OOA (Teil III)

31) Strukturelle Analyse, getrieben von Metamodellen

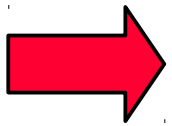
Prof. Dr. rer. nat. habil. Uwe Aßmann
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 15-1.2, 20.06.15

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

- Zuser, Kap. 7-9
- Störrle 5.2-5.5
- Balzert Kap. 6-7, 9-10

Objektorientierte Analyseverfahren



Strukturgetriebene (metamodellgetriebene) Analyse für das Domänenmodell

Strukturelle Analyse Kontextmodell und Top-Level-Architektur

Strukturelle Analyse

Szenarienanalyse (Punktweise Verfeinerung)

Szenarienanalyse (Querschneidende Verfeinerung)

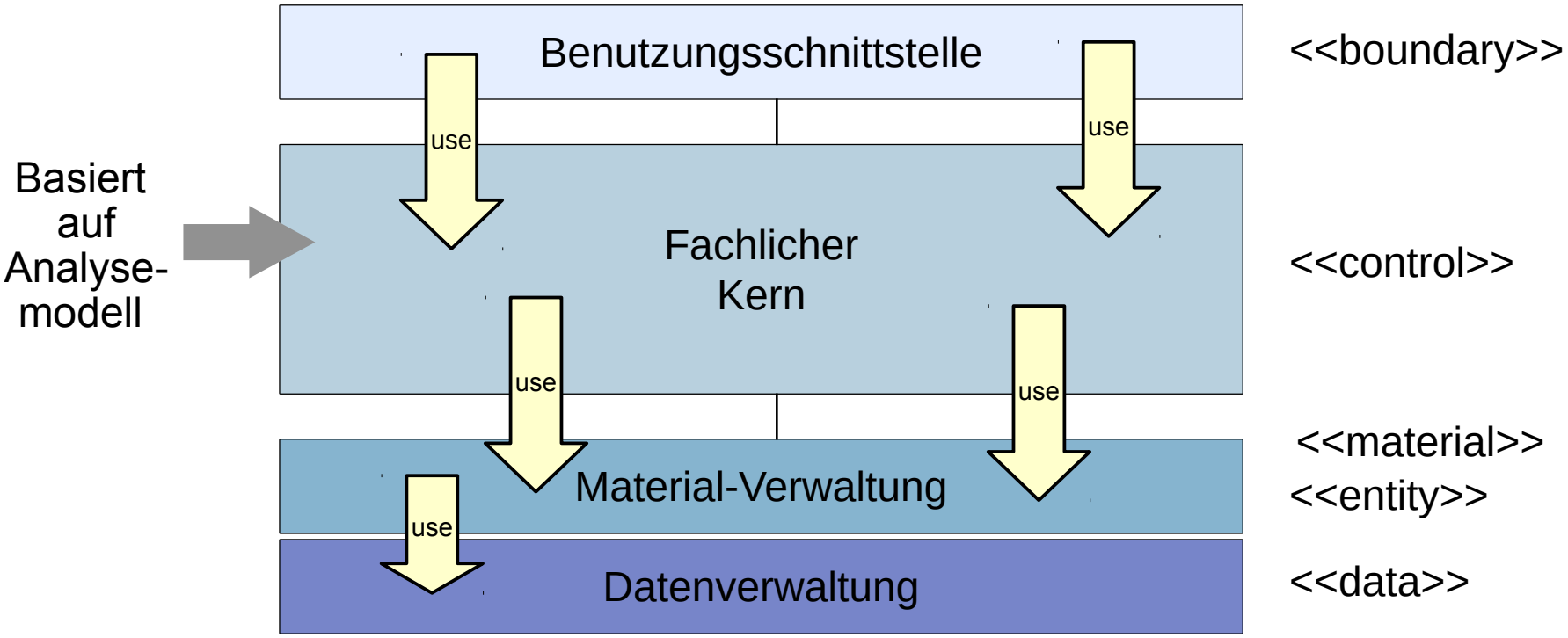
Verhaltens-Analyse

Überblick Teil III: Objektorientierte Analyse (OOA)

1. Überblick Objektorientierte Analyse
   1. Strukturelle Modellierung mit CRC-Karten
2. Strukturelle metamodellgetriebene Modellierung mit UML
   - Analyse des Domänenmodells: Strukturelle metamodellgetriebene Modellierung
     1. Modellierung von komplexen Objekten
   1. Systemanalyse: Strukturelle Modellierung für Kontextmodell und Top-Level-Architektur
3. Analyse von funktionalen Anforderungen (Verhaltensmodell)
   1. Funktionale Verfeinerung: Dynamische Modellierung von Lebenszyklen mit Aktionsdiagrammen
   2. Funktionale querschneidende Verfeinerung: Szenarienanalyse mit Anwendungsfällen, Kollaborationen und Interaktionsdiagrammen
4. Beispiel Fallstudie EU-Rent

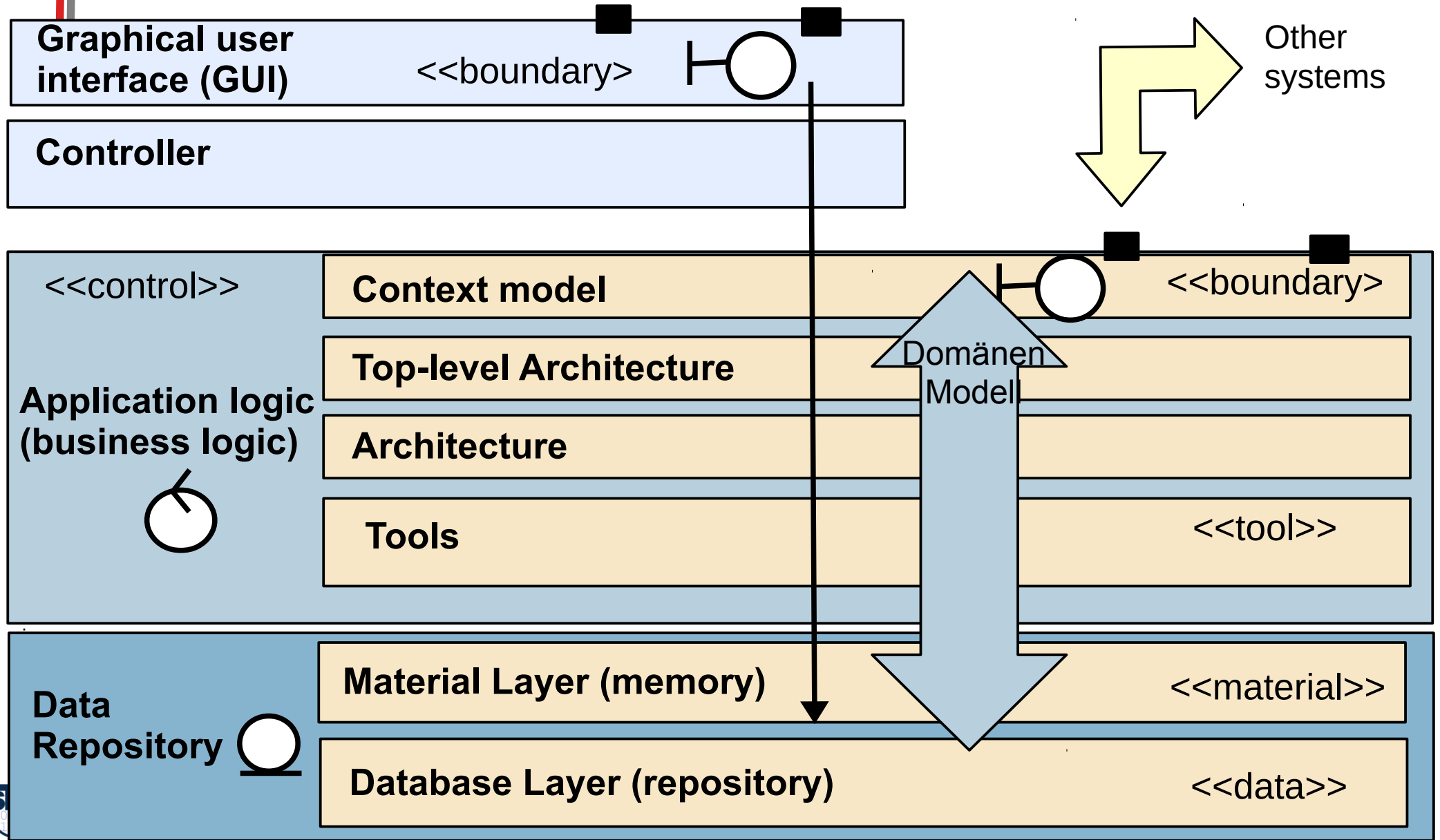
Architekturstil für Interaktive Anwendungen: Vier-Schichten-Architektur (BCED)

- Klassische Struktur eines interaktiven Anwendungssystems
- Integrationstest verläuft wegen azyklischer Benutzungsrelation (use) bottom-up: erst D, dann ED, dann CDE, dann BCED
- Fachlicher Kern (Anwendungslogik) kann weitere Schichten enthalten
  - Oft kapselt eine Facade eine Schicht nach oben ab, dann existieren bereits zwei Teil-Schichten



Q7: Verfeinerte BCE-Schichtung eines Systems

- Im Teil III+IV verwenden wir 7 Schichten in 3 Gruppen:



Q5: Schritte der Modellierung in Bezug auf Schichten des Systems

	Schichten	Analyse aUML	Entwurf dUML	Feinentwurf jUML	Implementierung Java
Benutzungs-schnittstelle (Boundary)	GUI	Verfeinerung			
	Controller				
Anwendungs logik (Control)	Kontextmodell	Aufstellung aus Domänenmodell; Erarbeitung Systemschnitt-stellen in aUML	stabil	Umsetzen auf jUML	stabil
	Top-Level-Architektur	Verfeinerung des Kontextmodells	stabil	Umsetzen auf jUML	stabil
	Architektur		Ausarbeitung Architektur (PSM) in dUML	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

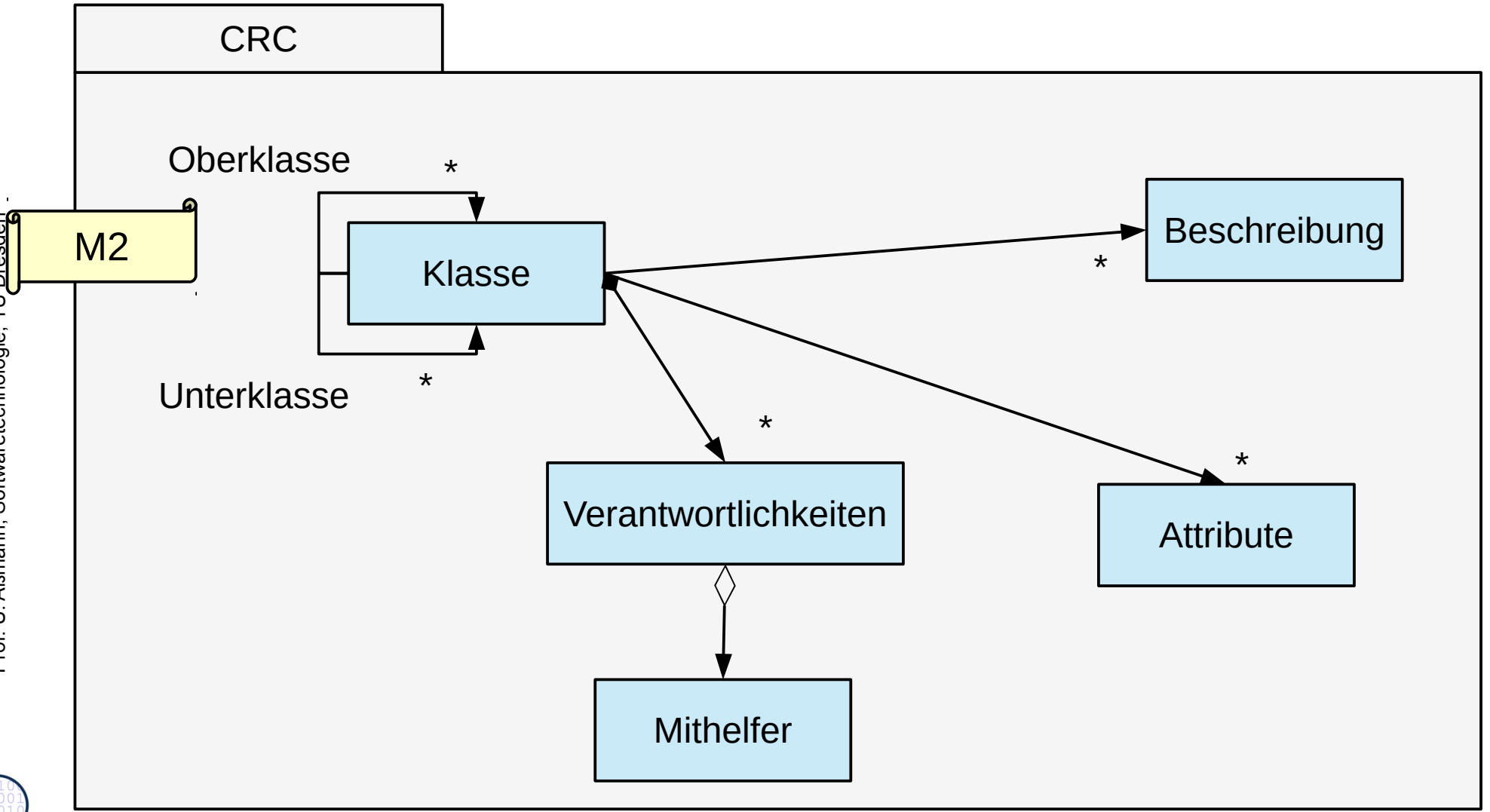
31.1 Strukturmodelle (Metamodelle)

Modelle sind überall...

Ihre Struktur wird beschrieben durch *Metamodelle*

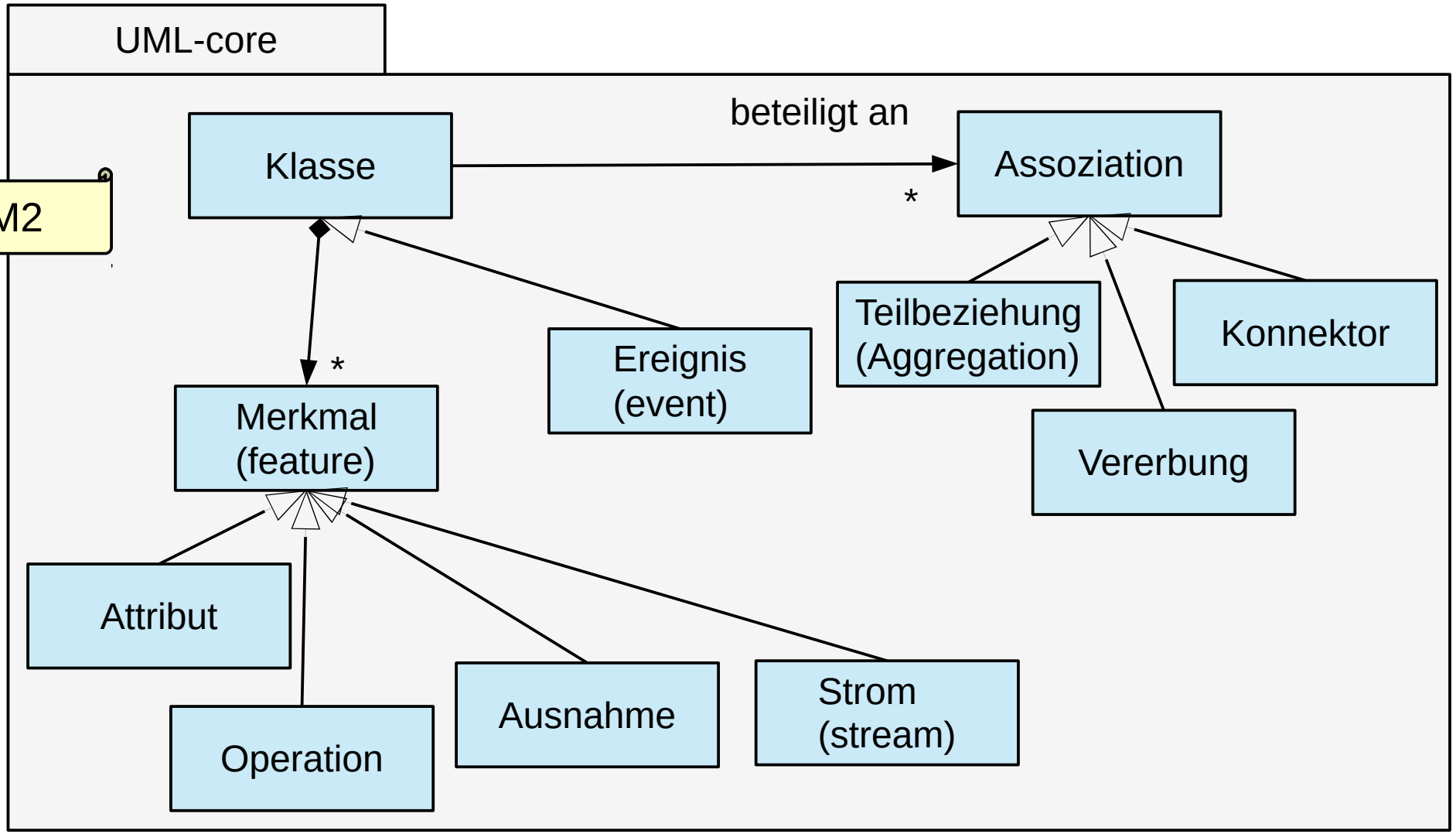
Metamodelle

- Ein **Strukturmodell (Metamodell, Schema, Typsystem)** beschreibt die *Struktur einer Menge von Modellen* mit Hilfe eines (Meta-)Klassendiagramms
- Bsp.: Vereinfachtes Metamodell von CRC (Metaklassen in Blau):



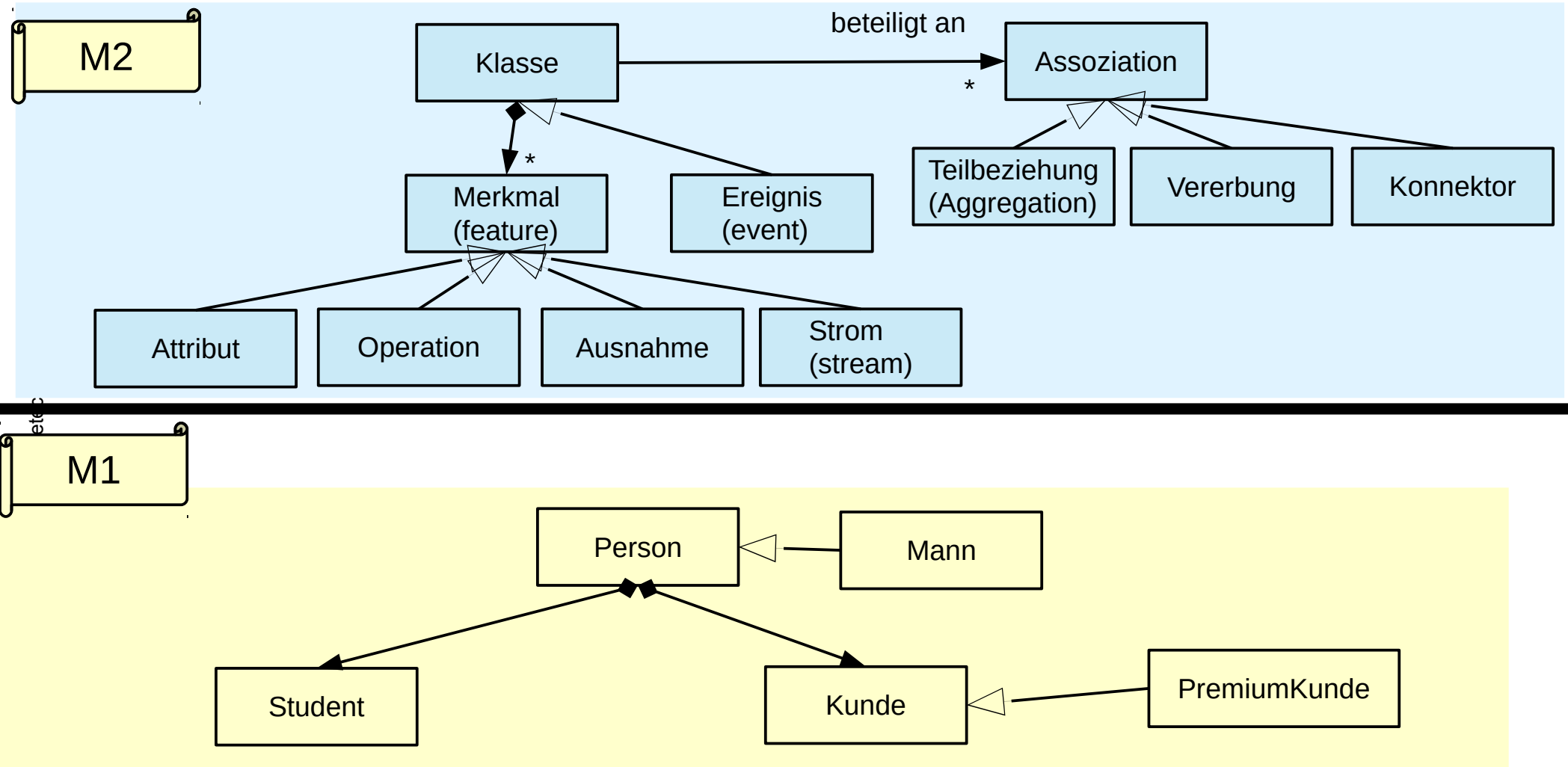
Metamodelle

- In einem Metamodell stellen die Metaklassen die *Sprachkonzepte* der modellierten Sprache vor
- Vereinfachtes Metamodell von UML:



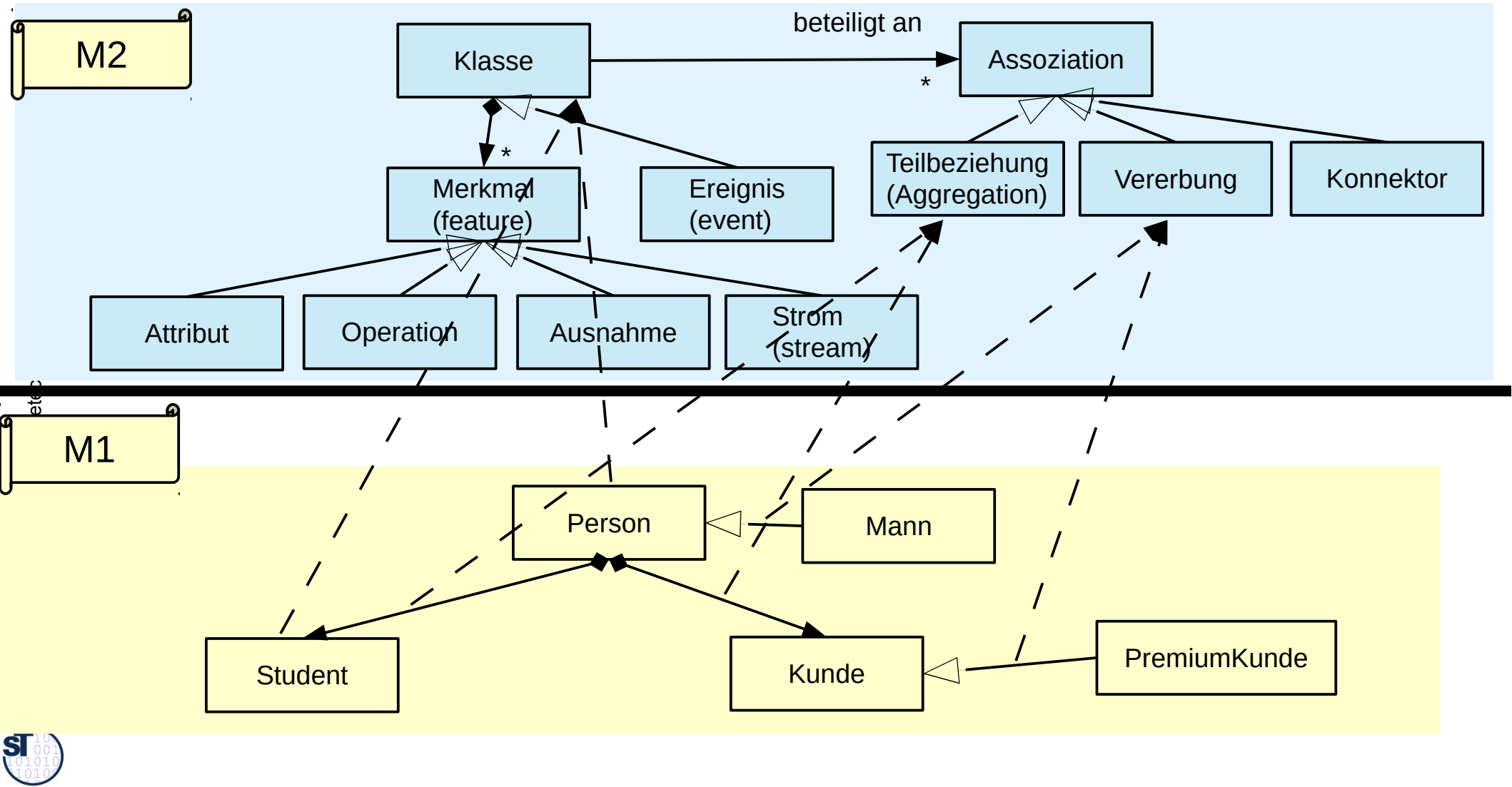
Metamodelle

- Metamodelle liegen auf einer neuen Modellierungsebene M2
- Wie sind M1 und M2 miteinander verbunden?



