34. Querschneidende Verfeinerung und die Erweiterung komplexer Objekte durch Mehrfach-Brücken

Prof. Dr. rer. nat. habil. Uwe Aßmann

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

Version 16-1.0, 20.06.16

1) Komplexe Objekte in Informationssystemen
2) Modellierung von komplexen Objekten
   1) Komplexe Objekte
      - Private und essentielle Teile
   2) Natürliche Typen und Rollen
   3) Realisierung von Rollen und Teilen durch Bridge
   4) Objektanreicherung
3) Anhang
   1) Facetten
   2) Phasen

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

© Prof. U. Aßmann

Obligatorische Literatur

- Störrle 5.3, 5.4

- Weitere Literatur:
  - L. Maciaszek. Requirements Analysis and System Design – Developing Information Systems with UML. Addison-Wesley.
  - Giancarlo W. Guizzardi. Ontological foundations for structure conceptual models. PhD thesis, Twente University, Enschede, Netherlands, 2005.
  - Nicola Guarino, Chris Welty. Supporting ontological analysis of taxonomic relationships. Data and Knowledge Engineering, 39:51-74, 2001.
  - Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. Data Knowl. Eng, 35(1):83-106, 2000.

© Prof. U. Aßmann

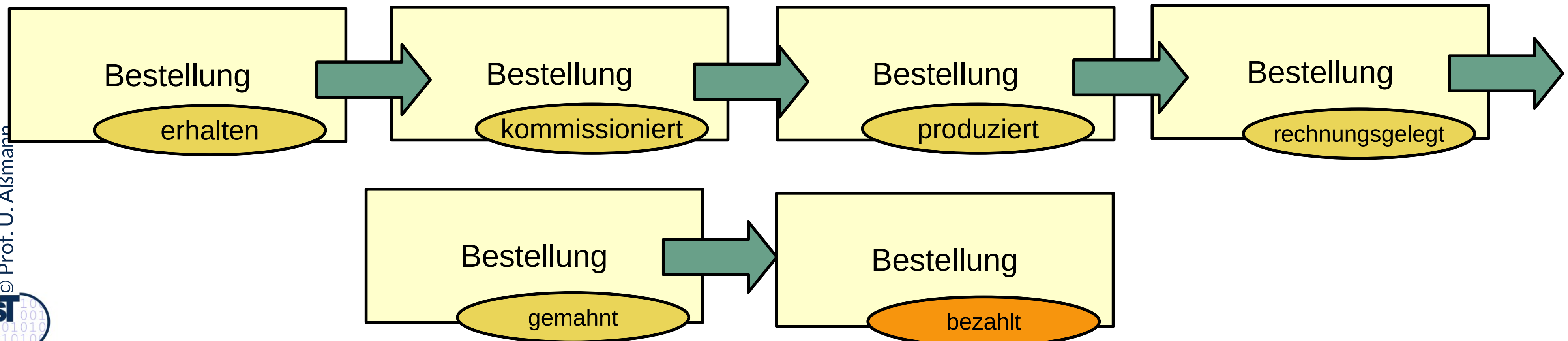
34.1 Komplexe Objekte in Informationssystemen

Wir haben in Kap. 31 (Strukturelle Modellierung) bereits die Analyse komplexer Objekte kennengelernt. Hier folgen mehr Details

DRESDEN concept Exzellenz aus Wissenschaft und Kultur

Geschäftsobjekte (Business Objects)

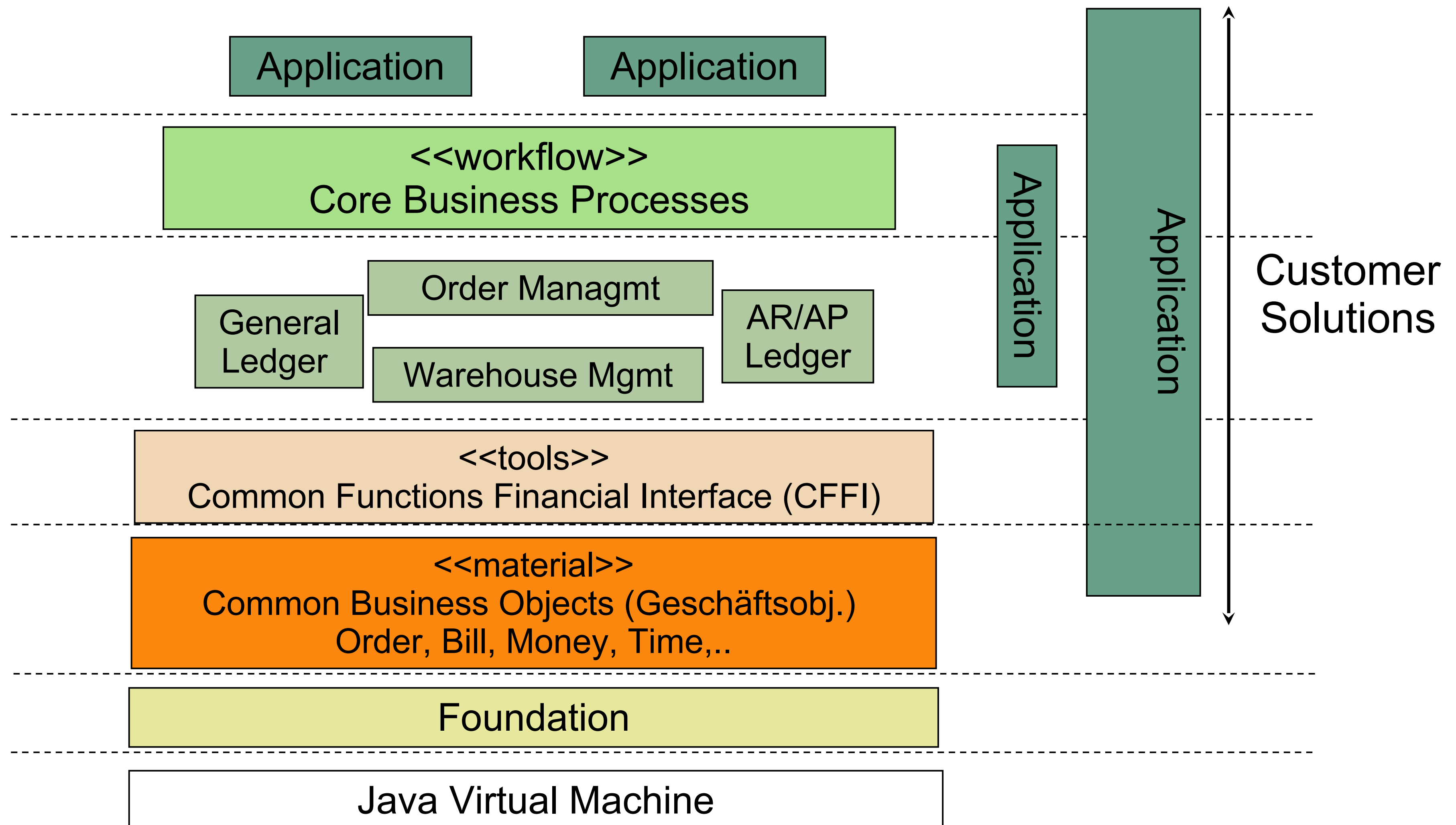
- In großen objektorientierten Frameworks werden die Objekte sehr komplex
- Beispiel: `Bestellung`, ein Geschäftsobjekts in Geschäftssoftware (Enterprise Resource Planning, ERP):
  - eine Bestellung ist ein langlebiges, persistentes komplexes Objekt, das von dem
    - Auftragseingang des Kundens an durch die
    - Produktion, Kommissionierung, Rechnungserstellung,
    - Auslieferung und Mahnwesen
  - erhalten bleiben muss
- Dynamische Erweiterbarkeit nötig, da die Bestellung verschiedene Phasen durchläuft und daher viele verschiedene Rollen spielt
  - Enthält viele Teile, und wird in neuen Phasen ständig mit neuen Teilen, Attributen und Methoden versehen
  - Verhalten muss an die Phase adaptiert werden



© Prof. U. Aßmann

Architektur von IBM San Francisco
Java-Framework für Geschäftsanwendungen (ERP)

- P. Monday, J. Carey, M. Dangler. SanFrancisco Component Framework: an introduction. Addison-Wesley, 2000.



© Prof. U. Aßmann

Beispiel: Geschäftsobjekte in der SanFrancisco Bibliothek

- Wertobjekte:
  - Adresse, Währung, Kalender

Allgemeine Geschäftsobjekte:

- Firma
- Geschäftspartner
- Kunde (und Person)
- Zahlen mit beliebiger genauigkeit mit Nachkommastellen
- Fiskalische Kalender
- Bezahlmethode
- Maßeinheit

- Finanzielle Geschäftsobjekte
  - Geld
  - Währungsgewinn
  - Konto
  - Verlustkonto
- Allgemeine Mechanismen für Geschäftssoftware:
  - Buchführungskonten
  - Klassifikationen
  - Schlüsselobjekte

© Prof. U. Aßmann

Problem

- Start des Projekts: 1995
- IBM stellte 1999 sein Framework wieder ein
- Technische Gründe liegen in Java:
  - Einfache Vererbung von Java erschwert die Wiederverwendung von Code
  - Komplexe Objekte (Geschäftsobjekte) können in Java schlecht repräsentiert werden (keine Endorelationen)
  - Dynamisch sich wandelnde komplexe Objekte können schlecht repräsentiert werden:
    - Dynamische Teile wie Phasen und Rollen können nicht einfach abgebildet werden. Dynamische Anpassung ist zu komplex
  - In Wirklichkeit hat Java Probleme, komplexe *und* dynamische Objekte zu beschreiben. Es braucht dazu Entwurfsmuster, aber auch Mechanismen für große Objekte
- Das Folgende stellt Modellierungsmethoden für komplexe Objekte vor

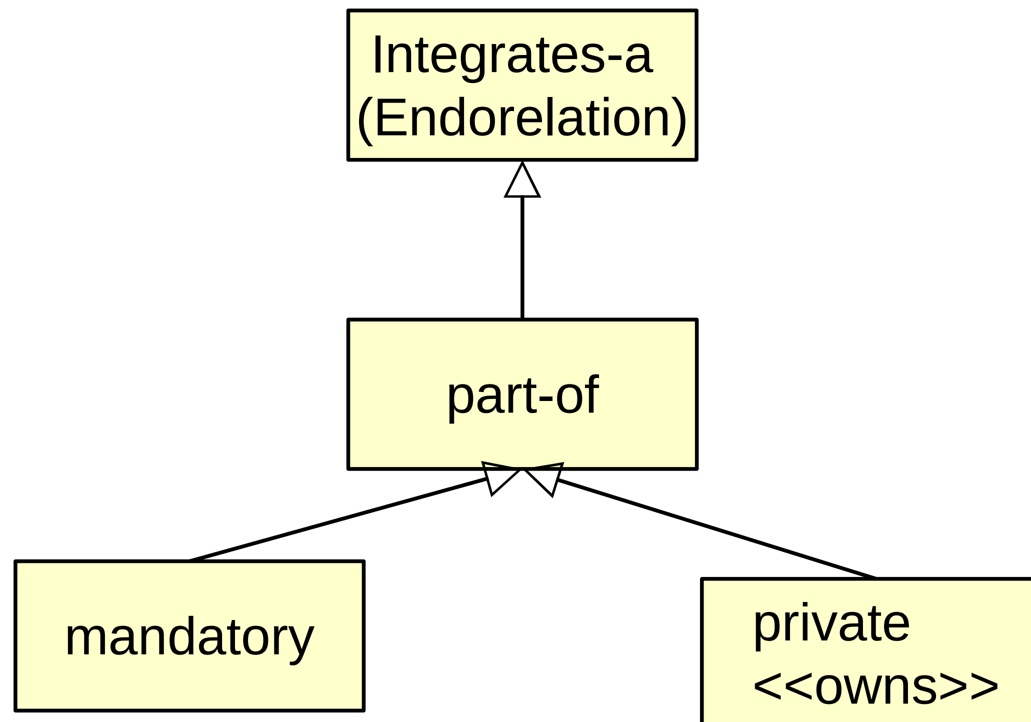
© Prof. U. Aßmann

Ein **komplexes Objekt (Subjekt, big object)** ist ein Objekt,
das auf Programmierniveau
wegen seiner Komplexität durch mehrere Objekte dargestellt wird.

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

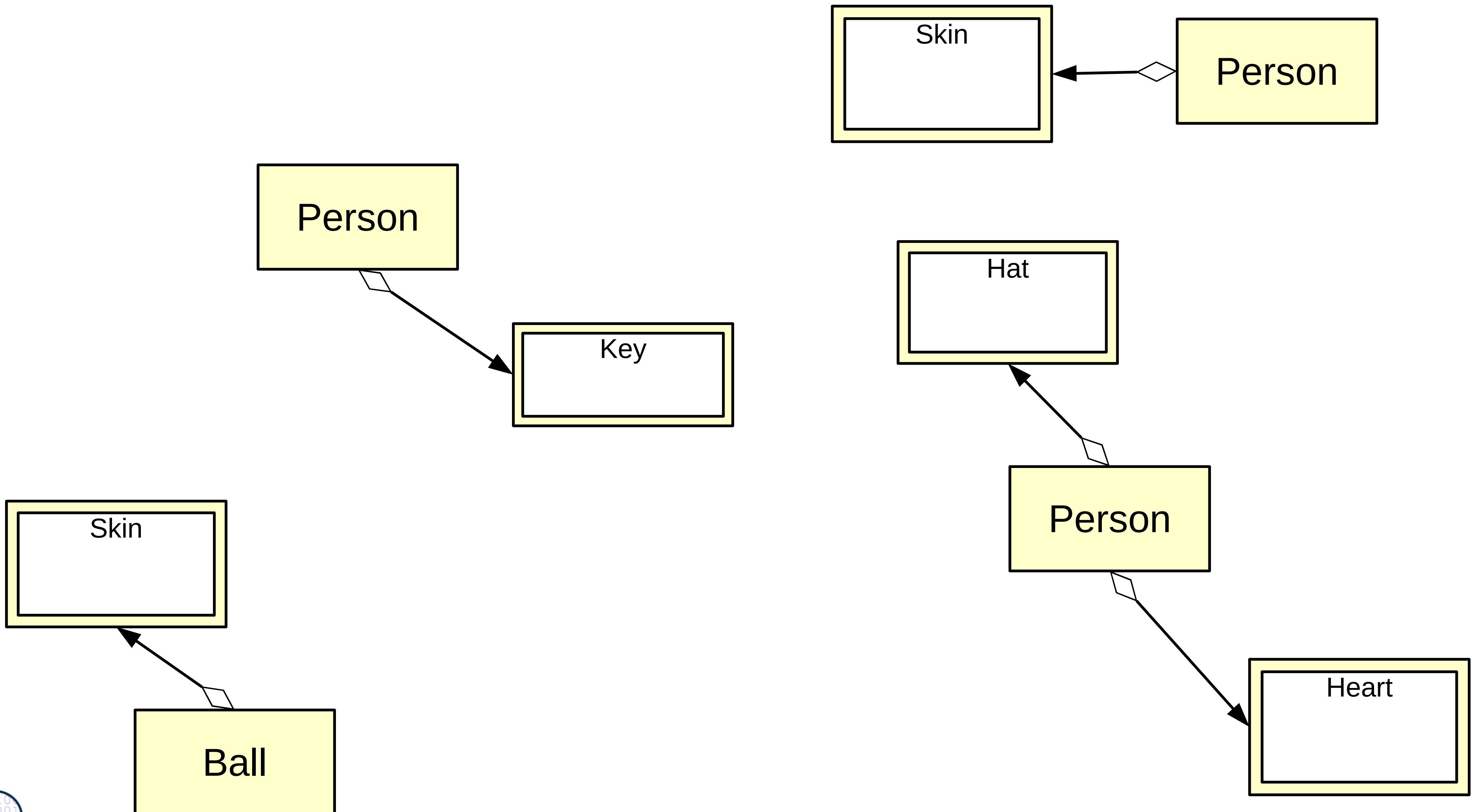
© Prof. U. Aßmann

34.2 Private und essentielle Teile von kompositen Objekten



Private und essentielle Teile

- Gibt es hier einen Unterschied in der Bedeutung der Aggregationen?



