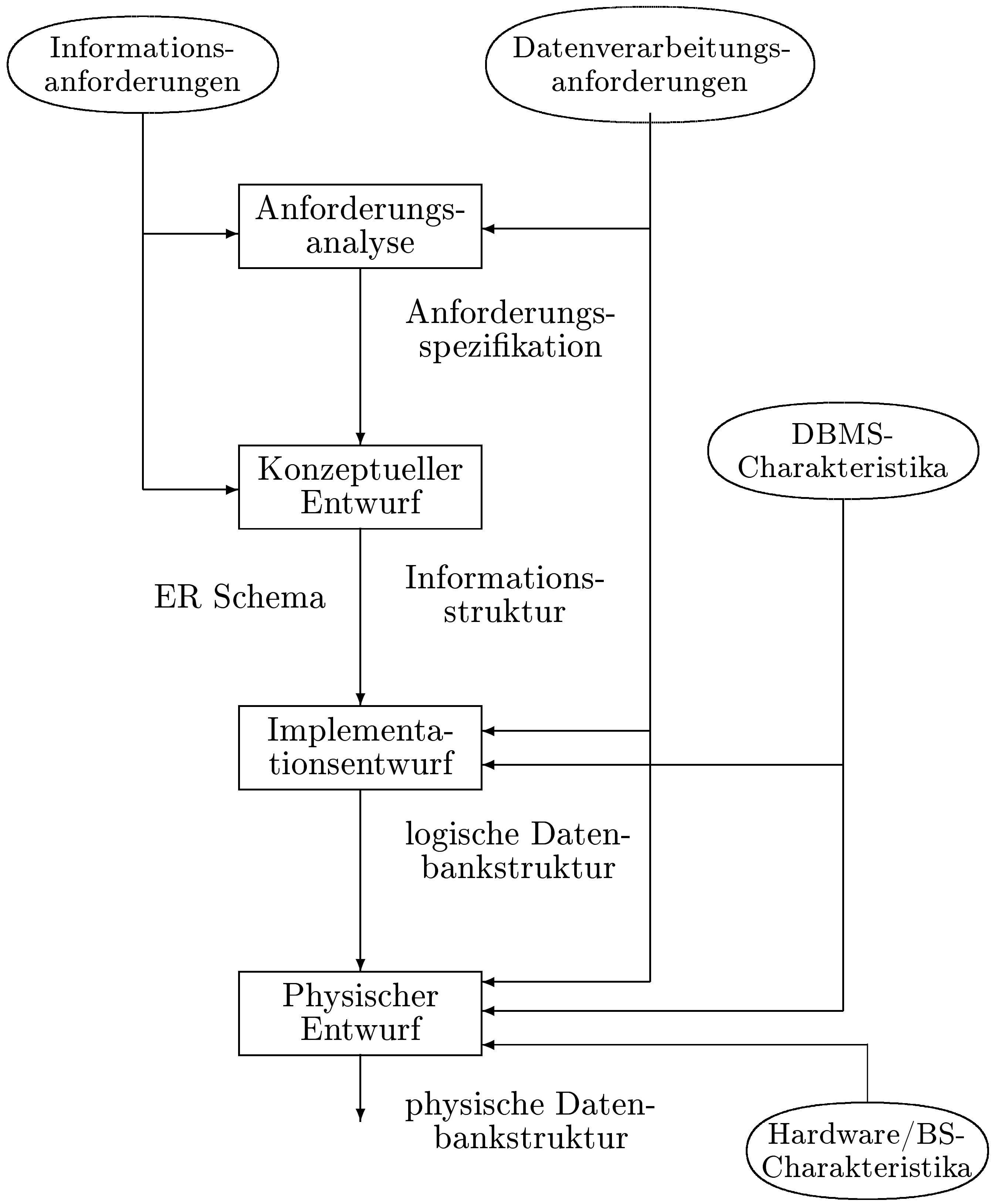
2. Implementationsebene

3. physische Ebene

Allgemeiner „top-down Entwurf“



Phasen des Datenbankentwurfs



Anforderungsanalyse

1. Identifikation von Organisationseinheiten
2. Identifikation der zu unterstützenden Aufgaben
3. Anforderungs-Sammelplan
4. Anforderungs-Sammlung
5. Filterung
6. Satzklassifikationen
7. Formalisierung

Objektbeschreibung

- Uni-Angestellte
  - Anzahl: 1000
  - Attribute
    - PersonalNummer
      - Typ: char
      - Länge: 9
      - Wertebereich: 0 . . . 999.999.99
      - Anzahl Wiederholungen: 0
      - Definiertheit: 100%
      - Identifizierend: ja
    - Gehalt
      - Typ: dezimal
      - Länge: (8,2)
      - Anzahl Wiederholungen: 0
      - Definiertheit: 10%
      - Identifizierend: nein
    - Rang
      - . . .
      - . . .
      - . . .

Beziehungsbeschreibung: *prüfen*

- Beteiligte Objekte:
  - Professor als Prüfer
  - Student als Prüfling
  - Vorlesung als Prüfungsstoff
- Attribute der Beziehung
  - Datum
  - Uhrzeit
  - Note
- Anzahl: 100 000 (pro Jahr)

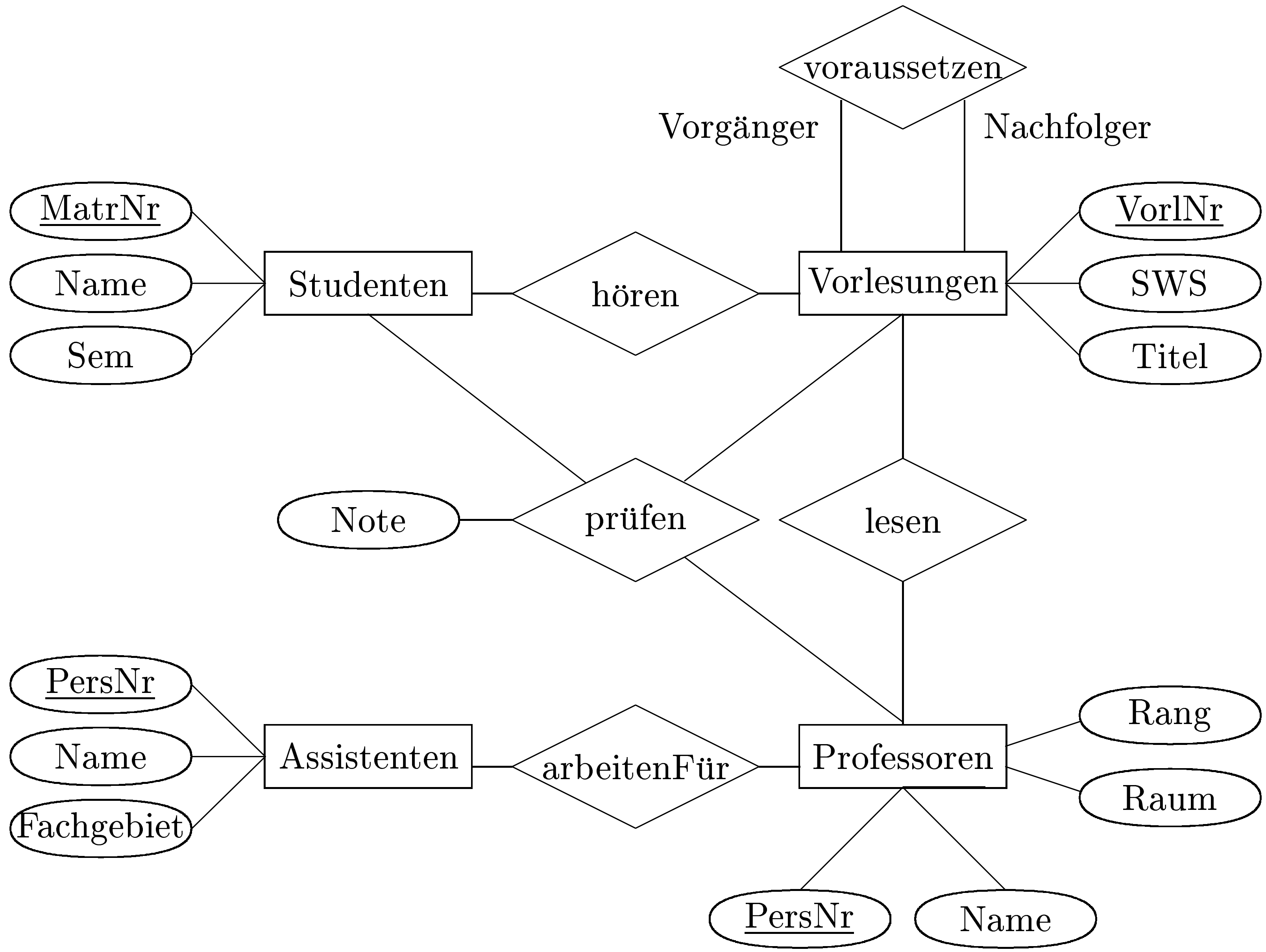
Prozeßbeschreibung: Zeugnisausstellung

- **Prozeßbeschreibung:** *Zeugnisausstellung*
  - Häufigkeit: halbjährlich
  - benötigte Daten
    - Prüfungen
    - Studienordnungen
    - Studenteninformation
    - ...
  - Priorität: hoch
  - zu verarbeitende Datenmenge
    - 500 Studenten
    - 3000 Prüfungen
    - 10 Studienordnungen

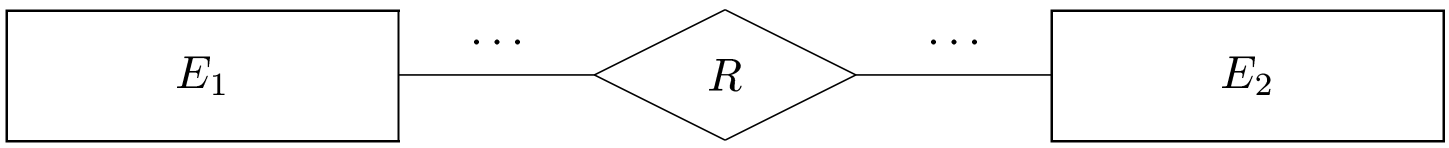
Entity-Relationship Modellierung

- Entity (Gegenstandstyp)
- Relationship (Beziehungstyp)
- Attribut (Eigenschaft)
- Schlüssel
- Rolle

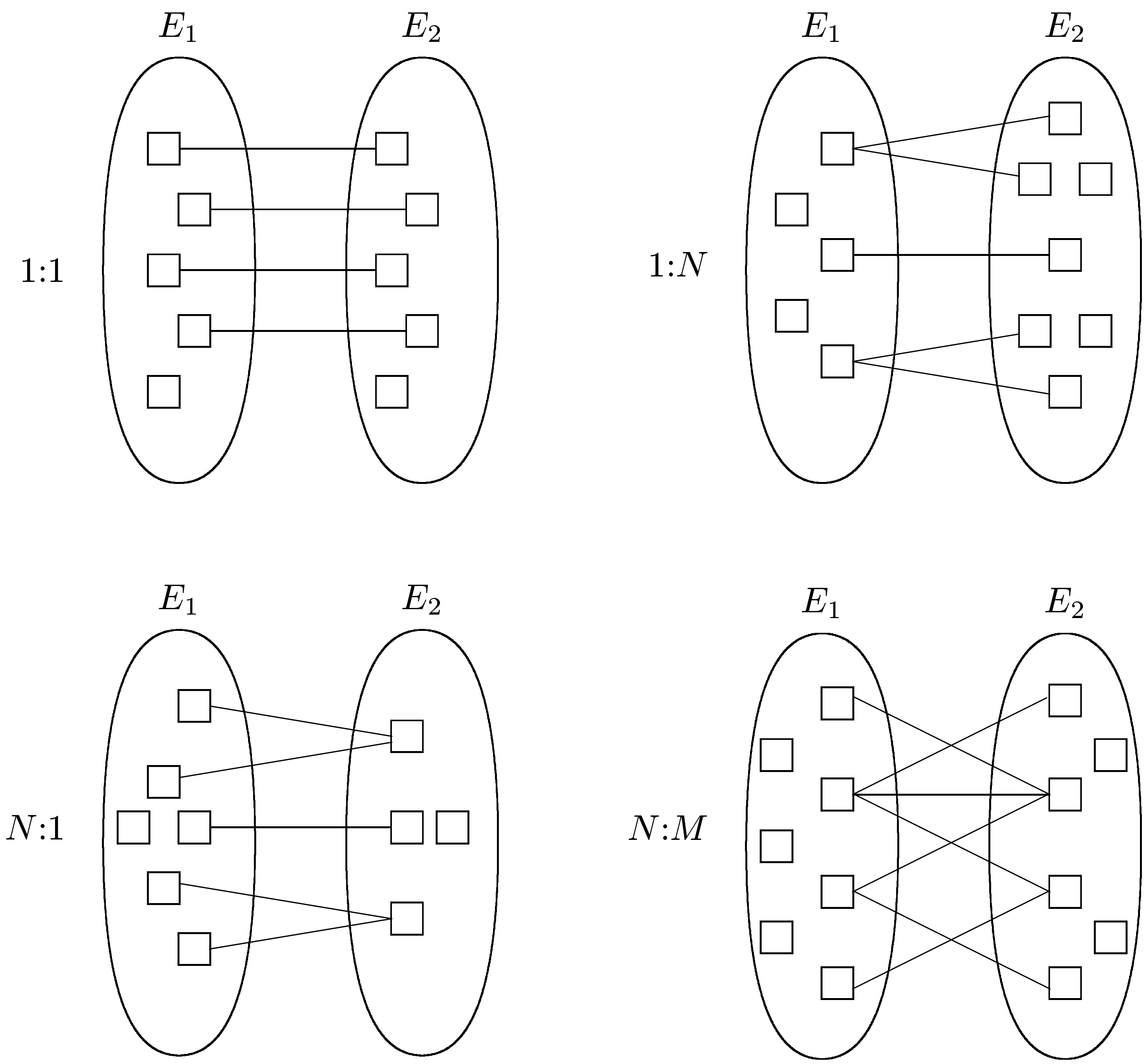
Universitätsschema



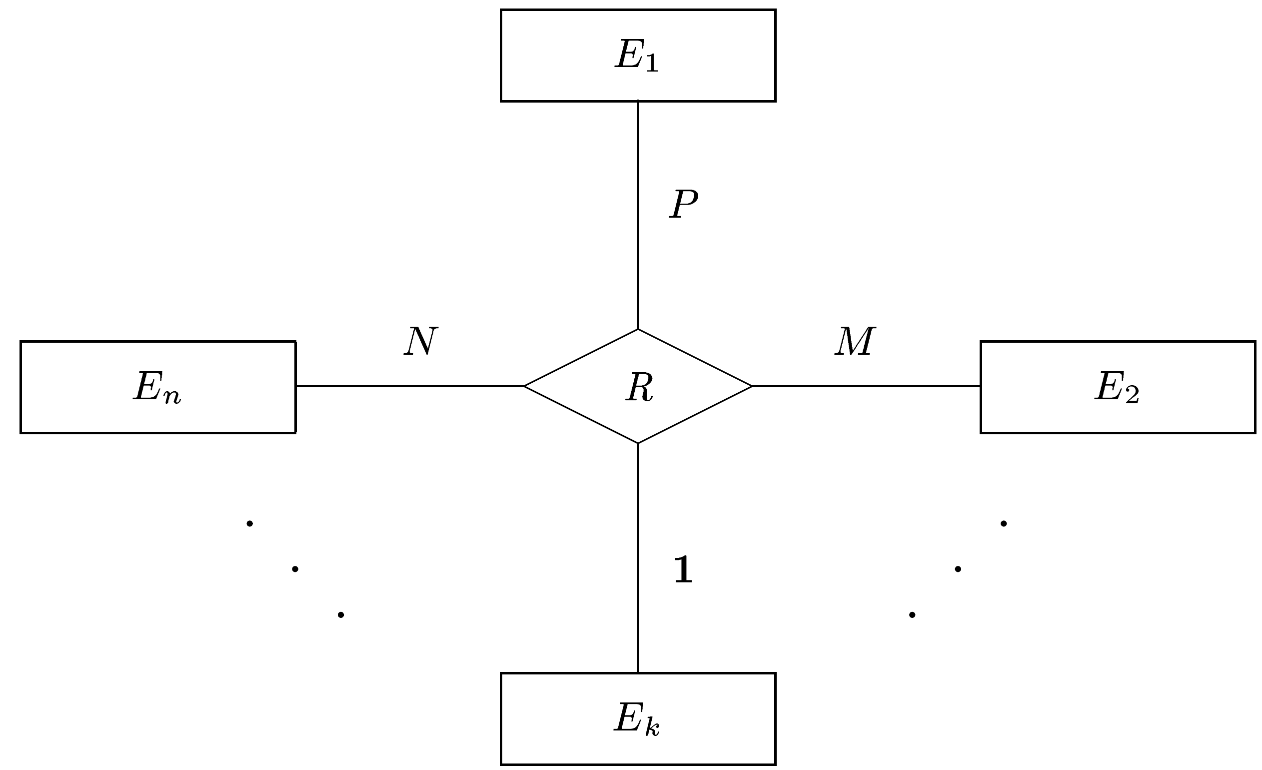
Funktionalitäten



$$R \subseteq E_1 \times E_2$$

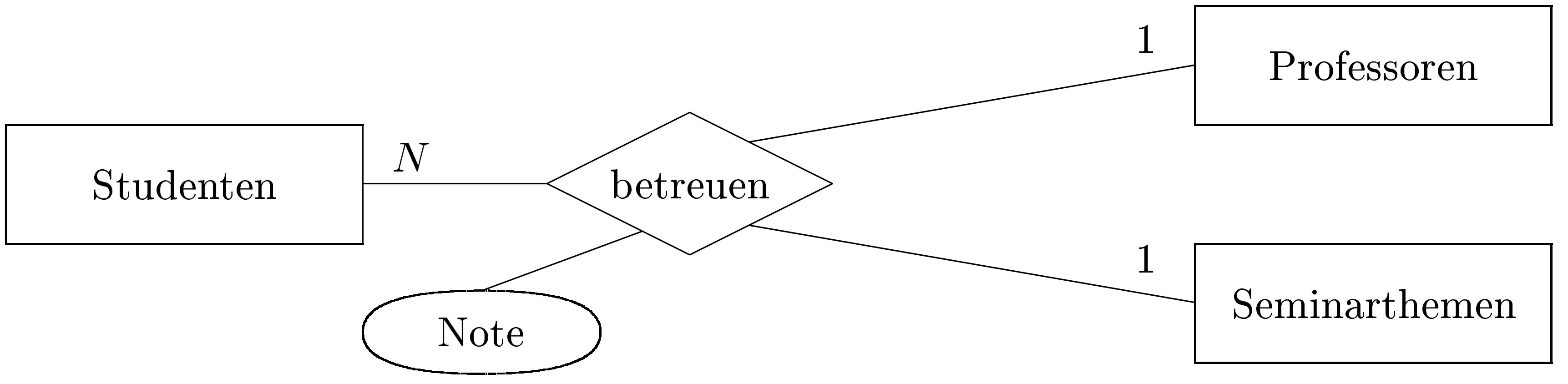


Funktionalitätsangaben bei n-stelligen Beziehungen



$$R : E_1 \times \ldots \times E_{k-1} \times E_{k+1} \times \ldots \times E_n \to E_k$$

Beispiel-Beziehung: *betreuen*



$$\begin{aligned}\text{betreuen} : \text{Professoren} \times \text{Studenten} &\rightarrow \text{Seminarthemen}\\ \text{betreuen} : \text{Seminarthemen} \times \text{Studenten} &\rightarrow \text{Professoren}\end{aligned}$$

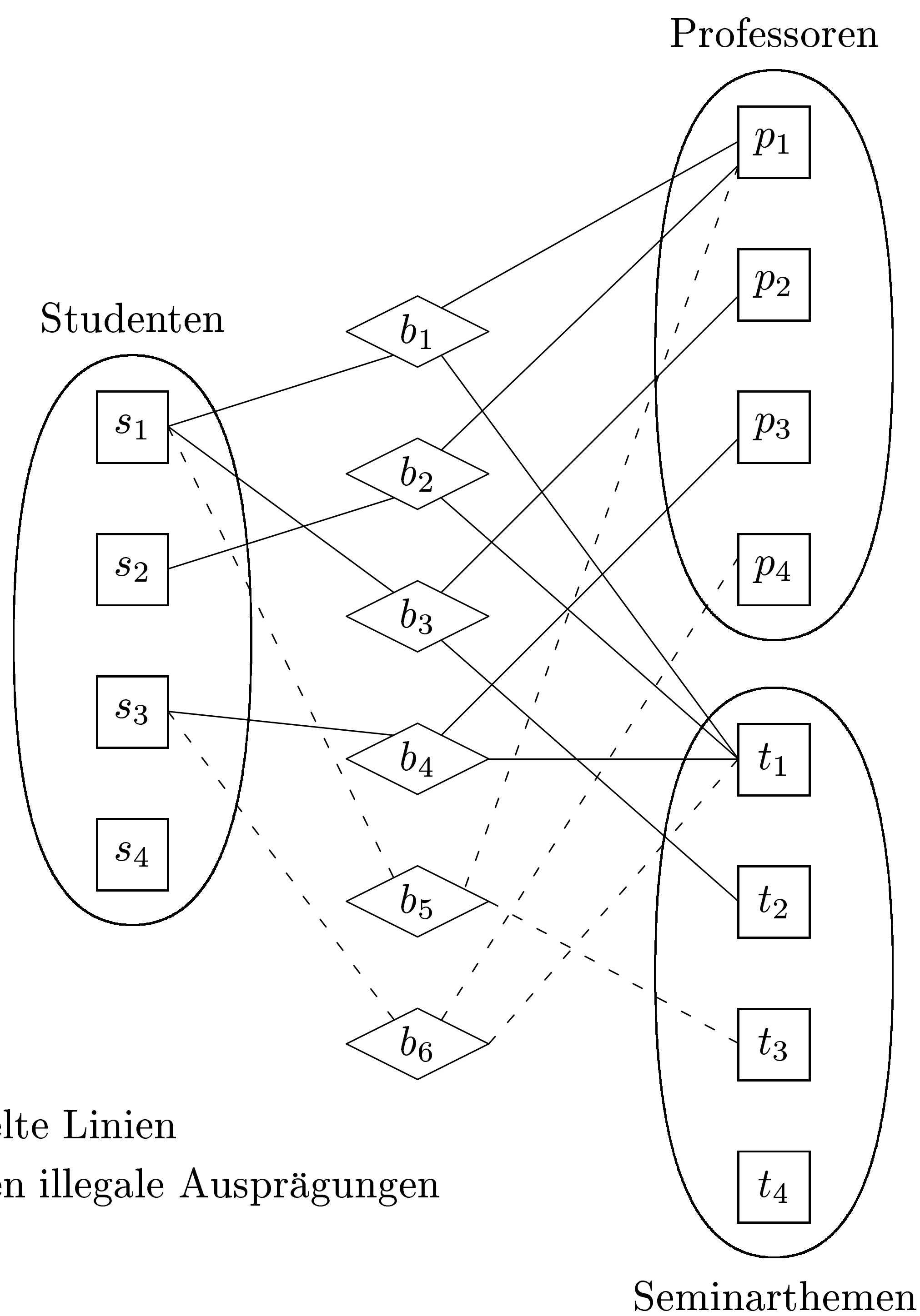
Dadurch erzwungene Konsistensbedingungen

1. Studenten dürfen bei demselben Professor bzw. derselben Professorin nur ein Seminarthema „ableisten“ (damit ein breites Spektrum abgedeckt wird).
2. Studenten dürfen dasselbe Seminarthema nur einmal bearbeiten – sie dürfen also nicht bei anderen Professoren ein schon einmal erteiltes Seminarthema nochmals bearbeiten.

Es sind aber folgende Datenbankzustände nach wie vor möglich:

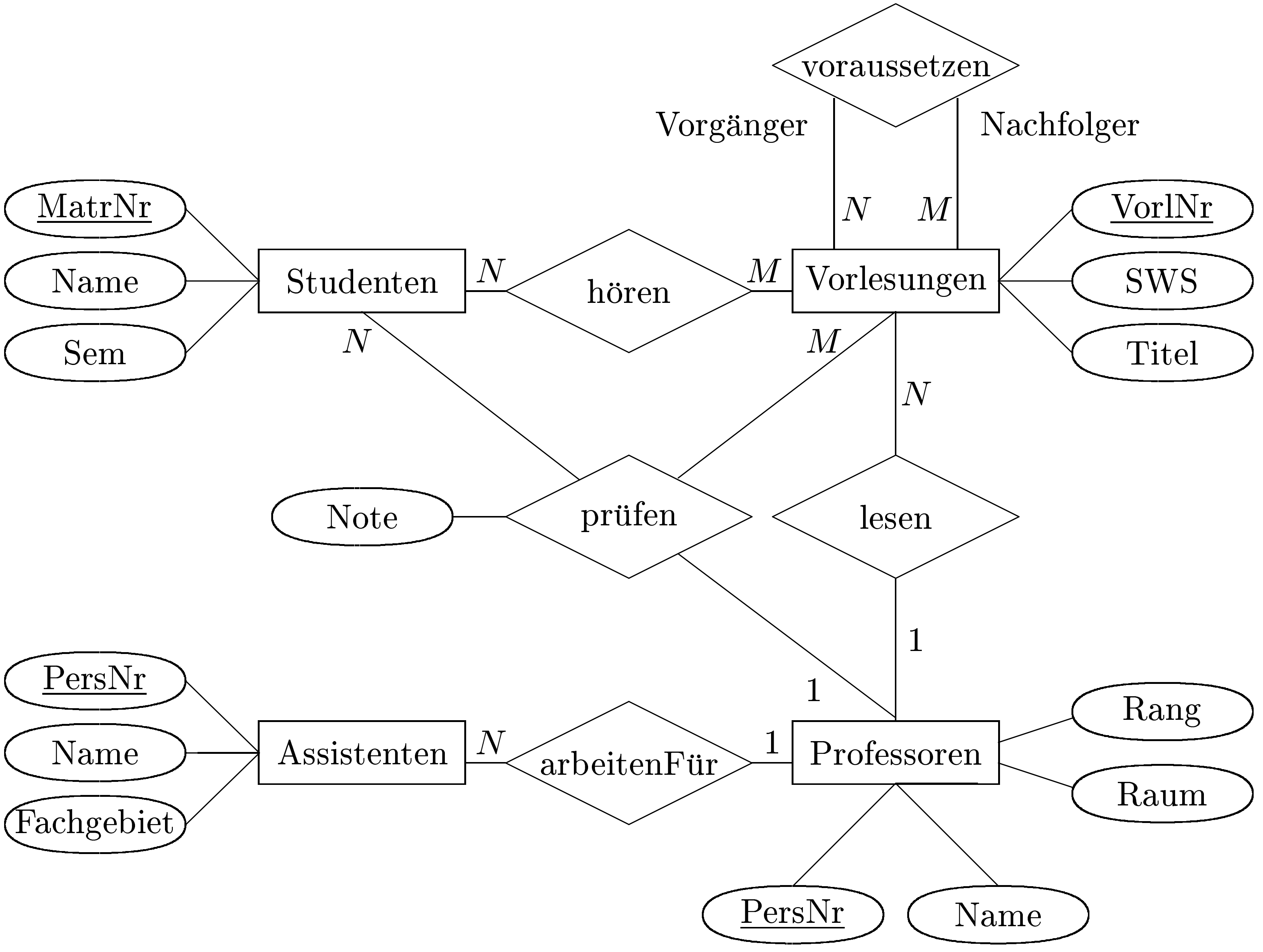
- Professoren können dasselbe Seminarthema „wiederverwenden“ – also dasselbe Thema auch mehreren Studenten erteilen.
- Ein Thema kann von mehreren Professoren vergeben werden – aber an unterschiedliche Studenten.

Ausprägungen der Beziehung *betreuen*

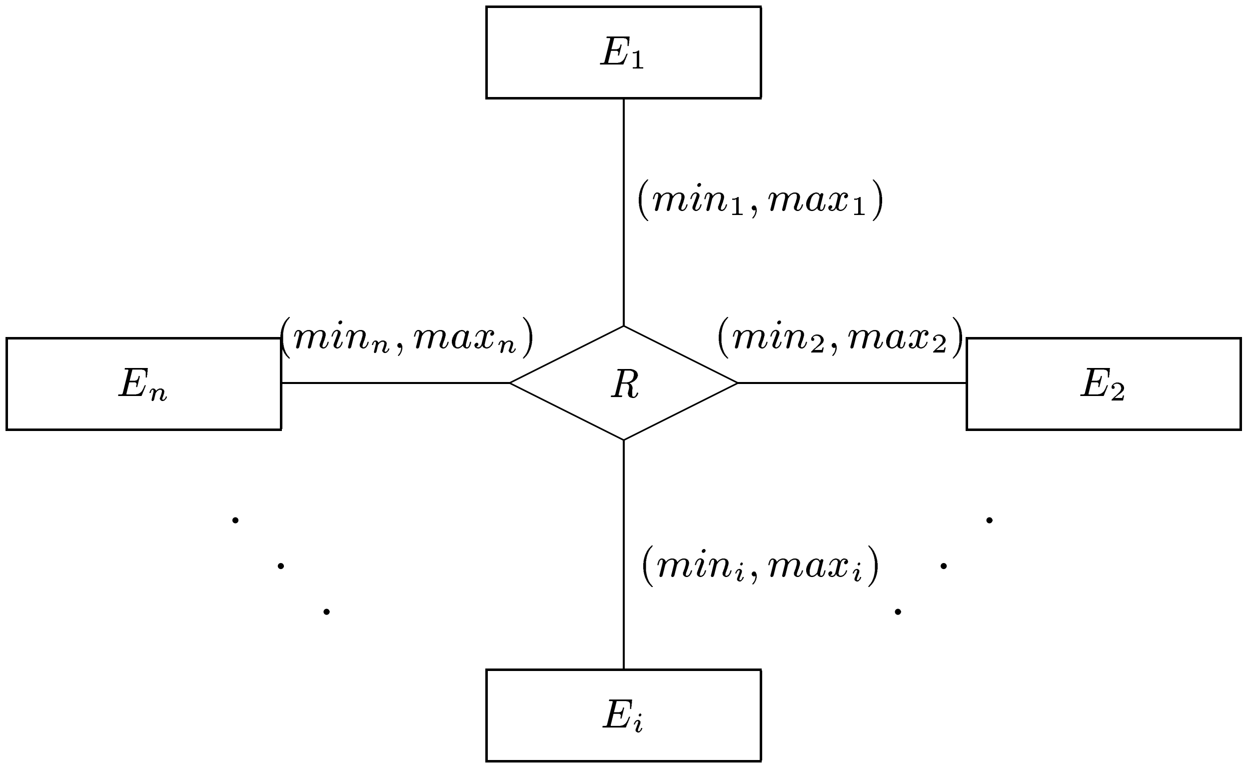


gestrichelte Linien
markieren illegale Ausprägungen

Markierung der Funktionalitäten



(min, max)-Notation

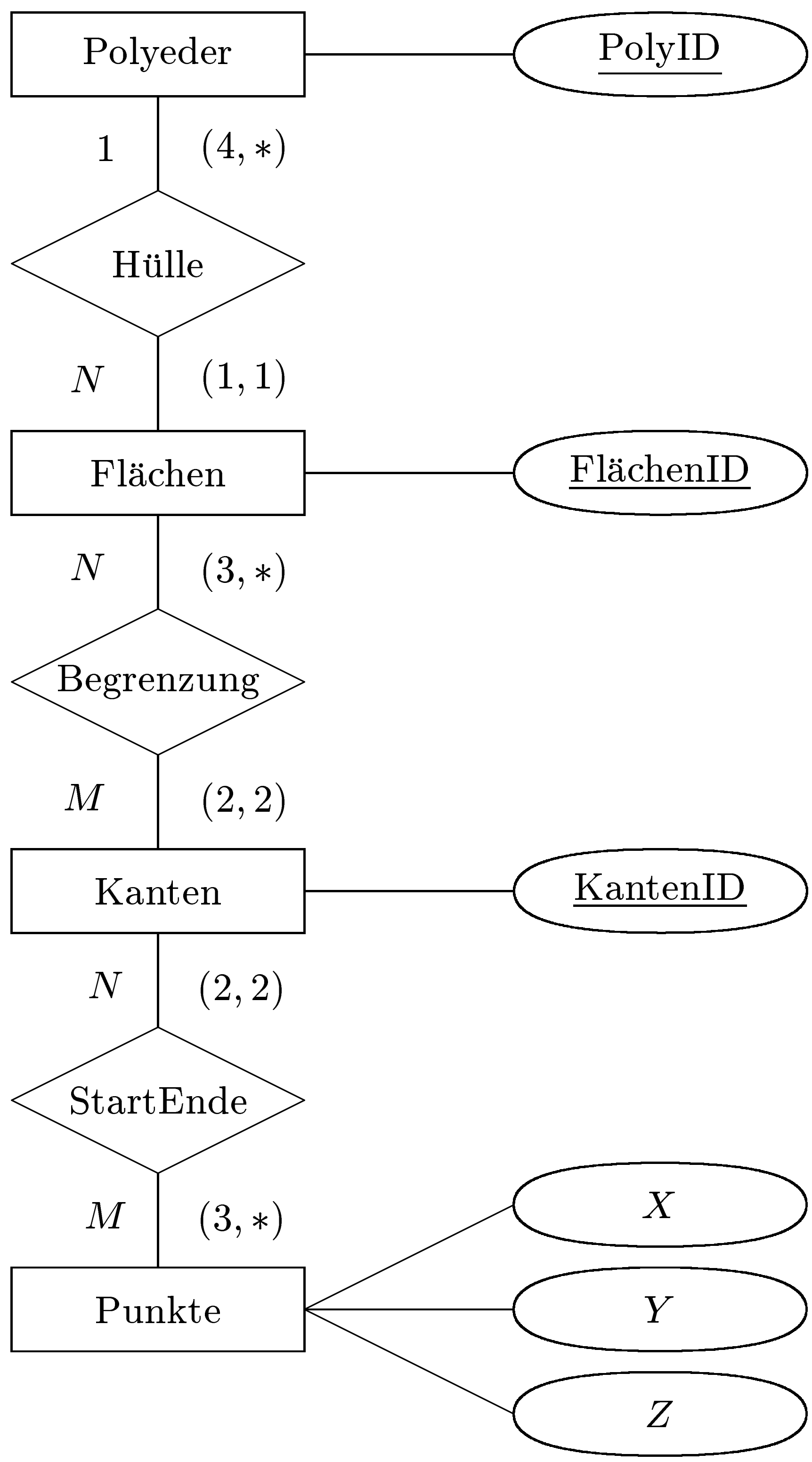


$$R \subseteq E_1 \times \cdots \times E_i \times \cdots \times E_n$$

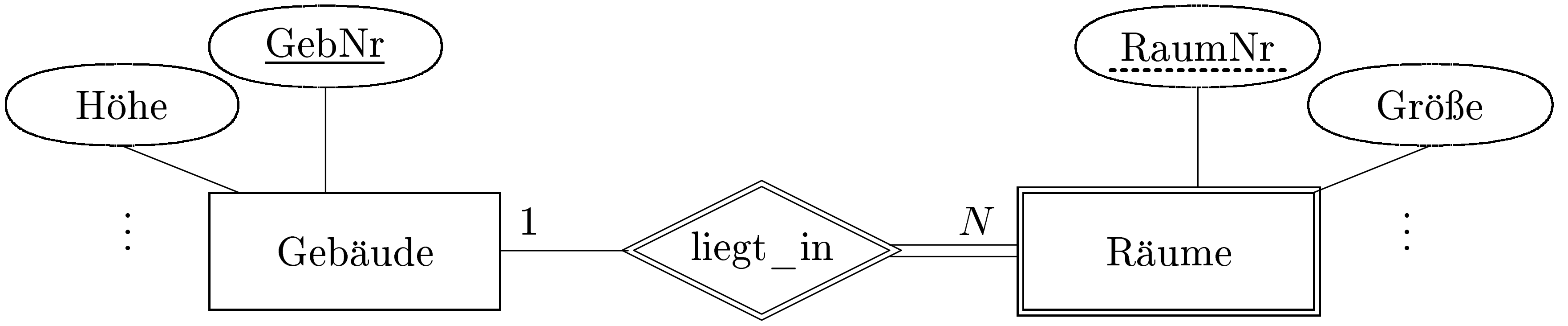
Für jedes $e_i \in E_i$ gibt es

- mindestens min_i Tupel der Art $(\ldots, e_i, \ldots)$ und
- höchstens max_i viele Tupel der Art $(\ldots, e_i, \ldots) \in R$

Begrenzungsflächendarstellung

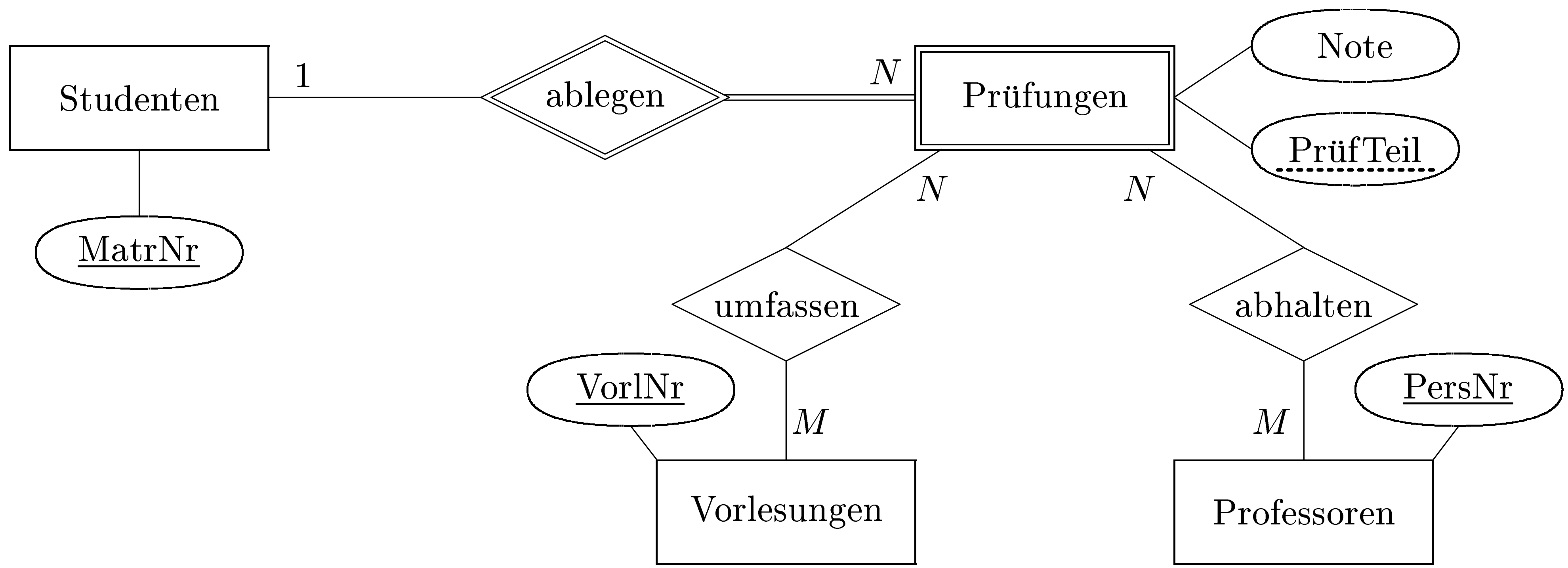


Schwache, existenzabhängige Entities



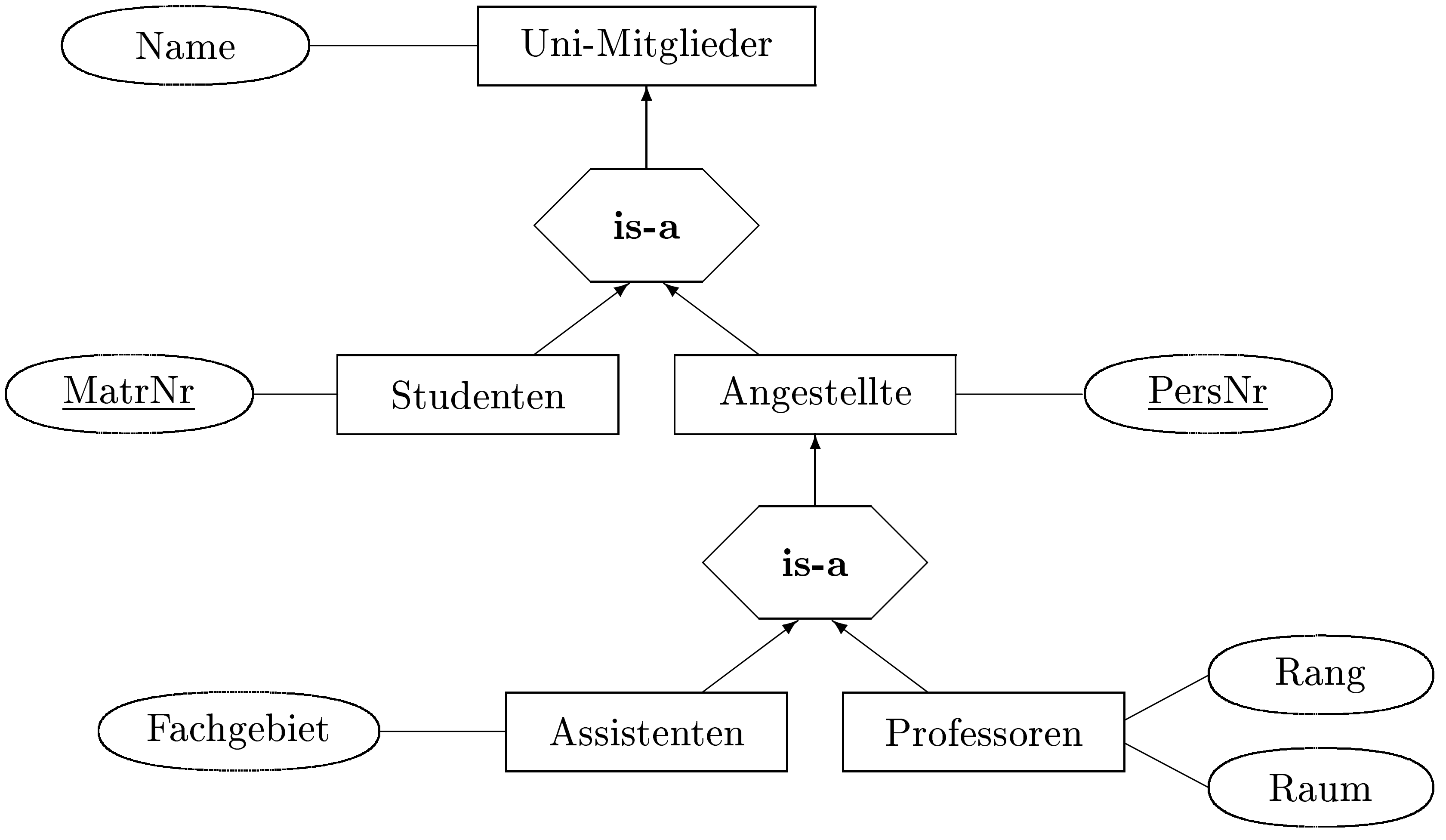
- Beziehung zwischen „starkem“ und schwachem Typ ist immer $1 : N$ (oder $1 : 1$ in seltenen Fällen)
- Warum kann das keine $N : M$-Beziehung sein?
- RaumNr ist nur innerhalb eines Gebäudes eindeutig
- Schlüssel ist: GebNr **und** RaumNr

Prüfungen als schacher Entitytyp

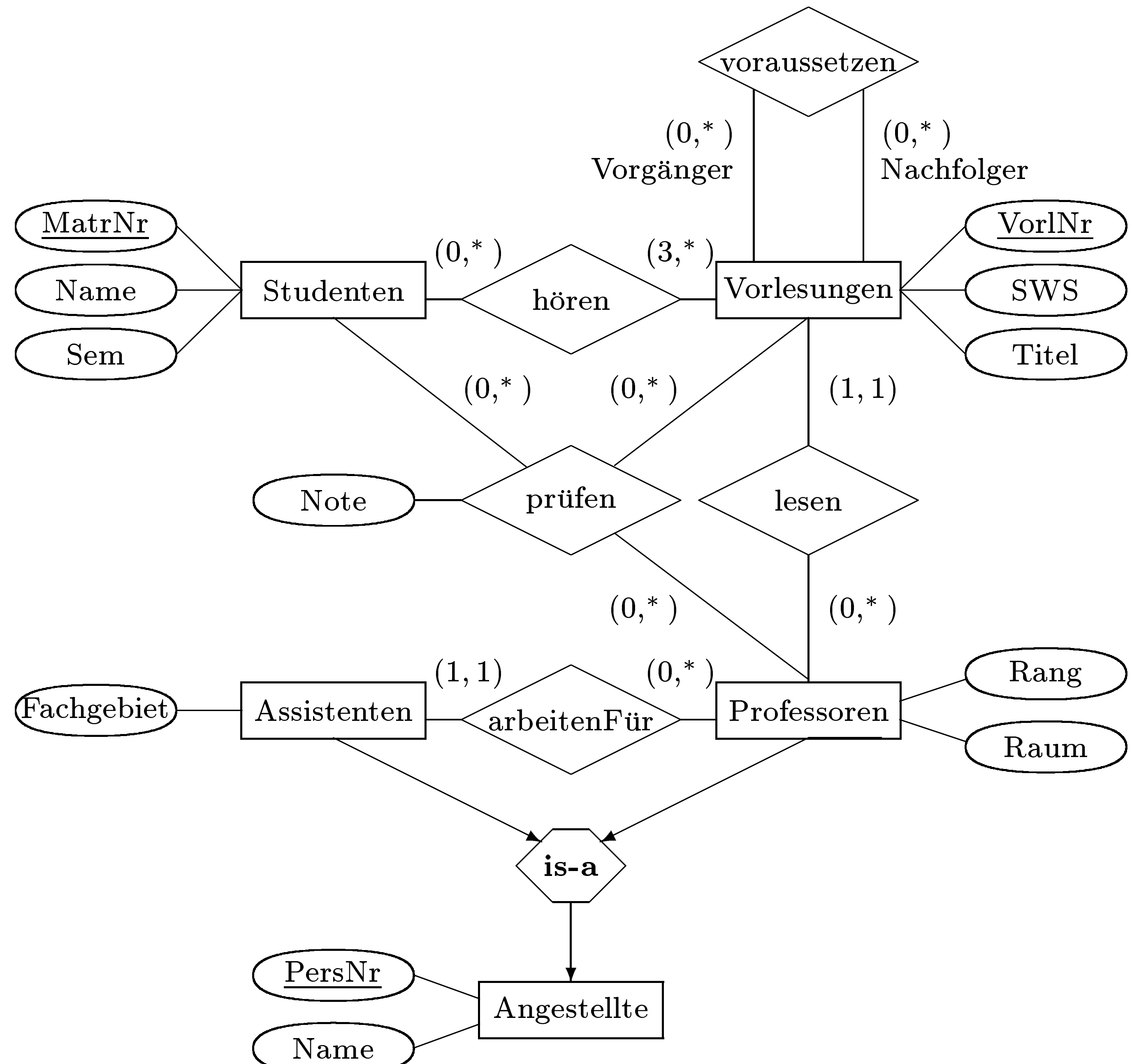


- mehrere Prüfer in einer Prüfung
- mehrere Vorlesungen werden in einer Prüfung abgefragt

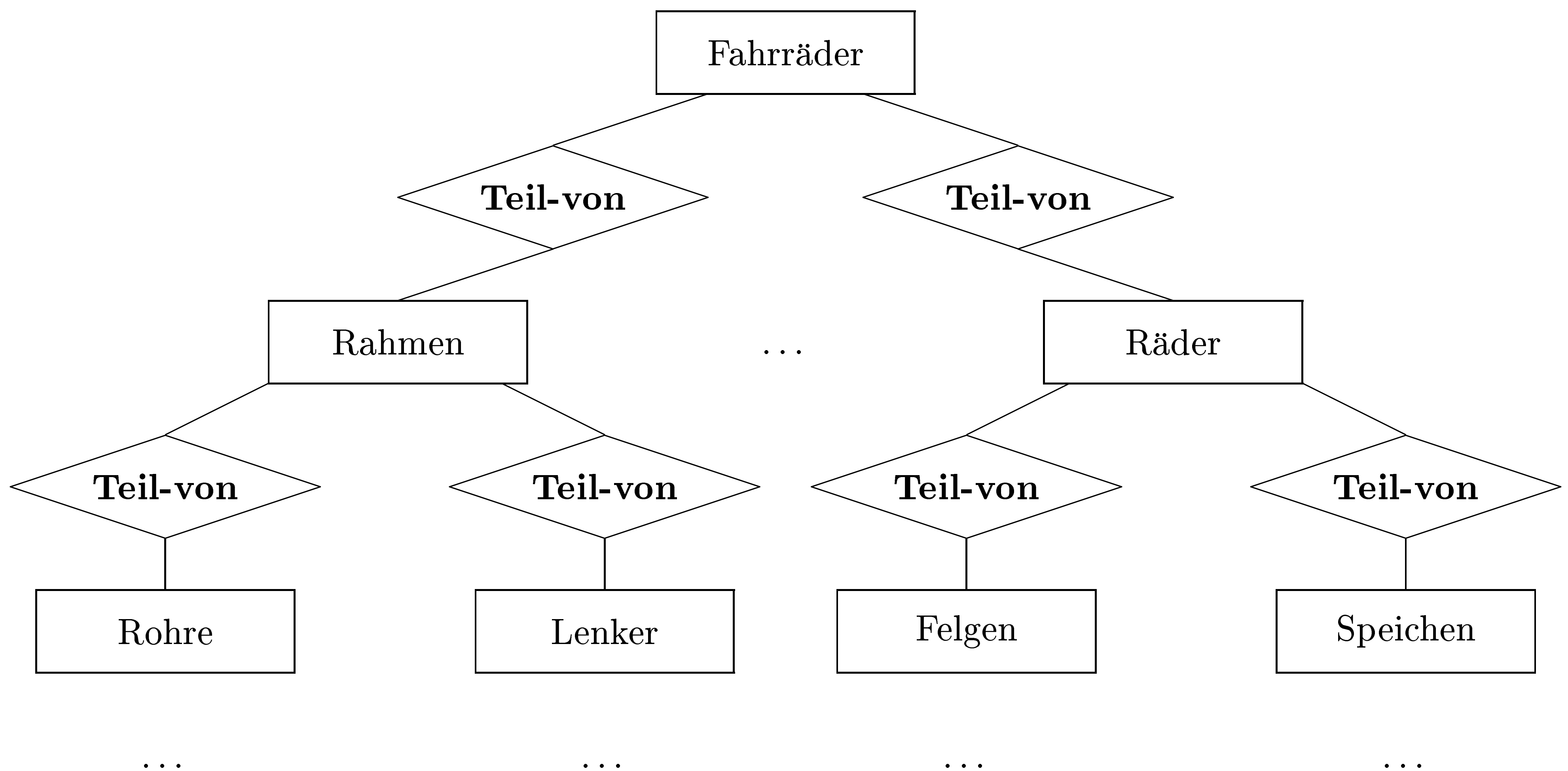
Generalisierung



Universitätsschema mit Generalisierung und (min,max)-Markierung

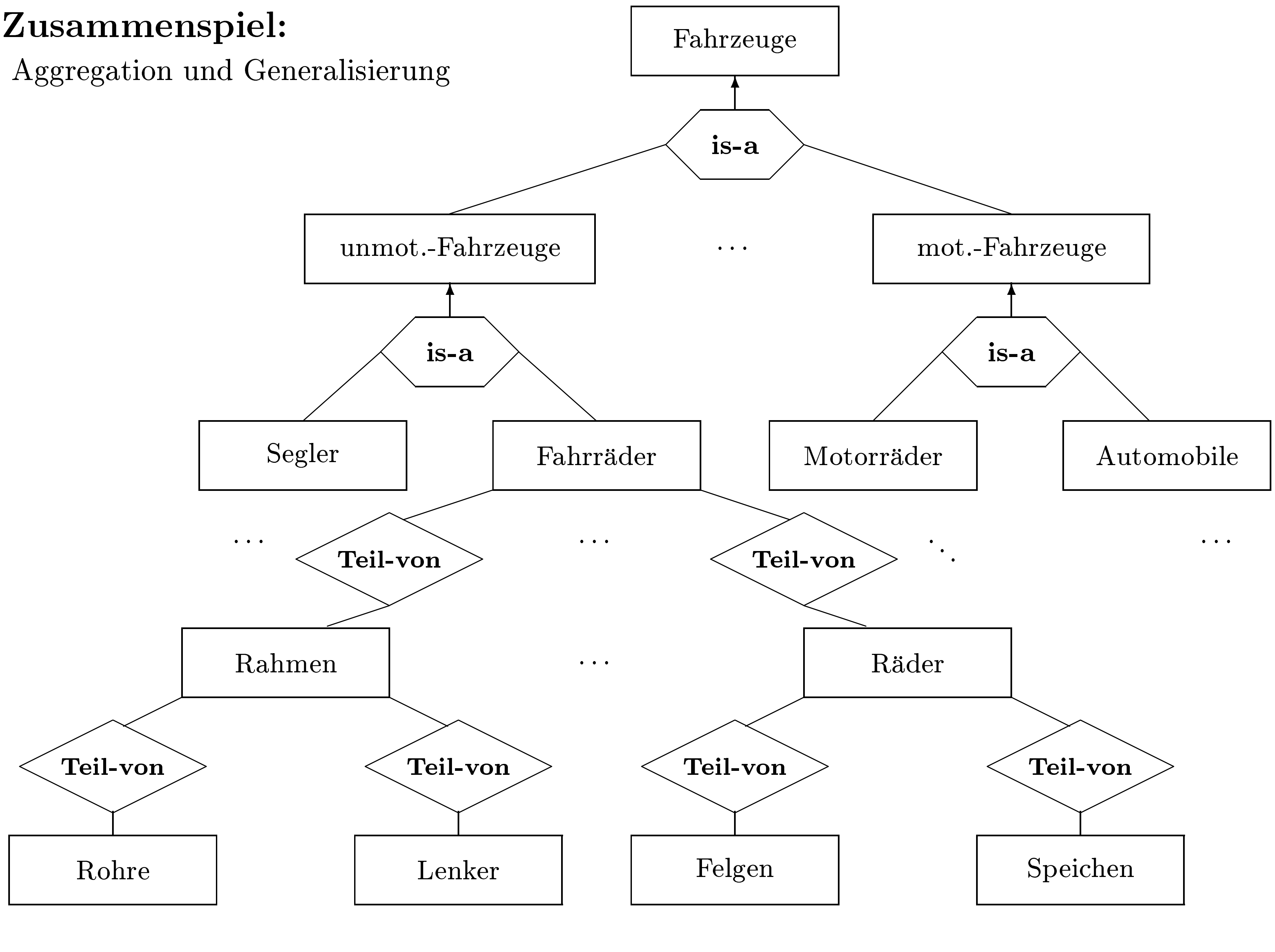


Aggregation

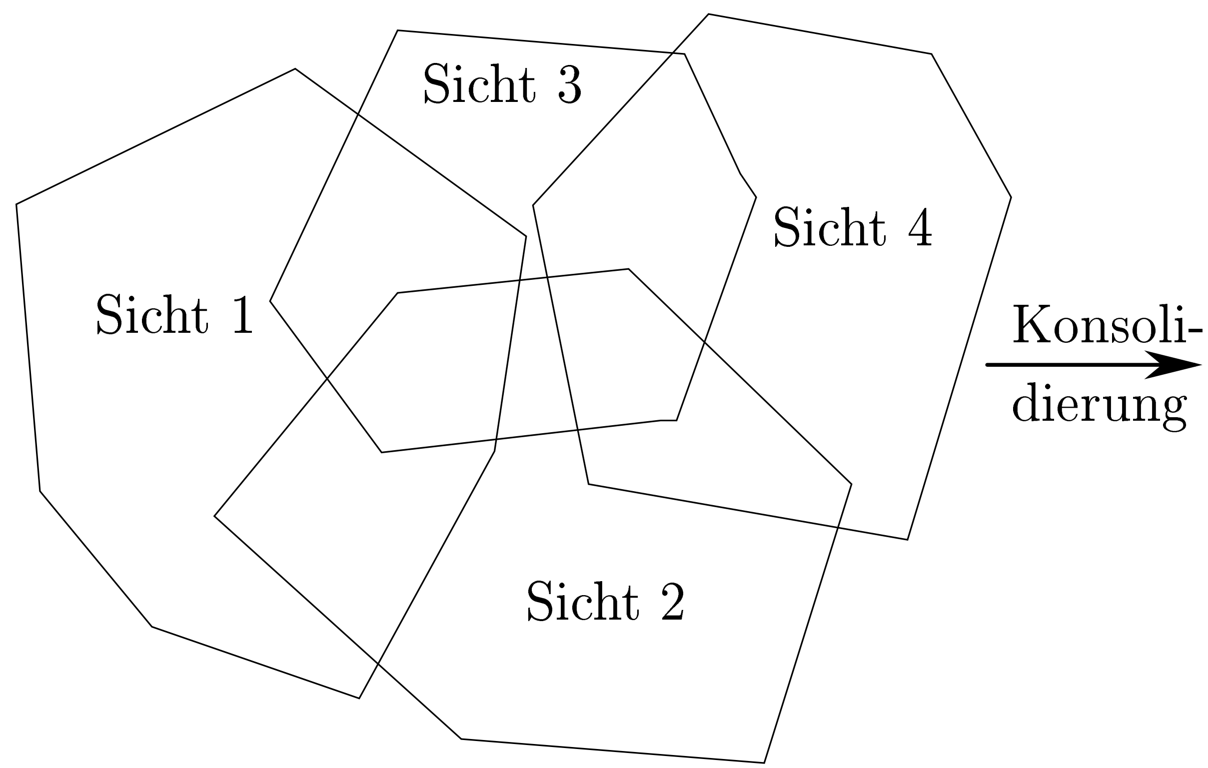


Zusammenspiel:

Aggregation und Generalisierung



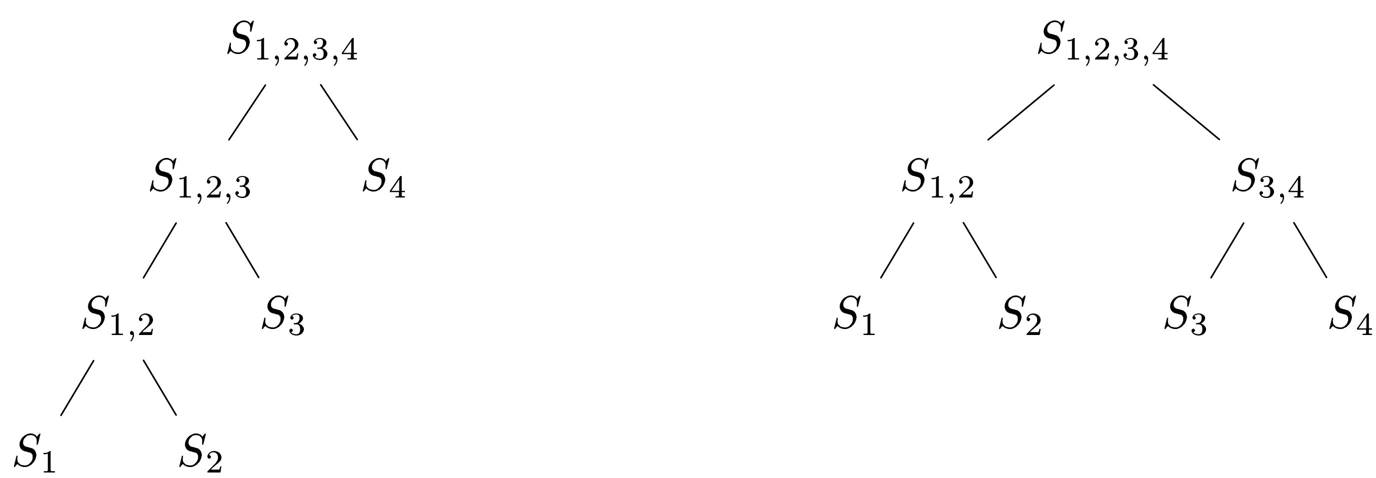
Konsolidierung von Teilschemata oder Sichtenintegration



globales Schema

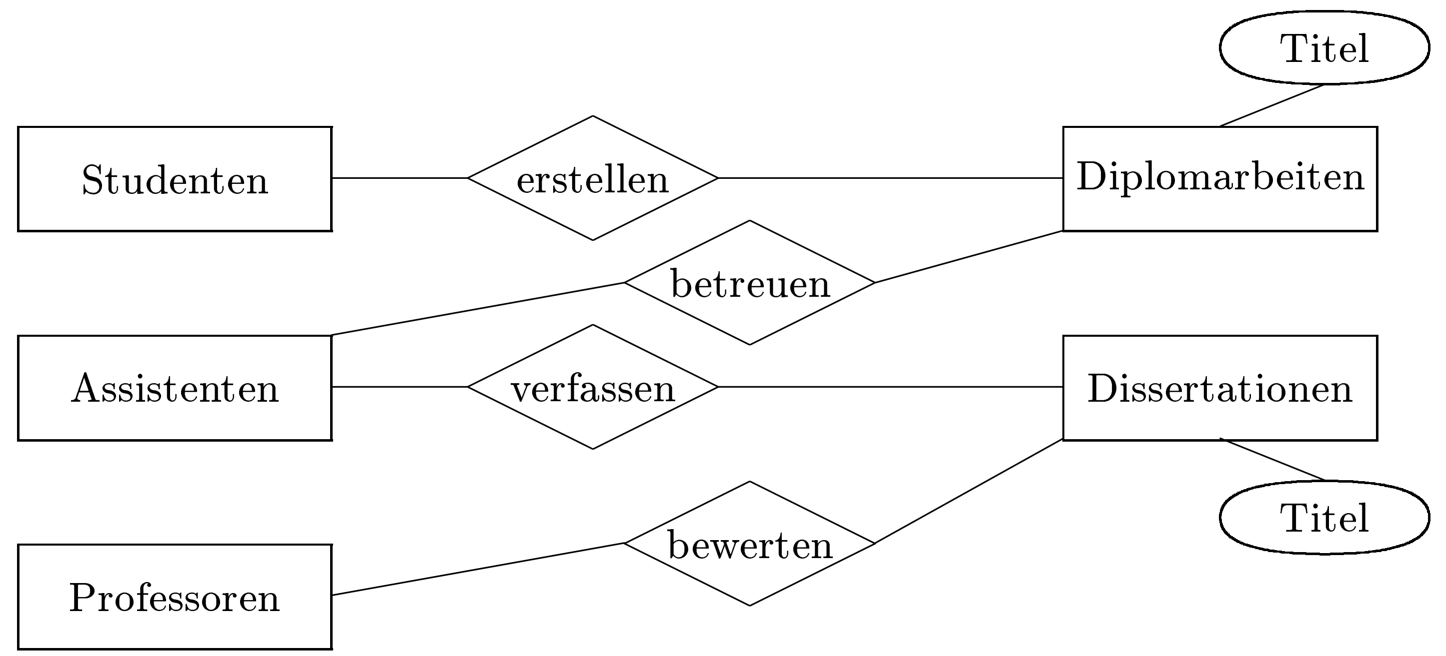
- redundanzfrei
- widerspruchsfrei
- Synonyme bereinigt
- Homonyme bereinigt

Mögliche Konsolidierungsbäume

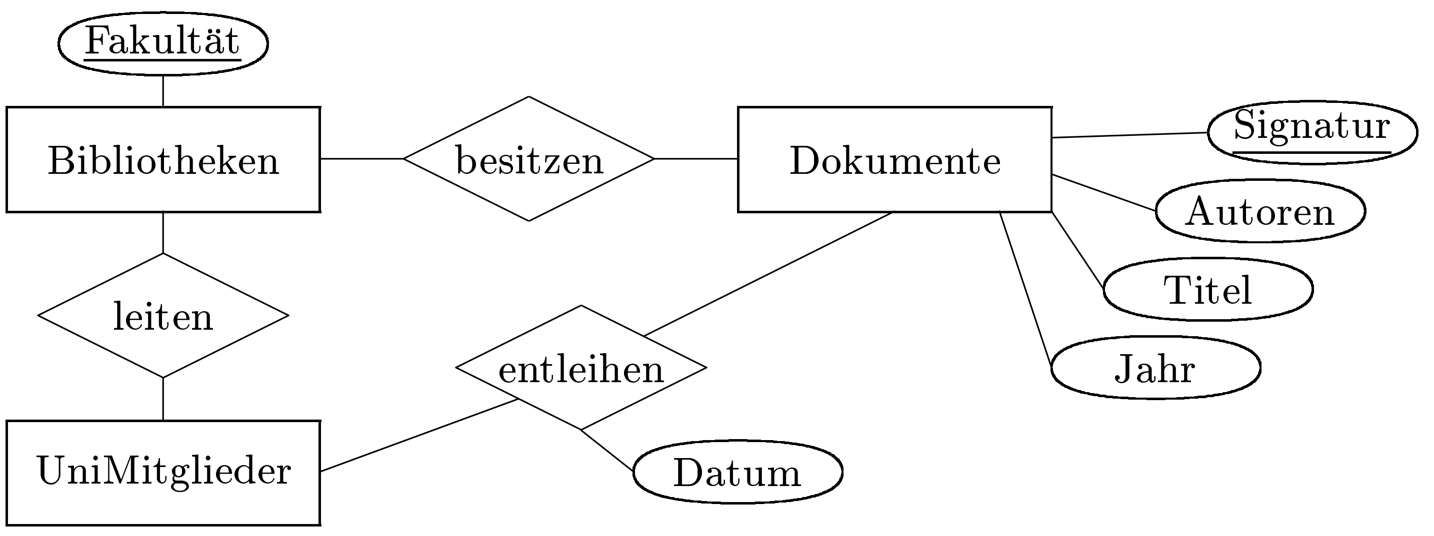


- Mögliche Konsolidierungsbäume zur Herleitung des globalen Schemas $S_{1,2,3,4}$ aus 4 Teilschemata S_1, S_2, S_3 und S_4
- links ein maximal hoher Konsolidierungsbaum
- rechts ein minimal hoher Konsolidierungsbaum

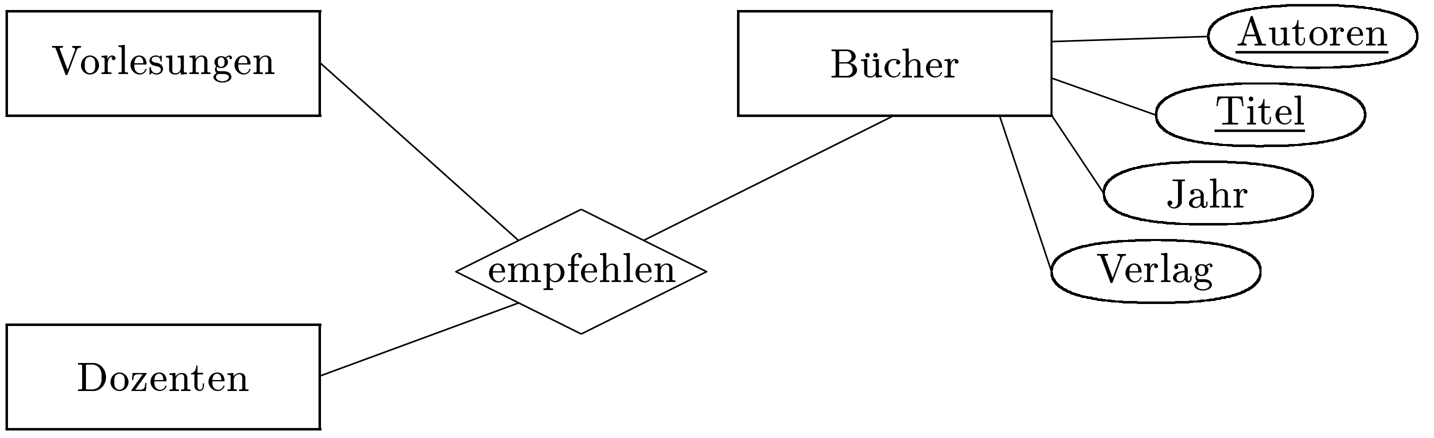
Drei Sichten einer Universitäts-Datenbank



Sicht 1: Erstellung von Dokumenten als Prüfungsleistung



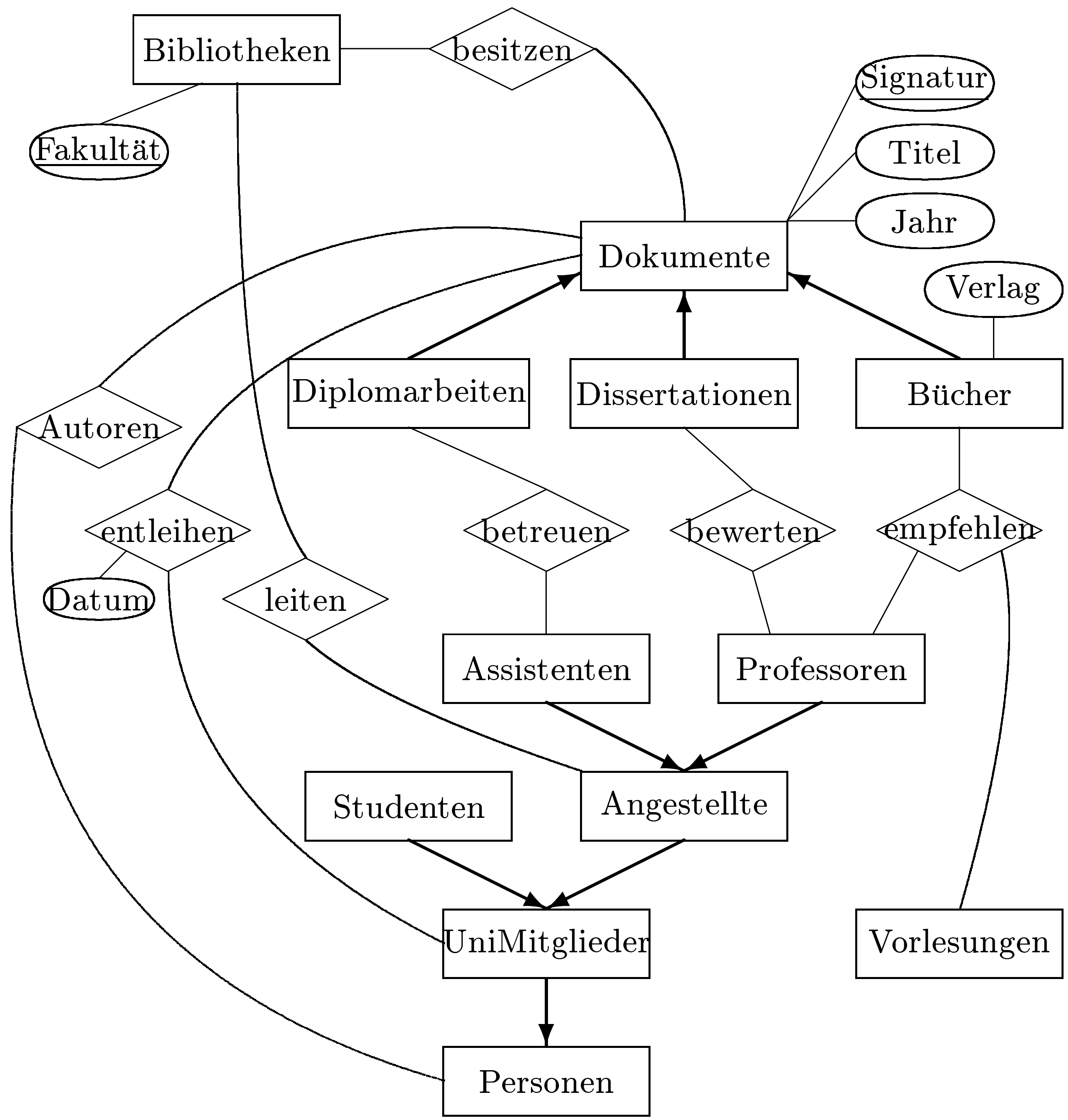
Sicht 2: Bibliotheksverwaltung



Sicht 3: Buchempfehlungen für Vorlesungen

Beobachtungen

- Die Begriffe *Dozenten* und *Professoren* sind synonym verwendet worden.
- Der Entitytyp *UniMitglieder* ist eine Generalisierung von *Studenten*, *Professoren* und *Assistenten*.
- Fakultätsbibliotheken werden sicherlich von *Angestellten* (und nicht von *Studenten*) geleitet. Insofern ist die in Sicht 2 festgelegte Beziehung *leiten* revisionsbedürftig, sobald wir im globalen Schema ohnehin eine Spezialisierung von *UniMitglieder* in *Studenten* und *Angestellte* vornehmen.
- *Dissertationen*, *Diplomarbeiten* und *Bücher* sind Spezialisierungen von *Dokumenten*, die in den *Bibliotheken* verwaltet werden.
- Wir können davon ausgehen, daß alle an der Universität erstellten *Diplomarbeiten* und *Dissertationen* in *Bibliotheken* verwaltet werden.
- Die in Sicht 1 festgelegten Beziehungen *erstellen* und *verfassen* modellieren denselben Sachverhalt wie das Attribut *Autoren* von *Büchern* in Sicht 3.
- Alle in einer Bibliothek verwalteten Dokumente werden durch die *Signatur* identifiziert.


Bibliotheken
besitzen
Signatur
Fakultät
Titel
Jahr
Dokumente
Verlag
Diplomarbeiten
Dissertationen
Bücher
Autoren
entleihen
betreuen
bewerten
empfehlen
Datum
leiten
Assistenten
Professoren
Studenten
Angestellte
UniMitglieder
Vorlesungen
Personen

Grundlagen des relationalen Modells

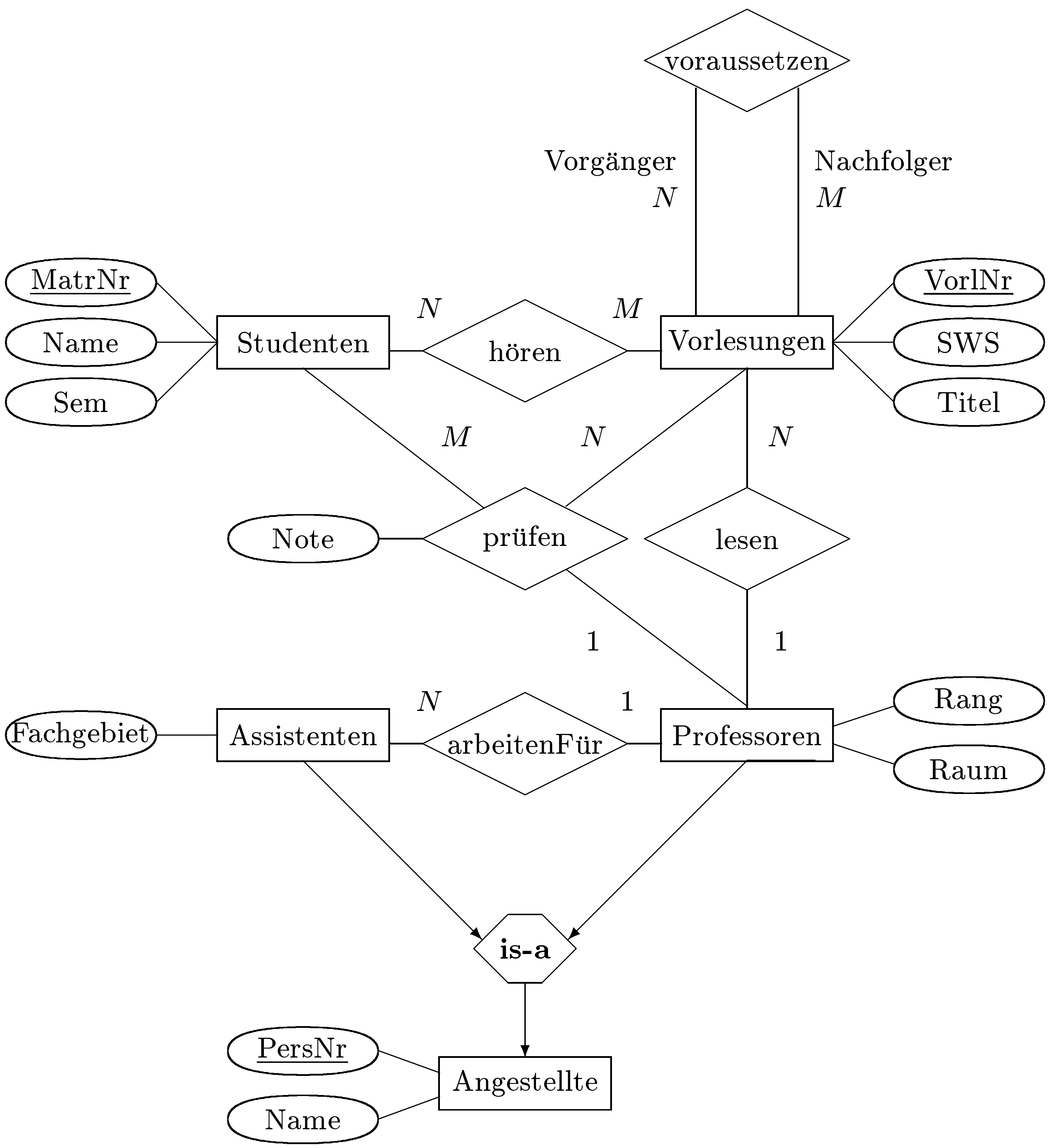
Seien $D_1, D_2, \ldots, D_n$ Domänen (Wertebereiche)

- *Relation*: $R \subseteq D_1 \times \ldots \times D_n$
- Bsp.: $\mathit{Telefonbuch} \subseteq \mathit{string} \times \mathit{string} \times \mathit{integer}$
- *Tupel*: $t \in R$
  Bsp.: $t =$ („Mickey Mouse", „Main Street", 4711)
- *Schema*: legt die Struktur der gespeicherten Daten fest
  Bsp.:
  $\mathit{Telefonbuch} : \{[\mathit{Name} : \mathit{string}, \mathit{Adresse} : \mathit{string}, \underline{\mathit{Telefon\#} : \mathit{integer}}]\}$

Telefonbuch		
Name	Straße	Telefon#
Mickey Mouse	Main Street	4711
Mini Mouse	Broadway	94725
Donald Duck	Highway	95672
...	...	...

- **Ausprägung:** der aktuelle Zustand der Datenbasis.
- **Schlüssel:** minimale Menge von Attributen, deren Werte ein Tupel eindeutig identifiziert.
- **Primärschlüssel** wird unterstrichen.

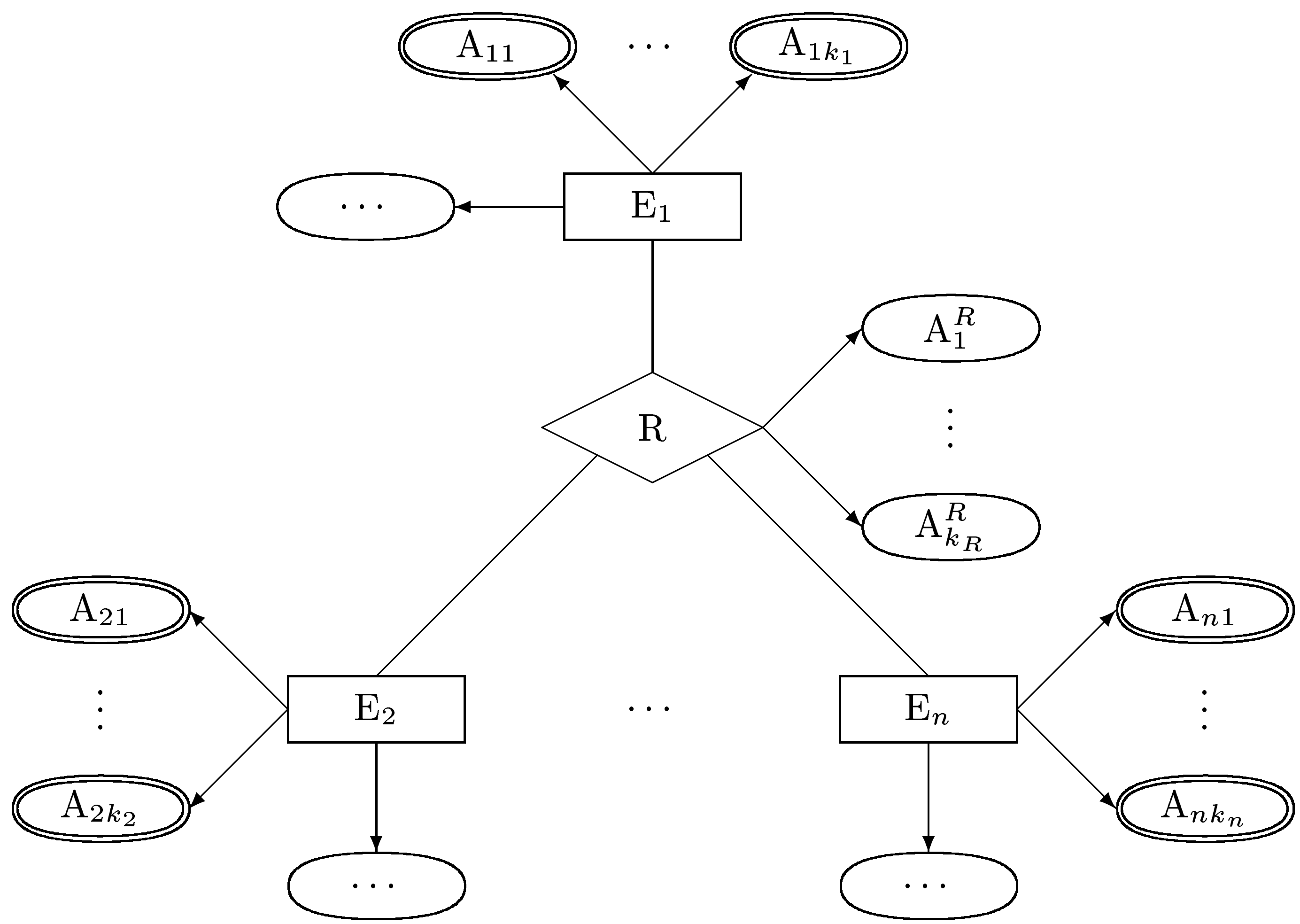
Das ER-Schema der Uni-Datenbank



Relationale Darstellung von Entitytypen

$$
\begin{array}{rcl}
\text{Studenten} & : & \{[\underline{\text{MatrNr : integer,}}\ Name : string, Semester : integer]\} \\
\text{Vorlesungen} & : & \{[\underline{\text{VorlNr : integer,}}\ Titel : string, SWS : integer]\} \\
\text{Professoren} & : & \{[\underline{\text{PersNr : integer,}}\ Name : string, Rang : string, Raum : integer]\} \\
\text{Assistenten} & : & \{[\underline{\text{PersNr : integer,}}\ Name : string, Fachgebiet : string]\}
\end{array}
$$

Relationale Darstellung von Beziehungen



$$R : \{[\underbrace{A_{11}, \ldots, A_{1k_1}}_{Schlüssel\ von\ E_1}, \underbrace{A_{21}, \ldots, A_{2k_2}}_{Schlüssel\ von\ E_2}, \ldots, \underbrace{A_{n1}, \ldots, A_{nk_n}}_{Schlüssel\ von\ E_n}, \underbrace{A_1^R, \ldots, A_{k_R}^R}_{Attribute\ von\ R}]\}$$

Beziehungen unseres Beispiel-Schemas

$$
\begin{array}{rcl}
\text{hören} & : & \{[\underline{\text{MatrNr} : \text{integer}, \text{VorlNr} : \text{integer}}]\} \\
\text{lesen} & : & \{[\text{PersNr} : \text{integer}, \underline{\text{VorlNr} : \text{integer}}]\} \\
\text{arbeitenFür} & : & \{[\underline{\text{AssistentPersNr} : \text{integer}}, ProfPersNr : integer]\} \\
\text{voraussetzen} & : & \{[\underline{\text{Vorgänger} : \text{integer}, \text{Nachfolger} : \text{integer}}]\} \\
\text{prüfen} & : & \{[\underline{\text{MatrNr} : \text{integer}, \text{VorlNr} : \text{integer}}, \text{PersNr} : \text{integer}, \\
 & & \text{Note} : \text{decimal}]\}
\end{array}
$$

Ausprägung der Beziehung *hören*

Studenten	
MatrNr	...
26120	...
27550	...
...	...

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Vorlesungen	
VorlNr	...
5001	...
4052	...
...	...

Verfeinerung des relationalen Schemas

1:N-Beziehungen

- Initial-Entwurf

$$\begin{aligned} \mathit{Vorlesungen} &: \{[\underline{\mathit{VorlNr}}, \mathit{Titel}, \mathit{SWS}]\} \\ \mathit{Professoren} &: \{[\underline{\mathit{PersNr}}, \mathit{Name}, \mathit{Rang}, \mathit{Raum}]\} \\ \mathit{lesen} &: \{[\underline{\mathit{VorlNr}}, \mathit{PersNr}]\} \end{aligned}$$

- Verfeinerung durch Zusammenfassung

$$\begin{aligned} \mathit{Vorlesungen} &: \{[\underline{\mathit{VorlNr}}, \mathit{Titel}, \mathit{SWS}, \mathit{gelesenVon}]\} \\ \mathit{Professoren} &: \{[\underline{\mathit{PersNr}}, \mathit{Name}, \mathit{Rang}, \mathit{Raum}]\} \end{aligned}$$

Regel

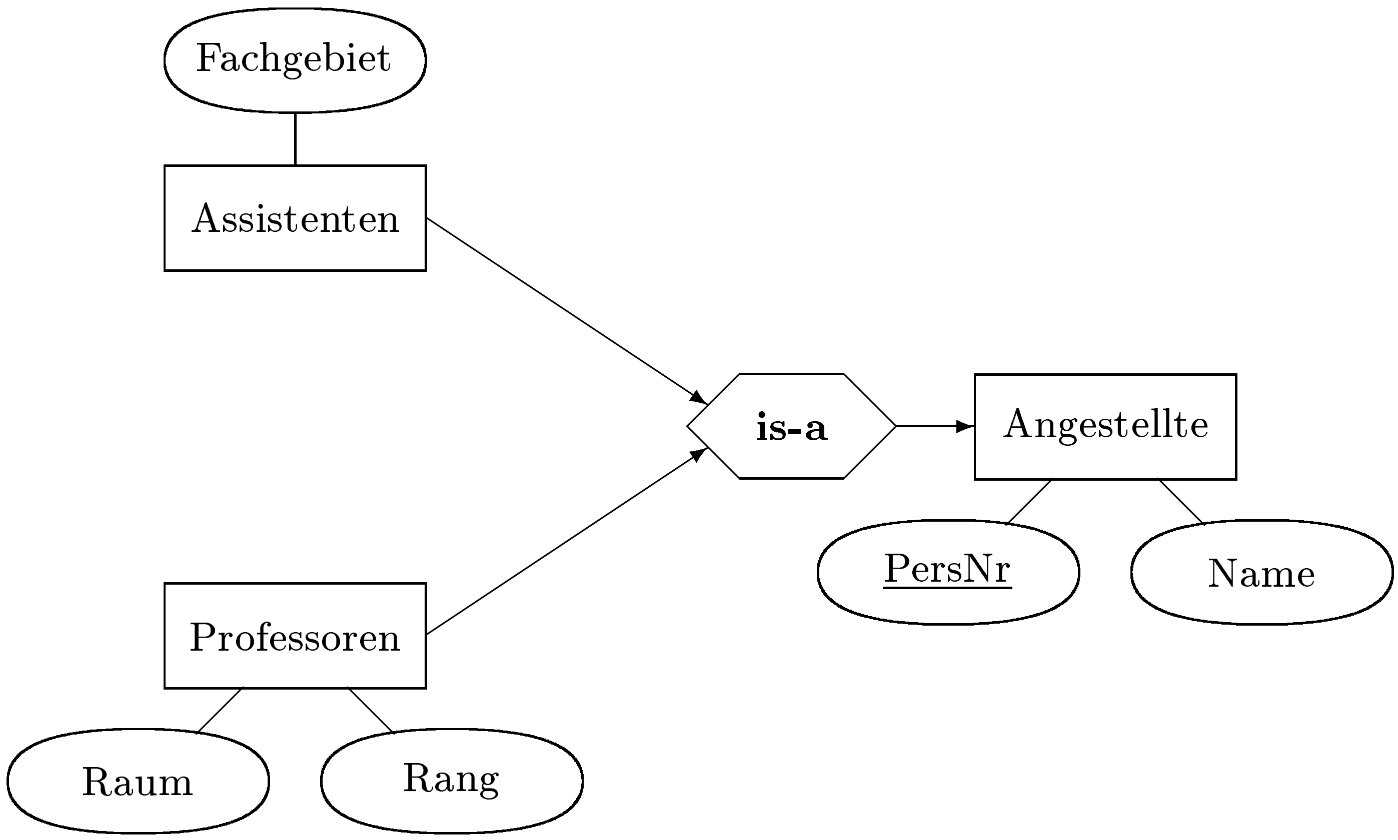
Relationen mit gleichem Schlüssel kann man zusammenfassen – **aber nur diese und keine anderen!**

Ausprägung von *Professoren* und *Vorlesungen*

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

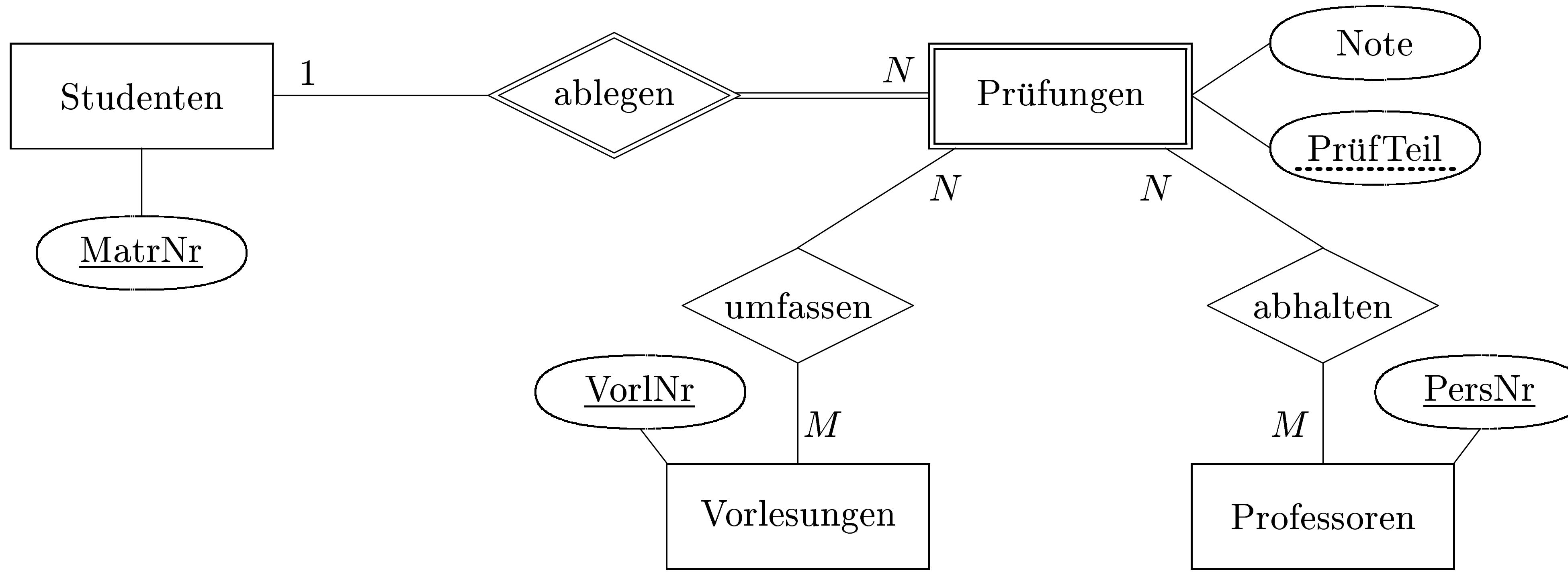
Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

Relationale Modellierung der Generalisierung



$$
\begin{aligned}
\mathit{Angestellte} &: \{[\underline{\mathit{PersNr}}, \mathit{Name}]\} \\
\mathit{Professoren} &: \{[\underline{\mathit{PersNr}}, \mathit{Rang}, \mathit{Raum}]\} \\
\mathit{Assistenten} &: \{[\underline{\mathit{PersNr}}, \mathit{Fachgebiet}]\}
\end{aligned}
$$

Relationale Modellierung schwacher Entitytypen



$$
\begin{aligned}
\text{Prüfungen} &: \{[\underline{\text{MatrNr} : \text{integer}, \text{PrüfTeil} : \text{string}}, \text{Note} : \text{integer}]\}\\
\text{umfassen} &: \{[\underline{\text{MatrNr} : \text{integer}, \text{PrüfTeil} : \text{string}, \text{VorlNr} : \text{integer}}]\}\\
\text{abhalten} &: \{[\underline{\text{MatrNr} : \text{integer}, \text{PrüfTeil} : \text{string}, \text{PersNr} : \text{integer}}]\}
\end{aligned}
$$

Man beachte, daß in diesem Fall der (global eindeutige) Schlüssel der Relation *Prüfungen*, nämlich *MatrNr* **und** *PrüfTeil* als Fremdschlüssel in die Relationen *umfassen* und *abhalten* übernommen werden muß.

Die relationale Uni-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Assistenten			
PersNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Die relationale Algebra

Operatoren der Relationenalgebra

- σ Selektion
- π Projektion
- $\times$ Kreuzprodukt
- $\bowtie$ Join (Verbund)
- ρ Umbenennung (rename)
- $-$ Mengendifferenz
- $\div$ Division
- $\cup$ Vereinigung
- $\cap$ Mengendurchschnitt
- $\rtimes$ Semi-Join (linker)
- $\ltimes$ Semi-Join (rechter)
- ⟗ outer (äußerer) Join
- ⟕ linker äußerer Join
- ⟖ rechter äußerer Join

Selektion

$$\sigma_{\text{Semester}>10}(Studenten)$$

$\sigma_{\text{Semester}>10}(Studenten)$		
MatrNr	*Name*	*Semester*
24002	Xenokrates	18
25403	Jonas	12

Projektion

$$\Pi_{\text{Rang}}(Professoren)$$

$\Pi_{\text{Rang}}(Professoren)$
Rang
C4
C3

Kartesisches Produkt

$$Professoren \times hören$$

Professoren				*hören*	
PersNr	*Name*	*Rang*	*Raum*	*MatrNr*	*VorlNr*
2125	Sokrates	C4	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	C4	226	29555	5001
...	...	...	...	...	...
2137	Kant	C4	7	29555	5001

- Problem: riesige Zwischenergebnisse
- Beispiel: (Professoren × hören) × Studenten
- "bessere" Operation: Join (siehe unten)

Umbenennung

- Umbenennung von Relationen
- Beispiel: Ermittlung indirekter Vorgänger 2. Stufe der Vorlesung 5216

$$\Pi_{\text{V1.Vorgänger}}(\sigma_{\text{V2.Nachfolger=5216} \wedge \text{V1.Nachfolger=V2.Vorgänger}}$$
$$(\rho_{\text{V1}}(voraussetzen) \times \rho_{\text{V2}}(voraussetzen)))$$

- Umbennenung von Attributen

$$\rho_{\text{Voraussetzung} \leftarrow \text{Vorgänger}}(\text{voraussetzen})$$

Formale Definition der Algebra

Basisausdrücke:

- Relationen der Datenbank oder
- konstante Relationen.

Operationen:

- Selektion: $\sigma_p(E_1)$
- Projektion: $\Pi_S(E_1)$
- Kartesisches Produkt: $E_1 \times E_2$
- Umbenennung: $\rho_V(E_1)$, $\rho_{A \leftarrow B}(E_1)$
- Vereinigung: $E_1 \cup E_2$
- Differenz: $E_1 - E_2$

Der natürliche Verbund (Join)

Gegeben seien:

- $R(A_1, \ldots, A_m, B_1, \ldots, B_k)$
- $S(B_1, \ldots, B_k, C_1, \ldots, C_n)$

$$R \bowtie S = \Pi_{A_1,\ldots,A_m,R.B_1,\ldots,R.B_k,C_1,\ldots,C_n}(\sigma_{R.B_1=S.B_1 \wedge \ldots \wedge R.B_k=S.B_k}(R \times S))$$

$R \bowtie S$											
$\mathcal{R} - \mathcal{S}$				$\mathcal{R} \cap \mathcal{S}$				$\mathcal{S} - \mathcal{R}$			
A_1	A_2	$\ldots$	A_m	B_1	B_2	$\ldots$	B_k	C_1	C_2	$\ldots$	C_n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Drei-Wege-Join

$$(Studenten \bowtie hören) \bowtie Vorlesungen$$

$(Studenten \bowtie hören) \bowtie \ Vorlesungen$						
MatrNr	*Name*	*Semester*	*VorlNr*	*Titel*	*SWS*	*gelesenVon*
26120	Fichte	10	5001	Grundzüge	4	2137
27550	Jonas	12	5022	Galube und Wissen	2	2134
28106	Carnap	3	4052	Wissenschaftstheorie	3	2126
...	...	...	...	...	...	...

