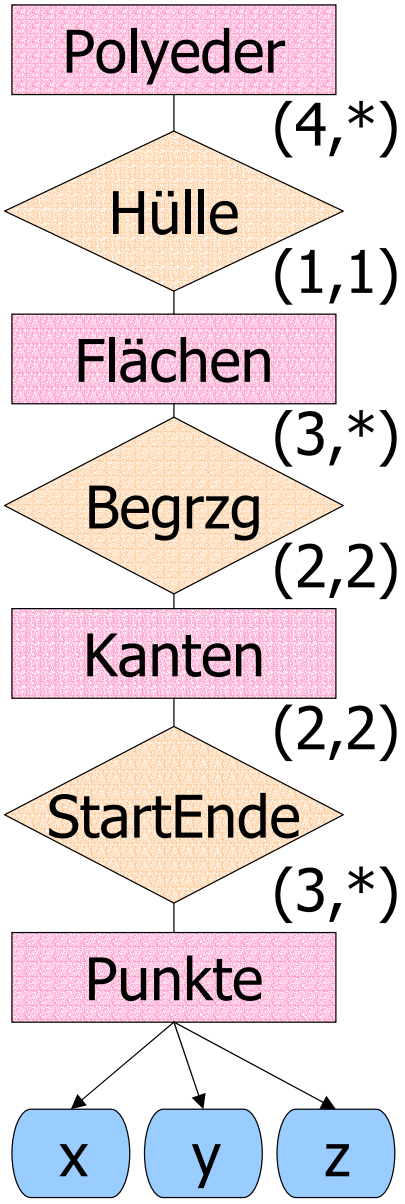
Objektorientierte Datenbanken

- 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
  - „Nischen-Dasein“
  - Konzepte wurden in Relationalen Datenbanken übernommen
  - Objekt-Relationale Dtaenbanken
- seit 1993 erster Standard (ODMG)

Nachteile relationaler Modellierung



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

Flächen		
FlächenID	PolyID	Oberfläche
f1	cubo#5	. . .
f2	cubo#2	. . .
. . .	. . .	. . .
f6	cubo#5	. . .
f7	tetra#7	. . .

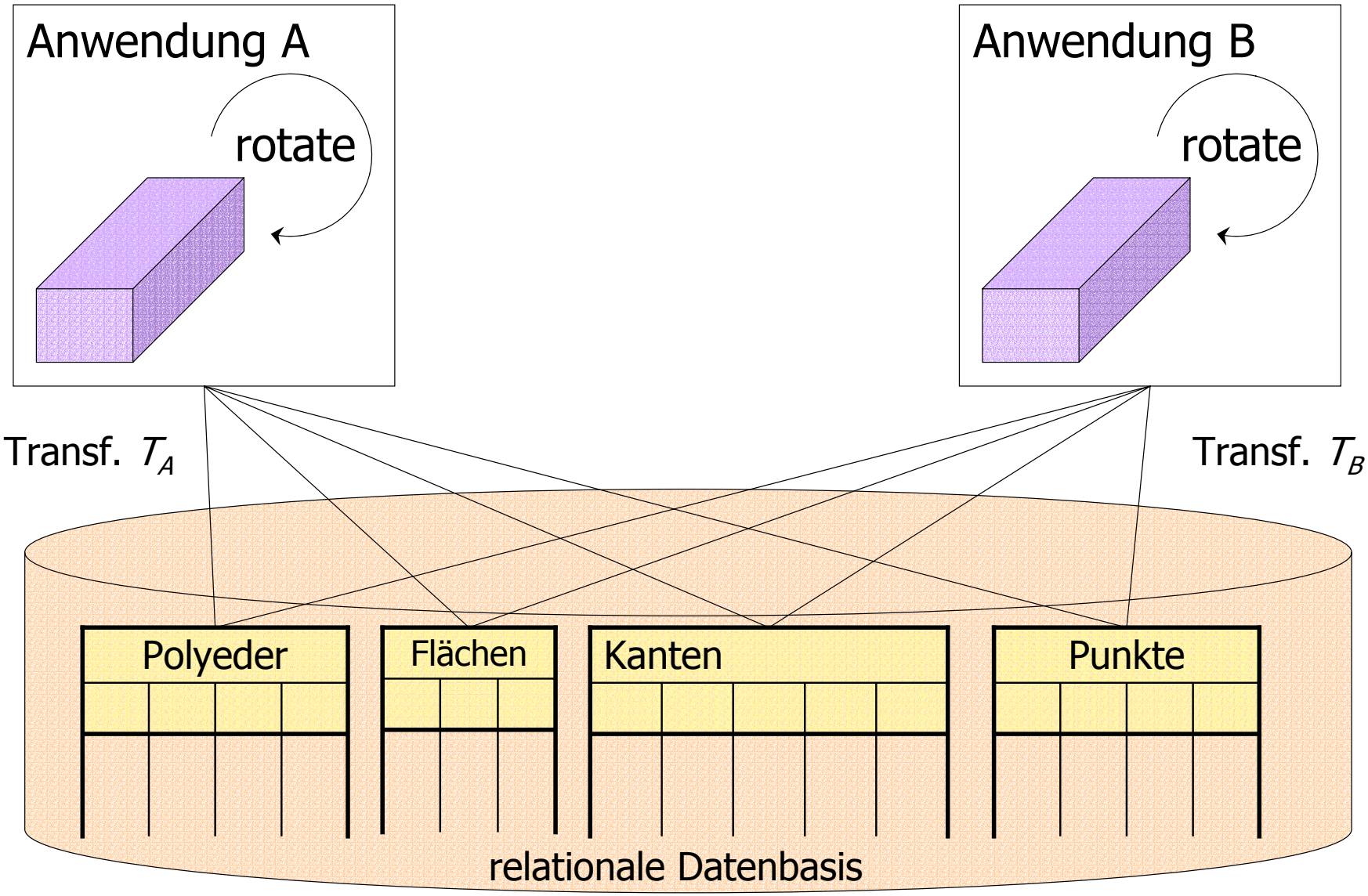
Kanten				
KantenID	F1	F2	P1	P2
k1	f1	f4	p1	p4
k2	f1	f2	p2	p3
. . .	. . .	. . .		

Kanten			
PunktID	X	Y	Z
p1	0.0	0.0	0.0
p2	1.0	0.0	0.0
. . .	. . .	. . .	

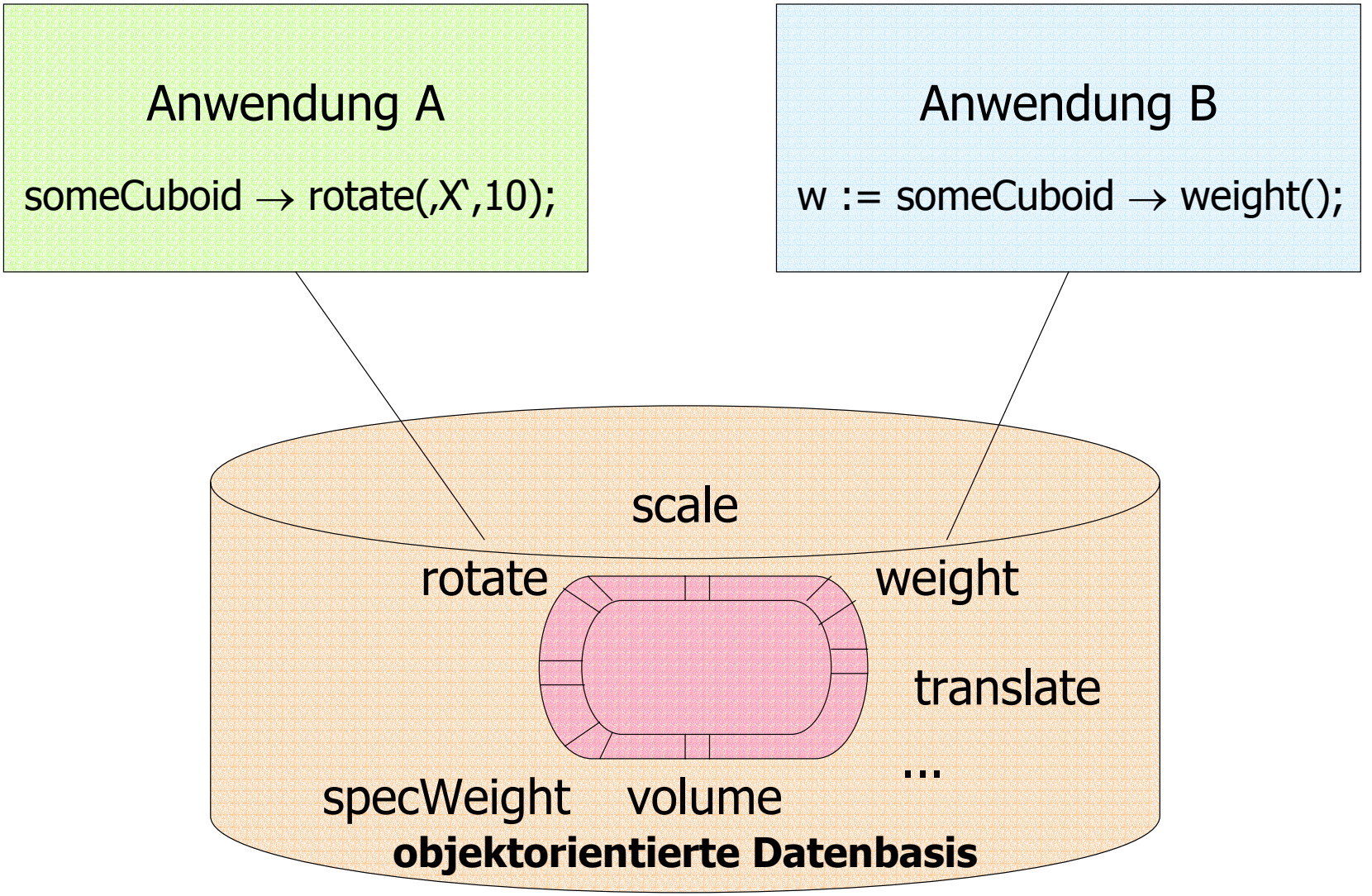
Nachteile relationaler Modellierung

- **Segmentierung**
- **Künstliche Schlüsselattribute**
- **Fehlendes Verhalten**
- **Externe Programmierschnittstelle**

Visualisierung des „Impedance Mismatch“



Vorteile objektorientierter Datenmodellierung



Vorteile objektorientierter Datenmodellierung

- „information hiding“/Objektkapselung
- Wiederverwendbarkeit
- Operationen direkt in Sprache des Objektmodells realisiert (kein Impedance Mismatch)

