

# OOA (Teil III)

## 31) Strukturelle Analyse, getrieben von aus Metamodellen abgeleiteten Fragen - Wie verstehe ich bloß genau, was mein Kunde will?

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 19.0.1, 15.06.19

- 1) Metamodelle
- 2) Analyse von Klassen und Merkmalen
- 3) Analyse von Klassen-Beziehungen
- 4) Analyse von Endo-Relationen
  - 1) Aggregation und Komposition
- 5) Mehrfachvererbung



# Obligatorische Literatur

2

Softwaretechnologie (ST)

- ▶ Zuser, Kap. 7-9
- ▶ Störrle 5.2-5.5
- ▶ Balzert Kap. 6-7, 9-10. Die strukturelle, metamodelldgetriebene Analyse wird im Balzert diskutiert.

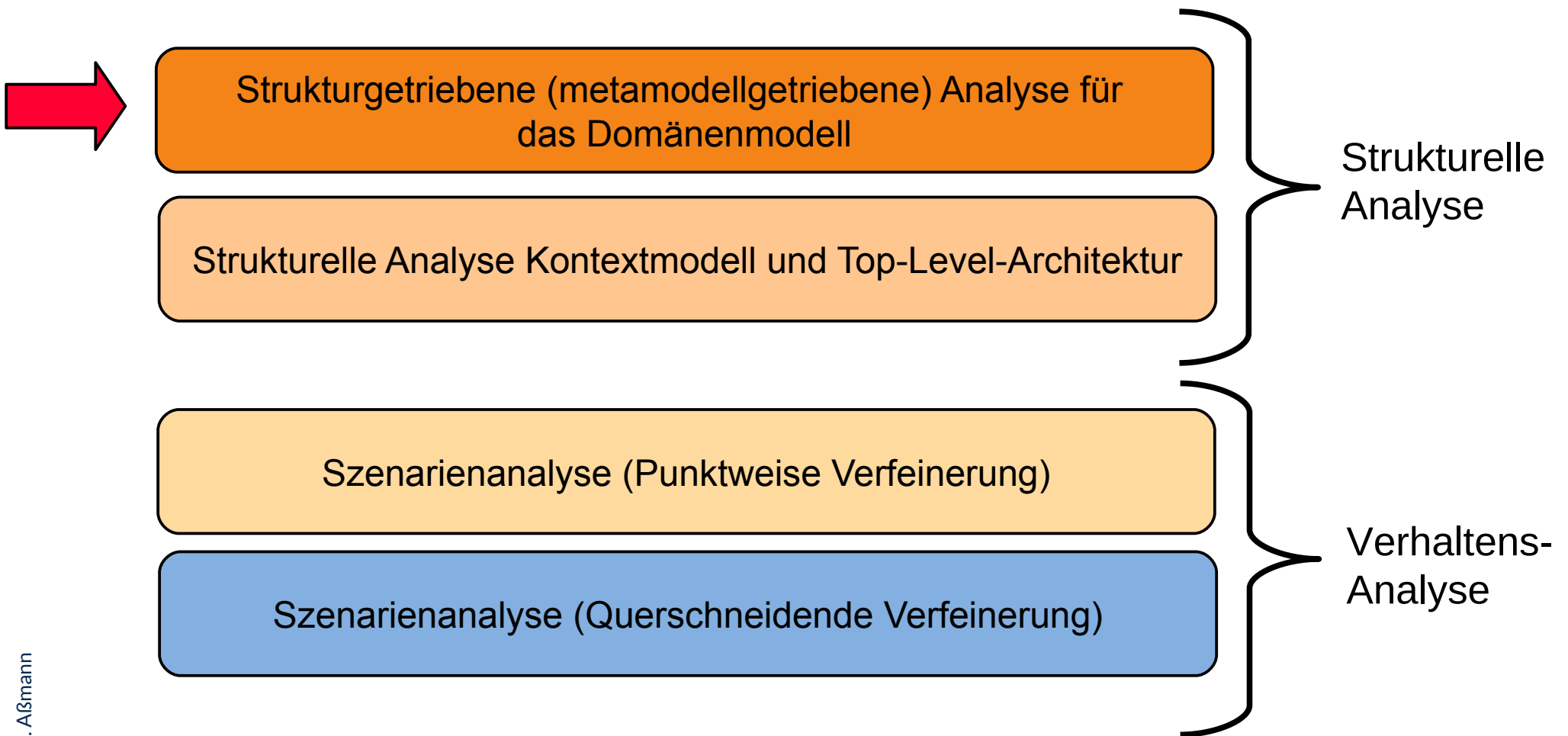
Nicht-obligatorisch:

- ▶ OOA ist eine spezielle Form des Requirements Engineering
  - <https://www.sophist.de/anforderungen/requirements-engineering/faq-requirements-engineering/>

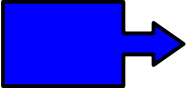
Notation: Wir schreiben:

- ▶ Analysefragen in Font LeckerliOne: *Which tools does a text contain?*
- ▶ Zu analysierende Texte des Kunden in Font ComicJens: *Eine Teambesprechung ist ein Termin.*

# Objektorientierte Analyseverfahren



# Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
  1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodelgetriebene Modellierung mit UML
  1. Strukturelle metamodelgetriebene Modellierung für das Domänenmodell
  2. Strukturelle Modellierung von komplexen Objekten
  3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
  1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
  2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
  3. (Funktionale querschneidende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent

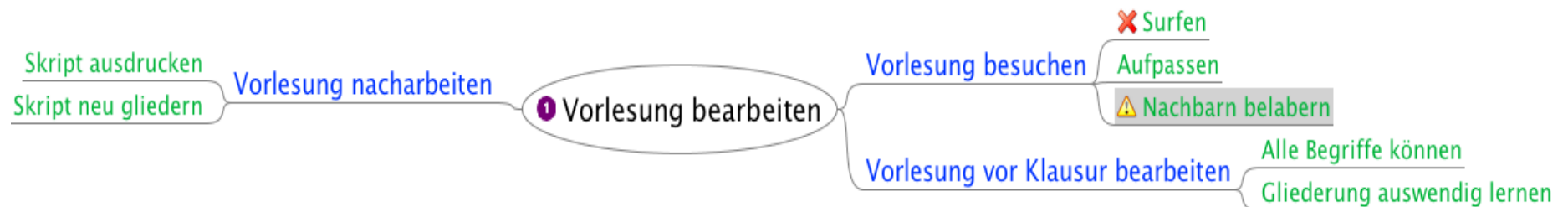
# Exkurs E31.2: Lernen und Hierarchien

- ▶ Zum Lernen verwendet man Begriffshierarchien [Wolf, 2.1]
  - Einen Lernstoff arbeitet man von einer Quelle aus in eine Begriffshierarchie um
  - Aus einem Buch, aus einer Vorlesung, aus einer Diskussion oder Brainstorm
- ▶ Die Begriffshierarchie formt man so lange um, bis sie vollständig das Lerngebiet abdeckt und man sich abei allen Begriffen “wohlfühlt”
- ▶ Wohlfühlen heißt:
  - Man kann für jeden Begriff der Begriffshierarchie eine Definition geben
  - Man kann den Begriff anderen erklären
  - Man kann die Begriffe gegen verwandte Begriffe abgrenzen bzw. sie vergleichen
  - Man kann zu einem Begriff ein Problem schildern, bei dem der Begriff eine Rolle spielt
  - Man kann eine Aufgabe lösen, die mit dem Begriff zu tun hat
- ▶ Die Begriffshierarchie muss vollständig auswendig gelernt werden: sie muss zuerst in den Kopf, dann ins Herz wandern (siehe Bloomsche Taxonomie des Lernens)

Haben Sie schon eine Begriffshierarchie für alle Begriffe der Vorlesung begonnen?

# Bsp.: Lernen mit hierarchischen Wissenskarten (Mind Maps)

- ▶ Mindmaps (Wissenskarten) sind Begriffshierarchien, die von der Mitte des Blattes her monozentral gezeichnet werden (“mapping your mind”)
- ▶ Sie können als Startpunkt beim Lernen eingesetzt werden
  - Untersuchen Sie, ob Sie mit Zeilenhierarchien oder Mindmaps besser zurecht kommen
  - Aufgabe: Suchen Sie mit Google nach freien Mindmap-Werkzeugen für den Computer Ihrer Wahl
  - Zeichnen Sie eine Mind map von diesem Kapitel



# Objektorientierte Analyse

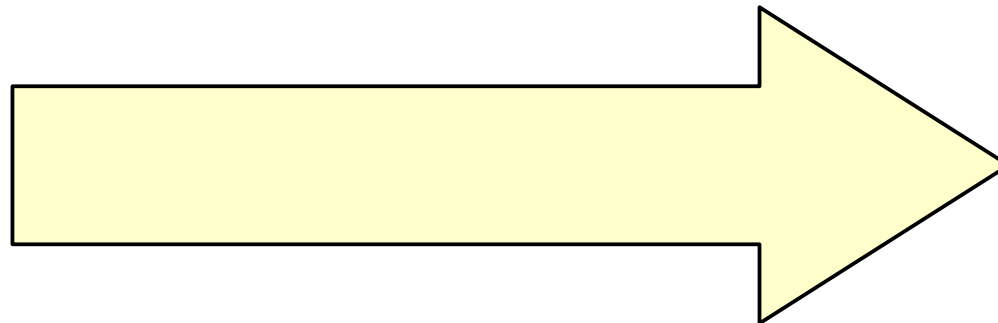
Objektorientierte Analyse analysiert die Texte des Kunden mit objektorientierten Mitteln, um Missverständnisse zu vermeiden.

Der natürlichsprachliche Text wird in ein Modell einer objektorientierten Sprache übersetzt (Verb-Substantiv-Analyse).

User Story 1

User Story 2

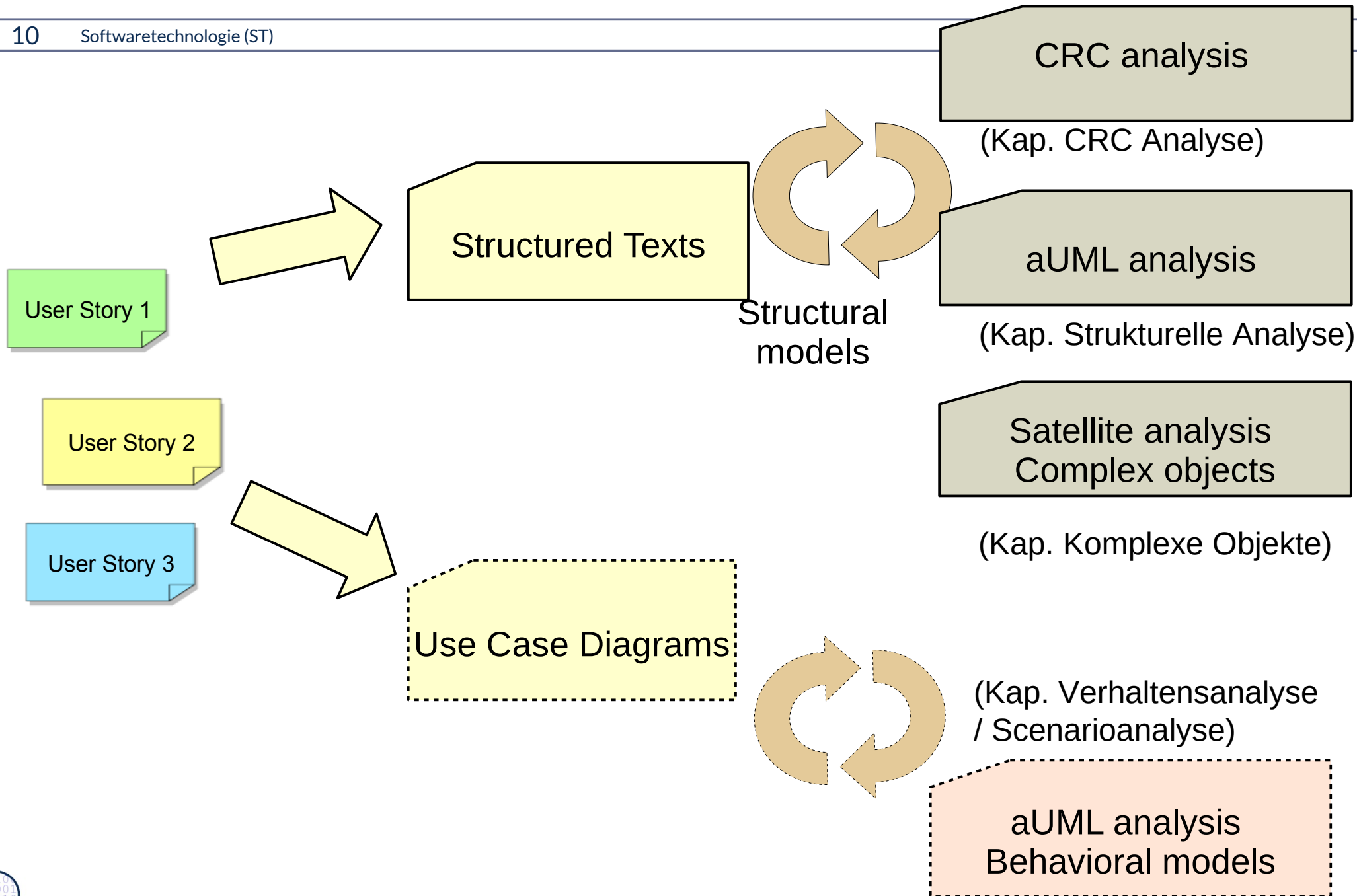
User Story 3



Modell in  
objektorientierter  
Sprache

„Softwareingenieure machen die Analyse genauer als Psychoanalysten.“

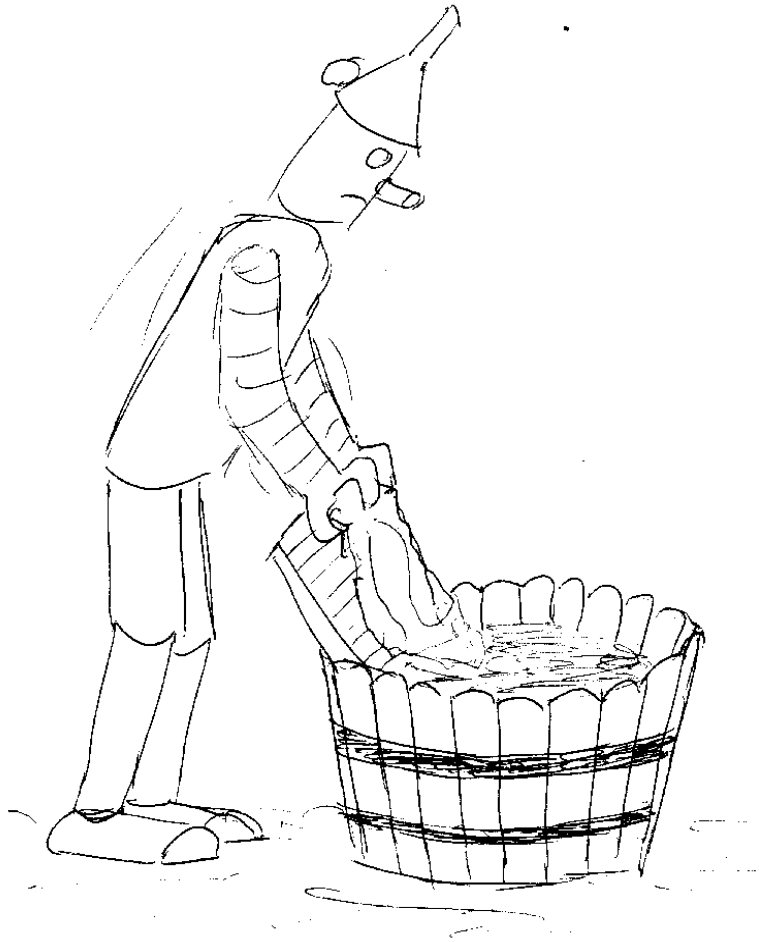
# Problem: Wie komme ich von Interviews zum Gerüst des fachlichen Modells? (Inhalt des Pflichtenhefts)





# Attention: Model the Real Need of the Customer

*„I need a machine washing my clothes.“*



→ Prototype of a washing machine

**Find the right abstractions  
in modeling!**

[Andreas  
Ludwig]

# Famous Misunderstandings

- ▶ “Do you have four-volt, two-watt light bulbs?
- ▶ For what?
- ▶ No, two watt.
- ▶ To what purpose does this go?
- ▶ No, two watt.....”
- ▶ [Internet Joke. cited from James C. Jarrad. The Case of the Missing Pronoun. Xlibris. 2012]

# The Basic Laws of Misunderstanding in Analysis [K. Lorenz]

Spoken is not heard  
Heard is not listened  
Listened is not understood  
Understood is not accepted  
Accepted is not done

# Objektorientierte Analyse vermeidet Missverständnisse

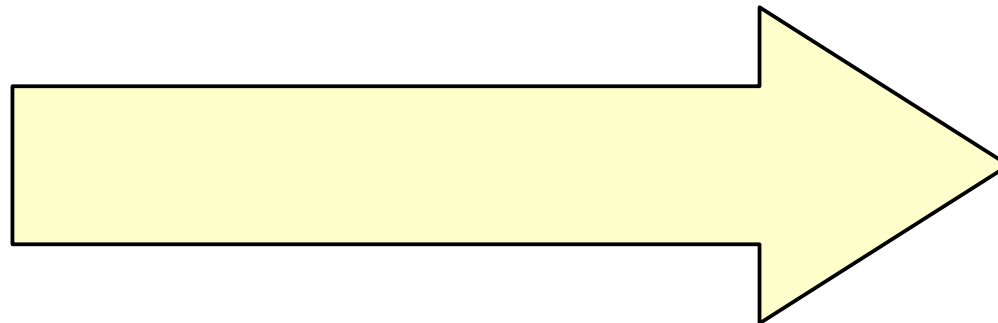
Objektorientierte Analyse analysiert die Texte des Kunden mit objektorientierten Mitteln, um Missverständnisse zu vermeiden.

Der natürlichsprachliche Text wird in ein Modell einer **formalen** objektorientierten Sprache mit **Vokabularium (Metamodell)** übersetzt.

User Story 1

User Story 2

User Story 3



Modell in **formaler** objektorientierter Sprache

„Softwareingenieure machen die Analyse genauer als Psychoanalysten.“

# 31.1 Wie man aus Sprachstrukturmodellen (Metamodellen, Vokabularien) Fragen für die Analyse gewinnt

Modelle sind überall...

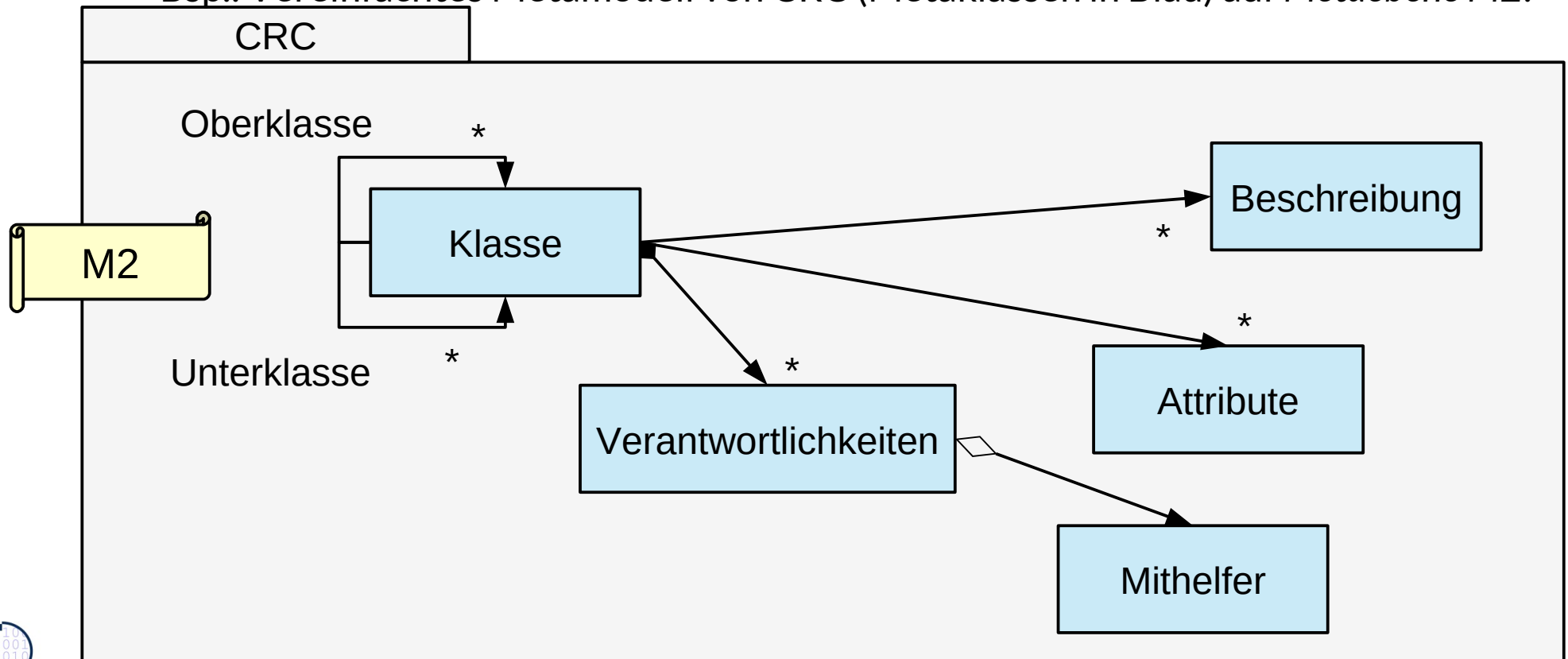
Ihre Struktur wird beschrieben durch *Metamodelle*



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

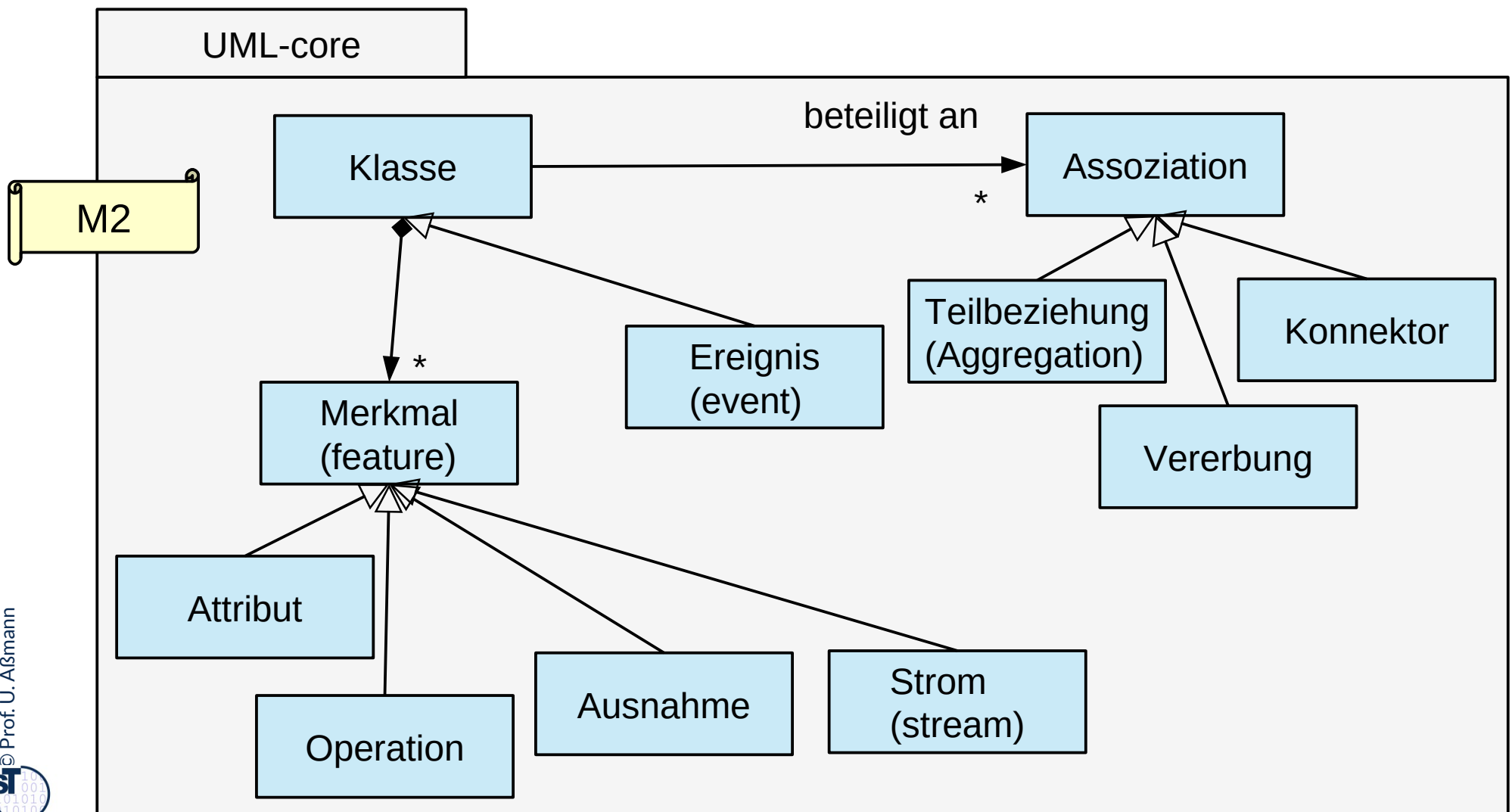
# Sprachstrukturmodelle (Metamodelle, Vokabularien)

- ▶ Ein **Sprachstrukturmodell (Metamodell, Vokabularium)** beschreibt die *Struktur einer Menge von Modellen* mit Hilfe eines (Meta-)Klassendiagramms, d.i. die *Struktur einer Modellierungssprache*
- ▶ Ein **Sprachstrukturmodell (Metamodell, Vokabularium)** verallgemeinert eine Begriffshierarchie (Taxonomie) mit Beziehungen zwischen den Begriffen. Ein Metamodell ist ein *Vokabularium* von Sprachkonzepten und ihren Beziehungen.
- ▶ Bsp.: Vereinfachtes Metamodell von CRC (Metaklassen in Blau) auf *Metaebene M2*:



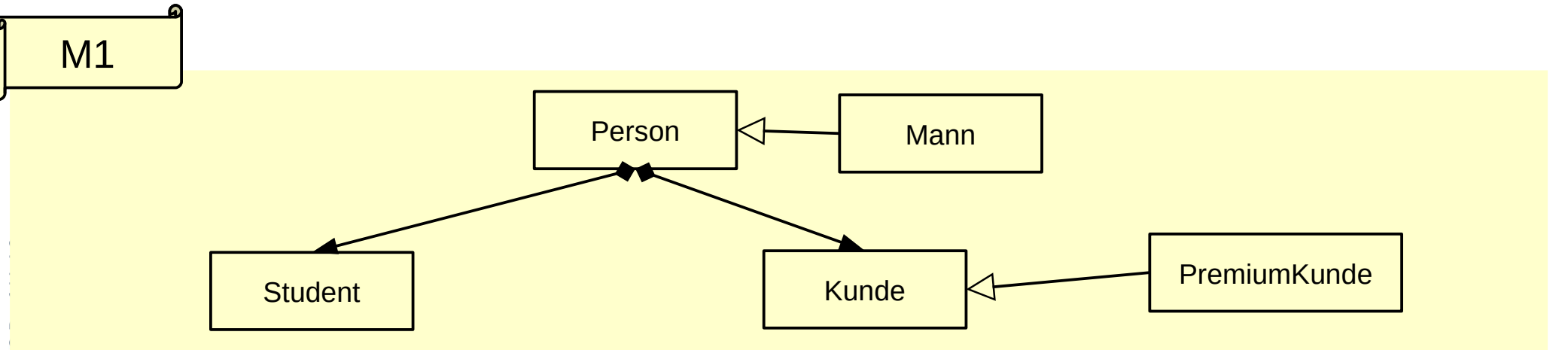
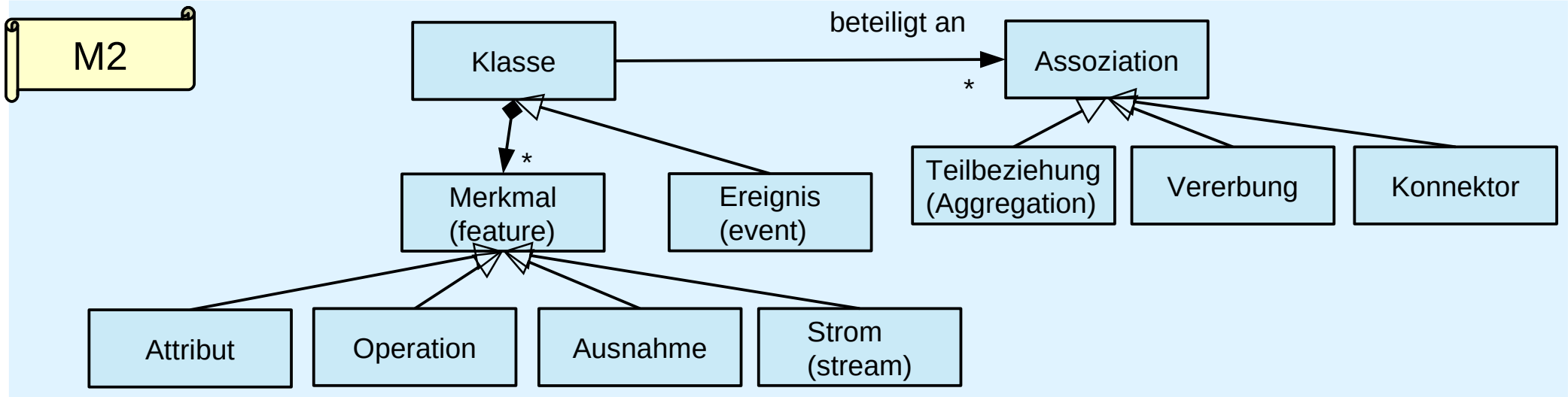
# Sprachstrukturmodelle (Metamodelle)

- ▶ In einem Metamodell (Vokabularium) stellen die Metaklassen die *Sprachkonzepte* der modellierten Sprache vor.
- ▶ Vereinfachtes Metamodell von UML:



# Q: Wie sind Modelle mit ihren Metamodellen (Vokabularien) verbunden?

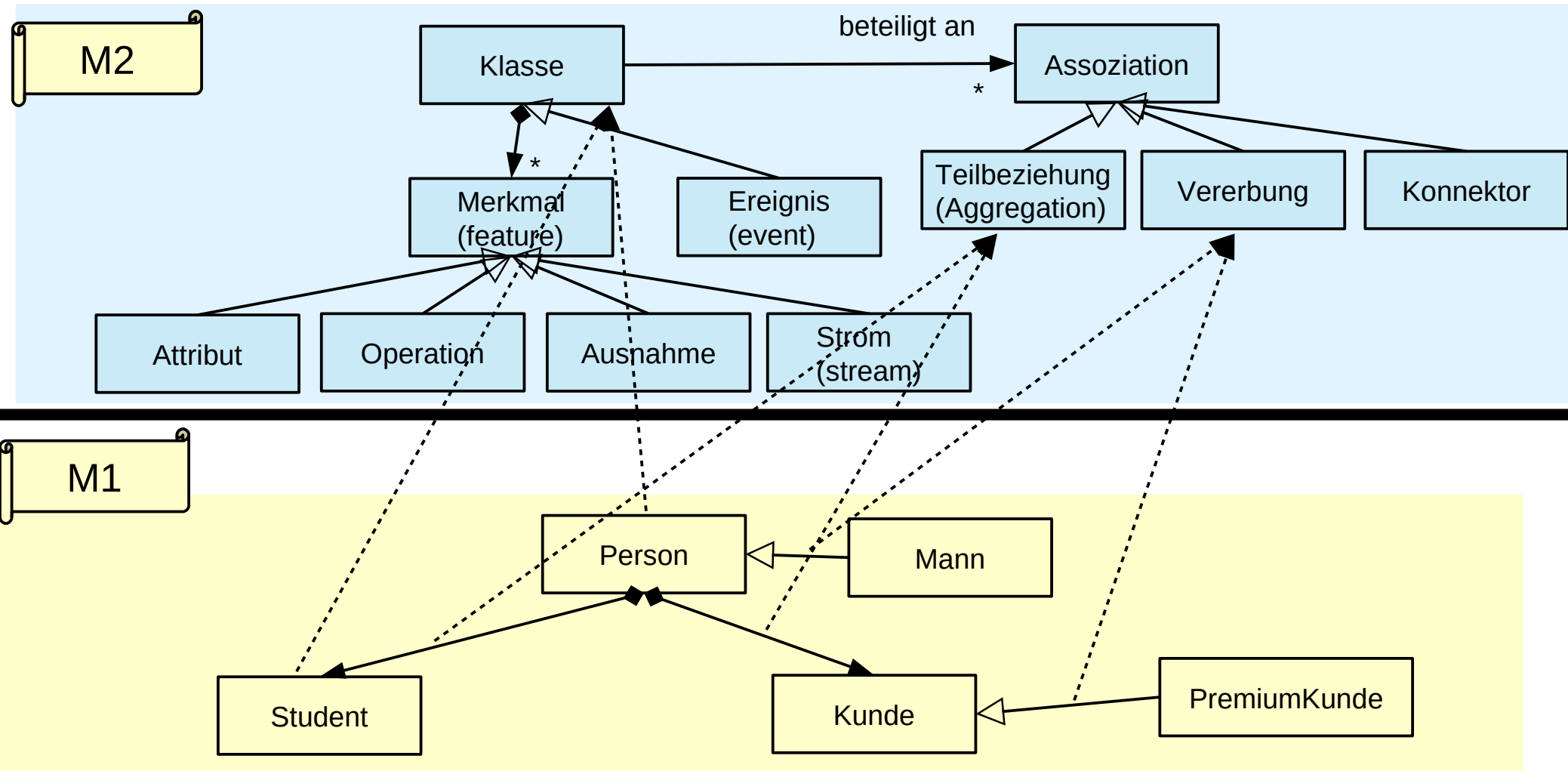
- ▶ Sprachstrukturmodelle (Metamodelle, Vokabularien) liegen auf einer neuen Modellierungsebene M2
- ▶ Wie sind M1 und M2 miteinander korreliert?





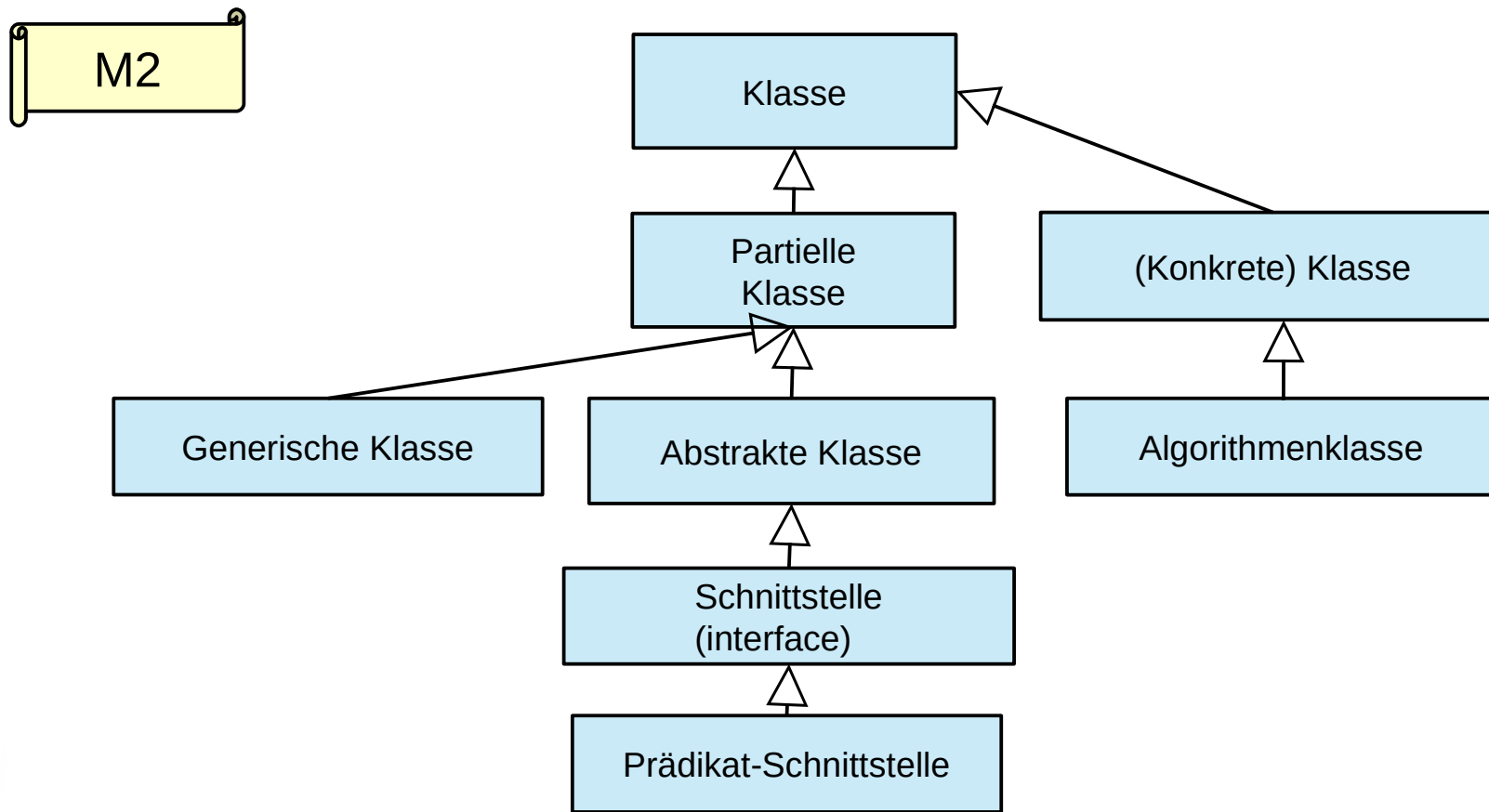
# A: instanceOf Relation koppelt alle Modellelemente zu einem Element des Metamodells

- ▶ Klassen und weitere Elemente von M1 sind durch `instanceOf` verbunden (Darstellung nur auszugsweise)



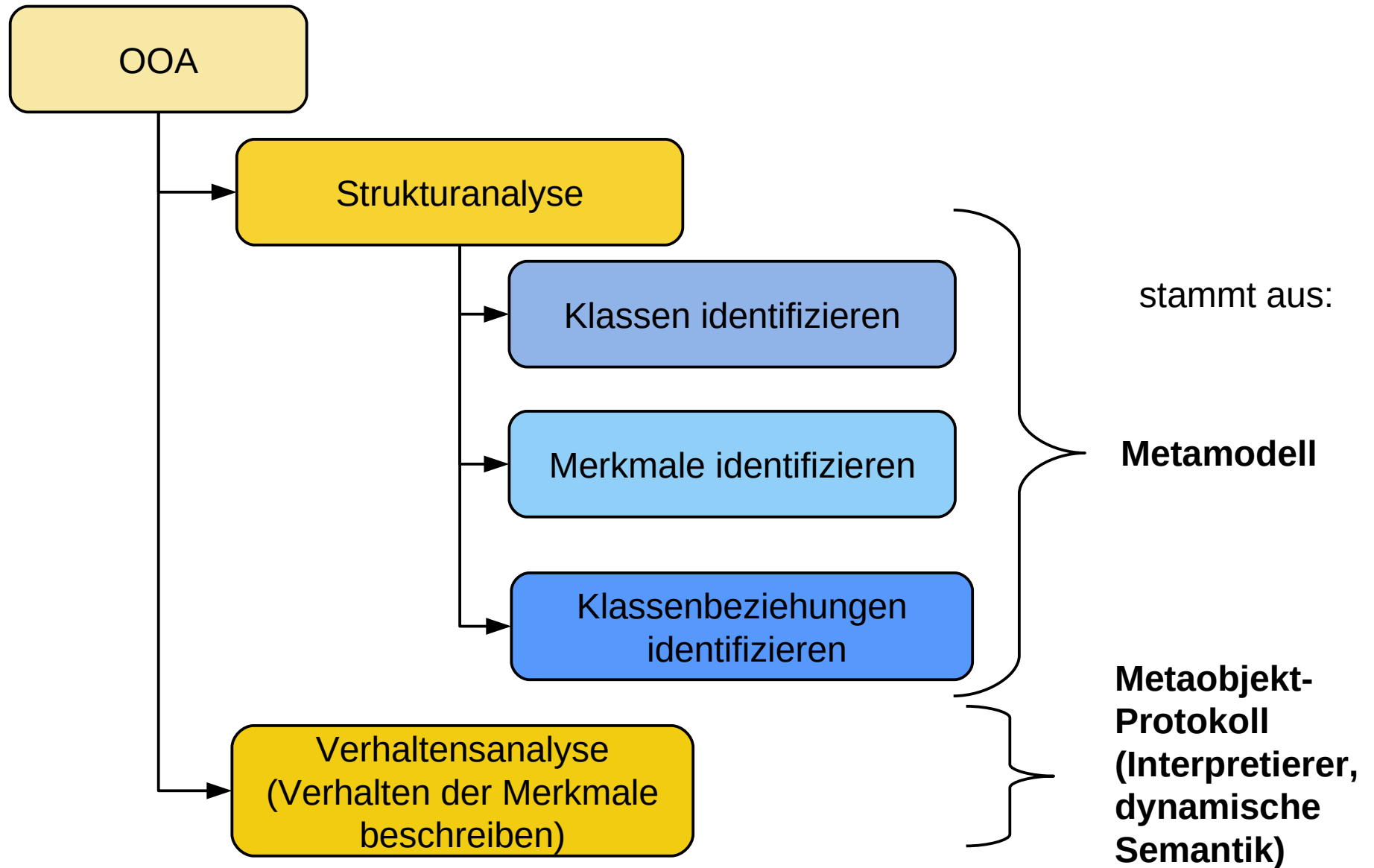
# Exkurs 31.E.1 Begriffshierarchie von Klassen (Erinnerung)

- ▶ Jede Folie des *roten Fadens der Klassenarten* bestand aus einem Metamodell, d.h. die Begriffshierarchien enthalten **Metaklassen (Sprachkonzepte)**
  - Eine Begriffshierarchie ist ein spezielles Metamodell, das hauptsächlich Vererbung verwendet
- ▶ Von diesem Java-Metamodell kann man in Java-Programmen Programmelemente und -fragmente instanziiieren



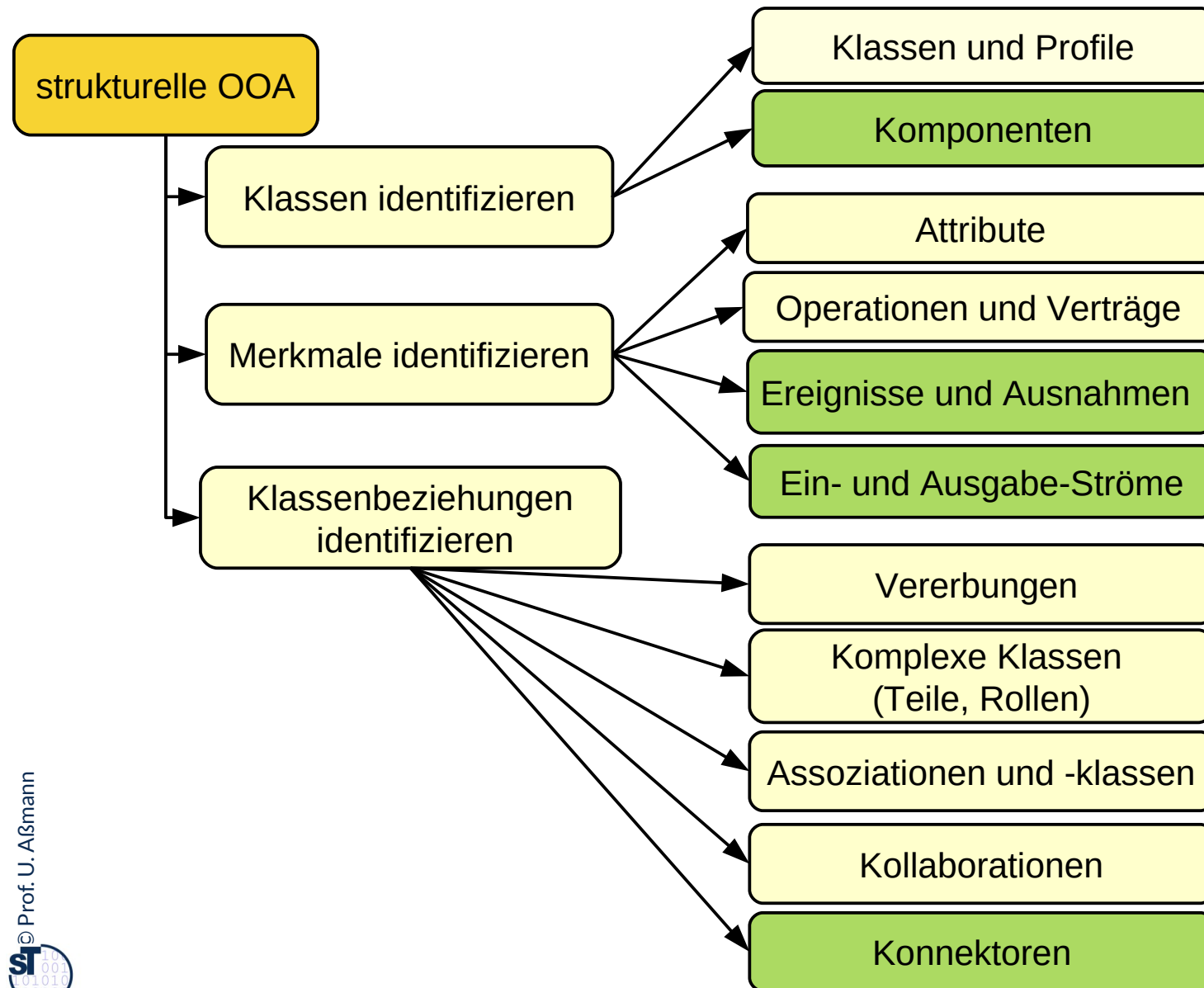
# Hauptschritte der strukturellen, metamodelgetriebenen Analyse mit UML

- ▶ Ähnlich zur CRC Analyse, mit UML-Metamodell, das reichere Struktur besitzt



# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



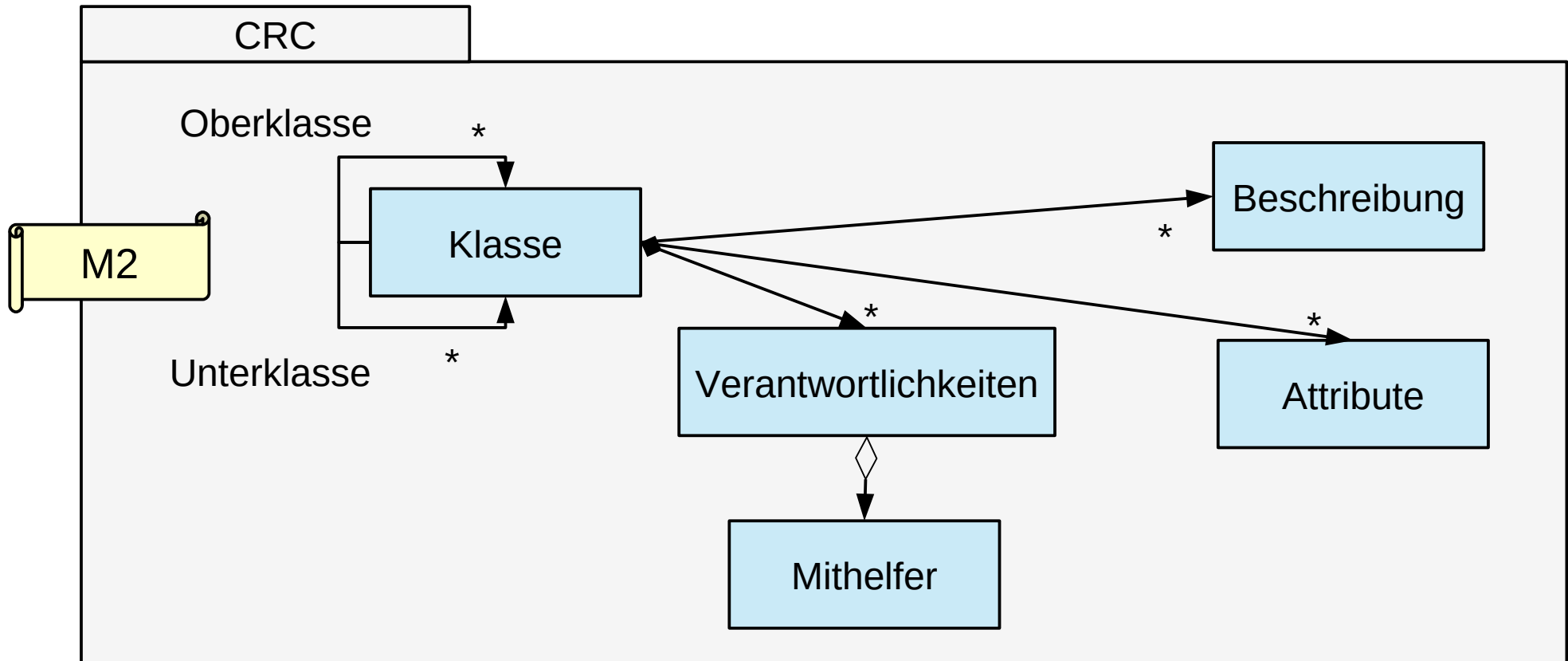
Während einer **strukturgetriebenen Analyse** sucht man in der Problemwelt des Kunden nach Ausprägungen aller Begriffe des Metamodells.  
Daraus erstellt man ein Analysemodell.

Während der **strukturgetriebenen Text-Analyse** sucht man in dem Interviewtext des Kunden nach Ausprägungen aller Begriffe des Metamodells.  
Daraus erstellt man ein Analysemodell.

**Man nutzt das Metamodell, um Analysefragen abzuleiten:**

- Enthält ein Metamodell das Konzept einer **Klasse**, fragen wir den Kunden *„Welche Klassen und Objekte brauchen wir?“*
  - Enthält ein Metamodell das Konzept einer **Methode**, fragen wir *„Welche Methoden brauchen wir?“*
  - Enthält ein Metamodell das Konzept eines **Konnektors**, fragen wir *„Welche Konnektoren brauchen wir?“*
- USW.

# Ableitung von Analysefragen aus dem Metamodell



Die CRC-Methodik ist eine besondere Ausprägung der strukturgetriebenen Analyse.