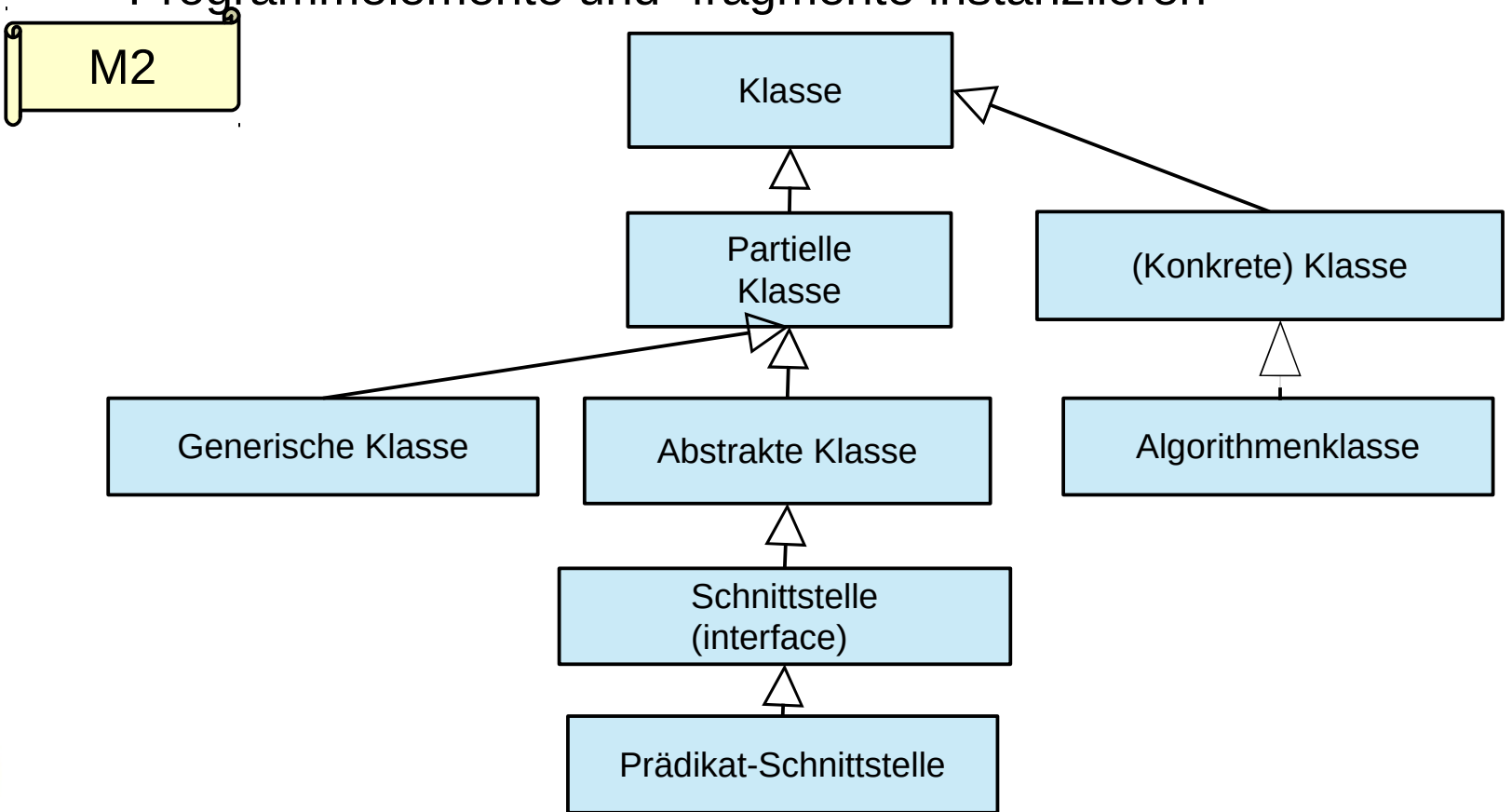
Metamodelle

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



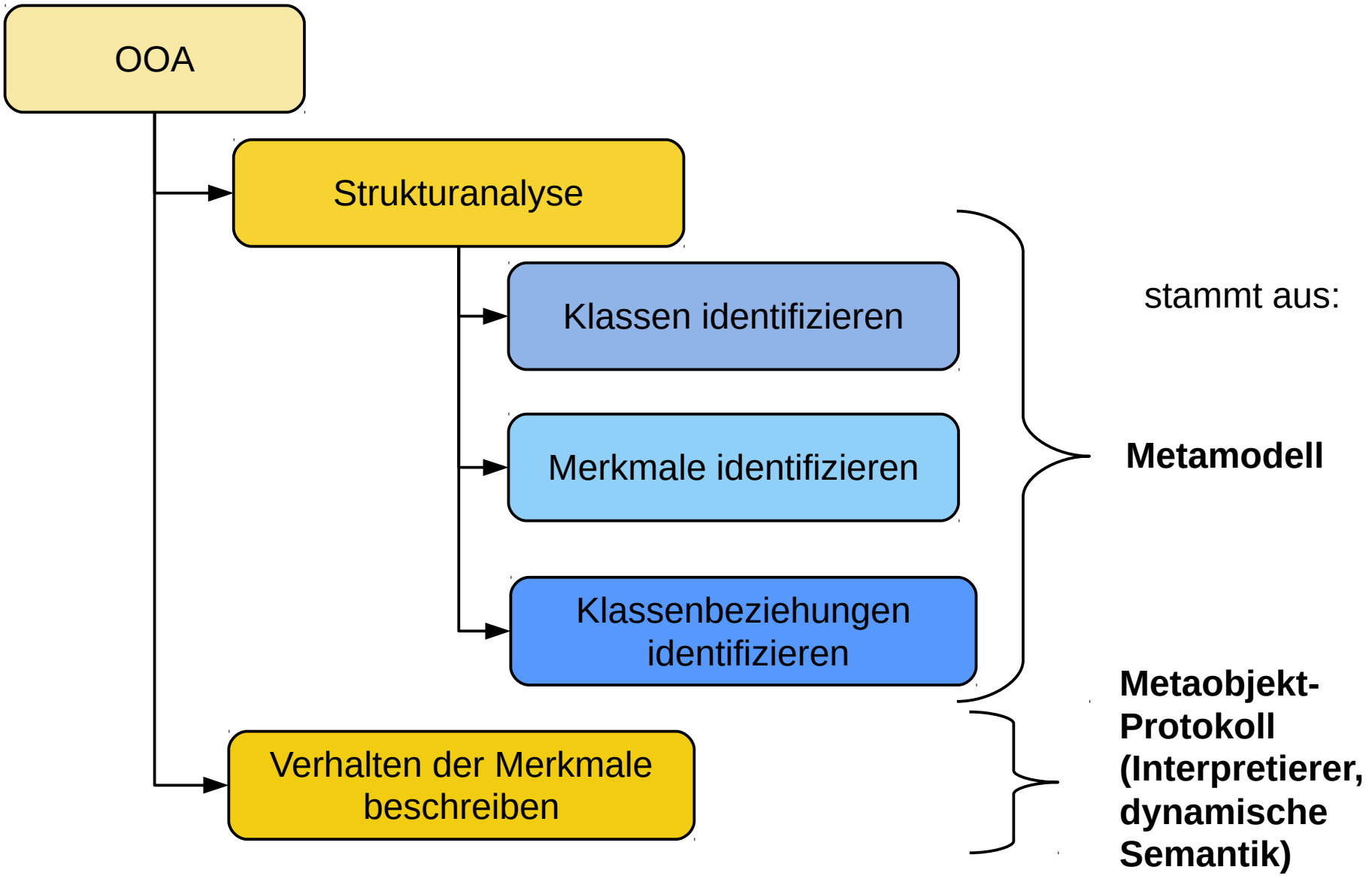
Exkurs 31.E.1 Begriffshierarchie von Klassen (Erinnerung)

- 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 instanziieren



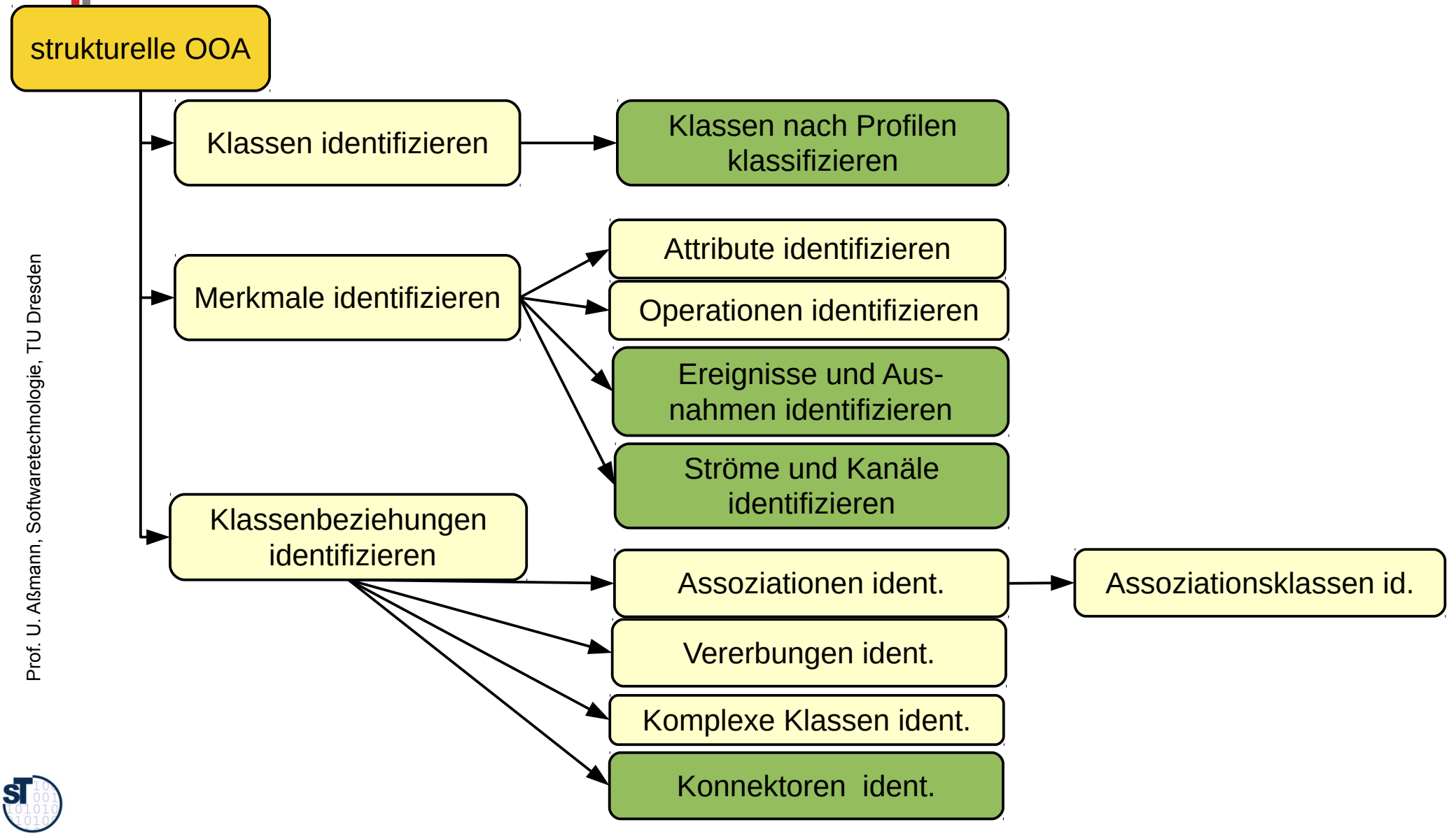
Hauptschritte der strukturellen, metamodellgetriebenen Analyse mit UML

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



Schritte der strukturellen, metamodellgetriebenen Analyse

- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



Strukturgetriebene Analyse

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

Enthält ein Metamodell das Konzept einer Klasse, fragen wir „Welche Klassen und Objekte brauchen wir?“

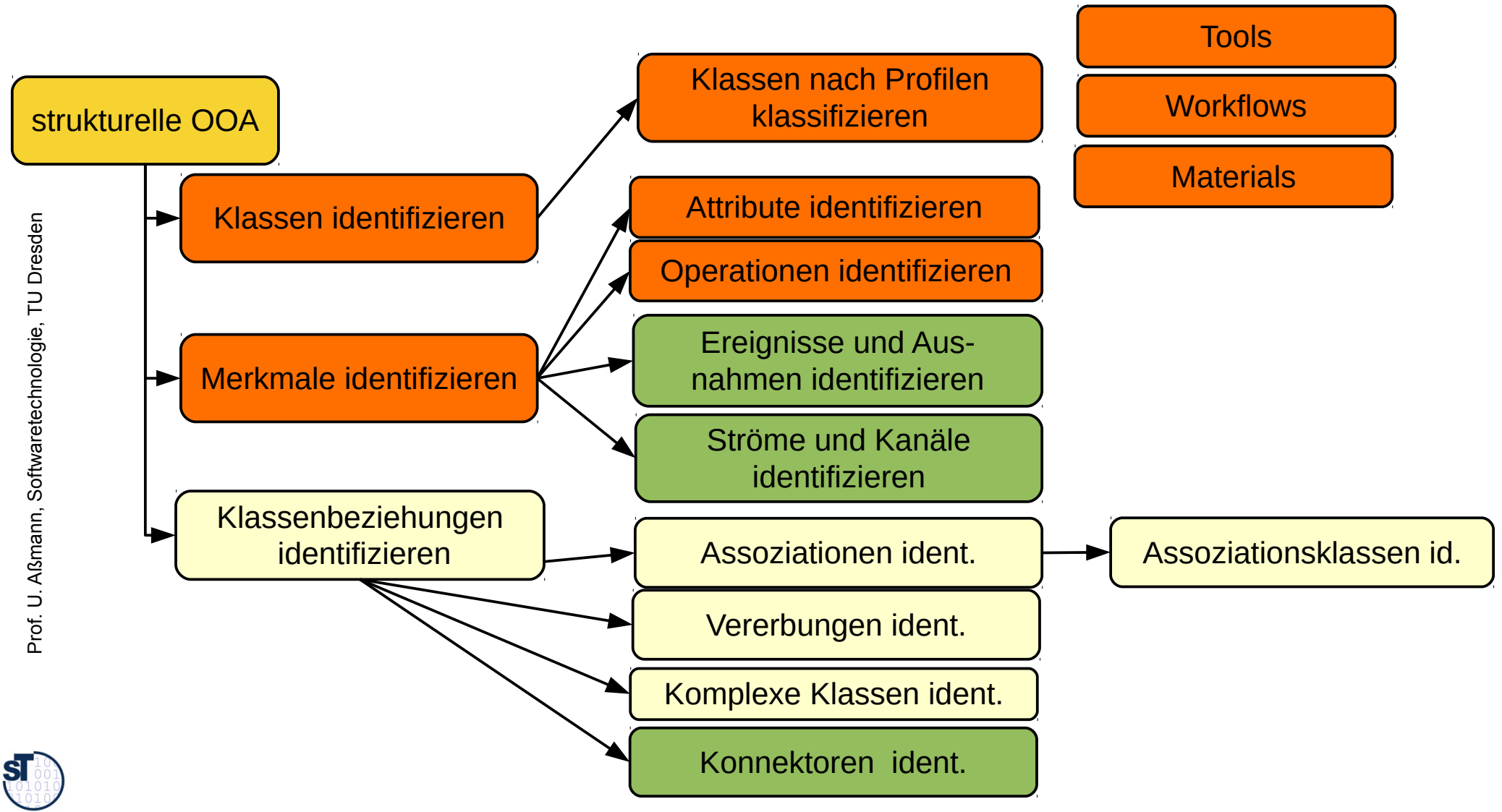
Enthält ein Metamodell das Konzept einer Methode, fragen wir „Welche Methoden brauchen wir?“

Enthält ein Metamodell das Konzept eines Streams, fragen wir „Welche Streams brauchen wir?“
usw.

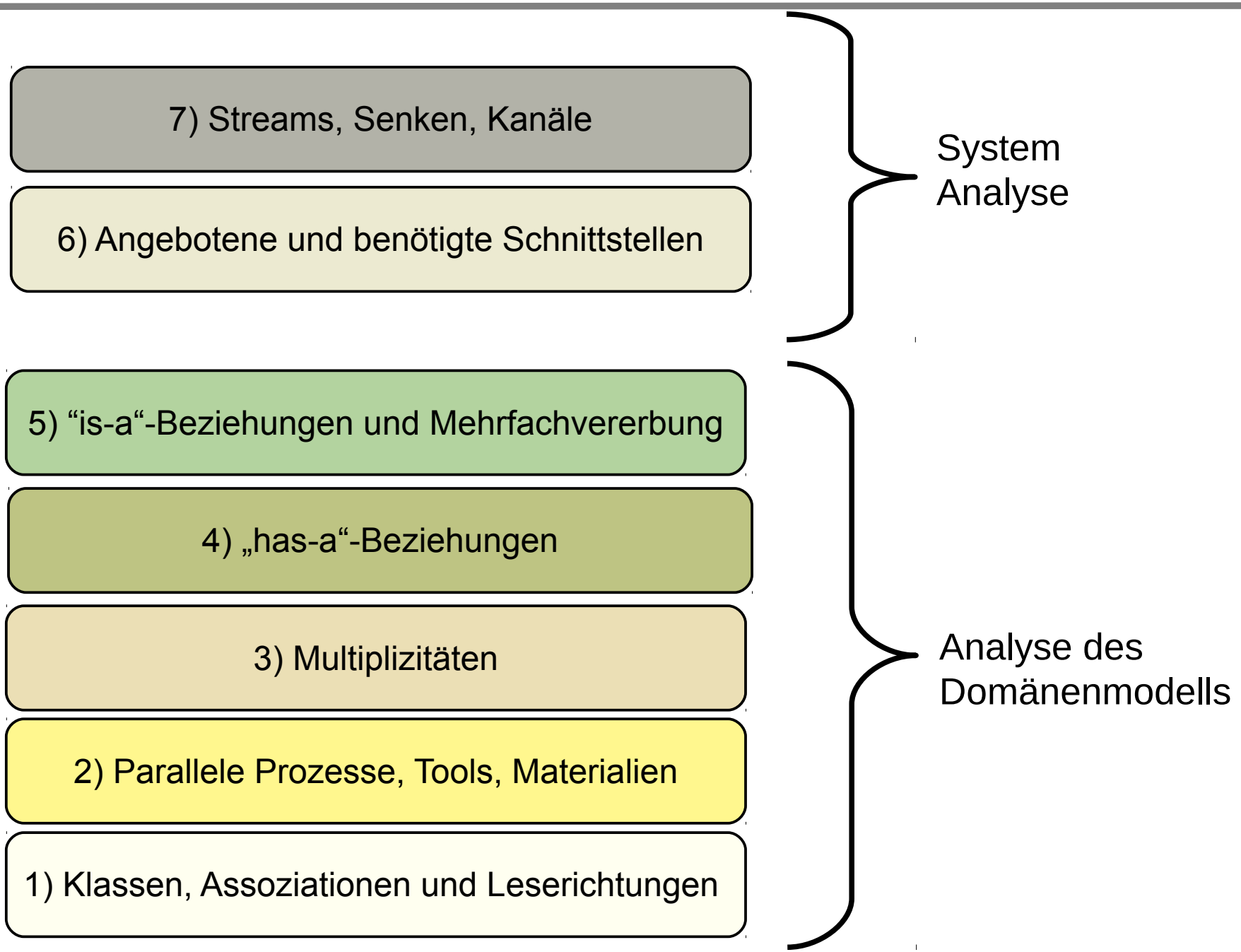
31.2 Analyse von Klassen und Merkmalen

Schritte der strukturellen, metamodellgetriebenen Analyse

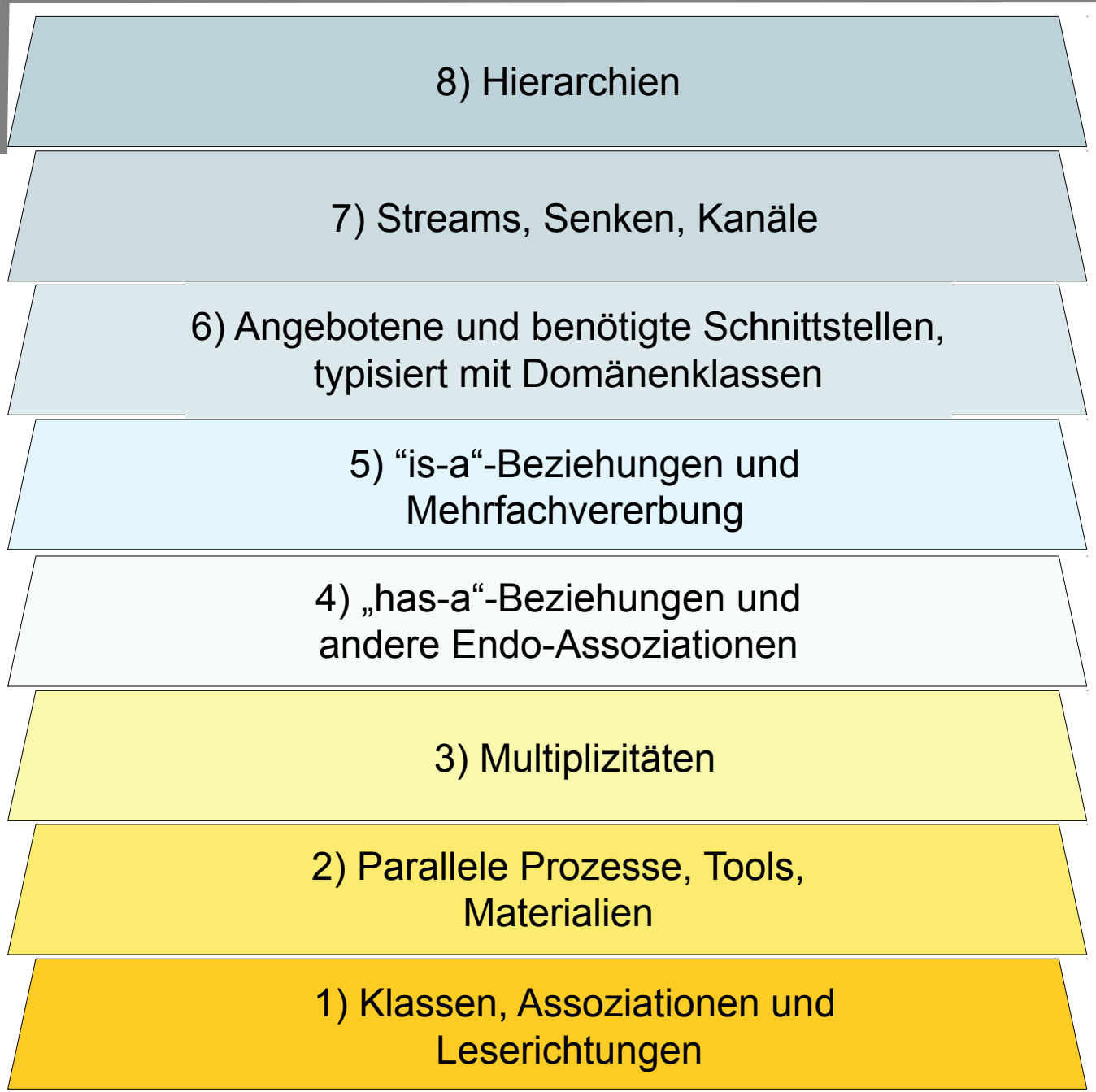
- Idee: Laufe das Metamodell ab und finde Ausprägungen aller Begriffe
- Rot: hier, gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



Schritte der Anreicherung des Analysemodells



Schritte der Anreicherung des Analysemodells



System Analyse

Analyse des Domänenmodells

Tonband des Kunden

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

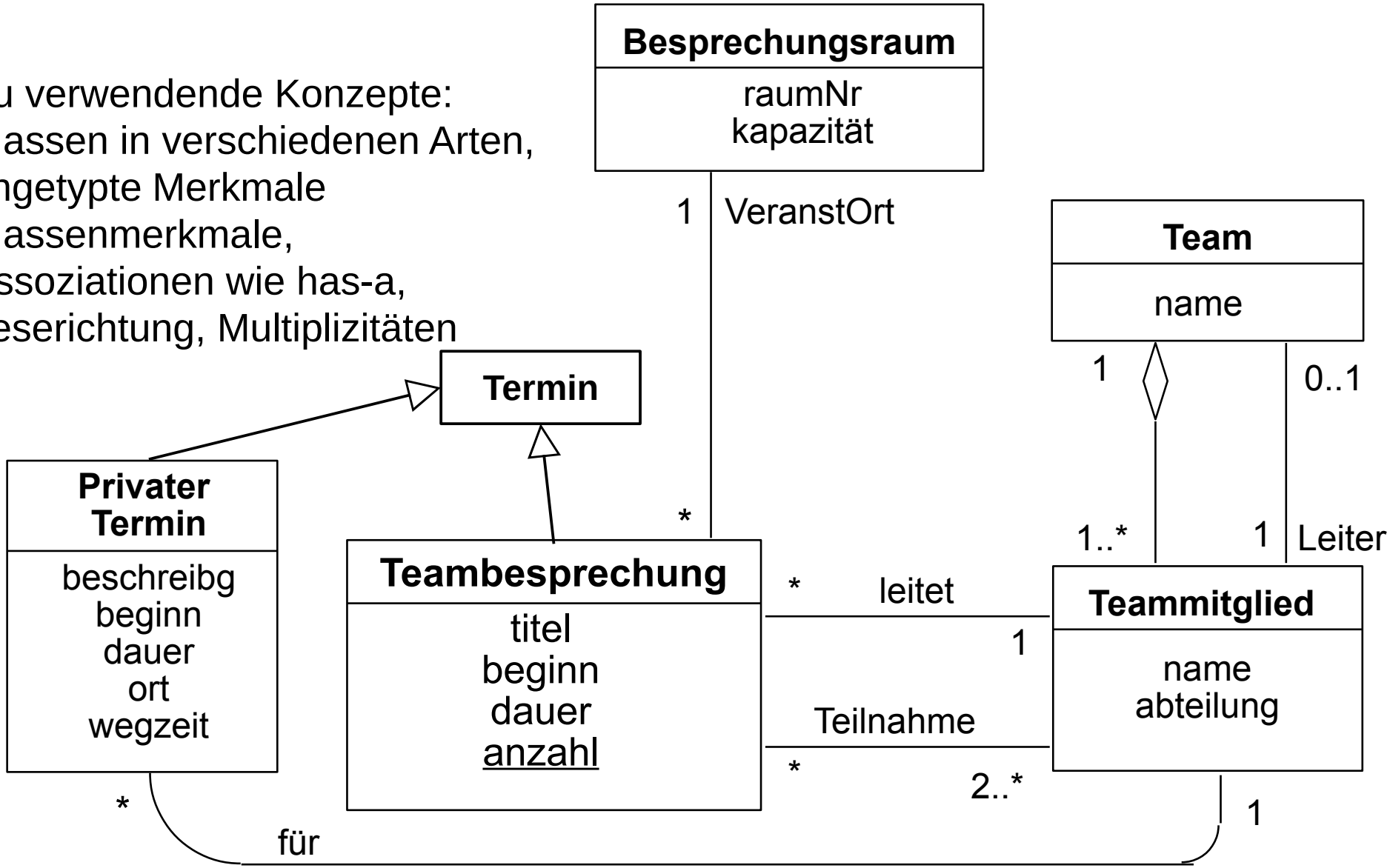
Erstellen Sie mir eine Terminverwaltung.