© Prof. U. Aßmann

Private Teile für komposite Objekte

Private Teile

- Ein **privates Teil (owned part)** ist ein nicht-fundiertes Unterobjekt, das ausschließlich zu einem Kernobjekt gehört
  - Beispiel: Eine Person hat einen Hut
  - Zu einer Zeit hat das Teil genau einen Eigner (alias-freie Ganz/Teile-Beziehung), aber das Teil kann das Ganze wechseln
  - Stereotyp <<owns>>
- Ein **zugeeignetes Teil (exclusively owned part)** ist ein rigides privates Unterobjekt, das immer zu einem Kernobjekt gehört
  - Beispiel: eine Person hat einen Arm
  - Teil gehört exklusiv dem Ganzen und kann die Zugehörigkeit nicht ändern
  - Stereotyp <<exclusively-owns>>

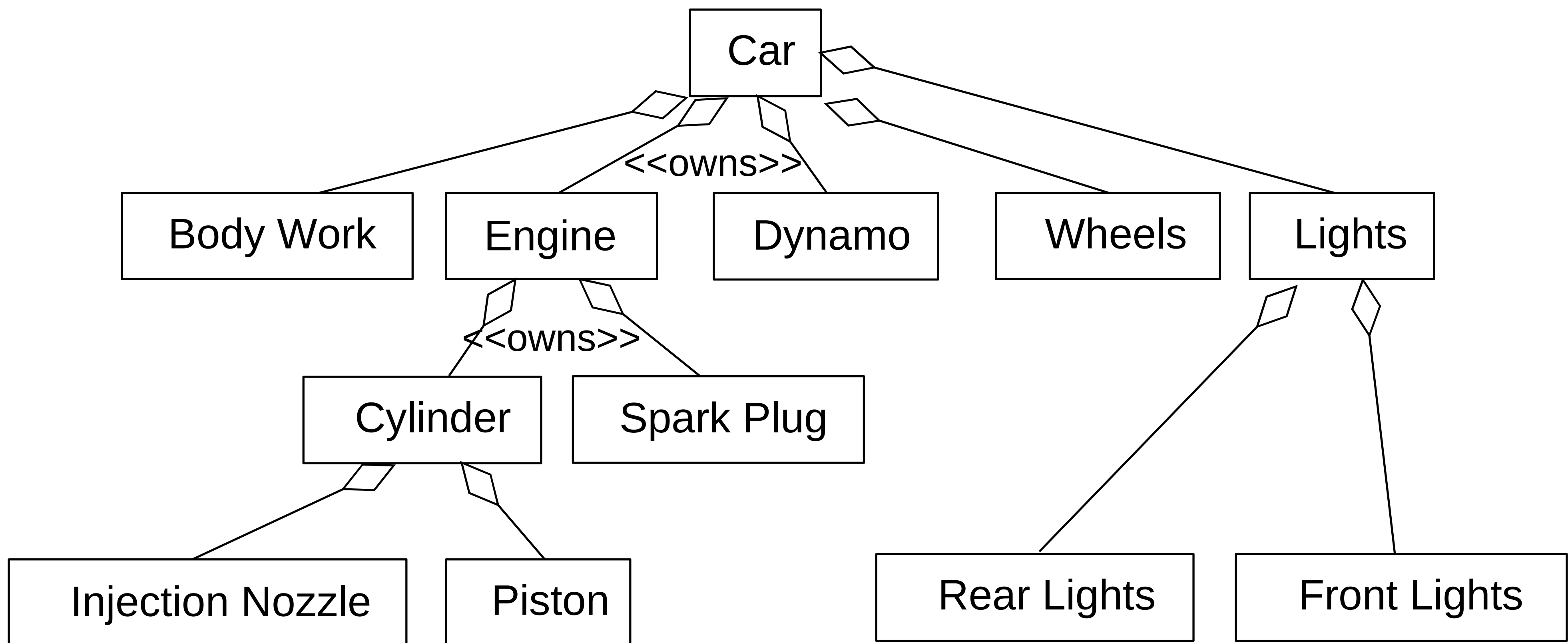
Obligatorische Teile:

- Ein **obligatorisches Teil (mandatory part)** ist ein zugeeignetes Unterobjekt, von dessen Typ das Kernobjekt unbedingt eines braucht
  - Beispiel: Eine Person braucht ein Herz
  - das Ganze braucht ein Teil vom Typ des Teils
  - Stereotyp <<mandatory>>
- Ein **wesenhaftes Teil (essential part)** ist ein essentielles Teilobjekt, das nicht ausgewechselt werden kann, ohne das Kernobjekt zu zerstören
  - Beispiel: Eine Person braucht ein Gehirn
  - Das Ganze braucht genau dieses Teilobjekt zum Leben
  - Stereotyp <<essential>>

© Prof. U. Aßmann

Hierarchische Systemzerlegung mit privaten Teilen (<<owns>>)

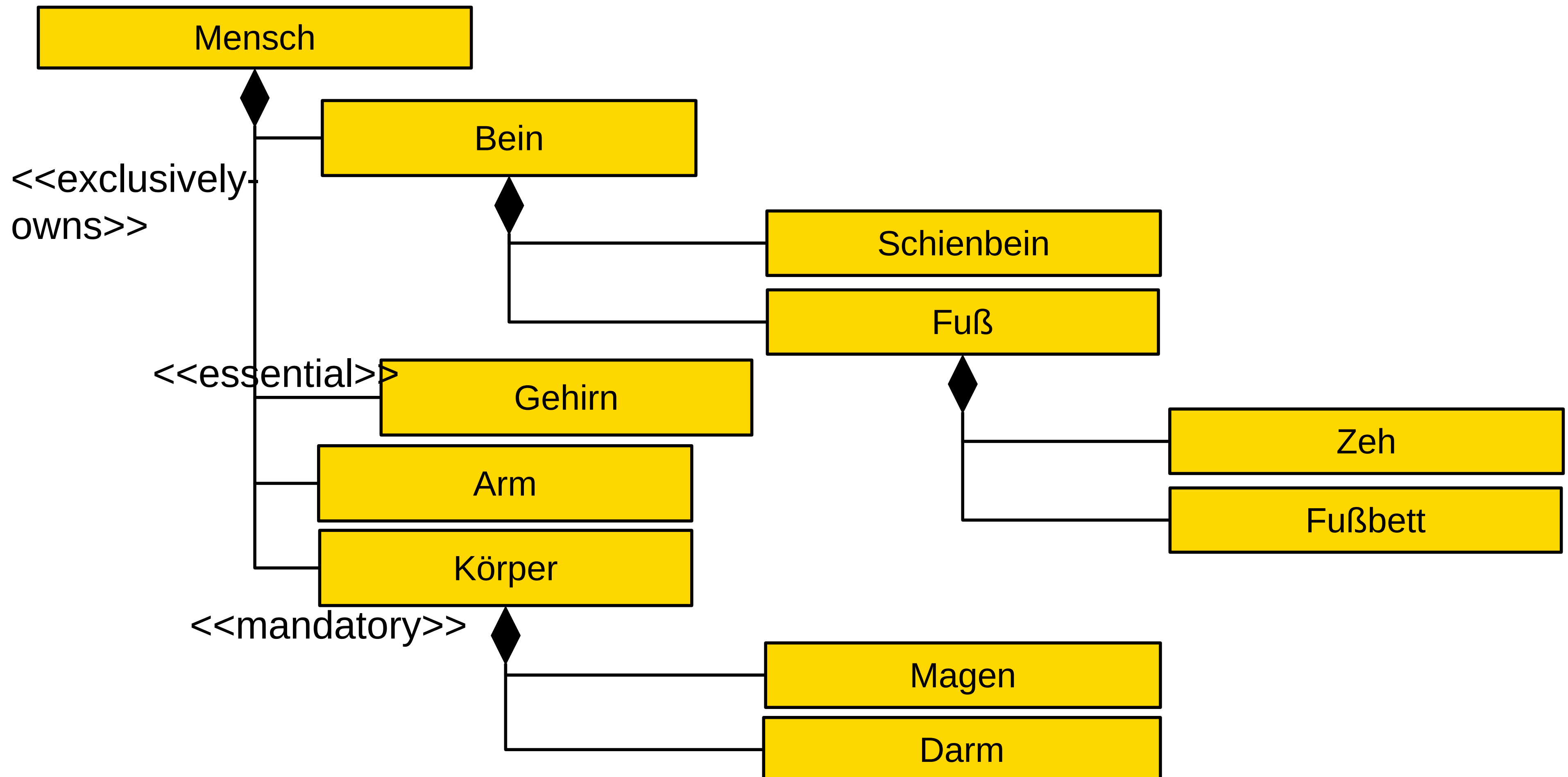
- Bei Eigentumsbeziehungen gibt es kein Teilen von Unterteilen (kein *sharing*, kein *aliasing*)
- Merke: die spezielle Semantik von Teile-Relationen kann durch *Stereotypen* angegeben werden



© Prof. U. Aßmann

Darstellung komplexer Objekte mit privaten Teilen

- <<exclusively-owns>> kann durch <<essential>> und <<mandatory>> spezialisiert werden



© Prof. U. Aßmann

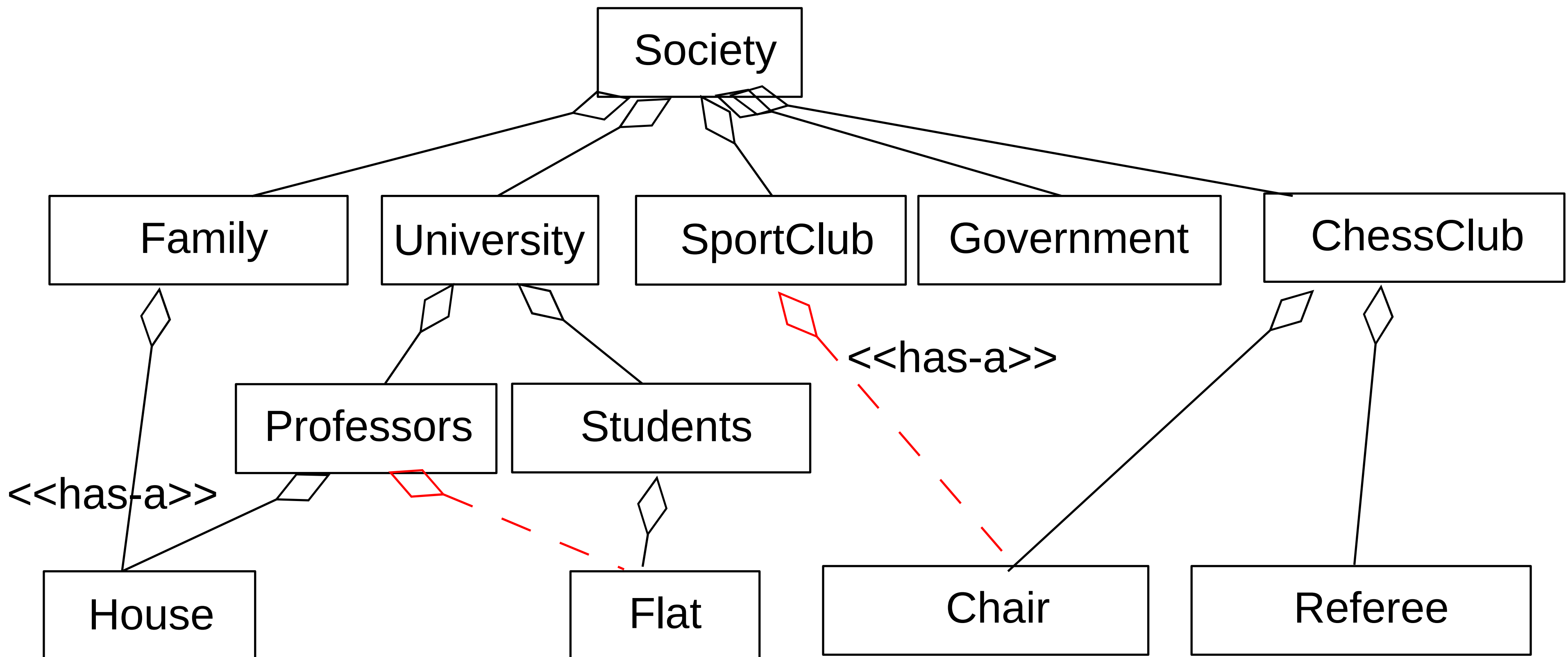
Weitere Ganz/Teile-Beziehungen (Whole-Part Relationships)

- **Eigentumsbeziehungen** bilden baumförmige Relationen
  - *Private Teile (s. vorige Folie)*
  - *Abhängiges Teil (composed-of)* (Komposition in UML): das Teil hat die gleiche Lebenszeit wie das Ganze und kann nicht alleine existieren
- **Einfache Teilebeziehungen** sind azyklisch, bilden aber keine Hierarchien
  - *has-a*: Aggregation, einfache Teilebeziehung. Das Teil kann Teil von mehreren Ganzen sein (Aliase möglich)
  - *member-of*: Wie has-a, aber Gleichheit mit Geschwistern gefordert

© Prof. U. Aßmann

Geschichtete Systemzerlegung mit nicht-privaten Teilen (<<has-a>>)

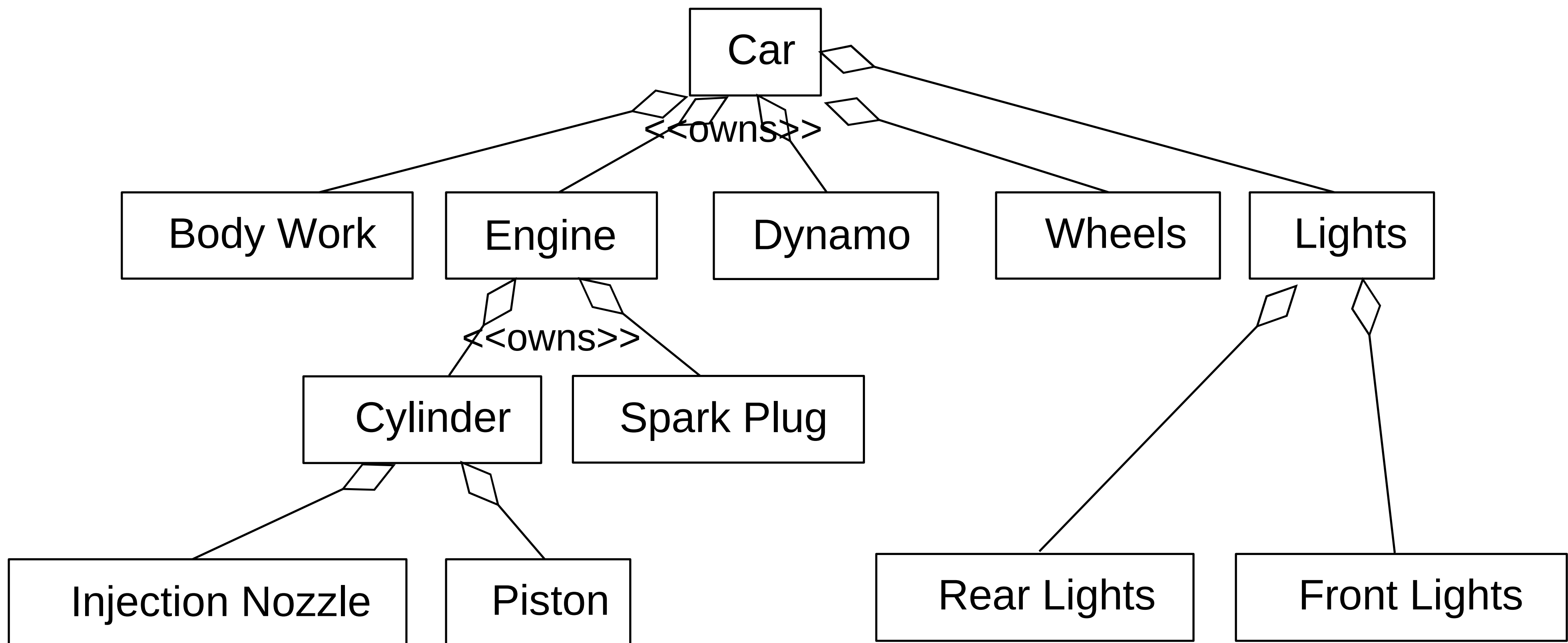
- Das Teilen von Teilen erzeugt gerichtete azyklische Graphen, die schichtbar sind (*has-a* Relation)



© Prof. U. Aßmann

Teile-Verfeinerung durch hierarchische Systemzerlegung mit statisch bekannter Anzahl von privaten Teilen

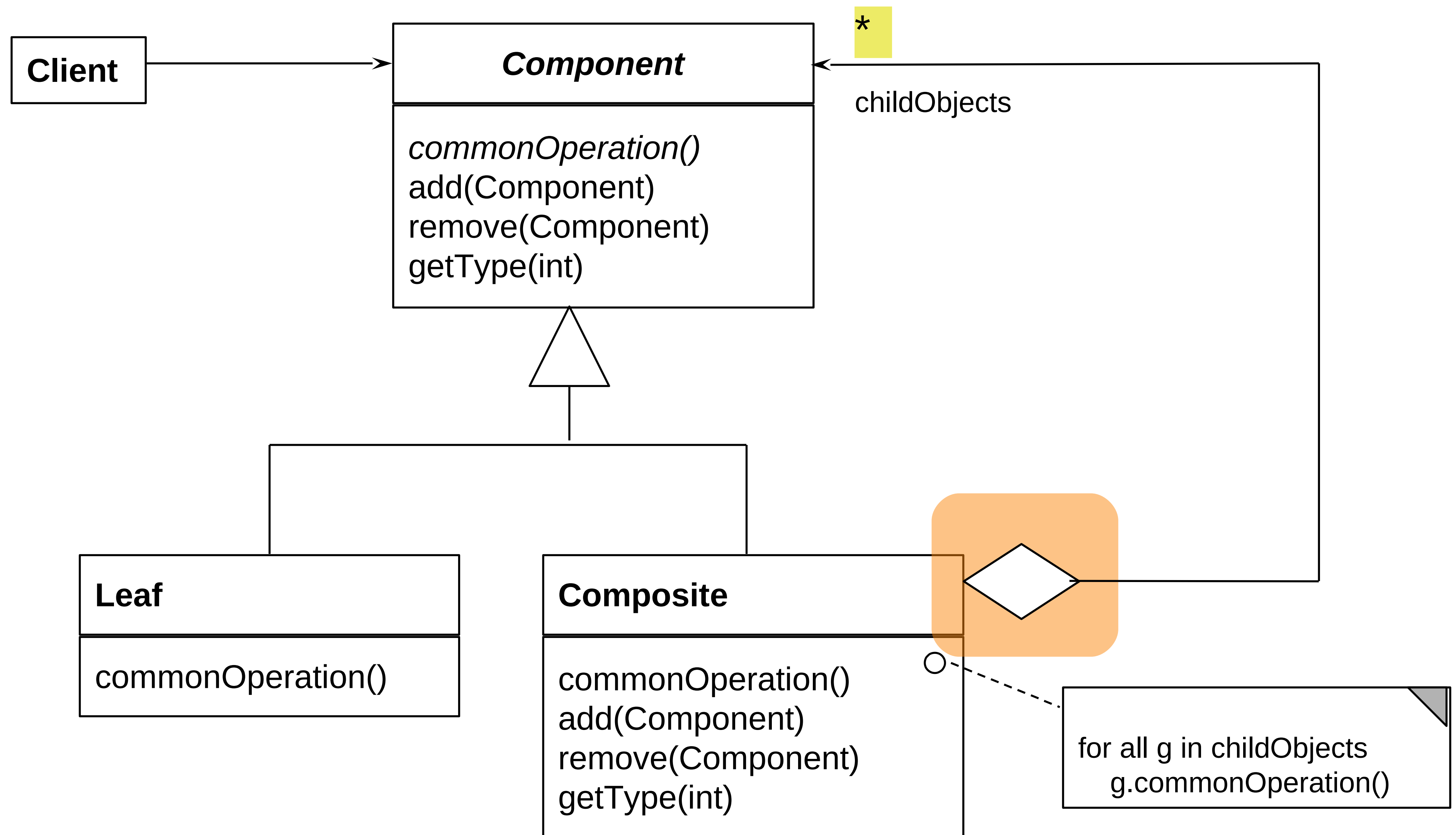
- **Teile-Verfeinerung** beginnt mit den komplexen Ganzen
  - .. und findet Schritt für Schritt neue Teile



© Prof. U. Aßmann

Unbekannt viele Teile
Achtung: Welche Aggregation steht im Composite?

