23. Entwurfsmuster für Kommunikation in Teams – Programmierung von fixen Netzen

für die Programmierung der I/O und des Internet

Kommunikation mit Iteratoren, Senken, Kanälen und Konnektoren

Datenhaltung mit potentiell unendlichen Collections

Prof. Dr. Uwe Aßmann

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

Technische Universität Dresden

Version 20-0.3, 6/25/20

1) Teams und Konnektoren (Connectors)
2) Aktoren und Kanäle
3) Entwurfmuster Channel
    1) Entwurfsmuster Iterator (Stream)
    2) Entwurfsmuster Sink
    3) Entwurfsmuster Channel
4) I/O und Persistente Datenhaltung mit Channels
    1) Ereigniskanäle

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Betreff: "Softwaretechnologie für Einsteiger" 2. Auflage

- zur Info: o.g. Titel steht zur Verfügung:
  - 50 Exemplare ausleihbar in der Lehrbuchsammlung
  - 1 Präsenz-Exemplar im DrePunct
  - https://katalogbeta.slub-dresden.de/id/0011358900/#detail
- Jeweils unter ST 230 Z96 S68(2).

Be Careful, The Exam Will be Coming!

https://de.wikipedia.org/wiki/A_Londonderry_Air

https://de.wikipedia.org/wiki/Danny_Boy

Oh, Danny boy, the pipes, the pipes are calling
From glen to glen, and down the mountain side
The summer's gone, and all the roses falling
'Tis you, 'tis you must go and I must bide.

But come ye back when summer's in the meadow
Or when the valley's hushed and white with snow
'Tis I'll be there in sunshine or in shadow
Oh, Danny boy, oh Danny boy, I love you so!

Frederic Weatherly (1910)
Text aus Wikipedia/McCourt, Danny Boy, S. 87 f.

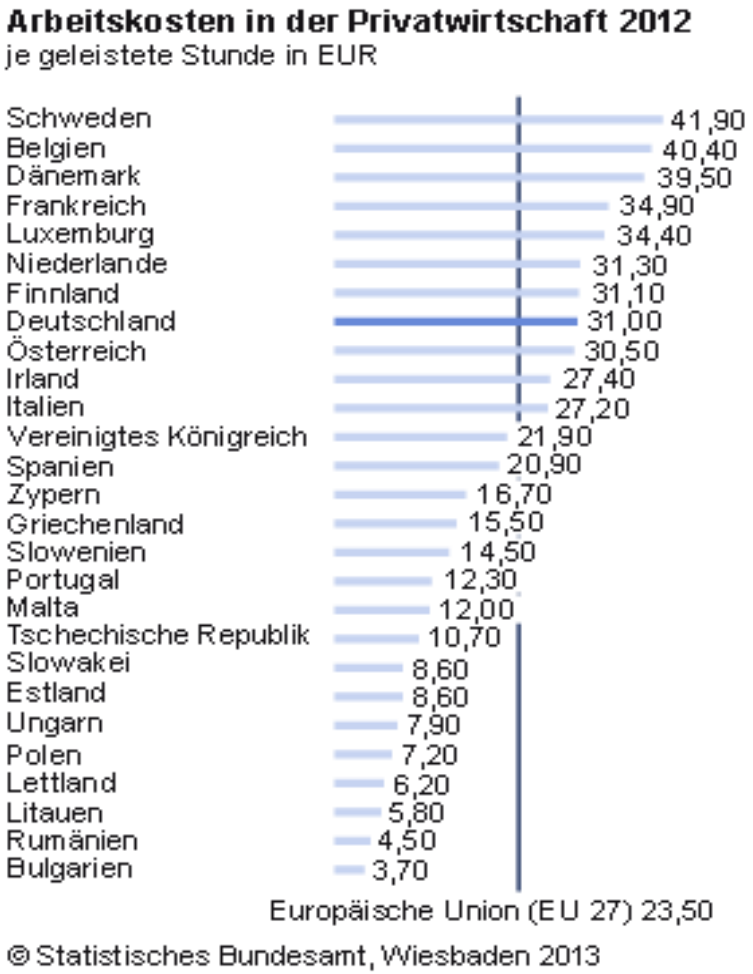
Johnny Cash's classic implementation
https://www.youtube.com/watch?v=egf4X1UzsAA
see also seine CD "American IV"

And when ye come, and all the flow'rs are dying
If I am dead, as dead I well may be
Ye'll come and find the place where I am lying
And kneel and say an Ave there for me.

And I shall hear, though soft you tread above me
And all my grave will warmer, sweeter be
For you will bend and tell me that you love me,
And I shall sleep in peace until you come to me.

Warum müssen Softwareingenieure fortgeschrittenes Wissen besitzen?

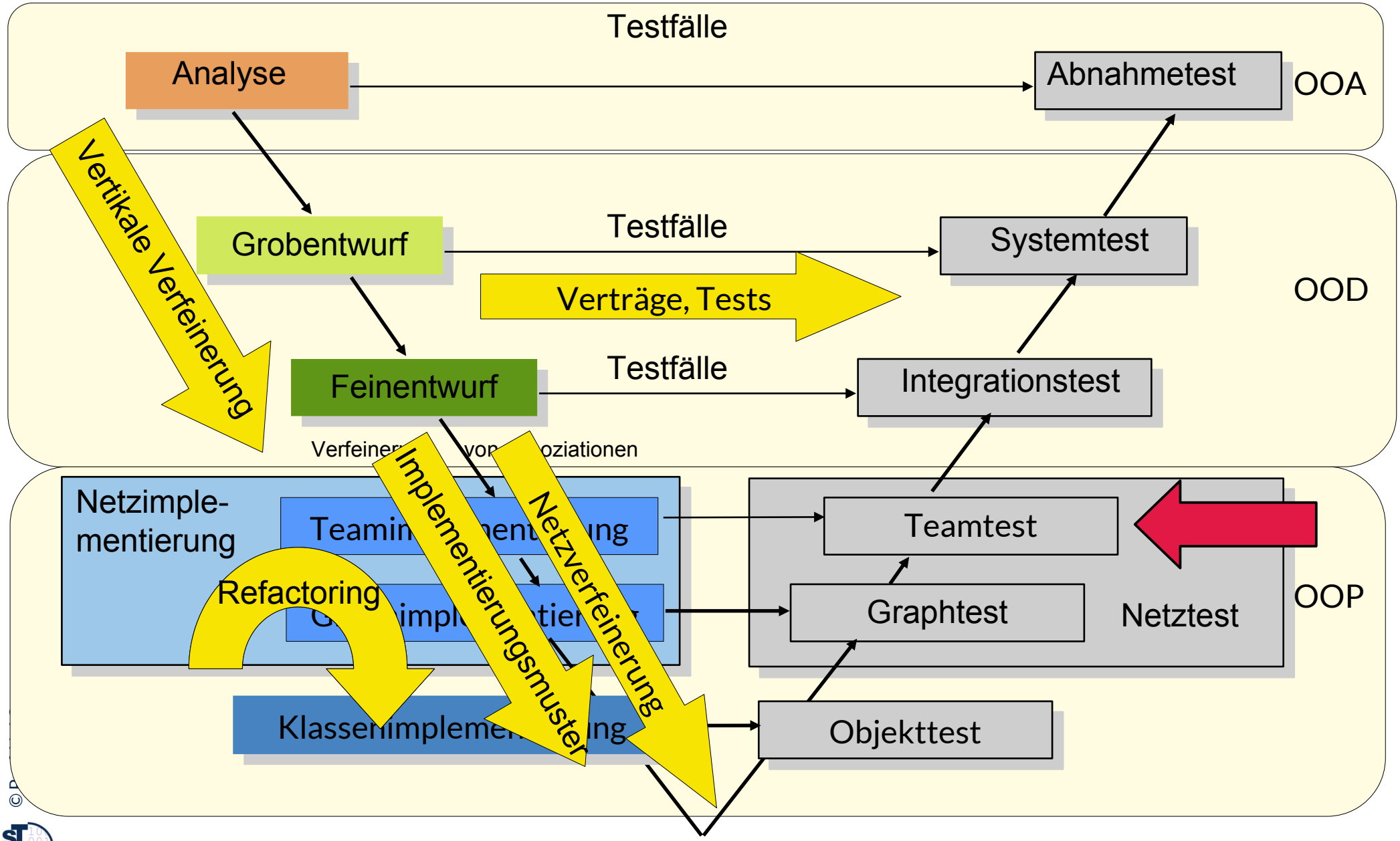
- Die Konkurrenz ist hart: Zu den Kosten der Arbeit:



http://www.heise.de/resale/imgs/17/1/0/0/1/3/4/1/ArbeitskostenEULaenderStart2012-9bb2e8b041f1342e.png

Q4: Softwareentwicklung im V-Modell

[Boehm 1979]

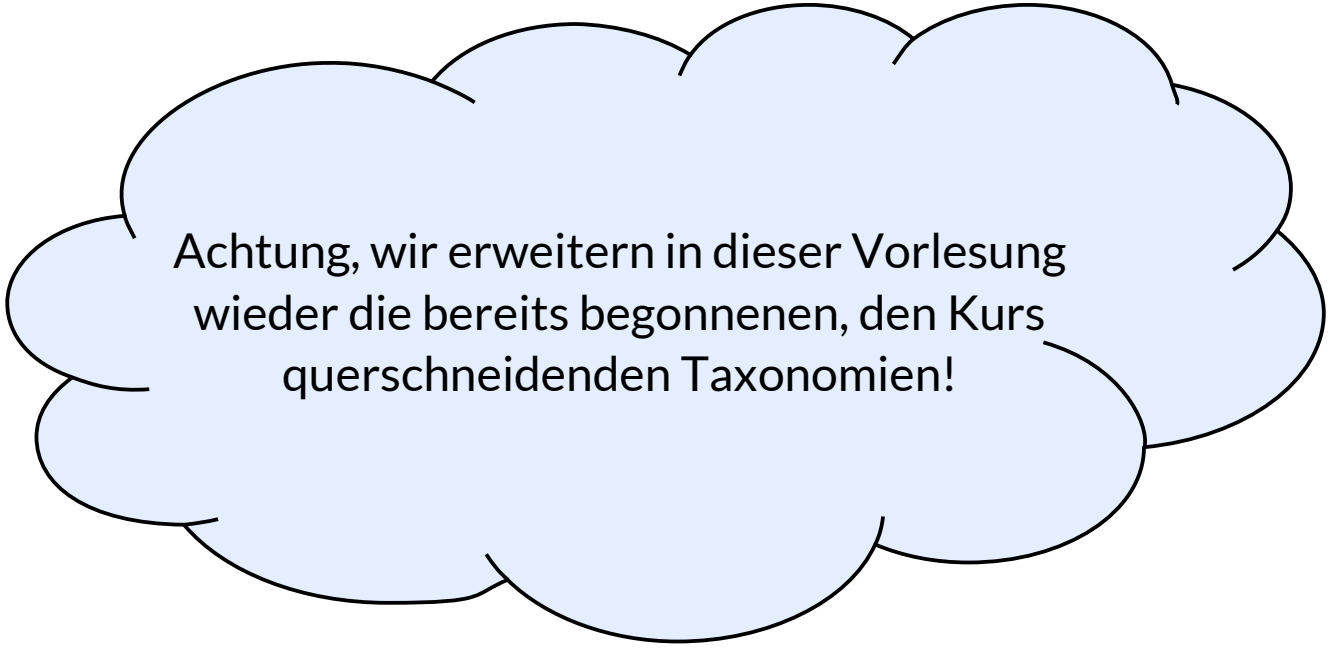


Repräsentation von flexiblen und fixen Objektnetzen als Datenstrukturen (Netzverfeinerung)

E.23.1 Lernen mit Begriffshierarchien, die die Vorlesung querschneiden

Querschneidende Begriffshierarchien

- Wie lernt man mit Ihnen?
  - Klassen-Taxonomie
  - Methoden-Taxonomie
  - Realisierungen von Graphen



23.1 Teams für fixe Netze

- Objekte kommunizieren in Teams (fixen Netzen)
- Teams kommunizieren oft auf kontinuierliche Art, mit wechselnden Partnern, aber in einem fixen Netz
- Auf dem Internet ist das die Regel

Teams (fixe Objektnetze)

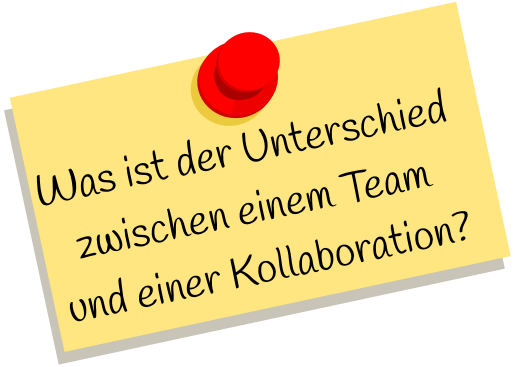
- **Problem**:
  - Das Management von Objekt-Netzen ist eine der schwierigsten und fehleranfälligsten Aufgaben im objektorientierten Programmieren.
  - Objektnetz-Programmierung ist sehr wichtig für Softwarequalität
- Neue Sprachen:
  - Java gehört zur 1. Generation von objektorientierten Programmiersprachen.
  - Die 2. Generation verbessert die Beschreibung von Teams, Netzen von Objekten:
    - Object Teams Www.objectteams.org
    - SCROLL https://github.com/max-leuthaeuser/SCROLL
- Wir lernen im diesem Kapitel, wie man mit Bibliotheken und Entwurfsmustern das Programmieren von Objektnetzen auch in Java verbessern kann!

Teams

Def.: Ein **Team** besteht aus einer festen Anzahl von interagierenden und kommunizierenden Objekten (fixes Netz).

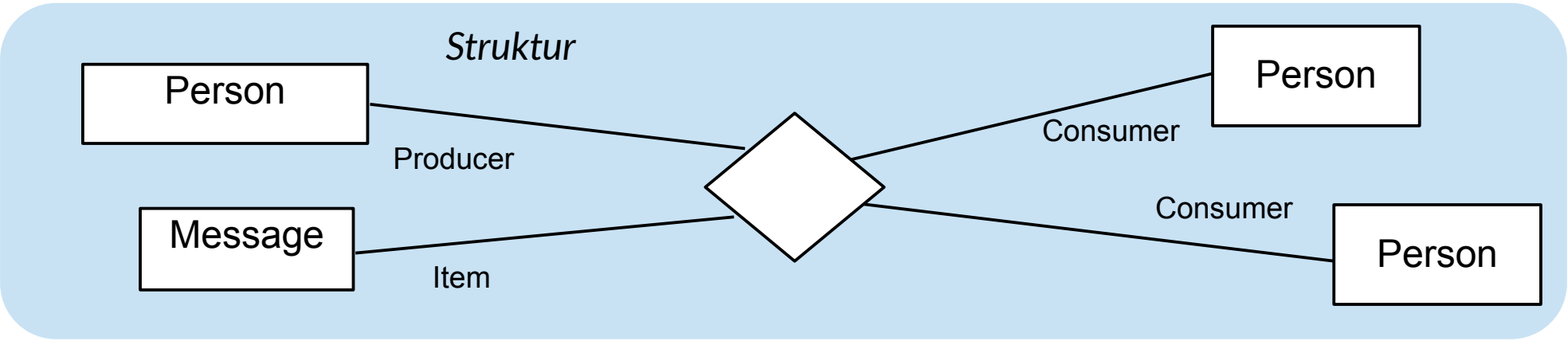
Die Interaktion eines Teams wird in UML durch eine **Kollaboration** beschrieben.

