

# Software Engineering

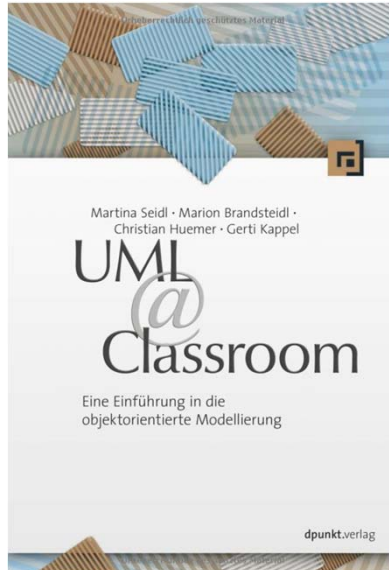
## 6. Modellierungstechniken

Ruth Breu

# Übersicht

- 6.1 Statische Struktur von Systemen
- 6.2 Grundlagen der Verhaltensmodellierung
- 6.3 Interaktionsdiagramme
- 6.4 Zustandsdiagramme
- 6.5 Aktivitätsdiagramme
- 6.6 Datenflussdiagramme

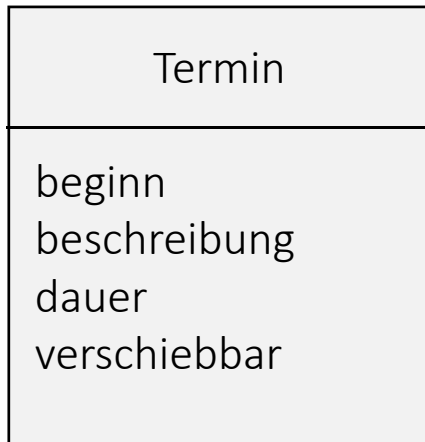
# Literatur



# 6.1 Statische Struktur von Systemen

- Klassendiagramme
- Objektdiagramme
- Komponentendiagramme -> VO Software-Architektur

# Klassendiagramme

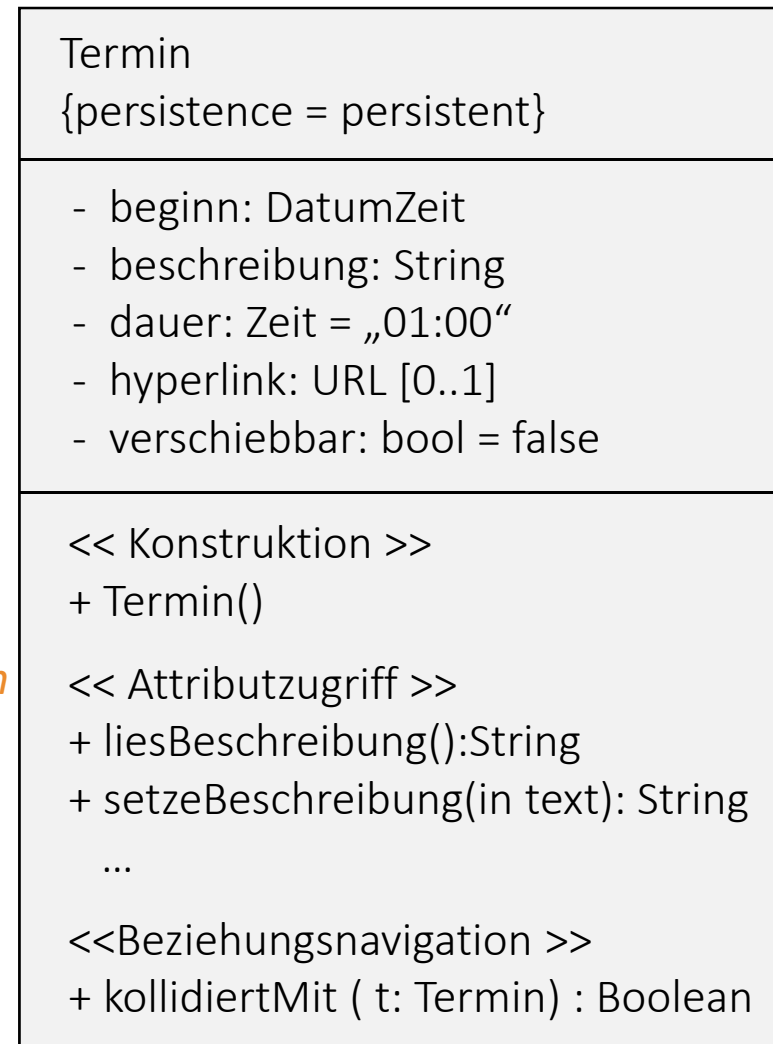


Klasse in einem fachlichen Klassendiagramm

- Klassen in Klassen-  
diagrammen können  
auf unterschiedlichem  
Detaillierungsniveau  
dargestellt werden

*Attribute*

*Operationen*

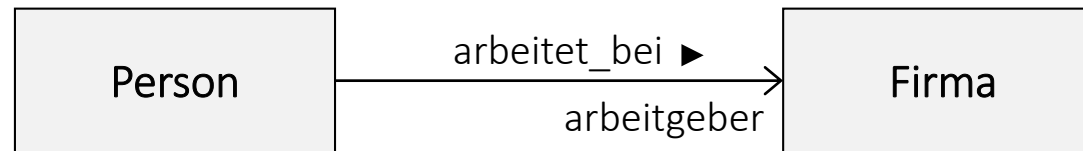


Klasse auf Ebene der SW-Architektur

# Typisierung in UML

- UML enthält keine Vorgabe für die Typisierung von Attributen und Methodenparametern
  - Möglichkeit 1: Verwendung einer Spezifikationssprache für die Definition von Typen, im Kontext der UML können OCL-Typen verwendet werden
  - Möglichkeit 2: Verwendung von Typausdrücken einer Programmiersprache
  - Für die Definition von Default-Werten gibt die UML ebenfalls keine Syntax vor

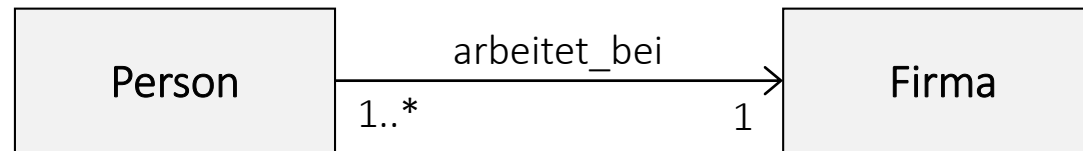
# Assoziationen



- Eine Assoziation beschreibt eine strukturelle Beziehung zwischen Objekten der verbundenen Klassen
  - Jedes Objekt der Klasse **Person** hat eine Beziehung zu einem Objekt (oder mehreren) der Klasse **Firma**
  - Die Firma hat in der gegebenen Beziehung die Rolle des Arbeitgebers
  - Beziehungsname und Rollenname sind optional

► = Leserichtung (optional)

# Assoziationen - Kardinalitäten (*engl. multiplicities*)

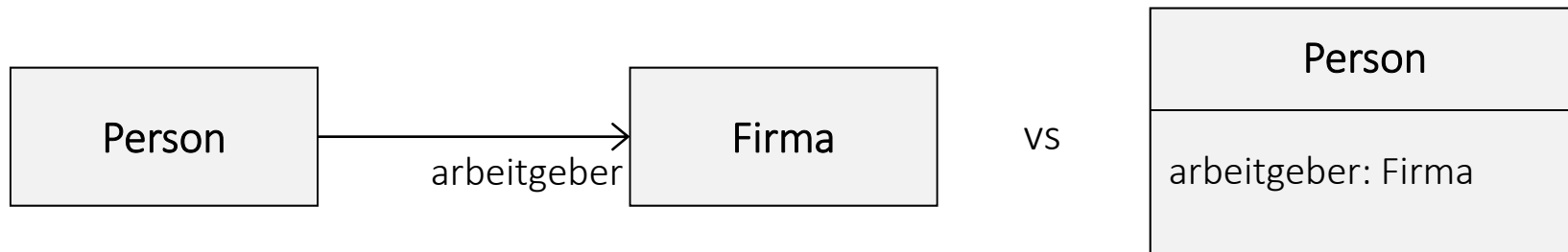


*„Jede Person arbeitet bei genau einer Firma,  
beliebig viele Personen (aber mindestens eine) arbeiten in einer Firma“*

Allgemein:	a..b	a natürliche Zahl, b natürliche Zahl oder *
	z.B. 3..*	3, 4, 5, ... (beliebig viele, aber mindestens 3)
	0..*	beliebig viele
	1 = 1..1	genau ein
	0..1	optionale Beziehung
	default:	0..*



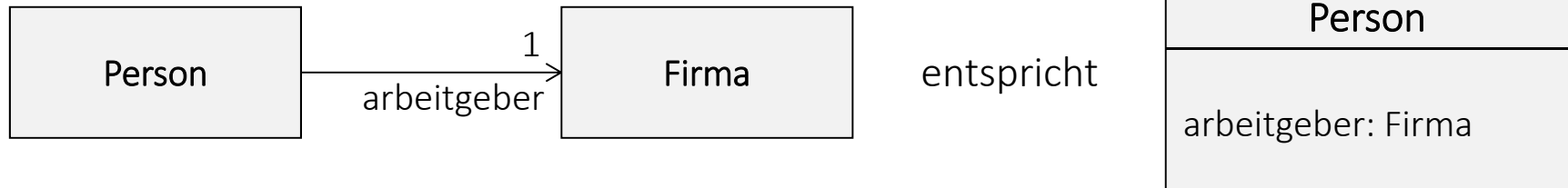
# Assoziationen und Attribute



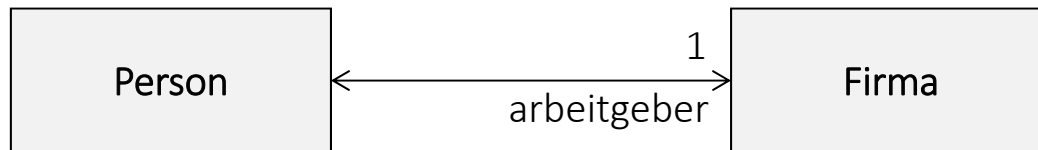
- (Gerichtete) Assoziationen und Bezugsattribute haben die gleiche Bedeutung
- Assoziationen mit Kardinalität 0..1 oder 1 können direkt durch Attribute implementiert werden
  - oft wird der Rollename als Attributname verwendet
- bei anderen Kardinalitäten: Verwendung von Containern bei der Implementierung
- Problem: in frühen Phasen des Entwurfs ist die Richtung einer Assoziation oft nicht klar
  - in diesem Fall bleibt die Assoziation ungerichtet

# Gerichtete und ungerichtete Assoziationen (1)

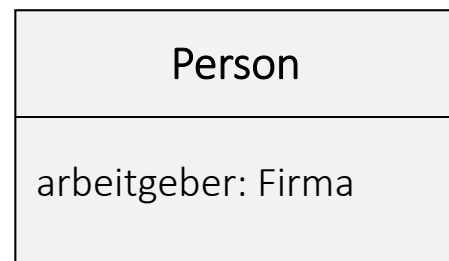
einseitig gerichtet



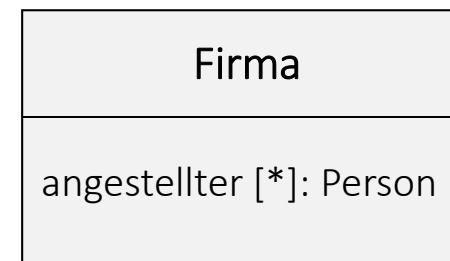
beidseitig gerichtet



entspricht



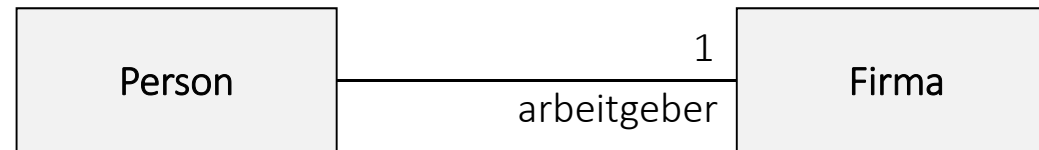
und



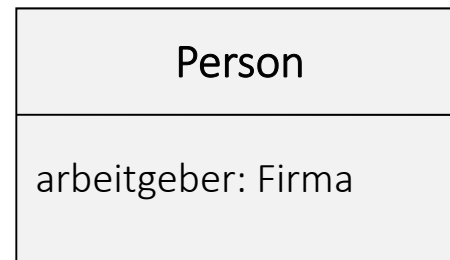
Achtung: Doppelseitiger Bezug muss konsistent gehalten werden!

# Gerichtete und ungerichtete Assoziationen (2)

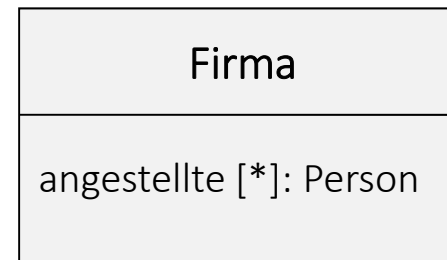
ungerichtet



entspricht

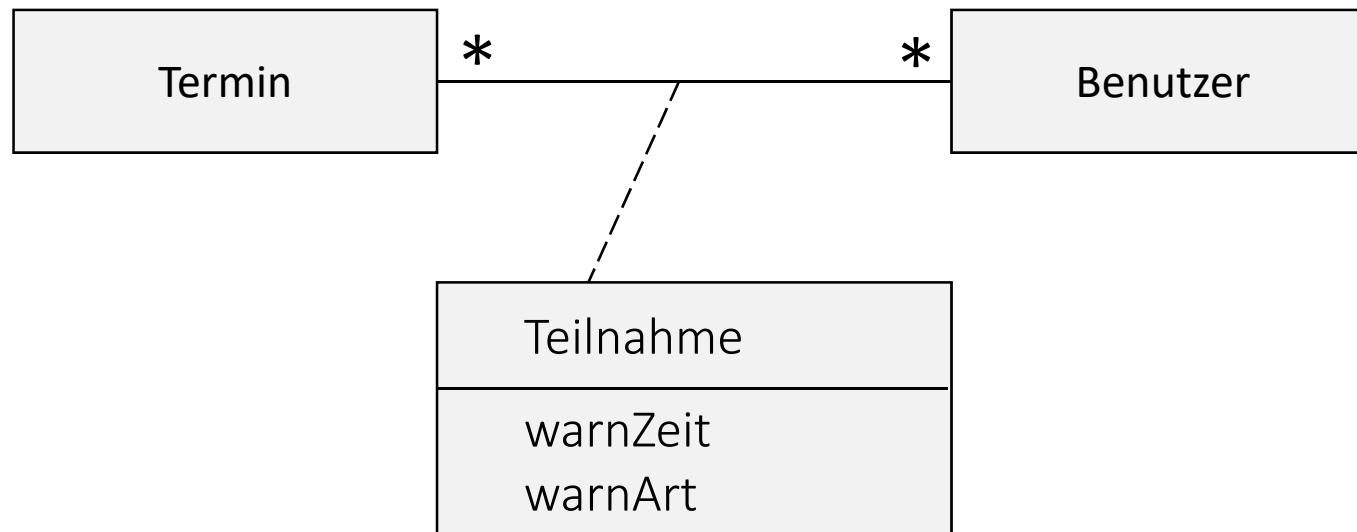


oder

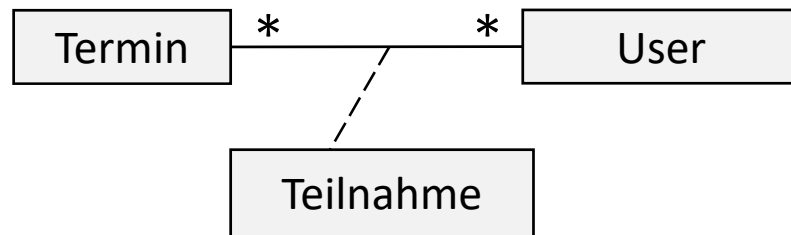


- tatsächliche Implementierung der Assoziation wird auf spätere Phasen verschoben

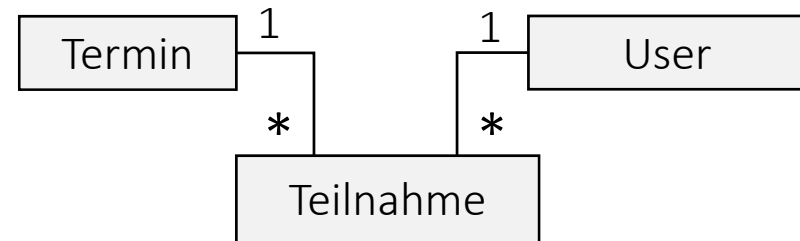
# Assoziationsklassen



# Assoziationsklassen vs Assoziationen



(1)



(2)

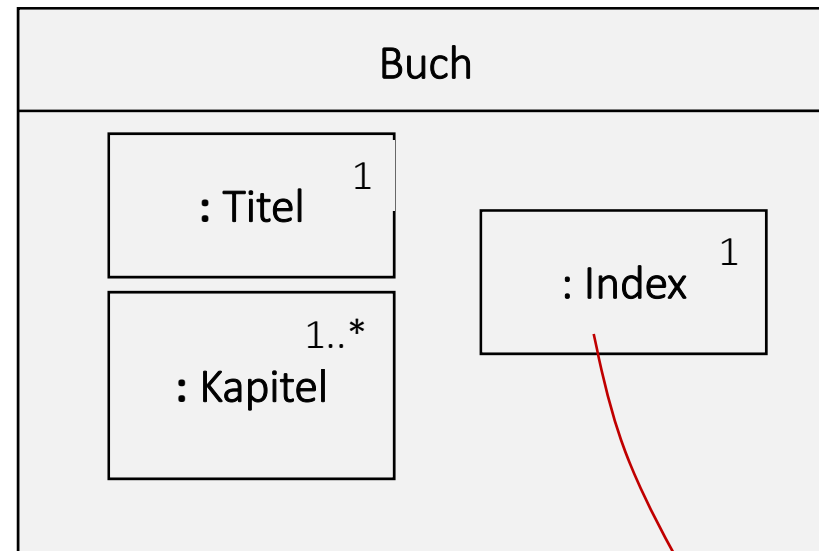
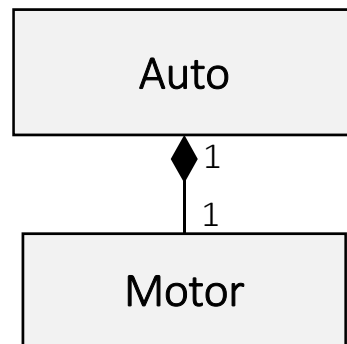
- In Modell (2) kann ein Benutzer über zwei Teilnahmeobjekte mit dem gleichen Terminobjekt verbunden sein, in Modell (1) ist dies nicht möglich.