- Welchen Unterschied macht es, ob die Kinder eines Composite-Knoten aggregiert, komponiert, oder rollenspielend sind?



© Prof. U. Aßmann

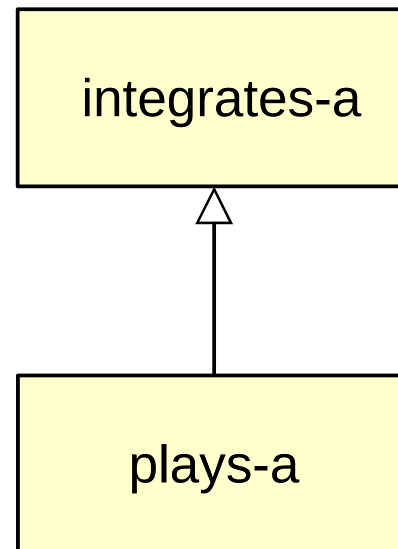
Folgen für Informationssysteme

- Materialien in Informationssystemen sind hierarchisch strukturiert

Die genaue Analyse der Ganz/Teile-Beziehung ermöglicht es, Aussagen über die Lebenszeit von Teilen in komplexen Geschäftsobjekten zu treffen.

© Prof. U. Aßmann

34.3 Natürliche Typen und Rollen



Verschiedene Arten von integrierten Unterobjekten

- Die verschiedene Arten von Unterobjekten ergeben sich aus einer Matrix von Typqualitäten
  - Hier: Überblick über das Folgende

	Nicht-Fundiert	Fundiert
Rigide	Natürlicher Typ Kern Facette Privates Teil	
Nicht-rigide	Phase	Rolle

© Prof. U. Aßmann

Rigide Typen

Besitzt ein Objekt einen **rigiden Typ**, stirbt es, sobald es die Typeigenschaft verliert [N. Guarino]

- Beispiele:
  - *Buch* ist ein rigider Typ
  - *Leser* ist ein nicht-rigider Typ
  - Ein Leser kann aufhören zu lesen, aber ein Buch bleibt ein Buch
  - Weitere rigide Typen: Begriffe der realen Welt: *Person*, *Car*, *Chicken*
- Rigidität ist eine Typqualität:
  - Rigide Typen sind verknüpft mit der *Identität* der Objekte
  - Nicht-rigide Typen sind dynamische Typen, die den Zustand eines Objektes anzeigen

© Prof. U. Aßmann

Fundierte Typen

Ein **fundierter Typ (kontextbezogener Typ)** beschreibt Eigenschaften eines Objektes, die in Abhängigkeit von einem Kooperationspartner bestehen

- Beispiel:
  - *Buch* ist ein nicht-fundierter Typ
  - *Leser* ist ein fundierter Typ
  - Ein Leser ist nur ein Leser, wenn er ein Buch liest (kontextbasiert), während ein Buch ein Buch ist, auch wenn es niemand liest
- Fundiertheit ist eine Typqualität, die Kontextbezug eines Typs ausdrückt

© Prof. U. Aßmann

Natürliche Typen und Rollen

Ein ***natürlicher Typ*** ist ein nicht-fundierter und rigider Typ.

Ein ***Rollentyp (Fähigkeit)*** ist ein fundierter und nicht-rigider Typ.

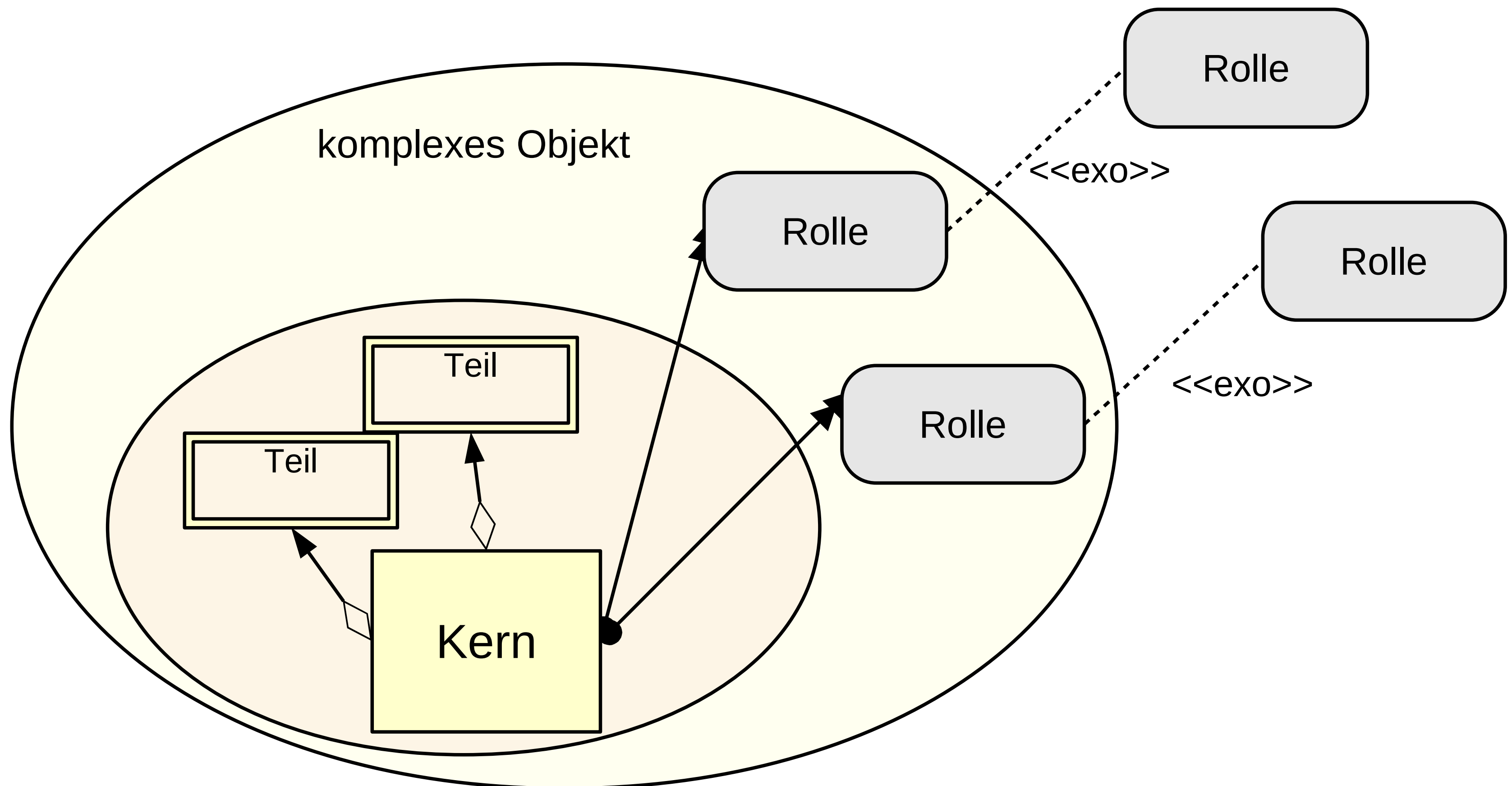
- **Rollentypen (Fähigkeiten)** existieren nur in Abhängigkeit von einer Kollaboration.
  - Ein Rollentyp entspricht also einer partiellen Klasse
  - in UML: **Role (or role type)**: “The named set of features defined over a collection of entities participating in a particular context.”

	Nicht-Fundiert	Fundiert
Rigide	(* natürlich *) Kern, Facette, privates Teil	
Nicht rigide	Phase	Rolle

© Prof. U. Aßmann

Komplexe Objekte mit Rollen

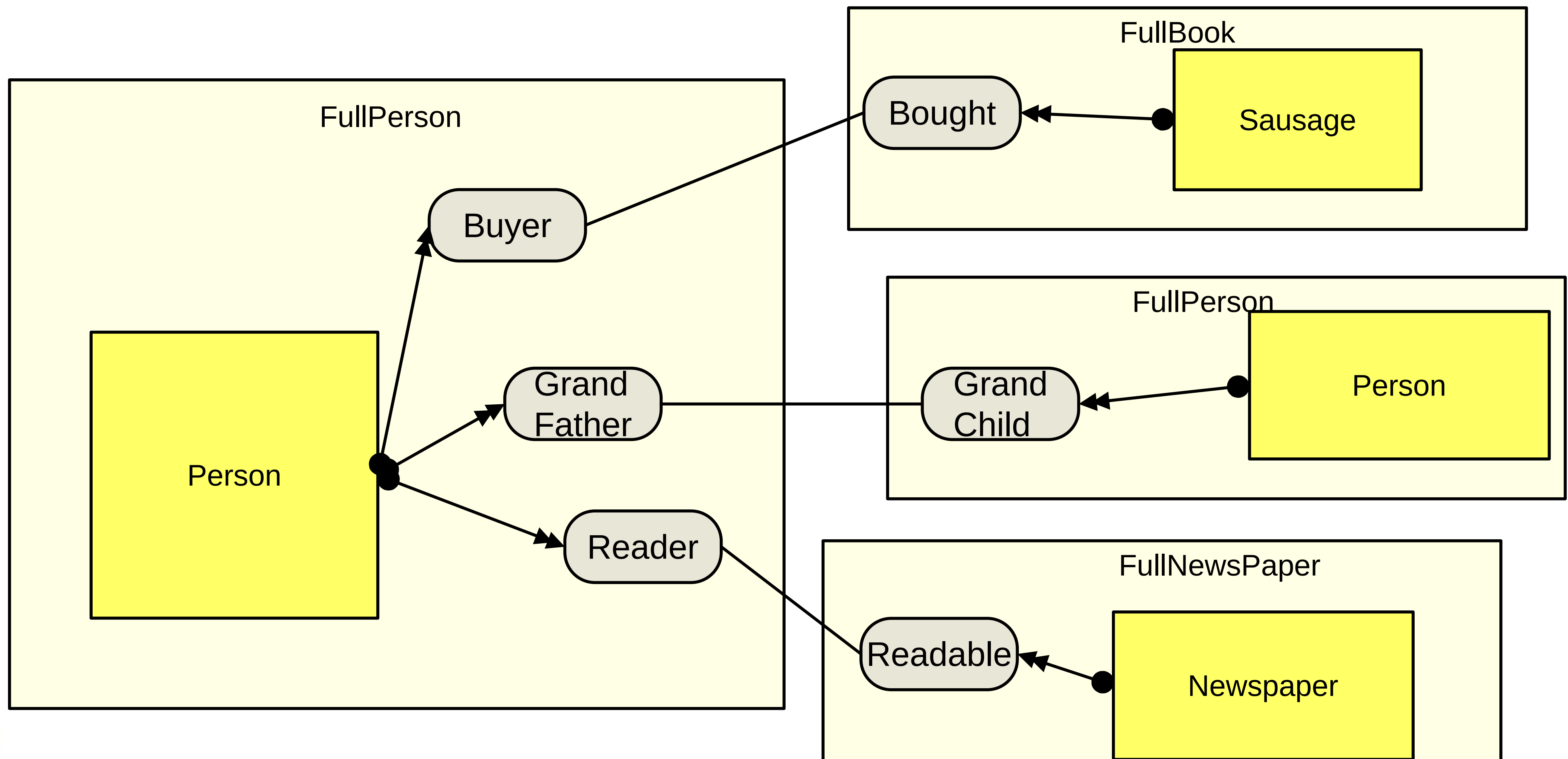
- Man sagt: ein Objekt *spielt eine Rolle (object plays a role)*
  - Ein Rollentyp (Fähigkeit) wird zur Laufzeit durch ein kontextbezogenes Unterobjekt (Rollensatellit) repräsentiert, das in Abhängigkeit von einem Kooperationspartner existiert
  - Rollen sind abhängige Teile, d.h. sie leben nur solange wie das Kernobjekt



© Prof. U. Aßmann

Die Steimann-Faktorisierung [Steimann]

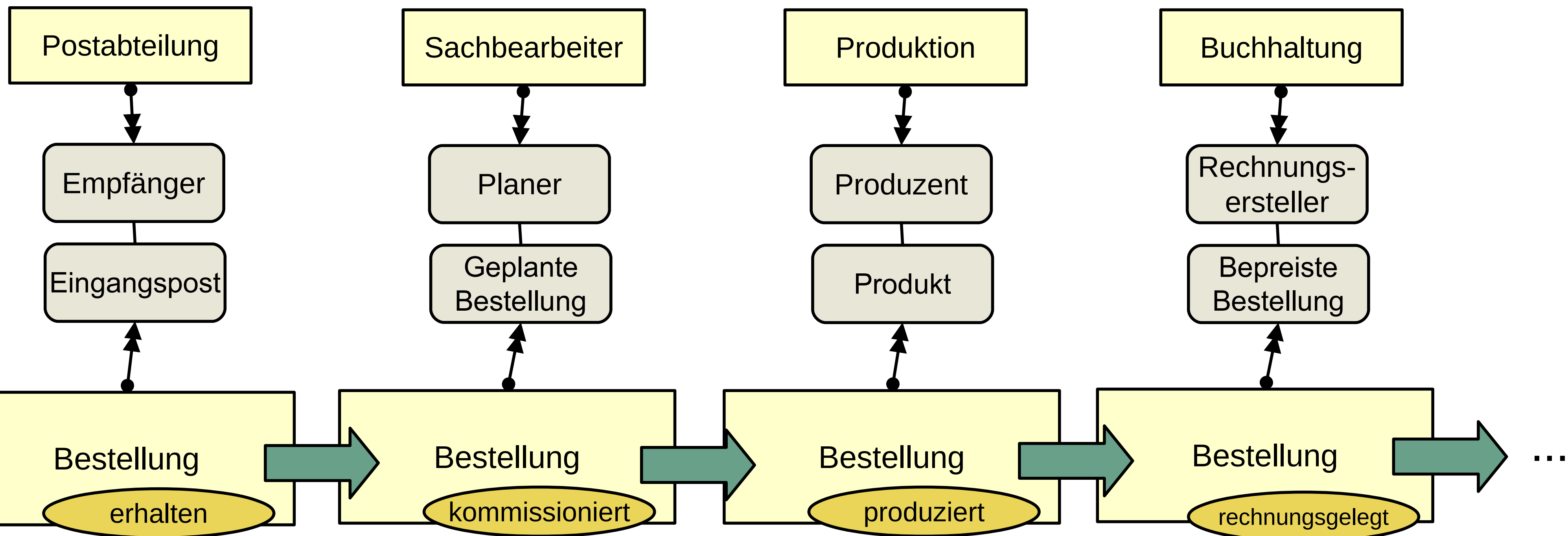
- Man teilt den Typ eines Objektes in seine *natürlichen Kern* und seine *Rollen-Unterobjekte* auf
  - FullType = Natural-type x (role-type, role-type, ...)
  - Bsp.: FullPerson = Person x (Reader, Father, Customer, ..)



© Prof. U. Aßmann

Rollen in Geschäftsobjekten (Business Objects)

- In Geschäftsobjekten kommen immer Rollen-Unterobjekte vor
- `Bestellung` als Beispiel: eine Bestellung durchwandert mehrere Bearbeiter
  - Auftragseingang des Kunden, Produktion, Kommissionierung, Rechnungserstellung, Auslieferung und Mahnwesen
- Dynamische Erweiterbarkeit bzw. Adaption durch neue bzw. wechselnde Rollen-Unterobjekte nötig:



© Prof. U. Aßmann

34.3 Realisierung von Teilen und Rollen

.. mit dem Bridge-Muster

Polymorphie ergibt sich aus dem Wechsel von Rollen-Unterobjekten

Programm PersonRoles.java

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

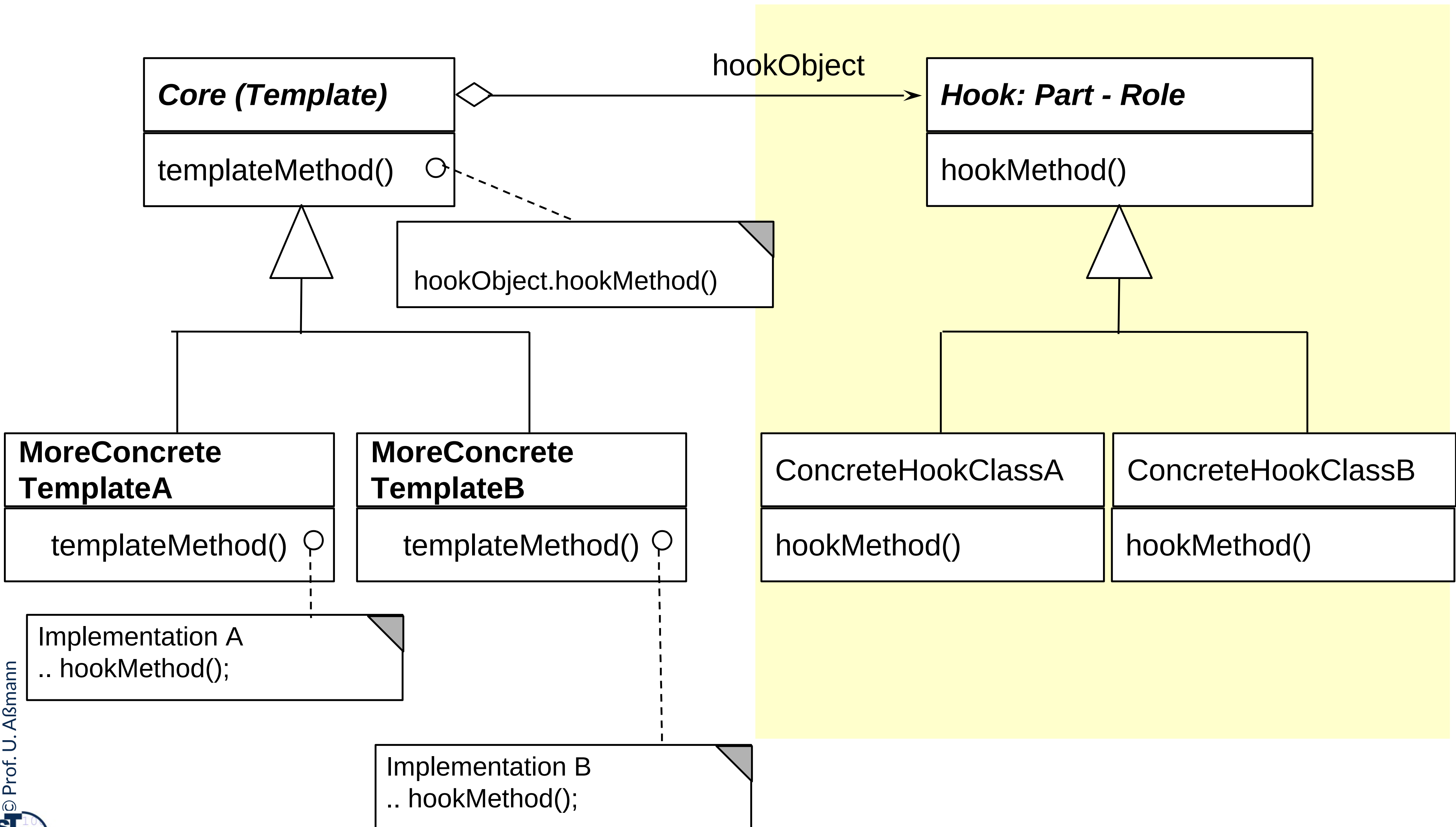
Klassische Realisierung: Unterobjekte mit Bridge-Muster

- Das Bridge-Muster bildet ein Großobjekt ab
  - Abstraction layer (Kern)
  - Implementation layer (Unterobjekt)
- Das Multi-Bridge-Muster enthält weitere Unterobjekte, d.h. mehrere Bridge
- Welche Arten von Unterobjekten gibt es?