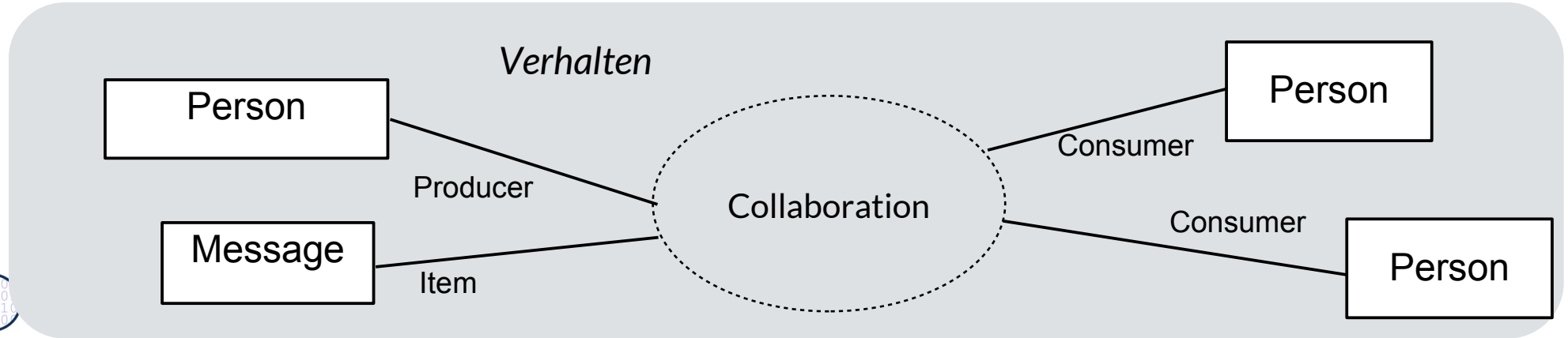
- Historik: Der Begriff des Teams wurde von der TU Berlin im Projekt www.objectteams.org geprägt.



Kollaborationen kapseln das Verhalten von fixen Netzen (Teams)

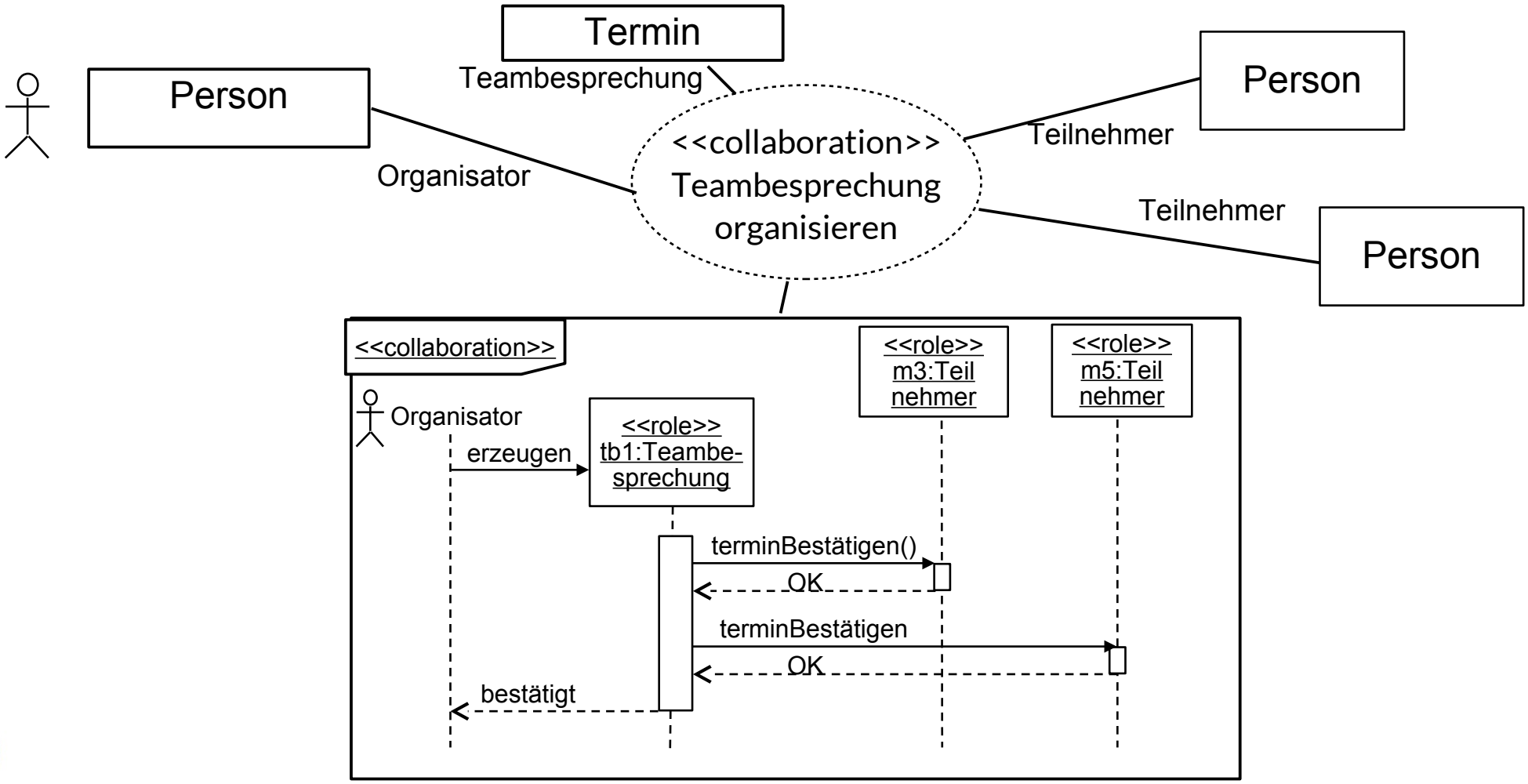
- Die *Struktur* von Teams (fixes Netz mit n Mitgliedern) wird in UML durch n-stellige Assoziationen dargestellt
- Das *Verhalten* eines fixen Netzes und seine Kommunikation durch Kollaborationen
- Def.: Eine **Kollaboration (collaboration, Kollaborationsklasse)** realisiert die Kommunikation eines Teams mit einem festen anwendungsspezifischen Protokoll





Kollaborationen kapseln Verhalten eines Teams durch Interaktionsprotokolle

- Eine **Kollaboration** beschreibt die anwendungsspezifische Interaktion, Nebenläufigkeit und Kommunikation eines Teams, d.h. einer fixen Menge von beteiligten Objekten.
- Die Kollaboration beschreibt also ein Szenario querschneidend durch die Lebenszyklen mehrerer Objekte



Teams und technische Konnektoren

Def.: Ein **Teamobjekt** ist ein Objekt,
das die Kommunikation einer Gruppe (feste Anzahl) von
interagierenden und kommunizierenden Objekten kapselt.
Ist ein Teamobjekt hauptsächlich mit dem technischen Austausch von Daten
zwischen den Objekten beschäftigt, heißt es **Konnektor.**

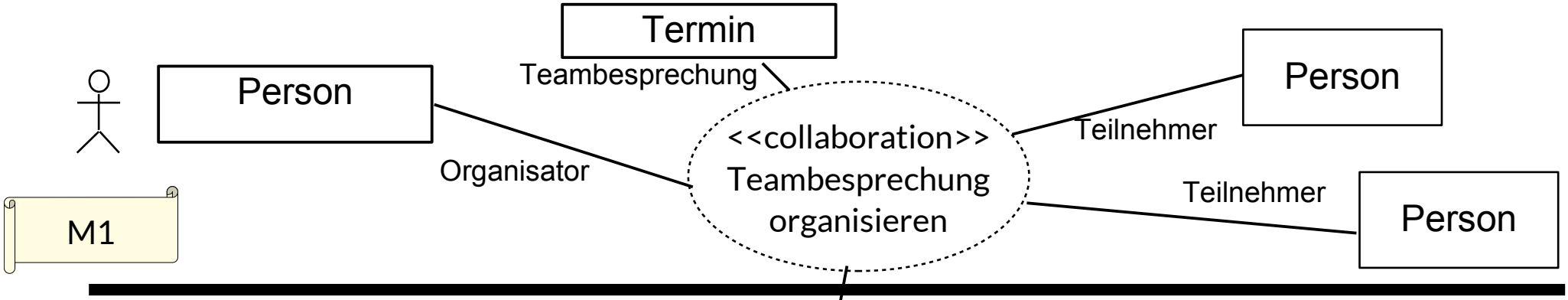
Bsp: Teams: Dynamo Dresden Team, Staatskapelle
Konnektoren: Aldi--Peter Müller:Käufer, Finanzamt--Jenny Klein:Steuerzahler

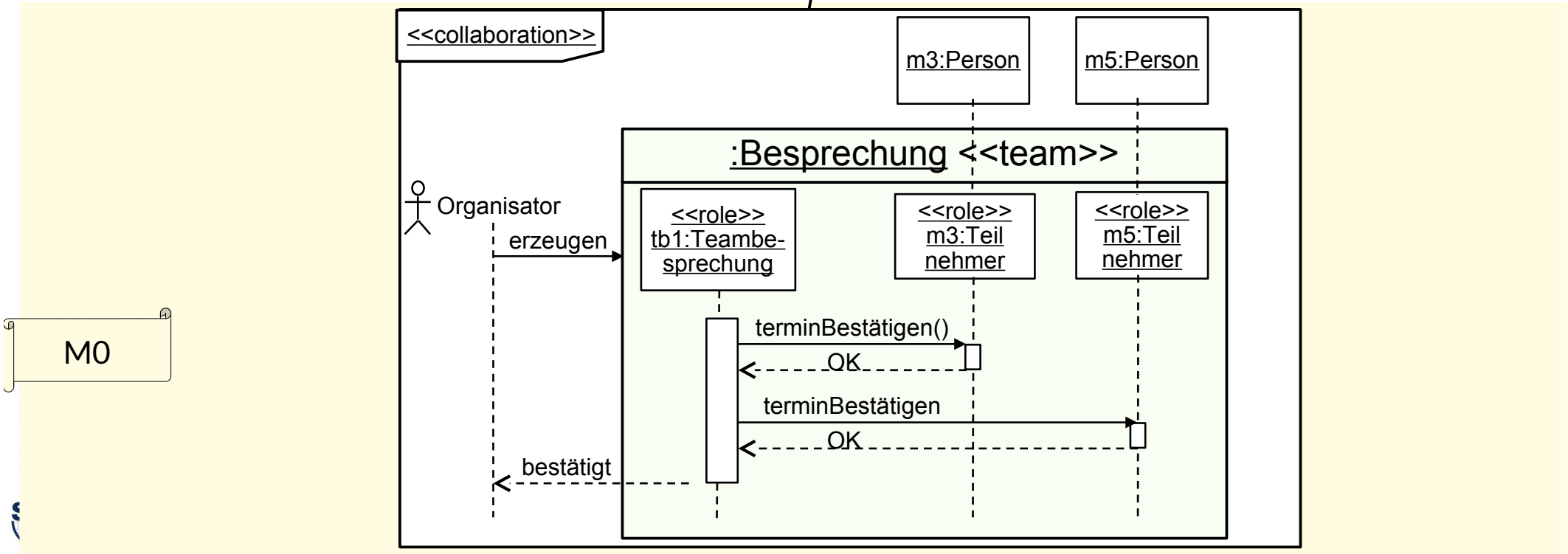
Def.: Kann eine Kollaboration durch eine Klasse gekapselt werden,
spricht man von einer **Teamklasse.**
Spezialfall beim Datenaustausch: **Konnektorklasse,** kurz **Konnektor.**

Bsp: Teamklassen: Fußballmannschaft, Kapelle
Konnektorklassen: Produzent-Konsument, Client-Server

Teamklassen kapseln Verhalten durch Interaktionsprotokolle

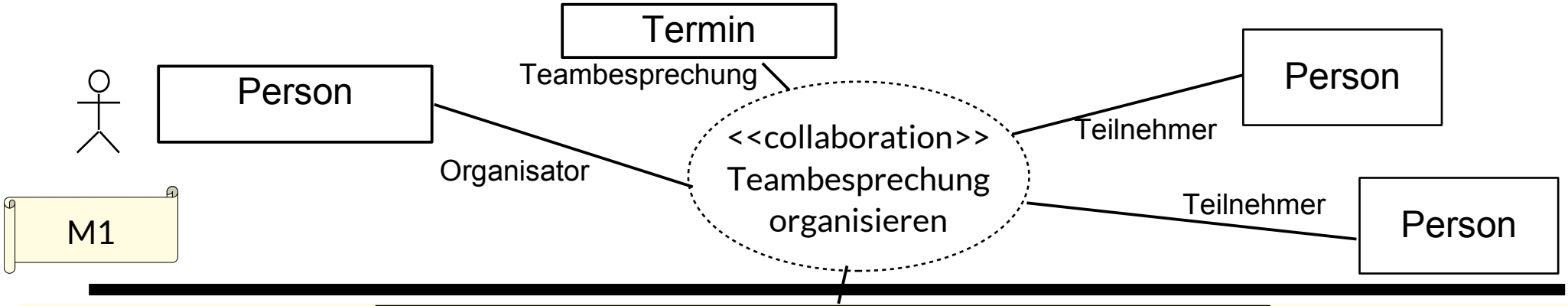
- Def.: Ein **Team (Konnektor)** realisiert eine Kollaboration durch eine Klasse (ein Hauptobjekt)