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

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



Aufgabe der Analyse von Klassen in den einzelnen Schichten des Systems

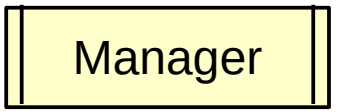
- im **Domänenmodell**:
  - Klassen können sowohl Objekte als Begriffe repräsentieren
  - Vererbung zwischen Begriffen bilden Metamodelle (Begriffshierarchien, Taxonomien)
  - Assoziation, Aggregation und Komposition (Ganz-Teile-Beziehungen)
- in **Kontextmodell und Top-level-Architektur**:
  - Klassen 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

Profile und Stereotypen für Analysemodelle

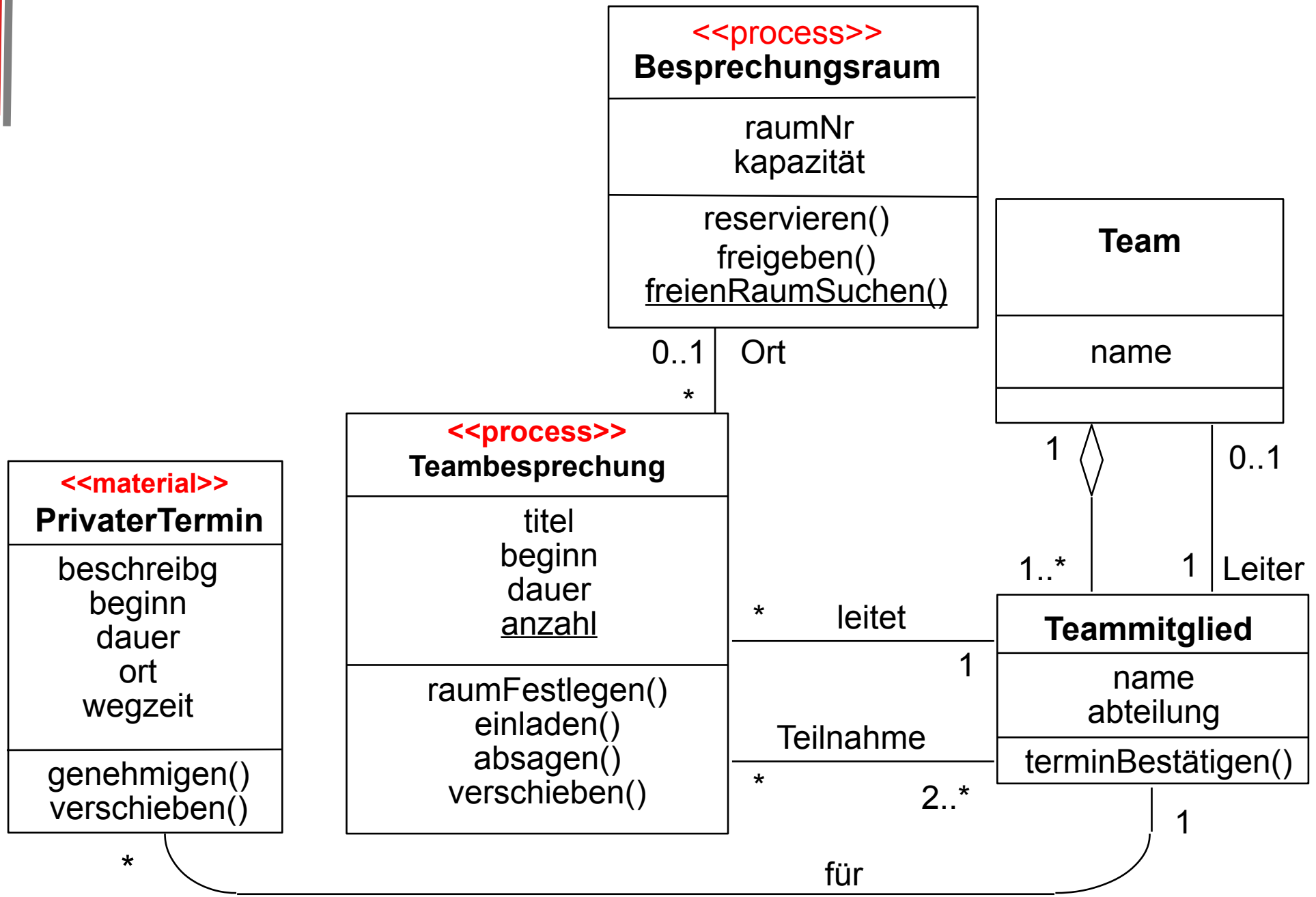
- Für die Spezifikation von Analysemodellen erlaubt es UML, Klassen mit **Stereotypen** zu annotieren
  - Klassen werden “markiert”, klassifiziert, dh. mit mehr Semantik ausgestattet
  - Komponenten, Ports, Ströme können unterschieden werden
- Kataloge von Stereotypen heissen **Profile**
  - Wenn der 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 <<actor>> <<process>> <<active class>>
  - Werkzeuge (Tools): Aktives Objekt, dass Materialien schreibt) <<tool>>
  - Workflow (interaktiver Prozess): Prozess, der Tools aufruft <<workflow>>
- **Passive Objekte** (Materials, entities, data objects)
  - Aktive Objekte arbeiten auf Materialien <<material>>
  - Persistente Materialien <<data>>
  - Rollen: Dynamische Sicht auf ein Material <<role>>
- **Kanäle (channels, pipes)**: Beschreiben, wie aktive Objekte miteinander kommunizieren
  - Über Kanäle fliessen Daten, sie werden mit Senken geschrieben und mit Streams gelesen



Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell

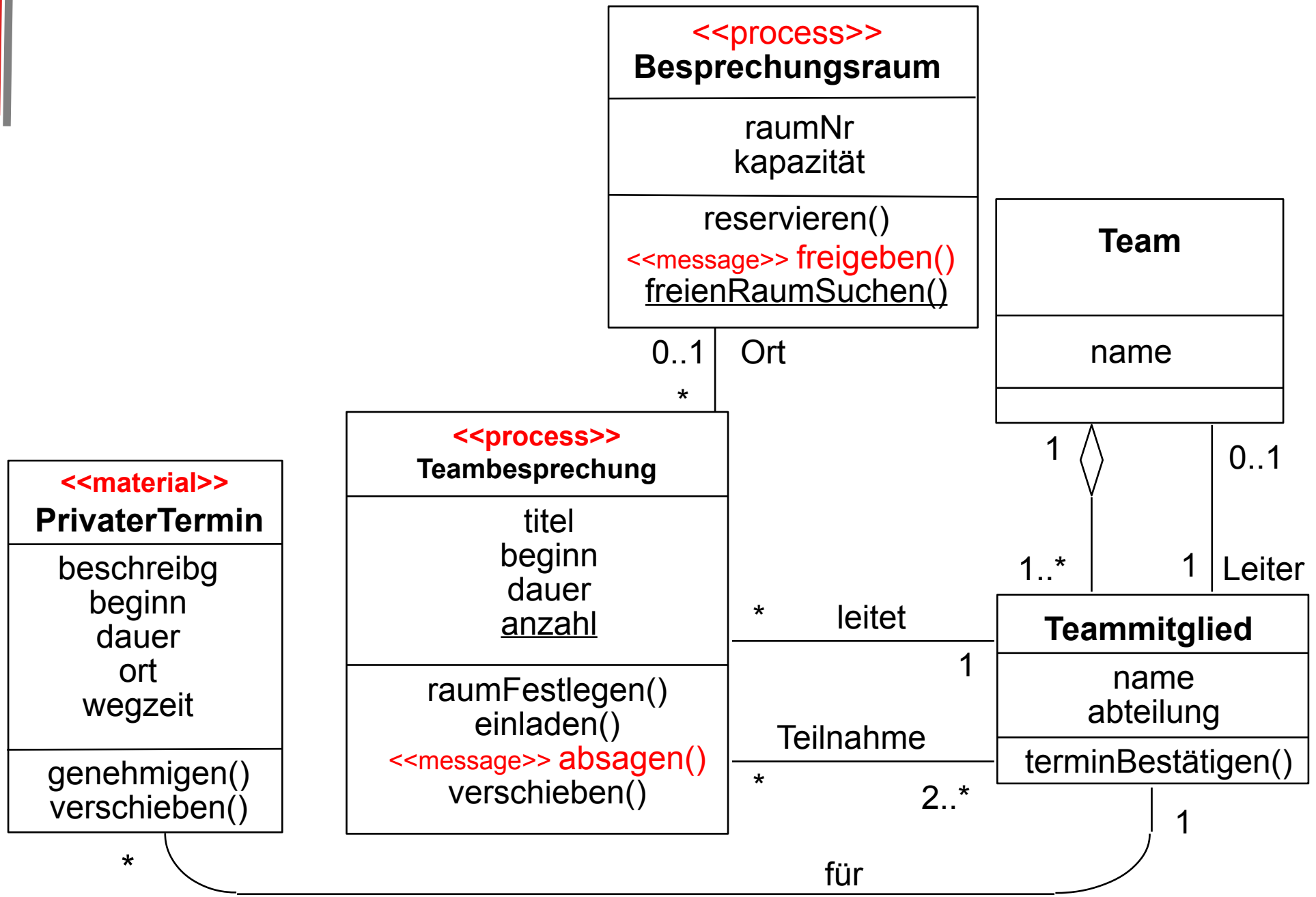


Prof. U. Aßmann, Softwaretechnologie, TU Dresden

Dienstanfragen in sequentiellen und parallelen OO Sprachen

- 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)

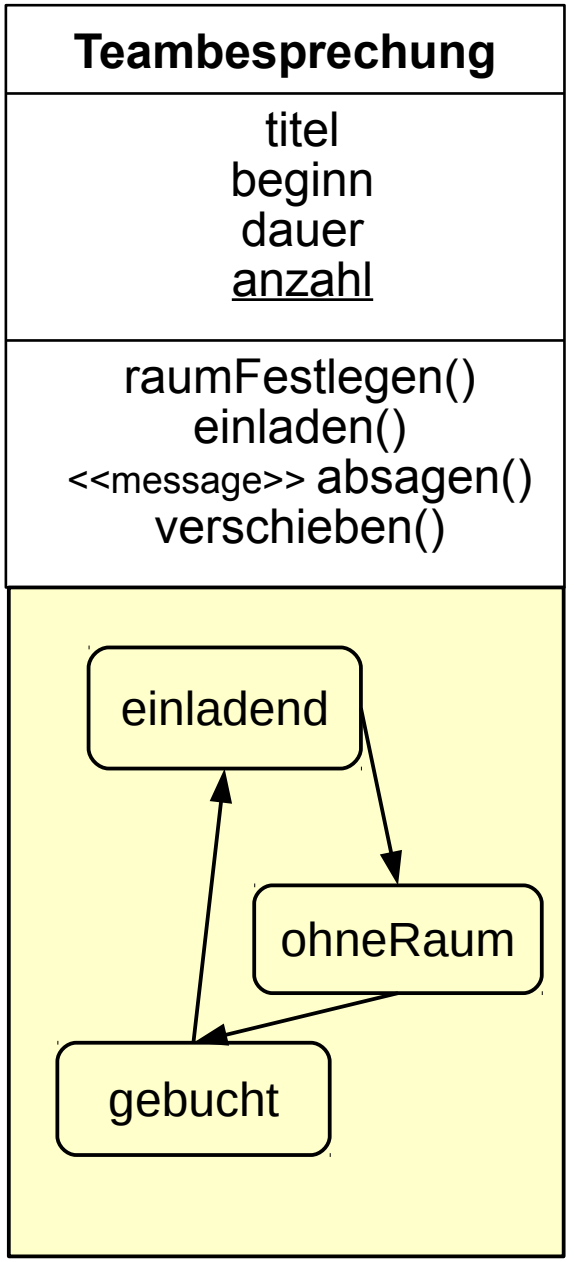
Beispiel: Parallelität und verschiedene Arten von Operationen im Analysemodell



Prof. U. Aßmann, Softwaretechnologie, TU Dresden

Abteile (Compartments)

- 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)

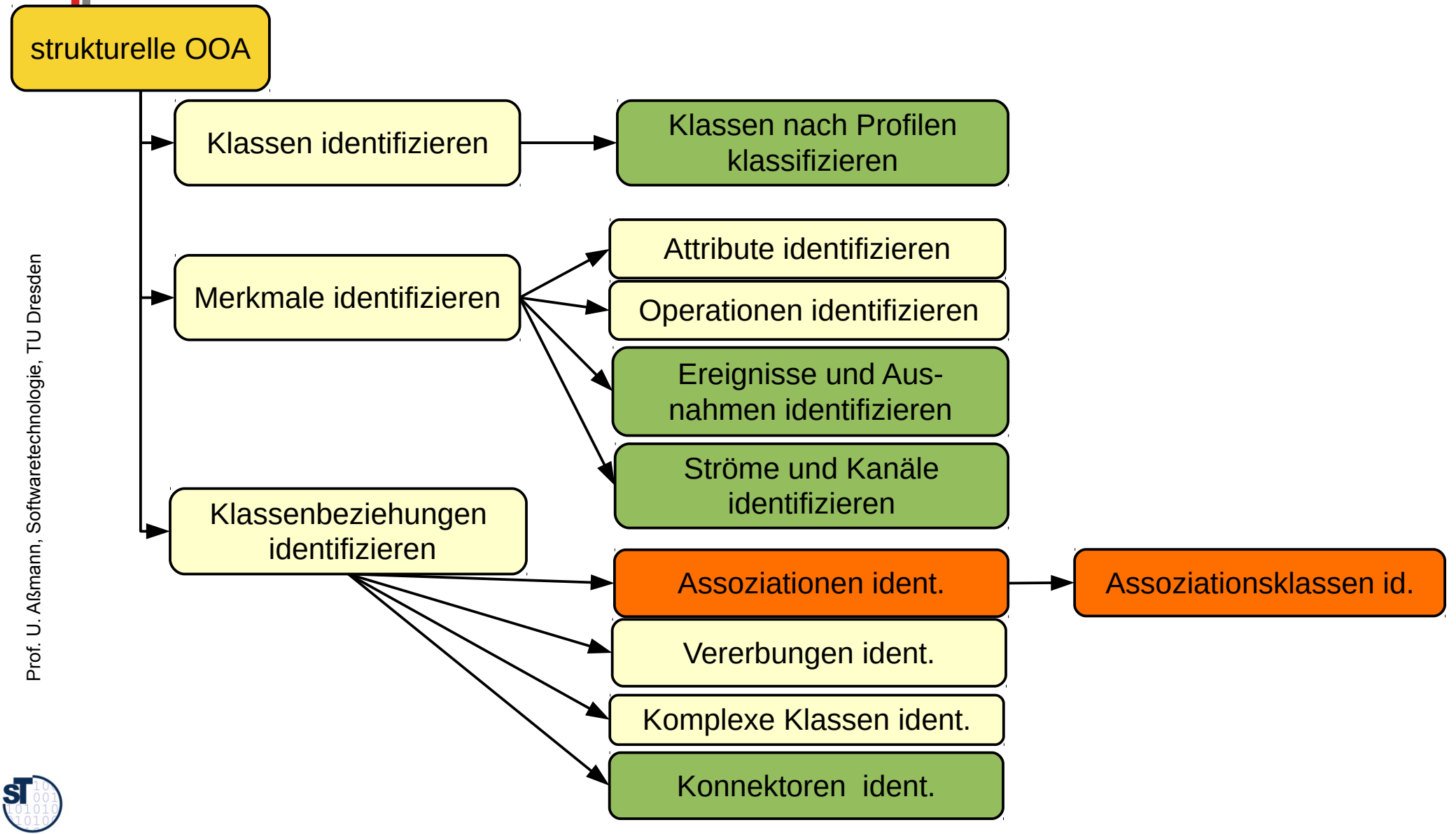


31.3 Analyse von Klassenbeziehungen und Leserichtungen

- Texte des Kunden werden in Diagramme umgesetzt und wieder vorgelesen

Schritte der strukturellen, metamodellgetriebenen Analyse

- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur

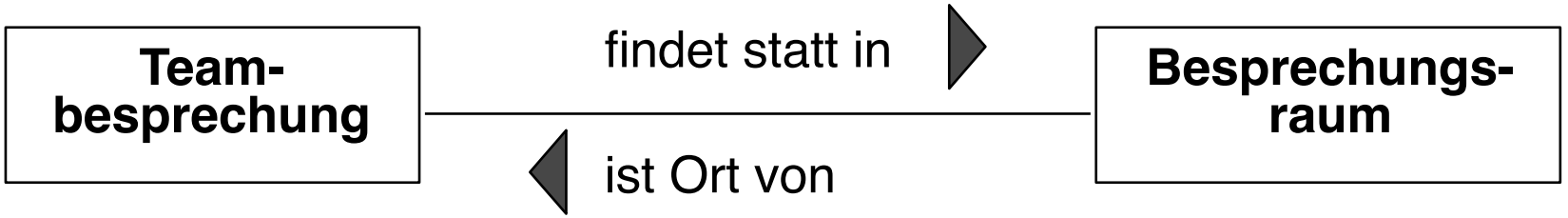


Analyse von Assoziationen (Klassenbeziehungen)

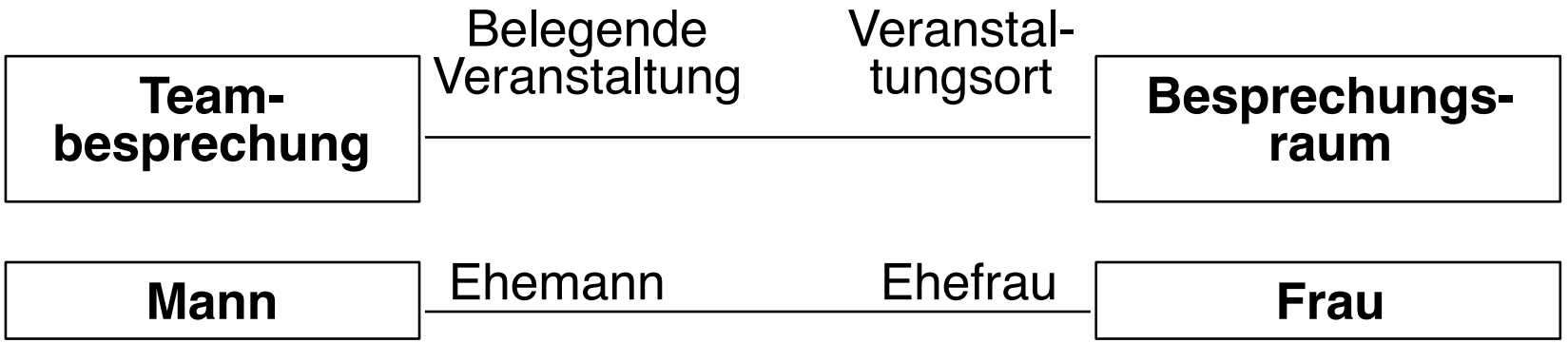
- Assoziationen entstehen, wenn *Objekte sich kennen*
  - Sie stellen Relationen, Graphen oder Hypergraphen dar, d.h. bilden Abstraktionen von Referenzen
- **Definition:** Eine (binäre) **Assoziation (Beziehung, relationship)** AS zwischen zwei Klassen *K1* und *K2* beschreibt, daß die Instanzen der 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

Leserichtung und Assoziationsenden

- Für Assoziationsnamen kann die **Leserichtung** angegeben werden.

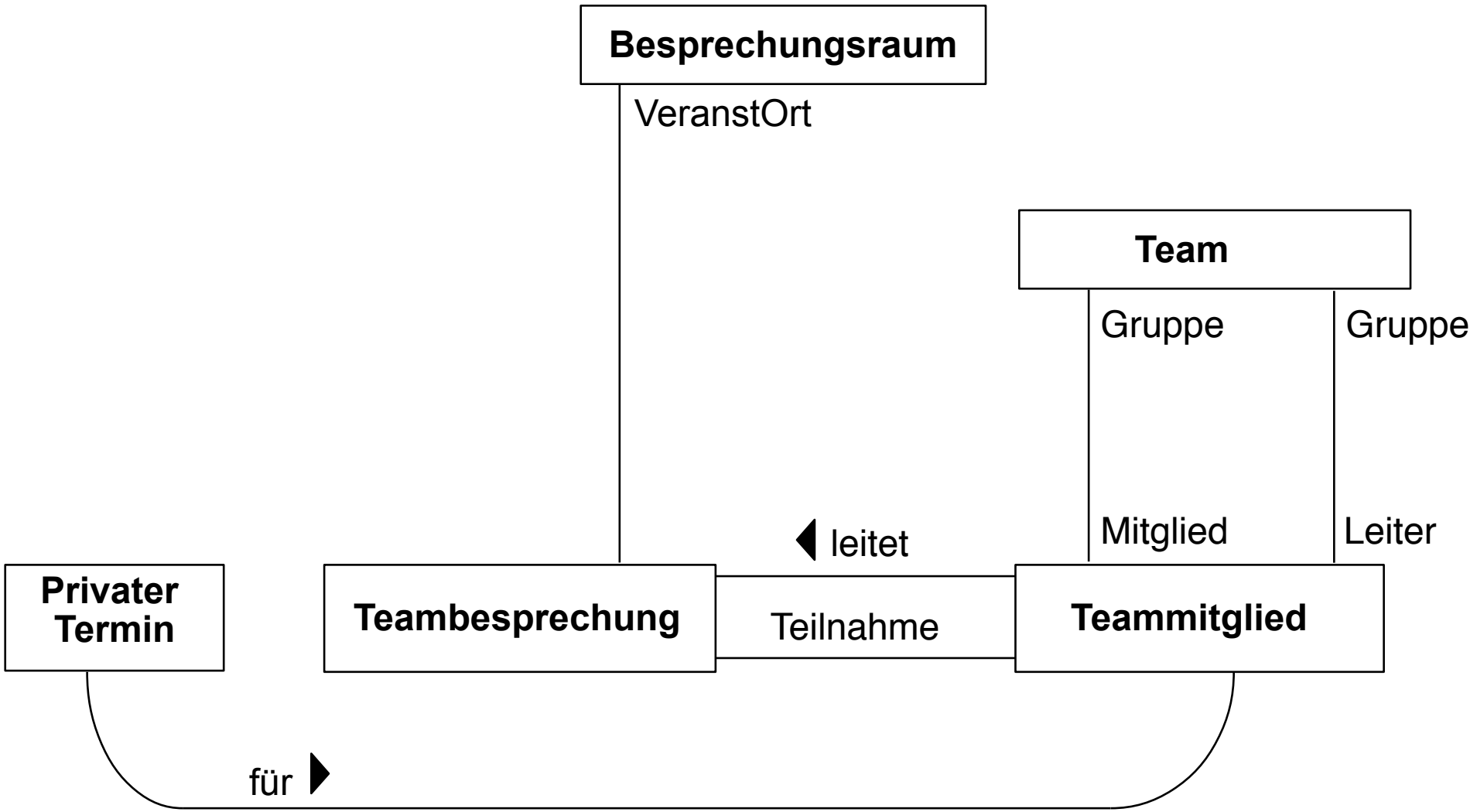


- Ein Name für ein **Assoziationsende** bezeichnet die Assoziation aus der Sicht einer der teilnehmenden Klassen.
  - Ein Assoziationende beschreibt die ***Rolle*** einer Klasse in einer Assoziation



Beispiel: Assoziationen

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



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)

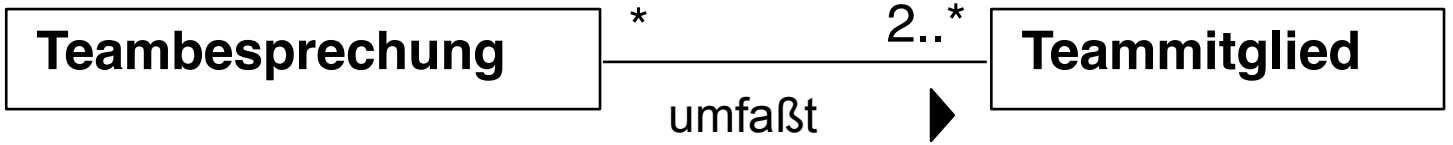
- **Notation:**

Multiplizität *Mult*:

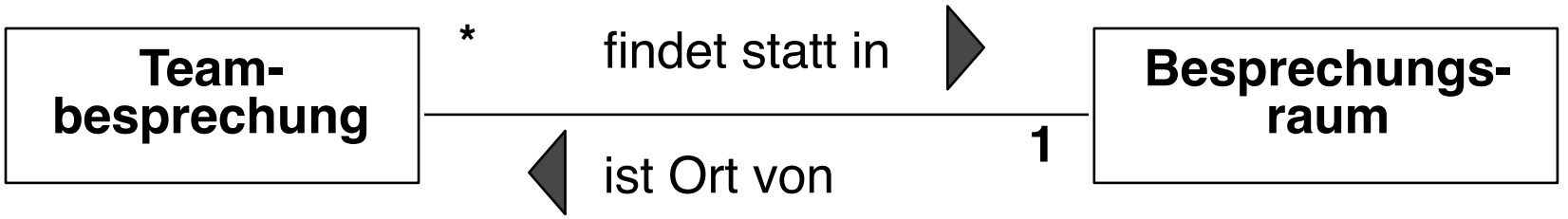
n	(genau *n* Objekte der Klasse *K2*)
n..m	(*n* bis *m* Objekte der Klasse *K2*)
n1, n2	(*n1* oder *n2* Objekte der Klasse *K2*)

Zulässig für *n* und *m* :

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



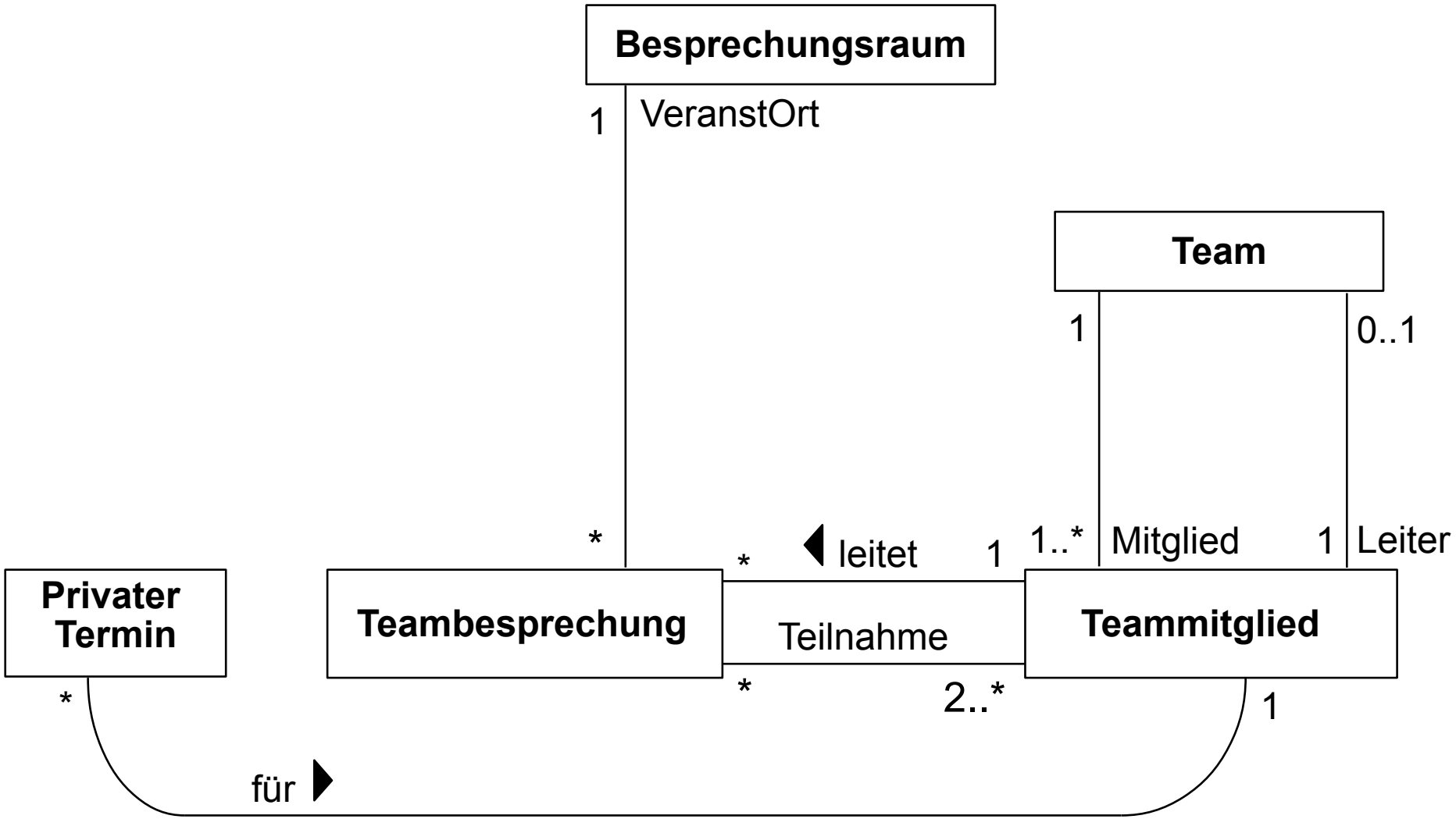
Multiplizitäten bestimmen durch "Vorlesen"



- 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.