© Prof. U. Aßmann

Rept.: Bridge and Dimensional Class Hierarchies

- Bridge can implement parts and roles easily

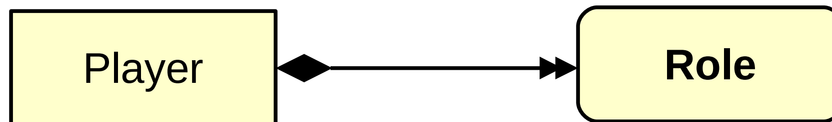


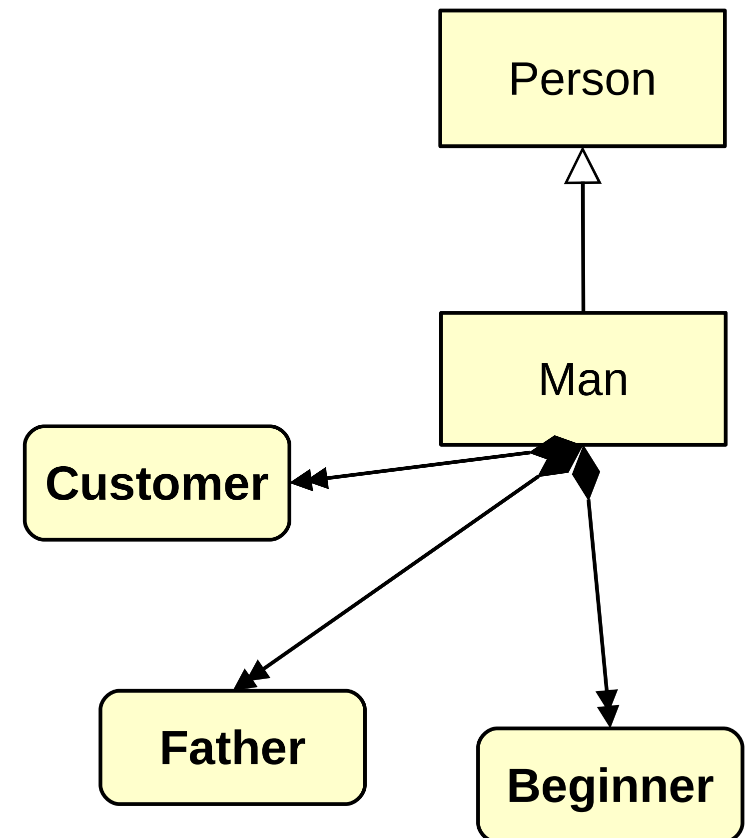
© Prof. U. Aßmann

Spezielle Icons für Stereotypen von Aggregation in Brücken

- Für alle grafischen Elemente in UML können spezielle Icone vereinbart werden.
- Für Rollenklassen wählen wir ein Oval, für die Plays-a-Relation einen Pfeil mit Doppel-Kopf-Dreieck.

Abgekürzt:

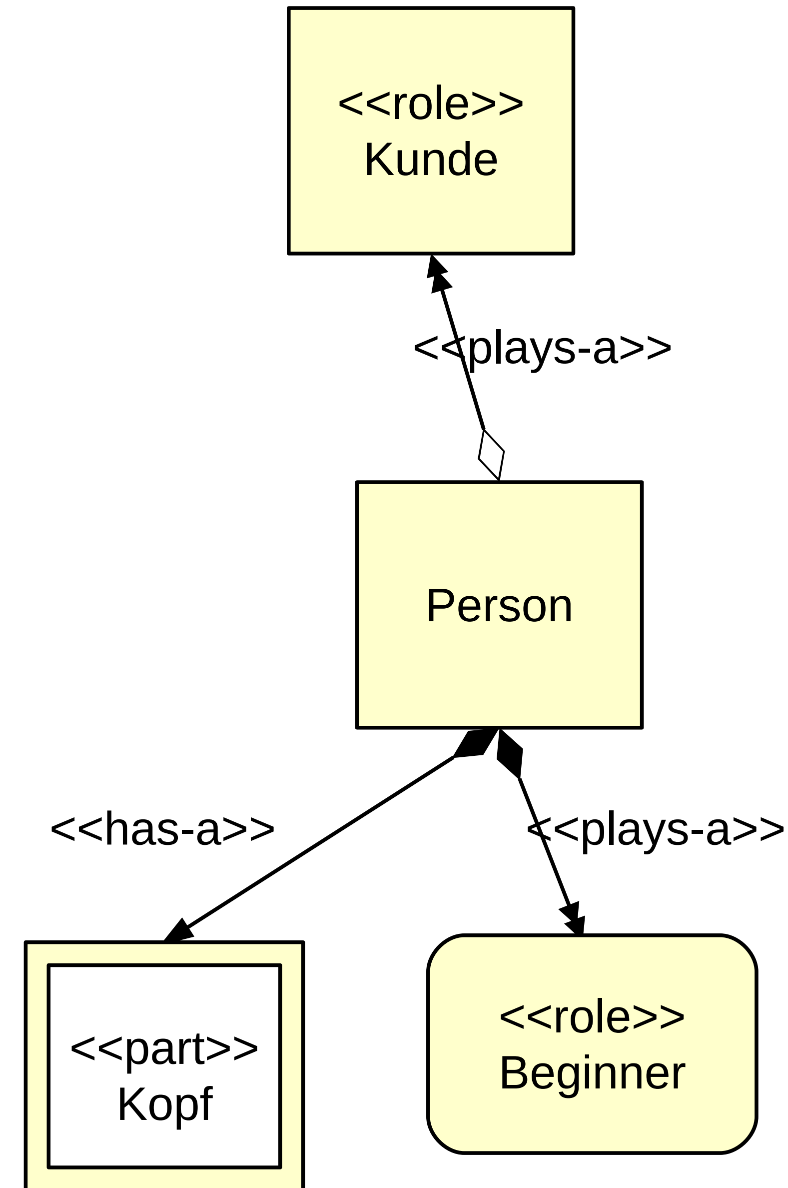




© Prof. U. Aßmann

Unterobjekte

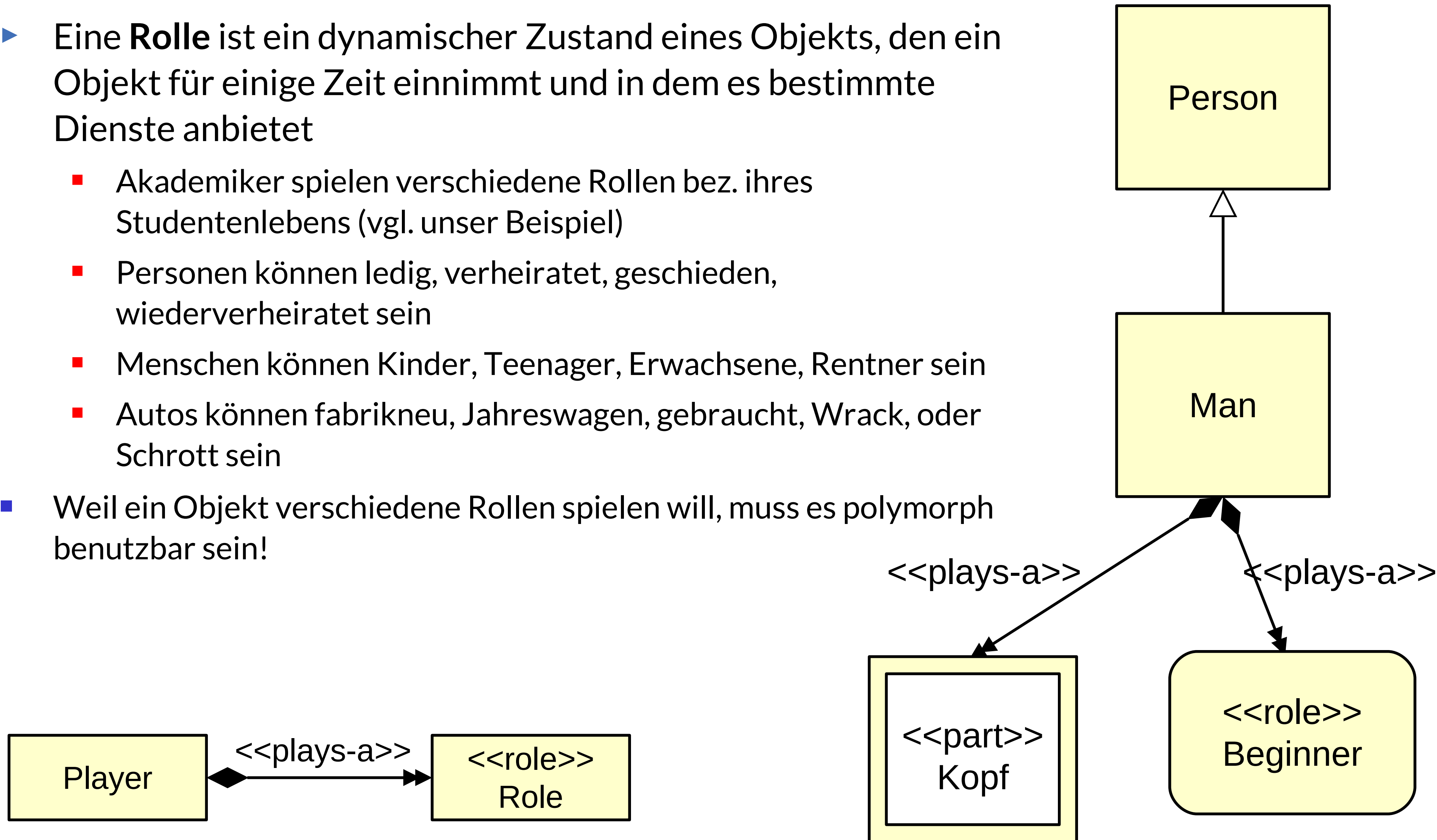
- **Großobjekt:** Manche Objekte sind sehr groß und leben lang. Sie werden daher in Unterobjekte gegliedert.
- Ein **Unterobjekt** gehört semantisch zu einem **Kernobjekt**
  - teilt also die Identität des Kernobjektes
  - muss diese auf die Frage “Wer bist du?” melden und *nicht den eigenen Identifikator*
  - Hier:
    - ein `Kopf` muss den Identifikator von `Person` melden, *nicht seinen eigenen*
    - ein `Beginner` muss den Identifikator von `Person` melden, *nicht seinen eigenen*



© Prof. U. Aßmann

Rollen als Grund für Polymorphie

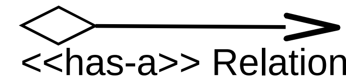
- Polymorphie ist gerade dann ein wichtiges Konzept, wenn Objekte temporär **Rollen spielen**
- Eine **Rolle** ist ein dynamischer Zustand eines Objekts, den ein Objekt für einige Zeit einnimmt und in dem es bestimmte Dienste anbietet
  - Akademiker spielen verschiedene Rollen bez. ihres Studentenlebens (vgl. unser Beispiel)
  - Personen können ledig, verheiratet, geschieden, wiederverheiratet sein
  - Menschen können Kinder, Teenager, Erwachsene, Rentner sein
  - Autos können fabrikneu, Jahreswagen, gebraucht, Wrack, oder Schrott sein
- Weil ein Objekt verschiedene Rollen spielen will, muss es polymorph benutzbar sein!

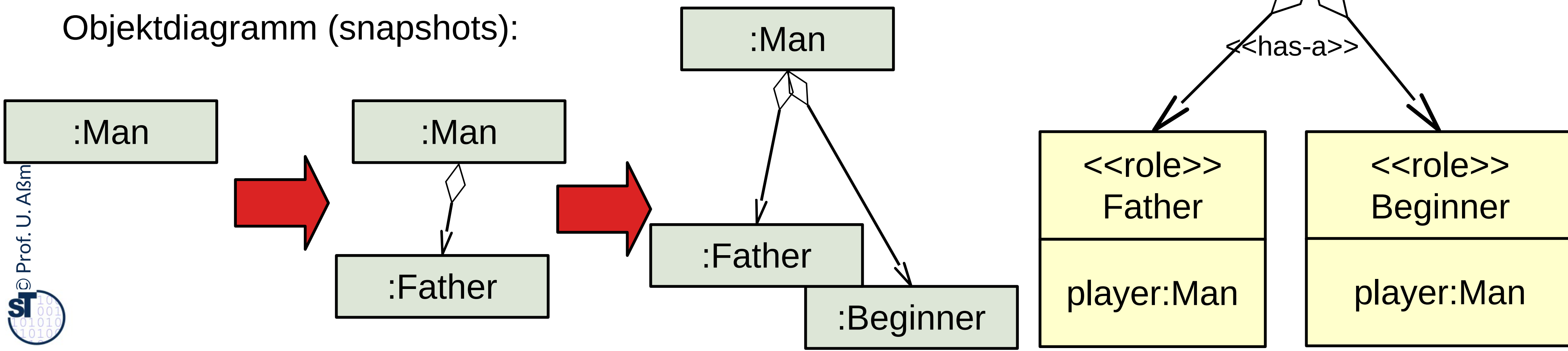


© Prof. U. Aßmann

Implementierungsmuster: “Rollen mit Bridge”

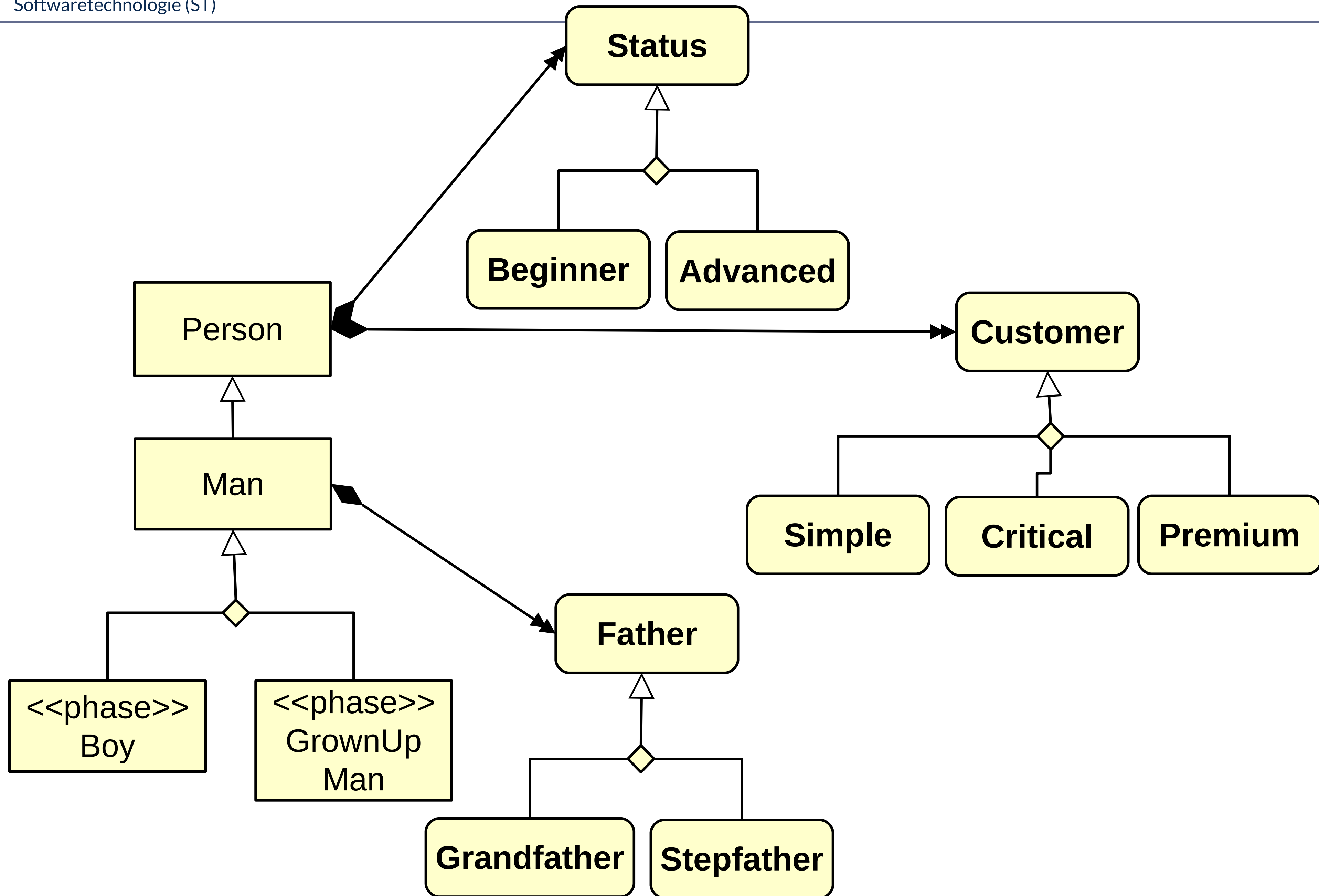
- In Java gibt es leider kein Sprachkonzept “Rolle”
- Rollen können durch ***Unterobjekte*** eines ***Kernobjektes*** modelliert werden (Bridge-Muster, Implementierung von *plays-a* durch *has-a*, Aggregation)
  - Vorteil: Kernobjekt kann viele Rollen spielen
  - Vorteil: Referenzen auf Kernobjekt bleiben bei Rollenwechsel erhalten!
- Man implementiert den Wechsel einer Rolle durch den Wechsel der entsprechenden Unterklasse des Rollen-Unterobjektes durch
  - Alloziere und fülle neues Rollen-Unterobjekt
  - Setze Variable um auf neues Objekt
  - Dealloziere altes Rollen-Unterobjekt