Allgemeiner Join (Theta-Join)

Gegeben R(A_1, …, A_n) und S(B_1, …, B_m)

$$R \bowtie_\theta S$$

$R \bowtie_\theta S$							
R				S			
A_1	A_2	$\ldots$	A_n	B_1	B_2	$\ldots$	B_m
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Andere Join-Arten

- **natürlicher Join**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

- **linker äußerer Join**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$⟕$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	–	–

- **rechter äußerer Join**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$⟖$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
–	–	c_3	d_2	e_2

Andere Join-Arten

- **äußerer Join**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

⟗

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	–	–
–	–	c_3	d_2	e_2

- **Semi-Join von L mit R**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat		
A	B	C
a_1	b_1	c_1

- **Semi-Join von R mit L**

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Resultat		
C	D	E
c_1	d_1	e_1

Die relationale Division

Bsp.: Finde MatrNr der Studenten, die alle vierstündigen Vorlesungen hören

$$L := \Pi_{\text{VorlNr}}(\sigma_{\text{SWS}=4}(\text{Vorlesungen}))$$

$$\text{hören} \div \overbrace{\Pi_{\text{VorlNr}}(\sigma_{\text{SWS}=4}(\text{Vorlesungen}))}^{L}$$

Definition der Division: $t \in R \div S$, falls für jedes $t_s \in S$ ein $t_r \in R$ existiert, so daß gilt:

$$\begin{aligned} t_r.\mathcal{S} &= t_s.\mathcal{S} \\ t_r.(\mathcal{R} - \mathcal{S}) &= t.(\mathcal{R} - \mathcal{S}) \end{aligned}$$

$$\begin{array}{|c|c|} \hline \multicolumn{2}{|c|}{H} \\ \hline M & V \\ \hline m_1 & v_1 \\ m_1 & v_2 \\ m_1 & v_3 \\ m_2 & v_2 \\ m_2 & v_3 \\ \hline \end{array} \div \begin{array}{|c|} \hline L \\ \hline V \\ \hline v_1 \\ v_2 \\ \hline \end{array} = \begin{array}{|c|} \hline H \div L \\ \hline M \\ \hline m_1 \\ \hline \end{array}$$

Die Division $R \div S$ kann auch durch Differenz, Kreuzprodukt und Projektion ausgedrückt werden.

$$R \div S = \Pi_{(\mathcal{R}-\mathcal{S})}(R) - \Pi_{(\mathcal{R}-\mathcal{S})}((\Pi_{(\mathcal{R}-\mathcal{S})}(R) \times S) - R)$$

Mengendurchschnitt

Als Beispielanwendung für den Mengendurchschnitt (Operatorsymbol $\cap$) betrachten wir folgende Anfrage: Finde die *PersNr* aller C4-Professoren, die mindestens eine Vorlesung halten.

$$\Pi_{\text{PersNr}}(\rho_{\text{PersNr}\leftarrow\text{gelesenVon}}(\text{Vorlesungen}))\cap\Pi_{\text{PersNr}}(\sigma_{\text{Rang=C4}}(\text{Professoren}))$$

- Mengendurchschnitt nur auf zwei Argumentrelationen mit gleichem Schema anwendbar
- Deshalb ist die Umbenennung des Attributs *gelesenVon* in *PersNr* in der Relation *Vorlesungen* notwendig
- Der Mengendurchschnitt zweier Relationen $R \cap S$ kann durch die Mengendifferenz wie folgt ausgedrückt werden:

$$R \cap S = R - (R - S)$$

Der Relationenkalkül

Eine Anfrage im Relationenkalkül hat die Form

$$\{t \mid P(t)\}$$

mit $P(t)$ Formel.

Beispiele:

- C4-Professoren

$$\{p \mid p \in \mathit{Professoren} \wedge p.\mathit{Rang} = \text{`}C4'\}$$

- Studenten mit mindestens einer Vorlesung von Curie

$$\begin{aligned}\{&\mathrm{s} \mid \mathrm{s} \in \mathrm{Studenten}\\ &\quad \wedge \exists \mathrm{h} \in \mathrm{h\ddot{o}ren}(\mathrm{s.MatrNr}=\mathrm{h.MatrNr}\\ &\quad \wedge \exists \mathrm{v} \in \mathrm{Vorlesungen}(\mathrm{h.VorlNr}=\mathrm{v.VorlNr}\\ &\quad \wedge \exists \mathrm{p} \in \mathrm{Professoren}(\mathrm{p.PersNr}=\mathrm{v.gelesenVon}\\ &\quad \wedge \mathrm{p.Name} = \text{`Curie'})))\}\end{aligned}$$

- Wer hat **alle** vierstündigen Vorlesungen gehört

$$\{\text{s} \mid \text{s} \in \text{Studenten} \wedge \forall \text{v} \in \text{Vorlesungen}(\text{v.SWS}=4 \Rightarrow$$
$$\exists \text{h} \in \text{hören}(\text{h.VorlNr}=\text{v.VorlNr} \wedge \text{h.MatrNr}=\text{s.MatrNr}))\}$$

Definition des Tupelkalküls

Atome

- $s \in R$, mit s Tupelvariable und R Relationenname
- $s.A\,\phi\,t.B$, mit s und t Tupelvariablen, A und B Attributnamen und ϕ Vergleichsoperator ($=$, $\neq$, $\leq$, ...)
- $s.A\,\phi\,c$ mit c Konstante

Formeln

- Alle Atome sind Formeln
- Ist P Formel, so auch $\neg P$ und (P)
- Sind P_1 und P_2 Formeln, so auch $P_1 \wedge P_2$, $P_1 \vee P_2$ und $P_1 \Rightarrow P_2$
- Ist $P(t)$ Formel mit freier Variable t, so auch

$$\forall t \in R(P(t)) \quad \text{und} \quad \exists t \in R(P(t))$$

Sicherheit

Einschränkung auf Anfragen mit *endlichem* Ergebnis. Die folgende Beispielanfrage

$$\{n \mid \neg(n \in \mathit{Professoren})\}$$

ist *nicht* sicher. Das Ergebnis ist unendlich.

- Bedingung: Ergebnis des Ausdrucks muß Teilmenge der *Domäne* der Formel sein.
- Die Domäne einer Formel enthält
  - alle in der Formel vorkommenden Konstanten
  - alle Attributwerte von Relationen, die in der Formel referenziert werden

Der Domänenkalkül

Ein Ausdruck des Domänenkalküls hat die Form

$$\{[v_1, v_2, \ldots, v_n] \mid P(v_1, \ldots, v_n)\}$$

mit $v_1, \ldots, v_2$ Domänenvariablen und P Formel.

Beispiel: *MatrNr* und *Namen* der Prüflinge von Curie

$$\{[\text{m, n}] \mid \exists \text{s}([\text{m, n, s}] \in \text{Studenten} \land \exists \text{v, p, g}([\text{m, v, p, g}] \in \text{prüfen} \land$$
$$\exists \text{a, r, b}([\text{p, a, r, b}] \in \text{Professoren} \land \text{a} = \text{`Curie'})))\}$$

Definition des Domänenkalküls

Atome

- $[w_1, w_2, \ldots, w_m] \in R$, mit m-stelliger Relation R und Domänenvariablen $w_1, \ldots, w_m$
- $x\,\phi\,y$, mit x und y Domänenvariablen, ϕ Vergleichsoperator
- $x\,\phi\,c$, mit Konstanter c

Formeln

- Alle Atome sind Formeln
- Ist P Formel, so auch $\neg P$ und (P)
- Sind P_1 und P_2 Formeln, so auch $P_1 \vee P_2$, $P_1 \wedge P_2$ und $P_1 \Rightarrow P_2$
- Ist $P(v)$ Formel mit freier Variable v, so auch $\exists v(P(v))$ und $\forall v(P(v))$

Sicherheit des Domänenkalküls

- Sicherheit ist analog zum Tupelkalkül
- zum Beispiel ist

$$\{[p, n, r, o] \mid \neg([p, n, r, o] \in \mathit{Professoren})\}$$

  nicht sicher.

Ein Ausdruck

$$\{[x_1, x_2, \ldots, x_n] \mid P(x_1, x_2, \ldots, x_n)\}$$

ist sicher, falls folgende drei Bedingungen gelten:

1. Falls das Tupel $[c_1, c_2, \ldots, c_n]$ mit Konstante c_i im Ergebnis enthalten ist, so muß jedes c_i $(1 \leq i \leq n)$ in der Domäne von P enthalten sein.
2. Für jede existenz-quantifizierte Teilformel $\exists x(P_1(x))$ muß gelten, daß P_1 nur für Elemente aus der Domäne von P_1 erfüllbar sein kann – oder evtl. für gar keine. Mit anderen Worten, wenn für eine Konstante c das Prädikat $P_1(c)$ erfüllt ist, so muß c in der Domäne von P_1 enthalten sein.
3. Für jede universal-quantifizierte Teilformel $\forall x(P_1(x))$ muß gelten, daß sie dann und nur dann erfüllt ist, wenn $P_1(x)$ für alle Werte der Domäne von P_1 erfüllt ist. Mit anderen Worten, $P_1(d)$ muß für alle d, die *nicht* in der Domäne von P_1 enthalten sind, auf jeden Fall erfüllt sein.

Ausdruckskraft

Die drei Sprachen

1. relationale Algebra,
2. relationaler Tupelkalkül, eingeschränkt auf *sichere* Ausdrücke und
3. relationaler Domänenkalkül, eingeschränkt auf *sichere* Ausdrücke.

sind gleich mächtig.

SQL

- standardisierte
  - Datendefinitions (DDL)-
  - Datenmanipulations (DML)-
  - Anfrage (Query)-Sprache
- derzeit aktueller Standard ist SQL 92 oder kurz SQL 2
- SQL 3 wird derzeit ausgearbeitet
  - objektrelationale Erweiterung
- Für praktische Übungen steht eine Web-Seite zur Verfügung: http://www.db.fmi.uni-passau.de/sql
- Man kann eigene Relationen anlegen und/oder die Uni-DB verwenden
- Oracle 8-Dialekt von SQL

(Einfache) Datendefinition in SQL

Datentypen

- **character** (n), **char** (n)
- **character varying** (n), **varchar** (n)
- **numeric** (p, s), **integer**
- **blob** oder **raw** für sehr große binäre Daten

Anlegen von Tabelle

```
create table Professoren
    ( PersNr integer not null,
      Name   varchar(30) not null,
      Rang   character(2) );
```

Einfache SQL-Anfragen

select PersNr, Name
from Professoren
where Rang = 'C4';

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Einfache SQL-Anfragen

Sortierung

select PersNr, Name, Rang
from Professoren
order by Rang **desc**, Name **asc**;

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Duplikateliminierung

select distinct Rang
from Professoren;

Rang
C3
C4

Anfragen über mehrere Relationen

Welcher Professor liest „Mäeutik“?

select Name, Titel
from Professoren, Vorlesungen
where PersNr = gelesenVon **and** Titel = 'Mäeutik';

$$\Pi_{\text{Name, Titel}}(\sigma_{\text{PersNr}=\text{gelesenVon} \wedge \text{Titel}=\text{'Mäeutik'}}(\text{Professoren} \times \text{Vorlesungen}))$$

Anfragen über mehrere Relationen

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
⋮	⋮	⋮	⋮
2137	Kant	C4	7

Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

$\searrow$ Verknüpfung $\swarrow$

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Auswahl

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projektion

Name	Titel
Sokrates	Mäeutik

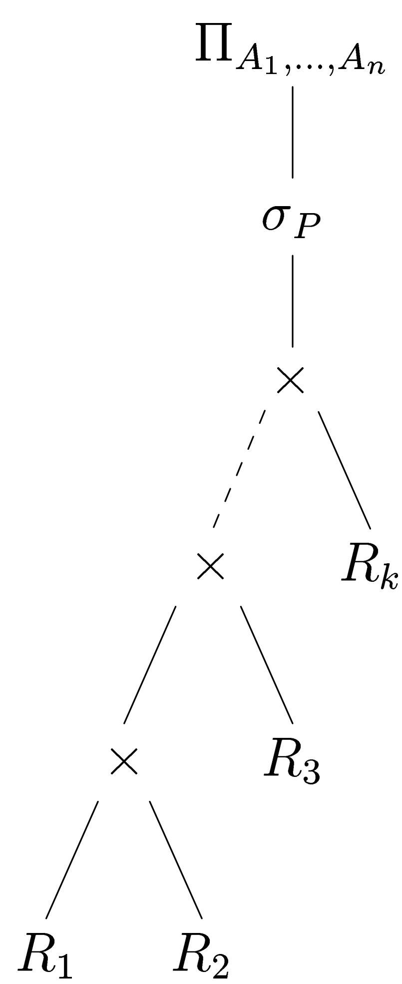
Kanonische Übersetzung in die relationale Algebra

Allgemein hat eine (ungeschachtelte) SQL-Anfrage die Form:

select $A_1, \ldots, A_n$
from $R_1, \ldots, R_k$
where P;

Übersetzung in die relationale Algebra:

$$\Pi_{A_1,\ldots,A_n}(\sigma_P(R_1 \times \ldots \times R_k))$$



Anfragen über mehrere Relationen

Welche Studenten hören welche Vorlesungen?

select Name, Titel
from Studenten, hören, Vorlesungen
where Studenten.MatrNr = hören.MatrNr **and**
hören.VorlNr = Vorlesungen.VorlNr;

Alternativ:

select s.Name, v.Titel
from Studenten s, hören h, Vorlesungen v
where s.MatrNr = h.MatrNr **and**
h.VorlNr = v.VorlNr;

Mengenoperationen und geschachtelte Anfragen

Mengenoperationen **union, intersect, minus**

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren );
```

Existenzquantor **exists**

```
select Name
from Professoren
where not exists ( select *
                   from Vorlesungen
                   where gelesenVon = PersNr );
```

Mengenmitgliedschaft **in**

```
select Name
from Professoren
where PersNr not in ( select gelesenVon
                      from Vorlesungen );
```

Der Vergleich mit „all“

Kein vollwertiger Allquantor!

select Name
from Studenten
where Semester >= **all** (**select** Semester
                                          from Studenten);

Aggregatfunktionen und Gruppierung

Aggregatfunktionen **avg, max, min, count, sum**

> **select avg**(Semester)
> **from** Studenten;
>
> **select** gelesenVon, **sum**(SWS)
> **from** Vorlesungen
> **group by** gelesenVon;
>
> **select** gelesenVon, Name, **sum**(SWS)
> **from** Vorlesungen, Professoren
> **where** gelesenVon = PersNr **and** Rang = 'C4'
> **group by** gelesenVon, Name
>   **having avg**(SWS) > 3;

- SQL erzeugt pro Gruppe ein Ergebnistupel
- Deshalb müssen alle in der **select**-Klausel aufgeführten Attribute – außer den aggregierten – auch in der **group by**-Klausel aufgeführt werden
- Nur so kann SQL sicherstellen, daß sich das Attrbiut nicht innerhalb der Gruppe ändert

Ausführung einer Anfrage mit group by

Vorlesungen × Professoren							
VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **where**-Bedingung

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Gruppierung

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

$\Downarrow$ **having**-Bedingung

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

$\Downarrow$ Aggregation (**sum**) und Projektion

gelesenVon	Name	**sum**(SWS)
2125	Sokrates	10
2137	Kant	8

Geschachtelte Anfragen (Forts.)

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind exakt durchschnittlich verlaufen?

```
select *
from prüfen
where Note = ( select avg(Note)
               from prüfen );
```

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, daß die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select PersNr, Name, (select sum(SWS) as Lehrbelastung
                      from Vorlesungen
                      where gelesenVon = PersNr)
from Professoren;
```

Unkorrelierte versus korrelierte Unteranfragen

- korrelierte Formulierung

```
select s.*
from Studenten s
where exists
        ( select p.*
          from Professoren p
          where p.GebDatum > s.GebDatum );
```

- Äquivalente, unkorrelierte Formulierung

```
select s.*
from Studenten s
where s.GebDatum <
        ( select max(p.GebDatum)
          from Professoren p );
```

- Vorteil: Unteranfrageergebnis kann materialisiert werden
- Unteranfrage braucht nur einmal ausgewertet zu werden

Entschachtelung korrelierter Unteranfragen – Forts.

```
select a.*
from Assistenten a
where exists
        ( select p.*
          from Professoren p
          where a.Boss = p.PersNr and p.GebDatum > a.GebDatum );
```

- Entschachtelung durch Join

```
select a.*
from Assistenten a, Professoren p
where a.Boss = p.PersNr and p.GebDatum > a.GebDatum;
```

Verwertung der Ergebnismenge einer Unteranfrage

select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (**select** s.MatrNr, s.Name, **count**(∗) **as** VorlAnzahl
    from Studenten s, hören h
    where s.MatrNr = h.MatrNr
    group by s.MatrNr, s.Name) tmp
where tmp.VorlAnzahl > 2;

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

Decision-Support-Anfrage mit geschachtelten Unteranfragen

select h.VorlNr, h.AnzProVorl, g.GesamtAnz,
    h.AnzProVorl/g.GesamtAnz **as** Marktanteil
from (**select** VorlNr, **count**(*) **as** AnzProVorl
    from hören
    group by VorlNr) h,
   (**select count**(*) **as** GesamtAnz
    from Studenten) g;

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	.125
5001	4	8	.5
5022	2	8	.25
...	...	...	...

Weitere Anfragen mit Unteranfragen

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren );
```

```
select Name
from Professoren
where PersNr not in ( select gelesenVon
                      from Vorlesungen );
```

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten );
```

Quantifizierte Anfragen in SQL

- Existenzquantor: **exists**

select Name
from Professoren
where not exists (**select** $*$
                                from Vorlesungen
                                where gelesenVon = PersNr);

Allquantifizierung

- SQL-92 hat keinen Allquantor
- Allquantifizierung muß also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden
- Kalkülformulierung der Anfrage: Wer hat **alle** vierstündigen Vorlesungen gehört?

$$\{\mathrm{s} \mid \mathrm{s} \in \mathrm{Studenten} \wedge \forall \mathrm{v} \in \mathrm{Vorlesungen}(\mathrm{v.SWS}{=}4 \Rightarrow$$
$$\exists \mathrm{h} \in \mathrm{hören}(\mathrm{h.VorlNr}{=}\mathrm{v.VorlNr} \wedge \mathrm{h.MatrNr}{=}\mathrm{s.MatrNr}))\}$$

- Elimination von $\forall$ und $\Rightarrow$
- Dazu sind folgende Äquivalenzen anzuwenden:

$$\begin{aligned} \forall t \in R(P(t)) &= \neg(\exists t \in R(\neg P(t))) \\ R \Rightarrow T &= \neg R \vee T \end{aligned}$$

- Wie erhalten:

$$\{s \mid s \in \text{Studenten} \land \neg(\exists v \in \text{Vorlesungen}\neg(\neg(v.\text{SWS}{=}4) \lor \\ \exists h \in \text{hören}(h.\text{VorlNr}{=}v.\text{VorlNr} \land h.\text{MatrNr}{=}s.\text{MatrNr}))\}$$

- Anwendung von DeMorgan ergibt schließlich:

$$\{s \mid s \in \text{Studenten} \land \neg(\exists v \in \text{Vorlesungen}(v.\text{SWS}{=}4 \land \\ \neg(\exists h \in \text{hören}(h.\text{VorlNr}{=}v.\text{VorlNr} \land h.\text{MatrNr}{=}s.\text{MatrNr}))))\}$$

- SQL-Umsetzung folgt direkt:

```
select s.*
from Studenten s
where not exists
        (select *
         from Vorlesungen v
         where v.SWS = 4 and not exists
                (select *
                 from hören h
                 where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

Allquantifizierung durch count-Aggregation

- die Allquantifizierung kann immer auch durch eine **count**-Aggregation ausgedrückt werden
- Wir betrachten dazu eine etwas einfachere Anfrage, in der wir die (*MatrNr* der) Studenten ermitteln wollen, die *alle* Vorlesungen hören:

  select h.MatrNr
  from hören h
  group by h.MatrNr
      having count(∗) = (**select** count(∗) **from** Vorlesungen);

Herausforderung

- Wie formuliert man die komplexere Anfrage: Wer hat alle vierstündigen Vorlesungen gehört
- Grundidee besteht darin, vorher durch einen Join die Studenten/Vorlesungs-Paare einzuschränken und danach das Zählen durchzuführen

Nullwerte

- unbekannter Wert
- wird vielleicht später nachgereicht
- Nullwerte können auch im Zuge der Anfrageauswertung entstehen (Bsp. äußere Joins)
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen

  select **count**(∗)
  from Studenten
  where Semester < 13 **or** Semester >= 13

- Wenn es Studenten gibt, deren *Semester*-Attribut den Wert **null** hat, werden diese nicht mitgezählt.
- Der Grund liegt in folgenden Regeln für den Umgang mit **null**-Werten begründet:

Auswertungsregeln bei Null-Werten

1. In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis **null**. Dementsprechend wird z.B. **null** $+ 1$ zu **null** ausgewertet – aber auch **null** $* 0$ wird zu **null** ausgewertet.
2. SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente **null** ist. Beispielsweise wertet SQL das Prädikat $(PersNr = \ldots)$ immer zu **unknown** aus, wenn die *PersNr* des betreffenden Tupels den Wert **null** hat.
3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

Diese Berechnungsvorschriften sind recht intuitiv. **unknown or true** wird z.B. zu **true** – die Disjunktion ist mit dem **true**-Wert des rechten Arguments immer erfüllt, unabhängig von der Belegung des linken Arguments. Analog ist **unknown and false** automatisch **false** – keine Belegung des linken Arguments könnte die Konjunktion mehr erfüllen.

4. In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** auswertet, nicht ins Ergebnis aufgenommen.

5. Bei einer Gruppierung wird **null** als ein eigenständiger Wert aufgefaßt und in eine eigene Gruppe eingeordnet.

Spezielle Sprachkonstrukte („syntaktischer Zucker“)

```
select *
from Studenten
where Semester >= 1 and Semester <= 4;

select *
from Studenten
where Semester between 1 and 4;

select *
from Studenten
where Semester in (1,2,3,4);

select *
from Studenten
where Name like 'T%eophrastos';

select distinct s.Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
      v.Titel like '%thik%';
```

Das case-Konstrukt

select MatrNr, (**case** **when** Note < 1.5 **then** 'sehr gut'
when Note < 2.5 **then** 'gut'
when Note < 3.5 **then** 'befriedigend'
when Note <= 4.0 **then** 'ausreichend'
else 'nicht bestanden' **end**)
from prüfen;

- Die **erste** qualifizierende **when**-Klausel wird ausgeführt

Vergleiche mit like

Platzhalter „%“,„_“

- „%“ steht für beliebig viele (auch gar kein) Zeichen
- „_“ steht für genau ein Zeichen

select ∗
from Studenten
where Name **like** ’T%eophrastos’;

select distinct Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr **and** h.VorlNr = v.VorlNr **and**
v.Titel = ’%thik%’;

Joins in SQL-92

- **cross join**: Kreuzprodukt
- **natural join**: natürlicher Join
- **join** oder **inner join**: Theta-Join
- **left**, **right** oder **full outer join**: äußerer Join
- **union join**: Vereinigungs-Join (wird hier nicht vorgestellt)

select $*$
from R_1, R_2
where $R_1.A = R_2.B$;

$\Downarrow$

select $*$
from R_1 **join** R_2 **on** $R_1.A = R_2.B$;

Äußere Joins

select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from Professoren p **left outer join**
(prüfen f **left outer join** Studenten s **on** f.MatrNr = s.MatrNr)
on p.PersNr = f.PersNr;

p.PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopenhauer
2136	Curie	–	–	–	–	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Äußere Joins

select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from Professoren p **right outer join**
(prüfen f **right outer join** Studenten s **on** f.MatrNr = s.MatrNr)
on p.PersNr = f.PersNr;

p.PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopenhauer
–	–	–	–	–	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

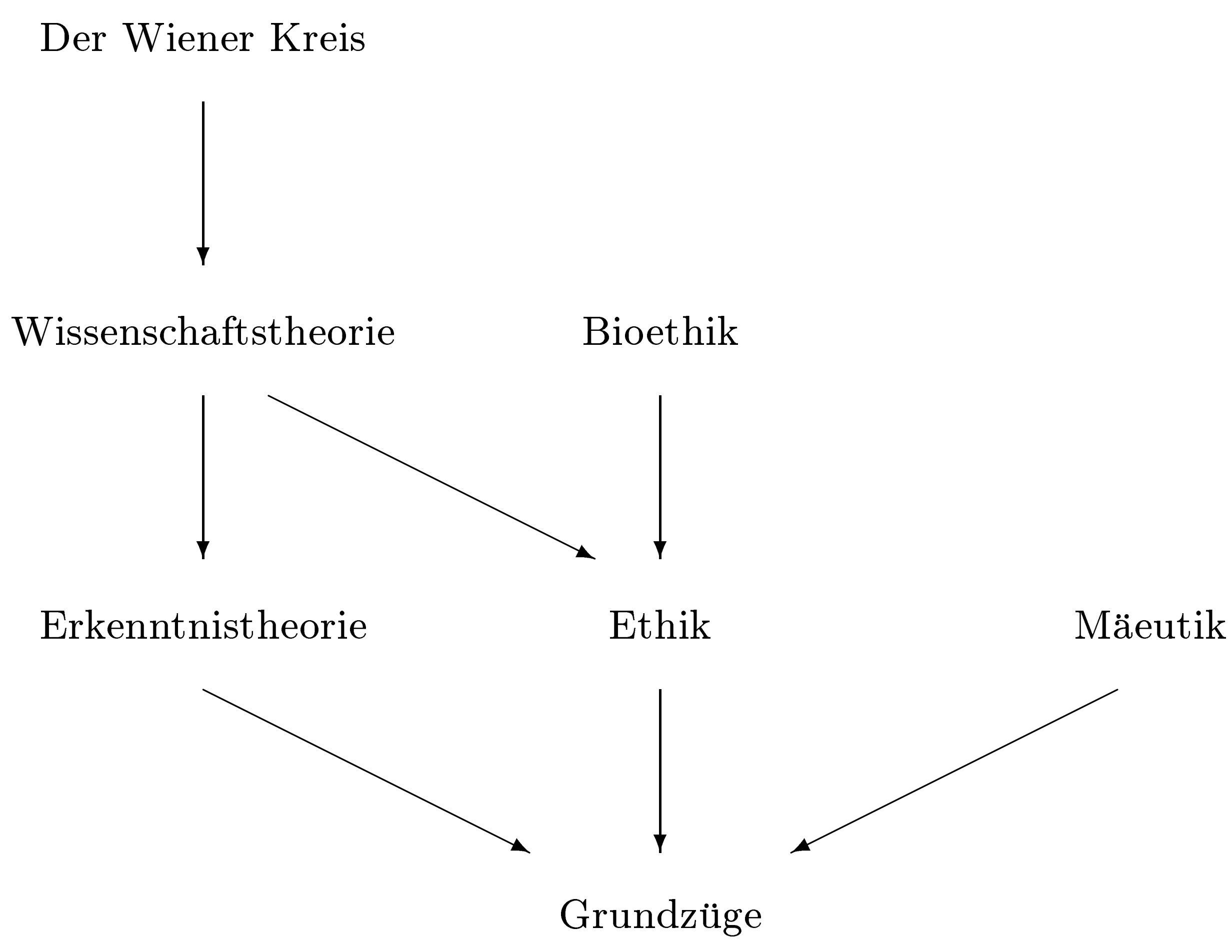
Äußere Joins

select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from Professoren p **full outer join**
(prüfen f **full outer join** Studenten s **on** f.MatrNr = s.MatrNr)
on p.PersNr = f.PersNr;

p.PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopenhauer
–	–	–	–	–	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2136	Curie	–	–	–	–	–
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Rekursion

```
select Vorgänger
from voraussetzen, Vorlesungen
where Nachfolger = VorlNr and
      Titel = 'Der Wiener Kreis';
```

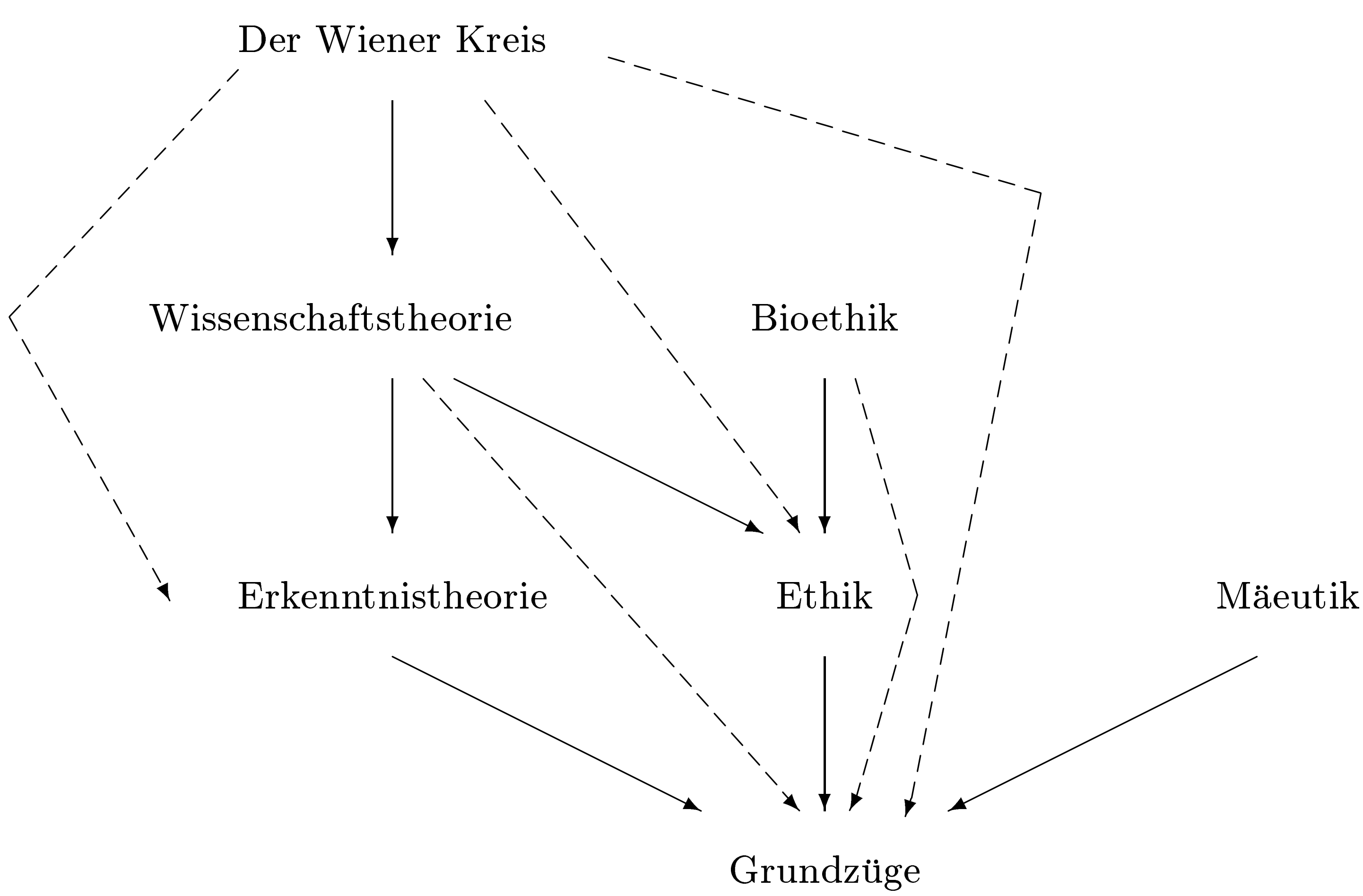


select v1.Vorgänger
from voraussetzen v1, voraussetzen v2, Vorlesungen v
where v1.Nachfolger = v2.Vorgänger **and**
v2.Nachfolger = v.VorlNr **and**
v.Title = 'Der Wiener Kreis';

select v1.Vorgänger
from voraussetzen v1,
⋮
voraussetzen vn_minus_1
voraussetzen vn,
Vorlesungen v
where v1.Nachfolger = v2.Vorgänger **and**
⋮
vn_minus_1.Nachfolger = vn.Vorgänger **and**
vn.Nachfolger = v.VorlNr **and**
v.Titel = 'Der Wiener Kreis' ;

Transitive Hülle

$$
\begin{aligned}
trans_{A,B}(R) = \{(a,b) \mid\ & \exists k \in I\!N\ (\exists \tau_1, \ldots, \tau_k \in R(\\
& \tau_1.A = \tau_2.B\ \wedge \\
& \vdots \\
& \tau_{k-1}.A = \tau_k.B\ \wedge \\
& \tau_1.A = a\ \wedge \\
& \tau_k.B = b))\}
\end{aligned}
$$



Die connect by-Klausel

select Titel
from Vorlesungen
where VorlNr **in** (**select** Vorgänger
  from voraussetzen
  connect by Nachfolger = **prior** Vorgänger
  start with Nachfolger = (**select** VorlNr
    from Vorlesungen
    where Titel = 'Der Wiener Kreis'));

Titel
Grundzüge
Ethik
Erkenntnistheorie
Wissenschaftstheorie

Rekursion in DB2/SQL3: gleiche Anfrage

```
with TransVorl (Vorg , Nachf)
as ( select Vorgänger, Nachfolger from voraussetzen
     union all
     select t.Vorg, v.Nachfolger
     from TransVorl t, voraussetzen v
     where t.Nachf = v.Vorgänger )

select Titel from Vorlesungen where VorlNr in
     ( select Vorg from TransVorl where Nachf in
          ( select VorlNr from Vorlesungen where Titel = 'Der Wiener Kreis' ) )
```

- zuerst wird eine temporäre Sicht *TransVorl* mit der **with**-Klausel angelegt
- Diese Sicht *TransVorl* ist rekursiv definiert, da sie selbst in der Definition vorkommt
- Aus dieser Sicht werden dann die gewünschten Tupel extrahiert
- Ergebnis ist natürlich wie gehabt

Veränderungen am Datenbestand

Einfügen von Tupeln

insert into hören
  select MatrNr, VorlNr
  from Studenten, Vorlesungen
  where Titel = 'Logik';

insert into Studenten (MatrNr, Name)
  values (28121, 'Archimedes');

Studenten		
MatrNr	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	–

Veränderungen am Datenbestand

Löschen von Tupeln

delete Studenten
where Semester > 13;

Verändern von Tupeln

update Studenten
    set Semester = Semester + 1;

Zweistufiges Vorgehen bei Änderungen

1. die Kandidaten für die Änderung werden ermittelt und "markiert"
2. die Änderung wird an den in Schritt 1. ermittelten Kandidaten durchgeführt

Anderenfalls könnte die Änderungsoperation von der Reihenfolge der Tupel abhängen, wie folgendes Beispiel zeigt:

delete from voraussetzen
  where Vorgänger **in** (**select** Nachfolger
                from voraussetzen);

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Ohne einen Markierungsschritt hängt das Ergebnis dieser Anfrage von der Reihenfolge der Tupel in der Relation ab. Eine Abarbeitung in der Reihenfolge der Beispielausprägung würde das letzte Tupel (5052, 5259) fälschlicherweise erhalten, da vorher bereits alle Tupel mit 5052 als *Nachfolger* entfernt wurden.

Sichten ...

für den Datenschutz

create view prüfenSicht **as**
  select MatrNr, VorlNr, PersNr
  from prüfen;

für die Vereinfachung von Anfragen

create view StudProf(SName, Semester, Titel, PName) **as**
  select s.Name, s.Semester, v.Titel, p.Name
  from Studenten s, hören h, Vorlesungen v, Professoren p
  where s.MatrNr = h.MatrNr **and** h.VorlNr = v.VorlNr **and**
    v.gelesenVon = p.PersNr;

select distinct Semester
from StudProf
where PName = 'Sokrates';

Sichten zur Modellierung von Generalisierungen

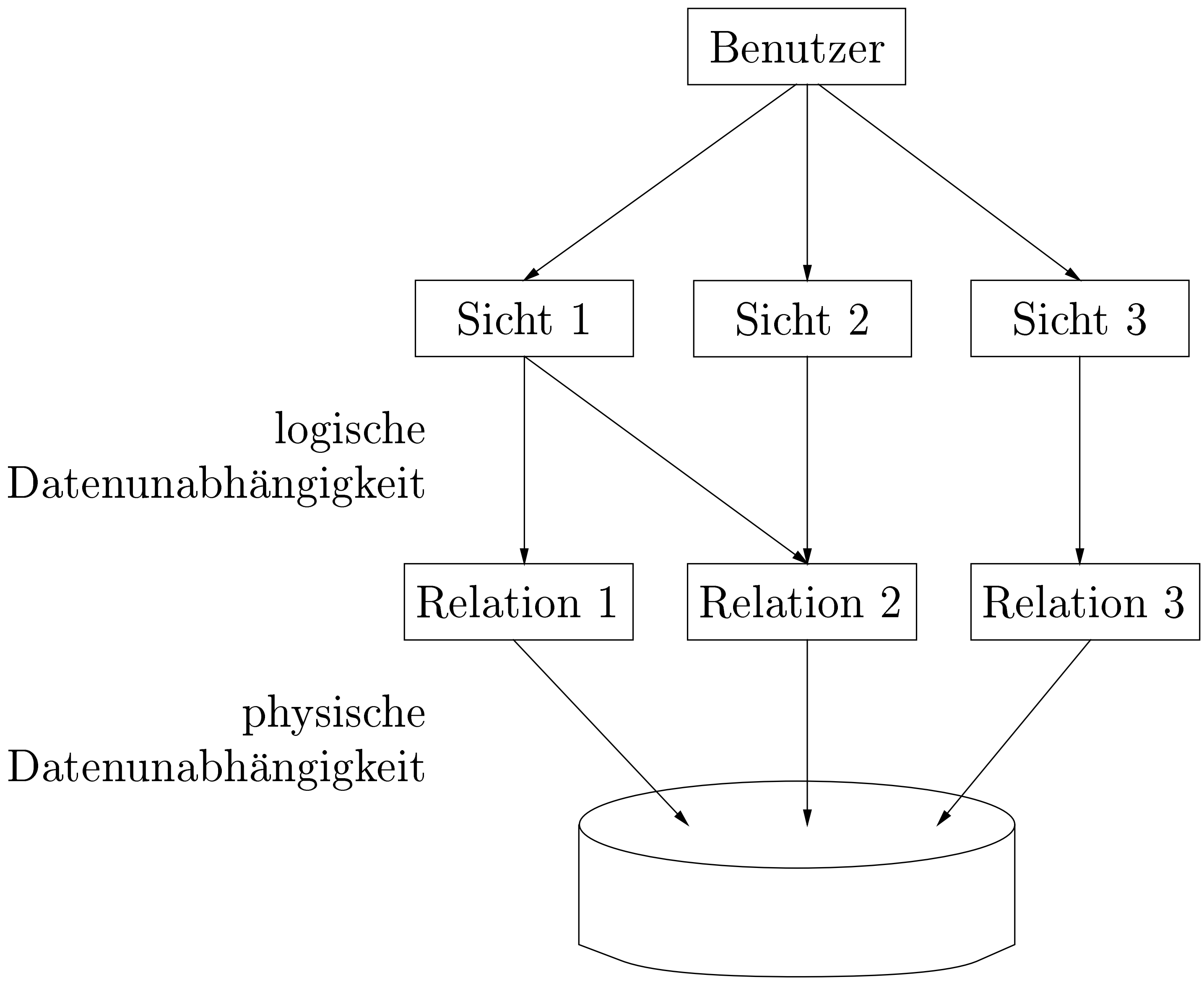
```
create table Angestellte
  ( PersNr     integer not null,
    Name       varchar(30) not null );

create table ProfDaten
  ( PersNr     integer not null,
    Rang       character(2),
    Raum       integer);

create table AssiDaten
  ( PersNr     integer not null,
    Fachgebiet varchar(30),
    Boss       integer);

create view Professoren as
    select *
    from Angestellte a, ProfDaten d
    where a.PersNr = d.PersNr;

create view Assistenten as
    select *
    from Angestellte a, AssiDaten d
    where a.PersNr = d.PersNr;
```

a) Untertypen als Sicht

```
create table Professoren
  ( PersNr     integer not null,
    Name       varchar(30) not null,
    Rang       character(2),
    Raum       integer);

create table Assistenten
  ( PersNr     integer not null,
    Name       varchar(30) not null,
    Fachgebiet varchar(30),
    Boss       integer);

create table AndereAngestellte
  ( PersNr     integer not null,
    Name       varchar(30) not null);

create view Angestellte as
   ( select PersNr, Name
     from Professoren )
   union
   ( select PersNr, Name
     from Assistenten )
   union
   ( select *
     from AndereAngestellte );
```

b) Obertypen als Sicht

Sichten zur Gewährleistung von Datenunabhängigkeit



Änderbarkeit von Sichten

Beispiele für nicht änderbare Sichten:

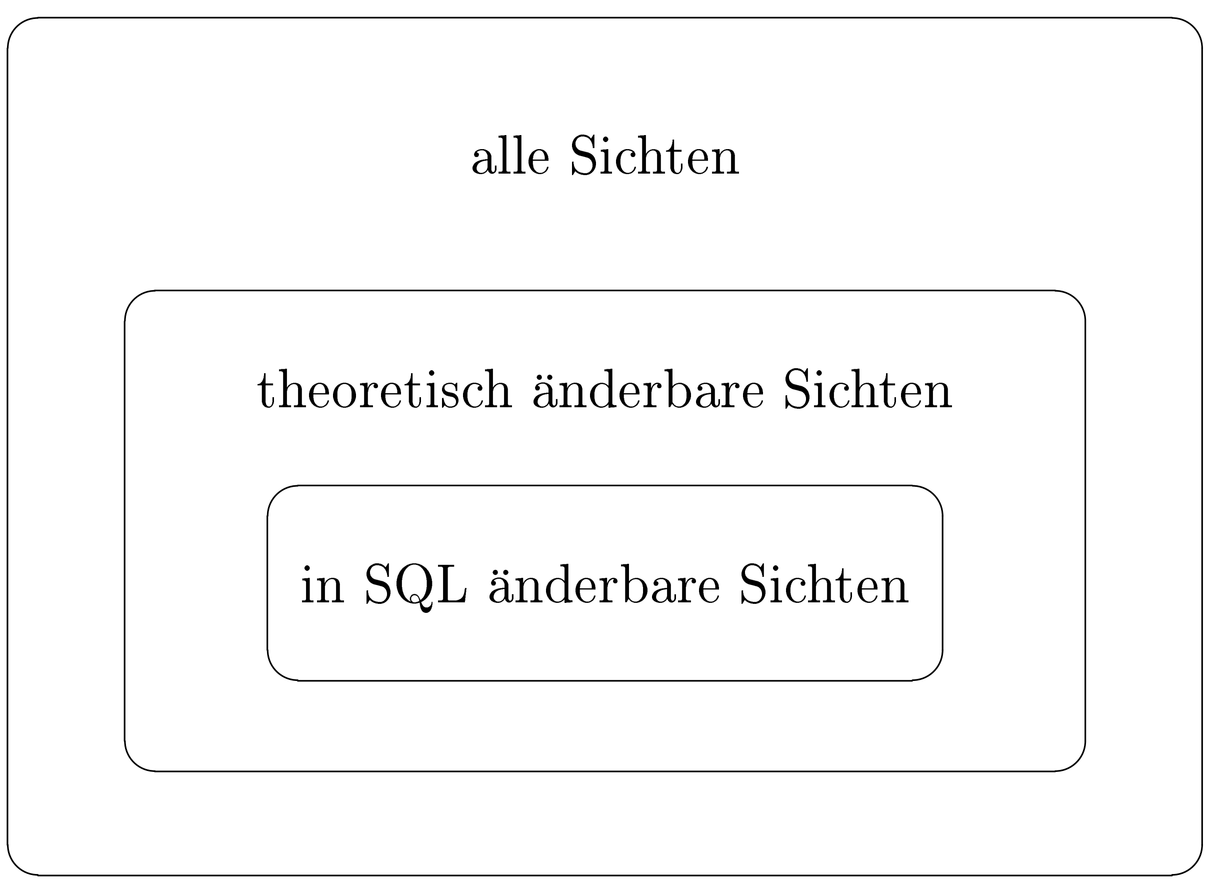
create view WieHartAlsPrüfer(PersNr, Durchschnittsnote) **as**
  select PersNr, **avg**(Note)
  from prüfen
  group by PersNr;

create view VorlesungenSicht **as**
  select Titel, SWS, Name
  from Vorlesungen, Professoren
  where gelesenVon = PersNr;

insert into VorlesungenSicht
  values ('Nihilismus', 2, 'Nobody');

Änderbarkeit von Sichten

- in SQL
  - nur eine Basisrelation
  - Schlüssel muß vorhanden sein
  - keine Aggregatfunktionen, Gruppierung und Duplikateliminerung
- allgemein



Embedded SQL

```
#include <stdio.h>

/* Kommunikationsvariablen deklarieren */
exec sql begin declare section;
    varchar user_passwd[30];
    int exMatrNr;
exec sql end declare section;

exec sql include SQLCA;

main()
{
    printf("Name/Password: ");
    scanf("%s", user_passwd.arr);
    user_passwd.len = strlen(user_passwd.arr);

    exec sql whenever sqlerror goto error;
    exec sql connect :user_passwd;

    while (1) {
        printf("Matrikelnummer (0 zum beenden): ");
        scanf("%d", &exMatrNr);
        if (!exMatrNr) break;

        exec sql delete from Studenten
                where MatrNr = :exMatrNr;
    }
    exec sql commit work release;
    exit(0);
error:
    exec sql whenever sqlerror continue;
    exec sql rollback work release;
    printf("Fehler aufgetreten!\n");
    exit(-1); }
```

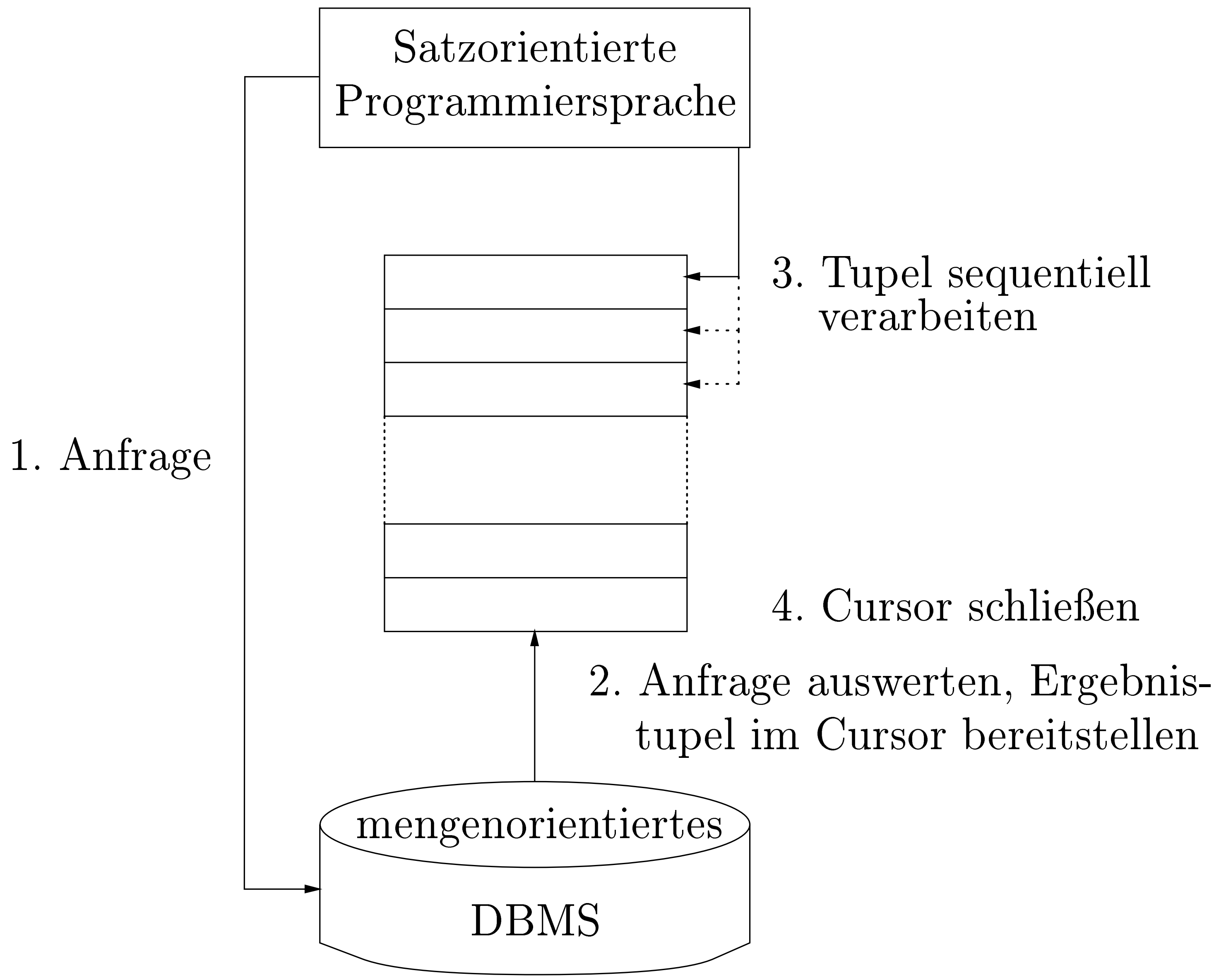
Anfragen in Anwendungsprogrammen

- genau ein Tupel im Ergebnis

  exec sql select avg(Semester)
  into :avgsem
  from Studenten;

Anfragen in Anwendungsprogrammen

- mehrere Tupel im Ergebnis



Cursor-Schnittstelle in SQL

1. **exec sql declare** c4profs **cursor for**
   select Name, Raum
   from Professoren
   where Rang = 'C4';
2. **exec sql open** c4profs;
3. **exec sql fetch** c4profs **into** :pname, :praum;
4. **exec sql close** c4profs;

Query by Example

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
		p. _t	>3	

Analog

$$\{[t] \mid \exists v, s, r([v, t, s, r] \in \text{Vorlesungen} \ \land \ s > 3)\}$$

Join in QBE

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
		Mäeutik		_x

Professoren	PersNr	Name	Rang	Raum
	_x	**p.** _n		

Die Condition Box

Studenten	MatrNr	Name	Semester
		_s	_a

Studenten	MatrNr	Name	Semester
		_t	_b

conditions
_a > _b

Betreuen	potentiellerTutor	Betreuter
p.	_s	_t

Aggregatfunktionen und Gruppierung

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
			p.sum.all. _x	**p.g.**

conditions
avg.all. _x>2

Updates in QBE

Professoren	PersNr	Name	Rang	Raum
d.	_x	Sokrates		

Vorlesungen	VorlNr	Titel	SWS	gelesenVon
d.	_y			_x

hören	VorlNr	MatrNr
d.	_y	

Datenintegrität

Integritätsbedingungen

- Schlüssel
- Beziehungskardinalitäten
- Attributdomänen
- Inklusion bei Generalisierungen

statische Integritätsbedingungen

- Bedingungen an den Zustand der Datenbasis

dynamische Integritätsbedingungen

- Bedingungen an Zustandsübergänge

Referentielle Integrität

Fremdschlüssel

- verweisen auf Tupel einer Relation
- z.B. *gelesenVon* in *Vorlesungen* verweist auf Tupel in Professoren

referentielle Integrität

- Fremdschlüssel müssen auf existierende Tupel verweisen oder einen Nullwert enthalten

Referentielle Integrität in SQL

- Kandidatenschlüssel: **unique**
- Primärschlüssel: **primary key**
- Fremdschlüssel: **foreign key**

Beispiel:

> **create table** R
> (α **integer primary key**,
> $\ldots$);
> **create table** S
> ($\ldots$,
> κ **integer references** R);

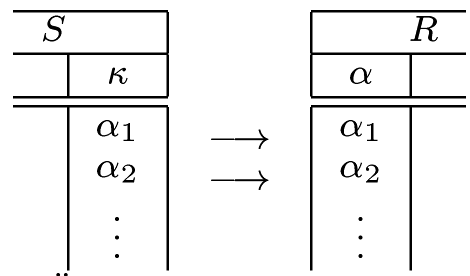
Einhaltung referentieller Integrität

Änderungen von referenzierten Daten

1. Default: Zurückweisen der Änderungsoperation
2. Propagieren der Änderungen: **cascade**
3. Verweise auf Nullwert setzen: **set null**

Einhaltung referentieller Integrität

Originalzustand

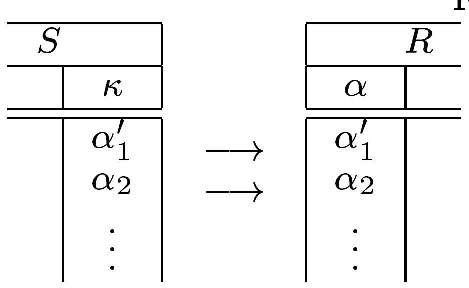


Änderungsoperationen

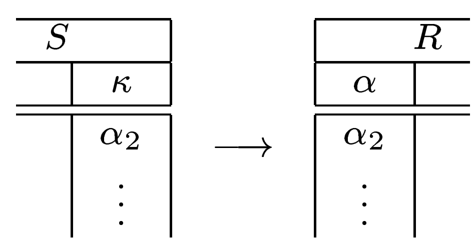
update R
  set $\alpha = \alpha_1'$
  where $\alpha = \alpha_1$;

delete from R
  where $\alpha = \alpha_1$;

Kaskadieren

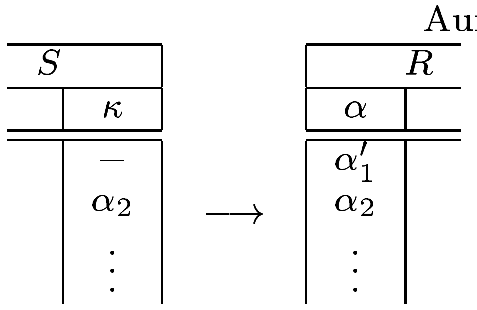


create table S
  $(\ldots,$
  κ **integer references** R
  on update cascade);

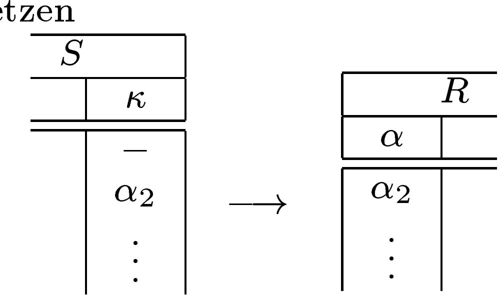


create table S
  $(\ldots,$
  κ **integer references** R
  on delete cascade);

Auf Null setzen

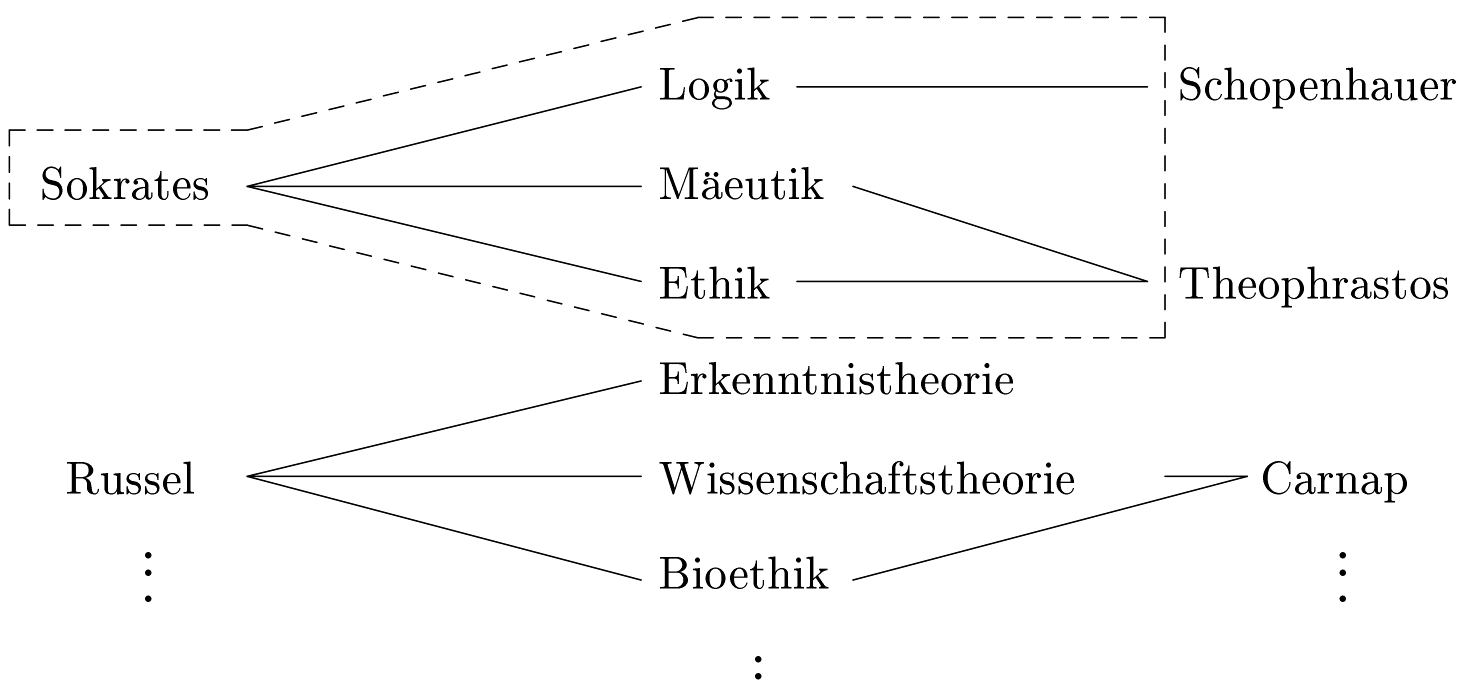


create table S
  $(\ldots,$
  κ **integer references** R
  on update set null);



create table S
  $(\ldots,$
  κ **integer references** R
  on delete set null);

Kaskadierendes Löschen




```
create table Vorlesungen
  ( ...,
    gelesenVon integer
               references Professoren
               on delete cascade);
```

```
create table hören
  ( ...,
    VorlNr integer
           references Vorlesungen
           on delete cascade);
```

Einfache statistische Integritätsbedingungen

- Wertebereichseinschränkungen
  ...**check** Semester **between** 1 **and** 13
- Aufzählungstypen
  ...**check** Rang **in** (‘C2’,‘C3’,‘C4’)...

Das Universitätsschema mit Integritätsbedingungen

```
create table Studenten
    ( MatrNr        integer primary key,
      Name          varchar(30) not null,
      Semester      integer check Semester between 1 and 13);

create table Professoren
    ( PersNr        integer primary key,
      Name          varchar(30) not null,
      Rang          character(2) check (Rang in ('C2', 'C3', 'C4')),
      Raum          integer unique);

create table Assistenten
    ( PersNr        integer primary key,
      Name          varchar(30) not null,
      Fachgebiet    varchar(30),
      Boss          integer,
      foreign key   (Boss) references Professoren on delete set null);

create table Vorlesungen
    ( VorlNr        integer primary key,
      Titel         varchar(30),
      SWS           integer,
      gelesenVon    integer references Professoren on delete set null);
```

Das Universitätsschema mit Integritätsbedingungen

```
create table hören
    ( MatrNr        integer references Studenten on delete cascade,
      VorlNr        integer references Vorlesungen on delete cascade,
      primary key   (MatrNr, VorlNr));

create table voraussetzen
    ( Vorgänger     integer references Vorlesungen on delete cascade,
      Nachfolger    integer references Vorlesungen on delete cascade,
      primary key   (Vorgänger, Nachfolger));

create table prüfen
    ( MatrNr        integer references Studenten on delete cascade,
      VorlNr        integer references Vorlesungen,
      PersNr        integer references Professoren on delete set null,
      Note          numeric(2,1) check (Note between 0.7 and 5.0),
      primary key   (MatrNr, VorlNr));
```