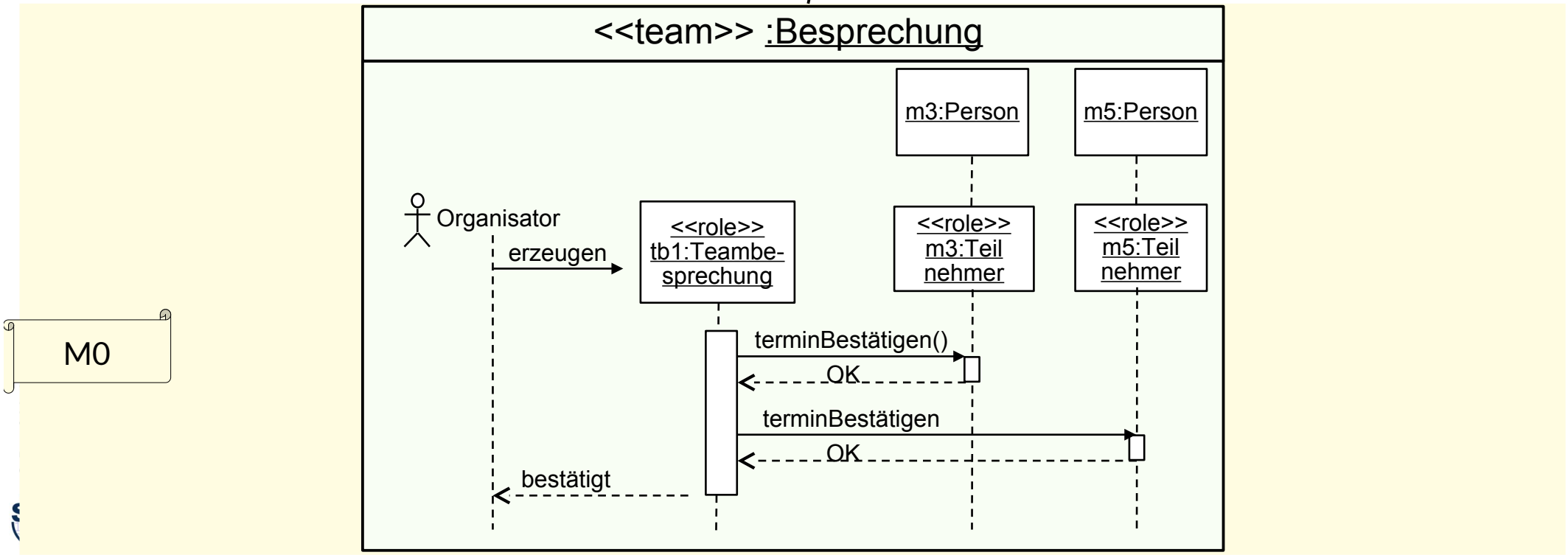




Kurznotation

- Wir schreiben die Kollaboration in den Konnektor





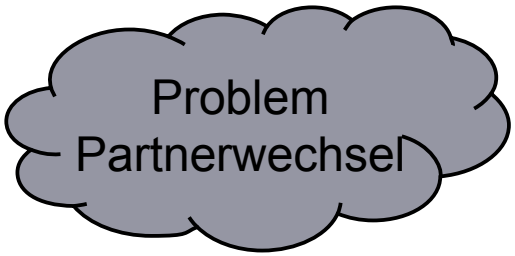
Teams und Entwurfsmuster

Beobachtung:
Viele Entwurfsmuster beschreiben
Schemata für **Teams** und ihre **Kollaborationen**.

23.2 Aktoren und Känale (Actors and Channels) für lose gekoppelte Teams

Wie man Konnektoren für die Programmierung des Internets nutzt

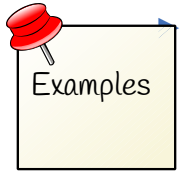
- Kanäle sind einfache Konnektoren (Teamklassen)
- Web-Objekte kommunizieren oft in Teams auf kontinuierliche Art, mit wechselnden Partnern, aber in einem fixen Netz



DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

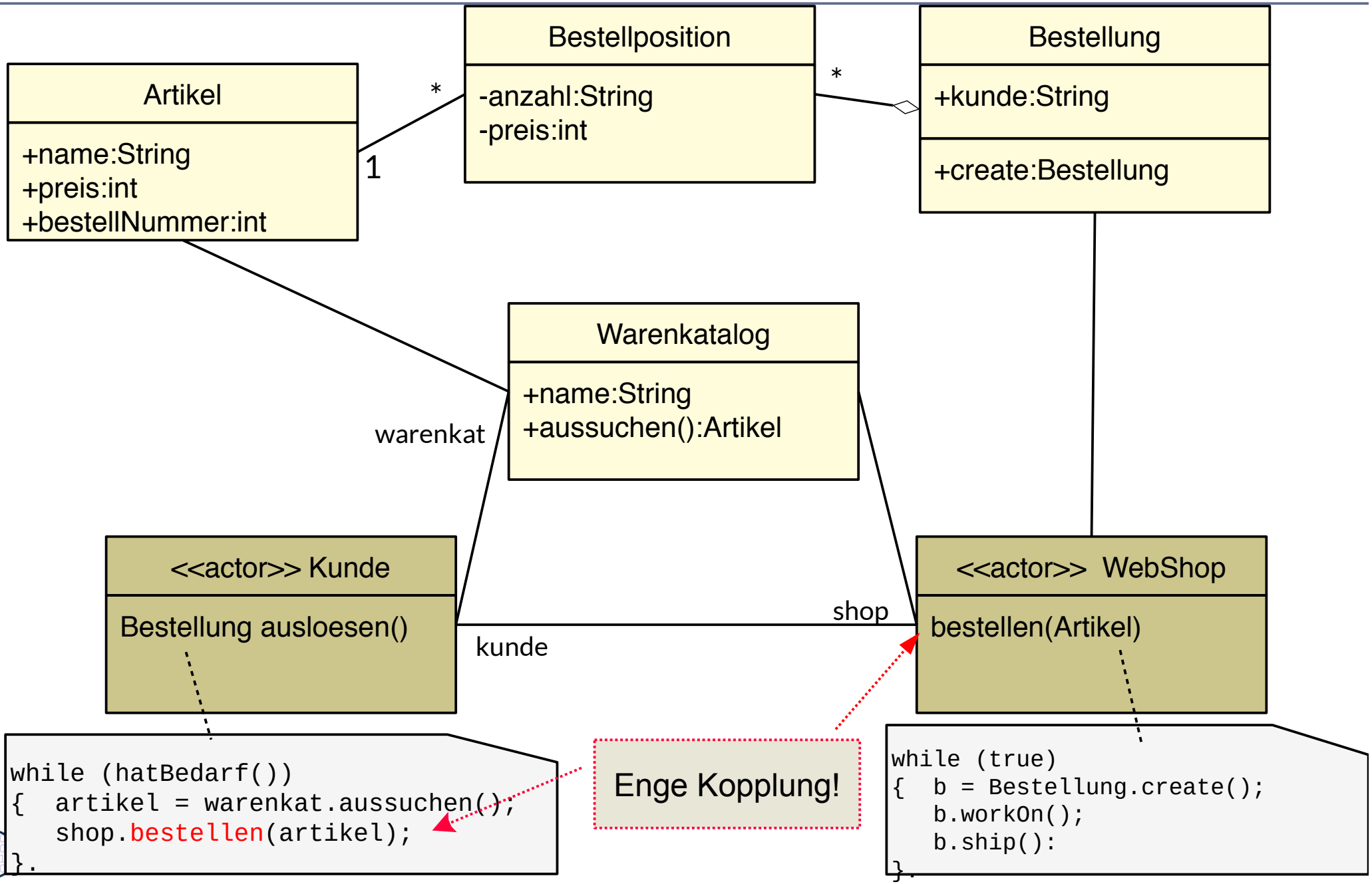
Teams und Entwurfsmuster

Def.: Ein **Aktor (actor)** ist ein aktives Objekt mit einem eigenen Lebensfaden (thread).
Ein Aktor besitzt Kanäle, über die er Botschaften empfängt und sendet

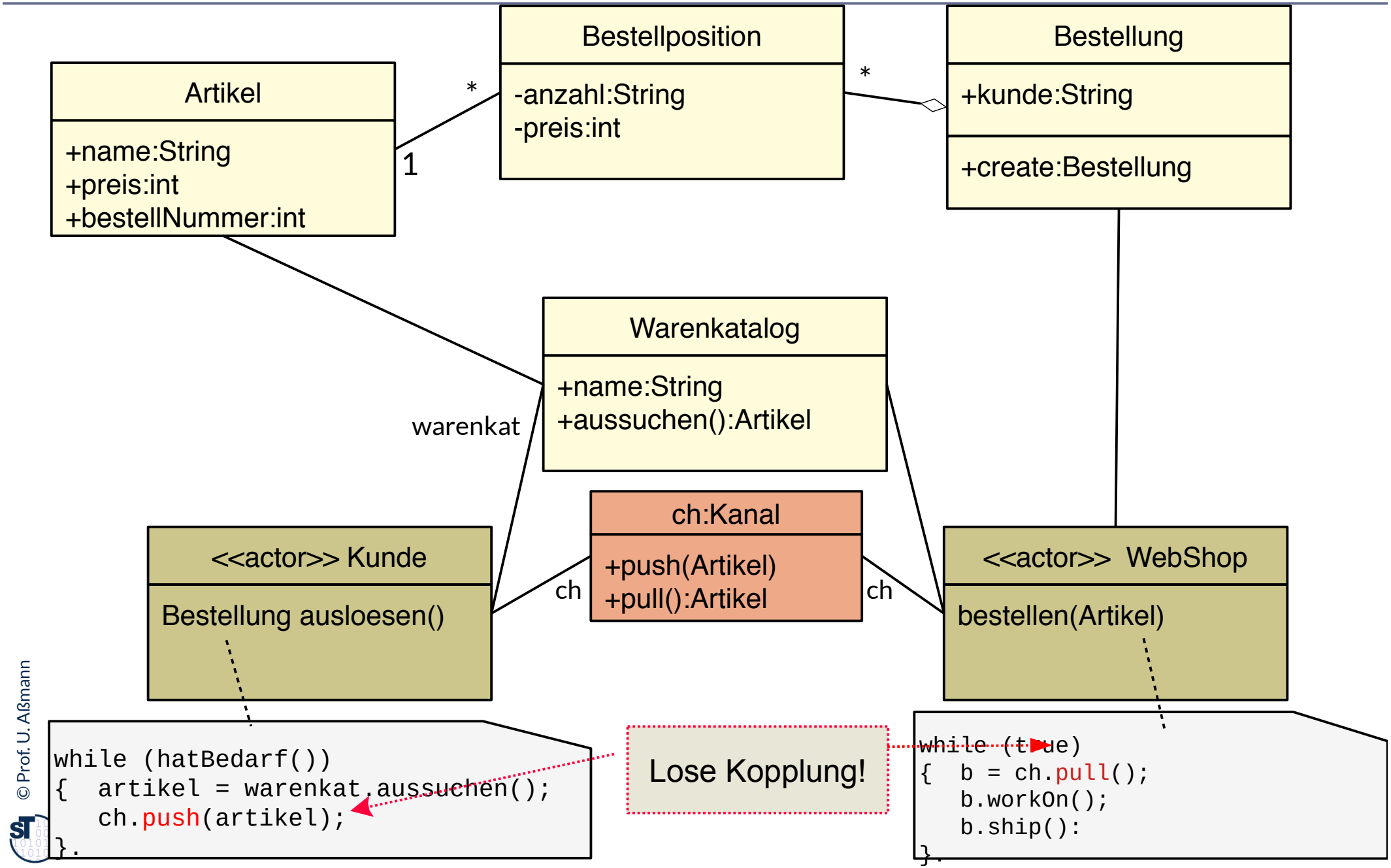


- BeEin Webserver ist ein Aktor und besitzt *ports* als Kanäle (http-Port 80)
  - Liste der standardisierten Ports
    https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports
- Unter LINUX heißt der Aktor ***Prozess*** und besitzt die Kanäle
  - stdin (Standardeingabe)
  - stdout (Standardausgabe)
  - sterr (Standardfehlerausgabe)

Beispiel: Bestellung auf einem Webshop



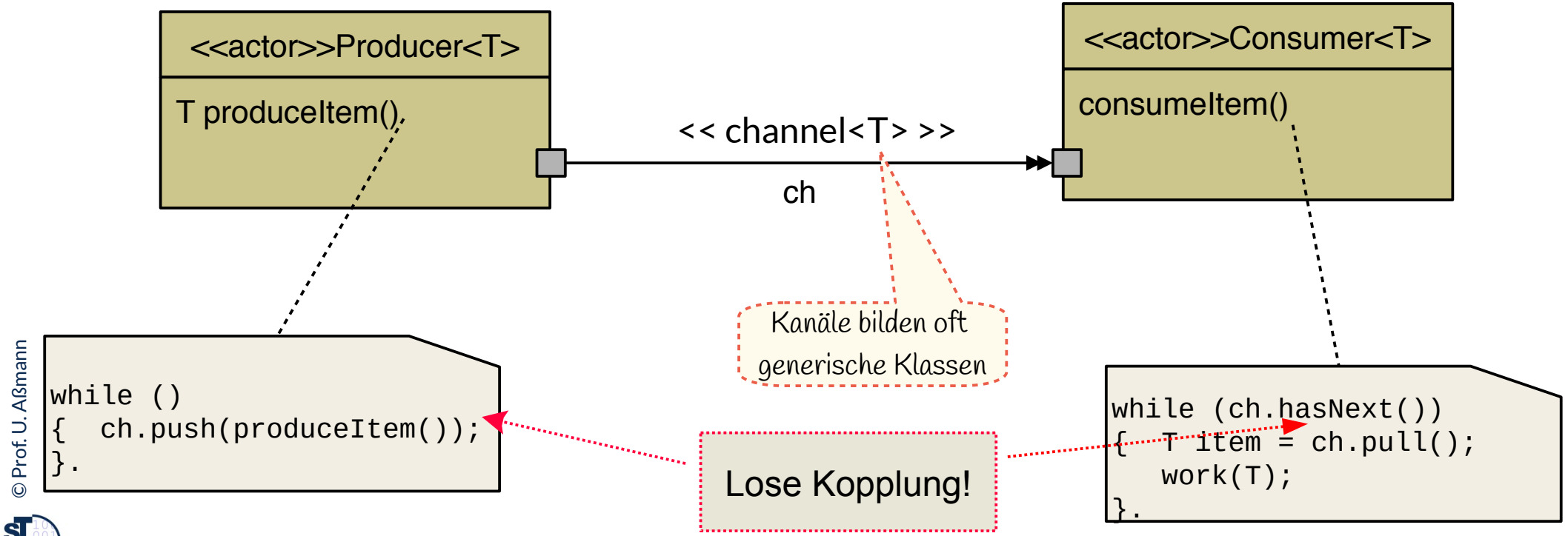
Beispiel: Bestellung auf einem Webshop; nun lose gekoppelt



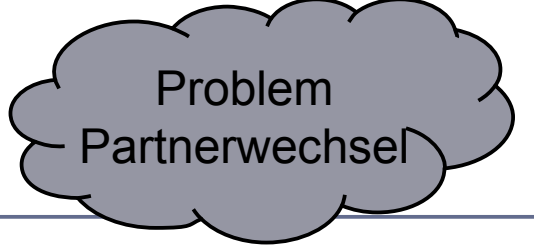
Kanäle bilden Netze mit Datenfluss

Def.: Ein **Kanal (channel, pipe, stream)** ist ein Konnektor, der zur Kommunikation von Anwendungsklassen mit *Datenfluss* dient. In den Kanal werden Daten gegeben und Daten entnommen.