Instanz von (2), aber nicht von (1)

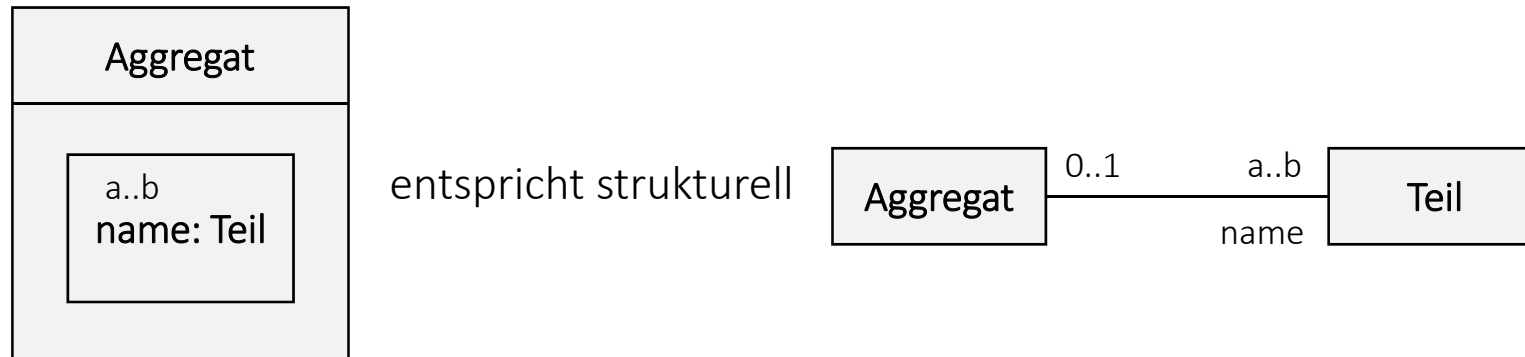
# Komposition und Aggregation

- „ist\_Teil\_von“-Beziehung



- spezielle Art von Assoziation
- Konstrukt der Modellierung
- UML unterscheidet zwischen Aggregation und Komposition – diese Unterscheidung ist allerdings unklar, die beiden Begriffe werden deshalb im Folgenden synonym verwendet

# Aggregation und Assoziation

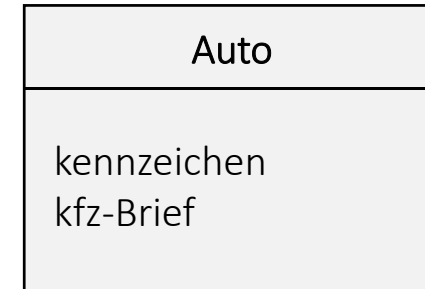


*d.h. jedes Teilobjekt ist in jedem Systemzustand höchstens einem Aggregatobjekt zugeordnet*

# Potentielle Eigenschaften der Komposition (1)

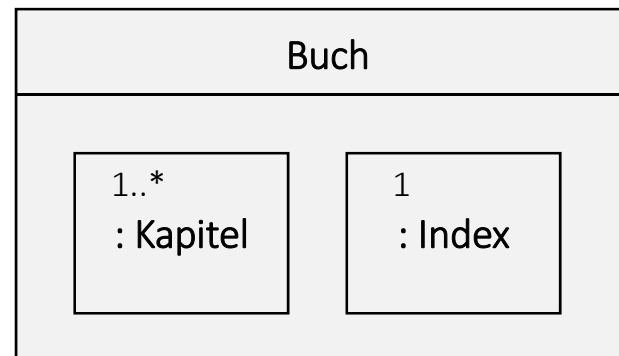
- „Ganzes“ ist selbst Klasse

*Ganzes ist mehr als die Summe seiner Teile*



- Konstante Zuordnung von Teil- zu Aggregatobjekten

*Ein Kapitel bleibt immer dem gleichen Buch zugeordnet*





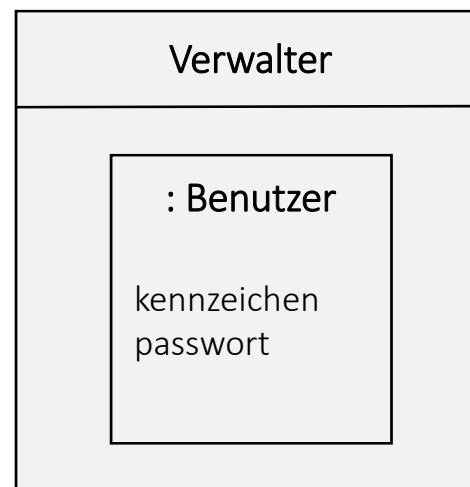
## Potentielle Eigenschaften (2)

- Abhängigkeit der Lebenszeiten

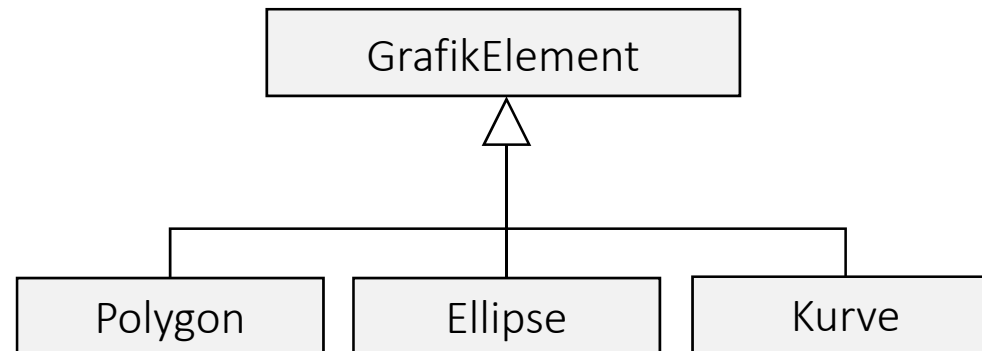
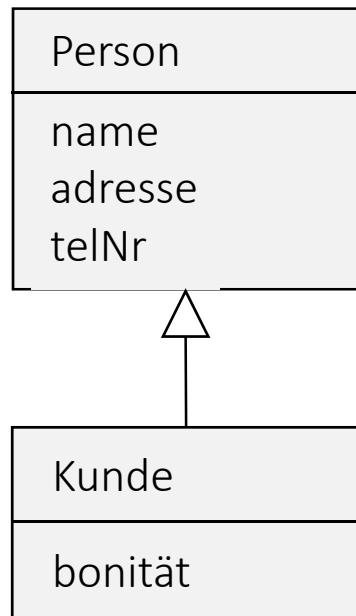
*Ein Kapitel-Objekt kann nicht ohne ein Buch-Objekt existieren*

- Kapselung der Teilklassen

*Objekte von Teilklassen können nur Nachrichten von Aggregatobjekten oder von anderen Teilobjekten der Aggregation empfangen*



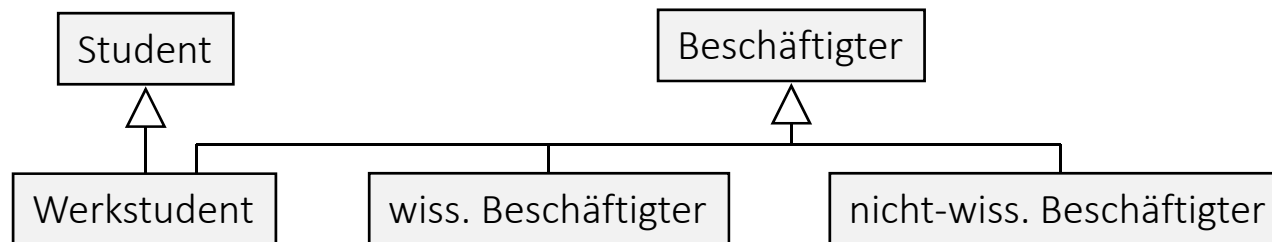
# Generalisierung



- Manifestieren von Ähnlichkeiten im Modell
- Subklassen besitzen alle Attribute, Assoziationen und Operationen ihrer Superklassen
- Ein Objekt einer Subklasse ist auch Instanz der Superklasse (Polymorphismus)

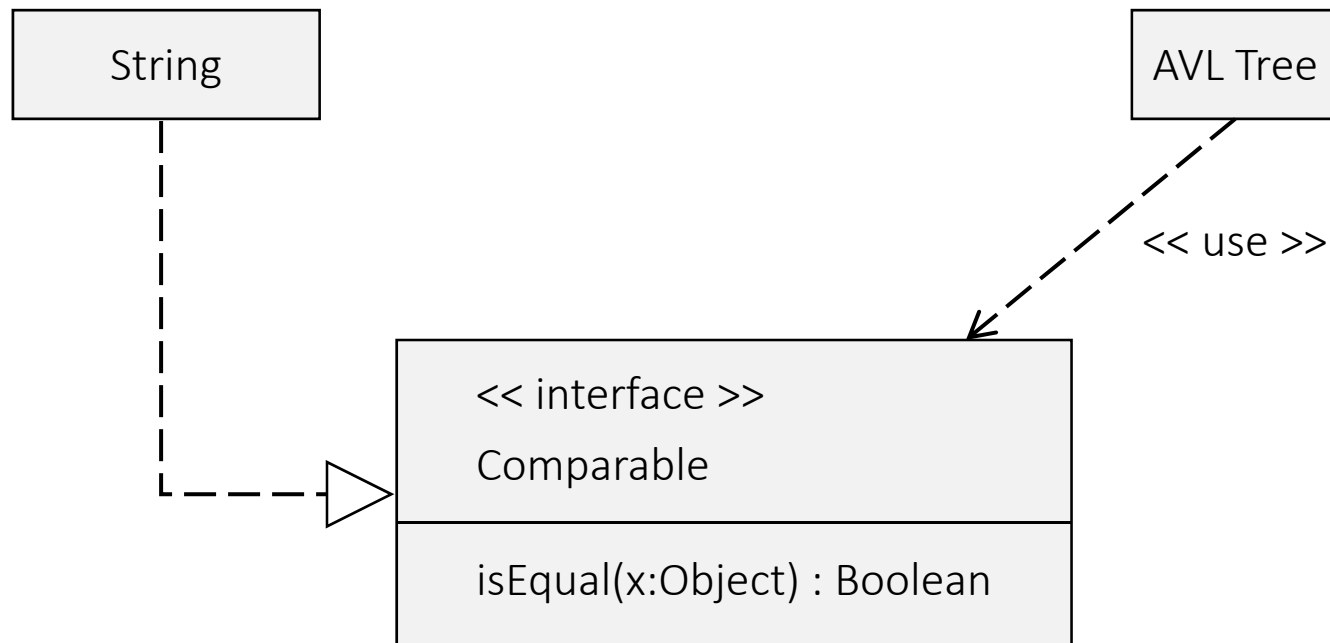
# Mehrfache Generalisierung

- UML unterstützt auch vielfache Generalisierung (eine Subklasse hat mehrere Superklassen)

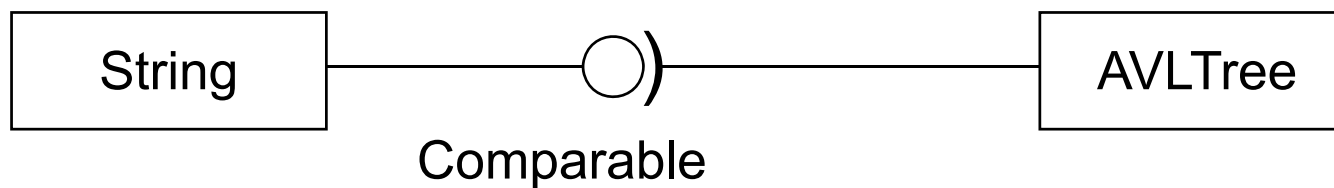


- Achtung: Verwendung mehrfacher Generalisierungsbeziehungen kann zu unüberschaubaren Strukturen führen!

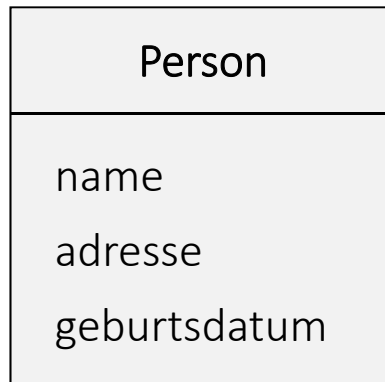
# Interfaces



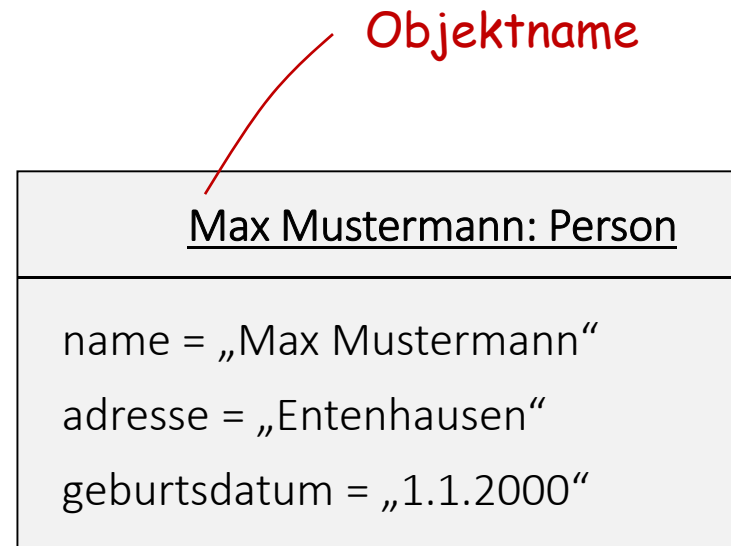
**Kompakt:**



# Objektdiagramme



Klasse

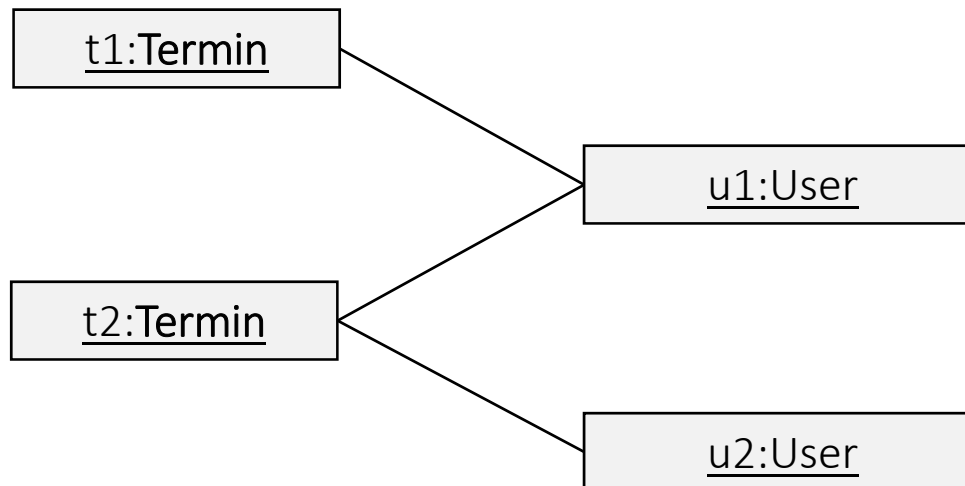


Objekt



anonymes Objekt

# Assoziationen auf Objektebene



# Klassen- und Objektdiagramm

- Ein UML-Objektdiagramm ist ein UML-Klassendiagramm, das nur Konzepte der Instanzebene (d.h. Objekte und ihre Beziehungen) enthält
- Typ- und Instanzebene sollten nie vermischt werden, deshalb verstehen wir unter Klassendiagrammen im folgenden nur Diagramme, die ausschließlich Konzepte der Typebene enthalten
  - d.h. Klassen, Interfaces, Pakete und ihre Beziehungen
- Verwendung von Objektdiagrammen
  - Beschreibung von **beispielhaften** Objektstrukturen

## 6.2 Grundlagen der Verhaltensmodellierung

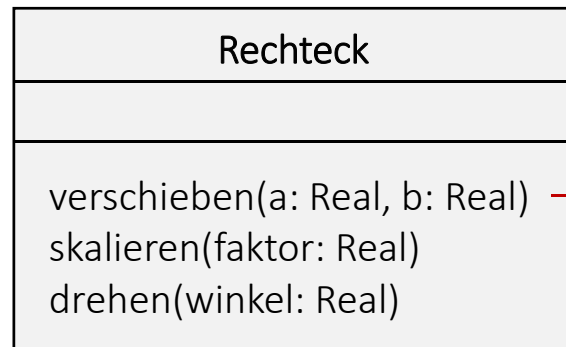
- Objekte kommunizieren miteinander durch Senden und Empfangen von **Nachrichten** (*messages*)
- **Ereignis** (*event*) = Senden einer Nachricht bzw. Eintreffen einer Nachricht bei einem Objekt
- Ein Ereignis löst Reaktionen bei dem betroffenen Objekt aus
  - Änderung von Attributwerten
  - Senden weiterer Nachrichten



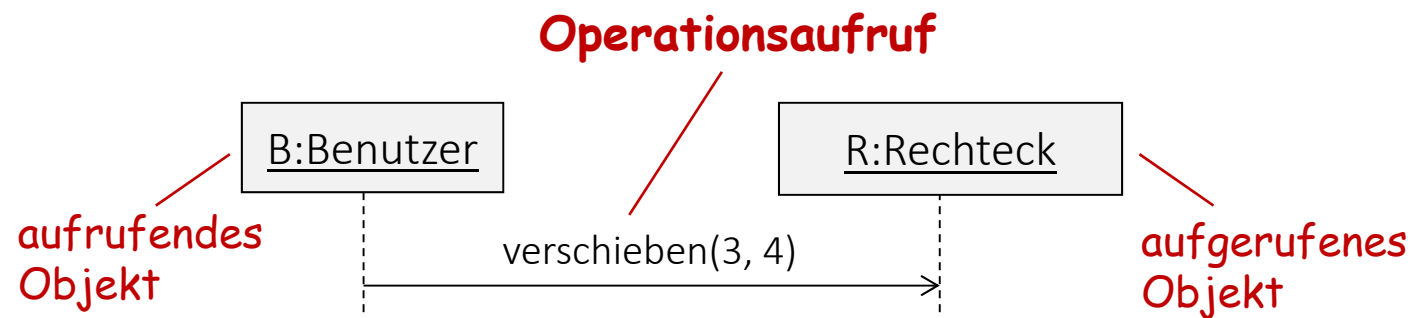
# Nachricht und Nachrichtenaustausch

- Nachrichten haben einen Namen und optionale Parameter  
    `drucken_beenden(drucker1)`
- Arten von Nachrichten:
  - Operationsaufrufe  
    `skalieren(3), umfang`
  - von Operationen zurückgegebene Werte  
    `return(5)`
  - externe Nachrichten  
    Zeitereignisse: „es ist Monatsende“
- Zwei Formen des Nachrichtenaustauschs
  - synchron  
    *Beispiel: Telefongespräch*
  - asynchron  
    *Beispiel: Senden einer E-Mail*