```
create trigger keineDegradierung
before update on Professoren
for each row
when (old.Rang is not null)
begin
    if :old.Rang = 'C3' and :new.Rang = 'C2' then
        :new.Rang := 'C3';
    end if;
    if :old.Rang = 'C4' then
        :new.Rang := 'C4';
    end if;
    if :new.Rang is null then
        :new.Rang := :old.Rang;
    end if;
end
```

Relationale Entwurfstheorie

- Bewertung der Qualität eines relationalen Datenbankentwurfs
  - Redundanz
  - Einhaltung von Konsistenzbedingungen (funktionaler Abhängigkeiten)
- Normalformen
- Ggfls. Verbesserung der Qualität durch Zerlegung der Relationen

Funktionale Abhängigkeiten

Schema: $\mathcal{R} = \{A, B, C, D\}$
Ausprägung: R

- $\alpha \subseteq \mathcal{R}, \beta \subseteq \mathcal{R}$
- $\alpha \rightarrow \beta$ falls $\forall r, s \in R$ mit $r.\alpha = s.\alpha$ gilt: $r.\beta = s.\beta$

Funktionale Abhängigkeiten

	R			
	A	B	C	D
t	a_4	b_2	c_4	d_3
p	a_1	b_1	c_1	d_1
q	a_1	b_1	c_1	d_2
r	a_2	b_2	c_3	d_2
s	a_3	b_2	c_4	d_3

$$\{A\} \rightarrow \{B\}$$
$$\{C, D\} \rightarrow \{B\}$$
$$\{B\} \not\rightarrow \{C\}$$

Konventionen zur Notation

$$CD \rightarrow A$$

$$\{C, D\} \rightarrow \{A\}$$

Einhaltung einer funktionalen Abhängigkeit

Die FD $\alpha \rightarrow \beta$ ist in R eingehalten genau dann wenn für alle c gilt:

$$| \Pi_\beta(\sigma_{\alpha=c}(R)) | \leq 1$$

- Eingabe: eine Relation R und eine FD $\alpha \rightarrow \beta$
- Ausgabe: *ja*, falls $\alpha \rightarrow \beta$ in R erfüllt ist; *nein* sonst
- $\mathit{Einhaltung}(R, \alpha \rightarrow \beta)$
  - sortiere R nach α-Werten
  - falls alle Gruppen bestehend aus Tupeln mit gleichen α-Werten auch gleiche β-Werte aufweisen: Ausgabe *ja*; sonst: Ausgabe *nein*

- $\alpha \subseteq \mathcal{R}$ ist ein *Superschlüssel wenn gilt:*

$$\alpha \to \mathcal{R}$$

- β ist *voll funktional abhängig* von α – in Zeichen $\alpha \stackrel{\bullet}{\to} \beta$ – falls beide nachfolgenden Kriterien gelten:

  1. $\alpha \to \beta$, d.h. β ist funktional abhängig von α und
  2. α kann nicht mehr „verkleinert“ werden, d.h.

$$\forall A \in \alpha : \alpha - \{A\} \not\to \beta$$

- $\alpha \subseteq \mathcal{R}$ ist ein Kandidatenschlüssel wenn gilt:

$$\alpha \stackrel{\bullet}{\to} \mathcal{R}$$

Städte			
Name	BLand	Vorwahl	EW
Frankfurt	Hessen	069	650000
Frankfurt	Brandenburg	0335	84000
München	Bayern	089	1200000
Passau	Bayern	0851	50000
...	...	...	...

Die Kandidatenschlüssel für die Relation *Städte* sind:

- {Name, BLand}
- {Name, Vorwahl}
  Man beachte, daß zwei (kleinere) Städte dieselbe Vorwahl haben können.

Bestimmung funktionaler Abhängigkeiten

$$
\begin{aligned}
\text{Professoren} \quad : \quad & \{[\text{PersNr}, \text{Name}, \text{Rang}, \text{Raum}, \text{Ort}, \text{Straße}, \\
& \text{PLZ}, \text{Vorwahl}, \text{BLand}, \text{EW}, \text{Landesregierung}]\}
\end{aligned}
$$

1. {PersNr} $\rightarrow$ {PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, BLand, EW, Landesregierung}
2. {Ort, BLand} $\rightarrow$ {EW, Vorwahl}
3. {PLZ} $\rightarrow$ {BLand, Ort, EW}
4. {Ort, BLand, Straße} $\rightarrow$ {PLZ}
5. {BLand} $\rightarrow$ {Landesregierung}
6. {Raum} $\rightarrow$ {PersNr}

Zusätzliche Abhängigkeiten, die aus obigen herleitbar sind:

- {Raum} $\rightarrow$ {PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, BLand, EW, Landesregierung}
- {PLZ} $\rightarrow$ {Landesregierung}

Bestimmung funktionaler Abhängigkeiten

Armstrong-Axiome

- *Reflexivität*: Falls β eine Teilmenge von α ist ($\beta \subseteq \alpha$) dann gilt immer $\alpha \to \beta$. Insbesondere gilt also immer $\alpha \to \alpha$.
- *Verstärkung*: Falls $\alpha \to \beta$ gilt, dann gilt auch $\alpha\gamma \to \beta\gamma$. Hierbei stehe z.B. $\alpha\gamma$ für $\alpha \cup \gamma$.
- *Transitivität*: Falls $\alpha \to \beta$ und $\beta \to \gamma$ gilt, dann gilt auch $\alpha \to \gamma$.

Diese Axiome sind *vollständig* und *korrekt*.

Zusätzliche Axiome erleichtern die Herleitung:

- *Vereinigungsregel*: Wenn $\alpha \to \beta$ und $\alpha \to \gamma$ gelten, dann gilt auch $\alpha \to \beta\gamma$.
- *Dekompositionsregel*: Wenn $\alpha \to \beta\gamma$ gilt, dann gelten auch $\alpha \to \beta$ und $\alpha \to \gamma$.
- *Pseudotransitivitätsregel*: Wenn $\alpha \to \beta$ und $\gamma\beta \to \delta$, dann gilt auch $\alpha\gamma \to \delta$.

Bestimmung der Hülle einer Attributmenge

- **Eingabe**: eine Menge F von FDs und eine Menge von Attributen α
- **Ausgabe**: die vollständige Menge von Attributen α^+, für die gilt $\alpha \to \alpha^+$
- $\mathit{AttrHülle}(F, \alpha)$

$\mathit{Erg} := \alpha$
while (Änderungen an Erg) **do**
    foreach FD $\beta \to \gamma$ **in** F **do**
        if $\beta \subseteq \mathit{Erg}$ **then** $\mathit{Erg} := \mathit{Erg} \cup \gamma$
Ausgabe $\alpha^+ = \mathit{Erg}$

Kanonische Überdeckung

F_c heißt kanonische Überdeckung von F, wenn die folgenden 3 Kriterien erfüllt sind:

1. $F_c \equiv F$, d.h. $F_c^+ = F^+$
2. In F_c existieren keine FDs $\alpha \to \beta$, bei denen α oder β überflüssige Attribute enthalten. D.h. es muß folgendes gelten:
   (a) $\forall A \in \alpha : (F_c - (\alpha \to \beta) \cup ((\alpha - A) \to \beta)) \not\equiv F_c$
   (b) $\forall B \in \beta : (F_c - (\alpha \to \beta) \cup (\alpha \to (\beta - B))) \not\equiv F_c$
3. Jede linke Seite einer funktionalen Abhängigkeit in F_c ist einzigartig. Dies kann durch sukzessive Anwendung der Vereinigungsregel auf FDs der Art $\alpha \to \beta$ und $\alpha \to \gamma$ erzielt werden, so daß die beiden FDs durch $\alpha \to \beta\gamma$ ersetzt werden.

Berechnung der kanonischen Überdeckung

1. Führe für jede FD $\alpha \to \beta \in F$ die Linksreduktion durch, also:

   - Überprüfe für alle $A \in \alpha$, ob A überflüssig ist, d.h. ob

     $$\beta \subseteq \mathit{AttrHülle}(F, \alpha - A)$$

     gilt. Falls dies der Fall ist, ersetze $\alpha \to \beta$ durch $(\alpha - A) \to \beta$.

2. Führe für jede (verbliebene) FD $\alpha \to \beta$ die Rechtsreduktion durch, also:

   - Überprüfe für alle $B \in \beta$, ob

     $$B \in \mathit{AttrHülle}(F - (\alpha \to \beta) \cup (\alpha \to (\beta - B)), \alpha)$$

     gilt. In diesem Fall ist B auf der rechten Seite überflüssig und kann eliminiert werden, d.h. $\alpha \to \beta$ wird durch $\alpha \to (\beta - B)$ ersetzt.

3. Entferne die FDs der Form $\alpha \to \emptyset$, die im 2. Schritt möglicherweise entstanden sind.

4. Fasse mittels der Vereinigungsregel FDs der Form $\alpha \to (\beta_1, \ldots, \alpha \to \beta_n)$ zusammen, so daß $\alpha \to \beta_1 \cup \cdots \cup \beta_n$ verbleibt.

„Schlechte“ Relationenschemata

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

Update-Anomalien

Einfügeanomalien

Löschanomalien

Zerlegung (Dekomposition) von Relationen

Es gibt zwei sehr grundlegende Korrektheitskriterien für eine solche Zerlegung von Relationenschemata:

1. *Verlustlosigkeit*: Die in der ursprünglichen Relationenausprägung R des Schemas $\mathcal{R}$ enthaltenen Informationen müssen aus den Ausprägungen $R_1, \ldots, R_n$ der neuen Relationenschemata $\mathcal{R}_1, \ldots, \mathcal{R}_n$ rekonstruierbar sein.

2. *Abhängigkeitserhaltung*: Die für $\mathcal{R}$ geltenden funktionalen Abhängigkeiten müssen auf die Schemata $\mathcal{R}_1, \ldots, \mathcal{R}_n$ übertragbar sein.

Kriterien für die Verlustlosigkeit einer Zerlegung

- $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$

$$
\begin{aligned}
R_1 &:= \Pi_{\mathcal{R}_1}(R) \\
R_2 &:= \Pi_{\mathcal{R}_2}(R)
\end{aligned}
$$

Die Zerlegung von $\mathcal{R}$ in $\mathcal{R}_1$ und $\mathcal{R}_2$ ist *verlustlos*, falls für jede mögliche (gültige) Ausprägung R von $\mathcal{R}$ gilt:

$$R = R_1 \bowtie R_2$$

Hinreichende Bedingung für die Verlustlosigkeit einer Zerlegung

- $(\mathcal{R}_1 \cap \mathcal{R}_2) \rightarrow \mathcal{R}_1 \in F_{\mathcal{R}}^+$ oder
- $(\mathcal{R}_1 \cap \mathcal{R}_2) \rightarrow \mathcal{R}_2 \in F_{\mathcal{R}}^+$

Andere Formulierung

- $\mathcal{R} = \alpha \cup \beta \cup \gamma$
- $\mathcal{R}_1 = \alpha \cup \beta$
- $\mathcal{R}_2 = \beta \cup \gamma$
- $\alpha \cap \gamma = \emptyset$
- $\beta \subseteq \mathit{AttrHülle}(F_{\mathcal{R}}, \alpha)$ oder
- $\gamma \subseteq \mathit{AttrHülle}(F_{\mathcal{R}}, \alpha)$

Beispiel eines Informationsverlustes nach Zerlegung

Biertrinker		
Kneipe	Gast	Bier
Kowalski	Kemper	Pils
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Hefeweizen

$\Pi_{\text{Kneipe, Gast}}$

$\Pi_{\text{Gast, Bier}}$

Besucht	
Kneipe	Gast
Kowalski	Kemper
Kowalski	Eickler
Innsteg	Kemper

Trinkt	
Gast	Bier
Kemper	Pils
Eickler	Hefeweizen
Kemper	Hefeweizen

$\bowtie$

$\neq$

Besucht $\bowtie$ Trinkt		
Kneipe	Gast	Bier
Kowalski	Kemper	Pils
Kowalski	*Kemper*	*Hefeweizen*
Kowalski	Eickler	Hefeweizen
Innsteg	*Kemper*	*Pils*
Innsteg	Kemper	Hefeweizen

Erläuterung des Biertrinker-Beispiels

Unser *Biertrinker*-Beispiel war eine „verlustige“ Zerlegung und dementsprechend war die Bedingung verletzt. Es gilt nämlich nur die eine nicht-triviale funktionale Abhängigkeit

- $\{\text{Kneipe}, \text{Gast}\} \rightarrow \{\text{Bier}\}$

wohingegen keine der zwei möglichen, die Verlustlosigkeit garantierenden FDs

- $\{\text{Gast}\} \rightarrow \{\text{Bier}\}$
- $\{\text{Gast}\} \rightarrow \{\text{Kneipe}\}$

erfüllt ist.

Beispiel einer verlustfreien Zerlegung

$Eltern : \{[Vater, Mutter, Kind]\}$
$V\ddot{a}ter : \{[Vater, Kind]\}$
$M\ddot{u}tter : \{[Mutter, Kind]\}$

Diese Zerlegung ist verlustlos, da sogar beide funktionalen Abhängigkeiten

- {Kind} $\rightarrow$ {Mutter}
- {Kind} $\rightarrow$ {Vater}

erfüllt sind.

Eltern		
Vater	Mutter	Kind
Johann	Martha	Else
Johann	Maria	Theo
Heinz	Martha	Cleo

$\Pi_{\text{Vater, Kind}}$

Väter	
Vater	Kind
Johann	Else
Johann	Theo
Heinz	Cleo

$\Pi_{\text{Mutter, Kind}}$

Mütter	
Mutter	Kind
Martha	Else
Maria	Theo
Martha	Cleo

Abhängigkeitsbewahrung

- $\mathcal{R}$ ist zerlegt in $\mathcal{R}_1$, ..., $\mathcal{R}_n$
- $F_{\mathcal{R}} \equiv (F_{\mathcal{R}_1} \cup \ldots \cup F_{\mathcal{R}_\mathrm{n}})$ bzw. $F_{\mathcal{R}}^+ = (F_{\mathcal{R}_1} \cup \ldots \cup F_{\mathcal{R}_n})^+$

Beispiel für Abhängigkeitsverlust

$$PLZverzeichnis : \{[\text{Straße}, \text{Ort}, \text{BLand}, \text{PLZ}]\}$$

- Orte werden durch ihren Namen (*Ort*) und das Bundesland (*BLand*) eindeutig identifiziert.
- Innerhalb einer Straße ändert sich die Postleitzahl nicht.
- Postleitzahlengebiete gehen nicht über Ortsgrenzen und Orte nicht über Bundeslandgrenzen hinweg.

- $\{\text{PLZ}\} \rightarrow \{\text{Ort, BLand}\}$
- $\{\text{Straße, Ort, BLand}\} \rightarrow \{\text{PLZ}\}$

$$\begin{aligned} \text{Straßen} &: \{[\text{PLZ}, \text{Straße}]\} \\ \text{Orte} &: \{[\text{PLZ}, \text{Ort}, \text{BLand}]\} \end{aligned}$$

Nicht abhängigkeitsbewahrende Zerlegung

PLZverzeichnis			
Ort	**BLand**	**Straße**	PLZ
Frankfurt	Hessen	Goethestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234

$\Pi_{\text{PLZ, Straße}}$ $\Pi_{\text{Stadt, BLand, PLZ}}$

Straßen	
PLZ	**Straße**
15234	Goethestraße
60313	Goethestraße
60437	Galgenstraße
15235	Goethestraße

Orte		
Ort	BLand	**PLZ**
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234
Frankfurt	Brandenburg	15235

- Die FD {Straße, Ort, BLand} $\rightarrow$ PLZ ist im zerlegten Schema nicht mehr „enthalten“
- Die beiden eingerahmten Tupel verletzen diese FD

Erste Normalform

- nur atomare Domänen

Eltern		
Vater	Mutter	Kinder
Johann	Martha	{Else, Lucia}
Johann	Maria	{Theo, Josef}
Heinz	Martha	{Cleo}

- 1 NF

Eltern		
Vater	Mutter	Kind
Johann	Martha	Else
Johann	Martha	Lucia
Johann	Maria	Theo
Johann	Maria	Josef
Heinz	Martha	Cleo

- NF^2-Relation

Eltern			
Vater	Mutter	Kinder	
		KName	KAlter
Johann	Martha	Else	5
		Lucia	3
Johann	Maria	Theo	3
		Josef	1
Heinz	Martha	Cleo	9

Zweite Normalform

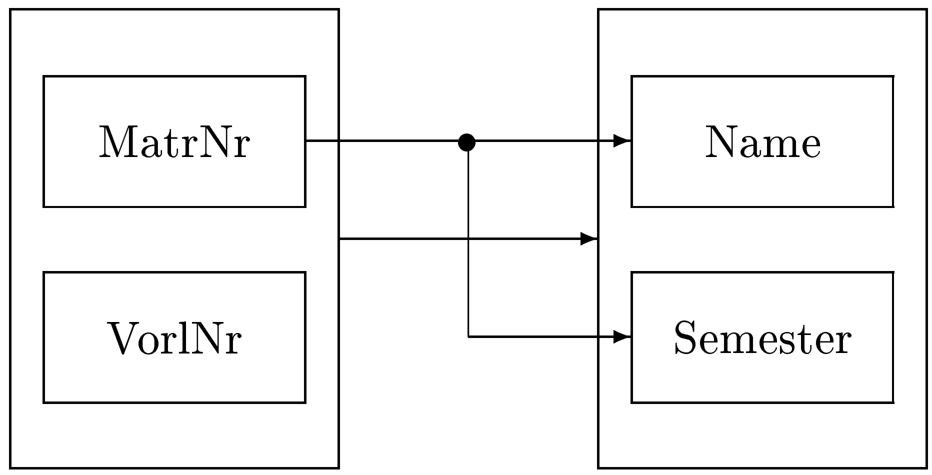
Eine Relation $\mathcal{R}$ mit zugehörigen FDs F ist in zweiter Normalform, falls jedes Nichtschlüssel-Attribut $A \in \mathcal{R}$ voll funktional abhängig ist von jedem Kandidatenschlüssel der Relation.

StudentenBelegung			
MatrNr	**VorlNr**	Name	Semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

Studentenbelegung ist nicht 2 NF wegen

- $\{\text{MatrNr}\} \rightarrow \{\text{Name}\}$ und
- $\{\text{MatrNr}\} \rightarrow \{\text{Semester}\}$

Zweite Normalform



- Einfügeanomalie: Was macht man mit Studenten, die keine Vorlesungen hören?

- Updateanomalien: Wenn z.B. „Carnap“ ins vierte Semester kommt, muß sichergestellt werden, daß alle vier Tupel geändert werden.

- Löschanomalien: Was passiert, wenn „Fichte“ ihre einzige Vorlesung absagt?

Zerlegung in:

- hören: {[MatrNr, VorlNr]} und

- Studenten: {[MatrNr, Name, Semester]}

Beide Relationen sind 2 NF (erfüllen sogar noch „höhere Gütekriterien“)

Dritte Normalform

Ein Relationenschema $\mathcal{R}$ ist in *dritter Normalform*, wenn für jede für $\mathcal{R}$ geltende funktionale Abhängigkeit der Form $\alpha \rightarrow B$ mit $\alpha \subseteq \mathcal{R}$ und $B \in \mathcal{R}$ mindestens *eine* von drei Bedingungen gilt:

- $B \in \alpha$, d.h. die FD ist trivial.
- Das Attribut B ist in einem Kandidatenschlüssel von $\mathcal{R}$ enthalten – also B ist *prim*.
- α ist Superschlüssel von $\mathcal{R}$.

Zerlegung mit dem Synthesealgorithmus

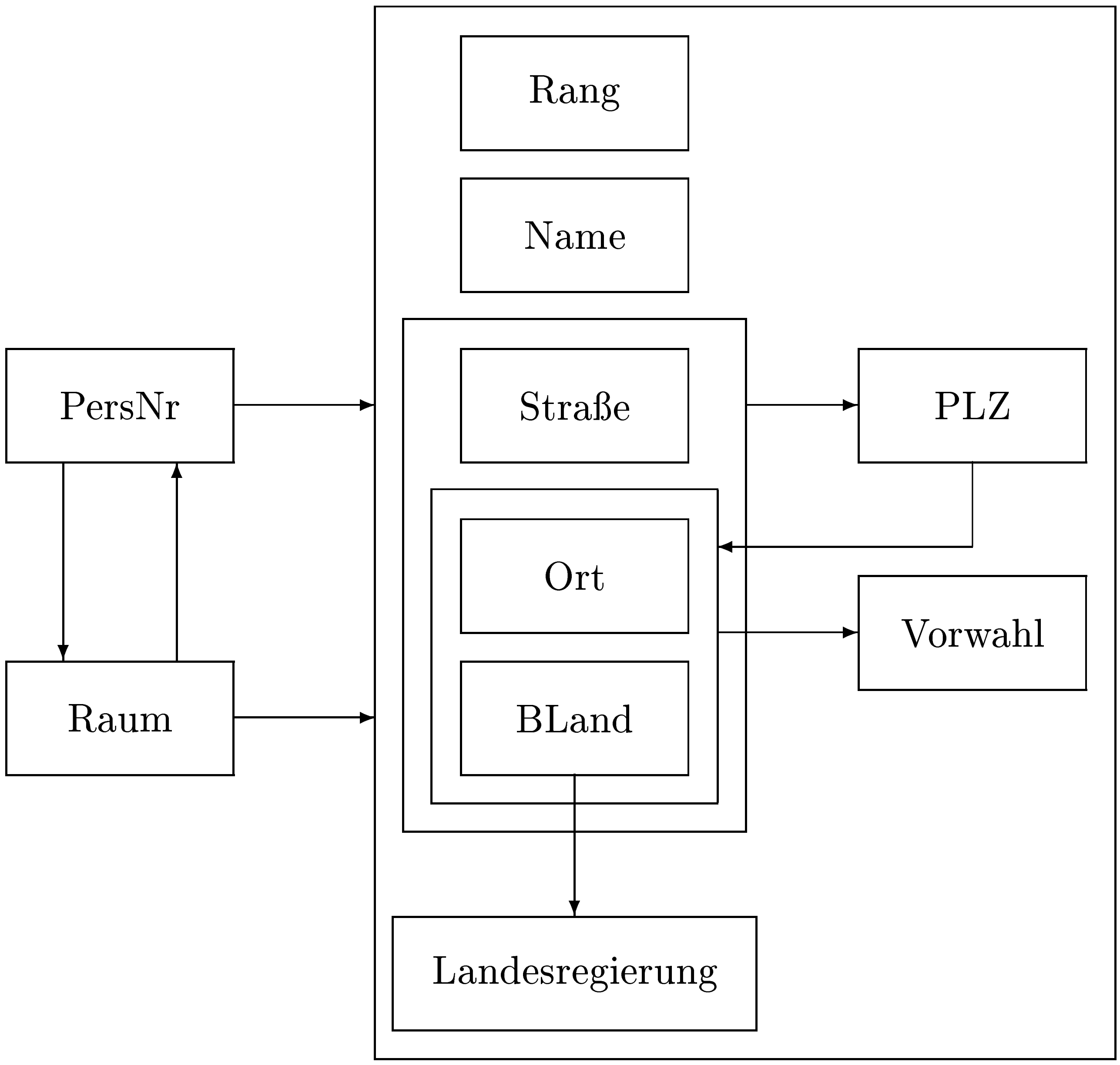
Wir geben jetzt einen sogenannten Synthesealgorithmus an, mit dem zu einem gegebenen Relationenschma $\mathcal{R}$ mit funktionalen Abhängigkeiten F eine Zerlegung in $\mathcal{R}_1, \ldots, \mathcal{R}_n$ ermittelt wird, die alle drei fogenden Kriterien erfüllt:

- $\mathcal{R}_1, \ldots, \mathcal{R}_n$ ist eine verlustlose Zerlegung von $\mathcal{R}$.
- Die Zerlegung ist abhängigkeitsbewahrend
- Alle $\mathcal{R}_i$ $(1 \leq i \leq n)$ sind in dritter Normalform.

Synthesealgorithmus

1. Bestimme die kanonische Überdeckung F_c zu F. Zur Wiederholung:
   (a) Linksreduktion
   (b) Rechtsreduktion
   (c) Entfernung von FDs der Form $\alpha \to \emptyset$
   (d) Zusammenfassung gleicher linker Seiten
2. Für jede funktionale Abhängigkeit $\alpha \to \beta \in F_c$:
   - Kreiere ein Relationenschema $\mathcal{R}_\alpha := \alpha \cup \beta$.
   - Ordne $\mathcal{R}_\alpha$ die FDs $F_\alpha := \{\alpha' \to \beta' \in F_c \mid \alpha' \cup \beta' \subseteq \mathcal{R}_\alpha\}$ zu.
3. Falls eines der in Schritt 2. erzeugten Schemata $\mathcal{R}_\alpha$ einen Kandidatenschlüssel von $\mathcal{R}$ bzgl. F_c enthält, sind wir fertig; sonst wähle einen Kandidatenschlüssel $\kappa \subseteq \mathcal{R}$ aus und definiere folgendes zusätzliche Schema:
   - $\mathcal{R}_\kappa := \kappa$
   - $F_\kappa := \emptyset$
4. Eliminiere diejenigen Schemata $\mathcal{R}_\alpha$, die in einem anderen Relationenschema $\mathcal{R}_{\alpha'}$ enthalten sind, d.h.
   - $\mathcal{R}_\alpha \subseteq \mathcal{R}_{\alpha'}$

Anwendung des Synthesealgorithmus'



Anwendung des Synthesealgorithmus' (Forts.)

ProfessorenAdr : {[PersNr, Name, Rang, Raum, Straße, Ort, BLand, EW, PLZ, Vorwahl, Landesregierung]}

$$
\begin{aligned}
fd_1 &: \{\text{PersNr}\} \rightarrow \{\text{Raum}, \text{Name}, \text{Rang}, \text{Straße}, \text{Ort}, \text{BLand}\} \\
fd_2 &: \{\text{Raum}\} \rightarrow \{\text{PersNr}\} \\
fd_3 &: \{\text{Straße}, \text{Ort}, \text{BLand}\} \rightarrow \{\text{PLZ}\} \\
fd_4 &: \{\text{Ort}, \text{BLand}\} \rightarrow \{\text{Vorwahl}\} \\
fd_5 &: \{\text{BLand}\} \rightarrow \{\text{Landesregierung}\} \\
fd_6 &: \{\text{PLZ}\} \rightarrow \{\text{Ort}, \text{BLand}\}
\end{aligned}
$$

- Professoren:{[PersNr, Name, Rang, Raum, Straße, Ort, BLand}
- PLZverzeichnis:{[Straße, Ort, BLand, PLZ]}
- Vorwahlverzeichnis:{[Ort, BLand, Vorwahl]}
- Regierungen:{[BLand, Landesregierung]}

Boyce-Codd Normalform

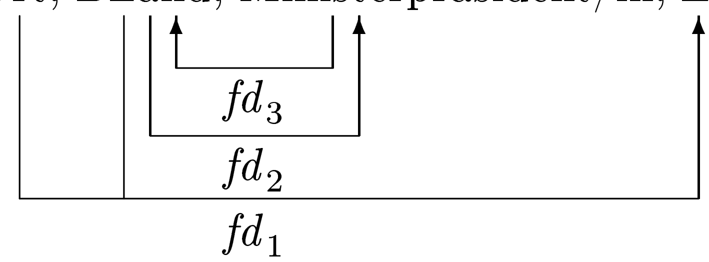
Die Boyce-Codd Normalform (BCNF) stellt nochmals eine Verschärfung dar. Ein Relationenschema $\mathcal{R}$ mit FDs F ist in BCNF, falls für jede funktionale Abhängigkeit $\alpha \rightarrow \beta \in F$ gilt:

- $\beta \subseteq \alpha$, d.h. die Abhängigkeit ist trivial oder
- α ist Superschüssel von $\mathcal{R}$.

- Man kann jede Relation **verlustlos** in BCNF-Relationen zerlegen.
- Manchmal gehen dabei aber Abhängigkeiten verloren.

Städte ist in 3 NF, aber nicht BCNF

Städte : {[Ort, BLand, Ministerpräsident/in, EW]}



- $\kappa_1 = \{\text{Ort}, \text{BLand}\}$
- $\kappa_2 = \{\text{Ort}, \text{Ministerpräsident}\}$

Man kann grundsätzlich jedes Relationenschema $\mathcal{R}$ mit zugeordneten FDs F so in $\mathcal{R}_1$, ..., $\mathcal{R}_n$ zerlegen , daß gilt:

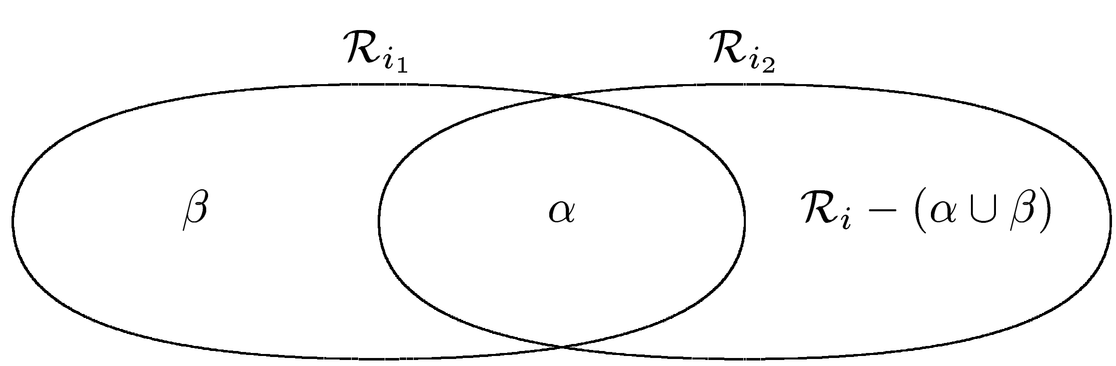
- Die Zerlegung ist verlustlos und
- die $\mathcal{R}_i$ $(1 \leq i \leq n)$ sind alle in BCNF.

Dekompositionsalgorithmus

- Starte mit $Z = \{\mathcal{R}\}$
- Solange es noch ein Relationenschema $\mathcal{R}_i \in Z$ gibt, das nicht in BCNF ist, mache folgendes:
  - Es gibt also eine für $\mathcal{R}_i$ geltende nicht-triviale FD $(\alpha \to \beta)$ mit
    - $\alpha \cap \beta = \emptyset$
    - $\alpha \not\to \mathcal{R}_i$

    Finde eine solche FD – man sollte sie so wählen, daß β alle von α funktional abhängigen Attribute $B \in (\mathcal{R}_i - \alpha)$ enthält, damit der Dekompositionsalgorithmus möglichst schnell terminiert.
  - 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

$$Z := (Z - \{\mathcal{R}_i\}) \cup \{\mathcal{R}_{i_1}\} \cup \{\mathcal{R}_{i_2}\}$$



Dekomposition in BCNF

Zerlegung der Relation *Städte*:

- $\underbrace{\text{Regierungen}}_{\mathcal{R}_{i_1}} : \{\text{BLand}, \text{Ministerpräsident}\}$
- $\underbrace{\text{Städte}}_{\mathcal{R}_{i_2}} : \{\text{Ort}, \text{BLand}, \text{EW}\}$

Zerlegung ist verlustlos und abhängigkeitserhaltend.

Dekomposition in BCNF (Forts.)

fd_1

$$\text{PLZverzeichnis} : \{[\text{Straße, Ort, BLand, PLZ}]\}$$

fd_2

- Straßen : $\{[\text{Straße}, \text{PLZ}]\}$
- Orte : $\{[\text{Ort}, \text{BLand}, \text{PLZ}]\}$
- Diese Zerlegung ist
- verlustlos aber
- nicht abhängigkeitserhaltend
- (siehe oben).

Mehrwertige Abhängigkeiten

Seien α, β, $\gamma \subseteq \mathcal{R}$, so daß $\mathcal{R} = \alpha \cup \beta \cup \gamma$. Dann ist β mehrwertig abhängig von α – in Zeichen $\alpha \twoheadrightarrow \beta$ – wenn in jeder gültigen Ausprägung von $\mathcal{R}$ gilt: Für jedes Paar von Tupeln t_1 und t_2 mit $t_1.\alpha = t_2.\alpha$ existieren zwei weitere Tupel t_3 und t_4 mit folgenden Eigenschaften:

$$\begin{array}{ccccccc}
t_1.\alpha & = & t_2.\alpha & = & t_3.\alpha & = & t_4.\alpha \\
 & & t_3.\beta & = & t_1.\beta & & \\
 & & t_3.\gamma & = & t_2.\gamma & & \\
 & & t_4.\beta & = & t_2.\beta & & \\
 & & t_4.\gamma & = & t_1.\gamma & &
\end{array}$$

	R		
	$\overbrace{A_1 \ldots A_i}^{\alpha}$	$\overbrace{A_{i+1} \ldots A_j}^{\beta}$	$\overbrace{A_{j+1} \ldots A_n}^{\gamma}$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$
t_4	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$

Jede FD ist auch eine MVD!

Mehrwertige Abhängigkeiten

Fähigkeiten		
PersNr	Sprache	ProgSprache
3002	griechisch	C
3002	lateinisch	Pascal
3002	griechisch	Pascal
3002	lateinisch	C
3005	deutsch	Ada

$$\{PersNr\} \twoheadrightarrow \{Sprache\} \text{ und } \{PersNr\} \twoheadrightarrow \{ProgrSprache\}$$

Sprachen	
PersNr	Sprache
3002	griechisch
3002	lateinisch
3005	deutsch

ProgrSprachen	
PersNr	ProgrSprache
3002	C
3002	Pascal
3005	Ada

$$\text{Fähigkeiten} \quad = \quad \underbrace{\Pi_{\text{PersNr,Sprache}}(\text{Fähigkeiten})}_{\text{Sprachen}} \bowtie \underbrace{\Pi_{\text{PersNr,ProgrSprache}}(\text{Fähigkeiten})}_{\text{ProgrSprachen}}$$

Verlustlose Zerlegung bei mehrwertiger Abhängigkeit

Ein Relationenschema $\mathcal{R}$ mit einer Menge D von zugeordneten funktionalen und mehrwertigen Abhängigkeiten kann **genau dann** verlustlos in die beiden Schemata $\mathcal{R}_1$ und $\mathcal{R}_2$ zerlegt werden, wenn gilt:

- $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ und
- mindestens eine von zwei MVDs gilt:
  1. $\mathcal{R}_1 \cap \mathcal{R}_2 \twoheadrightarrow \mathcal{R}_1$ oder
  2. $\mathcal{R}_1 \cap \mathcal{R}_2 \twoheadrightarrow \mathcal{R}_2$.

Inferenzregeln für MVDs

- *Reflexivität*: $\beta \subseteq \alpha \Rightarrow \alpha \to \beta$
- *Verstärkung*: Sei $\alpha \to \beta$. Dann gilt $\gamma\alpha \to \gamma\beta$.
- *Transitivität*: Sei $\alpha \to \beta$ und $\beta \to \gamma$. Dann gilt $\alpha \to \gamma$.
- *Komplement*: $\alpha \twoheadrightarrow \beta$. Dann gilt $\alpha \twoheadrightarrow \mathcal{R} - \beta - \alpha$.
- *Mehrwertige Verstärkung*: Sei $\alpha \twoheadrightarrow \beta$ und $\delta \subseteq \gamma$. Dann gilt $\gamma\alpha \twoheadrightarrow \delta\beta$.
- *Mehrwertige Transitivität*: Sei $\alpha \twoheadrightarrow \beta$ und $\beta \twoheadrightarrow \gamma$. Dann gilt $\alpha \twoheadrightarrow \gamma - \beta$.
- *Verallgemeinerung*: Sei $\alpha \to \beta$. Dann gilt $\alpha \twoheadrightarrow \beta$.
- *Koaleszenz*: Sei $\alpha \twoheadrightarrow \beta$ und $\gamma \subseteq \beta$. Existiert ein $\delta \subseteq \mathcal{R}$, so daß $\delta \cap \beta = \emptyset$ und $\delta \to \gamma$, gilt $\alpha \to \gamma$.
- *Mehrwertige Vereinigung*: sei $\alpha \twoheadrightarrow \beta$ und $\alpha \twoheadrightarrow \gamma$. Dann gilt $\alpha \twoheadrightarrow \gamma\beta$.
- *Schnittmenge*: Sei $\alpha \twoheadrightarrow \beta$ und $\alpha \twoheadrightarrow \gamma$. Dann gilt $\alpha \twoheadrightarrow \beta \cap \gamma$.
- *Differenz*: Sei $\alpha \twoheadrightarrow \beta$ und $\alpha \twoheadrightarrow \gamma$. Dann gilt $\alpha \twoheadrightarrow \beta - \gamma$ und $\alpha \twoheadrightarrow \gamma - \beta$.

Vierte Normalform

Triviale MVD

Eine MVD $\alpha \twoheadrightarrow \beta$ bezogen auf $\mathcal{R} \supseteq \alpha \cup \beta$ ist trivial, wenn *jede* mögliche Ausprägung R von $\mathcal{R}$ diese MVD erfüllt. Man kann zeigen, daß $\alpha \twoheadrightarrow \beta$ trivial ist, genau dann wenn

1. $\beta \subseteq \alpha$ oder
2. $\beta = \mathcal{R} - \alpha$

Definition 4NF

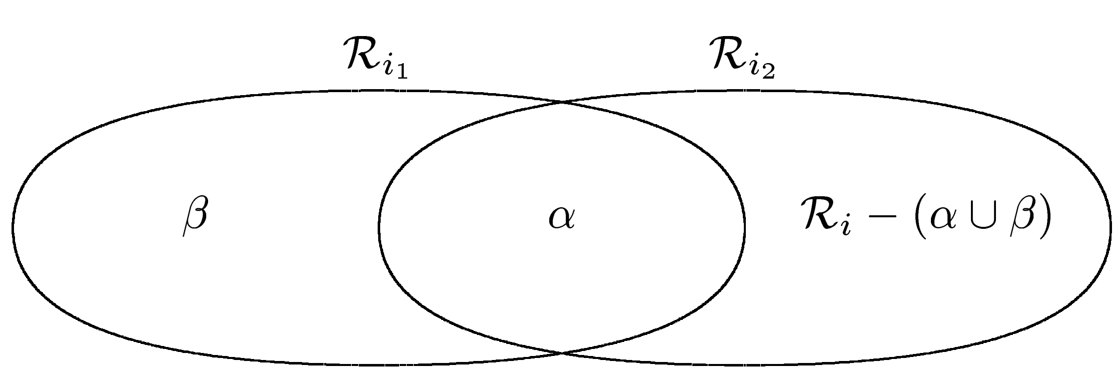
Eine Relation $\mathcal{R}$ mit zugeordneter Menge D von funktionalen und mehrwertigen Abhängigkeiten ist in 4NF wenn für jede MVD $\alpha \twoheadrightarrow \beta \in D^+$ eine der folgenden Bedingungen gilt:

1. Die MVD ist trivial oder
2. α ist ein Superschlüssel von $\mathcal{R}$.

Dekomposition in 4NF

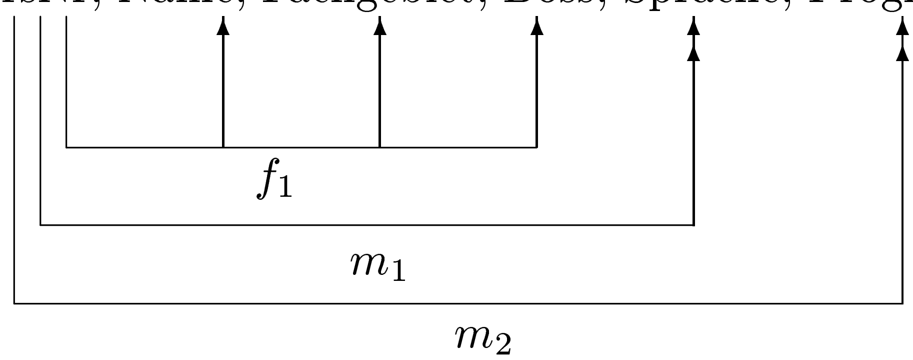
- Starte mit der Menge $Z := \{\mathcal{R}\}$,
- Solange es eine Relation $\mathcal{R}_i \in Z$ gibt, die nicht in 4NF ist, mache folgendes:
  - finde eine für $\mathcal{R}_i$ geltende nicht-triviale MVD $\alpha \twoheadrightarrow \beta$, für die gilt
    * $\alpha \cap \beta = \emptyset$
    * $\alpha \not\rightarrow \mathcal{R}_i$
  - 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

$$Z := (Z - \{\mathcal{R}_1\}) \cup \{\mathcal{R}_{i_1}\} \cup \{\mathcal{R}_{i_2}\}.$$



Beispielzerlegung

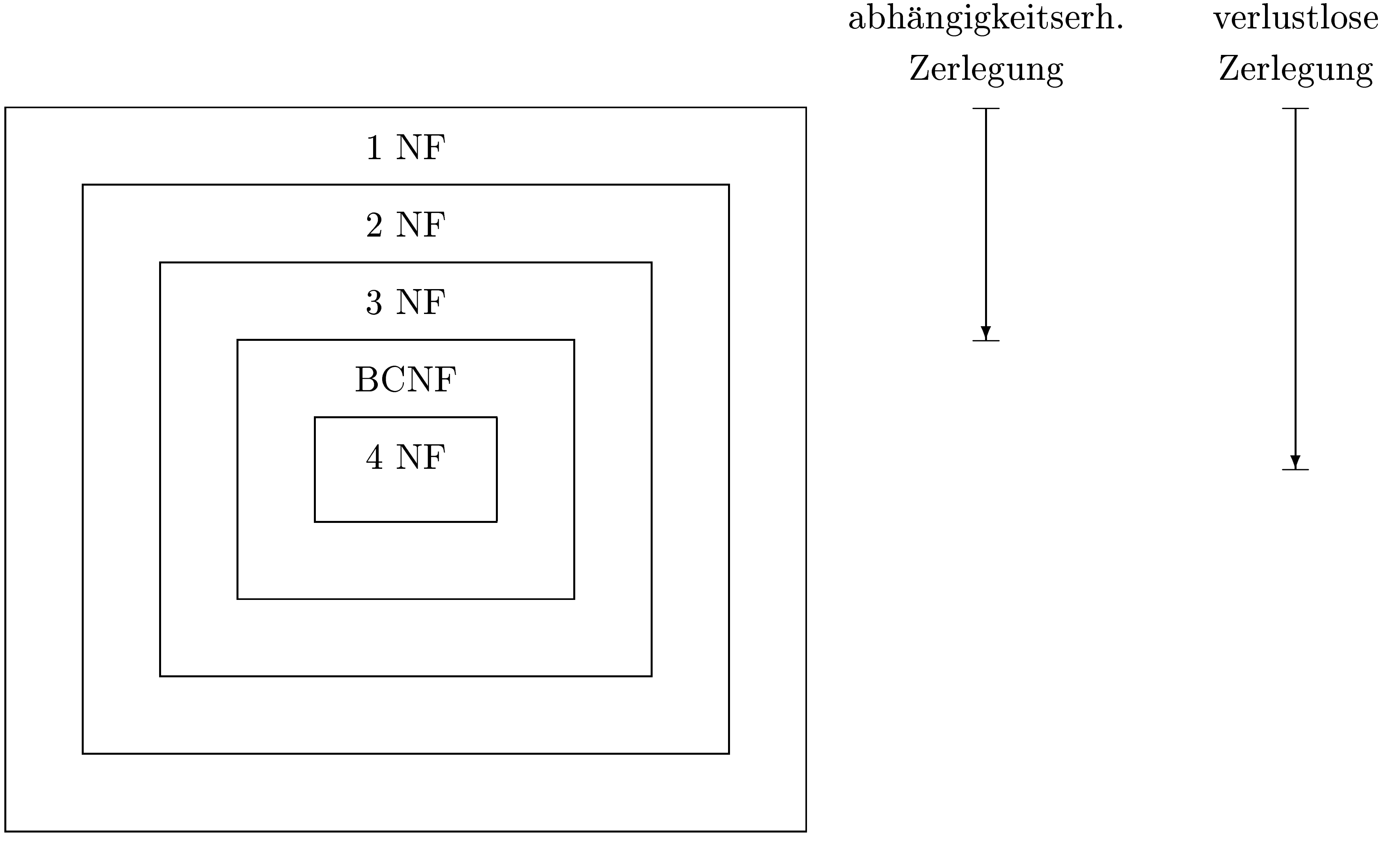
Assistenten′: {[PersNr, Name, Fachgebiet, Boss, Sprache, ProgrSprache]}



- Assistenten: {[PersNr, Name, Fachgebiet, Boss]}
- Fähigkeiten: {[PersNr, Sprache, ProgrSprache]}
- Sprachen: {[PersNr, Sprache]}
- ProgrSprachen: {[PersNr, ProgrSprache]}

Zusammenfassung

- Die Verlustlosigkeit ist für alle Zerlegungsalgorithmen in alle Normalformen garantiert.
- Die Abhängigkeitserhaltung kann nur bei den Zerlegungen bis zur dritten Normalform garantiert werden.



Übung: FDs, MVDs, Normalisierung

Vorlesungen: {[VorlNr, Titel, SWS, gelesenVon, VTermin,VRaum, ÜTermin, ÜRaum]}

- FDs