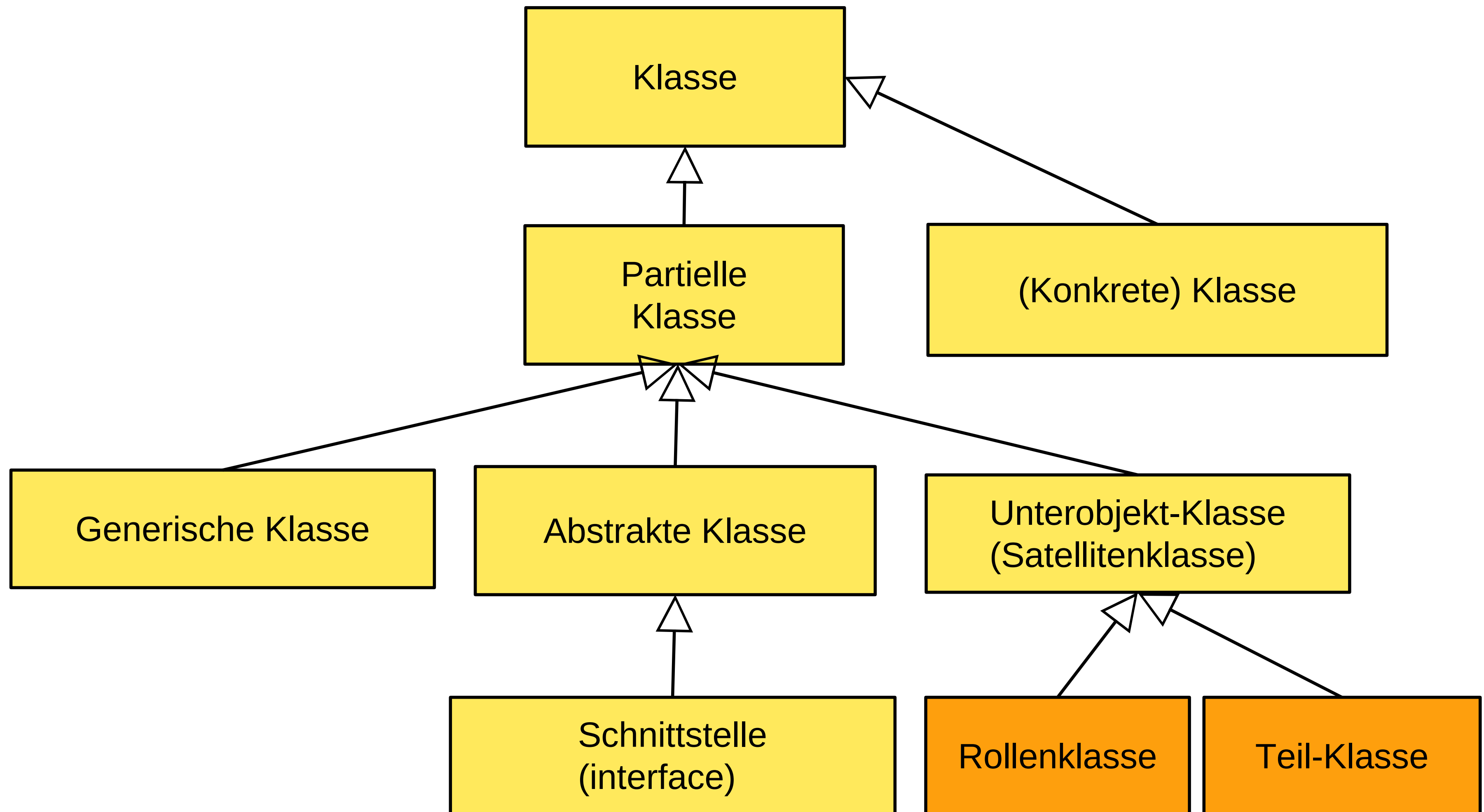
© Prof. U. Aßmann

Variation von Rollen mit Multi-Bridge



© Prof. U. Aßmann

Begriffshierarchie von Klassen (Erweiterung)

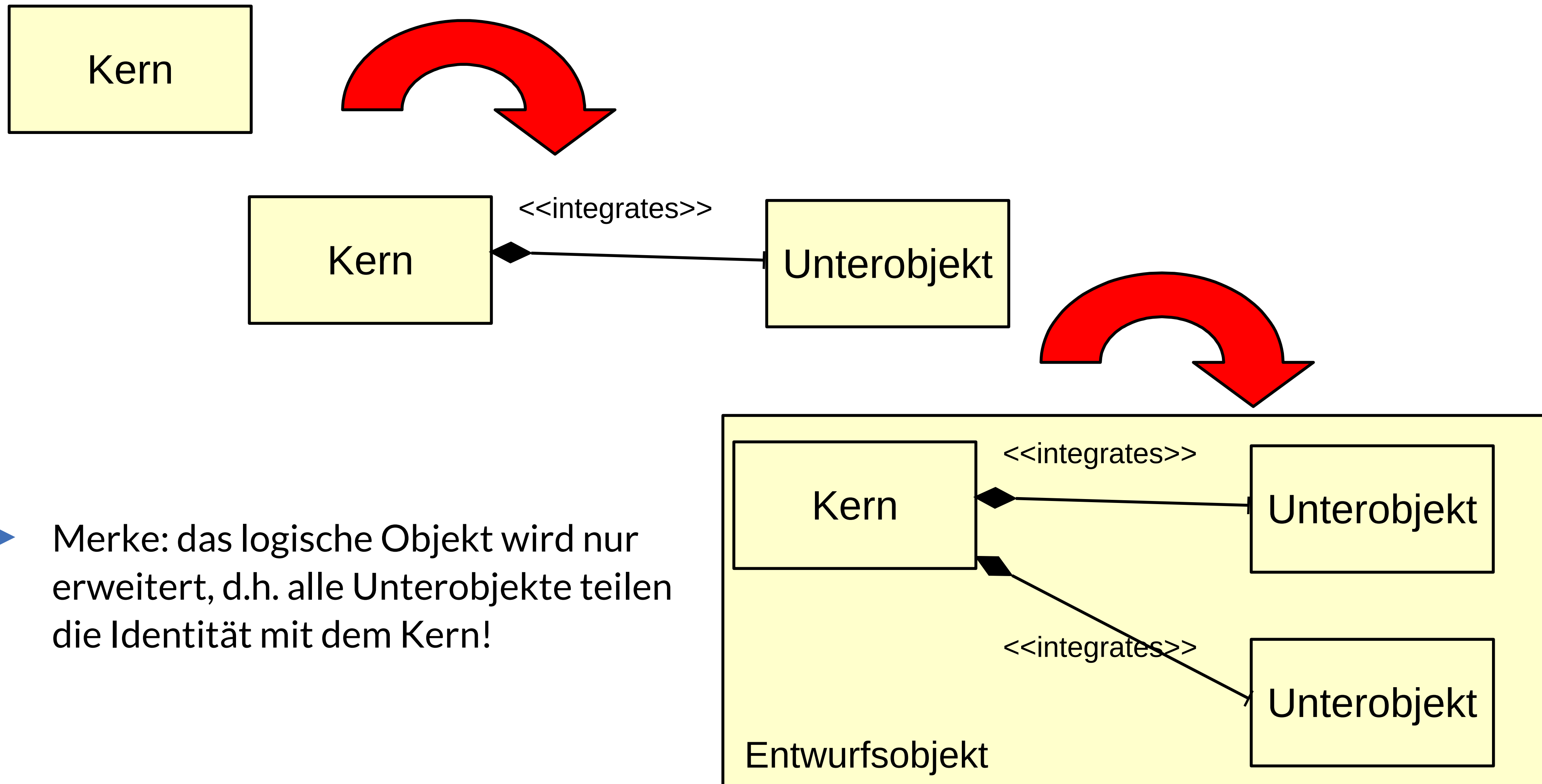


© Prof. U. Aßmann

34.4 Objektanreicherung durch Teile und Rollen

Integration von Unterobjekten in Analyseobjekte zu komplexen Objekten

- **Integration von Unterobjekten** besteht aus der Anreicherung von Analyseobjekten aus dem Domänenmodell

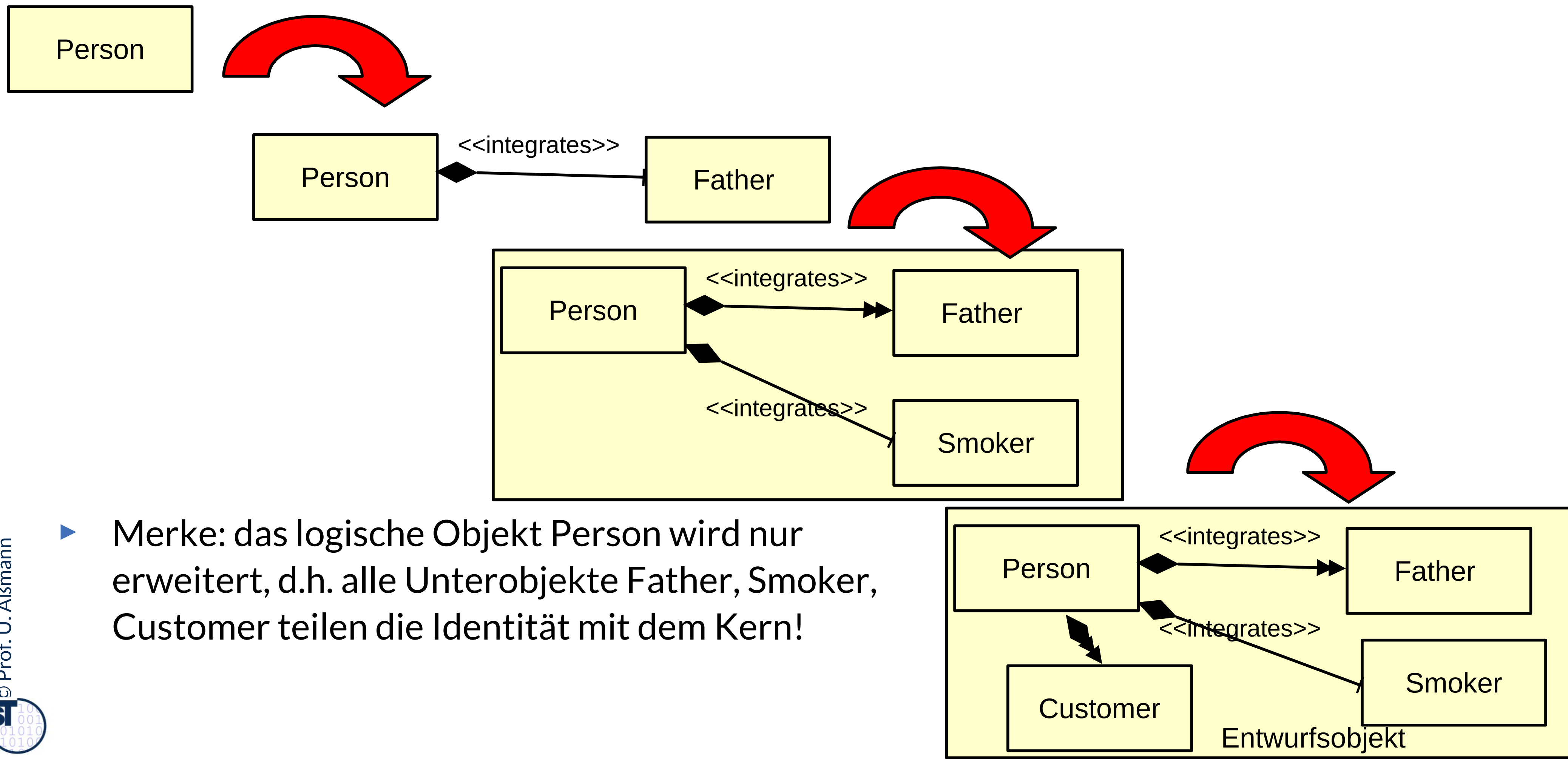


- Merke: das logische Objekt wird nur erweitert, d.h. alle Unterobjekte teilen die Identität mit dem Kern!

© Prof. U. Aßmann

Beispiel: Integration von Unterobjekten in Personen

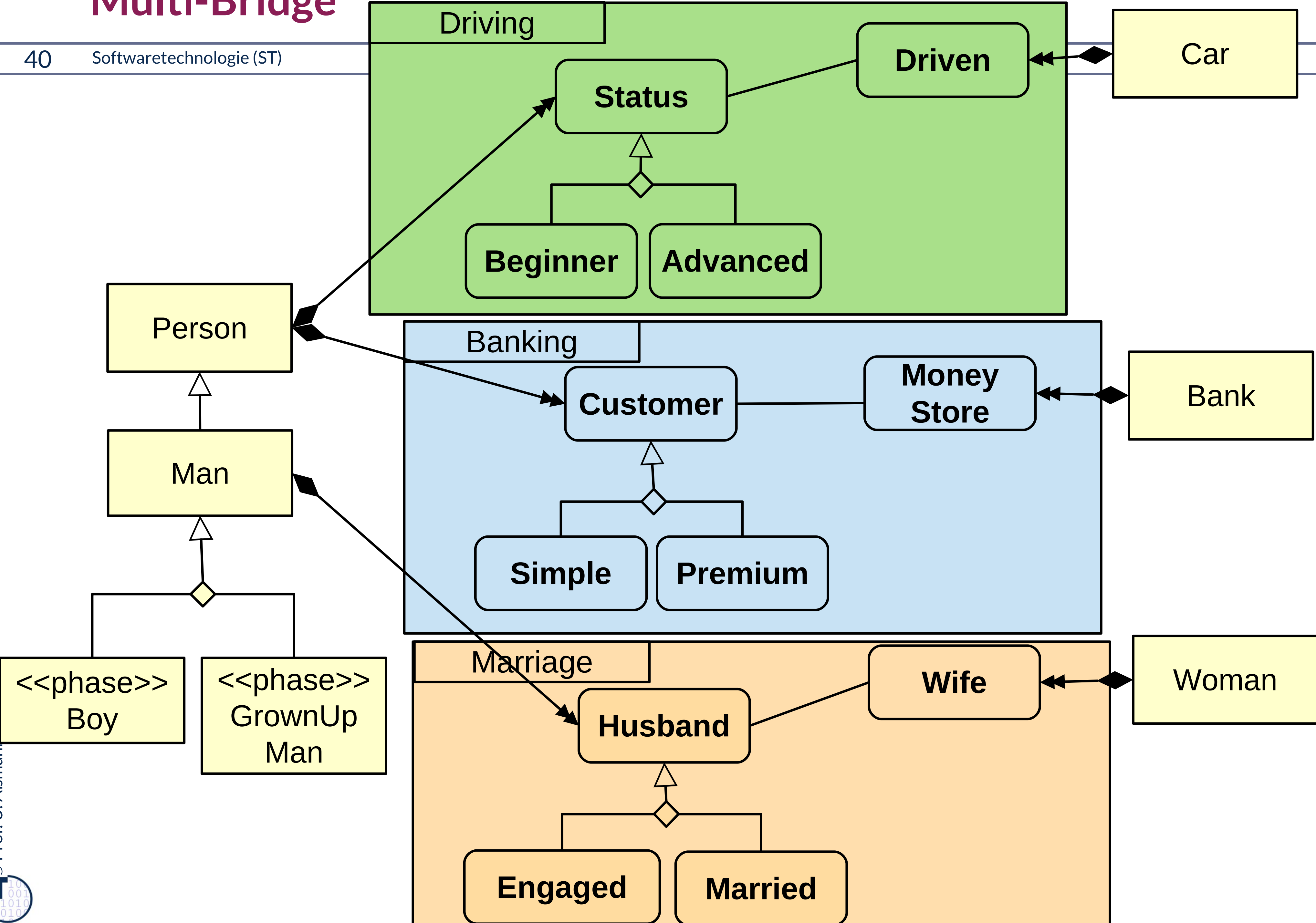
- Die Modellierung von Personen ist ein wesentliches Problem vieler Anwendungen.
- Hier kann mit der **Integration von Unterobjekten** an einen Personen-Kern zusätzliche Funktionalität modelliert werden



- Merke: das logische Objekt Person wird nur erweitert, d.h. alle Unterobjekte Father, Smoker, Customer teilen die Identität mit dem Kern!