ODMG-Standardisierung

Beteiligte

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

Reviewer

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

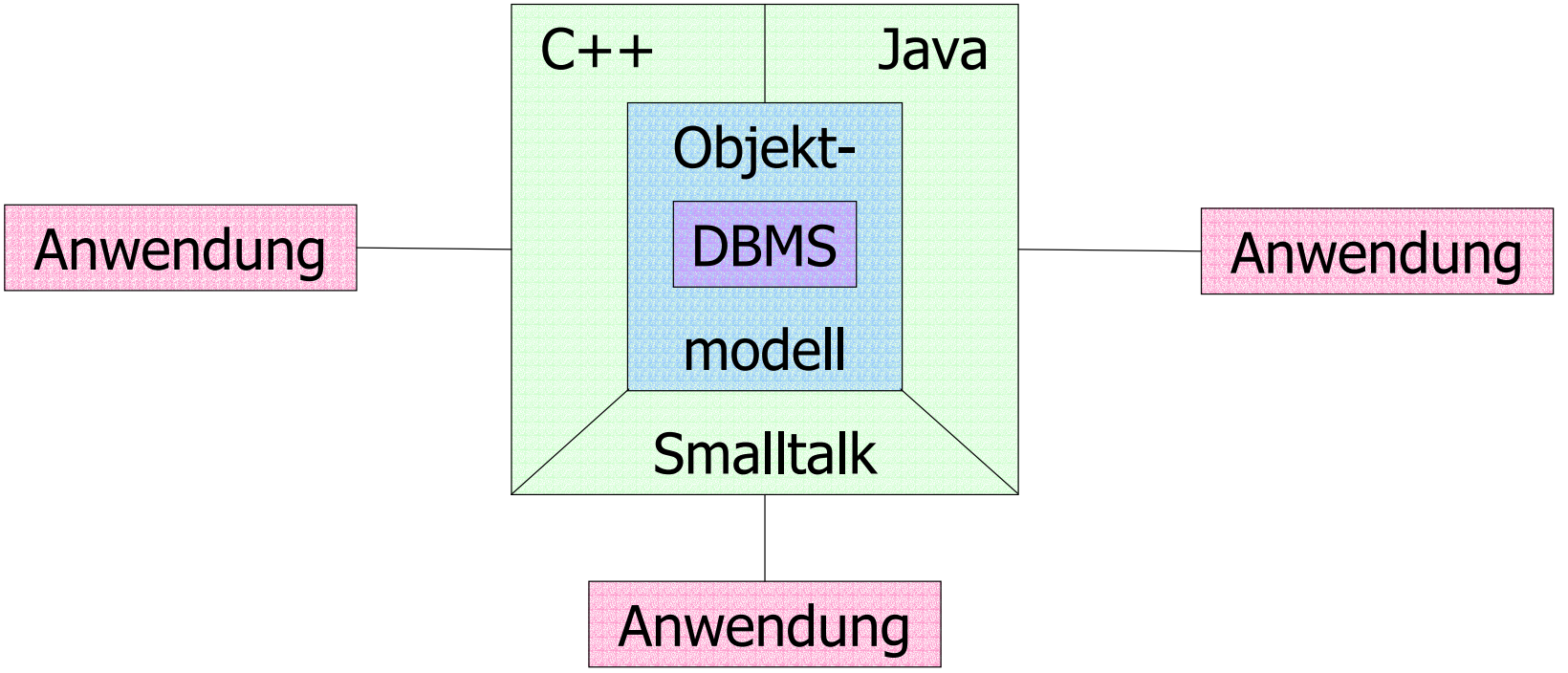
Bestandteile des Standards

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

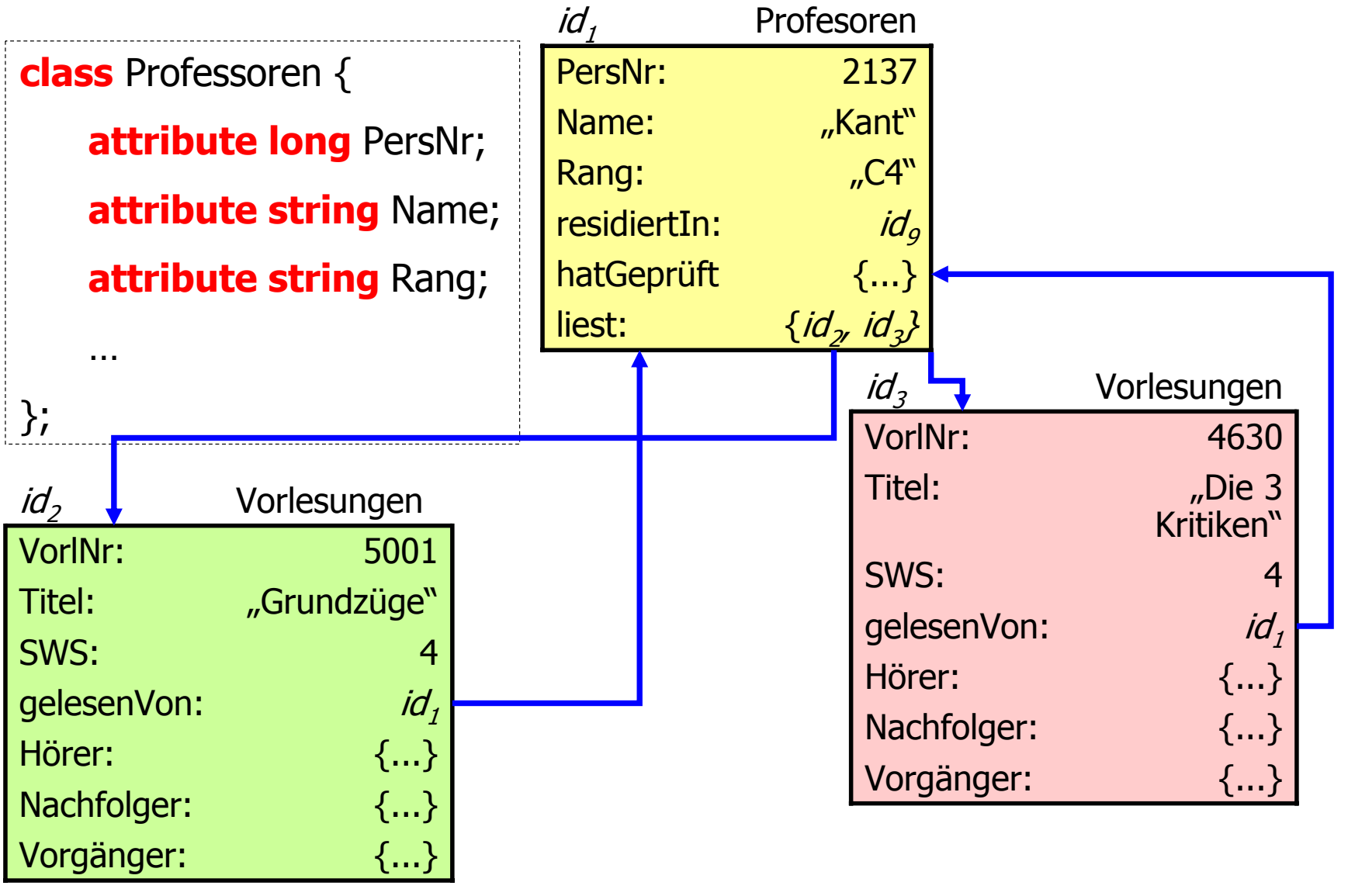
Motivation der Standardisierung

- Portabilitäts-Standard
- kein Interoperabilitäts-Standard

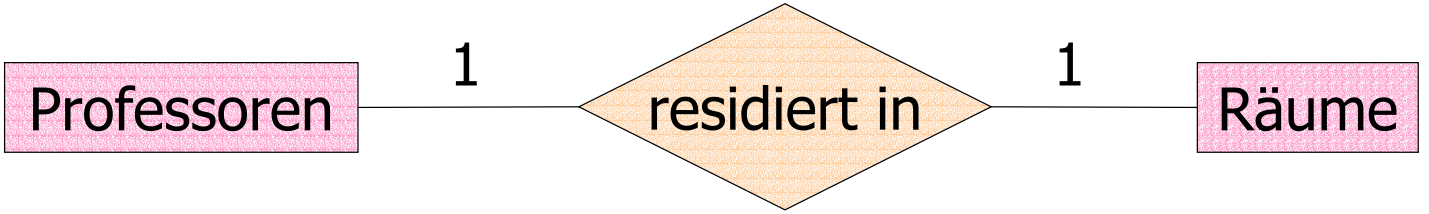
Integration des ODMG-Objektmodells



Einige Objekte aus der Universitätswelt

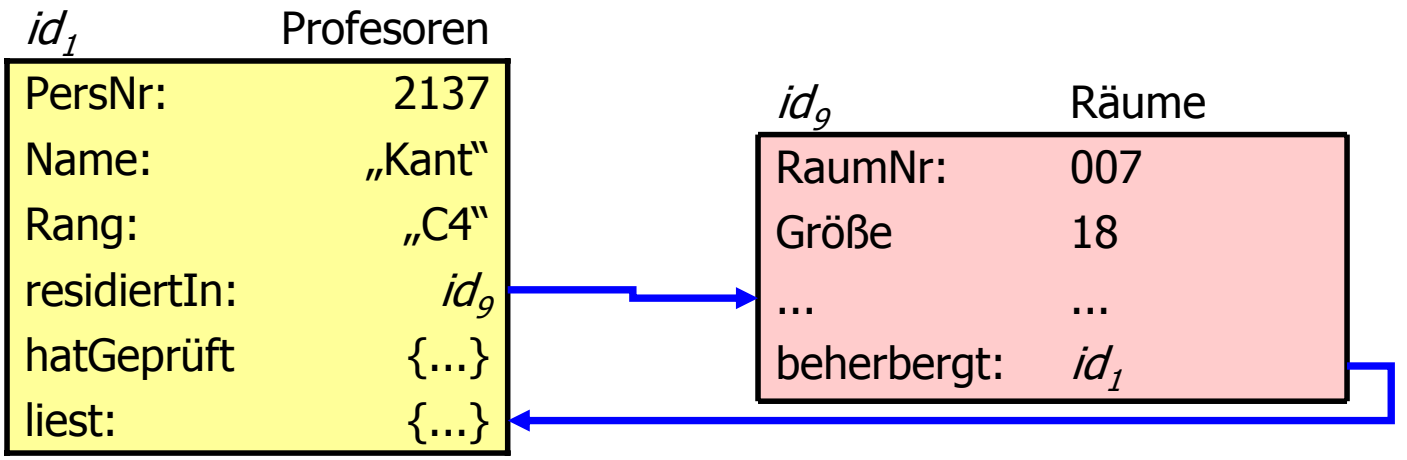


1 : 1-Beziehungen

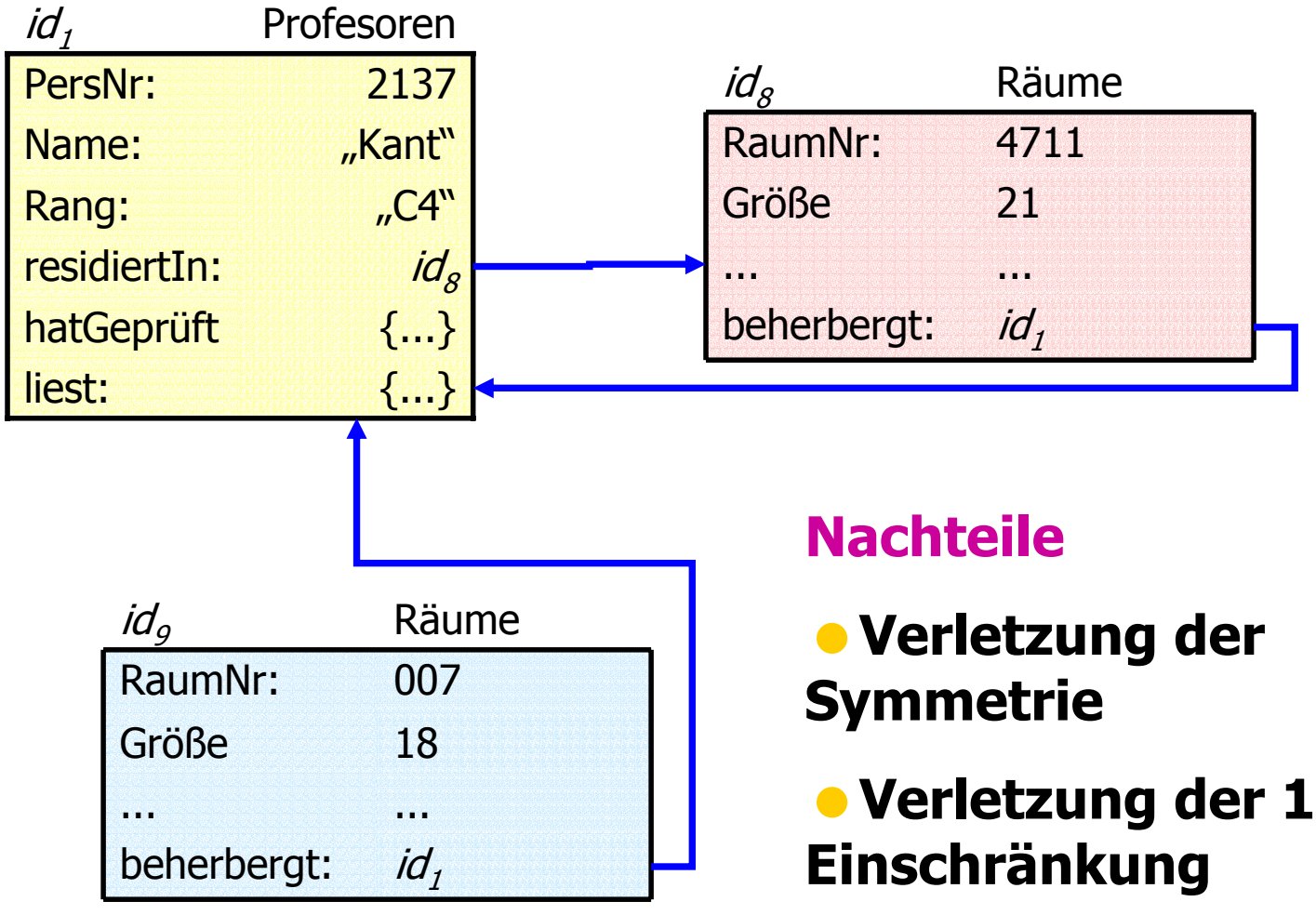



```
class Professoren {
    attribute long PersNr;
    ...
    relationship Räume resisdiertIn;
};
class Räume {
    attribute long RaumNr;
    attribute short Größe;
    ...
    relationship Professoren beherbergt;
};
```