# Nachrichten und Operationen (1)

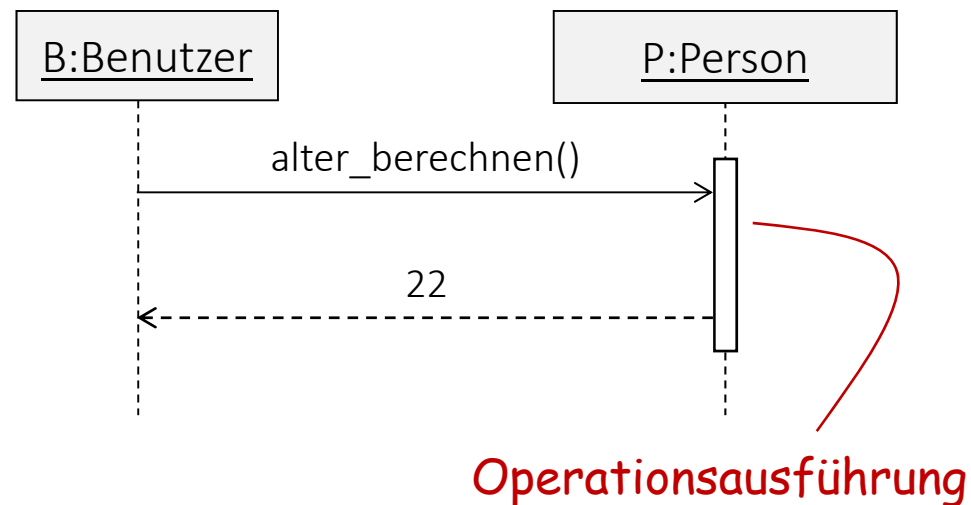
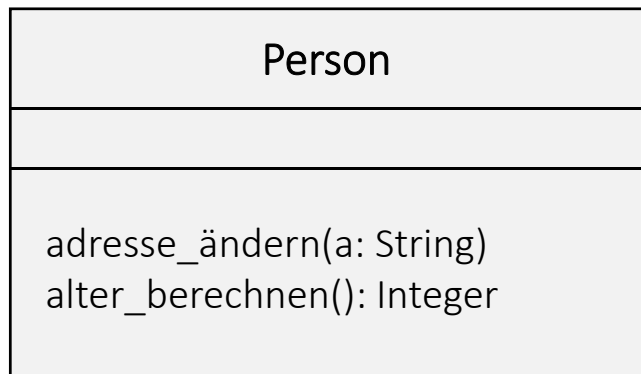


**Operation**



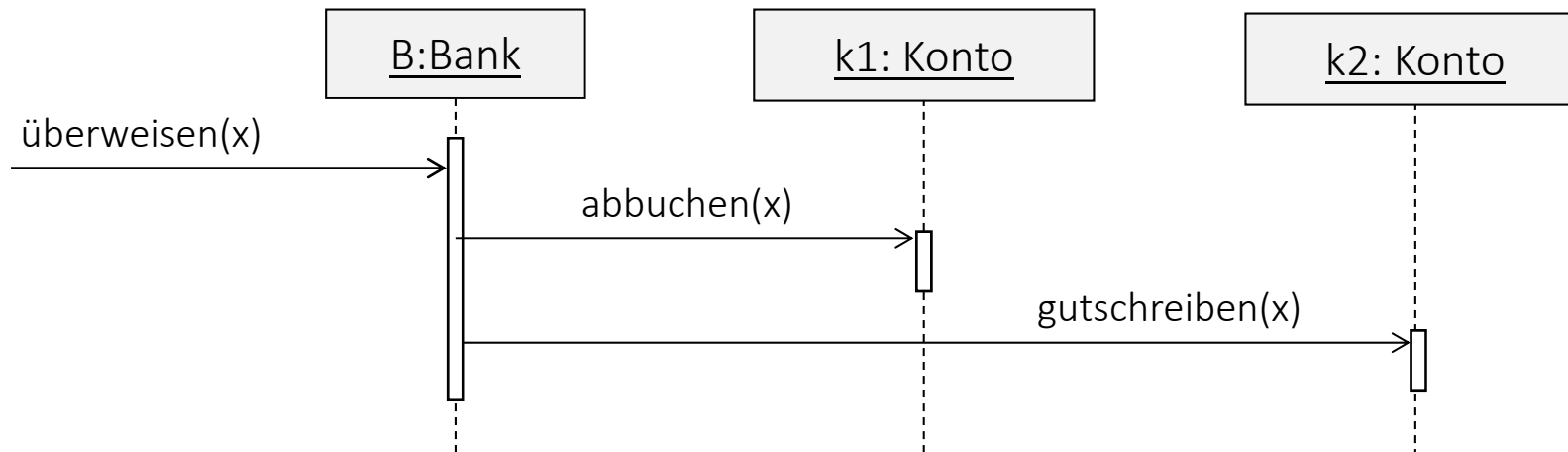
- Operationsaufrufe sind Nachrichten

## Nachrichten und Operationen (2)



- Objekte senden im Rahmen einer Operationausführung Nachrichten an den Aufrufer zurück

## Nachrichten und Operationen (3)

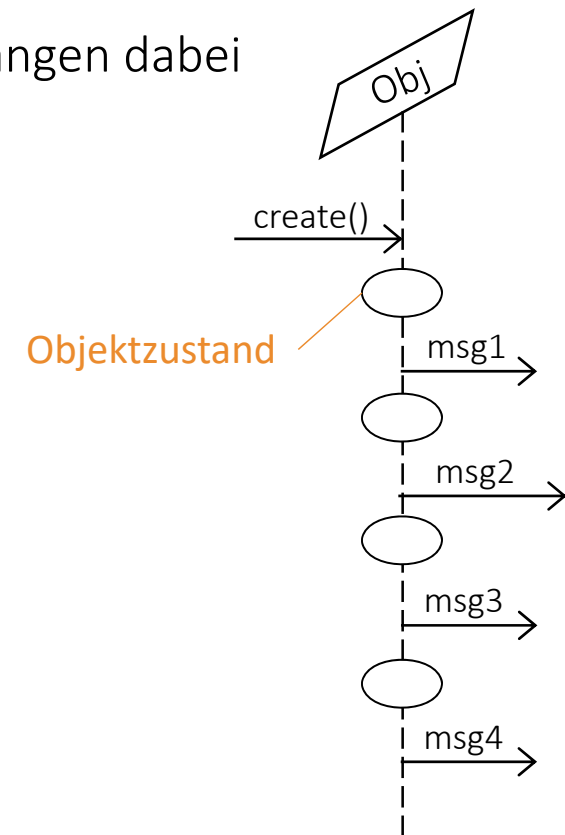


- Allgemein: Eine Operationsausführung besteht aus einer Folge von Nachrichten
  - Operationsaufruf
  - zurückgesendete Antwort
  - evtl. andere aufgerufene Operationen

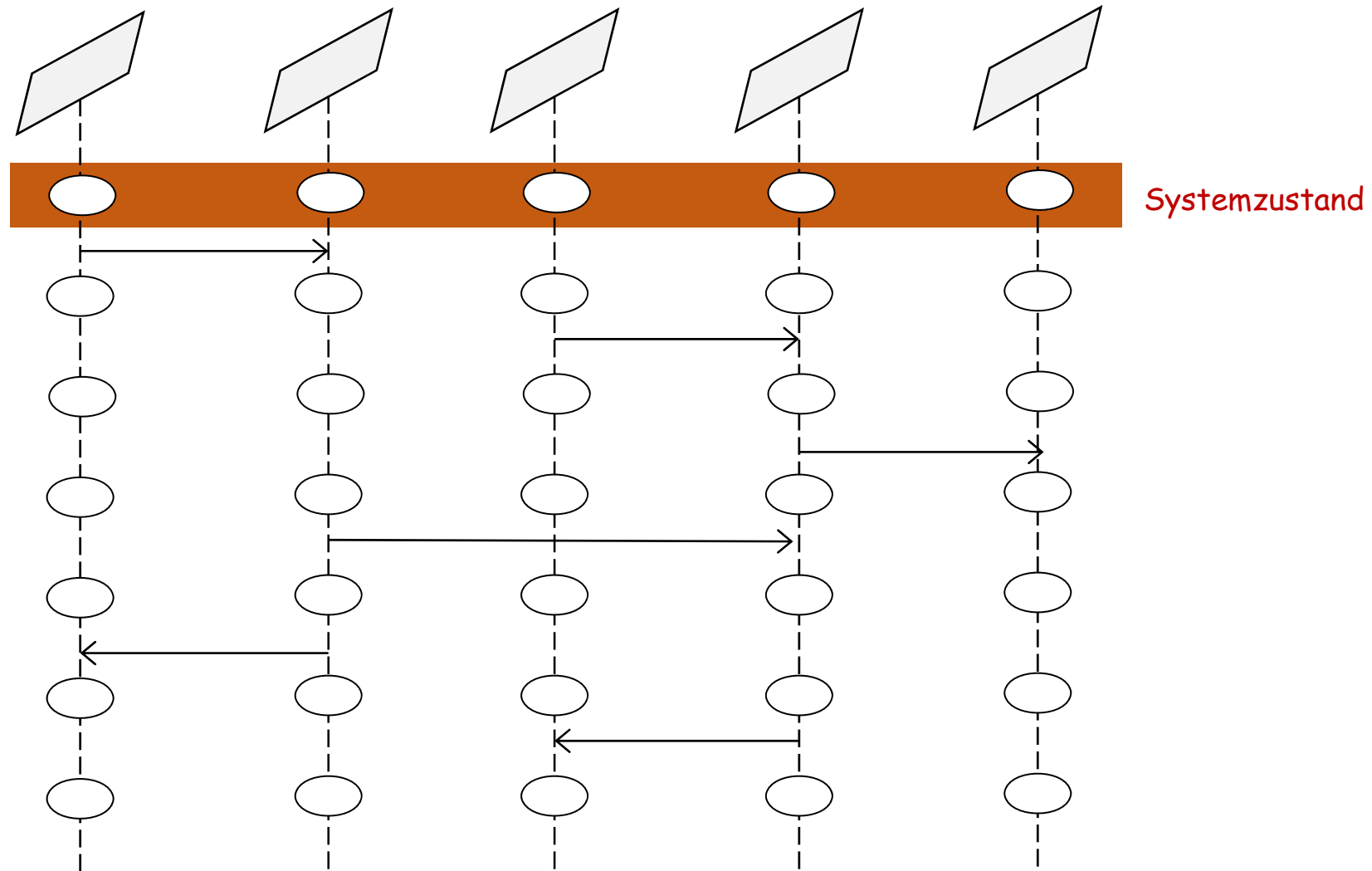
# Objektlebenszyklus

Objekte haben einen **Lebenszyklus**

- sie werden **kreiert**
- sie führen **Operationen** aus, senden und empfangen dabei Nachrichten und verändern ihren Zustand
- sie können evtl. **gelöscht** werden



# Die objektorientierte dynamische Systemsicht



# Aspekte der Verhaltensspezifikation

- Die Diagrammtypen der Dynamik beschreiben
  - welche Folgen von Nachrichten im System ausgetauscht werden
  - welche Reaktionen diese Nachrichten auslösen
- Die einzelnen Diagrammtypen variieren dabei in Sicht und Detaillierungsgrad
  - die Sicht kann lokal oder global sein: das Verhalten eines einzelnen Objekts wird beschrieben oder die Interaktion zwischen mehreren Objekten
  - das Verhalten kann an Hand von Beispielen oder vollständig beschrieben werden
- Wichtig: Interpretation der Diagramme in einem Gesamtkontext
  - Zusammenspiel unterschiedlicher Diagramme
  - Konsistenz der Gesamtspezifikation

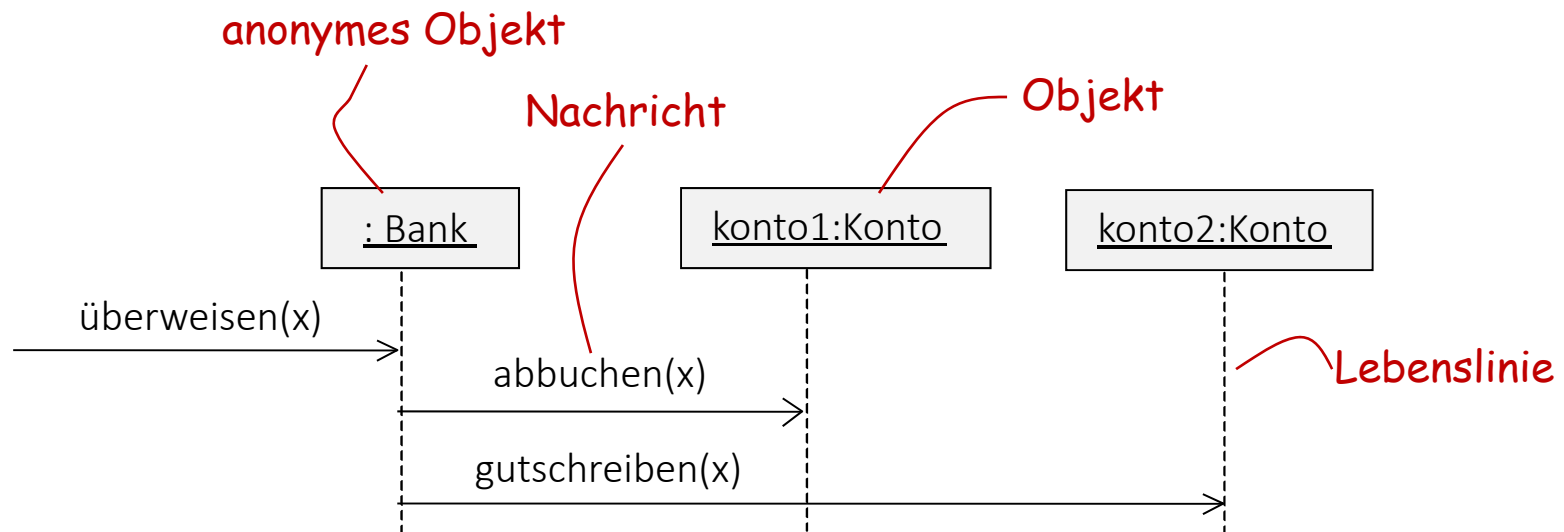
# Die UML-Beschreibungstechniken der Dynamik

- Interaktionsdiagramme (6.3)
  - Sequenzdiagramme
  - Kollaborationsdiagramme
- Zustandsdiagramme (6.4)
- Aktivitätsdiagramme (6.5)



## 6.3 Interaktionsdiagramme

### Sequenzdiagramm



Exemplarische Beschreibung von Interaktion zwischen Objekten

# Sequenzdiagramme

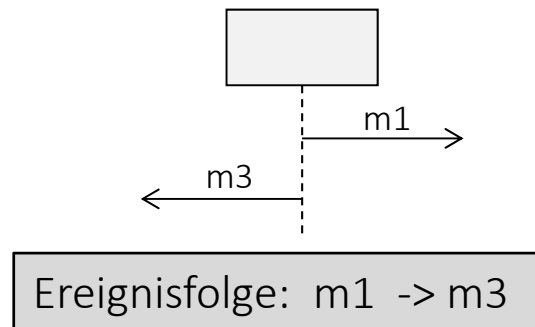
- Ursprung: *Message Sequence Charts* (MSCs)
  - entwickelt für den Telekommunikationsbereich
- Reiche Syntax
  - Referenzieren, Iteration, Fallunterscheidung, Interleaving, ...
- UML-Sequenzdiagramme haben zusätzliche Elemente angepasst auf objektorientierte Programmiersprachen
  - synchrone Kommunikation, Operationsbegriff

# Sprachelemente von UML-SDs

1. Spezifische Beschreibungselemente für die objektorientierte Sicht
  - Operationsbegriff (Aktivierung und Returns)
  - Objektkreierung und -löschen
2. Erweiterung der beschriebenen Abläufe
  - Fallunterscheidung
  - Schleifen
3. Beschreibungselemente für die Spezifikation eingebetteter Systeme
  - Unterscheidung von synchronen/asynchronen Nachrichten
  - Zeitbedingungen

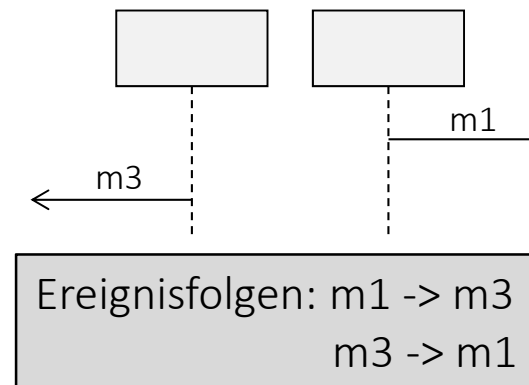
# Lebenslinie: Reihenfolge von Ereigniseintritten

... auf einer Lebenslinie

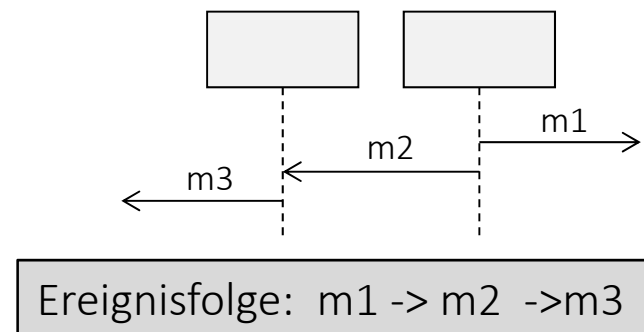


*vgl. Traces im Interleaving-Modell,  
Kapitel 5*

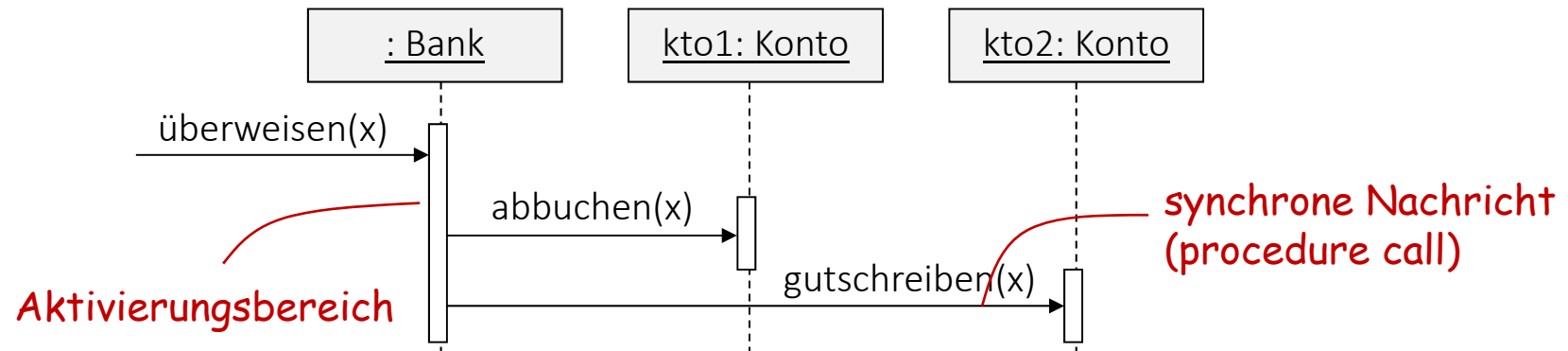
... auf verschiedenen Lebenslinien



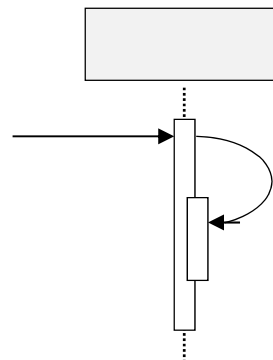
... auf verschiedenen Lebenslinien,  
verbunden durch Nachrichtenaustausch