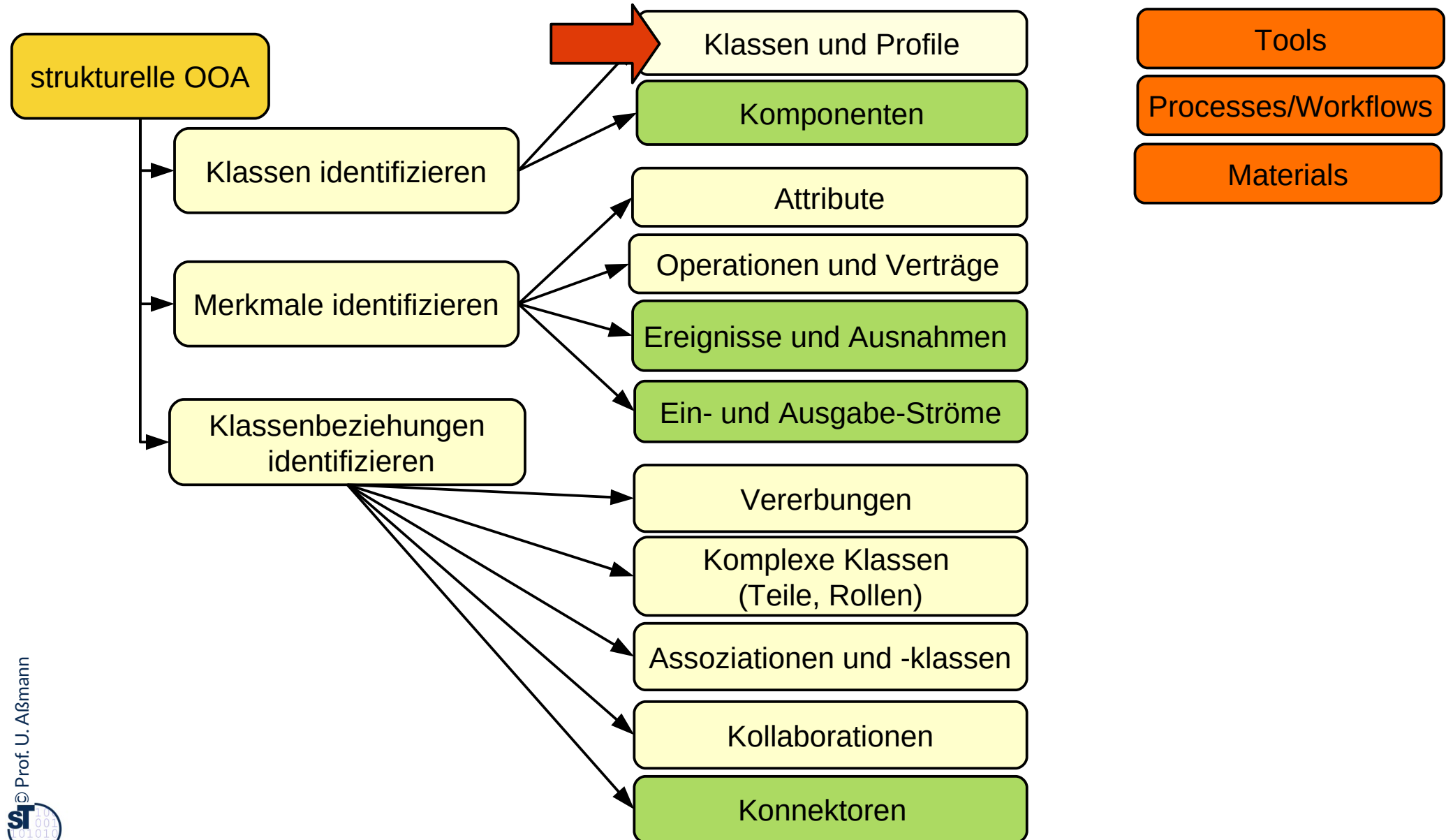
- ▶ *Welche Klassen haben welche Verantwortlichkeiten?*
- ▶ *Wie kann man die Verantwortlichkeiten beschreiben?*
- ▶ *Wer hilft mit, die Verantwortlichkeit zu erledigen?*
- ▶ *Welche Attribute hat eine Klasse?*
- ▶ *Welche Attribute teilt sie mit anderen Klassen und gehören in eine Oberklasse?*

## 31.2 Analyse von Klassen und Merkmalen



# Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



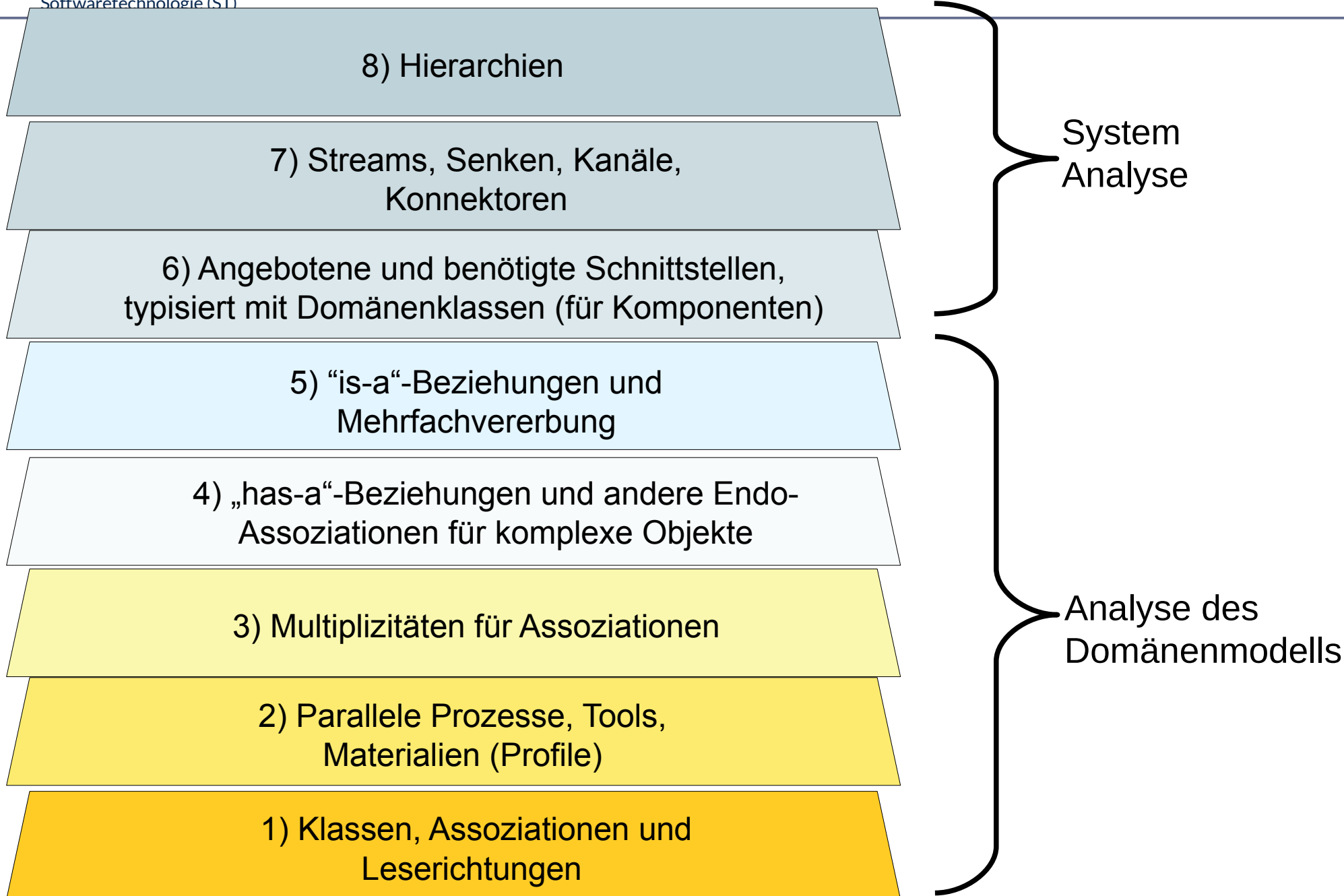


# Schritte der Anreicherung des Analysemodells

## “Was man alles so im Text des Kunden sucht”

28

Softwaretechnologie (ST)



# Beispiel: Ein Tonband des Kunden

- ▶ Ein Analysemodell entsteht aus einem Interview mit einem Kunden, in dem Sätze in Fragmente des Analysemodells umgesetzt werden (“user story”)
- ▶ Ein Analysemodell kann “vorgelesen” werden, und es entsteht der Text des Interviews

## **Erstellen Sie mir eine Terminverwaltung.**

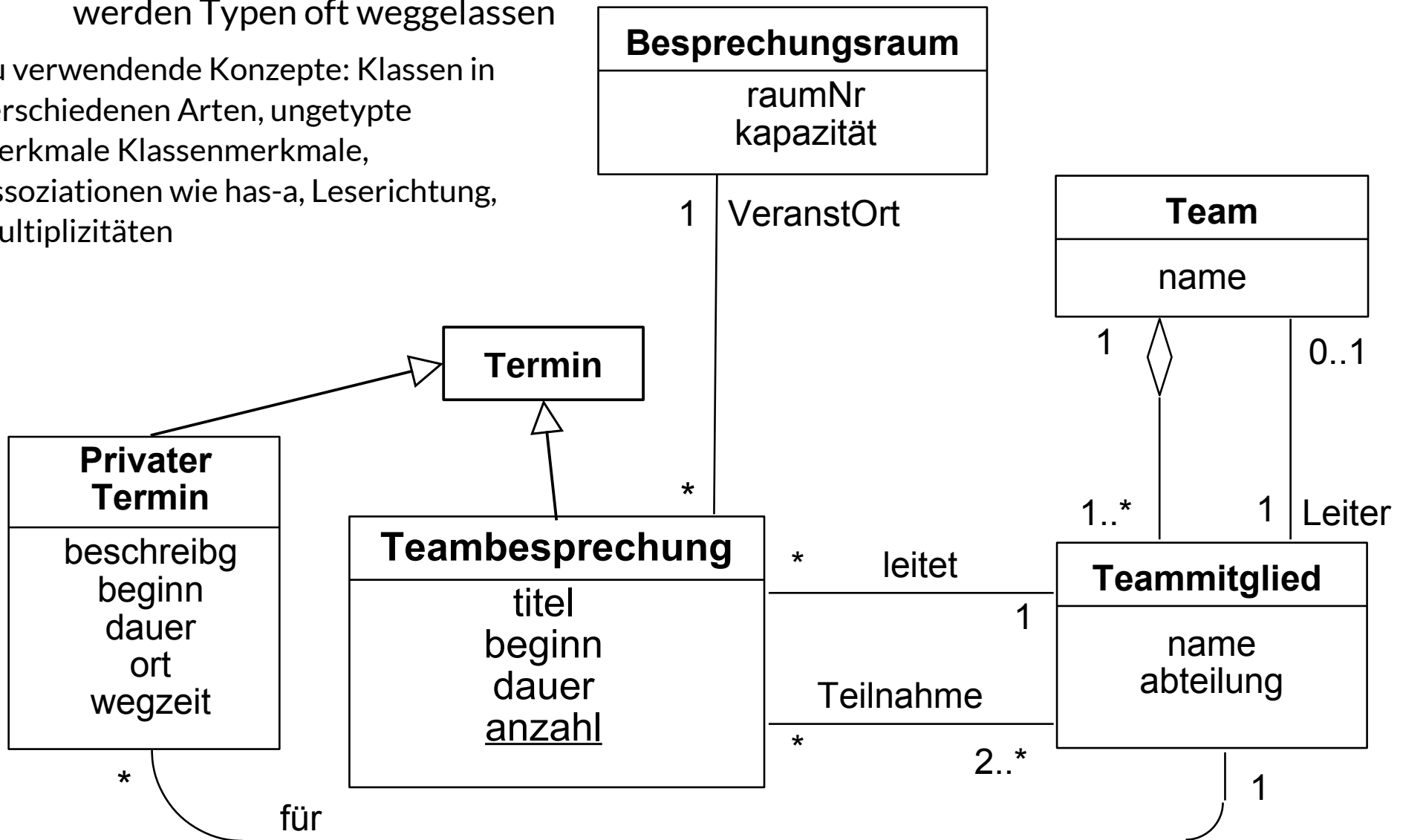
*Teams, bestehend aus Mitgliedern, müssen Besprechungen buchen können. Ein Team hat einen Leiter. Besprechungsräume müssen einer Teambesprechung zugebucht werden. Eine Besprechung hat einen Titel, einen Beginn, eine Dauer, und ist mit einer maximalen Anzahl von Teilnehmern zu versehen. Eine Besprechung ist ein Termin.*

*Private Termine sollen auch aufgenommen werden, damit Konflikte der Teammitglieder erfasst werden.*

# Beispiel: Analysemodelle beginnen mit unvollständiger Information zu Attributen und Methoden (Fragmente)

- ▶ Ein Analysemodell entsteht aus einem Interview mit einem Kunden
- ▶ Es ist immer **fragmentarisch**, d.h. partiell, und enthält **Fragmente** von UML-CD, z.B. werden Typen oft weggelassen

- ▶ Zu verwendende Konzepte: Klassen in verschiedenen Arten, ungetypte Merkmale Klassenmerkmale, Assoziationen wie has-a, Leserichtung, Multiplizitäten



# Übung: Reproduktion des Analysetextes

- ▶ Versuchen Sie, das Modell auf der vorigen Folie in einen Text rückzuverwandeln.
- ▶ Vorgehensweise:
  - Notieren Sie sich alle wichtigen Analysefragen aus dem Metamodell der UML-Klassendiagrammen
  - Schrittweises Vorgehen:
    - Setzen Sie ein Häkchen an jedes bereits in einen Satz rückverwandelte Modellelement.
- ▶ Nachuntersuchung:
  - Fragen Sie sich, ob sie alle Analysefragen berücksichtigt haben.
  - Wie unterscheidet sich Ihr generierter Text von der User Story? Warum ist er (in der Regel) nicht gleich?

# Aufgabe der Analyse von Klassen in den einzelnen Schichten des Systems

- ▶ im **Domänenmodell**:
  - Klassen können sowohl Objekte als Begriffe repräsentieren
  - Vererbung zwischen Begriffen bilden Metamodelle (Begriffshierarchien, Taxonomien)
  - Assoziation, Aggregation und Komposition (Ganz-Teile-Beziehungen)
- ▶ im **Analysemodell (funktionale Anforderungen)**:
  - Klassen repräsentieren Daten, die der Kunde als Ergebnis der Funktionen des Systems haben will
- ▶ in **Kontextmodell und Top-level-Architektur**:
  - Klassen repräsentieren Daten, die auf Kanälen fließen
  - Prozesse, die die Daten bearbeiten
  - in funktionalen Anforderungen: funktionale Anforderungen werden Operationen, die Klassen zugeordnet sind
- ▶ Im **Datenmodell** (“Materialien” aus Domänenmodell abgeleitet):
  - Klassen repräsentieren passive Materialien, die Wissen repräsentieren

## 31.2.2 Stereotypen und Profile



# Profile und Stereotypen für Analysemodelle

- ▶ Für die Spezifikation von Analysemodellen erlaubt es UML, Klassen mit **Stereotypen** zu annotieren
  - Klassen werden “markiert”, klassifiziert, dh. mit mehr Semantik ausgestattet
  - Komponenten, Ports, Konnektoren können unterschieden werden
- ▶ Kataloge von Stereotypen heissen **Profile**
  - Wenn der Ingenieur einige Profile (inkl. ihrer Stereotypen) kennt, kann er seine Kontextmodelle mit mehr Semantik ausstatten
  - UML 2.0 superstructure, Appendix B, enthält eine grosse Menge von technischen Standard-Stereotypen
  - Man kann auch seine eigenen Profile definieren

# Profil “Aktive und Passive Klassen”

- ▶ Zum Analysemodell gehört die Unterscheidung von *aktiven Klassen (Prozessen)* und *passiven Klassen*

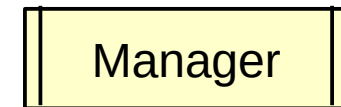
- Im einfachsten Fall existiert *eine* aktive Klasse

- ▶ **Aktive Objekte**

- Aktoren, Prozesse

- Werkzeuge (Tools): Aktives Objekt, das Materialien schreibt)

- Workflow (interaktiver Prozess): Prozess, der Tools aufruft



<<actor>>

<<active class>>

<<tool>>

<<workflow>>

<<process>>

- ▶ **Passive Objekte** (Daten, Materials, entities, data objects)

- Aktive Objekte arbeiten auf Materialien

- Persistente Materialien

- Rollen: Dynamische Sicht auf ein Material

<<material>>

<<data>>

<<role>> <<passive class>>

- ▶ **Konnektoren (connectors) und Kanäle (channels, pipes):** Beschreiben, wie aktive Objekte über Datenobjekte miteinander kommunizieren

- Über Kanäle fließen Datenobjekte, sie werden mit Senken geschrieben und mit Streams gelesen

<<entity>>

<<channel>> <<connector>>

<<call>>

<<input stream>>



Jedes Stereotyp eines Profils definiert eine Frage für die OOA

- ▶ <<active object>>: *Which active objects does a text contain?*
- ▶ <<tool>>: *Which tools does a text contain?*
- ▶ <<material>>: *Is the object a passive object, a material of a tool?*
- ▶ <<connector>>: *Is the object solely concerned with communication/interaction?*

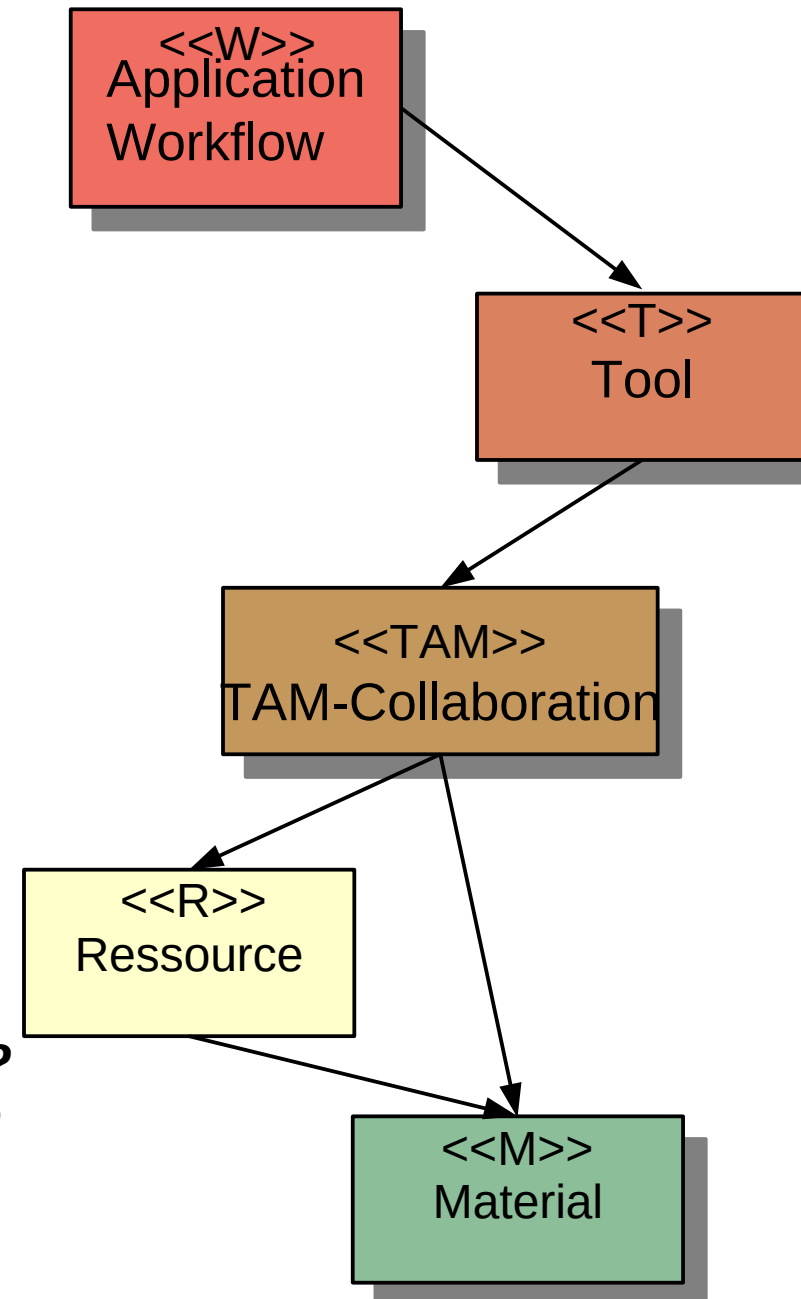
# Profil TAM: Trennung von aktiven und passiven Komponenten

**Tools-and-Materials** [Züllighoven] ist ein Profil, das folgende Aspekte definiert:

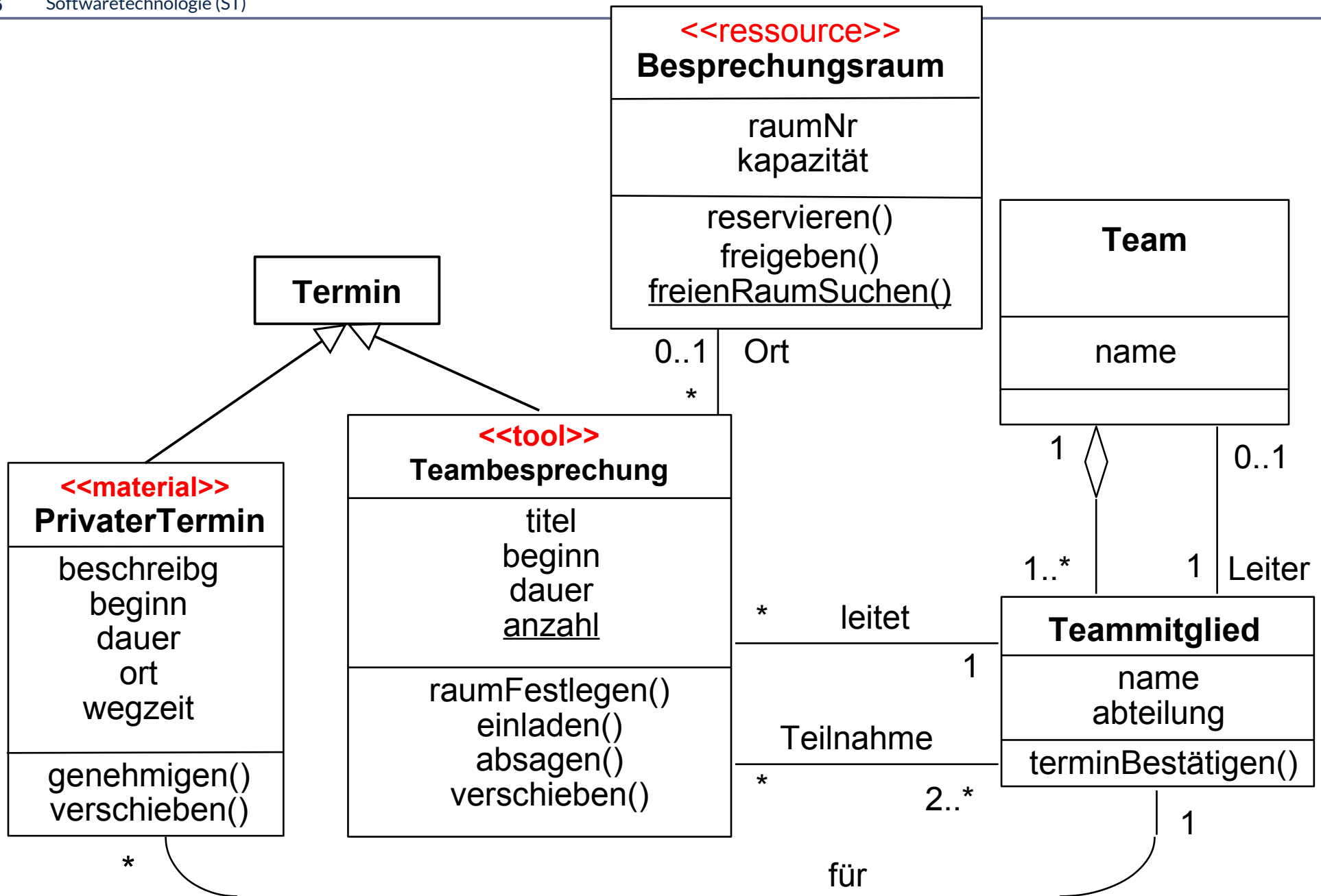
- 1) Tools (aktive Objekte, Command)
- 2) Ressourcen (belegbar)
- 3) Materials (passive Daten, Schicht E)
- 4) TAM-Collaboration
- 5) Workflows (Prozesse) koordinieren Tools

- Klassen, Module, Komponenten, Pakete sollten mit diesen Aspekten qualifiziert werden

- ▶ *Which tools does a text contain?*
- ▶ *Is the object a passive object, a material of a tool?*
- ▶ *How does this tool collaborate with this material?*
- ▶ *How are these tools controlled by a workflow?*
- ▶ *Which materials must be reserved, are resources?*



# Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell

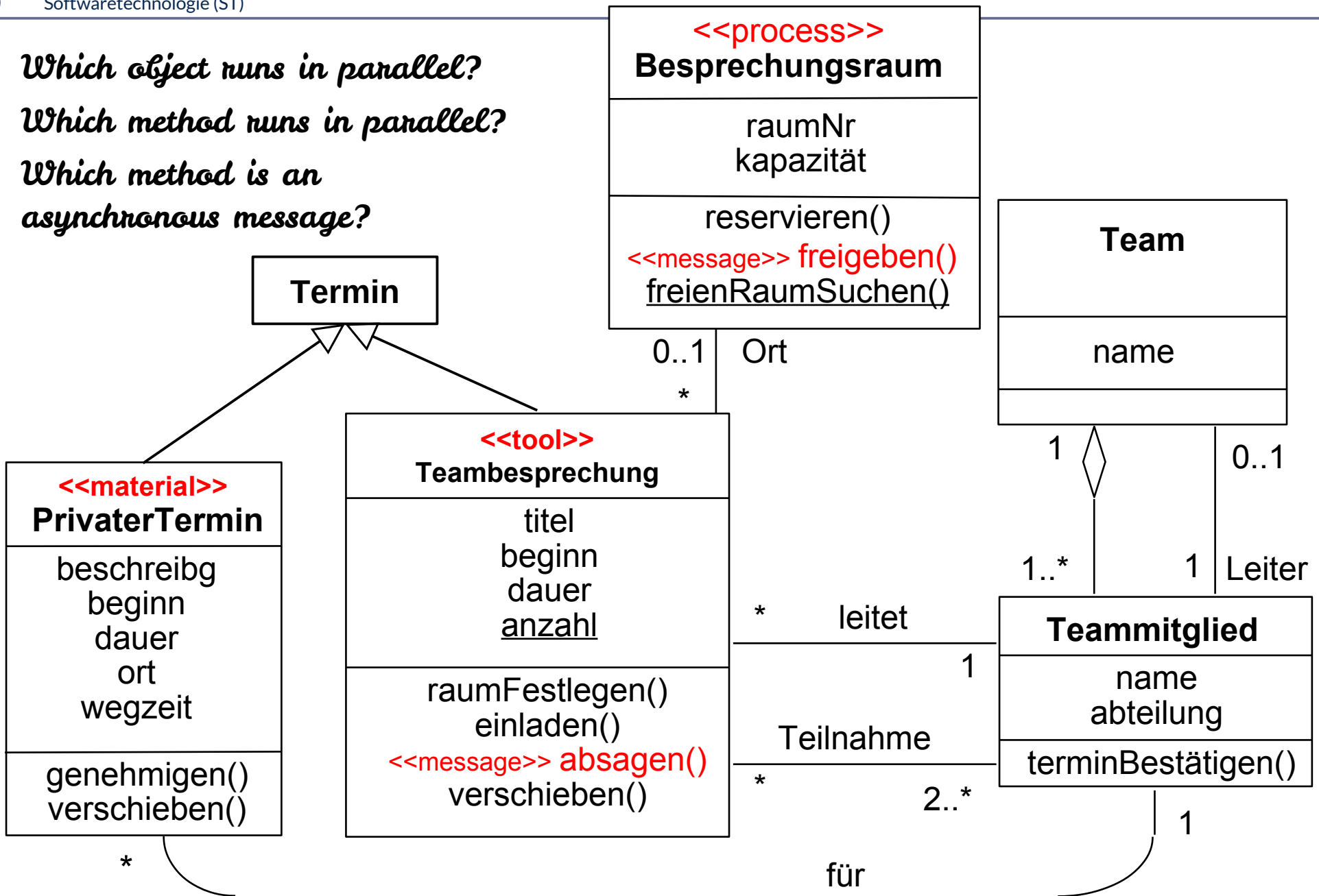


# Einsatz von sequentiellen und parallelen OO Sprachen in der Analyse

- ▶ In der Analyse wird oft **Parallelität** eingesetzt, weil die Prozesse des Domänenmodells parallel zueinander laufen
  - Parallele Objekte arbeiten asynchron
  - sie kommunizieren mit Botschaftenaustausch (messages) und reagieren auf den Empfang einer Nachricht mit dem Ausführen einer (oder mehrerer) Methode(n)
- ▶ In einer **parallelen objekt-orientierten Sprache** setzt sich eine **asynchrone Dienstanfrage (service request)** an ein Objekt zusammen aus:
  - einer **Aufruf-Nachricht (Botschaft, message)**,
  - einer *asynchronen* Ausführung von Methoden (der Sender kann parallel weiterlaufen)
  - optional einer Aufruf-Fertigmeldung (mit Rückgabe), die vom Sender ausdrücklich abgefragt werden muss
- ▶ In einer **sequentiellen objekt-orientierten Sprache** setzt sich eine **synchrone Dienstanfrage** an ein Objekt zusammen aus:
  - einer Aufruf-Nachricht (Botschaft, message),
  - einer *synchronen* Ausführung einer Methoden und
  - einer Aufruf-Fertigmeldung (Aufrufer wartet auf Rückgabe)

# Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell

- ▶ Which object runs in parallel?
- ▶ Which method runs in parallel?
- ▶ Which method is an asynchronous message?

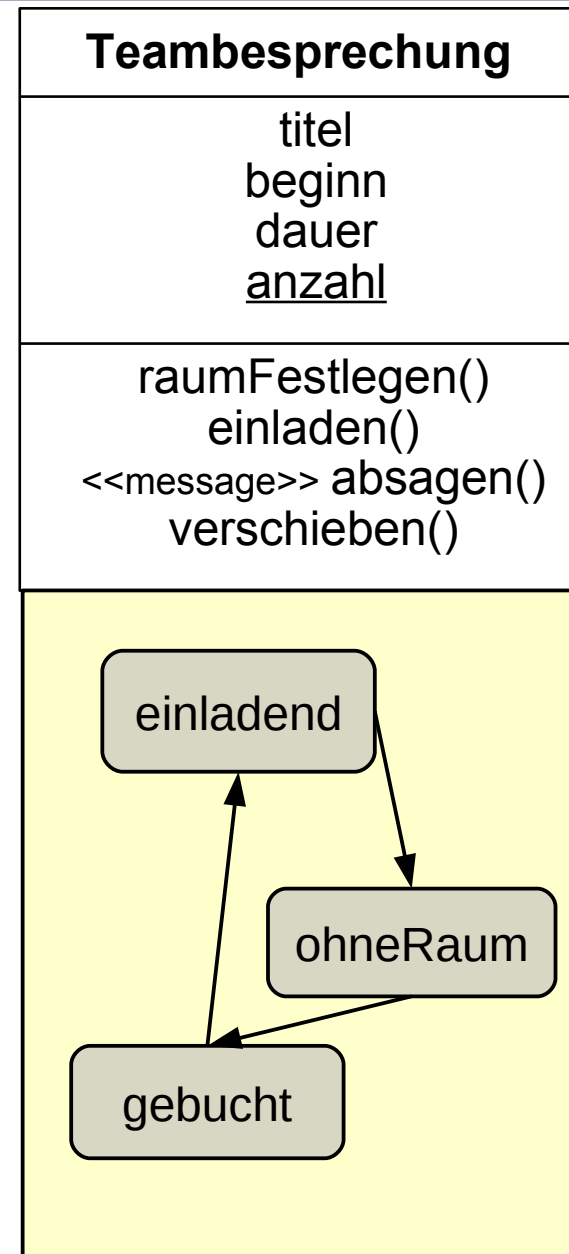


# Abteile (Compartments) von UML-Klassen

41

Softwaretechnologie (ST)

- ▶ Weitere **Abteile (compartments)** in Klassen:
  - Neben dem Attribut- und Operationsabteil können weitere Abteile angehängt werden
  - Verhaltensmodelle können in Abteilen hinzukommen (Statecharts, Aktivitätendiagramme)
  - Mit solchen Verhaltensbeschreibungen können *Objektlebenszyklen* beschrieben werden (siehe später)



## 31.3 Analyse von Klassenbeziehungen und Leserichtungen

- ▶ Texte des Kunden werden in Diagramme umgesetzt und wieder vorgelesen










# Motivation: Gehälter von Requirements-Analysten

- ▶ [https://www.glassdoor.de/Geh%C3%A4lter/requirements-engineer-gehalt-SRCH\\_KO0,21.htm](https://www.glassdoor.de/Geh%C3%A4lter/requirements-engineer-gehalt-SRCH_KO0,21.htm)

Gehälter nach Unternehmen Sortieren: Beliebtheit | Meiste Gehaltsberichte | Gehalt

Nach Standort filtern   oder

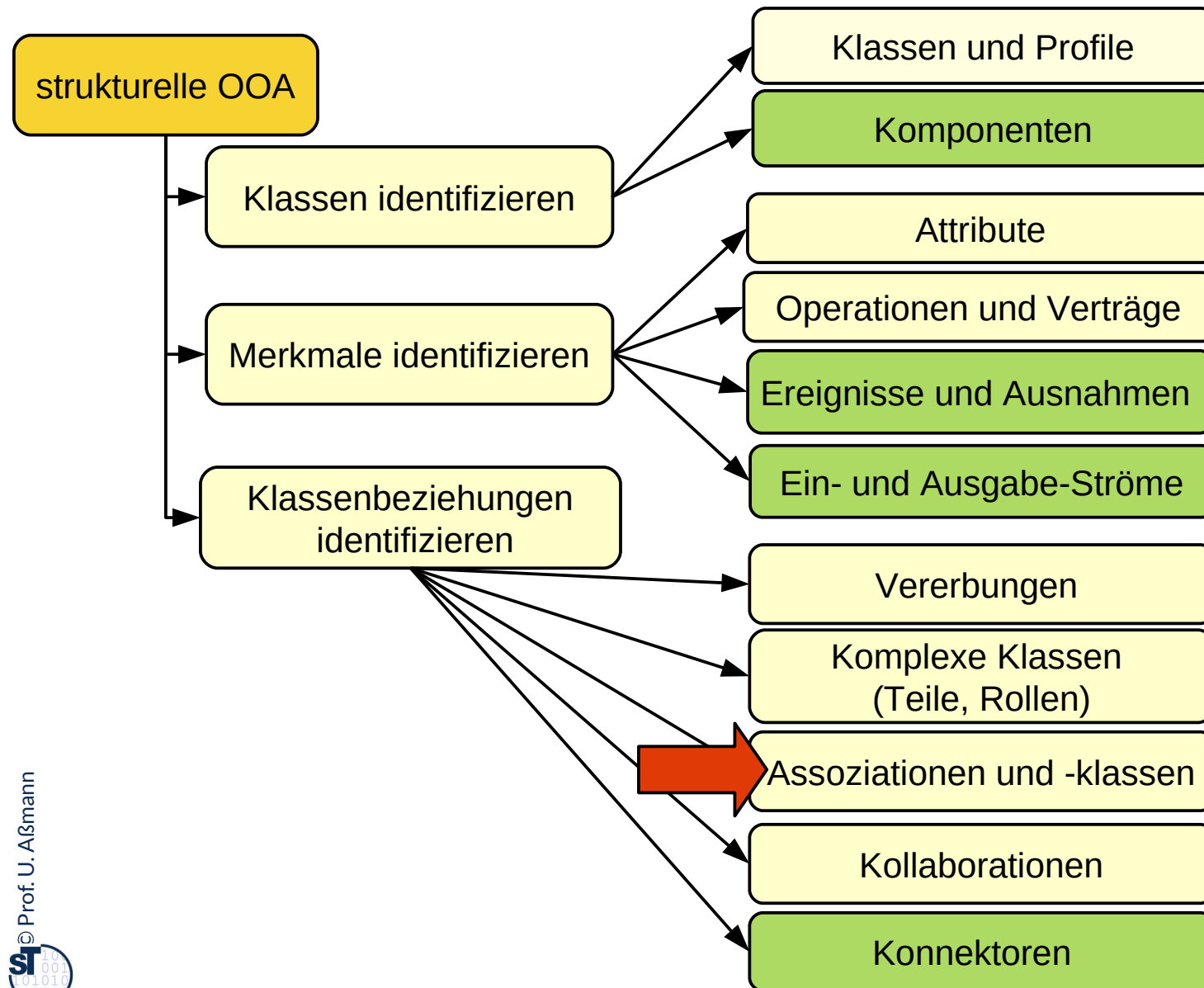
Nach bestimmtem Arbeitgeber suchen

Unternehmen	Durchschn. Grundgehälter (in EUR)	Niedrig	Hoch
 <b>Requirements Engineer</b> Siemens 1 Mitarbeitergehalt oder Schätzwert	Über 61 Tsd € - 66 Tsd €	61 Tsd €	66 Tsd €
 <b>Requirements Engineer</b> Airbus 1 Mitarbeitergehalt oder Schätzwert	Über 50 Tsd € - 54 Tsd €	50 Tsd €	54 Tsd €
 <b>Requirements Engineer</b> Continental 1 Mitarbeitergehalt oder Schätzwert	Über 73 Tsd € - 77 Tsd €	73 Tsd €	77 Tsd €
 <b>Requirements Engineer</b> msg systems (Germany) 1 Mitarbeitergehalt oder Schätzwert	Über 48 Tsd € - 52 Tsd €	48 Tsd €	52 Tsd €
 <b>Requirements Engineer</b> Fraport 1 Mitarbeitergehalt oder Schätzwert	Über 42 Tsd € - 45 Tsd €	42 Tsd €	45 Tsd €
 <b>Requirements Engineer</b> Baker Hughes 1 Mitarbeitergehalt oder Schätzwert	Über 62 Tsd € - 68 Tsd €	62 Tsd €	68 Tsd €
 <b>Requirements Engineer</b> peiker acoustic 1 Mitarbeitergehalt oder Schätzwert	Über 50 Tsd € - 54 Tsd €	50 Tsd €	54 Tsd €



# Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



# Analyse von Assoziationen (Klassenbeziehungen)

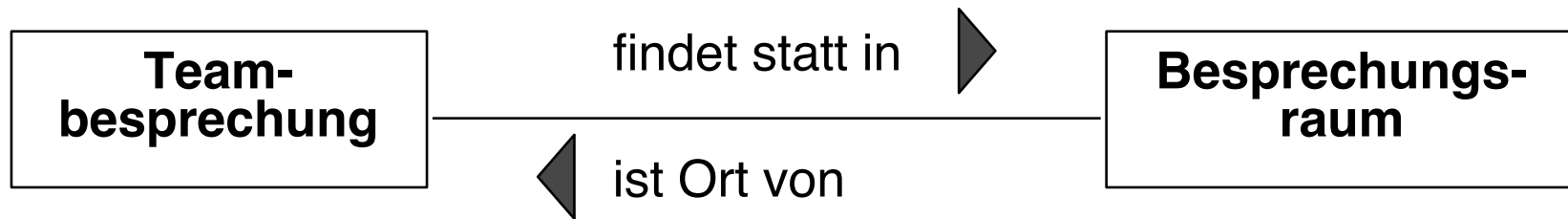
- ▶ Assoziationen entstehen, wenn *Objekte sich kennen*
  - Sie stellen Relationen, Graphen oder Hypergraphen dar, d.h. bilden Abstraktionen von Referenzen
- ▶ **Def.:** Eine (binäre) **Assoziation (Beziehung, relationship) AS** zwischen zwei Klassen  $K1$  und  $K2$  beschreibt, daß die Instanzen der beiden Klassen in einer Beziehung zueinander stehen.
- ▶ **Semantik:** Für jedes Objekt  $O1$  der Klasse  $K1$  gibt es eine individuelle, veränderbare und endliche Menge  $AS$  von Objekten der Klasse  $K2$ , mit dem die Assoziation  $AS$  besteht.  
Analoges gilt für Objekte von  $K2$ .
- ▶ Mathematisch ist dies eine Relation zwischen dem Extent von  $K1$  und dem Extent von  $K2$



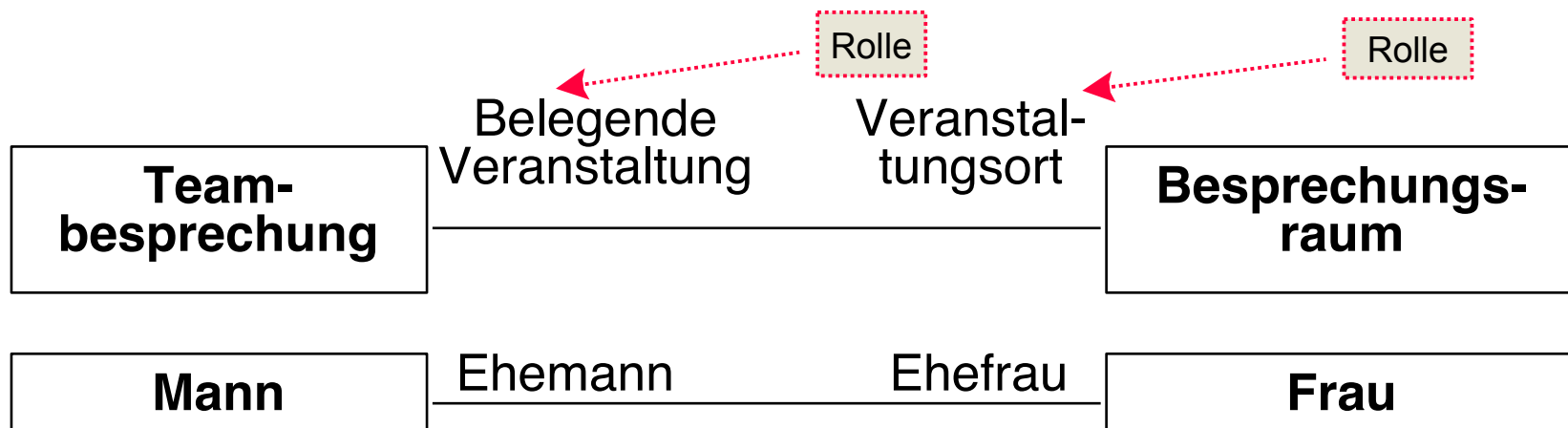
*Welche Assoziationen zwischen Objekten enthält den Text des Kunden?*

# Leserichtung und Assoziationsenden

- ▶ Für Assoziationsnamen kann die **Leserichtung** angegeben werden.
  - Sie wird aus dem Interviewtext entnommen und dazu genutzt, den Text aus dem Modell zu rekonstruieren.

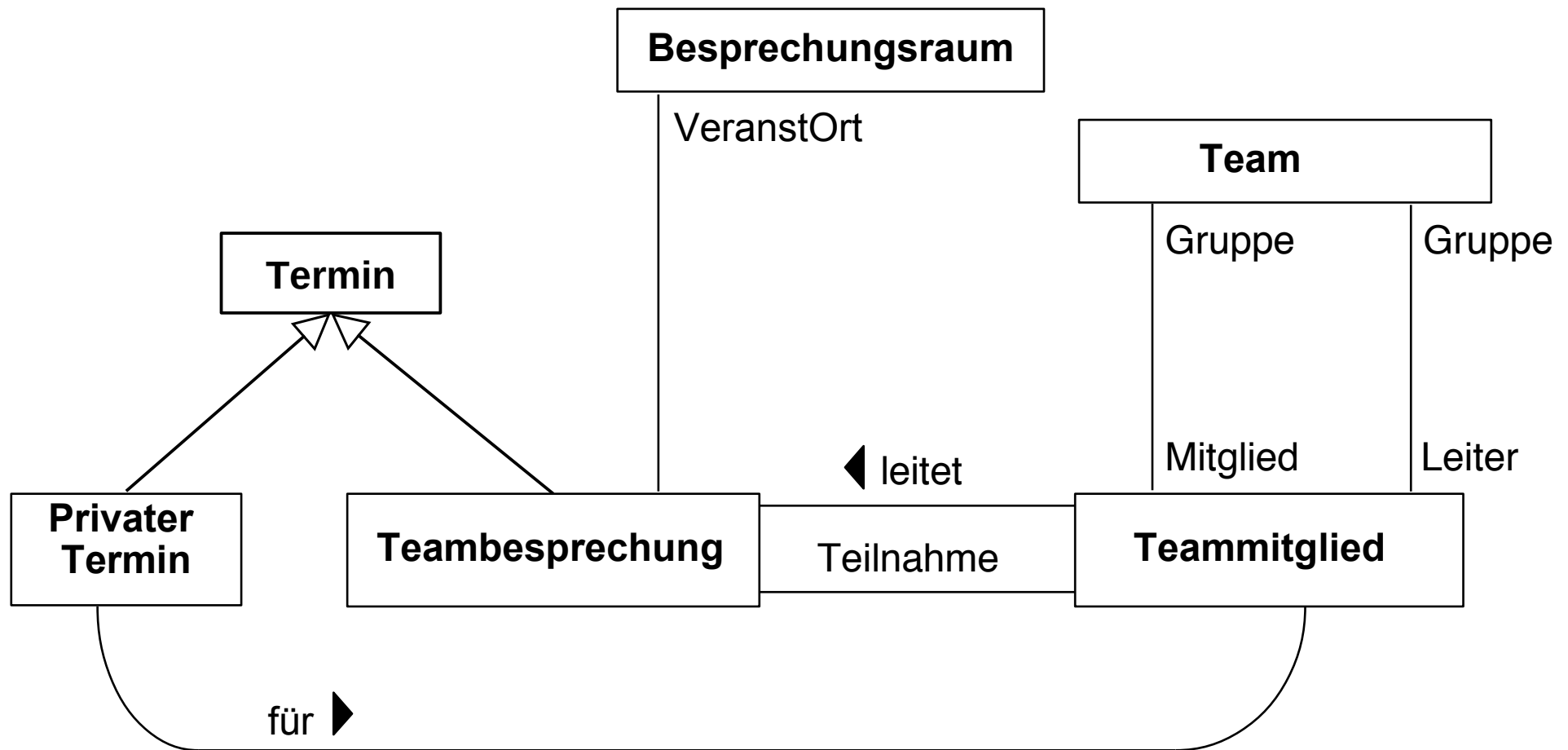


- ▶ Ein Name für ein **Assoziationsende** bezeichnet die Assoziation aus der Sicht einer der teilnehmenden Klassen.
  - Ein Assoziationsende beschreibt die **Rolle** einer Klasse in einer Assoziation



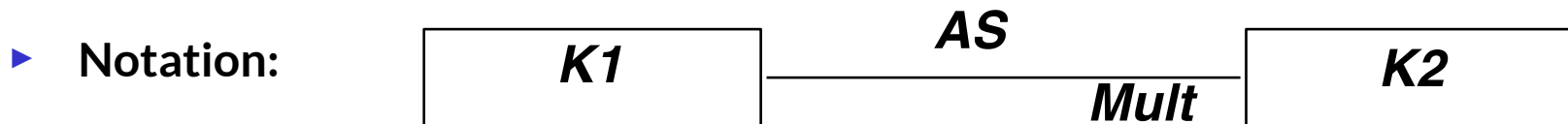
# Beispiel: Das Lesen Assoziationen im Kundentext und im Analysediagramm

- ▶ Die Assoziationen eines Analysediagramms können “gelesen” werden:
  - Ein Team besitzt als Mitglieder die Teammitglieder, als Leiter ein Teammitglied. Teammitglieder gehören zur Gruppe des Teams.
- ▶ Umgekehrt können “user stories” in Analysediagramme umgesetzt werden



# Multiplizität bei Assoziationen

- ▶ **Definition** Die **Multiplizität (Kardinalität)** einer Klasse  $K1$  in einer Assoziation  $AS$  mit einer Klasse  $K2$  begrenzt die Anzahl der Objekte der Klasse  $K2$ , mit denen ein Objekt von  $K1$  in der Assoziation  $AS$  stehen darf. (**Weite** der Relation)



Multiplizität *Mult* :

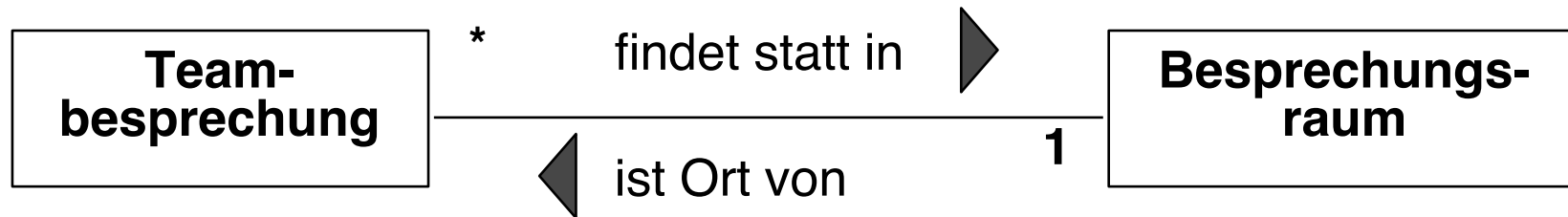
- $n$  (genau  $n$  Objekte der Klasse  $K2$ )
- $n..m$  ( $n$  bis  $m$  Objekte der Klasse  $K2$ )
- $n1, n2$  ( $n1$  oder  $n2$  Objekte der Klasse  $K2$ )

Zulässig für  $n$  und  $m$  :

- Zahlenwerte (auch 0)
- \*
- (d.h. beliebiger Wert, einschließlich 0)



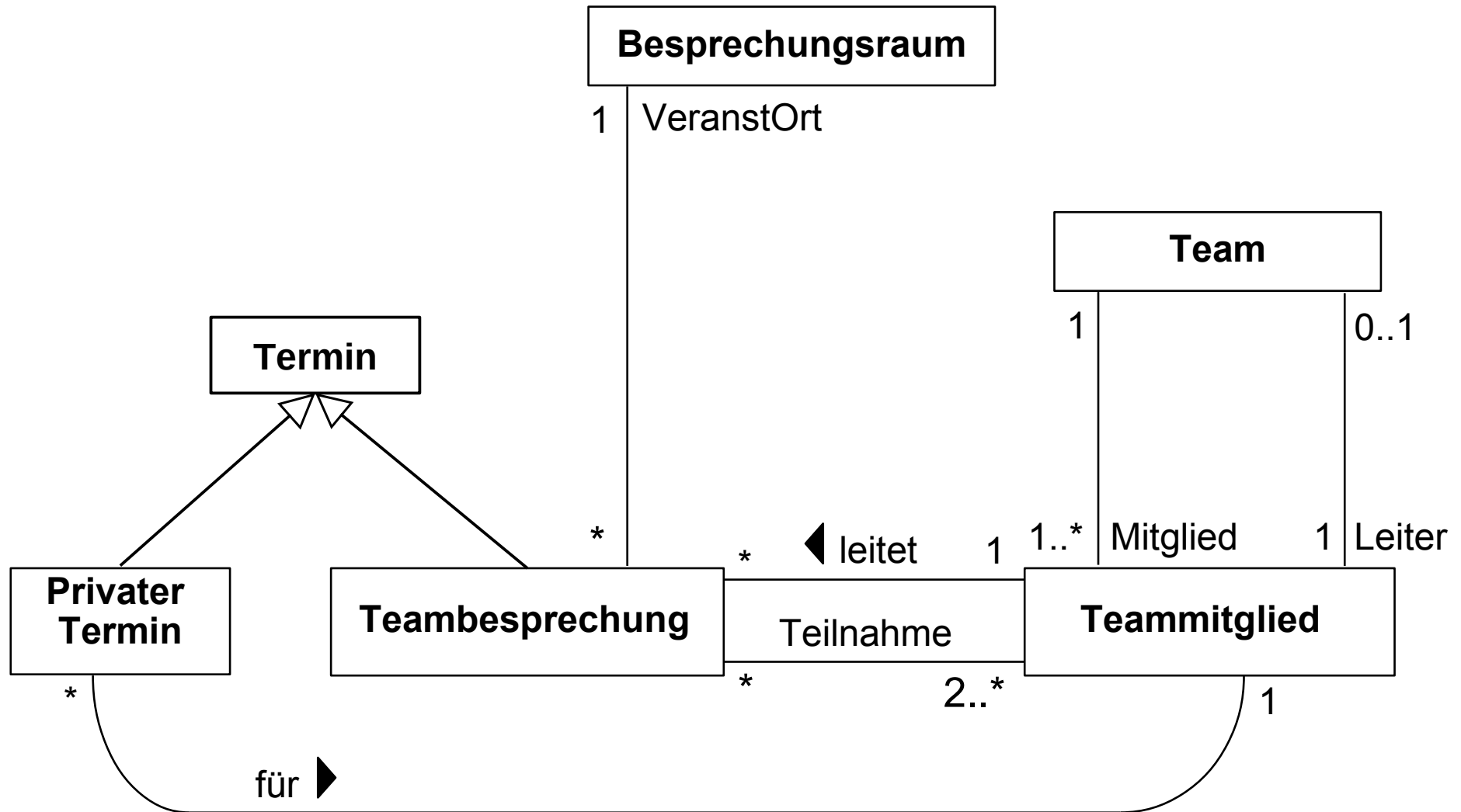
# Multiplizitäten bestimmen durch "Vorlesen"



- ▶ Von links nach rechts:
  - "Jede Teambesprechung findet statt in (wie vielen?) Besprechungsräumen."
  - "Jede Teambesprechung findet statt in genau 1 Besprechungsraum."
- ▶ Von rechts nach links:
  - "Jeder Besprechungsraum ist Ort von (wie vielen?) Teambesprechungen."
  - Jeder Besprechungsraum ist Ort von 0 oder mehreren Teambesprechungen."

Die Multiplizitätsbeschränkung steht an der Klasse, für die die Anzahl der Teilnehmer an der Assoziation beschränkt werden.