Beispielausprägungen



Beispielausprägungen



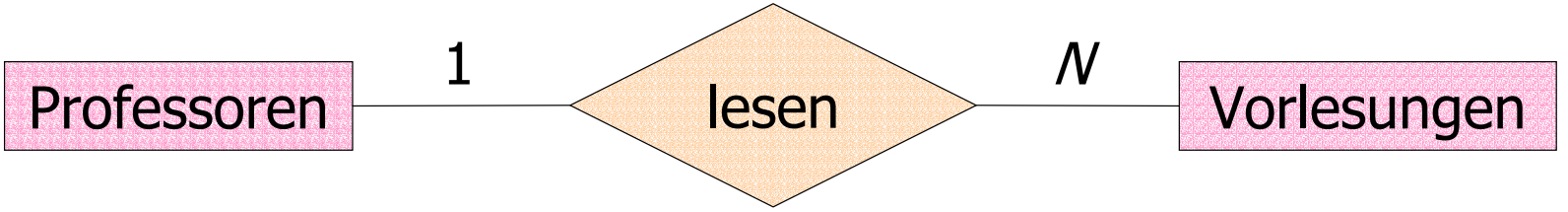
Nachteile

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

Bessere Modellierung mit „inverse“

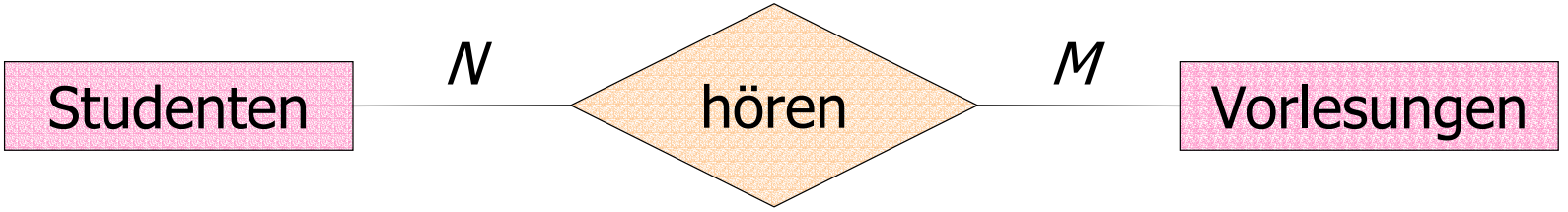
```
class Professoren {
    attribute long PersNr;
    ...
    relationship Räume residiertIn inverse Räume::beherbergt;
};
class Räume {
    attribute long RaumNr;
    attribute short Größe;
    ...
    relationship Professoren beherbergt inverse Professoren::residiertIn
};
```

1 : *N*-Beziehungen



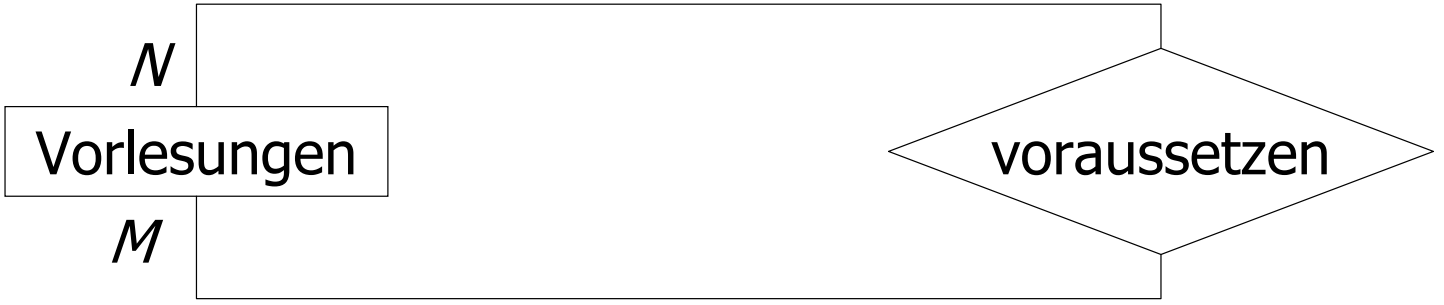

```
class Professoren {
    ...
    relationship set(Vorlesungen) liest inverse Vorlesungen::gelesenVon;
};
class Vorlesungen {
    ...
    relationship Professoren gelesenVon inverse Professoren::liest;
};
```

N : *M*-Beziehungen



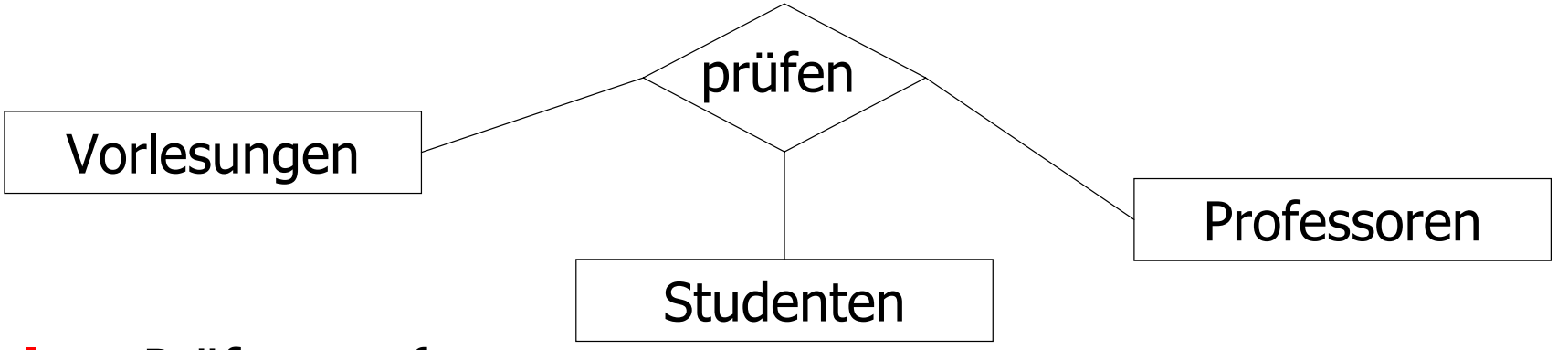

```
class Studenten {
  ...
  relationship set(Vorlesungen) hört inverse Vorlesungen::Hörer;
};
class Vorlesungen {
  ...
  relationship set(Studenten) Hörer inverse Studenten::hört;
};
```

Rekursive *N* : *M*-Beziehungen




```
class Vorlesungen {
  ...
  relationship set(Vorlesungen) Vorgänger inverse Vorlesungen::Nachfolger;
  relationship set(Vorlesungen) Nachfolger inverse Vorlesungen::Vorgänger;
};
```

Ternäre Beziehungen

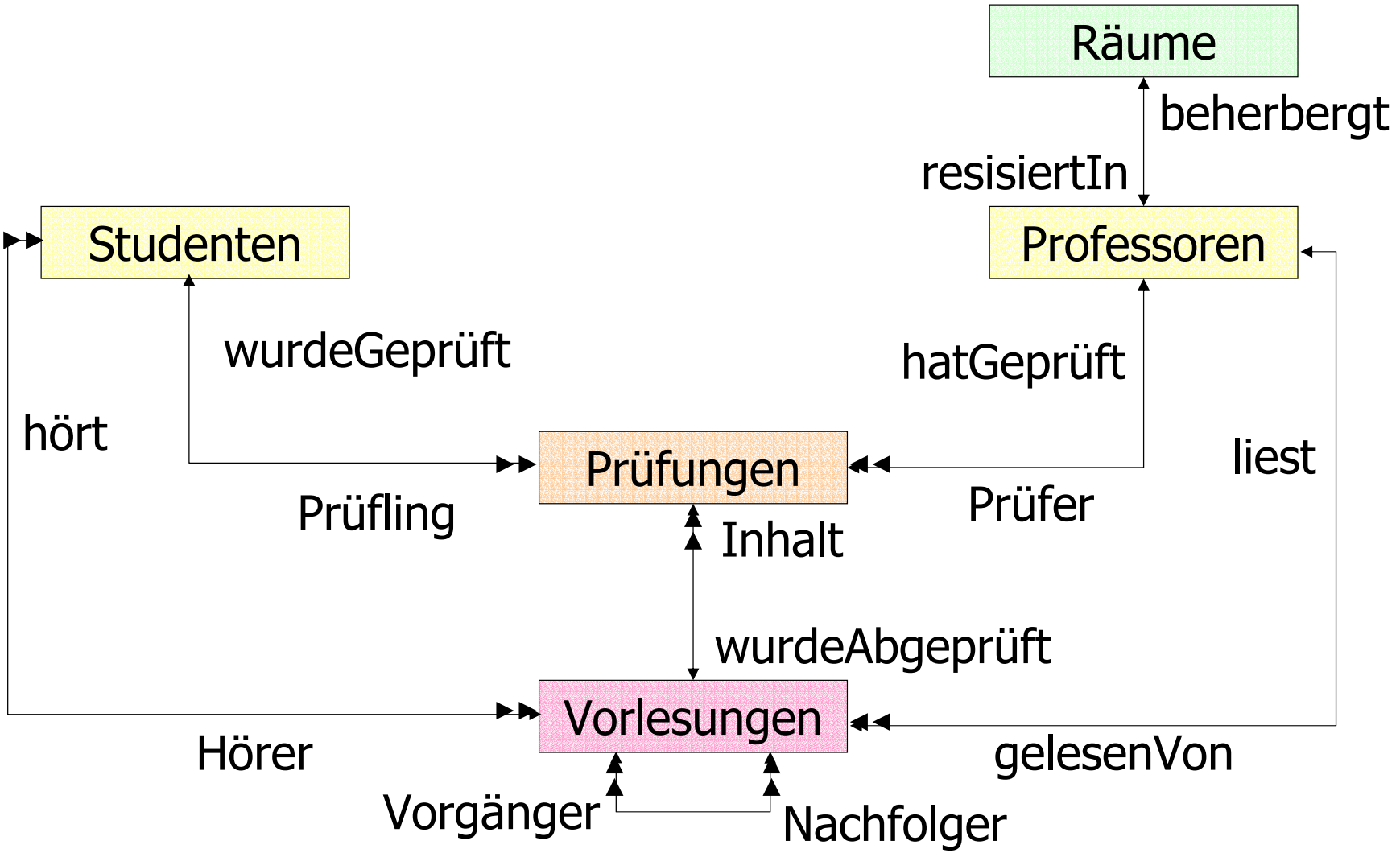



```
class Prüfungen {
    attribute struct Datum
        { short Tag; short Monat; short Jahr} Prüfdatum;
    attribute float Note;
    relationship Professoren Prüfer inverse
        Professoren::hatGeprüft;
    relationship Studenten Prüfling inverse
        Studenten::wurdeGeprüft;
    relationship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

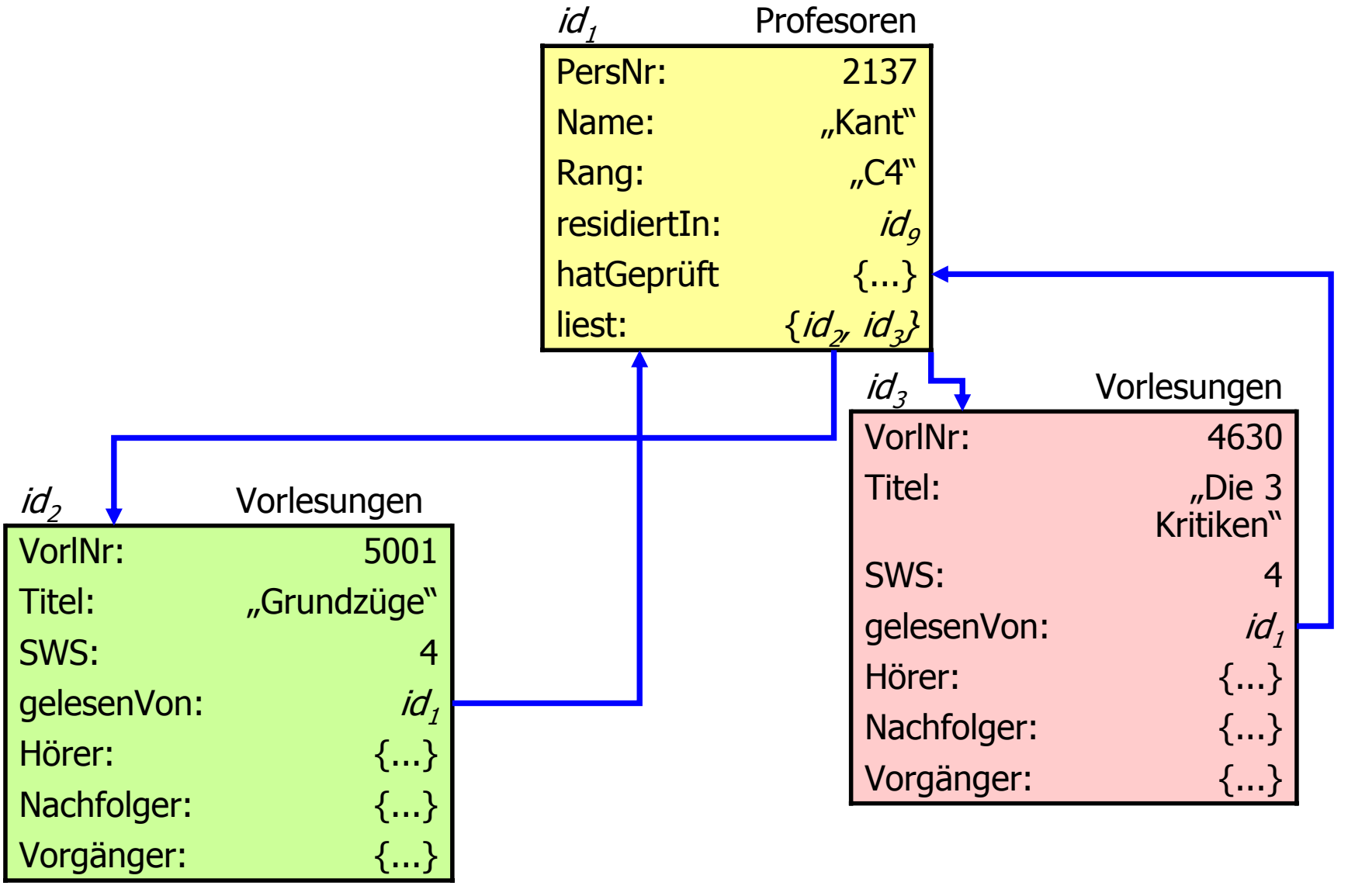
Vervollständigtes Universitäts-Schema