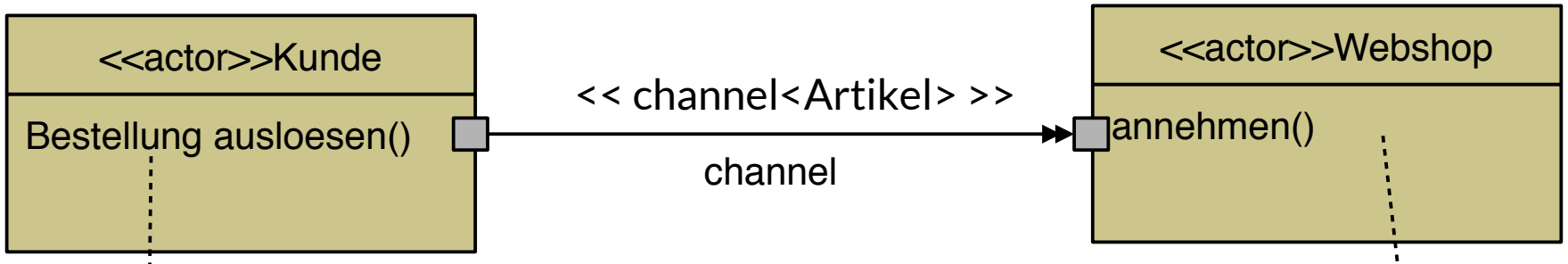
- UML Notation: Andocken eines Kanals an *sockets* (*pins*, *ports*)
- https://en.wikipedia.org/wiki/Douglas_McIlroy
- https://en.wikipedia.org/wiki/Pipeline_(Unix)



Anwendung von Kanälen: lose-gekoppelte Kommuniktation, auch im Internet



- Webshops dürfen die konkreten Objekte ihrer Kunden nicht kennen
- Kanäle erlauben, die Partner zu wechseln, ohne Kenntnis des Netzes
  - Ideal für fixe Netze mit dynamisch wechselnden Partnern
  - Ideal für Webprogrammierung




```
while (hatBedarf())
{  artikel = warenkat.aussuchen();
   channel.bestellen(artikel);
}.
```

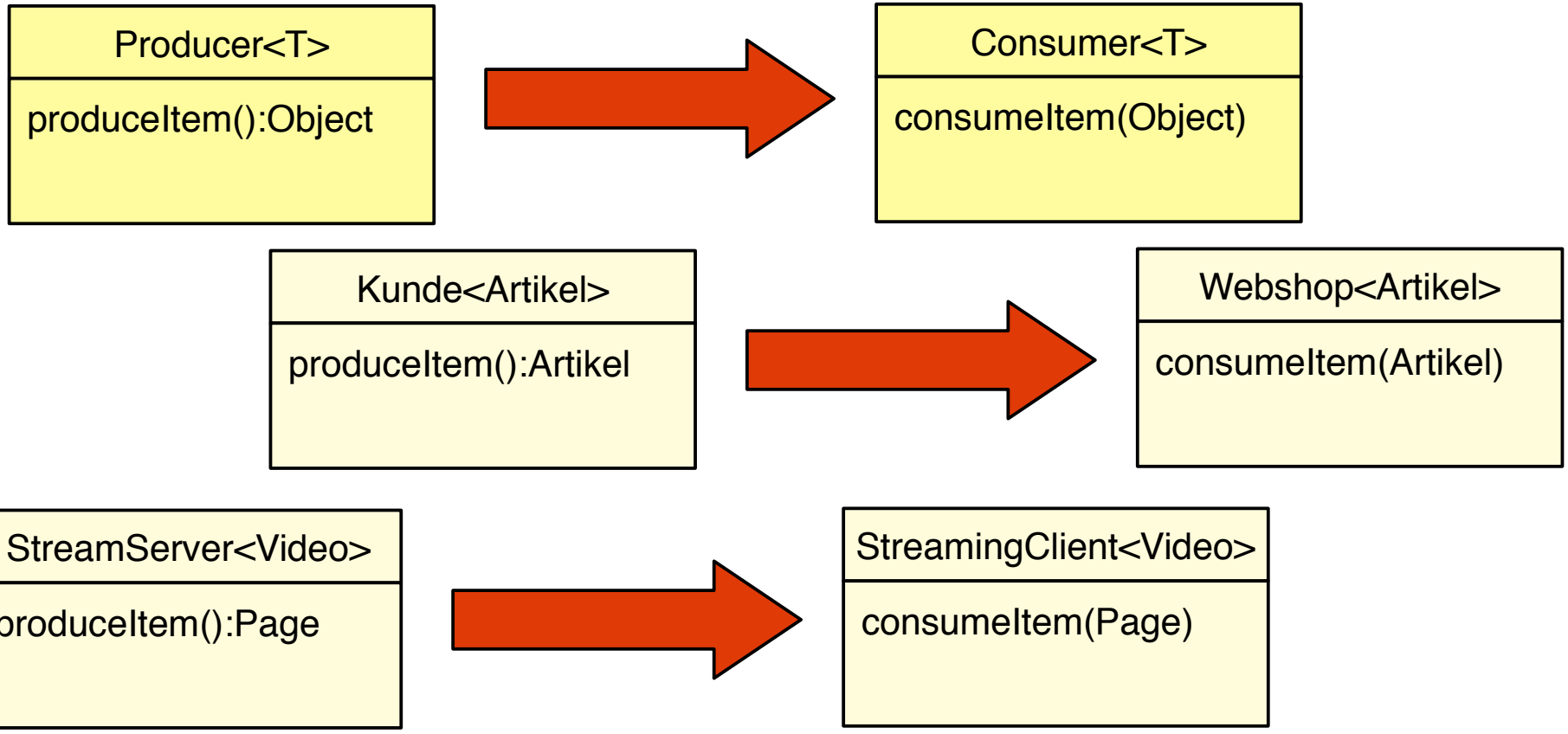
```
while (true)
{  b = Bestellung.create();
   artikel = channel.pull();
   b.workOn(artikel);
   b.ship():
}.
```

23.3 Entwurfsmuster Channel

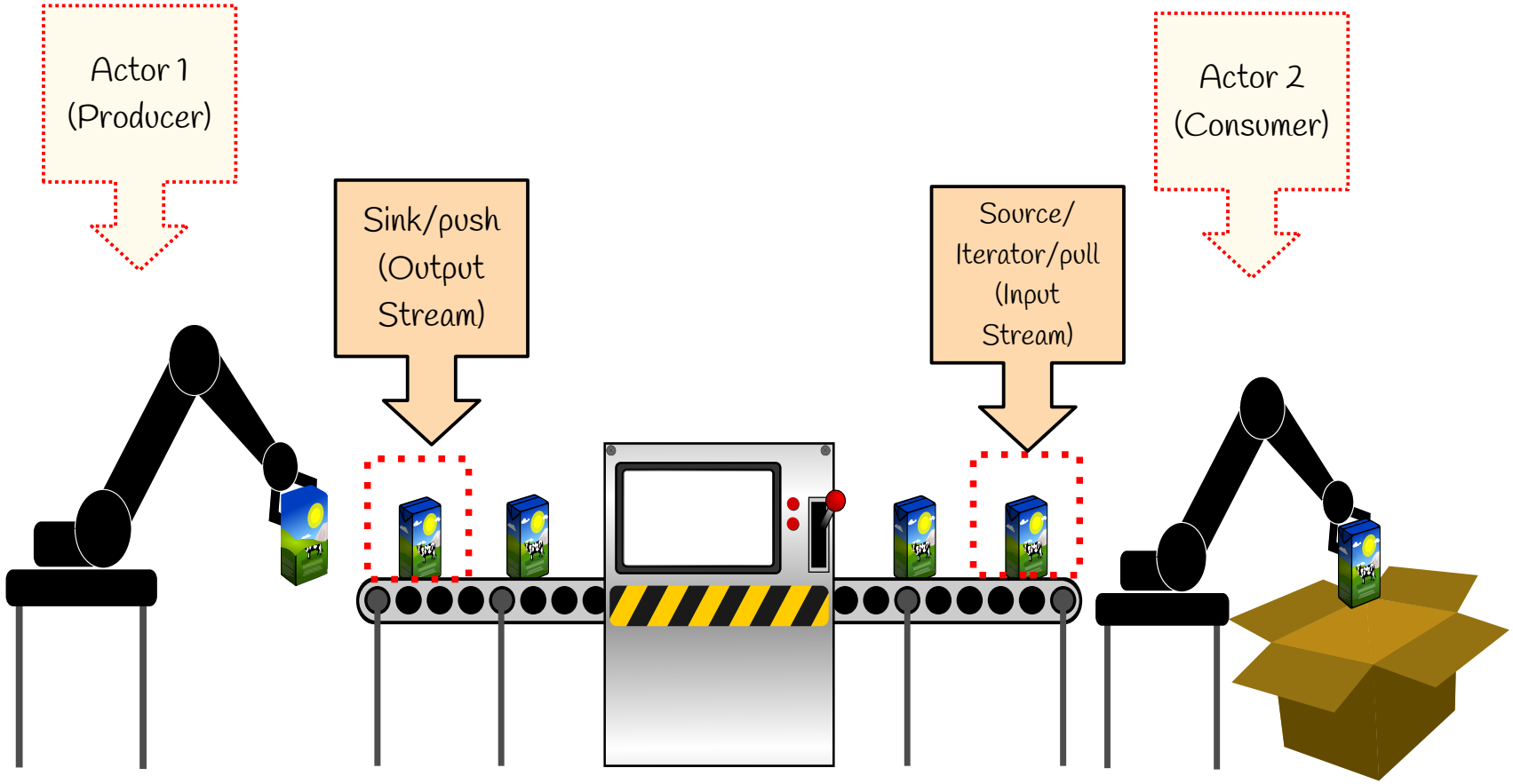
- Eines der wesentlichen Entwurfsmuster für Internet- und Webprogrammierung.
- Channel sind technische Objekte, zum olympischen Ring "Infrastruktur" gehörend
- Darstellung von potentiell unendlichen Collections

Wie organisiere ich die “unendlich lange” Kommunikation zweier Aktoren?

- **Problem**: Über der Zeit laufen in einem Webshop eine Menge von Bestellungen auf
  - Sie sind aber nicht in endlicher Form in Collections zu repräsentieren
- **Frage**: Wie repräsentiert man potentiell unendliche, unbestimmte Collections?
- **Antwort**: mit Kanälen.

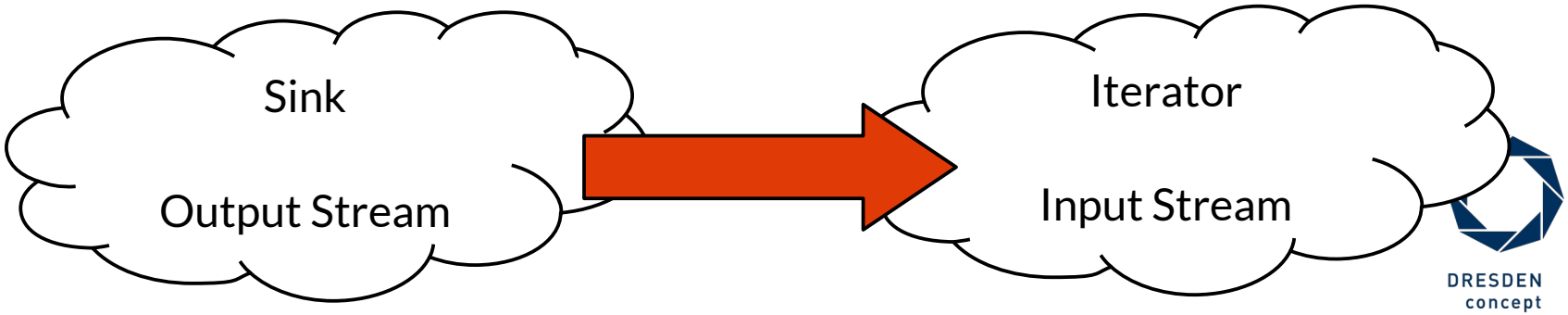


Actors and Channels with Sinks and Iterators



23.3.1 Entwurfsmuster Iterator (Eingabestrom, input stream)

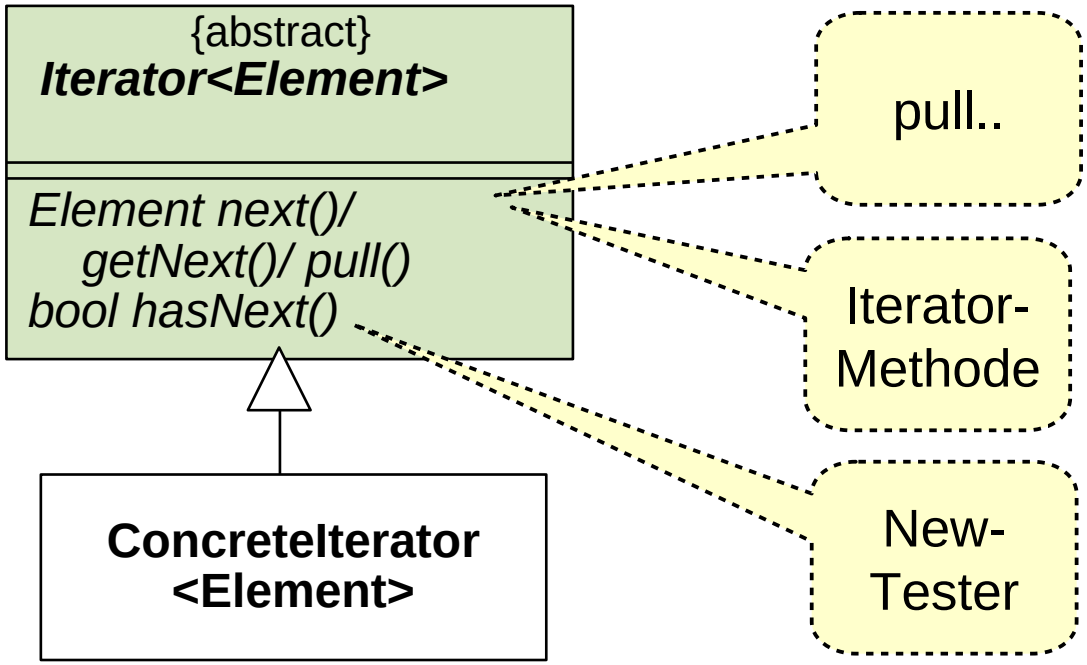
Kanäle bestehen aus mit einander verbundenen Enden, mindestens zweien



DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

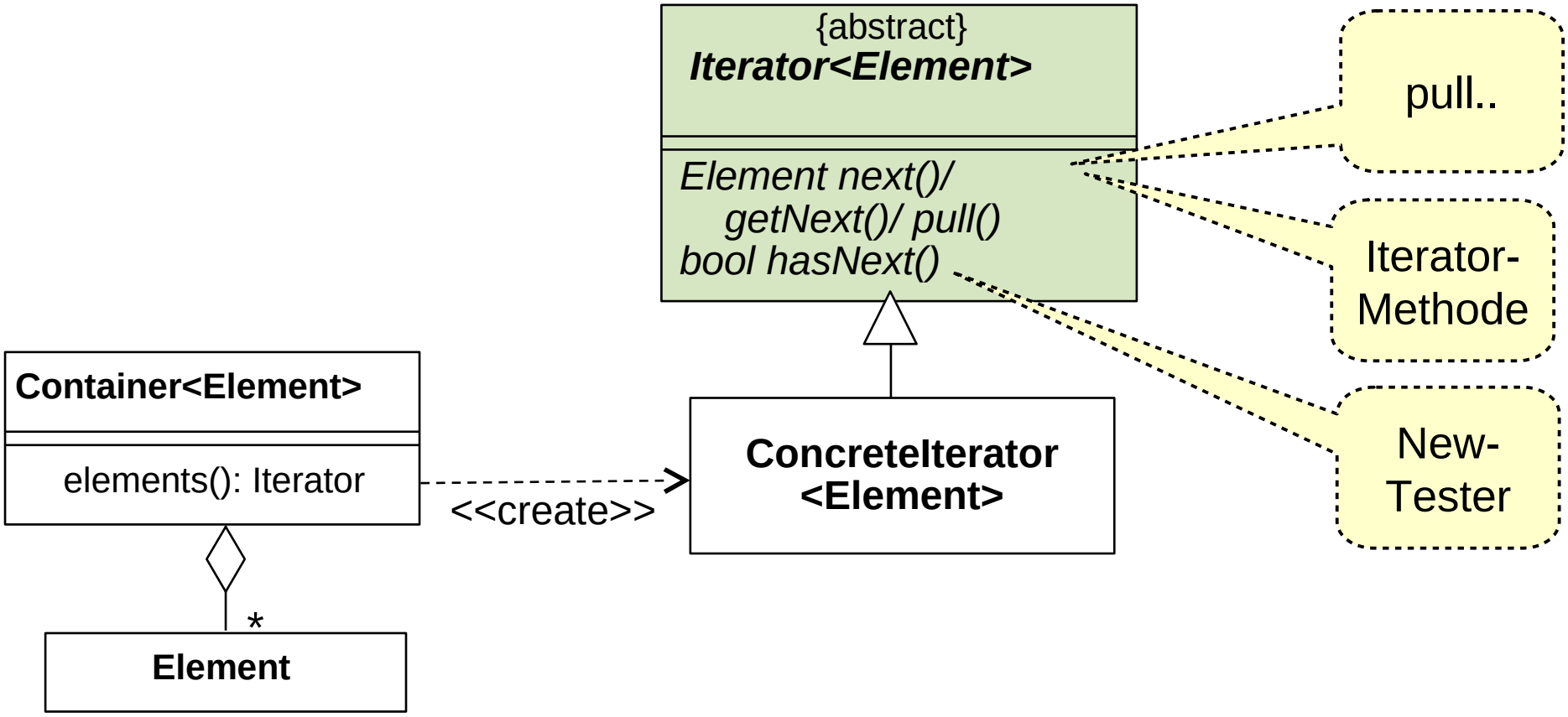
Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

- Ein **Eingabestrom (input stream, Iterator, iterator)** ist eine potentiell unendliche Folge von Objekten (zeitliche Anordnung einer pot. unendlichen Folge bzw. collection)
- Eingabe in eine Klasse oder Komponente

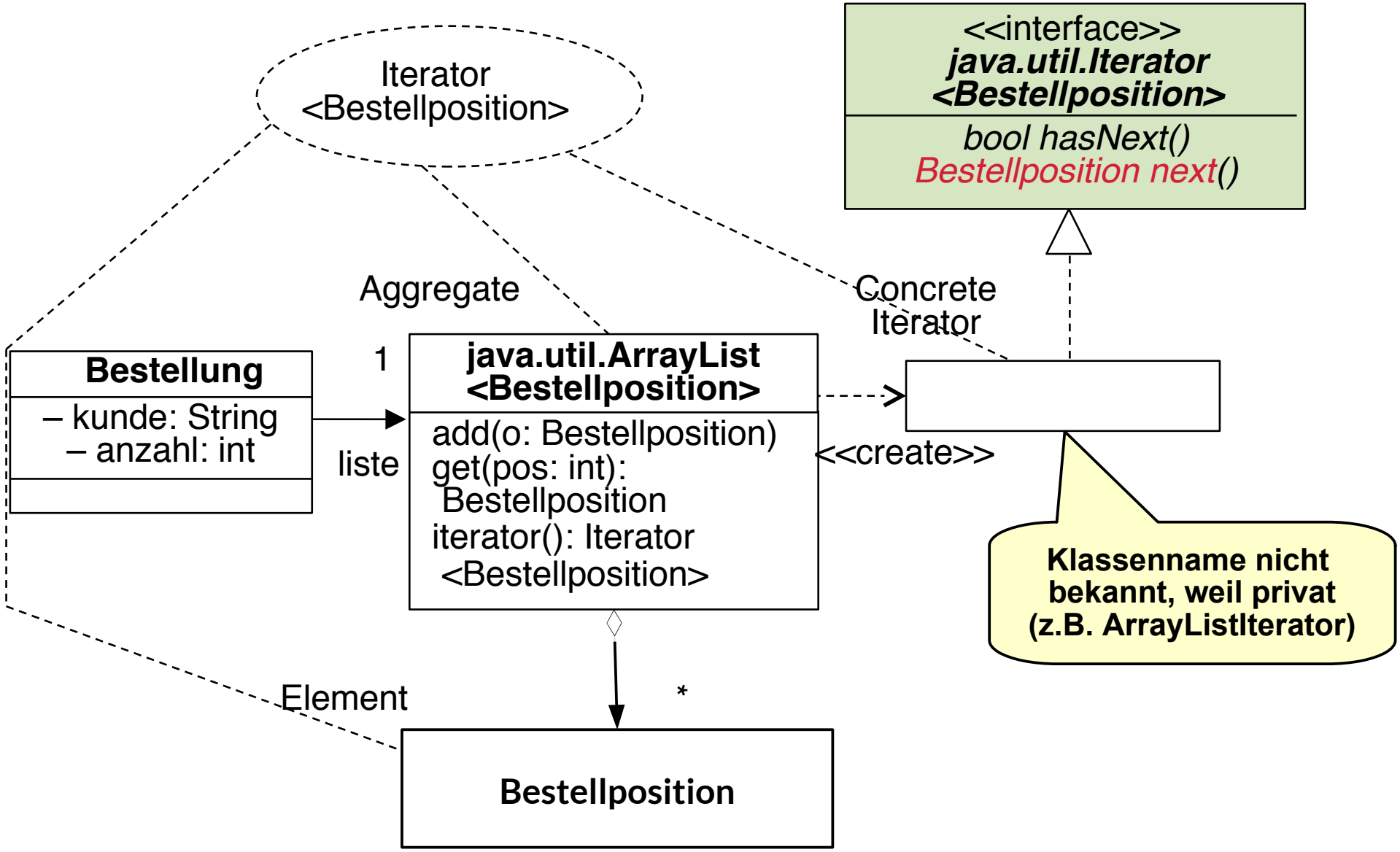


Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

- Containerklassen haben Iteratoren



Iterator-Beispiel in der JDK (ArrayList)



Implementierungsmuster Iterator

- Verwendungsbeispiel:

```
T thing;
List<T> list;
..
Iterator<T> i = list.iterator();
while (i.hasNext()) {
      doSomeThing(i.next());
}
```

- Einsatzzwecke:
  - Verbergen der inneren Struktur einer Datenstruktur
  - **bedarfsgesteuerte Berechnungen** auf der Struktur
  - “unendliche” Datenstrukturen

Anwendungsbeispiel mit Iteratoren

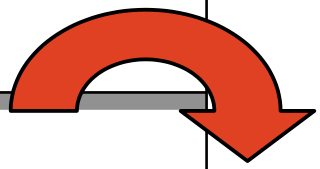

```
import java.util.Iterator;
...
class Bestellung {

  private String kunde;
  private List<Bestellposition> liste;
  ...
  public int auftragssumme() {
    Iterator<Bestellposition> i = liste.iterator();
    Bestellposition b;

    int s = 0;
    while (i.hasNext()) {// Test whether next element
                         // (on the source) is available
      b = i.next();      // pull the next element
      s += b.positionspreis();
    return s;
  }
  ...
}
```

Online:
Bestellung2.java

For-Schleifen auf Iterable-Prädikatschnittstellen

- Erbt eine Klasse von `Iterable`, kann sie in einer *vereinfachten for-Schleife* benutzt werden
- Typisches Implementierungsmuster

```
class BillItem extends Iterable {
  int price; }
class Bill {
  int sum = 0;
  private List billItems;
  public void sumUp() {
  for (BillItem item: billItems) {
    sum += item.price;
  }
  return sum;
  }
}
```

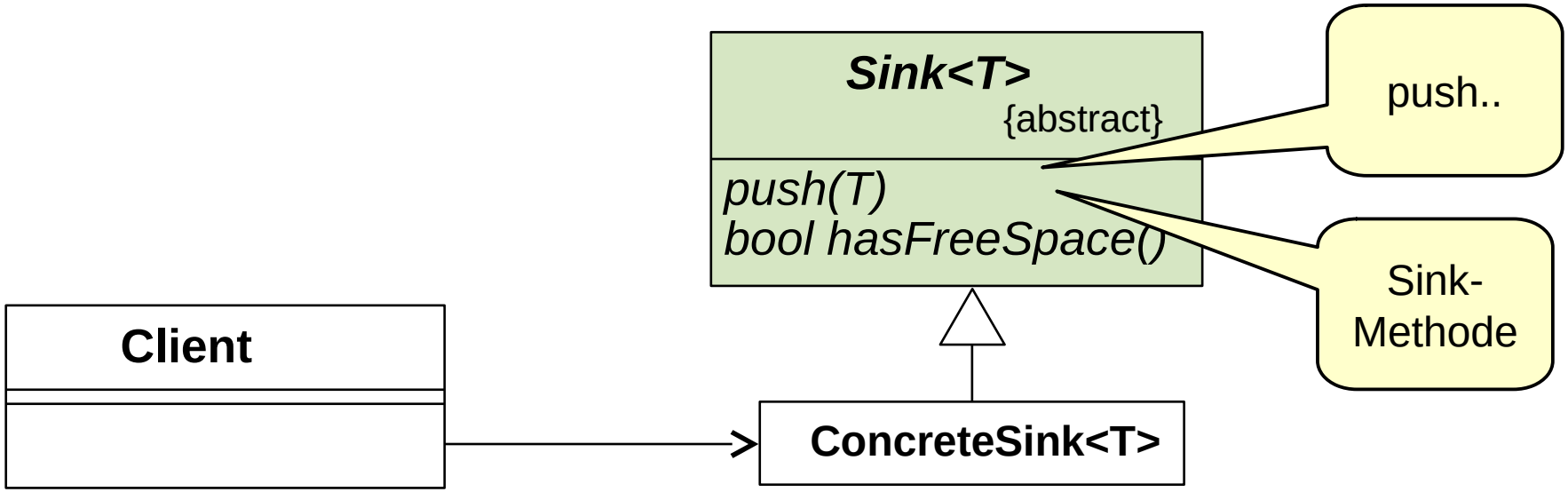


```
class BillItem { int price; }
class Bill {
  int sum = 0;
  private List billItems;
  public void sumUp() {
  for (Iterator i = billItems.iterator();
       i.hasNext(); ) {
    item = i.next();
    sum += item.price;
  }
  return sum;
  }
}
```

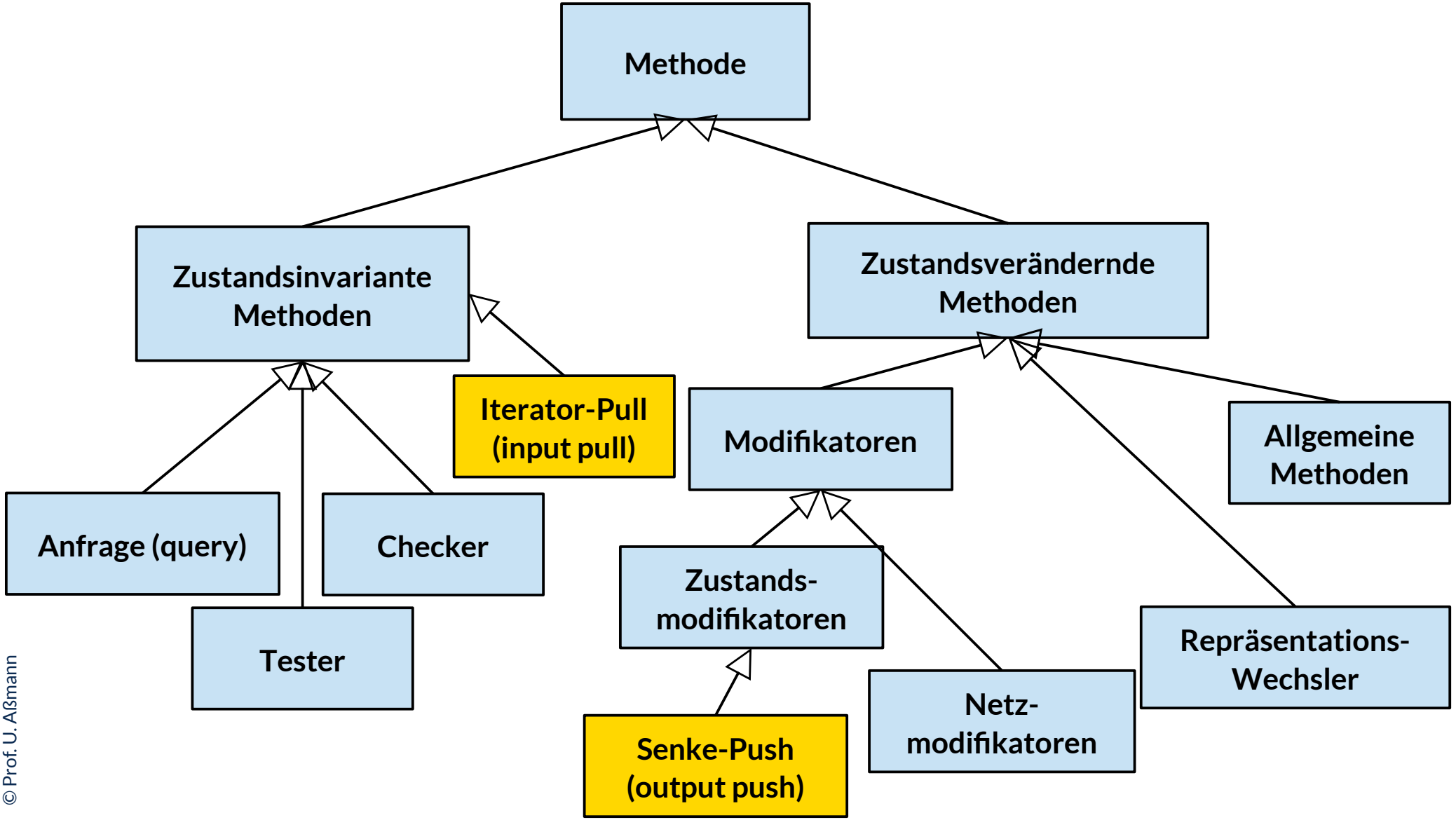
23.3.2 Senken (Sinks, OutputStream)