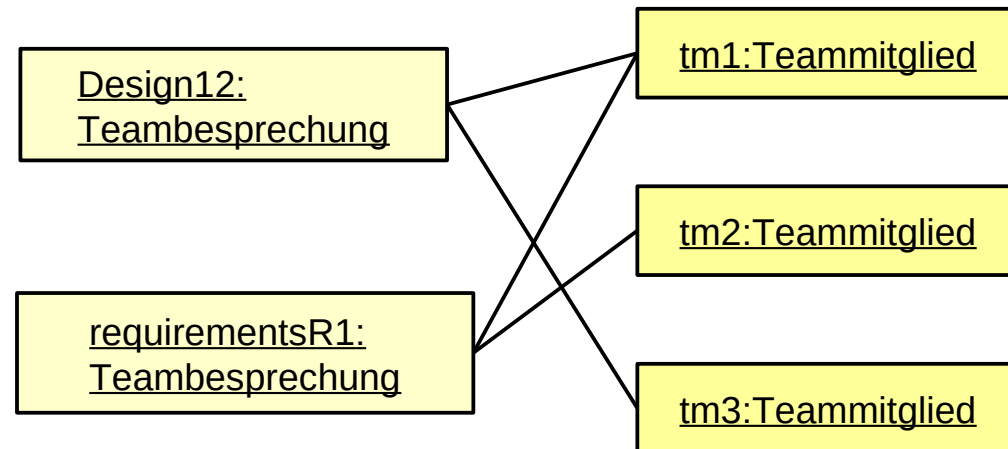
# Beispiel: Erweiterung eines Analysediagramms durch Multiplizitäten



# Semantik (bidirektionaler) Assoziationen

- Ein **Extent (Inhalt)** einer Assoziation (auch *Relation*, *Graph*) ist ähnlich einer *Tabelle*:

Teilnahme-Assoziation	
Teambesprechung	Teammitglied
design12	tm1
design12	tm3
requirementsR1	tm1
requirementsR1	tm2



- Der Extent kann mit einer Graph-Bibliothek wie jgraphxt realisiert werden
- Von einem beteiligten Objekt aus betrachtet, gibt eine Assoziation eine *Menge* von assoziierten Objekten an (**Nachbarmenge**):

Objekt design12: Teambesprechung

Teammitglied-Objekte in Teilnahme-Assoziation: {tm1, tm3}

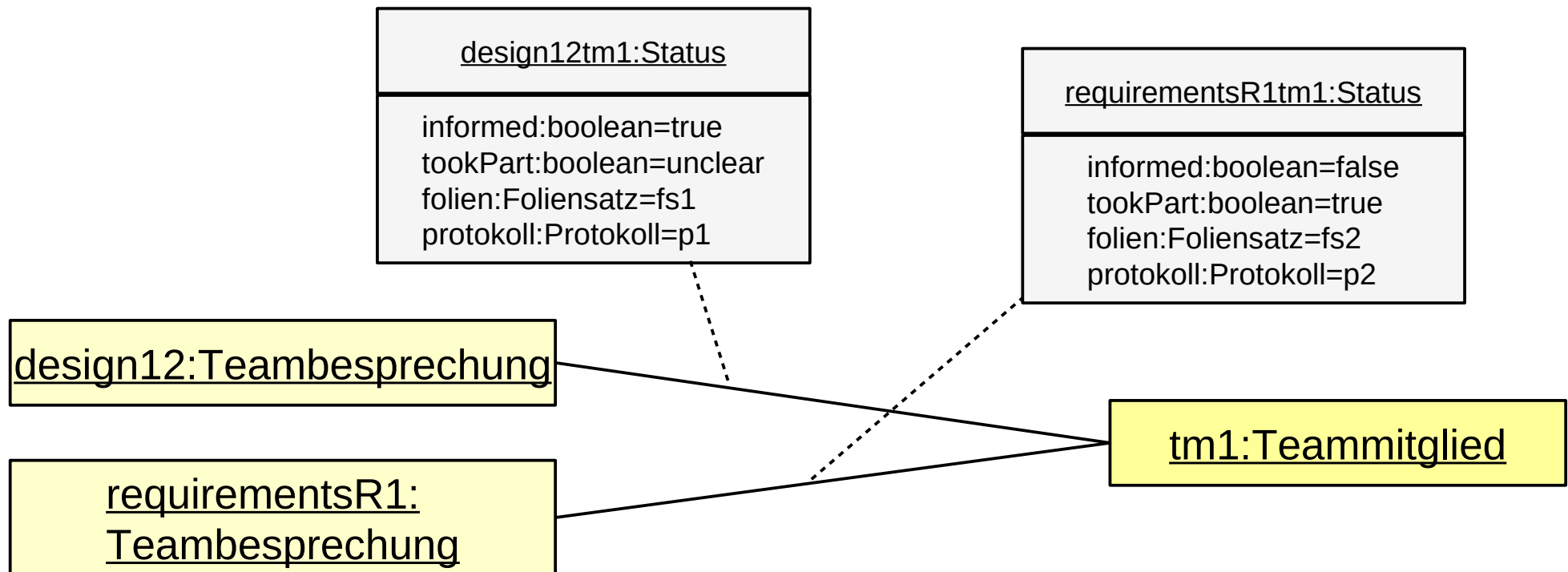
Objekt tm1: Teammitglied

Teambesprechung-Objekte in Teilnahme-Assoziation: {design12, requirementsR1}



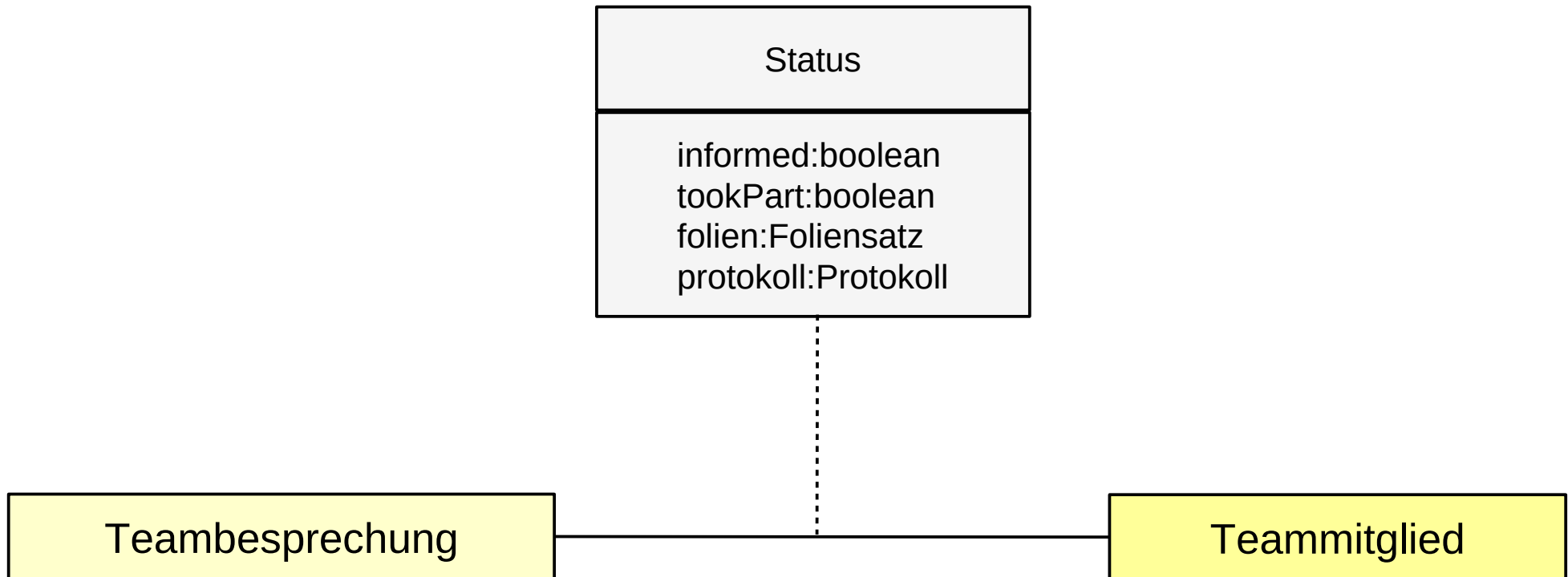
# Assoziationsattribute

- ▶ Oft tragen Assoziationen (Tupel der Relationen, Kanten des Graphen) *Kantenattribute*
  - Diese werden durch *Kantenobjekte* modelliert:

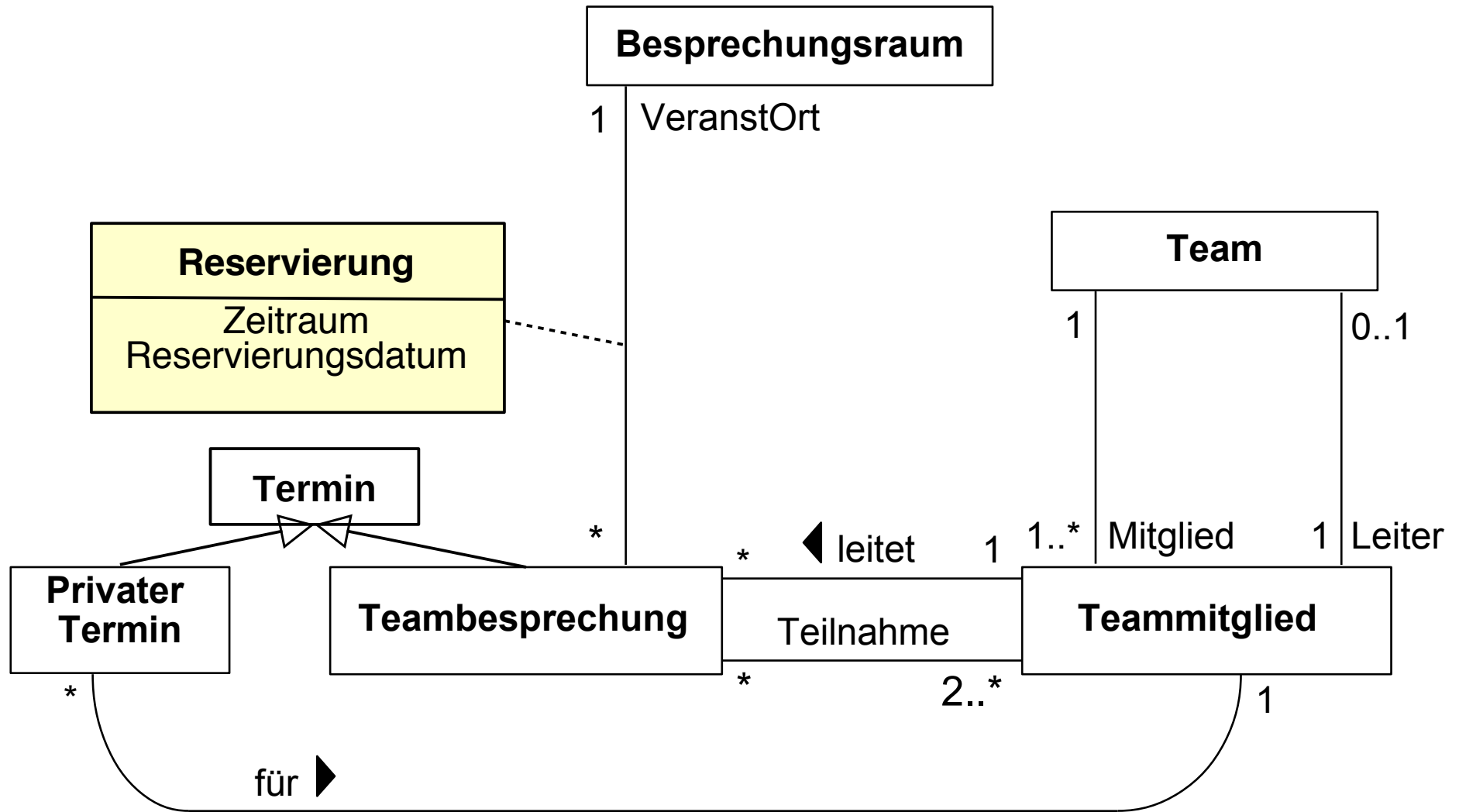


# Assoziationsklassen (Kantenklassen)

- ▶ **Def.:** Eine *Assoziationsklasse (Kantenklasse)* beschreibt die Kantenobjekte einer Assoziation.
- ▶ Assoziationsklassen werden benötigt, wenn Wissen dargestellt werden soll, das für jede Kante (Tupel) *unterschiedlich* ist, d.h. Wissen über die Assoziation der Objekte darstellt
- ▶ In jgrapht entsprechen Assoziationsklassen einer Unterklasse von DefaultEdge



# Beispiel: Erweiterung des Analysediagramms mit einer Reservierung als Assoziationsklasse



## 31.3.2 Realisierung von Assoziationen

- ▶ Assoziationen beschreiben Objektnetze beliebig großer Art.
- ▶ Wir nutzen jetzt das in Abschnitt II Gelernte, um die Abbildung von UML-Assoziationen auf Java zu verstehen



# Realisierung von Assoziationen

- ▶ 1) Realisierung durch **Graphen** einer Graph-Bibliothek (siehe Kap. 23)
- ▶ 2) Realisierung durch **zwei gerichtete Assoziationen** (siehe Kap. 21)
  - Redundanz: zusätzlicher Speicheraufwand, Gefahr von Inkonsistenzen
  - Aber: schnelle Navigation
- ▶ 3) Realisierung nur in **einer Richtung**: (Anhang, siehe Kap. 21)
  - Gibt nicht die volle Semantik des Modells wieder
  - Abhängig von Benutzung (Navigation) der Assoziation
- ▶ 4) Realisierung durch **Assoziationsklassen** bzw. Zwischenklassen, die Assoziationen repräsentieren: (Anhang)
  - Geeignete Datenstrukturen erforderlich
    - "Beidseitige" Abfragemöglichkeit
    - Abflachung zu normalen Klassen nötig
- ▶ 5) Realisierung durch **Konnektoren** (siehe Kapitel Konnektoren)
- ▶ 6) Realisierung durch **explizite Rollenklassen** (Anhang)
- ▶ 6) Realisierung durch **Tabellen in einer relationalen Datenbank**
- ▶ **Genauere Entscheidung erst im Entwurf !**

# Realisierung von Assoziationen

57 Softwaretechnologie (ST)

Höhere  
Sprachen

Graphen als Sprachkonstrukte

Graphen als Konnektoren (Rollen-Kollaborationen)

Fixe Netze als Entwurfsmuster

Fixe Netze mit Datenfluss mit Konnektoren

Graphen als Bibliotheken (Java)

Graphen als Collections abgeflacht (Java)

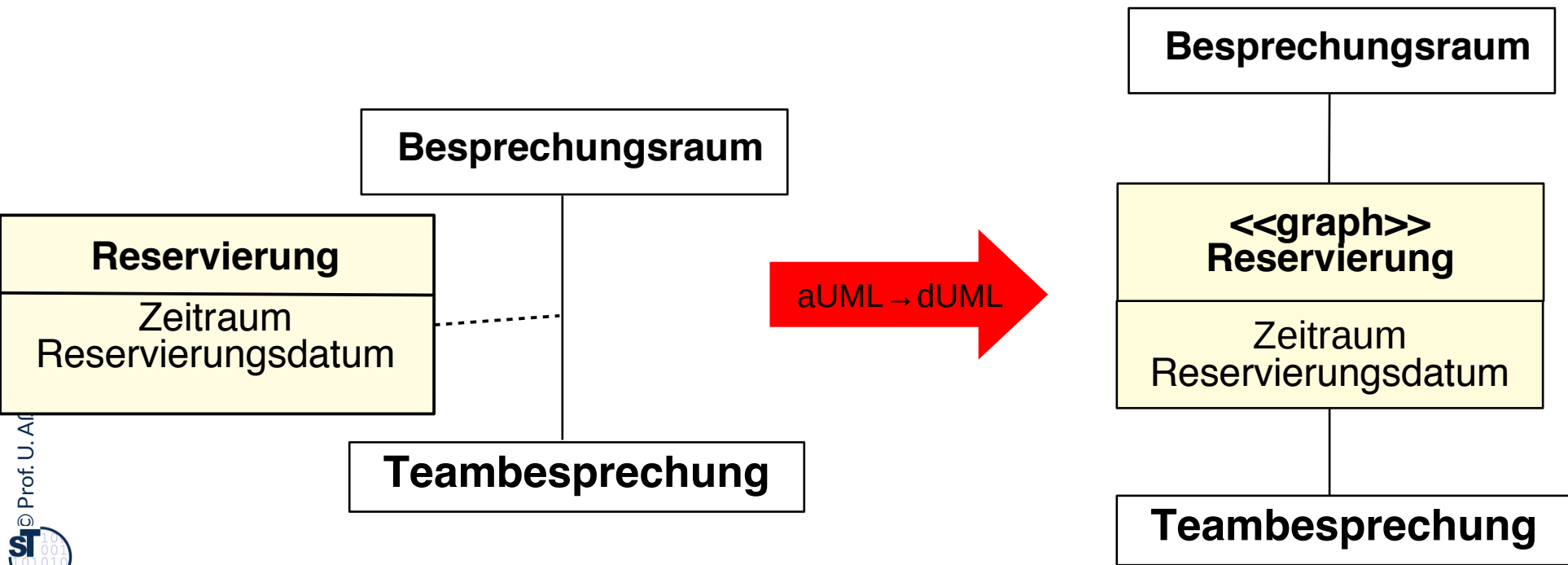
Graphen als Endoassoziationen:  
Netze von Unterobjekten in komplexen Objekten (Java)

System-  
programmier-  
sprache (C)

Graphen als Datenstrukturen fester Länge abgeflacht

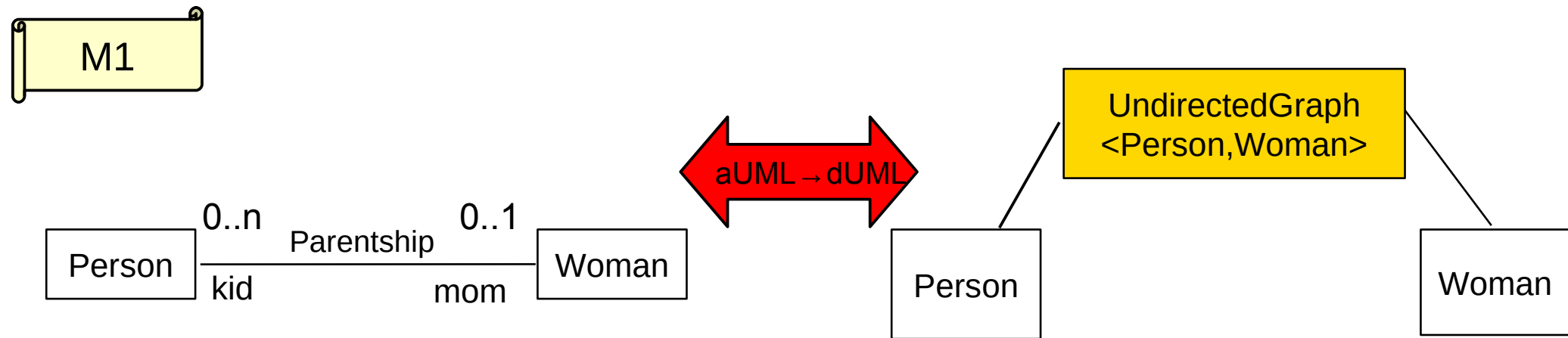
# 31.3.2.1 Realisierung von bidirektionalen Assoziationen durch Graphklassen

- ▶ Assoziationsklassen aus dem Analysemodell können im Implementierungsmodell in Graphklassen abgeflacht werden (Kap. 23):

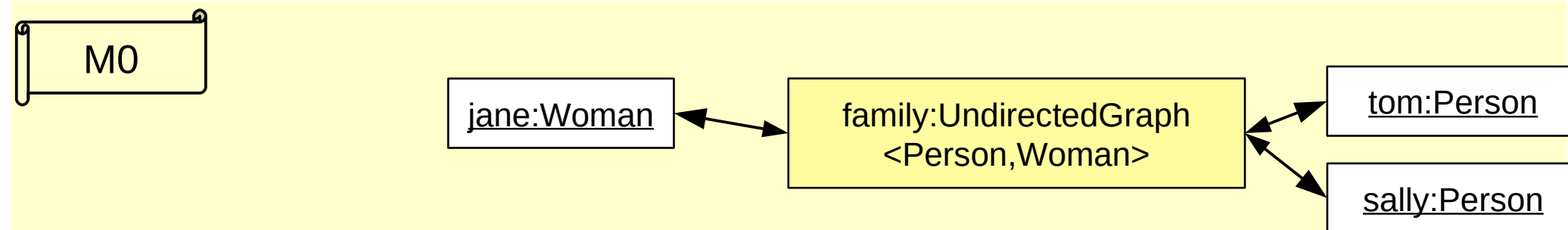


# Beispiel: Realisierung von bidirektionalen Assoziationen durch Graphklassen

- ▶ Auch bidirektionale Assoziationen können durch Graphklassen realisiert werden (Kap. 23)



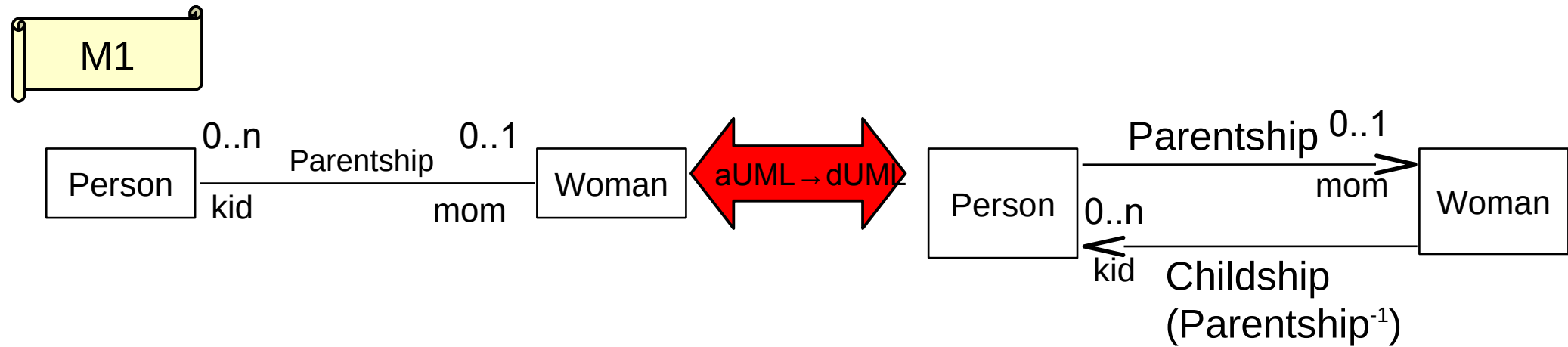
Laufzeit: zwei Referenzen zwischen Graphobjekt und den assoziierten Objekten



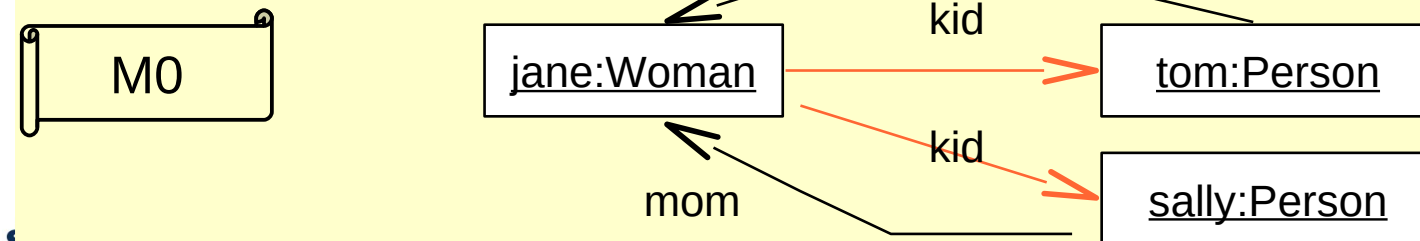


# 31.3.2 Realisierung von bidirektionalen Assoziationen durch unidirektionale

- ▶ Bidirektionale Assoziationen können, falls sie keine Attribute tragen, in gerichtete Assoziationen umgewandelt werden (Kap. 21-collections)
- ▶ Realisierung im Implementierungsmodell in jUML durch Einführung von *gerichteten* Assoziationen



Laufzeit:

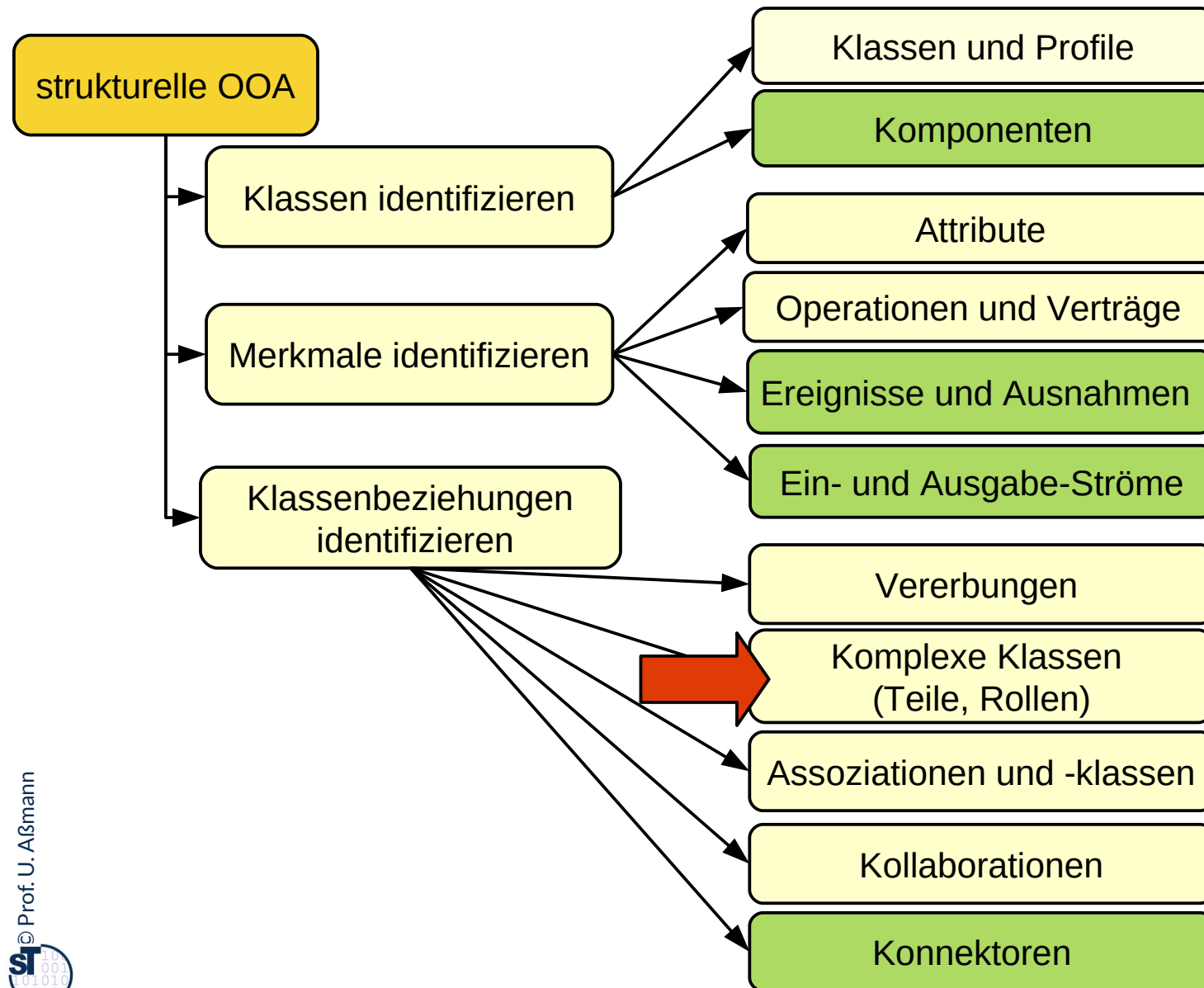


# 31.4 Modellierung von Hierarchien und komplexen Objekten (Verbundobjekten, Teilehierarchien)



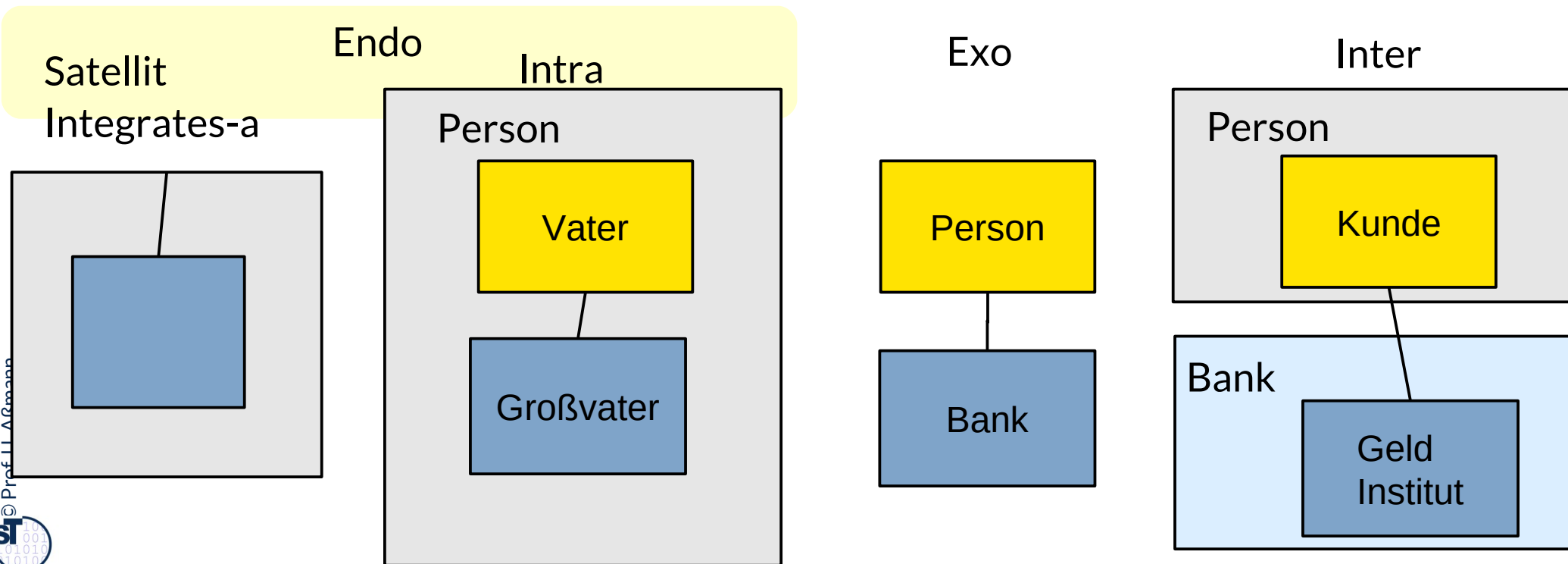
# Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



# Endo- vs. Exo-Assoziation

- ▶ Komplexe Objekte (Verbundobjekte) verwenden Assoziationen inner- und außerhalb ihrer selbst.
- ▶ Frage: "Gehören die beiden Objekte zu einem Ganzen"? (logische Einheit, physische Getrenntheit)
- ▶ **Endo-Assoziation:** Assoziation innerhalb eines komplexen Objektes
  - **Intra-Assoziation:** beide werden von einem dritten umschlossen
  - **Satellit-Assoziation (integrates-a):** einer der Partner umschließt den anderen.
    - "Teil" ist eine Satellit-Endo-Assoziation vom Ganzen zum Teil
- ▶ **Exo-Assoziation:** keiner der Partner umschließt den anderen.
  - **Inter-Assoziation:** jeder der Partner wird von einem anderen umschlossen

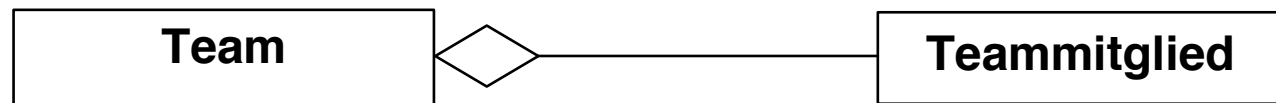


## 31.4.1 Aggregation, Komposition, Rollenspiel als Endo-Assoziationen



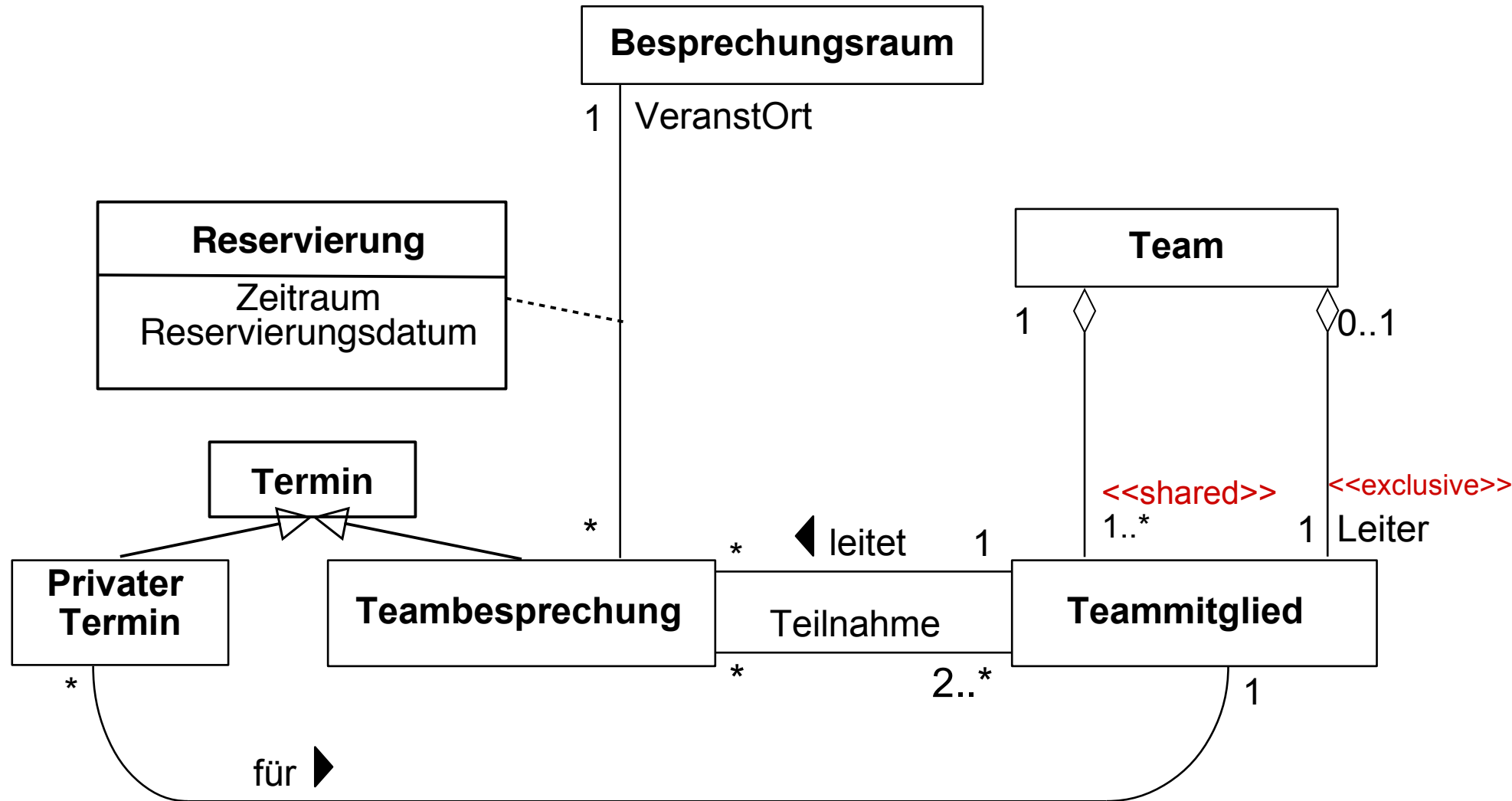
# Wdh: Aggregation (has-a)

- ▶ **Definition:** Wenn eine Assoziation den Namen „hat-ein“ oder "besteht-aus" tragen könnte, handelt es sich um eine **Aggregation** (Ganzes/Teile-Relation).
  - Eine Aggregation besteht zwischen einem *Aggregat*, dem *Ganzen*, und seinen *Teilen* (**Endo-Assoziation**)
  - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklensfreien Graphen, *dag*)
  - Ein Teil kann
    - zu mehreren Ganzen gehören (*<<shared>>*),
    - zu einem Ganzen (*<<owns-a>>*) und
    - exklusiv zu einem Ganzen (*<<exclusively-owns-a>>*)



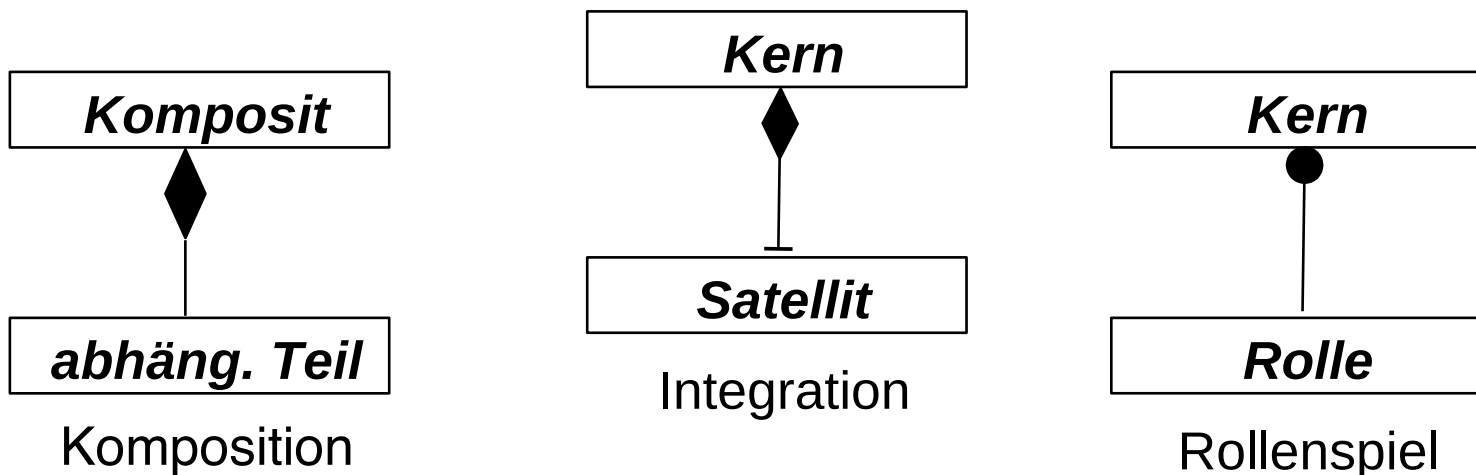
Lies: „Team hat ein (wechselbares) Teammitglied“

# Beispiel: Erweiterung eines Analysediagramms mit Aggregationen



# Komposition (owns-a)

- ▶ **Definition:** Ein Spezialfall der Aggregation ist die **Komposition** zwischen einem **Komposit** und seinen **Teilen**.
  - Ein Objekt kann Teil höchstens eines Komposits sein (Eigentums-Relation *exclusively-owns-a*).
  - Das Teil ist *abhängig* vom Komposit (*dependent part*).
    - Das Komposit hat die alleinige Verantwortung für Erzeugung und Löschung seiner Teile (gleiche Lebenszeit)
- ▶ Def: Aggregation, Komposition und Rollenspiel sind Spezialfälle der **Integration** von Unterobjekten als Satellit eines Kernobjekts (*integrates-a*)





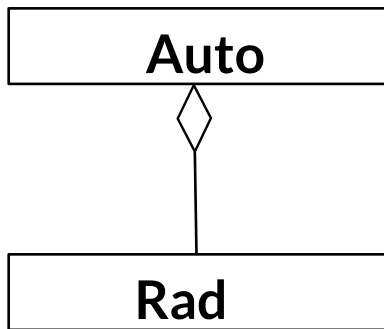
# Beispiele von Endo-Assoziationen

*„Welches wechselbare Teil hat das Objekt?“*

*„Welches dauerhafte Teil hat das Objekt?“*

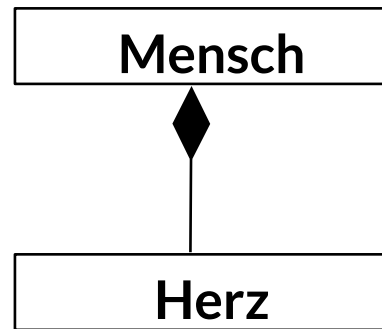
*„Welche Rolle spielt das Objekt?“*

Aggregation



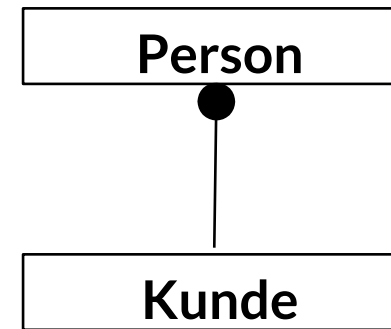
„Ein Auto hat ein Rad“

Komposition

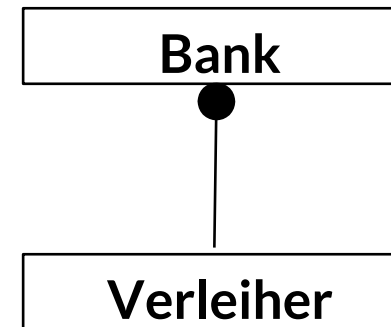
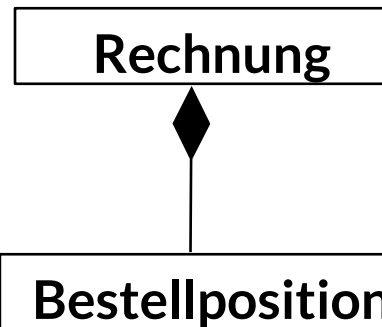
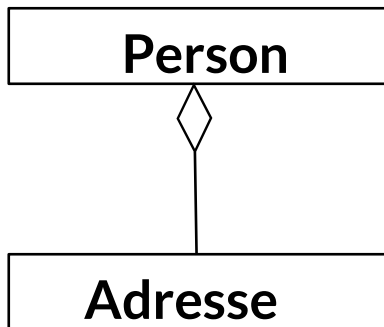


„Ein Mensch braucht ein Herz“

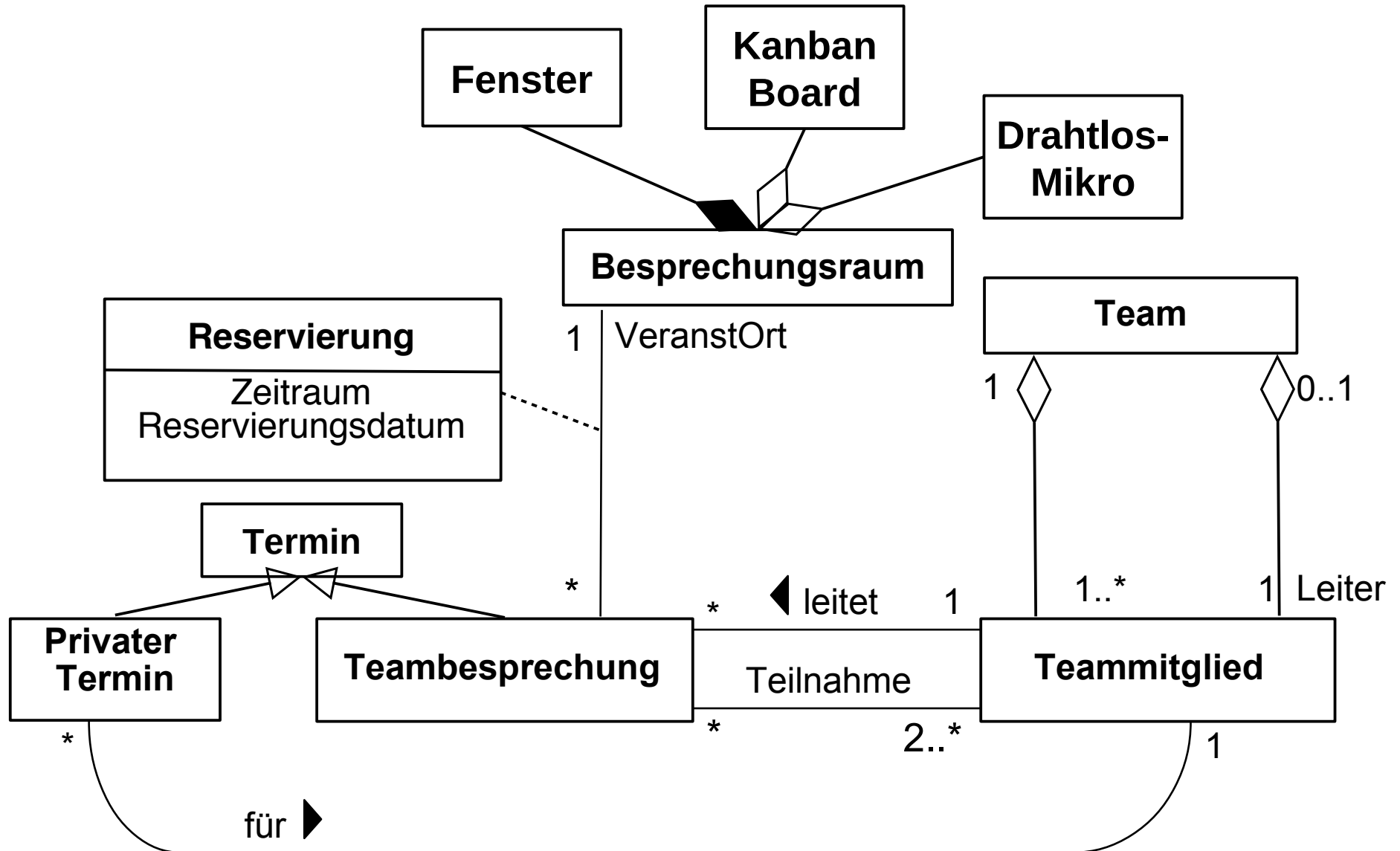
Rollenspiel



„Eine Person spielt einen Kunden“



# Beispiel: Erweiterung des Analysediagramms mit Kompositionen und Aggregationen



# Darstellung von Hierarchien und kompositen Objekten

- ▶ Hierarchien und komposite Objekte (whole-part hierarchies) mit ihren Endo-Assoziationen können mit visuellen Darstellungen für Bäume dargestellt werden:
  - Baum
  - Venn-Diagramm
  - Zeilenhierarchie (Tree-Widget)
  - Mind map
- ▶ Alle Darstellungen sind äquivalent und können ineinander umgewandelt werden

Als Baumrelation können verschiedene Relationen dienen:  
Vererbung, Teil, Aufruf, Operatoren, Unterobjekte (s. später)

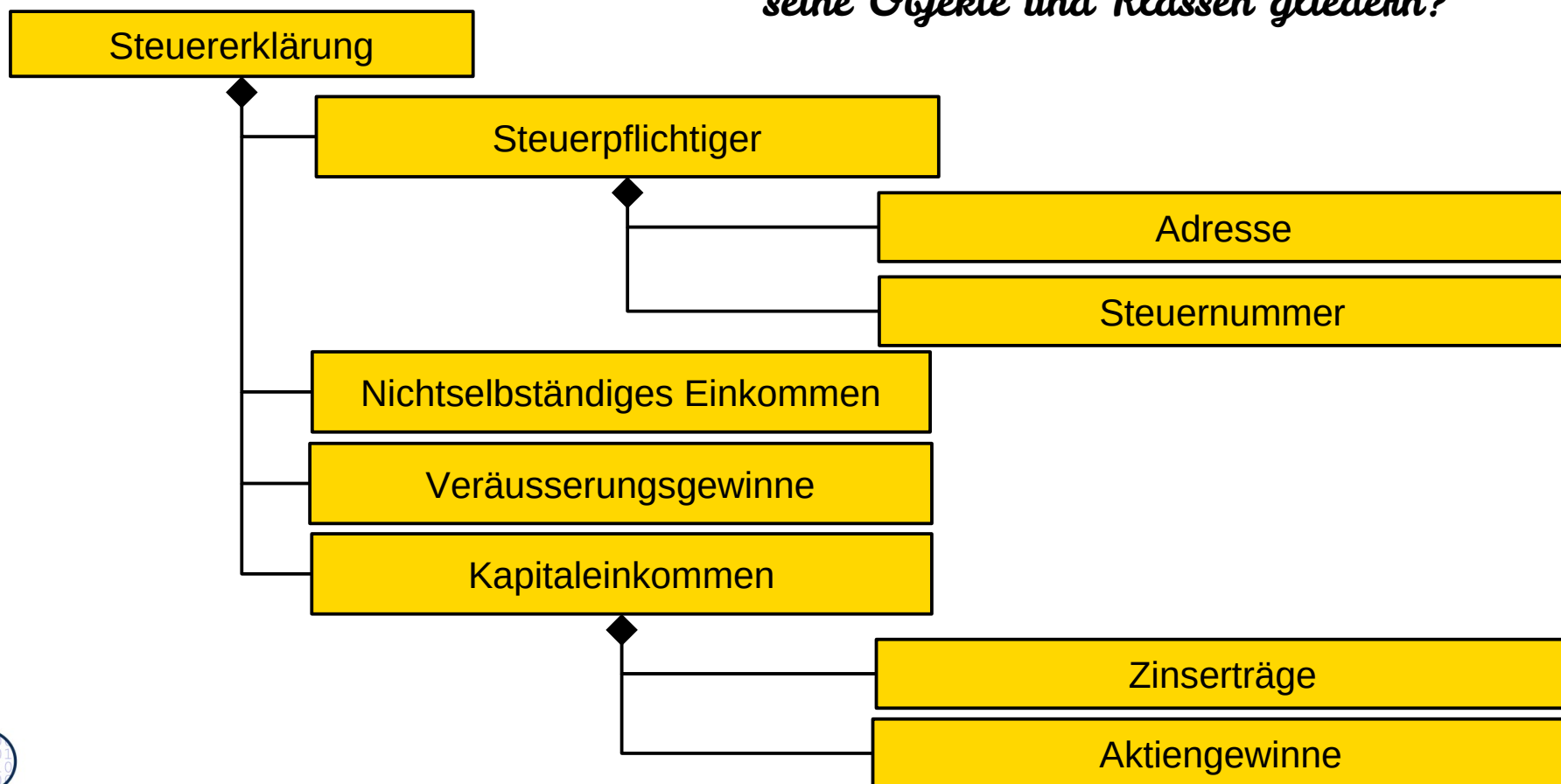
# Ganzes/Teile-Zeilenhierarchie (whole-part hierarchies)

71

Softwaretechnologie (ST)

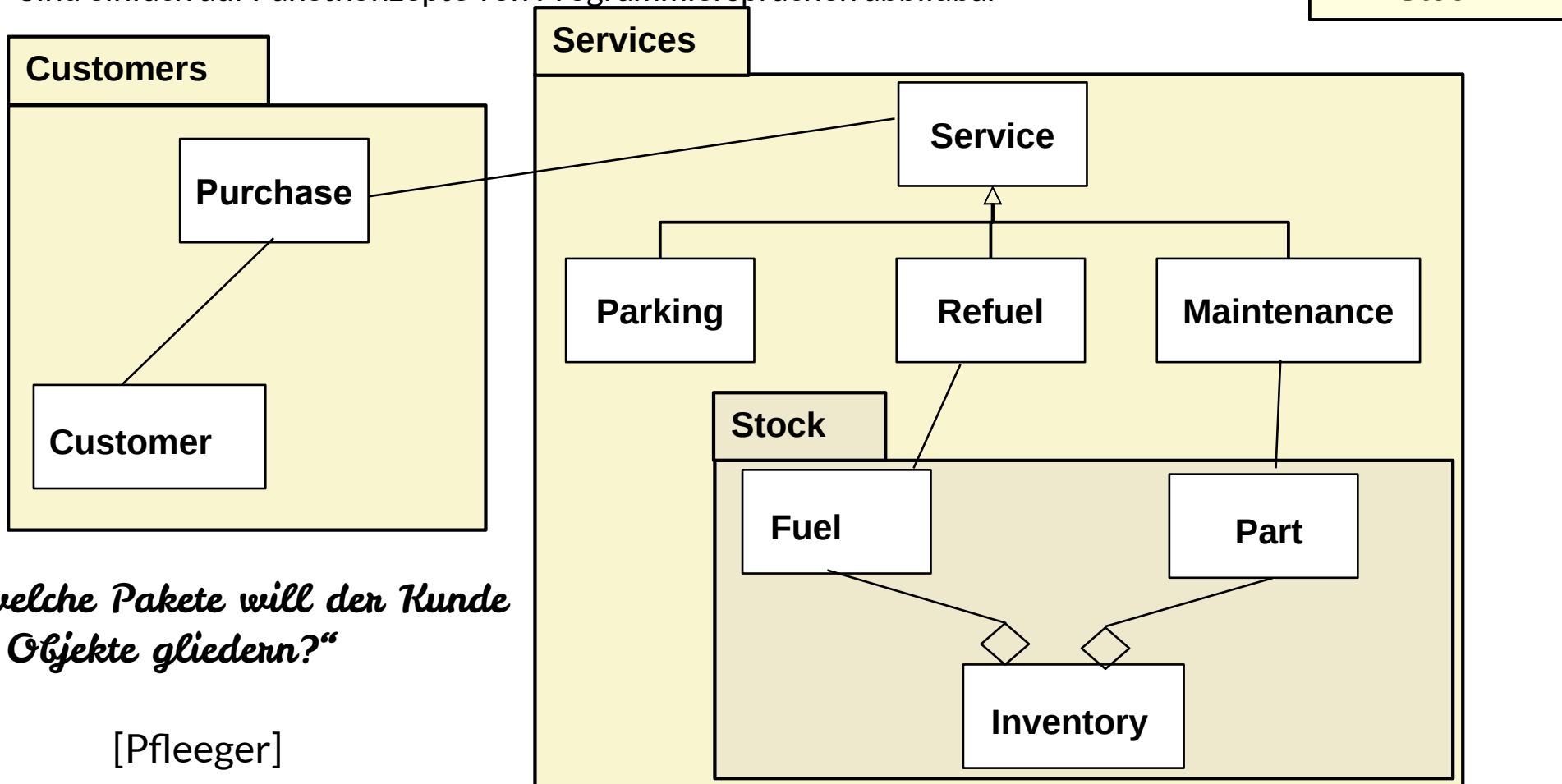
- ▶ Komplexe Objekte bestehen oft aus Ganz-Teile-Hierarchien (owns-a-Hierarchien)
- ▶ Man kann Komplexe Objekte als auch als **Aggregations-** oder **Kompositionshierarchien** anzeigen
  - Ganzes/Teile-Hierarchien können mit Klassen oder auch Objekten gezeichnet werden

*„In welche hierarchische Struktur will den Kunde seine Objekte und Klassen gliedern?“*



# Bsp: Pakete und ihre Aggregationsrelationen in UML

- ▶ Ein **UML-Paket** fasst eine Menge von UML-Diagrammfragmenten zusammen.
- ▶ Klassen sind *flach*, Pakete dagegen *hierarchisch* strukturierbar
  - Pakete gruppieren Klassen, Objekte, andere UML-Fragmente
- ▶ UML-Paketrelationen sind hierarchisch oder azyklisch
  - Sind einfach auf Paketkonzepte von Programmiersprachen abbildbar

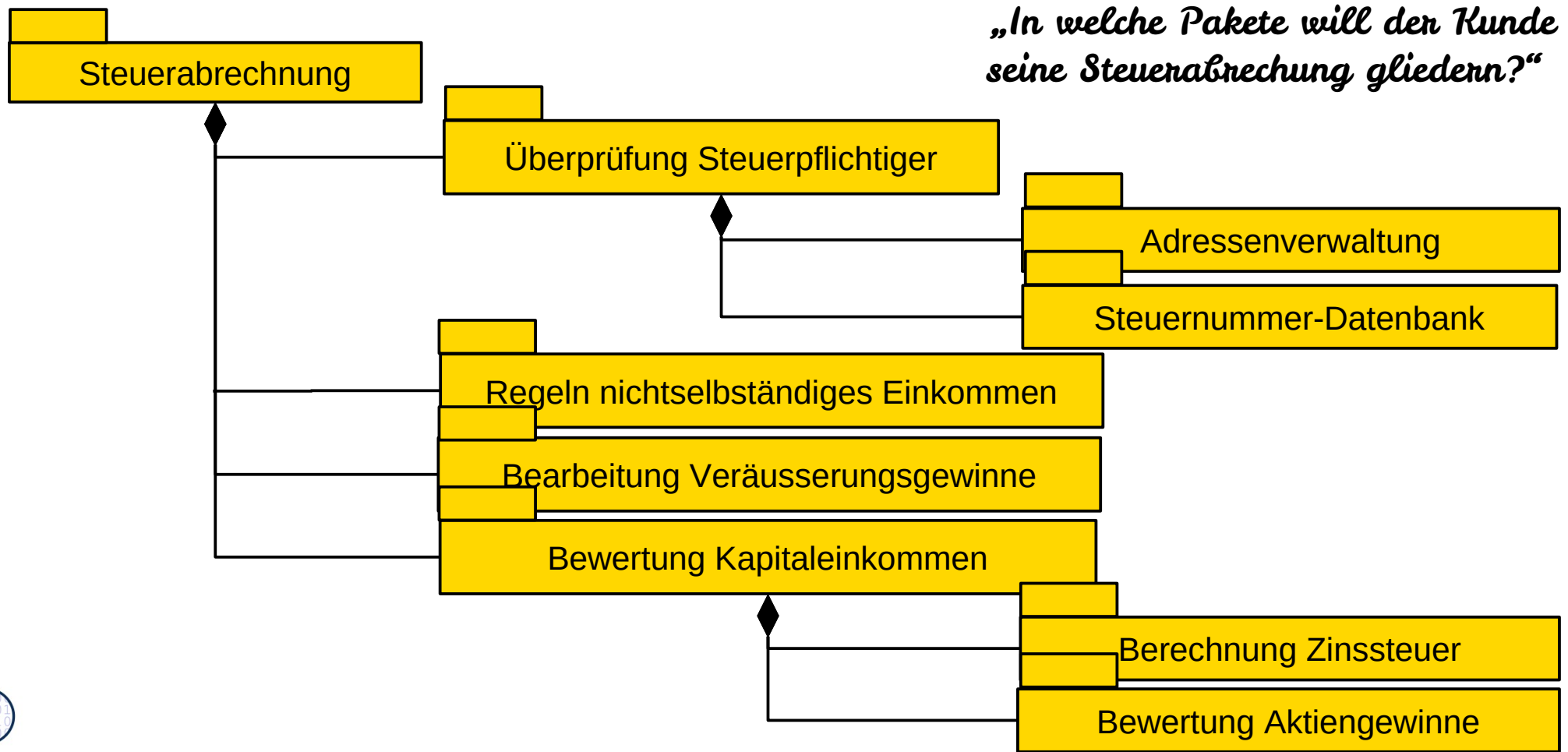


„In welche Pakete will der Kunde seine Objekte gliedern?“

[Pfleeger]

# Paket-Zeilenhierarchie

- ▶ Auch Paket-Hierarchien kann man als auch als Zeilenhierarchien (Textbäume) anordnen
- ▶ Operator ist die Paketvereinigung
- ▶ Pakete sind Container von UML-Fragmenten (z.B. von Klassen)



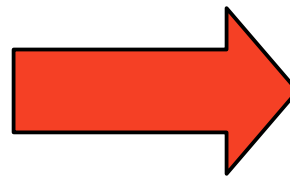
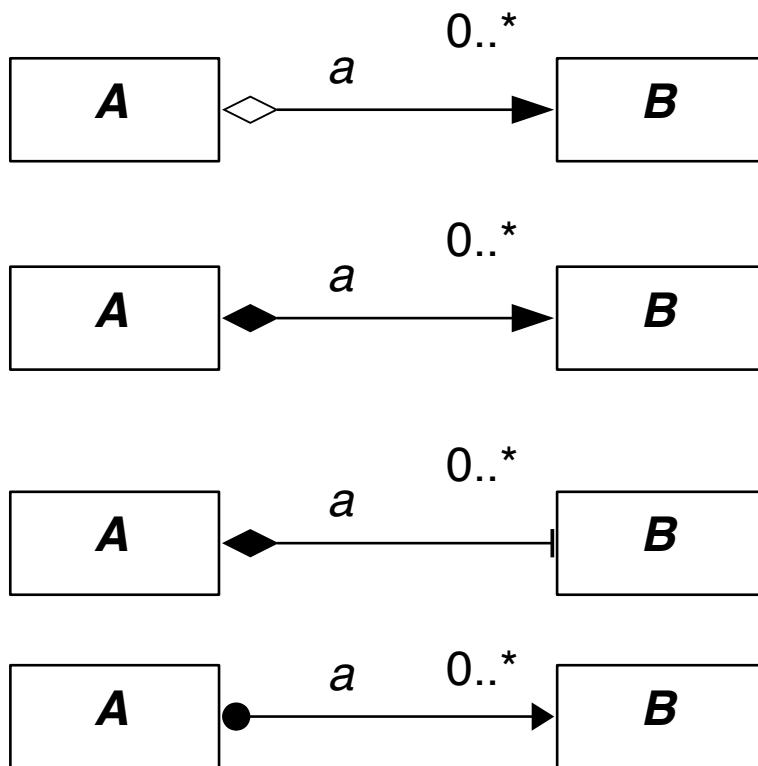
## 31.4.2 Realisierung von komplexen Analyseobjekten mit ihren Endo-Assoziationen

- ▶ Siehe auch Kap. “Collections” und “Graphen”



# Realisierung von Aggregation, Komposition und Rollenspiel in Java

- ▶ Aggregationen stellen azyklische Graphen dar und können daher genau wie Assoziationen abgebildet werden
  - Komposition und Rollen werden in gleicher Weise abgebildet
- ▶ Daher gehen in Java die Analyseinformationen verloren!
- ▶ Überlege auch den Einsatz der Entwurfsmuster Decorator (für Listen) und Composite (für Bäume)



```
class A {
    ...
    Collection<B> a;
    // B[] a;
    ...
}
```



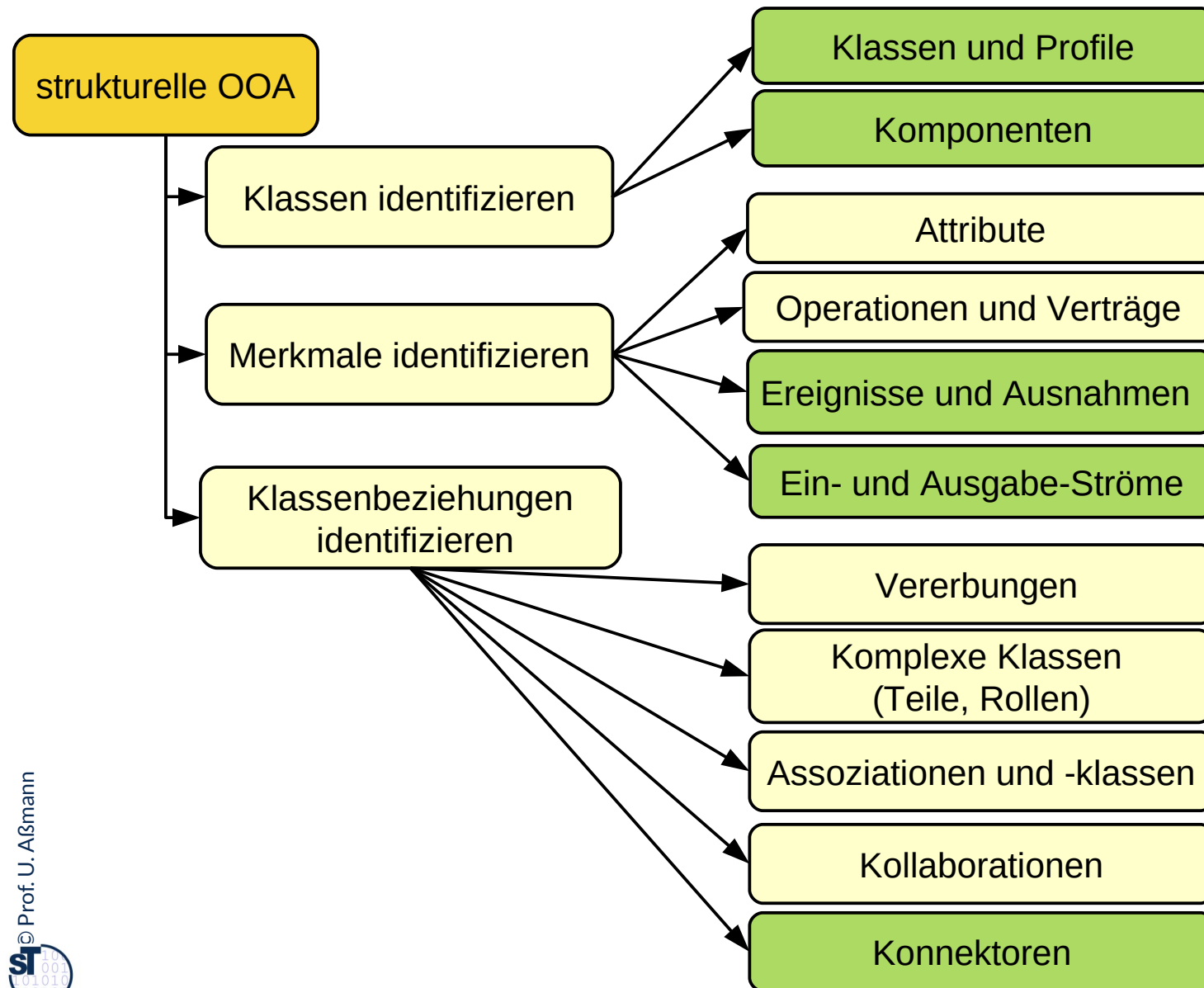
## 31.4.3 Modellierung von komplexen Analyseobjekten aus Kernen und Unterobjekten

- ▶ Große Objekte bestehen aus Kernen und Unterobjekten, die mit Endoassoziationen verbunden sind



# Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

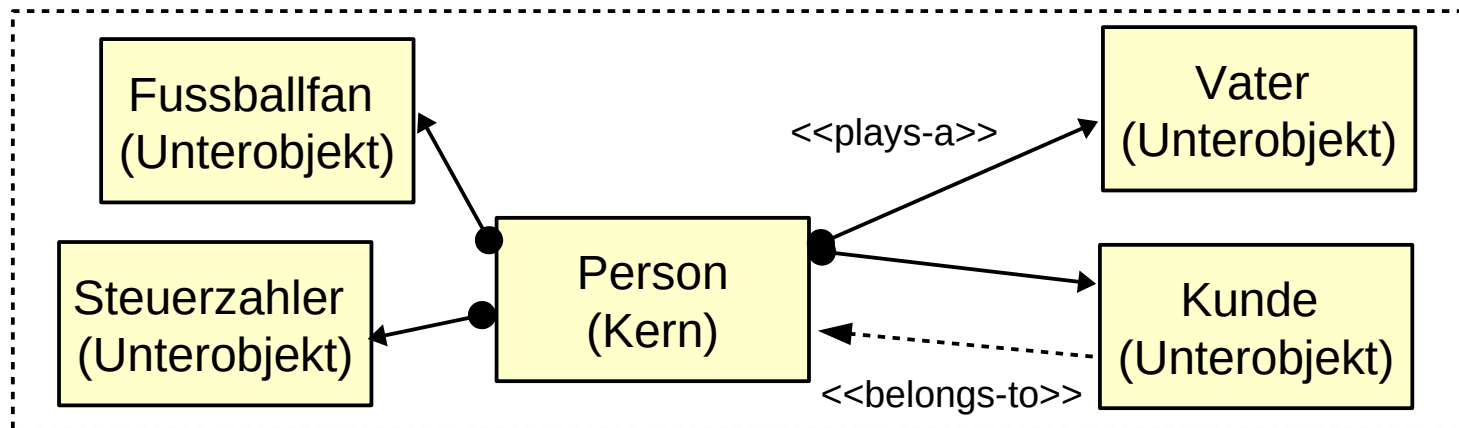
- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



**Def.:** Ein **komplexes Objekt (Subjekt, Großobjekt, complex object, big object)** ist ein Objekt, das auf Programmierniveau wegen seiner Komplexität durch *ein Kernobjekt und mehrere Unterobjekte (Teilobjekte)* dargestellt wird.