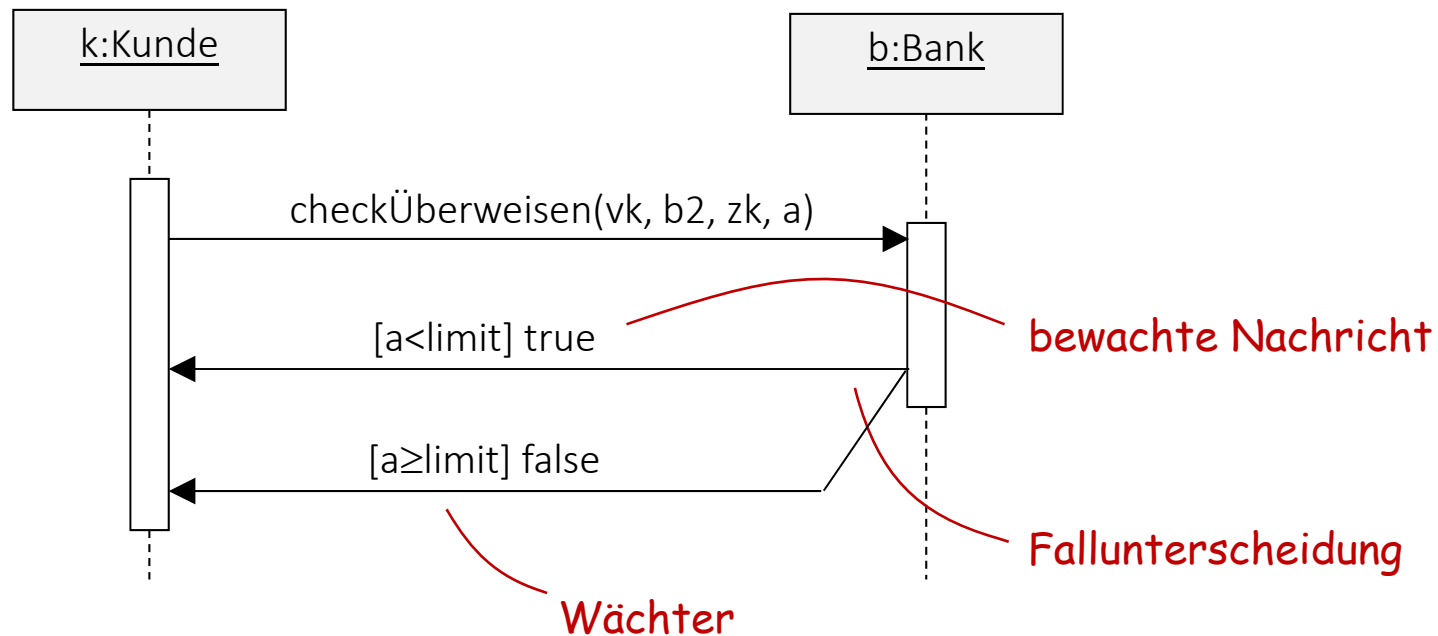
# Aktivierungsbereiche



- Ein Aktivierungsbereich zeigt an, dass das Objekt **aktiv** ist, d.h. eine Operation ausführt
- Vernestete Aktivierungsbereiche möglich



# Fallunterscheidung - durch bewachte Nachrichten

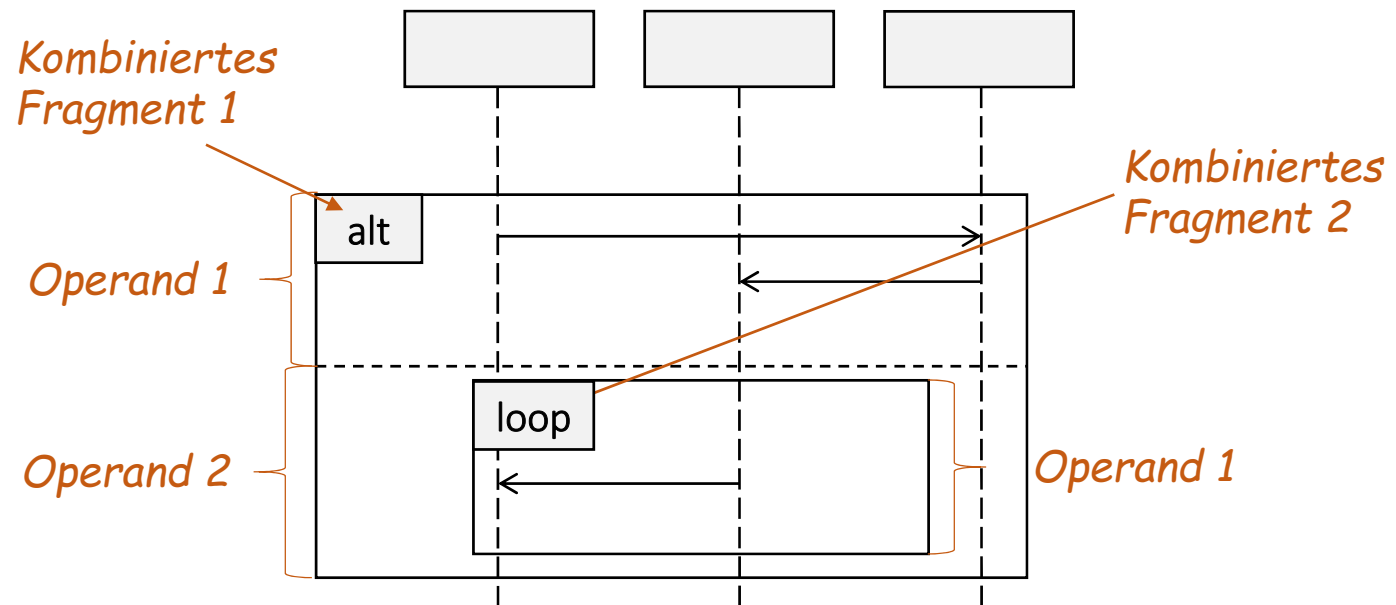


# Interaktionsoperatoren

- Kombination von Interaktionen P, Q durch Operatoren
  - $\text{strict}(P, Q)$
  - $\text{seq}(P, Q)$
  - $\text{loop}(P, \text{min}, \text{max})$
  - $\text{par}(P, Q)$
  - $\text{alt}(P, Q)$
  - $\text{opt}(P)$
  - $\text{brk}(P)$

# Kombinierte Fragmente - Notation

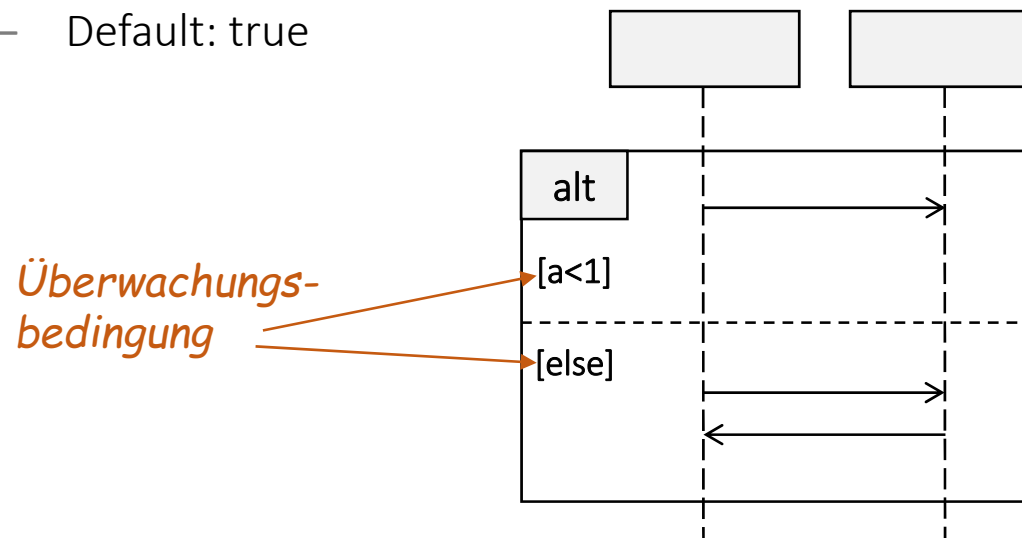
- Kombiniertes Fragment wird wie Sequenzdiagramm mit Rahmen dargestellt
- Art des Fragments wird durch Operator festgelegt
  - default: seq
- Operanden werden durch gestrichelte Linien voneinander getrennt





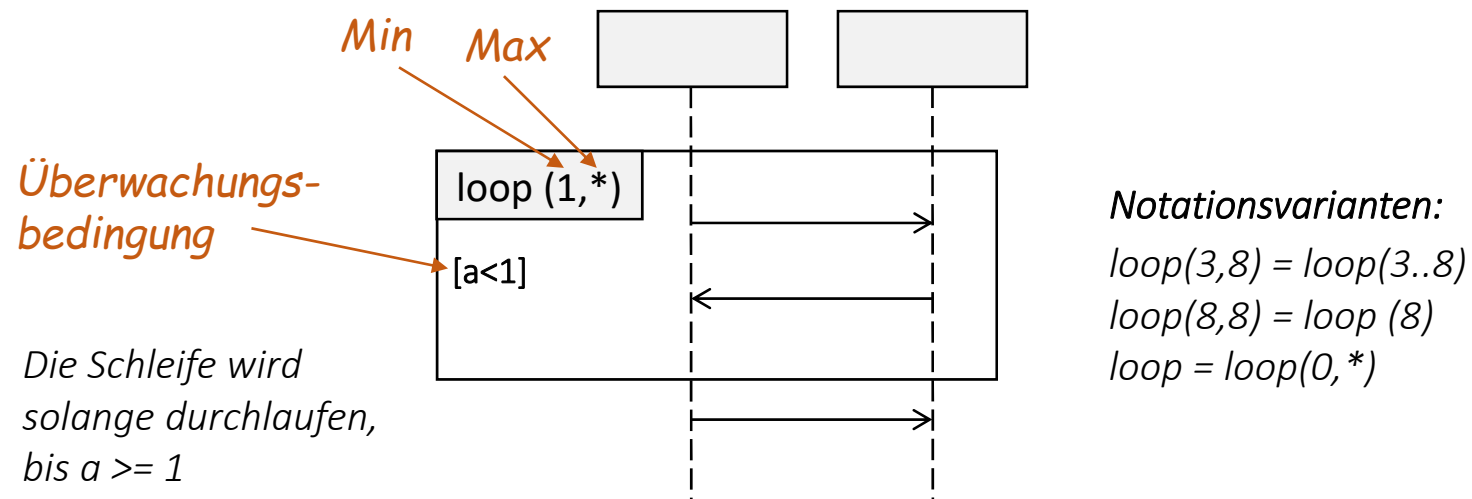
# Verzweigungen und Schleifen: alt-Operator

- Darstellung von zwei oder mehreren **alternativen Interaktionsabläufen** (mind. 2)
- Zur Laufzeit wird maximal ein Operand ausgeführt
- Auswahl eines Operanden anhand von Überwachungsbedingungen
  - Boolescher Ausdruck in eckigen Klammern
  - Vordefinierte else-Bedingung: Operand wird ausgeführt, falls die Bedingungen aller anderen Operanden nicht erfüllt sind
  - Default: true



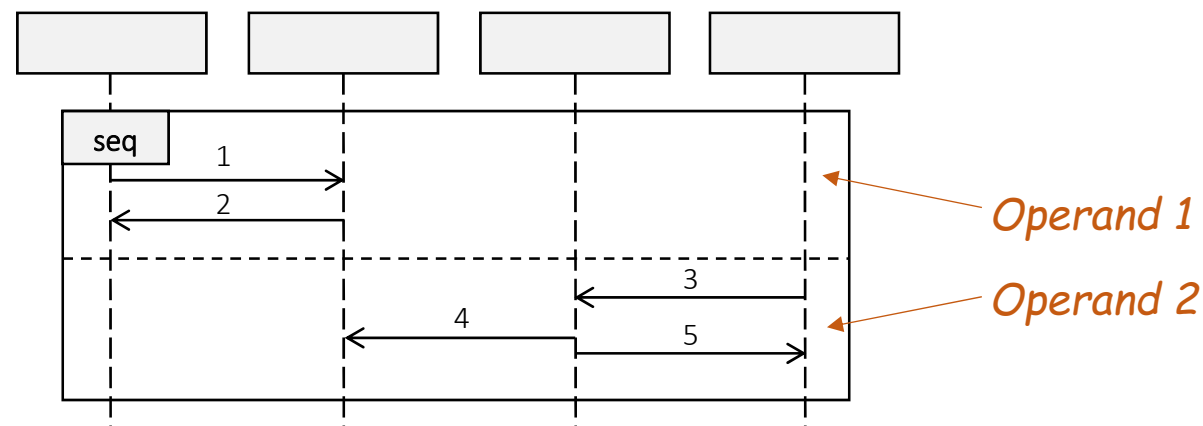
# Verzweigungen und Schleifen: loop-Operator

- Darstellung einer Schleife über einen bestimmten Interaktionsablauf
  - Fragment enthält nur einen Operanden
  - Ausführungshäufigkeit wird durch Zähler mit Unter- und Obergrenze dargestellt
  - Optional: Überwachungsbedingung; wird bei jedem Durchlauf überprüft, sobald die minimale Anzahl an Durchläufen stattgefunden hat

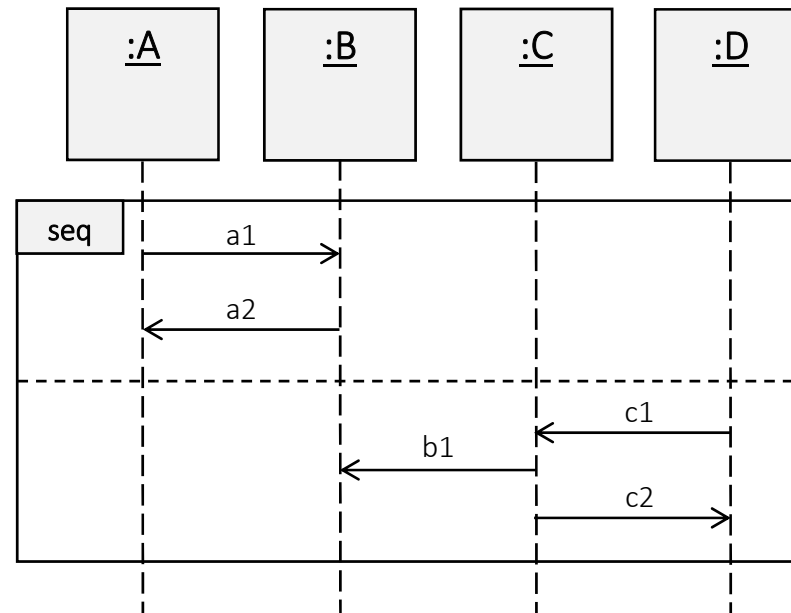


# Nebenläufigkeit u. Ordnung: seq-Operator

- Sequentielle Interaktion mit schwacher Ordnung
- Reihenfolge der Ereigniseintritte:
  - Seq-Operator ist der default-Wert für Interaktionen
  - Beschreibt Folgen von Nachrichten wie auf Folie 36 definiert



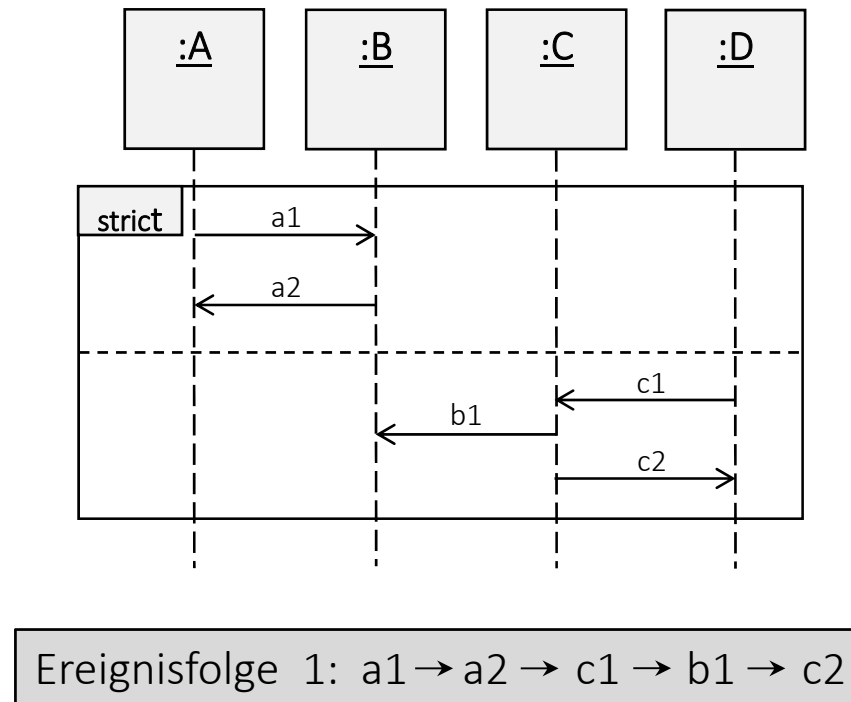
# Nebenläufigkeit u. Ordnung: seq-Operator - Beispiel



Ereignisfolge 1:  $a1 \rightarrow a2 \rightarrow c1 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 2:  $a1 \rightarrow c1 \rightarrow a2 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 3:  $c1 \rightarrow a1 \rightarrow a2 \rightarrow b1 \rightarrow c2$

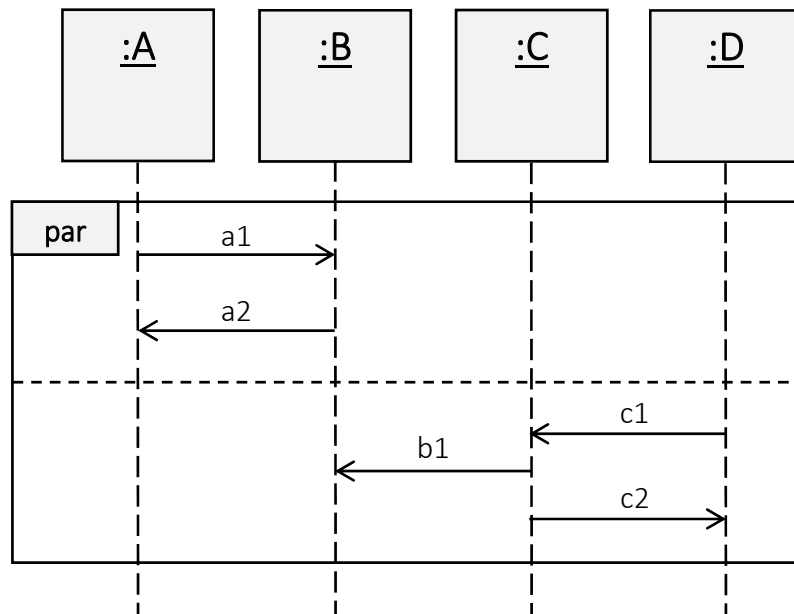
# Nebenläufigkeit und Ordnung: strict-Operator