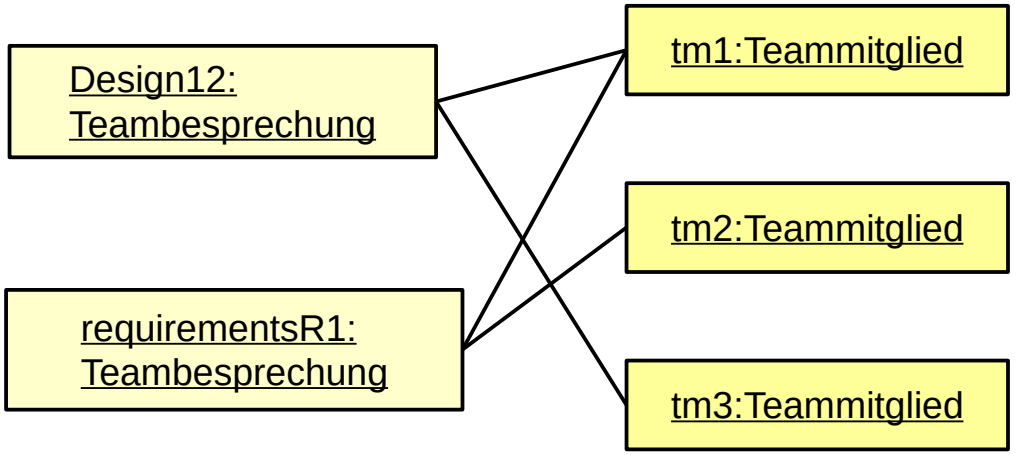
Beispiel: Multiplizitäten



Semantik (bidirektionaler) Assoziationen

- Ein **Extent** 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

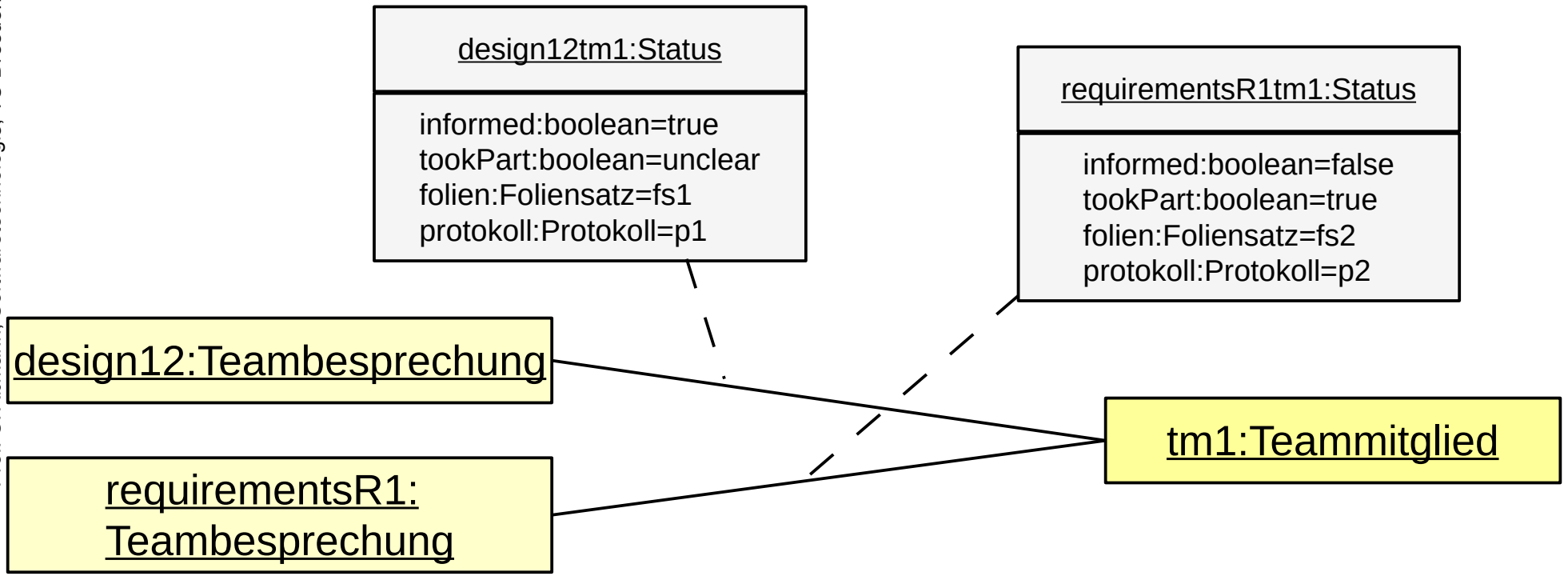
  Teammitglied-Objekte in Teilnahme-Assoziation: {tm1, tm3}

  Objekt tm1:Teammitglied

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

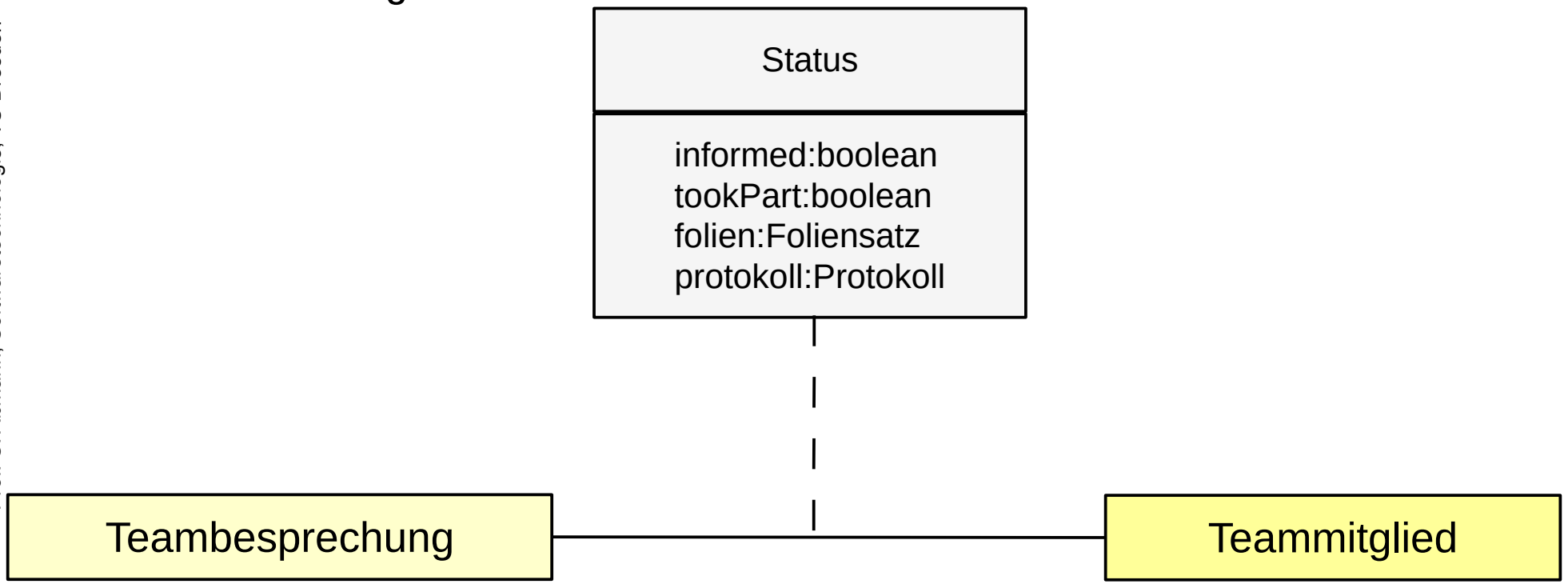
Assoziationsattribute

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

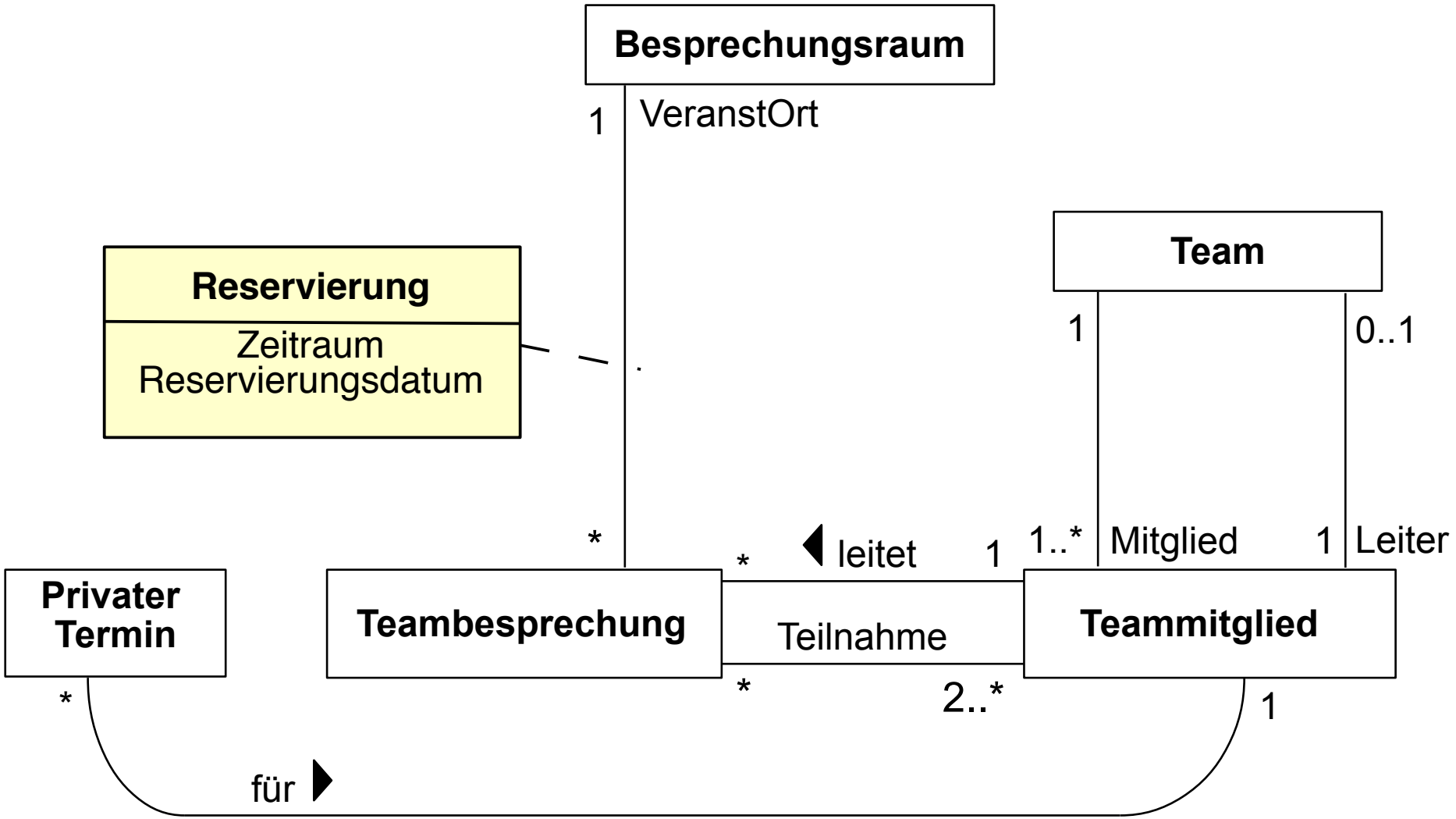


Assoziationsklassen

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



Beispiel: Reservierung als Assoziationsklasse



31.3.2 Realisierung von Assoziationen

- Wir nutzen jetzt das in Abschnitt II Gelernte, um die Abbildung von UML-Assoziationen auf Java zu verstehen

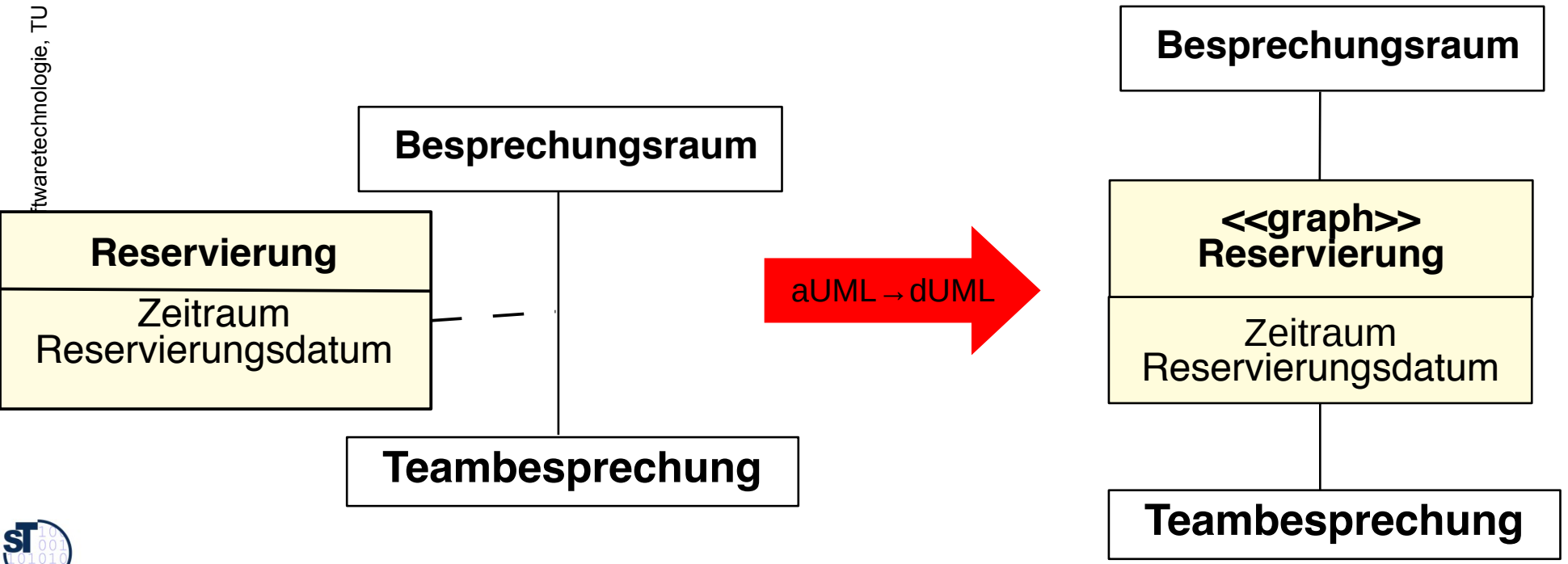
Realisierung von Assoziationen

- 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 von Assoziationen durch explizite **Rollenklassen** (Anhang)
- 6) Realisierung durch **Tabellen** in einer relationalen Datenbank

- **Genaue Entscheidung erst im Entwurf !**

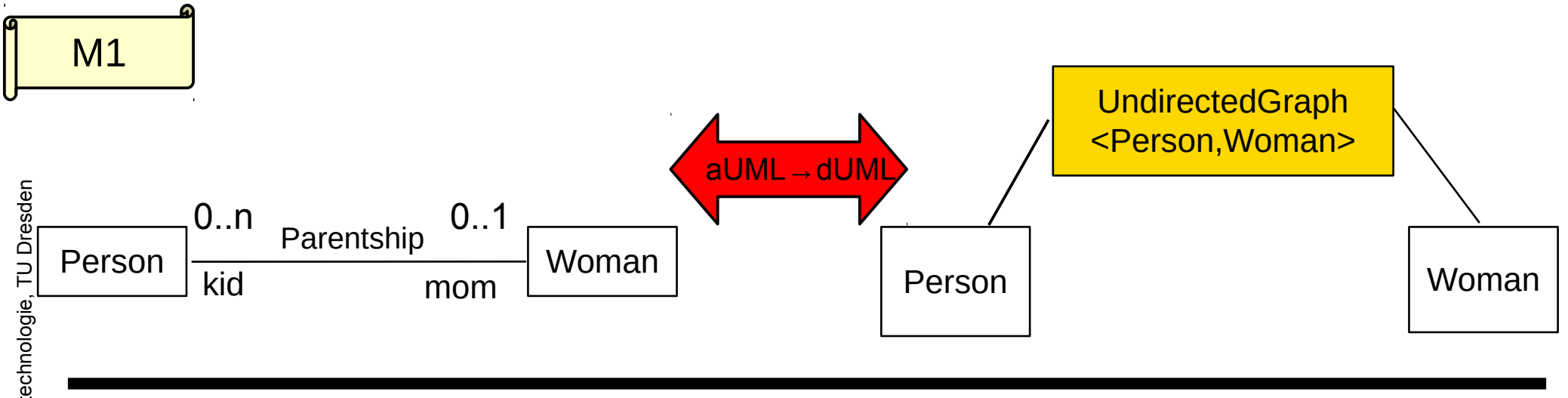
31.3.2.1 Realisierung von bidirektionalen Assoziationen durch Graphklassen

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

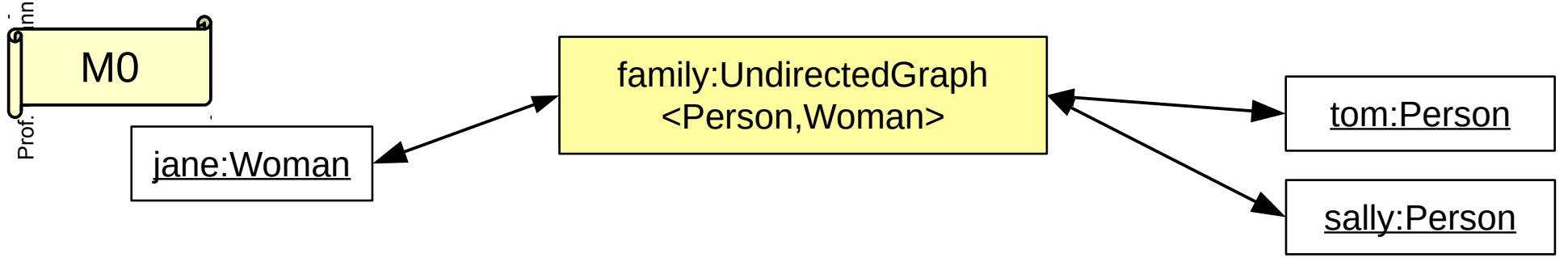


Beispiel: Realisierung von bidirektionalen Assoziationen durch Graphklassen

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

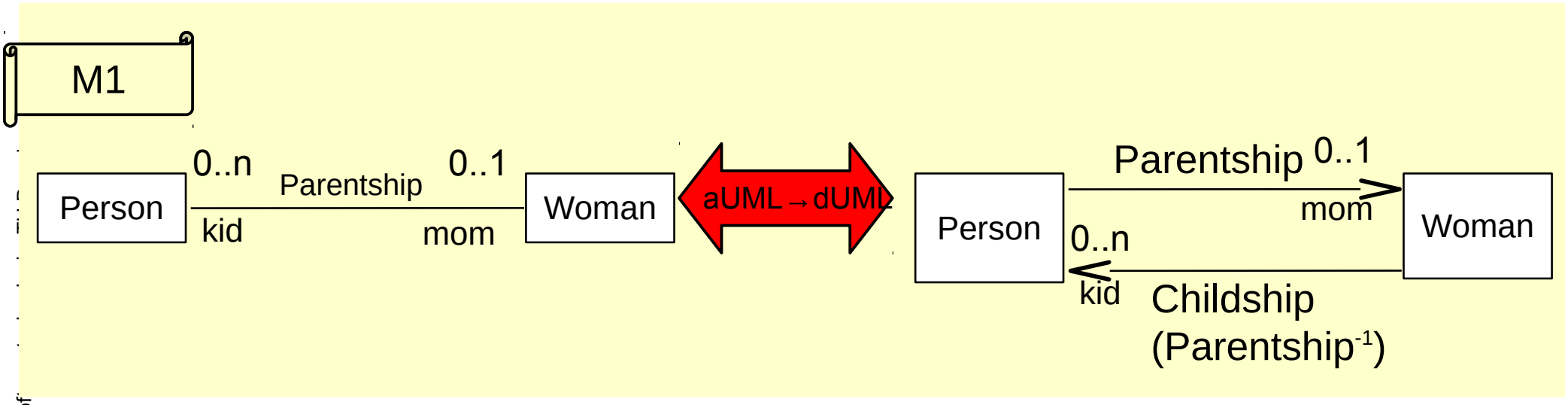


Laufzeit: zwei Referenzen zwischen Graphobjekt und den assoziierten Objekten

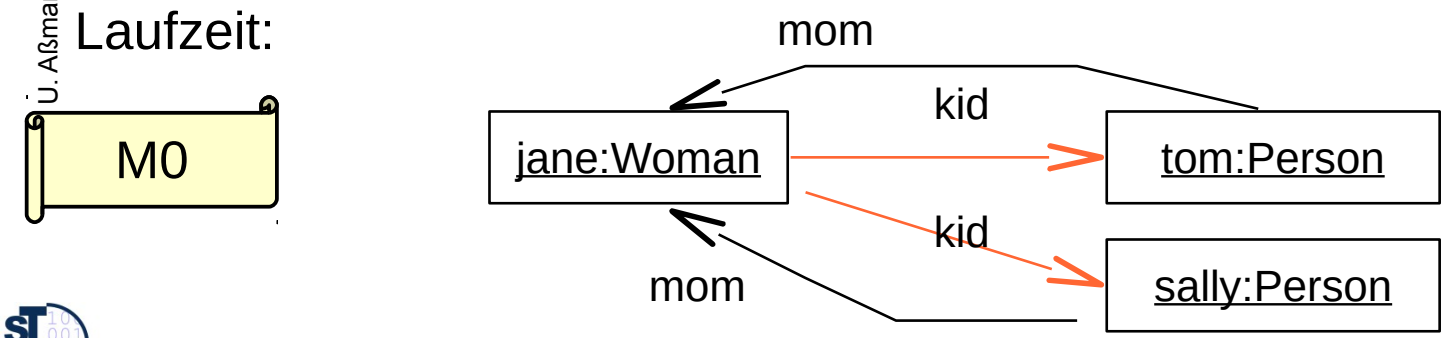


31.3.2.2 Realisierung von bidirektionalen Assoziationen durch unidirektionale

- 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:

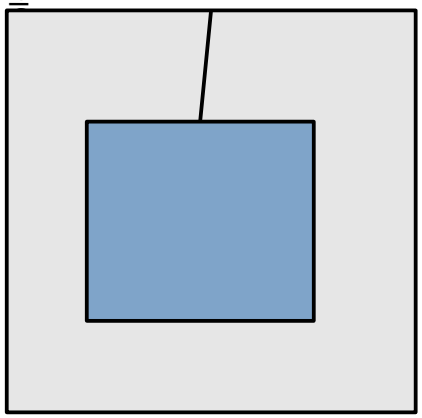


31.4 Modellierung von Hierarchien und komplexen Objekten (Teilehierarchien)

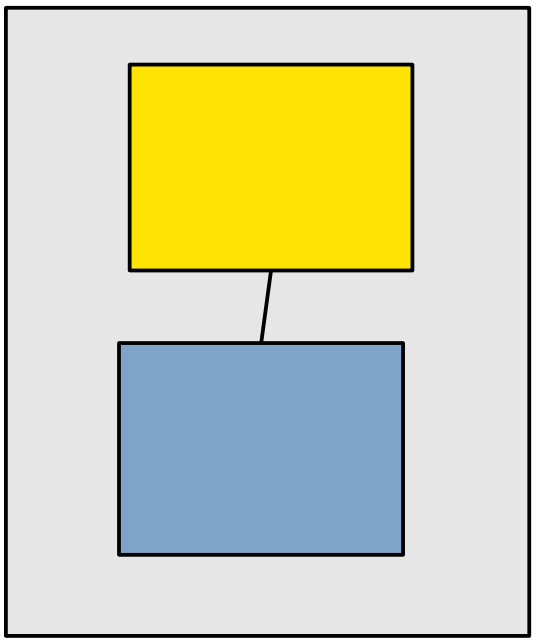
Endo- vs. Exo-Assoziation

- Komplexe Objekte verwenden Assoziationen innerhalb 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

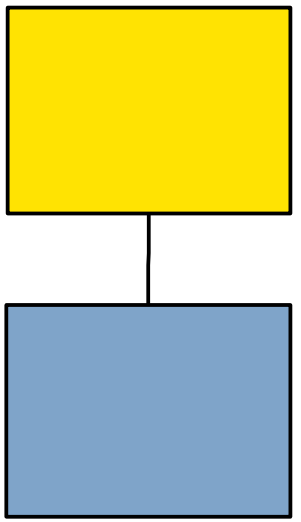
Satellit
Integrates-a



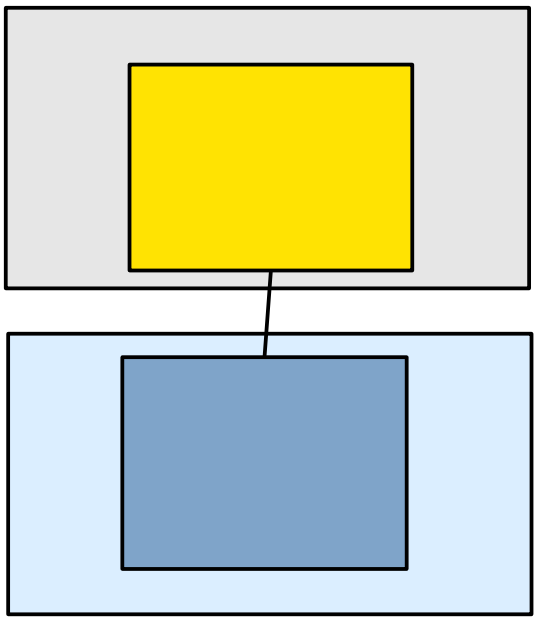
Endo-Intra



Exo



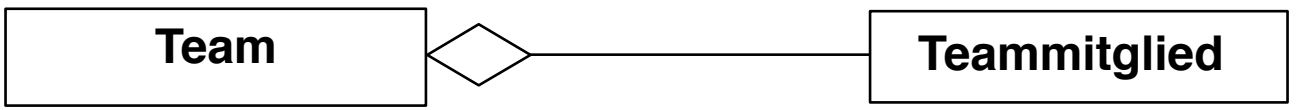
Inter



31.4.1 Aggregation, Komposition, Rollenspiel als Endo-Assoziationen

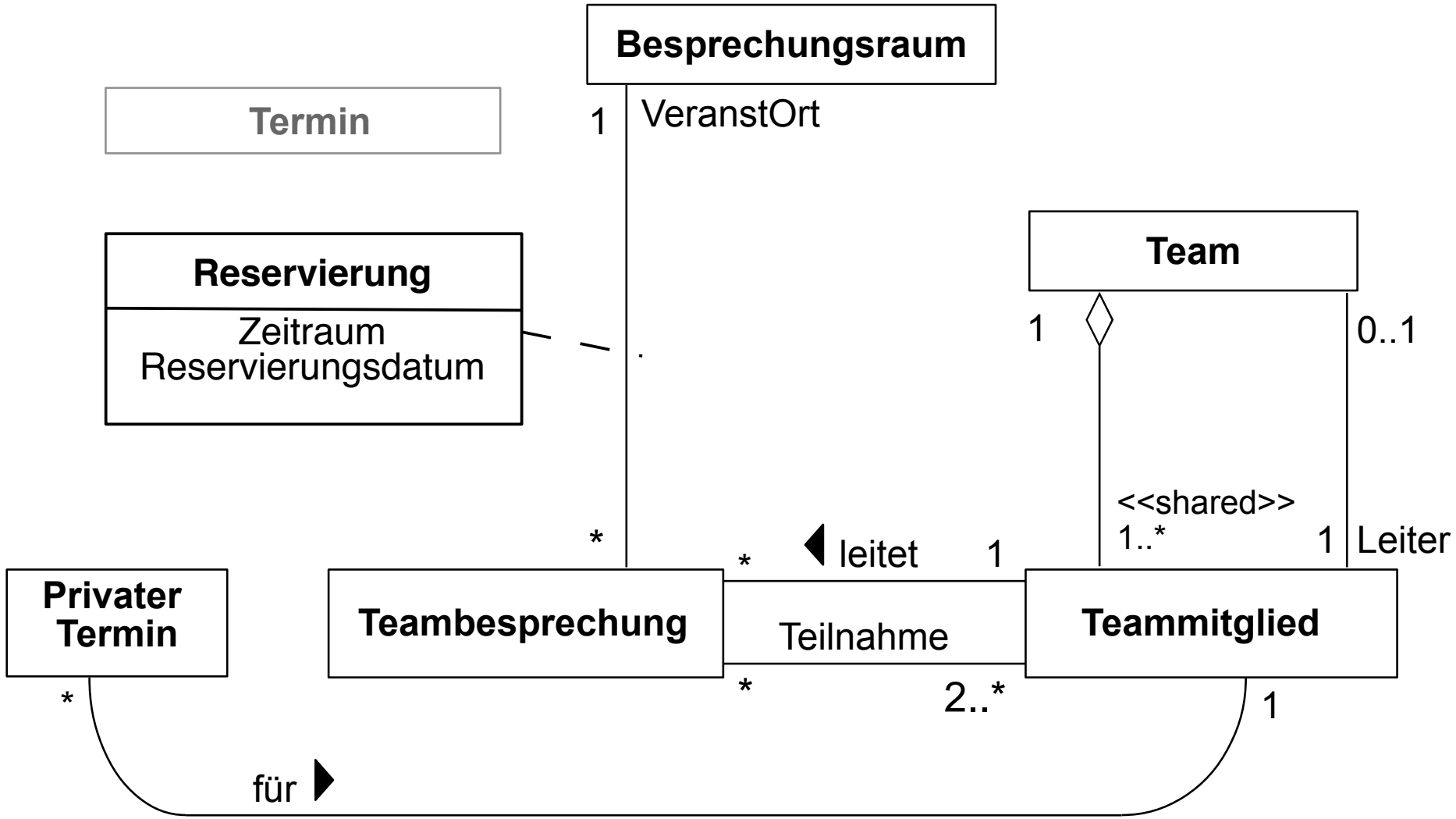
Aggregation (has-a)

- **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.*
  - Sie schließt die Teile mit dem Ganzen zusammen, ist also eine **Endo-Assoziation.**
  - Die auftretenden Aggregationen bilden auf den Objekten immer eine transitive, antisymmetrische Relation (einen gerichteten zyklenfreien Graphen, *dag*).
  - Ein Teil kann zu mehreren Ganzen gehören (<<*shared*>>), zu einem Ganzen (<<*owns-a*>>) und exklusiv zu einem Ganzen (<<*exclusively-owns-a*>>)



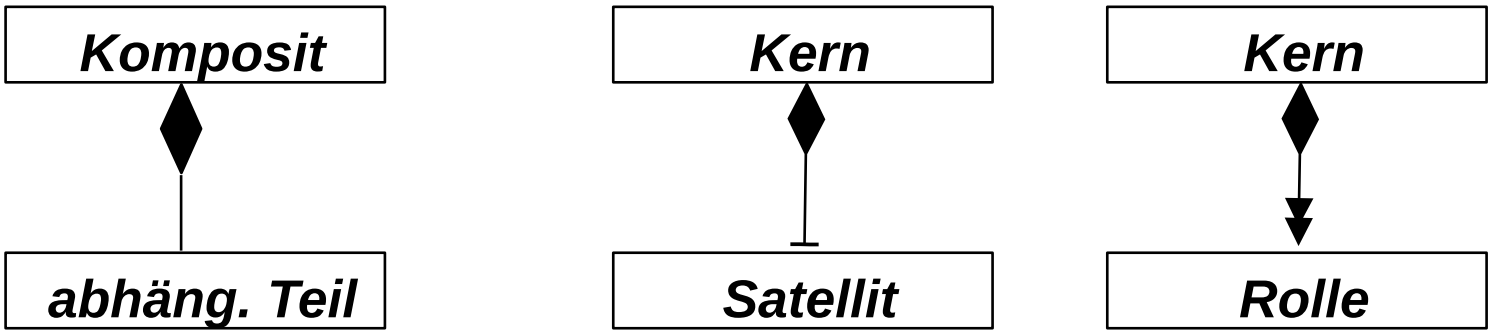
Lies: „Team hat ein Teammitglied“

Beispiel: Assoziationen und Aggregationen

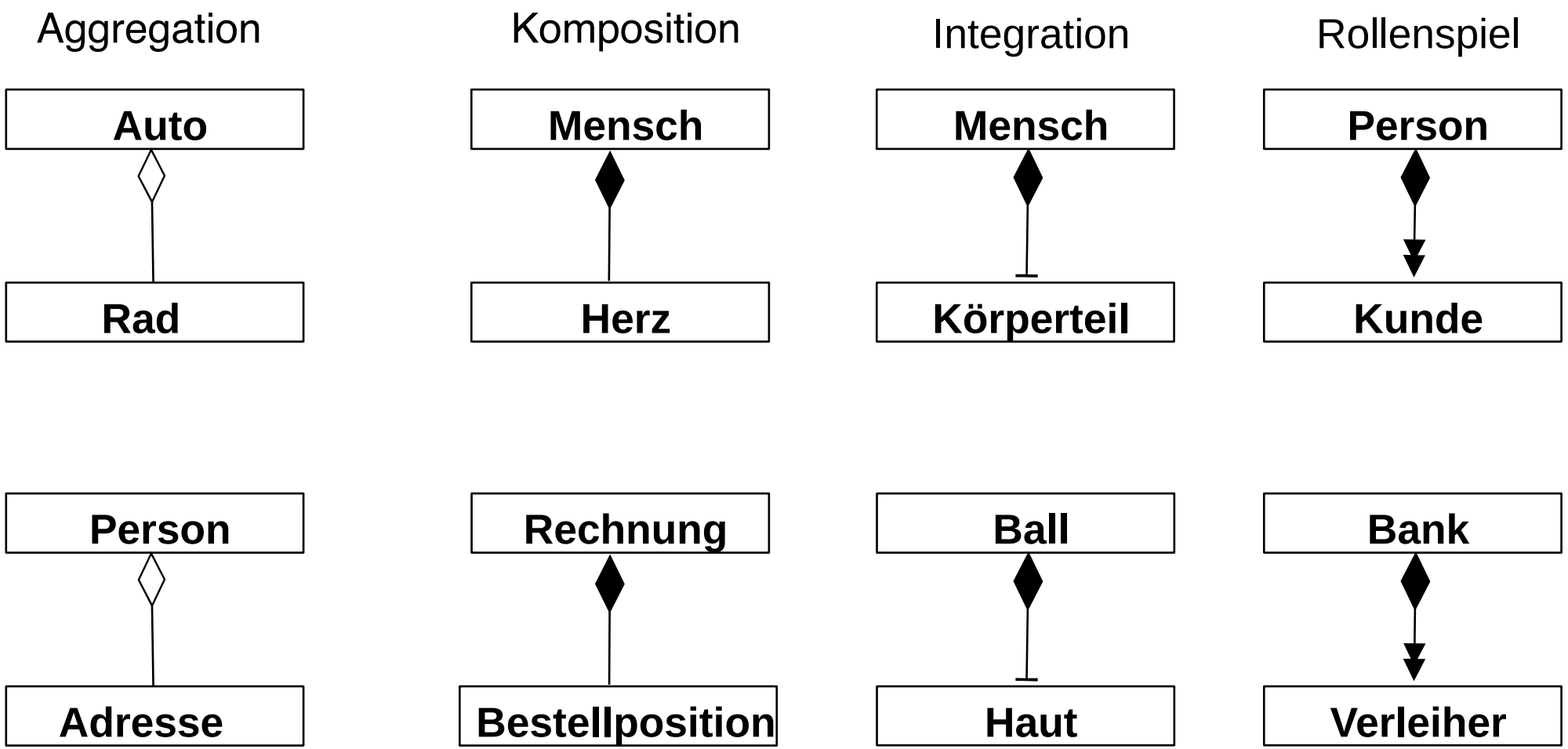


Komposition (owns-a)

- **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



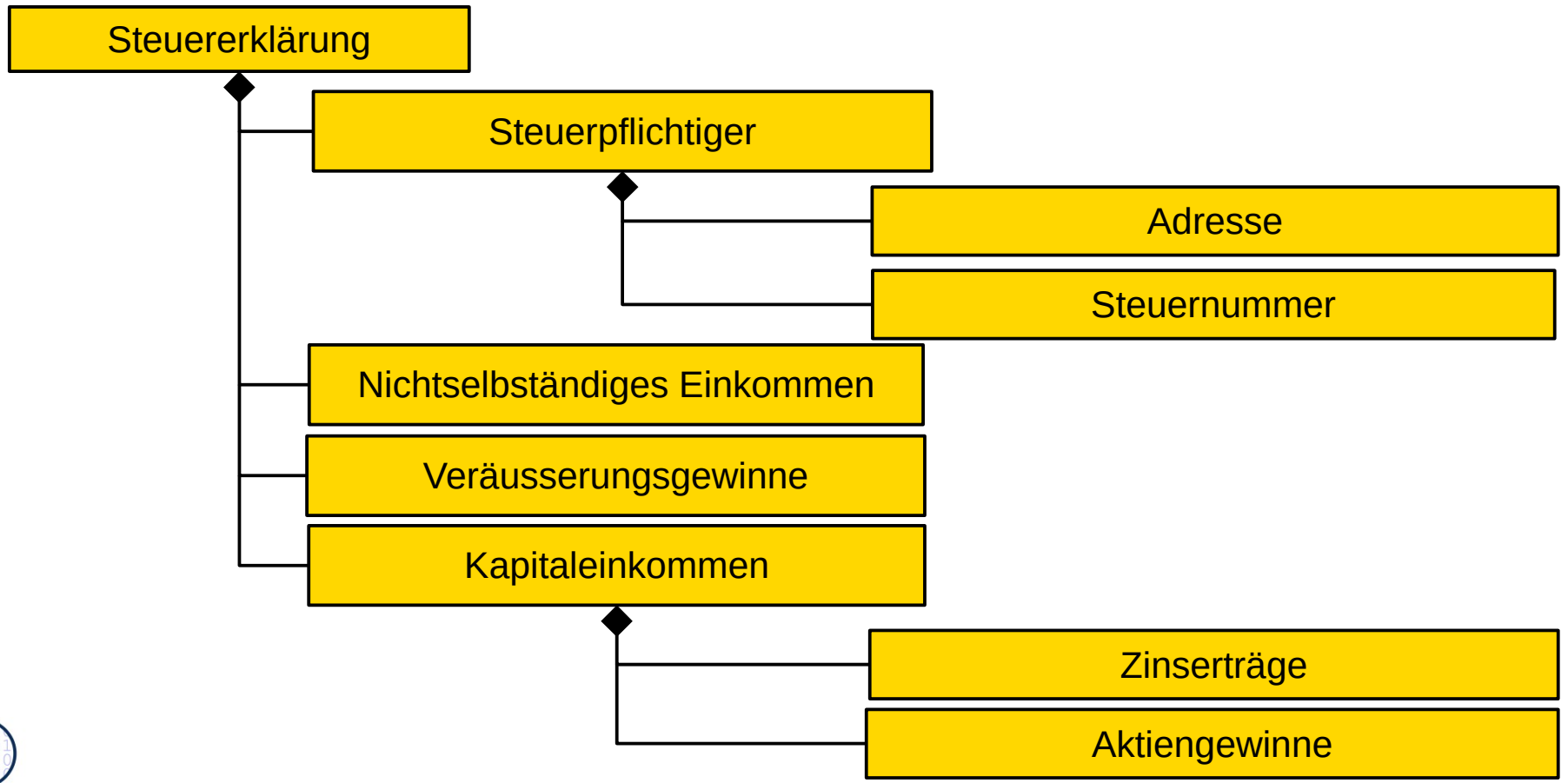
Darstellung von Hierarchien und kompositen Objekten

- 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)

- 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

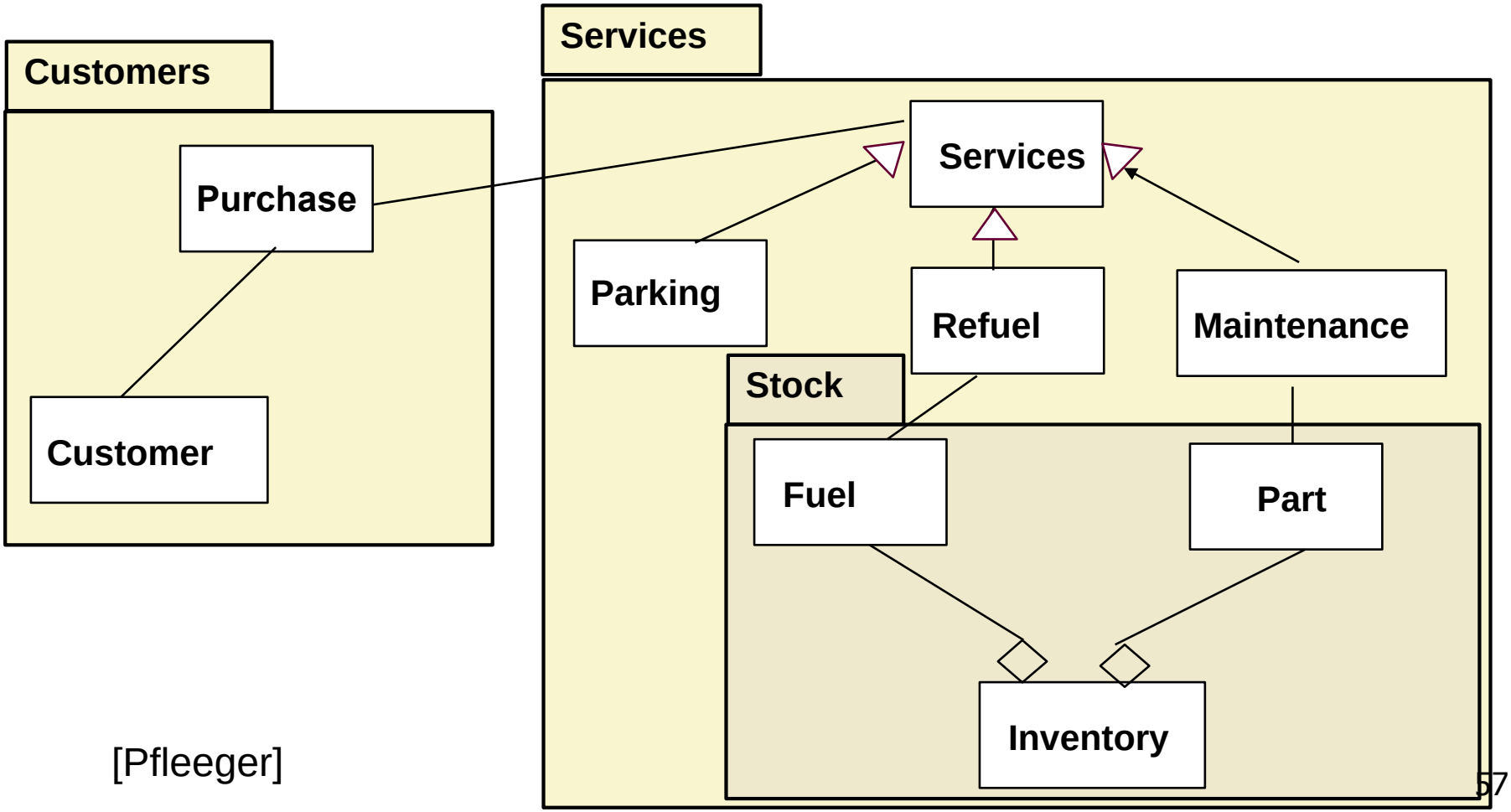


Frage

- Wie realisiert man eine Aggregationshierarchie?

Bsp: Pakete und ihre Aggregationsrelationen in UML

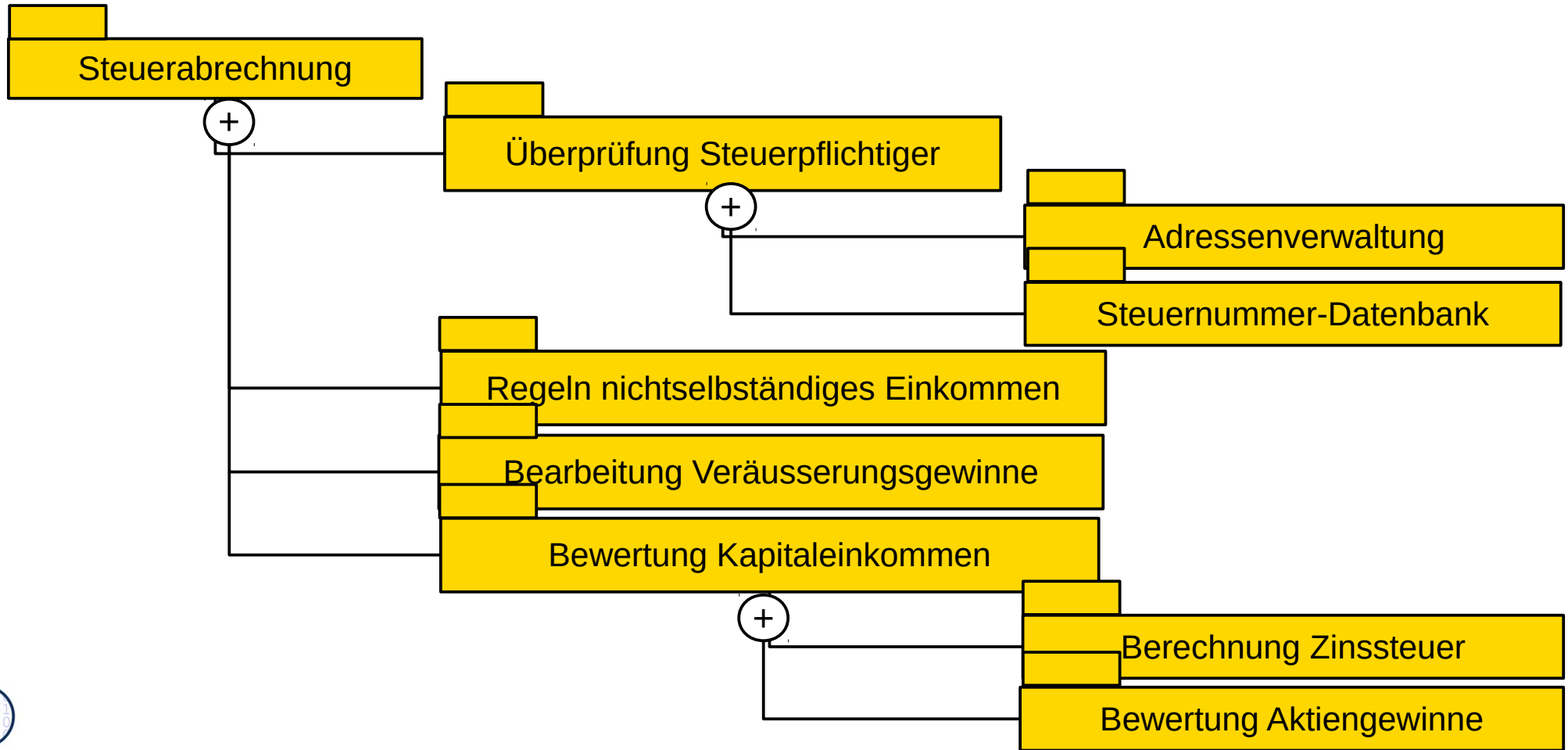
- Klassen sind *flach,* Pakete dagegen *hierarchisch* strukturierbar
  - Pakete gruppieren Klassen, Objekte, andere UML-Fragmente
- UML-Paketrelationen sind hierarchisch oder azyklisch
  - Sind einfach auf Paketkonzepte von Programmiersprachen abbildbar