Seine innere Struktur ist meist hierarchisch, immer aber azyklisch angelegt.

- ▶ Ein **Unterobjekt (Satellit)** ist ein Objekt, das an ein **Kernobjekt** angelagert ist und mit ihm ein integriertes **komplexes Objekt** bildet
  - Das Unterobjekt hat also keine eigene Identität, sondern teilt seine Identität mit dem Kernobjekt (logische Einheit)
  - Es repräsentiert eine Eigenschaft bzw. Teilzustand des komplexen Objekts
  - Als **Integrationsrelation** `<<integrates>>` wird eine Endo-Assoziation verwendet (spezielle Fälle sind Aggregation, Komposition, Rollenspiel)



Schon in der Analyse werden komplexe Objekte aus einem *Kern* und einer Menge von zugehörigen *Unterobjekten* (Teilobjekte, Satelliten) dargestellt

- ▶ **Satz:** Die Endoassoziation eines komplexen Objekt ist immer *hierarchisch* oder *azyklisch*, bildet also immer eine Hierarchie oder einen gerichteten azyklischen Graphen (dag):
  - **Flache komplexe Objekte** als Kern mit **Unterobjekten**, die in den Kern mit integrates-a integriert sind
  - **Hierarchische komplexe Objekte** in **Baum-Notation** in einem **hierarchischen** komplexen Objekt (Kernobjekt als Wurzel)
  - **Geschichtete komplexe Objekte** in **Dag-Notation** in einem **geschichteten** komplexen Objekt (Kernobjekt als Wurzel eines gerichteten azyklischen Graphen, directed acyclic graph, dag)

# Rätsel

- ▶ Welche Endoassoziation trägt das Entwurfsmuster “Bridge”?

# Arten von komplexen Objekten (Analyse- und Entwurfsobjekten)

- ▶ Komplexe Objekte kommen in allen Phasen vor
  - Ihr **Lebenszyklus** wird durch eine Zustandsmaschine beschrieben (Kap 33)
  - Interpretierer-Funktion: Sie empfängt Befehle wie Create(), Open(), Read(), Write(), Do(), Enter()
  - Steuerungsfunktion: Sie sendet weitere Befehle aus (Steuerungsmaschine)
- ▶ **Analyseobjekte** stammen oft von **Domänenobjekten** her
  - **Simuliertes Objekt**: kommt in der Umgebung des Programms vor (Simulation)
    - Kunde, Bestellung, Huhn, Auto, Radar, ...
  - **Geschäftsobjekt**: abstraktes Material-Objekt der Anwendung
    - Rechnung, Termin, Bestellung, Bestellposition
- ▶ **Entwurfsobjekte** modellieren zusätzlich technische Eigenschaften:
  - **Schnittstellenobjekte** (im Kontextmodell): Widget, Stream, File
  - **Technische Objekte**: Objekte des Systems, von dem der Kunde nichts sieht: Komponente, Treiber, Datenbank, ...
- ▶ **Implementierungsobjekte** (einfache, physikalischen Objekte) sind dagegen einfach, flach und passen direkt auf die Maschine
  - Sie entsprechen Verbunden (records). Sie tragen keine Analyseinformation mehr, sind nackt

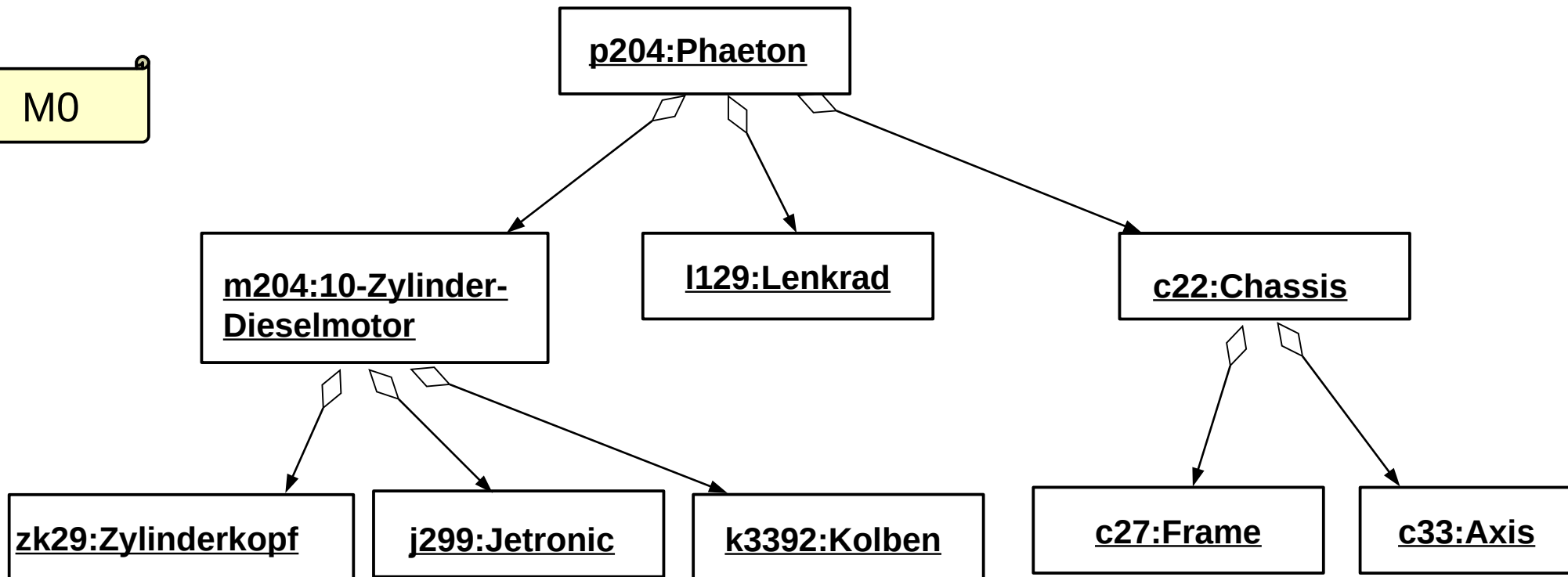
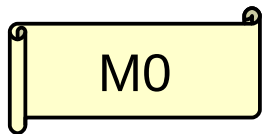
# Komposite Material-Objekte in Anwendungen

- ▶ Viele Daten, die in Anwendungen gehandhabt werden, sind **komposite Objekte** (*Materialien*)
  - Materialien gehören zur Datenhaltungsschicht
  - Materialien sind oft komplex
- ▶ Beispiele:
  - **Informationssysteme** verwalten komplexe Informationen (“EDV”)
    - Buchungen, Daten
  - **Produktionsplanungssysteme** verwalten
    - *Produkte*. Die Teile eines Produkts werden in Stücklisten (eigentlich Stückbäume) verwaltet
    - *Fabriken*. Die Teile und Maschinen einer Fabrik werden modelliert
  - **Geschäftsprozesssoftware** verwaltet *Dokumente von Geschäftsvorgängen* (Bestellungen, Rechnungen, Löhne, Mitarbeiter...), die ebenfalls komposite Objekte darstellen
    - *Geschäftsobjekte (business objects)*, Objekte des Domänenmodells, sind oft komposit

*“Welche Stücklisten kommen mit welchen Teilen in dem Text des Kunden vor?”*

# Beispiel: Komposite Material-Objekte als Stücklisten

- ▶ Eine der meistgebrauchten Datenstrukturen sind hierarchische *Stücklisten (part lists)*
  - *Bestellung, Formular, Rechnung* sind alles Stücklisten
- ▶ Produktionsplanungssysteme (PPS) verwalten Produkte als Materialien
  - Die Teile eines Produkts, Artikels, Materials werden in *Stücklisten (eigentlich Stückbäume)* verwaltet
- ▶ Stückliste eines Phaeton (alle Teile sind lebenslang nummeriert, um verfolgbar zu sein)





Auf dem Weg vom Analysemodell über das Entwurfsmodell zum Implementierungsmodell und Code werden wir folgende Arten von Verfeinerungsmethoden kennenlernen:

- 1) **Punktweise Verfeinerung von Lebenszyklen von komplexen Objekten:** Abbilden von Lebenszyklen auf niederere Schichten
  - Interpreterverfeinerung (Automatverfeinerung, Verfeinerung von abstrakten Maschinen) (Kapitel OOD.34)
  - Tool-Verfeinerung
  - Material-Verfeinerung
- 2) **Querschneidende Objektorreicherung (object fattening) von komplexen Objekten:** Verfeinerung von Objekten aus dem Domänenmodell durch Integration von Unterobjekten
  - Jedes Unterobjekt stellt eine neue Eigenschaft des Domänenobjektes dar
  - Szenarioanalyse (Kapitel OOA.35)

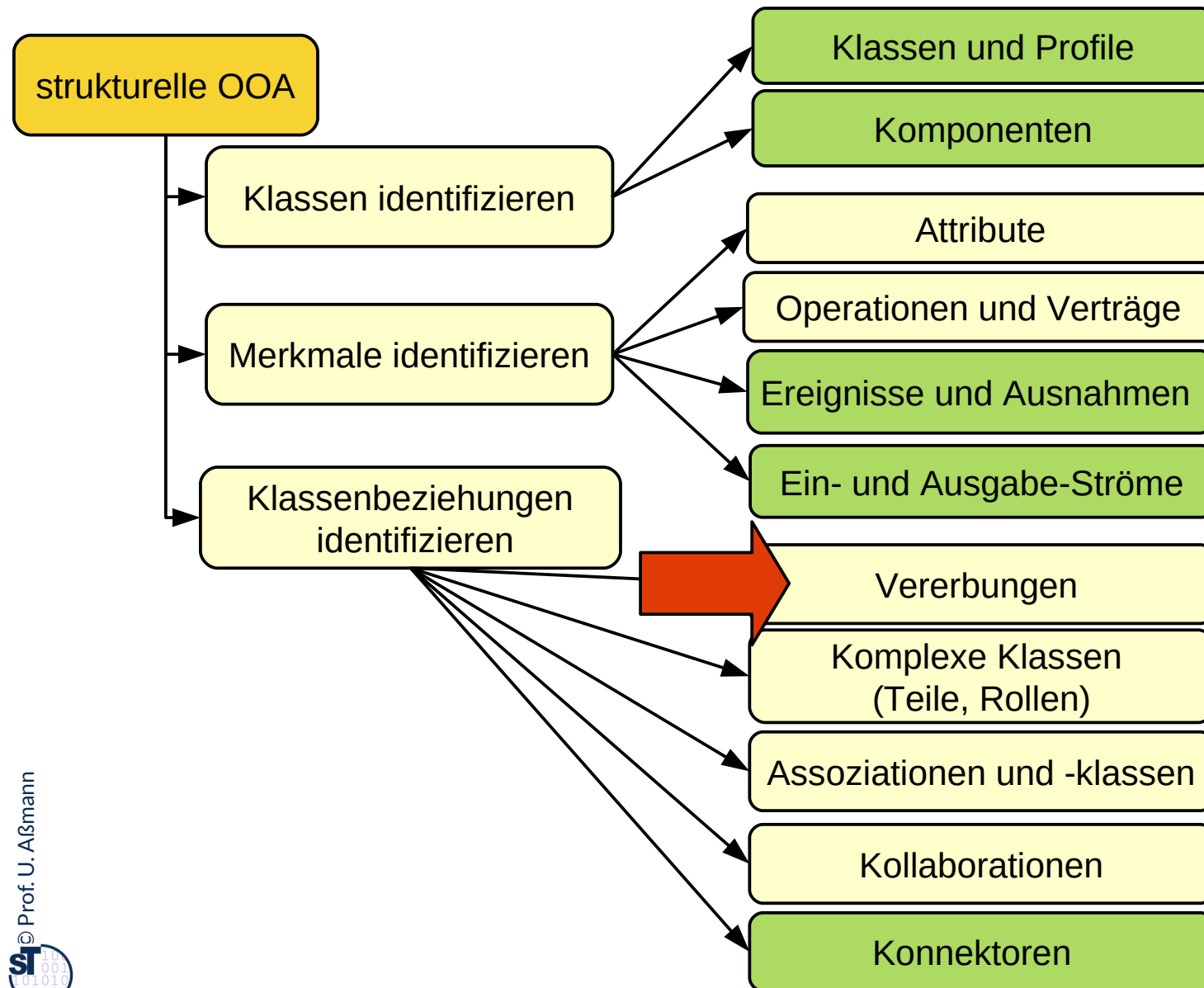
## 31.5 Mehrfachvererbung zwischen Klassen

- ▶ In Analyse und Grobentwurf spielt mehrfache Vererbung eine große Rolle



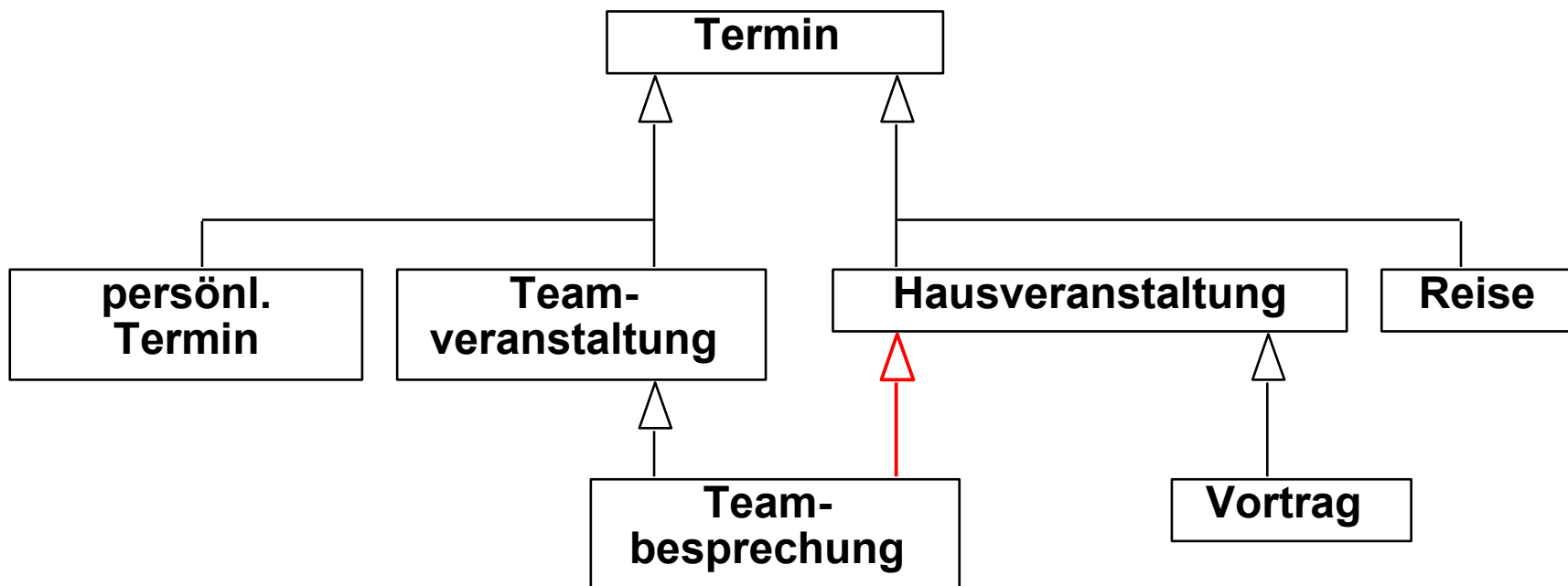
# Q6: Schritte der strukturellen, metamodellgetriebenen Analyse

- ▶ gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



# Mehrfachvererbung in aUML

- ▶ "Eine Teambesprechung ist eine Teamveranstaltung, die ein Termin ist."
- ▶ "Eine Teambesprechung ist eine Hausveranstaltung. Die Hausveranstaltung ist eine Art von Termin."
- ▶ In UML ist es prinzipiell möglich, daß eine konkrete Klasse von mehreren Klassen erbt:

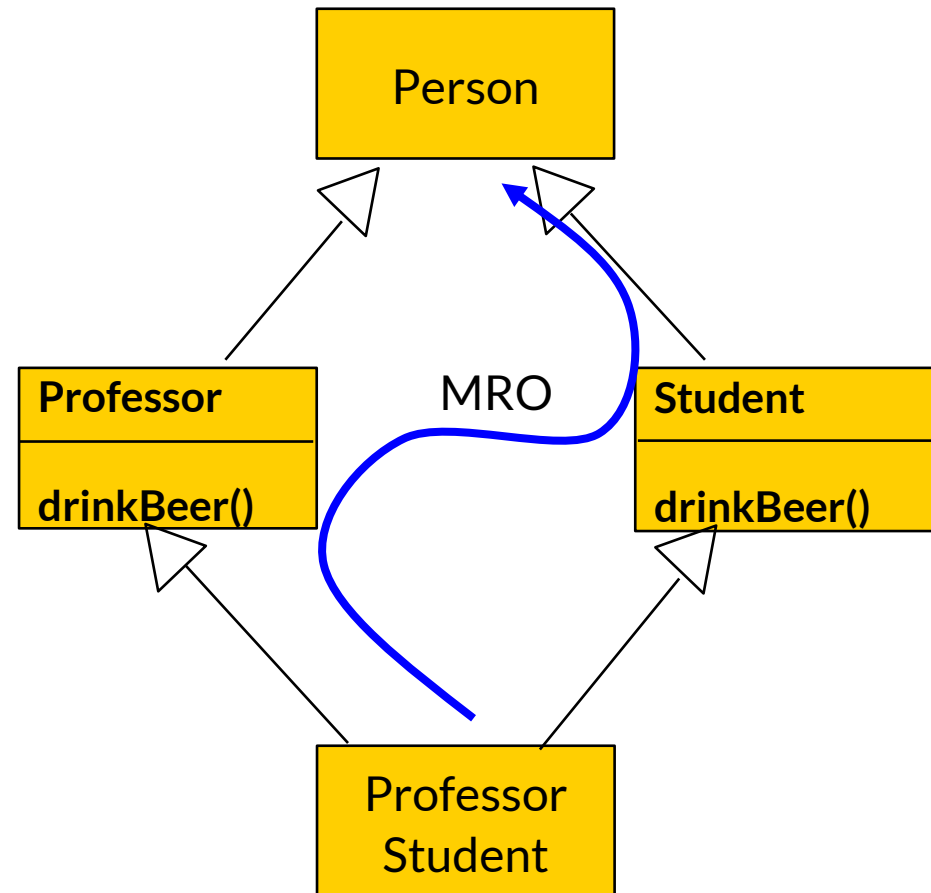


# Mehrfachvererbung (Multiple Inheritance)

90

Softwaretechnologie (ST)

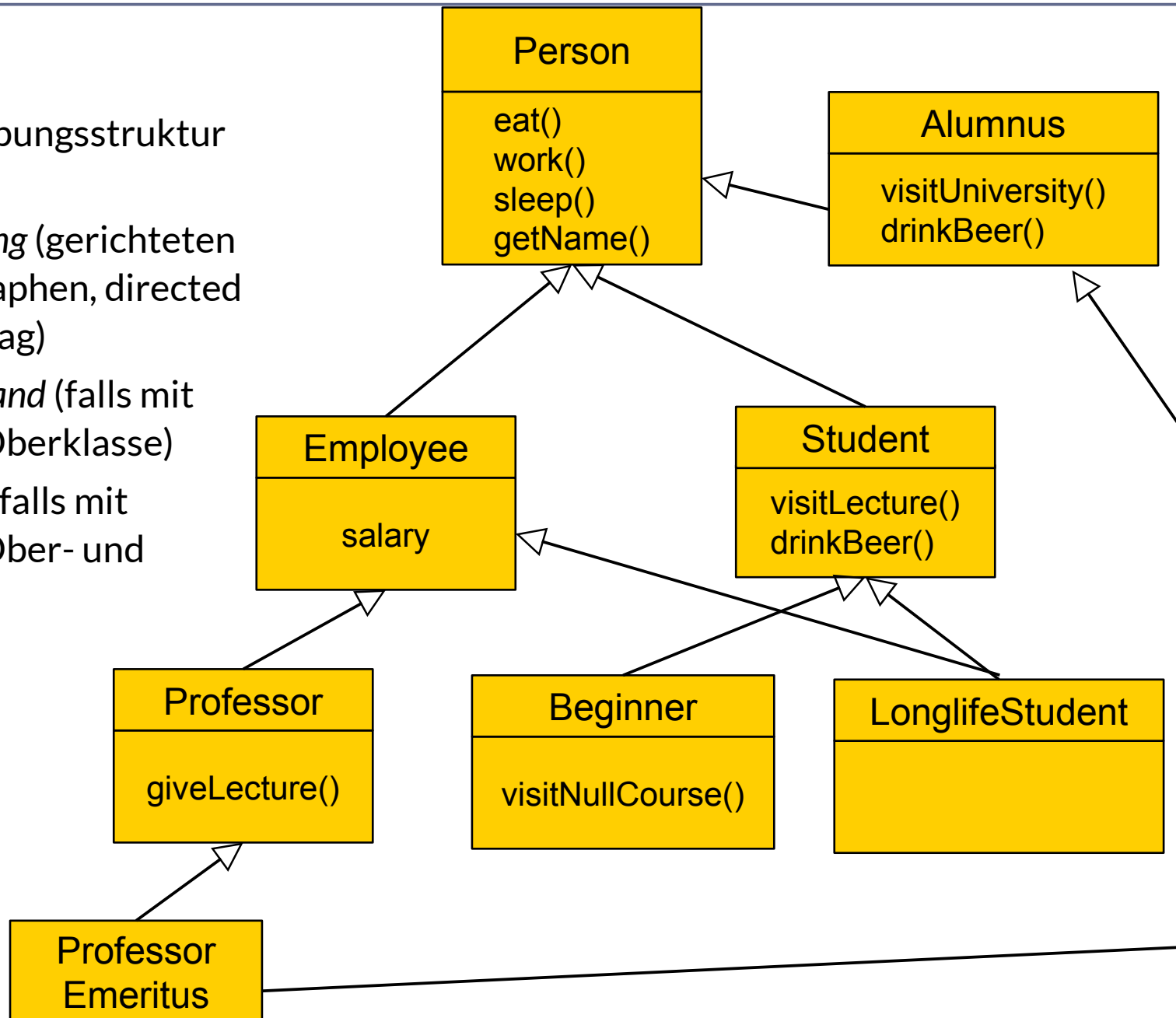
- ▶ **Mehrfachvererbung**
  - Eine Klasse kann mehrere Oberklassen besitzen
  - Dadurch wird die Vererbungsrelation ein gerichteter azyklischer Graph (dag)
- ▶ Eine Klasse kann ein Merkmal mehrmals erben
  - Daher muss die Merkmalsuche einer Strategie gehorchen, der Merkmalsauflösungsordnung (method-resolution-order, MRO), einer Navigationsstrategie, die aufwärts nach Merkmalen sucht
  - Oft links-nach-rechts-aufwärts-Suche



Von wo wird drinkBeer() geerbt?

# Ein grosser Mehrfachvererbungsverband

- ▶ Eine Mehrfachvererbungsstruktur bildet
  - eine *Halbordnung* (gerichteten azyklischen Graphen, directed acyclic graph, dag)
  - einen *Halbverband* (falls mit gemeinsamer Oberklasse)
  - Einen *Verband* (falls mit gemeinsamer Ober- und Unterklasse)

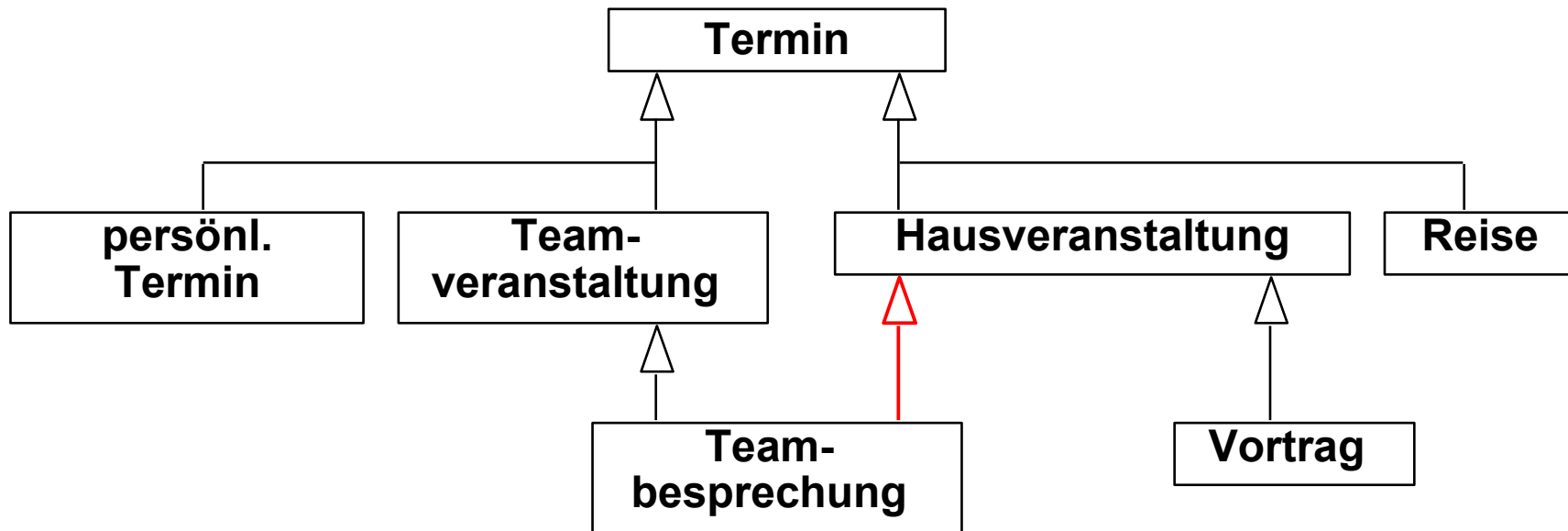


## 31.5.2 Realisierung von Mehrfachvererbung



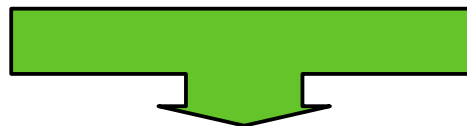
# Rätsel: Wie realisiert man Code-Mehrfachvererbung in Java?

- ▶ In UML ist es prinzipiell möglich, daß eine konkrete Klasse von mehreren Klassen erbt:



- ▶ Java unterstützt Mehrfachvererbung von konkreten Klassen **nicht**, nur von Schnittstellen:

```
class Teambesprechung extends Teamveranstaltung, Hausveranstaltung
```

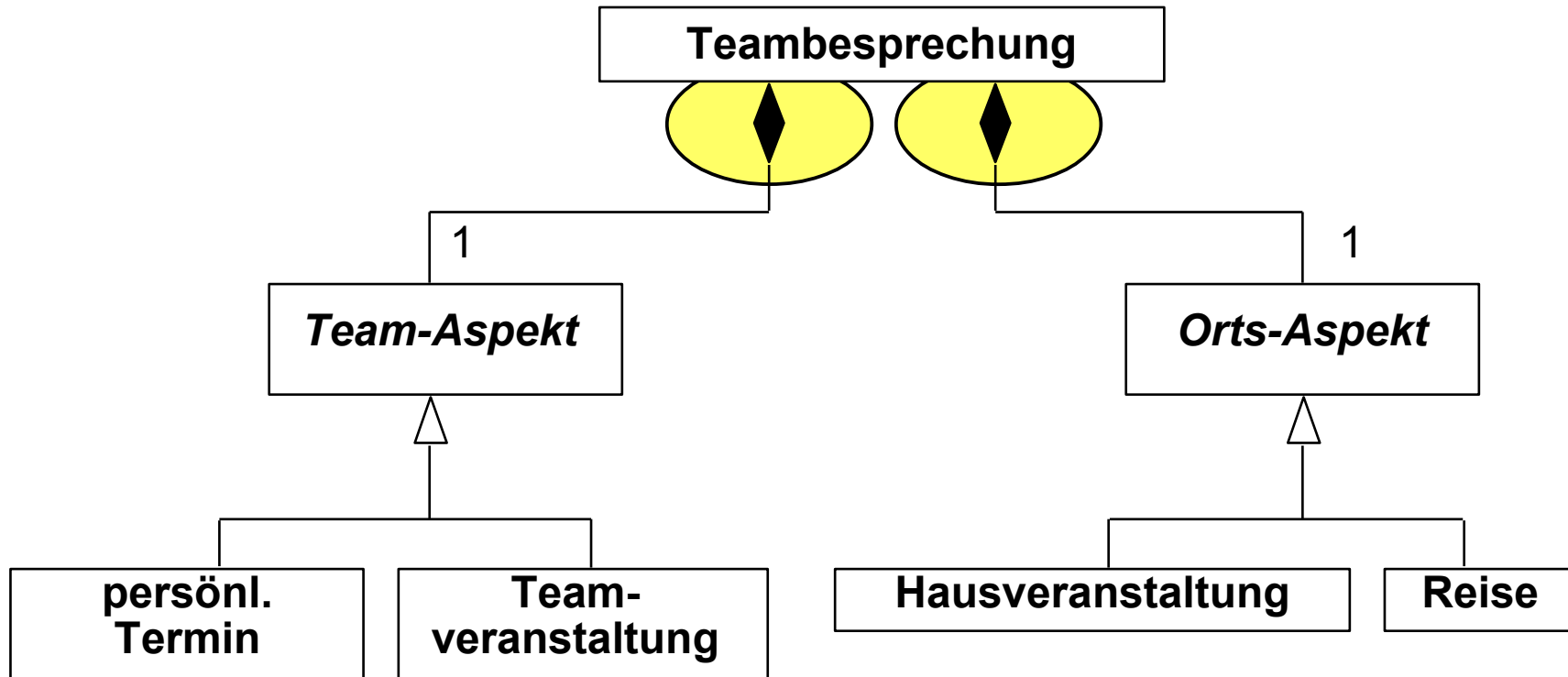


```
class Teambesprechung extends Teamveranstaltung
    implements Hausveranstaltung
```



# Lösung: Realisierung von Mehrfachvererbung durch Komposition in kompositen Objekten

- ▶ Mehrfachvererbung wird häufig durch komposite Objekte simuliert



```
class Teambesprechung {
    private Teamaspekt ta;
    private Ortsaspekt oa;
    ...
}
```

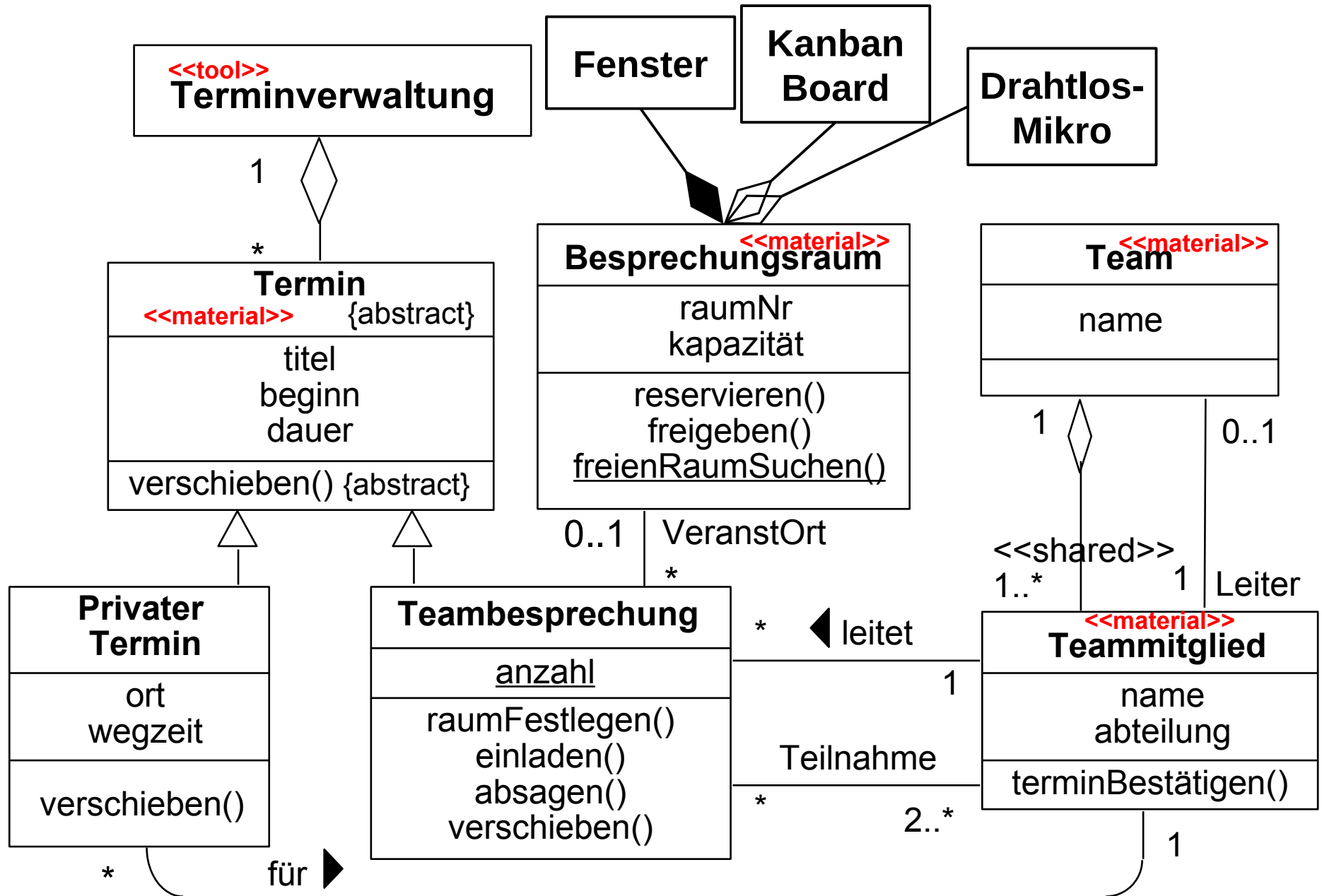
```
abstract class Teamaspekt {
}

abstract class Teamveranstaltung
    extends Teamaspekt {
    ...
}
```

# Verschiedene Ähnlichkeitsrelationen in Analysemodellen (Similarity Relationships)

- ▶ **is-a (ist-ein)**: zeigt in Analysetexten und Analysemodellen *allgemeine Ähnlichkeit* an:
  - In Mengenhierarchien ist die Untermengenrelation gemeint (subset)
  - In Begriffshierarchien Unterkonzept-Relation (subconcept)
  - is-a ist azyklische Relation, bei einfacher Vererbung baumförmig
- ▶ **is-structured-like (teilt-Struktur-mit)**: zeigt ähnliche Struktur an (strukturelle Ähnlichkeit oder Gleichheit)
- ▶ **behaves-like (verhält-sich-wie)**: zeigt Verhaltensähnlichkeit an
  - **conformant (always-behaves-like)**: *ständige Verhaltensgleichheit* (Konformanz) bedeutet Ersetzbarkeit (substitutability)
  - **sometimes-behaves-like**: *gelegentlich verhaltensgleich*
  - **restrictedly-behaves-like**: meist konformant, aber nicht in speziellen Situationen (extravagance, restriction inheritance)
- ▶ Achtung: *is-a, is-structured-like, behaves-like, conformant* werden alle *Vererbung* genannt
  - Dagegen: *instance-of*: A ist aus einer Schablone S gemacht worden

# Beispiel: Analyse-Klassendiagramm



# Ausblick: Verfeinerung von Analyse- zum Entwurfsmodell

## Analyse-Modell (Fragmente)

Notation: aUML

Objekte: Fachgegenstände

Klassen: Fachbegriffe, parallele Prozesse,  
komplexe Objekte mit Endorelationen

Attribute ohne Typen und Sichtbarkeiten

Operationen: ohne Typen,  
Parameter und Rückgabewerte  
Assoziationen: partiell, bidirektional  
i. Allg. ohne Datentypen  
Aggregationen, Kompositionen  
Leserichtung, partielle Multiplizitäten

Vererbung: Begriffsstruktur

Annahme perfekter Technologie

Funktionale Essenz

Völlig projektspezifisch

Grobe Strukturskizze

## Entwurfs-Modell (Mehr Struktur & mehr Details)

Notation: dUML

Objekte: Softwareeinheiten

Klassen: mit Abstrakt, Interface, Stereotyp; Einsatz von  
Entwurfsmustern für komplexe Objekte

Attribute: Sichtbarkeiten, Ableitung, Klassenattribute,  
Initialisierung, weitere spezielle Eigenschaften

Operationen: voll typisiert, mit  
Parameter, Rückgabewert, Klassenoperation

Unidirektionale Assoziationen mit  
voller Multiplizität, Navigation, qualifizierte A.

Vererbung: Programmableitung

Annahme perfekter Technologie

Erfüllung konkreter Rahmenbedingungen

Gesamtstruktur des Systems

Ähnlichkeiten zwischen verwandten  
Projekten (zwecks Wiederverwendung)

Genaue Strukturdefinition

# Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

	Schichten	Analyse	Entwurf	Feinentwurf	Implementierung
Benutzungs- schnittstelle (Boundary)	GUI		Verfeinerung		
	Controller				
Anwendungslo- gik (Control)	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung System-schnittstellen	stabil	Umsetzen auf jUML	stabil
	Top-Level-Architektur	Verfeinerung des Kontextmodells	stabil	Umsetzen auf jUML	stabil
	Architektur		Ausarbeitung Architektur (PSM)	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Sequentialisierung	Details ausfüllen, Methoden ausprogrammieren
Daten-haltung (Database)	Tools		Ausarbeitung Tools	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Sequentialisierung	Details ausfüllen, Methoden ausprogrammieren
	Material	Aufstellung aus Domänenmodell		Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML;	Details ausfüllen, Methoden ausprogrammieren

# Was haben wir gelernt?

- ▶ Strukturelle Analyse spürt die Struktur von objektorientierten Anwendungen auf
- ▶ Strukturgetriebene Analyse verwendet das Metamodell, um die Elemente von Modellen aufzuspüren
  - Strukturelle Analyse mittels CRC und UML-Klassendiagramme sind beides Beispiele für strukturgetriebene Analyse
- ▶ Das UML-Metamodell gibt Klassen, Merkmale, und Beziehungen als Strukturelemente vor
  - Diese werden in Interviews mit dem Kunden als Konzepte abgefragt
  - Analysemodelle entstehen durch Analyse von Interviews und können “vorgelesen” werden, um das Interview zu rekonstruieren
- ▶ Analysemodelle müssen beim Übergang ins Entwurfsmodell abgeflacht werden
  - Mehrfachvererbung kann durch Komposition ausgedrückt werden
  - Assoziationen werden durch Graphen und Collections abgebildet
  - Endo-Relationen werden durch Entwurfsmuster wie Bridge, Composite abgebildet

# The End

- ▶ Erklären Sie den Begriff des Metamodells und den Unterschied von Strukturgleichheit und Verhaltensgleichheit zweier Objekte.
- ▶ Wie schafft man es, aus einer UserStory möglichst viel herauszuholen?
- ▶ Wie leitet man aus einem Metamodell Fragen zur Analyse ab?
- ▶ Wie führt man den Prozess der strukturellen fragegetriebenen Analyse durch?
- ▶ Was ist ein komplexes Objekt und welche Arten von Endorelationen gibt es?
- ▶ Wie unterscheidet sich die <<part-of>> Relation von <<plays-a>>?
- ▶ Warum kann ein komplexes Objekte eine seiner Rollen verlieren und wieder annehmen?
- ▶ Wie kann man in C++, das Mehrfachvererbung besitzt, eine Klasse mit 10 Oberklassen realisieren?
- ▶ Einige Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

# Rätsel-Lösungen

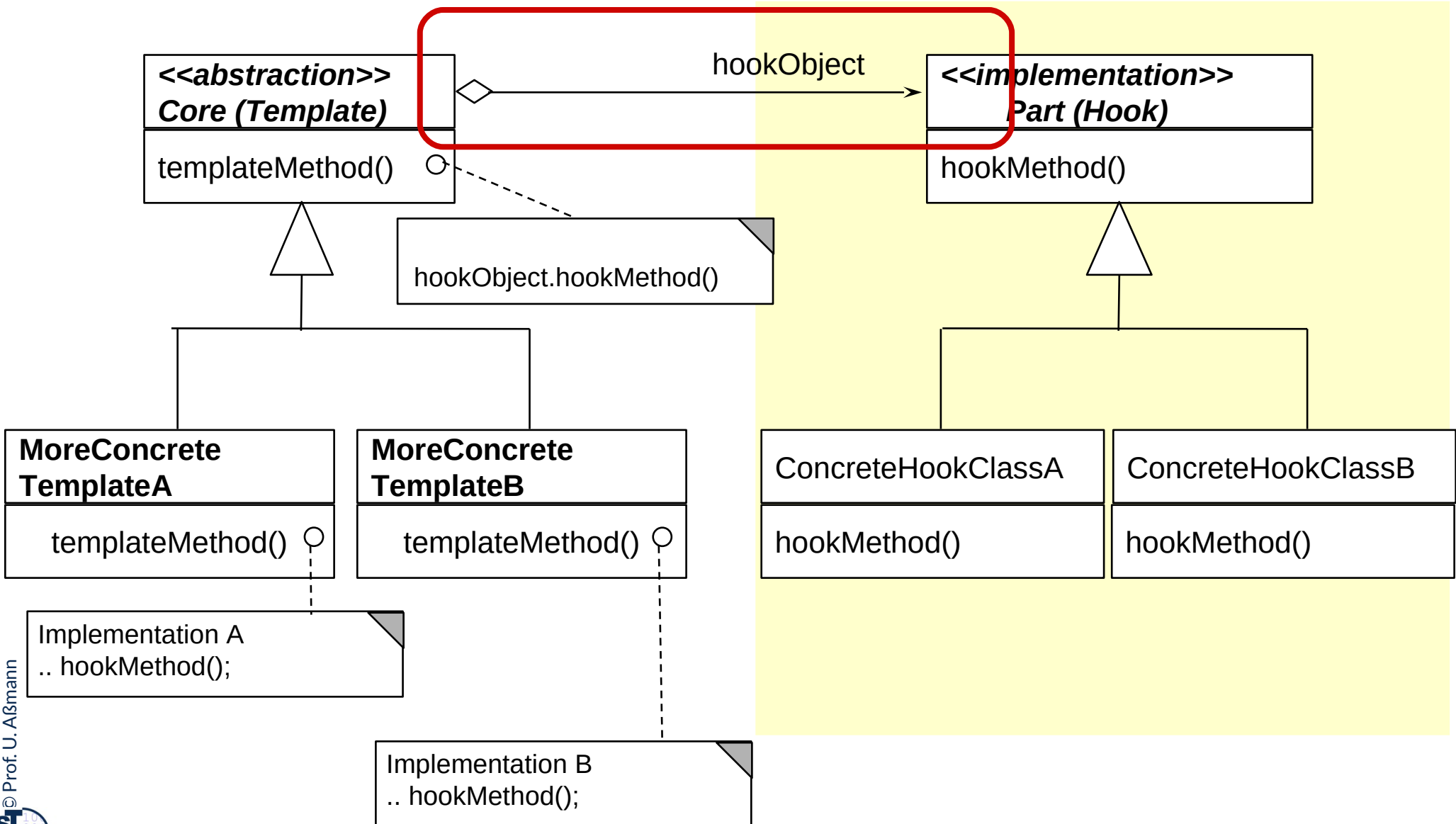


DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur



# Endoassoziation in Bridge und Dimensional Class Hierarchies

- ▶ Bridge implementiert komplexe Objekte mit "hookObject" als Endoassoziation



## A.31.1 Analyse von Merkmalen

Einige Details zu bereits in Teil I eingeführten Konzepten



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Botschaften, Operation, Methode

- ▶ Eine **Botschaft (Message, Nachricht)** ist eine Nachricht eines Senders an ein Empfänger-Objekt, eine Operation auszuführen
- ▶ Eine **Rückmeldung** ist eine Botschaft, die ein Sender einer Botschaft als Antwort erhält
- ▶ Eine **Operation** koppelt eine Botschaft mit einer Reaktion und einer Rückmeldung: "a service that can be requested from an object to effect behaviour" (UML-Standard)
- ▶ Eine **Methode (method)** "is the implementation of an operation" (das "Wie" einer Operation)
  - "In den Methoden wird all das programmiert, was geschehen soll, wenn das Objekt die betreffende Botschaft erhält." (Middendorf/Singer)
  - *synchrone Operation / Methode*: der Sender wartet auf die Beendigung des Service
  - *asynchrone Operation*: ein Service mit Verhalten aber ohne Rückgabe, d.h. der Sender braucht nicht zu warten
- ▶ Ein **Push** ist eine Botschaft mit einem Datum an ein Objekt zur Ablage
- ▶ Ein **Pull** ist eine Botschaft an ein Objekt zur Ablage, das als Rückmeldung ein Datum mitführt

# Klassenattribute (Statische Attribute)

- ▶ Ein **Klassenattribut** beschreibt ein Datenelement, das genau einen Wert für die gesamte Klasse annehmen kann.
  - Es ist also ein Attribut des Klassenprototypen
- ▶ **Notation:** Unterstreichung
- ▶ **Implementierung:**
  - Implementierungsmuster Singleton
  - Klassenattribute und -operationen: Schlüsselwort **static**

<b>Teambesprechung</b>
titel: String beginn: Date dauer: Int <u>anzahl: Int</u>

# Klassenoperation (Statische Operation)

- ▶ **Definition** Eine *Klassenoperation* *A* einer Klasse *K* ist die Beschreibung einer Aufgabe, die nur unter Kenntnis der aktuellen Gesamtheit der Instanzen der Klasse ausgeführt werden kann.  
Gewöhnliche Operationen heißen auch *Instanzoperationen*.
- ▶ **UML Notation:** Unterstreichung analog zu Klassenattributen.
- ▶ **Java:** Die Methode `main()` ist statisch, und kann vom Betriebssystem aus aufgerufen werden

Besprechungsraum
raumNr kapazität
reservieren() freigeben() <u>freienRaumSuchen()</u>

```
class Steuererklaerung {  
  
    public static main (String[] args) {  
        Steuerzahler hans =  
            new Steuerzahler();  
        ...  
    }  
}
```

# Operationen in der Analyse

- ▶ **Def.:** Eine *Operation* einer Klasse *K* ist die Beschreibung einer Aufgabe, die jede Instanz der Klasse *K* ausführen kann.

Teambesprechung	Teambesprechung
titel beginn dauer	titel beginn dauer
raumFestlegen einladen absagen	raumFestlegen() einladen() absagen()

- ▶ "Leere Klammern":
  - In vielen Büchern (und den Unterlagen zur Vorlesung) zur Unterscheidung von Attributnamen: raumFestlegen(), einladen(), absagen() etc.
  - Auf Analyseebene gleichwertig zu Version ohne Klammern

# Parameter und Datentypen für Operationen

- ▶ Detaillierungsgrad in der Analysephase gering
  - meist Operationsname ausreichend
  - Signatur kann angegeben werden
  - Entwurfsphase und Implementierungsmodell: vollständige Angaben der Typen ist nötig, um Fehler in der Programmierung früher zu erkennen (frühe oder statische Typisierung)

- ▶ **Notation:**

**Operation (Art Parameter:ParamTyp=DefWert, ...): ResTyp**

- *Art* (des Parameters): **in**, **out**, oder **inout** (weglassen heißt **in**)
- *DefWert* legt einen Default-Parameterwert fest, der bei Weglassen des Parameters im Aufruf gilt.

- ▶ **Beispiel** (Klasse Teambesprechung):

raumFestlegen (in wunschRaum: Besprechungsraum): Boolean

- ▶ **Definition** Die *Spezifikation* einer Operation legt das Verhalten der Operation fest, ohne einen Algorithmus festzuschreiben.

Es wird das "**Was**" beschrieben und noch nicht das "**Wie**".