- Sequentielle Interaktion mit **strenger** Ordnung
- Reihenfolge auf unterschiedlichen Lebenslinien von unterschiedlichen Operanden ist signifikant



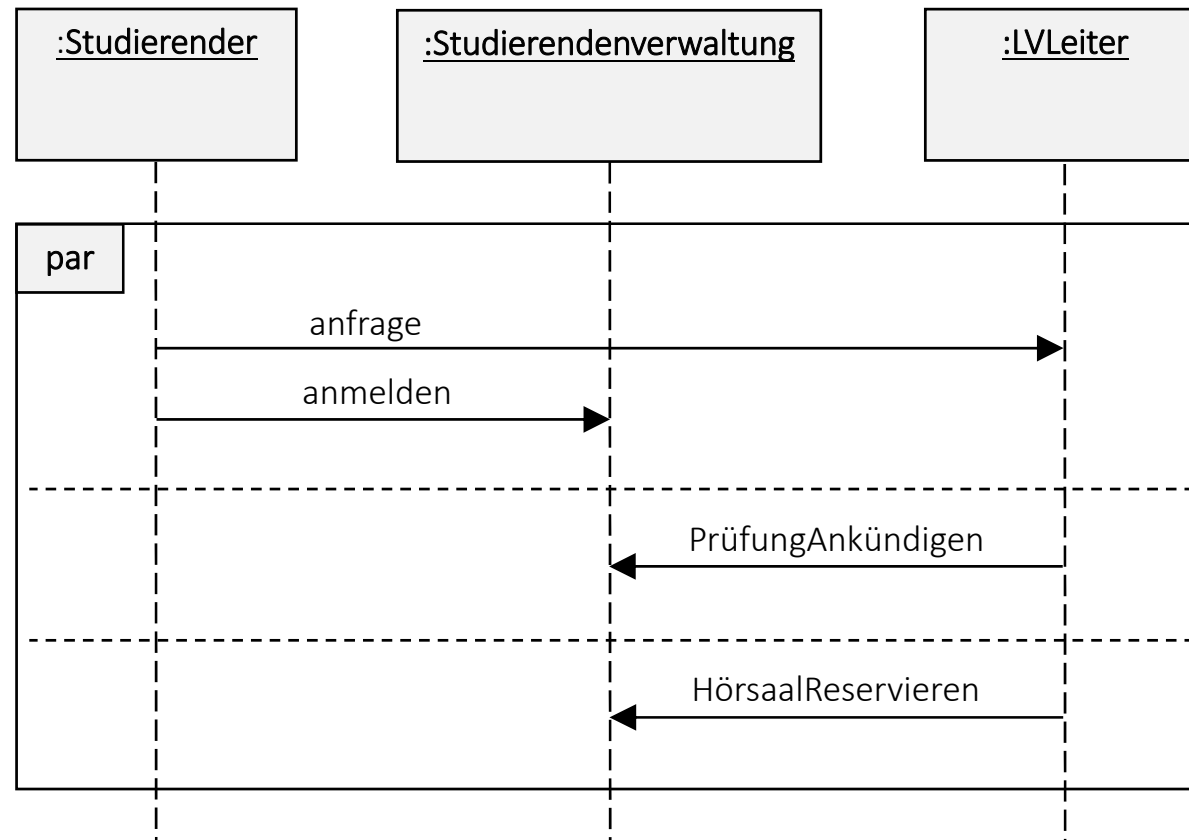
# Nebenläufigkeit und Ordnung: par-Operator

- Nebenläufige Interaktionen
  - Lokale Reihenfolge pro Operand muss erhalten bleiben
  - Reihenfolge der Operanden im Diagramm ist irrelevant!
  - mind. 2 Operanden



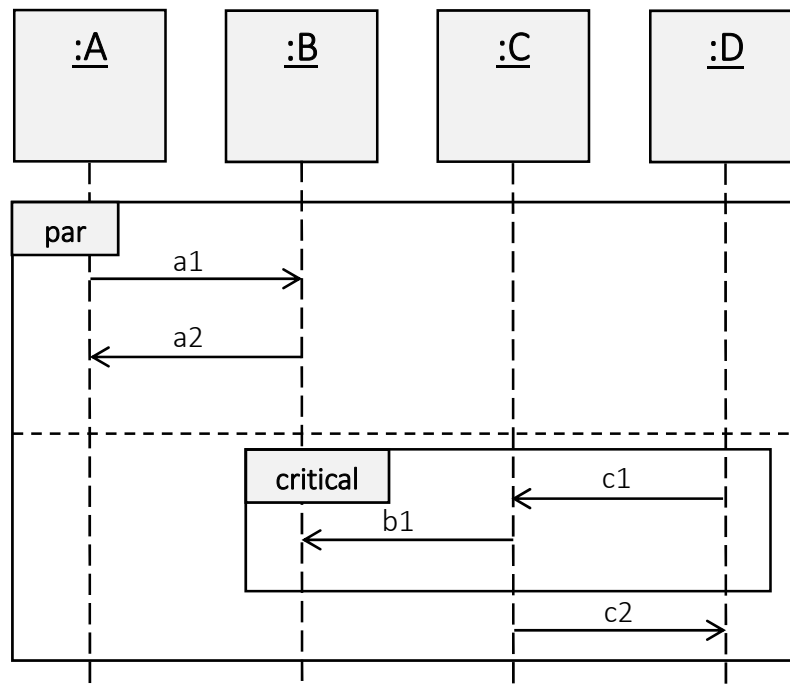
Ereignisfolge 1:  $a1 \rightarrow a2 \rightarrow c1 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 2:  $a1 \rightarrow c1 \rightarrow a2 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 3:  $a1 \rightarrow c1 \rightarrow b1 \rightarrow a2 \rightarrow c2$   
Ereignisfolge 4:  $a1 \rightarrow c1 \rightarrow b1 \rightarrow c2 \rightarrow a2$   
Ereignisfolge 5:  $c1 \rightarrow a1 \rightarrow a2 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 6:  $c1 \rightarrow a1 \rightarrow b1 \rightarrow a2 \rightarrow c2$   
Ereignisfolge 7:  $c1 \rightarrow a1 \rightarrow b1 \rightarrow c2 \rightarrow a2$   
Ereignisfolge 8:  $c1 \rightarrow b1 \rightarrow a1 \rightarrow a2 \rightarrow c2$   
Ereignisfolge 9:  $c1 \rightarrow b1 \rightarrow a1 \rightarrow c2 \rightarrow a2$   
Ereignisfolge 10:  $c1 \rightarrow b1 \rightarrow c2 \rightarrow a1 \rightarrow a2$

# Par-Operator - Beispiel



# Nebenläufigkeit und Ordnung: critical-Operator

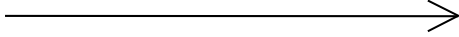
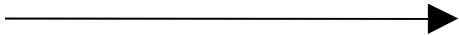
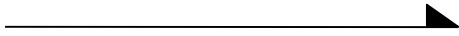
- Kritischer Bereich: atomarer (nicht unterbrechbarer) Interaktionsablauf
- Keine Beschränkung auf Interaktionen außerhalb des kritischen Bereichs



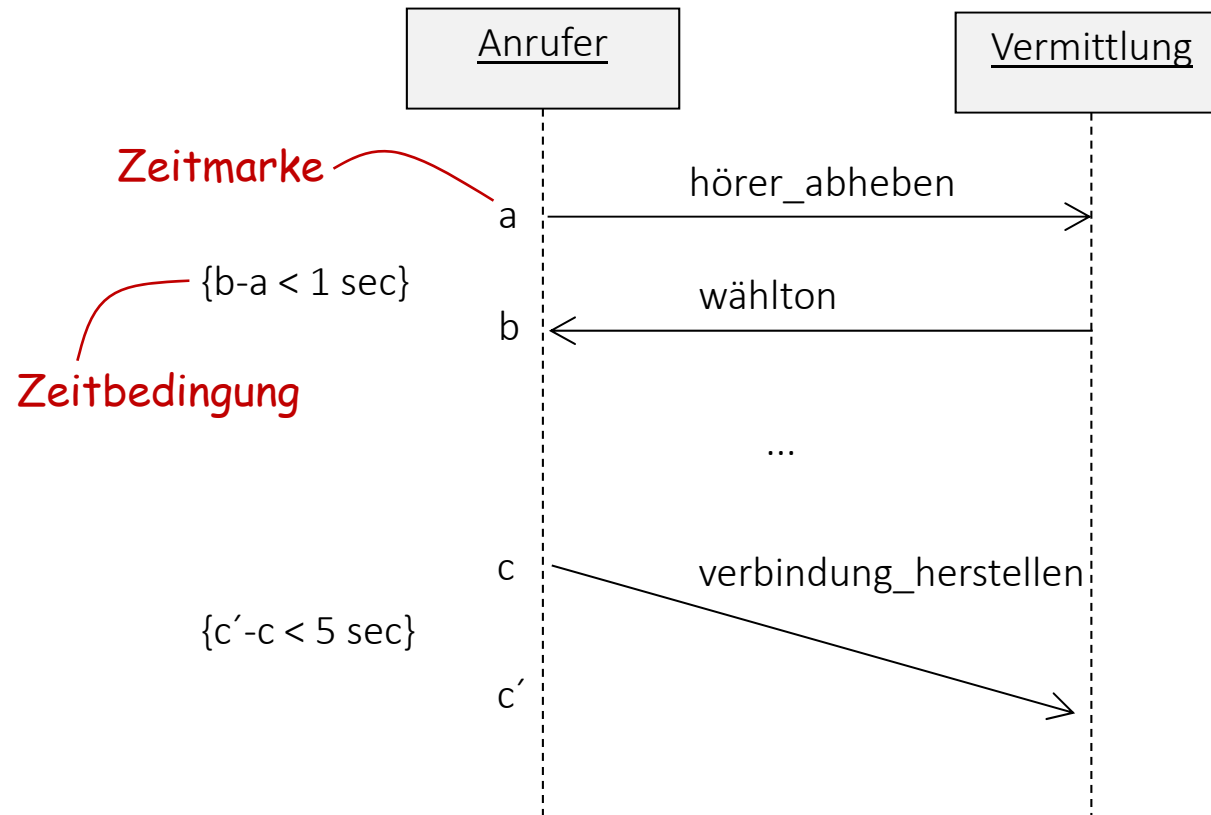
Ereignisfolge 1:  $a1 \rightarrow a2 \rightarrow c1 \rightarrow b1 \rightarrow c2$   
Ereignisfolge 2:  $a1 \rightarrow c1 \rightarrow b1 \rightarrow a2 \rightarrow c2$   
Ereignisfolge 3:  $a1 \rightarrow c1 \rightarrow b1 \rightarrow c2 \rightarrow a2$   
Ereignisfolge 4:  $c1 \rightarrow b1 \rightarrow a1 \rightarrow a2 \rightarrow c2$   
Ereignisfolge 5:  $c1 \rightarrow b1 \rightarrow a1 \rightarrow c2 \rightarrow a2$   
Ereignisfolge 6:  $c1 \rightarrow b1 \rightarrow c2 \rightarrow a1 \rightarrow a2$



# Nachrichtentypen

- Neutraler Pfeil 
  - lässt den unterliegenden Kommunikationsmechanismus offen
  - Entscheidung, ob synchrone oder asynchrone Nachricht, wird auf spätere Entwicklungsschritte verschoben
- Synchrone Nachricht 
  - Interaktion ist einziges, gemeinsames Ereignis zwischen Sender und Empfänger
  - modelliert typischerweise Operationsaufrufe
- Asynchrone Nachricht 
  - Senden und Empfangen der Nachricht sind zwei verschiedene Ereignisse
  - Zwischen Senden und Empfangen kann Zeit verstreichen

# Zeitbedingungen



# Interpretation von Sequenzdiagrammen

- (1) Exemplarische Interpretation
- (2) Vollständige Interpretation

# Exemplarische Interpretation von SDs (1)

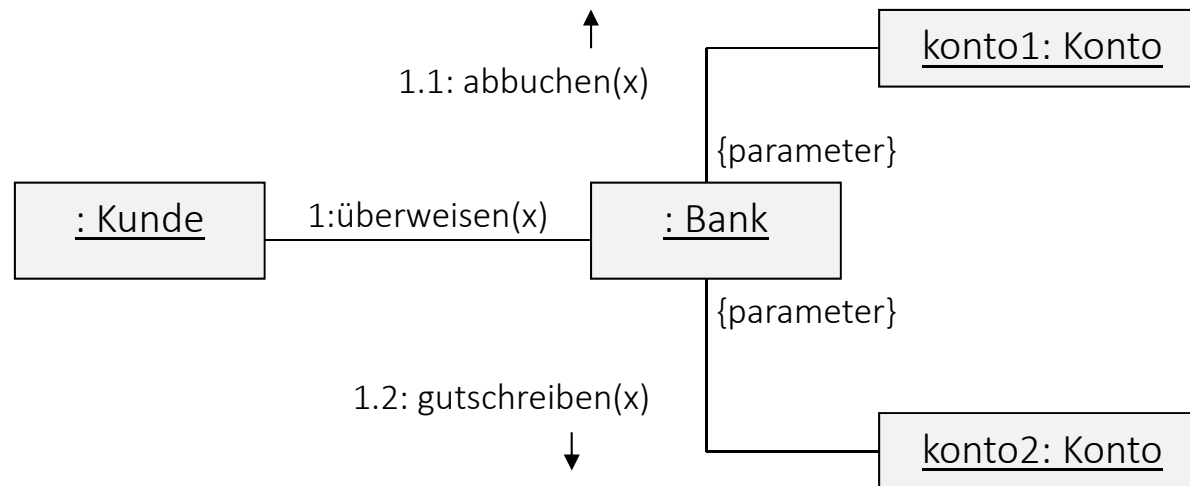
- Das Sequenzdiagramm zeigt den Ausschnitt eines Systemablaufs
- Beschreibung von „Szenarien“
- wenig oder keine Verwendung der erweiterten Sprachelemente
- Anwendung:
  - Dokumentation typischer Nachrichtenflüsse
  - Exploration von Objekten und Nachrichten

# Vollständige Interpretation von SDs

- Das Sequenzdiagramm beschreibt alle möglichen Interaktionen der beteiligten Objekte
- Darstellung von Algorithmen
- „Programmieren“ mit SDs (evtl. mit Generierung von Code)
- extensive Nutzung der erweiterten Sprachelemente
- Aber: Grenzen der Graphikfähigkeit

Art der Interpretation (exemplarisch/vollständig) muss eigentlich angegeben werden.

# Kollaborationsdiagramme

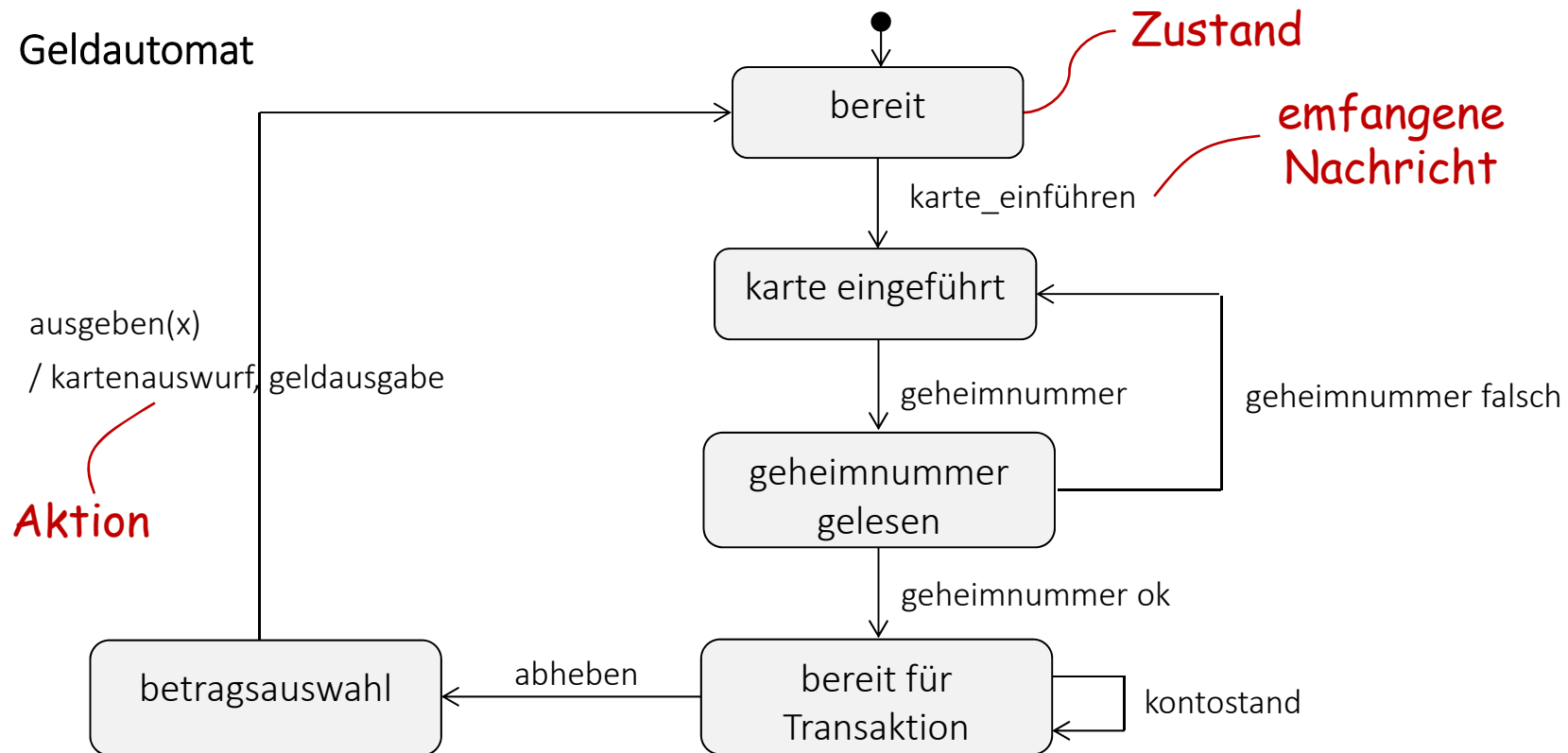


- Wie SDs beschreiben Kollaborationsdiagramme Interaktionen zwischen Objekten
- Reihenfolge der Nachrichten durch Nummerierung spezifiziert

# Kollaborationsdiagramme (2)

- Basis: Objektdiagramme
- Ähnliche Ausdrucksmächtigkeit wie SDs
  - exemplarische/vollständige Interpretation
- alternative Beschreibungstechnik zu SDs
  - Fokus bei Sequenzdiagrammen: zeitlicher Ablauf der Interaktionen
  - Fokus bei Kollaborationsdiagrammen:  
Darstellung der Wege, über die die Nachrichten fließen

## 6.4 Zustandsdiagramme



Ein Zustandsdiagramm ist einer Klasse zugeordnet und beschreibt eine Automatenansicht von Objekten dieser Klasse.