```
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

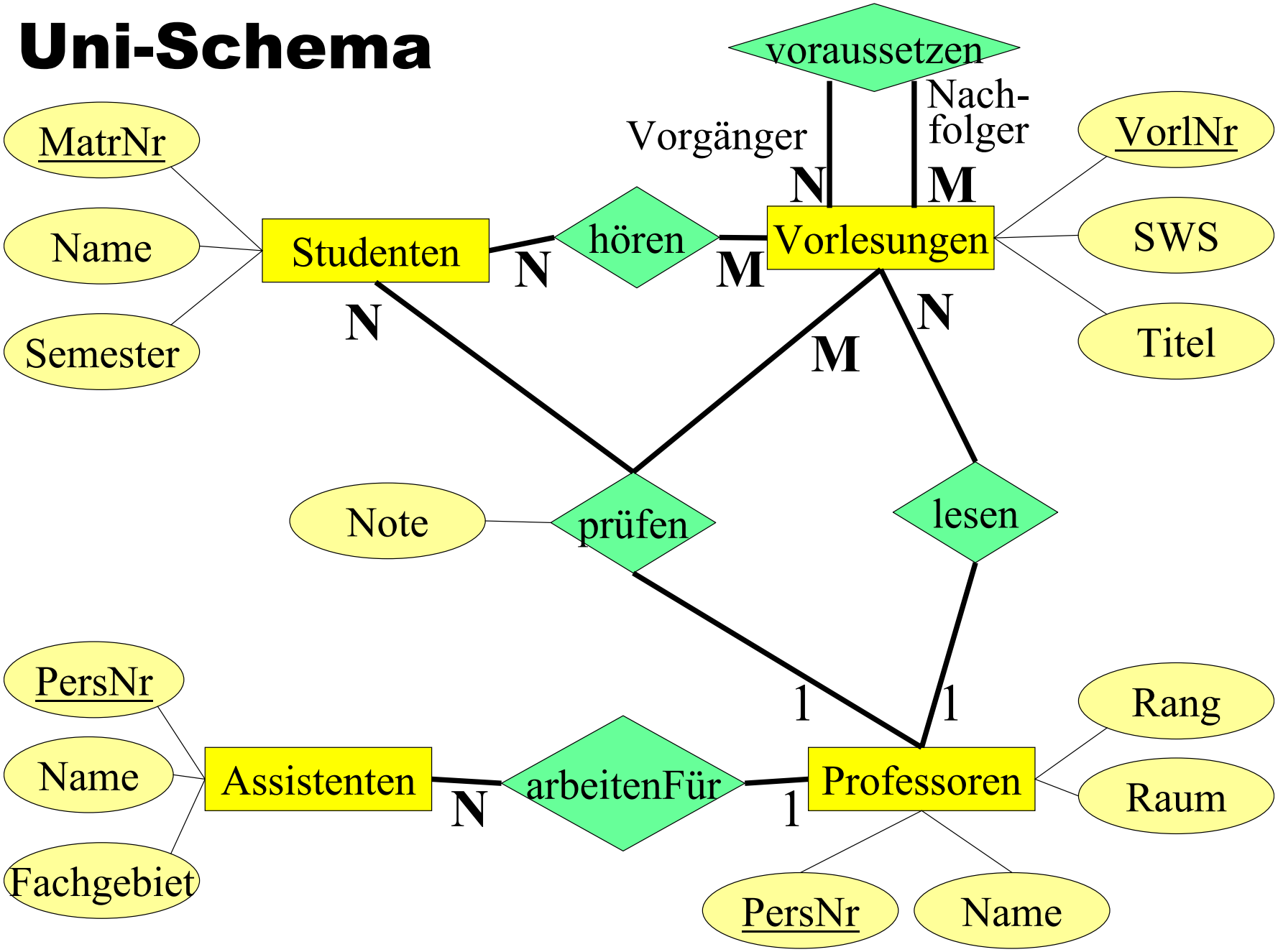


Einige Objekte aus der Universitätswelt



Eigenschaften von Objekten

- **Im objektorientierten Modell hat ein Objekt drei Bestandteile:**
  - **Identität**
  - **Typ**
  - **Wert/Zustand**


Uni-Schema
voraussetzen
Vorgänger
Nach-
folger
N
M
MatrNr
Name
Semester
Studenten
N
hören
M
Vorlesungen
VorlNr
SWS
Titel
N
M
N
Note
prüfen
lesen
1
1
PersNr
Name
Fachgebiet
Assistenten
N
arbeitenFür
1
Professoren
Rang
Raum
PersNr
Name

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	gelesen Von
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