- ▶ Häufigste Formen von Spezifikationen:
  - Signaturen (Typen der Parameter und Rückgabewerte)
  - Text in natürlicher Sprache (oft mit speziellen Konventionen)
    - Oft in Programmcode eingebettet (Kommentare)
    - Werkzeugunterstützung zur Dokumentationsgenerierung, z.B. "javadoc"
  - Vor- und Nachbedingungen (Verträge, contracts)
  - Tabellen, spezielle Notationen
  - "Pseudocode" (Programmiersprachenartiger Text)
  - Zustandsmaschinen, Aktivitätendiagramme



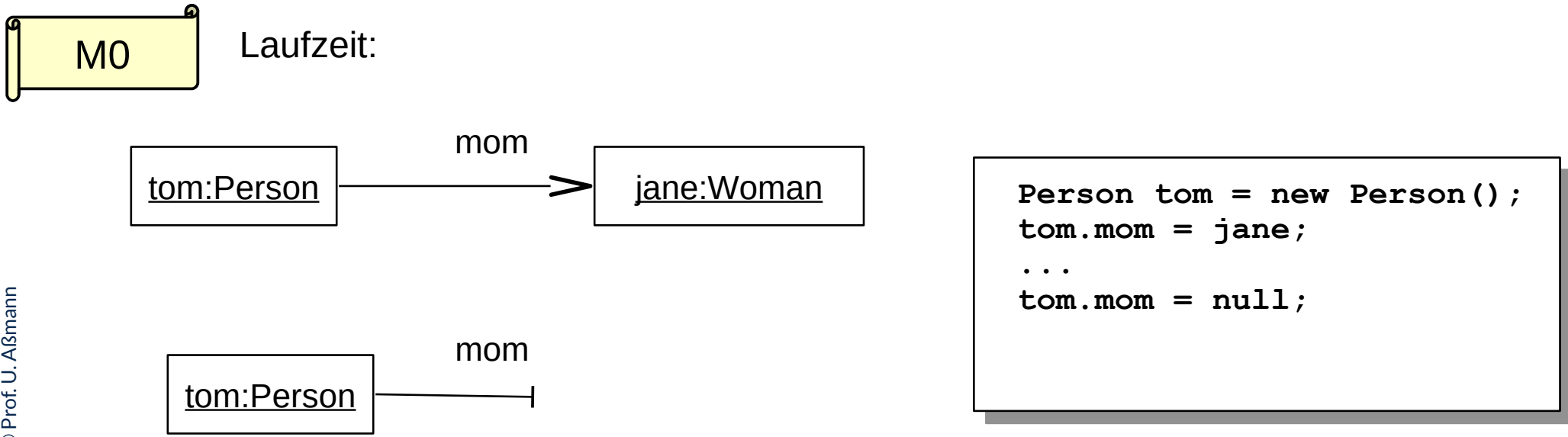
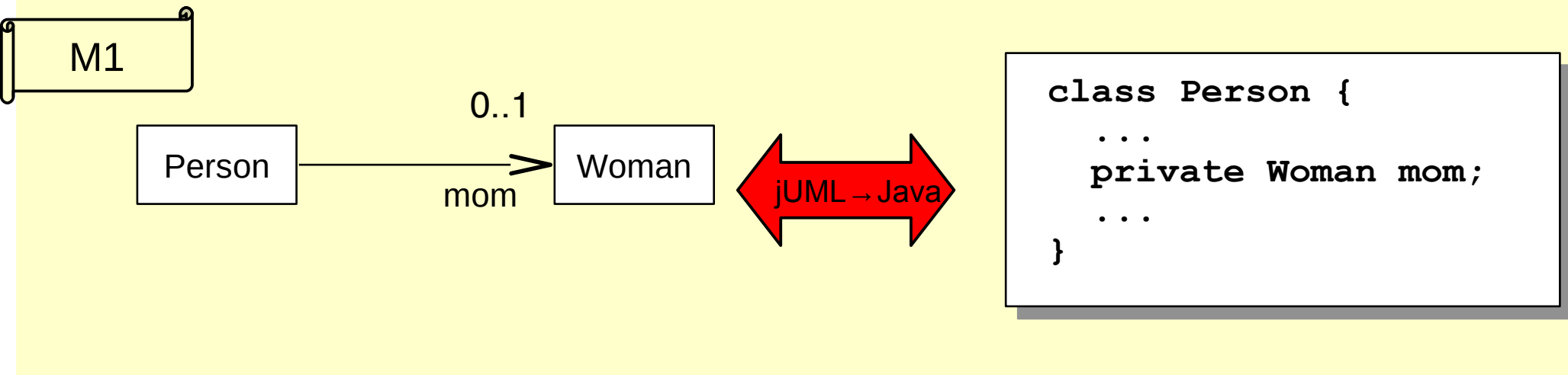
## A.31.3 Analyse von Assoziationen

Einige Details zu bereits in Teil I eingeführten Konzepten



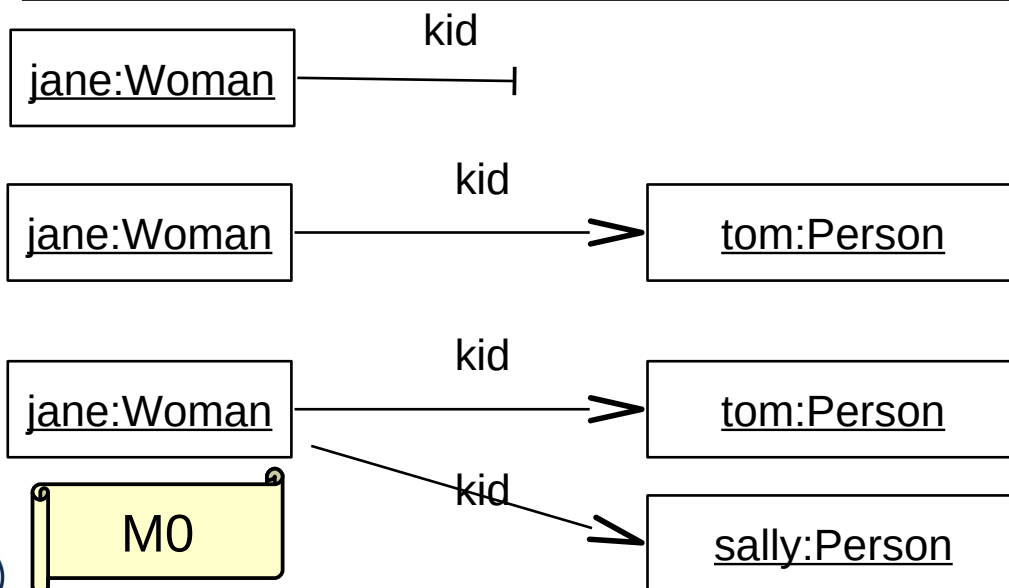
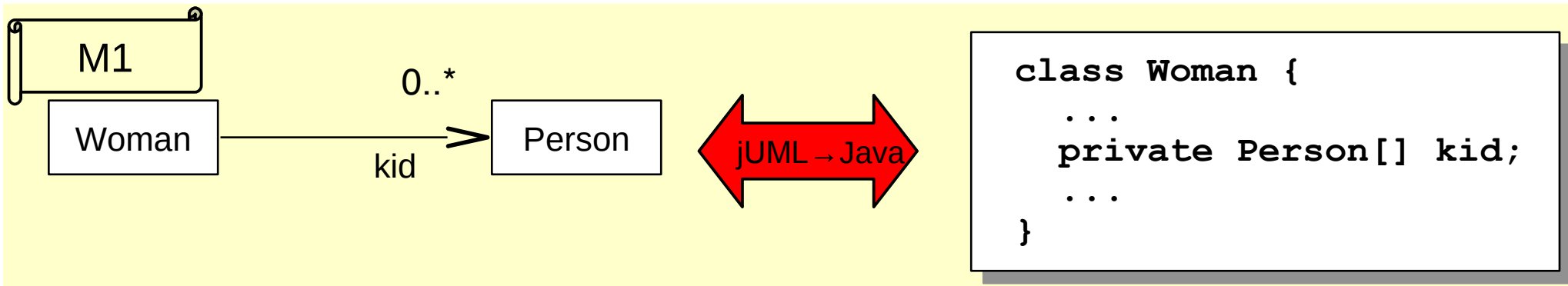
# Einseitige einstellige Assoziationen in Java und jUML (Wdh.)

- Ein Kind kann höchstens eine Mutter haben



# Einseitige mehrstellige Assoziationen (Wdh.)

- ▶ Eine Mutter kann aber viele Kinder haben
  - Annahme: Die Obergrenze der Anzahl der Child-Objekte spätestens bei erstmaliger Eintragung von Assoziationsinstanzen bekannt und relativ klein. (Allgemeinere Realisierungen siehe später.)

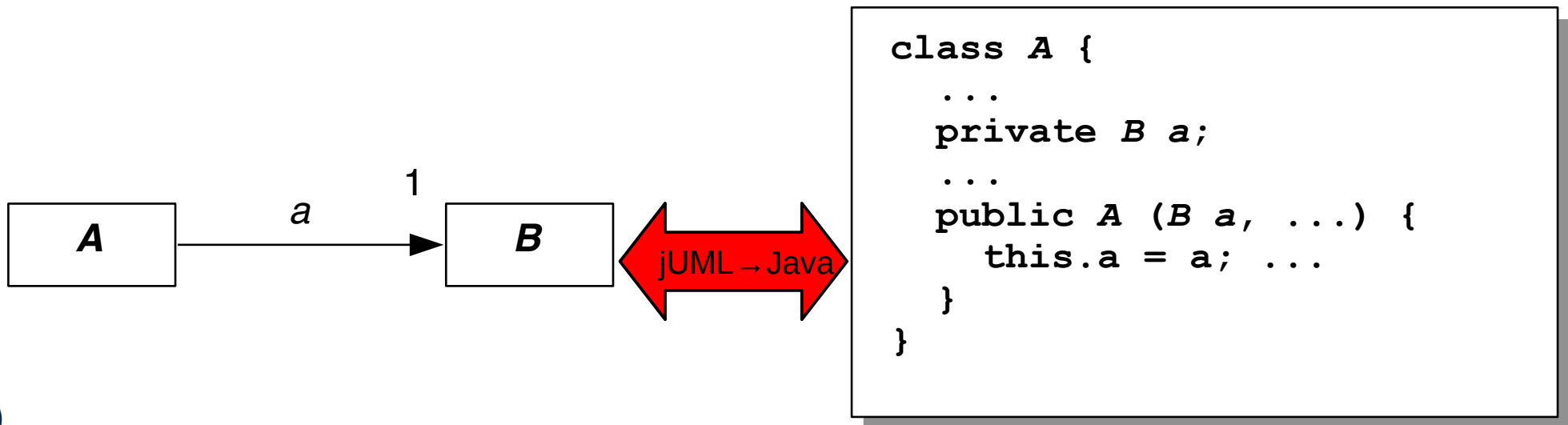
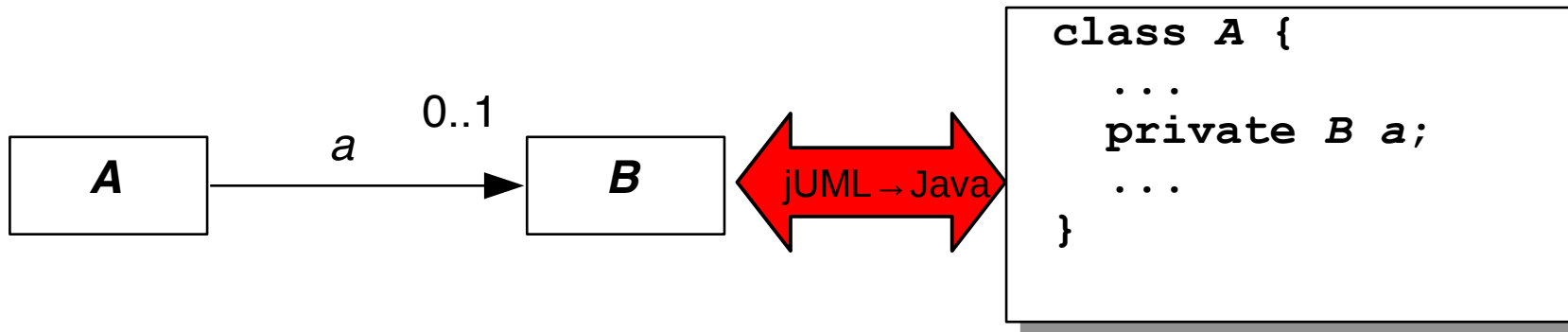


Laufzeit:

```
Woman jane = new Woman();
jane.kid[0] = tom;
...
jane.kid[1] = sally;
```

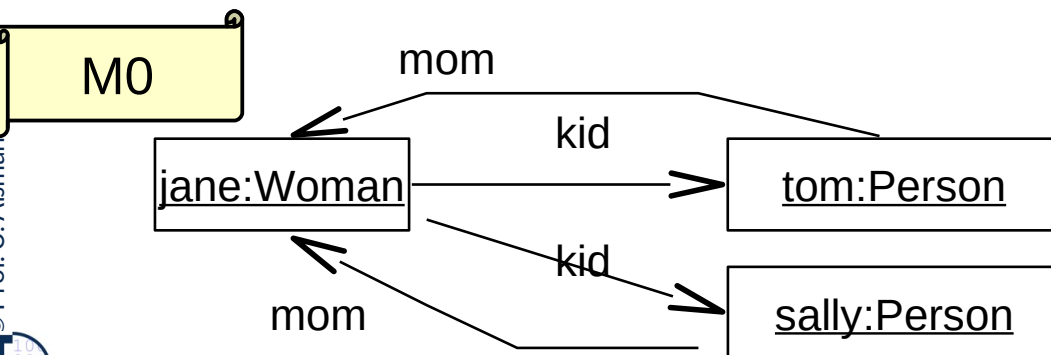
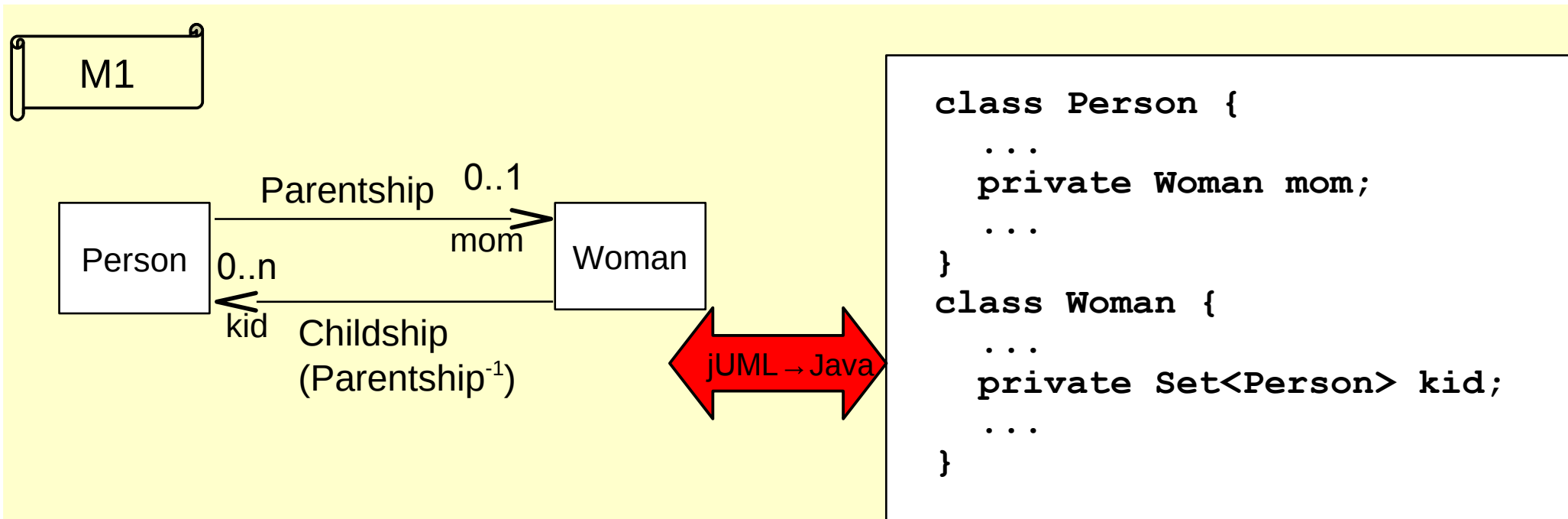
# Optionale und notwendige Assoziationen

- ▶ Untere und obere Schranken von unidirektionalen Assoziationen können durch die Einführung von Argumenten in Konstruktoren eingehalten werden
  - Analog z.B. für Multiplizitäten 0..\* und 1..\*



# Zu 31.3.2. Realisierung von bidirektionalen Assoziationen durch unidirektionale

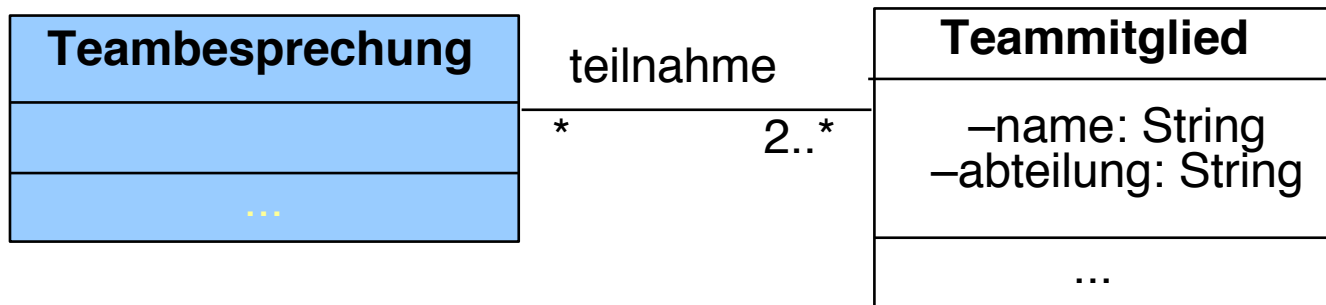
- Realisierung von jUML in Java durch Arrays oder Collections (s. Kap. 21-collections)



```
Woman jane = new Woman();
jane.kid.add(tom);
tom.mom = jane;
...
jane.kid.add(sally);
```

# Realisierung von bidirektionalen Assoziationen durch unidirektionale: Beispiel UML/Java

- ▶ Achtung: fixe Multiplizitäten müssen in Java durch Programmierung kontrolliert werden, z.B. in Konstruktoren (s. Kap. 21-collections)

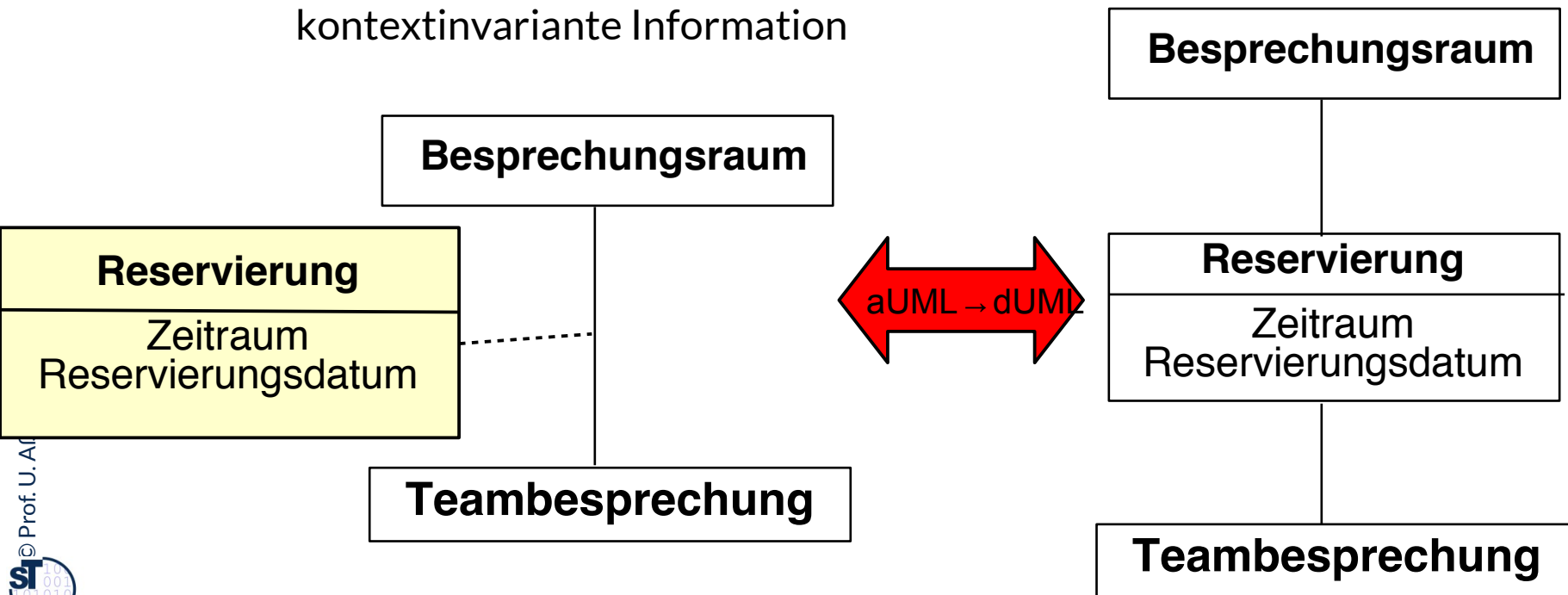


```
class Teambesprechung {
    private Teammitglied[] teilnahme;
    ...
    public Teambesprechung (
        Teammitglied[] teilnehmer) {
        this.teilnahme = teilnehmer;
    }
}
```

```
class Teammitglied {
    private String name;
    private String abteilung;
    private Teambesprechung[] teilnahme;
    public Teammitglied (
        Teambesprechung[] teilnahme) {
        if (teilnahme.size() < 2) error();
        this.teilnahme = teilnahme;
    }...
}
```

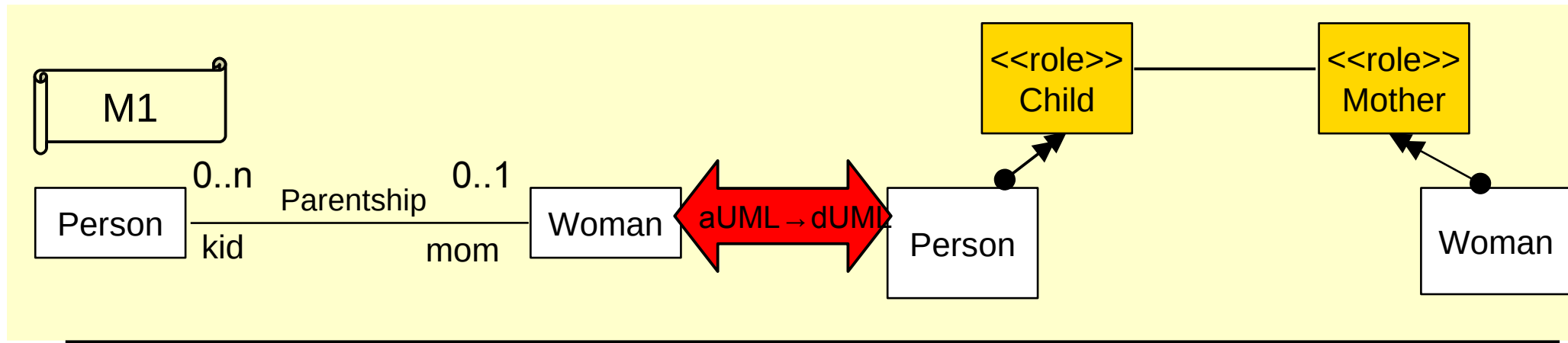
# 31.3.2.4 Realisierung von Assoziationsklassen mit „Zwischenklassen“

- ▶ Assoziationsklassen können im Implementierungsmodell in normale relationale „Zwischen“-Klassen abgeflacht werden:
  - Attribut "Reservierungsdatum" für eine Raumreservierung wird für Assoziation benötigt (z.B. um Reservierungskonflikte aufzulösen).
  - Die Klasse "Reservierung" wird in die bestehende Assoziation eingefügt und "zerlegt" sie in zwei neue Assoziationen.
  - Die Klasse "Reservierung" trägt die kontextspezifische Information, "Besprechungsraum" und "Teambesprechung" fokussieren sich auf die kontextinvariante Information



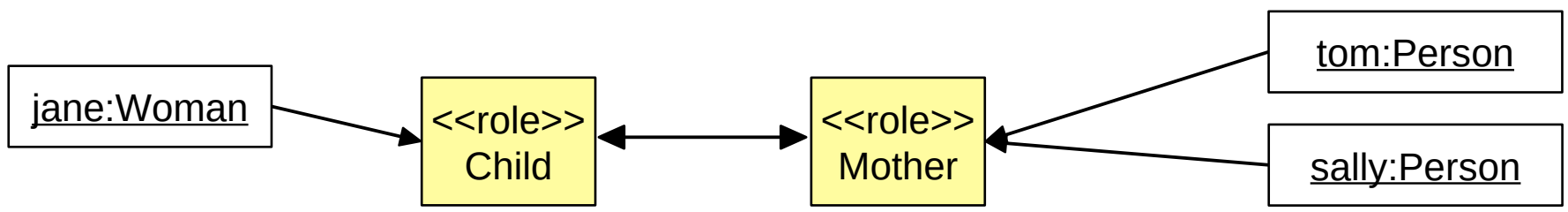
# 31.3.2.5 Realisierung von bidirektionalen Assoziationen durch Rollenklassen

- ▶ Assoziationen können durch Rollenklassen realisiert werden, die das kontextspezifische Verhalten tragen
- ▶ Person, Woman tragen das kontextinvariante Verhalten
- ▶ Grundlage aller kontextadaptiven Software



M0

Laufzeit: Referenzen zwischen Rollenobjekten und den assoziierten Objekten



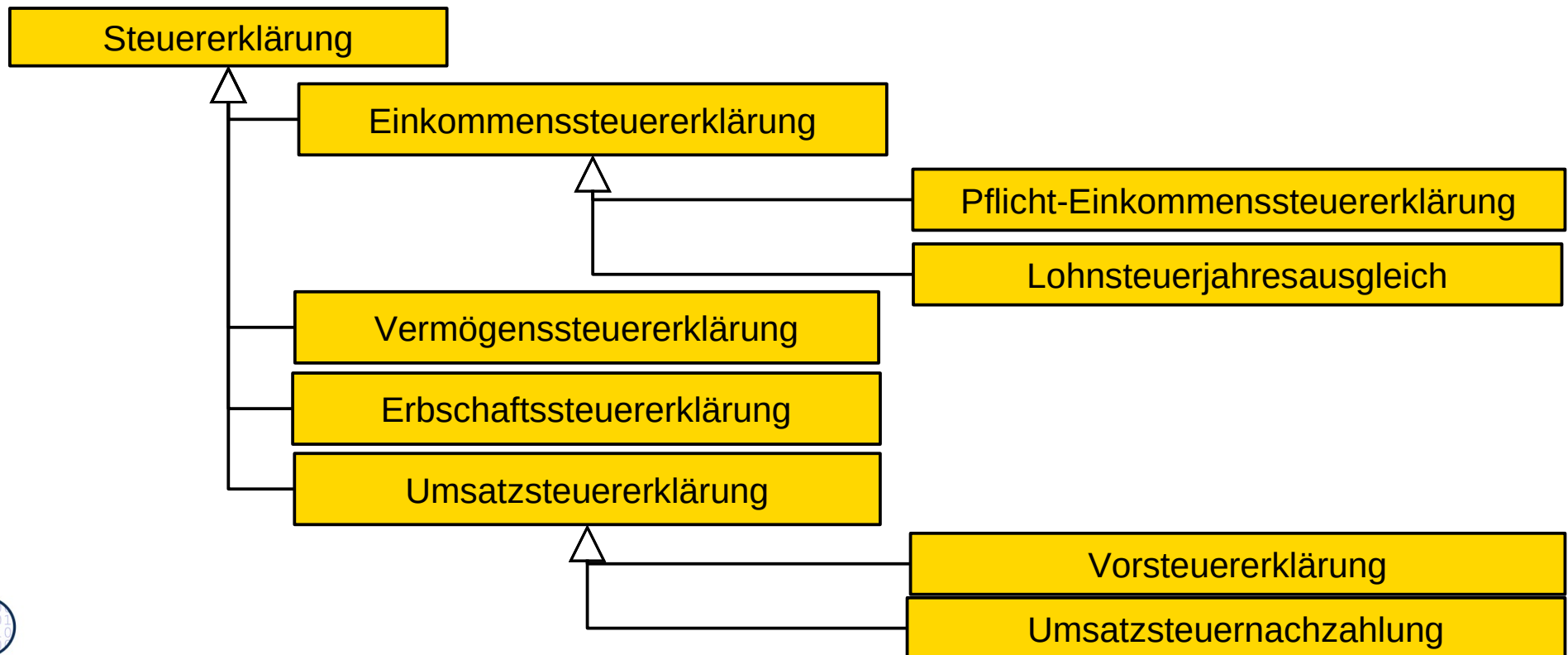


## A.31.4 Weitere Beispiele für Zeilenhierarchien



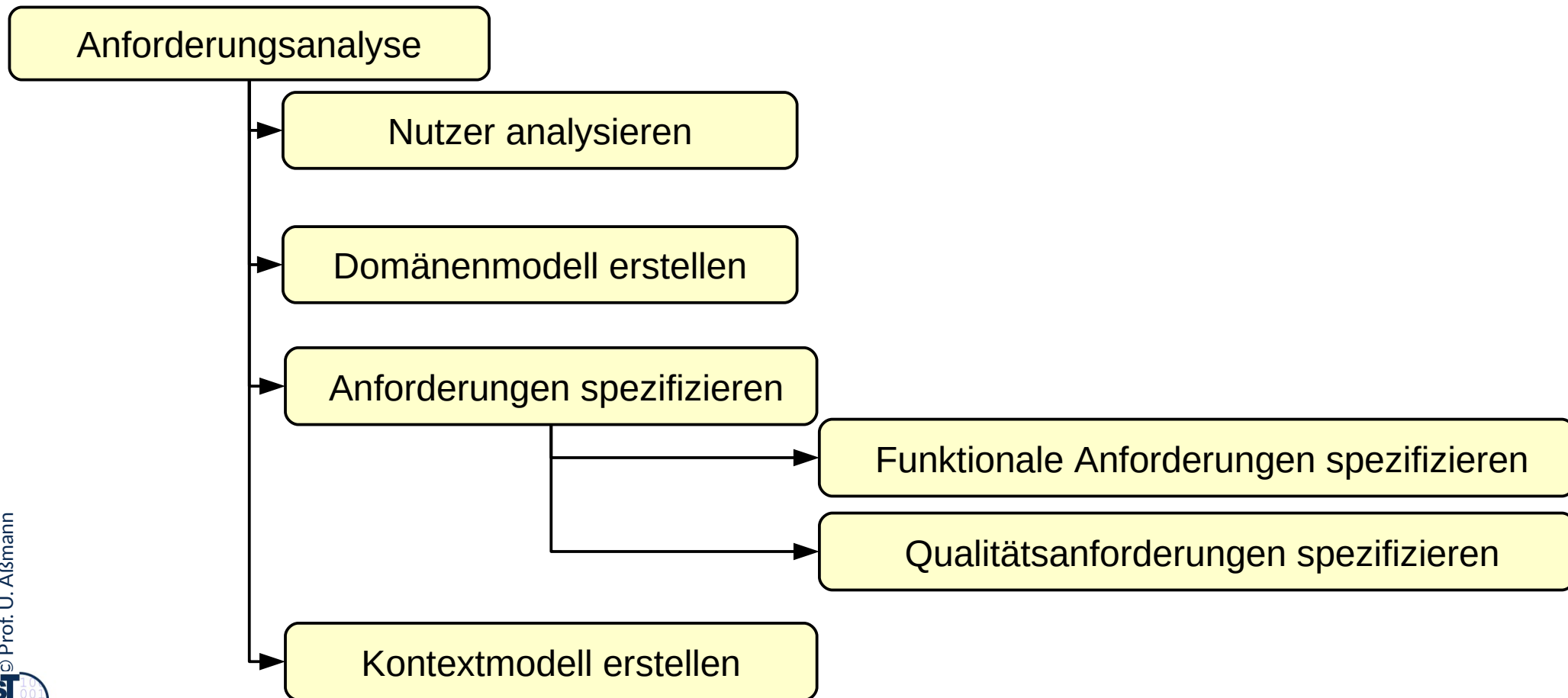
# Zeilenhierarchien

- ▶ Hierarchien kann man als auch als *Zeilenhierarchien (horizontal dekomponierte Textbäume)* anordnen, analog zu einem *tree widget*
  - Teile können leicht zu- und aufgeklappt werden
  - Textuelle Annotationen können einfach in den Zeilen hinzugefügt werden
- ▶ Am einfachsten sieht man das an einer **Begriffshierarchie (Taxonomie)**
  - Ontologieeditoren benutzen dieses Format (z.B. Protege)
  - Zum Lernen verwendet man Begriffshierarchien [Wolf, 2.1]



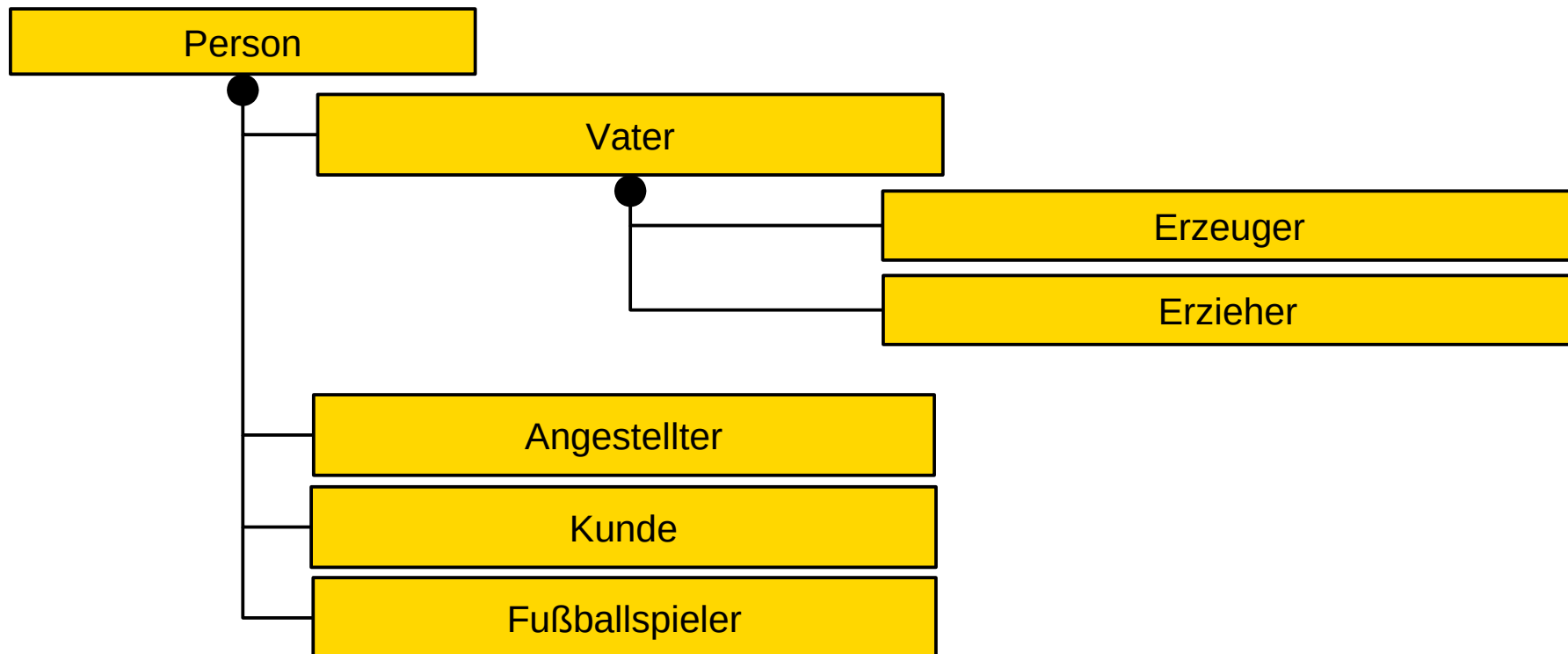
# Bsp. Aktivitätenhierarchie (Aktivitäts- oder Funktionsbaum)

- ▶ Aktivitäten werden in UML durch leicht abgerundete Rechtecke dargestellt
- ▶ Strukturierte Aktivitäten können in Zeilenhierarchien dargestellt werden



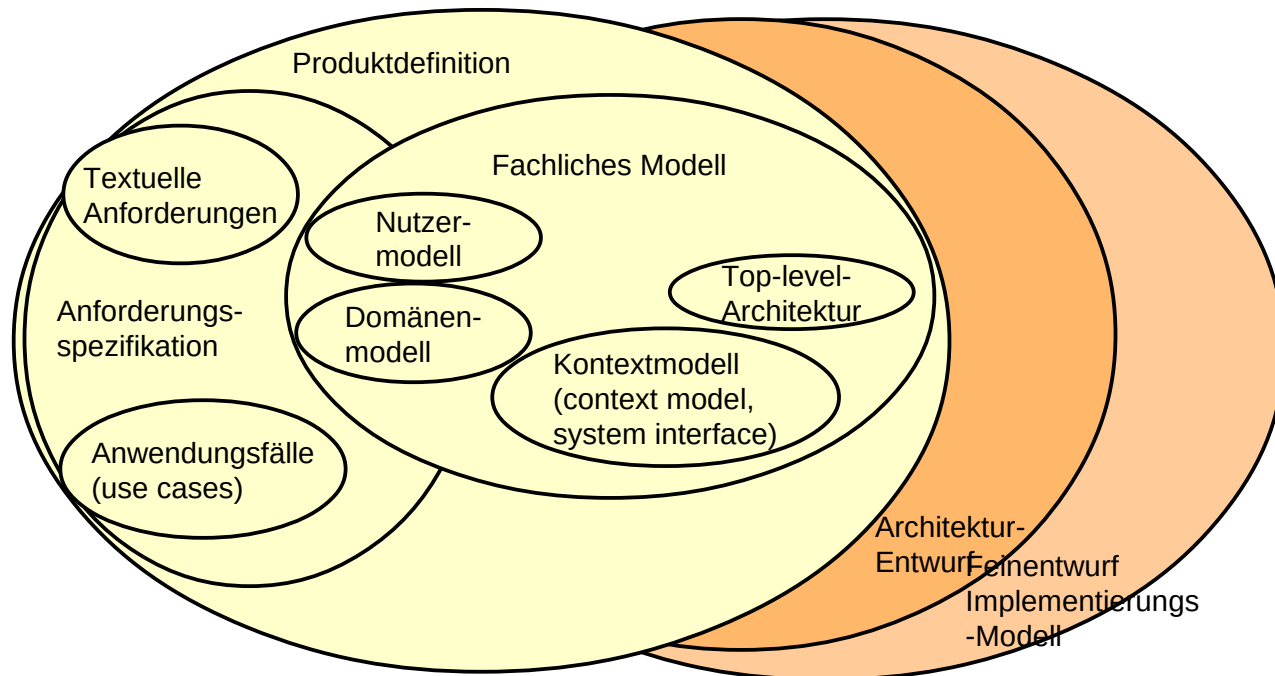
# Eigenschaftshierarchie

- ▶ Auch integrates-a-Hierarchien kann man als auch als Zeilenhierarchien anordnen
- ▶ Integrates-a stellt Hierarchien von Prädikaten über ein großes Objekt dar
- ▶ Eine Integrationshierarchie verwendet die Endo-Assoziation integrates-a



# Vom Analysemodell zum Entwurfsmodell

- ▶ Analysemodelle (in aUML) sind *größer* als Entwurfsmodelle (in dUML).
  - Beim Übergang zum Entwurfsmodell wird das Analysemodell *verfeinert*, d.h. *ausgefüllt* und *detailliert*.
  - Das Analysemodell ist sozusagen das *Skelett* des Entwurfsmodells
    - Domänenmodell ergibt Kontextmodell ergibt Top-Level-Architektur ergibt Architekturmodell ergibt Implementierungsmodell
    - Domänenmodell ergibt Materialentwurf



Das **Analysemodell** besteht aus Fragmentgruppen (Fragmenten und generischen Fragmenten) von UML

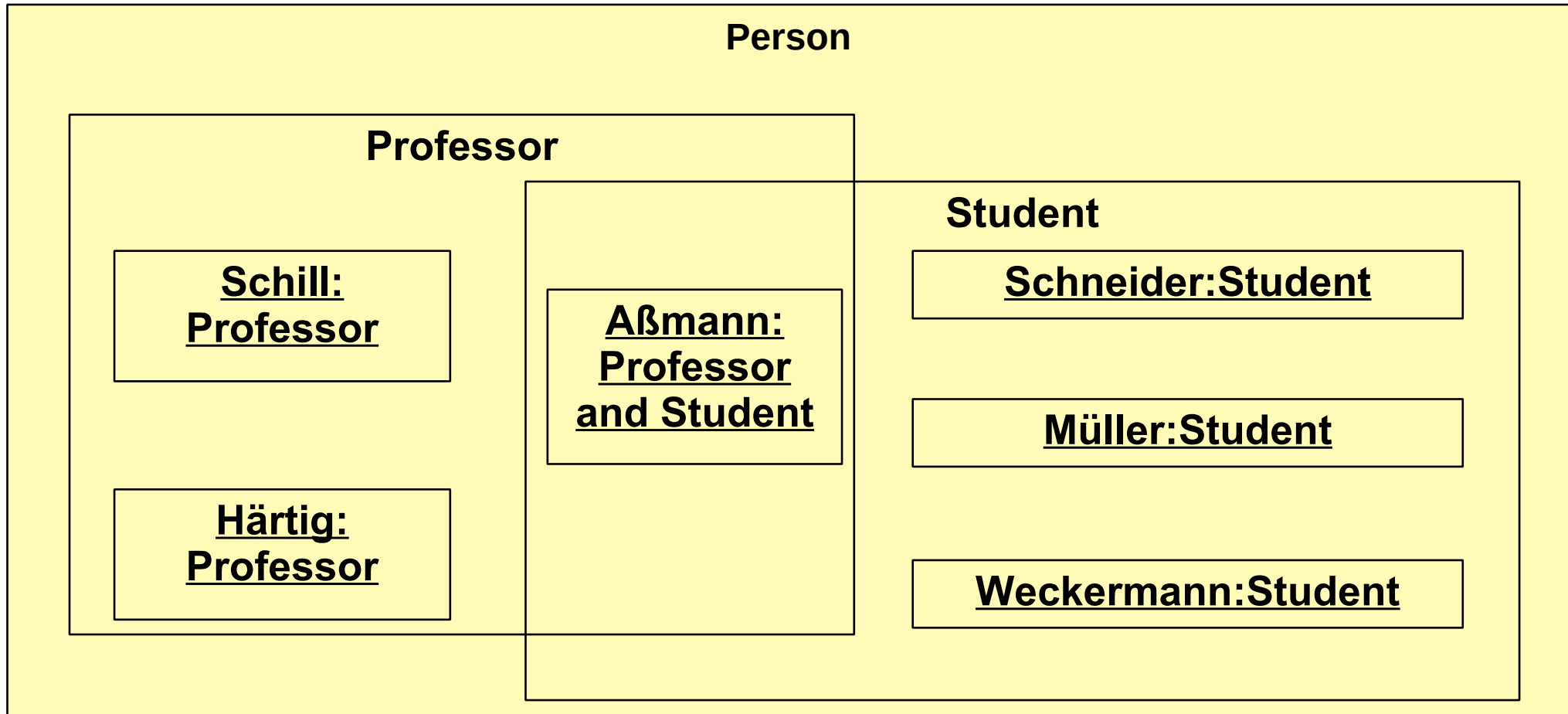
- ▶ Verfeinerungsschritte vom Analysemodell (aUML) zum Entwurfsmodell (dUML)
  - **Vervollständigung** (Ausfüllen, Elaboration) von Fragmenten zu vollständigen Modellen
    - Detaillierung von Fragmenten mit optionalen Einzelheiten
    - Anfügen von Typen, Multiplizitäten und Constraints
  - **Strukturierung**, Restrukturierung von Fragmenten
    - Vererbung, Generizität, Entwurfsmuster
  - **Abflachen** von Fragmenten (Flachklopfen, lowering, **Realisierung**): Ersetzen von ausdrucksstarken Konstrukten durch weniger ausdrucksstarke, implementierungsnähere
  - **Erhöhung der Zuverlässigkeit**: Einziehen von qualitätssteigernden Fragmenten
    - z.B. Typen von Parametern, generische Typen

# Was man vom Analysemodell abflachen muss

- ▶ Das Analysemodell nutzt verschiedene ausdrucksstarke Sprachkonstrukte (hier aUML), die nicht in der Programmiersprache (Java) vorhanden sind
- ▶ Beim Übergang vom Analysemodell und Entwurfsmodell zum Implementierungsmodell muss man diese in die Programmiersprache umsetzen (*Realisieren, Flachklopfen, lowering*) (→ Teil IV)
- ▶ Abflachen struktureller Eigenschaften
  - Klassen und Objekte:
    - Stereotypen aus Profilen
    - Mehrfachvererbung von Code
    - Aktive Objekte (Parallelität)
    - Komplexe Objekte wie Unterobjekte (Rollen, Facetten, Teile)
    - Konnektoren und Kanäle
  - Relationen:
    - n-stellige Assoziationen, bidirektionale Assoziationen
    - Aggregationen, Kompositionen
- ▶ Abflachen von Verhalten
  - States
  - Activites
  - Ereignisse auffangen und behandeln

# A.31.5. Mehrfachvererbung als Venn-Diagramm

- ▶ Betrachtet man Klassen als Mengen, bildet sich Unterklassenbeziehung auf Teilmengenbeziehung ab
- ▶ Mehrfach-Vererbung ergibt ein Venn-Diagramm mit Überschneidungen







## OOA (Teil III)

### 31) Strukturelle Analyse, getrieben von aus Metamodellen abgeleiteten Fragen

- Wie verstehe ich bloß genau, was mein Kunde will?

Prof. Dr. rer. nat. Uwe Aßmann  
Institut für Software- und Multimediatechnik  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
TU Dresden  
Version 19.0.1, 15.06.19

- 1) Metamodelle
- 2) Analyse von Klassen und Merkmalen
- 3) Analyse von Klassen-Beziehungen
- 4) Analyse von Endo-Relationen
  - 1) Aggregation und Komposition
- 5) Mehrfachvererbung



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

- ▶ Zuser, Kap. 7-9
- ▶ Störrle 5.2-5.5
- ▶ Balzert Kap. 6-7, 9-10. Die strukturelle, metamodellgetriebene Analyse wird im Balzert diskutiert.

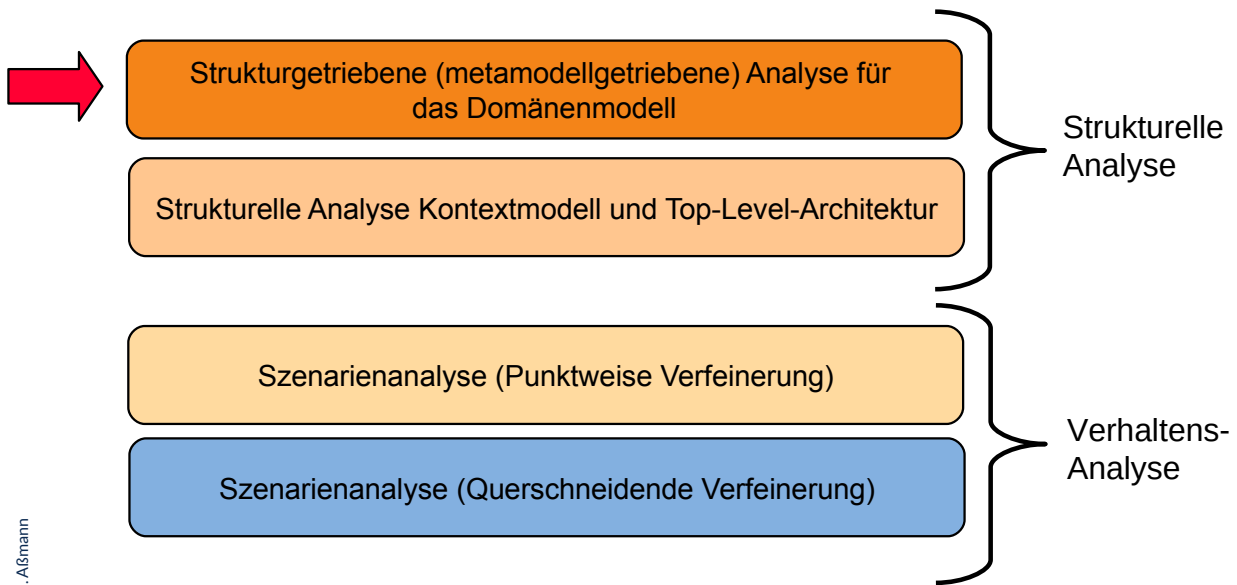
Nicht-obligatorisch:

- ▶ OOA ist eine spezielle Form des Requirements Engineering
  - <https://www.sophist.de/anforderungen/requirements-engineering/faq-requirements-engineering/>

Notation: Wir schreiben:


- ▶ Analysefragen in Font LeckerliOne: *Which tools does a text contain?*
- ▶ Zu analysierende Texte des Kunden in Font ComicJens: *Eine Teambesprechung ist ein Termin.*

# Objektorientierte Analyseverfahren



## Überblick Teil III: Objektorientierte Analyse (OOA)

4 Softwaretechnologie (ST)

1. Überblick Objektorientierte Analyse
  1. (schon gehabt:) Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodelgetriebene Modellierung mit UML
  1. Strukturelle metamodelgetriebene Modellierung für das Domänenmodell
  2. Strukturelle Modellierung von komplexen Objekten
  3. Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensanalyse)
  1. Funktionale Verfeinerung: Dynamische Modellierung und Szenarienanalyse mit Aktionsdiagrammen
  2. Funktionale querschnittende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
  3. (Funktionale querschnittende Verfeinerung für komplexe Objekte)
4. Beispiel Fallstudie EU-Rent



## Exkurs E31.2: Lernen und Hierarchien

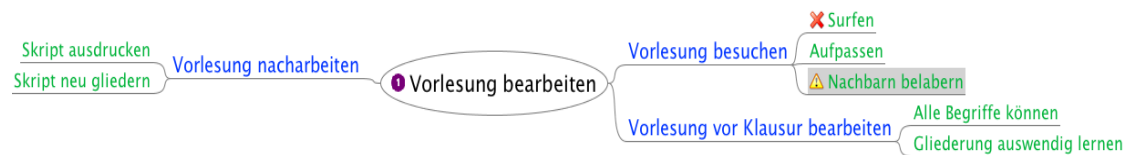
- ▶ Zum Lernen verwendet man Begriffshierarchien [Wolf, 2.1]
  - Einen Lernstoff arbeitet man von einer Quelle aus in eine Begriffshierarchie um
  - Aus einem Buch, aus einer Vorlesung, aus einer Diskussion oder Brainstorm
- ▶ Die Begriffshierarchie formt man so lange um, bis sie vollständig das Lerngebiet abdeckt und man sich abei allen Begriffen “wohlfühlt”
- ▶ Wohlfühlen heißt:
  - Man kann für jeden Begriff der Begriffshierarchie eine Definition geben
  - Man kann den Begriff anderen erklären
  - Man kann die Begriffe gegen verwandte Begriffe abgrenzen bzw. sie vergleichen
  - Man kann zu einem Begriff ein Problem schildern, bei dem der Begriff eine Rolle spielt
  - Man kann eine Aufgabe lösen, die mit dem Begriff zu tun hat
- ▶ Die Begriffshierarchie muss vollständig auswendig gelernt werden: sie muss zuerst in den Kopf, dann ins Herz wandern (siehe Bloomsche Taxonomie des Lernens)

Haben Sie schon eine Begriffshierarchie für alle  
Begriffe der Vorlesung begonnen?



# Bsp.: Lernen mit hierarchischen Wissenskarten (Mind Maps)

- ▶ Mindmaps (Wissenskarten) sind Begriffshierarchien, die von der Mitte des Blattes her monozentral gezeichnet werden ("mapping your mind")
- ▶ Sie können als Startpunkt beim Lernen eingesetzt werden
  - Untersuchen Sie, ob Sie mit Zeilenhierarchien oder Mindmaps besser zurecht kommen
  - Aufgabe: Suchen Sie mit Google nach freien Mindmap-Werkzeugen für den Computer Ihrer Wahl
  - Zeichnen Sie eine Mind map von diesem Kapitel



Objektorientierte Analyse analysiert die Texte des Kunden mit objektorientierten Mitteln, um Missverständnisse zu vermeiden.

Der natürlichsprachliche Text wird in ein Modell einer objektorientierten Sprache übersetzt (Verb-Substantiv-Analyse).

User Story 1

User Story 2

User Story 3

Modell in  
objektorientierter  
Sprache

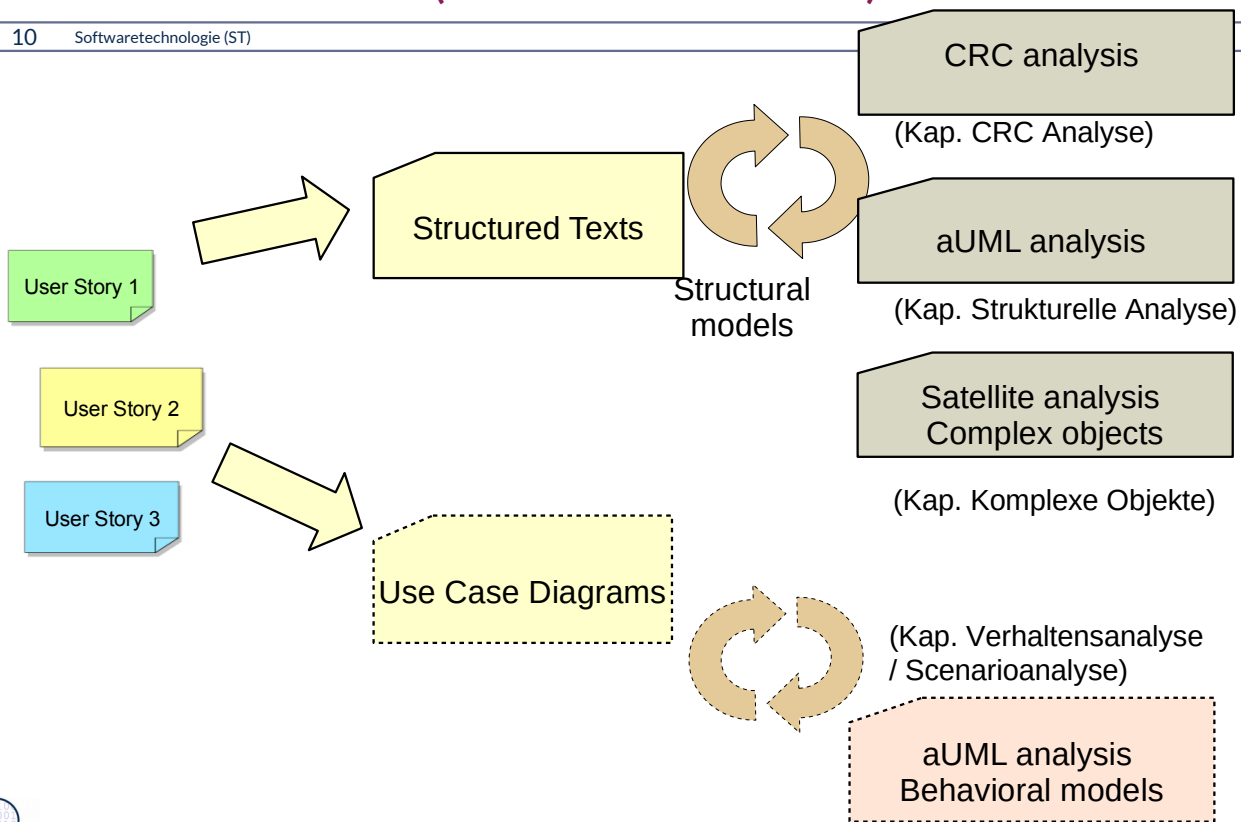
„Softwareingenieure machen die Analyse genauer als Psychoanalysten.“

## Vorteile:

- Das Modell ist präzise, die User Stories sind ungenau
- Das Modell ist maschinenverarbeitbar, da es einer Sprache mit ihren Regeln folgt
-

## Problem: Wie komme ich von Interviews zum Gerüst des fachlichen Modells? (Inhalt des Pflichtenhefts)

10 Softwaretechnologie (ST)



Es gibt völlig verschiedene Arten der Analyse und völlig verschiedenen Modellierungssprachen, die benutzt werden. In diesem Kurs werden CRC-Modelle, aUML-Sprachstrukturmodelle, Satellitenmodelle (UML mit Satellitenprofil) sowie aUML-Verhaltensmodelle gelehrt.



## Attention: Model the Real Need of the Customer

11 Softwaretechnologie (ST)

„I need a machine washing my clothes.“



→ Prototype of a  
washing machine

**Find the right abstractions  
in modeling!**

[Andreas  
Ludwig]

Es ist unheimlich leicht, Kunden misszuverstehen.

Der Softwareingenieur muss das Problem und das Ziel des Kunden verstehen, um seine Welt korrekt zu modellieren (→ Validation).

Dazu muss er aber intensiv mit dem Kunden reden, um die richtigen Modellbildungen zu betreiben.

# Famous Misunderstandings

- ▶ “Do you have four-volt, two-watt light bulbs?
- ▶ For what?
- ▶ No, two watt.
- ▶ To what purpose does this go?
- ▶ No, two watt.....”
- ▶ [Internet Joke. cited from James C. Jarrad. The Case of the Missing Pronoun. Xlibris. 2012]



Spoken is not heard  
Heard is not listened  
Listened is not understood  
Understood is not accepted  
Accepted is not done

Objektorientierte Analyse analysiert die Texte des Kunden mit objektorientierten Mitteln, um Missverständnisse zu vermeiden.

Der natürlichsprachliche Text wird in ein Modell einer **formalen** objektorientierten Sprache mit **Vokabularium (Metamodell)** übersetzt.