  - VorlNr $\rightarrow$ Titel, SWS, gelesenVon
  - VRaum, VTermin $\rightarrow$ VorlNr
  - VorlNr, VTermin $\rightarrow$ VRaum
  - ÜTermin, ÜRaum $\rightarrow$ VorlNr
- Schlüssel: {VTermin,ÜTermin, ÜRaum}

- MVDs
  - VorlNr $\rightarrow\!\!\!\rightarrow$ VTermin, VRaum
  - VorlNr $\rightarrow\!\!\!\rightarrow$ ÜTermin, ÜRaum

Normalisierung: 3NF, BCNF, 4NF

Physische Datenorganisation

Hintergrundspeicherung:

- Aufbau eines Plattenspeichers
- Speicherarrays (RAID)

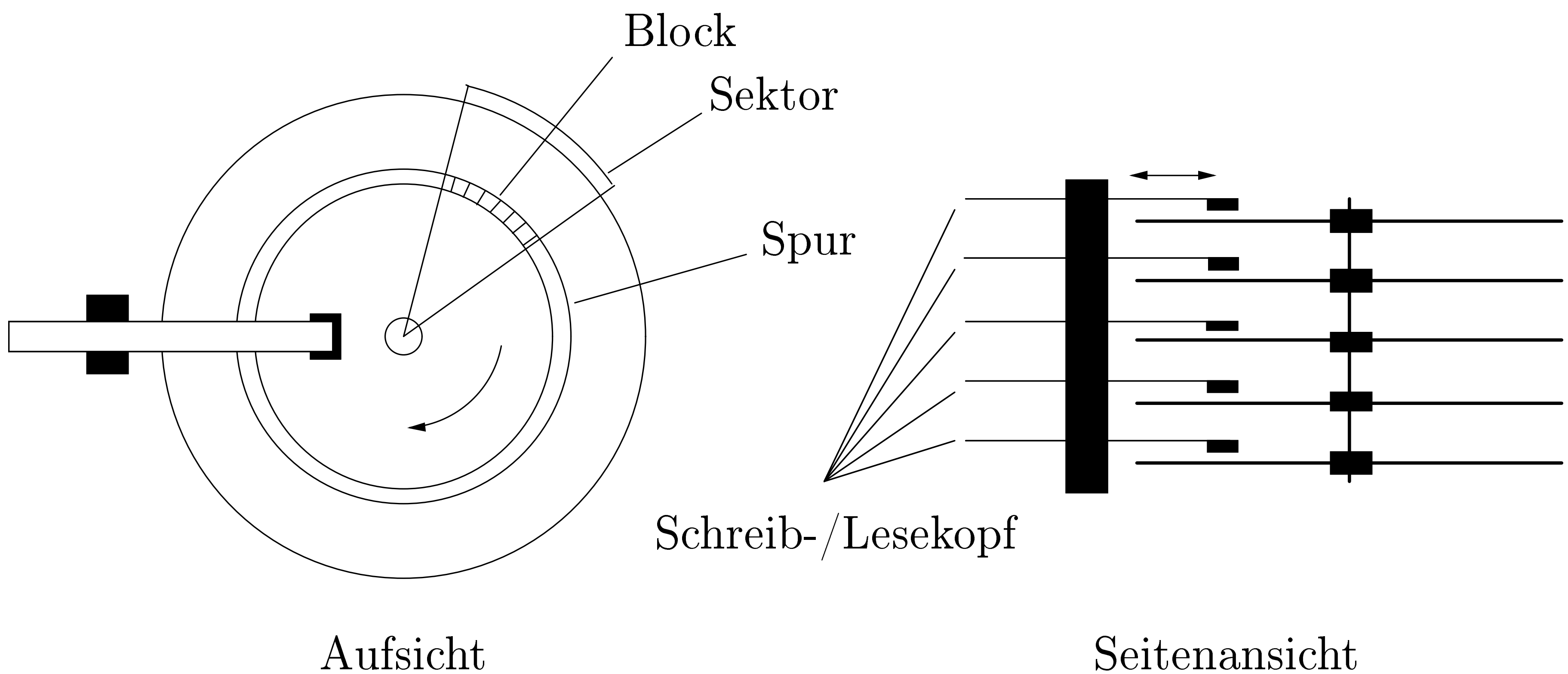
Indexstrukturen:

- Ausnutzung des Direktzugriffs des Speichermediums
- ISAM
- B-Bäume
- Hashing

Ballung:

- Ausnutzung der Plazierung von Objekten auf Seiten

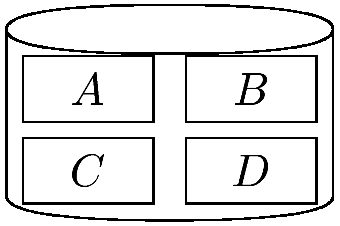
Schematischer Aufbau einer Festplatte



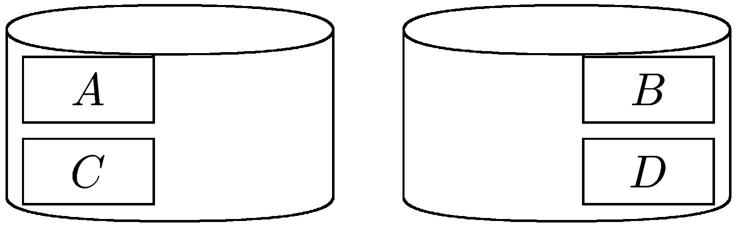
Lesen eines Blocks:

1. Positionierung des Kopfes (Seek-Time)
2. Rotation zum Anfang des Blocks (Latenzzeit)
3. Lesen des Blocks (Lesezeit)

virtuelle/logische Platte (hier mit vier Datenblöcken)

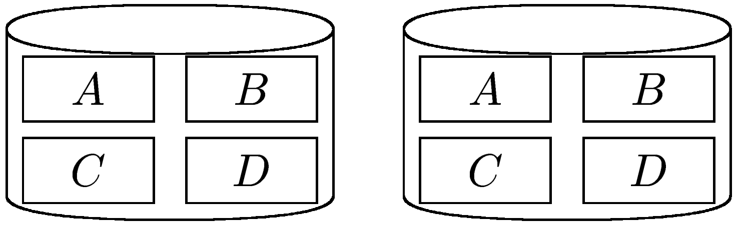


RAID 0: Striping der Blöcke (hier auf nur zwei Platten)



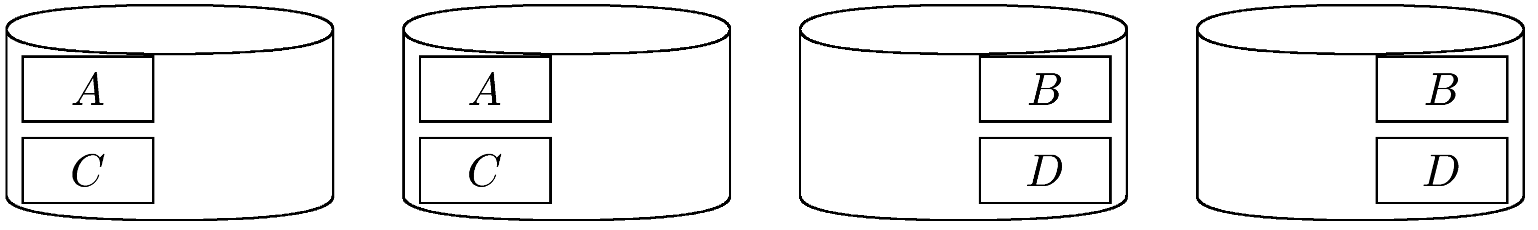
- In **RAID 0** wird die Datenmenge des logischen Laufwerks durch blockweise Rotation auf die physischen Laufwerke verteilt
- Dieses Vorgehen nennt man *Striping*
- Größe der Datenblöcke nennt man die *Stripinggranularität*, Anzahl der Platten *Stripingbreite*

RAID 1: Spiegelung (mirroring)



- **RAID 1** berücksichtigt auch die Datensicherheit
- Jedes Laufwerk besitzt eine sogenannte Spiegelkopie (engl. *mirror*)
- paralleles Schreiben der Spiegelkopien

RAID 0+1: Striping und Spiegelung

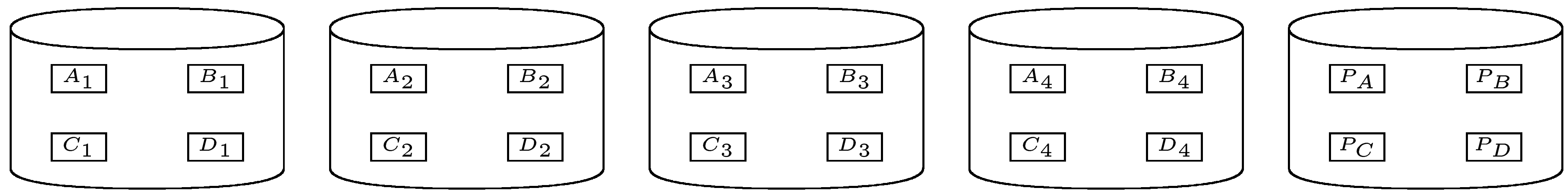


- **RAID 0+1** kombiniert einfach RAID 0 und RAID 1
- RAID 1 und RAID 0+1 verursachen einen doppelten Speicherplatzbedarf

RAID 2

- **RAID 2** führt ein Striping auf Bitebene durch
- zusätzliche Platten zur Speicherung von Paritätsinformationen
- Fehlererkennungs- und Korrekturcodes

RAID 3: Bit-Level-Striping + separate Parity-Platte

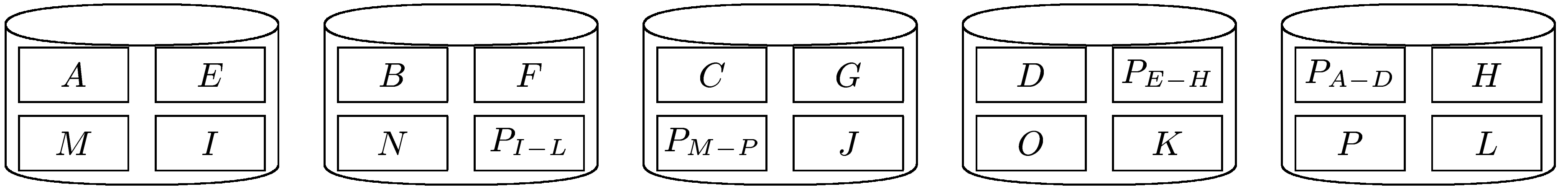


- **RAID 3** und **RAID 4** verwenden für die Paritätsinformationen eine einzige, dedizierte Festplatte
- In RAID 3 werden die Daten bit- oder byteweise auf die Datenplatten verteilt
- A_1 enthält die Bits/Bytes $A[1], A[5], \ldots$; A_2 die Bits/Bytes $A[2], A[6], \ldots$ usw.
  Bit $A[i]$ wird auf i Platte mod 4 plaziert.
- Paritätsinformation wird wie folgt berechnet ($A[i]$ ist das i-te Bit/Byte des Blocks A):

$$\overbrace{A[1] \oplus A[2] \oplus A[3] \oplus A[4]}^{P_A[1]} \; , \; \overbrace{A[5] \oplus A[6] \oplus A[7] \oplus A[8]}^{P_A[2]} \; , \; \ldots$$

- Bei RAID 3 muß eine Leseanforderung auf alle Datenplatten zugreifen
- RAID 4 verteilt die Daten wieder blockweise auf die Platten

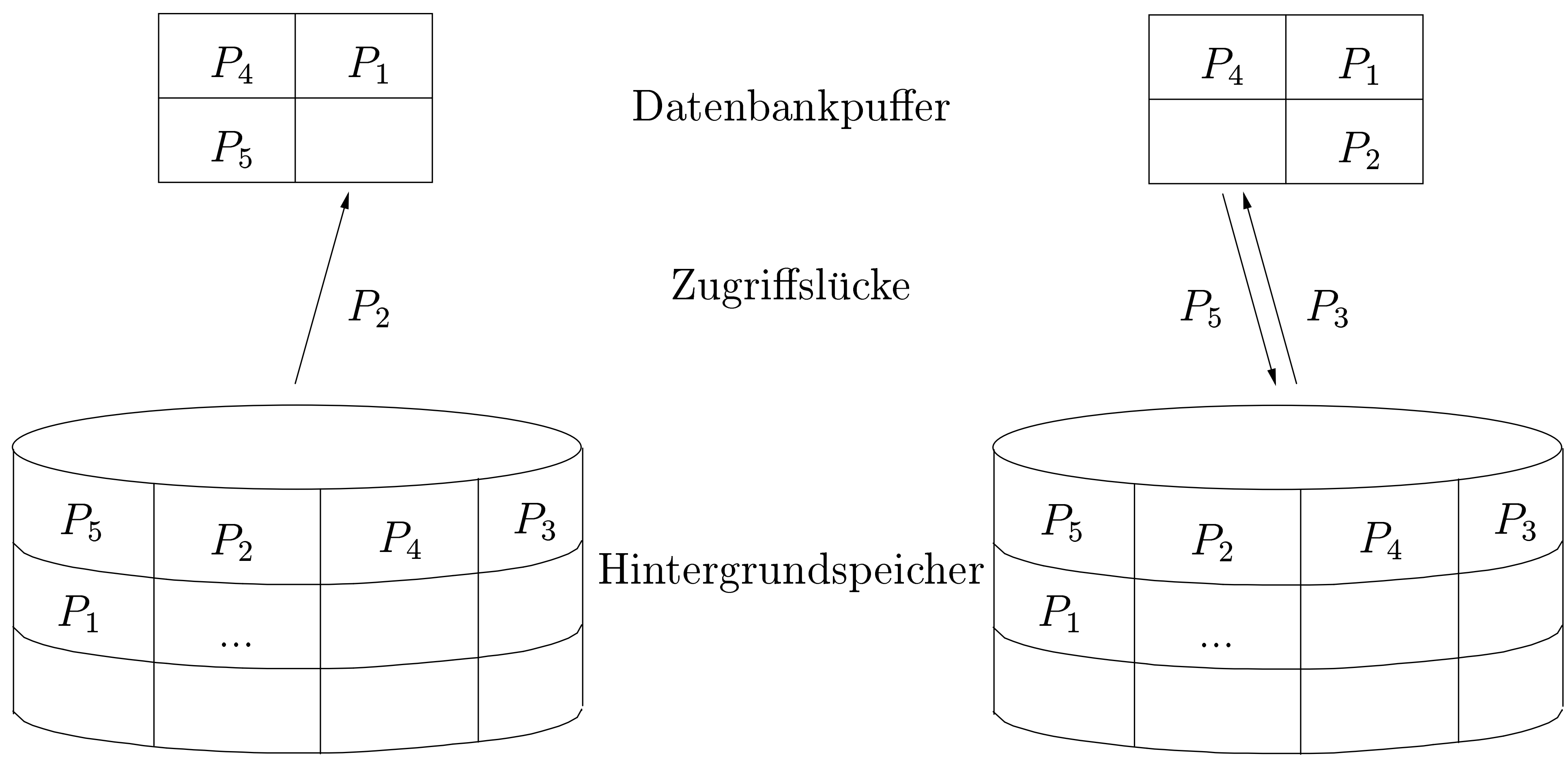
RAID 5: Block-Level-Striping + verteilte Parity-Blöcke



- **RAID 5** arbeitet ähnlich wie RAID 4, verteilt jedoch die Paritätsinformationen auf alle Laufwerke
- Nach wie vor ist aber der Overhead von Schreiboperationen nicht zu vernachlässigen

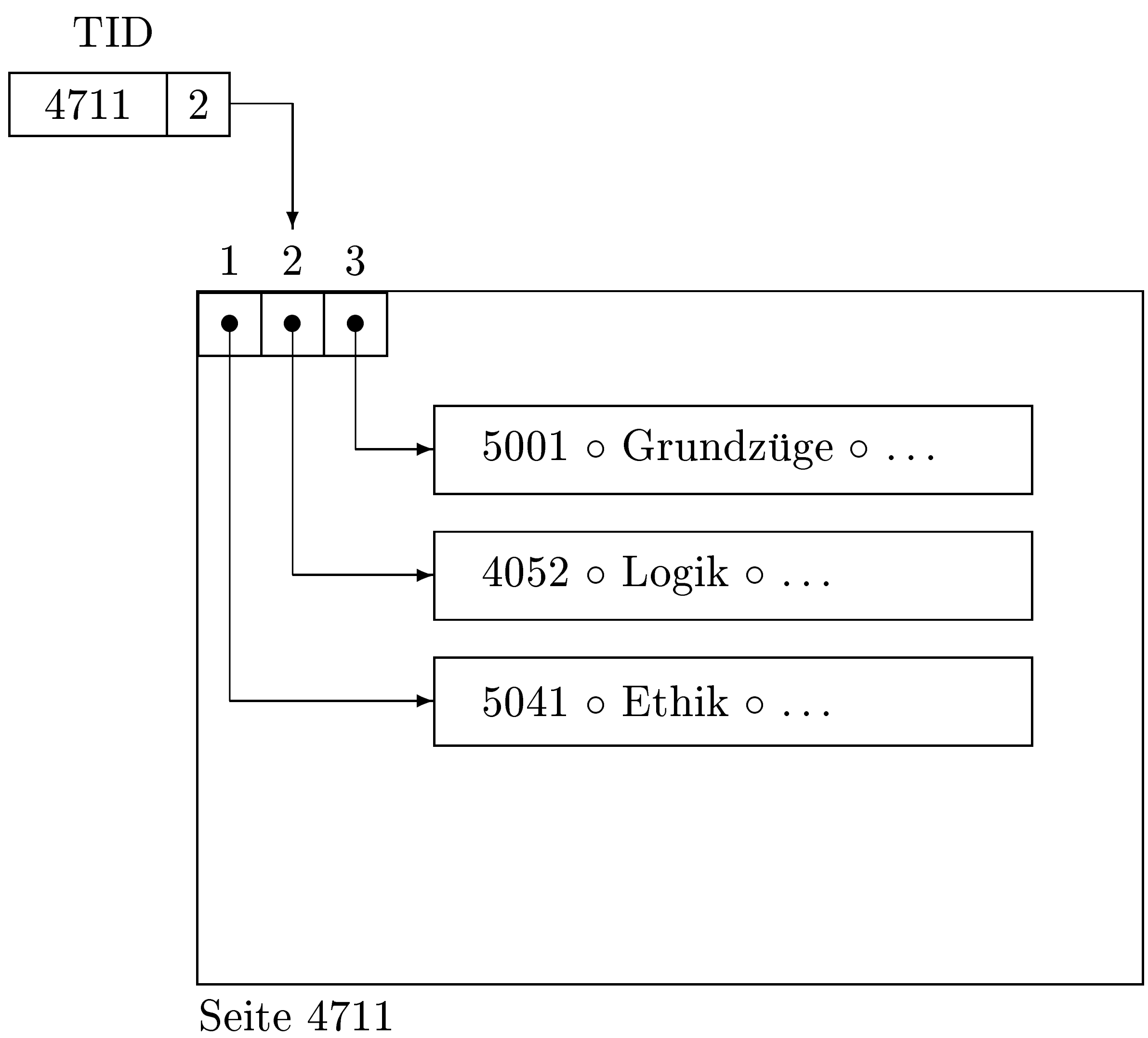
Warnung

- Trotz der Fehlertoleranz von RAID-Systemen, seien die Leser eindringlich davor gewarnt, die systematische Archivierung und Protokollierung von Datenbankzuständen für die Fehlerrecovery – wie sie in Kapitel **10** behandelt wird – zu vernachlässigen

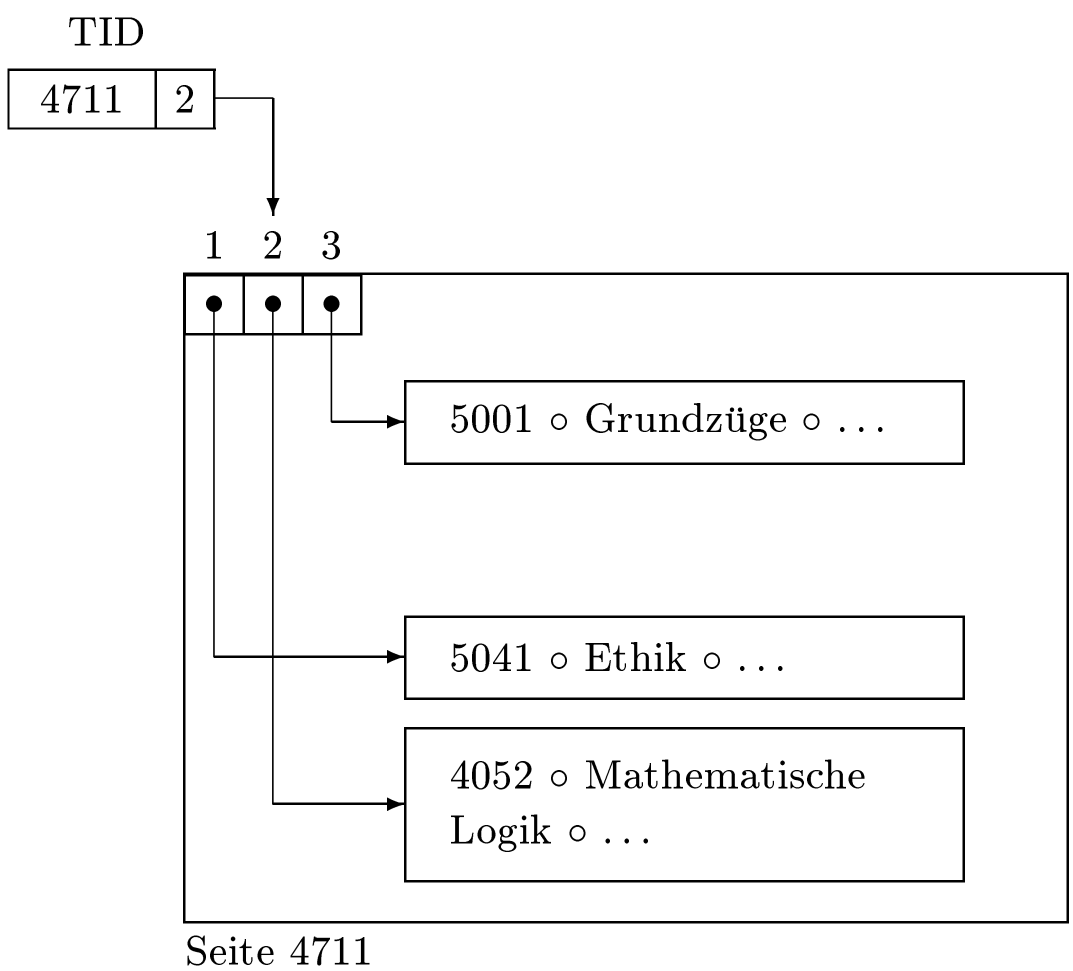


(a) Laden von P_2 (b) Ersetzen von P_5

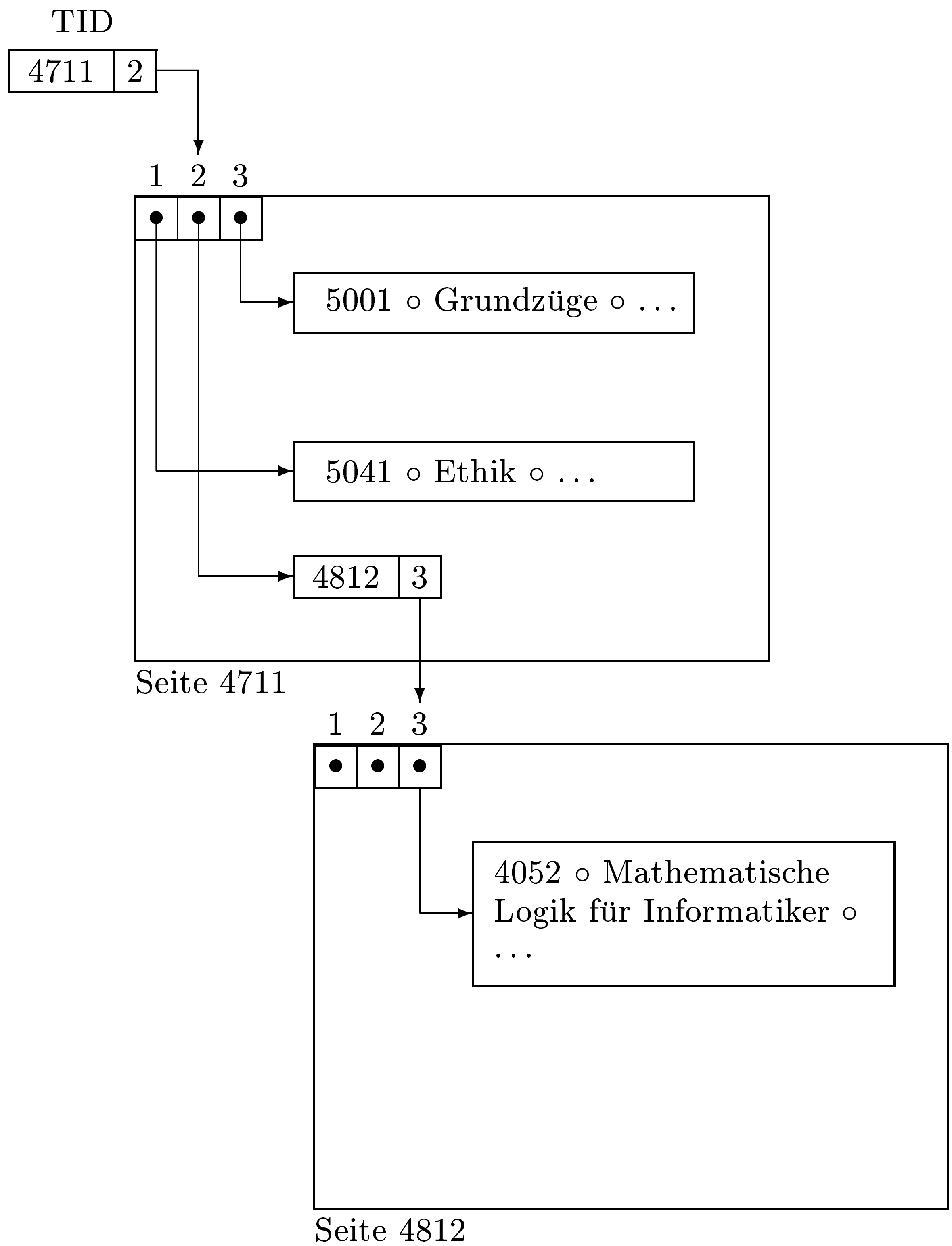
Das TID-Konzept



Verschieben von Tupeln innerhalb einer Seite

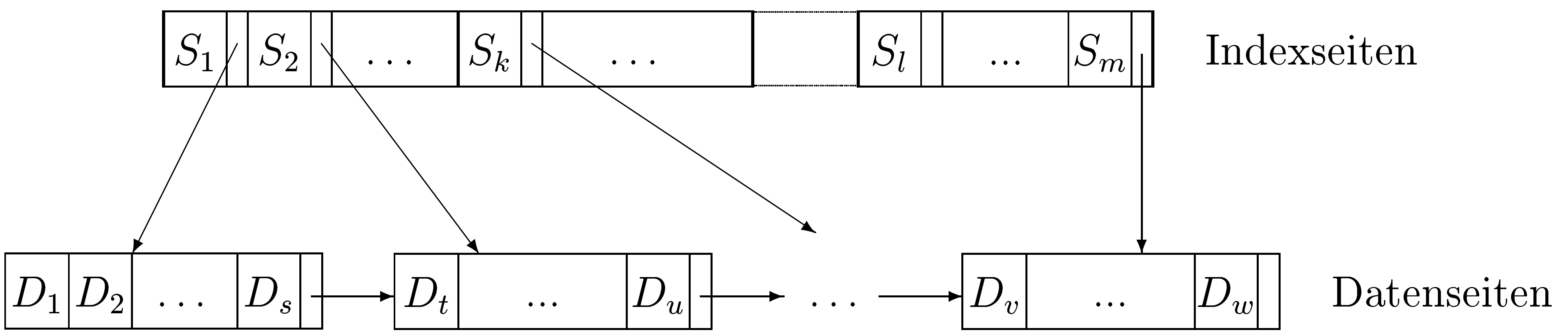


Verdrängung eines Tupels von einer Seite



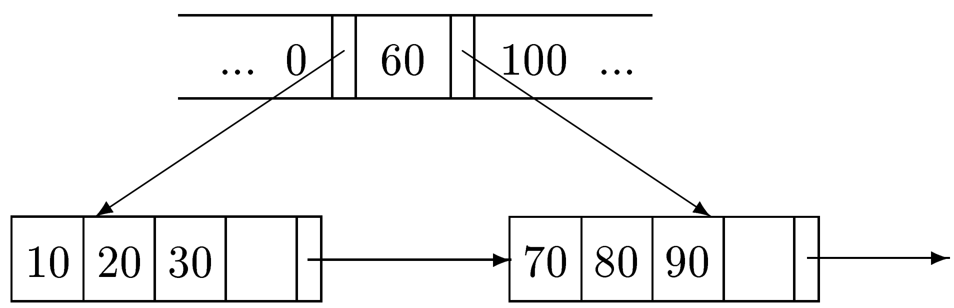
- Bei nochmaliger Verdrängung wird der „Vorwärtsverweis“ (Engl. *forward*) auf Seite 4711 geändert – also keine längeren Verweisketten.

Index-Sequential Access Method (ISAM)

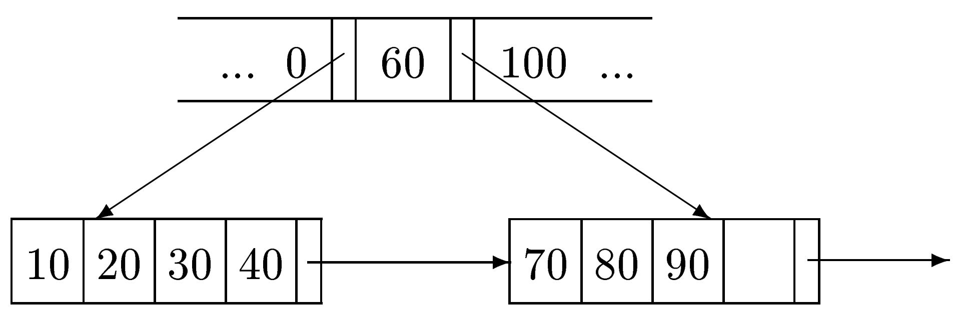


- sequentiell abgespeicherte Indexseite
- binäre Suche im Index

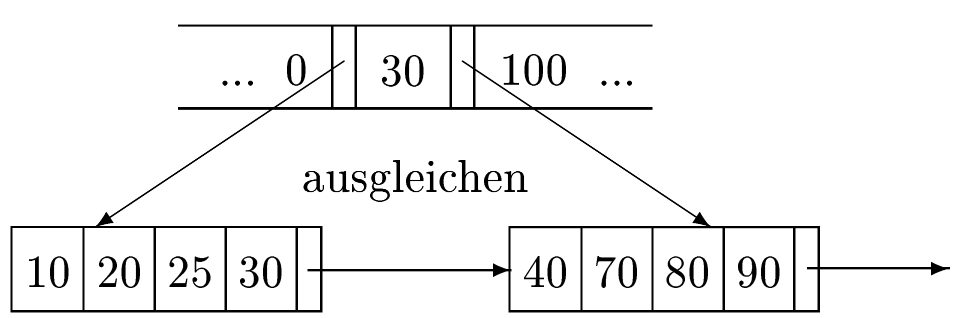
Einfügen in eine ISAM-Indexstruktur



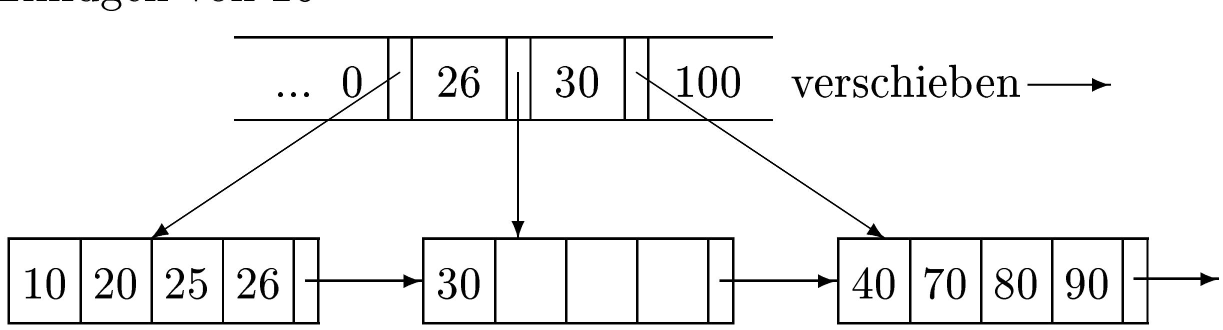
a) Einfügen von 40



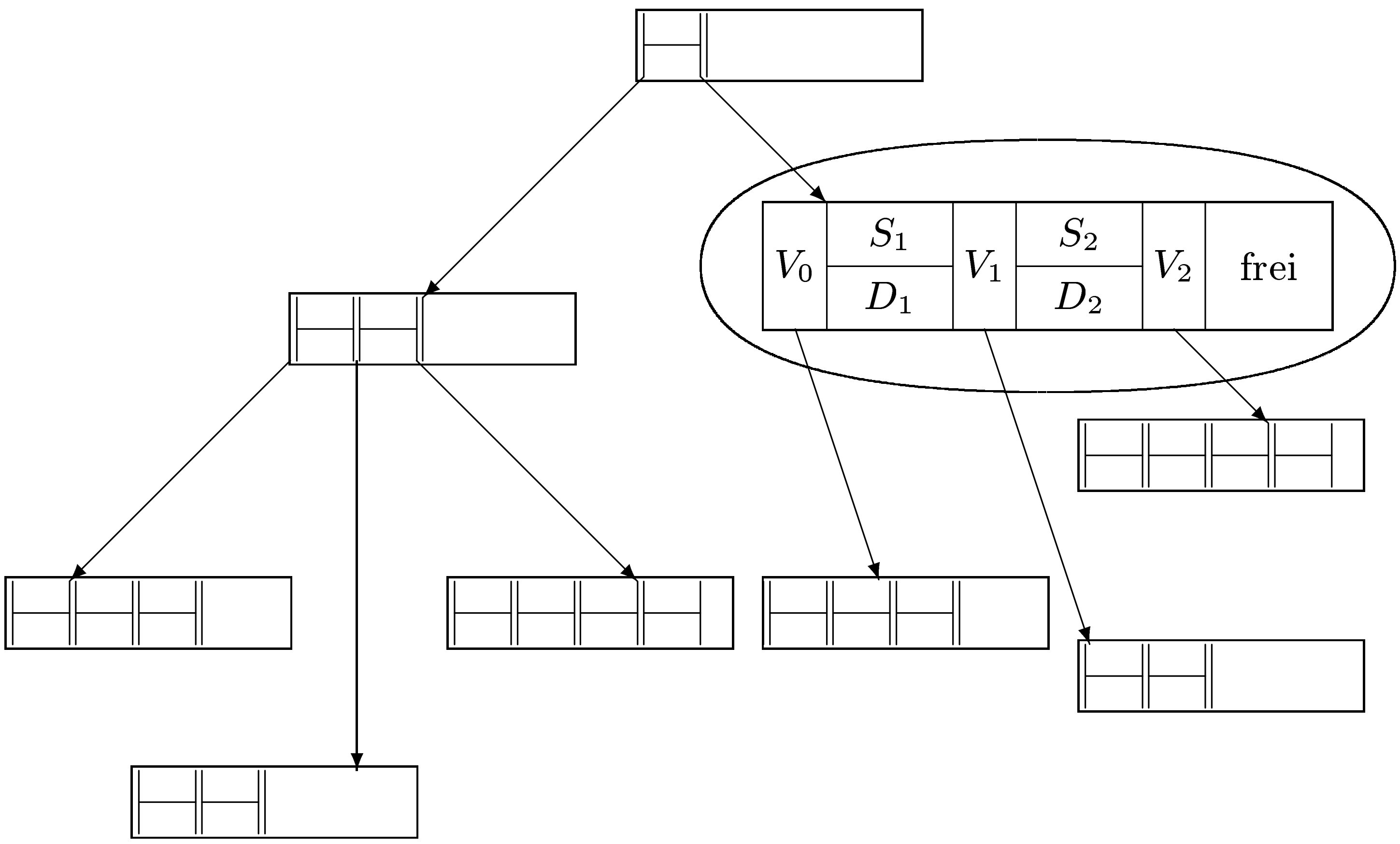
b) Einfügen von 25



c) Einfügen von 26



B-Bäume



- ein Knoten entspricht einer Seite
- balanciert
- garantierte Auslastung $\geq 50\%$

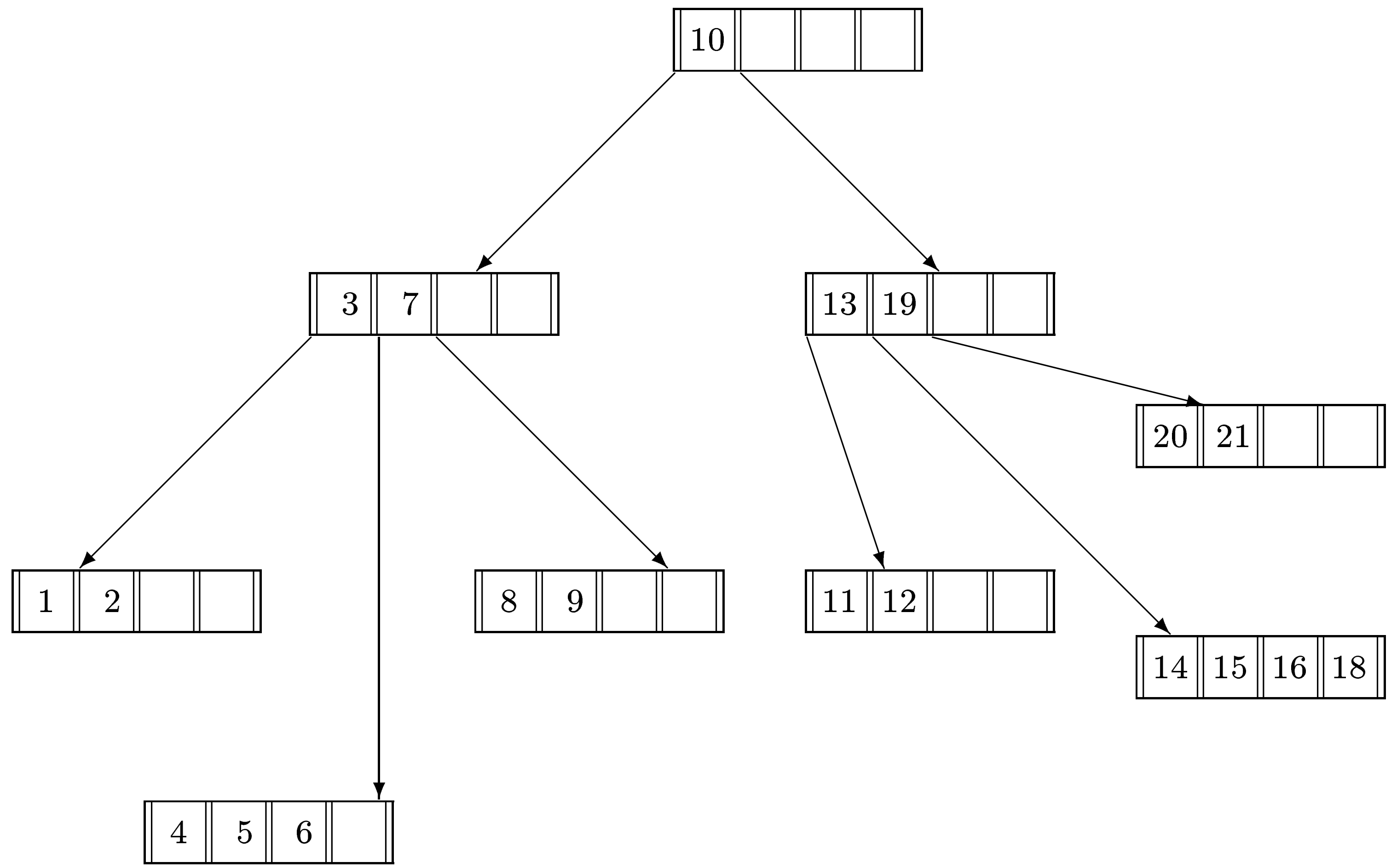
Eigenschaften eines B-Baums

B-Baum von Grad k:

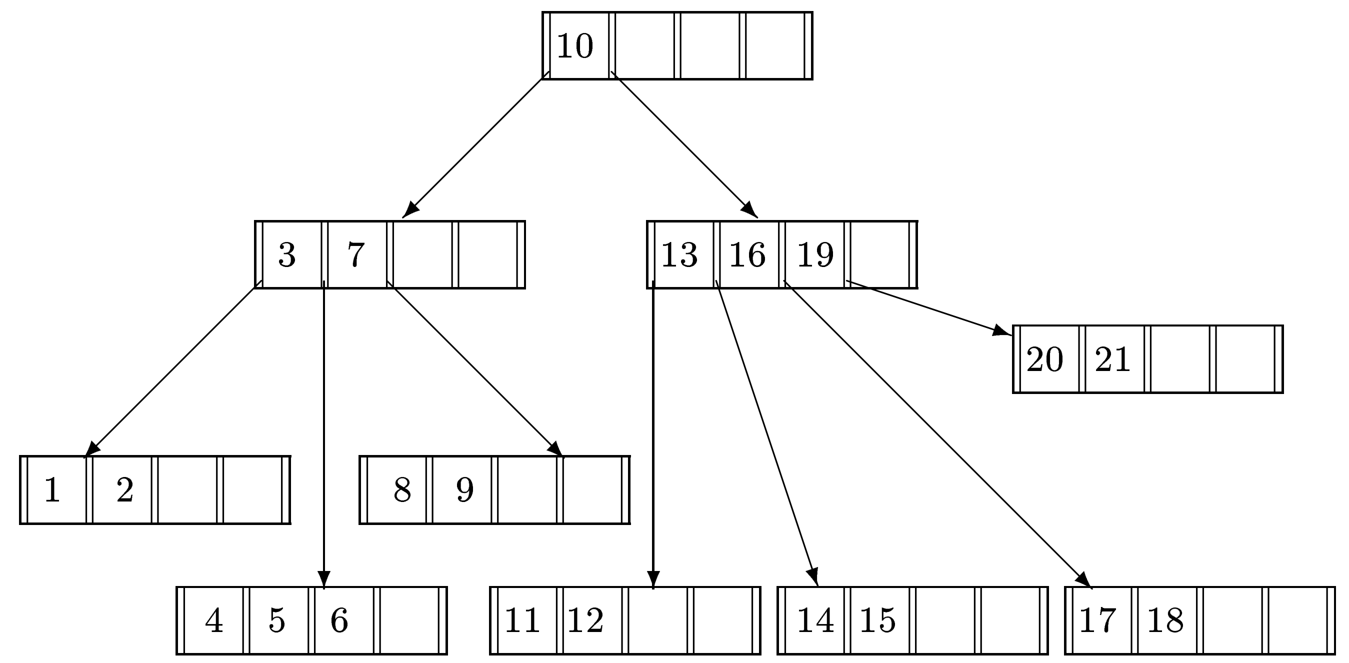
1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten außer der Wurzel hat mindestens k und höchstens $2k$ Einträge. Die Wurzel hat höchstens $2k$ Einträge. Die Einträge werden in allen Knoten sortiert gehalten.
3. Alle Knoten mit n Einträgen, außer den Blättern, haben $n + 1$ Kinder.
4. Seien $S_1, \ldots, S_n$ die Schlüssel eines Knotens mit $n + 1$ Kindern. $V_0, V_1, \ldots, V_n$ seien die Verweise auf diese Kinder. Dann gilt:
   (a) V_0 weist auf den Teilbaum mit Schlüsseln kleiner als S_1.
   (b) V_i $(i = 1, \ldots, n - 1)$ weist auf den Teilbaum, dessen Schlüssel zwischen S_i und S_{i+1} liegen.
   (c) V_n weist auf den Teilbaum mit Schlüsseln größer als S_n.
   (d) In den Blattknoten sind die Zeiger nicht definiert.

(Im Algorithmus wurde die Eindeutigkeit der Suchschlüssel angenommen)

Ein Beispielbaum



Einfügen einer 17



Einfügen in einen B-Baum

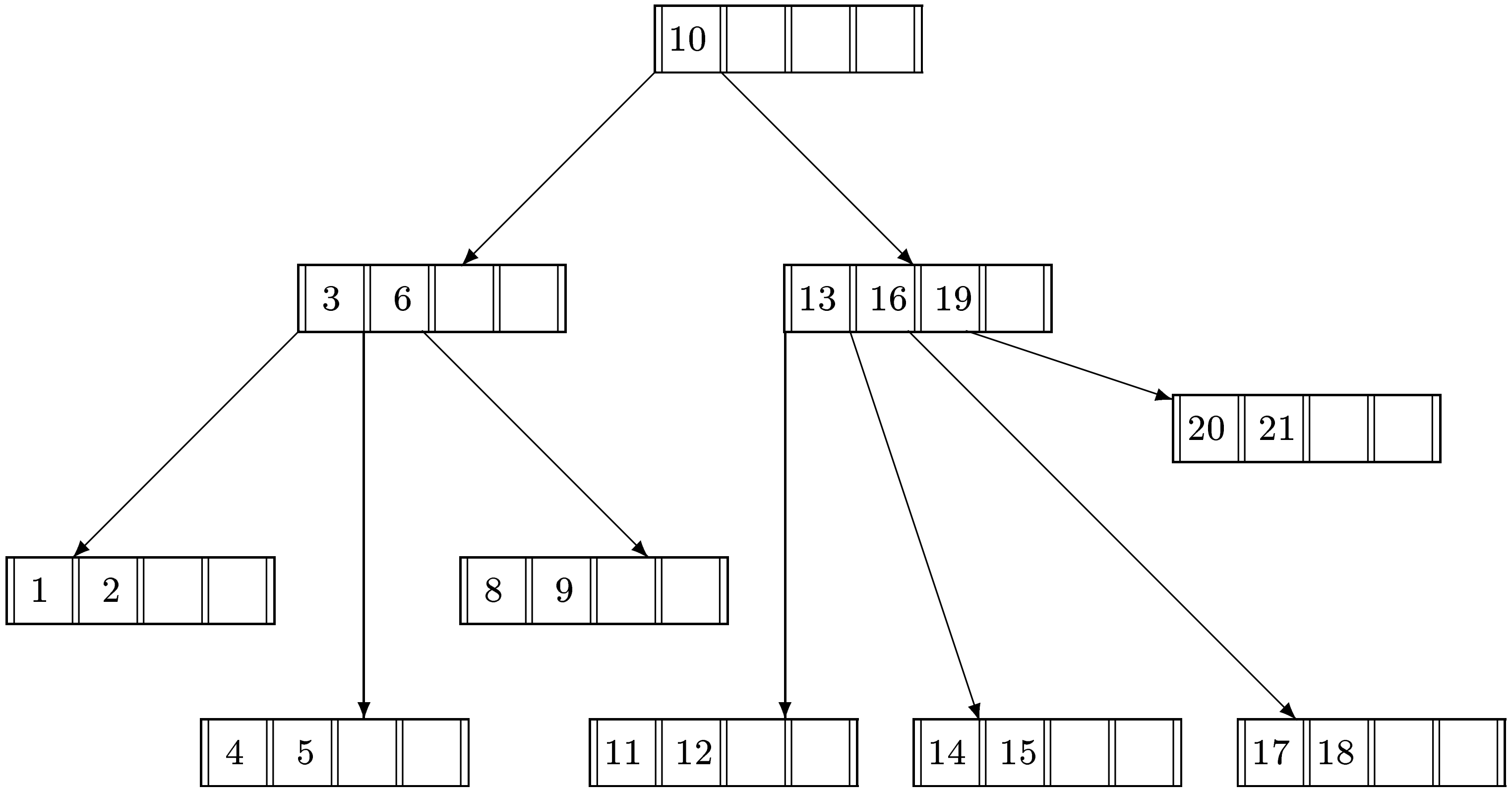
1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.

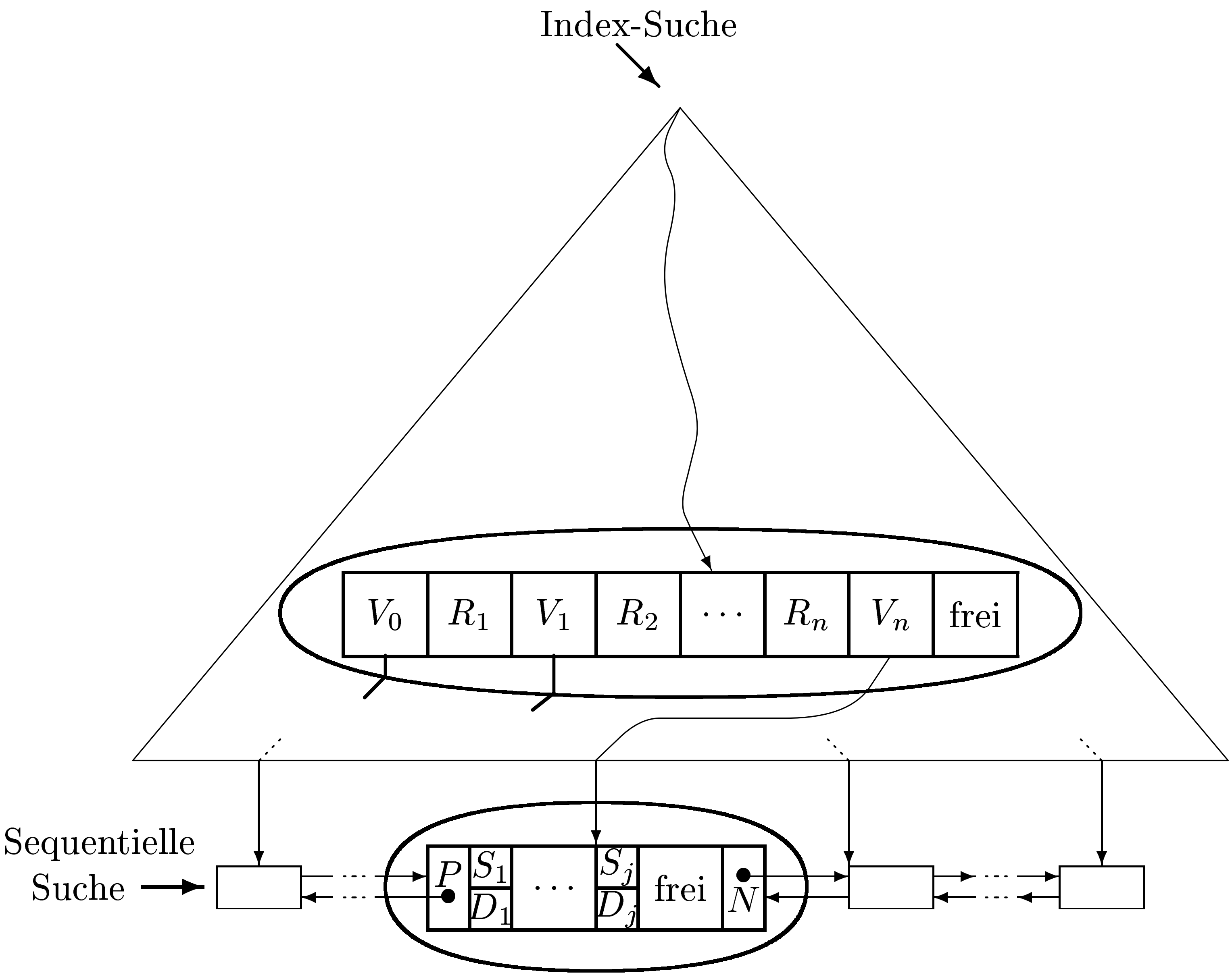
2. Füge den Schlüssel dort ein.

3. Ist der Knoten überfüllt, teile ihn
   - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
   - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
   - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten

4. Ist der Vaterknoten jetzt überfüllt?
   - Handelt es sich um die Wurzel, so lege eine neue Wurzel an.
   - Wiederhole Schritt 3 mit dem Vaterknoten.

Löschen der 7




Index-Suche
V_0
R_1
V_1
R_2
$\cdots$
R_n
V_n
frei
Sequentielle Suche
P
S_1
D_1
$\cdots$
S_j
D_j
frei
N

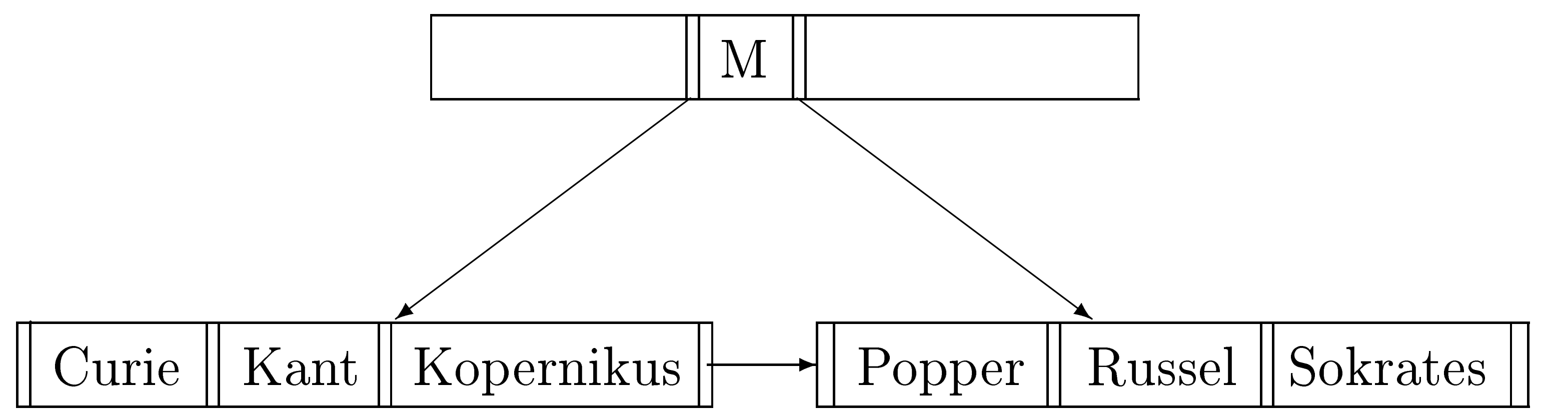
Eigenschaften von B^+-Bäumen

B^+-Baum vom Typ (k, k^n)

1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten außer der Wurzel hat mindestens k und höchstens $2k$ Einträge. Die Wurzel hat zwischen einem und $2k$ Einträgen. Die Einträge werden in allen Knoten sortiert gehalten.
3. Alle Knoten mit n Einträgen, außer den Blättern, haben $n+1$ Kinder.
4. Seien $S_1, \ldots, S_n$ die Schlüssel eines Knotens mit $n+1$ Kindern. $V_0, V_1, \ldots, V_n$ seien die Verweise auf diese Kinder.
5. Seien $R_1, \ldots, R_n$ die Referenzschlüssel eines inneren Knotens mit $n+1$ Kindern. Seien $V_0, V_1, \ldots, V_n$ die Verweise auf diese Kinder.
   (a) V_0 verweist auf den Teilbaum mit Schlüsseln kleiner oder gleich R_1.
   (b) V_i $(i = 1, \ldots, n-1)$ verweist auf den Teilbaum, dessen Schlüssel zwischen R_i und R_{i+1} liegen und R_{i+1} einschließen.
   (c) V_n verweist auf den Teilbaum mit Schlüsseln größer als R_n.

Präfix-B^+-Bäume

- Es werden nur *Referenz*schlüssel benötigt.



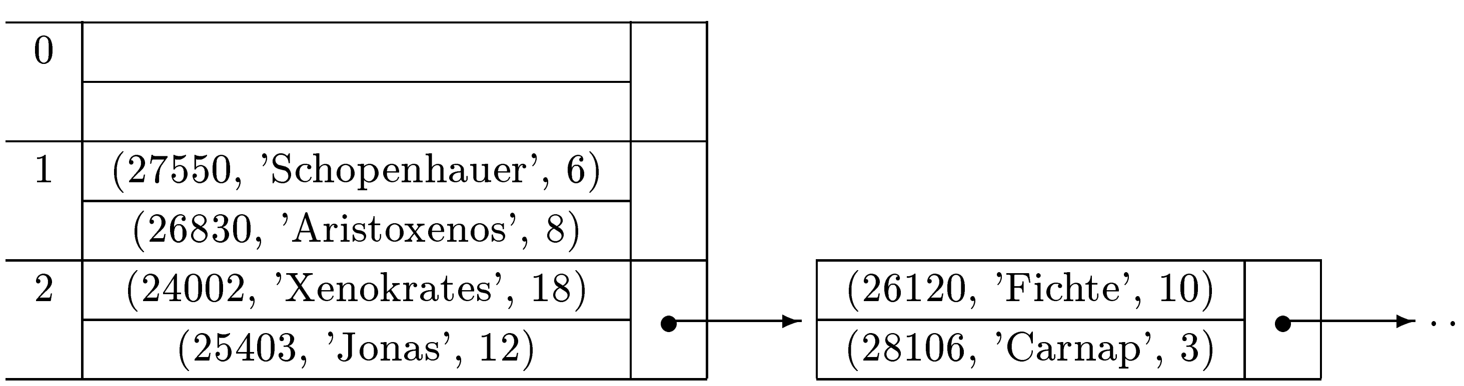
- beliebiger Referenzschlüssel R mit
  Kopernikus $\leq R <$ Popper

Hashing

- Hashfunktion $h(x) = x \bmod 3$

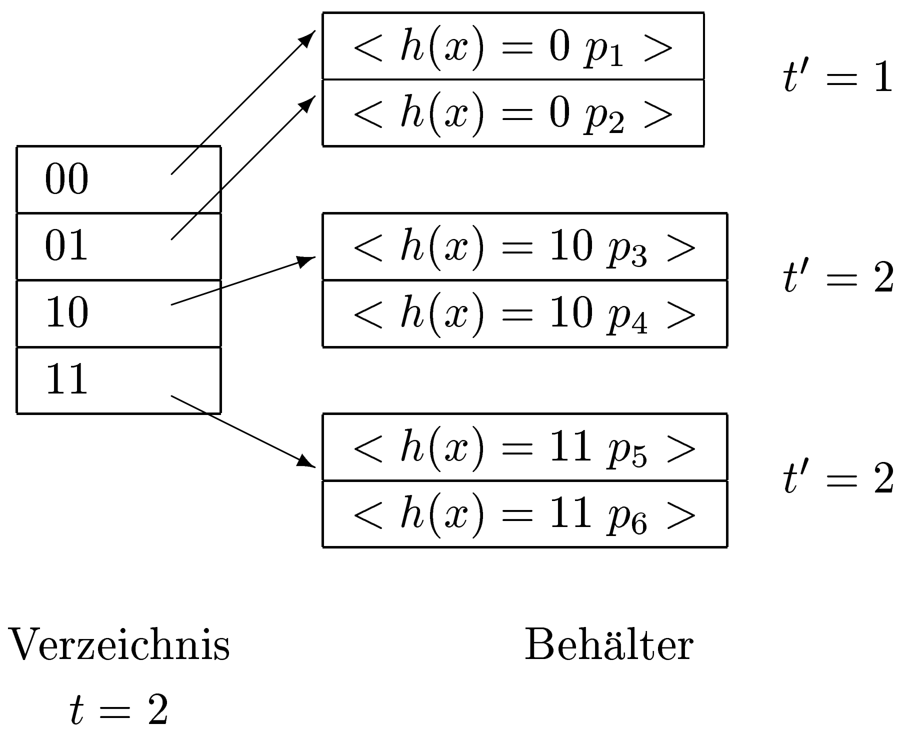
0	
1	(27550, 'Schopenhauer', 6)
2	(24002, 'Xenokrates', 18)
	(25403, 'Jonas', 12)

- Kollisionsbehandlung



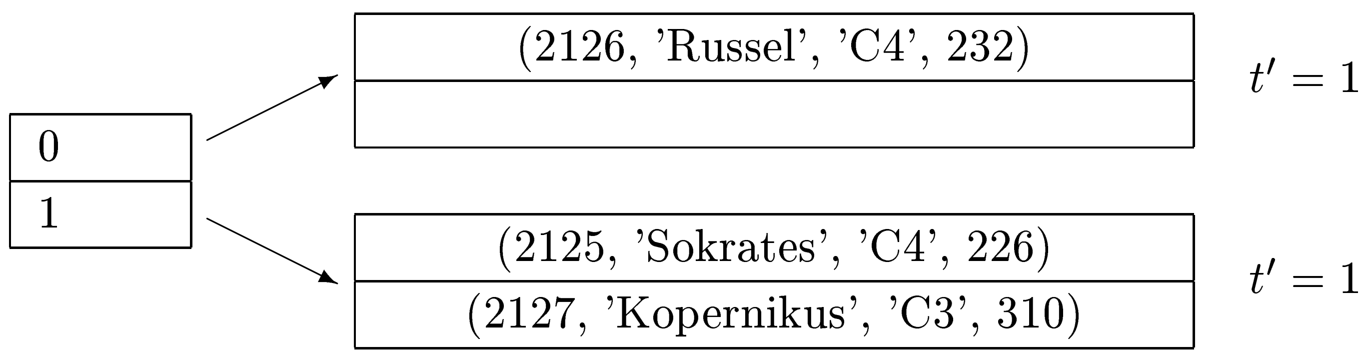
$\Longrightarrow$ ineffizient bei nicht vorhersehbarer Datenmenge

Erweiterbares Hashing



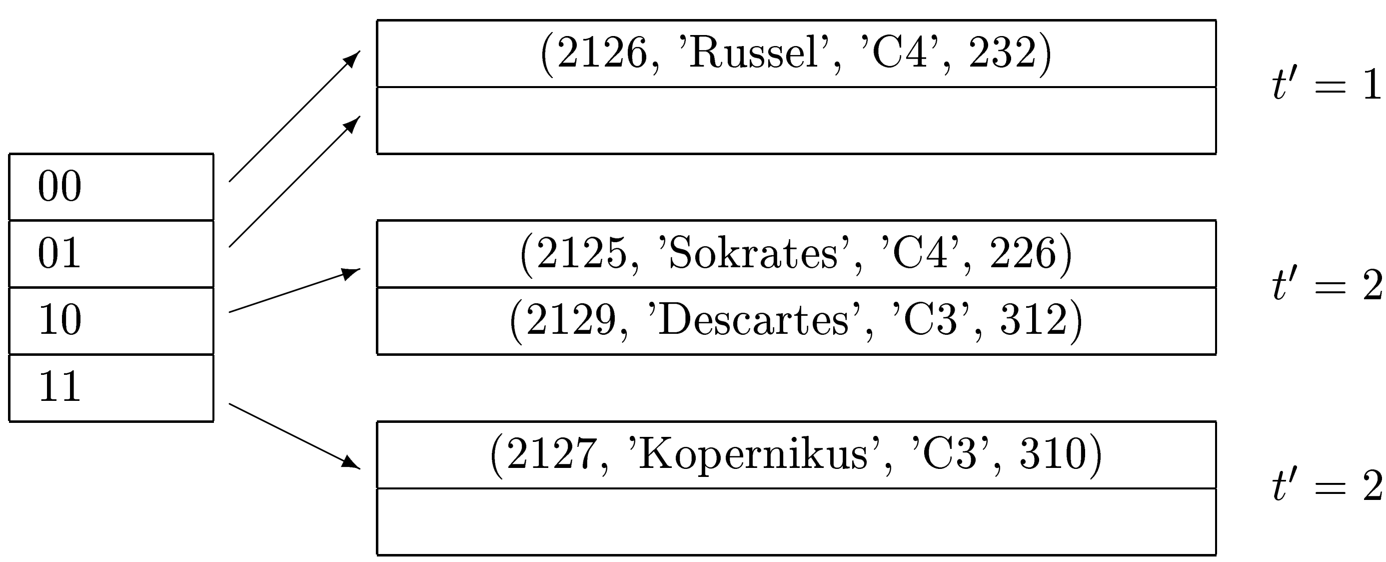
Demonstration des erweiterbaren Hashings

	$h(x)$	
x	d	p
2125	1	01100100001
2126	0	11100100001
2127	1	11100100001



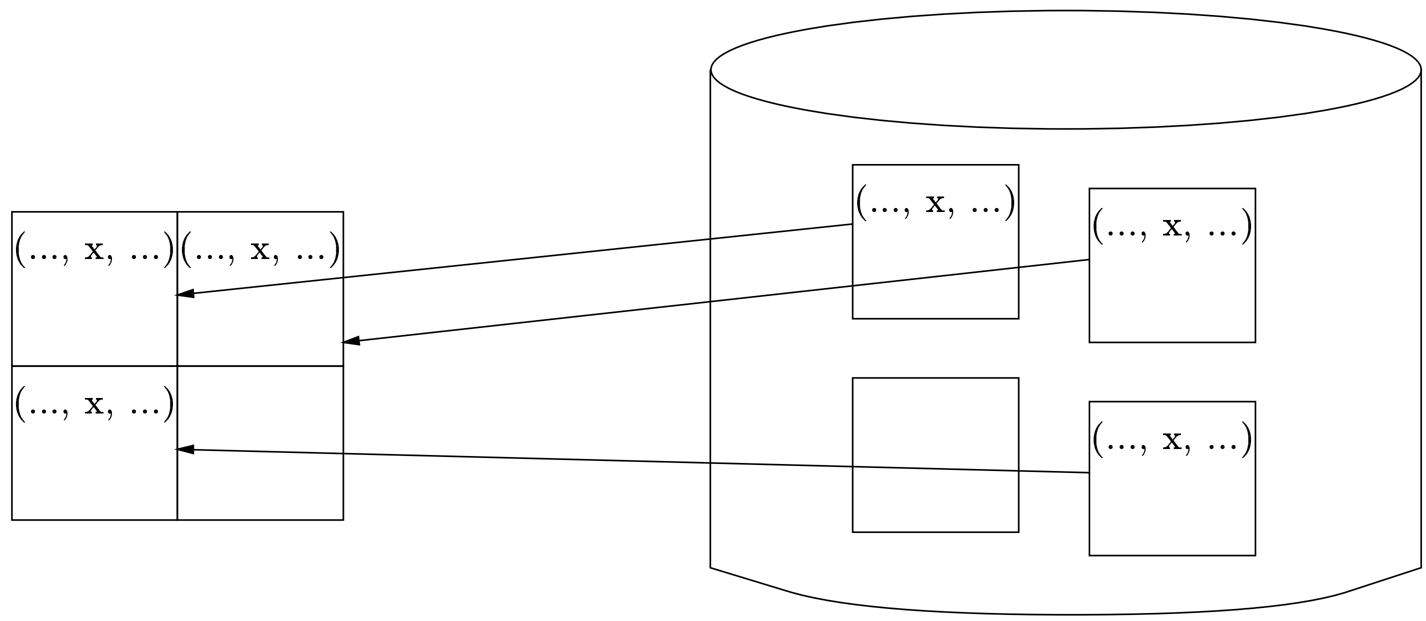
Einfügen von (2129, Descartes, C3, 312)

	$h(x)$	
x	d	p
2125	10	1100100001
2126	01	1100100001
2127	11	1100100001
2129	10	0010100001

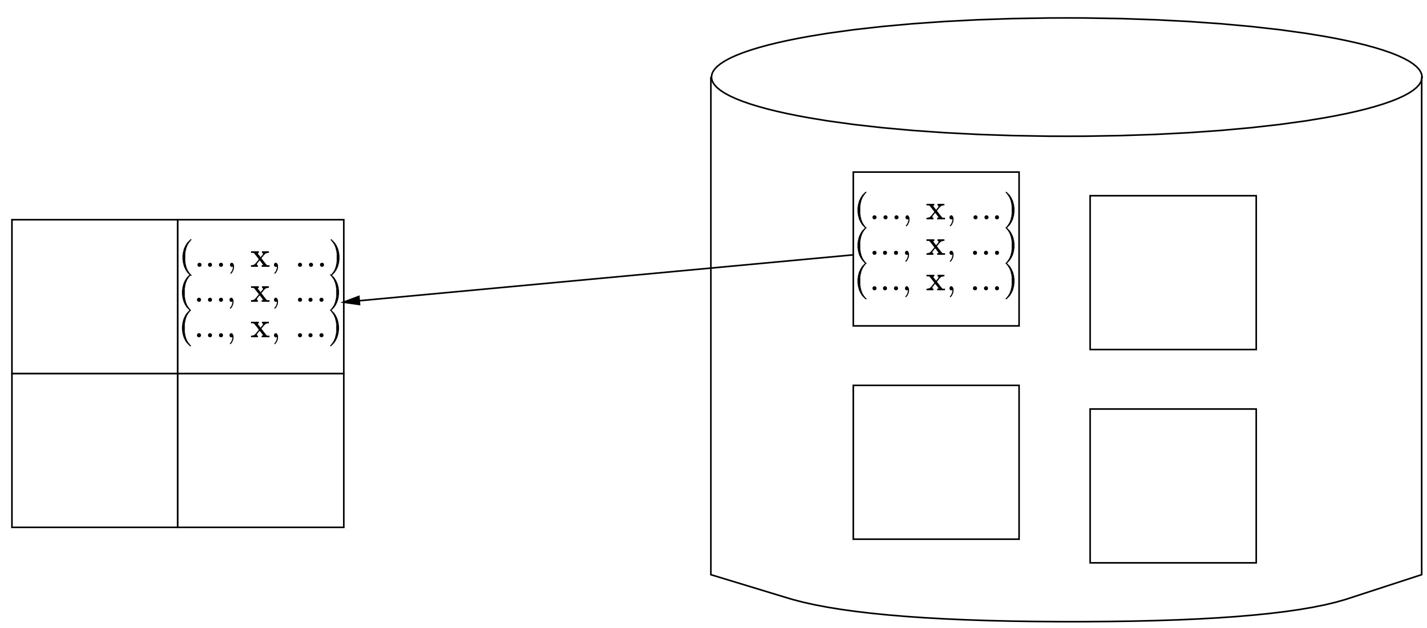


Ballung logisch verwandter Datensätze

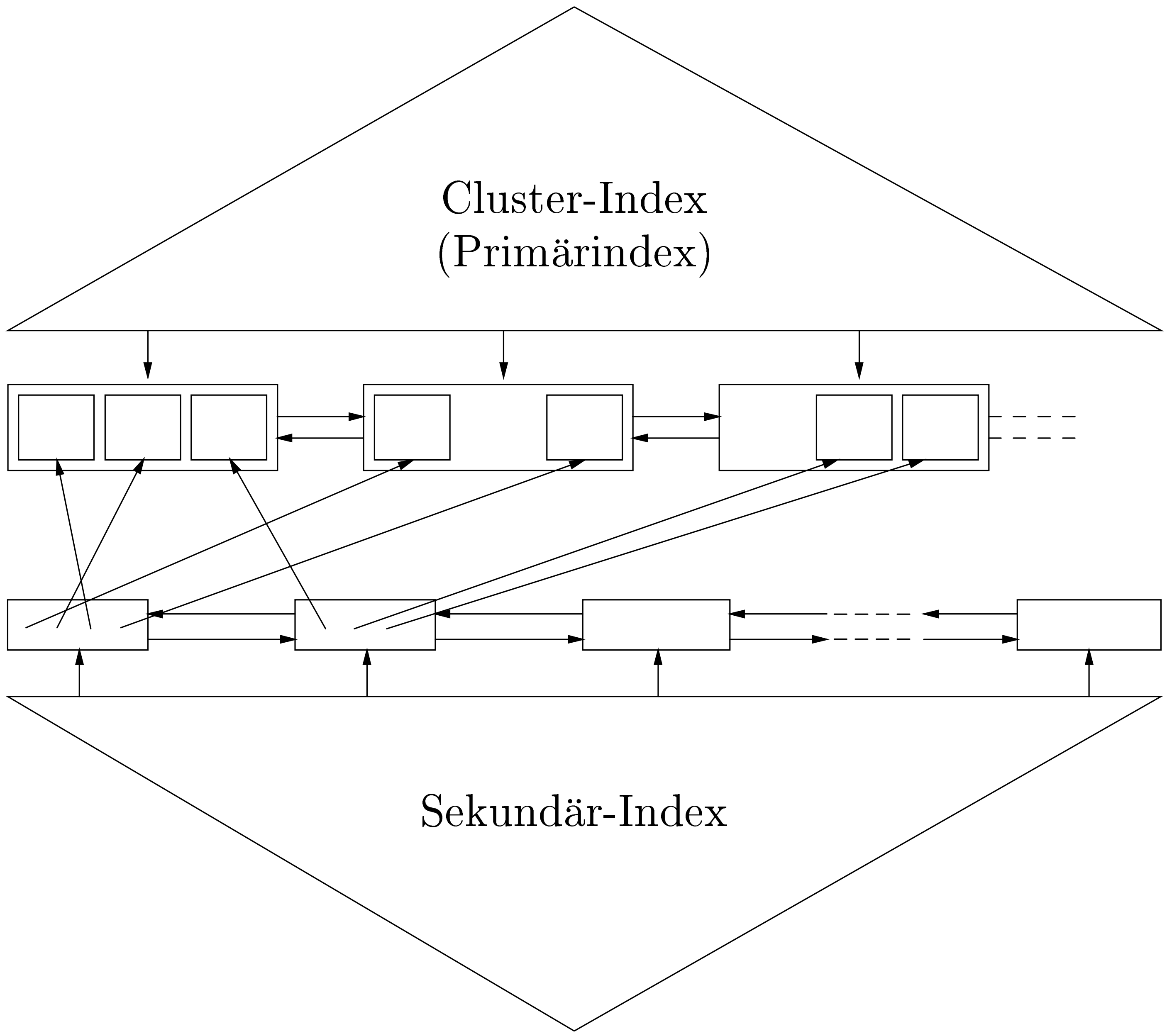
select $*$
from R
where $A = x$;



Hauptspeicher ⟵ Zugriffslücke ⟶ Hintergrundspeicher



Indexe und Ballung



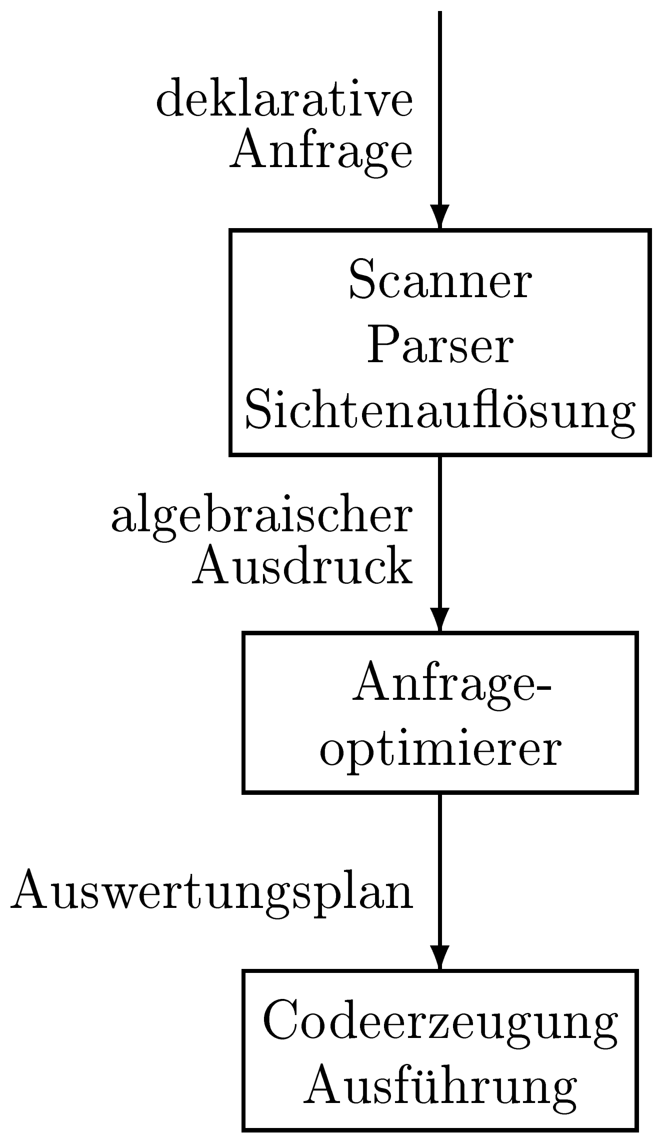
Seite P_i

2125 ∘	Sokrates	∘ C4 ∘	226	•
5041 ∘	Ethik	∘ 4 ∘	2125	•
5049 ∘	Mäeutik	∘ 2 ∘	2125	•
4052 ∘	Logik	∘ 4 ∘	2125	•
2126 ∘	Russel	∘ C4 ∘	232	•
5043 ∘	Erkenntnistheorie	∘ 3 ∘	2126	•
5052 ∘	Wissenschaftstheorie	∘ 3 ∘	2126	•
5216 ∘	Bioethik	∘ 2 ∘	2126	•

Seite P_{i+1}

2133 ∘	Popper	∘ C3 ∘	52	•
5259 ∘	Der Wiener Kreis	∘ 2 ∘	2133	•
2134 ∘	Augustinus	∘ C3 ∘	309	•
5022 ∘	Glaube und Wissen	∘ 2 ∘	2134	•
2137 ∘	Kant	∘ C4 ∘	7	•
5001 ∘	Grundzüge	∘ 4 ∘	2137	•
4630 ∘	Die 3 Kritiken	∘ 4 ∘	2137	•
	⋮			

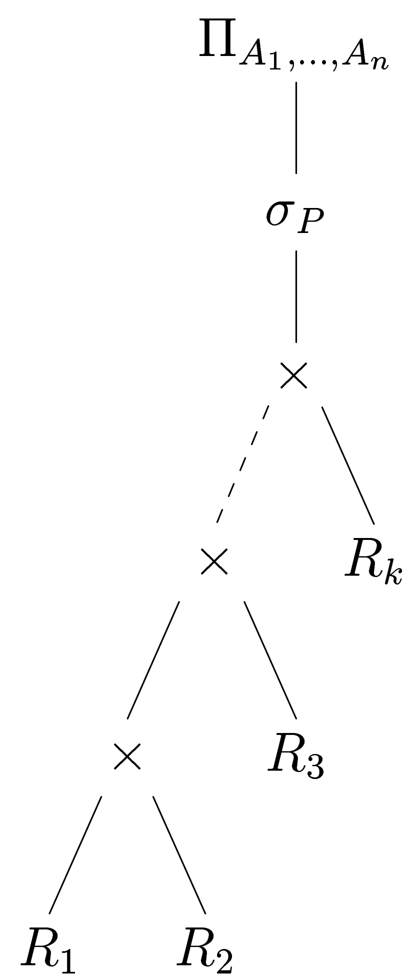
Ablauf der Anfragebearbeitung



Kanonische Übersetzung einer SQL-Anfrage

select $A_1, \ldots, A_n$
from $R_1, \ldots, R_k$
where P;

kanonische
$\Rightarrow$
Übersetzung



Demonstration der logischen Optimierung

select Titel
from Professoren, Vorlesungen
where Name = 'Popper' **and** PersNr = gelesenVon;

Kanonische Übersetzung:

$$\Pi_{\text{Titel}}(\sigma_{\text{Name}='\text{Popper}' \wedge \text{PersNr}=\text{gelesenVon}}(\text{Professoren} \times \text{Vorlesungen}))$$

Auswertung anhand der Beispielausprägung:

- 70 Tupel im Kreuzprodukt!

Demonstration der logischen Optimierung

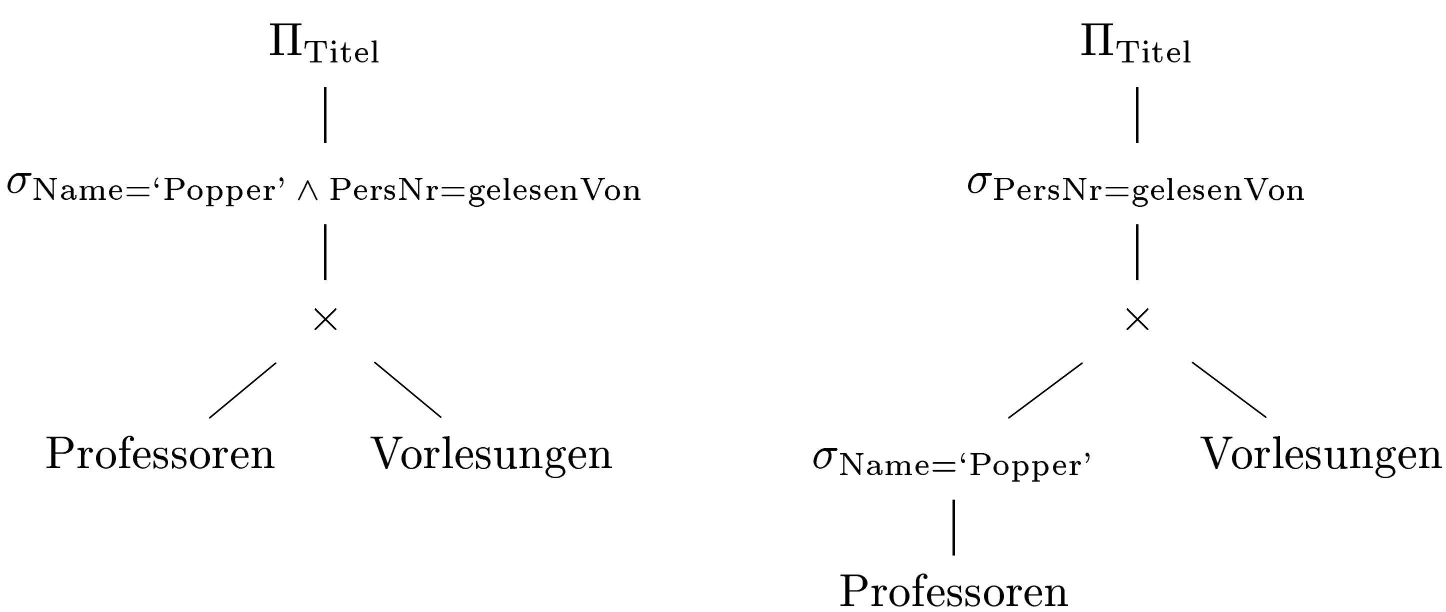
Verschiebung der Selektion:

$$\Pi_{\text{Titel}}(\sigma_{\text{PersNr=gelesenVon}}(\sigma_{\text{Name='Popper'}}(\text{Professoren}) \times \text{Vorlesungen}))$$

Auswertung anhand der Beispielprägung:

- 7 Tupel aus *Professoren* für die Selektion „anfassen"
- 10 Tupel aus *Vorlesungen* für das Kreuzprodukt „anfassen"

Zum Vergleich:



Äquivalenzen in der relationalen Algebra

1. Join, Vereinigung, Schnitt und Kreuzprodukt sind kommutativ, also:
$$\begin{aligned} R_1 \bowtie R_2 &= R_2 \bowtie R_1 \\ R_1 \cup R_2 &= R_2 \cup R_1 \\ R_1 \cap R_2 &= R_2 \cap R_1 \\ R_1 \times R_2 &= R_2 \times R_1 \end{aligned}$$

2. Selektionen sind untereinander vertauschbar.
$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

3. Join, Vereinigung, Schnitt und Kreuzprodukt sind assoziativ, also:
$$\begin{aligned} R_1 \bowtie (R_2 \bowtie R_3) &= (R_1 \bowtie R_2) \bowtie R_3 \\ R_1 \cup (R_2 \cup R_3) &= (R_1 \cup R_2) \cup R_3 \\ R_1 \cap (R_2 \cap R_3) &= (R_1 \cap R_2) \cap R_3 \\ R_1 \times (R_2 \times R_3) &= (R_1 \times R_2) \times R_3 \end{aligned}$$

4. Konjunktionen in einer Selektionsbedingung können in mehrere Selektionen aufgebrochen, bzw. nacheinander ausgeführte Selektionen können durch Konjunktionen zusammengefügt werden.
$$\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots))$$

5. Geschachtelte Projektionen können eliminiert werden.
$$\Pi_{l_1}(\Pi_{l_2}(\ldots(\Pi_{l_n}(R))\ldots)) = \Pi_{l_1}(R)$$
   Damit eine solche Schachtelung überhaupt sinnvoll ist, muß gelten:
$$l_1 \subseteq l_2 \subseteq \ldots \subseteq l_n \subseteq \mathcal{R} = sch(R)$$

Äquivalenzen in der relationalen Algebra

6. Eine Selektion kann an einer Projektion „vorbeigeschoben" werden, falls die Projektion keine Attribute aus der Selektionsbedingung entfernt. Es gilt also

$$\Pi_l(\sigma_p(R)) = \sigma_p(\Pi_l(R)), \text{ falls } \mathit{attr}(p) \subseteq l$$

7. Selektionen können an Joinoperationen (oder Kreuzprodukten) vorbeigeschoben werden, falls sie nur Attribute *eines* der beiden Join-Argumente verwenden. Enthält die Bedingung p beispielsweise nur Attribute aus $\mathcal{R}_1$, dann gilt

$$\sigma_p(R_1 \bowtie R_2) = \sigma_p(R_1) \bowtie R_2$$

8. Auf ähnliche Weise können auch Projektionen verschoben werden. Hier muß allerdings beachtet werden, daß die Join-Attribute bis zum Join erhalten bleiben.

$$\begin{aligned}
&\Pi_l(R_1 \bowtie_p R_2) = \Pi_l(\Pi_{l_1}(R_1) \bowtie_p \Pi_{l_2}(R_2)) \text{ mit} \\
&l_1 = \{A|A \in \mathcal{R}_1 \cap l\} \cup \{A|A \in \mathcal{R}_1 \cap \mathit{attr}(p)\} \text{ und} \\
&l_2 = \{A|A \in \mathcal{R}_2 \cap l\} \cup \{A|A \in \mathcal{R}_2 \cap \mathit{attr}(p)\}
\end{aligned}$$

9. Selektionen können mit Mengenoperationen wie Vereinigung, Schnitt und Differenz vertauscht werden, also:

$$\begin{aligned}
\sigma_p(R \cup S) &= \sigma_p(R) \cup \sigma_p(S) \\
\sigma_p(R \cap S) &= \sigma_p(R) \cap \sigma_p(S) \\
\sigma_p(R - S) &= \sigma_p(R) - \sigma_p(S)
\end{aligned}$$

10. Der Projektions-Operator kann mit der Vereinigung vertauscht werden.
$$\Pi_l(R_1 \cup R_2) = \Pi_l(R_1) \cup \Pi_l(R_2)$$
Eine Vertauschung der Projektion mit Durchschnitt und Differenz ist allerdings nicht zulässig.

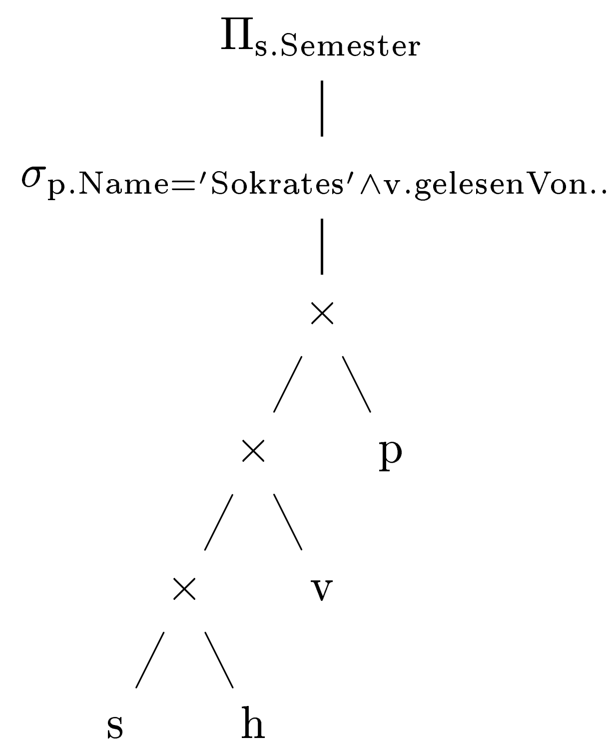
11. Eine Selektion und ein Kreuzprodukt können zu einem Join zusammengefaßt werden, wenn die Selektionsbedingung eine Joinbedingung ist. Für Equijoins gilt beispielsweise
$$\sigma_{R_1.A_1=R_2.A_2}(R_1 \times R_2) = R_1 \bowtie_{R_1.A_1=R_2.A_2} R_2$$

12. Auch an Bedingungen können Veränderungen vorgenommen werden. Beispielsweise kann eine Disjunktion mit Hilfe von DeMorgan's Gesetz in eine Konjunktion umgewandelt werden, um vielleicht später die Anwendung von Regel 4 zu ermöglichen:
$$\neg(p_1 \vee p_2) = \neg p_1 \wedge \neg p_2$$

Anwendung der Transformationsregeln

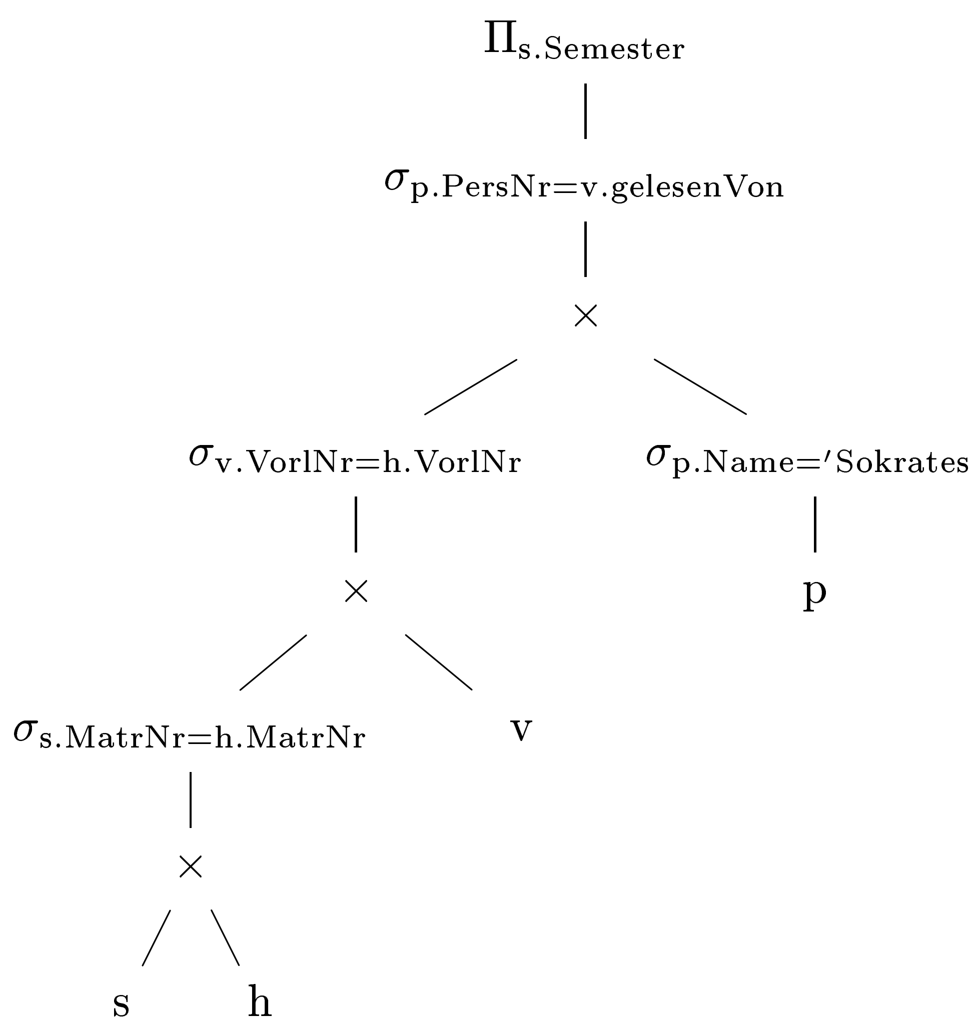
Die Ausgangsanfrage und ihre kanonische Übersetzung:

select distinct s.Semester
from Studenten s, hören h,
        Vorlesungen v, Professoren p
where p.Name = 'Sokrates' **and**
          v.gelesenVon = p.PersNr **and**
          v.VorlNr = h.VorlNr **and**
          h.MatrNr = s.MatrNr;



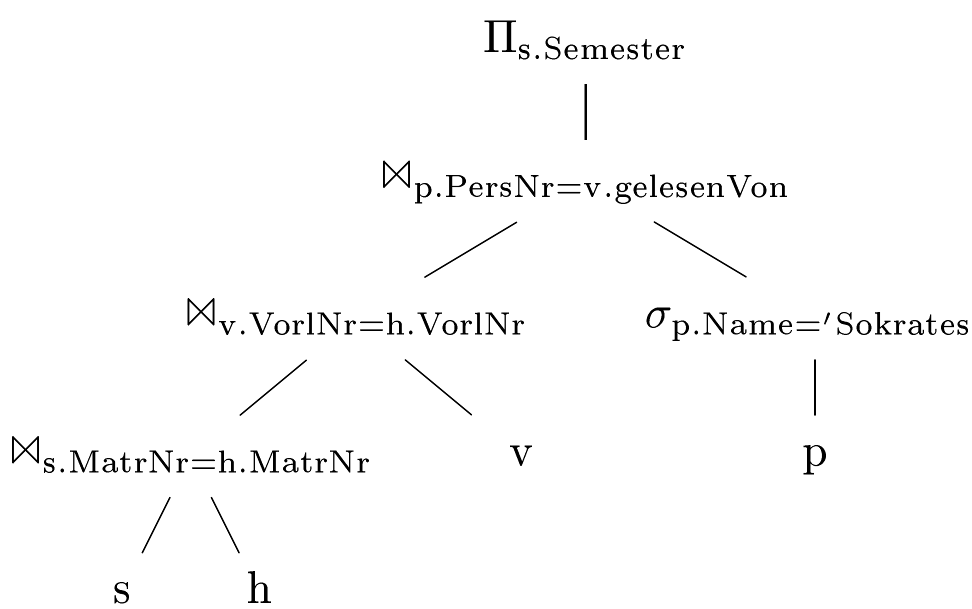
Verschieben der Selektionen

- Aufbrechen der Konjunktionen (Regel 4)
- Verschieben der Selektionen „nach unten“ (Regel 2, 6, 7 und 9)



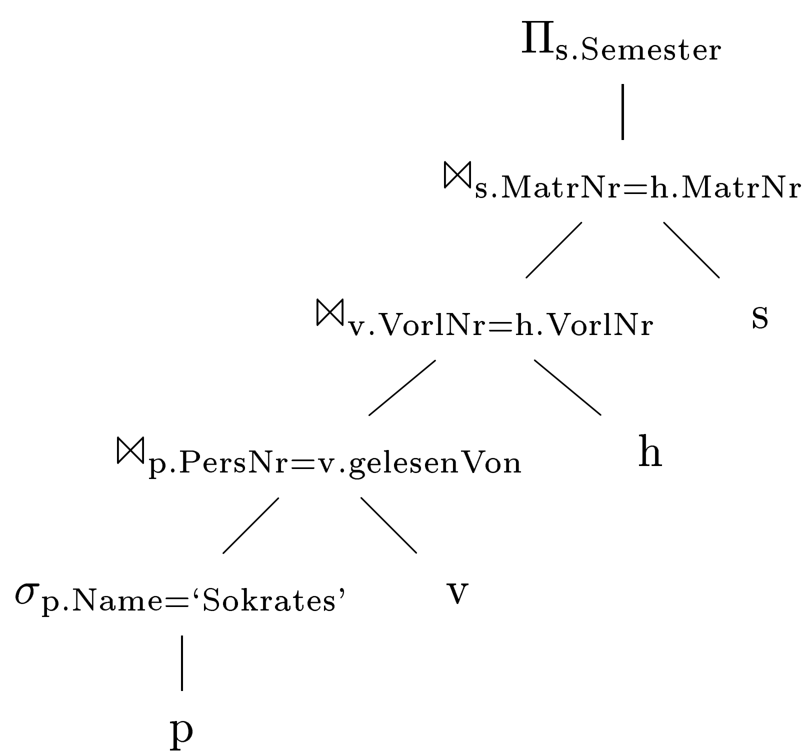
Erzeugen von Joins aus Kreuzprodukten

- Zusammenfassen von Selektionen und Kreuzprodukten (Regel 5 und 11)

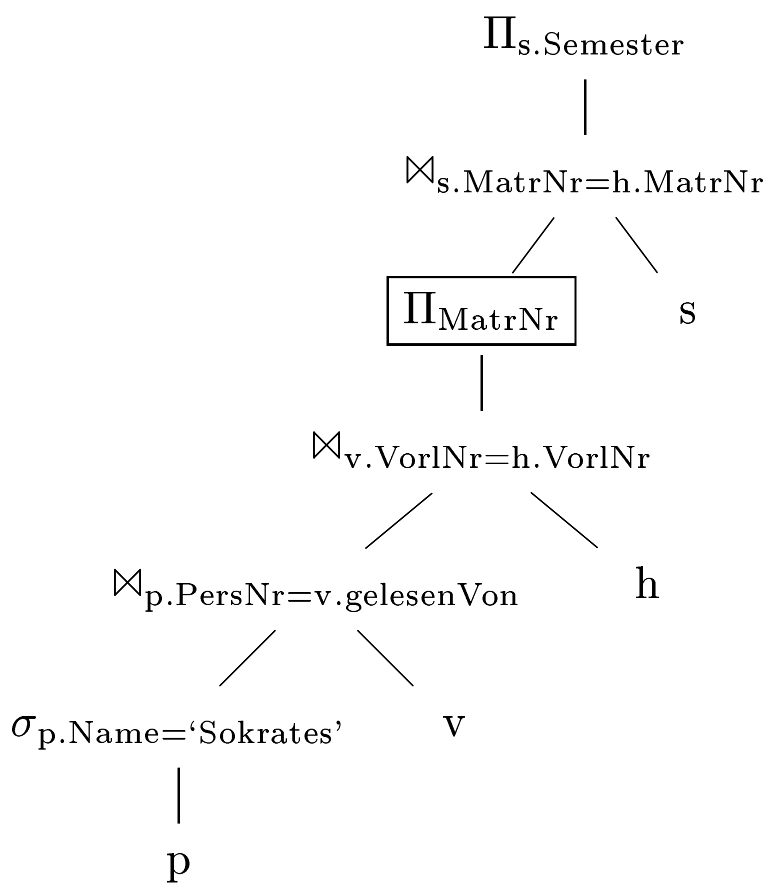


Bestimmung der Joinreihenfolge

- Kommutativität des Joins (Regel 1)
- Assoziativität des Joins (Regel 3)



Einfügen und Verschieben von Projektionen

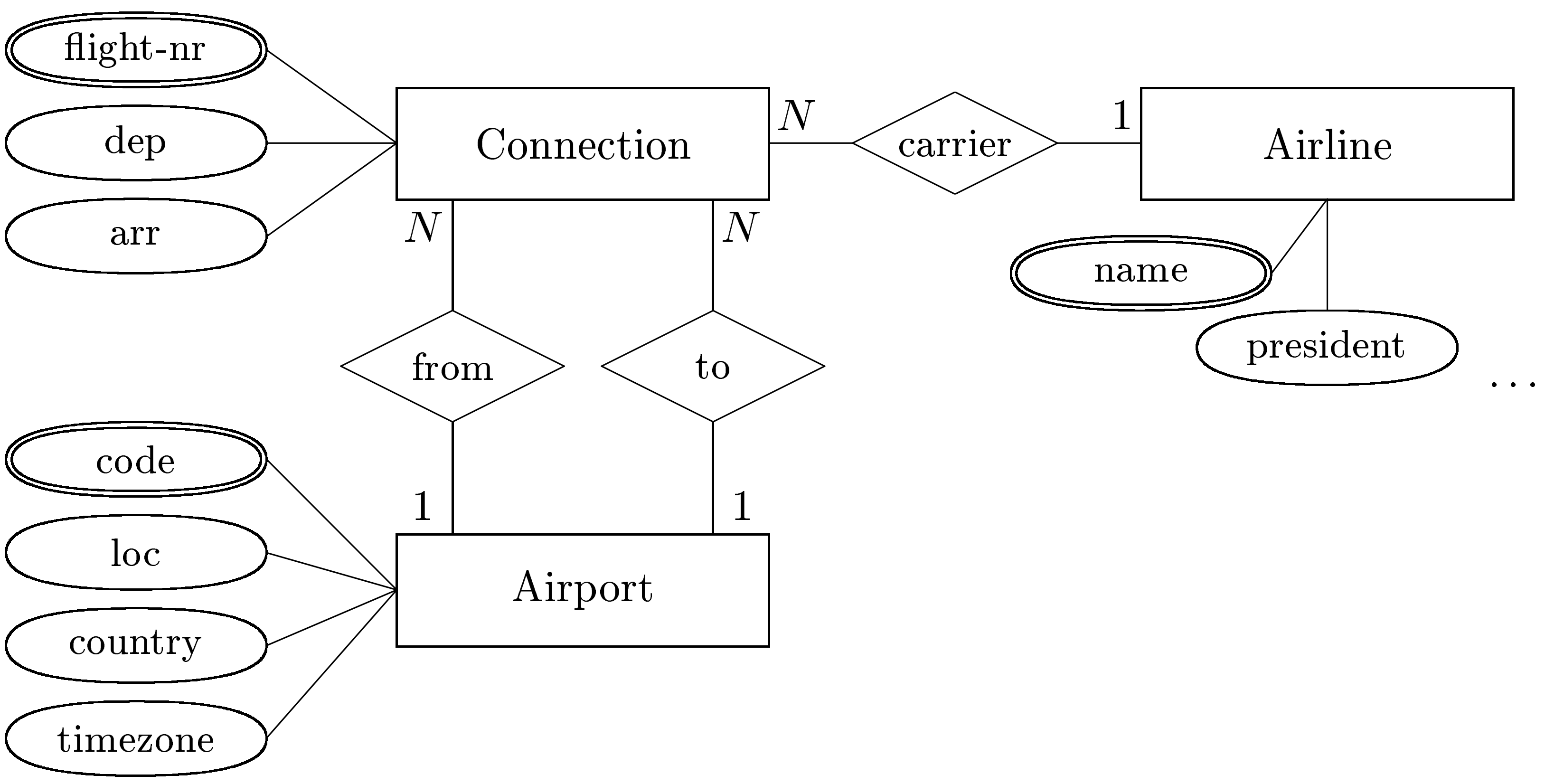


Zusammenfassung

1. Aufbrechen von Selektionen
2. Verschieben der Selektionen soweit wie möglich nach unten im Operatorbaum
3. Zusammenfassen von Selektionen und Kreuzprodukten zu Joins
4. Bestimmung der Anordnung der Joins
5. u.U. Einfügen von Projektionen
6. Verschieben der Projektionen soweit wie möglich nach unten im Operatorbaum

Ein weiteres Beispiel: Flugreservierung

Entity-Relationship-Schema



Relationales Schema

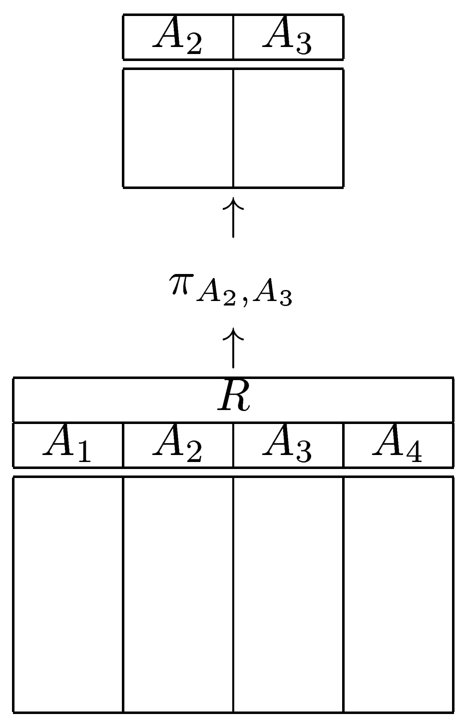
Connection					
flight-nr	from	to	dep	arr	carrier
4711	FRA	JFK	1500	1730	UA
4567	FRA	LGA	1230	1415	AA
3412	JFK	LAX	1830	2115	DL
6734	MUC	FRA	1310	1400	LH
...	...	...	...	...	...

Airline	
name	...
AA	...
DL	...
LH	...
UA	...
...	...

Airport			
code	loc	country	timezone
FRA	Frankfurt	Germany	MET
MUC	München	Germany	MET
LAX	Los Angeles	USA	PST
JFK	New York	USA	EST
LGA	New York	USA	EST
...	...	...	...

Relationenalgebra: $\pi, \sigma, \times, \bowtie, \cup, -, \cap, \div, \rtimes, \ltimes, ⟗, ⟖, ⟕, \ldots$

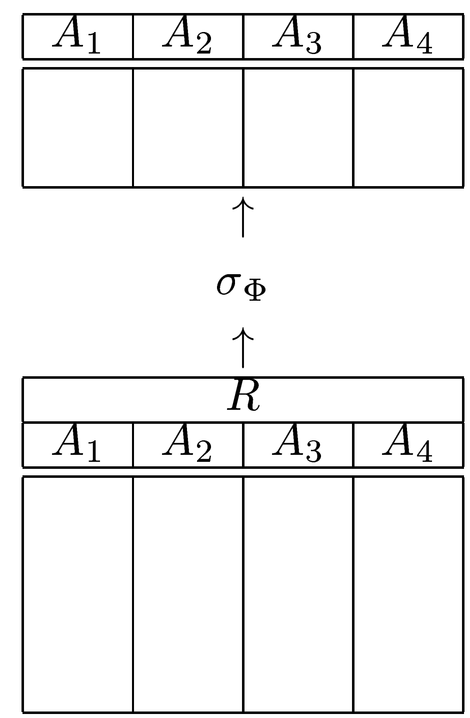
Projektion π



Beispiel

$\pi_{\text{loc}}(\text{Airport})$

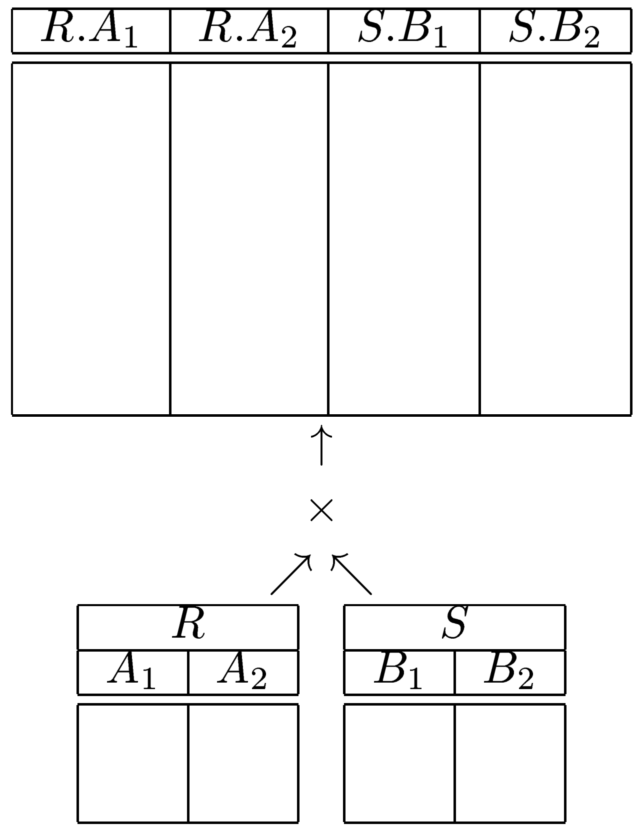
Selektion σ



Beispiel

$\sigma_{\text{to}=\text{from}}(\text{Connection})$

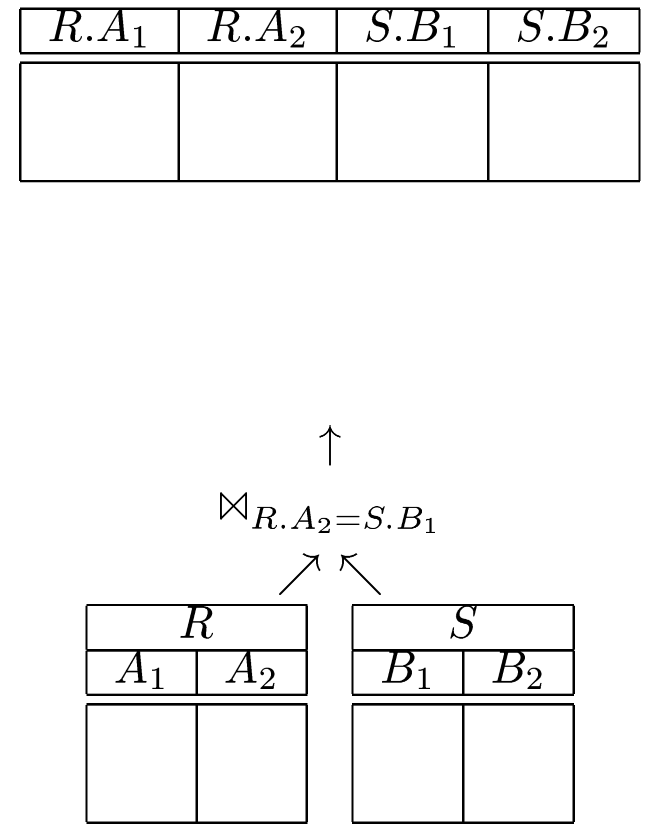
Kreuzprodukt $\times$



Beispiel

Airport $\times$ Connection

Verbund, Join $\bowtie_{\ldots}$



Beispiel

Airport $\bowtie_{\text{code}=\text{from}}$ Connection

SQL-Anfrage: Von München direkt nach NY?

select c.dep
from Airport n, Connection c, Airport p
where n.loc = ‘New York‘ **and**
      n.code = c.to **and**
      c.from = p.code **and**
      p.loc = ‘München‘

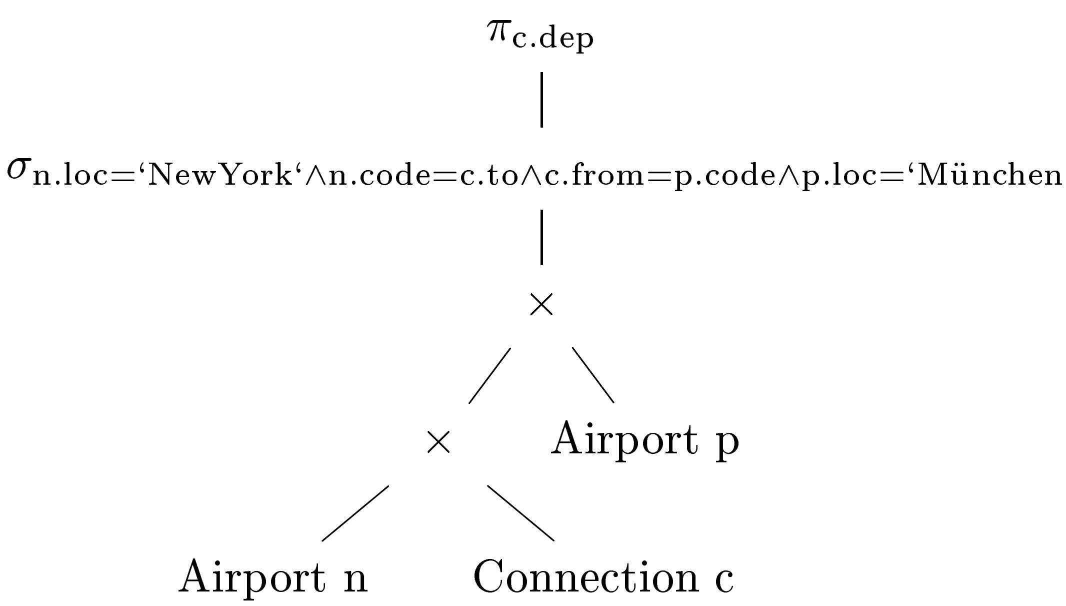
Airport p		
p.code	p.loc	...
...	...	...
MUC	München	...
...	...	...

Connection c			
...	c.from	c.to	...

Airport n		
n.code	n.loc	...
...	...	...
JFK	New York	...
LGA	New York	...
...	...	...

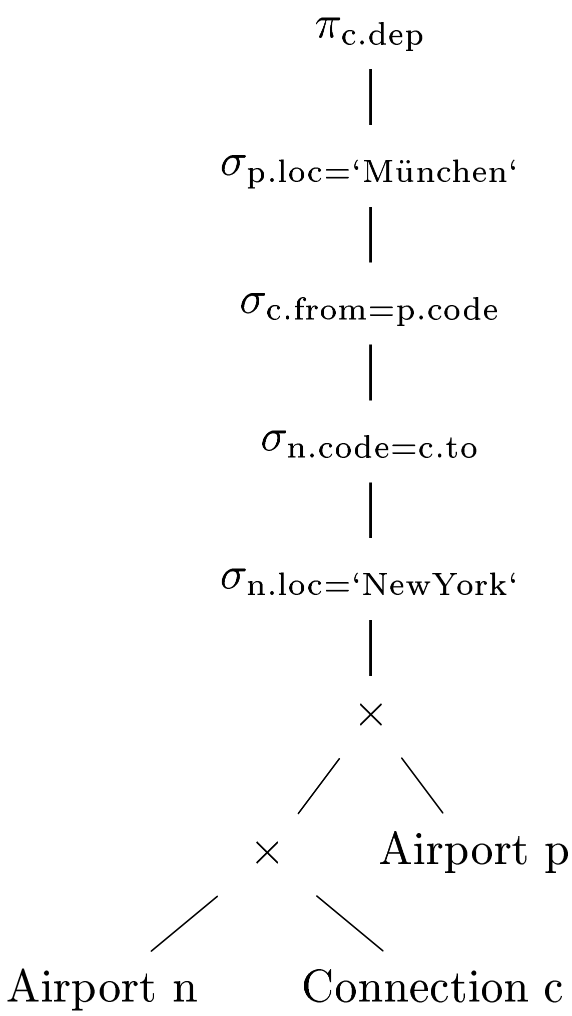
Kanonische Übersetzung

select c.dep
from Airport n, Connection c, Airport p
where n.loc = ‘New York‘ **and**
n.code = c.to **and**
c.from = p.code **and**
p.loc = ‘München‘



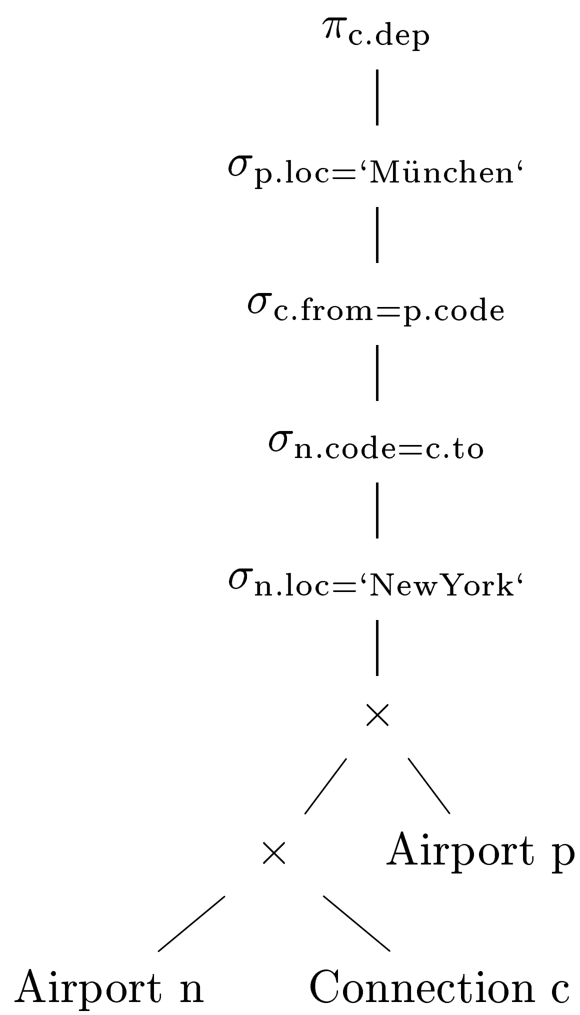
Selektionsprädikate „aufbrechen“

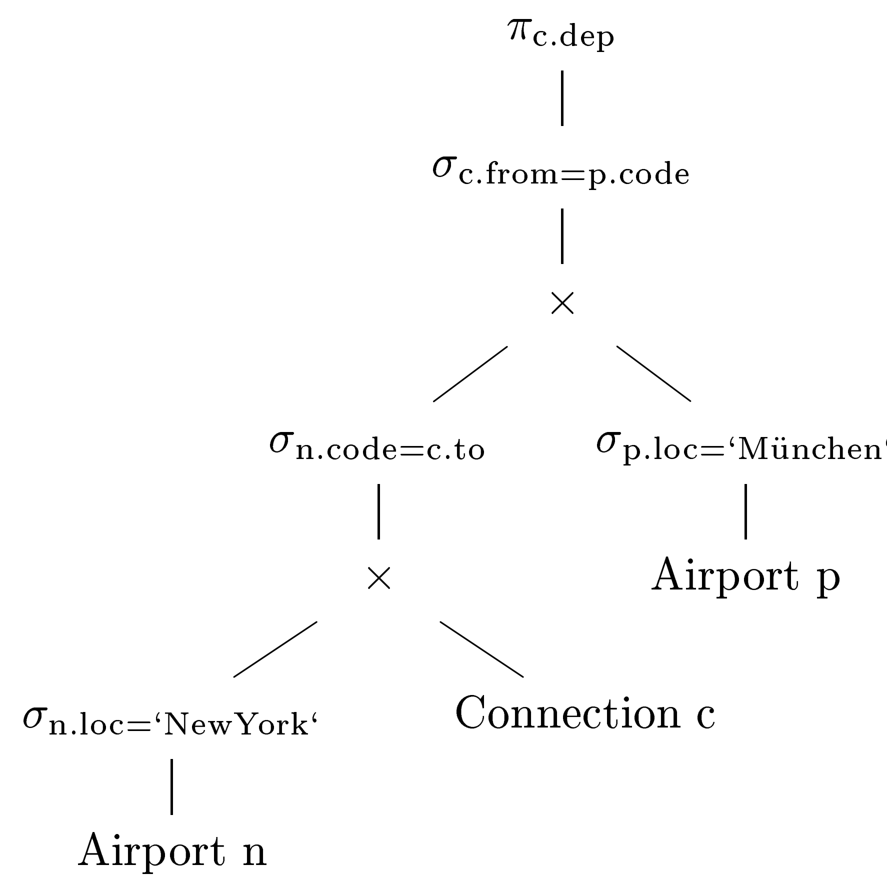
$$\boxed{\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots))}$$



„Pushing Selections“

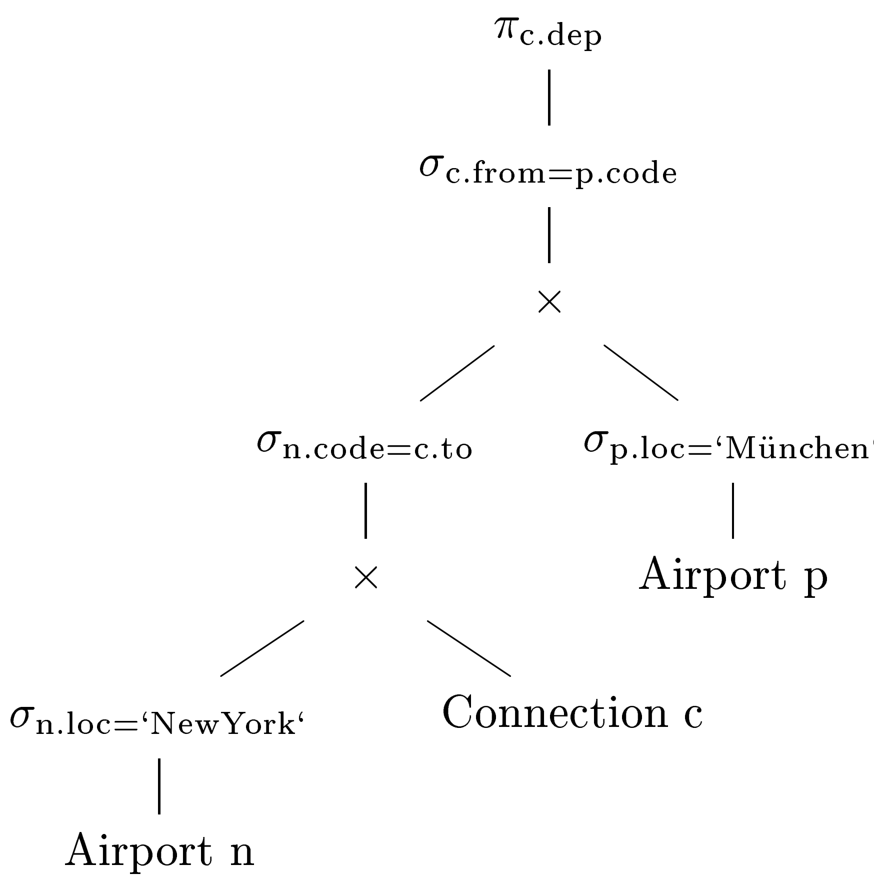
$$
\begin{array}{rcl}
\sigma_p(\sigma_q(R)) & = & \sigma_q(\sigma_p(R)) \\
\sigma_p(R_1 \bowtie R_2) & = & \sigma_p(R_1) \bowtie R_2 \\
\sigma_p(R_1 \times R_2) & = & \sigma_p(R_1) \times R_2
\end{array}
$$

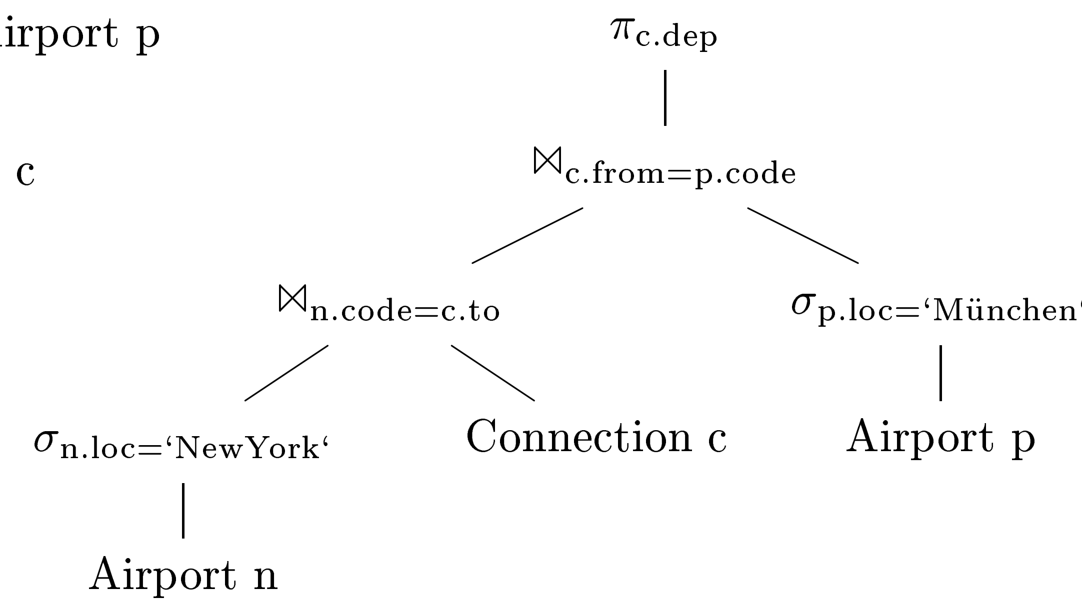


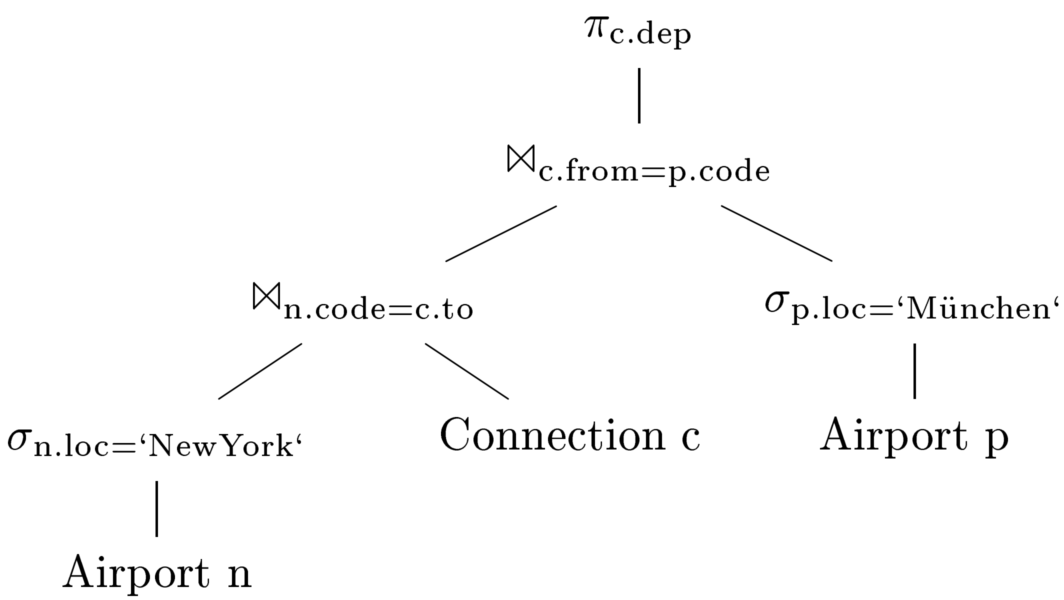


Zusammenfassung von $\sigma \times$ zu $\bowtie$

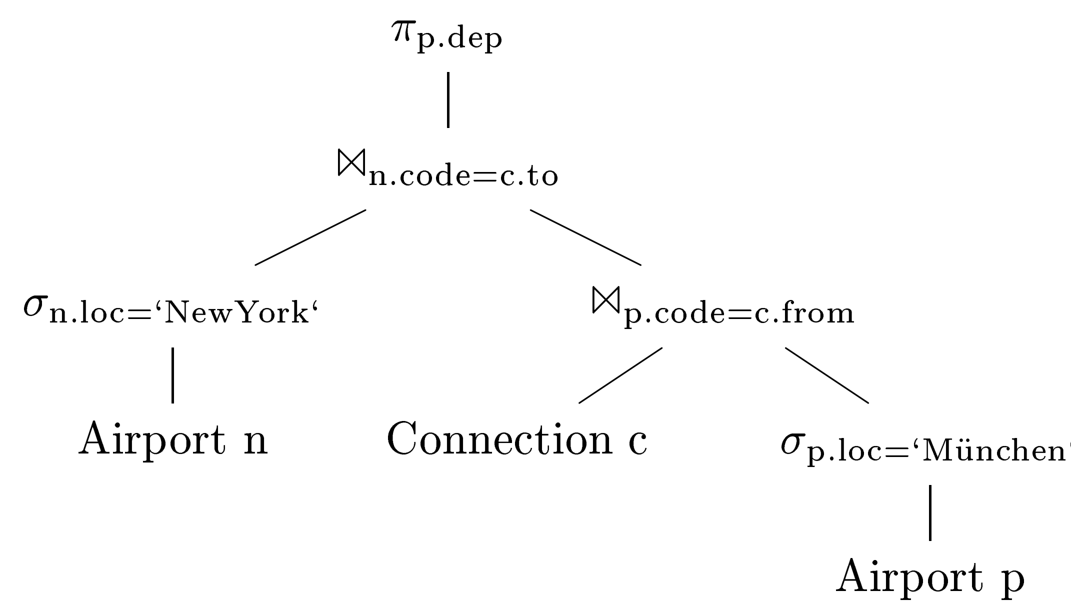
$$\boxed{\sigma_{R.A=S.B}(R \times S) = R \bowtie_{R.A=S.B} S}$$





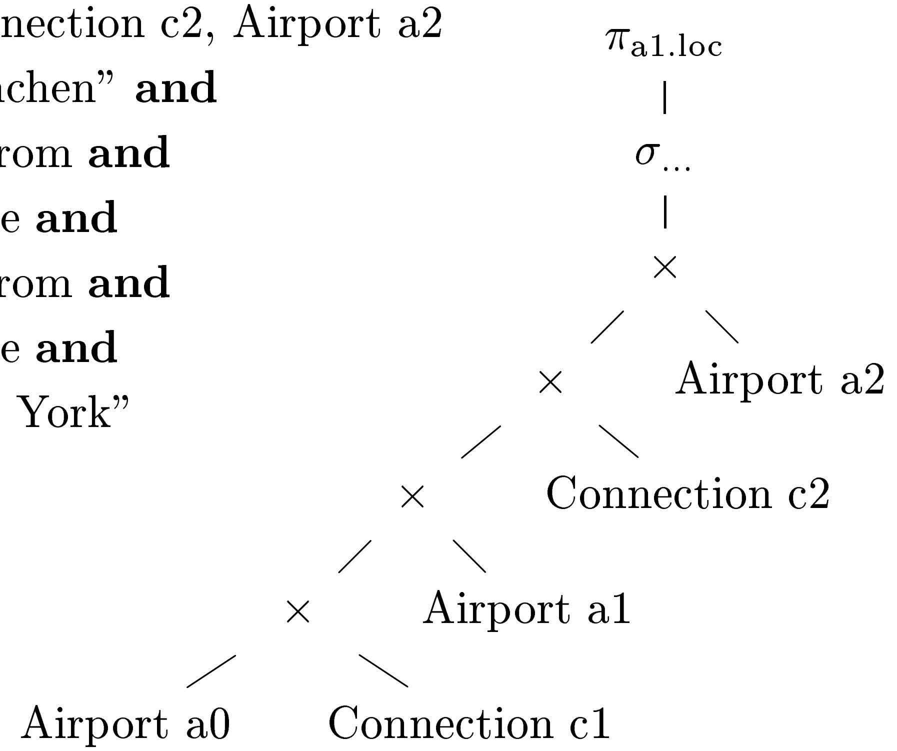


$$\boxed{(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)}$$

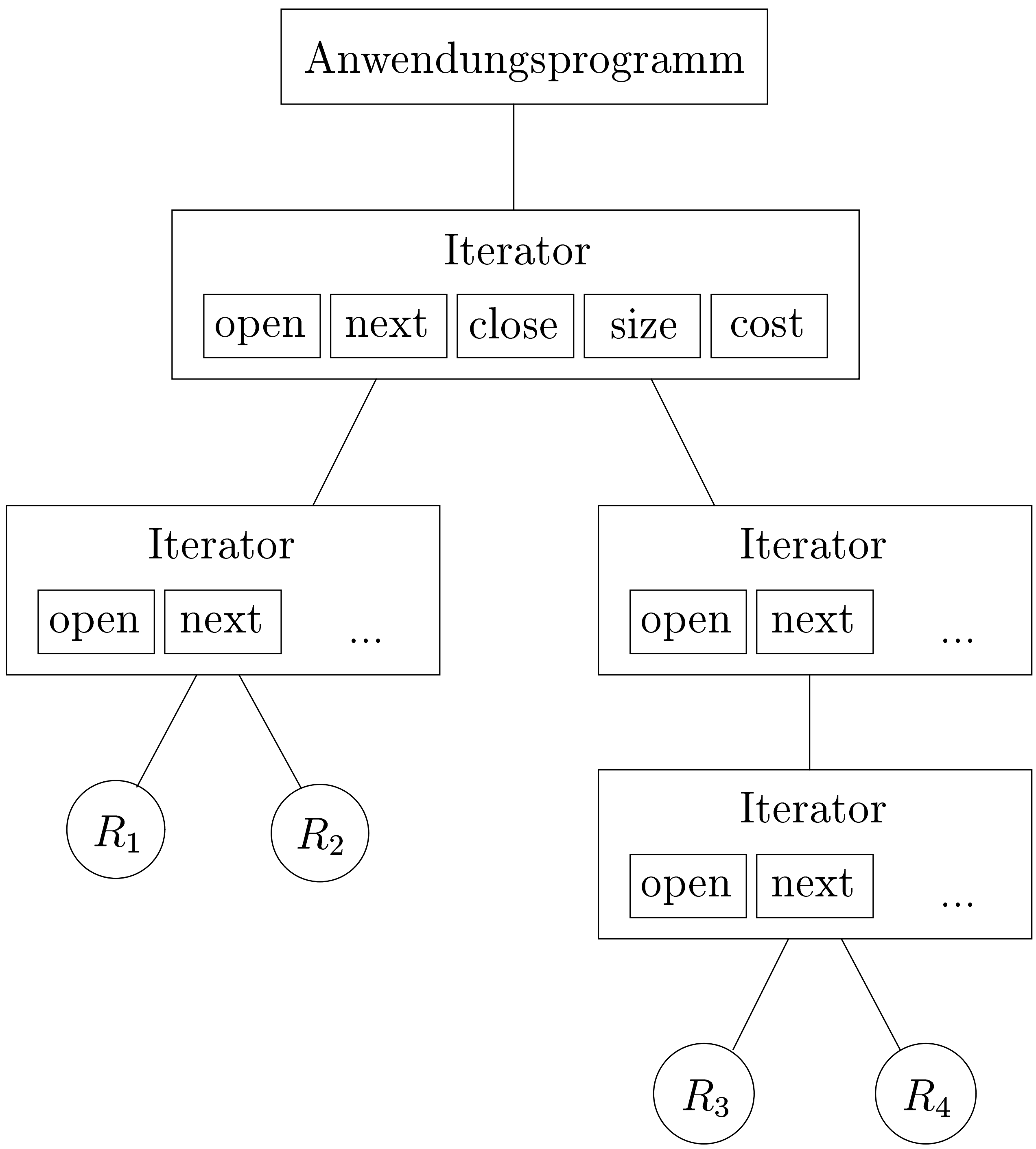


MUC → NY mit genau einmal Umsteigen

select a1.loc
from Airport a0, Connection c1,
Airport a1, Connection c2, Airport a2
where a0.loc = “München” **and**
a0.code = c1.from **and**
c1.to = a1.code **and**
a1.code = c2.from **and**
c2.to = a2.code **and**
a2.loc = “New York”



- Bau von Auswertungsplänen mit Hilfe des *Iteratorkonzepts*



Implementierung der Selektion

a) **iterator** Scan_p

open

- Öffne Eingabe

next

- Hole solange nächstes Tupel, bis eines die Bedingung p erfüllt
- Gebe dieses Tupel zurück

close

- Schließe Eingabe

b) **iterator** IndexScan_p

open

- Schlage im Index das erste Tupel nach, das die Bedingung erfüllt
- Öffne Eingabe

next

- Gebe nächstes Tupel zurück, falls es die Bedingung p noch erfüllt

close

- Schließe Eingabe

Implementierung der Joinoperation

- Mengendifferenz und -durchschnitt können analog zum Join implementiert werden
- hier nur Equi-Joins betrachtet

Nested-Loop-Join:

for each $r \in R$
  for each $s \in S$
    if $r.A = s.B$ **then**
      $res := res \cup (r \times s)$

Iteratordarstellung:

iterator NestedLoop$_p$

open

- Öffne die linke Eingabe

next

- Rechte Eingabe geschlossen?
  - Öffne sie
- Fordere rechts solange Tupel an, bis Bedingung p erfüllt ist
- Sollte zwischendurch rechte Eingabe erschöpft sein
  - Schließe rechte Eingabe
  - Fordere nächstes Tupel der linken Eingabe an
  - Starte **next** neu
- Gib den Verbund von aktuellem linken und aktuellem rechten Tupel zurück

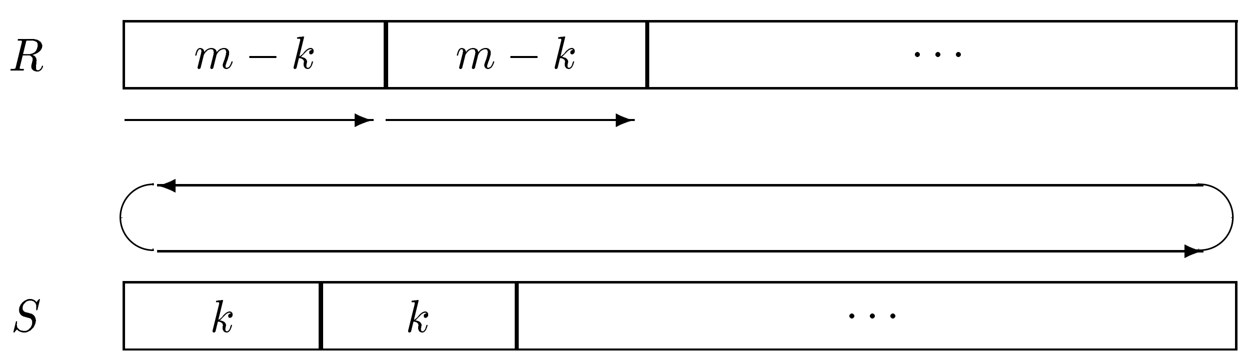
close

- Schließe beide Eingabequellen

Ein verfeinerter Join-Algorithmus

- Relationen sind *seitenweise* abgespeichert
- Es stehen m Pufferrahmen im Hauptspeicher zur Verfügung:
  - k für die innere Schleife des Nested Loop
  - $m - k$ für die äußere

Join von R und S:



Der Merge-Join

- Voraussetzung: R und S sind sortiert (notfalls vorher sortieren)

Beispiel:

R					S
	A			B	
	0	$\xleftarrow{z_r}$	$\xrightarrow{z_s}$	5	
	7			6	
$\dots$	7			7	$\dots$
	8			8	
	8			8	
	10			11	

Der Merge-Join

iterator MergeJoin_p

open

- Öffne beide Eingaben
- Setze *akt* auf linke Eingabe
- Markiere rechte Eingabe

next

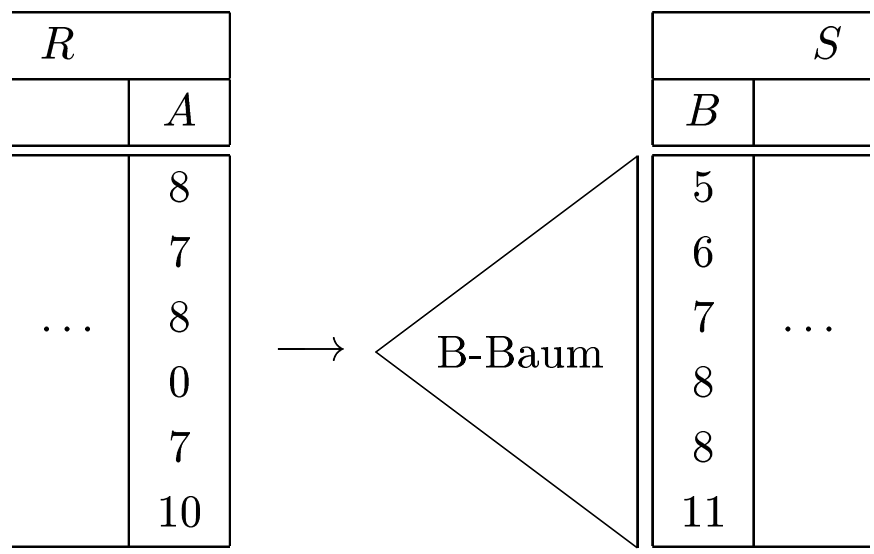
- Solange Bedingung nicht erfüllt
  - Setze *akt* auf Eingabe mit dem kleinsten anliegenden Wert im Joinattribut
  - Rufe **next** auf *akt* auf
  - Markiere andere Eingabe
- Gebe Verbund der aktuellen Tupel der linken und rechten Eingabe zurück
- Bewege andere Eingabe vor
- Ist Bedingung nicht mehr erfüllt oder andere Eingabe erschöpft?
  - Bewege *akt* vor
  - Wert des Joinattributes in *akt* verändert?
    - Nein, dann setze andere Eingabe auf Markierung zurück
    - Ansonsten markiere andere Eingabe

close

- Schließe beide Eingabequellen

Index-Join

Beispiel:



Iteratorstellung:

iterator IndexJoin$_p$

open

- Sei Index auf Joinattribut der rechten Eingabe vorhanden
- Öffne die linke Eingabe
- Hole erstes Tupel aus linker Eingabe
- Schlage Joinattributwert im Index nach

next

- Bilde Join, falls Index weiteres Tupel zu diesem Attributwert liefert
- Ansonsten bewege linke Eingabe vor und schlage Joinattributwert im Index nach

close

- Schließe die Eingabe

Hash-Join

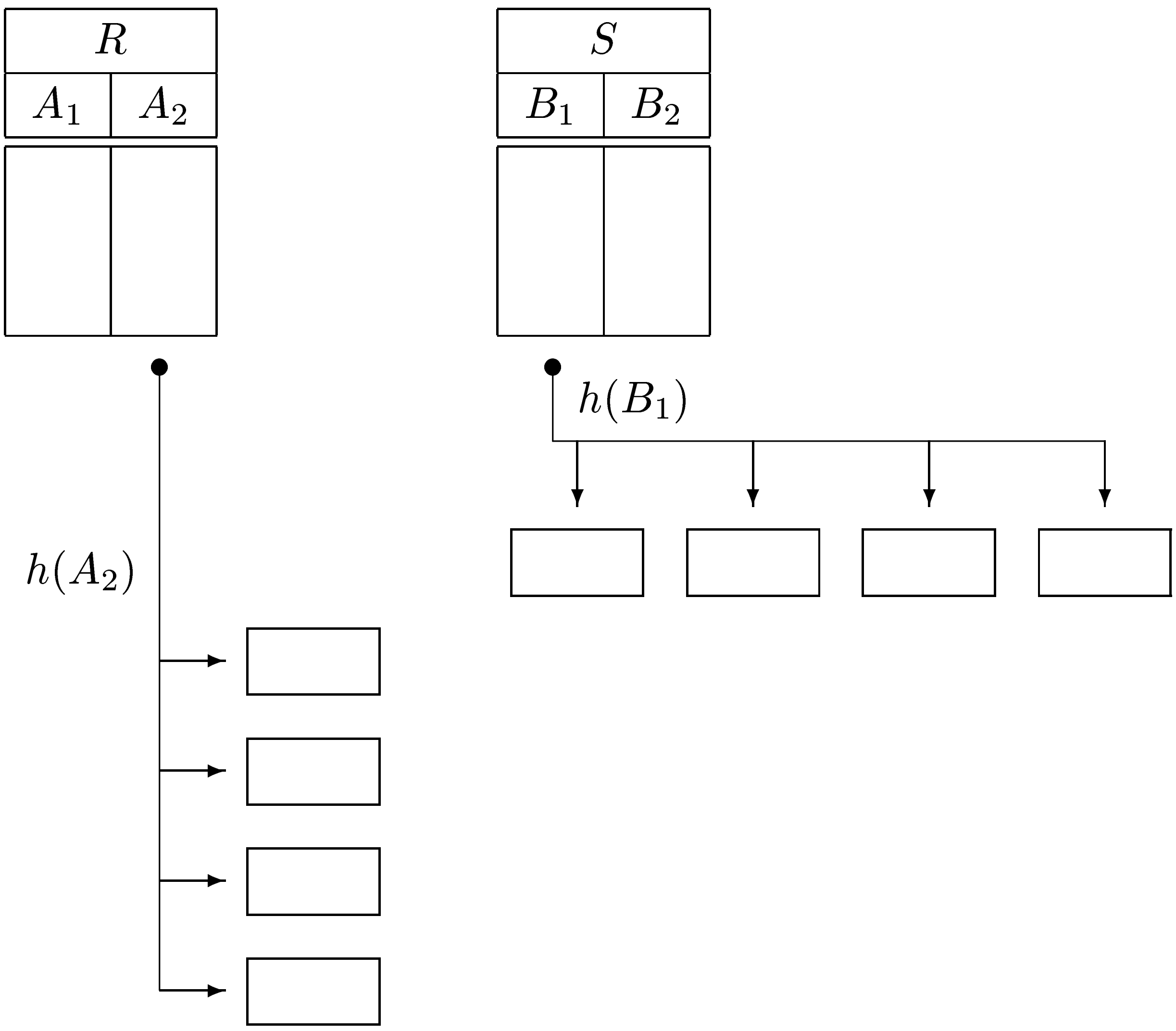
Nachteile des Index-Joins:

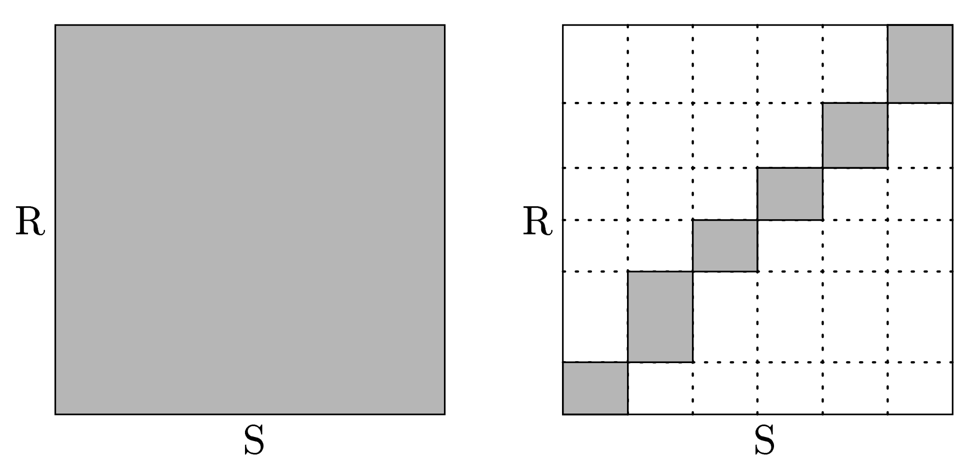
- auf Zwischenergebnissen existieren keine Indexstrukturen
- temporäres Anlegen i.A. zu aufwendig
- Nachschlagen im Index i.A. zu aufwendig

Idee:

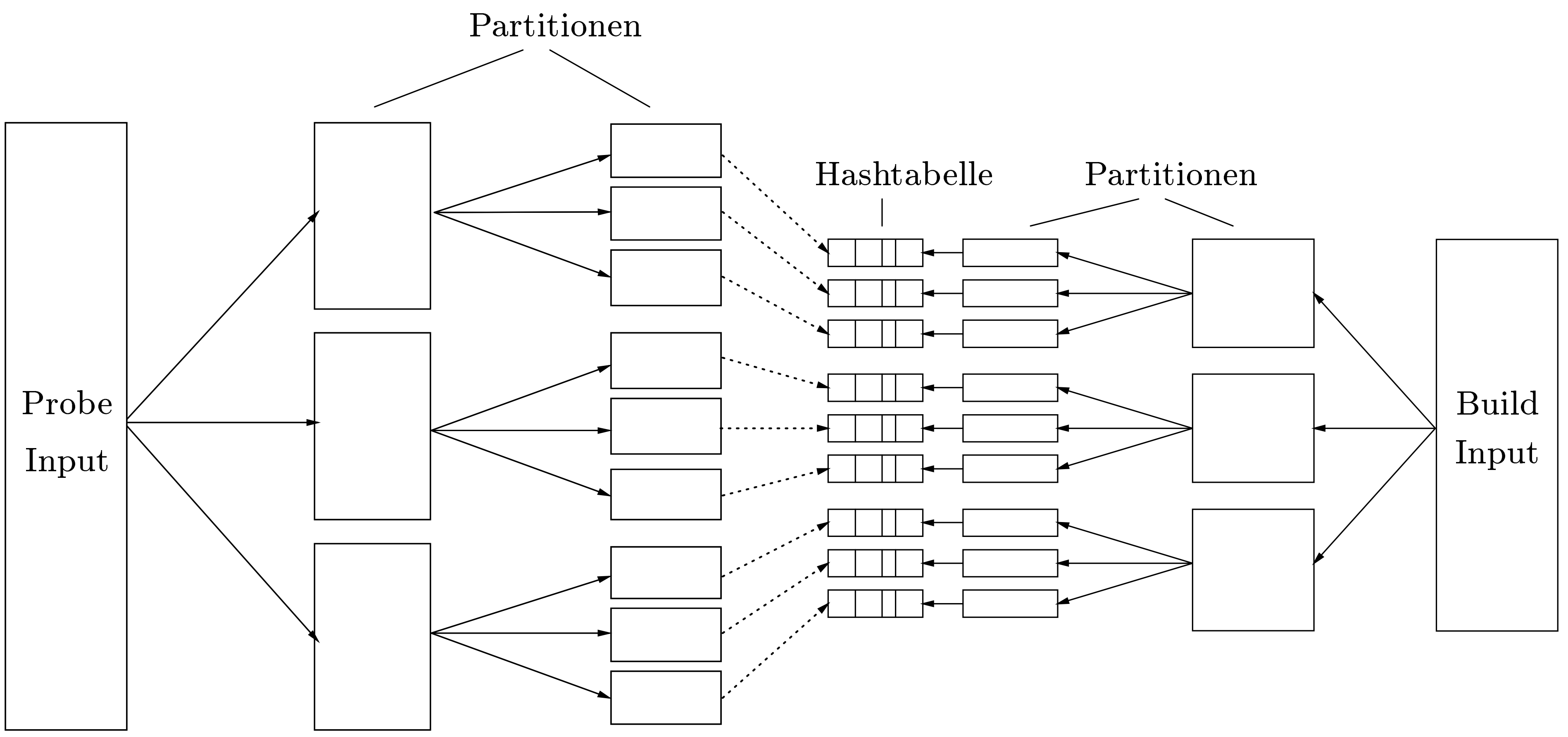
- Partitionieren der Relationen
- Anlegen von *Hauptspeicher*-Indexstrukturen (Hashtabellen) je Partition

Vergleich der Tupel in der “Diagonalen”

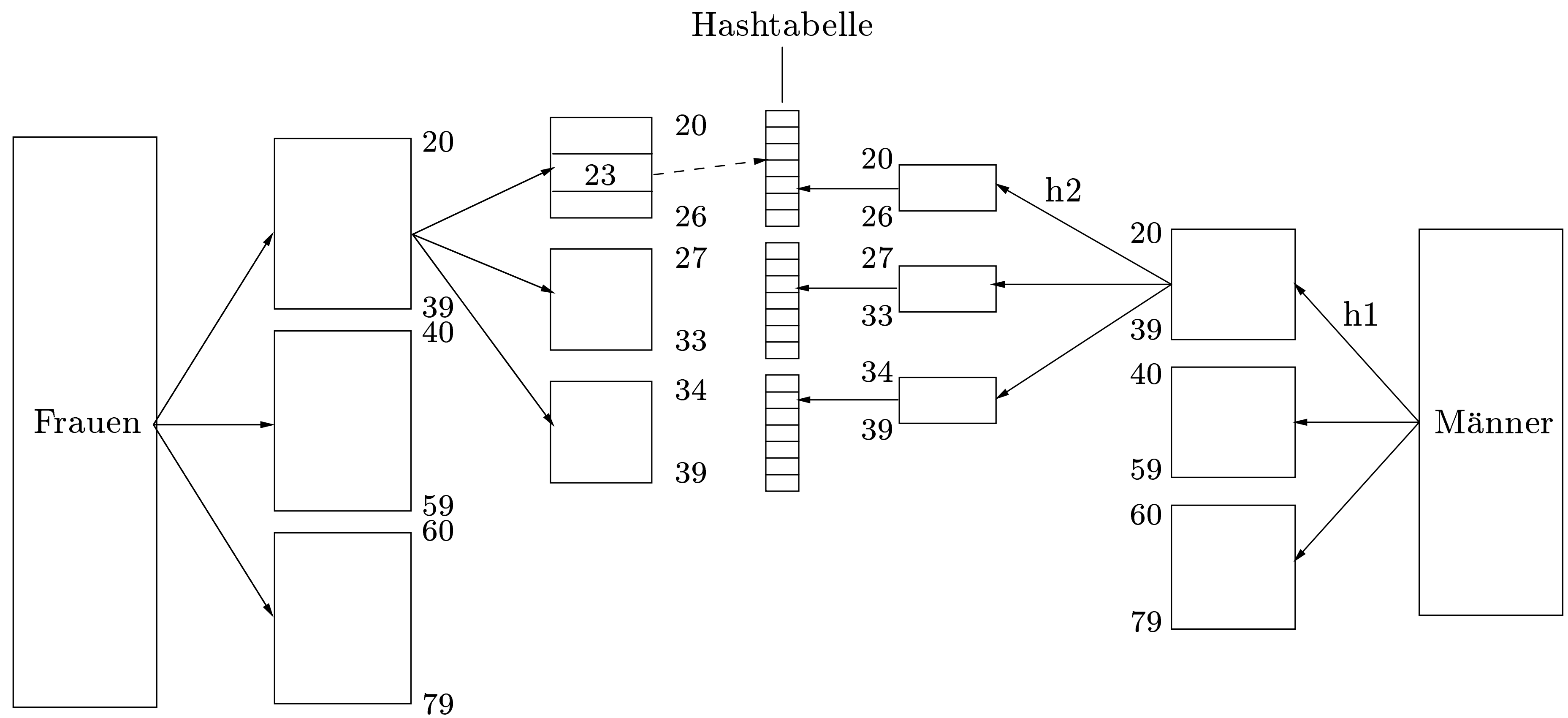




Partitionierung von Relationen



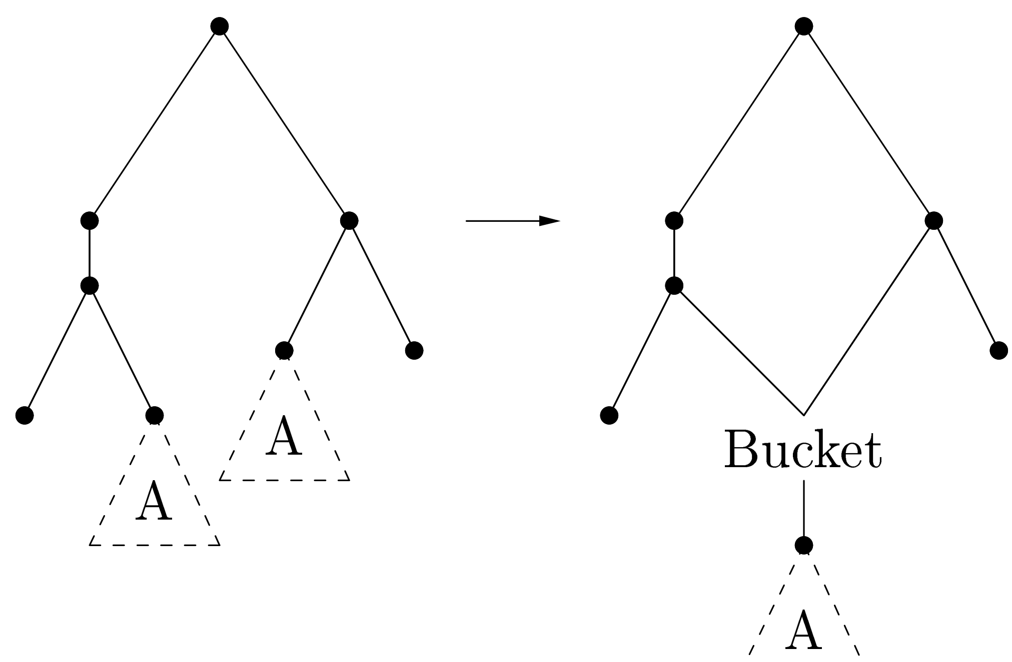
Demonstration der Partitionierung



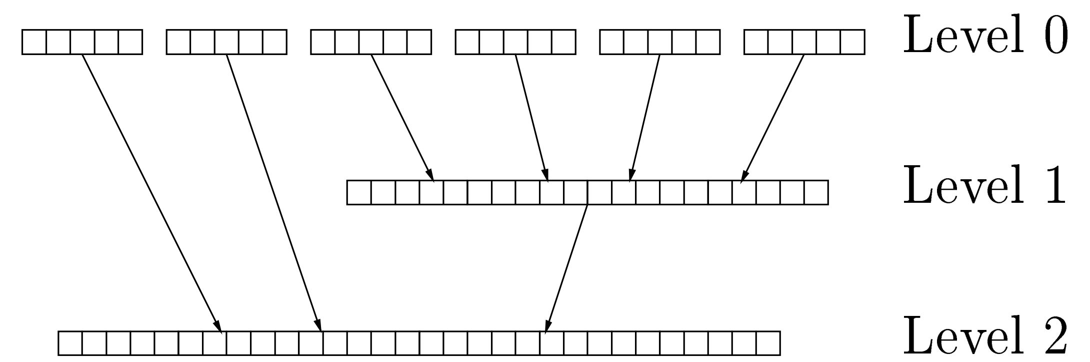
Zwischenspeicherung

Speicherung von Zwischenergebnissen notwendig, falls

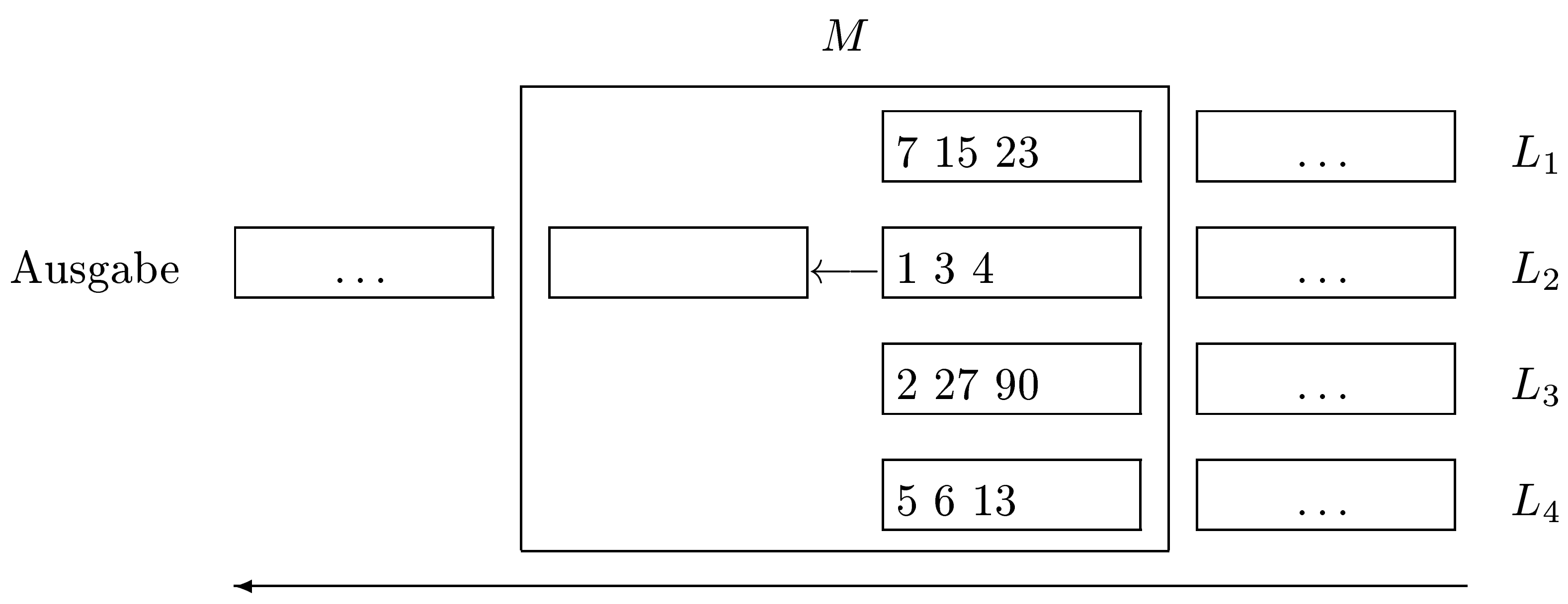
- mehrere Operationen mit hohem Hauptspeicherverbrauch vorkommen (z.B. Hash-Join)
- gemeinsame Teilausdrücke eliminiert werden sollen



Mergesort:



Ein Mischvorgang:



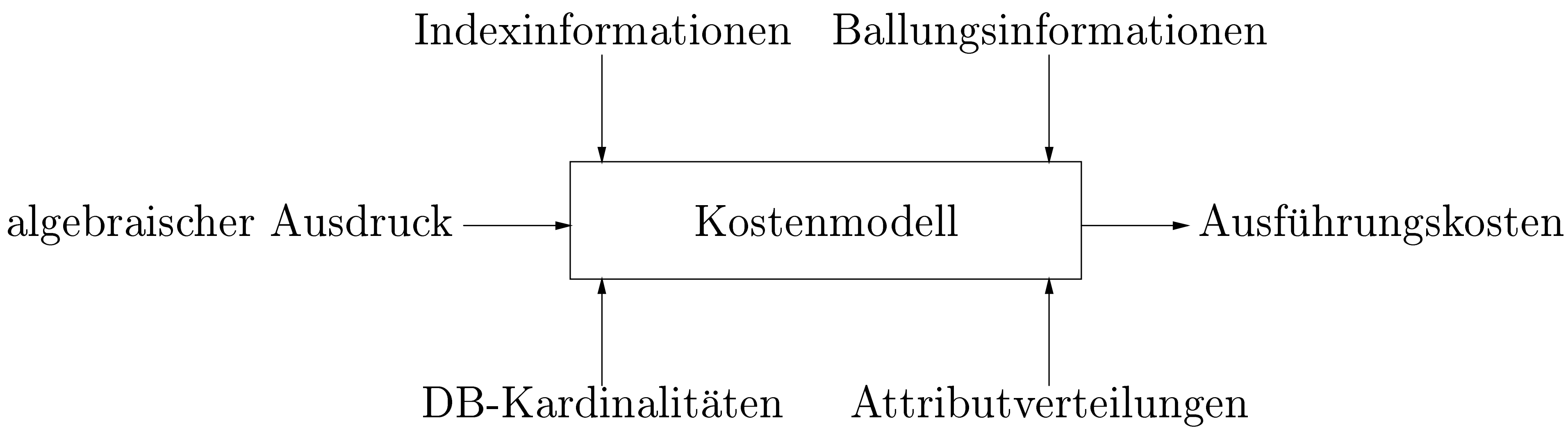
Replacement Selection

Ausgabe						Speicher				Eingabe						
						10	20	30	40	25	73	16	26	33	50	31
					10	20	25	30	40	73	16	26	33	50	31	
				10	20	25	30	40	73	16	26	33	50	31		
			10	20	25	(16)	30	40	73	26	33	50	31			
		10	20	25	30	(16)	(26)	40	73	33	50	31				
	10	20	25	30	40	(16)	(26)	(33)	73	50	31					
10	20	25	30	40	73	(16)	(26)	(33)	(50)	31						
					16	26	31	33	50							

- Vergrößerung der initialen Läufe um den Faktor 2 gegenüber:

  1. Laden des Hauptspeicherbereichs
  2. Sortieren mit Quicksort
  3. Ausschreiben des Laufs

Kostenmodelle



Selektivitäten

- Anteil der qualifizierenden Tupel einer Operation
- Selektion mit Bedingung p:

$$sel_p := \frac{|\sigma_p(R)|}{|R|}$$

- Join von R mit S:

$$sel_{RS} := \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$$

Abschätzung der Selektivität:

- $sel_{R.A=C} = \frac{1}{|R|}$
  falls A Schlüssel von R
- $sel_{R.A=C} = \frac{1}{i}$
  falls i die Anzahl der Attributwerte von $R.A$ ist (Gleichverteilung)
- $sel_{R.A=S.B} = \frac{1}{|R|}$
  bei Equijoin von R mit S über Fremdschlüssel in S

Ansonsten z.B. Stichprobenverfahren

Kostenabschätzungen

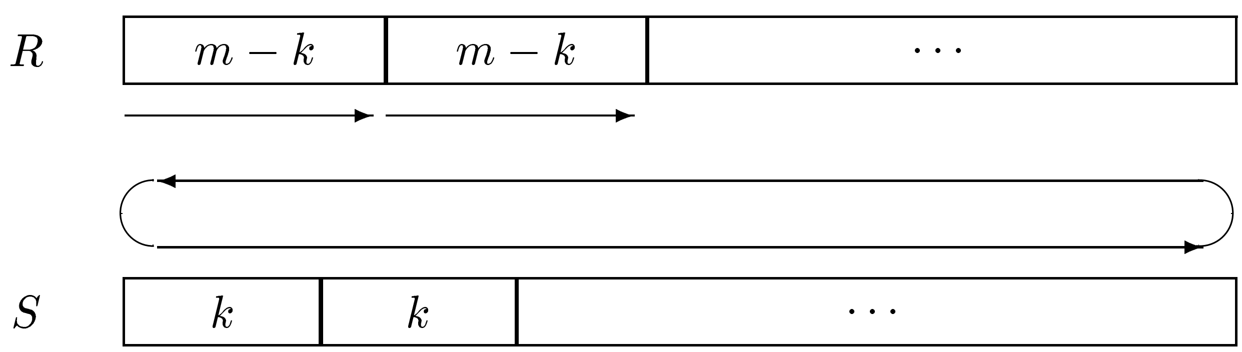
Selektion:

- Brute Force: Lesen aller Seiten von R
- B^+-Baum-Index: $t + \lceil sel_{A\theta c} \cdot b_R \rceil$
  - Absteigen der Indexstruktur
  - Lesen der qualifizierenden Tupel
- Hash-Index: für jeden die Bedingung erfüllenden Wert einen Look-up

Kostenabschätzungen

Blockorientierte Nested-Loops

Join:



- Durchlaufen aller Seiten von R: b_R
- Durchläufe der inneren Schleife: $\lceil b_R/(m-k) \rceil$
- Insgesamt: $b_R + k + \lceil b_R/(m-k) \rceil \cdot (b_S - k)$
- minimal, falls $k = 1$ und R die kleinere Relation

„Tuning“ von Datenbankanfragen

- viele DBMS-Produkte bieten unterschiedliche Optimierungslevel an
- Fast alle DBMS-Produkte haben beute u.a. einen kostenbasierten Optimierer
- Der kostenbasierte Optimierer benötigt Statistiken über die sepeicherten Daten, wie z.B.
  - Kardinalitäten der Relationen
  - Attributverteilungen (Histogramme) für Selektivitätsabschätzungen
  - Größe der Tupel
  - Clustering der Tupel
  - Indexkonfiguration
  - etc
- Die Datenbankadministratoren müssen die Generierung der Statistiken explizit anstoßen. Dazu dient z.B. in Oracle7 der Befehl

  analyze table Professoren **compute statistics for table**;

- in DB2:

  runstats on table ...

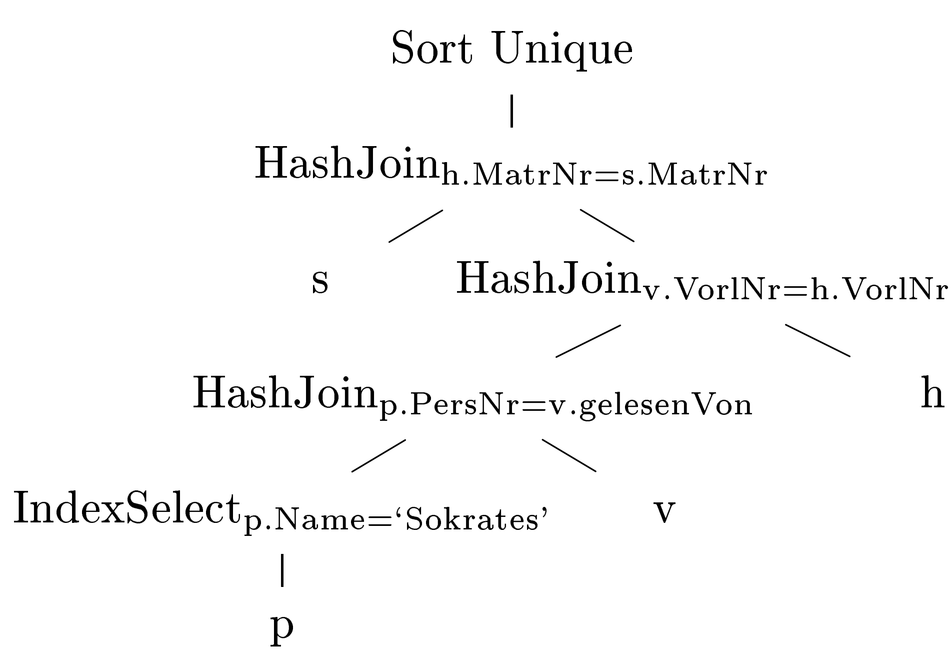
Analysieren der Auswertungspläne

- Man kann sich die generierten Anfrageauswertungspläne anzeigen lassen
- Dazu gibt es den **explain plan**-Befehl