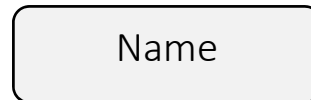


# Geschichte

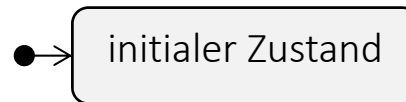
- Herkunft: endliche Automaten (Sprachentheorie)
- Spezifikation technischer Systeme
  - Prominenter Vertreter: Statecharts (David Harel)
- in unterschiedlichen Varianten in den meisten objektorientierten Entwurfsmethoden unterstützt
  - UML: integriert Harel's Statecharts

# Basiselemente - Zustände

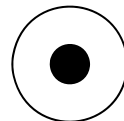
Zustand



initialer Zustand (obligat)



Endzustand (optional)



# Automaten- und Objektzustände

- Objektzustände
  - werden durch die Belegungen der Attribute bestimmt  
⇒ unendlich viele Objektzustände



- Automatenzustände
  - beschreiben eine abstrakte Sicht der Objektzustände
  - fassen alle Objektzustände zusammen, die bezüglich ein- und ausgehender Nachrichten gleiches Verhalten zeigen



# Aktivitäten innerhalb eines Zustands

- **entry / aktivität**
  - Wird beim Eingang in den Zustand ausgeführt
- **exit / aktivität**
  - Wird beim Verlassen des Zustands ausgeführt
- **do / aktivität**
  - Wird ausgeführt, während Objekt im Zustand ist
- **event / aktivität**

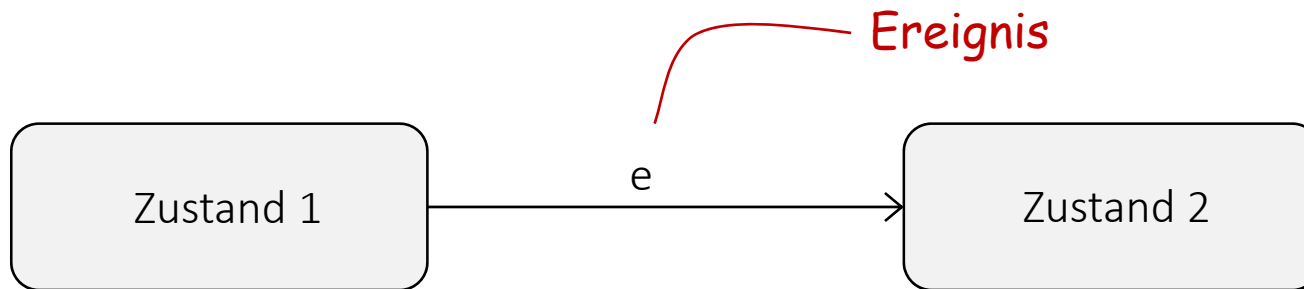


## Aktivität behandelt Ereignis innerhalb des Zustands

- Wird ausgeführt, wenn sich das System in dem Zustand befindet und das Ereignis eintritt

# Zustandsübergänge

Grundform



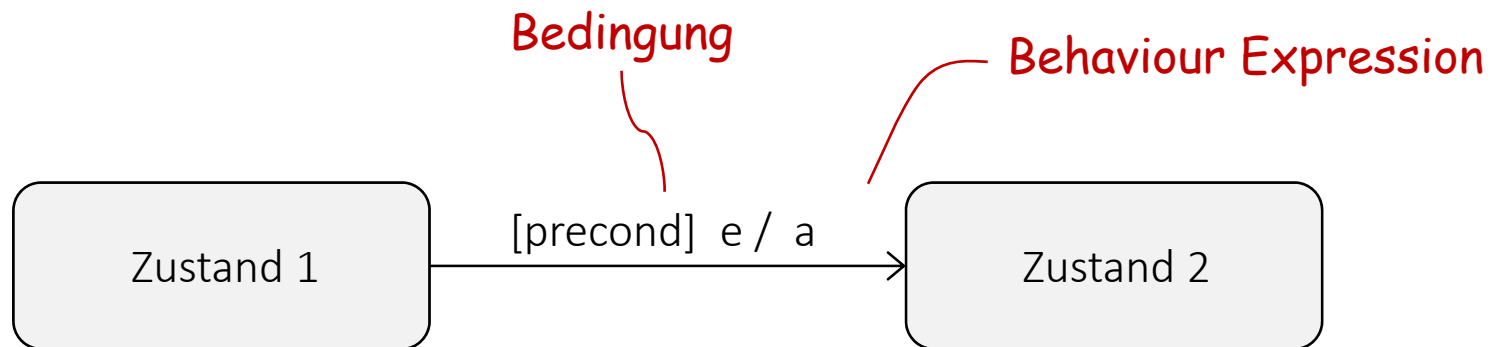
Das Ereignis *e* ist Auslöser für den Wechsel von *Zustand 1* nach *Zustand 2*.

Evtl. andauernde Aktivität in Zustand 1 wird unterbrochen.

# Ereignistypen (1/2)

- **CallEvent**  
Empfang einer Nachricht (Operationsaufruf)
  - Bsp.: stornieren(), kollidiertMit(Termin)
- **SignalEvent**  
Empfang eines Signals
  - Bsp.: right-mouse-button-down, ok-Taste-gedrueckt
- **ChangeEvent**  
Eine Bedingung wird wahr
  - Bsp.: when( $x < y$ ), when( $a = 1$ ), when(terminBestaetigt)
- **TimeEvent**  
Zeitablauf oder Zeitpunkt
  - Bsp.: after(5 sec.), when(date=31.01.20xx)

# Wächter und Aktionen

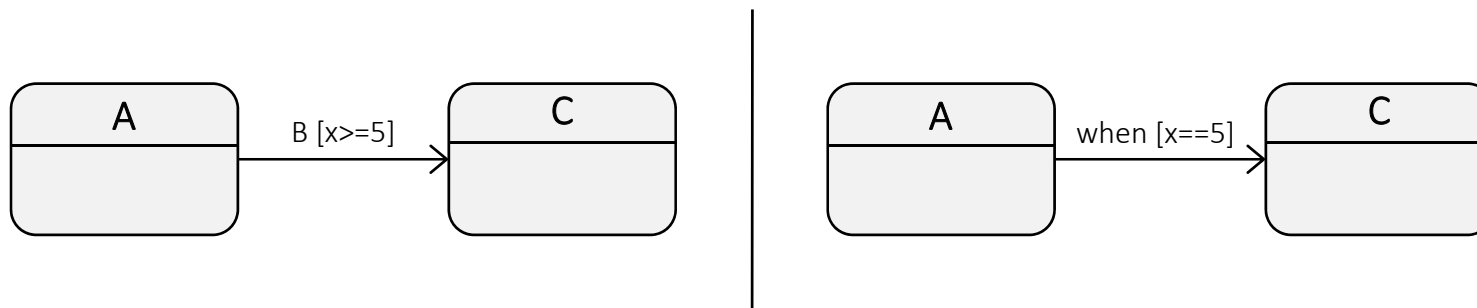


Wenn das Objekt in *Zustand1* ist, das Ereignis *e* auftritt und Bedingung *precond* erfüllt ist, dann führt es die Behaviour Expression (z.B. eine Aktion) *a* aus und geht in *Zustand2* über.

- Nichtdeterminismus bei überlappenden Bedingungen
- Tritt das Ereignis *e* auf und *precond* ist nicht erfüllt, geht *e* „verloren“

# Zustandsübergang: Ereignistypen (2/2)

- Unterschied ChangeEvent und Bedingung
- **ChangeEvent:**
  - Bedingung wird permanent geprüft
  - wenn Bedingung wahr ist, kann zugehöriger Zustandsübergang ausgelöst werden (falls nicht durch zugehörige Überwachungsbedingung blockiert)
- **Bedingung:**
  - wird nur geprüft, wenn zugeordnetes Ereignis eintritt
  - kann selbst keinen Zustandsübergang auslösen



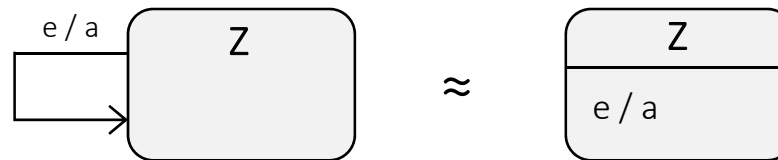


# Behaviour Expression

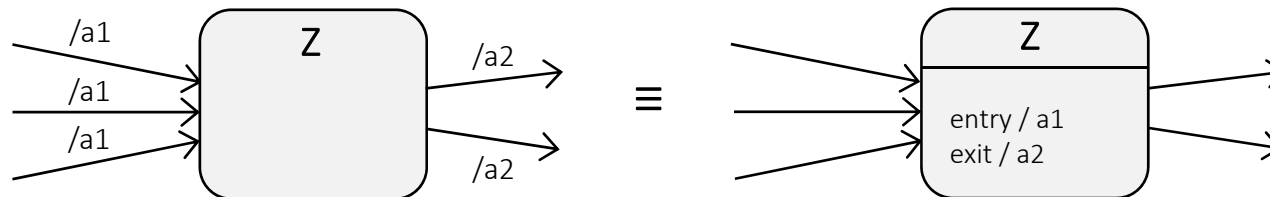
- Beispiele:
  - Lokale Änderung eines Attributwertes
  - Versenden einer Nachricht an ein anderes Objekt
  - Erzeugen oder Löschen eines Objekts
  - Rückgabe eines Ergebniswertes für eine früher empfangene Nachricht
- Die Aktionen werden in der Analysephase meist textuell beschrieben
- Die UML sieht dafür keine Syntax vor

# Zustandsübergang: Innere Transitionen

- Werden wie »äußere« Transitionen von Ereignissen ausgelöst, **verlassen** aber den **aktuellen Zustand nicht**
- Äquivalent zu Selbsttransition, sofern keine entry / exit-Aktivitäten vorhanden

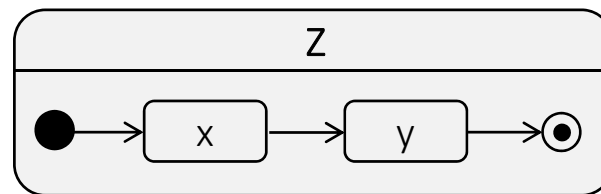


- Gleiche Aktionen können in den Zustand hineingezogen werden:



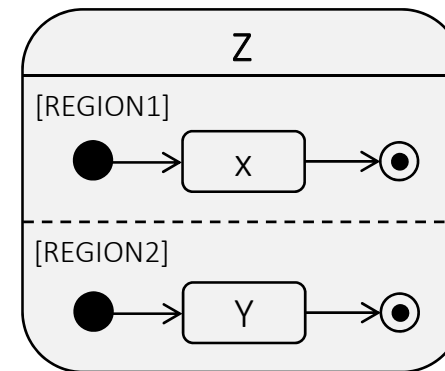
# Komplexe Zustände (1/2)

- = Zustände, die aus mehreren Subzuständen zusammengesetzt sind  
⇒ geschachteltes Zustandsdiagramm
- Die Subzustände sind disjunkt, d.h. genau ein Subzustand ist aktiv, wenn der komplexe Zustand aktiv ist



*Zu einem Zeitpunkt kann  
nur X ODER Y aktiv sein!*

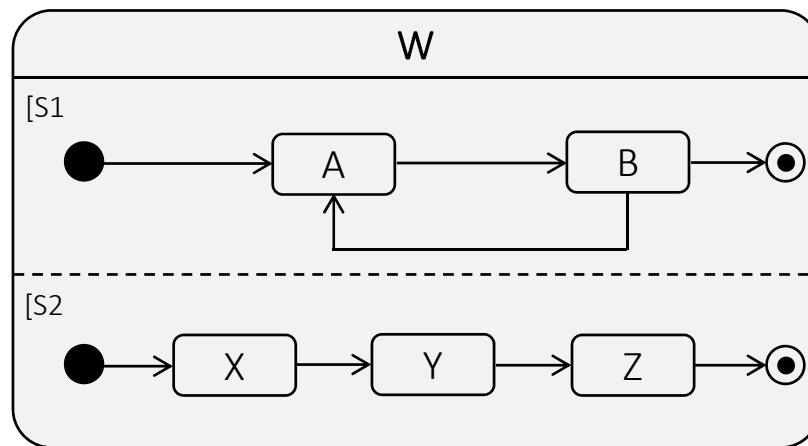
- Teilung des Superzustandes in mehrere Regionen
  - ⇒ die Subzustände sind nebenläufig, gleichzeitig aktiv
  - Z = „orthogonaler Zustand“



*Zu einem Zeitpunkt  
sind X UND Y aktiv!*

# Komplexe Zustände (2/2)

- Beispiel



*Zu einem Zeitpunkt kann  
nur A ODER B aktiv sein!*

*Zu einem Zeitpunkt kann  
nur X ODER Y aktiv sein!*

*Zu einem Zeitpunkt ist jeweils ein Subzustand  
jeder der beiden orthogonalen (=parallelen)  
Regionen von W aktiv!*

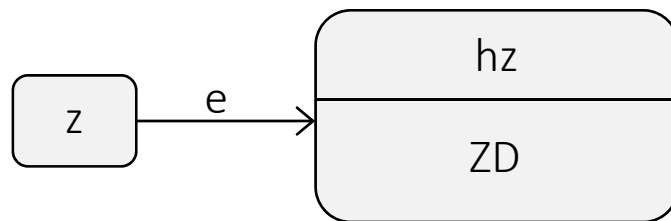
– Mögliche Kombinationen von gleichzeitig aktiven Zuständen:

A & X oder A & Y oder A & Z oder A & Endzustand von [S2]

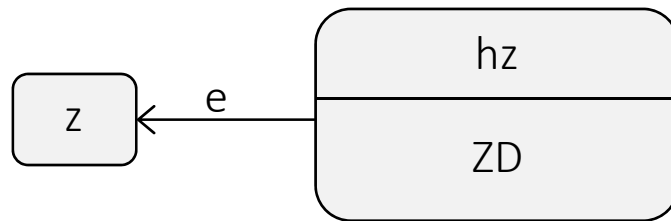
B & X oder B & Y oder B & Z oder B & Endzustand von [S2]

Endzustand von [S1] & X oder Endzustand von [S1] & Y oder Endzustand von [S1]  
& Z oder Endzustand von [S1] & Endzustand von [S2]

# Transitionen von und zu komplexen Zuständen

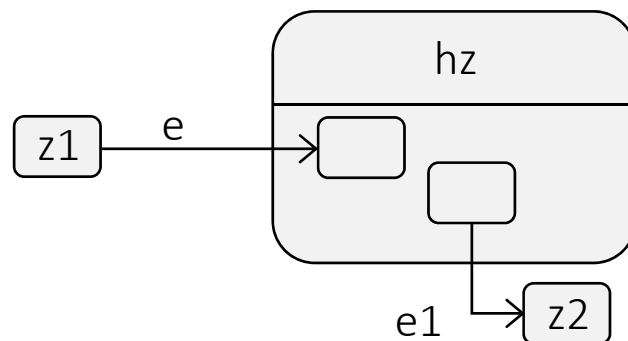


$\overset{\wedge}{=}$  Transition von z zu allen  
**initialen** Zuständen von ZD (\*)



$\overset{\wedge}{=}$  Transition von allen  
Zuständen von ZD  
nach z (\*)

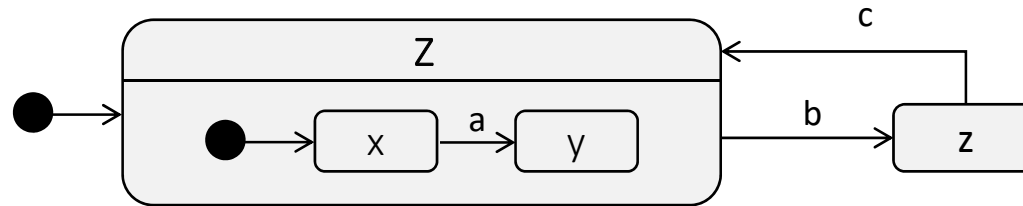
(\*) entry/exit-Aktivitäten des hierarchischen Zustands werden beim Eintreten in / Austreten aus dem hierarchischen Zustand ausgeführt.