User Story 1

User Story 2

User Story 3

Modell in **formaler** objektorientierter Sprache

„Softwareingenieure machen die Analyse genauer als Psychoanalysten.“

## Vorteile:

- Das Modell ist präzise, die User Stories sind ungenau
- Das Modell ist maschinenverarbeitbar, da es einer Sprache mit ihren Regeln folgt
-

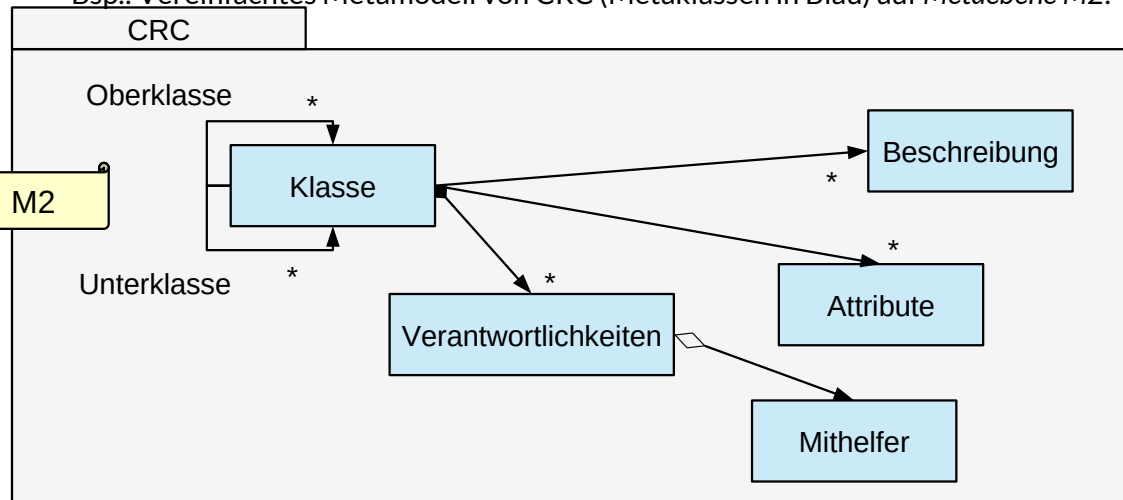


## 31.1 Wie man aus Sprachstrukturmodellen (Metamodellen, Vokabularien) Fragen für die Analyse gewinnt

Modelle sind überall...  
Ihre Struktur wird beschrieben durch *Metamodelle*



- ▶ Ein **Sprachstrukturmodell (Metamodell, Vokabularium)** beschreibt die *Struktur einer Menge von Modellen* mit Hilfe eines (Meta-)Klassendiagramms, d.i. die *Struktur einer Modellierungssprache*
- ▶ Ein **Sprachstrukturmodell (Metamodell, Vokabularium)** verallgemeinert eine Begriffshierarchie (Taxonomie) mit Beziehungen zwischen den Begriffen. Ein Metamodell ist ein *Vokabularium* von Sprachkonzepten und ihren Beziehungen.
- ▶ Bsp.: Vereinfachtes Metamodell von CRC (Metaklassen in Blau) auf *Metaebene M2*:



Ein Metamodell enthält die Konzepte einer Sprache und ihre Beziehungen. Ist ein Metamodell eine Taxonomie, sprechen wir von einer Begriffshierarchie (von Sprachkonzepten).

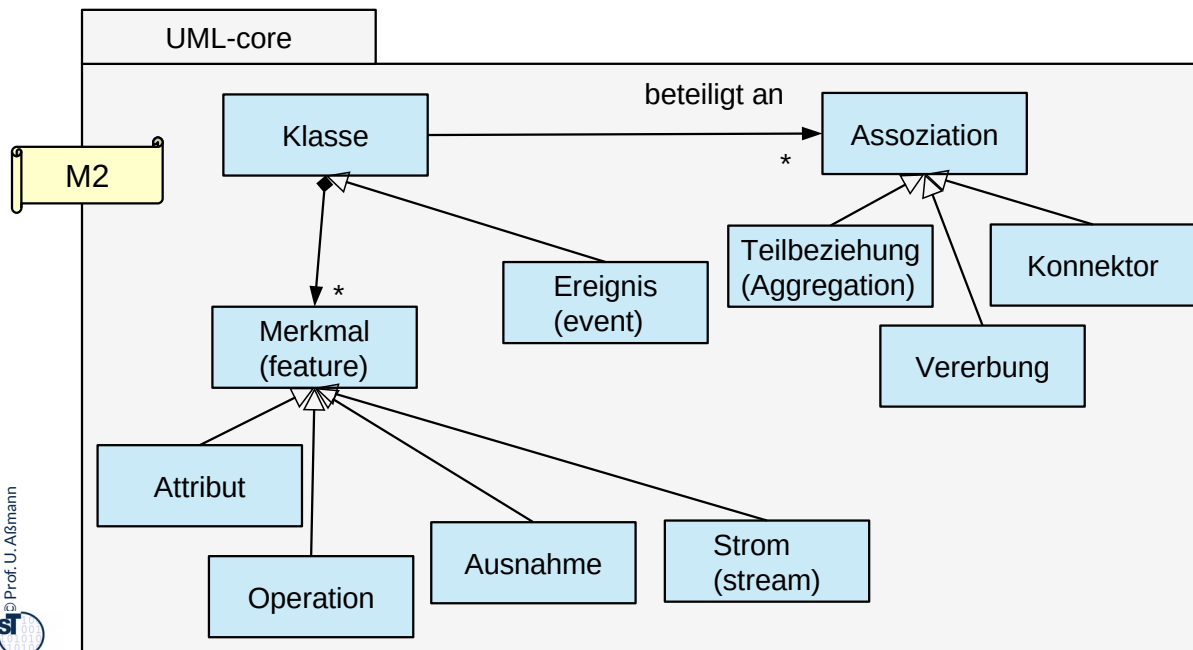
Metaklassen entsprechen also Sprachkonzepte.

Die Struktur des Metamodells bildet die Struktur der Sprache; d.h. es schreibt vor, wie die Modellelemente eines Modells (oder die Programmelemente eines Programms) strukturell zueinander stehen.

Metamodelle kommen als Spezifikationen in Sprachberichten vor.

# Sprachstrukturmodelle (Metamodelle)

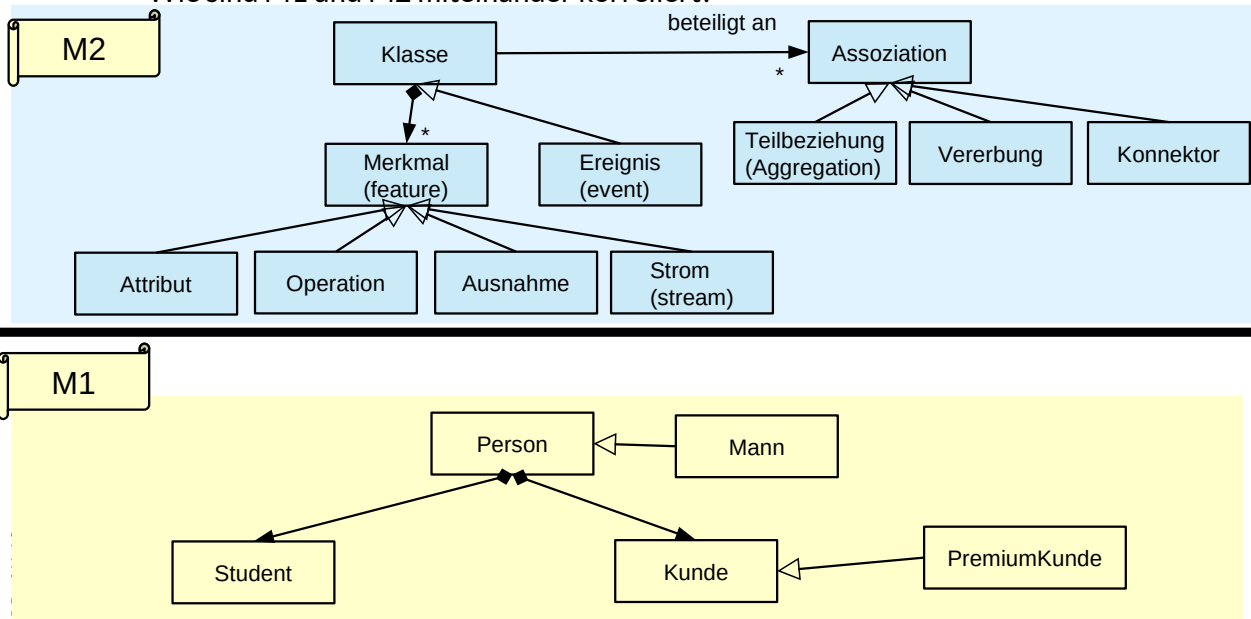
- ▶ In einem Metamodell (Vokabularium) stellen die Metaklassen die *Sprachkonzepte* der modellierten Sprache vor.
- ▶ Vereinfachtes Metamodell von UML:



UML ist mit Hilfe von Metamodellen definiert. Der Sprachbericht enthält sehr viele, von einem Kern (UML-core) ausgehende, immer weiter erweiterte Metamodelle.

# Q: Wie sind Modelle mit ihren Metamodellen (Vokabularien) verbunden?

- ▶ Sprachstrukturmodelle (Metamodelle, Vokabularien) liegen auf einer neuen Modellierungsebene M2
- ▶ Wie sind M1 und M2 miteinander korreliert?



M0 – enthält Objekte.

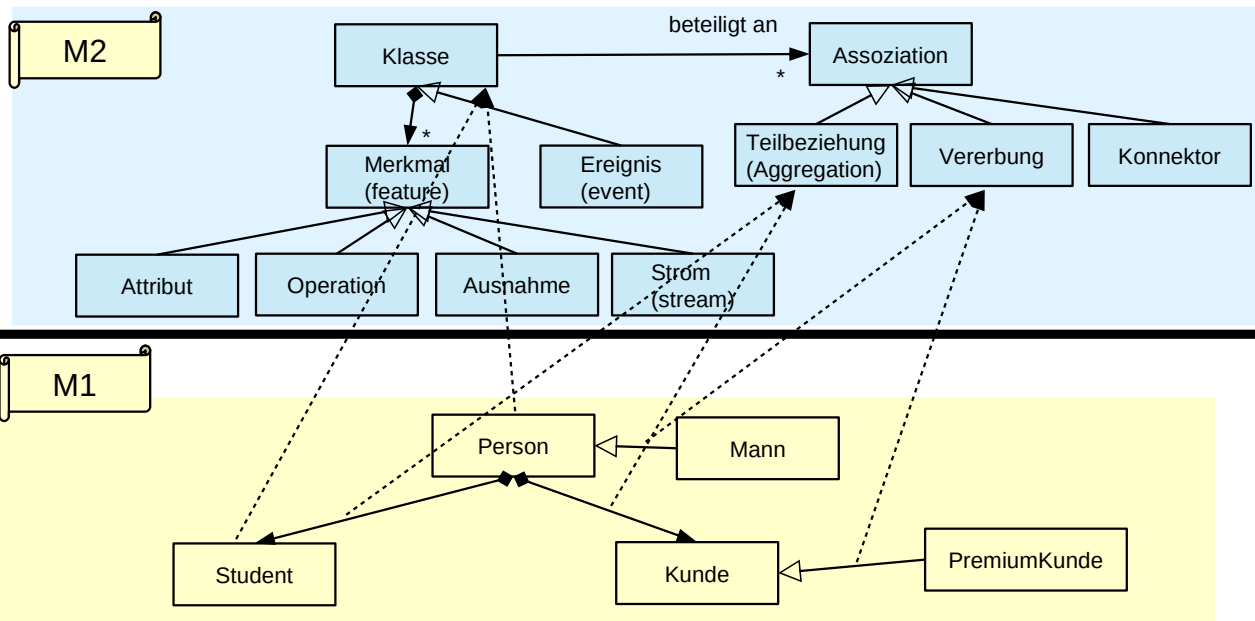
M1 – enthält die Klassen, Typen und den Code.

M2 – enthält die Sprachkonzepte (Metaklassen).



# A: instanceof Relation koppelt alle Modellelemente zu einem Element des Metamodells

- ▶ Klassen und weitere Elemente von M1 sind durch instanceof verbunden (Darstellung nur auszugsweise)



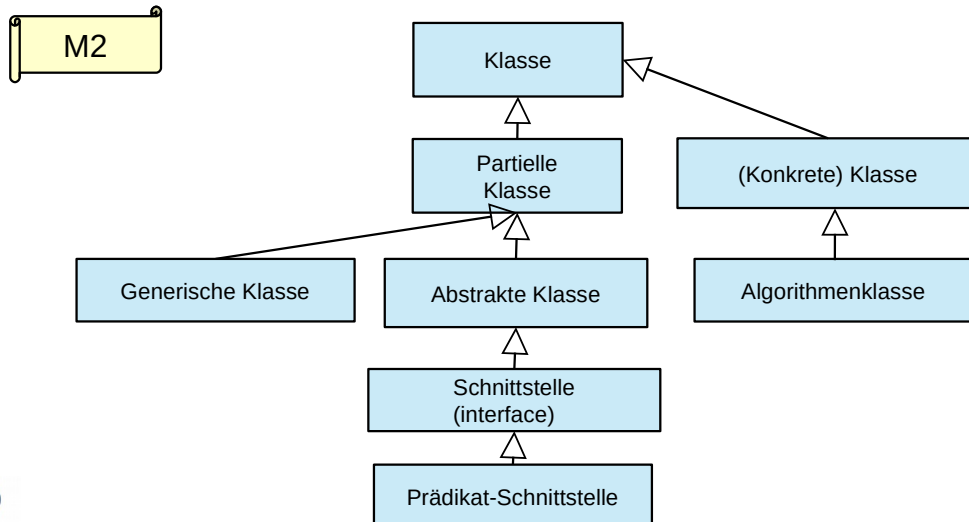
M0 – enthält Objekte.

M1 – enthält die Klassen, Typen und den Code.

M2 – enthält die Sprachkonzepte (Metaklassen).

## Exkurs 31.E.1 Begriffshierarchie von Klassen (Erinnerung)

- ▶ Jede Folie des roten Fadens der Klassenarten bestand aus einem Metamodell, d.h. die Begriffshierarchien enthalten **Metaklassen (Sprachkonzepte)**
  - Eine Begriffshierarchie ist ein spezielles Metamodell, das hauptsächlich Vererbung verwendet
- ▶ Von diesem Java-Metamodell kann man in Java-Programmen Programmelemente und -fragmente instanzieren

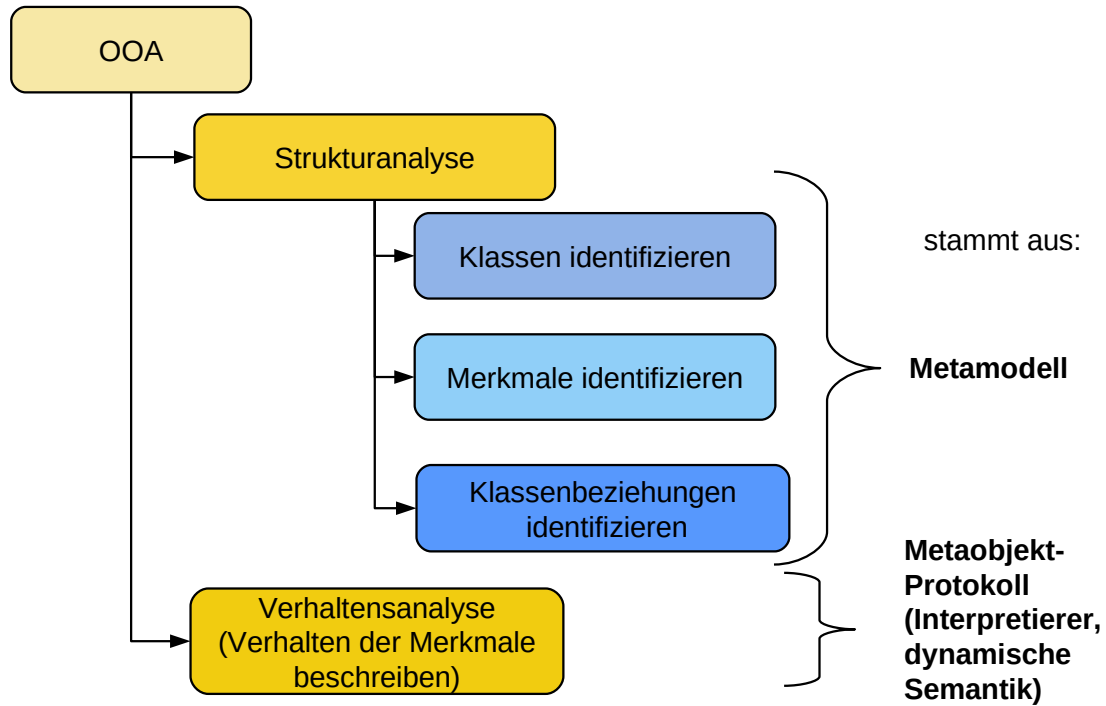


Lernen benötigt Metamodelle.

Das Erarbeiten eines Metamodells für ein Lerngebiet bildet einen wesentlichen Schritt zur Beherrschung des Lerngebiets, denn es enthält alle Begriffe eines Lerngebiets, sowie ihre Beziehungen.

# Hauptschritte der strukturellen, metamodelgetriebenen Analyse mit UML

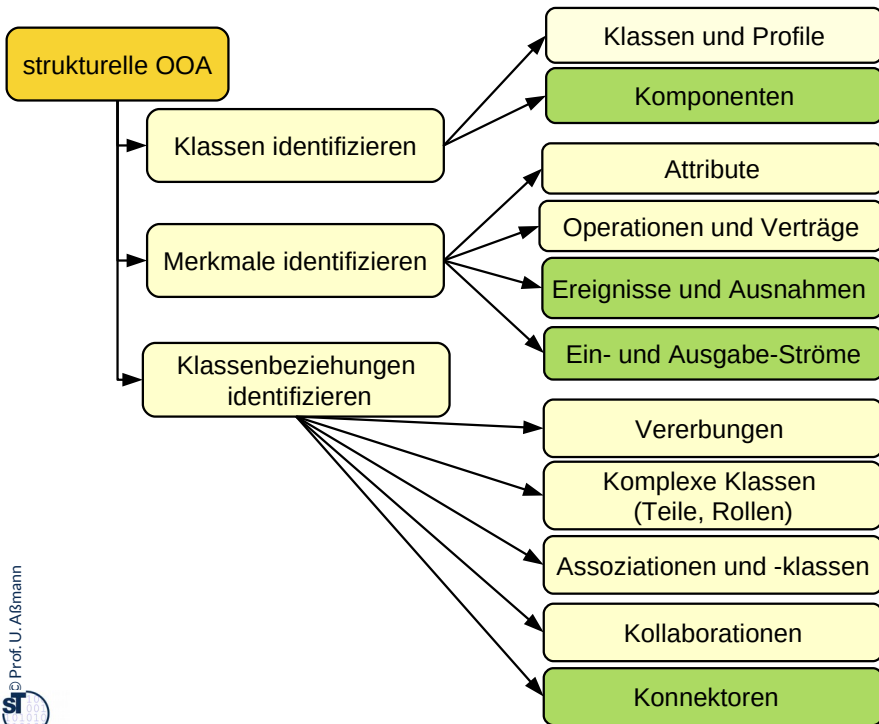
- ▶ Ähnlich zur CRC Analyse, mit UML-Metamodell, das reichere Struktur besitzt



Man findet diese Methodik z.B. auch in der 10-Schritte-OOA-Methodik, die bei Balzert beschrieben ist.

# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



Während einer **strukturgetriebenen Analyse** sucht man in der Problemwelt des Kunden nach Ausprägungen aller Begriffe des Metamodells. Daraus erstellt man ein Analysemodell.

Während der **strukturgetriebenen Text-Analyse** sucht man in dem Interviewtext des Kunden nach Ausprägungen aller Begriffe des Metamodells. Daraus erstellt man ein Analysemodell.

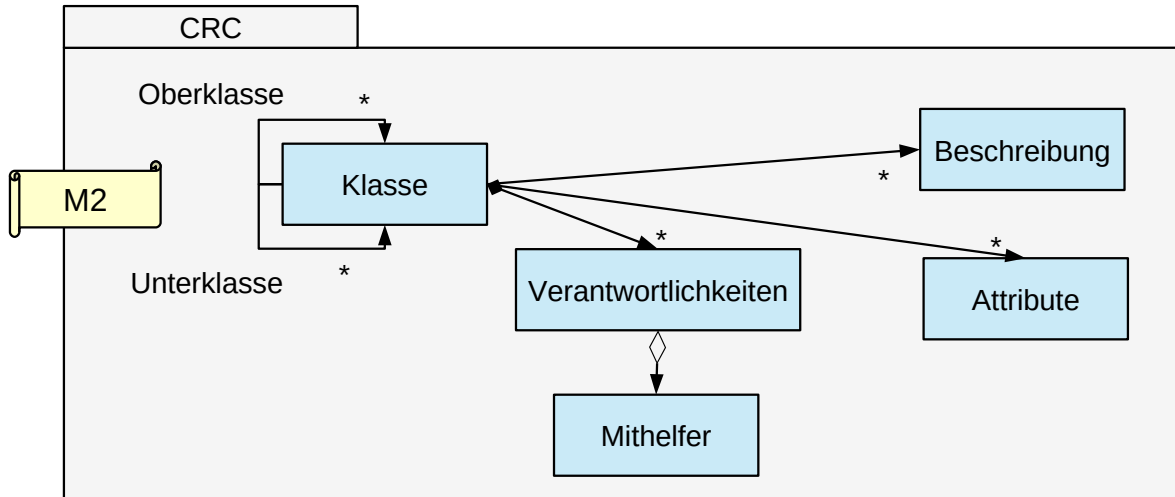
**Man nutzt das Metamodell, um Analysefragen abzuleiten:**

- Enthält ein Metamodell das Konzept einer **Klasse**, fragen wir den Kunden *„Welche Klassen und Objekte brauchen wir?“*
  - Enthält ein Metamodell das Konzept einer **Methode**, fragen wir *„Welche Methoden brauchen wir?“*
  - Enthält ein Metamodell das Konzept eines **Konnektors**, fragen wir *„Welche Konnektoren brauchen wir?“*
- usw.

Die strukturelle Analyse findet strukturgetrieben statt, d.h. die Elemente des Metamodells werden genutzt, um nach den Inhalten der Modelle zu fragen.

- Enthält das Metamodell das Element “Klasse”, fragt man nach den Klassen der Modelle.
- Enthält das Metamodell das Element “Stream”, fragt man nach den Streams in den Modellen.
- Sind im Metamodell Beziehungen zwischen “Klasse” und “Stream” festgelegt, also z.B. “Klasse hat Streams”, dann müssen auch im Modell zwischen Klassen und Konnektoren Beziehungen (has-a) angelegt werden, d.h. man muss Ströme Klassen zuordnen.
- Die Schritte der metamodellgetriebenen Analyse gelten für alle Arten von Analysemodellen in der Produktdefinition
  - Anforderungsspezifikationen
  - Fachliche Modelle (Domänenmodell, Kontextmodell, Top-level-Architektur)
- Im folgenden werden sie hauptsächlich für ein Analysemodell dargestellt, das alle obigen Teilmodelle integriert
  - Man kann aber Klassen den jeweiligen Teilmodellen zuordnen
- Wir werden noch weitere Analyseschritte für Kontextmodell und

# Ableitung von Analysefragen aus dem Metamodell



Die CRC-Methodik ist eine besondere Ausprägung der strukturgetriebenen Analyse.

- ▶ *Welche Klassen haben welche Verantwortlichkeiten?*
- ▶ *Wie kann man die Verantwortlichkeiten beschreiben?*
- ▶ *Wen hilft mit, die Verantwortlichkeit zu erledigen?*
- ▶ *Welche Attribute hat eine Klasse?*
- ▶ *Welche Attribute teilt sie mit anderen Klassen und gehören in eine Oberklasse?*



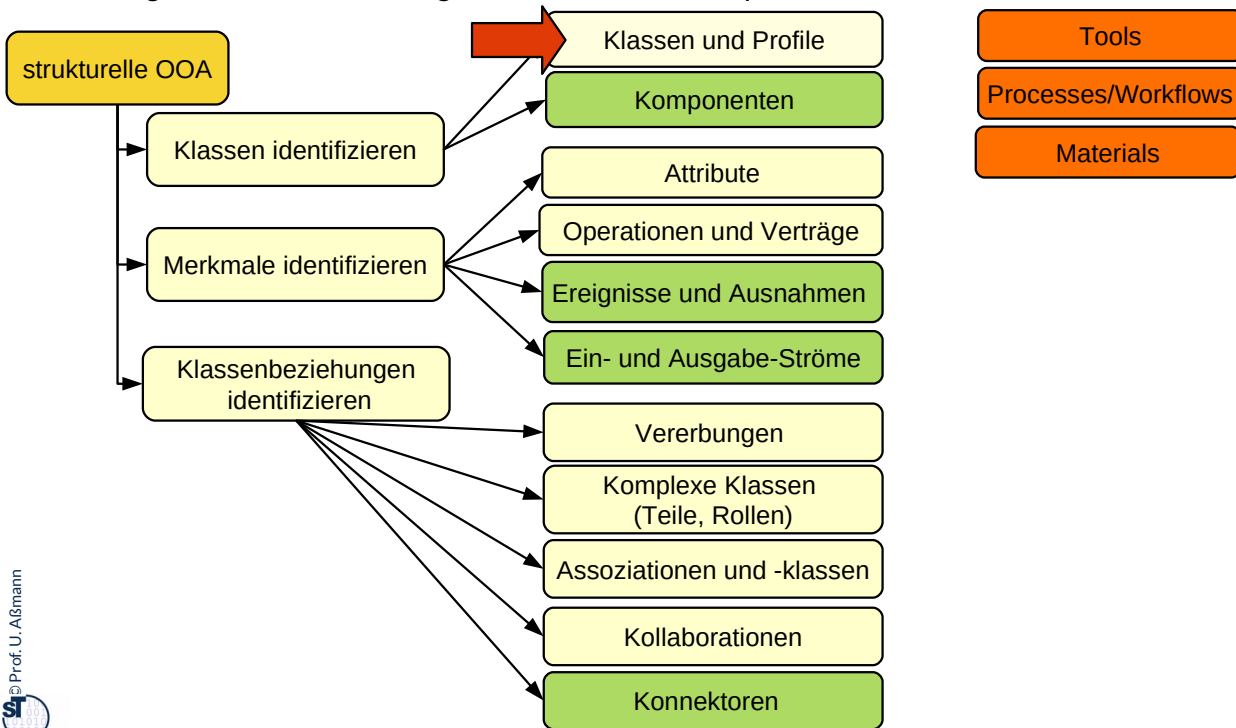


## 31.2 Analyse von Klassen und Merkmalen



# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

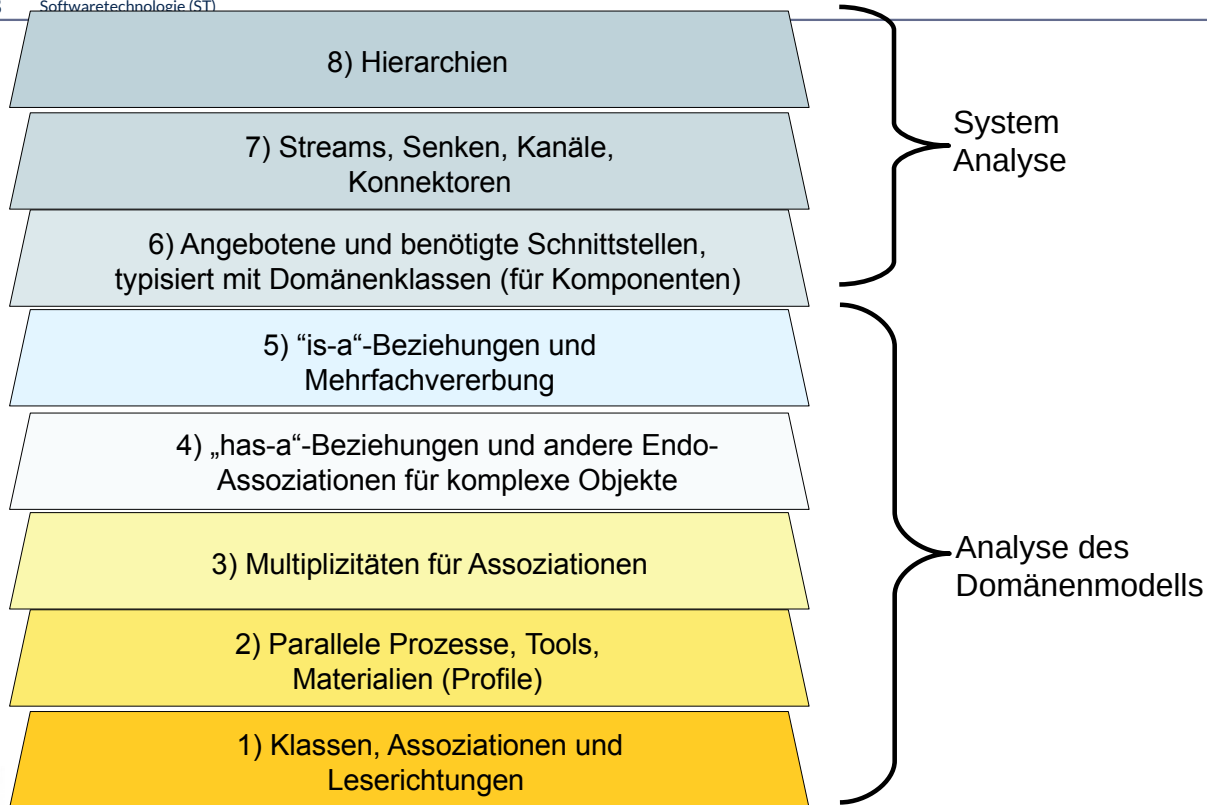
► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur





## Schritte der Anreicherung des Analysemodells “Was man alles so im Text des Kunden sucht”

28 Softwaretechnologie (ST)



Ein initiales Analysemodell, das aus dem ersten Interview mit dem Kunden stammt, muss i.d.R. verfeinert, d.h. angereichert werden. Daher muss der Analyst mehrere Durchgänge mit dem Kunden organisieren. Er muss nach Multiplizitäten, Ganz-Teile-Beziehungen, Vererbungsbeziehungen, benötigten Schnittstellen, sowie Kommunikationskanälen fragen.

Mehrere Durchgänge verfeinern das Analysemodell und machen es inhaltsreicher und präziser. Für die Aufnahme in das Pflichtenheft sollte es möglichst präzise sein.

- ▶ Ein Analysemodell entsteht aus einem Interview mit einem Kunden, in dem Sätze in Fragmente des Analysemodells umgesetzt werden (“user story”)
- ▶ Ein Analysemodell kann “vorgelesen” werden, und es entsteht der Text des Interviews

### Erstellen Sie mir eine Terminverwaltung.

*Teams, bestehend aus Mitgliedern, müssen Besprechungen buchen können. Ein Team hat einen Leiter. Besprechungsräume müssen einer Teambesprechung zugebucht werden. Eine Besprechung hat einen Titel, einen Beginn, eine Dauer, und ist mit einer maximalen Anzahl von Teilnehmern zu versehen. Eine Besprechung ist ein Termin.*

*Private Termine sollen auch aufgenommen werden, damit Konflikte der Teammitglieder erfasst werden.*

Das Tonband des Kundeninterviews (user story) wird I.d.R. mehrfach abgehört, um das initiale Analysediagramm zu erstellen.

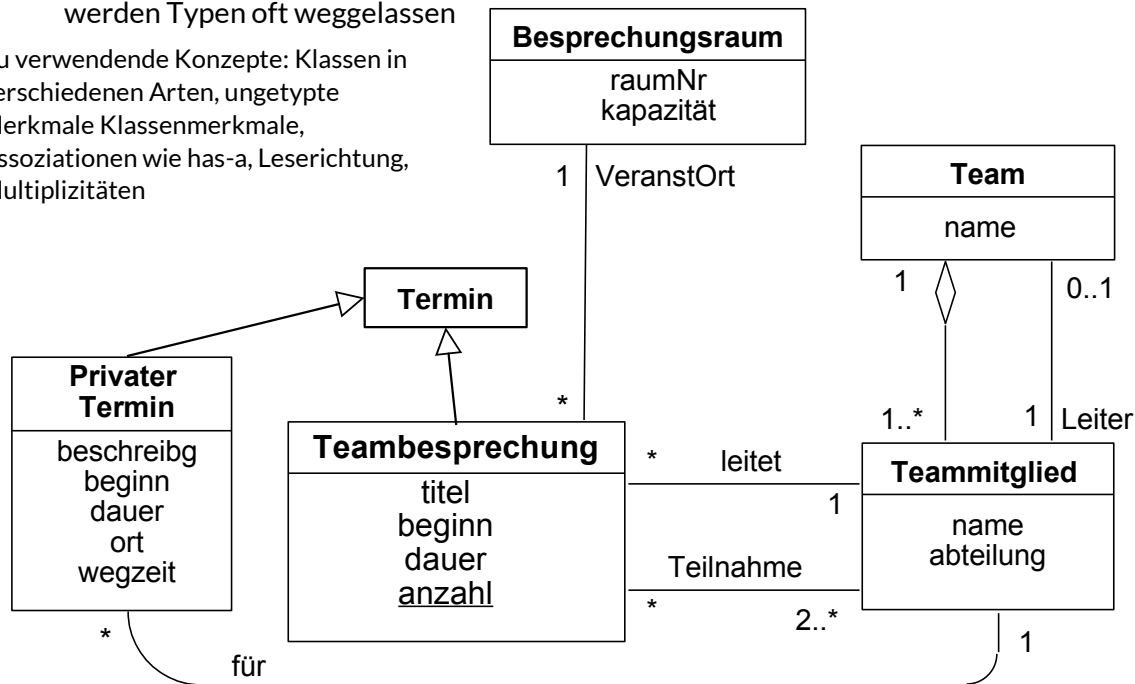
Jeder Satz wird in ein Fragment des Analysemodells umgewandelt. Nicht erfasste Sätze werden rot markiert, damit sie nicht vergessen werden.

Aus dem Analysemodell kann durch Vorlesen (oder Prettyprinting) ein Text erstellt werden, der dem Original-Interview-Text möglichst ähnlich sein sollte.

## Beispiel: Analysemodelle beginnen mit unvollständiger Information zu Attributen und Methoden (Fragmente)

- ▶ Ein Analysemodell entsteht aus einem Interview mit einem Kunden
- ▶ Es ist immer **fragmentarisch**, d.h. partiell, und enthält **Fragmente** von UML-CD, z.B. werden Typen oft weggelassen

- ▶ Zu verwendende Konzepte: Klassen in verschiedenen Arten, ungetypte Merkmale Klassenmerkmale, Assoziationen wie has-a, Leserichtung, Multiplizitäten



# Übung: Reproduktion des Analysetextes

- ▶ Versuchen Sie, das Modell auf der vorigen Folie in einen Text rückzuverwandeln.
- ▶ Vorgehensweise:
  - Notieren Sie sich alle wichtigen Analysefragen aus dem Metamodell der UML-Klassendiagrammen
  - Schrittweises Vorgehen:
    - Setzen Sie ein Häkchen an jedes bereits in einen Satz rückverwandelte Modellelement.
- ▶ Nachuntersuchung:
  - Fragen Sie sich, ob sie alle Analysefragen berücksichtigt haben.
  - Wie unterscheidet sich Ihr generierter Text von der User Story? Warum ist er (in der Regel) nicht gleich?



# Aufgabe der Analyse von Klassen in den einzelnen Schichten des Systems

- ▶ im **Domänenmodell**:
  - Klassen können sowohl Objekte als Begriffe repräsentieren
  - Vererbung zwischen Begriffen bilden Metamodelle (Begriffshierarchien, Taxonomien)
  - Assoziation, Aggregation und Komposition (Ganz-Teile-Beziehungen)
- ▶ im **Analysemodell (funktionale Anforderungen)**:
  - Klassen repräsentieren Daten, die der Kunde als Ergebnis der Funktionen des Systems haben will
- ▶ in **Kontextmodell und Top-level-Architektur**:
  - Klassen repräsentieren Daten, die auf Kanälen fließen
  - Prozesse, die die Daten bearbeiten
  - in funktionalen Anforderungen: funktionale Anforderungen werden Operationen, die Klassen zugeordnet sind
- ▶ Im **Datenmodell** ("Materialien" aus Domänenmodell abgeleitet):
  - Klassen repräsentieren passive Materialien, die Wissen repräsentieren



## 31.2.2 Stereotypen und Profile



- ▶ Für die Spezifikation von Analysemodellen erlaubt es UML, Klassen mit **Stereotypen** zu annotieren
  - Klassen werden “markiert”, klassifiziert, dh. mit mehr Semantik ausgestattet
  - Komponenten, Ports, Konnektoren können unterschieden werden
- ▶ Kataloge von Stereotypen heissen **Profile**
  - Wenn der Ingenieur einige Profile (inkl. ihrer Stereotypen) kennt, kann er seine Kontextmodelle mit mehr Semantik ausstatten
  - UML 2.0 superstructure, Appendix B, enthält eine grosse Menge von technischen Standard-Stereotypen
  - Man kann auch seine eigenen Profile definieren

## Profil "Aktive und Passive Klassen"

- ▶ Zum Analysemodell gehört die Unterscheidung von *aktiven Klassen (Prozessen)* und *passiven Klassen*

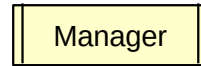
- Im einfachsten Fall existiert *eine* aktive Klasse

- ▶ **Aktive Objekte**

- Akteure, Prozesse

<<actor>>

<<active class>>



- Werkzeuge (Tools): Aktives Objekt, das Materialien schreibt)

<<tool>>

- Workflow (interaktiver Prozess): Prozess, der Tools aufruft

<<workflow>>

<<process>>

- ▶ **Passive Objekte** (Daten, Materials, entities, data objects)

- Aktive Objekte arbeiten auf Materialien

<<material>>

- Persistente Materialien

<<data>>

- Rollen: Dynamische Sicht auf ein Material

<<role>> <<passive class>>

- ▶ **Konnektoren (connectors) und Kanäle (channels, pipes):** Beschreiben, wie aktive Objekte über Datenobjekte miteinander kommunizieren

<<entity>>

- Über Kanäle fließen Datenobjekte, sie werden mit Senken geschrieben und mit Streams gelesen

<<channel>> <<connector>>

<<call>>

<<input stream>>

Profile enthalten Stereotypen, und diese können an Klassen und Objekte, allgemein an alle UML Fragmente annotiert werden.

Besonders nützlich sind für Analysen die Stereotypen:

- Tool (Werkzeug): ein aktives, I.d.R. Parallel laufendes Objekt
- Material: passive Datenobjekte
- Konnektor: Klasse zur Kommunikation



Jedes Stereotyp eines Profils definiert eine Frage für die OOA

- ▶ <<active object>>: *Which active objects does a text contain?*
- ▶ <<tool>>: *Which tools does a text contain?*
- ▶ <<material>>: *Is the object a passive object, a material of a tool?*
- ▶ <<connector>>: *Is the object solely concerned with communication/interaction?*

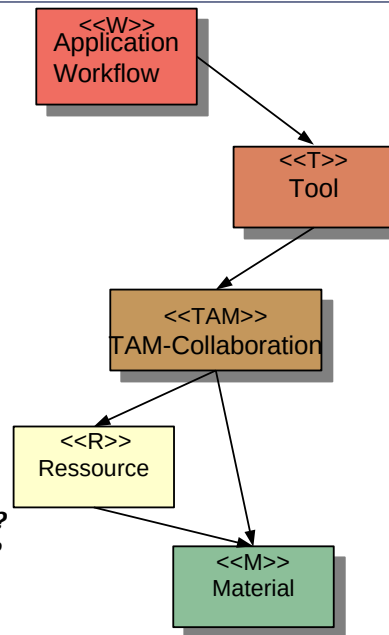
# Profil TAM: Trennung von aktiven und passiven Komponenten

Tools-and-Materials [Züllighoven] ist ein Profil, das folgende Aspekte definiert:

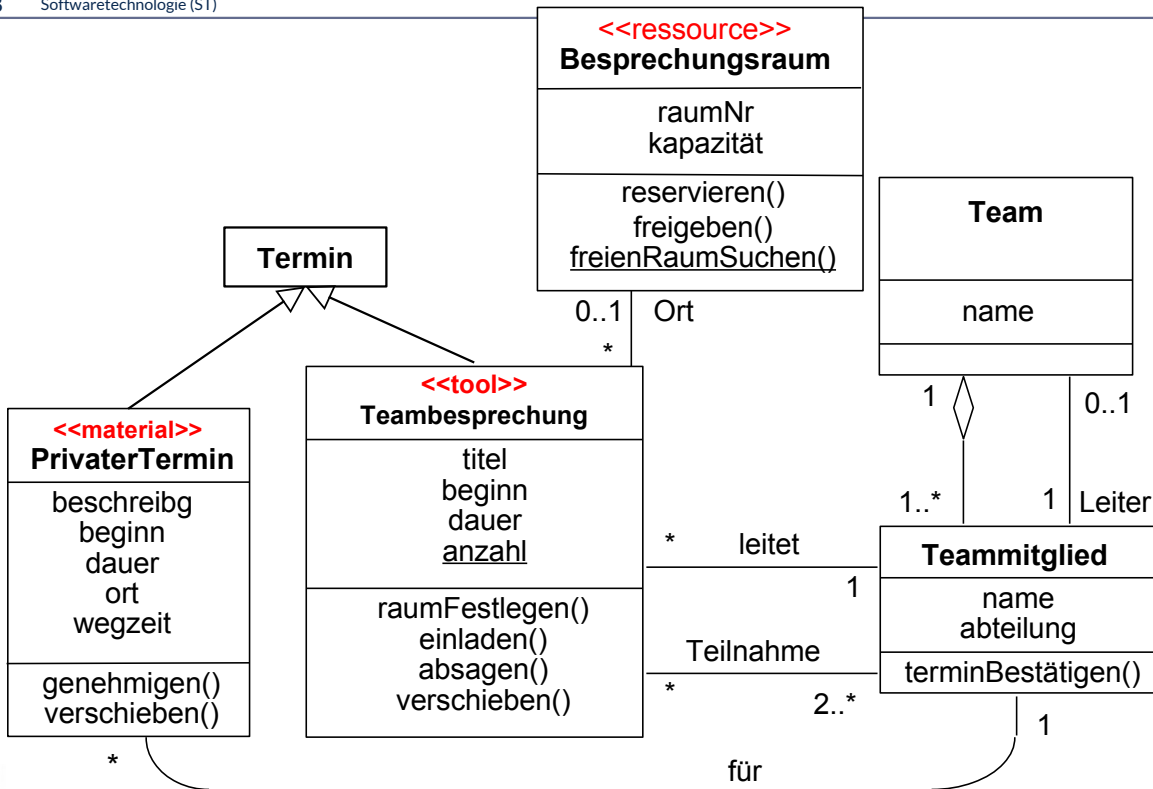
- 1) Tools (aktive Objekte, Command)
- 2) Ressourcen (belegbar)
- 3) Materials (passive Daten, Schicht E)
- 4) TAM-Collaboration
- 5) Workflows (Prozesse) koordinieren Tools

- Klassen, Module, Komponenten, Pakete sollten mit diesen Aspekten qualifiziert werden

- ▶ *Which tools does a text contain?*
- ▶ *Is the object a passive object, a material of a tool?*
- ▶ *How does this tool collaborate with this material?*
- ▶ *How are these tools controlled by a workflow?*
- ▶ *Which materials must be reserved, are resources?*



# Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell



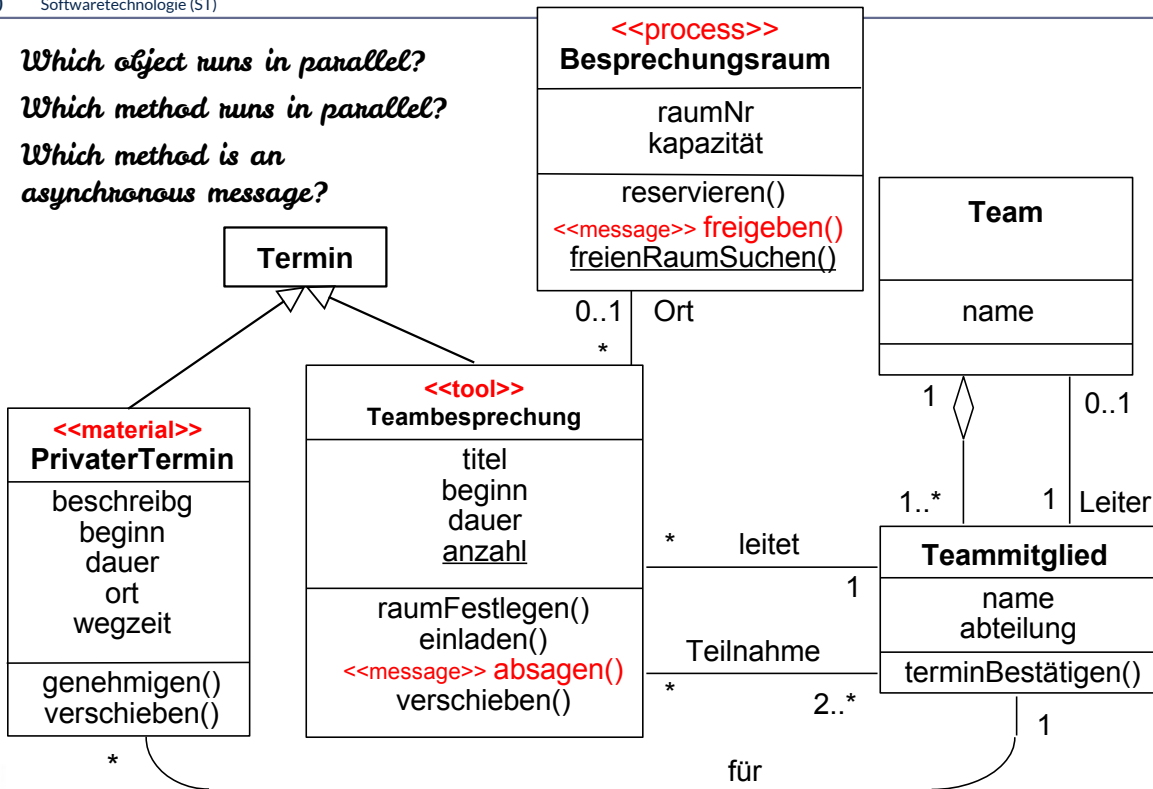
# Einsatz von sequentiellen und parallelen OO Sprachen in der Analyse

- ▶ In der Analyse wird oft **Parallelität** eingesetzt, weil die Prozesse des Domänenmodells parallel zueinander laufen
  - Parallele Objekte arbeiten asynchron
  - sie kommunizieren mit Botschaftenaustausch (messages) und reagieren auf den Empfang einer Nachricht mit dem Ausführen einer (oder mehrerer) Methode(n)
- ▶ In einer **parallelen objekt-orientierten Sprache** setzt sich eine **asynchrone Dienstanfrage (service request)** an ein Objekt zusammen aus:
  - einer **Aufruf-Nachricht (Botschaft, message)**,
  - einer *asynchronen* Ausführung von Methoden (der Sender kann parallel weiterlaufen)
  - optional einer Aufruf-Fertigmeldung (mit Rückgabe), die vom Sender ausdrücklich abgefragt werden muss
- ▶ In einer **sequentiellen objekt-orientierten Sprache** setzt sich eine **synchrone Dienstanfrage** an ein Objekt zusammen aus:
  - einer Aufruf-Nachricht (Botschaft, message),
  - einer *synchronen* Ausführung einer Methoden und
  - einer Aufruf-Fertigmeldung (Aufrufer wartet auf Rückgabe)



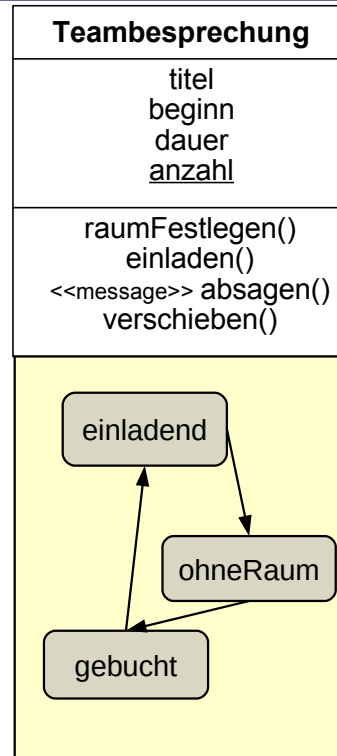
# Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell

- ▶ Which object runs in parallel?
- ▶ Which method runs in parallel?
- ▶ Which method is an asynchronous message?



# Abteile (Compartments) von UML-Klassen

- ▶ Weitere **Abteile (compartments)** in Klassen:
  - Neben dem Attribut- und Operationsabteil können weitere Abteile angehängt werden
  - Verhaltensmodelle können in Abteilen hinzukommen (Statecharts, Aktivitätendiagramme)
  - Mit solchen Verhaltensbeschreibungen können *Objektlebenszyklen* beschrieben werden (siehe später)





## 31.3 Analyse von Klassenbeziehungen und Leserichtungen

- ▶ Texte des Kunden werden in Diagramme umgesetzt und wieder vorgelesen



# Motivation: Gehälter von Requirements-Analysten








- ▶ [https://www.glassdoor.de/Geh%C3%A4lter/requirements-engineer-gehalt-SRCH\\_KO0,21.htm](https://www.glassdoor.de/Geh%C3%A4lter/requirements-engineer-gehalt-SRCH_KO0,21.htm)

Gehälter nach Unternehmen    Sortieren: Beliebtheit | Meiste Gehaltsberichte | Gehalt

Nach Standort filtern    Nach bestimmtem Arbeitgeber suchen

Suchen    oder     Suchen

Unternehmen    Durchschn. Grundgehälter (in EUR)    Niedrig    Hoch

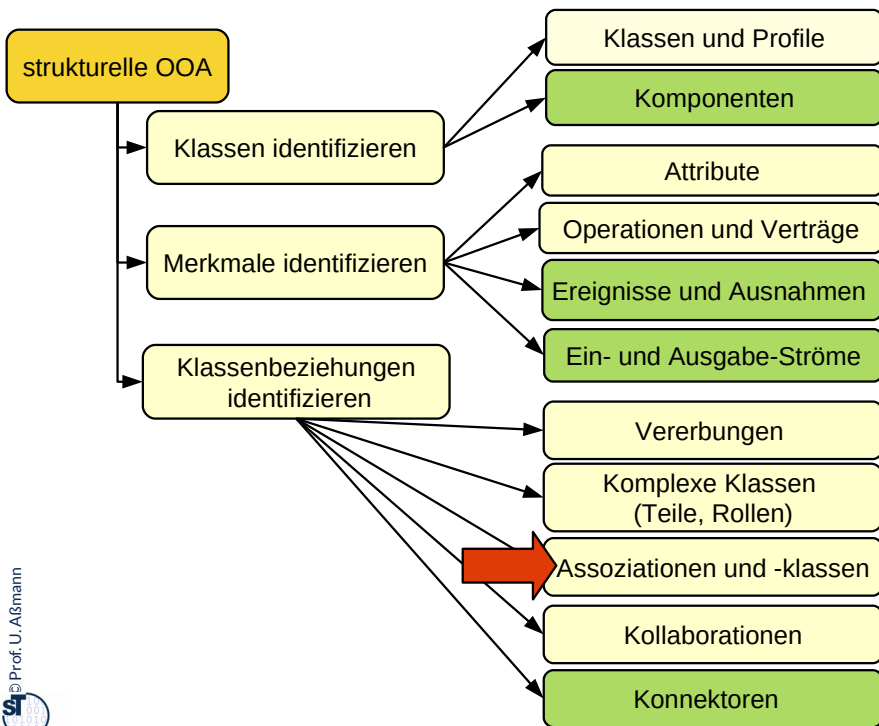
	<b>Requirements Engineer</b> Siemens 1 Mitarbeitergehalt oder Schätzwert	Über 61 Tsd € - 66 Tsd €	61 Tsd €	66 Tsd €
	<b>Requirements Engineer</b> Airbus 1 Mitarbeitergehalt oder Schätzwert	Über 50 Tsd € - 54 Tsd €	50 Tsd €	54 Tsd €
	<b>Requirements Engineer</b> Continental 1 Mitarbeitergehalt oder Schätzwert	Über 73 Tsd € - 77 Tsd €	73 Tsd €	77 Tsd €
	<b>Requirements Engineer</b> msg systems (Germany) 1 Mitarbeitergehalt oder Schätzwert	Über 48 Tsd € - 52 Tsd €	48 Tsd €	52 Tsd €
	<b>Requirements Engineer</b> Fraport 1 Mitarbeitergehalt oder Schätzwert	Über 42 Tsd € - 45 Tsd €	42 Tsd €	45 Tsd €
	<b>Requirements Engineer</b> Baker Hughes 1 Mitarbeitergehalt oder Schätzwert	Über 62 Tsd € - 68 Tsd €	62 Tsd €	68 Tsd €
	<b>Requirements Engineer</b> peiker acoustic 1 Mitarbeitergehalt oder Schätzwert	Über 50 Tsd € - 54 Tsd €	50 Tsd €	54 Tsd €





# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



# Analyse von Assoziationen (Klassenbeziehungen)

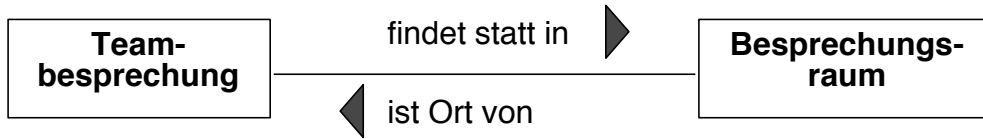
- ▶ Assoziationen entstehen, wenn *Objekte sich kennen*
  - Sie stellen Relationen, Graphen oder Hypergraphen dar, d.h. bilden Abstraktionen von Referenzen
- ▶ **Def.:** Eine (binäre) **Assoziation (Beziehung, relationship) AS** zwischen zwei Klassen  $K1$  und  $K2$  beschreibt, daß die Instanzen der beiden Klassen in einer Beziehung zueinander stehen.
- ▶ **Semantik:** Für jedes Objekt  $O1$  der Klasse  $K1$  gibt es eine individuelle, veränderbare und endliche Menge  $AS$  von Objekten der Klasse  $K2$ , mit dem die Assoziation  $AS$  besteht.  
Analoges gilt für Objekte von  $K2$ .
- ▶ Mathematisch ist dies eine Relation zwischen dem Extent von  $K1$  und dem Extent von  $K2$



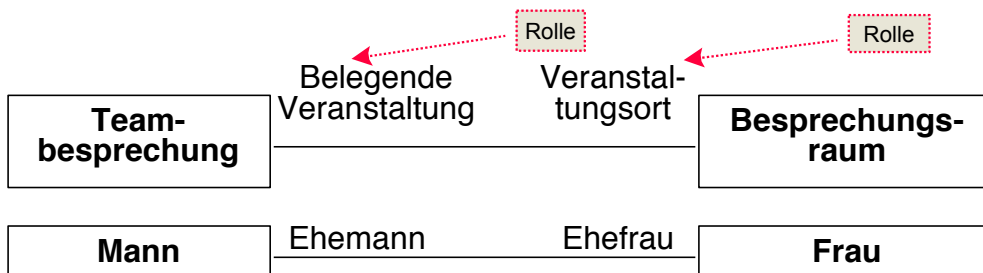
*Welche Assoziationen zwischen Objekten enthält der Text des Kunden?*

# Leserichtung und Assoziationsenden

- ▶ Für Assoziationsnamen kann die **Leserichtung** angegeben werden.
  - Sie wird aus dem Interviewtext entnommen und dazu genutzt, den Text aus dem Modell zu rekonstruieren.