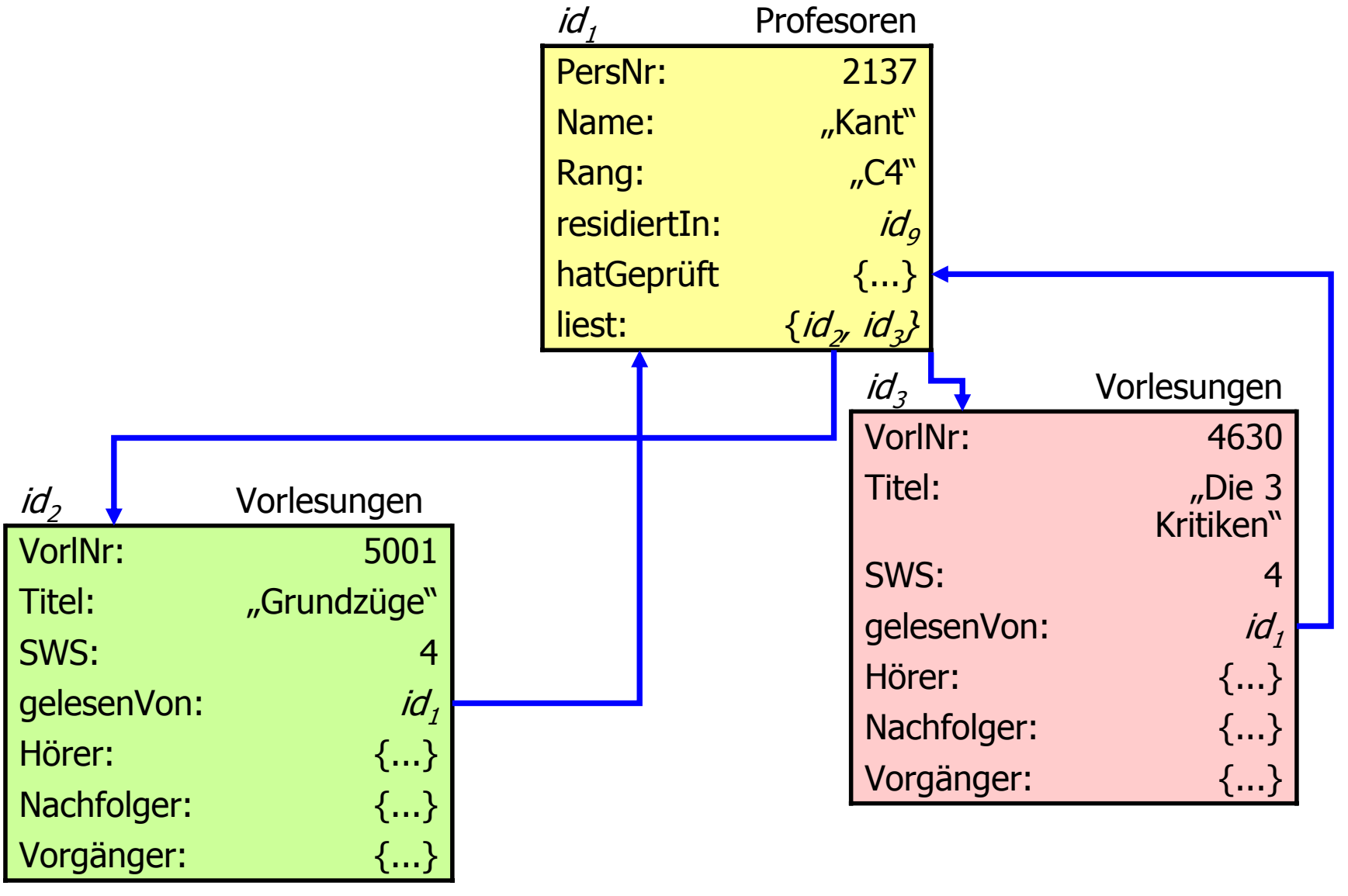
prüfen

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

Assistenten

PerslNr	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	2126

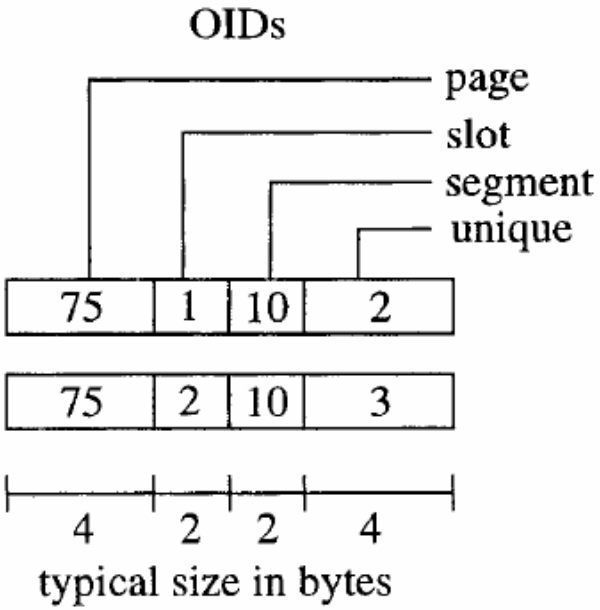
Einige Objekte aus der Universitätswelt

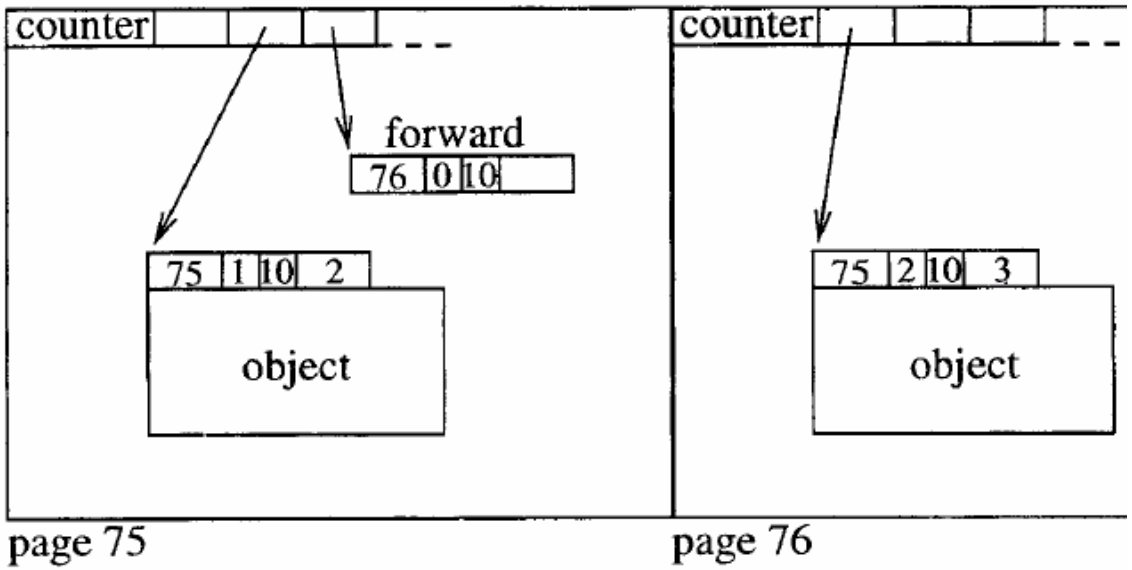


Objekt-Identität

- Das wesentliche Charakteristikum objekt-orientierter Datenmodellierung
- Verweise werden über die OID realisiert
- Zwei Realisierungsformen
  - Physische OIDs
    - Enthalten den Speicherort des Objekts
    - Im wesentlichen entsprechen diese den Tupel IDentifikatoren (TIDs)
  - Logische OIDs
    - Unabhängig vom Speicherort der Objekte
    - D.h. Objekte können verschoben werden
    - Indirektion über eine „Mapping"-Struktur
      - B-Baum
      - Hash-Tabelle
      - Direct Mapping

Realisierung physischer OIDs





Drei Realisierungstechniken für logische OIDs

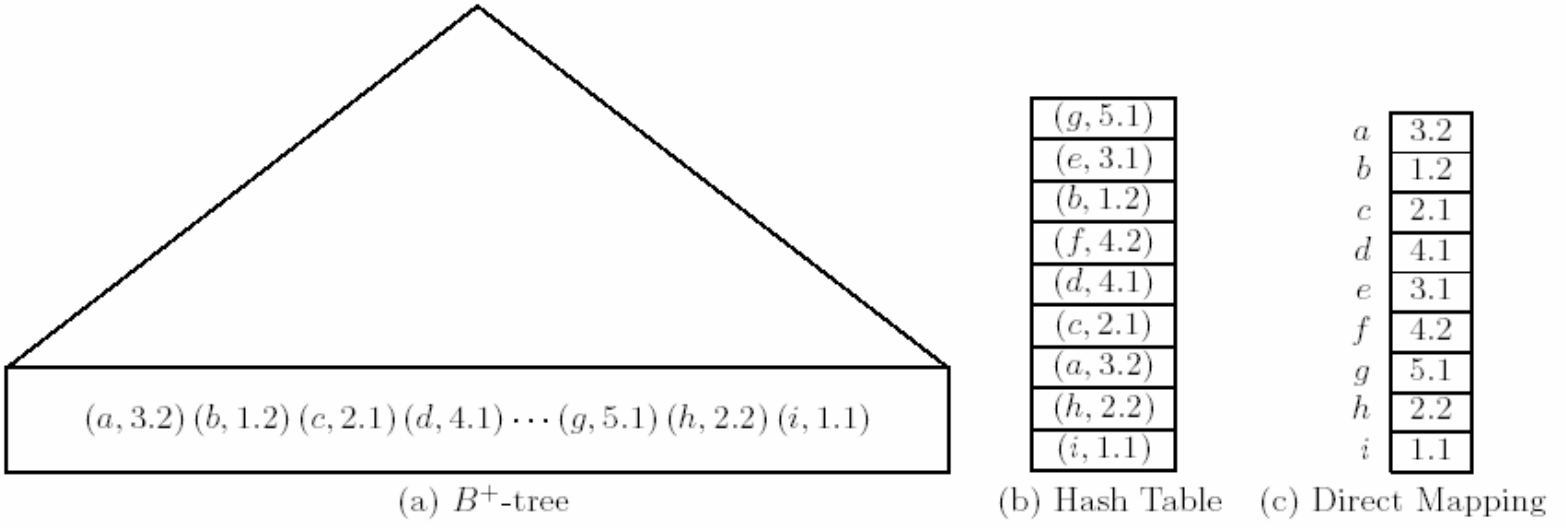


Fig. 2. Mapping techniques

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.

Modellierung des Verhaltens: Operationen II

Drei Klassen von Operationen:

- *Beobachter* (engl. *observer*)*:*
  - oft auch Funktionen genannt
  - Objektzustand „erfragen"
  - Beobachter-Operationen haben keinerlei objektändernde Seiteneffekte
- *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*.

Modellierung des Verhaltens: Operationen III

- *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 dem gegenüber 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*).

Klassen-Definition von Operationen in ODL

Beispiel-Operationen

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

Aufruf der Operationen

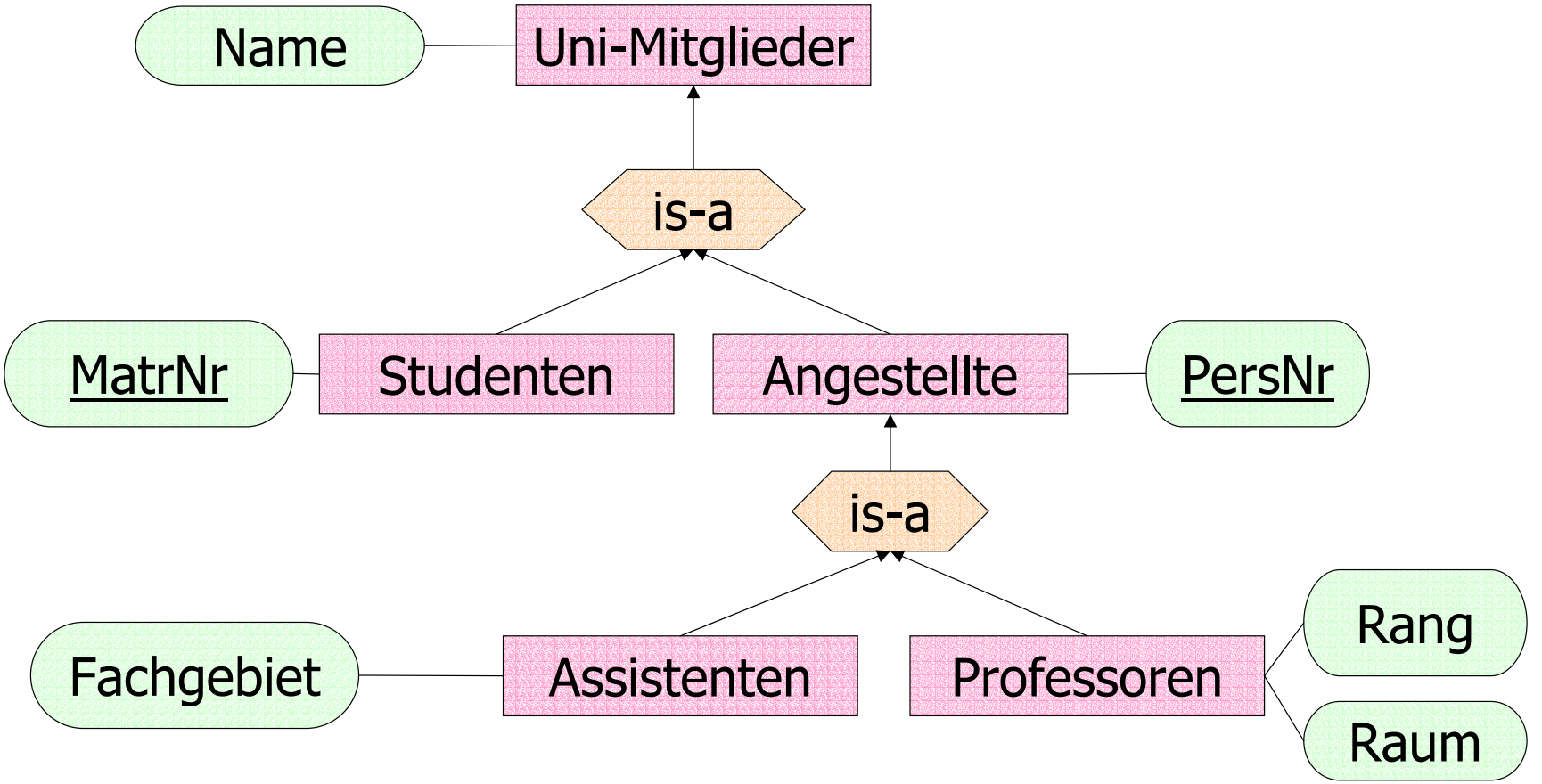
im Anwendungsprogramm:

```
meinLieblingsProf→befördert();
```

in OQL:

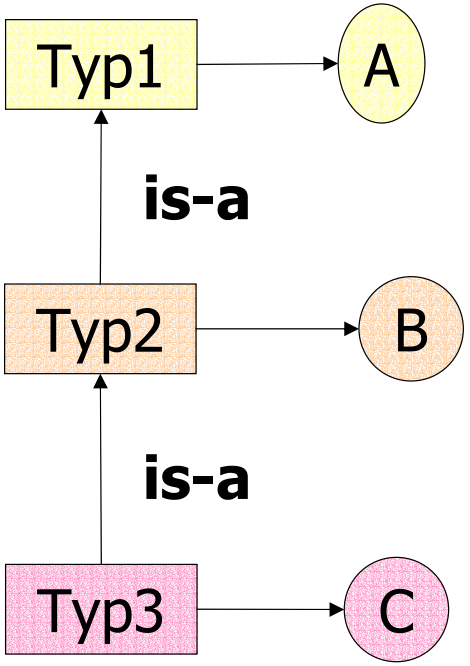
```
select p.wieHartAlsPrüfer()
    from p in AlleProfessoren
    where p.Name = „Curie";
```