© Prof. U. Aßmann

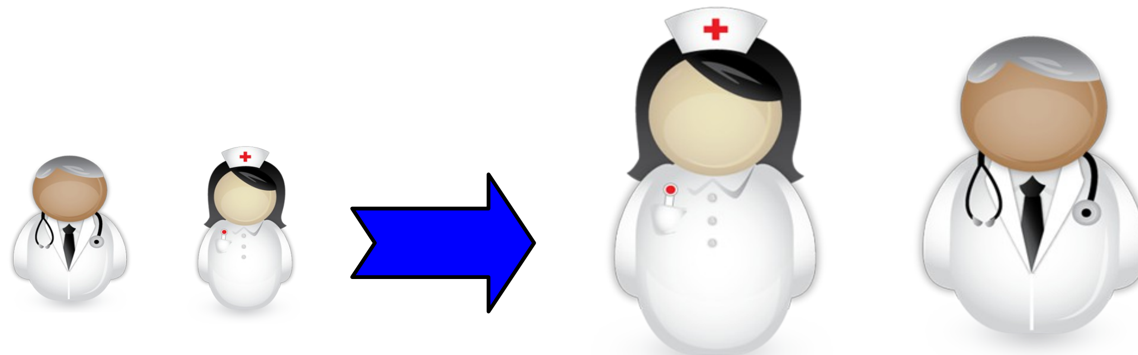
Beipiel: Querschneidende Erweiterung von Objekten mit Multi-Bridge



© Prof. U. Aßmann

Objektanreicherung (Object Fattening)

- **Objektanreicherung (Object fattening, Objektverfettung)** ist ein Verfeinerungsprozess zur *Entwurfszeit*, der an ein *Kernobjekt aus dem Domänenmodell* Unterobjekte anlagert (Domänenobjekt-Verfeinerung durch Integration), die Unterobjekte ergänzt, die Beziehungen klären zu
  - Plattformen (middleware, Sprachen, Komponenten-services)
  - Komponentenmodellen (durch Adaptergenerierung)
- Ziel: Ableitung von Entwurfs- und Implementierungsobjekten
- Merk-Brücke “object fattening”:
  - Objekte (“Hühner”, “Ärzte”) aus dem Domänenmodell werden Stück für Stück mit Implementierungsinformation in Unterobjekten verfettet
  - Die Objekte aus dem Domänenmodell bleiben erhalten, werden aber immer dicker
- Objektanreicherung geschieht mit dem Entwurfsmuster Brücke. Jede Anreicherung erzeugt eine Brücke, insgesamt ergibt sich eine Multi-Bridge



© Prof. U. Aßmann

Arten von Verfeinerung durch Integration von Unterobjekten (Object fattening)

Bei der Objektanreicherung können verschiedene Arten von Unterobjekten zur Verfeinerung benutzt werden:

- **Teileverfeinerung (Teileanlagerung):**
  - Finden von privaten Teilen von komplexen Analyseobjekten, die integriert werden
  - Schichtung ihrer Lebenszyklen
- **Rollenverfeinerung (Rollenanlagerung)**:
  - Finden von **Konnektoren (teams, collaborations)** zwischen Anwendungsobjekten
- Optional:
  - **Facettenverfeinerung (Facettenanlagerung)**: Finden von Facetten-Unterobjekten, die Mehrfachklassifikationen ausdrücken
  - **Phasenverfeinerung (Phasenanlagerung):** Finden von Phasen-Unterobjekten, die Lebensphasen des komplexen Objektes ausdrücken

Unterobjekte (Satelliten) gehören immer zu einer dieser speziellen Kategorien.

© Prof. U. Aßmann

Informationssysteme

- Ein **Informationssystem** ist ein Softwaresystem, dessen primäre Aufgabe in der Information und Verwaltung von physischen oder immateriellen Materialien besteht (Produkte, Vorräte, Teile, Geld, Guthaben, etc.)
- Der Fluss der Materialien durch die Firma verändert die Materialien

In Informationssystemen modellieren Rollen die wechselnden Kontexteigenschaften von Materialien.

Bei der Entwicklung von Informationssystemen werden ihre Tools und ihre Materialien durch Mehrfach-Brücken querschneidend verfeinert, die Satelliten (Rollen und Teile) anlagern.

© Prof. U. Aßmann

Warum ist das wichtig?

- Auf dem Weg vom Analysemodell zum Implementierungsmodell entstehen *komplexe Objekte*
- Komplexe Objekte werden, durch mehrfache Anwendung des Entwurfsmusters Brücke, mit Satelliten erweitert (Objektanreicherung, Objektverfettung)
- Plattformportabilität:
  - Da Wiederverwendung ist das Hauptmittel der Softwarefirmen, um profitabel arbeiten zu können, erweitert man einen Entwurf zu *verschiedenen* Implementierungsmodellen, indem man verschiedene Brücken einzieht
- Funktionale Variabilität:
  - Will man verschiedenen Kundengruppen verschiedene Funktionalitäten anbieten, verfeinert man spezifisch für die Kundengruppen
  - Und erhält mehrere *Produktvarianten*
  - Und insgesamt eine *Produktlinie*

© Prof. U. Aßmann

Was haben wir gelernt?

- Es gibt komplexe Objekte, die so groß sind, dass sie aus Kern und Unterobjekten (Satelliten) bestehen
  - Komplexe Objekte sind immer hierarchisch oder azyklisch und verwenden Endo-Assoziationen. Sie werden oft mit Multi-Bridge realisiert
  - Der Kern bildet meist eine Facade für die Unterobjekte
  - Unterobjekte sind typisch einer Kategorie zugeordnet (Rollen, Teile, Facetten, Phasen)
- Teile-Klassen sind unvollständige Klassen
- Rollenklassen sind dynamische, unvollständige Klassen
- Einfache Implementierung mit Bridge und Multi-Bridge
- Objektanreicherung besteht darin, ein komplexes Objekt aus dem Analysemodell durch weitere Unterobjekte im Entwurf und in der Implementierung anzureichern
- Informationssysteme nutzen für die Repräsentation von Geschäftsobjekten Rollen und Hierarchien
  - Die objektorientierte Systementwicklung nutzt als hauptsächliches Mittel die Objektanreicherung, um vom Problem des Kunden zum Analysemodell, dann über das Entwurfs- und Implementierungsmodell zum Programm zu kommen.

© Prof. U. Aßmann

Anhang

- Wie implementiert man ein Großobjekt mit fester Anzahl von Unterobjekten?
- Was passiert zur Laufzeit in einer Multi-Bridge mit Rollen-Unterobjekten? Wie vermeidet man einen Absturz des Systems, wenn die Rolle nicht gespielt wird?

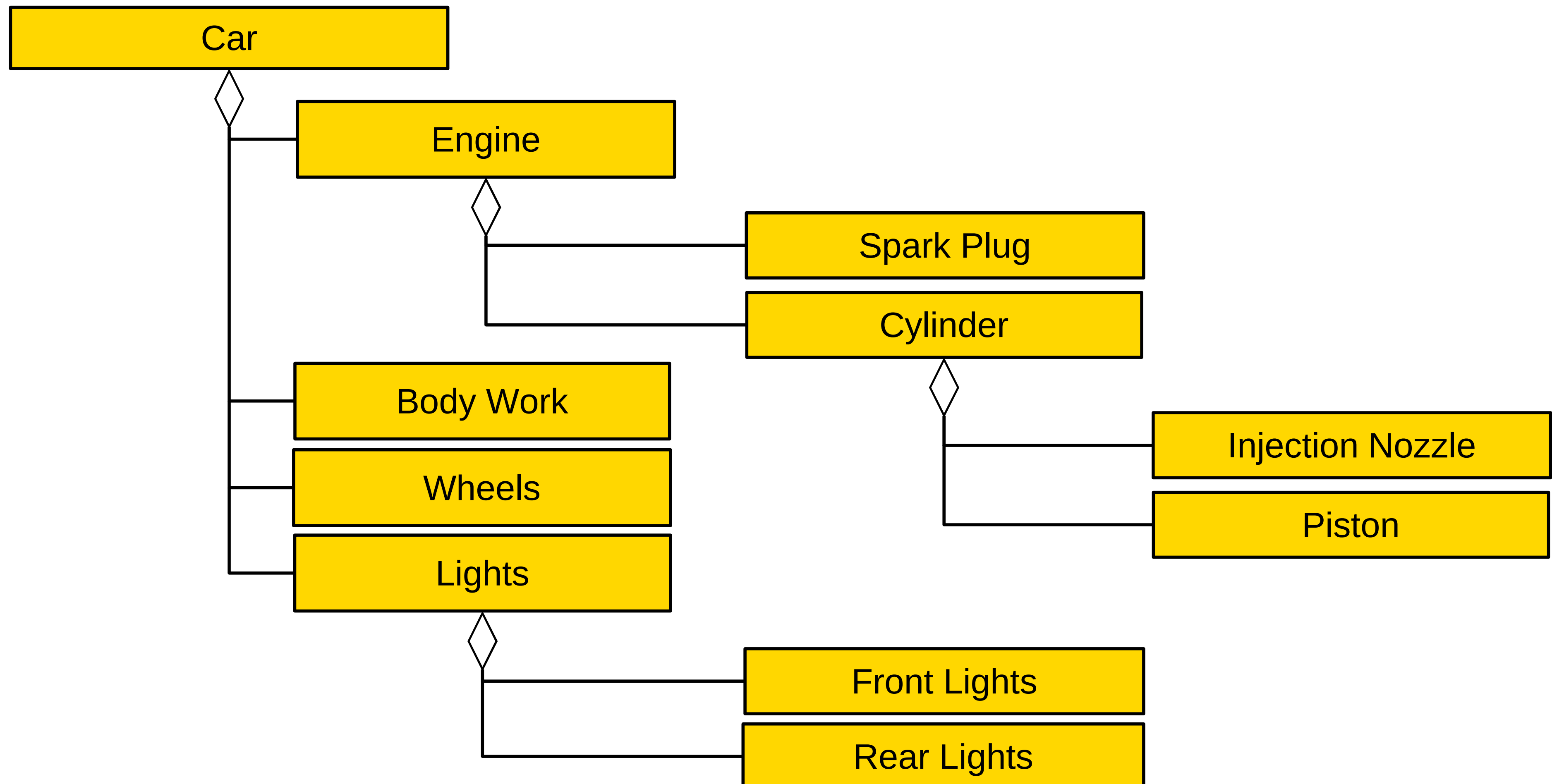
© Prof. U. Aßmann

Ende des obligatorischen Materials

© Prof. U. Aßmann

Darstellung komplexer Objekte mit privaten Teilen

- Komplexe Objekte können dargestellt werden
  - Mit Zeilenhierarchien
  - Mit Mind maps



© Prof. U. Aßmann

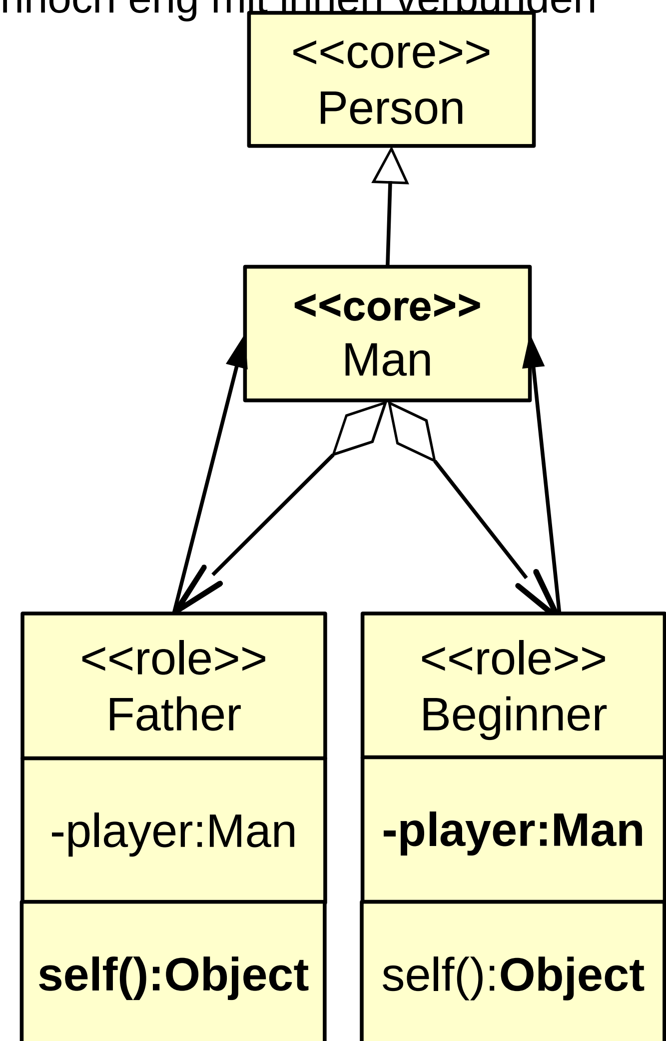
34.A.1 Weitere Realisierungen von Rollen-Unterobjekten

(optional, nicht klausurrelevant)

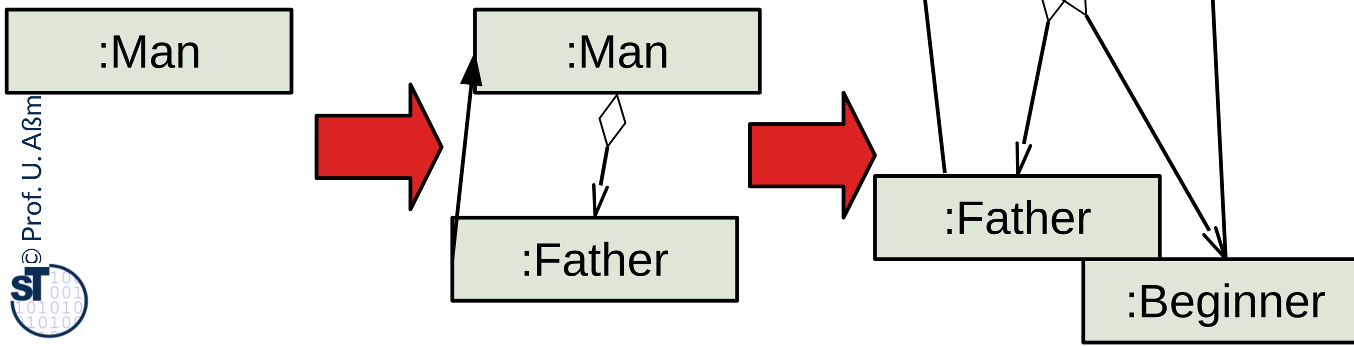
DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Implementierungsmuster: “Rollen mit Aggregation und Kern-Link”

- Rollen sollten zusätzlich durch einen Rückwärts-Link `player` mit dem Kernobjekt verbunden sein, da sie semantisch zu ihm gehören
  - Vorteil: Kernobjekt kann viele Rollen spielen und ist dennoch eng mit ihnen verbunden
  - **Identifikationsmethode**: self():Object



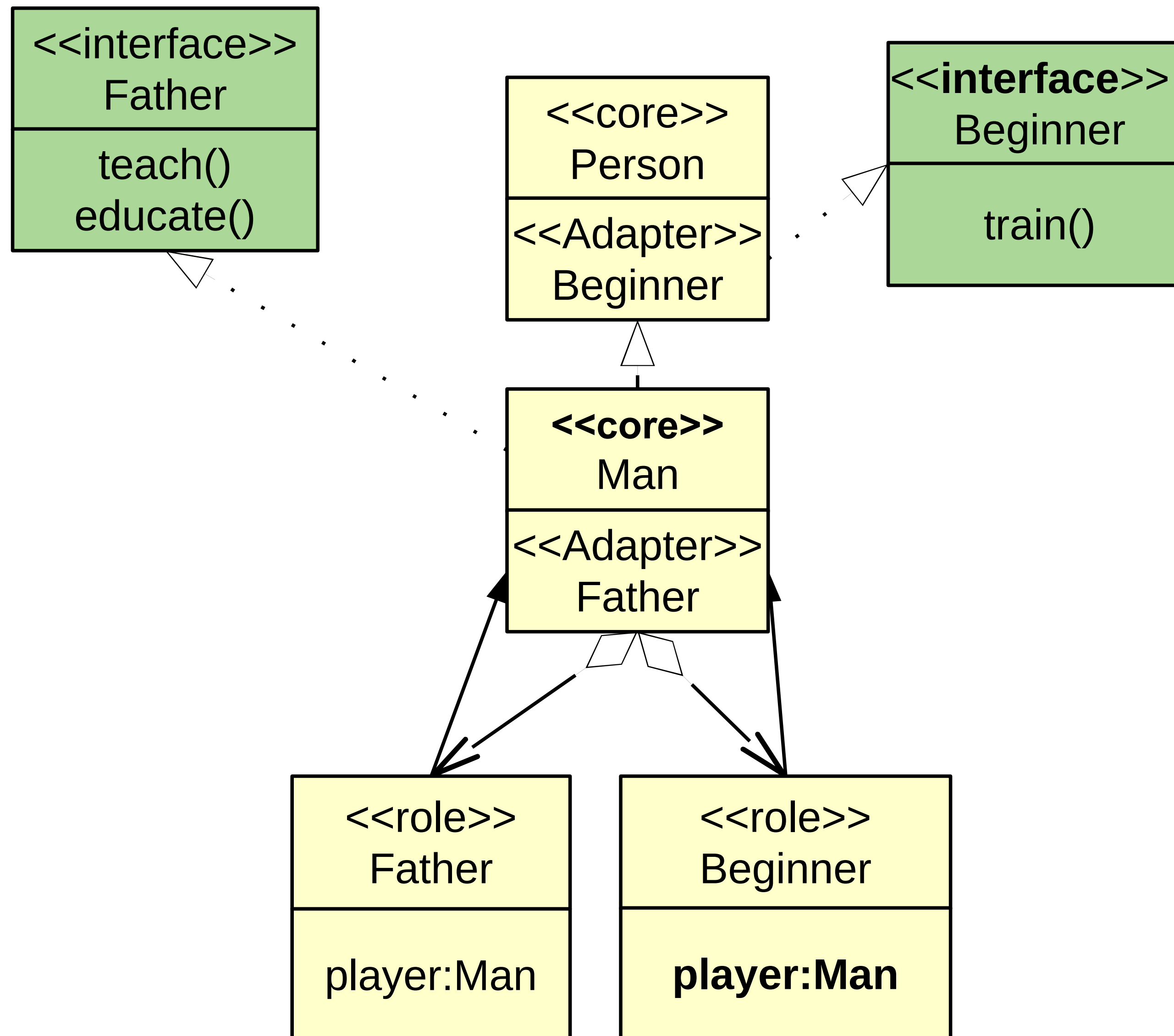
Objektdiagramm (snapshots):



© Prof. U. Aßmann

Implementierungsmuster: “Rollen mit Aggregation und Kern-Link”

- Rollen führen zu Schnittstellen



© Prof. U. Aßmann

Implementierungsmuster: “Rollen mit generischem Player-Link”

- Rollen können als generische Klassen gesehen werden, die als generischen Parameter den Player erwarten
  - Dann ist das Netz zwischen Kernobjekt und Rollen typsicher