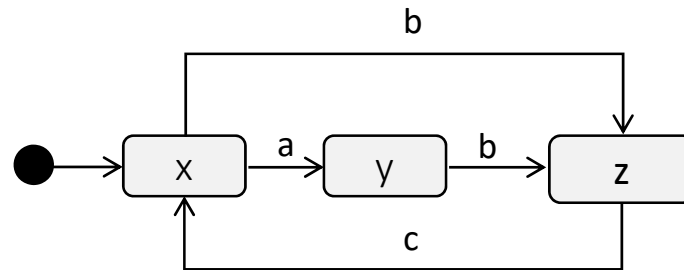
Überschreiten von  
Hierarchien möglich

Sobald e1 auftritt,  
wird der Oberzustand hz verlassen

# ODER-Zustände

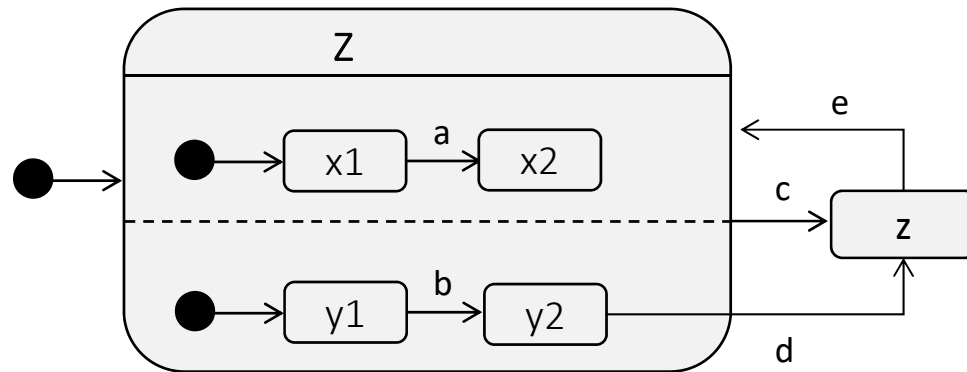


## Äquivalente, flache Darstellung

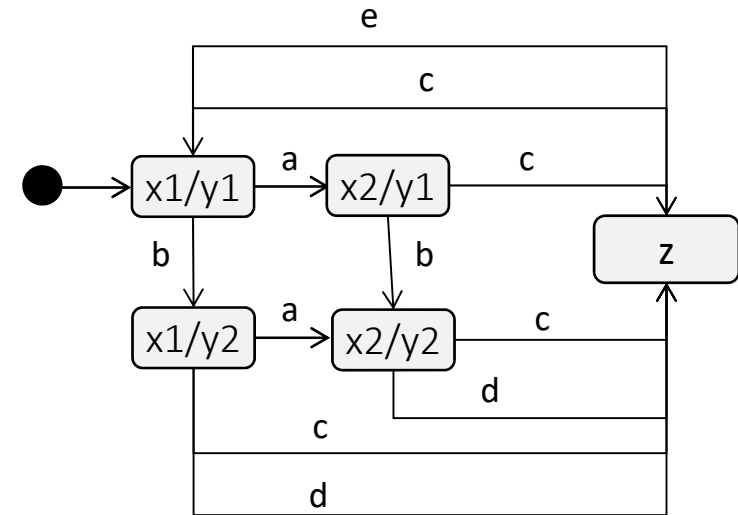


- Mit ODER-Zuständen kann eine kompaktere Darstellung erreicht werden, die Anzahl der Transitionen wird verringert
- Beispiel: Zustand **z** entspricht einem Fehlerzustand, der von jedem anderen Zustand erreicht werden kann

# UND-Zustände

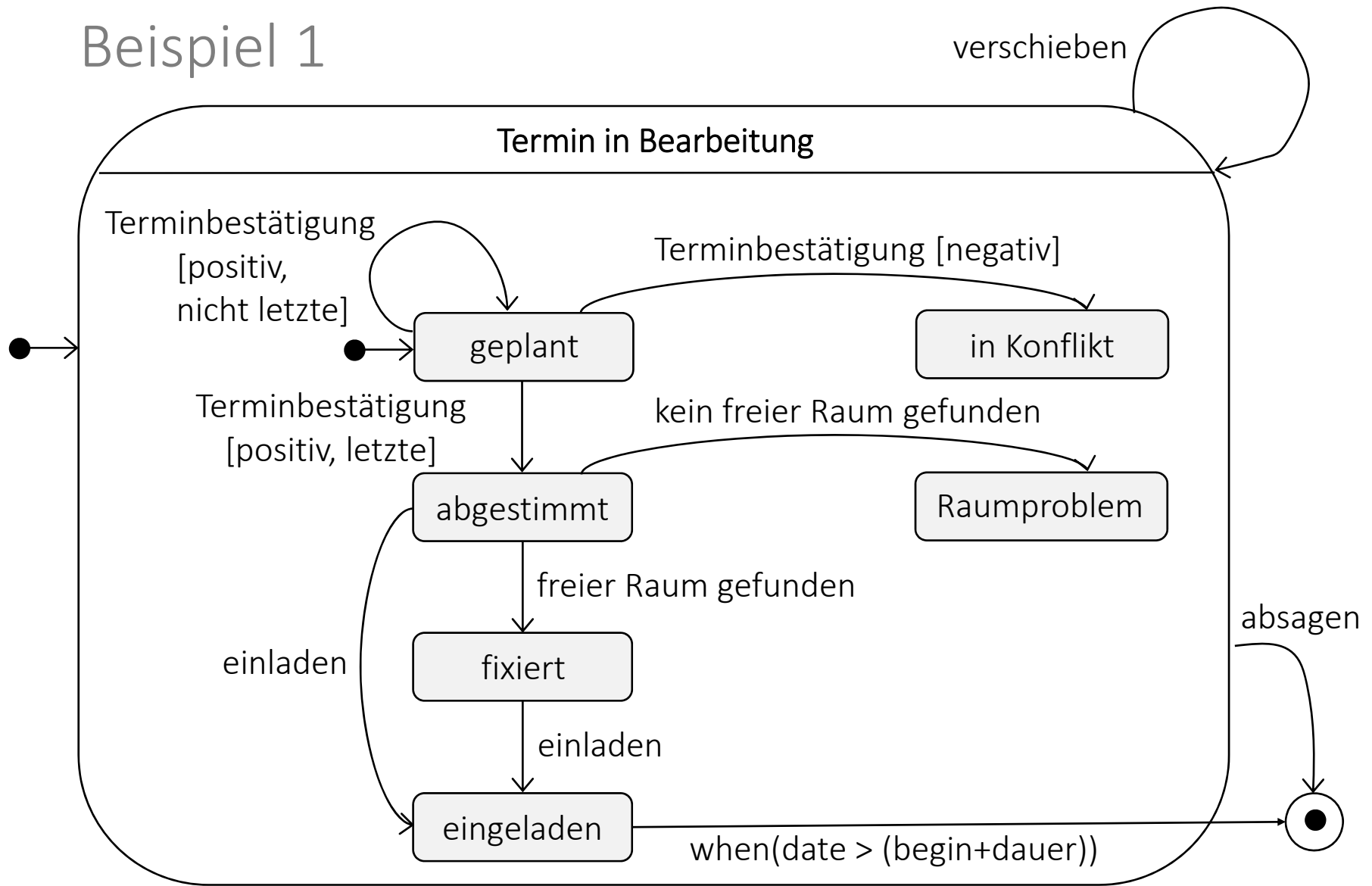


## Äquivalente, flache Darstellung



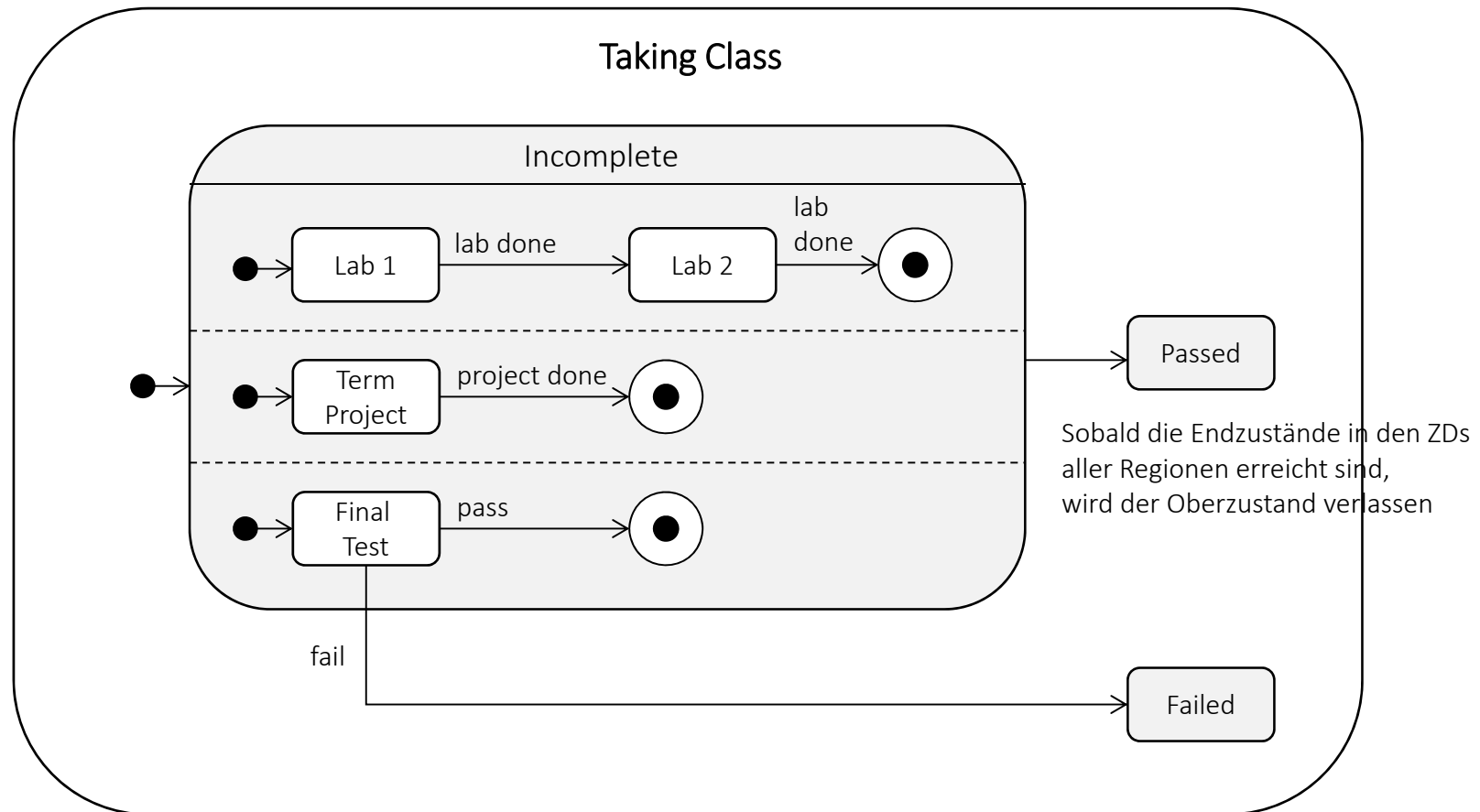
- Mit UND-Zuständen kann sowohl die Zahl der Zustände als auch die Zahl der Transitionen reduziert werden, dadurch wird eine kompaktere Darstellung erreicht

# Beispiel 1





## Beispiel 2



Die Zustandsübergänge in den Regionen unabhängig voneinander

# Interpretationen nebenläufiger Zustände

- (1) Nebenläufige Zustände werden durch nebenläufige Prozesse interpretiert
- (2) Nebenläufigkeit bei Zuständen dient der kompakten Darstellung des Zustandsraums und vermeidet die Darstellung des Kreuzprodukts

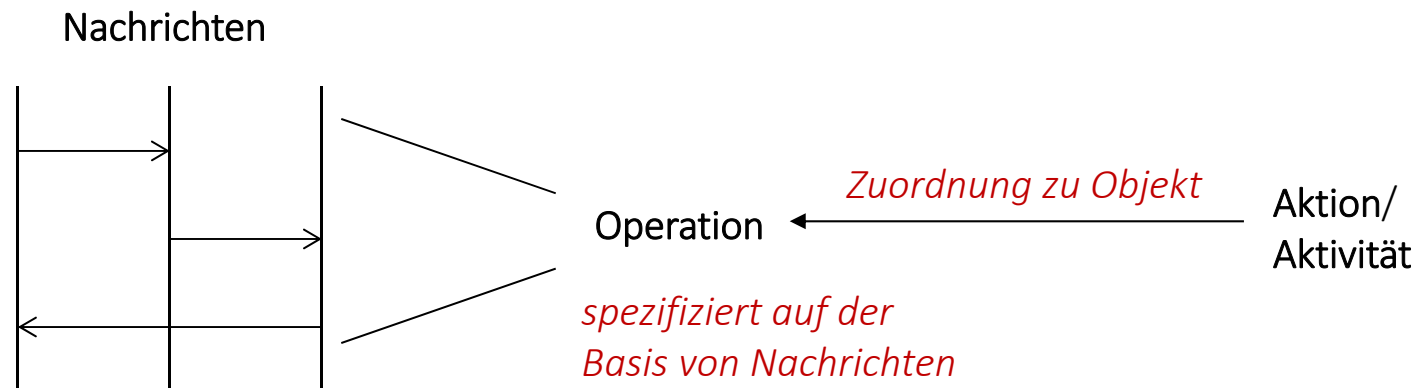
Nebenläufige Teilzustände erfordern nicht unbedingt eine nebenläufige Implementierung.

# Unvollständige Diagramme

- Basissicht: reaktive Objekte  
Ein reaktives Objekt akzeptiert zu jedem Zeitpunkt jede Nachricht und reagiert auf sie
- Was passiert, wenn ein Objekt in einem Zustand eine Nachricht erhält, die keiner ausgehenden Transition im Diagramm entspricht?
  1. Die Nachricht wird ignoriert, d.h. der Zustand bleibt unverändert (impliziter „Schleifen“-Übergang)
  2. Standardisierte Fehlerbehandlung durch einen Fehlerzustand
  3. Unterspezifikation in der Modellierungssicht („Verhalten wird später festgelegt“)

# Übersicht - Konzepte der Dynamik

- Nachrichten/Ereignisse (atomar)
- Operationen (komplex, einem Objekt zugeordnet)
- Aktionen/Aktivitäten (informell, nicht unbedingt einem Objekt zugeordnet)



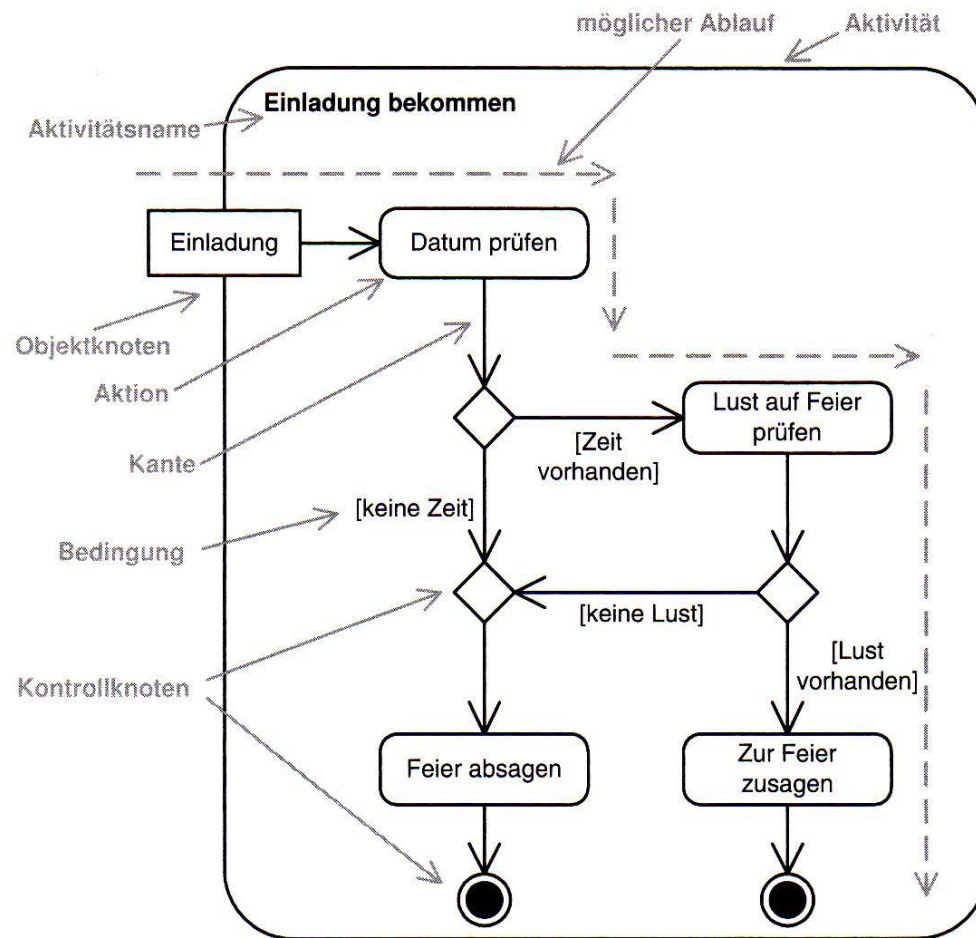
# Übersicht - Beschreibungstechniken der Dynamik

	<i>Basis</i>	<i>beschriebene Abläufe</i>	<i>Änderung von Objektzuständen</i>
Sequenzdiagramme Kollaborationsdiagramme	Nachrichten	meist: beispielhaft Abläufe zwischen Objekten	-
Zustandsdiagramme	Nachrichten, Aktionen	alle möglichen Abläufe eines Objekts	grob (Wechsel von Automatenzuständen)
Aktivitätsdiagramme	Aktionen	Mengen von Abläufen, Nicht notwendigerweise Objekten zugeordnet	-
Vor-/Nachbedingungen	Operationen	-	Änderung der Attributwerte

## 6.5 Aktivitätsdiagramme

- Fokus des Aktivitätsdiagramms: **prozedurale Verarbeitungsaspekte**
- Spezifikation von **Kontroll**-und/oder **Datenfluss** zwischen Arbeitsschritten (Aktionen) zur Realisierung einer Aktivität
- **Aktivitätsdiagramm in UML:**
  - ablauforientierte Sprachkonzepte
  - basierend u.a. auf Petri-Netzen
- Sprachkonzepte und Notationsvarianten decken ein **breites Anwendungsgebiet** ab
  - Modellierung objektorientierter und nichtobjektorientierter Systeme wird gleichermaßen unterstützt

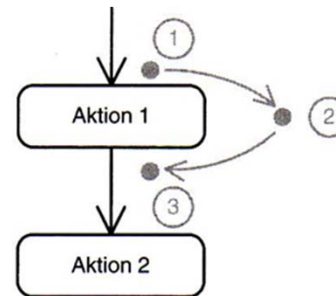
# Beispiel



Quelle im folgenden: UML@classroom

# Das Tokenkonzept

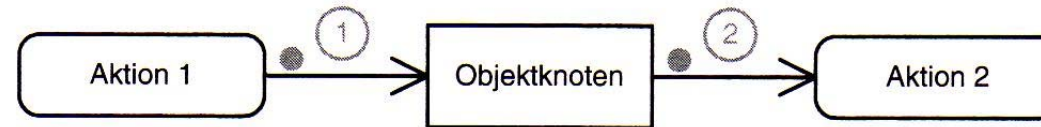
- Aktivitätsdiagramme modellieren potentiell parallele Abläufe auf der Basis eines Tokenkonzepts
  - Vgl. Petrinetze



- Kontroll-Token
  - Eine Aktion startet, wenn ein Token auf der eingehenden Kante angeboten wird
  - Während der Ausführung der Aktion wird das Token in der Aktion „aufbewahrt“
  - Ist der Ablauf abgeschlossen, wird das Token an der ausgehenden Kante weitergereicht



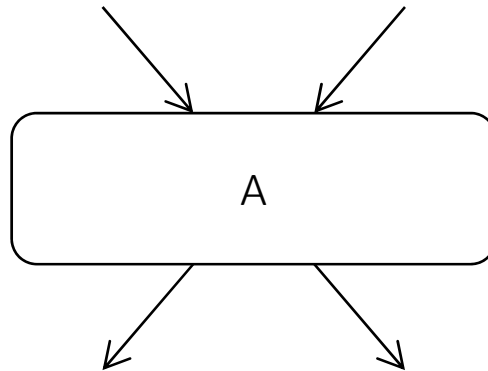
# Objektflüsse



- Daten-Token
  - In einen Objektknoten hineingehende Token repräsentieren Objekte
  - Aus einem Objektknoten herausgehende Token transportieren das Objekt in die Folgeaktion
- Objektknoten können mit einem Zustand verknüpft sein

Bestellung  
[ausgefüllt]

# Aktionen



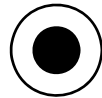
- Alle eingehenden Kanten müssen ein Token besitzen
- Nach Ausführung von A erhalten alle ausgehenden Kanten ein Token

# Kontrollelemente

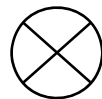


- Startknoten
  - Ein Diagramm kann beliebig viele Startknoten enthalten
  - Ein Startknoten kann beliebig viele ausgehende Kanten haben
  - Beim Start bietet der Startknoten allen ausgehenden Kanten je ein Token an
  - Alle Startknoten starten gleichzeitig