Vererbung und Subtypisierung

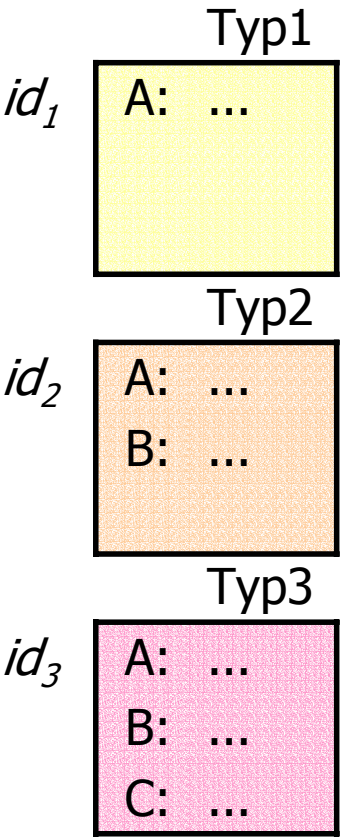


Terminologie

Objekttypen

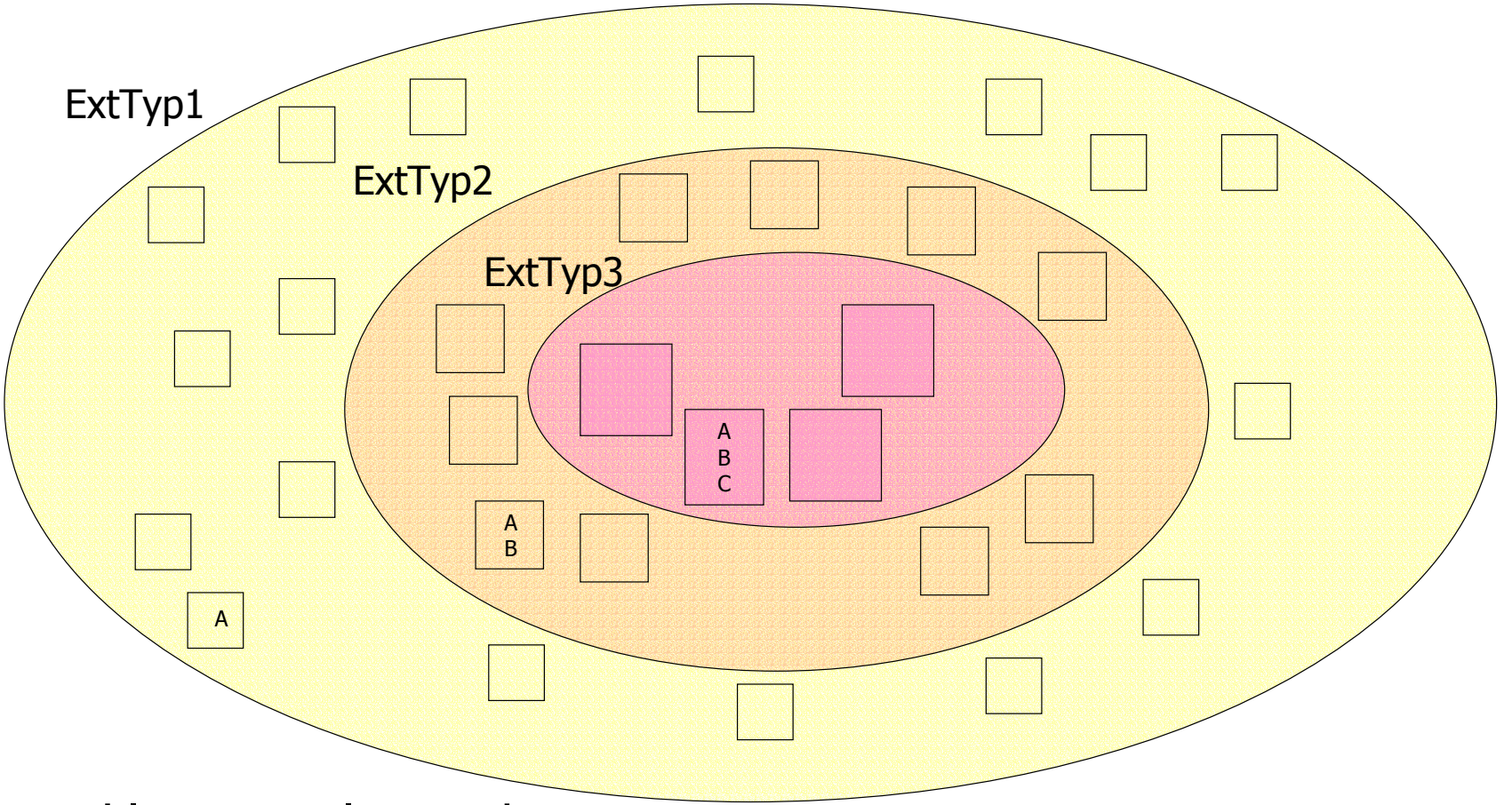


Instanzen



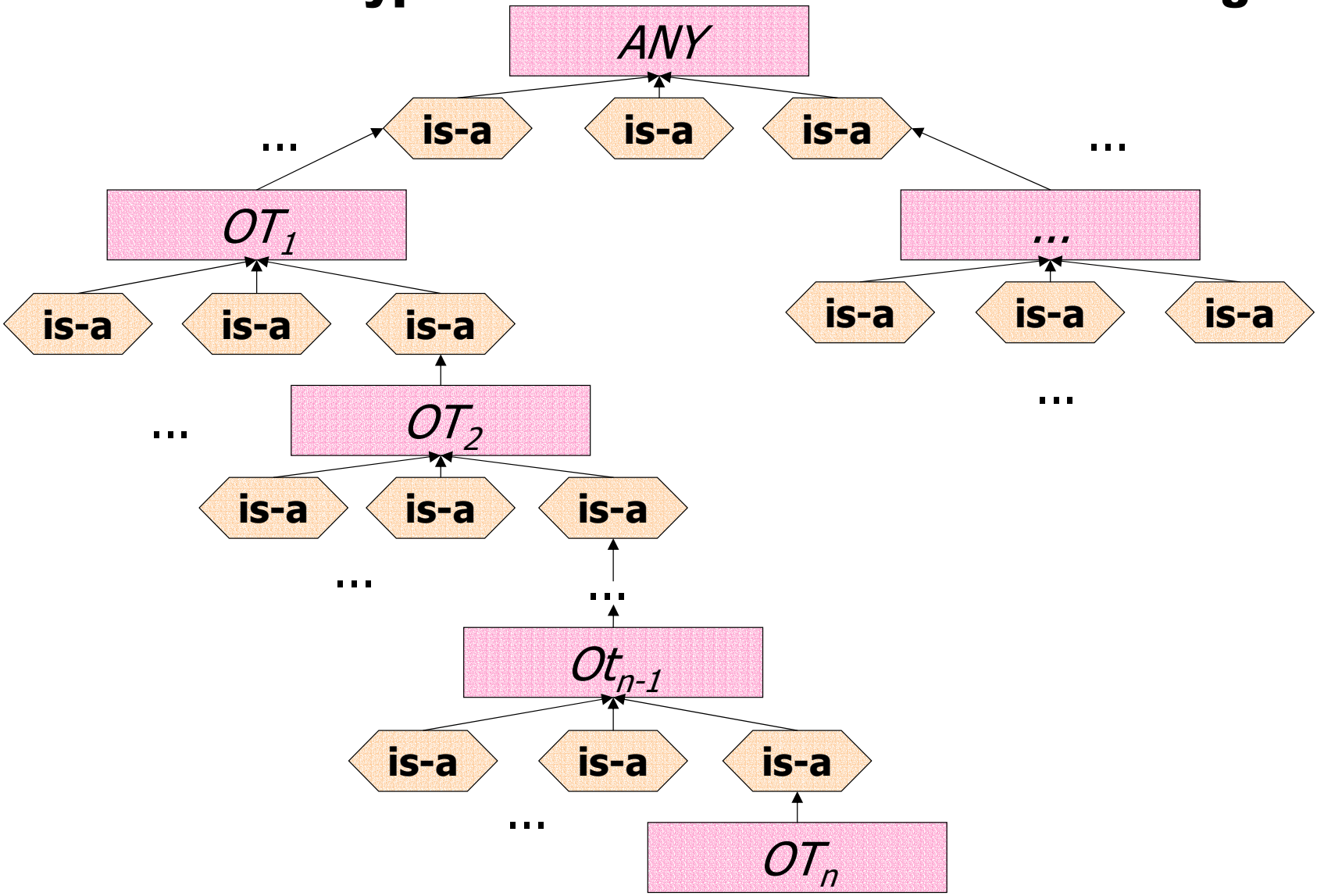
- Untertyp / Obertyp
- Instanz eines Untertyps gehört auch zur Extension des Obertyps
- Vererbung der Eigenschaften eines Obertyps an den Untertyp

Darstellung der Subtypisierung



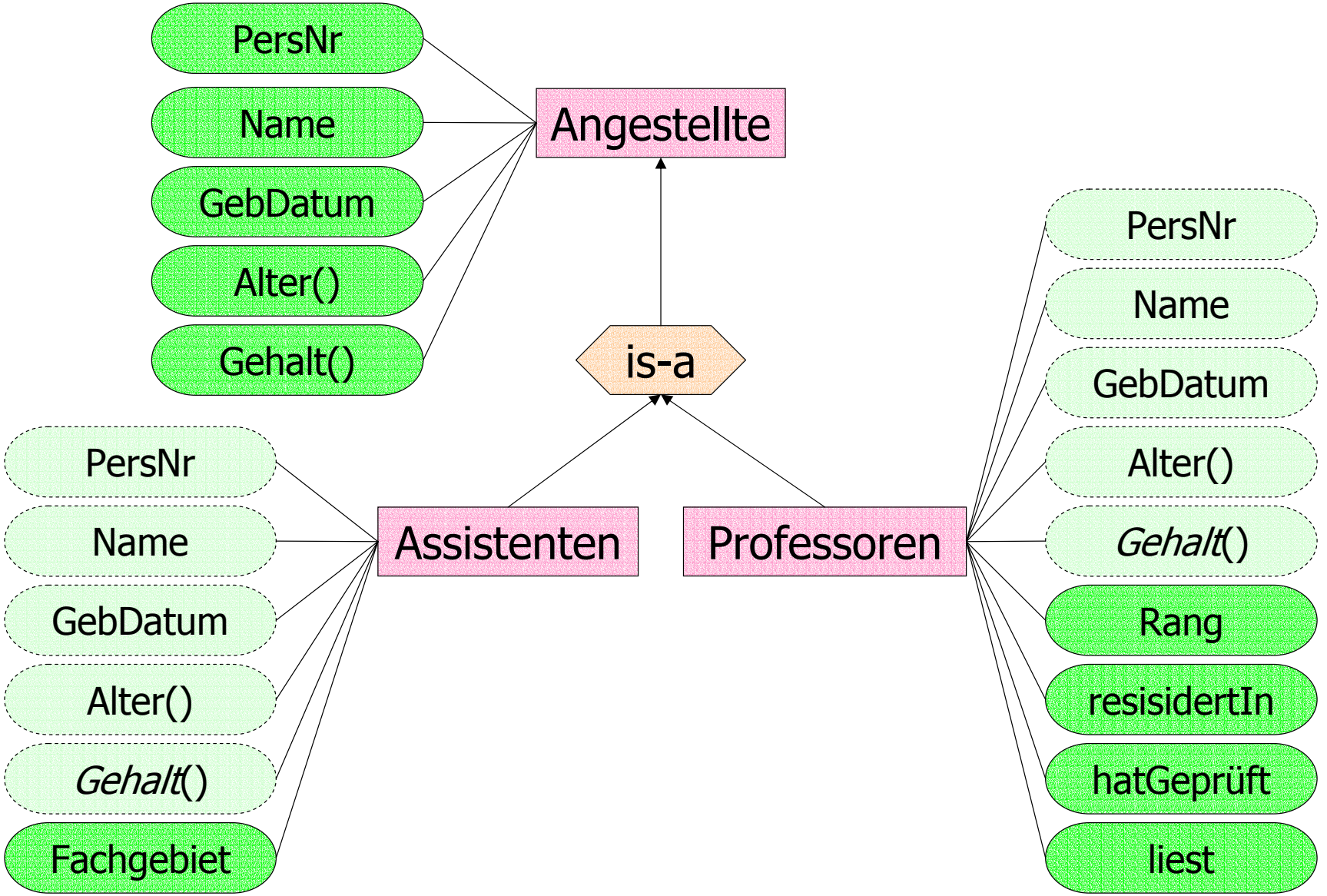
- Inklusionspolymorphismus
- Subtituierbarkeit
  - Eine Untertyp-Instanz ist überall dort einsetzbar, wo eine Obertyp-Instanz gefordert ist.