Player

<<role>>
Father

player:Player

<<expanded-from>>

<<core>>
Person

<<core>>
Man

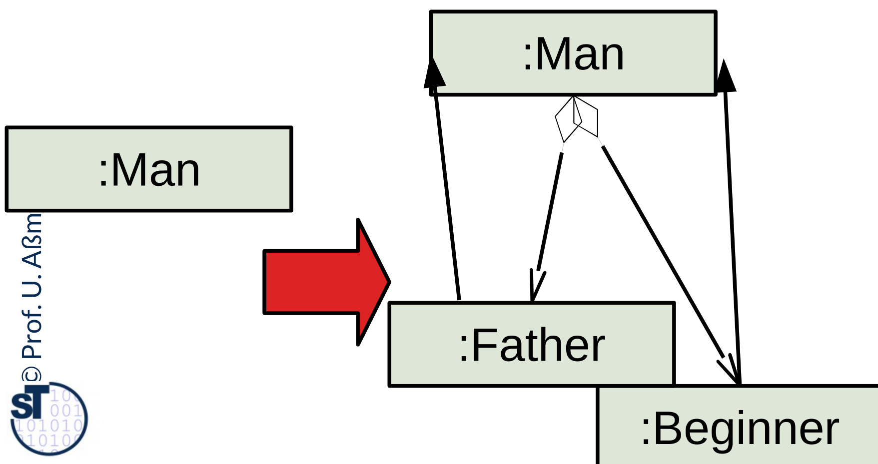
<<role>>
Father<Man>

player:Man

<<role>>
Beginner<Man>

player:Man

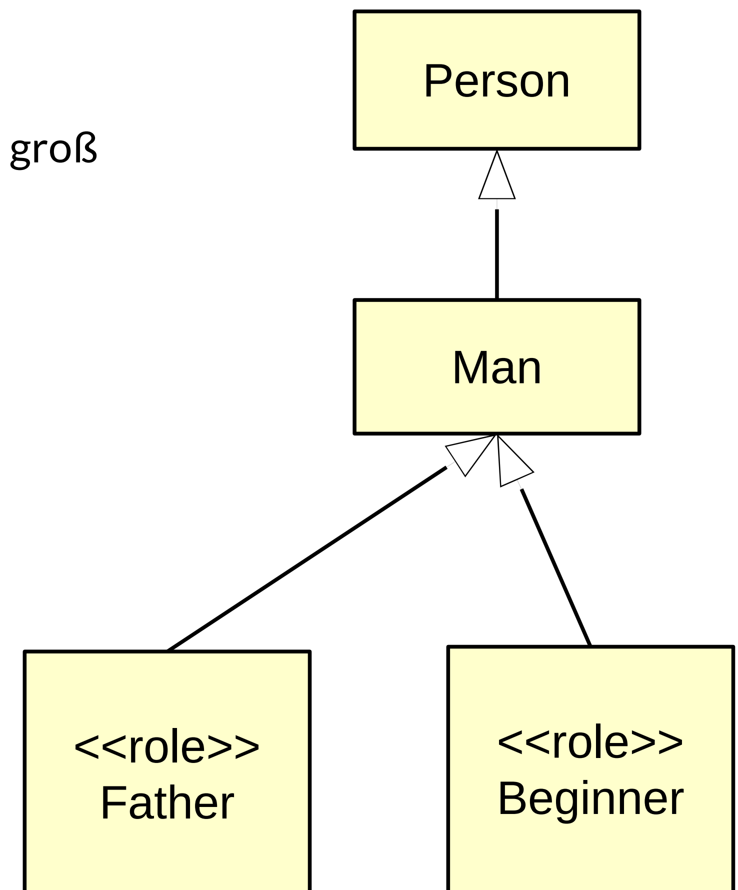
Objektdiagramm (typsicher):



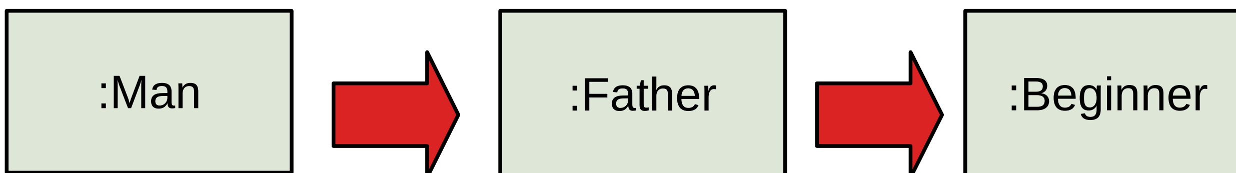
© Prof. U. Aßmann

Oft genutztes, aber ungeschicktes Implementierungs-muster: “Rollen als Unterklassen”

- Man kann mit Unterklassen die Rollen einer Oberklasse realisieren (“Implementierungsmuster”)
- Man implementiert den Wechsel einer Rolle durch den Wechsel der entsprechenden Unterklasse durch
  - Alloziere und fülle neues Objekt aus den Werten des alten Objektes heraus
  - Setze Variable um auf neues Objekt
  - (Dealloziere altes Objekt)
- Problem: bei vielen Kontexten werden die Objekte viel zu groß



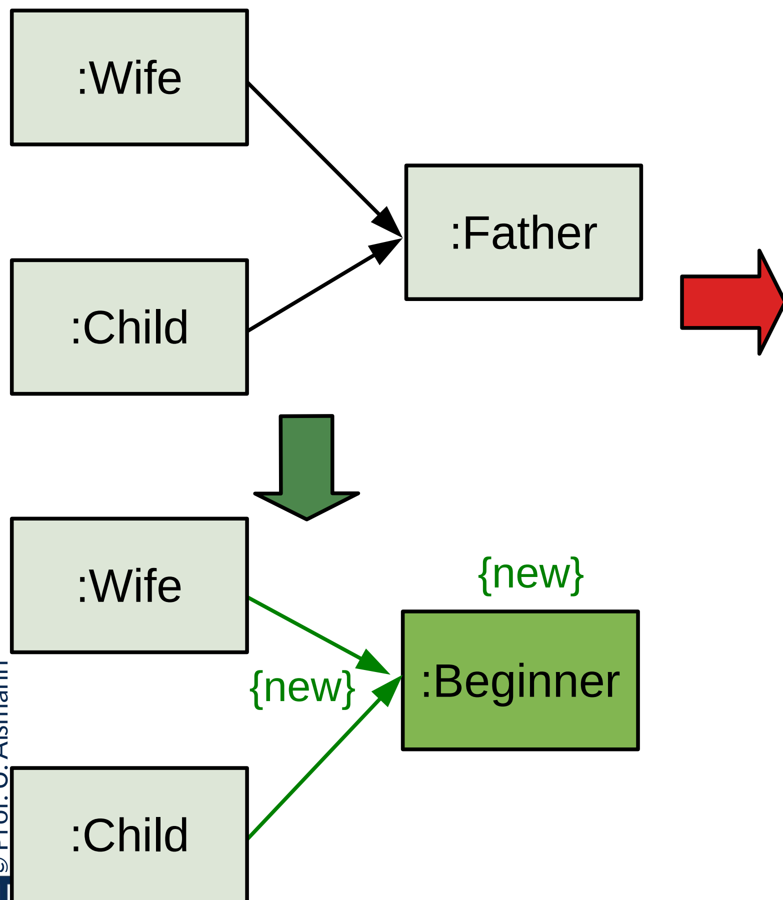
Objektdiagramm (snapshots):

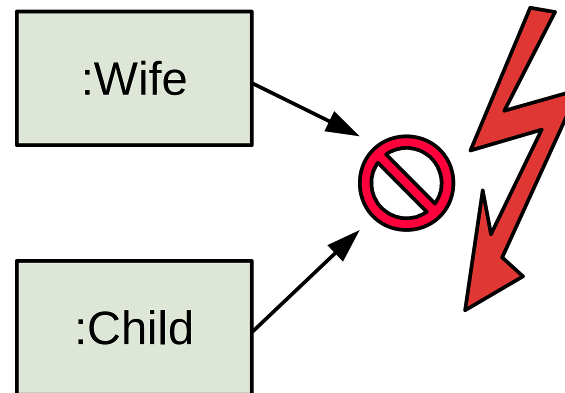


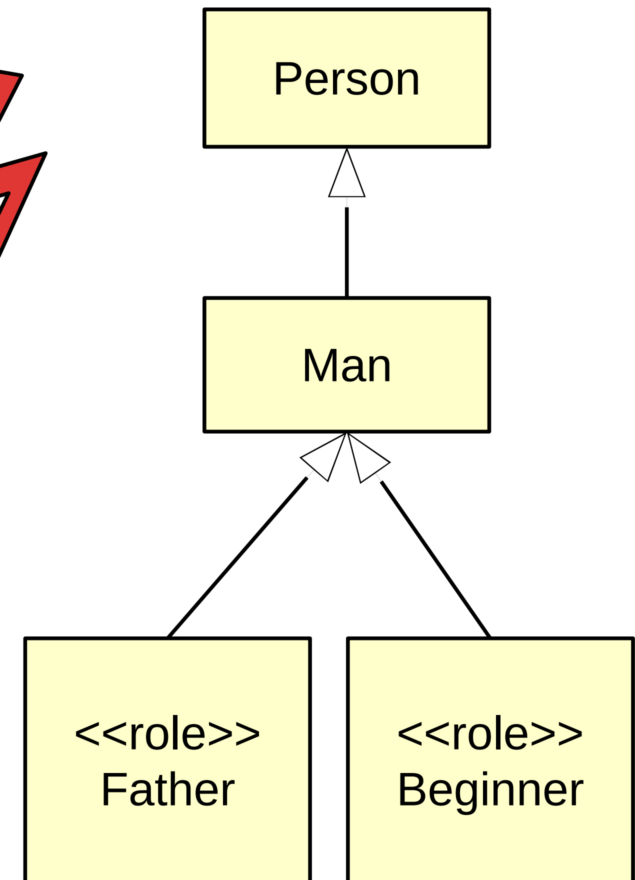
© Prof. U. Aßmann

“Nirvana” im Implementierungsmuster: “Rollen als Unterklassen”

- Problem: Referenzen auf Objekte müssen bei Rollenwechsel umgesetzt werden und hinterlassen “Nirvana-Kanten” (dangling edges)
- Einer der häufigsten Programmierfehler überhaupt



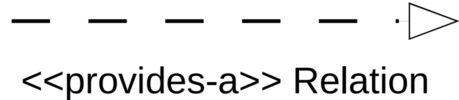


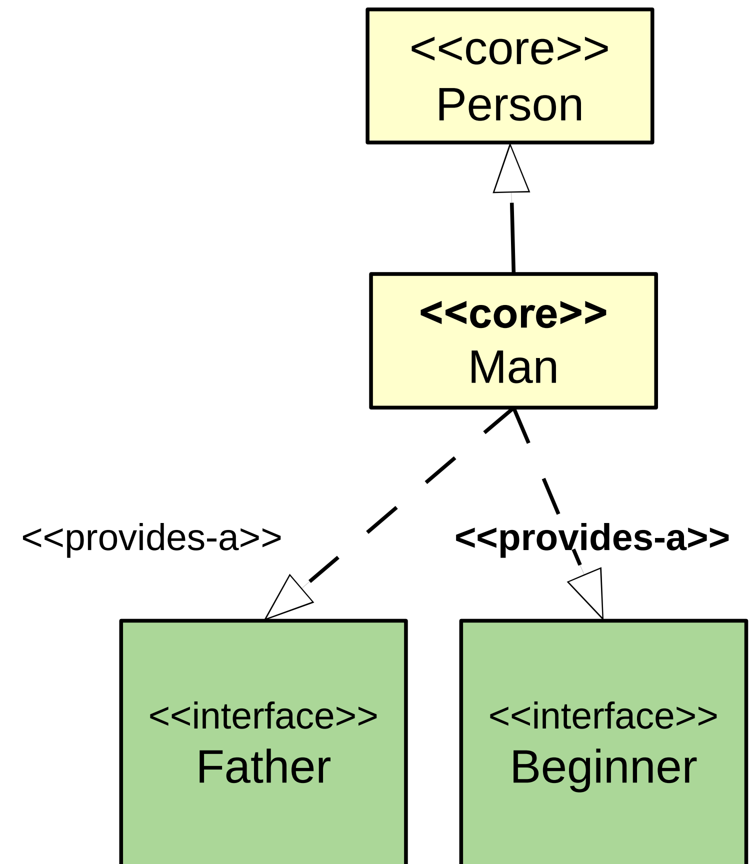


© Prof. U. Aßmann

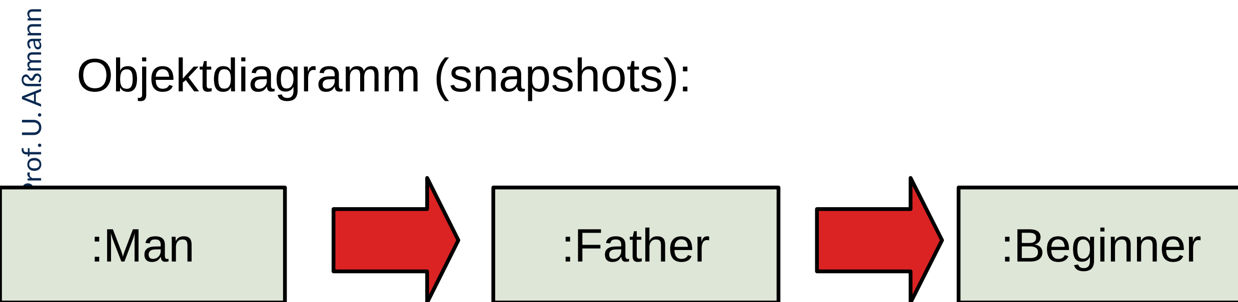
Implementierungsmuster: “Rollen mit Mehrfachvererbung”

- Rollen können auch durch *Mehrfachvererbung* realisiert werden
  - .. ähnlich zur Realisierung mit Einfachvererbung
- Vorteil: Kernobjekt erbt Rollenverhalten
- Nachteil: nur in Scala und C++ möglich





Objektdiagramm (snapshots):



34.A.2 Facettenklassifikation und Facetten-Unterobjekte

(optional, nicht klausurrelevant)

Facettenklassifikation

- Manchmal kann ein Objekt mehrfach klassifiziert werden
  - Person: (Raucher / Nichtraucher), (Gourmet/Gourmand), (Vegetarier/AllesEsser)
- Im Allgemeinen gilt: Eine *Facette* ist eine Klassifikationsdimension eines Objektes
  - Jede Facette hat ein eigenes Klassendiagramm
  - Die Facetten sind unabhängig von einander
  - Die Facetten bilden einen zusammengesetzten Typ, den *powertype*
- Eine Facette ist ein rigider Typ
- Im Speziellen ist eine Facette ein rigides Unterobjekt, das eine Typdimension eines komplexen repräsentiert.
  - Es kann seinen Typ unabhängig von allen anderen Facetten-Unterobjekten wechseln

http://en.wikipedia.org/wiki/Faceted_classification

http://www.webdesignpractices.com/navigation/facets.html

© Prof. U. Aßmann

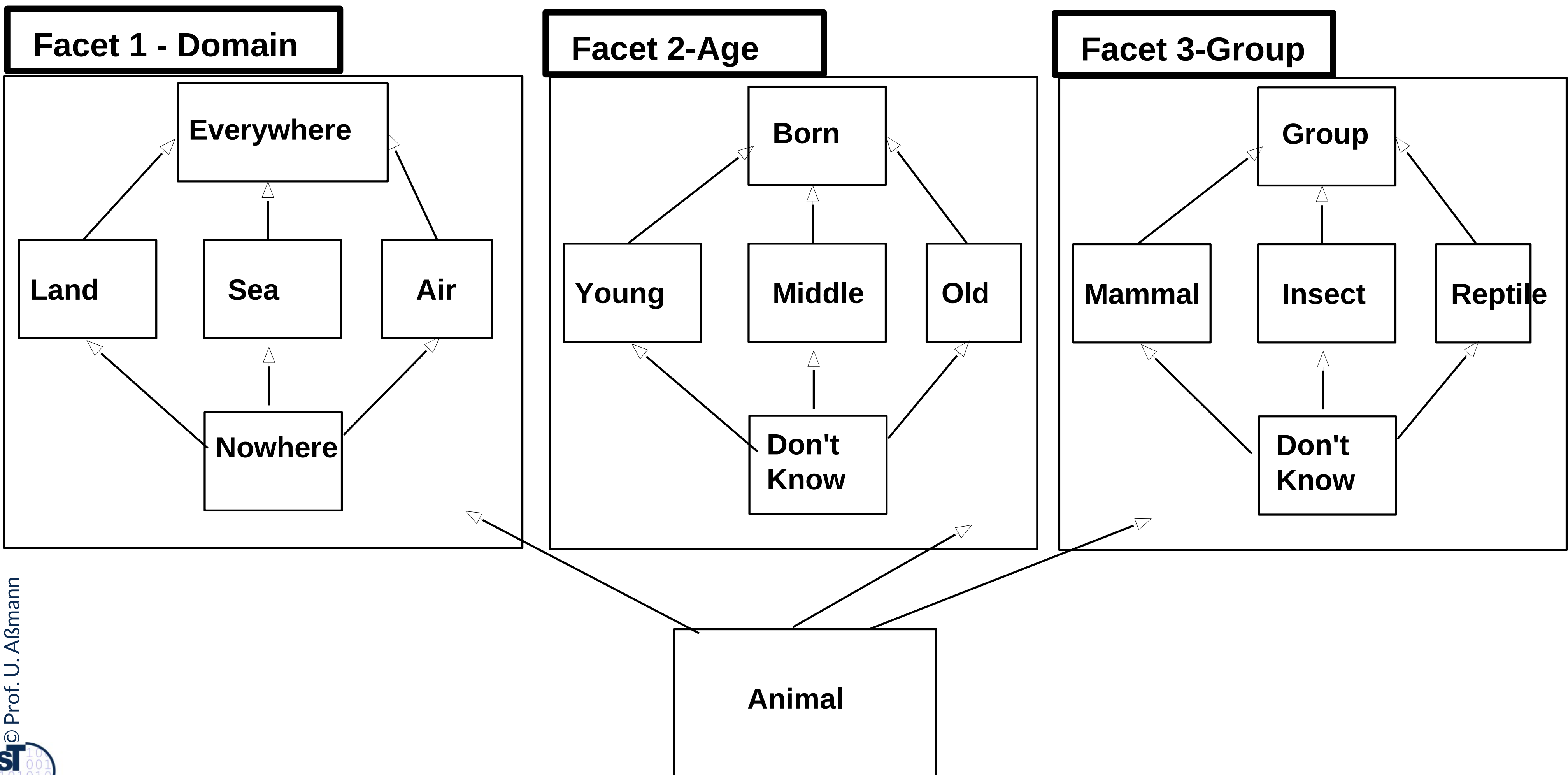
Facetten von Lebewesen

- Das folgende Modell von Lebewesen hat 3 Facetten:
  - Lebensbereich
  - Alter
  - Biologische Gruppe
- Ein Tier hat also 3 Facettenunterobjekte