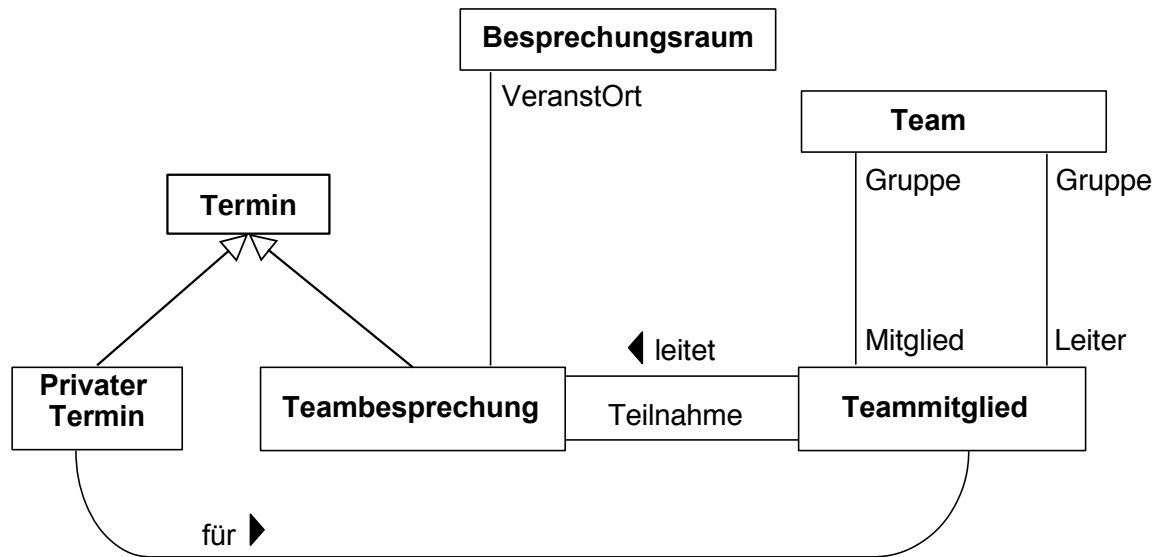


- ▶ Ein Name für ein **Assoziationsende** bezeichnet die Assoziation aus der Sicht einer der teilnehmenden Klassen.
  - Ein Assoziationsende beschreibt die **Rolle** einer Klasse in einer Assoziation



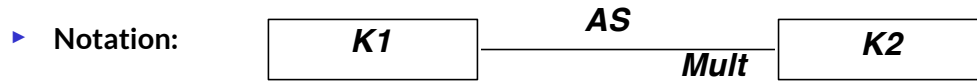
# Beispiel: Das Lesen Assoziationen im Kundentext und im Analysediagramm

- ▶ Die Assoziationen eines Analysediagramms können “gelesen” werden:
  - Ein Team besitzt als Mitglieder die Teammitglieder, als Leiter ein Teammitglied. Teammitglieder gehören zur Gruppe des Teams.
- ▶ Umgekehrt können “user stories” in Analysediagramme umgesetzt werden



# Multiplizität bei Assoziationen

- ▶ **Definition** Die **Multiplizität (Kardinalität)** einer Klasse  $K1$  in einer Assoziation  $AS$  mit einer Klasse  $K2$  begrenzt die Anzahl der Objekte der Klasse  $K2$ , mit denen ein Objekt von  $K1$  in der Assoziation  $AS$  stehen darf. (**Weite** der Relation)

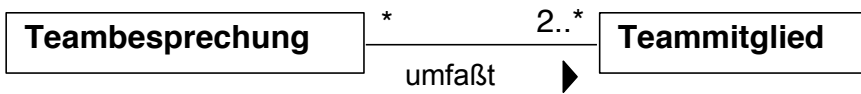


Multiplizität *Mult* :

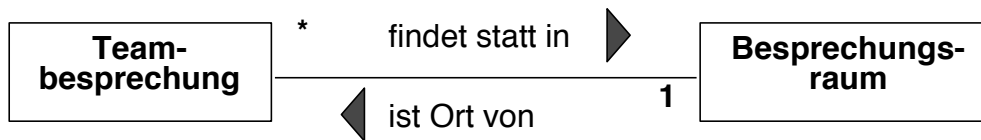
- $n$  (genau  $n$  Objekte der Klasse  $K2$ )
- $n..m$  ( $n$  bis  $m$  Objekte der Klasse  $K2$ )
- $n1, n2$  ( $n1$  oder  $n2$  Objekte der Klasse  $K2$ )

Zulässig für  $n$  und  $m$  :

- Zahlenwerte (auch 0)
- \*
- (d.h. beliebiger Wert, einschließlich 0)



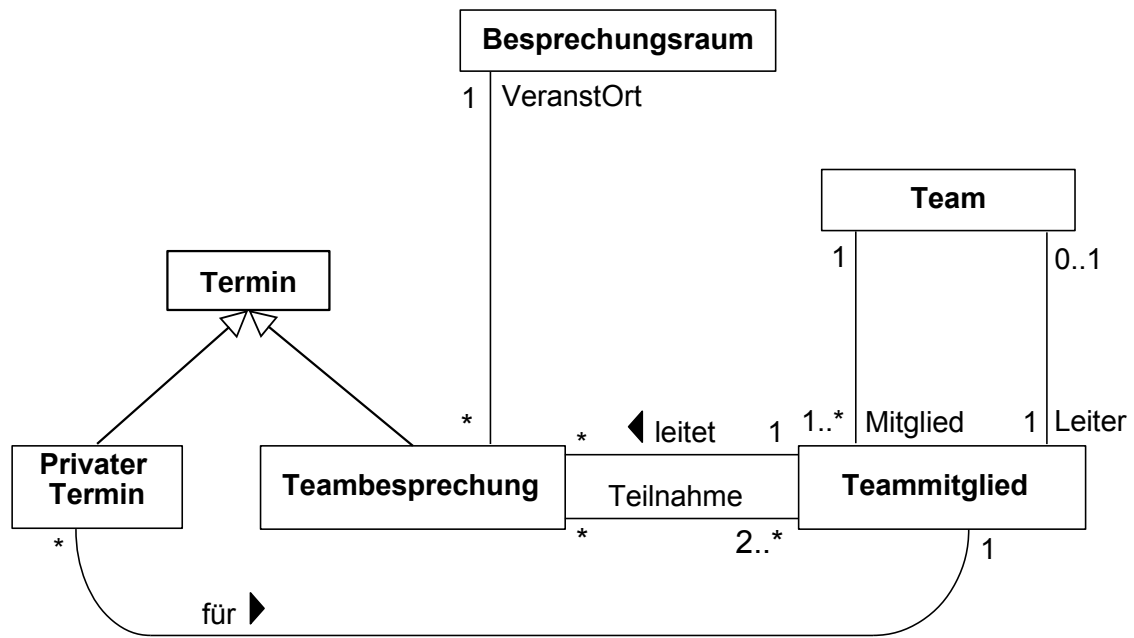
## Multiplizitäten bestimmen durch "Vorlesen"



- ▶ Von links nach rechts:
  - "Jede Teambesprechung findet statt in (wie vielen?) Besprechungsräumen."
  - "Jede Teambesprechung findet statt in genau 1 Besprechungsraum."
- ▶ Von rechts nach links:
  - "Jeder Besprechungsraum ist Ort von (wie vielen?) Teambesprechungen."
  - Jeder Besprechungsraum ist Ort von 0 oder mehreren Teambesprechungen."

Die Multiplizitätsbeschränkung steht an der Klasse, für die die Anzahl der Teilnehmer an der Assoziation beschränkt werden.

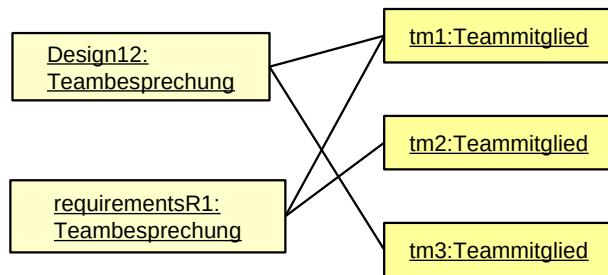
# Beispiel: Erweiterung eines Analysediagramms durch Multiplizitäten



# Semantik (bidirektionaler) Assoziationen

- ▶ Ein **Extent (Inhalt)** einer Assoziation (auch *Relation, Graph*) ist ähnlich einer *Tabelle*:

Teilnahme-Assoziation	
Teambesprechung	Teammitglied
design12	tm1
design12	tm3
requirementsR1	tm1
requirementsR1	tm2



- ▶ Der Extent kann mit einer Graph-Bibliothek wie jgraphT realisiert werden
- ▶ Von einem beteiligten Objekt aus betrachtet, gibt eine Assoziation eine *Menge* von assoziierten Objekten an (**Nachbarmenge**):

Objekt design12: Teambesprechung

Teammitglied-Objekte in Teilnahme-Assoziation: {tm1, tm3}

Objekt tm1: Teammitglied

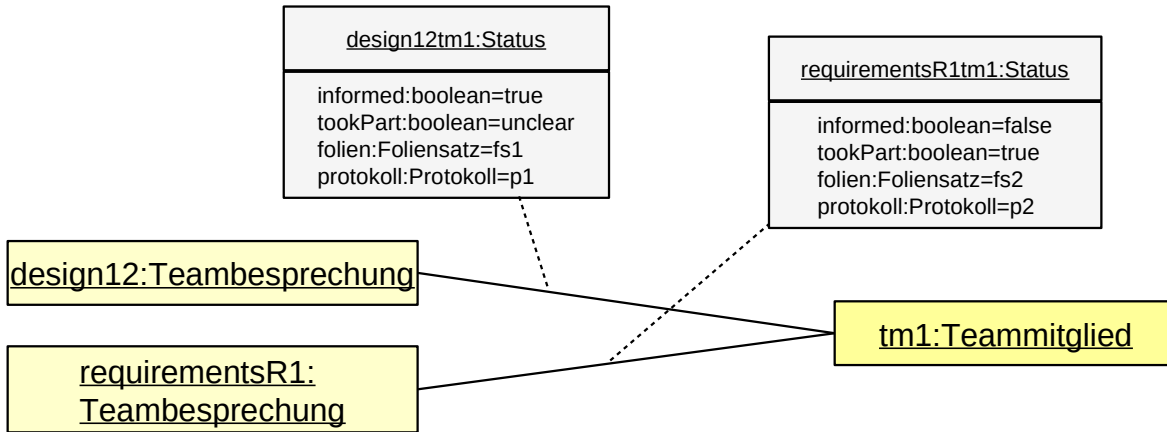
Teambesprechung-Objekte in Teilnahme-Assoziation: {design12, requirementsR1}





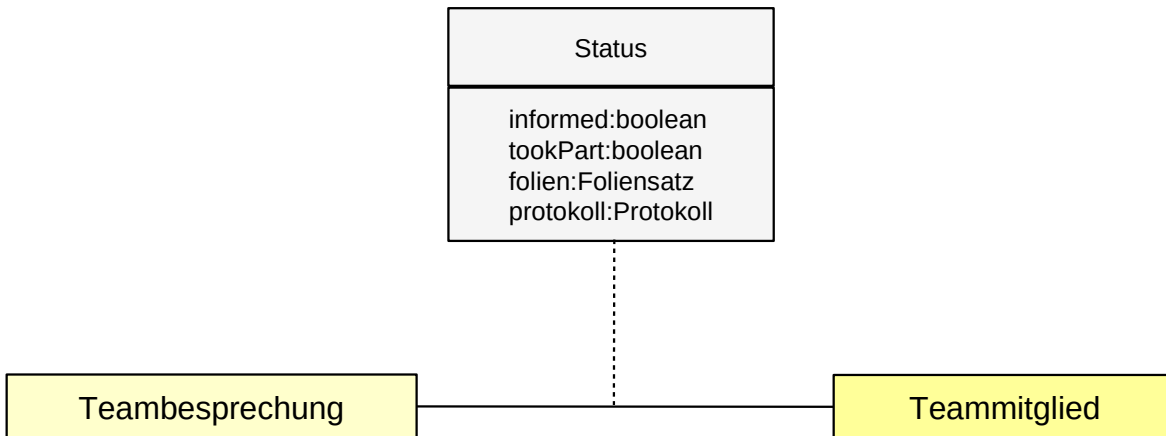
# Assoziationsattribute

- ▶ Oft tragen Assoziationen (Tupel der Relationen, Kanten des Graphen) *Kantenattribute*
  - Diese werden durch *Kantenobjekte* modelliert:

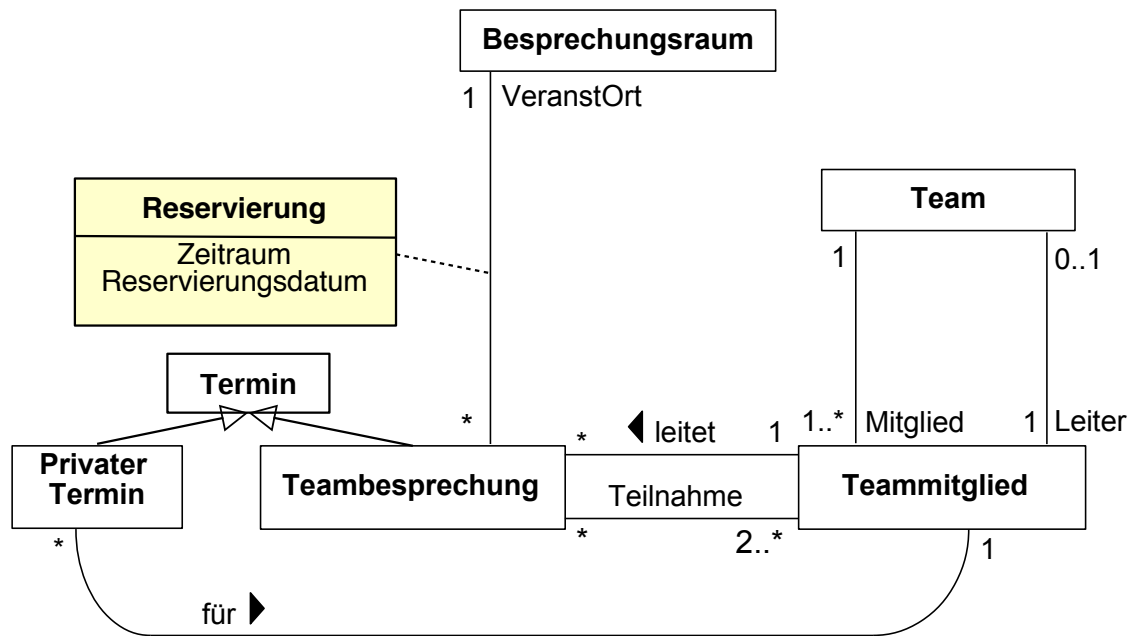


# Assoziationsklassen (Kantenklassen)

- ▶ **Def.:** Eine **Assoziationsklasse (Kantenklasse)** beschreibt die Kantenobjekte einer Assoziation.
- ▶ Assoziationsklassen werden benötigt, wenn Wissen dargestellt werden soll, das für jede Kante (Tupel) *unterschiedlich* ist, d.h. Wissen über die Assoziation der Objekte darstellt
- ▶ In jgraphT entsprechen Assoziationsklassen einer Unterklasse von DefaultEdge



# Beispiel: Erweiterung des Analysediagramms mit einer Reservierung als Assoziationsklasse





## 31.3.2 Realisierung von Assoziationen

- ▶ Assoziationen beschreiben Objektnetze beliebig großer Art.
- ▶ Wir nutzen jetzt das in Abschnitt II Gelernte, um die Abbildung von UML-Assoziationen auf Java zu verstehen



- ▶ 1) Realisierung durch **Graphen** einer Graph-Bibliothek (siehe Kap. 23)
- ▶ 2) Realisierung durch **zwei gerichtete Assoziationen** (siehe Kap. 21)
  - Redundanz: zusätzlicher Speicheraufwand, Gefahr von Inkonsistenzen
  - Aber: schnelle Navigation
- ▶ 3) Realisierung nur in **einer Richtung**: (Anhang, siehe Kap. 21)
  - Gibt nicht die volle Semantik des Modells wieder
  - Abhängig von Benutzung (Navigation) der Assoziation
- ▶ 4) Realisierung durch **Assoziationsklassen** bzw. Zwischenklassen, die Assoziationen repräsentieren: (Anhang)
  - Geeignete Datenstrukturen erforderlich
    - "Beidseitige" Abfragemöglichkeit
    - Abflachung zu normalen Klassen nötig
- ▶ 5) Realisierung durch **Konnektoren** (siehe Kapitel Konnektoren)
- ▶ 6) Realisierung durch **explizite Rollenklassen** (Anhang)
- ▶ 6) Realisierung durch **Tabellen in einer relationalen Datenbank**
- ▶
- ▶ **Genauere Entscheidung erst im Entwurf !**

# Realisierung von Assoziationen

Höhere  
Sprachen

Graphen als Sprachkonstrukte

Graphen als Konnektoren (Rollen-Kollaborationen)

Fixe Netze als Entwurfsmuster

Java

Fixe Netze mit Datenfluss mit Konnektoren

Graphen als Bibliotheken (Java)

Graphen als Collections abgeflacht (Java)

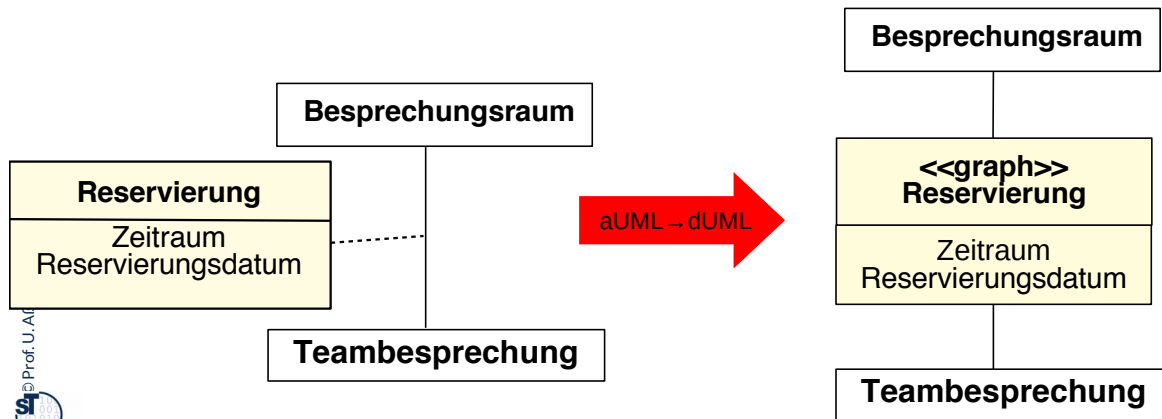
Graphen als Endoassoziationen:  
Netze von Unterobjekten in komplexen Objekten (Java)

System-  
programmier-  
sprache (C)

Graphen als Datenstrukturen fester Länge abgeflacht

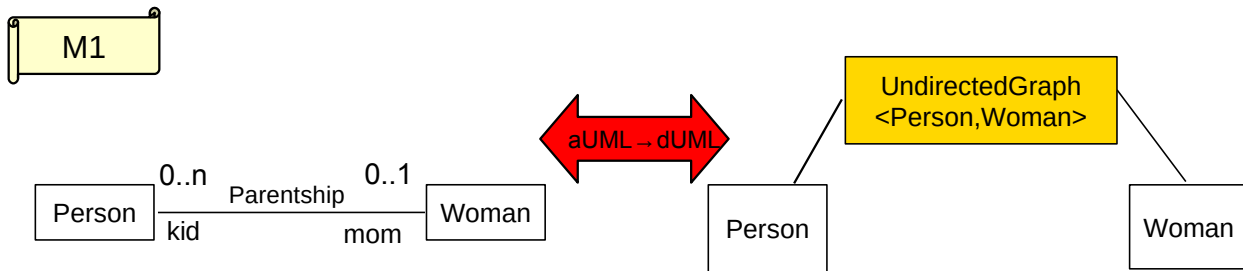
## 31.3.2.1 Realisierung von bidirektionalen Assoziationen durch Graphklassen

- ▶ Assoziationsklassen aus dem Analysemodell können im Implementierungsmodell in Graphklassen abgeflacht werden (Kap. 23):

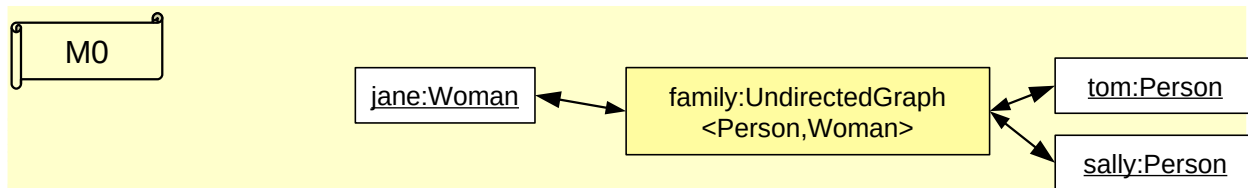


# Beispiel: Realisierung von bidirektionalen Assoziationen durch Graphklassen

- ▶ Auch bidirektionale Assoziationen können durch Graphklassen realisiert werden (Kap. 23)



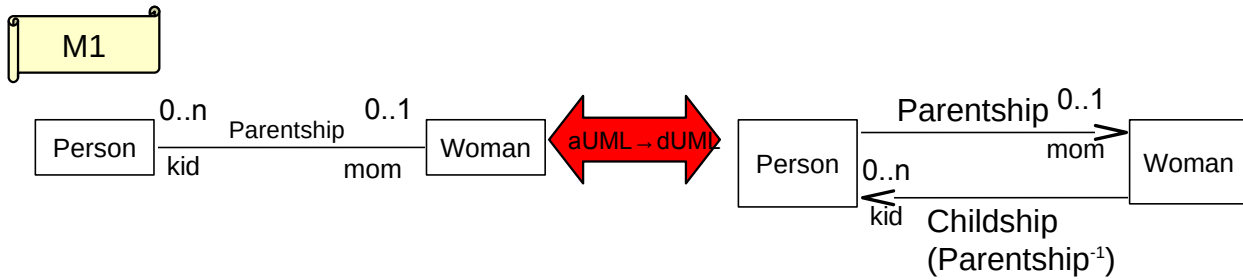
Laufzeit: zwei Referenzen zwischen Graphobjekt und den assoziierten Objekten



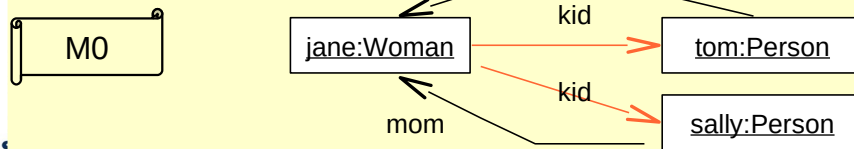


## 31.3.2 Realisierung von bidirektionalen Assoziationen durch unidirektionale

- ▶ Bidirektionale Assoziationen können, falls sie keine Attribute tragen, in gerichtete Assoziationen umgewandelt werden (Kap. 21-collections)
- ▶ Realisierung im Implementierungsmodell in jUML durch Einführung von *gerichteten* Assoziationen



Laufzeit:



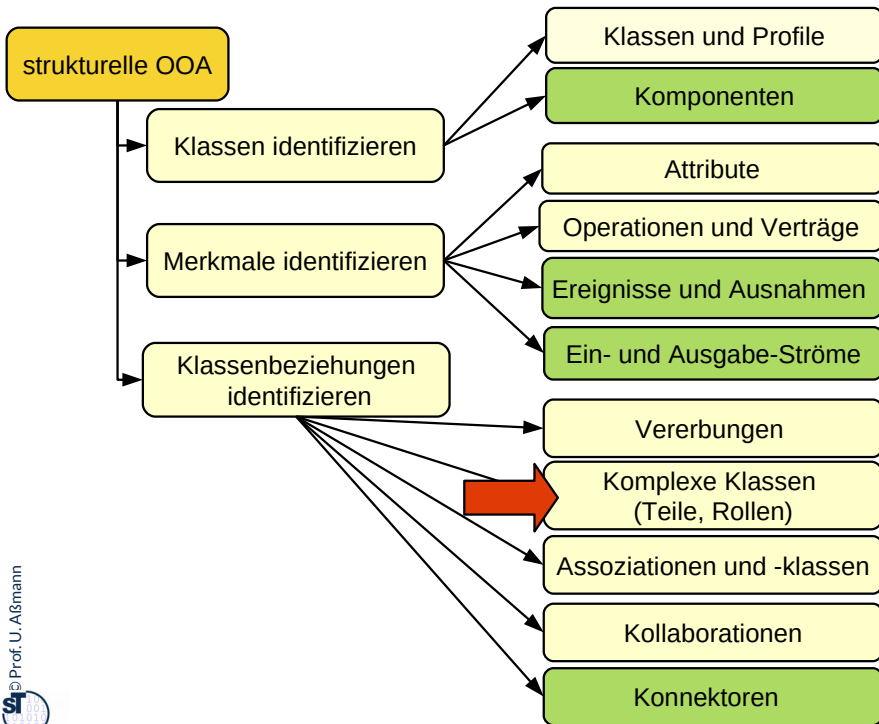


## 31.4 Modellierung von Hierarchien und komplexen Objekten (Verbundobjekten, Teilehierarchien)



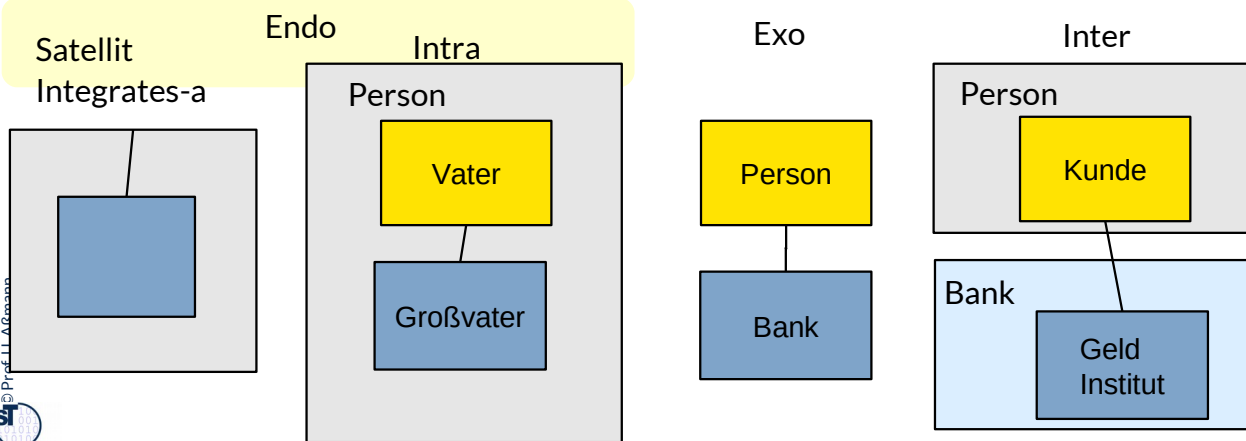
# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



# Endo- vs. Exo-Assoziation

- ▶ Komplexe Objekte (Verbundobjekte) verwenden Assoziationen inner- und außerhalb ihrer selbst.
- ▶ Frage: "Gehören die beiden Objekte zu einem Ganzen"? (logische Einheit, physische Getrentheit)
- ▶ **Endo-Assoziation:** Assoziation innerhalb eines komplexen Objektes
  - **Intra-Assoziation:** beide werden von einem dritten umschlossen
  - **Satellit-Assoziation (integrates-a):** einer der Partner umschließt den anderen.
    - "Teil" ist eine Satellit-Endo-Assoziation vom Ganzen zum Teil
- ▶ **Exo-Assoziation:** keiner der Partner umschließt den anderen.
  - **Inter-Assoziation:** jeder der Partner wird von einem anderen umschlossen



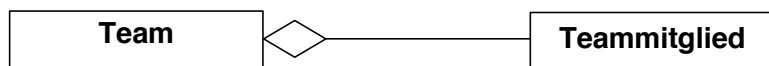


## 31.4.1 Aggregation, Komposition, Rollenspiel als Endo-Assoziationen



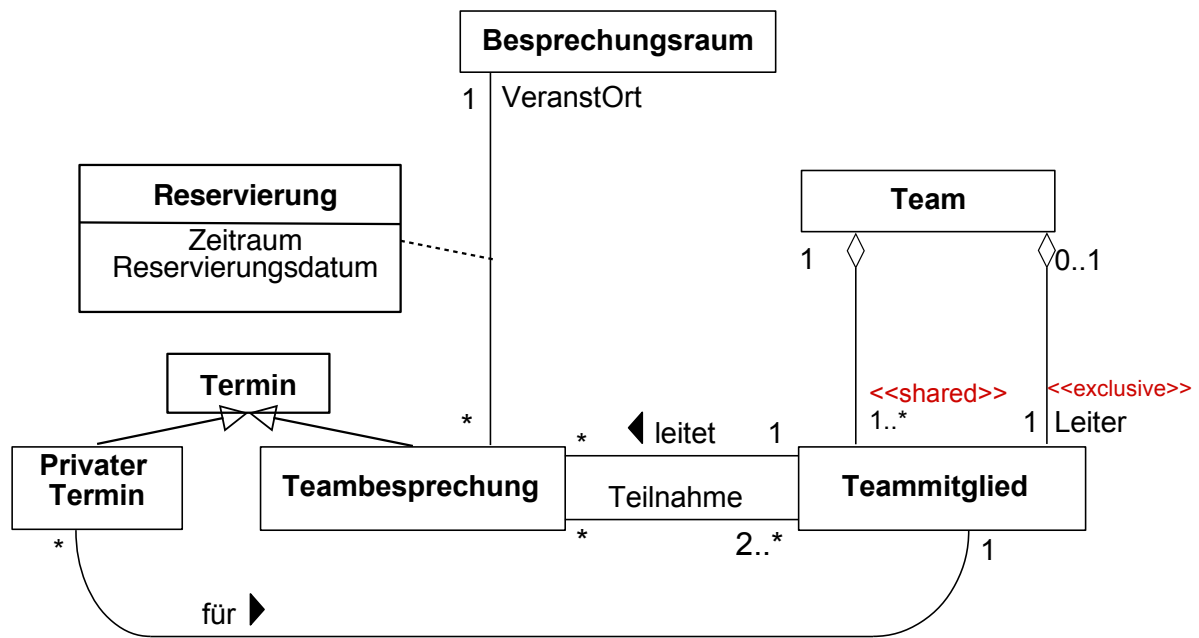
## Wdh: Aggregation (has-a)

- ▶ **Definition:** Wenn eine Assoziation den Namen „hat-ein“ oder "besteht-aus" tragen könnte, handelt es sich um eine **Aggregation** (Ganzes/Teile-Relation).
  - Eine Aggregation besteht zwischen einem *Aggregat*, dem *Ganzen*, und seinen *Teilen* (**Endo-Assoziation**)
  - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklensfreien Graphen, *dag*)
  - Ein Teil kann
    - zu mehreren Ganzen gehören (*<<shared>>*),
    - zu einem Ganzen (*<<owns-a>>*) und
    - exklusiv zu einem Ganzen (*<<exclusively-owns-a>>*)



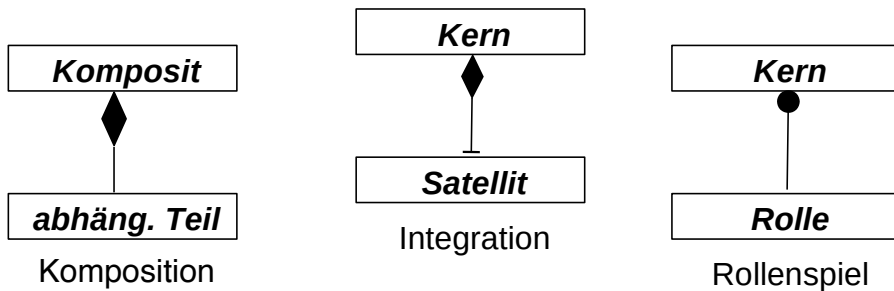
Lies: „Team hat ein (wechselbares) Teammitglied“

# Beispiel: Erweiterung eines Analysediagramms mit Aggregationen



# Komposition (owns-a)

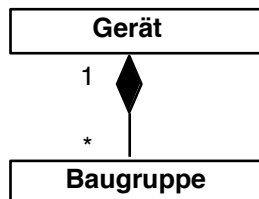
- ▶ **Definition:** Ein Spezialfall der Aggregation ist die **Komposition** zwischen einem **Komposit** und seinen **Teilen**.
  - Ein Objekt kann Teil höchstens eines Komposits sein (Eigentums-Relation *exclusively-owns-a*).
  - Das Teil ist *abhängig* vom Komposit (*dependent part*).
    - Das Komposit hat die alleinige Verantwortung für Erzeugung und Löschung seiner Teile (gleiche Lebenszeit)
- ▶ Def: Aggregation, Komposition und Rollenspiel sind Spezialfälle der **Integration** von Unterobjekten als Satellit eines Kernobjekts (*integrates-a*)



Die Multiplizität auf Kompositseite kann also höchstens eins sein!

Wenn die Komposition von Komponente zu Komposit die Multiplizität 0 zulässt (d.h. 0..1), kann eine Komponente abgetrennt werden und z.B. in ein anderes Komposit eingegliedert werden!

Komposition ist meist einfach zu erkennen. Attribute stehen in UML in Kompositionsbeziehung zu ihrem Objekt !





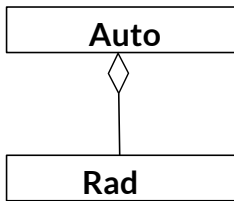
# Beispiele von Endo-Assoziationen

„Welches wechselbare Teil hat das Objekt?“

„Welches dauerhafte Teil hat das Objekt?“

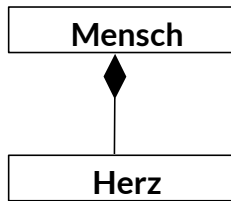
„Welche Rolle spielt das Objekt?“

## Aggregation



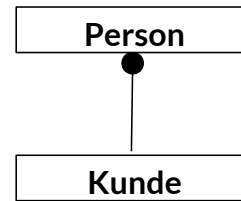
„Ein Auto hat ein Rad“

## Komposition

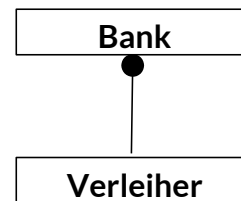
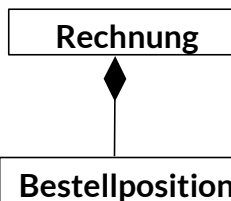
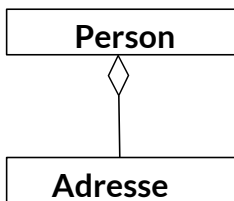


„Ein Mensch braucht ein Herz“

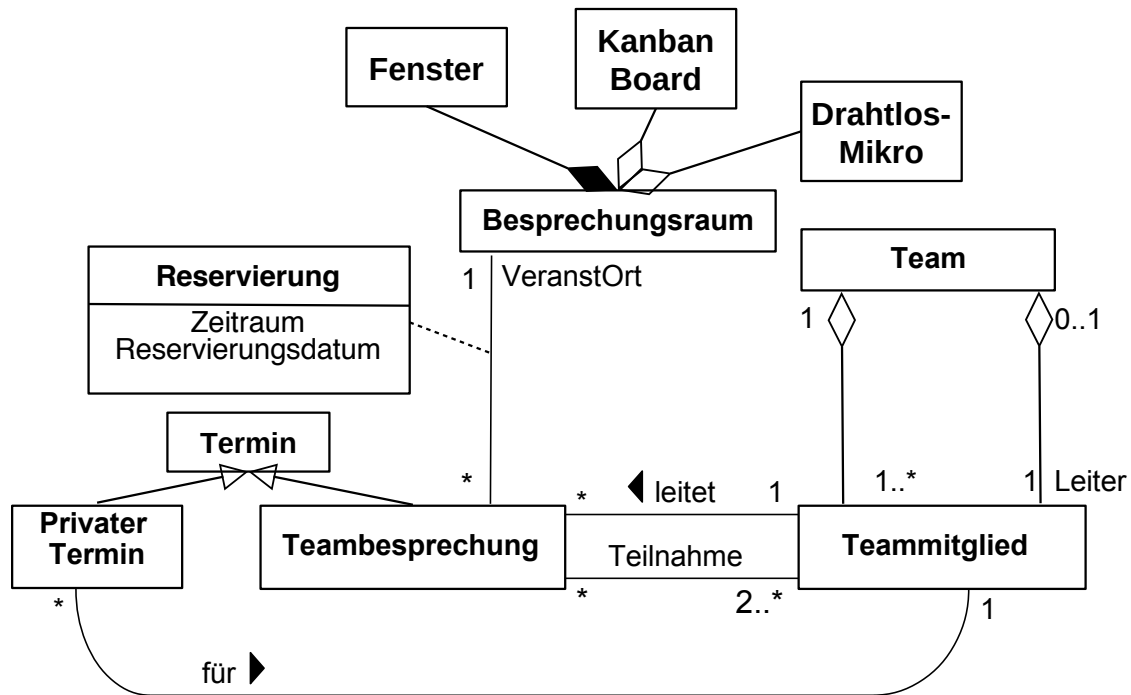
## Rollenspiel



„Eine Person spielt einen Kunden“



# Beispiel: Erweiterung des Analysediagramms mit Kompositionen und Aggregationen



# Darstellung von Hierarchien und kompositen Objekten

- ▶ Hierarchien und komposite Objekte (whole-part hierarchies) mit ihren Endo-Assoziationen können mit visuellen Darstellungen für Bäume dargestellt werden:
  - Baum
  - Venn-Diagramm
  - Zeilenhierarchie (Tree-Widget)
  - Mind map
- ▶ Alle Darstellungen sind äquivalent und können ineinander umgewandelt werden

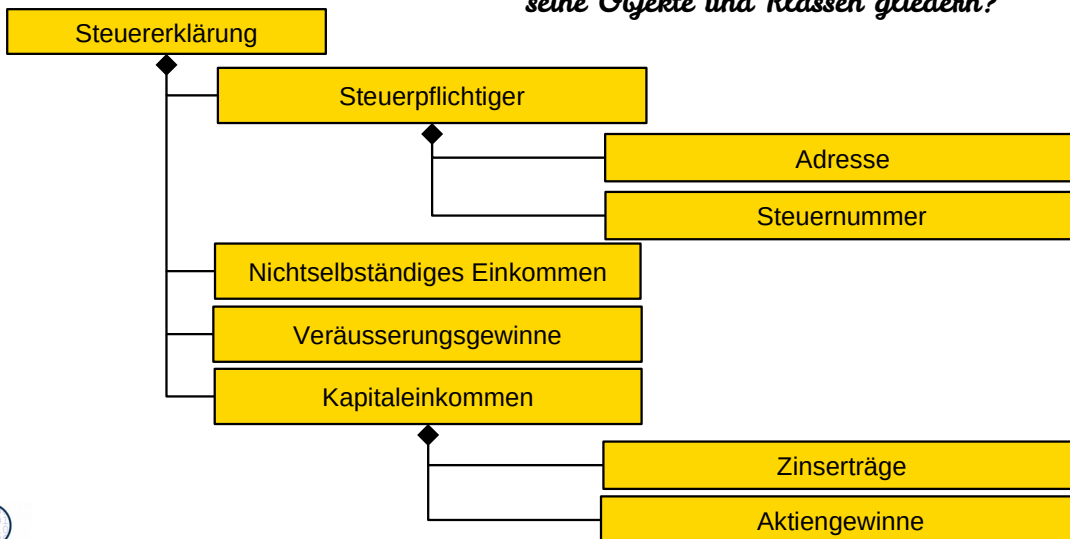
Als Baumrelation können verschiedene Relationen dienen:  
Vererbung, Teil, Aufruf, Operatoren, Unterobjekte (s. später)

## Ganzes/Teile-Zeilenhierarchie (whole-part hierarchies)

71 Softwaretechnologie (ST)

- ▶ Komplexe Objekte bestehen oft aus Ganz-Teile-Hierarchien (owns-a-Hierarchien)
- ▶ Man kann Komplexe Objekte als auch als **Aggregations-** oder **Kompositionshierarchien** anzeigen
  - Ganzes/Teile-Hierarchien können mit Klassen oder auch Objekten gezeichnet werden

*„In welche hierarchische Struktur will der Kunde seine Objekte und Klassen gliedern?“*

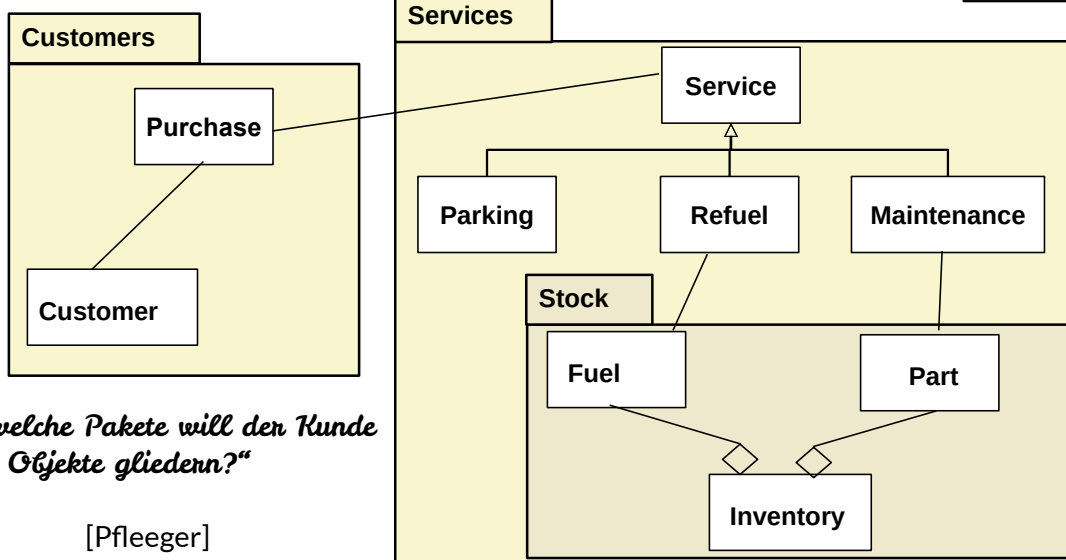
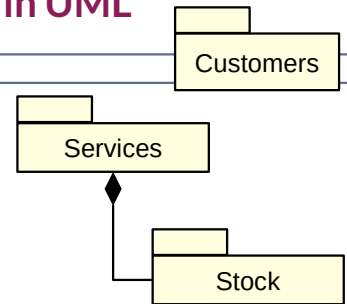


Viele unserer Beispiele der Vorlesung sind Ganz-Teile-Hierarchien

- Siehe Beispiele Bestellung, Rechnung, Formular, Steuererklärung

# Bsp: Pakete und ihre Aggregationsrelationen in UML

- ▶ Ein **UML-Paket** fasst eine Menge von UML-Diagrammfragmenten zusammen.
- ▶ Klassen sind *flach*, Pakete dagegen *hierarchisch* strukturierbar
  - Pakete gruppieren Klassen, Objekte, andere UML-Fragmente
- ▶ UML-Paketrelationen sind hierarchisch oder *azyklisch*
  - Sind einfach auf Paketkonzepte von Programmiersprachen abbildbar

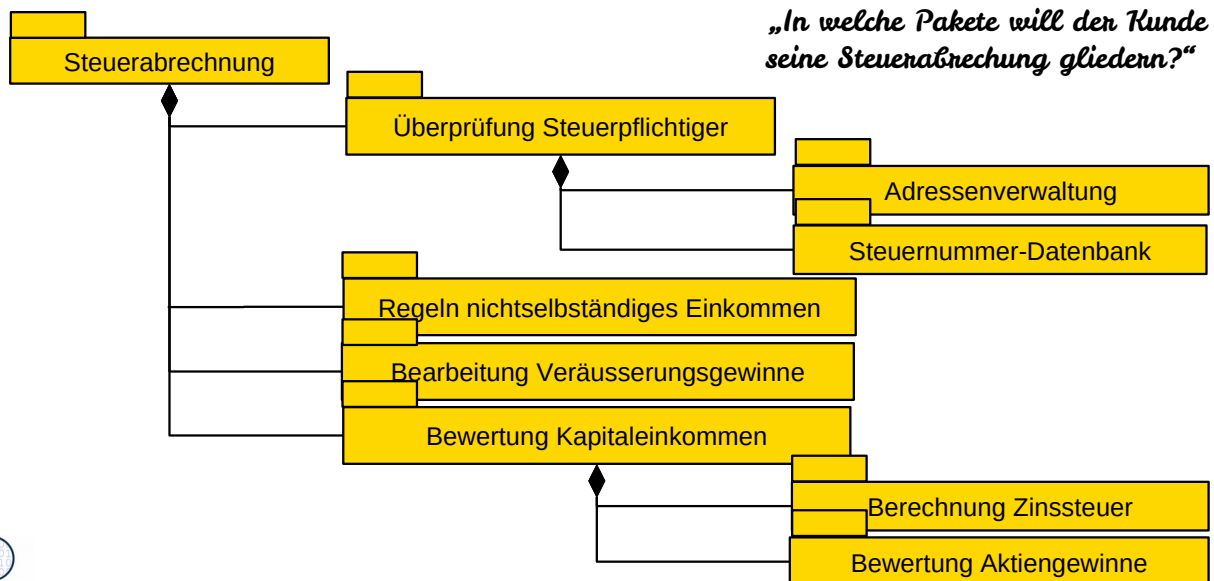


„In welche Pakete will der Kunde seine Objekte gliedern?“

[Pfleeger]

# Paket-Zeilenhierarchie

- ▶ Auch Paket-Hierarchien kann man als auch als Zeilenhierarchien (Textbäume) anordnen
- ▶ Operator ist die Paketvereinigung
- ▶ Pakete sind Container von UML-Fragmenten (z.B. von Klassen)



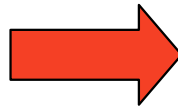
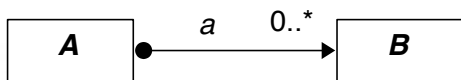
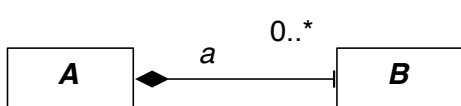
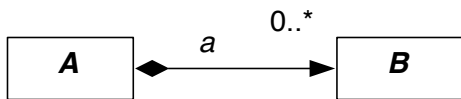
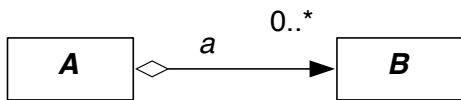
## 31.4.2 Realisierung von komplexen Analyseobjekten mit ihren Endo-Assoziationen

- ▶ Siehe auch Kap. "Collections" und "Graphen"



# Realisierung von Aggregation, Komposition und Rollenspiel in Java

- ▶ Aggregationen stellen azyklische Graphen dar und können daher genau wie Assoziationen abgebildet werden
  - Komposition und Rollen werden in gleicher Weise abgebildet
- ▶ Daher gehen in Java die Analyseinformationen verloren!
- ▶ Überlege auch den Einsatz der Entwurfsmuster Decorator (für Listen) und Composite (für Bäume)



```
class A {  
    ...  
    Collection<B> a;  
    // B[] a;  
    ...  
}
```





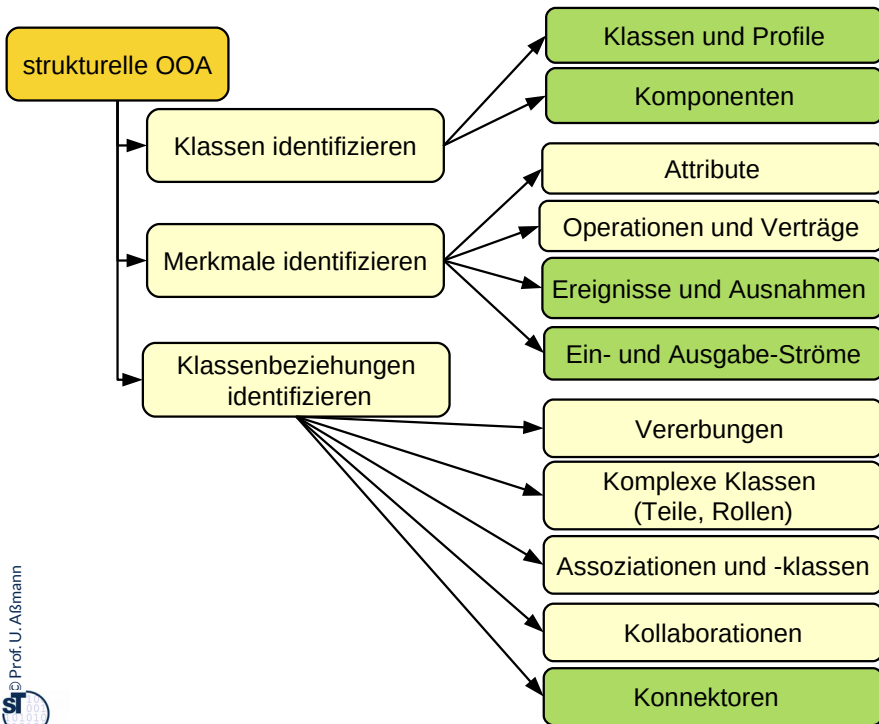
### 31.4.3 Modellierung von komplexen Analyseobjekten aus Kernen und Unterobjekten

- ▶ Große Objekte bestehen aus Kernen und Unterobjekten, die mit Endoassoziationen verbunden sind



# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

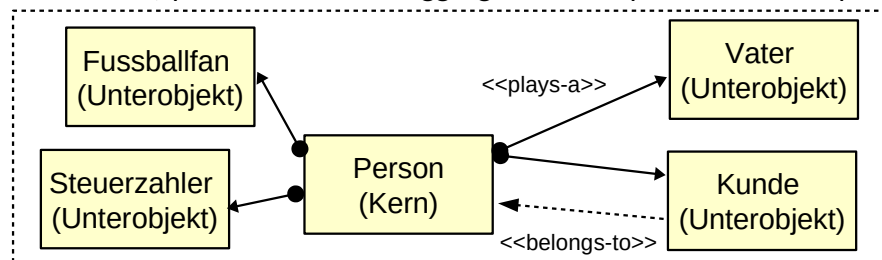
► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



**Def.: Ein komplexes Objekt (Subjekt, Großobjekt, complex object, big object)** ist ein Objekt, das auf Programmierniveau wegen seiner Komplexität durch *ein Kernobjekt und mehrere Unterobjekte (Teilobjekte)* dargestellt wird.

Seine innere Struktur ist meist hierarchisch, immer aber azyklisch angelegt.

- ▶ Ein **Unterobjekt (Satellit)** ist ein Objekt, das an ein **Kernobjekt** angelagert ist und mit ihm ein integriertes **komplexes Objekt** bildet
  - Das Unterobjekt hat also keine eigene Identität, sondern teilt seine Identität mit dem Kernobjekt (logische Einheit)
  - Es repräsentiert eine Eigenschaft bzw. Teilzustand des komplexen Objekts
  - Als **Integrationsrelation** `<<integrates>>` wird eine Endo-Assoziation verwendet (spezielle Fälle sind Aggregation, Komposition, Rollenspiel)



Schon in der Analyse werden komplexe Objekte aus einem *Kern* und einer Menge von zugehörigen *Unterbjekten* (Teilobjekte, Satelliten) dargestellt

- ▶ **Satz:** Die Endoassoziation eines komplexen Objekt ist immer *hierarchisch* oder *azyklisch*, bildet also immer eine Hierarchie oder einen gerichteten azyklischen Graphen (dag):
  - **Flache komplexe Objekte** als Kern mit **Unterbjekten**, die in den Kern mit integrates-a integriert sind
  - **Hierarchische komplexe Objekte** in **Baum-Notation** in einem **hierarchischen** komplexen Objekt (Kernobjekt als Wurzel)
  - **Geschichtete komplexe Objekte** in **Dag-Notation** in einem **geschichteten** komplexen Objekt (Kernobjekt als Wurzel eines gerichteten azyklischen Graphen, directed acyclic graph, dag)

- ▶ Welche Endoassoziation trägt das Entwurfsmuster “Bridge”?