Abstrakte Typhierarchie bei Einfach-Vererbung



eindeutiger Pfad: $Ot_n \rightarrow Ot_{n-1} \rightarrow ... \rightarrow OT_2 \rightarrow OT_1 \rightarrow ANY$

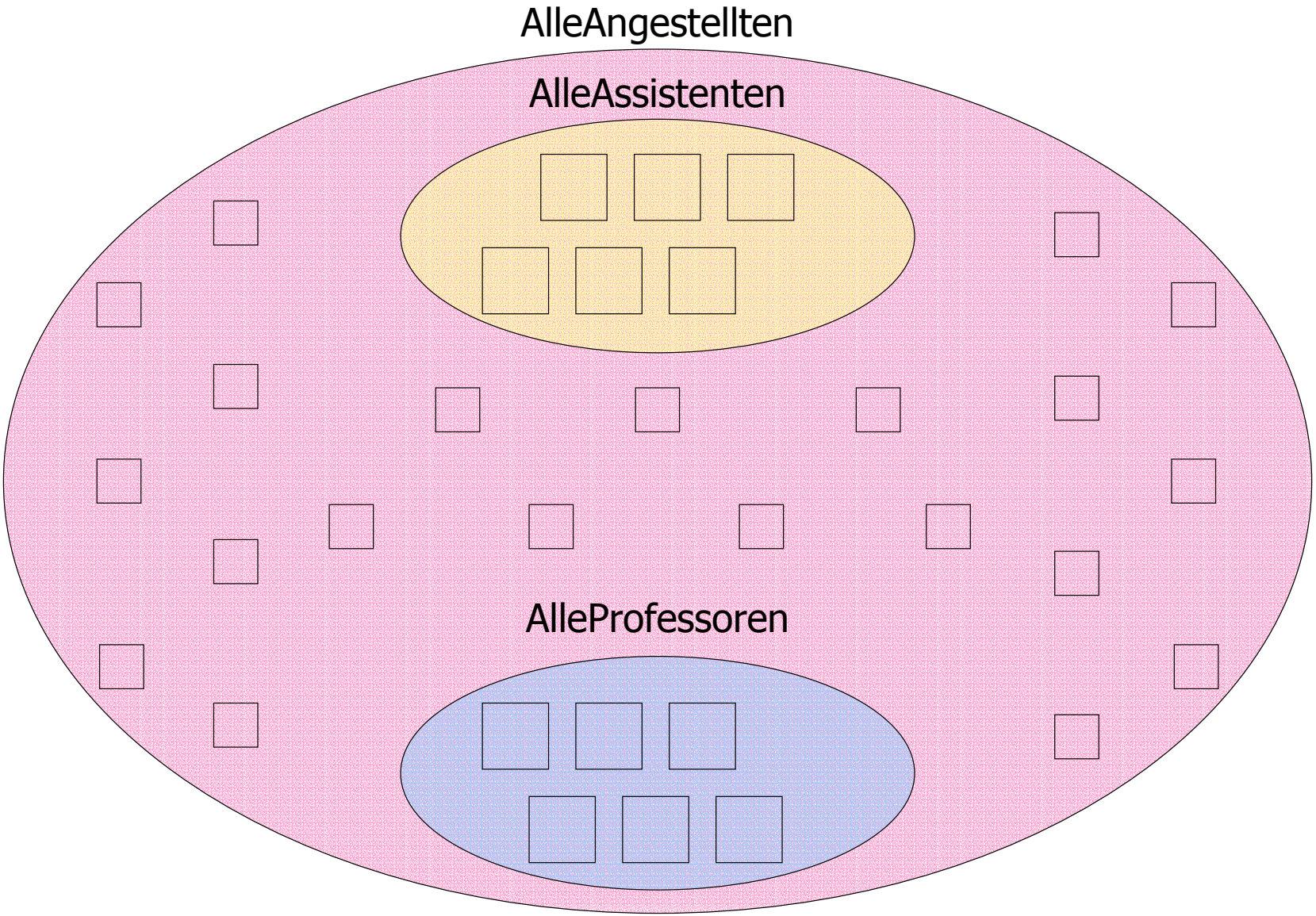
Vererbung von Eigenschaften



Interface-Definition in ODL

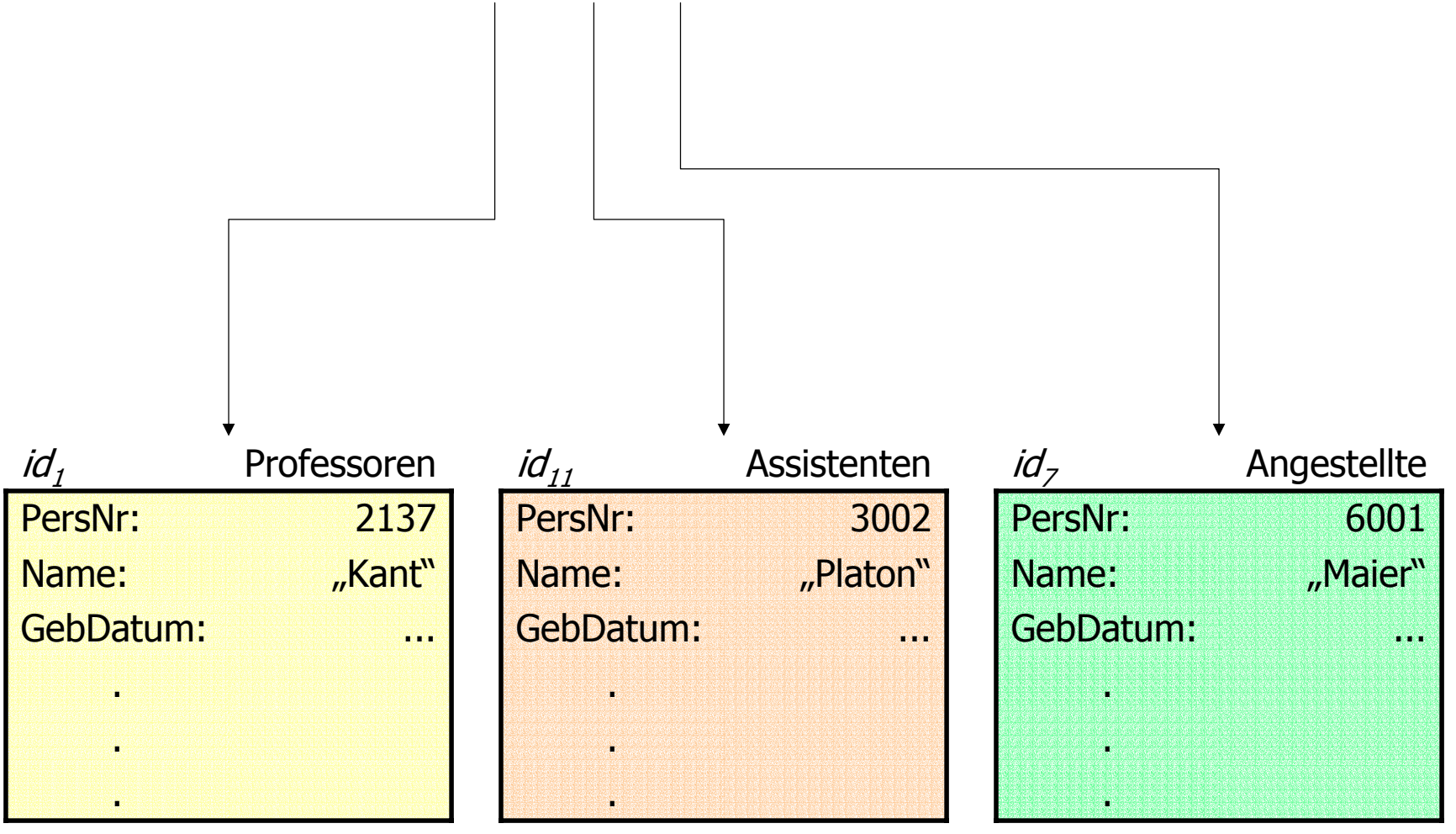
```
class Angestellte (extent AlleAngestellte) {
    attribute long PersNr;
    attribute string Name;
    attribute date GebDatum;
    short Alter();
    long Gehalt();
};

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}



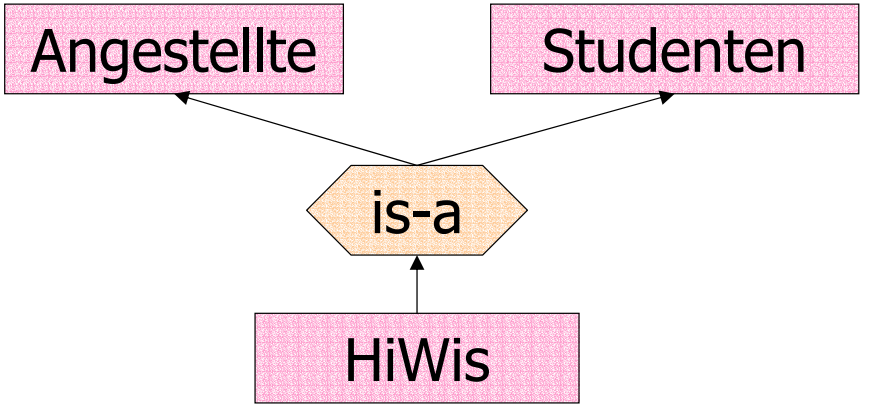
Verfeinerung (Spezialisierung) der Operation Gehalt

- *Angestellte* erhalten: 2000 + (Alter() – 21) * 100
- *Assistenten* bekommen: 2500 + (Alter() – 21) * 125
- *Professoren* erhalten: 3000 + (Alter() – 21) * 150

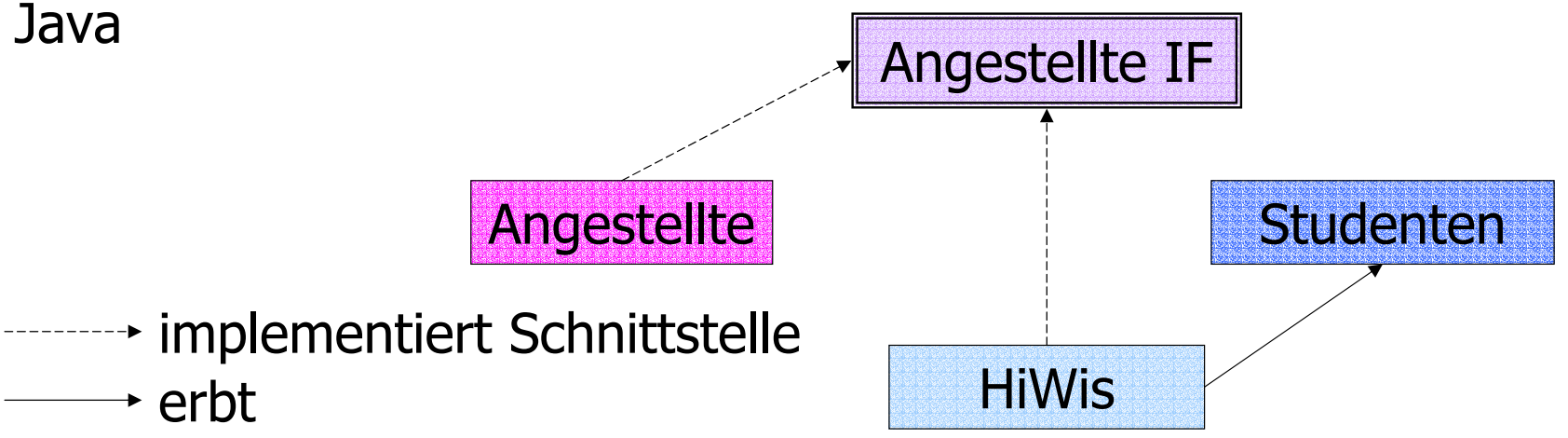
```
select sum(a.Gehalt())
from a in AlleAngestellten
```

- für das Objekt id_1 wird die *Professoren*-spezifische *Gehalts*-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 auch 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 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 Alle Professoren
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 Pfadausdruckes

- ein längerer Pfadausdruck
- *eineVorlesung.gelesenVon.residiertIn.Größe*

Vorlesungen

Professoren

Räume

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

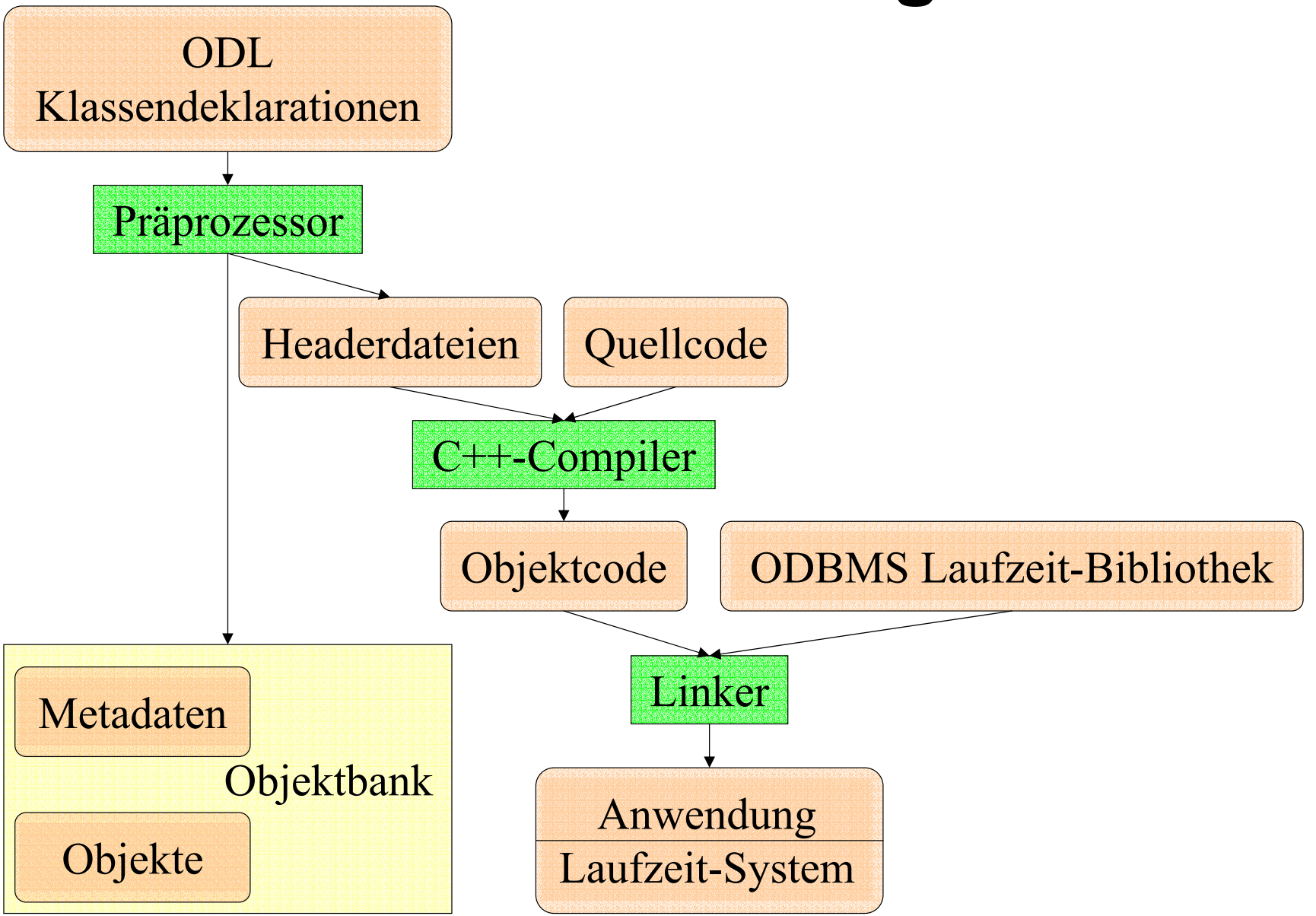
```
select a.Name
from a in AlleAngestellte
where a.Gehalt() > 100.000;
```

Programmiersprachen-Anbindung

Entwurfsentscheidung

- Entwurf einer neuen Sprache
  - eleganteste Methode,
  - hoher Realisierungsaufwand
  - Benutzer müssen eine neue Programmiersprache lernen
- Erweiterung einer bestehenden Sprache
  - Benutzer müssen keine neue Sprache lernen
  - manchmal unnatürlich wirkende Erweiterungen der Basissprache
- Datenbankfähigkeit 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



Objektidentität