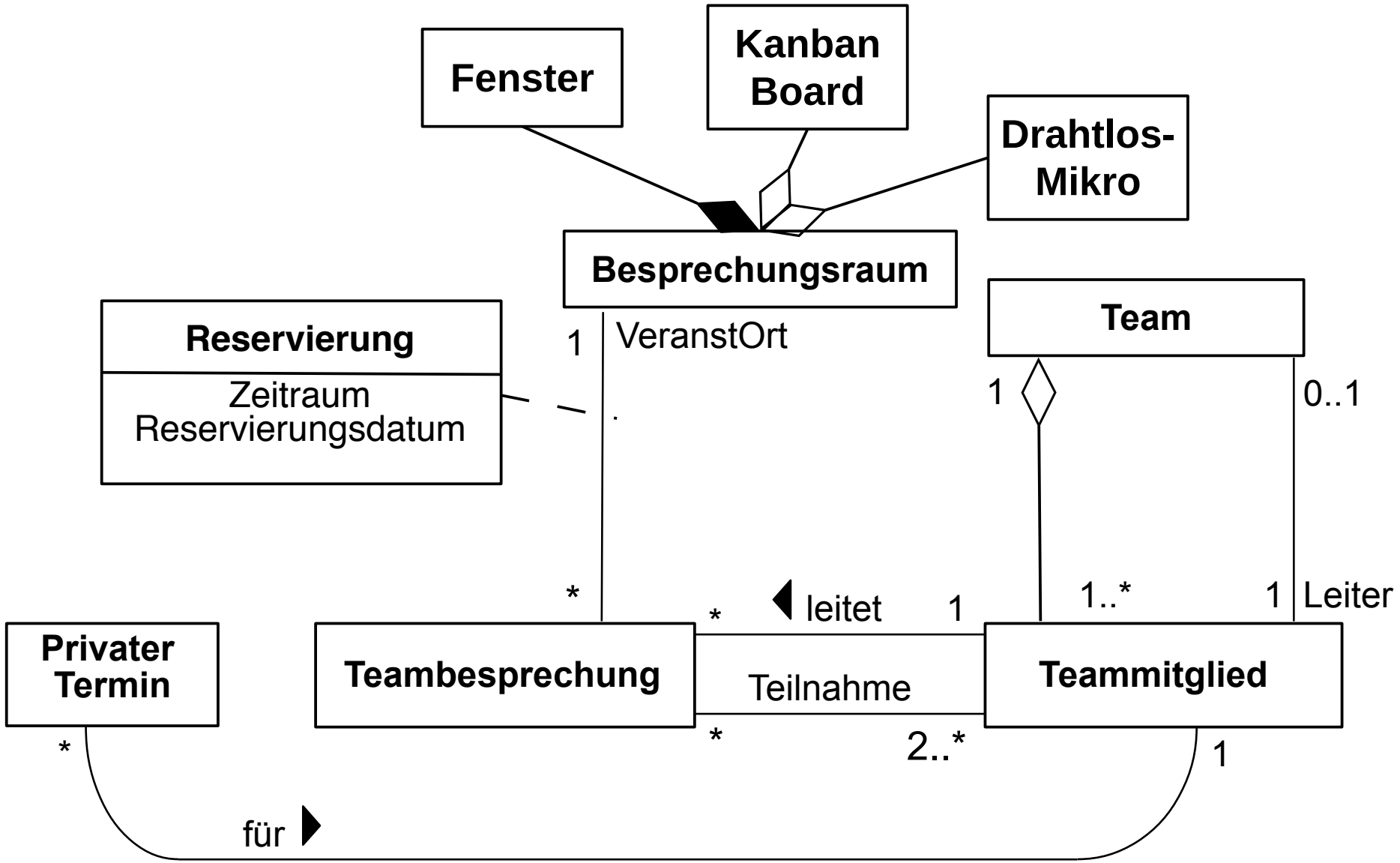
[Pfleeger]

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)

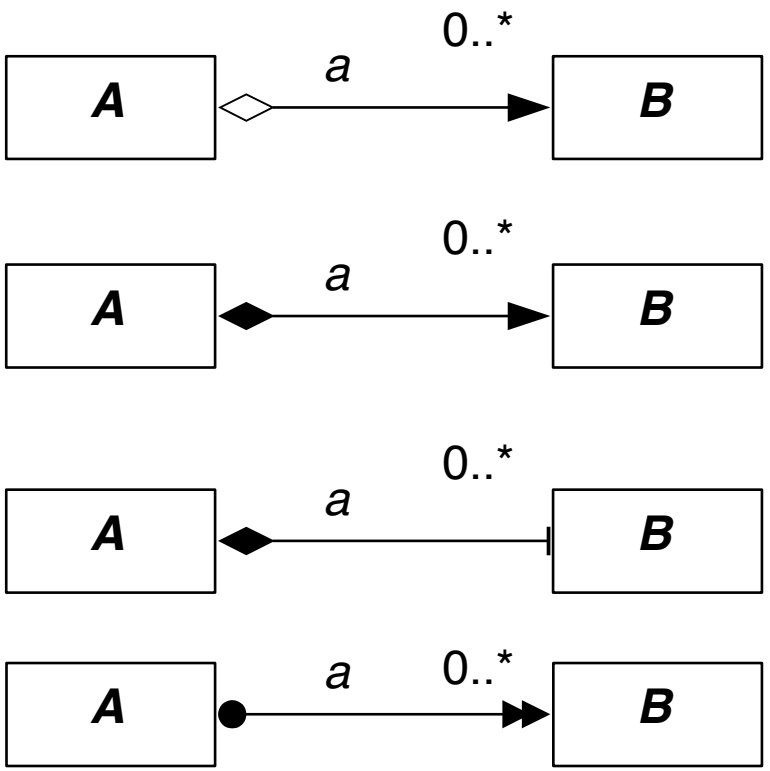


Beispiel: Assoziationen und Aggregationen



Realisierung von Aggregation und Komposition 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 des Entwurfsmusters Composite




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

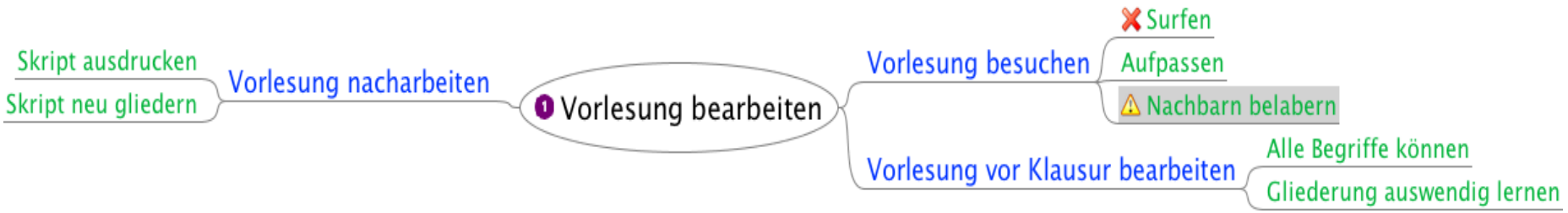
Exkurs E31.2: Lernen und Hierarchien

- **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)

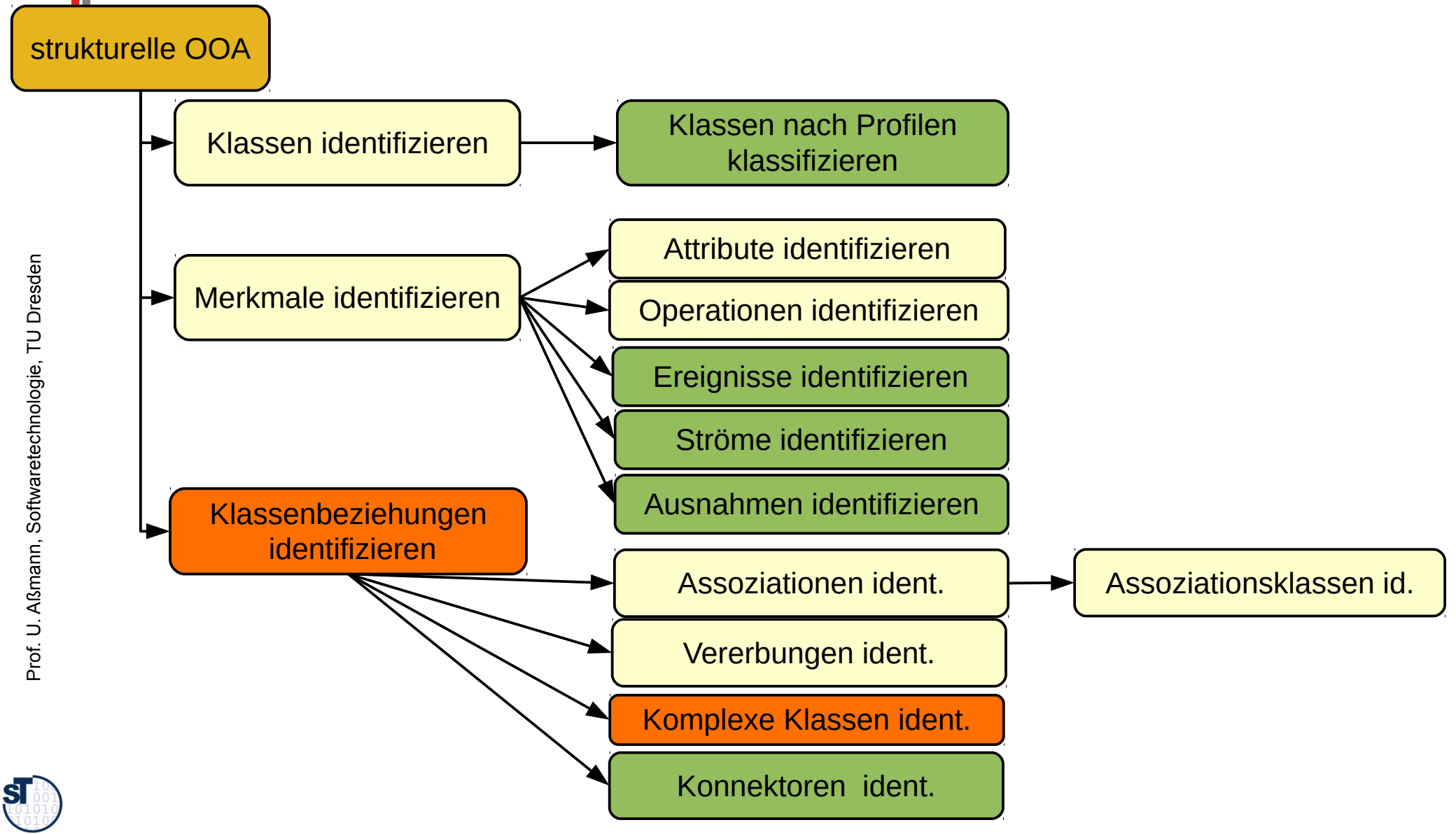
- **Mindmaps** (**Wissenskarten**) sind Begriffshierarchien, die von der Mitte des Blattes her *monozentral* gezeichnet werden (“mapping your mind”)
- Sie können als Startpunkt beim Lernen eingesetzt werden
  - Untersuchen Sie, ob Sie mit Zeilenhierarchien oder Mindmaps besser zurecht kommen
  - Aufgabe: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



31.4.2 Modellierung von komplexen Objekten aus Kernen und Unterobjekten

Schritte der Analyse

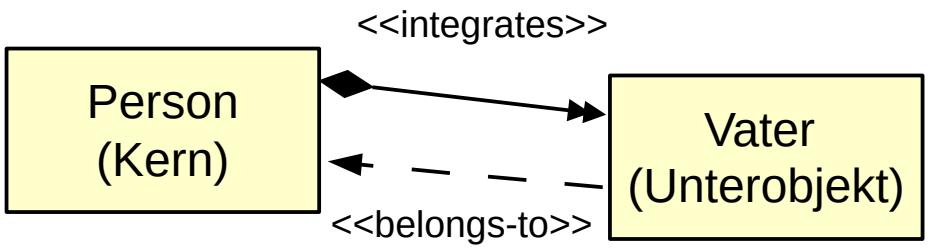
- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



Ein **komplexes Objekt (Subjekt, 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)
    - repräsentiert eine Eigenschaft bzw. einen Teilzustand des komplexen Objekts
  - Als **Integrationsrelation** <<integrates>> wird eine Endo-Relation verwendet (spezielle Fälle sind Aggregation, Komposition, Rollenspiel)



Verwendung der Hierarchie-Notationen in der Analyse

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

- Komplexe Objekte sind immer *hierarchisch* oder *azyklisch*, bilden also immer Hierarchien oder gerichtete azyklische Graphen:
  - Komplexe Objekte als Kern mit **Satelliten**, die in den Kern mit integrates-a integriert sind
  - Hierarchische Objekte in **Baum-Notation** in einem **hierarchischen** komplexen Objekt (Kernobjekt als Wurzel)
  - Geschichtete Objekte in **Dag-Notation** in einem **geschichteten** komplexen Objekt (Kernobjekt als Wurzel eines gerichteten azyklischen Graphen, directed acyclic graph, dag)

Arten von komplexen Objekten (Analyse- und Entwurfsobjekten)

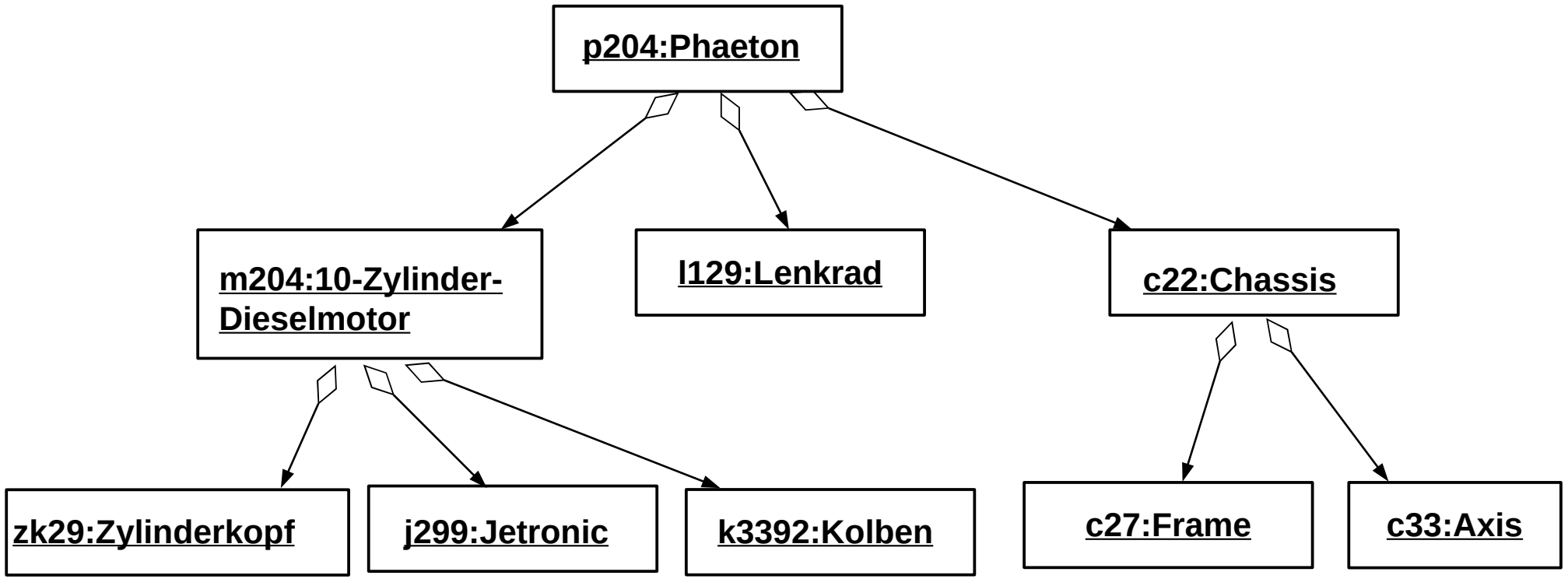
- Komplexe Objekte kommen in allen Phasen vor
  - Ihr **Lebenszyklus** wird durch eine Zustandsmaschine beschrieben (Kap 33)
  - Interpretierer-Funktion: Sie empfängt Befehle wie Create(), Open(), Read(), Write(), Do(), Enter()
  - Steuerungsfunktion: Sie sendet weitere Befehle aus (Steuerungsmaschine)
- **Analyseobjekte** stammen oft von **Domänenobjekten** her
  - **Simuliertes Objekt**: kommt in der Umgebung des Programms vor (Simulation der Umwelt)
    - 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:
  - 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
    - Komposition ist wichtig im Domänenmodell und Datenhaltung!

Beispiel: Komposite Material-Objekte als Stücklisten

- Produktionsplanungssysteme (PPS) verwalten Produkte als Materialien
  - Die Teile eines Produkts werden in *Stücklisten* (eigentlich *Stückbäume*) verwaltet
  - Stückliste eines Phaeton (alle Teile sind lebenslang numeriert, um verfolgbar zu sein)



Arten der Verfeinerung des Analysemodells

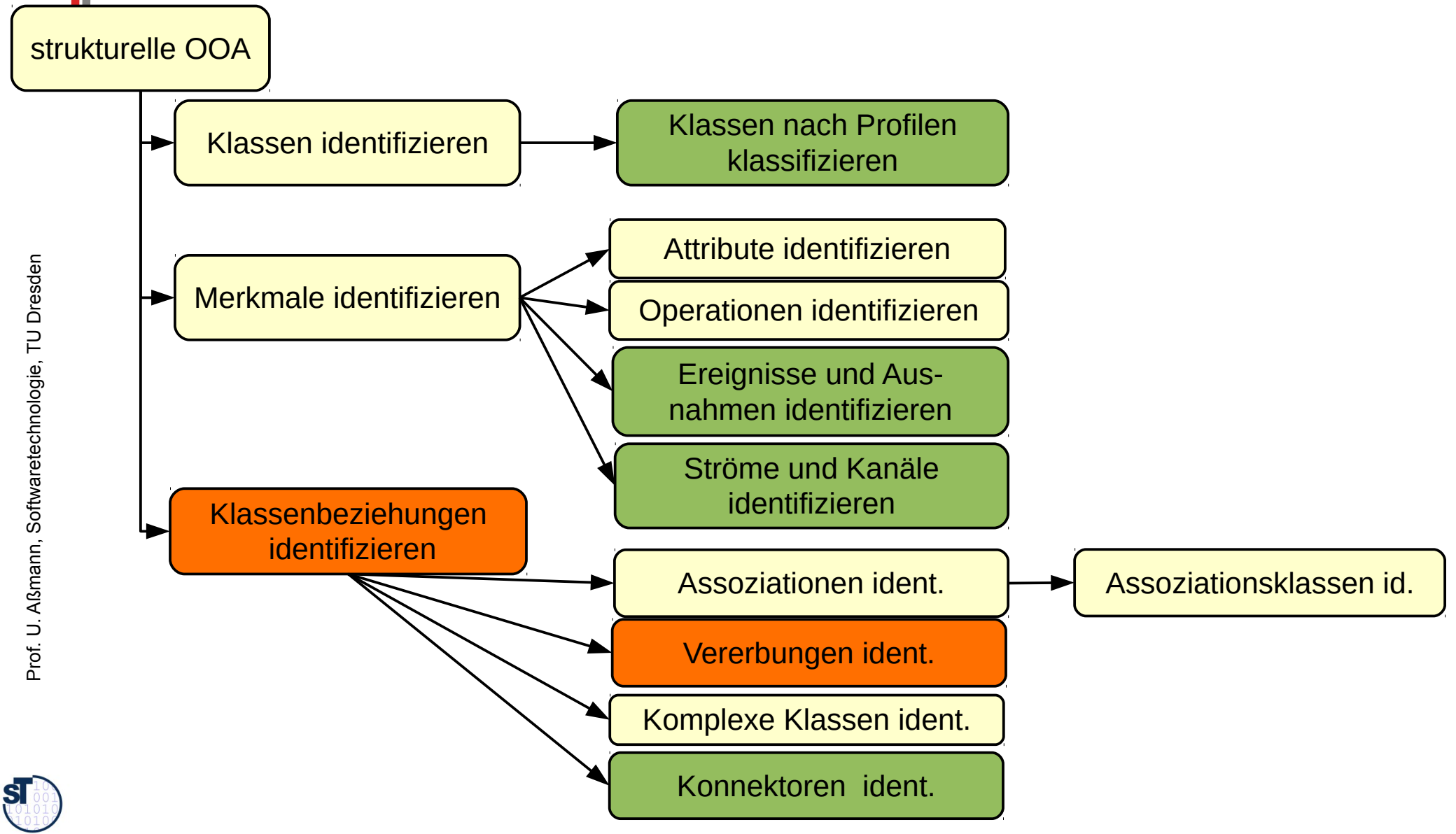
Auf dem Weg vom Analysemodell über das Entwurfsmodell zum Implementierungsmodell und Code werden wir folgende Arten von Verfeinerungsmethoden kennenlernen:

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 Objektanreicherung (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

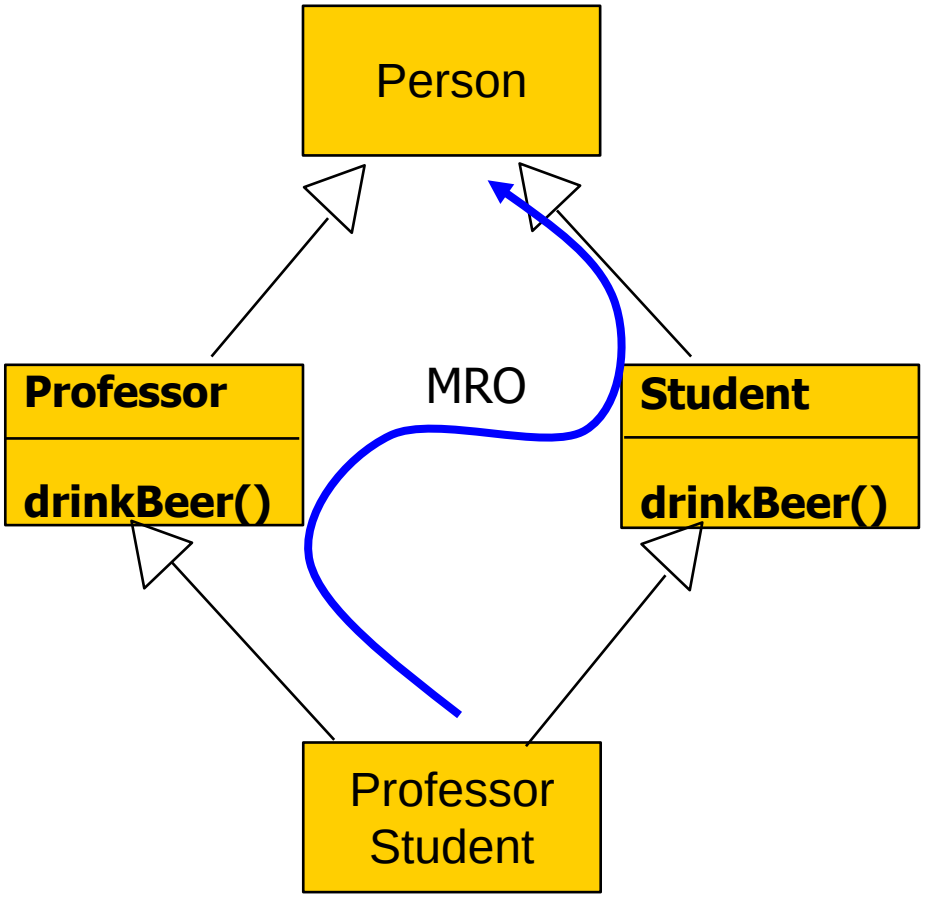
Schritte der strukturellen, metamodellgetriebenen Analyse

- gelb: Domänenmodell; grün: Kontextmodell, TL-Architektur



Mehrfachvererbung (Multiple Inheritance)

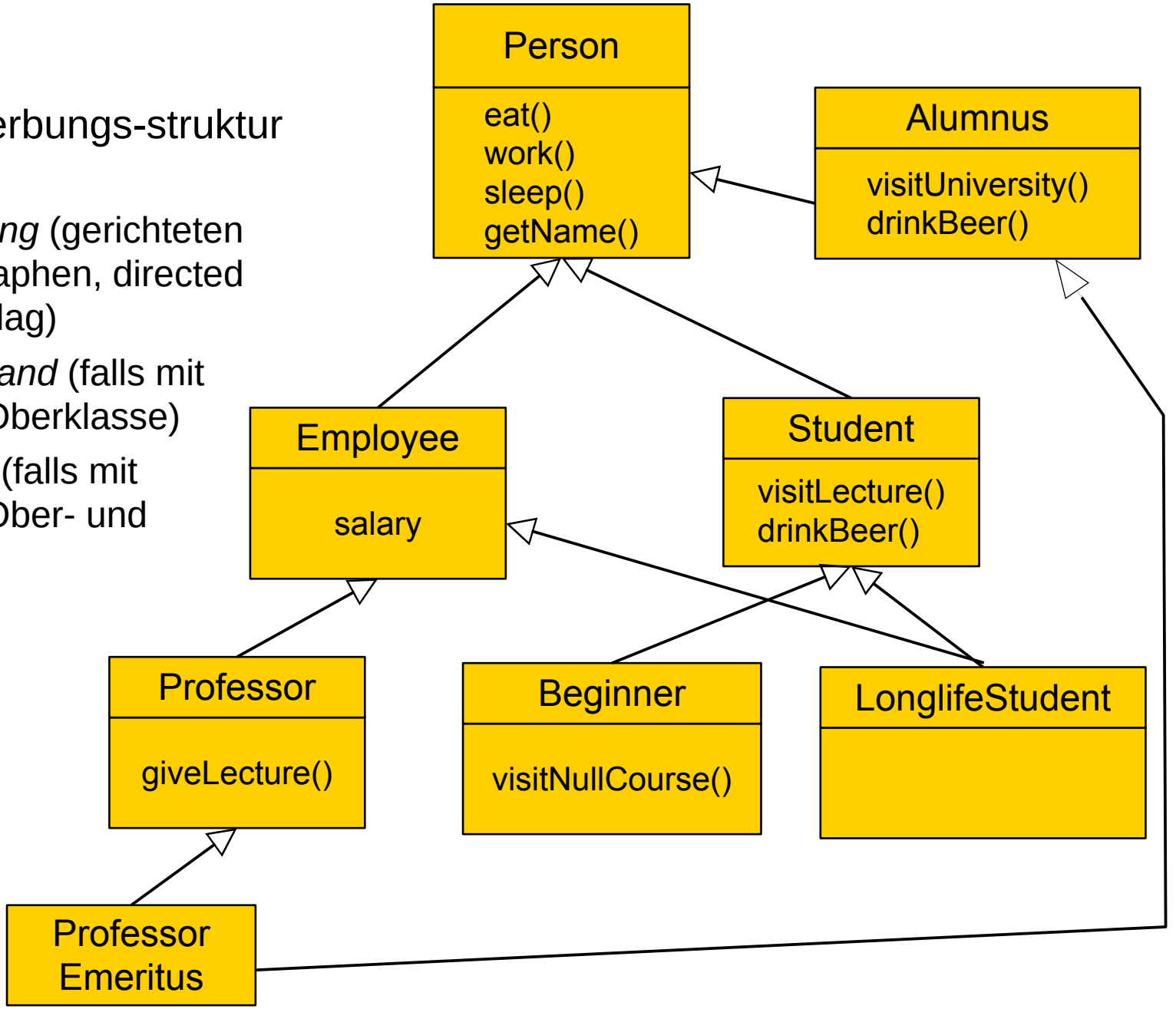
- Mehrfachvererbung
  - Eine Klasse kann mehrere Oberklassen besitzen
  - Dadurch wird die Vererbungsrelation ein gerichteter azyklischer Graph (dag)
- Eine Klasse kann ein Merkmal mehrmals erben
  - Daher muss die Merkmalssuche einer Strategie gehorchen, der Merkmalsauflösungsordnung (method-resolution-order,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



Von wo wird drinkBeer() geerbt?