```
explain plan for
   select distinct s.Semester
   from Studenten s, hören h, Vorlesungen v, Professoren p
   where p.Name = 'Sokrates' and v.gelesenVon = p.PersNr and
           v.VorlNr = h.VorlNr and h.MatrNr = s.MatrNr;
```

Beispiel-Plan

```
SELECT STATEMENT   Cost = 37710
  SORT UNIQUE
    HASH JOIN
      TABLE ACCESS FULL STUDENTEN
      HASH JOIN
        HASH JOIN
          TABLE ACCESS BY ROWID PROFESSOREN
            INDEX RANGE SCAN PROFNAMEINDEX
          TABLE ACCESS FULL VORLESUNGEN
        TABLE ACCESS FULL HOEREN
```



Anforderungen

1. **Recovery**, d.h. die Behebung von eingetretenen, oft unvermeidbaren Fehlersituationen.

2. **Synchronisation** von mehreren gleichzeitig auf der Datenbank ablaufenden Transaktionen.

Beispiel-Transaktion

1. Lese den Kontostand von A in die Variable a: **read**(A,a);

2. Reduziere den Kontostand um 50,– DM: $a := a - 50$;

3. Schreibe den neuen Kontostand in die Datenbasis: **write**(A,a);

4. Lese den Kontostand von B in die Variable b: **read**(B,b);

5. Erhöhe den Kontostand um 50,– DM: $b := b + 50$;

6. Schreibe den neuen Kontostand in die Datenbasis: **write**(B,b);

Anforderungen an die Transaktionsverwaltung

- gleichzeitig (nebenläufig) ablaufende Transaktionen
- Synchronisation
- Datenbanken gegen Soft- und Hardwarefehler schützen
- Abgeschlossene Transaktionen müssen erhalten bleiben
- Nicht abgeschlossene Transaktionen müssen vollständig revidiert (zurückgesetzt) werden.

Operationen auf Transaktions-Ebene

- **begin of transaction** (**BOT**): Mit diesem Befehl wird der Beginn einer eine Transaktion darstellenden Befehlsfolge gekennzeichnet.
- **commit**:
  - Alle Änderungen der Datenbasis werden durch diesen Befehl *festgeschrieben*,
  - sie werden **dauerhaft** in die Datenbank eingebaut.
- **abort**:
  - Selbstabbruch der Transaktion
  - Datenbanksystem muß sicherstellen, daß die Datenbasis wieder in den Zustand zurückgesetzt wird, der vor Beginn der Transaktionsausführung existierte.
- **define savepoint**:
  - Sicherungspunkt definiert, auf den sich die (noch aktive) Transaktion zurücksetzen läßt.
- **backup transaction**:
  - die noch aktive Transaktion wird auf den jüngsten – also den zuletzt angelegten – Sicherungspunkt zurückgesetzt.
  - evtl. ist auch ein Rücksetzen auf weiter zurückliegende Sicherungspunkte möglich.

Abschluß einer Transaktion

1. erfolgreicher Abschluß durch ein **commit**

2. erfolgloser Abschluß durch ein **abort** oder durch Fehler

BOT

op_1

op_2

$\vdots$

op_n

commit

BOT

op_1

op_2

$\vdots$

op_j

abort

BOT

op_1

op_2

$\vdots$

op_k

$\sim\sim\sim\sim\sim\sim$ Fehler

ACID-Paradigma

Atomicity (Atomarität)

- Transaktion ist kleinste, nicht mehr weiter zerlegbare Einheit
- Entweder werden alle Änderungen der Transaktion festgeschrieben oder gar keine
- Man kann sich dies auch als „alles-oder-nichts“-Prinzip merken

Consistency

- Transaktion hinterläßt einen konsistenten Datenbasiszustand
- Anderenfalls wird sie komplett (siehe *Atomarität*) zurückgesetzt
- Zwischenzustände während der TA-Bearbeitung dürfen inkonsistent sein
- Endzustand muß die im Schema definierten Konsistenzbedingungen (z.B. referentielle Integrität) erfüllen

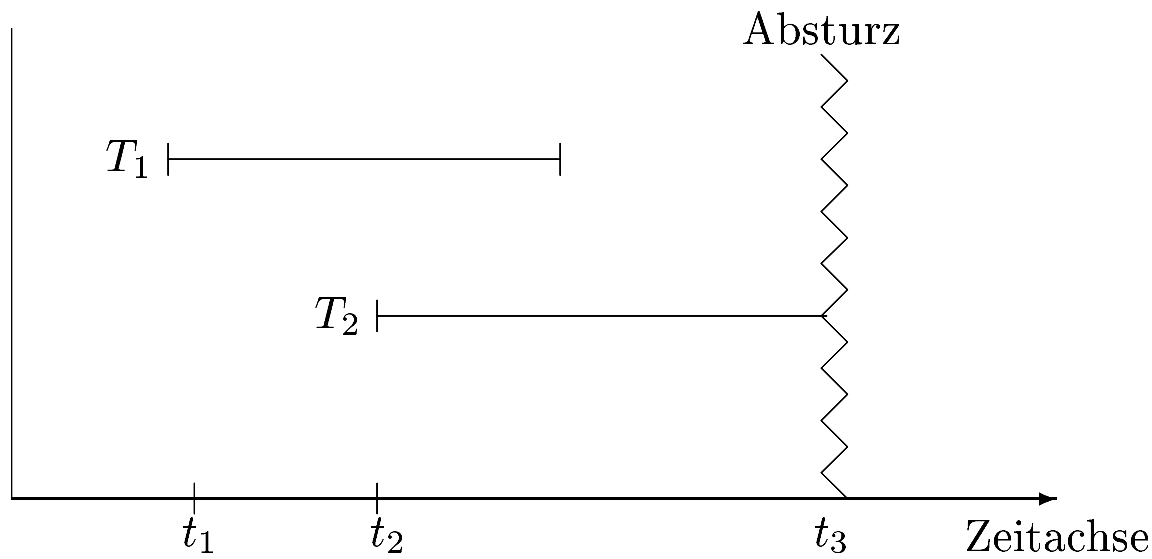
Isolation

- nebenläufig (parallel, gleichzeitig) ausgeführte Transaktionen dürfen sich nicht gegenseitig beeinflussen
- alle anderen parallel ausgeführten Transaktionen bzw. deren Effekte dürfen nicht sichtbar sein

Durability (Dauerhaftigkeit)

- Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten
- Transaktionsverwaltung muß sicherstellen, daß dies auch nach einem Systemfehler (Hardware oder Systemsoftware) gewährleistet ist
- Wirkungen einer einmal erfolgreich abgeschlossenen Transaktion kann nur durch eine sogenannte kompensierende Transaktion aufgehoben werden

Transaktionsbeginn und -ende



1. Die Wirkungen der zum Zeitpunkt t_3 abgeschlossenen Transaktion T_1 müssen in der Datenbasis vonhanden sein.

2. Die Wirkungen der zum Zeitpunkt des Systemabsturzes noch nicht abgeschlossenen Transaktion T_2 müssen vollständig aus der Datenbasis entfernt sein. Diese Transaktion kann man nur durch ein erneutes Starten durchführen.

Transaktionsverwaltung in SQL

- **commit work**:
  - Änderungen werden – falls keine Konsistenzverletzungen oder andere Probleme aufgedeckt werden – festgeschrieben.
  - Das Schlüsselwort **work** ist optional,
  - d.h. das Transaktionsende kann auch einfach mit **commit** „befohlen“ werden.
- **rollback work**:
  - Alle Änderungen sollen zurückgesetzt werden.
  - Anders als der **commit**-Befehl muß das DBMS die „erfolgreiche“ Ausführung eines **rollback**-Befehls immer garantieren können.
- **Beispiel-Transaktion**

insert into Vorlesungen
    values (5275, 'Kernphysik', 3, 2141);

insert into Professoren
    values (2141, 'Meitner', 'C4', 205);

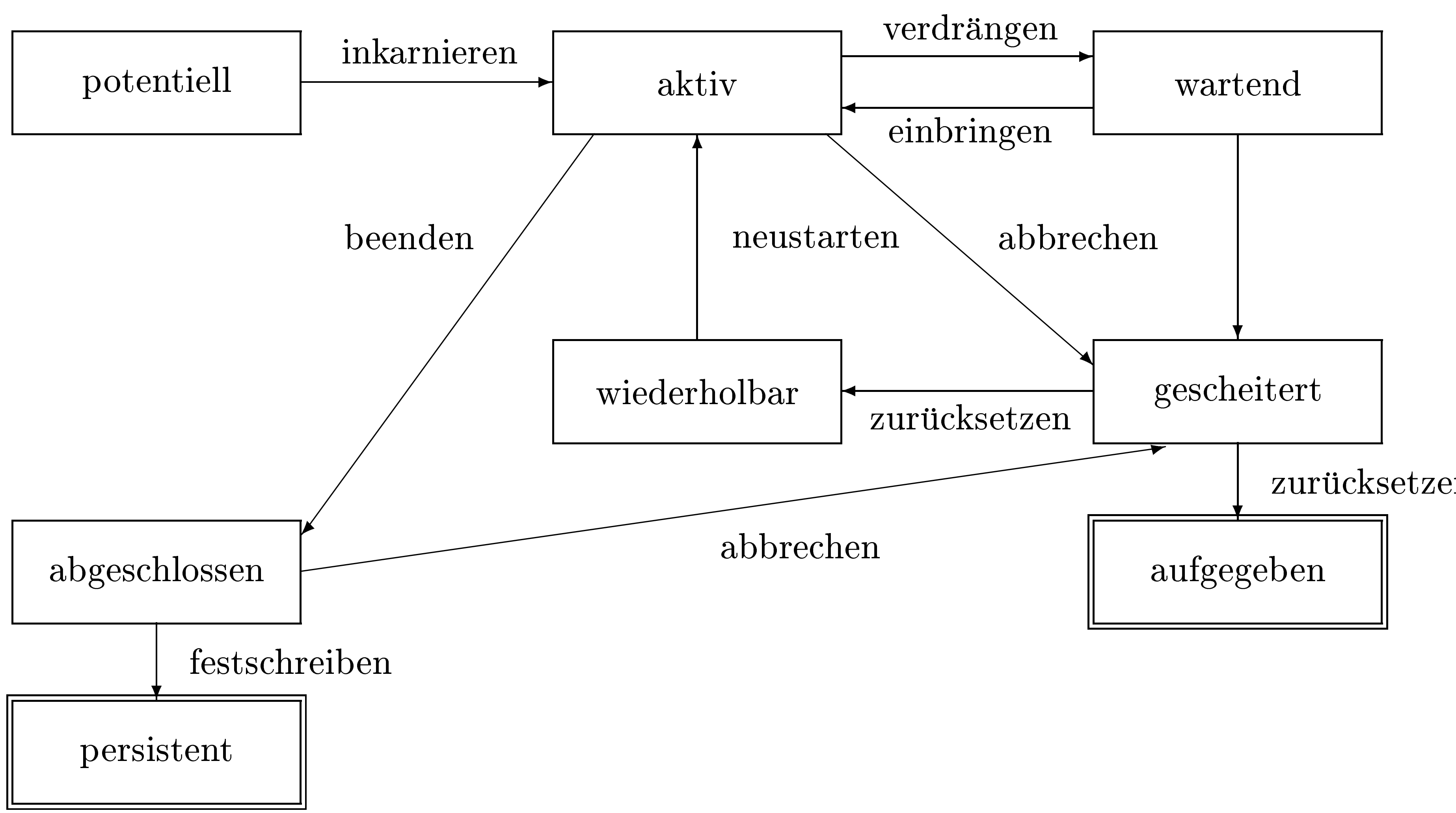
commit work

- Man beachte: ein **commit**-Versuch nach dem ersten **insert** könnte nicht erfolgreich durchgeführt werden, da zu diesem Zeitpunkt die referentielle Integrität verletzt ist.

Zustandsübergänge einer Transaktion

- *potentiell*:
- *aktiv*:
- *wartend*:
- *abgeschlossen*:
- *persistent*:
- *gescheitert*:
- *wiederholbar*:
- *aufgegeben*:

Zustandsübergangs-Diagramm für Transaktionen

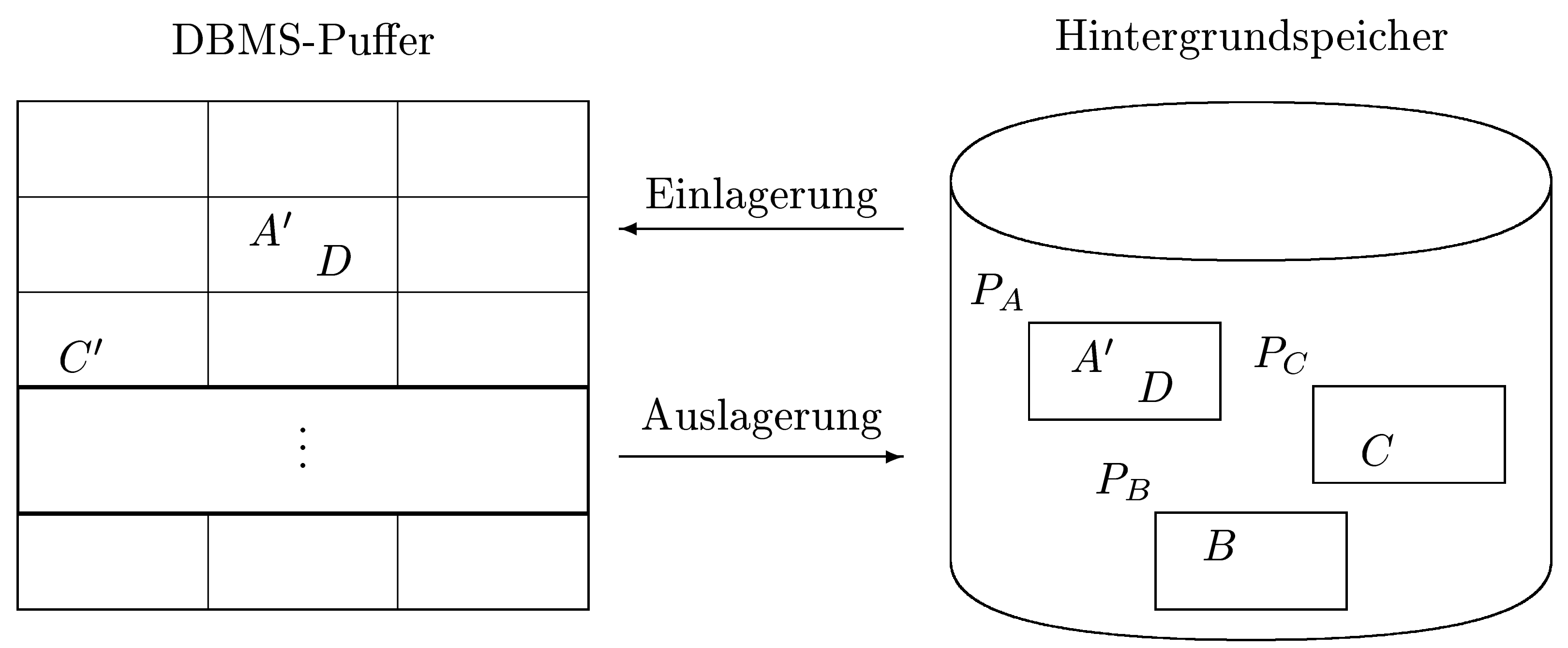


Fehlerbehandlung (Recovery)

Fehlerklassifikation

1. Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion
   - Wirkung muß zurückgesetzt werden
   - $R1$-Recovery
2. Fehler mit Hauptspeicherverlust
   - abgeschlossene TAs müssen erhalten bleiben (R2-Recovery)
   - noch nicht abgeschlossene TAs müssen zurückgesetzt werden (R3-Recovery)
3. Fehler mit Hintergrundspeicherverlust
   - R4-Recovery

Zweistufige Speicherhierarchie



Ersetzung von Puffer-Seiten

- $\neg$*steal*: Bei dieser Strategie wird die Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ausgeschlossen.
- *steal*: Jede nicht fixierte Seite ist prinzipiell ein Kandidat für die Ersetzung, falls neue Seiten eingelagert werden müssen.

Einbringen von Änderungen abgeschlossener TAs

- *force*-Strategie: Änderungen werden zum Transaktionsende auf den Hintergrundspeicher geschrieben.
- $\neg$*force*-Strategie: geänderte Seiten können im Puffer verbleiben.

Auswirkung auf Recovery

	force	$\neg$force
$\neg$steal	• kein Redo • kein Undo	• Redo • kein Undo
steal	• kein Redo • Undo	• Redo • Undo

Einbringstrategie

Update in Place

- jede Seite hat genau eine „Heimat“ auf dem Hintergrundspeicher
- der alte Zustand der Seite wird überschrieben

Twin-Block-Verfahren
Anordnung der Seiten P_A, P_B und P_C.

P_A^0	P_A^1	P_B^0	P_B^1	P_C^0	P_C^1	
						...

Schattenspeicherkonzept

- nur geänderte Seiten werden dupliziert
- weniger Redundanz als beim Twin-Block-Verfahren

Hier zugrunde gelegte Systemkonfiguration

- *steal*:
  - „dreckige Seiten“ können in die Datenbank (auf Platte) geschrieben werden

- $\neg$*force*:
  - geänderte Seiten sind möglicherweise noch nicht auf die Platte geschrieben

- *update-in-place*:
  - Es gibt von jeder Seite nur eine Kopie auf der Platte

- *Kleine Sperrgranulate*:
  - auf Satzebene
  - also kann eine Seite gleichzeitig „dreckige“ Daten (einer noch nicht abgeschlossenen TA) und „commited updates“ enthalten
  - das gilt sowohl für Puffer – als auch Datenbankseiten

Protokollierung von Änderungsoperationen

Struktur der Log-Einträge

[LSN, TransaktionsID, PageID, Redo, Undo, PrevLSN]

- *LSN* (*Log Sequence Number*),
  - eine eindeutige Kennung des Log-Eintrags.
  - *LSNs* müssen monoton aufsteigend vergeben werden,
  - die chronologische Reihenfolge der Protokolleinträge kann dadurch ermittelt werden.
- *Transaktionskennung TA* der Transaktion, die die Änderung durchgeführt hat.
- *PageID*
  - die Kennung der Seite, auf der die Änderungsoperation vollzogen wurde.
  - Wenn eine Änderung mehr als eine Seite betrifft, müssen entsprechend viele Log-Einträge generiert werden.

- Die *Redo*-Information gibt an, wie die Änderung nachvollzogen werden kann.
- Die *Undo*-Information beschreibt, wie die Änderung rückgängig gemacht werden kann.
- *PrevLSN*, einen Zeiger auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion. Diesen Eintrag benötigt man aus Effizienzgründen.

Beispiel einer Log-Datei

Schritt	T_1	T_2	Log
			[LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	**BOT**		$[\#1, T_1,$**BOT**$, 0]$
2.	$r(A, a_1)$		
3.		**BOT**	$[\#2, T_2,$ **BOT**$, 0]$
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		$[\#3, T_1, P_A, A{-}{=}50, A{+}{=}50, \#1]$
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	$[\#4, T_2, P_C, C{+}{=}100, C{-}{=}100, \#2]$
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		$[\#5, T_1, P_B, B{+}{=}50, B{-}{=}50, \#3]$
12.	**commit**		$[\#6, T_1,$ **commit**$, \#5]$
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	$[\#7, T_2, P_A, A{-}{=}100, A{+}{=}100, \#4]$
16.		**commit**	$[\#8, T_2,$ **commit**$, \#7\ ]$

Logische oder physische Protokollierung

Physische Protokollierung

Es werden Inhalte/Zustände protokolliert:

1. **before-image** enthält den Zustand vor Ausführung der Operation
2. **after-image** enthält den Zustand nach Ausführung der Operation

Logische Protokollierung

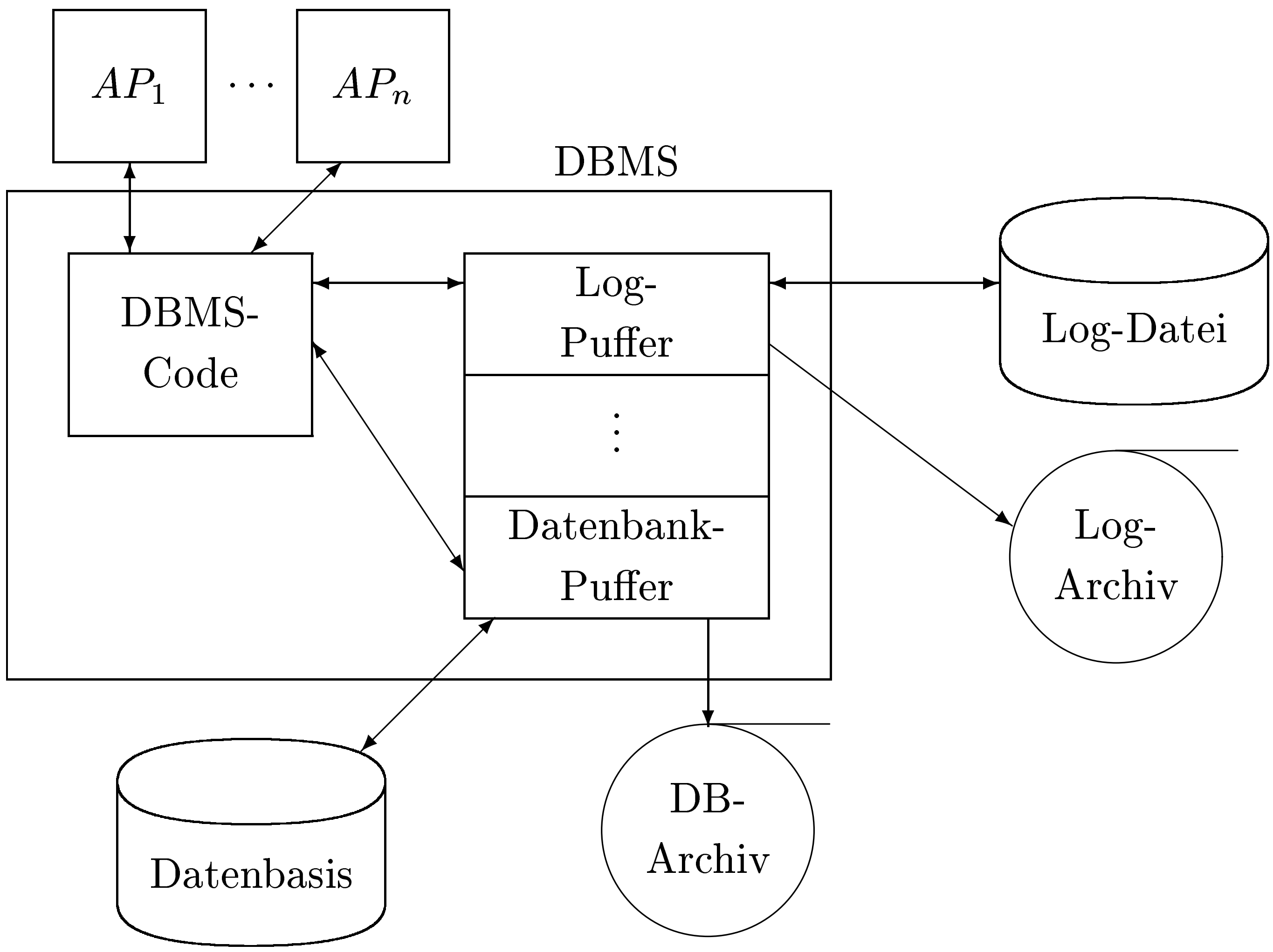
- das *Before-Image* wird durch Ausführung des *Undo*-Codes aus dem *After-Image* generiert und
- das *After-Image* durch Ausführung des *Redo*-Codes aus dem *Before-Image* berechnet.

Speicherung der Seiten-LSN

Die "Herausforderung" besteht darin, beim Wiederanlauf zu entscheiden, ob man das Before- oder das After-Image auf dem Hintergrundspeicher vorgefunden hat.

Dazu wird auf jeder Seite die LSN des jüngsten diese Seite betreffenden Log-Eintrags gespeichert.

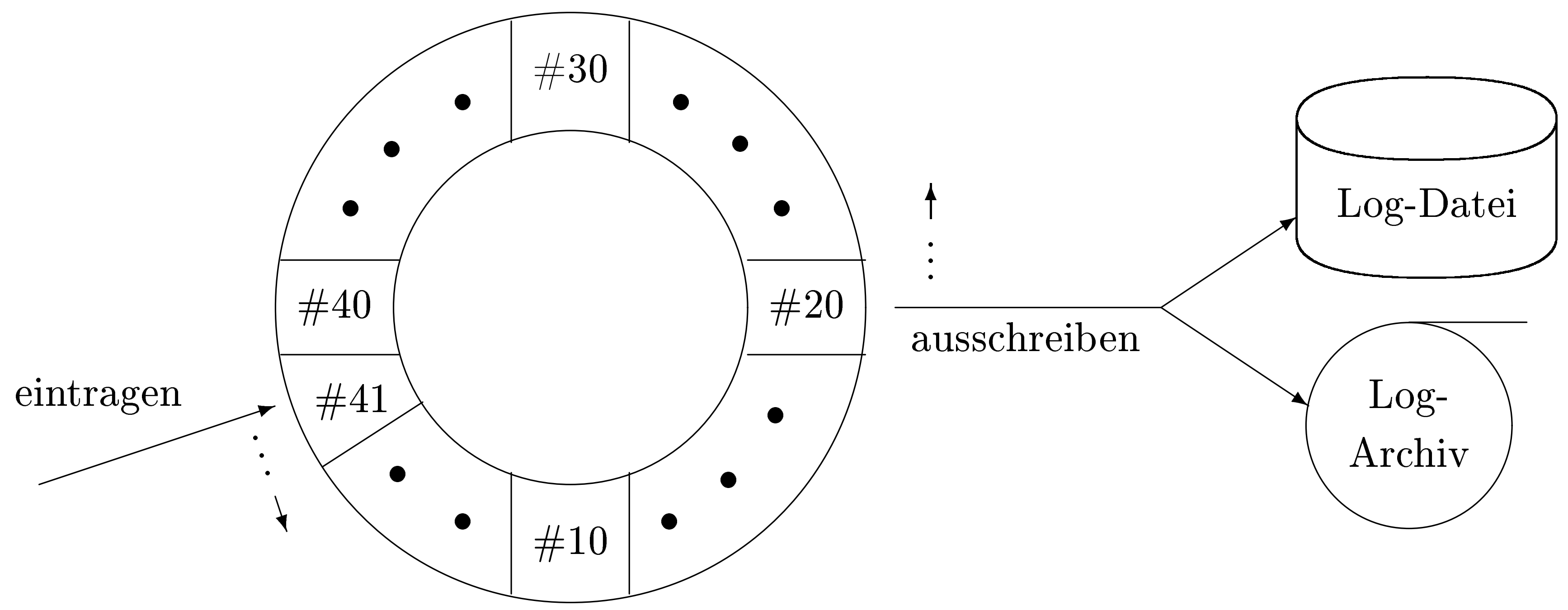
Schreiben der Log-Information



- Die Log-Information wird zweimal geschrieben

  1. Log-Datei für schnellen Zugriff
     - R1, R2 und R3-Recovery
  2. Log-Archiv
     - R4-Recovery

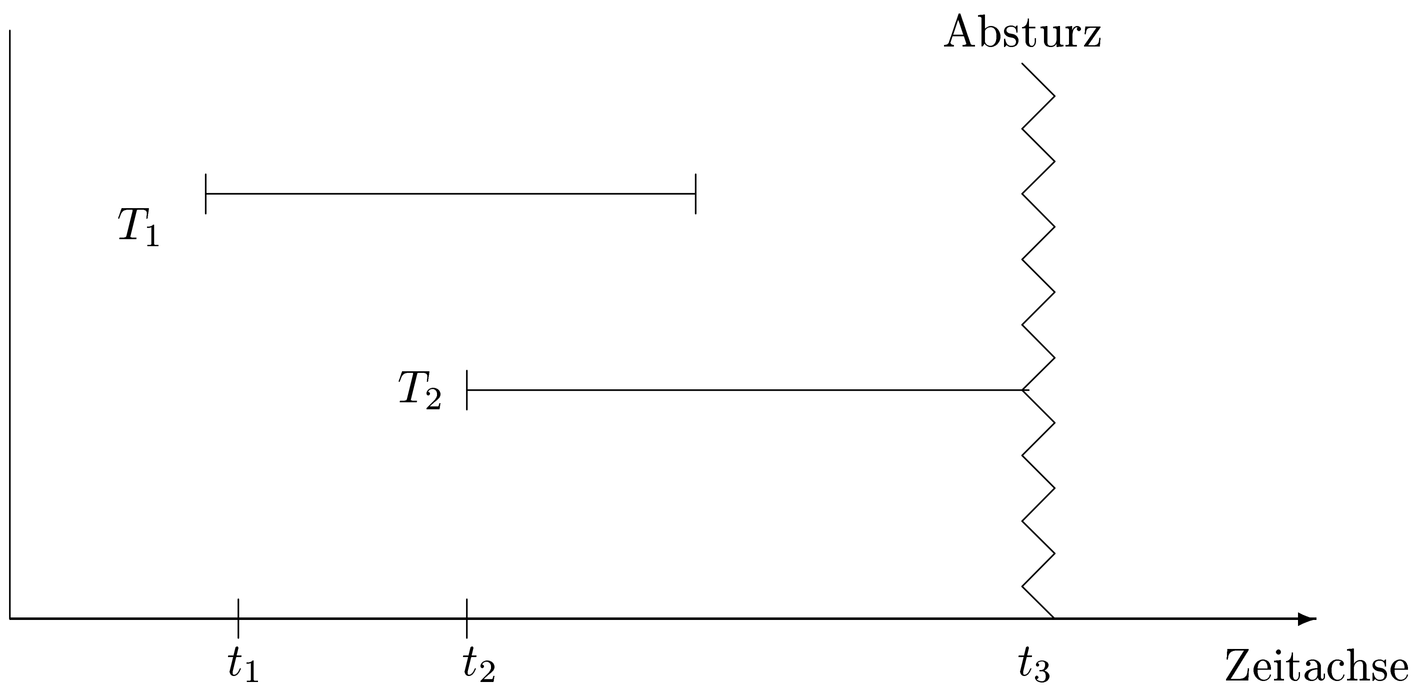
Anordnung des Log-Ringpuffers



Write Ahead Log-Prinzip

1. Bevor eine Transaktion festgeschrieben (**committed**) wird, müssen alle „zu ihr gehörenden“ Log-Einträge ausgeschrieben werden.
2. Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in das temporäre und das Log-Archiv ausgeschrieben werden.

Transaktionsbeginn und -ende relativ zu einem Systemabsturz

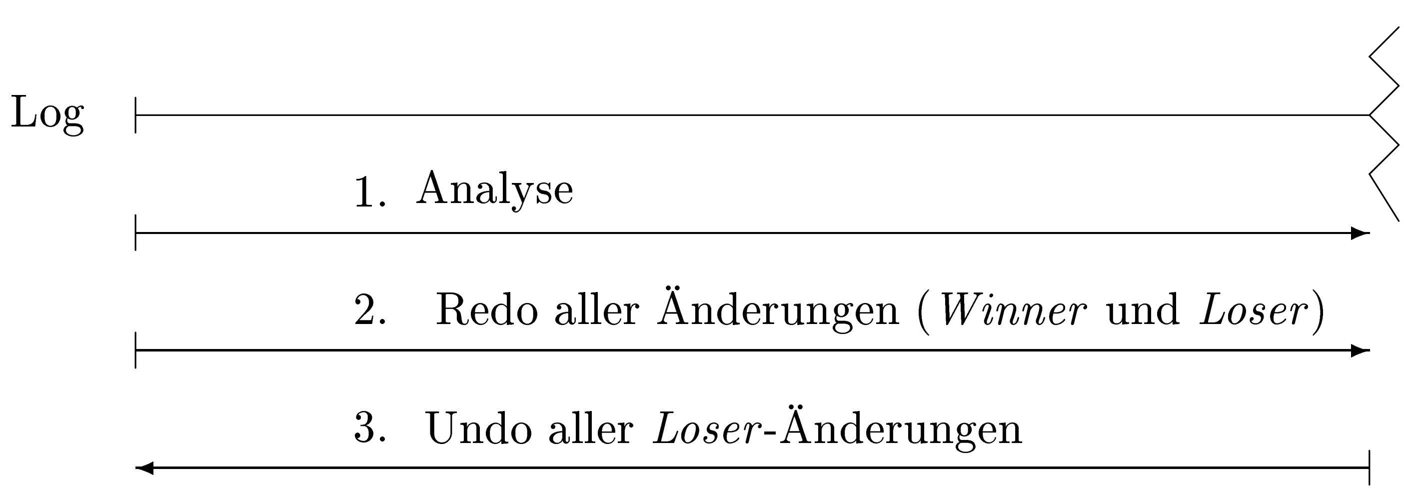


- Transaktionen der Art T_1 müssen hinsichlich ihrer Wirkung vollständig nachvollzogen werden. Transaktionen dieser Art nennt man *Winner*.
- Transaktionen, die wie T_2 zum Zeitpunkt des Absturzes noch aktiv waren, müssen rückgängig gemacht werden. Diese Transaktionen bezeichnen wir als *Loser*.

Drei Phasen des Wiederanlaufs

1. *Analyse*:
   - Die temporäre Log-Datei wird von Anfang bis Ende analysiert,
   - Ermittlung der *Winner*-Menge von Transaktionen des Typs T_1
   - Ermittlung der *Loser*-Menge von Transaktionen der Art T_2.
2. *Wiederholung der Historie*:
   - *alle* protokollierten Änderungen werden in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht.
3. *Undo der Loser*:
   - Die Änderungoperationen der *Loser*-Transaktionen werden in umgekehrter Reihenfoge ihrer ursprünglichen Ausführung rückgängig gemacht.

Wiederanlauf in drei Phasen

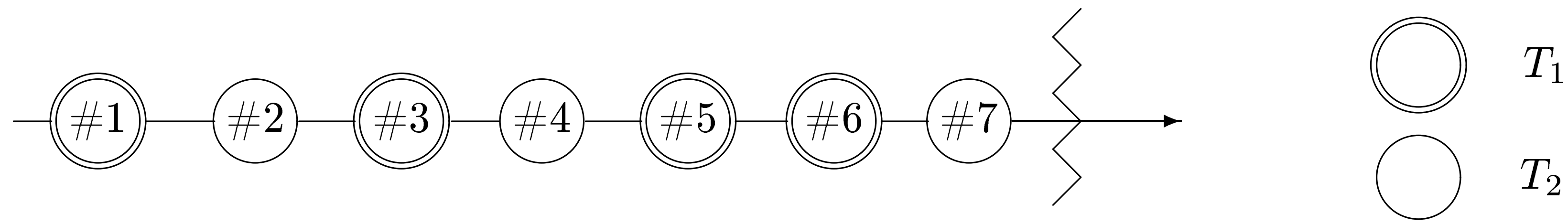


Fehlertoleranz (Idempotenz) des Wiederanlaufs

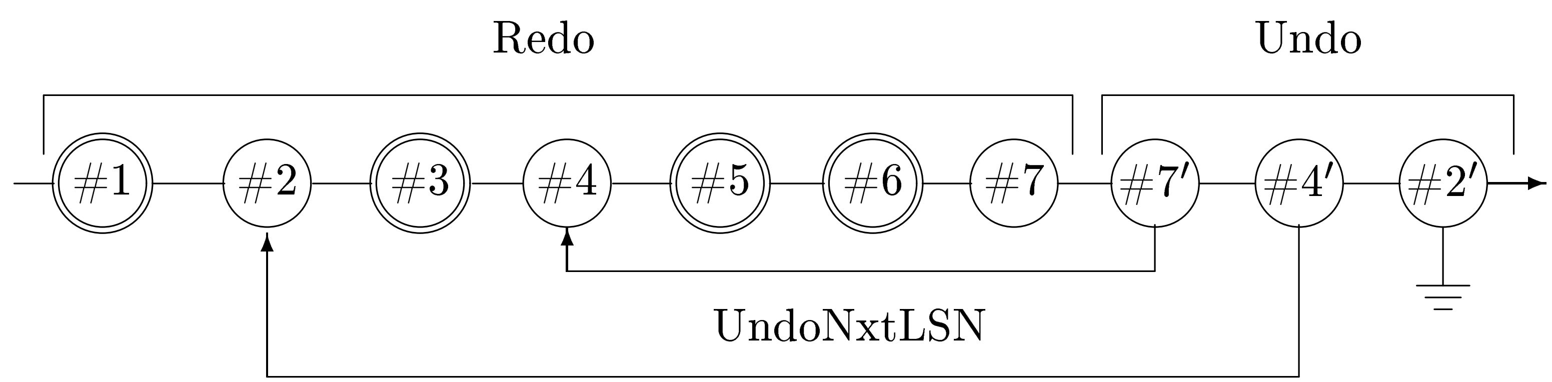
$$\begin{aligned} undo(undo(\cdots(undo(a))\cdots)) &= undo(a) \\ redo(redo(\cdots(redo(a))\cdots)) &= redo(a) \end{aligned}$$

- auch während der Recoveryphase kann des System abstürzen

Kompensationseinträge im Log



Wiederanlauf und Log



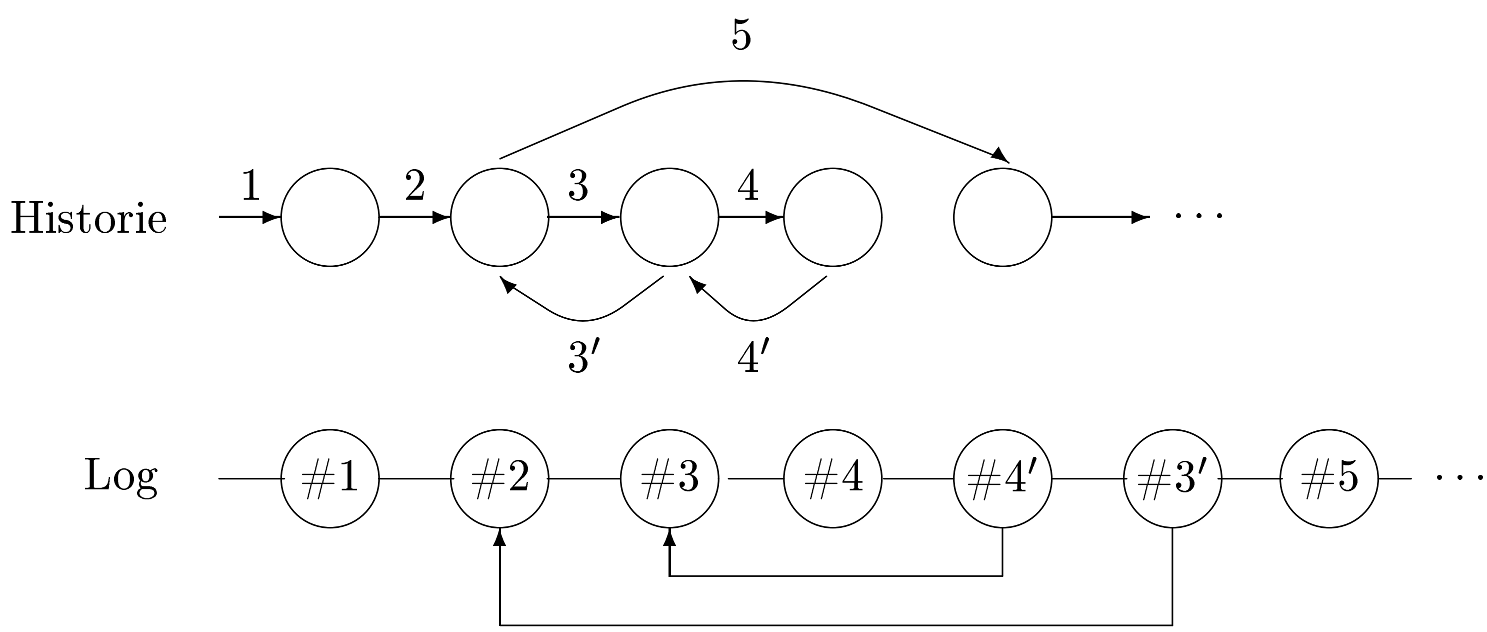
- Kompensationseinträge (CLR: compensating log record) für rückgängig gemachte Änderungen.
  - #7 ist CLR für #7
  - #4 ist CLR für #4

Logeinträge nach abgeschlossenem Wiederanlauf

$$[\#1, T_1, \mathbf{BOT}, 0]$$
$$[\#2, T_2, \mathbf{BOT}, 0]$$
$$[\#3, T_1, P_A, A\text{–=}50, A\text{+=}50, \#1]$$
$$[\#4, T_2, P_C, C\text{+=}100, C\text{–=}100, \#2]$$
$$[\#5, T_1, P_B, B\text{+=}50, B\text{–=}50, \#3]$$
$$[\#6, T_1, \mathbf{commit}, \#5]$$
$$[\#7, T_2, P_A, A\text{–=}100, A\text{+=}100, \#4]$$
$$\langle \#7', T_2, P_A, A\text{+=}100, \#7, \#4 \rangle$$
$$\langle \#4', T_2, P_C, C\text{–=}100, \#7', \#2 \rangle$$
$$\langle \#2', T_2, -, -, \#4', 0 \rangle$$

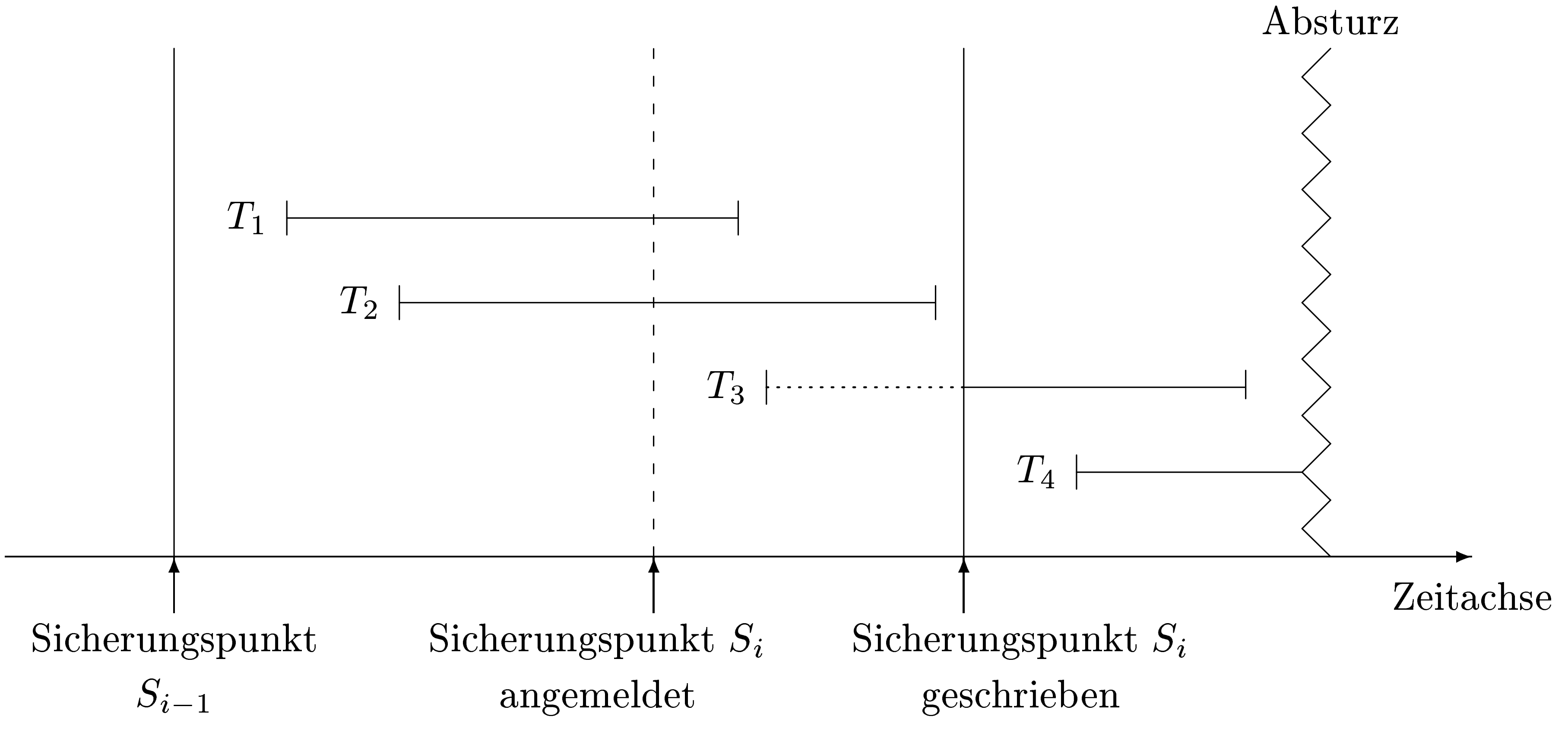
- CLRs sind durch spitze Klammern $\langle \ldots \rangle$ gekennzeichnet.
- der Aufbau einer CLR ist wie folgt
  - LSN
  - TA-Identifikator
  - betroffene Seite
  - Redo-Information
  - PrevLSN
  - UndoNxtLSN (Verweis auf die nächste rückgängig zu machende Änderung)
- CLRs enthalten keine Undo-Information
  - warum nicht?

Partielles Zurücksetzen einer Transaktion

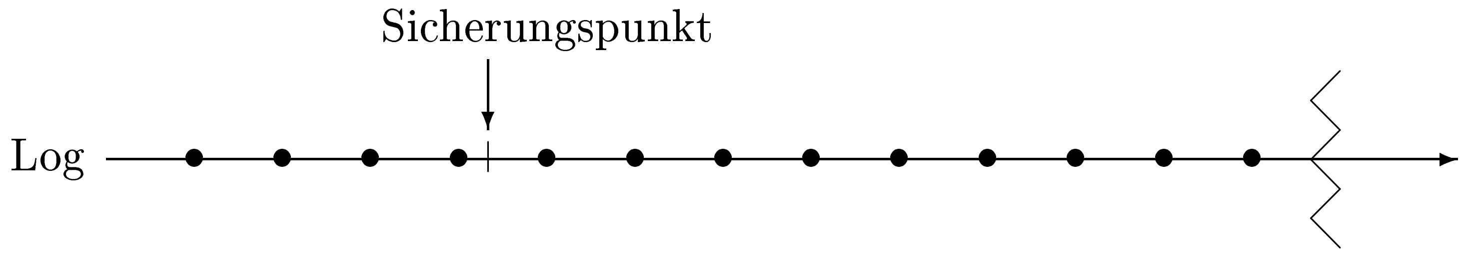


- Schritte 3 und 4 werden zurückgenommen
- notwendig für die Realisierung von Sicherungspunkten innerhalb einer TA

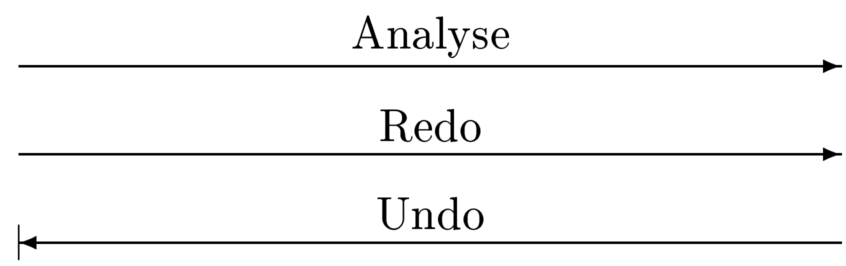
Transaktionskonsistente Sicherungspunkte



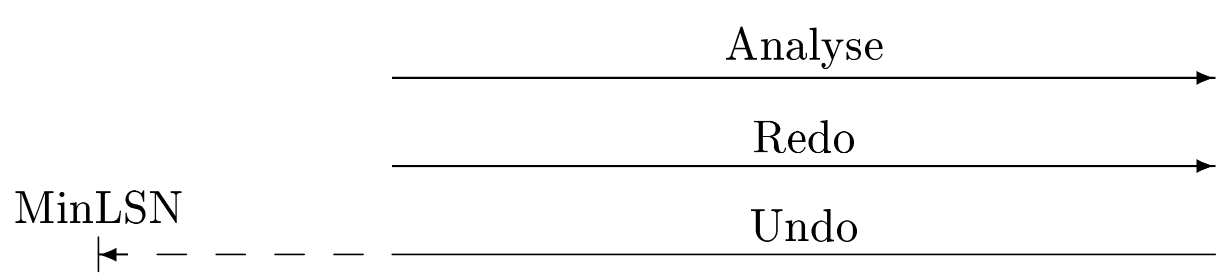
Drei unterschiedliche Sicherungspunkt-Qualitäten



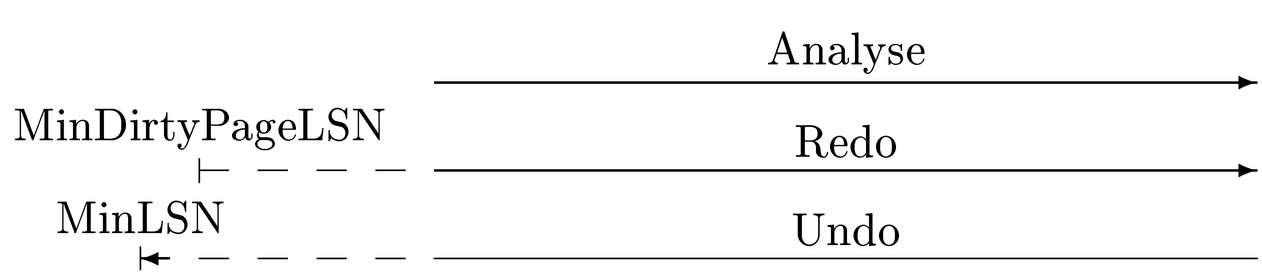
(a) **transaktionskonsistent**



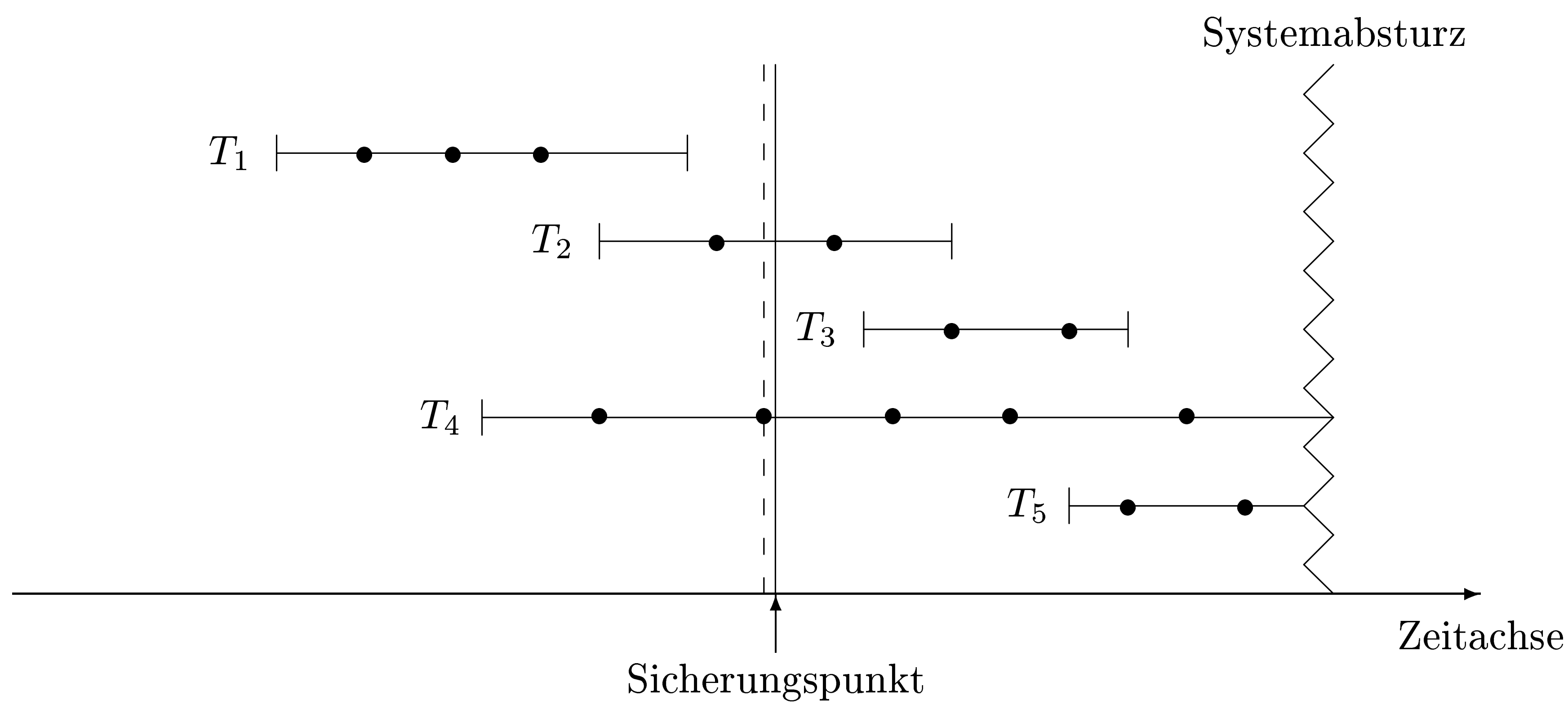
(b) **aktionskonsistent**



(c) **unscharf (fuzzy)**



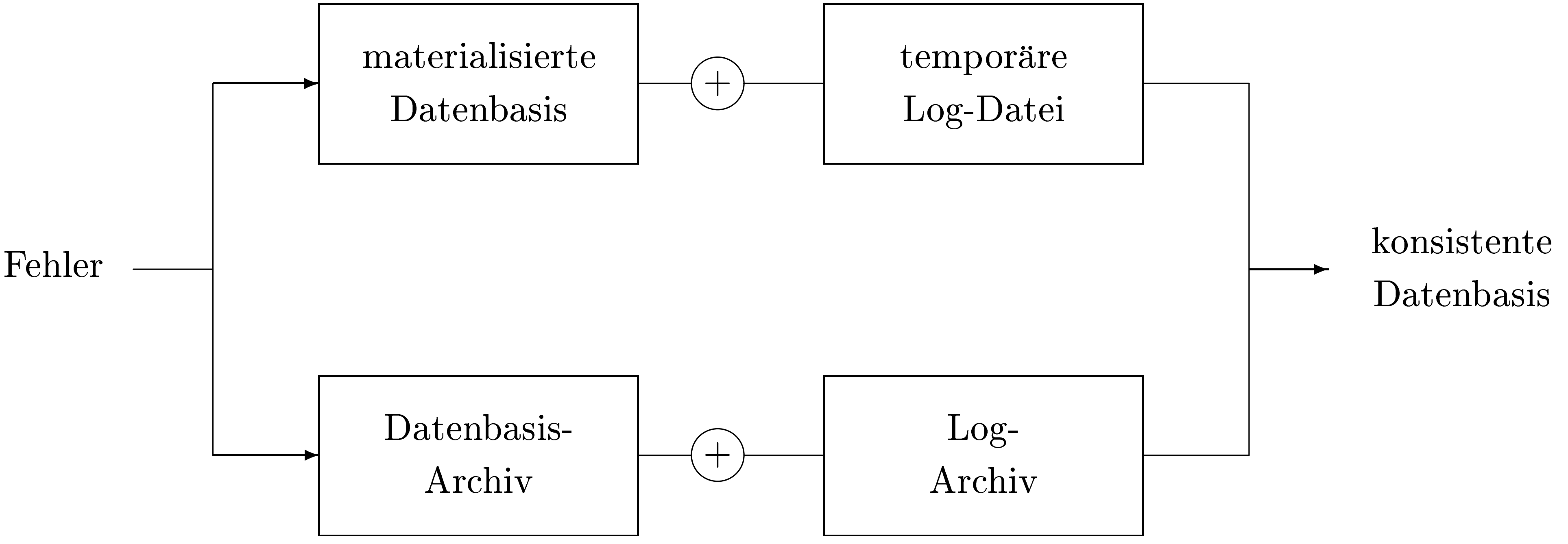
Transaktionsausführung relativ zu einem aktionskonsistenten Sicherungspunkt und einem Systemabsturz



Unscharfe (fuzzy) Sicherungspunkte

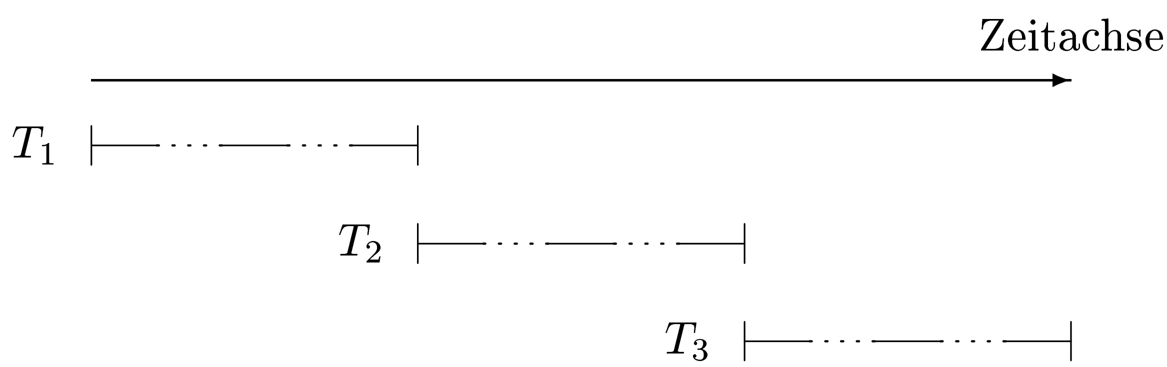
- modifizierte Seiten werden nicht ausgeschrieben
- nur deren Kennung wird ausgeschrieben
  - *Dirty Pages*=Menge der modifizierten Seiten
- *MinDirtyPageLSN*: die minimale LSN, deren Änderungen noch nicht ausgeschrieben wurde
- *MinLSN*: die kleinste LSN der zum Sicherungszeitpunkt aktiven TAs

Recovery nach einem Verlust der materialisierten Datenbasis

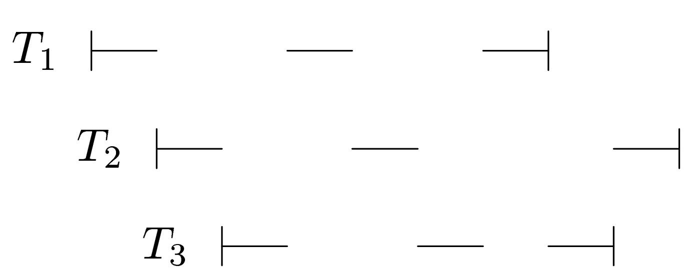


Ausführung der drei Transaktionen T_1, T_2 und T_3:

(a) im Einbenutzerbetrieb und



(b) im (verzahnten) Mehrbenutzerbetrieb (gestrichelte Linien repräsentieren Wartezeiten)



Fehler bei unkontrolliertem Mehrbenutzerbetrieb

Verlorengegangene Änderungen (*lost update*)

Schritt	T_1	T_2
1.	$\text{read}(A, a_1)$	
2.	$a_1 := a_1 - 300$	
3.		$\text{read}(A, a_2)$
4.		$a_2 := a_2 * 1.03$
5.		$\text{write}(A, a_2)$
6.	$\text{write}(A, a_1)$	
7.	$\text{read}(B, b_1)$	
8.	$b_1 := b_1 + 300$	
9.	$\text{write}(B, b_1)$	

Abhängigkeit von nicht freigegebenen Änderungen

Schritt	T_1	T_2
1.	$\text{read}(A, a_1)$	
2.	$a_1 := a_1 - 300$	
3.	$\text{write}(A, a_1)$	
4.		$\text{read}(A, a_2)$
5.		$a_2 := a_2 * 1.03$
6.		$\text{write}(A, a_2)$
7.	$\text{read}(B, b_1)$	
8.	$\cdots$	
9.	**abort**	

Fehler ... (Forts.)

Phantomproblem

T_1	T_2
	select sum(KontoStand)
	from Konten
insert into Konten	
values $(C, 1000, \ldots)$	
	select sum(KontoStand)
	from Konten

Serialisierbarkeit

- Historie ist „äqivalent“ zu einer seriellen Historie
- dennoch parallele (verzahnte) Ausführung möglich

Serialisierbare Historie von T_1 und T_2

Schritt	T_1	T_2
1.	**BOT**	
2.	read(A)	
3.		**BOT**
4.		read(C)
5.	write(A)	
6.		write(C)
7.	read(B)	
8.	write(B)	
9.	**commit**	
10.		read(A)
11.		write(A)
12.		**commit**

Serielle Ausführung von T_1 vor T_2, also $T_1 \mid T_2$

Schritt	T_1	T_2
1.	**BOT**	
2.	read(A)	
3.	write(A)	
4.	read(B)	
5.	write(B)	
6.	**commit**	
7.		**BOT**
8.		read(C)
9.		write(C)
10.		read(A)
11.		write(A)
12.		**commit**

Nicht serialisierbare Historie

Schritt	T_1	T_3
1.	**BOT**	
2.	read(A)	
3.	write(A)	
4.		**BOT**
5.		read(A)
6.		write(A)
7.		read(B)
8.		write(B)
9.		**commit**
10.	read(B)	
11.	write(B)	
12.	**commit**	

Zwei verzahnte Überweisungs-Transaktionen

Schritt	T_1	T_3
1.	**BOT**	
2.	read(A, a_1)	
3.	$a_1 := a_1 - 50$	
4.	write(A, a_1)	
5.		**BOT**
6.		read(A, a_2)
7.		$a_2 := a_2 - 100$
8.		write(A, a_2)
9.		read(B, b_2)
10.		$b_2 := b_2 + 100$
11.		write(B, b_2)
12.		**commit**
13.	read(B, b_1)	
14.	$b_1 := b_1 + 50$	
15.	write(B, b_1)	
16.	**commit**	

Eine Überweisung (T_1) und eine Zinsgutschrift (T_3)

Schritt	T_1	T_3
1.	**BOT**	
2.	read(A, a_1)	
3.	$a_1 := a_1 - 50$	
4.	write(A, a_1)	
5.		**BOT**
6.		read(A, a_2)
7.		$a_2 := a_2 * 1.03$
8.		write(A, a_2)
9.		read(B, b_2)
10.		$b_2 := b_2 * 1.03$
11.		write(B, b_2)
12.		**commit**
13.	read(B, b_1)	
14.	$b_1 := b_1 + 50$	
15.	write(B, b_1)	
16.	**commit**	

"Formale" Definition einer Transaktion

Operationen einer Transaktion T_i

- $r_i(A)$ zum Lesen des Datenobjekts A,
- $w_i(A)$ zum Schreiben des Datenobjekts A,
- a_i zur Durchführung eines **abort**,
- c_i zur Durchführung des **commit**.

Konsistenzanforderung an die Transaktion

- entweder **abort** oder **commit** – aber nicht beides!
- Falls T_i ein **abort** durchführt, müssen alle anderen Operationen $p_i(A)$ vor a_i ausgeführt werden, also $p_i(A) <_i a_i$.
- Analoges gilt für das **commit**, d.h. $p_i(A) <_i c_i$ falls T_i „**committed**".
- Wenn T_i ein Datum A liest und auch schreibt, muß die Reihenfolge festgelegt werden, also entweder $r_i(A) <_i w_i(A)$ oder $w_i(A) <_i r_i(A)$.

Ordnung der Operationen zweier TAs T_i und T_j

- $r_i(A)$ und $r_j(A)$: In diesem Fall ist die Reihenfolge der Ausführungen irrelavant, da beide TAs in jedem Fall denselben Zustand lesen. Diese beiden Operationen stehen also nicht in Konflikt zueinander, so daß in der Historie ihre Reihenfolge zueinander irrelevant ist

- $r_i(A)$ und $w_j(A)$: Hierbei handelt es sich um einen Konflikt, da T_i entweder den alten oder den neuen Wert von A liest. Es muß also entweder $r_i(A)$ *vor* $w_j(A)$ oder $w_j(A)$ *vor* $r_i(A)$ spezifiziert werden.

- $w_i(A)$ und $r_j(A)$: analog.

- $w_i(A)$ und $w_j(A)$: Auch in diesem Fall ist die Reihenfolge der Ausführung entscheidend für den Zustand der Datenbasis; also handelt es sich um Konfliktoperationen, für die die Reihenfolge festzulegen ist.

Formale Definition einer Historie

- $H = \bigcup_{i=1}^{n} T_i$,

- $<_H$ ist verträglich mit allen $<_i$-Ordnungen, d.h.:

$$<_H \quad \supseteq \quad \bigcup_{i=1}^{n} <_i$$

- für zwei Konfliktoperationen $p, q \in H$ gilt entweder
  - $p <_H q$ oder
  - $q <_H p$.

Historie für drei Transaktionen

Beispiel-Historie für 3 TAs

$$
\begin{array}{cccccc}
 & & r_2(A) \rightarrow & w_2(B) \rightarrow & w_2(C) \rightarrow & c_2 \\
 & & \uparrow & \uparrow & \uparrow & \\
H = & r_3(B) \rightarrow & w_3(A) \rightarrow & w_3(B) \rightarrow & w_3(C) \rightarrow & c_3 \\
 & & \uparrow & & & \\
 & r_1(A) \rightarrow & w_1(A) \rightarrow & c_1 & &
\end{array}
$$

Äquivalenz zweier Historien

- $H \equiv H'$ wenn sie die Konfliktoperationen der nicht abgebrochenen Transaktionen in derselben Reihenfolge ausführen

$$r_1(A) \to r_2(C) \to w_1(A) \to w_2(C) \to r_1(B) \to w_1(B) \to c_1 \to r_2(A) \to w_2(A) \to c_2$$

$$r_1(A) \to w_1(A) \to r_2(C) \to w_2(C) \to r_1(B) \to w_1(B) \to c_1 \to r_2(A) \to w_2(A) \to c_2$$

$$r_1(A) \to w_1(A) \to r_1(B) \to r_2(C) \to w_2(C) \to w_1(B) \to c_1 \to r_2(A) \to w_2(A) \to c_2$$

$$r_1(A) \to w_1(A) \to r_1(B) \to w_1(B) \to c_1 \to r_2(C) \to w_2(C) \to r_2(A) \to w_2(A) \to c_2$$

Serialisierbare Historie

Eine Historie ist *serialisierbar* wenn sie äquivalent zu einer seriellen Historie Hs ist.

Historie und zugehöriger Serialisierbarkeitsgraph

$$
H = \begin{array}{ccccccc}
r_1(A) & \rightarrow & w_1(A) & \rightarrow & w_1(B) & \rightarrow & c_1 \\
 & & \uparrow & & \uparrow & & \\
 & \swarrow & r_2(A) & \rightarrow & w_2(B) & \rightarrow & c_2 \\
 & & \downarrow & & & & \\
r_3(A) & \rightarrow & w_3(A) & \rightarrow & c_3 & &
\end{array}
$$

$$
SG(H) = \begin{array}{ccc}
 & & T_3 \\
 & \nearrow & \\
T_2 & & \uparrow \\
 & \searrow & \\
 & & T_1
\end{array}
$$

- $w_1(A) \rightarrow r_3(A)$ der Historie H führt zur Kante $T_1 \rightarrow T_3$ des SG
- weitere Kanten analog

Serialisierbarkeitstheorem

Eine Historie H ist genau dann *serialisierbar*, wenn der zugehörige Serialisierbarkeitsgraph $SG(H)$ azyklisch ist.

Historie

$$H =$$

$$w_1(A) \to w_1(B) \to c_1 \to r_2(A) \to r_3(B) \to w_2(A) \to c_2 \to w_3(B) \to c_3$$

Serialisierbarkeitsgraph

$$SG(H) = \begin{tikzcd} & T_2 \\ T_1 \arrow[ur] \arrow[dr] & \\ & T_3 \end{tikzcd}$$

Topologische Ordnung(en)

$$\begin{aligned} H_s^1 &= T_1 \mid T_2 \mid T_3 \\ H_s^2 &= T_1 \mid T_3 \mid T_2 \\ H &\equiv H_s^1 \equiv H_s^2 \end{aligned}$$

Eigenschaften von Historien bezüglich der Recovery

Terminologie

Wir sagen, daß in der Historie H T_i von T_j liest, wenn folgendes gilt:

1. T_j schreibt mindestens ein Datum A, das T_i nachfolgend liest, also:

$$w_j(A) <_H r_i(A)$$

2. T_j wird (zumindest) nicht vor dem Lesevorgang von T_i zurückgesetzt, also:

$$a_j \not<_H r_i(A)$$

3. Alle anderen zwischenzeitlichen Schreibvorgänge auf A durch andere Transaktionen T_k werden vor dem Lesen durch T_i zurückgesetzt. Falls also ein $w_k(A)$ mit $w_j(A) < w_k(A) < r_i(A)$ existiert, so muß es auch ein $a_k < r_i(A)$ geben.

Rücksetzbare Historien

Eine Historie heißt rücksetzbar, falls immer die schreibende Transaktion (in unserer Notation T_j) vor der lesenden Transaktion (T_i genannt) ihr **commit** durchführt, also: $c_j <_H c_i$. Anders ausgedrückt: Eine Transaktion darf erst dann ihr **commit** durchführen, wenn alle Transaktionen, von denen sie gelesen hat, beendet sind.

Beispiel-Historie mit kaskadierendem Rücksetzen

Schritt	T_1	T_2	T_3	T_4	T_5
0.	$\cdots$				
1.	$w_1(A)$				
2.		$r_2(A)$			
3.		$w_2(B)$			
4.			$r_3(B)$		
5.			$w_3(C)$		
6.				$r_4(C)$	
7.				$w_4(D)$	
8.					$r_5(D)$
9.	a_1 (**abort**)				

Historien ohne kaskadierendes Rücksetzen

Eine Historie vermeidet kaskadierendes Rücksetzten, wenn für je zwei TAs T_i und T_j gilt:

- $c_j <_H r_i(A)$ gilt, wann immer T_i ein Datum A von T_j liest.

Strikte Historien

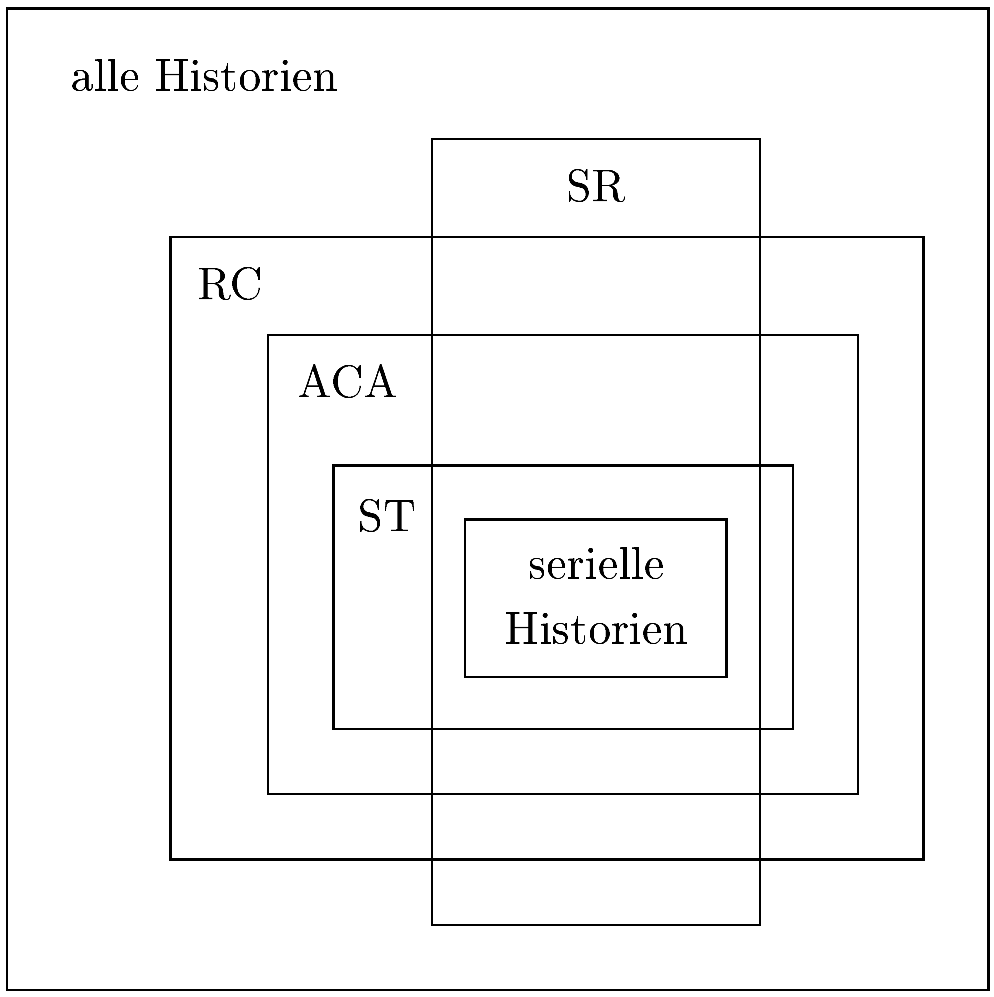
Eine Historie ist strikt wenn für je zwei TAs T_i und T_j gilt: Wenn

$$w_j(A) <_H o_i(A)$$

Dann muß gelten:

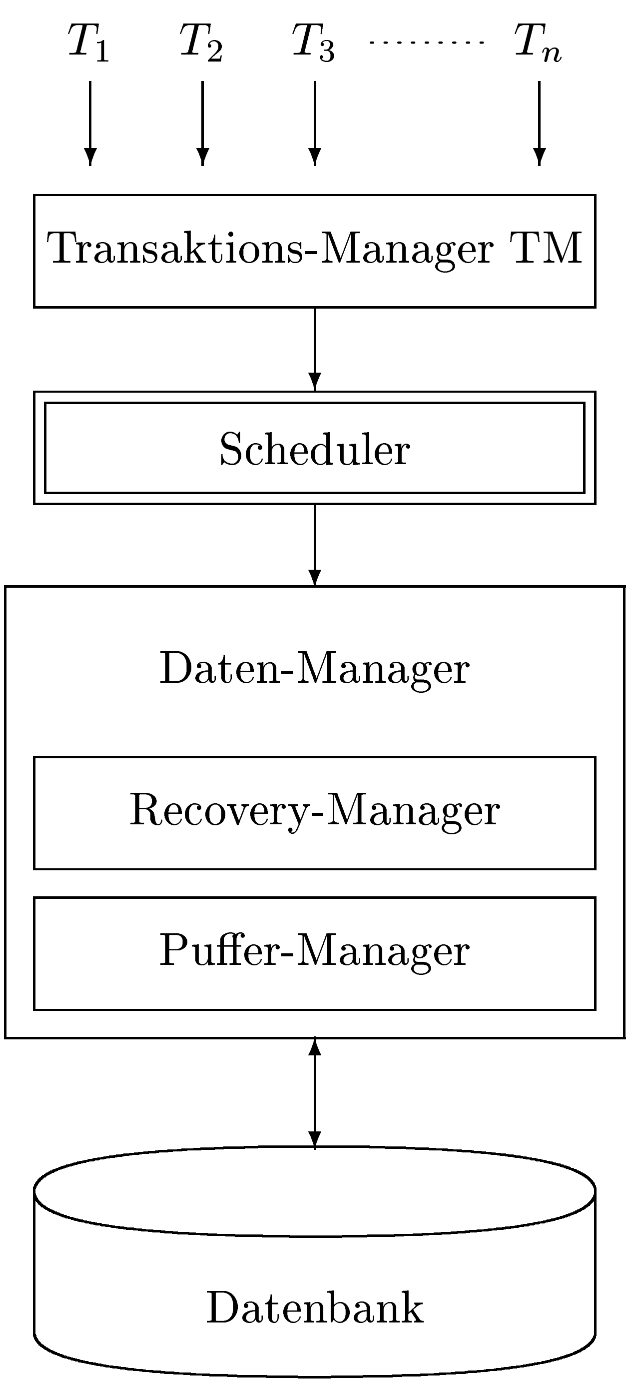
- $c_j <_H o_i(A)$ oder
- $a_j <_H o_i(A)$

Beziehungen zwischen den Klassen von Historien



- SR: serialisierbare Historien
- RC: rücksetzbare Historien
- ACA: Historien ohne kaskadierendes Rücksetzen
- ST: strikte Historien

Der Datenbank-Scheduler



Zwei Sperrmodi

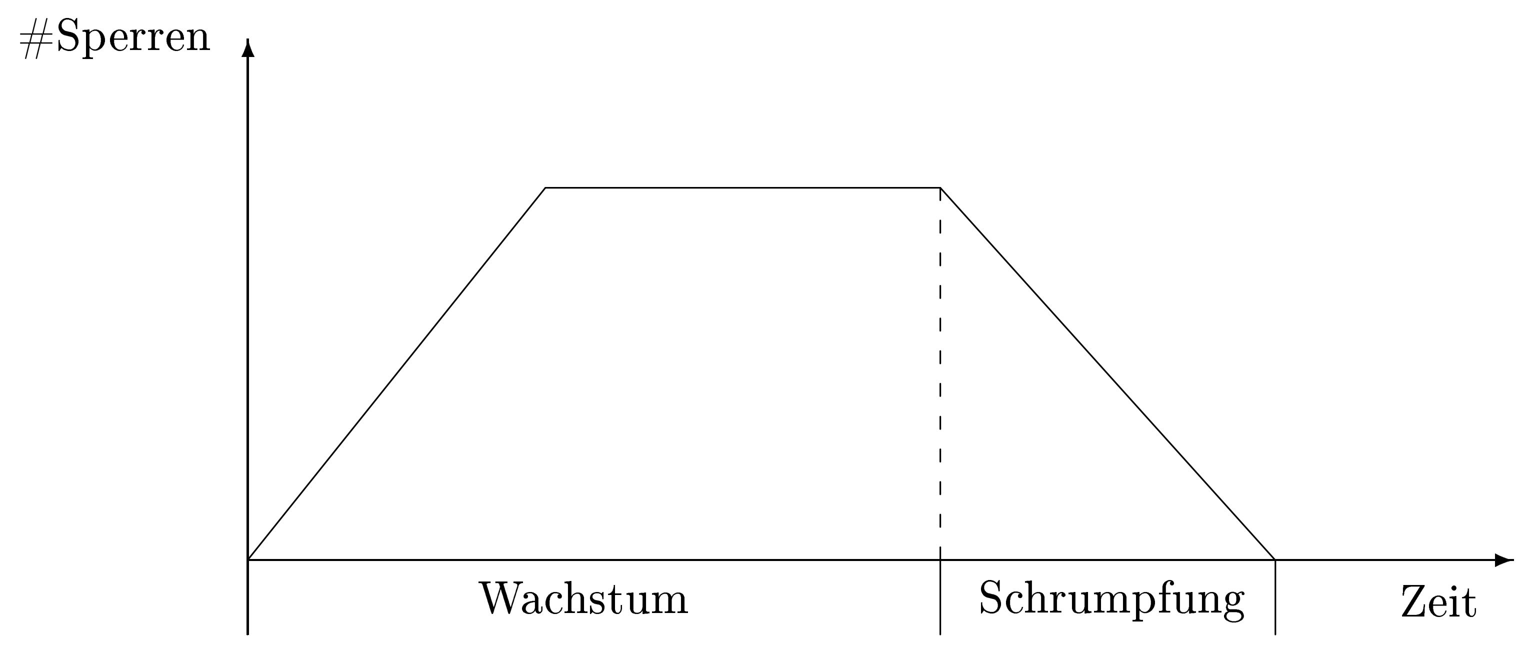
- S (shared, read lock, Lesesperre):
- X (exclusive, write lock, Schreibsperre):
- *Verträglichkeitsmatrix* (auch *Kompatibilitätsmatrix* genannt)

	NL	S	X
S	$\surd$	$\surd$	$-$
X	$\surd$	$-$	$-$

Zwei-Phasen-Sperrprotokoll: Definition

1. Jedes Objekt, das von einer Transaktion benutzt werden soll, muß vorher entsprechend gesperrt werden.
2. Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.
3. eine Transaktion muß die Sperren anderer Transaktionen auf dem von ihr benötigten Objekt gemäß der Verträglichkeitstabelle beachten. Wenn die Sperre nicht gewährt werden kann, wird die Transaktion in eine entsprechende Warteschlange eingereiht – bis die Sperre gewährt werden kann.
4. Jede Transaktion durchläuft zwei Phasen:
   - Eine *Wachstumsphase*, in der sie Sperren anfordern, aber keine freigeben darf und
   - Eine *Schrumpfungsphase*, in der sie ihre bisher erworbenen Sperren freigibt, aber keine weiteren anfordern darf.
5. Bei EOT (Transaktionende) muß eine Transaktion alle ihre Sperren zurückgeben.

Zwei-Phasen Sperrprotokoll: Graphik



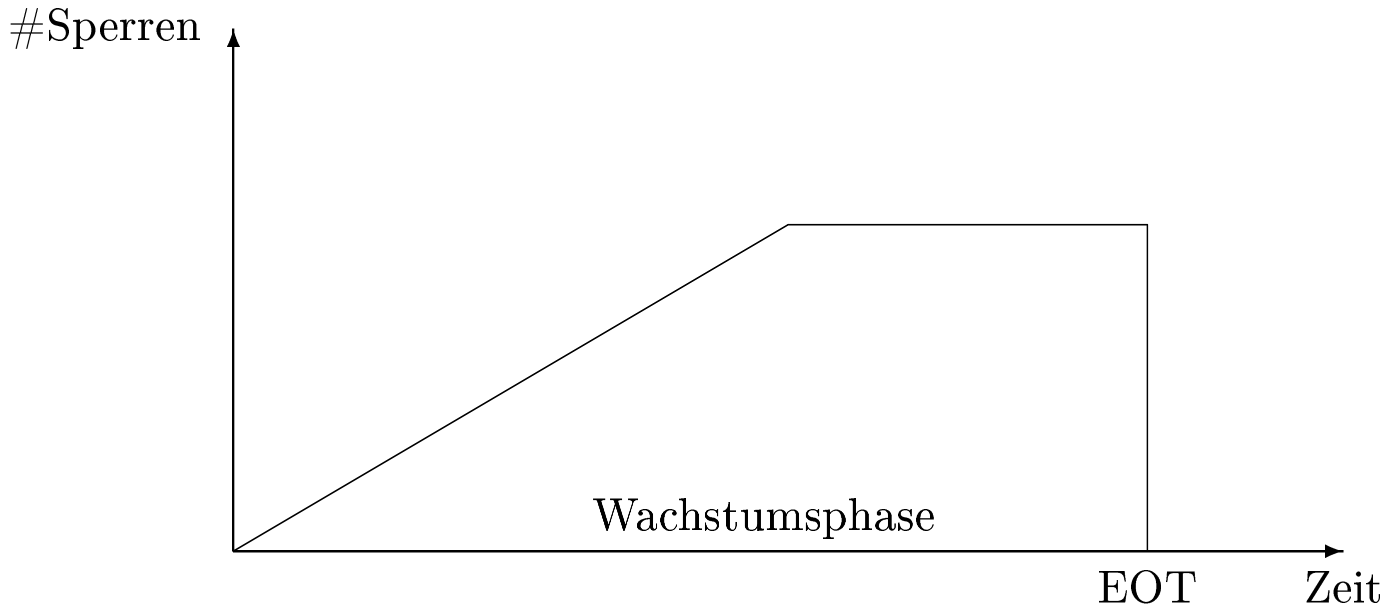
Verzahnung zweier TAs gemäß 2PL

- T_1 modifiziert nacheinander die Datenobjekte A und B (z.B. eine Überweisung)
- T_2 liest nacheinander dieselben Datenobjekte A und B (z.B. zur Aufsummierung der beiden Kontostände).

Schritt	T_1	T_2	Bemerkung
1.	**BOT**		
2.	**lockX**(A)		
3.	read(A)		
4.	write(A)		
5.		**BOT**	
6.		**lockS**(A)	T_2 muß warten
7.	**lockX**(B)		
8.	read(B)		
9.	**unlockX**(A)		T_2 wecken
10.		read(A)	
11.		**lockS**(B)	T_2 muß warten
12.	write(B)		
13.	**unlockX**(B)		T_2 wecken
14.		read(B)	
15.	**commit**		
16.		**unlockS**(A)	
17.		**unlockS**(B)	
18.		**commit**	

Strenges Zwei-Phasen Sperrprotokoll

- 2PL schließt kaskadierendes Rücksetzen nicht aus
- Erweiterung zum *strengen* 2PL:
  - alle Sperren werden bis EOT gehalten
  - damit ist kaskadierendes Rücksetzen ausgeschlossen



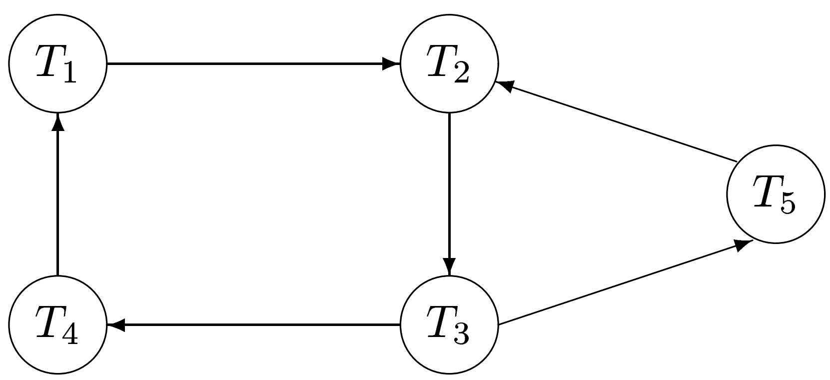
Ein verklemmter Schedule

Schritt	T_1	T_2	Bemerkung
1.	**BOT**		
2.	**lockX**(A)		
3.		**BOT**	
4.		**lockS**(B)	
5.		read(B)	
6.	read(A)		
7.	write(A)		
8.	**lockX**(B)		T_1 muß warten auf T_2
9.		**lockS**(A)	T_2 muß warten auf T_1
10.	...	...	$\Rightarrow$ *Deadlock*

Erkennung von Verklemmungen

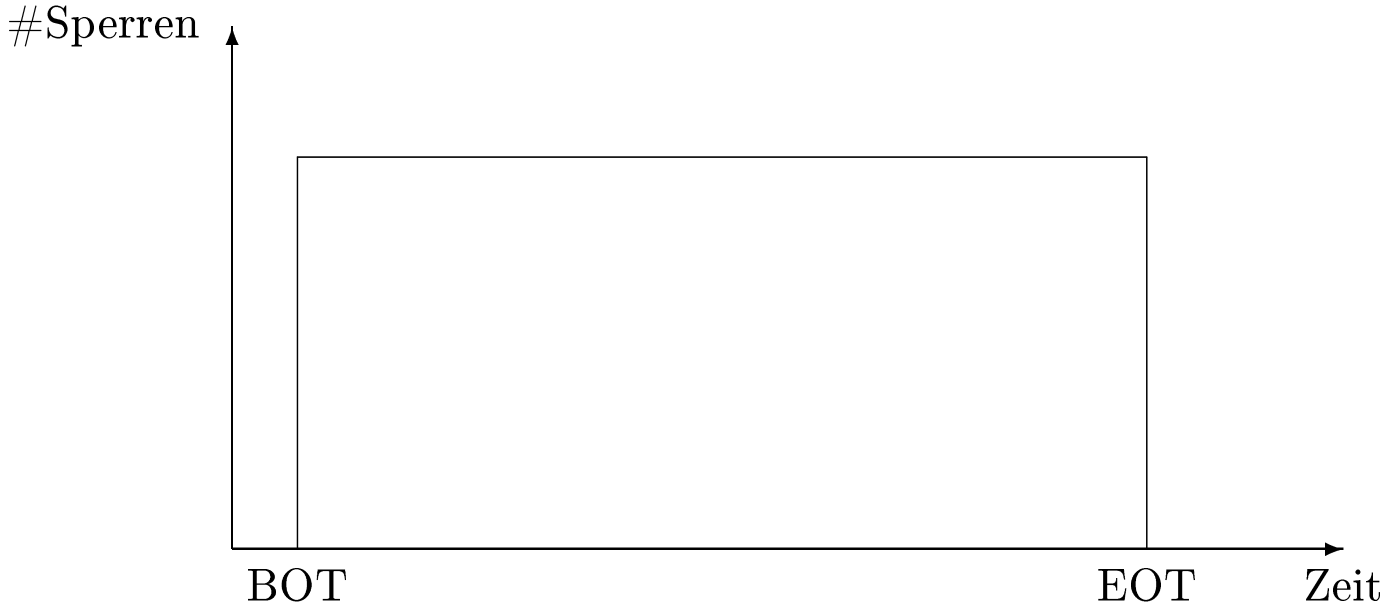
Wartegraph mit zwei Zyklen:

- $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$
- $T_2 \rightarrow T_3 \rightarrow T_5 \rightarrow T_2$



- beide Zyklen können durch Rücksetzen von T_3 „gelöst“ werden
- Zyklenerkennung durch Tiefensuche im Wartegraphen

Preclaiming in Verbindung mit dem strengen 2 PL-Protokoll



Verklemmungsvermeidung durch Zeitstempel

- Jeder Transaktion wird ein eindeutiger Zeitstempel (TS) zugeordnet
- ältere TAs haben einen kleineren Zeitstempel als jüngere TAs
- TAs dürfen nicht mehr „bedingungslos“ auf eine Sperre warten

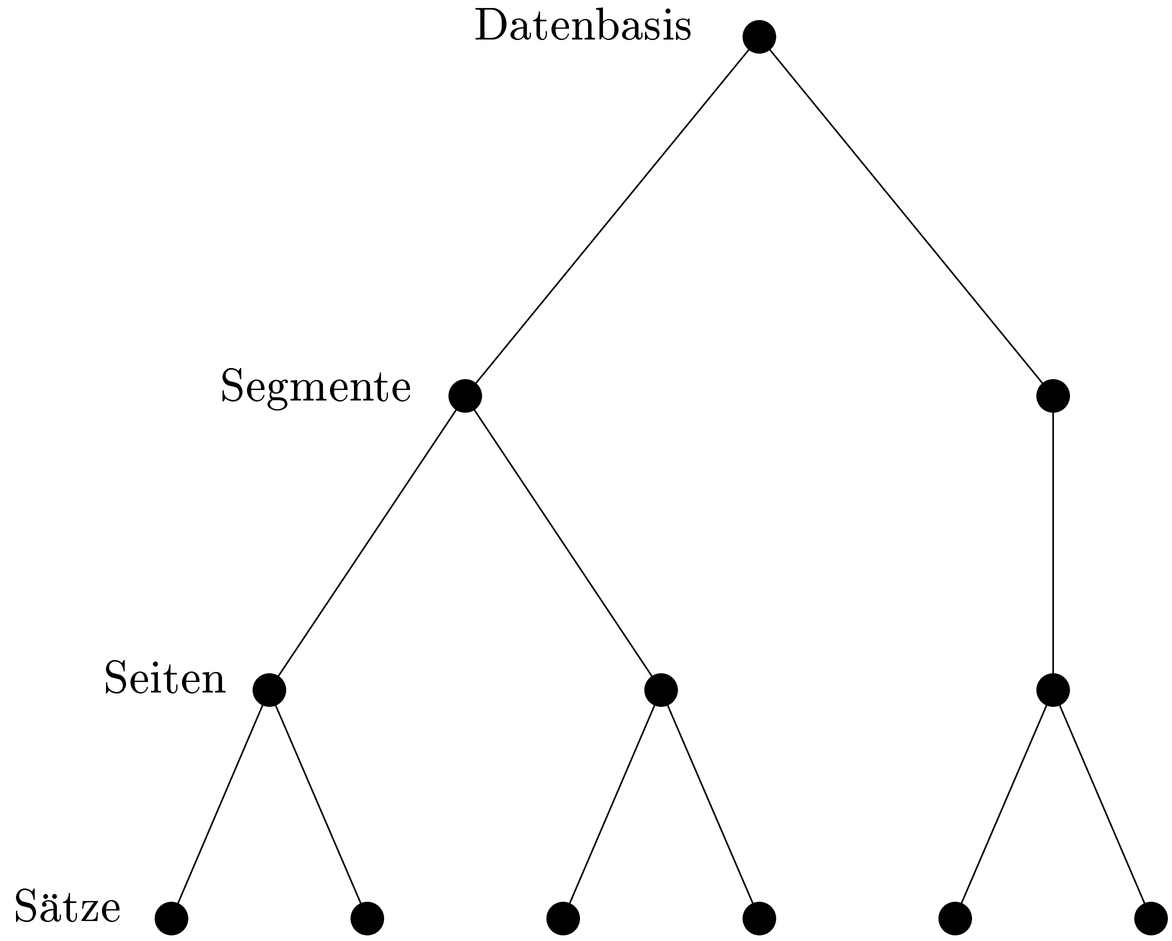
wound-wait Strategie

- T_1 will Sperre erwerben, die von T_2 gehalten wird
- Wenn T_1 älter als T_2 ist, wird T_2 abgebrochen und zurückgesetzt, so daß T_1 weiterlaufen kann.
- Sonst wartet T_1 auf die Freigabe der Sperre durch T_2.

wait-die Strategie

- T_1 will Sperre erwerben, die von T_2 gehalten wird
- Wenn T_1 älter als T_2 ist, wartet T_1 auf die Freigabe der Sperre.
- Sonst wird T_1 abgebrochen und zurückgesetzt.

Hierarchische Anordnung möglicher Sperrgranulate



Erweiterte Sperrmodi

- *NL*: keine Sperrung (no lock),
- S: Sperrung durch Leser,
- X: Sperrung durch Schreiber,
- *IS* (intention share): Weiter unten in der Hierarchie ist eine Lesesperre (S) beabsichtigt,
- *IX* (intention exclusive): Weiter unten in der Hierarchie ist eine Schreibsperre (X) beabsichtigt.

Multi-Granularity Locking (MGL)

Kompatibilitätsmatrix

	NL	S	X	IS	IX
S	$\surd$	$\surd$	–	$\surd$	–
X	$\surd$	–	–	–	–
IS	$\surd$	$\surd$	–	$\surd$	$\surd$
IX	$\surd$	–	–	$\surd$	$\surd$

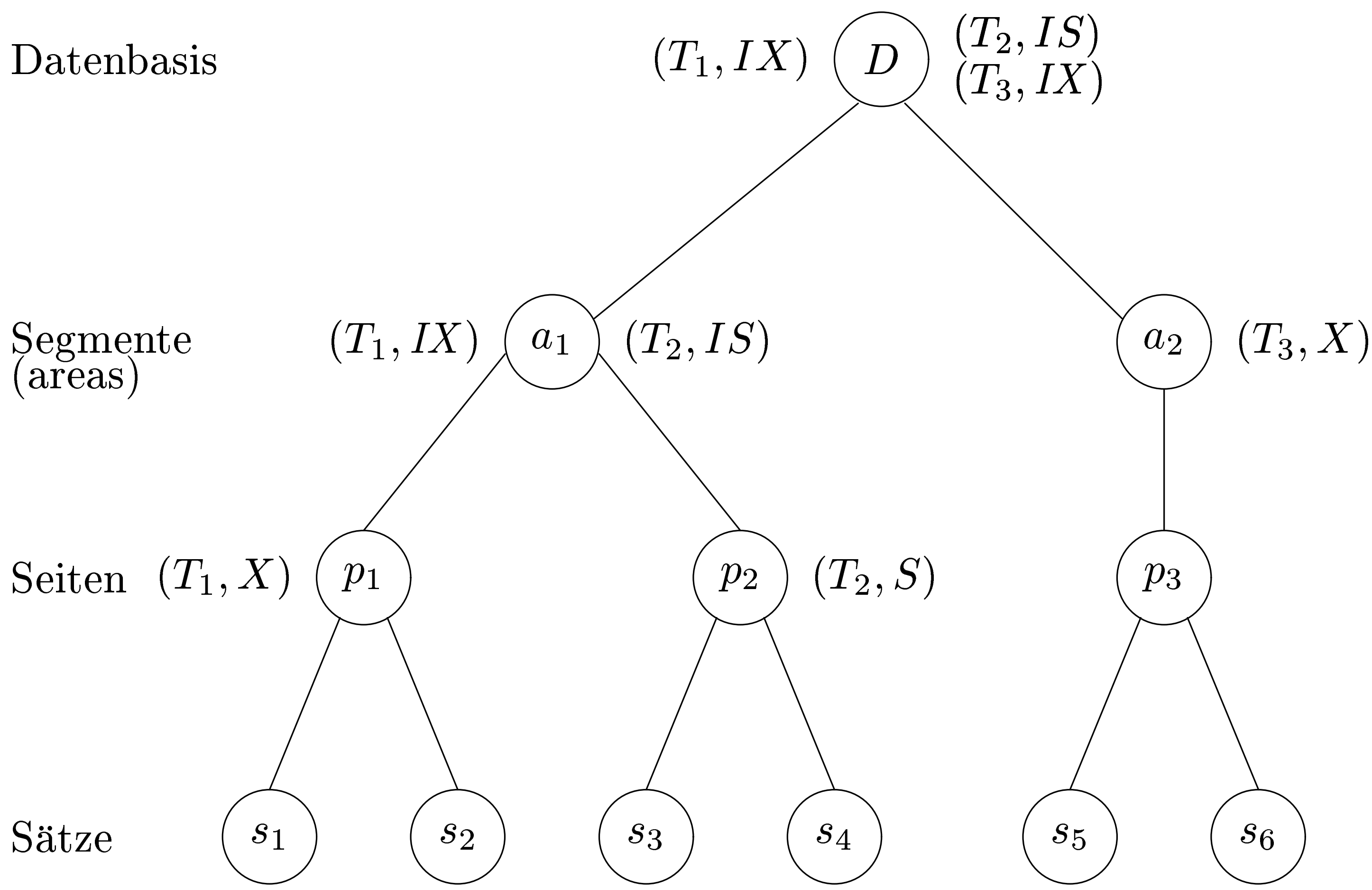
Sperrprotokoll des MGL

1. Bevor ein Knoten mit S oder IS gesperrt wird, müssen alle Vorgänger in der Hierarchie vom Sperrer (also der Transaktion, die die Sperre anfordert) im IX- oder IS- Modus gehalten werden.

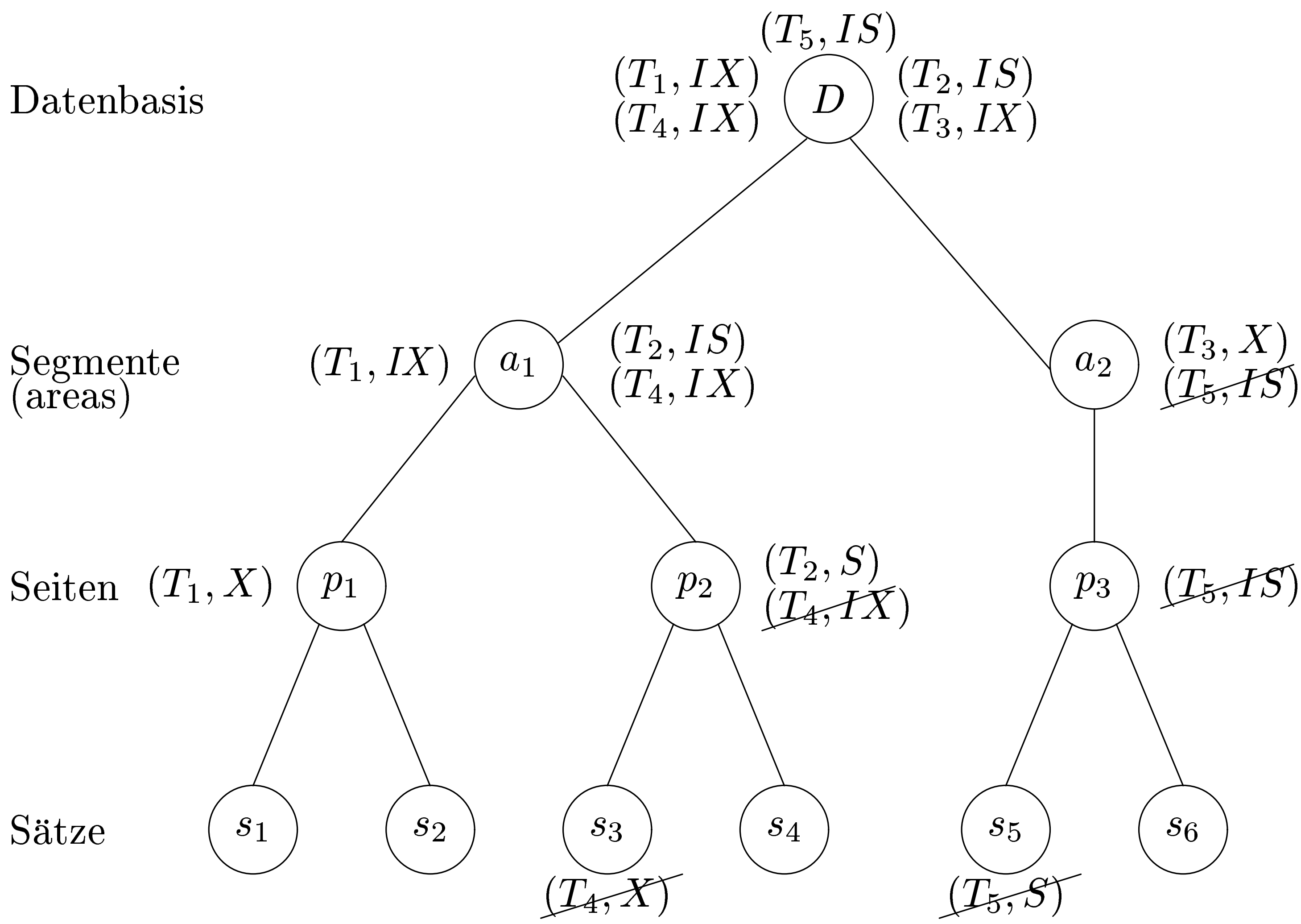
2. Bevor ein Knoten mit X oder IX gesperrt wird, müssen alle Vorgänger vom Sperrer im IX-Modus gehalten werden.

3. Die Sperren werden von unten nach oben (bottom up) freigegeben, so daß bei keinem Knoten die Sperre freigegeben wird, wenn die betreffende Transaktion noch Nachfolger dieses Knotens gesperrt hat.

Datenbasis-Hierarchie mit Sperren



Datenbasis-Hierarchie mit blockierten Transaktionen



- die TAs T_4 und T_5 sind blockiert (warten auf Freigabe von Sperren)
- es gibt aber in diesem Beispiel (noch) keine Verklemmung
- Verklemmungen sind aber auch bei MGL möglich

Einfüge- und Löschoperationen, Phantome

- Vor dem Löschen eines Objekts muß die Transaktion eine X-Sperre für dieses Objekt erwerben. Man beachte aber, daß eine andere TA, die für dieses Objekt ebenfalls eine Sperre erwerben will, diese nicht mehr erhalten kann, falls die Löschtransaktion erfolgreich (mit **commit**) abschließt.

- Beim Einfügen eines neuen Objekts erwirbt die einfügende Transaktion eine X-Sperre.

Phantomprobleme

T_1	T_2
select count(∗) **from** prüfen **where** Note **between** 1 **and** 2;	
	insert into prüfen **values**(29555, 5001, 2137, 1);
select count(∗) **from** prüfen **where** Note **between** 1 **and** 2;	

- Das Problem läßt sich dadurch lösen, daß man zusätzlich zu den Tupeln auch den Zugriffsweg, auf dem man zu den Objekten gelangt ist, sperrt
- Wenn also ein Index für das Attribut *Note* existiert, würde der Indexbereich $[1, 2]$ für T_1 mit einer S-Sperre belegt
- Wenn jetzt also Transaktion T_2 versucht, das Tupel [29555, 5001, 2137, 1] in *prüfen* einzufügen, wird die TA blockiert

Zeitstempel-basierende Synchronisation

Jedem Datum A in der Datenbasis werden bei diesem Synchronisationsverfahren zwei Marken zugeordnet:

1. $readTS(A)$:
2. $writeTS(A)$:

Synchronisationsverfahren

- T_i will A lesen, also $r_i(A)$
  - Falls $TS(T_i) < writeTS(A)$ gilt, haben wir ein Problem:
    - ∗ Die Transaktion T_i ist älter als eine andere Transaktion, die A schon geschrieben hat.
    - ∗ Also muß T_i zurückgesetzt werden.
  - Anderenfalls, wenn also $TS(T_i) \geq writeTS(A)$ gilt, kann T_i ihre Leseoperation durchführen und die Marke $readTS(A)$ wird auf $max(TS(T_i), readTS(A))$ gesetzt.

- T_i will A schreiben, also $w_i(A)$
  - Falls $TS(T_i) < readTS(A)$ gilt, gab es eine jüngere Lesetransaktion, die den neuen Wert von A, den T_i gerade beabsichtigt zu schreiben, hätte lesen müssen. Also muß T_i zurückgesetzt werden.
  - Falls $TS(T_i) < writeTS(A)$ gilt, gab es eine jüngere Schreibtransaktion. D.h. T_i beabsichtigt einen Wert einer jüngeren Transaktion zu überschreiben. Das muß natürlich verhindert werden, so daß T_i auch in diesem Fall zurückgesetzt werden muß.
  - Anderenfalls darf T_i das Datum A schreiben und die Marke $writeTS(A)$ wird auf $TS(T_i)$ gesetzt.

Optimistische Synchronisation

1. *Lesephase*:
   - In dieser Phase werden alle Operationen der Transaktion ausgeführt – also auch die Änderungsoperationen.
   - Gegenüber der Datenbasis tritt die Transaktion in dieser Phase aber nur als Leser in Erscheinung, da alle gelesenen Daten in lokalen Variablen der Transaktion gespeichert werden.
   - alle Schreiboperationen werden (zunächst) auf diesen lokalen Variablen ausgeführt.
2. *Validierungsphase*:
   - In dieser Phase wird entschieden, ob die Transaktion möglicherweise in Konflikt mit anderen Transaktionen geraten ist.
   - Dies wird anhand von Zeitstempeln entschieden, die den Transaktionen in der Reihenfolge zugewiesen werden, in der sie in die Validierungsphase eintreten.
3. *Schreibphase*:
   - Die Änderungen der Transaktionen, bei denen die Validierung positiv verlaufen ist, werden in dieser Phase in die Datenbank eingebracht.

Validierung bei der optimistischen Synchronisation

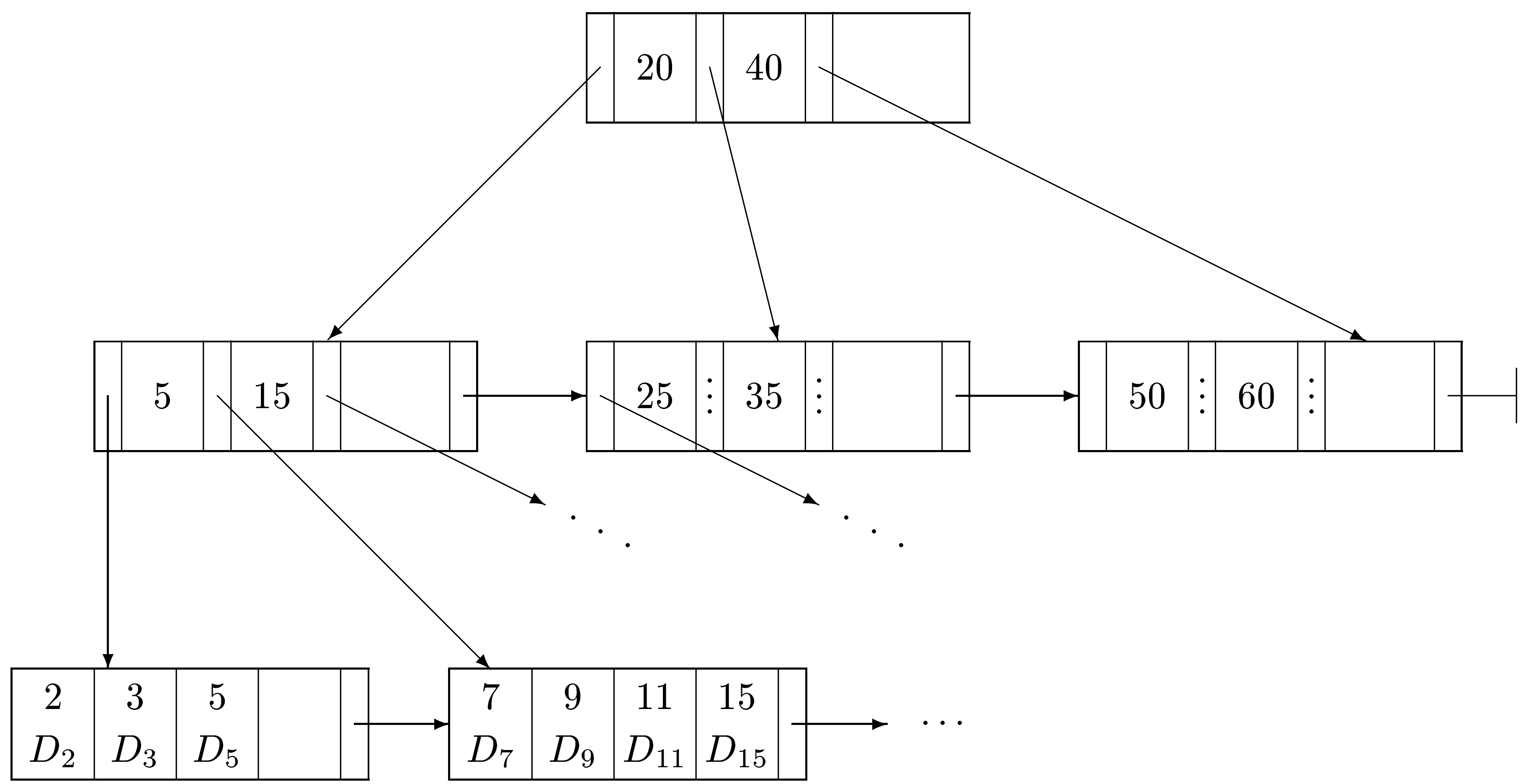
Vereinfachende Annahme: Es ist immer nur eine TA in der Validierungsphase!

Wir wollen eine Transaktion T_j validieren. Die Validierung ist erfolgreich falls für **alle** älteren Transaktionen T_a – also solche die früher ihre Validierung abgeschlossen haben – eine der beiden folgenden Bedingungen gelten:

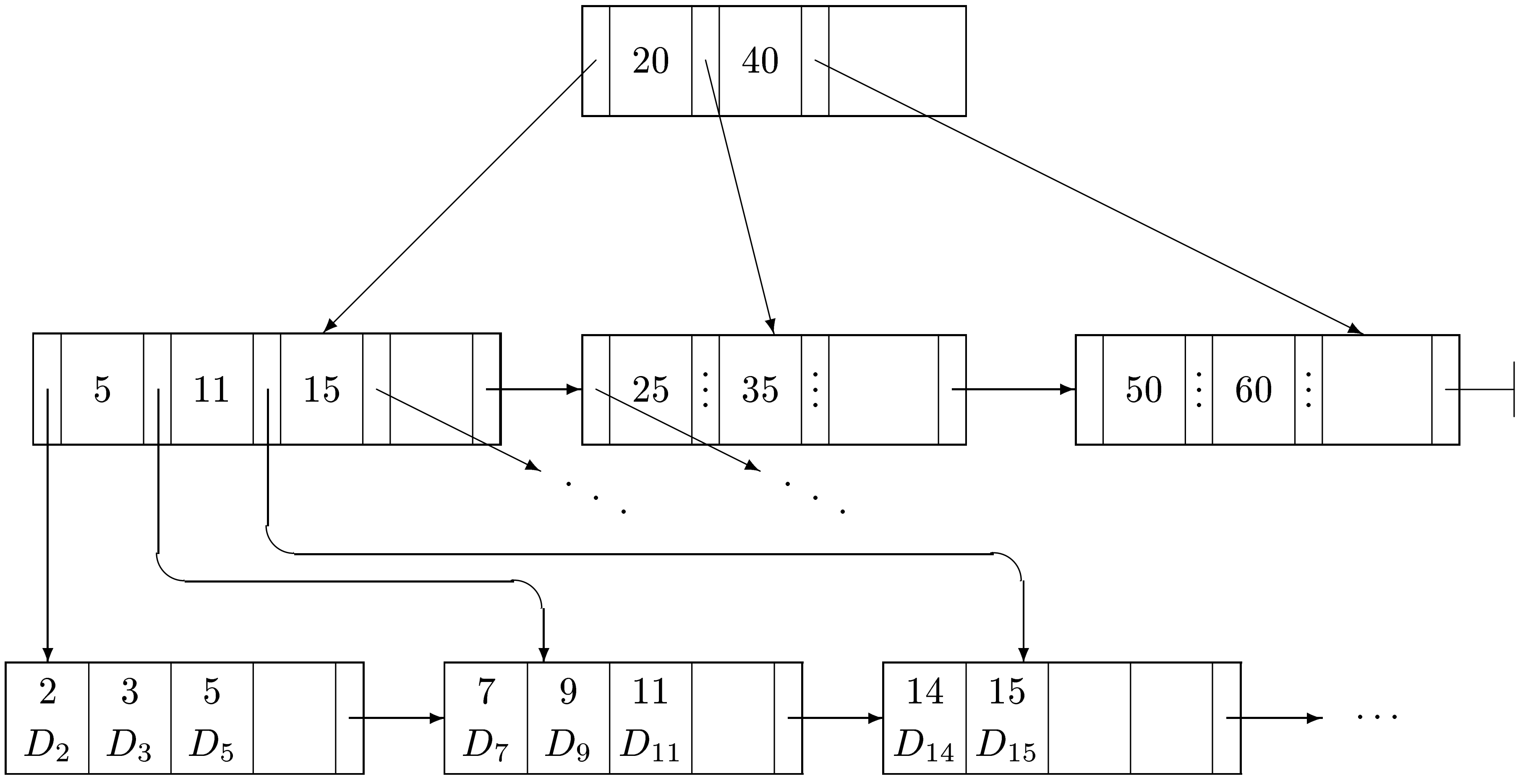
1. T_a war zum Beginn der Transaktion T_j schon abgeschlossen – einschließlich der Schreibphase.
2. Die Menge der von T_a geschriebenen Datenelemente, genannt $WriteSet(T_a)$, enthält keine Elemente der Menge der gelesenen Datenelemente von T_j, genannt $ReadSet(T_j)$. Es muß also gelten:

$$WriteSet(T_a) \cap ReadSet(T_j) = \emptyset$$

B^+-Baum mit *rechts*-Verweisen zur Synchronisation



B$^+$-Baum mit *rechts*-Verweisen nach Einfügen von 14



Transaktionsverwaltung in SQL92

set transaction
    [read only, | <u>read write</u>,]
    [**isolation level**
        read uncommitted, |
        read committed, |
        repeatable read, |
        <u>serializable</u>,]
    [**diagnostics size** . . . ,]

- **read uncommitted**: Dies ist die schwächste Konsistenzstufe. Sie darf auch nur für **read only**-Transaktionen spezifiziert werden. Eine derartige Transaktion hat Zugriff auf noch nicht festgeschriebene Daten. Zum Beispiel ist folgender Schedule möglich:

T_1	T_2
	read(A)
	. . .
	write(A)
read(A)	
. . .	
	rollback

- **read committed**: Diese Transaktionen lesen nur festgeschriebene Werte. Allerdings können sie unterschiedliche Zustände der Datenbasis-Objekte zu sehen bekommen:

T_1	T_2
read(A)	
	write(A)
	write(B)
	commit
read(B)	
read(A)	
...	

- **repeatable read**: Das oben aufgeführte Problem des *non repeatable read* wird durch diese Konsistenzstufe ausgeschlossen. Allerdings kann es hierbei noch zum Phantomproblem kommen. Dies kann z.B. dann passieren, wenn eine parallele Änderungstransaktion dazu führt, daß Tupel ein Selektionsprädikat erfüllen, das sie zuvor nicht erfüllten.

- **serializable**: Diese Konsistenzstufe fordert die Serialisierbarkeit. Dies ist der Default.

Sicherheit in DBMS

- Identifikation und Authentisierung
- Autorisierung und Zugriffskontrolle
- Auditing

Angriffsarten

- Mißbrauch von Autorität
- Inferenz und Aggregation
- Maskierung
- Umgehung der Zugriffskontrolle
- Browsing
- Trojanische Pferde
- Versteckte Kanäle

Discretionary Access Control

Zugriffsregeln (o, s, t, p, f) mit

- $o \in O$, der Menge der Objekte (z.B. Relationen, Tupel, Attribute),
- $s \in S$, der Menge der Subjekte (z.B. Benutzer, Prozesse),
- $t \in T$, der Menge der Zugriffsrechte (z.B. $T = \{\text{lesen}, \text{schreiben}, \text{löschen}\}$),
- p ein Prädikat (z.B. $\mathit{Rang} =$ 'C4' für die Relation *Professoren*), und
- f ein Boolescher Wert, der angibt, ob s das Recht (o, t, p) an ein anderes Subjekt s' weitergeben darf.

Discretionary Access Control

Realisierung:

- Zugriffsmatrix
- Sichten
- „Query Modification“

Nachteile:

- Erzeuger der Daten= Verantwortlicher für deren Sicherheit

Zugriffskontrolle in SQL

Beispiel:

grant select
  on Professoren
  to eickler;

grant update (MatrNr, VorlNr, PersNr)
  on prüfen
  to eickler;

Zugriffskontrolle in SQL

Weitere Rechte:

- delete
- insert
- references

Weitergabe von Rechten:

- with grant option

Entzug von Rechten:

revoke update (MatrNr, VorlNr, PersNr)
    on prüfen
    from eickler **cascade**;

Realisierung des Zugriffsprädikats:

```
create view ErstSemestler as
    select  *
    from    Studenten
    where   Semester = 1;
grant select
    on      ErstSemestler
    to      tutor;
```

Schutz von Individualdaten durch Aggregation:

```
create view VorlesungsHärte (VorlNr, Härte) as
    select VorlNr, avg(Note)
    from prüfen
    group by VorlNr;
```

Beispiele:

audit session by system
    whenever not successful;

audit insert, delete, update on Professoren;

Verfeinerung des Autorisierungsmodells

- explizite/implizite Autorisierung
- positive/negative Autorisierung
- starke/schwache Autorisierung

Autorisierungsalgorithmus:

wenn es eine explizite oder implizite starke Autorisierung (o, s, t) gibt,
  dann erlaube die Operation
wenn es eine explizite oder implizite starke negative Autorisierung $(o, s, \neg t)$ gibt,
  dann verbiete die Operation
ansonsten
  wenn es eine explizite oder implizite schwache Autorisierung $[o, s, t]$ gibt,
    dann erlaube die Operation
  wenn es eine explizite oder implizite schwache Autorisierung $[o, s, \neg t]$ gibt,
    dann verbiete die Operation

Implizite Autorisierung von Subjekten

$$
\begin{array}{ccc}
 & \text{Rektor/in} & \\
\swarrow & & \searrow \\
\text{Dekane} & & \\
\downarrow & & \text{Referatsleiter} \\
\text{Professoren} & & \downarrow \\
\downarrow & & \text{Verwaltungs-} \\
\text{wissenschaftliche} & & \text{angestellte} \\
\text{Angestellte} & & \\
\searrow & & \swarrow \\
 & \text{Angestellte} &
\end{array}
$$

- explizite positive Autorisierung

  ⇒ implizite positive Autorisierung auf allen *höheren* Stufen

- explizite negative Autorisierung

  ⇒ implizite negative Autorisierung auf allen *niedrigeren* Stufen

Implizite Autorisierung von Operationen

$$\begin{array}{c}\text{schreiben}\\ \downarrow\\ \text{lesen}\end{array}$$

- explizite positive Autorisierung

  $\Rightarrow$ implizite positive Autorisierung auf allen *niedrigeren* Stufen

- explizite negative Autorisierung

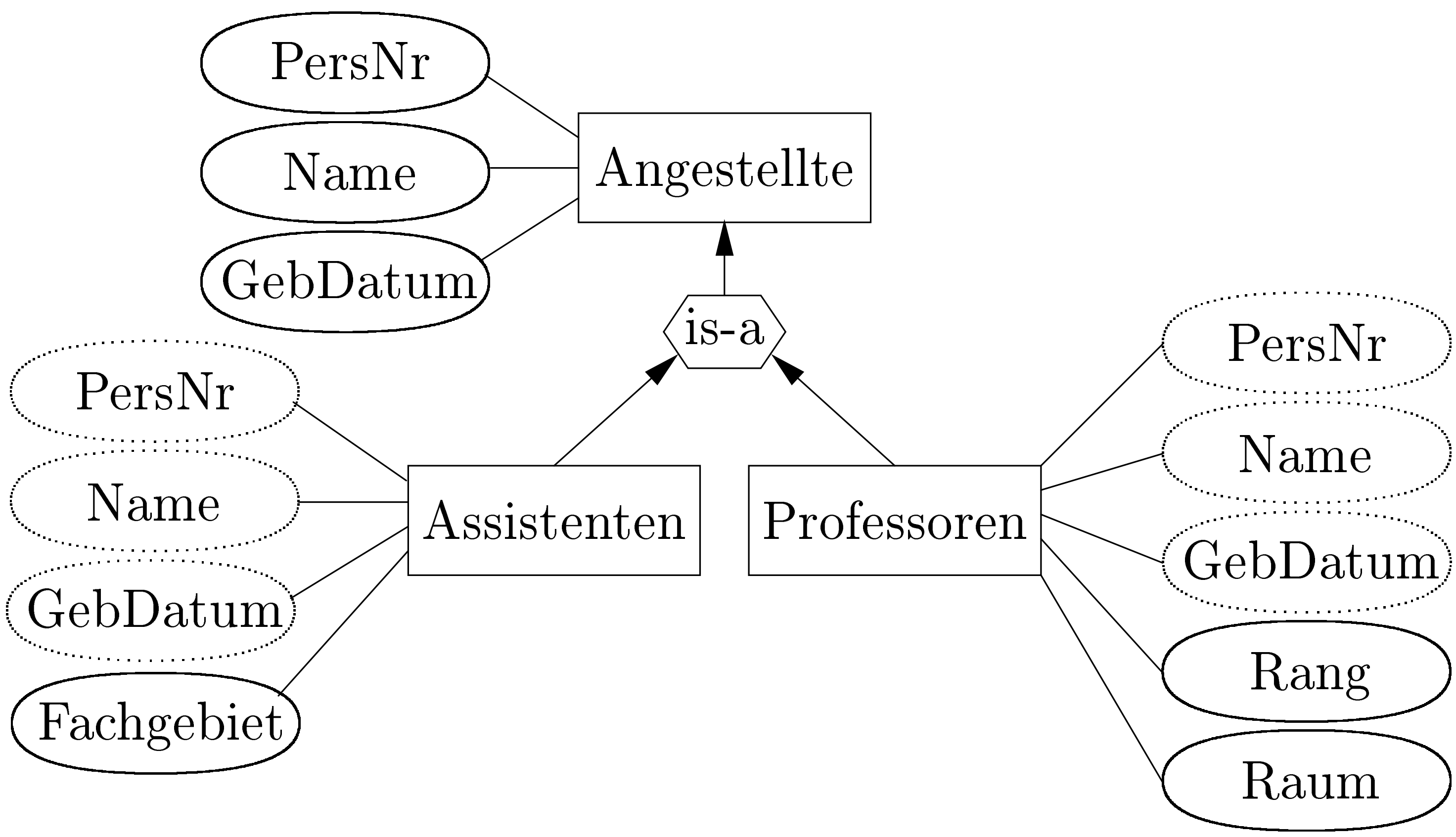
  $\Rightarrow$ implizite negative Autorisierung auf allen *höheren* Stufen

Implizite Autorisierung von Objekten

$$\begin{array}{c} \text{Datenbank} \\ \downarrow \\ \text{Schema} \\ \downarrow \\ \text{Relation} \\ \downarrow \\ \text{Tupel} \\ \downarrow \\ \text{Attribut} \end{array}$$

- Implikationen abhängig von Operation

Implizite Autorisierung entlang einer Typhierarchie



Benutzergruppen:

- Verwaltungsangestellte dürfen die Namen aller Angestellten lesen
- wissenschaftliche Angestellte dürfen Namen und Rang aller Professoren lesen

Anfragen:

- lese die Namen aller Angestellten
- lese Namen und Rang aller Professoren

Implizite Autorisierung entlang einer Typhierarchie

Regeln:

- Benutzer mit einem Zugriffsrecht auf einen Objekttypen haben auf die geerbten Attribute in den Untertypen ein gleichartiges Zugriffsrecht.
- Ein Zugriffsrecht auf einen Objekttyp impliziert auch ein Zugriffsrecht auf alle von Obertypen geerbten Attribute in diesem Typ.
- Ein Attribut, das in einem Untertyp definiert wurde, ist nicht von einem Obertyp aus erreichbar.

Mandatory Access Control

- hierarchische Klassifikation von Vertrauenswürdigkeit und Sensitivität
- $clear(s)$, mit s Subjekt (clearance)
- $class(o)$, mit o Objekt (classification)
- Ein Subjekt s darf ein Objekt o nur lesen, wenn das Objekt eine geringere Sicherheitseinstufung besitzt ($class(o) \leq clear(s)$).
- Ein Objekt o muß mit mindestens der Einstufung des Subjektes s geschrieben werden ($clear(s) \leq class(o)$).

- Benutzer soll sich der Existenz unzugänglicher Daten nicht bewußt sein

Beispiel (TC= Klassifizierung des gesamten Tupels):

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	meucheln	sg
sg	008	sg	Mata, Harry	sg	spitzeln	sg

Sichtweise eines „geheim“ eingestuften Benutzers:

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	–	g

Probleme:

- „geheimer“ Benutzer fügt Tupel mit Schlüssel „008“ ein
- „geheimer“ Benutzer modifiziert Spezialität von „007“

Multilevel-Relationen

Multilevel-Relation R mit Schema

$$\mathcal{R} = \{A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC\}$$

Relationeninstanzen R_c mit Tupeln

$$[a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc]$$

- $c \geq c_i$
- a_i ist sichtbar, wenn $class\ (s) \geq c_i$

Integritätsbedingungen

Sei κ *sichtbarer Schlüssel* der Multilevel-Relation R

Entity-Integrität. R erfüllt die Entity-Integrität genau dann, wenn für alle Instanzen R_c und $r \in R_c$ die folgenden Bedingungen gelten:

1. $A_i \in \kappa \Rightarrow r.A_i \neq \text{Null}$
2. $A_i, A_j \in \kappa \Rightarrow r.C_i = r.C_j$
3. $A_i \notin \kappa \Rightarrow r.C_i \geq r.C_\kappa$ (wobei C_κ die Zugriffsklasse des Schlüssels ist)

Null-Integrität. R erfüllt die Null-Integrität genau dann, wenn für jede Instanz R_c von R gilt:

1. $\forall r \in R_c, r.A_i = \text{Null} \Rightarrow r.C_i = r.C_\kappa$
2. R_c ist subsumierungsfrei, d.h. es existieren keine zwei Tupel r und s, bei denen für alle Attribute A_i entweder
   - $r.A_i = s.A_i$ und $r.C_i = s.C_i$ oder
   - $r.A_i \neq \text{Null}$ und $s.A_i = \text{Null}$ gilt.

Subsumptionsfreiheit von Relationen

a) R_{sg}

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	–	g

b) Änderung von R_{sg}

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
sg	007	g	Blond, James	g	meucheln	sg

c) Fehlende Subsumtionsfreiheit

Agenten						
TC	Kennung	KC	Name	NC	Spezialität	SC
g	007	g	Blond, James	g	–	g
sg	007	g	Blond, James	g	meucheln	sg

Interinstanz-Integrität. R erfüllt die Interinstanz-Integrität genau dann, wenn für alle Instanzen R_c und $R_{c'}$ von R mit $c' < c$

$$R_{c'} = f(R_c, c')$$

gilt. Die Filterfunktion f arbeitet wie folgt:

1. Für jedes $r \in R_c$ mit $r.C_\kappa \leq c'$ muß ein Tupel $s \in R_{c'}$ existieren, mit
$$s.A_i = \begin{cases} r.A_i & \text{wenn } r.C_i \leq c' \\ \text{Null} & \text{sonst} \end{cases}$$
$$s.C_i = \begin{cases} r.C_i & \text{wenn } r.C_i \leq c' \\ r.C_\kappa & \text{sonst} \end{cases}$$

2. $R_{c'}$ enthält außer diesen keine weiteren Tupel.

3. Subsumierte Tupel werden eliminiert.

Integritätsbedingungen

Polyinstanziierungsintegrität. R erfüllt die Polyinstanziierungsintegrität genau dann, wenn für jede Instanz R_c für alle A_i die folgende funktionale Abhängigkeit gilt: $\{\kappa, C_\kappa, C_i\} \rightarrow A_i$.

- Gerade die Gefahr des Abhörens von Kommunikationskanälen ist in heutigen Datenbankarchitekturen und Anwendungen sehr groß.
- Die meisten Datenbankanwendungen werden in einer verteilten Umgebung betrieben – sei es als Client/Server-System oder als „echte“ verteilte Datenbank.
- In beiden Fällen ist die Gefahr des unlegitimierten Abhörens sowohl innerhalb eines LAN (local area network, z.B. Ethernet) als auch im WAN (wide area network, z.B. Internet) gegeben und kann technisch fast nicht ausgeschlossen werden.
- Deshalb kann nur die Verschlüsselung der gesendeten Information einen effektiven Datenschutz gewährleisten.

Ebenen des Datenschutzes

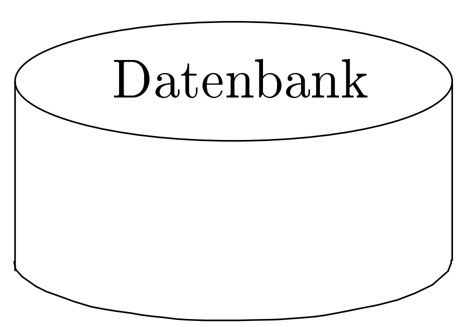
legislative Maßnahmen

organisatorische Maßnahmen

Authentisierung

Zugriffskontrolle

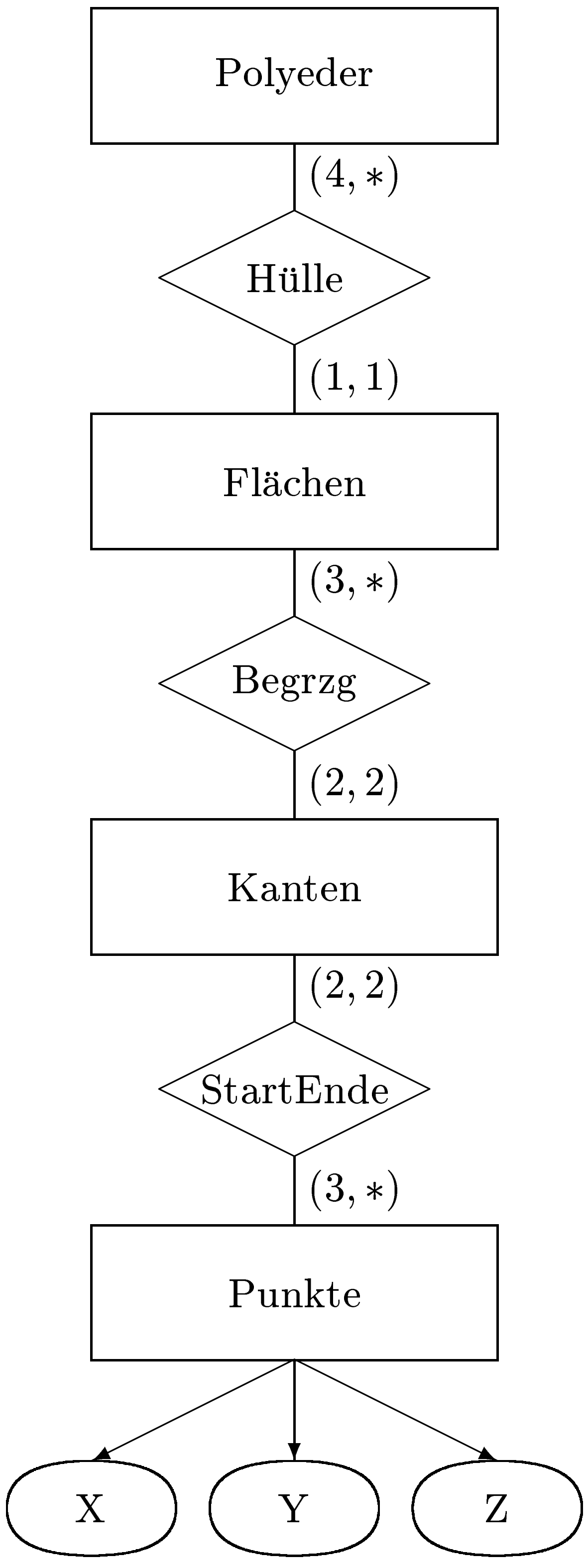
Kryptographie



Objektorientierte Datenbanken

- die nächste Generation der Datenbanktechnologie?
- A. Kemper, G. Moerkotte
  Object-Oriented Database Management: Applications in Engineering and Computer Science, Prentice Hall, 1994.
- ca 12 kommerzielle Produkte
- seit 1993 erster Standard (ODMG)

Nachteile relationaler Modellierung



(a)

Polyeder			
PolyID	Gewicht	Material	...
cubo#5	25.765	Eisen	...
tetra#7	37.985	Glas	...
...	...	...	...

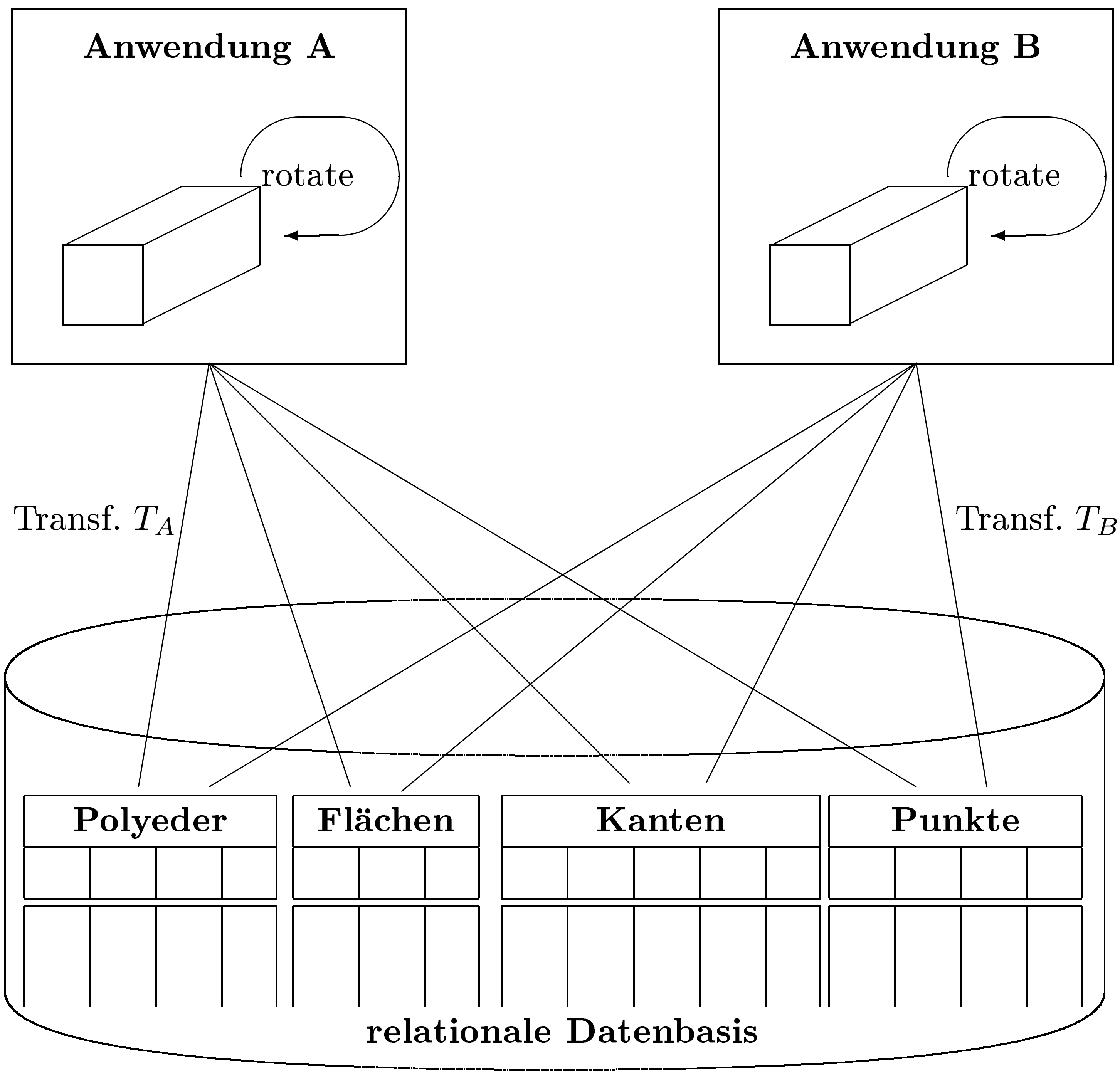
Flächen		
FlächenID	PolyID	Oberfläche
f1	cubo#5	...
f2	cubo#5	...
...	...	...
f6	cubo#5	...
f7	tetra#7	...

Kanten				
KantenID	F1	F2	P1	P2
k1	f1	f4	p1	p4
k2	f1	f2	p2	p3
...	...	...	...	...

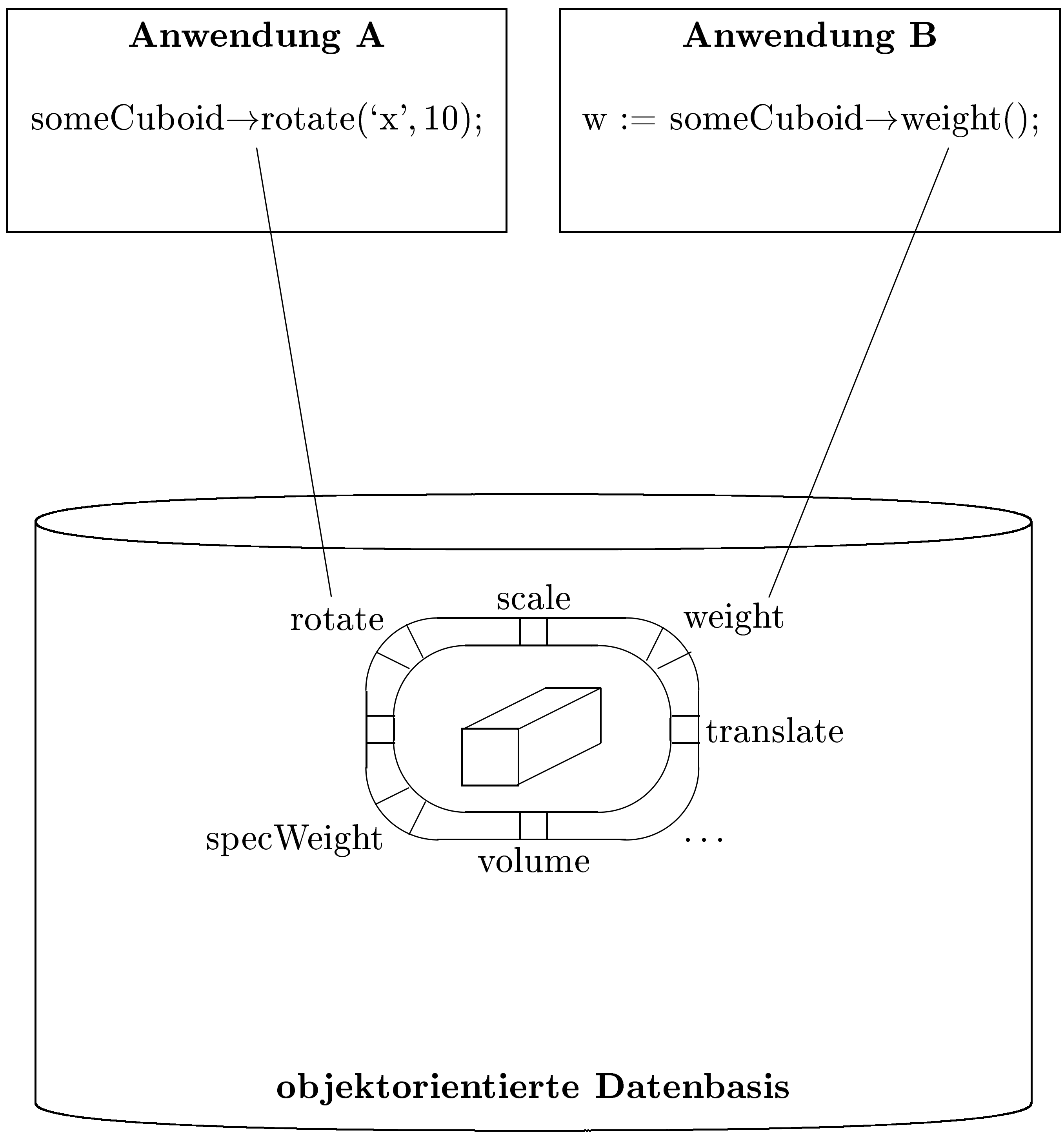
Punkte			
PunktID	X	Y	Z
p1	0.0	0.0	0.0
p2	1.0	0.0	0.0
...	...	...	...
p8	0.0	1.0	1.0
...	...	...	...

(b)

Visualisierung des „Impedance Mismatch“



Vorteile objektorientierter Datenmodellierung



ODMG-Standardisierung

Beteiligte

- SunSoft (Organisator: R. Cattell)
- Object Design
- Ontos
- O_2Technology
- Versant
- Objectivity

Reviewer

- Hewlett-Packard
- Poet
- Itasca
- Intellitic
- DEC
- Servio
- Texas Instruments

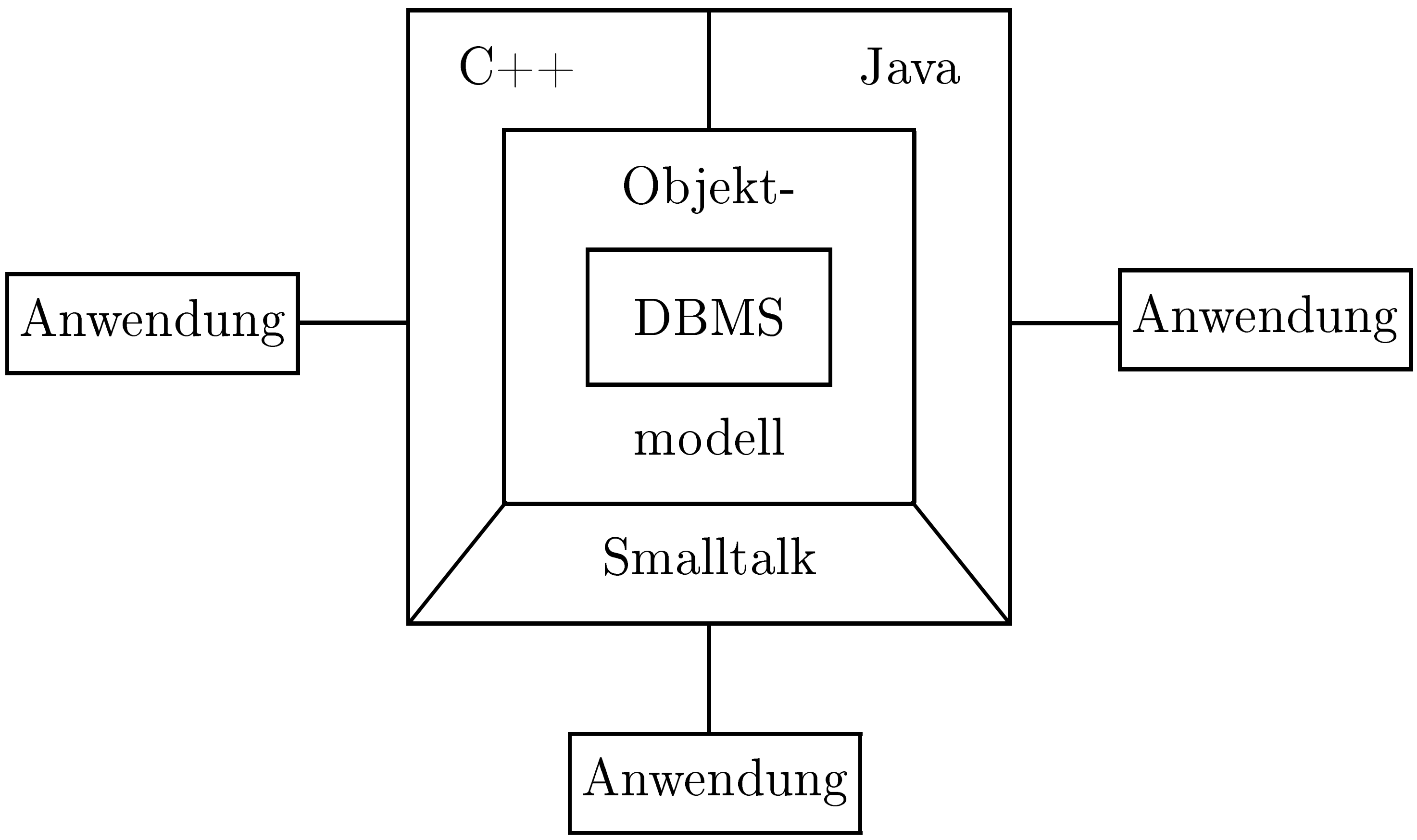
Bestandteile des Standards

1. Objektmodell
2. Object Definition Language (ODL)
3. Object Query Language (OQL)
4. C++ Anbindung
5. Smalltalk Anbindung

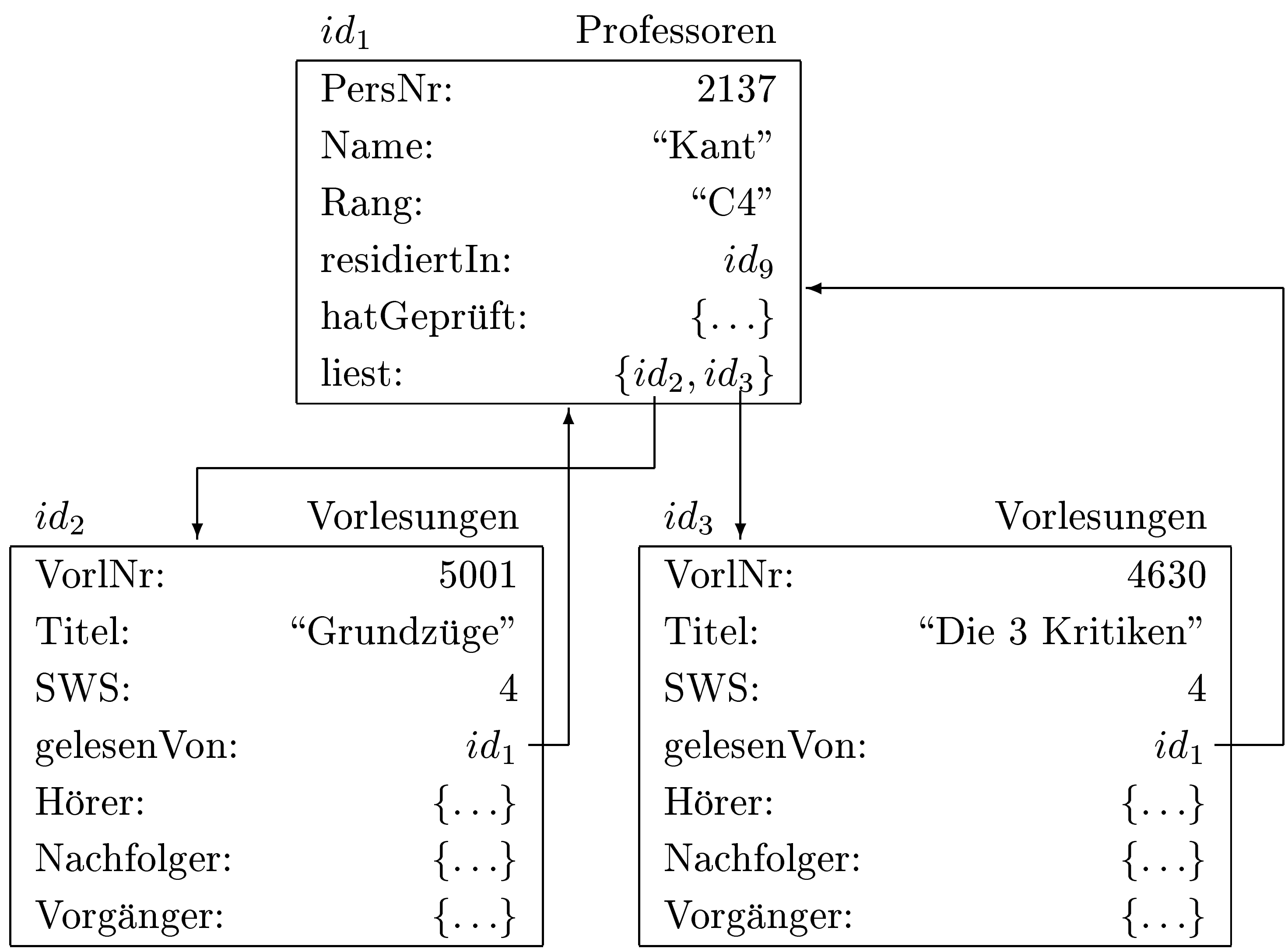
Motivation der Standardisierung

- Portabilitäts-Standard
- kein Interoperabilitäts-Standard

Integration des ODMG-Objektmodells

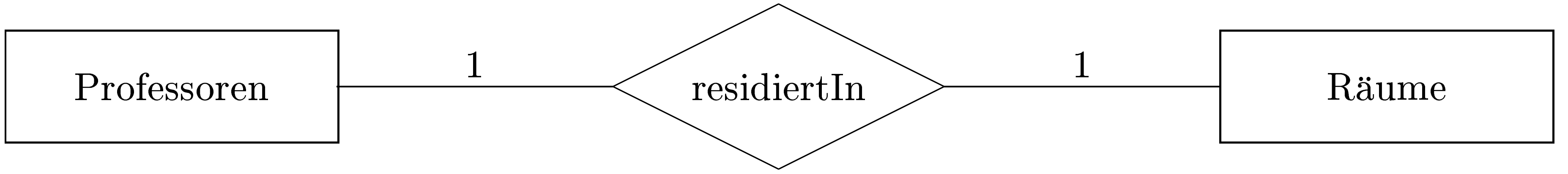


Einige Objekte aus der Universitätswelt



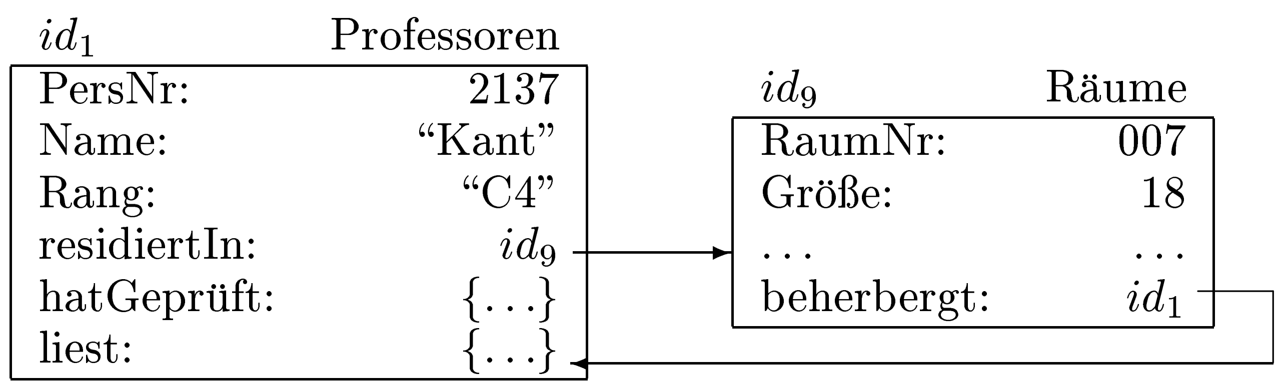

```
class Professoren {
    attribute long PersNr;
    attribute string Name;
    attribute string Rang;
};
```

1 : 1-Beziehungen



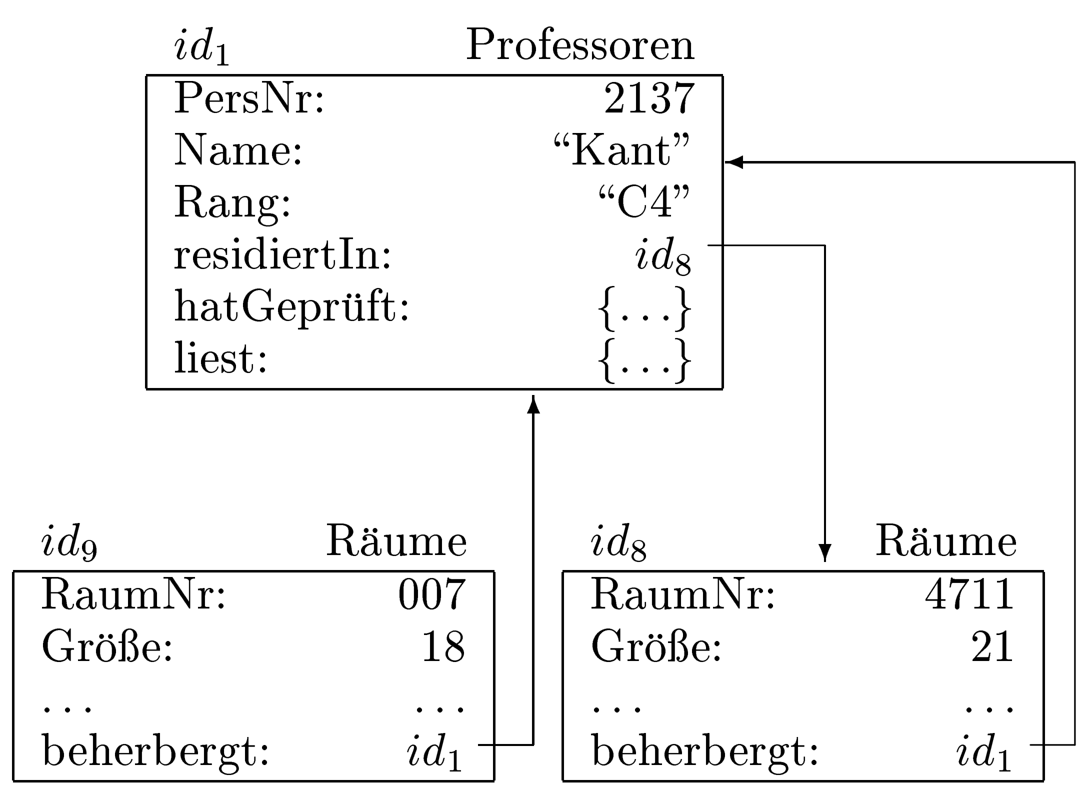
class Professoren {
    attribute long PersNr;
    ...
    relationship Räume residiertIn;
};
class Räume {
    attribute long RaumNr;
    attribute short Größe;
    ...
    relationship Professoren beherbergt;
};

Beispielausprägungen



Nachteile

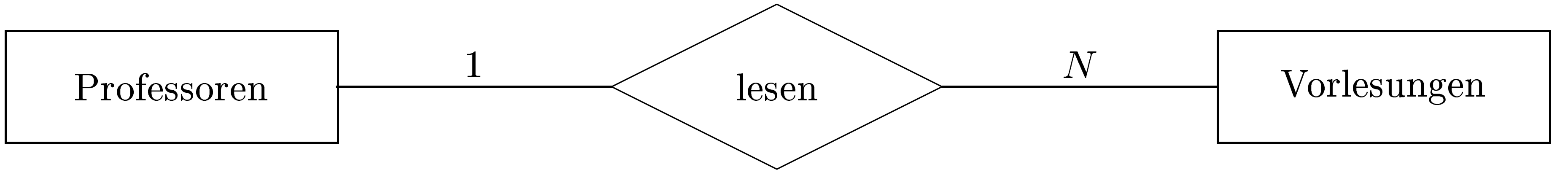
- Verletzung der Symmetrie
- Verletzung der 1:1-Einschränkung



Bessere Modellierung mit “inverse”

```
class Professoren {
    attribute long PersNr;
    ...
    relationship Räume residiertIn inverse Räume::beherbergt;
};
class Räume {
    attribute long RaumNr;
    attribute short Größe;
    ...
    relationship Professoren beherbergt inverse Professoren::residiertIn;
};
```

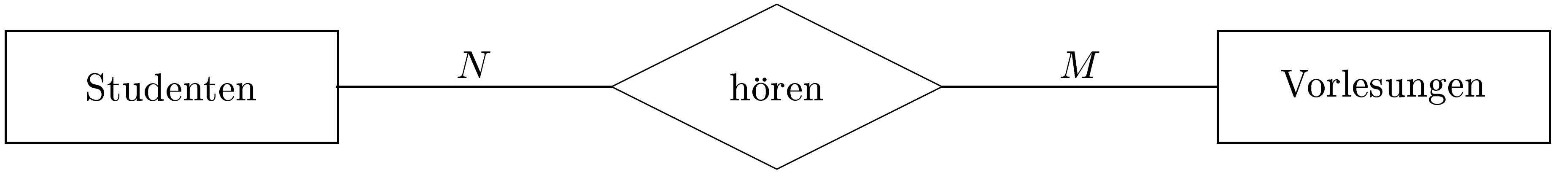
$1 : N$-Beziehungen



class Professoren {
    ...
    relationship set⟨Vorlesungen⟩ liest **inverse** Vorlesungen::gelesenVon;
};

class Vorlesungen {
    ...
    relationship Professoren gelesenVon **inverse** Professoren::liest;
};

$N : M$-Beziehungen



class Studenten {

$\quad\ldots$

$\quad$**relationship set**⟨Vorlesungen⟩ hört **inverse** Vorlesungen::Hörer;

};

class Vorlesungen {

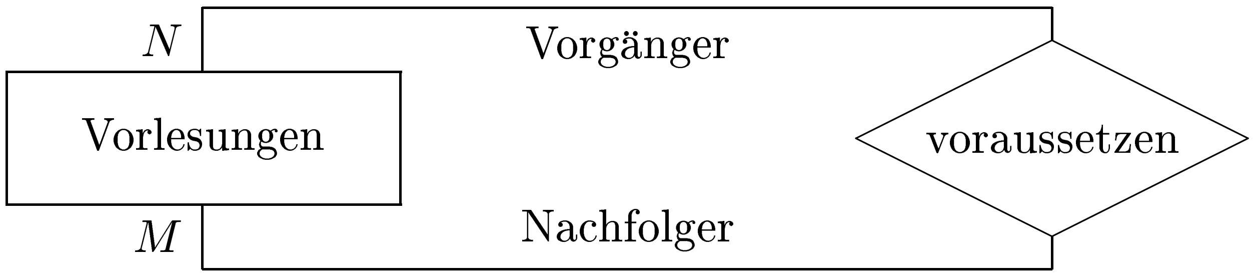
$\quad\ldots$

$\quad$**relationship set**⟨Vorlesungen⟩ Vorgänger **inverse** Vorlesungen::Nachfolger;

$\quad$**relationship set**⟨Vorlesungen⟩ Nachfolger **inverse** Vorlesungen::Vorgänger;

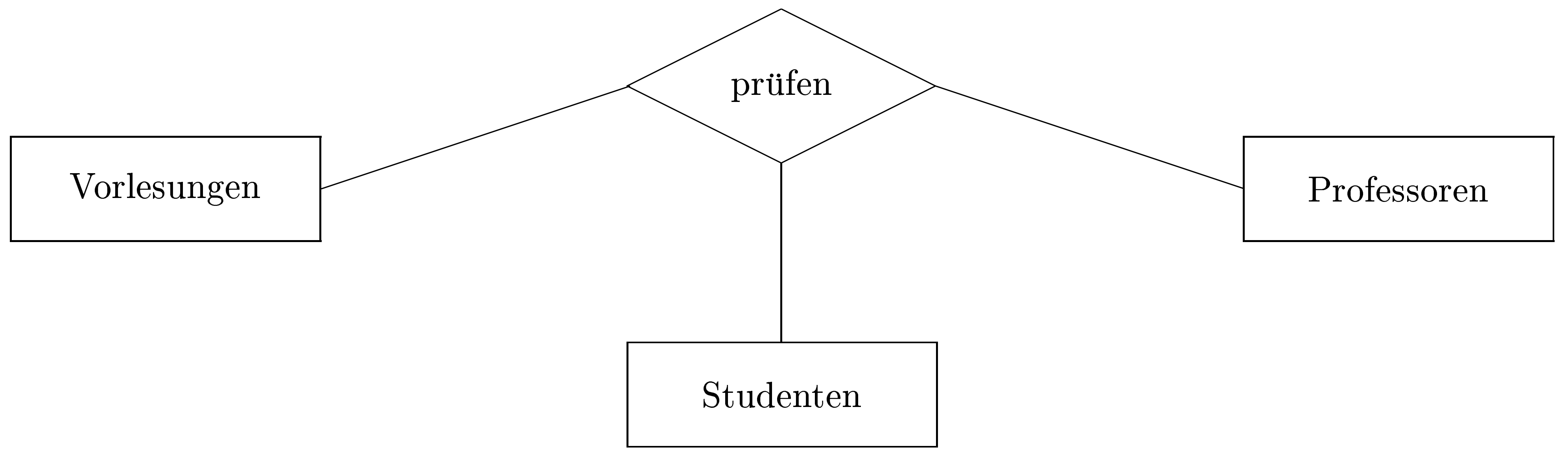
};

Rekursive $N:M$-Beziehungen



class Vorlesungen {
 ...
 relationship set⟨Vorlesungen⟩ Vorgänger **inverse** Vorlesungen::Nachfolger;
 relationship set⟨Vorlesungen⟩ Nachfolger **inverse** Vorlesungen::Vorgänger;
};

Ternäre Beziehungen




```
class Prüfungen {
    attribute struct Datum
        { short Tag; short Monat; short Jahr; } PrüfDatum;
    attribute float Note;
    relationship Professoren Prüfer inverse Professoren::hatGeprüft;
    relationship Studenten Prüfling inverse Studenten::wurdeGeprüft;
    relationship Vorlesungen Inhalt inverse Vorlesungen::wurdeAbgeprüft;
};
```

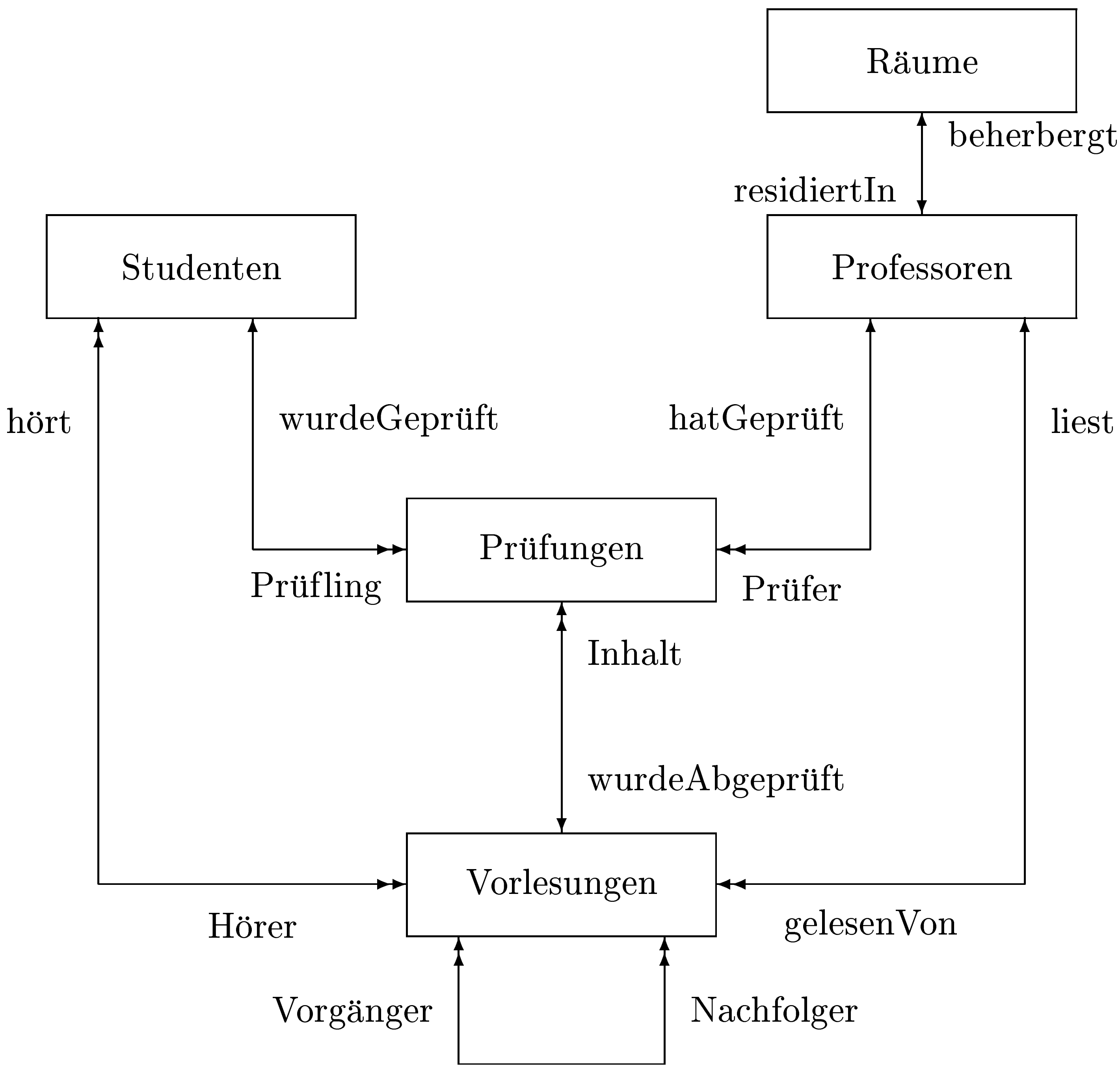
Vervollständigtes Universitäts-Schema

class Professoren {
    attribute long PersNr;
    attribute string Name;
    attribute string Rang;
    relationship Räume residiertIn **inverse** Räume::beherbergt;
    relationship set⟨Vorlesungen⟩ liest **inverse** Vorlesungen::gelesenVon;
    relationship set⟨Prüfungen⟩ hatGeprüft **inverse** Prüfungen::Prüfer;
};

class Vorlesungen {
    attribute long VorlNr;
    attribute string Titel;
    attribute short SWS;
    relationship Professoren gelesenVon
    inverse Professoren::liest;
    relationship set⟨Studenten⟩ Hörer **inverse** Studenten::hört;
    relationship set⟨Vorlesungen⟩ Nachfolger
        inverse Vorlesungen::Vorgänger;
    relationship set⟨Vorlesungen⟩ Vorgänger
        inverse Vorlesungen::Nachfolger;
    relationship set⟨Prüfungen⟩ wurdeAbgeprüft
        inverse Prüfungen::Inhalt;
};

class Studenten {
    ...
    relationship set⟨Prüfungen⟩ wurdeGeprüft
        inverse Prüfungen::Prüfling;
}

Modellierung von Beziehungen im Objektmodell



Typeigenschaften: Extensionen und Schlüssel

class Studenten (**extent** AlleStudenten **key** MatrNr) {
    attribute long MatrNr;
    attribute string Name;
    attribute short Semester
    relationship set⟨Vorlesungen⟩ hört **inverse** Vorlesungen::Hörer;
    relationship set⟨Prüfungen⟩ wurdeGeprüft **inverse** Prüfungen::Prüfling;
};

Modellierung des Verhaltens: Operationen

Operationen um ...

- Objekte zu erzeugen (instanziieren) und zu initialisieren,
- die für Klienten interessanten Teile des Zustands der Objekte zu erfragen,
- legale und konsistenzerhaltende Operationen auf diesen Objekten auszuführen und letztendlich
- die Objekte wieder zu zerstören.

Drei Klassen von Operationen:

1. *Beobachter* (engl. *observer*):
    - oft auch Funktionen genannt
    - Objektzustand „erfragen“
    - Beobachter-Operationen haben keinerlei objektändernde Seiteneffekte
2. *Mutatoren*:
    - Änderungen am Zustand der Objekte.
    - Einen Objekttyp mit mindestens einer Mutator-Operation bezeichnet man als *mutierbar*.

- Objekte eines Typs ohne jegliche Mutatoren sind unveränderbar (engl. *immutable*).
- Unveränderbare Typen bezeichnet man oft als *Literale* oder *Wertetypen*.

3. *Konstruktoren und Destruktoren*:

- Erstere werden verwendet, um neue Objekte eines bestimmten Objekttyps zu erzeugen.
- Instanziierung.
- Der Destruktor wird dazu verwendet, ein existierendes Objekt auf Dauer zu zerstören
- Konstruktoren werden sozusagen auf einem Objekttyp angewandt, um ein neues Objekt zu erzeugen.
- Destruktoren werden demgegenüber auf existierende Objekte angewandt und können demnach eigentlich auch den Mutatoren zugerechnet werden.

Klassen-Definition von Operationen in ODL

Man spezifiziert

- den Namen der Operation;
- die Anzahl und die Typen der Parameter;
- den Typ des Rückgabewerts der Operation;
- eine eventuell durch die Operationsausführung ausgelöste *Ausnahmebehandlung* (engl. *exception handling*).

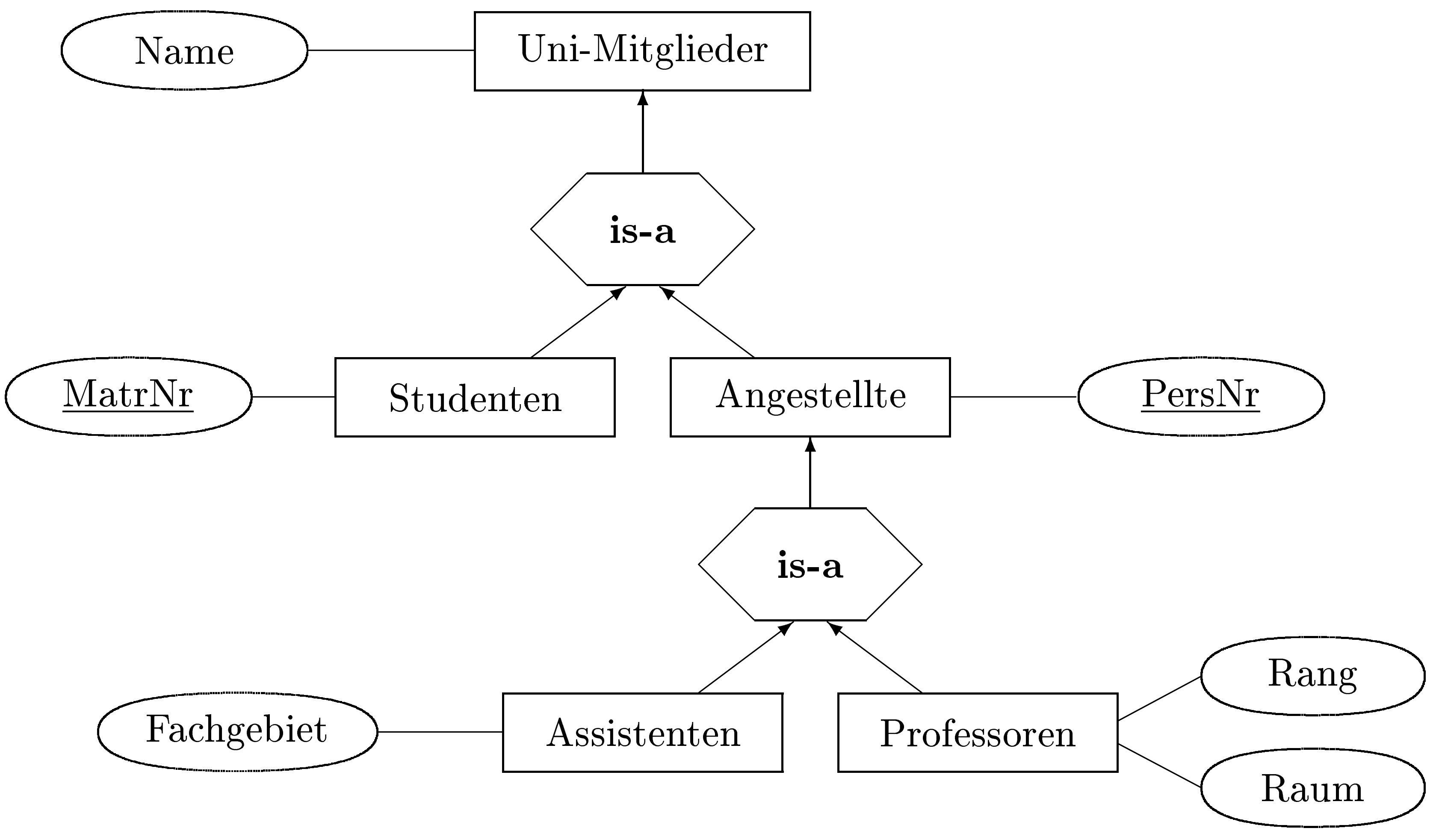
Beispiel-Operationen

```
class Professoren {
    exception hatNochNichtGeprüft { };
    exception schonHöchsteStufe { };
    ...
    float wieHartAlsPrüfer() raises (hatNochNichtGeprüft);
    void befördert() raises (schonHöchsteStufe);
};
```

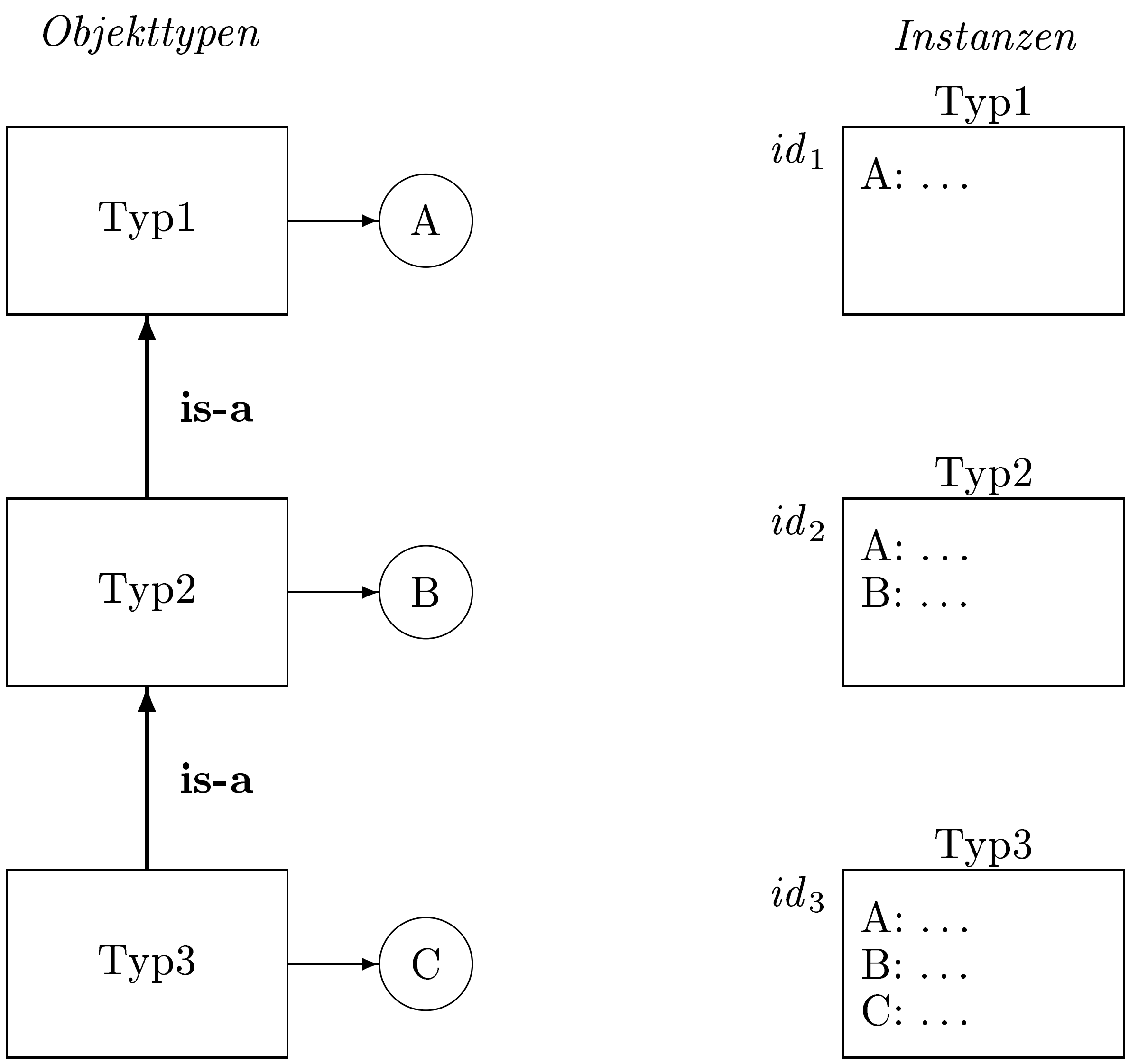
Aufruf der Operationen

im Anwendungsprogramm:	in OQL:
meinLieblingsProf→befördert();	**select** p.wieHartAlsPrüfer() **from** p **in** AlleProfessoren **where** p.Name = “Curie”;

Vererbung und Subtypisierung

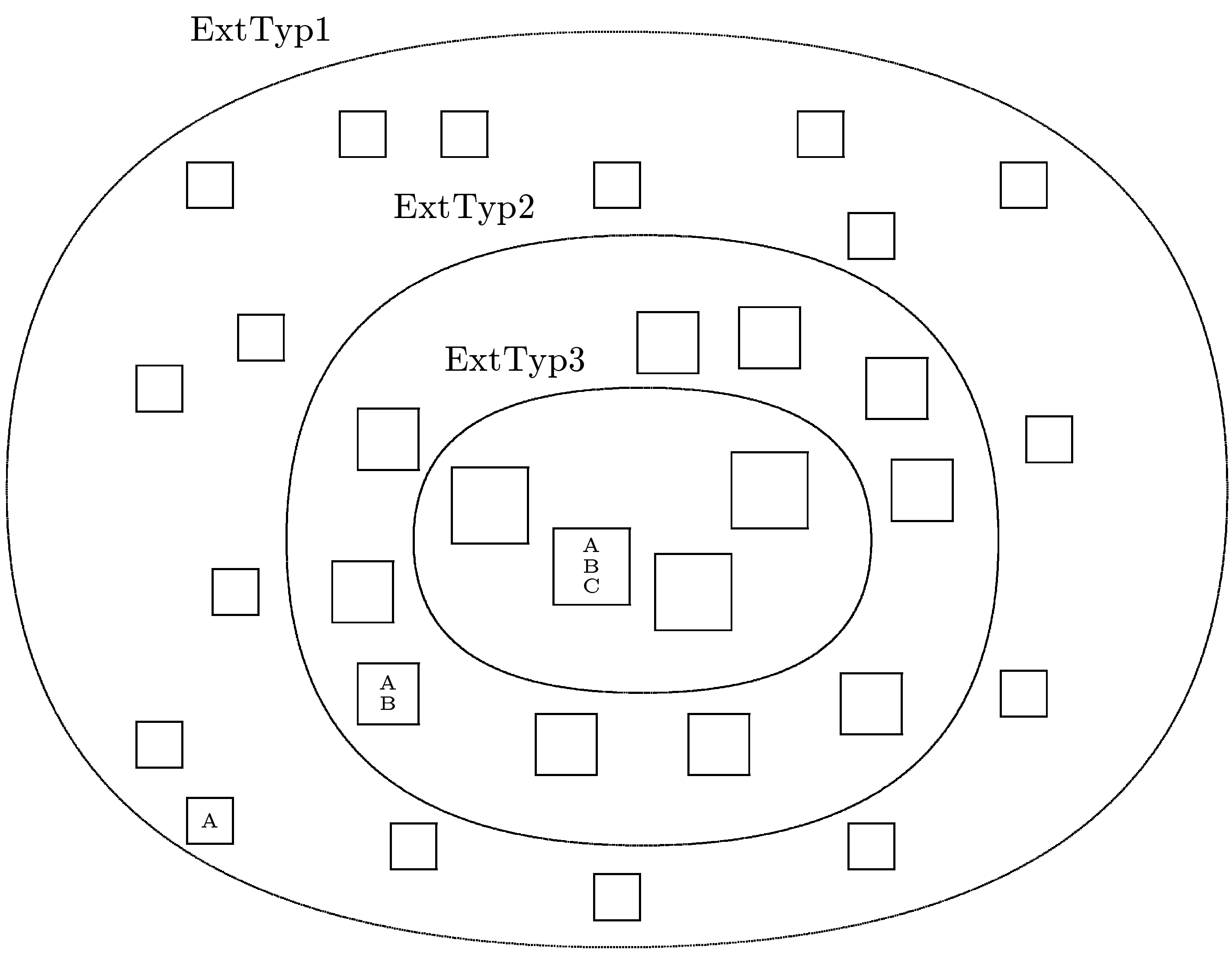


Terminologie



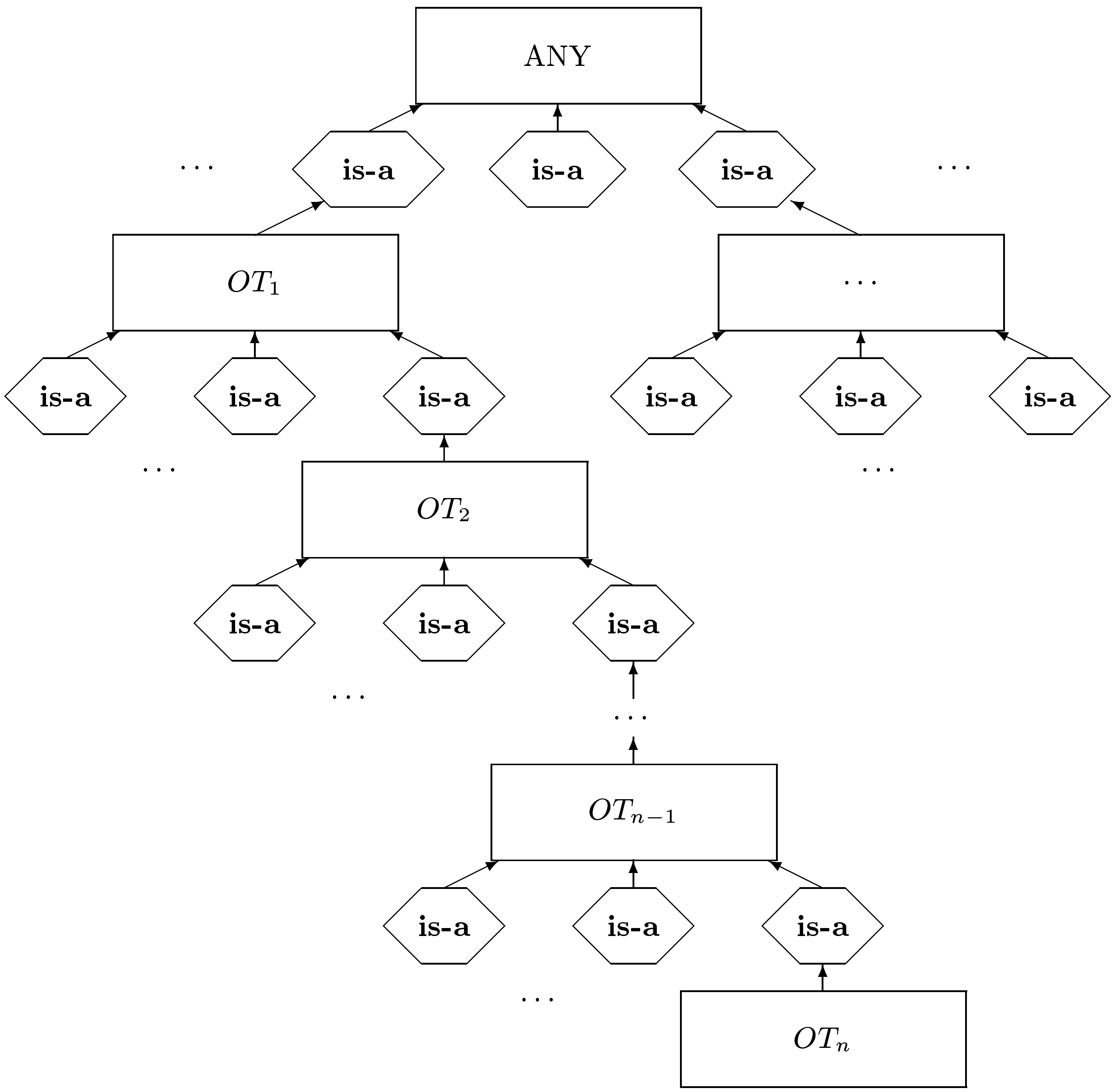
- Untertyp/Obertyp
- Instanz eines Untertyps gehört auch zur Extension des Obertyps
- Vererbung der Eigenschften eines Obertyps an den Untertyp

Darstellung der Subtypisierung



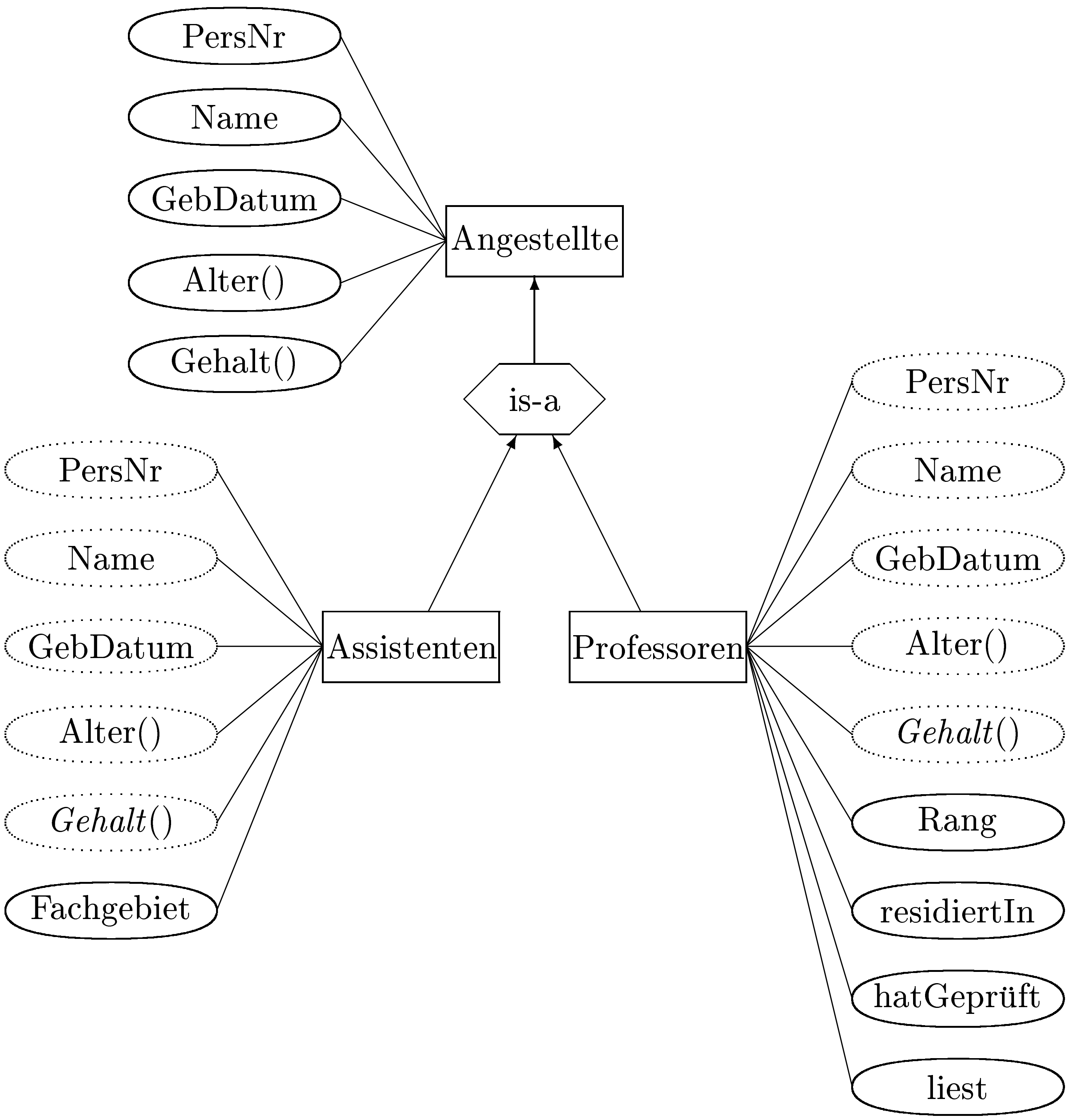
- Inklusionspolymorphismus
- Subtituierbarkeit
  - Eine Untertyp-Instanz ist überall dort einsetzbar, wo eine Obertyp-Instanz gefordert ist.

Abstrakte Typhierarchie bei Einfach-Vererbung



eindeutiger Pfad : $OT_n \rightarrow OT_{n-1} \rightarrow \ldots \rightarrow OT_2 \rightarrow OT_1 \rightarrow ANY$

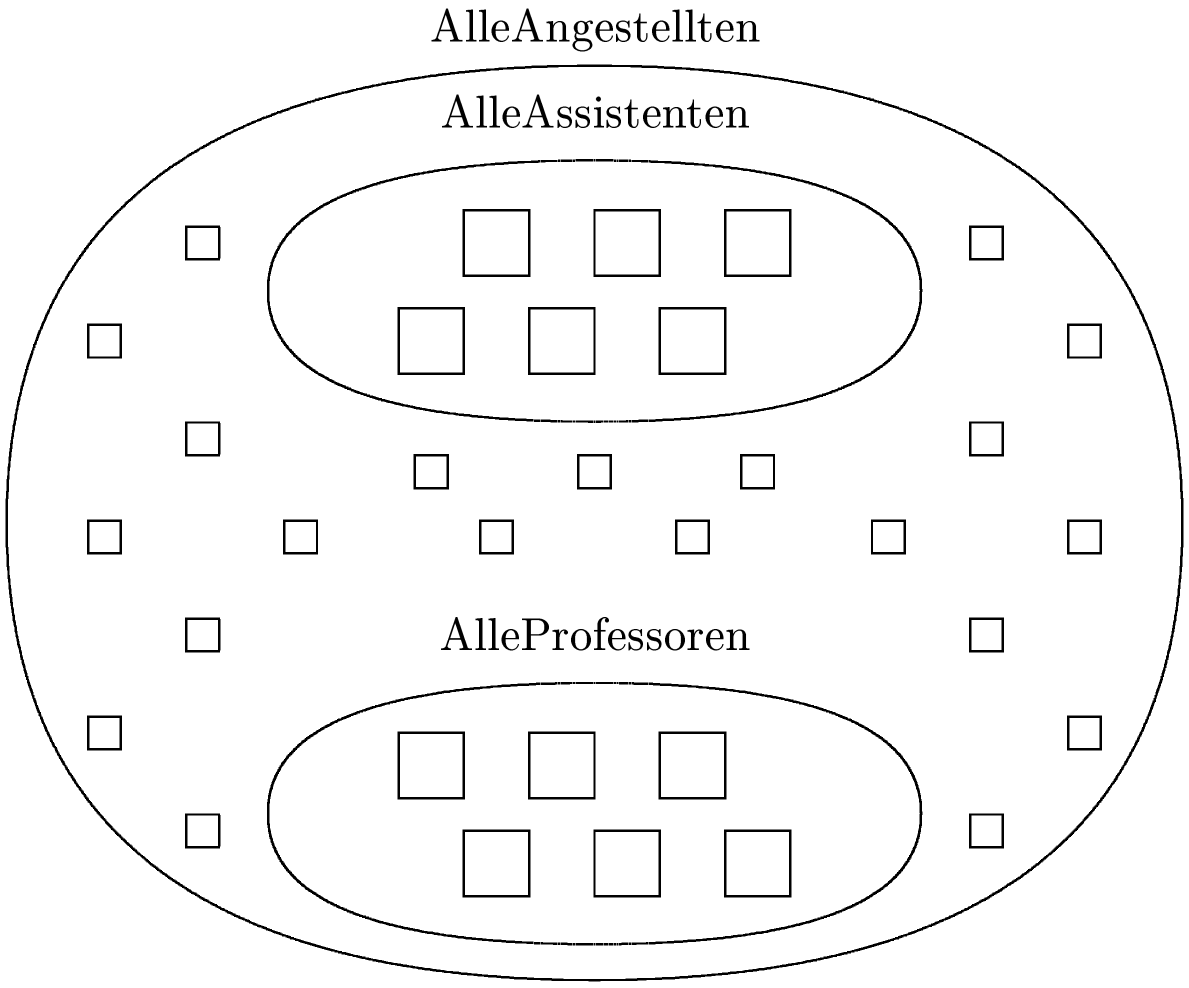
Vererbung von Eigenschaften



Interface-Definition in ODL

```
class Angestellte (extent AlleAngestellten) {
    attribute long PersNr;
    attribute string Name;
    attribute date GebDatum;
    short Alter();
    long Gehalt();
};
class Assistenten extends Angestellte (extent AlleAssistenten) {
    attribute string Fachgebiet;
};
class Professoren extends Angestellte (extent AlleProfessoren) {
    attribute string Rang;
    relationship Räume residiertIn inverse Räume::beherbergt;
    relationship set⟨Vorlesungen⟩ liest inverse Vorlesungen::gelesenVon;
    relationship set⟨Prüfungen⟩ hatGeprüft inverse Prüfungen::Prüfer;
};
```

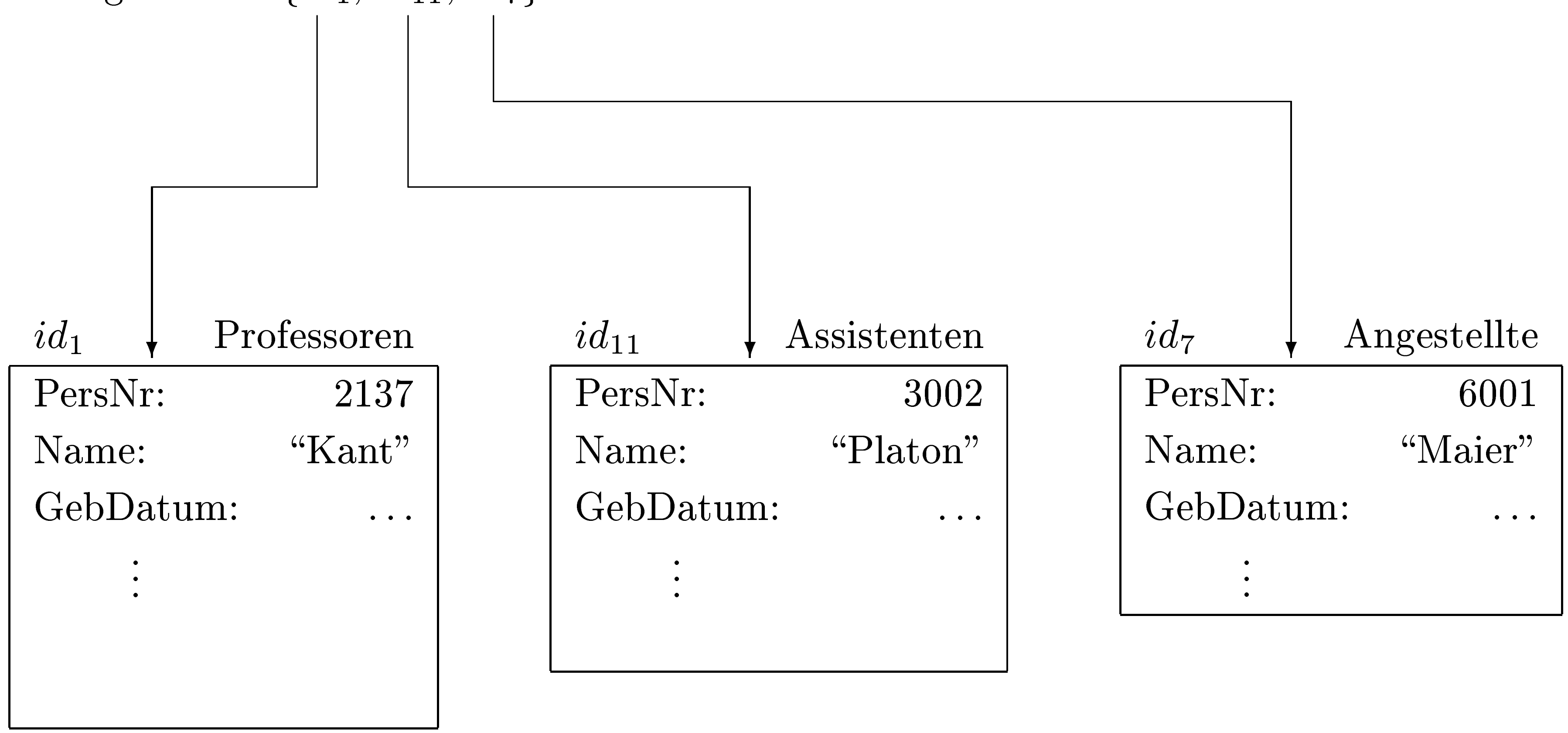
Darstellung der Extensionen



Verfeinerung und spätes Binden

- Die Extension *AlleAngestellten* mit (nur) drei Objekten

AlleAngestellten:$\{id_1, id_{11}, id_7\}$



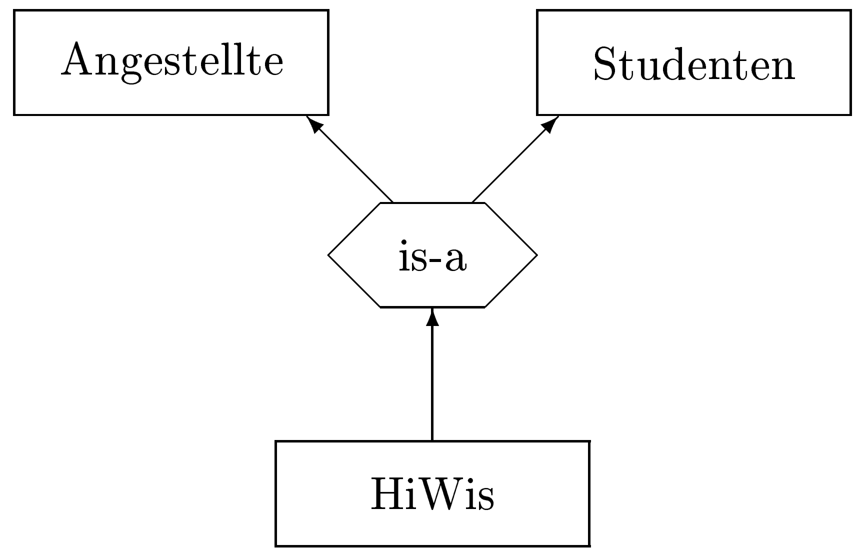
Verfeinerung (Spezialisierung) der Operation Gehalt

- *Angestellte* erhalten: $2000 + (\text{Alter}() - 21) * 100$
- *Assistenten* bekommen: $2500 + (\text{Alter}() - 21) * 125$
- *Professoren* erhalten: $3000 + (\text{Alter}() - 21) * 150$

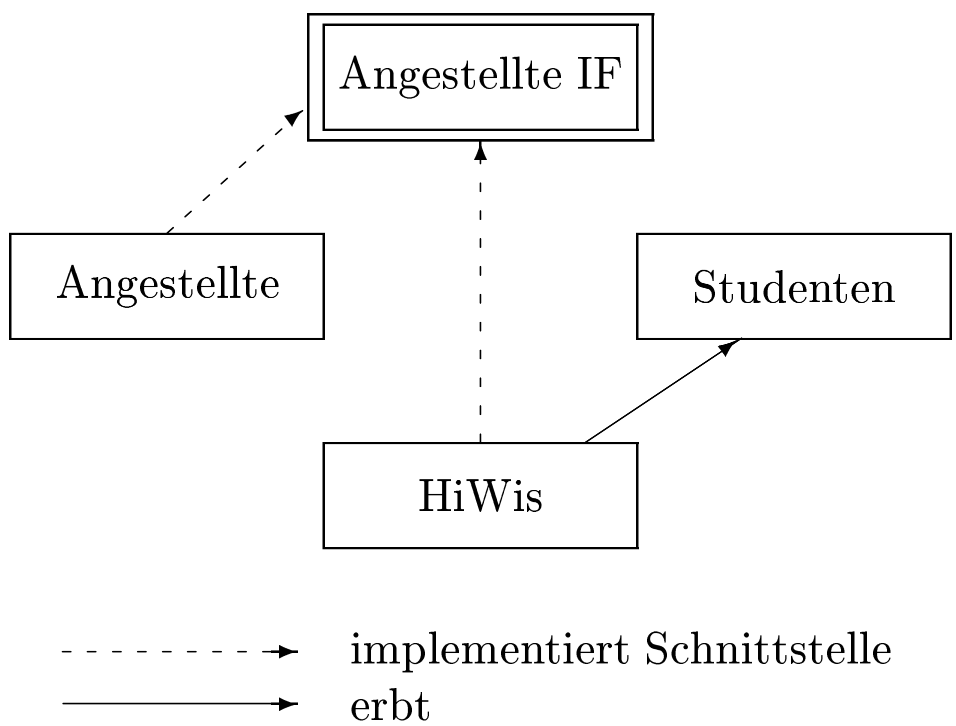
  select sum(a.Gehalt())
  from a **in** AlleAngestellten

- für das Objekt wird id_1 die *Professoren*-spezifische *Gehalt*-Berechnung durchgeführt,
- für das Objekt id_{11} die *Assistenten*-spezifische und
- für das Objekt id_7 die allgemeinste, also *Angestellten*-spezifische Realisierung der Operation *Gehalt* gebunden.

Graphik: Mehrfachvererbung



- geht so in ODMG nicht
- eine Klasse kann nur von **einer** Klasse erben
- sie kann aber mehrere Interfaces implementieren – à la Java



Interface-/Klassendefinition in ODL

class HiWis **extends** Studenten, Angestellte (**extent** AlleHiWis) {
  attribute **short** Arbeitsstunden;
  ...
};

interface AngestellteIF {
  short Alter();
  long Gehalt();
};

class Angestellte : AngestellteIF (**extent** AlleAngestellten) {
  attribute **long** PersNr;
  attribute **string** Name;
  attribute **date** GebDatum;
};

class HiWis **extends** Studenten : AngestellteIF (**extent** AlleHiWis) {
  attribute **long** PersNr;
  attribute **date** GebDatum;
  attribute **short** Arbeitsstunden;
};

Die Anfragesprache OQL

Einfache Anfragen

- finde die Namen der C4-Professoren

select p.Name
from p **in** AlleProfessoren
where p.Rang = "C4";

- Generiere Namen- und Rang-Tupel der C4-Professoren

select **Struct**(n: p.Name, r: p.Rang)
from p **in** AlleProfessoren
where p.Rang = "C4";

Geschachtelte Anfragen und Partitionierung

select **struct**(n: p.Name, a: **sum**(**select** v.SWS **from** v **in** p.liest))
from p **in** AlleProfessoren
where **avg**(**select** v.SWS **from** v **in** p.liest) > 2;

Pfadausdrücke in OQL-Anfragen

select s.Name
from s **in** AlleStudenten, v **in** s.hört
where v.gelesenVon.Name = “Sokrates”;

- Visualisierung des Pfadausdrucks

- ein längerer Pfadausdruck:
- $\underbrace{\underbrace{\underbrace{\underbrace{eineVorlesung}_{Vorlesungen}.gelesenVon}_{Professoren}.residiertIn}_{Räume}.Größe}_{float}$

Erzeugung von Objekten

Vorlesungen(VorlNr: 5555, Titel: “Ethik II”, SWS: 4, gelesenVon: (
    select p
    from p **in** AlleProfessoren
    where p.Name = “Sokrates”));

Operationsaufruf in OQL-Anfragen

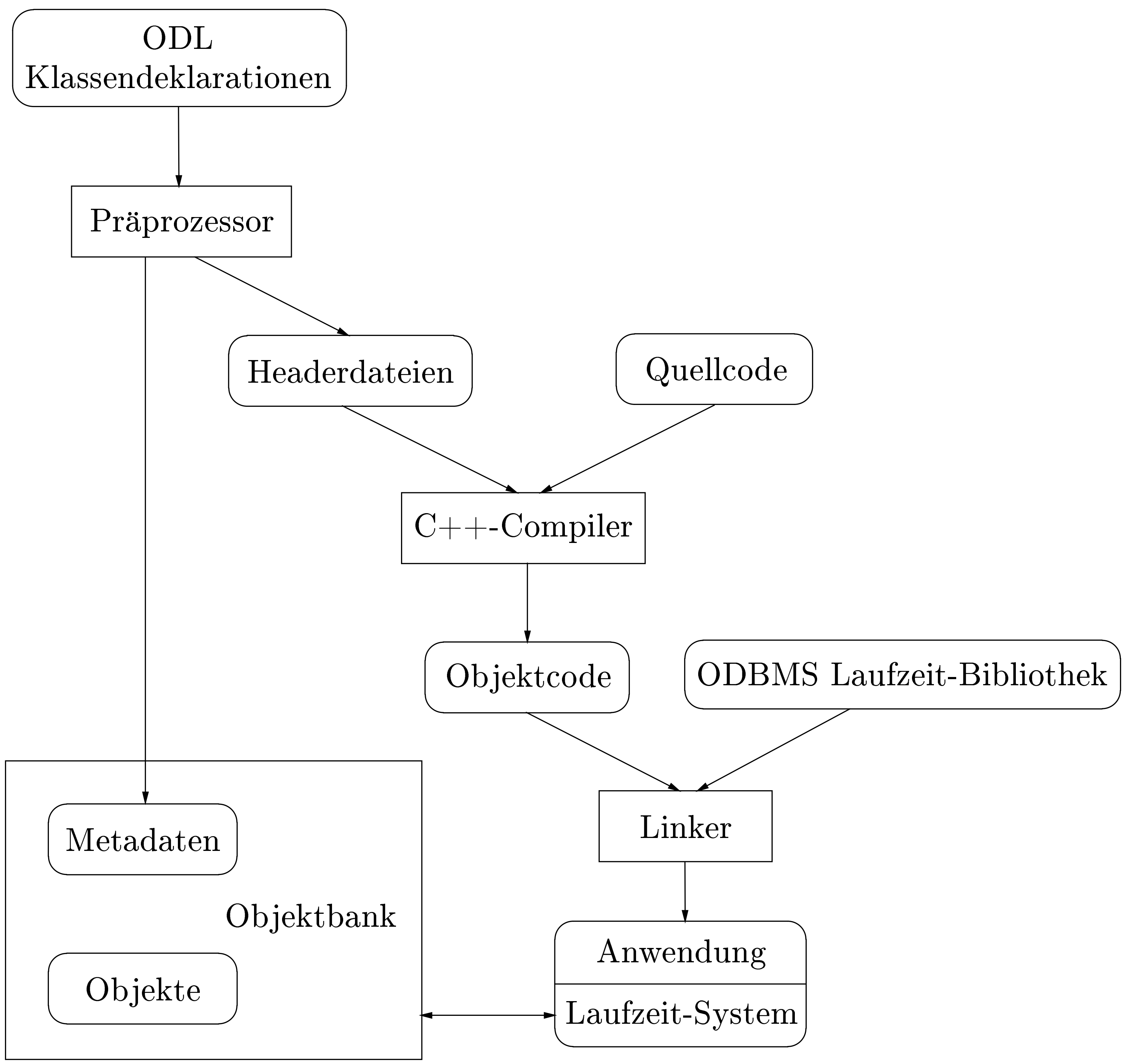
select a.Name
from a **in** AlleAngestellte
where a.Gehalt() > 100.000;

Programmiersprachen-Anbindung

Enwurfsentscheidung

- **Entwurf einer neuen Sprache**
  - eleganteste Methode,
  - hoher Realisierungsaufwand
  - Benutzer müssen eine neue Programmiersprache lernen
- **Erweiterung einer bestehenden Sprache**
  - Benutzer müssen keine vollständig neue Sprache lernen
  - manchmal unnatürlich wirkende Erweiterungen der Basissprache
- **Datenbankfähigkeiten durch Typbibliothek**
  - einfachste Möglichkeit für das Erreichen von Persistenz
  - mit den höchsten „Reibungsverlusten“
  - evtl. Probleme mit der Transparenz der Einbindung und der Typüberprüfung der Programmiersprache
  - ODMG-Ansatz

C++-Einbindung



```
class Vorlesungen {
    String Titel;
    short SWS;
    Ref⟨Professoren⟩ gelesenVon inverse Professoren::liest;
};
```

Objekterzeugung und Ballung

```
Ref⟨Professoren⟩ Russel = new(UniDB) Professoren(2126, “Russel”, “C4”, ...);
Ref⟨Professoren⟩ Popper = new(Russel) Professoren(2133, “Popper”, “C3”, ...);
```

Transaktionen

- Schachtelung von Transaktionen
- notwendig um Operationen, die TAs repräsentieren, geschachtelt aufrufen zu können.

```
void Professoren::Umziehen(Ref⟨Räume⟩ neuerRaum) {
    Transaction TAumziehen;
    TAumziehen.start();

    ...
    if ( /* Fehler? */ )
        TAumziehen.abort();
    ...