Entwurfsmuster Senke (Implementierungsmuster)

- Name: **Senke** (auch: Ablage, sink, **output stream**, belt, output-socket)
- Problem: Ablage eines beliebig großen Datenstromes.
  - push
  - ggf. mit Abfrage, ob noch freier Platz in der Ablage vorhanden
- Senken (sockets) organisieren den Datenverkehr zum Internet



Q3: Erweiterung: Begriffshierarchie der Methodenarten

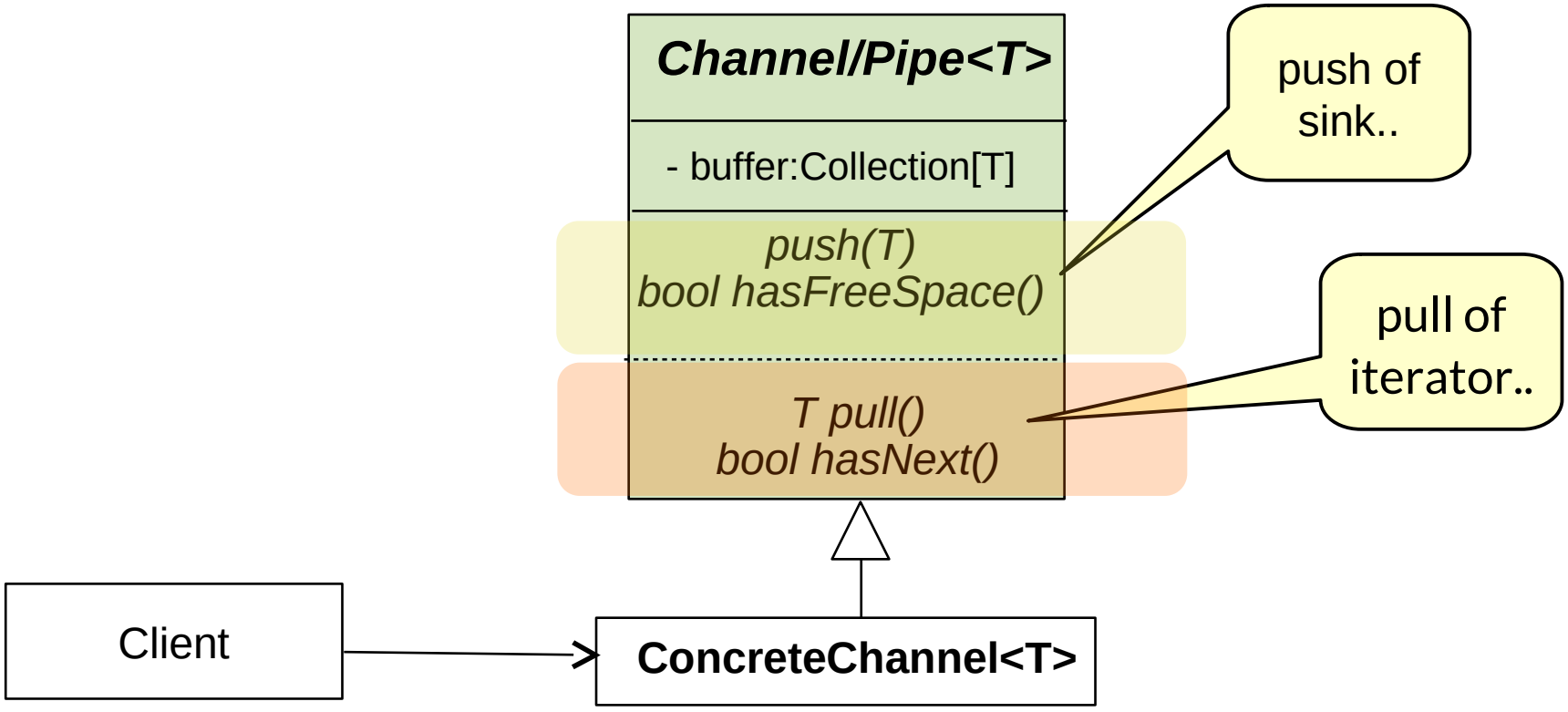


23.4 Channels (Pipes)

- Die Kombination aus Senken und Iteratoren, ggf. mit beliebig großem Datenspeicher
- Eine Pipe ist ein einfacher Konnektor

Entwurfsmuster Channel und Pipe (Implementierungsmuster)

- **Name**: Ein **Channel** (Kanal, full stream) ist ein Konnektor, der die gerichtete Kommunikation (Datenfluss) zwischen Produzenten und Konsumenten organisiert. Er kombiniert einen *Iterator* mit einer *Senke*.
- **Zweck**: asynchrone Kommunikation mit Hilfe eines internen Puffers buffer
- Wir sprechen von einer **Pipe (Puffer, buffer)**, wenn die Kapazität des Kanals endlich ist, d.h. `hasFreeSpace()` irgendwann false liefert.

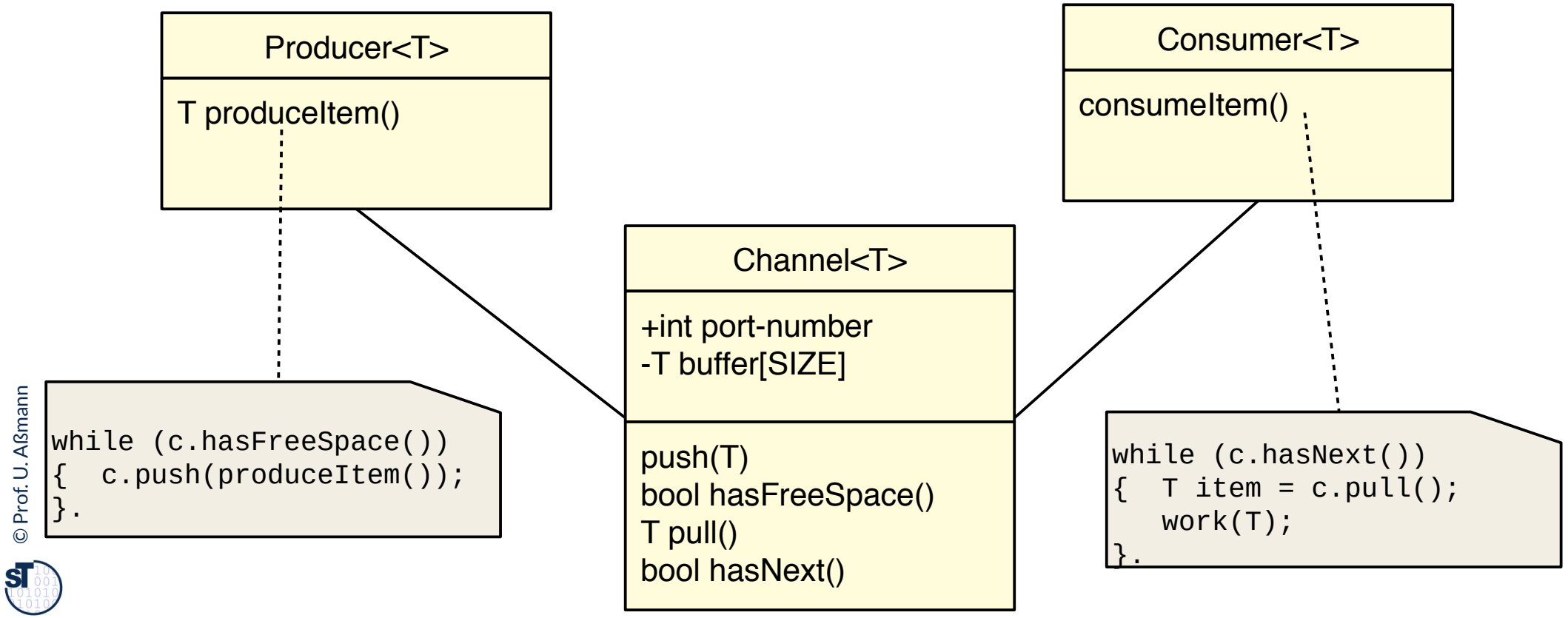


Channels in anderen Programmiersprachen

- Channels (pipes) kommen in vielen Sprachen als Konstrukte vor
- Linux versieht jeden Prozess mit 3 Streams (stdin, stdout, stderr), zwischen denen Kanäle gezogen werden können.
- Textuelle Syntax:
  - **Shell-Skripte in Linux** bestehen oft aus Produzenten-Konsumenten-Teams, die mit Kanal kommunizieren (Operator für pipes: "|")
    - `$ ls | wc`
    - `$ cat file.txt | grep "Rechnung"`
    - `$ sed -e "s/Rechung/Bestellung/" < file.txt`
  - **Communicating Sequential Processes** (CSP [Hoare], Ada, Erlang):
    - Operator für pull: "?"
    - Operator für push: "!"
  - **C++**: Eingabe- und Ausgabestream stdin, stdout, stderr
    - Operatoren "<<" (read, pull) und ">>" (push, write)
  - Architectural Description Languages (ADL, Kurs CBSE)
- Sie sind ein elementares Muster für die Kommunikation von parallelen Prozessen (producer-consumer-Muster)

Wie organisiere ich die Kommunikation zweier Aktoren?

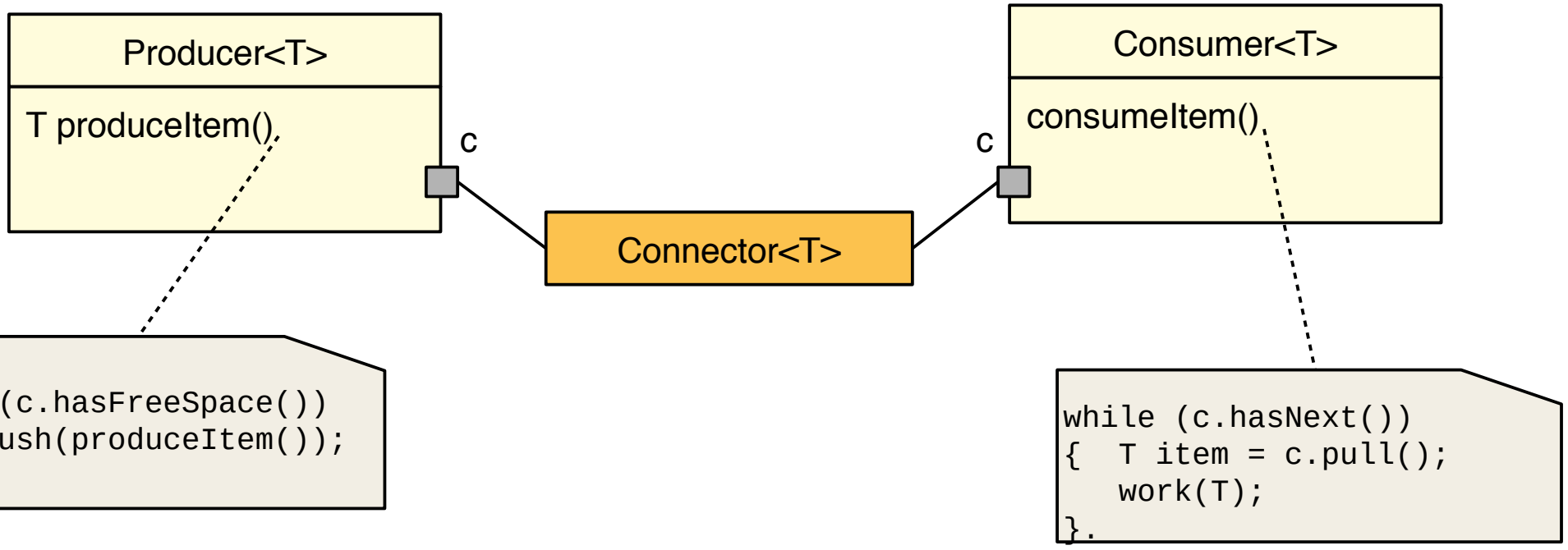
- Einsatzzweck: Ein ***Aktor (Prozess)*** ist ein parallel arbeitendes Objekt. Zwei Aktoren können mit Hilfe eines Kanals kommunizieren und lose gekoppelt arbeiten
- Bsp.: Pipes mit ihren Endpunkten (Sockets) organisieren den Verkehr auf den Internet; sie bilden Kanäle zur Kommunikation zwischen Prozessen (Producer-Consumer-Muster)



Konnektoren als Verallgemeinerung von Kanälen

Wdh.:
Def.: Eine **Konnektorklasse** ist eine Teamklasse, die zur Kommunikation von Mitgliedern eines Teams dient.