```
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

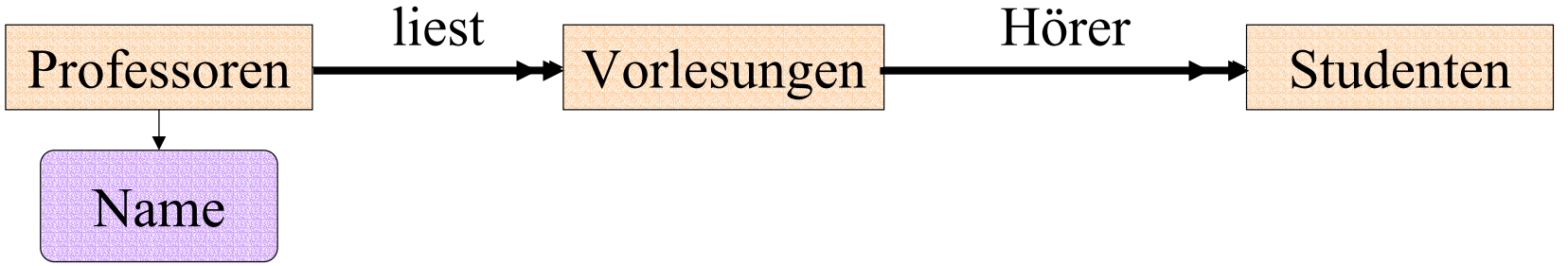
- Schachtelung von Transaktionen
- notwendig um Operationen, die TAs repräsentieren, geschachtelt aufrufen zu können.

```
void Professoren::Umziehen(Ref ⟨Räume⟩ neuerRaum) {
    Transaction TAumziehen;
    TAumziehen.start();
    ...
    if ( /*Fehler? */ )
        TAumziehen.abort();
    ...
    TAumziehen.commit();
};
```

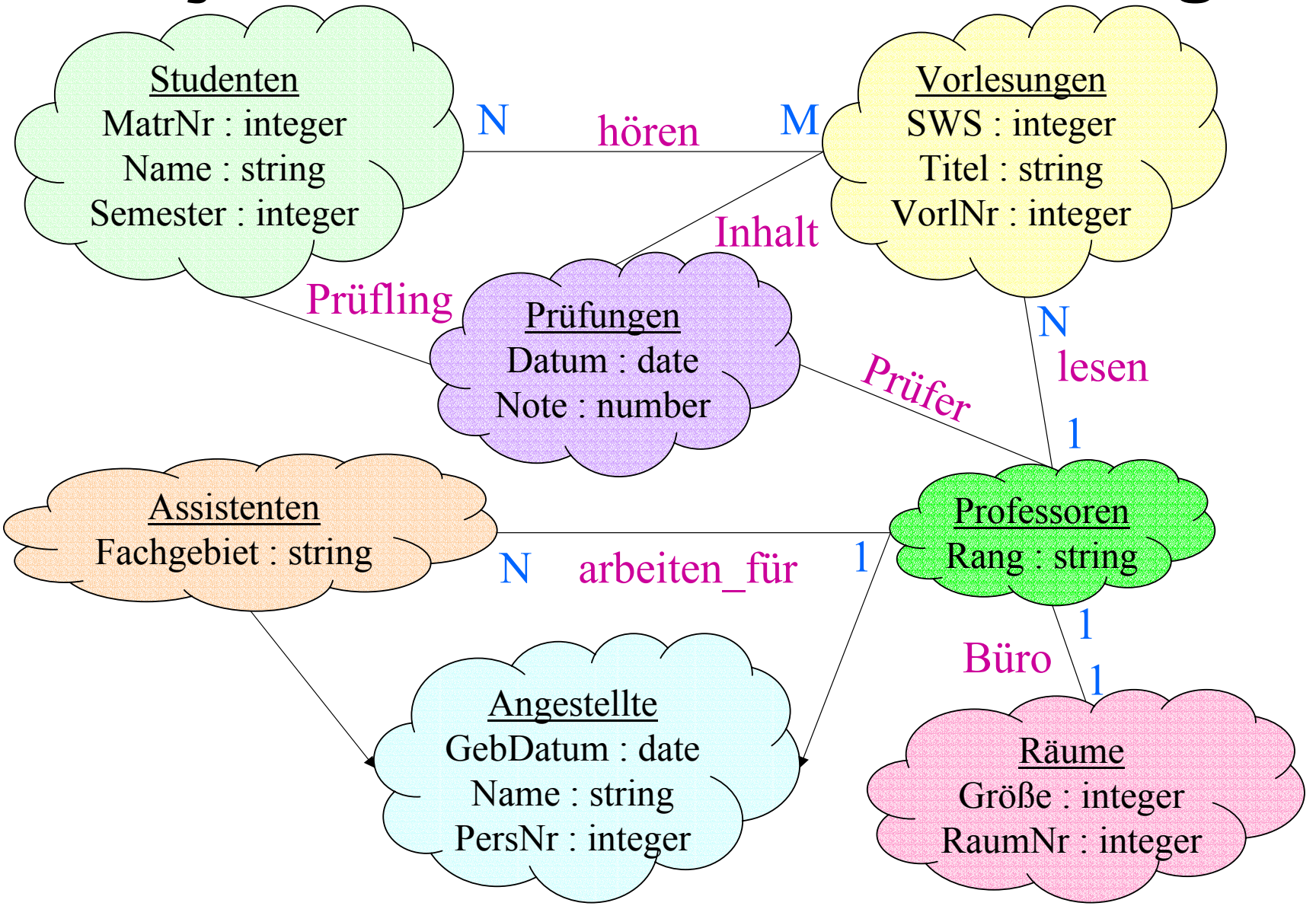
Einbettung von Anfragen

```
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



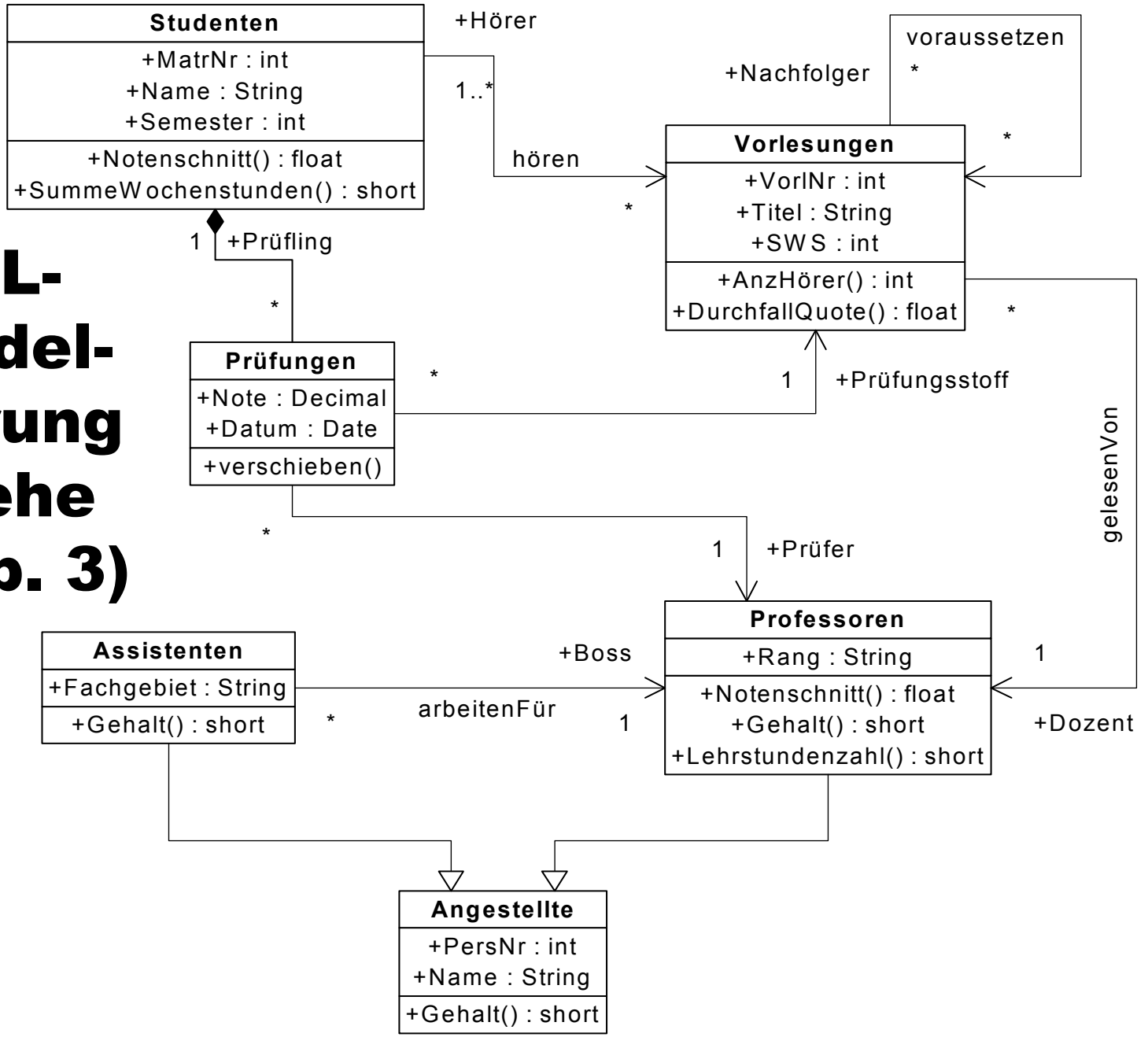
Objektorientierte Modellierung



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, Lorenson: Object-oriented Modelling and Design,
  - Prentice Hall,Englewood Cliffs, New Jersey, 1991.
- Mittlerweile wurden die beiden **Methoden** (Notationen) *„vereinigt"*
  - UML-Standard (Unified Modelling Language)

UML-Modellierung (siehe Kap. 3)



Kommerziell verfügbare Produkte

GemStone

Illustra

Itasca

MATISSE

O_2

Objectivity/DB

ObjectStore

Ontos

OpenOBD

POET

UniSQL

Statice

Versant

Objektrelationale Datenbanken

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