    TAumziehen.commit();
};
```

Einbettung von Anfragen

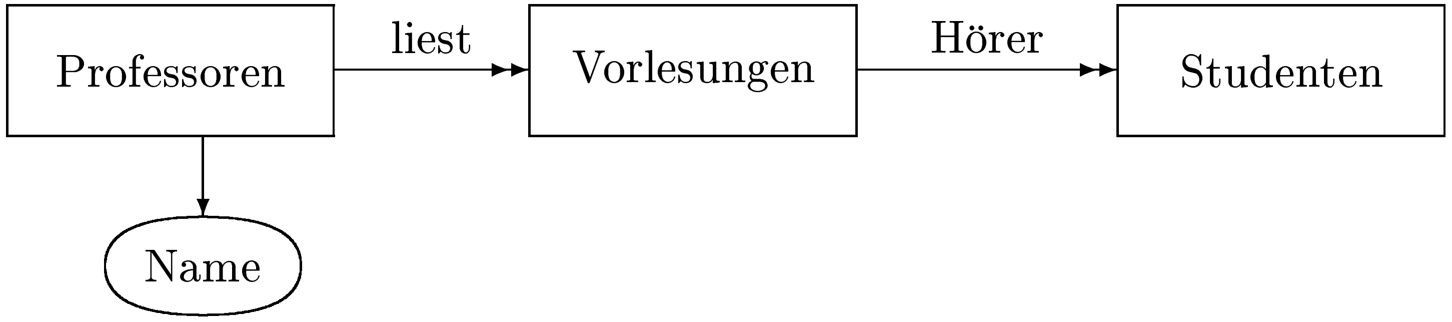
d_Bag⟨Studenten⟩ Schüler;
char* profname = ...;

d_OQL_Query anfrage("**select** s
from s **in** v.Hörer, v **in** p.liest, p **in** AlleProfessoren
where p.Name = $1");

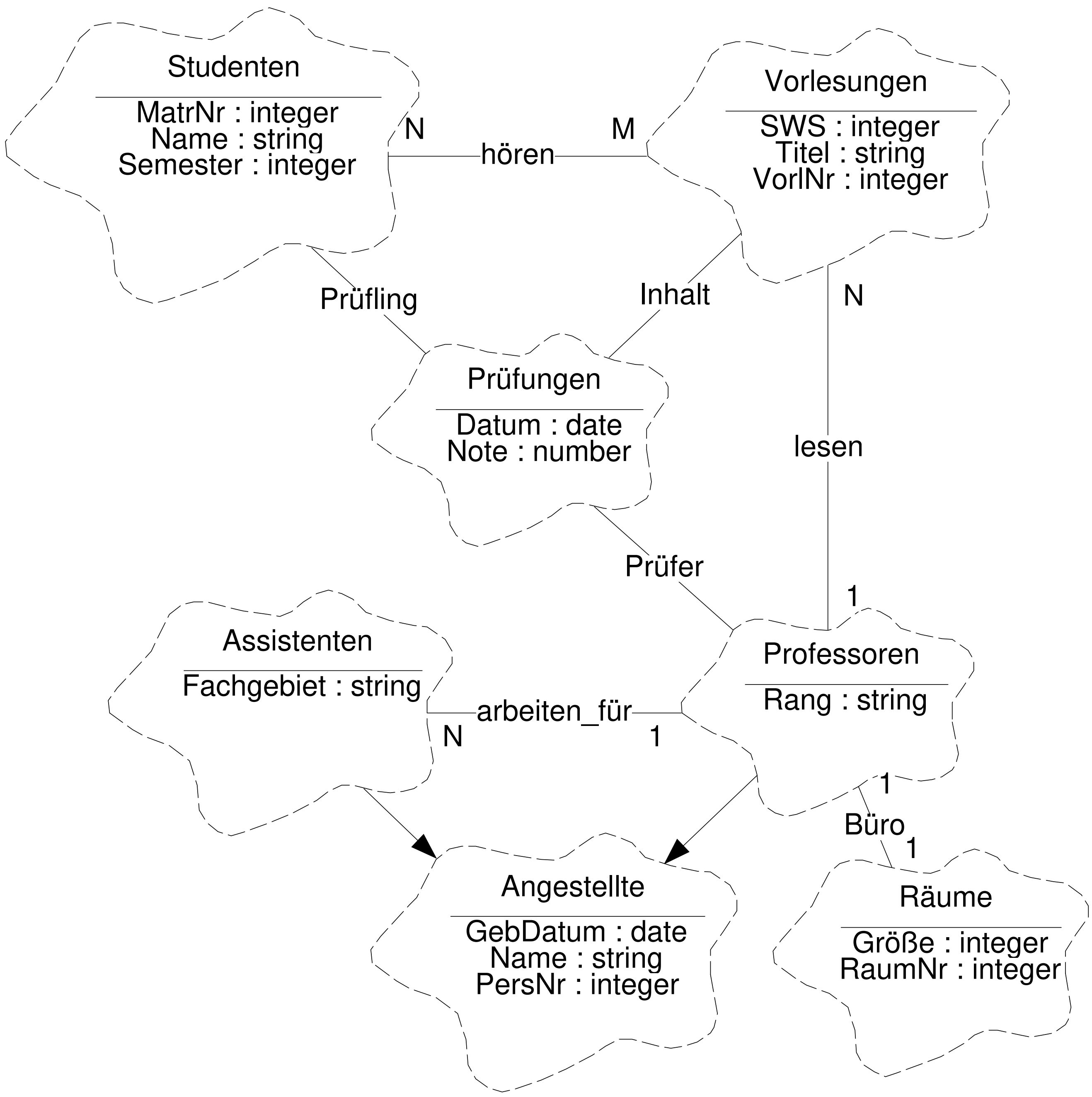
anfrage ≪ profname;

d_oql_execute(anfrage, Schüler);

Graphische Darstellung des Pfadausdrucks



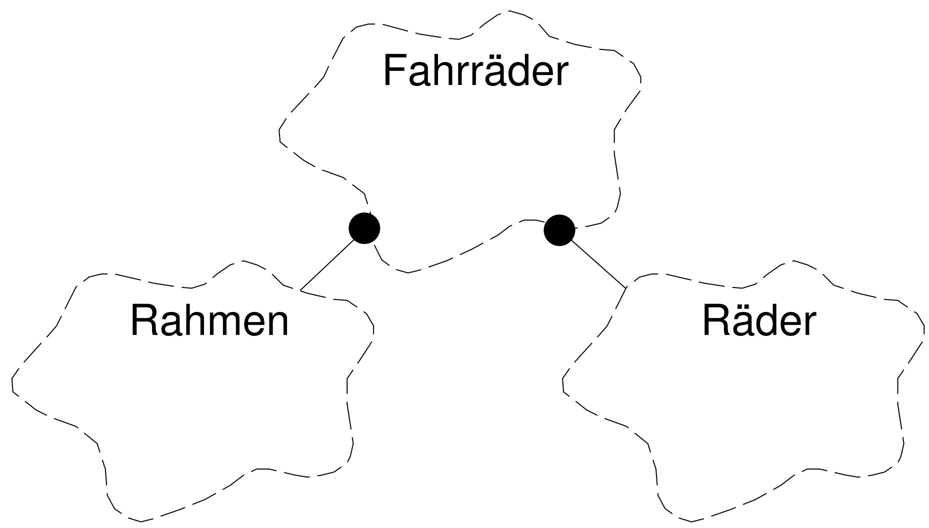
Objektorientierte Modellierung



Objektorientierte Entwurfsmethode

- Booch-Notation
  - Grady Booch: Object-oriented Analysis and Design,
    The Benjamin/Cummings Publication Company, Inc., Redwood City, California, 1994.
  - Rational Rose ist ein System, das die Booch-Notation unterstützt
- Rumbaugh-Notation
  - Rumbaugh, Blaha, Premerlani, Eddy, Lorensen: Object-oriented Modelling and Design,
    Prentice Hall, Englewood Cliffs, New Jersey, 1991.
- Derzeit werden die beiden Methoden (Notationen) „vereinigt“

Modellierung der Aggregation von Objekten



Modellierung des Verhaltens von Objekten

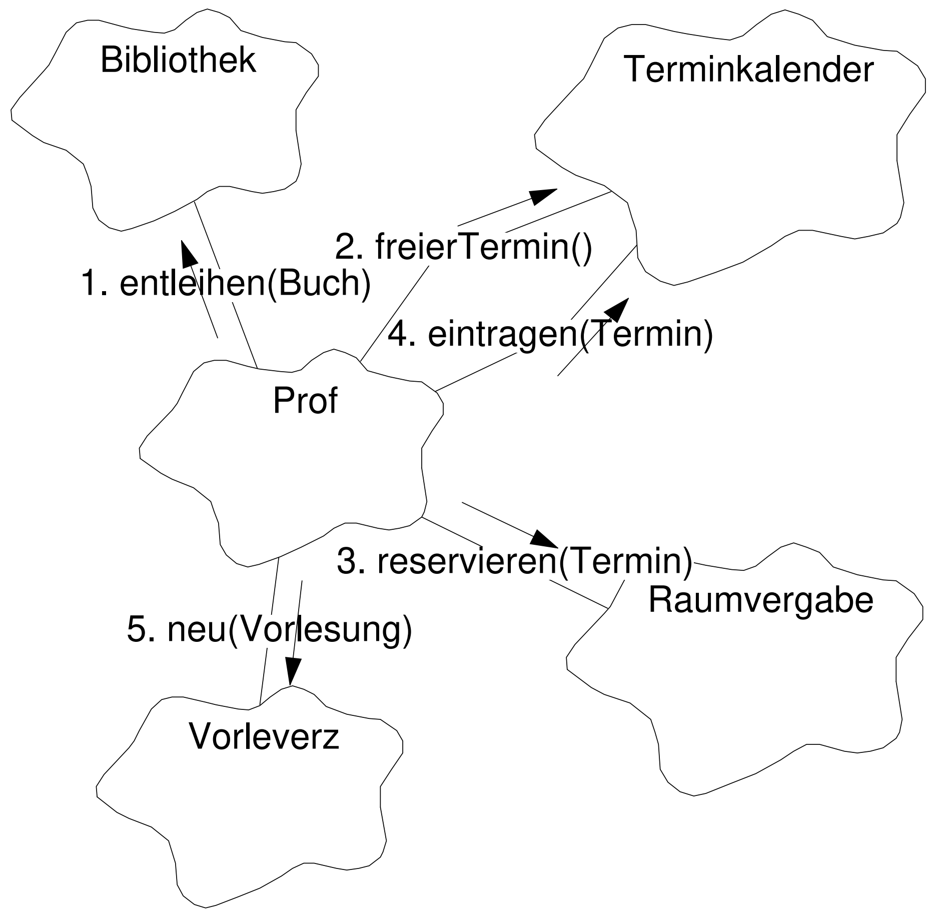
Use Cases

Neue Vorlesung anbieten.

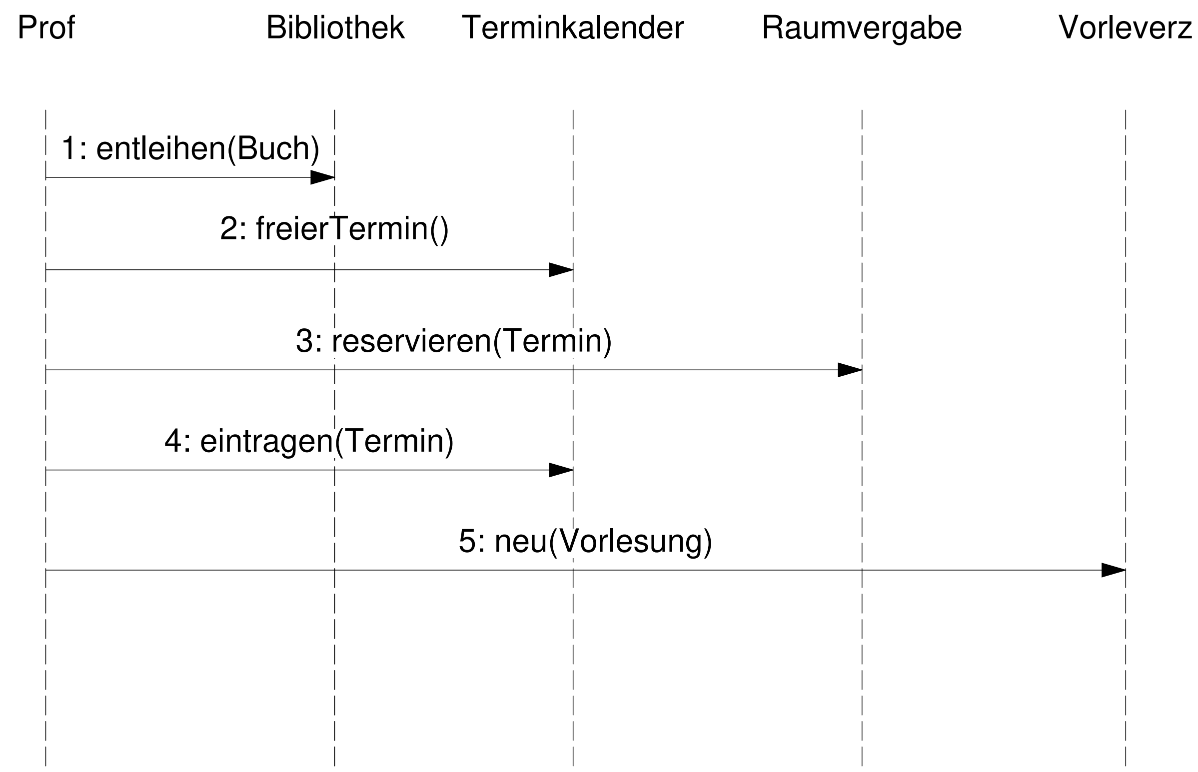
1. begleitende Literatur auswählen
2. möglichen Vorlesungstermin bestimmen
3. Hörsaal reservieren
4. Eintrag ins Vorlesungsverzeichnis vornehmen

Neue Vorlesung anbieten

1. Leihe Buch aus, falls vorhanden, ansonsten Abbruch.
2. Wähle nächsten freien Termin.
3. Reserviere Raum, falls möglich, ansonsten gehe zu Schritt 2.
4. Belege Termin im Terminkalender.
5. Trage Vorlesung im Vorlesungsverzeichnis ein.


Bibliothek
Terminkalender
2. freierTermin()
1. entleihen(Buch)
4. eintragen(Termin)
Prof
3. reservieren(Termin)
Raumvergabe
5. neu(Vorlesung)
Vorleverz

Ein Interaktionsdiagramm



Kommerziell verfügbare Produkte

GemStone

Illustra

Itasca

MATISSE

O_2

Objectivity/DB

ObjectStore

Ontos

OpenODB

Poet

UniSQL

Statice

Versant

Objektrelationale Datenbanken

- Mengenwertige Attribute
- Typdeklarationen
- Referenzen
- Objektidentität
- Pfadausdrücke
- Vererbung
- Operationen

Objektrelationales Schema in Oracle8

```
CREATE TYPE line_item_t
CREATE TYPE purchase_order_t
CREATE TYPE stock_info_t

CREATE TYPE phone_list_t AS VARRAY(10) OF VARCHAR2(20) ;


CREATE TYPE address_t AS OBJECT (
  street  VARCHAR2(200),
  city    VARCHAR2(200),
  state   CHAR(2),
  zip     VARCHAR2(20)
  ) ;

CREATE TYPE customer_info_t AS OBJECT (
  custno     NUMBER,
  custname   VARCHAR2(200),
  address    address_t,
  phone_list phone_list_t,

  ORDER MEMBER FUNCTION
    cust_order(x IN customer_info_t) RETURN INTEGER,

  PRAGMA RESTRICT_REFERENCES (
    cust_order,  WNDS, WNPS, RNPS, RNDS)
  ) ;
```

```
CREATE TYPE line_item_t AS OBJECT (
  lineitemno NUMBER,
  stockref   REF stock_info_t,
  quantity   NUMBER,
  discount   NUMBER
  ) ;

CREATE TYPE line_item_list_t AS TABLE OF line_item_t ;

CREATE TYPE purchase_order_t AS OBJECT (
  pono           NUMBER,
  custref        REF customer_info_t,
  orderdate      DATE,
  shipdate       DATE,
  line_item_list line_item_list_t,
  shiptoaddr     address_t,

  MAP MEMBER FUNCTION
    ret_value RETURN NUMBER,
    PRAGMA RESTRICT_REFERENCES (
      ret_value, WNDS, WNPS, RNPS, RNDS),

  MEMBER FUNCTION
    total_value RETURN NUMBER,
    PRAGMA RESTRICT_REFERENCES (total_value, WNDS, WNPS)
  ) ;
```

```
CREATE TYPE stock_info_t AS OBJECT (
  stockno    NUMBER,
  cost       NUMBER,
  tax_code   NUMBER
  ) ;


CREATE OR REPLACE TYPE BODY purchase_order_t AS
  MEMBER FUNCTION total_value RETURN NUMBER IS
    i          INTEGER;
    stock      stock_info_t;
    line_item  line_item_t;
    total      NUMBER := 0;
    cost       NUMBER;

  BEGIN
    FOR i IN 1..SELF.line_item_list.COUNT  LOOP

      line_item := SELF.line_item_list(i);
      SELECT DEREF(line_item.stockref) INTO stock FROM DUAL ;

      total := total + line_item.quantity * stock.cost ;

      END LOOP;
    RETURN total;
  END;

  MAP MEMBER FUNCTION ret_value RETURN NUMBER IS
  BEGIN
    RETURN pono;
```

```
  END;
END;


CREATE OR REPLACE TYPE BODY customer_info_t AS

  ORDER MEMBER FUNCTION
  cust_order (x IN customer_info_t) RETURN INTEGER IS
  BEGIN
    RETURN custno - x.custno;
  END;

END;

CREATE TABLE customer_tab OF customer_info_t
 (custno PRIMARY KEY);

CREATE TABLE stock_tab OF stock_info_t
 (stockno PRIMARY KEY) ;

CREATE TABLE purchase_tab OF purchase_order_t (
  PRIMARY KEY (pono),
  SCOPE FOR (custref) IS customer_tab
  )
  NESTED TABLE line_item_list STORE AS po_line_tab ;

ALTER TABLE po_line_tab
  ADD (SCOPE FOR (stockref) IS stock_tab) ;

CREATE TYPE line_item_list_t AS TABLE OF line_item_t ;
```

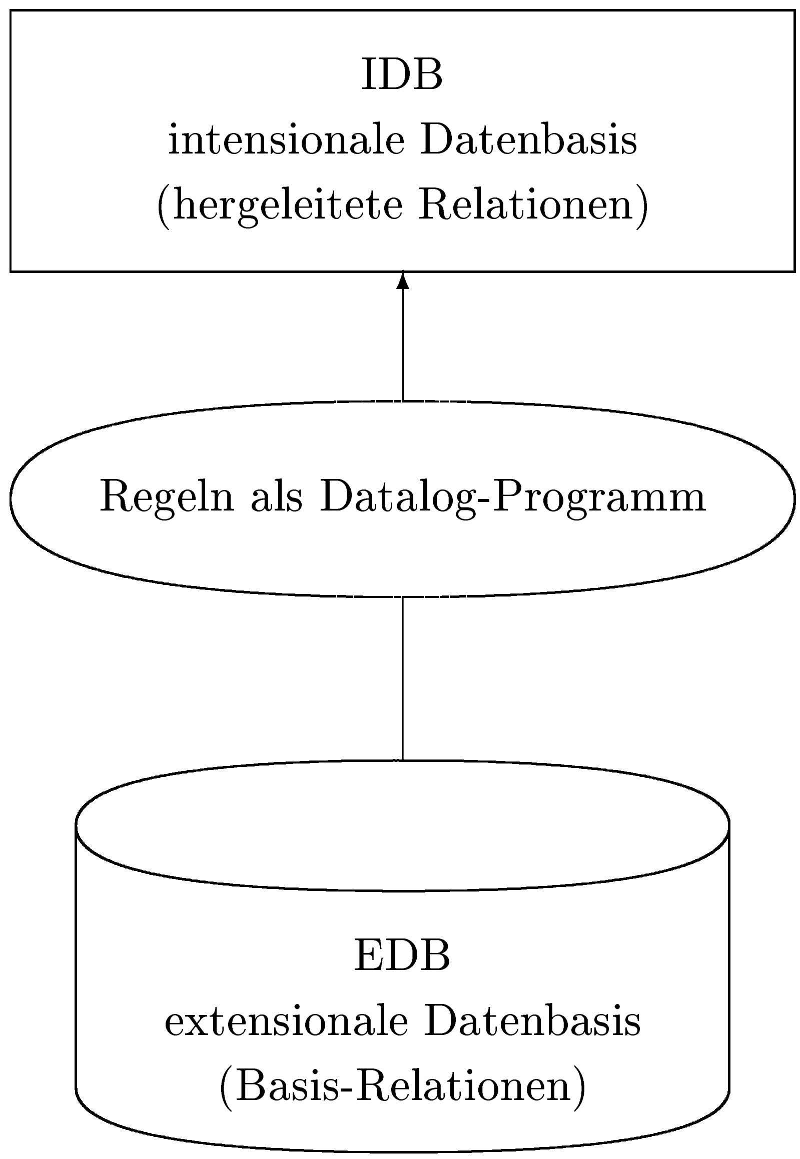
```
ALTER TABLE po_line_tab
  STORAGE (NEXT 5K PCTINCREASE 5 MINEXTENTS 1 MAXEXTENTS 20) ;

CREATE INDEX po_nested_in
  ON         po_line_tab (NESTED_TABLE_ID) ;


CREATE UNIQUE INDEX po_nested
  ON                po_line_tab (NESTED_TABLE_ID, lineitemno) ;


INSERT INTO stock_tab VALUES(1004, 6750.00, 2);
INSERT INTO stock_tab VALUES(1011, 4500.23, 2);
INSERT INTO stock_tab VALUES(1534, 2234.00, 2);
INSERT INTO stock_tab VALUES(1535, 3456.23, 2);

INSERT INTO customer_tab
  VALUES (
    1, 'Jean Nance',
    address_t('2 Avocet Drive', 'Redwood Shores', 'CA', '95054'),
    phone_list_t('415-555-1212')
    ) ;

INSERT INTO customer_tab
  VALUES (
    2, 'John Nike',
    address_t('323 College Drive', 'Edison', 'NJ', '08820'),
    phone_list_t('609-555-1212','201-555-1212')
    ) ;
```

```
INSERT INTO purchase_tab
  SELECT  1001, REF(C),
          SYSDATE,'10-MAY-1998',
          line_item_list_t(),
          NULL
   FROM   customer_tab C
   WHERE  C.custno = 1 ;


INSERT INTO THE (
  SELECT  P.line_item_list
   FROM   purchase_tab P
   WHERE  P.pono = 1001
  )
  SELECT  01, REF(S), 12, 0
   FROM   stock_tab S
   WHERE  S.stockno = 1534;


INSERT INTO purchase_tab
  SELECT  2001, REF(C),
          SYSDATE,'20-MAY-1998',
          line_item_list_t(),
          address_t('55 Madison Ave','Madison','WI','53715')
   FROM   customer_tab C
   WHERE  C.custno = 2;

INSERT INTO THE (
  SELECT  P.line_item_list
```

```
   FROM   purchase_tab P
   WHERE  P.pono = 1001
  )
  SELECT  02, REF(S), 10, 10
   FROM   stock_tab S
   WHERE  S.stockno = 1535;

INSERT INTO THE (
  SELECT  P.line_item_list
   FROM   purchase_tab P
   WHERE  P.pono = 2001
  )
  SELECT  10, REF(S), 1, 0
   FROM   stock_tab S
   WHERE  S.stockno = 1004;

INSERT INTO THE (
  SELECT  P.line_item_list
   FROM   purchase_tab P
   WHERE  P.pono = 2001
  )
  VALUES( line_item_t(11, NULL, 2, 1) ) ;

UPDATE THE (
  SELECT  P.line_item_list
   FROM   purchase_tab P
   WHERE  P.pono = 2001
  ) plist
  SET plist.stockref =
   (SELECT REF(S)
```

```
     FROM  stock_tab S
     WHERE S.stockno = 1011
     )
  WHERE plist.lineitemno = 11 ;

SELECT  p.pono
 FROM   purchase_tab p
 ORDER BY VALUE(p);


SELECT  DEREF(p.custref), p.shiptoaddr, p.pono,
        p.orderdate, line_item_list

 FROM   purchase_tab p
 WHERE  p.pono = 1001 ;


SELECT   p.pono, p.total_value()
FROM    purchase_tab p ;

SELECT   po.pono, po.custref.custno,
         CURSOR (
           SELECT  sum(L.stockref.cost*L.quantity)
            FROM   TABLE (po.line_item_list) L
           )
FROM    purchase_tab po ;

DELETE
 FROM   purchase_order
 WHERE  pono = 1001 ;
```

Grundkonzepte einer deduktiven Datenbank



Terminologie

- Die *extensionale Datenbasis (EDB)*, die manchmal auch *Faktenbasis* genannt wird. Die EDB besteht aus einer Menge von Relationen(ausprägungen) und entspricht einer „ganz normalen" relationalen Datenbasis.
- Die *Deduktionskomponente*, die aus einer Menge von (Herleitungs-)*Regeln* besteht. Die Regelsprache heißt *Datalog* – abgeleitet von dem Wort *Data* und dem Namen der Logikprogrammiersprache *Prolog*.
- Die *intensionale Datenbasis (IDB)*, die aus einer Menge von hergeleiteten Relationen(ausprägungen) besteht. Die IDB wird durch Auswertung des Datalog-Programms aus der EDB generiert.

$$\text{sokLV}(T, S) :\!-\ \text{vorlesungen}(V, T, S, P),\ \text{professoren}(P, \text{“Sokrates”}, R, Z),\ >\!(S, 2).$$

$$\begin{aligned}\{[t, s] \mid\ & \exists v, p([v, t, s, p] \in \text{Vorlesungen} \wedge \\ & \exists n, r, z([p, n, r, z] \in \text{Professoren} \wedge \\ & n = \text{“Sokrates”} \wedge s > 2))\}\end{aligned}$$

atomare Formeln **oder** *Literale*

$$q(A_1, \ldots, A_m).$$

$$\text{professoren}(S, \text{“Sokrates”}, R, Z).$$

Eine Dataog-Regel

$$p(X_1, \ldots, X_m) :- q_1(A_{i_1}, \ldots, A_{i_{m_1}}), \ldots, q_n(A_{i_n}, \ldots, A_{i_{m_n}}).$$

- Jedes $q_j(\ldots)$ ist eine atomare Formel. Die q_j werden oft als *Subgoals* bezeichnet.
- $X_1, \ldots, X_m$ sind Variablen, die mindestens einmal auch auf der rechten Seite des Zeichens :– vorkommen müssen.

$$p(\ldots) \vee \neg q_1(\ldots) \vee \ldots \vee \neg q_n(\ldots)$$

- Die Prädikate beginnen mit einem Kleinbuchstaben.
- Die zugehörigen Relationen – seien es EDB- oder IDB-Relationen – werden mit gleichem Namen, aber mit einem Großbuchstaben beginnend, bezeichnet.

Beispiel Datalog-Programm

- Zur Bestimmung von (thematisch) verwandten Vorlesungspaaren

$$
\begin{aligned}
\text{geschwisterVorl}(N1, N2) \quad &:- \quad \text{voraussetzen}(V, N1),\\
& \text{voraussetzen}(V, N2), N1 < N2.
\end{aligned}
$$

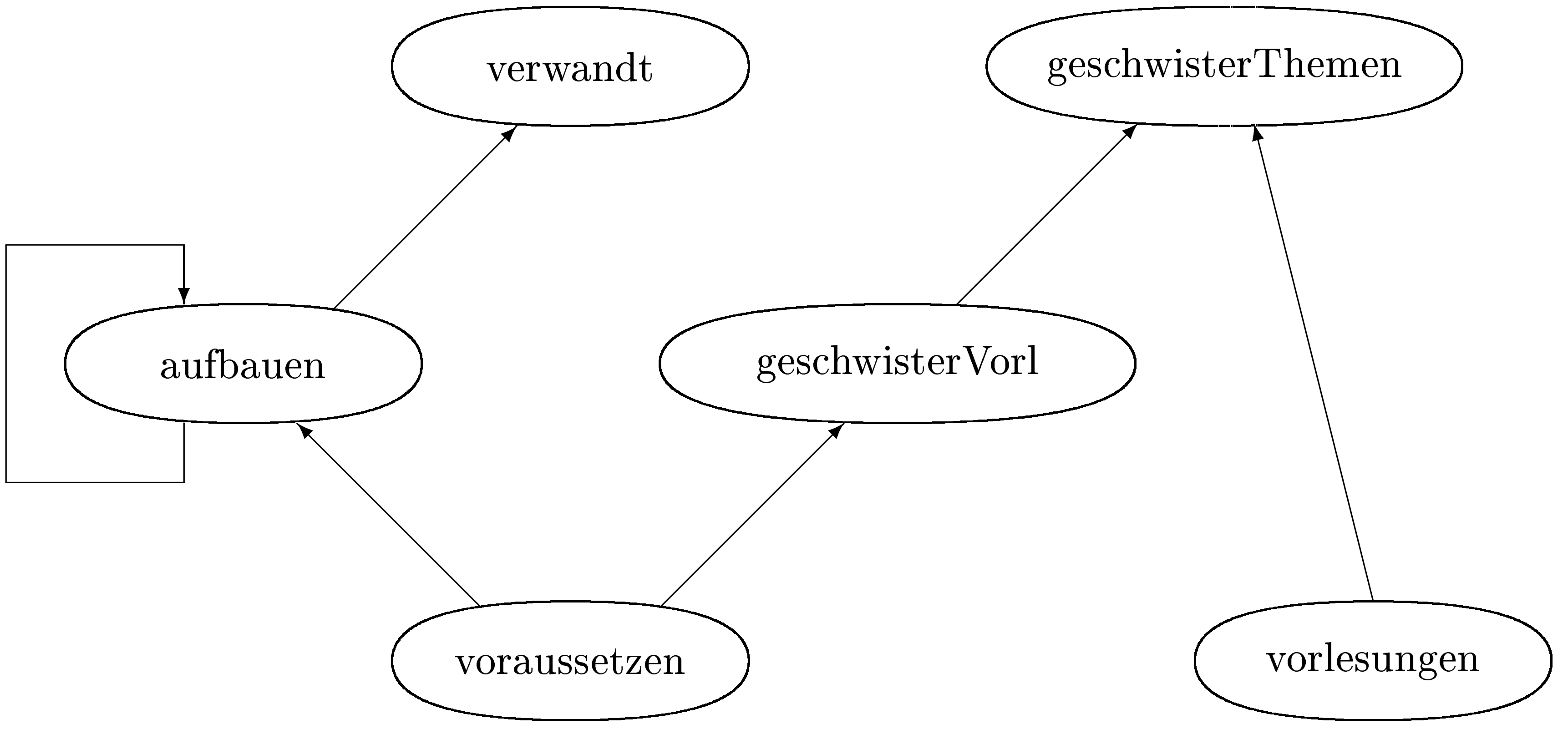
$$
\begin{aligned}
\text{geschwisterThemen}(T1, T2) \quad &:- \quad \text{geschwisterVorl}(N1, N2),\\
& \text{vorlesungen}(N1, T1, S1, R1),\\
& \text{vorlesungen}(N2, T2, S2, R2).
\end{aligned}
$$

$$
\begin{aligned}
\text{aufbauen}(V, N) \quad &:- \quad \text{voraussetzen}(V, N).\\
\text{aufbauen}(V, N) \quad &:- \quad \text{aufbauen}(V, M), \text{voraussetzen}(M, N).
\end{aligned}
$$

$$
\begin{aligned}
\text{verwandt}(N, M) \quad &:- \quad \text{aufbauen}(N, M).\\
\text{verwandt}(N, M) \quad &:- \quad \text{aufbauen}(M, N).\\
\text{verwandt}(N, M) \quad &:- \quad \text{aufbauen}(V, N), \text{aufbauen}(V, M).
\end{aligned}
$$

- Voraussetzen: {[Vorgänger, Nachfolger]}
- Vorlesungen: {[VorlNr, Titel, SWS, gelesenVon]}

Abhängigkeitsgraph



- Ein Datalog-Programm ist rekursiv, wenn der Abhängigkeitsgraph einen (oder mehrere) Zyklen hat
- Unser Beispielprogramm ist rekursiv wegen aufbauen $\rightarrow$ aufbauen

Sicherheit von Datalog-Regeln

- unsichere Regeln, wie z.B.

$$\text{ungleich}(X,Y) \quad :- \quad X \neq Y.$$

$$\text{aufbauend}(V,N) \quad :- \quad \text{vorlesungen}(V, \text{“Grundzüge“}, S, R).$$

  haben unendliche Ergebnisse

- Eine Datalog-Regel ist sicher, wenn alle Variablen im Kopf beschränkt (range restricted) sind. Dies ist für eine Variable X dann der Fall, wenn:

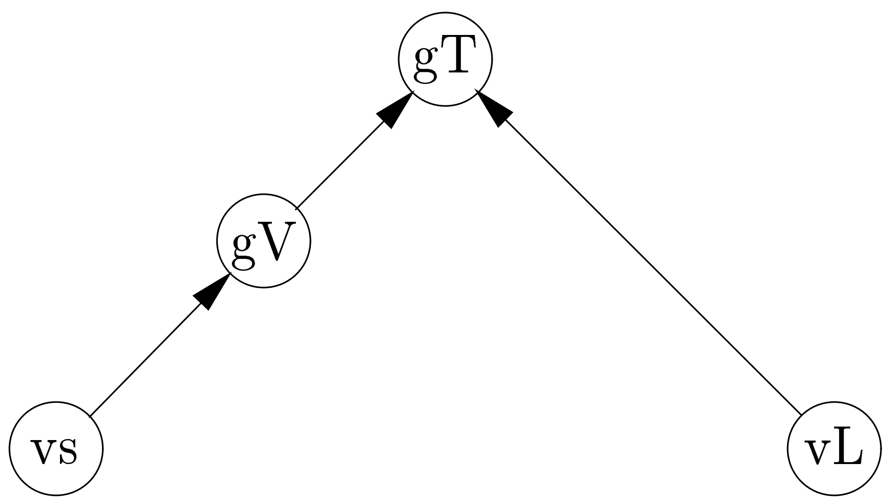
* die Variable im Rumpf der Regel in mindestens einem normalen Prädikat – also nicht nur in eingebauten Vergleichsprädikaten – vorkommt oder

* ein Prädikat der Form $X = c$ mit einer Konstanten c im Rumpf der Regel existiert oder

* ein Prädikat der Form $X = Y$ im Rumpf vorkommt, und man schon nachgewiesen hat, daß Y eingeschränkt ist.

Ein zyklenfreier Abhängigkeitsgraph

$$\begin{aligned} gV(N1, N2) &\coloneq vs(V, N1), vs(V, N2), N1 < N2. \\ gT(T1, T2) &\coloneq gV(N1, N2), vL(N1, T1, S1, R1), vL(N2, T2, S2, R2). \end{aligned}$$



- Topologische Sortierung vs, gV, vL, gT

Auswertung nicht-rekursiver Datalog-Programme

1 Für jede Regel mit dem Kopf $p(\ldots)$, also

$$p(\ldots) :- q_1(\ldots), \ldots, q_n(\ldots).$$

bilde eine Relation, in der alle im Körper der Regel vorkommenden Variablen als Attribute vorkommen. Diese Relation wird im wesentlichen durch einen natürlichen Verbund der Relationen $Q_1, \ldots, Q_n$, die den Relationen der Prädikate $q_1, \ldots, q_n$ entsprechen, gebildet. Man beachte, daß diese Relationen $Q_1, \ldots, Q_n$ wegen der Einhaltung der topologischen Sortierung bereits ausgewertet (materialisiert) sind.

2. Da das Prädikat p durch mehrere Regeln definiert sein kann, werden in diesem zweiten Schritt die Relationen aus Schritt 1. vereinigt. Hierzu muß man aber vorher noch auf die im Kopf der Regeln vorkommenden Attribute projizieren. Wir nehmen an, daß alle Köpfe der Regeln für p dieselben Attributnamen an derselben Stelle verwenden – durch Umformung der Regeln kann man dies immer erreichen.

Auswertung von Geschwister Vorlesungen und Geschwister Themen

$$\sigma_{N1<N2}(\,Vs1(V,N1) \bowtie Vs2(V,N2))$$

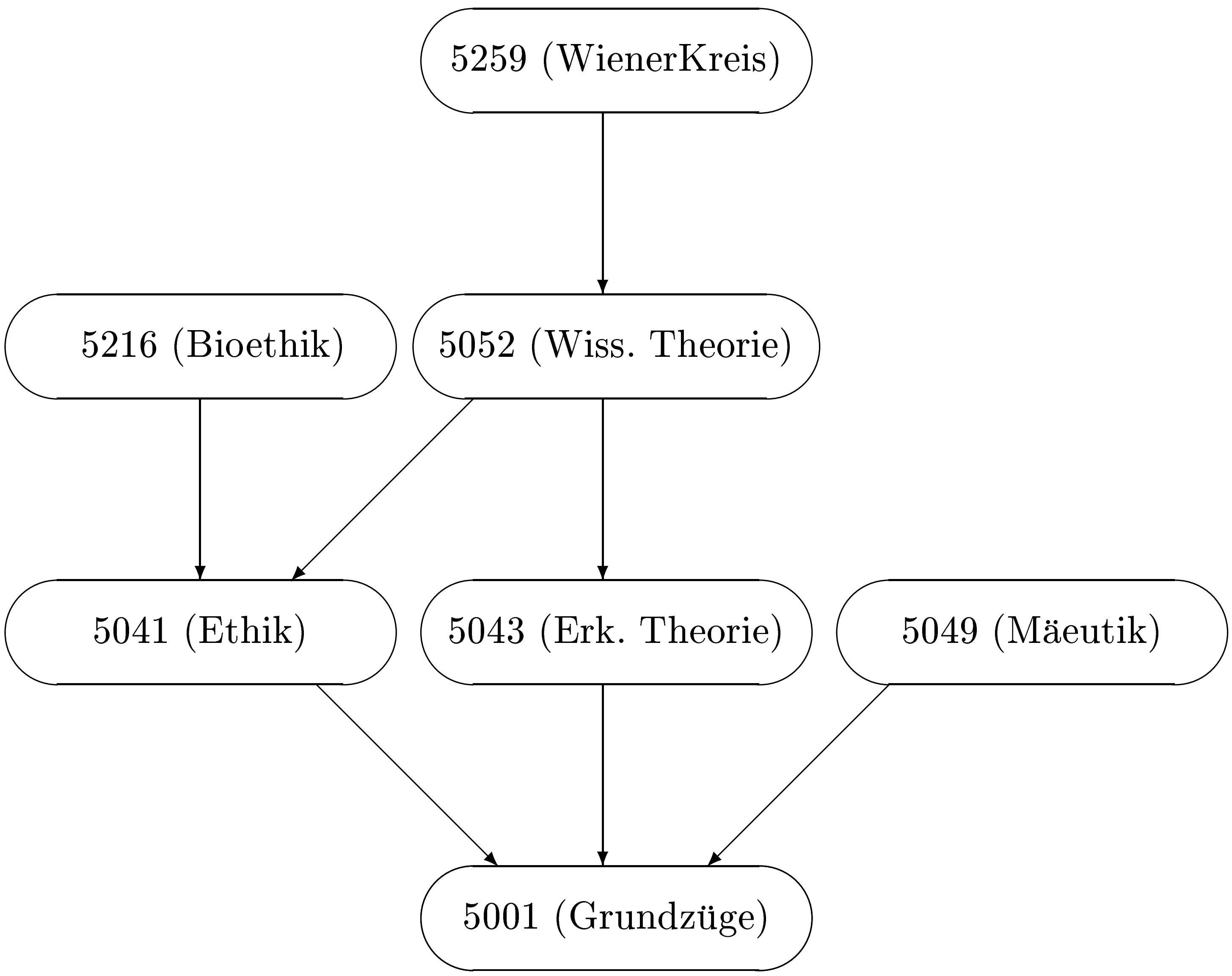
$$Vs1(V,N1) := \rho_{V\leftarrow\$1}(\rho_{N1\leftarrow\$2}(\rho_{Vs1}(\text{Voraussetzen})))$$

- das Tupel $[v, n1]$ ist in der Relation *Voraussetzen* enthalten,
- das Tupel $[v, n2]$ ist in der Relation *Voraussetzen* enthalten und
- $n1 < n2$.

$$GV(N1,N2) := \Pi_{N1,N2}(\sigma_{N1<N2}(\,Vs1(V,N1) \bowtie Vs2(V,N2)))$$

$$GT(T1,T2) := \Pi_{T1,T2}(GV(N1,N2) \bowtie VL1(N1,T1,S1,R1) \bowtie VL2(N2,T2,S2,R2))$$

Veranschaulichung der EDB-Relation *Voraussetzen*



Ausprägung der Relationen *GeschwisterVorl* und *GeschwisterThemen*

GeschwisterVorl	
$N1$	$N2$
5041	5043
5043	5049
5041	5049
5052	5216

GeschwisterThemen	
$T1$	$T2$
Ethik	Erkenntnistheorie
Erkenntnistheorie	Mäeutik
Ethik	Mäeutik
Wissenschaftstheorie	Bioethik

Auswertungs-Algorithmus (abstrakt)

$$p(X_1, \ldots, X_m) \quad :- \quad q_1(A_{i_1}, \ldots, , A_{i_{m_1}}), \ldots, q_n(A_{i_n}, \ldots, A_{i_{m_n}}).$$

$$Q_i : \{[\$1, \ldots, \$m_i]\}$$

$$q_i(A_{i_1}, \ldots, A_{i_{m_i}})$$

$$E_i := \Pi_{V_i}(\sigma_{F_i}(Q_i))$$

- Falls in $q_i(\ldots, c, \ldots)$ eine Konstante c an j-ter Stelle vorkommt, füge die Bedingung
$$\$j = c$$
hinzu.

- Falls eine Variable X mehrfach an Positionen k und l in $q_i(\ldots, X, \ldots, X, \ldots)$ vorkommt, füge für jedes solche Paar die Bedingung
$$\$k = \$l$$
hinzu.

Für eine Variable Y, die nicht in den normalen Prädikaten vorkommt, gibt es zwei Möglichkeiten:

- Sie kommt nur als Prädikat
$$Y = c$$
für eine Konstante c vor. Dann wird eine einstellige Relation mit einem Tupel
$$Q_Y := \{[c]\}$$
gebildet.

- Sie kommt als Prädikat
$$X = Y$$
vor, und X kommt in einem normalen Prädikat $q_i(\ldots, X, \ldots)$ an k-ter Stelle vor. In diesem Fall setze
$$Q_Y := \rho_{Y \leftarrow \$k}(\Pi_{\$k}(Q_i))$$
.

Nun bilde man den Algebra-Ausdruck

$$E := E_1 \bowtie \ldots \bowtie E_n$$

und wende abschließend

$$\sigma_F(E)$$

an, wobei F aus der konjunktiven Verknüpfung der Vergleichsprädikate

$$X \phi Y$$

Beispiel: nahe verwandte Vorlesungen

Wir wollen diese Vorgehensweise nochmals am Beispiel demonstrieren:

$$
\begin{array}{llll}
(r_1) & nvV(N1, N2) & :- & gV(N1, N2). \\
(r_2) & nvV(N1, N2) & :- & gV(M1, M2), vs(M1, N1), vs(M2, N2).
\end{array}
$$

Dieses Beispielprogramm baut auf dem Prädikat gV auf und ermittelt nahe verwandte Vorlesungen, die einen gemeinsamen Vorgänger erster oder zweiter Stufe haben. Für die erste Regel erhält man folgenden Algebra-Ausdruck:

$$E_{r_1} := \Pi_{N1,N2}(\sigma_{\text{true}}(GV(N1, N2)))$$

Für die zweite Regel ergibt sich gemäß dem oben skizzierten Algorithmus:

$$E_{r_2} := \Pi_{N1,N2}(GV(M1, M2) \bowtie Vs1(M1, N1) \bowtie Vs2(M2, N2)).$$

Daraus ergibt sich dann durch die Vereinigung

$$NvV := E_{r_1} \cup E_{r_2}$$

die Relation NvV, die durch das Prädikat nvV definiert ist. Die Leser mögen bitte die Auswertung dieses Relationenalgebra-Ausdrucks an unserer Beispiel-Datenbasis durchführen.

$$a(V,N) \quad :- \quad vs(V,N).$$
$$a(V,N) \quad :- \quad a(V,M), vs(M,N).$$

Aufbauen	
V	N
5001	5041
5001	5043
5001	5049
5041	5216
5041	5052
5043	5052
5052	5259
5001	5216
5001	5052
5001	5259
5041	5259
5043	5259

Betrachten wir das Tupel $[5001, 5052]$ aus der Relation *Aufbauen*. Dieses Tupel kann wie folgt hergeleitet werden:

1. $a(5001, 5043)$ folgt aus der ersten Regel, da $vs(5001, 5043)$ gilt.
2. $a(5001, 5052)$ folgt aus der zweiten Regel, da
   (a) $a(5001, 5043)$ nach Schritt 1. gilt und
   (b) $vs(5043, 5052)$ gemäß der EDB-Relation *Voraussetzen* gilt.

Naive Auswertung durch Iteration

$$A(V,N) = Vs(V,N) \cup \Pi_{V,N}(A(V,M) \bowtie Vs(M,N))$$

$A := \{\};$ /* Initialisierung auf die leere Menge */
repeat
  $A' := A;$
  $A := Vs(V,N);$ /* erste Regel */
  $A := A \cup \Pi_{V,N}(A'(V,M) \bowtie Vs(M,N));$ /* zweite Regel */
until $A' = A$
output A;

1. Im ersten Durchlauf werden nur die 7 Tupel aus *Voraussetzen* nach A „übertragen", da der Join leer ist (das linke Argument A' des Joins wurde zur leeren Relation $\{\}$ initialisiert).
2. Im zweiten Schritt kommen zusätzlich die Tupel $[5001, 5216]$, $[5001, 5052]$, $[5041, 5259]$ und $[5043, 5259]$ hinzu.
3. Jetzt wird nur noch das eine Tupel $[5001, 5259]$ neu generiert.
4. In diesem Schritt kommt kein neues Tupel mehr hinzu, so daß die Abbruchbedingung $A' = A$ erfüllt ist.

(Naive) Auswertung der rekursiven Regel *aufbauen*

Schritt	A
1	[5001, 5041], [5001, 5043], [5001, 5049], [5041, 5216], [5041, 5052], [5043, 5052], [5052, 5259]
2	[5001, 5041], [5001, 5043], [5001, 5049], [5041, 5216], [5041, 5052], [5043, 5052], [5052, 5259] [5001, 5216], [5001, 5052], [5041, 5259], [5043, 5259],
3	[5001, 5041], [5001, 5043], [5001, 5049], [5041, 5216], [5041, 5052], [5043, 5052], [5052, 5259] [5001, 5216], [5001, 5052], [5041, 5259], [5043, 5259], [5001, 5259]
4	wie in Schritt 3 (keine Veränderung, also Terminierung des Algorithmus)

Inkrementelle (semi-naive) Auswertung rekursiver Regeln

Die Schlüsselidee der *semi-naiven Auswertung* liegt in der Beobachtung, daß für die Generierung eines neuen Tupels t der rekursiv definierten IDB-Relation P eine bestimmte Regel

$$p(\ldots) :- q_1(\ldots), \ldots, q_n(\ldots).$$

für Prädikat p „verantwortlich" ist. Dann wird also im iterativen Auswertungprogramm ein Algebra-Ausdruck der Art

$$E(Q_1 \bowtie \ldots \bowtie Q_n)$$

iterativ ausgewertet. Es reicht aber aus

$$E(\Delta Q_1 \bowtie Q_2 \bowtie \ldots \bowtie Q_n) \cup E(Q_1 \bowtie \Delta Q_2 \bowtie \ldots \bowtie Q_n) \cup \ldots \cup E(Q_1 \bowtie Q_2 \bowtie \ldots \bowtie \Delta Q_n)$$

auszuwerten.

$$E(\underbrace{Q_1}_{t_1} \bowtie \ldots \bowtie \underbrace{Q_{i-1}}_{t_{i-1}} \bowtie \underbrace{\Delta Q_i}_{t_i} \bowtie \underbrace{Q_{i+1}}_{t_{i+1}} \bowtie \ldots \bowtie \underbrace{Q_n}_{t_n})$$

$$t = [5001, 5259]$$

Dieses Tupel wurde aus dem folgenden Join gebildet:

$$\underbrace{[5001, 5052]}_{t_1 \in A} \bowtie \underbrace{[5052, 5259]}_{t_2 \in Vs}$$

Programm zur semi-naiven Auswertung von *aufbauen*

$A := \{\}; \Delta Vs := \{\};$
$\Delta A := Vs(V,N);$ /* erste Regel */
$\Delta A := \Delta A \cup \Pi_{V,N}(A(V,M) \bowtie Vs(M,N));$ /* zweite Regel */
$A := \Delta A;$
repeat
    $\Delta A' := \Delta A;$
    $\Delta A := \Delta Vs(V,N);$ /* erste Regel, liefert $\emptyset$ */
    $\Delta A := \Delta A \cup$ /* zweite Regel */;
        $\Pi_{V,N}(\Delta A'(V,M) \bowtie Vs(M,N)) \cup$
        $\Pi_{V,N}(A(V,M) \bowtie \Delta Vs(M,N));$
    $\Delta A := \Delta A - A;$ /* entferne „neue“ Tupel, die schon vorhanden waren */
    $A := A \cup \Delta A;$
until $\Delta A = \emptyset;$

$$\underbrace{Vs(V,N)}_{\text{erste Regel}} \cup \underbrace{\Pi_{V,N}(A(V,M) \bowtie Vs(M,N))}_{\text{zweite Regel}}$$

Illustration der semi-naiven Auswertung von $Aufbauen$

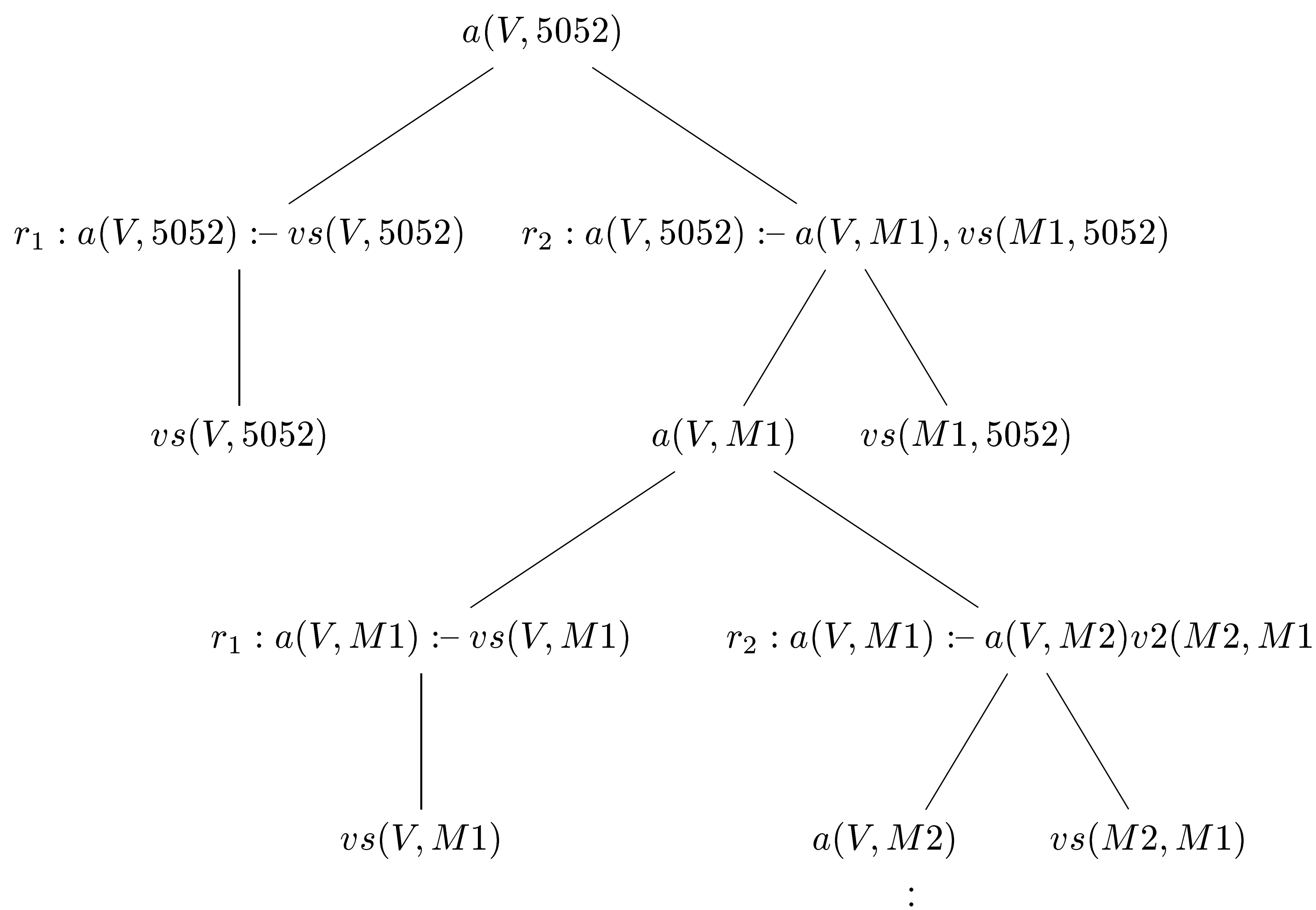
Schritt	ΔA
Initialisierung (Zeile 2. und 3.)	(sieben Tupel aus Vs) $[5001, 5042], [5001, 5043]$ $[5043, 5052], [5041, 5052]$ $[5001, 5049], [5041, 5216]$ $[5052, 5259]$
1. Iteration	(Pfade der Länge 2) $[5001, 5216], [5001, 5052]$ $[5041, 5259], [5043, 5259]$
2. Iteration	(Pfade der Länge 3) $[5001, 5259]$
3. Iteration	$\emptyset$ (Terminierung)

Bottom-Up oder Top-Down Auswertung

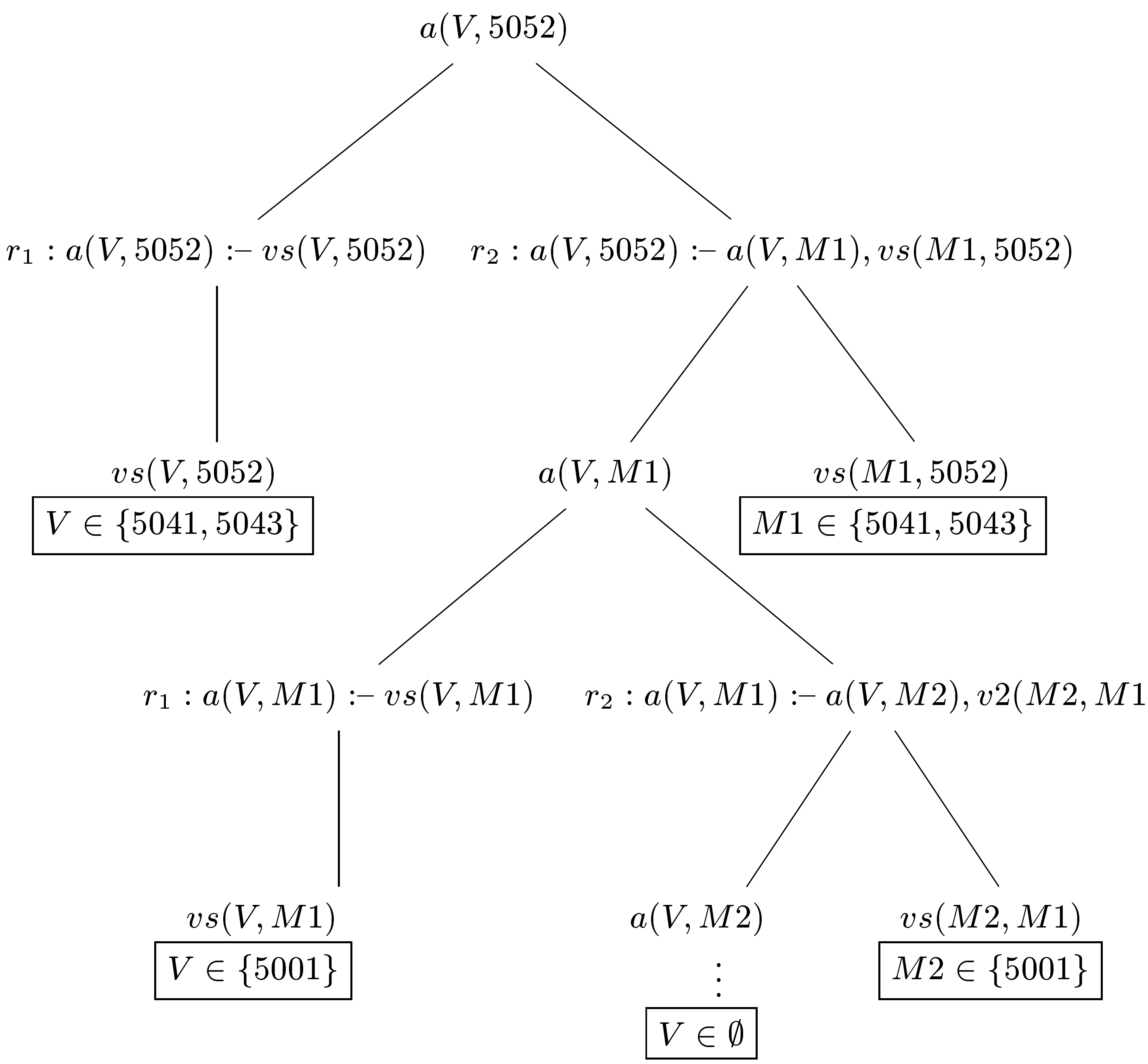
$$
\begin{array}{ll}
(r_1) & a(V,N) \coloneq vs(V,N). \\
(r_2) & a(V,N) \coloneq a(V,M), vs(M,N).
\end{array}
$$

$$
\text{query}(V) \coloneq a(V, 5052).
$$

Rule/Goal-Baum zur Top-Down Auswertung



Rule/Goal-Baum mit Auswertung



$$\text{indirektAufbauen}(V, N) \coloneq \text{aufbauen}(V, N), \neg\text{voraussetzen}(V, N).$$

Stratifizierte Datalog-Programme

Eine Regel mit einem negierten Prädikat im Rumpf, wie z.B.

$$r \equiv p(\ldots) \coloneq q_1(\ldots), \ldots, \neg q_i(\ldots), \ldots, q_n(\ldots).$$

kann nur dann sinnvoll ausgewertet werden, wenn Q_1 schon materialisiert ist. Also müssen zuerst alle Regeln mit Kopf

$$q_i(\ldots) \coloneq \ldots$$

ausgewertet sein. Also darf der Abhängigkeitsgraph keine Pfade von q_1 nach p enthalten. Wenn das für alle Regeln der Fall ist, nennt man das Datalog-Programm *stratifiziert*.

Auswertung von Regeln mit Negation

$$iA(V,N) :\!-\; a(V,N), \neg vs(V,N).$$

$$\begin{aligned} iA(V,N) &= \Pi_{V,N}(A(V,N) \bowtie \overline{Vs}(V,N)) \\ &= A(V,N) - Vs(V,N) \end{aligned}$$

$$\overline{Q}_i := \underbrace{(\mathrm{DOM} \times \ldots \times DOM)}_{k-\mathrm{mal}} - Q_i$$

Ein etwas komplexeres Beispiel

$$\begin{aligned}
\text{grundlagen}(V) &\quad :\!-\quad \text{voraussetzen}(V, N).\\
\text{spezialVorl}(V) &\quad :\!-\quad \text{vorlesungen}(V, T, S, R), \neg\text{grundlagen}(V).
\end{aligned}$$

$$\begin{aligned}
\text{Grundlagen}(V) &\quad := \quad \Pi_V(\text{Voraussetzen}(V, N))\\
\text{SpezialVorl}(V) &\quad := \quad \Pi_V(\text{Vorlesungen}(V, T, S, R) \bowtie \overline{\text{Grundlagen}}(V))
\end{aligned}$$

Hierbei ist $\overline{Grundlagen}(V)$ als $DOM -\ Grundlagen(V)$ definiert.

$$\text{SpezialVorl}(V) := \Pi_V(\text{Vorlesungen}(V, T, S, R)) - \text{Grundlagen}(V)$$

Ausdruckskraft von Datalog

- Die Sprache Datalog, eingeschränkt auf nicht-rekursive Programme aber erweitert um Negation, wird in der Literatur manchmal als $Datalog^{\neg}_{non\text{-}rec}$ bezeichnet
- Diese Sprache $Datalog^{\neg}_{non\text{-}rec}$ hat genau die gleiche Ausdruckskraft wie die relationale Algebra – und damit ist sie hinsichtlich Ausdruckskraft auch äquivalent zum relationalen Tupel- und Domänenkalkül
- Datalog mit Negation und Rekursion geht natürlich über die Ausdruckskraft der relationalen Algebra hinaus – man konnte in Datalog ja z.B. die transitive Hülle der Relation *Voraussetzen* definieren

Datalog-Formulierung der relationalen Algebra-Operatoren

Selektion

$$\sigma_{SWS>3}(\text{Vorlesungen}),$$

$$\text{query}(V,T,S,R) :\!- \text{vorlesungen}(V,T,S,R), S > 3.$$

$$\text{query}(V,S,R) :\!- \text{vorlesungen}(V, \text{“Mäeutik”}, S, R).$$

Projektion

$$\text{query}(\text{Name}, \text{Rang}) :\!- \text{professoren}(\text{PersNr}, \text{Name}, \text{Rang}, \text{Raum}).$$

Join

$$\Pi_{\text{Titel,Name}}(\text{Vorlesungen} \bowtie_{\text{gelesenVon=PersNr}} \text{Professoren})$$

$$\text{query}(T,N) :\!- \text{vorlesungen}(V,T,S,R), \text{professoren}(R,N,Rg,Ra).$$

Kreuzprodukt

$$\begin{aligned}\text{query}(V1,V2,V3,V4,P1,P2,P3,P4) \quad :\!- \quad & \text{vorlesungen}(V1,V2,V3,V4),\\ & \text{professoren}(P1,P2,P3,P4).\end{aligned}$$

$$\text{Professoren} \times \text{Vorlesungen}$$

Vereinigung

$$\Pi_{\text{PersNr,Name}}(\text{Assistenten}) \cup \Pi_{\text{PersNr,Name}}(\text{Professoren})$$

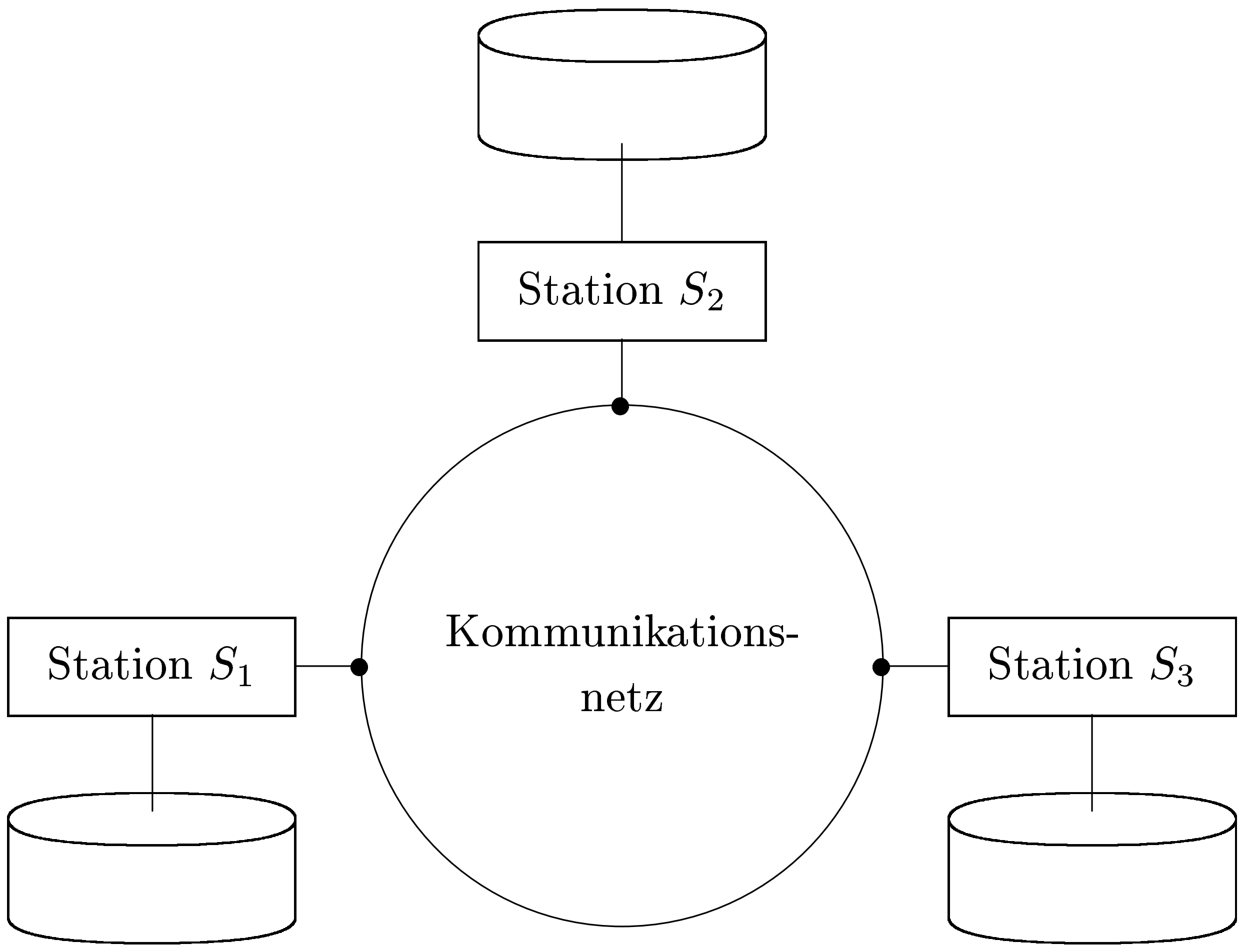
$$\begin{aligned}
\text{query}(\text{PersNr}, \text{Name}) \quad &:- \quad \text{assistenten}(\text{PersNr}, \text{Name}, \text{F}, \text{B}). \\
\text{query}(\text{PersNr}, \text{Name}) \quad &:- \quad \text{professoren}(\text{PersNr}, \text{Name}, \text{Rg}, \text{Ra}).
\end{aligned}$$

Mengendifferenz

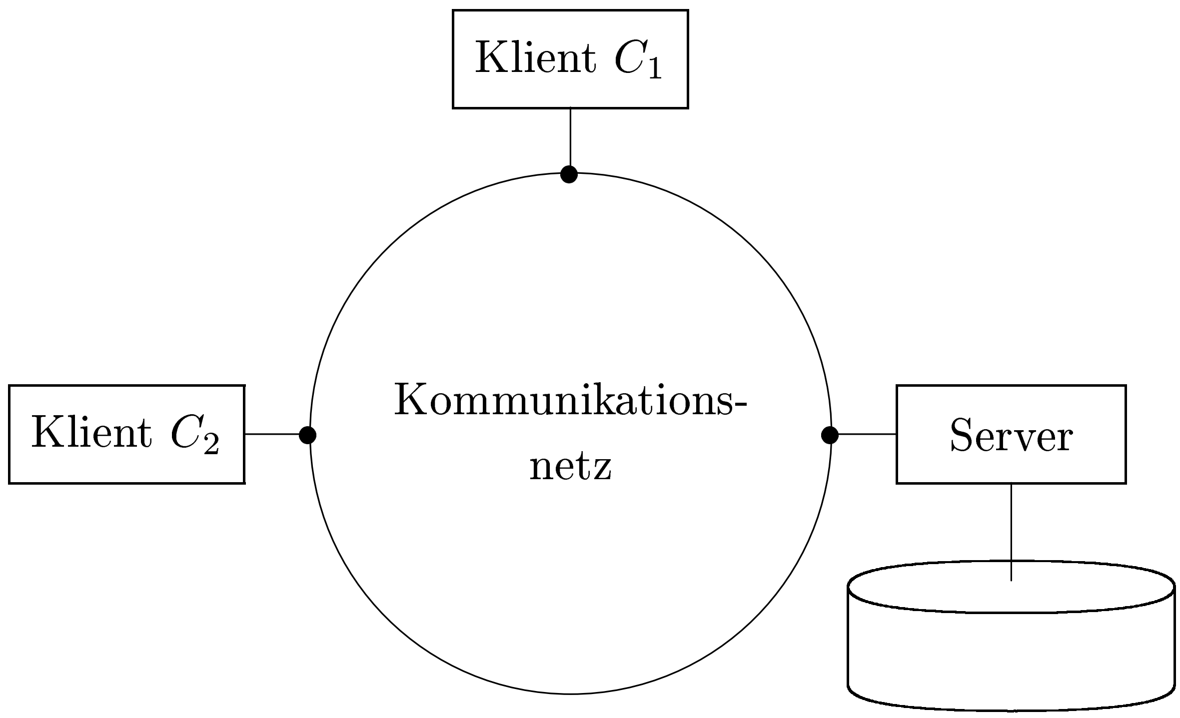
$$\Pi_{\text{VorlNr}}(\text{Vorlesungen}) - \Pi_{\text{Vorgänger}}(\text{Voraussetzen})$$

$$\begin{aligned}
\text{vorlNr}(V) \quad &:- \quad \text{vorlesungen}(V, T, S, R). \\
\text{grundlagen}(V) \quad &:- \quad \text{voraussetzen}(V, N). \\
\text{query}(V) \quad &:- \quad \text{vorlNr}(V), \neg\text{grundlagen}(V).
\end{aligned}$$

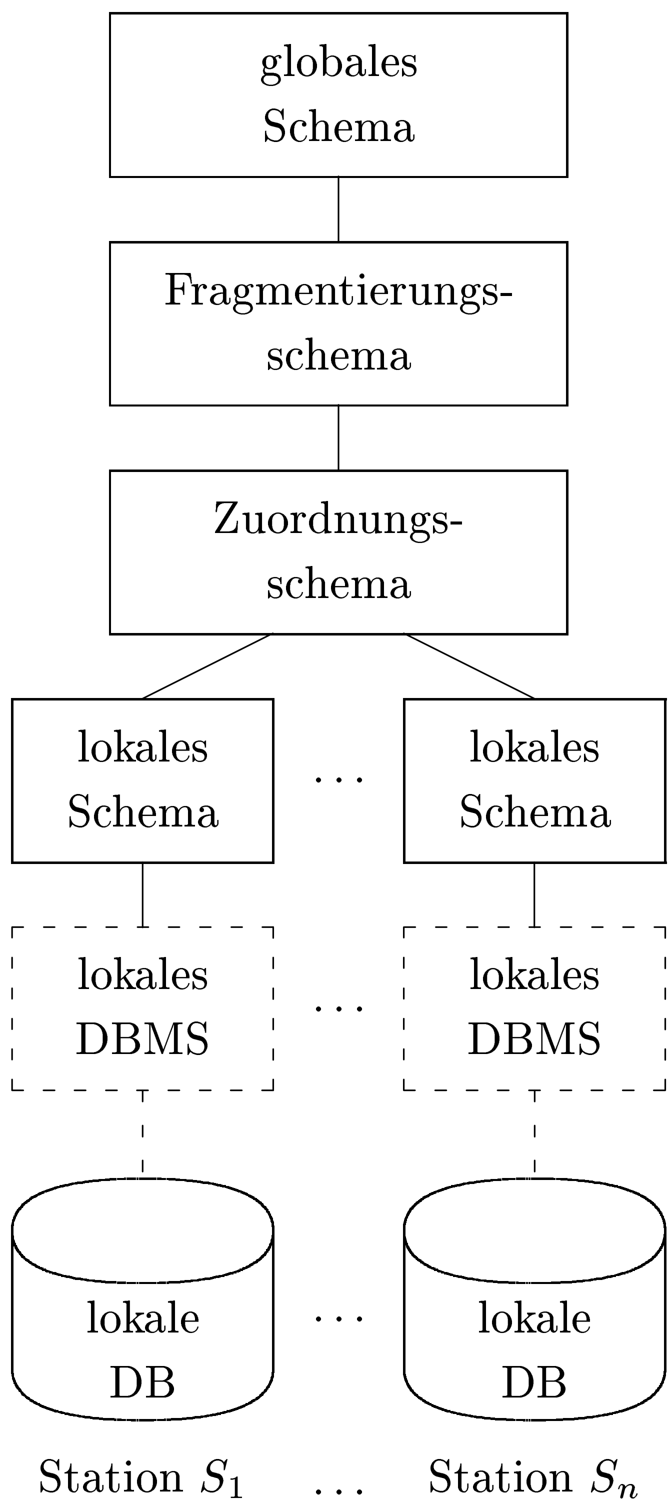
Verteilte Datenbanken



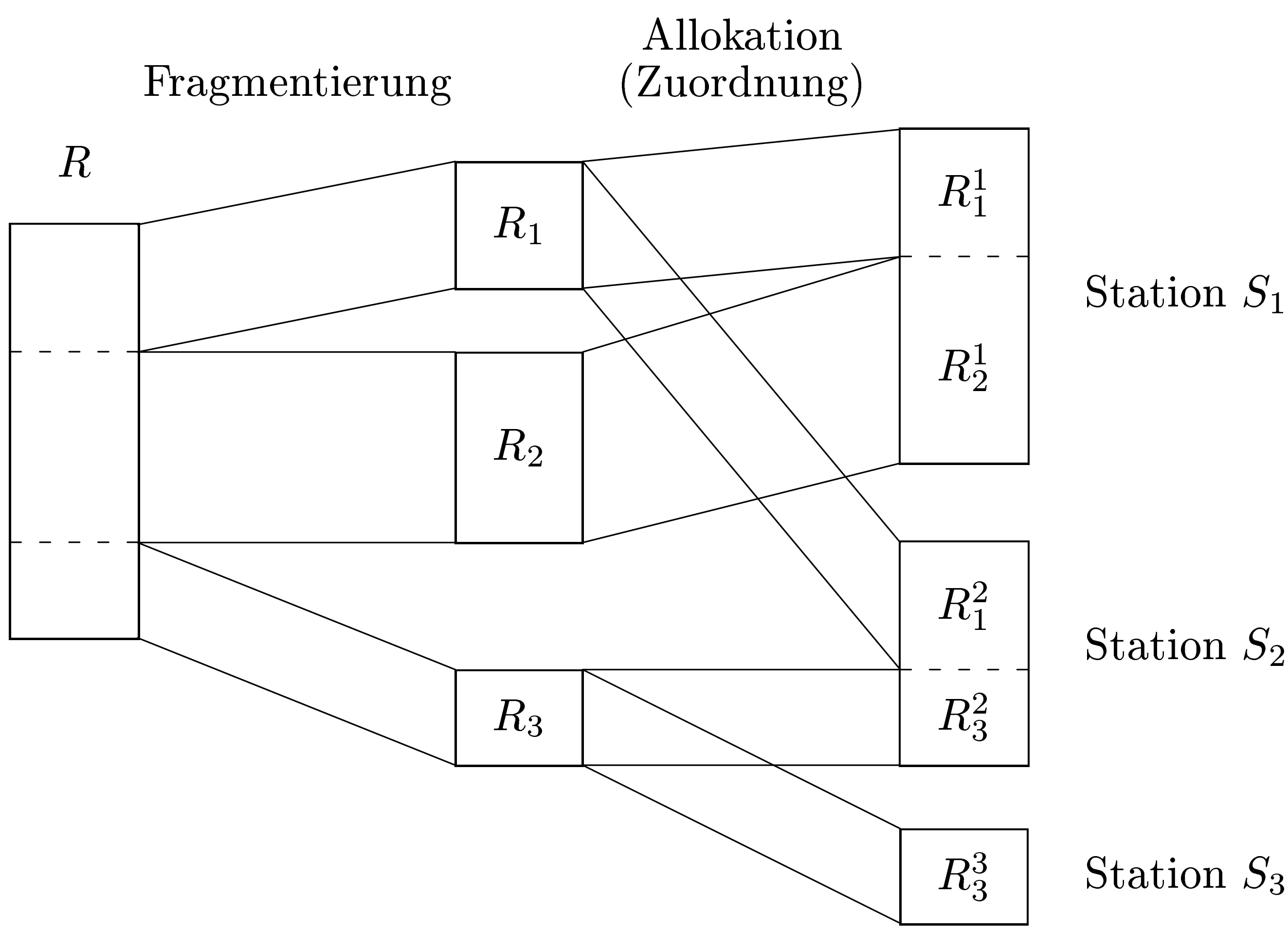
Client-Server-Architektur



Aufbau und Entwurf eines verteilten Datenbanksytems



Fragmentierung und Allokation einer Relation



Fragmentierung

- Fragmente enthalten Daten mit gleichem Zugriffsverhalten

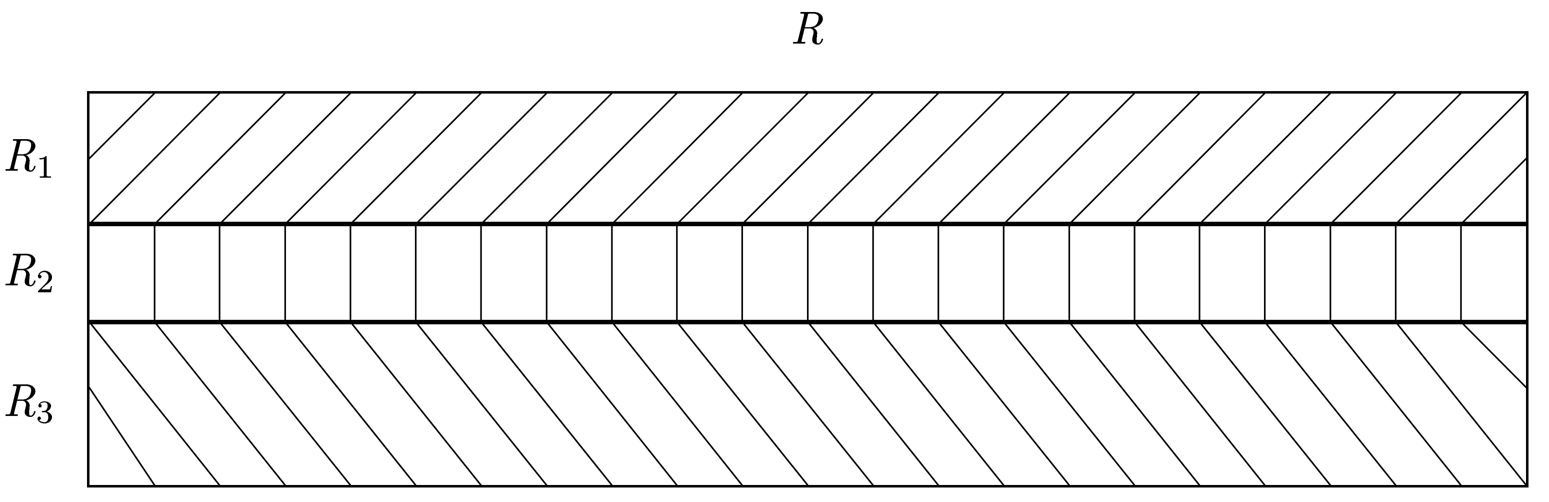
Allokation

- Fragmente werden den Stationen zugeordnet
  - mit Replikation (redundanzfrei)
  - ohne Replikation

Beispielausprägung der Relation *Professoren*

Professoren						
PersNr	Name	Rang	Raum	Fakultät	Gehalt	Steuerklasse
2125	Sokrates	C4	226	Philosophie	85000	1
2126	Russel	C4	232	Philosophie	80000	3
2127	Kopernikus	C3	310	Physik	65000	5
2133	Popper	C3	52	Philosophie	68000	1
2134	Augustinus	C3	309	Theologie	55000	5
2136	Curie	C4	36	Physik	95000	3
2137	Kant	C4	7	Philosophie	98000	1

Horizontale Fragmentierung



$$
\begin{aligned}
R_1 &:= \sigma_{p_1 \wedge p_2}(R)\\
R_2 &:= \sigma_{p_1 \wedge \neg p_2}(R)\\
R_3 &:= \sigma_{\neg p_1 \wedge p_2}(R)\\
R_4 &:= \sigma_{\neg p_1 \wedge \neg p_2}(R)
\end{aligned}
$$

Fragmentierung der Beispielrelation Professoren

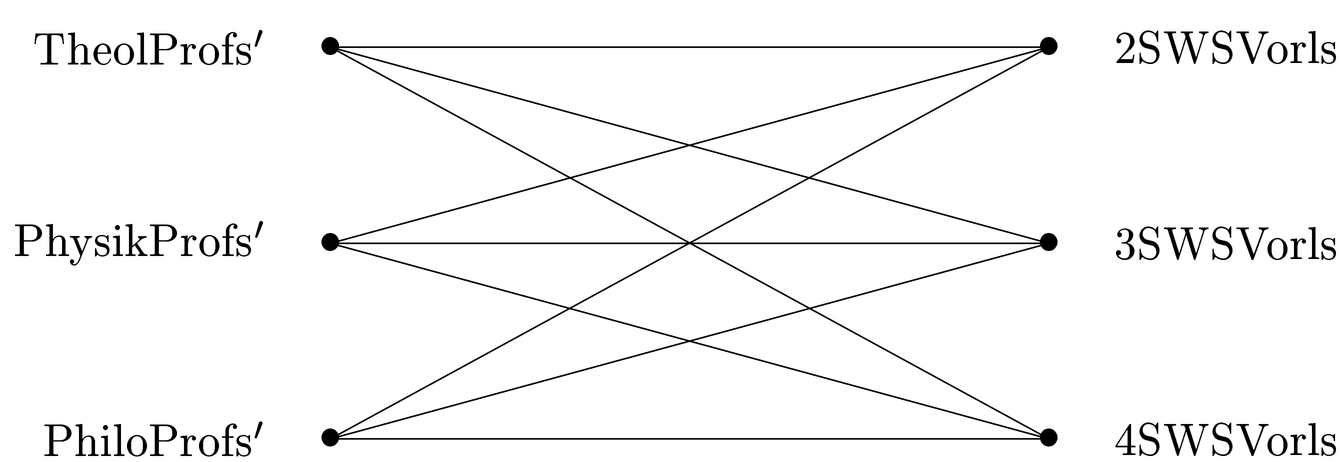
$$
\begin{aligned}
p_1 &\equiv \text{Fakultät} = \text{'Theologie'} \\
p_2 &\equiv \text{Fakultät} = \text{'Physik'} \\
p_3 &\equiv \text{Fakultät} = \text{'Philosophie'}
\end{aligned}
$$

$$
\begin{aligned}
\text{TheolProfs}' &:= \sigma_{p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professoren}) = \sigma_{p_1}(\text{Professoren}) \\
\text{PhysikProfs}' &:= \sigma_{\neg p_1 \wedge p_2 \wedge \neg p_3}(\text{Professoren}) = \sigma_{p_2}(\text{Professoren}) \\
\text{PhiloProfs}' &:= \sigma_{\neg p_1 \wedge \neg p_2 \wedge p_3}(\text{Professoren}) = \sigma_{p_3}(\text{Professoren}) \\
\text{AndereProfs}' &:= \sigma_{\neg p_1 \wedge \neg p_2 \wedge \neg p_3}(\text{Professoren})
\end{aligned}
$$

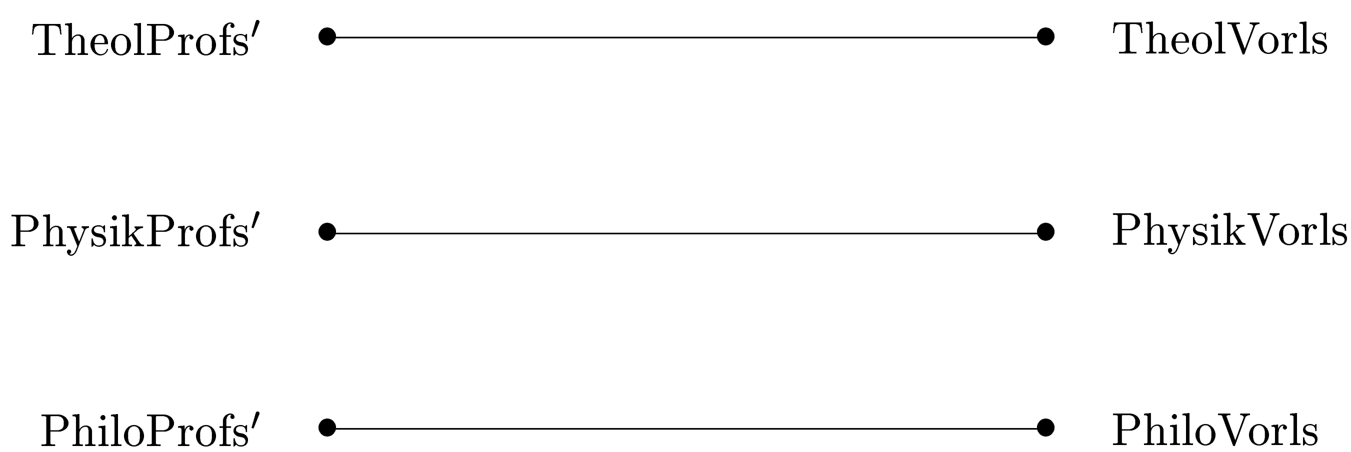
Join-Graph bei ungeeigneter Fragmentierung

$$\begin{aligned}
\text{2SWSVorls} &:= \sigma_{\text{SWS}=2}(\text{Vorlesungen}) \\
\text{3SWSVorls} &:= \sigma_{\text{SWS}=3}(\text{Vorlesungen}) \\
\text{4SWSVorls} &:= \sigma_{\text{SWS}=4}(\text{Vorlesungen})
\end{aligned}$$

select Titel, Name
from Vorlesungen, Professoren
where gelesenVon = PersNr;



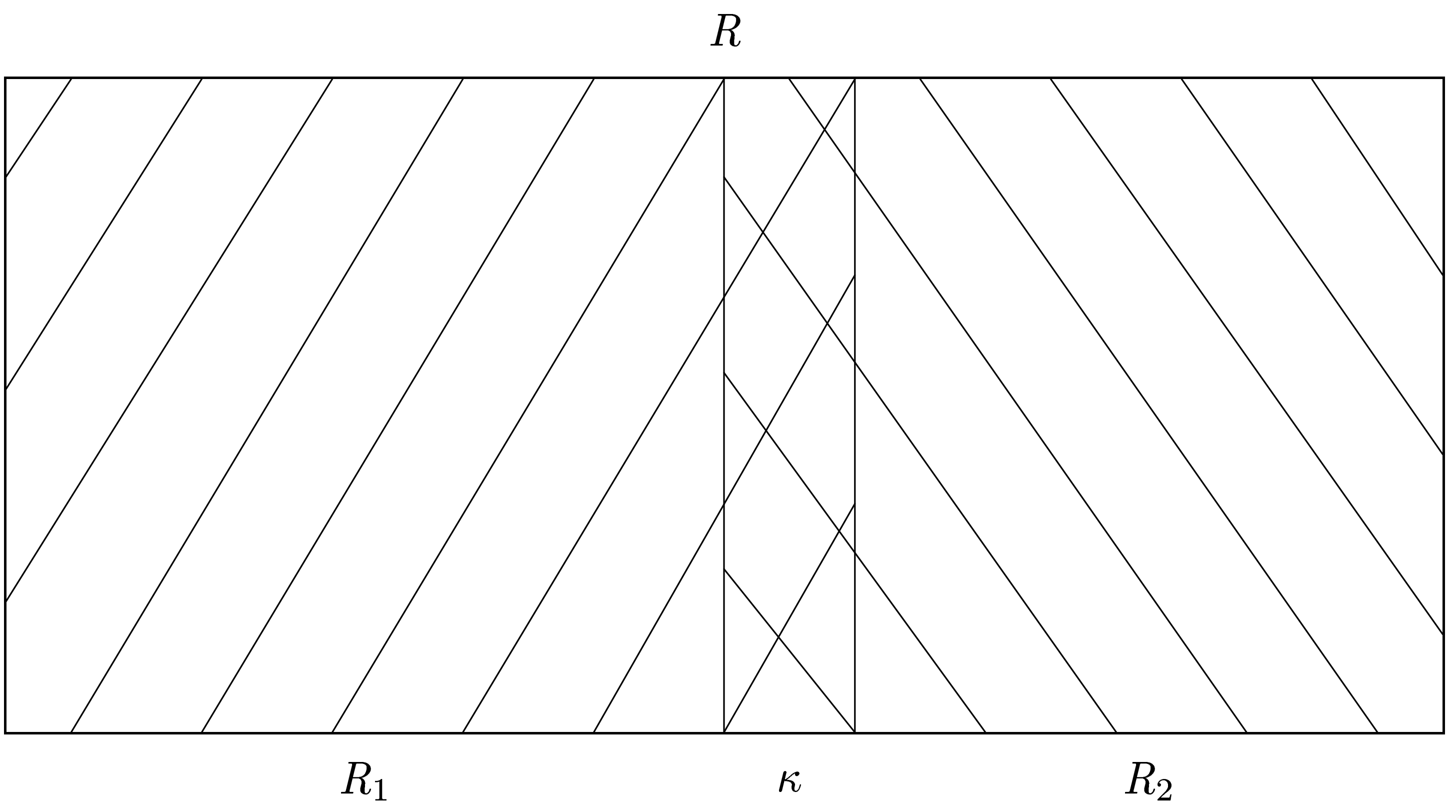
Join-Graph nach abgeleiteter Fragmentierung



$$
\begin{aligned}
\text{TheolVorls} &:= \text{Vorlesungen} \ltimes_{\text{gelesenVon}=\text{PersNr}} \text{TheolProfs}' \\
\text{PhysikVorls} &:= \text{Vorlesungen} \ltimes_{\text{gelesenVon}=\text{PersNr}} \text{PhysikProfs}' \\
\text{PhiloVorls} &:= \text{Vorlesungen} \ltimes_{\text{gelesenVon}=\text{PersNr}} \text{PhiloProfs}'
\end{aligned}
$$

$$
\begin{aligned}
\Pi_{\text{Titel, Name}}((\text{TheolProfs}' \bowtie_p \text{TheolVorls}) \\
\cup (\text{PhysikProfs}' \bowtie_p \text{PhysikVorls}) \\
\cup (\text{PhiloProfs}' \bowtie_p \text{PhiloVorls}))
\end{aligned}
$$

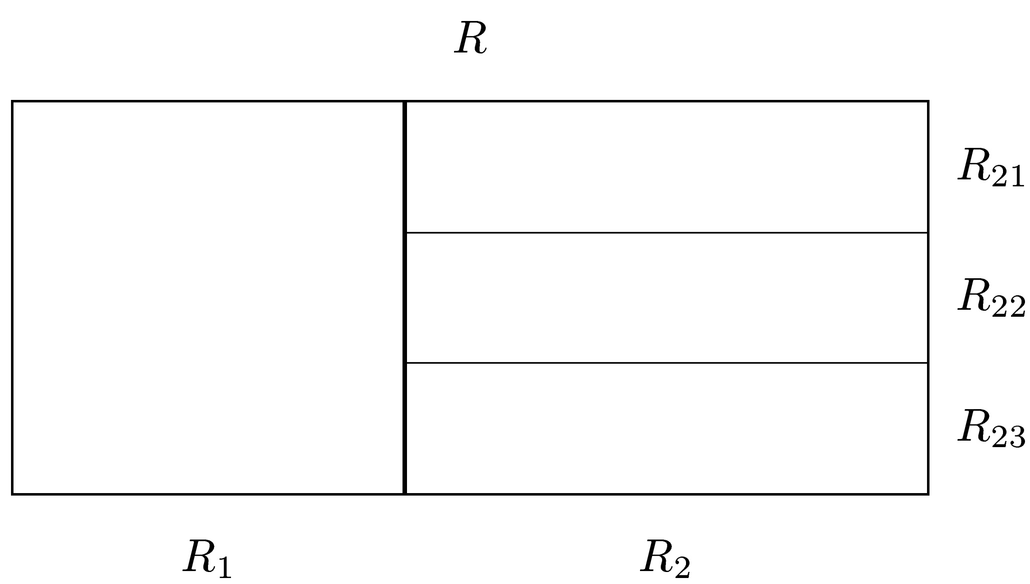
Vertikale Fragmentierung

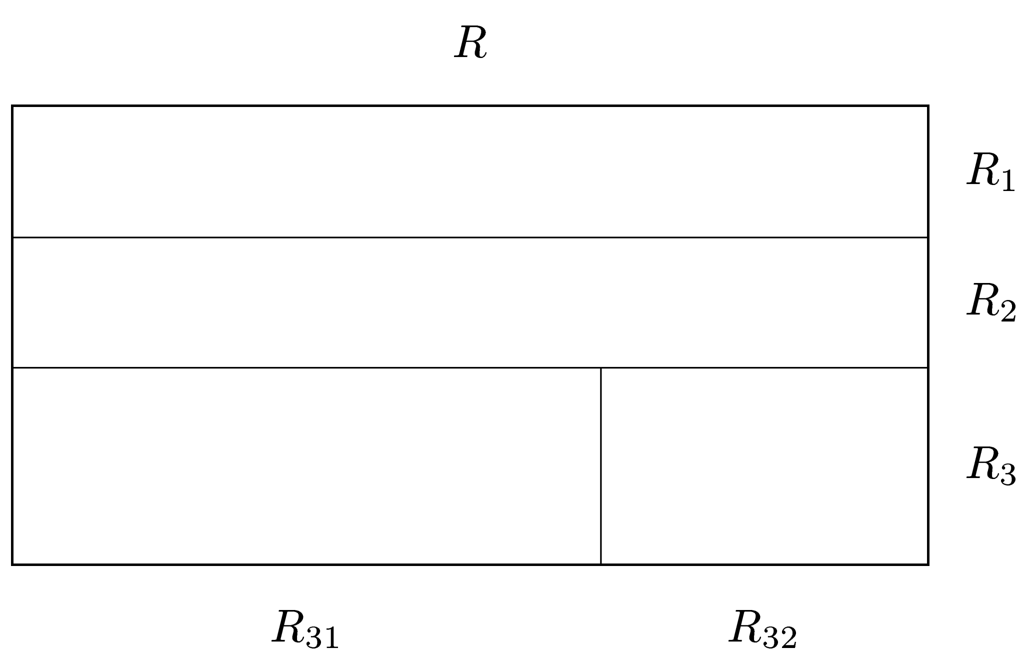


$$\begin{aligned} \text{ProfVerw} &:= \Pi_{\text{PersNr, Name, Gehalt, Steuerklasse}}(\text{Professoren}) \\ \text{Profs} &:= \Pi_{\text{PersNr, Name, Rang, Raum, Fakultät}}(\text{Professoren}) \end{aligned}$$

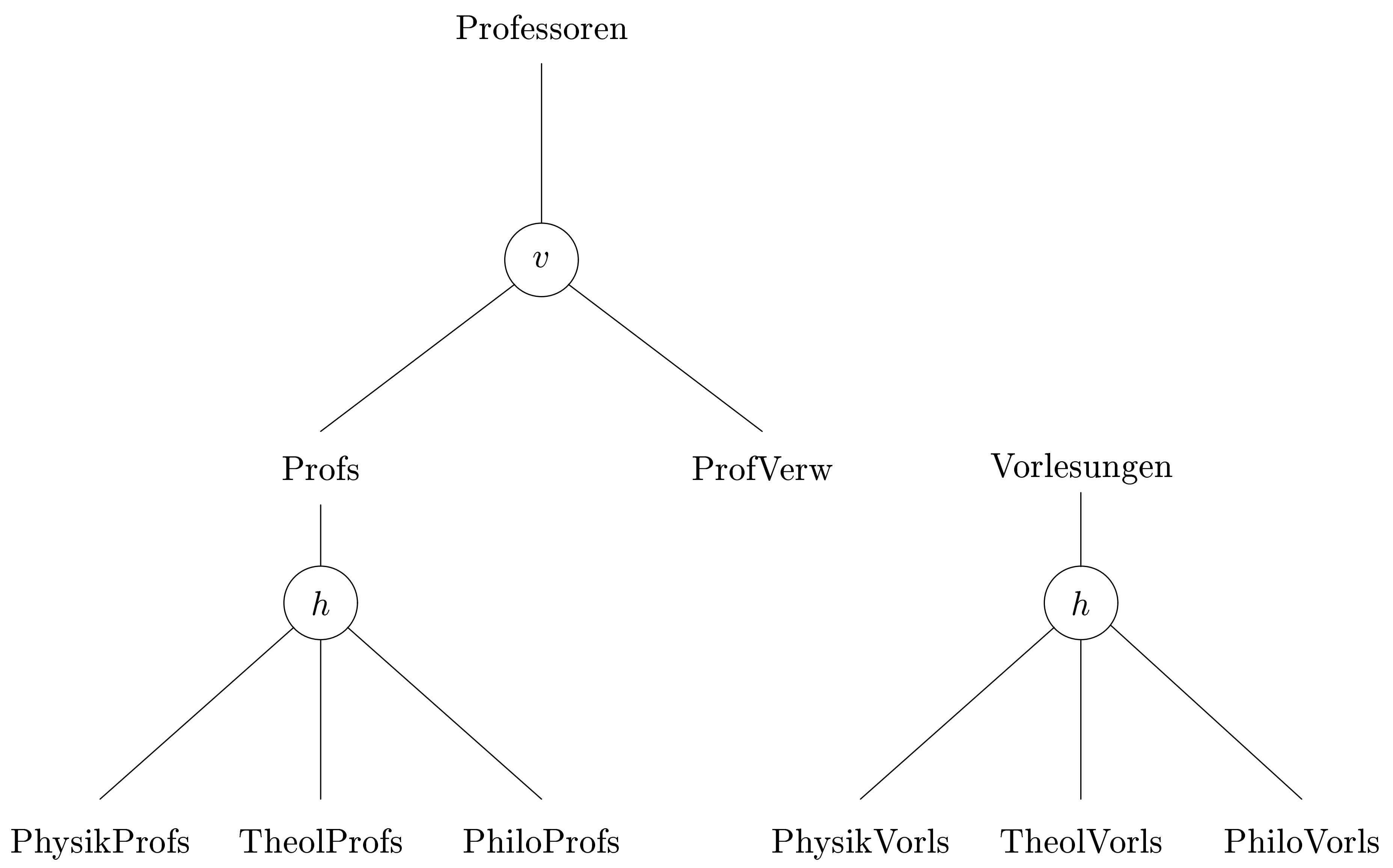
$$\text{Professoren} = \text{ProfVerw} \bowtie_{\text{ProfVerw.PersNr=Profs.PersNr}} \text{Profs}$$

Kombinierte Fragmentierung





Baumdarstellung der Fragmentierungen unserer Beispielanwendung



Allokation

Station	Bemerkung	zugeordnete Fragmente
S_{Verw}	Verwaltungsrechner	$\{ProfVerw\}$
S_{Physik}	Dekanat Physik	$\{PhysikVorls, PhysikProfs\}$
S_{Philo}	Dekanat Philosophie	$\{PhiloVorls, PhiloProfs\}$
S_{Theol}	Dekanat Theologie	$\{TheolVorls, TheolProfs\}$

- hier: redundanzfreie Allokation
- generell ist es aber auch möglich, dasselbe Fragment mehreren Stationen zuzuordnen

Fragmentierungstransparenz

select Titel, Name
from Vorlesungen, Professoren
where gelesenVon = PersNr

update Professoren
  set Fakultät = 'Theologie'
  where Name = 'Sokrates';

1. Ändern des Attributwertes von *Fakultät* in dem betreffenden Tupel
2. Einfügen des „Sokrates-Tupels" in *TheolProfs*
3. Löschen des Tupels aus *PhiloProfs*
4. Einfügen der von Sokrates gehaltenen Vorlesungen in *TheolVorls*
5. Löschen der von Sokrates gehaltenen Vorlesungen aus *PhiloVorls*

Allokationstransparenz