- Konnektoren sind bi- oder multidirektional, sie fassen zwei oder mehrere Kanäle zusammen; Kanäle bilden spezielle *gerichtete Konnektoren*
- Kommunikation über Konnektoren muss nicht gerichtet sein; es können komplexe Protokolle herrschen
- Konnektoren können mehrere Input und Output Streams koppeln

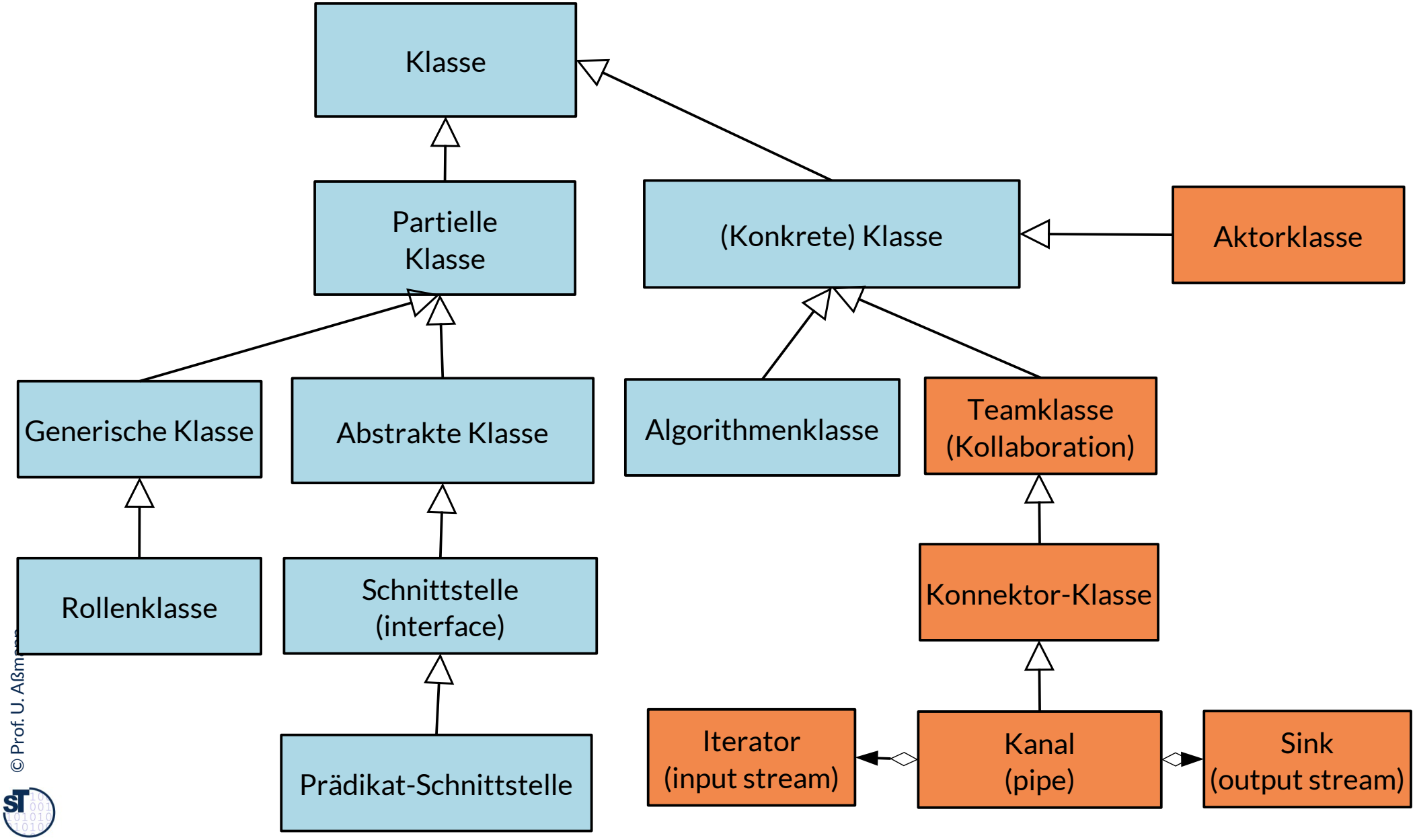


Konnektor-Gesetz der Softwarearchitektur

Trenne Konnektoren von Anwendungsklassen ab, um die Wiederverwendung zu erhöhen

- Konnektoren gehören zum olympischen Ring “Infrastruktur”

Q2: Begriffshierarchie von Klassen (Erweiterung)

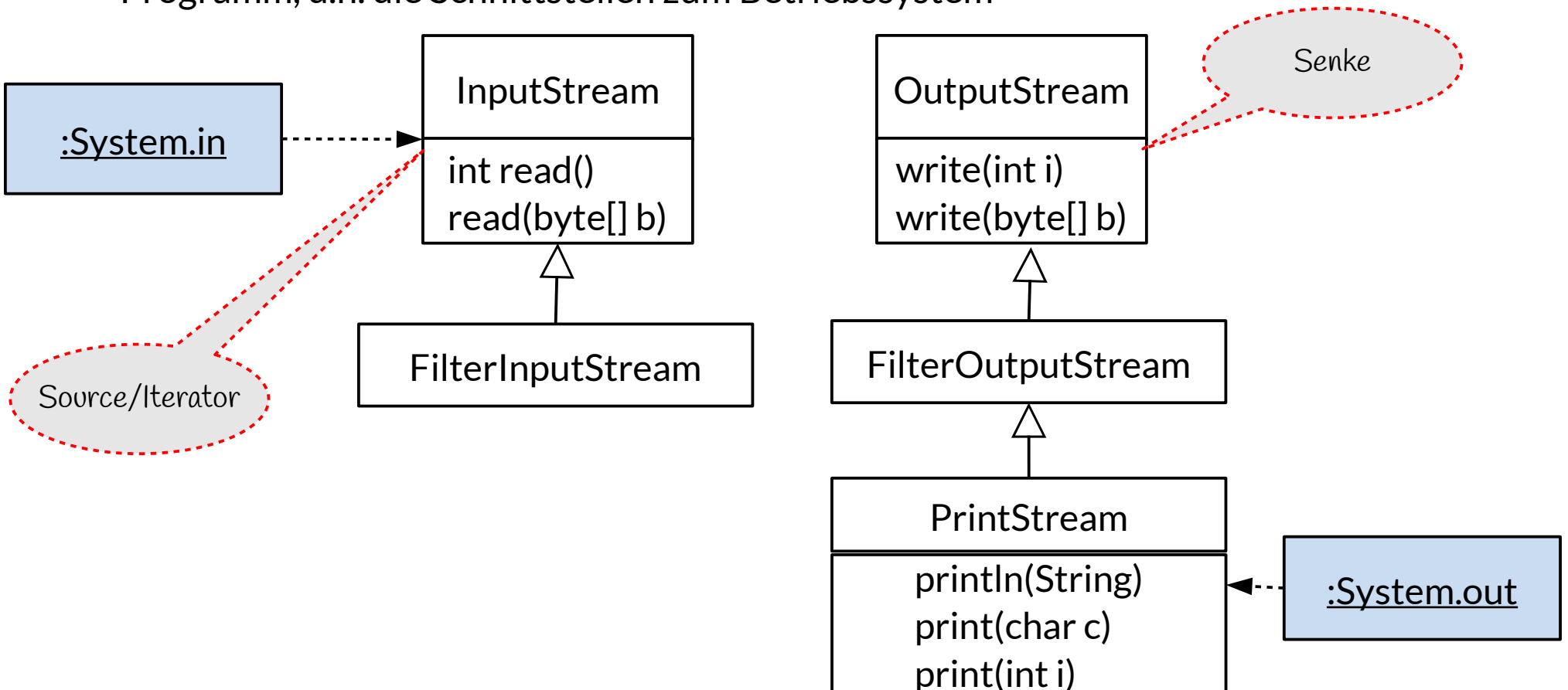


23.4 Channels und Streams sind überall: Strombasierte Ein- und Ausgabe (Input/Output) und persistente Datenhaltung

- In der Regel wird in einer Programmiersprache Ein- und Ausgabe über Ströme bzw. Kanäle realisiert.
- Das JDK nutzt Iteratoren/Streams an verschiedenen Stellen

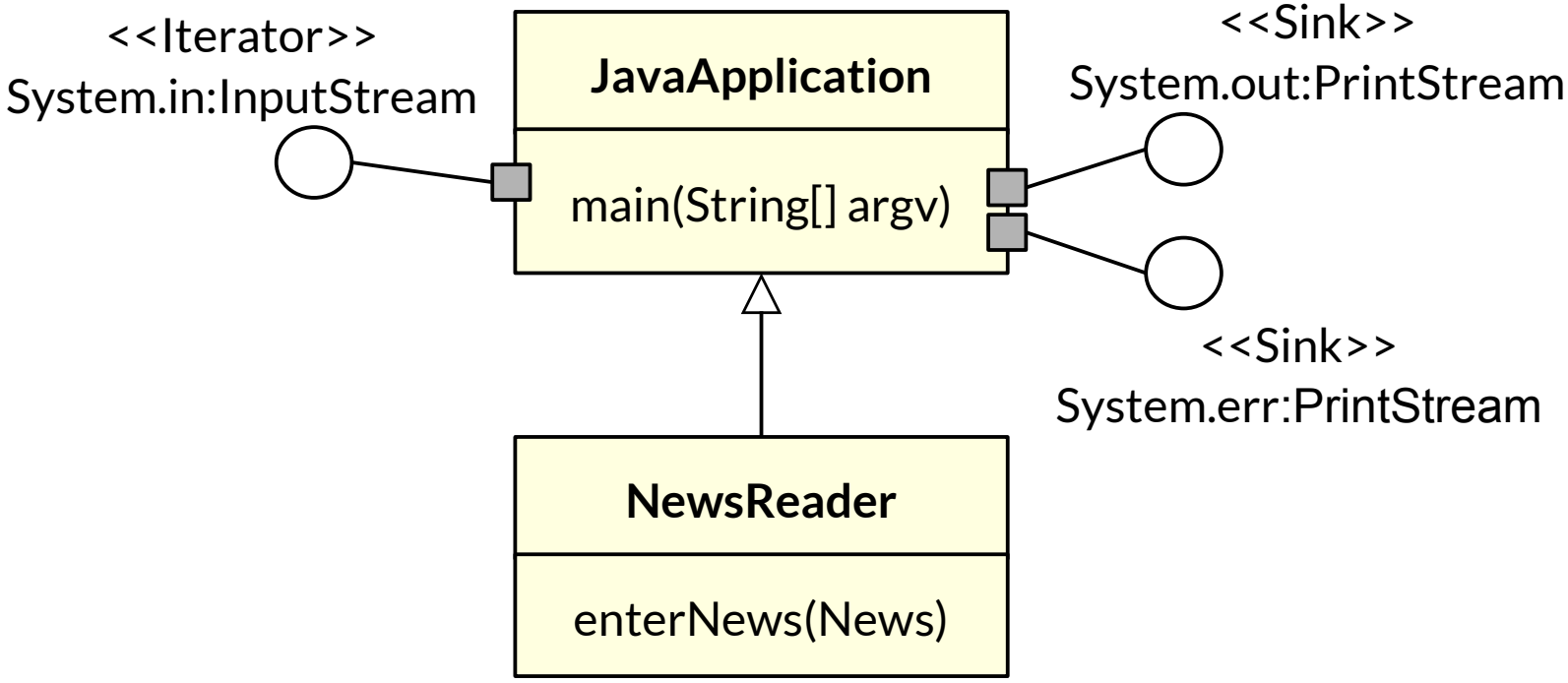
23.4.1 Ein- und Ausgabe in Java mit Strömen und Senken

- Die Klasse **`java.io.InputStream`** stellt einen Iterator/Stream in unserem Sinne dar. Sie enthält Methoden, um Werte einzulesen
- **`java.io.OutputStream`** stellt eine Senke dar. Sie enthält Methoden, um Werte auszugeben
- Die statischen Objekte `in, out, err` bilden die Sinks und Streams in und aus einem Programm, d.h. die Schnittstellen zum Betriebssystem



Java-Anwendungen mit ihren Standard-Ein/Ausgabe-Strömen

- Ein Programm in Java hat 3 Standard-Ströme `in, out, err`
  - Entwurfsidee stammt aus dem UNIX/Linux-System
- Notation: UML-Komponenten



23.4.2 Temporäre und persistente Daten mit Streams organisieren

- Daten sind
  - ***temporär***, wenn sie mit Beendigung des Programms verloren gehen, das sie verwaltet;
  - ***persistent***, wenn sie über die Beendigung des verwaltenden Programms hinaus erhalten bleiben.
    - Steuererklärungen, Bestellungen, ...
- Objektorientierte Programme benötigen Mechanismen zur Realisierung der *Persistenz von Objekten*.
  - Einsatz eines Datenbank-Systems (Siehe Vorlesung “Datenbanken”)
    - Objektorientiertes Datenbank-System
    - Relationales Datenbank-System
      Java: Java Data Base Connectivity (JDBC)
    - Zugriffsschicht auf Datenhaltung
      Java: Java Data Objects (JDO)
  - Speicherung von Objektstrukturen in Dateien mit Senken und Iteratoren
    - Objekt-Serialisierung *(Object Serialization)*
    - Die Dateien werden als Channels benutzt:
    - Zuerst schreiben in eine Sink
    - Dann lesen mit Iterator