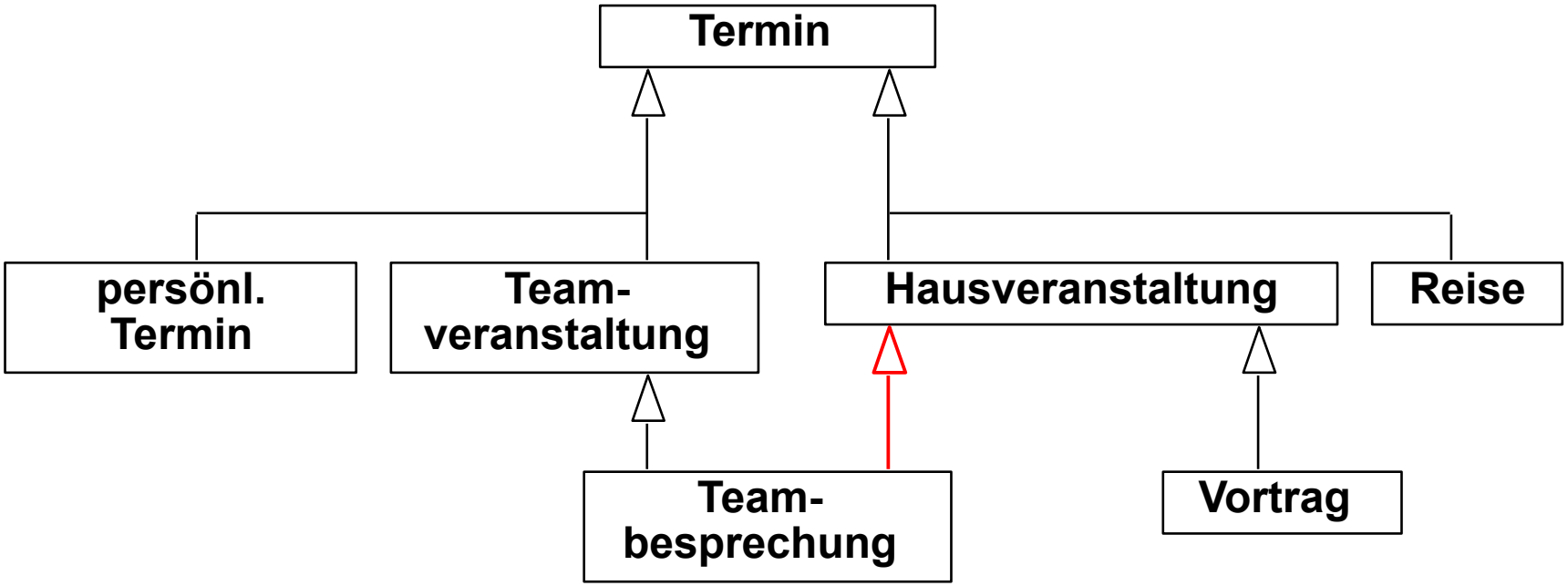
Ein grosser Mehrfachvererbungsverband

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



Realisierung von Code-Mehrfachvererbung in Java

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

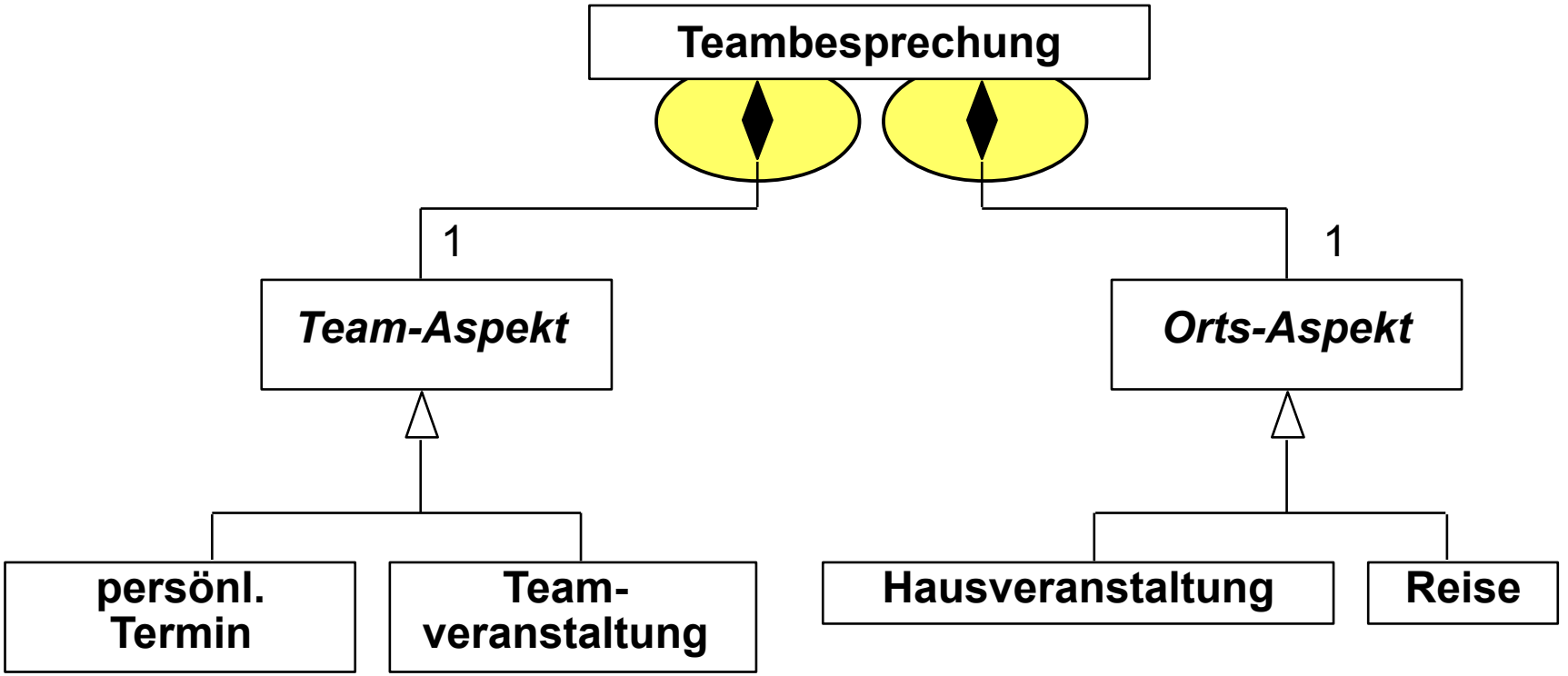


- Java unterstützt Mehrfachvererbung von konkreten Klassen ***nicht,*** nur von Schnittstellen

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

Realisierung von Mehrfachvererbung durch Komposition

- Mehrfachvererbung wird häufig durch komposite Objekte simuliert




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

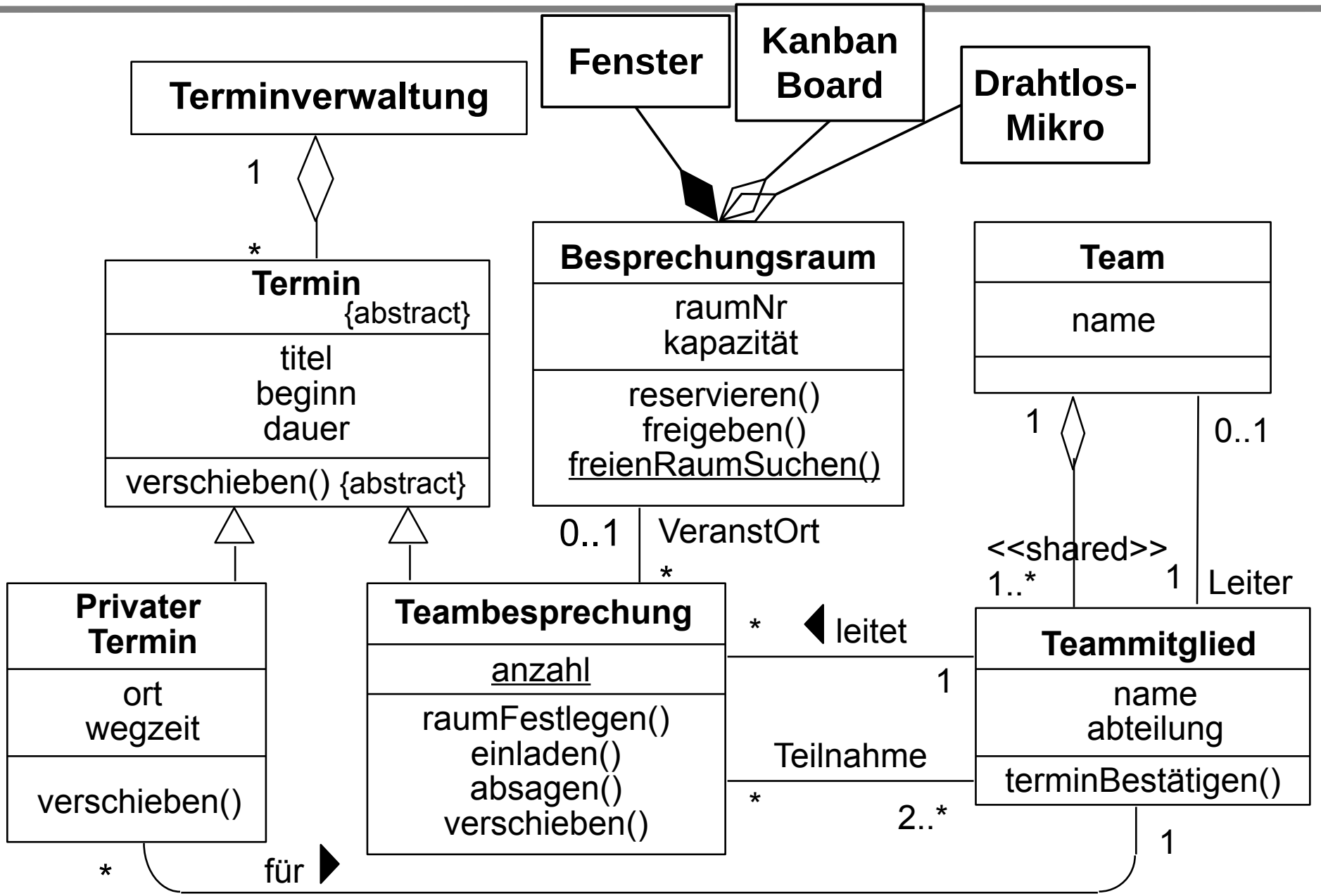
```
abstract class Teamaspekt {
}

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

Verschiedene Ähnlichkeitsrelationen in Analysemodellen (Similarity Relationships)

- *is-a:* zeigt *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:* zeigt ähnliche Struktur an (strukturelle Ähnlichkeit oder Gleichheit)
- *behaves-like:* zeigt Verhaltensähnlichkeit an
  - *conformant (always-behaves-like)*: Konformanz, Ersetzbarkeit (substitutability)
  - *sometimes-behaves-like*: gelegentlich verhaltensgleich
  - *restrictedly-behaves-like*: im allgemeinen 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)	Entwurfs-Modell (Mehr Struktur & mehr Details)
Notation: aUML	Notation: dUML
Objekte: Fachgegenstände	Objekte: Softwareeinheiten
Klassen: Fachbegriffe, parallele Prozesse, komplexe Objekte mit Endorelationen	Klassen: mit Abstrakt, Interface, Stereotyp; Einsatz von Entwurfsmustern für komplexe Objekte
Attribute ohne Typen und Sichtbarkeiten	Attribute: Sichtbarkeiten, Ableitung, Klassenattribute, Initialisierung, weitere spezielle Eigenschaften
Operationen: ohne Typen, Parameter und Rückgabewerte	Operationen: voll typisiert, mit Parameter, Rückgabewert, Klassenoperation
Assoziationen: partiell, bidirektional i. Allg. ohne Datentypen	Unidirektionale Assoziationen mit voller Multiplizität, Navigation, qualifizierte A.
Aggregationen, Kompositionen	
Leserichtung, partielle Multiplizitäten	
Vererbung: Begriffsstruktur	Vererbung: Programmableitung
Annahme perfekter Technologie	Annahme perfekter Technologie
Funktionale Essenz	Erfüllung konkreter Rahmenbedingungen
Völlig projektspezifisch	Gesamtstruktur des Systems
Grobe Strukturskizze	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				
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

- Einige Folien sind eine überarbeitete Version der Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

A.31.1 Analyse von Merkmalen

Einige Details zu bereits in Teil I eingeführten Konzepten

Botschaften, Operation, Methode

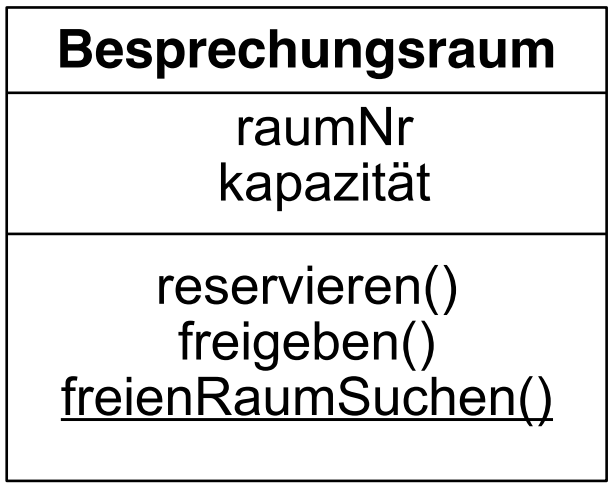
- Eine **Botschaft** (**Message, Nachricht)** ist eine Nachricht eines Senders an ein Empfänger-Objekt, eine Operation auszuführen
- Eine **Rückmeldung** ist eine Botschaft, die ein Sender einer Botschaft als Antwort erhält
- Eine **Operation** koppelt eine Botschaft mit einer Reaktion und einer Rückmeldung: “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
- **Imlementierung:**
  - Implmentierungsmuster Singleton
  - Klassenattribute und -operationen: Schlüsselwort **`static`**

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




```
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
```

Spezifikation von Operationen

- **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.31.2 Weitere Profile für die Analyse

Beispiel: Bahrami-Profil für Domänenmodell

- Das Bahrami Profil wird hauptsächlich im Domänenmodell eingesetzt
  - Und damit als Typen für die Schnittstellen im Kontextmodell

- Konzept, Begriff (concept) <<concept>>
  - Etwas, worauf sich viele Leute eines Anwendungsbereiches geeinigt haben. Oft angeordnet in Taxonomien oder Ontologien
  - Benutzt im Domänenmodell
- Ereignisklasse (Event) <<event>>
  - Ereignis in der Zeit, extern zum Objekt
- Organisation <<organization>>
  - Verkörpert Wissen über eine Organisationseinheit
- Menschen (People) <<people>>

- Plätze (Places class) <<place>>

Beispiel: Rumbaugh-Profil

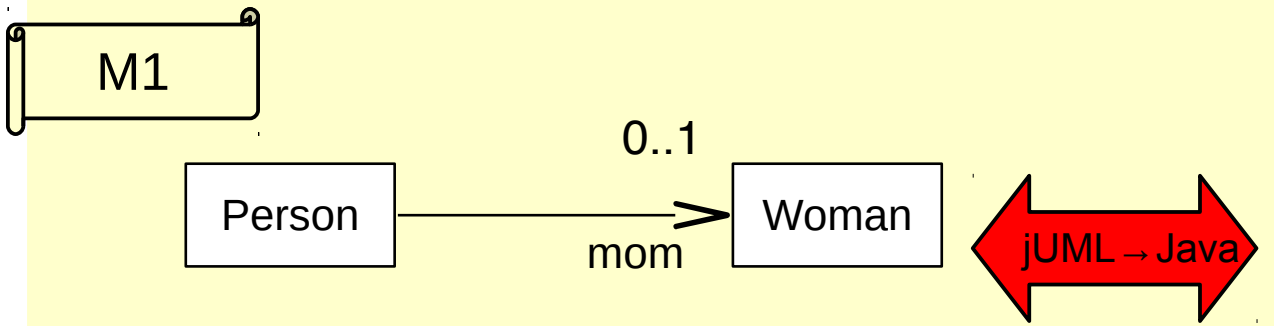
- Domänenmodell:
  - Physical class (e.g., Boat) <<physical>>
  - Business class (e.g., Bill) <<business>>
- Anwendungslogik in Kontextmodell und Toplevel-Architektur:
  - Logical class (e.g., Timetable) <<logical>>
  - Application class (e.g., BillingTransaction) <<application>>
  - Behavioral class (e.g., Cancellation) <<behavior>>
- Plattform im Implementierungsmodell:
  - Computer class (e.g., Network) <<computer>>

A.31.3 Analyse von Assoziationen

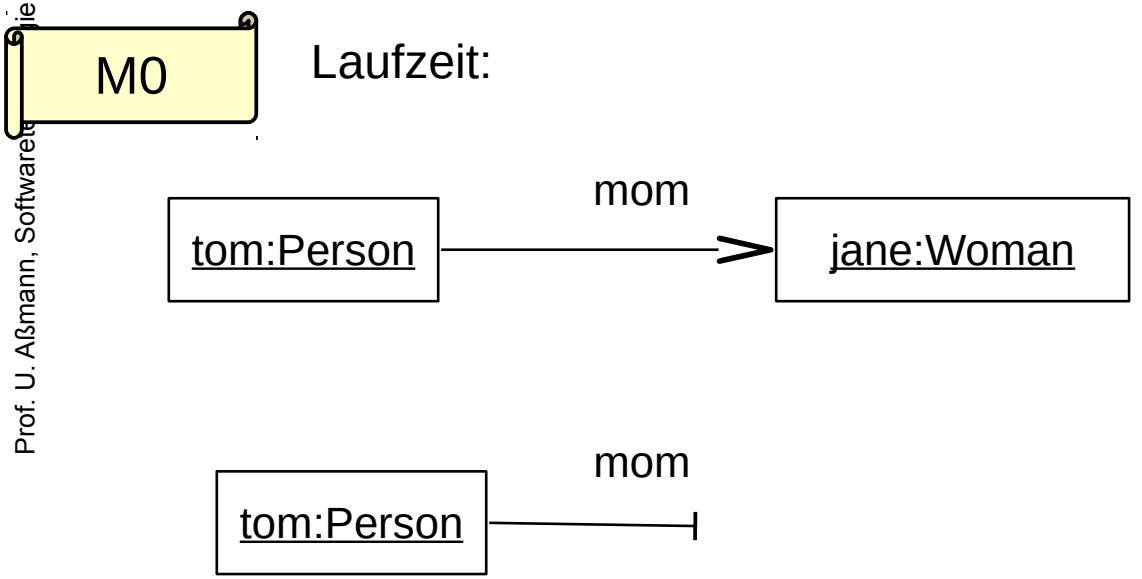
Einige Details zu bereits in Teil I eingeführten Konzepten

Einseitige einstellige Assoziationen in Java und jUML (Wdh.)

- Ein Kind kann höchstens eine Mutter haben



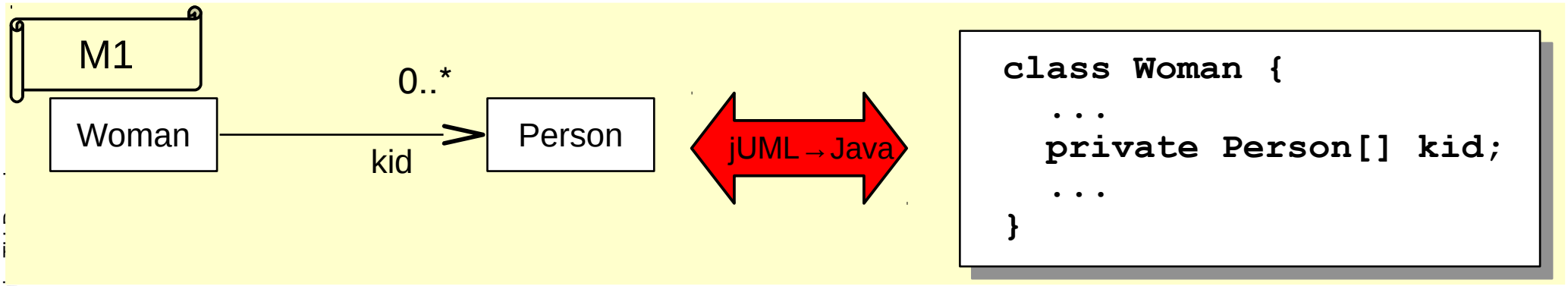

```
class Person {
   ...
   private Woman mom;
   ...
}
```

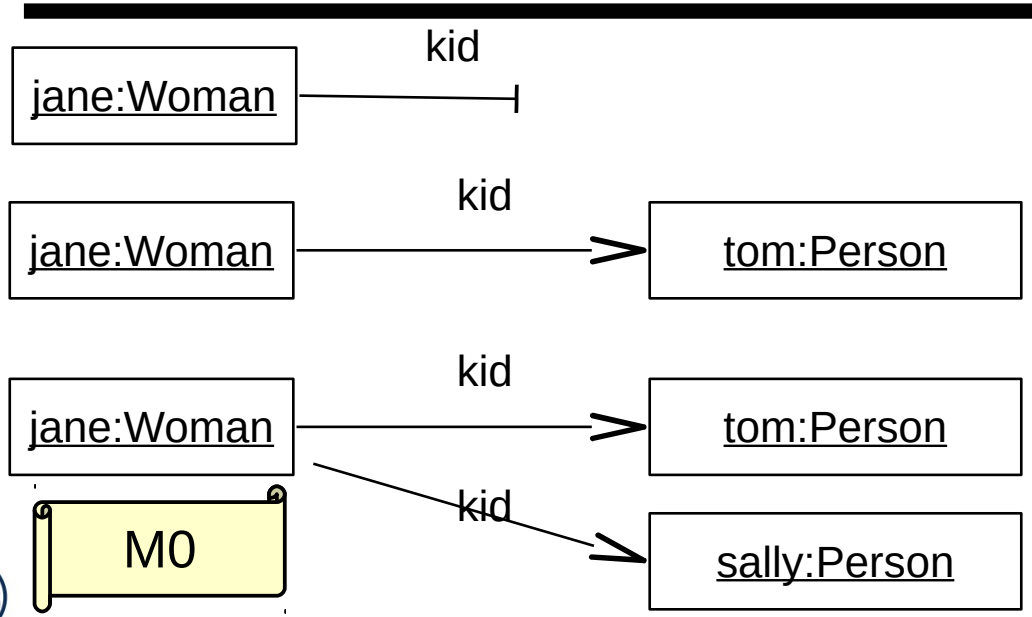