```
select Gehalt
from ProfVerw
where Name = 'Sokrates'
```

```
select sum(Gehalt)
from ProfVerw, TheolProfs
where ProfVerw.PersNr = TheolProfs.PersNr and
      Rang='C4'
```

Lokale Schema-Transparenz

select Name
from TheolProfs **at** S_{Theol}
where Rang = 'C3';

Anfragebearbeitung bei horizontaler Fragmentierung

select Titel
from Vorlesungen, Profs
where gelesenVon = PersNr **and**
Rang = 'C4';

1. Rekonstruiere alle in der Anfrage vorkommenden globalen Relationen aus den Fragmenten, in die sie während der Fragmentierungsphase zerlegt wurden. Hierfür erhält man einen algebraischen Ausdruck.

2. Kombiniere den Rekonstruktionsausdruck mit dem algebraischen Anfrageausdruck, der sich aus der Übersetzung der SQL-Anfrage ergibt.

Algebraische Äquivalenzen

$$S_i = S \ltimes_p R_i \quad \text{mit} \quad S = S_1 \cup \cdots \cup S_n$$

$$R_i \bowtie_p S_j = \emptyset \quad \text{für} \quad i \neq j.$$

$$(R_1 \cup \cdots \cup R_n) \bowtie_p (S_1 \cup \cdots \cup S_n) = (R_1 \bowtie_p S_1) \cup (R_2 \bowtie_p S_2) \cup \cdots \cup (R_n \bowtie_p S_n)$$

Beispiel

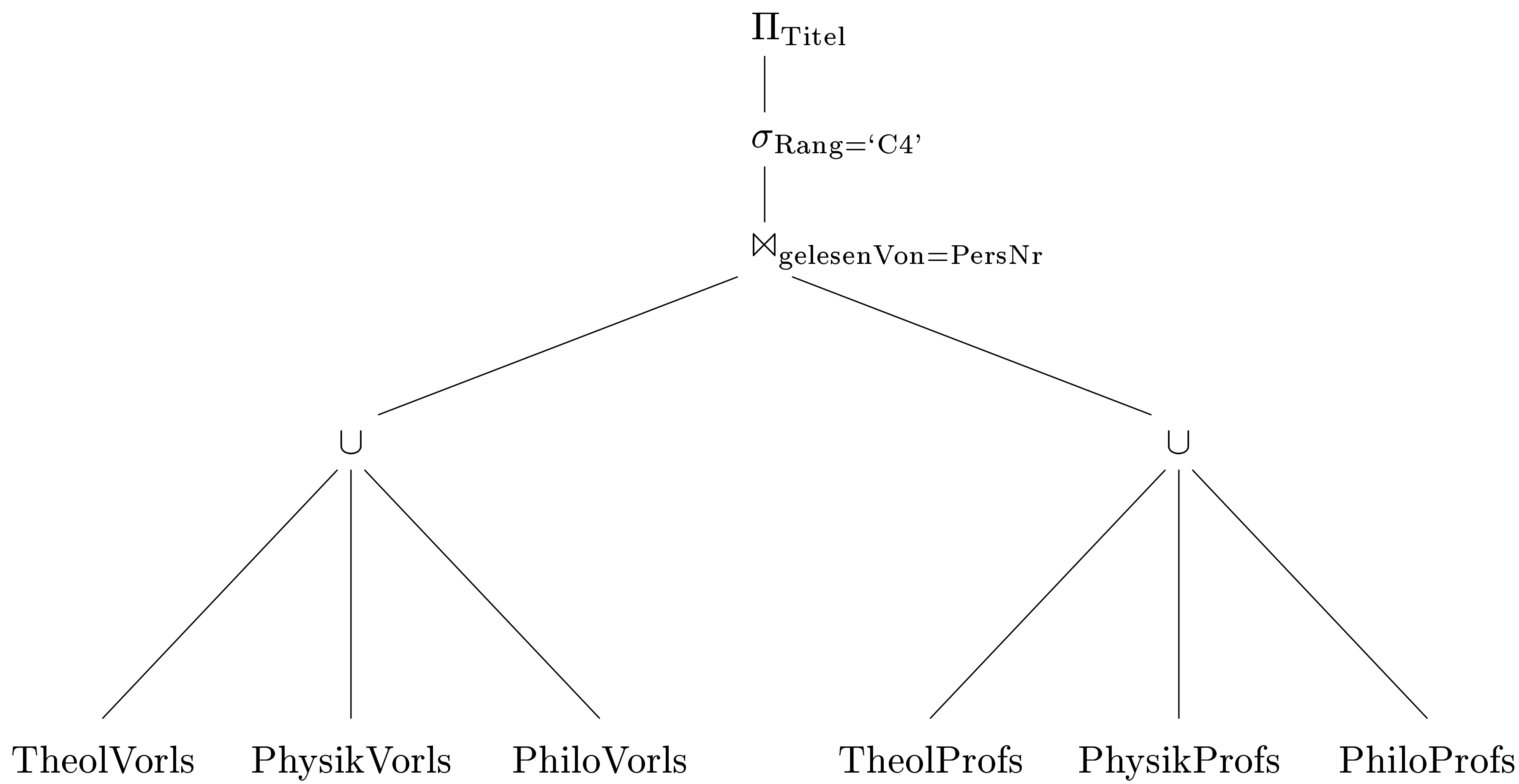
$$(\text{TheolVorls} \cup \text{PhysikVorls} \cup \text{PhiloVorls}) \bowtie_{\ldots}$$
$$(\text{TheolProfs} \cup \text{PhysikProfs} \cup \text{PhiloProfs})$$

es reichen folgende Joins

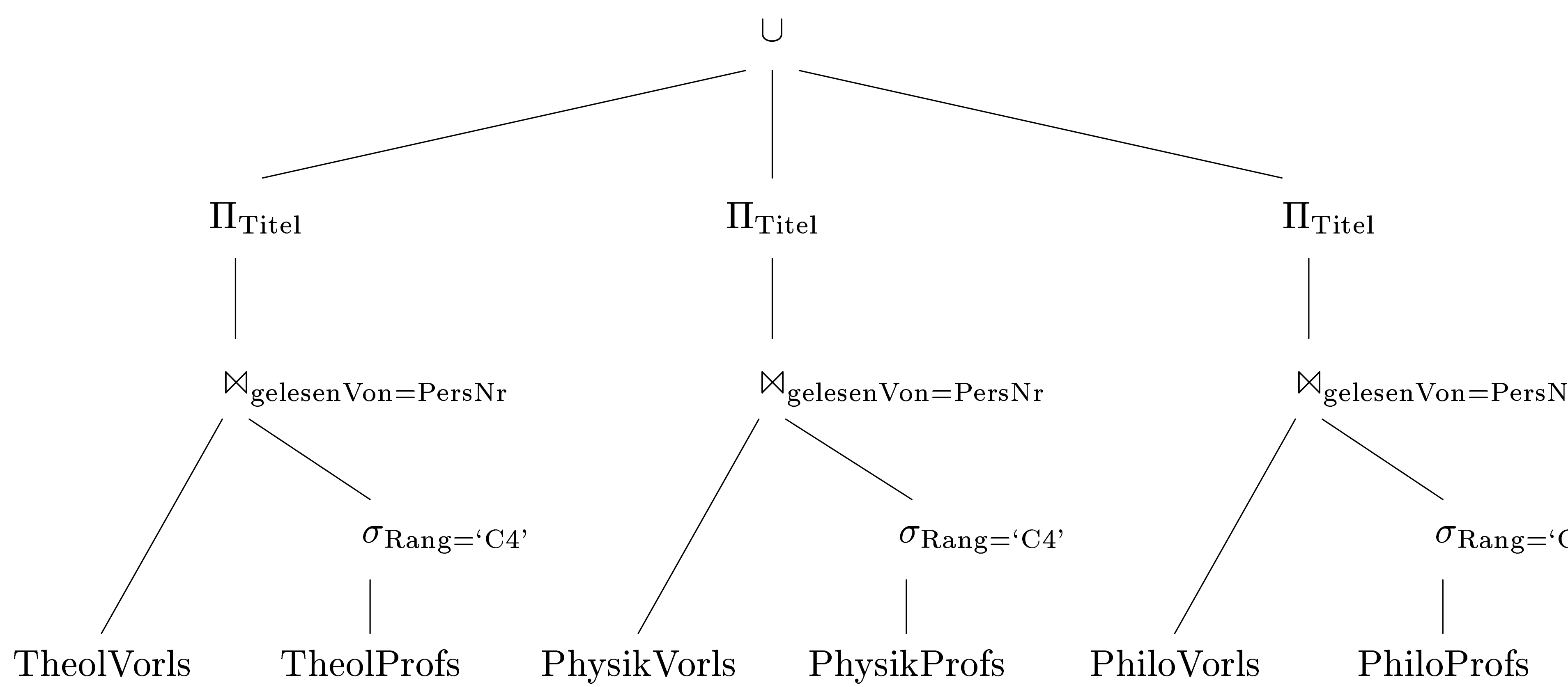
TheolProfs′	TheolVorls
PhysikProfs′	PhysikVorls
PhiloProfs′	PhiloVorls

$$\begin{aligned} \sigma_p(R_1 \cup R_2) &= \sigma_p(R_1) \cup \sigma_p(R_2) \\ \Pi_L(R_1 \cup R_2) &= \Pi_L(R_1) \cup \Pi_L(R_2) \end{aligned}$$

Kanonische Form der Anfrage



Optimale Form der Anfrage



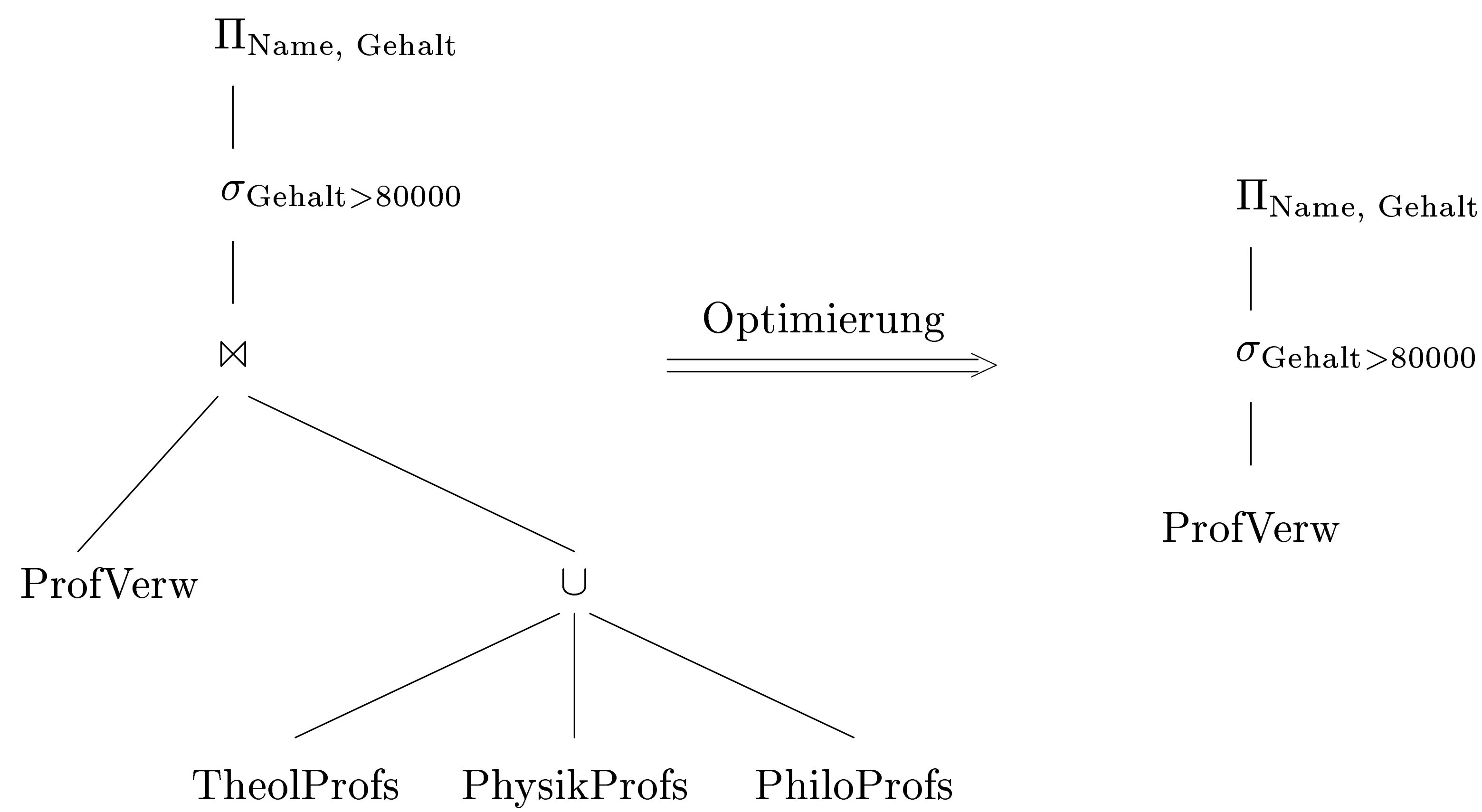
Anfragebearbeitung bei vertikaler Fragmentierung

select Name, Gehalt
from Professoren
where Gehalt $>$ 80000;

- Der Join kann bei der Optimierung eliminiert werden
- alle benötigten Informationen sind in *ProfVerw* enthalten
- Aber die folgende Anfrage wäre nicht gut zu optimieren (Das Attribut *Rang* fehlt in *ProfVerw*):

select Name, Gehalt, Rang
from Professoren
where Gehalt $>$ 80000;

Optimierung bei vertikaler Fragmentierung



Der natürliche Verbund zweier Relationen R und S

R		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_1
a_4	b_4	c_2
a_5	b_5	c_3
a_6	b_6	c_2
a_7	b_7	c_6

$\bowtie$

S		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
c_4	d_3	e_3
c_5	d_4	e_4
c_7	d_5	e_5
c_8	d_6	e_6
c_5	d_7	e_7

$=$

$R \bowtie S$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_3	b_3	c_1	d_1	e_1
a_5	b_5	c_3	d_2	e_2

Join-Auswertung ohne Filterung

1. *Nested-Loops*:

2. *Transfer einer Argumentrelation*:

3. *Transfer beider Argumentrelationen*:

Join-Auswertung mit Filterung ...

... einer Argumentrelation – hier S

$$R \bowtie S \quad = \quad R \bowtie (R \rtimes S)$$

$$R \rtimes S \quad = \quad \Pi_C(R) \rtimes S$$

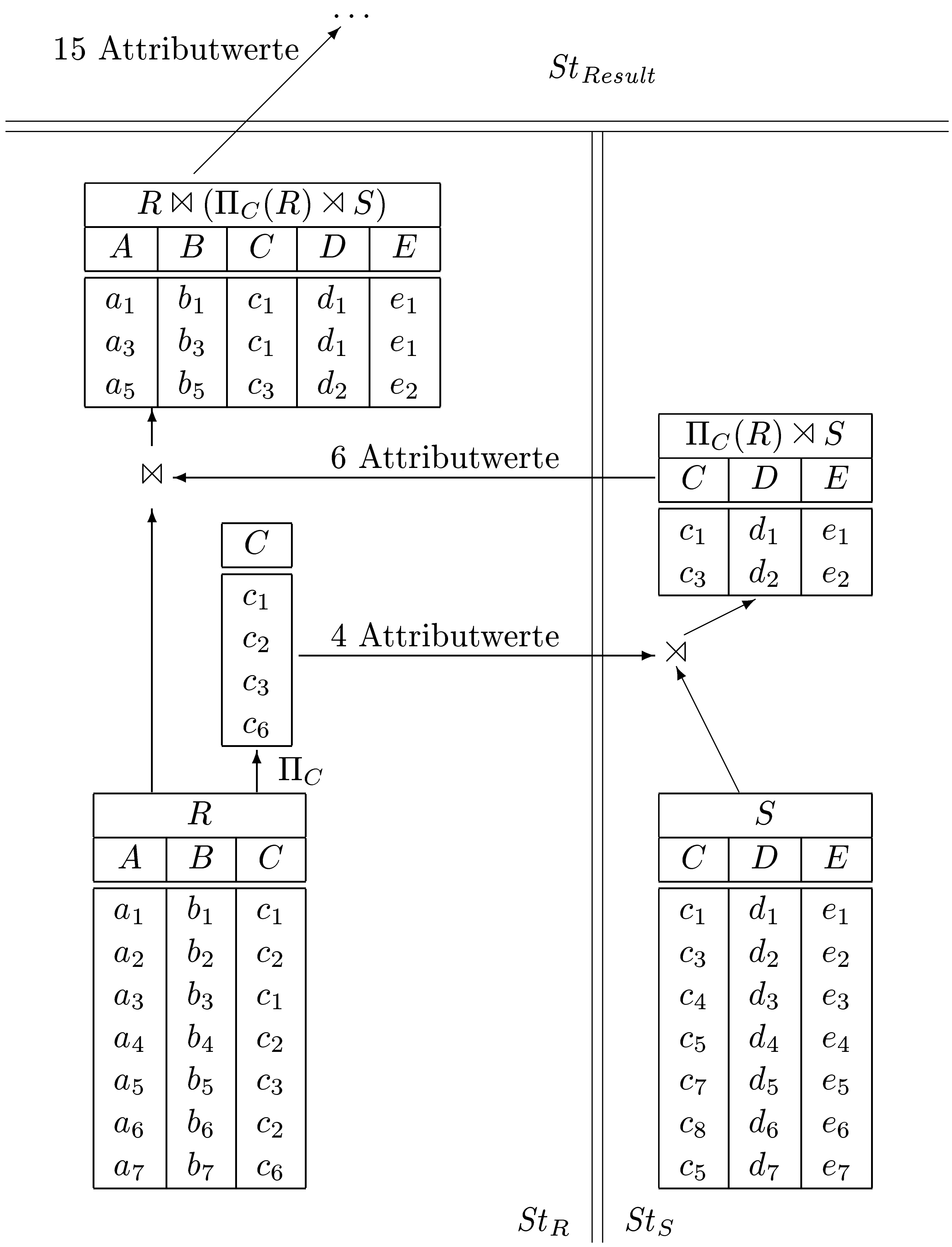
$$R \rtimes S = \Pi_C(R) \rtimes S$$

$$||\, \Pi_C(R)\, || + ||\, R \rtimes S\, || \quad < \quad ||\, S\, ||$$

... beider Argumentrelationen

$$(R \ltimes \Pi_C(S)) \bowtie (\Pi_C(R) \rtimes S)$$

Auswertung von $R \bowtie S$ mit Semi-Join-Filterung von S



$R \bowtie (\Pi_C(R) \rtimes S)$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_3	b_3	c_1	d_1	e_1
a_5	b_5	c_3	d_2	e_2

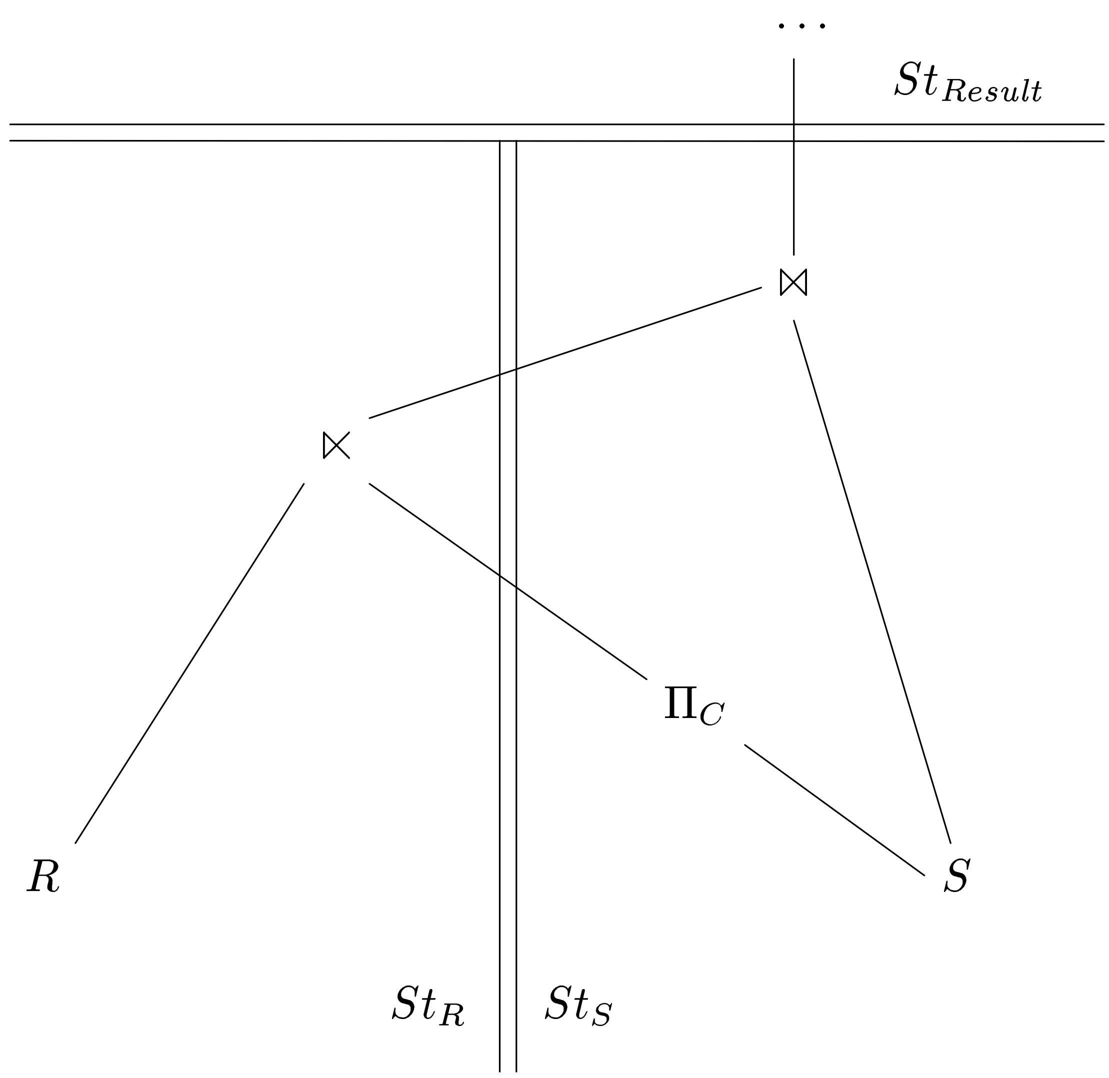
$\Pi_C(R) \rtimes S$		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

C
c_1
c_2
c_3
c_6

R		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_1
a_4	b_4	c_2
a_5	b_5	c_3
a_6	b_6	c_2
a_7	b_7	c_6

S		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
c_4	d_3	e_3
c_5	d_4	e_4
c_7	d_5	e_5
c_8	d_6	e_6
c_5	d_7	e_7

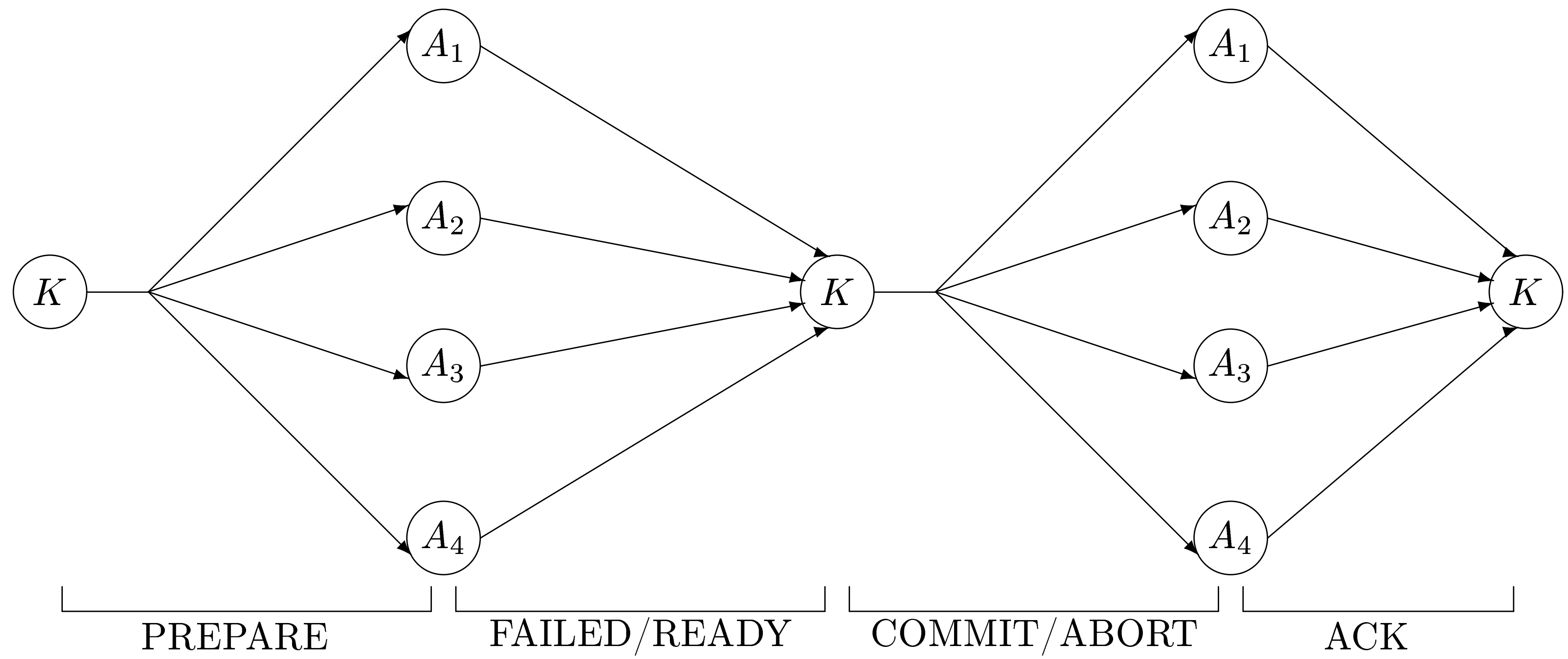
Alternativer Auswertungsplan mit Semi-Join-Filterung



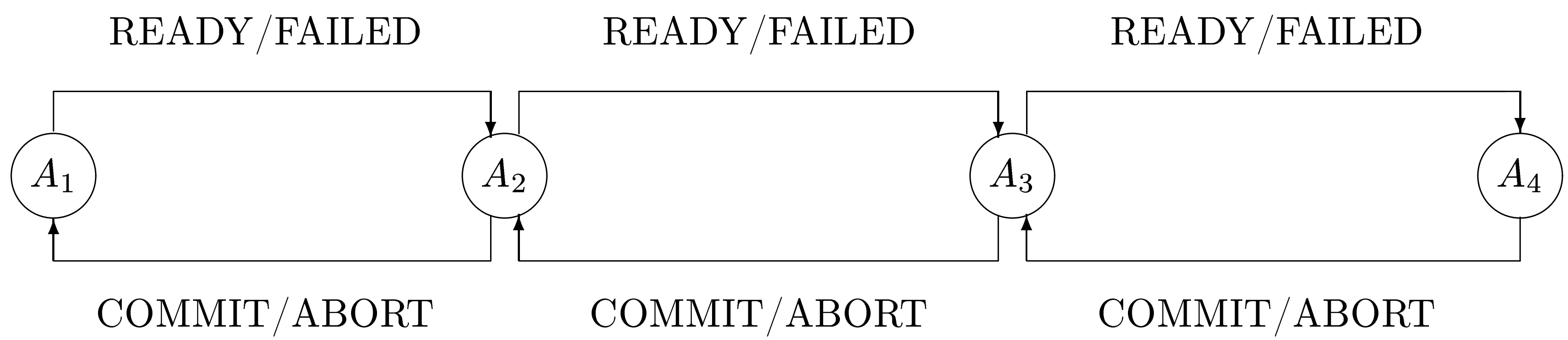
Recovery in VDBMS

- *Redo*: Wenn eine Station nach einem Fehler wiederanläuft, müssen alle Änderungen einmal abgeschlossener Transaktionen – seien sie lokal auf dieser Station oder global über mehrere Stationen ausgeführt worden – auf den an dieser Station abgelegten Daten wiederhergestellt werden.
- *Undo*: Die Änderungen noch nicht abgeschlossener lokaler und globaler Transaktionen müssen auf den an der abgestürzten Station vorliegenden Daten rückgängig gemacht werden.

Nachrichtenaustausch zwischen *Koordinator* und *Agenten* beim 2PC-Protokoll

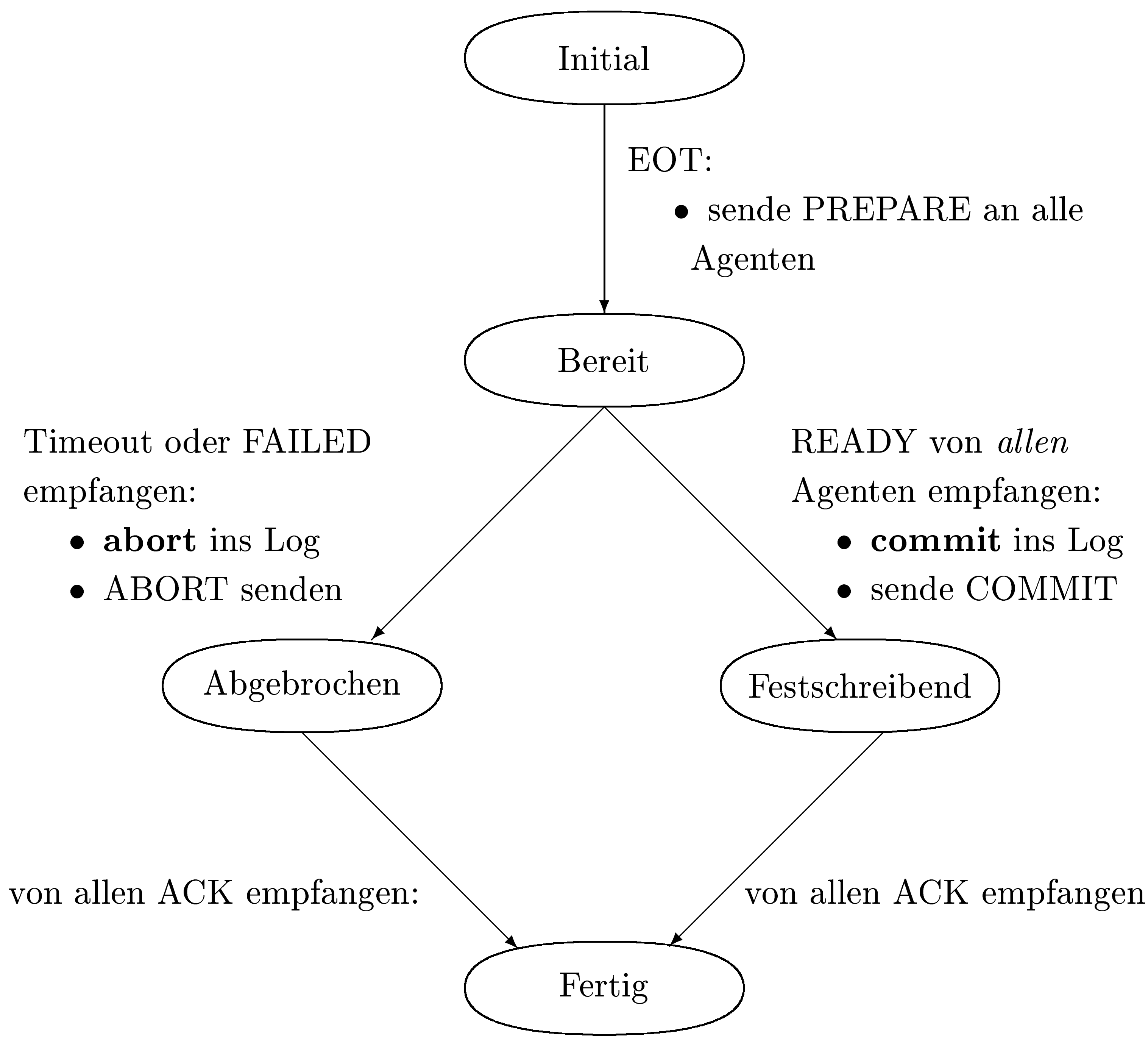


Lineare Organisationsform bei 2PC-Protokoll

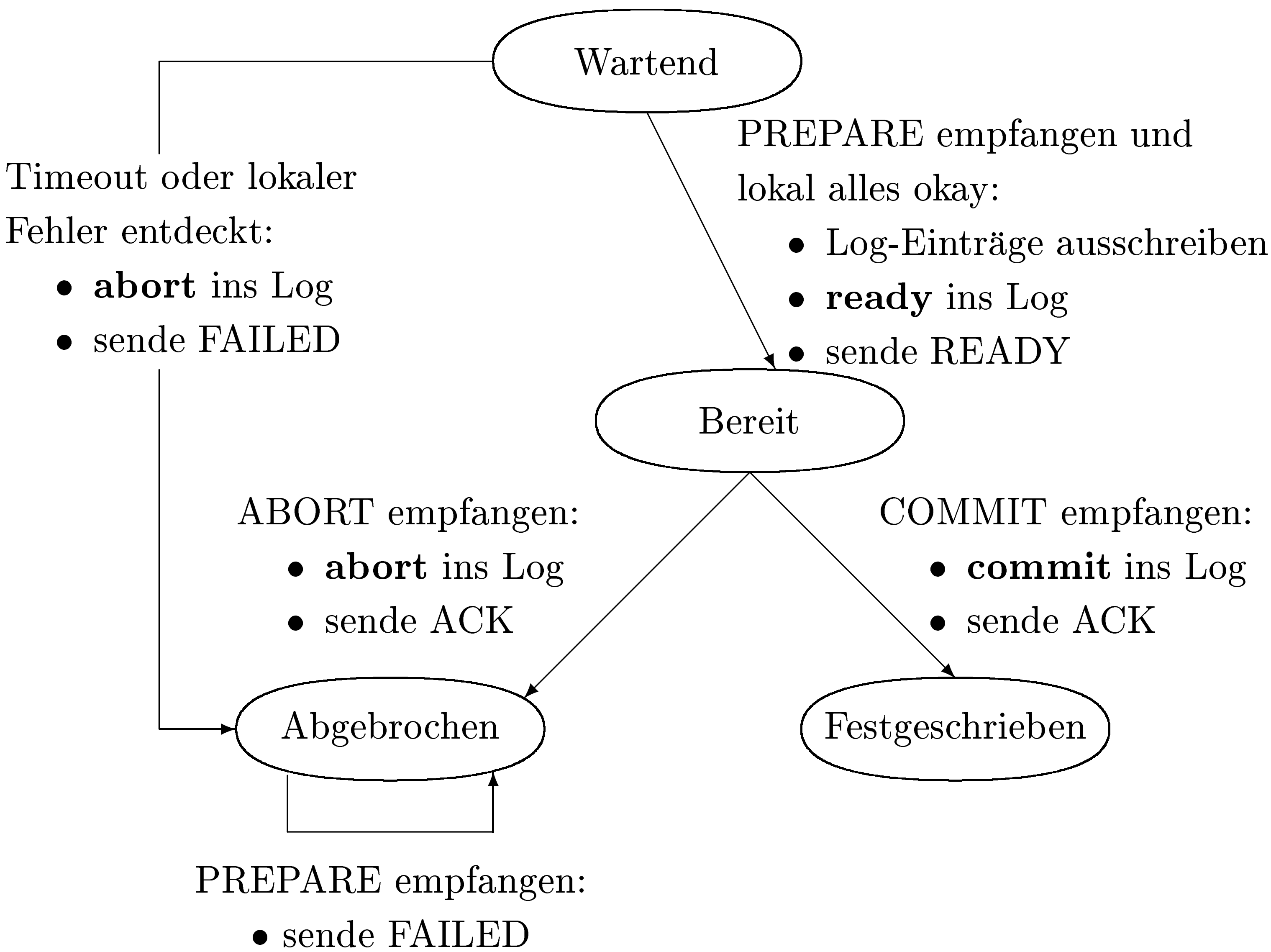


Zustände des Zweiphasen-Commit-Protokolls

(a) Koordinator



(b) Agent


Wartend
PREPARE empfangen und
lokal alles okay:
• Log-Einträge ausschreiben
• **ready** ins Log
• sende READY
Timeout oder lokaler
Fehler entdeckt:
• **abort** ins Log
• sende FAILED
Bereit
ABORT empfangen:
• **abort** ins Log
• sende ACK
COMMIT empfangen:
• **commit** ins Log
• sende ACK
Abgebrochen
Festgeschrieben
PREPARE empfangen:
• sende FAILED

Fehlersituation des 2PC-Protokolls

Absturz eines Agenten

Verlorengegangene Nachrichten

Mehrbenutzersynchronisation in VDBMS

S_1

Schritt	T_1	T_2
1.	$r(A)$	
2.		$w(A)$

S_2

Schritt	T_1	T_2
3.		$w(B)$
4.	$r(B)$	

- 2PL garantiert globale Serialisierbarkeit
- Sperrenverwaltung
  - zentral/global
  - lokal

Deadlock in VDBMS

S_1

Schritt	T_1	T_2
0.	**BOT**	
1.	**lockS**(A)	
2.	$r(A)$	
6.		**lockX**(A)
		$\sim\sim\sim\sim$

S_2

Schritt	T_1	T_2
3.		**BOT**
4.		**lockX**(B)
5.		$w(B)$
7.	**lockS**(B)	
	$\sim\sim\sim\sim$	

Erkennung von Deadlocks

Timeout

- einfach zu realisieren
- Problem: wie lange wartet man?

Zentralisierte Deadlock-Erkennung

- sichere Lösung
- aber: Flaschenhals des Systems

Dezentrale (verteilte) Deadlock-Erkennung

Der Algorithmus aus System R*

$$External \rightarrow T_i$$

- Diese Kante wird für jede "von außen" kommende Transaktion T_i kreiert

$$T_j \rightarrow External$$

- Diese Kante wird für jede Transaktion T_j dieser Station kreiert, falls die TA "nach außen" geht.

Unser Beispiel

$$S_1 : \boxed{External \rightarrow T_2 \rightarrow T_1 \rightarrow External}$$

$$S_2 : \boxed{External \rightarrow T_1 \rightarrow T_2 \rightarrow External}$$

$$S_2 : \boxed{External \rightleftarrows T_1 \rightleftarrows T_2 \rightleftarrows External}$$

$$T_1 \rightarrow T_2 \rightarrow T_1$$

$$T_2 \rightarrow T_1 \rightarrow T_2$$

Reduzierung des Nachrichtenaufkommens

$$External \rightarrow T_1' \rightarrow T_2' \rightarrow \ldots \rightarrow T_n' \rightarrow External$$

- dieser Pfad wird nur weitergereicht (path pushing) wenn T_1' einen kleineren Identifikator hat als T_n'

Deadlock-Vermeidung

- Die optimistische Mehrbenuztersynchronisation
- Die Zeitstempel-basierende Synchronisation
- *wound/wait*:
- *wait/die*:
- Generierung eindeutiger Zeitstempel in verteilten Systemen:

lokale Zeit	Stations-ID

Station (S_i)	Kopie (A_i)	Gewicht (w_i)
S_1	A_1	3
S_2	A_2	1
S_3	A_3	2
S_4	A_4	2

$W(A) = \sum_{i=1}^{4} w_i(A) = 8.$

Lesequorum $Q_r(A)$

Schreibquorum $Q_w(A)$

Die beiden folgenden Bedingungen müssen gelten:

1. $Q_w(A) + Q_w(A) > W(A)$
   (verhindert zwei gleichzeitige Schreib-TAs)

2. $Q_r(A) + Q_w(A) > W(A)$
   (verhindert gleichzeitige Schreib-TA mit Lese-TA)

Beispiel

- $Q_r(A) = 4$
- $Q_w(A) = 5$

Zustand (a) vor und (b) nach Schreiben eines Schreibquorums

(a)

Station	Kopie	Gewicht	Wert	Versions#
S_1	A_1	3	1000	1
S_2	A_2	1	1000	1
S_3	A_3	2	1000	1
S_4	A_4	2	1000	1

(b)

Station	Kopie	Gewicht	Wert	Versions#
S_1	A_1	3	1100	2
S_2	A_2	1	1000	1
S_3	A_3	2	1100	2
S_4	A_4	2	1000	1

Leistungsbewertung und neuere Datenbankanwendungen

- Leistungsbewertung von Datenbanksystemen anhand standardisierter Benchmarks
  - TPC-C: OLTP (Online Transaction Processing)-Benchmark
  - TPC-D: OLAP (Online Analytical Processing)/Decision Support-Benchmark
  - OO7: Benchmark für objektorientierte DBMS
- SAP R/3: Integriertes betriebswirtschaftliches Anwendungssystem
- Data Warehouse-Konzepte
- Data Mining

TPC-C Datenbanksystem-Benchmark

- Online Transaction Processing
- vornehmlich kurze Transaktionen
- Zugriff auf eng begrenztes Datenvolumen

Die Datenbank: Schema und Ausprägung

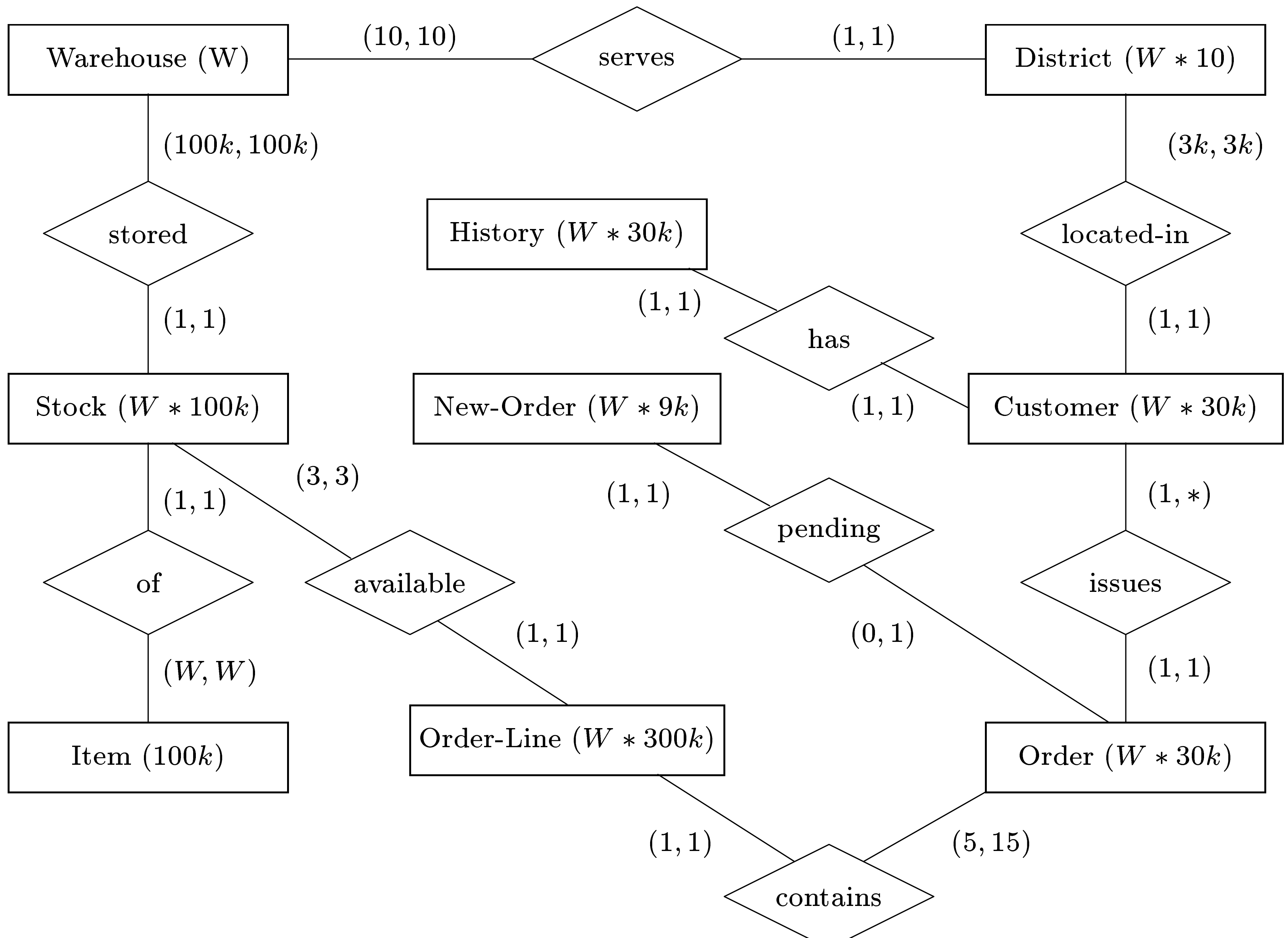
- *Warehouse*: Es werden $W \geq 1$ Warenhäuser durch je ein Tupel modelliert.
- *District*: Pro Warenhaus gibt es 10 Distrikte, deren Kunden vornehmlich (wenn die bestellten Waren vorhanden sind) von dem zugehörigen Warenhaus beliefert werden.
- *Customer*: In jedem Distrikt gibt es 3000 (3k) Kunden.
- *Order*: In der Anfangskonfiguration hat jeder Kunde bereits eine Bestellung aufgegeben. Es kommen dann im Laufe der Benchmark-Durchführung neue Bestellungen hinzu und ausstehende (engl. *pending*) Bestellungen werden kontinuierlich abgearbeitet.
- *New-Order*: Eine neu aufgenommene Bestellung wird bis zur Belieferung in dieser Relation eingetragen. Genauer gesagt, die Tupel dieser Relation stellen Verweise auf noch nicht abgearbeitete Einträge in *Order* dar.
- *Order-Line*: Jede Bestellung besteht aus durchschnittlich zehn (variierend zwischen fünf bis fünfzehn) Auftragspositionen.

- *Stock*: Diese Relation modelliert die Verfügbarkeit von Produkten in den einzelnen Warenhäusern. Stock enthält pro (Warenhaus, Produkt)-Paar einen Eintrag – also $W * 100k$ Tupel. Eine Auftragsposition wird aus dem Warenbestand (Stock) eines Warenhauses abgedeckt, was durch die Beziehung *available* modelliert wird.

- *Item*: Diese Relation enthält ein Tupel für jedes der 100000 Produkte (Item), die das Handelsunternehmen anbietet. Die Relation *Item* nimmt bei der Skalierung der Datenbasis eine Sonderstellung ein; sie wird in der Größe nicht verändert, auch wenn die Anzahl der Warenhäuser (W) erhöht wird.

- *History*: Diese Relation enthält Daten zur Bestellhistorie der einzelnen Kunden.

ER-Schema



Warehouse (W)
(10, 10)
serves
(1, 1)
District ($W * 10$)
(100k, 100k)
(3k, 3k)
stored
History ($W * 30k$)
located-in
(1, 1)
has
(1, 1)
(1, 1)
Stock ($W * 100k$)
New-Order ($W * 9k$)
(1, 1)
Customer ($W * 30k$)
(3, 3)
(1, 1)
(1, 1)
(1, *)
pending
of
available
issues
(1, 1)
(0, 1)
(1, 1)
(W, W)
Item (100k)
Order-Line ($W * 300k$)
Order ($W * 30k$)
(1, 1)
(5, 15)
contains

Die Operationen

1. *New-Order*: In dieser Transaktion wird eine komplette Neubestellung von fünf bis fünfzehn Auftragspositionen in die Datenbasis eingegeben. Für jede dieser Auftragspositionen wird die Verfügbarkeit des jeweiligen Produkts in der *Stock*-Relation überprüft.

2. *Payment*: Die Zahlung eines Kunden wird verbucht. Dazu werden zusätzlich Verkaufsstatistiken in den Relationen *District* und *Warehouse* fortgeschrieben.

3. *Order-Status*: Dies ist eine reine Lesetransaktion, in der der Status der letzten Bestellung eines bestimmten Kunden überprüft wird.

4. *Delivery*: In dieser Transaktion werden zehn Bestellungen aus der *New-Order* Relation im Batch-Modus (also ohne Benutzerinteraktion) bearbeitet. Die bearbeiteten Bestellungen werden aus der *New-Order* Relation entfernt.

5. *Stock-Level*: Dies ist eine Lesetransaktion, die den Warenbestand der in letzter Zeit bestellten Produkte kontrolliert. Der TPC-C-Benchmark erlaubt die Aufspaltung dieser eine große Anzahl von Tupeln lesenden Transaktion in kleinere Datenbank-Transaktionen, um dadurch den Overhead der Mehrbenutzersynchronisation zu reduzieren.

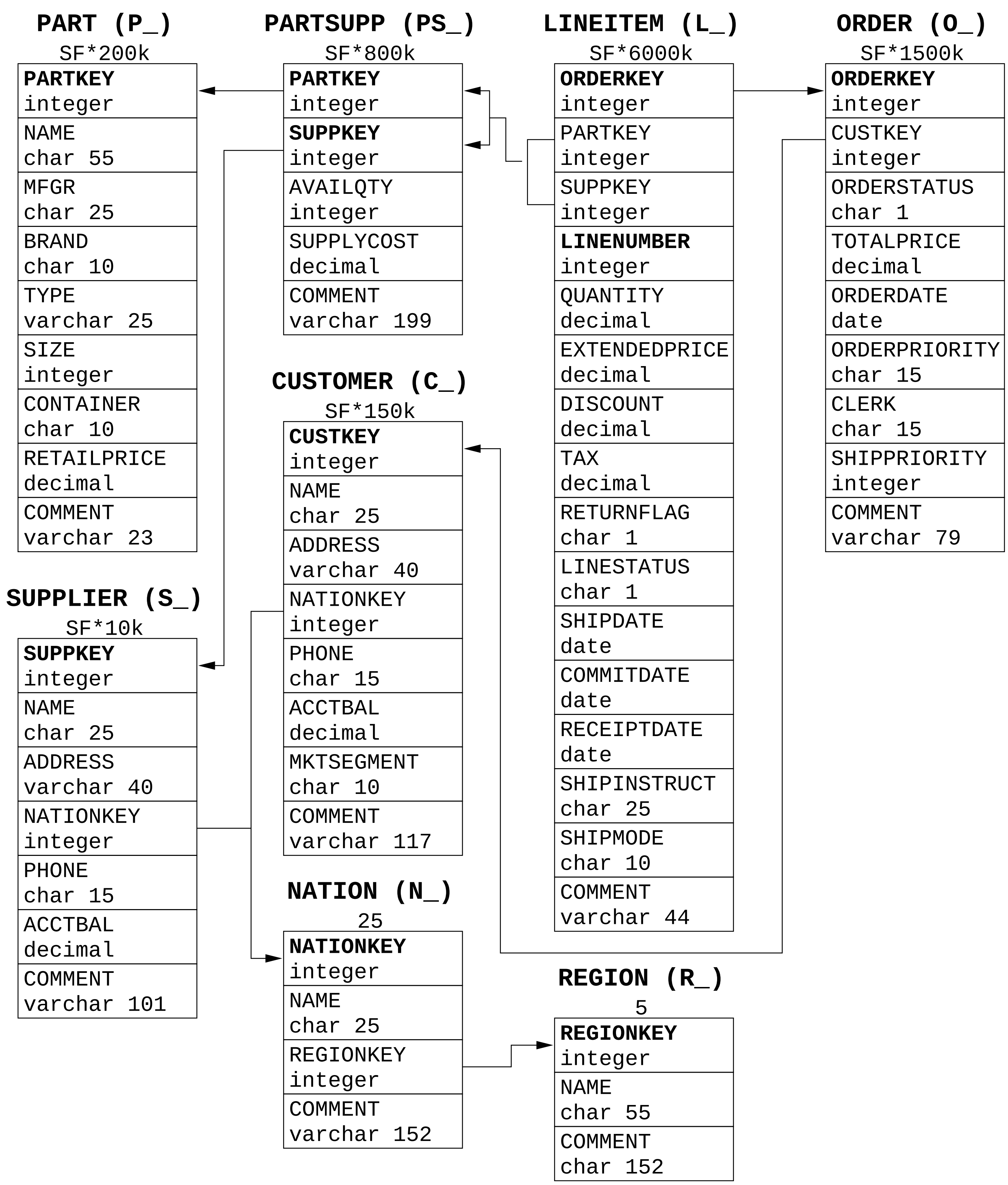
Leistungs-Kennzahlen

- Nur die Anzahl der *New-Order*-Transaktionen wird gemessen – die anderen müssen natürlich in einem bestimmten Prozentsatz auch ausgeführt werden
- *Durchsatz* pro Minute wird bekannt gegeben
- *Systempreis* errechnet sich aus Hardware und Software, inkl. 5 Jahre Wartung
- Das *Preis/Leistungsverhältnis* (in Dollar pro Transaktion) wird errechnet aus dem Systempreis und dem Durchsatz

Heute erzielbare Leistungszahlen: "Eckwerte"

- 2.500 Transaktionen pro Minute bei einem Systempreis von ca 200.000 US Dollar (also etwa 70 Dollar pro Transaktion im Preis/Leistungsverhältnis)
- 23.000 Transaktionen pro Minute bei einem Systempreis von ca 2.750.000 US Dollar (also etwa 120 Dollar pro Transaktion im Preis/Leistungsverhältnis)
- Ergebnisse werden auf dem Webserver der TPC-Organisation veröffentlicht: http://www.tpc.org/

Der TPC-D-Benchmark



Beispiel-Anfragen: die ersten 3 von 17

1. Man erstelle einen aufsummierten Preisbericht über alle Auftragspositionen, die spätestens 90 Tage vor dem 1. Dezember 1998 versandt wurden. Die Ausgabe soll nach *RETURNFLAG* und *LINESTATUS* gruppiert und in aufsteigender Reihenfolge nach diesen Attributen sortiert werden. Für jede Gruppe soll die gesamte Menge, der Gesamtpreis, der ermäßigte Gesamtpreis, der ermäßigte Gesamtpreis inklusive Steuern, die durchschnittliche Anzahl, der durchschnittliche Gesamtpreis und der durchschnittliche Nachlaß und die Anzahl der Auftragspositionen aufgelistet werden.

2. Für jedes Teil aus Messing (engl. *brass*) mit Größe 15 soll festgestellt werden, welcher Zulieferer in Europa beim nächsten Auftrag ausgewählt werden sollte. Das Kriterium für die Wahl eines Lieferanten sind dabei minimale Lieferkosten. Die Anfrage soll für jeden qualifizierenden Lieferanten den Kontostand, Namen, Land, Teilenummer, Hersteller des Teils, sowie Adresse und Telefonnummer des Lieferanten auflisten.

3. Man berechne den durchschnittlichen jährlichen Einnahmenverlust, der sich ergeben würde, falls Aufträge mit kleineren Mengen (unter 20% der Durchschnittsmenge für dieses Teil) für die Sorte 23 im Container „LG BOX“ nicht mehr angenommen würden.

UF1 Mit Hilfe dieser Updatefunktion werden neue Verkaufsinformationen in die Datenbank eingefügt. Dazu lädt sie zusätzliche Datensätze in die Tabellen *ORDER* und *LINEITEM*, welche zuvor mit dem Programm DBGEN erzeugt wurden. Insgesamt müssen $SF * 1500$ neue Tupel in die Relation *ORDER* und pro neuer Bestellung eine zufällig im Bereich 1 bis 7 gewählte Anzahl von zugeordneten *LINEITEM*-Tupeln eingefügt werden.

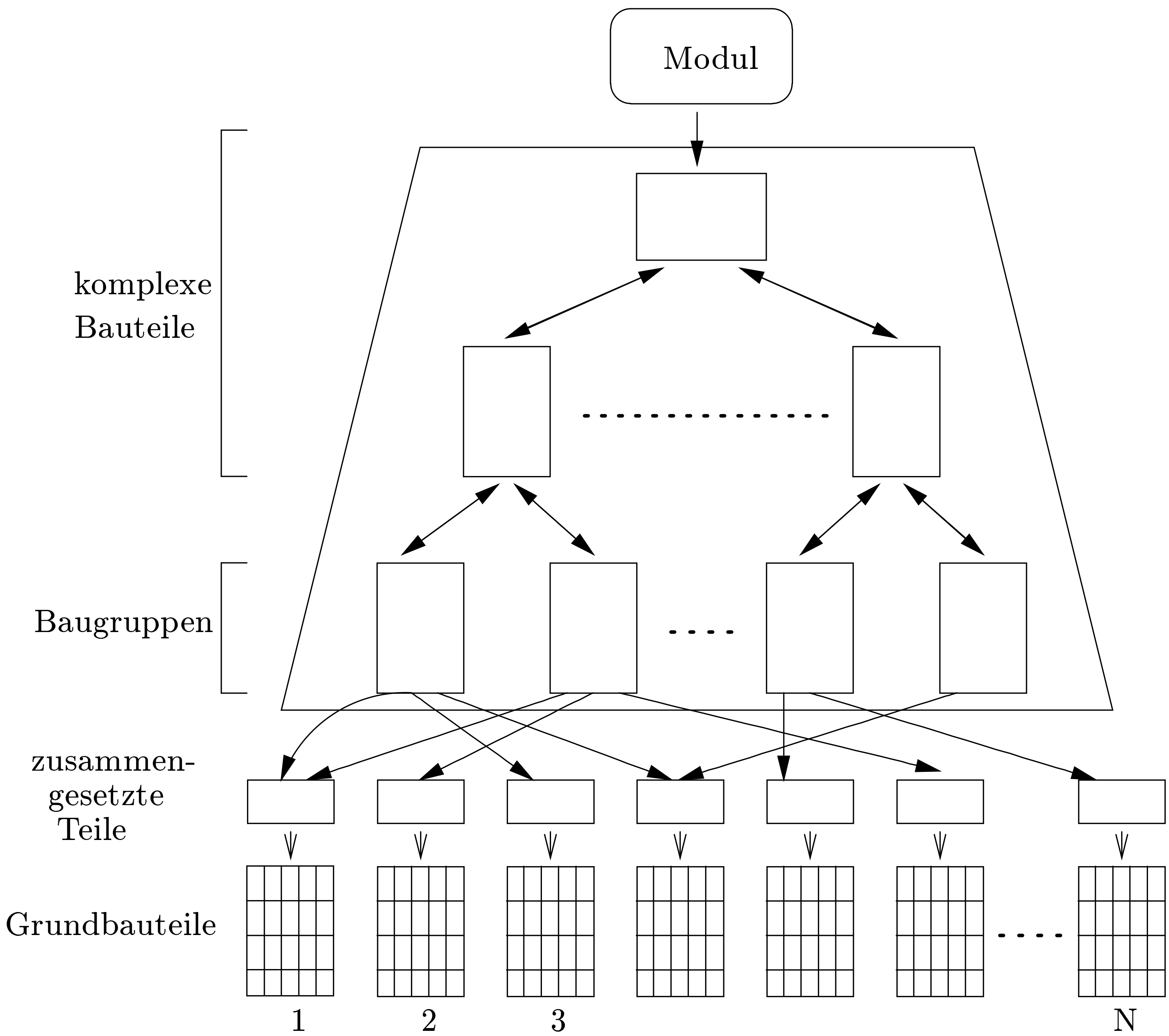
UF2 Diese Funktion entfernt überholte bzw. überflüssige Informationen aus der Datenbank, indem sie die entsprechenden Datensätze in den Tabellen *ORDER* und *LINEITEM* löscht. Insgesamt werden $SF * 1500$ Tupel aus *ORDER* gelöscht und alle zu diesen gelöschten Bestellungen gehörenden Einträge aus *LINEITEM*.

Leistungsgrößen

- Der Systempreis
- Die TPC-D Powermetrik QppD@Size, die in Anzahl von sequentiell ausgeführten Anfragen und Änderungen pro Stunde angegeben wird
- Der Durchsatz QthD@Size, der sich aus der Anzahl bearbeiteter Anfragen pro Stunde ergibt
- Das Preis/Leistungsverhältnis, das in Dollar pro Anfrage pro Stunde angegeben wird

Heutige Leistungs-Kennzahlen

- `http://www.tpc.org/`
- Die besten veröffentlichten Zahlen für eine 300 GB große Datenbank liegen etwa bei Werten von QppD@300GB = 2000 für den Powertest und QthD@300GB = 1200 $
- Für 1 Terabyte sind im Powertest ca. 4700 und im Throughput ca. 1600 Anfragen bei einem Systempreis von 10.000.000 $ möglich


Modul
komplexe Bauteile
Baugruppen
zusammen-gesetzte Teile
Grundbauteile
1
2
3
N

SAP R/3: Ein betriebswirtschaftliches Datenbankanwendungssystem

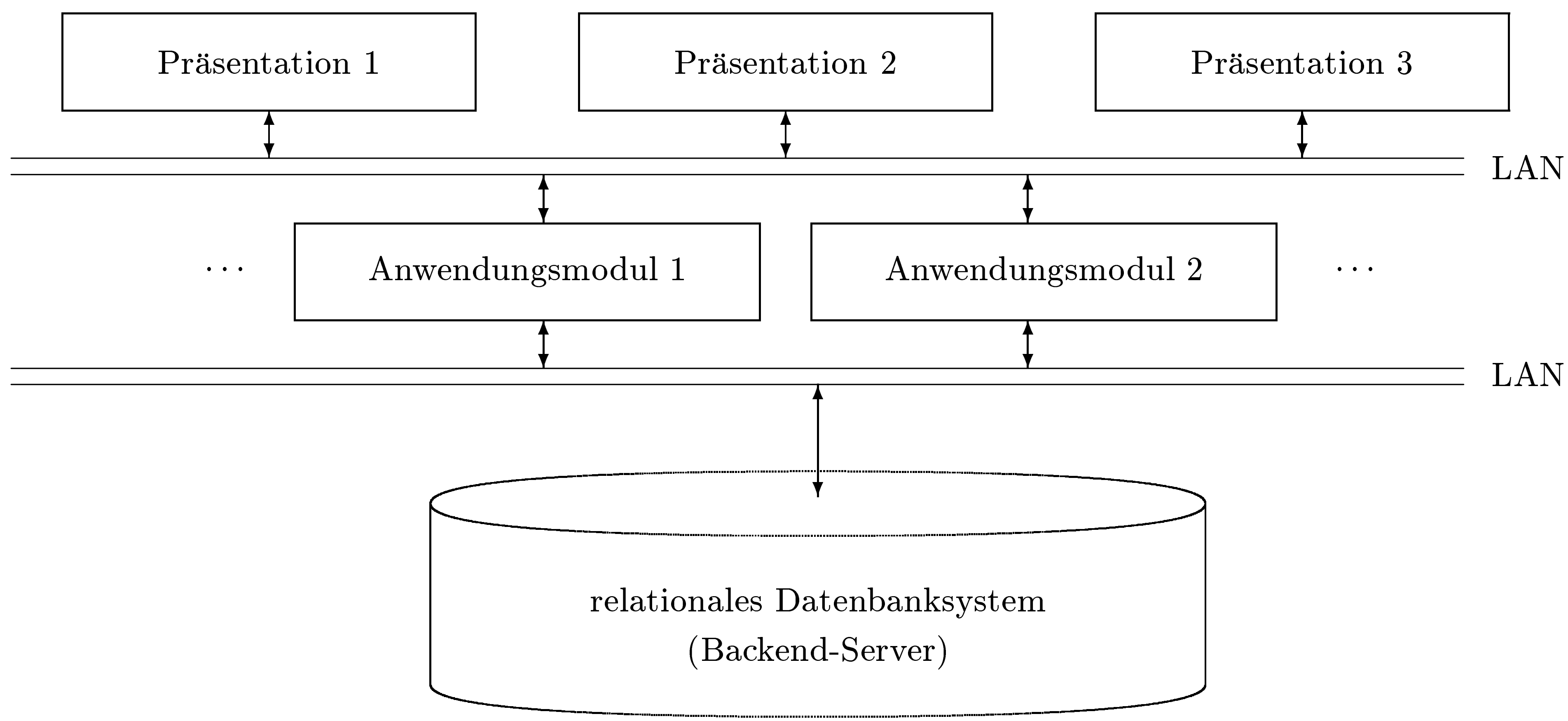
Drei-stufige Client/Server-Architektur von SAP R/3, bestehend aus

1. der Präsentationsebene, die den Endanwendern eine graphische (GUI) Dialogschnittstelle zur Verfügung stellt;

2. der Anwendungsebene, die das betriebswirtschaftliche „Know How“ (also die eigentlichen betriebswirtschaftlichen Anwendungsprogramme) beinhaltet;

3. der Datenhaltungsebene, die auf einem fremdbezogenen relationalen Datenbanksystem basiert.

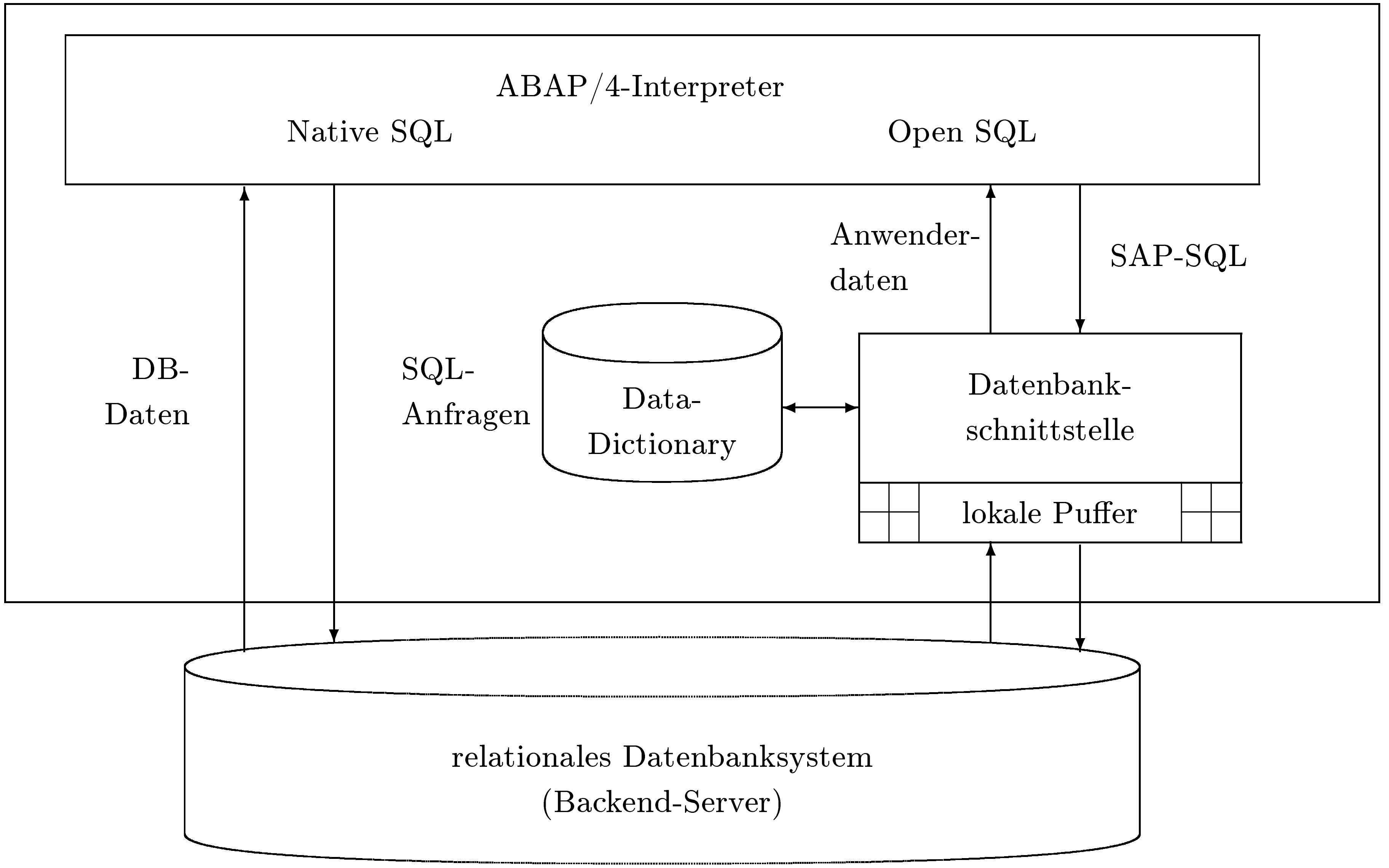
Architektur-Übersicht



Datenmodell und Schema von SAP R/3

- basiert auf einem relationalen Datenbanksystem
- mögliche Plattformen sind: Oracle, Informix, (Adabas D), MS SQL Server, DB 2 von IBM
- (all-)umfassendes betriebswirtschaftliches Informationsmodell
- ca. 10.000 Relationen
- *transparente* Tabellen
  - Abbildung 1 : 1 auf DB-Relationen
- *Pool*- und *Cluster*-Tabellen des R/3-Systems
  - Abbildung $N : 1$ auf DB-Relationen

SAP R/3


ABAP/4-Interpreter
Native SQL
Open SQL
DB-
Daten
SQL-
Anfragen
Data-
Dictionary
Anwender-
daten
SAP-SQL
Datenbank-
schnittstelle
lokale Puffer
relationales Datenbanksystem
(Backend-Server)

Syntax von Datenbankzugriffen aus ABAP/4

```
SELECT ⟨Attributliste⟩
FROM ⟨eine Tabelle⟩
WHERE ⟨einfaches Prädikat⟩
...     Bearbeitung des aktuellen Tupels
ENDSELECT.
```

```
SELECT SINGLE ⟨Attributliste⟩
FROM ⟨eine Tabelle⟩
WHERE ⟨einfaches Prädikat⟩
... Bearbeitung des einen Tupels
```

```
SELECT ⟨Attributliste⟩
FROM ⟨äußere Tabelle⟩
WHERE ⟨einfaches Prädikat⟩.
        SELECT ⟨Attributliste⟩
        FROM ⟨innere Tabelle⟩
        WHERE ⟨Joinprädikat⟩.
        ... Bearbeitung des aktuellen inneren Tupels
        ENDSELECT.
... Bearbeitung des aktuellen äußeren Tupels
ENDSELECT.
```

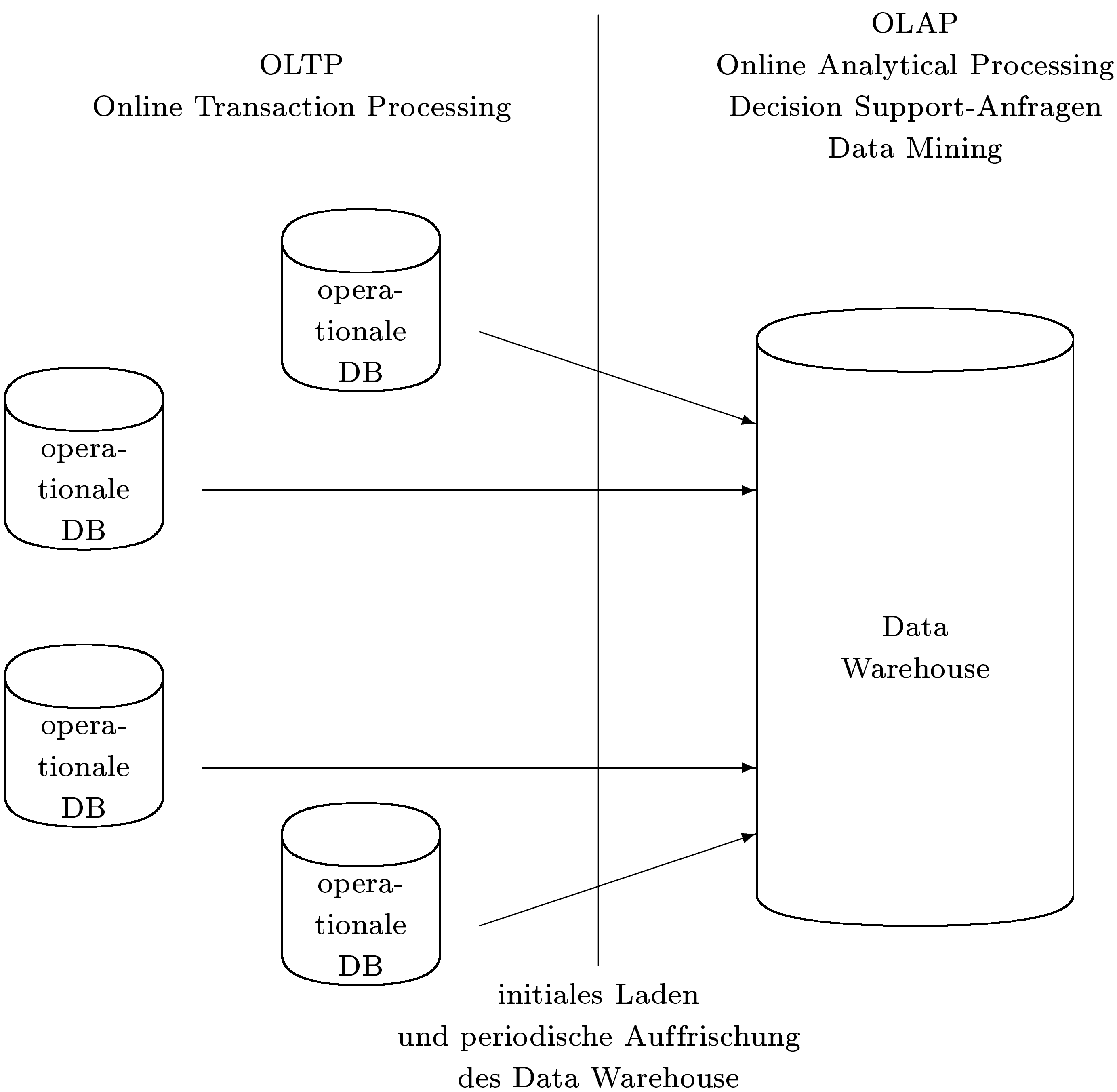
Seit neeuestem: Joins in ABAP/4

SELECT $\langle$Attributliste$\rangle$
FROM $\langle$Tabelle1$\rangle$ JOIN $\langle$Tabelle2$\rangle$ ON $\langle$Join Prädikat$\rangle$
WHERE $\langle$Selektions-Prädikat$\rangle$.

Data Warehouse, Decision-Support, OLAP

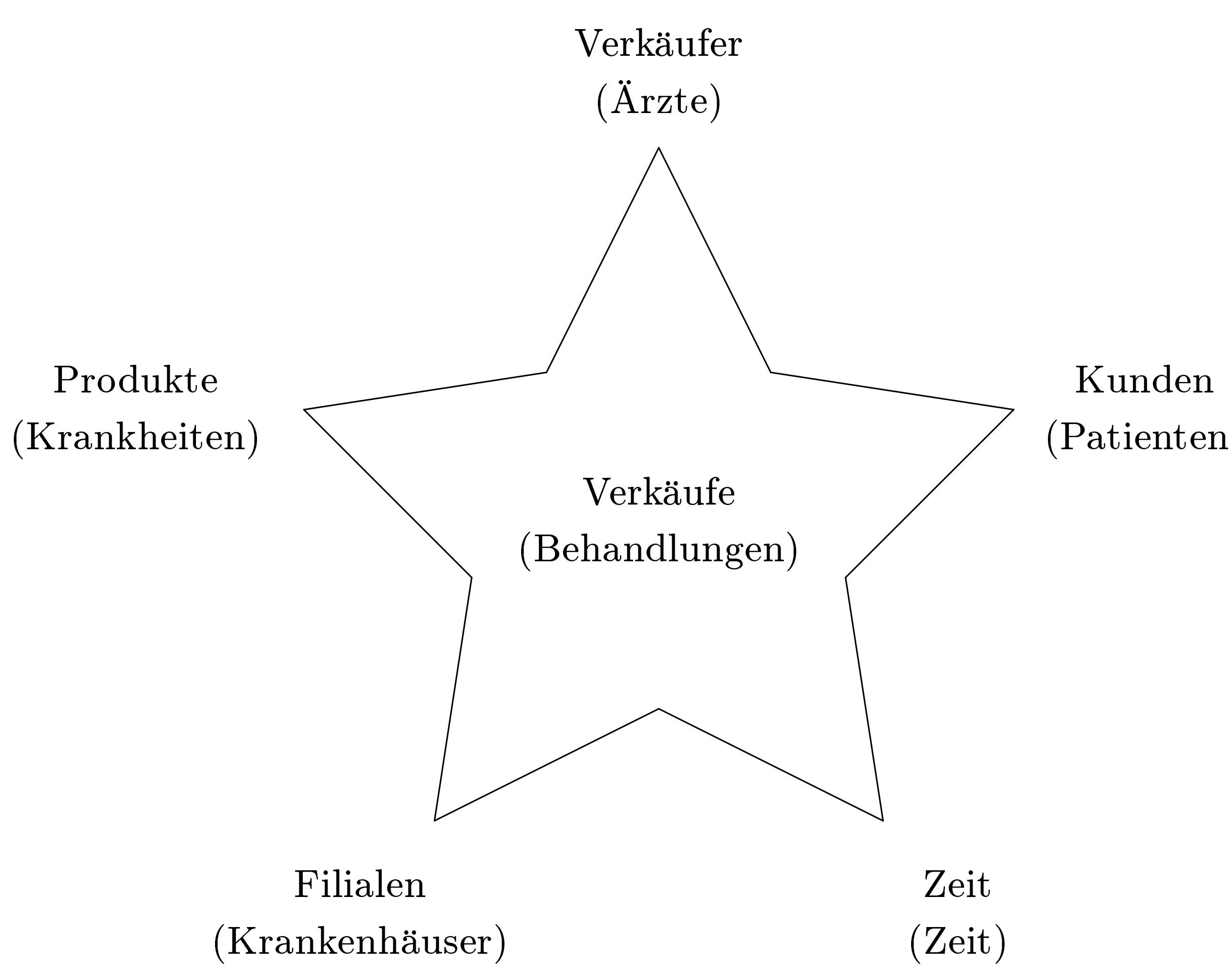
- Wie hat sich die Auslastung der Transatlantikflüge über die letzten zwei Jahre entwickelt? oder
- Wie haben sich besondere offensive Marketingstrategien für bestimmte Produktlinien auf die Verkaufszahlen ausgewirkt?

Stellung des Data Warehouse's



Datenbankentwurf für das Data Warehouse

- Sternschema
  - Faktentabelle (Verkäufe bzw. Behandlungen)
  - Dimensionstabellen



- Schneeflockenschema (snow flake schema)
  - entsteht wenn man die Dimensionstabellen normalisiert

Ausprägung eines Data Warehouses für Verkaufszahlen

Verkäufe					
VerkDatum	Filiale	Produkt	Anzahl	Kunde	Verkäufer
30-Jul-96	Passau	1347	1	4711	825
...	...	...	...	...	...

Filialen			
Filialenkennung	Land	Bezirk	...
Passau	D	Bayern	...
...	...	...	...

Kunden			
KundenNr	Name	wiealt	...
4711	Kemper	38	...
...	...	...	...

Verkäufer					
VerkäuferNr	Name	Fachgebiet	Manager	wiealt	...
825	Handyman	Elektronik	119	23	...
...	...	...	...	...	...

Zeit								
Datum	Tag	Monat	Jahr	Quartal	KW	Wochentag	Saison	...
...	...	...	...	...	...	...	...	...
30-Jul-96	30	Juli	1996	3	31	Dienstag	Hochsommer	...
...	...	...	...	...	...	...	...	...
23-Dec-97	27	Dezember	1997	4	52	Dienstag	Weihnachten	...
...	...	...	...	...	...	...	...	...

Produkte					
ProduktNr	Produkttyp	Produktgruppe	Produkthauptgruppe	Hersteller	...
1347	Handy	Mobiltelekom	Telekom	Siemens	...
...	...	...	...	...	...

Anfragen im Sternschema: Star Join

- Einschränkung der Dimensionen
- sternförmiger Join der (relevanten) Dimensionstabellen mit der Faktentabelle
- Verdichtung der Kennzahlen durch Gruppierung und Aggregation
- Beispielanfrage: Welche Handy-Hersteller wurden von jungen Kunden in Bayern zu Weihnachen 1996 favorisiert?

```
select sum(v.Anzahl), p.Hersteller
from Verkäufe v, Filialen f, Produkte p, Zeit z, Kunden k
where z.Saison = 'Weihnachten' and z.Jahr = 1996 and k.wiealt < 30 and
      p.Produkttyp = 'Handy' and f.Bezirk = 'Bayern' and
      v.VerkDatum = z.Datum and v.Produkt = p.ProduktNr and
      v.Filiale = f.FilialenKennung and v.Kunde = k.KundenNr
group by p.Hersteller;
```

Roll-Up/Drill-Down-Anfragen

- Roll-Up: Elimination eines Attributs aus der **group by**-Klausel
- Drill-Down: Hinzufügen eines Attributs in die **group by**-Klausel

select Jahr, Hersteller, **sum**(Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr **and** v.VerkDatum = z.Datum
  and p.Produkttyp = 'Handy'
group by p.Hersteller, z.Jahr;

select Jahr, **sum**(Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr **and** v.VerkDatum = z.Datum
  and p.Produkttyp = 'Handy'
group by z.Jahr;

select **sum**(Anzahl)
from Verkäufe v, Produkte p
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy';

Ergebnisse der Anfragen

Handyverkäufe nach Hersteller und Jahr		
Hersteller	Jahr	Anzahl
Siemens	1994	2.000
Siemens	1995	3.000
Siemens	1996	3.500
Motorola	1994	1.000
Motorola	1995	1.000
Motorola	1996	1.500
Bosch	1994	500
Bosch	1995	1.000
Bosch	1996	1.500
Nokia	1995	1.000
Nokia	1996	1.500
Nokia	1996	2.000

Handyverkäufe nach Jahr	
Jahr	Anzahl
1994	4.500
1995	6.500
1996	8.500

Handyverkäufe nach Hersteller	
Hersteller	Anzahl
Siemens	8.500
Motorola	3.500
Bosch	3.000
Nokia	4.500

Handyverkäufe
Anzahl
19.500

Crosstab-Darstellung

Hersteller \ Jahr	1994	1995	1996	Σ
Siemens	2.000	3.000	3.500	8.500
Motorola	1.000	1.000	1.500	3.500
Bosch	500	1.000	1.500	3.000
Nokia	1.000	1.500	2.000	4.500
Σ	4.500	6.500	8.500	19.500

create table Handy2DCube
(Hersteller varchar(20), Jahr integer, Anzahl integer);

insert into Handy2DCube
(**select** p.Hersteller, z.Jahr, **sum**(v.Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy'
and v.VerkDatum = z.Datum
group by z.Jahr, p.Hersteller)
union
(**select** p.Hersteller, to_number(**null**), **sum**(v.Anzahl)
from Verkäufe v, Produkte p
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy'
group by p.Hersteller)
union
(**select** **null**, z.Jahr, **sum**(v.Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy'
and v.VerkDatum = z.Datum
group by z.Jahr)
union
(**select** **null**, to_number(**null**), **sum**(v.Anzahl)
from Verkäufe v, Produkte p
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy');

Ausprägung der Relationen

Handy2DCube		
Hersteller	Jahr	Anzahl
Siemens	1994	2.000
Siemens	1995	3.000
Siemens	1996	3.500
Motorola	1994	1.000
Motorola	1995	1.000
Motorola	1996	1.500
Bosch	1994	500
Bosch	1995	1.000
Bosch	1996	1.500
Nokia	1995	1.000
Nokia	1996	1.500
Nokia	1996	2.000
null	1994	4.500
null	1995	6.500
null	1996	8.500
Siemens	**null**	8.500
Motorola	**null**	3.500
Bosch	**null**	3.000
Nokia	**null**	4.500
null	**null**	19.500

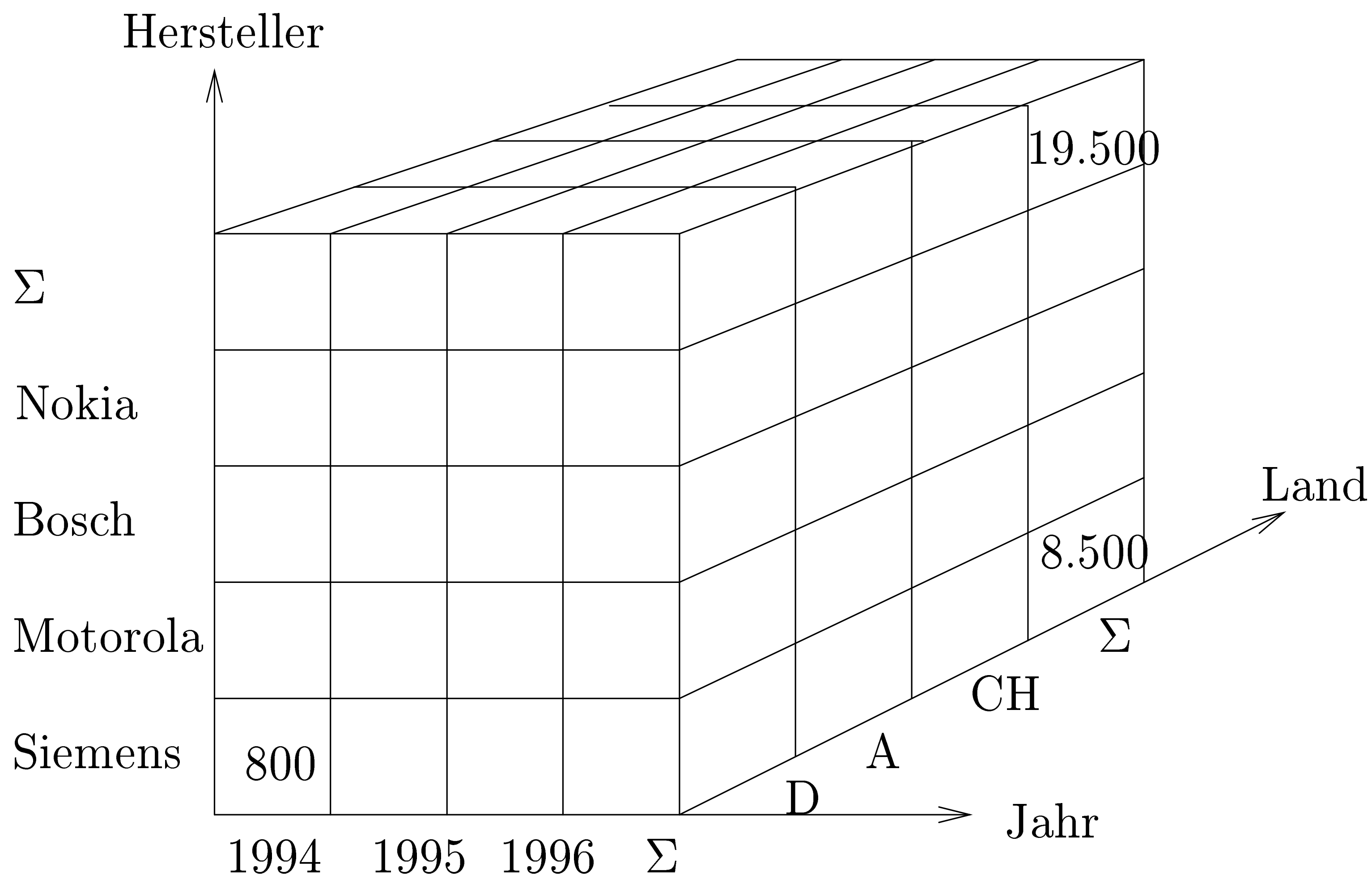
Handy3DCube			
Hersteller	Jahr	Land	Anzahl
Siemens	1994	D	800
Siemens	1994	A	600
Siemens	1994	CH	600
Siemens	1995	D	1.200
Siemens	1995	A	800
Siemens	1995	CH	1.000
Siemens	1996	D	1.400
...	...	...	...
Motorola	1994	D	400
Motorola	1994	A	300
Motorola	1994	CH	300
...	...	...	...
Bosch	...	...	...
...	...	...	...
null	1994	D	...
null	1995	D	...
...	...	...	...
Siemens	**null**	**null**	8.500
...	...	...	...
null	**null**	**null**	19.500

Der **cube**-Operator

select p.Hersteller, z.Jahr, f.Land, **sum**(v.Anzahl)
from Verkäufe v, Produkte p, Zeit z, Filialen f
where v.Produkt = p.ProduktNr **and** p.Produkttyp = 'Handy'
    and v.VerkDatum = z.Datum **and** v.Filiale = f.Filialenkennung
group by z.Jahr, p.Hersteller, f.Land **with cube**;

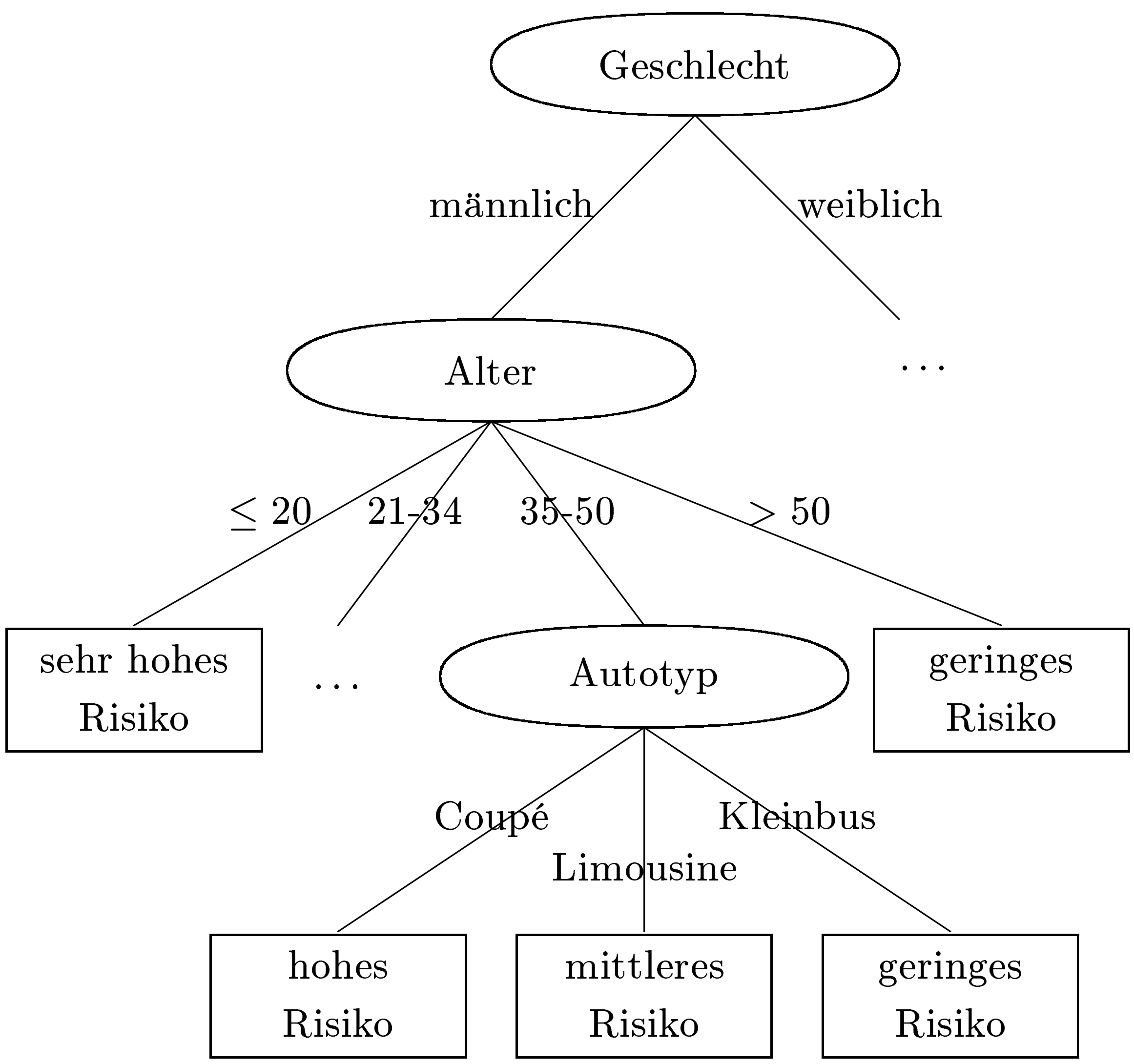
- Man erspart sich die Formulierung von 2^n (bei n Attributen in der **group by**-Klausel) **union**-Anfragen
- Die Aggregation geht sehr viel effizienter, da Zwischenergebnisse wiederverwendet werden.

Schematische Darstellung als Datenwürfel/Data Cube



1. *ROLAP*: Das Data Warehouse-System wird auf der Basis eines relationalen Datenmodells realisiert
2. *MOLAP*: Unter multi-dimensionalen OLAP-Systemen versteht man solche, die die Daten nicht in relationaler Form abspeichern sondern in speziellen mehr-dimensionalen Datenstrukturen

Klassifikation von Objekten: Risikoabschätzung



Assoziationsregeln

Wenn jemand einen PC kauft **dann** kauft er/sie auch einen Drucker

1. *Confidence*:
   - Dieser Wert legt fest, bei welchem Prozentsatz der Datenmenge, bei der die Voraussetzung (linke Seite) erfüllt ist, die Regel (rechte Seite) auch erfüllt ist.
   - Eine *Confidence* von 80% für unsere Beispielregel sagt aus, daß vier Fünftel der Leute, die einen PC gekauft haben, auch einen Drucker dazu gekauft haben.
2. *Support*:
   - Dieser Wert legt fest, wieviele Datensätze überhaupt gefunden wurden, um die Gültigkeit der Regel zu verifizieren.
   - Bei einem *Support* von 1% wäre also jeder Hunderste Verkauf ein PC zusammen mit einem Drucker.