Objekt-Serialisierung in Java, eine einfache Form von persistenten Objekten

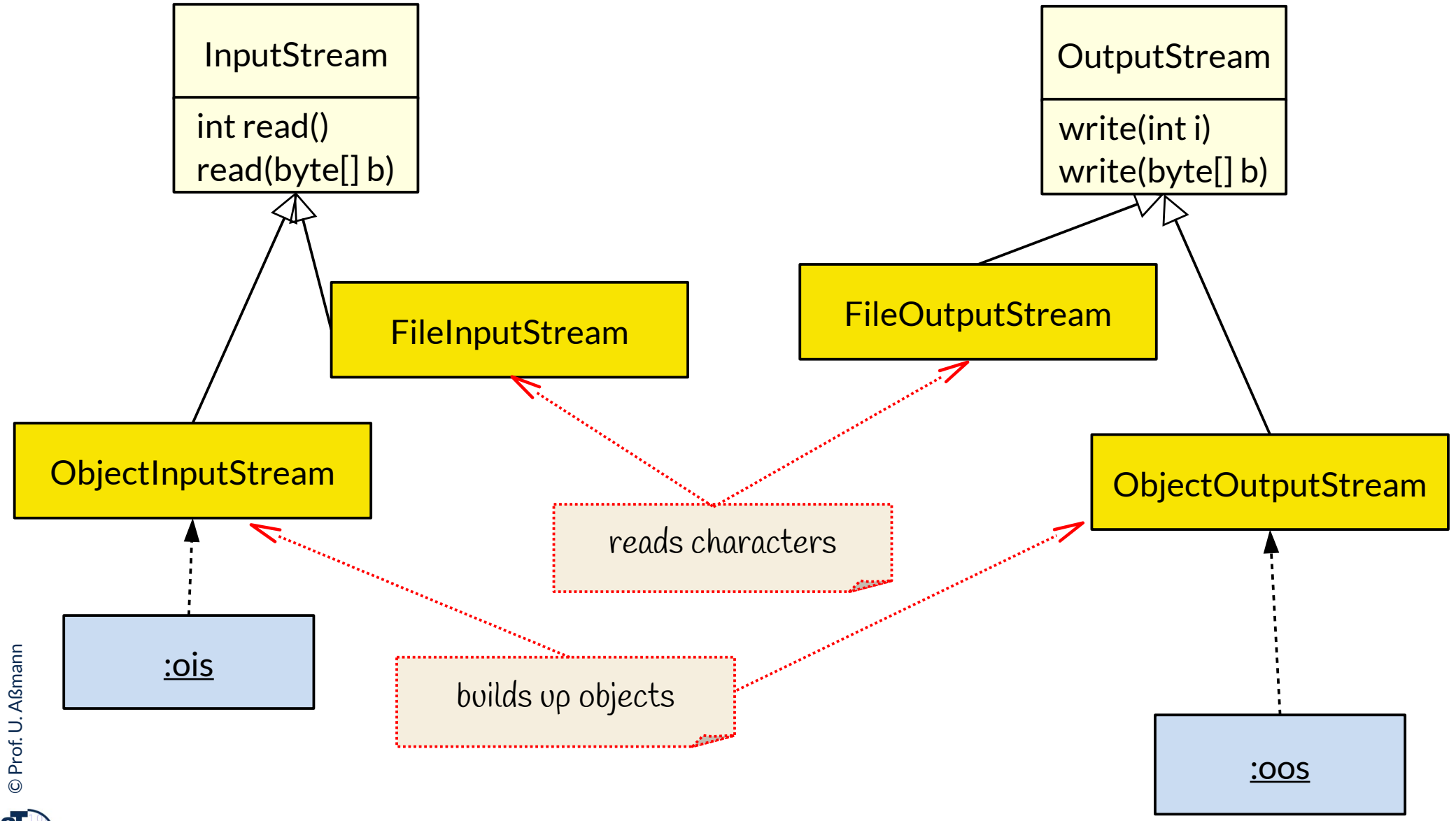
- Die Klasse **`java.io.ObjectOutputStream`** stellt eine Senke dar
  - Methoden, um ein Geflecht von Objekten linear darzustellen (zu *serialisieren*) bzw. aus dieser Darstellung zu rekonstruieren.
  - Ein OutputStream entspricht dem Entwurfsmuster Sink
  - Ein InputStream entspricht dem Entwurfsmuster Iterator
- Eine Klasse, die Serialisierung zulassen will, muß die (leere!) Prädikat-Schnittstelle **`java.io.Serializable`** implementieren.

```
class ObjectOutputStream {
  public ObjectOutputStream (OutputStream out)
     throws IOException;
  // push Method
  public void writeObject (Object obj)
     throws IOException;
}
```

Objekt-Serialisierung: Abspeichern

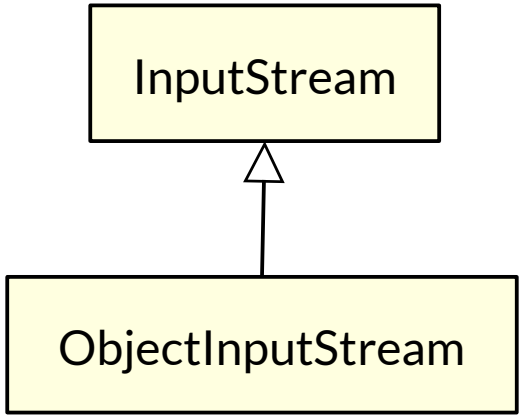

```
import java.io.*;

class XClass implements Serializable {
  private int x;
  public XClass (int x) {
    this.x = x;
  }
}

...
XClass xobj;
...
FileOutputStream fos = new FileOutputStream("Xfile.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);

// internally realized as push for all child objects
oos.writeObject(xobj); // push
...
```

Verschiedene Arten von Input und Output Streams im JDK



ObjectInputStream aus dem JDK

- **`Die Klasse java.io.ObjectInputStream`** stellt einen Iterator/InputStream in unserem Sinne dar
  - Methoden, um ein Geflecht von Objekten linear darzustellen (zu *serialisieren*) bzw. aus dieser Darstellung zu rekonstruieren (zu deseralisieren)
  - Ein OutputStream entspricht dem Entwurfsmuster Sink
  - Ein InputStream entspricht dem Entwurfsmuster Iterator



Objekt-Serialisierung: Einlesen mit ObjectInputStream aufsetzend auf FileInputStream


```
import java.io.*;

class XClass implements Serializable {
  private int x;
  public XClass (int x) {
    this.x = x;
  }
}

...
XClass xobj;
...
FileInputStream fis = new FileInputStream("Xfile.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
// internally realised as pull
xobj = (XClass) ois.readObject(); // pull
```

23.4.3 Ereignisse und Kanäle

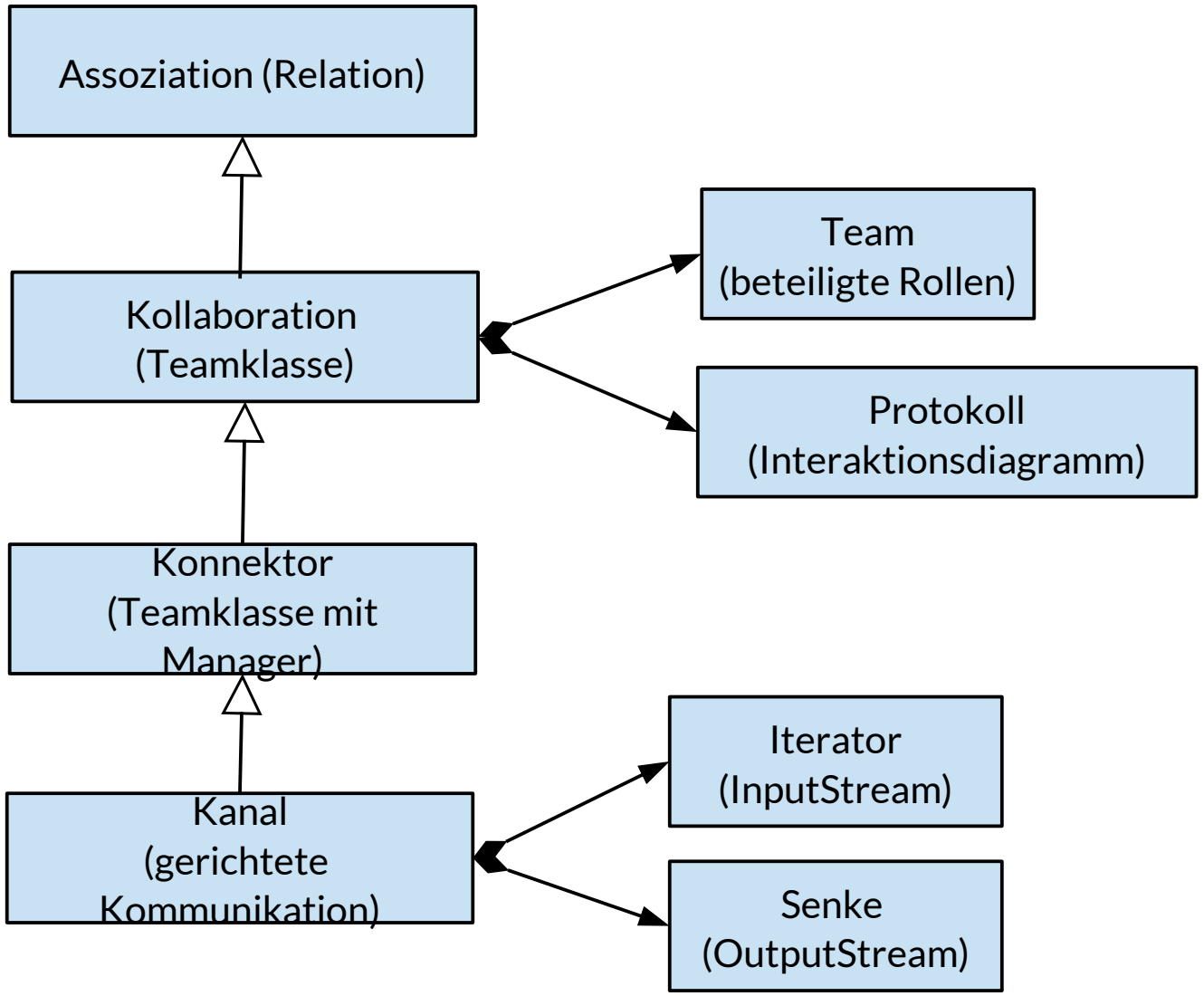
- Kanäle (gerichtete Konnektoren) eignen sich hervorragend zur Kommunikation mit der Außenwelt, da sie die Außenwelt und die Innenwelt eines Softwaresystems **entkoppeln**
- Ereignisse können in der Außenwelt **asynchron** (“losgelöst vom System”) stattfinden und auf einem Kanal in die Anwendung transportiert werden
  - Dann ist der Typ der Daten ein Ereignis-Objekt
  - In Java wird ein externes oder internes Ereignis immer durch ein Objekt repräsentiert

The End

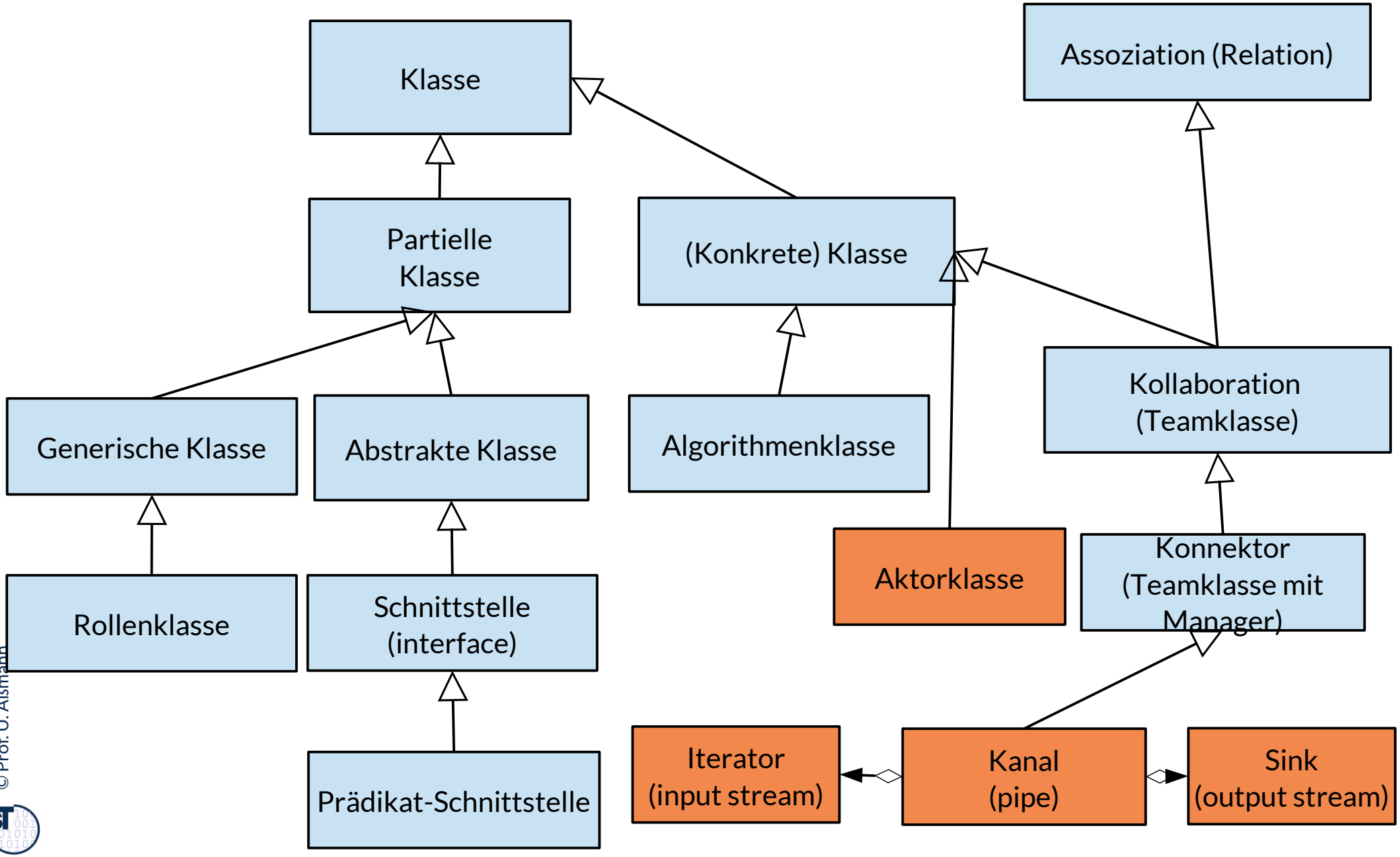
- Was unterscheidet ein Team von einer Kollaboration?
- Was unterscheidet einen Iterator von einer Senke?
- Was ist der Unterschied zwischen einer fixen und einer unbestimmt großen Datenstruktur?
- Warum läuft Ein- und Ausgabe in Programmen immer über Ströme?
- Wie heißen die Java-Stromobjekte für Ein- und Ausgabe?
- Was unterscheidet ein Team von einer Kollaboration?
- Wann macht es Sinn, die Kollaboration eines Teams durch ein Objekt, den Konnektor, zu kapseln?
- Warum ist ein Kanal ein spezieller Konnektor?
- Was ist besonders genial am UNIX/Linux Ein-/Ausgabesystem?
- Einige Folien stammen aus den Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

Anhang

Relationale Klassen (Konnektoren)



Q2: Begriffshierarchie von Klassen (Erweiterung)



Iterator-Implementierungsmuster in modernen Sprachen

- In vielen Programmiersprachen (Sather, Scala, Ada) stehen **Iteratormethoden (stream methods)** als spezielle Prozeduren zur Verfügung, die die Unterobjekte eines Objekts liefern können
  - Die yield-Anweisung gibt aus der Prozedur die Elemente zurück
  - Iterator-Prozedur kann mehrfach aufgerufen werden und damit als input-stream verwendet werden
  - Beim letzten Mal liefert sie null

```
class bigObject {
  private List subObjects;
  public iterator Object deliverThem() {
    while (i in subObjects) {
       yield i;
       // Dieser Punkt im Ablauf wird sich als Zustand gemerkt
       // Beim nächsten Aufruf wird hier fortgesetzt
    }
  }
}
.. BigObject bo = new BigObject(); ...
..  a = bo.deliverThem();
    b = bo.deliverThem();..
```