# Arten von komplexen Objekten (Analyse- und Entwurfsobjekten)

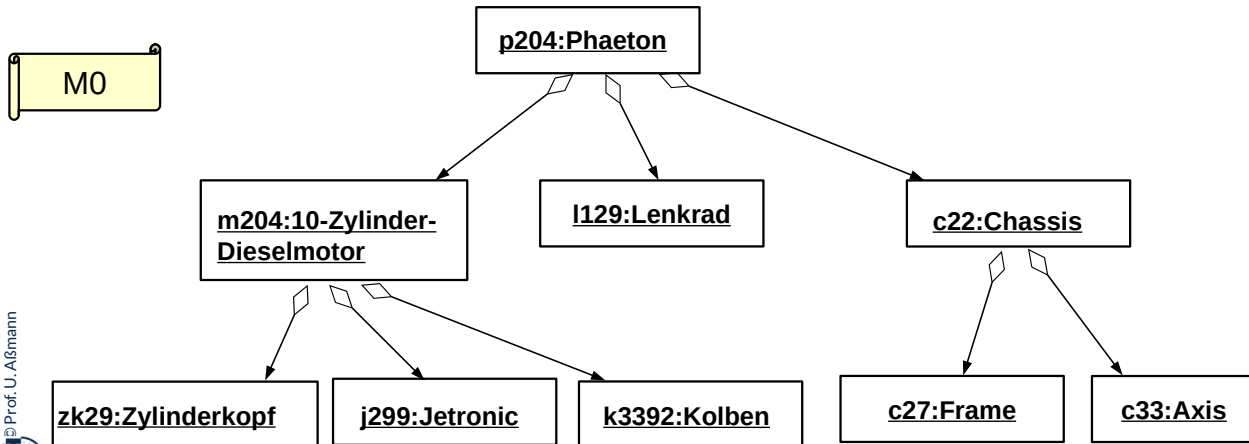
- ▶ Komplexe Objekte kommen in allen Phasen vor
  - Ihr **Lebenszyklus** wird durch eine Zustandsmaschine beschrieben (Kap 33)
  - Interpretierer-Funktion: Sie empfängt Befehle wie Create(), Open(), Read(), Write(), Do(), Enter()
  - Steuerungsfunktion: Sie sendet weitere Befehle aus (Steuerungsmaschine)
- ▶ **Analyseobjekte** stammen oft von **Domänenobjekten** her
  - **Simuliertes Objekt**: kommt in der Umgebung des Programms vor (Simulation)
    - Kunde, Bestellung, Huhn, Auto, Radar, ...
  - **Geschäftsobjekt**: abstraktes Material-Objekt der Anwendung
    - Rechnung, Termin, Bestellung, Bestellposition
- ▶ **Entwurfsobjekte** modellieren zusätzlich technische Eigenschaften:
  - **Schnittstellenobjekte** (im Kontextmodell): Widget, Stream, File
  - **Technische Objekte**: Objekte des Systems, von dem der Kunde nichts sieht: Komponente, Treiber, Datenbank, ...
- ▶ **Implementierungsobjekte** (einfache, physikalischen Objekte) sind dagegen einfach, flach und passen direkt auf die Maschine
  - Sie entsprechen Verbunden (records). Sie tragen keine Analyseinformation mehr, sind nackt

- ▶ Viele Daten, die in Anwendungen gehandhabt werden, sind **komposite Objekte (Materialien)**
  - Materialien gehören zur Datenhaltungsschicht
  - Materialien sind oft komplex
- ▶ Beispiele:
  - **Informationssysteme** verwalten komplexe Informationen (“EDV”)
    - Buchungen, Daten
  - **Produktionsplanungssysteme** verwalten
    - *Produkte*. Die Teile eines Produkts werden in Stücklisten (eigentlich Stückbäume) verwaltet
    - *Fabriken*. Die Teile und Maschinen einer Fabrik werden modelliert
  - **Geschäftsprozesssoftware** verwaltet *Dokumente von Geschäftsvorgängen* (Bestellungen, Rechnungen, Löhne, Mitarbeiter...), die ebenfalls komposite Objekte darstellen
    - *Geschäftsobjekte (business objects)*, Objekte des Domänenmodells, sind oft komposit

*“Welche Stücklisten kommen mit welchen Teilen in dem Text des Kunden vor?”*

# Beispiel: Komposite Material-Objekte als Stücklisten

- ▶ Eine der meistgebrauchten Datenstrukturen sind hierarchische *Stücklisten* (*part lists*)
  - *Bestellung, Formular, Rechnung* sind alles Stücklisten
- ▶ Produktionsplanungssysteme (PPS) verwalten Produkte als Materialien
  - Die Teile eines Produkts, Artikels, Materials werden in *Stücklisten* (eigentlich *Stückbäume*) verwaltet
- ▶ Stückliste eines Phaeton (alle Teile sind lebenslang nummeriert, um verfolgbar zu sein)





Auf dem Weg vom Analysemodell über das Entwurfsmodell zum Implementierungsmodell und Code werden wir folgende Arten von Verfeinerungsmethoden kennenlernen:

- 1) **Punktweise Verfeinerung von Lebenszyklen von komplexen Objekten:** Abbilden von Lebenszyklen auf niederere Schichten
  - Interpreterverfeinerung (Automatverfeinerung, Verfeinerung von abstrakten Maschinen) (Kapitel OOD.34)
  - Tool-Verfeinerung
  - Material-Verfeinerung
- 2) **Querschneidende Objektorreicherung (object fattening) von komplexen Objekten:** Verfeinerung von Objekten aus dem Domänenmodell durch Integration von Unterobjekten
  - Jedes Unterobjekt stellt eine neue Eigenschaft des Domänenobjektes dar
  - Szenarioanalyse (Kapitel OOA.35)



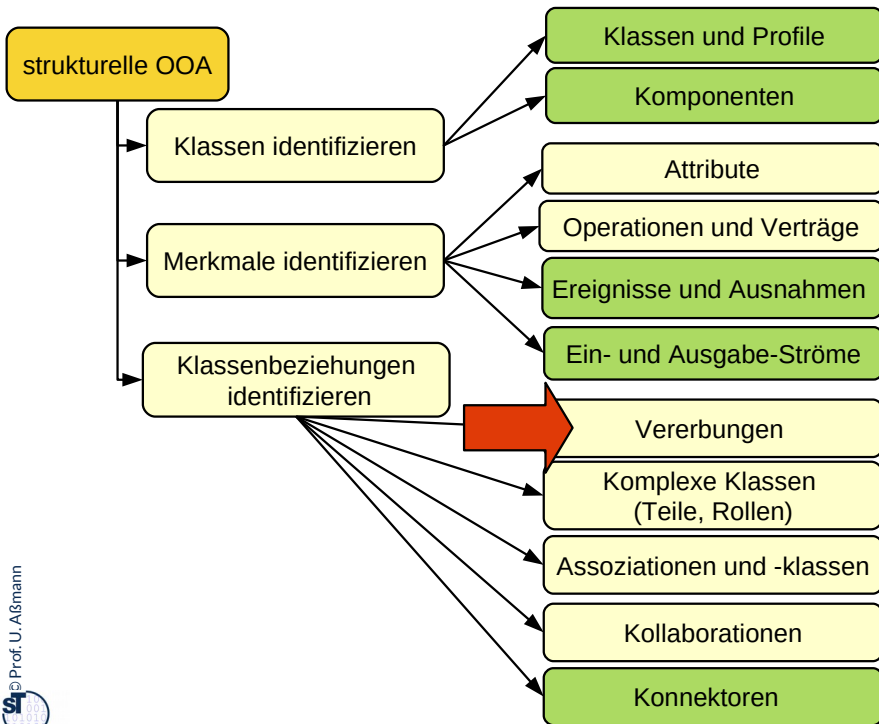
## 31.5 Mehrfachvererbung zwischen Klassen

- ▶ In Analyse und Grobentwurf spielt mehrfache Vererbung eine große Rolle

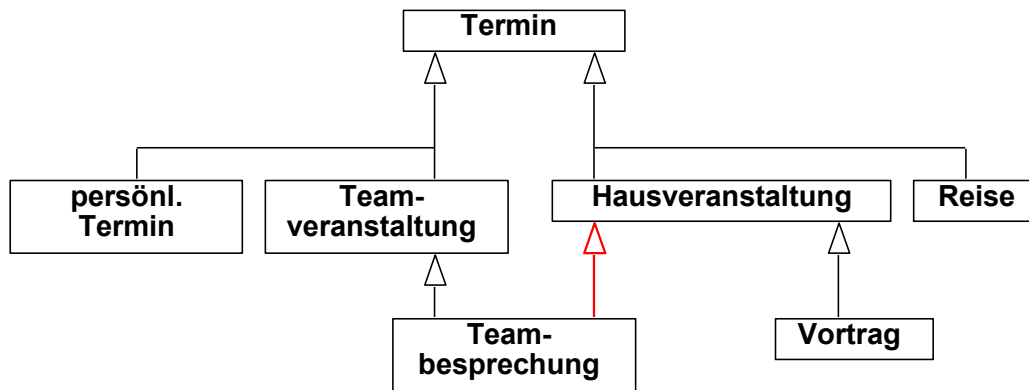


# Q6: Schritte der strukturellen, metamodelgetriebenen Analyse

► gelb: Domänenmodell; grün: Kontextmodell, TopLevel-Architektur



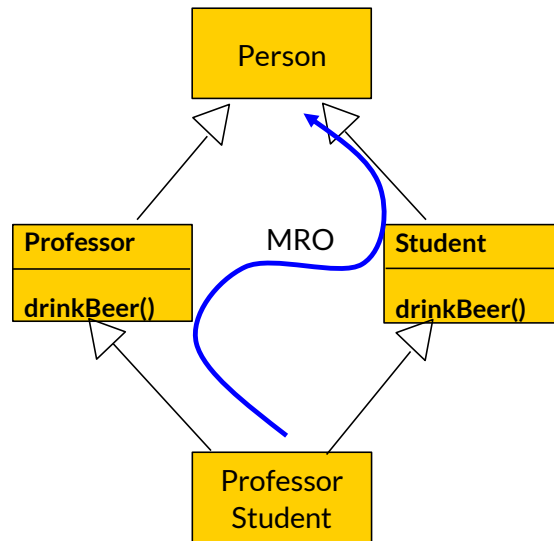
- ▶ "Eine Teambesprechung ist eine Teamveranstaltung, die ein Termin ist."
- ▶ "Eine Teambesprechung ist eine Hausveranstaltung. Die Hausveranstaltung ist eine Art von Termin."
- ▶ In UML ist es prinzipiell möglich, daß eine konkrete Klasse von mehreren Klassen erbt:



UML besitzt die Möglichkeit, für eine Klasse mehrere Oberklassen zu spezifizieren. Das ist insbesondere in der OOA wichtig.

# Mehrfachvererbung (Multiple Inheritance)

- ▶ **Mehrfachvererbung**
  - Eine Klasse kann mehrere Oberklassen besitzen
  - Dadurch wird die Vererbungsrelation ein gerichteter azyklischer Graph (dag)
- ▶ Eine Klasse kann ein Merkmal mehrmals erben
  - Daher muss die Merkmalsuche einer Strategie gehorchen, der Merkmalsauflösungsordnung (method-resolution-order, MRO), einer Navigationsstrategie, die aufwärts nach Merkmalen sucht
  - Oft links-nach-rechts-aufwärts-Suche

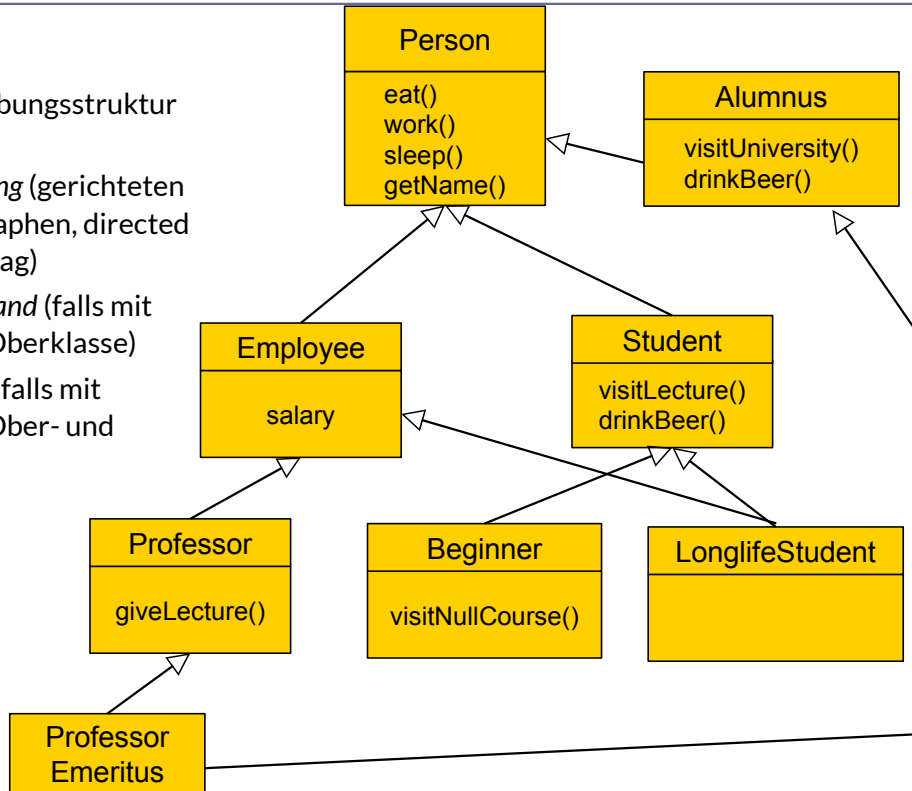


Von wo wird drinkBeer() geerbt?



# Ein grosser Mehrfachvererbungsverband

- ▶ Eine Mehrfachvererbungsstruktur bildet
  - eine *Halbordnung* (gerichteten azyklischen Graphen, directed acyclic graph, dag)
  - einen *Halbverband* (falls mit gemeinsamer Oberklasse)
  - Einen *Verband* (falls mit gemeinsamer Ober- und Unterklasse)



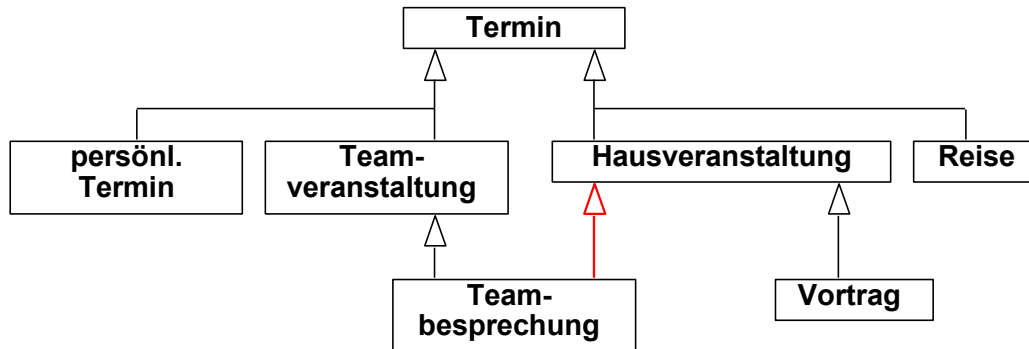


## 31.5.2 Realisierung von Mehrfachvererbung



## Rätsel: Wie realisiert man Code-Mehrfachvererbung in Java?

- ▶ In UML ist es prinzipiell möglich, daß eine konkrete Klasse von mehreren Klassen erbt:



- ▶ Java unterstützt Mehrfachvererbung von konkreten Klassen *nicht*, nur von Schnittstellen:

```
class Teambesprechung extends Teamveranstaltung, Hausveranstaltung
```



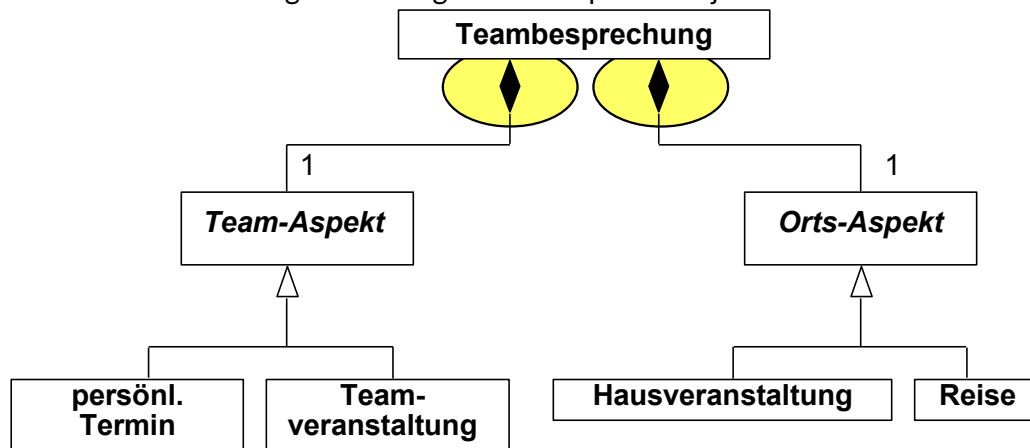
```
class Teambesprechung extends Teamveranstaltung
    implements Hausveranstaltung
```

Beim Übergang zu Java muss eine der Vererbungen ausgewählt und durch Komposition ersetzt werden, wahlweise auch alle.



## Lösung: Realisierung von Mehrfachvererbung durch Komposition in kompositen Objekten

- ▶ Mehrfachvererbung wird häufig durch komposite Objekte simuliert



```
class Teambesprechung {
    private Teamaspekt ta;
    private Ortsaspekt oa;
    ...
}
```

```
abstract class Teamaspekt {
}

abstract class Teamveranstaltung
    extends Teamaspekt {
    ...
}
```

Hier haben wir ein komposities Objekt mit zwei Kompositionsrelationen, das nun auf Java abbildbar ist. Vererbung wird also durch Aggregation “simuliert”.

Allerdings verlieren wir die Information der Komposition, weil Java nur Referenzen kennt. D.h. ein Teilobjekt “Team-Aspekt” weiß zur Laufzeit nicht mehr, dass es Teil von “Teambesprechung” ist. “Objektschizophrenie” ist ein Phänomen, das in der Vorlesung “Design Patterns and Frameworks” im WS näher untersucht wird.

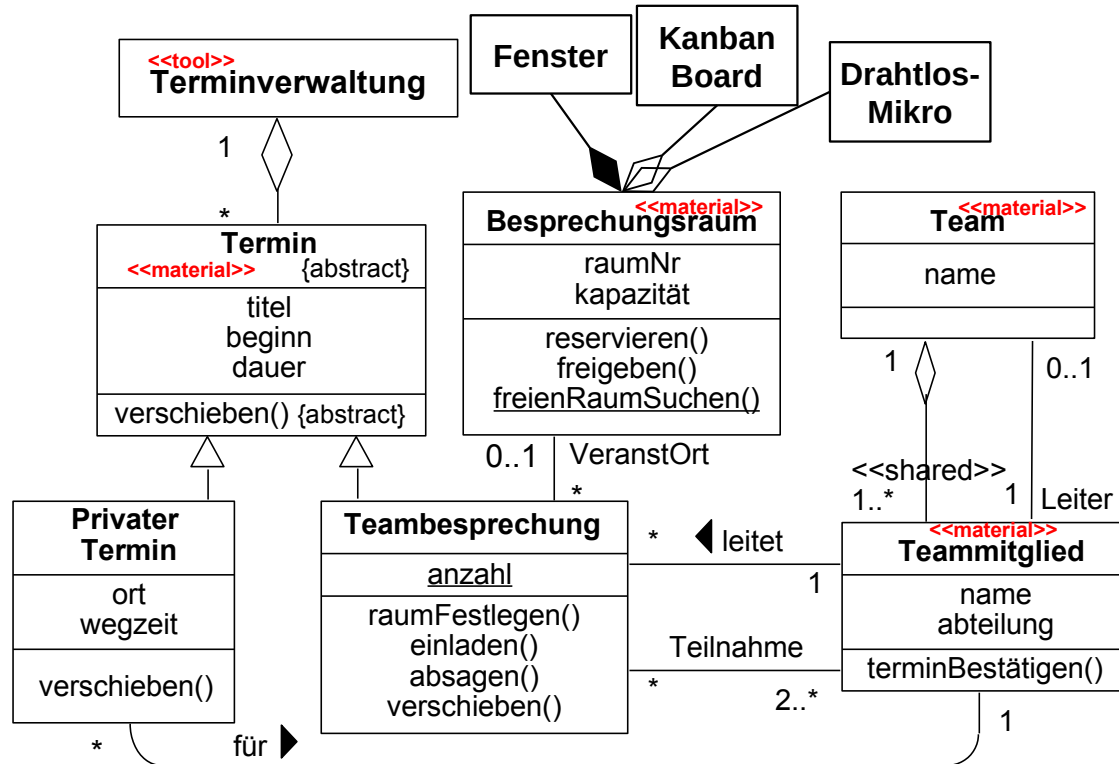
## Verschiedene Ähnlichkeitsrelationen in Analysemodellen (Similarity Relationships)

- ▶ **is-a (ist-ein):** zeigt in Analysetexten und Analysemodellen *allgemeine Ähnlichkeit* an:
  - In Mengenhierarchien ist die Untermengenrelation gemeint (subset)
  - In Begriffshierarchien Unterkonzept-Relation (subconcept)
  - is-a ist azyklische Relation, bei einfacher Vererbung baumförmig
- ▶ **is-structured-like (teilt-Struktur-mit):** zeigt ähnliche Struktur an (strukturelle Ähnlichkeit oder Gleichheit)
- ▶ **behaves-like (verhält-sich-wie):** zeigt Verhaltensähnlichkeit an
  - **conformant (always-behaves-like):** *ständige Verhaltensgleichheit* (Konformanz) bedeutet Ersetzbarkeit (substitutability)
  - **sometimes-behaves-like:** *gelegentlich verhaltensgleich*
  - **restrictedly-behaves-like:** meist konformant, aber nicht in speziellen Situationen (extravagance, restriction inheritance)
- ▶ **Achtung:** *is-a, is-structured-like, behaves-like, conformant* werden alle *Vererbung* genannt
  - Dagegen: *instance-of:* A ist aus einer Schablone S gemacht worden

Das Wort “Vererbung” erzeugt oft Verwirrung, weil für es unterschiedliche Semantiken vorliegen. In der Analyse werden alle diese Arten von Vererbung genutzt und sollten sprachlich bzw. durch Stereotypen unterschieden werden.

## Beispiel: Analyse-Klassendiagramm

96 Softwaretechnologie (ST)



Das hier ist nun ein vollständiges Analysediagramm des fachlichen Modells (Domänenmodells), mit fragmentarischen Attributen, Methoden, aber doch einer präzisen Relationsmodellierung.

Viele Klassen sind Materialklassen, d.h. passive Datenklassen, die persistente Daten enthalten.

## Ausblick: Verfeinerung von Analyse- zum Entwurfsmodell

Analyse-Modell (Fragmente)	Entwurfs-Modell (Mehr Struktur & mehr Details)
<p>Notation: aUML</p> <p>Objekte: Fachgegenstände</p> <p>Klassen: Fachbegriffe, parallele Prozesse, komplexe Objekte mit Endorelationen</p> <p>Attribute ohne Typen und Sichtbarkeiten</p> <p>Operationen: ohne Typen, Parameter und Rückgabewerte</p> <p>Assoziationen: partiell, bidirektional i. Allg. ohne Datentypen</p> <p>Aggregationen, Kompositionen</p> <p>Leserichtung, partielle Multiplizitäten</p> <p>Vererbung: Begriffsstruktur</p> <p>Annahme perfekter Technologie</p> <p>Funktionale Essenz</p> <p>Völlig projektspezifisch</p> <p>Grobe Strukturskizze</p>	<p>Notation: dUML</p> <p>Objekte: Softwareeinheiten</p> <p>Klassen: mit Abstrakt, Interface, Stereotyp; Einsatz von Entwurfsmustern für komplexe Objekte</p> <p>Attribute: Sichtbarkeiten, Ableitung, Klassenattribute, Initialisierung, weitere spezielle Eigenschaften</p> <p>Operationen: voll typisiert, mit Parameter, Rückgabewert, Klassenoperation</p> <p>Unidirektionale Assoziationen mit voller Multiplizität, Navigation, qualifizierte A.</p> <p>Vererbung: Programmableitung</p> <p>Annahme perfekter Technologie</p> <p>Erfüllung konkreter Rahmenbedingungen</p> <p>Gesamtstruktur des Systems</p> <p>Ähnlichkeiten zwischen verwandten Projekten (zwecks Wiederverwendung)</p> <p>Genaue Strukturdefinition</p>

Die Unterschiede von Analyse- und Entwurfsmodellen sind für die Klausur wichtig.

# Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

	Schichten	Analyse	Entwurf	Feinentwurf	Implementierung
Benutzungs-schnittstelle (Boundary)	GUI		Verfeinerung		
	Controller				
Anwendungslogik (Control)	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung System-schnittstellen	stabil	Umsetzen auf jUML	stabil
	Top-Level-Architektur	Verfeinerung des Kontextmodells	stabil	Umsetzen auf jUML	stabil
	Architektur		Ausarbeitung Architektur (PSM)	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Sequentialisierung	Details ausfüllen, Methoden ausprogrammieren
	Tools		Ausarbeitung Tools	Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML; Sequentialisierung	Details ausfüllen, Methoden ausprogrammieren
Daten-haltung (Database)	Material	Aufstellung aus Domänenmodell		Einziehen von Plattformabhängigkeiten (PSM); Umsetzen auf jUML;	Details ausfüllen, Methoden ausprogrammieren

# Was haben wir gelernt?

- ▶ Strukturelle Analyse spürt die Struktur von objektorientierten Anwendungen auf
- ▶ Strukturgetriebene Analyse verwendet das Metamodell, um die Elemente von Modellen aufzuspüren
  - Strukturelle Analyse mittels CRC und UML-Klassendiagramme sind beides Beispiele für strukturgetriebene Analyse
- ▶ Das UML-Metamodell gibt Klassen, Merkmale, und Beziehungen als Strukturelemente vor
  - Diese werden in Interviews mit dem Kunden als Konzepte abgefragt
  - Analysemodelle entstehen durch Analyse von Interviews und können “vorgelesen” werden, um das Interview zu rekonstruieren
- ▶ Analysemodelle müssen beim Übergang ins Entwurfsmodell abgeflacht werden
  - Mehrfachvererbung kann durch Komposition ausgedrückt werden
  - Assoziationen werden durch Graphen und Collections abgebildet
  - Endo-Relationen werden durch Entwurfsmuster wie Bridge, Composite abgebildet



# The End

- ▶ Erklären Sie den Begriff des Metamodells und den Unterschied von Strukturgleichheit und Verhaltensgleichheit zweier Objekte.
- ▶ Wie schafft man es, aus einer UserStory möglichst viel herauszuholen?
- ▶ Wie leitet man aus einem Metamodell Fragen zur Analyse ab?
- ▶ Wie führt man den Prozess der strukturellen fragegetriebenen Analyse durch?
- ▶ Was ist ein komplexes Objekt und welche Arten von Endorelationen gibt es?
- ▶ Wie unterscheidet sich die <<part-of>> Relation von <<plays-a>>?
- ▶ Warum kann ein komplexes Objekte eine seiner Rollen verlieren und wieder annehmen?
- ▶ Wie kann man in C++, das Mehrfachvererbung besitzt, eine Klasse mit 10 Oberklassen realisieren?
- ▶ Einige Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.





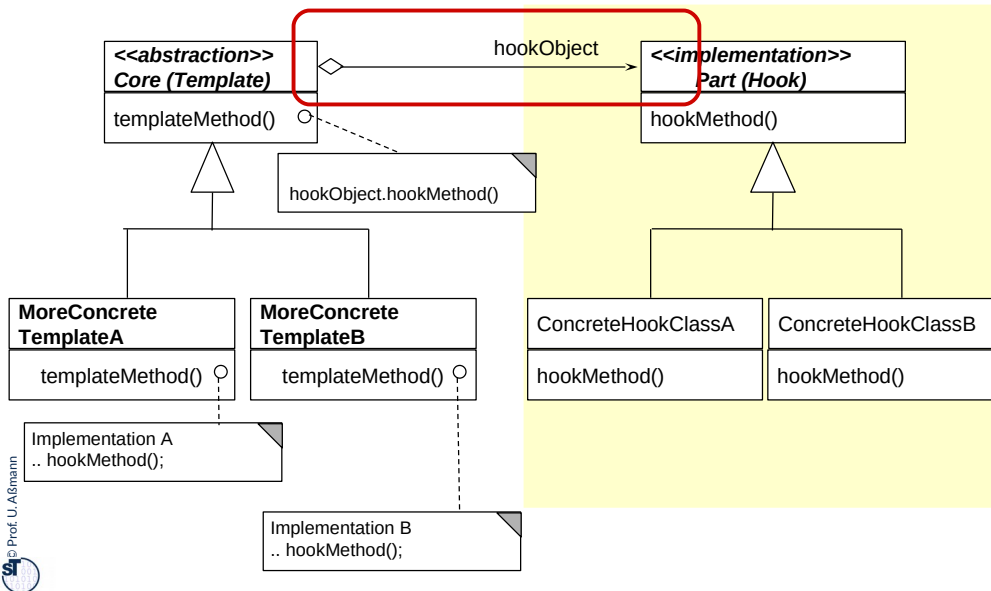
## Rätsel-Lösungen





## Endoassoziation in Bridge und Dimensional Class Hierarchies

- ▶ Bridge implementiert komplexe Objekte mit "hookObject" als Endoassoziation



- Bridge besitzt genau eine Endoassoziation von "Abstractum" zu "Implementation".



## A.31.1 Analyse von Merkmalen

Einige Details zu bereits in Teil I eingeführten Konzepten



- ▶ Eine **Botschaft (Message, Nachricht)** ist eine Nachricht eines Senders an ein Empfänger-Objekt, eine Operation auszuführen
- ▶ Eine **Rückmeldung** ist eine Botschaft, die ein Sender einer Botschaft als Antwort erhält
- ▶ Eine **Operation** koppelt eine Botschaft mit einer Reaktion und einer Rückmeldung: "a service that can be requested from an object to effect behaviour" (UML-Standard)
- ▶ Eine **Methode** (method) "is the implementation of an operation" (das "Wie" einer Operation)
  - "In den Methoden wird all das programmiert, was geschehen soll, wenn das Objekt die betreffende Botschaft erhält." (Middendorf/Singer)
  - *synchrone Operation / Methode*: der Sender wartet auf die Beendigung des Service
  - *asynchrone Operation*: ein Service mit Verhalten aber ohne Rückgabe, d.h. der Sender braucht nicht zu warten
- ▶ Ein **Push** ist eine Botschaft mit einem Datum an ein Objekt zur Ablage
- ▶ Ein **Pull** ist eine Botschaft an ein Objekt zur Ablage, das als Rückmeldung ein Datum mitführt

# Klassenattribute (Statische Attribute)

- ▶ Ein **Klassenattribut** beschreibt ein Datenelement, das genau einen Wert für die gesamte Klasse annehmen kann.
  - Es ist also ein Attribut des Klassenprototypen
- ▶ **Notation:** Unterstreichung
- ▶ **Implementierung:**
  - Implementierungsmuster Singleton
  - Klassenattribute und -operationen: Schlüsselwort **static**

<b>Teambesprechung</b>
titel: String beginn: Date dauer: Int <u>anzahl: Int</u>



# Klassenoperation (Statische Operation)

- ▶ **Definition** Eine *Klassenoperation* *A* einer Klasse *K* ist die Beschreibung einer Aufgabe, die nur unter Kenntnis der aktuellen Gesamtheit der Instanzen der Klasse ausgeführt werden kann.  
Gewöhnliche Operationen heißen auch *Instanzoperationen*.
- ▶ **UML Notation:** Unterstreichung analog zu Klassenattributen.
- ▶ **Java:** Die Methode `main()` ist statisch, und kann vom Betriebssystem aus aufgerufen werden

## Besprechungsraum

raumNr  
kapazität

reservieren()  
freigeben()  
freienRaumSuchen()

```
class Steuererklaerung {  
    public static main (String[] args) {  
        Steuerzahler hans =  
            new Steuerzahler();  
        ...  
    }  
}
```

# Operationen in der Analyse

- ▶ **Def.:** Eine **Operation** einer Klasse *K* ist die Beschreibung einer Aufgabe, die jede Instanz der Klasse *K* ausführen kann.

Teambesprechung
titel beginn dauer
raumFestlegen einladen absagen

Teambesprechung
titel beginn dauer
raumFestlegen() einladen() absagen()

- ▶ "Leere Klammern":
  - In vielen Büchern (und den Unterlagen zur Vorlesung) zur Unterscheidung von Attributnamen: raumFestlegen(), einladen(), absagen() etc.
  - Auf Analyseebene gleichwertig zu Version ohne Klammern



# Parameter und Datentypen für Operationen

- ▶ Detaillierungsgrad in der Analysephase gering
  - meist Operationsname ausreichend
  - Signatur kann angegeben werden
  - Entwurfsphase und Implementierungsmodell: vollständige Angaben der Typen ist nötig, um Fehler in der Programmierung früher zu erkennen (frühe oder statische Typisierung)
- ▶ **Notation:**  
**Operation (Art Parameter:ParamTyp=DefWert, ...): ResTyp**
  - *Art* (des Parameters): **in**, **out**, oder **inout** (weglassen heißt **in**)
  - *DefWert* legt einen Default-Parameterwert fest, der bei Weglassen des Parameters im Aufruf gilt.
- ▶ **Beispiel** (Klasse Teambesprechung):  
raumFestlegen (in wunschRaum: Besprechungsraum): Boolean



- ▶ **Definition** Die *Spezifikation* einer Operation legt das Verhalten der Operation fest, ohne einen Algorithmus festzuschreiben.

Es wird das "**Was**" beschrieben und noch nicht das "**Wie**".

- ▶ Häufigste Formen von Spezifikationen:
  - Signaturen (Typen der Parameter und Rückgabewerte)
  - Text in natürlicher Sprache (oft mit speziellen Konventionen)
    - Oft in Programmcode eingebettet (Kommentare)
    - Werkzeugunterstützung zur Dokumentationsgenerierung, z.B. "javadoc"
  - Vor- und Nachbedingungen (Verträge, contracts)
  - Tabellen, spezielle Notationen
  - "Pseudocode" (Programmiersprachenartiger Text)
  - Zustandsmaschinen, Aktivitätendiagramme





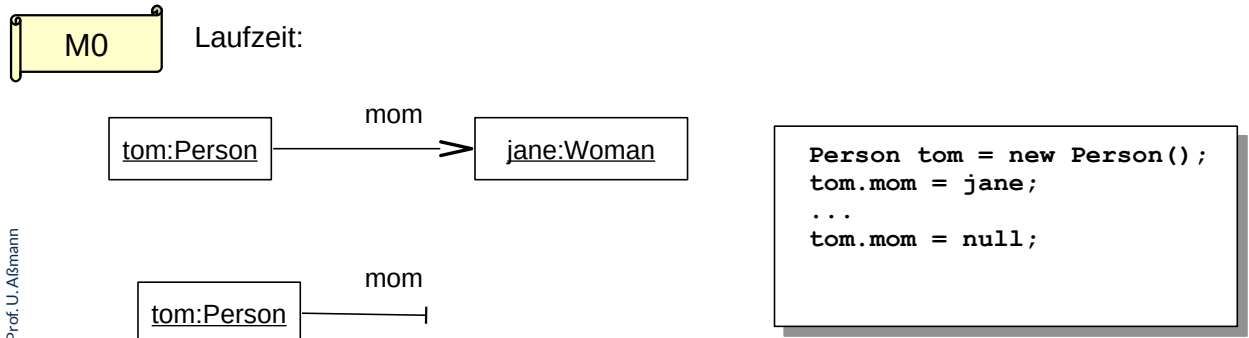
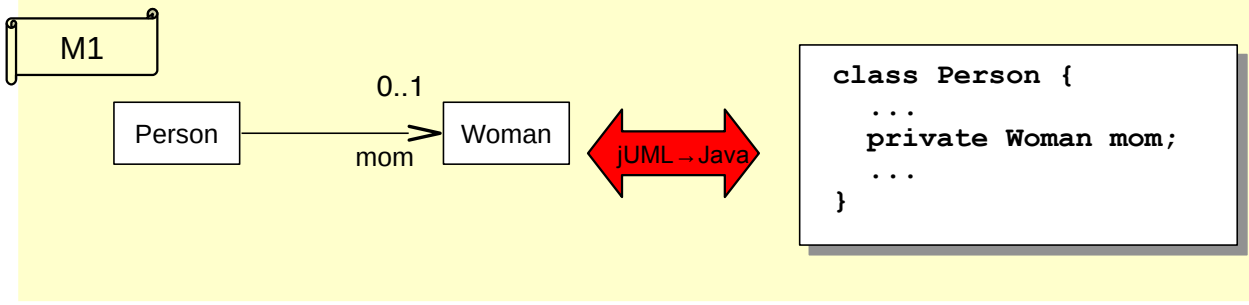
## A.31.3 Analyse von Assoziationen

Einige Details zu bereits in Teil I eingeführten Konzepten



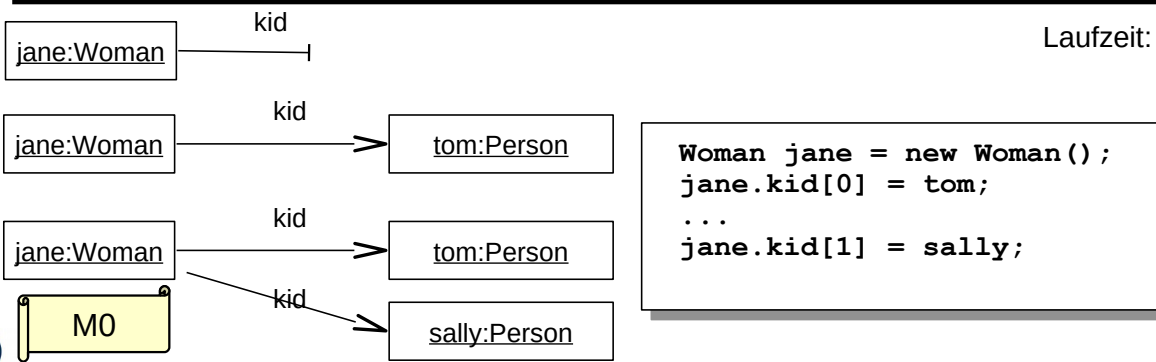
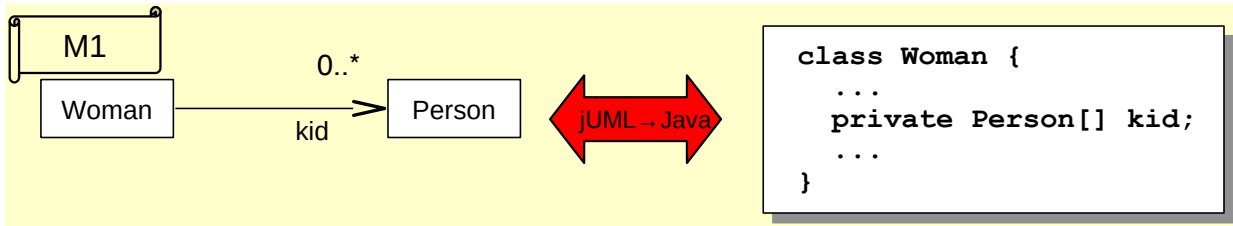
# Einseitige einstellige Assoziationen in Java und jUML (Wdh.)

- Ein Kind kann höchstens eine Mutter haben



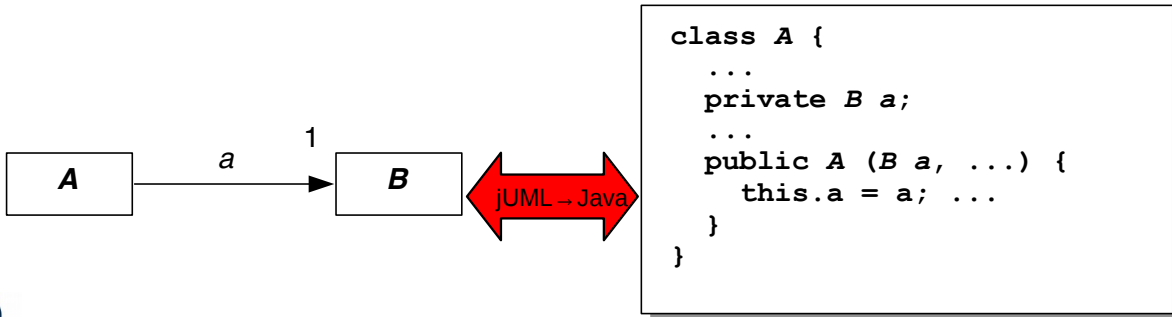
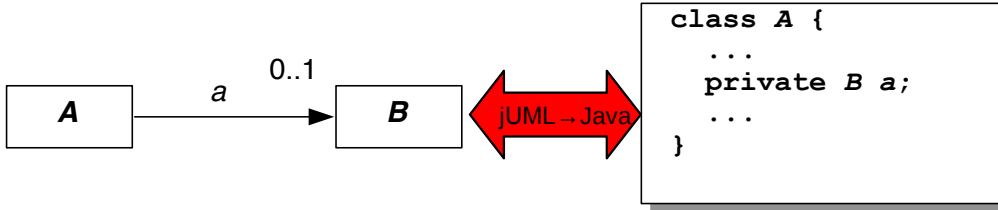
# Einseitige mehrstellige Assoziationen (Wdh.)

- ▶ Eine Mutter kann aber viele Kinder haben
  - Annahme: Die Obergrenze der Anzahl der Child-Objekte spätestens bei erstmaliger Eintragung von Assoziationsinstanzen bekannt und relativ klein. (Allgemeinere Realisierungen siehe später.)



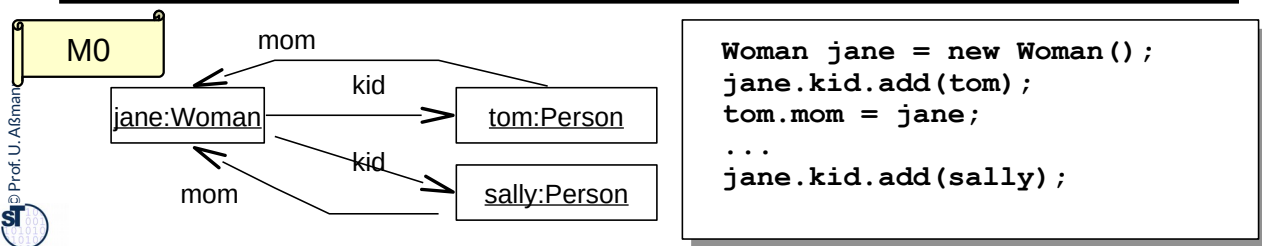
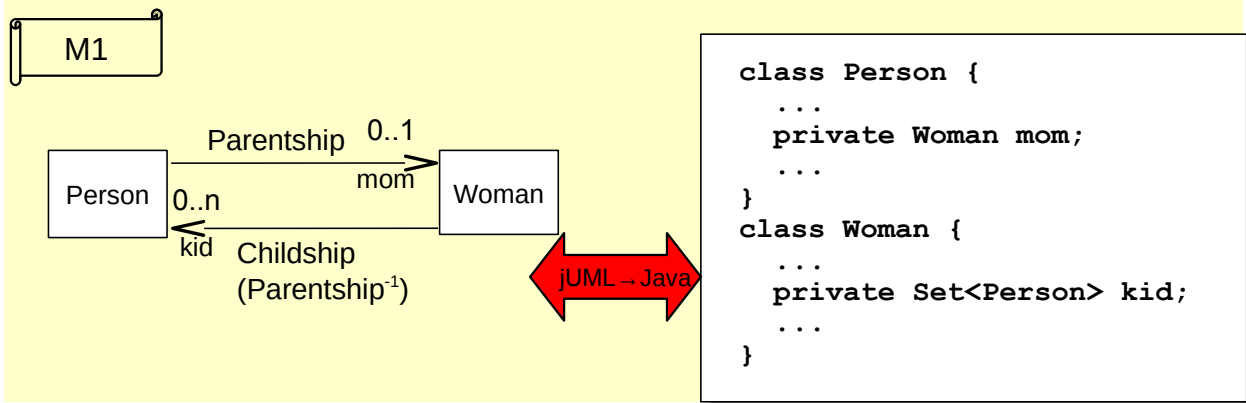
# Optionale und notwendige Assoziationen

- ▶ Untere und obere Schranken von unidirektionalen Assoziationen können durch die Einführung von Argumenten in Konstruktoren eingehalten werden
  - Analog z.B. für Multiplizitäten 0..\* und 1..\*



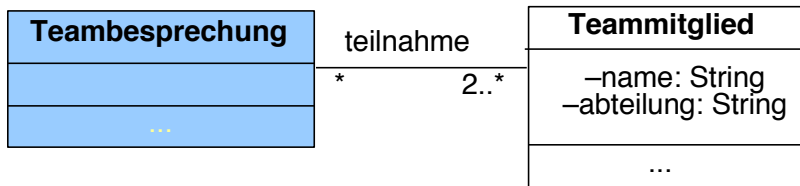
## Zu 31.3.2. Realisierung von bidirektionalen Assoziationen durch unidirektionale

- Realisierung von jUML in Java durch Arrays oder Collections (s. Kap. 21-collections)



# Realisierung von bidirektionalen Assoziationen durch unidirektionale: Beispiel UML/Java

- ▶ Achtung: fixe Multiplizitäten müssen in Java durch Programmierung kontrolliert werden, z.B. in Konstruktoren (s. Kap. 21-collections)

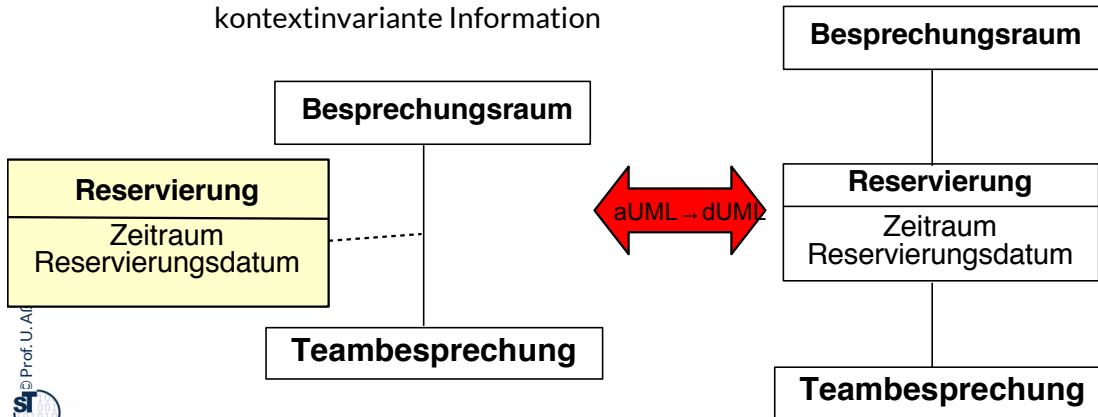


```
class Teambesprechung {
    private Teammitglied[] teilnahme;
    ...
    public Teambesprechung (
        Teammitglied[] teilnehmer) {
        this.teilnahme = teilnehmer;
    }
}
```

```
class Teammitglied {
    private String name;
    private String abteilung;
    private Teambesprechung[] teilnahme;
    public Teammitglied (
        Teambesprechung[] teilnahme) {
        if (teilnahme.size() < 2) error();
        this.teilnahme = teilnahme;
    }...
}
```

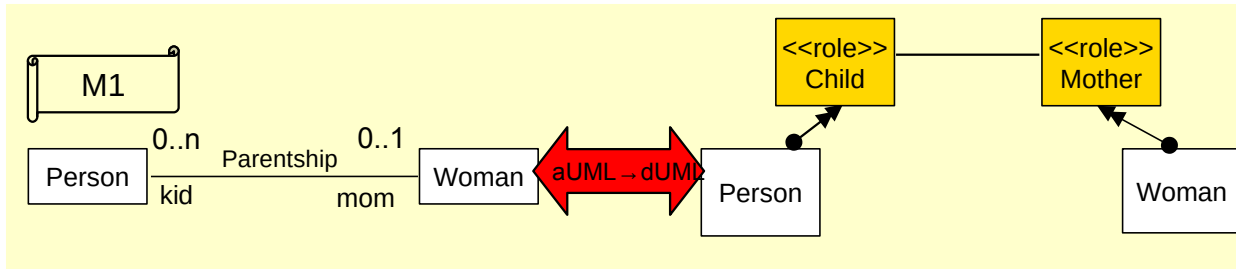
## 31.3.2.4 Realisierung von Assoziationsklassen mit „Zwischenklassen“

- ▶ Assoziationsklassen können im Implementierungsmodell in normale relationale „Zwischen“-Klassen abgeflacht werden:
  - Attribut "Reservierungsdatum" für eine Raumreservierung wird für Assoziation benötigt (z.B. um Reservierungskonflikte aufzulösen).
  - Die Klasse "Reservierung" wird in die bestehende Assoziation eingefügt und "zerlegt" sie in zwei neue Assoziationen.
  - Die Klasse "Reservierung" trägt die kontextspezifische Information, "Besprechungsraum" und "Teambesprechung" fokussieren sich auf die kontextinvariante Information

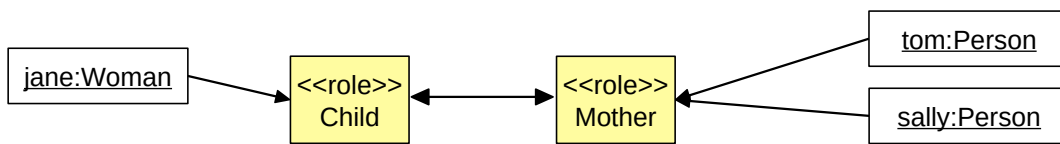


## 31.3.2.5 Realisierung von bidirektionalen Assoziationen durch Rollenklassen

- ▶ Assoziationen können durch Rollenklassen realisiert werden, die das kontextspezifische Verhalten tragen
- ▶ Person, Woman tragen das kontextinvariante Verhalten
- ▶ Grundlage aller kontextadaptiven Software



M0 Laufzeit: Referenzen zwischen Rollenobjekten und den assoziierten Objekten





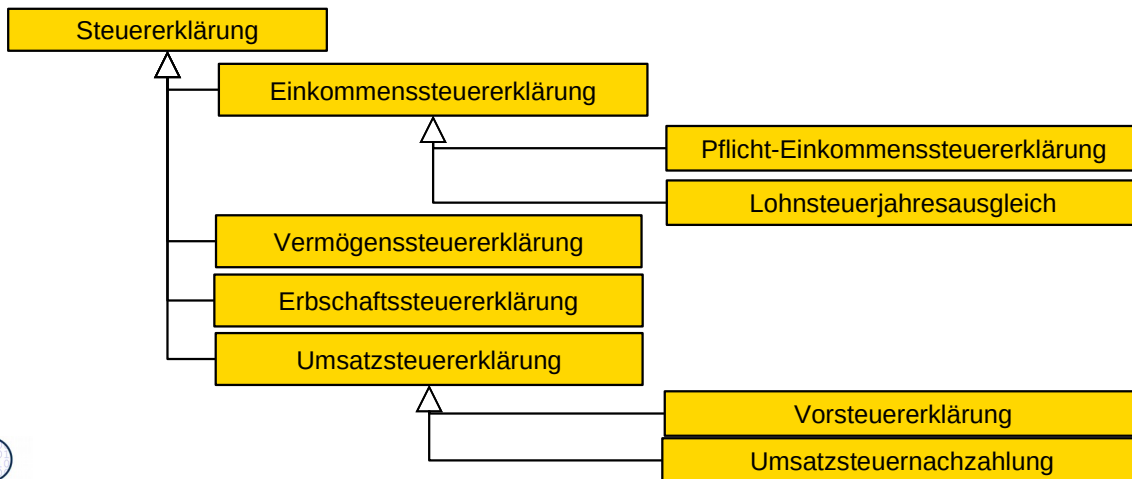


## A.31.4 Weitere Beispiele für Zeilenhierarchien



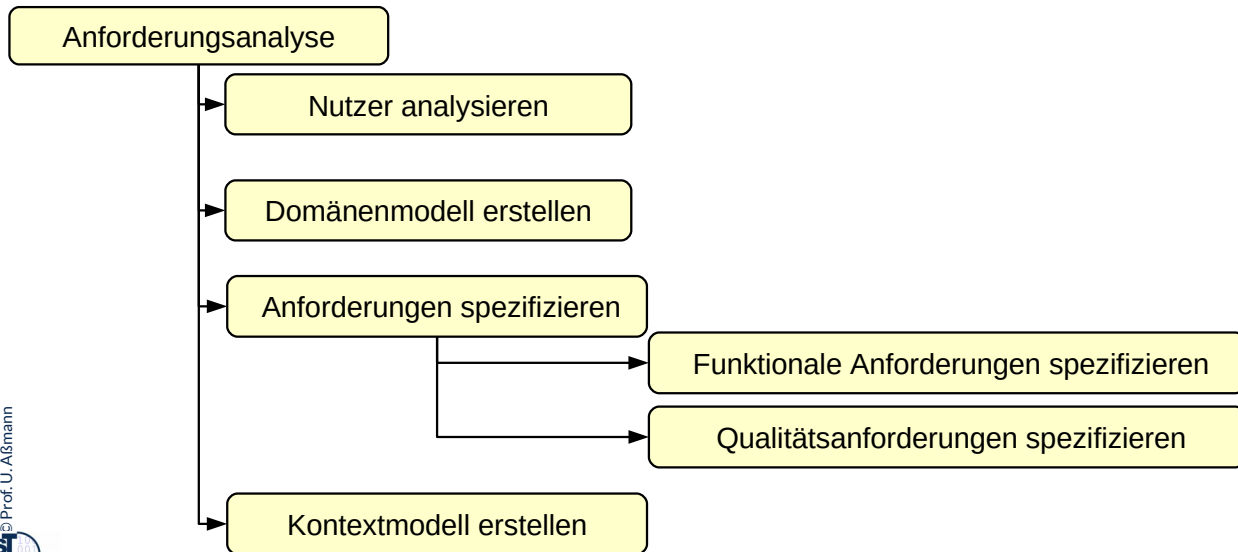
# Zeilenhierarchien

- ▶ Hierarchien kann man als auch als *Zeilenhierarchien (horizontal dekomponierte Textbäume)* anordnen, analog zu einem *tree widget*
  - Teile können leicht zu- und aufgeklappt werden
  - Textuelle Annotationen können einfach in den Zeilen hinzugefügt werden
- ▶ Am einfachsten sieht man das an einer **Begriffshierarchie (Taxonomie)**
  - Ontologieeditoren benutzen dieses Format (z.B. Protege)
  - Zum Lernen verwendet man Begriffshierarchien [Wolf, 2.1]



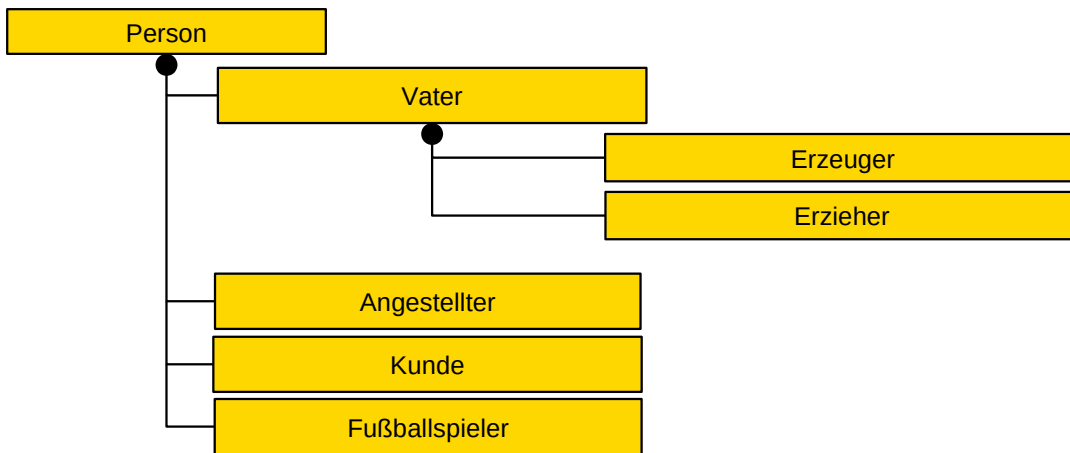
## Bsp. Aktivitätenhierarchie (Aktivitäts- oder Funktionsbaum)

- ▶ Aktivitäten werden in UML durch leicht abgerundete Rechtecke dargestellt
- ▶ Strukturierte Aktivitäten können in Zeilenhierarchien dargestellt werden



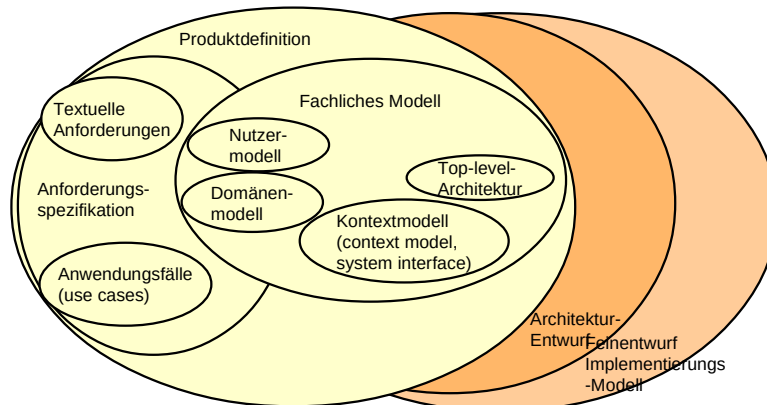
# Eigenschaftshierarchie

- ▶ Auch integrates-a-Hierarchien kann man als auch als Zeilenhierarchien anordnen
- ▶ Integrates-a stellt Hierarchien von Prädikaten über ein großes Objekt dar
- ▶ Eine Integrationshierarchie verwendet die Endo-Assoziation integrates-a



# Vom Analysemodell zum Entwurfsmodell

- ▶ Analysemodelle (in aUML) sind *größer* als Entwurfsmodelle (in dUML).
  - Beim Übergang zum Entwurfsmodell wird das Analysemodell *verfeinert*, d.h. *ausgefüllt* und *detailliert*.
  - Das Analysemodell ist sozusagen das *Skelett* des Entwurfsmodells
    - Domänenmodell ergibt Kontextmodell ergibt Top-Level-Architektur ergibt Architekturmodell ergibt Implementierungsmodell
    - Domänenmodell ergibt Materialentwurf



Das **Analysemodell** besteht aus Fragmentgruppen (Fragmenten und generischen Fragmenten) von UML

- ▶ Verfeinerungsschritte vom Analysemodell (aUML) zum Entwurfsmodell (dUML)
  - **Vervollständigung** (Ausfüllen, Elaboration) von Fragmenten zu vollständigen Modellen
    - Detaillierung von Fragmenten mit optionalen Einzelheiten
    - Anfügen von Typen, Multiplizitäten und Constraints
  - **Strukturierung**, Restrukturierung von Fragmenten
    - Vererbung, Generizität, Entwurfsmuster
  - **Abflachen** von Fragmenten (Flachklopfen, lowering, **Realisierung**): Ersetzen von ausdrucksstarken Konstrukten durch weniger ausdrucksstarke, implementierungsnähere
  - **Erhöhung der Zuverlässigkeit**: Einziehen von qualitätssteigernden Fragmenten
    - z.B. Typen von Parametern, generische Typen

# Was man vom Analysemodell abflachen muss

- ▶ Das Analysemodell nutzt verschiedene ausdrucksstarke Sprachkonstrukte (hier aUML), die nicht in der Programmiersprache (Java) vorhanden sind
- ▶ Beim Übergang vom Analysemodell und Entwurfsmodell zum Implementierungsmodell muss man diese in die Programmiersprache umsetzen (*Realisieren, Flachklopfen, lowering*) (→ Teil IV)
- ▶ Abflachen struktureller Eigenschaften
  - Klassen und Objekte:
    - Stereotypen aus Profilen
    - Mehrfachvererbung von Code
    - Aktive Objekte (Parallelität)
    - Komplexe Objekte wie Unterobjekte (Rollen, Facetten, Teile)
    - Konnektoren und Kanäle
  - Relationen:
    - n-stellige Assoziationen, bidirektionale Assoziationen
    - Aggregationen, Kompositionen
- ▶ Abflachen von Verhalten
  - States
  - Activites
  - Ereignisse auffangen und behandeln



## A.31.5. Mehrfachvererbung als Venn-Diagramm

- ▶ Betrachtet man Klassen als Mengen, bildet sich Unterklassenbeziehung auf Teilmengenbeziehung ab
- ▶ Mehrfach-Vererbung ergibt ein Venn-Diagramm mit Überschneidungen