```
Person tom = new Person();
tom.mom = jane;
...
tom.mom = null;
```

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.)



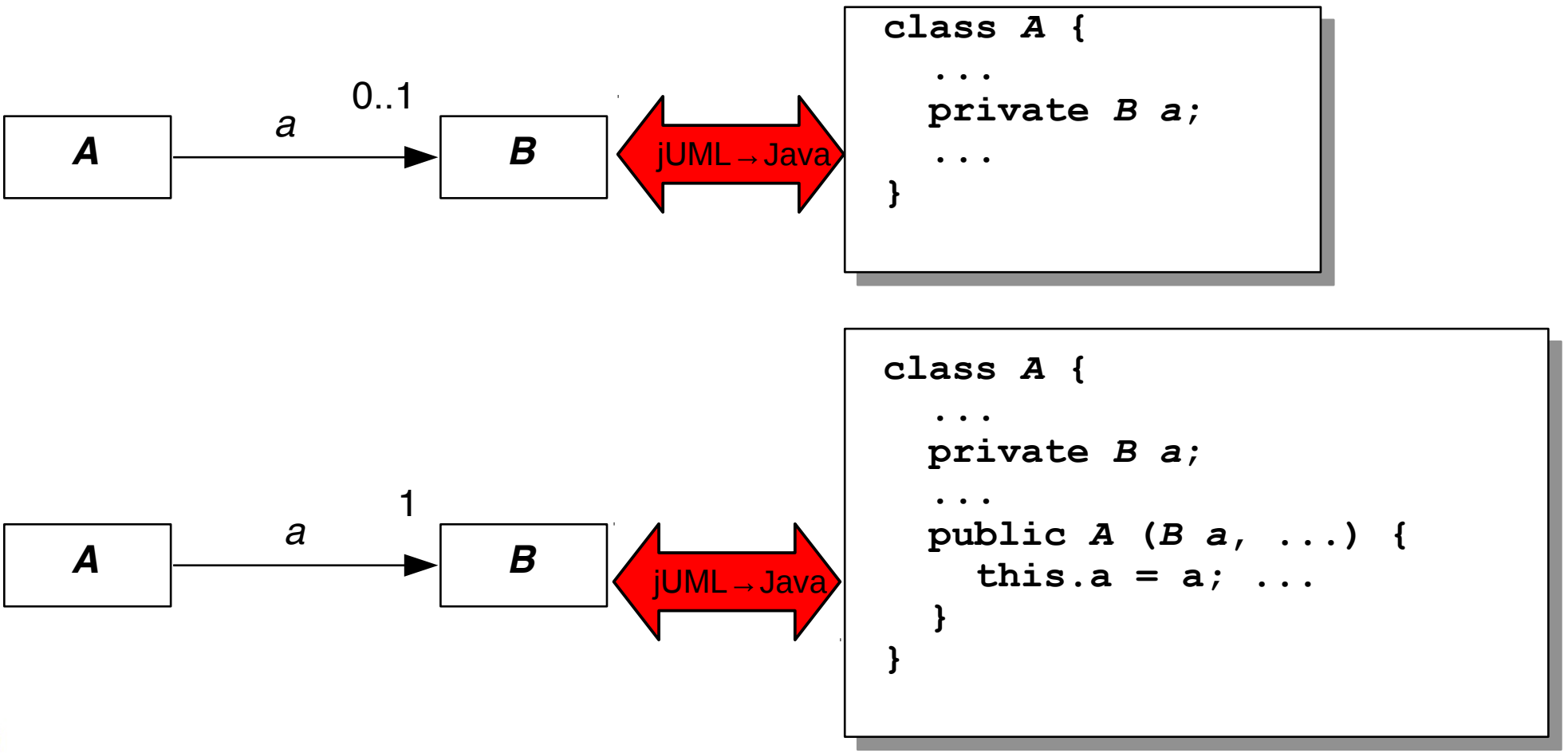


Laufzeit:

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

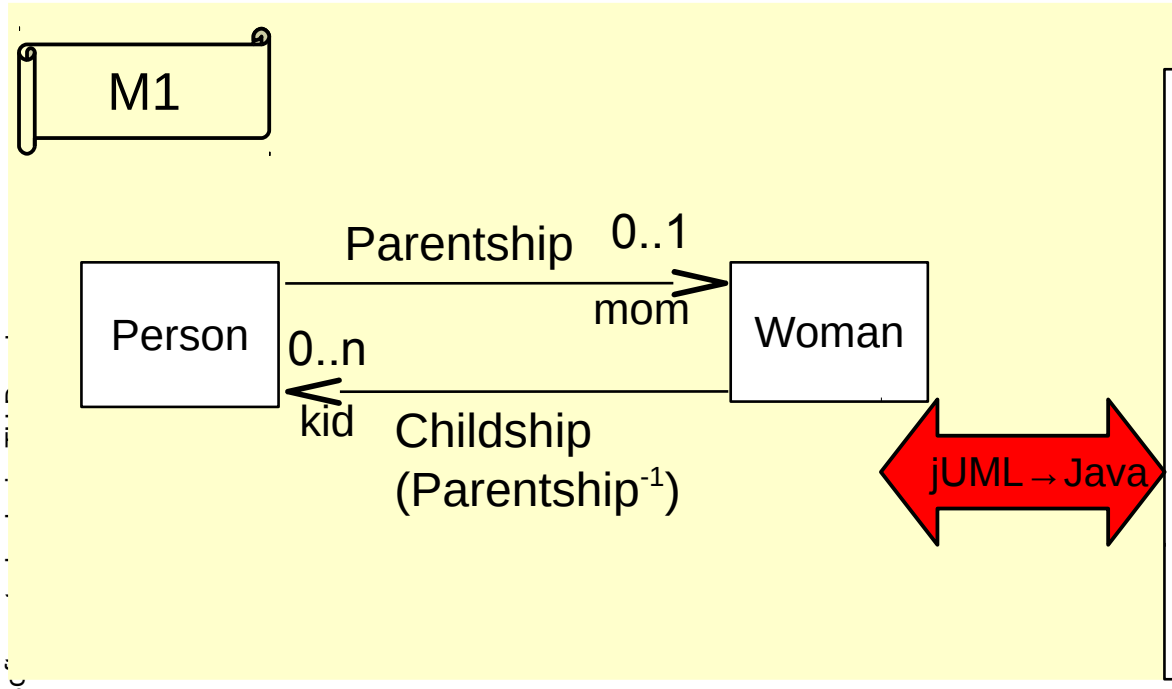
Optionale und notwendige Assoziationen

- 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..*

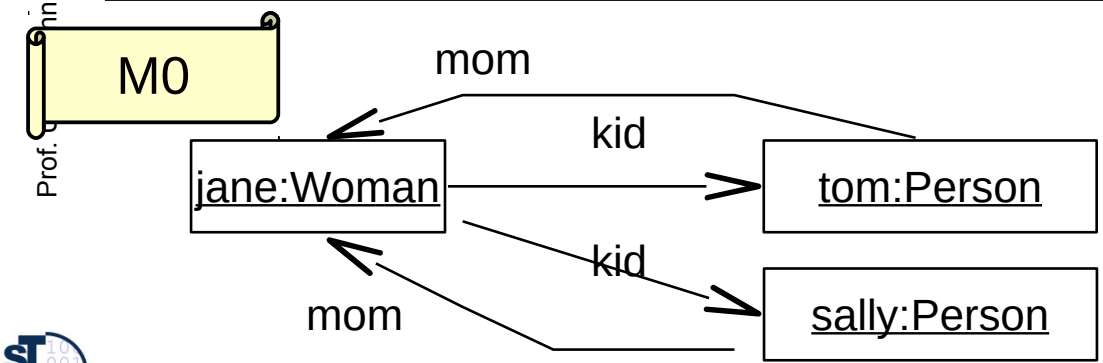


Zu 31.3.2. Realisierung von bidirektionalen Assoziationen durch unidirektionale

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



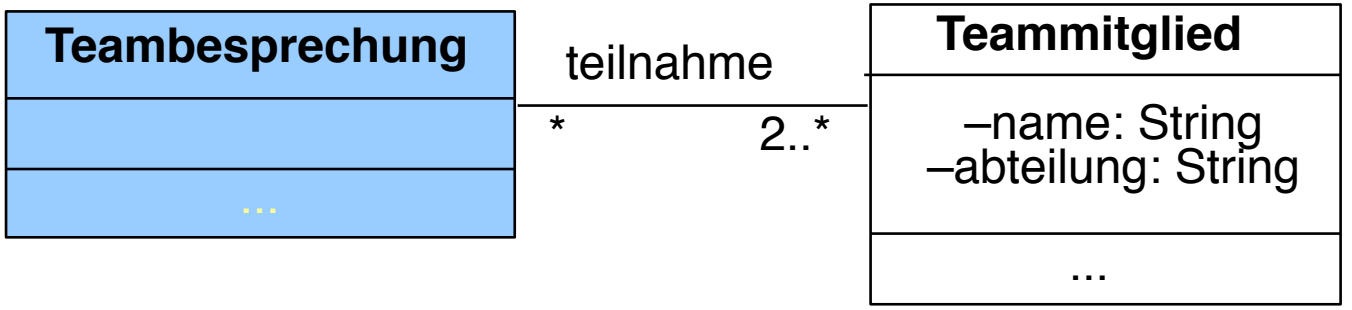

```
class Person {
   ...
   private Woman mom;
   ...
}
class Woman {
   ...
   private Set<Person> kid;
   ...
}
```




```
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)



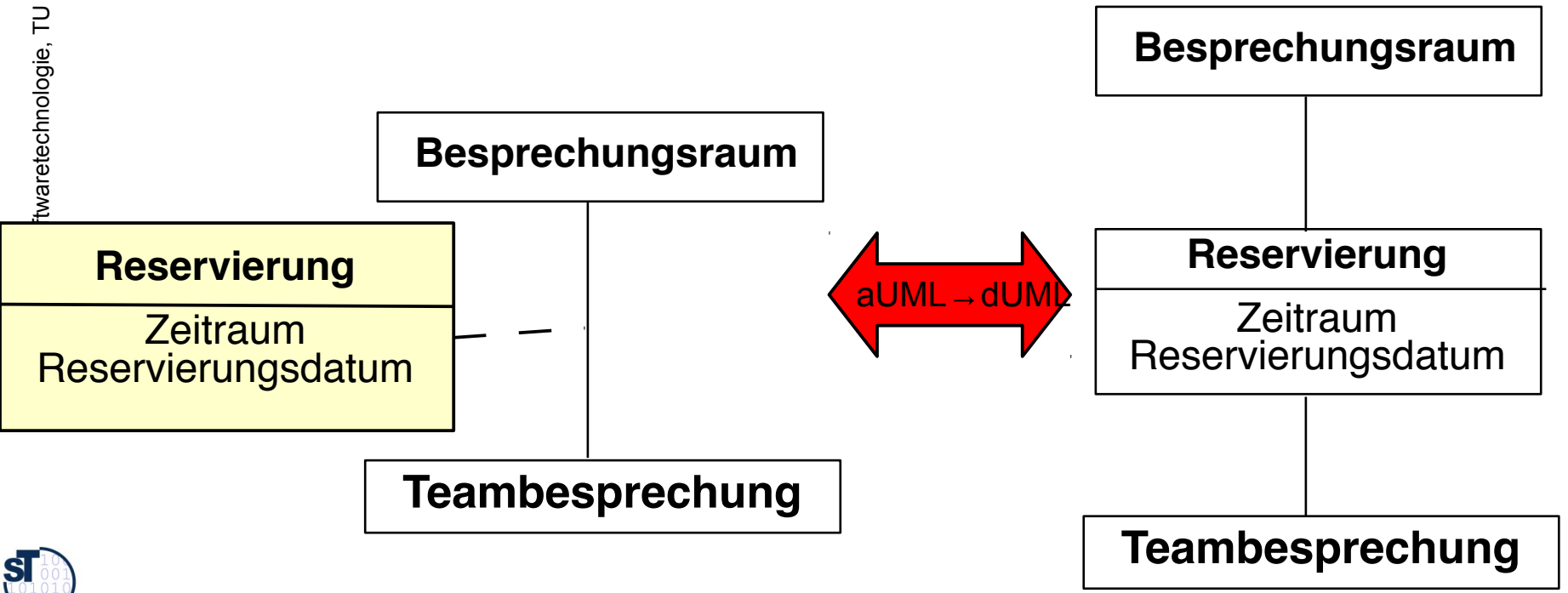

```
class Teambesprechung {

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

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

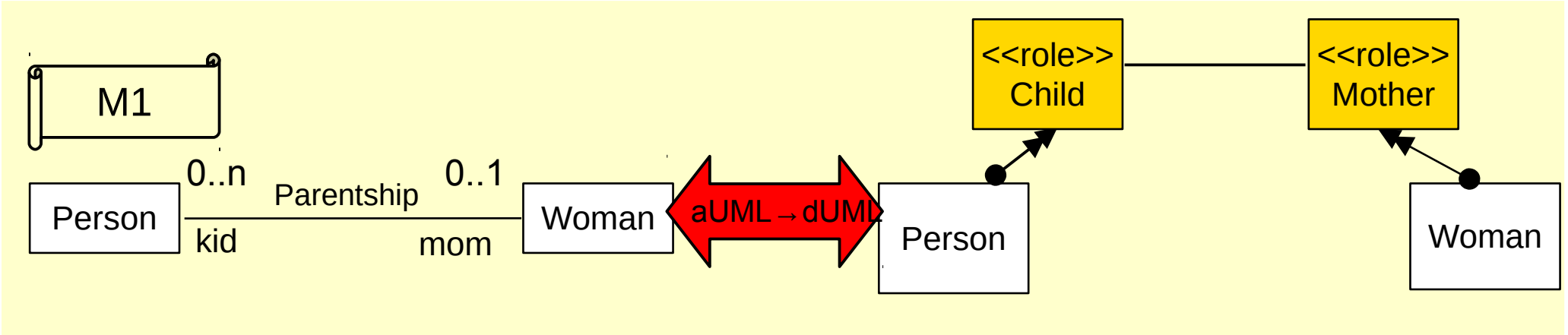
31.3.2.4 Realisierung von Assoziationsklassen mit “Zwischenklassen”

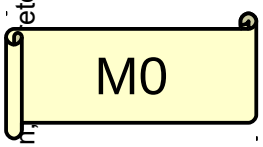
- Assoziationsklassen können im Implementierungsmodell in normale relationale „Zwischen“-Klassen abgeflacht werden:
  - Attribut "Reservierungsdatum" für eine Raumreservierung wird für Assotiation benötigt (z.B. um Reservierungskonflikte aufzulösen).
  - Die Klasse "Reservierung" wird in die bestehende 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



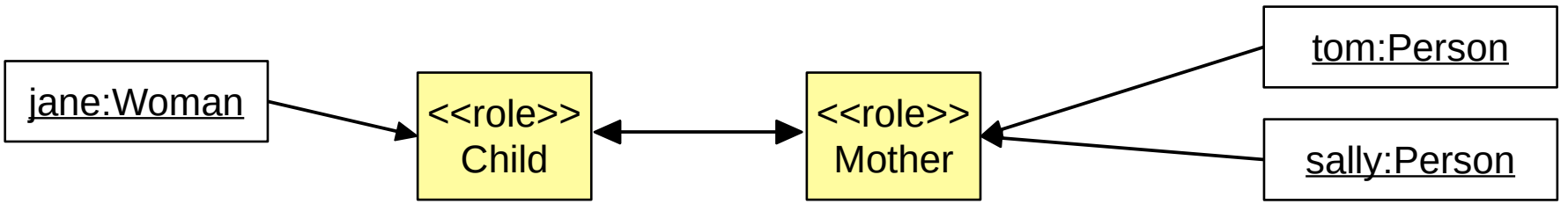
31.3.2.5 Realisierung von bidirektionalen Assoziationen durch Rollenklassen

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





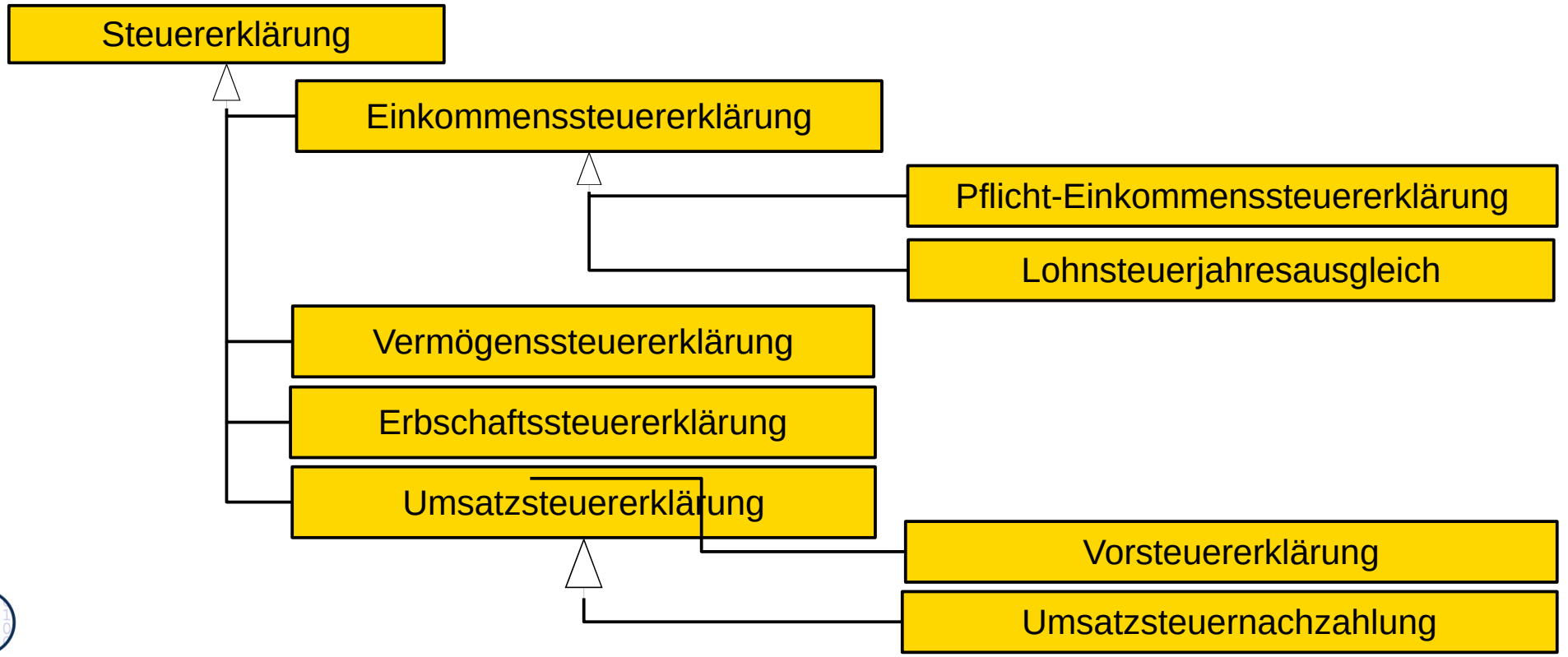
Laufzeit: Referenzen zwischen Rollenobjekten und den assoziierten Objekten



A.31.4 Weitere Beispiele für Zeilenhierarchien

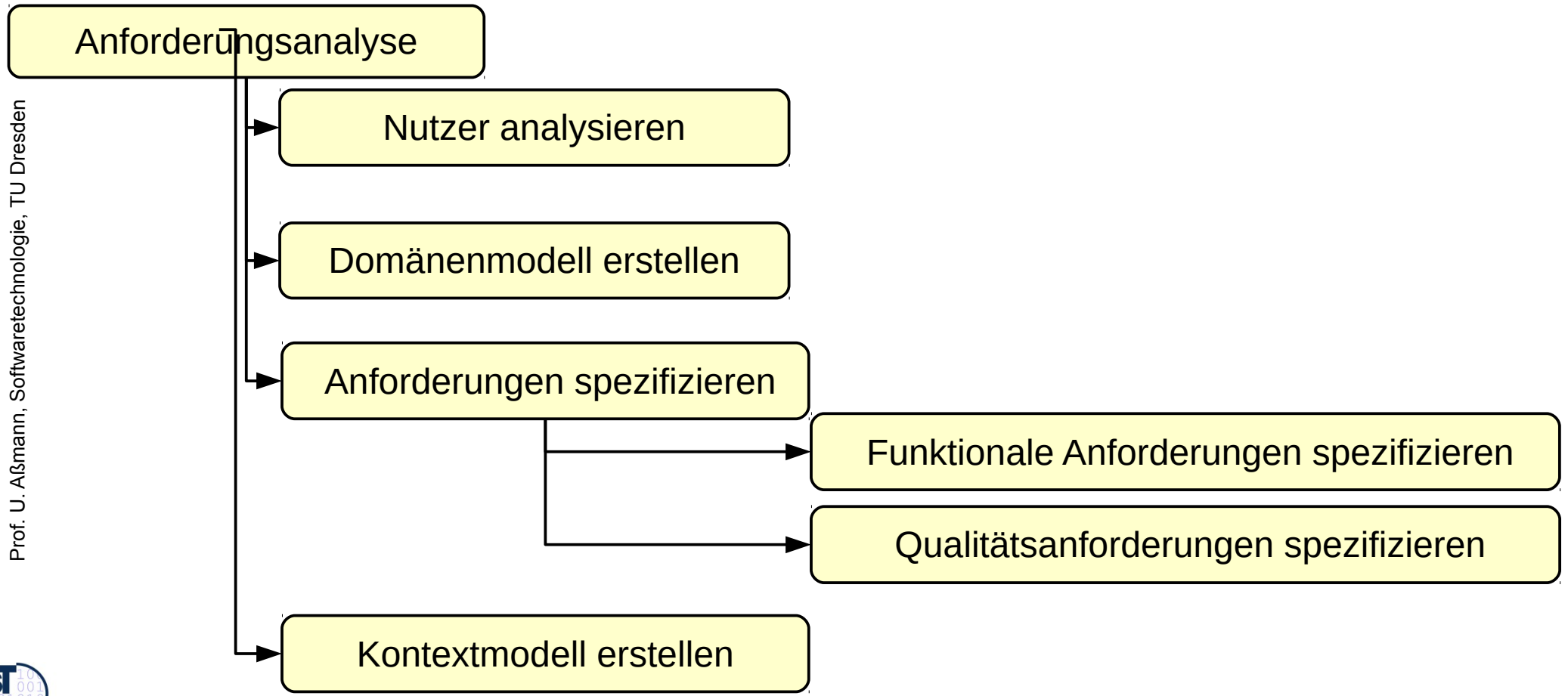
Zeilenhierarchien

- 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]



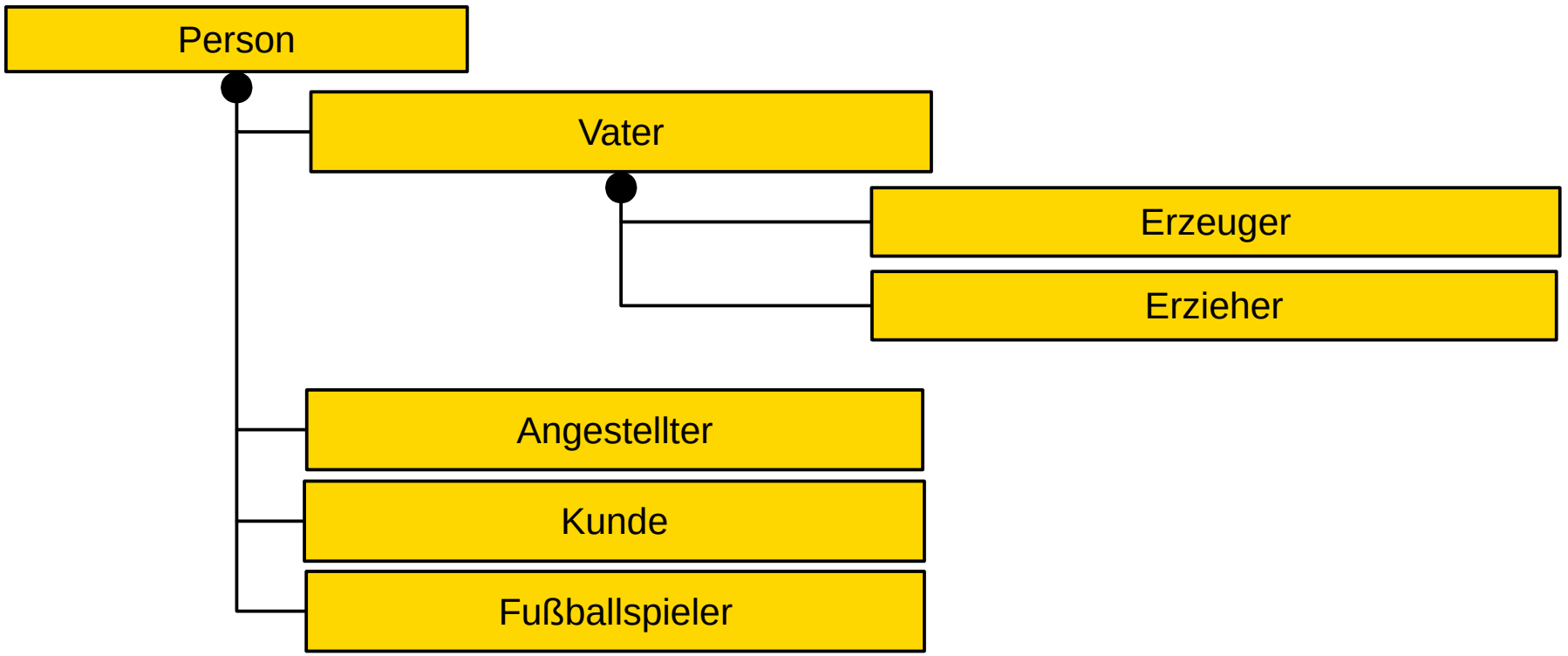
Bsp. Aktivitätenhierarchie (Aktivitäts- oder Funktionsbaum)

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



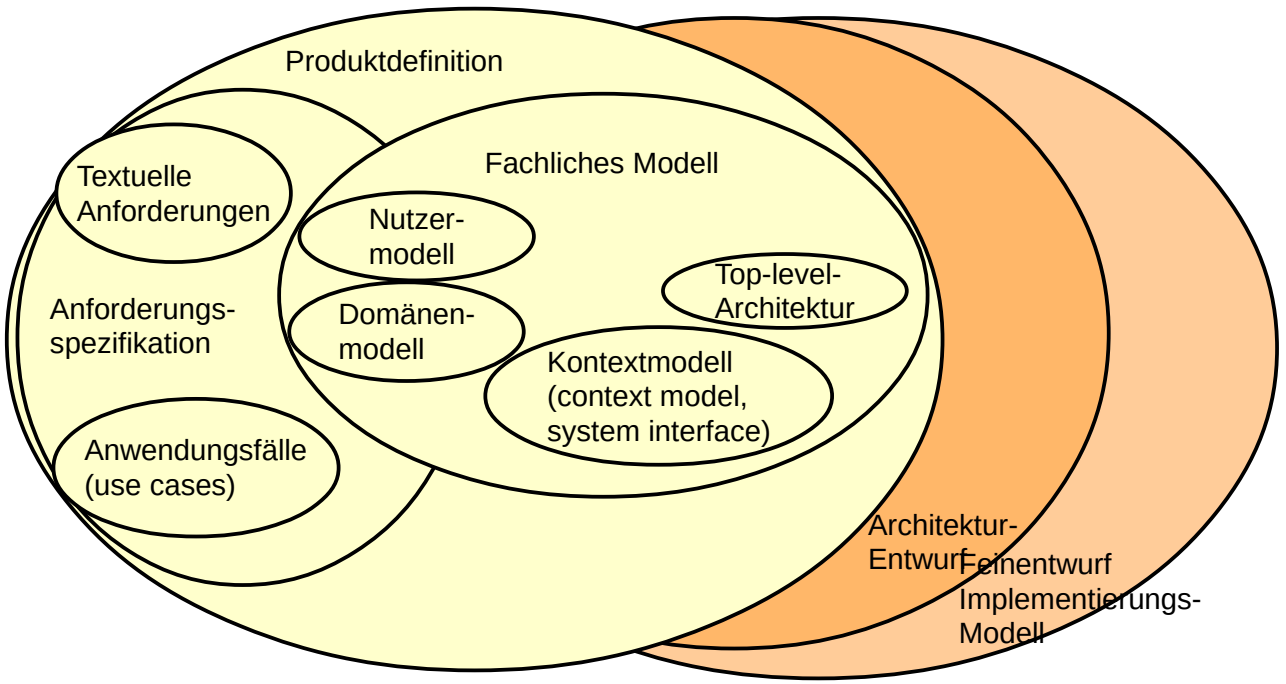
Eigenschaftshierarchie

- 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 *gröber* 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



Verfeinerung von der Analyse zum Entwurf

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 Zuverlässigkei**t: 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)
    - Ströme und Kanäle
  - Relationen:
    - n-stellige Assoziationen, bidirektionale Assoziationen
    - Aggregationen, Kompositionen
    - Konnektoren
- 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