# Endknoten

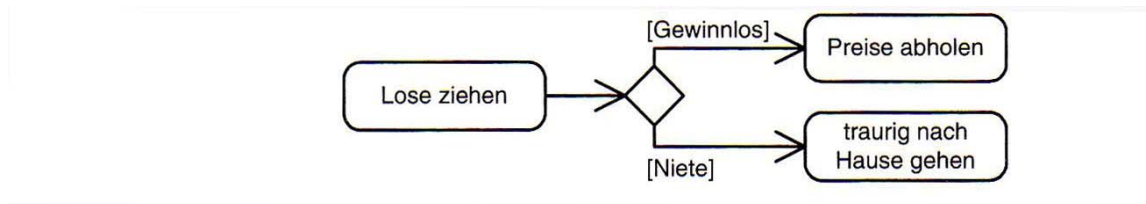


- Endknoten für Aktivitäten
  - Beendet die gesamte Aktivität, parallele Abläufe der gleichen Aktivität werden ebenfalls beendet, Daten-Token in Ausgabe-Aktivitäten werden an den Aufrufer übergeben
  - Falls mehrere Endknoten existieren, wird die Aktivität beendet, sobald einer dieser Knoten einen Token enthält

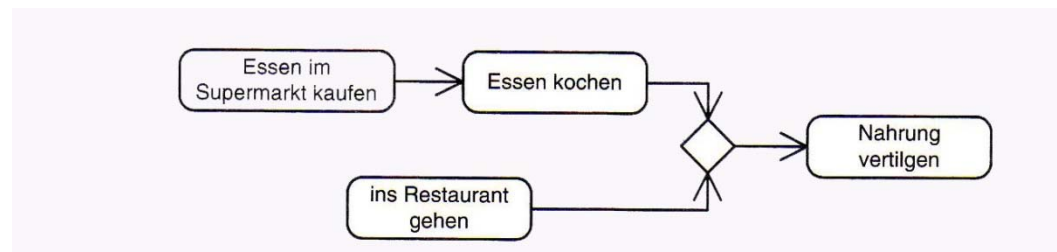


- Endknoten für Kontrollflüsse
  - Beendet einen Ablauf durch Löschen des zugehörigen Tokens

# Verzweigungs- und Verbindungsknoten

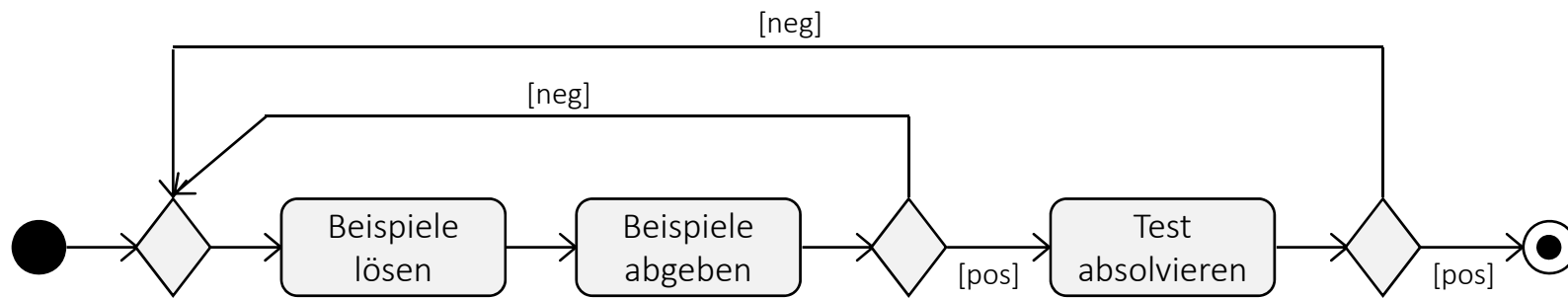


- Verzweigungsknoten
  - Ein Token, das einen Verzweigungsknoten passiert, passiert eine ausgehende Kante

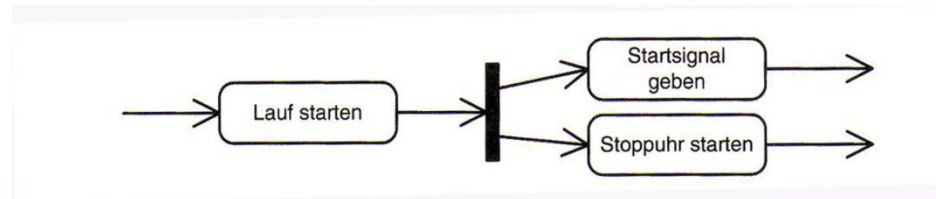


- Verbindungsknoten
  - Liegen mehrere Tokens an, werden sie serialisiert und an der ausgehenden Kante angeboten

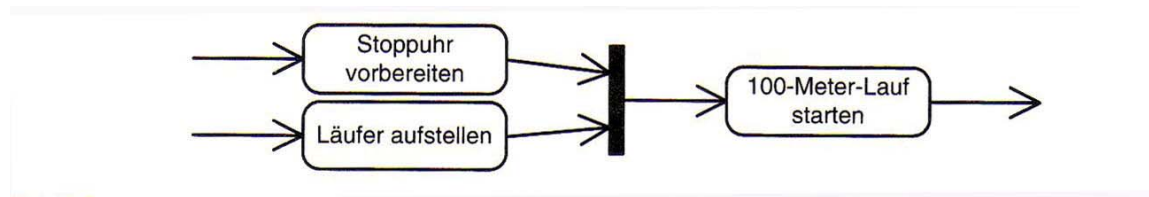
# Alternative Abläufe – Bsp.: Absolvieren einer LV



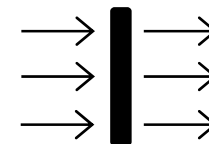
# Synchronisations- und Parallelisierungsknoten



- Parallelisierungsknoten
  - Der eingehende Ablauf wird in mehrere parallele Abläufe aufgeteilt, jedes eingehende Token wird dabei an jeder ausgehenden Kante angeboten

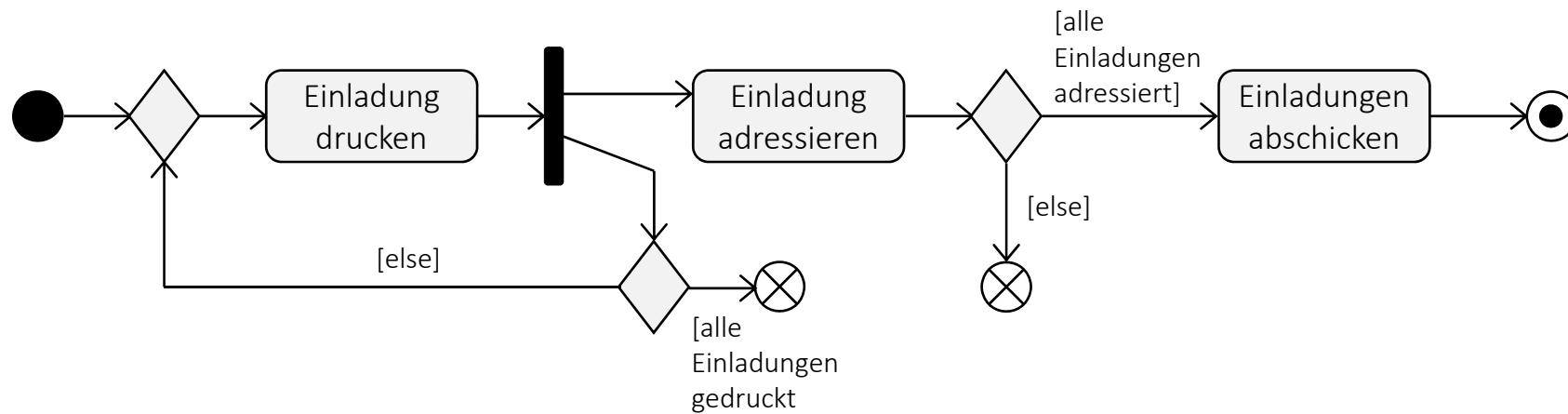


- Synchronisationsknoten
  - Falls an allen eingehenden Kanten Token anliegen, werden sie zu einem Token verschmolzen und an der ausgehenden Kante weitergeleitet
  - Kombiniertes Parallelisierungs-/Synchr. Knoten



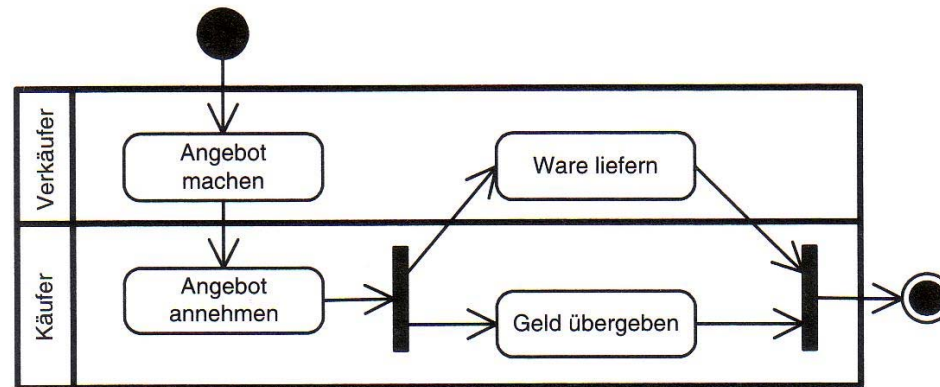
# Beispiel

- Versenden von Einladungen



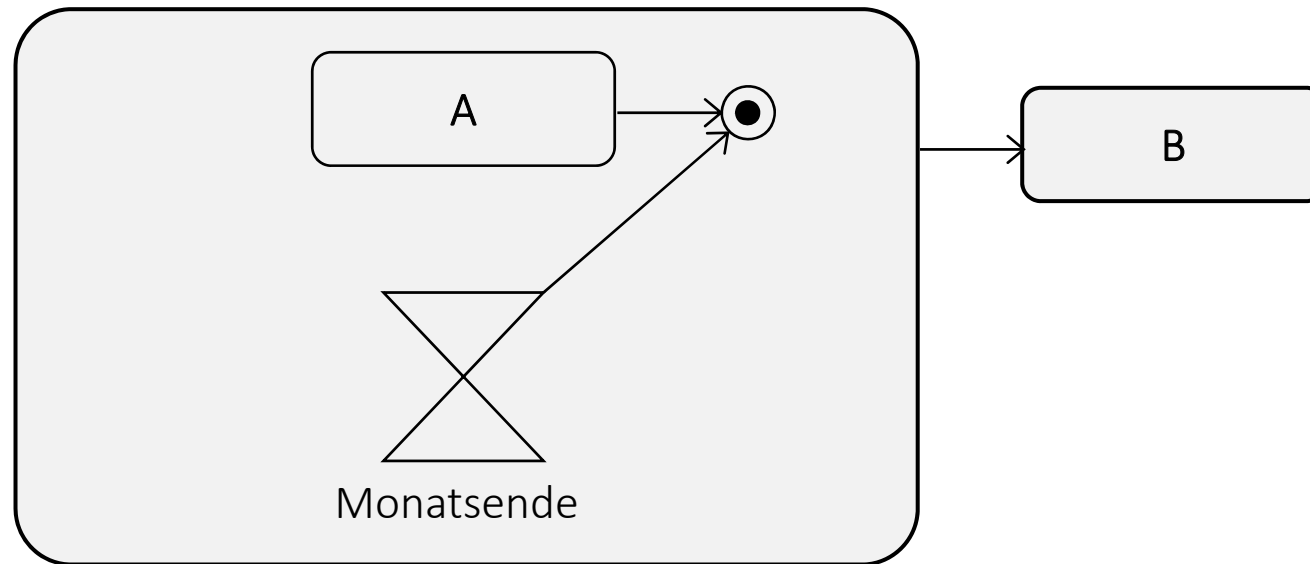


# Aktivitätsbereiche



- Auch mehrdimensionale Aktivitätsbereiche und hierarchische Aktivitätsbereiche möglich

# Zeitereignisse



# Weitere Beschreibungselemente

- Signale und Ereignisse
- Hierarchische Aktivitätsdiagramme
  - Aktionen können Aktivitäten aufrufen
  - Übergabe von Parametern möglich
- Sprungmarken
  - Unterbrechung von Kanten zur übersichtlichen Gestaltung von Diagrammen
- Exception-Handling
- Mengenverarbeitungsbereiche
  - Unterstützung des Bearbeitens von Container-Strukturen
- ...

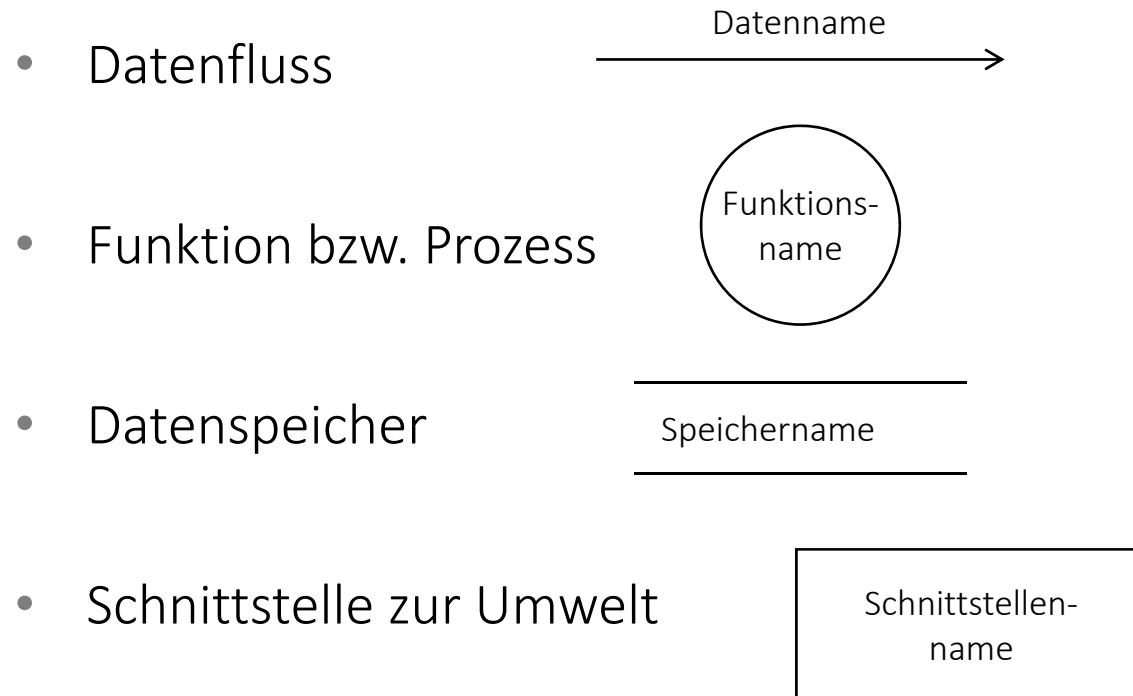
# Verwendung von Aktivitätsdiagrammen

- Aktivitätsdiagramme sind eine Beschreibungstechnik für die frühen Phasen des Entwurfs (bevor Nachrichten und Operationen bekannt sind)
  - Beschreibungstechnik für die Geschäftsprozessmodellierung
- **Achtung:** Aktivitätsdiagramme unterstützen mehr eine funktionsorientierte als eine objektorientierte Sicht!

## 6.6 Datenflussdiagramme

- Datenflussdiagramme sind nicht Teil der UML, werden aber in der Praxis oft verwendet
- Ursprung: DeMarco, entwickelt im Rahmen der Structured Analysis Method
- Beschreibung der Wege von Daten zwischen Prozessen, Speichern und Schnittstellen

# Grundkonzepte

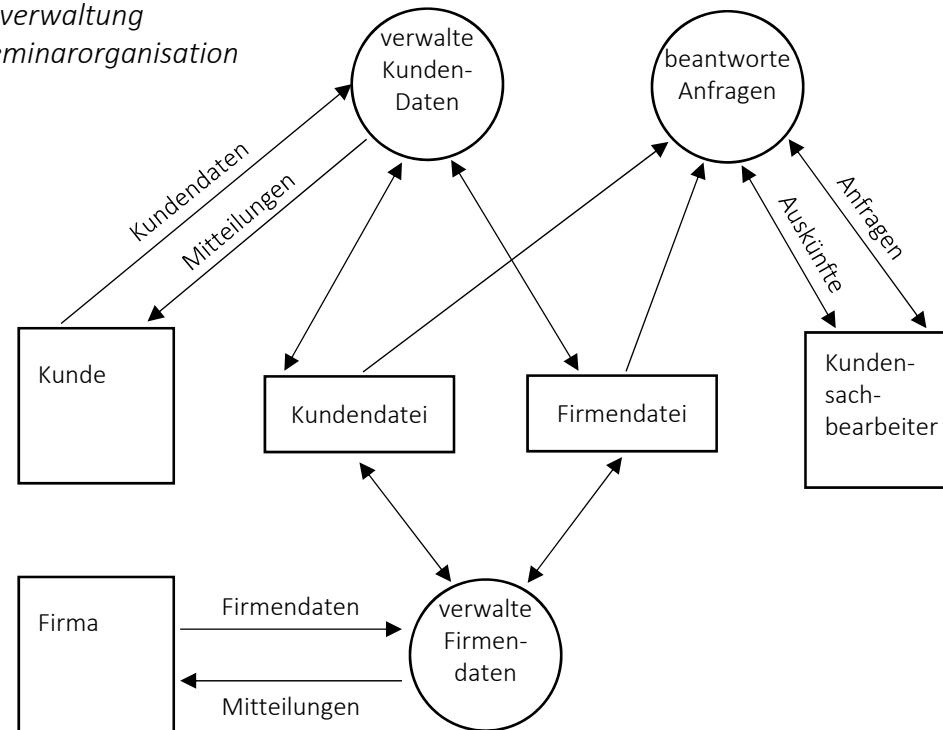


# Systemsicht

- Die Umwelt besteht für das System aus Informationsquellen und –senken.
- Informationen entstehen in Informationsquellen und fließen zu Funktionen; eine Funktion transformiert ankommende Datenflüsse in ausgehende Datenflüsse
- Speicher sind Hilfsmittel zur Ablage von Informationen; in einen Speicher können Informationen hineinfließen (eingehender Pfeil), Informationen können gelesen werden (ausgehender Pfeil), auf einen Speicher kann lesend und schreibend zugegriffen werden (Doppel-Pfeil)

# Beispiel

Abb. 2.7-1:  
DFD der Kundenverwaltung  
Innerhalb der Seminarorganisation



Quelle: Balzert, Lehrbuch der SW-Technik



# Bewertung

- DFDs können leicht erstellt werden, sind gut lesbar
- DFDs sind auch Fachexperten gut vermittelbar
- Unübersichtlich für große Systeme
- Es ist schwierig, ein einheitliches Abstraktionsniveau einzuhalten
- In den letzten Jahren im Kontext der Datenschutzgrundverordnung wieder mehr in Verwendung