© Prof. U. Aßmann

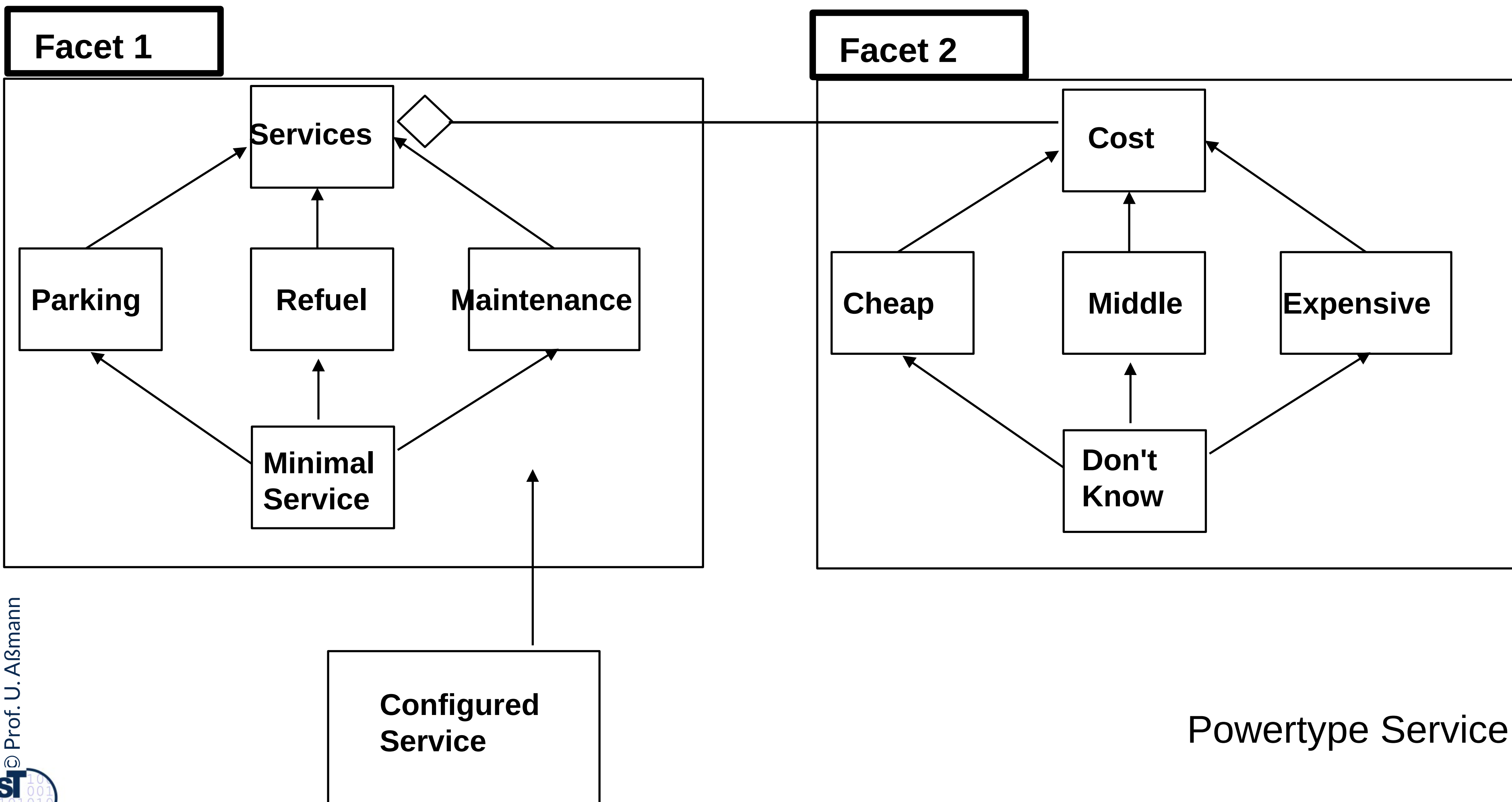
Facetten faktorisieren aus

- Eine Facettenklassifikation ist i.A. einfacher als eine ausmultiplizierte Vererbungshierarchie
  - Bei 3 Facetten braucht ein solches 3^n Klassen



Einfache Realisierung durch Delegation

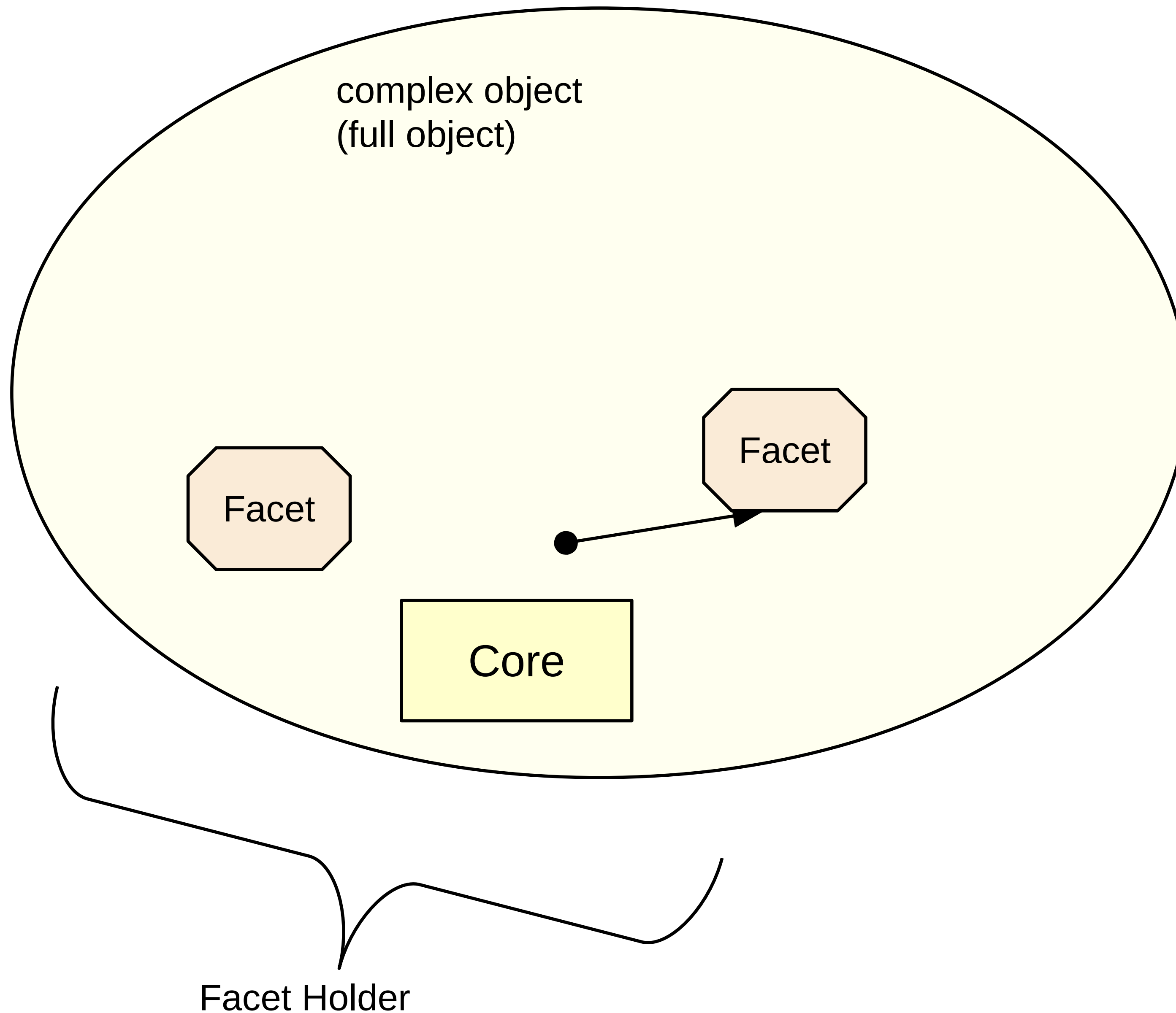
- Eine zentrale Facette, die anderen angekoppelt durch Aggregation (Delegation)



Powertype Service

© Prof. U. Aßmann

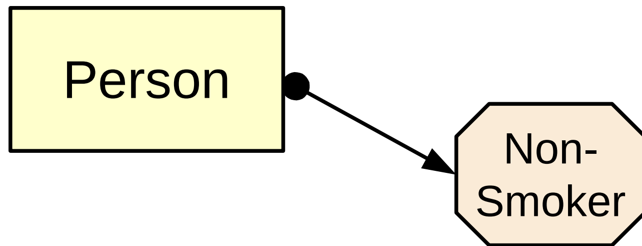
Facetten, repräsentiert als Unterobjekte

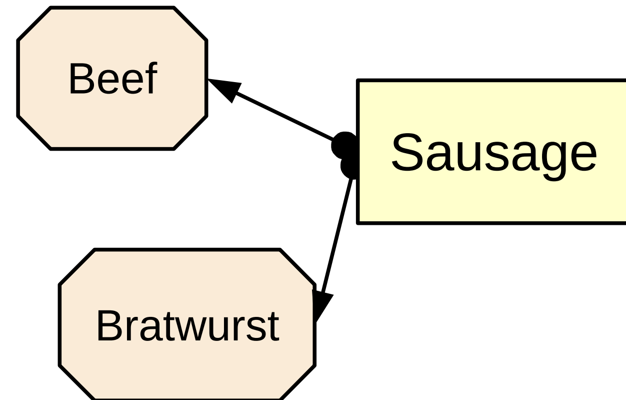


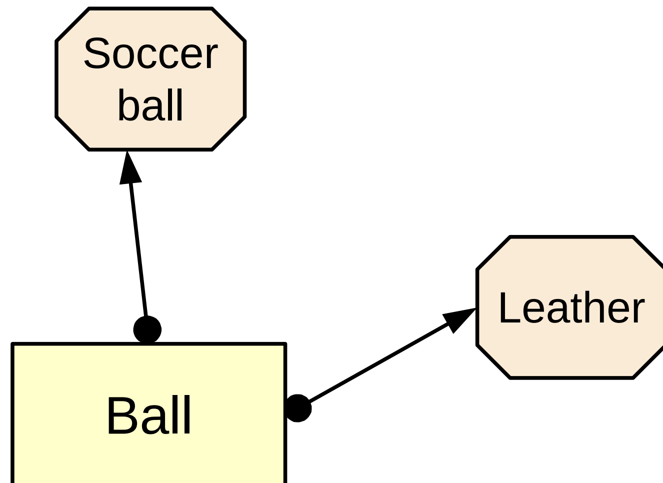
© Prof. U. Aßmann

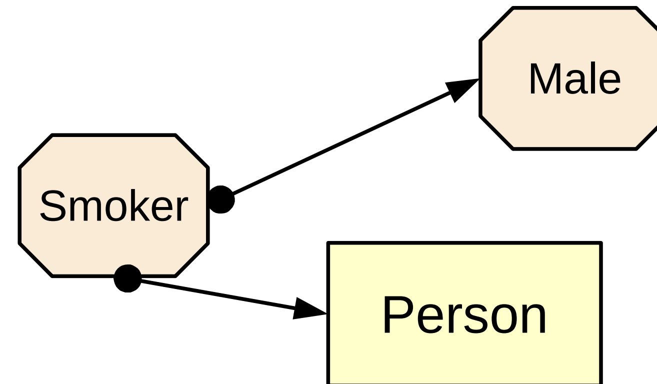
Facetten, repräsentiert als Unterobjekte

- non-founded; rigid









© Prof. U. Aßmann

34.A.2 Phasen als Typen

(optional)

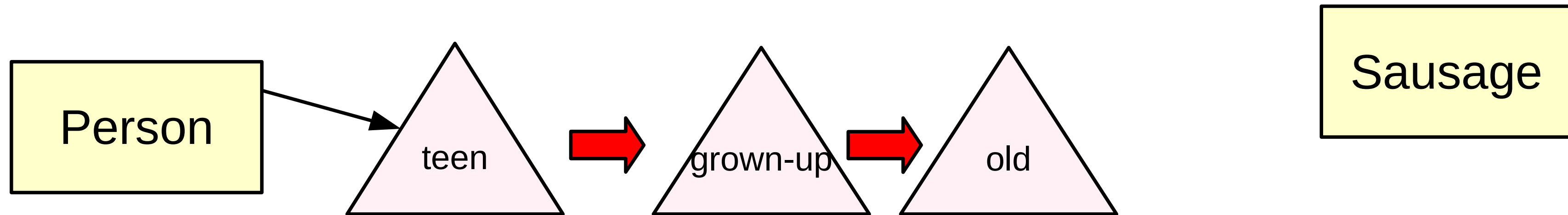
Phasentypen

- Ein **Phasenobjekt** ist ein Unterobjekt, das eine Lebensphase (Zustand) eines komplexen Objektes beschreibt
- Ein **Phasentyp** charakterisiert die Lebensphase eines Objektes in seinem Lebenszyklus
  - Ein Phasentyp ist nicht-rigide, da er sich ändert im Laufe des Lebens

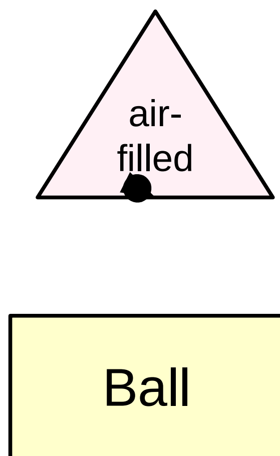
© Prof. U. Aßmann

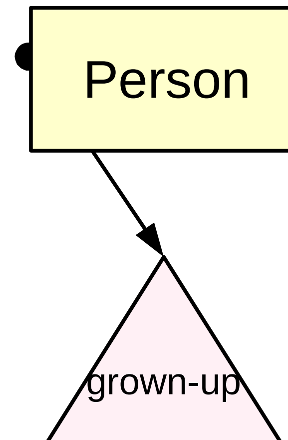
Was sind Phasen?

- Phasen sind nicht-fundiert, nicht-rigide Typen



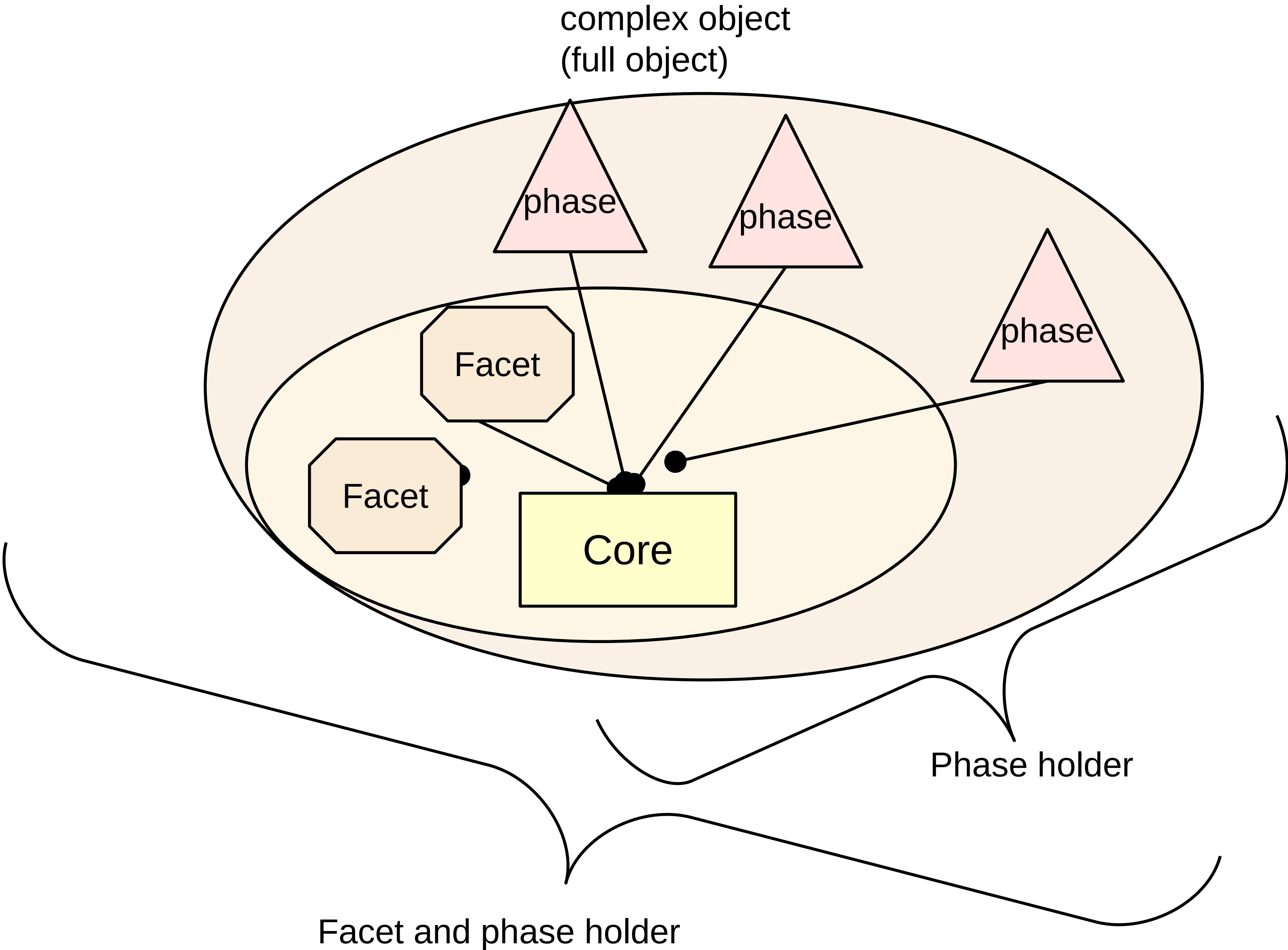
States in a lifecycle





© Prof. U. Aßmann

Komplexes Objekt mit Facetten und Phasen



© Prof. U. Aßmann

The End

© Prof. U. Aßmann